

# Deep Reinforcement Learning – Winter 2018/19

[Home](#)[Lectures](#)[Assignments](#)[Exam Questions](#)[Related Courses](#)

You should submit the assignments in the ReCodEx Code Examiner (<https://recodex.mff.cuni.cz/>), where they will be either automatically or manually evaluated (depending on the assignment). The evaluation is performed using Python 3.6, TensorFlow 1.11.0, NumPy 1.15.2 and OpenAI Gym 0.9.5. For those using PyTorch, CPU version 1.0.0 is available.

You can install TensorFlow and Gym either to user packages using `pip3 install --user tensorflow==1.11.0 gym==0.9.5 scipy box2d-py atari-py` (with the last three packages being optional dependencies of `gym`), or create a virtual environment using `python3 -m venv VENV_DIR` and then installing the packages inside it by running `VENV_DIR/bin/pip3 install tensorflow==1.11.0 gym==0.9.5 scipy box2d-py atari-py`. On Windows, you can use third-party precompiled versions of `box2d-py` (<https://www.lfd.uci.edu/~gohlke/pythonlibs/>) and `atari-py` (<https://github.com/Kojoley/atari-py/releases>). Note that when your CPU does not support AVX, you need to install TensorFlow 1.5.

## Submitting Data Files to ReCodEx

Even if ReCodEx (<https://recodex.mff.cuni.cz/>) allows submitting data files beside Python sources, the data files are not available during evaluation. Therefore, in order to submit models, you need to embed them in Python sources. You can use the `embed.py` script (<https://github.com/ufal/npfl122/blob/master/labs/embed.py>), which compressed and embeds given files and directories into a Python module providing an `extract()` method.

## Teamwork

Working in teams of size 2 (or at most 3) is encouraged. All members of the team must submit in ReCodEx individually, but can have exactly the same sources/models/results. **However, each such solution must explicitly list all members of the team to allow plagiarism detection using this template** ([https://github.com/ufal/npfl122/tree/master/labs/team\\_description.py](https://github.com/ufal/npfl122/tree/master/labs/team_description.py)).

## multiarmed\_bandits



Deadline: Oct 21, 23:59



compulsory

Perform a parameter study of various approaches to solving multiarmed bandits. For every hyperparameter choice, perform 1000 episodes, each consisting of 1000 trials, and report averaged return (a single number).

Start with the `multiarmed_bandits.py`

([https://github.com/ufal/npfl122/tree/master/labs/01/multiarmed\\_bandits.py](https://github.com/ufal/npfl122/tree/master/labs/01/multiarmed_bandits.py)) template, which defines `MultiArmedBandits` environment. We use API based on OpenAI Gym (<https://gym.openai.com/>) `Environment` class, notably the following two methods:


- `reset()` → `new_state` : starts a new episode
- `step(action)` → `new_state`, `reward`, `done`, `info` : perform the chosen action in the environment, returning the new state, obtained reward, a boolean flag indicating an end of episode, and additional environment-specific information. Of course, the states are not used by the multiarmed bandits ( `None` is returned).

Your goal is to implement the following modes of calculation. For each mode evaluate the performance given specified hyperparameters and plot the results for all modes together in a single graph.

- `greedy` : perform  $\epsilon$ -greedy search with parameter `epsilon` , computing the value function using averaging. Plot results for  $\epsilon \in \{1/64, 1/32, 1/16, 1/8, 1/4\}$ .
- `greedy` and `alpha`  $\neq 0$ : perform  $\epsilon$ -greedy search with parameter `epsilon` and initial function estimate of 0, using fixed learning rate `alpha` . Plot results for  $\alpha = 0.15$  and  $\epsilon \in \{1/64, 1/32, 1/16, 1/8, 1/4\}$
- `greedy` , `alpha`  $\neq 0$  and `initial`  $\neq 0$ : perform  $\epsilon$ -greedy search with parameter `epsilon` , given `initial` value as starting value function and fixed learning rate `alpha` . Plot results for `initial` = 1,  $\alpha = 0.15$  and  $\epsilon \in \{1/128, 1/64, 1/32, 1/16\}$ .
- `ucb` : perform UCB search with confidence level `c` and computing the value function using averaging. Plot results for  $c \in \{1/4, 1/2, 1, 2, 4\}$ .
- `gradient` : choose actions according to softmax distribution, updating the parameters using SGD to maximize expected reward. Plot results for  $\alpha \in \{1/16, 1/8, 1/4, 1/2\}$ .

