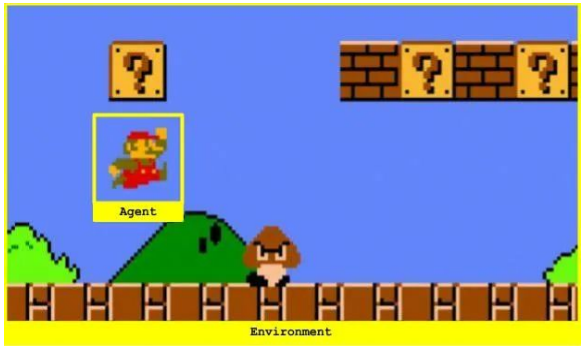
CNN-LSTM Q-Learning for Super Mario Bros

Luke Luo, Andrew Chavarria, Hari Shanmugaraja, Abdulrahman Azizi, Varun Lagadapati

Mario RL agent with “revamped” Deep-Q network architecture

Introduction: Original agent uses a regular CNN network to process a single frame of play at a time to make an informed decision in the game, at each frame.

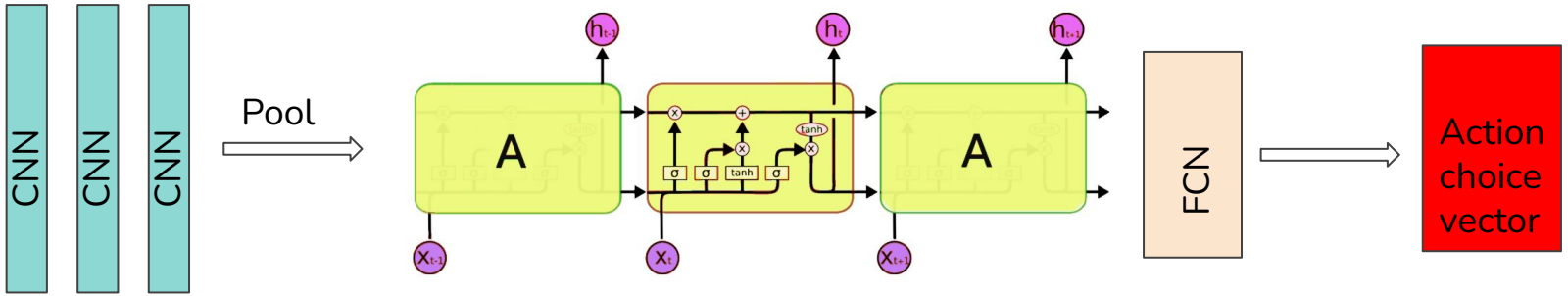
Goal: Renovate the existing Deep-Q architecture to use a CNN-LSTM, which uses the current frame, as well as previous frames, for context to make a decision in the game. The objective is to complete as many levels as possible, using the new architecture.



Approach

At its core, Super Mario Bros., and most video games, have infinitely many possible states as opposed to board games. The agent must decide for itself what actions to take based on observable details of the state

- Double Deep Q learning
  - Use CNNs to extract feature data from game states.
  - Use LSTMS to, in theory, consider past sequences of game states. The current baseline uses a random recall, but we want to attempt to combine it of the memory of recent states.



LSTM Layers

Procedure

1. Set up actor critic model, and obtain a working policy using only CNNs to extract feature data from the game state.
2. Reformed Mario RL's original CNN architecture into a separate CNN class
3. Implemented LSTM architecture which accepted timesteps and batch size as parameters
4. Combined CNN and LSTM models, to form a CNN-LSTM architecture
5. Configure VM to support our model to train, using Google Cloud Platform
6. Train the model and record the learning via mean Loss, Reward, Length, and Q Value.

Issues encountered

1) CNN LSTM implementation is not “PyTorch-friendly”. Existing implementations, which use TimeDistributed, are used with the Keras framework. TimeDistributed wrapper allows to apply a layer to every temporal slice of an input.

Resolution: Used .view() on tensor to combine batch and timestep before running convolutions. .view() allows us to view the tensor without making an explicit copy, making it memory efficient.

2) Reinforcement learning takes an extremely long time, about 4 minutes per 20 epochs. Adding LSTMs made it about 5.5 minutes per 20 epochs. Models needed at least 1.5 days to train for less than ¼ of the recommended performance training epochs. Every training session takes an extremely long time to compare performance to previous benchmarks, especially due to limited compute resources.

Resolution: Reduced buffer sizes by 90% to barely fit in memory, set up multiple VMs hosted by GCC to run algorithms.

Expected / Future Work

- **Expected**
  - Finish training the CNN-LSTM model. We expect at least comparable, if not higher, performance to the base model. Due to limited computing resources, it will be difficult to make the model learn / generalize to further levels.
- **2 weeks**
  - Run more comprehensive training - more epochs, larger buffer, stronger hardware
  - Implement moonshot training paradigms like random level selection
- **2 months**
  - Train versions of the model on different games to see what it's suited to
  - Compare performance to genetic algorithms
- **1 semester**
  - Create an architecture using Soft Actor-Critic
  - Attempt level regeneration using LSTMs and try to evaluate on them
  - Evaluate performance on custom levels
  - Other wider range of architectures like GANs tested, and over more levels

References

- Kauten, C. (n.d.). *Gym-super-mario-bros*. PyPI. Retrieved May 4, 2023, from https://pypi.org/project/gym-super-mario-bros/
- Musib, A. *Introduction to Q learning and reinforcement learning*. 2019.
- Yuansong Feng et al. *Train A Mario-Playing RL Agent*. 2020.