TetrisWorld: Learning a Neural Simulator for Tetris

Project description

World models are neural networks that learn how an environment changes over time from observed state transitions. Instead of relying on hand-written rules, a learned world model predicts the next state of an environment from its current state and an action.

This project investigates world models using Tetris as a compact and well-controlled testbed. Tetris is particularly suitable because its dynamics are deterministic and its ground truth is exactly known, allowing learned predictions to be evaluated precisely.

The student will first implement a simplified Tetris simulator and define a complete state representation including the board, active piece, position and rotation. The simulator will then be used to automatically generate a dataset of state-action-next-state transitions. Particular attention will be given to ensuring that important but relatively rare events, such as line clears, wall interactions and near top-out states, are sufficiently represented.

The student will then develop a small neural network that predicts the next Tetris state from the current state and selected action. Different model designs and state representations may be compared. The learned model will subsequently replace the original game dynamics to create a playable "neural Tetris" demonstration.

A key focus of the project will be understanding how small prediction errors accumulate when the model repeatedly consumes its own predictions. Evaluation will therefore consider both one-step prediction accuracy and multi-step rollout fidelity.

The project is deliberately designed around small models and locally generated datasets so that all core experiments can be completed on a standard desktop or laptop without specialist computing hardware.

Deliverables

- A literature review covering world models, learned environment simulators and forward dynamics models.
- A working Tetris simulator with a documented state representation and action space.
- A locally generated transition dataset with a sampling strategy covering important game events.
- A trained neural forward dynamics model.
- An experimental comparison of selected model architectures, representations or training strategies.
- A playable neural Tetris demonstration in which the learned model predicts game transitions.
- An evaluation of one-step exact-match accuracy and multi-step rollout fidelity.
- An analysis of prediction failures and rule violations during long rollouts.
- Source code, experimental scripts, results and documentation.
- A final dissertation presenting the methodology, results, analysis and conclusions.

Optional extension for a strong student: use the learned world model to search short action sequences towards a simple goal, such as clearing a line.

Suggested timeline

- **Months 1–2:** Literature review, simulator implementation and dataset generation.
- **Months 2–3:** Forward model design and initial experiments.
- **Months 3–4:** Model evaluation and neural Tetris demonstration.
- **Months 4–5:** Multi-step rollout analysis and additional experiments.
- **Months 5–6:** Final evaluation and dissertation writing.

Equipment/Software required to carry out this project

- Standard personal computer or laptop: existing equipment, £0 additional cost.
- Python: open source, £0.
- PyTorch: open source, £0.
- NumPy, Matplotlib and Pygame: open source, £0.
- Git/GitHub: free educational use, £0.

Total additional equipment/software cost: £0.

All core experiments are designed to run on a normal desktop or laptop. A GPU is not required.

Skills developed in carrying out this project

- Understanding of world models and learned environment simulators.
- Neural-network design using PyTorch.
- State representation for sequential decision-making problems.
- Dataset generation and sampling strategy.
- Performance evaluation of predictive models.
- Understanding of compounding error in autoregressive prediction.
- Python programming and simulation.
- Experimental design, data analysis and visualisation.
- Software engineering and reproducible experimentation.
- Scientific writing and presentation.

Skills required to carry out this project

- Programming experience, preferably in Python.
- Basic understanding of machine learning or neural networks.
- Basic mathematical and analytical skills.
- Willingness to learn PyTorch.
- No prior experience with reinforcement learning, world models or GPU programming is required.