cs285 hw1

Code

Atari

1. DeepMind-style Atari env:

```
def wrap_deepmind(env):
    """Make end-of-life == end-of-episode, but only reset on true game over. Done by Dee
    since it helps value estimation. 没有命了立马重启."""
    env = EpisodicLifeEnv(env)
    """Sample initial states by taking random number of no-ops on reset. No-op is assume
    env = NoopResetEnv(env, noop_max=30)
    """Return only every `skip`-th frame"""
    env = MaxAndSkipEnv(env, skip=4)
    """Take action on reset for environments that are fixed until firing."""
    if 'FIRE' in env.unwrapped.get_action_meanings():
        env = FireResetEnv(env)
    """把原始图片处理成84*84*1的大小"""
    env = ProcessFrame84(env)
    """Bin reward to {+1, 0, -1} by its sign."""
    env = ClipRewardEnv(env)
    return env
```

2. config

```
{
    'learning_starts': 50000,
    'target_update_freq': 10000,
    'replay_buffer_size': int(1e6),
    'num_timesteps': int(2e8),
    'learning_freq': 4,  # train freq, 和'frame_history_len'一致
    'grad_norm_clipping': 10,
    'frame_history_len': 4, # 合并相邻4 frames to represent state (84, 84, 4)
}
```

atari optimizer config:

```
def atari_optimizer(num_timesteps):
    lr_schedule = PiecewiseSchedule(
        [
            (0, 1e-1),
            (num_timesteps / 40, 1e-1),
            (num_timesteps / 8, 5e-2),
        ],
        outside_value=5e-2,
    )

    return OptimizerSpec(
        constructor=optim.Adam,
        optim_kwargs=dict(lr=1e-3, eps=1e-4),
        learning_rate_schedule=lambda t: lr_schedule.value(t),
    )
```

atari explore config:

```
def atari_exploration_schedule(num_timesteps):
    return PiecewiseSchedule([
        (0, 1.0),
        (1e6, 0.1),
        (num_timesteps / 8, 0.01),
    ],
                                outside_value=0.01)
```

DQN

1. replay_buffer
   每个时刻存取(obs, act, rew, next_obs, done), 有最大容量, 循环存取. frame存的是uint8,节省内存.
2. critic loss:

   ```
   loss = nn.SmoothL1Loss()
   ```

Pytorch

1. optimizer中定的是base_lr, learning_rate_scheduler中的学习率是调整的比率, 它和base_lr相乘得到最终的lr.
2. LambdaLR, 传入lambda函数来调整lr

```
# Assuming optimizer has two groups.
lambda1 = lambda epoch: epoch // 30
lambda2 = lambda epoch: 0.95 ** epoch
scheduler = LambdaLR(optimizer, lr_lambda=[lambda1, lambda2])
for epoch in range(100):
    train(...)
    validate(...)
    scheduler.step()
```

3. 增加接口的通用性, 可以传入一些函数类型的参数(lambda函数, 或者回调函数)
4. 把一些常用的函数, 常量统一放入到一个module文件中, 然后用这个module来访问这些函数, 常量. 如pytorch_utils.py中:

```
Activation = Union[str, nn.Module]


_str_to_activation = {
    'relu': nn.ReLU(),
    'tanh': nn.Tanh(),
    'leaky_relu': nn.LeakyReLU(),
    'sigmoid': nn.Sigmoid(),
    'selu': nn.SELU(),
    'softplus': nn.Softplus(),
    'identity': nn.Identity(),
}


def build_mlp(
        input_size: int,
        output_size: int,
        n_layers: int,
        size: int,
        activation: Activation = 'tanh',
        output_activation: Activation = 'identity',
):
```

```
device = None


def init_gpu(use_gpu=True, gpu_id=0):
    global device
    if torch.cuda.is_available() and use_gpu:
        device = torch.device("cuda:" + str(gpu_id))
        print("Using GPU id {}".format(gpu_id))
    else:
        device = torch.device("cpu")
        print("GPU not detected. Defaulting to CPU.")
```

