Лабораторная работа №9.

Алгоритм Trust Region Policy Optimization (TRPO)

В этой работе мы напишем код оптимизации политики одного доверенного региона. Как обычно, он содержит несколько разных частей, которые мы собираемся воспроизвести.

1. Предварительные действия

```
import gym
from time import sleep

from IPython.display import clear_output

import numpy as np
import tensorflow.compat.v1 as tf
tf.disable_v2_behavior()

import gym

env = gym.make("Acrobot-v1")
env.reset()
observation_shape = env.observation_space.shape
n_actions = env.action_space.n
print("Observation Space", env.observation_space)
print("Action Space", env.action_space)
```

Вывод:

Observation Space Box([-1. -1. -1. -1. -12.566371 -28.274334], [1. 1. 1. 1. 12.566371 28.274334], (6,), float32)

Action Space Discrete(3)

```
import matplotlib.pyplot as plt
plt.imshow(env.render('rgb_array'))
```

2. Определение сети

При всей своей сложности TRPO по своей сути является еще одним методом градиента политики. По сути, это означает, что мы на самом деле тренируем стохастическую политику. Будет использована нейронная сеть.

```
# входной тензор
observations_ph = tf.placeholder(
    shape=(None, observation_shape[0]), dtype=tf.float32)
# Совершенные действия
actions_ph = tf.placeholder(shape=(None,), dtype=tf.int32)
# "G = r + gamma*r' + gamma^2*r'' + ..."
cummulative_returns_ph = tf.placeholder(shape=(None,), dtype=tf.float32)
# Вероятность действий из предыдущей итерации
old_probs_ph = tf.placeholder(shape=(None, n_actions), dtype=tf.float32)

all_inputs = [observations_ph, actions_ph,
              cummulative_returns_ph, old_probs_ph]
```

```
def denselayer(name, x, out_dim, nonlinearity=None):
    with tf.variable_scope(name):
        if nonlinearity is None:
            nonlinearity = tf.identity
```

```
        x_shape = x.get_shape().as_list()

        w = tf.get_variable('w', shape=[x_shape[1], out_dim])
        b = tf.get_variable(
            'b', shape=[out_dim], initializer=tf.constant_initializer(0))
        o = nonlinearity(tf.matmul(x, w) + b)

        return o
```

Требуется дополнить код. Что должен выполнять код в каждой позиции <ВАШ КОД> указано в примечаниях к коду.

```
sess = tf.InteractiveSession()

nn = observations_ph

<ВАШ КОД: ИНС>

policy_out = <ВАШ КОД: слой, который прогнозирует дейтсвия log-probabilities>

probs_out = tf.exp(policy_out)

weights = tf.trainable_variables()
sess.run(tf.global_variables_initializer())
```

3. Действия и развертывания

В этом разделе мы определим функции $a \sim \pi_\theta(a|s)$, которые выполняют действия и развертывания $< s_0, a_0, s_1, a_1, s_2, a_2, \ldots s_n, a_n >$.

```
# функция

def act(obs, sample=True):
    """
    Выборка действия из распространения политики (образец = True) или
выполнение наиболее вероятного действия (образец = False)
     :param: obs - единичный вектор наблюдения
     :param sample: если True, выборки из \pi, в противном случае выполняется
наиболее вероятное действие
     :returns: действие (одно целое число) и вероятности для всех действий"""

    probs = sess.run(probs_out, feed_dict={
                     observations_ph: obs.reshape((1, -1))})[0]

    if sample:
        action = int(np.random.choice(n_actions))
    else:
        action = int(np.argmax(probs))

    return action, probs
```

```
# демо
print("sampled:", [act(env.reset()) for _ in range(5)])
print("greedy:", [act(env.reset(), sample=False) for _ in range(5)])
```

Вывод:

sampled: [(1, array([0.00321023, nan, nan], dtype=float32)), (0, array([nan, 0.03026781, nan], dtype=float32)), (2, array([0.0192967 , 0.01878625, nan], dtype=float32)), (1, array([0.0144313, nan, nan], dtype=float32)), (2, array([nan, nan, nan], dtype=float32))]

greedy: [(1, array([0.00756173, nan, nan], dtype=float32)), (2, array([0.00590326, 0.03594141, nan], dtype=float32)), (1, array([0.0043306, nan, nan], dtype=float32)), (1, array([0.00883704, nan, nan], dtype=float32)), (0, array([nan, nan, nan], dtype=float32))]

