# Install gym Atari (pong is in this environment as a demo) in Windows

1. Install gym

pip install gym

pip install gym[accept-rom-license]

1. install VS build tools

* or we will have an error if we directly install gym Atari :

OSError: [WinError 126]

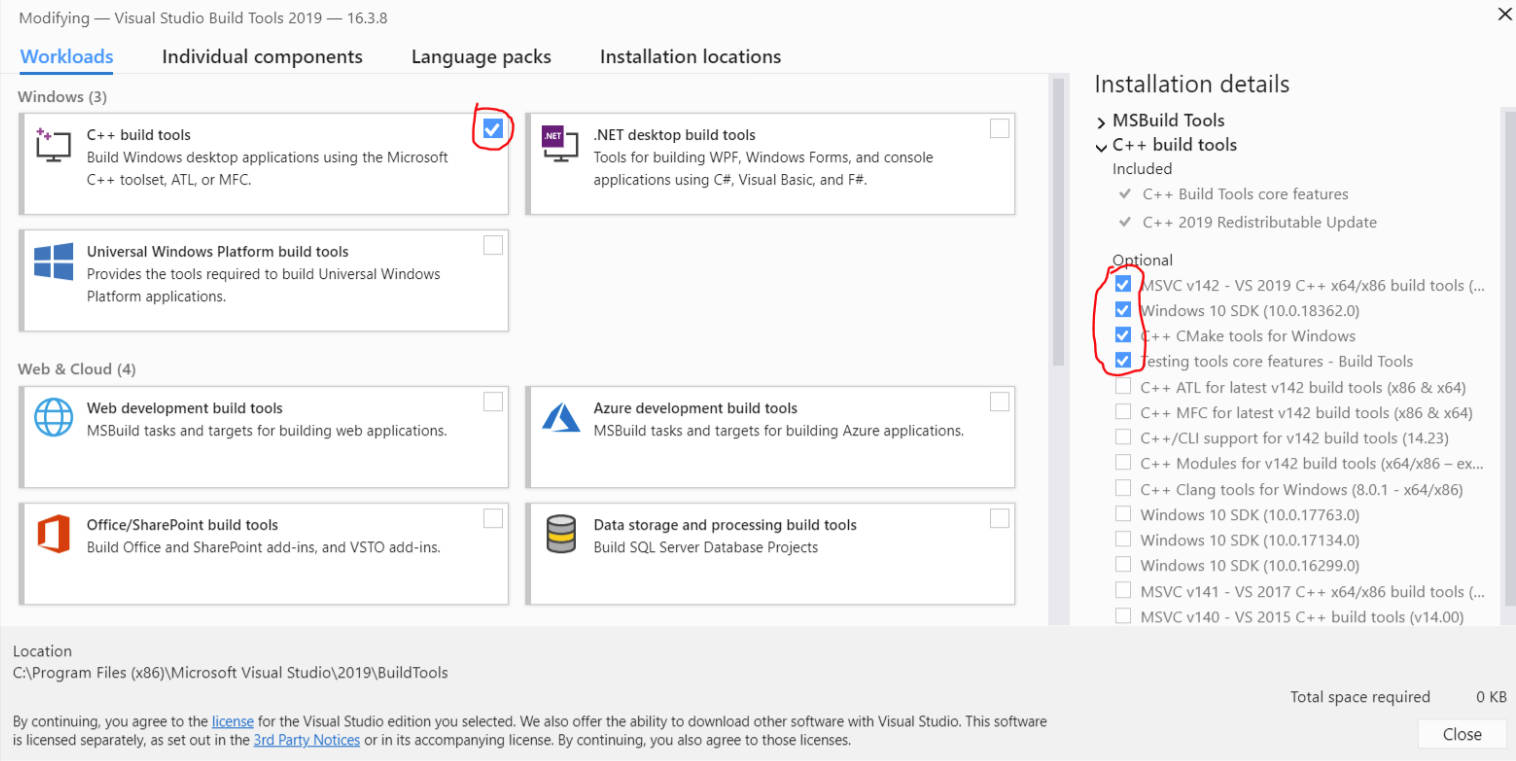
* If we have this error, we should uninstall gym[Atari]

pip uninstall atari-py

pip uninstall gym[atari]

* Then we download VS tools

<https://visualstudio.microsoft.com/thank-you-downloading-visual-studio/?sku=BuildTools&rel=16>



1. Restart the computer and run the commands

pip install cmake

pip install atari-py

pip install gym[atari]

