Sigve Røkenes

Generative Adversarial Reinforcement Learning with Proximal Policy Optimization

Master's thesis in Computer Science, Artificial Intelligence
Supervisor: Björn Gambäck
June 2022

Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Computer Science

Sigve Røkenes

Generative Adversarial Reinforcement Learning with Proximal Policy Optimization

Master's thesis in Computer Science, Artificial Intelligence
Supervisor: Björn Gambäck
June 2022

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Computer Science

Abstract

This thesis explores the topic of generative adversarial reinforcement learning. Specifically, the thesis proposes a new method that uses Proximal Policy Optimization and adversarial reward systems to train agents to paint using sequential strokes on a simulated canvas.

Generative adversarial networks have, since their introduction, proved to work very well on several challenging problems. Recent research has, for example, facilitated the generation of novel, high-quality images that are difficult to tell apart from real photos. Meanwhile, reinforcement learning has enabled agents to learn how to solve numerous complex problems, most commonly in domains such as video games and physics simulations with continuous control. Recent techniques, such as Proximal Policy Optimization, have proved to work in complex settings previously inaccessible to computers.

The intersection of reinforcement learning and generative adversarial networks is largely unexplored. Given the challenges involved in specifying objective functions in reinforcement learning and recent successes of adversarial learning, combining the techniques is a promising area of research. By introducing adversarial reward systems to the reinforcement learning setting, agents could learn to solve difficult problems using examples of solutions rather than handcrafted evaluation functions.

This thesis demonstrates for the first time that Proximal Policy Optimization can be used in this context, and establishes that both Wasserstein and minimax discriminators are suitable adversaries. Furthermore, the work proves that temporal rewards are crucial to enable learning in complex environments, and demonstrates that the quality of reward signals is greatly influenced by the training procedure of the adversarial opponent. Standard policy architectures are shown to be insufficient given the complexity of the environment dynamics, and the thesis presents a more extensive autoregressive decoder architecture that remains capable of learning and representing good policies. Finally, the thesis proves that population-based training and recurrent policies, previously used in similar techniques, are not necessary to achieve results comparable to the current state-of-the-art.

Sammendrag

Denne masteroppgaven handler om konkurransebasert genererende forsterkningslæring. Mer spesifikt presenterer oppgaven en ny metode som bruker *Proximal Policy Optimization*-algoritmen og en konkurrerende vurderingsmodell til å lære agenter å male bilder i et digitalt tegneprogram.

I løpet av de siste årene har datagenerering ved hjelp av konkurrerende nevrale nettverk oppnådd svært gode resultater på en rekke utfordrende problemer. Denne typen læringssystem trener to konkurrerende modeller samtidig, og lærer over tid å produsere falske eksempler basert på hvilken treningsdata som benyttes. Nyere forskning har utviklet modeller som for eksempel kan produsere originale bilder som er vanskelige å skille fra fotografier. Samtidig har forsterkningslæring blitt brukt til å trene agenter som kan løse mange komplekse problemer, særlig i forbindelse med videospill og fysikkbaserte robotsimuleringer. En rekke problemer som tidligere ble sett på som svært utfordrene å løse med datamaskiner har nå blitt løst, og i mange tilfeller overgår disse systemene mennesker.

Det finnes lite forskning som kombinerer forsterkningslæring og konkurrerende nevrale nettverk, men med tanke på hvor godt metodene har fungert i nyere tid finnes det potensiale i å kombinere dem. Forsterkingslæring er avhengig av en funksjon som vurderer oppførselen til agenter, og slike funksjoner er ofte vanskelige å definere. Konkurransebaserte metoder kan imidlertid lære modeller å estimere en slik vurdering på egen hånd. Et kombinert system kan dermed løse problemer basert på kun løsningseksempler, og vil ikke være avhengig av menneskedefinerte vurderingsfunksjoner på samme måte.

Denne oppgaven demonstrerer for første gang at *Proximal Policy Optimization*-algoritmen kan løse slike problemer, samt at minst to forskjellige læringsmetoder for vurderingsnettverk (*Wasserstein* og *minimax*) fungerer i denne sammenhengen. Videre viser oppgaven at både tidsbasert vurdering og komplekse agentarkitekturer er viktige for å kunne lære fra særlig vanskelige datasett. Til slutt beviser oppgaven at verken populasjonsbasert læring eller modeller med minne over tid er nødvendige for å oppnå resultater som er sammenlignbare med lignende metoder.

Preface

This thesis is the primary work required to obtain a degree of Master of Science in Computer Science at the Norwegian University of Science and Technology (NTNU). The work was conducted at the Department of Computer Science under the supervision of Prof. Björn Gambäck. The project revolved around reinforcement learning in computational creativity, but the specifics of the research were highly open ended. The thesis is preceded by a preparatory specialization project conducted in the fall of 2021. Some parts of the background and related work sections in this thesis are adapted from the preparatory project. This content is marked with the †-symbol.

I would like to thank the authors of all work cited in this thesis, as well as everyone involved in the development of tools and software such as Python, PyTorch, Numpy, Weights and Biases, and OpenGL, which have massively helped in the implementation of the method. In particular, I would like to thank the authors of SPIRAL (Ganin et al.) and SPIRAL++ (Mellor et al.) for the insights provided by their work, and for allowing me to use their figures in this thesis. I would also like to thank my supervisor, Prof. Björn Gambäck, for his encouragement and feedback throughout the process, and for allowing me to freely direct the research towards areas of particular potentional and personal interest. Finally, I would like to thank the NTNU HPC group for providing computational resources for model training, and the Department of Computer Science for the numerous artificial intelligence courses organized over the past two years — these have been invaluable in building a broader understanding and appreciation of the field as a whole.

Sigve Røkenes
Trondheim, 9th June 2022

Contents

List of Figures

List of Tables

1. Introduction

This thesis explores the topic of generative adversarial reinforcement learning. The work proposes an adversarial learning system that uses Proximal Policy Optimization to train agents to paint images on a simulated canvas. This chapter presents the motivation behind the work, the overarching goal and research questions considered, as well as the primary contributions of the thesis.

1.1. Motivation

Reinforcement Learning (RL) is an active area of research which recently has shown great promise on many complex problems such as Go by Silver et al. (2016), difficult video games such as Dota 2 by Berner et al. (2019) and multi-agent cooperation and competition by Baker et al. (2019). The majority of these problems are formulated as games, and were previously unsolvable by other learning algorithms.

Meanwhile, generative methods have exploded in capability since the introduction of Generative Adversarial Networks (GANs) by Goodfellow et al. (2014). Methods such as CycleGANs by Zhu et al. (2017) are capable of translating between domains, and improvements to the original GAN architecture in works such as DCGAN by Radford et al. (2015) and progressive GAN by Karras et al. (2017) have enabled generation of high-resolution photo-realistic images. More recently, the DALL-E 2 algorithm by Ramesh et al. (2022) has combined state-of-the-art language models with generative methods to enable impressive image synthesis from text descriptions.

Although some work, such as SPIRAL by Ganin et al. (2018), has combined reinforcement learning and adversarial methods, the intersection of the two research areas is largely unexplored. Given the results of reinforcement learning techniques and the impressive generative ability of adversarial techniques, there remains potential in employing ideas from both in new creative applications. Unlike the majority of techniques in machine learning, reinforcement learning is not dependent on a fully differentiable setting. This benefit can enable new types of problem-solving for creative processes that are by nature not differentiable, and therefore cannot be solved by current state-of-the-art methods.

Finally, exploration of generative reinforcement learning methods can shed further light on fundamental challenges in reinforcement learning in general. In their paper, Concrete Problems in AI Safety, Amodei et al. (2016) discuss problems relating to objective alignment and unexpected negative consequences resulting from underspecified objective functions in these types of systems. With adversarial reinforcement learning the objective function could be learned by example, and it is therefore a potential venue for increased safety in AI systems as their capabilities and influence continues to grow in the future.

1.2. Thesis Goal

This thesis explores the intersection of reinforcement learning and generative methods in the context of generative adversarial reinforcement learning (GARL). Although some work exists in this area, it does not employ insights and techniques from recent research. The goal of this thesis is to gather knowledge from related work and empirical studies to create a novel and capable method:

Thesis Goal
Create a novel generative adversarial reinforcement learning method capable of consistently producing high fidelity paintings.

The novelty of the method is considered with respect to the existing methods in generative reinforcement learning. By employing ideas and concepts from different research areas, a novel method suitable in this context can be developed. The fidelity of paintings is a measure of the output of the method, and can be compared to both RL-based and traditional generative techniques. Note, however, that the most common goal of GANs is to produce photo-realistic images. Here, the goal is not to produce outputs with the highest level of realism, but rather interesting high fidelity *paintings* learned from photos in real domains. This may be viewed as a type of generation with implicit style transfer, where the style is defined not by the model but by the environment dynamics.

Finally, the method should be consistent in its behavior. This includes the system stability during training as well as its consistency in painting fidelity. Understanding the strengths and weaknesses of novel methods is important, and this topic is therefore throroughly explored throughout this thesis.

Research Questions

The thesis goal represents the overarching purpose of the work. Specifically, this thesis attempts to answer the following research questions which cover important topics relating to the method as a whole:

- **RQ1** How can on-policy reinforcement learning methods be applied effectively in the context of adversarial image generation in non-differentiable environments?
- **RQ2** How can we design deep policy architectures capable of modeling complex mappings of multidimensional state and action spaces suitable for sequential image generation?
- **RQ3** Which discriminator architectures and learning algorithms can enable the implementation of an adversarial reward system that provides consistent and useful reward signals for a reinforcement learning agent in this context?

The first research question revolves around the behavior of on-policy reinforcement learning in the GARL context. More specficially, the previously successful Proximal Policy Optimization (PPO) algorithm is considered as a central component in this type of system. The theoretical background of PPO is presented in Section 2.2.4, whereas its application in this context is explored in Chapters 6 and 7.

The second research question considers the architectural design of the policy responsible for modeling the action space of the environment. As will be demonstrated in Chapter 7, standard architectures are insufficient for high quality behavior in complex adversarial painting environments. Chapter 5 is dedicated to structure of the neural networks used in the method, as well as the motivation guiding their design.

Finally, the third research question revolves around the adversarial component of the system. Similarly to other generative methods, a discriminator model is responsible for providing feedback to the generator component. As this work will show, the structure and learning process of the discriminator greatly affects the behavior of the system. Section 2.4.1 presents the background underlying generative adversarial methods. Chapters 6 and 7 explore the role of discriminators and adversarial rewards in this context.

1.3. Contributions

The contributions of this thesis include a review of background literature and related work, implementation of a new simulation environment, and the development of new neural network architectures and algorithms. The main contributions are:

- *The first demonstration of Proximal Policy Optimization used successfully for adversarial, reinforcement learning based image generation.*
- *An empirical study of various state-of-the-art discriminator techniques in the context of adversarial reward systems.*
- *Implementation and demonstration of a general autoregressive policy architecture capable of modeling complex, high-dimensional action spaces.*
- *The first demonstration of a non-recurrent policy working in a generative adversarial reinforcement learning setting.*
- *A novel temporal reward system improving the quality of adversarial rewards, enabling consistent learning and high-quality results for several different datasets.*
- *Implementation of a highly customizable and algorithm–agnostic painting environment suitable for reinforcement learning.*

1.4. Thesis Structure

The thesis consists of nine chapters exploring key topics motivated by the goal and research questons presented previously:

- Chapter 2 presents the theoretical background on which following chapters are based. The primary topics include reinforcement learning, neural network architectures and generative adversarial methods.
- Chapter 3 presents key work related to the proposed method, inluding other adversarial methods and differentiable techniques with similar applications.
- Chapter 4 presents the simulation environment developed as a part of this thesis. All results and experiments employ agents trained in this custom painting environment.
- Chapter 5 presents the neural network architectures of the policy and discriminator models, as well as the underlying insights motivating their design.
- Chapter 6 presents the system algorithm as a whole, including topics such as the adversarial reward system, the exploration technique and the relationship between the policy and discriminator models.
- Chapter 7 presents experiments conducted to explore the role and effect of different components and variations of the method.
- Chapter 8 presents and discusses the results of the method, including a qualitative analysis of the produced paintings.
- Chapter 9 concludes the thesis in light of the research questions and thesis goal presented previously, and proposes several areas of future work.

2. Background

This chapter presents central concepts and theoretical background underlying the work in this thesis. The chapter is divided into four primary parts, including a brief introduction to computational creativity, an extensive explanation of key concepts and algorithms in reinforcement learning, an introduction to important deep learning architecture components, as well as an introduction to adversarial learning.

2.1. Computational Creativity

Computational creativity is a subfield of computer science that explores how computer systems could behave in a way that humans consider creative. Creative endeavours may belong to many different domains. Art and music are typical examples, but creativity can also manifest in more pragmatic ways, for instance in a child discovering knowledge about the world and applying it to novel situations.

Creativity is a challenging term to define. Newell et al. (1962) consider the usefulness and novelty of ideas the main indication of creativity. Boden (2009) defines creativity as artifacts that are new, surprising, and valuable. Creative endeavours may build on existing ideas or completely break the standard mode of thinking, but some level of *novelty* and *value* is essential.

This thesis explores computational creativity in the context of image generation. Current state-of-the-art methods can produce impressive results, which certainly could be considered novel and valuable by many. Karras et al. (2017) have developed computer systems that generate photo-realistic images of people who do not exist. Ramesh et al. (2022) have developed a system capable of generating novel, high-quality images from simple text descriptions. Many observers would likely consider such works creative if they originated from human artists. Computer systems are often set to a higher standard.

Claims that computers inherently cannot be creative typically originate from the point of the creative process itself, and this view is comparable to the classic Chinese Room thought experiment by Searle (1980). Typical criticism is based on the conception of computing processes as predefined algorithms with a deterministic set of rules. Given the static nature of such systems, no novelty beyond what humans already included may be generated. Modern techniques in machine learning with large datasets from the real world could change this perception. As shown by Ramesh et al. (2022), given sufficient data a learning algorithm can generalize impressively well, and can combine ideas and concepts in novel and unexpected ways. The area of computational creativity will, however, likely remain a topic of philosophical debate for many years to come.

2.2. Reinforcement Learning

Reinforcement learning (RL) is a paradigm in machine learning that is very different from typical supervised and unsupervised learning. In the supervised setting, a model is typically trained to predict classes or values based on a known, stationary dataset. A good model is then capable of generalizing well to unseen data. Unsupervised methods typically attempt to group and categorize unlabeled data. In reinforcement learning there is no dataset, but rather an *agent* interacting with an *environment* (either simulated or real). Samples are collected by exploring the environment directly, and the goal in this setting is to train the agent to achieve a high reward as measured by some reward system.

2.2.1. Terminology†

In order to understand the algorithms and equations presented later in this chapter, it is necessary to define some notation and terminology related to reinforcement learning.

- The environment in which the agent acts is fully described at some timestep t by its state s_t, typically a real-valued multidimensional vector or tensor. The collection of possible states in the environment is known as the *state space*. Each timestep the agent receives an observation o_t, which is a function of s_t — in fully observable environments $s_t = o_t$. This thesis follows the convention of using s_t to indicate the current observation whether the environment is fully observable or not.
- The agent chooses an action a_t each timestep. The *action space* describes the type of applicable actions in the environment. Actions are typically multidimensional vectors, and can be either continuous or discrete. The actions chosen by the agent are given by some policy $\pi(\cdot|s_t)$.
- A trajectory τ is a sequence of state-action transitions in the environment. Each transition is associated with a real scalar reward r_t. $R(\tau)$ is used to denote the cumulative rewards for a given trajectory. Several variants of $R(\tau)$ exist, but following sections consider the most common formulation of the infinite horizon discounted reward $R(\tau) = \sum_{t=0}^{\infty} \gamma^t r_t$, where $\gamma \in (0, 1)$ is the discount factor.

The goal of reinforcement learning is to find a policy π such that $R(\tau)$ is maximized when generating τ using π. Specifically, we wish to maximize the objective function:

$$J(\pi) = \int_{\tau} P(\tau|\pi) R(\tau) = \mathop{\mathbb{E}}_{\tau \sim \pi} [R(\tau)] \tag{2.1}$$

Reinforcement learning as a field is not new, but has been an active area of research for many decades (see Kaelbling et al. 1996). Central concepts originate from several areas, such as control theory, Markov decision processes, dynamic programming and game theory. With the advent of backpropagation and more capable computers, deep reinforcement learning has expanded the scope of these techniques significantly, and enabled their application in more challenging contexts.

2.2.2. Deep Reinforcement Learning

Deep reinforcement learning is a subset of reinforcement learning that combines the theory of classic reinforcement learning with deep neural networks capable of universal function approximation. In this context, the policy π is parameterized by a deep network and optimized using gradient ascent. Many deep reinforcement learning techniques exist, but they can be broadly separated into Q-learning and policy gradient methods.

Q-Learning

In Q-learning we wish to learn the Q function, which is given by the expected future return for the current policy and an initial state-action pair:

$$Q^*(s,\ a) = \mathbb{E}_{\tau\sim\pi}[R(\tau) \mid s_0 = s,\ a_0 = a] \tag{2.2}$$

In environments where the action space is discrete, the optimal Q function is sufficient to find the optimal policy. Given any state s, the optimal action is given by:

$$\pi^*(s) = \underset{a}{\operatorname{argmax}}\, Q^*(s,\ a) \tag{2.3}$$

Which is equivalent to selecting the action with the highest expectation in future reward. In practice, we typically model a function that outputs a vector of expectations for each action. This eases the modeling task as well as action selection. In continuous action spaces there is unfortunately no way to analytically find the argmax from this equation alone. Some methods, such as Deep Deterministic Policy Gradients (DDPG) introduced by Lillicrap et al. (2015), overcome this issue by approximating the maximum using a secondary network while the Q function is learned.

In simple environments we can learn the optimal Q^* function using classic tabular methods. For more complex problems using deep learning, it is usually impossible to learn the true Q^* function in practice. Instead, the function is approximated using environment samples during training through methods such as Deep Q Learning by Huang (2020). This learning procedure typically uses a mean-squared-error metric to optimize the model, and given a sufficient amount of samples the model should converge to a good approximation of the true Q^* function for the environment. In Q-learning, samples may be collected by any policy (not only the Q-optimal policy). These methods are therefore often referred to as *off-policy* methods.

Policy Gradient Methods

Policy gradient methods directly optimize the policy through gradient ascent by estimating the gradient of model parameters with respect to the objective (Equation 2.1). The policy gradient is given by the equation:

$$\nabla_\theta \, J(\pi_\theta) = \mathop{\mathbb{E}}_{\tau \sim \pi_\theta}[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) \; R(\tau)] \tag{2.4}$$

The gradient for a single example is given by the probability of the current action $\log \pi_\theta(a_t \mid s_t)$, multiplied by its expectation in return $R(\tau)$. This translates to increasing the probability of good actions (as measured by R) when applying gradient ascent. Note that the policy gradient uses log probabilities instead of the raw probability. The log maintains the gradient direction while having desirable numerical properties, particularly when working with very small probabilities. See Appendix B.1 for the full derivation of the policy gradient.

The true policy gradient is not known, but can be estimated using samples from the environment. Because the policy gradient depends on π_θ, such samples must be collected using the same policy as we are optimizing in order for the estimate to be valid. These types of methods are therefore often referred to as *on-policy* methods.

After collecting a set of environment samples, the policy is optimized using gradient ascent:

$$\theta \leftarrow \theta + \alpha \nabla_\theta J(\pi_\theta) \tag{2.5}$$

Where α is a small hyperparameter (typically around $1e-4$) determining the learning rate. These equations consitute a direct interpretation of the policy gradient theorem by Sutton et al. (1999).

2.2.3. Advantage Actor-Critics

Unfortunately, the return sample $R(\tau)$ used in Equation 2.4 typically suffers from high variance which may prevent stable learning in more complex environments. The advantage actor-critic framework (A2C) tackles this issue by introducing a value function network and the concept of *action advantages.*

Consider a reinforcement learning agent learning to play chess. For any given state, estimating the win probability for every possible action is an incredibly difficult task due to the vast number of possible game trajectories. The high variance of $R(\tau)$ will be directly reflected in the policy gradient estimate, which is likely to prevent learning unless the number of samples is very large. Collecting a sufficient amount of examples is often computationally prohibitive. Instead of using the return sample directly, actor-critics estimate the *advantage function* $A^\pi(a, \ s)$. The advantage function is an estimate of the relative advantage of performing a specific action a in state s, compared to the expectation following the current policy. In this context, the chess agent need only consider the immediate benefits of a move, rather than the entire future trajectory. The advantage function is given by the equation:

$$A^\pi(s, \ a) = R(\tau \mid s_0 = s, \ a_0 = a) - V^\pi(s) \tag{2.6}$$

Where $V^\pi(s)$ is the value function. The advantage policy gradient is equivalent to

Equation 2.4, substituting $R(\tau)$ with the advantage estimate A^{π}. We can use any baseline in the advantage function without changing the (zero gradient) optimal, as long as the baseline is independent from θ. Actor-critics use the value function baseline, given by:

$$V^{\pi}(s) = \mathop{\mathbb{E}}_{\tau \sim \pi}[R(\tau \mid s_0 = s)] \tag{2.7}$$

The true value function is rarely known, and in deep reinforcement learning the function is estimated using a deep neural network. Learning the value function is a regression problem — typical implementations optimize the value model using the mean-squared-error loss formulation and environment samples.

2.2.4. Proximal Policy Optimization

Proximal Policy Optimization (PPO), as proposed by Schulman et al. (2017), is a policy gradient method intended to improve the performance of basic policy gradients while avoiding the complexity of second order optimization in trust region policy optimization (TRPO, see Schulman et al. 2015a). PPO is motivated by the observation that small changes in parameter space can lead to large changes in the policy. To prevent performance collapse during training, the method uses a surrogate objective to prevent large changes in the policy for individual optimization steps.[†] Proximal Policy Optimization has worked very well on a number of difficult problems, including continuous control, as shown by Schulman et al. (2017), and cooperation in multi-agent settings, as shown by Baker et al. (2019).

Surrogate Objective[†]

The PPO paper presents two variants of the surrogate objective, the clipped objective L^{CLIP} and the KL-divergence penalty objective L^{KLPEN}. This section describes the clipped variant, which was shown by Schulman et al. (2017) to be the best performing type in their experiments.

PPO uses a ratio $\mathrm{pr}_t(\theta)$ to measure the difference between the new and old policy. The probability ratio for a parameterized policy π_θ is given by:

$$\mathrm{pr}_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\mathrm{old}}}(a_t|s_t)} \tag{2.8}$$

Where θ_{old} is a copy of the old policy parameters. Note that these parameters are not changed during a single optimization step. The clipped surrogate objective in PPO is given by the equation:

$$L^{CLIP}(\theta) = \mathbb{E}_t[\,\min(\mathrm{pr}_t(\theta)A_t^{\pi},\ \mathrm{cpr}_t(\theta)A_t^{\pi})] \tag{2.9}$$

Where A_t^{π} is the advantage estimate at time t and $\mathrm{cpr}_t(\theta)$ is the clipped probability ratio:

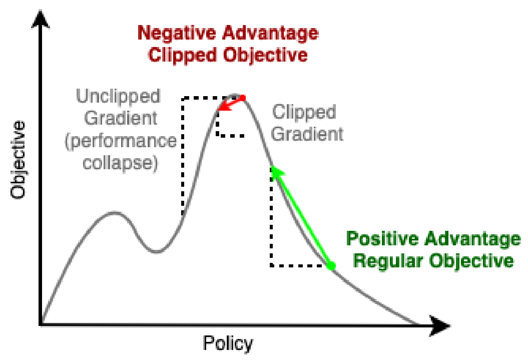


Figure 2.1.: **Clipped surrogate objective in PPO.**†

$$\mathrm{cpr}_t(\theta) = \mathrm{clip}(\mathrm{pr}_t(\theta),\ 1-\epsilon,\ 1+\epsilon) \tag{2.10}$$

Where ϵ is the PPO clipping hyperparameter. The clipping term removes the incentive to adjust the parameters in a way that moves the ratio outside the range $[1-\epsilon,\ 1+\epsilon]$. After applying the min function, we end up with a pessimistic estimate of $\mathrm{pr}_t(\theta)A_t^\pi$ where the ratio is considered only when changing it deteriorates the objective. An illustration of this effect is shown in Figure 2.1.

Sample Efficiency

The surrogate objective enables PPO to be more sample efficient than basic policy gradient methods. Recall that trajectory samples must be collected by the policy that is optimized in order for the policy gradient estimate to be valid. Due to this limitation, samples in these methods are usually discarded after a single gradient step. PPO enables the reuse of data through several epochs. Although data will be slightly off-policy after the initial update, Schulman et al. (2017) showed that multi-epoch optimization works well in practice and increases the sample efficiency of the algorithm.

2.3. Deep Network Architectures

2.3.1. Convolutional Networks (CNNs)

A Convolutional Neural Network (CNN) is a type of sparse neural network particularly well suited for computer vision tasks. Since the introduction of kernel training through backpropagation by LeCun et al. (1989), CNNs have been drivers of breakthroughs in several areas, including image classification by Krizhevsky et al. (2012), generative models by Radford et al. (2015), and reinforcement learning by Clark and Storkey (2014).

CNNs are loosely inspired by the visual cortex of animals, and process spatial data using a set of trainable kernels applied across the whole input. Because of their use of shared weights in this manner, CNNs are spatially invariant. In practice, this enables

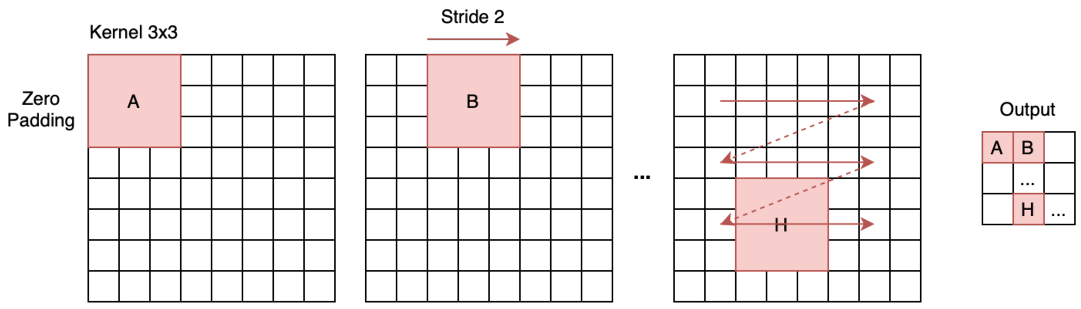


Figure 2.2.: **Illustration of the application of a single CNN kernel**

the networks to recognize features such as edges or shapes independently of where in an image they are located. By connecting several convolutional layers, a deep network can learn to recognize complex concepts such as animals or objects through the hierarchical relationship of simpler features.

Convolutional layers are controlled by a number of parameters defining their behavior. The output channel count determines the number of feature maps a single layer produces. The kernel size controls the size of each learnable kernel, which subsequently affects the complexity of features a single kernel can detect and the receptive field of the model. The kernel stride controls the size of each kernel step when processing the input. Along with input padding, these enable control of the output size of each layer. Figure 2.2 shows an example of how a single kernel is applied across an input image to produce an output.

2.3.2. Residual Networks (ResNet)

Deep residual networks, as presented by He et al. (2016), have since their introduction had broad implications in the field of deep learning. These types of networks employ residual connections, commonly referred to as skip connections, between layers in a model. This type of connection improves the gradient flow in the model during backpropagation, and can increase model performance on a number of different problems. In particular, residual connections have enabled the training of much deeper models which were previously detrimental to performance.

Figure 2.3 shows the basic structure of a deep residual network consisting of several residual blocks. A residual network may also contain longer skip connections across multiple layers, as shown between layers 3 and 5 in the figure. The layers in each block may be any type of network layer as long as the residual and processed output are appropriately shaped to be summed, but residual networks are most commonly seen in CNNs.