Вычислите кумулятивное вознаграждение так же, как вы это делали в ванильном REINFORCE.

```
def get_cummulative_returns(r, gamma=1):
    """
    Вычисляет кумулятивное вознаграждение со скидкой при немедленном вознаграждении
    G_i = r_i + gamma*r_{i+1} + gamma^2*r_{i+2} + ...
    Известный также как R(s,a).
    """
    r = np.array(r)
    assert r.ndim >= 1
    return scipy.signal.lfilter([1], [1, -gamma], r[::-1], axis=0)[::-1]
```

```
# простая демонстрация для вознаграждения [0,0,1,0,0,1]
get_cummulative_returns([0, 0, 1, 0, 0, 1], gamma=0.9)
```

Развертывание

```
def rollout(env, act, max_pathlength=2500, n_timesteps=50000):
    """
    Создание выпусков для обучения.
     :param:env - окружение, в котором мы будем производить действия по генерации выкатов.
     :param:act - функция, которая может возвращать политику и действие с учетом наблюдения.
     :param: max_pathlength - максимальный размер одного пути, который мы генерируем.
     :param: n_timesteps - общая сумма размеров всех путей, которые мы генерируем.
    """
    paths = []

    total_timesteps = 0
    while total_timesteps < n_timesteps:
        obervations, actions, rewards, action_probs = [], [], [], []
        obervation = env.reset()
        for _ in range(max_pathlength):
            action, policy = act(obervation)
            obervations.append(obervation)
            actions.append(action)
            action_probs.append(policy)
            obervation, reward, done, _ = env.step(action)
            rewards.append(reward)
            total_timesteps += 1
            if done or total_timesteps == n_timesteps:
                path = {"observations": np.array(obervations),
                        "policy": np.array(action_probs),
                        "actions": np.array(actions),
                        "rewards": np.array(rewards),
                        "cumulative_returns":
get_cummulative_returns(rewards),
                        }
                paths.append(path)
```

```
                break
    return paths
```

```
paths = rollout(env, act, max_pathlength=5, n_timesteps=100)
print(paths[-1])
assert (paths[0]['policy'].shape == (5, n_actions))
assert (paths[0]['cumulative_returns'].shape == (5,))
assert (paths[0]['rewards'].shape == (5,))
assert (paths[0]['observations'].shape == (5,)+observation_shape)
assert (paths[0]['actions'].shape == (5,))
print('It\'s ok')
```

Вывод:

{'observations': array([[0.99940807, 0.03440231, 0.9998454 , -0.01758484, -0.03131559,
        0.08672629],
       [0.9999422 , 0.01075217, 0.9992699 , 0.03820498, -0.19845793,
        0.45750472],
       [0.9996275 , -0.0272931 , 0.99253607, 0.12195132, -0.17179134,
        0.3627333],
       [0.9991583 , -0.04101944, 0.98993856, 0.14149801, 0.03745227,
       -0.17025656],
       [0.9992117 , -0.03969756, 0.9921739 , 0.12486374, -0.02492967,
        0.00533957]], dtype=float32), 'policy': array([[0.56596464,   nan,   nan],
       [0.4888617 ,   nan,   nan],
       [0.4916529 ,   nan,   nan],
       [0.580484 ,   nan,   nan],
       [0.55329365,   nan,   nan]], dtype=float32), 'actions': array([2, 1, 0, 2, 0]), 'rewards': array([-1., -1., -1., -1., -1.]), 'cumulative_returns': array([-5., -4., -3., -2., -1.])}
It's ok

4. Функция потерь

Теперь давайте определим функции потерь и ограничения для фактического обучения TRPO.

