THÈSE DE DOCTORAT DE L'UNIVERSITÉ DE LYON
opérée au sein de
INSA LYON

École Doctorale 512
Informatique et Mathématique de Lyon
(INFOMATHS)

Spécialité
Informatique

Présentée par

Edward Emanuel Beeching

Pour obtenir le grade de
DOCTEUR de L'UNIVERSITÉ DE LYON

Sujet de la thèse :

Large-scale Automatic Learning of Autonomous Agent Behavior with Structured Deep Reinforcement Learning

Soutenue publiquement le XX XXX 2022, devant le jury composé de :

Mme. Elisa Fromont	INRIA IRISA - Université de Rennes 1	Rapporteur
M. David Filliat	INRIA FLOWERS - ENSTA Paris	Rapporteur
M. Cédric Démonceaux	Université de Bourgogne	Présdient
M. Karteek Alahari	INRIA THOTH	Examinateur
Mme. Christine Solnon	INRIA CHROMA - INSA-Lyon	Examinateur
M. Olivier Simonin	INRIA CHROMA - INSA-Lyon	Directeur de thèse
M. Jilles Dibangoye	INRIA CHROMA - INSA-Lyon	Co-encadrant de thèse
M. Christian Wolf	Naver Labs Europe	Co-encadrant de thèse

ABSTRACT

Autonomous robotic agents have begun to impact many aspects of our society, with application in automated logistics, autonomous hospital porters, manufacturing and household assistants. The objective of this thesis is to explore Deep Reinforcement Learning approaches to planning and navigation in large and unknown 3D environments. In particular, we focus on tasks that require exploration and memory in simulated environments. An additional requirement is that learned policies should generalize to unseen map instances. Our long-term objective is the transfer of a learned policy to a real-world robotic system. Reinforcement learning algorithms **learn by interaction**. By acting with the objective of accumulating a task-based reward, an Embodied AI agent must learn to discover relevant semantic cues such as object recognition and obstacle avoidance, if these skills are pertinent to the task at hand.

This thesis introduces the field of Structure Deep Reinforcement Learning and then describes 5 contributions that were published during the PhD. We start by creating a set of challenging memory-based tasks whose performance is benchmarked with an unstructured memory-based agent. We then demonstrate how the incorporation of structure in the form of a learned metric map, differentiable inverse projective geometry and self-attention mechanisms; augments the unstructured agent, improving its performance and allowing us to interpret the agent's reasoning process.

We then move from complex tasks in visually simple environments, to more challenging environments with photo-realistic observations, extracted from scans of real-world buildings. In this work we demonstrate that augmenting such an agent with a topological map can improve its navigation performance. We achieve this by learning a neural approximation of a classical path planning algorithm, which can be utilized on graphs with uncertain connectivity.

From work undertaken over the course of a 4-month internship at the research and development department of Ubisoft, we demonstrate that structured methods can also be used for navigation and planning in challenging video game environments. Where we couple a lower level neural policy with a classical planning algorithm to improve long-distance planning and navigation performance in vast environments of 1km$\times$1km. We release an open-source version of the environment as a benchmark for navigation in large-scale environments.

Finally, we develop an open-source Deep Reinforcement Learning interface for the Godot Game Engine. Allowing for the construction of complex virtual worlds and the learning of agent behaviors with a suite of state-of-the-art algorithms. We release the tool with a permissive open-source (MIT) license, to aid researchers in their pursuit of complex embodied AI agents.

RÉSUMÉ

Les robots autonomes ont commencé à impacter de nombreux aspects de notre société avec, par exemple des applications dans la logistique automatisée, les robots hospitaliers autonomes, l'industrie ou encore les aides ménagères. L'objectif de cette thèse est d'explorer les approches d'apprentissage par renforcement profond pour la planification et la navigation dans des environnements 3D vastes et inconnus. Nous nous concentrons en particulier sur les tâches qui nécessitent d'explorer et mémoriser les environnements simulés. Une contrainte supplémentaire est que les stratégies apprises doivent se généraliser à des cartes inconnues. Notre objectif à long terme est le transfert d'une technique d'apprentissage vers un système robotique dans le monde réel. Les algorithmes d'apprentissage par renforcement **apprennent des interactions**. En agissant avec l'objectif d'accumuler des récompenses liées à une tâche, une IA incarnée doit apprendre à découvrir des informations sémantiques telles que la reconnaissance d'objets et l'évitement d'obstacles, si ces compétences sont pertinentes pour l'accomplissement de la tâche.

Cette thèse introduit le domaine de l'Apprentissage par Renforcement Profond Structuré (Structured Deep Reinforcement Learning) et décrit ensuite cinq contributions qui ont été publiées au cours de la thèse. Nous commençons par créer un ensemble de tâches complexes nécessitant de la mémoire pour comparer les performances avec un agent à la mémoire non structurée. Nous démontrons ensuite comment l'incorporation d'une structure telle qu'une carte métrique apprise, une géométrie projective inverse différentiable et des mécanismes d'auto-attention améliorent les performances de l'agent, ce qui nous permet d'analyser son processus de raisonnement.

Nous passons ensuite d'environnements visuellement simples à des environnements plus difficiles avec des observations photoréalistes extraites de scans de bâtiments du monde réel. Dans ce travail, nous démontrons qu'améliorer un agent avec une carte topologique peut améliorer ses performances de navigation. Nous y parvenons en lui apprenant une approximation neuronale d'un algorithme de planification de chemin classique, qui peut être utilisé sur des graphes avec une connectivité incertaine.

Ensuite, à partir des travaux menés lors d'un stage de quatre mois au sein du département recherche et développement d'Ubisoft, nous démontrons que les méthodes structurées peuvent également être utilisées pour la navigation et la planification dans des environnements de jeux vidéo complexes. Nous combi-

nons une politique neuronale de bas niveau avec un algorithme de planification classique pour améliorer la planification à longue distance et les performances de navigation dans de vastes environnements de 1km×1km. Nous publions une version open source de l'environnement comme référence pour la navigation dans des environnements à grande échelle.

Pour finir, nous développons une interface open source Deep Reinforcement Learning pour le Godot Game Engine. Elle permet la construction de mondes virtuels complexes et l'apprentissage de comportements agents avec une suite d'algorithmes de pointe. Nous publions l'outil avec une licence permissive open source (MIT) pour aider la recherche sur l'IA des agents incarnés complexes.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACRONYMS

API	Application Programming Interface
GPU	Graphics Processing Unit
GNN	Graph Neural Network
RL	Reinforcement Learning
CNN	Convolutional Neural Network
A2C	Advantage Actor Critic
AI	Artificial Intelligence
NLP	Natural Language Processing
CV	Computer Vision
DQN	Deep Q-Learning
GPU	Graphics Processing Unit
GRU	Gated Recurrent Unit
LSTM	Long-Short Term Memory
MDP	Markov Decision Process
MLP	Multi-Layer Perceptron
NN	Neural Network
POMDP	Partially Observable Markov Decision Process
PPO	Proximal Policy Optimization
RL	Reinforcement Learning
RNN	Recurrent Neural Network
SGD	Stochastic Gradient Descent
SLAM	Simultaneous Localization And Mapping
TD	Temporal-Difference
VQA	Visual Question Answering

CHAPTER 1

INTRODUCTION

Contents

1.1 Context

We are influenced by automated decision-making in almost every aspect of our lives. From the classical planning algorithms used in satellite navigation systems to the complex decision-making algorithms that recommend which advert to display next. As society becomes more reliant on technology and humanity extends its lifespan thanks to medical advancements, we will require physical assistance to be comfortable and safer in our daily lives. Automated systems have started to replace low-skilled jobs sectors such as supermarket checkouts, car manufacturing and warehouse logistics. These applications exist in highly controlled environments, perform one specific task very well and require years of handcrafted design by teams of engineers. The next steps in robotics will require more general systems that can adapt to new scenarios and environments. These physical AI systems will be required in order to provide home carers for an aging population, nursing assistants in hospitals and automated delivery of products.

As robotic systems are allowed to perform more general tasks they will be required to learn from their inevitable mistakes. Learning-based systems exist already that impact our daily lives, for example in automated translation systems, photo categorization and movie recommendation in our favorite streaming platforms. The systems are considered to be "narrow" AI, which perform a specific task extremely well. But they are not considered to be **Embodied AI**, where an AI agent interacts with a physical environment and performs sequential decision making, where our actions influence the state of our environment. The creation of Embodied AI systems poses many challenges, in particular safety guarantees,

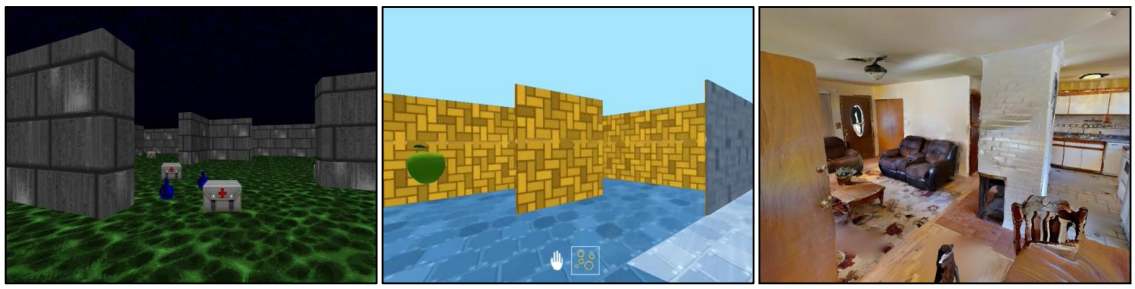


Figure 1.1: **The progression of visual representations in Embodied AI simulators.** Recent years have shown a rapid improvement in the quality of simulated environments and the observations rendered by a simulated monocular camera. Figures from (Kempka et al. 2017; Beattie et al. 2016a; Savva et al. 2019a)

availability of hardware and the time it takes to iterate on new ideas and algorithms. For this reason, researchers work predominantly in simulation. In the last five years there has been a proliferation of Embodied AI simulators, ranging from video-game-like environments (Kempka et al. 2017; Beattie et al. 2016a) to photo-realistic environments (Savva et al. 2019a) based on real-world 3D scans, see Figure 1.1. While photo-realism is of great importance when evaluating a vision-based agent that will one day be transferred into the real world, we must also focus on the difficulty of the tasks we propose to the Embodied AI system. Tasks in this field range from more simple GPS-guided point-to-point navigation, to challenging tasks that require memorization of key objects and localization under noisy motor actuation. Traditional approaches in robotics typically solve these problems by building a map of the environment and then performing localization, planning and navigation. Classical methods, however, do not allow for aspects of navigation the humans perform innately, we recognize similarities in the layout, identify free space and semantic classes in our environment to guide our decision-making process. We have learned general concepts such as that in a house, bedrooms are typically upstairs and kitchens downstairs. We can then exploit these regularities when navigating in a new environment.

Learning-based methods aim to exploit such structural and semantic regularities using end-to-end approaches, which typically take as input pixels from a monocular camera and directly output low-level actions such as rotating and movement in a particular direction.

These works build on recent advancements in Computer Vision (CV) and Supervised Learning. In particular, the application of Convolutional Neural Network (CNN) to perform non-linear feature extraction from images. Since the creation of AlexNet (Krizhevsky et al. 2012), there has been a massive advancement in the field of computer vision, which has influenced many other domains.

The recent advances in Deep Learning come from three main sources: the availability of large-scale annotated datasets to perform training, the action of performing batched stochastic gradient-based optimization of the Neural Network parameters. The availability of computing resources and the utilization of Graphics Processing Unit (GPU) for fast matrix multiplication, the fundamental operation used in Neural Networks. The availability of high quality open-source Deep Learning libraries such as TensorFlow (Abadi et al. 2015), PyTorch (Paszke et al. 2019), Jax (Bradbury et al. 2018) and MXNet (Chen et al. 2015). This trifecta has enabled a decade of enormous progress in the field of Artificial Intelligence. A further factor that has influenced advances in Deep Learning and Embodied Artificial Intelligence (AI) are network architectures that can learn to integrate sequences of data, with the most well-known being the Long-Short Term Memory (LSTM) module (Hochreiter et al. 1997a). Humans appear to focus their attention on particular objects of interest when analyzing a scene, group of objects or text. Recent successes in Natural Language Processing (NLP) are loosely inspired by principle of attention with leaps in performance being achieved in the field with the transformer network architecture (Vaswani et al. 2017a). Deep Neural Networks have drastically changed the fields of supervised and unsupervised learning and they have led to great advancements in the domain of Deep Reinforcement Learning. With one of the most cited publication being that of Deep Q-Learning (Mnih et al. 2015a), where a CNN is trained to play 57 Atari games, some at a superhuman level. The history of game-playing with neural networks is started my earlier, with the first instance of neural networks learning to plan games was TD-Gammon (Tesauro et al. 1995), which played backgammon. We leave further discussion to related works chapter.

1.2 Motivation

Successes in challenging video games have encouraged research in Reinforcement Learning (RL) and supervised learning for robotic applications. An early study using end-to-end learning for robotic control was published in 2016 (Levine et al. 2016a) who learned policies for grasping actions for a robotic arm. In 2016 (Zhu et al. 2017) published a work that performs end-to-end target-driven navigation, where the policy is learned, in simulation, to navigate to a target image location. A further work published in 2016 (Mirowski et al. 2016) learns an end-to-end policy in simulated 3D environments, with the end focus is on machine intelligence, rather than potential transfer to a real-world setting. The three aforementioned works, shown in Figure 1.2, provide the cornerstones, to an influx of learning-based approaches to navigation over the last five years, which are discussed in detail in chapter 2. While some of the strengths of learning-based approaches have been mentioned, it is pertinent to highlight the downsides. Deep

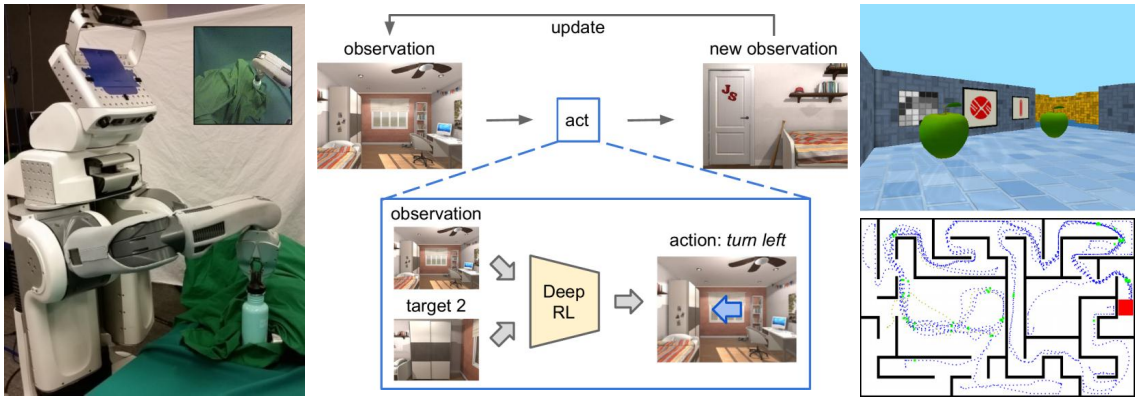


Figure 1.2: **Three pivotal works in the field of Deep Reinforcement Learning for robotic control** Recent years have shown a rapid improvement in the quality of simulated environments and the observations rendered by a simulated monocular camera. Figure from (Levine et al. 2016a; Zhu et al. 2017; Mirowski et al. 2016)

RL approaches are notoriously sample inefficient (Henderson et al. 2017), requiring tens to hundreds of millions of environment interactions in order to learn an effective policy. While this is an active area of research, the solution to date is massive scale and parallelization (Wijmans et al. 2019a; OpenAI 2018; Vinyals et al. 2019). A further challenge is a sensitivity to hyper-parameter configurations. In addition, end-to-end approaches typically treat the neural network model as a black box, so it can be difficult to interpret the reasoning behavior of a learning end-to-end policy.

In order to trust a system, we would like to understand how it works, why it makes the choices it makes, ideally the logical steps it follows to make a decision are similar to the steps we would take. Handcrafted solutions to planning and navigation can be decomposed into their individual parts in order to diagnose where there are deficiencies in the system. End-to-end learning-based approaches may outperform their handcrafted counterparts but lack interpretability.

This thesis aims to find a compromise between unstructured end-to-end learning approaches and handcrafted solutions. The compromise we aim to find leans more towards learning-based approaches, where typically we take a learning-based approach and add structure rather than adding learning to a handcrafted method. We believe that by imbuing our network architecture with additional structure and priors related to vision, planning and navigation, we will not only realize a more interpretable agent, but we will also increase both its performance and the sample efficiency. It would seem utopic to expect to achieve these benefits without a cost. In this instance, the cost comes in terms of the design and implementation of the structured learning system, which by definition is applicable to a narrower set of problems than its unstructured counterparts.

Structured network architectures have the potential to improve performance in a number of problem settings. For an example, consider a robotic agent deployed in a home, assigned with the task of finding a misplaced set of keys. If this is the first time the agent is in this environment, it must perform an exhaustive exploratory strategy that visits each room, performing object recognition in search of the set of keys. If however, the agent is already familiar with the environment, is can analyze its internal memory to try to identify if it has observed the keys and where they are located. In the case of a typical unstructured end-to-end Deep RL approach, the agent's internal memory is a vector and the agent must learn to interpret this unstructured information. If the agent is built with structured memory, the agent could potentially query for the object's location and use a structured neural approximation of a path planning algorithm to identify the sequence of way-points required to navigate to the location of the set of keys. By structuring the agents internal representation of the environment, we enable it to perform spatial reasoning about its environment and improve its understanding of the world it inhabits. In addition to improved performance, structured representations often provide interpretable information, that can inform external users about the agent's reasoning process.

Three key works in the domain have provided the foundation and inspiration to many of the models developed during this PhD.

Cognitive Mapping and Planning for Visual Navigation (Gupta et al. 2017a) was the first spatially structured neural network approach for robotic control. The authors created a Deep Neural Network architecture that contains a learned belief state, that is translated and rotated from one timestep to the next depending on ego-motion of the agent. An overview of the architecture is shown in Figure 1.3, the structured hidden state is passed from one timestep to the next, allowing the agent to perform spatial reason and planning. In this work, the policy was learned with a technique called imitation learning, where we are provided with sequences of observations and action labels, created by a (human) expert to supervise the training process. In this work the ego-motions of the agent are restricted to 90-degree rotations, which is one of the limitations we addressed in our *EgoMap* model.

The authors of *Neural map: Structured memory for deep reinforcement learning* (Parisotto et al. 2017a), develop a model with a spatially structured neural memory that is learned end-to-end with Deep reinforcement Learning, the memory is read with self-attention and global read operations with a CNN. A potential downside to this approach, that we address in *EgoMap*, is what the agent sees is mapped to where the agent is located, not to where the features from the observations are located in the environment.

In *Semi-parametric topological memory for navigation* (Savinov et al. 2018a), the authors build a graph-based representation by predicted connectivity between observations and thresholding the predictions to build a graph. They then use

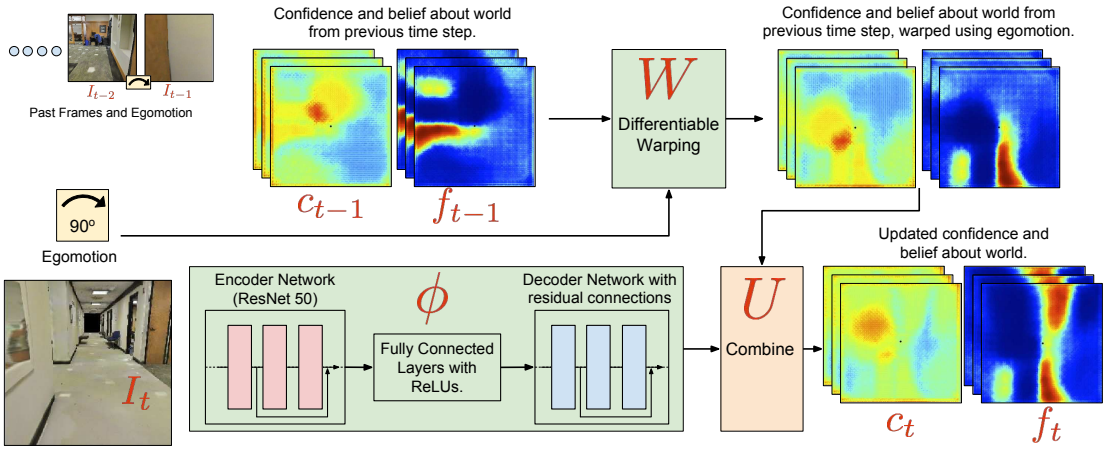


Figure 1.3: **Cognitive mapping and planning** Deep Neural network architecture from Cognitive mapping and planning. Figure from (Gupta et al. 2017a)

classical planning techniques to estimate paths to a target location, indicated by an image. In our *Learning to Plan* work we address two limitations of their work, planning under uncertainty by using the raw connectivity predictions and observations as input to a graph neural network and generalization to environment instances that were not observed during training.

The objective of this thesis was to explore Structured Deep Reinforcement Learning approaches to planning an navigation and by doing do, answer the following questions:

- Can Structured approaches improve the generalization performance of the agent when evaluated in unseen environment instances?
- Will sample efficiency be increased when learning a structured policy, when compared to an unstructured agent?
- Are we able to interpret the agent's reasoning process by visualizing its structured representations?

1.3 Contributions

This work was funded by "Institut national de recherche en sciences et technologies du numérique" (INRIA) Cordi-S grant. This work also participated scientifically to ANR projects: ANR Deepvision (ANR-15-CE23-0029, STPGP479356-15), a joint French/Canadian call by ANR & NSERC and ANR Delicio (ANR-19-CE23-0006).

In Chapter 2 we provide the reader with the essentials required to introduce the field of Structured Deep Reinforcement Learning. We give a high level overview of Supervised Learning, Reinforcement Learning, Deep Learning, Deep Reinforcement Learning and classical robotic navigation. The prerequisites required to introduce the contributions made during this thesis are vast, so we do not endeavor to provide a detailed descriptions of all related works.

In Chapter 3 we introduce a set of interactive reward-based tasks in 3D environments, that we have developed in order to benchmark memory-based Deep RL agents. These tasks involve navigation in procedurally generated labyrinths from an egocentric perspective, where observations are rendered with a virtual monocular camera. The tasks require the agent to learn to recognize objects and semantics through interaction with its environment. We conduct an analysis of the agent's internal "hidden" state representations, in order to understand how the agent reasons about its current task. Then, in Chapter 4 we adapt and extend the agent's neural network architecture with structured representations based on projective geometry, metric maps and agent ego-motion. This agent architecture called "EgoMap" augments the agent's performance on the memory-based tasks we created in the previous chapter and provided interpretable attention maps, which can be analyzed to comprehend the reasoning process of the agent.

After considering agents with metric maps, we explored the addition of topological memory as part of an agent's architecture. Where we developed a learned Neural Planner which learns to perform shortest path planning in graphs with uncertain connectivity and generalizes to unseen graphs sampled for new environment instances. We present this publication in Chapter 5, where we couple the Neural Planner with a low-level controller for planning and navigation in photo-realistic 3D environments.

Chapter 6 represents work undertaken during a 4-month internship at Ubisoft LaForge, Montreal. This work explores the application of Deep RL in vast video game environments with complex action spaces. In addition to creating the environment and learning an end-to-end RL policy, we augment the policy with a graph-based classical planner to perform hierarchical planning and low-level control.

In chapter 7 we develop an open-source Deep RL interface for a widely known open-source Game Engine, Godot. The tool we have developed allows for researchers, hobbyists and game designers to build and interface with worlds and games free of charge, learning agent behaviors with state-of-the-art Deep RL algorithms.

List of publications — This manuscript is based on the material published in the following papers:

- Edward Beeching, Jilles Dibangoye, Olivier Simonin, and Christian Wolf (2020a). "Deep Reinforcement Learning on a Budget: 3D Control and

Reasoning Without a Supercomputer." In: *Proceedings of the International Conference on Pattern Recognition (ICPR)* - Chapter 3;

- Edward Beeching, Jilles Dibangoye, Olivier Simonin, and Christian Wolf (2020c). "EgoMap: Projective mapping and structured egocentric memory for Deep RL". in: *Proceedings of the European Conference on Machine Learning and Knowledge Discovery in Databases (ECML-PKDD)* - Chapter 4;

- Edward Beeching, Jilles Dibangoye, Olivier Simonin, and Christian Wolf (2020d). "Learning to plan with uncertain topological maps." In: *Proceedings of the IEEE European conference on computer vision (ECCV)* - (spotlight presentation) - Chapter 5;

- Edward Beeching, Maxim Peter, Philippe Marcotte, Jilles Dibangoye, Olivier Simonin, Romoff Joshua, and Christian Wolf (2022b). "Graph augmented Deep Reinforcement Learning in the GameRLand3D environment". In: *Association for the Advancement of Artificial Intelligence Conference on Artificial Intelligence (AAAI). Workshop on Reinforcement Learning in Games* - Chapter 6;

- Edward Beeching, Jilles Dibangoye, Olivier Simonin, and Christian Wolf (2022a). "Godot Reinforcement Learning Agents". In: *Association for the Advancement of Artificial Intelligence Conference on Artificial Intelligence (AAAI). Workshop on Reinforcement Learning in Games* - Chapter 7;

CHAPTER 2

RELATED WORKS

Contents

In this chapter, we provide a foundation to the reader in order to introduce the domain of Structured Deep Reinforcement Learning. We start with a brief overview of Supervised Learning and Reinforcement Learning. We then introduce Deep Learning, first through the building blocks of Deep Neural Network architectures: Multi-Layer Perceptron (MLP)s, Convolutional Neural Network (CNN)s, Recurrent Neural Network (RNN)s, attention-based and Graph Neural Network (GNN). We then provide a broader overview of applications of Deep Learning in supervised settings. With this grounding, we can begin the introduction to Deep Reinforcement Learning, Embodied AI environments and finally Structured Deep Reinforcement Learning.

2.1 Supervised Learning

In a nutshell, supervised learning is the process of learning the internal parameters of a predictive model using labeled data and then using the model to estimate the labels of new data. Formally given a set of n pairs of examples $\boldsymbol{D} = \{\boldsymbol{x_i}, y_i\}$ where $\boldsymbol{x_i}$ is an input (vector, image, time-series) and y_i is an output label. We would like to learn a parametric function $f(\boldsymbol{x}; \boldsymbol{w})$ that maps an input $\boldsymbol{x}$ to an estimate of its output $\hat{y}$. Ideally, we would like to learn a predictive model that performs equally well on unseen *validation* and *test* data which have been drawn from the same underlying distribution. So that the model has identified patterns and regularities in the pairs of examples, rather than memorizing exactly the examples that were used for learning the model parameters. Using the framework of Empirical Risk Minimization, we wish to learn a hypothesis $h : \boldsymbol{X} \rightarrow \boldsymbol{Y}$, assuming there is a joint probability distribution $\boldsymbol{P}(\boldsymbol{x}, y)$ and the examples $(\boldsymbol{x_i}, y_i)$ in our dataset are drawn from $\boldsymbol{P}(\boldsymbol{x}, y)$. Broadly, supervised learning is split into two categories *regression* and *classification*. Both of these are relevant to applications of Deep Reinforcement Learning (RL), as we might treat the problem of estimating the value of a given state as regression, or consider learning a policy to estimate the best discrete action to take in a given state a form of classification problem.

2.1.1 Regression

When we perform regression, the estimates of our predictive model are scalar values, for this text we will restrict ourselves to estimating a single scalar value, $\hat{y}_i$, per example $\boldsymbol{x_i}$:

$$\hat{y}_i = f(\boldsymbol{x_i}; \boldsymbol{w}). \tag{2.1}$$

In the most simple case, where both x_i and y_i are single scalar values and the model is linear, this surmounts to learning the slope and intercept of $f(x; \boldsymbol{w})$,

$$\hat{y} = f(x; \boldsymbol{w}) = w_0 + xw_1. \tag{2.2}$$

The most common algorithm for learning these parameters is that of Least Squares Linear Regression or simply Linear Regression. Which aims to minimize the square differences between the model target predictions $\hat{y}_i$ the target y_i. Formally we have to optimize for the parameters:

$$\min_{\boldsymbol{w}} \|\boldsymbol{X}\boldsymbol{w}^T - y\|_2^2. \tag{2.3}$$

Where $\boldsymbol{X}$ is a matrix that contains all of the n examples. In order to learn the intercept, a value of 1 is appended to each input $\boldsymbol{x}$. There are a variety of techniques to identify the optimal parameters $\boldsymbol{w}^*$. If memory is not a concern and all data points are available the closed-form solution is:

$$\boldsymbol{w} = (\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{Y}. \tag{2.4}$$

Where $\boldsymbol{X}$ and $\boldsymbol{Y}$ are matrices that are a concatenation of input and outputs respectively. Alternatively, the parameters can be found iteratively through Stochastic Gradient Descent (SGD) where we take steps toward the optimal parameters $\boldsymbol{w}^*$ by making estimates and computing their error derivatives. To do this, we must first define a **loss function** $\mathcal{L}(x, y; \boldsymbol{w})$:

$$\mathcal{L}(x, y; \boldsymbol{w}) = (y - \hat{y})^2 = (y - f(x; \boldsymbol{w})^2 = (y - w_0 - xw_1)^2. \tag{2.5}$$

We can then calculate the analytic partial derivatives of the loss function with respect to its learn-able parameters $\boldsymbol{w}$. Where in the case of the two-parameter example are:

$$\begin{aligned} \frac{\partial \mathcal{L}(x_i, yi; \boldsymbol{w})}{\partial w_0} &= y_i - \hat{y}_i, \\ \frac{\partial \mathcal{L}(x_i, yi; \boldsymbol{w})}{\partial w_1} &= x_i(y_i - \hat{y}_i). \end{aligned} \tag{2.6}$$

In the linear setting, the loss function is *convex*, which means there is a single global minimum. Performing gradient descent from any initialization of the parameters $\boldsymbol{w}$ will eventually lead to the optimal solution. It is worth noting that the optimal parameters for a particular dataset do not necessarily correspond to the best performing parameters on data not observed during the optimization process. As we move to non-convex optimization in the Deep Learning section 2.3, we will be performing optimization of non-convex functions. While we have discussed here closed-form and first-order gradient-based solutions, there are a number of other methods and solvers that will find solutions to convex optimization problems in far few iterations (Gill et al. 2019) is an excellent reference for further information.

2.1.2 Classification

In the classification setting, instead of scalar values, the labels y represent several distinct classes. The example shown here is that of binary classification where there are two classes. But this can be extended to the multi-class setting. A common algorithm for binary classification is that of *Logistic Regression*, which is a generalized linear model for classification. The formulation is similar to that of the regression setting:

$$\hat{y}_i = f(\boldsymbol{x_i}; \boldsymbol{w}). \tag{2.7}$$

But in this case, we desire the prediction $\hat{y}$ to be a value from the set $\{0, 1\}$. This is typically achieved by using a logistic or sigmoid function on the output of f, which restricts the output range to be in 0-1, predictions from the trained model can be thresholded to compute the predicted class label:

$$\hat{y}_i = f(\boldsymbol{x_i}; \boldsymbol{w}) = \frac{1}{1 + e^{-(\boldsymbol{x_i^T w})}}. \tag{2.8}$$

Learning the parameters of this model can be achieved through gradient decent. To the authors' knowledge, unlike the case of Linear Regression, there is no general closed-form solution to the problem of Logistic Regression, although solutions exist for some specific cases (Lipovetsky 2015).

2.1.3 Regularization

Often the number of parameters available to the model exceeds the number required to model the underlying phenomenon. A-priori, unless the problem is trivial, it is unlikely that the required number of parameters can be estimated and must be found empirically. A technique to reduce the impact of an over-parameterized model is that of weight-decay, where the norm (typically l2) of the model's weight coefficients is penalized by adding it to the loss function, scaled by a Lagrange factor λ. By iteratively decaying the weights by a proportion of their magnitude, at each update step. This method restricts the number of large weight coefficients which in turn reduces over-fitting. Weight regularization has become ubiquitous when creating large deep networks and modifies the loss function as follows:

$$\mathcal{L}_{combined}(\boldsymbol{X}, \boldsymbol{Y}; \boldsymbol{w}) = \mathcal{L}_{predictions}(\boldsymbol{X}, \boldsymbol{Y}) + \lambda |\boldsymbol{w}^T \boldsymbol{w}|^2. \tag{2.9}$$

2.2 Reinforcement Learning

The objective of RL is to learn a policy π that maps states $\boldsymbol{s}$ to actions a in a reward-based environment, with the actions that are selected maximizing the

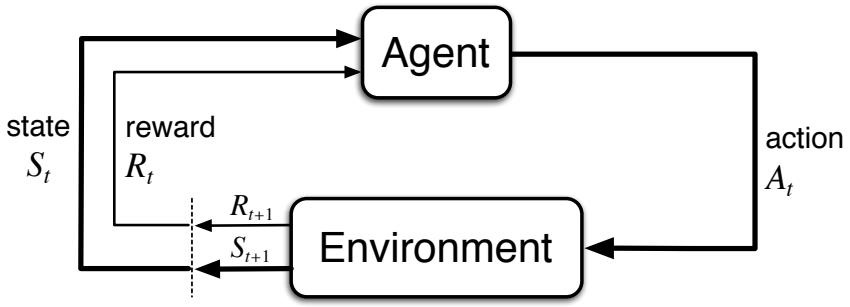


Figure 2.1: **Agent-environment interaction in a Markov decision process.** Figure from (Barto 2021).

cumulative reward received as the agent interacts with the environment. In this section, we provide a formal definition of a decision-making problem, a concrete example of an interactive environment and detail a tabular reinforcement learning algorithm that solves the example problem. Figure 2.1 shows the typical environment-agent interaction loop. An RL agent typically performs a sequence of actions in the environment over the course of an episode or trajectory, which ends in a termination condition. The termination condition typically corresponds to a state corresponding to success or failure, but may be triggered when a maximum number of actions per episode has been exceeded.

2.2.1 Markov Decision Processes

A Markov Decision Process (MDP) provides a formal definition for a sequential decision making process, defined as a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$ where:

- $\mathcal{S}$: a finite set of states
- $\mathcal{A}$: a finite set of actions
- $\mathcal{P}$: a state transition probability matrix, $P(s'|s, a)$
- $\mathcal{R}$: a reward function
- γ: a discount factor $\in (0, 1]$

An important property of an MDP is that the process is *Markov*, that is the optimal action to take for a particular state does not depend on the history of actions and states the agent has previously visited, the current state provides all the information the agent is required to act. As we move to Partially Observable Markov Decision Process (POMDP)s in section 2.5 we will see how the inclusion

of a partial observation of state, breaks the Markov property at least from the agent's perspective. The policy $\pi(\boldsymbol{s})$ can represent a stochastic distribution of possible action or a deterministic mapping from states to actions, depending on the algorithm and setting. When learning a policy that aims to solve an MDP we typically aim to maximize the expected cumulative sum of rewards, for trajectories or episodes in the environment, known as the total reward criterion for finite-horizon MDP (Puterman 1990). Ideally, we would find a policy that maximizes the expected sum of rewards for each trajectory, τ, a sequence of states and actions sampled from our policy π interacting in the environment from an initial state $\boldsymbol{s_0}$ to a terminal state $\boldsymbol{s_T}$:

$$\mathbb{E}_{\tau\sim\pi}\left[\sum_{t=0}^{T} r\left(\boldsymbol{s}_t, a_t\right)\right]. \tag{2.10}$$

The work presented in this thesis looked the optimize the discounted reward criterion. Provided with a trajectory τ of experience of length T, sampled for a given policy π in an MDP. The discounted return G_t can be calculated for each step on the trajectory, as the sum of all the reward received after a particular time-step t, discounted by how far into the future they were received, with the discount factor γ:

$$G_t = \sum_{t'=t}^{T} \gamma^{t'-t} r(\boldsymbol{s_{t'}}, a_{t'}). \tag{2.11}$$

We can also estimate the expected return of a given state $\boldsymbol{s}$ when acting under a particular policy π, this is the **value** of a state, $V^{\pi}(\boldsymbol{s})$:

$$V^{\pi}\left(\boldsymbol{s}_t\right) = \sum_{t'=t}^{T} \mathbb{E}_{\pi}\left[\gamma^{t'-t} r\left(\boldsymbol{s}_{t'}, a_{t'}\right) \mid \boldsymbol{s}_t\right]. \tag{2.12}$$

The discount factor γ means that we now aim to maximize a discounted total reward and is included for several reasons:

- Credit assignment, by limiting the temporal extent of a reward, we facilitate the identification of the appropriate actions that led to that reward.

- Bounding $V^{\pi}\left(\boldsymbol{s}\right)$, the discount factor ensures an upper limit in the infinite horizon setting.

- A further loose reasoning for the discount factor is that future rewards are of less value than rewards now, this has some relations with interest, inflation and growth of population. The reality is that, particularly in the Deep RL setting, the optimization problem is less challenging.

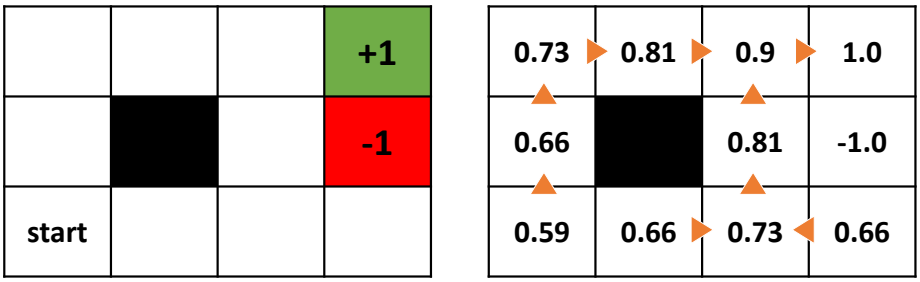


Figure 2.2: **An example grid-world problem** Left: An example RL problem, a grid-world with two terminal states in the top right and center right, with the starting state in the bottom left corner. Actions are deterministic and include moving left, right, up and down. Right: the values of each state for the optimal policy π^* with a γ factor of 0.9, with optimal actions show as orange arrows.

2.2.2 A concrete example

To ground the theory of Markov Decision Processes and RL algorithms we introduce the following toy example sequential decision-making problem, shown in Figure 2.2. In this grid world problem the agent starts in the bottom left and we wish to learn a policy that navigates to the top right cell. There are two terminal states, where the episode ends and the agent needs to be "reset" to its starting point, these are located on the upper right side indicated by the positive and negative rewards. As the objective is to learn a policy that maximizes cumulative reward, the end policy we aim to learn will navigate to the cell with a +1 reward. In Figure 2.2 we also show the value estimates of each state under a policy which outputs optimal actions π^*.

We will now introduce a well known method for solving this example MDP that can be applied to this form of decision making problem, tabular Q-learning (Watkins et al. 1992).

2.2.3 Tabular Q-learning

In tabular Q-learning (Watkins et al. 1992) for each state we aim to estimate the discounted return we will receive when the agent takes a particular action, transitions to the next state and then follows the actions that it estimates will maximize future discounted reward for the current policy. For each state, we have a table of estimated return for each state-action pair, the "best" action to chose is the action with the highest estimate of future return. Formally we define the following action-value function $Q(s, a)$ and perform updates according to the following learning rule:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t) \right]. \tag{2.13}$$

Where γ is the discount factor and α is a learning rate parameter. This equation does not, however, detail how to sample the transitions s_t, a_t, r_{t+1} and s_{t+1}. This leads us to a fundamental dilemma in RL, the **exploration-exploitation trade-off**. Should we be confident in the estimates of current Q-function and take the actions that we estimate will lead to the most cumulative reward, or should we perform exploration in order to discover potentially higher sources of reward? Q-learning, and its Deep Q-learning variants, discussed in section 2.5, use an extremely simple exploratory strategy called ϵ-greedy. In ϵ-greedy exploration at the start of learning typically 95% of the time a random action is selected from a uniform distribution of the available actions and 5% of the time "best" action is chosen, according to the current Q-value function. In most cases, the percentage decreases linearly over time, so that as the Q-value estimates become more robust, fewer random actions are taken. The entire algorithm is shown in 2.1. We will discuss the exploration-exploitation trade-off further in section 2.5.

Algorithm 2.1 Tabular Q-learning

```
Initialize Q(s, a) for all s ∈ S, a ∈ A, arbitrarily
while Q is not converged do
    Initialize S
    while s is not terminal do
        Choose action a for s using policy derived from Q (e.g. ε-greedy)
        Take action a, observe r, s'
        Q(s_t, a_t) ← Q(s_t, a_t) + α [r_{t+1} + γ max_a Q(s_{t+1}, a) − Q(s_t, a_t)]
        s ← s'
    end while
end while
```

2.3 Deep Learning

In this thesis, we investigate Deep RL approaches to planning and navigation problems. Which combines and extends classical RL algorithms with the non-linear function approximation of Deep Neural Networks, expanding the problem settings that RL can be applied in. Before we can introduce Deep RL, we must first cover the groundwork of Deep Learning. In this section, we introduce some of the key building blocks for the construction of neural network architectures.

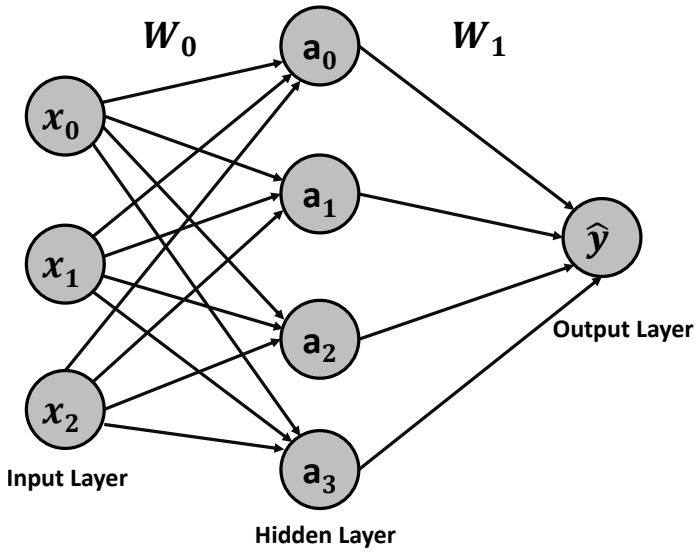


Figure 2.3: **A fully connected multi-layer perceptron** with one hidden layer, predicting a single scalar value. Arrows correspond to weight-input multiplications. Biases and non-linearities omitted for clarity.

2.3.1 Multi-Layer Perceptrons

Fully connected neural networks build upon Rosenblatt's perceptron neuronal model (Rosenblatt 1958). In section 2.1 we introduced an example of supervised learning, Linear Regression, which performs a linear multiplication of the input vector $\boldsymbol{x}$ with a weight vector $\boldsymbol{w}$. In a fully connected layer of a neural network, many of these multiplications are performed in parallel with a weight matrix $\boldsymbol{W}$. Figure 2.3 shows an MLP, which stacks these linear operations repeatedly such that there are many layers of matrix multiplications in the function $f(\boldsymbol{x}; \boldsymbol{W})$, which could naively be performed as following:

$$\hat{y} = f(\boldsymbol{x}; \boldsymbol{W}) = \boldsymbol{W_1 W_0 x}^T. \tag{2.14}$$

Of course, the repeated matrix multiplications in equation (2.14) correspond to a linear transformation, as the matrices $\boldsymbol{W_1}$ and $\boldsymbol{W_0}$ can be equally be represented by a matrix $\boldsymbol{W} = \boldsymbol{W_1 W_0}$. As the objective is to perform non-linear transforms of the input data, an activation function g is generally applied to the output of each matrix multiplication:

$$\hat{y} = f(\boldsymbol{x}; \boldsymbol{W}) = g_1(\boldsymbol{W_1}, g_0(\boldsymbol{W_0 x}^T)). \tag{2.15}$$

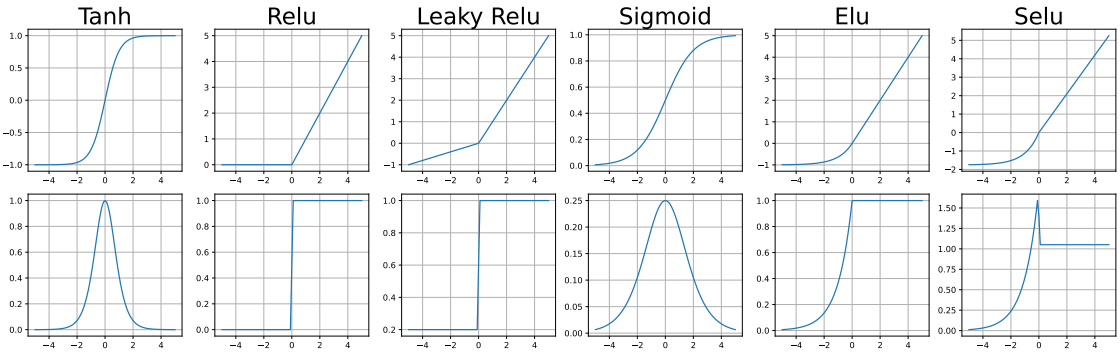


Figure 2.4: **Neural Network activation functions.** Top row: 6 common activation functions used in neural network architecture. Bottom row, the first order derivative of each activation function.

Generally, the same activation function is used throughout the network, with the exception of the last layer. In the case of multi-class classification a softmax function is typically used:

$$\sigma(\boldsymbol{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}. \tag{2.16}$$

In the case of regression an identity function (no activation), however there are special cases where, for example, the aim is to regress a bounded range, that a specific activation can be used.

There are many activation functions available, each with its own distinct properties, advantages and disadvantages. We provide an overview of some popular activation functions and their first-order derivatives in Figure 2.4. For many years, the sigmoid and tanh functions were popular in the field of Deep Learning, however, these suffer from this issue of "vanishing gradients", when the functions begin to saturate the gradient becomes near 0. This effect is compounded as networks become deeper due to the multiplicative aspect of back-propagation 2.3.5. As deeper networks have become more popular, practitioners have moved to the ReLU activation function for two main reasons: simplicity, and that the gradient of a ReLU function is either one or zero. This has led to other challenges such a "dead" ReLUs which output zero for every input in the dataset, there have been efforts to address this (Clevert et al. 2015).

2.3.2 Convolutional Neural Networks

Few would disagree that CNNs have revolutionized the field of computer vision. As researchers approached the problem of learning with images, they looked for solutions that are equivariant to translation. Convolutions have this property, and can be applied as 1D, 2D or 3D filters, or kernels. In this thesis, we apply CNNs

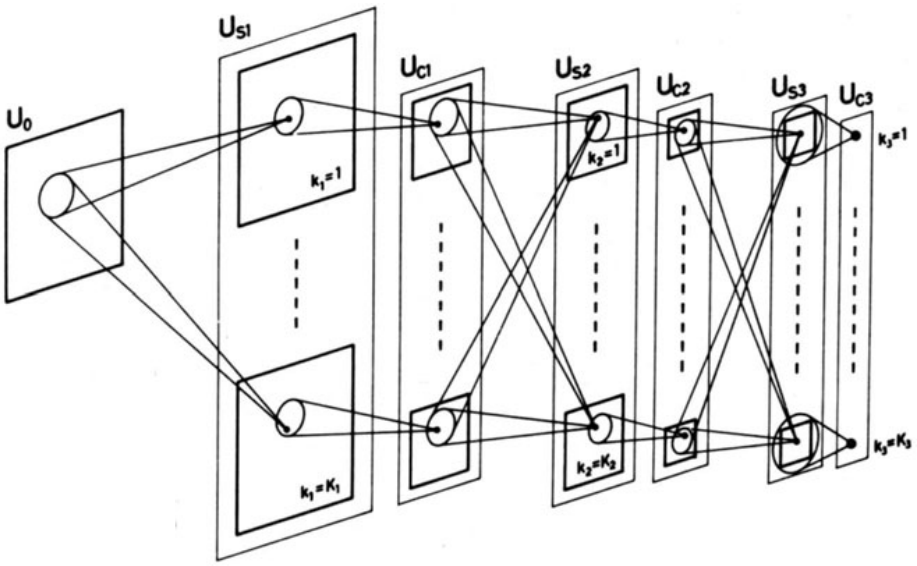


Figure 2.5: **Network architecture proposed by Fukushima** demonstrating how classification can be performed on digits. Figure from (Fukushima et al. 1982)

to 2D images or 2D virtual camera observations, so the examples shown in this section will focus on this domain of application, but are equally applicable to 1D and 3D data. The CNN implementation that inspired so many others is that of AlexNet (Krizhevsky et al. 2012) which won the 2012 ImageNet competition by a large margin. The idea of CNNs dates back to Fukushima, (Fukushima et al. 1982) who designed a feed-forward layer-based network architecture, that uses convolution operations to extract feature representation shown in Figure 2.8. To the authors' knowledge the first learning-based implementation of CNNs was that of LeNet (LeCun et al. 1989), where they were used to perform handwritten digit recognition.

If we consider a convolution filter f applied to a 2D image $\boldsymbol{I}$, where the pixels of the image can be indexed with $I[i, j]$, this filter has a certain height h and width w, typically filters in neural network architectures are small 3×3 and square, relying on repeated layers of convolutions to capture dependencies between far-away pixels. The filter f contains $h \times w$ parameters, which we would like to learn. We show this schematically in Figure 2.6. The mathematical operations required to apply f to $\boldsymbol{I}$ in order to produce an output, $\boldsymbol{G}$ are:

$$G[m, n] = \sum_i \sum_j f[j, k] I[m - j, n - k] \tag{2.17}$$

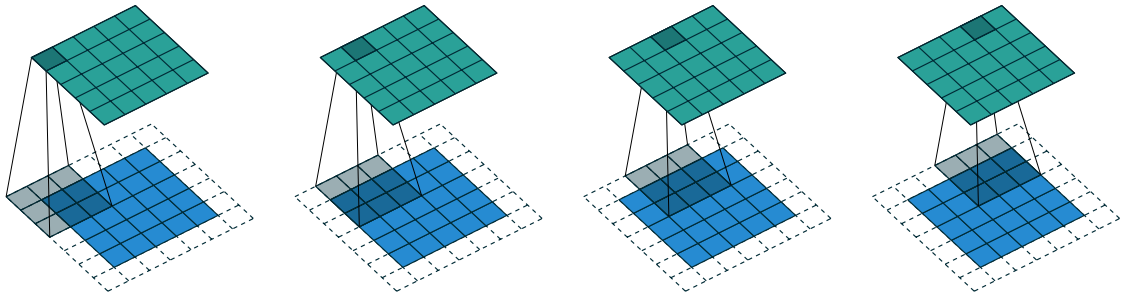


Figure 2.6: **An example convolution** of a 3×3 filter over a 5×5 input, the input is padded in so that the output is of the same size as the input. Figure from (Dumoulin et al. 2018).

The operations of the convolution are spatially equivariant, if the image $\boldsymbol{I}$ contains the same features in the top-left and bottom-right corner, the outputs in these regions will be the same.

Convolutions are stacked in many layers in order to extract more high-level features. In early layers filters respond to low level details such as edges, as we move to deeper layers the filters learn to detect more complex features, see Figure 2.7 which shows a visualization from (Zeiler et al. 2014). In addition to convolutions, the AlexNet, LeNet and many other CNN architectures interleave convolutions with the Max-Pooling operation, which down-samples feature-maps by selecting the highest activation within a small, typically 2×2 region. Although in recent years the Max-Pooling operation has begun to fall out of favor, with the preferred methods of down-sampling being strided convolutions.

