Командный тур

5.1. Задачи

Введение

Reinforcement Learning (RL, Обучение с подкреплением) - это один из типов Машинного обучения (МО), предназначенный для создания самообучающихся алгоритмов. Он представляет из себя обучение путем проб и ошибок. Это один из самых малоизученных типов МО, но в то же время и самых перспективных, поскольку у данных задачи нет заблаговременно подготовленных ответов, как в Обучении с учителем. Именно отсутствие необходимости в заранее известных ответах является неоспоримым преимуществом данного метода перед другими типами машинного обучения. Это позволяет решать задачи, которые ранее не могли быть решены известными методами.

В общем случае RL сводится к тому, чтобы без заранее известных ответов, как в Supervised Learning, вычислить ошибку решения алгоритма и далее, используя ее, провести коррекцию весов классическими методами, например, методом обратного распространения ошибки.

Для выполнения работы и обучения алгоритмов был выбран фреймворк OpenAI Gym. Его достоинство в том, что он содержит в себе несколько десятков сред, отлично подходящих для обучения RL алгоритмов и не имеет известных аналогов. Все среды являются классическими играми разной сложности, поэтому задачей алгоритмов становится научиться успешно играть в игры и достигать в них удовлетворительных результатов.

Формализация понятия игры, формализация нескольких классических игр

Игра представляет собой среду, в которой действует Агент. В используемой библиотеке OpenAI Gym все среды представлены в следующем виде.

Каждая из представленных игр имеет:

a. Observations (наблюдения, предоставляемые агенту) b. Actions (список его возможных действий в данной среде).

Observations представлены в виде векторов или многомерных массивов. Actions представляет собой вектор пронумерованных действий. Агент должен, исходя из своих наблюдений, выбрать наиболее подходящее действие, которое максимизирует его награду. Важно отметить: Агент не знает семантики выбранного действия, он может

со временем обучиться тому, что, находясь в определенном состоянии (Observations) при выборе определенного действия, он получит награду (положительную или отрицательную).

Задача "Пространственная ориентация квадрокоптера"

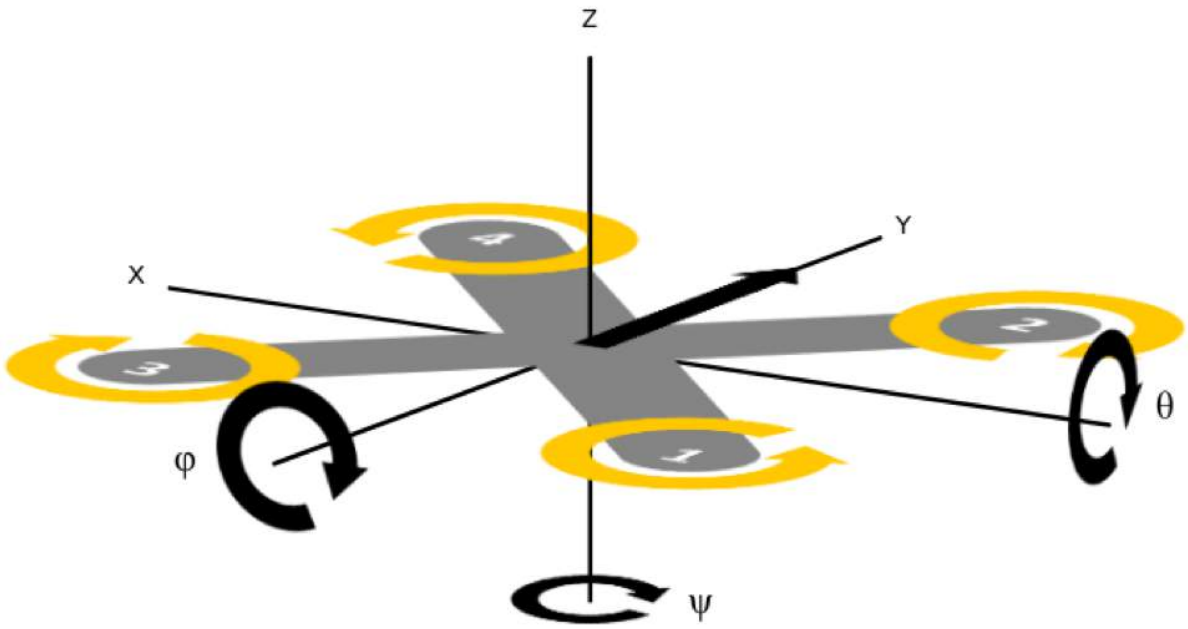
GymFC — это среда OpenAI Gym, специально разработанная для разработки интеллектуальных систем управления полетом с использованием обучения с подкреплением. Сама среда выполнена с использованием симулятора физики. В реализации участники должны научиться управлять пространственной ориентацией квадрокоптера. Квадрокоптер представляет собой летательный аппарат с шестью степенями свободы, тремя вращательными и тремя поступательными.