Суррогатное вознаграждение должно быть:

$$J_{surr} = \frac{1}{N} \sum_{i=0}^{N} \frac{\pi_\theta(s_i, a_i)}{\pi_{\theta_{old}}(s_i, a_i)} A_{\theta_{old}(s_i, a_i)}$$

Для простоты давайте пока будем использовать кумулятивную доходность вместо преимущества:

$$J'_{surr} = \frac{1}{N} \sum_{i=0}^{N} \frac{\pi_\theta(s_i, a_i)}{\pi_{\theta_{old}}(s_i, a_i)} G_{\theta_{old}(s_i, a_i)}$$

Или, альтернативно, минимизируйте суррогатную потерю:

$$L_{surr} = -J'_{surr}$$

```
# выбрать вероятности выбранных действий
batch_size = tf.shape(observations_ph)[0]
probs_all = tf.reshape(probs_out, [-1])
probs_for_actions = tf.gather(probs_all, tf.range(
```

```
    0, batch_size) * n_actions + actions_ph)
old_probs_all = tf.reshape(old_probs_ph, [-1])
old_probs_for_actions = tf.gather(
    old_probs_all, tf.range(0, batch_size) * n_actions + actions_ph)
```

Требуется дополнить код. Что должен выполнять код в каждой позиции <ВАШ КОД> указано в примечаниях к коду.

```
# Вычислить суррогатные потери: отрицательный политический градиент с
выборкой по важности

L_surr = <ВАШ КОД: вычисляйте суррогатные потери, также известные как
_отрицательный_ градиент политик с выборкой по важности>
```

```
# вычислить и вернуть суррогатный градиент политики
def var_shape(x):
    return [k.value for k in x.get_shape()]


def numel(x):
    return np.prod(var_shape(x))


def flatgrad(loss, var_list):
    grads = tf.gradients(loss, var_list)
    return tf.concat([tf.reshape(grad, [numel(v)])
                      for (v, grad) in zip(var_list, grads)], 0)


flat_grad_surr = flatgrad(L_surr, weights)
```

Мы можем подниматься по этим градиентам до тех пор, пока $pi_\theta(a|s)$ удовлетворяет ограничению:

$$E_{s,\pi_{\Theta_t}}\Big[KL(\pi(\Theta_t, s) \;||\; \pi(\Theta_{t+1}, s))\Big] < \alpha$$

где

$$KL(p||q) = E_p log(\frac{p}{q})$$

Требуется дополнить код. Что должен выполнять код в каждой позиции <ВАШ КОД> указано в примечаниях к коду.

```
# Вычислить расхождение Кульбака-Лейблера (см. формулу выше)
# Примечание: вам нужно суммировать KL и энтропию по всем действиям, а не
только по тем, которые совершил агент
old_log_probs = tf.log(old_probs_ph+1e-10)

kl = <ВАШ КОД: рассчитайте расхождение Кульбака-Лейблера>

# Вычислите энтропию политики
entropy = <ВАШ КОД: вычислите энтропию политики. Не забудьте знак!>
```

```
losses = [L_surr, kl, entropy]
```

Линейный поиск

TRPO по своей сути включает восходящий градиент суррогатной политики, ограниченный дивергенцией KL. Чтобы обеспечить соблюдение этого ограничения, мы будем использовать линейный поиск.

```
def linesearch(f, x, fullstep, max_kl):
    """
    Linesearch находит лучшие параметры нейронных сетей в направлении полного шага, ограниченного расхождением KL.
     :param:f - функция, которая возвращает потери, kl и произвольную третью компоненту.
     :param: x - старые параметры нейросети.
     :param: fullstep - направление, в котором ищем.
     :param: max_kl - ограничение расхождения KL.
     :возвращается:
    """
    max_backtracks = 10
    loss, _, _ = f(x)
    for stepfrac in .5**np.arange(max_backtracks):
        xnew = x + stepfrac * fullstep
        new_loss, kl, _ = f(xnew)
        actual_improve = new_loss - loss
        if kl <= max_kl and actual_improve < 0:
            x = xnew
            loss = new_loss
    return x
```

