Reinforcement Learning

Building a Complete RL System

Arrasy Rahman, Filippos Christianos, Lukas Schäfer

7 February 2020

THE UNIVERSITY of EDINBURGH
informatics

Lecture Outline

- What is Gym?
- How to implement your own environment?
- How to implement a RL algorithm?
- How to evaluate your results?
- Demonstration

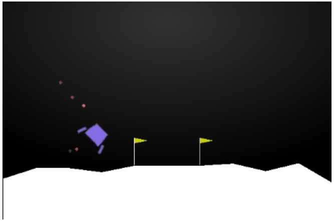
OpenAI Gym

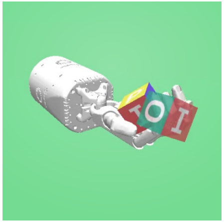
What is Gym? (Brockman et al., 2016)

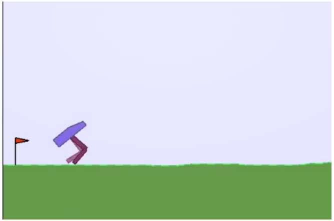
- Open source interface for sequential decision processes
- Developed and maintained by OpenAI Research Lab
- Collection of RL environments
- Standardised interface for RL environments

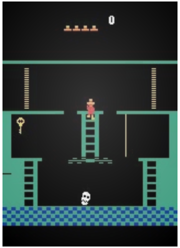
```
pip install gym
```

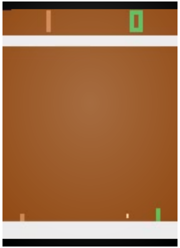
Lots of Interesting Environments! (Brockman et al., 2016)



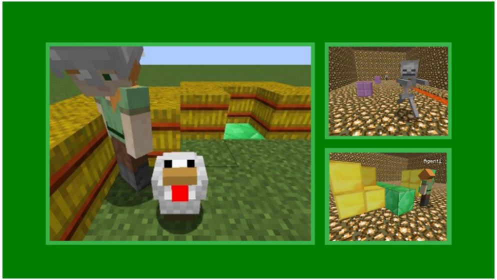








And many more... (Vinyals et al., 2017; Johnson et al., 2016; Kauten, 2018)



Gym Interface

- `gym.make(<environment_name>)` $\longrightarrow$ gym environment
  Create a gym environment
- `env.reset()` $\longrightarrow$ observation
  Resets environment for a new episode
- `env.step(action)` $\longrightarrow$ observation, reward, done, info
  Take an action and observe new information
- `env.render()`
  Render a visualisation of the current environmental state
- `env.close()`
  Close created environment

Gym Example Snippet

Gym control flow

```python
env = gym.make('CartPole-v0')
obs = env.reset()
done = False
while not done:
    env.render()
    action = agent.choose_action(obs)
    next_obs, reward, done, info = env.step(action)
    obs = next_obs
env.close()
```

Example: FrozenLake Environment

- Gridworld with 4×4 map
- **S** - starting cell | **G** - goal cell
  F - frozen cell | **H** - holes

Example: FrozenLake Environment

- Gridworld with 4×4 map
- **S** - starting cell | **G** - goal cell
  F - frozen cell | **H** - holes
- Observations:
  0 1 2 3
  4 5 6 7
  ...

Example: FrozenLake Environment

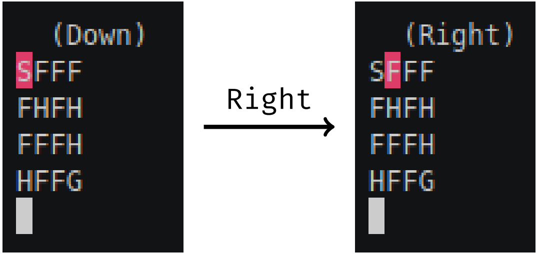
- Gridworld with 4×4 map
- **S** - starting cell | **G** - goal cell
  F - frozen cell | **H** - holes
- Observations:
  0 1 2 3
  4 5 6 7
  ...
- Actions:
  0 - `Left` 1 - `Down`
  2 - `Right` 3 - `Up`

