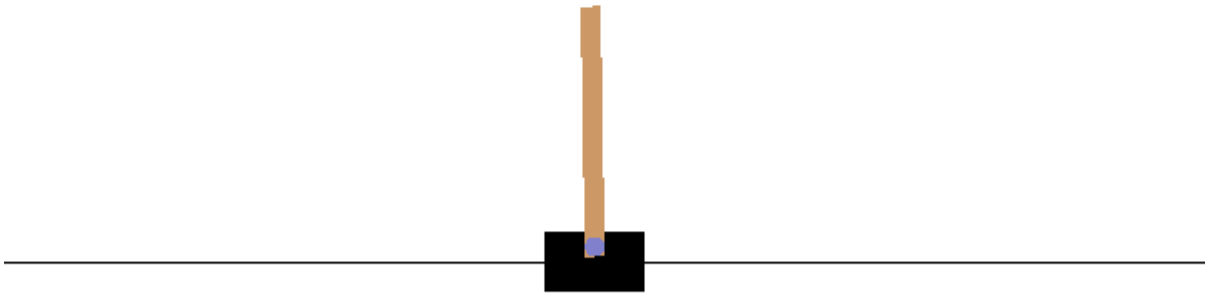
Train a Deep Q Network with TF-Agents

 View on TensorFlow.org  Run in Google Colab  View source on GitHub  Download notebook

Introduction

This example shows how to train a DQN (Deep Q Networks) agent on the Cartpole environment using the TF-Agents library.



It will walk you through all the components in a Reinforcement Learning (RL) pipeline for training, evaluation and data collection.

To run this code live, click the 'Run in Google Colab' link above.

Setup

If you haven't installed the following dependencies, run:

```
!sudo apt-get install -y xvfb ffmpeg
!pip install 'gym==0.10.11'
!pip install 'imageio==2.4.0'
!pip install PILLOW
!pip install 'pyglet==1.3.2'
```

```
!pip install pyvirtualdisplay
!pip install tf-agents
```

```
Reading package lists... Done
Building dependency tree
Reading state information... Done
ffmpeg is already the newest version (7:3.4.6-0ubuntu0.18.04.1).
The following package was automatically installed and is no longer required:
  libnvidia-common-440
Use 'sudo apt autoremove' to remove it.
The following NEW packages will be installed:
  xvfb
0 upgraded, 1 newly installed, 0 to remove and 35 not upgraded.
Need to get 784 kB of archives.
After this operation, 2,266 kB of additional disk space will be used.
Get:1 http://archive.ubuntu.com/ubuntu bionic-updates/universe amd64 xvfb amd64 2:1.19.6-1ubuntu4.4 [784 kB]
Fetched 784 kB in 1s (718 kB/s)
debconf: unable to initialize frontend: Dialog
debconf: (No usable dialog-like program is installed, so the dialog based frontend cannot be used. at /usr/share/perl5
debconf: falling back to frontend: Readline
debconf: unable to initialize frontend: Readline
debconf: (This frontend requires a controlling tty.)
debconf: falling back to frontend: Teletype
dpkg-preconfigure: unable to re-open stdin:
Selecting previously unselected package xvfb.
(Reading database ... 144465 files and directories currently installed.)
Preparing to unpack .../xvfb_2%3a1.19.6-1ubuntu4.4_amd64.deb ...
Unpacking xvfb (2:1.19.6-1ubuntu4.4) ...
Setting up xvfb (2:1.19.6-1ubuntu4.4) ...
Processing triggers for man-db (2.8.3-2ubuntu0.1) ...
Collecting gym==0.10.11
  Downloading https://files.pythonhosted.org/packages/87/04/70d4901b7105082c9742acd64728342f6da7cd471572fd0660a73f9cfe
     |████████████████████████████████| 1.5MB 4.7MB/s
Requirement already satisfied: scipy in /usr/local/lib/python3.6/dist-packages (from gym==0.10.11) (1.4.1)
Requirement already satisfied: numpy>=1.10.4 in /usr/local/lib/python3.6/dist-packages (from gym==0.10.11) (1.18.5)
Requirement already satisfied: requests>=2.0 in /usr/local/lib/python3.6/dist-packages (from gym==0.10.11) (2.23.0)
Requirement already satisfied: six in /usr/local/lib/python3.6/dist-packages (from gym==0.10.11) (1.15.0)
Requirement already satisfied: pyglet>=1.2.0 in /usr/local/lib/python3.6/dist-packages (from gym==0.10.11) (1.5.0)
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.6/dist-packages (from requests>=2.0->gym==0.10.1
Requirement already satisfied: chardet<4,>=3.0.2 in /usr/local/lib/python3.6/dist-packages (from requests>=2.0->gym==0
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.6/dist-packages (from requests>=2.0->gym==
Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 in /usr/local/lib/python3.6/dist-packages (from
Requirement already satisfied: future in /usr/local/lib/python3.6/dist-packages (from pyglet>=1.2.0->gym==0.10.11) (0.
Building wheels for collected packages: gym
  Building wheel for gym (setup.py) ... done
  Created wheel for gym: filename=gym-0.10.11-cp36-none-any.whl size=1588313 sha256=58559d119ca010fb86a63172aaa692e379
```

```
  Stored in directory: /root/.cache/pip/wheels/7b/eb/1f/22c4124f3c64943aa0646daf4612b1c1f00f27d89b81304ebd
Successfully built gym
Installing collected packages: gym
  Found existing installation: gym 0.17.2
    Uninstalling gym-0.17.2:
      Successfully uninstalled gym-0.17.2
Successfully installed gym-0.10.11
Collecting imageio==2.4.0
  Downloading https://files.pythonhosted.org/packages/ac/64/8e2bb6aac43d6ed7c2d9514320b43d5e80c00f150ee2b9408aee24359e
     |████████████████████████████████| 3.3MB 4.7MB/s
Requirement already satisfied: numpy in /usr/local/lib/python3.6/dist-packages (from imageio==2.4.0) (1.18.5)
Requirement already satisfied: pillow in /usr/local/lib/python3.6/dist-packages (from imageio==2.4.0) (7.0.0)
Building wheels for collected packages: imageio
  Building wheel for imageio (setup.py) ... done
  Created wheel for imageio: filename=imageio-2.4.0-cp36-none-any.whl size=3303880 sha256=9416147a55660c7767afd844b276
  Stored in directory: /root/.cache/pip/wheels/31/83/88/a1cba54ac06395d9e4ddcd9cf06911cd0b26cd78af9a61071b
Successfully built imageio
ERROR: albumentations 0.1.12 has requirement imgaug<0.2.7,>=0.2.5, but you'll have imgaug 0.2.9 which is incompatible.
Installing collected packages: imageio
  Found existing installation: imageio 2.4.1
```

```
from __future__ import absolute_import, division, print_function

import base64
import imageio
import IPython
import matplotlib
import matplotlib.pyplot as plt
import numpy as np
import PIL.Image
import pyvirtualdisplay

import tensorflow as tf

from tf_agents.agents.dqn import dqn_agent
from tf_agents.drivers import dynamic_step_driver
from tf_agents.environments import suite_gym
from tf_agents.environments import tf_py_environment
from tf_agents.eval import metric_utils
from tf_agents.metrics import tf_metrics
from tf_agents.networks import q_network
from tf_agents.policies import random_tf_policy
```

```
from tf_agents.policies import random_tf_policy
from tf_agents.replay_buffers import tf_uniform_replay_buffer
from tf_agents.trajectories import trajectory
from tf_agents.utils import common
```

```
     |████████████████████████████████| 51kB 7.0MB/s
```

```
tf.compat.v1.enable_v2_behavior()

# Set up a virtual display for rendering OpenAI gym environments.
display = pyvirtualdisplay.Display(visible=0, size=(1400, 900)).start()
```

```
Installing collected packages: gin-config, tf-agents
```

```
tf.version.VERSION
```

```
'2.2.0'
```

