From Pixels to Actions:
Streamlining Reinforcement Learning Strategies through Object Detection

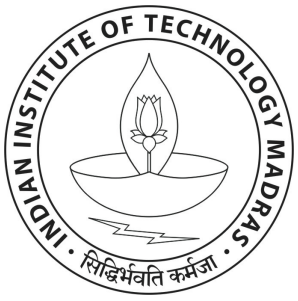
A Project Report

submitted by

JASHASWIMALYA ACHARJEE

in partial fulfilment of requirements
for the award of the degree of

BACHELOR OF TECHNOLOGY



DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY MADRAS

May 2023

THESIS CERTIFICATE

This is to certify that the thesis titled **From Pixels to Actions: Streamlining Reinforcement Learning Strategies through Object Detection-based Compact State Space Representations**, submitted by **Jashaswimalya Acharjee**, to the Indian Institute of Technology, Madras, for the award of the degree of **Bachelor of Technology**, is a bona fide record of the research work done by him under our supervision. The contents of this thesis, in full or in parts, have not been submitted to any other Institute or University for the award of any degree or diploma.

Prof. Balaraman Ravindran
Research Guide
Professor
Dept. of Computer Science
and Engineering
IIT-Madras, 600 036

Place: Chennai

Date: April 28, 2023

ACKNOWLEDGEMENTS

Completing this B.Tech. Project would not have been possible without the help, support, and guidance of numerous individuals and institutions. It is with deep gratitude that I extend my sincere appreciation to all those who contributed to this work.

First and foremost, I express my heartfelt gratitude to my thesis advisor, **Prof. Balaraman Ravindran**, for his invaluable guidance, insights, and support throughout the entire project. His unwavering encouragement, attention to detail, and constructive feedback were instrumental in shaping the direction and quality of this research.

I am also indebted to the faculty members of Computer Science and Engineering Department of Indian Institute of Technology Madras and National Institute of Technology Agartala, who have provided a rich academic environment and resources that enabled me to pursue my academic and research interests. Their collective expertise, insightful feedback, and encouragement were instrumental in shaping the quality of this work and myself included.

I would like to thank my colleagues and friends at both the institutes, Indian Institute of Technology Madras and National Institute of Technology Agartala, who provided an atmosphere of camaraderie and intellectual exchange that enriched my academic experience. Their support and encouragement kept me motivated during the ups and downs of this project.

Lastly, I would like to express my sincere thanks to my family for their unwavering love, support, and encouragement throughout my academic journey. Their sacrifices and encouragement were instrumental in helping me achieve my academic and personal goals.

Thank you all for your contributions to this project.

ABSTRACT

KEYWORDS: Reinforcement Learning ; HRL; meta-controller; object detection; ddqn; dqn; state space; reward; shaping.

Hierarchical Reinforcement Learning (HRL) has become a popular approach towards solving complex sequential decision-making problems. This paper explores the use of compressed state space representations with existing RL algorithms to address the challenges presented by the complex video game environments, such as Montezuma's Revenge and Super Mario Bros. These games are particularly difficult for traditional RL algorithms because they contain additional objectives to satisfy in order to complete the designated goal, as well as the requirement for long-term planning. These environments give rewards only after completion of the episode, resulting in very sparse rewards.

The proposed method utilizes an object detection model along with a hierarchy of policies associated with those objects that are learnt from repetitive game-plays and operate at different levels of abstraction. The top-level policy selects high-level goals, while the lower-level policies execute actions to achieve those goals. The TD learning algorithm is used to update the value functions for both the higher and lower-level policies.

The experiments conducted on Montezuma's Revenge and Super Mario Bros demonstrate that the proposed method performs without degradation when compared to traditional RL algorithms. Furthermore, the approach is shown to be more sample-efficient, which is particularly important when working with real-world applications.

Finally, the paper discusses recent advances in HRL and deep reinforcement learning to improve the scalability and flexibility of HRL algorithms. These developments are expected to enable the use of HRL in a wider range of applications and further improve the performance of HRL algorithms.

TABLE OF CONTENTS

LIST OF FIGURES

ABBREVIATIONS

RL	Reinforcement Learning
HRL	Hierarchical Reinforcement Learning
TD	Temporal Difference
DQN	Deep Q Network
DDQN	Double Deep Q Network
PPO	Proximal Policy Optimization
MDP	Markov Decision Process
SMDP	Semi-Markov Decision Process

NOTATION

s	State, s
s'	Next State, s'
$P(s'\|s,a)$	Probability of Transition to state s' if action a is performed at state s.
$R(s,a)$	Reward function
$V^{\pi}(s)$	Value function
$Q^{\pi}(s,a)$	Action-Value function
γ	Discount Factor

CHAPTER 1

INTRODUCTION

Reinforcement Learning Algorithms are a class of algorithms that enables "agents" to learn an optimal behaviour through iterative trial-and-error interactions with a given environment. RL Algorithms typically involve three main components — a Policy, a Value Function and a Reward Signal. The policy is what determines the agent's behaviour when it is allowed to act in an environment, the value function estimates the long-term rewards the agent can expect to receive in the future if the given policy is followed by the agent. The reward signal provides feedback on the quality of the agent's chosen action. At this point it is quite evident that in both discrete and continuous setting, the sheer volume of observations/state-space and their value functions and policies corresponding to each of the state will eventually exceed the computational feasibility metric. Hence one can state that Reinforcement Learning is ill-fated by the curse of dimensionality — the number of parameters to be learned grows exponentially with the size of the state and action space representations. Some of the popular works for tackling this curse, mostly exploit the temporal abstraction, such that decisions are not required at each timestep or they deal with temporally extended sub-objectives which ideally follow their own policies until termination. This in turn creates a reliance on the theory of Semi-Markov Decision Processes to provide a formal basis, which will be emphasized later on in details. The above notion of extending activities until their termination, naturally leads to Hierarchical Architectures of Reinforcement Learning Algorithms.

Shifting our focus from Reinforcement Learning algorithms to Humans, we are able to parse the world as a collection of objects, that are discrete, persistent, which can be interacted with. At any given point in time, the human eye can selectively focus on one object-of-interest while peripherally discarding the irrelevant information in the decision-making process. Humans can use this (unique) compact representation of the environment for planning, reasoning, and exploration. In certain tasks, for e.g., games which require the player to complete a set of complex sub-tasks or sub-objectives, a human can identify the different objects of interest (or sub-tasks) efficiently. One such

example can be seen from an Atari game called `Montezuma's Revenge`, which requires the agent to obtain a key in the very first room and then unlocking a door, after traversing the game environment. Possible objects of interest in such an environment could be the avatar that moves in a 2-D plane, a rolling skull, a key, etc. Or in the case of `Super Mario Bros`, where the primary objective is not to collect as many coins as possible, but to reach the green goal flag (atleast in level 1). These types of games may or may not be solvable in a linear fashion, but fulfilling certain optional (untold) objectives may yield more reward in the long-term, irrespective of short-term negative rewards, if any.

Without any prior information related to such an environment setting, a novice Human Player would have to explore all the possibilities before understanding the Dynamics of the environment in order to complete the given task. This can be interpreted formally in the following manner: the novice human player constructs a differentiable mapping from an image to a discrete tabular list of objects, where each object consists of a differentiable position, feature vector, and scalar presence value that allows the representation to be learnt using an attention mechanism.

Existing strategies to solve RL tasks of this nature, usually take the whole image as the state-space and let the Non-Linear Function Approximator learn the weights to better estimate state-action values. Using such strategy doesn't aid in mitigating the curse of dimensionality, rather it adds another layer of complexity. If any two consecutive frames of a game's state are analyzed, one can find that in some cases a large amount of pixels may vary, while on some other states a very negligible amount of pixels are changing. A consistent compact representation of the actual state space, should provide better overall results in terms of reduction of dimensionality of the input. It could be more robust to variations and supposedly easier to transfer between different environments — provided that the compressed representation captures the underlying structure of the environment. A similar association can be persuaded utilizing only visual frames as input to an object detection model or a neural network which would ideally decompose it into some sort of embedding for each of the objects present in that particular given frame. Our approach thus commences, taking this ideology of finding relations between multiple objects in a given frame, coming up with an unique compressed representation and training an RL Agent to solve the re-defined problem statement. Such an approach of using Object Embeddings instead of passing the frame as an input to the

agent has a potential to unravel newer avenues towards General Video Game AI.

1.1 State-Space Shaping

State-space shaping is a technique used in reinforcement learning to modify the state space of an environment with the objective to improve learning efficiency of an RL agent. In the context of an MDP, the state-space shaping involves adding additional informative features or removal of unwanted features from the observations or state-space to provide more informative and structured input to the agent. MDPs provide a formal framework for modeling sequential decision-making under uncertainty, and shaping state-space can be particularly useful with large or continuous state spaces. By reducing the dimensionality of the state space or providing more informative features could aid towards learning an optimal policy for the given task. One key consideration while shaping the state space for MDPs, is whether the resulting state space also remains Markovian.