This task will be evaluated manually and you should submit the Python source and the generated graph.

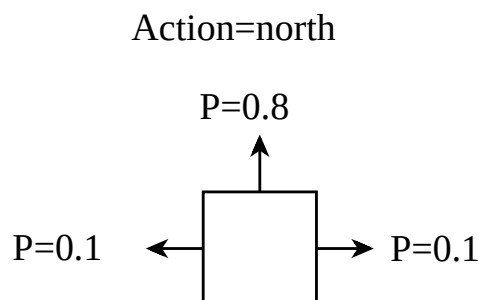
## policy\_iteration

 Deadline: Oct 28, 23:59

 compulsory

Consider the following gridworld:

0	0	0	1
0		0	-100
0	0	0	0



Start with `policy_iteration.py`

([https://github.com/ufal/npfl122/tree/master/labs/02/policy\\_iteration.py](https://github.com/ufal/npfl122/tree/master/labs/02/policy_iteration.py)), which implements the gridworld mechanics, by providing the following methods:

- `GridWorld.states` : return number of states ( 11 )
- `GridWorld.actions` : return lists with labels of the actions ( `["↑", "→", "↓", "←"]` )

- `GridWorld.step(state, action)` : return possible outcomes of performing the `action` in a given `state`, as a list of triples containing
  - `probability` : probability of the outcome
  - `reward` : reward of the outcome
  - `new_state` : new state of the outcome

Implement policy iteration algorithm, with `--steps` steps of policy evaluation/policy improvement. During policy evaluation, use the current value function and perform `--iterations` applications of the Bellman equation. Perform the policy evaluation synchronously (i.e., do not overwrite the current value function when computing its improvement). Assume the initial policy is “go North” and initial value function is zero.


After given number of steps and iterations, print the resulting value function and resulting policy. For example, the output after 4 steps and 4 iterations should be:

```

9.15→   10.30→   11.32→   12.33↑
8.12↑           3.35←   2.58←
6.95↑   5.90←   4.66←   -4.93↓

```

## monte\_carlo

 Deadline: Oct 28, 23:59

 compulsory

Solve the CartPole-v1 environment (<https://gym.openai.com/envs/CartPole-v1>) environment from the OpenAI Gym (<https://gym.openai.com/>) using the Monte Carlo reinforcement learning algorithm.

Use the supplied `cart_pole_evaluator.py` ([https://github.com/ufal/npfl122/tree/master/labs/02/cart\\_pole\\_evaluator.py](https://github.com/ufal/npfl122/tree/master/labs/02/cart_pole_evaluator.py)) module (depending on `gym_evaluator.py` ([https://github.com/ufal/npfl122/tree/master/labs/02/gym\\_evaluator.py](https://github.com/ufal/npfl122/tree/master/labs/02/gym_evaluator.py))) to interact with the discretized environment. The environment has the following methods and properties:


- `states` : number of states of the environment
- `actions` : number of actions of the environment
- `episode` : number of the current episode (zero-based)
- `reset(start_evaluate=False)` → `new_state` : starts a new episode
- `step(action)` → `new_state`, `reward`, `done`, `info` : perform the chosen action in the environment, returning the new state, obtained reward, a boolean flag indicating an end of episode, and additional environment-specific information
- `render()` : render current environment state

Once you finish training (which you indicate by passing `start_evaluate=True` to `reset`), your goal is to reach an average return of 490 during 100 evaluation episodes. Note that the environment prints your 100-episode average return each 10 episodes even during training.

You can start with the `monte_carlo.py` ([https://github.com/ufal/npfl122/tree/master/labs/02/monte\\_carlo.py](https://github.com/ufal/npfl122/tree/master/labs/02/monte_carlo.py)) template, which parses several useful parameters, creates the environment and illustrates the overall usage.

During evaluation in ReCodEx, three different random seeds will be employed, and you need to reach the required return on all of them. Time limit for each test is 5 minutes.

## q\_learning

 Deadline: Nov 04, 23:59

 **compulsory**


Solve the MountainCar-v0 environment (<https://gym.openai.com/envs/MountainCar-v0>) environment from the OpenAI Gym (<https://gym.openai.com/>) using the Q-learning reinforcement learning algorithm. Note that this task does not require TensorFlow.

