In [0]:
```
import collections
import cv2
import numpy as np
import matplotlib.pyplot as plt
import gym


def plot_learning_curve(x, scores, epsilons, filename, lines=None):
    fig=plt.figure()
    ax=fig.add_subplot(111, label="1")
    ax2=fig.add_subplot(111, label="2", frame_on=False)

    ax.plot(x, epsilons, color="C0")
    ax.set_xlabel("Training Steps", color="C0")
    ax.set_ylabel("Epsilon", color="C0")
    ax.tick_params(axis='x', colors="C0")
    ax.tick_params(axis='y', colors="C0")

    N = len(scores)
    running_avg = np.empty(N)
    for t in range(N):
            running_avg[t] = np.mean(scores[max(0, t-20):(t+1)])

    ax2.scatter(x, running_avg, color="C1")
    ax2.axes.get_xaxis().set_visible(False)
    ax2.yaxis.tick_right()
    ax2.set_ylabel('Score', color="C1")
    ax2.yaxis.set_label_position('right')
    ax2.tick_params(axis='y', colors="C1")

    if lines is not None:
        for line in lines:
            plt.axvline(x=line)

    plt.savefig(filename)

class RepeatActionAndMaxFrame(gym.Wrapper):
    def __init__(self, env=None, repeat=4, clip_reward=False, no_ops=0,
                 fire_first=False):
        super(RepeatActionAndMaxFrame, self).__init__(env)
        self.repeat = repeat
        self.shape = env.observation_space.low.shape
        self.frame_buffer = np.zeros_like((2, self.shape))
        self.clip_reward = clip_reward
        self.no_ops = no_ops
        self.fire_first = fire_first

    def step(self, action):
        t_reward = 0.0
        done = False
        for i in range(self.repeat):
            obs, reward, done, info = self.env.step(action)
            if self.clip_reward:
                reward = np.clip(np.array([reward]), -1, 1)[0]
            t_reward += reward
            idx = i % 2
```

```
                self.frame_buffer[idx] = obs
                if done:
                    break

            max_frame = np.maximum(self.frame_buffer[0], self.frame_buffer[1])
            return max_frame, t_reward, done, info

        def reset(self):
            obs = self.env.reset()
            no_ops = np.random.randint(self.no_ops)+1 if self.no_ops > 0 else 0
            for _ in range(no_ops):
                _, _, done, _ = self.env.step(0)
                if done:
                    self.env.reset()
            if self.fire_first:
                assert self.env.unwrapped.get_action_meanings()[1] == 'FIRE'
                obs, _, _, _ = self.env.step(1)

            self.frame_buffer = np.zeros_like((2,self.shape))
            self.frame_buffer[0] = obs

            return obs

class PreprocessFrame(gym.ObservationWrapper):
    def __init__(self, shape, env=None):
        super(PreprocessFrame, self).__init__(env)
        self.shape = (shape[2], shape[0], shape[1])
        self.observation_space = gym.spaces.Box(low=0.0, high=1.0,
                                    shape=self.shape, dtype=np.float32)

    def observation(self, obs):
        new_frame = cv2.cvtColor(obs, cv2.COLOR_RGB2GRAY)
        resized_screen = cv2.resize(new_frame, self.shape[1:],
                                    interpolation=cv2.INTER_AREA)
        new_obs = np.array(resized_screen, dtype=np.uint8).reshape(self.shape)
        new_obs = new_obs / 255.0

        return new_obs

class StackFrames(gym.ObservationWrapper):
    def __init__(self, env, repeat):
        super(StackFrames, self).__init__(env)
        self.observation_space = gym.spaces.Box(
                            env.observation_space.low.repeat(repeat, axis=0),
                            env.observation_space.high.repeat(repeat, axis=0),
                            dtype=np.float32)
        self.stack = collections.deque(maxlen=repeat)

    def reset(self):
        self.stack.clear()