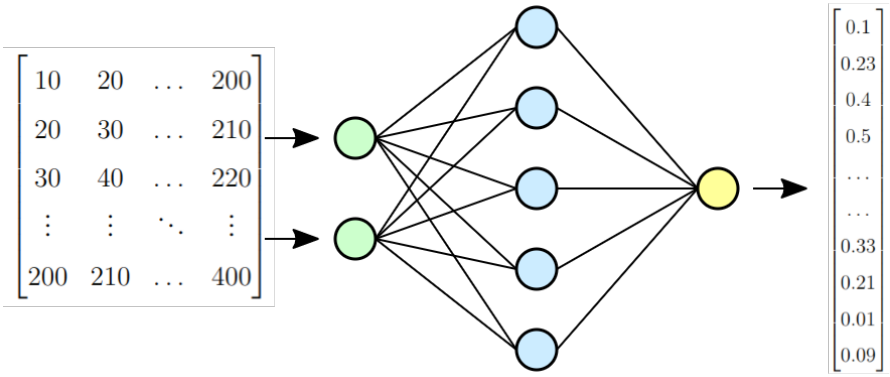


Figure 1.1: An illustration of conjectural State Space Shaping.

While shaping of state-space seems very trivial, it comes with some quirks of its own —

- **Complexity:** Creating a well-designed state-space can be a complex task, particularly in large or continuous state spaces. Choosing the right features or variables to include in the state space can be difficult, and even small changes to the state space can have a significant impact on the performance of the agent.
- **Overfitting:** There is a risk of overfitting when shaping the state space. If the state space is very unique to one environment or activity, the agent may be unable to generalise its knowledge to other settings or tasks.

- **Curse of dimensionality:** If one intends to add more features to the existing state space, in that case state-space shaping can amplify the curse of dimensionality problem. The number of possible states can increase exponentially, which can make it more difficult for the agent to learn an optimal policy corresponding to given states.
- **Evaluation:** Evaluating the effectiveness of a state-space shaping technique can be challenging task. It may be difficult to determine whether an improvement in performance is due to the state-space shaping or because of other factors, such as a change in the learning algorithm or hyperparameters.
- **Trade-offs:** There may be trade-offs between the benefits of state-space shaping and the cost of implementing it. For example, adding more features to the state space can improve the expectation of converging to a globally optimal policy, but it may also increase the computational cost of learning.

Overall, state-space shaping can be a powerful technique for improving the learning efficiency of an agent. However, it requires careful consideration of the challenges and trade-offs involved to ensure that the resulting state space is informative, generalizable, and computationally feasible.

CHAPTER 2

Background

2.1 Markov and Semi-Markov Decision Processes

The majority of research in RL revolves around the use of Markov decision processes (MDPs) as a formalism. Although RL is not limited to MDPs, researchers often choose this discrete-time, finite state and action framework as it provides a straightforward framework to study fundamental algorithms and their characteristics. At each timestep, the agent perceives the system's state s contained in a finite set $\mathcal{S}$ and executes an action a from a finite, non-empty set of Actions $\mathcal{A}$ of admissible actions. The Agent receives an immediate reward given by the environmental Reward Function $R(s, a)$, and transitions to the next state s' with a probability $P(s'|s, a)$. This entailment of Rewards and Transitional probabilities is often referred to as the *one-step model* of action a.

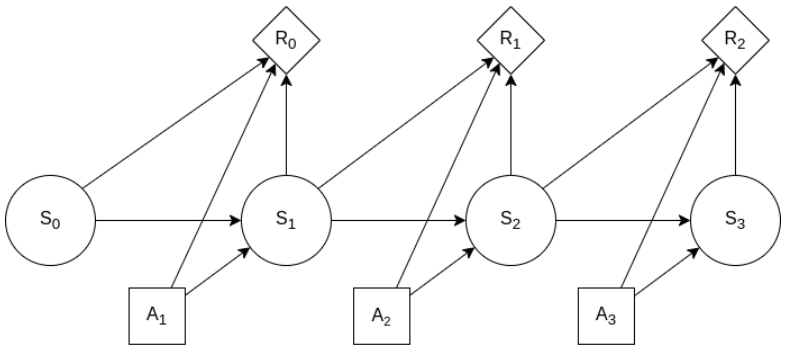


Figure 2.1: An example of the subset of a decision network depicting a finite part of an MDP

A policy $\pi(s, a)$ is defined as the probability with which the Agent executes action a whenever it observes the state s. For any policy π and $s \in S$, $V^\pi(s)$ denotes the expected infinite-horizon discounted return from s given that the Agent uses policy π:

$$V^\pi(s) = \mathbb{E}[r_{t+1}, \gamma r_{t+2} + \gamma^2 r_{t+3} + \ldots | s_t = s, \pi]$$

where γ, $0 \leq \gamma < 1$, is a discount factor. $V^\pi(s)$ is the value of the value function signifying the value of state s under policy π, where V^π is the value function corresponding to policy π.

The goal for such discounted MDP setting is to find an optimal policy, i.e. π^*, that maximizes the value of each state. The unique optimal value function corresponding to this optimal policy is defined by V^*.

Another perspective of looking at RL Algorithms is through their Action-Value Functions — which assigns values to admissible state-action pairs. Given a policy π, the value of (s, a), $a \in \mathcal{A}$, $Q^\pi(s, a)$ is the expected infinite-horizon discounted return for executing action a in state s and thereafter following policy π:

$$Q^\pi(s, a) = \mathbb{E}[r_{t+1}, \gamma r_{t+2} + \gamma^2 r_{t+3} + \ldots | s_t = s, a_t = a, \pi]$$

Similar to Value Function, and optimal action-value function, Q^* exists and is assigned to each admissible (s, a) pair.

Dynamic Programming methods exploit the fact that value functions satisfy Bellman Equations in the following manner:

$$V^\pi(s) = \sum_{a \in \mathcal{A}} \pi(s, a) \left[R(s, a) + \gamma \sum_{s'} P(s'|s, a) V^\pi(s') \right]$$

and

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

Analogous equations for Q^π and Q^* exist:

$$Q^*(s, a) = R(s, a) + \gamma \sum_{s'} P(s'|s, a) \max_{a \in \mathcal{A}} Q^*(s', a')$$

In an MDP, only the sequential nature of decision process is relevant, not the amount of time that passes between consecutive timesteps. A generalization of this can be seen from the Semi-Markov Decision Process (SMDP) in which the amount of time between one decision and the next is a random variable. If the random variable is real-valued, then the SMDP models continuous systems, whereas with discrete integer-values, the SMDP models discrete events systems.

2.2 Reinforcement Learning

Various techniques have been proposed for approximating Markov decision process (MDP) solutions with less effort than required by conventional dynamic programming (DP). However, reinforcement learning (RL) methods stand out due to their use of stochastic approximation, and function approximation techniques. In Reinforcement Learning, the agent can solve sequential decision-making problems by working in a trial-and-error fashion. The optimal policy thus learnt, through repetitive interactions with the environment, is the one that maximizes the expected cumulative reward over time, by learning an optimal mapping between the states and valid set of permissible actions. RL algorithms combine some or all of the following features:

- Computation is limited to states on or near multiple sample trajectories, real or simulated, avoiding the exhaustive sweeps of DP. This approach can exploit situations where many states have low probabilities of occurrence in actual experience since computation is guided along sample trajectories.
- The basic DP backup is simplified by sampling. Instead of generating and evaluating all possible immediate successors of a state, the effect of a backup is estimated by sampling from the appropriate distribution.
- Function approximation methods, such as linear combinations of basis functions, neural networks, or other methods, are used to represent value functions and/or policies more compactly than lookup-table representations.

The agent's policy doesn't need high precision in states which are visited rarely.

2.2.1 Elements of Reinforcement Learning

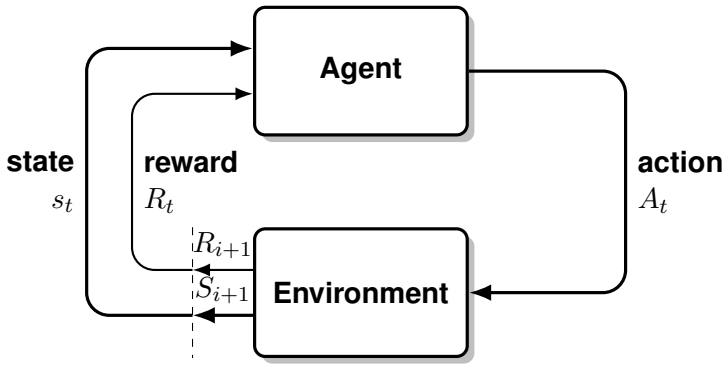


Figure 2.2: General framework for Agent-Environment interaction in a typical Reinforcement Learning problem.

Reinforcement Learning has the following crucial elements for its formulation (Sutton and Barto (2018)):

- **Agent:** The agent is the entity that interacts with the environment and takes actions based on its current state. The agent's goal is to learn an optimal policy that maps states to actions in a way that maximizes the expected cumulative reward over time.
- **Environment:** The environment is the entity with which the agent interacts. It provides the agent with feedback in the form of a reward signal for each action it takes. The environment can be modeled as a Markov Decision Process (MDP), where the agent's actions influence the next state of the environment.
- **State:** The state is a representation of the environment at a particular time step. It includes all relevant information about the environment that the agent can perceive.
- **Action:** The action is the decision made by the agent at a particular time step based on its current state. The action can influence the next state of the environment and the reward received by the agent.
- **Reward:** The reward is the feedback provided by the environment to the agent for each action it takes. It is typically a scalar value that represents the desirability of the outcome of the action taken according the current policy.
- **Policy:** The policy is a mapping from states to actions that determines the action to be taken by the agent in each state. In easier terms, it can be thought of being the strategy that the agent uses to choose actions. The goal of the agent is to learn an optimal policy that maximizes the expected cumulative reward over time.
- **Value function:** The value function is a function that estimates the expected cumulative reward of following a particular policy. The value function is used to evaluate the quality of a policy and to guide the agent's decision-making process.

