Task Description for “Practical Course: Bomberman Challenge”
Winter Term 2024

Issued: 23.12.2024, **Due:** 22.01.2025, **Discussion:** 24.01.2025

Introduction

In the final project, you will apply your knowledge of deep reinforcement learning (DRL) to develop an agent capable of competing in a Bomberman tournament. Starting from a standard approach, you should demonstrate how the agent learns inside the environment and, secondly, extend and improve a standard approach (this can address a curriculum, reward shaping, neural network training, exploration strategies, state space ...).

As a final submission, you have to upload one zip-file that includes two main parts (see checklist at end for details): Your training approach that includes an extension or adaptation of a standard DRL approach and a trained, runnable agent.

Part 1: Training Routine

Assignment Requirements

a. **Basic Training Routine**: Develop and implement a basic DRL training routine in Python on your own. This should involve setting up the environment (see the example environments used as well for your rule-based agent), defining the neural network (e.g., Q-network), and implementing a simple learning algorithm like DQN.

b. **Extended DRL Approach**: Furthermore, you should work on an improved learning agent. For this, you are allowed to incorporate further approaches or extend existing learning approaches (but you have to be able: to explain these and the reason why you decided to follow a particular approach). Your improvement can deal with all parts of the DRL training (learning routine, gathering experience, exploration, state representation, imitate another agent, rewards). As one remark: The only thing that is not allowed is to replace —in any form—the decision making with a heuristics like a rule-based approach (this means as well that you are not allowed to add additional Information to the state during the tournament (or later change actions) using, for example, strong heuristics that would guide the agent). Decisions have to be done by a trained component.

   Your goal is to improve the learning efficiency and agent performance significantly. This should be documented, on the one hand, for the training (showing a learning curve in comparison to your first approach). On the other hand, you should consider how to evaluate your agent, for example, demonstrating how he performs against a given opponent.

Documentation and Demonstration

Submit all source code with adequate documentation (as a jupyter notebook or as python code with additional documentation of training progress; submit as well your trained neural networks.). Demonstrate the learning effectiveness through plotting training curves (submit ones from your training runs as well), showing key performance metrics over time.

Part 2: Trained Agent

Tournament Readiness

Ensure your agent adheres to the game's interface requirements and can operate independently within the tournament environment without additional modifications (it is providing an action as an output and only using the original state as an input. The state might be preprocessed or condensed. But, as a reminder, you are not allowed to enrich it with information on optimal decisions).

Submission notes:

Include the complete code for your trained agent, detailing the neural network architecture, DRL techniques employed, and any preprocessing steps used.

Evaluation and Testing of Submissions

Testing Environment for Training

We will provide a simplified game environment for initial testing and training. Ensure your agent can be trained in these environments and solves the simple tasks (as collecting a coin) in these controlled settings. It has to show learning curves afterwards.

Evaluation Criteria for the Trained Agent

The evaluation of the trained agent, encompassing both the code and neural network weights, will be conducted in two phases:

- **Basic Test for Effectiveness of the Trained Agent:** The agent will first be assessed in the Bomberman arena, but in a simpler task without any opponents or against a non-aggressive, simple agent. This stage is intended to evaluate the basic functionality and decision-making capabilities of the agent in a controlled environment. Your agent should show that he is able to survive in this environment and gathering basic rewards.
- **Tournament Testing:** Following the initial tests, a tournament will be held where all submitted agents compete against each other. This phase is designed to measure each agent's strategic depth and adaptability in a competitive setting.

Final Presentation Guidelines for Students

For a brief final presentation at the tournament, you are required to prepare up to three slides (presentation time of maximum five minutes). The slides should articulate the development and performance of the trained agent. Consider the following key elements:

- **Positioning of the approach:** Start by describing which part of the Reinforcement Learning loop and Neural Network training your adaptation addresses and what the goal of your extensions are. This might include exploration strategies, neural network training dynamics, reward shaping techniques, or curriculum design to enhance learning effectiveness.
- **Overview of Approach and Architecture:** You should provide an overview of the proposed approach and the architectural design of the whole training system (RL and NN). It should ideally include a visual representation of the architecture, highlighting added components or modifications to standard algorithms.