In addition to the benefits in terms of gradient flow, residual blocks can more easily adapt to the complexity of input data. In terms of representative capacity, the hypothesis space of a larger model is necessarily a superset of any smaller model it contains. As shown by He et al. (2016), performance may still deteriorate with additional layers, even though the simpler model could be represented by modeling the identity in excessive

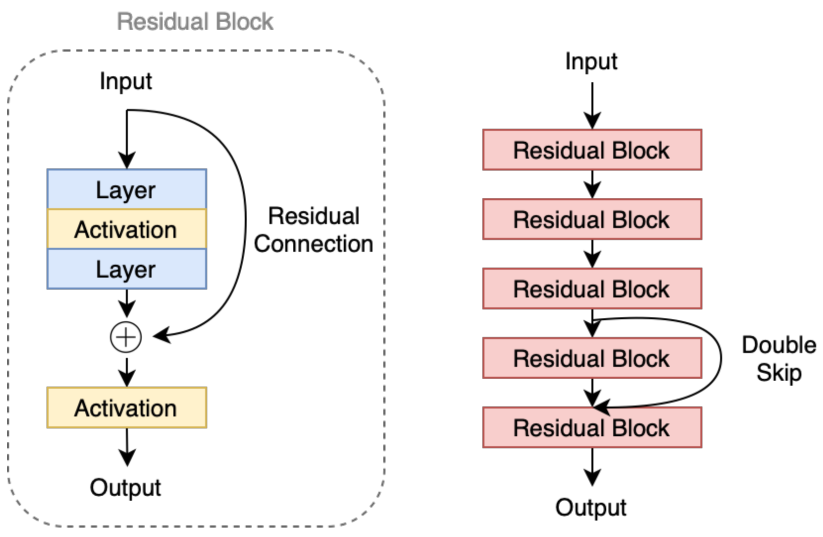


Figure 2.3.: **Structure of a residual block and deep residual neural network.**

layers. This indicates that the identity function is in fact quite difficult to learn. By including a direct skip connection, the identity function is much more easily represented. Intermediate layers need only model the zero function, as the residual connection preserves the original input.

2.4. Adversarial Learning

Advarsarial machine learning is a technique commonly used to train a model to be more resilient against various forms of attacks. In addition to the model trained to solve some problem, the technique introduces an adversarial opponent that attempts to exploit its weaknesses. As shown by Huang et al. (2011), this setup can often improve the learning ability of the system. Following sections present adversarial machine learning in the context of generative adversarial networks, and explain how the method can be combined with techniques in reinforcement learning.†

2.4.1. Generative Adversarial Networks†

Generative adversarial networks (GANs), first conceived by Goodfellow et al. (2014), use an adversarial framework to train a model to generate new examples from some data distribution. GANs are typically used to generate fake images, but can be used in many other domains as well. The method has been shown to be capable of generating impressive results, including photo-realistic images in work by Karras et al. (2017) and Ramesh et al. (2022), among others.

In GANs, two neural network models are trained simultaneously in a zero-sum competitive fashion. In the context of image generation, a generator model is trained to

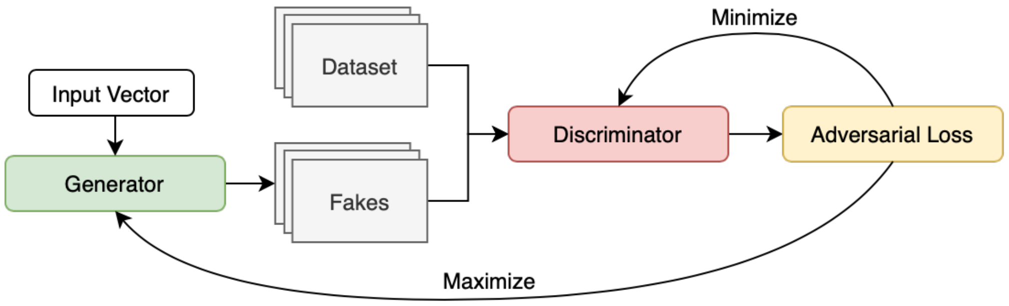


Figure 2.4.: **Structure of Generative Adversarial Networks**†

produce fake image examples. A second discriminator model acts as a binary classifier that attempts to recognize differences between fake and real images. The discriminator prediction serves directly as the learning signal for the generator. In the basic GAN implementation, both models use the same minimax objective function:

$$\mathcal{L}_{GAN} = \mathop{\mathbb{E}}_{x \sim X}[\log D(x)] + \mathop{\mathbb{E}}_{z \sim Z}[\log(1 - D(G(z)))] \tag{2.11}$$

Where D and G are the discriminator and generator models respectively, X is the training dataset and Z is the space of generator inputs, typically a normal distribution. The discriminator is optimized to minimize the function, whereas the generator attempts to maximize it. Other objective variants, such as the Wasserstein formulation by Arjovsky et al. (2017), are considered in later chapters. Figure 2.4 illustrates the typical setup of generative adversarial networks.

2.4.2. Adversarial Reinforcement Learning†

In adversarial reinforcement learning, the opponent attempts to exploit weaknesses in the agent policy. Pinto et al. (2017) have shown that this technique can improve the ability of agents to generalize in difficult continuous control settings.

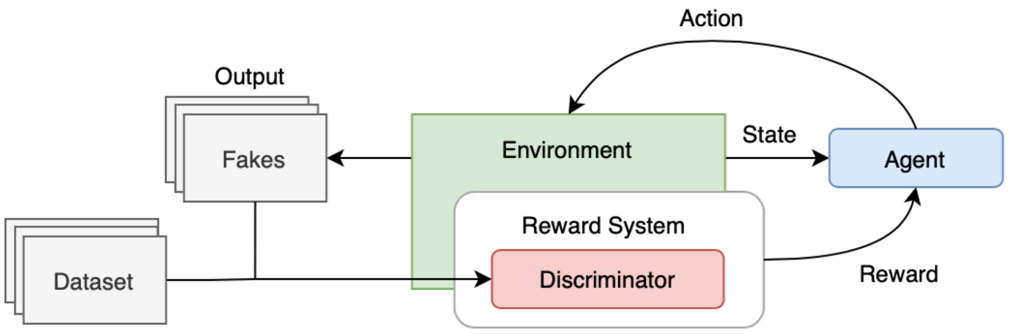
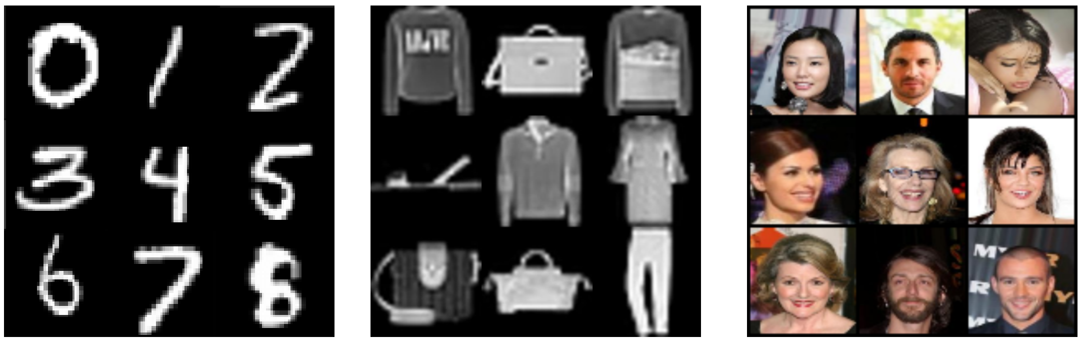


Figure 2.5.: **Generative Adversarial Reinforcement Learning**†



(a) MNIST (b) Fashion-MNIST (c) CelebA

Figure 2.6.: **Dataset examples.** The figure shows examples from the MNIST, Fashion-MNIST and CelebA datasets respectively.

This thesis examines the use of adversarial reinforcement learning in the context of image generation. This application closely mirrors the structure of generative adversarial networks, but substitutes the generator network with a learned policy. Since the output is a result of the policy interacting with a (non-differentiable) environment, the policy cannot be directly optimized based on discriminator outputs like in GANs. Instead, we can use techniques in reinforcement learning to learn from sampled transitions, where the discriminator acts as a reward system. Figure 2.5 illustrates this adversarial setup.

2.5. Tools and Datasets

This work employs a number of open source libraries and freely available pieces of software. The method is implemented in the Python programming language, which is broadly supported in the field. In particular, the implementation relies on PyTorch, provided by Paszke et al. (2019), to define neural network graphs and loss functions, as well as for automatic differentiation and model optimization. Numpy, provided by Harris et al. (2020), is used for various other calculations and data processing. Other libraries used for various auxillary tasks include ImageIO by Klein et al. (2022), torchvision by Marcel and Rodriguez (2010), and Weights and Biases by Biewald (2020).

The results and experiments presented in Chapters 7 and 8 are based on systems trained on three commonly used and freely available datasets. These include MNIST by LeCun and Cortes (2010), Fashion-MNIST by Xiao et al. (2017) and CelebA by Liu et al. (2014). Figure 2.6 shows examples from the datasets used in this thesis.

MNIST is the simplest dataset considered in this work. The dataset consists of grayscale images of handwritten digits at a 28x28 pixel resolution. MNIST is commonly used to show that a technique works on simple tasks, and was chosen for this project to provide an easy baseline for comparison and proof of learning.

Fashion-MNIST is a slightly more complex dataset consisting of grayscale images of various clothing items at a 28x28 pixel resolution. Fashion-MNIST represents a more difficult problem in generative adversarial reinforcement learning, but remains simple enough to enable quick iteration and evaluation.

Finally, CelebA is the most challenging dataset explored in this work. The dataset consists of aligned, colored photos of celebrities, and is a larger and more complex dataset than both MNIST and Fashion-MNIST. Increases in data complexity translate to a more difficult learning problem to produce good results. As a result, CelebA serves better to evaluate the capability of the proposed method, and it is therefore the most thoroughly explored dataset in this thesis. CelebA is also used in similar work, which enables easier comparison with previous methods.

3. Related Work

This chapter presents key work related to the proposed generative adversarial reinforcement learning method. As previously mentioned, this area of research is largely unexplored. Only two previous papers, SPIRAL by Ganin et al. (2018) and SPIRAL++ by Mellor et al. (2019), explore similar adversarial methods. Given their significant relevance, these works are the primary focus of this chapter, though alternatives such as differentiable algorithms are also considered.

3.1. Synthesizing Programs for Images using Reinforced Adversarial Learning (SPIRAL)†

Synthesizing Programs for Images using Reinforced Adversarial Learning (SPIRAL) by Ganin et al. (2018) employs an adversarial reinforcement learning framework to train a policy to generate images through a process of sequential actions. Their method employs an adversarial training setup similar to the example shown in Figure 2.5. A key discovery in the SPIRAL paper is that discriminator predictions can serve as adequate reward signals for stable, unsupervised policy training for this type of problem.

3.1.1. Method

SPIRAL uses a number of distributed actor, policy, and discriminator learners to speed up training. Actors generate policy training data in the form of trajectories in the environment. The policy is trained using an advantage actor critic (A2C) algorithm, using a value function estimator $V^{\pi}(s_t)$ as a baseline (see Section 2.2.3). The discriminator is trained to distinguish between real images from some dataset and the generator outputs, and is used as the environment reward function. The authors found that the minimax objective introduced by Goodfellow et al. (2014) was difficult to optimize in their experiments, and they achieved better results using the Wasserstein loss formulation by Arjovsky et al. (2017).

SPIRAL employs a model architecture similar to that of Deep Convolutional GANs (DCGANs) by Radford et al. (2015) for the discriminator network. The authors use a combination of ResNet feature extractors (as presented in Section 2.3.2), LSTMs (see Hochreiter and Schmidhuber 1997) and an autoregressive decoder to model actions in their policy network.

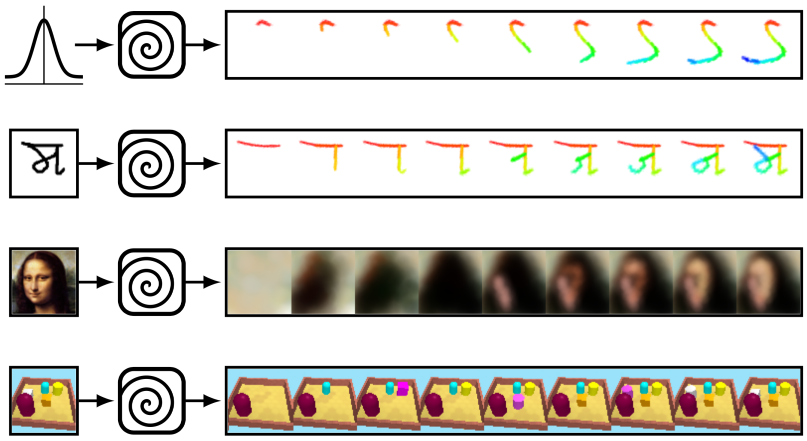


Figure 3.1.: **SPIRAL overview**. The top row shows unconditional rendering with random noise as input. The remaining three rows show conditional rendering on OMNIGLOT, Mona Lisa and a procedural MuJoCo targets respectively. Figure by Ganin et al. (2018), reused with permission.

3.1.2. Results

The authors evaluated their solution on the MNIST, OMNIGLOT (Lake et al. 2015) and CelebA datasets, as well as an original procedural MuJoCo (Todorov et al. 2012) dataset developed for the paper. Their painting environments are based on the open source libmypaint library by MyPaint (2021).

In the libmypaint environment, the authors model actions using 8-dimensional vectors controlling the shape (specified by a quadratic bezier curve), pressure and color of the stroke. In the scene experiment, each action specifies the type of object, as well as its position and color.

After 2×10^8 training steps, Ganin et al. achieved adequate results for both conditional and uncoditional rendering in all environments. The conditional rendering results for the CelebA dataset are shown in Figure 3.2. The results are blurry, but show clear high-level similarity to the target images.

3.2. Unsupervised Doodling and Painting with Improved SPIRAL (SPIRAL++)†

In Unsupervised Doodling and Painting with Improved SPIRAL, Mellor et al. (2019) extend on the SPIRAL algorithm to achieve superior results compared to the original implementation. The method introduces a number of small changes which combined yield the SPIRAL++ algorithm.

Unlike the original algorithm, SPIRAL++ rewards the agent every timestep instead of waiting until the fake image is complete. The reward signal is given by the relative

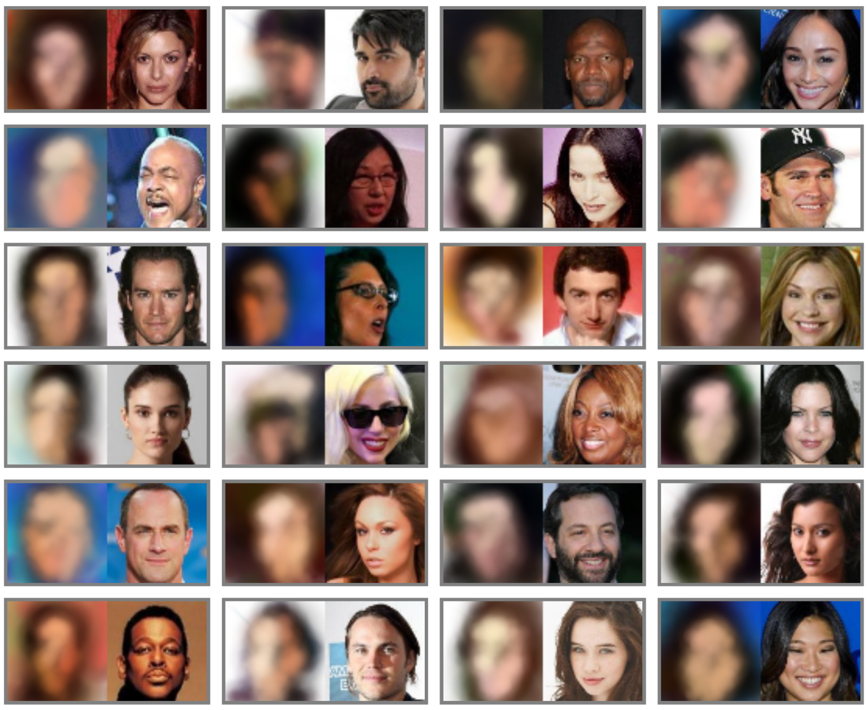


Figure 3.2.: **SPIRAL reconstructions.** Example of reconstructions conditioned on the CelebA dataset using 20 brush strokes. From SPIRAL paper, reused with permission from Ganin et al. (2018).

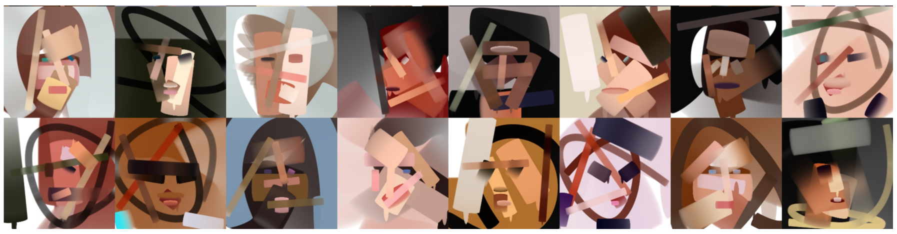


Figure 3.3.: **Unconditional paintings by SPIRAL++.** Examples of unconditional 32-step paintings trained on the CelebA dataset. From SPIRAL++ paper, reused with permission from Mellor et al. (2019).

change in discriminator output between each action. This change reduces the difficulty of the credit assignment problem,[1] but may introduce some bias when the discount factor is not one.

SPIRAL++ also introduces changes to stabilize discriminator training. First, Mellor et al. (2019) introduce spectral normalization in each layer, which serves as a regularizer of the model. The authors found that this empirically improved the quality of produced images. Second, they apply discriminator masking when training on conditional examples. This prevents the model from directly comparing pixel values to determine if a pair is fake, which would prevent useful reward signals for the agent.

Finally, SPIRAL++ trains a population of generator policies sharing the same discriminator. This modification serves to increase the capacity of the generator element and thus ensures that the data distribution is more thoroughly covered by fake examples. Due to the increased rate of data generation, this also enabled the authors to remove the fake image buffer used in SPIRAL. Figure 3.3 shows examples of paintings generated by unconditional SPIRAL++ agents.

3.3. PaintBot

Jia et al. (2019) present PaintBot, a method capable of recreating reference images using a sequence of brush strokes. The technique employs Proximal Policy Optimization to train the policy, but unlike SPIRAL it does not use an adversarial reward system. Instead, the authors propose pixel-wise and perceptual loss metrics to guide policy learning.[†] This type of learning excludes the possibility of generating novel examples, and is therefore more comparable to style transfer than typical generative adversarial methods.

3.3.1. Method[†]

PaintBot uses on-policy Proximal Policy Optimization to train a continuous action space policy. The policy uses a set of convolutional neural networks (CNNs), followed by a fully

[1]Given a long sequence of actions it is difficult to determine which action (or combination of actions) led to a reward at a later point in time. This problem is commonly known as the credit assignment problem.

connected layer parameterizing the action distribution. The authors use a combination of $L^{\frac{1}{2}}$ loss and perceptual loss as the reward signal to encourage image similarity beyond matching average colors. The perceptual loss is defined in terms of the convolutional feature maps:

$$L_{\text{percept}}(I, I_{\text{ref}}) = \sum_{n=1}^{N} \frac{|| \, \phi_n(I) - \phi_n(I_{\text{ref}}) \, ||_2^2}{h_n w_n d_n} \tag{3.1}$$

Where ϕ_n is the n-th feature map from the CNN, I and I_{ref} are the current and reference images, respectively, and $h_n w_n d_n$ is the size of the feature map. This function has the effect of encouraging the policy to match high level features in the image instead of exact colors. Equation by Jia et al. (2019).

The authors use curriculum learning to better handle large state spaces and long action sequences. Curriculum learning entails a gradual increase in the maximum number of simulated steps over the course of training. The number of steps is controlled by a reward threshold parameter. Each episode ends when the reward threshold is reached.

PaintBot uses difficulty based sampling to prevent overfitting on a subset of the data. Each episode, a new reference image I_{ref} is randomly chosen from the dataset, weighted based on the relative policy performance on each image.

3.3.2. Results

The PaintBot algorithm achieves good results on a number of different images with several different brush scales and training datasets. The algorithm converges in about 78k episodes without curriculum learning. With the addition of curriculum learning, the algorithm converges in the same amount of steps, but performs between 20-30% better as measured by the objective function. Refer to Jia et al. (2019) for example outputs generated by the PaintBot method.

3.4. Differentiable Techniques

While generative adversarial reinforcement learning is an uncommon area of research, other authors have explored related techniques for similar applications. Some of these techniques are differentiable, which allows for direct optimization of the objective.

3.4.1. Neural Painter

Nakano (2019) proposes a fully differentiable neural painter that enables intrinsic style transfer[2] of target images. The method relies on a model trained to reproduce images of strokes on a digital canvas. The author explores two variations of the neural rendering model — the first model is trained as a GAN, whereas the second uses variational

[2]In typical style transfer methods, the target style is learned from a reference image by a deep convolutional model. In intrinsic style transfer, the style is a direct result of the appearance of brush strokes in the environment.

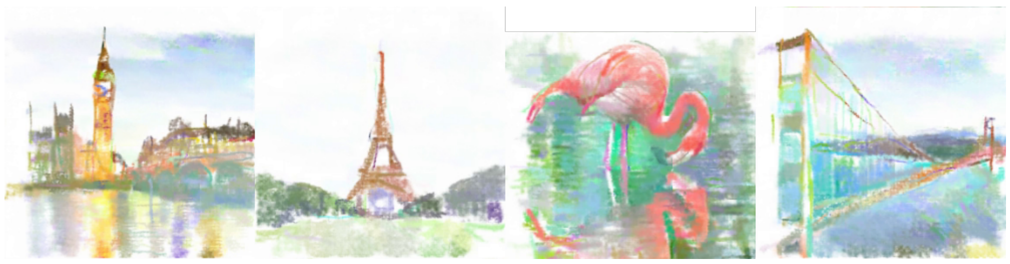


Figure 3.4.: **Paintings by a fully differentiable neural painter.** Directly optimized reconstructions by a differential neural painter. MIT License, courtesy of Nakano (2019).

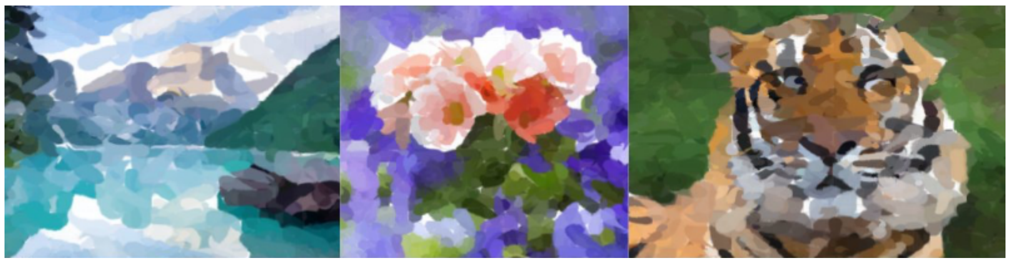


Figure 3.5.: **Paintings by another fully differentiable neural painter.** Directly optimized reconstructions by a differential neural painter. From an earlier personal research project, see Røkenes (2020).

autoencoders (VAEs) to learn stroke reconstruction. Both models are trained using random stroke samples generated by a non-differentiable environment.

The neural painter does not use reinforcement learning. Instead, a set of actions are simultaneously optimized with respect to the objective function to produce an image. Rather than optimizing for the pixel-wise error, the author proposes a content loss objective suitable for intrinsic style transfer, comparable to that of PaintBot. In this context, the loss is given by the mean-squared-error of activations in a pre-trained classifier network for the fake and target images respectively. This enables the model to create a reconstruction where the image content remains similar, without being limited by exact color use or object placement. Figures 3.4 and 3.5 show examples of images generated by differentiable neural painters.

3.4.2. Model-based Reinforcement Learning

Huang et al. (2019) propose a model-based reinforcement learning technique capable of reproducing target images using strokes on a simulated canvas. Their method includes a neural rendering model that learns the dynamics of the painting environment in a supervised fashion using random stroke samples.

The authors use Deep Deterministic Policy Gradients (DDPG) by Lillicrap et al. (2015), previously introduced in Section 2.2.2, to train the policy. Each step, the policy selects

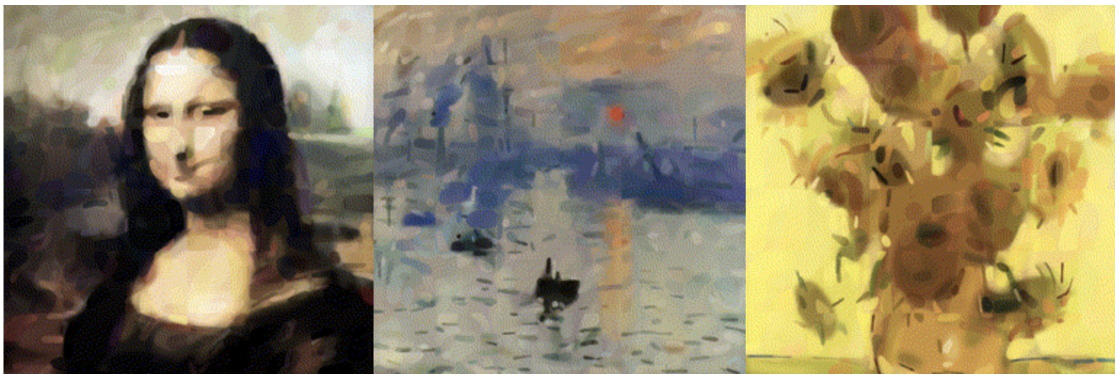


Figure 3.6.: **Reconstructions produced by a model-based DDPG agent.** The agent is capable of painting a wide variety of target images. MIT License, courtesy of Huang et al. (2019).

an action that is used by the neural renderer to estimate the next canvas. The output estimates and target images are evaluated by a WGAN discriminator with gradient-penalty regularization (WGAN-GP) trained to detect real and fake image pairs. The introduction of an environment model ensures that the entire process is fully differentiable. This provides the agent with gradients from the environment model, which enables direct optimization of the policy with respect to the reward. Figure 3.6 shows example outputs produced by this method.

4. Simulation Environment

The work presented in this thesis is based on a custom paint simulation environment that enables the generation of a wide variety of paintings. This chapter presents the implementation details of the system, including configuration options and action spaces relevant to the painting task. In this thesis, the custom environment is referred to as PaintGym.

The environment is implemented in low-level OpenGL[1] to enable the highest performance possible. Since on-policy algorithms do not reuse old data between training epochs, developing a fast environment suitable for rapid sampling of new experience is highly desirable.

To enable easy integration with any learning algorithm, the implementation follows the OpenAI Gym API standard by Brockman et al. (2016). The core of the environment is based on the custom brush implementation. Each brush type is defined by a YAML[2] configuration file that determines its parameterization and appearance.

4.1. Parameterization

The brushes in PaintGym are configured to use one of four different stroke parameterizations — bezier, line, triangle, or point stroke. The stroke parameterization determines how many spatial inputs are required to define the shape of the stroke. For example, a single coordinate is sufficient to draw a point stroke, whereas line strokes require both a start and end position. Bezier strokes require an additional control point to define a cubic bezier curve, and triangle strokes need three points to define the shape of the rendered triangle.

A stroke step setting determines the distance between each rendered texture in a stroke. Bezier strokes are sampled at a high resolution to generate approximately equidistant points — for other parameterizations the exact solution is easily found. Each sampled texture location is associated with approximate angles and percentages indicating their relative position on the curve. These are used for rendering certain types of brushes. Figure 4.1 shows an example of the action space of a line and bezier brush.

In addition to the stroke parameterization configuration, additional initialization settings determine the brushes' expressiveness. These settings include the number of

[1]OpenGL (Open Graphics Library) is a cross-platform API used for rendering computer graphics. The API is widely used for applications such as games, simulations, and other software. Refer to `www.opengl.org` for more information.

[2]YAML is a simple and easily readable markup language commonly used for configuration files. Refer to `www.yaml.org` for more details about YAML.

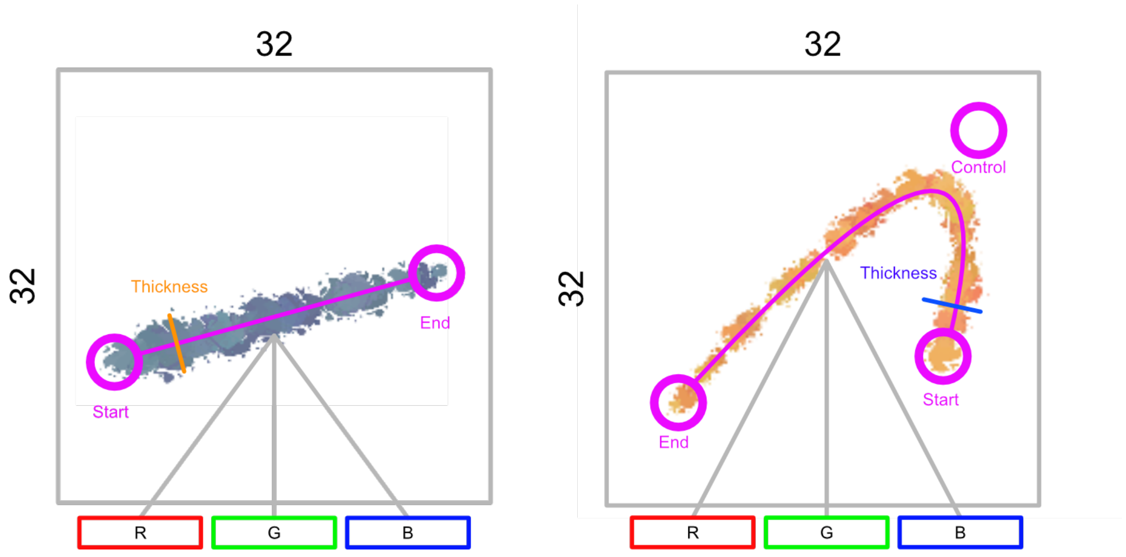


Figure 4.1.: **Illustration of different action spaces in the environment.** The figure shows the action space of line and bezier brush parameterizations, respectively.

color channels (0, 1, or 3, where 0 indicates the color is always white) and whether to control the opacity and thickness/pressure of the stroke. The environment also supports color parameterization in HSV-space (hue, saturation, value), which is applied directly at the shader level when enabled.

