Hands-On: An introduction to Reinforcement Learning and Smarties

Computational **S**cience and **E**ngineering **Lab**
ETH Zürich

Recap Ex. 1 & 2: SARSA, REINFORCE and (CMA-)ES

Tuesday

SARSA

Initialize $\mathrm{w}^{(0)}$
for iteration i in 1, 2, ...

- Collect E episodes by executing $\pi_{\mathrm{w}^{(i)}}$
  - max entropy: $\pi_{\mathrm{w}^{(i)}}(a \mid s) \propto \exp\left[Q_{\mathrm{w}^{(i)}}(s, a)\right]$
  - ϵ-greedy
- # of collected steps (i.e. actions) may be $N = TE$
- For each step t, compute TD target:

$$\hat{q}_t^{\pi_{\mathrm{w}^{(i)}}} = r_{t+1} + \gamma Q_{\mathrm{w}^{(i)}}(s_{t+1}, a_{t+1})$$

- Update $\mathrm{w}^{(i)}$ by SGD by minimizing loss:

$$\mathscr{L}^{MSE}(\mathrm{w}^{(i)}) = \frac{1}{N} \sum_{t=0}^{N} \left[\frac{1}{2}\left[\hat{q}_t^{\pi_{\mathrm{w}^{(i)}}} - Q_{\mathrm{w}^{(i)}}(s_t, a_t)\right]^2\right]$$

Wednesday

REINFORCE

Initialize $\mathrm{w}^{(0)}$ and baseline $b^{(0)} = 0$
for iteration i in 1, 2, ..., N

- Collect E episodes by executing $\pi_{\mathrm{w}^{(i-1)}}$
- For time step t in each episode compute:

$$\hat{q}_t^{\pi_{\mathrm{w}}} = \Sigma_{t'=t}^{T-1} \gamma^{t'-t} r_{t'+1}$$

- Update: $b^{(i)} = \frac{1}{E \cdot T} \sum_{t=1}^{E \cdot T} \hat{q}_t^{\pi_{\mathrm{w}}}$
- Update $\mathrm{w}^{(i)}$ by SGD by minimizing loss:

$$\mathscr{L}^{PG}\left(\mathrm{w}^{(i-1)}\right) = -\frac{1}{E \cdot T} \sum_{t=1}^{E \cdot T} \left[\hat{q}_t^{\pi_{\mathrm{w}}} - b^{(i-1)}\right] \log \pi_{\mathrm{w}^{(i-1)}}(a_t \mid s_t)$$

$$\mathrm{w}^{(i)} = \mathrm{w}^{(i-1)} - c\, \nabla_{\mathrm{w}^{(i-1)}} \mathscr{L}^{PG}\left(\mathrm{w}^{(i-1)}\right)$$

ES and CMA-ES algorithm for policy optimisation

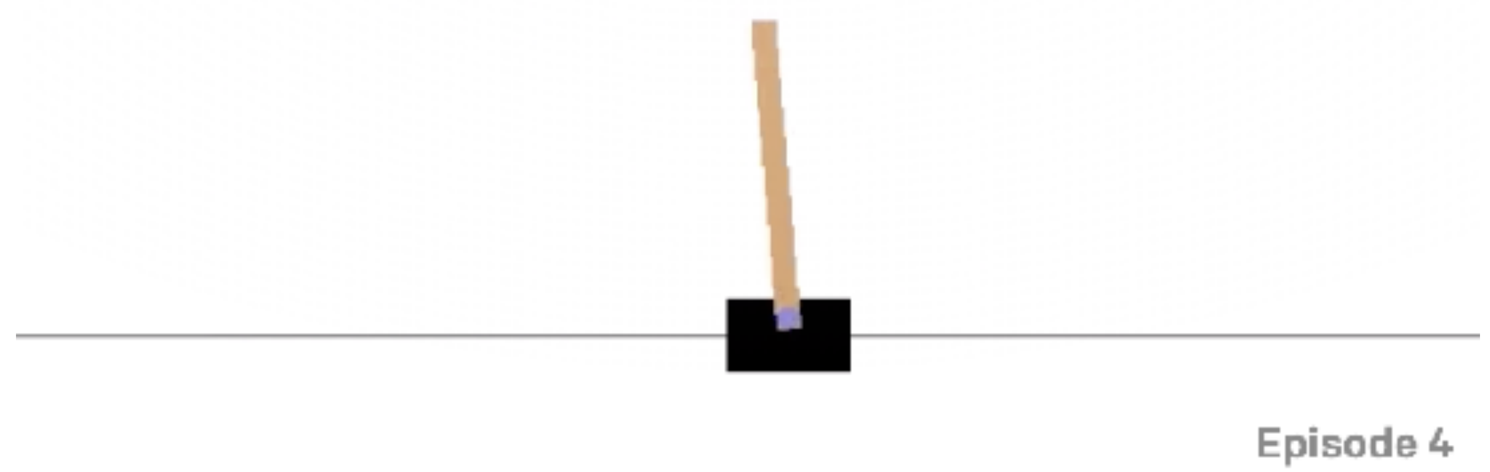
python PYTORCH

Action

State
Reward
Done
Info

Gym

```
# Get number of (observable) states.
env.observation_space.shape[0]

# Get number of available actions.
env.action_space.n

# Reset the environment to initial state.
env.reset()

# Perfom one action and observe reward and new state.
state, reward, done, info = env.step(action)
```