Задача предусматривает 2 режима работы:

- эпизодическая среда (AttFC_GyroErr-MotorVel_M4_Ep-v0) — В начале каждого эпизода квадрокоптер находится в покое. Выбирается случайная целевая угловая скорость, которую необходимо достигнуть агенту в течение 1 секунды, изначально находясь в начальном состоянии/покое (отсутствие угловых скоростей).
- полная среда (AttFC_GyroErr-MotorVel_M4_Con-v0) — по существу, такая же, как и эпизодический вариант, однако она работает в течение 60 секунд и непрерывно произвольно меняет целевые угловые скорости случайным образом в интервале времени между [0.1, 1] секундами.

Список возможных действий(Actions) представляет собой вектор размерности 4 и соответствует четырем входами управления (по одному на каждый двигатель), соответственно есть возможность напрямую управлять мощностью вращения каждого из 4 двигателей.

Observations представляет собой вектор размерности 7. Где первые 4 элемента - мощности вращения каждого из 4 двигателей. А оставшиеся 3 элемента вектора - углы Эйлера(крен, тангаж и рыскание), которые описывают поворот объекта в евклидовом пространстве относительно исходной оси.



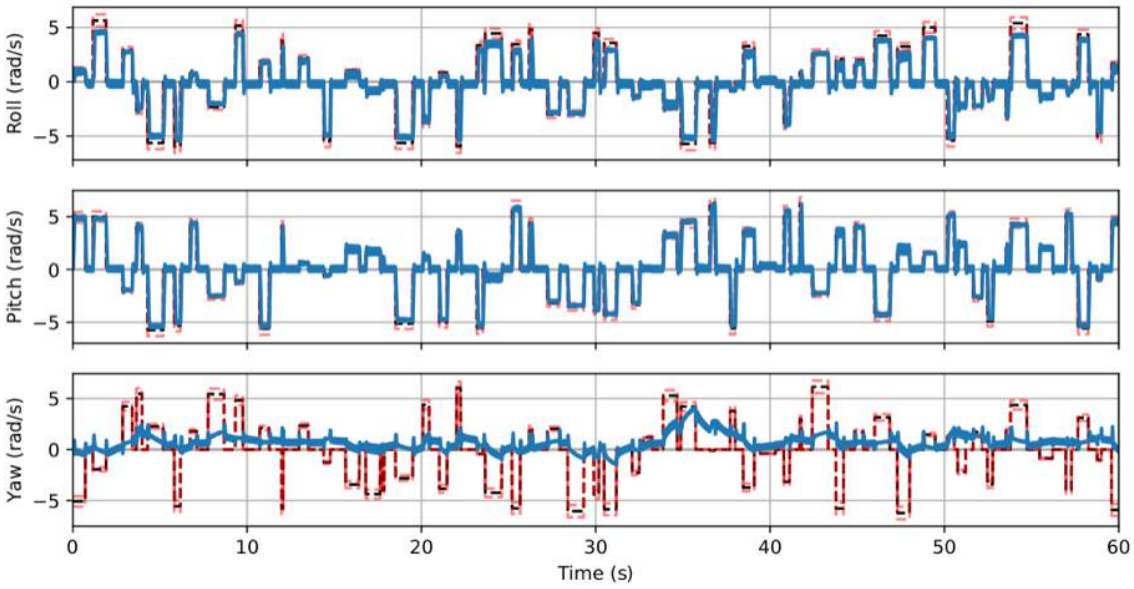
Немного о наградах и ошибках среды

Ошибка/награда нормализована между $[-1, 0]$, представляющей, насколько близка угловая скорость к цели, вычисленной с помощью $-clip(SUM(|\Omega^* - \Omega|)/3\Omega_{max})$, где функция *clip* ограничивает результат в интервале $[-1, 0]$ и Ω_{max} — исходная ошибка от момента, когда заданная угловая скорость установлена.