The brush configuration also determines the appearance of the rendered stroke, including which texture to render at sampled points, whether the texture should rotate to follow the stroke direction, the minimum and maximum texture size, and random jitter in color and opacity. Note that the inclusion of jitter makes the environment stochastic.

For an example of a complete brush configuration file, see Appendix A.1. In combination, the configuration parameters enable considerable variation in possible brush appearances. Figure 4.2 shows an example of different brushes possible to create in the environment.

4.2. Rendering

When a brush is provided with an action vector, it generates a multi-sprite buffer that contains the necessary information about each texture in the brush stroke. The data is passed to the OpenGL rendering engine, which generates the necessary vertex buffers and batches them into a single draw call. This process is repeated for every parallel environment, producing a grid of strokes in GPU memory. The resulting frame buffer is then rendered on top of the environment canvas in a second draw call.

The separation of the draw calls enables storage and visualization of painting history, and the engine utilizes a decoupled virtual frame buffer process to support headless rendering. Several rendering processes can run in parallel, which ensures high performance

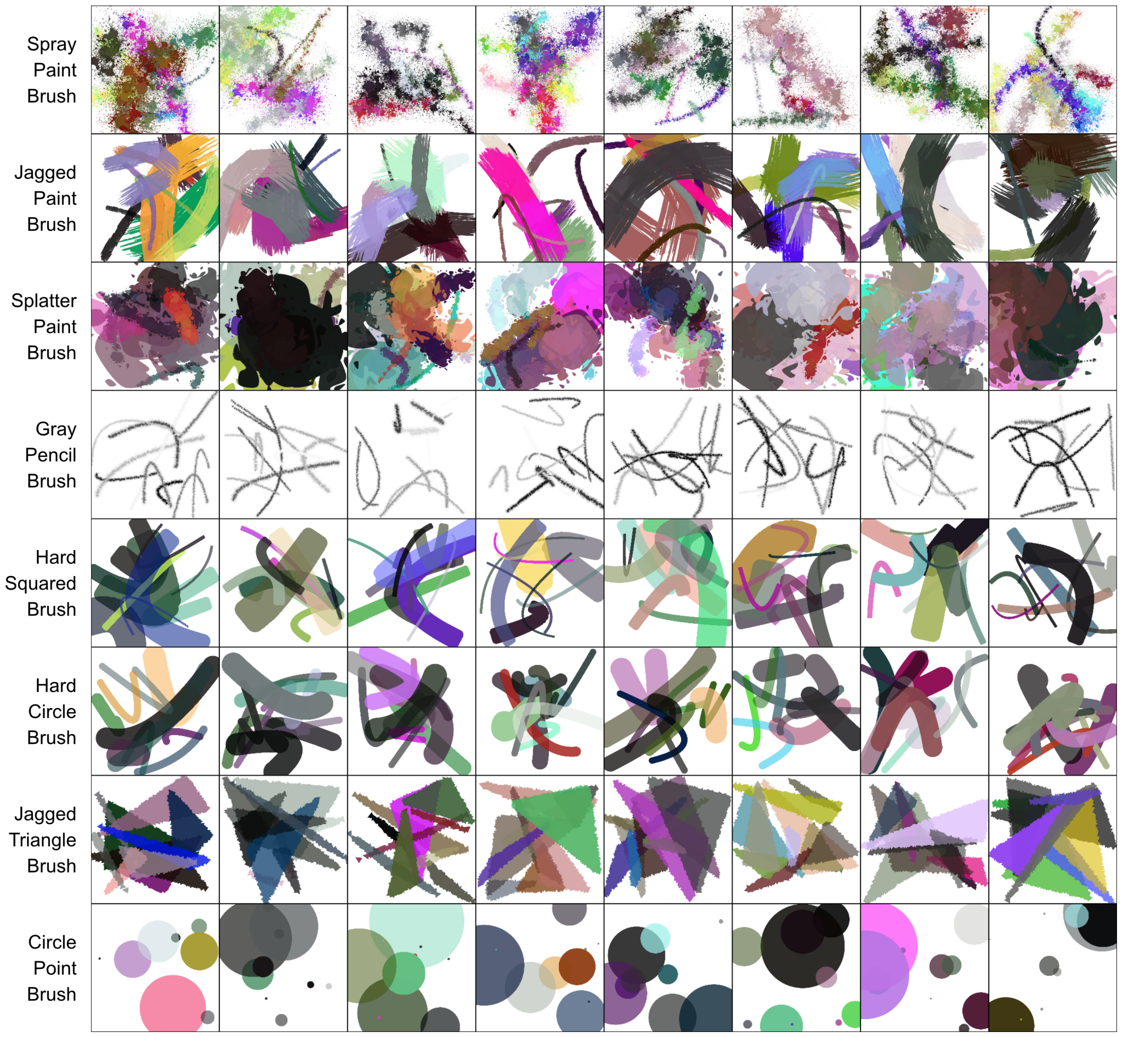


Figure 4.2.: **Examples of various brushes supported by the custom painting environment**. Each image consists of 10 random strokes in the action space.

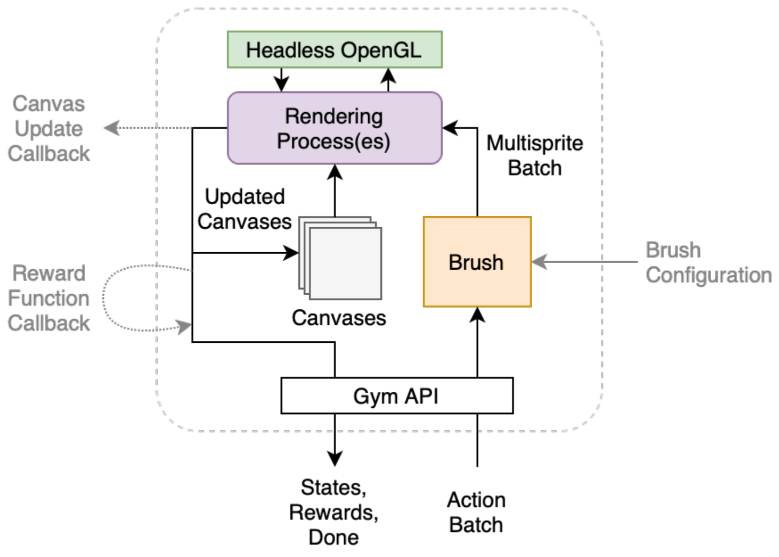


Figure 4.3.: **Overview of the PaintGym architecture.**

on capable computers.

4.3. Interface

With the exception of providing a brush configuration file, learning algorithms need not consider the underlying implementation of the environment. PaintGym follows the standard OpenAI Gym interface, but includes a few extensions necessary for its application.

First, the environment is vectorized and renders multiple images from a batch of actions in parallel. Vectorization improves performance and eases the task of running inference on batches of states when the learning algorithm chooses actions. In practice, the observation and action spaces in the vectorized environment are (n_envs, n_channels, width, height) and (n_envs, <brush dimensions>), respectively. Note that the image size may be different internally in the rendering engine. Images are typically rendered at 2 to 3 times the target resolution, and are resized before being returned to the learning algorithm. Rendering at a higher resolution reduces issues associated with computer graphics, such as aliasing, and increases the quality of resulting image observations.

Second, the environment includes an attachment property that defines callback functions necessary to communicate with the environment. Since the environment defines no reward system on its own, the reward function must be set externally. The reward function also works on batches of images, improving performance when a discriminator model is used as a proxy reward system. Callbacks can also be used to receive finished paintings necessary to train a discriminator. Figure 4.3 shows an overview of the environment architecture.

5. Model Architectures

This thesis proposes several deep network architectures suitable for generative adversarial reinforcement learning. The architectures are based on many previous works, including Deep Convolutional GANs (DCGANs) by Radford et al. (2015), Wasserstein GANs (WGANs) by Arjovsky et al. (2017), WGAN-GP by Gulrajani et al. (2017), SPIRAL by Ganin et al. (2018), and ResNet by He et al. (2016), but combine and extend on these in several ways.

The method employs three primary models — the discriminator, feature extractor, and policy head. The discriminator and feature extractor models build directly on conventional convolutional architectures. The policy head architecture models a partially spatial probability distribution using an autoregressive model similar to that of SPIRAL.

5.1. Discriminator

The discriminator model is a conventional deep convolutional model similar to previous work such as DCGANs by Radford et al. (2015) and WGANs by Arjovsky et al. (2017) and Gulrajani et al. (2017). The model consists of four 2D convolutional feature extractor layers (conv2d) followed by a final linear or convolutional layer producing a single value prediction. Each hidden layer is followed by an optional normalization layer and leaky rectified linear unit (ReLU) activations. The effect of various normalization layers is discussed further in Section 7.4.4.

Following conventions from similar architectures, the discriminator model uses kernel sizes of 3x3. In combination with strides of 2 and (zero) padding of 1, each layer produces feature maps with half the spatial size of its input. The hidden layers employ leaky rectified linear units with a negative slope of 0.2. This type of activation enables gradient

Layer	Shape	Kernel	Stride	Padding	Normalization	Activation
conv2d	$(32, 32, k * 2^1)$	3, 3	2, 2	1, 1	*optional*	leaky ReLU
conv2d	$(16, 16, k * 2^2)$	3, 3	2, 2	1, 1	*optional*	leaky ReLU
conv2d	$(8, 8, k * 2^4)$	3, 3	2, 2	1, 1	*optional*	leaky ReLU
conv2d	$(4, 4, k * 2^8)$	3, 3	2, 2	1, 1	*optional*	leaky ReLU

Table 5.1.: **Discriminator extraction stack**. The extractor stack consists of four 2d convolution layers. The hyperparameter k defines the number of feature maps in the model. Each layer may optionally use a normalization layer such as instance norm, batch norm and/or spectral norm.

Layer	Shape	Kernel	Stride	Padding	Normalization	Activation
conv2d	$(1, 1, 1)$	4, 4	1, 1	0, 0	*none*	*sigmoid/none*
linear	(1)	-	-	-	*none*	*sigmoid/none*

Table 5.2.: **Discriminator output layers**. The discriminator uses either a single convolutional layer or a dense linear layer to produce a single value output prediction.

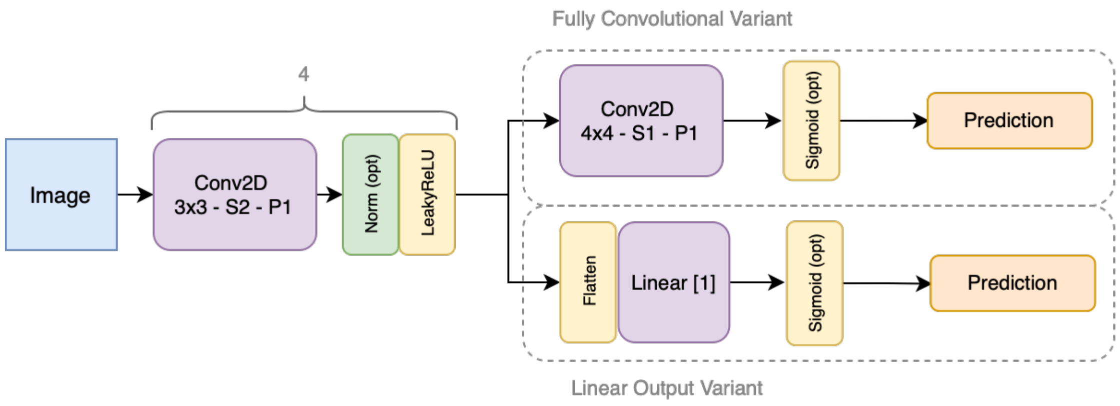


Figure 5.1.: **Discriminator architecture.**

flow for all units, even when their input is below zero. As shown by Radford et al. (2015), the leaky variant is superior to the pure rectified linear unit when used in GAN discriminator models. Tables 5.1 and 5.2 show an overview of the discriminator layers, and Figure 5.1 shows an illustration of the architecture.

All layers except the output layer use trainable biases. These are often omitted in other discriminator architectures because the implicit bias in subsequent batch normalization layers leaves the parameters redundant. As will be discussed in Section 7.4.4, batch normalization is not well suited to this application area and bias is therefore included directly. The output layer is followed by a final sigmoid activation whenever the cross-entropy loss formulation is used. Other training algorithms, such as WGANs by Arjovsky et al. (2017) and Gulrajani et al. (2017), and Least Squares GANs (LSGANs) by Mao et al. (2016), use the linear prediction directly.

5.2. Feature Extraction

The feature extractor used for the policy is a deep convolutional architecture with residual connections. The architecture is built on previous work by Ganin et al. (2018) and He et al. (2016). A hyperparameter c determines the number of filters in the model, and along with the number of residual blocks, it determines the model's capacity. This work uses a channel count of 64 and 8 residual blocks, which provides a balance between

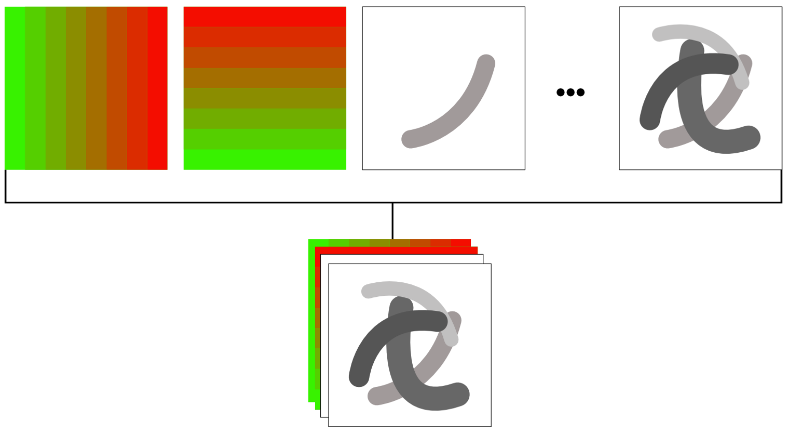


Figure 5.2.: **Feature extractor input.** The input consists of two channels of linear spatial grids, and one or more stacked frames from the environment canvas.

capacity and model size, and worked well in preliminary experiments.

The feature extractor input consists of one or more stacked canvases, enabling the model to observe changes in the canvas over time if enabled. Stacking observations is a common design pattern in image-based reinforcement learning, previously used by Mnih et al. (2013) among others.

Two channels of constant values are concatenated to the image before it is processed. These channels define a spatial grid and contain linear gradients from -1 to 1 on the X and Y-axis, respectively. Additional grid channels were also used by Ganin et al. (2018) and Mellor et al. (2019) in their implementations. Unlike typical classification models, the spatial location of detected features is crucially important for action selection. As discussed in Section 2.3.1, convolutional layers are spatially invariant. The inclusion of a grid enables the model to assess the location of features at any level of abstraction. Figure 5.2 shows the structure of the feature extractor input.

Architecture

The feature extractor network contains an initial 2D convolutional layer with a kernel size of 5, a stride of 1, and zero padding of 2, producing c feature maps with the same spatial size as the input. This enables the embedding of additional information independently to the size of the canvas, as will be discussed in Section 5.5.

The initial convolution is followed by three hidden convolutional layers using rectified linear unit activations. The layers use c channels, kernel sizes of 3, padding of 1, and strides of 2. Each layer halves the size of the feature map, resulting in c 8x8 feature maps after the final layer for, given the canvas size of 64 used in this work.

The hidden layers are followed by a stack of residual blocks. Each residual block consists of two convolutional layers with kernel sizes of 3 and 1 and paddings of 1 and 0, respectively. Both layers use a stride of 1, and are followed by a rectified linear unit activation.

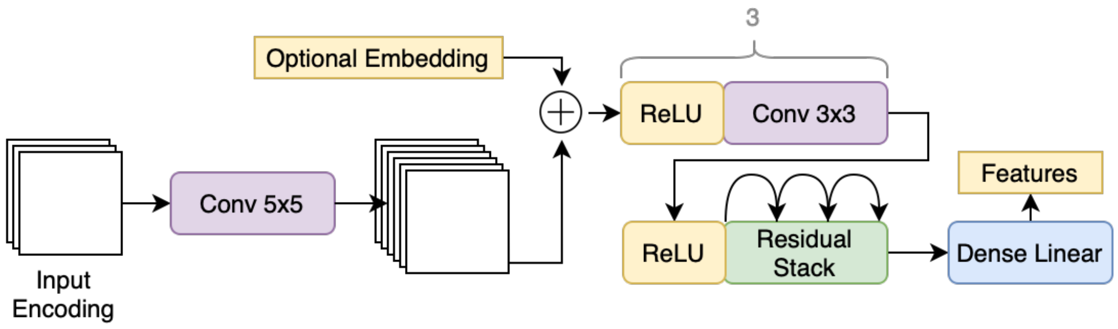


Figure 5.3.: **Feature extractor model.** The feature extractor consists of a number of convolutional layers followed by a residual stack and a dense linear layer. Embeddings are optionally added after the initial 5x5 convolution.

The output of the convolutional layers is added to the input and passed through a rectified linear unit activation to produce the final output. As discussed in Section 2.3.2, these skip connections enable the model to be very capable while maintaining gradients through a large number of layers. The output is finally passed through a dense linear network to produce a feature vector. All experiments in this thesis use a feature vector of size 256, which was shown by Ganin et al. (2018) to work well in practice for similar problems. Figure 5.3 shows an overview of the feature extractor model.

5.3. Critic

The critic model uses the same feature extractor as the policy, and model parameters are shared. This design decision assumes that there exists a common representation suitable for modeling both actions and values. Although this is a strong assumption, it can work well in practice for visual problems, as shown by Schulman et al. (2017) on Atari and Ganin et al. (2018) in SPIRAL.

The critic head is a small neural network with one hidden linear layer with 64 units, and a linear output layer that produces a single value. A rectified linear unit activation separates the two layers. Given the simplicity of the network, the majority of the value function modeling capacity exists in the feature extractor. The critic head combines the previously extracted features to produce a single numerical estimate.

5.4. Policy

The policy head decodes the representation provided by the feature extractor to produce a probability distribution over stroke parameters. The policy is represented by a discrete categorical distribution translated to continuous values before being executed in the environment.

The resolution of each action parameter is configurable, but for a typical bezier brush

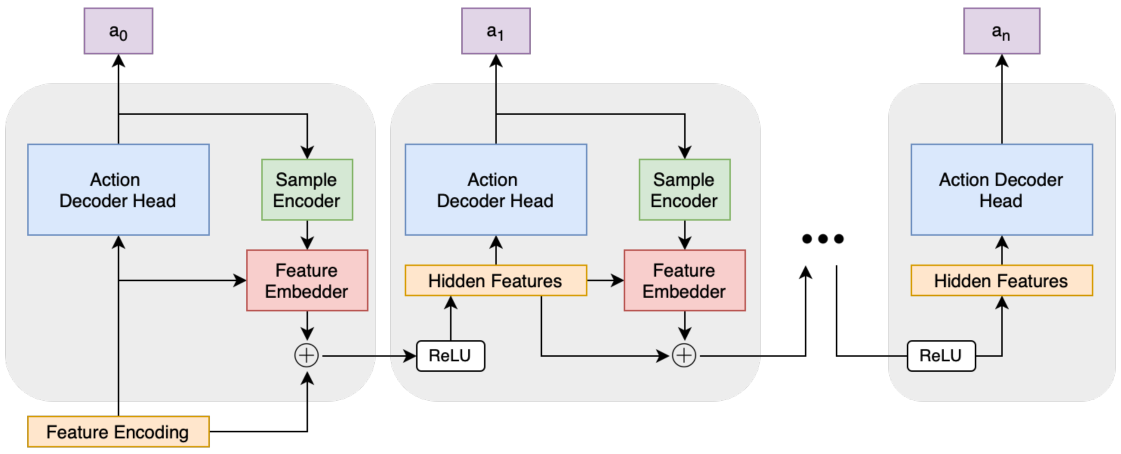


Figure 5.4.: **Overview of the autoregressive policy head**. The policy head consists of a chain of action decoders that individually model the distributions of each action parameter. The first head decodes the feature encoding directly, whereas later heads are conditioned on the sampled value of the preceding heads. This is achieved using sample encoding and feature embedding networks with residual connections.

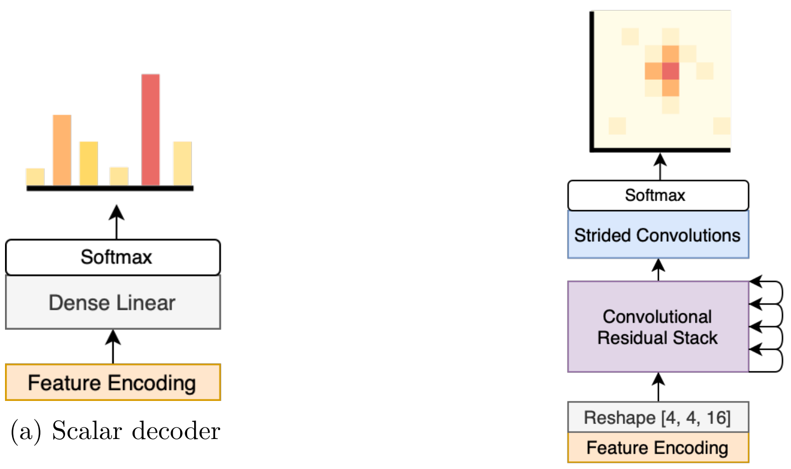
environment, the action space contains 32^2 options for each stroke coordinate (start, end, and control positions), 16 options for each color channel, and 16 options for the thickness and opacity respectively. The total number of possible strokes is very large — for this example, we have $32^6 \times 16^5 \approx 1.2 \times 10^{15}$ (more than 1200 trillion possible strokes). The action space is too large to be practically modeled by a single distribution. To enable learning in such a large action space, the distribution is modeled by an autoregressive multi-head policy. In this setting, the probability of some action a is given by a chained conditional probability of each value in the action vector:

$$P(a \mid s) = P(a_0 \mid s) \times P(a_1 \mid a_0,\ s) \times \cdots \times P(a_n \mid a_0,\ a_1,\ \ldots\ a_{n-1},\ s) \tag{5.1}$$

To model this type of distribution, the policy head contains a chain of action decoder networks connected by a set of sample encoders, feature embedders and residual connections. Figure 5.4 shows an overview of the policy head architecture. The following sections explain each part of the architecture in detail and motivate its various design decisions.

5.4.1. Decoder Heads

The action decoder heads are responsible for decoding input feature vectors to model categorical probability distributions for each action space dimension. Scalar dimensions (such as thickness and opacity) are modeled by a single linear layer with a softmax activation function. Spatial dimensions employ a more complex convolutional network



(a) Scalar decoder

(b) Spatial decoder

Figure 5.5.: **Spatial and scalar decoder heads.** Scalar dimensions are modeled by a single linear layer followed by a softmax activation function. Spatial dimensions are modeled by a more complex decoder containing a residual stack and strided convolutions.

with 8 residual blocks. Figure 5.5 shows the architecture of the scalar and spatial decoder heads respectively.

The design of the spatial heads is motivated by two primary factors. First, the x and y coordinates of a stroke position depend strongly on each other. A stroke starting at some position x may be very good *only when the y coordinate is some specific value.* This relationship may be captured through two conditional decoder heads but is more readily learned as a single distribution. Second, several nearby coordinates are likely comparable in performance for any given input state. This type of local relationship is captured better in a convolutional network with implicit spatial bias. Non-spatial parameters, such as the stroke thickness, are not bound by these considerations and are therefore modeled by a single linear layer.

As discussed in Section 2.3.2, the use of residual connections enables improved gradient flow in very deep models. The residual stack in the spatial decoder ensures that the gradient flows from the output distribution, all the way to the original input in the feature extractor. Strided convolutions are necessary to scale the distribution to the desired spatial resolution.

5.4.2. Sample Encoding

After a head has processed its input feature encoding, the result is a categorical distribution over the action space of a single index in the action vector. Before using the next head, we sample from the distribution to produce a value for the respective action dimension. The following heads are then conditioned on this sample to model the probability in

Equation 5.1.

Each sampled action is first encoded into a hidden representation with a common size. This work uses a hidden size of 16 to balance generality and accuracy for various action dimension resolutions. For scalar dimensions, the hidden representation is created by encoding the sample as a one-hot vector and passing it through a single linear layer. The encoding method is similar to the technique used by Ganin et al. (2018).

Spatial samples use a direct linear transformation of the values. Given the spatial resolution, one-hot encodings over the whole space would be comparatively large ($32^2 = 1024$), which could reduce the conditional generalization ability of the model. Different encodings per dimension (yielding two vectors of size 32) is an option, but this technique could make it more challenging to represent the relationship between the dimensions. Therefore, spatial encoders directly map the raw values using a linear layer ($x, y \to h$).

Note that the linear encoding layers for both scalar and spatial encodings use no activation function — the linear output is instead passed directly to the feature embedder. Since the encoded action size is smaller than the dimension of the sample, this step may become an information bottleneck if a majority of activations are low. The activation function is omitted to reduce potential adverse effects due to poor initial weights, and to provide the model with more direct control of the relationship between action samples and conditional feature encodings.

5.4.3. Feature Embedding

The feature embedder is responsible for providing the next head with an encoding containing the necessary information for the conditional on the state s as well as previous action samples. To achieve this, the hidden sample encoding and input feature encoding are concatenated and passed through a multi-layer perceptron (MLP). The architecture uses rectified linear activations and two hidden layers of size 32 and 64, respectively. The output works as a residual vector which is summed with the original feature vector and passed through a final rectified linear unit to produce the subsequent feature encoding.

The residual design of the feature embedder is motivated by two main factors. First, these connections have desirable properties in terms of gradient flow through the many layers in the policy head. Second, using a residual connection enables each head to more easily suppress or increase activations for features relevant to the previously selected action sample. Consider a toy example where an agent is drawing faces and receives a simple feature vector of size 3, where indices indicate the presence of a person's eyes, nose, and mouth, respectively. Given that the previous head has selected a red color, it could be beneficial to suppress signals relating to the eyes and nose (as these are typically a different color) because they are irrelevant to action selection in later heads. Such suppression can be achieved by producing a residual vector where the first two indices contain large negative values. Good suppression ensures that each action decoder only has to learn the simplest function necessary to achieve good results.

Note that the MLP feature embedder receives both the previous feature encoding and the most recent sample as input, enabling the embedder to produce a residual vector based on any relationship between these vectors. Building on the previous example, this

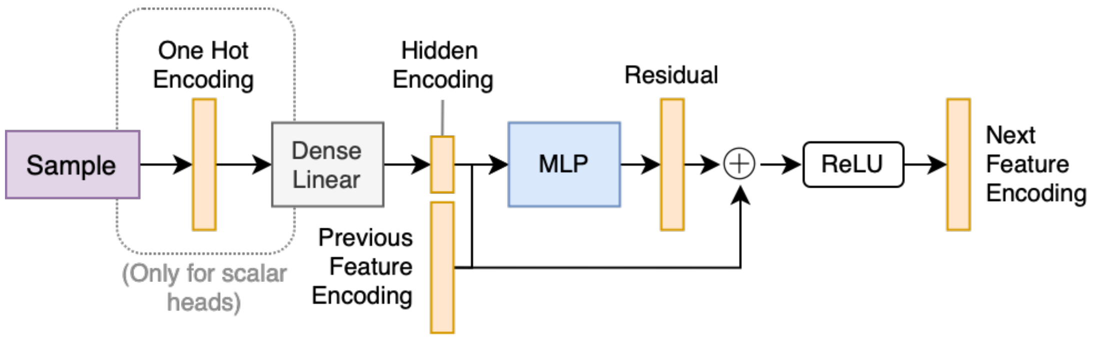


Figure 5.6.: **Sample encoding and feature embedding.** The action sample is transformed into a hidden encoding using a single linear layer. This encoding, as well as the previous feature encoding, is concatenated and passed through a multi-layer perceptron to produce a residual feature vector. The residual and original feature encodings are summed and passed through a rectified linear unit to produce the next hidden feature encoding.

could enable the network to suppress the eye and mouth features for red strokes *only when a mouth is not present*. If a mouth already exists, the red color may be useful for some other feature in the painting, and the information should not be suppressed. Figure 5.6 shows an overview of the sample encoding and feature embedding architecture.

5.5. Embedding Extensions

In addition to the internal sample and feature embeddings, the policy also employs embedding extensions for improved performance. Embedding extensions include spatial grids, previous actions, and observational noise. The purpose of these extensions is to provide the policy with more information beyond the current canvas observation to improve the model's capacity or encourage desirable behavior.