        observation = self.env.reset()
        for _ in range(self.stack.maxlen):
            self.stack.append(observation)

        return np.array(self.stack).reshape(self.observation_space.low.shape)

    def observation(self, observation):
        self.stack.append(observation)

        return np.array(self.stack).reshape(self.observation_space.low.shape)

def make_env(env_name, shape=(84,84,1), repeat=4, clip_rewards=False,
             no_ops=0, fire_first=False):
    env = gym.make(env_name)
    env = RepeatActionAndMaxFrame(env, repeat, clip_rewards, no_ops, fire_firs
t)
    env = PreprocessFrame(shape, env)
    env = StackFrames(env, repeat)

    return env
```

In [0]:

```
import numpy as np

class ReplayBuffer(object):
    def __init__(self, max_size, input_shape, n_actions):
        self.mem_size = max_size
        self.mem_cntr = 0
        self.state_memory = np.zeros((self.mem_size, *input_shape),
                                     dtype=np.float32)
        self.new_state_memory = np.zeros((self.mem_size, *input_shape),
                                         dtype=np.float32)

        self.action_memory = np.zeros(self.mem_size, dtype=np.int64)
        self.reward_memory = np.zeros(self.mem_size, dtype=np.float32)
        self.terminal_memory = np.zeros(self.mem_size, dtype=np.uint8)

    def store_transition(self, state, action, reward, state_, done):
        index = self.mem_cntr % self.mem_size
        self.state_memory[index] = state
        self.new_state_memory[index] = state_
        self.action_memory[index] = action
        self.reward_memory[index] = reward
        self.terminal_memory[index] = done
        self.mem_cntr += 1

    def sample_buffer(self, batch_size):
        max_mem = min(self.mem_cntr, self.mem_size)
        batch = np.random.choice(max_mem, batch_size, replace=False)

        states = self.state_memory[batch]
        actions = self.action_memory[batch]
        rewards = self.reward_memory[batch]
        states_ = self.new_state_memory[batch]
        terminal = self.terminal_memory[batch]

        return states, actions, rewards, states_, terminal
```

In [0]:
```
import tensorflow as tf
from tensorflow import keras
import os


class DuelingDeepQNetwork(keras.Model):
    def __init__(self,lr,n_actions,name,input_dims,chkpt_dir):
        super(DuelingDeepQNetwork, self).__init__()
        self.checkpoint_dir = chkpt_dir
        self.checkpoint_file = os.path.join(self.checkpoint_dir, name)
        self.conv1 = keras.layers.Conv2D(input_shape=(-1,*input_dims),
            filters=32,kernel_size=8,data_format='channels_first',strides=4,ac
tivation='relu')
        self.conv2 = keras.layers.Conv2D(filters=32,kernel_size=4,strides=2,da
ta_format='channels_first',
            activation='relu')
        self.conv3 = keras.layers.Conv2D(filters=64,kernel_size=3,strides=1,da
ta_format='channels_first',
            activation='relu')
        self.flatlayer = keras.layers.Flatten()
        self.dense1 = keras.layers.Dense(units=1024, activation='relu')
        self.dense2 = keras.layers.Dense(units=512, activation='relu')

        self.V = keras.layers.Dense(units=1, activation='relu')
        self.A = keras.layers.Dense(units=n_actions, activation='relu')
    def call(self, state):
        conv1 = self.conv1(state)
        conv2 = self.conv2(conv1)
        conv3 = self.conv3(conv2)
        flattened = self.flatlayer(conv3)
        dense1 = self.dense1(flattened)
        dense2 = self.dense2(dense1)
        V = self.V(dense2)
        A = self.A(dense2)

        return V, A

    def save_checkpoint(self):
      pass
      print('Saving weights...')
      self.save(self.checkpoint_file)

    def load_checkpoint(self):
      pass
      print('loading checkpoint...')