There have been many extensions, refinements and improvements to the AlexNet architecture. Here we discuss a few notable mentions, but make no attempt to create an exhaustive list. The VGG architecture (Simonyan et al. 2014) won the 2014 ImageNet challenge by increasing the depth to 19 weight layers and reducing the convolutional filter size to 3×3. The popular ResNet architecture (He et al. 2016) created a CNN with 152 layers which came first in the 2015 ImageNet challenge, they achieved such a depth by using "residual connections" which allow for shortcuts in the computation graph when performing back-propagation, alleviating the issue of vanishing gradients, these networks have some similarities with Highway Networks (Srivastava et al. 2015). While many architectures looked to improve performance with depth and additional parameters, the SqueezeNet architecture (Iandola et al. 2016) achieved AlexNet (Krizhevsky et al. 2012) performance with $50\times$ fewer parameters. DenseNets (Huang et al. 2017) explore the approach of neural networks with highly interconnected layers, where each layer in the network is connected to all of its proceeding layers. ResNeXt (Xie et al. 2017) performs aggregated residual transforms, where similar sets of transforma-

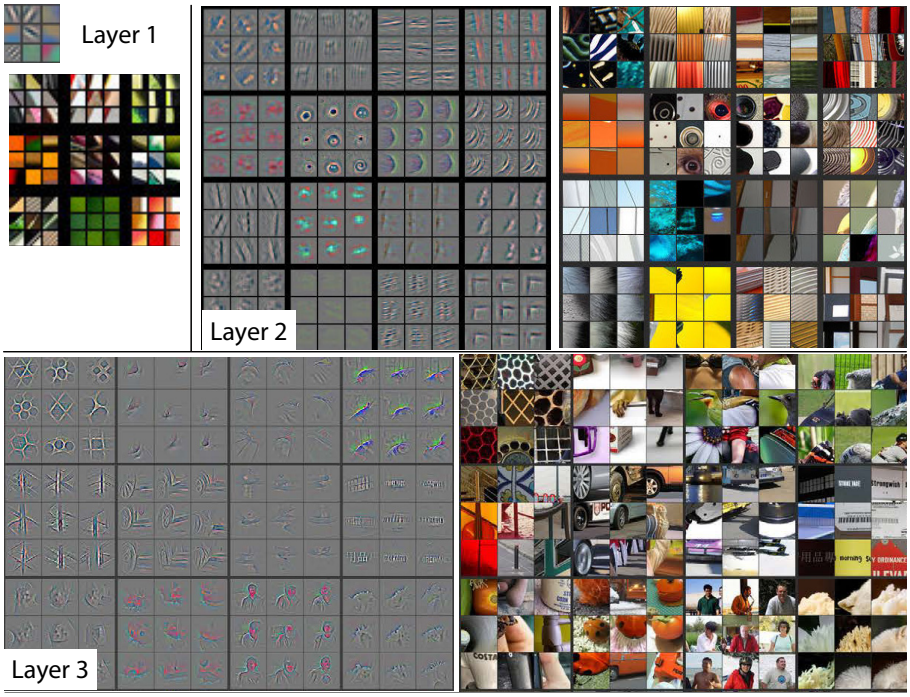


Figure 2.7: **CNN filter visualization** for a fully trained model. Shown are the top 9 activations on a random subset of feature maps. The reconstructions are not samples from the model, they are reconstructions of the features from the validation data that cause high activations in the model. Figure from (Zeiler et al. 2014).

tions are grouped to simplify the network architecture and reduce the number of hyper-parameters to evaluate. As CNN architectures have become ever more present on mobile devices, smaller and more efficient networks have become more desirable. MobileNet (Howard et al. 2019) and EfficientNet (Tan et al. 2020) are network architectures that are equally as performant while being smaller in size and having faster inference times.

2.3.3 Processing sequences

The need to process sequences of data arises in many fields, such as time-series data from sensors or sequences of words in Natural Language Processing (NLP). In this subsection, we discuss Deep Neural Network architectures for the analysis and extraction of information from sequences. This is particularly relevant in the field of sequential decision making in robotics, where we would like an agent to learn to remember key information it observed in the past and store it in an internal representation or "hidden-state". Sequences of are typically represented as ordering of vectors $[\boldsymbol{x}_0, \boldsymbol{x}_1, ..., \boldsymbol{x}_t]$. When building predictive models that work

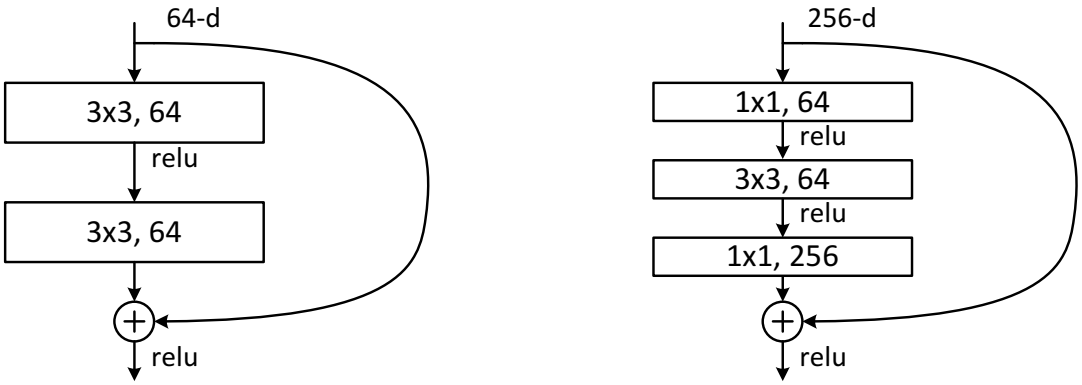


Figure 2.8: **Two ResNet blocks** from the ResNet (He et al. 2016) architecture. Left: a typical ResNet building block, right: a "bottle" block, used in deeper ResNet architectures. Figure from (He et al. 2016).

on sequences, there are many combinations of input and output, depending on the desired application, some examples are shown in Figure 2.13.

2.3.3.1 Recurrent Neural Networks

We will introduce three common recurrent network architectures that are used in the processing of sequences in Deep Learning architectures. Recurrent models condition their output on the current input $\boldsymbol{x_t}$ and an internal hidden state vector $\boldsymbol{h_{t-1}}$, and output the hidden state for the next time-step $\boldsymbol{h_t}$.

The classic vanilla RNN (Elman 1990) is a linear transformation of the inputs $\boldsymbol{x_t}$ and $\boldsymbol{h_{t-1}}$ with the addition of a bias term. In order to bound the outputs to a range of -1.0 to 1.0, a *tanh* non-linearity is applied to the output.

$$h_t = \tanh\left(W_{ih}x_t + b_{ih} + W_{hh}h_{t-1} + b_{hh}\right) \tag{2.18}$$

While the vanilla RNN is a powerful sequence processing tool, it comes with some downsides. It struggles with longer-term dependencies, see Figure 2.10. This phenomenon was studied in some detail by (Hochreiter et al. 2001; Bengio et al. 1994) who identified some of its deficiencies, which lead to the development of the more widely used RNN architecture, Long-Short Term Memory (LSTM) (Hochreiter et al. 1997a).

2.3.3.2 Gated RNN Variants

The LSTM architecture was designed to address the weaknesses of standard RNN. It achieves this through the use of *gating mechanisms* and two internal states, a *cell-state* $\boldsymbol{c_t}$ and a *hidden state* $\boldsymbol{h_t}$. The gating mechanisms in the LSTM architecture

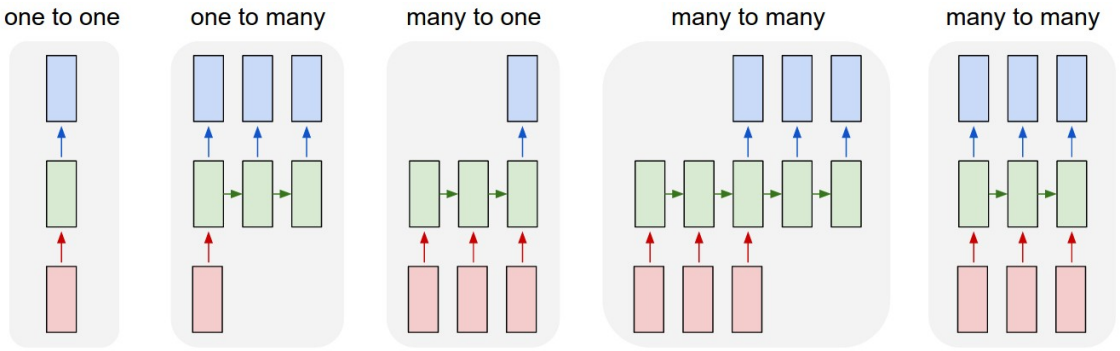


Figure 2.9: **Input/Output configurations of sequences.** Each rectangle is a vector and arrows represent functions (e.g. matrix multiply). Input vectors are in red, output vectors are in blue and green vectors hold the RNN's state. From left to right: (1) Vanilla mode of processing without RNN, from fixed-sized input to fixed-sized output (e.g. image classification). (2) Sequence output (e.g. image captioning takes an image and outputs a sentence of words). (3) Sequence input (e.g. sentiment analysis where a given sentence is classified as expressing positive or negative sentiment). (4) Sequence input and sequence output (e.g. Machine Translation: an RNN reads a sentence in English and then outputs a sentence in French). (5) Synced sequence input and output (e.g. video classification where we wish to label each frame of the video). Notice that in every case are no pre-specified constraints on the lengths sequences because the recurrent transformation (green) is fixed and can be applied as many times as we like. Figure and text from (Karpathy 2015)

control the flow of information from one time-step to the next and consists of $\boldsymbol{i_t}$ the input gate, $\boldsymbol{f_t}$ the forget gate, $\boldsymbol{g_t}$ the cell gate and $\boldsymbol{i_t}$ the output gate. The equations for the operation of an LSTM are shown in (2.19), where $\odot$ is the Hadamard product.

$$
\begin{aligned}
i_t &= \sigma\left(W_{ii}x_t + b_{ii} + W_{hi}h_{t-1} + b_{hi}\right) \\
f_t &= \sigma\left(W_{if}x_t + b_{if} + W_{hf}h_{t-1} + b_{hf}\right) \\
g_t &= \tanh\left(W_{ig}x_t + b_{ig} + W_{hg}h_{t-1} + b_{hg}\right) \\
o_t &= \sigma\left(W_{io}x_t + b_{io} + W_{ho}h_{t-1} + b_{ho}\right) \\
c_t &= f_t \odot c_{t-1} + i_t \odot g_t \\
h_t &= o_t \odot \tanh\left(c_t\right)
\end{aligned}
\tag{2.19}
$$

Since the invention of the LSTM module, there have been many attempts to improve upon it, but little success. One architecture has found its place as a competitor, the Gated Recurrent Unit (GRU) (Chung et al. 2014). GRUs have comparable performance to LSTM but with a simpler architecture and a single

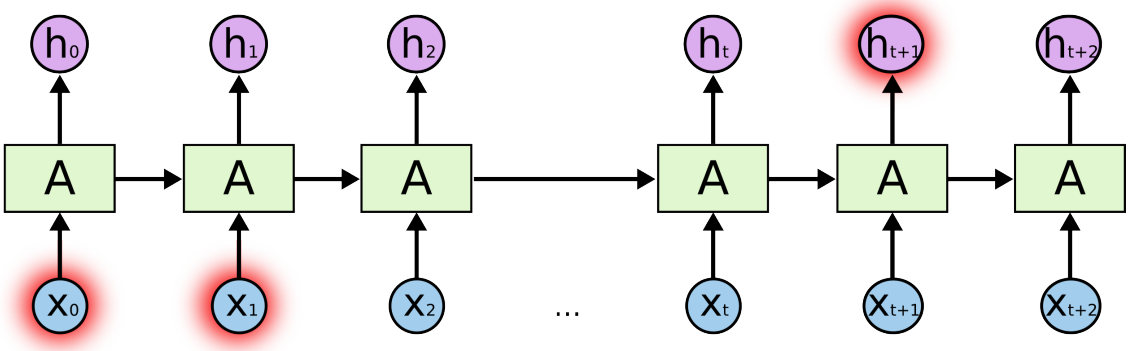


Figure 2.10: **Long term dependencies in a sequence processing problem** the output $\boldsymbol{h_{t+1}}$ depends on the inputs $\boldsymbol{x_0}$ and $\boldsymbol{x_1}$, vanilla RNNs tend to struggle when they are required to store information of many time-steps. Figure from (Olah 2015).

hidden state vector. GRUs also include gating mechanisms $\boldsymbol{r_t}$ the reset gate, $\boldsymbol{z_t}$ the update gate and $\boldsymbol{n_t}$ the new gate. The equations for the operation of an GRU are shown in (2.20).

$$\begin{aligned} r_t &= \sigma\left(W_{ir}x_t + b_{ir} + W_{hr}h_{t-1} + b_{hr}\right) \\ z_t &= \sigma\left(W_{iz}x_t + b_{iz} + W_{hz}h_{t-1} + b_{hz}\right) \\ n_t &= \tanh\left(W_{in}x_t + b_{in} + r_t * \left(W_{hn}h_{t-1} + b_{hn}\right)\right) \\ h_t &= (1 - z_t) * n_t + z_t * h_{t-1} \end{aligned} \tag{2.20}$$

Bidirectional variants of the aforementioned RNN architectures are also available and have been shown to perform well on sequences, such as video classification (Baccouche et al. 2011). Bidirectional RNN are not applicable to the field of sequential decision making so will not be discussed further.

2.3.3.3 Attention-based Networks

As humans we read and parse text sequentially, consuming one word token at a time, however, we often refer back or *attend* to earlier parts of a text in order to comprehend the overall meaning of a sentence or paragraph. An example of this when humans perform translation, consider the French sentence “L’accord sur la zone économique européenne a été signé en août 1992” which translates to “The agreement on the European Economic Area was signed in August 1992". Although this is almost a one-to-one translation, the ordering of parts of the text such as “European Economic Area" are reversed. RNNs and their variants can find these dependencies challenging, as we rely on the capacity of a 1D vector to store this information of many time-steps, this can be particularly challenging with

languages that structure their sentences in a very different order, such as Turkish or Russian.

Recent works on attention-based Deep Learning methods have led to great leaps in performance in NLP, with (Vaswani et al. 2017b) demonstrating the power of this architecture and self-supervised training of the well known BERT (Devlin et al. 2018) model outperforms many other approaches on eleven NLP benchmarks.

The general principle is as follows, given a sequence of inputs $[\boldsymbol{x_0}, \boldsymbol{x_1}, ..., \boldsymbol{x_t}]$ the attention operation performs three linear transforms in order to produce a sequence of *keys* $[\boldsymbol{k_0}, \boldsymbol{k_1}, ..., \boldsymbol{k_t}] = \boldsymbol{K}$, *values* $[\boldsymbol{v_0}, \boldsymbol{v_1}, ..., \boldsymbol{v_t}] = \boldsymbol{V}$ and *queries* $[\boldsymbol{q_0}, \boldsymbol{q_1}, ..., \boldsymbol{q_t}] = \boldsymbol{Q}$. By performing a dot-product comparison of similarity between the queries and keys, the network can learn to attend to different semantic concepts and features in the input sequence:

$$\text{Attention}(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) = \text{softmax}\left(\frac{\boldsymbol{Q}\boldsymbol{K}^T}{\sqrt{d_k}}\right)\boldsymbol{V} \tag{2.21}$$

Where the normalizing factor d_k is the size of the key vector. The attention distribution is calculated between the key and query pairs, but the output is a weighted average of the value vectors, allowing the network query and pass on information that is different to those is used to calculate the attention score.

By stacking and repeating this operation, shown in Figure 2.11, the network can learn to reason about complex interrelated concepts. For more detailed explanation of this method we refer to reader to The Annotated Transformer (Vaswani et al. 2017c), created by the author of (Vaswani et al. 2017b). Since their first release in 2017 there have been many improvements and applications of the technology across various domains, the Transformers repository (Wolf et al. 2019a) provides both an Application Programming Interface (API) and pre-trained models for NLP tasks.

2.3.3.4 Structured memory architectures

The principle of attention mechanisms has also been extended to augmented RNNs, the Neural Turing Machine (Graves et al. 2014) is an attention augmented recurrent neural network that contains a memory that can be addressed through attention. There have been several extensions to this architecture including the Differentiable Neural Computer (Graves et al. 2016) and (Rae et al. 2016). There are implementations of RNNs that attempt to compensate for the loss of information through time and vanishing gradients by processing information with a temporal hierarchy, (Koutnik et al. 2014; Neverova et al. 2016).

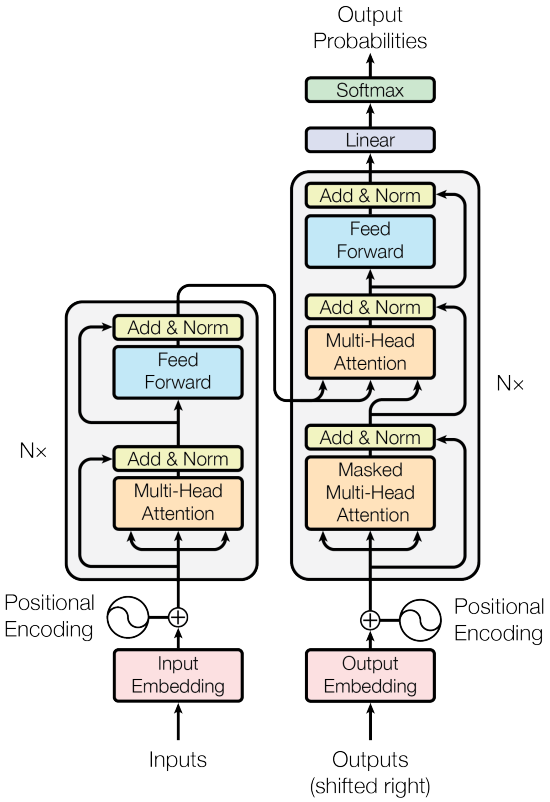


Figure 2.11: **The transformer architecture** applied to a self-supervised word classification task. Figure from (Vaswani et al. 2017b).

2.3.4 Graph Neural Networks

While fully connected networks process order-independent vectors on input, CNNs work with images with spatial invariance and RNNs for sequential reasoning with time invariance. GNNs operate on networks of nodes and edges and aim to be invariant to node and edge permutations.

Graph neural networks have gained popularity in recent years, but have a long history (Battaglia et al. 2018a). GNNs have been applied in many domains including for supervised learning, unsupervised learnin and RL. GNNs operate well on tasks that have a relational structure between nodes and have shown to be effective on Visual Question Answering (VQA) tasks (Raposo et al. 2017), learning the dynamics of physical systems (Chang et al. 2016; Baradel et al. 2020), predicting the properties of molecular chemicals (Duvenaud et al. 2015), translation systems (Gulcehre et al. 2018) and Deep RL (Hamrick et al. 2018).

This section uses the notation of (Battaglia et al. 2018a) for describing the operations in graph neural networks, refer to their work for a detailed overview. Formally a graph is defined as a tuple $G = (V, E)$, where $V = \{\boldsymbol{v_i}\}_{i=1,N^v}$, each $\boldsymbol{v_i}$

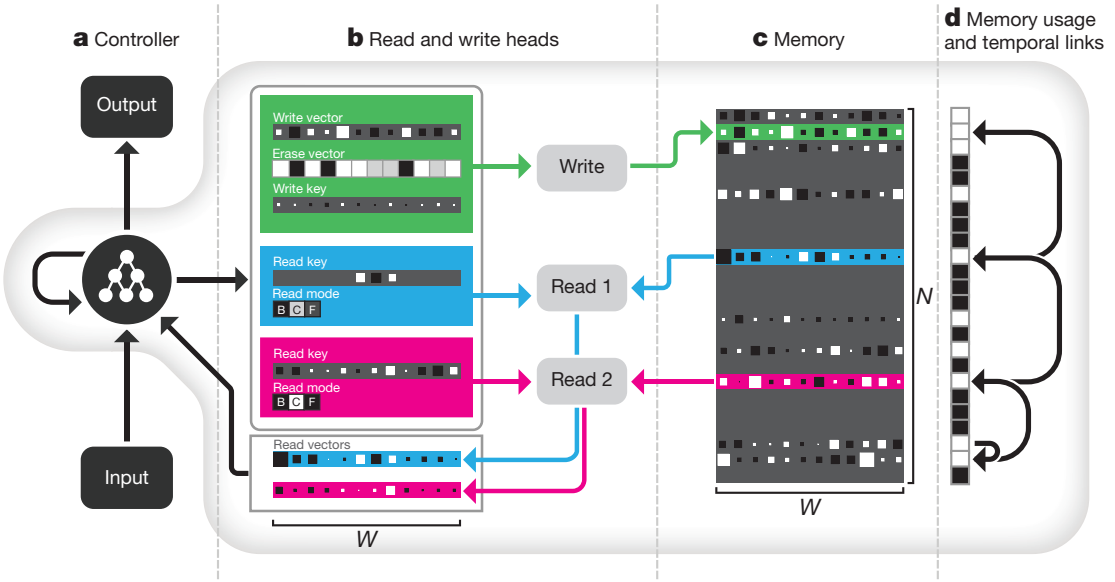


Figure 2.12: **The Differentiable Neural Computer** that combines a LSTM with a 2D memory matrix which can be read and written to with an attention mechanism. Figure from (Graves et al. 2016)

contains node attributes. An example could be a social network where V might be each user, with attributes such as their hobbies, age and gender. The edges $E = \{(\boldsymbol{e_k}, r_k, s_k)\}_{k=1,N^e}$ is the set of edges in the graph where each $\boldsymbol{e_k}$ contains edge attributes, r_k is the receiver node and s_k the sender node. In the social network example E might represent connections between friends, family and colleagues where the edge attributes could hold when the connection was made, the relation "father", "employer" or "employee". GNNs iteratively update their internal representations through message passing and aggregation of information. Typically this is conducted by three update functions, ϕ, and three aggregation functions, ρ:

$$\begin{aligned} \mathbf{e}'_k &= \phi^e\left(\mathbf{e}_k, \mathbf{v}_{r_k}, \mathbf{v}_{s_k}, \mathbf{u}\right) & \bar{\mathbf{e}}'_i &= \rho^{e\to v}\left(E'_i\right) \\ \mathbf{v}'_i &= \phi^v\left(\bar{\mathbf{e}}'_i, \mathbf{v}_i, \mathbf{u}\right) & \bar{\mathbf{e}}' &= \rho^{e\to u}\left(E'\right) \\ \mathbf{u}' &= \phi^u\left(\bar{\mathbf{e}}', \bar{\mathbf{v}}', \mathbf{u}\right) & \bar{\mathbf{v}}' &= \rho^{v\to u}\left(V'\right) \end{aligned} \tag{2.22}$$

Where:

$$\begin{aligned} E'_i &= \left\{\left(\mathbf{e}'_k, r_k, s_k\right)\right\}_{r_k=i,k=1:N^e} \\ V' &= \left\{\mathbf{v}'_i\right\}_{i=1:N^v} \\ \text{and } E' &= \left\{\left(\mathbf{e}'_k r_k, s_k\right)\right\}_{k=1:N^e} \end{aligned} \tag{2.23}$$

These operations are combined into a iterative update algorithm, 2.2.

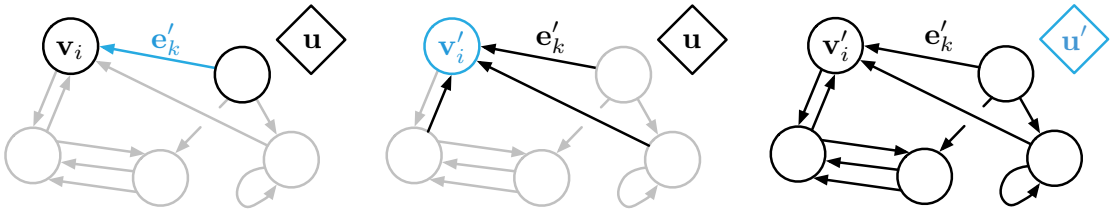


Figure 2.13: **Graph Neural Network operations** the elements colored black are involved in the update with the blue element being updated Figure from (Battaglia et al. 2018a)

Algorithm 2.2 Update operations of a GNN

for $k \in \{1...N^e\}$ **do**
  $\mathbf{e}'_k \leftarrow \phi^e\left(\mathbf{e}_k, \mathbf{v}_{r_k}, \mathbf{v}_{s_k}, \mathbf{u}\right)$
end for
for $i \in \{1...N^n\}$ **do**
  let $E'_i = \left\{\left(\mathbf{e}'_k, r_k, s_k\right)\right\}_{r_k=i,k=1:N^e}$
  $\overline{\mathbf{e}}'_i \leftarrow \rho^{e\rightarrow v}\left(E'_i\right)$
  $\mathbf{v}'_i \leftarrow \phi^v\left(\overline{\mathbf{e}}'_i, \mathbf{v}_i, \mathbf{u}\right)$
end for
let $V' = \{\mathbf{v}'\}_{i=1:N^v}$
let $E' = \left\{\left(\mathbf{e}'_k, r_k, s_k\right)\right\}_{k=1:N^e}$
$\overline{\mathbf{e}}' \leftarrow \rho^{e\rightarrow u}\left(E'\right)$
$\overline{\mathbf{v}}' \leftarrow \rho^{v\rightarrow u}\left(V'\right)$
$\mathbf{u}' \leftarrow \phi^u\left(\overline{\mathbf{e}}', \overline{\mathbf{v}}', \mathbf{u}\right)$
return $\left(E', V', \boldsymbol{u}'\right)$

2.3.5 Optimization and back-propagation

Now that the key building blocks of neural network architectures have been defined, it is important to understand how to learn the parameters of a predictive model. In order to make iterative updates to the model parameters we need to calculate gradients, that is for each parameter in the network we must calculate how changing that parameter will affect the model's predictions for a batch of data examples.

This involves several key aspects. The first is that of automatic differentiation, for every fundamental operation applied in a neural network, the analytic first-order derivative is known and defined, see examples in equations (2.24) and (2.25).

$$\begin{aligned} \sigma(x) &= \frac{1}{1+e^{-x}}, \\ \sigma'(x) = \frac{\partial \sigma(x)}{\partial x} &= \sigma(x)(1-\sigma(x)). \end{aligned} \tag{2.24}$$

$$\begin{aligned} \mathrm{ReLU}(x) &= \max(0, x), \\ \mathrm{ReLU}'(x) &= \begin{cases} 0 & \text{if } x \leq 0 \\ 1 & \text{if } x > 0 \end{cases} \end{aligned} \tag{2.25}$$

Some functions are non-continuous such as the ReLU operation at $x = 0$, although in practical terms this does not cause any real issues. For a more detailed study of automatic differentiation in machine learning (Baydin et al. 2018) provides a thorough overview. Each step of computation is stored in a directed graph, where during the forward pass through the network the sequence of operations and their intermediate outputs are stored. This is best demonstrated with an example, consider the MLP from section 2.3.1, preforming a linear regression task on a dataset of vectors $\boldsymbol{X}$ and their corresponding labels y.

$$\hat{y} = f(\boldsymbol{x}; \boldsymbol{W}) = \boldsymbol{W_1} \tanh(\boldsymbol{W_0 x^T}) \tag{2.26}$$

During the forward pass, the outputs of each intermediate operation, or layer activations must be stored. Consider the forward pass of an input example x, detailed in equation (2.27).

$$\begin{aligned} \boldsymbol{a} &= \boldsymbol{W_0 x^T}, \\ \boldsymbol{o} &= \tanh \boldsymbol{a}, \\ \hat{y} &= \boldsymbol{W_1 o} \end{aligned} \tag{2.27}$$

We then compute the loss, in this example we will use a mean squared error loss function $\mathcal{L}(\hat{y}, y) = (\hat{y} - y)^2$. We then wish to calculate the partial derivatives of this loss with respect to each parameter in the neural network, for example $\frac{\partial L}{\partial \boldsymbol{W_0}}$. This is achieved for performing repeated applications of the the chain rule $\frac{\partial x}{\partial z} = \frac{\partial x}{\partial y}\frac{\partial y}{\partial z}$. Performing the backwards pass over the previous operations results in the following partial derivatives:

$$\frac{\partial \mathcal{L}}{\partial \hat{y}} = 2(\hat{y} - y) \tag{2.28}$$

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{W_1}} = \frac{\partial \mathcal{L}}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial \boldsymbol{W_1}} = 2(\hat{y} - y)\boldsymbol{o} \tag{2.29}$$

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{o}} = \frac{\partial \mathcal{L}}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial \boldsymbol{o}} = 2(\hat{y} - y)\boldsymbol{W_1} \tag{2.30}$$

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{W_0}} = \frac{\partial \mathcal{L}}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial \boldsymbol{o}} \frac{\partial \boldsymbol{o}}{\partial \boldsymbol{a}} \frac{\partial \boldsymbol{a}}{\partial \boldsymbol{W_0}} = 2(\hat{y} - y)\boldsymbol{o}(1 - \tanh(\boldsymbol{a})^2)\boldsymbol{x}^T \tag{2.31}$$

The operations and derivatives above are the analytic derivatives for the particular network architecture. In modern deep learning frameworks the derivatives are computed automatically, intermediate partial derivatives are stored in order to reduce the computational complexity of repeated calculations.

Once the partial derivatives have been calculated, the next step is to perform SGD, the act of stepping in the direction of the gradient of the loss function, with the objective of minimizing it. Practically speaking, SGD is not performed on individual samples but on small subsets, batches, of the training examples. This is for two reasons, to have a less noisy estimate of the true gradient and because larger networks perform the optimization step of one or more Graphics Processing Unit (GPU)s and parallelize the forward and backward passes across the many samples in the batch.

Typically SGD is performed with an optimization algorithm, we will focus here on 1st order optimizers rather than second order-methods such as Conjugate gradient (Hestenes et al. 1952), BFGS, Levenberg–Marquardt (Moré 1978) and Gauss-Newton, as they do no scale well to Deep Networks that contain millions of parameters.

As Deep Learning research has expanded in the last decade, so have the number of optimizer available to the community. Example include SGD, Adam (Kingma et al. 2017), RMSprop (Hinton et al. 2012a), Adadelta (Zeiler 2012), AdamW (Loshchilov et al. 2019), NAdam (Dozat 2016), RAdam (Liu et al. 2021), to name a few.

We explain here SGD and Adam, as these optimizers were used frequently during the thesis. “Vanilla" SGD performs an update to the networks parameters W.

$$\boldsymbol{W} \leftarrow \boldsymbol{W} + \eta \frac{\partial \mathcal{L}}{\partial \boldsymbol{W}} \tag{2.32}$$

Where η is a learning rate parameter, which controls the step size; too small and little progress is made leading to slow convergence, too large and the optimizer is likely to fluctuate over or diverge from a (local) optimum.

Momentum (Qian 1999) can dampen oscillations in the direction of the gradient and accelerate SGD, this is achieved by computing a running average of the gradients of from the previous batches and using this to update the parameters of the network:

$$\begin{aligned} \boldsymbol{v_t} &= \gamma \boldsymbol{v_{t-1}} + \eta \frac{\partial \mathcal{L}}{\partial \boldsymbol{W}} \\ \boldsymbol{W} &\leftarrow \boldsymbol{W} - \boldsymbol{v_t} \end{aligned} \tag{2.33}$$

Where v_t is a running average of the gradients, η is the learning rate and γ controls the contribution gradients from previous updates. A further extension to classic momentum is that of Nesterov momentum (Nesterov 1983). Both RMSprop and

Adam aim to perform an adaptive learning rate control for each model parameter independently:

$$\begin{aligned} \boldsymbol{v_t} &= \gamma \boldsymbol{v_{t-1}} + (1-\gamma)\left(\frac{\partial \mathcal{L}}{\partial \boldsymbol{W}}\right)^2 \\ \boldsymbol{W} &\leftarrow \boldsymbol{W} - \frac{\eta}{\sqrt{\boldsymbol{v_t}+\epsilon}}\frac{\partial \mathcal{L}}{\partial \boldsymbol{W}} \end{aligned} \tag{2.34}$$

Where $\boldsymbol{v_t}$ is a running average of the square of gradients, η is the learning rate, γ (typically 0.9) controls the contribution gradients from previous updates, and *epsilon* (typically 1e-8) is used to stabilize the devision operation. Note that although Adam has quickly become the defacto choice of optimizer in Deep Learning problems, there are issues related to its convergence guarantees, it has been demonstrated to fail to converge on some toy problems (Reddi et al. 2018). For PyTorch users, enabling the "amsgrad" option will use the corrected algorithm. For a deeper dive into optimization algorithms for Deep Neural Networks we refer the reader to (Ruder 2017).

2.4 Supervised Deep Learning

While the previous section has discussed the building blocks that are used in Deep Learning, this section introduces some *applications* and large-scale architectures that are used in both research and industrial settings.

The field of Deep Learning is vast and it would be impossible to cover all works in the field. Deep Learning approaches have been applied to a range of problems, with the most common application is that of image recognition where is have been applied to handwritten digit recognition (Lecun et al. 1998) and the ImageNet classification task (Krizhevsky et al. 2012; He et al. 2015; Szegedy et al. 2016; Liu et al. 2018; Pham et al. 2021; Dai et al. 2021) discussed in section 2.3.2. The second application of that of semantic segmentation, where a CNN performs pixel-wise classification, an example is shown in Figure 2.14. Semantic segmentation has applications in labeling of photos, autonomous vehicles (Chen et al. 2018b; Chen et al. 2018a; Liu et al. 2019; Choi et al. 2020), medical image diagnostics (Novikov et al. 2018; Jha et al. 2020; Zhou et al. 2019; Zhou et al. 2020). Applications of semantic segmentation also raise some ethical concerns, in particular when these technologies are used for crowd monitoring and surveillance (Sreenu et al. 2019), causing the author of the well known "You Only Look Once" (Redmon et al. 2018) network architecture to discontinue his research in this direction.

Beyond image-based applications, Deep Learning has changed the face of NLP for Machine Translation (Devlin et al. 2018; Liu et al. 2020) , sentiment analysis (Thongtan et al. 2019; Sun et al. 2019; Howard et al. 2018) and question answering (Raffel et al. 2019; Garg et al. 2020; Wu et al. 2020). There are many *multi-modal* applications of Deep Learning, such as VQA (Yang et al. 2019; Zaheer et al. 2020;

Figure 2.14: **An example of semantic segmentation** performing pixelwise classification of two classes “cat" and “dog". Figure from (Chen et al. 2018b)

Garg et al. 2020), image captioning; where the network must output a string of text that describes an input image (Gu et al. 2017). Deep Learning has also been applied to Speech Recognition (Park et al. 2020; Ravanelli et al. 2018), Speech Synthesis (Shen et al. 2018) and Dialogue Generation (Bahdanau et al. 2014; Wolf et al. 2019b).

2.5 Deep Reinforcement Learning

This section provides an overview the required knowledge related to the Deep RL algorithms that were implemented and applied during this thesis. As the environment’s state space becomes ever larger, we are faced with two challenges: the number of finite states exceeds the available memory and there are so many states it would take infinitely long to explore them all. Ideally in this instance, we would expect that for two similar states, the actions selected by our would be similar. We require a generalization between similar states. Non-linear function approximation, with a Neural Network (NN), resolves the challenge of generalization between similar states, removes the requirement to hold a lookup table in memory for each state and allows us to model complex function mappings between the input states and output actions. The type of NN architecture depends on the problem at hand, in this thesis we deal with states or observations (see POMDP section 2.5.1) that are typically a multi-modal combination of image and vector data. Network architectures in Deep Reinforcement Learning are typically far smaller than those applied in supervised learning, this is due to a noisy, high variance estimates of the “true” gradient during optimization process, discussed further in 2.7. An example network architecture for image-based states from the

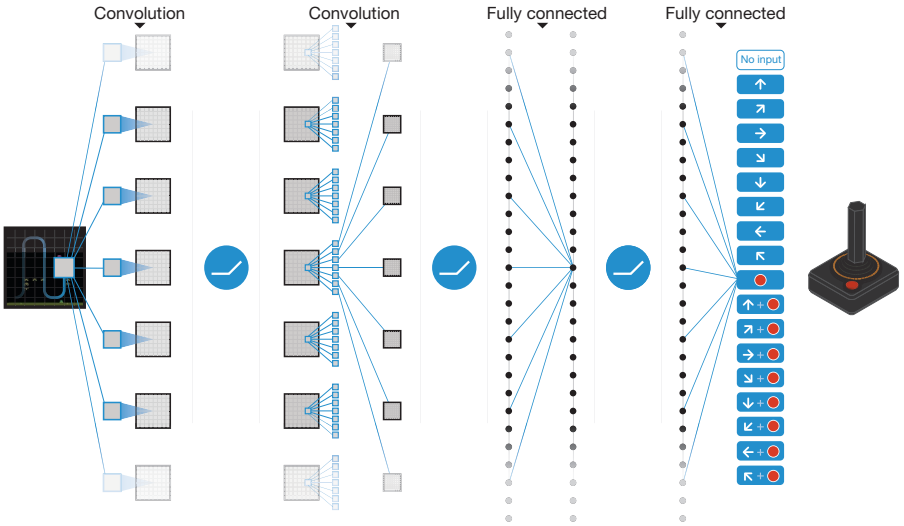


Figure 2.15: **The AtartNet architecture**, with two convolutional layers that process 84x84 gray-scale images. The architecture is relatively small compared to applications of Deep Learning in a supervised setting. Figure from (Mnih et al. 2015c).

Atari-57 benchmark is shown in 2.15, the network architecture is comparable in size to the work of (Lecun et al. 1998) but has been used to learn a policy that plays Atari games (Mnih et al. 2015c).

Certain environments are not considered to be Markov and do not provide the entire state to the agent at each time-step. The majority of environments that were used during this thesis have such a property and are considered to be POMDP. Note, in the section, we refer to the parameters of a neural network as θ rather than $\boldsymbol{W}$ to remain consistent with the Deep RL literature.

2.5.1 Partially Observable Markov Decision Processes

The POMDP (Åström 1965) is an extension of an MDP in a partially observable domain, where the agent, instead of receiving the state $\boldsymbol{s}$ to perform decision making, is provided by environment with an observation $\boldsymbol{o}$. There is a function unknown to the agent $\boldsymbol{o} = O(\boldsymbol{s})$, which maps the states to the observations.

The formal definition of a POMDP a sequential decision making problem, is defined as a tuple $\langle \boldsymbol{\mathcal{S}}, \boldsymbol{\mathcal{A}}, \boldsymbol{\mathcal{P}}, \boldsymbol{\mathcal{R}}, \boldsymbol{\mathcal{O}}, \gamma \rangle$ where:

- $\boldsymbol{\mathcal{S}}$: a finite set of states
- $\boldsymbol{\mathcal{A}}$: a finite set of actions

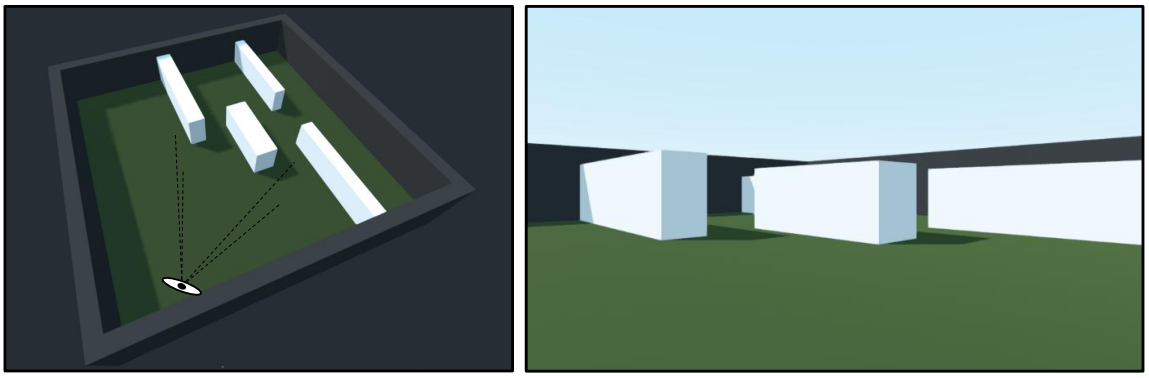


Figure 2.16: **Perspective projection 3D to a 2D plan projection**. Left: an example 3D scene with a monocular camera placed at the bottom right corner. Right: a planar projection of the 3D scene geometry onto a 2D virtual monocular camera. Figure created the Godot Game Engine (Linietsky et al. 2014).

- $\mathcal{P}$: a state transition probability matrix, $P(s'|s,a)$
- $\mathcal{R}$: a reward function
- Ω: a set of observations
- $\mathcal{O}$: a set of conditional observation probabilities
- γ: a discount factor $\in [0,1]$

Because the agent does not directly observe the state of the environment, the optimal decision to take at time-step may require information from previous time-steps. By incorporating information from previous time-steps into its decision making process, the agent can be more confident about its belief about the true state of the environment. As an example, consider an agent in a maze observing the world through a monocular camera, shown Figure in 2.16. In this example we can consider the observation to be the pixels of the image captured by the monocular camera and the state to be the configuration on the environment. The function $O(s)$ is the projection of 3D scene to the 2D camera coordinates. As an agent moves around this scene, it could potentially expand its knowledge about the actual state of the environment. However, in order to do this the agent requires a form of memory to store pertinent information about the environment. Typically Deep RL agents use an RNN such as an LSTM (Hochreiter et al. 1997a) or a GRU (Chung et al. 2014) to learning to store information about the environment from one time-step to the next, these were introduced in section 2.3.3. We demonstrate how an RNN can be incorporated in to a agent's network architecture in Figure 2.18.

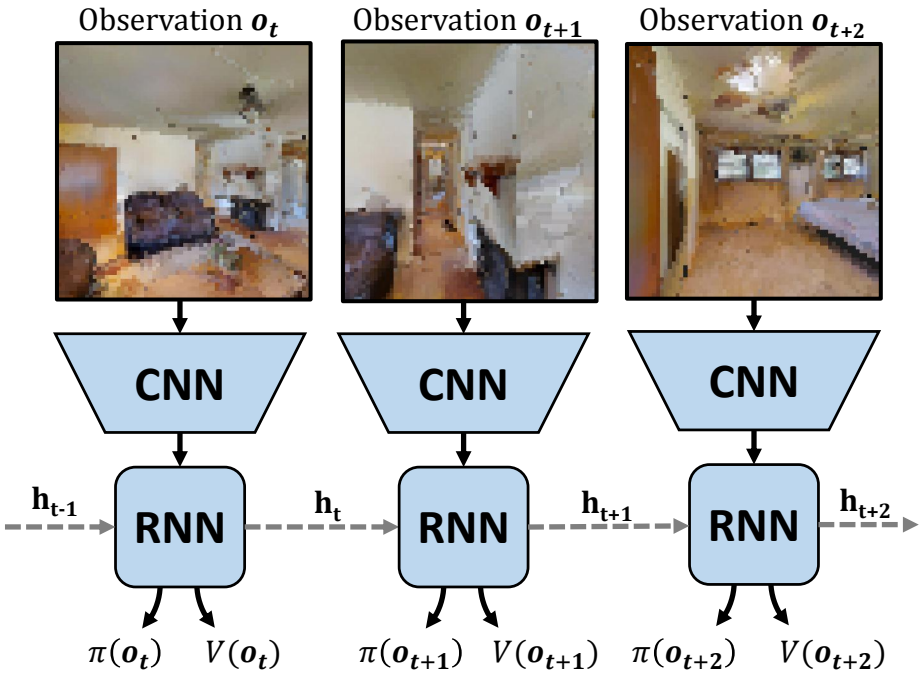


Figure 2.17: **Overview of a recurrent agent architecture**, the agent uses a recurrent neural network to learn to store information about the environment. The output of the RNN is used to parameterize an action distribution $\pi(o_t)$ and value function $V(o_t)$.

2.5.2 Algorithms

We will now introduce a taxonomy of RL algorithms. Broadly, Deep RL algorithms fit into two categories, model-free and model-based. In model-free RL we learn a policy in the absence of a model of the environment. In model-based RL we are either provided with a model of the environment which can be used for planning to aid us in the making of decisions, or we jointly learn a model of the environment and a policy that uses the learned model to improve its decision making process. In this thesis we have focused on model-free algorithms and their applications. Model-free learning algorithms can further be split into two categories, on-policy algorithms and off-policy algorithms, although there are recent advances that combine the two approaches in a judicious way. The key difference between on-policy and off-policy algorithm is that on-policy algorithms require experience that has been generated with the policy that is currently being optimized, whereas off-policy algorithms can use experience from any policy such as, for example, ϵ-greedy exploration. The requirement for on-policy algorithms to make updates with experience from their current policy typically makes them less sample efficient than off-policy algorithms, but comes with other advantages

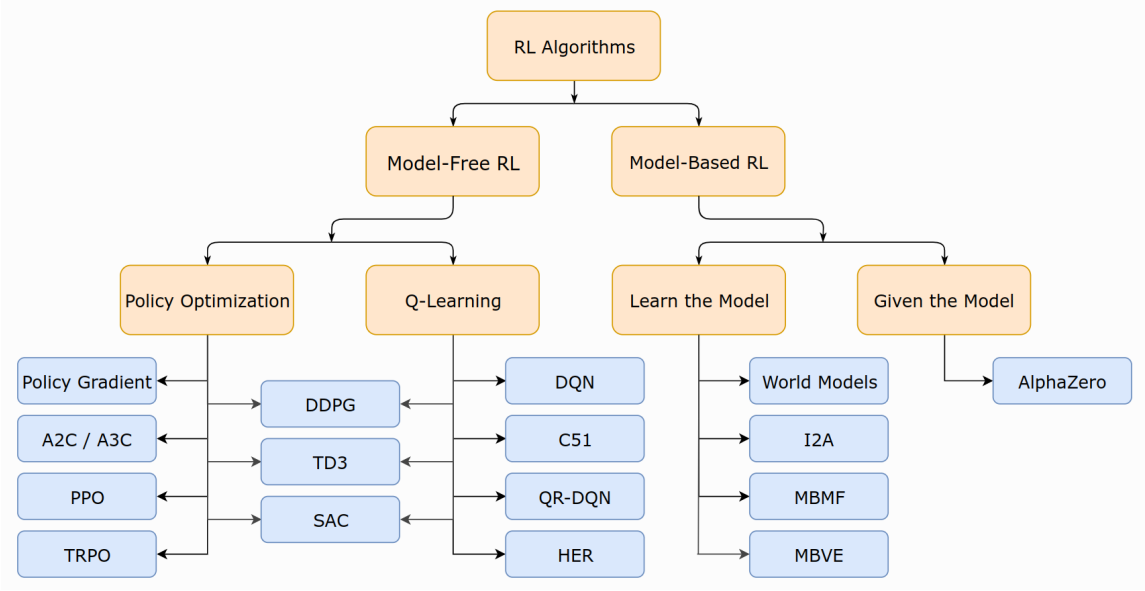


Figure 2.18: **A taxonomy of RL algorithms**, shown are a subset of modern Deep RL algorithms. Figure from (Achiam 2018).

including more algorithmic stability and suitability for the learning of recurrent policies.

Before we explain in detail a number a key algorithms, we will first provide an overview of a few key concepts. Discounted returns G, state-value functions $V(\boldsymbol{s})$ and advantage functions $A(\boldsymbol{s}, a)$. In an on-policy setting discounted returns are calculated independently for each trajectory τ of experience of length T, sampled for a given policy $\pi(\boldsymbol{s})$ in the environment. The discounted return G_t is a sum of all the reward received after a particular time-step t, discounted by how far into the future they were received with a discount factor γ:

$$G_t = \sum_{k=t}^{T} \gamma^{k-t} r(\boldsymbol{s_k}, a_k). \tag{2.35}$$

We can now assign a discount return to each state in a trajectory. In many on-policy methods, one of the objectives is to approximate the discounted return for a given policy. The value function $V^{\pi}(\boldsymbol{s})$ aims to model the expected discounted return for each state, for a given policy $\pi(\boldsymbol{s})$. During the training of an Deep RL agent a finite number of samples is used, instead of computing the expectation explicitly. The advantage function provides us with an estimate of the relative improvement an action gives in particular state when compared the other actions and is as follows:

$$A^{\pi}(\boldsymbol{s}, a) = Q^{\pi}(\boldsymbol{s}, a) - V^{\pi}(\boldsymbol{s}). \tag{2.36}$$

Often in an on-policy setting, we do not approximate both the Q function and the Value function. So the advantage is estimated, trajectory by trajectory, by taking

the difference between the discounted return for a state and the estimated value of that state.

2.5.3 On-policy algorithms

We now describe three on-policy Deep RL algorithms: REINFORCE (Williams 1992b), Advantage Actor Critic (Mnih et al. 2016a) and Proximal Policy Optimization (Schulman et al. 2017a)

2.5.3.1 REINFORCE

The REINFORCE algorithm (Williams 1992b) learns a parameterized policy distribution $\pi(\boldsymbol{s_t})$ and can operate in environments with either continuous or discrete action spaces. The algorithm samples a trajectory from the current policy and then performs an update based on the loss function defined as:

$$\mathcal{L}_{REINFORCE} = \sum_{t=0}^{T} G_t \log \pi \left(a_t \mid \boldsymbol{s_t}\right). \tag{2.37}$$

A downside of the algorithm is that there can be high variance in the sampled trajectories, in the limit the quantity of data should enable to model to converge. However, using non-linear function approximation in an RL setting is notoriously unstable (Henderson et al. 2018). In order to reduce this variance, the authors of REINFORCE recommend using a baseline variable b to aim to normalize the returns:

$$\mathcal{L}_{REINFORCE} = \sum_{t=0}^{T} (G_t - b) \log \pi \left(a_t \mid \boldsymbol{s_t}\right). \tag{2.38}$$

The value of b could be a (running) average of the discounted returns, or in the case of actor-critic methods 2.5.3.2, be an estimate of a states' value.

2.5.3.2 Advantage Actor Critic

The Advantage Actor Critic algorithm (Mnih et al. 2016b) involves the learning of two parametric functions. A policy $\pi(\boldsymbol{s}; \theta)$, the actor, and a value function $V(\boldsymbol{s}; \theta)$, the critic. Like REINFORCE (Williams 1992b) this algorithm is applicable in environments with discrete and continuous action spaces. The algorithm operates on batches of trajectories of experience sampled from the current iteration of the policy. To control the exploration-exploitation trade-off, the algorithm often includes an entropy bonus as part of its loss function. The algorithm performs joint optimization of three objective functions: A policy loss $\mathcal{L}_\pi$ (2.39), a value function loss $\mathcal{L}_V$ (2.40) and an entropy loss $\mathcal{L}_H$ (2.41):

$$\mathcal{L}_\pi = \sum_{\tau\sim\pi}\sum_{t=0}^{T} \log \pi(a_t|\boldsymbol{s_t};\theta) A^\pi(\boldsymbol{s_t}, a_t;\theta). \tag{2.39}$$

$$\mathcal{L}_V = \sum_{\tau\sim\pi}\sum_{t=0}^{T} (V^\pi(\boldsymbol{s_t};\theta) - r_t)^2. \tag{2.40}$$

$$\mathcal{L}_H = -\sum_{\tau\sim\pi}\sum_{t=0}^{T}\sum_{i=1}^{n} \pi(a_i|\boldsymbol{s_t};\theta) \log \pi(a_i|\boldsymbol{s_t};\theta). \tag{2.41}$$

The three losses are combined into one loss function $\mathcal{L}_{actor-critic}$, with two tunable weights λ_V and λ_H:

$$\mathcal{L}_{actor-critic} = \mathcal{L}_\pi + \lambda_V \mathcal{L}_V + \lambda_H \mathcal{L}_H \tag{2.42}$$

The Advantage Actor Critic algorithm is somewhat sample inefficient when compared with its off-policy equivalents, but as it is on-policy it cannot suffer from representational drift, which we discuss in section 2.5.4.3.

2.5.3.3 Proximal Policy Optimization

Proximal Policy Optimization (PPO) (Schulman et al. 2017a) is an on-policy algorithm that extends Advantage Actor Critic (A2C) with the aim of improving the sample efficiency. In A2C the algorithm operates on batches of episodes sampled from parallel agents interacting with parallel RL environments. For each batch the A2C algorithm performs one gradient step before discarding the batch and interacting with the simulator using the updated policy. PPO performs several updates to its policy with the same batch of episodes, improving the sample efficiency. There are, however, several potential downsides to this approach. After the first update of the policy the next batch of data is no longer on-policy, which can cause representation drift and updates that lead to a reduction in performance. To resolve this issue, the PPO algorithm defines a proximal zone, which restricts the size of updates that can be made to the policy. Once a sample of transition is outside of the proximal zone, its contributions to the gradients of the update are clipped to zero. PPO is inspired by a similar algorithm, Trust-Region Policy Optimization (TRPO) (Schulman et al. 2015), but is far simpler to implement and achieves comparable empirical performance. PPO enforces this proximal zone by modifying the actor-critic policy loss defined in (2.39) to the the following:

$$L\left(\boldsymbol{s}, a; \theta_k, \theta\right) = \min\left(\frac{\pi_\theta(a \mid \boldsymbol{s})}{\pi_{\theta_k}(a \mid \boldsymbol{s})} A^{\pi_{\theta_k}}(\boldsymbol{s}, a), \quad \text{clip}\left(\frac{\pi_\theta(a \mid \boldsymbol{s})}{\pi_{\theta_k}(a \mid \boldsymbol{s})}, 1-\epsilon, 1+\epsilon\right) A^{\pi_{\theta_k}(\boldsymbol{s},a)}\right) \tag{2.43}$$

This loss function limits the maximum change in the ratio of probabilities under the current policy and the old policy to be within a certain fraction controlled a parameter ϵ, typically 10-20%.