Use the supplied `mountain_car_evaluator.py` ([https://github.com/ufal/npfl122/tree/master/labs/03/mountain\\_car\\_evaluator.py](https://github.com/ufal/npfl122/tree/master/labs/03/mountain_car_evaluator.py)) module (depending on `gym_evaluator.py` ([https://github.com/ufal/npfl122/tree/master/labs/02/gym\\_evaluator.py](https://github.com/ufal/npfl122/tree/master/labs/02/gym_evaluator.py))) to interact with the discretized environment. The environment methods and properties are described in the `monte_car_lo` assignment. Your goal is to reach an average reward of -150 during 100 evaluation episodes.

You can start with the `q_learning.py` ([https://github.com/ufal/npfl122/tree/master/labs/03/q\\_learning.py](https://github.com/ufal/npfl122/tree/master/labs/03/q_learning.py)) template, which parses several useful parameters, creates the environment and illustrates the overall usage. Note that setting hyperparameters of Q-learning is a bit tricky – I usually start with a larger value of  $\varepsilon$  (like 0.2 or even 0.5) and then gradually decrease it to almost zero.

During evaluation in ReCodEx, three different random seeds will be employed, and you need to reach the required return on all of them. The time limit for each test is 5 minutes.

## importance\_sampling

 Deadline: Nov 04, 23:59

 **compulsory**


Using the FrozenLake-v0 environment (<https://gym.openai.com/envs/FrozenLake-v0>) environment, implement Monte Carlo weighted importance sampling to estimate state value function  $V$  of target policy, which uniformly chooses either action 1 (down) or action 2 (right), utilizing behaviour policy, which uniformly chooses among all four actions.

Start with the `importance_sampling.py` ([https://github.com/ufal/npfl122/tree/master/labs/03/importance\\_sampling.py](https://github.com/ufal/npfl122/tree/master/labs/03/importance_sampling.py)) template, which creates the environment and generates episodes according to behaviour policy.

For 1000 episodes, the output of your program should be the following:

```
0.00 0.00 0.00 0.00
0.00 0.00 0.00 0.00
0.00 0.00 0.21 0.00
0.00 0.00 0.45 0.00
```

## lunar\_lander

 Deadline: Nov 11, 23:59

 **compulsory & 7 bonus**

Solve the LunarLander-v2 environment (<https://gym.openai.com/envs/LunarLander-v2>) environment from the OpenAI Gym (<https://gym.openai.com/>). Note that this task does not require TensorFlow.

Use the supplied `lunar_lander_evaluator.py` ([https://github.com/ufal/npfl122/tree/master/labs/03/lunar\\_lander\\_evaluator.py](https://github.com/ufal/npfl122/tree/master/labs/03/lunar_lander_evaluator.py)) module (depending on `gym_evaluator.py` ([https://github.com/ufal/npfl122/tree/master/labs/02/gym\\_evaluator.py](https://github.com/ufal/npfl122/tree/master/labs/02/gym_evaluator.py))) to interact with the discretized environment. The environment methods and properties are described in the `monte_carlo` assignment, but include one additional method:


- `expert_trajectory()` → `initial_state`, `trajectory` This method generates one expert trajectory and returns a pair of `initial_state` and `trajectory`, where `trajectory` is a list of the tripples (`action`, `reward`, `next_state`). You can use this method only during training, **not during evaluation**.

To pass the task, you need to reach an average return of 0 during 100 evaluation episodes. During evaluation in ReCodEx, three different random seeds will be employed, and you need to reach the required return on all of them. Time limit for each test is 5 minutes.

The task is additionally a *competition* and at most 7 points will be awarded according to relative ordering of your solution performances.

You can start with the `lunar_lander.py` ([https://github.com/ufal/npfl122/tree/master/labs/03/lunar\\_lander.py](https://github.com/ufal/npfl122/tree/master/labs/03/lunar_lander.py)) template, which parses several useful parameters, creates the environment and illustrates the overall usage.

## q\_learning\_tiles

 Deadline: Nov 18, 23:59

 compulsory

Improve the `q_learning` task performance on the MountainCar-v0 environment (<https://gym.openai.com/envs/MountainCar-v0>) environment using linear function approximation with tile coding. Your goal is to reach an average reward of -110 during 100 evaluation episodes.