Оценка решений

Оценка решений осуществляется на основе презентации и демонстрации агента при необходимости. Тестирование агента будет проходить в режиме полной симуляции. Для всех участников будет сгенерировано 3 начальных значения для генератора случайных чисел (3 одинаковых для всех участников). На данных значениях генератора случайных чисел будет запущено 3 симуляции. Среднее значение REWARD от среды в 3 запусках будет финальным результатом для решения.

Для дополнительной интерпретируемости результатов участники могут использовать функцию “plot_step_response” из baseline решения, которая строит графики по 3 угловым скоростям.



Пример графиков. Из него видно что по двум угловым скоростям достигнут неплохой результат, а проседает третья угловая скорость.

Оформление решения

Участники должны предоставить одно решение, которое необходимо реализовать по аналогии с baseline решением.

```
def main(env_id, seed):
    agent = Agent()
    ob = env.reset()
    actuals = []
    desireds = []
    while True:
        desired = env.omega_target
        actual = env.omega_actual
        actuals.append(actual)
        desireds.append(desired)
        action = agent.predict(ob)
        ob, reward, done, _ = env.step(action)
        if done:
            break
```

Ссылка на Среду с Базовым решением (докер файл): `http://185.127.227.39:788/nti.tar.xz`

Настройка докера под Linux

1. Опционально поставить Anaconda на основную систему, отвечаем там yes на все кроме Visual Studio Code.

   wget `https://repo.anaconda.com/archive/Anaconda3-2018.12-Linux-x86_64.sh`

   sudo chmod a+rwx Anaconda3-2018.12-Linux-x86_64.sh

   ./Anaconda3-2018.12-Linux-x86_64.sh

2. Скачать докер
   wget `http://185.127.227.39:788/nti.tar.xz`
   tar -xf nti.tar.xz
   cd nti
   Все примеры через sudo, но можете завести пользователя и дать ему соответствующие права.
3. Собрать докер образ
   sudo snap install docker
   sudo docker-compose -f docker-compose.yml build
4. Запустить контейнер
   sudo docker-compose -f docker-compose.yml up -d
5. Если вы на нашей виртуалке загрузить совместимую верисб tensorflow
   sudo docker exec -it nti_rl_1 bash -c "pip3.6 install tensorflow==1.5"
6. Проверить работоспособность бейзлайна обучения
   sudo docker exec -it nti_rl_1 bash -c "cd /code/nti/gymfc/examples/controllers/; python3.6 run_baseline.py –env=AttFC_GyroErr-MotorVel_M4_Ep-v0 –num-timesteps=100"
7. Проверить работоспособность бейзлайна тестирования
   sudo docker exec -it nti_rl_1 bash -c "cd /code/nti/gymfc/examples/controllers/; python3.6 run_baseline.py –env=AttFC_GyroErr-MotorVel_M4_Ep-v0 –play"
   docker exec -it nti_rl_1 bash -c "

Полезные ссылки про докер

- про докер в целом `https://habr.com/ru/post/309556/`
- как настроить Jupiter Notebook из-под докера `https://proglib.io/p/docker-compose/` и `https://www.dataquest.io/blog/docker-data-science`
- как настроить все из-под докера `https://dev-ops-notes.com/docker/howto-run-jupiter-keras-tensorflow-pandas-sklearn-and-matplotlib-docker-container/`