5.5.1. Observational Noise

The inclusion of observational noise is motivated by the design of generator networks in generative adversarial algorithms such as GANs and WGANs. Recall that generators are conditioned on a random latent vector to produce an image, as described in Section 2.4.1. The input randomness enables the network to cover a broad range of the target distribution and is strictly necessary for non-deterministic outputs.'

The process through which images are produced in this adversarial reinforcement learning setting is not deterministic due to the stochastic nature of both the agent and environment. Unfortunately, this type of stochasticity is often insufficient to enable the policy to capture a complex multi-modal dataset adequately.

Consider the first stroke of an agent learning to paint eights and ones from the MNIST dataset. A straight vertical line in the middle of the canvas will likely yield a high reward, as the resulting output is similar to the class of ones present in the dataset. Likewise, a curved line matching part of the digit eight could also yield a high reward. Note that the initial observation is identical in both cases — the empty canvas. Since the algorithm reduces policy entropy, the policy will generally commit to only one of these actions, and by doing so, excluding the possibility of generating images matching the secondary mode.

This problem can be avoided by introducing observational noise. For this example, the inclusion of a single random bit is theoretically sufficient (though not reasonable in practice) to enable coverage of the two primary modes in this example. In practice, the embedding consists of a large, normally distributed vector.

The proposed architecture includes two different ways to embed the noise vector as a policy condition, as shown in Figure 5.7. In both cases, the vector is first processed by a small MLP to produce a vector of appropriate size.

- **Feature extractor embedding.** In this setting, the appropriate vector size is equivalent to the number of convolutional feature maps after the initial convolution in the feature extractor. The vector is added to the initial feature map on a per-channel basis before the activation function is applied.

- **Policy head embedding.** This variant embeds the processed noise vector directly in the policy head before the action heads. The feature vector provided by the extractor is summed with the appropriately sized noise encoding before the activation function is applied.

5.5.2. Action Conditional

The second embedding extension conditions the current action on the previous action performed in the environment, if any. This inclusion is motivated by the relationship between sequential strokes and provides the model with single-step memory of actions in the environment. As with observational noise, the embedding may occur at either the extractor or policy head level.

Note that the action vector is quite complex, as it contains information defining the whole stroke. To enable learning from the vector, each dimension is first encoded to a hidden representation in the same way as action samples in the policy head (see Section 5.4.2). The hidden encodings are then concatenated and further processed in the same way as the observational noise vector.

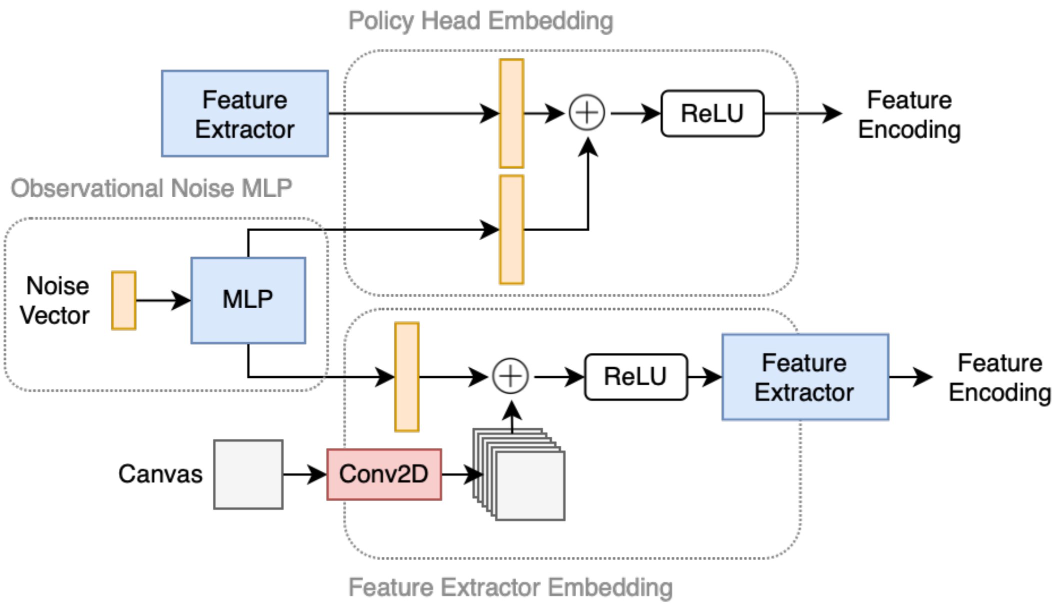


Figure 5.7.: **Observational noise embedding.** The observational noise is first processed by an MLP to produce a vector of appropriate size. The vector may be embedded in the channels of the first feature maps in the extractor (bottom) or directly in the policy head (top).

6. Learning Algorithm

This chapter presents the learning algorithm used to train the neural network models presented in the previous chapter. The primary focus of the following sections is the reinforcement learning system, whereas discriminator training is discussed primarily in the context of the adversarial reward system. The chapter explores how various algorithm choices relate to each other and previous work, and justifies the design decisions with respect to the application area.

6.1. Implementation

The learning system consists of three primary components. First, actor processes are responsible for acting in the environment using the current policy to generate trajectories and fake images for policy and discriminator training, respectively. Second, Proximal Policy Optimization, previously presented in Section 2.2.4, is used to optimize the policy. Finally, the discriminator is optimized using real examples and fake paintings from the previous simulation. The updated discriminator is used in the following rollout to collect reward signals, and the process repeats until convergence. Figure 6.1 shows an overview of the learning algorithm cycle.

The rollout step is the most computationally expensive part of the process due to the expensive rendering process and the many model inferences necessary (policy and reward predictions for each step). In a typical scenario, the rollout step consumes around 60% of the total wall clock time. Generating trajectories is an embarrassingly parallel[1] task due to the complete lack of dependency between different episodes. On capable computers, performance can be improved by sharing model memory and running several rollouts simultaneously.

Although the PPO algorithm is also parallelizable, as shown by Heess et al. (2017), this work implements the sequential variant. This choice is motivated by the comparatively lower computational cost of policy optimization compared to sample collection, and the higher implementation complexity associated with a parallel solution.

The system executes policy optimization and discriminator training in discrete sequential steps, but since these steps are independent, they could be easily parallelized. Discriminator training is, however, much faster than policy optimization in practice, and performance benefits would presumably be minor considering the overhead associated with parallel implementations.

[1]Embarrassingly parallel problems, known from parallel computing, can easily be separated into smaller parallel subtasks and executed without sequential dependencies or communication.

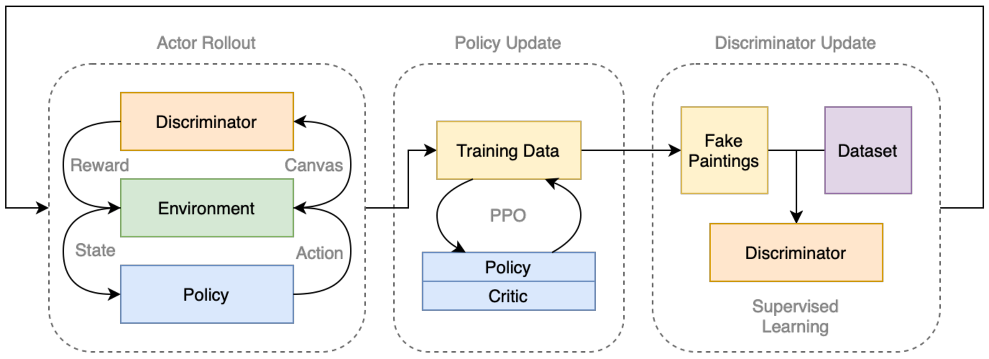


Figure 6.1.: **Illustration of the learning algorithm cycle.** Actors collect training data used by PPO to optimize the agent. The discriminator trainer optimizes the discriminator model used in the adversarial reward system, and the process repeats until convergence.

With the combination of vectorized environments (as discussed in Section 4.3) and parallel rollouts, typical runs execute the whole training cycle at a rate of between 50 - 200 environment steps per second[2]. Performance varies depending on training configuration hyperparameters, length of episodes (longer episodes are faster due to lower overhead) and the rendering complexity of the specific brush.

6.2. Policy Training

The algorithm uses an entropy bonus in the policy loss function to encourage exploration. This type of exploration is a common technique used in PPO, and was suggested by Schulman et al. (2017) in their original paper. The entropy loss is given by the equation:

$$L_E = -\alpha H(\pi(\cdot \mid s_t)) \tag{6.1}$$

Where α is the entropy coefficient hyperparameter, and H is the entropy function:

$$H(P) = \mathop{\mathbb{E}}_{x \sim P}[-\log P(x)] \tag{6.2}$$

This type of exploration ensures that the policy remains sufficiently random throughout training, and does not commit to suboptimal low-entropy distributions too early. The entropy coefficient is one of the most important hyperparameters in the algorithm. Most experiments in this work use an entropy coefficient of 0.01, which works well for most environments. Entropy and exploration is discussed in more detail in Section 7.5.

Each training step in PPO consists of several epochs. At the beginning of each epoch, the sampled environment transitions are first divided into a set of random minibatches. For each minibatch, the following steps are then performed:

[2]Performance as measured with one NVIDIA Tesla P100 GPU, an Intel Xeon E5-2650 v4 processor, and 128GB of RAM on the IDUN HPC cluster provided by Själander et al. (2019).

- Run inference with the agent model to estimate log probabilities, entropies and value estimates for the sampled data
- Estimate returns and advantages using either TD, RTG or GAE (see Section 6.3)
- Calculate the entropy, value, and PPO surrogate loss (as described in Section 2.2.4)
- Use backpropagation to estimate the gradient of model parameters with respect to the total loss, and optimize the model parameters using some optimization algorithm

This algorithm uses the RMSprop optimizer by Hinton (2018) to optimize all neural network models. The optimizer is set to use the default PyTorch configuration. Although the Adam optimizer by Kingma and Ba (2014) is also commonly used in reinforcement learning, RMSprop is chosen for this method because the role of momentum in Adam is unclear in a non-stationary adversarial setting.

6.3. Advantage Estimation

Advantage estimation is a core part of actor-critic reinforcement learning, and is crucial because it directly changes the direction of policy gradients during optimization. Advantages misaligned with the objective will always cause learning to fail. As discussed in Section 2.2.3, the advantage may use any baseline z_t independent of θ (typically the estimated value function $V(\cdot)$) and is given by the equation:

$$A(s_t,\ a_t) = \mathcal{R}_t - z_t \tag{6.3}$$

Where $\mathcal{R}_t$ is some measure of the expected reward following the current policy from state s_t. Standard reinforcement learning algorithms, including PPO by Schulman et al. (2017), DQN by Huang (2020) and DDPG by Lillicrap et al. (2015), among others, rely on three primary ways to calculate the expected future reward:

- **Monte Carlo Sampling**
  Monte Carlo samples (also known as returns-to-go, RTG) can be used to obtain an unbiased future estimate directly from the environment and thus requires a complete rollout of the episode. The sample expectation is given by:

  $$\mathcal{R}_t^{RTG} = \sum_{i=t}^{T} \gamma^{(i-t)} R(s_i, a_i) \tag{6.4}$$

  Where γ is the discount factor and $R(s_i, a_i)$ is the sampled reward received when performing action a_i in state s_i. Due to being sampled directly from the environment, this estimator has no bias but is susceptible to high variance.

- **Temporal Difference Learning**
  In temporal difference learning, we bootstrap the estimate using the current value function. In the simplest one-step variant (TD-1 learning), the return is given by:

$$\mathcal{R}_t^{TD} = R(s_t, a_t) + \gamma V(s_{t+1}) \tag{6.5}$$

  Temporal difference learning may bootstrap from any step after t (N-step TD learning), in which case some number of discounted environment samples precede the immediate bootstrapped value. Due to the bootstrapped value estimation, TD learning is biased but has a lower variance than direct sampling. Unlike RTG, temporal difference learning is not dependent on a complete rollout before updating the policy.

- **Generalized Advantage Estimation**
  Generalized Advantage Estimation (GAE), as presented in the context of policy gradient methods by Schulman et al. (2015b), is a generalization of the previous methods. GAE enables direct control of the bias-variance tradeoff and is given by the equation:

$$\mathcal{A}_t^{GAE} = \sum_{i=t}^{T} (\gamma\lambda)^i \delta_i^V \tag{6.6}$$

  Where δ_i^V is the temporal difference estimate at time i given by the value function V. The hyperparameter $\lambda \in (0, 1)$ directly controls the bias-variance tradeoff (greater values translate to higher variance and lower bias). The GAE estimator also has two special cases. When $\lambda = 1$, the estimator reduces to the pure Monte-Carlo advantage estimate, and when $\lambda = 0$, it is equivalent to the 1-step TD estimate.

Whereas similar previous work used the sampled Monte-Carlo return estimates exclusively, this thesis shows that agents can learn effectively using other alternatives as well. This is discussed in Section 7.8, which presents the effect of various advantage estimation techniques in more detail.

Advantage estimation is closely linked to the training of the value function. The return estimators presented above are used as output targets when training the value function with RTG or TD. Equation 6.6 defines the advantage directly, so the GAE return target is found by first adding the value estimate $V(s_t)$.

6.4. Reward System

Designing sound reward systems for reinforcement learning is a challenging task. Agents trained using gradient ascent are notoriously fond of simple solutions — as described by Amodei et al. (2016), reward hacking is not only difficult to predict, but is expected behavior whenever the objective is underspecified. In the context of adversarial learning, the difficulty is increased further by the inherent uncertainty associated with discriminator function approximation and continuous competition between models.

This thesis presents four reward system variants, two of which have previously been shown to work in similar contexts by Ganin et al. (2018) and Mellor et al. (2019). The remaining variants extend on these to increase the quality of reward signals and tackle challenges related to unbounded discriminator predictions.

6.4.1. Episodic Reward System

The episodic reward system introduced by Ganin et al. (2018) can be viewed as the direct parallel to the generator loss found in generative adversarial networks. In this setting, the agent is only rewarded at the terminal step of the episode. The basic episodic reward system is given by:

$$\mathcal{R}(t) = \begin{cases} D(C_t) & \text{if } t = N \\ 0 & \text{otherwise} \end{cases} \tag{6.7}$$

Where C_t is the canvas at timestep t, D is the discriminator model, and N is the total number of steps in the episode. The equation reduces to maximizing $D(C_N)$ for any return estimator. Although this system is guaranteed to capture the goal of maximizing the discriminator evaluation, the rewards are inherently sparse, which complicates the credit assignment problem for long episodes. Sparse rewards may reduce the speed of learning or prevent the learning of long-term dependencies entirely.

Because the evaluation of $D(\cdot)$ changes over time, this formulation is also problematic with respect to value functions in actor-critic algorithms. This issue is particularly evident in unbounded Wasserstein systems, where returns may be drastically biased from earlier experience, which is likely to prevent convergence.

Expectation Adjustment

This thesis introduces a novel expectation adjustment technique intended to remedy issues associated with changes in the discriminator model by incorporating an expectation of current policy performance in the reward system. The expectation adjustment function is defined as:

$$\mathcal{R}^A(t) = \frac{D(C_t) - \mathbb{E}[D(C_N^\pi)]}{\mathbb{S}[D(C_N^\pi)] + \epsilon} \tag{6.8}$$

Where $\mathbb{E}[\cdot]$ and $\mathbb{S}[\cdot]$ are the expectations in discriminator estimated mean and standard deviation of images C_N^π generated by the current policy. The constant ϵ is a small value included for numerical stability. This work uses $1e-8$, which provides a good balance between bias and stability. Values in this range are commonly used for similar tasks, for instance for advantage normalization in the PPO implementation by Raffin et al. (2021).

In the expectation adjusted basic reward system, $\mathcal{R}^A(C_t)$ replaces $D(C_t)$ in Equations 6.7 and 6.10. The expectation adjusted reward may be viewed as a type of on-policy reward normalization.

The implications of this change are two-fold. First, the rewards are implicitly normalized, which reduces numerical scaling issues associated with deep model optimization. Second, the expectation adjustment effectively modifies the pure maximization objective of the learning algorithm by introducing a dynamic baseline:

$$J(\pi_t) = \underset{\pi_t}{\operatorname{argmax}}\, \mathbb{E}[D(C_N^{\pi_t}) - \mathbb{E}[D(C_N^{\pi_{t-1}})]] \tag{6.9}$$

Note that due to the temporal dependency in this equation, there is no way to find the global maximum using gradient ascent, even if discriminator training is halted — this is counter to the typical maxim goal in reinforcement learning. The (local) maximum in terms of the intended goal of fooling the discriminator will occur when the objective converges to zero, at which point the policy cannot improve further.

In principle, this change necessitates early stopping at convergence, given that the policy remains stochastic. Since the expectation will be strictly ≤ 0 at the point of convergence, the policy entropy could increase unpredictably beyond this point and, consequently, reduce the quality of produced images. In practice, strict convergence does not occur for these deep neural network models, and the objective gradient remains useful throughout training.

6.4.2. Temporal Reward System

The temporal reward system was first introduced by Mellor et al. (2019) in SPIRAL++. The system introduces intermediate rewards every step to increase the density of rewards and hence reduce the difficulty of the credit assignment problem. The temporal reward system is given by:

$$\mathcal{R}^T(t) = \begin{cases} D(C_t) & \text{if } t = 0 \\ D(C_t) - D(C_{t-1}) & \text{otherwise} \end{cases} \tag{6.10}$$

The objective reduces to $D(C_N)$ given that the discount factor $\gamma = 1$:

$$D(C_0) + \gamma(D(C_1) - D(C_0)) + \ \ldots \ + \gamma^{N-1}(D(C_N) - D(C_{N-1})) = D(C_N)$$

The authors showed that the formulation worked well in practice for other values of γ as well.

Expectation Adjustment

The temporal system can be combined with expectation adjustment to yield the expectation adjusted temporal reward system:

$$\mathcal{R}^{AT}(t) = \begin{cases} R^A(t) & \text{if } t = 0 \\ R^A(t) - R^A(t-1) & \text{otherwise} \end{cases} \tag{6.11}$$

As shown by Mellor et al. (2019), temporal rewards improve results for long generative sequences. The results in this thesis confirm this claim (see Chapters 7 and 8), and the majority of this work, therefore, uses a temporal reward system. The effect of various reward systems is explored further in Section 7.7.

6.5. Reproducibility

As previously discussed by Pineau et al. (2020) among others, the lack of reproducibility of academic results is a consistent challenge in machine learning research. In many cases, these issues are caused by the lack of consideration to implementation detail — particularly by omittance of certain *tricks* and *adjustments* applied to the training procedure to stabilize its convergence properties. These changes are often motivated by the observed numerical properties of a system rather than strong, theoretical foundations.

As shown by Engstrom et al. (2020), minor implementation details have significant effects on policy gradient methods. This section presents such details of this implementation. Although secondary to the primary design decisions presented previously, they are necessary in practice to ensure the stability of the learning process.

6.5.1. Training Stability

The implementation relies on three primary implementation details for increased stability. First, L2 gradient clipping of policy parameters reduces issues with exploding gradients. All experiments in this work use a clipping value of 0.5, which Raffin et al. (2021) have shown to work well on many problems.

Second, the implementation uses reward scaling to ensure returns lie in similar ranges for various discriminator types. Scales between 1 and 10 work best for this method, but may depend on the specifics of the environment.

Finally, the implementation clips rewards exceeding a certain absolute threshold. Rewards are mostly well behaved, but on rare occasions (especially when using an unbounded discriminator), outliers can cause irreversible damage to a policy model. This work uses a default reward clipping threshold of 10, which works well for all environments considered. The exact value of the threshold is not crucial for performance — most values increase stability as long as a majority of rewards in the expected range are preserved.

6.5.2. Advantage Estimation

Although most implementations of PPO use normalized advantages based on temporal difference learning or generalized advantage estimation, the way in which they are calculated during training varies. Consider the main training procedure in PPO, which consists of some number of epochs iterating over minibatches of transition examples. The advantage may be calculated and normalized before the first epoch, across the entire batch once per iteration, or for each minibatch individually. Some implementations estimate the baseline per iteration, whereas others calculate it during rollout before training begins. These choices may significantly affect the stability and performance of a system.

This work bases these choices on a few high-quality implementations. Hill et al. (2018) at OpenAI provide *stable-baselines*, an open-source implementation of the most common reinforcement learning algorithms. Advantages in all their PPO implementations are calculated during rollout, whereas normalization occurs per batch in their initial PPO1 algorithm and per minibatch in the PPO2 extension. The improved *stable-baselines3* by Raffin et al. (2021) employs the same technique as PPO2. The *acme* PPO implementation by Hoffman et al. (2020) for DeepMind calculates advantages before training, and normalizes the values per minibatch.

Based on these insights, the implementation in this work calculates advantages and return targets during rollout and normalizes advantages per minibatch. This method proved reliable in practice.

7. Experiments

The following chapter presents the empirical studies conducted for the thesis. The primary purpose of these experiments is to collect the necessary evidence to answer the research questions presented in Section 1.2. The chapter also exposes challenges, strengths, and other insights related to the proposed method.

Experiments are divided into sections exploring specific aspects of the neural network architectures and learning algorithms. Important topics include policy parameterization, discriminator designs, reward systems, and exploration. Section 7.6 explores a particular challenge associated with the method, referred to as *mode oscillation*.

7.1. Experimental Setup

The majority of experiments presented here employ the IDUN high-performance cluster by Själander et al. (2019). Smaller experiments and development trials are executed on a desktop computer with an NVidia GeForce GTX 970 GPU, an Intel i5 4690k processor, and 16GB RAM. Due to the memory constraints of the GTX 970 GPU, most experiments use models consuming less than 3.5GB of video memory. This constraint encourages fewer memory expensive convolutional layers in the feature extractor and discriminator model, though their size remains typical for the context and works well in practice.

All experiments use Weights and Biases by Biewald (2020) for tracking. More than 100 different metrics are reported in real-time to enable analysis and debugging. The exact number of tracked metrics varies depending on various hyperparameters and the environment configuration. Key metrics include policy metrics such as entropy, action and advantage distributions, discriminator metrics such as error rate, separation, and reward distribution, and various training metrics such as losses, gradients, and model parameters. Each tracked experiment also stores its configuration and a git commit containing the relevant code, enabling a more straightforward comparison of low-level differences between experiments. See Appendix A.4 for an example of the tracking system used for experiments.

The method is highly customizable and relies on a large number of tweakable hyperparameters and configurations. Please refer to Appendix A.3 for an overview of the default configuration. Experiment-specific changes are detailed in the following sections where relevant.

Figure 7.1.: **Running mean as a proxy performance metric.** The running mean of generated images serves as a useful metric to detect learning progress. This example shows the running mean of the 512 most recent grayscale CelebA paintings over the course 5×10^5 timesteps of training in one experiment.

7.2. Detecting Learning Progress

It is notoriously difficult to evaluate learning progress and convergence in adversarial systems. Unlike typical problems in reinforcement learning, increases in cumulative reward are not necessarily indicative of learning due to the dynamic nature of the reward system, and the quality of produced images typically increases independently of changes in rewards. In fact, no metric is exactly correlated with the quality of produced paintings — if such a metric existed, there would be no need for adversarial learning. Instead, this work relies on multiple proxy metrics that typically correlate with learning progress.

Although an increase in (subjective) image quality evaluation is a good metric, it is not always possible to observe general improvement in individual examples, especially when the search is broad. To detect progress in these situations, the system uses per-pixel image averaging over several episodes. Throughout training, it is easy to detect patterns in these aggregates indicative of learning. Figure 7.1 shows an example of this effect, where the agent was trained on images from the CelebA dataset.

In addition to visual indications of progress, a large number of other metrics are tracked throughout training to evaluate the system. Some of these are particularly useful to detect learning progress:

- Difference in mean and standard deviation between real and painted images. This metric is useful because the discriminator requires the distributions to be similar to yield a high evaluation. Good agents almost always score low on this metric.
- Aggregate normalized discriminator predictions for each frame in the episode. Generally, we wish to observe an increasing trend in the discriminator evaluations throughout an episode. This metric indicates that the discriminator is learning (it is fooled less by incomplete paintings than complete paintings) and that the policy properly utilizes each stroke to improve the output.
- Policy entropy. The policy entropy is a useful metric to detect convergence in reinforcement learning problems. Lower entropy is not necessarily a sign of quality outputs, but consistent high-quality outputs are necessarily generated by a relatively low entropy policy. In combination with other metrics, the policy entropy serves as a good measure of convergence over time.

- Mean discriminator separation of real and fake images. This metric typically increases gradually over the course of training, but occasionally drops when the policy discovers a way to paint some significant feature of the target domain.

As previously mentioned, none of these metrics alone are sufficient to accurately measure model performance, but together they prove very useful to detect progress and enable comparison of different experiment configurations. Evaluation metrics are considered further in Section 8.1.1.

7.3. Policy Parameterization

This thesis builds on preliminary work in a preparatory project (see Røkenes 2021). The preliminary work employed a simple policy architecture based on continuous distribution parameterizations, rather than the autoregressive architecture described in Section 5.4. This section explores the performance of various policies on more complex problems. Four different policy variants are considered:

- **Continuous Gaussian Policy.** This parameterization is standard for reinforcement learning environments with continuous action spaces. The variant used here models the log standard deviation as a function of the input state, but it is also commonly implemented as an independent trainable variable.

- **Continuous Beta Policy.** Because the Gaussian distribution is unbounded, it may suffer from issues in bounded action spaces. This policy attempts to remedy these issues by using a beta distribution scaled to the target range.

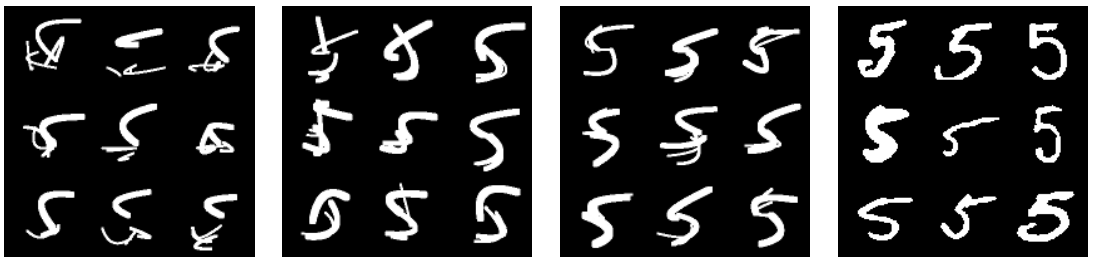
- **Discrete Categorical Policy.** This policy variant employs a standard categorical distribution that divides the continuous action space dimensions into a set of discrete bins.

- **Discrete Autoregressive Policy.** As described in Section 5.4, this policy enables a simpler representation of high dimensional categorical probability distributions through an autoregressive architecture with sample embeddings.

See Appendix A.2 for an overview of the architecture of the continuous and categorical policies. The policies are compared on the MNIST and CelebA datasets (see Section 2.5), which are the easiest and most complex settings explored in this thesis. The experiments use the WGAN-GP discriminator and basic temporal reward system without expectation adjustment. Discriminators and reward systems are discussed further in Sections 7.4 and 7.7.

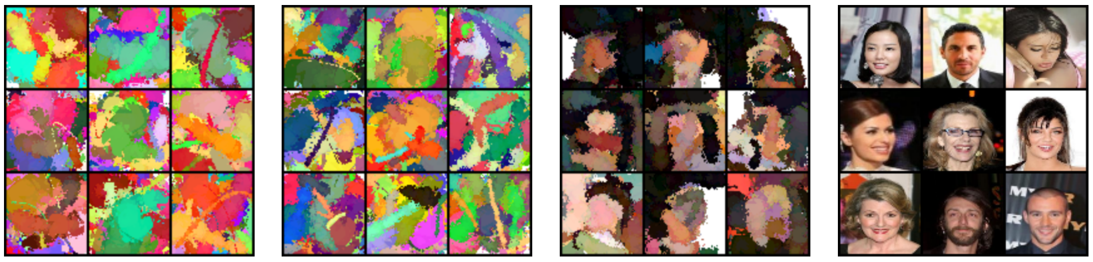
As shown in Figure 7.2, both the autoregressive and categorical policies are capable of producing adequate results using the MNIST dataset. The continuous policies show some learning ability but suffer from frequent instability and policy collapse.

The performance difference between policies is more evident for longer episodes trained using the CelebA dataset. Neither the categorical nor continuous policies can reach the



(a) Gaussian (b) Categorical (c) Autoregressive (d) Training Dataset

Figure 7.2.: **Comparison of MNIST paintings for different policiess.** The policies are trained to reproduce digit 5s using four hard rounded bezier strokes for 20k episodes. All policies are capable of some learning with this dataset.



