29

Is there something major I'm missing regarding the Quadcopter project?

Nathaniel W 8 months ago

I've been stuck on the Train a Quadcopter to fly project for a while now and it's been really frustrating since I feel like the objectives/challenges for this project have not been communicated well. Looking in the MLND Slack, I see that many students are experiencing the same frustrations, but I don't see anyone having any breakthroughs, which seems to indicate that everyone is missing some big, important piece of information. The closest I've heard are people saying to submit it with an agent that doesn't actually fly, and maybe you'll pass.

I want to change that, but I don't have all the answers. Below is a list of all the small breakthroughs I've had while working on this project, in no particular order. My biggest achievement so far is that sometimes my quadcopter doesn't crash into the ground for 5 seconds straight, so take this all for what it's worth. Please share if there's anything useful that you realized that's not covered here, and hopefully we can all figure out this project together.

- The reward function seems to be the critical factor holding back the agent from learning properly; *do this first*.
- The starter code doesn't have any way to recognize if the target position was reached. (*Big Eureka moment right here*)
- The physics_sim doesn't have a floor, if you set `init_pos` too low and if the agent doesn't apply thrust right away, it will just fall below 0 and end the episode. (Thanks to Serchan Soydan in the Slack for this helpful nugget)
- Without specifying clear and large rewards for performing well, the agent might just decide it's better to crash early and end the episode sooner.
- Euler angles are in radians and here's what they mean:
- `x`/`phi` corresponds to yaw
- `y`/`theta` corresponds to roll
- `z`/`psi` corresponds to pitch
- The angles in the pose variable are all positive, so numbers like 6.1 are actually just a small negative variance from 0 (360 degrees = 2 Pi radians)
- If implementing time-based rewards, use the time from the sim, because outside of that the time gets jumbled due to the replay buffer.

- The supplied code for a DDPG implementation is mostly plug and play (to get 'working', but tuning seems like it might be needed), but you'll probably want to add in score and noise_scale variables for training consistency with the PolicySearch_Agent.
- After an extensive grid_search on the agent's parameters (mu, theta, sigma, gamma, tau), it seems like the default ones in the sample code is optimal.
- I've heard it recommended to normalize the rewards to a range of -1 to 1, but that has the following issues:
- Most other successful reinforcement learning implementations don't seem to do this.
- It doesn't seem to make much sense why that would help.
- It didn't yield much better results for me in practice.
- It's really hard to iterate quickly and try to figure out what works when it takes a long time to train, even on a high end GPU. This is especially problematic with the DDPG approach. Other methods might be significantly faster to run.
- This is what the params in the Ornstein–Uhlenbeck Noise Process mean:
- mu: the mean, which controls exploration as learning progresses
- theta: the speed of the mean reversion
- sigma: the volatility
- The instructions suggested perhaps implementing the various types of tasks as helper functions (for example, takeoff, hover, land, etc.), but I thought it was a lot simpler and intuitive for this to be controlled directly within the `__init__` using a variable that can be passed in.
- Averaging the scores over the training period seems to be useful in seeing if the best score was just a fluke or if the agent is making serious progress.
- If programmatically trying out different values for the reward function, make sure to consider how those values affect the scores relative to the other training rounds; a higher score might just mean that crash penalties were lower. Also, *watch out for reward hacking!*
- Adding dropout layers doesn't seem to improve the learning (nor does it seem to hinder it), but it does seem to slightly speed up the processing.

Machine Learning Engineer Nanodegree   Teach a Quadcopter How to Fly

HIDE COMMENTS   ANSWER   REPORT

UPDATE: I've discovered the following two other potentially helpful points, but I'm still not getting acceptable performance from my agent.
- Don't use the rewards variable in the learn function to calculate scores, as those values will be out of context from the current state of the episode.
- If you want to run training for more than a few hundred episodes, you should increase the agent's `buffer_size` or else it may wander away from the optimal policy.

Nathaniel W 8 months ago REPORT

Hi!

I also had some hard issues with this task. Did you find the example code provided inside udacity itself? There is a more or less complete actor based on actor critic and a neuronal network that you can use. Just open up the project and go through each part. They are placed after the project workspace, what is a bad order in my opinion but a bit Offtopic. As a second hint, I reduced the runtime of each episode to only 1 instead of the default 5 - that allows you to run more iterations of your network in the same amount of time when you run it only on cpu. Of course it depends a bit on the task you implemented. I hope you get some progress, otherwise feel free to write again.

