CNN-LSTM Q-Learning for Super Mario Bros

Abdulrahman Azizi
University of Maryland, College Park
arazizi@terpmail.umd.edu

Andrew Chavarria
University of Maryland, College Park
andrew.chavarria77@gmail.com

Varun Lagadapati
University of Maryland, College Park
VLagadapati7@gmail.com

Luke Luo
University of Maryland
lluo1@terpmail.umd.edu

Hari Shanmugaraja
University of Maryland, College Park
hshanmug@terpmail.umd.edu

Abstract

This paper presents an improved Convolutional Neural Network (CNN) with Long Short-Term Memory (LSTM) architecture for training a Mario RL agent. The CNN-LSTM model uses current and previous frames of gameplay to make informed decisions. By incorporating LSTM, the model gains contextual information and improves its ability to recognize game elements like Spike Traps. The agent is trained on individual levels of Super Mario Bros using a gray-scale representation of the game environment. Experimental results show that the CNN-LSTM model outperforms the baseline CNN model by achieving higher completion rates and shorter time steps. However, memory constraints and the limitation of right-restricted actions were encountered during training. Future work includes running more comprehensive training, exploring genetic algorithms for comparison, and testing other wide range of architectures over more levels.

1 Introduction

An original Mario RL agent exists using a regular CNN with 3 convolution layers, along with two linear layers. This CNN network to process stacks of 4 frames of play at a time to make an informed decision in the game.

The goal of our project is to implement LSTM with CNN, to create a more competent Mario RL agent. The CNN-LSTM model uses the current frame, as well as previous frames, for context to make a game decision.

We chose CNN LSTM because they were developed for visual time series prediction problems. Being able to recognize activities and images better should in theory improve the decisions of our agent when playing the game. For example, if the agent can detect Spike Traps better, our agent can do a better job of taking the right action to avoid them. CNN LSTMs are simple to implement in PyTorch using nn.LSTM.

The goal of the project is to train the agent to complete the first level of Super Mario Bros. as fast as possible.

37th Conference on Neural Information Processing Systems (NeurIPS 2023).

2 Data

The nature of the model is to make observations in the environment of the game (which is represented by a frame of the game), so we use these frames to feed to the agent for it to make a decision.

We used a [4,84,84] size array to represent the image of the environment in which our agent is playing. The array which represents the image is more information than what the agent needs to perform well in the game, so preprocessing of our data includes converting our image to a gray scale ([1, 240, 256] array), and downsampling observations into a square image [1]. GrayScaleObservation and ResizeObservation were PyTorch wrappers used in preprocessing the environment.

In the CNN-LSTM implementation of the game, what changes is not the contents of the data itself, but rather, how many instances of it. We feed the agent current frame, along with the previous 9 states, totaling 10 per step-iteration. The purpose of adding previous states is to add context of Mario's progress to make more rewarding decisions that will ultimately lead to higher performance.

3 Related Work

Deep reinforcement learning is a very common technique to create AIs that play games, and therefore there are many well-established baselines that we hope to expand on. While we do not have a specific paper that provides a baseline for playing Super Mario Bros, relevant papers for the baselines we hope to improve upon include "Playing Atari with Deep Reinforcement Learning" by Mnih et al. (2013) [3] and potentially "Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor, Haarnoja et al, (2018)" [2]. We also have a pytorch tutorial (https://pytorch.org/tutorials/intermediate/mario_rl_tutorial.html) [1] that does not use an LSTM or actor-critic approach that we can use as a baseline to compare our performance to.

3.1 Deep Q Learning with Atari

Deep Q networks were largely popularized by their use in solving various Atari games. In "Playing Atari with Reinforcement Learning", researchers used a Deep-Q-Network (DQN) to play Atari games such as B.Rider, Breakout, Enduro, and Pong and achieved S.O.T.A results for the time [3]. Deep Q networks simultaneously allow optimization of uncountably large state spaces by optimizing an approximation to the true Q value using gradient descent.

3.2 The Pytorch Tutorial

This resource is our baseline to beat. This tutorial, which we set up to ensure that our environments were in working order, incorporates a double deep Q learning algorithm. The training network uses three CNN layers to extract feature data from 4-frame stacks to process states. We hope to see whether our changes, detailed in section 4, will provide results that exceed the ones shown in the tutorial [1].

3.3 CNN-LSTM Architecture

Combining CNNs with LSTM layers is not a unique type of model. In a related work, combining the spatial and temporal learning ability of CNNs and LSTMs were used to study the performance of penetrative passes in soccer analytics and outperformed both networks individually in doing so [4]. While not specifically aimed at video games, we were interested in the model's ability to capture spatial trends across time and attempted to apply the architecture to our deep Q network.

