rl-n-step-return-3-environments

March 29, 2023

[2]:
```
import numpy as np
import random
import torch
import torch.nn as nn
import torch.nn.functional as F
from collections import namedtuple, deque
import torch.optim as optim
import datetime
import gym
from gym.wrappers.record_video import RecordVideo
import glob
import io
import base64
import matplotlib.pyplot as plt
from IPython.display import HTML
from pyvirtualdisplay import Display
import tensorflow as tf
from IPython import display as ipythondisplay
from PIL import Image
import tensorflow_probability as tfp
```

[4]:
```
class ActorCriticModel(tf.keras.Model):
    """
    Defining policy and value networkss
    """
    def __init__(self, action_size,num_layers=2,neurons_each_layer=[1024,512]):
        super(ActorCriticModel, self).__init__()
        self.num_layers  =num_layers
        self.neurons_each_layer = neurons_each_layer
        self.linears = [tf.keras.layers.Dense(self.neurons_each_layer[0],␣
 ↪activation=tf.nn.relu)]
        self.linears.extend([tf.keras.layers.Dense(self.neurons_each_layer[i],␣
 ↪activation=tf.nn.relu) for i in range(1, self.num_layers)])

        #Output Layer for policy
        self.pi_out = tf.keras.layers.Dense(action_size, activation=tf.nn.
 ↪softmax)
```

```
        #Output Layer for state-value
        self.v_out = tf.keras.layers.Dense(1)

    def call(self, state):
        """
        Computes policy distribution and state-value for a given state
        """
        h = None
        for i in range(self.num_layers+1):
          if i == 0:
             h = tf.nn.relu(self.linears[0](state))
          elif i < self.num_layers:
             h = tf.nn.relu(self.linears[i](h))
          else:
             return self.pi_out(h), self.v_out(h)
```

[8]:
```
class Agent:
    """
    Agent class
    """
    def __init__(self, action_size, lr=0.001, gamma=0.99, seed = 85):
        self.gamma = gamma
        self.ac_model = ActorCriticModel(action_size=action_size)
        self.ac_model.compile(tf.keras.optimizers.Adam(learning_rate=lr))
        np.random.seed(seed)

    def sample_action(self, state):
        """
        Given a state, compute the policy distribution over all actions and␣
 ↪sample one action
        """
        pi,_ = self.ac_model(state)

        action_probabilities = tfp.distributions.Categorical(probs=pi)
        sample = action_probabilities.sample()

        return int(sample.numpy()[0])

    def actor_loss(self, action, pi, delta):
        """
        Compute Actor Loss
        """
        return -tf.math.log(pi[0,action]) * delta

    def critic_loss(self,delta):
        """
        Critic loss aims to minimize TD error
```

```
        """
        return delta**2

    @tf.function
    def learn_nstep_return(self, states, actions, rewards, next_states, dones,␣
↪n, T):
      with tf.GradientTape(persistent=True) as tape:
          delta_ts = []
          pis = []
          for t in range(0,T):
            delta_t = 0
            for t_ in range(t,t+n-1):
              delta_t += (((self.gamma)**(t_-t))*rewards[t_+1] if t_+1<␣
↪len(rewards) else 0)
            pi, V_s = self.ac_model(states[t])
            _, V_tn = self.ac_model(states[t+n]) if t+n < T else 0,0
            delta_t += (self.gamma**n*V_tn - V_s)
            V_s = tf.squeeze(V_s)
            pi, V_s = self.ac_model(states[t])
            delta_t = delta_t - V_s
            delta_ts.append(delta_t)
            pis.append(pi)

          loss_a = 0
          loss_c = 0
          for i in range(T):
            loss_a += self.actor_loss(actions[i], pis[i], delta_ts[i])
            loss_c += self.critic_loss(delta_ts[i])
          loss_total = loss_a + loss_c

      gradient = tape.gradient(loss_total, self.ac_model.trainable_variables,␣
↪unconnected_gradients=tf.UnconnectedGradients.ZERO)
      self.ac_model.optimizer.apply_gradients(zip(gradient, self.ac_model.
↪trainable_variables))
```

```
/usr/local/lib/python3.9/dist-packages/ipykernel/ipkernel.py:283:
DeprecationWarning: `should_run_async` will not call `transform_cell`
automatically in the future. Please pass the result to `transformed_cell`
argument and any exception that happen during thetransform in
`preprocessing_exc_tuple` in IPython 7.17 and above.
  and should_run_async(code)
```