Запуск базового решение (файл run_ baseline.py)

```
1   import os
2   from baselines.common import tf_util as U
3   from baselines import logger
4   import gymfc
5   import argparse
6   import numpy as np
7   import matplotlib
8
9   matplotlib.use("Agg")
10  import matplotlib.pyplot as plt
11  import gym
12  import math
13
14  def train(num_timesteps, seed, model_path=None, env_id=None):
15      from baselines.ppo1 import mlp_policy, pposgd_simple
```

```
        U.make_session(num_cpu=1).__enter__()

        def policy_fn(name, ob_space, ac_space):
            return mlp_policy.MlpPolicy(name=name, ob_space=ob_space, ac_space=ac_space,
                                        hid_size=64, num_hid_layers=2)

        env = gym.make(env_id)

        # parameters below were the best found in a simple random search
        # these are good enough to make humanoid walk, but whether those are
        # an absolute best or not is not certain
        env = RewScale(env, 0.1)
        pi = pposgd_simple.learn(env, policy_fn,
                                 max_timesteps=num_timesteps,
                                 timesteps_per_actorbatch=2048,
                                 clip_param=0.2, entcoeff=0.0,
                                 optim_epochs=10,
                                 optim_stepsize=3e-4,
                                 optim_batchsize=64,
                                 gamma=0.99,
                                 lam=0.95,
                                 schedule="linear",
                                 )
        env.close()
        if model_path:
            U.save_state(model_path)

        return pi

        class RewScale(gym.RewardWrapper):
        def __init__(self, env, scale):
            gym.RewardWrapper.__init__(self, env)
            self.scale = scale

        def reward(self, r):
            return r * self.scale

    def plot_step_response(desired, actual,
                           end=1., title=None,
                           step_size=0.001, threshold_percent=0.1):
        """
            Args:
                threshold (float): Percent of the start error
        """
        # actual = actual[:,:end,:]
        end_time = len(desired) * step_size
        t = np.arange(0, end_time, step_size)

        # desired = desired[:end]
        threshold = threshold_percent * desired

        plot_min = -math.radians(350)
        plot_max = math.radians(350)

        subplot_index = 3
        num_subplots = 3

        f, ax = plt.subplots(num_subplots, sharex=True, sharey=False)
        f.set_size_inches(10, 5)
        if title:
```

```
76              plt.suptitle(title)
77          ax[0].set_xlim([0, end_time])
78          res_linewidth = 2
79          linestyles = ["c", "m", "b", "g"]
80          reflinestyle = "k--"
81          error_linestyle = "r--"
82
83          # Always
84          ax[0].set_ylabel("Roll (rad/s)")
85          ax[1].set_ylabel("Pitch (rad/s)")
86          ax[2].set_ylabel("Yaw (rad/s)")
87
88          ax[-1].set_xlabel("Time (s)")
89
90          """ ROLL """
91          # Highlight the starting x axis
92          ax[0].axhline(0, color="#AAAAAA")
93          ax[0].plot(t, desired[:, 0], reflinestyle)
94          ax[0].plot(t, desired[:, 0] - threshold[:, 0], error_linestyle, alpha=0.5)
95          ax[0].plot(t, desired[:, 0] + threshold[:, 0], error_linestyle, alpha=0.5)
96
97          r = actual[:, 0]
98          ax[0].plot(t[:len(r)], r, linewidth=res_linewidth)
99
100         ax[0].grid(True)
101
102         """ PITCH """
103         ax[1].axhline(0, color="#AAAAAA")
104         ax[1].plot(t, desired[:, 1], reflinestyle)
105         ax[1].plot(t, desired[:, 1] - threshold[:, 1], error_linestyle, alpha=0.5)
106         ax[1].plot(t, desired[:, 1] + threshold[:, 1], error_linestyle, alpha=0.5)
107         p = actual[:, 1]
108         ax[1].plot(t[:len(p)], p, linewidth=res_linewidth)
109         ax[1].grid(True)
110
111         """ YAW """
112         ax[2].axhline(0, color="#AAAAAA")
113         ax[2].plot(t, desired[:, 2], reflinestyle)
114         ax[2].plot(t, desired[:, 2] - threshold[:, 2], error_linestyle, alpha=0.5)
115         ax[2].plot(t, desired[:, 2] + threshold[:, 2], error_linestyle, alpha=0.5)
116         y = actual[:, 2]
117         ax[2].plot(t[:len(y)], y, linewidth=res_linewidth)
118         ax[2].grid(True)
119
120         plt.savefig("gymfc-ppo-step-response.pdf")
121
122     def main():
123         parser = argparse.ArgumentParser()
124         logger.configure()
125         parser.add_argument("--env", type=str, help="The Gym environement ID",
126                             default="AttFC_GyroErr-MotorVel_M4_Con-v0")
127         parser.add_argument("--seed", help="RNG seed", type=int, default=0)
128         parser.add_argument("--model-path", default=os.path.join("/root/code/nti/
129                                                 gymfc/humanoid_policy", "hum"))
130         parser.add_argument("--play", action="store_true", default=False)
131         parser.add_argument("--num-timesteps", type=int, default=2*1e6)
132
133         current_dir = os.path.dirname(__file__)
134             config_path = os.path.join(current_dir,
135                                        "../configs/iris.config")
```

```
            print("Loading config from ", config_path)
            os.environ["GYMFC_CONFIG"] = config_path
            args = parser.parse_args()

            if not args.play:
                # train the model
                train(num_timesteps=args.num_timesteps, seed=args.seed,
                      model_path=args.model_path, env_id=args.env)
            else:
                print(" Making env=", args.env)
                # construct the model object, load pre-trained model and render
                pi = train(num_timesteps=1, seed=args.seed, env_id=args.env)
                U.load_state(args.model_path)

                env = gym.make(args.env)
                #env.render()
                ob = env.reset()
                actuals = []
                desireds = []
                while True:
                    desired = env.omega_target
                    actual = env.omega_actual
                    actuals.append(actual)
                    desireds.append(desired)
                    print("sp=", desired, " rate=", actual)
                    action = pi.act(stochastic=False, ob=ob)[0]
                    ob, _, done, _ = env.step(action)
                    if done:
                        break
                print(np.array(desireds))
                print(np.array(actuals))
                plot_step_response(np.array(desireds), np.array(actuals))

    if __name__ == "__main__":
        main()
```