3.4 Level Generation

Summerville and Mateas (2016), discuss the automated generation of levels for Super Mario Bros. using LSTMS [5]. This is methodology could prove useful in creating an application for self-supervised reinforcement learning. If we were able to train an LSTM to generate levels like described in this paper, we could then run our deep Q network on the generated levels. This self-supervised training solution would working something akin to GANS, assuming we could ensure the quality of the generated levels.

4 Your Approach

Since the state space for Super Mario Bros., as is for almost all video games, is continuous, the standard approach to training reinforcement learning models is to optimize a Deep Q Network to approximate the Q values of given game states to a target network to find optimal policies through Double Deep Q Learning. The DQN learns by sampling batches of experiences from its memory buffer and using them to train the Deep Q Network.

Our main experiment was modifying the architecture of the network to include an LSTM layer that takes the context of the previous 9 time steps. The previous nine frame stacks are kept in memory to the current state to create a continuous context for the current action, in addition to randomly sampling trajectories from the memory buffer. This is distinct from changing the size of the frame stack, as changing the rate of frame skip would make the agent attempt to input a command once every 40 frames instead of every 4, which is not what we wanted. This lead to the input to the deep Q network to be a [32x40x84x84], where 32 is the batch size and the other three dimensions represent the ten stacks of [4x84x84] frames. The frames are fed through the CNN layers to extract features. The extracted features are then max pooled, stacked again and then flattened into [32x10x576] to be passed through the LSTM block, which uses 10 LSTM layers. The final fully connected layer maps the output of the LSTM block to the estimated Q values for taking any of the possible actions in Mario's allowed action space. The recall network was also modified to sample states in contiguous batches of 10 to adapt to the updated architecture.

In the end, we did not elect to use our 5 iteration evaluation method outlined in the proposal in order to save time on the training. Under that proposal, the amount of training time would have been increased by approximately 11 to 15 percent due to adding extra evaluation specific epochs. Hyperparameters for the model were also not tuned, due to the excessive time requirement for training multiple models to compare hyperparameter performance.

5 Experiments & Results

5.1 Baseline Model

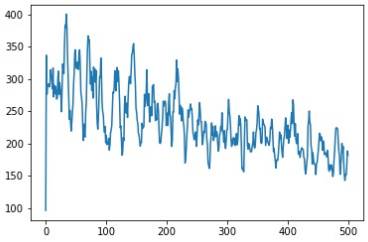

Figure 1: Average length of epochs for baseline CNN: Stacks of 4 frames per 20 epochs

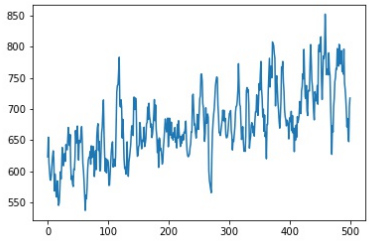

Figure 2: Average reward over batches of 20 epochs for baseline CNN

The main constraint that we encountered during model training was memory usage. The way the architecture works, the double deep Q learning algorithm requires that a number of Mario's past states be stored in active memory in order to access them for comparisons to update the Q value function. This ends up taking up a massive amount of memory and necessitates the use of a limiter on the memory buffer. However, this also means that Mario's Q table is treated as a cache: Because of the dynamic nature of the game, the number of possible states to keep track of is completely unreasonable to keep track of. Even in the context of a small part of the first level, the same state may take many iterations on average to be visited multiple times. Hard to reach or lesser reached states, such as ones from further in the level or remote locations like on top of blocks, can be 'forgotten' if they are not refreshed in the cache soon enough. Due to the limited architecture we had at our disposal, we had to reduce the size of the memory buffer to prevent the GPU from running out of memory while training. This resulted in the lookback of the model to be reduced by roughly 75 percent of the recommended 100,000 states, to roughly 60-80 iterations' worth of attempts. Performance of

the baseline model started to plateau about 60 percent of the way through the level on the CNN architecture. This is the base performance that our updated model will compete against. On average, the base CNN model took approximately 896 frames, on each iteration by the end of the training phase and was not observed to complete the stage with reasonable performance.