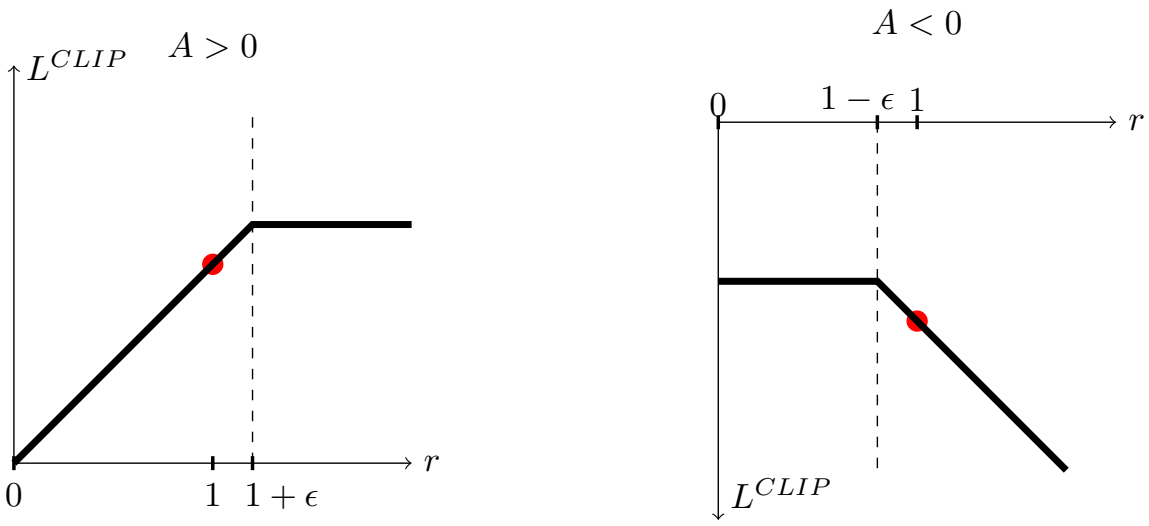


Figure 2.19: **The PPO clipping objective** shown for positive advantages (left) and negative advantages (right). The clipping parameter ϵ restricts the ratio of new to old action probabilities to be within $1-\epsilon$ to $1+\epsilon$. Figure from (Schulman et al. 2017a).

This equation is more clear when it is split into two cases, when the advantage is positive and when the advantage is negative, shown in Figure 2.19. When the advantage is negative, the objective is clipped on transitions where the ratio of action probabilities is below $1-\epsilon$. When it is positive the inverse is true and the objective function is clipped when the ratio it is above $1+\epsilon$. This restricts the updates such that they remain in a zone around the old policy parameters, often the clipping parameter ϵ is decayed to zero during the training in order to avoid large shifts in the policy when it is near to convergence.

2.5.4 Off-policy algorithms

2.5.4.1 Deep Q-Learning

Deep Q-Learning (DQN) (Mnih et al. 2015c), extend the tabular Q-Learning algorithm to settings that require function approximation, typically due a the vast number of possible states the agent can encounter. The standard DQN algorithm applies to settings where the action space of the agent is discrete, the vanilla algorithm is not directly applicable in environments where the action space is continuous. As is the case of tabular Q-Learning, the objective in DQN is learning a function that returns the expected return of a state-action pair, $Q(s, a)$. In this setting, the function $Q(s, a)$ is represented by a Deep NN, which can be an MLP or a CNN depending on the environments state representation. Performing optimiza-

tion of a Deep NN in an on-line setting with an ever changing data distribution poses several challenges, which can lead to catastrophic forgetting and general instability during training. One of the key ways to rectify these issues is that of a large *experience replay buffer*, which typically holds one million state transitions or more. During optimization, transitions are uniformly sampled from the replay buffer, such that a diverse range of experience is used to perform parameter updates to the network. The loss function in the DQN setting is based on the update rule in the tabular setting:.

$$\mathcal{L}(\theta) = \left[r + \gamma \max_{a} Q(\boldsymbol{s_{t+1}}, a; \theta) - Q(\boldsymbol{s_t}, a_t; \theta)\right]^2. \tag{2.44}$$

As with tabular Q-Learning, the exploration strategy is ϵ-greedy, with the probability annealed over time during training. A downside of DQN methods, is that in general they are only applicable in environments with a discrete action space, an issue that is addressed with the Soft Actor-Critic algorithm (Haarnoja et al. 2018a) introduced in section 2.5.4.4. Nevertheless the DQN algorithm was the first Deep RL approach that reached human level performance on the Atari-57 benchmark and convinced researchers of the applicability of Deep RL in domains with vast state spaces.

Algorithm 2.3 Deep Q-learning with Experience Replay

Initialize replay memory $\mathcal{D}$ to capacity N
Initialize action-value function Q with random weights
for episode $= 1, M$ **do**
  Initialize sequence $\boldsymbol{s_1} = \{\boldsymbol{x_1}\}$ and preprocessed sequenced $\boldsymbol{\phi_1} = \phi(\boldsymbol{s_1})$
  for $t = 1, T$ **do**
    With probability ϵ select a random action a_t
    otherwise select $a_t = \max_a Q^*(\phi(\boldsymbol{s_t}), a; \theta)$
    Execute action a_t in the simulator and observe the reward r_t and state observation $\boldsymbol{x_{t+1}}$
    Set $\boldsymbol{s_{t+1}} = \boldsymbol{s_t}, a_t, \boldsymbol{x_{t+1}}$ and preprocess $\boldsymbol{\phi_{t+1}} = \phi(\boldsymbol{s_{t+1}})$
    Store transition $(\boldsymbol{\phi_t}, a_t, r_t, \boldsymbol{\phi_{t+1}})$ in $\mathcal{D}$
    Sample random mini-batch of transitions $(\boldsymbol{\phi_j}, a_j, r_j, \boldsymbol{\phi_{j+1}})$ from $\mathcal{D}$
    Set $y_j = \begin{cases} r_j \text{ for terminal } \boldsymbol{\phi_{j+1}} \\ r_j + \gamma \max_{a'} Q(\boldsymbol{\phi_{j+1}}, a'; \theta) \text{ for non-terminal } \boldsymbol{\phi_{j+1}} \end{cases}$
    Perform a gradient descent step on $(y_j - Q(\boldsymbol{\phi_j}, a_j; \theta))^2$
  end for
end for

2.5.4.2 Extensions: Rainbow

The success of Deep Q-Learning in Atari games has spurred numerous follow-up works, modifications and improvements to the underlying algorithm, shown in 2.3. The Rainbow paper (Hessel et al. 2018a) sought to compare these improvements and show how complementary they are when combined together. The extensions that are compared and ablated are summarized as follows:

Double Q-Learning The standard Q-Learning equation is affected by overestimation bias, where the fact that taking the max of the Q-function (that we are currently estimating) leads to over optimistic estimates of the discounted future return of a particular action. In double Q-learning (Hasselt 2010; Van Hasselt et al. 2016), the max is performed on a separate target network, to decouple selection from evaluation.

Prioritized replay The standard DQN algorithm (Mnih et al. 2015c) samples from the replay buffer uniformly. However, some transitions contain more to learn that others, so we would like to adjust the distribution in order to sample these more frequently. Prioritized experience replay (Schaul et al. 2015) adjusts the probability distribution such that transitions with higher Temporal-Difference (TD) error are sampled with a higher probability, p_t, calculated with:

$$p_t \propto \left| R_{t+1} + \gamma_{t+1} \max_{a'} q_{\bar{\theta}} \left(\boldsymbol{s_{t+1}}, a' \right) - q_\theta \left(\boldsymbol{s_t}, A_t \right) \right|^\omega . \tag{2.45}$$

Where w is a hyper-parameter that controls the shape of the distribution. In order to encourage sampling of new experience, when transitions are first added to the replay buffer they are assigned this maximum priority.

Dueling networks Are a modified NN architecture by (Wang et al. 2016b) with two output heads that perform value and advantage predictions. The authors demonstrate that by factoring out the Q-value estimates into predictions of values and advantages, greater performance can be achieves compared to the standard DQN architecture.

Multi-step learning Multi-step learning extends the reward for each state to a truncated n-step return (Sutton 1988). Which, with a suitable truncation size has been shown to achieve faster learning. The truncated n-step return for a state $\boldsymbol{s_t}$ is calculated as follows:

$$r_t^{(n)} \equiv \sum_{k=0}^{n-1} \gamma_t^{(k)} r_{t+k+1}. \tag{2.46}$$

The authors (Hessel et al. 2018b) found 3 steps to be a suitable size to perform multi-step learning.

Distributional RL In distributional RL (Bellemare et al. 2017), instead of estimating the Q-value of each action for a given state s_t, the authors estimate a distribution over possible Q-values, which is represented as a neural network. The advantage of this approach is that if the Q-value distribution is bi-modal, the approach can model this distribution as part of the network outputs, rather that the mean of the distribution in the vanilla case.

Noisy Nets (Fortunato et al. 2017) aim to rectify some of the limitations of ϵ-greedy exploration in high-exploration, sparse-reward settings such as the Atari game, Montezuma's Revenge. Noisy nets are implemented by replacing linear layers in the NN with a noisy equivalent:

$$\boldsymbol{y} = (\boldsymbol{b} + \mathbf{W}\boldsymbol{x}) + \left(\boldsymbol{b}_{noisy} \odot \epsilon^{b} + \left(\mathbf{W}_{noisy} \odot \epsilon^{w}\right) \boldsymbol{x}\right). \tag{2.47}$$

Where ϵ^b and ϵ^w are random variables. As the network trains, it will learn to ignore the noisy random variables, by reducing $\mathbf{b}_{noisy}$ and $\boldsymbol{W}_{noisy}$ to zero. But initially this noise allows for more random exploration, allowing to agent to explore different parts of the state space and potentially discover new sources of reward.

The studied performed by the authors of Rainbow performs a detailed ablation study of these methods, a snapshot of which is shown in Figure 2.20.

2.5.4.3 Recurrent Q-Learning R2D2

Recurrent Replay Distributed DQN (R2D2) (Kapturowski et al. 2018) incorporates a recurrent memory cell in an off-policy setting. The challenge of using memory in an off-policy setting is that the recurrent hidden state of the agent must be stored in the replay buffer. When samples are taken from the replay buffer, there will be a mismatch between the stored hidden state activation and the activation that would have been calculated if the current parameters of the neural network were used to calculate the hidden state. This principle is called "representation drift". In R2D2, the authors perform a thorough analysis of the issue, comparing three options: storing the state, zeroing the state and burning in the state; where a fixed number of time-steps are used to estimate the hidden state which are not used for computing gradients. An example of the analysis is shown in Figure 2.21, which demonstrates the necessity to burn-in from a stored hidden state, particular in environments that rely on exploration. At the time of publication, R2D2 was shown to outperform state of the art in the Atari-57 benchmark and the 30 tasks from DeepMind Lab. In addition the authors introduce a technique to estimate the representational drift of a stored state, by comparing the start and end states from an episode stored in the replay buffer to that of a burned-in zeroed hidden state. The Q-discrepancy can be estimated by comparing the two estimated Q-values. Burning-in the hidden state for up to 40 time-steps comes

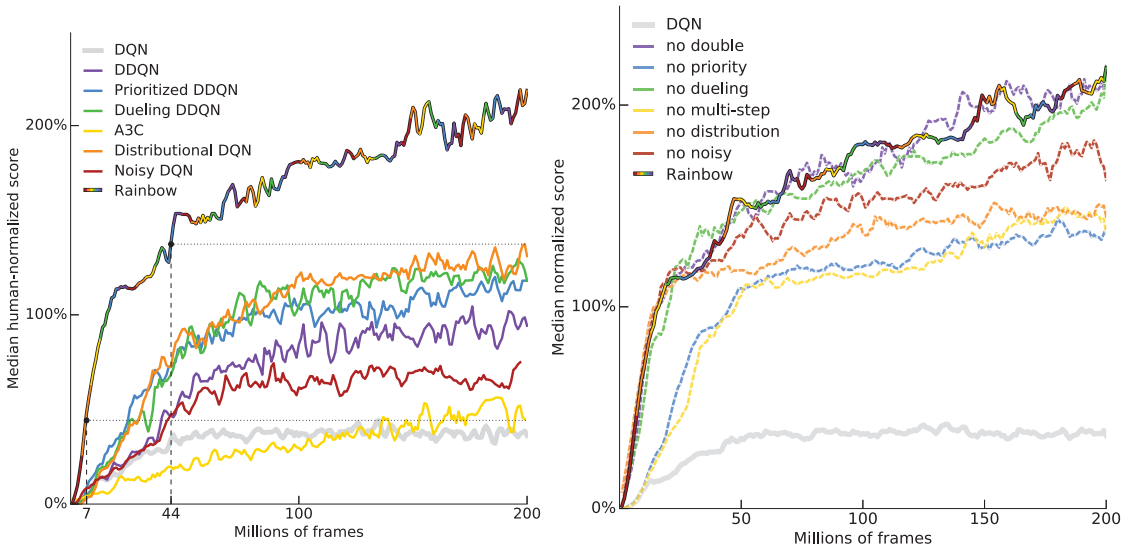


Figure 2.20: **Training curves for the combined components of Rainbow** left: Ablations of Rainbow with individual components switched on independently, we observe that distributional, prioritized and Dueling DQN produce the greatest individual improvement in performance. Right: Ablations of Rainbow where one component is excluded from the training algorithm, we observe that the exclusion of multi-step or prioritized experience replay have the most impact on the performance of the combined algorithm. Figure from (Hessel et al. 2018a).

at additional computational cost. For example, for sequences of 40 steps with 40 steps of burn-in the computation cost for the forward pass is doubled, however the computational requirements of the backwards pass remains the same as we do not need to calculate gradients over the first 40 steps. In this setup, there is therefore 50% more computation to perform, so clearly there is a trade-off to be made in terms of training time, environment interactions and the performance of the final policy.

It is worth mentioning that many of the contributions in the R2D2 work were previously made by the authors of (Lample et al. 2017a), however (Kapturowski et al. 2018) provide a more thorough and in depth empirical analysis.

2.5.4.4 Soft Actor-Critic

The Soft Actor-Critic algorithm (SAC) (Haarnoja et al. 2018a) aims to bridge to gap between on-policy and off-policy approaches by optimizing a stochastic policy in an off-policy way. The original version of SAC was only applicable in continuous action spaces, but recent works have addressed this issue (Christodoulou 2019). The SAC algorithm modifies the reward of each state by introducing entropy-

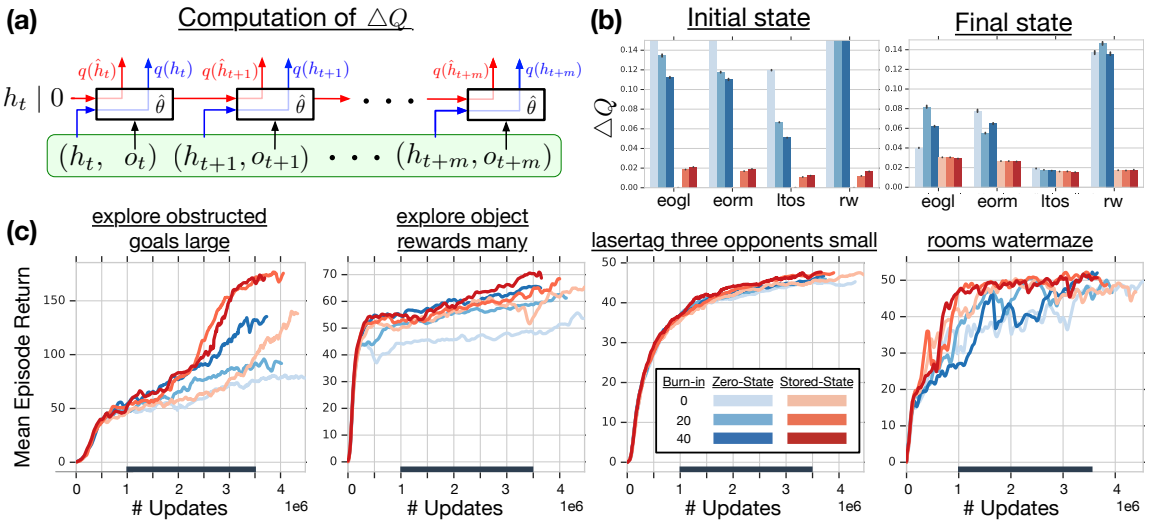


Figure 2.21: **Analysis of performance of the R2D2 algorithm**, (**a**) how the Q-value discrepancy is computed in order to estimate how how storing and burning in the hidden state affects the estimate of the Q-values. (**b**) the Q-value discrepancy for initial and final states in a sequence taken from the replay buffer. (**c**) Learning curves for a variety of scenarios with the different burn-in strategies. In environments that require exploration and sparse rewards, the approach of storing state with burn-in achieves the highest performance. Figure from (Kapturowski et al. 2018).

regularized RL. In entropy-regularized RL the agent's reward is augmented with the additional reward proportional to the entropy H of the policy for that state:

$$H(\pi(a, \boldsymbol{s})) = -\log \pi(a|\boldsymbol{s}). \tag{2.48}$$

This modifies the reward function to depend not only on the underlying MDP but also on the policy's state-action distribution, we define the updated reward function as:

$$r_{SAC}(\boldsymbol{s}, a) = r(\boldsymbol{s}, a) + \alpha H(\pi(a, \boldsymbol{s})). \tag{2.49}$$

Where the α parameter controls the trade-off between the standard reward and the entropy bonus. There are currently two variants of the SAC algorithm, one that varies the α parameter during training and another that keeps it fixed. The state-value and Q-value functions are trained based off this new reward function. The SAC algorithm jointly learns two Q-functions $Q_{\phi 1}, Q_{\phi 2}$ and a policy π_θ. The method implements a trick, known as the clipped double-Q-trick, which takes the minimum over the two Q function approximators:

$$\mathcal{L}\left(\phi_i, \mathcal{D}\right) = \underset{(\boldsymbol{s},a,r,\boldsymbol{s}',d)\sim\mathcal{D}}{\mathbb{E}} \left[\left(Q_{\phi_i}(\boldsymbol{s}, a) - y\left(r, \boldsymbol{s}', d\right) \right)^2 \right]. \tag{2.50}$$

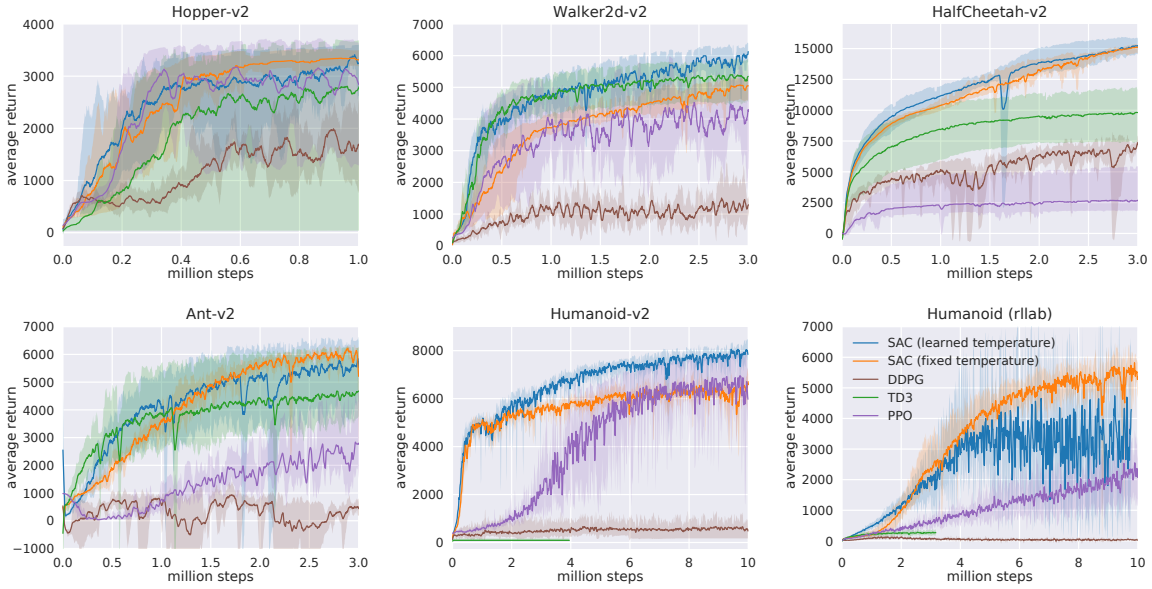


Figure 2.22: **Results of the SAC algorithm applied to continuous control tasks**, compared against PPO (Schulman et al. 2017b), DDPG (Silver et al. 2014) and TD3 (Fujimoto et al. 2018). Figure from (Haarnoja et al. 2018a).

$$y\left(r, \boldsymbol{s}', d\right) = r + \gamma(1-d)\left(\min_{j=1,2} Q_{\phi_{\text{targ},j}}\left(\boldsymbol{s}', \tilde{a}'\right) - \alpha \log \pi_\theta\left(\tilde{a}' \mid \boldsymbol{s}'\right)\right), \quad \tilde{a}' \sim \pi_\theta\left(\cdot \mid \boldsymbol{s}'\right). \tag{2.51}$$

As the policy should aim to maximize the value $V^\pi(\boldsymbol{s})$:

$$V^\pi(\boldsymbol{s}) = \mathop{\mathbb{E}}_{a\sim\pi(.|\boldsymbol{s})}\left[Q^\pi(\boldsymbol{s}, a) - \alpha \log \pi(a \mid \boldsymbol{s})\right]. \tag{2.52}$$

We can reformulate the value function in order to construct the policy loss function, where $\mathcal{D}$ is the agent's replay buffer:

$$L_\pi = \mathop{\mathbb{E}}_{(\boldsymbol{s},a)\sim\mathcal{D}}\left[\min_{j=1,2} Q_{\phi_j}\left(\boldsymbol{s}, a; \theta_j\right) - \alpha \log \pi\left(a \mid \boldsymbol{s}\right)\right]. \tag{2.53}$$

The parameters of which can be learned with gradient ascent. SAC has shown to be a powerful algorithm, that at the time of publication achieved state of the art results in numerous continuous control tasks shown in Figure 2.22.

2.5.5 Deep RL Frameworks

There are numerous Deep RL frameworks that implement the vast majority of the state of the art algorithms. We discuss here some popular RL frameworks. The most fully featured framework is probably that of Ray RLlib (Liang et al. 2018)

which contains over 16 RL algorithms and includes other approaches such as evolutionary strategies. StableBaselines (Hill et al. 2018) is a popular and well tested RL framework that includes 11 TensorFlow (Abadi et al. 2015) implementations and 7 PyTorch (Paszke et al. 2019) implementations of RL algorithms, the downside being that the PyTorch implementations do not yet support recurrent agent architectures. ACME (Hoffman et al. 2020) implements TensorFlow and Jax (Bradbury et al. 2018) versions of 14 RL algorithms. TorchBeast (Küttler et al. 2019) includes distributed implementations of the IMPALA (Espeholt et al. 2018b) and DD-PPO (Wijmans et al. 2019a) algorithms. Tianshou (Weng et al. 2021) offers 20 algorithms, recurrent support and a lightweight code-base, with algorithms implemented in PyTorch. SampleFactory (Petrenko et al. 2020) offers a highly optimized, 100,000 interaction per second, implementation of A-PPO, an asynchronous version of the PPO algorithm.

The availability of these frameworks is clearly a great benefit to the community, but in a research setting using a framework can be a double-edged sword. While it can be fast to implement a baseline agent, the level of abstraction required to make these frameworks flexible enough to handle a variety of algorithms can mean that they are bloated and challenging to implement custom architectures, algorithms and the addition of auxiliary losses. For these reasons, works presented in this thesis use custom implementations of, typically, a single Deep RL algorithm. The frameworks have provided us with the ability to validate the performance of our baseline agent and also the open-source nature of these frameworks means they can be used as a reference implementation.

2.6 Embodied AI Environments

There has been a large expansion to the number of Deep RL environments available in the last five years, the most well known being the gym framework (Brockman et al. 2016) which provides an interface to fully observable games such as the Atari-57 benchmark. Mujuco (Todorov et al. 2012) has remained the de facto standard for continuous control tasks, its recent open-source release can only benefit the larger RL community. OpenSpiel (Lanctot et al. 2019) provides a framework for RL in games such as Chess, Poker, Go, TicTacToe and Habani, to name a few. There are a variety of partially observable 3D simulators available, with more game-like environments such as the ViZDoom simulator (Wydmuch et al. 2018), DeepMind-Lab (Beattie et al. 2016b) and Malmo (Johnson et al. 2016b). A number of photo-realistic simulators have also been released such as Gibson (Xia et al. 2018), AI2Thor (Kolve et al. 2017a), and the highly efficient Habitat-Lab simulator (Savva et al. 2019a; Szot et al. 2021).

In this thesis our focus has been on agents learning to interact in 3D environments from a monocular, egocentric perspective. Here we introduce the two main

Embodied AI simulators that we used to evaluate our contributions. Before we begin it is important to introduce the Gym interface, which has become the norm when interacting with a RL environment. OpenAI Gym (Brockman et al. 2016) is an open source Python library for benchmarking RL algorithms. In addition, the library defines an API for the communication between learning algorithms and environments, the API has been widely adopted and most of the Deep RL frameworks introduced in 2.5.5 support a Gym interface. In 2.6 we show the interface for building and interacting with an environment that complies with the gym API.

```
import gym
env = gym.make('CartPole-v1')

# env is created, now we can interact with it:
for episode in range(10):
    obs = env.reset()
    done = False
    while not done:
        action = env.action_space.sample() # replace with your model
        obs, reward, done, info = env.step(action)
```

Listing 2.1: Example of the gym API

2.6.0.1 The ViZDoom simulator

ViZDoom (Kempka et al. 2017) is a highly efficient AI research platform based on the classical First Person Shooter game Doom. ViZDoom can simulate 3D environments of 10s of thousands of interactions per second on a single CPU and allows for the development of Artificial Intelligence (AI) agents that interact with only visual information. It is lightweight, fast, customizable and allows access to additional state information such as depth buffers and class segmentations. Out of the box, the simulator provides 8 tasks including single agent maze navigation tasks, object collection tasks, scenarios where the agent must depend against computer controlled bots and a multi-agent "DeathMatch" style scenario, example environments are shown in Figure 2.23. The original framework does not provide a gym interface, but one has been implemented by (Hakenes 2018), in chapter 3 we create and benchmark a set of challenging scenarios in the ViZDoom simulator.

2.6.0.2 The Habitat simulator

Habitat (Savva et al. 2019a; Szot et al. 2021) is a performant simulator that has been designed from the ground up for the learning of agent behaviors. The Habitat simulator provides several predefined embodied AI tasks such as point-to-point navigation, instruction following and embodied question answering. Agents can be configured with a number of sensors such as monocular RGB cameras, depth sensors and virtual LIDAR. The simulator provides standard

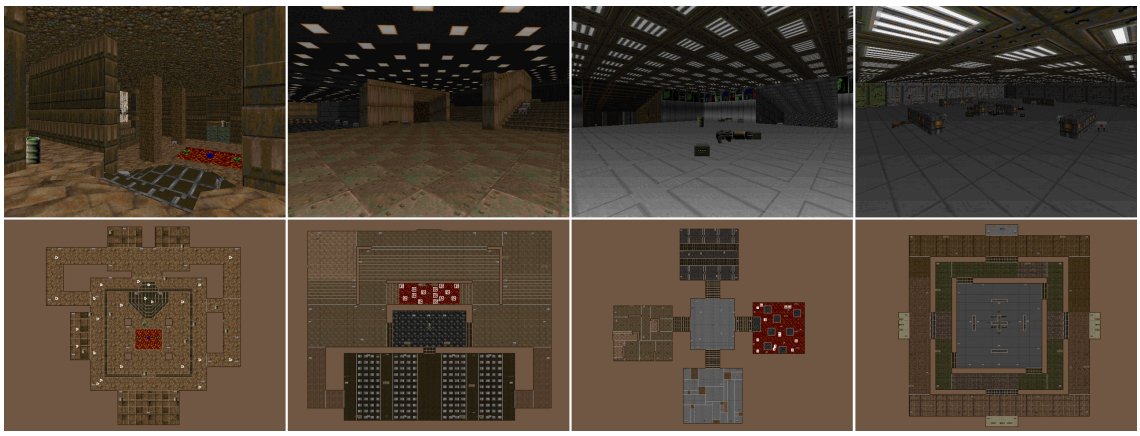


Figure 2.23: **Example maps in the ViZDoom simulator**, shown are environments used for the "DeathMatch" scenario. Figure from (Kempka et al. 2017).

metrics that are common in the evaluation of embodied agents (Anderson et al. 2018a). The simulator has the capacity to load a variety of photorealistic environments, including the Gibson dataset (Xia et al. 2018), Matterport (Chang et al. 2017) and Replica (Szot et al. 2021). The vast majority of datasets available in the simulator are 3D scans of real world buildings, apartments and homes, an example of these environments is shown in Figure 2.24. It is also fairly easy to add new environments to the habitat simulator. During this thesis we had an external company, `https://3dcreation.fr/`, perform a 3D a part of our laboratory which we loaded into the habitat simulator, shown in Figure 2.25. A detailed analysis of transfer of learned policies from simulator to the real world using this dataset was undertaken by (Sadek et al. 2021). Although environments in the habitat dataset are 3D, typically an instance of a problem is evaluated on one floor of the environment, although several episodes can take place on different floors, individual episodes are only on one floor. The environments available in Habitat are generally uncluttered when compared to real-world apartments and houses, which may pose a challenge when transferring policies learned in simulation to a household robot.

While there are various tasks available in the Habitat dataset, the two that we focused on during this PhD were that of point-goal and image-goal. The point-goal task is defined as follows, the agent starts at position p_{start} and must perform GPS guided navigation to position p_{goal}, that is at each time-step the relative position of the goal is provided to the agent. In addition to the relative goal position, the agent's observation can include an RGB(D) monocular image. The typical action space for this task is [TURN_LEFT, TURN_RIGHT, MOVE_FORWARD, STOP], where turns are in 10 degree increments and movements are 0.25 m. Movements can be considered perfect or modeled to use noisy motor actuation. The tasks

Figure 2.24: **Example environments in the Habitat simulator**, shown are renderings from virtual monocular cameras in two homes. There are artifacts in the 3D reconstruction due to interference from external intra-red light sources and from occlusions, nevertheless the reconstructions are of relatively high quality. (Savva et al. 2019a).

also include a STOP action which the agent must output when it considers it have completed the task. In most tasks the stopping distance d_{stop} is 0.2 m of p_{goal} in order to consider an episode to be successful.

In the image-goal task, (Zhu et al. 2017) the agent starts at position p_{start} and must navigate to position p_{goal}. But instead of being provided a relative goal position, the agent's observation includes an image, or set of images, captured at the goal location. Invariably, this task is far more challenging than that of point-goal as the agent must learn to recognize objects and features in the goal image and learn to navigate in towards them. In chapter 5 we extend this task to the sequential image-goal setting, where an agent much navigate to one image-goal location and then to another in order, which tests the agent's capacity to store information about the environment in its internal representation.

Typically two metrics are used to evaluate embodied agents, accuracy and Success weighted by Path Length (SPL) (Anderson et al. 2018a):

$$\frac{1}{N}\sum_{i=1}^{N} S_i \frac{\ell_i}{\max\left(p_i, \ell_i\right)}. \tag{2.54}$$

Which weights the performance of each episode by the ratio of the optimal path length and the episodes' path length, which provide a better indication of the overall optimality of the agent's performance. Accuracy is the average number of episodes that have been completed successfully divided by the number of episodes.

Figure 2.25: **3D scan of the first floor of the CITI laboratory**. Left: Rendered with a virtual monocular RGB camera, right: on overview of the scan.

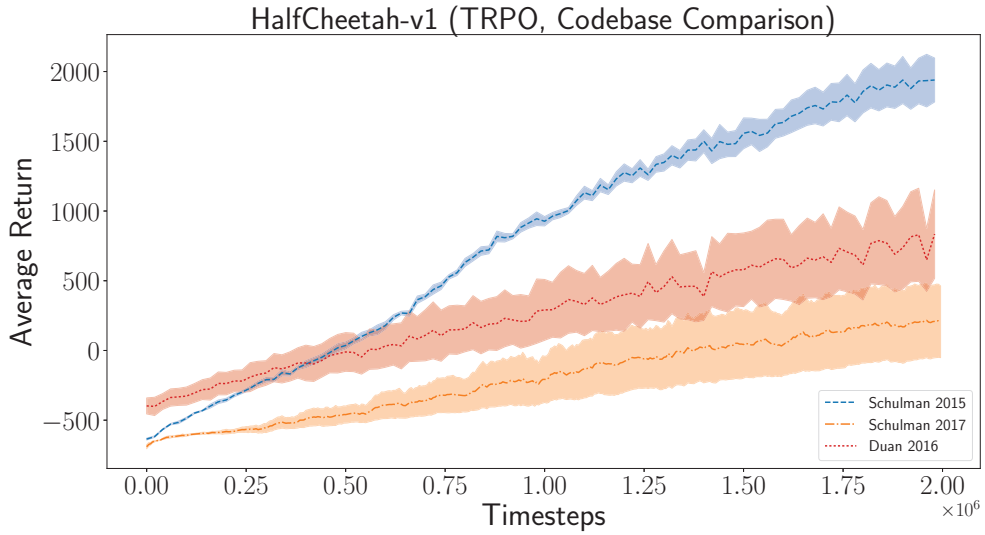


Figure 2.26: **A comparison of three implementations of the TRPO (Schulman et al. 2015) algorithm**. We observe that performance varies greatly between the different implementations. Two of the implementations the author of the TRPO algorithm in 2015 and 2017. Figure from (Henderson et al. 2018).

2.7 Practicalities of Deep RL experimentation

While Deep Reinforcement Learning has made enormous progress in the last decade, there are some inherent weaknesses to this approach. The authors of Deep Reinforcement Learning that Matters (Henderson et al. 2018) highlight several issues in the Deep RL community, notably: reproducibility, hyper-parameter sensitivity, non-deterministic environments and inconsistencies in the reporting of metrics. Through the publication of this work, the authors have spurred discussion in the community. The work is recommended reading for any researcher interested in applying Deep RL. We highlight one example, shown in Figure 2.26 which shows a comparison of different implementations of a Deep RL algorithm TRPO (Schulman et al. 2015). A large difference is observed between the performance of the different implementations, which is surprising considering two of the implementations are from the same author, who is also the inventor of the algorithm. The authors conclude with the following recommendations: When comparing two methods, many random seeds should be used to quantify the variance and mean performance of each approach. Report all hyper-parameters and share code where possible to foster reproducibility. Look to build hyper-parameter agnostic algorithms that are less sensitive to reward scaling. We have endeavored to follow these guidelines in the contributions made during this thesis. Recently (Agarwal et al. 2021) has released an open-source toolbox for analyzing Deep RL performance and provide a rigorous statistical analysis when comparing Deep RL methods.

2.8 Robotics, classical planning and navigation

We highlight here some key works from the field of robotic navigation, as the field is vast we provide a snapshot of related works that are pertinent to some of the structured methods we have developed during this thesis. In particular, we discuss Simultaneous Localization And Mapping (SLAM) and graph search methods.

2.8.0.1 Simultaneous Localization and Mapping

SLAM (Durrant-Whyte et al. 2006) is a technique for planning and navigation in 3D spaces. As the name suggests, the approach performs an estimation of the pose of the robotic agent while in parallel constructing a map of the environment. The approach operates in situations where the robot's odometry is noisy and measurements from the environment are used to correct for the error in the position of the robot. This is achieved by extracting key-points or landmarks from the environment and then measuring the relative location of these landmarks again

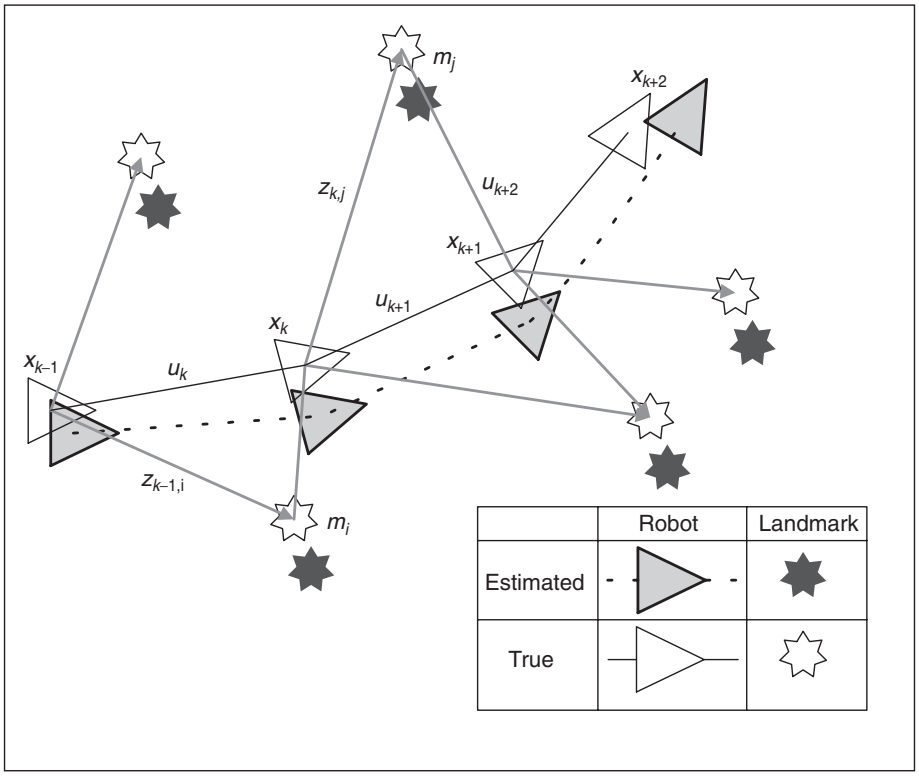


Figure 2.27: **An overview of the SLAM problem**, the system performs a simultaneous estimate of the robot's position and landmark locations. The ground truth locations are undisclosed and are not measured directly. Figure from (Durrant-Whyte et al. 2006).

after the robot has moved. An Extended Kalman filter (Ribeiro 2004) is responsible for updating the estimate of the agent's position based on the observed landmarks. The algorithm performs simultaneous tracking of the errors in the position of both the landmarks and the robot's position. There are numerous variants of the SLAM algorithm that can be based on LIDAR measurements, RGB, RGB-D or key-points extracted from a monocular camera. Popular SLAM variants are ORB SLAM (Mur-Artal et al. 2014; Mur-Artal et al. 2017), LSD SLAM (Engel et al. 2014) and Stereo Parallel Tracking and Mapping (Pire et al. 2017). Classical approaches rely on matching with handcrafted methods, but recently the incorporation of learned depth prediction has been shown to improve performance (Tateno et al. 2017; Bloesch et al. 2018). The addition of learned semantic predictions enables the use of Semantic SLAM (Bowman et al. 2017) which can create maps that include predictions of vehicles, buildings, pedestrians and navigable space.

Recent work (Mishkin et al. 2019) has benchmarked SLAM approaches against end-to-end learning-based methods in common simulated 3D environments and performs a detailed evaluation in environments of varying difficulty. (Mishkin

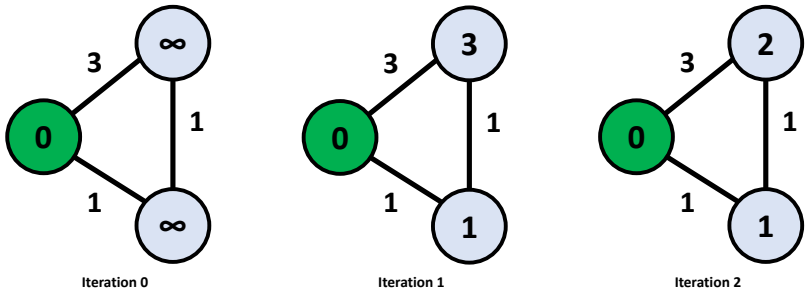


Figure 2.28: **Three iterations of the Bellman-Ford algorithm**. Shown for an example undirected graph with three vertices with the source node in green.

et al. 2019) found that the classic SLAM pipeline can perform well in complex cluttered environments. However, the learned end-to-end systems are more robust when operating with a limited sensor suite. In addition, both the handcrafted SLAM approach and the end-to-end approach are far below human performance.

2.8.1 Graph search

In environments where the map is known and a (directed) graph G can be constructed of vertices V and edges E, optimal planning algorithms can be used to find the shortest path between two locations. The most popular algorithms include Dijkstra (Dijkstra 1959) and heuristic-based algorithms such as A* (Hart et al. 1968b). There are extended versions of A* such as D* (Stentz 1997; Stentz et al. 1995; Koenig et al. 2005) and LPA* (Koenig et al. 2004b), which modify the heuristic used the incorporate unknown terrain and allow for updates to the map based on new information. There are implementations of hierarchical graph search algorithms (Fernandez et al. 1998; Fernández-Madrigal et al. 2002), where the graph is decomposed into several graphs at multiple scales allowing coarse-grained planning over large distances and finer planning over short distances. A further popular method is that of the Fast Marching method (Sethian 1996; Sethian 1999), which is a wavefront-based approach to calculate distances from one particular cell to its neighbors. Here we explain two classical graph search algorithms that were used during the thesis, Bellman-ford and Dijkstra.

2.8.1.1 Bellman-Ford

This algorithm computes the shortest path from a single source vertex to all other vertices in a weighted directed graph. A strength of this algorithm is that it is able to handle edges with a negative cost, although we will omit this feature for brevity as it was not required for the work undertaken in this thesis. In addition, the

algorithm does not require a priority queue, which provides certain advantages when we implemented a differentiable approximation, introduced in chapter 5. The algorithm runs in $O(|V||E|)$ time, where $|V|$ and $|E|$ are the number of vertices and edges. At the start of the algorithm, each vertex is initialized with a bound on the cost to the source node, which is first set to a value of infinity. The shortest paths are identified by iteratively updating a bound on the shortest distance from each node to a target vertex. At each update a vertex queries its neighbors, updating the bound based on:

$$distance_j = min[distance_i + w_{ij}]. \tag{2.55}$$

Where *distance* is the last bound estimate and w_{ij} is the cost between vertex i and vertex j. The algorithm is detailed in 2.4. Three iterations of the algorithm are demonstrated on a toy problem in Figure 2.28.

Algorithm 2.4 Bellman-Ford

```
function BELLMAN-FORD(List vertices, List edges, Vertex source)
    distance ← List of size n
    predecessor ← List of size n
    for vertex v in vertices do
        distance[v] ← inf
        predecessor[v] ← null
        distance[source] ← 0
    end for
    for i ← 1 to |V| do
        for edge (u, v, w) in edges do
            if distance[u] + w ≤ distance[v] then
                distance[v] ← distance[u] + w
                predecessor[v] ← u
            end if
        end for
    end for
    return distance, predecessor
end function
```

2.8.1.2 Dijkstra

Dijksta's algorithm is one of the most well known graph search algorithms. It can be used to find the shortest paths between two nodes on a digraph or the shortest paths to all nodes from a source node. This algorithm was used to generate the ground truth for our work in chapter 5. Unlike the Bellman-Ford algorithm, the algorithm requires all edges to have a positive cost. The algorithm requires a

priority queue and runs in $O((|V| + |E|)log|V|)$ time, where $|V|$ and $|E|$ are the number of vertices and edges respectively. The algorithm is detailed in 2.5.

Algorithm 2.5 Dijkstra

```
function DIJKSTRA(Graph graph, Vertex source)
    create priority queue Q
    for each vertex v in Graph do
        if v ≠ source then
            distance[v] ← inf
            predecessor[v] ← null
        end if
    end for
    Q.add(v, dist[v])
    while Q is not empty do
        u ← Q.extract_min()
        for each neighbor v of u do
            alt ← distance[u] + cost(u, v)
            if alt < distance[v] then
                distance[v] ← alt
                predecessor[v] ← u
            end if
        end for
    end while
    return distance, predecessor
end function
```

2.9 Structured Deep Reinforcement Learning

2.9.1 Motivation

As we deploy Embodied AI agents in ever more complex environments, their capacity to reason about the world they inhabit must be expanded. One approach to this problem, that we discussed in section 2.7, is increasing the number of parameters available in the agent's network architecture. The difficulty of this strategy is that many network components scale in a non-linear manner. For example, an LSTM module, common in memory-based agents, stores its internal representation in two 1D vectors $\boldsymbol{h}$ and $\boldsymbol{c}$, if we wish to double the capacity of $\boldsymbol{h}$ and $\boldsymbol{c}$ there is a quadratic scaling of the LSTM module's parameters. As we encounter ever more complex scenarios the strategy of scaling will inevitably reach its limits, particularly if we wish to deploy a trained system in the real world where inference time and hardware limitations must be considered. In addition, the unstable nature of Deep RL training is exacerbated by the increase in model complexity.

In this thesis, we explore an alternate approach, whereby analyzing the underlying problem or environment the agent interacts with, we structure components of the agent's network architecture to exploit physical properties of the environment such as projective geometry, estimated connectivity of free space, and building metric or topological maps. To enhance the agent's reasoning processes we add the capacity to self-attending to its internal memory and reason about the location of previously observed environment features. By constructing an embodied agent in such a manner we aim to:

- Improve the generalization performance of the agent when evaluated in unseen environment instances.
- Increase sample efficiency when learning a policy, when compared to an unstructured agent.
- Provide the ability to interpret the agent's reasoning process, through visualization of its structured representations.

The construction of structured network priors through the decomposition of the agent's task and environment is a niche but expanding sub-domain of Deep RL. The following section introduces prior and concurrent works that were studied during this thesis.

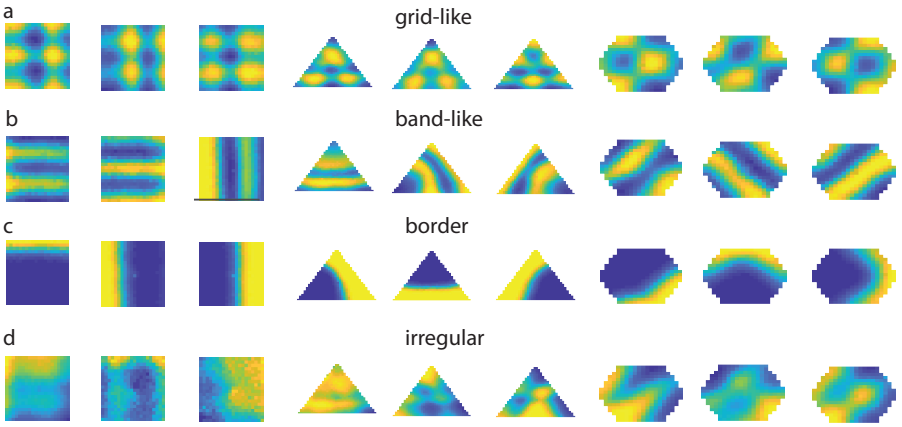


Figure 2.29: **Artificial grid cell responses in a trained RNN**. Color indicated the intensity of the activity ranging from low (blue) to high (yellow). a) Grid-like responses. b) Responses that contain banding. c) Border-cell-like responses. d) Irregular responses. Figure from (Cueva et al. 2018).

2.9.2 Structured approaches

Works in the domain of Structured Deep RL fit broadly in two categories: algorithmic structure and map-based structure. Within these approaches, there are also some variations such as metric and topological maps. The inspiration of many of the map-based approaches is biologically inspired. Innately humans store key-points about their environment and navigate using way-points with estimates of the shortest path between two locations. There is a plethora of neuroscience research into the internal representations of mammals. Rodents develop grid cells, which fire at different locations with different spatial frequencies and phases, a discovery that led to the 2014 Nobel prize in medicine (O'Keefe et al. 1971; Hafting et al. 2005). A similar structure emerges in artificial neural networks trained to localize themselves, discovered independently by two different research groups (Cueva et al. 2018; Banino et al. 2018), refer to Figure 2.29.

2.9.2.1 Algorithmic Structure

In several works, the network architecture has been structured such that it can learn the operations of a specific algorithm. In the case of Value Iteration Networks (Tamar et al. 2016) the authors aim to learn a general policy that solves navigation in either grid-world or continuous environments. The state representation is an image of the environment, in the case of the grid-world each pixel is an obstacle, free-space, the agent's location or the goal location. The authors demonstrate that a policy based on CNN architecture with 5 convolutional layers achieves a

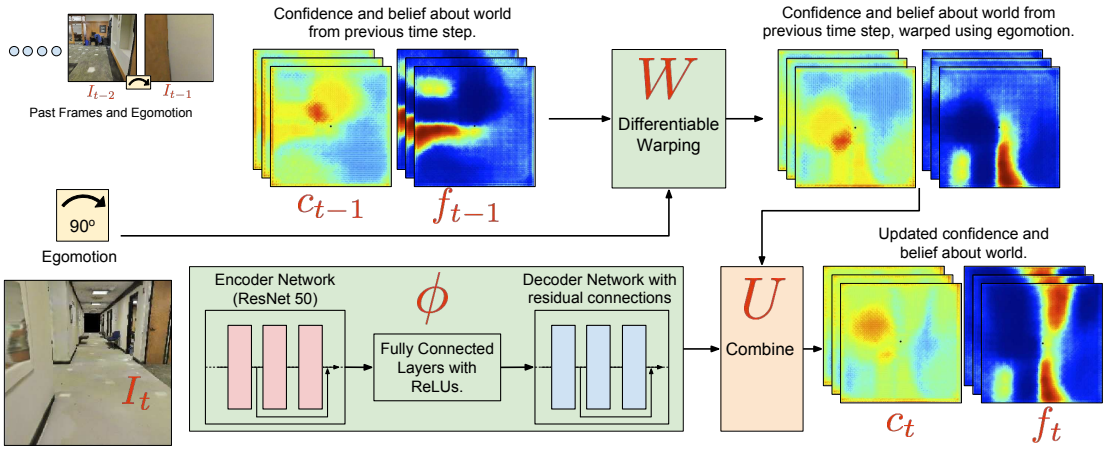


Figure 2.30: **Cognitive mapping and planning** Deep Neural network architecture from Cognitive mapping and planning. Figure from (Gupta et al. 2017a).

lower generalization performance than a CNN architecture structured so that it can represent a differential approximation of the value-iteration algorithm (Bellman 1957). Universal Planning Networks (Srinivas et al. 2018) perform goal-directed navigation in a continuous control setting, by learning a forward planning model in a latent space to infer the optimal sequence of actions to take. They exploit the learned UPN representations to create a dense reward function that is used to augment the performance of model-free RL on more challenging tasks with image-based goals.

The authors of QMDP-Net (Karkus et al. 2017) construct a recurrent policy network for planning in partially observable problems. The recurrent policy embeds a planning algorithm in the network's architecture. Allowing end-to-end training in a fully differentiable manner. It is intriguing that the network encodes a trainable representation of the QMDP algorithm (Littman et al. 1995), but is shown to outperform the QMDP algorithm in a subset of their experiments, thanks to the adaptability of end-to-end learning.

