04

January 23, 2024

[1]:
```
%pylab inline
import torch
import sys, os
import pystk
import ray
device = torch.device('cuda') if torch.cuda.is_available() else torch.
 ↪device('cpu')
print('device = ', device)
ray.init(logging_level=50)
```

```
%pylab is deprecated, use %matplotlib inline and import the required libraries.
Populating the interactive namespace from numpy and matplotlib
device =  cuda
```

[1]:
```
RayContext(dashboard_url='', python_version='3.10.13', ray_version='2.9.1',
ray_commit='cfbf98c315cfb2710c56039a3c96477d196de049', protocol_version=None)
```

[2]:
```
@ray.remote
class Rollout:
    def __init__(self, screen_width, screen_height, hd=True,␣
 ↪track='lighthouse', render=True):
        # Init supertuxkart
        if not render:
            config = pystk.GraphicsConfig.none()
        elif hd:
            config = pystk.GraphicsConfig.hd()
        else:
            config = pystk.GraphicsConfig.ld()
        config.screen_width = screen_width
        config.screen_height = screen_height
        pystk.init(config)

        self.render = render
        race_config = pystk.RaceConfig(track=track)
        self.race = pystk.Race(race_config)
        self.race.start()
```

```
    def __call__(self, agent, n_steps=200):
        torch.set_num_threads(1)
        self.race.restart()
        self.race.step()
        data = []
        track_info = pystk.Track()
        track_info.update()

        for i in range(n_steps):
            world_info = pystk.WorldState()
            world_info.update()

            # Gather world information
            kart_info = world_info.players[0].kart

            agent_data = {'track_info': track_info, 'kart_info': kart_info}
            if self.render:
                agent_data['image'] = np.array(self.race.render_data[0].image)

            # Act
            action = agent(**agent_data)
            agent_data['action'] = action

            # Take a step in the simulation
            self.race.step(action)

            # Save all the relevant data
            data.append(agent_data)
        return data

def show_video(frames, fps=30):
    import imageio
    from IPython.display import Video, display

    imageio.mimwrite('/tmp/test.mp4', frames, fps=fps, bitrate=10000000)
    display(Video('/tmp/test.mp4', width=800, height=600, embed=True))

viz_rollout = Rollout.remote(100, 75)
def show_agent(agent, n_steps=600):
    data = ray.get(viz_rollout.__call__.remote(agent, n_steps=n_steps))
    show_video([d['image'] for d in data])

rollouts = [Rollout.remote(100, 75) for i in range(10)]
def rollout_many(many_agents, **kwargs):
    ray_data = []
    for i, agent in enumerate(many_agents):
```

```
        ray_data.append( rollouts[i % len(rollouts)].__call__.remote(agent,␣
 ↪**kwargs) )
    return ray.get(ray_data)

def dummy_agent(**kwargs):
    action = pystk.Action()
    action.acceleration = 1
    return action

# Let's load a fancy auto-pilot. You'll write one yourself in your homework.
from _auto_pilot import auto_pilot

show_agent(auto_pilot)
```

```
---------------------------------------------------------------------------
ModuleNotFoundError                       Traceback (most recent call last)
Cell In[2], line 75
     72     return action
     74 # Let's load a fancy auto-pilot. You'll write one yourself in your␣
 ↪homework.
---> 75 from _auto_pilot import auto_pilot
     77 show_agent(auto_pilot)

ModuleNotFoundError: No module named '_auto_pilot'
```

[3]:
```
from torchvision.transforms import functional as TF

class ActionNet(torch.nn.Module):
    def __init__(self):
        super().__init__()
        self.network = torch.nn.Sequential(
            torch.nn.BatchNorm2d(3),
            torch.nn.Conv2d(3, 16, 5, stride=2),
            torch.nn.ReLU(),
            torch.nn.Conv2d(16, 32, 5, stride=2),
            torch.nn.ReLU(),
            torch.nn.Conv2d(32, 32, 5, stride=2),
            torch.nn.ReLU(),
            torch.nn.Conv2d(32, 32, 5, stride=2),
            torch.nn.ReLU()
        )
        self.classifier = torch.nn.Linear(32, 2)

    def forward(self, x):
        f = self.network(x)
        return self.classifier(f.mean(dim=(2,3)))
```

```
class Actor:
    def __init__(self, action_net):
        self.action_net = action_net.cpu().eval()

    def __call__(self, image, **kwargs):
        output = self.action_net(TF.to_tensor(image)[None])[0]

        action = pystk.Action()
        action.acceleration = 1
        action.steer = output[0]
        action.drift = output[1] > 0.5
        return action


def noisy_actor(actor, noise_std=5):
    def act(**kwargs):
        action = actor(**kwargs)
        action.steer += np.random.normal(0, noise_std)
        return action
    return act


action_net = ActionNet()
actor = Actor(action_net)
```

