Learning to Perceive in Deep Model-Free Reinforcement Learning

Gonçalo dos Santos Querido

Thesis to obtain the Master of Science Degree in

Information Systems and Computer Engineering

Supervisors: Prof. Francisco António Chaves Saraiva de Melo
Prof. José Alberto Rodrigues Pereira Sardinha

Examination Committee

Chairperson: Prof. Pedro Miguel dos Santos Alves Madeira Adão
Supervisor: Prof. Francisco António Chaves Saraiva de Melo
Member of the Committee: Prof. Alexandre José Malheiro Bernardino

November 2022

This work was created using LaTeX typesetting language
in the Overleaf environment (www.overleaf.com).

Acknowledgments

First and foremost, I want to thank my parents because without them my academic journey would not have been possible. Without your caring and full support, I would not be here today. Thanks for always making me want to do better.

I am extremely grateful to my supervisors, Prof. Francisco Melo and Prof. Alberto Sardinha for their feedback, wisdom, and guidance. I also want to thank them for proposing this really interesting topic that I have been researching in the past year.

I also wish to thank Bernardo, with whom I had the opportunity to discuss some topics of this thesis. Thanks for your help and all your answers. Without them, this thesis would not have been the same.

I would like to acknowledge the lab managers at GAIPS and at RNL for providing all the necessary hardware I had the opportunity to use during this thesis, and for your availability to assist me when there was a problem.

Last, but not least, I want to give a special thanks to the friends I have made during this journey in Técnico. Thanks Raquel, Tiago, Joana, Catarina, Rafael, Pedro, Isabel and Rodrigo. Thank you for all the moments we shared on this roller coaster that is university. I can definitely say you made it all worth it.

To each and every one of you – Thank you.

Abstract

This work proposes a novel model-free Reinforcement Learning (RL) agent that is able to learn how to complete an unknown task having access to only a part of the input observation. We take inspiration from the concepts of visual attention and active perception that are characteristic of humans and tried to apply them to our agent, creating a hard attention mechanism. In this mechanism, the model decides first which region of the input image it should look at, and only after that it has access to the pixels of that region. Current RL agents do not follow this principle and we have not seen these mechanisms applied to the same purpose as this work. In our architecture, we adapt an existing model called *recurrent attention model* (RAM) and combine it with the *proximal policy optimization* (PPO) algorithm. We investigate whether a model with these characteristics is capable of achieving similar performance to state-of-the-art model-free RL agents that access the full input observation. This analysis is made in two Atari games, Pong and SpaceInvaders, which have a discrete action space, and in CarRacing, which has a continuous action space. Besides assessing its performance, we also analyze the movement of the attention of our model and compare it with what would be an example of the human behavior. Even with such visual limitation, we show that our model matches the performance of PPO+LSTM in two of the three games tested.

Keywords

Reinforcement Learning; Model-free; Attention Mechanism; Hard Attention; Active Perception; Visual Attention;

Resumo

Este trabalho propõe um novo agente de aprendizagem por reforço (AR) sem modelo que é capaz de aprender como completar uma tarefa desconhecida, tendo apenas acesso a uma parte do ambiente. Inspirámo-nos nos conceitos de atenção visual e perceção ativa que são característicos dos seres humanos e tentámos aplicá-los ao nosso agente, criando um mecanismo de atenção seletiva. Neste mecanismo, o agente tem que primeiro decidir qual a região da imagem para onde deve olhar, e só depois é que tem acesso ao conteúdo dessa parte da imagem. Os modelos de AR atuais não seguem este princípio e não encontrámos este mecanismo de atenção aplicado ao mesmo propósito deste trabalho. Na nossa arquitetura, adaptámos um modelo já existente chamado *recurrent attention model* (RAM) e combinámo-lo com o algoritmo *proximal policy optimization* (PPO). Investigamos se um modelo com estas características é capaz de ter um desempenho semelhante aos agentes atuais de AR sem modelo. Esta análise é feita recorrendo a dois jogos do Atari, Pong e SpaceInvaders, que têm um espaço de ações discreto, e ao CarRacing, que possui um espaço de ações contínuo. Para além de medir o seu desempenho, também tentamos compreender qual a evolução do foco de atenção do nosso agente e se esta se compara ao comportamento humano. Mesmo com esta restrição visual, mostramos que o nosso modelo iguala a performance do PPO+LSTM em dois dos três jogos.

Palavras Chave

Aprendizagem por Reforço; Sem Modelo; Mecanismo de Atenção; Atenção Seletiva; Perceção Ativa; Atenção Visual;

Contents

1 Introduction **1**

1.1 Contributions . . . 4

1.2 Organization of the Document . . . 5

2 Background **7**

2.1 Decision Theoretical Models . . . 9

2.1.1 Markov Decision Processes . . . 9

2.1.2 Partially Observable Markov Decision Processes . . . 10

2.2 Reinforcement Learning . . . 11

2.2.1 Model-based Methods . . . 11

2.2.2 Value-based Methods . . . 11

2.2.3 Policy-based Methods . . . 11

2.3 Neural Networks . . . 12

2.3.1 Convolutional Neural Networks . . . 13

2.3.2 Recurrent Neural Networks . . . 14

2.3.3 Autoencoders . . . 14

2.4 Active Perception and Attention Mechanisms . . . 14

3 Related Work **17**

3.1 Regular Reinforcement Learning Models . . . 19

3.1.1 Model-Free Reinforcement Learning . . . 19

3.1.2 Model-Based Reinforcement Learning . . . 21

3.2 Reinforcement Learning Models with Attention . . . 23

3.2.1 Soft Attention . . . 23

3.2.2 Hard Attention . . . 25

3.3 Active Visual Exploration . . . 28

4 Model **31**

4.1 Architecture . . . 33

4.1.1 Glimpse Network . . . 35

4.1.2 Action Network 36
4.1.3 Location Network 36
4.2 Training 37

5 Experimental Evaluation **39**
5.1 Evaluation Process 41
5.1.1 Game Environments 41
5.1.2 State-of-the-art Reinforcement Learning Agents 42
5.1.3 Evaluation Metrics 43
5.2 Base Models Selection 43
5.3 GBAC Performance Analysis 45
5.3.1 Glimpse Sizes Study 45
5.3.2 Glimpse Location Analysis 48
5.4 Base Models Comparison 50
5.5 Hard Attention vs. Soft Attention 53

6 Conclusion **57**
6.1 Future Work 59

Bibliography **61**

A Extended Results **67**

B Algorithms hyperparameters **73**

List of Figures

2.1 MDPs vs. POMDPs. Adapted from Sutton and Barto [1] 9
2.2 MLP with two hidden layers 13

3.1 DQN architecture presented by Mnih et al. [2]. Adapted from Polvara et al. [3] 20
3.2 Flow diagram of Ha and Schmidhuber's agent. Adapted from [4] 22
3.3 DARQN architecture. Adapted from [5] 25
3.4 DRAM architecture represented across several timesteps. Adapted from [6] 26

4.1 Architecture of RAM. Image adapted from the original paper of Mnih et al. [7] 33
4.2 Overview of the architecture of the Glimpse-Based Actor-Critic 34
4.3 Detailed diagram of the Glimpse Network of GBAC 35
4.4 Detailed diagram of the Action Network of GBAC 36
4.5 Detailed diagram of the Location Network of GBAC 37

5.1 Representation of a glimpse with three patches in SpaceInvaders 45
5.2 Heatmaps representating the amount of times during an episode that the agent chose a specific set of image coordinates to be the center of the glimpse. 48
5.3 Example of a 40x40 glimpse with three patches in Pong. 49
5.4 Heatmaps representing an example of the evolution of choices the agent made for the location of its glimpses during training in the CarRacing game. 50
5.5 Evolution of the average return over the last 100 training episodes of Pong across three different runs, for the four models compared. 50
5.6 Evolution of the average return over the last 100 training episodes of SpaceInvaders across three different runs, for the four models compared. 51
5.7 Evolution of the average return over the last 100 training episodes of CarRacing across three different runs, for the four compared models. 52
5.8 Illustration of the 10 patches the soft attention model selected as the most important regions of the frame to look at 54

A.1 Evolution of the average return over the last 100 training episodes of Pong across three different runs, for PPO and PPO+LSTM, using either the full image frame or a resized version. 68
A.2 Evolution of the average return over the last 100 training episodes of SpaceInvaders across three different runs, for PPO and PPO+LSTM, using either the full image frame or a resized version. 68
A.3 Evolution of the average return over the last 100 training episodes of CarRacing across three different runs, for PPO and PPO+LSTM, only using the full image frame. 69

List of Tables

5.1 Comparison between the training and testing performance of PPO and PPO+LSTM, using a resized frame (84x84) of the input and also the full game frame (210x160) in both Atari games 44
5.2 Comparison between the training and testing performance of PPO and PPO+LSTM using the full game image (96x96) in the CarRacing game 44
5.3 Training and testing performance of the best glimpse size for every number of patches tested in Pong and SpaceInvaders. 47
5.4 Training and testing performance of the best glimpse size for every number of patches tested in CarRacing. 47
5.5 Comparison of the performance during training and testing achieved in Pong, by the four tested models. 51
5.6 Comparison of the performance during training and testing achieved in SpaceInvaders, by the four tested models. 51
5.7 Comparison of the performance during training and testing achieved in CarRacing, by the four tested models. 52
5.8 Comparison between the average testing performance of each model during 100 episodes 53

A.1 Training and testing performance of every glimpse size for every number of patches tested in Pong and SpaceInvaders. 70
A.2 Training and testing performance of every glimpse size for every number of patches tested in CarRacing. 71

B.1 List of parameters used in GBAC, in the Atari games and CarRacing, respectively 74
B.2 List of parameters used in the soft attention model of Tang el al. [8], in the Atari games and CarRacing, respectively 75

Acronyms

A3C	Asynchronous Advantage Actor-Critic
AI	Artificial Intelligence
CMA-ES	Covariance-Matrix Adaptation Evolution Strategy
CNN	Convolutional Neural Network
DARQN	Deep Attention Recurrent Q-Network
DDQN	Double Deep Q-Network
DMM	Dynamic Memory Map
DQN	Deep Q-Network
DRAM	Deep Recurrent Attention Model
DRQN	Deep Recurrent Q-Network
FNN	Feedforward Neural Network
GBAC	Glimpse-Based Actor-Critic
LSTM	Long Short Term Memory
MCTS	Monte Carlo Tree Search
MDN	Mixture Density Network
MDP	Markov Decision Process
MLP	Multilayer Perceptron
POMDP	Partially Observable Markov Decision Process
PPO	Proximal Policy Optimization
RAM	Recurrent Attention Model
RL	Reinforcement Learning
RNN	Recurrent Neural Network
VAE	Variational Autoencoder

1

Introduction

Contents

1.1 Contributions 4
1.2 Organization of the Document 5

In our everyday lives, even though we are constantly being flooded with visual stimuli, we do not give the same importance to everything in our field of view. Instead, we focus on small regions that attract us the most. In those moments, we take advantage of a cognitive process called *visual attention* [9]. Unconsciously, we interpret those regions and extract meaning from them using another mental process named *perception* [10]. The combination of both these processes allows us to solve complex tasks because, from all the visual information we receive, we filter the most important to perform our activities and not pay attention to irrelevant elements in our surroundings.

In Artificial Intelligence (AI), a field that has been the target of plenty of research is computer vision [11]. Examples of typical tasks from this field of AI are image classification, object detection, object tracking, and face recognition. In computer vision, the most popular models created take advantage of deep learning [12], using neural networks that receive images as input and process them to learn how to complete their tasks. One technique often used to train those models is Reinforcement Learning (RL).

In RL, an agent interacts with its environment and learns, by trial and error, an action-selection rule (a *policy*) that maximizes the agent's reward over time. This learning technique has two classes of algorithms: model-based and model-free. Model-based agents try to learn a model that represents how their environment works. Then, they can use this model to predict how the environment will react in the future, allowing them to build a plan with the best set of actions to perform. In opposition, model-free agents do not know how their environment behaves and explicitly learn by trial and error.

Current RL models, even though they achieve excellent performance in a broad range of tasks, do not follow the cognitive processes of visual attention and perception typical of humans. For example, when learning to play a video game, RL algorithms typically process the whole input image, giving the same importance to every region of the input game frame. Such design results in models that rely on large Convolutional Neural Networks (CNNs) that process a large number of pixels, making the model take too long to train, requiring high computational power, and potentially limiting their applicability [13]. To keep the training time reasonable, images are often preprocessed to reduce the size of the input, losing some of its details. Using these low-resolution images can hamper the models from completely understanding what is present in their input, which lowers their performance [14].

With these limitations in mind, researchers are taking inspiration from human cognitive skills, such as visual attention and perception, to develop new architectures. In the literature [15–17], we find two types of approaches to implement attention mechanisms:

- **Soft attention**: these models divide their entire input into multiple areas and then search for the region that looks to be the most important for the task.

- **Hard attention**: before having access to a specific region of the input, these models already choose the coordinates where they should look at.

These new architectures allow us to better understand how neural networks make their decisions since we can see which portions of the image they focus on to solve their task. In the case of the second approach, the models do not even need to process the entire input because they discard the information that is not useful.

Unlike soft attention which has already been widely studied, research on hard attention mechanisms is more scarce [17], especially regarding RL agents that are capable of completing complex tasks, such as playing video games. In this thesis we investigate attention and active perception in RL and contribute with a model-free architecture that implements an attention mechanism similar to the one humans have. Applying such a mechanism allows the model to only process the pixels it perceives as the most useful, which makes it much more computationally efficient and able to use the original images without resizing them.

In this work, we address the following research questions:

- Is it possible to attain state-of-the-art performance in complex control tasks with limited (but active) perception?
- Is there any similarity between the attention movements of the model and human behavior?
- Is one type of attention mechanism better than the other?

1.1 Contributions

The major contributions of this thesis are:

- Development of a model-free RL agent that is capable of learning both a policy to solve the tasks present in multiple video games and a policy to determine where to look while performing such tasks
- Proposal of an architecture whose number of training parameters do not depend directly on the size of the input
- Comparison with a widely studied state-of-the-art RL model such as Proximal Policy Optimization (PPO) and a soft attention RL model available in the literature
- Proof that, in some games, a model that does not process the entire input image already achieves similar performance to the agents that process it entirely
- Analysis of the model's behavior when playing video games like Pong, SpaceInvaders, and CarRacing and comparison with human attention

1.2 Organization of the Document

This thesis is organized as follows: Chapter 2 introduces RL and deep learning theoretical concepts used throughout this work. Chapter 3 provides an overview of the models available in the literature related to the topic of this project. Chapter 4 covers in detail the architecture of the proposed model-free RL agent. Chapter 5 describes the evaluation process and presents the experimental results for our model and the others we are comparing it against. Finally, Chapter 6 makes the conclusion of this thesis and proposes future work.

2

Background

Contents

2.1 Decision Theoretical Models 9
2.2 Reinforcement Learning 11
2.3 Neural Networks 12
2.4 Active Perception and Attention Mechanisms 14

In this chapter, we introduce core concepts of RL and deep learning that are used in the remainder of the document. Additionally, definitions of active perception and attention in the context of AI are also provided.

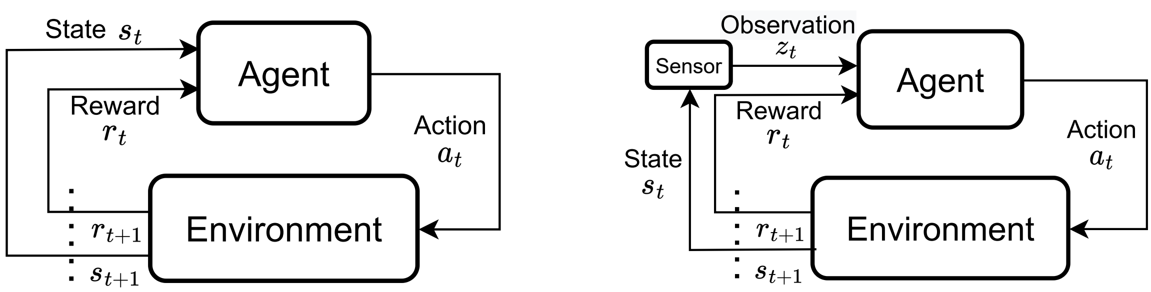
2.1 Decision Theoretical Models

There are many mathematical frameworks to represent decision-making problems. In this section, we will look at two of them, Markov Decision Processes (MDPs) and Partially Observable Markov Decision Processes (POMDPs).

2.1.1 Markov Decision Processes

An MDP is a framework to solve sequential decision-making problems in stochastic environments with full observability. In these problems, an *agent* observes its surroundings—the *environment*—and decides to execute *actions*. The environment reacts to those actions and gives feedback to the agent in the form of *rewards*. The objective of the agent is to maximize its rewards over time. Additionally, MDPs verify the *Markov property*, which says that any given state depends only on the previous state and on the action that was taken.

Formally, this problem can be defined as a tuple $(\mathcal{S}, \mathcal{A}, \mathcal{P}, r)$, where $\mathcal{S}$ is a discrete set of states, $\mathcal{A}$ is a discrete set of actions, $\mathcal{P}(s' \mid s, a)$ represents the transition probability of going from state s to s' performing action a, and $r(s, a)$ is the reward function $r : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$. At each timestep t, the agent is in a state $s_t \in \mathcal{S}$ and selects an action $a_t \in \mathcal{A}$. Consequently, the agent receives a reward $r(s_t, a_t)$ for choosing action a_t and the environment transitions to state s_{t+1} with probability $\mathcal{P}(s_{t+1} \mid s_t, a_t)$. Figure 2.1(a) illustrates this interaction.



