Advantage Actor-Critic Methods
for CarRacing

Douwe van der Wal
11042206

Bachelor thesis
Credits: 18 EC

Bachelor Opleiding Kunstmatige Intelligentie

University of Amsterdam
Faculty of Science
Science Park 904
1098 XH Amsterdam

Supervisor
Wenling Shang

Informatics Institute
Faculty of Science
University of Amsterdam
Science Park 904
1098 XH Amsterdam

July 15, 2018
Abstract

Autonomous driving, a complex integration of perception, planning and control, is one of today’s most prominent, fast-growing technology. Recent works have proposed to pose the learning of autonomous driving agent as a Reinforcement Learning problem [Sallab, Abdou, Perot, & Yogamani 2017], motivated by the interactive nature between the autonomous vehicle and its driving environment. As operating autonomous driving tests in the physical world is extremely costly, most of relevance decision making algorithms are developed in simulation. In this thesis, we aim at prototyping agents via one of the most successful and popular RL frameworks, advantage-based actor-critic, to solve a toy problem, CarRacing that embeds the core driving logic.

We first construct a mini-batch asynchronous advantage actor-critic baseline with a recurrent LSTM module, serving as the foundation block to the rest of our algorithm development. Next, giving due consideration, we choose among the many factors impacting the quality of agents to specifically focus on investigating and optimizing over the construction of action spaces, the implementation of exploration mechanisms and the employment of experience replays, as a thorough understanding of the functionality of these components will be able to guide future developments of actor-critic methods to more complex simulations.

After conducting systematic control experiments, we reach the conclusions that it is the most desirable to train with discrete factorized action space (i.e. discretizing the continuous control factors into categorical values), it is advantageous to apply stochastic activation to encourage exploration especially when coupling with an action space composed of manually specified action factor combinations, and it is beneficial to sample efficiency when incorporating an appropriate amount of off-policy training on experience replays.

Finally, in Discussion, we perform more qualitatively and quantitatively analysis to provide reasoning on why the default continuous action space is incompatible with actor-critic training, to better grasp the compositional properties of the learned factorized action space, and to identify the pros and cons of utilizing experience replays.

We hope the findings from this thesis will shed light on future research in learning autonomous agents with advantage actor-critic frameworks.
Acknowledgements

First and foremost I would like to thank my supervisor Wenling Shang for the great guidance throughout the project. With so much enthusiasm she has provided an extraordinary amount of feedback on my work and given me many enlightening explanations related to the theory of this thesis. In addition she has also provided the computational resources to make this thesis possible, which I am very grateful for. Finally I would also like to thank her for the opportunity to work on a scientific paper for the first time from which I’ve learned a lot. Furthermore I would like to thank my fellow student Laurens Weitkamp for useful peer reviews and enlightening discussions on subjects related to this thesis. Finally I would like to thank the Intelligent Robotics Lab at the UvA for providing additional computing power to complete the experiments.
Contents

1 Introduction ........................................... 5

2 Background ........................................... 6
   2.1 Reinforcement Learning ................................. 6
   2.2 Deep Reinforcement Learning ......................... 8
   2.3 Driving Simulators ..................................... 9

3 Related Works ........................................ 9
   3.1 DRL-based Advantage Actor-Critic Methods .......... 10
   3.2 Continuous Action Space and Discretization .......... 11
   3.3 Exploration in an Environment ......................... 11
   3.4 Algorithms for Driving Simulators ...................... 12

4 Methodology ......................................... 12
   4.1 Mini-batch LSTM Advantage Actor-Critic Methods .... 13
   4.2 Policy Networks for Various Action Spaces .......... 14
   4.3 Exploration for A3C ................................ 15
   4.4 Advantage Actor-Critic with Experience Replay ....... 16

5 Environment .......................................... 16

6 Experiments and Result Analysis ....................... 18
   6.1 Model Architecture and Hyperparameters ............. 19
   6.2 Discover the Optimal Action Space ................... 19
   6.3 Exploration Mechanisms ................................ 21
   6.4 Actor-Critic with Experience Replay .................. 22

7 Discussion ........................................... 23
   7.1 Explaining the Deficiency of Continuous Space ....... 23
   7.2 Looking into the Learned Factorized Action Space .... 24
   7.3 Understanding ACER Performance ...................... 25

8 Conclusion ............................................ 26

References ............................................. 28
1 Introduction

Machine learning is a subdivision of Artificial Intelligence where machines learn to make decisions and recognize patterns from data without being explicitly programmed. There are three main types of learning: supervised learning, unsupervised learning and reinforcement learning. Supervised learning attempts to learn a map between an input space and its label space, where correct input-label examples are provided during training. In contrary, unsupervised learning aims to extract information from unlabeled input data. Unsupervised learning can provide a better representation for a supervised task or come up with its own task.

Reinforcement learning (RL) aims at learning to interact with the environment in a way such that an accumulative reward is maximized. Reinforcement learning provides solutions to goal-oriented tasks in replacement of supervised learning methods when a simulator of the environment is available instead of input-label training pairs. Consequentially, unlike supervised learning, RL does not output a specific target given an input (in this case an observation) but rather a means to interact with the environment. For example, for the task of learning to drive, a supervised learning algorithm would leverage a dataset consisting of pairs of observations and their corresponding optimal actions. However, collecting such dataset is highly challenging. On the other hand, given a driving simulator, a RL approach can directly perform training basing on the reward signal from the simulator.

Deep reinforcement learning (DRL) integrates the recent advances in deep neural networks (DNNs) to RL by for example parametrizing value approximators with DNNs (Mnih et al., 2013)(Mnih et al., 2015). In this thesis, motivated by the important application of autonomous driving, we study the RL problem of operating a simulated racing car using DRL-based asynchronous actor-critic methods (Mnih et al., 2016; Z. Wang et al., 2016). A real-world autonomous driving setup involves many substantially more complicated components, on top of which it is very costly to evaluate a proposed RL algorithm. Therefore, by prototyping on a toy example like ours, we hope to gain useful insights in designing effective actor-critic models to potentially avoid investing an excessive amount of resources when experimenting with real-world scenarios.

There are many factors in the success of an RL solution. In this thesis, we selectively attend to investigate and optimize 3 fundamental aspects, namely the design of action space design, the exploration mechanisms, and the utilization of experience replay. We believe the empirical findings and the high-level intuition attained from our studies are not only limited to the CarRacing problem but can be carried over to a diverse range of autonomous driving applications corroborated by advantage actor-critic methods. Our goals are summarized as follows:

- Establish a batch-version asynchronous actor-critic framework with LSTM module (A3C-LSTM) on CarRacing (Shang, van der Wal, van Hoof, & Welling, 2018).
- Compare different types of action space for the actor, including the default continuous action space, the discrete factorized action space and the
handcrafted set of discrete action factor combinations.

- Experiment additional exploration techniques in combination to the default Boltzman Exploration, namely NoisyNet (Fortunato et al., 2017) and Stochastic Activations (Shang et al., 2018).
- Implement and evaluate Actor-Critic with Experience Replay (ACER) on CarRacing (Z. Wang et al., 2016) as an attempt to improve sample efficiency, in preparation of a scenario when simulation data is scarce.

The rest of the thesis is organized as follows: we first introduce background information on RL, DRL, and driving simulators in Section 2. Then we will discuss the related works of relevance to this thesis in greater details (Section 3). Next Section 4 rigorously describes the algorithms used in this thesis and Section 5 the environment. The experimental results and preliminary analysis are presented in Section 6 followed by more in-depth discussions in Section 7. Finally, we conclude the thesis with Section 8.