[4]:
```
%load_ext tensorboard
import tempfile
log_dir = 'log_04'
%tensorboard --logdir {log_dir} --reload_interval 1
```

```
<IPython.core.display.HTML object>
```

[5]:
```
import torch.utils.tensorboard as tb
from datetime import datetime

n_epochs = 10
batch_size = 128
n_trajectories = 10

# Create the network
action_net = ActionNet().to(device)

# Create the optimizer
optimizer = torch.optim.Adam(action_net.parameters())

# Create the loss
```

```
loss = torch.nn.MSELoss()

# Collect the data
train_data = []
for data in rollout_many([auto_pilot]*n_trajectories):
    train_data.extend(data)

# Upload to the GPU
train_images = torch.stack([torch.as_tensor(d['image']) for d in train_data]).
 ↪permute(0,3,1,2).to(device).float()/255.
train_labels = torch.stack([torch.as_tensor((d['action'].steer, d['action'].
 ↪drift)) for d in train_data]).to(device).float()

# Start training
global_step = 0
action_net.train().to(device)
logger = tb.SummaryWriter(log_dir+'/'+str(datetime.now()), flush_secs=1)

for epoch in range(n_epochs):
    for iteration in range(0, len(train_data), batch_size):
        batch_ids = torch.randint(0, len(train_data), (batch_size,),␣
 ↪device=device)
        batch_images = train_images[batch_ids]
        batch_labels = train_labels[batch_ids]
        o = action_net(batch_images)
        loss_val = loss(o, batch_labels)

        logger.add_scalar('train/loss', loss_val, global_step)
        global_step += 1

        optimizer.zero_grad()
        loss_val.backward()
        optimizer.step()
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[5], line 19
     17 # Collect the data
     18 train_data = []
---> 19 for data in rollout_many([auto_pilot]*n_trajectories):
     20     train_data.extend(data)
     22 # Upload to the GPU

NameError: name 'auto_pilot' is not defined
```

[ ]:
```
show_agent(Actor(action_net), n_steps=600)
```

[8]:
```
# Collect the data
for data in rollout_many([Actor(action_net)]*n_trajectories):
    train_data.extend(data)
# for data in rollout_many([noisy_actor(auto_pilot, noise_std=i//2) for i in␣
 ↪range(n_trajectories)]):
#     train_data.extend(data)

# Supervise using the auto-pilot actions
for d in train_data:
    d['action'] = auto_pilot(**d)

# Upload to the GPU
train_images = torch.stack([torch.as_tensor(d['image']) for d in train_data]).
 ↪permute(0,3,1,2).to(device).float()/255.
train_labels = torch.stack([torch.as_tensor((d['action'].steer, d['action'].
 ↪drift)) for d in train_data]).to(device).float()

# Start training
logger = tb.SummaryWriter(log_dir+'/'+str(datetime.now()), flush_secs=1)
global_step = 0
action_net.train().to(device)

for epoch in range(n_epochs):
    for iteration in range(0, len(train_data), batch_size):
        batch_ids = torch.randint(0, len(train_data), (batch_size,),␣
 ↪device=device)
        batch_images = train_images[batch_ids]
        batch_labels = train_labels[batch_ids]
        o = action_net(batch_images)
        loss_val = loss(o, batch_labels)

        logger.add_scalar('train/loss', loss_val, global_step)
        global_step += 1

        optimizer.zero_grad()
        loss_val.backward()
        optimizer.step()
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[8], line 9
      4 # for data in rollout_many([noisy_actor(auto_pilot, noise_std=i//2) for␣
 ↪i in range(n_trajectories)]):
      5 #     train_data.extend(data)
      6
      7 # Supervise using the auto-pilot actions
      8 for d in train_data:
```

```
----> 9     d['action'] = auto_pilot(**d)
     11 # Upload to the GPU
     12 train_images = torch.stack([torch.as_tensor(d['image']) for d in␣
 ↪train_data]).permute(0,3,1,2).to(device).float()/255.

NameError: name 'auto_pilot' is not defined
```

[ ]:
```
show_agent(Actor(action_net), n_steps=600)
```

[ ]:
```

```