(a) Gaussian (b) Categorical (c) Autoregressive (d) Training Dataset

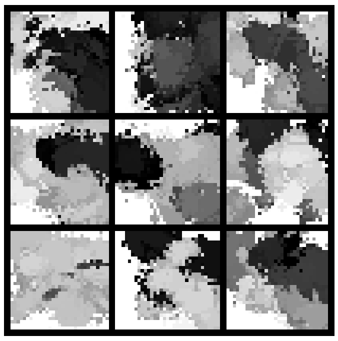
Figure 7.3.: **Comparison of CelebA paintings for different policies.** The figure shows result after initial training for 20k episodes. Whereas the autoregressive policy displays clear evidence of learning, the categorical and Gaussian policies remain similar to the untrained policy.

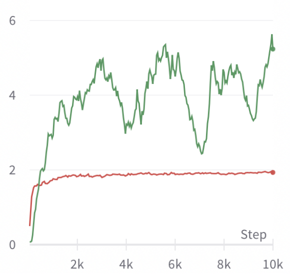
level of the autoregressive variant and fail to converge to recognizable results. As shown in Figure 7.3, the autoregressive policy is much more capable on difficult problems. The figure illustrates the necessity of a more complex model for these types of problems. All other experiments described in this thesis use the autoregressive policy presented in Section 5.4.

7.4. Discriminators

Developing a stable and capable discriminator suitable for adversarial rewards is challenging. Due to the stochastic nature of both the environment and policy, the distribution of fake images can change substantially between training iterations. Long generative sequences further exacerbate this problem. If the distribution of canvases early in a trajectory changes only slightly, the implicit chaos in the system can easily cause a large shift in the final distribution. These system properties greatly increase the challenge of training a discriminator with sufficient generalization ability.

Furthermore, high-quality predictions on final images exclusively are not sufficient for


(a) LSGAN



(b) Mean Separation

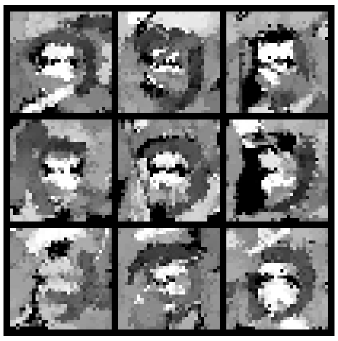

(c) WGAN-GP

Figure 7.4.: **Comparison of LSGAN and WGAN-GP.** Although LSGAN is able to separate fake and real examples, the policy fails to converge with this discriminator. Figures show the mean separation between fake and real images for LSGAN (red) and WGAN-GP (green) and sample outputs after 40k episodes of training on CelebA (grayscale).

high-quality temporal rewards. The relative predictions for intermediate canvases must also be informative to enable useful signals to the agent. The discriminator may never see these examples, yet the model must capture the presence or absence of desirable features in the images to enable policy learning. This is an unusual problem, as we typically do not care about model behavior on data that is explicitly not part of the learned distribution. Motivated by this observation, implementations with intermediate canvases included in the training dataset are explored in Section 7.4.4.

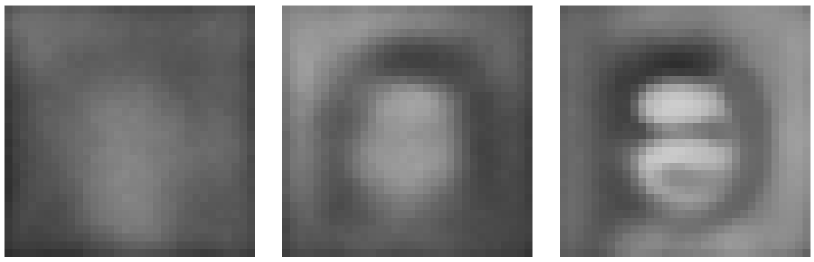
As will be shown in the following sections, the system's performance is greatly affected by the training procedure and setup of the discriminator model. Various algorithms, regularization techniques, and novel extensions are explored to discover their properties and applicability in this context. The GAN algorithms investigated include WGAN by Arjovsky et al. (2017), WGAN-GP by Gulrajani et al. (2017), minimax by Goodfellow et al. (2014), and LSGAN by Mao et al. (2016).

7.4.1. Least Squares GANs

Mao et al. (2016) propose Least Squares GAN (LSGAN) as a more stable algorithm that overcomes problems of diminishing gradients in cross-entropy based methods. The LSGAN discriminator loss is given by the mean squared error of predictions and target labels:

$$L = \frac{1}{2} \mathop{\mathbb{E}}_{x\sim R}[(D(x) - a)^2] + \frac{1}{2} \mathop{\mathbb{E}}_{x\sim G}[(D(x) - b)^2] \tag{7.1}$$

Where a and b are target labels, typically 1 and -1 respectively. LSGANs can separate fake and real examples consistently in this setting (see Figure 7.4b), which is promising in terms of the learning signals they could provide.



(a) 1.5k Episodes (b) 10k Episodes (c) 40k Episodes

Figure 7.5.: **Early indications of learning in WGAN-GP.** WGANs typically show indications of learning quicker than other methods. This example shows the pixel-wise mean of the 512 most recent paintings. Some indications of learning are visible after only 1500 episodes.

Unfortunately, experimental results demonstrate that LSGAN is unsuitable in the adversarial painting context. Despite the separation ability of the discriminator, agents trained with the LSGAN reward system do not convergence on any dataset, including short 3-step episodes using MNIST. A comparison of LSGAN and WGAN-GP is shown in Figure 7.4.

These results indicate that the reward surface produced by LSGAN is incompatible with temporal rewards and is worse than alternatives in general. Since the generator (policy) is not optimized directly as in typical GANs, but through a detached reinforcement learning procedure, claims of improved discriminator gradients with respect to the input are also inconsequential to agent training.

7.4.2. Wasserstein GANs

Wasserstein GANs (WGANs) are generative adversarial networks where the discriminator attempts to maximize the earth mover distance between real and fake examples. As shown by Ganin et al. (2018) in the SPIRAL paper, WGANs can be used successfully in the context of generative adversarial reinforcement learning.

This work provides further evidence that the algorithm is well suited for adversarial learning. Compared to other types of discriminators, WGANs prove more lenient in terms of hyperparameters and show signs of convergence (as discussed in Section 7.2) faster than alternatives. Figure 7.5 shows an example where learning is detectable after only 1500 episodes. For minimax discriminators, these patterns are usually not visible until at least 10k episodes of training. Even the pure WGAN objective with no regularization displays initial signs of learning, as shown in Figure 7.6, though quality quickly deteriorates after further training.

In the original WGAN paper, Arjovsky et al. (2017) propose the use of parameter clipping to regularize the model and enforce the Lipschitz constraint.[1] The gradient

[1]A 1-constraint on the Lipschitz constant in Lipschitz continuous functions. In this context, the constraint entails a limit to how fast network predictions can change based on changes in the input.

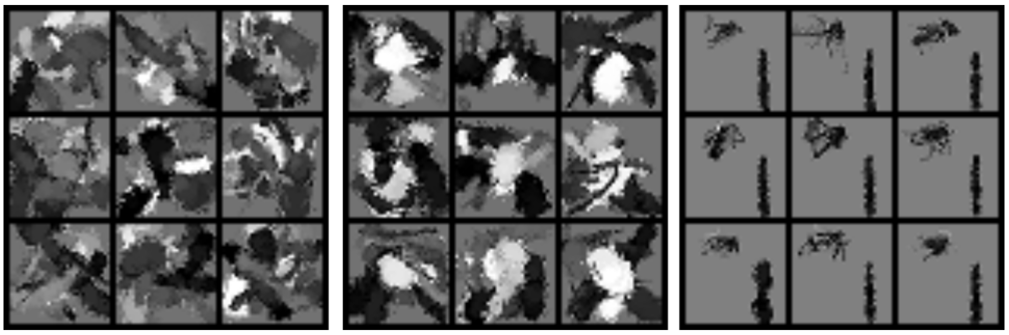


Figure 7.6.: **Grayscale results on CelebA using an unregularized WGAN discriminator.** Results after no training, 20k episodes and 30k episodes, respectively. Although training with the unregularized discriminator shows signs of learning (such as brighter areas in the center where faces should be), the model quickly deteriorates after further training.

penalty extension in WGAN-GP, later introduced by Gulrajani et al. (2017), encourages the constraint using an extension to the loss formulation instead. This work shows that although both types enable learning in the application area, the gradient regularization method has clear benefits over the simpler clipping variant.

Parameter Clipping

The parameter clipping regularization constraint is very simple. Following each gradient descent update, discriminator parameters are clipped to a range determined by a hyperparameter c:

$$\theta^D = \text{clamp}(\theta^D, -c, c) \tag{7.2}$$

Arjovsky et al. (2017) suggest a clipping value of 0.01. This work explores performance with values of 0.001, 0.01, and 0.1. The smallest value performs poorly and does not show signs of convergence. The other choices of c enable learning comparable to Figure 7.6. The clipped model still suffers from policy collapse, although the issue is less prominent than in the unregularized variant. The technique is not explored extensively due to apparent issues and poor performance in initial tests.

Parameter clipping has been criticized in previous literature as it entails a hard mathematical constraint on the learning procedure. As indicated by Arjovsky et al. (2017), clipping is a poor way to enforce the Lipschitz constraint and often suffers from issues such as diminishing gradients and a slow rate of convergence. Gulrajani et al. (2017) further suggest that gradient clipping typically leads to a pathological[2] discriminator surface. This is particularly problematic in the temporal reward context because the

[2]Pathological objects, in mathematics, are objects that are irregular and not *well-behaved* compared to other objects in the same domain. In this contex, a pathological prediction surface may be particularly noisy or uneven.

agent relies on relative signals based on small changes in the canvas. A pathological surface may introduce excessive amounts of noise, which could inhibit learning. These claims are supported by the observed behavior of the method and are especially evident when compared with the gradient penalty variant.

Gradient Penalty

The Wasserstein gradient penalty algorithm augments the discriminator loss function with a soft constraint on the norm of the gradient. The gradient penalty has the effect of encouraging the L2 norm of the discriminator gradient to be close to 1 with respect to relevant inputs:

$$GP = \lambda \mathop{\mathbb{E}}_{x \sim Z}[(\| \nabla_x D(x) \|_2 - 1)^2] \tag{7.3}$$

Where λ is the gradient penalty coefficient hyperparameter, and Z is the distribution of interpolated samples on a straight line between the real and fake distributions. Gulrajani et al. (2017) showed that a λ of 10 worked well for most datasets. Initial results indicated that this value is appropriate in this context as well, and it is therefore used for all WGAN-GP training. The distribution Z is generated from uniform linear interpolations between some random fake image x and real image y:

$$\gamma x + (1 - \gamma) y, \quad \gamma \sim U(0, 1) \tag{7.4}$$

As explained by Gulrajani et al. (2017), enforcing the constraint for all inputs is intractable. In practice, using only Z is sufficient for good experimental results.

The gradient penalty discriminator works better than the clipped variant and has produced some of the best results in this thesis (see Sections 7.4.5 and 8). Unlike the LSGAN discriminator, temporal rewards provided by this model are sufficient for stable learning.

Although the method shows desirable behavior, output images tend to lack finer details produced by the minimax variant described in the next section. The quality of produced images improves quicker than in other methods, but the policy is more frequently captured in suboptimal local maxima. This is discussed further in Section 7.4.5.

7.4.3. Minimax GANs

The minimax objective was introduced in the original GAN paper by Goodfellow et al. (2014). This algorithm trains a discriminator as a binary classifier minimizing the cross-entropy of predictions, as described in Section 2.4.1.

Initial experiments with this algorithm were unsuccessful. Although the system can generate recognizable images from the MNIST dataset in some trials, the method is susceptible to network initialization and often collapses after less than 100 episodes. This issue is caused by an imbalance between the discriminator and generative agent. The minimax discriminator quickly learns to separate between real examples and fake

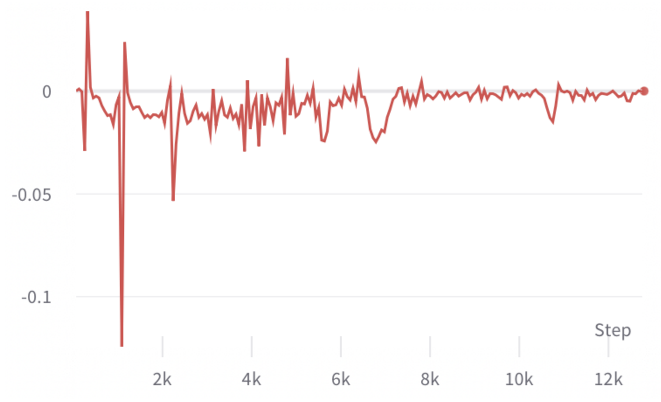


Figure 7.7.: **Collapsing rewards for pure minimax discriminators.** Mean per-episode reward for an agent trained using the pure minimax discriminator. The reward signals quickly collapse to small values, which prevents the agent from learning.

paintings, which causes the resulting rewards to grow very small. The poor reward distribution inhibits agent learning, as small inaccuracies in the critic model will prevent accurate advantage estimation. An example of collapsing minimax rewards is shown in Figure 7.7.

Logit Reward System

Motivated by the behavior of minimax, the reward system is modified to employ the raw logit predictions rather than probability estimations produced by the following sigmoid activation. In this implementation, the training procedure of the discriminator is identical — only the reward signal is different. This change proved one of the most impactful design decisions in the adversarial system. The best results in this thesis employ this type of minimax discriminator with logit rewards. Directly employing the linear output of the discriminator enables the system to provide a more informative landscape for temporal rewards. Meanwhile, retaining the cross-entropy loss formulation ensures that the discriminator grows highly competent over time. Together, these factors facilitate both stable training and higher quality results.

Given that the policy discount factor is 1, the maximization of the logit reward objective is equivalent to the minimax formulation. As shown in Section 6.4.2, the temporal reward system in these cases reduce to $\sigma(D(C_N))$, where D is the linear discriminator model and σ is the sigmoid function. For the logit variant, we have $D(C_N)$. Since the sigmoid function monotonically increases with its input, the maxima are equivalent.

The temporal distribution of rewards is not equivalent in this setting. Consider the graph of logit and sigmoid deltas over the bounded probability range shown in Figure 7.8. Given a linear increase in image quality as evaluated by D, rewards will be proportional to the logit and sigmoid deltas. In the sigmoid case, rewards are concentrated near the

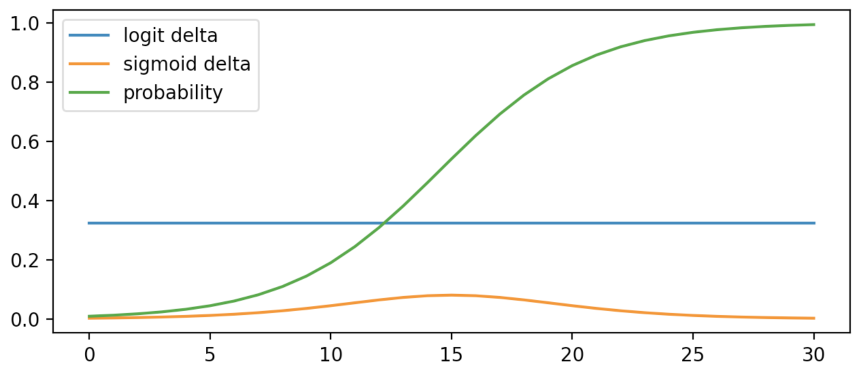


Figure 7.8.: **Distribution of logit and sigmoid delta rewards**

center, where the probability increases most rapidly. The logit deltas are constant over the whole range.

Although paintings do not improve linearly in practice, the reward landscape is more evenly distributed over the full range of environment steps when using the logit reward variant. This change eases the task of estimating advantages, particularly for small strokes late in an episode when image evaluations change more slowly. This, in turn, enables the policy to utilize late strokes more efficiently, thus improves the fidelity of paintings. The logit-based system is also less sensitive to the absolute quality of images relative to discriminator performance.

Equivalent reward shaping can likely be emulated using other techniques, such as reward or return normalization. This thesis does not explore these options due to scope considerations, but the topic remains an interesting area of future work.

7.4.4. Regularization Techniques

In supervised learning, regularization is typically used to improve model generalization. In the context of this thesis, the primary goal is not generalization, but to improve the quality of rewards. Although generalization could improve performance in situations with high distributional shift, common techniques can have detrimental side effects on the reward system. This section explores regularization techniques with these considerations in mind.

Label Smoothing Regularization

Szegedy et al. (2015) introduce the concept of label smoothing regularization (LSR) intended to improve the generalization of classifier models. LSR builds on the default cross-entropy loss, but with some probability ϵ, the true label is replaced with a sample from the prior distribution. This modification prevents the model from becoming too confident and overfitting to the training data. The authors showed that this kind of regularization reduced their top-5 error rate on ImageNet with 1000 classes.

In the painting context, generated images are necessarily quite different from the target dataset — especially when more abstract brushes (such as the hard squared brush) are used. If the discriminator grows too confident and consistently relies on photo-realistic features that are impossible to reproduce in the painting environment, the reward signals could deteriorate and prevent the agent from learning. These insights motivate the introduction of LSP in this system.

Since LSR was designed for the cross-entropy loss formulation, the technique is only considered for the logit-based minimax discriminator. Unfortunately, LSR shows no signs of improved system performance or change in learning behavior for any $\epsilon \in \{0.01, 0.05, 0.1\}$.

Intermediate Canvas Training

As previously discussed, temporal reward systems depend on predictions on incomplete canvases from a distribution never seen by the discriminator. This section explores whether training on intermediate paintings can improve generalization to this data, and thus increase reward quality. Three variants of this technique are considered:

1. **Complete intermediate training.** In this implementation, the discriminator is trained using every intermediate and complete painting. This results in more discriminator updates per agent update, depending on the length of episodes.

2. **Mixed intermediate training.** With this variant, each training batch consists of a mix of 50% finished and 50% intermediate paintings. As a result, a large portion of available training data must be discarded each iteration.

3. **Temporally weighted intermediate training.** In this setting, the discriminator loss is weighted per sample based on its relative step in the episode. Earlier steps are weighted more strongly, encouraging the discriminator to evaluate fake images on a temporally consistent scale. Weights are given by the equation $\frac{0.5(N-t)}{N} + 0.5$, where N is the number of steps in the episode. See Figure 7.9 for an illustration of this technique.

The introduction of intermediate paintings has either negative or inconsequential effects on both the logit-minimax and WGAN-GP discriminators. The first variant performs equivalently to the unregularized model on both MNIST and Fashion-MNIST, but is slower to train than the unregularized model. The technique does not converge using CelebA. The second variant performs similarly to the unregularized method and learns using all datasets. The third variant does not work on any dataset besides MNIST.

The method is associated with increases in computational cost. As a result of the parallel and vectorized implementation of the environment, including intermediate canvases necessarily increases the overhead of copying images from GPU to CPU memory and transferring data between processes. Due to additional overhead and poor results in initial tests, this technique is not considered further.

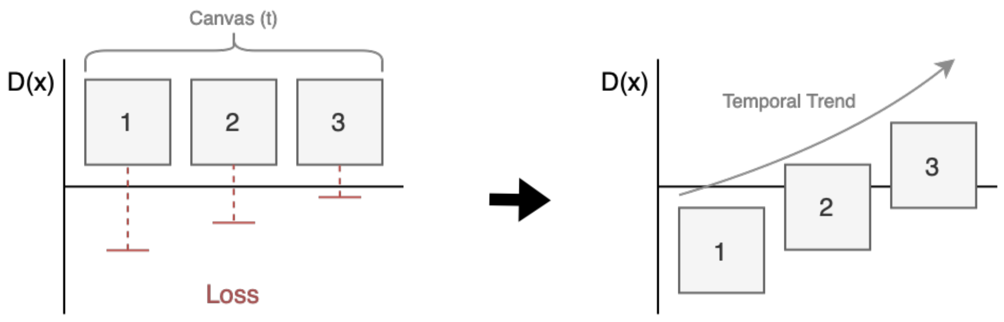


Figure 7.9.: **Temporally weighted intermediate training in the discriminator.** Early canvases contain fewer details, and should generally be evaluated lower than later canvases. This technique is intended to encourage the model to produce a temporally consistent reward landscape.

Normalization Layers

Three different normalization layers are considered for the discriminator model — batch normalization, layer normalization, and instance normalization. These layers are commonly used in the field and work well for many different models.

Batch normalization was shown by Ioffe and Szegedy (2015) to significantly improve the performance of classifier models while reducing sensitivity to model initialization. Batch normalization remains standard for advanced generative methods such as DCGAN by Radford et al. (2015). As shown by Gulrajani et al. (2017), this type of normalization breaks the gradient penalty loss in WGAN-GP due to dependencies between instances in a batch. This layer is therefore only considered for the minimax discriminator.

Batch normalization works by learning properties of the input distribution throughout training and using the statistics during inference. Although batch normalization can reduce the variance of model predictions, the technique is incompatible with temporal rewards. The learned statistics are not representative of intermediate canvases, which results in invalid predictions. Performance of batch normalization combined with intermediate canvas training remains comparable to the unregularized model.

Gulrajani et al. (2017) recommends layer normalization as a replacement for batch normalization layers when using the gradient penalty loss formulation. In this context, layer normalization works well for both the WGAN-GP and logit-minimax discriminator. Interestingly, layer normalization causes the agent to be more agnostic in terms of color choice, and paintings produced by this type of agent include strong, unconventional colors (compared to the dataset) more often than without layer normalization. This is shown in Figures 7.10 and 7.11. Besides this observation, layer normalization causes no apparent differences in overall system performance.

Instance norm, previously used in generative methods such as CycleGAN by Zhu et al. (2017), applies normalization across features on a per-sample basis rather than the whole training batch. Unfortunately, instance norm works poorly in this context and is unable to achieve results comparable to both the unregularized and layer normalized

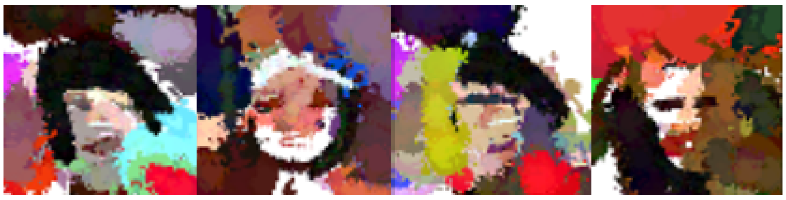


Figure 7.10.: **Stronger colors in agents trained with layer normalized discriminators.** This example shows outputs by an agent trained for 100k episodes using a logit-minimax discriminator with layer normalization.

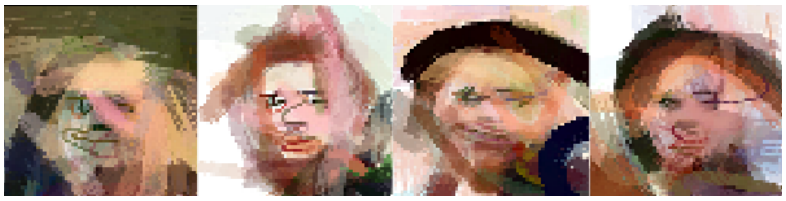


Figure 7.11.: **Muted colors in agents trained with unregularized discriminators.** This example shows a more natural use of color in an agent trained for 120k episodes using a logit-minimax discriminator with no layer normalization.

discriminator.

Spectral Normalization

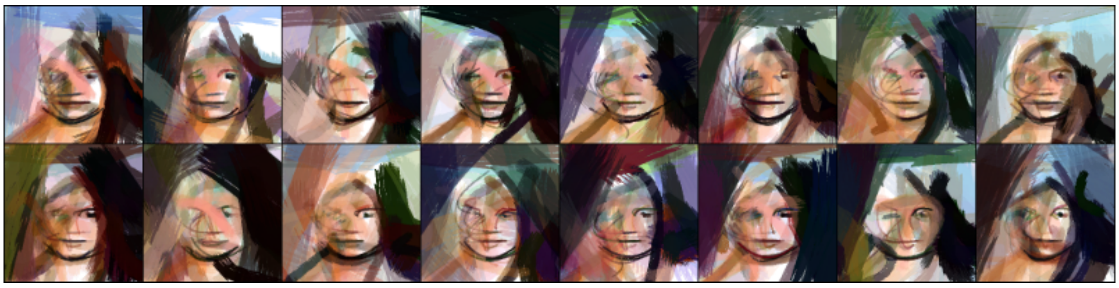
One of the improvements over SPIRAL introduced by Mellor et al. (2019) in SPIRAL++, was the inclusion of spectral normalization in the discriminator model. This type of normalization is applied to the weights of convolutional network layers directly, and ensures that the spectral norm[3] of the learned parameters is equal to one. Spectral normalization was first introduced by Miyato et al. (2018), who demonstrated that it works well in generative adversarial networks.

The findings of SPIRAL++ are supported by this thesis. The inclusion of spectral normalization in the discriminator leads to higher quality results for this method as well, though differences are not as pronounced as in previous work. An example of the effect of spectral normalization can be seen in Figure 7.12.

7.4.5. Comparison

The comparison between discriminators is based on a set of experiments conducted on the CelebA training dataset. Both minimax and WGAN-GP discriminators work well in the adversarial setting but have surprising effects on the learned policy. Generally, policies trained with minimax discriminators tend to favor small details rather than overall

[3]The spectral norm is equivalent to the largest singular value in the parameter weight matrix of a particular network layer. In PyTorch, spectral normalization is calculated using the power iteration method.


(a) Spectral normalization

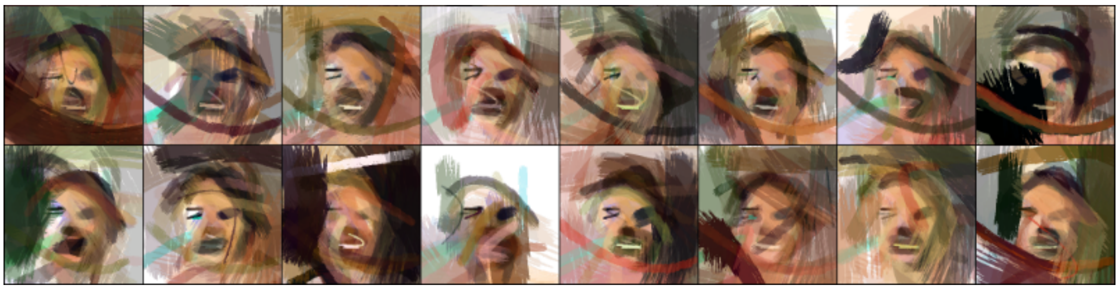

(b) Unregularized

Figure 7.12.: **Effect of spectral normalization.** Regularizing the discriminator using spectral normalization increases the quality of produced paintings. These examples show paintings generated by two identical policies trained for 100k episodes using a regularized and unregularized discriminator respectively.

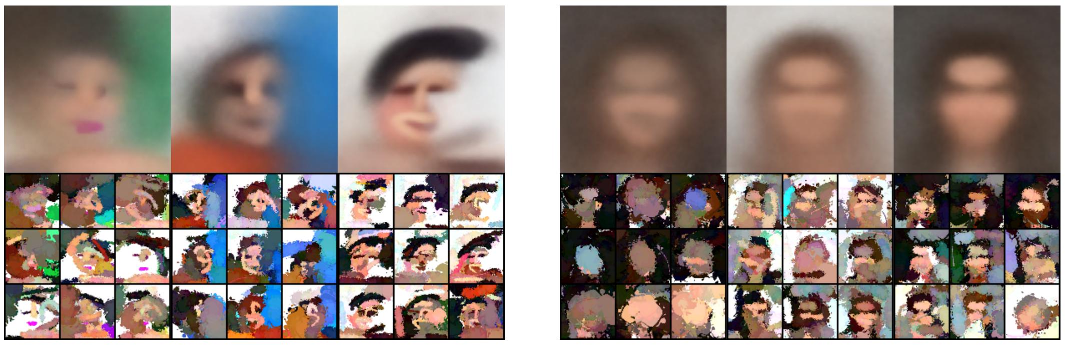

(a) Minimax (b) WGAN-GP

Figure 7.13.: **Comparison of minimax and WGAN-GP after 50k episodes of training.** Shows sample paintings (bottom) and mean outputs (top) for six identical agents trained with the logit minimax and WGAN-GP discriminators respectively. The choice of discriminator has clear effects on the produced images. The minimax discriminator enables the agent to learn small details such as mouths and eyes, but is less accurate in terms of overall structure. The WGAN-GP discriminator is more mean-seeking in its behavior, which results in paintings which lack smaller details but retain consistent color use and structure.