2 Background

In this Section, we present the most relevant Reinforcement Learning (RL) background knowledge required to grasp the content of this thesis. We start with general RL concepts (Section 2.1), followed by more elaborations on the recently popularized Deep Learning (DL) based approaches to RL (Section 2.2). In addition, Section 2.3 provides more information on driving simulator, including their roles in building an autonomous driving system, several popular open source examples in research communities, and the rationales for this thesis to select the CarRacing simulator.

2.1 Reinforcement Learning

Reinforcement Learning is an area of machine learning, aiming at learning the optimal behaviour in an environment which maximizes the accumulative reward. The decision-making process to control the environment is often considered to be an Markov Decision Process (MDP) (Sutton & Barto, 1998). More precisely, an MDP consists of a set of states \( S \), which can be finite, infinite or even continuous, a set of executable actions \( A \), which can be discrete or continuous, a transition probability \( P(s_{t+1}|a_t, s_t) \) describing the dynamics of an agent interacting with its environment, and a reward function at a given initial, final state after taking a certain action \( r(s_t, a_t, s_{t+1}) = E[R_{t+1}|s_t, a_t, s_{t+1}] \).

Many RL problems follow the setup of an MDP, including the CarRacing task. A RL algorithm aims at solving a RL problem, which can either be model-based (Sutton & Barto, 1998), meaning that it leverages the underlying environment model for planning, or model-free (Sutton & Barto, 1998), meaning that its decision making process does not explicitly consider how the environment changes in response to an action. Model-free algorithms are generally preferred when it is difficult to obtain a sufficiently accurate environment model. The observations of CarRacing are composed of raw pixels, resulting in a complex
environment that is difficult to model, hence model-free algorithms, such as actor-critic methods, are better suited.

The training of a RL algorithm can be done on-policy or off-policy. An off-policy training is done on top of trajectories generated using a policy different from the current one, whereas on-policy training improves the current policy based on the actions it generates. A significant portion of the thesis focuses on on-policy training (See Section 4.1) and another on/off-policy hybrid training scheme is also considered (See Section 4.4).

The direct objectives of RL algorithms generally fall into two categories: value function approximation and policy function approximation.

Value function approximation targets at estimating either the state values, $V_t$ (or $V(s_t)$ at time $t$) – i.e. how desirable it is for an agent to be in a state, or the state-action values, $Q_t$ (or $Q(s_t, a_t)$ at time $t$) – how desirable it is to take a certain action in a state (Sutton & Barto, 1998). Their mathematical definitions are

$$Q_t = Q(s_t, a_t) = \mathbb{E}_{s_{t+1}, a_{t+1}} \left[ \sum_{i=0}^{\infty} \gamma^i r_{t+i} \right], V_t = V(s_t) = \mathbb{E}_{a_t} \left[ Q(s_t, a_t) | s_t \right],$$

where $\gamma (0 \leq \gamma \leq 1)$ is the discount factor determining the importance of future reward to the current state values. Classic RL methods for value approximation includes Monte Carlo methods (Sutton & Barto, 1998) for state or state-action value approximation, temporal-difference methods such as the off-line Q-learning (Watkins, 1989) and the on-line SARSA (Rummery & Niranjan, 1994) for state-action value approximation. Once a value approximator is readily trained, planning can be done via e.g. greedily selecting an action that is optimal based on the value approximator.

Policy gradient method, a gradient-based policy learning approach, directly optimize on top of the action space. Intuitively, the policy parameter updates should encourage actions resulting in positive rewards and vise versa. Specifically, the Policy Gradient Theorem (Sutton, McAllester, Singh, & Mansour, 2000) requires the parameters of the policy to be updated using the gradients with respect to the corresponding state-action value. One of the most commonly used policy gradient methods is REINFORCE (Williams, 1992) where the state-action value is approximated via long-term return. However, a major issue, as one can imagine, for policy gradient is its high variance. To address this, an unbiased state-dependent baseline is usually subtracted from the score.

Another way to reduce policy gradient variance is to simultaneously learn a critic that directly approximates the state-action values, arriving at the actor-critic methods. However, such an approximation can introduce bias. To tackle this, the (approximated) state-value, functioning as a baseline, is often subtracted from the approximated state-action value to form the policy gradient, referred to as the advantage function (Grondman, Busoniu, Lopes, & Babuska, 2012). In this thesis, we leverage the actor-critic framework as our algorithmic foundation and more related mathematical details are in Section 4.
2.2 Deep Reinforcement Learning

Classic RL algorithms, despite success with finite MDP and its mathematical rigorosity, experience limitations when facing large-scale problems. For example, many classic value-approximation RL algorithms require to maintain a value look-up table (Sutton & Barto, 1998) for each state or a state-action pair; however, when the state space grows, it becomes difficult to scale not only in terms of memory but also the excessive amount of samples needed in order to fill up the table relatively accurately—each state or state-action pair needs to be visited multiple times. A step towards solving such scalability issues is, borrowing an idea from supervised learning to generalize with limited training data, function approximation for value and policy (Tsitsiklis & Van Roy, 1997). Intuitively, given a previously unseen state, function approximators can achieve reasonable estimation by generalizing from experiences on seen states with similar properties. However, due to computational constraints, most classic RL algorithms apply linear function approximation, and mostly on handcrafted features if the observation is of high dimensionality (Bellemare, Veness, & Bowling, 2012). These practices would clearly limit the capacity of a RL agent, which is especially problematic with a complex environment or task.

Thanks to the recent advances in computational power, tools in Deep Learning (DL) in replacement of classic function approximators have been utilized to overcome this limitation.

An example of Deep RL (DRL) application is Deep Q-networks (DQN) where an observation is processed using an CNN (Mnih et al., 2013), the subsequent features are fed into a value function represented by a DNN, and the system is trained end-to-end following the theoretical principles of off-policy Q-learning. With the additional help of experience replay (Lin, 1993) to decorrelate replay sampling—i.e. avoid overly similar examples for each training batch, superhuman level can be achieved for many Atari 2600 games. DNNs can also represent policies and be trained with either supervised objectives by transforming policy search (Levine, Finn, Darrell, & Abbeel, 2016) or policy gradients such as using REINFORCE (Xu et al., 2015). DRL-based actor-critic methods use DNNs to represent both values and policies, where the actor is updated via policy gradients through the feedback from the critic (Mnih et al., 2016). This family of methods have achieved great success across many RL applications, from mastering the game of GO (Silver et al., 2016) to Robotics locomotion (Schulman, Moritz, Levine, Jordan, & Abbeel, 2015). As aforementioned, in this thesis, actor-critic methods serve as our algorithmic backbone and we in particularly focus on exploring DRL-based advantage actor-critic models. More details on the models used in our work will be discussed in the later sections.

It is also worth mentioning that many DRL models (Wierstra, Förster, Peters, & Schmidhuber, 2010; Mnih et al., 2016; Hausknecht & Stone, 2015) can seamlessly incorporate recurrent modules such as a long short-term memory unit (LSTM), to take into account of longer temporal dependencies (Hochreiter & Schmidhuber, 1997). Such property is especially important when the environment is partially observable or if a long horizon planning is essential for an agent to excel. Although, the hidden units with compressed history leads to a non-Markovian policy, more resembling the belief states of a partially observable
MDP (POMDP) [Rao, 2010]. Alternatively, without using recurrent modules, a stack of consecutive frames can be used as input to engage more temporal information but on a much shorter term. Therefore, to gain a good temporal understanding of the environment, our models also incorporate a LSTM module, which will be further detailed in Section 6.