(a) Agent-environment interaction in MDPs **(b)** Agent-environment interaction in POMDPs

Figure 2.1: MDPs vs. POMDPs. Adapted from Sutton and Barto [1]

In an MDP, the goal is to find a policy π, a function $\pi : \mathcal{S} \rightarrow \Delta(\mathcal{A})$ that is a mapping from states to a distribution over actions, which maximizes the accumulated rewards through time. We define the state-value function $V^{\pi}(s)$ as the expected return an agent can receive when it starts in state s and follows policy π. We can also define the action-value function $Q^{\pi}(s,a)$ to compute the expected return an agent can receive when it starts in state s, takes action a, and then follows policy π thereafter. Both functions use a discount rate $\gamma \in [0,1[$. The optimal policy π^* is a policy such that $V^{\pi^*}(s) \geq V^{\pi}(s)$ for all $s \in \mathcal{S}$ and all π. To compute the optimal state-value function, we can use a method called Value Iteration, which arbitrarily initializes $V(s)$ and recursively applies the Bellman equation

$$V^*(s) = \max_{a \in \mathcal{A}} \left[r(s,a) + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}(s' \mid s, a) V^*(s') \right]$$

until convergence. Similarly, to compute Q^*, we use

$$Q^*(s,a) = r(s,a) + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}(s' \mid s, a) \max_{a' \in \mathcal{A}} Q^*(s', a').$$

2.1.2 Partially Observable Markov Decision Processes

Not every environment is fully observable, therefore, if the agent is in an environment where the state cannot be fully observed, we can use the corresponding framework called the POMDP. This type of problem also verifies the Markov property, but in this case, since the agent does not have access to all the information about the states of the environment, it must rely on its own observations. Figure 2.1(b) presents this interaction and we can see that the only difference to the MDPs is the addition of a sensor between the agent and the environment, which extracts the observations from the latter.

Formally, this problem is defined as a tuple $(\mathcal{S}, \mathcal{A}, \mathcal{Z}, \mathcal{P}, \mathcal{O}, r)$, where it was added a discrete set of observations $\mathcal{Z}$ and the observation probabilities $\mathcal{O}(z \mid s', a)$, that represent the probability of receiving the observation z, while being in state s' after taking action a. The interaction between the agent and the environment is represented in Fig. 2.1(b).

Since the agent cannot know exactly in which state the environment is in, it must build a set of internal beliefs. These beliefs are represented as a probability distribution over $\mathcal{S}$, $\mathbf{b}_t$. Then, Value Iteration can be used to extract the optimal policy π^*:

$$V^*(\mathbf{b}) = \max_{a \in \mathcal{A}} \left[r(\mathbf{b}, a) + \gamma \sum_{\mathbf{b'}} \mathcal{P}(\mathbf{b}' \mid \mathbf{b}, a) V^*(\mathbf{b'}) \right]$$

However, representing $V^*(\mathbf{b'})$ is not trivial. But since this function is piecewise-linear and convex [18], we can approximate it using a finite set of vectors, which can be computed using dynamic programming.

2.2 Reinforcement Learning

RL problems can be modeled using either MDPs or POMDPs. However, the agent is not able to know either the transition probabilities $\mathcal{P}$ or the reward function r. Therefore, it has to estimate both of them or learn the optimal policy by trial and error.

We can make an RL agent learn using three different approaches: model-based methods, value-based methods, and policy-based methods. The last two are also called model-free methods.

2.2.1 Model-based Methods

Model-based methods estimate the transition probabilities and the reward function. If the environment is fully observable, the agent can know these two parameters, enabling it to understand everything about its surroundings. To build those estimates, the agent interacts with the environment for some time, keeping track of the rewards it receives and how often it visits each state. Then, we can use Value Iteration to find the optimal policy, and as long as the agent visits every state infinitely often, $\mathcal{P}$ and r converge to their real values.

2.2.2 Value-based Methods

Value-based methods do not learn estimates for $\mathcal{P}$ and r, instead, they discover the state-value or the action-value functions. One of the most famous RL algorithms belongs to this category and is called Q-learning. At each timestep of this algorithm, the agent gets a sample (s_t, a_t, r_t, s_{t+1}) and updates Q_t using:

$$Q_{t+1}(s_t, a_t) = Q_t(s_t, a_t) + \alpha_t \Big[r_t + \gamma \max_{a' \in \mathcal{A}} Q_t(s_{t+1}, a') - Q_t(s_t, a_t)\Big]$$

where $\alpha_t \in [0, 1]$ is the learning rate.

Q-learning is an off-policy algorithm, which means it can learn the optimal policy while following a different one.

2.2.3 Policy-based Methods

Instead of learning the transition probabilities or the value functions, policy-based methods try to directly discover an approximation for the policy.

The two families of algorithms mentioned in this work are Policy Gradient algorithms and Evolutionary Strategies.

Policy gradient methods learn a parameterized policy $\pi_\theta(a \mid s)$ using gradients. An example is the REINFORCE algorithm [19] that uses the following update rule, for each $t = 0, ..., T$:

$$\theta \leftarrow \theta + \alpha\gamma^t G_t \nabla \ln \pi_\theta(a_t \,|\, s_t) \text{ , where } G_t \text{ is the return at time t.}$$

One problem with this algorithm is that it has a high variance.

Another policy gradient method is Actor-Critic, an algorithm that learns both the policy (actor) and the value function (critic). It also uses the previous update rule but replaces the return G_t with the critic's estimate.

Evolutionary Strategies [20, 21], on the other hand, apply a completely different concept to learn the policy $r(a \,|\, s)$. They are a family of black-box optimization algorithms based on Darwin's theory of evolution by natural selection. In this approach, every possible solution interacts with the environment and is evaluated using a fitness score. In each epoch, only the best solutions survive. After that, they are recombined with each other (gene mutation) to create the set of solutions that will be evaluated in the next epoch.

When we deal with large domains, finding the exact values for all the functions becomes too complex. Therefore, in those situations, we have to adapt some of the algorithms presented and start calculating function approximations using, for example, neural networks. This moment is when we start exploring the field of deep RL.

In this thesis, we decided to train our model using an actor-critic. Besides being simpler, this method is also more efficient than other policy-based methods such as the evolutionary strategies, which are more computationally demanding. We opted for the proposal of a model-free RL agent instead of a model-based because we decided to understand first if an RL model was capable of having good performance with such visual restriction on the input. Building a complete model of the environment while just seeing a region of it, is another challenge that we leave for future work.

2.3 Neural Networks

The study of neural networks was inspired by how our brains and the connections between their neurons work. Neural networks are formed by multiple layers containing nodes, also known as artificial neurons or units. The first layer is called the input layer, the last one is the output layer, and all the other layers in between are the hidden layers. If the links between the nodes of each layer do not create cycles, that is, if all the connections are from the current layer to the next ones, we call the network a Feedforward Neural Network (FNN) or a Multilayer Perceptron (MLP). Figure 2.2 presents an example of such a network.

Each node of a neural network works as a mathematical function that receives as input: a vector **x** and a vector of weights **w**, where each parameter $x_i \in$ **x** is associated with its corresponding weight

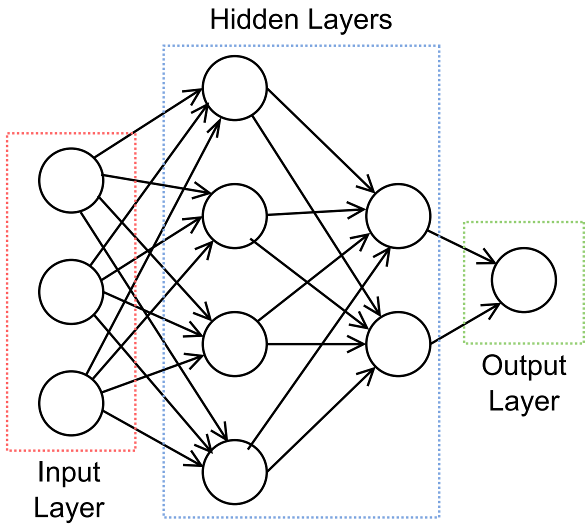


Figure 2.2: MLP with two hidden layers

$w_i \in \mathbf{w}$, and also a bias term b. The weighted sum of the input passes through an activation function, and if its value is greater than a threshold, we say that the neuron "fires". If the node is in the last layer, this value will be the output of the network. Otherwise, the value will be propagated to the nodes of the next layer.

Neural networks are function approximators that learn a correspondence between the input and the output. In order to build the best match possible between those two, these neural networks need training, which is done by minimizing the error between the given output and the expected output. If the activation functions are differentiable, we can use *backpropagation* to train an MLP. This algorithm computes the gradient of the error function with respect to the weights and biases of the network. Starting in the last layer, it uses the chain rule to compute the gradient and update the weights and biases of each layer individually until reaching the first.

2.3.1 Convolutional Neural Networks

Besides MLPs, there are more types of neural networks. CNNs are MLPs specifically designed to receive images as input. These networks can have three types of layers: convolutional layers that extract features from the input, pooling layers that reduce the dimensionality of the input, and fully connected layers where the classification is done, based on the features extracted in the previous layers.

2.3.2 Recurrent Neural Networks

Recurrent Neural Networks (RNNs) are neural networks that have a cycle, which means they have nodes that receive information previously computed in earlier timesteps. This configuration allows RNNs to have memory capabilities that can be used to process sequential or temporal data, like images and text. However, RNNs can only memorize relatively recent information. If we want the network to learn long-term dependencies between the input, we must use a special type of RNNs called Long Short Term Memory (LSTM) networks.

2.3.3 Autoencoders

Autoencoders are another specific type of MLPs, formed by two components: an encoder and a decoder. The objective of the encoder is to learn an efficient way to compress the input. Then, the decoder does the opposite, learning how to reconstruct the encoded version of the input back into a representation that is similar to the original.

2.4 Active Perception and Attention Mechanisms

In 1988, the concept of *active perception* was defined by Bajcsy as the intelligent acquisition of information about an environment in order to understand it better [22]. In comparison to a passive perception agent that statically senses its environment and takes actions accordingly, an active perception agent can improve its performance by actively moving its sensors to better reason about its environment [23].

More recently, after collecting numerous ideas and methods developed by robotics, AI, and computer vision communities, from decades ago to 2016, Bajcsy et al. [24] revisited the definition of active perception, redefining it as the capability of an agent to know *why* it has the desire to sense and then select *what* it should perceive, knowing *how*, *when*, and *where*, it will accomplish that perception. Therefore, they considered an agent to be actively perceiving if it understands the motivation of its behavior and knows at least one answer for the *what*, *how*, *when* or *where* questions.

In AI, active perception models can be implemented using deep learning methods such as CNNs [23]. Since the selection of *what* an agent should sense can take inspiration from the human visual attention process, an attention mechanism can be used to replicate such behavior. In machine learning, attention is a technique that consists in choosing which parts of the input are the most important, resulting in more computational power being allocated to them. In the literature [15–17], we can find two categories of attention mechanisms:

- **Soft attention**: is a mechanism that splits the input into multiple parts, assigning a weight to each one of them. The weights represent the importance associated with each portion of the image, and since this model is differentiable, they can be updated using backpropagation.

- **Hard attention**: is a mechanism where the model does not go through some parts of its input because the neural network itself stochastically decides which are the parts it should pay attention to. However, this mechanism is not differentiable but can be trained using RL.

In the context of this work, hard attention is the most interesting mechanism because it is the closest to human behavior. The model developed is capable of manipulating the stochasticity of its decision, by changing the mean of the normal distribution responsible for the choice of the location to look at.

3

Related Work

Contents

3.1 Regular Reinforcement Learning Models 19

3.2 Reinforcement Learning Models with Attention 23

3.3 Active Visual Exploration 28

Deep reinforcement learning is a field with numerous applications, one of them being the resolution of complex control tasks. This section presents multiple algorithms that can solve such demanding tasks. Almost all agents use RL, and some have attention mechanisms implemented, but in general, every paper has a unique perspective and solution that helped us understand how the literature has been dealing with similar problems to ours. The architectures shown in some of the papers are used as a base for our agent, and other models will be the reference we are trying to match.

3.1 Regular Reinforcement Learning Models

The first RL models developed to solve visual tasks, such as playing video games, received the entire image as input and did not have specific mechanisms to give the agents additional skills to help them complete their tasks. For this reason, in their original version, these architectures are not aligned with the objectives of this work. Since attention is just a mechanism to influence what the agent sees, the models we show later are often based on the architectures we present in this section, which do not implement such a mechanism. In addition, since one of our objectives is to compare the performance of our agent with state-of-the-art models, this section also reports what these architectures are already capable of accomplishing.

3.1.1 Model-Free Reinforcement Learning

Model-free RL algorithms are usually simpler to create because they do not need to build a representation of the environment. Therefore, they were the first to be widely studied.

The first algorithm to achieve human-level performance on Atari games was introduced by Mnih et al. [2], and it was the popular Deep Q-Network (DQN), a model that combines a CNN with Q-learning. Its neural network calculates an approximation for the parameterized optimal action-value function $Q_\theta(s, a)$. DQN takes advantage of a technique called *experience replay* [25], which saves the agent's past experiences (s_t, a_t, r_t, s_{t+1}) into a memory, at each timestep. From that memory, a random minibatch of experiences is selected to perform the Q-learning update. Additionally, the parameters of a target network are updated with the parameters of the Q-network after a fixed number of timesteps. This model receives as input from the environment its game frames and the latest game score. Since the DQN does not memorize the past frames, instead of using just the last frame as input, it uses the last four in order to understand the direction and speed of the objects present in the games. Regarding the outputs, the model gives the Q-values for the individual actions of the agent. Figure 3.1 illustrates the neural network architecture of a DQN. The network is a CNN that receives four images as input, and has three convolutional layers and one fully connected layer. Moreover, the outputs given by another fully connected layer are the Q-values for the individual actions of the agent.

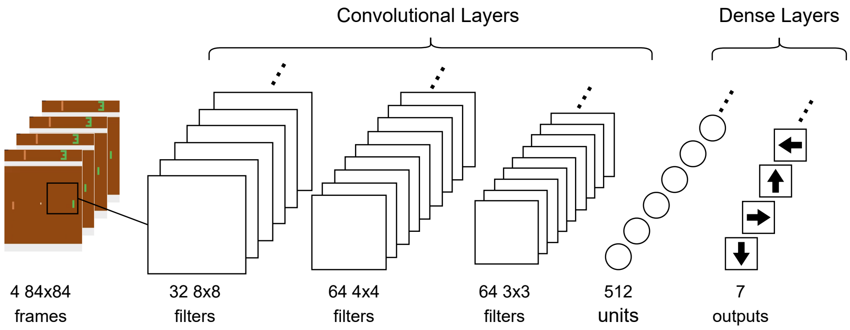


Figure 3.1: DQN architecture presented by Mnih et al. [2]. Adapted from Polvara et al. [3]

This paper was such an important breakthrough in RL that some of its implementation details, like the CNN layout, and optimizations made to play the Atari games are still followed nowadays.

After the publication of this paper, many variants were developed to address different limitations of the DQN. Some of the proposed modifications were the Double Deep Q-Network (DDQN) [26] that addresses the overestimation problem, which happens when the action values are estimated inaccurately, and the Dueling DDQN [27] that presents an architecture that has two streams making estimations, one for the value function and other for the advantage function, which computes the difference between the Q-value and the value function of a given state. In particular, Hessel et al. created Rainbow [28], a model that integrates six of those DQN modifications into a single agent. On the Atari 2600 games, Rainbow outperformed each variant alone.

In [29], Mnih et al. attempted a different approach to deep RL. Instead of using experience replay, they explored the asynchronous parallelization of multiple agents on multiple instances of the same environment. Since more than one agent is running simultaneously, a lot more states can be explored at each time step. This distribution allowed the models to be much less computationally intensive. The best model presented in this paper was the Asynchronous Advantage Actor-Critic (A3C). This model is a CNN that gives two outputs: the policy and the value function, taking advantage of parallelization and accumulated updates. A3C achieved better results than DQN while taking half of the training time.

Schulman et al. [30] proposed a family of policy gradient methods called PPO algorithms. These methods alternate between interacting with the environment to get data and updating the policy using stochastic gradient ascent. In every policy update, this policy gradient method guarantees that the difference between the new policy and the old policy is small, which prevents the algorithm from having a high variance during training. Having the probability ratio

$$r_t(\theta) = \frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{\text{old}}}(a_t \mid s_t)} \tag{3.1}$$

and $\hat{A}_t$, an estimator of the advantage function at timestep t, PPO maximizes the following “surrogate" objective:

$$L^{CLIP}(\theta) = \hat{\mathbb{E}}_t\Big[\min(r_t(\theta)\hat{A}_t, \mathrm{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t)\Big] \quad (3.2)$$

where ϵ is a hyperparameter. The clipping in Equation (3.2) prevents large policy updates and penalizes the probability ratio $r_t(\theta)$ when it tries to move far away from 1.