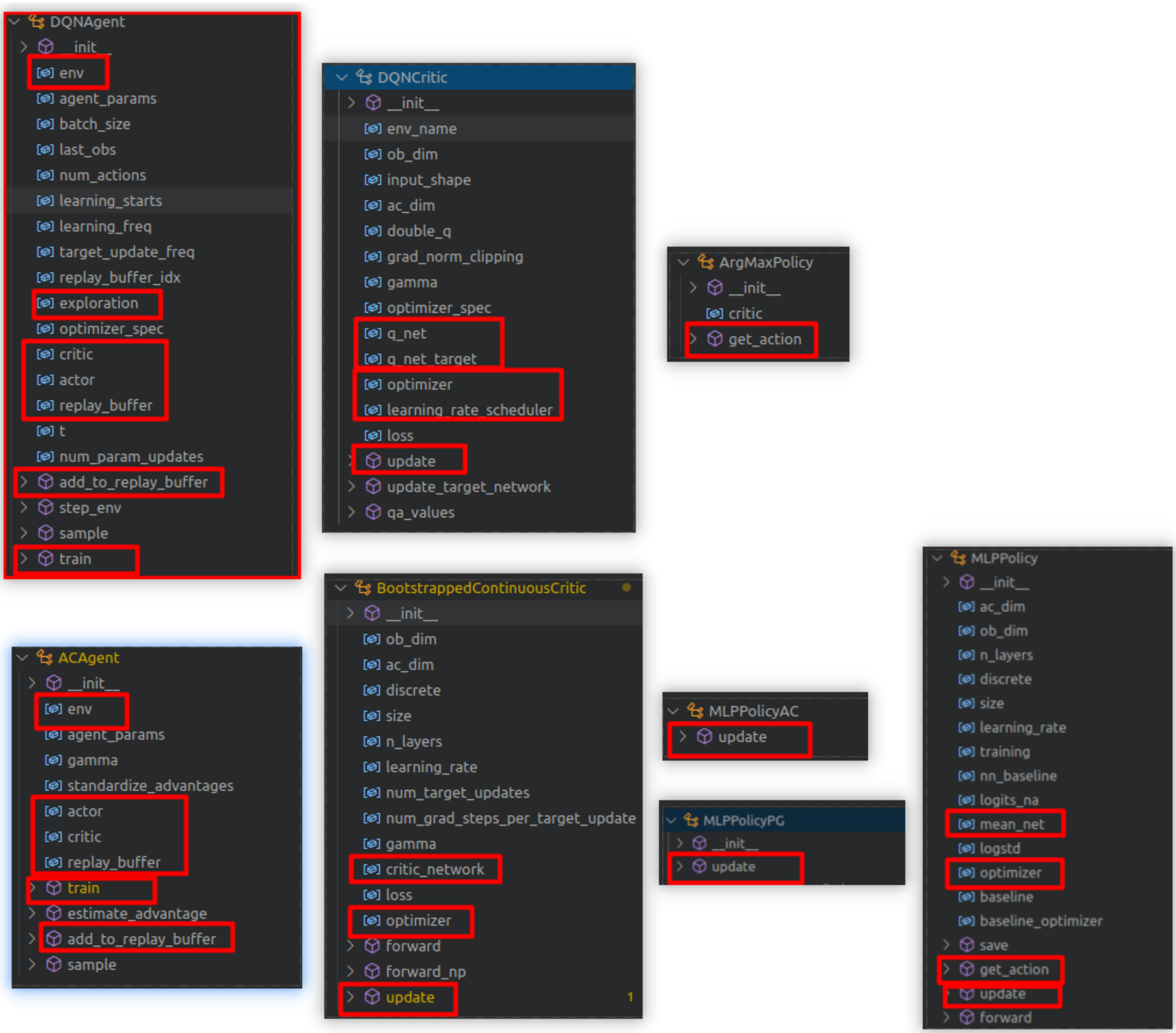
这些方法和变量在不同的项目中都能用到, 而且不用改变.

