Applications of Neural Networks for Real-Time AI in Video Games

By Tobias Hendricks

Part 1

Introduction

Neural Networks have been used for a variety of applications in recent times; while their use in video games has been numerous, but limited in scope. We very rarely see Neural Networks (NNs) used in commercial video games despite the advancements in learning technology. Issues such as low performance, limited complexity, and limited control, can make NNs a daunting choice for artificial intelligence (AI) systems in video games. This project investigate ways in which Neural Networks can be used for real-time AI in video games without sacrificing performance, control, or simplicity.

Current Methods and their Issues

The typical current method for implementing NNs in video games is as follows (Flappy bird AI, https://bit.ly/2koKNN9). Every frame (or every few frames) data is fed into the NN. This data represents the AIs perception of the game world. Once the NN receives its input, it will generate an action to perform. These actions often correlate to the most basic functions within the game (for example move left/right, jump, shoot). I'm going to call this method an Atomic Action Neural Network (AANN), as they're used to produce the most basic actions in a game at high-frequency. The primary issue is that these atomic actions, by themselves, are largely meaningless unless performed as part of a meaningful sequence. This is the root cause behind the low performance, limited complexity and limited control of AANNs.

NNs have found their way into video game AI, but because of these limitations they either require very powerful computers or can only be applied to very simple games. A few popular games such as StarCraft and Dota have had complex AIs built by DeepMind and OpenAI to challenge the greatest human players, but the compute power required to run these AIs vastly eclipses that of a traditional game console or home computer. At the other end of the spectrum, non-commercial communities have used systems with limited compute power to play simple video games; game development communities on social networks such as YouTube have found success training NNs to play games like Snake and Flappy Bird. However, performance issues arise when the games become more complex. These two sides are dodging the same issue - that their method is extremely performance heavy, making it problematic for commercial video game AIs. As the complexity of a game increases, the intelligence of the AI needs to increase with it. A more complex game requires longer, more complex sequences of decisions to play, decisions that often need to be trained on even more data. The reason that these AANNs experience performance issues is that every few milliseconds, a new action must be generated, resulting in low performance. This method also results in behavioural complexity being inherently limited by compute power. If the game is too complex, the NN may never figure out how to play the game. If the AI needs to generate long sequences of precise actions to exhibit intelligent behaviour, there's a chance that the NN will never discover the right combination of actions.

Lastly, using AANNs results in limited control over the AI's behaviour. Unless the AI only experiences situations identical to its training (unlikely in complex video games), there is no guarantee that the AI will

behave in a desired manner. There's a chance that the training will be ineffective, causing the AI to make bad decisions. On the other hand, the AI may learn to become too good at playing the game. Both can result in a poor experience for the player. Video Game AI must strike a balance, if the AI is too easy to defeat the player gets bored, too difficult and the player gets frustrated, too stupid and the player loses their immersion. This balance is difficult to strike without a direct method of tweaking the AI behaviour.

Approach

As discussed earlier, the main issue with the current usage of AANNs is that the atomic decisions they generate are meaningless unless part of a meaningful sequence. In this project, instead of asking NNs to frequently generate AANNs (a sequence of independently meaningless actions), I propose that we use NNs to generate more meaningful decisions, less frequently. These decisions require an algorithm or system in order to translate that meaningful decision into a sequence of atomic actions. However, because such algorithms can be simple hand-coded heuristics for a specific goal, it has the potential to be faster than frequently using the AANNs.