Основа базового решения (pposgd_simple), код с обучением политикам модели

```
    from baselines.common import Dataset, explained_variance, fmt_row, zipsame
    from baselines import logger
    import baselines.common.tf_util as U
    import tensorflow as tf, numpy as np
    import time
    from baselines.common.mpi_adam import MpiAdam
    from baselines.common.mpi_moments import mpi_moments
    from mpi4py import MPI
    from collections import deque

    def traj_segment_generator(pi, env, horizon, stochastic):
        t = 0
        ac = env.action_space.sample() # not used, just so we have the datatype
        new = True # marks if we're on first timestep of an episode
        ob = env.reset()

        cur_ep_ret = 0 # return in current episode
        cur_ep_len = 0 # len of current episode
        ep_rets = [] # returns of completed episodes in this segment
        ep_lens = [] # lengths of ...
```

```
21
22          # Initialize history arrays
23          obs = np.array([ob for _ in range(horizon)])
24          rews = np.zeros(horizon, 'float32')
25          vpreds = np.zeros(horizon, 'float32')
26          news = np.zeros(horizon, 'int32')
27          acs = np.array([ac for _ in range(horizon)])
28          prevacs = acs.copy()
29
30          while True:
31              prevac = ac
32              ac, vpred = pi.act(stochastic, ob)
33              # Slight weirdness here because we need value function at time T
34              # before returning segment [0, T-1] so we get the correct
35              # terminal value
36              if t > 0 and t % horizon == 0:
37                  yield {"ob" : obs, "rew" : rews, "vpred" : vpreds, "new" : news,
38                          "ac" : acs, "prevac" : prevacs, "nextvpred": vpred * (1 - new),
39                          "ep_rets" : ep_rets, "ep_lens" : ep_lens}
40                  # Be careful!!! if you change the downstream algorithm to aggregate
41                  # several of these batches, then be sure to do a deepcopy
42                  ep_rets = []
43                  ep_lens = []
44              i = t % horizon
45              obs[i] = ob
46              vpreds[i] = vpred
47              news[i] = new
48              acs[i] = ac
49              prevacs[i] = prevac
50
51              ob, rew, new, _ = env.step(ac)
52              rews[i] = rew
53
54              cur_ep_ret += rew
55              cur_ep_len += 1
56              if new:
57                  ep_rets.append(cur_ep_ret)
58                  ep_lens.append(cur_ep_len)
59                  cur_ep_ret = 0
60                  cur_ep_len = 0
61                  ob = env.reset()
62              t += 1
63
64      def add_vtarg_and_adv(seg, gamma, lam):
65          """
66          Compute target value using TD(lambda) estimator, and advantage with GAE(lambda)
67          """
68          new = np.append(seg["new"], 0) # last element is only used for last vtarg,
69          # but we already zeroed it if last new = 1
70          vpred = np.append(seg["vpred"], seg["nextvpred"])
71          T = len(seg["rew"])
72          seg["adv"] = gaelam = np.empty(T, 'float32')
73          rew = seg["rew"]
74          lastgaelam = 0
75          for t in reversed(range(T)):
76              nonterminal = 1-new[t+1]
77              delta = rew[t] + gamma * vpred[t+1] * nonterminal - vpred[t]
78              gaelam[t] = lastgaelam = delta + gamma * lam * nonterminal * lastgaelam
79          seg["tdlamret"] = seg["adv"] + seg["vpred"]
80
```

```
81      def learn(env, policy_fn, *,
82              timesteps_per_actorbatch, # timesteps per actor per update
83              clip_param, entcoeff, # clipping parameter epsilon, entropy coeff
84              optim_epochs, optim_stepsize, optim_batchsize,# optimization hypers
85              gamma, lam, # advantage estimation
86              max_timesteps=0, max_episodes=0, max_iters=0, max_seconds=0,  # time constraint