2.3 Driving Simulators

An autonomous driving system involves many complex technologies, from both software and hardware side. Directly instrumenting cars in the physical world to validate algorithms for autonomous driving is not feasible due to the excessive cost of money, labor, and time. Training and testing in simulation that reflects certain real-world driving principles, on the other hand, serves as a great alternative. There are two general steps: first is algorithm development within simulation [Pomerleau, 1989] and next is algorithm adaptation from the simulator to the physical world [You, Pan, Wang, & Lu, 2017]. In this thesis, we are interested in the first step.

Several popular open source simulators are available for prototyping autonomous driving agents. CarRacing-v0 from the OpenAI Gym box2d module is a simple 2D racing game, providing a top view of the car and track (Figure 1(a)), VDrift and TORCS [Wymann et al., 2000] (Figure 1(b)) simulate more realistic 3D car racing scenes. Grand Theft Auto V [Richter, Vineet, Roth, & Koltun, 2016], a commercial game, and CARLA [Dosovitskiy, Ros, Codevilla, Lopez, & Koltun, 2017] (Figure 1(c)), another recently proposed platform, both simulate urban driving scenes at high fidelity but are computationally expensive.

Given limited time and resources, and also consider that our primary goal is to demonstrate the fitness of various algorithm designs for learning the core driving logic (such as steering and lane following), we decide on the computationally light and fast-to-render CarRacing-v0 simulator.

3 Related Works

The related works section first presents recent progress on DRL-based asynchronous advantage actor-critic methods. Next, in preparation to compare different types of action space constructions and different exploration techniques,
we examine existing treatments of the continuous action space (Section 3.2) and exploration mechanisms (Section 3.3) respectively. Lastly, we review past RL efforts in training self-driving agents in simulation (Section 3.4).

3.1 DRL-based Advantage Actor-Critic Methods

In recent years, advantage actor-critic methods have achieved many impressive results, from visual navigation (Zhu et al., 2017) to game playing (Y. Wu & Tian, 2016). The most fundamental work behind these successes is the asynchronous advantage actor-critic (A3C) (Mnih et al., 2016), a simple, light-weighted, model-free, on-policy algorithm, originally designed for distributed CPU training. A3C spawns multiple agents to simultaneously interact with their own copies of environment, while maintaining a single shared model to approximate state-value and policy through asynchronous gradient descents from each agent.

The baseline A3C algorithm can be further enhanced by more carefully modulating the policy gradients, for instance through a more advanced baseline such as generalized advantage estimator (GAE) (Schulman, Moritz, et al., 2015), better constrained policy updates such as trust region policy optimization (TRPO) (Schulman, Levine, Abbeel, Jordan, & Moritz, 2015), better constrained policy updates such as trust region policy optimization (TRPO) (Schulman, Levine, Abbeel, Jordan, & Moritz, 2015), better constrained policy updates such as trust region policy optimization (TRPO) (Schulman, Levine, Abbeel, Jordan, & Moritz, 2015), and proximal policy optimization (PPO) (Schulman, Wolski, Dhariwal, Radford, & Klimov, 2017). Along this direction, ACKTR (Y. Wu, Mansimov, Grosse, Liao, & Ba, 2017) further approximates the Fisher information matrix in the trust region constraint with Kronecker-factored approximate curvature, to achieve a natural-gradient flavored update instead of 1st order gradient descent. Recent efforts have also incorporated off-policy training via importance sampling to improve sample efficiency, such as in the case of ACER (Z. Wang et al., 2016).

In parallel to algorithmic improvements, many recent works have also managed to scale up and speed up actor-critic training. Batched advantage actor-critic models (Clemente, Castejón, & Chandra, 2017) with synchronous gradient updates enable more efficient usage of GPUs and more stable training, though bottlenecked by the speed of the slowest agent. In another type of batch training, such as IMPALA (Espeholt et al., 2018) and GA3C (Babaeizadeh, Frosio, Tyree, Clemons, & Kautz, 2016), the agents continuously submit experiences to a queue while another process samples from the queue, creating batches of replays. As the replays remain in the queue for a short period of time after their original policy being updated, off-policy correctness thus needs to be applied. These systems are highly scalable, efficiently distributing learning over a single GPU or even across machines.

In this thesis, our algorithmic backbone is built on top of the recurrent A3C baseline (Mnih et al., 2016), enhanced with GAE (Schulman, Moritz, et al., 2015) and batch-training (Clemente et al., 2017). For our additional investigation on the effects of experience replay, we directly adopt ACER models from (Z. Wang et al., 2016) while keeping the recurrent modules and batch-training.
3.2 Continuous Action Space and Discretization

One of our goals is to identify the best treatment of the CarRacing action space, whose default form is composed of 3 continuous factors, namely turning, acceleration and braking. Continuous action space has posed much challenges to many classic RL algorithms targeting at finite state and action spaces e.g. SARSA as it is unfeasible to learn infinitely many state-action value pairs (Santamaría, Sutton, & Ram, 1997) or thoroughly search the space for an optimal real-valued action (Lazaric, Restelli, & Bonarini, 2008). A strategy to tackle continuous action space in traditional RL is to discretize it into finite pieces. For example, the wire fitting (Baird & Klopf, 1993) approach learns a finite set of state-action value pairs and then interpolate for an interlying action. Continuous-action Q-learning (Millán, Posenato, & Dedieu, 2002) incrementally learns a set of regions to cover the input space and each region is associated with a finite number of most probable discrete actions. However, for such methods, there is always a sensitive trade-off between the granularity of the discretization and the cost in computation.

Alternatively, with the aid of policy gradient (Williams, 1992), continuous policy can be directly output deterministically (Silver et al., 2014) or as a Gaussian unit (Degris, Pilarski, & Sutton, 2012), which has proven to be especially effective in the context of DRL (Lillicrap et al., 2015; Schulman, Levine, et al., 2015). Note that in most of these cases, each continuous factor, e.g. turning, acceleration and braking for CarRacing, is assumed to be conditionally independent from the rest given the state information. However, direct continuous policy learning still struggle with many issues, such as the lack of multi-modality in output policy and optimization hurdles. Therefore, many notable applications discretize continuous control in one of the following two ways: discretize each factor individually and retain the conditional independence assumption as done in the OpenAI-5 Dota Bolt (openAI, 2018) or manually assemble a finite set of policy combinations from expert knowledge (You et al., 2017). In this thesis, we strive to comprehensively compare the 3 different action treatments of CarRacing under A3C.

3.3 Exploration in an Environment

One central problem in RL that distinguishes itself from other areas of ML is the trade-off between exploitation and exploration (Kaelbling, Littman, & Moore, 1996). An agent naturally inclines to take measure following the current best model (exploitation), but in order to break out of a local optimal policy, it must learn to venture into less visited states which could potentially yield a reward higher than expected (exploration). There are multiple options to encourage exploration. Another goal of this thesis is to evaluate several important exploration mechanisms for A3C and ACER on CarRacing.

Action-dithering scheme is the most basic exploration mechanism. An example is $\epsilon$-greedy, where a random variable is uniformly sampled in $[0, 1]$ at every step, deciding whether to execute a random move (if $< \epsilon$) or act greedily (Mnih et al., 2015). For continuous control, deterministic policy methods (Lillicrap et al., 2015) add additional noise such as sampled from the Ornstein-Uhlenbeck process.
to the output action during training, in order to accommodate the presence of inertia; exploration for Gaussian policies can be directly done through sampling from the Gaussian distribution (Degris et al., 2012). Conventionally, discrete action space actor-critic methods employ Boltzmann Exploration (Kaelbling et al., 1996) that samples the next action from the softmax probability distribution (Bishop, 2006) given the current state of the environment.