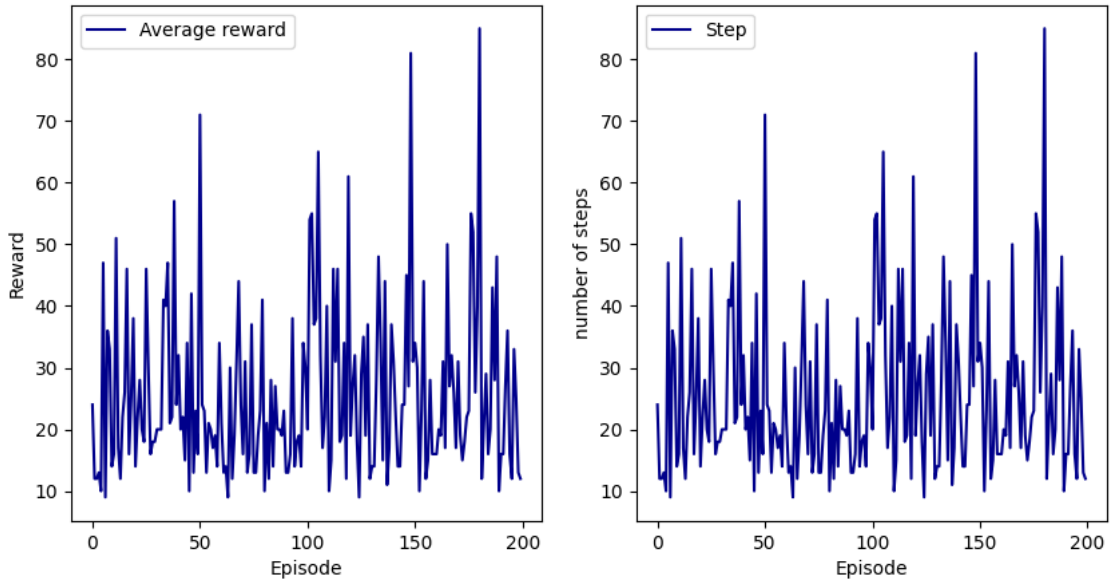
[ ]:
```
import warnings
warnings.filterwarnings("ignore", category=FutureWarning)
```

```
env = gym.make('CartPole-v1')

#Initializing Agent
agent = Agent(lr=1e-4, action_size=env.action_space.n)
#Number of episodes
episodes = 200
tf.compat.v1.reset_default_graph()

reward_list = []
average_reward_list = []
begin_time = datetime.datetime.now()
step= []
for ep in range(1, episodes + 1):
    state = env.reset().reshape(1,-1)
    done = False
    ep_rew = 0
    T = 0
    states,actions,next_states,rewards,dones = [],[],[],[0],[None]
    while not done:
        T+=1
        action = agent.sample_action(state) ##Sample Action
        next_state, reward, done, info = env.step(action) ##Take action
        next_state = next_state.reshape(1,-1)
        ep_rew += reward  ##Updating episode reward
        states.append(state)
        actions.append(action)
        next_states.append(next_state)
        rewards.append(reward)
        dones.append(done)
        state = next_state ##Updating State
    agent.learn_nstep_return(states, actions, rewards, next_states, dones,T,5)
    reward_list.append(ep_rew)
    step.append(T)

    if ep % 10 == 0:
        avg_rew = np.mean(reward_list[-10:])
        print('Episode ', ep, 'Reward %f' % ep_rew, 'Average Reward %f' %␣
 ↪avg_rew)

    if ep % 100:
        avg_100 =  np.mean(reward_list[-100:])
        if avg_100 > 195.0:
            print('Stopped at Episode ',ep-100)
            break

time_taken = datetime.datetime.now() - begin_time
print(time_taken)
```

```
/usr/local/lib/python3.9/dist-packages/gym/core.py:317: DeprecationWarning:
WARN: Initializing wrapper in old step API which returns one bool instead
of two. It is recommended to set `new_step_api=True` to use new step API. This
will be the default behaviour in future.
  deprecation(
/usr/local/lib/python3.9/dist-
packages/gym/wrappers/step_api_compatibility.py:39: DeprecationWarning:
WARN: Initializing environment in old step API which returns one bool
instead of two. It is recommended to set `new_step_api=True` to use new step
API. This will be the default behaviour in future.
  deprecation(
WARNING:tensorflow:5 out of the last 5 calls to <function
Agent.learn_nstep_return at 0x7f36785c51f0> triggered tf.function retracing.
Tracing is expensive and the excessive number of tracings could be due to (1)
creating @tf.function repeatedly in a loop, (2) passing tensors with different
shapes, (3) passing Python objects instead of tensors. For (1), please define
your @tf.function outside of the loop. For (2), @tf.function has
reduce_retracing=True option that can avoid unnecessary retracing. For (3),
please refer to https://www.tensorflow.org/guide/function#controlling_retracing
and https://www.tensorflow.org/api_docs/python/tf/function for  more details.
WARNING:tensorflow:6 out of the last 6 calls to <function
Agent.learn_nstep_return at 0x7f36785c51f0> triggered tf.function retracing.
Tracing is expensive and the excessive number of tracings could be due to (1)
creating @tf.function repeatedly in a loop, (2) passing tensors with different
shapes, (3) passing Python objects instead of tensors. For (1), please define
your @tf.function outside of the loop. For (2), @tf.function has
reduce_retracing=True option that can avoid unnecessary retracing. For (3),
please refer to https://www.tensorflow.org/guide/function#controlling_retracing
and https://www.tensorflow.org/api_docs/python/tf/function for  more details.

Episode  10 Reward 14.000000 Average Reward 21.000000
Episode  20 Reward 38.000000 Average Reward 26.900000
Episode  30 Reward 18.000000 Average Reward 23.000000
Episode  40 Reward 24.000000 Average Reward 31.200000
Episode  50 Reward 16.000000 Average Reward 22.700000
Episode  60 Reward 34.000000 Average Reward 25.600000
Episode  70 Reward 24.000000 Average Reward 21.700000
Episode  80 Reward 41.000000 Average Reward 22.200000
Episode  90 Reward 23.000000 Average Reward 19.400000
Episode  100 Reward 30.000000 Average Reward 20.900000
Episode  110 Reward 40.000000 Average Reward 38.000000
Episode  120 Reward 61.000000 Average Reward 29.200000
Episode  130 Reward 12.000000 Average Reward 23.700000
Episode  140 Reward 37.000000 Average Reward 26.200000
Episode  150 Reward 31.000000 Average Reward 31.300000
Episode  160 Reward 16.000000 Average Reward 22.800000
```

```
Episode  170 Reward 17.000000 Average Reward 25.500000
Episode  180 Reward 40.000000 Average Reward 30.100000
Episode  190 Reward 10.000000 Average Reward 31.200000
Episode  200 Reward 12.000000 Average Reward 20.600000
0:03:17.331154
```