5. 运行时才sampling

```
idxes = sample_n_unique(lambda: random.randint(0, 10), batch_size)
```

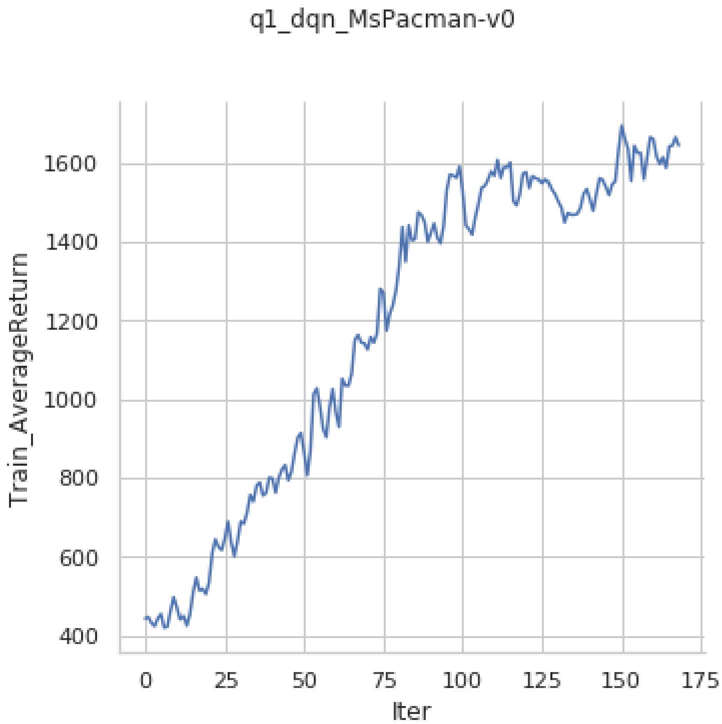
```
def sample_n_unique(sampling_f, n):
    """Helper function. Given a function `sampling_f` that returns
    comparable objects, sample n such unique objects.
    """
    res = []
    while len(res) < n:
        candidate = sampling_f()
        if candidate not in res:
            res.append(candidate)
    return res
```