Use the updated `mountain_car_evaluator.py` ([https://github.com/ufal/npfl122/tree/master/labs/03/mountain\\_car\\_evaluator.py](https://github.com/ufal/npfl122/tree/master/labs/03/mountain_car_evaluator.py)) module (depending on updated `gym_evaluator.py` ([https://github.com/ufal/npfl122/tree/master/labs/02/gym\\_evaluator.py](https://github.com/ufal/npfl122/tree/master/labs/02/gym_evaluator.py))) to interact with the discretized environment. The environment methods and properties are described in the `monte_carlo` assignment, with the following changes:


- The `env.weights` method return the number of weights of the linear function approximation.
- The `state` returned by the `env.step` method is a *list* containing weight indices of the current state (i.e., the feature vector of the state consists of zeros and ones, and only the indices of the ones are returned). The (action-)value function for a state is therefore approximated as a sum of the weights whose indices are returned by `env.step`.

The default number of tiles in tile encoding (i.e., the size of the list with weight indices) is `args.tiles=8`, but you can use any number you want (but the assignment is solvable with 8).

You can start with the `q_learning_tiles.py` ([https://github.com/ufal/npfl122/tree/master/labs/04/q\\_learning\\_tiles.py](https://github.com/ufal/npfl122/tree/master/labs/04/q_learning_tiles.py)) template, which parses several useful parameters, creates the environment and illustrates the overall usage. Implementing Q-learning is enough to pass the assignment, even if both N-step Sarsa and Tree Backup converge a little faster.

During evaluation in ReCodEx, three different random seeds will be employed, and you need to reach the required return on all of them. The time limit for each test is 5 minutes.

## q\_network

 Deadline: Nov 25, 23:59

 **compulsory**

Solve the CartPole-v1 environment (<https://gym.openai.com/envs/CartPole-v1>) environment from the OpenAI Gym (<https://gym.openai.com/>) using Q-learning with neural network as a function approximation.

The supplied `cart_pole_evaluator.py` ([https://github.com/ufal/npfl122/tree/master/labs/02/cart\\_pole\\_evaluator.py](https://github.com/ufal/npfl122/tree/master/labs/02/cart_pole_evaluator.py)) module (depending on `gym_evaluator.py` ([https://github.com/ufal/npfl122/tree/master/labs/02/gym\\_evaluator.py](https://github.com/ufal/npfl122/tree/master/labs/02/gym_evaluator.py))) can also create a continuous environment using `environment(discrete=False)`. The continuous environment is very similar to the discrete environment, except that the states are vectors of real-valued observations with shape `environment.state_shape`.


Use Q-learning with neural network as a function approximation, which for a given states returns state-action values for all actions. You can use any network architecture, but two hidden layers of 20 ReLU units are a good start.

Your goal is to reach an average return of 400 during 100 evaluation episodes.

You can start with the `q_network.py` ([https://github.com/ufal/npfl122/tree/master/labs/05/q\\_network.py](https://github.com/ufal/npfl122/tree/master/labs/05/q_network.py)) template, which provides a simple network implementation in TensorFlow.

During evaluation in ReCodEx, two different random seeds will be employed, and you need to reach the required return on all of them. Time limit for each test is 10 minutes (so you can train in ReCodEx, but you can also pretrain your network if you like).

## car\_racing

 Deadline: Dec 2, 23:59

 **10 bonus only**

Nov 27: The evaluator has been returning a reference to the same numpy array with the state, which could have caused problems if you did not create a copy (when stacking images or resizing it). It has now been fixed.

In this bonus-only exercise to play with Deep Q Network and its variants, try solving the CarRacing-v0 environment (<https://gym.openai.com/envs/CarRacing-v0>) environment from the OpenAI Gym (<https://gym.openai.com/>).

Use the supplied `car_racing_evaluator.py` ([https://github.com/ufal/npfl122/tree/master/labs/06/car\\_racing\\_evaluator.py](https://github.com/ufal/npfl122/tree/master/labs/06/car_racing_evaluator.py)) module (depending on `gym_evaluator.py` ([https://github.com/ufal/npfl122/tree/master/labs/02/gym\\_evaluator.py](https://github.com/ufal/npfl122/tree/master/labs/02/gym_evaluator.py))) to interact with the environment. The environment is continuous and states are RGB images of size  $96 \times 96 \times 3$ , but you can downsample them even more. The actions are also continuous and consist of an array with the following three elements:

- `steer` in range `[-1, 1]`
- `gas` in range `[0, 1]`
- `brake` in range `[0, 1]`


Internally you should generate discrete actions and convert them to the required representation before the `step` call. Good initial action space is to use 9 actions – a Cartesian product of 3 steering actions (left/right/none) and 3 driving actions (gas/brake/none).