6. Обучение

В этом разделе мы строим остальные части нашего вычислительного графа.

```
def slice_vector(vector, shapes):
    """
    Разбивает символьный вектор на несколько символьных тензоров заданных форм.
     Вспомогательная функция, используемая для разглаживания градиентов, касательных и т. д.
     :param vector: 1-мерный символьный вектор
     :param shape: список или кортеж фигур (список, кортеж или символ)
     :returns: список символических тензоров заданных форм
    """
    assert len(vector.get_shape()) == 1, "vector must be 1-dimensional"
    start = 0
    tensors = []
    for shape in shapes:
        size = np.prod(shape)
        tensor = tf.reshape(vector[start:(start + size)], shape)
        tensors.append(tensor)
        start += size
    return tensors
```

```
# средний уровень в conjugate_gradient
conjugate_grad_intermediate_vector = tf.placeholder(
    dtype=tf.float32, shape=(None,))

# разрезать flat_tangent на куски для каждого веса
```

```
weight_shapes = [sess.run(var).shape for var in weights]
tangents = slice_vector(conjugate_grad_intermediate_vector, weight_shapes)

# Расхождение KL, где первый аргумент фиксирован
kl_firstfixed = tf.reduce_sum((tf.stop_gradient(probs_out) *
(tf.stop_gradient(
    tf.log(probs_out)) - tf.log(probs_out)))) / tf.cast(batch_size,
tf.float32)

# вычислить информационную матрицу Фишера (используется для сопряженных
градиентов и для оценки KL)
gradients = tf.gradients(kl_firstfixed, weights)
gradient_vector_product = [tf.reduce_sum(
    g[0] * t) for (g, t) in zip(gradients, tangents)]

fisher_vec_prod = flatgrad(gradient_vector_product, weights)
```

7. Вспомогательные функции

Сопряженные градиенты

Поскольку TRPO включает оптимизацию с ограничениями, нам нужно будет решить Ax=b, используя сопряженные градиенты. В общем, CG — это алгоритм, который решает Ax=b, где A положительно определено. A — матрица Гессе, поэтому A положительно определена.

```
def conjugate_gradient(f_Ax, b, cg_iters=10, residual_tol=1e-10):
    """
    Этот метод решает систему уравнений Ax=b, используя итерационный метод,
называемый сопряженными градиентами.
     :f_Ax: функция, которая возвращает Ax
     :b: цели для топора
     :cg_iters: сколько итераций должен делать этот метод
     :residual_tol: эпсилон для стабильности
    """
    p = b.copy()
    r = b.copy()
    x = np.zeros_like(b)
    rdotr = r.dot(r)
    for i in range(cg_iters):
        z = f_Ax(p)
        v = rdotr / (p.dot(z) + 1e-8)
        x += v * p
        r -= v * z
        newrdotr = r.dot(r)
        mu = newrdotr / (rdotr + 1e-8)
        p = r + mu * p
        rdotr = newrdotr
        if rdotr < residual_tol:
            break
    return x
```

```
# Этот код проверяет сопряженные градиенты
A = np.random.rand(8, 8)
A = np.matmul(np.transpose(A), A)


def f_Ax(x):
    return np.matmul(A, x.reshape(-1, 1)).reshape(-1)


b = np.random.rand(8)
```

```
w = np.matmul(np.matmul(inv(np.matmul(np.transpose(A), A)),
                        np.transpose(A)), b.reshape((-1, 1))).reshape(-1)
print(w)
print(conjugate_gradient(f_Ax, b))
```

Вывод:

```
[-149.86919555 -108.59416764  131.789362    359.94990928 -156.74041779
  -49.10863303 -209.36261454  124.61801108]
[-149.86938114 -108.59413385  131.78915904  359.94983677 -156.74051792
  -49.10868357 -209.36247309  124.61786152]
```

```
# Скомпилируйте функцию, которая экспортирует веса сети в виде вектора
flat_weights = tf.concat([tf.reshape(var, [-1]) for var in weights], axis=0)

# ... и еще одна функция, которая импортирует вектор обратно в веса сети
flat_weights_placeholder = tf.placeholder(tf.float32, shape=(None,))
assigns = slice_vector(flat_weights_placeholder, weight_shapes)

load_flat_weights = [w.assign(ph) for w, ph in zip(weights, assigns)]
```