Besides the above, there is also the notion of a **"Model"**. The model is the agent's abstract representation of the environment which aims to mimic the real world environment's stochastic behaviour. The model may be imperfect or incomplete, which may lead to errors in the learning process. The model of the environment is composed of primarily two parts:

- *State transition function ($\mathcal{P}$):* The state transition function is responsible for the environment dynamics and predicts the next state s', which the environment transitions to after the agent takes an action a from its current state s. In other words, the state transition function define how the environment evolves over time in response to the agent's actions. The transition dynamics can be deterministic or stochastic, depending on the specific problem being addressed.

$$\mathcal{P}(s, a, s') = P(s_{t+1} = s' | s_t = s, A_t = a)$$

- *Reward function ($\mathcal{R}$):* The reward function maps each state and action pair to a scalar value that represents the immediate feedback or reward that the agent receives from the environment, upon transition from state s to state s' upon taking

an action a. The reward function encodes the goal of the RL problem and provides the agent with information about which actions are desirable and which are not. The reward function can be designed in different ways, depending on the specific problem being addressed.

$$\mathcal{R}(s, a, s') = \mathbb{E}[R_t | S_t = s, A_t = a, S_{t+1} = s]$$

Overall, the model of an environment in RL is a mathematical representation of the system that the agent is trying to learn to operate in. It provides the agent with a set of rules to follow and a feedback signal to guide its learning process.

These elements work together in a feedback loop to enable the agent to learn an optimal policy through trial-and-error interactions with the environment. The agent takes actions, receives feedback in the form of a reward signal, updates its estimate of the value function, and adjusts its policy accordingly. The entire cycle of *state* $\rightarrow$ *action* $\rightarrow$ *reward* represents a single *time-step* of the decision-making process. The process continues for multiple such timesteps till it completes the desired objective. Through this process, the agent learns to make decisions that maximize the expected cumulative reward over time.

2.3 Overview of Intrinsic Motivations

2.3.1 From a Cognitive and Neuroscience perspective

Looking at the intrinsic motivation from a perspective of cognitive and neuroscience, the concept and source of intrinsic goals in Humans is quite complex. Existing research in cognitive sciences, however, provides some valuable insights. The foundational ideas discussed in this section about the cognitive systems used by humans to form their fundamental knowledge are based on research in developmental psychology through the works of Spelke and Kinzler (2007) and Lake *et al.* (2017). Developmental psychology suggests that humans across different cultures, including infants, children, and adults, use certain cognitive systems and intuitive theories on numerical quantities to form their fundamental knowledge — analogous to an agent's observations and its respective actions. Even the youngest of our species, i.e. newborns and infants, also seem to represent the visual world based on principles of cohesion, continuity and contact. Infants

can differentiate between sizes and distances and understand higher order numerical relations. Toddlers typically use this knowledge to generate intrinsic goals during its curiosity-driven activities. Questions starting with “What happens if ... ?”, “Maybe if I do this...”, etc. expand their knowledge space and help them learn hierarchical causal relationships between events and how one event can lead to another one or affect some other event. A clear notion about how infants represent the visual world and other agent’s behaviour has revolved around the works of Woodward (1998).

The successor representation is a concept that has been further investigated in Neuroscience. It involves representing a value function based on the expected future occupancy of states, which can be useful for solving spatial navigation problems by identifying reasonable sub-goals. In a review paper, Botvinick *et al.* (2009) provided an extensive overview of hierarchical reinforcement learning, discussing its relevance to cognitive and neuroscience research.

2.3.2 In the Context of Reinforcement Learning

In the context of reinforcement learning (RL), intrinsic goal-driven strategy refers to a learning approach in which the agent is driven by internal or intrinsic goals rather than external or extrinsic goals. This means that the agent learns to perform a task not just because of the rewards that it receives from the environment, but because it has an internal motivation to achieve certain goals or objectives.

The use of intrinsic goals can help to address some of the limitations of traditional RL approaches, which rely heavily on external rewards to guide the learning process. Intrinsic goals can encourage the agent to explore its environment and learn more efficiently, even in cases where external rewards may be sparse or delayed. Additionally, intrinsic goals can lead to more diverse and adaptive behavior, as the agent is not solely focused on maximizing the external rewards.

There are different approaches to incorporating intrinsic goals into RL, such as curiosity-driven exploration, self-supervised learning, and goal-conditioned reinforcement learning. These approaches vary in terms of the specific goals or objectives that the agent is encouraged to pursue, and the mechanisms by which these goals are incorporated into the learning process.

2.4 Object Detection Framework

For the shaping of state space, and decomposition of input frames to corresponding object we are using a popular state-of-the-art Object Detection model architecture called YOLOv7 (You Only Look Once). The model is based on the original YOLO algorithm, which is a single-stage detection method and is able to directly predict the bounding boxes and class probabilities for each object in a given input image. The broader view of the architecture of YOLOv7 model consists of a backbone network, a neck network and a head network. The backbone network is responsible for extracting features from the input image, whereas the neck network is a small convolutional layer that is used to combine the features extracted from the backbone network. The last and final network, i.e. head network, is used to predict the bounding boxes and class probabilities for each object in the image.

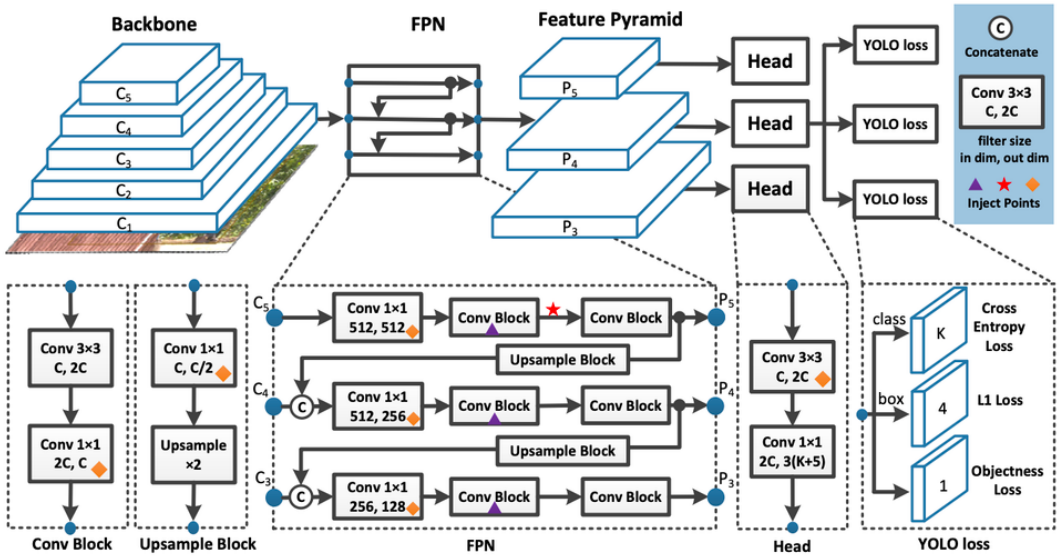


Figure 2.3: YOLO network architecture as depicted in Roboflow Blog.

The backbone network of the YOLOv7 model is based on the CSPDarknet53 Architecture, which is a mixed variant of WongKinYiu's Cross Stage Partial networks (CSPs) and Alexey Bochkovskiy's Darknet framework on real-time object detection. The optimized implementation of the backbone network, at its current iteration, consists of a series of convolutional layers, followed by a set of shortcut connections. The shortcut connections are used to skip over some of the convolutional layers, which helps to reduce computational cost of the model.

2.4.1 YOLOv7-tiny

A full-fledged YOLOv7 architecture is still too computationally heavy in terms of run-time inference performance in the context of using it in RL where typically training is done in the order of Millions of timesteps. To combat this issue, we are using a scaled down version of YOLOv7, called *"YOLOv7-tiny"*. The backbone in this case is a YOLOv7-tiny backbone, which consists of a series of convolutional layers with varying numbers of channels, kernel sizes, and stride sizes. The backbone has four levels of feature maps, each with a different spatial resolution. The head takes the features generated by the backbone and uses them to predict the bounding boxes, class probabilities, and objectness scores. The head is composed of a series of convolutional layers, followed by spatial pooling layers, and then concatenated together. The resulting features are passed through a few more convolutional layers before finally being used to generate the object detection predictions. Compared to the other versions, the edge-optimized YOLOv7-tiny uses Leaky-ReLU as the activation function, while other models use SiLU as the activation function. YOLOv7-tiny can effectively reduce about 40% of the total parameters and 50% computation of state-of-the-art real-time object detections, and achieve faster inference speed and higher detection accuracy. All YOLO architectures, including this one, need to conduct a post-processing of bounding boxes via non-maximum supression (NMS) to arrive at its final prediction.

2.5 An Overview of RL Algorithms Used

2.5.1 DQN

The Deep Q Network (or DQN) is an RL Algorithm proposed by Mnih *et al.* (2015) from DeepMind and was the first successful attempt to combine reinforcement learning with deep neural networks. The basic idea behind DQN is to use a neural network to approximate the Q-function, which is used to determine the optimal action to take in a given state. The Q-function maps a state-action pair to the expected total reward that can be obtained by taking that action in that state and then following the optimal policy.