The environment supports frame skipping without rendering the skipped frames – the second argument to `env.step` determines how many time is the given action repeated.

The task is a *competition* and at most 10 points will be awarded according to relative ordering of your solution performances. In ReCodEx, your solution is evaluated on 15 different tracks with a total time limit of 15 minutes. If your average return is at least 100, ReCodEx shows the solution as correct.

The `car_racing.py` ([https://github.com/ufal/npfl122/tree/master/labs/06/car\\_racing.py](https://github.com/ufal/npfl122/tree/master/labs/06/car_racing.py)) template parses several useful parameters and creates the environment. Note that the `car_racing_evaluator.py` ([https://github.com/ufal/npfl122/tree/master/labs/06/car\\_racing\\_evaluator.py](https://github.com/ufal/npfl122/tree/master/labs/06/car_racing_evaluator.py)) can be executed directly and in that case you can drive the car using arrows.

## reinforce

 Deadline: Dec 02, 23:59

 compulsory

Solve the CartPole-v1 environment (<https://gym.openai.com/envs/CartPole-v1>) environment from the OpenAI Gym (<https://gym.openai.com/>) using the REINFORCE algorithm.

The supplied `cart_pole_evaluator.py` ([https://github.com/ufal/npfl122/tree/master/labs/02/cart\\_pole\\_evaluator.py](https://github.com/ufal/npfl122/tree/master/labs/02/cart_pole_evaluator.py)) module (depending on `gym_evaluator.py` ([https://github.com/ufal/npfl122/tree/master/labs/02/gym\\_evaluator.py](https://github.com/ufal/npfl122/tree/master/labs/02/gym_evaluator.py))) can create a continuous environment using `environment(discrete=False)`. The continuous environment is very similar to the discrete environment, except that the states are vectors of real-valued observations with shape `environment.state_shape`.

Your goal is to reach an average return of 490 during 100 evaluation episodes.

You can start with the `reinforce.py` (<https://github.com/ufal/npfl122/tree/master/labs/06/reinforce.py>) template, which provides a simple network implementation in TensorFlow.

During evaluation in ReCodEx, two different random seeds will be employed, and you need to reach the required return on all of them. Time limit for each test is 5 minutes.

## reinforce\_with\_baseline

 Deadline: Dec 09, 23:59

 compulsory

This is a continuation of `reinforce` assignment.

Using the `reinforce_with_baseline.py` ([https://github.com/ufal/npfl122/tree/master/labs/07/reinforce\\_with\\_baseline.py](https://github.com/ufal/npfl122/tree/master/labs/07/reinforce_with_baseline.py)) template, solve the CartPole-v1 environment (<https://gym.openai.com/envs/CartPole-v1>) environment using the REINFORCE with baseline algorithm.


Using a baseline lowers the variance of the value function gradient estimator, which allows faster training and decreases sensitivity to hyperparameter values. To reflect this effect in ReCodEx, note that the evaluation phase will *automatically start after 200 episodes*. Using only 200

episodes for training in this setting is probably too little for the REINFORCE algorithm, but suffices for the variant with a baseline.

Your goal is to reach an average return of 490 during 100 evaluation episodes.

During evaluation in ReCodEx, two different random seeds will be employed, and you need to reach the required return on all of them. Time limit for each test is 5 minutes.

## cart\_pole\_pixels

 Deadline: Dec 09, 23:59

 **compulsory & 7 bonus**

The supplied `cart_pole_pixels_evaluator.py` ([https://github.com/ufal/npfl122/tree/master/labs/07/cart\\_pole\\_pixels\\_evaluator.py](https://github.com/ufal/npfl122/tree/master/labs/07/cart_pole_pixels_evaluator.py)) module (depending on `gym_evaluator.py` ([https://github.com/ufal/npfl122/tree/master/labs/02/gym\\_evaluator.py](https://github.com/ufal/npfl122/tree/master/labs/02/gym_evaluator.py))) generates a pixel representation of the `CartPole` environment as an  $80 \times 80$  image with three channels, with each channel representing one time step (i.e., the current observation and the two previous ones).

To pass the compulsory part of the assignment, you need to reach an average return of 50 during 100 evaluation episodes. During evaluation in ReCodEx, two different random seeds will be employed, and you need to reach the required return on all of them. Time limit for each test is 10 minutes.

The task is additionally a *competition* and at most 7 points will be awarded according to relative ordering of your solution performances.

