CONNECT4_RL: A GAME OF CONNECT 4 WITH REINFORCEMENT LEARNING

A REPORT

Paweł Fornalik[*1] **and Marcin Dąbrowski**[†1]

[1]Faculty of Mathematics and Computer Science, Jagiellonian University in Kraków

ABSTRACT

This report presents a demonstrative implementation of a Neural Network-based Connect Four playing system utilizing Deep Q Networks (DQN). The objective of this project was to develop an AI agent system capable of producing agents able to play a game of Connect Four at a competitive level by leveraging reinforcement learning techniques. The DQN architecture combines the principles of Q-learning with deep learning, enabling the agent to learn optimal strategies solely through experience. The implementation details include the design of the neural network, the state representation of the game board, and the reward structure that guides the learning process. This work not only highlights the usability of DQNs in game-playing applications but also serves as a cornerstone for further exploration into more complex reinforcement learning environments.

Keywords Deep Q Networks (DQN) · Neural Networks · Reinforcement Learning · Game-Playing Agents

1 Introduction — Marcin Dąbrowski

Connect Four is a two-player board game that requires strategic thinking and planning, making it an ideal candidate for the application of artificial intelligence techniques. Deep Q Networks (DQN), a type of reinforcement learning algorithm, have been widely and successfully applied to various games to demonstrate their potential in effectively learning complex decision-making policies. However, we have not found a practical working example of application of DQNs to a player for a game of Connect Four, so we have decided to alleviate that issue.

We present a working example implementation of a Neural Network-based Connect Four playing system utilizing Deep Q Networks. The implementation described in this report leverages the DQN architecture to train an agent to play Connect Four against a human opponent or another AI agent. The system consists of a neural network that learns to predict the expected return or reward for each possible action in a given game state, allowing the agent to make informed decisions and improve its gameplay over time.

This report provides an overview of the system's architecture, the training and inference processes, and the results obtained from testing the implementation under baseline conditions. The goal of this project is to demonstrate the feasibility and efficacy of using DQN-based competitive Connect Four playing system, and to further explore reinforcement learning in game-playing applications.

2 Deep Q Networks — Marcin Dąbrowski

Deep Q Networks (DQN) are a type of deep learning model used in reinforcement learning to make decisions in complex environments. They were first introduced in a 2013 paper by Mnih et al.[3] and have since become a widely

[*] architecture, code

[†] editorial, reproducibility, documentation

[3] arXiv:1312.5602

used and influential algorithm in the field of artificial intelligence.

A Deep Q Network is a type of neural network that approximates the Q-function, which is a mathematical function that estimates the expected return or reward for taking a particular action in a given state. The Q-function is defined as:

$$Q(s, a) = E[r + \gamma \max(Q(s', a'))]$$

where

- $Q(s, a)$ — Expected return of $a$ction in $s$tate
- r — $r$eward for taking $a$ction in $s$tate
- γ — discount factor, that is, how much the agent values next rewards
- $\max(Q(s', a'))$ — maximum expected return for taking any next a'ction in the next s'tate

An archetypical Deep Q Network usually internally consists of fully-connected and (de-)convolutional layers to process vectors representing the expected return for each possible action, named Q-values. Training such network involves experience replay, using a buffer of state, action, reward, and transitions to next state, sampled by used agents to update q-values with α learning rate:

$$Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma \max(Q(s', a')) - Q(s, a)]$$

Aside from game-playing capabilities, Deep Q Networks have found use in prototypical informed control applications (e.g. machine-vision-driven robotics) and financial technology (portfolio management, trading strategy planning), mostly due to their inherent abilities for handling multi-dimensional data without feature engineering and resistance to noise and incomplete information sourcing without learning collapse.