The vanilla DQN Algorithm has several key components:

- **Replay Buffer:** It is used to store experience tuples (s, a, r, s') that the agent

encounters during training. The replay buffer allows the agent to learn from past experiences and helps to decorrelate the data which can improve learning stability.

- **The Target Network:** While in the training phase, the Q-network is updated using the Bellman equation, which involves taking the maximum expected reward over all possible actions in the next state. But the Q-Network is also used to take actions during training, which can lead to instability in the learning process. Thus, the target network is used to generate the target values for the Q-Learning updates. The target network is actually a copy of the main network, updated periodically — either by copying the current Q-Network directly (Hard update) or using Polyak update or soft copying the current Q-Network.
- **ε-Greedy Exploration:** During the training phase, the agent employs an ε-greedy policy to select its actions. This policy involves choosing the action with the highest Q-value with a probability of $(1 - \varepsilon)$, and selecting a random action with a probability of ε. This ensures that the agent explores the state space and learns about the optimal policy.

However, there are some limitations to the DQN algorithm. One limitation is that it can be slow to converge and may require a large amount of data to learn a good policy. It can be very sensitive to certain set of hyperparameters, such as the learning rate and the mini-batch size. Finally, the DQN algorithm can struggle with tasks that require long-term planning, as it only considers the immediate reward and does not take into account the future rewards that may be obtained by taking a particular action.

2.5.2 DDQN

Similar to DQN, Double DQN uses two identical neural network models. One of them learns during the experience replay, similar to DQN. While the other one is a copy of the last t episodes of the first model. The Q-values are actually computed from this lagging model, inorder to reduce the overestimation bias. In DQN, the Q-values are calculated with the reward added to the next state's maximum Q-value. Naturally, if every time the Q-value calculates a higher number for a certain state, the value for that specific state will become disproportionately higher. This can lead to overestimation of Q-values, especially for actions that have been selected more frequently in the past. Thus in Double DQN, the Q-values are calculated using the main network, but the action selection is based on the Q-values calculated using the second network. This reduces the overestimation of Q-values, as the secondary network is less likely to be biased towards frequently selected actions. Theoretically, and practically, the Double

DQN algorithm can be more accurate in determining the optimal action for a given state.

2.6 An Overview of Object-Oriented MDPs

The initial motivation for focusing on Objects to facilitate the RL Agent's decision taking process, has been provoked through the works of Diuk *et al.* (2008) in 2008. The authors' initial motivation behind their work emerged with the intention to solve the problem with traditional MDP framework that it is limited in its ability to represent complex real-world problems, as it assumes a fixed set of variables and actions that may not be relevant or informative for the given problem. They introduced Object-Oriented MPDs a representation based on objects and their interactions, which is a natural way of modeling environments and offers important generalization opportunities. Their implementation methodology focused on starting from attributes that can be directly perceived by the Agent, rather than attributes introduced by the designer. The state in this approach is represented as a collection of objects, each of which has a set of attributes and methods that define its behaviour and interactions with other objects. Similar to the work on Relational MDPs, Guestrin *et al.* (2003) they have also defined a set of classes $\mathcal{C} = \{C_1, C_2, \ldots, C_c\}$, with each class includes attributes, $Attr(\mathcal{C}) = \{C.a_1, C.a_2, \ldots, C.a_a\}$, and each attribute has a domain $Dom(\mathcal{C}.a)$. Any particular environment contains a set of objects, $O = \{o_1, o_2, \ldots, o_o\}$, where each object can be mapped to a certain class — in other words, each object is an instance of one class: $o \in \mathcal{C}_i$. The state of the underlying MDP is the union of the states of all its objects: $s = \cup_{i=1}^{o} o_i.state$, where $o.state$ is a value assignment to all the attributes for that specific object. When two objects interact in some way, they define a relation between them — a change in value of one or multiple attributes in either or both interacting objects, determines an effect. This behaviour can be interpreted in a class-level, i.e. multiple objects belonging to the same class behave in the same way when interacting with other objects (or even the Agent). Actions are also defined as methods that can be applied to the objects, resulting in changes to their attributes and the environmental state. Different domains require different relations, which depend on the transition dynamics of the environment. In their work, they have used a Boolean Relational Function, as per the complexity of their hypothetical testing environment, `Taxi Domain`.

By incorporating an object-oriented perspective, the framework is able to capture more reined and complex relationships between objects and actions, leading to more effective and efficient decision-making in a wide range of real-world applications.
Hence, one can get a vivid perception on how to represent multiple objects, from a given environment, that may or may not belong to the same class.

The concept of learning functions that operate on sets of variable-sized inputs has been explored in the novel work by Zaheer *et al.* (2017), namely, Deep Sets. Their approach provides an unique insight into solving problems that deal with variable-sized inputs, where the objective functions remain invariant to permutations. The authors presents a family of functions that characterize such permutation-invariant functions, which can be used to design deep network architectures that operate on sets. Unlike typical Machine Learning Algorithms, such as regression or classification, which are designed for fixed-dimensional data instances, their proposed approach handles inputs or outputs that are permutation invariant sets or variable dimensions. A standard problem of similarity search and metric learning of Unsupervised Learning could be of valuable importance in the work of discussion. The authors of Deep Sets paper have proposed fundamental architectures to handle "sets" as inputs and allowing conditioning on arbitrary objects. Specifically, the authors propose a deep neural network architecture that consists of two stages: an encoding stage that maps each element in the input set to a fixed-size embedding, and an aggregation stage that combines the embeddings into a single output. The idea is to design the encoding stage as a traditional neural network architecture, such as LNN or CNN, that maps each input element to a fixed-size embedding, while the aggregation stage is implemented using a permutation-invariant function, such as the mean or max pooling functions, that combine embeddings into a single output.

According to their work, each instance (or object) x_m is transformed into some representation $\phi(x_m)$. These representations $\phi(x_m)$ are then added up and the output is processed using a ρ network, similar to any deep network. In case any additional meta-information z is available, then the above mentioned networks could be conditioned to obtain a conditional mapping $\phi(x_m|z)$ — In other words, adding all representations and applying some non-linear transformation.

Equivariant Model: The Equivariant model proposes a neural network layer, denoted by $f_\Theta(x)$, to be permutation equivariant during training. This is true when all off-

diagonal elements of θ are tied together and all diagonal elements are equal, such that $\Theta = \lambda\mathbf{I} + \gamma(\mathbf{1}\mathbf{1}^T)$ for $\lambda, \gamma \in \mathcal{R}$. The layer applies a non-linear function to a weighted combination of its input ($\mathbf{I}\mathbf{x}$) and the sum of input values $(\mathbf{1}\mathbf{1}^T)\mathbf{x}$. Since, `summation` does not depend on permutation, thus the layer is permutation-equivariant. The authors also implemented a variation of this approach that utilizes maxpooling over the elements of the set, which is commutative in nature.

$$f(\mathbf{x}) = \sigma(\lambda\mathbf{I}\mathbf{x} + \gamma \text{ maxpool}(x)\mathbf{1})$$

According to their demonstrations, this variation performs better in certain applications — possibly due to the fact that for $\gamma = \lambda$, the input to the non-linearity is max-normalized.

Samothrakis *et al.* (2015) through their novel work proposed an approach for developing agents that can play a wide range of video games. Their work employs a fixed set of general features, i.e., distance to the nearest enemy, number of tokens collected and so on, to encapsulate the current game-state. The approach proposed in the paper combines two techniques: neuroevolution and novelty search. They first compare different policies (ϵ-greedy and Softmax) and function approximators using their evolutionary algorithm **S-NES** as a learning scheme. Neuroevolution involves using evolutionary algorithms to evolve the structure and weights of neural networks, while novelty search involves searching for novel behaviors that may not necessarily be rewarded by a specific task. As mentioned in their paper, initially they planned that the Agent will learn how to maximize the score in 10 different games, without any domain specific heuristic — But failed to accomplish (partially) for environments requiring long-term planning. The lack of a learning signal in their approach proved fatal on the long run for their agent on environments with long-term planning or requiring substantial amount of exploration to begin with. They note that their approach may be particularly effective for games that have no clear objective or reward structure, as novelty search is able to find behaviors that are not explicitly rewarded. While this approach often works well, those general features have to be handcrafted, which is laborious and requires human expertise. This pattern of approach is relatively game-specific and could not be applied for the context of General Video Gameplay AI.

Malinowski *et al.* (2018) proposed a novel approach for Visual Question Answering (VQA) which incorporates a hard attention mechanism. This approach achieves highly

competitive performance on a VQA dataset and even surpasses soft attention architectures in some cases. The hard attention mechanism selects a subset of information for further processing, focusing computation on important parts of the input. Malinowski et al. were inspired by attention mechanisms in biological perception, which select subsets of perceptual information for more sophisticated processing, which would be prohibitively expensive to perform on all sensory inputs. However, hard attention-based networks have a key downside within a gradient-based learning framework: the choice of which information to process is discrete and non-differentiable. As a result, gradients cannot be backpropagated into the selection mechanism to support gradient-based optimization.

To address this challenge, they incorporated L2-norms of feature vectors into their hard attention mechanism to select subsets of information for further processing. In the first version of their approach, a fixed number of feature vectors is selected in descending order of norms, while in the second version, the number of feature vectors selected depends on the input. This approach allows for the generation of hard attention masks that can contain semantically meaningful information, such as coherent objects.

