university of groningen / faculty of science and engineering / bernoulli institute

Incorporating a Distance Metric to Induce Safe Behavior in Super Mario Bros using Deep Reinforcement Learning

Bachelor's Project Thesis

Viktor Milenov, s4310861, v.milenov@student.rug.nl
Supervisors: J.D. Cardenas Cartagena, M.Sc., j.d.cardenas.cartagena@rug.nl

Abstract: Safe Reinforcement Learning (Safe RL) is a sub-branch of machine learning and data-driven algorithms that strives to guarantee safe performance in a system while optimizing its performance efficiency. To experiment with such algorithms and models we use video games, specifically Super Mario Bros, which offer environments where faulty behavior does not lead to serious real life consequences, allowing for the modification and improvement of such algorithms and models. We employ the Actor-Critic method because of its effective and stable training procedure, which permits learning a value function and a policy at the same time and directly modifies the gradient descent direction to guarantee system safety. Our model consists of an actor network and an ensemble of critic networks to obtain a more accurate value function approximation. Furthermore, we incorporate a distance metric that stands for the distance between our agent and the closest danger in the environment. We compare the performance of this safer model with a baseline model to assess if the distance metric indirectly induces safe behavior of the agent in the environment. We are investigating ways to improve an agent's training to make it more "safe".

1 Introduction

Individuals and professionals alike must make decisions every day that could have a big impact. These choices can have very different possible outcomes and degrees of importance. Think about a cybersecurity analyst employed by a major company, for example. This person must decide how to react in the event of a potential security breach. A poor choice here could result in significant data loss, financial harm, and a decline in the public's confidence in the organisation. Decisions have a varying degree of importance and relevance which depends on our circumstances. On the other hand, an appropriate and timely response could prevent a major crisis. There is a clear contrast between this situation and decisions we make on a daily basis, like whether to carry an umbrella. The stakes are much higher in the context of cybersecurity, and a bad choice can have far-reaching effects that could impact thousands or even millions of people in addition to the individual. These high-stakes decision-making situations highlight the value of good judgement and the significant influence that decisions can have in specific crucial and professional situations.

Furthermore, the medication given by a doctor to their patient can impact the patient's physical well-being positively or negatively. In this second example, the consequences are still relevant due to the fact that an entire life will be affected. Therefore, we can see that the decisions we make depend on the situation we are currently in, and that the consequences of our decisions can lead to unwanted situations.

Decision-making gets increasingly more complex when we realise that, normally, there is more than one specific decision that we could make in any situation. For instance, the doctor has to be aware of the positive and negative effects of the medication that they are going to give to their patient and then decide which one of them will be the most beneficial.

To make decisions, we need to use knowledge about the world, which serves as guidance in our decision-making process. There are many theories on how this knowledge is acquired, but we are going to focus specifically on the trial-and-error method of learning introduced in the book *Animal Intelligence: An Experimental Study of the Associative Processes in Animals* by Thorndike (1898). The trial-and-error method of acquiring knowledge states that learning occurs when trying out different approaches to something until one finds the most effective one. This method alone can be used in various domains to test its efficacy and one such domain is the ever growing paradigm of artificial intelligence (AI).

2 Theoretical Framework

In this section the necessary theory for understanding the goal of this project is presented. Firstly, we introduce reinforcement learning (RL) as the underlying branch of machine learning (ML) that we will be using and give a brief explanation as to what actor-critic methods are. We then move onto deep reinforcement learning (DRL) and its role in video games and present some of the current achievements obtained. Finally, we introduce safe reinforcement learning as an extension to RL that we will be focusing on in this project.

2.1 Reinforcement Learning

A branch of machine learning methods known as Reinforcement Learning (RL) encapsulates the idea of trial-and-error learning according to Whitehead and Ballard (1991). To be consistent, the decision-maker will be called an **agent**, the decisions that an agent can make will be referred to as **actions**, and the situations that the agent finds itself in will be referred to as **states**. The entire **environment** can be thought of as the place that contains all of these states. It can be represented by a Markov Decision Process (MDP) introduced by Bellman (1954) which is a discrete-time stochastic control process consisting of a 4-tuple $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R})$ where $\mathcal{S}$ is the state-space, a set of all states the agent can be in; $\mathcal{A}$ is the action-space, a set of all the actions the agent can make in a given state; $\mathcal{P}(s, s')$ is the transition probability of ending up in state s' from state s; and $\mathcal{R}(s, s')$ is the immediate reward obtained from moving from state s to state s'. When an agent is in a specific state it will have the opportunity to select a single action from a set of possible actions that will bring the most reward in the future. More concisely, the key idea of RL is this: while interacting with an environment, the agent seeks to maximise the total amount of reward it receives, Sutton and Barto (2018). This interaction between the agent and the environment is represented in Fig. 2.1:

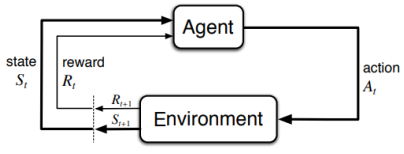

Figure 2.1: This diagram shows the interaction between the agent and the environment where the agent is constantly acting upon the environment. These actions obtain some reward and change the state the agent is currently in.