Another line of DRL-specific exploration techniques leverages stochastic neural networks (SNNs) (Turchetti, 2004). The SNNs can be equipped with either randomized weights, such as in the case of NoisyNet (Fortunato et al., 2017), or activations, such as in the case of Stochastic A3C (Shang et al., 2018), both of which can be seamlessly applied to A3C and ACER. We will later on evaluate exploration with Boltzmann Exploration alone versus in conjunction with one of these SNNs.

Although not studied in this thesis, another important type of exploration is driven by curiosity (Chentanez, Barto, & Singh, 2005). That is, the agent is explicitly encouraged e.g. through pseudo-reward to gather information of a state which has not been visited often enough (Schmidhuber, 1991). However, it requires domain expertise to choose an appropriate enticement.

3.4 Algorithms for Driving Simulators

There are multiple solutions to learn an autonomous driving agent with a driving simulator. One of them is to formulate the task as a classification problem. In (Heylen et al., 2018), the authors manage to learn the required steering angle in a supervised manner for an agent to drive in the Udacity simulator and GTA V. An alternative utilizes tools in RL, among which actor-critic methods such as DDPG and A3C (Khan & Elibol, n.d.; Jang, Min, & Lee, n.d.; Mnih et al., 2016; Sharma, Lakshminarayanan, & Ravindran, 2017) have been especially favored thanks to their empirical performances. For example, in (Mnih et al., 2016), an A3C agent is able to achieve 75% to 90% the performance of a human tester on TORCS. These works also have served as inspiration for us to choose A3C as a starting point to solve the CarRacing environment.

4 Methodology

In this section, we start with the mathematical formulation and training objectives of our foundation algorithm A3C (Section 4.1). Next, we describe the construction of the policy networks associated with each candidate action space (Section 4.2). Then Section 4.3 details the procedures of the 3 exploration mechanisms to be compared in experiments. Finally, we introduce ACER, an additional actor-critic algorithm studied in this thesis to assess the effectiveness of integrating experience replays for sample efficiency.
4.1 Mini-batch LSTM Advantage Actor-Critic Methods

The fundamental framework behind our work is the model-free, on-policy, asynchronous advantage actor-critic (A3C) (Mnih et al., 2016). In A3C, multiple active agents interact with their own independent environments at the same time. The gradients generated from each agent then contribute to the optimization of a shared model that approximates both the policy and value functions. As aforementioned, the advantage policy gradients manage to better trade-off bias and variance. In addition, the asynchronous setup of A3C naturally decorrelates training data, hence eliminates the need of the memory-demanding replay buffer and stabilizes the training.

The most common A3C architecture encodes the raw observations with a DNN, frequently a Convolutional Neural Network as also done in our case, and the subsequent policy and value networks both take on the processed features as inputs. As explained in Section 2.2, a recurrent module on top of the encoding DNN can engage longer temporal dependency. Our model therefore also uses an LSTM on top of the CNN encoder and is referred to as A3C-LSTM. More concretely, A3C-LSTM (Figure 2) extracts features from raw observations via the CNN and then pass the features through the LSTM to obtain an output representation $h_t$, where $t$ is the time step, that is,

$$h_t = f_{\text{LSTM}}(\text{CNN}(o_t), h_{t-1}).$$

There are two parallel Fully-Connected (FC) layers taking $h_t$ as input,

$$[f_{\text{FC1}}(h_t), f_{\text{FC2}}(h_t)],$$

so that we can make one of the FC layers stochastic for exploration as done in (Shang et al., 2018). Finally, this concatenation is sent to the value and policy networks, for which we now elaborate the training objectives.

**Value Learning.** The value network approximates the state-value $V_t$ by minimizing the $k$-step TD error, i.e. the difference between the output $V_t$ and the estimated $k$-step return $R_t^k$, where $k$ is at most the number of rollout steps for the LSTM, denote as $k_{\text{max}}$. Note that the TD error minimization term here precisely approximates the advantage, $A_t = Q_t(a_t, s_t) - V_t(s_t)$, where $a_t$ is the action taken and $s_t$ is the state at time $t$, because $R_t^k$ is an unbiased estimation of $Q_t(a_t, s_t)$ (UC-Berkeley, 2017). Mathematically, the objective becomes

$$(A_t)^2 = [V_t - R_t^k]^2 = [V_t - (r_{t+1} + \gamma r_{t+2} + \cdots + \gamma^k V_t(s_{t+k}))]^2,$$

recall that $r_t$ is the reward at time $t$, and $\gamma$ is the discount factor.

**Policy Learning.** The original A3C model directly applies the advantage to the score function for policy gradient, that is

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_t}[\nabla_\theta \log \pi_t(s_t, a_t) A_t],$$

1More precisely, here $a_t$ is in fact the observation; recall that due to the LSTM module, the original MDP assumption does not hold exactly and $h_t$ resembles a belief state of a POMDP.
where \( \pi_t \) is the policy at time \( t \). We indeed use this formulation later on for ACER. However, for A3C models, the advantage function is replaced with a more advanced variance reduction baseline, the general advantage estimator (GAE) [Schulman, Moritz, et al. 2015], which further reduces the variance at the cost of introducing some bias. Inspired by TD(\( \lambda \)) [Sutton & Barto 1998], GAE is the exponentially-weighted average over advantage estimators with different horizons. Mathematically, we can first define the \( i \)th horizon advantage estimator

\[
\hat{A}_i^t = -V_t(s_t) + r_t + \gamma r_{t+1} + \cdots + \gamma^{i-1} r_{t+i-1} + \gamma^i V_t(s_{t+i}),
\]

and the resulting GAE with weighting parameter \( \lambda \) is

\[
\hat{A}_{\text{GAE}}^t = (1 - \lambda)(\hat{A}_1^t + \lambda \hat{A}_2^t + \cdots + \lambda^{k-1} \hat{A}_k^t).
\]

Besides policy gradient, the policy network can further include an entropy regularization term \( H(\pi) \) to avoid spiking activation around a single action, which would hinder exploration. Algorithm 4.4 presents the pseudo-code to provide a clear coverage of the core A3C-LSTM baseline, on top of which we will build up the other comparative models.

Another divergence between our A3C-LSTM training from the original A3C is the usage of mini-batch to enable more efficient, stable training on GPU. In particular, we follow the protocol from [Shang et al. 2018], similar to batched A2C [J. X. Wang et al. 2016]: for each time step, we wait for each agent to complete an action, collect the observations from each environment, combine them into a mini-batch, which is then processed on the GPU. Such synchronous mini-batches help avoid stale gradients [Chen, Pan, Monga, Bengio, & Jozefowicz 2016] while avoiding any complex off-policy corrections, but with the downside of having to wait on the slowest agent.

### 4.2 Policy Networks for Various Action Spaces

Recall one of the areas we strive to optimize for A3C on CarRacing is the design of action space. Prior works suggest three candidates: continuous control, discrete factorized actions, and a finite set of handcrafted policies. For each of them, the construction of the corresponding policy network slightly differs.

The default CarRacing action space is continuous. To be computationally savvy, following the protocol from many existing works [C. Wu et al. 2018], we assume conditional independence of each factor given the state information and represent the (stochastic) policies as Gaussian distributions with diagonal covariance. In other words, for each action factor, the policy network outputs a mean and a variance:

\[
\pi(a^1, a^2, a^3|s_t) = \prod \mathcal{N}(\mu_i^t, (\sigma_i^t)^2),
\]

and the entropy term is thus calculated from the variance,

\[
H(\pi) = \sum H(\mathcal{N}(\mu_i^t, (\sigma_i^t)^2)) = \sum \frac{1}{2} \ln[2\pi e(\sigma_i^t)^2].
\]
By discretizing each continuous factor into categorical values while retaining the conditional independence assumption, we arrive at the discrete factorized action space. Its policy network simply replaces the Gaussian policy from the discrete case with a Multinomial one. However, the resulting entropy term can become computationally intractable,

\[ H(\pi_t) = \sum_{i \in A_1} \sum_{j \in A_2} \sum_{k \in A_3} \pi(i|s_t) \pi(j|s_t) \pi(k|s_t) \log \nabla \pi(i|s_t) \pi(j|s_t) \pi(k|s_t). \]

where \( A_i \) is the possible action values for the \( i \)th factor. Fortunately, we discover empirically that in this case A3C performs most optimally with no entropy regularization. Thus, for the rest of the thesis, we omit this term for all experiments on the discrete factorized space.