[ ]:
```
<matplotlib.legend.Legend at 0x7f34a2db47f0>
```



[11]:
```
import warnings
warnings.filterwarnings("ignore", category=FutureWarning)



env = gym.make('Acrobot-v1')

#Initializing Agent
agent = Agent(lr=1e-4, action_size=env.action_space.n)
#Number of episodes
episodes = 300
tf.compat.v1.reset_default_graph()

reward_list = []
average_reward_list = []
begin_time = datetime.datetime.now()
step= []
for ep in range(1, episodes + 1):
    state = env.reset().reshape(1,-1)
    done = False
```

```
    ep_rew = 0
    T = 0
    states,actions,next_states,rewards,dones = [],[],[],[0],[None]
    while not done:
        T+=1
        action = agent.sample_action(state) ##Sample Action
        next_state, reward, done, info = env.step(action) ##Take action
        next_state = next_state.reshape(1,-1)
        ep_rew += reward  ##Updating episode reward
        states.append(state)
        actions.append(action)
        next_states.append(next_state)
        rewards.append(reward)
        dones.append(done)
        state = next_state ##Updating State
    agent.learn_nstep_return(states, actions, rewards, next_states, dones,T,5)
    reward_list.append(ep_rew)
    step.append(T)

    if ep % 10 == 0:
        avg_rew = np.mean(reward_list[-10:])
        print('Episode ', ep, 'Reward %f' % ep_rew, 'Average Reward %f' %␣
↪avg_rew)

    if ep % 100:
        avg_100 =  np.mean(reward_list[-100:])
        if avg_100 > -100.0:
            print('Stopped at Episode ',ep-100)
            break

time_taken = datetime.datetime.now() - begin_time
print(time_taken)
```