A policy $\boldsymbol{\pi}(s, a)$ is a mapping from state to action that tells the agent how to act according to the given state. Ultimately, we want the agent to select the best action according to the current state and to achieve this there are two main approaches: value-based methods, and policy-gradient methods. The former utilizes the concept of a value function which represents the expected cumulative reward that an agent can achieve by starting from a certain state, adhering to that policy throughout the agent's interactions with the environment, and achieving that benefit over time. The value function is given by:

$$v_\pi(s) = E_\pi[\sum_{t=0}^{\infty} \gamma^k R_{t+1} \mid S_t = s] \tag{2.1}$$

where E_π denotes the expectation under policy π, $R_{t+1} \in \mathbb{R}$ is the reward obtained at time step $t+1$, and $\gamma \in (0, 1]$ is the discount factor, which determines the importance of future rewards. The policy is then updated so that the agent performs optimally by selecting actions greedily (choose the action with the maximum value). The other possibility is to use Q functions, which are defined as the expected cumulative reward that an agent can achieve by starting from a certain state-action pair. In simple terms, it is the total expected reward if we start in a specific state, perform a specific action and follow our policy then-after:

$$q_\pi(s, a) = E_\pi[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t = s, A_t = a] \tag{2.2}$$

where the only difference is the addition of the action in the equation.
The latter approach updates the policy by directly optimizing the parameters of the policy network to maximize the expected cumulative reward from the environment which is typically done using gradient-based optimization algorithms where the reward function is defined as follows:

$$J(\theta) = \sum_{s \in S} d^\pi(s) V^\pi(s) = \sum_{s \in S} d^\pi(s) \sum_{a \in A} \pi_\theta(a \mid s) Q^\pi(s, a) \tag{2.3}$$

where $d^\pi(s)$ is the stationary distribution of the Markov chain for policy π_θ, which stands for the notion that the system will ultimately settle into a regular pattern of being in each condition over time, regardless of where you start. The gradient of equation (2.3) is computed in order to update the policy:

$$\bigtriangledown_\theta J(\theta) = E_\pi[Q^\pi(s, a) \bigtriangledown_\theta \ln \pi_\theta(a \mid s)] \tag{2.4}$$

Furthermore, we will be focusing primarily on a third type of method which aims to create a bridge

between value-based methods and policy-gradient methods, Konda and Tsitsiklis (1999). Actor-critic methods, which have their roots in RL, combine the best features of both techniques: the 'critic' estimates the value function to evaluate the actions selected by the 'actor,' which is the component that makes decisions (selects actions) based on a policy. When utilising both approaches simultaneously, learning can be accomplished more effectively and steadily than when employing just one. Actor-critic approaches benefit from both the value-based approach, in which the optimisation process is guided by a value function (the critic's assessment), and the policy-gradient approach, in which the policy (the actor's behaviour) is directly optimised. By lowering the variance of the policy gradient, the critic's evaluation contributes to longer-term learning that is more stable. Actor-critic approaches can successfully balance exploration and exploitation, two essential elements in navigating complex environments, by combining these two features.

RL has advanced significantly thanks to these techniques, which enable agents to learn the best courses of action in high-dimensional state and action spaces. Their efficiency and adaptability have produced important advances in a variety of fields, including robotics and gaming.

2.2 Deep Reinforcement Learning and Video Games

Deep learning (DL) enables the creation of computational models made up of multiple layers that can learn progressively abstract representations of data. This breakthrough has significantly advanced various fields, including speech recognition, visual object recognition, object detection, and even domains like drug discovery and genomics, where it has pushed the boundaries of what was previously possible, LeCun, Bengio, and Hinton (2015). Deep Reinforcement Learning (DRL) is the fusion of reinforcement learning and deep learning, offering a powerful approach to tackle intricate decision-making challenges that were once beyond the capabilities of machines. As a result, this field has unlocked a myriad of new possibilities in various domains, including healthcare, robotics, smart grids, finance, and more according to Francois-Lavet, Henderson, Islam, Bellemare, and Pineau (2018). Furthermore, DRL holds great potential for advancing medicine, drug development, and scientific research, as suggested in Arulkumaran, Deisenroth, Brundage, and Bharath (2017).

Regardless of the many possibilities and benefits given to us by DRL we need to find a way to test our methods in a manner that does not endanger living beings. The concept of video games comes to mind when we think of such an experimental environment. The advantage of using video game environments is that they are easily modifiable and offer various levels of complexity in their internal dynamics. The use of DRL in video games is not something novel and researchers have exploited different environments to experiment and test numerous DRL methods. Many video games create complex and engaging challenges for AI agents to solve, making them perfect for AI research. These virtual environments are safe and easy to control for experiments. Plus, they provide endless useful data for machine learning methods, and can run much faster than real-life situations, Shao, Tang, Zhu, Li, and Zhao (2019).

A key highlight is AI's success in mastering 49 different classic 2D Atari games, achieving human-level performance as shown in Mnih, Kavukcuoglu, Silver, Rusu, Veness, Bellemare, Graves, Riedmiller, Fidjeland, Ostrovski, Petersen, Beattie, Sadik, Antonoglou, King, Kumaran, Wierstra, Legg, and Hassabis (2015). This research was groundbreaking, demonstrating DRL's capability to tackle a wide array of gaming challenges, each with its unique set of rules and dynamics.

Then, there's the significant progress in more complex and dynamic gaming environments like Minecraft, referenced in Tessler, Givony, Zahavy, Mankowitz, and Mannor (2016) and the work of Jin, Keutzer, and Levine (2018). These advancements underscore AI's growing proficiency in navigating and adapting to open-world games, which are far more unpredictable and varied compared to more linear games. This leap into intricate gaming worlds is particularly notable for showing how AI can deal with less structured, real-world-like scenarios.

Another intriguing study was the application of a Q-learning model in Super Mario, as cited in Liao and Yi (2012). This research is fascinating for its demonstration of how quickly AI can learn optimal strategies in a classic platform game, achieving impressive success rates. The efficiency and adaptability shown by DRL in this context not only illustrate its potential in gaming but also hint at broader applications in various complex tasks.