The `cart_pole_pixels.py` ([https://github.com/ufal/npfl122/tree/master/labs/07/cart\\_pole\\_pixels.py](https://github.com/ufal/npfl122/tree/master/labs/07/cart_pole_pixels.py)) template parses several parameters, creates the environment and shows how to save and load neural networks in TensorFlow. To upload the trained model to ReCodEx, you need to embed the trained model files using `embed.py` (<https://github.com/ufal/npfl122/blob/master/labs/embed.py>), submit the resulting `embedded_data.py` along your solution, and in your solution you need to `import embedded_data` and then `embedded_data.extract()` (the template does this for you).

## paac

 Deadline: Dec 16, 23:59

 **compulsory**

Using the `paac.py` (<https://github.com/ufal/npfl122/tree/master/labs/08/paac.py>) template, solve the `CartPole-v1` environment (<https://gym.openai.com/envs/CartPole-v1>) environment using parallel actor-critic algorithm.

The `gym_environment` now provides the following two methods:


- `parallel_init(num_workers)` → `initial_states`, which initializes the given number of parallel workers and returns their environment initial states. This method can be called at most once.
- `parallel_step(actions)` → `List[next_state, reward, done, info]`, which performs given action in respective environment, and return the usual information with one exception: **If `done=True`, then `next_state` is already a new state of newly started episode.**

Your goal is to reach an average return of 450 during 100 evaluation episodes.



During evaluation in ReCodEx, two different random seeds will be employed, and you need to reach the required return on all of them. Time limit for each test is 10 minutes.

## paac\_continuous

 Deadline: Dec 16, 23:59

 **compulsory**

Using the `paac_continuous.py`

([https://github.com/ufal/npfl122/tree/master/labs/08/paac\\_continuous.py](https://github.com/ufal/npfl122/tree/master/labs/08/paac_continuous.py)) template, solve the MountainCarContinuous-v0 environment (<https://gym.openai.com/envs/MountainCarContinuous-v0/>) environment using parallel actor-critic algorithm with continuous actions.


The `gym_environment` now provides two additional methods:

- `action_shape` : returns required shape of continuous action. You can assume the actions are always an one-dimensional vector.
- `action_ranges` : returns a pair of vectors `low` , `high` . These denote valid ranges for the actions, so  $low[i] \leq action[i] \leq high[i]$  .

Your goal is to reach an average return of 90 during 100 evaluation episodes.

During evaluation in ReCodEx, two different random seeds will be employed, and you need to reach the required return on all of them. Time limit for each test is 10 minutes.

## ddpg

 Deadline: Jan 06, 23:59

 **compulsory**

Using the `ddpg.py` (<https://github.com/ufal/npfl122/tree/master/labs/09/ddpg.py>) template, solve the Pendulum-v0 environment (<https://gym.openai.com/envs/Pendulum-v0/>) environment using deep deterministic policy gradient algorithm.


To create the evaluator, use `gym_evaluator.py`

([https://github.com/ufal/npfl122/tree/master/labs/02/gym\\_evaluator.py](https://github.com/ufal/npfl122/tree/master/labs/02/gym_evaluator.py)) `.GymEvaluator("Pendulum-v0")` . The environment is continuous, states and actions are described at OpenAI Gym Wiki (<https://github.com/openai/gym/wiki/Pendulum-v0>).

Your goal is to reach an average return of -200 during 100 evaluation episodes.

During evaluation in ReCodEx, two different random seeds will be employed, and you need to reach the required return on all of them. Time limit for each test is 10 minutes.

## walker

 Deadline: Jan 06, 23:59

 **10 bonus only**

In this bonus-only exercise exploring continuous robot control, try solving the BipedalWalker-v2 environment (<https://gym.openai.com/envs/BipedalWalker-v2/>) environment from the OpenAI Gym (<https://gym.openai.com/>).


To create the evaluator, use `gym_evaluator.py`


([https://github.com/ufal/npfl122/tree/master/labs/02/gym\\_evaluator.py](https://github.com/ufal/npfl122/tree/master/labs/02/gym_evaluator.py)) `.GymEvaluator("BipedalWalker-v2")` . The environment is continuous, states and actions are described at OpenAI Gym Wiki (<https://github.com/openai/gym/wiki/BipedalWalker-v2>).