When compared to the other algorithms, PPO is simpler and more data-efficient. Combining these two reasons with the fact that this model is well studied and was already tried alongside the hard attention mechanism we are using [31], made us choose PPO as the foundation for the model presented in this thesis. Therefore, we are not only adapting PPO to our architecture but also comparing our model with several versions of this algorithm.

3.1.2 Model-Based Reinforcement Learning

Although model-free agents are easier to build, they are also less sample efficient than model-based agents because they take many more epochs to train [32]. Since model-based RL agents learn how their environment behaves, they end up building their own world model, which makes them take much fewer iterations to train. Nonetheless, this comes with the cost of needing to create more complex architectures. These models condense all the past experiences of the agents and can be used to predict the future states of the environment.

If the inputs are high-resolution images, it is beneficial to represent them in a compact manner. This compressed and more efficient way of representing states is called a latent space. Since these models can predict future frames, after some training they are even capable of dreaming, that is, creating their own reality. Hence, they can be trained in a world created by themselves.

Taking advantage of the previous concepts, Ha and Schmidhuber [4] developed an agent that consists of three components, which are represented in Figure 3.2:

- **Vision**: is a Variational Autoencoder (VAE) responsible for creating a latent space z of the input observations. A VAE is a special autoencoder that, instead of returning single values, outputs a probability distribution for each latent attribute;

- **Memory**: combines a Mixture Density Network (MDN) [33] with a recurrent neural network (MDN-RNN) that stores what happened in the past and is capable of predicting the future. This module estimates the probability distribution of the latent vectors;

- **Controller**: has access to the outputs of the other two components (z and h) and is able to select the action to perform, maximizing its expected cumulative reward.

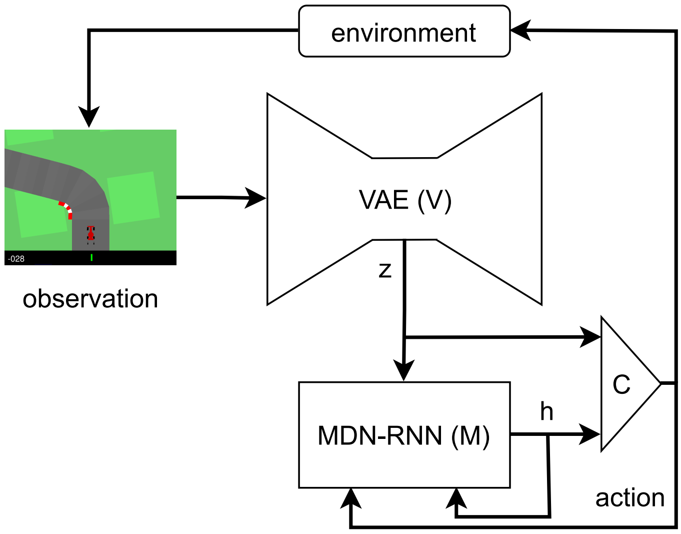


Figure 3.2: Flow diagram of Ha and Schmidhuber's agent. Adapted from [4]

The Vision and Memory models were trained using a backpropagation algorithm, unlike the Controller whose parameters were optimized using an algorithm called Covariance-Matrix Adaptation Evolution Strategy (CMA-ES). The authors tested this agent in two games, CarRacing and DoomTakeCover, and it was capable of surpassing the required score to beat the game in both. In the first game, this agent performed significantly better than models like DQN or A3C.

Following the previous model, Hafner et al. created DreamerV1 [34] and DreamerV2 [35], two models that are focused on learning long-horizon behaviors in their imaginary representation of the environment. They can both be used to solve complex control tasks with continuous or discrete sets of actions. DreamerV1 is an actor-critic model that uses the VAE from Ha and Schmidhuber, a Recurrent State-Space Model (RSSM) [36] and three dense layers. DreamerV2 is more focused on discrete sets of actions, beating Rainbow on 55 Atari games. This second version makes some improvements over the first one and has six components that are neural networks: a recurrent model, a representation model, a transition model, an image predictor, a reward predictor, and a discount predictor.

According to Zhu et al. [37], models like Dreamer that are focused on optimizing their policy on the imaginary state space, take the risk of deviating from reality and getting stuck in local solutions that are not optimal. With this problem in mind, Zhu et al. presented BrIdging Reality and Dream (BIRD), a model that uses information from the real and imaginary trajectories, such that the resultant policy learned using imagination can be a good generalization to the real world.

More recently, after the success that was AlphaGo Zero [38] and AlphaZero [39], Schrittwieser et al. developed a more general model called MuZero [40]. Unlike its predecessors, this model did not have the rules of games like Go or Chess incorporated into it. This meant it was capable of playing a wide variety of games, even without having any knowledge about the environment dynamics. This algorithm accomplished such achievement by learning a model that, at each timestep, predicts three values: the policy, the value function and the immediate reward. Given all past observations, this model can look for future trajectories of actions using a search algorithm such as Monte Carlo Tree Search (MCTS). MCTS is a probabilistic search algorithm that uses heuristics to efficiently search huge game trees. It combines the previous three values using lookahead search and produces an improvement of the policy and the value function. After that, we can use the policy to extract the set of actions to perform. When compared to the other model-based RL agents, this model has a stronger emphasis on planning, which allows it to play very complex classic board games such as Go, Chess and Shogi (Japanese Chess). In those board games, MuZero matched the superhuman performance of its two predecessors, AlphaGo Zero and Alpha Zero. Additionally, it also established a new state-of-the-art performance on 57 Atari games. For example, when compared to Rainbow, MuZero achieved a median normalised score of 2041.1%, while the first only got 231.1%.

After the presentation of all these models, we conclude that MuZero might be the current strongest model, setting the benchmark for discrete action space games such as the Atari 2600 games. For games with continuous actions like CarRacing-v0, the model from Ha and Schmidhuber has state-of-the-art performance. Although they achieve such scores, all the previous models have too many parameters and have to use low-resolution images of the environment, because they do not discard pixels having irrelevant information for their task.

3.2 Reinforcement Learning Models with Attention

Although our eyes capture a large visual field, we have limited processing power, which means we need to select which regions we want to allocate our resources to. Since computers also face the same dilemma, our attention process has inspired the deep learning community to build similar mechanisms to allocate better their resources. This research resulted in the development of models that are capable of choosing which parts of the input are the most important, allocating more computational power to them. Attention mechanisms already have a great impact on areas like natural language processing [41] and speech recognition [42].

3.2.1 Soft Attention

Unlike hard attention, which is the focus of this work, soft attention has been more explored [17], so there already exist some models we can use for comparison.

Mott et al. [43] developed a model that uses a soft, top-down attention mechanism to discover the most relevant zones of the input image. The architecture of this model combines a CNN with LSTM networks and MLPs, resulting in a fully differentiable model that can be trained using backpropagation. The attention mechanism produces attention maps that can be used to discover which parts of the input the agent considered the most relevant for its task. This agent outperformed models like Rainbow across multiple Atari games.

With a connection to the previous model of Ha and Schmidhuber [4], hereafter referenced as World Models, Tang et al. [8] explored the concept of self-attention as a form of indirect encoding, enabling the neural networks to have many fewer parameters. More precisely, this model has only 3,667 total parameters, while World Models has 4,733,485. This model starts by resizing the input image, which is slightly bigger than what World Models uses. Next, it divides the input into patches that are later flattened. After that, the attention mechanism assigns a level of importance to each one of the patches applying softmax, which can be seen as a voting mechanism. Then, the agent selects the K most important patches and retrieves relevant features from them. These features are based on the location of the patches and not on their contents. Finally, with this information, the controller, which is an LSTM network, selects the action to perform. The authors decided not to use a backpropagation algorithm, opting instead for CMA-ES. Like World Models, this agent was also tested on the CarRacing and DoomTakeCover games, achieving slightly better performance on both. This model, unlike the other, has good generalization capabilities, still being able to achieve competitive scores when deployed in modifications of the two game environments, with color perturbations or different textures applied, after being trained in the normal environments. This result is probably a good indication that a model with attention mechanisms is more robust than one that has to process the entire input image.

There are also some variants of DQNs with attention mechanisms, such as the work of Sorokin et al. [5]. Actually, their work is based on another derivation called Deep Recurrent Q-Network (DRQN) [44], where the first fully connected layer of the DQN is replaced with an LSTM layer. The addition of a memory component removes the need to use the last four frames as input every timestep, making it use just the last one. Based on this change, Sorokin et al. presented the Deep Attention Recurrent Q-Network (DARQN) with an architecture that has three layers: a CNN, an attention network, and an LSTM. Figure 3.3 illustrates this architecture. Every timestep, the CNN receives a game frame s_t and produces a set of feature maps v_t. Then, the attention network receives v_t, transforms it into a set of vectors, and feeds the LSTM with their linear combination, calling them context vectors z_t. The LSTM using its previous hidden state h_{t-1} and the memory state c_{t-1}, calculates the Q-value. The authors tested two approaches for the attention network. The first one is a soft attention mechanism where different levels of importance are given to each context vector, and it can be trained using backpropagation. The other approach is based on hard attention, so it uses a policy to choose the attention location, which can be trained using REINFORCE. In four of the five Atari games where both attention mechanisms were tested, the soft attention network outperformed the hard. However, only in two of the five games, both attention mechanisms achieved higher scores than the DQN model.

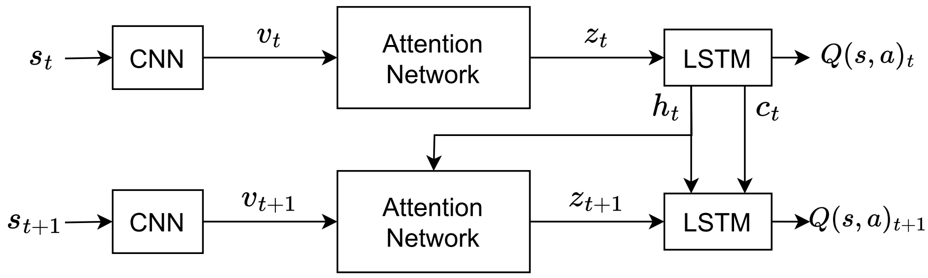


Figure 3.3: DARQN architecture. Adapted from [5]

The fact that two of the previous soft attention models outperformed some well-established regular RL agents is a good indication of the capabilities of attention mechanisms.

3.2.2 Hard Attention

Hard attention is the mechanism this work is focus on since it allows the model to control the reception of its input. This is the condition necessary for the model to have active perception. In this mechanism, the agent must choose where to look before having access to the context of the images, which makes it discard the regions of the input that are not relevant. For this reason, hard attention is considered more efficient and faster to train [15, 17].

The paper that was found to replicate better the behavior of the human eye was the work of Mnih et al. [7], where they present a model called Recurrent Attention Model (RAM). In their problem specification, the agent does not have access to an entire image of the environment, so it uses a **glimpse**, a retina-like representation of a portion of an image centered around a location l. The region of the image around l has high resolution; regions further away from l have increasingly lower resolution. Such representation is crucial to the performance of the agent and is what makes the complexity of the model not dependent on the size of the input images. Instead, it depends on the size of the glimpses. This model has the following components:

- **Glimpse Network**: receives the glimpse and its location and extracts into a vector a set of features from them;
- **Core Network**: is an LSTM that builds the internal representation of the environment. It receives the vector from the Glimpse Network and updates its hidden state;
- **Action Network**: uses the hidden state of the Core Network to predict the class label of the input;
- **Location Network**: uses the hidden state of the Core Network to select the coordinates for the next glimpse location;

The differentiable modules, which are the Glimpse, Core and Action Networks were trained using backpropagation, while the Location Network used REINFORCE. This model was tested in image classification tasks on the MNIST dataset and playing the game of ”Catch”, outperforming a regular CNN in both.

Later, Ba et al. [6] proposed an improvement on RAM called Deep Recurrent Attention Model (DRAM), which was capable of dealing with multiple objects and real-world tasks. Figure 3.4 presents the architecture of this model. Each timestep, the model receives an image x_n and a set of coordinates l_n. In the Glimpse Network, a glimpse is extracted from x_n and the features are transmitted to the Recurrent Network, which has two LSTMs, $r_n^{(1)}$ and $r_n^{(2)}$. To initialize the second LSTM, the Context Network takes a low resolution image of the entire input that will help the Emission Network decide the coordinates of the first glimpse. The output of first LSTM is used by the Classification Network to classify the image received, while the output of the second LSTM is used by the Emission Network. The major differences from RAM are the following:

- the addition of three convolutional layers to the Glimpse Network
- the inclusion of a second LSTM that receives the hidden state of the Core Network and outputs the next location coordinates. This change allowed the separation between the representation of the location policy and the representation of the classification/action policy;
- the introduction of a Context Network that uses a down-scaled image of the entire input to help guide the model to the most relevant zones of the environment;

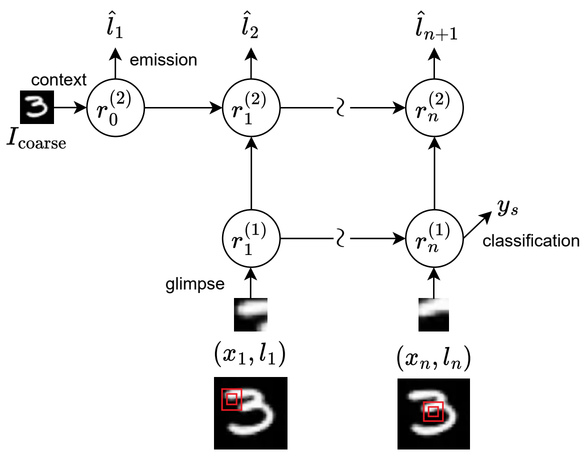


Figure 3.4: DRAM architecture represented across several timesteps. Adapted from [6]

DRAM was tested on multi-object classification tasks in a cluttered and translated variant of the MNIST dataset and transcribing house numbers from Google Street View images. The authors are confident that their model is powerful and flexible enough to tackle other complex computer vision tasks.

More recently, in his thesis, Zuur [31] tested a few variations of RAM and DRAM. The author created two models, each one based on each architecture, where the Action (or Classification) Network and the Location (or Emission) Network were separated from each other. This separation resulted in the use of two Glimpse Networks, one that provided a smaller high-resolution image to the part of the network that computed the next action, and another that gave a larger low-resolution image to help choose the next glimpse location. Zuur also made several experiments: discovered the impact of computing the Horizon-Aware value function instead of the usual state-value function; the effect of adding noise to the exploration; and the difference between training the model using PPO or the REINFORCE algorithm. The latter was the only experiment where a considerable difference was noticed. The author concluded that when the tasks became more complex, PPO outperformed REINFORCE because its capacity for extracting relevant information from a noisy environment was greater.

The previous architectures were mainly developed to complete image classification tasks. However, there is also little research about models that were tested on video games.

Starting with [45], Gilmour et al. proved that is possible to adapt the DQN to choose not only an action to take but also an image to look at. In their problem, the agent has to choose between three different images of the input, each one of them having only one-third of the image not hidden. However, outputting two different actions at the same time has the problem of combinatorial explosion. In order to solve that problem, the CNN was branched into two different heads, each making one of the decisions. The agent was trained using curriculum learning, learning first how to complete an easier task, and only after succeeding on it, a more difficult task was given. The model was tested on the Atari game Pong, and it showed worse performance than the regular DQN.

Gregor et al. [46] also adapted DQN to take advantage of visual attention capabilities, but now the goal was to play the Pac-Man game. This solution implemented the concept of glimpses that we saw on models like RAM, but unlike that paper, the authors assumed that the most relevant information was always around Pac-Man. Hence, the agent did not need to choose where to look. This work demonstrates that, at least in the Pac-Man game, an agent with attention outperforms another agent without it, which receives an image of the entire environment as input. Additionally, they showed that besides feeding the model with just a high-resolution glimpse, providing it also with low-resolution images of the remaining surroundings, can improve the performance of the agent.

Recently, Sahni and Isbell [47] proposed a slightly different approach to implement a hard attention agent. In their paper, the model tries, at every timestep, to reconstruct the full state of the environment using only its partial observations while having the least possible reconstruction error. The authors presented an architecture with two separate RL agents. Firstly, we have the Glimpse Agent that controls the attention mechanism, choosing the location of the next glimpse. This choice has to reduce the uncertainty of the internal representation of the environment state, in other words, it tries to maximize the amount of surprise, which leads to the maximization of the mutual information. In order to do this, it uses a recurrent memory unit called Dynamic Memory Map (DMM), which is an adaptation for partially observable environments of the SpatialNet [48], a model that learns the physical dynamics of an environment, to predict them in the future. Since its only focus is to find the zones that the DMM considers more difficult to reconstruct, this agent is independent of the task. Secondly, we have the Environment Agent that uses its internal representation to select the next action to take. This architecture was trained using PPO and tested in two games, whose objective was to reach a certain goal while avoiding obstacles and enemies. In the end, the authors found out that this agent tends to get distracted by random and unpredictable events in the environment.

The research on hard attention mechanisms applied to RL agents that play video games is scarce, and from the papers presented here, we can see mixed results. In general, it is possible to note a trend in all the presented architectures: the use of a CNN to extract features from the small region of the input, and the use of two separate networks to deal with the selection of the action to take and the location to look next. The use of LSTMs is also common in the majority of the models. After analyzing all the models, RAM and its improvements, despite being mainly focused on image classification, looked the most promising to be adapted to play video games. Combining this with the fact that the work of Zuur [31] proved PPO is a good algorithm to be used in this type of architecture, we decided to choose RAM as the base architecture of our model.