Therefore, by combining DRL and video games we can achieve behaviour that mimics or surpasses human-level performance in a multitude of environments with various dynamics and complexity. The next step is to ensure that the algorithms tested take the necessary precautions when faced with risky or dangerous situations to further prevent any complications and problems when applying these algorithms to the real world.

2.3 Safe Reinforcement Learning

Safe Reinforcement Learning (SRL) can be defined as the process of learning policies that maximize the expectation of the return in problems in which it is important to ensure reasonable system performance and/or respect safety constraints during the learning and/or deployment processes according to Garcia and Fernandez (2015). Therefore, we can combine all of the aforementioned concepts into one: using reinforcement learning in tandem with deep neural networks and incorporating safety in order to maximize performance while at the same time accounting for risky situations encountered in video game environments. This combination allows for the testing of various DRL algorithms and potentially finding ones that can be implemented in real world applications with a lower risk of making the wrong decision in a situation where risk is involved.

To capture the environment more fully we implemented an ensemble of critics. An ensemble, in this context, refers to a group of models (critics) that work together, each providing its own estimation of the value function. By combining these multiple estimations, we aim to capture a more accurate and comprehensive representation of the environment, leading to a single, better approximation. This ensemble approach helps in reducing the variance and improving the reliability of our predictions.

Furthermore, we plan to incorporate a risk signal that detects enemies in the vicinity of the environment. This risk signal, coupled with a distance metric, will provide additional, crucial information to the model. With this data, the model can learn to successfully navigate and avoid enemies, enhancing its ability to make safe decisions.

3 Methods

In this section we present the methodology. We introduce the environment we used as well as the agent itself and the learning process that the agent utilized to gain knowledge. The full source code can be obtained here: *

3.1 Super Mario Bros

Often cited as one of the greatest games of all time and admired for its accurate controls Super Mario Bros is a 1985 platform game developed and published by Nintendo for the Nintendo Entertainment System (NES).

The objective of the game is simple. The idea is to make it from the starting point to the end which is to the far right of the world. On the way to the end Mario can pick up super powers, such as mushrooms which make Mario bigger so he can jump over taller obstacles. He can also jump on platforms and over cliffs. Throughout the world there are also enemies that Mario can get rid off by jumping directly on top of them or can just bypass them by jumping over them as shown in Fig 3.1.

Figure 3.1: A screenshot of the game showing Mario jumping over/onto a Goomba (a type of enemy that Mario should avoid) as well as the boxes marked with question marks that give Mario super powers if he hits them either from the top or from the bottom.

The reason we chose this video game for this project is because of the simplicity and linearity that the game offers. The action space, when constrained, offers Mario only 7 actions to choose from which is not at all complicated when compared to other games where there are many more actions possible in a single situation as it is in games that operate in 3-dimensional instead of 2-dimensional environments. Furthermore, it is easy to not get lost when playing the game since the goal is always positioned somewhere to the far right which eliminates the necessity for Mario to backtrack in case he gets lost somewhere in the world. Super Mario Bros offers a simple environment that offers us the possibility to explore and exploit reinforcement learning techniques and test their performance before applying them to real world applications.

3.2 Environment

Prior to building the actual model we first needed to set up the necessary environment and ensure that it is set correctly and that we can obtain valuable information, such as the immediate reward r, the current state s and the subsequent state s', the overall score that Mario has achieved, as well as information related to Mario's internal state (dead or alive) from the environment via interaction with it. To do this we used gym-super-mario-bros (Kauten (2018)) which is a an OpenAI Gym (OpenAI, 2016) environment for Super Mario Bros. and

*Source code adapted from: `https://github.com/vokifrenik/Bachelor-Thesis.git`

Super Mario Bros. 2 (Lost Levels) on The Nintendo Entertainment System (NES) using the nes-py emulator.
We utilized a SuperMarioBros_v0 environment which is the first world in the game using the gym-super-mario-bros package. We then made some minor modifications to the mechanics of the game specifically the movements that Mario can exhibit. The mechanics of the game are straightforward, the complete set of actions consists of twelve moves, however, for our purposes we have limited Mario's movements to just seven distinct actions, seen in Table 3.1, by wrapping SIMPLEMOVEMENT which is a sub-list of actions around the action space in order to simplify the action selection process using the JoypadSpace functionality provided to us by the nes-py emulator:

Code Representation	Description
('NOOP')	Do nothing
('right')	Walk right
('right', 'A')	Jump right
('right', 'B')	Sprint to the right
('right', 'A', 'B')	Sprint-jump right
('A')	Jump
('left')	Go left

Table 3.1: This table shows the action space after wrapping SIMPLEMOVEMENT around the action space. The left column is the action representation in the code and the right column describes what that action does.

After setting up the environment we had to preprocess it so that it can be used in our neural network. In our case we are feeding the neural network our current state at each iteration of the training process. For our purposes we have defined a state to be represented as a stack of frames that represents what is currently happening in the environment. We need multiple frames in order to determine the direction of motion of the characters within the environment and a single screenshot does not provide this information. Furthermore, to make the processing of the states in the network easier and less computationally expensive we used gray-scaling to reduce the number of colour channels that we are dealing with. To achieve this we used the GreyScaleObservation function provided to us by the Gym package so that the screenshots only contained shades of gray. Then we created a stack of 4 frames using another one of their functions, namely FrameStack shown in Fig 3.2:

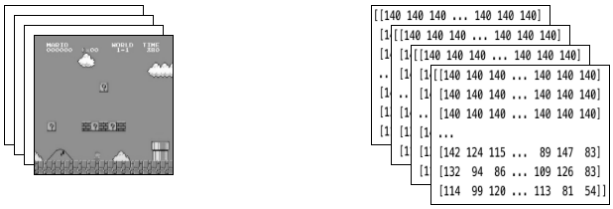

Figure 3.2: The figure on the left shows the stacking of the individual frames of the environment and the figure on the right shows the numerical values of the frames that are used by the network. Each stack of frames had the following dimensions: [4, 240, 256] corresponding to 4 frames of size 240 by 256 pixels.