Darius

Darius M 8 months ago REPORT

Thank you Darius, yes I'm using the DDPG Actor Critic code that was provided after the workspace. One second seems a little fast. I want to pass this project, but I don't want to do it by making it too easy on the agent. But that is a very good point that I didn't think about, so I'm going to use a runtime of 3 seconds now.

Another UPDATE: If you're getting `nan` values for the simulation and/or errors from the sim, your activation functions might be outputting actions outside the 0-900 acceptable range. Some activations scale the output differently from others. Either change your activation function back or scale the output from your model to account for this.

Nathaniel W 7 months ago REPORT

2 Answers

4

Accepted Answer

I finally figured it out and got an agent that learns acceptably well!
- Stopping conditions are key to this project:
- Stopping the episode when the agent reaches the target.
It won't learn what you want if you don't acknowledge that it succeeded.
- Stopping training when it reliably succeeds, and before it forgets how to succeed.
With the Replay Buffer, it'll eventually forget helpful information. You can increase the buffer size, but then you also decrease the signal/noise ratio. Alternatively, you can stop training when the agent hits certain performance benchmarks, such as succeeding on 7 out of the last 10 episodes.

Additionally, I learned that a simpler reward function is better. I kept adding convergent goals to my reward function to help guide it to hitting the target, but that just ended up confusing it. Ultimately, I just had a small, simple convergent reward to let it know generally how it is doing, a terminal penalty for crashing, and a terminal reward for getting within a radius of the target. Oddly enough, I found that the agent would do naturally do all the things I'd specified in my convergent goals (such as not rolling too much and keeping all the propellers spinning at about the same speed), even though I wasn't rewarding it for doing them.

Nathaniel W 7 months ago

HIDE COMMENTS

REPORT

Hi Nathaniel (I'm also Nathaniel),

UPDATE: I don't know where my manners went. First off, thank you so much for posting your experience with this. I think a lot of us are having frustrations of our own. It feels like the lectures toward the end there sort of took a huge leap in complexity without really building up properly to it, but that's just my opinion. Anyway,

Can you provide any pointers as to how to implement this:

"- Stopping training when it reliably succeeds, and before it forgets how to succeed. With the Replay Buffer, it'll eventually forget helpful information. You can increase the buffer size, but then you also decrease the signal/noise ratio. Alternatively, you can stop training when the agent hits certain performance benchmarks, such as succeeding on 7 out of the last 10 episodes."

I haven't tried the quadcopter yet, but I think this might be a problem that I'm having with not being able to succeed with implementing the DDPG agent in Mountain Car and Pendulum. The agent starts to learn but then stops and doesn't really seem to come out from the hole it's fallen in.

Also, would you mind elaborating a bit more on what you mean by stopping training. Do you mean stop running episodes and just training the network parameters on what it is in the replay buffer at that time (because you know its useful?) or something else? For example, do you mean if it hits that benchmark you deem the agent to be successfully trained and call it a day?

Did you use the DDPG class provided by Udacity for your project?

Thank you!
Nate

(edited)

Nathaniel B 7 months ago REPORT

No worries. Yes, I did use the DDPG class provided by Udacity (also had tried making a Deep Q Network agent, but didn't find much success there). It worked okay, but definitely needed some tinkering to get it to do what I wanted.

I had turned the code to train the agent (provided in the Quadcopter notebook) into a function, so I could feed it an instance of the class and run training for a set number of episodes. After that, I modified it to detect if it was successful at the end of an episode (you can do this by comparing the position of the agent relative to the target position when done). Once you're detecting success/failure from within the train function, there are a few ways you can keep track of whether the last so many were successful, such as using a double ended queue, or a simple count variable. Then once the agent meets that criteria, you can break out of the `for i_episode in num_episodes:` loop; this leaves that instance of the agent (make sure to `return` the instance of the agent class from the train function) with the set of weights that had allowed it to succeed an acceptable number of times. So you may set `num_episodes = 2000` but if the agent meets your criteria, it may stop training after episode #142 or whatever.

I had also created a plotting function that runs an episode and plots the results, so then you can just feed the successful instance of the agent into the plotting function and it'll likely be a successful run.