2.9.2.2 Metric map-based approaches

Grid-based structured neural memory comes as a natural extension to the discretized metric maps used in SLAM and other planning methods. These approaches can be considered to be loosely biologically inspired, with examples found in rodents of grid-cells and place-cells. Cognitive Mapping and Planning for Visual Navigation (Gupta et al. 2017a) was the first spatially structured neural network approach for robotic control. The authors created a Deep Neural Network architecture that contains a learned belief state, that is translated and rotated from one

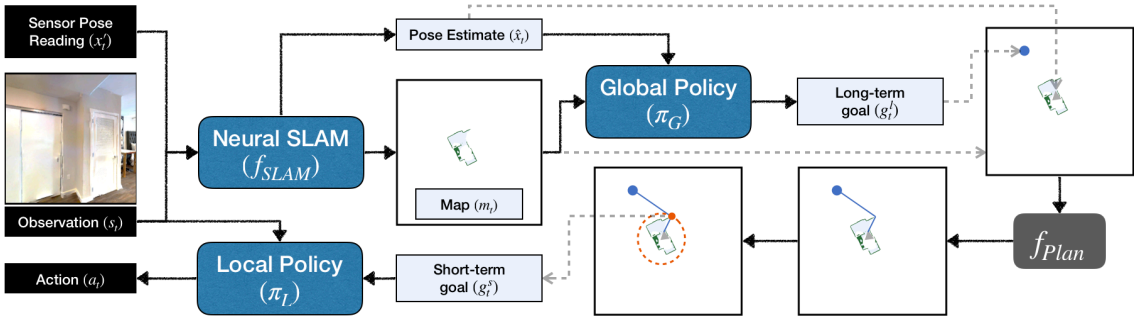


Figure 2.31: **The active neural SLAM architecture**, the system predicts both the agent pose and a map from RGB and sensor observations. Planning and navigation are decoupled by using a global policy to output long-term goals and a local policy to navigate to short-term goals which are computed with an analytic path planner. Figure from (Chaplot et al. 2020a).

time-step to the next depending on the ego-motion of the agent. An overview of the architecture is shown in Figure 2.30, the structured hidden state is passed from one time-step to the next, allowing the agent to perform spatial reason and planning. In this work, the policy was learned with a technique called imitation learning, where we are provided with sequences of observations and action labels, created by a (human) expert to supervise the training process. A limitation is that the ego-motions of the agent are restricted to 90-degree rotations.

MapNet (Henriques et al. 2018a) build an allocentric map of the environment using a *ground projection module* to map CNN feature vectors from screen-space to world-space, which are integrated into the map with an LSTM-based update operation. The parameters of the architecture are learned with imitation learning, rather than a Deep RL approach. Their work also performs a localization step so it is not dependent on measurements of the agent's ego-motion.

The authors of Neural map: Structured memory for deep reinforcement learning (Parisotto et al. 2017a), develop a model with a spatially structured neural memory that is learned end-to-end with Deep reinforcement Learning, the memory is read with self-attention and global read operations with a CNN. A potential downside to this approach is what the agent observes is mapped to where the agent is located, not to where the features from the observations are located in the environment. In addition, the agent's action space is limited to 90-degree rotations, limiting the applicability of this approach.

Neural SLAM (Zhang et al. 2017a) aims to mimic the SLAM algorithm in a differentiable manner, with the use of an external memory architecture which is addressed with attention. They demonstrate a large improvement in performance when compared to an unstructured baseline agent, in environments where long-

Figure 2.32: **Steps of Search on the Replay buffer**. When provided with an initial state and a goal state, their system finds a sequence of intermediate waypoints. A lower level agent can follow these waypoints to navigate to the goal. Figure from (Eysenbach et al. 2019a).

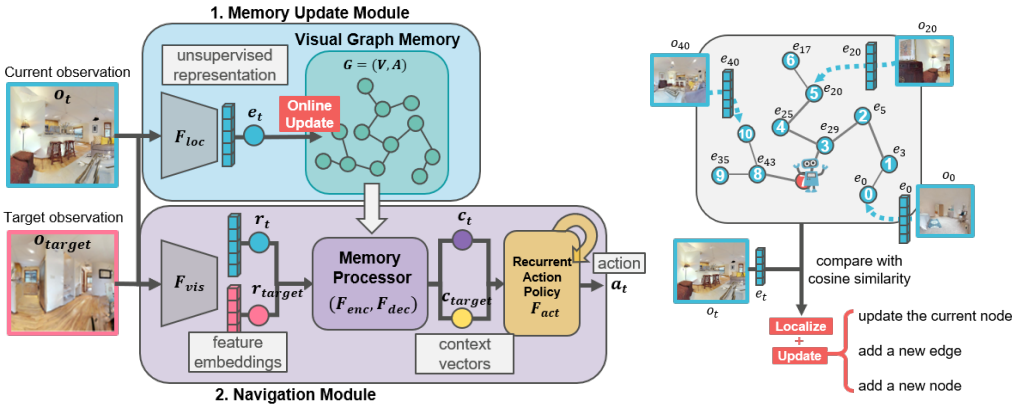
term memory is a requirement. The approach is limited to grid-worlds and some initial experimentation in Gazebo (Koenig et al. 2004a) environments.

Learning to explore using active neural SLAM (Chaplot et al. 2020a) couples a high-level global policy, an analytic planner and a low-level local policy to perform long-term planning and low-level control. The authors incorporate noise models for both the sensors and the robot's actuation, based on data collected using the LoCoBot platform (Wögerer et al. 2012). The network is trained with a combination of PPO (Schulman et al. 2017b) and supervised learning to perform an exploratory task that maximizes coverage, the free space observed by the agent during each episode. The authors demonstrate that the learned policy can be transferred to the task of point-goal navigation, achieving state of the art performance at the time of publication. An overview of the overall agent architecture is shown in Figure 2.31.

(Wani et al. 2020) benchmark a number of metric map methods (Fang et al. 2019; Oh et al. 2016), in addition to our EgoMap publication (Beeching et al. 2020c). They extend these works by creating a global map that stored predictions of object categories, that is trained with an auxiliary loss supervised with ground truth object categories. A similar approach is employed by (Chaplot et al. 2020d) to build an episodic semantic map of the environment.

2.9.2.3 Topological map-based approaches

Humans appear to innately navigate through key-point or landmark-based planning, we recognize and store key locations in our environment and use these to self-localize and perform hierarchical planning and navigation. Topological



(a) Overview

(b) Memory Update

Figure 2.33: **Overview of the network architecture from (Kwon et al. 2021),** shown are two modules: 1) the memory update model, which builds a visual graph memory using features extracted from visual observations. 2) The navigation module which encodes the current observation and target images, the memory processor looks to associate each encoded image with node on the graph. The output of the memory processor is used as input to a recurrent action policy. Figure from (Kwon et al. 2021).

map-based approaches aim to incorporate this form of structure in order to improve their performance in planning and navigation tasks. In (Savinov et al. 2018b) the authors introduce a novel memory architecture for navigation in unseen environment instances. The work comprises of two components: a parametric network that is trained to identify graph nodes based on observations and a non-parametric classical planner. The authors build a graph-based representation by predicting connectivity between observations and thresholding the predictions to build a graph. They then use classical planning techniques to estimate paths to a target location, indicated by an image. They apply this approach to the task of image-goal navigation. A 5-minute roll-out in the environment is used to instantiate the graph, which makes it challenging to compare this work to a memory-based baseline that has not previously observed the environment.

In (Eysenbach et al. 2019a) the authors build a graph-based representation from data available on the RL agent's replay buffer. A goal-conditioned value function allows for the creation of a graph where the edges are the value estimates, using this approach the authors generate a sequence of intermediate sub-goals for the agent to navigate to, illustrated in Figure 2.32. The downside of this approach, as with (Savinov et al. 2018b) is the reliance on the replay buffer dataset, which restricts the application of this method to environment instances that are available in the replay buffer. (Chaplot et al. 2020e) builds upon their work in metric maps

(Chaplot et al. 2020a) to perform image-goal based-navigation in photo-realistic 3D environments. This work exploits graph-based representations, as in the work of (Savinov et al. 2018b), but has the advantage of building the topological representation in a on-line dynamic way. Allowing for generalization to new scenes without the need for an exploratory roll-out. The photo-realistic nature of the simulator used in this work (Savva et al. 2019b; Szot et al. 2021) enables models that can extract feature representations based on rich semantic observations. This allows the model to exploit regularities in the structure and layout of homes that may not exist in synthetic datasets. The authors also deploy their embodied agent under noisy actuation, to mimic the challenges of real-world robotic interactions. The simulated robotic agent is equipped with a 360-degree panoramic camera, which simplifies the task as the agent can observe in all directions without the need to interact with the environment.

(Kwon et al. 2021) implement a self-supervised method to learn observation embeddings for visual representations sampled in photo-realistic environments in the Habitat simulator. They couple the observation embeddings with an end-to-end approach to planning and navigation, involving graph neural networks and a recurrent action policy. The architecture is shown in Figure 2.33. The authors compare against a number of state of the art approaches (Savinov et al. 2018b; Chaplot et al. 2020a; Chaplot et al. 2020e; Fang et al. 2019), including our Learning to Plan with Uncertain Topological Maps publication (Beeching et al. 2020e).

2.10 Conclusions

In this chapter, we have aimed to introduce a foundational knowledge to learning-based approaches and their applications. We first introduced Supervised Learning and its application in classification and regression problems. We then describe methods that learn through interaction with an agent-environment framework, Reinforcement Learning. The deep learning section provided an overview of key modules and architectures including Multi-layer perceptrons, convolutional neural networks, recurrent neural networks, and graph neural networks. We developed the mathematics of gradient-based learning with back-propagation. After highlighting some of the applications of Deep Learning we discussed model-free Deep Reinforcement Learning in on-policy and off-policy settings. We introduced the Embodied AI environments that have been used and built upon during the thesis. We then discussed some of the practicalities and limitations of Deep Reinforcement Learning approaches. After a brief overview of classical planning and navigation in robotics, we introduced the sub-domain of Structured Deep Reinforcement Learning and described key contributions and related works.

In the upcoming five chapters, we introduce five contributions that we have made during this three-year thesis. We begin with Chapter 3, the creation of a set of memory-based tasks, baseline agents and analysis of their behaviors.

CHAPTER 3

DEEP REINFORCEMENT LEARNING ON A BUDGET: 3D CONTROL AND REASONING WITHOUT A SUPERCOMPUTER

Chapter abstract

An important goal of research in Deep Reinforcement Learning in mobile robotics is to train agents capable of solving complex tasks, which require a high level of scene understanding and reasoning from an egocentric perspective. When trained from simulations, optimal environments should satisfy a currently unobtainable combination of high-fidelity photographic observations, massive amounts of different environment configurations and fast simulation speeds. In this chapter we argue that research on training agents capable of complex reasoning can be simplified by decoupling from the requirement of high fidelity photographic observations. We present a suite of tasks requiring complex reasoning and exploration in continuous, partially observable 3D environments. The objective is to provide challenging scenarios and a robust baseline agent architecture that can be trained on mid-range consumer hardware in under 24h. Our scenarios combine two key advantages: (i) they are based on a simple but highly efficient 3D environment (ViZDoom) which allows high speed simulation (12000fps); (ii) the scenarios provide the user with a range of difficulty settings, in order to identify the limitations of current state of the art algorithms and network architectures. We aim to increase accessibility to the field of Deep-RL by providing baselines for challenging scenarios where new ideas can be iterated on quickly. We argue that the community should be able to address challenging problems in reasoning of mobile agents without the need for a large compute infrastructure. Code for the generation of scenarios and training of baselines is available online at the following repository [1].

The work in this chapter has led to the publication of conference paper:

- Edward Beeching, Jilles Dibangoye, Olivier Simonin, and Christian Wolf (2020a). "Deep Reinforcement Learning on a Budget: 3D Control and Reasoning Without a Supercomputer." In: *Proceedings of the International Conference on Pattern Recognition (ICPR)*;

[1] https://github.com/edbeeching/3d_control_deep_rl

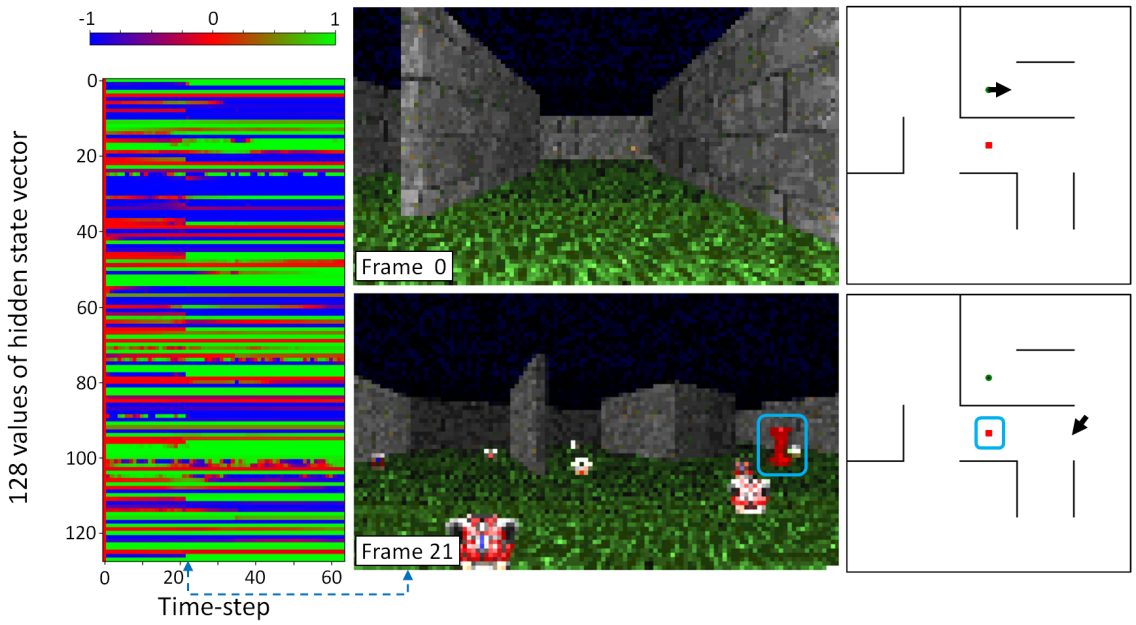


Figure 3.1: **Agent hidden state analysis.** One of the proposed scenarios, called "*Two color correlation*": an agent collects items that are the same color as a fixed object at the center of the scenario, which can be either red or green. Left: analysis of the hidden state of a trained agent during an episode. We observe a large change in the activations of the hidden state at step 21, when the fixed object is first observed. Top-center: the agent's viewpoint at step 0. Bottom-center: the agent's viewpoint at step 21, where the agent first observes the fixed object that indicates the objective of the current task (circled in blue). Right: the positions of the agent in the environment.

3.1 Introduction

Much of the state of the art research in the field of Deep-RL focuses on problems in fully observable state spaces such as Atari games or in continuous control tasks in robotics, with typical work in grasping (Levine et al. 2016b), locomotion (Lillicrap et al. 2015a), and in joint navigation and visual recognition (Mirowski et al. 2017; Das et al. 2018). We address the latter category of problems, which have long term goals and applications in mobile robotics, service robotics, autonomous vehicles and others. We argue that these problems are particularly challenging, as they require the agent to solve difficult perception and scene understanding problems in complex and cluttered environments. Generalization to unseen environments is particularly important, as an agent should be deployable to a new environment, for instance an apartment, house or factory, without being retrained, or with minimal adaptation. In these scenarios, partial observations of the environment

are received from a first person viewpoint. 3D environments require solving the problems of vision, reasoning and long term planning, and are often less reactive than the Atari environments.

The current state of the art Deep RL algorithms are data hungry, as complex scenarios that necessitate high-level reasoning require hundreds of millions of interactions with the environment in order to converge to a reasonable policy. Even with more sample efficient algorithms such as PPO (Schulman et al. 2017b), TRPO (Wu et al. 2017) or DDPG (Lillicrap et al. 2015a) the sample efficiency is still in the tens of millions. Therefore, even though the question of whether to learn from real interactions or simulated ones is still open, there seems to be a consensus that learning from simulations is currently required at least in parts in order to have access to large scale and high-speed data necessary to train high-capacity neural models. In this respect, an optimal simulated environment should have the following properties: (i) relatively **complex reasoning** should be required to solve the tasks; (ii) the simulated observations are of **high-fidelity / photo-realistic**, which makes it easier to transfer policies from simulation to real environments; (iii) **large-scale**, in a sense that the amount of settings (apartments or houses for home robotics) can be varied. The learned policy should generalize to unseen environments and not encode any particular spatial configuration; (iv) **high simulation speed** is required in order to train agents on a massive amount of interactions in a relatively short time.

Unfortunately most of these goals are contradictory. In the past two years many realistic environments have been proposed that offer proxy robotic control tasks. Many of these simulators allow photo-realistic quality rendering but unfortunately only provide a low amount of configurations (apartments) and are slow to simulate even at the lowest of resolutions (Brodeur et al. 2018), (Savva et al. 2017), (Kolve et al. 2017b). This has led to a push to run simulation in parallel on hundreds of CPU cores.

In this work, we argue that an important step in current research is training agents which are capable of solving tasks requiring fairly complex reasoning, navigation in 3D environments from egocentric observations, interactions with a large amount of objects and affordances etc. Transfer of learned policies to the real world, while certainly being important, can arguably explored in parallel for a large part of our work (as a community). We therefore propose set of highly configurable tasks which can be procedurally generated, while at the same time providing highly efficient simulation speed (see Fig. 4.1 for an example). The work presented aims to be a starting point for research groups who wish to apply Deep RL to challenging 3D control tasks that require understanding of vision, reasoning and long term planning, without investing in large compute resources.

We also provide a widely used training algorithm and benchmarks that can be trained on mid-range hardware, such as 4 CPU cores and a single mid-range GPU (K80, Titan X, GTX 1080), within 24 hours. We provide benchmark results

Table 3.1: **Comparison of the first person 3D environment simulators available in the RL community.** Frames per second (FPS) is shown for a single CPU core.

Simulator	Difficulty	Duration	Extendable	Realistic	Scenarios	FPS
VizDoom(Kempka et al. 2017)	Simple	Short	✓	✗	8	3000+
Deepmind Lab(Beattie et al. 2016b)	Complex	Long	✓	✗	30	200+
HoME(Brodeur et al. 2018)	Simple	Short	✗	✓	45000	200+
Habitat(Savva et al. 2019b)	Simple + text	Short	✗	✓	45662	200+
Minos(Savva et al. 2017)	Simple	Short	✗	✓	90	100+
Gibson(Xia et al. 2018)	Simple + text	Short	✗	✓	572	50+
Matterport(Anderson et al. 2018b)	Simple + text	Short	✗	✓	90	10+
AI2-thor(Kolve et al. 2017b)	Simple	Short	✗	✓	32	10+

for each scenario in a number of different difficulty settings and identify where there are areas for improvement for future work. We evaluate the generalization performance of the agents on unseen test scenario configurations.

To summarize, we present the following contributions:

- A benchmark suite of proxy tasks for robotics, which require cognitive reasoning while at the same time being computationally efficient. The scenarios have been built on top of the ViZDoom simulator (Kempka et al. 2017) and run at 12,000 environment interactions per second on limited compute, or 9,600 per second including the optimization of the agent's policy network.
- A baseline agent, which solves the standard ViZDoom tasks and is benchmarked against our more challenging and diverse scenarios of varying difficulty settings.
- Experiments that demonstrate generalization performance of an agent in unseen scenario configurations.
- An analysis of the activation's in the hidden state of a trained agent during an episode.

This chapter is structured as follows: section 7.2 discusses the advantages and limitations of simulators available in Deep RL, pre-existing benchmarks and recent analyses of the instability and generalization performance of RL. Section 3.3 describes each scenario and the reasoning we expect the agent to acquire. In section 3.4 we describe the baseline agent's architecture, key components of the training algorithm and the experimental strategy. Results and analysis are shown in section 3.5 and conclusions made in section 3.6.

3.2 Related Work

Deep Reinforcement Learning — In recent years the field of Deep Reinforcement Learning (RL) has gained attention with successes on board games (Silver et al. 2016a) and Atari Games (Mnih et al. 2015b). One key component was the application of deep neural networks (Lecun et al. 1998) to frames from the environment or

game board states. Recent works that have applied Deep RL for the control of an agent in 3D environments such as maze navigation are (Mirowski et al. 2017) and (Jaderberg et al. 2016a) which explored the use of auxiliary tasks such as depth prediction, loop detection and reward prediction to accelerate learning. Meta RL approaches for 3D navigation have been applied by (Wang et al. 2016a) and (Lample et al. 2017b) also accelerated the learning process in 3D environments by prediction of tailored game features. There has also been recent work in the use of street-view scenes in order train an agent to navigate in city environments (Kayalibay et al. 2018). In order to infer long term dependencies and store pertinent information about the environment; network architectures typically incorporate recurrent memory such as Gated Recurrent Units (Chung et al. 2015a) or Long Short-Term Memory (Hochreiter et al. 1997b).
The scaling of Deep RL has produced some impressive results, such as in the IMPALA(Espeholt et al. 2018a) architecture which successfully trained an agent that can achieve human level performance in all 30 of the 3D partially observable DeepMind Lab (Beattie et al. 2016b) tasks; accelerated methods such as in (Stooke et al. 2018) which solve many Atari environments in tens of minutes and the recent achievement in long term planning on the game of Dota by the OpenAI5 (OpenAI 2018) system have shown that Deep RL agents can be trained with long horizons and a variety of settings. All of these systems represent the state of the art in the domain of Deep RL, the downside is that the hardware requirements required to train the agents make this level of research prohibitive for all but the largest of institutions.

Environments — Beyond the environments available in the Gym framework (Brockman et al. 2016) for Atari and continuous control tasks, there are numerous 3D simulators available with a range advantages and disadvantages. Before starting this work, we investigated the limitations of each simulator, which are summarized in table 3.1, as we focus on proxies for in home mobile robots we exclude comparisons with car and drone simulators such as CARLA (Dosovitskiy et al. 2017) and Sim4CV (Müller et al. 2017). In choosing the simulator, speed was of utmost importance as we did not want to starve the GPU of data. The fidelity and realism of the simulator was not of great concern, as the intention was to run the simulator at a low resolution where fine detail cannot be captured. We chose ViZDoom as it is orders of magnitude faster than its competitors. The downside of ViZDoom is that there are only 8 scenarios available which are limited in scope and cannot be used to evaluate the generalization performance of an agent. For this reason we have constructed an ever growing collection of scenarios that aim to test navigation, reasoning and memorization. Four of the scenarios have been included in the proposed benchmark.

Other benchmarks — There are a number of open source benchmark results and libraries available in the field of Deep RL. The most popular being OpenAI baselines (Dhariwal et al. 2017) which focuses on the Atari and Mujuco continuous

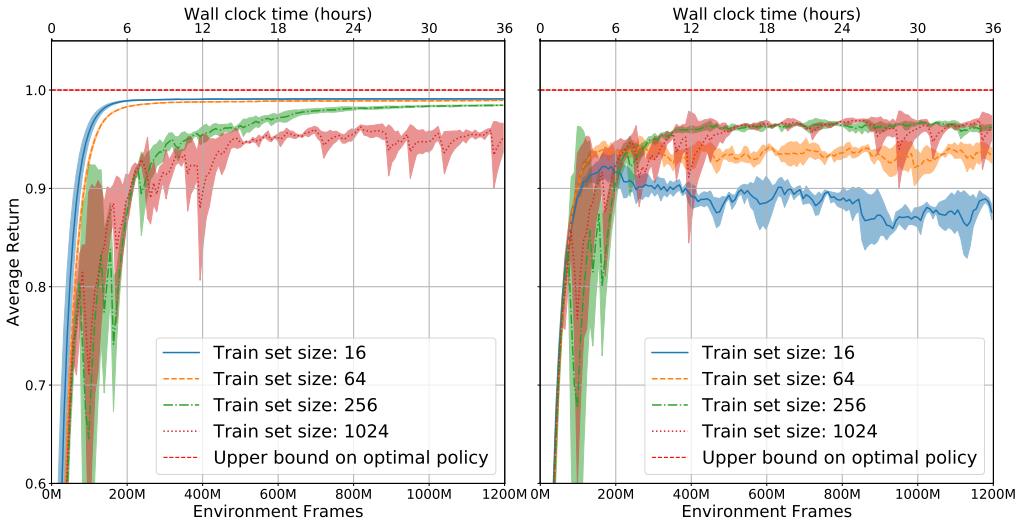


Figure 3.2: **Generalization to unseen maze configurations** The average return and std. of three separate experiments trained on the labyrinth scenario with training set sizes of 16, 64, 256 and 1024 mazes, evaluated on the training set (left) and a held out set of 64 mazes (right). We observe that the training sets of size 16 and 64 quickly overfit to the training data and their generalization performance initially improves and then degrades over time. With 256 and 1024 maze configurations, the agent's test set performance plateaus at the same level.

control tasks. There are several libraries which such as Ray (Moritz et al. 2017) and TensorForce (Schaarschmidt et al. 2017) that provide Open Source implementations of many RL algorithms. Unfortunately these implementations are applied in environments such Atari and Mujuco continuous control and standard benchmark results are not available for Deep RL applied in 3D environments.

To our knowledge, the most similar open-source benchmark of Deep RL applied to 3D scenarios is IMPALA (Espeholt et al. 2018a). Although an impressive feat of engineering, the downside of this solution is the high compute requirement with parallel environments running on up to 500 compute cores. Our scenarios provide comparable levels of reasoning and run on limited hardware due a highly optimized simulator.

Algorithmic instability and generalization — In the last year work has been published on the analysis of the generalization and the often under-documented challenge of algorithmic stability in the field of Deep RL. Much of the focus in Deep RL is training an agent to master a specific task, which is in contrast to supervised learning where data is held out in order to evaluate the performance on similar but unseen data drawn for the same underlying distribution. Recent work has studied over-fitting in Deep RL in 2D gridworld scenarios (Zhang et al. 2018) which motivated us to repeat similar experiments in 3D scenarios with

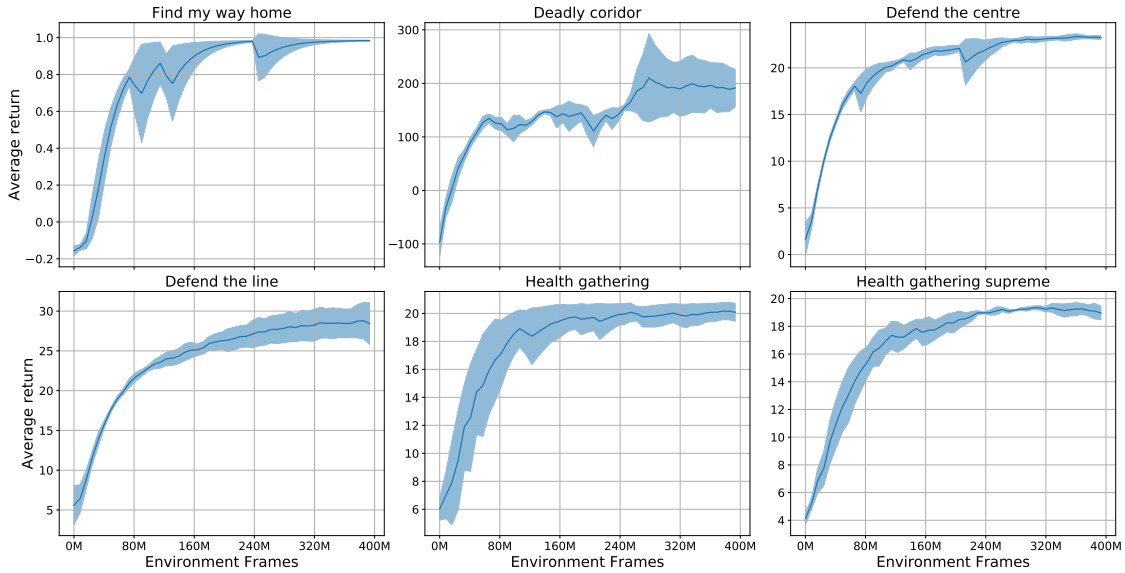


Figure 3.3: **Training curves for the ViZDoom standard scenarios.** Shown are the mean and standard deviation for three independent experiments conducted for each scenario.

separate training and test datasets. An experimental study (Henderson et al. 2018) highlights the sensitivity of Deep RL to weight initialization and hyper parameter configurations which are often overlooked when results are discussed. In the benchmark details section, we explicitly state our training strategy and evaluation procedures.

3.3 Scenarios and required reasoning

We have designed scenarios that aim to:
Be proxies for mobile robotics — the tasks represent simplified versions of real world tasks in mobile and service robotics, where an agent needs to act in an initially unknown environment, discovering key areas and objects. This also requires the tasks to be partially observable and to provide 3D egocentric observations.
Test spatial reasoning — the scenarios require the agent to explore the environment in an efficient manner and to learn spatial-temporal regularities and affordances. The agent needs to autonomously navigate, discover objects, eventually store their positions for later use if they are relevant, their possible interactions, the eventual relationships between the objects and the task at hand. Semantic mapping is a key feature in these scenarios.
Discover semantics from interactions — while solutions exist for semantic mapping and semantic SLAM (Civera et al. 2011a), we are interested in tasks where

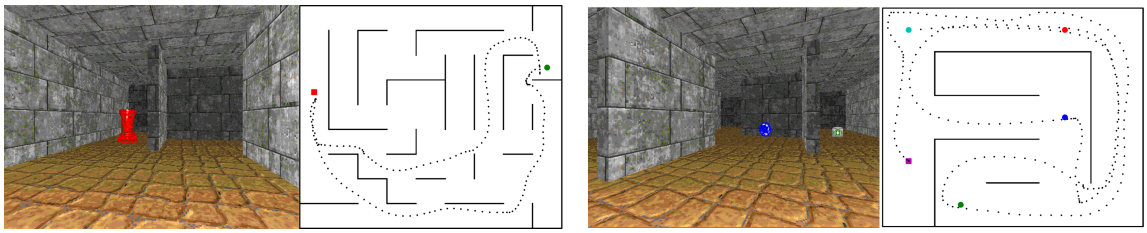


Figure 3.4: **Two scenarios *"Find and return"* and "Ordered $K-$*item*".** Left: The *"Find and return"* scenario showing an agent's egocentric viewpoint and a top down view, the agent must start at the green object, find the red object and then return to the starting point. Right: The "Ordered $K-$*item*" scenario, where the agent must collect 4 items in order.

the semantics of objects and their affordances are not supervised, but defined through the task and thus learned from reward.

Generalize to unseen environments — related to being proxies for robotics, the trained policies should not encode knowledge about a specific environment configuration[2]. The spatial reasoning capabilities described above should thus be learned such that an agent can make a difference between affordances, which generalize over configurations, and between spatial properties which need to be discovered in each instance/episode. We thus generate a large number of different scenarios in a variety of configurations and difficult settings and split them into different subsets, evaluating the performance on test data.

We propose a set of four tasks called "*Labyrinth*", "*Find and return*", "*Ordered K-item*" and "*Two color correlation*". They are build on top of the ViZDoom simulator (Kempka et al. 2017) and involve an agent moving in a maze with various objects. They have continuous partially observable state spaces with a discrete action space including {*forward, backward, turn left, turn right,...*}. The mazes were generated with a depth first search, further interconnections were added by randomly removing 40% of the walls. For the two color scenario the percentage of walls removed was varied in order to evaluate performance with changing scenario complexity. For each scenario, 256 configurations were used for training and 64 for testing. This was determined with a study on the labyrinth scenario (described in section 3.3) with the baseline agent described in section 3.4. We concluded that 256 scenarios are sufficient to train a general policy, shown in Fig. 3.2.

We chose to run the simulator at a resolution of size 64×112 (h×w), which is comparable in size to the state of the art (Espeholt et al. 2018a) of 72×96. We observed that a wider aspect ratio increases the field of view of the agent and enables each observation to provide more information about the environment for a similar pixel budget. We run the same architeture and hyperparameter settings on the standard ViZDoom scenarios, results are shown in Figure 3.3.

[2]In service robotics we require a robot to work in an unseen home.

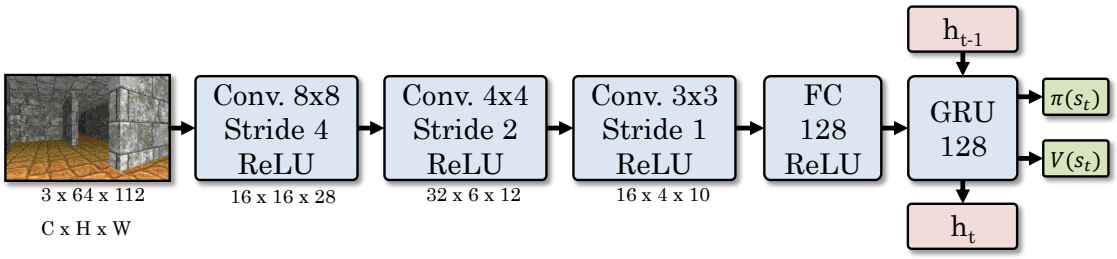


Figure 3.5: **The benchmark agent's architecture**

Labyrinth — The easiest scenario, labyrinth, tests the ability of an agent to efficiently explore an environment and learn to recognize an exit object. A typical maze scenario, with wall configurations created with procedural generation. Rewards are sparse with long time dependencies between actions and rewards. **Objective:** Find the exit of a randomly generated maze of size n×n in the shortest time. **Reward:** +1 for finding the exit, -0.0001 for each time-step, limited to 2100 steps.

Find and return — A general and open source implementation of the "Minotaur" scenario detailed in (Parisotto et al. 2017b), it tests the spatial reasoning of an agent by encouraging it to store in it's hidden state the configuration of an environment, so that it can quickly return to the starting point once the intermediate object has been discovered (see Fig. 3.4). **Objective:** Find an object in a maze and then return to the starting point. **Reward:** +0.5 for finding the object, +0.5 for return to the starting point, -0.0001 for each time-step, limited to 2100 steps.

Ordered k-item — In this more challenging scenario, an agent must find K objects in a fixed order. It tests three aspects of an agent, it's ability to: explore the environment in an efficient manner, learn to collect items in a predefined order and store as part of it's hidden state where items were located so they can be retrieved in the correct order. Shown in Fig. 3.4. **Objective:** Collect k items in a predefined order. **Reward:** +0.5 for finding an object in the right order, -0.25 for finding an object in the wrong order, which also ends the scenario, -0.0001 for each time-step, limited to 2100 steps.

Two color correlation — Potentially the most challenging of the scenarios, here the agent must infer the correlation between a large red or green colored object in the scenario and the correct items to collect. The scenario tests an agent's ability to explore the environment quickly in order to identify the current task and then to store the current task in it's hidden state for the duration of the scenario. **Objective:** Collect the correct color objects in a maze full of red and green objects, the task is to pick up the object which is the same color as a totem at the centre of the environment. The agent also starts with 100 health, collecting the correct item increases health by 25, collecting the wrong item decreases the agent's health by 25, if the agent's health is below 0, the scenario is reset. The agent's health also

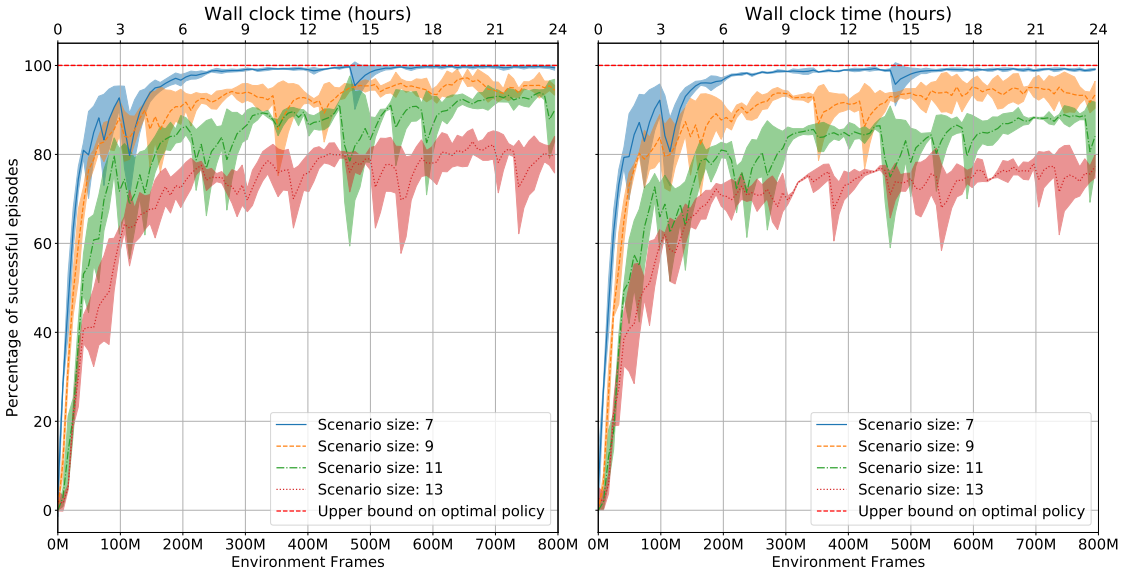


Figure 3.6: **Results from the *"Labyrinth"* scenario.** Environments of sizes $[7, 9, 11, 13]$, three experiments were conducted per size. Shown are the mean and std. of the percentage of successful episodes evaluated on the training data (left) and the test data (right). The performance of the benchmark agent diminishes as the size of the scenario increases.

decays by 1 each time-step to encourage exploration. **Reward:** +0.1 for the correct item, -0.01 per time-step.

3.4 Benchmark agent details

We apply a popular model-free policy gradient algorithm, Advantage Actor Critic (A2C) (Mnih et al. 2016c), implemented in the PyTorch framework by (Kostrikov 2018). A2C optimizes the expected discounted future return $R_t = \sum_{t'=t}^{T} \gamma^{t-t'} r_{t'}$ over trajectories of states s_t, action a_t and reward r_t triplets collected during rollouts of a policy in the environment. The algorithm achieves this by optimizing a policy $\pi(.|s_t;\theta)$ and a value function $V^{\pi}(s_t;\theta)$. Both the policy and value function are represented by neural networks with shared parameters θ that are optimized by gradient descent.

The A2C algorithm is an extension of REINFORCE (Williams 1992a) where the variance of the updates are reduced by weighting by the advantage $A(a, s_t) = R_t - V(s_t)$. The objective function that is optimized is detailed in equation 3.1, the λ factor weights the trade-off between the policy and value functions, and is typically 0.5 for most A2C implementations.

$$L(\theta) = -\log \pi(a_t|s_t;\theta) A(a_t, s_t) + \lambda (R_t - V(s_t;\theta))^2 \tag{3.1}$$

Table 3.2: **Summary of results on the training and test sets for the four scenarios in a variety of difficulty settings.**

Labyrinth			Ordered K-item		
Size	**Train**	**Test**	**# items**	**Train**	**Test**
7	99.8 ± 0.2	99.5 ± 0.2	2	0.968 ± 0.004	0.932 ± 0.012
9	98.1 ± 0.8	97.4 ± 1.3	4	0.910 ± 0.012	0.861 ± 0.015
11	95.2 ± 0.7	90.1 ± 1.0	6	0.577 ± 0.024	0.522 ± 0.023
13	84.1 ± 1.8	79.8 ± 1.9	8	0.084 ± 0.071	0.083 ± 0.070

Find Return			Two colors		
Size	**Train**	**Test**	**Complexity**	**Train**	**Test**
7	98.7 ± 0.3	95.1 ± 0.2	1	1903 ± 27	1941 ± 4
9	87.0 ± 1.9	81.6 ± 1.8	3	1789 ± 24	1781 ± 24
11	70.9 ± 1.6	64.0 ± 1.6	5	1436 ± 158	1432 ± 161
13	57.7 ± 0.8	52.9 ± 3.7	7	1159 ± 126	1128 ± 140

An optional entropy term is included to encourage exploration by penalizing peaky, low entropy distributions. We found the weighting of the entropy term to be key during hyper-parameter tuning.

The algorithm was run in a batched mode with 16 parallel agents sharing trajectories of over 128 environment steps, with discounted returns bootstrapped from value estimates for non-terminal states. The forward and backward passes were batched and computed on a GPU for increased efficiency. The downside of this setup is that the slowest simulator step will bottleneck the performance, as the chosen simulator is exceptionally fast we found in practice that this did not have a large impact. We achieved a rate of training of approximately 9,200 environment frames per second. This includes a 4 step frame skip where physics are simulated but the environment is not rendered, so on average we simulate and process 2,300 observations per second. The training efficiency was benchmarked on Xeon E5-2640v3 CPUs, with 16GB of memory and one NVIDIA GK210 GPU. The benchmark agent's network architecture is a 3 layer CNN similar to that used in (Mnih et al. 2015b) with 128 GRU to capture long term information, shown in Fig. 3.5. Three independent experiments were undertaken for each scenario and difficulty combination in order to evaluate algorithmic stability. Agents were trained for 200M observations from the environment, which is equal to 800M steps with a frame skip of 4. For the more complex variants of the scenarios we do not train the networks to convergence, as training was capped at 200M observations.

All experiments were conducted with the same hyper-parameter configuration, which was observed to be stable for all 4 custom scenarios and also the default scenarios provided with the ViZDoom environment. The hyper-parameters were chosen after a limited sweep across learning rate, gamma factor, entropy weight

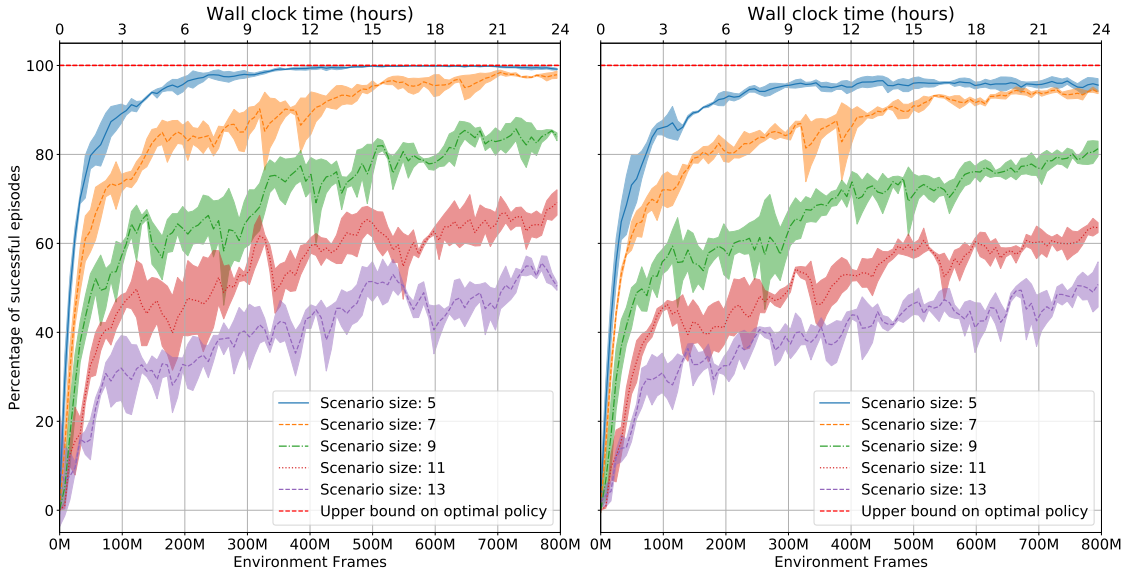


Figure 3.7: **Results from the *"Find and return"* scenario.** Maps of sizes $[5, 7, 9, 11, 13]$, three experiments were conducted per size. Displayed are the mean and std. of the percentage of successful episodes evaluated on training data (left) and test data (right).

and n-step length (the number of environment steps between network updates) on the labyrinth scenario. Weights were initialized with orthogonal initialization and the appropriate gain factor was applied for each layer.

3.5 Results and Analysis

For each scenario there are a variety of possible difficulty settings. We have explored a range of simple to challenging configuration options and evaluated the agent's performance when trained for 200 M observations from the environment.

For each possible sub-configuration, 3 independent experiments were conducted with different seeds for the weight initialization in order to get an estimate of the stability of the training process. Results of the agent's mean performance on training and test datasets are shown with errors of one standard deviation across the 3 seeds. A summary of all results is included in table 6.1. For each plot in the next section, we show a measure of the agent's performance such as average return or percentage of successful episodes on the y-axis and the number of environment frames on the x-axis.

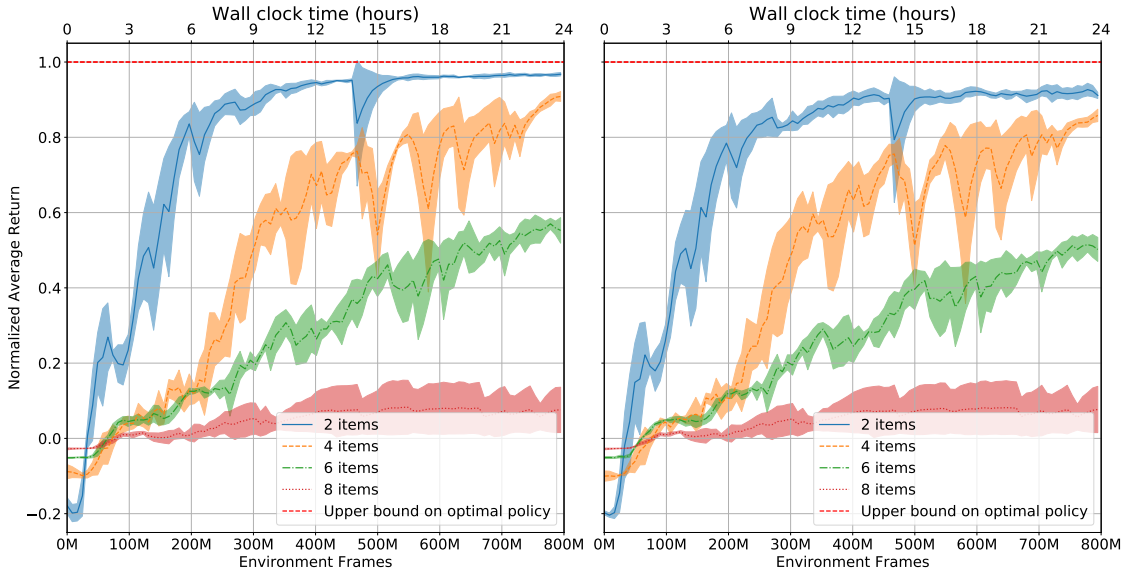


Figure 3.8: **Results from the "*Ordered k-item*" scenario.** For $k \in [2, 4, 6, 8]$, three experiments were undertaken for each k. Shown are the mean and std. of the item-normalized average return of the agent evaluated on the training (left) and testing (right) sets. The 6-item scenario is challenging and the 8-item is too difficult for the baseline agent, these are good scenarios to compare future work.

3.5.1 Results

Labyrinth: Experiments were conducted on 4 sizes of the find and return scenario ranging from 7x7 to 13x13, results of the agent's performance shown in Fig. 3.6. We observe that the performance of the benchmark plateaus near to the optimal policy for the scenarios of size 7 and 9. Reasonable performance is achieved with the size 9 scenario and performance is reduced for the size 13 scenario, which is challenging for the agent.

Find and return: Experiments were conducted on 5 sizes of the find and return scenario ranging from 5x5 to 13x13, results of the agent's performance are shown in Fig. 3.7. We observe reasonable agent performance on the smaller scenario configurations and scenarios of size 11+ are currently challenging.

Ordered k-item Experiments were conducted on 4 different scenarios of a fixed size of 5x5 with 2,4,6 and 8 items, results of the agent's performance are shown in Fig. 3.8. Here, the more critical parameter is the number K of items to be retrieved in correct order. For the 2 and 4-item scenario configurations, we observe that the policies plateau near to the optimal policy, the benchmark agent struggles with the 6-item scenario and fails on the 8-item scenario.

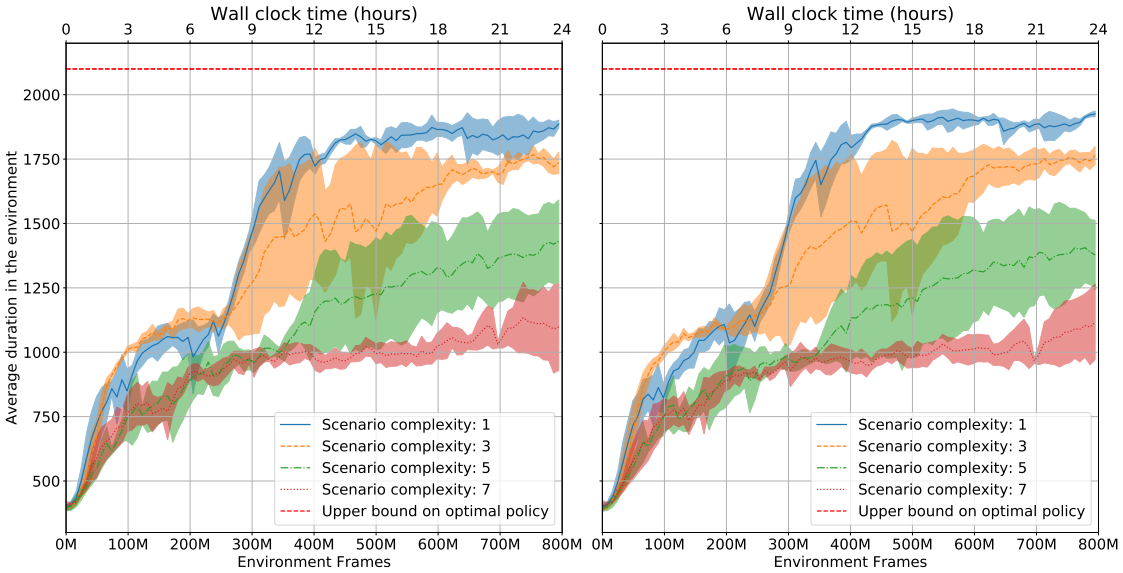


Figure 3.9: **Results from the "*Two color correlation*" scenario** for four difficulty settings, three experiments were performed for each setting. Shown are the mean and std. of the agent's average duration in the environment when evaluated on training (left) and testing (right) datasets.

Two color correlation: Experiments were conducted on 4 different scenarios of a fixed size of 5x5 with increasing environment complexity, results of the agent's performance are shown in Fig. 3.9. We observe for the less complex scenarios with 10 or 30% of the walls retained, the agent's performance plateaus near the optimal policy. The policies learned on more complex scenarios require further improvement and would be good places to test agent architectures and auxiliary losses.

3.5.2 Analysis and Discussion

To provide insight into the decision making process of the agent, we conducted an analysis of the activations in a trained agent's architecture over episodes in the two color correlation scenario. In Fig. 4.1, page 80, we show an analysis of the progression of the agent's hidden state during the first 64 steps in the environment. We observe a large change in the hidden state at step 21, further analysis indicates this is the first time the agent observes the indicator object that identifies the task objective. When the totem is observed there are many boolean flags of the hidden state that are switched on and off, which modifies the agent's behavior policy. This is an example of an agent that has learned to explore, reason and store information, purely from a weak reward signal.

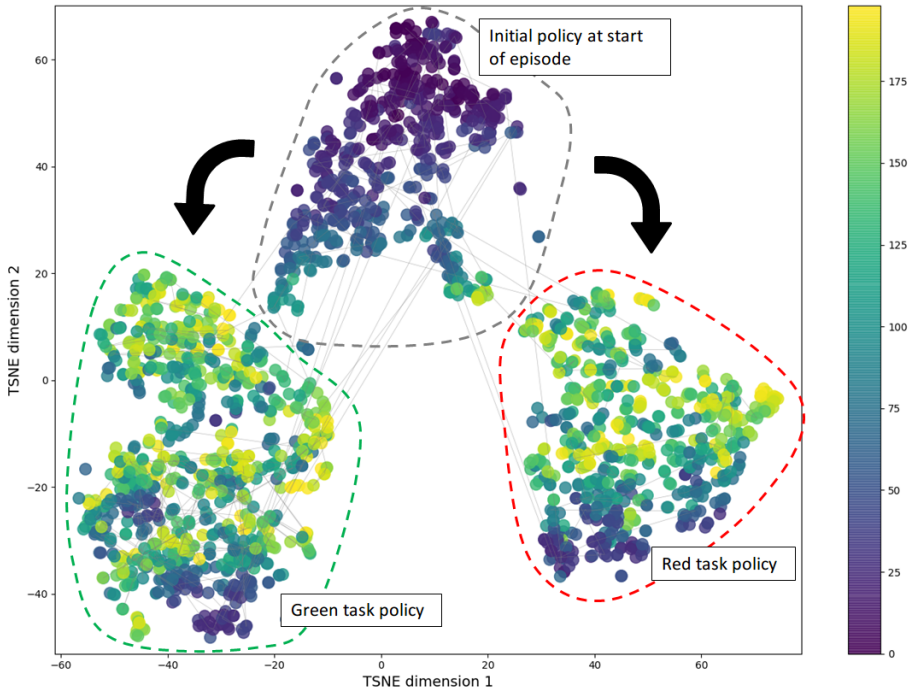


Figure 3.10: **T-SNE analysis of agent's hidden state**, sampled from 100 episodes in the "*Two color correlation*" scenario. Each point is a 2D representation of one hidden state activation, the color is the time-step, grey lines connect 20 independent trajectories. Once the indicator item has been discovered the policy transitions from an exploration policy to a one of two policies associated with the green task or the red task.