87              callback=None, # you can do anything in the callback,
88              #                since it takes locals(), globals()
89              adam_epsilon=1e-5,
90              schedule='constant' # annealing for stepsize parameters (epsilon and adam)
91              ):
92          # Setup losses and stuff
93          # ----------------------------------------
94          ob_space = env.observation_space
95          ac_space = env.action_space
96          pi = policy_fn("pi", ob_space, ac_space) # Construct network for new policy
97          oldpi = policy_fn("oldpi", ob_space, ac_space) # Network for old policy
98          atarg = tf.placeholder(dtype=tf.float32, shape=[None]) # Target advantage function
99          #                                                       (if applicable)
100         ret = tf.placeholder(dtype=tf.float32, shape=[None]) # Empirical return
101
102         lrmult = tf.placeholder(name='lrmult', dtype=tf.float32, shape=[]) # learning rate
103         #                                                   multiplier, updated with schedule
104
105         ob = U.get_placeholder_cached(name="ob")
106         ac = pi.pdtype.sample_placeholder([None])
107
108         kloldnew = oldpi.pd.kl(pi.pd)
109         ent = pi.pd.entropy()
110         meankl = tf.reduce_mean(kloldnew)
111         meanent = tf.reduce_mean(ent)
112         pol_entpen = (-entcoeff) * meanent
113
114         ratio = tf.exp(pi.pd.logp(ac) - oldpi.pd.logp(ac)) # pnew / pold
115         surr1 = ratio * atarg # surrogate from conservative policy iteration
116         surr2 = tf.clip_by_value(ratio, 1.0 - clip_param, 1.0 + clip_param) * atarg #
117         pol_surr = - tf.reduce_mean(tf.minimum(surr1, surr2)) # PPO's pessimistic
118         #                                                      surrogate (L^CLIP)
119         vf_loss = tf.reduce_mean(tf.square(pi.vpred - ret))
120         total_loss = pol_surr + pol_entpen + vf_loss
121         losses = [pol_surr, pol_entpen, vf_loss, meankl, meanent]
122         loss_names = ["pol_surr", "pol_entpen", "vf_loss", "kl", "ent"]
123
124         var_list = pi.get_trainable_variables()
125         lossandgrad = U.function([ob, ac, atarg, ret, lrmult], losses +
126                                  [U.flatgrad(total_loss, var_list)])
127         adam = MpiAdam(var_list, epsilon=adam_epsilon)
128
129         assign_old_eq_new = U.function([],[], updates=[tf.assign(oldv, newv)
130             for (oldv, newv) in zipsame(oldpi.get_variables(), pi.get_variables())])
131         compute_losses = U.function([ob, ac, atarg, ret, lrmult], losses)
132
133         U.initialize()
134         adam.sync()
135
136         # Prepare for rollouts
137         # ----------------------------------------
138         seg_gen = traj_segment_generator(pi, env, t
139                                          imesteps_per_actorbatch, stochastic=True)
140
```

```
141         episodes_so_far = 0
142         timesteps_so_far = 0
143         iters_so_far = 0
144         tstart = time.time()
145         lenbuffer = deque(maxlen=100) # rolling buffer for episode lengths
146 
147         rewbuffer = deque(maxlen=100) # rolling buffer for episode rewards
148 
149         assert sum([max_iters>0, max_timesteps>0, max_episodes>0, max_seconds>0])==1,
150                 "Only one time constraint permitted"
151 
152         while True:
153             if callback: callback(locals(), globals())
154             if max_timesteps and timesteps_so_far >= max_timesteps:
155                 break
156             elif max_episodes and episodes_so_far >= max_episodes:
157                 break
158             elif max_iters and iters_so_far >= max_iters:
159                 break
160             elif max_seconds and time.time() - tstart >= max_seconds:
161                 break
162 