      self.load_weights(self.checkpoint_file)
```

In [0]:
```
class DuelingDQNAgent(object):
    def __init__(self, gamma, epsilon, lr, n_actions, input_dims,
                 mem_size, batch_size, eps_min=0.01, eps_dec=5e-7,
                 replace=1000, algo=None, env_name=None, chkpt_dir='tmp/dqn'):
        self.gamma = gamma
        self.epsilon = epsilon
        self.lr = lr
        self.n_actions = n_actions
        self.input_dims = input_dims
        self.batch_size = batch_size
        self.eps_min = eps_min
        self.eps_dec = eps_dec
        self.replace_target_cnt = replace
        self.algo = algo
        self.env_name = env_name
        self.chkpt_dir = chkpt_dir
        self.action_space = [i for i in range(n_actions)]
        self.learn_step_counter = 0

        self.memory = ReplayBuffer(mem_size, input_dims, n_actions)

        self.q_eval = DuelingDeepQNetwork(self.lr, self.n_actions,
                                  input_dims=self.input_dims,
                                  name=self.env_name+'_'+self.algo+'_q_eval'
,
                                  chkpt_dir=self.chkpt_dir)

        self.q_next = DuelingDeepQNetwork(self.lr, self.n_actions,
                                  input_dims=self.input_dims,
                                  name=self.env_name+'_'+self.algo+'_q_next'
,
                                  chkpt_dir=self.chkpt_dir)

    def choose_action(self, observation):
        if tf.random.uniform([1]) > self.epsilon:
          value, actions = self.q_eval.call(tf.expand_dims(observation, axis=0
))

          action = tf.argmax(actions, axis=1)
        else:
          action = np.random.choice(self.action_space)
        return action

    def store_transition(self, state, action, reward, state_, done):
        self.memory.store_transition(state,action,reward,state_,done)

    def sample_memory(self):
        states,actions,rewards,new_states,dones = self.memory.sample_buffer(se
lf.batch_size)
        return states, actions, rewards, new_states,dones

    def replace_target_network(self):
        if self.learn_step_counter & self.replace_target_cnt ==0:
            self.q_next = self.q_eval

    def decrement_epsilon(self):
        self.epsilon = self.epsilon - self.eps_dec \
                           if self.epsilon > self.eps_min else self.eps_min

    def save_models(self):
        self.q_eval.save_checkpoint()
        self.q_next.save_checkpoint()

    def load_models(self):
        self.q_eval.load_checkpoint()
        self.q_next.load_checkpoint()

    def learn(self):
        if self.memory.mem_cntr < self.batch_size:
            return

        self.replace_target_network()
        states,actions,rewards,states_,dones = self.sample_memory()
        optimizer = keras.optimizers.RMSprop(learning_rate=self.lr)
        indices = tf.range(self.batch_size)

        with tf.GradientTape() as tape:
            V_s, A_s = self.q_eval.call(states)
            V_s_, A_s_ = self.q_next.call(states_)
            q_pred = tf.add(V_s,(
                A_s - tf.reduce_mean(A_s, axis=1, keepdims=True)
            ))
            q_pred = tf.gather_nd(q_pred, list(zip(indices, actions)))

            q_next = tf.add(V_s_,(
                A_s_ - tf.reduce_mean(A_s_, axis=1, keepdims=True))
            )
            q_next = tf.reduce_max(q_next, axis=1) * tf.cast(dones, dtype=tf.f
loat32)
            q_target = tf.cast(rewards,dtype=tf.float32) + self.gamma * q_next

            loss = keras.losses.MSE(q_target, q_pred)
        gradient = tape.gradient(loss, self.q_eval.trainable_variables)
        optimizer.apply_gradients(zip(gradient, self.q_eval.trainable_variable
s))
        self.decrement_epsilon()
        self.learn_step_counter += 1
```

In [0]:
```
def main(num_games= 10,load_checkpoint=False, env_name='PongNoFrameskip-v4'):
    env  = make_env(env_name)
    best_score = -np.inf

    agent = DuelingDQNAgent(gamma=0.99, epsilon=1.0,lr=0.0001,input_dims=(env.