Hyperparameters

```
num_iterations = 20000 # @param {type:"integer"}

initial_collect_steps = 1000  # @param {type:"integer"}
collect_steps_per_iteration = 1  # @param {type:"integer"}
replay_buffer_max_length = 100000  # @param {type:"integer"}

batch_size = 64  # @param {type:"integer"}
learning_rate = 1e-3  # @param {type:"number"}
log_interval = 200  # @param {type:"integer"}

num_eval_episodes = 10  # @param {type:"integer"}
eval_interval = 1000  # @param {type:"integer"}
```

num_iterations: 20000

initial_collect_steps: 1000

collect_steps_per_iteration: 1

replay_buffer_max_length: 100000

batch_size: 64

learning_rate: 1e-3

log_interval: 200

num_eval_episodes: 10

eval_interval: 1000

Environment

In Reinforcement Learning (RL), an environment represents the task or problem to be solved. Standard environments can be created in TF-Agents using `tf_agents.environments` suites. TF-Agents has suites for loading environments from sources such as the OpenAI Gym, Atari, and DM Control.

Load the CartPole environment from the OpenAI Gym suite.

```
env_name = 'CartPole-v0'
env = suite_gym.load(env_name)
```

You can render this environment to see how it looks. A free-swinging pole is attached to a cart. The goal is to move the cart right or left in order to keep the pole pointing up.

```
#@test {"skip": true}
env.reset()
PIL.Image.fromarray(env.render())
```

The `environment.step` method takes an `action` in the environment and returns a `TimeStep` tuple containing the next observation of the environment and the reward for the action.

The `time_step_spec()` method returns the specification for the `TimeStep` tuple. Its `observation` attribute shows the shape of observations, the data types, and the ranges of allowed values. The `reward` attribute shows the same details for the reward.

```
print('Observation Spec:')
print(env.time_step_spec().observation)
```

```
Observation Spec:
BoundedArraySpec(shape=(4,), dtype=dtype('float32'), name='observation', minimum=[-4.8000002e+00 -3.4028235e+38 -4.188
```

```
print('Reward Spec:')
print(env.time_step_spec().reward)
```

```
Reward Spec:
ArraySpec(shape=(), dtype=dtype('float32'), name='reward')
```

The `action_spec()` method returns the shape, data types, and allowed values of valid actions.

```
print('Action Spec:')
print(env.action_spec())
```

```
Action Spec:
BoundedArraySpec(shape=(), dtype=dtype('int64'), name='action', minimum=0, maximum=1)
```

In the Cartpole environment:

- `observation` is an array of 4 floats:
  - the position and velocity of the cart
  - the angular position and velocity of the pole
- `reward` is a scalar float value
- `action` is a scalar integer with only two possible values:
  - `0` — "move left"
  - `1` — "move right"

```
time_step = env.reset()
print('Time step:')
print(time_step)

action = np.array(1, dtype=np.int32)

next_time_step = env.step(action)
print('Next time step:')
print(next_time_step)
```

```
Time step:
TimeStep(step_type=array(0, dtype=int32), reward=array(0., dtype=float32), discount=array(1., dtype=float32), observat
Next time step:
TimeStep(step_type=array(1, dtype=int32), reward=array(1., dtype=float32), discount=array(1., dtype=float32), observat
```

Usually two environments are instantiated: one for training and one for evaluation.

```
train_py_env = suite_gym.load(env_name)
eval_py_env = suite_gym.load(env_name)
```

The Cartpole environment, like most environments, is written in pure Python. This is converted to TensorFlow using the `TFPyEnvironment` wrapper.

The original environment's API uses Numpy arrays. The `TFPyEnvironment` converts these to `Tensors` to make it compatible with Tensorflow agents and policies.

```
train_env = tf_py_environment.TFPyEnvironment(train_py_env)
eval_env = tf_py_environment.TFPyEnvironment(eval_py_env)
```

Agent

The algorithm used to solve an RL problem is represented by an `Agent`. TF-Agents provides standard implementations of a variety of `Agents`, including:

- DQN (used in this tutorial)
- REINFORCE
- DDPG
- TD3
- PPO
- SAC.

The DQN agent can be used in any environment which has a discrete action space.

At the heart of a DQN Agent is a `QNetwork`, a neural network model that can learn to predict `QValues` (expected returns) for all actions, given an observation from the environment.

Use `tf_agents.networks.q_network` to create a `QNetwork`, passing in the `observation_spec`, `action_spec`, and a tuple describing the number and size of the model's hidden layers.

```
fc_layer_params = (100,)

q_net = q_network.QNetwork(
    train_env.observation_spec(),
```

```
    train_env.action_spec(),
    fc_layer_params=fc_layer_params)
```

Now use `tf_agents.agents.dqn.dqn_agent` to instantiate a `DqnAgent`. In addition to the `time_step_spec`, `action_spec` and the QNetwork, the agent constructor also requires an optimizer (in this case, `AdamOptimizer`), a loss function, and an integer step counter.

```
optimizer = tf.compat.v1.train.AdamOptimizer(learning_rate=learning_rate)

train_step_counter = tf.Variable(0)

agent = dqn_agent.DqnAgent(
    train_env.time_step_spec(),
    train_env.action_spec(),
    q_network=q_net,
    optimizer=optimizer,
    td_errors_loss_fn=common.element_wise_squared_loss,
    train_step_counter=train_step_counter)

agent.initialize()
```

Policies

A policy defines the way an agent acts in an environment. Typically, the goal of reinforcement learning is to train the underlying model until the policy produces the desired outcome.