Today Ex. 3: Use smarties and build your own env.

smarties

Action

State
Reward
Done

Setup & Coupling

```cpp
inline void app_main(smarties::Communicator*const comm, int argc, char**argv)
{
  const int control_vars = 1; // force along x
  const int state_vars = 6;   // x, y, angvel, angle, cosine, sine
  comm->setStateActionDims(state_vars, control_vars);

  bool bounded = true;
  std::vector<double> upper_action_bound{10}, lower_action_bound{-10};
  comm->setActionScales(upper_action_bound, lower_action_bound, bounded);

  std::vector<bool> b_observable = {true, true, true, false, true, true};
  comm->setStateObservable(b_observable);

  std::vector<double> upper_state_bound{ 1,  1,  1,  1,  1,  1};
  std::vector<double> lower_state_bound{-1, -1, -1, -1, -1, -1};
  comm->setStateScales(upper_state_bound, lower_state_bound);

  CartPole env;

  while(true) // train loop
  {
    env.reset(comm->getPRNG());
    comm->sendInitState(env.getState());

    while (true) //simulation loop
    {
      std::vector<double> action = comm->recvAction();
      if(comm->terminateTraining()) return; // terminate

      bool poleFallen = env.advance(action); // advance

      std::vector<double> state = env.getState();
      double reward = env.getReward();

      if(poleFallen) { //tell smarties that this is a terminal state
        comm->sendTermState(state, reward);
        break;
      } else comm->sendState(state, reward);
    }
  }
}
```

Environment

```cpp
struct CartPole
{
  const double mp = 0.1;
  const double mc = 1;
  const double l = 0.5;
  const double g = 9.81;
  const double dt = 4e-4;
  const int nsteps = 50;

  int step=0;
  double F=0, t=0;
  Vec4 u;

  void reset(std::mt19937& gen)
  {
    ...
  }

  int advance(std::vector<double> action)
  {
    ...
  }

  ...
}
```

SMARTIES installation

1. Get smarties here: https://gitlab.ethz.ch/mavt-cse/smarties
2. **Follow** the installation procedure

3. Get skeleton code here:

https://gitlab.ethz.ch/mavt-cse/RL_retreat/tree/master/day-5-skeleton

4. In directory `cart_pole`, build with `make` and test your setup:

```
$ ./main
$ smarties_plot_rew.py .
```

generates output files in current directory

visualize reward with smarties helper file from ${SMARTIES_ROOT}/bin

```
================================================================================
              Continuous-action V-RACER with Gaussian policy
================================================================================
Experience Replay storage: remove most 'off policy' episode if and only if policy is better.
Experience Replay sampling algorithm: uniform probability.
    Single net with outputs: [0] : V(s),
                             [1 2] : policy mean and stdev,
    Size per entry = [1 1 1].
Initializing net approximator.
Layers composition:
(0) Input Layer of size:1
(1) SoftSign InnerProduct Layer of size:128 linked to Layer:0 of size:1
(2) SoftSign InnerProduct Layer of size:128 linked to Layer:1 of size:128
(3) Parametric Residual Connection of size:128
(4) Linear output InnerProduct Layer of size:2 linked to Layer:3 of size:128
(5) Parameter Layer of size:1. Initialized: -2.064742
Optimizer: Parameter updates using Adam SGD algorithm.
Collected 95% of data required to begin training. Initial reward std 1.209108e-06
ID #/T   |  avgR   | avgr | stdr | DKL | nEp |  nObs | totEp | totObs | oldEp |nDel|nFarP | beta | net    | RMSE | avgQ | stdQ | minQ | maxQ | polC | penC | proj | dAdv
00 00001  99.90000  99.90 0.0000 0.000 263229  263229  263229   263229  262144    0      0 0.0478 15.2111 0.0000 -0.000 0.0000 -0.000  0.000 0.0000 0.0000 -0.000 0.0000
00 00002  99.90000  99.90 0.0000 0.000 264229  264229  264229   264229  262144    0      0 0.0931 15.2111 0.0000  0.000 0.0000 -0.000  0.000 0.0001 0.0000  0.000 0.0000
```

Part I: Setup & Coupling

`main.cpp`

Core RL Functions

```
void sendInitState(const std::vector<double>& state)

void sendState(const std::vector<double>& state, const double reward)

void sendTermState(const std::vector<double>& state, const double reward)
```