We conducted further analysis by collecting trajectories on hidden states vectors from 100 rollouts of the trained policy, dimensionality reduction was applied using T-SNE (Van Der Maaten et al. 2008) on 2000 hidden state vectors selected from a random permutation of the collected trajectories. We identify 3 separate groups of hidden state vectors, shown in Fig. 3.10. Further analysis indicated that there are three sub-policies that the agent has learned for this scenario; i) an initial policy which explores the environment to identify the color of the task, which transitions to either ii) a policy that collects red items in the case of the red task or iii) a policy that collects green items.

Challenges in spatial reasoning — the experiments on our proposed suite of benchmarks indicate that current state of the art models and algorithms still struggle to learn complex tasks, involving several different objects in a different places, and whose appearance and relationships to the task itself need to be learned from reward. The difficulty of the proposed scenarios can be adjusted, to scale the task to different levels of required reasoning. For instance, while

the ordered $K-$ items scenario is solvable for up to 4 items, it rapidly becomes challenging when the number of items or the size of the environment is increased.

3.6 Conclusions

We have presented four scenarios that require exploration and reasoning in 3D environments. The scenarios can be simulated at a rate that exceeds 12,000 FPS (frame skip=4) on average, including environment resets. Training can be done up to 9,200 FPS (frame skip=4), including forward and backward passes.

Experiments have been conducted for each scenario for a wide variety of difficulty settings. We have shown robust agent performance across random weight initialization for a fixed set of hyper-parameters. Generalization performance has been analyzed on held out test data, which demonstrates the ability of the benchmarks to generalize to unseen environment configurations that are drawn from the same general distribution. We have highlighted limitations of a typical RL baseline agent and have identified suitable scenarios for future research.

Code for the scenarios, benchmarks and training algorithms is available online[1], with detailed instructions of how to reproduce this work.

In this chapter, we have created a set of benchmark environments and quantified the limitations of an unstructured end-to-end baseline agent. In the next chapter, we will demonstrate that structured agent architecture can improve performance, sample efficiency and interpretability, in a wide variety of memory-based tasks.

CHAPTER 4

EGOMAP: PROJECTIVE MAPPING AND STRUCTURED EGOCENTRIC MEMORY FOR DEEP RL

Chapter abstract

Tasks involving localization, memorization and planning in partially observable 3D environments are an ongoing challenge in Deep Reinforcement Learning. In this chapter we present EgoMap, a spatially structured neural memory architecture. EgoMap augments a deep reinforcement learning agent's performance in 3D environments on challenging tasks with multi-step objectives. The EgoMap architecture incorporates several inductive biases including a differentiable inverse projection of CNN feature vectors onto a top-down spatially structured map. The map is updated with ego-motion measurements through a differentiable affine transform. We show this architecture outperforms both standard recurrent agents and state of the art agents with structured memory. We demonstrate that incorporating these inductive biases into an agent's architecture allows for stable training with reward alone, circumventing the expense of acquiring and labelling expert trajectories. A detailed ablation study demonstrates the impact of key aspects of the architecture and through extensive qualitative analysis, we show how the agent exploits its structured internal memory to achieve higher performance. The work in this chapter has led to the publication of conference paper:

- Edward Beeching, Jilles Dibangoye, Olivier Simonin, and Christian Wolf (2020c). "EgoMap: Projective mapping and structured egocentric memory for Deep RL". in: *Proceedings of the European Conference on Machine Learning and Knowledge Discovery in Databases (ECML-PKDD)*.

4.1 Introduction

A critical part of intelligence is navigation, memory and planning. An animal that is able to store and recall pertinent information about their environment is likely to exceed the performance of an animal whose behavior is purely reactive. Many control problems in partially observed 3D environments involve long

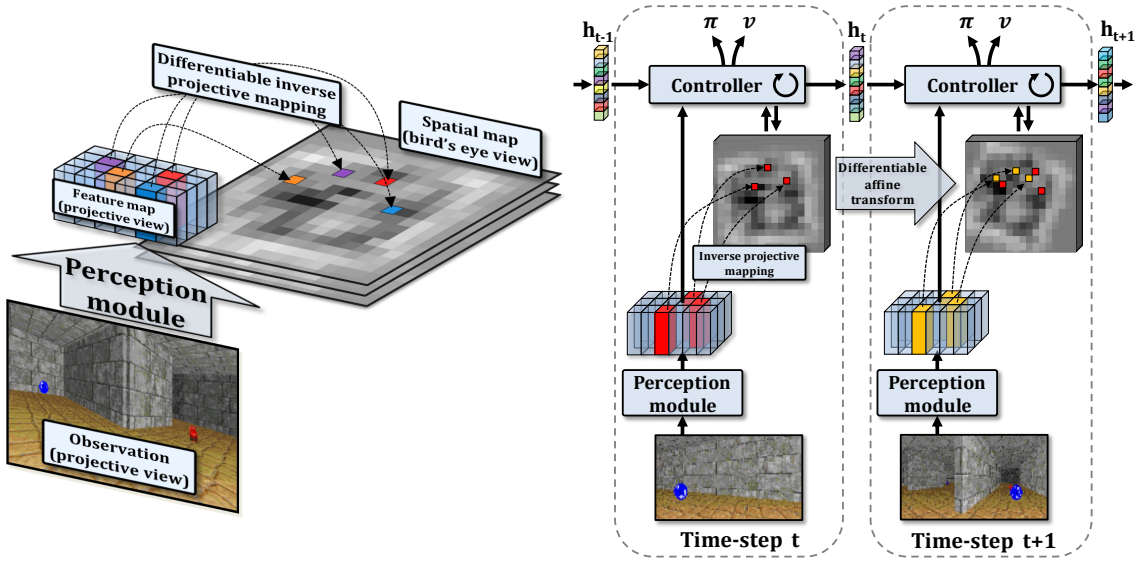


Figure 4.1: **Overview of the mapping module (left) and EgoMap agent architecture (right).** Visual features are mapped to the top-down egocentric map. Observations from a first-person viewpoint are passed through a perception module, extracted features are projected with the inverse camera matrix and depth buffer to their 3D coordinates. The operations are implemented in a differentiable manner, so derivatives can be back-propagated to update the weights of the perception module. Our model learns to unproject learned task-oriented semantic embeddings of observations and to map the positions of relevant objects in a spatially structured (bird's eye view) neural memory, where different objects in the same input image are stored in different map locations. Using the ego-motion of the agent, the map is updated by differentiable affine resampling at each step in the environment.

term dependencies and planning. Solving these problems requires agents to learn several key capacities: *spatial reasoning* — to explore the environment in an efficient manner and to learn spatio-temporal regularities and affordances. The agent needs to autonomously navigate, discover relevant objects, store their positions for later use, their possible interactions and the eventual relationships between the objects and the task at hand. Semantic mapping is a key feature in these tasks. A second feature is *discovering semantics from interactions* — while solutions exist for semantic mapping and semantic SLAM (Civera et al. 2011b; Tateno et al. 2017), a more interesting problem arises when the semantics of objects and their affordances are not supervised, but defined through the task and thus learned from reward.

A typical approach for these types of problems are agents based on deep neural networks including recurrent hidden states, which encode the relevant

information of the history of observations (Mirowski et al. 2017; Jaderberg et al. 2016a). If the task requires navigation, the hidden state will naturally be required to store spatially structured information. It has been recently reported that spatial structure as inductive bias can improve the performance on these tasks. In (Parisotto et al. 2018), for instance, different cells in a neural map correspond to different positions of the agent.

In our work, we go beyond structuring the agent's memory with respect to the agent's position. We use projective geometry as an inductive bias to neural networks, allowing the agent to structure its memory with respect to the locations of objects it perceives, as illustrated in Figure 4.1. The model performs an inverse projection of CNN feature vectors in order to map and store observations in an egocentric spatially structured memory. The EgoMap is complementary to the hidden state vector of the agent and is read with a combination of a global convolutional read operation and an attention model allowing the agent to query the presence of specific content. We show that incorporating projective spatial memory enables the agent to learn policies that exceed the performance of a standard recurrent agent. Two different objects visible in the same input image could be at very different places in the environment. In contrast to (Parisotto et al. 2018), our model maps these observations to their respective locations, and not to cells corresponding to the agent's position, as shown in Figure 4.1.

The model bears a certain structural resemblance with Bayesian occupancy grids (BOG), which have been used in mobile robotics for many years (Moravec 1989; Rummelhard et al. 2015). As in BOGs, we perform inverse projections of observations and dynamically resample the map to take into account ego-motion. However, in contrast to BOGs, our model does not require a handcrafted observation model and it learns semantics directly from interactions with the environment through reward. It is fully differentiable and trained end-to-end with backpropagation of derivatives calculated with policy gradient methods. Our contributions are as follows:

- To our knowledge, we present the first method using a differentiable SLAM-like mapping of visual features into a top-down egocentric feature map using projective geometry while training this representation using RL from reward.
- Our spatial map can be translated and rotated through a differentiable affine transform and read globally and through self-attention.
- We show that the mapping, spatial memory and self-attention can be learned end-to-end with RL, avoiding the cost of labelling trajectories from domain experts, auxiliary supervision or pre-training specific parts of the architecture.
- We demonstrate the improvement in performance over recurrent and spatial structured baselines without projective geometry.
- We illustrate the reasoning abilities of the agent by visualizing the content of the spatial memory and the self-attention process, tying it to the different objects and affordances related to the task.

- Experiments with noisy actions demonstrate the agent is robust to actions tolerances of up to 10%.

4.2 Related Work

Reinforcement learning — In recent years the field of Deep Reinforcement Learning (RL) has gained attention with successes on board games (Silver et al. 2018) and Atari games (Mnih et al. 2015c). One key component was the application of deep neural networks (Lecun et al. 1998) to frames from the environment or game board states. Recent works that have applied Deep RL for the control of an agent in 3D environments such as maze navigation are (Mirowski et al. 2017) and (Jaderberg et al. 2016a) which explored the use of auxiliary tasks such as depth prediction, loop detection and reward prediction to accelerate learning. Meta RL approaches for 3D navigation have been applied by (Wang et al. 2016a) and (Lample et al. 2017b) also accelerated the learning process in 3D environments by prediction of tailored game features. There has also been recent work in the use of street-view scenes to train an agent to navigate in city environments (Mirowski et al. 2018a). In order to infer long term dependencies and store pertinent information about the partially observable environment; network architectures typically incorporate recurrent memory such as Gated Recurrent Units (Chung et al. 2015a) or Long Short-Term Memory (Hochreiter et al. 1997b).

Differentiable memory — Differentiable memory such as Neural Turing Machines (Graves et al. 2014) and Differential Neural Computers (DNC) (Graves et al. 2016) have shown promise where long term dependencies and storage are required. Neural Networks augmented with these memory structures have been shown to learn tasks such as copying, repeating and sorting. Some recent works for control in 2D and 3D environments have included structured memory-based architectures and mapping of observations. Neural SLAM (Zhang et al. 2017c) aims to incorporate a SLAM-like mapping module as part of the network architecture, but uses simulated sensor data rather than RGB observations from the environment, so the agent is unable to extract semantic meaning from its observations. The experimental results focus on 2D environments and the 3D results are limited. Playing Doom with SLAM augmented memory (Bhatti et al. 2016a) implements a non-differentiable inverse projective mapping with a fixed feature extractor based on Faster-RCNN (Ren et al. 2015), pre-trained in a supervised manner. A downside of this approach is that the network does not learn to extract features pertinent to the task at hand as it is not trained end-to-end with RL. (Fang et al. 2019) replace recurrent memory with a transformer ((Vaswani et al. 2017a)) attention distribution over previous observation embeddings, to highlight that recurrent architectures can struggle to capture long term dependencies. The

downside is the storage of previous observations grows linearly with each step in the environment and the agent cannot chose to discard redundant information.

Grid cells — there is evidence that biological agents learn to encode spatial structure. Rats develop grid cells, which fire at different locations with different frequencies and phases, a discovery that led to the 2014 Nobel prize in medicine (O'Keefe et al. 1971; Hafting et al. 2005). A similar structure emerges in artificial neural networks trained to localize themselves, discovered independently by two different research groups (Cueva et al. 2018; Banino et al. 2018).

Projective geometry and spatial memory — Our work encodes spatial structure as additional inductive bias. We argue that projective geometry is a strong law imposed on any vision system working from egocentric observations, justifying a fully differentiable model of perception. To our knowledge, we present the first method which uses projective geometry as inductive bias while at the same time learning spatial semantic features with RL from reward.

The past decade has seen an influx of affordable depth sensors. This has led to a many works in the domain reconstruction of 3D environments, which can be incorporated into robotic systems. Seminal works in this field include (Izadi et al. 2011) who performed 3D reconstruction scenes using a moving Kinect sensor and (Henry et al. 2014) who created dense 3D maps using RGB-D cameras.

Neural Map (Parisotto et al. 2018) implements a structured 2D differentiable memory which was tested in both egocentric and world reference frames, but does not map observations in a SLAM-like manner and instead stores a single feature vector at the agent's current location. The agent's position is also discretized to fixed cells and orientation quantized to four angles (North, South, East, West). A further downside is that the movement of the memory is fixed to discrete translations and the map is not rotated to the agent's current viewpoint.

MapNet (Henriques et al. 2018b) includes an inverse mapping of CNN features, trained in a supervised manner to predict x,y position and rotation from human trajectories, but does not use the map for control in an environment. Visual Question Answering in Interactive Environments (Gordon et al. 2018) creates semantic maps from 3D observations for planning and question answering and is applied in a discrete state space.

Unsupervised Predictive Memory in a Goal-Directed Agent (Wayne et al. 2018a) incorporates a DNC in an RL agent's architecture and was applied to simulated memory-based tasks. The architecture achieves improved performance over a typical LSTM (Hochreiter et al. 1997b) based RL agent, but does not include spatial structure or projective mapping. In addition, visual features and neural memory are learned through the reconstruction of observations and actions, rather than for a specific task.

Cognitive Mapping and Planning for Visual Navigation (Gupta et al. 2017b) applies a differentiable mapping process on 3D viewpoints in a discrete grid-world, trained with imitation learning. The downside of discretization is that

Table 4.1: **Comparison of key features of related works**

Related work	Projective Geometry	Spatial map learned from reward	Reinforcement Learning	Imitation learning	Visual input	Multiple goals	Spatially Structured	Task-specific semantic features	Continuous State Space	Continuous Affine transform	End-to-end differentiable	Continuous translations	Continous rotations	End-to-end training	Intrinsic rewards
Semantic mapping (Civera et al. 2011b)	✓	✗	✗	✗	✓	✗	✓	✗	✓	✓	✗	✓	✓	✗	✗
Playing Doom with SLAM (Bhatti et al. 2016a)	✓	✗	✓	✗	✓	✗	✓	✗	✓	✓	✗	✓	✓	✗	✗
Neural SLAM (Zhang et al. 2017c)	✗	✗	✓	✗	✗	✗	✓	✓	✓	✗	✓	✗	✗	✓	✗
Cog. Map (Gupta et al. 2017b)	✗	✗	✗	✓	✓	✗	✓	✓	✗	✗	✓	✓	✗	✓	✗
Semantic SLAM (Tateno et al. 2017)	✓	✗	✗	✗	✓	✗	✓	✗	✓	✓	✗	✓	✓	✗	✗
IQA (Gordon et al. 2018)	✗	✗	✓	✓	✓	✗	✓	✗	✗	✗	✗	✗	✗	✗	✗
MapNet (Henriques et al. 2018b)	✓	✗	✗	✗	✓	✗	✓	✗	✓	✓	✓	✓	✓	✓	✗
Neural Map (Parisotto et al. 2018)	✗	✓	✓	✗	✓	✓	✓	✓	✓	✗	✓	✗	✗	✓	✗
MERLIN (Wayne et al. 2018a)	✗	✗	✓	✗	✓	✓	✗	✗	✓	✗	✓	✓	✓	✓	✗
Learning exploration policies (Chen et al. 2019a)	✓	✗	✓	✓	✓	✗	✓	✗	✓	✓	✗	✓	✓	✓	✓
EgoMap	✓	✓	✓	✗	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✗

affine sampling is trivial for rotations of 90-degree increments, and this motion is not representative of the real world. Their tasks are simple point-goal problems of up to 32 time-steps, whereas our work focused on complex multi-step objectives in a continuous state space. Their reliance on imitation learning highlights the challenge of training complex neural architectures with reward alone, particular on tasks with sparse reward such as the ones presented in this chapter.

Learning Exploration Policies for Navigation (Chen et al. 2019a), do not learn a perception module but instead map the depth buffer to a 2D map to provide a map-based exploration reward. Our work learns the features that can be mapped so the agent can query not only occupancy, but task-related semantic content.

Our work greatly exceeds the performance of Neural Map (Parisotto et al. 2018), by embedding a differentiable inverse projective transform and a continuous egocentric map into the agent's network architecture. The mapping of the environment is in the agent's reference frame, including translation and rotation with a differentiable affine transform. We demonstrate stable training with reinforcement learning alone, over several challenging tasks and random initializations, and do not require the expense of acquiring expert trajectories. We detail the key similarities and differences with related work in table 4.1.

4.3 EgoMap

We consider partially observable Markov decision processes (POMDPs) (Kaelbling et al. 1998) in 3D environments and extend recent Deep-RL models, which include

a recurrent hidden layer to store pertinent long term information (Mirowski et al. 2017; Jaderberg et al. 2016a). In particular, RGBD observations I_t at time step t are passed through a perception module extracting features s_t, which are used to update the recurrent state:

$$s_t = f_p(I_t; \theta_p) \qquad h_t = f_r(h_{t-1}, s_t; \theta_r) \tag{4.1}$$

where f_p is a convolutional neural network and f_r is a Gated Recurrent Unit (Chung et al. 2015a). Gates and their equations have been omitted for simplicity. Above and in the rest of this chapter, θ_* are trainable parameters, exact architectures are provided in the appendix. The controller outputs an estimate of the policy (the action distribution) and the value function given its hidden state:

$$\pi_t = f_\pi(h_t; \theta_\pi) \qquad v_t = f_v(h_t; \theta_v) \tag{4.2}$$

The proposed model is motivated by the regularities which govern 3D physical environments. When an agent perceives an observation of the 3D world, it observes a 2D planar perspective projection of the world based on its current viewpoint. This projection is a well understood physical process, we aim to imbue the agent's architecture with an inductive bias based on inverting the 3D to 2D planar projective process. This inverse mapping operation appears to be second nature to many organisms, with the initial step of depth estimation being well studied in the field of Physiology (Frégnac et al. 2004). We believe that providing this mechanism implicitly in the agent's architecture will improve its reasoning capabilities in new environments to bypass a large part of the learning process.

The overall concept is that as the agent explores the environment, the perception module f_p produces a 2D feature map s_t, in which each feature vector represents a learned semantic representation of a small receptive field from the agent's egocentric observation. While they are integrated into the flat (not spatially structured) recurrent hidden state h_t through function f_r (Equation 4.1), we propose its integration into a second tensor M_t, a top-down egocentric memory, which we call *EgoMap*. The feature vectors are mapped to their egocentric positions using the inverse projection matrix and depth estimates. This requires an agent with a calibrated camera (known intrinsic parameters), which is a soft constraint easily satisfied. The map can then be read by the agent in two ways: a global convolutional read operation and a self-attention operation.

Let the agent's position and angle at time t be (x_t, y_t) and ϕ_t respectively, M_t is the current EgoMap. s_t are the feature vectors extracted by the perception module, D_t are the depth buffer values. The change in agent position and orientation between time-step t and $t-1$ are $(dx_t, dy_t, d\phi_t)$.
There are three key steps to the operation of the EgoMap:
1. Transform the map to the agent's egocentric frame of reference:

$$\hat{M}_t = \text{Affine}(M_{t-1}, dx_t, dy_t, d\phi_t) \tag{4.3}$$

2. Update the map to include new observations:

$$\tilde{M}_t = \text{InverseProject}(s_t, D_t) \qquad M'_t = \text{Combine}(\hat{M}_t, \tilde{M}_t) \tag{4.4}$$

3. Perform a global read and attention based read, the outputs of which are fed into the policy and value heads:

$$r_t = \text{Read}(M'_t) \qquad c_t = \text{Context}(M'_t, s_t, r_t) \tag{4.5}$$

These three operations will be further detailed below in individual subsections. Projective mapping and structured memory should augment the agent's performance where spatial reasoning and long term recollection are required. On simpler tasks the network can still perform as well as the baseline, assuming the extra parameters do not cause further instability in the RL training process.

Affine transform — At each time-step we translate and rotate the map into the agent's frame of reference, this is achieved with a differentiable affine transform, popularized by the well known Spatial Transformer Networks (Jaderberg et al. 2015). Relying on the simulator to provide the change in position (dx, dy) and orientation $d\phi$, we convert the deltas to the agent's egocentric frame of reference and transform the map with a differentiable affine transform. The effect of noise on the change in position on the agent's performance is analysed in the experimental section.

Inverse projective mapping — We take the agent's current observation, extract relevant semantic embeddings and map them from a 2D planar projection to their 3D positions in an egocentric frame of reference. At each time-step, the agent's egocentric observation is encoded by the perception module (a convolutional neural network) to produce feature vectors, this step is a mapping from $R^{4\times64\times112} \rightarrow R^{16\times4\times10}$. Given the inverse camera projection matrix and the depth buffer provided by the simulator, we can compute the approximate location of the features in the agent's egocentric frame of reference. As the mapping is a many to one operation, several features can be mapped to the same location. Features that share the same spatial location are averaged element-wise.

The newly mapped features must then be combined with the translated map from the previous time-step. We found that the use of a momentum hyper-parameter, α, enabled a smooth blending of new and previously observed features. We use an α value of 0.9 for the tests presented in the chapter. We ensured that the blending only occurs where the locations of new projected features and the map from the previous time-step are co-located, see Equation 4.6.

$$\begin{aligned} M'^{(x,y)}_t &= \eta \hat{M}_t{}^{(x,y)} + (1-\eta)\tilde{M}^{(x,y)}_t \\ \eta &= \begin{cases} 1.0, & \text{if } \tilde{M}^{(x,y)}_t = 0 \;\&\; \hat{M}_t{}^{(x,y)} \neq 0 \\ 0.0, & \text{if } \tilde{M}_t{}^{(x,y)} \neq 0 \;\&\; \hat{M}^{(x,y)}_t = 0 \\ \alpha, & \text{otherwise} \end{cases} \end{aligned} \tag{4.6}$$

Sampling from a global map — A naive approach to the storage and transformation of the egocentric feature map would be to apply an affine transformation to the map at each time-step. A fundamental downside of applying repeated affine transforms is that at each step a bilinear interpolation operation is applied, which causes smearing and degradation of the features in the map. We mitigated this issue by storing the map in a global reference frame and mapping the agent's observations to the global reference frame. For the read operation an offline affine transform is applied.

Read operations — We wanted the agent to be able to summarize the whole spatial map and also selectively query for pertinent information. This was achieved by incorporating two types of read operation into the agent's architecture, a *Global Read* operation and a *Self-attention Read*.

The global read operation is a CNN that takes as input the egocentric map and outputs a 32-dimensional feature vector that summarizes the map's contents. The output of the global read is concatenated with the visual CNN output. To query for relevant features in the map, the agent's controller network can output a query vector q_t, the network then compares this vector to each location in the map with a cosine similarity function in order to produce scores, which are the same width and height as the map. The scores are normalized with a softmax operation to produce a soft-attention in the lines of (Bahdanau et al. 2014) and used to compute a weighted average of the map, allowing the agent to selectively query and focus on parts of the map. This querying mechanism was used in both the Neural Map (Parisotto et al. 2018) and MERLIN (Wayne et al. 2018a) RL agents. We made the following improvements: *Attention Temperature* and *Query Position*.

$$\sigma(x)_i = \frac{e^{\beta x_i}}{\sum e^{\beta x_j}} \tag{4.7}$$

Query Position: A limitation of self-attention is that the agent can query *what* it has observed but not *where* it had observed it. To improve the spatial reasoning performance of the agent we augmented the neural memory with two fixed additional coordinate planes representing the x,y egocentric coordinate system normalized to $(-1.0, 1.0)$, as introduced in (Liang et al. 2017). The agent still queries based on the features in the map, but the returned context vector includes two extra scalar quantities which are the weighted averages of the x,y planes. The impacts of these additions are discussed and quantified in the ablation study, Section 4.4.

Attention Temperature: To provide the agent with the ability to learn to modify the attention distribution, the query includes an additional learnable temperature parameter, β, which can adjust the softmax distribution detailed in Equation 4.7. This parameter can vary query by query and is constrained to be one or greater by a Oneplus function.

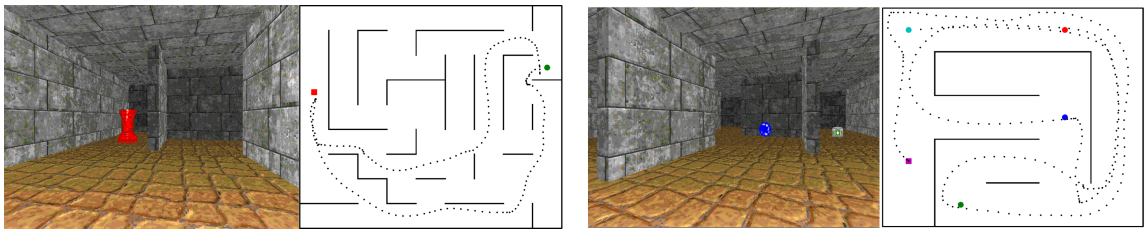


Figure 4.2: **Memory-based navigation scenarios**: Left - The "*Find and return*" scenario. With the agent's egocentric viewpoint and top down view unseen by the agent; the agent starts at the green object, finds the red object and then returns to the entry. Right - The "*Ordered K−item scenario*", where K=4. An episode where the agent collects the 4 items in the correct order.

4.4 Experiments

The EgoMap and baseline architectures were evaluated on four challenging 3D scenarios, which require navigation and different levels of spatial memory. The scenarios are taken from (Beeching et al. 2020b) who extended the 3D ViZDoom environment (Kempka et al. 2017) with various scenarios that are partially observable, require memory, spatial understanding, have long horizons and sparse rewards. Whilst more visually realistic simulators are available such as Gibson (Xia et al. 2018), Matterport (Anderson et al. 2018b), Home (Brodeur et al. 2018) and Habitat (Savva et al. 2019b), the tasks available are simple point-goal tasks which do not require long term memory and recollection. We target the following three tasks that we implemented in chapter 3:

Labyrinth: The agent must quickly find the exit, the reward is a sparse positive reward for finding the exit. This tests the ability to explore in an efficient manner.

Ordered k-item: An agent must find k objects in a fixed order. It tests three aspects of an agent: its ability to explore the environment efficiently, the ability to learn to collect items in a predefined order and its ability to store where items were located so they can be retrieved in the correct order. We tested two versions of this scenario with 4-items or 6-items, an example is shown in 4.2.

Find and return: The agent starts next to a green totem, explores the environment to find a red totem and then returns to the start. This is our implementation of "Minotaur" scenario from (Parisotto et al. 2018). The scenario tests the ability to navigate and retain information over long time periods, an example is shown in 4.2.

All the tasks require different levels of spatial memory and reasoning. For example, if an agent observes an item out of order it can store the item's location in its memory and navigate back to it later. The scenarios that require more spatial reasoning, long term planning and recollection are where the agent has the

Table 4.2: **Results of on four scenarios for 1.2 B environment steps.** We show the mean and std. for three independent experiments.

	Scenario							
	4 item		6 item		Find and Return		Labyrinth	
Agent	**Train**	**Test**	**Train**	**Test**	**Train**	**Test**	**Train**	**Test**
Random	-0.18	-0.21	-0.21	-0.21	-0.21	-0.21	-0.16	-0.09
Baseline	2.34 ± 0.03	2.27 ± 0.04	2.86 ± 0.16	2.54 ± 0.23	0.66 ± 0.00	0.63 ± 0.03	0.73 ± 0.02	0.69 ± 0.01
Neural Map	2.34 ± 0.04	2.22 ± 0.04	2.75 ± 0.06	2.47 ± 0.03	0.83 ± 0.07	0.72 ± 0.03	**0.77 ± 0.04**	0.71 ± 0.02
EgoMap	**2.40 ± 0.01**	**2.29 ± 0.02**	**3.21 ± 0.01**	**2.80 ± 0.05**	**0.89 ± 0.01**	**0.85 ± 0.02**	0.75 ± 0.00	**0.73 ± 0.02**
Optimum (Upper Bound)	2.5	2.5	3.5	3.5	1.0	1.0	1.0	1.0

greatest improvement in performance. In all scenarios there is a small negative reward for each time-step to encourage the agent to complete the task quickly.

Experimental strategy and generalization to unseen environments — Many configurations of each scenario were created through procedural generation and partitioned into separated training and testing sets of size 256 and 64 respectively for each scenario type. Although the task in a scenario is fixed, we vary the locations of the walls, item locations, and start and end points; to ensure a diverse range of possible scenario configurations. A limited hyper-parameter sweep was undertaken with the baseline architecture to select the hyper-parameters, which were fixed for both the baseline, Neural Map and EgoMap agents. Three independent experiments were conducted per task to evaluate the algorithmic stability of the training process. To avoid information asymmetry, we provide the baseline agent with $dx, dy, sin(d\theta), cos(d\theta)$ concatenated with its visual features.

Training Details — The model parameters were optimized with an on-policy, policy gradient algorithm; batched Advantage Actor Critic (A2C) (Mnih et al. 2016a), we used the PyTorch (Paszke et al. 2017) implementation of A2C (Kostrikov 2018). We sampled trajectories from 16 parallel agents and updated every 128 steps in the environment with discounted returns bootstrapped from value estimates for non-terminal states. The gamma factor was 0.99, the entropy weight was 0.001, RMSProp (Hinton et al. 2012b) was used with a learning rate of 7e-4. The EgoMap agent map size was 16×24×24 with a grid sampling chosen to cover the environment size with a 20% padding. The agent's policy was updated over 1.2B environment steps, with a frame skip of 4. Training took 36 hours for the baseline and 8 days for the EgoMap, on 4 Xeon E5-2640v3 CPUs, with 32GB of memory and one NVIDIA GK210 GPU.

Results — Results from the baseline and EgoMap policies evaluated on the 4 scenarios are shown in table 4.2, all tasks benefit from the inclusion of inverse projective mapping and spatial memory, with the largest improvement on the *Find and Return* scenario. We postulate that the greater improvement in performance is due to two factors; firstly this scenario always requires spatial memory as the agent must return to its starting point and secondly the objects in this scenario are larger and occupy more space in the map. We also compared to the state of

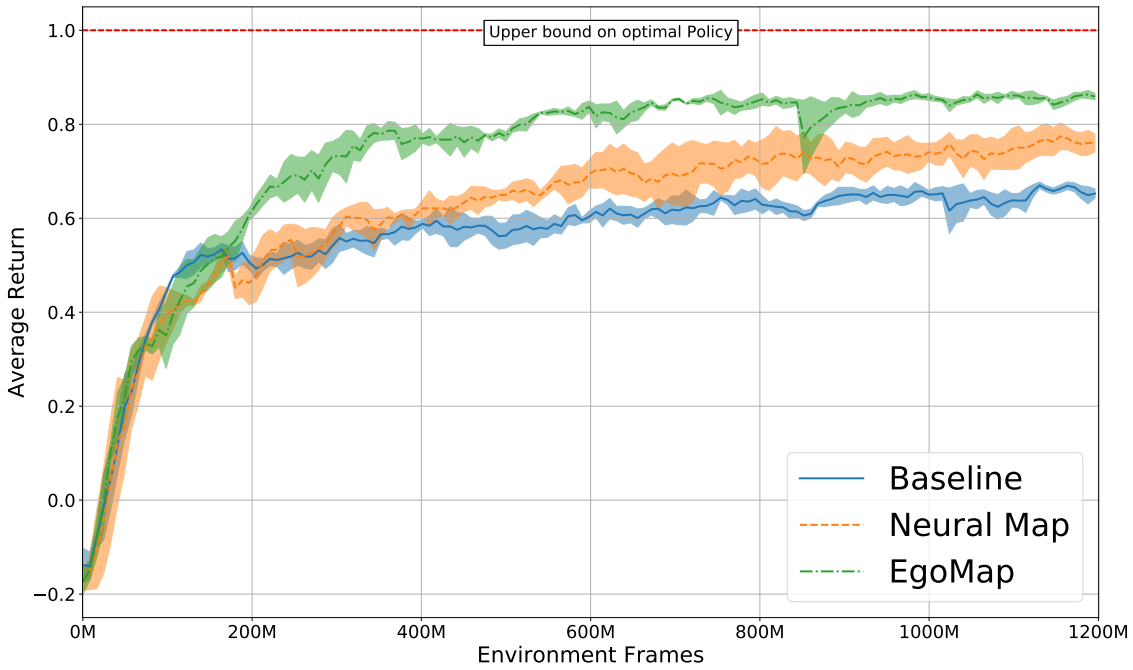


Figure 4.3: **Training curves for the *Find and Return* test set.**

the art in spatially structured neural memory, Neural Map (Parisotto et al. 2018). Figure 4.3 shows agent training curves for the recurrent baseline, Neural Map and EgoMap, on the *Find and Return* test set configurations.

Ablation study — An ablation study was carried out on the improvements made by the EgoMap architecture. We were interested to see the influence of key options such as the global and attention-based reads, the similarity function used when querying the map, the learnable temperature parameter and the incorporation of location-based querying. The Cartesian product of these options is large and it was not feasible to test them all, we therefore decided to selectively switch off key options to understand which aspects contribute to the improvement in perfor-

Table 4.3: **Egomap ablation study**, *Find and Return* scenario, conducted after 800M steps.

Ablation	Train	Test
Baseline	0.668 ± 0.028	0.662 ± 0.036
No global read	0.787 ± 0.007	0.771 ± 0.029
No query	0.838 ± 0.003	0.811 ± 0.013
No query temperature	0.845 ± 0.014	0.815 ± 0.019
No query position	0.839 ± 0.007	0.814 ± 0.008
EgoMap	0.847 ± 0.011	0.814 ± 0.017
EgoMap (L1 query)	**0.851 ± 0.014**	**0.828 ± 0.011**

mance. The results of the ablation study are shown in Table 4.3. Both the global and self-attention reads provide large improvements in performance over the baseline recurrent agent. The position-based query provides a small improvement. A comparison of the similarity metric of the attention mechanism highlights the L1-similarity achieved higher performance than cosine. A qualitative analysis of the self-attention mechanism is shown in the next section.

4.5 Analysis

Visualization — The EgoMap architecture is highly interpretable and provides insights about how the agent reasons in 3D environments. In Figure 4.4 we show analysis of the spatially structured memory and how the agent has learned to query and self-attend to recall pertinent information. The Figure shows key steps during an episode in the *Ordered 6-item* and *Find and Return* scenarios, including the first three principal components of the 16-dimensional EgoMap, the attention distribution and the vector returned from position queries. Refer to the caption for further details. The agent is seen to attend to key objects at certain phases of the task, in the *Ordered 6-item* scenario the agent attends the next item in the sequence and in the *Find and Return* scenario the agent attends to the green totem located at the start/return point once it has found the intermediate goal. This demonstrates that augmenting an agent with spatially structured memory not only improves performance but also increases interpretability of the agent's decision making process.

Noisy Actions — One common criticism of agents trained in simulation is that the agent can query its environment for information that would not be readily available in the real world. In the case of EgoMap, the agent is trained with ground truth ego-motion measurements. Real-world robots have noisy estimates of ego-motion due to the tolerances of available hardware. We performed an analysis of the EgoMap agent trained in the presence of a noisy oracle, which adds noise to the ego-motion measurements. Noise is drawn from a normal distribution centered at one and is multiplied by the agent's ground-truth motion, the effect of the noise is cumulative but unbiased. Tests were conducted with standard deviations of up to 0.2 which is a tolerance of more than 20% on the agent's ego-motion measurements, results are shown in Figure 4.5. We observed that the agent retains the performance increase over the baseline for up to 10% of noisy actions, the performance degrades to that of the baseline agent. This demonstrates the EgoMap architecture learns to be robust to noisy odometry.

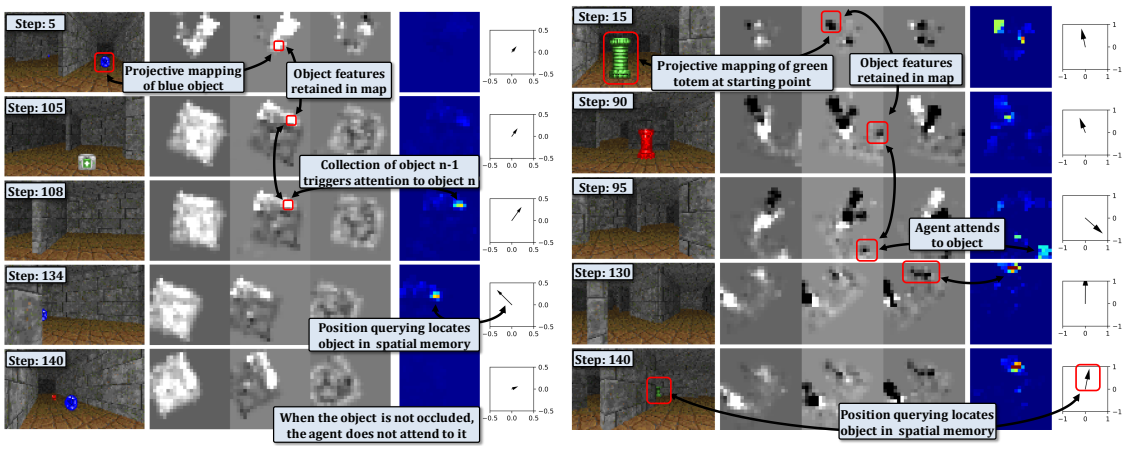
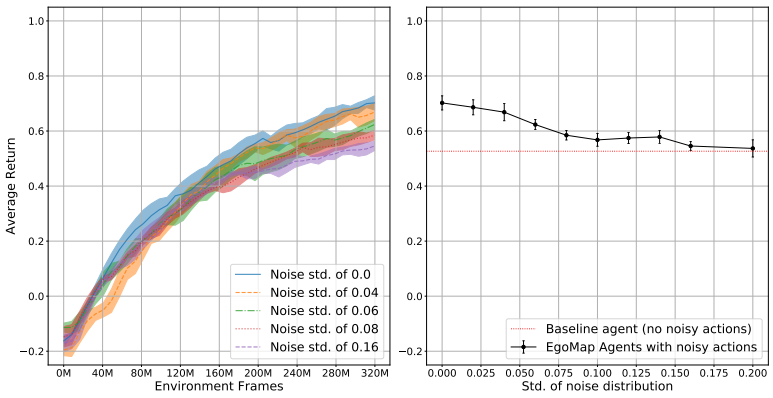


Figure 4.4: **Analysis for key steps (different rows) during an episode from the *Ordered 6-item* (left) and *Find and Return* (right) scenarios.** Within each sub-figure: Left column - RGB observations, central column - the three largest PCA components of features mapped in the spatially structured memory, right - attention heat map and x,y query position vector. The agent maps and stores features from key objects and attends to them when they are pertinent to the current stage of the task. For example, for the left figure on the first row at time-step 5 the blue spherical object, which is ordered 4 of 6, is mapped into the agent's spatial memory. The agent explores the environment collecting the items in order, it collects item 3 of 6 between time-step 105 and 108, shown on rows 2 and 3. As soon as the agent has collected item 3 it queries its internal memory for the presence of item 4, which is shown by the attention distribution on rows 3 and 4. On the last row, time-step 140, the agent observes the item and no longer attempts to query for it, as the item is in the agent's field of view.

4.6 Conclusion

We have presented EgoMap, an egocentric spatially structured neural memory that augments an RL agent's performance in 3D navigation, spatial reasoning and control tasks. EgoMap includes a differentiable inverse projective transform that maps learned task-specific semantic embeddings of agent observations to their world positions. We have shown that through the use of global and self-attentive read mechanisms an agent can learn to focus on important features from the environment. We demonstrate that an RL agent can benefit from spatial memory, particularly in 3D scenarios with sparse rewards that require localization and memorization of objects. EgoMap out-performs existing state of the art baselines, including Neural Map, a spatial memory architecture. The increase in performance compared to Neural Map is due to two aspects. 1) The differential projective



Noise std.	Average return
0.00	0.702 ± 0.026
0.02	0.686 ± 0.027
0.04	0.669 ± 0.031
0.06	0.623 ± 0.018
0.08	0.585 ± 0.017
0.10	0.568 ± 0.023
0.12	0.575 ± 0.020
0.14	0.578 ± 0.023
0.16	0.546 ± 0.017
0.20	0.537 ± 0.031
Baseline	0.527 ± 0.047

Figure 4.5: **Test set performance of the EgoMap agent during training with noisy ego-motion measurements**, conducted for 320 M environment frames. Shown are test set performance during training (left), final performance for a range of noise values (centre) which are tabulated (right). We retain an improvement in performance over the baseline agent for noisy actions of up to 10%.

transform maps *what* the objects are to *where* they are in the map, which allows for direct localization with attention queries and global reads. In comparison, Neural Map writes *what* the agent observes to *where* the agent is on the map, this means that the same object viewed from two different directions will be written to two different locations on the map, which leads to poorer localization of the object. 2) Neural Map splits the map to four 90-degree angles, which alleviates blurring, our novel solution to this issue stores a single unified map in an allocentric frame of reference and performs an offline egocentric read, which allows an agent to act in states spaces where the angle is continuous, without the need to quantize the agent's angle to 90-degree increments.

We have shown, with detailed analysis, how the agent has learned to interact with its structured internal memory. The ablation study has the benefits of the global and self-attention operations and that these can be augmented with temperature, position querying and other similarity metrics. We have demonstrated that the EgoMap architecture is robust to actions with tolerances of up to 10%. Future work in this area of research would evaluate the architecture in situations with noisy depth measurements and in a larger variety of simulators such as MALMO (Johnson et al. 2016a), DeepMind Lab (Beattie et al. 2016a) and Habitat (Savva et al. 2019b).

The focus of this chapter has been on the incorporation of spatially structured metric maps and differentiable inverse projective geometry. In the next chapter, we explore an irregular topological map structure and look to construct neural network architectures to perform an approximation of a classical planning algorithm.

CHAPTER 5

LEARNING TO PLAN WITH UNCERTAIN TOPOLOGICAL MAPS

Chapter abstract

In this chapter we train an agent to navigate in 3D environments using a hierarchical strategy including a high-level graph based planner and a local policy. Our main contribution is a data driven learning based approach for planning under uncertainty in topological maps, requiring an estimate of shortest paths in valued graphs with a probabilistic structure. Whereas classical symbolic algorithms achieve optimal results on noise-less topologies, or optimal results in a probabilistic sense on graphs with probabilistic structure, we aim to show that machine learning can overcome missing information in the graph by taking into account rich high-dimensional node features, for instance visual information available at each location of the map. Compared to purely learned neural white box algorithms, we structure our neural model with an inductive bias for dynamic programming based shortest path algorithms, and we show that a particular parameterization of our neural model corresponds to the Bellman-Ford algorithm. By performing an empirical analysis of our method in simulated photo-realistic 3D environments, we demonstrate that the inclusion of visual features in the learned neural planner outperforms classical symbolic solutions for graph based planning.
The work in this chapter has led to the publication of a conference paper:

- Edward Beeching, Jilles Dibangoye, Olivier Simonin, and Christian Wolf (2020d). "Learning to plan with uncertain topological maps." In: *Proceedings of the IEEE European conference on computer vision (ECCV) - (**spotlight** presentation).*

5.1 Introduction

It has been shown that humans and other animals navigate through the use of waypoints combined with a local locomotion policy (Wang et al. 2002; Gupta et al. 2017c). In this work, we mimic this strategy by proposing a hierarchical planner, which performs high-level long term planning using an uncertain topological map (a valued graph including visual features) combined with a local RL-based policy

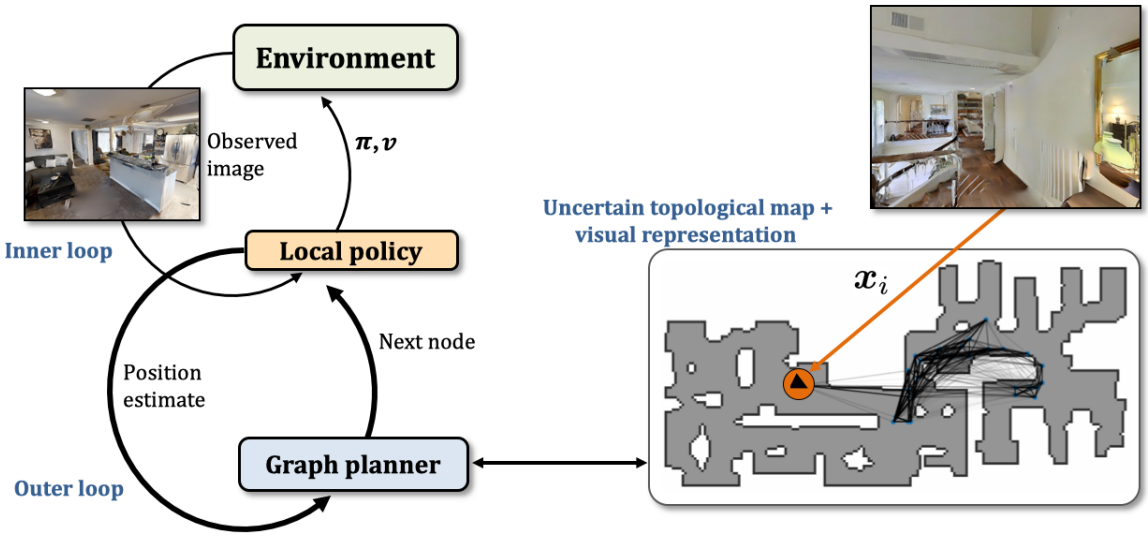


Figure 5.1: **An agent navigates to a goal location with a hierarchical planner.** A high-level planner proposes target nodes in a topological map, which are used as an objective for a local point-goal policy. The graph is estimated from an explorative rollout and, as such, uncertain: the opacities of the edges correspond to estimations of connectivity between nodes (darker lines = higher confidence). We observe a low probability of connection between the node at the agent's position and its nearest neighbor, whereas from the visual observation associated to the node we can see there is a traversable space between the two nodes.

navigating between high-level waypoints proposed by the graph planner. Our main contribution is a way to combine symbolic planning with machine learning, and we look to structure a neural network architecture to incorporate landmark based planning in unseen 3D environments.

When solving visual navigation tasks, biological or artificial agents require an internal representation of the environment if they want to solve more complex tasks than random exploration. We target a scenario where an agent is trained on a large-scale set of 3D environments to learn to reason on planning and navigation. When faced with a previously unseen environment, the agent is given the opportunity to build a representation by doing an explorative rollout from a previously learned explorative policy. It can then exploit this internal representation in subsequent visual navigation tasks. This corresponds to many realistic situations, where robots are deployed to indoor environments and are allowed to familiarize themselves before performing their tasks (Savinov et al. 2018b).

Our agent constructs an imperfect topological map of its environment, where nodes correspond to places and valued edges to connections. Edges are assigned

two different values, spatial distances and probabilities indicating whether it is possible to navigate between the two nodes. Nodes are also assigned rich visual features extracted from images taken at the corresponding places in the environment. After deployment, the agent faces visual navigation tasks requiring it to find a specific location in the environment provided by a set of images corresponding to different viewpoints, extending the task proposed in (Zhu et al. 2017). The objective is to identify the goal location in the internal representation, and to provide an estimate for the shortest path to it. The main difficulty we address here is the fact that this path is an estimate only, since the ground truth path is not available during testing.

Whilst planning in graphs with known connectivity has been solved for many decades (Dijkstra 1959; Bellman 1958), planning under uncertainty remains an ongoing area of research. Whereas optimal results in a probabilistic sense exist for graphs with probabilistic connectivity, we aim to show that machine learning can overcome missing information in the graph by taking into account rich high-dimensional node features extracted from image observations associated with specific nodes. We train a graph neural network in a fully supervised way to predict estimates of the shortest path, using vision to overcome uncertainty in the connectivity information. We present a new variant of graph neural networks imbued with specific inductive bias, and we show that this structure can be parameterized to fallback to the classical Bellman-Ford algorithm.

Figure 5.1 illustrates the hierarchical planner: a neural graph based planner runs an outer loop providing estimates for next way-point on a graph, which are used as target nodes for a local RL-based policy running an inner loop and providing feedback to high-level planner on reached locations. Both planners take into account visual features, either stored in the graph (graph based planner), or directly as observations provided by the environment (local policy). The two planners are trained separately — the graph based planner in a fully supervised way from ground truth graphs, the local policy with RL and a point-goal strategy. This work makes the following contributions:

- A hierarchical model combining high-level graph based planning with a local point goal policy for robot navigation;
- A neural planner that combines an uncertain topological map with node features to learn to estimate shortest paths in noisy and unknown environments.
- A variant of graph networks encoding inductive bias inspired by dynamic programming-based shortest path algorithms.
- We evaluate this method in challenging and visually realistic 3D environments and show it outperforms symbolic planning on noisy topological maps.

5.2 Related work

Classical planning and graph search — A large body of work is available on classical planning on graphs, notable references include (LaValle 2006; Remolina et al. 2004). In robotics, there have been a number of works applying classical planning in topological maps for indoor robot navigation, for instance (Shatkay et al. 1997; Thrun 1998).

Planning under imperfect information — In many realistic robotic problems, the current state of the world is unknown. Though sensor observations provide measurements about the current state of the world, these measurements are usually incomplete or noisy because of disturbances that distort their values. Planning problems that face these issues are referred to as planning problems under imperfect information. Research on this topic has a long history, starting with the seminal work by (Åström 1965) presenting the first non-trivial exact dynamic programming algorithm for partially observable Markov decision processes (POMDPs). While there are other models LaValle 2006, chap 12, POMDPs emerged as the standard framework to formalize and solve (single-agent) sequential decision-making problems with imperfect information about the state of the world (Kaelbling et al. 1998). As the agent does not have access to the actual state of the world, it acts based solely on its entire history of actions and observations, or the corresponding belief state, *i.e.,* the posterior probability distribution over the states given the history (Åström 1965; Smallwood et al. 1973). Approaches for finding optimal solutions have been investigated in the 2000s, ranging from dynamic programming (Kaelbling et al. 1998) to heuristic search methods (Smith et al. 2004; Kurniawati 2008). Key to these approaches is the idea that one can recast the original problem into a continuous-state fully observable Markov decision process, where states are belief states or histories (Åström 1965). Doing so allows theory and algorithm that applies for MDPs to also apply to POMDPs, albeit in much larger (and possibly continuous) state space. Another significant result of this literature is proof that the optimal value function is a piece-wise linear and convex function of the belief states, which allows the design of algorithms with faster rates of convergence (Smallwood et al. 1973). For a thorough discussion on existing solvers for POMDPs, the reader can refer to (Shani et al. 2013).