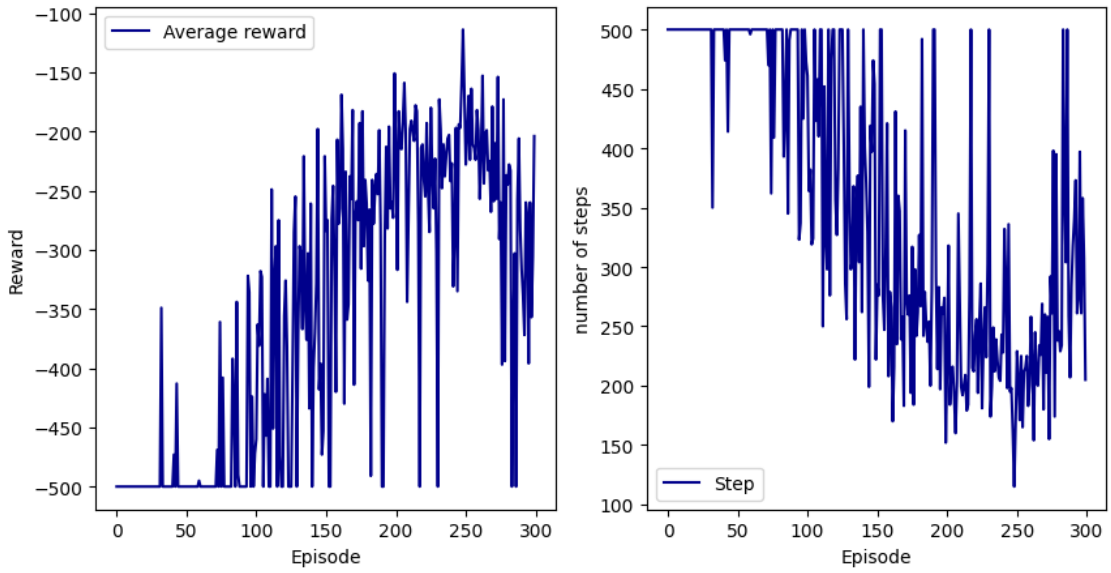
```
/usr/local/lib/python3.9/dist-packages/gym/core.py:317: DeprecationWarning:
WARN: Initializing wrapper in old step API which returns one bool instead
of two. It is recommended to set `new_step_api=True` to use new step API. This
will be the default behaviour in future.
  deprecation(
/usr/local/lib/python3.9/dist-
packages/gym/wrappers/step_api_compatibility.py:39: DeprecationWarning:
WARN: Initializing environment in old step API which returns one bool
instead of two. It is recommended to set `new_step_api=True` to use new step
API. This will be the default behaviour in future.
  deprecation(
WARNING:tensorflow:5 out of the last 5 calls to <function
```

```
Agent.learn_nstep_return at 0x7fae84258670> triggered tf.function retracing.
Tracing is expensive and the excessive number of tracings could be due to (1)
creating @tf.function repeatedly in a loop, (2) passing tensors with different
shapes, (3) passing Python objects instead of tensors. For (1), please define
your @tf.function outside of the loop. For (2), @tf.function has
reduce_retracing=True option that can avoid unnecessary retracing. For (3),
please refer to https://www.tensorflow.org/guide/function#controlling_retracing
and https://www.tensorflow.org/api_docs/python/tf/function for  more details.
WARNING:tensorflow:6 out of the last 6 calls to <function
Agent.learn_nstep_return at 0x7fae84258670> triggered tf.function retracing.
Tracing is expensive and the excessive number of tracings could be due to (1)
creating @tf.function repeatedly in a loop, (2) passing tensors with different
shapes, (3) passing Python objects instead of tensors. For (1), please define
your @tf.function outside of the loop. For (2), @tf.function has
reduce_retracing=True option that can avoid unnecessary retracing. For (3),
please refer to https://www.tensorflow.org/guide/function#controlling_retracing
and https://www.tensorflow.org/api_docs/python/tf/function for  more details.

Episode  10 Reward -500.000000 Average Reward -500.000000
Episode  20 Reward -500.000000 Average Reward -500.000000
Episode  30 Reward -500.000000 Average Reward -500.000000
Episode  40 Reward -500.000000 Average Reward -484.900000
Episode  50 Reward -500.000000 Average Reward -488.600000
Episode  60 Reward -495.000000 Average Reward -499.500000
Episode  70 Reward -500.000000 Average Reward -500.000000
Episode  80 Reward -500.000000 Average Reward -473.800000
Episode  90 Reward -500.000000 Average Reward -465.600000
Episode  100 Reward -472.000000 Average Reward -455.400000
Episode  110 Reward -500.000000 Average Reward -413.300000
Episode  120 Reward -500.000000 Average Reward -407.000000
Episode  130 Reward -500.000000 Average Reward -400.500000
Episode  140 Reward -261.000000 Average Reward -327.900000
Episode  150 Reward -221.000000 Average Reward -375.700000
Episode  160 Reward -278.000000 Average Reward -330.000000
Episode  170 Reward -182.000000 Average Reward -269.500000
Episode  180 Reward -261.000000 Average Reward -271.100000
Episode  190 Reward -279.000000 Average Reward -282.000000
Episode  200 Reward -151.000000 Average Reward -293.000000
Episode  210 Reward -268.000000 Average Reward -231.200000
Episode  220 Reward -211.000000 Average Reward -236.100000
Episode  230 Reward -302.000000 Average Reward -243.000000
Episode  240 Reward -242.000000 Average Reward -243.900000
Episode  250 Reward -167.000000 Average Reward -221.000000
Episode  260 Reward -199.000000 Average Reward -202.000000
Episode  270 Reward -179.000000 Average Reward -217.800000
Episode  280 Reward -237.000000 Average Reward -263.200000
Episode  290 Reward -283.000000 Average Reward -312.300000
Episode  300 Reward -204.000000 Average Reward -308.300000
```