3.3 Agent

The agent in this case is Mario who is acting upon the environment in order to maximize the reward he receives. The agent's internal workings are comprised of two networks that share most of their architecture with some discrepancies when it comes to the final output layers. We are using an actor-critic approach, and therefore, require two networks: one for the actor and one for the critic. Both networks receive a state as input which has the dimensions [4, 240, 256] which means that there are 4 lazy frames that have the size 240x256 pixels, but as an output the actor returns the mean and variance for the action probability distribution, while the critic returns the expected value of the state in question. Therefore, the actor network has an output dimension of 2, whereas the critic network has an output dimension of 1. In our case we are using one actor model and an ensemble consisting of three critic models which allow us to better estimate the predicted state values by the critics. Furthermore, this configuration introduces benefits when it comes to dealing with uncertainty in the environment and in the model itself.
Since we are dealing with images as input the networks are comprised of a sequence of convolutional layers and pooling layers connected through a ReLU activation function to introduce non-linearity. The overall architecture of the neural network can be seen in Fig. 3.3.

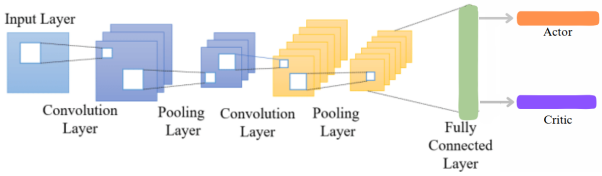

Figure 3.3: A diagram showing the architecture of the neural network.

We use the convolutional layers since they are designed to handle grid-like data in a way that preserves translational invariance and captures local

dependencies. The use of pooling layers helps us reduce the amount of relevant features by either picking the maximum feature value out of a neighbourhood of feature values or by computing the average of the feature values to reduce the computational complexity and time spent processing the information from the states. Specifically, the actor and critic network both share two convolutional layers and two pooling layers as well as two ReLU activation functions. Followed is a shared fully connected dense layer that flattens the input passed on by the convolutional and pooling layers, so it can be used during learning. Finally, our network architecture contains two output heads, one actor head for the mean and variance that are used to define the action probability distribution, and a critic head that returns the estimated value function. Both these heads are comprised of fully-connected dense layers.
Furthermore, the agent takes various parameters during initialization that affect the network architecture and the performance of the learning process. The first two parameters are the α and β learning rates. The former is used in the actor network and the latter is used in the critic network. The learning rate defines the step size at each iteration while moving toward a minimum of a loss function during the training of a model. It influences how much we are adjusting the weights of our network with respect to the loss gradient. We have kept constrained both of these parameters to small values since it can lead to a more stable convergence even if it requires more epochs to train.
The next parameter we had was the discount factor, γ, that balances between immediate and future rewards. We have set this parameter's value to slightly below 1 in order to raise the importance of future distant rewards compared to immediate rewards.
The final two parameters that we have defined are simply the number of neurons that the layers should have which defines the architecture of our networks.

3.4 Learning Process

The learning process consists of a for loop in which the agent chooses what action to take in the given situation by calling the *choose_action()* function. This function retrieves the mean and variance of the action probability distribution from the forward pass of the actor network. It uses this mean and variance to construct a probability distribution and samples an action from this distribution that has the greatest value. This sample is used for calculating the log probabilities of the actions, which are used in the policy gradient update to guide the learning process of the actor network. Actions that result in higher rewards are encouraged by adjusting the policy parameters, while actions leading to lower rewards are discouraged. In the context of the learned policy, the log probabilities offer a means to quantify the likelihood of the selected actions.
To ensure that the agent is exploring new possible actions while also exploiting the learned knowledge we have implemented a Boltzmann exploration strategy. It is generally used for discrete action spaces and assumes that each action in the action space has a Q-value associated to it. It uses a *softmax* function to convert these values into a probability over the actions and then samples an action from that distribution. This strategy utilizes a temperature parameter τ that controls the level of exploration. The following equation describes the Boltzmann exploration process:

$$P(a) = \frac{e^{\frac{Q(a)}{\tau}}}{\sum_i e^{\frac{Q(i)}{\tau}}} \tag{3.1}$$

where *P(a)* is the probability of selecting action a, *Q(a)* is the estimated value of action a and τ is the temperature parameter. Higher values for τ enable a wider range of actions to be taken into consideration because the *softmax* function assigns more comparable probabilities to various actions. Due to the fact that actions with lower Q-values still have a chance of being chosen, this leads to a more exploratory behaviour. On the other hand, decreasing the temperature promotes exploitation, sharpening the distribution and supporting actions with higher Q-values. In our implementation the Q-values are represented by the mean, μ, of the action distribution.
After the action has been chosen, we take a step in the environment using this action to obtain the next state, and an indicator telling us if the agent has died or not. This information is sent through to the *learn* function that the agent uses to improve its performance. In the learn function the agent retrieves the estimated values for the current state and the next state given the action that was selected from the forward pass of the critic network. This information is then used to compute the temporal difference δ,

$$\delta = R + \gamma \hat{v}\left(S', w\right) - \hat{v}\left(S, w\right) \tag{3.2}$$

which refers to the advantage which is defined as: how much better this current action is that the usual action taken in this state. It is formally defined as the difference between the Q function and the *value* function where R is the immediate reward received when performing the chosen action, γ is the discount factor, $\hat{v}\left(S', w\right)$ is the value function starting from the successive state and $\hat{v}\left(S, w\right)$ is the value function starting from the current state.