- **Demonstration of Results:** Finally, show results for your approach and explain how this demonstrates the impacts of the changes made to a standard algorithm. This should include comparative data and a discussion of how the modifications improved or affected the agent's performance.

Resources and Support

There are two GitLab repositories that are designed to support and facilitate the development of the reinforcement learning agent and training the Neural Networks:

- **Bomberman Reinforcement Learning Environment:** Accessible at `https://zivgitlab.uni-muenster.de/ai-systems/bomberman_rl`, this repository contains the game environment compliant with the Gymnasium standard. It includes an example agent to help students get started. The repository will also be updated to include additional test environments to assist in the development and testing phases of your project.
- **Deep Learning Course Material:** Located at `https://zivgitlab.uni-muenster.de/ai-systems/drl_deep_course`, this repository features code relevant to the deep neural network (DNN) part of the course. Specifically, at `https://zivgitlab.uni-muenster.de/ai-systems/drl_deep_course/-/tree/main/Bomberman/Q-Learning_Examples?ref_type=heads`, there are examples for training neural networks using PyTorch and TensorFlow to learn the Q-function, which is central to many reinforcement learning strategies.

These resources are intended to provide foundational tools and examples that shall aid you. For questions and remarks (difficulties, but as well successful hints) use the forum in the LearnWeb.

In your project, you can use well-established Python libraries to aid in the development of your deep reinforcement learning models. Core libraries such as **PyTorch** or **TensorFlow** should form the backbone of your neural network implementations. Additionally, **gymnasium** is used for the test environments.

You are also permitted to employ other standard libraries that facilitate machine learning tasks, such as **scikit-learn** for various preprocessing and analytics functions, or **stable-baselines** for ready-to-use RL algorithms.

Important: It is imperative that you clearly document the use of all external libraries in your project. For each library utilized, you must provide the following details:

- The version of the library.
- A brief description of how and why the library was used in your project.
- Any modifications made to library components, if applicable.

This information is crucial not only for the reproducibility of your results but also to ensure transparency and adherence to best practices in software development.

Final Submission Checklist

Please ensure each part of your submission meets the following criteria (code realized in python):

- **Training Routine:**
  - ☐ Implement and submit a simple version of a standard algorithm.
  - ☐ Extend the simple implementation to a more sophisticated deep reinforcement learning (DRL) approach.
  - ☐ Provide an explanation of the extension, including citations of relevant sources and used libraries.
  - ☐ Ensure the implementation adheres to the predefined environment interface.
  - ☐ Output and include a training curve showing the performance improvement. This has to show improvement over time.
  - ☐ (All) Code should be well-documented.
- **Trained Agent:**
  - ☐ Submit the complete code necessary for running the trained agent.
  - ☐ Include saved neural network weights, all relevant parameters, and hyperparameters (this should be reproducible).
  - ☐ Provide a loading routine that allows to load all these relevant parameters and run the agent from your saved state.
  - ☐ Ensure that the agent adheres to the standard agent interface, allowing it to be easily integrated and run in the environments.
  - ☐ Detail your training time and briefly mention the computation costs involved (as well for preprocessing).
- **Presentation Slide:**
  - ☐ Prepare a slide deck (up to three slides) for the final presentation (up to five minutes).
  - ☐ Explaining the reinforcement learning and neural network aspects addressed in the project.
  - ☐ Containing an overview of the approach and architecture.
  - ☐ Showcasing the results and demonstrating the effect of the changes made to the standard algorithm.
- **README File:**
  - ☐ Provide an overview of the folder structure and files.
  - ☐ Detail your development environment including versions used for Python, PyTorch, TensorFlow, and any other significant libraries.
  - ☐ Include instructions on how to train, run, and test the agent.

Acknowledgments

OpenAI's ChatGPT was used for generating parts of the text used in this document. The tool provided initial drafts that were further refined and contextualized to fit the project.