Лабораторная работа №6.

Простой policy gradient алгоритм (REINFORCE)

1. Использование Tensorflow

Как и раньше для Q-обучения мы разработаем сеть TensorFlow для изучения CartPole-v0 с помощью градиента политик (REINFORCE).

```
import sys, os

# Этот код создает виртуальный дисплей для рисования игровых изображений.
# Это не будет иметь никакого эффекта, если на вашей машине есть монитор.
if type(os.environ.get("DISPLAY")) is not str or
len(os.environ.get("DISPLAY")) == 0:
    os.environ['DISPLAY'] = ':1'
```

```
import gym
import numpy as np
import matplotlib.pyplot as plt
```

```
env = gym.make("CartPole-v0")

# gym compatibility: unwrap TimeLimit
if hasattr(env, '_max_episode_steps'):
    env = env.env

env.reset()
n_actions = env.action_space.n
state_dim = env.observation_space.shape

plt.imshow(env.render("rgb_array"))
```

Для алгоритма REINFORCE нам понадобится модель, которая предсказывает вероятности действий при заданных состояниях. Для стабильной работы не включайте слой softmax в вашу сетевую архитектуру. Мы будем использовать softmax или log-softmax, где это уместно.

```
import tensorflow.compat.v1 as tf
tf.disable_v2_behavior()

sess = tf.InteractiveSession()
```

```
# создать входные переменные. Нам нужны только <s, a, r> для REINFORCE
ph_states = tf.placeholder('float32', (None,) + state_dim, name="states")
ph_actions = tf.placeholder('int32', name="action_ids")
ph_cumulative_rewards = tf.placeholder('float32', name="cumulative_returns")
```

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

```
import keras
import keras.layers as L


<ВАШ КОД: определить сетевой граф, используя TF>

logits = <ВАШ КОД: символьный вывод ИНС _before_ softmax>
```

```
policy = tf.nn.softmax(logits)
log_policy = tf.nn.log_softmax(logits)
```

```
# инициализация параметров модели
sess.run(tf.global_variables_initializer())
```

```
def predict_probs(states):
    """
    Прогнозировать вероятности действий при заданных состояниях.
     :paramstates: numpy массив форм [batch, state_shape]
     :returns: numpy массив формы [пакет, n_actions]
    """
    return policy.eval({ph_states: [states]})[0]
```

Игра с моделью.

Теперь мы можем использовать нашего недавно созданного агента для игры. Требуется дополнить код. Что должен выполнять код в каждой позиции <ВАШ КОД> указано в примечаниях к коду.

```
def generate_session(env, t_max=1000):
    """
    Сыграйте полную сессию с агентом REINFORCE.
     Возвращает последовательности состояний, действий и наград.
    """
    # массивы для записи сессии
    states, actions, rewards = [], [], []
    s = env.reset()

    for t in range(t_max):
        # массив вероятностей действий pi(a|s)
        action_probs = predict_probs(s)

        # Пример действия с заданной вероятностью
        a = <ВАШ КОД>
        new_s, r, done, info = env.step(a)

        # записать историю сессий, для последующего обучения
        states.append(s)
        actions.append(a)
        rewards.append(r)

        s = new_s
        if done:
            break

    return states, actions, rewards
```

```
# Проверка
states, actions, rewards = generate_session(env)
```

Расчет кумулятивных вознаграждений.

$$
\begin{aligned}
G_t &= r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \dots \\
&= \sum_{i=t}^{T} \gamma^{i-t} r_i \\
&= r_t + \gamma * G_{t+1}
\end{aligned}
$$