3.3 Active Visual Exploration

Although in this work we do not intend to build a model that is capable of reconstructing the entire environment from its internal representation (model-based RL), we decided to have this section because it proves that such task is possible and it could be useful for future work.

As we have seen in the last paper of the previous section, another solution to the problem of having an agent with a limited vision of the environment is to build a model that is capable of imagining, from the small glimpses it takes, how the entire environment is. This solution can take us to another type of problem called active visual exploration. In this kind of problem, an agent is put in a new environment and it must be able to explore it efficiently, in order to build a reconstruction of its entire surroundings.

In [49], Jayaraman and Grauman tackled this same problem. In their definition, the RL agent has a limited number of timesteps to explore the environment. In the end, it has to output a representation of the entire environment, which can be a scene or a 3D object. Since the agent has partial observability, it cannot see the whole environment during the available time. This means it must choose the zones that have the most amount of information, for it to make good generalizations of the unseen places. To solve this task, the authors split their architecture into five modules, each one with a different function:

- **Sense**: comprises a CNN, which receives as input an image of the environment, and a vector, which stores proprioceptive metadata such as coordinates and relative motion from the previous input. Sense outputs two vectors: one resulting from the processing of the image by the CNN and the other is the metadata vector that was kept unchanged;
- **Fuse**: combines those two vectors and sends this information to Aggregate;
- **Aggregate**: is an LSTM that remembers the information from past observations. At each timestep, it updates its encoded internal representation, sending it to Act and to Decode;
- **Decode**: translates the coded information and upscales it in order to build the predicted reconstruction of the environment;
- **Act**: selects the next action to move its viewpoint.

This agent was trained using REINFORCE and was tested on the SUN360 [50] and ModelNet [51] environments.

In [52], Ramakrishnan and Grauman showed another solution to the active visual exploration problem. The authors used the previous implementation from Jayaraman and Grauman and added what they called a sidekick, an entity that complements the agent with alternative points of view, knowledge, or skills. During training, the sidekick can see the full environment and solve a simpler problem. The sidekick influences policy learning, accelerating the training of the agent and helping it converge to better policies. In this work, two types of sidekicks are presented: a reward-based sidekick that defines a set of views that can provide the most information about the environment; and a demonstration-based sidekick that selects the trajectories that can be the most informative. The paper also presents a policy visualization technique that displays a sequence of heatmaps showing the areas of the environment that were most responsible for the actions of the agent.

Seifi and Tuytelaars [53] have the opinion that the agents from the previous two papers, despite having the image coordinates of each glimpse, struggle to deduce the spatial correlation between them. With this in mind, they proposed a model that does not use RL and where the LSTM layers in [49] and [52] are replaced with spatial memory maps. The architecture of the agents is composed of three modules:

- **Local Reconstruction Module**: receives glimpses and uses a super-resolution architecture to upscale them to their full resolution
- **Full Reconstruction Module**: fits each upscaled glimpse in its correct location in a spatial memory map. This map is a matrix that has the same dimensions as the input and is initially filled with zeros
- **Attention Module**: chooses where the agent should look next. The location that will be chosen is the area of the input where the model's reconstruction has the least information.

This network was trained end-to-end with Adam optimizer [54].

In this topic, we saw that even while having limited vision of the environment, agents can still learn how to fully reconstruct it.

From this chapter, we can conclude the works presented here show that regular RL agents achieve state-of-the-art performance, surpassing even humans. However, those architectures are too complex and take many hours to train because they do not avoid processing pixels that contain irrelevant information for their task. In order to minimize this problem, the existent models have been adapted to comprehend attention mechanisms, creating more efficient solutions. In this chapter, we showed that, when playing games such as CarRacing or DoomTakeCover, there already exist agents with soft attention mechanisms agents capable of maintaining (at least) the performance of a model without those mechanisms, being even more robust. However, the exploration of hard attention mechanisms applied to this task has not been the focus of much research. This gap in the literature is what our work wants to fill.

4

Model

Contents

4.1 Architecture 33
4.2 Training 37

In this chapter, we introduce a novel model called Glimpse-Based Actor-Critic (GBAC) that combines a hard attention mechanism with a model-free RL algorithm. When compared to other RL models, GBAC processes much fewer pixels, and its training parameters do not depend on the size of the input, which makes it more efficient.

In our problem, the game environment gives, at each timestep, a frame of the game. Since our model cannot have access to all the information of that frame, it has to select only a portion of it to be its observation. That observation has the coordinates that were chosen by the model in the previous timestep. During its interaction with the environment, our agent has to learn the best policy that selects the action to be performed in the game. While doing this, the model also has to understand which regions of the game frame have the most valuable information and learn another policy to choose the set of coordinates to be taken in the next timestep.

As we have seen in Section 2.1.2, this problem can be seen as an instance of a POMDP. In our case, the observations the agent takes are represented by the glimpses, and the history of past interactions with the environment is stored in the hidden state of two LSTMs. When combining the memory, the current glimpse, and the previously chosen location, the LSTMs have all the information needed to learn the policies that select the actions and the locations to take next.

4.1 Architecture

The architecture of RAM [7] presented in Section 3.2.2 was the basis for our proposal, as a result, we present it in Figure 4.1 and will describe it in more detail next.

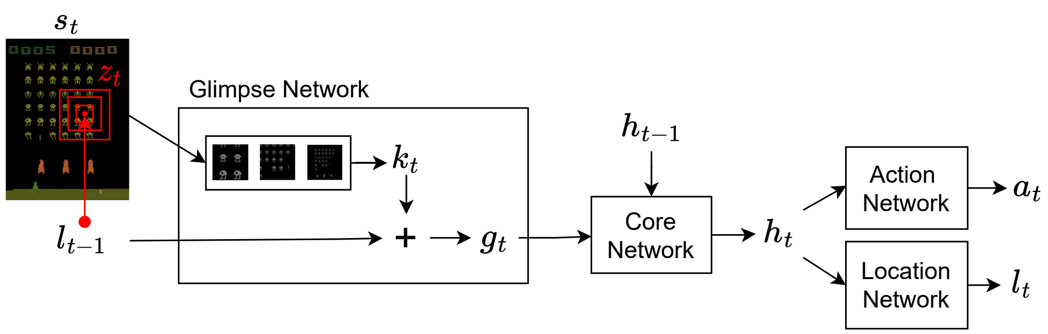


Figure 4.1: Architecture of RAM. Image adapted from the original paper of Mnih et al. [7]

Every timestep, RAM receives the game frame s_t and a set of coordinates l_{t-1}. Its Glimpse Network takes a glimpse z_t centered at l_{t-1} and extracts features, not only from z_t, which are represented in Figure 4.1 by k_t, but also from l_{t-1}. Using fully connected layers the two features are merged into the vector g_t. Next, this vector is fed to an LSTM, the Core Network, which stores all the previous information from the glimpses and the coordinates chosen, building its internal memory h_t. After that, h_t is the input for two other networks, the Action Network and the Location Network, which are also fully connected layers that select the next action a_t and coordinates l_t, respectively. Since the Location Network was non-differentiable, it was trained using the policy gradient method REINFORCE, while the other components used backpropagation.

In contrast with this architecture, Figure 4.2 presents the architecture of our model where we can see the modifications we have made to suit our problem. We start by receiving the same two inputs, a game frame s_t and the set of coordinates l_{t-1}, which our agent chose at the end of the previous timestep. The Glimpse Network takes a glimpse from s_t centered at l_{t-1} and saves into k_t the features extracted. After that, the vector k_t is used as the input of the Action Network. This network outputs not only the action a_t the agent will take next, but also an estimate v_t of the value function because we are now using an actor-critic. k_t is also used by the Location Network, which merges it with the features extracted from the location l_{t-1} to select the next location coordinates l_t. In opposition to RAM, instead of doing this merging earlier, inside the Glimpse Network, we only do it in the Location Network.

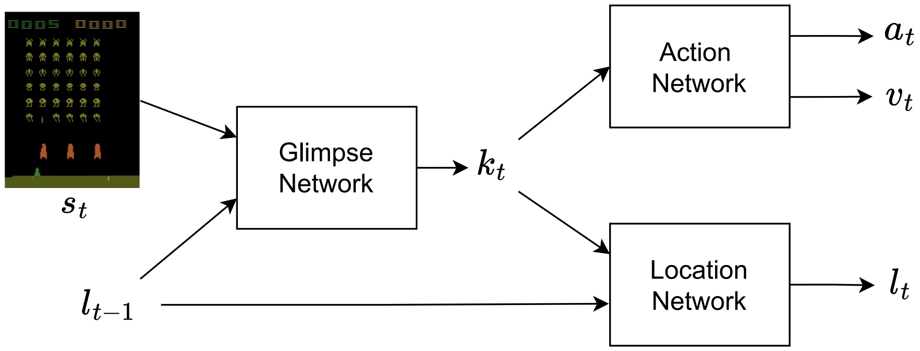


Figure 4.2: Overview of the architecture of the Glimpse-Based Actor-Critic

We have already seen that after the publication of RAM, some papers, such as the works of Ba et al. [6] and Zuur [31], proposed some improvements to that model. We also took some of those refinements into consideration and adapted them to our problem. To start, we followed Ba et al. [6] and added a CNN to the Glimpse Network (represented in Figure 4.3) because it was necessary for the model to extract better features from the video frames. As one of the experiments done by Zuur [31] suggested, we also found that it was beneficial to provide separate inputs for the Action and Location Networks (Figure 4.2). Therefore, the configuration which performed best was the one that fed to the Action Network only the features (k_t) extracted from the glimpse, and to the Location Network the vector resultant from the merge between k_t and the features extracted from l_{t-1}. Still with this idea of separating the processes of choosing the next action and the next glimpse coordinates, we added an extra LSTM, making the Action Network and the Location Network use their separate LSTM (Figures 4.4 and 4.5). This separation destroyed the need for having an explicit Core Network like in RAM, avoiding the mix of information that is necessary for each task, and allowing us to fine-tune the parameters for each LSTM. Before adding another LSTM to the model, we tested using just one, but our agent performed poorly. The last change we made was the selection of PPO instead of REINFORCE to train the model since it achieves better results in harder environments [31] and we found it simpler to train. This change meant that in each timestep, our Action Network also needed to make a prediction v_t about the quality of performing the action a_t in the current state of the environment.

4.1.1 Glimpse Network

The Glimpse Network, represented in Figure 4.3, is the module responsible for extracting from the game frame, the region the agent chose to focus its attention. With the image coordinates l_{t-1} chosen in the previous timestep, this network extracts an observation z_t from s_t, which is called a *glimpse*. This glimpse can have multiple patches, each having double the size of the previous. For example, Figure 4.3 presents a glimpse with three patches. However, to simulate peripheral vision, all the larger patches are downscaled to the dimension of the smallest. That smallest patch will have the highest resolution, making it the focal point. Regarding the other patches, the larger they are, the further away they are from the focus point, so the lower their resolution is. When compared to the original image, this process results in a vector with fewer pixels.

After rescaling, the resultant vector passes through a set of convolutional layers and a fully connected layer to extract a vector of features k_t. The number of convolutional layers and their respective kernel sizes and stride are changed depending on the size of the glimpse..

During this work, we assumed that the glimpses are always squares. If the coordinates of l_t make a patch catch pixels that are out of the bounds of s_t, that patch will be moved in order to fit inside the frame. This mechanism proved to achieve better results than simply filling the pixels out of bounds with a value.

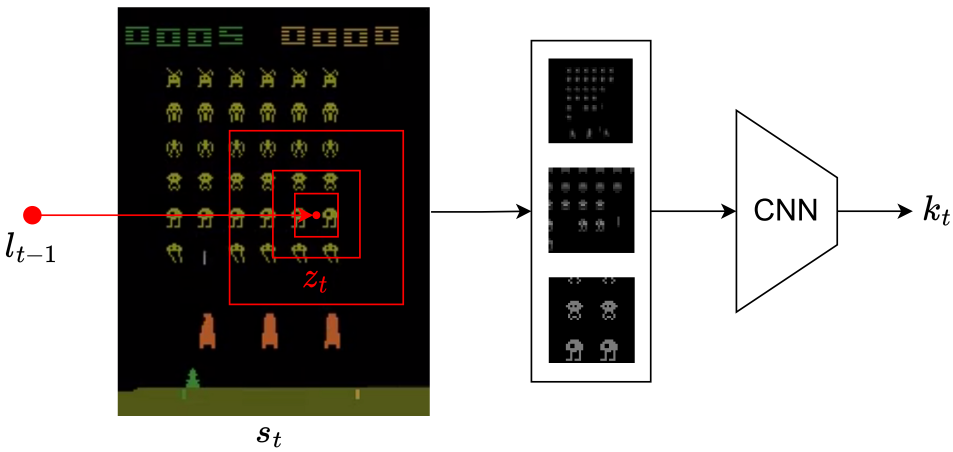


Figure 4.3: Detailed diagram of the Glimpse Network of GBAC

4.1.2 Action Network

The Action Network, depicted in Figure 4.4, is responsible to choose the game action a_t the agent should perform in each timestep. Since we chose an actor-critic algorithm to train GBAC, the Action Network also estimates the value function, which helps the agent to understand if it is performing well or not.

In this network, its LSTM saves the information k_t extracted previously from the glimpse and combines it with its hidden memory h^g_{t-1}, which stores all the information gathered in earlier timesteps to build an internal representation of the game environment. The new hidden state h^g_t is fed to two fully connected layers, the Actor and the Critic, that output the action a_t and the value v_t, respectively.

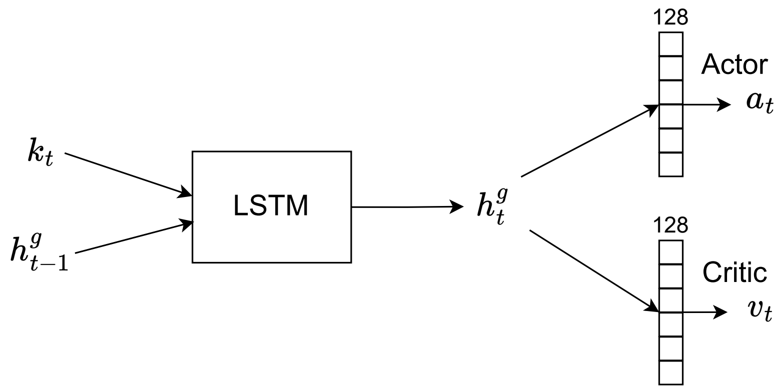


Figure 4.4: Detailed diagram of the Action Network of GBAC

4.1.3 Location Network

The Location Network, illustrated in Figure 4.5, is the module behind the hard attention mechanism and is responsible to choose the image coordinates l_t where the agent should look in the next timestep. Those coordinates are sampled from a truncated normal distribution whose mean is given by this network, being the standard deviation a fixed value.

In order to choose the mean value, the Location Network has a neural network that extracts features from the coordinates l_{t-1}, merging them with the features k_t. The resultant vector g_t is then used to update the internal state of an LSTM. Its internal state h^l_t is fed to the Locator, which is a neural network with two fully connected layers and ReLU and Tanh activation functions, respectively. The Locator chooses the mean value used in the truncated normal distribution from which the next set of coordinates l_t is extracted.

Since this model only uses a small portion of an image, its complexity is not dependent on the size of the input image but rather on the size of its glimpses. This can be an advantage if the model has to handle large images.

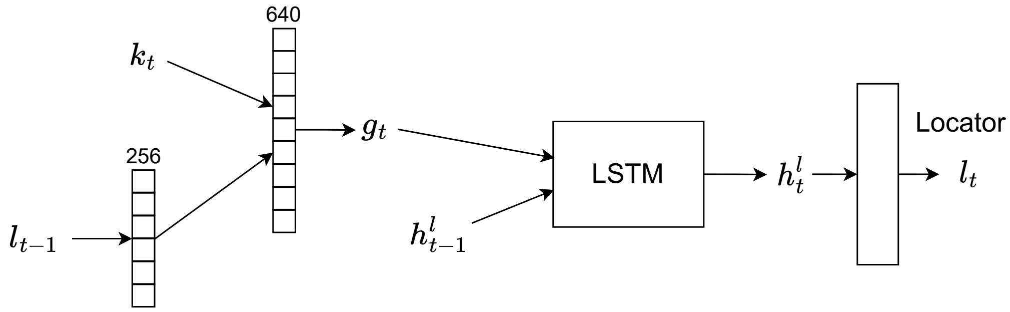


Figure 4.5: Detailed diagram of the Location Network of GBAC

4.2 Training

The training process of our model is very similar to the one presented by Schulman et al. in the PPO paper [30]. We follow the suggestions they proposed, and the only difference is that we have two policy losses, instead of just one. Like PPO, our model alternates between interacting with the environment to get new information and updating both its policies with that new data.

When exploring the environment, in each timestep, the model saves the action and coordinates of the glimpse it chose, the value function estimated by the critic, the reward received from the game, and a flag that indicates if the current episode has finished.