The temporal difference is then used to compute the actor and the critic loss respectively which are then backpropagated to obtain the gradients for the actor and critic network parameters. These gradients are then used to update the weights of the two networks.
The entire agent's learning process can be summarized in the pseudo-code provided in Algorithm 3.1.

Algorithm 3.1 Actor-Critic Algorithm

Initialize policy parameters θ and value function parameters w
Initialize environment state s
while not converged **do**
  Choose action a from policy $\pi(a|s,\theta)$
  Take action a, observe reward r and next state s'
  $\delta \leftarrow r + \gamma V(s', w) - V(s, w)$
  $w \leftarrow w + \alpha_w \delta \nabla_w V(s, w)$
  $\theta \leftarrow \theta + \alpha_\theta \delta \nabla_\theta \ln \pi(a|s,\theta)$
  $s \leftarrow s'$
end while

This algorithm takes two inputs: a policy $\pi(a \mid s;\theta)$ that decides actions and a value function $V(s, w)$ that evaluates states. Both are adjustable and use parameters θ and w. We also set two step sizes, α_θ and α_w.
We start by giving random small values or zeros to the policy parameters θ and state-value weights w. Then, we repeatedly run the algorithm until the policy stops changing much. In each cycle, we do the following: pick an action using the policy $\pi(a \mid s;\theta)$, act in the environment to get a reward r and the next state s'. We use this new info to adjust our calculations (the δ) and update the weights w and θ. Finally, we move to the new state and repeat the process.

3.5 Uncertainty Quantification

In real-life scenarios, the level of uncertainty varies. Some situations are less predictable, requiring more thoughtful decision-making to handle potential surprises. In the context of our agent and model, methods for uncertainty quantification (UQ) are key. They help reduce the impact of uncertainties in optimization and decision-making processes Abdar, Pourpanah, Hussain, Rezazadegan, and Liu (2021). Moreover, uncertainty exists both in the agents playing video games and within the games themselves. To develop effective deep learning agents, we must address and integrate this uncertainty. While there has been considerable progress in understanding and managing uncertainty, the research on uncertainty-aware deep reinforcement learning is not as advanced as in supervised learning. The interactive nature of gaming environments introduces extra sources of uncertainty, although many challenges common to neural networks in supervised learning also apply to reinforcement learning Lockwood and Si (2022).
As noted before, our model features one actor and an ensemble of three critic models. Each critic in the ensemble estimates a state value, and these three estimates are merged to form a more accurate single state value. This averaging method should yield a state value closer to its true value by leveraging the diverse perspectives of the three critics. Each critic has a unique loss, calculated to enable backpropagation through the networks and facilitate weight updates in the models.
The uncertainty in the critic loss is indicative of the variations in state value estimations. To address this, we calculate the variance using the three critics' estimates. This variance is multiplied by a specific weight to determine its impact relative to the original loss. This step ensures that the model accounts for the variability in the critics' assessments, fine-tuning the learning process. The resulting value, after multiplication, is then added to the original critic loss.
Similarly, the uncertainty in the actor loss relates to the fluctuation in log probabilities of the chosen actions. We use the variance of the log probability as an uncertainty measure for the actor model. This variance is multiplied by a weight to scale its significance alongside the standard loss. Adding this product to the original actor loss helps in adjusting the learning process to account for the variability and uncertainty in the actions chosen by the actor model.
We include uncertainty in the loss function for several key reasons. It helps make the training process more stable and robust, acting like a check to prevent the model from overfitting, especially in complex situations. In real-world cases, where data can be noisy or uncertain, it is important to consider this uncertainty. Understanding how uncertain our model's estimates are can make the model easier to interpret and lets us use adaptive learning methods by showing us how confident the model is in its predictions. In reinforcement learning, this uncertainty plays a role in balancing exploration (trying new things) and exploitation (using known strategies), guiding the model to focus on areas where it's less certain. Also, when adjusted correctly, models that are aware of uncertainty can give predictions that closely match the actual uncertainty present in the data.

3.6 Risk Classifier

So far our agent has the capacity to try out different actions, improve on the selection of these actions

in the states given enough training time and has some ability to recognize how certain it is in its estimation of the values of the states. This is a solid start, however, we need to incorporate perception which the agent can use to detect potential danger in its current state. To achieve this awareness we implemented a detector that detects danger in the environment. For instance, in the game there is a creature that goes by the name "Goomba", shwon in Fig.3.4, that instantly kills Mario if he comes in contact with it without jumping on its head. So, if such a Goomba is present in the current state our agent must be able to detect it and act accordingly to the danger at hand.

For this purpose we used the template matching functionality offered to us by Bradski (2000). The premise is simple; we have two images that are converted to numpy arrays. One of the images is a representation of the current state by taking the first frame from the stack of frames that make up the full embodiment of the state. The second image is a reference image that is used to detect the Goomba in the state image that has been reduced in size so as to match the size of the Goomba within the state image (the environment).

