Udacity Deep Reinforcement Learning Nanodegree Project

Navigation

Author: Piotr Bazan

Introduction

In order to solve the Navigation 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.

Experiment

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

- agent - there is only one implementation - `DqnAgent`
- environment - there is only one implementation - `BananaEvn`

Environment

`BananaEnv` is an environment with API similar to openAi gym. This allows an abstraction on Unity environment.

Agent

`DqnAgent` class implements DQN reinforcement learning algorithm and requires several things:

- model - a pytorch model – currently only there is one model `DqnModel`
- memory - a reply buffer – currently only there is one buffer `ReplyBuffer`
- train_strategy - one of the strategies like `LinearEpsilonGreedyStrategy` or `ExponentialEpsilonGreedyStrategy`

The DqnModel

Model consists of:

- input layer
- hidden layers
- output layer

All layers except last one have activation function RELU. I picked the following architecture:

```
DqnModel(
  (layers): ModuleList(
    (0): Linear(in_features=37, out_features=64, bias=True)
    (1): ReLU()
    (2): Linear(in_features=64, out_features=64, bias=True)
    (3): ReLU()
    (4): Linear(in_features=64, out_features=4, bias=True)
  )
```

)

ReplyBuffer

Memory is a reply buffer with limited size - I have chosen limit of 50_000 items.

Action picking strategy

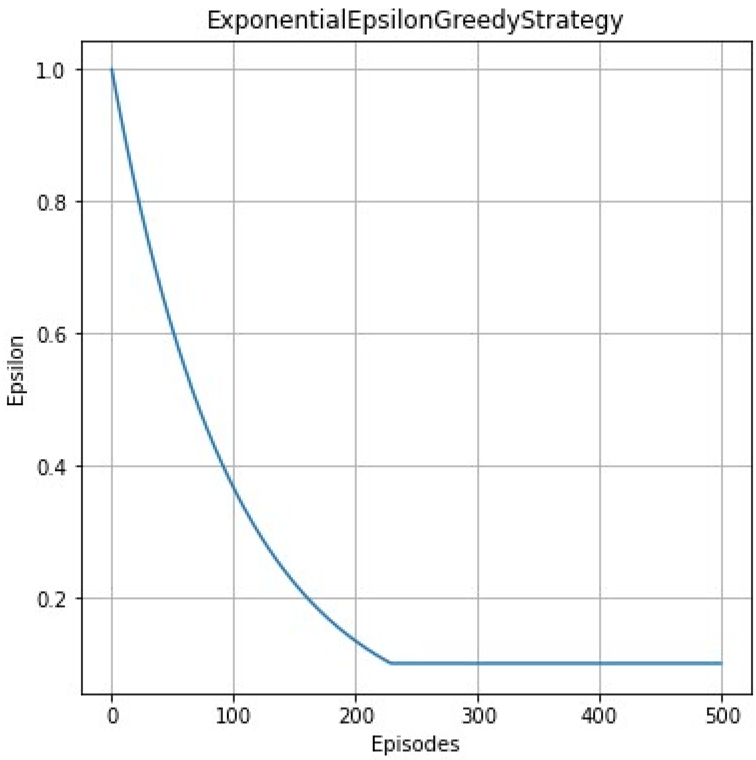
Agent requires training and evaluation strategy.

- for training we have two options: `LinearEpsilonGreedyStrategy` or `ExponentialEpsilonGreedyStrategy`
- for evaluation - there should be `GreedyStrategy`

For training I picked ExponentialEpsilonGreedyStrategy with parameters:

- eps_start = 1. - value at start
- eps_min = .1 - min. value
- decay = .99 – decay ratio.

Below is an example of strategy for 500 episodes



DqnAgent

The agent has two models

- online model
- target model - every several steps its weight are overwritten by online model weights

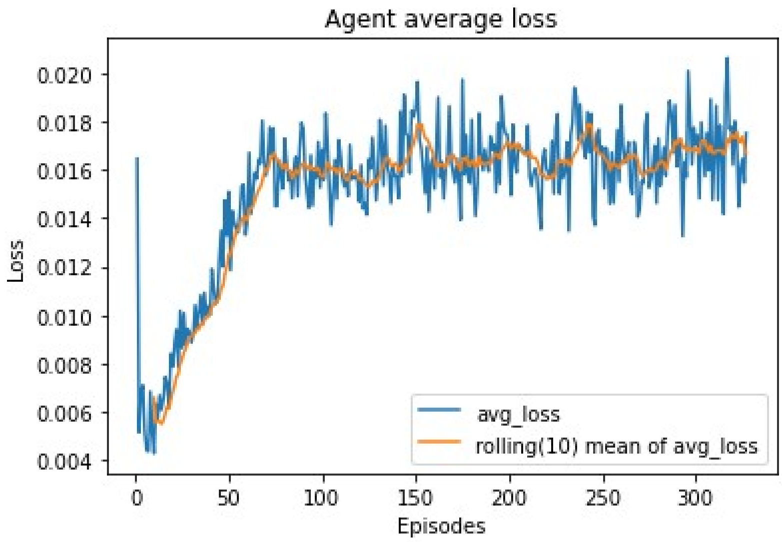
Hyperparameters:

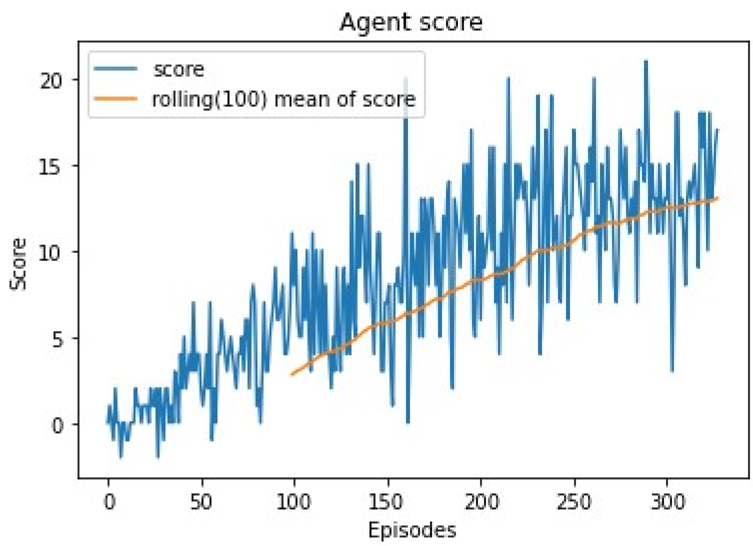
- `gamma` - discounted reward factor. I picked .95
- `batch_size` - how many samples to train at once. I picked 64.
- `warm_up_batches` - how many initials batches are required before starting training. Defaults to 5.
- `lr` - learning rate. I picked 5e-4
- `train_every_steps` - how often we want to train model. I picked 4 to speed things up.
- `updatet_target_every_steps` - how often we copy online model to target model. I picked 4.
- `tau` - factor for updating target model. When `tau=1` there is a copy of weights. When `tau < 1` there is a Polyak averaging. I picked .01

Dqn Experiment results

The experiment had set target of mean of 13 points in 100 consecutive trials.

Agent passed grading achieving min score:3.0, mean score: 13.02





Evaluation

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

scores	
count	100.000000
mean	14.710000
std	5.345242
min	0.000000
25%	12.000000
50%	15.500000
75%	19.000000
max	24.000000

Ideas for improvement

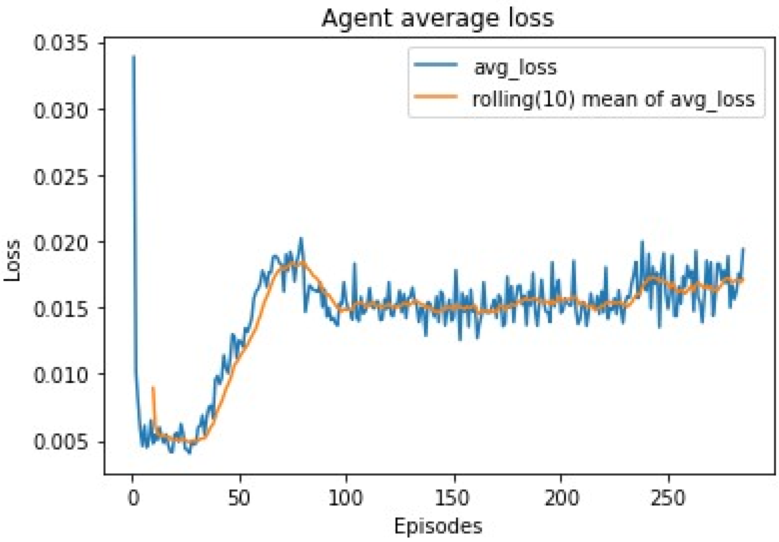
- Hyper parameters tuning
    - gamma – I would check lesser gamma 0.95 or .9
    - learning rate – I would increase learning rate to speed up training
    - train/update every episodes – I would increase to speed up training
    - tau – other values to consider .001 and .1 – see which gives better results
    - epsilon decay and min value – perhaps lesser decay would be better
- model architecture
    - hidden layer – I would check bigger and smaller architecture
- implementing DDQN model
- implementing Dueling DQN
- implementing prioritized replay buffer

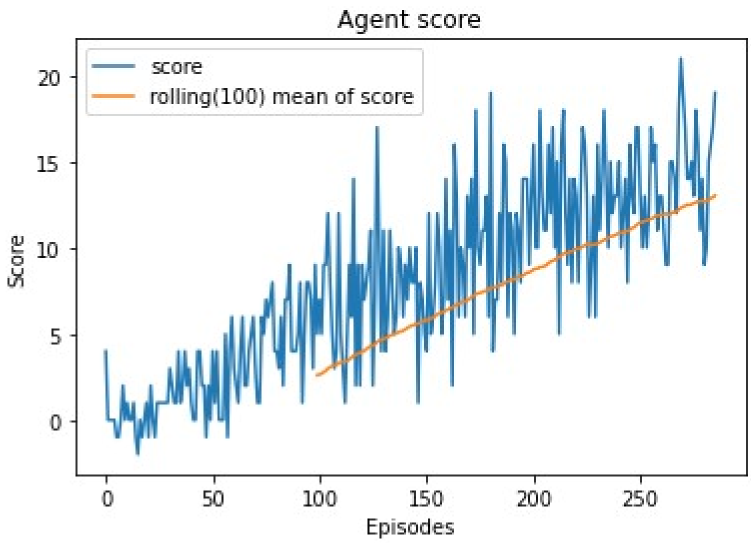
Improvement

I picked implementing DDQN as an improvement. I did not change any hyper parameters.

Results

Agent passed grading achieving min score:5.0, mean score: 13.03





Evaluation

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

scores	
count	100.00000
mean	10.43000
std	6.82724
min	-1.00000
25%	4.75000
50%	10.00000
75%	17.00000
max	24.0000

The evaluation results of DDQN are a bit worse than DQN (lesser mean 10.4 vs 14.7 and bigger standard deviation 6.8 vs 5.3). I may be due to the fact of random seeds not being set for Banana Environment during evaluation. Another improvement would be to gather data for larger number of episodes.