observation_space.shape),
                     n_actions=env.action_space.n, mem_size=20000, eps_min=0.1
, batch_size=32,replace=100,
                     eps_dec=1e-5, chkpt_dir='models/',algo='DuelingDQNAgent',
env_name=env_name)
    if load_checkpoint:
          agent.load_models()
    fname = agent.algo + '_' + agent.env_name + '_lr' + str(agent.lr) +'_' \
            + str(num_games) + 'games'
    figure_file = 'plots/' + fname + '.png'
    n_steps = 0
    scores, eps_history,steps_array = [], [], []

    for i in range(num_games):
          done = False
          observation = env.reset()
          score = 0
          while not done:
                action = agent.choose_action(observation)
                observation_,reward,done,info = env.step(action)
                score += reward
                if not load_checkpoint:
                      agent.store_transition(observation, action, reward,obser
vation_, int(done))
                      agent.learn()

                observation = observation_
          scores.append(score)
          steps_array.append(n_steps)
          avg_score = np.mean(scores[-100:])
          print('episode: ', i,'score: ', score,
             ' average score %.1f' % avg_score, 'best score %.2f' % best_score
,
            'epsilon %.2f' % agent.epsilon, 'steps', n_steps)

          if avg_score > best_score:

              best_score = avg_score

          eps_history.append(agent.epsilon)
          if load_checkpoint and n_steps >= 18000:
              break

    x = [i+1 for i in range(len(scores))]
    plot_learning_curve(steps_array, scores, eps_history, figure_file)
```

In [0]:
```
main(500)
```

```
/usr/local/lib/python3.6/dist-packages/gym/logger.py:30: UserWarning: WARN: B
ox bound precision lowered by casting to float32
  warnings.warn(colorize('%s: %s'%('WARN', msg % args), 'yellow'))

WARNING:tensorflow:Layer conv2d_6 is casting an input tensor from dtype float
64 to the layer's dtype of float32, which is new behavior in TensorFlow 2.  T
he layer has dtype float32 because it's dtype defaults to floatx.

If you intended to run this layer in float32, you can safely ignore this warn
ing. If in doubt, this warning is likely only an issue if you are porting a T
ensorFlow 1.X model to TensorFlow 2.

To change all layers to have dtype float64 by default, call `tf.keras.backen
d.set_floatx('float64')`. To change just this layer, pass dtype='float64' to
the layer constructor. If you are the author of this layer, you can disable a
utocasting by passing autocast=False to the base Layer constructor.


/usr/local/lib/python3.6/dist-packages/gym/envs/atari/atari_env.py:113: Futur
eWarning: Using a non-tuple sequence for multidimensional indexing is depreca
ted; use `arr[tuple(seq)]` instead of `arr[seq]`. In the future this will be
interpreted as an array index, `arr[np.array(seq)]`, which will result either
in an error or a different result.
  action = self._action_set[a]
```

```
episode:  0 score:  -20.0  average score -20.0 best score -inf epsilon 0.99 s
teps 0
episode:  1 score:  -20.0  average score -20.0 best score -20.00 epsilon 0.98
steps 0
episode:  2 score:  -21.0  average score -20.3 best score -20.00 epsilon 0.97
steps 0
episode:  3 score:  -21.0  average score -20.5 best score -20.00 epsilon 0.97
steps 0
episode:  4 score:  -20.0  average score -20.4 best score -20.00 epsilon 0.96
steps 0
episode:  5 score:  -20.0  average score -20.3 best score -20.00 epsilon 0.95
steps 0
episode:  6 score:  -20.0  average score -20.3 best score -20.00 epsilon 0.94
steps 0