In this tutorial:

- The desired outcome is keeping the pole balanced upright over the cart.
- The policy returns an action (left or right) for each `time_step` observation.

Agents contain two policies:

- `agent.policy` — The main policy that is used for evaluation and deployment.
- `agent.collect_policy` — A second policy that is used for data collection.

```
eval_policy = agent.policy
collect_policy = agent.collect_policy
```

Policies can be created independently of agents. For example, use `tf_agents.policies.random_tf_policy` to create a policy which will randomly select an action for each `time_step`.

```
random_policy = random_tf_policy.RandomTFPolicy(train_env.time_step_spec(),
                                                train_env.action_spec())
```

To get an action from a policy, call the `policy.action(time_step)` method. The `time_step` contains the observation from the environment. This method returns a `PolicyStep`, which is a named tuple with three components:

- `action` — the action to be taken (in this case, `0` or `1`)
- `state` — used for stateful (that is, RNN-based) policies
- `info` — auxiliary data, such as log probabilities of actions

```
example_environment = tf_py_environment.TFPyEnvironment(
    suite_gym.load('CartPole-v0'))
```

```
time_step = example_environment.reset()
```

```
random_policy.action(time_step)
```

```
PolicyStep(action=<tf.Tensor: shape=(1,), dtype=int64, numpy=array([0])>, state=(), info=())
```

Metrics and Evaluation

The most common metric used to evaluate a policy is the average return. The return is the sum of rewards obtained while running a policy in an environment for an episode. Several episodes are run, creating an average return.

The following function computes the average return of a policy, given the policy, environment, and a number of episodes.

```
#@test {"skip": true}
def compute_avg_return(environment, policy, num_episodes=10):

  total_return = 0.0
  for _ in range(num_episodes):

    time_step = environment.reset()
    episode_return = 0.0

    while not time_step.is_last():
      action_step = policy.action(time_step)
      time_step = environment.step(action_step.action)
      episode_return += time_step.reward
    total_return += episode_return

  avg_return = total_return / num_episodes
  return avg_return.numpy()[0]


# See also the metrics module for standard implementations of different metrics.
# https://github.com/tensorflow/agents/tree/master/tf_agents/metrics
```

Running this computation on the `random_policy` shows a baseline performance in the environment.

```
compute_avg_return(eval_env, random_policy, num_eval_episodes)
```

```
30.1
```

Replay Buffer

The replay buffer keeps track of data collected from the environment. This tutorial uses `tf_agents.replay_buffers.tf_uniform_replay_buffer.TFUniformReplayBuffer`, as it is the most common.

The constructor requires the specs for the data it will be collecting. This is available from the agent using the `collect_data_spec` method. The batch size and maximum buffer length are also required.

```
replay_buffer = tf_uniform_replay_buffer.TFUniformReplayBuffer(
    data_spec=agent.collect_data_spec,
    batch_size=train_env.batch_size,
    max_length=replay_buffer_max_length)
```

For most agents, `collect_data_spec` is a named tuple called `Trajectory`, containing the specs for observations, actions, rewards, and other items.

```
agent.collect_data_spec
```

```
Trajectory(step_type=TensorSpec(shape=(), dtype=tf.int32, name='step_type'), observation=BoundedTensorSpec(shape=(4,),
      dtype=float32), maximum=array([4.8000002e+00, 3.4028235e+38, 4.1887903e-01, 3.4028235e+38],
      dtype=float32)), action=BoundedTensorSpec(shape=(), dtype=tf.int64, name='action', minimum=array(0), maximum=arr
```

```
agent.collect_data_spec._fields
```

```
('step_type',
 'observation',
 'action',
 'policy_info',
 'next_step_type',
 'reward',
 'discount')
```

Data Collection

Now execute the random policy in the environment for a few steps, recording the data in the replay buffer.

```
#@test {"skip": true}
```

```
def collect_step(environment, policy, buffer):
  time_step = environment.current_time_step()
  action_step = policy.action(time_step)
  next_time_step = environment.step(action_step.action)
  traj = trajectory.from_transition(time_step, action_step, next_time_step)

  # Add trajectory to the replay buffer
  buffer.add_batch(traj)

def collect_data(env, policy, buffer, steps):
  for _ in range(steps):
    collect_step(env, policy, buffer)

collect_data(train_env, random_policy, replay_buffer, steps=100)

# This loop is so common in RL, that we provide standard implementations.
# For more details see the drivers module.
# https://www.tensorflow.org/agents/api_docs/python/tf_agents/drivers
```

The replay buffer is now a collection of Trajectories.

```
# For the curious:
# Uncomment to peel one of these off and inspect it.
iter(replay_buffer.as_dataset()).next()
```

```
(Trajectory(step_type=<tf.Tensor: shape=(), dtype=int32, numpy=1>, observation=<tf.Tensor: shape=(4,), dtype=float32,
 BufferInfo(ids=<tf.Tensor: shape=(), dtype=int64, numpy=19>, probabilities=<tf.Tensor: shape=(), dtype=float32, numpy
```

The agent needs access to the replay buffer. This is provided by creating an iterable `tf.data.Dataset` pipeline which will feed data to the agent.

Each row of the replay buffer only stores a single observation step. But since the DQN Agent needs both the current and next observation to compute the loss, the dataset pipeline will sample two adjacent rows for each item in the batch (`num_steps=2`).