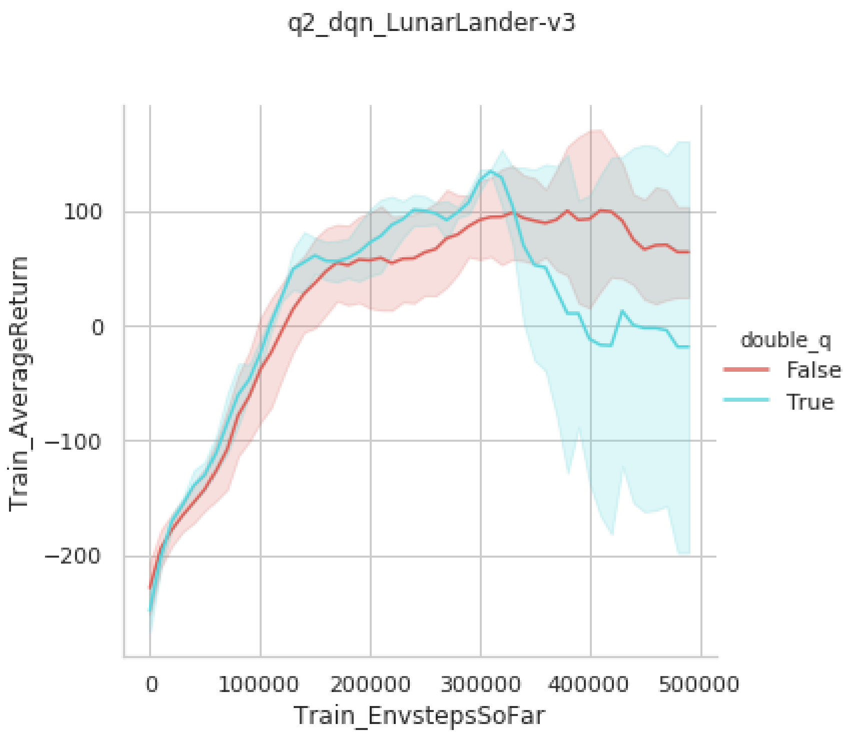
RL framework design


DQNAgent
__init__
env
agent_params
batch_size
last_obs
num_actions
learning_starts
learning_freq
target_update_freq
replay_buffer_idx
exploration
optimizer_spec
critic
actor
replay_buffer
t
num_param_updates
add_to_replay_buffer
step_env
sample
train
DQNCritic
__init__
env_name
ob_dim
input_shape
ac_dim
double_q
grad_norm_clipping
gamma
optimizer_spec
q_net
q_net_target
optimizer
learning_rate_scheduler
loss
update
update_target_network
qa_values
ArgMaxPolicy
__init__
critic
get_action
ACAgent
__init__
env
agent_params
gamma
standardize_advantages
actor
critic
replay_buffer
train
estimate_advantage
add_to_replay_buffer
sample
BootstrappedContinuousCritic
__init__
ob_dim
ac_dim
discrete
size
n_layers
learning_rate
num_target_updates
num_grad_steps_per_target_update
gamma
critic_network
loss
optimizer
forward
forward_np
update
1
MLPPolicyAC
update
MLPPolicyPG
__init__
update
MLPPolicy
__init__
ac_dim
ob_dim
n_layers
discrete
size
learning_rate
training
nn_baseline
logits_na
mean_net
logstd
optimizer
baseline
baseline_optimizer
save
get_action
update
forward