episode:  7 score:  -20.0  average score -20.2 best score -20.00 epsilon 0.93
steps 0
episode:  8 score:  -20.0  average score -20.2 best score -20.00 epsilon 0.92
steps 0
episode:  9 score:  -21.0  average score -20.3 best score -20.00 epsilon 0.91
steps 0
episode:  10 score:  -21.0  average score -20.4 best score -20.00 epsilon 0.9
0 steps 0
episode:  11 score:  -21.0  average score -20.4 best score -20.00 epsilon 0.8
9 steps 0
episode:  12 score:  -20.0  average score -20.4 best score -20.00 epsilon 0.8
8 steps 0
episode:  13 score:  -21.0  average score -20.4 best score -20.00 epsilon 0.8
7 steps 0
episode:  14 score:  -20.0  average score -20.4 best score -20.00 epsilon 0.8
6 steps 0
episode:  15 score:  -21.0  average score -20.4 best score -20.00 epsilon 0.8
5 steps 0
episode:  16 score:  -20.0  average score -20.4 best score -20.00 epsilon 0.8
4 steps 0
episode:  17 score:  -20.0  average score -20.4 best score -20.00 epsilon 0.8
3 steps 0
episode:  18 score:  -21.0  average score -20.4 best score -20.00 epsilon 0.8
3 steps 0
episode:  19 score:  -20.0  average score -20.4 best score -20.00 epsilon 0.8
2 steps 0
episode:  20 score:  -20.0  average score -20.4 best score -20.00 epsilon 0.8
1 steps 0
episode:  21 score:  -21.0  average score -20.4 best score -20.00 epsilon 0.8
0 steps 0
episode:  22 score:  -19.0  average score -20.3 best score -20.00 epsilon 0.7
9 steps 0
episode:  23 score:  -21.0  average score -20.4 best score -20.00 epsilon 0.7
8 steps 0
episode:  24 score:  -20.0  average score -20.4 best score -20.00 epsilon 0.7
7 steps 0
episode:  25 score:  -19.0  average score -20.3 best score -20.00 epsilon 0.7
6 steps 0
episode:  26 score:  -21.0  average score -20.3 best score -20.00 epsilon 0.7
5 steps 0
episode:  27 score:  -21.0  average score -20.4 best score -20.00 epsilon 0.7
4 steps 0
episode:  28 score:  -19.0  average score -20.3 best score -20.00 epsilon 0.7
```

```
3 steps 0
episode:  29 score:  -20.0  average score -20.3 best score -20.00 epsilon 0.7
2 steps 0
episode:  30 score:  -20.0  average score -20.3 best score -20.00 epsilon 0.7
1 steps 0
episode:  31 score:  -20.0  average score -20.3 best score -20.00 epsilon 0.7
0 steps 0
episode:  32 score:  -21.0  average score -20.3 best score -20.00 epsilon 0.7
0 steps 0
episode:  33 score:  -21.0  average score -20.3 best score -20.00 epsilon 0.6
9 steps 0
episode:  34 score:  -21.0  average score -20.3 best score -20.00 epsilon 0.6
8 steps 0
episode:  35 score:  -21.0  average score -20.4 best score -20.00 epsilon 0.6
7 steps 0
episode:  36 score:  -19.0  average score -20.3 best score -20.00 epsilon 0.6
6 steps 0
episode:  37 score:  -21.0  average score -20.3 best score -20.00 epsilon 0.6
5 steps 0
episode:  38 score:  -21.0  average score -20.4 best score -20.00 epsilon 0.6
4 steps 0
episode:  39 score:  -21.0  average score -20.4 best score -20.00 epsilon 0.6
3 steps 0
episode:  40 score:  -21.0  average score -20.4 best score -20.00 epsilon 0.6
2 steps 0
episode:  41 score:  -21.0  average score -20.4 best score -20.00 epsilon 0.6
1 steps 0
episode:  42 score:  -21.0  average score -20.4 best score -20.00 epsilon 0.6
0 steps 0
episode:  43 score:  -21.0  average score -20.4 best score -20.00 epsilon 0.5
9 steps 0
episode:  44 score:  -19.0  average score -20.4 best score -20.00 epsilon 0.5
8 steps 0
episode:  45 score:  -21.0  average score -20.4 best score -20.00 epsilon 0.5
8 steps 0
episode:  46 score:  -21.0  average score -20.4 best score -20.00 epsilon 0.5
7 steps 0
episode:  47 score:  -21.0  average score -20.4 best score -20.00 epsilon 0.5
6 steps 0