3 Google™ OpenSpiel research environment — Paweł Fornalik

Google™ OpenSpiel is a comprehensive library of environments and algorithms designed for research in general reinforcement learning and search/planning within games. It accommodates n-player (single- and multi-agent) zero-sum, cooperative, and general-sum games, covering one-shot and sequential formats, strictly turn-based and simultaneous-move systems, as well as games with perfect or imperfect information. It also supports traditional multi-agent environments like grid worlds (both partially and fully observable) and social dilemmas. OpenSpiel provides tools for analyzing learning dynamics and common evaluation metrics. The games are modeled as procedural extensive-form games with natural extensions, with a core API and game implementations in C++ and interfaces available in Python. Both C++ and Python are used for writing algorithms and tools. [4]

4 Software Architecture — Marcin Dąbrowski

While being based on Google™ OpenSpiel as our backend to ease lower-level operations and specific interations on top of Google™ TensorFlow™, software is intentionally split into separate Python 3.9 (OpenSpiel uses elements that have been deprecated in 3.10) scripts, `connect4_train.py` for training and `connect4_play.py` for inferring, that is playing games via trained models. [5] With parameters received from defaults, environment setup or runtime flags, main program loop in both sides consists of calls to TensorFlow™-based lower layers, with aid of OpenSpiel to keep track of DQN-specific parameters, while the difference comes in training loop versus inference evaluation, that is "game play".

This clean and uncomplicated architecture allowed my colleague to focus on core mechanisms while leaving repetitive components abstracted away by underlying industry proven TensorFlow™ and game-oriented OpenSpiel, while I got to ensure repeatability and reproducibility across hardware platforms and operating systems, which was at one point a major obstacle, requiring a rewrite into current, more transparent and seemingly less ideal setup.

[4] The Github repository of Open-Spiel collection can be found at: `https://github.com/google-deepmind/open_spiel`

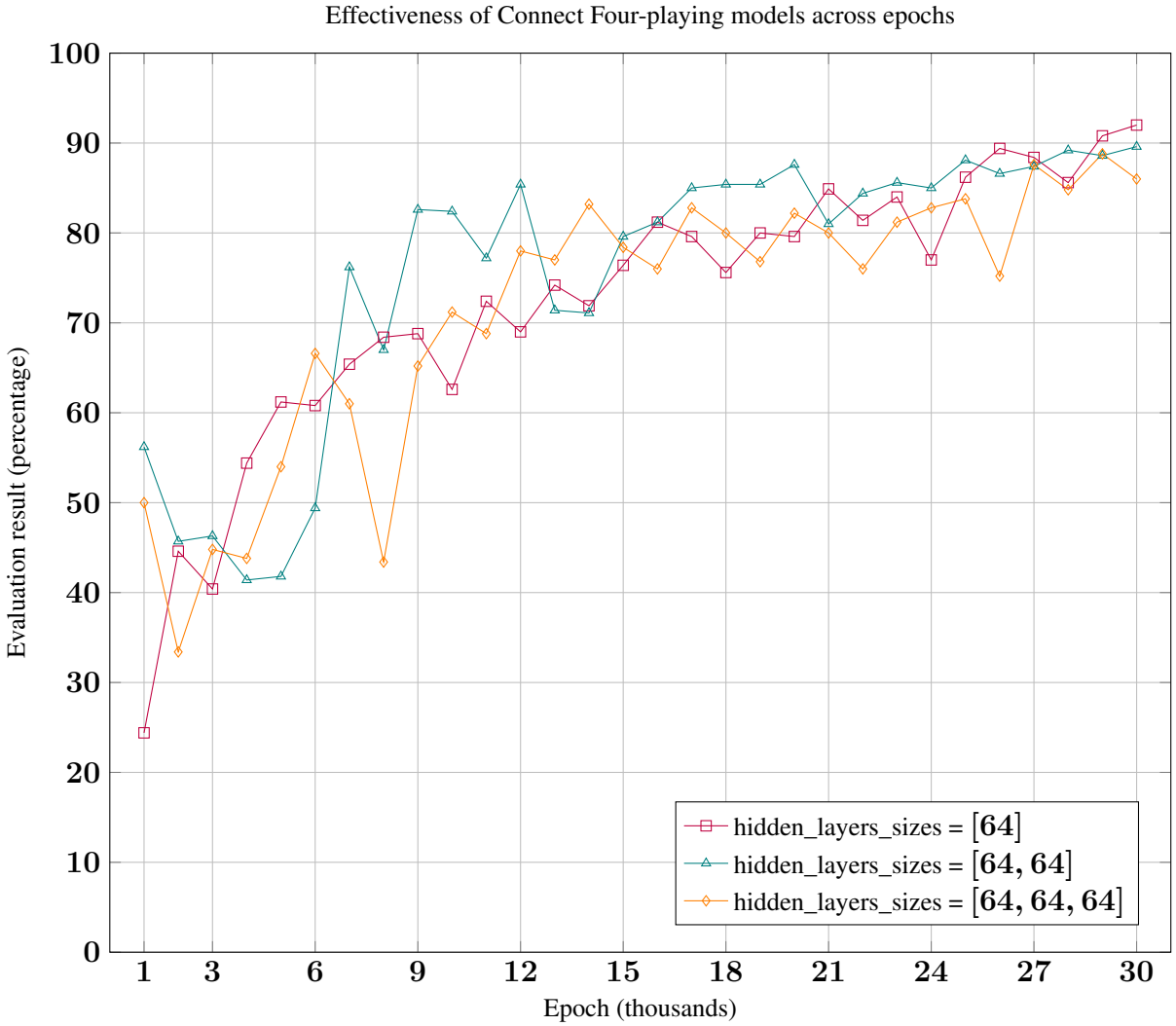
[5] This report does not focus on usage, please refer to `README.md` included in project's repository for further information into usage of these scripts.