163             if schedule == 'constant':
164                 cur_lrmult = 1.0
165             elif schedule == 'linear':
166                 cur_lrmult =  max(1.0 - float(timesteps_so_far) / max_timesteps, 0)
167             else:
168                 raise NotImplementedError
169 
170             logger.log("********** Iteration %i ************"%iters_so_far)
171 
172             seg = seg_gen.__next__()
173             add_vtarg_and_adv(seg, gamma, lam)
174 
175             # ob, ac, atarg, ret, td1ret = map(np.concatenate,
176             #                                  (obs, acs, atargs, rets, td1rets))
177             ob, ac, atarg, tdlamret = seg["ob"], seg["ac"], seg["adv"], seg["tdlamret"]
178             vpredbefore = seg["vpred"] # predicted value function before udpate
179             atarg = (atarg - atarg.mean()) / atarg.std() # standardized advantage
180             #                                              function estimate
181             d = Dataset(dict(ob=ob, ac=ac, atarg=atarg, vtarg=tdlamret),
182                                                     shuffle=not pi.recurrent)
183             optim_batchsize = optim_batchsize or ob.shape[0]
184 
185             if hasattr(pi, "ob_rms"): pi.ob_rms.update(ob) # update running
186                                                            # mean/std for policy
187 
188             assign_old_eq_new() # set old parameter values to new parameter values
189             logger.log("Optimizing...")
190             logger.log(fmt_row(13, loss_names))
191             # Here we do a bunch of optimization epochs over the data
192             for _ in range(optim_epochs):
193                 losses = [] # list of tuples, each of which gives the loss for a minibatch
194                 for batch in d.iterate_once(optim_batchsize):
195                     *newlosses, g = lossandgrad(batch["ob"], batch["ac"], batch["atarg"],
196                                                 batch["vtarg"], cur_lrmult)
197                     adam.update(g, optim_stepsize * cur_lrmult)
198                     losses.append(newlosses)
199                 logger.log(fmt_row(13, np.mean(losses, axis=0)))
200 
```

```
            logger.log("Evaluating losses...")
            losses = []
            for batch in d.iterate_once(optim_batchsize):
                newlosses = compute_losses(batch["ob"], batch["ac"], batch["atarg"],
                                           batch["vtarg"], cur_lrmult)
                losses.append(newlosses)
            meanlosses,_,_ = mpi_moments(losses, axis=0)
            logger.log(fmt_row(13, meanlosses))
            for (lossval, name) in zipsame(meanlosses, loss_names):
                logger.record_tabular("loss_"+name, lossval)
            logger.record_tabular("ev_tdlam_before",
                                  explained_variance(vpredbefore, tdlamret))
            lrlocal = (seg["ep_lens"], seg["ep_rets"]) # local values
            listoflrpairs = MPI.COMM_WORLD.allgather(lrlocal) # list of tuples
            lens, rews = map(flatten_lists, zip(*listoflrpairs))
            lenbuffer.extend(lens)
            rewbuffer.extend(rews)
            logger.record_tabular("EpLenMean", np.mean(lenbuffer))
            logger.record_tabular("EpRewMean", np.mean(rewbuffer))
            logger.record_tabular("EpThisIter", len(lens))
            episodes_so_far += len(lens)
            timesteps_so_far += sum(lens)
            iters_so_far += 1
            logger.record_tabular("EpisodesSoFar", episodes_so_far)
            logger.record_tabular("TimestepsSoFar", timesteps_so_far)
            logger.record_tabular("TimeElapsed", time.time() - tstart)
            if MPI.COMM_WORLD.Get_rank()==0:
                logger.dump_tabular()

        return pi

    def flatten_lists(listoflists):
        return [el for list_ in listoflists for el in list_]
```