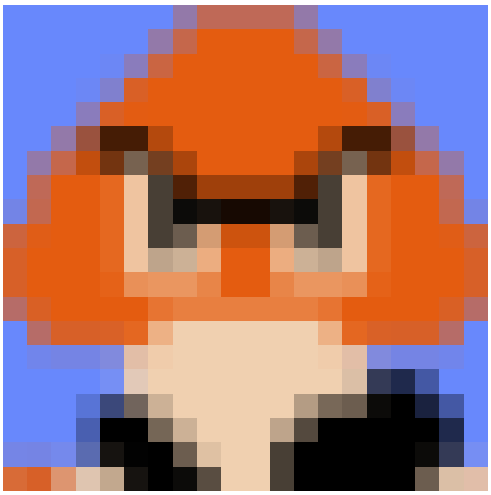

Figure 3.4: This image is the reference image for the Goomba used in the classifier.

The template matching is done by comparing the numerical values that represent the pixel colour in the numpy arrays. We have defined a threshold parameter set to 0.4, so that whenever the similarity metric from the template matching is greater than this threshold parameter we get the coordinates of the Goomba in the state image. If the similarity metric is smaller than the threshold, then the classifier returns "None" signifying the absence of a Goomba in the current state.
In addition, the risk classifier is equipped with a distance function that calculates the distance between our agent and the nearest visible Goomba. The calculation of the distance is only based on the x-axis coordinate:

$$distance = |xagent - xgoomba| \tag{3.3}$$

where *xagent* refers to the x-coordinate of the agent and *xgoomba* refers to the x-coordinate of the seen Goomba. This value is then fed into the shared dense layer of the network so that the actor and the critic can use it as extra information from the environment, and take it into consideration when updating their respective weights.

3.7 Training and Testing the Model

We performed three individual experiments. The first model we trained was the base actor critic model without using an ensemble of critics and with no safety mechanisms implemented. The second model we trained was an actor critic with a critic ensemble but without the safety mechanisms. Finally, we trained an actor critic model with both an ensemble of critics and the risk signal in combination with the distance metric.
We investigated whether or not the addition of an ensemble will improve the performance of the model, and if that will further be improved by the risk signal and distance metric. Specifically, we are going to be comparing the scores per episode obtained by the model and the total number of deaths that occurred during the training process.

4 Results

During the training of the models we recorded the scores per episode and the total number of deaths and saved them to a text file that we then used to plot the results. Due to the computational complexity of the model we decided to end the current episode if the agent's score has not increased in the past one hundred moves. This allows for the model to run through more episodes during the allocated time of the training process. One thing to keep in mind is that if the agent gets stuck somewhere stopping the episodes and moving onto the next one is not counted as a death. A death in our case is Mario getting killed by a Goomba, turtle or falling down a cliff.
After training the base actor critic model and plotting the moving average scores per episode we observed the following pattern:

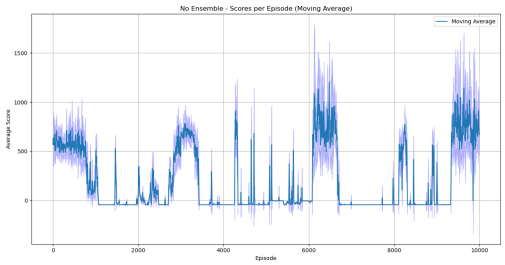

Figure 4.1: Moving average score per episode for the no ensemble actor critic model during training. The light gray areas correspond to the variance of the scores.

From Fig 4.1 we can deduce the behavior of the model. Firstly, we can see that the base model's performance seems to greatly fluctuate across the different episodes. There are periods where Mario's score is quite high and remains that way for a while before reaching these sharp drops in performance where Mario seems to struggle. The performance of the base actor critic model is not as stable and consistent as we would like it to be.
Afterwards, we investigated the results obtained from training the ensemble actor critic model which are presented in Fig 4.2:

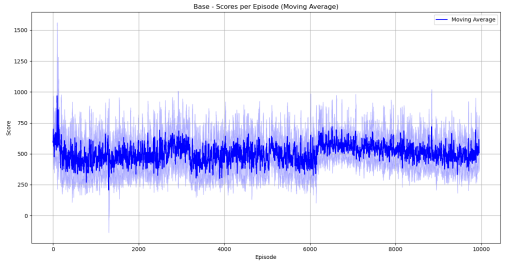

Figure 4.2: Moving average score per episode for the base actor critic model during training. The light gray areas correspond to the variance of the scores.

From the plot presented above we can see that the overall score performance of the ensemble actor critic model seems to be more stable across the episodes. This increased stability in the score can be due to the ensemble of critics used to approximate the value function of the different states. Combining the predictions of the value function from three individual critics seems to provide the model with a more accurate representation of the states and the environment compared to the base actor critic model as we can see in the more consistent score values across the training episodes. However, the performance of this model is still not perfect, there are moments where the score is negative, but for the most part we can deduce that the ensemble model seems to be performing better than the base model.
Finally, we plotted the results from the safe actor critic model which has the ensemble of critics and the risk signal and distance metric functionality. The moving average score obtained per episode is shown in Fig 4.3:

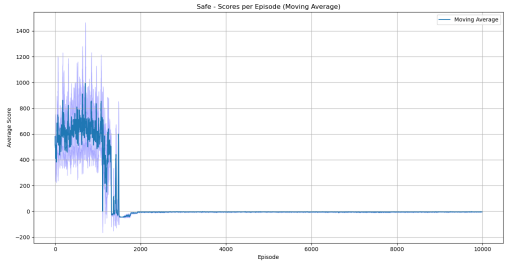

Figure 4.3: Moving average score per episode for the safe actor critic model during training. The light gray areas correspond to the variance of the scores.