Example: FrozenLake Environment

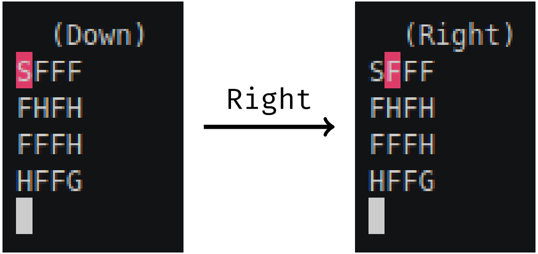
- Goal: reach **G** without falling into holes
- Reward: $+1$ for **G**, 0 otherwise
- Challenge: **F** are slippery! $\longrightarrow$ chance of moving in neighboured directions

- Gridworld with 4×4 map
- **S** - starting cell | **G** - goal cell
  F - frozen cell | **H** - holes
- Observations:
  0 1 2 3
  4 5 6 7
  ...
- Actions:
  0 - `Left` 1 - `Down`
  2 - `Right` 3 - `Up`

FrozenLake Environment Steps

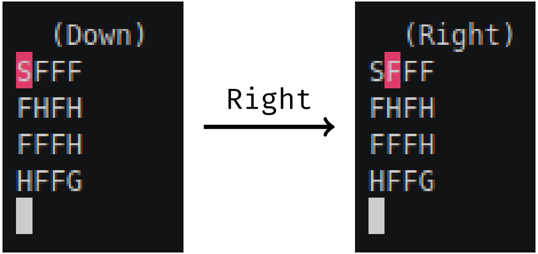


FrozenLake Environment Steps



`nobs, r, done, = env.step(a)` :

$$\mathtt{o} = 0 \xrightarrow{\mathtt{a}=2\ \ \mathtt{(Right)}} \langle \mathtt{nobs} = 1, \mathtt{r} = 0.0, \mathtt{done} = \mathtt{False} \rangle$$

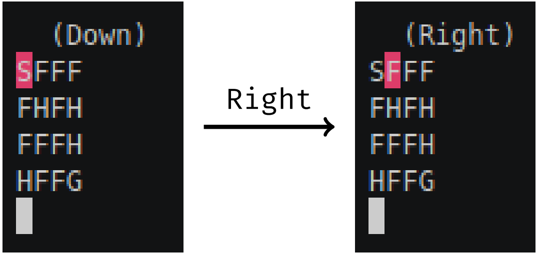


Environment dynamics: $p(o', r|o, a)$

`nobs, r, done, = env.step(a)` :

$$\texttt{o} = 0 \xrightarrow{\texttt{a}=2\ \texttt{(Right)}} \langle \texttt{nobs} = 1, \texttt{r} = 0.0, \texttt{done} = \texttt{False} \rangle$$

FrozenLake Environment Steps



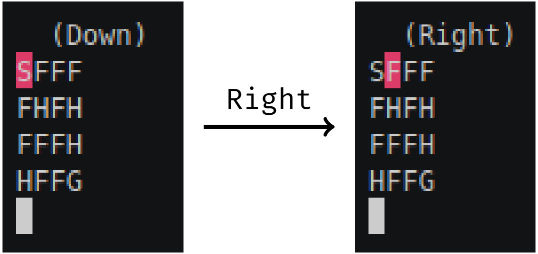
Environment dynamics: $p(o', r|o, a)$

- $p(1, 0.0|0, 2) = \frac{1}{3}$ (right)

`nobs, r, done, = env.step(a)`:

$$\texttt{o} = 0 \xrightarrow{\texttt{a}=2\ \ \texttt{(Right)}} \langle \texttt{nobs} = 1, \texttt{r} = 0.0, \texttt{done} = \texttt{False} \rangle$$