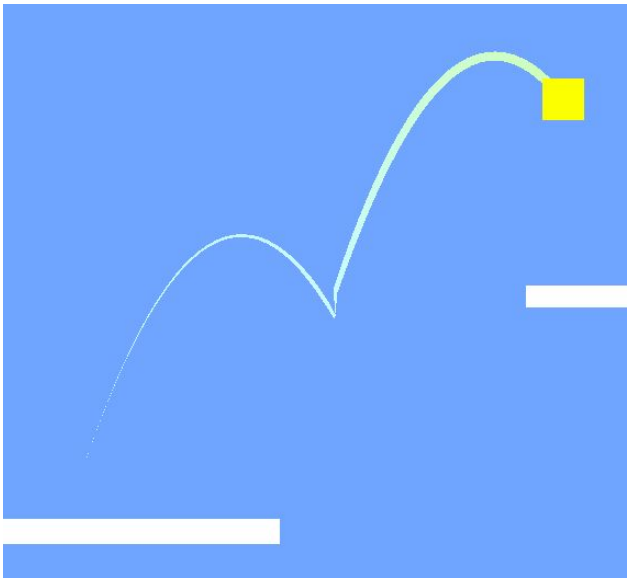
A meaningful decision could be a plan for the AI to execute, it could be a behaviour to enable, or a goal to achieve. I'm going to call this approach a Behaviour Control Network (BCN), as it uses a NN to control some sort of predefined behaviour. Breaking down the AI system like this allows a developer to evaluate each decision that the AI can make, and maximise the performance of that decision. In addition, as the developer now controls how the AI handles that decision, they're able to retain a level of control, preventing any undesirable behaviours by altering how decisions are handled, or by dropping undesirable decisions altogether.

Experimenting with the Method

To evaluate the method over the first research period I designed two contrasting experiments that focus on using NNs to make infrequent, meaningful decisions instead of frequent, independently meaningless decisions.

Traversing a Gap

The first experiment I designed was to train a NN to become proficient at traversing a ledge gap in a simple 2D platform game environment. The AI would have to navigate from the start ledge on the left, to a randomly placed end ledge on the right. The AI has the ability to move left, move right and jump (from the ground and once in mid air). Initially, I attempted to train the agents using a traditional AANN, every 5 frames the NN would decide whether to move left, move right and or jump, however, my success was limited in this case. As this problem required a relatively specific sequence of actions to solve, I found training to be problematic. I was able to train a single batch that could perform moderately well; however, I was not able to recreate these results. I believe that this is would be possible to reproduce this result with more time, thankfully, this was not the focus of the experiment.



Using a BCN to traverse the gap

Instead of repeatedly asking the NN whether to move left, move right or jump, I applied my BCN approach to use the NN to generate a set of coordinates, indicating where it should jump. Following this, I wrote a simple hand-coded behaviour that would navigate the agent to those coordinates, jumping when it came close to each coordinate. This yielded far better results, with the AIs quickly learning how to place their coordinates in such a way that allowed them to traverse the gap. In addition to this, performance was far better.

To train the BCN I simulated 150 parallel agents. During training and evaluation the BCN was able to generate coordinates while maintaining a frame rate above 60 frames per second. Using the AANN resulted in frequent frame drops below 15 frames per second when simulating the same number of units. The coordinate system representation had a number of other benefits too. For the AANN to work, lots of data must be observed by the NN; this includes the agents relative position to both the start and goal ledge, whether the agent is on the ground or not, and if it has performed its mid air jump. As the coordinate system is simply generating a plan, the only observations it makes are the position of the end ledge relative to the goal ledge. As this is the only data we need, the AI doesn't need to be present to generate the coordinates, this could even be done before the level starts, with coordinates being reused whenever a gap needs to be retraversed. In this context using a BCN proved far more efficient and effective than an AANN, the BCN was able to yield good results without impacting performance.

Video Log: https://bit.ly/2kuraU8

Survival

The second experiment was designed to focus on a NN's decision making capabilities. I created a simple survival game that would put decision making to the test. The AIs objective is to gather sufficient food, water and gain enough rest to survive. In addition to this, monsters will come out at night, which can only

be repelled by fire. Every hour the BCN chooses a behaviour to execute, executing the wrong behaviour, or the right behaviour at the wrong time will often result in death and failure. To survive, the AI must learn what it needs and what actions are available at any given time.

The world that the AI lives in

As the goal of the project was to focus on decision making, I was able to make the game as simple as possible. The methods by which an agent might collect water in video games varies; in this project I focus on ensuring the Network knows it needs (and is able to get) water, the *how* is handled by the hand-coded behaviour.