In an effort to overcome the potential limitations from the conditional independence assumption, we also consider action spaces composed of finite sets of manually defined action configurations, Then the formulation of the policy network and the entropy term is identical to A3C with a standard discrete action space, where

\[ H(\pi_t) = \sum_{a \in A} \pi(a|s_t) \log \nabla \pi(a|s_t). \]

4.3 Exploration for A3C

Another primary focus in this thesis is an empirical comparison among different means to explore. The most basic method is action space noise–or action space “dithering”, such as sampling from the Gaussian policy for continuous actions or Multinomial distribution (i.e. Boltzmann Exploration) for discrete actions. A more advanced venue adds SNN components on top of the action dithering. Here we opt to study the effects from 2 types of SNN-based exploration. The first one, NoisyNet (Fortunato et al., 2017), utilizes stochastic weights. We refer it as A3CNN-LSTM. Concretely, it replaces the linear layers within the policy and value networks with the Noisy linear layers whose weights are parametrized by Gaussian units and the variances are also learned on the fly:

\[ y = wx + b, \text{ where } w \sim \mathcal{N}(\mu_w, (\sigma_w)^2), b \sim \mathcal{N}(\mu_b, (\sigma_b)^2). \]

The second one (Shang et al., 2018), instead of sampling weights, samples intermediate activations in the hope of propagating more complex perturbation to the value and policy networks. We refer it as SA3C-LSTM. Specifically one of the FC channels after LSTM now outputs the mean of a Gaussian distribution with some fixed hyperparameter variance and the activation is thus drawn from this distribution

\[ z_t \sim \mathcal{N}(f_{FC2}(h_t), \sigma^2). \]

Figure 2 displays the stochastic components for A3CNN-LSTM and SA3C-LSTM.
Figure 2: from left to right, A3C-LSTM, SA3C-LSTM and A3CNN-LSTM where the red square stands for the stochastic activations and the red arrows for the insertion of parametric noise.

4.4 Advantage Actor-Critic with Experience Replay

Efficient Atari game simulation has become readily available with today’s computing power, which in turn has catalyzed important research progress in RL. But once the environment engages more realistic elements, simulation quickly becomes computationally expensive. For this reason, techniques empowering sample efficiency, such as learning from experience replay, have attracted much attention. Due to the nature of policy gradients, it is not possible to directly update the model with experiences generated by an older model. Therefore when a sample generated by an older policy is used to update the current policy, its impact on the current policy is weighted by how far the old and the current policy are apart from each other via e.g. importance sampling.

Building on top of this intuition, ACER (Z. Wang et al., 2016) extends A3C to use experience replay for better sample efficiency, while leveraging Retrace (Munos, Stepleton, Harutyunyan, & Bellemare, 2016) to correct off-policy bias in Q value estimation and trust-region optimization over the policy gradients (Schulman, Levine, et al., 2015). Here we perform experimental assessment on ACER to verify the effectiveness of the experience replay and to test whether additional exploration techniques can further speed up the convergence. Full details of ACER training is in Algorithm 2.

5 Environment

Our CarRacing environment is provided by the Box2D module interfaced with the OpenAI Gym platform (Brockman et al., 2016). In this simple environment, the agent controls a car and has to follow the road while going as fast as possible. Approximately 300 tiles are placed over the track. For every tile the car touches, the agent receives a reward of 1000/N where N is the number of tiles. For every
Initialize network parameters $\theta$;

for $k = 0, 1, 2, \cdots$ do

Clear gradients $d\theta \leftarrow 0$;

Simulate under current policy $\pi_{t-1}$ until $t_{\text{max}}$ steps are obtained, where,

$h_i = f_{\text{LSTM}}(\text{CNN}(o_t), h_{t-1}), w_t = f_{\text{FC1}}(h_t), k_t = f_{\text{FC2}}(h_t), V_t = f_v(w_t, k_t),$

$\pi_t = f_p(w_t, k_t)$, $t = 1, \cdots, t_{\text{max}}$;

$R = \begin{cases} 0, \text{ if terminal} \\ V_{t_{\text{max}}+1}, \text{ otherwise} \end{cases}$;

for $t = t_{\text{max}}, \cdots, 1$ do

$R \leftarrow r_t + \gamma R$;

$A_t \leftarrow R - V_t$;

Accumulate gradients from value loss: $d\theta \leftarrow d\theta + \lambda \frac{\partial A_t}{\partial \theta}$;

$\delta_t \leftarrow r_t + \gamma V_{t+1} - V_t$;

$A_t \leftarrow \gamma \tau A_{t-1} + \delta_t$;

Accumulate policy gradients with entropy regularization:

$d\theta \leftarrow d\theta + \nabla \log \pi_t(a_t) A_t + \beta \nabla H(\pi_t)$;

end

end

Algorithm 1: Core A3C-LSTM

second the agent uses to complete the track, it receives a negative reward of $-5$. These rewards are given per step the agent takes, and thus depend on the frame rate (measured by frames per second, FPS) the environment is running at.

The game is deemed to be completed when an agent consistently scores over 900. The unpublished OpenAI oracle scores 837 on average over 100 runs.

An observation given by the environment consists of a $96 \times 96$ image, containing a top-view of the car, its current surroundings, and an assemble of engine indicators such as ABS, gyro and speed (see figure 1(a) for an example). These observations are preprocessed in the following way: first rescale the pixel values from the 0 to 255 range to from 0 to 1; then subtract the channel-wise mean and divide the standard deviation—the mean and standard deviation are estimated from 1000 randomly generated frames; finally, resize the image to $80 \times 80$.