After a predetermined number of timesteps, the model uses the collected data to update its policies. For both policies, we use the "surrogate" objective already presented in Section 3.1.1:

$$L^{CLIP_\pi}(\theta) = \hat{\mathbb{E}}_t\Big[\min(r_t^\pi(\theta)\hat{A}_t, \mathrm{clip}(r_t^\pi(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t)\Big] \tag{4.1}$$

where ϵ is a hyperparameter and the advantage function $\hat{A}_t$ is computed recurring to Generalized Advantage Estimation [55].

Besides the two policy losses, the objective has two more terms: an entropy bonus B that promotes the exploration of the environment, and the squared-error loss of the value function, L_t^{VF}. With these additions, the objective's formula is the following:

$$L_t(\theta) = \hat{\mathbb{E}}_t\Big[L_t^{CLIP_\mathrm{a}}(\theta) + L_t^{CLIP_\mathrm{g}}(\theta) - \alpha L_t^{VF}(\theta) + \beta B[\pi_a](s_t)\Big] \tag{4.2}$$

where α and β are coefficients. Note that the entropy bonus is only calculated for the action policy, not for both policies. Adding the same bonus for the location policy did not seem to improve the results, thus we decided to keep the objective simpler.

5

Experimental Evaluation

Contents

5.1 Evaluation Process 41
5.2 Base Models Selection 43
5.3 GBAC Performance Analysis 45
5.4 Base Models Comparison 50
5.5 Hard Attention vs. Soft Attention 53

In this chapter, we present all the experiments that enabled us to test our model, establishing which base PPO algorithm is the fairest choice to compare our model with, discovering which size of the glimpses gives better results, and seeing how our model compares against another that uses a different attention mechanism. In the end, we have all the information necessary to answer the questions in Chapter 1.

5.1 Evaluation Process

In order to measure the performance of GBAC and compare it against state-of-the-art RL agents, we selected three game environments with different characteristics. The comparison is made with three versions of PPO [30] and the soft-attention agent from Tang et al. [8], measuring the episodic returns the models get during training and testing.

5.1.1 Game Environments

The first two games are both from the Atari 2600 and have a discrete action space, while the third is CarRacing [56] from OpenAI Gym [57] and has a continuous action space. We tried to select three games that were not the easiest ones available and that required the agents to learn different sets of skills, such that we could see how they were capable to adapt to each type of task.

Pong is a tennis-like game where we control a paddle vertically (in this case, the one on the left) and play against another paddle controlled by the computer. The objective is to defend our side and deflect the ball to the opponent's side without it being able to catch it. The first player to reach 21 points wins. In this game, the reward function gives +1 when the agent scores a point and -1 when the opponent manages to score a point.

Space Invaders is a space combat game whose objective is to destroy all the enemies before they reach the player. However, the enemies can also shoot lasers, so the agent has to be able to dodge them too. The objective is to score the most amount of points by destroying enemies. The game ends when the agent loses three lives or the enemies reach it. Depending on the distance to the enemies, the reward to destroy them is different. For example, the enemies from the row that is the furthest from the agent are worth 30 points, while the enemies from the closest row are only worth 5 points.

CarRacing is a top-down racing environment that consists in driving a car through a randomly generated track while taking the least amount of time. Being an environment with a continuous action space, at each timestep, the agent has to choose values for the amount of steering input (-1 represents full left and +1 full right), gas and breaking (0 is no input, +1 is maximum input). Being N the number of visited tiles of the track, the agent receives as a reward +1000/N for every track tile that it visits and -0.1 for every frame that passes. The creator of this game environment considers the game is solved when an agent consistently achieves scores $\geq$ 900 points.

In order to obtain the best performance in the Atari games, we used the same environment modifications proposed by Mnih et al. [2] and Machado et al. [58], thus leading to better results for these games. Those modifications are well accepted and used in the literature and include:

- sampling initial states by taking between 1 to 30 no-ops actions when the environment resets to create some stochasticity;
- always skipping four frames, repeating the agent's last action and summing their rewards in those skipped frames, in order to speed up the algorithm;
- marking the loss of one life as the end of an episode, in games that have a life counter like SpaceInvaders;
- resizing the original image from 210x160 pixels to 84x84 pixels;
- clipping the rewards to +1, 0, or -1 depending on their sign;
- stacking four frames (when an LSTM is not used) to make the agent capable of understanding the velocity and direction of moving objects;
- scaling the image from their [0, 255] original values to [0, 1].

5.1.2 State-of-the-art Reinforcement Learning Agents

We decided to compare GBAC with three different versions of PPO. The first one is the original PPO agent presented by Schulman et al. [30] because it was used as the base for our model. However, since our implementation uses LSTMs in its architecture and the version of PPO with an LSTM is also quite common in the RL literature, we decided to choose the PPO+LSTM agent for comparison as well. The third version of PPO we used is a modification of the PPO+LSTM algorithm where the restriction of only using a small portion of the game frame was imposed. But, different from GBAC, the coordinates where the agent looks are chosen completely randomly. This allows us to test if the perception mechanism implemented in our model is better than one that makes its choices randomly.

The PPO implementation used in this work was not the original one provided by Schulman et al. [30] in OpenAI Baselines [59], but rather a revised implementation presented by Shengyi et al. [60] that closely follows the performance of the original. This implementation provides many versions of PPO including one with an LSTM. To make the PPO agents capable of playing the selected games, we added to their architecture a CNN with the same layout as the one presented in the DQN paper [2].

By comparing GBAC with the first two versions of PPO, we can discover if it is possible to achieve state-of-the-art performance playing video games, despite having a limited (but active) perception of the environment, which was our first question.

Finally, in order to compare the two types of attention mechanisms, our model is also measured against the soft attention agent of Tang et al. [8]. Consequently, we can answer the second question and discover if one of the attention mechanisms is better than the other or if it depends on the task.

5.1.3 Evaluation Metrics

Evaluating RL models is never a straightforward task due to their high variance. For this reason, during training and testing, we average the episodic return each agent achieved over the last 100 episodes.

In training, Pong, SpaceInvaders, and CarRacing learned during 15 million, 20 million, and 5 million timesteps, respectively. Then, the agent that achieved the best performance over the last 100 episodes is tested for another 100 episodes. For each configuration, the results are always the average over three runs and their respective standard deviations are also presented.

Seeding is another aspect that can have an impact on the performance of the agent. Certain seeds can make the agent perform significantly better or worse. Thus, in each run, the seed used in the environment is chosen arbitrarily.

Regarding the policy that chooses the locations of the glimpses, in order to understand if our model learns a behavior that resembles the human vision, we analyze the evolution of the policy throughout the training phase and present a visual representation of the results. This is the information we need to answer our third and final question.

Now that the entire evaluation process is detailed, we present, in the next sections, the results we obtained.

5.2 Base Models Selection

In contrast to one of the modifications presented in Section 5.1.1, our model does not perform better when the original image is resized. In fact, it never learned a way to beat its adversary in Pong when a glimpse smaller than the resized size of 84x84 was used. This meant that with that setup, we could not take advantage of the glimpse's structure because the model needed the entire image to perform well, which does not follow the restriction of our problem. An explanation for this result can be the fact that since the glimpses taken by GBAC have multiple patches that end up being resized to a lower resolution, and the input frame fed to the model was also resized, the loss in information might be so much that our agent is not capable of solving the game.

Therefore, to make the comparisons fair, we decided that the base PPO models should also use the entire image as input. In order to discover how much this decision could impact the performance of the base agents when playing the two Atari games, we studied the difference in performance between using the original 210x160 image and the resized 84x84 input. In CarRacing, since the original image is just 96x96, we decided not to compare the base models with a resized version of the input. In addition, since GBAC uses LSTMs, we compared PPO with PPO+LSTM to find out if they perform any differently.

		PongNoFrameskip-v4		SpaceInvadersNoFrameskip-v4	
Model	Full Img.	Max. Train Avg.	Test Avg.	Max. Train Avg.	Test Avg.
PPO	No	21.00 ± 0.01	20.98 ± 0.02	2090.00 ± 254.16	2013.07 ± 244.56
PPO	Yes	20.91 ± 0.11	20.83 ± 0.13	2261.90 ± 295.87	2221.62 ± 201.31
PPO + LSTM	No	20.11 ± 0.25	20.00 ± 0.31	1182.57 ± 259.03	1077.63 ± 245.82
PPO + LSTM	Yes	20.03 ± 0.23	19.85 ± 0.39	900.20 ± 79.16	812.58 ± 111.51

Table 5.1: Comparison between the training and testing performance of PPO and PPO+LSTM, using a resized frame (84x84) of the input and also the full game frame (210x160) in both Atari games

In Table 5.1, we show the training and testing performances that PPO and PPO+LSTM had when using either the full image frame or the resized input (training graphics are also available in Appendix A, Figure A.1 and Figure A.2). The results shown do not allow us to conclude with absolute certainty that one way is better than the other. In Pong, there are not any significant differences in performance either in regular PPO or in PPO+LSTM. In SpaceInvaders, for the regular PPO, the agent that uses the full image achieves average returns that are around 200 points better than the agent that resizes the input. This result indicates that, for this game, some useful information is lost during the resizing of the image. However, for the model that uses PPO+LSTM, the opposite is verified. Since the size of the LSTMs was the same in both cases, this suggests that for the full image, the LSTM needed to be bigger because it could extract better data from fewer pixels.

	CarRacing-v0	
Model	Max. Train Avg.	Test Avg.
PPO	867.16 ± 6.64	824.31 ± 8.04
PPO+LSTM	783.62 ± 11.58	659.73 ± 24.42

Table 5.2: Comparison between the training and testing performance of PPO and PPO+LSTM using the full game image (96x96) in the CarRacing game

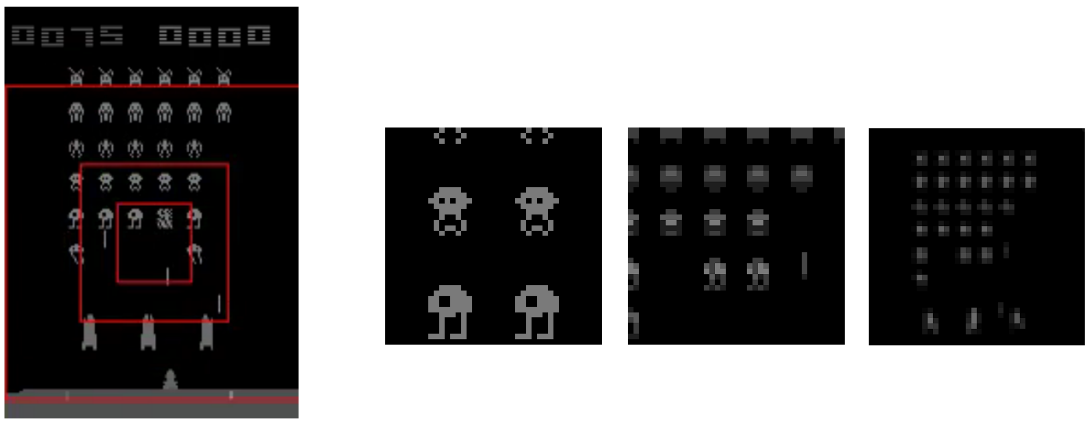
Table 5.1 and Table 5.2 also present a comparison between the PPO and the PPO+LSTM agents for the same type of input (the training graphic for the CarRacing game is also available in Figure A.3 from Appendix A). When analyzing their performance in the three games, we can see that the use of an LSTM deteriorates the performance of the agent. Although in Pong, the difference is marginal because it only means that on average instead of our agent winning most games with a 21-0 score, it wins them by 21-1. In SpaceInvaders, the performance drops by half, which is a significant reduction. This difference in performance in SpaceInvaders can also be verified in the Stable-Baselines3's [61] implementation of these two PPO versions. In CarRacing, there is also a drop in performance, even though it is less accentuated than in SpaceInvaders. The major difference between the PPO and PPO+LSTM models is that the former uses a stack of four frames as input, while the latter receives just one frame at a time, counting on its LSTM to discover and store the information that is useful to the agent. Therefore, in SpaceInvaders and in CarRacing, the LSTM is not able to store all the information needed, like the velocity and direction of the objects from the game, which would allow the agent to perform better. In CarRacing, the performance drop is less noticeable than in SpaceInvaders because the game frame is smaller (96x96 vs. 210x160).

In short, from these results, we cannot conclude that using the resized frame is better than using the full frame, and since our model utilizes the full image, in order to make the comparisons fairer, with as many equal variables as possible, from here onwards, we will be referring to the version that uses an LSTM and the entire frame as input when mentioning the base model.

5.3 GBAC Performance Analysis

In this section, we study the impact that different sizes of glimpses and different numbers of patches have on the performance of our agent, as well as the evolution of the choices of the glimpse locations and their comparison with human behavior.

5.3.1 Glimpse Sizes Study



(a) Example of a 40x40 glimpse with three patches in SpaceInvaders.

(b) Example of the resized patches the model receives from such glimpse.

Figure 5.1: Representation of a glimpse with three patches in SpaceInvaders

In our architecture, each glimpse can have one or more patches. Since we stipulated that each new patch has double the size of the previous, increasing the number of patches results in glimpses with a smaller focus region. This means that if we want glimpses with two patches, the largest possible size for the smallest patch is 80x80, with three patches it will be 40x40, and with four patches 20x20. Figure 5.1 demonstrates an example of a 40x40 glimpse with three patches and how the model sees the largest patches after being rescaled.

The higher the number of patches, the larger the “peripheral vision” of our model. Nonetheless, this increase in information comes with the price of it not being as detailed as the portions of the image closer to the focal point.

With this in mind, besides their architectures, using glimpses with just one patch makes our model no different from the base PPO agents. Therefore, the results from the agents that use glimpses with one patch are just useful to compare the two architectures, and to discover how large the input image has to be, in order for the agent to maintain its performance. Therefore, the results that are relevant to understand the performance of our model are the ones given by configurations that use more than one patch.

Tables 5.3 and 5.4 present the maximum training average and the testing performance of the best glimpse size for each number of patches tried. A more detailed version where every possible glimpse size is tested for each number of patches is available in Appendix A, in Table A.1 and Table A.2.

In the Pong Atari game, since the returns for this game are bounded between -21 and +21, when analyzing the performance in the runs for each combination, we can see that one of three scenarios occurred: the agent learned how to beat its adversary and ended up with scores close to +20; the agent did not manage to learn anything useful, resulting in scores around -19; or the timesteps were not enough for the agent to learn a good policy so it achieved a score between the previous two.

Regarding the glimpses with one patch, there is a clear difference in performance between the glimpses with size 160x160 and all the others. In Pong, our agent did not manage to achieve scores near +20 with the larger glimpse consistently because, in two of the three runs, the agent was still improving its performance when the timesteps finished. Nonetheless, in SpaceInvaders the average score of our model was closer to the one seen previously by the PPO+LSTM agent. In CarRacing, the glimpse size with one patch that performed best was also the largest, 96x96, and it was capable of matching the performance of the base agent in testing, beating it during training. Here, we see that the largest glimpse size possible was the one that achieved the highest results for the three game. This trend is expected since those are the sizes that gather the most number of pixels, which implies more information for the agents to work with. Since we are just using one patch, the agent cannot gather the information that is outside the glimpse size like it can when using glimpses with more than one patch.

Overall, these results prove that our agent is still capable of performing relatively well, even though not consistently, when the entire image is provided.

		PongNoFrameskip-v4		SpaceInvadersNoFrameskip-v4	
No. Patches	Glimpse Size	Max. Train Avg.	Test Avg.	Max. Train Avg.	Test Avg.
1	160	7.61 $\pm$ 10.42	6.95 $\pm$ 10.89	741.45 $\pm$ 392.54	607.42 $\pm$ 287.84
2	80	7.15 $\pm$ 20.80	6.64 $\pm$ 21.14	444.18 $\pm$ 40.78	341.47 $\pm$ 25.71
3	40	20.06 $\pm$ 0.44	19.82 $\pm$ 0.88	596.50 $\pm$ 182.35	544.43 $\pm$ 166.79
4	20	-14.86 $\pm$ 5.39	-15.77 $\pm$ 4.82	439.72 $\pm$ 26.27	378.00 $\pm$ 28.70

Table 5.3: Training and testing performance of the best glimpse size for every number of patches tested in Pong and SpaceInvaders.

		CarRacing-v0	
No. Patches	Glimpse Size	Max. Train Avg.	Test Avg.
1	96	815.12 $\pm$ 5.75	660.02 $\pm$ 69.38
2	40	694.50 $\pm$ 107.94	641.11 $\pm$ 57.42
3	20	676.82 $\pm$ 84.89	564.00 $\pm$ 56.42

Table 5.4: Training and testing performance of the best glimpse size for every number of patches tested in CarRacing.

Now, we will take a look at the results that are relevant to solve our problem, that is, the ones that refer to using more than one patch. We see that in Pong, the performance when using glimpses with two patches was slightly better than the previous because the standard deviation is much higher. The model achieved a score of around +19 in two of the three runs. This slight difference might simply be due to the random variance of the tests. However, in SpaceInvaders the results dropped almost by half. This drop in performance from one patch to two, only happened in this game and we do not have a explanation for it. In CarRacing, the achieved results maintain the performance seen while using one patch, although with a much higher standard deviation during training.