As we can see from the graph above, the safe actor critic model has the most interesting pattern of behavior. The score performance of the model starts similarly to the performance of the ensemble actor critic model which is to be expected since the safe model also utilizes an ensemble of critics. The surprising part, however, is that a few hundred episodes before the 2,000th episode the performance drops drastically and remains low for the entirety of the training process. This is especially intriguing when we take a look at the total death per model plot shown in Fig. 4.4:

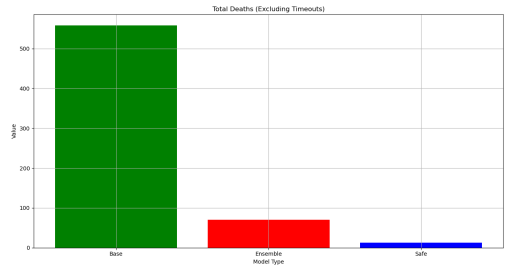

Figure 4.4: Total deaths per model excluding timeouts (restarting the episode).

As can be seen from the above graph the base actor critic model obtained over 500 deaths during training. The ensemble actor critic and the safe actor critic model performed better in this domain with 70 and 13 deaths respectively. The decrease in deaths between the base actor critic and the ensemble actor critic can be because of the ensemble of critics. The ensemble provides the model with a better value function representation of the environment so the model can pick which

actions to take more accordingly. The safe actor critic model had only 13 deaths during the 10,000 episodes of training which suggests that the risk signal and distance metric have some effect on the model's performance. We cannot say if this effect is positive or negative since on the one hand we have a worse score performance of the model but at the same time this model has experienced death less times than both the base actor critic and the ensemble actor critic models.

5 Discussion

As previously mentioned it appears that using an ensemble of critics instead of a single critic to estimate the value function of the states in the environment results in a more stable and consistent score performance. In the Mario environment as in many other virtual and real life environments, there is always some uncertainty about the various states. For instance, when we compare the predictions of the value function for a state given by two different critics we can see that their predictions vary ever so slightly. This variance in the estimations is directly related to the uncertainty that the model has about its environment and by using multiple critics and combining their predictions we can obtain a more realistic and useful representation of the environment. The decrease in deaths can also be associated with this better representation of the environment since Mario can select more appropriate states to move to.
Furthermore, the decrease in performance in the safe actor critic model can be linked to the nature of the distance metric that we are using. Currently it is only a numerical value that is fed into the shared dense layer of the model without any additional information informing the model what this value actually represents in the environment. This lack of information can hinder the model's performance. When a Goomba is present the distance metric is ever changing since Mario and the Goomba are both moving either towards each other or away from one another, so the model can potentially learn what this distance metric is related to. On the other hand, when a Goomba is not present the distance metric is replaced by a huge value that represents that a Goomba is very far away in this specific instant. In this situation the model has nothing in the environment to associate this distance metric with because the value of it will not be changed unless a Goomba is detected in the environment and then this placeholder value is replaced by the actual x-axis distance between Mario and the Goomba. It is possible to provide more context for the risk signal by directly modifying the reward function, however, this is beyond the scope of this project. Another possibility is that the model successfully learns how to avoid the initial Goomba in the environment and after that gets stuck somewhere and this process is repeated over the following episodes. This is plausible because as we already saw the safe actor critic model has the least amount of deaths which leads us to believe that the model, even though under performing in the score department, has some ability to reduce the amount of times that Mario dies even though it may not be consistent throughout the entire environment.

6 Conclusions

In this section we are going to briefly summarize the results we observed and the performance of the three individual models.

- The base actor critic model had the most fluctuating score performance and the highest number of deaths. This was to be expected since it does not use an ensemble of critics to estimate the value function of the states and it does not utilize the risk signal in tandem with the distance metric.
- The ensemble actor critic model had a more stable and consistent score performance throughout the episodes and had a lower number of deaths than the base actor critic model. This increase in performance can be associated with the better representation of the environment due to the ensemble of critics providing a more accurate value function approximation for the different states.
- The safe actor critic model's score performance in the beginning of the training was similar to that of the ensemble actor critic model, however, there was a massive drop in performance at around the 2,000 episode mark, but had the least amount of deaths out of all the models trained. The drop in score performance can be associated to the distance metric not being explicit enough in what it actually stands for and so the model can be getting confused how to use this additional information. Still, the number of deaths suggests that the safety signal in combination with the distance metric have some effect on the model's behavior when it encounters a state involving risk.
- An actor critic model that utilizes an ensemble of critics and a more refined distance metric that offers further information to the model could improve upon the results we have seen in this study.

7 Future Research

Incorporating Safe RL in video game scenarios, such as Super Mario Bros, is an exciting avenue for research. The field is ripe for a myriad of experiments and uncharted explorations. An interesting angle for future studies is to play around with different distance metrics. For instance, shifting from a simple x-axis distance to a more comprehensive Euclidean distance could offer new insights. More crucially, the introduction of a safety signal that encompasses a range of environmental risks – like turtles, cliffs, and groups of Goombas – could be a game-changer. This risk signal, acting as a key motivator in our research, could potentially guide Mario through a safer, yet challenging, journey, offering a richer understanding of the game dynamics under varied risk factors.

8 Acknowledgments

I would like to express my profound gratitude to my supervisor who helped along this journey and offered assistance whenever it was required. This project would not have been finished without their knowledge on the topic and their support.

References

Moloud Abdar, Farhad Pourpanah, Sadiq Hussain, Dana Rezazadegan, and Li Liu. A review of uncertainty quantification in deep learning: Techniques, applications and challenges. *Information Fusion*, 76(Y):243–297, 2021. doi:10.1016/j.inffus.2021.05.008. URL `https://linkinghub.elsevier.com/retrieve/pii/S1566253521001081`.