Environment specification

specify number of dimensions of Cart Pole state

```
void setStateActionDims(const int dimState, const int dimAct)

void setActionScales(const std::vector<double> upper, const std::vector<double> lower, std::vector<bool> bound)

void setStateObservable(const std::vector<bool> observable)

void setStateScales(const std::vector<double> upper, const std::vector<double> lower)
```

length of argument vectors must match `dimState` or `dimAct`

Part II: Environment

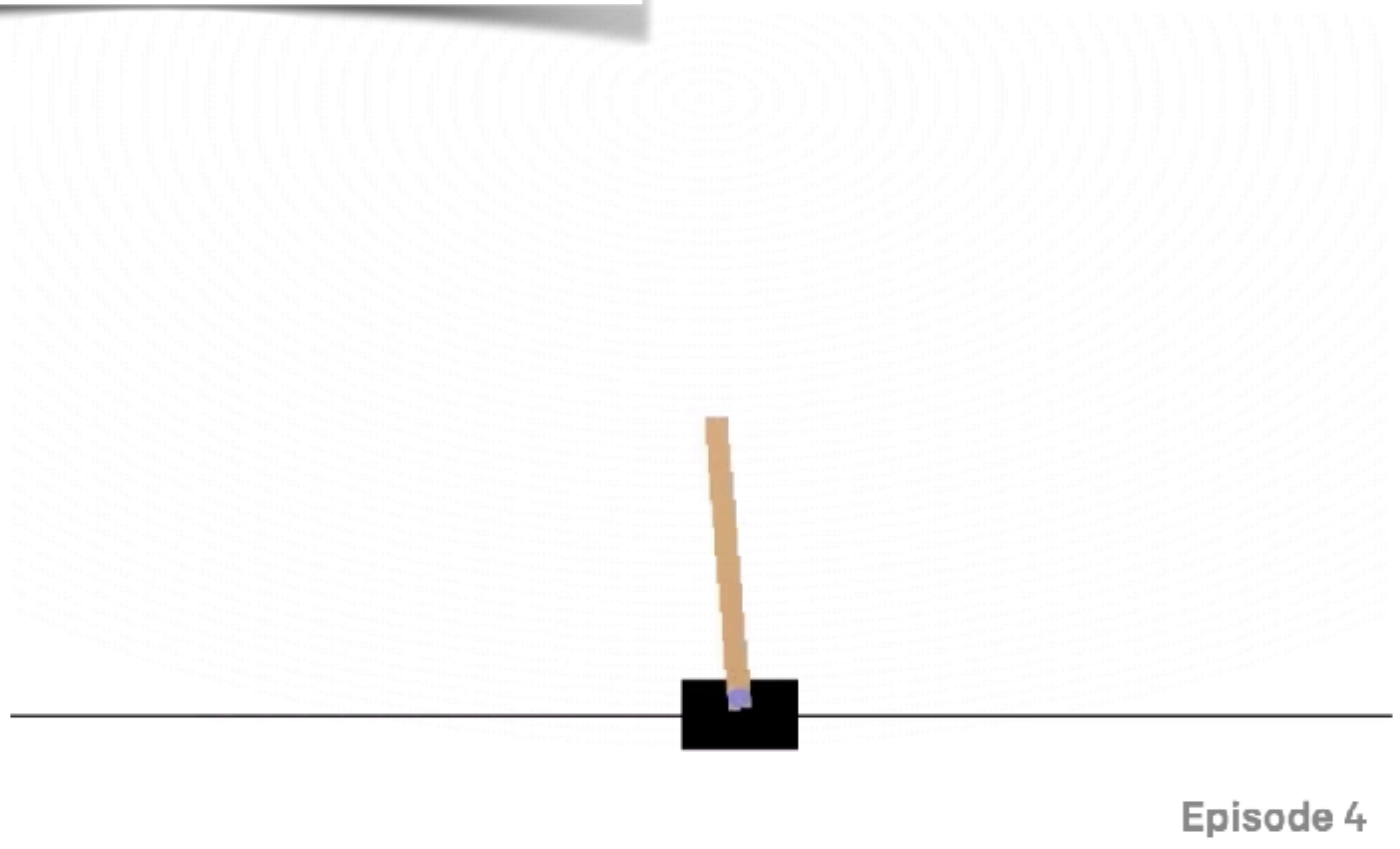
`cart-pole.hpp`

"Interesting" Environment Functions:

some of them may be used in `main.cpp`

```
void reset(std::mt19937& gen)

bool hasFailed()

bool isOver()

bool advance(const std::vector<double>& action)

double getReward()

std::vector<double> getState()
```

to implement



Physics of "hanging" Cart Pole

http://www.matthewpeterkelly.com/tutorials/cartPole/cartPoleEqns.pdf