This dataset is also optimized by running parallel calls and prefetching data.

```
# Dataset generates trajectories with shape [Bx2x...]
dataset = replay_buffer.as_dataset(
    num_parallel_calls=3,
    sample_batch_size=batch_size,
    num_steps=2).prefetch(3)


dataset
```

```
<PrefetchDataset shapes: (Trajectory(step_type=(64, 2), observation=(64, 2, 4), action=(64, 2), policy_info=(), next_s
```

```
iterator = iter(dataset)

print(iterator)
```

```
<tensorflow.python.data.ops.iterator_ops.OwnedIterator object at 0x7f8a02a186d8>
```

```
# For the curious:
# Uncomment to see what the dataset iterator is feeding to the agent.
# Compare this representation of replay data
# to the collection of individual trajectories shown earlier.

iterator.next()
```

```
(Trajectory(step_type=<tf.Tensor: shape=(64, 2), dtype=int32, numpy=
array([[1, 1],
       [2, 0],
       [1, 1],
       [1, 1],
       [1, 1],
       [0, 1],
       [1, 1],
       [1, 1],
       [1, 1],
       [1, 1],
       [1, 1],
       [1, 1],
       [1, 1],
       [1, 1],
       [1, 1],
       [1, 1],
       [1, 1],
       [1, 1],
       [1, 1],
       [1, 1],
       [1, 1],
       [1, 1],
       [1, 1],
       [1, 1],
       [1, 1],
       [1, 1],
       [1, 1],
       [1, 1],
       [1, 1],
       [1, 1],
       [1, 1],
       [1, 1],
       [1, 1],
       [1, 1],
       [1, 1],
       [1, 1],
       [1, 1],
       [1, 1],
       [1, 1],
       [1, 1],
       [1, 1],
       [0, 1],
       [0, 1],
       [1, 1],
       [1, 1],
       [2, 0],
       [1, 1],
```

```
        [1, 1],
        [1, 1],
        [1, 1],
        [1, 1],
        [1, 1],
        [1, 1],
        [1, 1],
        [1, 1],
        [1, 1],
        [1, 1],
        [1, 1],
        [1, 1],
        [1, 1],
        [1, 1],
        [1, 1],
        [1, 1],
        [1, 1],
        [1, 1],
        [1, 1],
        [1, 1],
        [1, 1]], dtype=int32)>, observation=<tf.Tensor: shape=(64, 2, 4), dtype=float32, numpy=
array([[[ 4.46458459e-02,  1.81610674e-01, -3.88418473e-02,
         -3.42700750e-01],
        [ 4.82780598e-02, -1.29377712e-02, -4.56958637e-02,
         -6.25148118e-02]],

       [[ 1.04432702e-01, -4.01968323e-02,  2.09584549e-01,
          1.23149288e+00],
        [ 4.49265689e-02, -1.40360268e-02, -3.80768180e-02,
         -3.82515267e-02]],

       [[ 8.20634216e-02,  7.72538364e-01, -1.10777624e-01,
         -1.34970200e+00],
        [ 9.75141898e-02,  5.78968763e-01, -1.37771666e-01,
         -1.09362936e+00]],

       [[ 4.26519699e-02,  2.15767264e-01,  1.17772445e-01,
          2.34734535e-01],
        [ 4.69673127e-02,  4.09026831e-01,  1.22467138e-01,
         -1.86016131e-02]],

       [[-4.46951725e-02, -3.45672816e-01,  3.45627218e-02,
          5.83720922e-01],
```

```
   -5.16086295e-02, -1.51051655e-01,  4.62371409e-02,
    3.02122951e-01]],

  [[ 4.49265689e-02, -1.40360268e-02, -3.80768180e-02,
    -3.82515267e-02],
   [ 4.46458459e-02,  1.81610674e-01, -3.88418473e-02,
    -3.42700750e-01]],

  [[ 2.77761854e-02,  4.20307875e-01,  3.01261805e-02,
    -2.67398030e-01],
   [ 3.61823440e-02,  2.24769205e-01,  2.47782208e-02,
     3.46326269e-02]],

  [[-3.99961583e-02,  3.84843946e-01,  1.50386006e-01,
    -1.47278413e-01],
   [-3.22992802e-02,  5.77528238e-01,  1.47440434e-01,
    -3.88995677e-01]],

  [[ 1.16811641e-01,  5.82801342e-01, -1.76587313e-01,
    -1.18547988e+00],
   [ 1.28467664e-01,  7.79717982e-01, -2.00296909e-01,
    -1.52790868e+00]],

  [[ 4.85234335e-02, -1.69285849e-01,  7.00470358e-02,
     7.05473423e-01],
   [ 4.51377183e-02,  2.47992296e-02,  8.41565058e-02,
     4.35636759e-01]],

  [[-6.59381039e-03, -1.81848004e-01,  5.17802611e-02,
     3.21032554e-01],
   [-1.02307703e-02,  1.24997813e-02,  5.82009144e-02,
     4.51181978e-02]],

  [[-1.66508369e-03,  7.66429186e-01,  1.32601276e-01,
    -5.45228422e-01],
   [ 1.36635005e-02,  5.69717705e-01,  1.21696711e-01,
    -2.13880837e-01]],

  [[ 4.80193049e-02,  1.82808548e-01, -4.69461605e-02,
    -3.69257718e-01],
   [ 5.16754761e-02,  3.78565013e-01, -5.43313138e-02,
    -6.76365972e-01]],
```

```
[[-8.68500443e-04,  3.42540890e-02,  2.36211643e-02,
   2.25739211e-01],
 [-1.83418670e-04,  2.29030624e-01,  2.81359479e-02,
  -5.94001114e-02]],

[[ 2.77761854e-02,  4.20307875e-01,  3.01261805e-02,