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

```
def get_cumulative_rewards(rewards,  # вознаграждение на каждом шаге
                           gamma=0.99  # дисконт для вознаграждение
                           ):
    """
    Возьмите список немедленных вознаграждений r(s,a) за всю сессию
     и вычислить кумулятивную доходность (также известную как G(s,a) в Sutton
'16).

     G_t = r_t + гамма*r_{t+1} + гамма^2*r_{t+2} + ...

     Простой способ вычислить кумулятивное вознаграждение состоит в том,
чтобы выполнить итерацию, начиная с последнего
     к первому временному шагу и рекуррентно вычислить G_t = r_t +
gamma*G_{t+1}

     Вы должны вернуть массив/список кумулятивных вознаграждений с таким же
количеством элементов, как и в начальных вознаграждениях.
    """
    < ВАШ КОД >
    return < ВАШ КОД: массив кумулятивных вознаграждений>
```

```
assert len(get_cumulative_rewards(range(100))) == 100
assert np.allclose(
    get_cumulative_rewards([0, 0, 1, 0, 0, 1, 0], gamma=0.9),
    [1.40049, 1.5561, 1.729, 0.81, 0.9, 1.0, 0.0])
assert np.allclose(
    get_cumulative_rewards([0, 0, 1, -2, 3, -4, 0], gamma=0.5),
    [0.0625, 0.125, 0.25, -1.5, 1.0, -4.0, 0.0])
assert np.allclose(
    get_cumulative_rewards([0, 0, 1, 2, 3, 4, 0], gamma=0),
    [0, 0, 1, 2, 3, 4, 0])
print("looks good!")
```

Функция потерь и обновления.

Теперь нам нужно определить цель и обновить градиент политики.

Наша целевая функция

$$
J \approx \frac{1}{N} \sum_{s_i, a_i} G(s_i, a_i)
$$

REINFORCE определяет способ вычисления градиента ожидаемого вознаграждения по отношению к параметрам политики. Формула выглядит следующим образом:

$$\nabla_\theta \hat{J}(\theta) \approx \frac{1}{N} \sum_{s_i, a_i} \nabla_\theta \log \pi_\theta(a_i \mid s_i) \cdot G_t(s_i, a_i)$$

Мы можем использовать возможности Tensorflow для автоматического дифференцирования, определяя нашу целевую функцию следующим образом:

$$\hat{J}(\theta) \approx \frac{1}{N} \sum_{s_i, a_i} \log \pi_\theta(a_i \mid s_i) \cdot G_t(s_i, a_i)$$

Когда вы вычисляете градиент этой функции по отношению к весам сети θ , он станет именно градиентом политики.

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

```
# Этот код выбирает логарифмические вероятности (log pi(a_i|s_i)) для тех
действий, которые были фактически сыграны.
indices = tf.stack([tf.range(tf.shape(log_policy)[0]), ph_actions], axis=-1)
log_policy_for_actions = tf.gather_nd(log_policy, indices)
```

```
# Цель политики, как в последней формуле. Пожалуйста, используйте
reduce_mean, а не reduce_sum.
# Вы можете использовать log_policy_for_actions для получения логарифмических
вероятностей предпринятых действий.
# Также помните, что ранее мы определили ph_cumulative_rewards.
J = <ВАШ КОД>
```

Напоминаем, что для дискретного распределения вероятностей (подобного тому, которое дает наша политика) энтропия определяется как:

$$\text{entropy}(p) = -\sum_{i=1}^{n} p_i \cdot \log p_i$$

```
# Регуляризация энтропии. Если вы его не добавите, политика быстро испортится
до
# будучи детерминированным, вредит исследованию.

entropy = <ВАШ КОД: вычислить энтропию. Не забудьте знак!>
```

```
#Максимизация X аналогична минимизации -X, отсюда и знак.
loss = -(J + 0.1 * entropy)
```

```
def train_on_session(states, actions, rewards, t_max=1000):
    """в полном объеме обучает агента градиенту политики"""
    cumulative_rewards = get_cumulative_rewards(rewards)
    update.run({
        ph_states: states,
        ph_actions: actions,
        ph_cumulative_rewards: cumulative_rewards,
    })
    return sum(rewards)
```

```
# Инициализация оптимизированных параметров
sess.run(tf.global_variables_initializer())
```

Актуальное обучение

```
for i in range(100):
    rewards = [train_on_session(*generate_session(env)) for _ in range(100)]
# создание новой сессии

    print("mean reward: %.3f" % (np.mean(rewards)))

    if np.mean(rewards) > 300:
        print("You Win!")  # но обучение может быть продолжено
        break
```

Вывод:

looks good!
mean reward: 26.530
mean reward: 25.540
mean reward: 33.940
mean reward: 65.190
mean reward: 109.290
mean reward: 175.360
mean reward: 232.870
mean reward: 248.980
mean reward: 159.860
mean reward: 230.800
mean reward: 459.320
You Win!

Задания к лабораторной работе №1

1. Написать свой код к разделу «Использование Tensorflow» 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 раздела «Использование Tensorflow».

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