FrozenLake Environment Steps



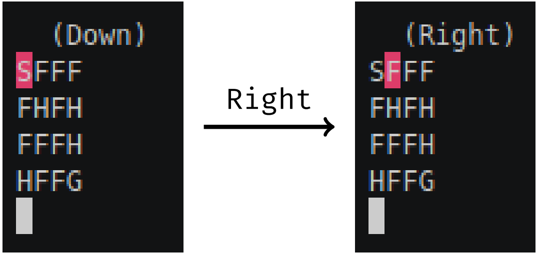
Environment dynamics: $p(o', r | o, a)$

- $p(1, 0.0|0, 2) = \frac{1}{3}$ (right)
- $p(0, 0.0|0, 2) = \frac{1}{3}$ (up)
- $p(4, 0.0|0, 2) = \frac{1}{3}$ (down)

`nobs, r, done, = env.step(a)` :

$$\mathtt{o} = 0 \xrightarrow{\mathtt{a}=2\ \ \mathtt{(Right)}} \langle \mathtt{nobs} = 1, \mathtt{r} = 0.0, \mathtt{done} = \mathtt{False} \rangle$$

FrozenLake Environment Steps



Environment dynamics: $p(o', r|o, a)$

- $p(1, 0.0|0, 2) = \frac{1}{3}$ (right)
- $p(0, 0.0|0, 2) = \frac{1}{3}$ (up)
- $p(4, 0.0|0, 2) = \frac{1}{3}$ (down)
- $p(o', r|0, 2) = 0$ for all other transitions

`nobs, r, done, = env.step(a)`:

$$\texttt{o} \ = 0 \xrightarrow{\texttt{a}=2 \ \ \texttt{(Right)}} \langle \texttt{nobs} = 1, \texttt{r} = 0.0, \texttt{done} = \texttt{False} \rangle$$

Implement your RL Agent

Recap: SARSA

On-Policy TD Control: Sarsa

$\longrightarrow$ learn q_π and improve π while following π

Updates: $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)\right]$

Exploration: ϵ-soft policy π

Recap: SARSA

On-Policy TD Control: Sarsa

$\longrightarrow$ learn q_π and improve π while following π

Updates: $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)\right]$

Exploration: ϵ-soft policy π

```
Initialize Q(s, a), ∀s ∈ S, a ∈ A(s), arbitrarily, and Q(terminal-state, ·) = 0
Repeat (for each episode):
    Initialize S
    Choose A from S using policy derived from Q (e.g., ε-greedy)
    Repeat (for each step of episode):
        Take action A, observe R, S′
        Choose A′ from S′ using policy derived from Q (e.g., ε-greedy)
        Q(S, A) ← Q(S, A) + α[R + γQ(S′, A′) − Q(S, A)]
        S ← S′; A ← A′;
    until S is terminal
```

SARSA Agent Class Structure

- `__init__` Initialise agent and Q-table as dictionary mapping `(obs, act) -> q-val`
- **`act`**: Epsilon-soft policy
- **`learn`**: Update Q-table given new experience

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t) \right]$$

- **`schedule_hyperparameters`**: Update hyperparameters given training progress

And now in Code ... act

Epsilon-soft Action Selection

```
def act(self, obs):
    act_vals = [self.q_table[(obs, act)] for act in range(self.
   n_acts)]
    max_val = max(act_vals)
    max_acts = [idx for idx, act_val in enumerate(act_vals) if
   act_val == max_val]

    if random.random() < self.epsilon:
        return random.randint(0, self.n_acts - 1)
    else:
        return random.choice(max_acts)
```

And now in Code ... learn

SARSA Q-Update

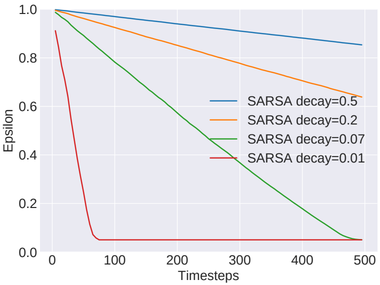
```
def learn(self, obs, action, reward, n_obs, n_action, done):
    target_value = reward + self.gamma * (1 - done) * self.
  q_table[(n_obs, n_action)]
    self.q_table[(obs, action)] += self.alpha * (
        target_value - self.q_table[(obs, action)]
    )
    return self.q_table[(obs, action)]
```