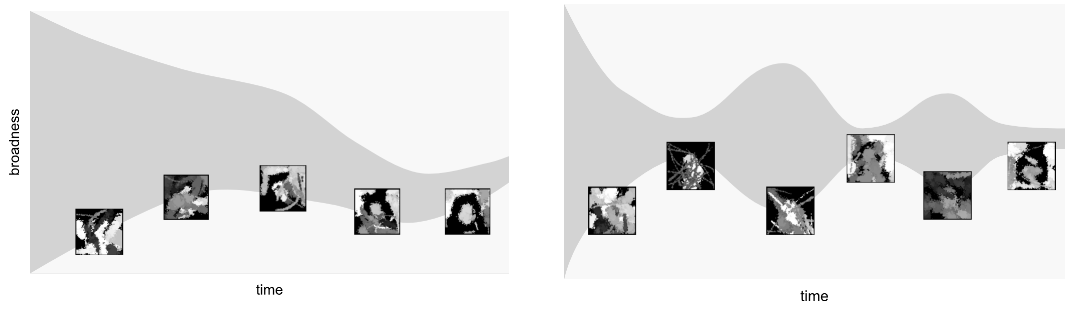
structure. Paintings by such agents typically contain easily recognizable features such as eyes and mouths, but their shapes and placements are not consistent with photo-realistic examples. This is shown in Figure 7.13a.

Policies trained with WGAN-GP reward systems favor overall structure more heavily at the cost of smaller details and features. Paintings by these agents usually contain clear facial shapes with indications of mouths and eyes at appropriate sizes and locations while lacking finer details. Figure 7.13b shows examples generated using the WGAN-GP discriminator.

As mentioned in Section 1.2, the goal of this type of adversarial painting is explicitly not to produce photo-realistic images, as is the case for many other generative methods. For this reason, this thesis considers the minimax variant the most appropriate technique, although both methods show learning ability. Refer to Section 8.1.2 for further discussions and qualitative analysis of produced paintings.

7.5. Exploration and Search

Exploration in the system is controlled by the entropy coefficient hyperparameter, which integrates directly into the agent's objective function as explained in Section 6.2. The choice of this parameter greatly affects the search properties of the system. Although factors such as the choice of return estimator and discriminator training algorithm also affect the explorative behavior, the entropy coefficient remains the most important



(a) Broad Search (b) Narrow (commit) Search

Figure 7.14.: **An illustration of the two extreme modes of search.**

hyperparameter to balance to achieve good results.

High entropy coefficients translate to a broad search in policy space. Since the entropy of the policy remains high, outputs are typically indistinguishable from randomly generated images for a large number of iterations. If the entropy coefficient is too high, the policy will fail to improve entirely. After long periods of training, the generated images may slowly start to improve. An illustration of this mode of search is show in Figure 7.14a.

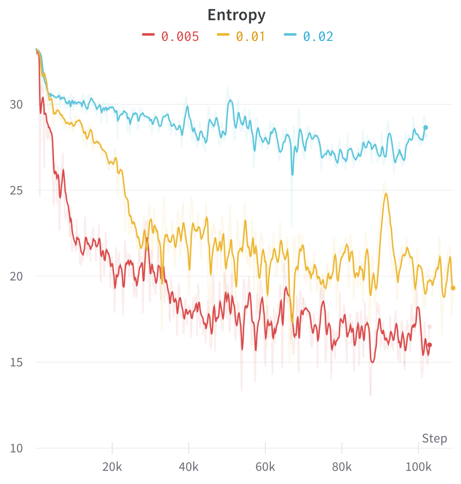
Low entropy coefficients translate to a more narrow search. The policy entropy will decrease much more rapidly, resulting in low variation in output paintings. Since the adversarial opponent changes continuously during training, suboptimal outputs will quickly be detected by the discriminator, yielding low reward. This effect leads to an oscillating mode of search. After several iterations of consistent (but suboptimal) outputs, the policy may start to produce seemingly random paintings again. Given this behavior, the search mode may also be thought of as a *commit-search*. An example of this mode is shown in Figure 7.14b.

The broad and narrow search modes describe the behavior on the extreme ends of the spectrum, of which neither are desirable in practice. In the adversarial context, the search mode implicitly affects the reward system as well:

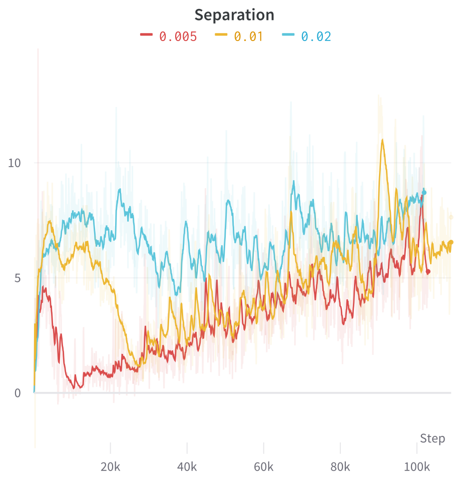
- When the search is very broad, the discriminator will typically outperform the policy, and very few examples will yield a high reward. The variation in output images is reflected directly in the reward system. The discriminator typically fails to learn high level features because the vast majority of fakes are easily detected without them.

- When the search is too narrow, the discriminator will overfit to the current mode of images, which causes large oscillations over time. In the worst case, the discriminator fails to generalize at all, and the system never converges. Oscillation is discussed in detail in Section 7.6.

For the proposed method, an entropy coefficient between $5e-3$ and $1e-2$ provides a good balance between exploration and exploitation. Figure 7.15 shows how entropy



(a) Entropy



(b) Separation

Figure 7.15.: **Search entropy and discriminator separation.** The entropy and separation varies significantly depending on the entropy coefficient hyperparameter.

and discriminator separation [4] varies for different entropy coefficients. As seen in the figure, values greater than $1e-2$ prevent the policy from converging to a sufficiently low entropy. As a result, discriminator separation remains high throughout training, and as shown in Figure 7.16, the outputs are comparatively worse.

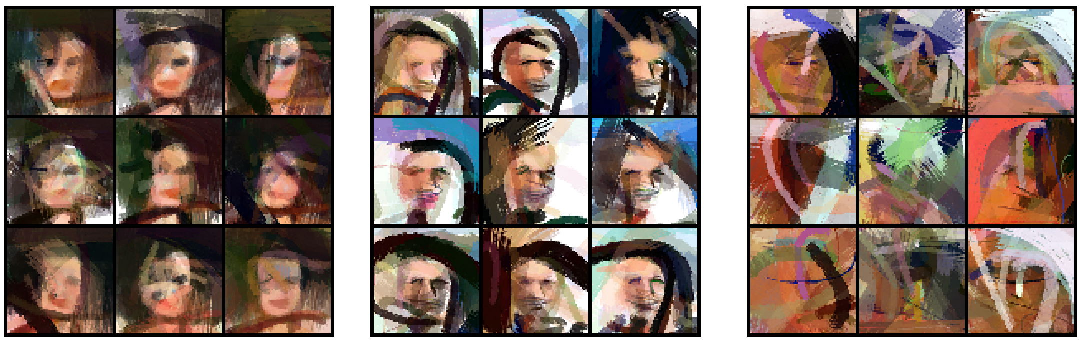
Although entropy regularization is a common exploration technique in PPO-based systems, more sophisticated search strategies such as Monte-Carlo Tree Search exist. These are briefly discussed in Section 9.1.

7.6. Policy Mode Oscillation

Mode collapse is a well known problem in generative adversarial networks, previously discussed by Thanh-Tung and Tran (2020) and many others. When mode collapse occurs, the generator consistently produces images that are almost identical. Although the generated data can be of high quality, the lack of variation is a problem. Improvements to GAN architectures are often motivated by this issue, and introduce changes intended to prevent or reduce its rate of occurence. The dynamics of policy-based generators is different, but the method proposed in this thesis suffers from a similar type of problem referred to as *mode oscillation.*

Mode oscillation entails a policy that is continuously switching between two or more suboptimal modes. Once the discriminator catches up with the current mode, the policy quickly changes mode again, causing the discriminator to perpetually chase the changes

[4]Discriminator separation is given by the mean difference in predictions for fake and real examples.



(a) 0.005 (b) 0.01 (c) 0.02

Figure 7.16.: **Example outputs for various entropy coefficients.** If the entropy coefficient is too high, the policy tends to converge too early and produces paintings of lower quality.

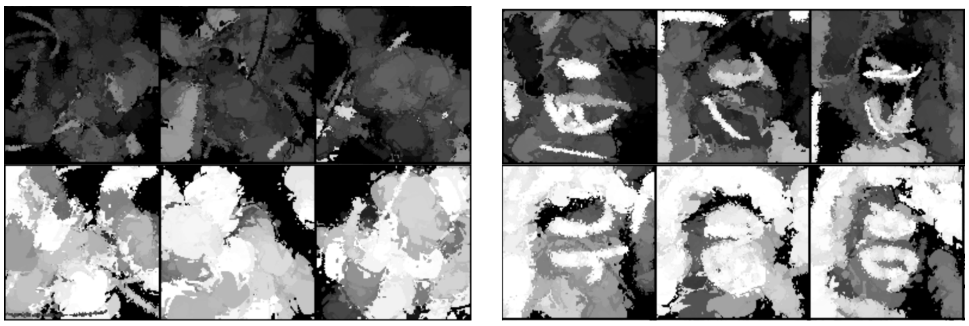


Figure 7.17.: **Mode oscillation early and later during training.** Each row shows a policy mode. For the CelebA dataset, the policy typically oscillates between dark and light backgrounds without proper discriminator regularization.

in the output distribution. Since the agent oscillates between different modes, it is usually unable to improve the painting quality of each individual mode beyond a certain point. In a majority of cases, the policy entropy converges at a value too great to produce quality results consistently.

In the beginning of training, observed modes are almost always very bright and very dark images. After further training, the model typically oscillates between dark and light backgrounds for grayscale paintings trained on the CelebA dataset. After long training sessions, modes can even be recognizable traits, such as the painted hair color. Figure 7.17 shows an example of observed modes in a policy trained on the CelebA dataset.

This type of failure is evident from discriminator predictions which oscillate at a predictable frequency (see Figure 7.18), in addition to being observable in the generated outputs. The following sections present insights collected from empirical studies intended to discover the causes of mode oscillation and reduce its adverse effects.

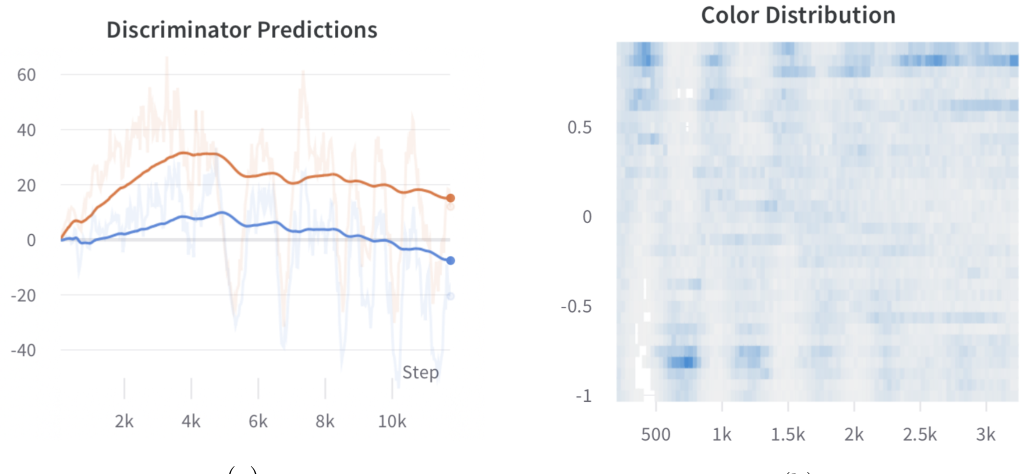


Figure 7.18.: **Mode oscillation detectable in discriminator predictions and action distributions.** a) Real (orange) and fake (blue) discriminator predictions. Policy mode oscillations are visible as periodic oscillations in discriminator outputs. b) Early oscillations are apparent in the distribution of action color (this example shows a single grayscale channel). Once modes become more complex later in training, such patterns in actions distributions are difficult to detect.

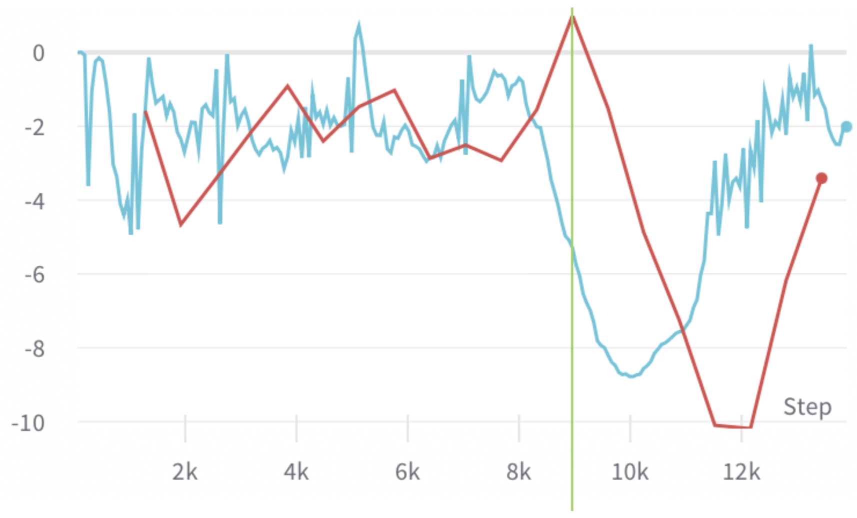


Figure 7.19.: **Catastrophic forgetting in the discriminator.** The graph shows WGAN-GP predictions on the most recent paintings (blue) and a batch of 1000-episode old paintings (red). The point of catastrophic forgetting is indicated by the green line. At this point in time, old paintings are evaluate much higher despite recent improvements in the policy.

7.6.1. Causes

Discovering the underlying cause of mode oscillation is challenging due to the intertwined dependencies between the policy, training algorithms, and reward system. Observations indicate that the causes are related to two primary issues, which in combination induce oscillation:

- **Lack of policy multi-modality.** Although the policy is capable of representing policies with high variation, producing a wide array of different outputs, this does not always occur in practice. Paintings generated by a single policy are often very similar, particularly with respect to the color palette. Variation is caused by the stochasticity of the agent and environment, rather than being explicitly represented by the policy model. This can be seen in Figures 7.12 and 7.17.

- **Catastrophic forgetting and overfitting of recent examples in the discriminator.** As a result of low variation in policy outputs, the discriminator model overfits to recent examples. This, in turn, causes catastrophic forgetting of older paintings, even if the quality of the new paintings is comparatively better. An example of catastrophic forgetting is shown in Figure 7.19.

Mode oscillation is closely related to exploitation in the policy. If the entropy coefficient is sufficiently high, this type of oscillation does not occur, but the policy is also unable to improve. Given an appropriate rate of exploration, mode oscillation is a natural consequence of the multi-modal behavior of the policy, as well as a lack of long term memory of old examples in the discriminator model.

Recall that the autoregressive policy head is a large model capable of representing a complex multi-modal function. The consistency of individual modes and periodicity of oscillations indicate that the policy is in fact representing two or more (essentially) separate policies simultaneously — although only one mode is active at any given time. Changing a single network bias (for instance causing the color of the first stroke to be black instead of white) is theoretically sufficient to change the entire trajectory of the policy to a different mode without forgetting state-action mappings of older trajectories.

7.6.2. Prevention

Several ways of preventing policy mode oscillation are explored, including changes in the architecture of the discriminator and modifications to the learning algorithm. Some of these techniques demonstrably reduce the prevalance of the problem, though this thesis has not discovered a way to prevent the issue entirely.

Discriminator

Dropout, first introduced by Srivastava et al. (2014), is a type of neural network layer intended to prevent overfitting and increase the generalization ability of deep models. These layers randomly suppress some portion of activations during training, encouraging the model to learn a broader set of features rather than relying on a smaller number of strong predictors. Dropout is included in all hidden layers in the discriminator, with the goal of preventing catastrophic forgetting. The introduction of dropout layers unfortunately does not reduce the prevelance of mode oscillation for this method.

As discussed previously, LSGAN by Mao et al. (2016) is an extension on the original GAN algorithm intended to reduce the problem of vanishing gradients. Motivated by ideas from LSGAN and PPO, an additional loss metric is introduced as a regularizer in this context. The modification consists of a squared-error-loss encouraging the discriminator predictions to remain close to the previous distribution, and hence prevent overfitting to recent modes. This loss is comparable to the surrogate objective in PPO, and is intended to prevent the discriminator from changing too rapidly. The change reduces mode oscillation in some situations, but is not usable in practice as it causes severe instability issues in the policy.

Training Data Buffering

Training data buffering entails managing a buffer of fake images that is uniformly sampled from to train the discriminator. Given a sufficiently large buffer, training data will necessarily be more varied and cover several policy modes. Ganin et al. (2018) used a buffering system in their implementation, but its inclusion was motivated by the training setup rather than mode oscillation issues. The buffer was later removed in the improved implementation by Mellor et al. (2019).

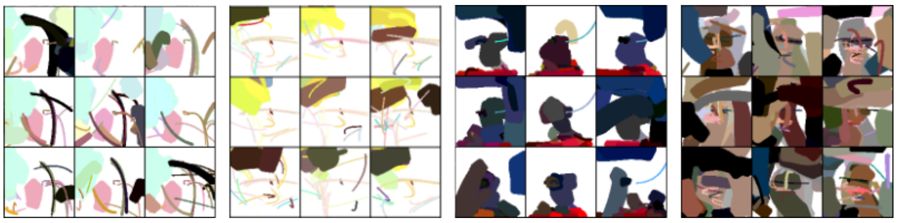


Figure 7.20.: **Bad local optimas of different severities.** Several factors such as training data buffering or too small exploration coefficients may cause the policy to get stuck in bad local optimas. When this occurs, the entropy is too low to enable new discoveries. As a result, the agent will struggle to improve.

The introduction of a discriminator buffer reduces the frequency of mode oscillation for the proposed method. By training the discriminator on a set of random paintings from older policies, the system explicitly prevents the discriminator from forgetting older variants, and indirectly forces the policy to improve the general quality of images rather than tricking the discriminator through oscillating targets. This work found that a buffer size of 1500 images works well in practice. Note that this size is greater than the typical oscillation frequency of around 400 episodes, which is crucial to have any regularizing effect.

Unfortunately, the reuse of fake examples inevitably leads to a smoothing of the fake distribution. In practice, this causes policies to converge to bad local optima quicker and earlier in training without an increase to the exploration entropy coefficient. An example of this is shown in Figure 7.20. Given sufficiently broad exploration this can be prevented, but training is slower and less sample efficient as a result. Mode oscillation can still occur with this modification, but the frequency covers longer periods of training.

Population-based Training

Population-based training entails simultaneous training of several agent policies sharing the same discriminator. Similarly to data buffering, this type of training increases the variation in fake examples available to the discriminator. This is benefitial in terms of model generalization, but also avoids the distribution smoothing caused by buffering. Population-based training was previously used by Mellor et al. (2019) in SPIRAL++.

This thesis only considers small populations of 2–4 agents due to the high computational cost associated with training and simulating multiple policies. The population-based training procedure is identical to the single-policy implementation discussed in Chapter 6, with the exception of parallel rollouts and policy training facilitating multiple agents.

Population-based training does not prevent oscillation to the same extent as data buffering. In the majority of cases, a single policy greatly outperforms the remaining population, leading to a severe imbalance in the quality of fake examples. This effect can be seen in Figure 7.21. Although the best policy performs comparably to the single-population setting, the remaining policies fail to converge to an acceptable level entirely.

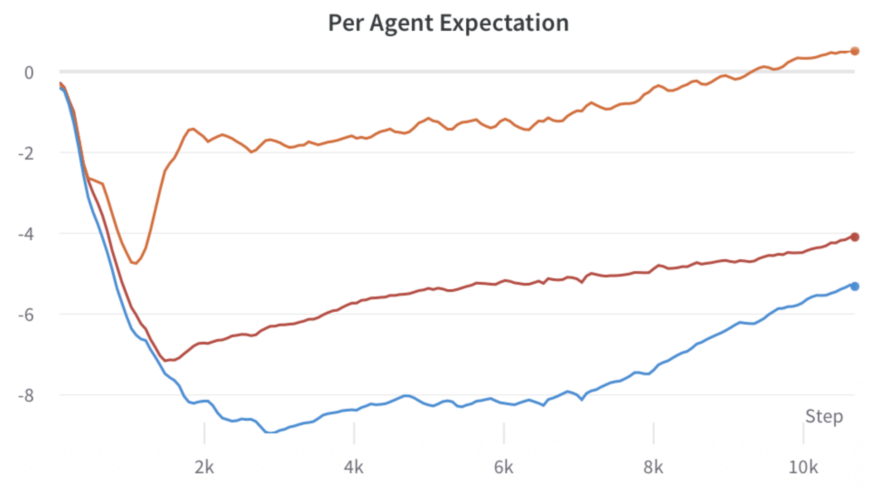


Figure 7.21.: **Multi agent performance expectation.** The graph shows the performance expectation, as measured by the discriminator, for three different agents in a population. A single policy outperforms the others, which prevents the poorly performing agents from converging to good solutions.

The poor quality fakes generated by these agents do little in terms of reducing oscillations in the best policy.

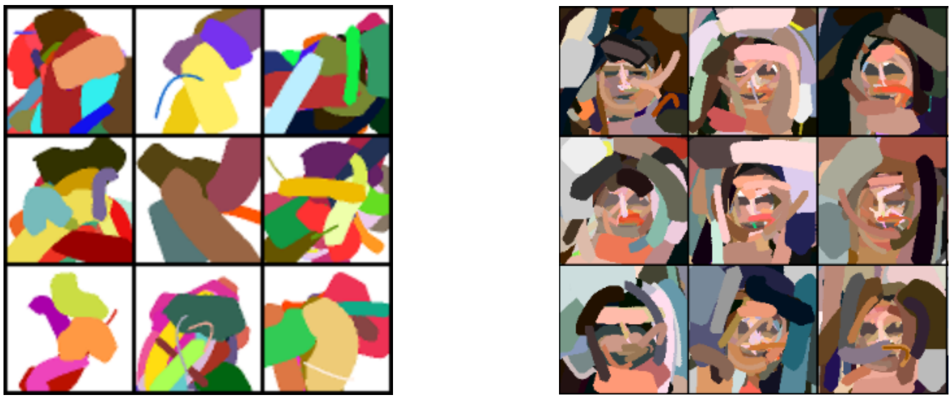
These observations suggest that the reward system fails to provide useful reward signals to the worst performing agents in the population. To enable better results, some mechanism is necessary to properly balance the policies in the population. Prioritizing policy training or discriminator data using a suitable performance metric are examples of potential solutions, though this thesis does not explore such alternatives.

7.7. Reward System

There is a large difference in performance between the temporal and episodic reward system. The episodic reward system works on simple datasets with short episodes (fewer than 5-10 steps), but is unable to learn useful policies in more difficult settings. As discussed in Section 6.4, the temporal reward system is motivated by the challenge of credit assignment in long episodes.This system is strictly necessary to learn complex policies trained with CelebA. All the best results achieved with the proposed method use the temporal reward system exclusively. A comparison of outputs for the different reward systems is shown in Figure 7.22.

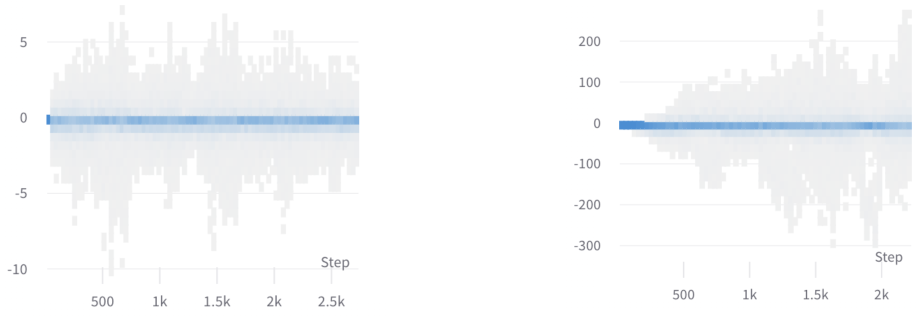
Expectation Adjustment

When using the Wasserstein discriminator formulation, fake predictions increase in variance over time. The unjusted reward system consequently yields a similar pattern in rewards, and the policy fails to converge with this setup.



(a) Episodic (b) Temporal

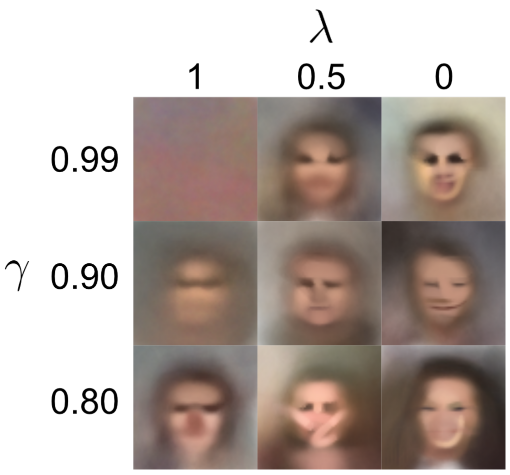
Figure 7.22.: **Difference between agents trained using episodic and temporal rewards.** The temporal reward system enables agents to learn good policies when trained on CelebA. Convergence does not occur in agents trained with episodic rewards on this dataset.



(a) Adjusted Temporal (b) Regular Temporal

Figure 7.23.: **Adjusted and unadjusted reward distribution.** The figures show the difference in reward distribution for temporal and adjusted temporal reward systems for a WGAN-based reward system. The regular temporal rewards leads to large variance over time, which prevents effective learning in the policy.

Improvements associated with the gradient penalty regularized discriminator (WGAN-GP) and the expectation adjusted temporal reward system enable consistent convergence in the system. Figure 7.23 shows an example of reward distributions for the temporal and adjusted temporal reward systems. Differences are less pronounced in the logit-minimax variant, but expectation adjustment improves the overall stability of the system in this setting as well.



Estimated Policy Entropy

γ \ λ	1	0.5	0
0.99	31.3	25.3	20.7
0.9	27.6	22.8	22.8
0.8	25.8	22.3	**18.6**

Figure 7.24.: **Effect of advantage estimate bias and discount factor.** The figure shows mean outputs of agents trained for 10k 256 step episodes using the minimax logit reward system.

7.8. Advantage Estimation

This section explores the role of advantage estimators in the reinforcement learning algorithm. As discussed in Section 6.3, there are three main variants used in the field. Since Generalized Advantage Estimation (GAE) can generalize to both TD-learning and Monte Carlo (returns-to-go) sampling, it is the only type considered here.

The variance of the estimator has large effects on the exploration of the system, second only to the entropy coefficient. Figure 7.24 shows the mean outputs and resulting entropies for different values of γ and λ for nine different agents trained on CelebA. For the proposed method, lambdas lower than 0.5 and discount factors between 0.9 - 0.99 work best on this dataset.

High values of lambda prevent the system from converging using the standard configuration. This effect is exacerbated by the variance in rewards caused by function approximation inaccuracies in the discriminator model. In order for the system to converge using direct return samples, both the entropy coefficient and return discount must be reduced.

Low values of lambda converge most quickly. In rare cases the algorithm can achieve impressive results with comparatively few samples, but the estimator is more varied in its performance. Due to the presence of high bias, the policy is prone to bad local optimas. This type of estimator also suffers more from mode oscillation than lower-bias alternatives.

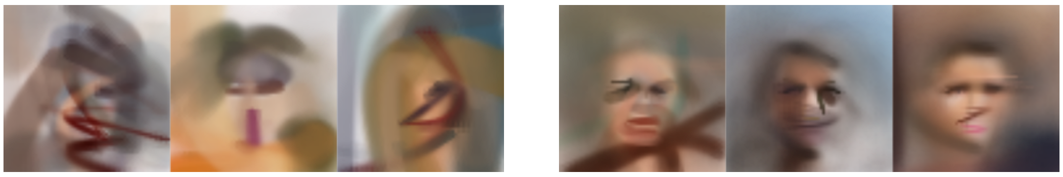
7.9. Recurrency

Both Ganin et al. (2018) and Mellor et al. (2019) use recurrent policies with LSTM cells to generate paintings sequentially. This section further explores the importance of recurrence with respect to output quality of trained policies.

The recurrent policy architecture used here employs a structure similar to SPIRAL. Following the policy feature extractor, a single LSTM cell processes the feature vector to produce a recurrent vector used by the policy head. Refer to Section 5.4 for details about the policy architecture.

In order to train a recurrent policy using PPO, the algorithm employs truncated backpropagation through time (TBPTT). This technique has previously been used for reinforcement learning in work by Mnih et al. (2016) and Heess et al. (2017). In the TBPTT setting, each training sample in a given minibatch contains a fixed length sequence of steps that is optimized simultaneously to propagate gradients through time. Sequences are masked based on the episode length to enable sequences to start at any point in time. This thesis considers sequence lengths of 4, 8 and 16. These values are chosen to discover the importance of sequence length in learning temporal dependencies, but remain limited to a maximum of 16 steps due to high computational cost.