The environment has been altered from the original one in three significant ways. First, the start of the environment first shows an overall top view of the entire track, while slowly zooming in during the first second of a trial. As this period is irrelevant to learning, it is skipped. In addition, the FPS has been reduced from 50 to 15, to imitate the deterministic frame skipping while simultaneously skipping additional rendering of the intermediate frames to save computational cost. Finally, we redefine the termination criterion and stop the simulation if the car has not touched a tile in $\frac{2}{3}$ of a second.
Initialize network parameters $\theta$;
Initialize average network parameters $\theta_a$;
for $k = 0, 1, 2, \ldots$ do
  Clear gradients $d\theta \leftarrow 0$;
  if on policy then
    Simulate under current policy $\pi_{t-1}$ until $t_{\text{max}}$ steps are obtained,
    where, $h_t = f_{\text{LSTM}}(\text{CNN}(o_t), h_{t-1})$, $w_t = f_{\text{FC1}}(h_t)$, $k_t = f_{\text{FC2}}(h_t)$,
    $Q_t = f_v(w_t, k_t)$, $\pi_t = f_p(w_t, k_t)$, $V_t = Q_t \cdot \pi_t$, $t = 1, \ldots, t_{\text{max}}$, set $\rho_t = 1$;
  else
    Retrieve experience $(o_1, \ldots, o_{t_{\text{max}}}, r_1, \ldots, r_{t_{\text{max}}}, a_1, \ldots, a_{t_{\text{max}}}, \mu_1, \ldots, \mu_{t_{\text{max}}})$ from the memory buffer, compute
    $h_t = f_{\text{LSTM}}(\text{CNN}(o_t), h_{t-1})$, $w_t = f_{\text{FC1}}(h_t)$, $k_t = f_{\text{FC2}}(h_t)$, $Q_t = f_v(w_t, k_t)$, $\pi_t = f_p(w_t, k_t)$,
    $V_t = Q_t \cdot \pi_t$, $\rho_t = \pi_t / \mu_t$, $t = 1, \ldots, t_{\text{max}}$.
  end
  $R = \begin{cases} 0, & \text{if terminal} \\ V_{t_{\text{max}}+1}, & \text{otherwise} \end{cases}$
  for $t = t_{\text{max}}, \ldots, 1$ do
    $R \leftarrow r_t + \gamma R$;
    $A_t \leftarrow R - V_t$;
    $L_V \leftarrow \frac{1}{2}(R - Q_t(a_t))^2$;
    Calculate advantage policy gradients: $g \leftarrow \nabla \log \pi(a_t) A_t$;
    Calculate KL gradients: $k \leftarrow \nabla D_{KL}(\pi_t \parallel \pi_\theta)$;
    Accumulate trust region policy gradients with entropy regularization:
    $d\theta \leftarrow d\theta + \nabla \theta (g - \max(0, \frac{k^T g - 1}{\|k\|^2} k) + \beta \nabla H(\pi_t))$;
    Accumulate gradients from value loss: $d\theta \leftarrow d\theta + \lambda \frac{\partial L_V}{\partial \theta}$;
    Update Retrace target: $R \leftarrow \rho_t (R - Q_t(a_t)) + V_t$
  end
  Update average model parameter: $\theta_a \leftarrow 0.99 \theta_a + 0.01 \theta$
end

Algorithm 2: ACER

6 Experiments and Result Analysis

In the section, we start with introducing experimental setups (Section 6.1),
including the observation encoding module that consists of a CNN followed by an
LSTM and the training hyperparameters. Next, extensive control experiments
are performed to identify the optimal design of the action space (Section 6.2).
After selecting the action space, we move on to investigate different exploration
mechanisms (Section 6.3). Finally, building on top of the intuition from A3C-
LSTM results, we further experiment an asynchronous actor-critic method with
off-policy replay, ACER (Z. Wang et al., 2016), on CarRacing (Section 6.4).
<table>
<thead>
<tr>
<th>layer</th>
<th>input</th>
<th>output size</th>
<th>parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>conv1</td>
<td>observation</td>
<td>32×80×80</td>
<td>32, 5, 1, 2</td>
</tr>
<tr>
<td>conv2</td>
<td>conv1</td>
<td>32×40×40</td>
<td>32, 3, 2, 1</td>
</tr>
<tr>
<td>conv3</td>
<td>conv2</td>
<td>32×40×40</td>
<td>32, 5, 1, 2</td>
</tr>
<tr>
<td>conv4</td>
<td>conv3</td>
<td>32×20×20</td>
<td>32, 3, 2, 1</td>
</tr>
<tr>
<td>conv5</td>
<td>conv4</td>
<td>64×20×20</td>
<td>64, 3, 1, 1</td>
</tr>
<tr>
<td>conv6</td>
<td>conv5</td>
<td>64×10×10</td>
<td>64, 3, 2, 1</td>
</tr>
<tr>
<td>conv7</td>
<td>conv6</td>
<td>64×10×10</td>
<td>64, 3, 1, 0</td>
</tr>
<tr>
<td>conv8</td>
<td>conv7</td>
<td>64×5×5</td>
<td>64, 3, 2, 1</td>
</tr>
<tr>
<td>lstm</td>
<td>conv8</td>
<td>512</td>
<td>1024</td>
</tr>
<tr>
<td>[FC1, FC2]</td>
<td>lstm</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 1: The observation encoding module shared across all models in this thesis. The parameter tuple for convolutional layers corresponds to number of filter, kernel size, stride size and padding size and for LSTM corresponds to number of hidden units.

### 6.1 Model Architecture and Hyperparameters

In our experiments, training is done through mini-batch gradient descent with Adam (Kingma & Ba, 2014) optimizer; due to computational constraint, we set batchsize to 8, i.e. spawning 8 agents with their own environments. We borrow many hyperparameter setups from Shang et al.’s A3C-LSTM model on Atari, where initial learning rate is 0.0005; \(\epsilon = 0.001, \beta_1 = 0.9, \beta_2 = 0.999\) for Adam; gradient clip threshold set at 40; \(\gamma = 0.95, k_{\text{max}} = 20; \lambda = 5\) for the value head objective coefficient; if entropy term is used then its coefficient is \(\beta = 0.01\); rewards clipped between \(-1\) and \(1\); the trace decay parameter for the generalized advantage estimator is \(\tau = 1.0\); for the variance of the stochastic activations, we set \(\log(\sigma^2) = -5\) and for the initial variance of the NoisyNet layers weights, we set \(\sigma = 0.017\). ACER and its variants include the following additional hyperparameters: the replay ratio is 1, replay buffer size 15000 and replay start when buffer reaches 5000, trust decay for TRPO 0.99, trust threshold 1, model averaging ratio 0.99, and importance weight truncation 10.

A3C-LSTM, ACER and their variants in this thesis share the same encoding module architecture to process the raw observations. This module is composed of a CNN then followed by an LSTM. The CNN is a generic stack of convolutional layers with LeakyReLU nonlinearities. The LSTM has a single layer with 1024 hidden units. The LSTM output \(h_L\) is then fed into two parallel FC layers and their outputs, each of dimension 512, are concatenated back together. This configuration allows us to make one of these pathways stochastic as an additional way to encourage exploration later on. The detailed architecture is in Table 1.

### 6.2 Discover the Optimal Action Space

In this section, we endeavor to pinpoint the optimal action space design for CarRacing with A3C-LSTM. Section 5 previews the three different types of candidate action spaces. For the continuous space, we follow the default setup
<table>
<thead>
<tr>
<th>Value</th>
<th>Turning</th>
<th>Acceleration</th>
<th>Braking</th>
<th>Variance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linear(1024,1)</td>
<td>Linear(1024,1)</td>
<td>Linear(1024,1)</td>
<td>Linear(1024,3)</td>
<td></td>
</tr>
<tr>
<td>+ tanh</td>
<td>+ sigmoid</td>
<td>+ sigmoid</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 2: Value and policy networks for continuous action space.

<table>
<thead>
<tr>
<th>Value</th>
<th>Turning</th>
<th>Acceleration</th>
<th>Braking</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linear(1024,1)</td>
<td>Linear(1024,7)</td>
<td>Linear(1024,4)</td>
<td>Linear(1024,2)</td>
</tr>
<tr>
<td>+softmax</td>
<td>+softmax</td>
<td>+softmax</td>
<td></td>
</tr>
</tbody>
</table>

Table 3: Value and policy networks for discrete factorized action space.

<table>
<thead>
<tr>
<th>Value</th>
<th>Policy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linear(1024,1)</td>
<td>Linear(1024,N)+softmax</td>
</tr>
</tbody>
</table>

Table 4: Value and policy networks for the finite set of actions, note that N is the cardinality of the set.

To range turning from $[-1,1]$, acceleration $[0,1]$ and braking $[0,1]$ and use 3 independent FC layers to output these policies with an additional FC layer to model the variances (Table 2), referring to the final model cA3C-LSTM. For the discrete factorized space, we divide turning into $\{\pm0.75, \pm0.5, \pm0.25, 0\}$, acceleration to $\{0, 0.4, 0.7, 1\}$, and braking to $\{0, 0.3\}$. The policy network outputs each factor with an independent FC layer (Table 3) and the final model is donated fA3C-LSTM. Lastly, for the handcrafted action sets, we examine several options in the next paragraph and the policy network is treated in the same way as a finite discrete action space, i.e. a single FC layer followed by softmax (Table 4).