  -2.67398030e-01],
 [ 3.61823440e-02,  2.24769205e-01,  2.47782208e-02,
   3.46326269e-02]],

[[ 4.90682721e-02,  2.24050969e-01,  2.04682592e-02,
   5.04762158e-02],
 [ 5.35492897e-02,  2.86415983e-02,  2.14777850e-02,
   3.49546134e-01]],

[[-4.57422249e-02,  4.29024458e-01,  5.42156883e-02,
  -4.60334748e-01],
 [-3.71617377e-02,  2.33179778e-01,  4.50089946e-02,
  -1.51067749e-01]],

[[ 7.05366209e-02,  5.76339781e-01, -9.01751369e-02,
  -1.03012431e+00],
 [ 8.20634216e-02,  7.72538364e-01, -1.10777624e-01,
  -1.34970200e+00]],

[[ 4.79902215e-02,  2.66606715e-02,  6.21699281e-02,
   3.93855304e-01],
 [ 4.85234335e-02, -1.69285849e-01,  7.00470358e-02,
   7.05473423e-01]],

[[ 5.16754761e-02,  3.78565013e-01, -5.43313138e-02,
  -6.76365972e-01],
 [ 5.92467785e-02,  1.84238404e-01, -6.78586289e-02,
  -4.01271731e-01]],

[[-3.24981436e-02,  3.74431983e-02,  4.19876389e-02,
   1.55467957e-01],
 [-3.17492783e-02,  2.31939614e-01,  4.50969972e-02,
  -1.23678796e-01]],

[[ 1.09093562e-01,  3.85903955e-01, -1.59644246e-01,
  -8.47153127e-01],
 [ 1.16811641e-01,  5.82801342e-01, -1.76587313e-01,
  -1.18547988e+00]],
```

```
   2.1294793e-01]],

[[ 6.98286593e-02,  1.69769317e-01,  1.24694780e-01,
   5.91452837e-01],
 [ 7.32240453e-02, -2.68574674e-02,  1.36523828e-01,
   9.20668304e-01]],

[[ 5.41221239e-02, -1.66779131e-01,  2.84687057e-02,
   6.48923576e-01],
 [ 5.07865399e-02,  2.79349126e-02,  4.14471775e-02,
   3.65339547e-01]],

[[ 3.61823440e-02,  2.24769205e-01,  2.47782208e-02,
   3.46326269e-02],
 [ 4.06777263e-02,  4.19527233e-01,  2.54708733e-02,
  -2.50130683e-01]],

[[ 5.12410924e-02,  5.63002706e-01,  1.20714054e-01,
  -6.45811185e-02],
 [ 6.25011474e-02,  3.66375446e-01,  1.19422428e-01,
   2.63617277e-01]],

[[-1.39689837e-02,  4.25297260e-01,  3.26676369e-02,
  -3.77562553e-01],
 [-5.46303904e-03,  2.29726940e-01,  2.51163840e-02,
  -7.47610182e-02]],

[[ 1.78565942e-02,  2.09436163e-01,  9.87184793e-03,
  -2.87680507e-01],
 [ 2.20453180e-02,  4.04415965e-01,  4.11823764e-03,
  -5.77233672e-01]],

[[ 7.05366209e-02,  5.76339781e-01, -9.01751369e-02,
  -1.03012431e+00],
 [ 8.20634216e-02,  7.72538364e-01, -1.10777624e-01,
  -1.34970200e+00]],

[[ 9.75141898e-02,  5.78968763e-01, -1.37771666e-01,
  -1.09362936e+00],
 [ 1.09093562e-01,  3.85903955e-01, -1.59644246e-01,
  -8.47153127e-01]],

[[-1.83418670e-04,  2.29030624e-01,  2.81359479e-02,
  -5.94001114e-02],
```

```
       [ 4.39719390e-03,  3.35167982e-02,  2.69479472e-02,
         2.42025435e-01]],

       [[ 3.61823440e-02,  2.24769205e-01,  2.47782208e-02,
         3.46326269e-02],
        [ 4.06777263e-02,  4.19527233e-01,  2.54708733e-02,
        -2.50130683e-01]],

       [[-3.99480164e-02, -1.93810657e-01,  1.29902095e-01,
         5.89090049e-01],
        [-4.38242294e-02, -7.24015990e-04,  1.41683891e-01,
         3.39984268e-01]],

       [[ 5.12410924e-02,  5.63002706e-01,  1.20714054e-01,
        -6.45811185e-02],
        [ 6.25011474e-02,  3.66375446e-01,  1.19422428e-01,
         2.63617277e-01]],

       [[ 4.51377183e-02,  2.47992296e-02,  8.41565058e-02,
         4.35636759e-01],
        [ 4.56337035e-02, -1.71406955e-01,  9.28692371e-02,
         7.53617287e-01]],

       [[-2.49387361e-02, -3.81330252e-01,  8.74444693e-02,
         7.11841464e-01],
        [-3.25653404e-02, -1.87520981e-01,  1.01681300e-01,
         4.47914243e-01]],

       [[ 4.49265689e-02, -1.40360268e-02, -3.80768180e-02,
        -3.82515267e-02],
        [ 4.46458459e-02,  1.81610674e-01, -3.88418473e-02,
        -3.42700750e-01]],

       [[ 4.49265689e-02, -1.40360268e-02, -3.80768180e-02,
        -3.82515267e-02],
        [ 4.46458459e-02,  1.81610674e-01, -3.88418473e-02,
        -3.42700750e-01]],

       [[ 9.75141898e-02,  5.78968763e-01, -1.37771666e-01,
        -1.09362936e+00],
        [ 1.09093562e-01,  3.85903955e-01, -1.59644246e-01,
        -8.47153127e-01]],

       [[ 1.54564362e-02,  6.15987420e-01,  4.15838957e-02,
```

```
  -5.72885811e-01],
 [ 2.77761854e-02,  4.20307875e-01,  3.01261805e-02,
  -2.67398030e-01]],

[[ 1.04432702e-01, -4.01968323e-02,  2.09584549e-01,
   1.23149288e+00],
 [ 4.49265689e-02, -1.40360268e-02, -3.80768180e-02,
  -3.82515267e-02]],

[[-4.57422249e-02,  4.29024458e-01,  5.42156883e-02,
  -4.60334748e-01],
 [-3.71617377e-02,  2.33179778e-01,  4.50089946e-02,