Deep Reinforcement Learning — The field of Deep Reinforcement Learning (RL) has gained attention with successes on board games (Silver et al. 2018) and Atari games (Mnih et al. 2015c). Recent works have applied Deep RL for the control of an agent in 3D environments (Mirowski et al. 2017) (Jaderberg et al. 2016a), exploring the use of auxiliary tasks such as depth prediction, loop detection and reward prediction to accelerate learning. Other recent work uses street-view scenes to train an agent to navigate in city environments (Mirowski et al. 2018a). To infer long term dependencies and store pertinent information about the partially observable environment, network architectures typically incorporate

recurrent memory such as Gated Recurrent Units (Chung et al. 2015a) or Long Short-Term Memory (Hochreiter et al. 1997b). Extensions to memory based neural approaches began with Neural Turing Machines (Graves et al. 2014) and Differentiable Neural Computers (Graves et al. 2016), and have since been adapted to expand the capacity of Deep RL agents (Wayne et al. 2018a). Spatially structured memory architectures have been shown to augment an agent's performance in 3D environments and are broadly split into two categories: metric maps which discretize the environment into a grid based structure and topological maps which produce node embeddings at key points in the environment. Research in learning to use a metric map is extensive and includes spatially structured memory (Parisotto et al. 2018), Neural SLAM based approaches (Zhang et al. 2017c) and approaches incorporating projective geometry and neural memory (Gupta et al. 2017b; Bhatti et al. 2016a), these techniques are combined, extended and evaluated in (Beeching et al. 2020c). Other works include that of Value Iteration Networks (VIN) (Tamar et al. 2016) which approximate the value iteration algorithm with a CNN, applied planning in small fully observable state spaces (grid worlds). While VIN and our work structure planners, VINs use convolutions to approximate classical value iteration, while we use a graph representation and a novel GNN architecture with recurrent updates to approximate the Bellman-Ford algorithm. (Karkus et al. 2017) plans under uncertainty in partially observable gridworld environments. Here uncertainty refers to POMPs, the classical QMDP algorithm is used as inductive bias for a neural network, whereas in our work uncertainty is over node connectivity in a graph constructed in a previously unseen environment. (Srinivas et al. 2018) is applied in observable state spaces to learn a forward model in a latent space to plan appropriate actions; they are not hierarchical, are not graph-based and do not appear to plan under uncertainty.

Research combining learning, navigation in 3D environments and topological representations has been limited in recent years with notable works being (Savinov et al. 2018b) who create graph a through random exploration in ViZDoom RL environment (Kempka et al. 2017). (Eysenbach et al. 2019b) performs planning in 3D environments on a graph-based structure created from randomly sampled observations, with node distances estimated with value estimates. The downside of these approaches is that to generalize to an unseen environment, many random samples must be taken to populate the graph.

Graph neural networks — Graph Neural Networks (GNN) are deep networks that operate on graphs directly. They have recently shown great promise in domains such as knowledge graphs (Schlichtkrull et al. 2018), chemical analysis (Gilmer et al. 2017), protein interactions (Fout et al. 2017), physics simulations (Battaglia et al. 2016) and social network analysis (Kipf et al. 2017). These types of architectures enable learning from both node features and graph connectivity. Several review papers have covered graph neural networks in great detail (Bronstein et al. 2017; Battaglia et al. 2018a; Wu et al. 2019; Zhou et al. 2018). GNNs

have been applied to shortest path planning in travelling salesmen problems (Li et al. 2018; Joshi et al. 2019) and it has been reasoned that they can approximate optimal symbolic planning algorithms such as the Bellman-Ford algorithm (Xu et al. 2019b). This work applies a novel variant of GNN in order to solve approximate planning problems, where classical methods may struggle to deal with uncertainty.

5.3 Hierarchical navigation with uncertain graphs

We train an agent to navigate in a 3D visual environment and to exploit an internal representation, which it is allowed to obtain from an explorative rollout before the episode. Our objective is image goal, i.e. target-driven navigation to a location which is provided through a (visual) image. We extend the task introduced in (Zhu et al. 2017) by generalizing to unseen environment configurations without the need to retrain the agent for a novel environment.

From the explorative rollout obtained with an agent trained with RL, which is further described in section 5.3.3, we create an uncertain topological map covering the environment, i.e. a valued graph $\mathcal{G}=\{\mathcal{V}, \boldsymbol{V}, \boldsymbol{E}, \boldsymbol{L}, \boldsymbol{D}\}$, where $\mathcal{V}=\{1, \dots N\}$ is a set of nodes, $\boldsymbol{V}$ is a $K \times N$ matrix of rich visual node features of dimensions K, $\boldsymbol{E} \in [0,1]^{N \times N}$ is a set of edge probabilities where $\boldsymbol{E}_{i,j}$ is the probability of having an edge between nodes i and j, $\boldsymbol{L}$ is a matrix of node locations and $\boldsymbol{D}$ is a distance matrix, where $\boldsymbol{D}_{i,j}$ is a distance between nodes i and j. While $\boldsymbol{D}$ encodes a distance in a path planning sense, $\boldsymbol{E}$ encodes the probability of j being directly accessible from i with obstructions. The uncertainty encoded by this probability can be considered to be a combination of aleatory variability, i.e. uncertainty associated with natural randomness of the environment, as well as epistemic uncertainty, i.e. uncertainty associated with variability in computational models for estimating the graph, in our case the explorative policy trained with RL and taking into account visual observations.

Once the topological map is obtained, the objective of the agent at each episode is to navigate to a location given an image, which is provided as additional observation at each time step. The agent acts in 3D environments like Habitat(Savva et al. 2019a) (see section 6.6), receiving images of the environment as observations and predicting actions from a discrete space (*forward*, *turn left 10 °*, *turn right 10 °*). We propose a hierarchical planner performing actions at two different levels:

A HIGH-LEVEL GRAPH BASED PLANNER that operates on longer time scale τ and iteratively proposes new point-goals nodes p_g^τ that are predicted, by a Graph Neural Network, to be on the shortest path from the agent to the estimated location of the target image.

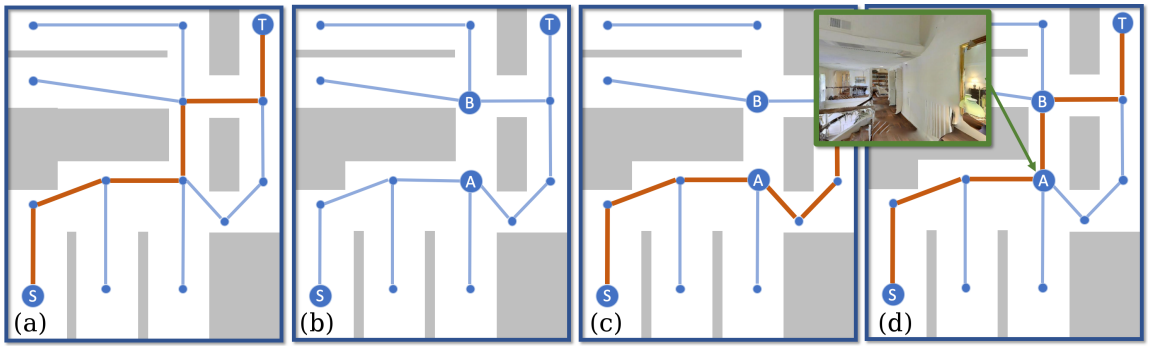


Figure 5.2: **Illustration of the different types of solutions to the high-level graph planning problem**: (a) the ground truth graph (unavailable during testing) with the shortest path from node S to node T in red; (b) the uncertain graph available during test time. This graph is fully connected and for each edge a connection probability is available. For clarity we here show only edges where the connection probability is above a threshold. The edge from $A \rightarrow B$ is wrongly estimated as not connected; (c) an "optimal" path taking into account both probabilities and distances; (d) A learned shortest path, where the visual features at node A indicate passage to node B. We supervise a network to predict the GT path (a).

A LOCAL POLICY that has been trained to navigate to a local point-goal p_g^τ, which has been provided by the high-level policy. The local policy operates for a maximum of m time-steps, where m is a hyper-parameter, set to 10. The agent has been trained with an additional *STOP* action, so that it can learn to terminate the local policy in the case that it reaches p_g^τ in under m steps.

The two planners communicate through estimated locations, the graph planner indicating the next waypoint to the local policy as a location, and the local policy providing an estimate of its location back to the high-level planner. The planner updates its current node estimate as the nearest node and planning continues.

5.3.1 High-level planning with uncertain graphs

The objective of the high-level planner is to estimate the shortest path from the current position $S \in \mathcal{V}$ in the graph to a terminal node $T \in \mathcal{V}$, whose identity is estimated as the node whose visual features are closest to the target image in cosine distance. Planning takes into account the distances between nodes encoded in $\boldsymbol{D}$ as well as estimated edge connectivity encoded in $\boldsymbol{E}$. As an edge (i, j) may have a large connection probability $\boldsymbol{E}_{i,j}$ but still be obstructed in reality, the goal is to learn a trainable planner parameterized by parameters θ, which takes into account visual features $\boldsymbol{V}$ to overcome the uncertainty in the graph connectivity.

To this end, we assume the ground truth connectivity $\boldsymbol{E}^*$ available during training only. Figure 5.2 illustrates the different types of solutions this problem admits: the optimal shortest path is only available on ground truth data (Figure 5.2a), the objective is to use the noisy uncertain graph (Figure 5.2b) and provide an estimate of the optimal solution taking into account visual features (Figure 5.2d). This is unlike the optimal solution in a probabilistic sense calculated from a symbolic algorithm (Figure 5.2c).

We propose a trainable planner, which consists of a novel graph neural network architecture with dedicated inductive bias for planning. Akin to graph networks (Battaglia et al. 2018b), the node embeddings are updated with messages over the edges, which propagate information over the full graph. While it has been shown that graph networks can be trained to perform planning (Xu et al. 2019a), we aim to closely mimic the structure of the Bellman-Ford algorithm and we embue the planner with additional inductive bias and a supervised objective to explicitly learn to calculate shortest paths from data. To this end, each node i of the graph is assigned an embedding $\boldsymbol{x_i} = [\boldsymbol{v_i}, \boldsymbol{e_i}, t_i, \boldsymbol{d_i}, \boldsymbol{s_i}]$ where $\boldsymbol{v_i}$ are visual features from the memory matrix $\boldsymbol{V}$, t_i is a boolean value indicating if the node is the target, $\boldsymbol{e_i}$ are the edge connection probabilities from node i to all other nodes, $\boldsymbol{d_i}$ are the distances for node i to all other nodes, $\boldsymbol{s_i}$ is a one hot vector identifying the node.

We motivate our neural model with the following objective: the planner should be able to exploit information contained in the graph connectivity, but also in the visual features, to be able to find the shortest path from a given current node to a target node. As with classical planning algorithms, it will thus be required to keep for each node a latent representation of the bound d_i on the shortest distance as well as information on the identity of the outgoing edge to the neighbor on the shortest path, the predecessor function $\Pi(i)$. Known algorithms (Dijstra, Bellman-Ford) perform iterative updates of these variables (d_i, Π_i) by comparing them with neighboring nodes and their inter-node distances, updating the bound d_i and Π_i when a shorter path is found than the current one. This is usually done by iterating over the successors of a given node i.

In our trained model, these variables are not made explicit, but they are supposed to be learned as a unique vectorial latent representation for each node i in the form of an internal state $\boldsymbol{r_i}$, which generally holds current information on the reasoning of the agent. The input to each iteration of the graph network is, for each node i, the node embedding $\boldsymbol{x_i}$, and the node state $\boldsymbol{r}_i$, which we concatenate to form a single node vector $\boldsymbol{n_i} = [\boldsymbol{x}_i, \boldsymbol{r}_i] = [\boldsymbol{v_i}, \boldsymbol{e_i}, t_i, \boldsymbol{d_i}, \boldsymbol{s_i}, \boldsymbol{r_i}]$. As classically done in graph neural networks, this representation is updated iteratively by exchanging messages between nodes in the form of trainable functions. The messages and

trainable functions of our model are given as follows, illustrated in Figure 5.3, and will be motivated in detail further below.

$$\boldsymbol{m}_{i,j} = \boldsymbol{W}_1[\boldsymbol{n}_i, \boldsymbol{n}_j] \odot \sigma(\boldsymbol{W}_2[\boldsymbol{n}_i, \boldsymbol{n}_j]) \tag{5.1}$$

$$\boldsymbol{r}'_i = \phi^{r \leftarrow h}(\{\boldsymbol{m}_{i,j}\}_{\forall j}, \boldsymbol{h}_i) \tag{5.2}$$

Here, $\odot$ is the Hadamard product, $\boldsymbol{W}_\cdot$ are weight matrices, and $\boldsymbol{r}'_i$ is the updated latent representation. The features $\boldsymbol{x}_i$ do not change during these operations.

Equation (5.1) is inspired from gated linear layers (Dauphin et al. 2017), and enables each node to identify whether it is the target, and update its representation of the bound. We use gated linear layers in order to provide the network with the capacity to update bound estimates for its neighbors.

Equation (5.2) integrates messages from all neighbors j of node i, updating its latent representation. Since planning requires this step to update internal bounds on shortest paths, akin to shortest path algorithms that rely on dynamic programming, we serialize the updates from different neighbors into a sequence of updates, which allows the network to learn to calculate minimum functions on bound estimates. In particular, we model this through a Gated Recurrent Unit (Chung et al. 2014), using a hidden state vector $\boldsymbol{h}_i$ associated to each node i. The step is structured to mimic the min operation of the Bellman-Ford algorithm (see section 5.3.2 for details on this equivalence).

Equation (5.2) can thus be rewritten in more detail as follows: Going sequentially over the different neighbors j of node i, the hidden state $\boldsymbol{h}_i$ is updated as follows:

$$\boldsymbol{h}_i^{[j]} = \boldsymbol{W}_3 \boldsymbol{m}_{i,j} + \boldsymbol{W}_4 \boldsymbol{h}_i^{[j-1]} \tag{5.3}$$

For simplicity, we omitted the gating equations of GRUs and presented a single layer GRU. In practice we include all gating operations and use a stacked GRU with two layers. The output of the recurrent unit is a non-linear function of the last hidden state, providing the new latent value $\boldsymbol{r}'_i = MLP(\boldsymbol{h}_i^N)$.

The above messages are exchanged and accumulated for k steps where k is a hyper-parameter which should be at least the largest span of the graphs in the dataset. The action distribution is then estimated for each node as a linear mapping of the node embeddings followed by a softmax activation function.

5.3.2 Relations to optimal symbolic planners

As mentioned before, our neural planner could in theory be instantiated with a specific set of network parameters such that it corresponds to a known symbolic planner calculating an optimal path in a certain sense. To illustrate the relationship of the network structure, in particular the recurrent nature of the graph updates, we will layout details for the case where the planner performs the estimation of a shortest path given the distance matrix and ignoring the uncertainty information

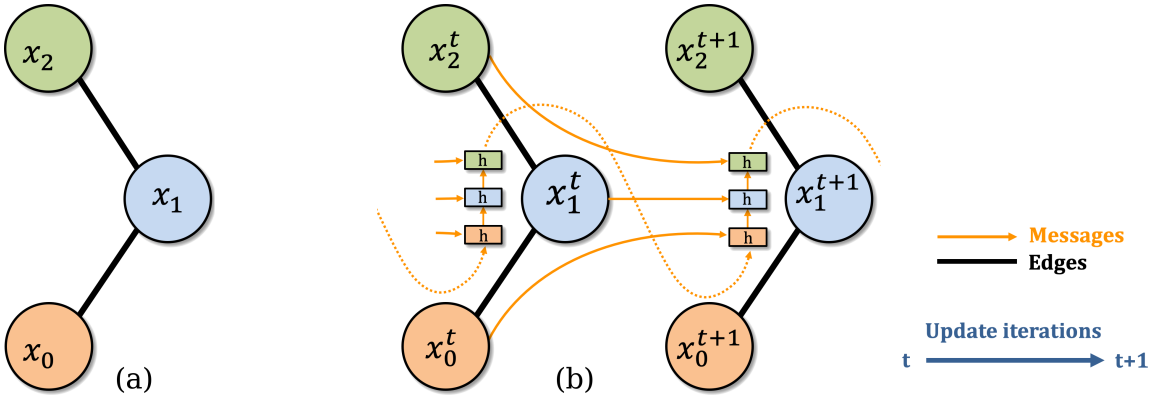


Figure 5.3: **(a) An example graph; (b) One iteration of the neural planner's message passing and bound update.** Messages from neighbors are serialized and fed through a GRU, an inductive bias for learning minima necessary for bound updates.

— an adaptation to an optimal planner in the probabilistic sense can be done in a straightforward manner. To avoid misunderstandings, we insist that the reasoning developed in this sub section is for illustration and general understanding of the chosen inductive network bias only, the real network parameters are fully trained with supervised learning as explained in section 5.4.

Handcrafting a parameterization requires imposing a structure on the node state $\boldsymbol{r}_i$, which otherwise is a learned representation. In our case, the node state will be composed of the bound b_i on the shortest path from the given node to the target node (a scalar), and the current estimate Π_i of the identity of predecessor node of node i w.r.t. the shortest path, which can be represented as a 1-in-K encoded vector indicating a distribution over nodes. Standard Bellman-Ford iteratively updates the bound for a given node i by examining all its neighbors j and checking whether a shorter path can be found passing through neighbor j. This can be written in a sequential form s.t. the bound gets updated iterating through the neighbors $j{=}1 \ldots J_i$ of node i:

$$\begin{aligned} b_i^{[0]} &= b_i \\ b_i^{[j]} &= \min(b_i^{[j-1]}, b_j + d_{ij}) \\ b_i' &= b_i^{[J_i]} \end{aligned} \tag{5.4}$$

where b_i and b_i' are the bounds before and after the round of updates for node i.

In our neural formulation, the message updates given in equation (5.2), further developed in (5.3), mimic the Bellman-Ford bound update given in Equations (5.4). This provided motivation for our choice of a recurrent neural network in the graph neural network, as we require the update of the recurrent state h_i^j in Equation (5.3) to be able to perform a minimum operation and an arg min operation (or differentiable approximations of min and arg min).

5.3.3 Graph creation from explorative rollouts

Graphs were generated during the initial rollout from an exploratory policy trained with Reinforcement Learning. During training, the agent interacts with training environments and receives RGB-D image observations calculated as a projection from the 3D environment. The agent is trained to explore the environment and to maximize coverage, similar to (Chaplot et al. 2020a; Chen et al. 2019b).

To learn to estimate the graph connectivity, we add an auxiliary loss to the agent's objective function, $f_{link}(\boldsymbol{o_i}, \boldsymbol{o_j}, \boldsymbol{h_i})$ which is trained to classify whether two locations are in line of sight of each other, conditioned on the visual features $\boldsymbol{o_i}, \boldsymbol{o_j}$ from the two locations and the agent's hidden state $\boldsymbol{h_i}$. Node features were calculated with a CNN (Lecun et al. 1998). Ground truth line of sight measurements were computed by 2D ray tracing on an occupancy map of each environment. In order to limit the size of the graph to a maximum number of nodes k, we aim to maximize each node's coverage of the environment using a Gaussian kernel function. At each time step a new node is observed by the agent, previous node positions are compared with a Gaussian kernel function (eq. 5.5) in order to identify the index of the most redundant node r, which is removed from the graph and replaced with the new node, node connectivities are then recomputed with $f_{link}(.)$, where $\boldsymbol{L}_i$ is the location of node i.

$$r = \arg\min_i \sum_j K(\boldsymbol{L}_i, \boldsymbol{L}_j), \qquad K(\boldsymbol{v}, \boldsymbol{v'}) = \exp\left(\frac{-\|\boldsymbol{v} - \boldsymbol{v'}\|^2}{2\sigma^2}\right), \tag{5.5}$$

5.4 Training

The high-level graph based planner — is trained in a purely supervised way. We generate ground truth labels by running a symbolic algorithm (Dijkstra (Dijkstra 1959)) on a set of valued ground truth training graphs described with the method detailed in Section 5.3.3. In particular, the supervised training algorithm takes as input *uncertain/noisy* graphs, which include visual features, and is supervised to learn to produce paths, which are calculated from known ground truth graphs unavailable during test time. During training we treat path planning as a classification problem where for a given target, each node must learn to predict the subsequent node on the optimal path to the target. Formally for each node i we predict a distribution $\boldsymbol{A_i}$ and aim to match a ground-truth distribution $\boldsymbol{A_i^*}$, which is a one-hot vector, minimizing cross entropy loss $\mathcal{L}(\boldsymbol{A}, \boldsymbol{A^*}) = -\sum_{i=1}^{n} \boldsymbol{A}_i^* \log \boldsymbol{A}_i$.

We augment training with a novel version of mod-drop (Neverova et al. 2015), an algorithm for multi-modal data, which drops modalities probabilistically during training. During training we extend the node connection probabilities with the ground truth node adjacencies and mask either the probabilities or

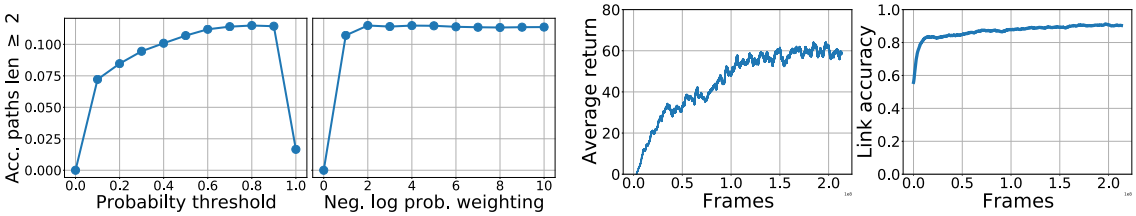


Figure 5.4: **Symbolic baselines and linkage prediction training curves.** Left: Symbolic baselines Dijkstra on thresholded probs. & cost function(5.5.1). Right: Average return & Acc. of line of sight predictions.

Table 5.1: **H-SPL and acc. of the neural planner's predictions.** Left: Uncertrain graphs. Right: Ground truth graphs.

Method	Acc	H-SPL
Symbolic (threshold)	0.114	0.184
Symbolic (custom cost)	0.115	0.269
Neural (w/o visual)	0.251	0.468
Neural (w visual)	**0.262**	**0.501**

Method	Acc	H-SPL
Symbolic (GT)	**1.00**	**1.00**
Neural planner (GT)	0.921	0.983

the adjacencies with a probability of 50%, during training we linearly taper the masking probability from 50% to 100% over the first 250 epochs. Ensuring that the final model requires only connection probabilities, but the reasoning performed during message passing and recurrent updates can be bootstrapped from the ground truth adjacencies. Training curves on unseen validation data are shown in figure 5.5.

The local policy — is a recurrent version of AtariNet (Mnih et al. 2015b) with two output heads for the action distribution and value estimates. The network was trained with a reinforcement learning algorithm Proximal Policy Optimization (PPO) (Schulman et al. 2017b) to navigate with discrete actions to a local point-goal. Point-goals were generated to be within 5m of the spawn location of agent. A dense reward was provided that corresponds to a decrease in geodesic distance to the target, a large reward (10.0) was provided when the agent reached the target and the *STOP* action was used. The episode was terminated when either the *STOP* action was used or after 500 time-steps. A negative reward of -0.01 was given at each time-step .

The explorative policy for graph creation — is trained with PPO (Schulman et al. 2017a). We aim to maximize coverage that is within the field of view of the agent. We create an occupancy grid of the environment with a grid spacing of 10cm. The first time a cell is observed the agent receives a reward of 0.1. A cell is considered to observable if it is free space, within 3m of the agent and in the field of view of the agent. Agent performance is shown in Figure 5.4.

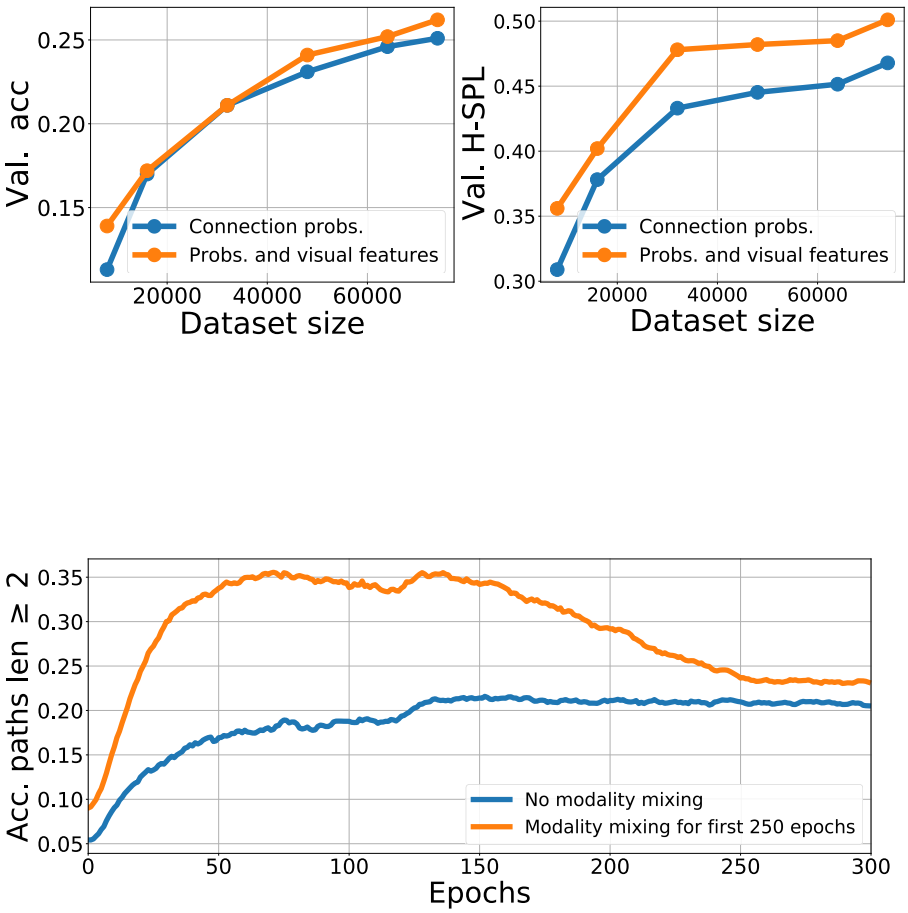


Figure 5.5: **Ablations.** Top: Accuracy and H-SPL with increasing size of data when training with and without visual features. Bottom: Modality mixing.

5.5 Experiments

We evaluated our method in simulated 3D environments in the Habitat (Savva et al. 2019a) simulator with the visually realistic Gibson dataset (Xia et al. 2018). During training, the agent interacts with 72 different environments, where each environment corresponds to a different home. We evaluate on a set of 16 held out environments.

5.5.1 High-level graph-based planner

The neural planner was implemented in PyTorch (Paszke et al. 2019).We compare two metrics, accuracy of prediction of the next way-point along the optimal path and the SPL metric (Anderson et al. 2018a), both for paths of length two or

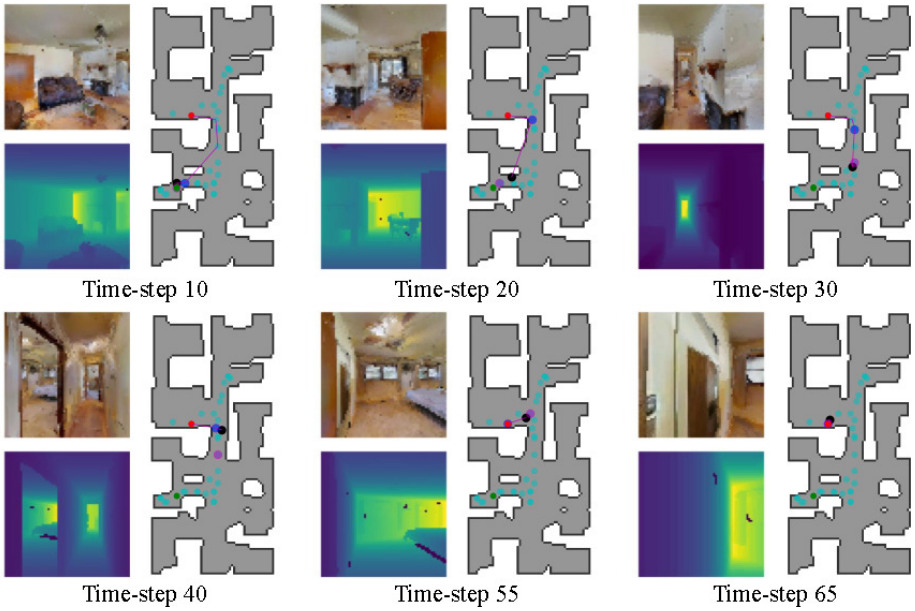


Figure 5.6: **Time-steps from a rollout of the hierarchical planner (graph+local).** For each time-step: left – RGB-D obs., right – map of the environment (unseen) with graph nodes, source node, target node, agent position(black), agent's nearest neighbour, local point-goal provided by the planner and planned path.

greater. As we evaluate SPL for both the high level planner and the hierarchical planner-controller, we refer to the high-level planner's SPL as H-SPL to avoid ambiguity.

Symbolic baselines — We compare the neural planner to two symbolic baselines, which reason on the uncertain graph only, without taking into account rich node features. While these baselines are *"optimal"* with respect to their respective objective functions, they are optimal with respect to the amount of information available to them, which is uncertain: *(i) Thresholding* — In order to generate non-probabilistic edge connections, we threshold the connection probabilities with values ranging from 0-1 in steps of 0.1. After threshholding the graph, path planning was performed with Dijkstra's algorithm; *(ii) A custom cost function* for Dijkstra's algorithm weighting distances and probabilities: $cost(i,j) = \boldsymbol{D}_{i,j} - \lambda \log(\boldsymbol{E}_{i,j})$. We vary λ in order to control the trade-off of distance and connection probability. When λ is 0, the graph is a fully connected graph, whereas high values of λ would lead to finding the most probable path.

Results of both symbolic baselines are shown in Figure 5.4, we observe that they perform poorly under uncertainty. We evaluate the accuracy of their predictions with respect to the symbolic baseline on the ground truth graph. We report accuracy on source-target pairs separated by at least 2 steps.

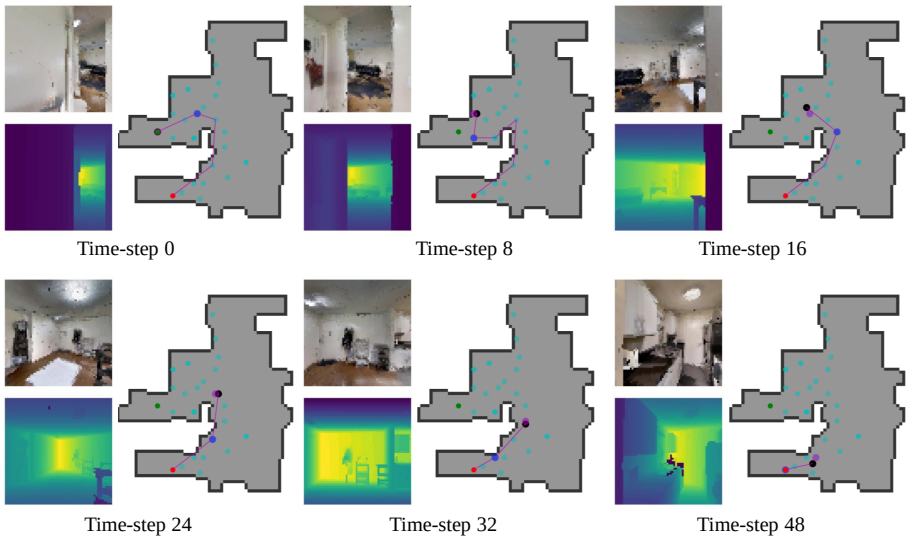


Figure 5.7: **Six time-steps from a rollout of the hierarchical planner** (graph+local) in an unseen testing environment. For each time-step: left – RGB-D observation, right – map of the environment (unseen) with graph nodes, source node, target node, agent position(black), nearest neighbour to the agent, local point-goal provided by the high level planner and planned path.

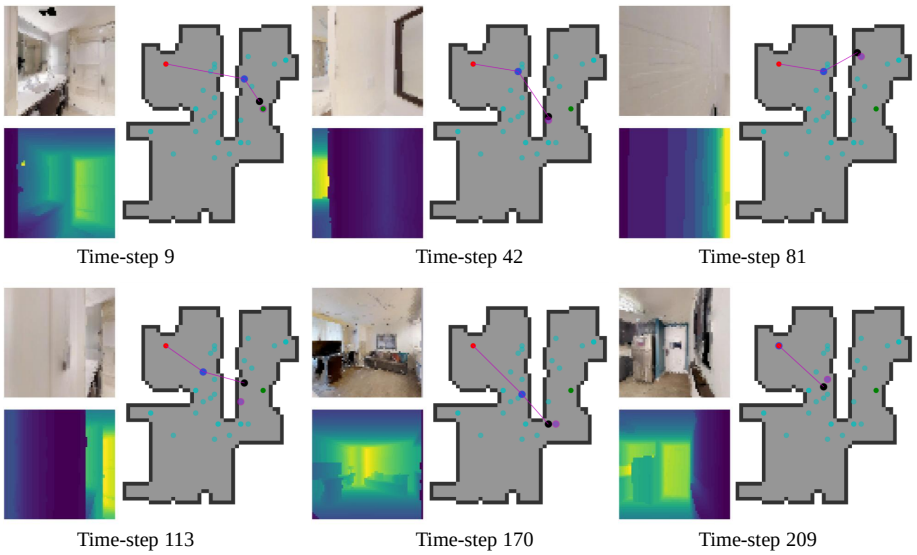


Figure 5.8: **Failure case - Six time-steps from a rollout of the hierarchical planner** (graph+local) in an unseen testing environment. For each time-step: left – RGB-D observation, right – map of the environment (unseen) with graph nodes, source node, target node, agent position(black), nearest neighbour to the agent, local point-goal provided by the high level planner and planned path.

Table 5.2: **Performance of the hierarchical graph planner & local policy**

Method: Planner + Local policy	Success rate	SPL
Graph oracle (optimal point-goals, not comparable)	0.963	0.882
Random	0.152	0.111
Recurrent Image-goal agent	0.548	0.248
Symbolic (threshold)	0.621	0.527
Symbolic (custom cost)	0.707	0.585
Neural planner (sampling)	0.966	0.796
Neural planner (deterministic)	**0.983**	**0.877**

Image driven recurrent baseline — We also compare to an end-to-end RL approach where the current obs. and target are provided to a CNN based RL agent. The architecture is a siamese CNN with a recurrent GRU. We train with a dense reward of improvement in geodesic dist. and provide a reward of 10 when the agent reaches the goal. We train with PPO for 200 M frames.

In Table 5.1, we compare the neural planner with the baselines. We can see, that even without visual features, the neural planner is able to outperform the "optimal" symbolic baselines. This can be explained with the fact, that the baselines optimize a fixed criterion, whereas the neural planner can learn to exploit patterns in the connection probability matrix $\boldsymbol{E}$ to infer valuable information on shortest ground truth path. The gap further increases when the neural planner can use visual features. Results of modality mixing (see section 5.4) are shown in Figure 5.5. As a sanity check, table 5.1 compares the optimal symbolic planner against the neural planner trained with ground truth adjacencies provided as input. The results of the neural planner are close to optimum in this case.

We evaluated our approach on dataset sizes ranging from 8,000 graphs to 74,000 graphs (Figure 5.5). One graph contains 32×32 possible source-target combinations, leading to a maximum amount of 75,000,000 training instances.

5.5.2 Hierarchical planning and control (topological & local policy)

We evaluated the neural graph planner coupled with the local policy. For a given episode, the graph planner estimates the next node in the path to a target image and provides its location to the local policy, which executes for m time-steps. The planner then re-plans from the nearest neighbor to the agent's current position, this back and forth process of planning and navigating continues until either the agent reaches the target or 500 low-level time-steps have been conducted. We report accuracy as percentage of runs completed successfully and SPL in table 5.2, albeit measured on low-level trajectories as opposed to graph space. We combine the local policy with various graph planners, and can see that the

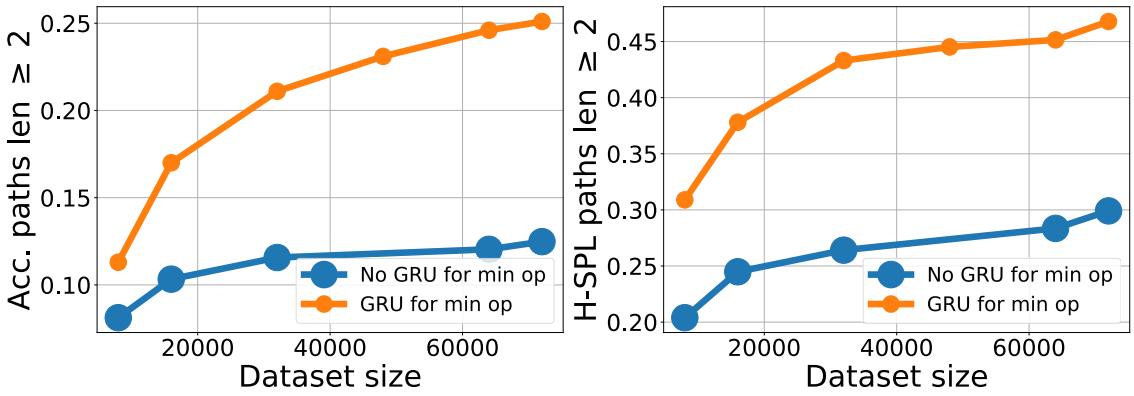


Figure 5.9: **Ablation of a GRU for the accumulation of incoming messages**. The GRU was added to ensure that the model could represent the Bellman-Ford algorithm.

neural graph planners greatly outperform the symbolic baselines. We perform two evaluations of the neural planner; a deterministic evaluation where point-goals are chosen with the argmax of the $\boldsymbol{A}$ distribution and a non-deterministic one by sampling from $\boldsymbol{A}$. The motivation is that by sampling, the planner can escape from local minima and loops created by errors in approximation. This is confirmed when studying rollouts from the agents, and also quantitatively through the performances shown in table 5.2. Visualizations of steps from several episodes are shown in Figures 5.6, 5.7 and 5.8.

5.5.3 Ablation: Effect of chosen inductive bias

As developed in sections 3.1 and 3.2, our graph based planner includes a particular inductive bias, which allows it to represent the Bellman-Ford algorithm for the calculation of shortest or best paths. This bias is implemented as a recurrent model (a GRU) running sequentially over the message passing procedures, as illustrated in Figure 5.3. Figure 5.9 ablates the effect of this additional bias as a function of data sizes ranging from 8,000 to 74,000 example graphs, each one evaluated with 32^2=1,024 different combinations of starting and end points.

The differences are substantial, we can see that our model is able to exploit increasing amounts of data and translate them into gains in performance, whereas standard graph convolutional networks do not — we conjecture that they lack in structure allowing them to pick up the required reasoning.

Figures 5.7 and 5.8 of this document show additional rollouts of the hierarchical planner, complementary to Figure 5.6.

5.6 Conclusion

We have demonstrated that path planning can be approximated with learning when structured in a manner that is akin to classical path planning algorithms. We have performed an empirical analysis of the proposed solution in photo-realistic 3D environments and have shown that in uncertain environments graph neural networks can outperform their symbolic counterparts by incorporating rich visual features. Our method can be used to augment a vision based agent with the ability to form long term plans under uncertainty in novel environments, without a priori knowledge of the particular environment. We have analyzed the empirical performance of the neural planning algorithm with a variety of dataset sizes, shown that the high-level planner can be coupled with a low-level policy and evaluated the hierarchical performance on an image-goal task.

This chapter has shown the power of decomposing planning and navigation into a hierarchy, with long-term neural planning learned through supervision and a low-level control policy learning with Deep Reinforcement Learning (RL). As environments grow in scale and scope, and the space of actions available to the agent grows, the requirement of decoupling planning and action becomes ever more essential. In the following chapter, we explore hierarchical topological approaches in vast simulated worlds, with an end application of navigation in video games.

CHAPTER 6

GRAPH AUGMENTED DEEP REINFORCEMENT LEARNING IN THE GAMERLAND3D ENVIRONMENT

Chapter abstract

In this chapter we address planning and navigation in challenging 3D video games featuring maps with disconnected regions reachable by agents using special actions. In this setting, classical symbolic planners are not applicable or difficult to adapt. We introduce a hybrid technique combining a low level policy trained with reinforcement learning and a graph based high level classical planner. In addition to providing human-interpretable paths, the approach improves the generalization performance of an end-to-end approach in unseen maps, where it achieves a 20% absolute increase in success rate over a recurrent end-to-end agent on a point to point navigation task in yet unseen large-scale maps of size 1km×1km. In an in-depth experimental study, we quantify the limitations of end-to-end Deep RL approaches in vast environments and we also introduce "GameRLand3D", a new benchmark and soon to be released environment built with the Unity engine able to generate complex procedural 3D maps for navigation tasks.

The work in this chapter has led to the following workshop paper:

- Edward Beeching, Maxim Peter, Philippe Marcotte, Jilles Dibangoye, Olivier Simonin, Romoff Joshua, and Christian Wolf (2022b). "Graph augmented Deep Reinforcement Learning in the GameRLand3D environment". In: *Association for the Advancement of Artificial Intelligence Conference on Artificial Intelligence (AAAI). Workshop on Reinforcement Learning in Games*.

6.1 Introduction

The work presented in this chapter was performed during a 4-month an industrial intership at Ubisoft LaForge, Montreal. The focus of which was applications for Deep Reinforcement Learning (RL) approaches in video-games. Long term planning and navigation is an important component when looking to realistically control a non-player character (NPC) in a video-game. Whether the NPC is player

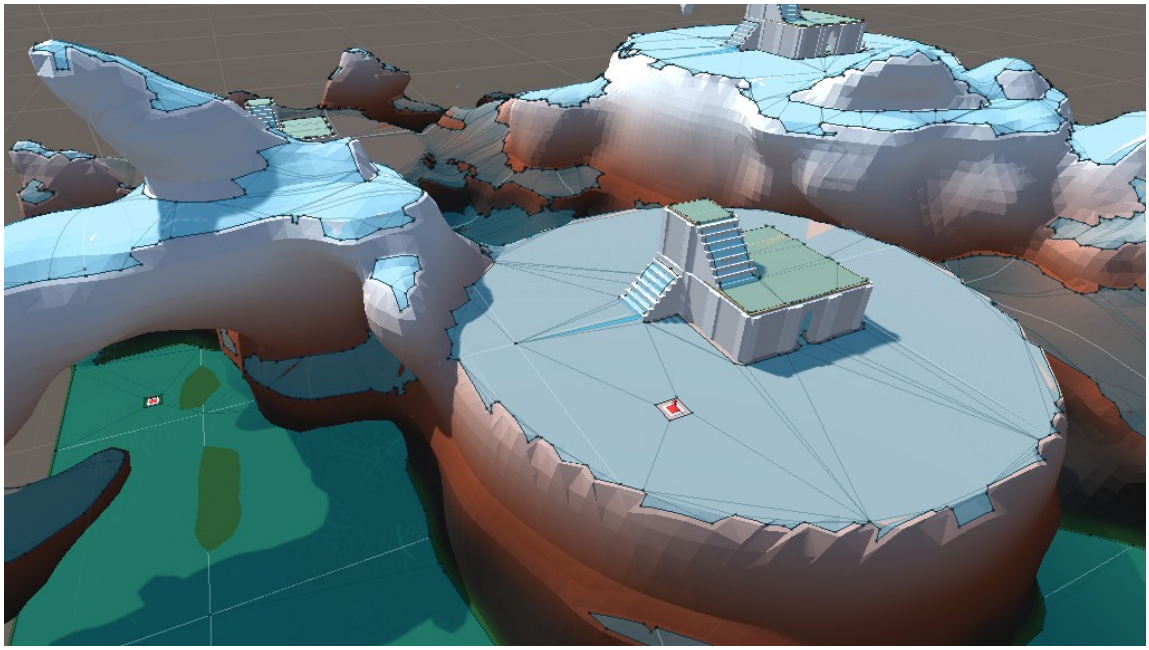


Figure 6.1: **An example Navigation mesh.** The proposed hybrid planner is capable of performing real-time navigation in complex 3D gaming environments featuring geometrically disconnected regions only reachable by agents through special actions. These environments are difficult for approaches based on the NavMesh.

facing or used for automated testing to find bugs and exploits, the ability to navigate in a complex environment is paramount and is the foundation of every NPC. In the field of video games the Navigation Mesh (NavMesh) (Snook 2000) has become the ubiquitous approach for planning and navigation. In recent years, the sheer scale of video game environments have highlighted the limitations of the NavMesh, particularly in games with complex navigation abilities such as double jumping, teleportation, jump pads, grappling hooks, and wall runs. Given the successes of Deep Reinforcement Learning (RL) in domains of navigation (Mirowski et al. 2016; Jaderberg et al. 2016b; Jaderberg et al. 2019), board games (Tesauro 1994; Silver et al. 2016b; Silver et al. 2017) and Atari video games (Mnih et al. 2013; Hessel et al. 2018a; Hafner et al. 2020), recent works (Alonso et al. 2021) have looked to exploit these approaches for navigation in large 3D video game environments. These works have shown promise in smaller game worlds, but have highlighted that as environments scale to hundreds of thousands of square metres, the limitations of end-to-end learning-based methods are encountered.

Currently, with few exceptions, navigation in video games relies on the NavMesh, which is created and used as follows.

1. A graph representation of the world is pre-generated from the game geometry.

2. At runtime a pathfinding algorithm such as A* (Hart et al. 1968b) is run on this graph to find the shortest path between two locations in the game.

3. A player controller, attached to the character, is used for point based navigation to follow the sequence of way-points returned by the planner.

While the navigation mesh provides a compact representation of the world, it is somewhat limited and results in disconnected regions in complex environments, as demonstrated in Figure 6.1. The disconnected regions are due to some assumptions of the underlying NavMesh generation algorithm. The user can provide the maximum inclination and height the agent can walk and jump, but more complex actions such as double jumping or using a jump pad are not taken into account. Connections due to special character abilities such as jump pads, grappling hooks and teleporters often involve time consuming manual editing of the graph in order to add additional links. While human intervention is possible, it is not feasible when the environments are procedurally generated as there can be a limitless number of new environment instances. Some automatic approaches do exist but are currently limited to simple actions (Axelrod 2008; Roumimper 2017; Budde 2013).

Recent works have looked to address this problem with an end-to-end Deep Reinforcement Learning (RL) approach (Alonso et al. 2021) and have shown that when trained and evaluated on small (50m×50m) to medium (100m×100m) sized maps, end-to-end approaches achieve good performance. However, when evaluated in larger (300m×300m) environments, policies learned using RL perform well on near and medium distance goals but poorly on farther challenging goals. Euclidean distance is not the best measure of complexity in this case, but as the odds of encountering a hard obstacle increase with the size of the map, it is a decent proxy for the complexity of navigation.

In this chapter we explore a hybrid approach that aims to combine the strengths of a classical graph-based NavMesh, with the capacity of end-to-end RL to incorporate complex abilities to improve navigation performance. Designed around the stringent constraints of online gaming environments, our method is able to generate new navigable graphs with low computational complexity. The proposed graph augmented RL agent outperforms the generalization performance of a learning-based agent on a set of maps of 250m×250m. In addition, the augmented approach also greatly improves the performance of a map specific agent that is trained and evaluated on a fixed game world of 1km×1km, 10 times larger than those typically used in the research community (Alonso et al. 2021).

We also introduce the "*GameRLand3D*" environment, a new benchmark, which pushes the size and complexity of gaming environments to new limits. It features realistic production-size maps as well as well as complex navigation tasks suited for modern 3D video games featuring navigation actions like jumps, pads, zip

lines etc. We think this benchmark is well suited to quantify the performance of end-to-end approaches and to explore potential novel solutions.

Our contributions are as follows:

- We quantify the limitations of the performance of end-to-end Deep RL approaches in complex and large-scale environments.
- We propose a novel hybrid graph and RL approach that decouples long distances planning and low level navigation and outperforms end-to-end Deep RL methods in both a generalization and map specific setup. Notably, this approach provides interpretable plans which can be tweaked by a game designer in order to modify the agent behavior without the need to fully retrain it.
- We release of an open-source 3D procedural environment in *Unity* which can generate complex 3D worlds of more than one million square meters.
- We introduce a detailed empirical evaluation and ablation of two hybrid RL and graph based approaches, with a special focus on their run time performance, a key requirement of video games operating in real time at 60 frames per second.

6.2 Related work

6.2.1 Deep RL environments

There has been a proliferation of new Deep RL environments in the last five years, the most well known being the gym framework (Brockman et al. 2016) which provides an interface to fully observable games such as the Atari-57 benchmark and several continuous control tasks. There are a variety of partially observable 3D simulators available, starting with more game-like environments such as the ViZDoom simulator (Wydmuch et al. 2018), DeepMind-Lab (Beattie et al. 2016b) and Malmo (Johnson et al. 2016b). A number of photo-realistic simulators have also been released such as Gibson (Xia et al. 2018) and AI2Thor (Kolve et al. 2017a). The highly efficient Habitat-Lab simulator (Savva et al. 2019a) builds on the datasets of Gibson and AI2Thor and implements a number of benchmark 3D navigation tasks. The aforementioned environments were designed to train and evaluate RL agents in relatively small environments of the order of hundreds of square metres, with relatively small action spaces. Moreover, they do not consider the complex actions that are in modern video games, such as double jumping, teleportation, jump pads, grappling hooks, and wall runs. The objective of the GameRLand3D environment, shown in Figure 6.2, is to provide a realistic benchmark for real world video game environments, to identify the limitations

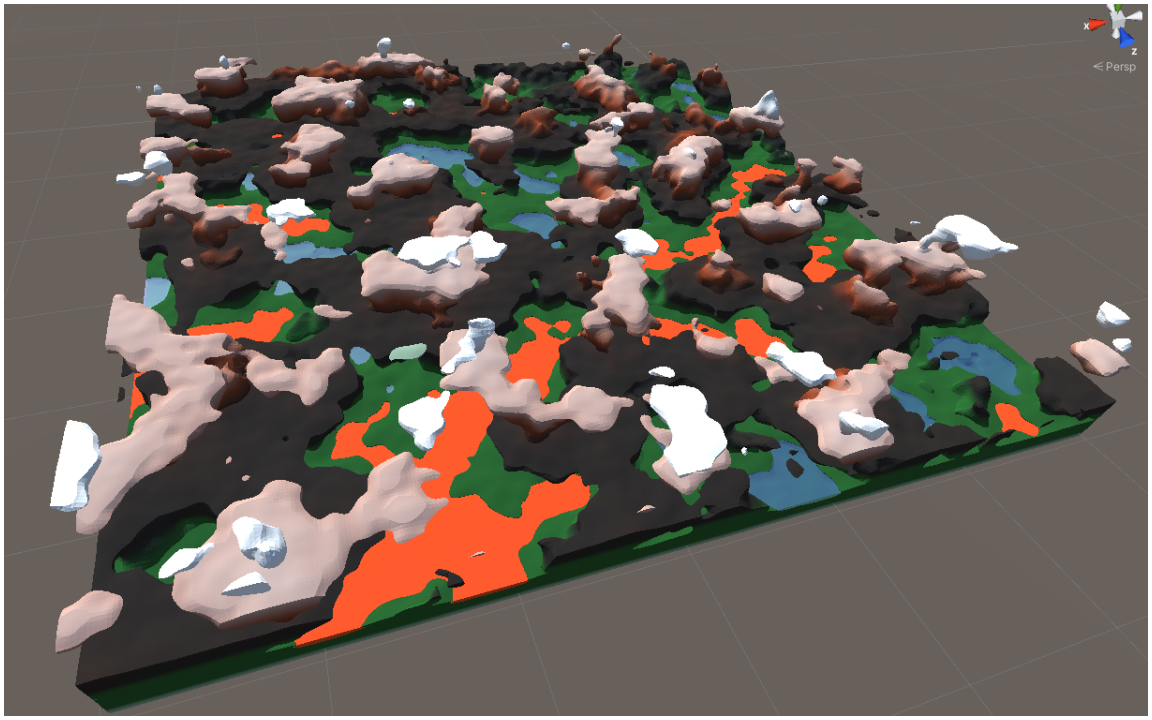


Figure 6.2: **A vast, 1km by 1km procedurally generated map generated with in the GameRLand3D environment.**

of current learning-based techniques and to allow reasearcher to explore novel alternatives.

6.2.2 Classical planning and navigation in video games

Existing methods for path planning in the video game industry mostly rely on building a Navigation Mesh (NavMesh)(Snook 2000). A NavMesh is a 2D/3D graph whose nodes represent convex polygonal regions of the walkable space, typically triangles, and whose edges indicate possibilities of navigation between regions. To find a path between two locations in the world, one can then apply path finding algorithms like A* (Hart et al. 1968a) or one of its variants on this graph. This classical approach is robust for many applications but presents some limitations. In particular, when it is possible for the agent to use mobility actions (like dashing, jumping, double jumping, using a jetpack), the high number of places it can reach from each location dramatically increases the connectivity of the NavMesh (Alonso et al. 2021; Gordillo et al. 2021). Therefore, it increases its memory cost as well as the runtime cost of the pathfinding algorithms, to the point where using a NavMesh ends up being too costly, constraining game design. Another important aspect that a NavMesh does not account for is the kind of constraint that a character can have, such as the turn radius of a car or a plane

in 3D. This kind of constraint can be impossible to encode in a NavMesh in an efficient manner.