The task is a *competition* and at most 10 points will be awarded according to relative ordering of your solution performances. In ReCodEx, your solution will be evaluated on 100 different tracks with a total time limit of 10 minutes. If your average return is at least 0, ReCodEx shows the solution as correct.

You can start with the `ddpg.py` (<https://github.com/ufal/npfl122/tree/master/labs/09/ddpg.py>) template, only set `args.env` to `BipedalWalker-v2`.

## walker\_hardcore

 Deadline: Jan 06, 23:59


 5 bonus only

As an extension of the `walker` assignment, try solving the `BipedalWalkerHardcore-v2` environment (<https://gym.openai.com/envs/BipedalWalkerHardcore-v2>) environment from the OpenAI Gym (<https://gym.openai.com/>).

The task is a *competition* and at most 5 points will be awarded according to relative ordering of your solution performances. In ReCodEx, your solution will be evaluated on 100 different tracks with a total time limit of 10 minutes. If your average return is at least 0, ReCodEx shows the solution as correct.

You can start with the `ddpg.py` (<https://github.com/ufal/npfl122/tree/master/labs/09/ddpg.py>) template, only set `args.env` to `BipedalWalkerHardcore-v2`.

## az\_quiz

 Deadline: Jan 13, 23:59

 10 bonus only

In this bonus-only exercise, use Monte Carlo Tree Search to learn an agent for a simplified version of AZ-kvíz (<https://cs.wikipedia.org/wiki/AZ-kv%C3%ADz>). In our version, the agent does not have to answer questions and we assume that **all answers are correct**.


The game itself is implemented in the `az_quiz.py` ([https://github.com/ufal/npfl122/tree/master/labs/10/az\\_quiz.py](https://github.com/ufal/npfl122/tree/master/labs/10/az_quiz.py)) module, using `randomized=False` constructor argument.


The evaluation in ReCodEx should be implemented by importing a module `az_quiz_evaluator_recodex` and calling its `evaluate` function. The argument this function is an object providing a method `play` which given an AZ-kvíz instance returns the chosen move. The illustration of the interface is in the `az_quiz_evaluator_recodex.py` ([https://github.com/ufal/npfl122/tree/master/labs/10/az\\_quiz\\_evaluator\\_recodex.py](https://github.com/ufal/npfl122/tree/master/labs/10/az_quiz_evaluator_recodex.py)) module.

Your solution in ReCodEx is automatically evaluated only against a random player `az_quiz_player_random.py` ([https://github.com/ufal/npfl122/tree/master/labs/10/az\\_quiz\\_player\\_random.py](https://github.com/ufal/npfl122/tree/master/labs/10/az_quiz_player_random.py)) and a very simple heuristic `az_quiz_player_simple_heuristic.py` ([https://github.com/ufal/npfl122/tree/master/labs/10/az\\_quiz\\_player\\_simple\\_heuristic.py](https://github.com/ufal/npfl122/tree/master/labs/10/az_quiz_player_simple_heuristic.py)), playing against each of them 10 games as a starting player and 10 games as a non-starting player. The time limit for the games is 10 minutes and you should see win rate directly in ReCodEx. The final evaluation will be performed after the deadline by a round-robin tournament, utilizing your latest submission with non-zero win rate.

For inspiration, use the official pseudocode for AlphaZero ([http://science.sciencemag.org/highwire/filestream/719481/field\\_highwire\\_adjunct\\_files/1/aar6404\\_DataS1.zip](http://science.sciencemag.org/highwire/filestream/719481/field_highwire_adjunct_files/1/aar6404_DataS1.zip)).

## az\_quiz\_randomized

 Deadline: Jan 13, 23:59

 5 bonus only

Extend the `az_quiz` assignment to handle the possibility of wrong answers. Therefore, when choosing a field, the agent might answer incorrectly.

To instantiate this randomized game variant, pass `randomized=True` to the `AZQuiz` class of `az_quiz.py` ([https://github.com/ufal/npfl122/tree/master/labs/10/az\\_quiz.py](https://github.com/ufal/npfl122/tree/master/labs/10/az_quiz.py)).


The Monte Carlo Tree Search has to be slightly modified to handle stochastic MDP. The information about distribution of possible next states is provided by the `AZQuiz.all_moves` method, which returns a list of `(probability, az_quiz_instance)` next states (in our environment, there are always two possible next states).