  -1.51067749e-01]],

[[ 3.43183689e-02,  1.42212827e-02, -1.30915595e-02,
   7.07521848e-03],
 [ 3.46027948e-02, -1.80710495e-01, -1.29500553e-02,
   2.95599014e-01]],

[[ 9.34026146e-04, -3.76391828e-01,  3.97616476e-02,
   6.00930810e-01],
 [-6.59381039e-03, -1.81848004e-01,  5.17802611e-02,
   3.21032554e-01]],

[[-4.90076914e-02,  4.18824106e-02,  4.77397144e-02,
   5.74813373e-02],
 [-4.81700450e-02, -1.53890386e-01,  4.88893427e-02,
   3.64836097e-01]],

[[ 6.25011474e-02,  3.66375446e-01,  1.19422428e-01,
   2.63617277e-01],
 [ 6.98286593e-02,  1.69769317e-01,  1.24694780e-01,
   5.91452837e-01]],

[[ 1.82793953e-03,  3.26816067e-02,  4.26501594e-02,
   2.60585248e-01],
 [ 2.48157163e-03,  2.27169588e-01,  4.78618629e-02,
  -1.83460899e-02]],

[[ 4.46458459e-02,  1.81610674e-01, -3.88418473e-02,
  -3.42700750e-01],
 [ 4.82780598e-02, -1.29377712e-02, -4.56958637e-02,
  -6.25148118e-02]],
```

```
[[-3.17492783e-02,  2.31939614e-01,  4.50969972e-02,
  -1.23678796e-01],
 [-2.71104854e-02,  4.26387429e-01,  4.26234230e-02,
  -4.01799977e-01]],

[[ 6.25011474e-02,  3.66375446e-01,  1.19422428e-01,
   2.63617277e-01],
 [ 6.98286593e-02,  1.69769317e-01,  1.24694780e-01,
   5.91452837e-01]],

[[ 1.16811641e-01,  5.82801342e-01, -1.76587313e-01,
  -1.18547988e+00],
 [ 1.28467664e-01,  7.79717982e-01, -2.00296909e-01,
  -1.52790868e+00]],

[[-3.99961583e-02,  3.84843946e-01,  1.50386006e-01,
  -1.47278413e-01],
 [-3.22992802e-02,  5.77528238e-01,  1.47440434e-01,
  -3.88995677e-01]],

[[ 5.41221239e-02, -1.66779131e-01,  2.84687057e-02,
   6.48923576e-01],
 [ 5.07865399e-02,  2.79349126e-02,  4.14471775e-02,
   3.65339547e-01]],

[[-5.46296611e-02,  4.33818512e-02,  5.22795990e-02,
   2.43732501e-02],
 [-5.37620261e-02,  2.37716600e-01,  5.27670644e-02,
  -2.51367420e-01]],

[[ 1.16811641e-01,  5.82801342e-01, -1.76587313e-01,
  -1.18547988e+00],
 [ 1.28467664e-01,  7.79717982e-01, -2.00296909e-01,
  -1.52790868e+00]],

[[-3.99480164e-02, -1.93810657e-01,  1.29902095e-01,
   5.89090049e-01],
 [-4.38242294e-02, -7.24015990e-04,  1.41683891e-01,
   3.39984268e-01]],

[[-3.99480164e-02, -1.93810657e-01,  1.29902095e-01,
   5.89090049e-01],
 [-4.38242294e-02, -7.24015990e-04,  1.41683891e-01,
```

```
          3.39984268e-01]],

        [[-5.16086295e-02, -1.51051655e-01,  4.62371409e-02,
          3.02122951e-01],
         [-5.46296611e-02,  4.33818512e-02,  5.22795990e-02,
          2.43732501e-02]],

        [[ 7.02496339e-03,  4.21573639e-01,  4.74949405e-02,
         -2.95552194e-01],
         [ 1.54564362e-02,  6.15987420e-01,  4.15838957e-02,
         -5.72885811e-01]],

        [[ 4.22055647e-02,  2.23202370e-02,  1.07941583e-01,
          4.91543025e-01],
         [ 4.26519699e-02,  2.15767264e-01,  1.17772445e-01,
          2.34734535e-01]],

        [[ 4.85234335e-02, -1.69285849e-01,  7.00470358e-02,
          7.05473423e-01],
         [ 4.51377183e-02,  2.47992296e-02,  8.41565058e-02,
          4.35636759e-01]],

        [[ 8.20634216e-02,  7.72538364e-01, -1.10777624e-01,
         -1.34970200e+00],
         [ 9.75141898e-02,  5.78968763e-01, -1.37771666e-01,
         -1.09362936e+00]],

        [[-1.31356176e-02,  5.73526740e-01,  1.38586536e-01,
         -2.99263060e-01],
         [-1.66508369e-03,  7.66429186e-01,  1.32601276e-01,
         -5.45228422e-01]],

        [[-2.49387361e-02, -3.81330252e-01,  8.74444693e-02,
          7.11841464e-01],
         [-3.25653404e-02, -1.87520981e-01,  1.01681300e-01,
          4.47914243e-01]]], dtype=float32)>, action=<tf.Tensor: shape=(64, 2), dtype=int64, numpy=
array([[0, 1],
       [0, 1],
       [0, 0],
       [1, 0],
       [1, 1],
       [1, 0],
       [0, 1],
```

```
 [1, 0],
 [1, 0],
 [1, 0],
 [1, 0],
 [0, 0],
 [1, 0],
 [1, 0],
 [0, 1],
 [0, 0],
 [0, 0],
 [1, 0],
 [0, 1],
 [0, 1],
 [1, 1],
 [1, 1],
 [0, 1],
 [1, 0],
 [1, 0],
 [0, 0],
 [0, 0],
 [1, 0],
 [1, 0],
 [0, 1],
 [0, 0],
 [1, 0],
 [1, 1],
 [0, 0],
 [0, 1],
 [1, 1],
 [1, 0],
 [1, 0],
 [0, 1],
 [0, 0],
 [0, 1],
 [0, 0],
 [0, 0],
 [1, 1],
 [0, 1],
 [0, 0],
 [1, 1],
 [0, 1],
 [1, 0],
 [0, 0],
 [1, 0],
```

```
            [1, 0],
            [1, 0],
            [1, 0],
            [1, 0],
            [1, 1],
            [1, 1],
            [1, 1],
            [1, 0],
            [1, 1],