5 **Experiment** — Paweł Fornalik

We conducted an experiment to find the best model to play against random bots in Connect 4. We used an environment for this game defined in Open-Spiel and a DQN algorithm, which is implemented in Open-Spiel repository as well. We hypothesized that the most important parameter to test was `hidden_layers_sizes` since it represents a depth and complexity of the network that choses an action of the agent in DQN framework. To be specific, it defines an MLP which is a simple network comprised of linear layers. Three different values have been compared:

1. `hidden_layers_sizes` = [64] — a network made from one linear layer with output dimensionality being 64
2. `hidden_layers_sizes` = [64, 64] — a network made from two linear layers with each output dimensionality being 64
3. `hidden_layers_sizes` = [64, 64, 64] — a network made from three linear layers with output dimensionality each being 64

Other parameters of the DQN have been hold the same across test runs for all three model runs: each one of them has been trained for 30000 iterations (matches played) and every 1000 iteration the model has been evaluated, where evaluation was defined as a percentage out of matches won against random bots out of 1000 games, so 50% meant that learned agent won 500 out of 1000 games. Random bot is an agent that chooses random column every turn. The results of an experiment can be seen in a figure below:



Suprisingly, the model with simplest network had the best results reaching 92% of matches won against random bots at the last epoch. That may be because a complex network isn't needed to learn to exceed in this task and simple

network can do its job perffectly well. Complicating the process can make it actually worse, although we can see that results don't differ very much.

6 Conclusions — Marcin Dąbrowski

With this demonstrative implementation we have shown the successful implementation of a self-playing game of Connect Four using a Deep Q-Network (DQN) architecture. The results show that the DQN agent is capable of learning effective strategies to play the game, achieving an almost monotonic increase in performance, even at single hidden layer.

The analysis of the agent's performance reveals that the DQN is able to learn a robust and generalizable policy, adapting to game scenario. The use of experience replay and target networks seems crucial in stabilizing the training process and improving the agent's performance.

The software demonstrates the potential of DQN-based approaches for solving complex, sequential decision-making problems, while leaving a useable window for selection of hyperparameters to tune for achieving optimal performance easy to access.

Future work could focus on exploring alternative architectures, such as policy gradient methods or actor-critic models, and investigating the application of DQN-based approaches to more complex games, such as chess or Go. Additionally, the development of more sophisticated exploration strategies and the incorporation of domain knowledge into the learning process could further improve the performance of the agents.

Overall, this study provides a baseline evaluation of the DQN-based implementation of a self-playing game of Connect Four, and demonstrates the potential of deep reinforcement learning for solving complex game-playing tasks.