1. Test gym in python

import atari\_py

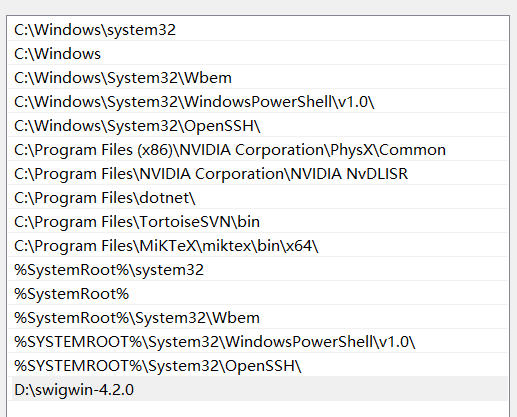
print(atari\_py.list\_games())

1. install box2d environment (optional)

* download swigwin

<http://www.swig.org/download.html>

* Extract the file and add the extracted root directory (xx\swigwin) to the environment variable: path



* Install box2d, ale-py, pyglet, opencv-python

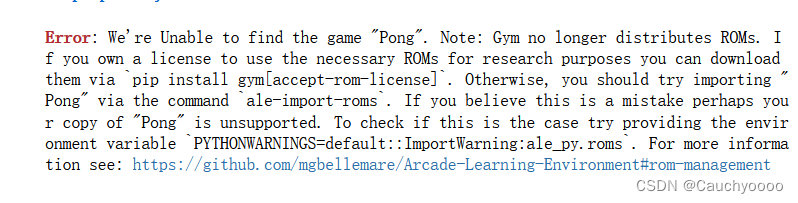
pip install gym[box2d]

pip install ale-py

pip install pyglet

pip install opencv-python

1. if we are unable to find the game “pong”



* Download a file in

<http://www.atarimania.com/rom_collection_archive_atari_2600_roms.html>



* Then extract this file
* Install autorom

pip install autorom

* + then import autorom

AutoROM –-accept-license

1. Check gym Atari:
   * Print the game names

import gym

from gym import envs

env\_games = [spec.id for spec in env.registry.all()]

for name in sorted(env\_names):

print(name)

* run the game “pong”

import gym

env = gym.make("Pong-ram-v4", render\_mode='human') # ansi | human | rgb\_array  
observation = env.reset()  
for t in range(1000):  
 #env.render()  
 action = env.action\_space.sample()  
 observation, reward, done, \_, info = env.step(action)  
 print(action)  
  
 if done:  
 observation = env.reset()  
env.close()

# A demo of pong for PPO in pytorch

import random  
import gym  
import numpy as np  
from PIL import Image  
import torch  
from torch.nn import functional as F  
from torch import nn  
  
class Policy(nn.Module):  
 def \_\_init\_\_(self):  
 super(Policy, self).\_\_init\_\_()  
  
 self.gamma = 0.99  
 self.eps\_clip = 0.1  
  
 self.layers = nn.Sequential(  
 nn.Linear(6000, 512), nn.ReLU(),  
 nn.Linear(512, 2),  
 )  
  
 def state\_to\_tensor(self, It):  
 *""" prepro 210x160x3 uint8 frame into 6000 (75x80) 1D float vector. See Karpathy's post: http://karpathy.github.io/2016/05/31/rl/ """* if It is None:  
 *#if len(I) == 0:* return torch.zeros(1, 6000)  
 if len(It) == 2: *# It is a tuple or a matrix* I = It[0]  
 else:  
 I = It  
 I = I[35:185] *# crop - remove 35px from start & 25px from end of image in x, to reduce redundant parts of image (i.e. after ball passes paddle)* I = I[::2,::2,0] *# downsample by factor of 2.* I[I == 144] = 0 *# erase background (background type 1)* I[I == 109] = 0 *# erase background (background type 2)* I[I != 0] = 1 *# everything else (paddles, ball) just set to 1. this makes the image grayscale effectively* return torch.from\_numpy(I.astype(np.float32).ravel()).unsqueeze(0)  
  
 def pre\_process(self, x, prev\_x):  
 aa = self.state\_to\_tensor(x)  
 bb = self.state\_to\_tensor(prev\_x)  
 return aa - bb  
  
 def convert\_action(self, action):  
 return action + 2  
  
 def forward(self, d\_obs, action=None, action\_prob=None, advantage=None, deterministic=False):  
 if action is None:  
 with torch.no\_grad():  
 logits = self.layers(d\_obs)  
 if deterministic:  
 action = int(torch.argmax(logits[0]).detach().cpu().numpy())  
 action\_prob = 1.0  
 else:  
 c = torch.distributions.Categorical(logits=logits)  
 action = int(c.sample().cpu().numpy()[0])  
 action\_prob = float(c.probs[0, action].detach().cpu().numpy())  
 return action, action\_prob  
 '''  
 # policy gradient (REINFORCE)  
 logits = self.layers(d\_obs)  
 loss = F.cross\_entropy(logits, action, reduction='none') \* advantage  
 return loss.mean()  
 '''  
  