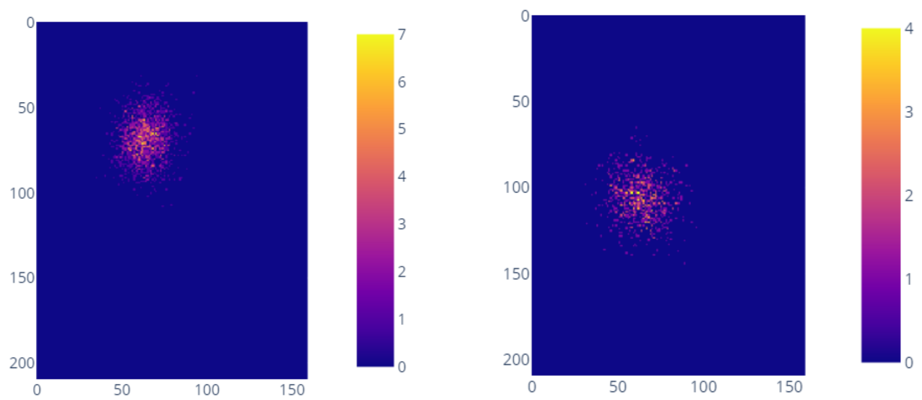
In Pong, using three patches was the combination that achieved the best performance, being our agent consistently capable of beating its opponent by 21-1. In SpaceInvaders, it was capable of performing better than when using two patches but not as good as with one patch. However, in CarRacing, we start to see the performance drop slightly in testing. In CarRacing, when compared to the Atari games, the different in performance between each number of patches is very low. This may suggest that this game might be easier to learn that the others.

Regarding the Pong game, after seeing the results until this point, we might think that the performance keeps improving while we increase the number of patches of each glimpse. However, this is not the case when we look at the performance when using four patches. In Pong, our model was not capable of achieving a performance of +20 in, at least, one of the three runs, which was true in any of the previous results. Regarding SpaceInvaders, the decrease in performance was not as severe as in Pong, and the standard deviation in both training and testing was one of the smallest of any of the experiments. We believe this drop in performance is due to the fact that the model may have reached the point where the glimpses start being so small and the larger patches have such lower resolution that it has difficulties extracting useful information from them.

From this study, we can conclude that the performance of GBAC does not increase linearly with the number of patches. It reaches a point where the information lost with the reduction of the glimpse size is more significant than the information gained with the addition of another patch. The optimal number of patches is three for the Atari games and two for CarRacing. Those numbers of patches proved to be the right balance between having a patch size that discarded the irrelevant information, allowing the model to just focus on the most important, and not being too small such that after rescaling the larger patches, it was still possible to understand what was present in the ”peripheral vision” of the agent. Finally, we saw that in most cases, the largest glimpse size possible for each number of patches is the one that produces the best results. In CarRacing, for two and three patches this was not the case, and the sizes 48x48 and 24x24, respectively, did not give the best results, even though they were very close.

5.3.2 Glimpse Location Analysis

After discovering which glimpse sizes performed best, it is also important to understand how the location of these glimpses is evolving throughout the training, and which behavior the model finds outs to be the best. While making this analysis, we also compare the agent's decisions with the choices we would consider when playing the games.



(a) Glimpse coordinates heatmap for Pong

(b) Glimpse coordinates heatmap for SpaceInvaders

Figure 5.2: Heatmaps representating the amount of times during an episode that the agent chose a specific set of image coordinates to be the center of the glimpse.

Regarding the regions that our best agents chose to look at during the episodes, we can see in Figure 5.2 that, for the Atari games, their distribution is relatively similar in both games. In SpaceInvaders, our agent keeps its focus near the center of the image, while in Pong, it chooses locations slightly upwards from the center. In general, the distribution of locations in both games has more choices closer to the center and becomes more sparse the further they are from it.

Having the focus point almost near the center of the image, as we can see in Figure 5.2(a), means that, in Pong, the agent gets the location of each paddle from its ”peripheral vision” (Figure 5.3). We think that this choice is different from what a human would select to focus on in this particular game. When we played, we paid more attention to the position of our own paddle and the ball.

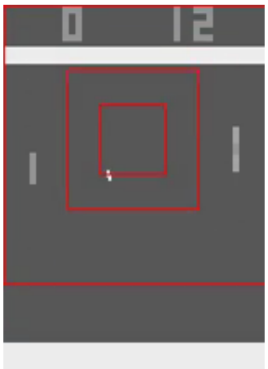
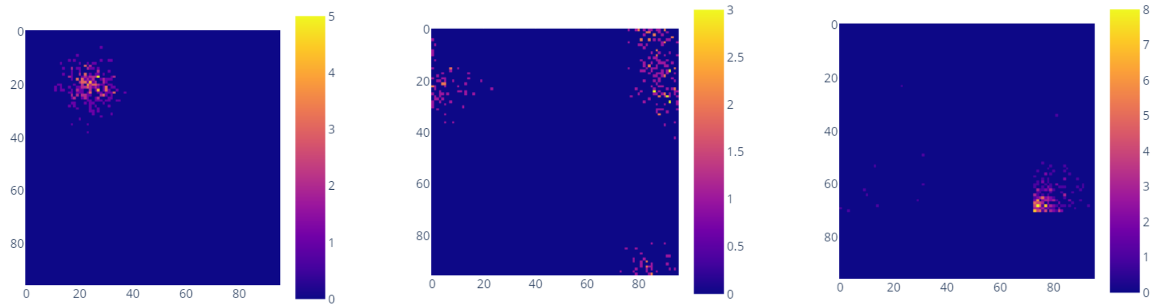


Figure 5.3: Example of a 40x40 glimpse with three patches in Pong.

In relation to SpaceInvaders (Figure 5.2(b)), in our opinion, the choices are much closer to what a human would do because the agent keeps its focus on the lower rows of enemies (Figure 5.1(a)), which are the ones that need to be destroyed first.

Regarding Car Racing, our model presents quite unusual behavior during the training process, which we could not find a plausible reason for. It starts similarly to the other two games with a circular normal distribution for the location coordinates (Figure 5.4(a)). However, after a few training epochs, that distribution starts dispersing over multiple parts of the entire image (Figure 5.4(b)). Then, it ends up in just one region of the image but creates some kind of borders, which constrain the choice of coordinates, and that do not correspond to the limits of the image (Figure 5.4(c)). This last behavior is the one that is present in the best solution at the end of training.

In our opinion, even though in two of the three games, our model does not follow exactly what we think is the human vision behavior when playing those video games, we would need a more systematic way of analyzing the regions that we select to focus our attention, using, for example, an eye-tracking device, to better compare the agent's behavior in relation to humans.



(a) Beginning of training **(b)** Right after the beginning **(c)** Rest of training

Figure 5.4: Heatmaps representing an example of the evolution of choices the agent made for the location of its glimpses during training in the CarRacing game.

5.4 Base Models Comparison

In this section, we compare, during training and testing, GBAC and the three PPO models described earlier. Besides comparing it with the two PPO versions discussed earlier, we also built a PPO+LSTM variant that uses our glimpses as input, but the locations for those glimpses are chosen randomly.

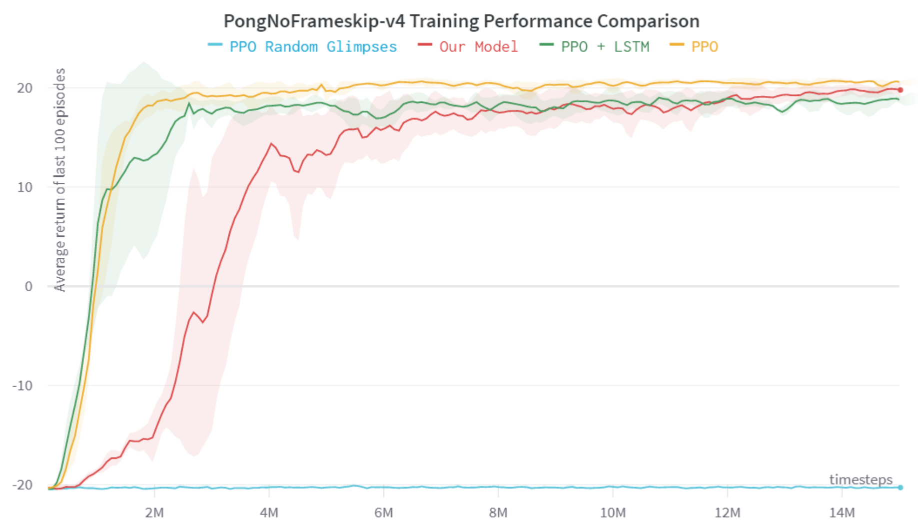


Figure 5.5: Evolution of the average return over the last 100 training episodes of Pong across three different runs, for the four models compared.

Table 5.5 presents a comparison between the performance of the PPO agents and our model in the Pong game. From it, we can conclude that GBAC was capable of matching the performance of the PPO+LSTM agent, consistently beating its opponent by 21-1. The difference to the regular PPO agent, only means that our agents let the opponent score a point, while the other did not. In Figure 5.5, we see that, during training, the two models that use an LSTM have a much higher variance in their average

Model	No. Patches	Glimpse Size	PongNoFrameskip-v4 Max. Train Avg.	Test Avg.
PPO	-	-	20.91 ± 0.11	20.83 ± 0.13
PPO + LSTM	-	-	20.03 ± 0.23	19.85 ± 0.39
GBAC	3	40	20.06 ± 0.44	19.82 ± 0.88
PPO Random Glimpses	3	40	-19.87 ± 0.05	-20.16 ± 0.11

Table 5.5: Comparison of the performance during training and testing achieved in Pong, by the four tested models.

episodic returns than the base PPO. Despite our model taking longer to reach good scores, it end ups training between the two other models.

Regarding the PPO agent that took glimpses at random locations, we see that it performed very poorly, not being able to learn an optimal policy to play Pong. Therefore, we can say that, in Pong, choosing good glimpse locations matters.

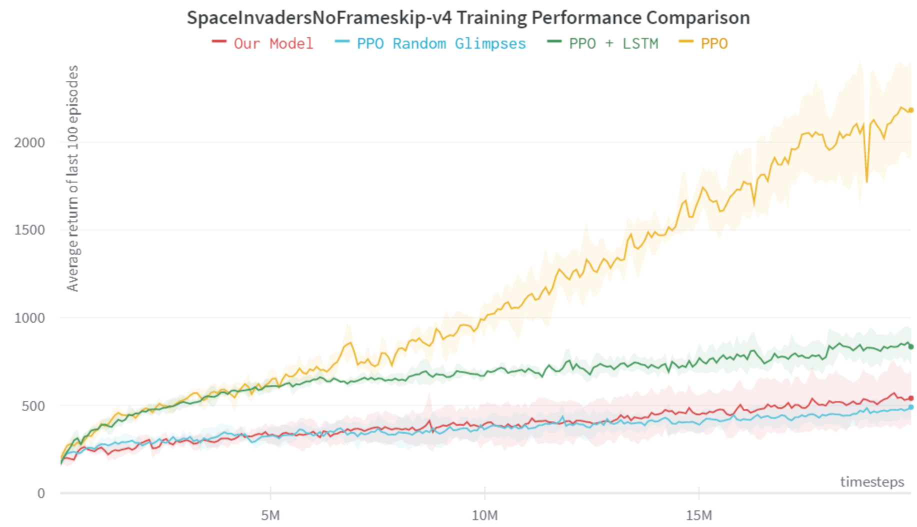


Figure 5.6: Evolution of the average return over the last 100 training episodes of SpaceInvaders across three different runs, for the four models compared.

Model	No. Patches	Glimpse Size	SpaceInvadersNoFrameskip-v4 Max. Train Avg.	Test Avg.
PPO	-	-	2261.90 ± 295.87	2221.62 ± 201.31
PPO + LSTM	-	-	900.20 ± 79.16	812.58 ± 111.51
GBAC	3	40	596.50 ± 182.35	544.43 ± 166.79
PPO Random Glimpses	3	40	516.62 ± 60.59	467.38 ± 50.53

Table 5.6: Comparison of the performance during training and testing achieved in SpaceInvaders, by the four tested models.

Figure 5.6 and Table 5.6 present the results in SpaceInvaders. Besides already having a big loss of performance when using an LSTM instead of stacking frames, our model is not capable of matching the scores of the PPO+LSTM agent. However, we still consider it an interesting result, considering the viewing restrictions of our problem. While processing 86% fewer pixels than PPO+LSTM (4.800 vs. 33.600) in each timestep, GBAC only had a performance drop of 33%.

Unlike the previous game, this time the PPO model has a larger variance than the PPO+LSTM agent. However, our model is still the one with the largest variance of the four.

In this game, the random agent has a performance that is on par with our model, which possibly means that in SpaceInvaders the location of a glimpse is not that important. One possible reason behind this proximity could be the fact that, in this game, GBAC can pay attention to a lot more things that can improve the return received from the environment. As opposed to Pong, where our agent only had three objects (two paddles and one ball) to keep track of.

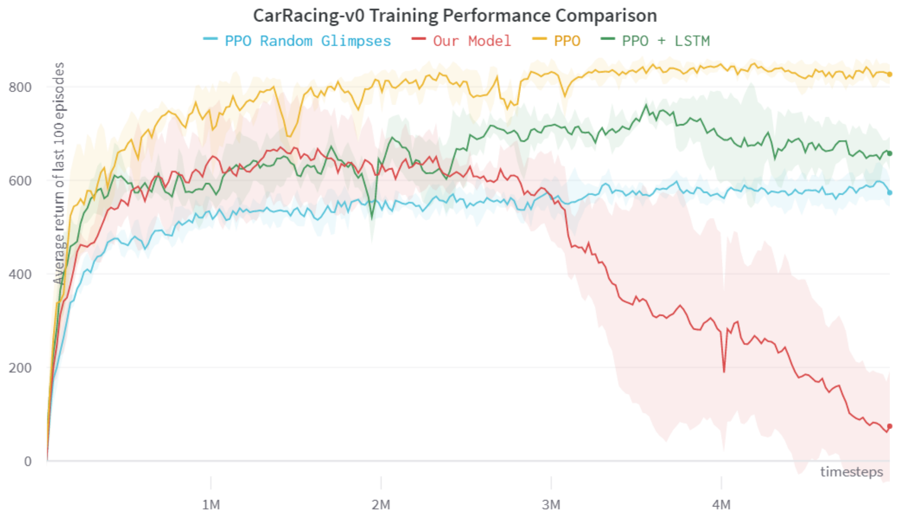


Figure 5.7: Evolution of the average return over the last 100 training episodes of CarRacing across three different runs, for the four compared models.

			CarRacing-v0	
Model	No. Patches	Glimpse Size	Max. Train Avg.	Test Avg.
PPO	-	-	867.16 $\pm$ 6.64	824.31 $\pm$ 8.04
PPO + LSTM	-	-	783.62 $\pm$ 11.58	659.73 $\pm$ 24.42
GBAC	2	40	694.50 $\pm$ 107.94	641.11 $\pm$ 57.42
PPO Random Glimpses	2	40	622.41 $\pm$ 17.89	589.87 $\pm$ 13.19

Table 5.7: Comparison of the performance during training and testing achieved in CarRacing, by the four tested models.

Finally, Table 5.7 and Figure 5.7 show the performance of all the agents in CarRacing. We can see that the performance drop from PPO to PPO+LSTM is much less accentuated, and our agent can match the performance of the latter, having again the largest variance of the four models. However, in this game, we see in Figure 5.7 that our model starts losing its good scores after the middle of training. This may be due to that strange behavior when selecting the glimpse locations presented in the previous section.

Regarding the PPO with random glimpses, the performance difference is again not that far behind our model. Since the generated track occupies a good portion of the image, it may be easier for the random model to select good actions from every glimpse it receives.

In conclusion, we see that our model is capable of matching the performance of the PPO+LSTM agent in two of the three games, while just using a portion of the game frame. An important fact we should also mention is that, since the complexity of our model is independent of the size of the input, we can achieve these results with a model that, in the Atari games, has 15% fewer total training parameters ($\sim$1.7M vs. $\sim$2.0M) than the PPO+LSTM model, and almost the same number has PPO. In CarRacing, the model is bigger, but it still has 10% fewer parameters than PPO+LSTM ($\sim$2.2M vs. $\sim$2.5M) and only more 70k than PPO. If we use bigger environments having frames with many more pixels, this difference between the number of training parameters required will only keep increasing.

5.5 Hard Attention vs. Soft Attention

In Section 3.2.1, we presented a soft attention model from Tang et al. [8] that divides the entire input image into multiple squares, assigns a level of importance to each one, and selects the N most important, which will be used by the model. Since this process refers to the other type of attention used in the literature, we found it was relevant to compare our model to it too. Although the model was only tested in CarRacing and DoomTakeCover, the code from the authors is available, thus we modified the model to also support Atari games.

