Udacity Deep Reinforcement Learning Nanodegree Project

Collaboration and Competition

Author: Piotr Bazan

1. Introduction

In order to solve the Collaboration and Competition project I came up with an idea to split different parts of the problem into separate pieces (concerns) and implemented it in the object-orientated way so that each component could be either exchanged or re-implemented. This allowed experimenting and being open for future improvements.

2. Components

Experiment

`Experiment` is a central class for conducting training, evaluation and viewing sample play. In order to create it we need an agent and an environment.

- agent - there is only one implementation - `MIADDPG`
- environment - there is a wrapper around unity environment - `UnityEnv`

Environment

`UnityEnv` is an environment with API similar to openAi gym. This allows an abstraction over Unity environment.

Agent

`MIADDPG` – "Multi-Independent-Agent DDPG" consists of two DDPGAgents playing against each other. Each agent is independent in a sense it has it own local observations, actor, critic and reply buffer.

`DDPGAgent` class implements DDPG reinforcement learning algorithm and requires several components:

- actor - a Pytorch model for an actor – currently there is one model `FCDP` (fully connected deterministic policy)
- critic – a Pytorch model for a critic - currently there is one model `FCQV` (fully connected Q-value)
- memory - a reply buffer – currently only there is one buffer `ReplyBuffer`
- train_strategy – a strategy that supports continuous actions i.e. child of `ContinuousActionBaseStrategy` for example `NormalNoiseDecayStrategy`

The agent works in two modes: train and evaluation. During training is uses online actor to pick action which is passed to the environment. The resulting next state, rewards, terminal information and state are stored in reply buffer.
Once the reply buffer is filled with enough data (batch size * warm up batches) there is a sampling of data and both actor and critic are trained. During training online critic is used to evaluate the next state Q-value. After each training target actor and critic adjust their weights according to Polyak averaging of online actor and critic.
In evaluation mode the online actor is used to pick actions.

The FCDP model – the actor

Model consists of:

- input layer
- hidden layers
- rescale layer

The input layer accepts environment state as an input, then it applies RELU activation and passes result to hidden layers. Each of the hidden layer except the last one applies RELU. Last layer calculates TANH activation so that actions are distributed within <-1; 1> range. The rescale layer shifts action values to range required by the environment.

The FCQV model – the critic

Model consists of:

- input layer
- hidden layers
- output layer

The input layer accepts environment state. Than it calculates RELU activation. The results is concatenated with action tensor and put through hidden layers, each completed with RELU activation. At the end result is passed to output layer which does not have any activation function so that the Q-value is unlimited.

ReplyBuffer

Memory is a reply buffer (cyclic buffer) with limited size - I have chosen limit of 50_000 items. Each DDPGAgent has it own reply buffer.

Action picking strategy

Agent requires training and evaluation strategy.

- for training the `NormalNoiseDecayStrategy` – picks actions using actor and adds noise (with normal distribution, mean=0, standard deviation=.1). The class support also decaying mechanism so that noise (which can be seen as exploration) is decreased with each episode
- for evaluation the `ContinuousGreedyStrategy` – picks actions using actor. This can be seen as greedy strategy since there is no noise added

3. Hyper parameters for the agents

The agents have the following components:

- `actor = FCDP(nS, action_bounds, (128, 128)) – two hidden layers with 128 units`
- `critic = FCQV(nS, nA, (128, 128)) – two hidden layers with 128 units`
- `memory = ReplayBuffer(50000) – buffer size 50000`

where:

- `nS – state space size`
- `nA – action space size`
- `action_bounds – mininum and maximum values for each action (-1, 1)`

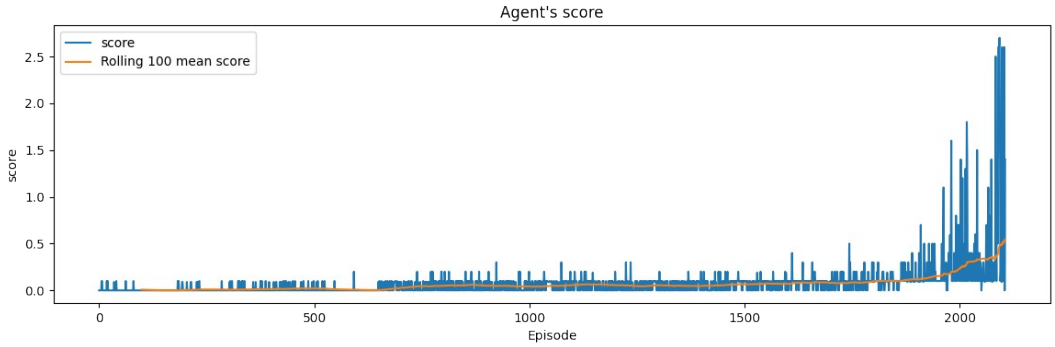
The agent has the following parameters:

- `gamma=.99 – discounted rate`
- `batch_size=256 – number of samples for training`
- `warm_up_batches=5 – required number of batches before training happens`
- `actor_lr=1e-4 – actor learning rate`
- `critic_lr=3e-4 – critic learning rate`
- `actor_max_grad_norm=float('inf') – actor maximum gradient during back propagation`
- `critic_max_grad_norm=float('inf') – similar to the above`
- `train_every_steps=1 – number of steps before training occurs`
- `update_target_every_steps=1 – number of steps before update occurs`
- `tau=.005 – weight for updating target actor/critic with online actor/critic parameters`

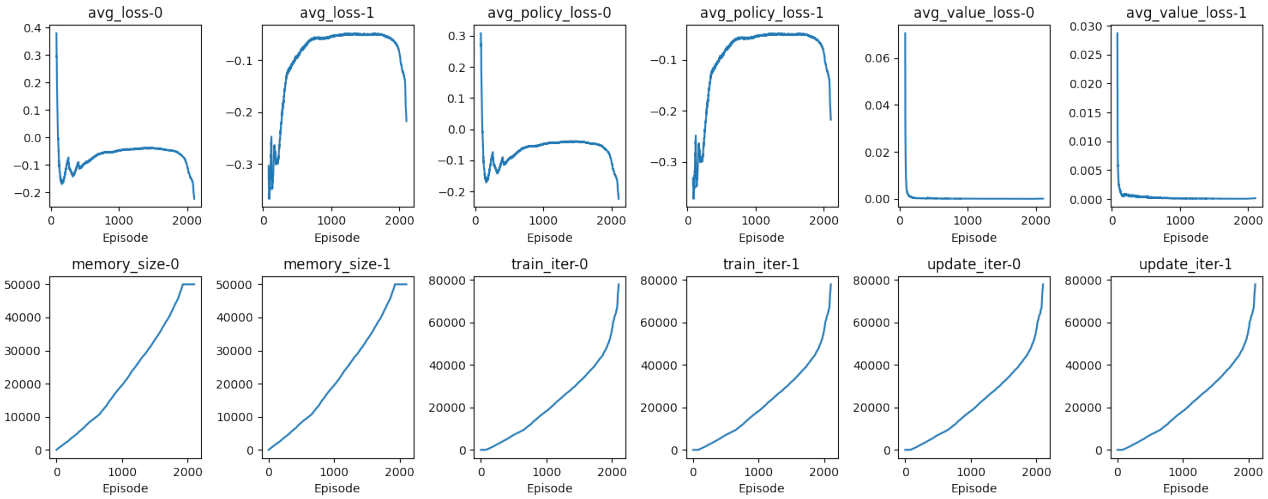
4. Experiment results

The experiment has target of mean of .5 points in 100 consecutive trials. Points are averaged between agents.

Agent passed grading achieving mean score .51. Number of required episodes is around 2100.



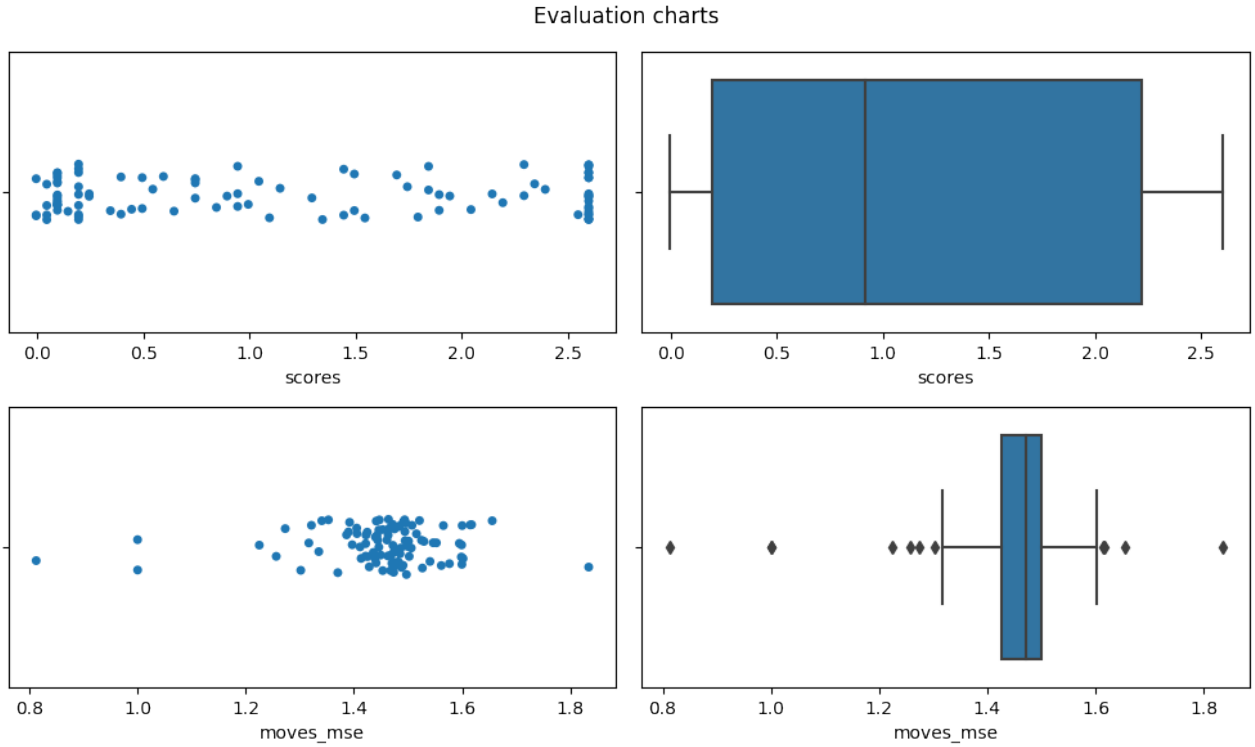
The agent had the following statistics:



5. Evaluation

When I was watching agents play I noticed that they are making many random, unnecessary moves. That is why I tried to come up with a metric i.e. a measure how many movements are done by calculating difference between adjacent actions of each agent and calculating MSE. The metric is called moves_mse.

I evaluated agent on 100 episodes and obtained the following results.



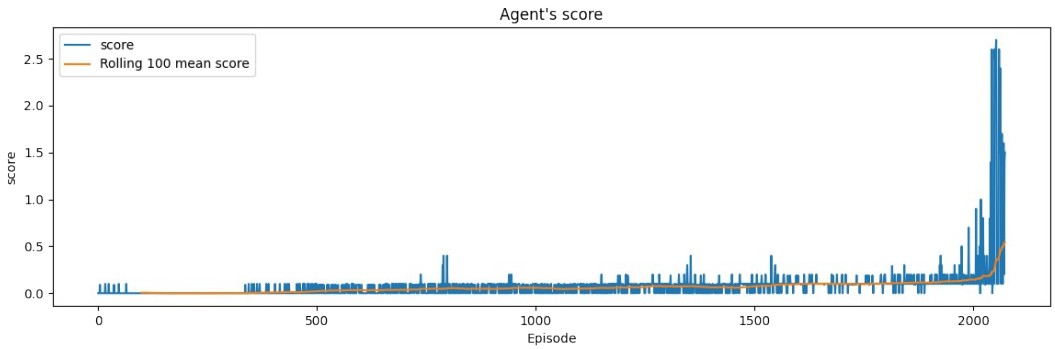
The median score is .92 and mean 1.15. The median moves_mse is 1.47, mean 1.46.

6. Ideas for improvement

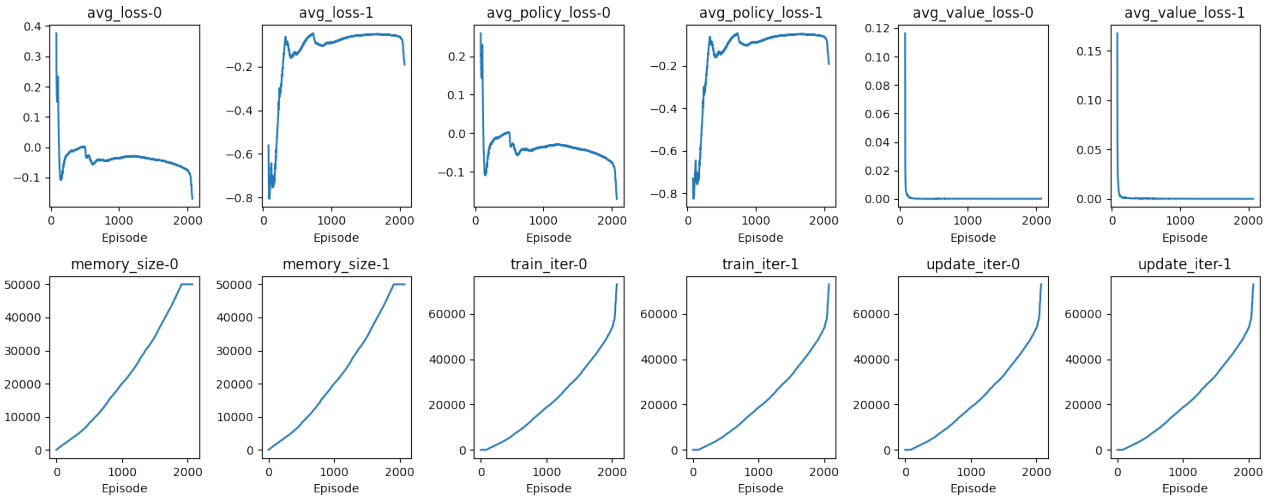
Each agent learns only to play on one side. I have an idea to switch agents each episode so that they play on left/right and vice versa. This could possibly speed up training and definitely improve generality as the agent could play on both sides.