Based on our prior knowledge of driving, we propose several options to manually orchestrate finite sets of action factor combinations. The first set, $A_1$, granulates on top of 4 action types, namely turning in either direction, acceleration and braking, to allow gentle to extreme movements, amounting to 12 actions in total; $A_2$ follows similar logic as $A_1$ but with 8 more actions that allow acceleration while turning and more gentle braking. Also similarly to $A_1$, $A_3$ again has 12 actions but pays more attention to driving safely at lower speeds. Finally, as a control setup to compare with the discrete factorized action space later on, $A_4$ contains all potential combinations from the factor discretization for fA3C-LSTM, hence there are $7 \times 4 \times 2$ total actions for $A_4$. Table 5 summarizes the actions in each set. We then experimentally compare $A_1$ to $A_4$. Table 6 lists the final median total return averaging over 10 validation runs. Even though $A_4$ is in theory equivalent to the discrete factorized action spaces, we observe that training cannot be initiated, likely due to the high dimensionality in the softmax layer. Next, to better compare the other 3 sets, we plot the median validation curves during training in Figure 3(a). Although these action sets share the same high-level intuition, there is a clear discrepancy in their performances, indicating the importance of possessing a precise prior knowledge so as to arrive at the most suitable action options. Between $A_1$ and $A_2$, the former gives a slightly superior median score while the latter presents a more desirable median learning curve yet with more actions. To conclude, effective construction of a finite action set can be non-trivial and require additional trial-and-errors. In our case, all things considered, we decide to use $A_1$ for the rest of mA3C-LSTM experiments.
In other words, the discrete factorization structure is capable of achieving satisfactory exploration without aid of other tricks, making it an even more attractive alternative considering that there is no entropy regularization term for fA3C-LSTM. We will return to some insight we can acquire from the success of fA3C-LSTM. We will return to these questions in the Discussion section.

6.3 Exploration Mechanisms

Our second suits of experiments aim at evaluating exploration mechanisms pairing with the best performing A3C-LSTM model from Section 6.2, fA3C-LSTM. As illustrated in Section 6.2, we compare 3 exploration setups, namely the default Boltzman exploration (fA3C-LSTM) – i.e. sampling each action factor from its corresponding softmax, Boltzman exploration in combination with stochastic units (fSA3C-LSTM) or NoisyNet layers (fA3CNN-LSTM). We plot the median validation curves during trainint in Figure 3(c).

Now, we proceed to compare cA3C-LSTM, fA3C-LSTM and mA3C-LSTM. Figure 5(b) shows the median validation curves during training in terms of total return; Table 7 displays the median final scores averaging over 10 validation runs for each action space, where only fA3C-LSTM manages to solve the environment with a median score over 900. For the other models, mA3C-LSTM performs reasonably, whereas cA3C-LSTM barely trains. Empirically, we reach the conclusion that fA3C-LSTM is optimal for CarRacing, whereas agents trained from cA3C-LSTM behave poorly. These conclusions in turn raise additional questions (1) why continuous action space is suboptimal for CarRacing and (2) if there is some insight we can acquire from the success of fA3C-LSTM. We will return to these questions in the Discussion section.

<table>
<thead>
<tr>
<th>$A_1$</th>
<th>$A_2$</th>
<th>$A_3$</th>
<th>$A_4$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$(\pm 1.05, 0.05), (\pm 0.7, 0.05, 0), (\pm 0.3, 0.05, 0)$</td>
<td>$(\pm 1.05, 0.0), (\pm 0.7, 0.05, 0), (\pm 0.3, 0.05, 0)$</td>
<td>$(\pm 1.05, 0.0), (\pm 0.7, 0.05, 0), (\pm 0.3, 0.05, 0)$</td>
<td>$(a^1, a^2, a^3)$, where $a^1 \in {0.05, 0.25, 0.75}, a^2 \in {0, 0.4, 0.7, 1}, a^3 \in {0, 0.3}$</td>
</tr>
</tbody>
</table>

Table 5: Actions for each handcrafted action set.

<table>
<thead>
<tr>
<th>$A_1$</th>
<th>$A_2$</th>
<th>$A_3$</th>
<th>$A_4$</th>
</tr>
</thead>
<tbody>
<tr>
<td>865</td>
<td>844</td>
<td>689</td>
<td>89</td>
</tr>
</tbody>
</table>

Table 6: Different Handcrafted Sets

<table>
<thead>
<tr>
<th>cA3C</th>
<th>fA3C</th>
<th>mA3C</th>
</tr>
</thead>
<tbody>
<tr>
<td>404</td>
<td>900+</td>
<td>865</td>
</tr>
</tbody>
</table>

Table 7: Different Action Spaces

In terms of final results, all three models manage to solve the environment with final scores over 900. Table 8 shows the median iterations required to solve the environment. Both Table 8 and Figure 6.3 show that all three converges at comparable pace, while fSA3C-LSTM enjoys a slight speed advantage over the other two. These results in fact contradict with our initial expectations where we project more advanced exploration techniques could be more helpful, especially considering that there is no entropy regularization term for fA3C-LSTM. In other words, the discrete factorization structure is capable of achieving satisfactory exploration without aid of other tricks, making it an even more attractive action space design.
Table 8: Different Explorations

<table>
<thead>
<tr>
<th>fA3C</th>
<th>fSA3C</th>
<th>fA3CNN</th>
</tr>
</thead>
<tbody>
<tr>
<td>13.1K</td>
<td>11.8K</td>
<td>17.4K</td>
</tr>
</tbody>
</table>

Table 9: ACER and Variants

<table>
<thead>
<tr>
<th>ACER</th>
<th>SACER</th>
<th>ACER-NN</th>
</tr>
</thead>
<tbody>
<tr>
<td>891</td>
<td>900+</td>
<td>859</td>
</tr>
</tbody>
</table>

Figure 3: Moving average with $n = 5$ of the median validation score out of three validation runs per model during training. Horizontal axis is the number of iterations and vertical the total return.

6.4 Actor-Critic with Experience Replay

CarRacing is a light-weighted environment. But once adapting a more complex simulator with expensive rendering, sample efficiency will become a critical concern for selecting algorithms. To this end, we conduct additional experiments to assess ACER, an A3C inspired algorithm that is supposed to improve sample efficiency by integrating off-policy replay. Since Retrace, a bias-correction technique employed by ACER, requires state-action pair evaluations, we cannot easily apply discrete factorized action space. Therefore, we use $A_1$ from mA3C-LSTM as discrete action space instead. Since the usage of $A_1$ can diminish exploration comparing with the discrete factorized space, we also augment the stochastic units or weights to ACER, denoted SACER and ACER-NN respectively. Among ACER and its variants, SACER appears superior to the other two both in terms of convergence speed (Figure 3(d)) and final performance (Table 9). We indeed also observe a boost in sample efficiency, where SACER converges around 5K, more than twice as fast as fSA3C-LSTM (~11K). Despite the promising initial results, there are still questions remaining such as the role played by TRPO and potential side effects from off-policy training. We will carry out more analysis along this direction in Section 7.
7 Discussion

In this section, we look deeper into our empirical results so as to gain a more fundamental understanding of the following questions: why is continuous space suboptimal for our problem? what action combinations do fA3C-LSTM learn? which component of ACER contributes the most to its enhanced performance?