Malinowski et al. encoded images using a CNN (ResNet-101) and questions using an LSTM to obtain fixed-length vector representations. After a few layers of combined processing, attention over spatial locations was applied, similar to what soft attention mechanisms do, followed by Sum-Pooling or Max-Pooling. The authors assumed that important outputs of the CNN would tend to grow in norm and thus are likely to be selected.

The main contribution of this work is the incorporation of hard attention mechanisms into VQA, which improves the accuracy and efficiency of learning by focusing computation on important parts of the input. The use of L2-norms of feature vectors is an innovative solution to the non-differentiability challenge posed by hard attention mechanisms. Furthermore, the generation of hard attention masks that contain semantically meaningful information is an exciting development that could be useful in other applications beyond VQA.

One limitation of this approach is the reliance on fixed-length vector representations for both images and questions, which may not capture all the necessary information. Additionally, while the hard attention mechanism improves efficiency and accuracy, it

may not be as interpretable as soft attention mechanisms. Another limitation is the use of only one type of attention mechanism, whereas a combination of hard and soft attention mechanisms may yield even better results.

In conclusion, the incorporation of hard attention mechanisms into VQA by Malinowski et al. is a novel and exciting development that improves the accuracy and efficiency of learning by focusing computation on important parts of the input. The use of L2-norms of feature vectors is an innovative solution to the non-differentiability challenge posed by hard attention mechanisms. However, the approach also has limitations, such as the reliance on fixed-length vector representations and the lack of interpretability of hard attention mechanisms. Further research could explore the combination of hard and soft attention mechanisms to improve results even further.

In the context of hierarchical RL, Goel and Huber (2003) discuss a framework for subgoal discovery using the structural aspects of a learned policy model. The authors argue that traditional approaches to subgoal discovery often require a large amount of prior knowledge or manual intervention, and that learning subgoals from scratch could be more effective. Their approach involves using a set of learned policies to discover subgoals that can be used in an HRL framework. The policies that are learned using unsupervised learning techniques are used to cluster similar states together. The resulting clusters are then used as subgoals in an HRL framework. The authors reduce the requirement for manual intervention or prior knowledge by employing unsupervised learning approaches to uncover subgoals, resulting in increased performance on benchmark tasks. Overall, Goel and Huber's research demonstrates a viable approach for subgoal identification in HRL utilising learnt policies.

Şimşek *et al.* (2005) provide a graph partitioning approach aimed towards subgoal identification. Their approach involves constructing a graph representation of the problem space, where nodes represent states and edges represent transitions between states. Afterwards, the graph is partitioned into clusters using a local graph partitioning algorithm, which identifies sets of states that are tightly connected to each other but weakly connected to the other parts of the graph. These clusters are then used as subgoals in the RL Framework. The authors also examine the features of the subgoals identified by their method. They discover that the subgoals are frequently positioned near obstacles or other factors that are necessary for successful task completion in areas with high

state density. This strategy of incorporating graph partitioning for subgoal identification could be beneficial in the our problem statement.

The context of Graphs brings us to the work by Mylavarapu *et al.* (2020*b*,*a*), where the authors have presented a novel Multi-Relational Graph Convolutional Network (MRGCN) based framework. The input to the MRGCN is a multi-relational graph where the graph's nodes denote the active and passive agents/objects in the scene, and the bidirectional edges that connect every pair of nodes are encodings of their Spatio-temporal relations. Now, they use this strategy for a different problem statement, i.e. Vehicle Behaviour Classification Task. The proposed Interaction graph, models different agents and objects in the scene as nodes of a graph. This graph captures the spatio-temporal evolution of relations between all-pair of objects in the scene with appropriate bi-directional asymmetric edges. For the extraction of only active objects of interest, behavioural relations between the nodes have been used to denote the overall behaviour of the Agent. This in turn provides us with an encoding from a sequence of single-camera observations using straight forward projective geometry techniques. The viability of an additional Neural Attention Component has also been addressed by the authors, which can boost the MRGCN's performance further to perfect predictions. As indicated, this is one of the takeaways from their work which could potentially be beneficial in the context of Object/Goal/Subgoal Prioritization.

2.7 Related Works

Kulkarni *et al.* (2016), came up with Hierarchical Deep Reinforcement Learning , which primarily follows the principle of hierarchy consisting of a top-level value function which learns a policy over intrinsic goals, and a lower-level function learns a policy over atomic actions with intrinsic rewards to satisfy the given goal. It allows flexible goal specifications, efficient space for exploration in complicated environments. Similar to their demonstrations, our environment setting also provides delayed or sparse rewards. A probable solution as mentioned by the authors (of this work), was to first learn ways to achieve intrinsically generated goals and subsequently learn an optimal policy to chain them together. Each value function $V(s, g)$ can be used to generate a policy that terminates when the agent reaches the (sub)goal state g. The collection of these policies

can be hierarchically arranged with temporal dynamics for the case of semi-Markov Decision Processes. For higher dimensional problems, the authors have mentioned that these Value Function can be approximated by Neural Networks in the form $V(s, g; \theta)$ as per their implementation goes, their model consists of two modules — (i) The *meta-controller*, which takes the state as input and picks a (sub)goal, (ii) The lower-level *controller*, which takes both state and chosen goal by the *meta-controller* and selects action until that goal is optimally reached or the episode is terminated. This cycle goes on between *meta-controller* and *controller* till a desired optimal policy is learned by both. For the *controller*, the following Q-value Function has been utilized by the authors:

$$\begin{aligned} Q_1^*(s, a; g) &= \max_{ag} \mathbb{E}[\sum_{t'=t}^{\infty} \gamma^{t'-t} r'_t | s_t = s, a_t = a, g_t = g, \pi_{ag}] \\ &= \max_{ag} \mathbb{E}[r_t + \gamma \max_{a_{t+1}} Q_1^*(s_{t+1}, a_{t+1}; g) | s_t = s, a_t = a, g_t = g, \pi_{ag}] \end{aligned}$$

where, g is the agent's (sub)goal in state s and $\pi_{ag} = P(a|s, g)$ is the action policy. Similarly, for the *meta-controller*, we have:

$$Q_2^*(s, g) = \max_{ag} \mathbb{E}[\sum_{t'=t}^{t+N} f'_t + \gamma \max_{g'} Q_2^*(s_{t+N}, g') | s_t = s, g_t = g, \pi_g]$$

where, N denotes the number of time steps until the controller halts given the current goal, g' is the agent's goal in state s_{t+N} , and $\pi_g = P(g|s)$ is the policy over goals. The transitions (s_t, g_t, f_t, s_{t+N}) generated by Q_2 run at a slower time-scale, than the transitions $(s_t, a_t, g_t, r_t, s_{t+1})$ generated by Q_1. Regarding the Model architecture, for Montezuma's Revenge, they have used stacked convolutional layers with rectified linear units (ReLU). The input to their *meta-controller* was a set of four consecutive images of 84×84. A binary mask is appended over the images and this augmented input is passed to the lower-level controller. Two experience replay buffers D_1 and D_2 were set up. In the two-phased training procedure — the first phase is basically a fully exploratory step with ϵ of the *meta-controller* set to 1 and only the *controller* is trained on actions, whereas, in the second phase jointly both the *meta-controller* and *controller* are trained. Upon keen observation of how their agent learns, it has been observed that the agent first learns to properly reach smaller goals like `Ladder`, avoiding `skull`, etc. but only at a later stage, it learns to properly reach 'harder' goals.

The authors don't speak openly regarding whether any strategies related to automatically dientanglement of objects from the given raw pixels were applied or not. Their work surely motivates other works of similar stature which could focus on a general purpose approach apart from being domain/env specific.

The ability to perceive and represent visual sensory data into useful and concise descriptions is considered a fundamental cognitive capability in humans, and thus crucial for building intelligent agents. Thus the ability to capture latent generative factors of an environment, is crucial for building intelligent agents that can perform a wide variety of tasks. Learning such representations without supervision from rewards is a challenging open problem. The method introduced by Anand et al. Anand *et al.* (2020) tries to tackle this problem by learning state representations by maximizing mutual information across spatially and temporally distinct features of a neural encoder of the observations. Their representation primarily focused on high-level semantics and ignored the low-level details (such as precise texture of the background). Prior work in the field of neurosciences suggested that the brain maximizes *predictive information* Bialek and Tishby (1999) at an abstract level to avoid sensory overload. Thus their proposed representation learning approach relies on maximizing an estimate based on a lower bound on the mutual information over consecutive observations x_t and x_{t+1}. Evaluation of such representations are done based on how well they capture the ground truth state variables.

The proposed Spatio-Temporal DeepInfoMax (ST-DIM) architecture consists of two different Mutual Information Objectives:

- **Local-Local Infomax**

  $$\mathcal{L}_{\mathcal{LL}} = \sum_{m=1}^{M}\sum_{n=1}^{N} -\log \frac{\exp(f_{m,n}(x_t, x_{t+1}))}{\sum_{x_t^* \in X_{next}} \exp(f_{m,n}(x_t, x_{t*}))}$$

  here, the score function of the Local-Local Objective is $f_{m,n}$ and can be defined as

  $$f_{m,n}(x_t, x_{t+1}) = \phi_{m,n}(x_t)^T W_l \phi_{m,n}(x_{t+1})$$

  .

