Contents

DDPG Model implementation

Description of DDGP model

The goal of this DDGP model is train an Agent composed of an actor (the Actor) whose task is to approximate the action value function Q(s,a) by minimizing the squared distance of the Actor's predicted action/value vs it equivalent Bellman Equation while also training a second agent (the Critic) whose task is to evaluate a given situation Q(s).

The model uses 2 networks(Critic and Actor) for solving the continuous control game. is a fully connected deep neural network with 2 hidden layers each containing 64 hidden nodes. This is the model that approximate the action value function Q(s,a) that guides the agent. Its output is a vector describing the respective

Agent Parameters:

BUFFER_SIZE = int(1e6) # replay buffer size

BATCH_SIZE = 512 # minibatch size

GAMMA = 0.99 # discount factor

TAU = 1e-3 # for soft update of target parameters

LR_ACTOR = 1e-3 # learning rate of the actor

LR_CRITIC = 1e-3 # learning rate of the critic

WEIGHT_DECAY = 0 # L2 weight decay

UPDATE_EVERY = 20 # how often to update the network

Parameters update

The parameters of the Agent are updated 10 times every 20 steps. The code that performs this is as follow:

```
# Learn, if enough samples are available in memory
        if len(self.memory) > BATCH_SIZE:
            for nbtimes in range(10):
                experiences = self.memory.sample()
                self.learn(experiences, GAMMA)
```

Agent's Actor and Critic network:

The model's architecture for approximating the Q(s,a) is the one described by the Google Deep Mind research paper. In pseudo code it performs the following.

The Actor's parameters:

The actor consist of one single hidden layer fully connected neural network of with one hidden layer of 128 nodes and an output layer of 4 nodes describing the action value function for each 4 possible actions. The output uses the tanH so that the model's output are between -1 and 1

The Critic's parameters:

The critic consist of a DNN fully connected with 3 hidden layers of 128,64,32 nodes reaching a final output of 1 that is between 0 and 1 to value the current state being passed to him.

The critic model also has a gradient clipping to improve performance

torch.nn.utils.clip_grad_norm(self.critic_local.parameters(), 1)

Batch normalization of actor and critic:

The input of both the actor and the critic use batch normalization.

```
super(Critic, self).__init__()

self.seed = torch.manual_seed(seed)

self.bn0 = nn.BatchNorm1d(state_size)
```

```
def forward(self, state, action):

    """Build a critic (value) network that maps (state, action) pairs -> Q-values."""

    stateN=self.bn0(state)

    xs = F.relu(self.fcs1(stateN))
```

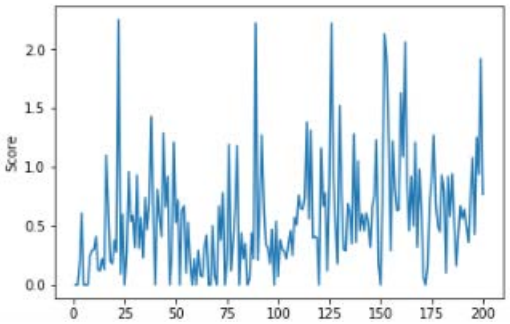
This has for effect to dampen the volatility that would be cause by an input with high low values. Example one dimension of the input is between -1000,1000 and the other one between 0-1, the first one will dominate the model and training will be poor.

Results of Model Training:

The model is run for 250 episodes with a maximal step of 500 per episode. One brain is used and not 20 since benchmark implementation shows model can be trained with one brain.

Final results after 200 iterations: 0.57

Graph of results



By simply looking at the graph, it appears that as soon as the model reaches values in the 2+ ranges it immediately reverts to values close to 0.

Potential Improvements:

Results of the training are poor even though multiples improvements have been made, all of those listed in this document (batchNorm, hyperparameter tweaking, NN architecture changes). Using 20 agents cannot make a meaningful difference currently not until the training phase can break above a meaningful threshold of results (20+ for 100 episodes).