7.1 Explaining the Deficiency of Continuous Space

In many other machine learning applications, from age classification to image generation [Rothe, Timofte, & Van Gool, 2015; Oord, Kalchbrenner, & Kavukcuoglu, 2016], it is a common practice to discretize continuous space. One intuition behind it argues that classification loss such as cross-entropy loss can send out more clear training signal than mean square error loss, especially if the outcomes show signs of clustering. We also witness similar trends qualitatively in our experiments. Figure 4 visualizes saliency map (Greydanus, Koul, Dodge, & Fern, 2017) for the learned value and policy networks from both cA3C-LSTM and fA3C-LSTM at a given frame where the car is about to turn. On a high-level, the saliency map highlights the areas that influence the final outputs the most, red for policy and blue for value. There is a very notable difference between the policy saliency between the two models, where the former one only pays limited attention to the road and almost no attention to the engine indicator, the opposite from fA3C-LSTM. Explicitly, it means masking any regions from the input does not cause much perturbation to the policy when trained with continuous space as targets, likely because the real consequence from a small change in action, e.g. no braking \(a_3 = 0\) versus braking \(a_3 = 0.3\), can be very substantial but numerically too subtle for the network to capture during optimization on the continuous spectrum. Subsequently, this issue may inhibit the model to adventurously explore. To verify, we collect 20 trajectories by operating cA3C-LSTM agents on different tracks and group together the actions based on fA3C-LSTM’s discretization; among the 1212 actions taken, 667 fall into the category \((0, 0.7, 0)\), 255 into \((-0.25, 0.7, 0)\), 155 into \((-0.5, 0.7, 0)\), 128 into \((0.25, 0.7, 0)\) and last 7 to \((-0.75, 0.7, 0)\). Clearly, the model fails exploring more options for acceleration and braking as approximately the same value is used every time for both of them, whereas the exploration does relatively fine with steering, where the reward signal is stronger.
Similar saliency differences are also spotted for the value functions, which is expected as the value and policy learning are intertwined for actor-critic methods. If we plot the approximated value for each time step over an entire episode for both models (Figure 5), it is even more clear that cA3C-LSTM cannot deduce the value of different states while fA3C-LSTM understands when the environment is about to terminate.

7.2 Looking into the Learned Factorized Action Space

In spite of the clear empirical superiority of discrete factored action space, it still assumes conditional independence among each action factor given the state information. This assumption can potentially overlook important intrinsic interplays over the factors (C. Wu et al., 2018). Moreover, certain factor combinations can even be invalid, such as heavy braking and accelerating at the same time. For these reasons, we scrutinize the learned action combinations under the conditional independence assumption, hoping to comprehend the effect of structuring actions in factorization and extrapolating potential pattern to retrospectively improve the training.

Similar to the procedure in Section 7.1, we collect trajectories from 20 independent runs executed by fA3C-LSTM, fSA3C-LSTM, and A3CNN-LSTMs agents. In addition, to see if models trained with different random seeds produce consistent results in terms of action factor compositions, for each the aforementioned agents, we repeat the same procedure over 2 independently trained models on 2 different random seeds.

We first compare the action distributions from the independently trained models of the same algorithm, where we combine all the action factor combinations from both models, calculate the number of appearances of each combination, and plot the resulting histograms (Figure 7(a), 7(b), 7(c) for fA3C-LSTM, fSA3C-LSTM and A3CNN-LSTMs respectively). To our surprise, the distributions from the 2 fA3C-LSTM runs are very different, those from the 2 fSA3C-LSTM reach a consensus for the most part but disagree on the actions concerning left turns, and those from A3CNN-LSTM do align well. We conjecture the alignment for the models with stochastic weights or units stems from the reduced model uncertainty, without which the action space distribution is much less regularized and displays more random strategies as in the case of fA3C-LSTM.
Moving on, we compare the action distributions across models trained with different algorithms. Specifically, we compare between the actions associated with the best performing fA3C-LSTM model and the best performing fSA3C-LSTM model (Figure 7(d)). Although in terms of final agent performance, these two models are very comparable but their final action distributions substantially differ from one another—not only that they tend to select different action combinations but also the fSA3C-LSTM model holds a more fine-grained selection.

Despite the differences, we also observe a universal trend across most of these models: the majority of the actions correspond to going straight, then there are approximately equal amount of actions dedicated to steering left and right and a few to braking. Therefore, in a naive effort to make use of this observation, we train a new mA3C-LSTM model with the 7 action combinations appearing in the fA3C-LSTM. However, after 15K iterations, none of the 3 runs manage to solve the environment or even score over 600. This result implies that there are optimization advantage of using factorized action space and that one needs a more carefully-thought method to distill knowledge from the learned factorized action space in order to retrospectively improve training, which we reserve for future work.

7.3 Understanding ACER Performance

ACER has shown promising results in improving sample efficiency (Section 6.4). Its more advanced version, SACER, with stochastic units for better exploration, outperforms ACER in terms of agent quality, training stability and convergence speed. We henceforth focus on this variant here. Since off-policy gradients have
been known to suffer from training instability (Sutton, Mahmood, & White, 2016), it is important to identify the balance point to prevent excessive replay usage from negatively influencing the learning. For the experiments in Section 6.4, we set the replay ratio to be 1—meaning approximately half of the training is sampled from off-policy—and here we increase the ratio to 2 to see if training continues to improve. Additionally, since SACER employs a more advanced policy gradient TRPO—partially to alleviate the training instability from experience replays, it is also important to tease out the contributions from TRPO when evaluating the boosted performance comparing to A3C-LSTM baselines. To do so, we run an on-policy version of SACER. The median validation curves for SACER-OP (on-policy), SACER (replay ratio 1) and SACER-R2 (replay ratio 2) are compared in Figure 6. We confirm that although TRPO indeed positively impact the training, reflected by faster convergence from SACER-OP than A3C-LSTM, the experience replay certainly plays an essential as well, especially during the later half of the training where the replay is of higher quality. Nevertheless, an increased replay ratio not only fails exerting further improvement but moreover drags the performance below even SACER-OP, which shows the importance of carefully balancing on- and off-policy training in practice.

8 Conclusion

The goal of this thesis is to realize effective applications of advantage actor-critic methods to the CarRacing environment. In particular, we focus on investigating and optimizing 3 essential components, namely the action space, the exploration mechanisms and the integration of off-policy training.

Through extensive experimental efforts, we found that a discrete factorized action space performs the best out of all tested action spaces, although similar models with the same factor structure can exhibit diverse final action combination outcomes. Manually designed action sets have the potential to reach a similar performance level but demand expert knowledge—which can be hard to acquire for more complex tasks. Meanwhile, continuous space functions very suboptimally, potentially due to its hindered optimization and exploration.

Additional encouragement to explore via stochastic weights or units is, empirically found, not necessary for the factorized action spaces. In the case of ACER where a handcrafted discrete action set is used, the use of stochastic units does boost the performance.

Also, we find that a replay buffer can potentially improve sample efficiency, but care needs to be especially taken in this case since the off-policy training instability can nullify its original advantage. Also note that, due to the replay buffer and TRPO which we have shown is powerful enough to by itself speed up convergence, ACER is more computationally demanding than A3C models.

One immediate future work direction involves integrating the discrete factorized action space to ACER by for instance replacing Q-learning and Retrace with V-learning and V-trace. Another follow-up is to research possibilities of forming a prior over the space of action factors and use the prior to guide learning so as to alleviate shortcomings from the conditional independence assumption. Last
but not least, we hope to transfer the insight gained from this thesis to more complex driving simulation to both validate our findings and to bring us a step closer to achieve a real-world autonomous driving agent.
References


Jang, S. W., Min, J., & Lee, C. (n.d.). Reinforcement car racing with a3c.