- **Global-Local Infomax** The global-local objective in the given below equation maximizes the mutual information between the full observation at time t with small patches of the observation at time $t+1$.

  $$\mathcal{L}_{GL} = \sum_{m=1}^{M}\sum_{n=1}^{N} -\log \frac{\exp(g_{m,n}(x_t, x_{t+1}))}{\sum_{x_t^* \in X_{next}} \exp(g_{m,n}(x_t, x_{t*}))}$$

The score function for Global-Local Objective $g_{m,n}$ can be defined as

$$g_{m,n}(x_t, x_{t+1}) = \phi(x_t)^T W_g \phi_{m,n}(x_{t+1})$$

and $\phi_{m,n}$ is the local feature vector produced by the intermediate layer in ϕ at the (m, n) spatial location.

where, $\mathbf{X} = \{(x_t, x_{t+1})_i\}_{i=1}^{B}$ be a minibatch of consecutive observations that are randomly sampled from several collected episodes. $\mathbf{X}_{next} = X[:, 1]$ correspond to the set of next observations. In this context, the positive samples correspond to pairs of consecutive observations (x_t, x_{t+1}) and negative samples correspond to pairs of non-consecutive observations (x_t, x_{t*}), where x_{t*} is a randomly sampled observation from the same minibatch. Their model was able to successfully categorize state variables of all the tested games among six major categories: agent localization, small object localization, other localization, score/clock/lives/display and miscellaneous.

So far their work has been a gold mine in creating a model of the representation as part of my proposed work.

CHAPTER 3

Methodology

To address the limitations of existing Reinforcement Learning strategies with large state-spaces, we propose a streamlined framework that uses an Object Detection model, which acts as a wrapper for generating embeddings for the state space. Generally, the state space for not-so-trivial Atari 2600 games is actually the frame for that particular game-state, which is interpreted in the form of continuous states. We have used a state-of-the-art object detection network, called YOLOv7, and supervisedly trained it beforehand to detect the bounding boxes for our objects of interest in a particular frame of the game-state.

3.1 Environment Description

3.1.1 Montezuma's Revenge

The first game environment in which we tried on is one of the hardest Atari game, namely Montezuma's Revenge. In this game environment, your Agent plays as a treasure hunter named Panama Joe, whose goal is to find an ancient treasure hidden by Aztec warrior deep inside the catacombs. The catacombs are inhabited by monsters

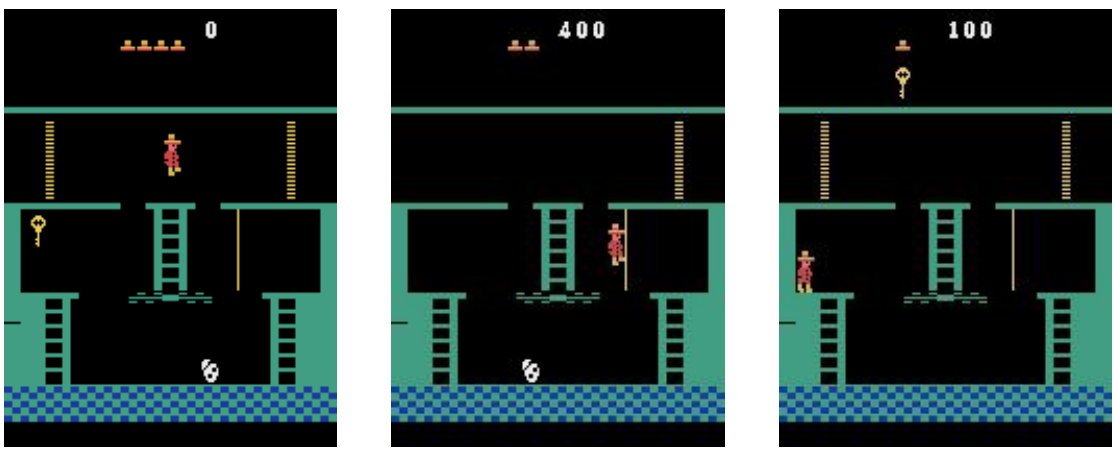


Figure 3.1: Montezuma's Revenge

and is a network of rooms, forming a large maze. Your character/agent can perform basic movement actions like jump, climb ladders, but lacks skills in 1v1 combat. Thus the monsters in the game should be avoided at all cost — a collision with an enemy would result in the loss of a game life. The Atari 2600 version has 24 rooms, but for our research we are limiting it to the first room itself.

It is harder to solve for RL Agents in general because of the fact that there is no notion of reward at every timestep. The first level (first room) of this environment has an objective of opening any of the doors, which would ideally allow the Agent to transition to other rooms. But to unlock the doors, you would need a "Key". Reaching the Key is not as simple as it seems to be. There are some obstacles in its path and complex platforms to traverse to finally reach the Key. There is a Skull type enemy or the Key Monster, interacting with which would result in immediate death of the player and a negative reward as the death penalty. Specialty of Montezuma's Revenge is that the rewards are only received from the environment after completion of the episode or end of Horizon or death of the Agent. Adding more complexity to the problem is the environment's stochasticity due to the presence of an enemy (skull) whose initial start state is pseudo-randomly initiated by the environment. Thus the initial state is also not fixed if one chooses to use the raw image frames as state space representations for their RL Agents.

3.1.2 Super Mario Bros.

Super Mario Bros. is a video game made by Nintendo in 1985. At this point of time, typically everyone is already familiar with what Super Mario Bros. is and how it is played. Super Mario Bros is a fairly straightforward linear game with the primary objective being reaching the rightmost flag at the end of the level as fast as possible.

Mario must jump, run, walk across the level while avoiding enemies and platforms and holes on its way there. Jumping on the top of enemies eliminates them, while jumping onto a block of interest results in gaining extra reward, in the form of coins, or new mushroom pops up which can potentially give Mario "power-ups". A Red Mushroom will make Mario bigger, which in turn would make Mario smaller if he hits an enemy but not kill him. When Mario is in his larger form and strikes a question block with a

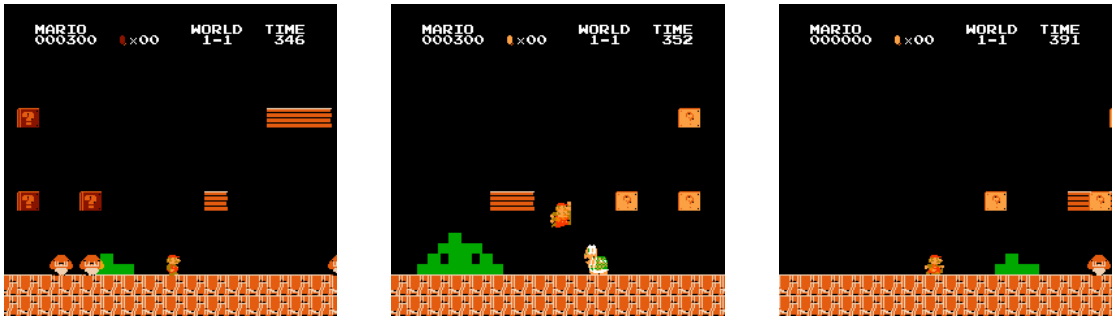


Figure 3.2: Super Mario Bros.

power-up, a fire blossom is released, granting him the ability to fling fireballs and eliminate his enemies. If he is hit by an enemy while in his Fire Mario form, he will revert to his normal size. Each world is divided into four levels. Some levels are underground and others are underwater. When Mario is underwater, he cannot jump on any enemy and he will shrink or die if he touches one. But for our experiments we are limiting Mario to only Level 1.

3.2 Agent Architecture

3.2.1 Pipeline Overview

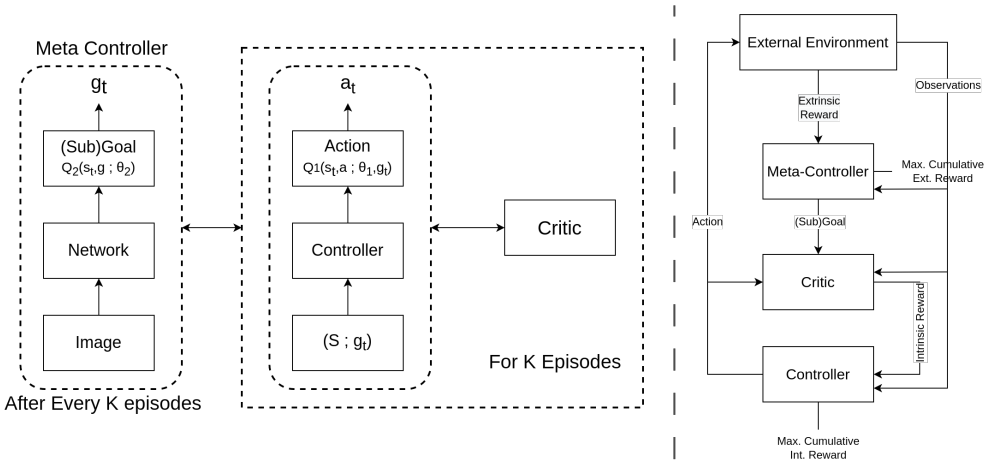


Figure 3.3: Overview of the Proposed Architecture.

Our proposed architecture contains a three layer architecture — (a) Meta-Controller, (b) Critic, and (c) Controller. It follows the following process pipeline:

- The Agent steps through the environment with some initial random action based on the randomly initiated policy.