Contrary to NavMesh-based approaches, RL for point-to-point navigation merges the path finding and the character controller (code the agent actually uses to move and follow the path) components by directly outputting actions the agent needs to take at every step to go from point A to point B. This makes using RL especially attractive since it could compress the semantics/topology of the map in a much more efficient way while having a fixed low cost to run.

6.2.3 Neural approaches to navigation

The field of Deep Reinforcement Learning (RL) has gained attention with successes on board games (Silver et al. 2016b) and Atari games (Mnih et al. 2015a). Recent works have applied Deep RL for the control of an agent in 3D environments (Mirowski et al. 2016; Jaderberg et al. 2016b), exploring the use of auxiliary tasks such as depth prediction, loop detection and reward prediction to accelerate learning. Other recent work uses street-view scenes to train an agent to navigate in city environments (Mirowski et al. 2018b).

To infer long term dependencies and store pertinent information about the partially observable environment, network architectures typically incorporate recurrent memory such as Gated Recurrent Units (Chung et al. 2015b) or Long Short-Term Memory (Hochreiter et al. 1997a). Extensions to memory based neural approaches began with Neural Turing Machines (Graves et al. 2014) and Differentiable Neural Computers (Graves et al. 2016) and have been adapted to Deep RL agents (Wayne et al. 2018b).

Spatially structured memory architectures have been shown to augment an agent's performance in 3D environments and are broadly split into two categories: metric maps which discretize the environment into a grid based structure and topological maps which produce node embeddings at key points in the environment. Research in learning to use a metric map is extensive and includes spatially structured memory (Parisotto et al. 2017a), Neural SLAM based approaches (Zhang et al. 2017b) and approaches incorporating projective geometry and neural memory (Gupta et al. 2017a; Bhatti et al. 2016b). These techniques are combined, extended and evaluated in (Beeching et al. 2020c; Wani et al. 2020). Research combining learning, navigation in 3D environments and topological representations has been limited in recent years with notable works being (Savinov et al. 2018a) who create a graph through random exploration in the ViZDoom RL environment (Wydmuch et al. 2018). (Eysenbach et al. 2019a) also performs planning in 3D environments on a graph-based structure created from randomly sampled observations, with node distances estimated with value estimates. (Beeching et al. 2020e) perform topological planning with a neural approximation of a classical planning algorithm that can be applied in uncertain environments. Other recent

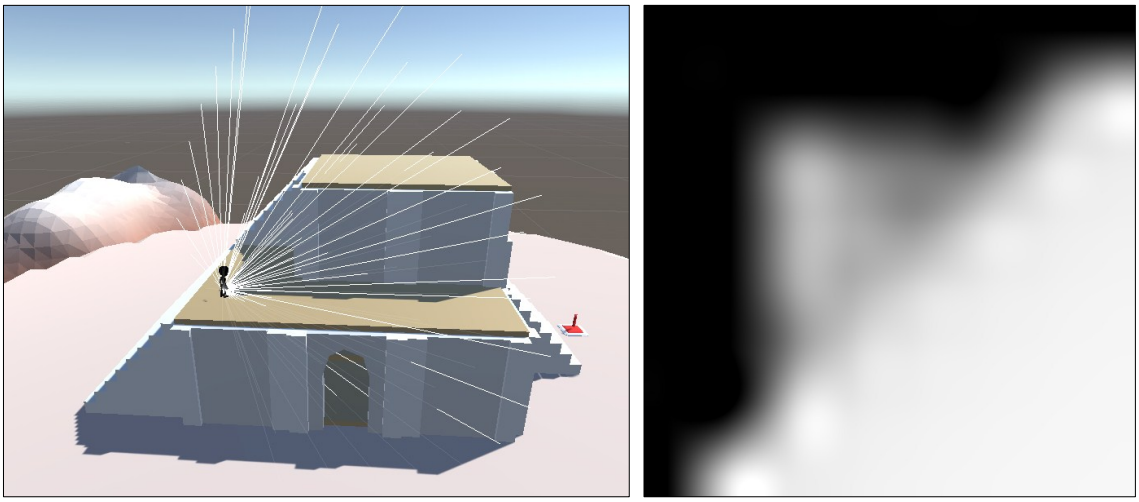


Figure 6.3: **Agent observations.** Left: An agent's depth observation in the GameRLand3D environment. Right: the resulting 2D observation in tensor form, that will be processed by the agent's CNN.

approaches such as (Chaplot et al. 2020e) build a graph structure, but rely on 360 degree camera measurements. Practically all of the aforementioned approaches are applied in small planar environments of hundreds of square metres, and scaling these methods to the environment where we apply our hybrid approach poses several challenges. Availability of ground truth: most graph based methods require ground truth actions and graph connectivity, as the calculation of optimal actions is often intractable in 3D environments of hundreds of thousands of square metres this precludes this possibility. We are also constrained due to inference time: many of these methods do not operate at real time speed, particularly when controlling hundreds of agents in parallel.

6.3 Environment and task definition

In order to address the challenging problem of navigation in vast complex 3D environments we have developed the GameRLand3D environment. GameRLand3D allows the creation of enormous worlds and enables complex interactions such as double jumping and jump-pads, includes buildings, water and lava hazards. The environment is built using the Unity-ML agents (Juliani et al. 2018) framework, which provides a versatile tool to build RL environments. We interact with the environment by controlling an agent, represented as a 1.8 meter humanoid character, shown in Figure 6.3.

Our environment implementation uses 3D Perlin noise (Perlin 1985) for the procedural generation of the game world, followed by random placements of

buildings, plateaus and jump pads to create a challenging state space for agents to interact with, explore and navigate.

6.3.1 Task

The task we consider is point-to-point navigation where the agent is spawned at a location p_{spawn} and must navigate to a goal location p_{goal}. Start and goal locations are sample from navigable terrain, using a NavMesh for simplicity, through a function $f_{navmesh}$ from which we can sample valid navigation locations. The NavMesh contains many disconnected regions and cannot be directly used for long distances navigation, but can be used to identify valid spawn locations. To reduce overall inference time and to allow for planning over longer time horizons, we request a decision from the agent's policy network every 10 time-steps, repeating the actions for the intermediary 10 steps.

6.3.2 Constraints

Video game environments provide certain advantages for training RL agents when compared to learning a model that may be transferred to the real world. The pose of the agent is known and we are not expected to rely on noisy pose estimates and fulfill safety guarantees, which add an additional challenge to robotic navigation. Video games do have other requirements and constraints, model implementations must be highly optimized, parallelized and run at real time speeds, making decisions at a minimum of 60 times per second. If an agent is player facing, its behavior must be credibly "human-like", a subjective metric that is discussed in (Devlin et al. 2021). Ideally, methods should provide interpretable information, so that game designers can be confident about adopting it in production.

6.3.3 Observations

Rendering agent observations from the environment with a virtual monocular camera is prohibitively expensive for NPC control in video games, so the agent's observations are a depth measurement of a 8×8 cone of raycasts in front of the agent (see Figure 6.3). We also provide useful information such as the location of the goal in the agent's frame of reference, the agent's velocity and acceleration.

6.3.4 Actions

The environment actions are continuous and correspond to actions that are typically available to a human player: forward, backward, strafe, turn, jump and

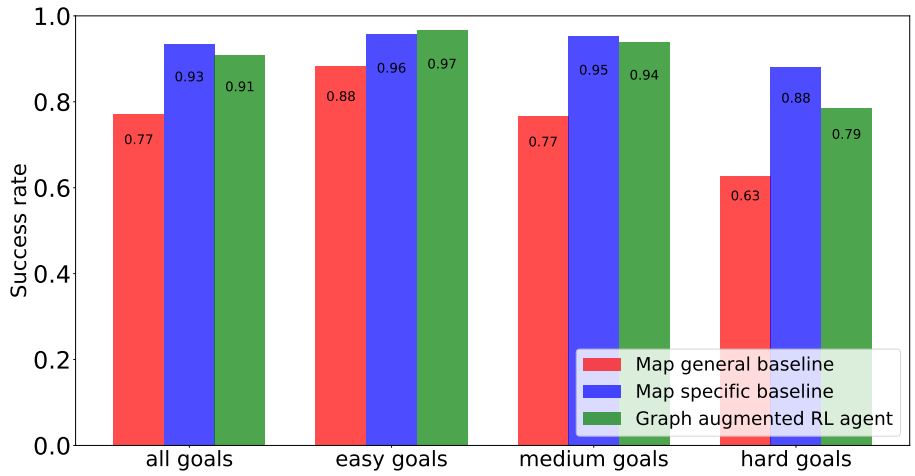


Figure 6.4: **The generalization gap** in performance between a map specific recurrent agent trained on a single map and a general recurrent agent trained on 128 maps and then evaluated on the same unseen map. Compared against a Graph augmented RL agents, introduced in this chapter.

double-jump (where a player can execute a second jump while in mid-air). The jump action is treated like a continuous action for the RL policy and is discretized in the environment.

6.3.5 Rewards

Building on the work of (Alonso et al. 2021), we combine three rewards: A **sparse reward** of +1 when the agent succeeds in the task and -1 when the agent fails. A **dense reward** every step in which the agent reduces its best euclidean distance to the goal location, divided by a normalization factor λ. A **time-step penalty** of -0.0005 per step, to encourage the agent to perform the task quickly. An episode is terminated when the agent is within 1m of its goal, the agent falls off the map or the agent has not decreased its best euclidean distance to the goal for 300 time-steps.

6.4 Empirical evaluation of end-to-end methods

We initially trained a baseline policy, in order to quantify the empirical performance of end-to-end Deep RL approaches in such large-scale worlds. In particular, we were interested in generalization, how well a policy trained in one set of environments transfers to a new environment. The baseline policy was trained using a recurrent version of a sample efficient off-policy actor critic RL algorithm,

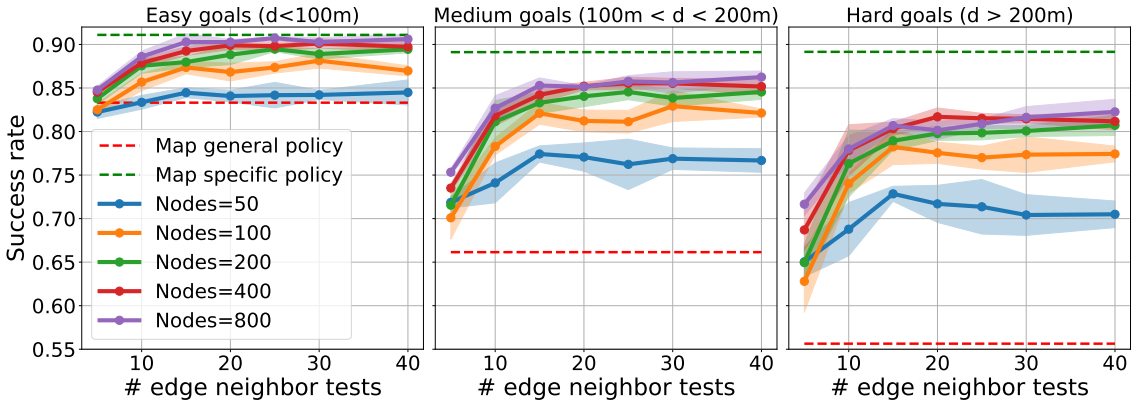


Figure 6.5: **Results for different hyper-parameters** for the method *Distance constrained random vertex placement*: graph sizes of 50 to 800 nodes; edge candidates of 5 to 40 neighbors

Soft Actor Critic (SAC) (Haarnoja et al. 2018a), with a learned entropy coefficient and no state value network (Haarnoja et al. 2018b). The SAC algorithm aims to finds a policy that maximizes cumulative reward and the entropy of each visited state.

$$\pi^* = \arg\max_{\pi} \sum_{t=0}^{T} \mathbb{E}_{(\mathbf{s}_t, \mathbf{a}_t) \sim \rho_\pi} \left[r\left(\mathbf{s}_t, \mathbf{a}_t\right) + \alpha \mathcal{H}\left(\pi\left(\cdot \mid \mathbf{s}_t\right)\right) \right] \tag{6.1}$$

We include a learnable entropy temperature parameter τ, to enable automatic optimization of the entropy of the objective function. We found that having independent policy and Q-network embeddings improves the reliability of the Q-value estimates and led to more robust, stable training. We trained several versions of the baseline agent architecture. A **reactive** agent that receives the 8x8 raycast observation and a **memory-based** recurrent agent that extends the reactive agent with an LSTM in order to be able to learn longer term dependencies. We use a burn-in of 20 time-steps to initialize the hidden state as recommended in (Kapturowski et al. 2018; Paine et al. 2019).

In order to quantify the reduction in performance when a policy is learned on one specific map or when a policy is learned on a set of maps and evaluated on another, we trained two versions of the memory based agent: a general agent and a map specific agent. The general agent is trained on a large dataset of 128 procedurally generated map instances. During training we evaluate the performance of the policy on a set of 4 validation maps and keep the model with the highest success rate. In addition, independently we train 4 map specific agents, one per test map, the objective being to quantify the decrease in performance, or *generalization gap*, between the map specific and general agent, see Figure 6.4. The graph augmented methods that we discuss in the following section use the general baseline for point-to-point navigation and achieve comparable performance to the

map specific agent. We note that a general agent is a crucial requirement in the video game production setting, as it removes the need of retraining on every new map instance.

6.5 Graph augmented RL agents

Pure end-to-end training of vanilla agents with recurrent memory leads to sub-optimal solutions for large and complex environments, as planning is performed on memory without spatial structure. While spatial organization of neural memory can be learned implicitly in principle, as shown by the emergence of grid-cells in unstructured agents trained for localization (Cueva et al. 2018), in practice this remains a hard task, in particular from reward alone. On the other hand, and as mentioned, the NavMesh alone is hardly applicable in the presence of special agent actions leading to disconnected regions.

We address this problem and propose a hybrid method which combines a directed graph (digraph) for planning and way-point to way-point navigation with a low-level policy trained with RL. This approach decomposes the problem of navigation over long distances into sequences of smaller sub-problems. The motivation is that (i) small-scale way-point navigation can be learned easily through reward, even in the presence of disconnected regions accessible with jump actions; (ii) large-scale navigation in a connected graph can be solved efficiently with optimal symbolic algorithms, like Dijkstra's algorithm or A*. We will show that the proposed method features a high success rate, provides interpretable paths and allows for identification of disconnected regions in the environment topology.

We explored several variants of the proposed method, which differ in the way how they dynamically create graphs when a new environment is spawned. For all variants, we build a graph G comprised of a set of vertices V and edges E, where edge $e(i, j)$ connects vertex v_i to vertex v_j. Each edge has a cost $c(i, j)$. Assigned to each vertex is a 3D position in the game. Planning from a position p_{start} to a goal position p_{goal} is performed by identifying the nearest vertex in the graph to the starting position v_{start} and the nearest vertex to the goal position v_{goal}. The optimal sequence of vertex locations to traverse in order to reach v_{goal} is found with the classical path planning algorithm Dijkstra (Dijkstra 1959).

The process of graph creation follows three key steps: identify vertex locations, testing edge connectivity and graph building. The following sections describe the different methods we explored for identifying the vertex locations, edges, edge costs and building the graph.

6.5.1 Unconstrained random vertex placement

In the unconstrained random approach we sample n vertices from the function $f_{navmesh}$. This provides vertices that are located on navigable terrain, but the edge connectivity is unknown. Edge connectivity is computed by first identifying the k nearest neighbors of each vertex and then evaluating a short point-goal episode between each vertex and its k-nearest neighbors. If the episode is successful then the directed edge is considered to be connected, with its cost equal to the number of steps in the episode. Using the agent steps as a cost over the euclidean distance leads to graph planning that takes into account local obstacles in the planning process.

6.5.2 Distance constrained random vertex placement

We augment the unconstrained random approach with coverage criteria, in order to spread nodes evenly across the environment and ensure that we can estimate connectivity all over the map. We aim to identify the set of vertex positions V that satisfy the criteria $d_V := \max_{x,y \in V, x \neq y} d(x, y)$, where $d(x, y)$ is the euclidean distance between two vertex positions. We find the set of positions V using a greedy strategy. We first calculating the optimal average distance between vertices, assuming the environment was a planar surface, and then sampling vertex locations from $f_{navmesh}$ until no nodes are present in the proximity of the sampled vertex location. In order to avoid infinite loops this test is repeated 100 times and after each test the distance criterion is decayed by a factor of 0.98.

Table 6.1: **Evaluation on a test set of 4 maps of size 250m×250m**. In addition to general reach and memory-based policies, we include the performance of 4 map specific policies, learned independently on each test map. We compare against the best performance of the unconstrained and constrained coverage-based hybrid methods.

	Success rate			
Method	**All goals**	**Easy goals**	**Medium goals**	**Hard goals**
Map specific LSTM baseline	0.90 ± 0.00	0.91 ± 0.00	0.89 ± 0.00	0.89 ± 0.00
General reactive baseline	0.68 ± 0.00	0.82 ± 0.00	0.62 ± 0.00	0.57 ± 0.00
General LSTM baseline	0.70 ± 0.00	0.83 ± 0.00	0.66 ± 0.00	0.56 ± 0.00
Unconstrained vertex placement	0.82 ± 0.02	0.88 ± 0.01	0.81 ± 0.03	0.77 ± 0.03
Constrained vertex placement	0.86 ± 0.00	0.90 ± 0.00	0.85 ± 0.00	0.81 ± 0.00

6.5.3 Optimizations

6.5.3.1 Robust estimation of edge connectivity

One weakness of performing the edge connectivity test once is that there is a small probability that agents can introduce an edge in the graph that often fails, leading to overconfident planning. This problem is compounded when there are several edges in the graph that are difficult to navigate. In order to identify and prune these edges, we evaluate an option to repeat the edge connectivity tests several times and keep the edges where the percentage of the tests exceed a given threshold. This approach leads to robust plans and higher performance, with the downside of a longer graph building step.

6.5.3.2 Re-planning

Qualitative analysis demonstrated that occasionally a graph augmented agent would fail to reach its next way-point, resulting in an agent that was often stuck as it had fallen off the side of an obstacle. This was resolved with the implementation of a re-planning step, where if the agent does not decrease its distance to the next way-point for 50 time-steps we automatically re-plan from its current location to the goal location. We observed that re-planning increased the success rate of all the examined methods.

6.6 Experiments

6.6.1 Setup

We generate maps of size 250m×250m×80m, 128 training maps, 4 validation maps and 4 tests maps to test the baseline and graph augmented methods. We then scale this approach to maps of 1km×1km ×80m to evaluate the performance in vast game worlds. Such a map is notably larger than the largest maps tested in the literature (Alonso et al. 2021).

While other works also include the Success weighted by Path Length (SPL) metric (Anderson et al. 2018a), we are unable to compare against this metric as the SPL metric. As SPL requires ground truth shortest paths, which cannot be computed in such large environments with complex action spaces. We separate the goals into three groups, easy, medium and hard, based on distance between the start and end points to break down the performance of each method. The results on hard, long distance goals, are of particular interest as this is where the baseline policy had the worst performance.

6.6.2 Hyper-parameter optimizations

We extensively optimized the hyper-parameters of each approach and report the results of the best performing configuration for each technique. Each test was evaluated with three seeds in order to quantify the average and variation of performance that can be achieved with these methods. We evaluate the performance of each method using the success rate metric on 4 test maps, with 2,500 goals per map. As an example, we report in Figure 7.1 the evaluation of a single method, *Random vertex with coverage constraint*, in numerous hyper-parameter configurations. In this case we have varied the number of nodes and the number of nearest neighbors with which to perform edge connectivity tests.

6.6.3 Comparisons

We evaluate the performance of the proposed graph augmented methods and compare them to the performance of the recurrent baselines in table 6.1. We observe that the *Distance Constrained Random Vertex Placement* outperforms the baseline method with a success rate of 85.9%, an improvement of 16.3%. We also implemented flooding and trajectory based methods which are excluded for brevity, but are discussed in the overview video .

6.6.4 Run-time analysis

Run-time performance is critical in real time video games, where the game logic and rendering must be executed 60 times per second. We perform an analysis of the run-time performance of the various approaches. In particular, we are interested in the time it takes to build the graph, to evaluate 2,500 goals on a test map and average duration of a graph search when exploiting the hybrid approach. Figure 6.6 summarizes the results for key sets of hyper-parameters for the *Distance Constrained Random Vertex Placement* approach. While larger graphs improve the performance of the augmented agent, they also increase the evaluation time and the time taken to perform a graph search. There is clearly a trade-off to be made, with the optimal choice depending on the exact use case, for an offline exhaustive test of a map instance a higher node count is suitable. Whereas if the graph must be created in near real-time a smaller node count would be required.

6.6.5 Scaling to production size maps

Previous experiments have focused on generalization of a learned policy to unseen maps. We have extended our method to a vast 1km×1km×200m map. We find that even a map specific baseline RL agent, whose policy is learned directly on

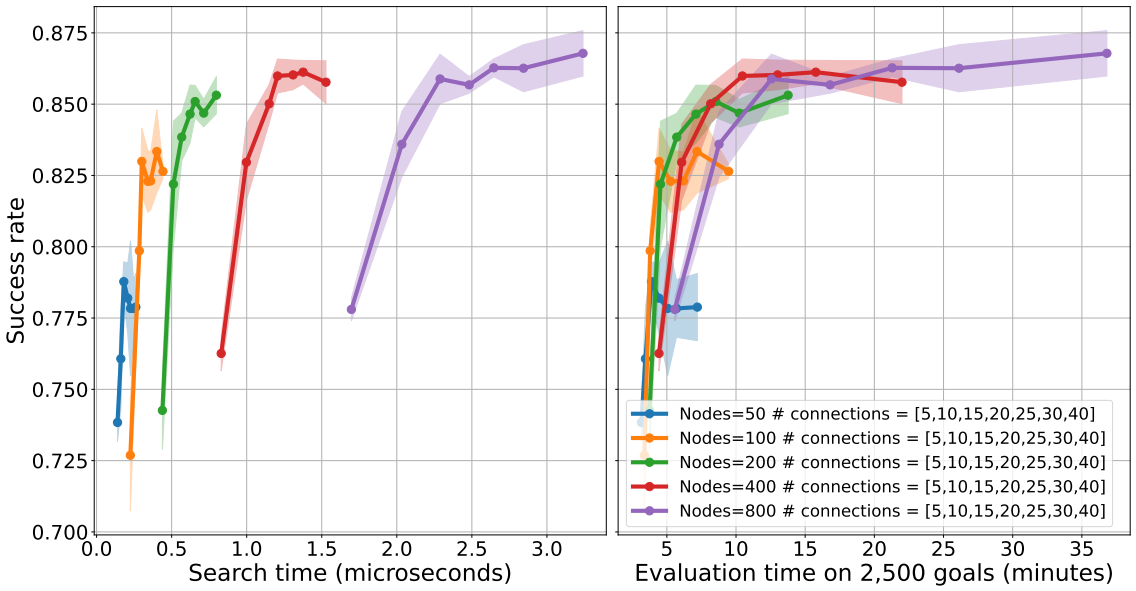


Figure 6.6: **Run-time trade-offs of the coverage constraint approach.** Left: Average graph search time in ms in several configurations. Right: Average graph construction and evaluation time in minutes on 2,500 goals.

the map it is evaluated on (environment overfit), performs poorly compared to our hybrid agent, with an overall success rate of 72%. We augment the map specific agent with the *Distance Constrained Random Vertex Placement approach* and evaluated the performance on 10,000 goals with graphs of size 200 to 1,600 nodes. We achieve an increase in success rate of 20% on all goals, shown in Figure 6.7.

6.7 Conclusions

We have introduced a set of graph augmented RL approaches for planning and navigation in vast 3D video game worlds. To demonstrate the capacity of these approaches we have developed the GameRLand3D environment, an open-source reinforcement learning environment built in the Unity game engine. GameRLand3D enables procedural generation of vast open world environments with hazards such as water and lava, and features such as jump pads, buildings and overhangs. We have identified that end-to-end Deep RL approaches underperform in large-scale environments. We evaluated two graph augmented RL techniques and performed extensive hyper-parameter tests to find a reasonable compromise between graph building time, run-time performance and success rate. Our hybrid approaches boost generalization performance from 69.6% to 86.2%. We also achieve a 20% increase in success rate in enormous environments.

In addition to support and maintenance of the GameRLand3D environment, our roadmap includes the addition of more complex building types, player abilities

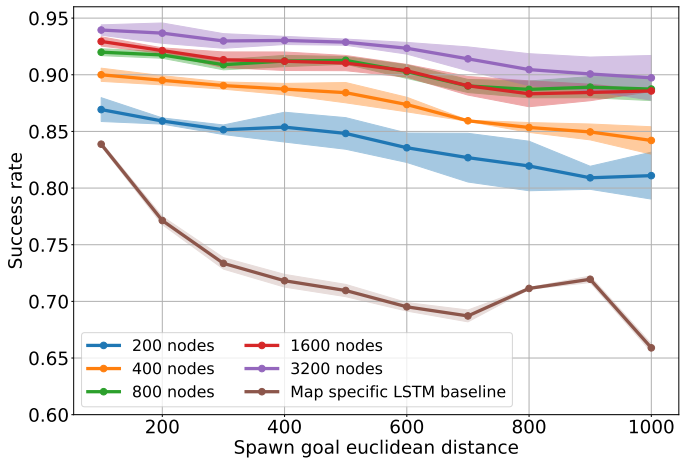


Figure 6.7: **Evaluation of the Random Graph with coverage approach on a single $1km^2$ map with 10,000 goals.** Compared against a map specific LSTM baseline policy. Shown is goal distance, in 100m bins, compared with the success rate of each approach. We observe that the method begins to reach diminishing returns at 1,600 vertex locations. Overall average success rate is increased from 72% to 92%.

such as teleportation and jetpacks, and other features such as ravines and caves. Future hybrid approaches in this environment will investigate the possibility of online graph building, using function approximation to predict edge connectivity in dynamic environments where the graph must be reconstructed when there is a change in map topology.

In this chapter, we have built a large 3D environment using the UnityML-Agents framework. While this framework is a useful tool to the research community it does come with drawbacks, its closed-source nature means researchers do not have access to the underlying protocols for exchanging observations between the python training loop and the environment running inside the Unity executable. In addition, Unity is a pay-to-use tool, which limits accessibility. In the following chapter, we develop an open-source tool to interface with an alternative Game Engine, allowing researchers and game designers a free and flexible framework to build and test simulated worlds.

CHAPTER 7

GODOT REINFORCEMENT LEARNING AGENTS

Chapter abstract

In this chapter we present Godot Reinforcement Learning (RL) Agents, an open-source interface for developing environments and agents in the Godot Game Engine. The Godot RL Agents interface allows the design, creation and learning of agent behaviors in challenging 2D and 3D environments with various on-policy and off-policy Deep RL algorithms. We provide a standard Gym interface, with wrappers for learning in the Ray RLlib and Stable Baselines RL frameworks. This allows users access to over 20 state of the art on-policy, off-policy and multi-agent RL algorithms. The framework is a versatile tool that allows researchers and game designers the ability to create environments with discrete, continuous and mixed action spaces. The interface is relatively performant, with 12k interactions per second on a high end laptop computer, when parallized on 4 CPU cores. An overview video is available here: https://youtu.be/g1MlZSFqIj4.

The work in this chapter has led to the following workshop paper:

- Edward Beeching, Jilles Dibangoye, Olivier Simonin, and Christian Wolf (2022a). "Godot Reinforcement Learning Agents". In: *Association for the Advancement of Artificial Intelligence Conference on Artificial Intelligence (AAAI). Workshop on Reinforcement Learning in Games*.

7.1 Introduction

Over the next decade advances in AI algorithms, notably in the fields of Deep Learning (Lecun et al. 2015) and Deep Reinforcement Learning, are primed to revolutionize the Video Games industry. Customizable enemies, worlds and story telling will lead to diverse game-play experiences and new genres of games. Currently the field is dominated by large organizations and pay to use engines that have the resources to create such AI enhanced agents. The objective of the Godot RL Agents package is to lower the bar of accessibility; so that game developers, researchers and hobbyists can take their idea from creation to publication end-to-end with an open-source and free package. The Godot RL Agent provides

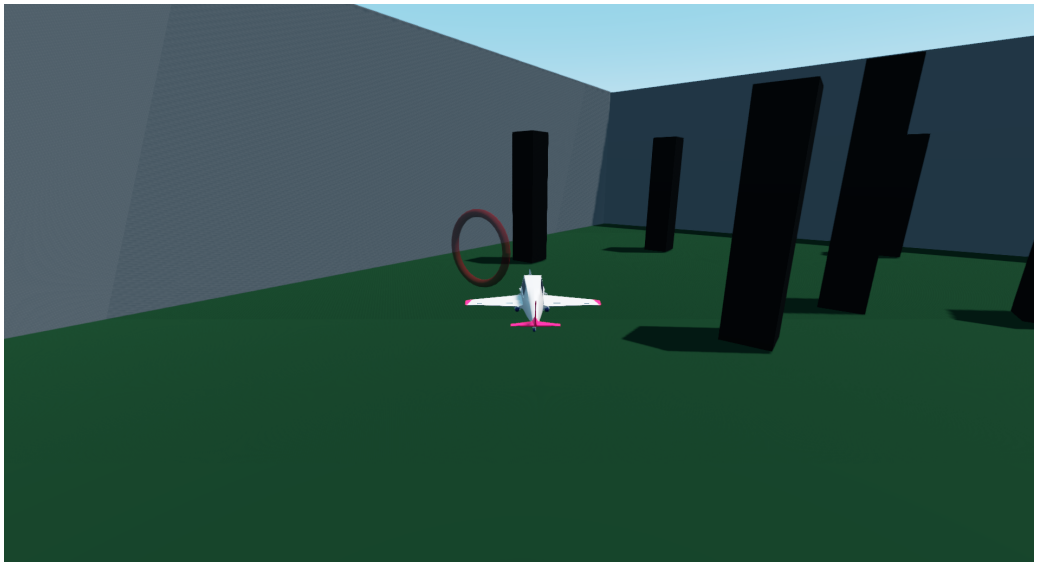


Figure 7.1: **An example Godot RL Agents environment *Fly By***, where an agent learns to perform way-point based aerial navigation in a 3D space.

wrappers for two well known open-source Deep Reinforcement Learning libraries: Ray Rllib (Liang et al. 2018) and Stable Baselines (Hill et al. 2018). In addition the interface is compatible with any RL framework that can interact with a gym wrapper. Although the Godot RL Agents interface was predominantly designed for the creation and prototyping of RL approaches in video games, there is nothing prohibiting users from the implementation of tasks involving mobile robotics, grasping or designing complex memory-based scenarios to test the limitations of current Deep RL architectures and algorithms. The contributions the Godot RL Agents packages are as follows:

- A free and open source tool for Deep RL research and game development.
- By providing an interface between the Godot Game Engine and Deep RL algorithms, we enable game creators to imbue their non-player characters with unique behaviors, learned through interaction.
- The framework enables automated game-play testing with an RL agent.
- The Godot RL Agents framework enables researchers and game designers the flexibility to design, create and perform rapid iteration on new ideas and scenarios.

The library is open-source with an MIT licence, code is available on our github page:
https://github.com/edbeeching/godot_rl_agents

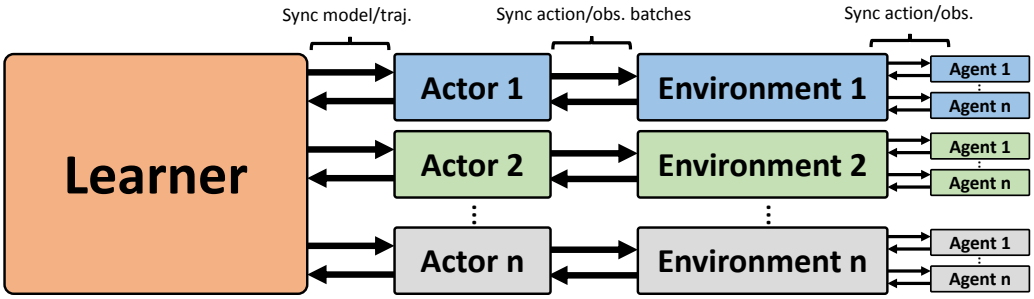


Figure 7.2: **The training architecture used for the Godot RL Agents interface.** Parallel actor processes collect trajectories from several environment executables. Each environment contains multiple parallel agents, each with their own instantiation of the environment.

7.2 Related work

The last decade has seen rapid advancement in the field of artificial intelligence, this has been driven by the application of Deep Learning approaches and in particular Convolutional Neural Networks (Fukushima et al. 1982; Lecun et al. 1998) . Supervised Deep Learning in its current state often considered to be narrow AI, that performs particularly well at performing predictions on static images but does not **learn from interaction**.

7.2.1 Deep Reinforcement Learning

Reinforcement Learning approaches provide the ability to learn in sequential decision making problems, where the objective is to maximize accumulated reward. As we encounter large state spaces, continuous actions, partial observability (Åström 1965) and the desire to generalize to unseen environment configurations; Deep Reinforcement Learning (RL) provides a general framework for solving these problems. In recent years the field of Deep RL has gained attention with successes on board games (Silver et al. 2016a) and Atari Games (Mnih et al. 2015b). One key component was the application of deep neural networks to frames from the environment or game board states. Recent works that have applied Deep RL for the control of an agent in 3D environments such as maze navigation are (Mirowski et al. 2017) and (Jaderberg et al. 2016a) which explored the use of auxiliary tasks such as depth prediction, loop detection and reward prediction to accelerate learning. Meta RL approaches for 3D navigation have been applied by (Wang et al. 2016a) and (Lample et al. 2017b) also accelerated the learning process in 3D environments by prediction of tailored game features. There has also been recent work in the use of street-view scenes in order train an agent to navigate in city environments (Kayalibay et al. 2018). In order to infer long term

dependencies and store pertinent information about the environment; network architectures typically incorporate recurrent memory such as Gated Recurrent Units (Chung et al. 2015a) or Long Short-Term Memory (Hochreiter et al. 1997b). The scaling of Deep RL has produced some impressive results, such as in the IMPALA (Espeholt et al. 2018c) architecture which successfully trained an agent that can achieve human level performance in all 30 of the 3D, partially observable, DeepMind Lab (Beattie et al. 2016b) tasks; accelerated methods such as in (Stooke et al. 2018) which solve many Atari environments in tens of minutes. The achievements in long term planing and strategy in DOTA (OpenAI 2018) and StartCraft 2 (Vinyals et al. 2019), has demonstrated the Deep RL can learn policies with complex reasoning and long term decision making. There has been a recent push towards photo-realistic simulation, with the release of the Gibson (Xia et al. 2018) dataset and the Habitat simulator (Savva et al. 2019a; Szot et al. 2021). Many research teams focus on Structured Deep Reinforcement Learning, where priors are incorporated in the agent architecture, leading to advancements in GPS guided point-to-point navigation (Chaplot et al. 2020b; Chaplot et al. 2020f), memory-based tasks (Wayne et al. 2018b; Beeching et al. 2020c; Beeching et al. 2020e), semantic prediction (Chaplot et al. 2020c; Chaplot et al. 2020d). Other approaches aim to solve these challenging problems with scale, using distributed computing to training larger networks for billions of environment interactions (Wijmans et al. 2019b; Espeholt et al. 2018c). The video games industry has identified the advancements in Deep Reinforcement Learning as an opportunity to enrich video game experiences. With short term objectives being that of automated testing (Gordillo et al. 2021) and content generation (Gisslén et al. 2021), and medium to longer term being player facing bots (Alonso et al. 2021), controlled by policies learned through interaction with the game environment.

7.2.2 Deep RL environments

There has been a large expansion to the number of Deep RL environments available in the last five years, the most well known being the gym framework (Brockman et al. 2016) which provides an interface to fully observable games such as the Atari-57 benchmark. Mujuco (Todorov et al. 2012) has remained the de facto standard for continuous control tasks, its recent open-source release can only benefit the larger RL community. OpenSpiel (Lanctot et al. 2019) provides a framework for RL in games such as Chess, Poker, Go, TicTacToe and Habani, to name a few. There are a variety of partially observable 3D simulators available, with more game-like environments such as the ViZDoom simulator (Wydmuch et al. 2018), DeepMind-Lab (Beattie et al. 2016b) and Malmo (Johnson et al. 2016b). A number of photo-realistic simulators have also been released such as Gibson (Xia et al. 2018), AI2Thor (Kolve et al. 2017a), and the highly efficient Habitat-Lab simulator (Savva et al. 2019a; Szot et al. 2021). The aforementioned environments

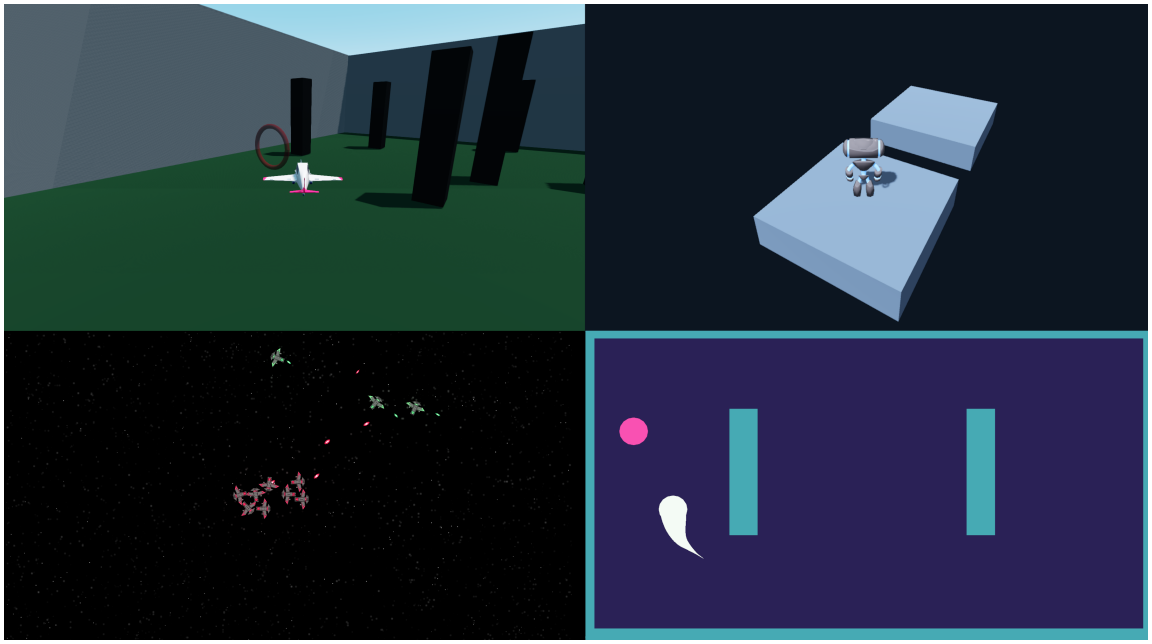


Figure 7.3: **Four of the example environments available in the Godot RL Agents framework.** Top left: *Fly By*, where the agent must learn to fly a place between way-points in 3D space. Top right: *Jumper*, where the objective is the navigated a bipedal robot between platforms, jumping where necessary. Bottom left: *Space Shooter*, a 2D multi agent scenario, where one team must shoot and and destroy the agents from the other team. Bottom-right: *Ball Chase*, where the agent must learn to collect pink fruits, while avoiding walls and obstacles.

were designed to train and evaluate RL agents in relatively small environments and typically are somewhat inflexible to addition of new tasks, abilities and action spaces. The Unity ML agents framework (Juliani et al. 2018), provides flexibility in terms of design of environments and tasks, but with the downside of being a closed-source and pay to use (if your organization has a revenue of more than $100k per year). In addition its closed-source implementation is inflexible if you wish to, for example, make changes to the protocol used to exchange observations between the environment and the RL algorithm. For these reasons the platform is somewhat prohibitive to the wider RL community.

7.2.3 The Godot Game Engine

The Godot Game Engine (Linietsky et al. 2014) is an open-source tool for developing 2D and 3D games. Currently Godot is the most popular open-source Game Engine available and can export to MacOS, Linux, Windows, Android, iOS, HTML and web assembly. Godot supports multiple programming languages including

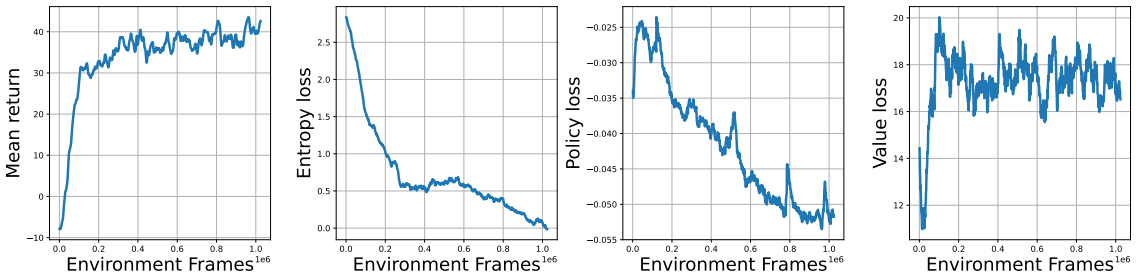


Figure 7.4: **Training curves available in Godot RL Agents for the Ball Chase environment.** Shown are a subset of available statistics output during training, Mean return, Entropy loss, Policy loss and Value loss

Python, C#, C++ and GDScript (a Python-like scripting language). The Game Engine provides an interactive editor for the design and creation of video game environments, characters, animations and menus, which enables fast iteration for prototyping new ideas. Godot has all the features of a modern Game Engine such physics simulation, custom animations, plugins and physically-based rendering.

7.2.4 Deep RL Frameworks

When it comes to applying a state of the art Deep RL algorithm, there are many quality open-source implementations available such as RLlib (Liang et al. 2018), ACME (Hoffman et al. 2020), SampleFactory (Petrenko et al. 2020), Seed RL (Espeholt et al. 2019) and Stable Baselines (Hill et al. 2018) to name a few. Due to the availability of such high-quality, featured and well tested Deep RL frameworks; in this work we chose to design and implement an interface between the Godot Game Engine and two well known RL frameworks: RLlib (Liang et al. 2018) and Stable Baselines (Hill et al. 2018). Our objective is to focus on the interface and quality examples of 2D and 3D environments. This is contrary to the Unity ML Agents framework, which also implemented a limited set of RL algorithms.

7.3 Godot RL Agents

The Godot RL Agents framework is an interface between the Godot Game Engine and Deep RL frameworks, allowing the creation of 2D, 3D and text-based environments, custom observations and reward functions. It is a flexible, fast and robust interface between RL training code running in Python and the Godot environment running interactively or as a compiled executable. We implement two forms of parallelization in the Godot RL Agents interface: in-game environment duplication and multiprocessing with several parallel executables. The interface has been designed to support continuous, discrete and mixed action spaces. Action repeti-

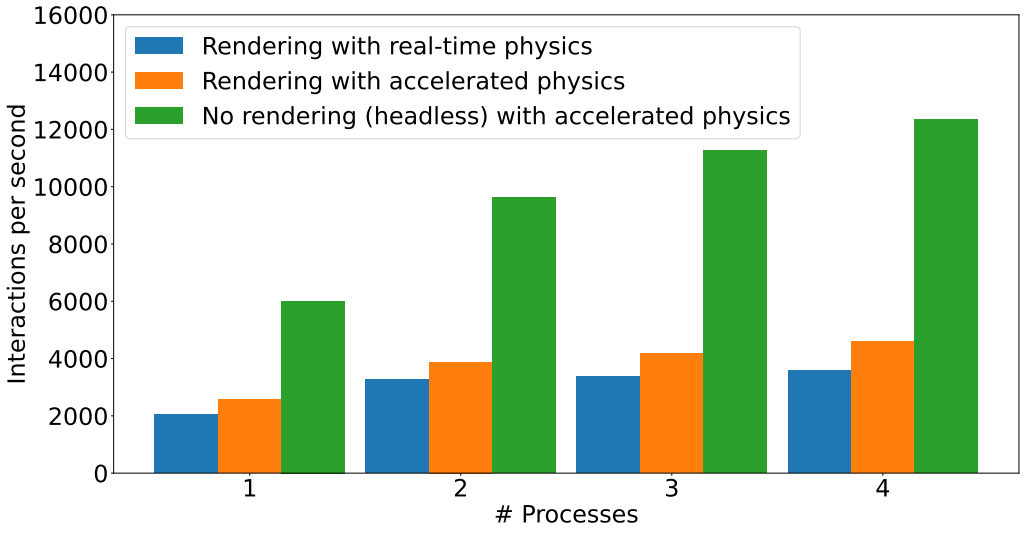


Figure 7.5: **Benchmark of interactions per second when performing multi-process training in the Jumper environment** with an action repeat of 4. We achieve 12k interactions per second with 4 parallel processes. Ideally scaling would be linear, future versions of Godot RL Agents will aim to profile and optimize the scaling of environment interactions.

tion is implemented on the environment side to avoid unnecessary inter-process communication. Training can be performed interactively, while the environment is running in the Godot game editor: enabling mid-training pausing, analysis and debugging of variables. Once the user is confident with their environment configuration though interactive testing, the environment can be exported and run as an executable. Environment export enables accelerated, faster than real-time physics and headless training. The connection between the Game Engine and the Python training code is implemented with a TCP socket client-server relationship; this allows the environment to be run locally or distributed across many machines. We provide a standard Gym wrapper and wrappers to Ray RLlib and Stable Baselines, which support the following algorithms.
On-policy algorithms:

- A2C/A3C (Mnih et al. 2016a)
- PPO/APPO (Schulman et al. 2017a)
- IMPALA (Espeholt et al. 2018c)
- DD-PPO (Wijmans et al. 2019b)

Off-policy algorithms:

- DQN (Mnih et al. 2013)

- Rainbow (Hessel et al. 2018a)
- CQL (Kumar et al. 2020)
- DDPG / TD3 (Lillicrap et al. 2015b)
- APEX (Horgan et al. 2018)
- R2D2 (Kapturowski et al. 2018)
- SAC continuous (Haarnoja et al. 2018a) and discrete (Christodoulou 2019).

Other algorithms are also available such as Behavior Cloning (Wang et al. 2018), Intrinsic curiosity (Pathak et al. 2017) and Evolutionary Strategies (Salimans et al. 2017). These pre-existing implementations enable researchers and game developers to rapidly evaluate different baselines architectures and algorithms, with robust implementations of state of the art RL algorithms. We detail the distributed architecture used in Godot RL Agents in Figure 7.2.

7.3.1 Sensors

Godot RL Agents allows users to add custom sensors to their agents to observe the environment. While simulated monocular cameras are possible, rendering is an expensive operation and rendered images do no always provide the more pertinent information. We provide implementations of circular and spherical "Raycast" sensors in both 2D and 3D that perform depth measurements, essentially a virtual LIDAR. Raycasts are a popular technique in video games and Game Engines such as Godot offer a highly optimized raycast implementation, making ray-casts a cheap option to augment an agent's observation. Other custom information, such as position, can be included in the agents observation, if the environment designer believes it is applicable to the task.

7.3.2 Example environments

In version 0.1.0 of Godot RL Agents, we have implemented 4 example environments as a reference for researchers and game creators who wish to use our tool to build new worlds. We have created two 3D environments and two 2D environments. The source-code and further details (such as reward functions, reset conditions, action spaces) of these reference implementations, is available in our open-source repository `https://github.com/edbeeching/godot_rl_agents`.

7.3.2.1 Jumper

Jumper is a 3D single agent environment where the agent must learn to jump from one randomly placed platform to the next. The action space is continuous for turning and movement, and discrete for the jump action. The episode is terminate

Figure 7.6: **Future work**, an example of a virtual monocular camera, rendering part of a scene from the Gibson dataset, loaded as an environment in the Godot RL Agents framework.

if the agent falls off the platform. Observations are a cone of 3D ray-casts in front of the agent and vector pointing towards the next platform.

7.3.2.2 Ball Chase

Ball Chase is a 2D single agent environment where the agent must navigate around room and collect pink balls. The episode is terminated if the agent hits a wall. Observations are a circle of 2D ray-casts around the agent and vector pointing towards the next fruit. The action space is a continuous 2D vector indicating the move direction in x and y.

7.3.2.3 Fly By

Fly By is a 3D single environment where the agent must learn to fly from one way-point to the next. The episode is terminated if the agent hits a leave the game are. Observations are two vectors to the next two subsequent way-points.

7.3.2.4 Space Shooter

Space Shooter is a 2D multi-agent environment with two teams of 8 agents, where the objective is to shoot and destroy the enemy team. Currently observations are an array of relative positions of the enemy and friend teams. Results, observations

and training methodology in the environment are still work in progress and will change in the next version of Godot RL Agents.

7.4 Experiments and Benchmarks

As part of our reference implementation, we provide a PPO example with reasonable default parameters in order to provide a good starting point for training RL agents. We believe this algorithm is a suitable starting point, as it allows for both continuous and discrete action spaces, is robust to hyper-parameter configurations, and is relatively sample efficient. In Figure 7.4 we show example training curves for the Ball Chase environment. We provide pre-trained models for all example environments in our repository, refer to our overview video for examples of these behaviors:

https://youtu.be/g1MlZSFqIj4.

We benchmark the interaction rate of the Jumper environment and achieve 12k environment interactions per second with up to 4 parallel processes, results shown in Figure 7.5. Benchmarking was performed on a relatively high-end laptop computer [1], future work on Godot RL Agents will benchmark on dedicated hardware.

7.5 Conclusions and future work

We have introduced the Godot RL Agents framework, a tool for building 2D and 3D environments in the Godot Game Engine and learning agent behaviors with Deep Reinforcement Learning. We have created 4 example environments to demonstrate the capabilities of this versatile tool. By interfacing with well known open-source Deep RL implementations, we provide access to over 20 RL algorithms out of the box. We have implemented sensors, tools for parallel interaction with environments, and support for continuous, discrete and mixed action spaces.

Future work on the framework aims to include: variable size observations coupled with attention mechanisms, text-based environments, environments with discrete state spaces, virtual monocular cameras for vision-based agent control, multi-modal observations (1D, 2D, text & audio) and model export for in-engine inference of trained behaviors. We plan to extend our focus to multi-agent environments, preliminary work in the *Space Shooter* environment shows this area is a challenging area of research, with applications in both video games and collaborative robotics.

[1] Dell XPS - i7-10750H (6 cores), GTX 1650 Ti, 32 GB RAM

External datasets can also be loaded in to the Godot Game Engine. In Figure 7.6 we load photo-realistic scans from the Gibson (Xia et al. 2018) dataset, to demonstrate that is tool can be used to build realistic 3D scenes for learning robotic control and sim2real transfer. We believe that there are near endless possibilities for this tool and are interesting to discuss with the research community the future direction of this work.

We have now reached the end of the five contributions made during this three-year PhD thesis, the following chapter will summarize our contributions and provide perspectives on future research directions.

CHAPTER 8

CONCLUSION

Contents

8.1 Summary of Contributions

In this manuscript, we have introduced, discussed and explained five contributions in the field of Deep Reinforcement Learning for planning and navigation in large-scale 3D environments.

Building and benchmarking Deep RL environments — Quantifying and benchmarking the limitations of unstructured end-to-end approaches is an ongoing area of research in the field of Deep Reinforcement Learning for vision-based planning and navigation. In chapter 3, we developed a set of challenging procedurally generated scenarios with sparse rewards that require exploration, planning, memory and navigation from pixels, in 3D environments. Our particular focus was on intra-task generalization of learned policies from a known to an unknown environment configuration. We identify which scenario configurations a recurrent baseline agent finds challenging and use this as a foundation for building structured agent architectures. In addition, we quantify how many unique map instances are required during the training of policy, in order to achieve comparable generalization performance in a new map instance.