Kai Arulkumaran, Marc Peter Deisenroth, Miles Brundage, and Anil Anthony Bharath. A brief survey of deep reinforcement learning. *IEEE Signal Process. Mag.*, 34(6):26–38, 2017. doi:10.1109/MSP.2017.2743240. URL `http://arxiv.org/abs/1708.05866`.

Richard Bellman. The theory of dynamic programming. *Bulletin of the American Mathematical Society*, 60(6):503–515, 1954. doi:10.1090/S0002-9904-1954-09848-8. URL `https://www.ams.org/bull/1954-60-06/S0002-9904-1954-09848-8/`.

G. Bradski. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000.

Vincent Francois-Lavet, Peter Henderson, Riashat Islam, Marc G. Bellemare, and Joelle Pineau. An introduction to deep reinforcement learning. *FNT in Machine Learning*, 11(3-4):219–354, 2018. doi:10.1561/2200000071. URL `http://arxiv.org/abs/1811.12560`.

Javier Garcia and Fernando Fernandez. A comprehensive survey on safe reinforcement learning. *Journal of Machine Learning Research*, 16(Y):1437–1480, 2015. doi:DOI. URL `URL`.

Peter Jin, Kurt Keutzer, and Sergey Levine. Regret minimization for partially observable deep reinforcement learning, 2018.

Christian Kauten. Super Mario Bros for OpenAI Gym. GitHub, 2018. URL `https://github.com/Kautenja/gym-super-mario-bros`.

Vijay Konda and John Tsitsiklis. Actor-critic algorithms. In S. Solla, T. Leen, and K. Müller, editors, *Advances in Neural Information Processing Systems*, volume 12. MIT Press, 1999. URL `https://proceedings.neurips.cc/paper_files/paper/1999/file/6449f44a102fde848669bdd9eb6b76fa-Paper.pdf`.

Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015. doi:10.1038/nature14539. URL `https://www.nature.com/articles/nature14539`.

Yizheng Liao and Kun Yi. Cs 229 final report reinforcement learning to play mario. 2012. URL `https://api.semanticscholar.org/CorpusID:16591185`.

Owen Lockwood and Mei Si. A review of uncertainty for deep reinforcement learning. *AIIDE*, 18 (1):155–162, 2022. doi:10.1609/aiide.v18i1.21959. URL `https://ojs.aaai.org/index.php/AIIDE/article/view/21959`.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei Rusu, Joel Veness, Marc Bellemare, Alex Graves, Martin Riedmiller, Andreas Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, February 2015.

Kun Shao, Zhentao Tang, Yuanheng Zhu, Nannan Li, and Dongbin Zhao. A survey of deep reinforcement learning in video games, 2019.

Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. A Bradford Book, Cambridge, MA, USA, 2018. ISBN 0262039249.

Chen Tessler, Shahar Givony, Tom Zahavy, Daniel J. Mankowitz, and Shie Mannor. A deep hierarchical approach to lifelong learning in minecraft, 2016.

Edward L Thorndike. Animal intelligence: An experimental study of the associative processes in animals. *The Psychological Review: Monograph Supplements*, 2(4):i–109, 1898. doi:10.1037/h0092987.

Steven D. Whitehead and Dana H. Ballard. Learning to perceive and act by trial and error. *Mach Learn*, 7(1):45–83, 1991. doi:10.1007/BF00058926. URL `http://link.springer.com/10.1007/BF00058926`.

A Appendix: Parameters and Configurations in environment.py

Environment Setup and Preprocessing

- Game Environment: Super Mario Bros (Version: SuperMarioBros-v0)
- Movement Control: SIMPLE_MOVEMENT
- Observation Preprocessing:
  - Grayscale Observation
  - Frame Stacking with 4 frames

Agent Configuration

- Learning Rate (Actor): 0.0000005
- Learning Rate (Critic): 0.0000001
- Discount Factor (Gamma): 0.95
- Number of Actions (n_actions): 7
- Layer Sizes: layer1_size = 64, layer2_size = 64
- Input Size: Derived from the environment's observation space

Training Process

- Number of Episodes: 15000
- Maximum Steps Without Score Increase: 100
- Model Save Frequency: Every 20 episodes
- Model Checkpoint File: 'checkpoint_base.pth'

Data Recording and Analysis

- Score History: Recorded per episode
- Death History: Recorded per episode
- Aggregated Scores and Deaths: Recorded every 100 episodes
- Output File: 'results.txt'

B Appendix: Parameters and Configurations in actor_critic.py

General Parameters

- Device: CUDA
- Image Path: 'GuillaumeGoomb.PNG'
- Threshold for Template Matching: 0.4

GeneralNetwork Class Parameters

- Learning Rate (lr): Specified at initialization
- Input Dimensions (input_dims): Specified at initialization
- First Layer Dimensions (fc1_dims): Specified at initialization
- Second Layer Dimensions (fc2_dims): Specified at initialization
- Output Dimensions (output_dims): Specified at initialization
- Convolutional Layers: Conv2d with kernel sizes 3 and 5
- Pooling Layers: MaxPool2d with kernel size 2
- Fully Connected Layer: Linear with input size 3477 and output size fc1_dims
- Actor Head: Linear layers for mu and log_sigma
- Critic Head: Linear layer for value function

Agent Class Parameters

- Alpha (learning rate for the actor network): Specified at initialization
- Beta (learning rate for the critic networks): Specified at initialization
- Gamma (discount factor): 0.90
- Number of Actions (n_actions): 7
- Layer Sizes: layer1_size = 64, layer2_size = 64
- Temperature Decay Rate: 0.9999

Model Training Parameters

- Gradient Clipping Norm: 0.5
- Uncertainty Weights: Actor 0.01, Critic 0.01