5.2 CNN-LSTM

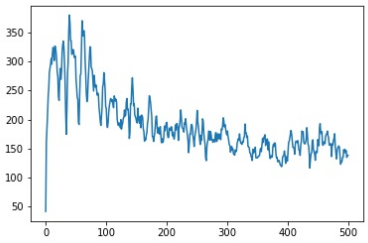

Figure 3: Average length of epochs for CNN-LSTM: Stacks of 4 frames per 20 epochs

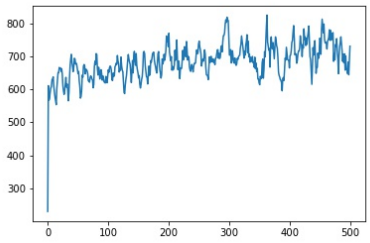

Figure 4: Average reward over batches of 20 epochs for CNN-LSTM

The CNN-LSTM modified version of the model showed higher levels of retentiveness across a given action space, which, for the sake of training consistency, was restricted to right movements only. We observed that it made him more likely to hold buttons down, allowing him to more consistently pass obstacles. One of the primary issues that we observed with the base model was that it would constantly get stuck on the taller blocks because the agent kept inputting short jumps instead of high jumps, which require the jump button to be held down. The CNN-LSTM model managed to clear the level for the first time at around 1600 and thereafter averaged one clear per 1000 epochs, gradually increasing to approximately one per 250. The average time steps per iteration were noticeably lower as well, reflecting that Mario spent much less time stuck behind harmless obstacles and timing out when we compared performance of the models playing the stage. He was also observed as being able to pass the 60 percent mark of the level much more consistently than his predecessor. Episodes took, on average, 600 frames at the end of the training phase.

5.3 Less Restricted Action Space Models

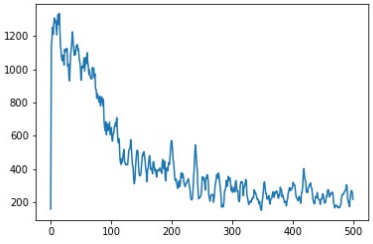

Figure 5: Average length of epochs with Mario allowed to move left: Stacks of 4 frames per 20 epochs

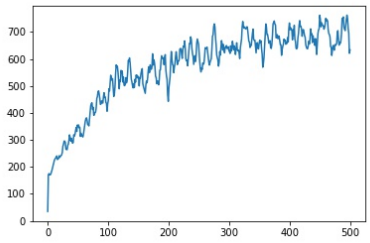

Figure 6: Average reward over batches of 20 epochs with Mario allowed to move left

Several models were also trained with alterations to Mario's allowed actions. In tuning them, it was found that the best performing models we could train with our resources were ones in which Mario's action space was restricted to actions which included moving right. Without this restriction, episodes would consistently time out, increase the overall training time by about two to six fold in early epochs, and necessitate many more training epochs to reasonably optimize out left movements. A CNN model trained with the model allowed to move left took around 2,000 iterations to reach baseline performance of right-only and plateaued lower than they did. The 'left movement' in this case refers to Mario being allowed to press 'left', but not in conjunction with any other buttons, as

this would cause the agent to select 'left' much more often while getting out of the more random phase of the training epochs. A model with unrestricted action space is trained and detailed below. As a side note, even when allowed to press down as a specific motivation to attempt to access the hidden area via the pipe, Mario never inputs the action.

The instability of reinforcement learning mainly showed in Mario seemingly 'forgetting' sections of the level that he had not been stuck on for some time. Since we reduced the size of the memory buffer, it would be much harder, if not impossible, to effectively train the more complicated models.

5.4 Unrestricted Action Space Models With Increased Buffer Size

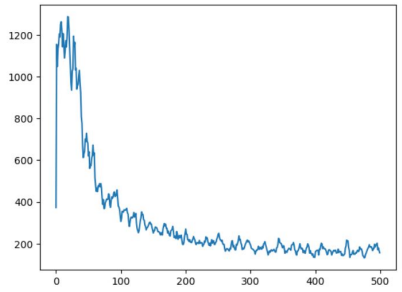

Figure 7: Average length of epochs with unrestricted Mario movement and increased buffer size of 50,000. Stacks of 4 frames per 20 epochs

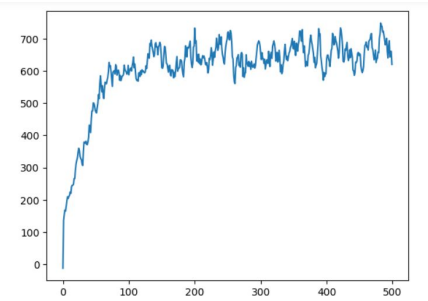

Figure 8: Average reward over batches of 20 epochs with unrestricted Mario movement and increased buffer size of 50,000

To address the issue of Mario "forgetting" actions on an unrestricted model, we increased the memory buffer from 25,000 to 50,000, which allowed Mario to remember more states. However, storing such a large amount of tensors required more resources than were available on our local GPU. As a result, we trained the fourth model on virtual compute servers.