Does that help clear it up? As for the replay buffer, see my comment on Chandramohan's question below, and let me know if there's more I can do to help explain it.

Nathaniel W 7 months ago REPORT

How did you stop it, did you just give it a very large reward when it hit the target?

Maximo G 6 months ago REPORT

If you change done to `True`, the episode stops. If you want to stop the training session, simply `Break` out of the for loop, or you can set a check condition on the training loop:

```
success_count = 0

while success_count < acceptable_threshold:
  <<do training>>
  if done and <<success criteria met>>:
    success_count += 1
  elif done and <<it crashed>>:
    success_count = 0
```

Ultimately, I just had a small, simple convergent reward to let it know generally how it is doing, a (large) terminal penalty for crashing, and a (large) terminal reward for getting within a radius of the target.

Nathaniel W 6 months ago REPORT

I posted the following on Slack, but it is a pretty good summary of my progress through this project, so I'm adding it here in case it's helpful for others to see my journey and thought process as things went along:

My first go at the reward function started with the biggest bonus for hitting the target at 80% of the total time, and an extra reward for minimal change in angular velocity (to reward "stable" flight that doesn't accelerate too quickly). After that didn't work, I added more bonuses/penalties for other aspects of performance, like a penalty if the difference between rotor speeds was drastic, and just to be sure that I was getting the balance between these reward sub-functions correct, I gave each of them a scalar variable and ran a random grid search for a long time to see which sub-functions had more/less impact on good convergence, but weeks later I got nowhere with this approach. Basically, I tried rewarding or penalizing every type of convergent goal that I could think of in ways that rewarded stable flight towards the target.

However, this just was very confusing for the model and didn't help anything. What I needed to do was get a better idea of when the model was reaching the target, and find a way to keep it doing that. Upon adding print statements for when the target was reached (I also recommend specifying a target area within a radius of the target, since the chances of 2 points colliding within 32bit floating point accuracy are near impossible) or when it crashed, I found that I could actually hit the target reliably after a hundred training episodes or so. The problem was that continuing training would make it "forget" how to reach the target after some time. That is due to your noise variables, which you could tune for full convergence, but I found a shortcut, which is stopping training early once a certain performance threshold is reached (say reaching the target 7 times in a row, or 9 out of the last 10 episodes).

Ultimately, you need a big reward for hitting the instrumental goal (reaching the target), and a big penalty for crashing, plus a small reward for getting close so that it can know which crashes are better than others. After getting that down, I actually found that my agent would reach the target with very conventionally stable flight (not spinning crazily on the way to get there, or zooming right up to it at full speed). Basically, it learned how to do all the convergent goals that I thought it needed to do to hit the target, but without me having to specify what those were.

Nathaniel W 5 months ago REPORT

Write a comment... If you are sharing an answer, please use the answer interface below.

▲
2
▼

The above comments are really helpful. I have more tips on this project. I used the DDPG code provided by the project. I found the model is good enough for this project but some parameters need to be changed, like gamma , tau or mu and so on. I also tried to learn from memories only when a episode is done (not each step). This really helps to converge fast. Finally, a big penalty and success reward value also help to converge faster .

xiuming Z 6 months ago

HIDE COMMENTS REPORT

When you say the model is good enough did you mean the default models provided in the class room/project?

Maximo G 6 months ago REPORT

yes. You do not need to modify the architecture, the activation function, stuff. But you might need to change gamma and tau.

xiuming Z 6 months ago REPORT

xiuming z - I saw that you used the DDPG code provided by the project - I'm having a hard time starting this project as I am not sure how to implement this code. Please contact me for a few question - kbenyehuda@yahoo.com Thanks!

Keren B 5 months ago REPORT

Hi Keren,

I'm happy to share what I learned on this project since it was extremely challenging, but I don't quite have the time to help 1-on-1 with it right now. I do pop into the #ml-reinforcement slack channel (https://mlnd.slack.com/messages/C0K448LBC) whenever possible to help out there, and there are also several others there that provide helpful advice. Please post your specific question(s) there for the best result, post in your study group, or create your own post here on Knowledge. If I see your post and have time, I'll do my best to help!

Good luck! I know you can do it!!

Nathaniel W 5 months ago REPORT

Write a comment... If you are sharing an answer, please use the answer interface below.