0:20:13.373106

[13]:
```
plt.figure(figsize = (10,5))
plt.subplot(121)
plt.plot(np.arange(len(reward_list)),reward_list, color =␣
 ↪'darkblue',label='Average reward')
plt.xlabel("Episode")
plt.ylabel("Reward")
plt.legend()
plt.subplot(122)
plt.plot(np.arange(len(step)),step, color = 'darkblue',label='Step')
plt.xlabel("Episode")
plt.ylabel("number of steps")
plt.legend()
```

[13]: <matplotlib.legend.Legend at 0x7fae096ef940>



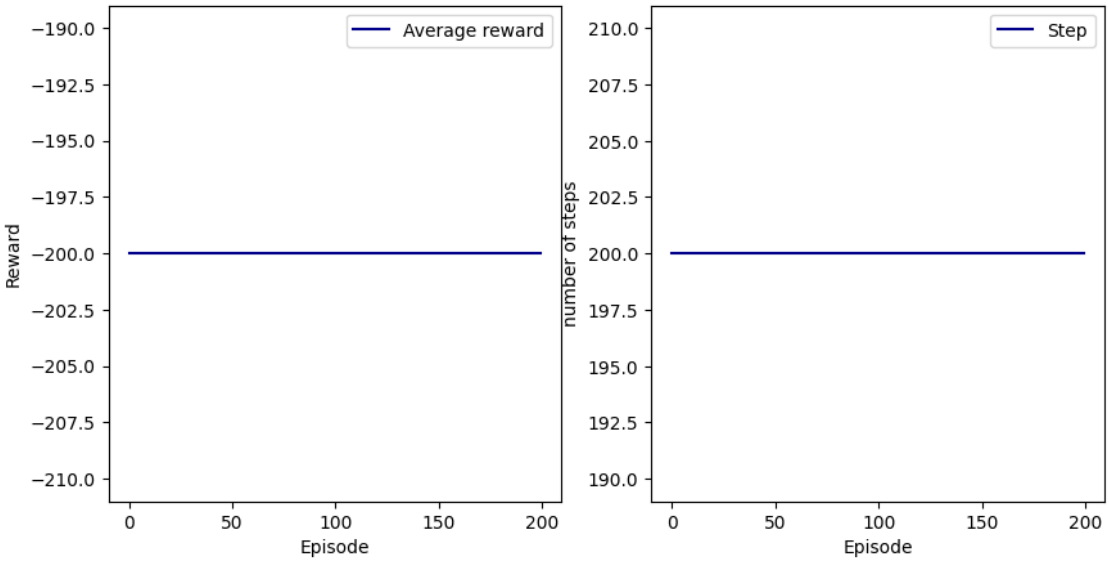
[15]:
```
import warnings
warnings.filterwarnings("ignore", category=FutureWarning)



env = gym.make('MountainCar-v0')

#Initializing Agent
agent = Agent(lr=1e-4, action_size=env.action_space.n)
#Number of episodes
episodes = 200
```

```
tf.compat.v1.reset_default_graph()

reward_list = []
average_reward_list = []
begin_time = datetime.datetime.now()
step= []
for ep in range(1, episodes + 1):
    state = env.reset().reshape(1,-1)
    done = False
    ep_rew = 0
    T = 0
    states,actions,next_states,rewards,dones = [],[],[],[0],[None]
    while not done:
        T+=1
        action = agent.sample_action(state) ##Sample Action
        next_state, reward, done, info = env.step(action) ##Take action
        next_state = next_state.reshape(1,-1)
        ep_rew += reward  ##Updating episode reward
        states.append(state)
        actions.append(action)
        next_states.append(next_state)
        rewards.append(reward)
        dones.append(done)
        state = next_state ##Updating State
    agent.learn_nstep_return(states, actions, rewards, next_states, dones,T,5)
    reward_list.append(ep_rew)
    step.append(T)

    if ep % 10 == 0:
        avg_rew = np.mean(reward_list[-10:])
        print('Episode ', ep, 'Reward %f' % ep_rew, 'Average Reward %f' %␣
 ↪avg_rew)

    if ep % 100:
        avg_100 =  np.mean(reward_list[-100:])
        if avg_100 > -110.0:
            print('Stopped at Episode ',ep-100)
            break

time_taken = datetime.datetime.now() - begin_time
print(time_taken)

plt.figure(figsize = (10,5))
plt.subplot(121)
plt.plot(np.arange(len(reward_list)),reward_list, color =␣
 ↪'darkblue',label='Average reward')
plt.xlabel("Episode")
```

```
plt.ylabel("Reward")
plt.legend()