My first approach was simple, to have the Network observe the current situation and generate a plan lasting a full in-game day. I focused on this method for 4 weeks of my research period, designing variations and tweaking parameters but overall I had little success. The AIs would often learn to execute a small fixed sequence of behaviours that guaranteed survival for 2 in-game days while neglecting other needs which would inevitably prevent them from living longer than 2 days. The issue was that asking a single NN to generate an entire plan based on the AI's situation at the start of the day, and a basic memory of behaviours it had previously done, was too difficult. It was unlikely the NN would learn the effect of each behaviour.

I concluded that the AI system would be more effective if it had two different NNs, trained for different, specialized, purposes. One NN would be responsible for choosing a single behaviour based on a given situation, and another would be trained to predict the resulting situation based on the completion of that behaviour. This could potentially be a far more effective planning technique. Due to time constraints I was only able to evaluate half of this method during the Summer research period. I focused on training a Network that was capable of choosing behaviours based on given situations. Every in-game hour, the Network observes the time and health condition of the player and chooses a behaviour to perform. Using this alone has proven to be far more effective than my previous attempt with the AIs learning how to survive over 100 days. The trained model was capable of making smart decisions based on its observation of the world. The AI quickly learned when not to go out, when to light a fire and when to hunt. It found a routine allowing it to maintain its food, water and energy levels. This shows that we can use NNs to successfully make meaningful decisions.

Video Log: https://bit.ly/2koLbv5

Possible Training Methods

A benefit of using Neural Networks is the large number of ways they can be trained. Reinforcement learning (RL), a method by which the NN is scored on its performance and alters itself to maximise its score, has been proven to be effective, however it has its drawbacks. If the AI needs to be trained to interact with players then large amounts of playtesting may be needed in order to successfully train the Network. Alternatively, the reinforcement Network could be trained against another AI, perhaps another Neural Network, however, the other AI may need to be programmed or trained to act like player for this to be effective.

An alternative to using RL would be be able to be trained directly from players using backpropagation, a method of providing the NN with examples of how to perform and allowing it to imitate the human players. However, some sort of behaviour analysis system may be needed inorder for the AI to identify player behaviour (perhaps this could be achieved using NNs).

Using a curriculum of different training methods may be the most effective way. Perhaps you could start by training the AI with real players, either by using reinforcement learning or backpropagation. This would ensure that the behaviour is somewhat predictable. Following this, you could use multiple AIs to train each other, allowing them to gain more training time. Post release you could then return to using real players to train the AIs by collecting data from active players.

Limitations

One major limitation of BCNs is that they may have trouble coping with games that contain a large number of situations with specific solutions to those situations. If the player performs an action and the AI needs to respond with a specific action or sequence of actions, then training a NN to recognise and respond to these may not be the best solution. For example, in an open-world city game the player may find themselves being pursued by an enemy, and decide to get into a car to evade them, the only way the AI could possibly keep up with the player is by finding some sort of vehicle to continue the pursuit. Using NNs in the ways discussed maintains a level of control, however, due to the nature of Machine Learning, some uncertainty will always be present. Considering the example above, without developer intervention, there’s no way to be 100% certain that the AI will decide to find a vehicle every time this situation occurs. As a result, this system is best utilised where there are multiple acceptable solutions.

Another limitation is the method of training. The NN will either need an environment to train in or data to train from, the exact obstacles involved will vary from project to project, some applications of this system may have too many obstacles for the implementation to be worthwhile. This can be a problem for AIs that don’t have a specific goal, as many training methods rely on the ability to score an AIs ability to complete some task, the lack of a well defined goal may make use of NNs unfeasible.

Conclusion

I have shown that Neural Networks can be effective behaviour control systems in video games, and how using them in this way can prove just as effective as using them in traditional ways, while providing other benefits such as better performance and increased control. This does have its limitations, as there is no guarantee which actions the AI will perform in every case, however, it can still be more adaptable than hard-coded behaviours. With the benefits that Machine Learning and Neural Networks provides, a BCN could be a viable approach to use for AI in video games.