            [1, 0],
            [0, 0],
            [1, 0],
            [1, 1]])>, policy_info=(), next_step_type=<tf.Tensor: shape=(64, 2), dtype=int32, numpy=
    array([[1, 1],
            [0, 1],
            [1, 1],
            [1, 1],
            [1, 1],
            [1, 1],
            [1, 1],
            [1, 1],
            [1, 2],
            [1, 1],
            [1, 1],
            [1, 1],
            [1, 1],
            [1, 1],
            [1, 1],
            [1, 1],
            [1, 1],
            [1, 1],
            [1, 1],
            [1, 1],
            [1, 1],
            [1, 1],
            [1, 1],
            [1, 1],
            [1, 1],
            [1, 1],
            [1, 1],
            [1, 1],
            [1, 1],
            [1, 1],
```

```
       [1, 1],
       [1, 1],
       [1, 1],
       [1, 1],
       [1, 1],
       [1, 1],
       [1, 1],
       [1, 1],
       [1, 1],
       [1, 1],
       [0, 1],
       [1, 1],
       [1, 1],
       [1, 1],
       [1, 1],
       [1, 1],
       [1, 1],
       [1, 1],
       [1, 1],
       [1, 1],
       [1, 1],
       [1, 2],
       [1, 1],
       [1, 1],
       [1, 1],
       [1, 2],
       [1, 1],
       [1, 1],
       [1, 1],
       [1, 1],
       [1, 1],
       [1, 1],
       [1, 1],
       [1, 1],
       [1, 1]], dtype=int32)>, reward=<tf.Tensor: shape=(64, 2), dtype=float32, numpy=
array([[1., 1.],
       [0., 1.],
       [1., 1.],
       [1., 1.],
       [1., 1.],
       [1., 1.],
       [1., 1.],
       [1., 1.],
       [1., 1.],
       [1., 1.],
       [1., 1.],
```

```
[1., 1.],
[1., 1.],
[1., 1.],
[1., 1.],
[1., 1.],
[1., 1.],
[1., 1.],
[1., 1.],
[1., 1.],
[1., 1.],
[1., 1.],
[1., 1.],
[1., 1.],
[1., 1.],
[1., 1.],
[1., 1.],
[1., 1.],
[1., 1.],
[1., 1.],
[1., 1.],
[1., 1.],
[1., 1.],
[1., 1.],
[1., 1.],
[1., 1.],
[1., 1.],
[1., 1.],
[1., 1.],
[1., 1.],
[1., 1.],
[0., 1.],
[1., 1.],
[1., 1.],
[1., 1.],
[1., 1.],
[1., 1.],
[1., 1.],
[1., 1.],
[1., 1.],
[1., 1.],
[1., 1.],
[1., 1.],
[1., 1.],
[1., 1.]
```

```
[1., 1.],
[1., 1.],
[1., 1.],
[1., 1.],
[1., 1.],
[1., 1.],
[1., 1.],
[1., 1.],
[1., 1.],
[1., 1.],
[1., 1.]], dtype=float32)>, discount=<tf.Tensor: shape=(64, 2), dtype=float32, numpy=
array([[1., 1.],
[1., 1.],
[1., 1.],
[1., 1.],
[1., 1.],
[1., 1.],
[1., 1.],
[1., 1.],
[1., 0.],
[1., 1.],
[1., 1.],
[1., 1.],
[1., 1.],
[1., 1.],
[1., 1.],
[1., 1.],
[1., 1.],
[1., 1.],
[1., 1.],
[1., 1.],
[1., 1.],
[1., 1.],
[1., 1.],
[1., 1.],
[1., 1.],
[1., 1.],
[1., 1.],
[1., 1.],
[1., 1.],
[1., 1.],
[1., 1.],
[1., 1.],
[1., 1.],
[1., 1.],
[1., 1.],
[1., 1.],
[1., 1.],
[1., 1.],
```

```
       [1., 1.],
       [1., 1.],
       [1., 1.],
       [1., 1.],
       [1., 1.],
       [1., 1.],
       [1., 1.],
       [1., 1.],
       [1., 1.],
       [1., 1.],
       [1., 1.],
       [1., 1.],
       [1., 1.],
       [1., 1.],
       [1., 1.],
       [1., 1.],
       [1., 1.],
       [1., 0.],
       [1., 1.],
       [1., 1.],
       [1., 1.],
       [1., 0.],
       [1., 1.],
       [1., 1.],
       [1., 1.],
       [1., 1.],
       [1., 1.],
       [1., 1.],
       [1., 1.],
       [1., 1.]], dtype=float32)>),
 BufferInfo(ids=<tf.Tensor: shape=(64, 2), dtype=int64, numpy=
array([[44, 45],
       [42, 43],
       [51, 52],
       [96, 97],
       [59, 60],
       [43, 44],
       [83, 84],
       [24, 25],
       [54, 55],
       [92, 93],
       [11, 12],
       [28, 29],
```

```
 [28, 29],
 [46, 47],
 [75, 76],
 [83, 84],
 [86, 87],
 [67, 68],
 [50, 51],
 [91, 92],
 [47, 48],
 [69, 70],
 [53, 54],
 [35, 36],
 [88, 89],
 [84, 85],
 [33, 34],
 [73, 74],
 [ 1,  2],
 [50, 51],
 [52, 53],
 [76, 77],
 [84, 85],
 [21, 22],
 [33, 34],
 [93, 94],
 [16, 17],
 [43, 44],
 [43, 44],
 [52, 53],
 [82, 83],
 [42, 43],
 [67, 68],
 [ 4,  5],
 [10, 11],
 [63, 64],
 [34, 35],
 [79, 80],
 [44, 45],
 [70, 71],
 [34, 35],
 [54, 55],
 [24, 25],
 [88, 89],
 [61, 62],
 [54, 55],
```

```
              [21, 22],
              [21, 22],
              [60, 61],
              [81, 82],
              [95, 96],
              [92, 93],
              [51, 52],
              [27, 28],
              [16, 17]])>, probabilities=<tf.Tensor: shape=(64,), dtype=float32, numpy=
       array([0.01010101, 0.01010101, 0.01010101, 0.01010101, 0.01010101,
              0.01010101, 0.01010101, 0.01010101, 0.01010101, 0.01010101,
```