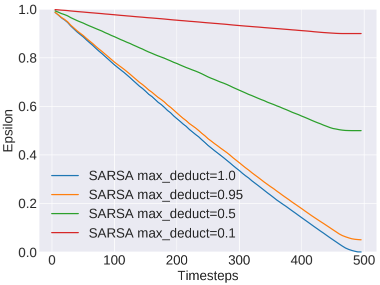
$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)\right]$$

And now in Code ... schedule_hyperparameters

SARSA ϵ-Scheduling

```
def schedule_hyperparameters(self, timestep, max_timestep):
    max_deduct, decay = 0.95, 0.07
    self.epsilon = 1.0 - (min(1.0, timestep/(decay *
  max_timestep))) * max_deduct
```





Evaluate your Results

Why do We Evaluate in the First Place?

- It gives our approach credibility
- Empirical evaluation is no proof, but can give strong indication about the strengths and limitations of an approach (when done right!)

Why do We Evaluate in the First Place?

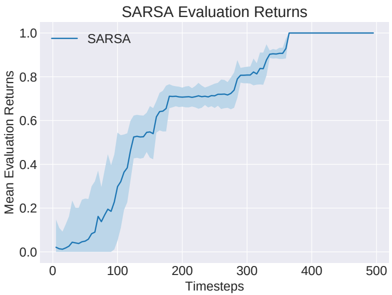
- It gives our approach credibility
- Empirical evaluation is no proof, but can give strong indication about the strengths and limitations of an approach (when done right!)

How to do it *right*?

What to Evaluate?

Evaluation Returns

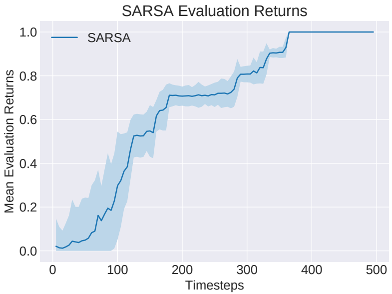
- Plot mean returns over multiple runs
- Visualise standard deviation or variance



What to Evaluate?

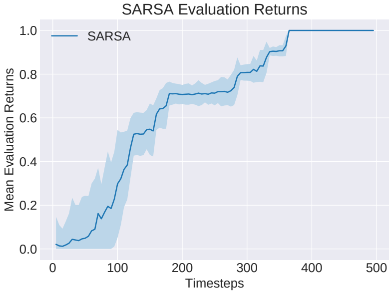
Evaluation Returns

- Plot mean returns over multiple runs
- Visualise standard deviation or variance



Which returns do we plot?

What to Evaluate?



Evaluation Returns

- Plot mean returns over multiple runs
- Visualise standard deviation or variance

Which returns do we plot?

- Execute multiple evaluation runs with $\epsilon = 0$ at fixed intervals
- Evaluation does not involve any learning!

Keep Track of Everything!

Unfrozen FrozenLake Policy

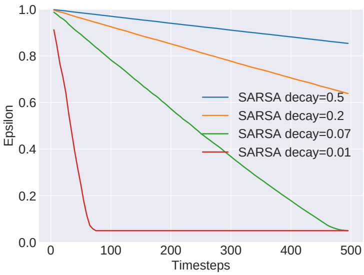
```
SFFF      RRDL
FHFH      DLDL
FFFH      RRDL
HFFG      LRRL
```

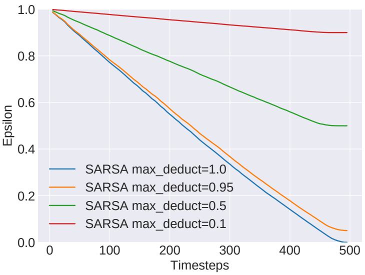
Keep Track of Everything!