The evaluation in ReCodEx should be implemented by importing a module `az_quiz_evaluator_recodex` and calling its `evaluate` function. The argument this functions is an object providing a method `play` which given an AZ-kvíz instance returns the chosen move. The illustration of the interface is in the `az_quiz_evaluator_recodex.py` ([https://github.com/ufal/npfl122/tree/master/labs/10/az\\_quiz\\_evaluator\\_recodex.py](https://github.com/ufal/npfl122/tree/master/labs/10/az_quiz_evaluator_recodex.py)) module.

Your solution in ReCodEx is automatically evaluated only against a random player `az_quiz_player_random.py` ([https://github.com/ufal/npfl122/tree/master/labs/10/az\\_quiz\\_player\\_random.py](https://github.com/ufal/npfl122/tree/master/labs/10/az_quiz_player_random.py)) and a very simple heuristic `az_quiz_player_simple_heuristic.py` ([https://github.com/ufal/npfl122/tree/master/labs/10/az\\_quiz\\_player\\_simple\\_heuristic.py](https://github.com/ufal/npfl122/tree/master/labs/10/az_quiz_player_simple_heuristic.py)), playing against each of them 10 games as a starting player and 10 games as a non-starting player. The time limit for the games is 10 minutes and you should see win rate directly in ReCodEx. The final evaluation will be performed after the deadline by a round-robin tournament, utilizing your latest submission with non-zero win rate.

For inspiration, use the official pseudocode for AlphaZero ([http://science.sciencemag.org/highwire/filestream/719481/field\\_highwire\\_adjunct\\_files/1/aar6404\\_DataS1.zip](http://science.sciencemag.org/highwire/filestream/719481/field_highwire_adjunct_files/1/aar6404_DataS1.zip)).

## vtrace

 Deadline: Jan 20, 23:59


 compulsory

Using the `vtrace.py` (<https://github.com/ufal/npfl122/tree/master/labs/11/vtrace.py>) template, implement the V-trace algorithm.

The template uses the `CartPole-v1` environment and a replay buffer to more thoroughly test the off-policy capability of the V-trace algorithm.

However, the evaluation in ReCodEx will be performed by calling only the `vtrace` method and comparing its results to a reference implementation. Several values of hyperparameters will be used, each test has a time limit of 1 minute, and all tests must pass.

## memory\_game

 Deadline: Feb 17, 23:59

 5 bonus only

In this bonus-only exercise we explore a partially observable environment. Consider a one-player variant of a memory game (pexeso), where a player repeatedly flip cards. If the player flips two cards with the same symbol in succession, the cards are removed and the player receives a reward of +2. Otherwise the player receives a reward of -1. An episode ends when all cards are removed. For a given even  $N$ , there are  $N$  actions – the card indices, and  $N/2$  observations, which encode card symbol. Every episode can be ended using at most  $3N/2$  actions. The environment is provided by the `memory_game_evaluator.py` ([https://github.com/ufal/npfl122/tree/master/labs/11/memory\\_game\\_evaluator.py](https://github.com/ufal/npfl122/tree/master/labs/11/memory_game_evaluator.py)) module.

Your goal is to solve the environment using an agent utilizing a LSTM cell. The reinforce algorithm with baseline seems an appropriate choice.

ReCodEx evaluates your solution on environments with 4, 6, 8 and 16 cards (utilizing the `--cards` argument). For each card number, 1000 episodes are simulated and your solution gets 1 point (2 points for 16 cards) if the average return is positive.

A template `memory_game.py` ([https://github.com/ufal/npfl122/tree/master/labs/11/memory\\_game.py](https://github.com/ufal/npfl122/tree/master/labs/11/memory_game.py)) is available. Depending on `memory_cells` argument, it employs either a vanilla LSTM or LSTM with external memory. Note that I was able to train it only for 4, 6, and 8 cards.




EVROPSKÁ UNIE  
Evropské strukturální a investiční fondy  
Operační program Výzkum, vývoj a vzdělávání



### Malostranské náměstí 25

118 00 Praha  
Czech Republic

+420 951 554 278 (phone)  
[ufal@mff.cuni.cz](mailto:ufal@mff.cuni.cz) (<mailto:ufal@mff.cuni.cz>)

(<https://twitter.com/lindatclarin>)  (<https://lindat.mff.cuni.cz/repository/xmlui/>)



(<https://www.facebook.com/UFALMFFUK>)

Page curated by straka (/~straka) | Sign in (/user/login?destination=node/1747)

Institute of Formal and Applied Linguistics © 2019

Powered by  Drupal (<http://drupal.org>)