plt.subplot(122)
plt.plot(np.arange(len(step)),step, color = 'darkblue',label='Step')
plt.xlabel("Episode")
plt.ylabel("number of steps")
plt.legend()
```

```
/usr/local/lib/python3.9/dist-packages/gym/core.py:317: DeprecationWarning:
WARN: Initializing wrapper in old step API which returns one bool instead
of two. It is recommended to set `new_step_api=True` to use new step API. This
will be the default behaviour in future.
  deprecation(
/usr/local/lib/python3.9/dist-
packages/gym/wrappers/step_api_compatibility.py:39: DeprecationWarning:
WARN: Initializing environment in old step API which returns one bool
instead of two. It is recommended to set `new_step_api=True` to use new step
API. This will be the default behaviour in future.
  deprecation(
Episode  10 Reward -200.000000 Average Reward -200.000000
Episode  20 Reward -200.000000 Average Reward -200.000000
Episode  30 Reward -200.000000 Average Reward -200.000000
Episode  40 Reward -200.000000 Average Reward -200.000000
Episode  50 Reward -200.000000 Average Reward -200.000000
Episode  60 Reward -200.000000 Average Reward -200.000000
Episode  70 Reward -200.000000 Average Reward -200.000000
Episode  80 Reward -200.000000 Average Reward -200.000000
Episode  90 Reward -200.000000 Average Reward -200.000000
Episode  100 Reward -200.000000 Average Reward -200.000000
Episode  110 Reward -200.000000 Average Reward -200.000000
Episode  120 Reward -200.000000 Average Reward -200.000000
Episode  130 Reward -200.000000 Average Reward -200.000000
Episode  140 Reward -200.000000 Average Reward -200.000000
Episode  150 Reward -200.000000 Average Reward -200.000000
Episode  160 Reward -200.000000 Average Reward -200.000000
Episode  170 Reward -200.000000 Average Reward -200.000000
Episode  180 Reward -200.000000 Average Reward -200.000000
Episode  190 Reward -200.000000 Average Reward -200.000000
Episode  200 Reward -200.000000 Average Reward -200.000000
0:09:48.248798
```

[15]: <matplotlib.legend.Legend at 0x7fae15ef6af0>


Average reward
−190.0
−192.5
−195.0
−197.5
−200.0
−202.5
−205.0
−207.5
−210.0
Reward
0
50
100
150
200
Episode
Step
210.0
207.5
205.0
202.5
200.0
197.5
195.0
192.5
190.0
number of steps
0
50
100
150
200
Episode