Despite starting off with the same drawbacks and timeouts associated with unrestricted movement, increasing the memory buffer had a significant advantage. It allowed the model to remember "better" choices more effectively, which led to faster drops in episode lengths and faster reward gain. However, once the model passed 50,000 states, it began to plateau, and older states were removed. Nevertheless, upon completion of training, the model was able to average one completion every 175 runs, beating the right-restricted CNN-LSTM completion rate of one per 250.

6 Conclusion & discussion

6.1 Technical Limitations

To train our model with the increased memory buffer, we had to explore several hosting platforms to find one that was both free and stable, with more than 8 Gigabytes of memory available (beyond our local GPU). We experimented with Google Cloud Platform (GCP) and quickly discovered that VMs were not beginner-friendly. GCP's VM container runs on Debian GNU/Linux 11, which required updated dependencies to compile and train our model with libraries such as nes_py and gym-super-mario-bros.

After putting many hours into this process, we realized that a spot VM would not work for our training due to frequent interruptions. We needed a standard provisioning model for the VM, but GCP did not offer the capability of migrating from Spot to Standard or creating a compatible machine image to switch over.

We eventually created a Standard VM from scratch and installed all necessary libraries, only to discover that the kernel still timed out and died halfway through training. It became clear that we needed a permanent solution, and there was not enough time or patience to continue debugging GCP errors.

Fortunately, some of our team members had access to servers hosted by the University of Maryland Institute for Advanced Computer Studies (UMIACS). With the knowledge acquired from debugging GCP issues, we were able to set up our notebook successfully on the compute nodes, which provided 15 Gigabytes of GPU memory and 30 Gigabytes of RAM to train models with increased buffer sizes.

The final limitation was the memory buffer tensors, which were contiguous and could only fit on one 15GB GPU. With more time, we could have used data parallelism to utilize multiple GPUs and train with the recommended memory buffer of 100,000, rather than our maximum of 50,000 on VMs.

6.2 RL Results

From our findings, we found that adding an LSTM component to the model does show improvement in performance. Overall, the limited amount of training we were able to achieve compared to the recommended baselines reinforces the idea that RL as an algorithm is extremely unstable and unpredictable. The agent would often 'forget' experiences it hadn't recently seen in the memory buffer, leading to repeated deaths in early parts of the stage after progressing into the later half of the stage over several hundred epochs. We anticipated the same to extend to other complex action models, though our unrestricted action model trained on a memory buffer twice as large did show unexpectedly competitive results.

There were several instances where Mario was shown to perform wall jumps, which are frame perfect inputs. However, it appears to be by coincidence and used randomly to recover from mistakes, rather than deliberately. He also happened upon a glitch where he is able to clip into walls and cause the screen to scroll by a few pixels. It is also used professionally, but again, Mario's usage of it is purely coincidental.

7 Future work

If we had 2 more weeks, we could run more comprehensive training with more epochs, larger buffers with data parallelism, and better usage of VM resources. We could also implement moonshot training paradigms like random level selection. If we had 2 more months, we could train versions of the model on different games like Tetris and Snake to see how well the architecture generalizes to new problems. It would also be interesting to compare the performance of our model to more traditional reinforcement learning methods like genetic algorithms. If we had 1 semester, we could create an architecture using Soft Actor-Critic and attempt level generation using LSTMs. Having the model evaluate on and train on the custom made levels would increase the robustness of the network. We would also consider testing a wider range of architectures like GANs or diffusion models to generate levels for us.

8 Project presentation link

`https://docs.google.com/presentation/d/1r_em_Hh7KXOuAzjQdeYJ4M8dSDzvkaD561FkcCVices/edit?usp=sharing`

References

[1] Yuansong Feng et al. *Train A Mario-Playing RL Agent*. 2020.

[2] Tuomas Haarnoja et al. *Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor*. 2018. arXiv: 1801.01290 [cs.LG].

[3] Volodymyr Mnih et al. *Playing Atari with Deep Reinforcement Learning*. 2013. arXiv: 1312.5602 [cs.LG].

[4] Pegah Rahimian et al. "Let's Penetrate the Defense: A machine learning model for prediction and valuation of penetrative passes". In: Oct. 2022.

[5] Adam Summerville and Michael Mateas. *Super Mario as a String: Platformer Level Generation Via LSTMs*. 2016. arXiv: 1603.00930 [cs.NE].