The recurrent algorithm is less sample efficient than the non-recurrent implementation, and converges between 2-3 times slower than the default method for all sequence lengths tested. Although the algorithm is capable of producing paintings of comparable quality, no improvements are observed. Figure 7.25 shows differences in mean outputs for three different recurrent and non-recurrent policies.



(a) Recurrent Policy (b) Standard Policy

Figure 7.25.: **Recurrent and non-recurrent policies.** Mean outputs after 100k episodes for the recurrent and default policy. Although the recurrent policy can achieve comparable results to the default implementation, it is less sample efficient and more computationally expensive.

These results indicate that recurrence is not a necessary component in this type of system, and that other design decisions (such as the discriminator and reward system) are more important in producing high-quality results. This work is the first time non-recurrent policies have been shown to work for painting agents trained with generative adversarial reinforcement learning.

8. Evaluation and Discussion

The following chapter presents results and discussions in terms of the output, performance and behavior of the method. The chapter focuses on the most challenging dataset tested (CelebA), but also explores how suitable the method is for simpler problems such as Fashion-MNIST. The chapter is centered around the questions posed in Section 1.2, which entails the results of the method as well as the role of on-policy training, adversarial systems and neural architectures.

8.1. Qualitative Results

8.1.1. Evaluation Metrics

In the context of generative methods, common metrics of evaluation include the Inception Score by Salimans et al. (2016), and later the Frechet Inception Distance by Heusel et al. (2017). Although these metrics enable an objective metric for evaluation of generative methods, they have issues in this particular context.

The Inception Score (IS) evaluates image quality using the fake image predictions of a large classifier model. The Frechet Inception Distance (FID) is currently a standard metric for evaluating GANs. Unlike the inception distance, FID uses both the training dataset and fake examples to produce an objective score. The metric is based on the difference in low level activations in the classifier network. The classifier used for these metrics is trained on real images, and may therefore not capture the value of novel abstractions present in painted images. The intent of this type of painting system is explicitly not to produce photo-realistic examples.

Work by Brendel and Bethge (2019) further suggests that CNNs are highly sensitive to low level features, such as textures. This is particularly problematic for images generated sequentially using predefined brushes, as the policy has little to no control over such image properties.

Due to these challenges, IS and FID are unsuitable in this context, and manual evaluation and comparison is used to determine the level of visual quality. Note that this type of evaluation is prone to bias, and reduces the number of images that can be examined in reasonable time.

8.1.2. Quality and Variation

The system is capable of producing easily recognizable paintings of various objects, even when trained on complex datasets such as CelebA. Trained policies show a high level of variation, and are able to produce a large amount of different paintings. Although mode

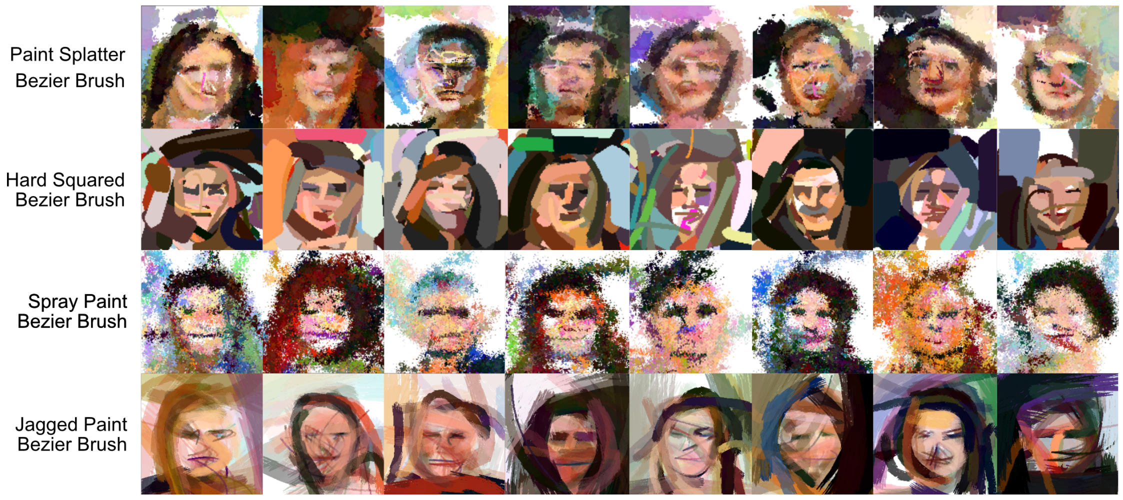

Paint Splatter Bezier Brush
Hard Squared Bezier Brush
Spray Paint Bezier Brush
Jagged Paint Bezier Brush

Figure 8.1.: **Example outputs for different brushes generated by agents trained on CelebA.** The system is general enough to learn the dynamics of a wide range of different environments.

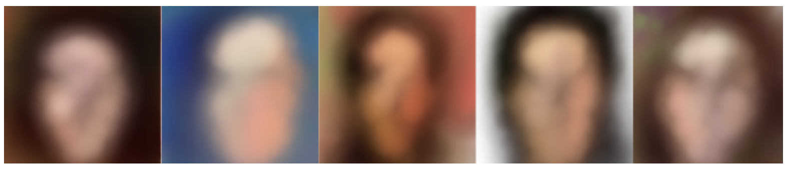


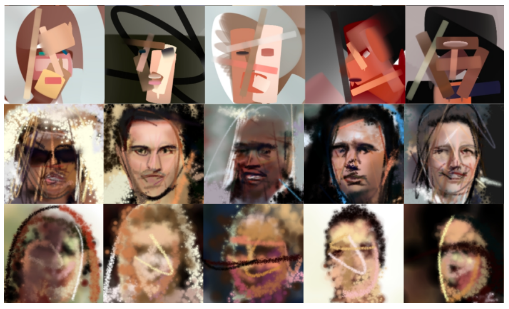
Figure 8.2.: **Example outputs for SPIRAL.** Example paintings from conditional SPIRAL agents trained on CelebA. Results by Ganin et al. (2018), reused with permission.

oscillation, as discussed in Section 7.6, remains a challenge, the trained policies show an ability to capture a broad range of the dataset distribution.

The appearance of the brush stroke is the primary way in which environment dynamics vary in the context of sequential painting. The system shows a high level of generality to very different brush types, and is capable of learning the specifics of each environment sufficiently well to produce good results. The discriminator is general in the same manner. Although the appearance of fake images varies greatly depending on the environment, the discriminator learning system remains capable of producing rewards adequate for policy training. Figure 8.1 shows sample results for the method using a set of different environment brushes.

The results of this method exceed the work by Ganin et al. (2018). As shown in Figure 8.2, SPIRAL produces blurry paintings with low levels of detail, whereas the proposed method is not limited in this way. Note that the included SPIRAL examples are generated by conditional agents, as the authors did not present outputs for uncoditional agents on this dataset.

Improvements introduced by Mellor et al. (2019) in SPIRAL++ are more comparable,


(a) SPIRAL++

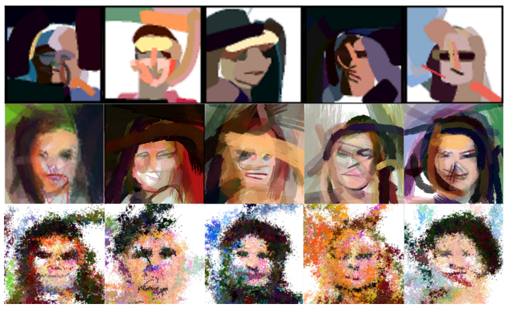

(b) This Method

Figure 8.3.: **Comparison with SPIRAL++.** Shows example paintings for SPIRAL++ and the proposed method. Each row shows a different brush type with similar appearances. SPIRAL++ examples by Mellor et al. (2019), reused with permission.

but the proposed method is less consistent in its use of small strokes for low level details. As a result, SPIRAL++ is capable of producing paintings of higher realism. Note that direct comparison between methods is difficult due to differences in the environments the agents interact with. SPIRAL++ uses a more advanced rendering engine, which in turn increases the expressiveness of interactions available to the agent.

Figure 8.3 shows a comparison between the methods for a set of brushes with similar appearances. This thesis considers the paintings by SPIRAL++ better overall, but results vary for different agents and environments. As previously indicated, evaluation of specific paintings also depends heavily on the personal preferences of the observer.

8.1.3. Abstraction

Under additional constraints, the challenge of producing paintings matching the training distribution increases. The system shows an ability to overcome these challenges, and can produce artistic simplifications of complex features using very few brush strokes. An example of such abstraction can be seen in Figure 8.4.

Since the dataset consists of photos exclusively, these results demonstrate that the method is capable of abstraction even under heavy constraints. For brushes with lower expressivity, such as the hard squared bezier brush, generating photo-realistic paintings is in fact impossible. The trained agents are able to overcome this, and can generate small details (such as eyes and mouths) using simplifications recognizable by humans.

These results mirror the findings of SPIRAL++, which shows a similar ability to learn feature abstractions (see first row in Figure 8.3). Abstraction is an important property in this type of system, and shows that the learning system is capable of sophisticated generalization in difficult adversarial settings. The behavior and generalizability of the learning system is discussed further in Section 8.2.

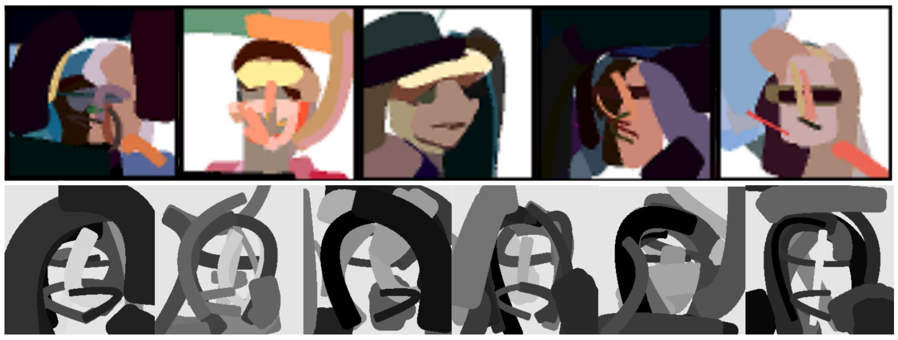


Figure 8.4.: **Example of abstraction under heavy constraints.** The system is able to produce interesting abstractions of complex features in the dataset, even for short episodes using non-realistic brushes. The figure shows outputs by agents trained on the CelebA dataset in color and grayscale, respectively.

8.1.4. Datasets

The system shows a high level of independence with respect to the training dataset. The policies and discriminators are able to learn from very different examples (MNIST, Fashion-MNIST and CelebA) using a range of different brushes and action parameterizations.

Figures 8.1 and 8.4 shows examples using different brushes painted by agents trained on CelebA. Although this is a complex dataset, the modes are generally quite similar. Future work could explore the performance of this method on even more challenging multimodal datasets. Figure 8.5 shows examples of shoes painted by the system after training on the Fashion-MNIST dataset.

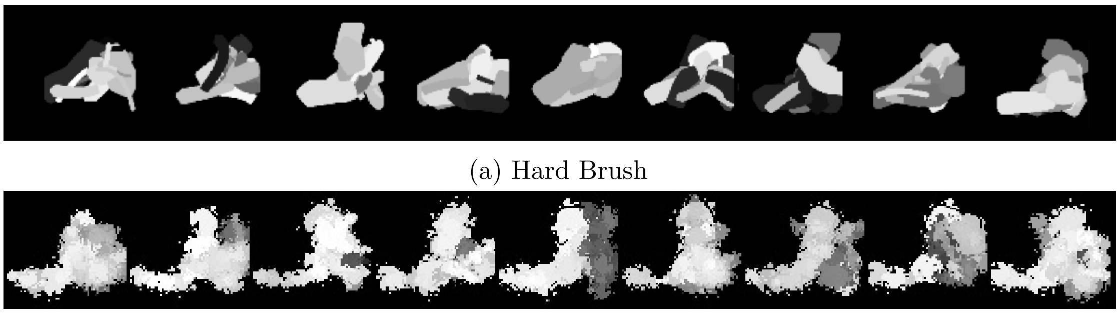
Although the mehod works for both datasets, the amount of training necessary is increased for more complex domains. Whereas the system can produce good outputs for Fashion-MNIST after only 7k episodes of training (Figure 8.5), agents using CelebA require more training. This is shown in Figure 8.6. After 7k episodes of training, paintings of CelebA are not yet recognizable. The agents require about 10 times the amount of training to produce comparable results.

8.2. Algorithmic Behavior

8.2.1. Policy Training and Exploration

This thesis poses the question of how to effectively apply on-policy reinforcement learning in the generative adversarial setting. Although Proximal Policy Optimization has been used successfully on difficult continuous control problems in the past, this work is the first time it has been demonstrated to work for painting agents in the generative adversarial setting. Learning in such settings entails unique additional challenges, and the PPO-based method proves capable of overcoming these.

The entropy based exploration technique in PPO is sufficient to learn the dynamics of both the reward system and environment. Recall that the size of both the action and


(a) Hard Brush

(b) Splatter Brush

Figure 8.5.: **Examples of painted shoes.** The system is able to learn from signficantly different datasets. These examples show paintings by agents trained on the shoe class from Fashion-MNIST using a hard squared brush and a paint splatter brush, respectively.

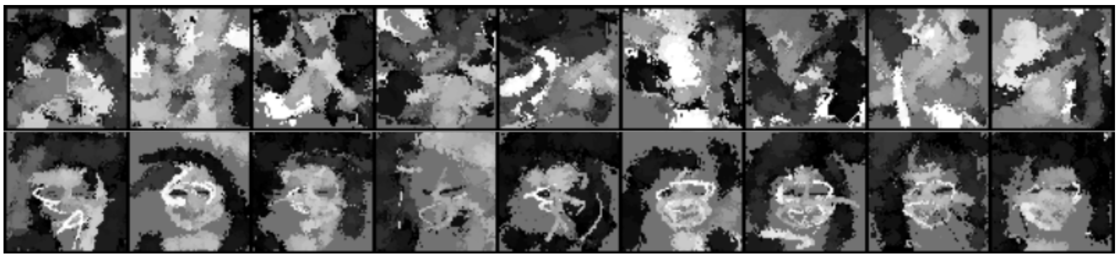


Figure 8.6.: **Training time for CelebA.** Whereas agents trained on Fashion-MNIST can produce good results after only 7k episodes, the CelebA dataset is more challenging. This example shows examples of grayscale paintings after 7k episodes (top row) and 70k episodes (bottom row) of training.

observation space in this type of environment is very large. The learning algorithm enables the policy to generalize across the space, despite it being computationally infeasible to thoroughly explore. As previously discussed, this type of exploration can still fail in certain situations, leading the policy into various bad local optima. Examples of suboptimal policies were previously presented in Figure 7.20.

As shown in Section 7.5, balancing the rate of exploration remains one of the most important aspects of the method. Although a broad range of entropy coefficients enable some level of learning, the quality of final outputs is highly sensitive to small changes in the entropy coefficient. Since the explorative behavior is also strongly affected by the choice of return estimator, achieving good results necessitates multiple trials across hyperparameter ranges. In practice, applying the method to new datasets or new environment dynamics is therefore not a straight forward process, but requires a fair amount of algorithmic tuning. This is not an unique problem in the field, but the combination of reinforcement and adversarial learning increases the difficulty of finding appropriate hyperparameters to balance the system for new problems.

8.2.2. Essential Components

Previous work by Ganin et al. (2018) and Mellor et al. (2019) include recurrent policies and population-based training. This thesis shows that these components are not essential to achieve good results in a PPO-based system.

The results of this method does not improve with the inclusion of population-based training due to a single policy outperforming others. Although populations have clear benefits in terms of discriminator training (see Section 7.6.2), it is not necessary in this context. Improving the behavior of PPO-based populations remains a potential venue of better results in the future. If the performance of the whole population is consistent throughout training, the discriminator could learn to generalize better, thus yielding higher quality rewards. As indicated by Mellor et al. (2019), agent populations increase the capabiliy of the generator element in the adversarial setting. It remains unclear whether a single PPO-based agent results in the best generator-discriminator balance for this method, or if the challenges associated with population-based training is caused by other factors.

As shown in Section 8.1.2, the proposed method is capable of learning high-quality policies with no reccurency in single agent settings. Non-recurrent policies have immediate benefits over recurrent variants developed previously. First, they are less computationally expensive to train. This eases the task of applying the method to new types of problems with respect to iteration time and algorithmic tuning. Second, the policy is not temporally dependent, which enables additional applications. A well trained policy could, for instance, be used to finish an incomplete human painting, or be integrated with painting software that is incompatible with the learned action space. A recurrent policy could be necessary for certain problems, but as shown in the previous chapter the method works in the recurrent setting as well.

8.2.3. Adversarial Learning

A large portion of this thesis is dedicated to the role of adversarial learning in the system. In addition to the design of the reward system itself, the training procedure of the adversarial opponent is crucially important to enable learning in complex environments.

Given the size of the observation space in the typical generative setting, the credit assignment problem is a significant challenge to overcome. This work suggests that the most important property to consider is the temporal distribution of rewards. As shown in Section 7.7, episodic reward systems are simply not sufficient to train good policies with long action sequences.

The distribution of temporal rewards is greatly affected by the training algorithm used for the discriminator. Section 7.4 shows that both the Wasserstein- and minimax loss formulations are compatible, whereas LSGAN performs poorly in this setting. Although the fact that multiple variants work is promising with respect to the generality of the method, it is difficult to conclude why certain techniques work or behave differently than others. The mean-seeking behavior of WGAN-GP implementations is a good example of this. Given the common objective of discriminating between fake and real examples, it remains unclear why this discriminator guides policy learning in such a different way compared to other solutions.

As mentioned in the previous chapter, temporal rewards in the standard implementation are based on paintings that are *explicitly* not a part of the discriminator training data. The learning required to solve this problem is not equivalent to generalization from a dataset to real world examples in typical classifier models. The model must instead generalize to incomplete paintings generated by an unobserved temporal painting process. This is a niche type of generalization that has not been investigated in previous literature.

Section 1.2 poses the question of which discriminator methods are suitable for generative adversarial reinforcement learning, and this thesis has demonstrated that at least two alternatives are applicable. Explaining why discriminators behave differently in this particular way remains outside the scope of the thesis, and would likely require significant effort and analysis of several models to resolve. This work still serves to highlight areas of research that could lead to useful insights, including topics such as adversarial reward shaping, the role of intermediate canvas training, and discriminator regularization.

8.3. Neural Network Architectures

Chapter 5 presents the neural network architecture of the discriminator and policy. The work in this thesis shows that the architectures are capable of challenging generative adversarial reinforcement learning, and exposes several interesting modes of behavior.

8.3.1. Discriminator

As discussed in Section 5.1, the proposed method uses a conventional convolutional neural network discriminator model. Such models have previously been used successfully for many types of adversarial learning. This work provides further evidence that these models are highly general, and are suitable in this type of reinforcement learning system as well. This observation supports the findings of Ganin et al. (2018) and Mellor et al. (2019) with respect to the performance of standard discriminator architectures.

Due to initial success using a conventional discriminator model, alernate architectures are not explored in this thesis. As discussed in Section 7.4, the system behavior is greatly affected by the learning algorithm of the discriminator, and this observation has guided the priority of the research towards discriminator training as opposed to model architecture. Results presented in Section 8.1.2 indicate that conventional architectures are more than sufficient in this context. Although the architecture works well in practice, other designs could have advantages that this thesis has not found.

8.3.2. Policy

The most complex neural network architecture presented in this thesis is the policy architecture. Recall that the policy consists of two primary parts, as presented in Section 5.4. A deep residual convolutional network first extracts features from the current canvas. The extractor is followed by a large autoregressive head that processes features to produce a probability distribution over the action space in the environment.

The feature extractor is a conventional deep residual network. Based on the performance of these models in previous literature, alternatives are not explored in this thesis. As shown by He et al. (2016), deep residual networks are capable of learning more complex functions than what is necessary in this context. Residual models are also appropriate with respect to gradient flow, given the depth of the following policy head. Although the work proves that the model works in practice, simpler models could have advantages due to smaller hypothesis spaces that are easier to explore.

Experiments demonstrate that the proposed autoregressive policy is capable of representing large, complex action spaces. The results support the findings of Ganin et al. (2018) and Mellor et al. (2019), which employ similar autoregressive models. Typical policy variants in reinforcement learning, including Gaussian and categorical policies, are shown to be insufficient for more complex datasets such as CelebA. These policy alternatives are not considered for generative adversarial reinforcement learning in previous work.

As discussed in Section 5.4, the design of the autoregressive architecture is partially motivated by a set of intuitions relating to information flow in the system. The sample and feature embeddings between each action head is one example of this. In order to represent the conditional action distribution, information relating to previous samples and observed features is strictly necessary to provide. The residual design used in this work is, however, only one way to incorporate such information.

Designing and developing a working policy involves substantial work, and exploring alternate designs therefore remains outside the scope of this thesis. The thesis has demonstrated one example of a working policy, and reasoned about its various components and their role in light of the information processing necessary. Additional research is necessary to quantify their relative importance when applied to various problems in practice.

9. Conclusion

This thesis has demonstrated the successful use of Proximal Policy Optimization (PPO) in generative adversarial reinforcement learning. The work includes the development of a customizable painting environment suitable for reinforcement learning, designs for a set of neural network architectures, and experiments exploring essential components in this type of learning system. Agents trained using the proposed method are capable of producing results comparable to the state-of-the-art, and are general with respect to both environment dynamics and training datasets.

The work has shown that temporal rewards are necessary to enable learning in complex settings, and that the quality of such rewards is greatly influenced by the training procedure of the adversarial opponent. Results demonstrate that both WGAN-GP and logit-minimax discriminators are applicable in this context, whereas LSGAN produces rewards inadequate for learning. In addition to the discriminator learning algorithm, several regularization techniques are also explored. The work has shown that these factors implicitly change the behavior of temporal rewards, and as a result may guide policy learning in different ways.

Exploration is a widely discussed topic in reinforcement learning, and its role remains important in this context as well. The PPO-based system uses the entropy exploration technique, and the entropy coefficient remains one of the most important hyperparameters to tune to achieve good results. Combined with Generalized Advantage Estimates, the technique enables the agent to explore a very large and complex environment sufficiently to produce high-quality paintings.

The thesis further shows that typical policy architectures in reinforcement learning are insufficient for this type of adversarial learning. Results demonstrate that the proposed policy, a large autoregressive model, is capable of representing these complex action spaces. The method enables model learning in both the recurrent and non-recurrent setting, which extends the range of possible applications beyond previous work. In addition, the thesis provides further evidence of the capability of conventional discriminator models. These models have shown great success in previous work, and the results of this thesis confirm that they are suitable in this context as well.

Although the method works well in several difficult settings, this type of learning is also associated with certain weaknesses and unique challenges. Mode collapse is a widely discussed topic in generative adversarial networks, and a variant of this issue can manifest in policy-based generators as well. Experiments suggest that the issue is rooted in both the adversarial relationship and modeling capability of the agent policy. Methods such as data buffering and population-based training are suggested as potentional solutions, but additional research is necessary to discover how the problem can be effectively prevented.

9.1. Future Work

Generative adversarial reinforcement learning is an exciting area of research that is not widely explored. Although this work exposes many properties of such systems, a lot of potential remains in the area. Following sections suggest various topics of future work that could be particularly interesting to explore.

9.1.1. Algorithm

Including this work, PPO and A2C are the only algorithms investigated in this context so far — both of which are on-policy techniques. Although Proximal Policy Optimization has shown great success in the field, other reinforcement learning algorithms may have advantages in the generative setting. State-of-the-art off-policy algorithms such as Deep Deterministic Policy Gradients (DDPG) by Lillicrap et al. (2015), Twin Delayed DDPG (TD3) by Fujimoto et al. (2018), and Soft Actor Critics (SAC) by Haarnoja et al. (2018) have been shown to perform comparably to PPO, and outperform it for certain problems. Off-policy algorithms have benefits in terms of sample efficiency, and these alternatives could therefore be well suited for the large observation space in generative environments. SAC is of particular interest due to its ability to learn stochastic policies. Stochastic policies are more desirable in this setting because of their inherent variation in generated outputs.

9.1.2. Exploration Techniques

The proposed method uses entropy based exploration to explore the environment. Given the size of the environment, more sophisticated search strategies could enable the algorithm to learn quicker and discover better policies. Monte Carlo Tree Search (MCTS) is a particularly interesting technique that employs heuristics to prioritize exploration of promising trajectories in the environment. MCTS is most commonly used to solve board games, but is applicable in many different settings. In earlier work, Silver et al. (2016) combined MCTS and deep learning to train an agent to play Go at a superhuman level. Although this problem is different at first glance, it has many similarities to the generative painting setting. In addition to the large action and observation spaces, the results of choices are typically not evident until many steps in the future, and rewards are inherently sparse. Considering previous successes of MCTS, this type of exploration could improve results and perhaps enable the removal of temporal rewards entirely.

The painting environment considered in this thesis entails a *hard exploration problem*, due to frequent sparsity and deceptivity of adversarial rewards. As found by Bellemare et al. (2016), classic reinforcement learning exploration techniques often perform poorly on these types of problems, and more sophisticated methods with intrinsic motivation could be explored in future work. The authors suggest count-based exploration using a density model that encourages agents to discover novel states. This type of exploration could also be benefitial in terms of the variety of generated outputs.

Finally, the painting environment is particularly well suited for model-based techniques. Although the environment is not differentiable, its dynamics are considerably easier to learn than those of typical reinforcement learning settings. Environment transition examples can easily be generated by the rendering engine, and these examples can be used to learn an environment model in a fully supervised fashion. This model could in turn be used for intrinsic curiosity driven exploration, which Burda et al. (2018) have shown to work well on several difficult problems.

9.1.3. Environments and Datasets

Future work could also examine the performance of the method on even more complex, multimodal datasets. Although the combination of MNIST, Fashion-MNIST, and CelebA covers a range of different images, other datasets may shed further light on the system behavior and applicability to other problems. Extending the environment to enable more expressive interactions is another interesting area to explore. Additional action dimensions, such as brush velocity, pressure throughout the stroke, or even the type of brush per step, could enable agents to produce even more varied and interesting paintings.

Since method is agnostic with respect to the implementation details of the environment, other types of visual simulations should also be considered. Vector-based drawing and 3D modeling are examples of interesting environments that could be combined with the existing algorithm. The observed behavior in these settings would provide additional insights into the generality of the method when exposed to vastly different environment dynamics.

9.1.4. Model Architectures

The proposed method uses conventional architectures for the discriminator and feature extractor, and a large autoregressive model for the policy. Other architectures could have benefits in this context, and should be considered in future work. The design of the policy model is of particular interest. As previously discussed, there are many ways to provide the necessary information to the autoregressive decoder, and further research is necessary to properly evaluate and justify these design decisions. Recent state-of-the-art techniques, such as the decision transformer introduced by Chen et al. (2021), may also be interesting to consider in this context.

9.1.5. Applications

Generative adversarial reinforcement learning can be applied to many different tasks, but is most interesting in settings where the generative environment can not be easily formulated in a differentiable manner. Sequential painting is one example of this, but other creative endeavours may be also be formulated in this way. Music generation could, for instance, be performed sequentially through actions that add instruments playing a particular set of notes. This formulation is more analogous to how music is produced by human professionals, and has particular potential with respect to the creative control and cooperative ability it could provide. Whereas fully differentiable methods are difficult to interpret, the output of this type of system is more readily understood and can easily be modified by humans.

Looking beyond the generative aspect of the method, adversarial reinforcement learning in general has great potential in solving problems where objective functions are difficult or impossible to define. If an adversarial opponent can learn the objective directly from examples, system dependency on engineered reward systems can be avoided. In many settings, the task of producing examples of desirable environmental states is in fact much easier than quantifying these states. This is particularly true for agents acting in the real world. Building models that can generalize to the complexity of the real world remains a central goal of machine learning as a field, and significant effort is required to overcome these challenges.

Bibliography

Dario Amodei, Chris Olah, Jacob Steinhardt, Paul Christiano, John Schulman, and Dan Mané. Concrete problems in AI safety, 2016. URL https://arxiv.org/abs/1606.06565.

Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein generative adversarial networks. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 214–223. PMLR, 2017. URL https://proceedings.mlr.press/v70/arjovsky17a.html.

Bowen Baker, Ingmar Kanitscheider, Todor Markov, Yi Wu, Glenn Powell, Bob McGrew, and Igor Mordatch. Emergent tool use from multi-agent autocurricula, 2019. URL https://arxiv.org/abs/1909.07528.

Marc G. Bellemare, Sriram Srinivasan, Georg Ostrovski, Tom Schaul, David Saxton, and Remi Munos. Unifying count-based exploration and intrinsic motivation, 2016. URL https://arxiv.org/abs/1606.01868.