7. Alternating agents – experiment results

The results of alternating agents

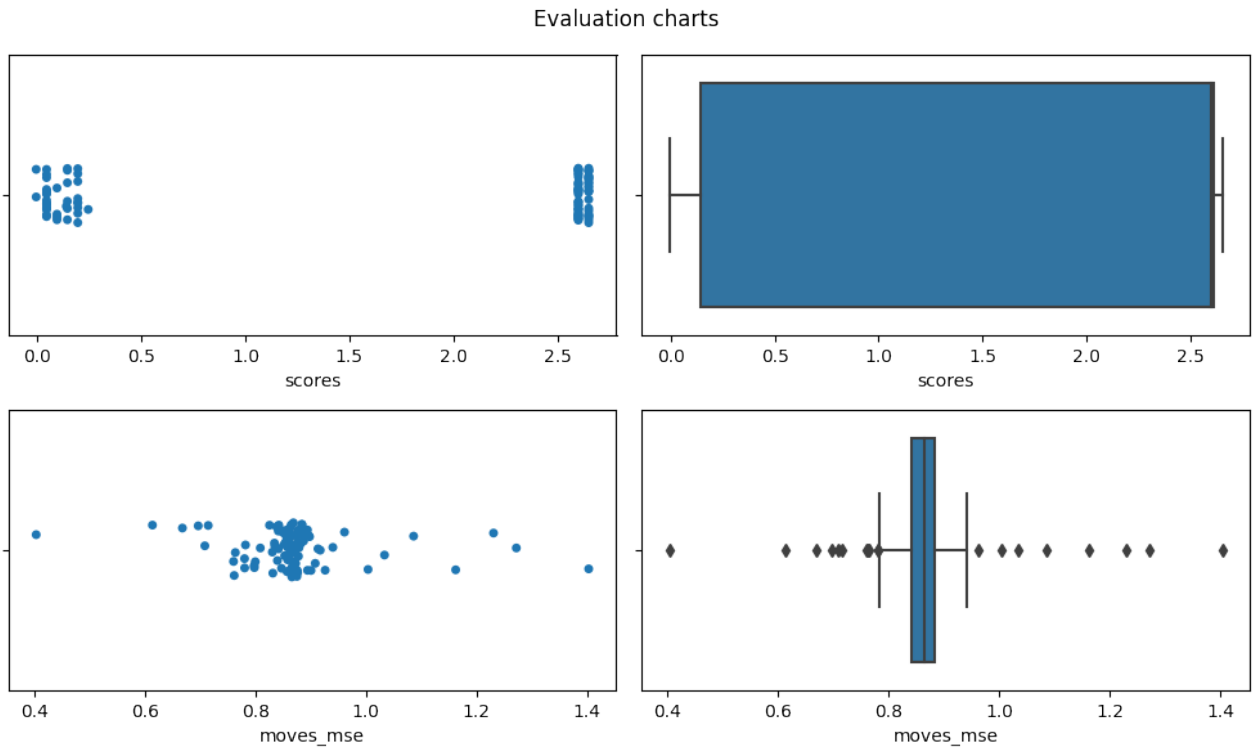


Agents' statistics



8. Alternating agents - evaluation

I evaluated agent 100 episodes.



The median score is 2.6 and mean 1.64. The median moves_mse is .85, mean .86. The agents performed pretty well. Also looking the play is more enjoyable as there are noticeable less random moves.

9. Ideas for future work

- Hyper parameters tuning
    - train/update every episodes – I would choose higher value to check how models behave, whether is can train quicker
    - tau – other values to consider .01 and .1 – see which gives better results
    - maximum gradient clipping – I used unlimited (infinity), also I checked value of 1 but the model learned really slowly. Perhaps some values like 10 would result in quicker and more stable learning
- Model architecture
    - currently actor and critic are similar in terms of hidden layers size. I would check different sizes for each model
- Adding PER – priority experience reply – this could speed up and stabilize training
- Trying different models – MADDPG – this could lead to quicker convergence and more stable training as the critic is trained on full observation i.e. local observations of the agents and their actions.