8. Главный цикл TRPO

```
import time
from itertools import count
from collections import OrderedDict

# это гиперпараметр TRPO. Он контролирует, насколько большим может быть
расхождение KL между старой и новой политикой на каждом этапе.
max_kl = 0.01
cg_damping = 0.1  # Эти параметры упорядочивают дополнение к
numeptotal = 0  # тому количеству эпизодов, которые мы сыграли.

start_time = time.time()

for i in count(1):

    print("\n********** Iteration %i ************" % i)

    # Generating paths.
    print("Rollout")
    paths = rollout(env, act)
    print("Made rollout")

    # Обновление политики.
    observations = np.concatenate([path["observations"] for path in paths])
    actions = np.concatenate([path["actions"] for path in paths])
    returns = np.concatenate([path["cumulative_returns"] for path in paths])
    old_probs = np.concatenate([path["policy"] for path in paths])
    inputs_batch = [observations, actions, returns, old_probs]
    feed_dict = {observations_ph: observations,
                 actions_ph: actions,
                 old_probs_ph: old_probs,
                 cummulative_returns_ph: returns,
                 }
    old_weights = sess.run(flat_weights)

    def fisher_vector_product(p):
        """gets intermediate grads (p) and computes fisher*vector """
        feed_dict[conjugate_grad_intermediate_vector] = p
        return sess.run(fisher_vec_prod, feed_dict) + cg_damping * p
```

```
    flat_grad = sess.run(flat_grad_surr, feed_dict)

    stepdir = conjugate_gradient(fisher_vector_product, -flat_grad)
    shs = .5 * stepdir.dot(fisher_vector_product(stepdir))
    lm = np.sqrt(shs / max_kl)
    fullstep = stepdir / lm

    # Вычислить новые веса с помощью линейного поиска в направлении, которое
мы нашли с помощью компьютерной графики.

    def losses_f(flat_weights):
        feed_dict[flat_weights_placeholder] = flat_weights
        sess.run(load_flat_weights, feed_dict)
        return sess.run(losses, feed_dict)

    new_weights = linesearch(losses_f, old_weights, fullstep, max_kl)
    feed_dict[flat_weights_placeholder] = new_weights
    sess.run(load_flat_weights, feed_dict)

    # Сообщить о текущем прогрессе
    L_surr, kl, entropy = sess.run(losses, feed_dict)
    episode_rewards = np.array([path["rewards"].sum() for path in paths])

    stats = OrderedDict()
    numeptotal += len(episode_rewards)
    stats["Total number of episodes"] = numeptotal
    stats["Average sum of rewards per episode"] = episode_rewards.mean()
    stats["Std of rewards per episode"] = episode_rewards.std()
    stats["Entropy"] = entropy
    stats["Time elapsed"] = "%.2f mins" % ((time.time() - start_time)/60.)
    stats["KL between old and new distribution"] = kl
    stats["Surrogate loss"] = L_surr
    for k, v in stats.items():
        print(k + ": " + " " * (40 - len(k)) + str(v))
    i += 1
```

Вывод:

********** Iteration 1 **********
Rollout
Made rollout
Total number of episodes: 100
Average sum of rewards per episode: -500.0
Std of rewards per episode: 0.0
Entropy: [nan]
Time elapsed: 0.64 mins
KL between old and new distribution: nan
Surrogate loss: [nan]
…
********** Iteration 4 **********
Rollout
Made rollout
Total number of episodes: 403
Average sum of rewards per episode: -495.009900990099
Std of rewards per episode: 24.995641223785736
Entropy: [nan]
Time elapsed: 2.55 mins
KL between old and new distribution: nan
Surrogate loss: [nan]
…

Задания к лабораторной работе №1

1. Написать свой код к разделам 1 - 8 согласно заданиям для всех фрагментов, помеченных как: <ВАШ КОД:>

```
<ВАШ КОД: >
```

2. Запустить полученную программу, получить ожидаемые выходные данные.

Задания к лабораторной работе №2

1. Замените библиотеку Tensorflow на Pytorch и воспроизведите все эксперименты из разделов 1-8.

Требование к отчету

1. В качестве отчета принимаются два Python файла с кодом (для Tensorflow и Pytorch).
2. Код должен быть в достаточной мере прокомментирован.
3. Отчет предоставляется в виде архива zip, формат имени архива: <номер группы>_<Фамилия_Имя>_LR<номер работы>.zip