Christopher Berner, Greg Brockman, Brooke Chan, Vicki Cheung, Przemyslaw Debiak, Christy Dennison, David Farhi, Quirin Fischer, Shariq Hashme, Christopher Hesse, Rafal Józefowicz, Scott Gray, Catherine Olsson, Jakub Pachocki, Michael Petrov, Henrique Pondé de Oliveira Pinto, Jonathan Raiman, Tim Salimans, Jeremy Schlatter, Jonas Schneider, Szymon Sidor, Ilya Sutskever, Jie Tang, Filip Wolski, and Susan Zhang. Dota 2 with large scale deep reinforcement learning. *CoRR*, abs/1912.06680, 2019. URL http://arxiv.org/abs/1912.06680.

Lukas Biewald. Experiment tracking with Weights and Biases, 2020. URL https://www.wandb.com/. Software available from wandb.com.

Margaret Boden. Creativity in a nutshell. *Think*, 5:83–96, 2009. doi:10.1017/S147717560000230X.

Wieland Brendel and Matthias Bethge. Approximating CNNs with bag-of-local-features models works surprisingly well on ImageNet, 2019. URL https://arxiv.org/abs/1904.00760.

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. OpenAI Gym, 2016. URL https://arxiv.org/abs/1606.01540.

Yuri Burda, Harri Edwards, Deepak Pathak, Amos Storkey, Trevor Darrell, and Alexei A. Efros. Large-scale study of curiosity-driven learning, 2018. URL https://arxiv.org/abs/1808.04355.

Lili Chen, Kevin Lu, Aravind Rajeswaran, Kimin Lee, Aditya Grover, Michael Laskin, Pieter Abbeel, Aravind Srinivas, and Igor Mordatch. Decision transformer: Reinforcement learning via sequence modeling. *CoRR*, abs/2106.01345, 2021. URL https://arxiv.org/abs/2106.01345.

Christopher Clark and Amos Storkey. Teaching deep convolutional neural networks to play Go, 2014. URL https://arxiv.org/abs/1412.3409.

Logan Engstrom, Andrew Ilyas, Shibani Santurkar, Dimitris Tsipras, Firdaus Janoos, Larry Rudolph, and Aleksander Madry. Implementation matters in deep policy gradients: A case study on PPO and TRPO, 2020. URL https://arxiv.org/abs/2005.12729.

Scott Fujimoto, Herke Van Hoof, and David Meger. Addressing Function Approximation Error in Actor-Critic Methods. *35th International Conference on Machine Learning, ICML 2018*, 4:2587–2601, 2018. ISSN 1938-7228.

Yaroslav Ganin, Tejas Kulkarni, Igor Babuschkin, S. M. Ali Eslami, and Oriol Vinyals. Synthesizing programs for images using reinforced adversarial learning. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 1666–1675. PMLR, 2018. URL https://proceedings.mlr.press/v80/ganin18a.html.

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. *Advances in neural information processing systems*, 27, 2014.

Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron Courville. Improved training of Wasserstein GANs, 2017. URL https://arxiv.org/abs/1704.00028.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *35th International Conference on Machine Learning, ICML 2018*, 5:2976–2989, 2018.

Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, 2020. doi:10.1038/s41586-020-2649-2. URL https://doi.org/10.1038/s41586-020-2649-2.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

Nicolas Heess, Dhruva TB, Srinivasan Sriram, Jay Lemmon, Josh Merel, Greg Wayne, Yuval Tassa, Tom Erez, Ziyu Wang, S. M. Ali Eslami, Martin Riedmiller, and David Silver. Emergence of locomotion behaviours in rich environments, 2017. URL https://arxiv.org/abs/1707.02286.

Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. GANs trained by a two time-scale update rule converge to a local nash equilibrium. 2017. doi:10.48550/ARXIV.1706.08500. URL https://arxiv.org/abs/1706.08500.

Ashley Hill, Antonin Raffin, Maximilian Ernestus, Adam Gleave, Anssi Kanervisto, Rene Traore, Prafulla Dhariwal, Christopher Hesse, Oleg Klimov, Alex Nichol, Matthias Plappert, Alec Radford, John Schulman, Szymon Sidor, and Yuhuai Wu. Stable Baselines. https://github.com/hill-a/stable-baselines, 2018.

Geoffrey Hinton. Overview of mini-batch gradient descent, 2018. URL https://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9:1735–80, 1997. doi:10.1162/neco.1997.9.8.1735.

Matt Hoffman, Bobak Shahriari, John Aslanides, Gabriel Barth-Maron, Feryal Behbahani, Tamara Norman, Abbas Abdolmaleki, Albin Cassirer, Fan Yang, Kate Baumli, Sarah Henderson, Alex Novikov, Sergio Gómez Colmenarejo, Serkan Cabi, Caglar Gulcehre, Tom Le Paine, Andrew Cowie, Ziyu Wang, Bilal Piot, and Nando de Freitas. Acme: A research framework for distributed reinforcement learning. *arXiv preprint arXiv:2006.00979*, 2020. URL https://arxiv.org/abs/2006.00979.

Ling Huang, Anthony D. Joseph, Blaine Nelson, Benjamin I.P. Rubinstein, and J. D. Tygar. Adversarial machine learning. In *Proceedings of the 4th ACM Workshop on Security and Artificial Intelligence*, AISec '11, page 43–58, New York, NY, USA, 2011. Association for Computing Machinery. ISBN 9781450310031. doi:10.1145/2046684.2046692. URL https://doi.org/10.1145/2046684.2046692.

Yanhua Huang. Deep Q-networks. *Deep Reinforcement Learning: Fundamentals, Research and Applications*, pages 135–160, 2020. doi:10.1007/978-981-15-4095-0_4.

Zhewei Huang, Shuchang Zhou, and Wen Heng. Learning to paint with model-based deep reinforcement learning. *Proceedings of the IEEE International Conference on Computer Vision*, 2019-Octob:8708–8717, 2019. ISSN 15505499. doi:10.1109/ICCV.2019.00880.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift, 2015. URL https://arxiv.org/abs/1502.03167.

Biao Jia, Chen Fang, Jonathan Brandt, Byungmoon Kim, and Dinesh Manocha. PaintBot: A Reinforcement Learning Approach for Natural Media Painting. 2019. URL http://arxiv.org/abs/1904.02201.

Leslie Pack Kaelbling, Michael L Littman, and Andrew W Moore. Reinforcement learning: A survey. *Journal of artificial intelligence research*, 4:237–285, 1996.

Tero Karras, Timo Aila, Samuli Laine, and Jaakko Lehtinen. Progressive growing of GANs for improved quality, stability, and variation. *CoRR*, abs/1710.10196, 2017. URL http://arxiv.org/abs/1710.10196.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2014. URL https://arxiv.org/abs/1412.6980.

Almar Klein, Sebastian Wallkötter, Steven Silvester, Anthony Tanbakuchi, et al. imageio/imageio: v2.19.2, 2022. URL https://doi.org/10.5281/zenodo.6551868.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. ImageNet classification with deep convolutional neural networks. In F. Pereira, C.J. Burges, L. Bottou, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc., 2012. URL https://proceedings.neurips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf.

Brenden M. Lake, Ruslan Salakhutdinov, and Joshua B. Tenenbaum. Human-level concept learning through probabilistic program induction. *Science*, 350(6266):1332–1338, 2015. doi:10.1126/science.aab3050. URL https://www.science.org/doi/abs/10.1126/science.aab3050.

Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural Computation*, 1(4):541–551, 1989. doi:10.1162/neco.1989.1.4.541.

Yann LeCun and Corinna Cortes. MNIST handwritten digit database. 2010. URL http://yann.lecun.com/exdb/mnist/.

Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning, 2015. URL https://arxiv.org/abs/1509.02971.

Ziwei Liu, Ping Luo, Xiaogang Wang, and Xiaoou Tang. Deep learning face attributes in the wild. *CoRR*, abs/1411.7766, 2014. URL http://arxiv.org/abs/1411.7766.

Xudong Mao, Qing Li, Haoran Xie, Raymond Y. K. Lau, Zhen Wang, and Stephen Paul Smolley. Least squares generative adversarial networks, 2016. URL https://arxiv.org/abs/1611.04076.

Sébastien Marcel and Yann Rodriguez. Torchvision the machine-vision package of Torch. In *Proceedings of the 18th ACM International Conference on Multimedia*, MM '10, page 1485–1488, New York, NY, USA, 2010. Association for Computing Machinery. ISBN 9781605589336. doi:10.1145/1873951.1874254. URL `https://doi.org/10.1145/1873951.1874254`.

John F. J. Mellor, Eunbyung Park, Yaroslav Ganin, Igor Babuschkin, Tejas Kulkarni, Dan Rosenbaum, Andy Ballard, Theophane Weber, Oriol Vinyals, and S. M. Ali Eslami. Unsupervised doodling and painting with improved SPIRAL. 2019. URL `https://arxiv.org/abs/1910.01007v1`.

Takeru Miyato, Toshiki Kataoka, Masanori Koyama, and Yuichi Yoshida. Spectral normalization for generative adversarial networks. *ICLR 18*, feb 2018. URL `https://arxiv.org/abs/1802.05957`.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin A. Riedmiller. Playing atari with deep reinforcement learning. *CoRR*, abs/1312.5602, 2013. URL `http://arxiv.org/abs/1312.5602`.

Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In Maria Florina Balcan and Kilian Q. Weinberger, editors, *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pages 1928–1937, New York, New York, USA, 2016. PMLR. URL `https://proceedings.mlr.press/v48/mniha16.html`.

Open Source Github Repository (120+ contributors) MyPaint. libmypaint, 2021. URL `https://github.com/mypaint/libmypaint`.

Reiichiro Nakano. Neural Painters: A learned differentiable constraint for generating brushstroke paintings. 2019. URL `http://arxiv.org/abs/1904.08410`.

Allen Newell, J Clifford Shaw, and Herbert A Simon. The processes of creative thinking. In *Contemporary Approaches to Creative Thinking, 1958, University of Colorado, CO, US; This paper was presented at the aforementioned symposium.* Atherton Press, 1962.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. PyTorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019. URL `http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf`.

Joelle Pineau, Philippe Vincent-Lamarre, Koustuv Sinha, Vincent Larivière, Alina Beygelzimer, Florence d'Alché Buc, Emily Fox, and Hugo Larochelle. Improving reproducibility in machine learning research (a report from the NeurIPS 2019 Reproducibility Program), 2020. URL https://arxiv.org/abs/2003.12206.

Lerrel Pinto, James Davidson, Rahul Sukthankar, and Abhinav Gupta. Robust adversarial reinforcement learning. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 2817–2826. PMLR, 2017. URL https://proceedings.mlr.press/v70/pinto17a.html.

Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks. *4th International Conference on Learning Representations, ICLR 2016 - Conference Track Proceedings*, 2015. URL https://arxiv.org/abs/1511.06434v2.

Antonin Raffin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, and Noah Dormann. Stable-Baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, 22(268):1–8, 2021. URL http://jmlr.org/papers/v22/20-1364.html.

Aditya Ramesh, Prafulla Dhariwal, Alex Nichol, Casey Chu, and Mark Chen. Hierarchical text-conditional image generation with CLIP latents, 2022. URL https://arxiv.org/abs/2204.06125.

Sigve Røkenes. Differentiable neural painter, 2020. URL https://github.com/evgiz/neural-painter.

Sigve Røkenes. Unsupervised image synthesis with deep reinforced adversarial learning, 2021. Specialization Project / Preparatory Work. Department of Computer Science, Norwegian University of Science and Technology.

Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. Improved techniques for training GANs, 2016. URL https://arxiv.org/abs/1606.03498.

John Schulman, Sergey Levine, Philipp Moritz, Michael Jordan, and Pieter Abbeel. Trust region policy optimization. In *32nd International Conference on Machine Learning, ICML 2015*, volume 3, pages 1889–1897. International Machine Learning Society (IMLS), 2015a. ISBN 9781510810587. URL https://arxiv.org/abs/1502.05477v5.

John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using Generalized Advantage Estimation, 2015b. URL https://arxiv.org/abs/1506.02438.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal Policy Optimization Algorithms, 2017. URL https://arxiv.org/abs/1707.06347.

John R Searle. Minds, brains, and programs. *Behavioral and brain sciences*, 3(3):417–424, 1980.

David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489, 2016.

Magnus Själander, Magnus Jahre, Gunnar Tufte, and Nico Reissmann. EPIC: An energy-efficient, high-performance GPGPU computing research infrastructure, 2019.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958, 2014. URL http://jmlr.org/papers/v15/srivastava14a.html.

Richard S Sutton, David McAllester, Satinder Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. 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/1999/file/464d828b85b0bed98e80ade0a5c43b0f-Paper.pdf.

Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision, 2015. URL https://arxiv.org/abs/1512.00567.

Hoang Thanh-Tung and Truyen Tran. Catastrophic forgetting and mode collapse in GANs. In *2020 International Joint Conference on Neural Networks (IJCNN)*, pages 1–10, 2020. doi:10.1109/IJCNN48605.2020.9207181.

Emanuel Todorov, Tom Erez, and Yuval Tassa. MuJoCo: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033, 2012. doi:10.1109/IROS.2012.6386109.

Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *CoRR*, abs/1708.07747, 2017. URL http://arxiv.org/abs/1708.07747.

Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A. Efros. Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks. *Proceedings of the IEEE International Conference on Computer Vision*, 2017-October:2242–2251, 2017. URL https://arxiv.org/abs/1703.10593v7.

A. Implementation

A.1. Brush Configuration

The following example shows how the appearance of a brush is defined in a configuration file. The configuration defines settings such as the type of stroke, brush texture, texture spacing, scaling, and random jitter variation.

```
name: Splatter Paint Brush
texture:
  path: splatter.png
  opacity: 1.0
stroke:
  type: bezier
  step: 0.2
control:
  max_thickness: 0.5
  min_thickness: 0.02
  max_opacity: 1
  min_opacity: 0.5
dab:
  start_scale: 1
  end_scale: 0.4
  jitter:
    scale: 0.5
    rotation: 180
    position: 0.15
    color: 0.2
    opacity: 0.1
  follow_stroke_angle: true
post_processing:
  blur: 0
  noise: 0
  edge_blend: 0
```

A.2. Alternate Policies

Three alternate policy variants are considered — the gaussian policy, beta policy and categorical policy. All policies employ a multi-layer perceptron which process the features provided by the feature extractor. In the gaussian- and beta policies, the layer is followed by two separate linear layers which model the means and standard deviations or the alpha and beta parameters respectively. The categorical policy uses a set of linear layers to model distributions for each action dimension. Figure A.1 shows an overview of these architectures.

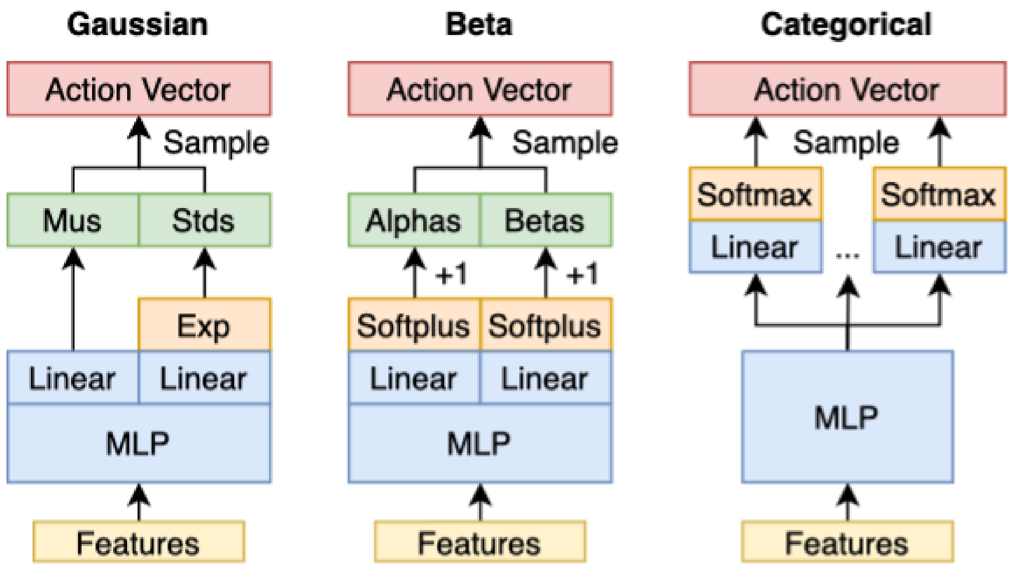


Figure A.1.: **Alternate policy architectures.**

A.3. Default Configuration

The following configuration file shows the default setup for the method. Given the large amount of available hyperparameters, the file serves as a good overview of the technique. This example uses the WGAN-GP discriminator, but this setting, as well as any others, are easily changed.

```
# ============ #
#    General   #
# ============ #

iterations: 1000000
dataset: celeba-align
```

```
agent_population: 1
discriminator_population: 1

target_batch: 4096
max_parallel_envs: 64
min_parallel_envs: 16

# ================= #
#    Algorithm      #
# ================= #

algorithm: PPO

# Common
alg:

  n_epochs: 5
  batch_size: 64

  use_rms_prop: true
  learning_rate: 0.0001
  max_grad_norm: 0.5
  lr_decay: null

  vf_coef: 0.5
  ent_coef: 0.01

# PPO specific
ppo:
  mode: clip
  clip_range: 0.2
  kl_d_target: 0.03

# Rollout
rollout:
  gae_gamma: 0.99
  gae_lambda: 0.95
  normalize_returns: false
  normalize_rewards: false
  estimator: gae
```

```
# =============== #
#    Environment    #
# =============== #

reward_system:
  kind: a_temporal
  clip: 10.0
  scale: 1.0

img:
  channels: 3
  width: 64
  height: 64

brush:
  channels: 3
  opacity: true
  thickness: true
  use_hsv: false
  contiguous: false
  path: ./siro_paint_gym/brushes/paint/splatter_fast_v3.yaml

env:
  n_strokes: 64
  background: !!python/tuple
  - 1.0
  - 1.0
  - 1.0
  - 1.0
  render_scale: 4
  render_downscale: null
  padding: 0.0

# =============== #
#  Architectures  #
# =============== #

policy_head:
  embed_steps: false
  embed_noise: false
  embed_action: false
```

```
value_model:
  embed_steps: false

extractor_model:
  value_separate: false
  resblocks: 8
  # Embedding
  embed_grid: true
  embed_steps: false
  embed_noise: true
  embed_action: true

discriminator_model:
  fc_out: false
  model_chn: 64
  spectral_norm: true
  norm: null

# ================ #
#  Discriminator  #
# ================ #

discriminator:

  dataset_label: null

  # Batch
  train_freq: 1
  batch_size: 64
  max_epochs: 1

  # Buffer
  on_policy: false
  buffer_size: 1500
  buffer_priority: recent
  train_on_incomplete: false

  # Optimizer
  use_rms_prop: true
  learning_rate: 0.0001
  max_grad_norm: null
```

```
# Loss
loss_type: wgan
gradient_penalty: 10.0
weight_clip: null
l2_penalty: null
soft_target: 0.0

# Expectation
expectation_batch: 64
```

A.4. Experiment Tracking

This work uses Weights and Biases by Biewald (2020) for experiment tracking. Figures A.2 and A.3 shows examples of key metrics tracked on a typical experiment dashboard.

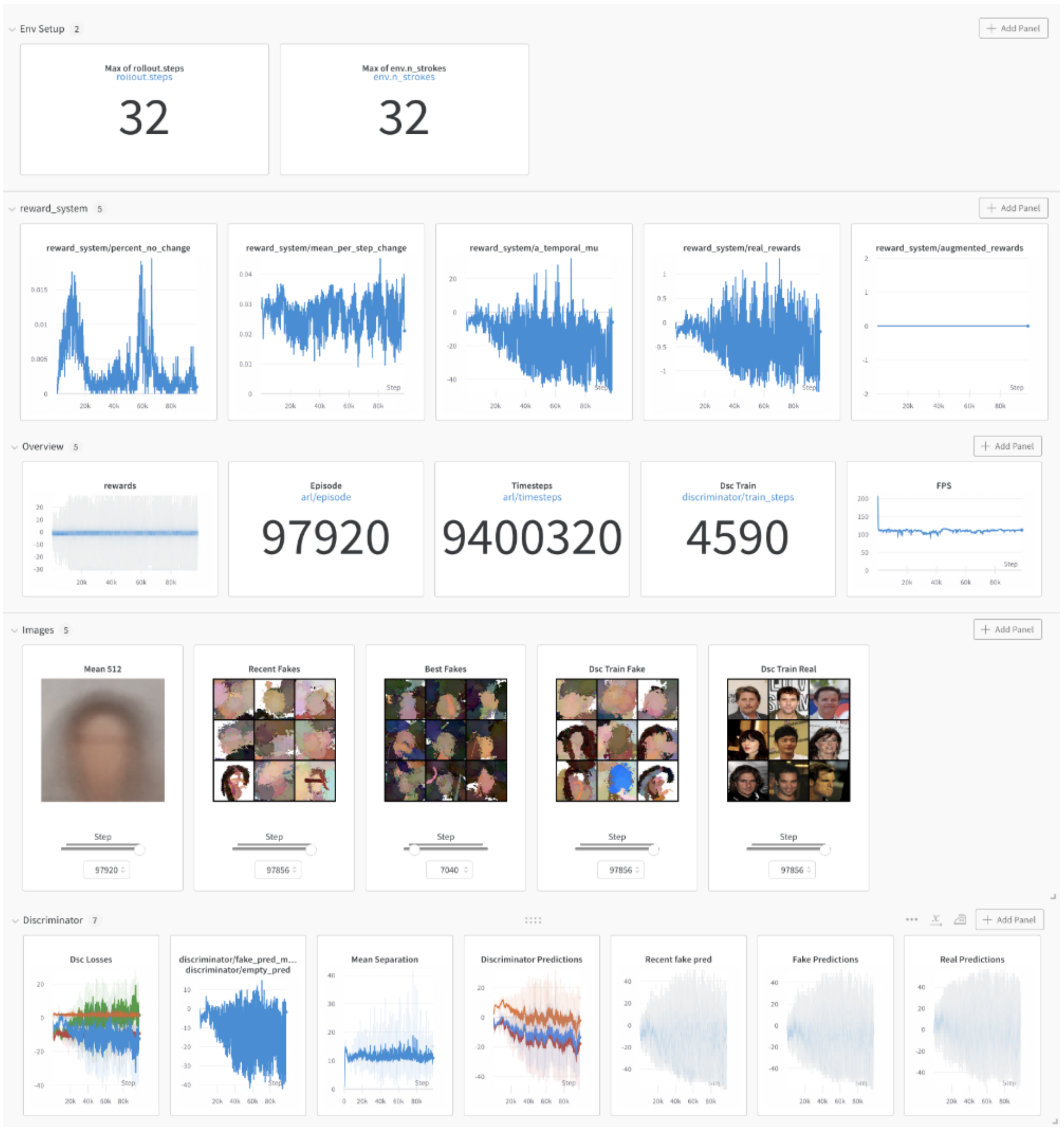


Figure A.2.: **Experiment dashboard 1/2.** Example of various metrics tracked using Weights and Biases during an experiment.

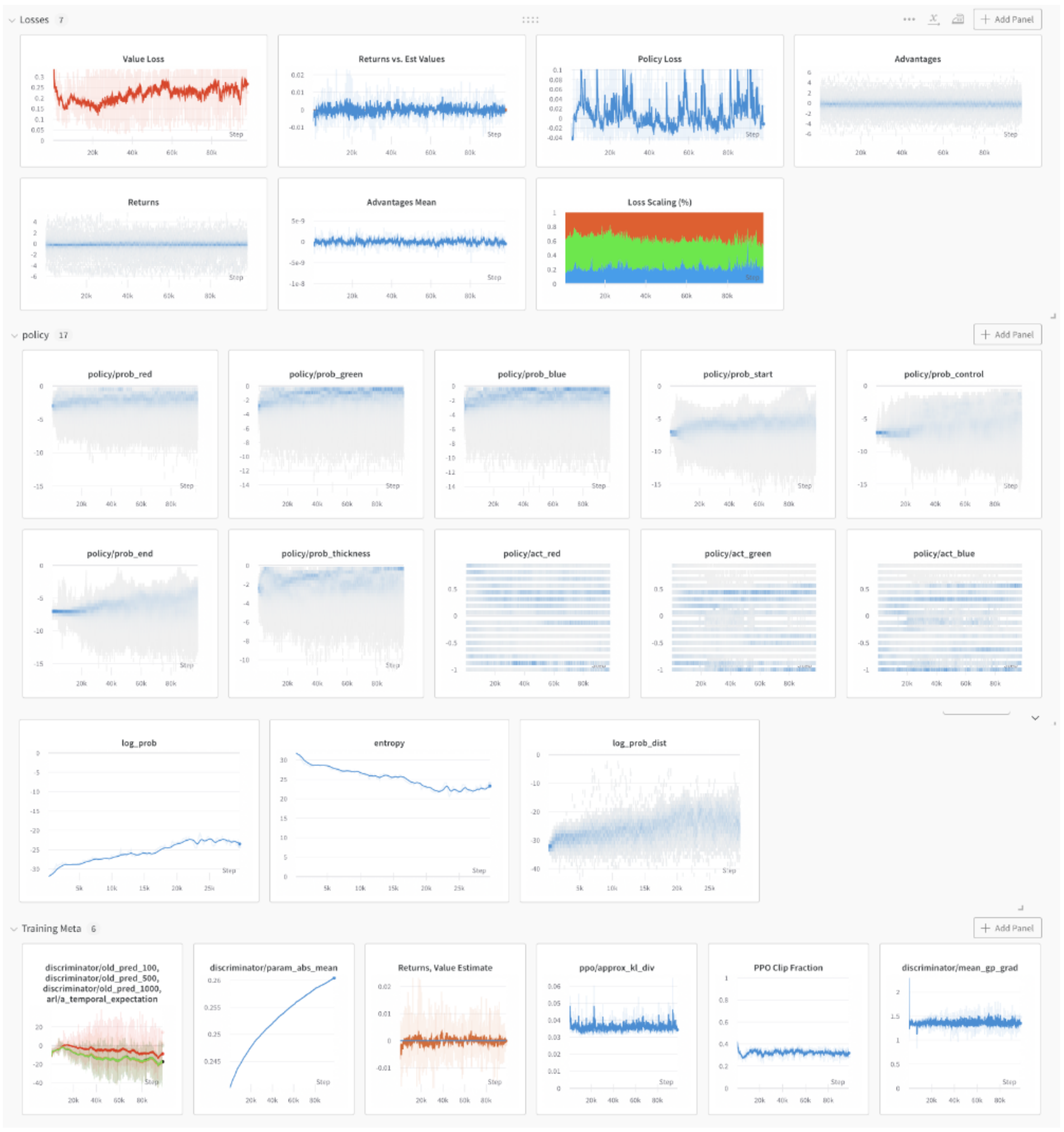


Figure A.3.: **Experiment dashboard 2/2.** Example of various metrics tracked using Weights and Biases during an experiment.

B. Equations

B.1. Derivation of the Policy Gradient†

The probability of a trajectory generated by π_θ is given by the equation:

$$P(\tau \mid \theta) = p_0(s_0) \prod_{t=0}^{T} P(s_{t+1} \mid s_t, a_t) \pi_\theta(a_t \mid s_t) \tag{B.1}$$

Where $p_0(s_0)$ is the probability of the initial state. We can use the log-probability to simplify calculations while maintaining a valid gradient. The log-probability of the trajectory is given by:

$$\log\ P(\tau \mid \theta) = \log\ p_0(s_0) + \sum_{t=0}^{T} (\log\ P(s_{t+1} \mid s_t, a_t) + \log\ \pi_\theta(a_t \mid s_t)) \tag{B.2}$$

The environment is independent from θ, so the $p_0(s_0)$ and $P(s_{t+1} \mid s_t, a_t)$ terms can be excluded from the gradient with respect to θ:

$$\nabla_\theta\ log\ P(\tau \mid \theta) = \sum_{t=0}^{T} \nabla_\theta\ \log\ \pi_\theta(a_t \mid s_t) \tag{B.3}$$

The gradient of the objective function is given by:

$$\nabla_\theta\ J(\pi_\theta) = \nabla_\theta \int_\tau P(\tau \mid \theta) R(\tau) \tag{B.4}$$

We move the gradient term inside the integral, change to expectation form, and use the log derivative trick to get:

$$\nabla_\theta\ J(\pi_\theta) = \mathop{\mathbb{E}}_{\tau \sim \pi_\theta} [\nabla_\theta\ \log\ P(\tau \mid \theta) R(\tau)] \tag{B.5}$$

Substituting the trajectory probability with Equation B.3 we get the final policy gradient:

$$\nabla_\theta\ J(\pi_\theta) = \mathop{\mathbb{E}}_{\tau \sim \pi_\theta} [\sum_{t=0}^{T} \nabla_\theta\ \log \pi_\theta(a_t \mid s_t) R(\tau)] \tag{B.6}$$

Sigve Røkenes

Generative Adversarial Reinforcement Learning

NTNU
Kunnskap for en bedre verden