Unfrozen FrozenLake	Policy
SFFF	RRDL
FHFH	DLDL
FFFH	RRDL
HFFG	LRRL

Hyperparameters

- Track hyperparameters, here ϵ-decay
- Try various values in a grid- or random-search and find good configuration





SARSA Gridsearch over Learning Rate α for (Frozen)Lake

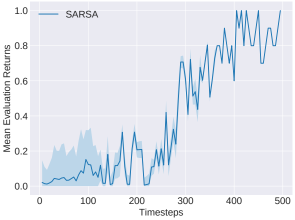


Figure 1: $\alpha = 0.9$

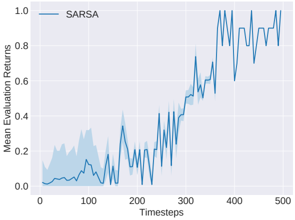


Figure 3: $\alpha = 0.7$

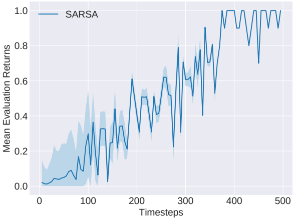


Figure 5: $\alpha = 0.5$

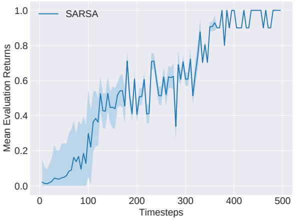


Figure 2: $\alpha = 0.3$

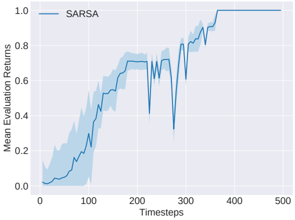


Figure 4: $\alpha = 0.2$

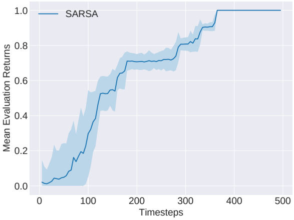


Figure 6: $\alpha = 0.1$

Common Pitfalls (1)

”But it worked last time!”

Common Pitfalls (1)

”But it worked last time!”

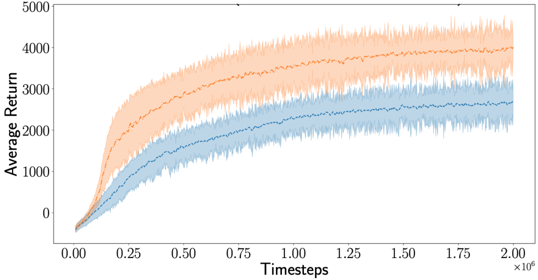
- It’s not enough to make it work once!
- Meaningful evaluation achieves consistent performance over multiple randomised runs
- Most RL algorithms have random components (e.g. ϵ-greedy policies)

Common Pifalls (2) (Henderson, 2018; Colas et al., 2019)

Is plotting the mean return, even with variance, enough?

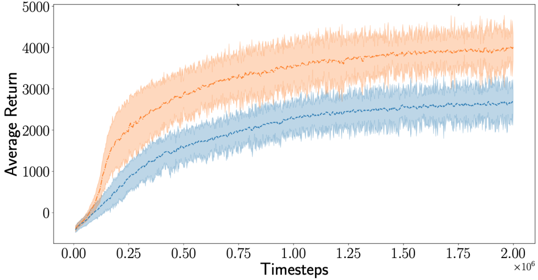
Common Pifalls (2) (Henderson, 2018; Colas et al., 2019)

Is plotting the mean return, even with variance, enough?



Common Pifalls (2) (Henderson, 2018; Colas et al., 2019)

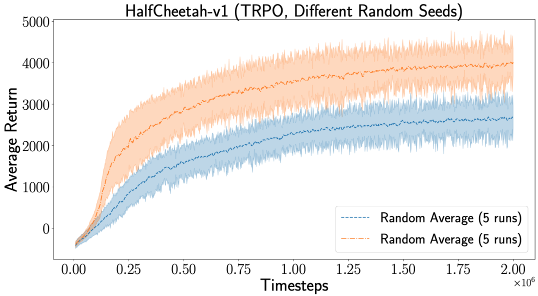
Is plotting the mean return, even with variance, enough?