 *# PPO* vs = np.array([[1., 0.], [0., 1.]])  
 ts = torch.FloatTensor(vs[action.cpu().numpy()])  
  
 logits = self.layers(d\_obs)  
 r = torch.sum(F.softmax(logits, dim=1) \* ts, dim=1) / action\_prob  
 loss1 = r \* advantage  
 loss2 = torch.clamp(r, 1-self.eps\_clip, 1+self.eps\_clip) \* advantage  
 loss = -torch.min(loss1, loss2)  
 loss = torch.mean(loss)  
  
 return loss  
  
env = gym.make('PongNoFrameskip-v4')  
env.reset()  
  
policy = Policy()  
  
opt = torch.optim.Adam(policy.parameters(), lr=1e-3)  
  
reward\_sum\_running\_avg = None  
for it in range(100000):  
 d\_obs\_history, action\_history, action\_prob\_history, reward\_history = [], [], [], []  
 for ep in range(10):  
 obs, prev\_obs = env.reset(), None *# obs is a tuple and prev\_obs is None* for t in range(190000):  
 *#env.render()  
  
 #d\_obs = policy.pre\_process(np.array(obs), np.array(prev\_obs))* d\_obs = policy.pre\_process(obs, prev\_obs)  
  
 with torch.no\_grad():  
 action, action\_prob = policy(d\_obs)  
  
 prev\_obs = obs *# if obs is a tuple, prev\_obs is a tuple; if obs is a matrix, prev\_obs is a matrix* obs, reward, done, \_, info = env.step(policy.convert\_action(action)) *# now obs is outputted as a matrix* d\_obs\_history.append(d\_obs)  
 action\_history.append(action)  
 action\_prob\_history.append(action\_prob)  
 reward\_history.append(reward)  
  
 if done:  
 reward\_sum = sum(reward\_history[-t:])  
 reward\_sum\_running\_avg = 0.99\*reward\_sum\_running\_avg + 0.01\*reward\_sum if reward\_sum\_running\_avg else reward\_sum  
 print('Iteration %d, Episode %d (%d timesteps) - last\_action: %d, last\_action\_prob: %.2f, reward\_sum: %.2f, running\_avg: %.2f' % (it, ep, t, action, action\_prob, reward\_sum, reward\_sum\_running\_avg))  
 break  
  
 *# compute advantage* R = 0  
 discounted\_rewards = []  
  
 for r in reward\_history[::-1]:  
 if r != 0: R = 0 *# scored/lost a point in pong, so reset reward sum* R = r + policy.gamma \* R  
 discounted\_rewards.insert(0, R)  
  
 discounted\_rewards = torch.FloatTensor(discounted\_rewards)  
 discounted\_rewards = (discounted\_rewards - discounted\_rewards.mean()) / discounted\_rewards.std()  
  
 *# update policy* for \_ in range(5):  
 n\_batch = 24576  
 idxs = random.sample(range(len(action\_history)), n\_batch)  
 d\_obs\_batch = torch.cat([d\_obs\_history[idx] for idx in idxs], 0)  
 action\_batch = torch.LongTensor([action\_history[idx] for idx in idxs])  
 action\_prob\_batch = torch.FloatTensor([action\_prob\_history[idx] for idx in idxs])  
 advantage\_batch = torch.FloatTensor([discounted\_rewards[idx] for idx in idxs])  
  
 opt.zero\_grad()  
 loss = policy(d\_obs\_batch, action\_batch, action\_prob\_batch, advantage\_batch)  
 loss.backward()  
 opt.step()  
  
 if it % 5 == 0:  
 torch.save(policy.state\_dict(), 'params.ckpt')  
  
env.close()

the log:

Iteration 0, Episode 0 (3539 timesteps) - last\_action: 0, last\_action\_prob: 0.50, reward\_sum: -21.00, running\_avg: -21.00

Iteration 0, Episode 1 (3543 timesteps) - last\_action: 1, last\_action\_prob: 0.50, reward\_sum: -21.00, running\_avg: -21.00

Iteration 0, Episode 2 (3922 timesteps) - last\_action: 1, last\_action\_prob: 0.49, reward\_sum: -20.00, running\_avg: -20.99

Iteration 0, Episode 3 (4398 timesteps) - last\_action: 0, last\_action\_prob: 0.50, reward\_sum: -20.00, running\_avg: -20.98

Iteration 0, Episode 4 (4021 timesteps) - last\_action: 0, last\_action\_prob: 0.50, reward\_sum: -21.00, running\_avg: -20.98

Iteration 0, Episode 5 (3783 timesteps) - last\_action: 0, last\_action\_prob: 0.50, reward\_sum: -21.00, running\_avg: -20.98

Iteration 0, Episode 6 (3795 timesteps) - last\_action: 1, last\_action\_prob: 0.50, reward\_sum: -21.00, running\_avg: -20.98

… 10 episodes in an iteration

100000 iterations