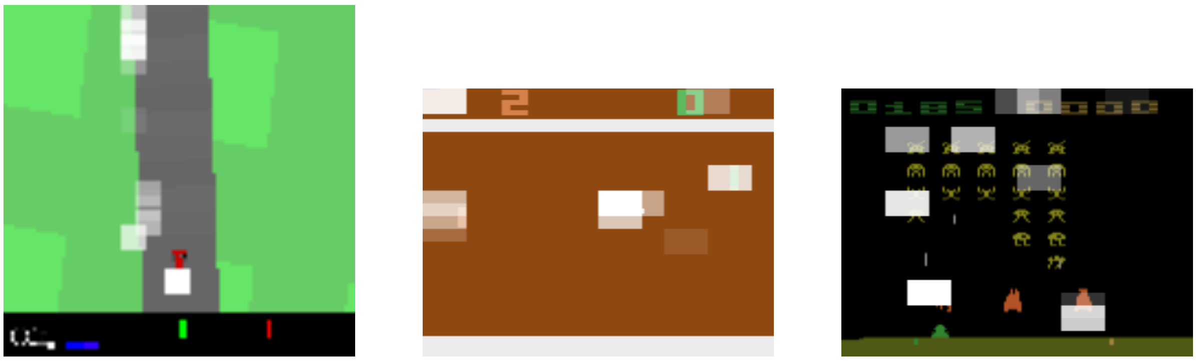
The model uses CMA-ES to train the controller, which is a very computationally intensive method that takes several days to run. Since we did not have either the computational power or the time to complete an entire session of training, we decided to run the soft attention model for as many hours as GBAC took to train. Therefore, both models had similar computational time which allows us to test their efficiency.

Table 5.8 compares the testing performance of the two attention models and shows the results presented by Tang et al. in their paper [8], as well as the scores we achieved with our training.

Model	CarRacing-v0	PongNoFrameskip-v4	SpaceInvadersNF-v4
Soft Attention [8] (paper)	914.00 $\pm$ 15.00	-	-
Soft Attention [8] (trained by us)	519.29 $\pm$ 293.53	-20.16 $\pm$ 0.87	67.66 $\pm$ 113.96
GBAC	641.11 $\pm$ 57.42	19.82 $\pm$ 0.88	544.43 $\pm$ 166.79

Table 5.8: Comparison between the average testing performance of each model during 100 episodes

From the results presented in Table 5.8, we can see that, regarding CarRacing, our model is not capable of achieving the same performance as the soft attention agent, which beats the required score of 900, established by the creator of the game. The version trained by us got a worse average than our model, but since it has such a high standard deviation, it might suggest that if we trained it for more time, it would also be capable of beating our model. Figure 5.8 presents the top ten patches the model found to be the most important during each frame, when playing each one of the games. Regarding CarRacing, Figure 5.8(a) shows the model focuses its attention on the borders of the track and on the car itself. This behavior matches the one presented by the authors.



(a) CarRacing **(b)** Pong **(c)** SpaceInvaders

Figure 5.8: Illustration of the 10 patches the soft attention model selected as the most important regions of the frame to look at

In the two Atari games, while our model is capable of achieving almost the highest score possible in Pong and has a decent performance in SpaceInvaders, we discovered that the soft attention model performs poorly, not being able to learn a good policy to play either Pong or SpaceInvaders. In Pong, Figure 5.8(b) shows that the model recognizes both paddles and the ball has the most important regions to look at. However, it also focuses its attention on the top of the frame where the current score is located. In SpaceInvaders, Figure 5.8(c) illustrates that the model, as opposed to GBAC, does not focus on the bottom row of the enemies, preferring instead to pay attention to the top rows and the objects that are between the enemies and our spaceship, which just serve as a place to get protection from the shots fired by the enemies. Once again, we see the soft attention model focusing on the current score at the top of the frame.

Since in the CarRacing game, we do not see the model focus on the bottom part of the frame where the score and metrics (steering, gas, and braking) are, we suspect that one of the reasons for the poor performance in the Atari games might be the focus on the scoreboard. Another suspicion might be that, like we have seen in GBAC, the CarRacing game seems to be an easier game to learn, thus if we made the controller in this model bigger, it might have been able to learn better how to play Pong and SpaceInvaders. Our final suspicion is related to the hyperparameters used in both those games. Since we did not make an exhaustive test for either the patch sizes to split the entire image frame or other hyperparameters, there is also the possibility of existing more performance that we could not extract.

Due to our limitations, from the results shown in this section, we cannot conclude that one attention mechanism is better than the other because we have mixed results depending on the game. However, we can say that attention mechanisms, which do not use the entire image as input, are showing promising results and are already capable, in some games, of at least matching the performance of regular RL algorithms.

6

Conclusion

Contents

6.1 Future Work 59

This thesis proposed a solution for the problem of an agent that has limited vision, and for that reason, besides deciding which action it has to take in the environment, it also has to choose which part of the environment it should look at.

As far as we are aware, there are not many papers in the literature regarding this exact problem, so we adapted a hard attention mechanism from an existing model called RAM, which applied it to image classification. To train both that mechanism and the policy that chooses the actions to play the game, we used PPO. Traditionally, this model-free RL algorithm has to receive the entire image as input, forcing it to process information that might not be useful, which ends up increasing the number of training parameters of the model because they are proportional to the size of the input. To minimize this problem, these models use low-resolution images as input, which normally restricts them on the games they are able to play and that are not representative of real-world tasks.

With the aim of trying to solve this problem, we proposed our model, whose number of parameters is independent of the size of the input image, only depending on the size of the glimpses and their number of patches.

Firstly, we proved that, for some games like Pong and CarRacing, our model is already capable of achieving similar performance to the PPO version that more closely resembles our model, that is, the variant that also uses an LSTM. On other games like SpaceInvaders, a drop in performance is verified, with means, there still is room for improvement.

Secondly, we concluded that our model does not necessarily choose the same regions of the image that we selected to look at when we played the video games ourselves. Only in SpaceInvaders we considered this was verified. In addition, in CarRacing, we are not able to explain the reason behind the unusual behavior of our model.

Finally, we understood that with the limitations we had, we were not able to conclude if a hard attention mechanism is better than a soft attention mechanism, or vice versa.

6.1 Future Work

Future work could comprise a thorough study of the multiple parameters used not only by the PPO algorithm but also by the attention mechanism. For example, during this work, we already realized that changing the kernel and stride values for the CNN present in the Glimpse Network can have a significant impact on the performance of the model, depending on the size of the glimpses used.

Another problem found that needs further investigation is the reason behind the high variance of this model, when compared to the base PPO algorithms. In all the games tested, our model was always the one with the highest variance. Therefore, solutions to reduce this variance could be addressed.

In addition, one suggestion that could improve our comparison between the glimpse movements of our model and human behavior, would be asking a group of people to play the video games used and record their eye movements using an eye-tracking device. This approach would make this comparison pass from only opinion-based to a more measurable and realistic metric.

Lastly, in this thesis we proposed a model-free agent and discovered that it already has an interesting performance. We can now also question how would a model-based RL agent perform with these same visual limitations.

Bibliography

[1] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, 2nd ed. MIT Press, 11 2018.

[2] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[3] R. Polvara, M. Patacchiola, S. Sharma, J. Wan, A. Manning, R. Sutton, and A. Cangelosi, "Autonomous quadrotor landing using deep reinforcement learning," *arXiv preprint arXiv:1709.03339*, 9 2017.

[4] D. Ha and J. Schmidhuber, "Recurrent world models facilitate policy evolution," in *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, ser. NIPS'18. Red Hook, NY, USA: Curran Associates Inc., 2018, p. 2455–2467.

[5] I. Sorokin, A. Seleznev, M. Pavlov, A. Fedorov, and A. Ignateva, "Deep attention recurrent q-network," *arXiv preprint arXiv:1512.01693*, 12 2015.

[6] J. Ba, V. Mnih, and K. Kavukcuoglu, "Multiple object recognition with visual attention," in *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

[7] V. Mnih, N. Heess, A. Graves, and K. Kavukcuoglu, "Recurrent models of visual attention," in *Proceedings of the 27th International Conference on Neural Information Processing Systems*, ser. NIPS'14, vol. 2. Cambridge, MA, USA: MIT Press, 2014, p. 2204–2212.

[8] Y. Tang, D. Nguyen, and D. Ha, "Neuroevolution of self-interpretable agents," in *Proceedings of the 2020 Genetic and Evolutionary Computation Conference*, ser. GECCO '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 414–424.

[9] C. E. Connor, H. E. Egeth, and S. Yantis, "Visual attention: Bottom-up versus top-down," *Current Biology*, vol. 14, no. 19, pp. R850–R852, 2004.

[10] D. L. Schacter, D. T. Gilbert, D. M. Wegner, and B. M. Hood, *Psychology*, 2nd ed. Palgrave, 2016.

[11] P. Stone, R. Brooks, E. Brynjolfsson, R. Calo, O. Etzioni, G. Hager, J. Hirschberg, S. Kalyanakrishnan, E. Kamar, S. Kraus, K. Leyton-Brown, D. Parkes, W. Press, A. Saxenian, J. Shah, M. Tambe, and A. Teller, "Artificial intelligence and life in 2030: the one hundred year study on artificial intelligence," Stanford University, Tech. Rep., 9 2016.

[12] A. Voulodimos, N. Doulamis, A. Doulamis, E. Protopapadakis, and D. Andina, "Deep learning for computer vision: A brief review," *Computational Intelligence and Neuroscience*, vol. 2018, jan 2018.

[13] A. Katharopoulos and F. Fleuret, "Processing megapixel images with deep attention-sampling models," in *Proceedings of the 36th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, vol. 97. PMLR, 2019, pp. 3282–3291.

[14] V. Thambawita, I. Strümke, S. A. Hicks, P. Halvorsen, S. Parasa, and M. A. Riegler, "Impact of image resolution on deep learning performance in endoscopy image classification: An experimental study using a large dataset of endoscopic images," *Diagnostics*, vol. 11, no. 12, 2021.

[15] S. Ghaffarian, J. Valente, M. van der Voort, and B. Tekinerdogan, "Effect of attention mechanism in deep learning-based remote sensing image processing: A systematic literature review," *Remote Sensing*, vol. 13, no. 15, 2021.

[16] F. Wang and D. M. J. Tax, "Survey on the attention based rnn model and its applications in computer vision," *arXiv preprint arXiv:1601.06823*, 1 2016.

[17] X. Yang, "An overview of the attention mechanisms in computer vision," *Journal of Physics: Conference Series*, vol. 1693, no. 1, p. 012173, dec 2020.

[18] R. D. Smallwood and E. J. Sondik, "The optimal control of partially observable markov processes over a finite horizon," *Operations research*, vol. 21, no. 5, pp. 1071–1088, 1973.

[19] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Machine Language*, vol. 8, no. 3–4, p. 229–256, may 1992.

[20] H.-G. Beyer and H.-P. Schwefel, "Evolution strategies –a comprehensive introduction," *Natural Computing: An International Journal*, vol. 1, no. 1, p. 3–52, may 2002.

[21] N. Hansen and A. Ostermeier, "Completely derandomized self-adaptation in evolution strategies," *Evolutionary Computation*, vol. 9, no. 2, p. 159–195, jun 2001.

[22] R. Bajcsy, “Active perception,” *Proceedings of the IEEE*, vol. 76, no. 8, pp. 966–1005, 1988.

[23] E. S. Lee, “Active perception with neural networks,” *arXiv preprint arXiv:2109.02744*, 9 2021.

[24] R. Bajcsy, Y. Aloimonos, and J. K. Tsotsos, “Revisiting active perception,” *Autonomous Robots*, vol. 42, no. 2, p. 177–196, feb 2018.

[25] L.-J. Lin, “Self-improving reactive agents based on reinforcement learning, planning and teaching,” *Machine Learning*, vol. 8, no. 3, pp. 293–321, May 1992.

[26] H. v. Hasselt, A. Guez, and D. Silver, “Deep reinforcement learning with double q-learning,” in *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, ser. AAAI'16. AAAI Press, 2016, p. 2094–2100.

[27] Z. Wang, T. Schaul, M. Hessel, H. Van Hasselt, M. Lanctot, and N. De Freitas, “Dueling network architectures for deep reinforcement learning,” in *Proceedings of the 33rd International Conference on International Conference on Machine Learning*, ser. ICML'16, vol. 48. JMLR, 2016, p. 1995–2003.

[28] M. Hessel, J. Modayil, H. van Hasselt, T. Schaul, G. Ostrovski, W. Dabney, D. Horgan, B. Piot, M. Azar, and D. Silver, “Rainbow: Combining improvements in deep reinforcement learning,” in *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence and Thirtieth Innovative Applications of Artificial Intelligence Conference and Eighth AAAI Symposium on Educational Advances in Artificial Intelligence*, ser. AAAI'18/IAAI'18/EAAI'18. AAAI Press, 2018.

[29] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Harley, T. P. Lillicrap, D. Silver, and K. Kavukcuoglu, “Asynchronous methods for deep reinforcement learning,” in *Proceedings of the 33rd International Conference on International Conference on Machine Learning*, ser. ICML'16, vol. 48. JMLR, 2016, p. 1928–1937.

[30] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. K. Openai, “Proximal policy optimization algorithms,” *arXiv preprint arXiv:1707.06347*, 7 2017.

[31] M. Zuur, “Deep reinforcement learning of active sensing strategies with pomdps,” Master's thesis, Radboud University - Faculteit der Sociale Wetenschappen, 8 2019.

[32] T. Wang, X. Bao, I. Clavera, J. Hoang, Y. Wen, E. Langlois, S. Zhang, G. Zhang, P. Abbeel, and J. Ba, “Benchmarking model-based reinforcement learning,” *arXiv preprint arXiv:1907.02057*, 7 2019.

[33] C. M. Bishop, “Mixture density networks,” Aston University, WorkingPaper, 1994.

[34] D. Hafner, T. L. Deepmind, J. Ba, and M. Norouzi, “Dream to control: Learning behaviors by latent imagination,” *arXiv preprint arXiv:1912.01603*, 12 2019.

[35] D. Hafner, T. P. Lillicrap, M. Norouzi, and J. Ba, “Mastering atari with discrete world models,” in *International Conference on Learning Representations*, 2021.

[36] D. Hafner, T. Lillicrap, I. Fischer, R. Villegas, D. Ha, H. Lee, and J. Davidson, “Learning latent dynamics for planning from pixels,” in *Proceedings of the 36th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, K. Chaudhuri and R. Salakhutdinov, Eds., vol. 97. PMLR, 09–15 Jun 2019, pp. 2555–2565.

[37] G. Zhu, M. Zhang, H. Lee, and C. Zhang, “Bridging imagination and reality for model-based deep reinforcement learning,” in *Proceedings of the 34th International Conference on Neural Information Processing Systems*, ser. NIPS'20. Red Hook, NY, USA: Curran Associates Inc., 2020, p. 8993–9006.

[38] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, Y. Chen, T. Lillicrap, F. Hui, L. Sifre, G. van den Driessche, T. Graepel, and D. Hassabis, “Mastering the game of Go without human knowledge,” *Nature*, vol. 550, no. 7676, pp. 354–359, Oct. 2017.

[39] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, T. Lillicrap, K. Simonyan, and D. Hassabis, “A general reinforcement learning algorithm that masters chess, shogi, and go through self-play,” *Science*, vol. 362, no. 6419, pp. 1140–1144, 2018.

[40] J. Schrittwieser, I. Antonoglou, T. Hubert, K. Simonyan, L. Sifre, S. Schmitt, A. Guez, E. Lockhart, D. Hassabis, T. Graepel, T. Lillicrap, and D. Silver, “Mastering Atari, Go, chess and shogi by planning with a learned model,” *Nature*, vol. 588, no. 7839, pp. 604–609, Dec. 2020.

[41] A. Galassi, M. Lippi, and P. Torroni, “Attention in natural language processing,” *IEEE Transactions on Neural Networks and Learning Systems*, vol. 32, no. 10, pp. 4291–4308, 2021.

[42] J. Chorowski, D. Bahdanau, D. Serdyuk, K. Cho, and Y. Bengio, “Attention-based models for speech recognition,” in *Proceedings of the 28th International Conference on Neural Information Processing Systems*, ser. NIPS'15, vol. 1. Cambridge, MA, USA: MIT Press, 2015, p. 577–585.

[43] A. Mott, D. Zoran, M. Chrzanowski, D. Wierstra, and D. J. Rezende, *Towards Interpretable Reinforcement Learning Using Attention Augmented Agents*. Red Hook, NY, USA: Curran Associates Inc., 2019.

[44] M. Hausknecht and P. Stone, “Deep recurrent q-learning for partially observable mdps,” *AAAI Fall Symposium Series*, 9 2015.

[45] E. Gilmour, N. Plotkin, and L. N. Smith, “An approach to partial observability in games: Learning to both act and observe,” in *2021 IEEE Conference on Games (CoG)*. IEEE Press, 2021, p. 01–05.

[46] M. Gregor, D. Nemec, A. Janota, and R. Pirník, “A visual attention operator for playing pac-man,” in *2018 ELEKTRO*. IEEE, 2018, pp. 1–6.

[47] H. Sahni and C. Isbell, “Hard attention control by mutual information maximization,” *arXiv preprint arXiv:2103.06371*, 3 2021.

[48] Y. Du and K. Narasimhan, “Task-agnostic dynamics priors for deep reinforcement learning,” in *Proceedings of the 36th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, K. Chaudhuri and R. Salakhutdinov, Eds., vol. 97. PMLR, 09–15 Jun 2019, pp. 1696–1705.

[49] D. Jayaraman and K. Grauman, “Learning to look around: Intelligently exploring unseen environments for unknown tasks,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018, pp. 1238–1247.

[50] J. Xiao, K. A. Ehinger, A. Oliva, and A. Torralba, “Recognizing scene viewpoint using panoramic place representation,” in *2012 IEEE Conference on Computer Vision and Pattern Recognition*, 2012, pp. 2695–2702.