- The observation or next state (s_{t+1}) gets passed through the Object Detection Model (or the Meta-Controller) and gets transformed to a compressed and consistent uniform representation consisting of the coordinates of objects present in the original frame.
- The Meta-controller chooses one of the object as a sub-goal (g_t), based on its own policy (π_o) and fixes it for "K" number of episodes.
- This transformed next state (s'_{t+1}) and subgoal (g_t) gets passed to our RL agent, i.e. the Critic-Controller part of the architecture, as its next observed game state.
- The rewards given by the actual environment are normalized and then shaped as per each environment's specific requirement.
- The critic gives intrinsic rewards to the agent to converge to an optimal policy to reach the subgoal chosen by the Meta-Controller.
- The Agent takes decision based on this transformed state and shaped rewards which it receives after each transition to the next state, and accordingly learns to solve the environment eventually.

3.2.2 Observation Preprocessing

Our initial attempts of just using one frame didn't yield satisfiable results, as just one frame was not providing seemingly enough information for the RL agent to get a notion of transitions. Moreover while processing some frames it has been noticed that the in some games a significant amount of "flickering" is noticed — where some objects appear only in even frames while the other objects appear only in odd frames. These anomalies are the result of limited computational capability of Atari 2600 system's display capabilities. As a solution to the aforementioned issue, was to stack multiple frames one on top of the other and then taking the max of each pixel value among the stacked frames. Since from one frame to the other there is a very minimal change, using this strategy doesn't result in corruption or misinterpretation of actual state space. The number of frames to stack is considered a hyperparameter and needs to be tweaked. In our experiments, we have stacked 4 frames back-to-back and taken the max of them as described by Mnih *et al.* (2015) in their implementation.

3.2.3 Primary Architecture

Our work proposes an architecture which consists of three parts (broadly), except the environment:

- Meta-Controller
- Critic
- Controller

As per the processing pipeline, each frame of the game state after going through some common transformations (discussed in the following subsection), gets decomposed by the `Object Detection Wrapper` into a compressed representation of the state space containing sufficient information for the RL algorithm to learn. The Controller or Actor part, on its own doesn't get any beneficial feedback from the environment since the environments we are dealing with have very sparse rewards. The Agent will get environmental rewards, only after completion of the entire episode. The Meta-Controller, according to our design, chooses a suitable subgoal based on the detected objects on the frame and sends an intrinsic reward signal at each timestep to the Agent or controller. The intrinsic reward metric used here is basically the L2-Norm between the agent and the chosen subgoal. The goal of the Controller is then to reach that subgoal or detected object. Once this is fulfilled, the control is handed back to the Meta-controller. The Meta-Controller then chooses another subgoal or object to reach from that state where the previous subgoal terminated. As an implementation quirk, we have limited the horizon for reaching a specific subgoal to be at most 1000 timesteps, after which the control goes back to the meta-controller which selects the next subgoal regardless of the termination of existing subgoal. The loop repeats till the both the Meta-controller and the Controller (Agent) converges to an optimal policy.

In other words,

- Let $\mathcal{O}$: be the given set of objects, decomposed from the given frame.
- Given a state s_t, the Meta-Controller chooses a subgoal $o_i \; \forall i \in \mathcal{O}$ according to its own policy π_o.
- The controller assumes this chosen subgoal (o) to be its goal and gets intrinsic rewards in order to reach this subgoal (o).
- When the agent reaches the goal, the Meta-Controller chooses another subgoal from $\mathcal{O}$, and the loop repeats.
- The Meta-Controller has no prior notion of whether the chosen subgoal is the actual goal or not. It will choose subgoals depending upon its own policy (π_o).

3.2.4 Implementational Maneuvers

Taking inspiration from a plethora of works which have been done earlier on Atari 2600 environments or NES environments, we have also implemented similar strategies in our work in the implementation part.

• One of the notable strategies is the "Frameskipping" strategy:

- Generally when an action is performed, the animation corresponding to that particular action is played by the environment. Moreover, each action results in very insignificant changes in most games — ideally you would need to repeat the same action for multiple timesteps to see actual visible changes. For e.g. there is a platform, or stairs or ladder to climb, performing one up/down/right/left action won't make the transition of the agent from the platform/stairs/ladder to some other platform. At least one needs to repeat the same action multiple times to traverse that specific region of the environment.
- So to reduce the computational expense of taking a decision every timestep, we are making the agent take decisions every k^{th} timestep.
- Running only the emulator for consecutive steps requires negligible resources, compared to having the agent select unnecessary actions at each timestep.
- More precisely, when the agent sees a state and selects an action to perform, that action is performed for the next k frames and then the next frame is passed back through the processing pipeline back to the Agent as its next state (s_{t+1})
- This strategy allows effectively play k times more timesteps without significantly loosing information nor increasing runtime duration.

• Another notable changes made by us is reducing the action space.

- The original Atari 2600 controller had 18 permissible actions, and NES_py simulator had 12 actions for all supported games.
- But in practice all the actions are not necessarily important for all environment.
- So for our environments, we reduced our permissible set of actions to —
  - For Montezuma's Revenge: the action space has been reduced to 7 `[FIRE, UP, RIGHT, LEFT, DOWN, UPRIGHTFIRE, UPLEFTFIRE]`.
  - For Super Mario Bros: the action space has been reduced to 5 `[['NOOP'], ['RIGHT'], ['RIGHT', 'A'], ['RIGHT', 'B'],['RIGHT', 'A', 'B']]`.

• In the Montezuma's Revenge environment the start state is not unique across different runs due to random initial position of the enemy.

- For that reason we have incorporated the NOOP Reset Environment strategy.

- The idea behind this strategy is to sample initial states by taking random number of no-ops on reset.
- A No-op action is assumed to be the action 0, which is a standard for Atari and NES games.

• In Super Mario Bros, the position of the enemies are randomly initiated by the game environment. Intentionally this has not been fixed, to induce some sort of generalizability and making the RL Agent resilient towards runtime stochastic behaviours of the environment.

• In the experimentation phase, it has been noticed that there has been some ambiguity noticed in the order in which these optimization strategies are applied. The order in which these strategies applied matters significantly while implementing them. Here is a probable order in which we applied the strategies which we mentioned earlier:

- Shape Action Space.
- Clip Environmental Reward.
- Limit Horizon.
- Noop reset.
- Max of n-frames.
- Frameskipping.
- Our proposed wrapper.

CHAPTER 4

Results and Discussion

This section includes a study of the observations and efficacy of our proposed methodology and analyze the different design choices proposed.

4.1 Analysis in Montezuma's Revenge Environment

Our first environment on which we intended to test our strategy was the famous Montezuma's Revenge. As per our model architecture, the agent was receiving intrinsic rewards at each timestep based on the Meta-Critic's policy. Through this strategy, the agent was able to learn to reach two of the four designated objects of interests. But miserably failed to reach any of the other objects with utmost certainty. Upon testing on other environments, it can be concluded that it is highly possible that target to reach was still very sparse for the agent to learn the optimal policy to reach that subgoal. The environment is not solvable as of writing this paper, within feasible limits, through regular RL algorithms — regardless of whether we apply our strategy on top of it or not.

4.2 Analysis in Super Mario Bros Environment

The second environment on which we tested our proposed methodology depicted that our shaping of the state space has negligible negative impact on how the agent learns to traverse the environment. In our experiments, we tested with both DQN and DDQN. Both of them were able to solve the environment's level-1 without any special changes to hyperparameters. Reward shaping played an important role in ensuring faster convergence to an optimal policy.

Upon applying our custom state space wrapper, the agent initially failed to cross over basic obstacles, as sufficient information regarding obstacles was not provided to

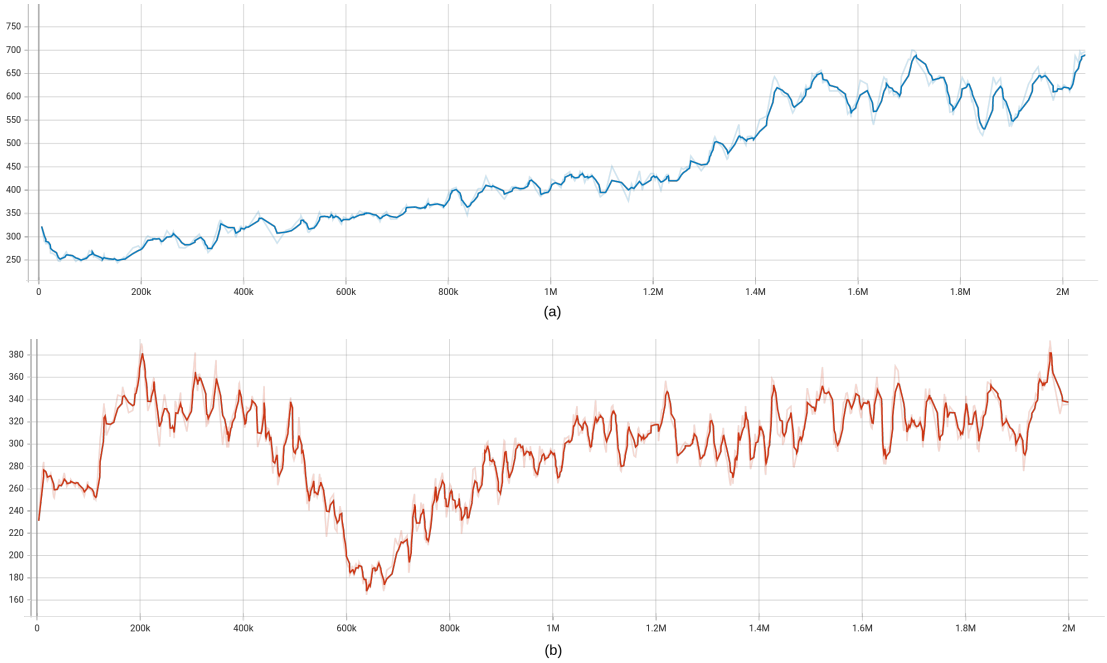


Figure 4.1: Comparison of Mean of reward per timestep. (a) The benchmark without our strategy. (b) With our strategy.

it. After successive exploratory steps, the agent started to learn the optimal policy eventually to reach the green flag and finish the level. With our compressed representation, the agent learnt to interact with boxes of interest regardless of the global objective, in order to maximize the cumulative reward received. At another phase of testing, the intrinsic rewards for Super Mario Bros was shaped in coordination with few other factors to ensure faster convergence.

