Reinforcement Learning: Challenge

Handout: 05.05.2025 08:00
Due: 13.05.2025 23:59

Spring 2025
Hands-On Deep Learning

</> Challenge

Open in IDE

Flappy Bird

Prototype Notebook: Open in Google Colab.

Important: you ahve to submit your solution on CodeExpert.

In this challenge, you train an agent to play the classic Flappy Bird mobile game. If you feel like taking a break, you can play the game here.

Environment

The goal is to pass as many pipes as possible. The game ends if the bird hits a pipe or the ground.

Action Space

- 0: Do nothing
- 1: Flap

Rewards

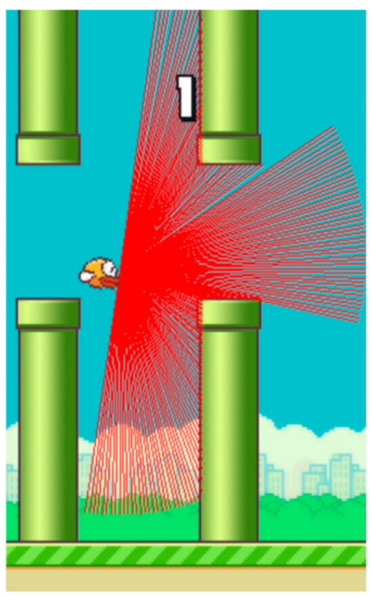
The agent receives

- +1.0 for each successfully passed pipe
- +0.1 for every frame it stays alive
- -0.5 for touching the top of the screen
- -1.0 for dying

In the original game, the only reward is from passing pipes. The additional rewards are designed to facilitate learning. During evaluation, the discount factor γ is set to 1.

Observations

The agent observes the environment using a lidar sensor with 180 rays in different directions.Each observation is a 180-element vector representing the distance to the nearest object in each direction.



For more details on the environment, check out this repository. This is a fork of the original repository, implementing the lidar observations.

Challenge details

Task

Your task is to complete the `init_model`, `train_model` and `apply_wrappers` functions. The first two are used to initialize and train the agent, while `apply_wrappers` lets you modify the environment to suit your agent's needs (e.g. observation preprocessing with a `TensorWrapper`). Do not change the function signatures. Your code will be executed via the following function:

```
def run():
  scores = []
  for i, seed in enumerate([42, 43, 44]):
    print(f"Run #{i}")
    set_seed(seed)

    start = time.time()

    train_env, valid_envs = create_envs(seed)

    # Initialize the model using student's init_model function
    agent = init_model(train_env)

    # Train the model using student's train_model function
    agent = train_model(agent, train_env)

    score = evaluate_model(agent, valid_envs)
    scores.append(score)

  return np.mean(score)
```

The validation enviroments are automatically wrapped using `apply_wrappers`. For the training environment you need to do it in `train_model`.

```
train_env = apply_wrappers(train_env)
```

Reinforcement learning is inherently unstable. To mitigate this, we train using three different random seeds and report the average score.

Training the agent

You're free to use DQN, Double DQN, or another RL method—but you must implement the algorithm yourself. Using external libraries for the core learning logic is not allowed.

Here are some tips for parameter tuning and debugging:

- The observations consist of 180 elements, so it's a good idea to start with at least that many neurons in the first hidden layer.
- Use a low learning rate, around 0.001 or even lower.
- Ensure the replay memory is large enough to store plenty of episodes, but not so large that older experiences dominate over more recent, relevant ones.
- Starting with $\epsilon = 1$ (fully random exploration) might not be ideal. Since the number of *hops* is generally low compared to *doing nothing*, too much initial randomness would cause most runs to fail quickly.
- The `pipe_gap` parameter of the Flappy Bird environment controls the vertical distance between pairs of pipes. You can temporarily increase this to test your agent in an easier environment. However, remember to reset it to 140 when training an agent for the final submission.
- Use `agent.net.load_state_dict(torch.load('policy_net.pth'))` to load network weights from a previous save
- `generate_gif(env, agent, n_frames=300)` generates a gif of your agent

Testing the agent

The agent is tested in an environment with `pipe_gap=140`. Additional tests with smaller `pipe_gap` values (as low as 80) can award a small number of bonus points. These tests on smaller gaps cannot reduce your score and only have a marginal effect, so you don't need to train your agent specifically for them.

Grading

Points are awarded based on the model's average score, according to the thresholds below:

Score (running)	CodeExpert score (submitting)	Points Earned
< 5	< 8.3%	0
≥ 5	≥ 8.3%	1
≥ 10	≥ 25.0%	2
≥ 15	≥ 41.7%	3
≥ 50	≥ 58.3%	4
≥ 120	≥ 75.0%	5
≥ 300	≥ 91.7%	6

The CodeExpert score is the percentage shown when submitting your job.

Rules

You must implement the RL algorithm yourself—using external libraries for the core learning logic (e.g., a prebuilt DQN implementation) is not allowed.

You cannot set a new seed.

Your code must run in 15 minutes.

Toblerone

The top three solutions will receive a Toblerone. Good luck! 🚀