Model dynamics

Just to make your waiting a bit more fun, our code saves the best model's weights. In the Deep Q Learning Agent /03_dqn_play.py file, we have a program which can load this model file and play one episode, displaying the model's dynamics.
The code is very simple, but seeing how several matrices, with a million parameters, play Pong with superhuman accuracy, by observing only the pixels can be like magic.

```
import gym
import time
import argparse
import numpy as np
import torch
from lib import wrappers
from lib import dqn_model
DEFAULT_ENV_NAME = "PongNoFrameskip-v4"
FPS = 25
```

In the beginning, we import the familiar PyTorch and Gym modules. The preceding FPS parameter specifies the approximate speed of the shown frames.

```
If __name__ == "__main__":
parser = argparse.ArgumentParser()
parser.add_argument("-m", "—model", required=True, help="Model file to load")
parser.add_argument("-e", "—env", default=DEFAULT_ENV_NAME, help="Environment
name to use, default=" +DEFAULT_ENV_NAME)
parser.add_argument("-r", "—record", help="Directory to store
video recording")
args = parser.parse_args()
```

The script accepts the filename of the saved model and allows the specification of the Gym environment (of course, the model and environment have to match).
Additionally, you can pass option -r with the name of a non existent directory, which will be used to save a video of your game (using the Monitor wrapper).
By default, the script just shows frames, but if you want to upload your model's gameplay to YouTube, for example, -r could be handy.

```
env = wrappers.make_env(args.env)
if args.record:
env = gym.wrappers.Monitor(env, args.record)
```

```
net = dqn_model.DQN(env.observation_space.shape,
env.action_space.n)
net.load_state_dict(torch.load(args.model))
```

The preceding code should be clear without comments: we create the environment and our model, then we load weights from the file passed in the arguments.

```
state = env.reset()
total_reward = 0.0
while True:
start_ts = time.time()
env.render()
state_v = torch.tensor(np.array([state], copy=False))
q_vals = net(state_v).data.numpy()[0]
action = np.argmax(q_vals)
```

This is almost an exact copy of Agent class' method play_step() from the training code, with the lack of epsilon-greedy action selection. We just pass our observation to the agent and select the action with maximum value. The only new thing here is the render() method in the environment, which is a standard way in Gym to display the current observation (you need to have a GUI for this).

```
state, reward, done, _ = env.step(action)
total_reward += reward
if done:
break
delta = 1/FPS - (time.time() - start_ts)
if delta > 0:
time.sleep(delta)
print("Total reward: %.2f" % total_reward)
```

The rest of the code is also simple. We pass the action to the environment, count the total reward, and stop our loop when the episode ends.