Training the agent

Two things must happen during the training loop:

- collect data from the environment
- use that data to train the agent's neural network(s)

This example also periodicially evaluates the policy and prints the current score.

The following will take ~5 minutes to run.

```
#@test {"skip": true}
try:
  %%time
except:
  pass

# (Optional) Optimize by wrapping some of the code in a graph using TF function.
agent.train = common.function(agent.train)

# Reset the train step
agent.train_step_counter.assign(0)

# Evaluate the agent's policy once before training.
avg_return = compute_avg_return(eval_env, agent.policy, num_eval_episodes)
returns = [avg_return]

for _ in range(num_iterations):
```

```
for _ in range(num_iterations):

  # Collect a few steps using collect_policy and save to the replay buffer.
  for _ in range(collect_steps_per_iteration):
    collect_step(train_env, agent.collect_policy, replay_buffer)

  # Sample a batch of data from the buffer and update the agent's network.
  experience, unused_info = next(iterator)
  train_loss = agent.train(experience).loss

  step = agent.train_step_counter.numpy()

  if step % log_interval == 0:
    print('step = {0}: loss = {1}'.format(step, train_loss))

  if step % eval_interval == 0:
    avg_return = compute_avg_return(eval_env, agent.policy, num_eval_episodes)
    print('step = {0}: Average Return = {1}'.format(step, avg_return))
    returns.append(avg_return)
```

```
CPU times: user 2 µs, sys: 0 ns, total: 2 µs
Wall time: 5.72 µs
step = 200: loss = 10.717947959899902
step = 400: loss = 12.38393497467041
step = 600: loss = 18.387475967407227
step = 800: loss = 10.651362419128418
step = 1000: loss = 93.07606506347656
step = 1000: Average Return = 18.799999237060547
step = 1200: loss = 70.29846954345703
step = 1400: loss = 44.89759826660156
step = 1600: loss = 41.00116729736328
step = 1800: loss = 35.69490051269531
step = 2000: loss = 68.86432647705078
step = 2000: Average Return = 49.099998474121094
step = 2200: loss = 43.24755096435547
step = 2400: loss = 127.55516052246094
step = 2600: loss = 53.361778259277344
step = 2800: loss = 54.785972595214844
step = 3000: loss = 59.30897903442383
step = 3000: Average Return = 56.400001525878906
step = 3200: loss = 10.138701438903809
step = 3400: loss = 27.239858627319336
step = 3600: loss = 11.668193817138672
step = 3800: loss = 75.79299926757812
step = 4000: loss = 177.84156799316406
step = 4000: Average Return = 146.89999389648438
step = 4200: loss = 29.700939178466797
step = 4400: loss = 254.27378845214844
step = 4600: loss = 207.34375
step = 4800: loss = 129.4775848388672
step = 5000: loss = 116.14445495605469
step = 5000: Average Return = 155.1999969482422
step = 5200: loss = 230.46336364746094
step = 5400: loss = 184.0150146484375
step = 5600: loss = 185.78878784179688
step = 5800: loss = 607.2044677734375
step = 6000: loss = 454.7425537109375
step = 6000: Average Return = 168.39999389648438
step = 6200: loss = 16.692855834960938
step = 6400: loss = 125.4916000366211
step = 6600: loss = 18.222740173339844
step = 6800: loss = 17.445449829101562
step = 7000: loss = 44.06896209716797
```

```
step = 7000: Average Return = 143.1999969482422
step = 7200: loss = 135.5415496826172
step = 7400: loss = 35.517730712890625
step = 7600: loss = 18.934978485107422
step = 7800: loss = 312.8609619140625
step = 8000: loss = 17.805965423583984
step = 8000: Average Return = 166.1999969482422
step = 8200: loss = 686.18115234375
step = 8400: loss = 294.57421875
step = 8600: loss = 15.815781593322754
step = 8800: loss = 746.2320556640625
step = 9000: loss = 470.3125915527344
step = 9000: Average Return = 177.39999389648438
step = 9200: loss = 26.100887298583984
step = 9400: loss = 218.4998779296875
step = 9600: loss = 349.7186279296875
step = 9800: loss = 13.304303169250488
step = 10000: loss = 490.5411682128906
step = 10000: Average Return = 194.0
step = 10200: loss = 22.076244354248047
step = 10400: loss = 316.521240234375
step = 10600: loss = 25.956804275512695
step = 10800: loss = 645.314208984375
step = 11000: loss = 964.9403076171875
step = 11000: Average Return = 170.3000030517578
step = 11200: loss = 26.43313980102539
step = 11400: loss = 33.0954475402832
step = 11600: loss = 977.22607421875
step = 11800: loss = 539.7288208007812
step = 12000: loss = 1215.1888427734375
step = 12000: Average Return = 185.10000610351562
step = 12200: loss = 28.999265670776367
step = 12400: loss = 24.258453369140625
step = 12600: loss = 105.99031066894531
step = 12800: loss = 633.2265625
step = 13000: loss = 42.167396545410156
step = 13000: Average Return = 200.0
step = 13200: loss = 29.743854522705078
step = 13400: loss = 40.86017608642578
step = 13600: loss = 58.689605712890625
step = 13800: loss = 22.16793441772461
step = 14000: loss = 45.48065185546875
step = 14000: Average Return = 200.0
step = 14200: loss = 793.3546752929688
```

```
step = 14400: loss = 34.449127197265625
step = 14600: loss = 42.500152587890625
step = 14800: loss = 1465.037109375
step = 15000: loss = 50.96833419799805
step = 15000: Average Return = 192.8000030517578
step = 15200: loss = 26.043331146240234
step = 15400: loss = 38.60572814941406
step = 15600: loss = 40.13157653808594
step = 15800: loss = 44.0748291015625
step = 16000: loss = 74.1549072265625
step = 16000: Average Return = 199.0
step = 16200: loss = 38.570655822753906
step = 16400: loss = 1087.758544921875
step = 16600: loss = 69.68978118896484
```

Visualization

Plots

Use `matplotlib.pyplot` to chart how the policy improved during training.

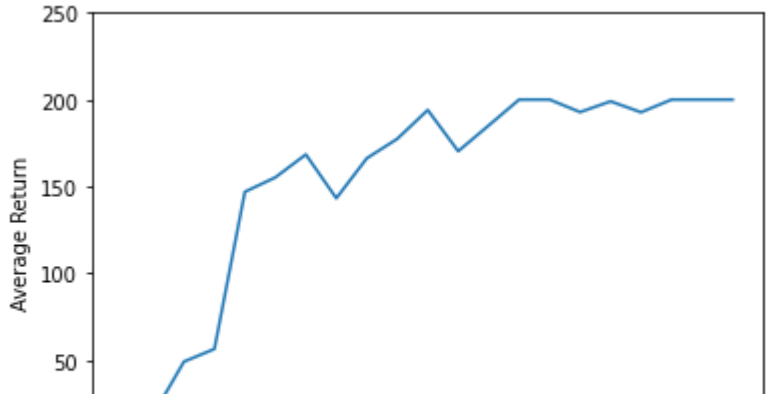
One iteration of `Cartpole-v0` consists of 200 time steps. The environment gives a reward of `+1` for each step the pole stays up, so the maximum return for one episode is 200. The charts shows the return increasing towards that maximum each time it is evaluated during training. (It may be a little unstable and not increase monotonically each time.)

```
step = 19000: Average Return = 200.0
```

```
#@test {"skip": true}

iterations = range(0, num_iterations + 1, eval_interval)
plt.plot(iterations, returns)
plt.ylabel('Average Return')
plt.xlabel('Iterations')
plt.ylim(top=250)
```

```
(-0.8650002002716075, 250.0)
```



Videos

Charts are nice. But more exciting is seeing an agent actually performing a task in an environment.

First, create a function to embed videos in the notebook.

```
def embed_mp4(filename):
  """Embeds an mp4 file in the notebook."""
  video = open(filename,'rb').read()
  b64 = base64.b64encode(video)
  tag = '''
  <video width="640" height="480" controls>
    <source src="data:video/mp4;base64,{0}" type="video/mp4">
  Your browser does not support the video tag.
  </video>'''.format(b64.decode())

  return IPython.display.HTML(tag)
```

Now iterate through a few episodes of the Cartpole game with the agent. The underlying Python environment (the one "inside" the TensorFlow environment wrapper) provides a `render()` method, which outputs an image of the environment state. These can be collected into a video.

```
def create_policy_eval_video(policy, filename, num_episodes=5, fps=30):
  filename = filename + ".mp4"
  with imageio.get_writer(filename, fps=fps) as video:
    for _ in range(num_episodes):
      time_step = eval_env.reset()
      video.append_data(eval_py_env.render())
      while not time_step.is_last():
        action_step = policy.action(time_step)
        time_step = eval_env.step(action_step.action)
        video.append_data(eval_py_env.render())
  return embed_mp4(filename)



create_policy_eval_video(agent.policy, "trained-agent")
```

```
WARNING:root:IMAGEIO FFMPEG_WRITER WARNING: input image is not divisible by macro_block_size=16, resizing from (400, 6
```

For fun, compare the trained agent (above) to an agent moving randomly. (It does not do as well.)

```
create_policy_eval_video(random_policy, "random-agent")
```

```
WARNING:root:IMAGEIO FFMPEG_WRITER WARNING: input image is not divisible by macro_block_size=16, resizing from (400, 6
```