Reward Function:

The new environmental reward function assumes the objective of the game is to move as far right as possible (increase the agent's X-value), as fast as possible, without dying. To model this, three separate variables compose the reward:

- v: the difference in agent X-coordinate values between states, gives the instantaneous velocity for the given step

  $$v = x_1 - x_0$$

  where, x_0 is the x-position before the step and x_1 is the x-position after the step. Thus summarising:

  $$v = \begin{cases} v > 0 & \text{moving right} \\ v < 0 & \text{moving left} \\ 0 & \text{not moving} \end{cases}$$

- c: The difference in the game clock between frames, this penalty prevents the

agent from standing still

$$c = c_0 - c_1$$

where c_0 and c_1 is the clock reading before and after the step, respectively.

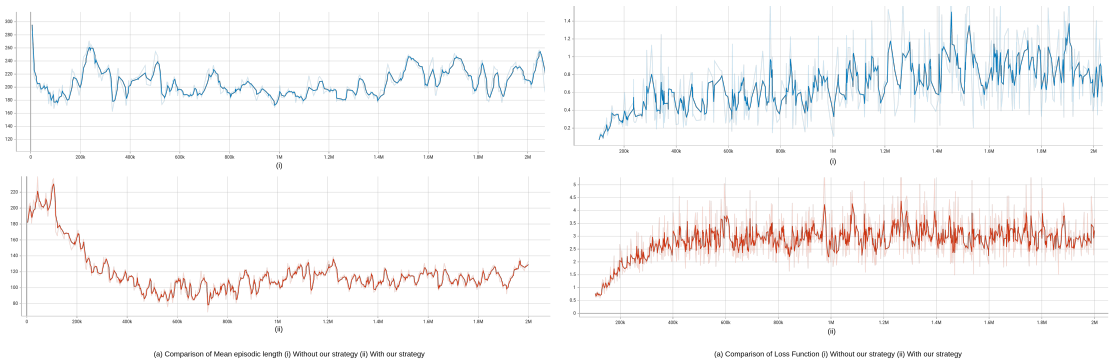
$$c = \begin{cases} c = 0 & \text{no clock tick} \\ c < 0 & \text{clock tick} \end{cases}$$

- d: The death penalty that penalizes the agent for dying in a state

$$d = \begin{cases} d = 0 & alive \\ d = -15 & dead \end{cases}$$

Finally the environmental reward is calculated of the form:

$$r = v + c + d$$



(a) Comparison of Mean episodic length (i) Without our strategy (ii) With our strategy

(a) Comparison of Loss Function (i) Without our strategy (ii) With our strategy

Figure 4.2: Comparison with and without our strategy applied.

CHAPTER 5

Conclusion and Future Work

In this work, we proposed a new way of approaching state-space shaping by incorporating object detection strategies. Use of the transformed representations, our work shows that Deep RL strategies can work with as minimal information as possible as long as its sufficient and unique for each state. Our method matches performance close to methods that rely solely on raw pixel obsevations. Furthermore, our method has the scope of generalizability across different levels or variations of the game environments, given that the transition dynamics and objects don't change.

Although our approach showed promising results, there are still several avenues for future research. One possible direction is to investigate the impact of different object detection strategies on the performance of the proposed method. Switching to alternative object detection approaches could have detrimental effects or may infact help improve over our existing strategy. Additionally, our method's success in video game environments highlights its potential applicability in other domains, such as robotics or autonomous driving. Future research could explore the method's generalization across different domains and environments.

Finally, an important avenue for future research is to investigate the interpretability and explainability of our approach. Understanding how our method makes decisions and what features it uses to shape the state space could help us gain insights into the behavior and limitations of the algorithm. This could also lead to more transparent and trustworthy AI systems, which are increasingly important in real-world applications.

In conclusion, our proposed approach for state space shaping incorporating object detection strategies and HRL with Deep RL shows great potential for solving challenging video game environments and could be applied in various other domains. We hope that this work inspires further research in this area and contributes to the development of more robust and efficient autonomous systems.

REFERENCES

1. **Anand, A.**, **E. Racah**, **S. Ozair**, **Y. Bengio**, **M.-A. Côté**, and **R. D. Hjelm** (2020). Unsupervised State Representation Learning in Atari.

2. **Bialek, W.** and **N. Tishby**, Predictive information. 1999.

3. **Botvinick, M. M.**, **Y. Niv**, and **A. C. Barto** (2009). Hierarchically organized behavior and its neural foundations: A reinforcement learning perspective. *Cognition*, **113**, 262–280.

4. **Diuk, C.**, **A. Cohen**, and **M. L. Littman**, An object-oriented representation for efficient reinforcement learning. *In Proceedings of the 25th International Conference on Machine Learning*, ICML '08. Association for Computing Machinery, New York, NY, USA, 2008. ISBN 9781605582054. URL `https://doi.org/10.1145/1390156.1390187`.

5. **Goel, S.** and **M. Huber**, Subgoal discovery for hierarchical reinforcement learning using learned policies. *In FLAIRS conference*. 2003.

6. **Guestrin, C.**, **D. Koller**, **R. Parr**, and **S. Venkataraman** (2003). Efficient solution algorithms for factored MDPs. *Journal of Artificial Intelligence Research*, **19**, 399–468. URL `https://doi.org/10.1613%2Fjair.1000`.

7. **Kulkarni, T. D.**, **K. R. Narasimhan**, **A. Saeedi**, and **J. B. Tenenbaum** (2016). Hierarchical Deep Reinforcement Learning: Integrating Temporal Abstraction and Intrinsic Motivation.

8. **Lake, B. M.**, **T. D. Ullman**, **J. B. Tenenbaum**, and **S. J. Gershman** (2017). Building machines that learn and think like people. *Behavioral and Brain Sciences*, **40**, e253.

9. **Malinowski, M.**, **C. Doersch**, **A. Santoro**, and **P. Battaglia**, Learning visual question answering by bootstrapping hard attention. *In Proceedings of the European Conference on Computer Vision (ECCV)*. 2018.

10. **Mnih, V.**, **K. Kavukcuoglu**, **D. Silver**, **A. A. Rusu**, **J. Veness**, **M. G. Bellemare**, **A. Graves**, **M. Riedmiller**, **A. Fidjeland**, **G. Ostrovski**, **S. Petersen**, **C. Beattie**, **A. Sadik**, **I. Antonoglou**, **H. King**, **D. Kumaran**, **D. Wierstra**, **S. Legg**, and **D. Hassabis** (2015). Human-level control through deep reinforcement learning. *Nature*, **518**(7540), 529–533. ISSN 0028-0836, 1476-4687.

11. **Mylavarapu, S.**, **M. Sandhu**, **P. Vijayan**, **K. M. Krishna**, **B. Ravindran**, and **A. Namboodiri**, Towards Accurate Vehicle Behaviour Classification With Multi-Relational Graph Convolutional Networks. *In 2020 IEEE Intelligent Vehicles Symposium (IV)*. 2020*a*. ISSN 2642-7214.

12. **Mylavarapu, S.**, **M. Sandhu**, **P. Vijayan**, **K. M. Krishna**, **B. Ravindran**, and **A. Namboodiri**, Understanding dynamic scenes using graph convolution networks. *In 2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2020*b*.

13. **Samothrakis, S.**, **D. Perez-Liebana**, **S. M. Lucas**, and **M. Fasli**, Neuroevolution for general video game playing. *In 2015 IEEE Conference on Computational Intelligence and Games (CIG)*. 2015.

14. **Spelke, E. S.** and **K. D. Kinzler** (2007). Core knowledge. *Developmental Science*, **10**(1), 89–96. URL https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1467-7687.2007.00569.x.

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

16. **Woodward, A. L.** (1998). Infants selectively encode the goal object of an actor's reach. *Cognition*, **69**(1), 1–34. ISSN 0010-0277. URL https://www.sciencedirect.com/science/article/pii/S0010027798000584.

17. **Zaheer, M.**, **S. Kottur**, **S. Ravanbakhsh**, **B. Poczos**, **R. R. Salakhutdinov**, and **A. J. Smola**, Deep sets. *In* **I. Guyon**, **U. V. Luxburg**, **S. Bengio**, **H. Wallach**, **R. Fergus**, **S. Vishwanathan**, and **R. Garnett** (eds.), *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper/2017/file/f22e4747da1aa27e363d86d40ff442fe-Paper.pdf.

18. **Şimşek, O.**, **A. P. Wolfe**, and **A. G. Barto**, Identifying useful subgoals in reinforcement learning by local graph partitioning. *In Proceedings of the 22nd international conference on Machine learning - ICML '05*. ACM Press, Bonn, Germany, 2005. ISBN 978-1-59593-180-1. URL http://portal.acm.org/citation.cfm?doid=1102351.1102454.