episode:  48 score:  -20.0  average score -20.4 best score -20.00 epsilon 0.5
5 steps 0
episode:  49 score:  -20.0  average score -20.4 best score -20.00 epsilon 0.5
4 steps 0
episode:  50 score:  -21.0  average score -20.4 best score -20.00 epsilon 0.5
3 steps 0
episode:  51 score:  -21.0  average score -20.4 best score -20.00 epsilon 0.5
2 steps 0
episode:  52 score:  -20.0  average score -20.4 best score -20.00 epsilon 0.5
1 steps 0
episode:  53 score:  -21.0  average score -20.4 best score -20.00 epsilon 0.5
0 steps 0
episode:  54 score:  -21.0  average score -20.5 best score -20.00 epsilon 0.4
9 steps 0
episode:  55 score:  -20.0  average score -20.4 best score -20.00 epsilon 0.4
8 steps 0
episode:  56 score:  -21.0  average score -20.5 best score -20.00 epsilon 0.4
7 steps 0
```

```
episode:  57 score:  -21.0  average score -20.5 best score -20.00 epsilon 0.4
6 steps 0
episode:  58 score:  -21.0  average score -20.5 best score -20.00 epsilon 0.4
5 steps 0
episode:  59 score:  -21.0  average score -20.5 best score -20.00 epsilon 0.4
5 steps 0
episode:  60 score:  -19.0  average score -20.5 best score -20.00 epsilon 0.4
3 steps 0
episode:  61 score:  -21.0  average score -20.5 best score -20.00 epsilon 0.4
3 steps 0
episode:  62 score:  -21.0  average score -20.5 best score -20.00 epsilon 0.4
1 steps 0
episode:  63 score:  -21.0  average score -20.5 best score -20.00 epsilon 0.4
1 steps 0
episode:  64 score:  -21.0  average score -20.5 best score -20.00 epsilon 0.4
0 steps 0
episode:  65 score:  -21.0  average score -20.5 best score -20.00 epsilon 0.3
9 steps 0
episode:  66 score:  -21.0  average score -20.5 best score -20.00 epsilon 0.3
8 steps 0
episode:  67 score:  -21.0  average score -20.5 best score -20.00 epsilon 0.3
7 steps 0
episode:  68 score:  -20.0  average score -20.5 best score -20.00 epsilon 0.3
6 steps 0
episode:  69 score:  -20.0  average score -20.5 best score -20.00 epsilon 0.3
5 steps 0
episode:  70 score:  -21.0  average score -20.5 best score -20.00 epsilon 0.3
4 steps 0
episode:  71 score:  -21.0  average score -20.5 best score -20.00 epsilon 0.3
3 steps 0
episode:  72 score:  -21.0  average score -20.5 best score -20.00 epsilon 0.3
2 steps 0
episode:  73 score:  -21.0  average score -20.5 best score -20.00 epsilon 0.3
1 steps 0
episode:  74 score:  -21.0  average score -20.5 best score -20.00 epsilon 0.3
1 steps 0
episode:  75 score:  -21.0  average score -20.5 best score -20.00 epsilon 0.3
0 steps 0
episode:  76 score:  -21.0  average score -20.5 best score -20.00 epsilon 0.2
9 steps 0
episode:  77 score:  -21.0  average score -20.6 best score -20.00 epsilon 0.2
8 steps 0
episode:  78 score:  -21.0  average score -20.6 best score -20.00 epsilon 0.2
7 steps 0
episode:  79 score:  -21.0  average score -20.6 best score -20.00 epsilon 0.2
6 steps 0
episode:  80 score:  -21.0  average score -20.6 best score -20.00 epsilon 0.2
6 steps 0
episode:  81 score:  -21.0  average score -20.6 best score -20.00 epsilon 0.2
5 steps 0
episode:  82 score:  -21.0  average score -20.6 best score -20.00 epsilon 0.2
4 steps 0
episode:  83 score:  -21.0  average score -20.6 best score -20.00 epsilon 0.2
3 steps 0
episode:  84 score:  -21.0  average score -20.6 best score -20.00 epsilon 0.2
2 steps 0
episode:  85 score:  -21.0  average score -20.6 best score -20.00 epsilon 0.2
```

```
1 steps 0
episode:  86 score:  -21.0  average score -20.6 best score -20.00 epsilon 0.2
0 steps 0
episode:  87 score:  -21.0  average score -20.6 best score -20.00 epsilon 0.1
9 steps 0
episode:  88 score:  -21.0  average score -20.6 best score -20.00 epsilon 0.1
8 steps 0
```