Deep Reinforcement Learning Nanodegree – Project 1 "Navigation"

Bernhard Langwallner

October 14, 2018

In this report I briefly summarize the learnings and final modeling decisions taken as part of the Continuous Control project. I first tried version 1 of the environment and stuck with it until I was able to solve it. Eventually, I was able to find a setup that solves the environment with around 250 steps but it took some optimizations from the original DDPG code from the Udacity repository to make it work.

Network Architectures.
Throughout my experimentation phase I did not change the basic network architecture much in terms of number of layers and number of units: it was always 2 fully connected hidden layers with ReLu activations for both the actor and the critic. I tried 64/64, 128/64 and 128/128 units for the two hidden layers and used 128/128 in my final setup. The main improvement to get the agent to train came from a suggestion in the Nanodegree Slack channel: When introducing batch normalization after the first hidden layer for the actor network the training started to get somewhere. Before (with the standard feedforward network and some optimizations outlined below), the training would stall at average score of $1-2$. I later decided to add one batch normalization layer also in the critic network. Looking at the training progress (see chart below), the smoothing/regularizing effect of batch normalization was pronounced in the sense that training progress started visibly after only a handful of episodes.

Further improvements.
An improvement that was already suggested in the benchmark implementation in the project description was gradient clipping:

$$\texttt{torch.nn.utils.clip\_grad\_norm\_}(\texttt{self.critic\_local.parameters}(), 1)$$

After not proceeding well in my early trials I decided to add the step in my code. I did not see any immediate benefit but kept the step in there in my final version.

Another idea from the Slack discussion was to set the weights of local and target actor, respectively, local and target critic to the same values: e.g.

$$\texttt{self.hard\_copy\_weights}(\texttt{self.actor\_target}, \texttt{self.actor\_local})$$

in the Agent's `init` function. I did this and found convergence to improve (however, the impact was not as strong as batch normalization).

I also experimented with a few different random seeds for the agent but did not see any major differences in convergence behavior.

Results.
As outlined above the environment could be solved in ~ 500 episodes. Below, we have the results of a training run with the final setup (individual episode scores are shown). After an initial phase with linear increasing trend in the scores the agent starts to quickly obtain higher scores after ~ 100 episodes. After ~ 180 episodes the progress seems to flatten but the average scores still rises until average score of 30 over 100 episodes is reached.

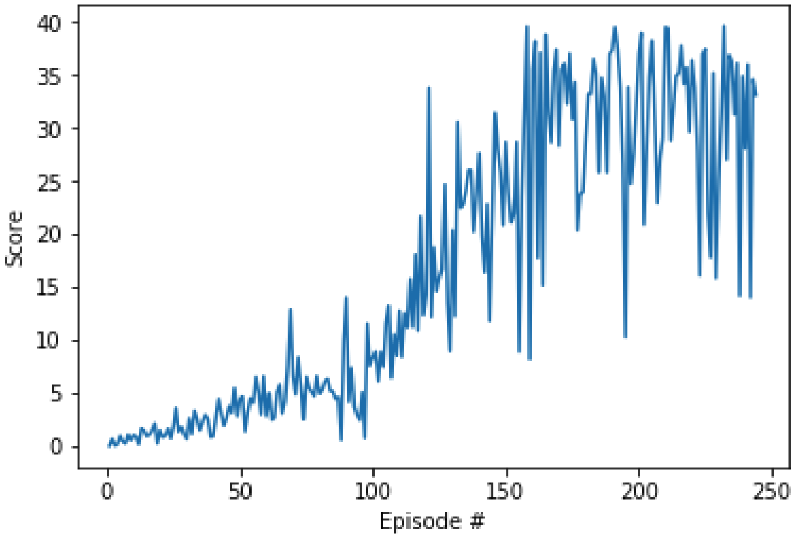

Figure 1: Example training run of the agent. The environment was solved in around 250 episodes.

Even later on in training there are some episodes with scores as low as $10-15$. It could be a potential direction for further work to eliminate or minimize these episodes.

Further work.
Besides the outlier episodes with lower scores, other directions of further work could include a more systematic study of the effect of batch normalization (e.g. for more layers) or dropout on the agents. As usual with experience replay, the sampling from the buffer is another crucial element of the model that could offer some levers of improvement.