Which one is better?

Common Pifalls (2) (Henderson, 2018; Colas et al., 2019)

Is plotting the mean return, even with variance, enough?

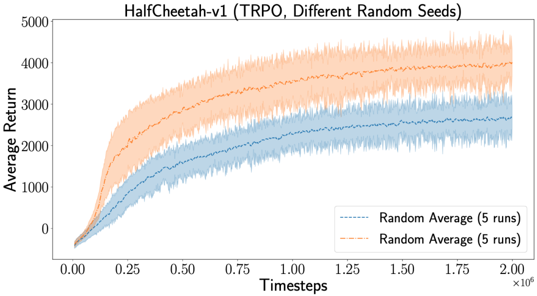


Which one is better?

It's actually the same method!

Common Pifalls (2) (Henderson, 2018; Colas et al., 2019)

Is plotting the mean return, even with variance, enough?



Which one is better?

It's actually the same method!

Apparently, it's not enough! $\longrightarrow$ Statistical hypothesis testing (Colas et al., 2019)

Common Pitfalls (3) (Bishop, 2019)

Common Pitfalls (3) (Bishop, 2019)

”Why should I use those random seeds? `random` already delivers random values!”

Common Pitfalls (3) (Bishop, 2019)

"Why should I use those random seeds? `random` already delivers random values!"

- Our goal with empirical evaluations is to make meaningful claims about the implemented approach and achieve **reproducible** performance
- Random seeds allow us to fixate random behaviour
- Reproducibility is key for meaningful research

Common Pitfalls (3) (Bishop, 2019)

”Why should I use those random seeds? `random` already delivers random values!”

- Our goal with empirical evaluations is to make meaningful claims about the implemented approach and achieve **reproducible** performance
- Random seeds allow us to fixate random behaviour
- Reproducibility is key for meaningful research

But NEVER choose/ tune your random seeds!

Common Pitfalls (3) (Bishop, 2019)

”Why should I use those random seeds? `random` already delivers random values!”

- Our goal with empirical evaluations is to make meaningful claims about the implemented approach and achieve **reproducible** performance
- Random seeds allow us to fixate random behaviour
- Reproducibility is key for meaningful research

But NEVER choose/ tune your random seeds!

Rein in the four horsemen of irreproducibility

Dorothy Bishop describes how threats to reproducibility, recognized but unaddressed for decades, might finally be brought under control.

Dorothy Bishop

Demonstration

All code is available at

`https://github.com/LukasSchaefer/RL2020_Building-a-Complete-RL-System_Demonstration`

References

Bishop, D. (2019). Rein in the Four Horsemen of Irreproducibility. *Nature*, 568(7753).

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W. (2016). OpenAI Gym.

Colas, C., Sigaud, O., and Oudeyer, P.-Y. (2019). A Hitchhiker's Guide to Statistical Comparisons of Reinforcement Learning Algorithms. *arXiv preprint arXiv:1904.06979*.

Henderson, P. (2018). *Reproducibility and Reusability in Deep Reinforcement Learning*. PhD thesis, McGill University Libraries.

Johnson, M., Hofmann, K., Hutton, T., and Bignell, D. (2016). The Malmo Platform for Artificial Intelligence Experimentation. In *IJCAI*, pages 4246–4247.

Kauten, C. (2018). Super Mario Bros for OpenAI Gym. GitHub.

Vinyals, O., Ewalds, T., Bartunov, S., Georgiev, P., Vezhnevets, A. S., Yeo, M., Makhzani, A., Küttler, H., Agapiou, J., Schrittwieser, J., et al. (2017). Starcraft II: A new Challenge for Reinforcement Learning. *arXiv preprint arXiv:1708.04782*.

Any questions about this lecture or the demonstration?