[51] Z. Wu, S. Song, A. Khosla, F. Yu, L. Zhang, X. Tang, and J. Xiao, “3d shapenets: A deep representation for volumetric shapes,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015.

[52] S. K. Ramakrishnan and K. Grauman, “Sidekick policy learning for active visual exploration,” in *Computer Vision – ECCV 2018: 15th European Conference*. Berlin, Heidelberg: Springer-Verlag, 2018, p. 424–442.

[53] S. Seifi and T. Tuytelaars, “Where to look next: Unsupervised active visual exploration on 360° input,” *arXiv preprint arXiv:1909.10304*, 9 2019.

[54] D. P. Kingma and J. L. Ba, “Adam: A method for stochastic optimization,” *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*, 12 2014.

[55] J. Schulman, P. Moritz, S. Levine, M. I. Jordan, and P. Abbeel, “High-dimensional continuous control using generalized advantage estimation,” *4th International Conference on Learning Representations, ICLR 2016 - Conference Track Proceedings*, 6 2015.

[56] O. Klimov. (2016) Carracing-v0. [Online]. Available: https://gym.openai.com/envs/CarRacing-v0/

[57] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," 2016.

[58] M. C. Machado, M. G. Bellemare, E. Talvitie, J. Veness, M. Hausknecht, and M. Bowling, "Revisiting the arcade learning environment: Evaluation protocols and open problems for general agents," *Journal of Artificial Intelligence Research*, vol. 61, no. 1, p. 523–562, jan 2018.

[59] P. Dhariwal, C. Hesse, O. Klimov, A. Nichol, M. Plappert, A. Radford, J. Schulman, S. Sidor, Y. Wu, and P. Zhokhov, "Openai baselines," https://github.com/openai/baselines, 2017.

[60] S. Huang, R. F. J. Dossa, A. Raffin, A. Kanervisto, and W. Wang, "The 37 implementation details of proximal policy optimization," in *ICLR Blog Track*, 2022. [Online]. Available: https://iclr-blog-track.github.io/2022/03/25/ppo-implementation-details/

[61] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dormann, "Stable-baselines3: Reliable reinforcement learning implementations," *Journal of Machine Learning Research*, vol. 22, no. 268, pp. 1–8, 2021. [Online]. Available: http://jmlr.org/papers/v22/20-1364.html

A

Extended Results

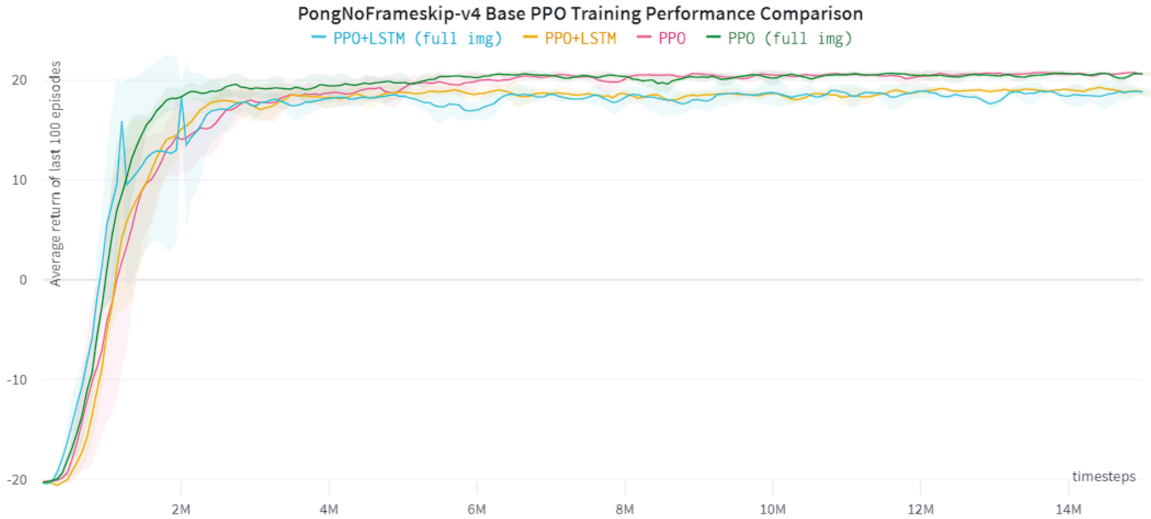


Figure A.1: Evolution of the average return over the last 100 training episodes of Pong across three different runs, for PPO and PPO+LSTM, using either the full image frame or a resized version.

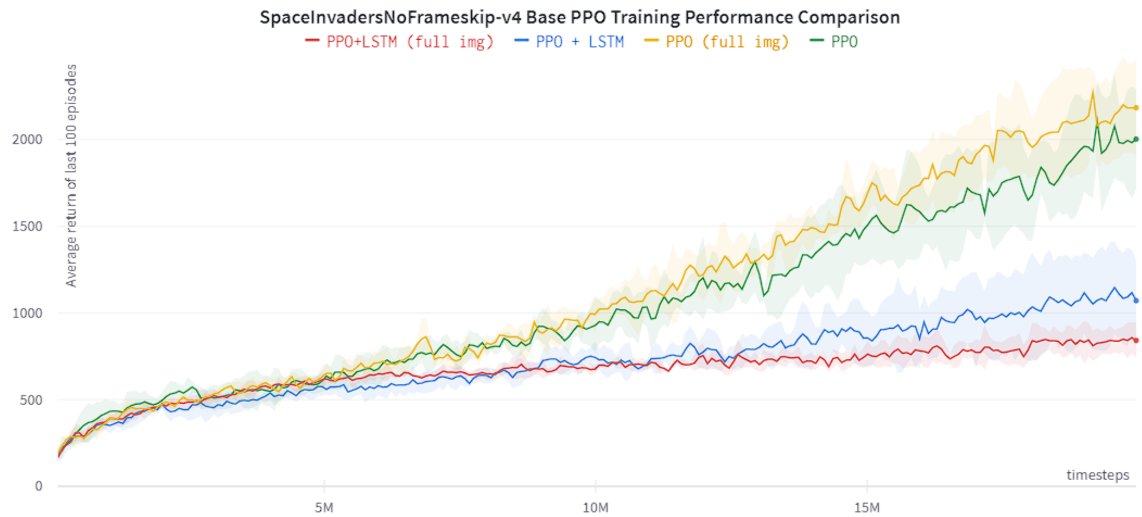


Figure A.2: Evolution of the average return over the last 100 training episodes of SpaceInvaders across three different runs, for PPO and PPO+LSTM, using either the full image frame or a resized version.

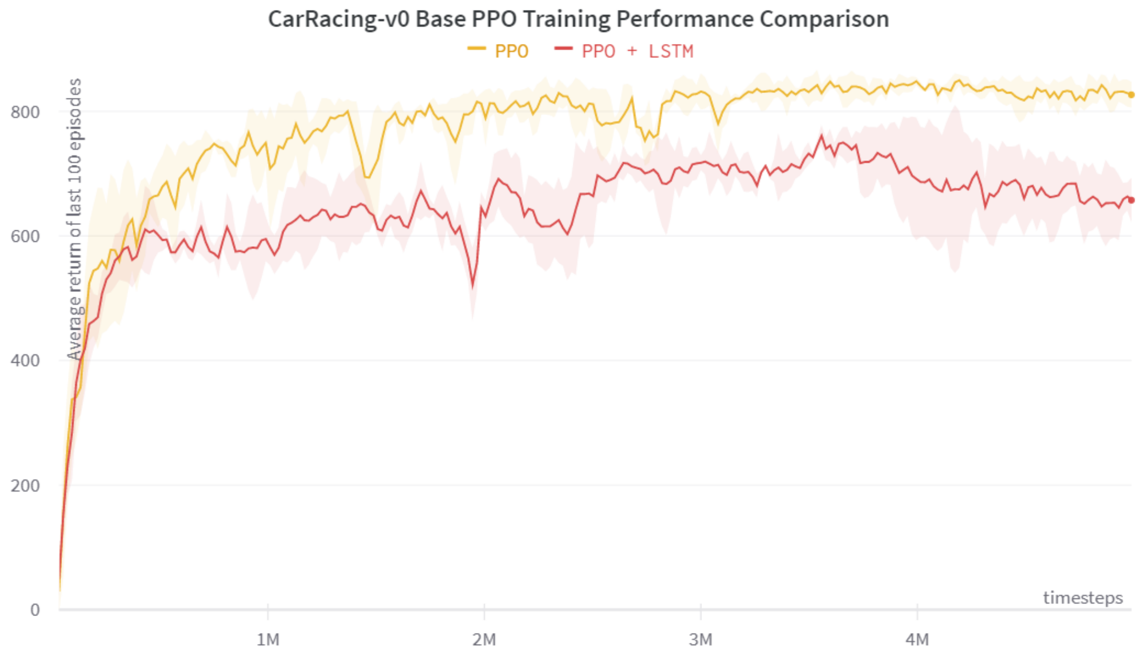


Figure A.3: Evolution of the average return over the last 100 training episodes of CarRacing across three different runs, for PPO and PPO+LSTM, only using the full image frame.

		PongNoFrameskip-v4		SpaceInvadersNoFrameskip-v4	
No. Patches	Glimpse Size	Max. Train Avg.	Test Avg.	Max. Train Avg.	Test Avg.
1	160	7.61 $\pm$ 10.42	6.95 $\pm$ 10.89	741.45 $\pm$ 392.54	607.42 $\pm$ 287.84
1	150	-18.96 $\pm$ 1.67	-19.32 $\pm$ 1.38	384.43 $\pm$ 72.19	338.45 $\pm$ 59.50
1	140	-18.93 $\pm$ 1.80	-19.36 $\pm$ 1.47	386.50 $\pm$ 75.94	347.08 $\pm$ 99.27
1	130	-19.92 $\pm$ 0.04	-20.19 $\pm$ 0.03	330.76 $\pm$ 77.16	265.00 $\pm$ 53.71
1	120	-19.86 $\pm$ 0.04	-20.06 $\pm$ 0.05	343.98 $\pm$ 137.35	273.00 $\pm$ 113.99
1	110	-19.89 $\pm$ 0.11	-20.22 $\pm$ 0.08	381.66 $\pm$ 85.91	340.12 $\pm$ 77.90
1	100	-19.89 $\pm$ 0.05	-20.24 $\pm$ 0.11	321.91 $\pm$ 51.51	256.78 $\pm$ 77.16
1	90	-19.65 $\pm$ 0.42	-19.90 $\pm$ 0.36	285.76 $\pm$ 61.01	214.35 $\pm$ 23.69
1	80	-19.90 $\pm$ 0.01	-20.28 $\pm$ 0.05	244.53 $\pm$ 7.94	190.83 $\pm$ 26.97
1	70	-18.47 $\pm$ 1.30	-18.88 $\pm$ 1.08	243.08 $\pm$ 23.85	186.83 $\pm$ 19.25
1	60	-19.89 $\pm$ 0.05	-20.16 $\pm$ 0.09	232.95 $\pm$ 17.10	214.58 $\pm$ 13.55
1	50	-19.63 $\pm$ 0.54	-19.80 $\pm$ 0.56	251.46 $\pm$ 33.61	190.62 $\pm$ 22.40
1	40	-19.93 $\pm$ 0.04	-20.19 $\pm$ 0.11	218.78 $\pm$ 2.93	163.42 $\pm$ 14.12
1	30	-19.95 $\pm$ 0.03	-20.17 $\pm$ 0.17	221.73 $\pm$ 4.20	170.33 $\pm$ 21.35
1	20	-19.75 $\pm$ 0.32	-20.11 $\pm$ 0.25	244.51 $\pm$ 4.90	203.97 $\pm$ 23.26
1	10	-19.87 $\pm$ 0.07	-20.23 $\pm$ 0.06	238.90$\pm$ 15.14	188.23 $\pm$ 32.58
2	80	7.15 $\pm$ 20.80	6.64 $\pm$ 21.14	444.18 $\pm$ 40.78	341.47 $\pm$ 25.71
2	70	-6.68 $\pm$ 22.92	-6.99 $\pm$ 22.80	371.68 $\pm$ 58.30	296.10 $\pm$ 36.34
2	60	-19.86 $\pm$ 0.05	-20.27 $\pm$ 0.18	371.21 $\pm$ 6.30	350.15 $\pm$ 10.13
2	50	-16.72 $\pm$ 5.48	-17.17 $\pm$ 5.20	283.81 $\pm$ 76.94	217.58$\pm$ 62.06
2	40	-19.91 $\pm$ 0.03	-20.19 $\pm$ 0.07	252.58 $\pm$ 22.31	194.52 $\pm$ 26.93
2	30	-17.24 $\pm$ 2.34	-17.52 $\pm$ 2.43	248.98 $\pm$ 5.78	208.03 $\pm$ 7.23
2	20	-19.87 $\pm$ 0.07	-20.17 $\pm$ 0.16	241.50 $\pm$ 10.81	190.52 $\pm$ 31.40
2	10	-19.91 $\pm$ 0.06	-20.24 $\pm$ 0.09	243.51 $\pm$ 5.16	209.43 $\pm$ 32.67
3	40	20.06 $\pm$ 0.44	19.82 $\pm$ 0.88	596.50 $\pm$ 182.35	544.43 $\pm$ 166.79
3	30	-10.99 $\pm$ 10.42	-11.80 $\pm$ 9.05	274.38 $\pm$ 5.24	212.07 $\pm$ 34.62
3	20	-19.92 $\pm$ 0.01	-20.12 $\pm$ 0.10	293.12 $\pm$ 28.46	228.63 $\pm$ 28.06
3	10	-19.93 $\pm$ 0.03	-20.35 $\pm$ 0.39	251.60 $\pm$ 12.27	194.13 $\pm$ 31.28
4	20	-14.86 $\pm$ 5.39	-15.77 $\pm$ 4.82	439.72 $\pm$ 26.27	378.00 $\pm$ 28.70
4	10	-19.86 $\pm$ 0.05	-20.23 $\pm$ 0.08	297.45 $\pm$ 9.98	252.77 $\pm$ 13.72

Table A.1: Training and testing performance of every glimpse size for every number of patches tested in Pong and SpaceInvaders.

		CarRacing-v0	
No. Patches	Glimpse Size	Max. Train Avg.	Test Avg.
1	96	815.12 ± 5.75	660.02 ± 69.38
1	90	675.12 ± 84.89	607.46 ± 39.81
1	80	741.03 ± 112.73	534.56 ± 72.77
1	70	747.12 ± 103.91	258.70 ± 274.03
1	60	692.40 ± 121.21	546.35 ± 40.01
1	50	559.05 ± 19.45	361.30 ± 73.27
1	40	539.68 ± 11.35	485.85 ± 9.80
1	30	187.67 ± 136.04	32.64 ± 81.82
1	20	155.15 ± 77.05	152.99 ± 75.43
1	10	138.51 ± 88.40	128.90 ± 94.17
2	48	748.70 ± 74.90	544.58 ± 132.04
2	40	694.50 ± 107.94	641.11 ± 57.42
2	30	616.74 ± 84.43	347.20 ± 243.31
2	20	465.76 ± 74.51	375.30 ± 164.83
2	10	161.44 ± 36.96	145.80 ± 48.40
3	24	700.53 ± 28.06	472.35 ± 295.73
3	20	676.82 ± 84.89	564.00 ± 56.42
3	10	545.18 ± 92.57	509.85 ± 70.74

Table A.2: Training and testing performance of every glimpse size for every number of patches tested in CarRacing.

B

Algorithms hyperparameters

Table B.1 and Table B.2 list the parameters that were used during training for the two Atari games and CarRacing, in the Glimpse-Based Actor-Critic and the soft attention model from Tang et al. [8].

	Hyperparameter	Value(s)
PPO	Advantage normalization	True
	Annealing learning rate	True
	Batch size	[1024, 2048]
	Clipping coefficient - action	[0.1, 0.2]
	Clipping coefficient - location	0.2
	Clipped value loss	True
	Entropy coefficient	[0.01, 0]
	GAE lambda	0.95
	Gamma	0.99
	Grayscale	True
	Learning rate - action	[2.5e-4, 3e-4]
	Learning rate - location	3e-5
	Locator normal distrib. variance	0.1
	Maximum gradient clipping norm	0.5
	Minibatch size	[256, 64]
	No. environments	[8, 1]
	No. minibatches	[4, 32]
	No. steps	[128, 2048]
	Optimizer	Adam
	Update k epochs	[4, 10]
	Value function coefficient	0.5
Glimpse	Glimpse scale	2
	Glimpse FC layer size	[384, 512]
	Location FC layer size	256
	LSTM size	128
	No. glimpses	1

Table B.1: List of parameters used in GBAC, in the Atari games and CarRacing, respectively

	Hyperparameter	Value(s)
CMA-ES	init_sigma	0.1
	n_repeat	[5, 16]
	population_size	256
	seed	0
Model	activation	Tanh
	data_dim	3
	l2_coefficient	0
	normalize_positions	True
	num_hiddens	16
	output_activation	Tanh
	output_dim	[6, 3]
	patch_size	[20, 7]
	patch_stride	[10, 4]
	query_dim	4
	top_k	10
	use_lstm_controller	True

Table B.2: List of parameters used in the soft attention model of Tang el al. [8], in the Atari games and CarRacing, respectively