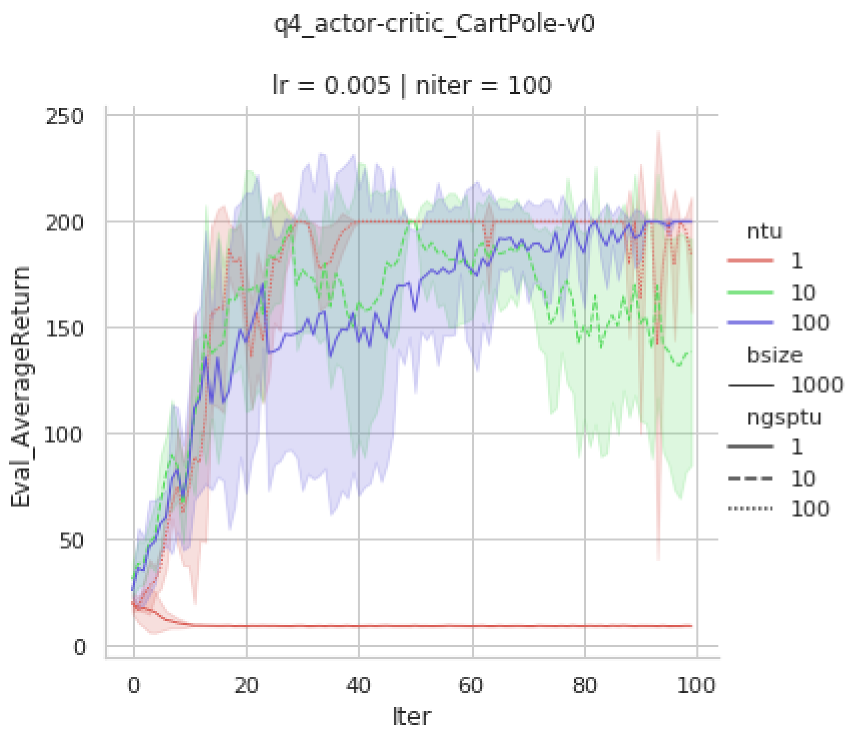
Exps

DQN

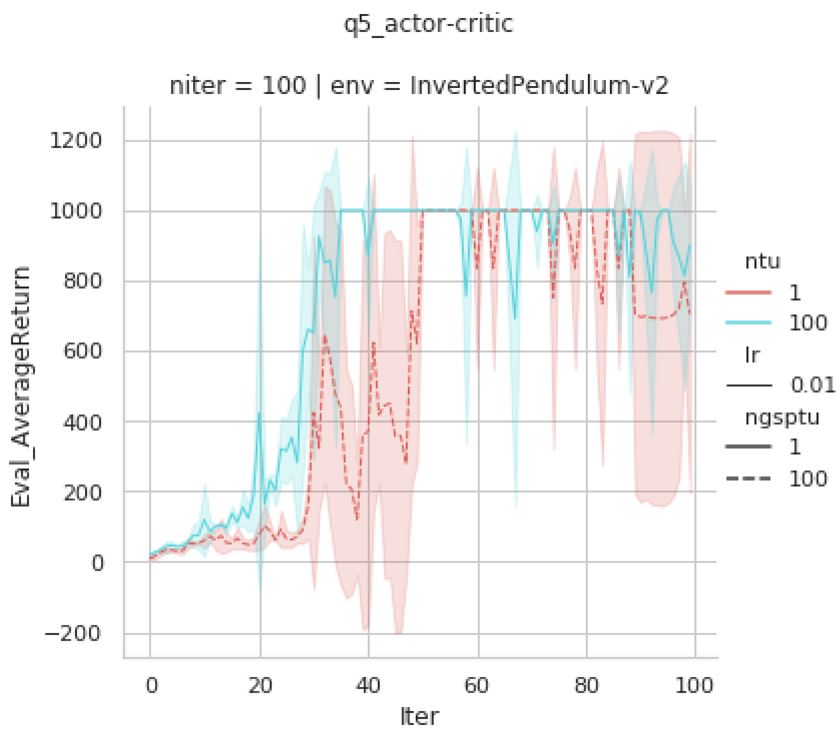

q1_dqn_MsPacman-v0
Train_AverageReturn
1600
1400
1200
1000
800
600
400
0
25
50
75
100
125
150
175
Iter


q2_dqn_LunarLander-v3
Train_AverageReturn
100
0
−100
−200
0
100000
200000
300000
400000
500000
Train_EnvstepsSoFar
double_q
False
True

AC


q4_actor-critic_CartPole-v0
lr = 0.005 | niter = 100
Eval_AverageReturn
Iter
ntu
1
10
100
bsize
1000
ngsptu
1
10
100


q5_actor-critic
niter = 150 | env = HalfCheetah-v2
Eval_AverageReturn
Iter
ntu
1
100
lr
0.02
ngsptu
1
100




```
for _ in range(self.num_target_updates):
    t_n = reward_n + self.gamma * self.forward_np(next_ob_no) * (1.0 - terminal_n)
    for _ in range(self.num_grad_steps_per_target_update):
        ob_no_tensor = ptu.from_numpy(ob_no)
        t_n_tensor = ptu.from_numpy(t_n)
        loss = self.loss(self.forward(ob_no_tensor), t_n_tensor)
        self.optimizer.zero_grad()
        loss.backward()
        self.optimizer.step()
```

> **ntu**: num_target_updates
>
> **ngsptu**: num_grad_steps_per_target_update

可以看出

1. ntu ~ ngsptu, 模型训练效果很差.
2. ntu >> ngsptu, 模型需要花更多时间收敛, 最终效果和 ngsptu >> ntu 相当
3. ngsptu >> ntu, 模型收敛快,但是收敛后容易震荡(因为保持target不变的情况下, 进行了更多次的梯度下降)