Improved reasoning with egocentric metric maps — An agent interacting with a 3D world through observations made with a simulated monocular camera has a view of the world that is based on many geometric and physical processes. Its observations are the projection of the 3D world onto a 2D plane and the world around the agent moves relative to its ego-motion. In chapter 4 we demonstrated how the geometric and physical properties can be exploited as part of a spatially structured hidden state, that includes a differentiable inverse projection of Convolutional Neural Network (CNN) feature vectors from screen-space to

world-space. We demonstrated through several benchmark scenarios, developed in chapter 3, that this end-to-end architecture out-performs unstructured recurrent agents in both sample efficiency and asymptotic performance. We conducted an extensive ablation study of EgoMap architectural components, including the self-attention and global read mechanisms. Analysis of the self-attention mechanisms that are built into the agent's structured memory cell provided insight into the agent's reasoning process, we observed that the agent attends to object of interest in order to augment its navigation performance.

Learning to plan with topological memory — As we move to complex, semantically rich and diverse environments, reconstructed from 3D scans of real-world buildings, we must build agents that learn to exploit structural regularities in the layouts of real world buildings, which may not be present in synthetic scenes. In chapter 5, we learn to incorporate free space-estimate and visual features in a graph, collected by an exploratory agent, in order to approximate a shortest path planning algorithm. In this work, instead of structuring our network architecture based on physical properties of our environment, we structure based on the operations of a path planning algorithm - Bellman-Ford. We achieve this with the application of Graph Neural Networks to perform message passing between graph nodes, coupled with a recurrent memory cell per node that integrates incoming messages. A detailed ablation study of the network architecture demonstrated that the addition of this structure is required to achieve an approximation of the classical planning algorithm. We should stipulate, however, that the parameters of our model do not exactly represent the operations of the classical planning algorithm. We also demonstrate that while visual features are required to achieve the best performance, the structure of the connection probability estimates play a large role in the estimation of the shortest path.

Automated construction of topological maps in hectare-scale video games — Video games provide an opportunity to test the limitations of Deep RL algorithms and architectures in enormous environments with complex action spaces, without the concerns of real world robotics such as safety guarantees and sample efficiency. Videos games do come with some additional constraints, inference time of training polices is required to be less than a millisecond on tens or hundreds of parallel agents, ideally agent behaviors should be interpretable to provide designers and game-creators the confidence to deploy a learning-based approach. Game-creators also desire that agents agents should behave in a human-like manner, in order to provide a convincing user-experience. In chapter 6 we quantify the limitations of end-to-end Deep RL approaches for point-to-point navigation in vast environments, through the creation of a large procedurally generated environment called "GameRLand3D". We then extend the performance of a recurrent baseline agent with a hybrid graph-based approach that combines

a classical planning algorithm with the low level recurrent policy, enabling long term planning and navigation.

Building new worlds with an open-source Deep RL interface — As applications of Deep Reinforcement Learning (RL) become ever more present in our society, researchers require software and tools to create environments to prototype new approaches and algorithms. Currently many of these tools are quite restrictive in the types of environments you can build, require expert domain knowledge or are pay to use. In chapter 7 we develop an open-source Deep RL interface for a widely known open-source Game Engine. The tool we have developed allows for researchers, hobbyists and game designers to build and interface with worlds and games free of charge, learning agent behaviors with state-of-the-art Deep RL algorithms. There are several applications of this open-source tool: the construction of 3D worlds and scenarios for research into Deep RL, to build video-games and learn player-facing agent behaviors, and automated testing of video-games and other software applications. We demonstrate this contribution is performant, versatile and provide several documented example implementations of RL environments in both 3D and 2D. By providing a standard Gym interface, we enable researched to directly plug these environments into pre-existing Deep RL implementations.

8.2 Perspectives for future work

The contributions made during this thesis and concurrent works have led to many exciting avenues and future research directions. We now propose some possible directions that could be taken to build upon our publications and the publications of our peers.

Transfer from simulation to reality As the visual quality of simulation reaches near photo-realistic levels, the domain gap between simulation and reality becomes small enough to explore the direct transfer of policies learned in simulation, "sim2real". Several examples of sim2real transfer have been conducted (Anderson et al. 2018c; Kadian et al. 2020; Sadek et al. 2021; Jaunet et al. 2021) which highlighted that transfer is possible with little domain adaptation, the studies also discovered bugs and physical inconsistencies in the underlying environment simulator.

To date, research on sim2real transfer of structured agent representations remains sparse, (Gupta et al. 2017a) perform an empirical analysis of the transfer of structured policies, demonstrating improvements in performance. Other than (Gupta et al. 2017a), to our knowledge, there are few other contributions, indicating that this is an area where fruitful research can be conducted.

Auxiliary losses In vast majority of the works we have published during this thesis, the objective function which we optimize is purely based on reward. While this poses some advantages such as the generality of the approach, it can be prudent to add auxiliary objectives to the loss function of our training loss. This can be particularly beneficial when the auxiliary objective is aligned with the agents primary task. Recent works (Lample et al. 2017b; Jaderberg et al. 2016a) have shown this can accelerate learning and improve the final performance in unstructured recurrent agents. Researchers have already demonstrated that the performance of structured approaches can be improved through the use of auxiliary losses. Indeed, (Marza et al. 2021) demonstrate that the performance of spatially structured memory architectures such as EgoMap (Beeching et al. 2020c) can be improved by added auxiliary losses based on the distance to a target object, the direction of a target object and estimation of whether the current target has previously been observed, their approach came first in the CVPR 2021 Multi-ON challenge (Wani et al. 2020).

Attention-based models The use of attention-based models has revolutionized the field of Natural Language Processing (NLP) Devlin et al. 2018; Wolf et al. 2019a and has begun to make an impact in image-based tasks (Han et al. 2020). While the potential of self-attention has been demonstrated in our EgoMap publication (Beeching et al. 2020c) there has been limited research on the applications of temporal and visual attention in the domain of Deep RL (Parisotto et al. 2020; Chen et al. 2021). The work that has been undertaken has found that incorporating vanilla transformers in a Deep RL agent's architecture and training end-to-end with RL algorithms poses several challenges related to training instability, through residual connections and modification of the transformers gating mechanism the instabilities can be diminished.

Multi-modal applications Multi-model applications in Deep RL enable the combination of image-based observations with textual instructions and have been demonstrated in a number of different environments (Nguyen et al. 2019; Chaplot et al. 2019). Recently published works (Team et al. 2021) extend multi-modal Deep RL to environments where the goal task is defined as a string of text, enabling the learning of policies that generalize to unseen tasks sample from a universe of possible task/environment combinations. Further extensions of this work could combine representations that are structured in both the visual and text domains, incorporating cross-modal attention to enable higher performance and increased sampled efficiency.

BIBLIOGRAPHY

Abadi, Martín, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng (2015). *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. URL: https://www.tensorflow.org/ (cit. on pp. 2, 44).

Achiam, Joshua (2018). *OpenAI Spinning Up*. URL: https://spinningup.openai.com/en/latest/index.html (visited on 11/26/2021) (cit. on p. 36).

Agarwal, Rishabh, Max Schwarzer, Pablo Samuel Castro, Aaron Courville, and Marc G Bellemare (2021). "Deep Reinforcement Learning at the Edge of the Statistical Precipice". In: *Advances in Neural Information Processing Systems* (cit. on p. 50).

Alonso, Eloi, Maxim Peter, David Goumard, and Joshua Romoff (Aug. 2021). "Deep Reinforcement Learning for Navigation in AAA Video Games". In: *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI-21*. Ed. by Zhi-Hua Zhou. Main Track. International Joint Conferences on Artificial Intelligence Organization, pp. 2133–2139. URL: https://doi.org/10.24963/ijcai.2021/294 (cit. on pp. 114, 115, 117, 121, 125, 132).

Anderson, Peter, Angel Chang, Devendra Singh Chaplot, Alexey Dosovitskiy, Saurabh Gupta, Vladlen Koltun, Jana Kosecka, Jitendra Malik, Roozbeh Mottaghi, Manolis Savva, et al. (2018a). *On evaluation of embodied navigation agents* (cit. on pp. 47, 49, 107, 125).

Anderson, Peter, Qi Wu, Damien Teney, Jake Bruce, Mark Johnson, Niko Sünderhauf, Ian Reid, Stephen Gould, and Anton den Hengel (2018b). "Vision-and-Language Navigation: Interpreting visually-grounded navigation instructions in real environments". In: *CVPR* (cit. on pp. 66, 88).

Anderson, Peter, Qi Wu, Damien Teney, Jake Bruce, Mark Johnson, Niko Sünderhauf, Ian Reid, Stephen Gould, and Anton Van Den Hengel (2018c). "Vision-and-language navigation: Interpreting visually-grounded navigation instructions in real environments". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 3674–3683 (cit. on p. 144).

Åström, Karl Johan (1965). "Optimal control of Markov processes with incomplete state information I". In: *Journal of Mathematical Analysis and Applications* 10, pp. 174–205 (cit. on pp. 33, 98, 131).

Axelrod, Ramon (2008). "Navigating graph generation in highly dynamic worlds". In: vol. 4. AI Game Programming Wisdom. Charles River Media. Chap. 2.6, pp. 125–141 (cit. on p. 115).

Baccouche, Moez, Franck Mamalet, Christian Wolf, Christophe Garcia, and Atilla Baskurt (2011). "Sequential deep learning for human action recognition". In: *International workshop on human behavior understanding*. Springer, pp. 29–39 (cit. on p. 22).

Bahdanau, Dzmitry, Kyunghyun Cho, and Yoshua Bengio (2014). "Neural machine translation by jointly learning to align and translate". In: *arXiv preprint arXiv:1409.0473* (cit. on pp. 31, 87).

Banino, Andrea, Caswell Barry, Benigno Uria, Charles Blundell, Timothy Lillicrap, Piotr Mirowski, Alexander Pritzel, Martin J Chadwick, Thomas Degris, Joseph Modayil, et al. (2018). "Vector-based navigation using grid-like representations in artificial agents". In: *Nature* 557.7705, pp. 429–433 (cit. on pp. 56, 83).

Baradel, Fabien, Natalia Neverova, Julien Mille, Greg Mori, and Christian Wolf (2020). "CoPhy: Counterfactual Learning of Physical Dynamics". In: *International Conference on Learning Representations*. URL: https://openreview.net/forum?id=SkeyppEFvS (cit. on p. 26).

Barto, Andrew G (2021). "Reinforcement Learning: An Introduc-tion. By Richard S. Sutton". In: *SIAM Review Vol. 63, Issue 2 (June 2021)* 63.2, p. 423 (cit. on p. 12).

Battaglia, Peter, Razvan Pascanu, Matthew Lai, Danilo Jimenez Rezende, et al. (2016). "Interaction networks for learning about objects, relations and physics". In: *Advances in neural information processing systems*, pp. 4502–4510 (cit. on p. 99).

Battaglia, Peter W, Jessica B Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, et al. (2018a). "Relational inductive biases, deep learning, and graph networks". In: *arXiv preprint arXiv:1806.01261* (cit. on pp. 26, 27, 99).

Battaglia, P.W., J.B. Hamrick, V. Bapst, A. Sanchez-Gonzalez, V.F. Zambaldi, M. Malinowski, A. Tacchetti, D. Raposo, A. Santoro, R. Faulkner, Ç. Gülçehre, F. Song, A.J. Ballard, J. Gilmer, G.E. Dahl, A. Vaswani, K. Allen, C. Nash, V/ Langston, C. Dyer, N. Heess, D. Wierstra, P. Kohli, M. Botvinick, O. Vinyals, Y. Li, and R. Pascanu (2018b). "Relational inductive biases, deep learning, and graph networks". In: *arXiv preprint 1807.09244* (cit. on p. 102).

Baydin, Atilim Gunes, Barak A. Pearlmutter, Alexey Andreyevich Radul, and Jeffrey Mark Siskind (2018). *Automatic differentiation in machine learning: a survey*. arXiv: 1502.05767 [cs.SC] (cit. on p. 28).

Beattie, Charles, Joel Z Leibo, Denis Teplyashin, Tom Ward, Marcus Wainwright, Heinrich Küttler, Andrew Lefrancq, Simon Green, Víctor Valdés, Amir Sadik,

et al. (2016a). "Deepmind lab". In: *arXiv preprint arXiv:1612.03801* (cit. on pp. 1, 2, 93).

Beattie, Charles, Joel Z. Leibo, Denis Teplyashin, Tom Ward, Marcus Wainwright, Heinrich Küttler, Andrew Lefrancq, Simon Green, Víctor Valdés, Amir Sadik, Julian Schrittwieser, Keith Anderson, Sarah York, Max Cant, Adam Cain, Adrian Bolton, Stephen Gaffney, Helen King, Demis Hassabis, Shane Legg, and Stig Petersen (2016b). "DeepMind Lab". In: *arxiv pre-print 1612.03801* (cit. on pp. 46, 66, 67, 116, 132).

Beeching, Edward, Jilles Dibangoye, Olivier Simonin, and Christian Wolf (2020a). "Deep Reinforcement Learning on a Budget: 3D Control and Reasoning Without a Supercomputer." In: *Proceedings of the International Conference on Pattern Recognition (ICPR)* (cit. on pp. 7, 63).

Beeching, Edward, Jilles Dibangoye, Olivier Simonin, and Christian Wolf (2020b). "Deep Reinforcement Learning on a Budget: 3D Control and Reasoning Without a Supercomputer." In: *International Conference on Pattern Recognition* (cit. on p. 88).

Beeching, Edward, Jilles Dibangoye, Olivier Simonin, and Christian Wolf (2020c). "EgoMap: Projective mapping and structured egocentric memory for Deep RL". In: *Proceedings of the European Conference on Machine Learning and Knowledge Discovery in Databases (ECML-PKDD)* (cit. on pp. 8, 59, 79, 99, 118, 132, 144).

Beeching, Edward, Jilles Dibangoye, Olivier Simonin, and Christian Wolf (2020d). "Learning to plan with uncertain topological maps." In: *Proceedings of the IEEE European conference on computer vision (ECCV)* (cit. on pp. 8, 95).

Beeching, Edward, Jilles Dibangoye, Olivier Simonin, and Christian Wolf (2020e). "Learning to plan with uncertain topological maps". In: *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part III 16*. Springer, pp. 473–490 (cit. on pp. 61, 118, 132).

Beeching, Edward, Jilles Dibangoye, Olivier Simonin, and Christian Wolf (2022a). "Godot Reinforcement Learning Agents". In: *Association for the Advancement of Artificial Intelligence Conference on Artificial Intelligence (AAAI). Workshop on Reinforcement Learning in Games* (cit. on pp. 8, 129).

Beeching, Edward, Maxim Peter, Philippe Marcotte, Jilles Dibangoye, Olivier Simonin, Romoff Joshua, and Christian Wolf (2022b). "Graph augmented Deep Reinforcement Learning in the GameRLand3D environment". In: *Association for the Advancement of Artificial Intelligence Conference on Artificial Intelligence (AAAI). Workshop on Reinforcement Learning in Games* (cit. on pp. 8, 113).

Bellemare, Marc G., Will Dabney, and Rémi Munos (2017). "A distributional perspective on reinforcement learning". In: *Proceedings of the Thirty-Fourth International Conference on Machine Learning* (cit. on p. 41).

Bellman, RICHARD (1957). "Dynamic programming, princeton univ". In: *Press Princeton, New Jersey* (cit. on p. 57).

Bellman, Richard (1958). "On a routing problem". In: *Quarterly of applied mathematics* 16.1, pp. 87–90 (cit. on p. 97).

Bengio, Yoshua, Patrice Simard, and Paolo Frasconi (1994). "Learning long-term dependencies with gradient descent is difficult". In: *IEEE transactions on neural networks* 5.2, pp. 157–166 (cit. on p. 20).

Bhatti, Shehroze, Alban Desmaison, Ondrej Miksik, Nantas Nardelli, N Siddharth, and Philip HS Torr (2016a). "Playing doom with slam-augmented deep reinforcement learning". In: *arXiv preprint arXiv:1612.00380* (cit. on pp. 82, 84, 99).

Bhatti, Shehroze, Alban Desmaison, Ondrej Miksik, Nantas Nardelli, N. Siddharth, and Philip HS Torr (2016b). "Playing Doom with slam-augmented deep reinforcement learning". In: *arXiv preprint arXiv:1612.00380* (cit. on p. 118).

Bloesch, Michael, Jan Czarnowski, Ronald Clark, Stefan Leutenegger, and Andrew J Davison (2018). "CodeSLAM—learning a compact, optimisable representation for dense visual SLAM". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2560–2568 (cit. on p. 51).

Bowman, Sean L, Nikolay Atanasov, Kostas Daniilidis, and George J Pappas (2017). "Probabilistic data association for semantic slam". In: *2017 IEEE international conference on robotics and automation (ICRA)*. IEEE, pp. 1722–1729 (cit. on p. 51).

Bradbury, James, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang (2018). *JAX: composable transformations of Python+NumPy programs*. Version 0.2.5. URL: http://github.com/google/jax (cit. on pp. 3, 45).

Brockman, Greg, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba (2016). "OpenAI Gym". In: *arxiv pre-print 1606.01540* (cit. on pp. 46, 67, 116, 132).

Brodeur, Simon, Ethan Perez, Ankesh Anand, Florian Golemo, Luca Celotti, Florian Strub, Jean Rouat, Hugo Larochelle, and Aaron Courville (2018). "HoME: a Household Multimodal Environment". In: *ICLR* (cit. on pp. 65, 66, 88).

Bronstein, Michael M, Joan Bruna, Yann LeCun, Arthur Szlam, and Pierre Vandergheynst (2017). "Geometric deep learning: going beyond euclidean data". In: *IEEE Signal Processing Magazine* 34.4, pp. 18–42 (cit. on p. 99).

Budde, Sara (2013). "Automatic generation of jump links in arbitrary 3d environments". MA thesis. Humboldt University of Berlin (cit. on p. 115).

Chang, Angel, Angela Dai, Thomas Funkhouser, Maciej Halber, Matthias Niessner, Manolis Savva, Shuran Song, Andy Zeng, and Yinda Zhang (2017). "Matterport3D: Learning from RGB-D Data in Indoor Environments". In: *International Conference on 3D Vision (3DV)* (cit. on p. 47).

Chang, Michael B, Tomer Ullman, Antonio Torralba, and Joshua B Tenenbaum (2016). "A compositional object-based approach to learning physical dynamics". In: *arXiv preprint arXiv:1612.00341* (cit. on p. 26).

Chaplot, Devendra Singh, Dhiraj Gandhi, Saurabh Gupta, Abhinav Gupta, and Ruslan Salakhutdinov (2020a). "Learning To Explore Using Active Neural SLAM". In: *International Conference on Learning Representations*. URL: https://openreview.net/forum?id=HklXn1BKDH (cit. on pp. 59, 61, 105).

Chaplot, Devendra Singh, Dhiraj Gandhi, Saurabh Gupta, Abhinav Gupta, and Ruslan Salakhutdinov (2020b). "Learning To Explore Using Active Neural SLAM". In: *International Conference on Learning Representations*. URL: https://openreview.net/forum?id=HklXn1BKDH (cit. on p. 132).

Chaplot, Devendra Singh, Dhiraj Prakashchand Gandhi, Abhinav Gupta, and Russ R Salakhutdinov (2020c). "Object goal navigation using goal-oriented semantic exploration". In: *Advances in Neural Information Processing Systems* 33 (cit. on p. 132).

Chaplot, Devendra Singh, Helen Jiang, Saurabh Gupta, and Abhinav Gupta (2020d). "Semantic curiosity for active visual learning". In: *European Conference on Computer Vision*. Springer, pp. 309–326 (cit. on pp. 59, 132).

Chaplot, Devendra Singh, Lisa Lee, Ruslan Salakhutdinov, Devi Parikh, and Dhruv Batra (2019). "Embodied multimodal multitask learning". In: *arXiv preprint arXiv:1902.01385* (cit. on p. 145).

Chaplot, Devendra Singh, Ruslan Salakhutdinov, Abhinav Gupta, and Saurabh Gupta (2020e). "Neural Topological SLAM for Visual Navigation". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (cit. on pp. 61, 119).

Chaplot, Devendra Singh, Ruslan Salakhutdinov, Abhinav Gupta, and Saurabh Gupta (2020f). "Neural topological slam for visual navigation". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 12875–12884 (cit. on p. 132).

Chen, Liang-Chieh, Maxwell D. Collins, Yukun Zhu, George Papandreou, Barret Zoph, Florian Schroff, Hartwig Adam, and Jonathon Shlens (2018a). "Searching for Efficient Multi-Scale Architectures for Dense Image Prediction". In: *NIPS* (cit. on p. 31).

Chen, Liang-Chieh, Yukun Zhu, George Papandreou, Florian Schroff, and Hartwig Adam (2018b). "Encoder-Decoder with Atrous Separable Convolution for Semantic Image Segmentation". In: *ECCV* (cit. on p. 31).

Chen, Lili, Kevin Lu, Aravind Rajeswaran, Kimin Lee, Aditya Grover, Michael Laskin, Pieter Abbeel, Aravind Srinivas, and Igor Mordatch (2021). "Decision transformer: Reinforcement learning via sequence modeling". In: *arXiv preprint arXiv:2106.01345* (cit. on p. 144).

Chen, Tao, Saurabh Gupta, and Abhinav Gupta (2019a). "Learning Exploration Policies for Navigation". In: *International Conference on Learning Representations*. URL: https://openreview.net/forum?id=SyMWn05F7 (cit. on p. 84).

Chen, Tao, Saurabh Gupta, and Abhinav Gupta (2019b). "Learning Exploration Policies for Navigation". In: *International Conference on Learning Representations*. URL: https://openreview.net/forum?id=SyMWn05F7 (cit. on p. 105).

Chen, Tianqi, Mu Li, Yutian Li, Min Lin, Naiyan Wang, Minjie Wang, Tianjun Xiao, Bing Xu, Chiyuan Zhang, and Zheng Zhang (2015). *MXNet: A Flexible and Efficient Machine Learning Library for Heterogeneous Distributed Systems*. arXiv: 1512.01274 [cs.DC] (cit. on p. 3).

Choi, Sungha, Joanne T Kim, and Jaegul Choo (2020). "Cars can't fly up in the sky: Improving urban-scene segmentation via height-driven attention networks". In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 9373–9383 (cit. on p. 31).

Christodoulou, Petros (2019). "Soft actor-critic for discrete action settings". In: *arXiv preprint arXiv:1910.07207* (cit. on pp. 43, 136).

Chung, Junyoung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio (2014). "Empirical evaluation of gated recurrent neural networks on sequence modeling". In: *arXiv preprint arXiv:1412.3555* (cit. on pp. 22, 33, 103).

Chung, Junyoung, Caglar Gulcehre, Kyunghyun Cho, and Yoshua Bengio (2015a). "Gated Feedback Recurrent Neural Networks". In: *ICML* (cit. on pp. 67, 82, 85, 99, 132).

Chung, Junyoung, Caglar Gulcehre, Kyunghyun Cho, and Yoshua Bengio (2015b). "Gated feedback recurrent neural networks". In: *International conference on machine learning*. PMLR, pp. 2067–2075 (cit. on p. 118).

Civera, J., D. Galvez-Lopez, L. Riazuelo, J. D. Tardós, and J. M. M. Montiel (2011a). "Towards semantic SLAM using a monocular camera". In: *IROS* (cit. on p. 69).

Civera, J., D. Galvez-Lopez, L. Riazuelo, J. D. Tardós, and J. M. M. Montiel (2011b). "Towards semantic SLAM using a monocular camera". In: *IROS* (cit. on pp. 80, 84).

Clevert, Djork-Arné, Thomas Unterthiner, and Sepp Hochreiter (2015). "Fast and accurate deep network learning by exponential linear units (elus)". In: *arXiv preprint arXiv:1511.07289* (cit. on p. 17).

Cueva, Christopher J. and Xue-Xin Wei (2018). "Emergence of grid-like representations by training recurrent neural networks to perform spatial localization". In: *International Conference on Learning Representations*. URL: https://openreview.net/forum?id=B17JTOe0- (cit. on pp. 56, 57, 83, 123).

Dai, Zihang, Hanxiao Liu, Quoc V Le, and Mingxing Tan (2021). "CoAtNet: Marrying Convolution and Attention for All Data Sizes". In: *arXiv preprint arXiv:2106.04803* (cit. on p. 30).

Das, A., G. Gkioxari, S. Lee, D. Parikh, and D. Batra (2018). "Neural Modular Control for Embodied Question Answering". In: *CVPR* (cit. on p. 64).

Dauphin, Yann N, Angela Fan, Michael Auli, and David Grangier (2017). "Language modeling with gated convolutional networks". In: *Proceedings of the 34th*

International Conference on Machine Learning-Volume 70. JMLR. org, pp. 933–941 (cit. on p. 103).

Devlin, Jacob, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova (2018). "Bert: Pre-training of deep bidirectional transformers for language understanding". In: *arXiv preprint arXiv:1810.04805* (cit. on pp. 24, 31, 144).

Devlin, Sam, Raluca Georgescu, Ida Momennejad, Jaroslaw Rzepecki, Evelyn Zuniga, Gavin Costello, Guy Leroy, Ali Shaw, and Katja Hofmann (2021). "Navigation Turing Test (NTT): Learning to Evaluate Human-Like Navigation". In: *2021 International Conference on Machine Learning*. Source code available at: https://github.com/microsoft/NTT. URL: https://www.microsoft.com/en-us/research/publication/navigation-turing-test-ntt-learning-to-evaluate-human-like-navigation/ (cit. on p. 120).

Dhariwal, Prafulla, Christopher Hesse, Oleg Klimov, Alex Nichol, Matthias Plappert, Alec Radford, John Schulman, Szymon Sidor, Yuhuai Wu, and Peter Zhokhov (2017). *OpenAI Baselines*. https://github.com/openai/baselines (cit. on p. 67).

Dijkstra, Edsger W (1959). "A note on two problems in connexion with graphs". In: *Numerische mathematik* 1.1, pp. 269–271 (cit. on pp. 51, 97, 105, 123).

Dosovitskiy, Alexey, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun (2017). "CARLA: An Open Urban Driving Simulator". In: *CoRL*, pp. 1–16 (cit. on p. 67).

Dozat, Timothy (2016). "Incorporating nesterov momentum into adam". In: (cit. on p. 29).

Dumoulin, Vincent and Francesco Visin (2018). *A guide to convolution arithmetic for deep learning*. arXiv: 1603.07285 [stat.ML] (cit. on p. 20).

Durrant-Whyte, Hugh and Tim Bailey (2006). "Simultaneous localization and mapping: part I". In: *IEEE robotics & automation magazine* 13.2, pp. 99–110 (cit. on pp. 51, 52).

Duvenaud, David, Dougal Maclaurin, Jorge Aguilera-Iparraguirre, Rafael Gómez-Bombarelli, Timothy Hirzel, Alán Aspuru-Guzik, and Ryan P Adams (2015). "Convolutional networks on graphs for learning molecular fingerprints". In: *arXiv preprint arXiv:1509.09292* (cit. on p. 26).

Elman, Jeffrey L (1990). "Finding structure in time". In: *Cognitive science* 14.2, pp. 179–211 (cit. on p. 20).

Engel, Jakob, Thomas Schöps, and Daniel Cremers (2014). "LSD-SLAM: Large-scale direct monocular SLAM". In: *European conference on computer vision*. Springer, pp. 834–849 (cit. on p. 51).

Espeholt, Lasse, Raphaël Marinier, Piotr Stanczyk, Ke Wang, and Marcin Michalski (2019). "SEED RL: Scalable and Efficient Deep-RL with Accelerated Central Inference". In: arXiv: 1910.06591 [cs.LG] (cit. on p. 134).

Espeholt, Lasse, Hubert Soyer, Remi Munos, Karen Simonyan, Volodymir Mnih, Tom Ward, Yotam Doron, Vlad Firoiu, Tim Harley, Iain Dunning, Shane Legg,

and Koray Kavukcuoglu (2018a). "IMPALA: Scalable Distributed Deep-RL with Importance Weighted Actor-Learner Architectures". In: *ICML* (cit. on pp. 67, 68, 70).

Espeholt, Lasse, Hubert Soyer, Remi Munos, Karen Simonyan, Volodymir Mnih, Tom Ward, Yotam Doron, Vlad Firoiu, Tim Harley, Iain Dunning, et al. (2018b). "Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures". In: (cit. on p. 45).

Espeholt, Lasse, Hubert Soyer, Remi Munos, Karen Simonyan, Volodymir Mnih, Tom Ward, Yotam Doron, Vlad Firoiu, Tim Harley, Iain Dunning, et al. (2018c). "IMPALA: Scalable Distributed Deep-RL with Importance Weighted Actor-Learner Architectures". In: *Proceedings of the International Conference on Machine Learning (ICML)* (cit. on pp. 132, 135).

Eysenbach, Ben, Russ R. Salakhutdinov, and Sergey Levine (2019a). "Search on the replay buffer: Bridging planning and reinforcement learning". In: *Advances in Neural Information Processing Systems* (cit. on pp. 60, 61, 118).

Eysenbach, Ben, Russ R Salakhutdinov, and Sergey Levine (2019b). "Search on the Replay Buffer: Bridging Planning and Reinforcement Learning". In: *Advances in Neural Information Processing Systems 32*. Curran Associates, Inc., pp. 15220–15231 (cit. on p. 99).

Fang, Kuan, Alexander Toshev, Li Fei-Fei, and Silvio Savarese (2019). "Scene memory transformer for embodied agents in long-horizon tasks". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 538–547 (cit. on pp. 59, 61, 82).

Fernandez, Juan A and Javier Gonzalez (1998). "Hierarchical graph search for mobile robot path planning". In: *Proceedings. 1998 IEEE International Conference on Robotics and Automation (Cat. No. 98CH36146)*. Vol. 1. IEEE, pp. 656–661 (cit. on p. 53).

Fernández-Madrigal, J-A and Javier González (2002). "Multihierarchical graph search". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 24.1, pp. 103–113 (cit. on p. 53).

Fortunato, Meire, Mohammad Gheshlaghi Azar, Bilal Piot, Jacob Menick, Ian Osband, Alex Graves, Vlad Mnih, Remi Munos, Demis Hassabis, Olivier Pietquin, Charles Blundell, and Shane Legg (2017). "Noisy networks for exploration". In: *arXiv preprint arXiv:1706.10295* (cit. on p. 41).

Fout, Alex, Jonathon Byrd, Basir Shariat, and Asa Ben-Hur (2017). "Protein interface prediction using graph convolutional networks". In: *Advances in neural information processing systems*, pp. 6530–6539 (cit. on p. 99).

Frégnac, Yves, Alice René, Jean Baptiste Durand, and Yves Trotter (2004). "Brain encoding and representation of 3D-space using different senses, in different species". In: *Journal of physiology, Paris* 98.1-3, pp. 1–18 (cit. on p. 85).

Fujimoto, Scott, Herke Hoof, and David Meger (2018). "Addressing function approximation error in actor-critic methods". In: *International Conference on Machine Learning*. PMLR, pp. 1587–1596 (cit. on p. 45).

Fukushima, Kunihiko and Sei Miyake (1982). "Neocognitron: A self-organizing neural network model for a mechanism of visual pattern recognition". In: *Competition and cooperation in neural nets*. Springer, pp. 267–285 (cit. on pp. 17, 19, 131).

Garg, Siddhant, Thuy Vu, and Alessandro Moschitti (2020). "Tanda: Transfer and adapt pre-trained transformer models for answer sentence selection". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 34. 05, pp. 7780–7788 (cit. on p. 31).

Gill, Philip E, Walter Murray, and Margaret H Wright (2019). *Practical optimization*. SIAM (cit. on p. 11).

Gilmer, Justin, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl (2017). "Neural message passing for quantum chemistry". In: *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org, pp. 1263–1272 (cit. on p. 99).

Gisslén, Linus, Andy Eakins, Camilo Gordillo, Joakim Bergdahl, and Konrad Tollmar (2021). "Adversarial reinforcement learning for procedural content generation". In: *arXiv preprint arXiv:2103.04847* (cit. on p. 132).

Gordillo, Camilo, Joakim Bergdahl, Konrad Tollmar, and Linus Gisslén (2021). *Improving Playtesting Coverage via Curiosity Driven Reinforcement Learning Agents*. arXiv: 2103.13798 [cs.LG] (cit. on pp. 117, 132).

Gordon, Daniel, Aniruddha Kembhavi, Mohammad Rastegari, Joseph Redmon, Dieter Fox, and Ali Farhadi (2018). *IQA: Visual Question Answering in Interactive Environments*. Tech. rep. (cit. on pp. 83, 84).

Graves, Alex, Greg Wayne, and Ivo Danihelka (2014). "Neural turing machines". In: *arXiv preprint arXiv:1410.5401* (cit. on pp. 25, 82, 99, 118).

Graves, Alex, Greg Wayne, Malcolm Reynolds, Tim Harley, Ivo Danihelka, Agnieszka Grabska-Barwińska, Sergio Gómez Colmenarejo, Edward Grefenstette, Tiago Ramalho, John Agapiou, et al. (2016). "Hybrid computing using a neural network with dynamic external memory". In: *Nature* 538.7626, pp. 471–476 (cit. on pp. 25, 26, 82, 99, 118).

Gu, Jiuxiang, Gang Wang, Jianfei Cai, and Tsuhan Chen (2017). "An empirical study of language cnn for image captioning". In: *Proceedings of the IEEE International Conference on Computer Vision*, pp. 1222–1231 (cit. on p. 31).

Gulcehre, Caglar, Misha Denil, Mateusz Malinowski, Ali Razavi, Razvan Pascanu, Karl Moritz Hermann, Peter Battaglia, Victor Bapst, David Raposo, Adam Santoro, et al. (2018). "Hyperbolic attention networks". In: *arXiv preprint arXiv:1805.09786* (cit. on p. 26).

Gupta, Saurabh, James Davidson, Sergey Levine, Rahul Sukthankar, and Jitendra Malik (2017a). "Cognitive mapping and planning for visual navigation". In:

Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (cit. on pp. 5, 6, 57, 58, 118, 144).

Gupta, Saurabh, James Davidson, Sergey Levine, Rahul Sukthankar, and Jitendra Malik (2017b). "Cognitive mapping and planning for visual navigation". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2616–2625 (cit. on pp. 83, 84, 99).

Gupta, Saurabh, David Fouhey, Sergey Levine, and Jitendra Malik (2017c). "Unifying map and landmark based representations for visual navigation". In: *arXiv preprint arXiv:1712.08125* (cit. on p. 95).

Haarnoja, Tuomas, Aurick Zhou, Kristian Hartikainen, George Tucker, Sehoon Ha, Jie Tan, Vikash Kumar, Henry Zhu, Abhishek Gupta, Pieter Abbeel, et al. (2018a). "Soft actor-critic algorithms and applications". In: *arXiv preprint arXiv:1812.05905* (cit. on pp. 39, 43, 45, 122, 136).

Haarnoja, Tuomas, Aurick Zhou, Kristian Hartikainen, George Tucker, Sehoon Ha, Jie Tan, Vikash Kumar, Henry Zhu, Abhishek Gupta, Pieter Abbeel, et al. (2018b). "Soft actor-critic algorithms and applications". In: *arXiv preprint arXiv:1812.05905* (cit. on p. 122).

Hafner, Danijar, Timothy Lillicrap, Mohammad Norouzi, and Jimmy Ba (2020). "Mastering atari with discrete world models". In: *arXiv preprint arXiv:2010.02193* (cit. on p. 114).

Hafting, Torkel, Marianne Fyhn, Sturla Molden, May-Britt Moser, and Edvard I Moser (2005). "Microstructure of a spatial map in the entorhinal cortex". In: *Nature* 436.7052, pp. 801–806 (cit. on pp. 56, 83).

Hakenes, Simon (2018). *ViZDoom Gym*. https://github.com/shakenes/vizdoomgym (cit. on p. 47).

Hamrick, Jessica B, Kelsey R Allen, Victor Bapst, Tina Zhu, Kevin R McKee, Joshua B Tenenbaum, and Peter W Battaglia (2018). "Relational inductive bias for physical construction in humans and machines". In: *arXiv preprint arXiv:1806.01203* (cit. on p. 26).

Han, Kai, Yunhe Wang, Hanting Chen, Xinghao Chen, Jianyuan Guo, Zhenhua Liu, Yehui Tang, An Xiao, Chunjing Xu, Yixing Xu, et al. (2020). "A survey on visual transformer". In: *arXiv preprint arXiv:2012.12556* (cit. on p. 144).

Hart, Peter, Nils Nilsson, and Bertram Raphael (1968a). "A Formal Basis for the Heuristic Determination of Minimum Cost Paths". In: *IEEE Transactions on Systems Science and Cybernetics* 4.2, pp. 100–107. URL: https://doi.org/10.1109/tssc.1968.300136 (cit. on p. 117).

Hart, Peter E., Nils J. Nilsson, and Bertram Raphael (1968b). "A formal basis for the heuristic determination of minimum cost paths". In: *IEEE Transactions on Systems Science and Cybernetics* SSC-4(2), pp. 100–107 (cit. on pp. 51, 115).

Hasselt, Hado (2010). "Double Q-learning". In: *Advances in neural information processing systems* 23, pp. 2613–2621 (cit. on p. 40).

He, Kaiming, Xiangyu Zhang, Shaoqing Ren, and Jian Sun (2015). "Spatial pyramid pooling in deep convolutional networks for visual recognition". In: *IEEE transactions on pattern analysis and machine intelligence* 37.9, pp. 1904–1916 (cit. on p. 30).

He, Kaiming, Xiangyu Zhang, Shaoqing Ren, and Jian Sun (2016). "Deep residual learning for image recognition". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778 (cit. on pp. 19, 22).

Henderson, Peter, Riashat Islam, Philip Bachman, Joelle Pineau, Doina Precup, and David Meger (2017). "Deep Reinforcement Learning that Matters". In: *arXiv preprint arXiv:1709.06560* (cit. on p. 3).

Henderson, Peter, Riashat Islam, Philip Bachman, Joelle Pineau, Doina Precup, and David Meger (2018). "Deep Reinforcement Learning that Matters". In: *AAAI* (cit. on pp. 36, 49, 50, 69).

Henriques, Joao F. and Andrea Vedaldi (2018a). "MapNet: An Allocentric Spatial Memory for Mapping Environments". In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. URL: http://www.robots.ox.ac.uk/~vgg/publications/2018/henriques18mapnet/henriques18mapnet.pdf (cit. on p. 58).

Henriques, Joao F and Andrea Vedaldi (2018b). "Mapnet: An allocentric spatial memory for mapping environments". In: *proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 8476–8484 (cit. on pp. 83, 84).

Henry, Peter, Michael Krainin, Evan Herbst, Xiaofeng Ren, and Dieter Fox (2014). "RGB-D mapping: Using depth cameras for dense 3D modeling of indoor environments". In: *Experimental robotics*. Springer, pp. 477–491 (cit. on p. 83).

Hessel, Matteo, Joseph Modayil, Hado Van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Azar, and David Silver (2018a). "Rainbow: Combining improvements in deep reinforcement learning". In: (cit. on pp. 39, 42, 114, 136).

Hessel, Matteo, Joseph Modayil, Hado Van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Azar, and David Silver (2018b). "Rainbow: Combining improvements in deep reinforcement learning". In: *Thirty-second AAAI conference on artificial intelligence* (cit. on p. 41).

Hestenes, Magnus Rudolph, Eduard Stiefel, et al. (1952). *Methods of conjugate gradients for solving linear systems*. Vol. 49. 1. NBS Washington, DC (cit. on p. 29).

Hill, Ashley, Antonin Raffin, Maximilian Ernestus, Adam Gleave, Anssi Kanervisto, Rene Traore, Prafulla Dhariwal, Christopher Hesse, Oleg Klimov, Alex Nichol, Matthias Plappert, Alec Radford, John Schulman, Szymon Sidor, and Yuhuai Wu (2018). *Stable Baselines*. https://github.com/hill-a/stable-baselines (cit. on pp. 44, 130, 134).

Hinton, Geoffrey, Nitish Srivastava, and Kevin Swersky (2012a). "Neural networks for machine learning lecture 6a overview of mini-batch gradient descent". In: *Cited on* 14.8, p. 2 (cit. on p. 29).

Hinton, Geoffrey, Nitish Srivastava, and Kevin Swersky (2012b). "Rmsprop: Divide the gradient by a running average of its recent magnitude". In: *Neural networks for machine learning, Coursera lecture 6e*, p. 13 (cit. on p. 89).

Hochreiter, Sepp, Yoshua Bengio, Paolo Frasconi, Jürgen Schmidhuber, et al. (2001). *Gradient flow in recurrent nets: the difficulty of learning long-term dependencies* (cit. on p. 20).

Hochreiter, Sepp and Jürgen Schmidhuber (1997a). "Long short-term memory". In: *Neural computation* 9.8, pp. 1735–1780 (cit. on pp. 3, 21, 33, 118).

Hochreiter, Sepp and Jürgen Schmidhuber (1997b). "Long short-term memory". In: *Neural computation* 9.8, pp. 1735–1780 (cit. on pp. 67, 82, 83, 99, 132).

Hoffman, Matt, Bobak Shahriari, John Aslanides, Gabriel Barth-Maron, Feryal Behbahani, Tamara Norman, Abbas Abdolmaleki, Albin Cassirer, Fan Yang, Kate Baumli, Sarah Henderson, Alex Novikov, Sergio Gómez Colmenarejo, Serkan Cabi, Caglar Gulcehre, Tom Le Paine, Andrew Cowie, Ziyu Wang, Bilal Piot, and Nando de Freitas (2020). "Acme: A Research Framework for Distributed Reinforcement Learning". In: *arXiv preprint arXiv:2006.00979*. URL: https://arxiv.org/abs/2006.00979 (cit. on pp. 44, 134).

Horgan, Dan, John Quan, David Budden, Gabriel Barth-Maron, Matteo Hessel, Hado Van Hasselt, and David Silver (2018). "Distributed prioritized experience replay". In: *arXiv preprint arXiv:1803.00933* (cit. on p. 136).

Howard, Andrew, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, Quoc V. Le, and Hartwig Adam (2019). *Searching for MobileNetV3*. arXiv: 1905.02244 [cs.CV] (cit. on p. 19).

Howard, Jeremy and Sebastian Ruder (2018). "Universal language model fine-tuning for text classification". In: *arXiv preprint arXiv:1801.06146* (cit. on p. 31).

Huang, Gao, Zhuang Liu, Laurens van der Maaten, and Kilian Q Weinberger (2017). "Densely Connected Convolutional Networks". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (cit. on p. 19).

Iandola, Forrest N, Song Han, Matthew W Moskewicz, Khalid Ashraf, William J Dally, and Kurt Keutzer (2016). "SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and< 0.5 MB model size". In: *arXiv preprint arXiv:1602.07360* (cit. on p. 19).

Izadi, Shahram, David Kim, Otmar Hilliges, David Molyneaux, Richard Newcombe, Pushmeet Kohli, Jamie Shotton, Steve Hodges, Dustin Freeman, Andrew Davison, et al. (2011). "KinectFusion: real-time 3D reconstruction and interaction using a moving depth camera". In: *Proceedings of the 24th annual ACM symposium on User interface software and technology*, pp. 559–568 (cit. on p. 83).

Jaderberg, Max, Wojciech M Czarnecki, Iain Dunning, Luke Marris, Guy Lever, Antonio Garcia Castaneda, Charles Beattie, Neil C Rabinowitz, Ari S Morcos, Avraham Ruderman, et al. (2019). "Human-level performance in 3D multiplayer

games with population-based reinforcement learning". In: *Science* 364.6443, pp. 859–865 (cit. on p. 114).

Jaderberg, Max, Volodymyr Mnih, Wojciech Marian Czarnecki, Tom Schaul, Joel Z Leibo, David Silver, and Koray Kavukcuoglu (2016a). "Reinforcement learning with unsupervised auxiliary tasks". In: (cit. on pp. 67, 81, 82, 85, 98, 131, 144).

Jaderberg, Max, Volodymyr Mnih, Wojciech Marian Czarnecki, Tom Schaul, Joel Z. Leibo, David Silver, and Koray Kavukcuoglu (2016b). "Reinforcement Learning with Unsupervised Auxiliary Tasks". In: (cit. on pp. 114, 118).

Jaderberg, Max, Karen Simonyan, Andrew Zisserman, and koray kavukcuoglu (2015). "Spatial Transformer Networks". In: (cit. on p. 86).

Jaunet, Theo, Guillaume Bono, Romain Vuillemot, and Christian Wolf (2021). "Sim2RealViz: Visualizing the Sim2Real Gap in Robot Ego-Pose Estimation". In: *arXiv preprint arXiv:2109.11801*. (cit. on p. 144).

Jha, Debesh, Michael A Riegler, Dag Johansen, Pål Halvorsen, and Håvard D Johansen (2020). "Doubleu-net: A deep convolutional neural network for medical image segmentation". In: *2020 IEEE 33rd International symposium on computer-based medical systems (CBMS)*. IEEE, pp. 558–564 (cit. on p. 31).

Johnson, Matthew, Katja Hofmann, Tim Hutton, and David Bignell (2016a). "The Malmo Platform for Artificial Intelligence Experimentation". In: *IJCAI*. New York, New York, USA: AAAI Press, 4246–4247 (cit. on p. 93).

Johnson, Matthew, Katja Hofmann, Tim Hutton, and David Bignell Microsoft (2016b). "The Malmo Platform for Artificial Intelligence Experimentation". In: *IJCAI* (cit. on pp. 46, 116, 132).

Joshi, Chaitanya K, Thomas Laurent, and Xavier Bresson (2019). "An efficient graph convolutional network technique for the travelling salesman problem". In: *arXiv preprint arXiv:1906.01227* (cit. on p. 100).

Juliani, Arthur, Vincent-Pierre Berges, Ervin Teng, Andrew Cohen, Jonathan Harper, Chris Elion, Chris Goy, Yuan Gao, Hunter Henry, Marwan Mattar, et al. (2018). "Unity: A general platform for intelligent agents". In: *arXiv preprint arXiv:1809.02627* (cit. on pp. 119, 133).

Kadian, Abhishek, Joanne Truong, Aaron Gokaslan, Alexander Clegg, Erik Wijmans, Stefan Lee, Manolis Savva, Sonia Chernova, and Dhruv Batra (2020). "Sim2Real predictivity: Does evaluation in simulation predict real-world performance?" In: *IEEE Robotics and Automation Letters* 5.4, pp. 6670–6677 (cit. on p. 144).

Kaelbling, Leslie Pack, Michael L Littman, and Anthony R Cassandra (1998). "Planning and acting in partially observable stochastic domains". In: *Artificial intelligence* 101.1-2, pp. 99–134 (cit. on pp. 84, 98).

Kapturowski, Steven, Georg Ostrovski, John Quan, Remi Munos, and Will Dabney (2018). "Recurrent experience replay in distributed reinforcement learning". In: *International conference on learning representations* (cit. on pp. 41, 43, 122, 136).

Karkus, Peter, David Hsu, and Wee Sun Lee (2017). *QMDP-Net: Deep Learning for Planning under Partial Observability*. arXiv: 1703.06692 [cs.AI] (cit. on pp. 57, 99).

Karpathy, Andrej (2015). *The Unreasonable Effectiveness of Recurrent Neural Networks*. URL: http://karpathy.github.io/2015/05/21/rnn-effectiveness/ (visited on 11/30/2021) (cit. on p. 23).

Kayalibay, Baris, Atanas Mirchev, Maximilian Soelch, Patrick Van Der Smagt, and Justin Bayer (2018). "Navigation and planning in latent maps". In: *FAIM workshop "Prediction and Generative Modeling in Reinforcement Learning"* (cit. on pp. 67, 131).

Kempka, Michal, Marek Wydmuch, Grzegorz Runc, Jakub Toczek, and Wojciech Jaskowski (2017). "ViZDoom: A Doom-based AI research platform for visual reinforcement learning". In: *IEEE Conference on Computatonal Intelligence and Games, CIG* (cit. on pp. 1, 2, 46, 47, 66, 70, 88, 99).

Kingma, Diederik P. and Jimmy Ba (2017). *Adam: A Method for Stochastic Optimization*. arXiv: 1412.6980 [cs.LG] (cit. on p. 29).

Kipf, Thomas N and Max Welling (2017). "Semi-supervised classification with graph convolutional networks". In: *International Conference on Learning Representations* (cit. on p. 99).

Koenig, Nathan and Andrew Howard (2004a). "Design and use paradigms for gazebo, an open-source multi-robot simulator". In: *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)(IEEE Cat. No. 04CH37566)*. Vol. 3. IEEE, pp. 2149–2154 (cit. on p. 59).

Koenig, Sven and Maxim Likhachev (2005). "Fast replanning for navigation in unknown terrain". In: *IEEE Transactions on Robotics* 21.3, pp. 354–363 (cit. on p. 51).

Koenig, Sven, Maxim Likhachev, and David Furcy (2004b). "Lifelong planning A*". In: *Artificial Intelligence* 155.1-2, pp. 93–146 (cit. on p. 51).

Kolve, Eric, Roozbeh Mottaghi, Daniel Gordon, Yuke Zhu, Abhinav Gupta, and Ali Farhadi (2017a). "AI2-THOR: An Interactive 3D Environment for Visual AI". In: *arxiv pre-print 1712.05474v1* (cit. on pp. 46, 116, 132).

Kolve, Eric, Roozbeh Mottaghi, Daniel Gordon, Yuke Zhu, Abhinav Gupta, and Ali Farhadi (2017b). "AI2-THOR: An Interactive 3D Environment for Visual AI". In: *arXiv* (cit. on pp. 65, 66).

Kostrikov, Ilya (2018). *PyTorch Implementations of Reinforcement Learning Algorithms*. https://github.com/ikostrikov/pytorch-a2c-ppo-acktr (cit. on pp. 72, 89).

Koutnik, Jan, Klaus Greff, Faustino Gomez, and Juergen Schmidhuber (2014). "A clockwork rnn". In: *International Conference on Machine Learning*. PMLR, pp. 1863–1871 (cit. on p. 25).

Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E Hinton (2012). "Imagenet classification with deep convolutional neural networks". In: *Advances in neural information processing systems* 25, pp. 1097–1105 (cit. on pp. 2, 17, 19, 30).

Kumar, Aviral, Aurick Zhou, George Tucker, and Sergey Levine (2020). "Conservative q-learning for offline reinforcement learning". In: *arXiv preprint arXiv:2006.04779* (cit. on p. 136).

Kurniawati, H (2008). "SARSOP: Efficient point-based POMDP planning by approximating optimally reachable belief spaces". In: *Proc. Robotics: Science and Systems, 2008* (cit. on p. 98).

Küttler, Heinrich, Nantas Nardelli, Thibaut Lavril, Marco Selvatici, Viswanath Sivakumar, Tim Rocktäschel, and Edward Grefenstette (2019). "TorchBeast: A PyTorch Platform for Distributed RL". In: *arXiv preprint arXiv:1910.03552*. URL: https://github.com/facebookresearch/torchbeast (cit. on p. 45).

Kwon, Obin, Nuri Kim, Yunho Choi, Hwiyeon Yoo, Jeongho Park, and Songhwai Oh (2021). "Visual Graph Memory with Unsupervised Representation for Visual Navigation". In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 15890–15899 (cit. on pp. 60, 61).

Lample, Guillaume and Devendra Singh Chaplot (2017a). "Playing FPS games with deep reinforcement learning". In: *Thirty-First AAAI Conference on Artificial Intelligence* (cit. on p. 43).

Lample, Guillaume and Devendra Singh Chaplot (2017b). "Playing FPS Games with Deep Reinforcement Learning". In: *AAAI* (cit. on pp. 67, 82, 131, 144).

Lanctot, Marc, Edward Lockhart, Jean-Baptiste Lespiau, Vinicius Zambaldi, Satyaki Upadhyay, Julien Pérolat, Sriram Srinivasan, Finbarr Timbers, Karl Tuyls, Shayegan Omidshafiei, Daniel Hennes, Dustin Morrill, Paul Muller, Timo Ewalds, Ryan Faulkner, János Kramár, Bart De Vylder, Brennan Saeta, James Bradbury, David Ding, Sebastian Borgeaud, Matthew Lai, Julian Schrittwieser, Thomas Anthony, Edward Hughes, Ivo Danihelka, and Jonah Ryan-Davis (2019). "OpenSpiel: A Framework for Reinforcement Learning in Games". In: *CoRR* abs/1908.09453. arXiv: 1908.09453 [cs.LG]. URL: http://arxiv.org/abs/1908.09453 (cit. on pp. 46, 132).

LaValle, Steven M (2006). *Planning algorithms*. Cambridge university press (cit. on p. 98).

Lecun, Yann, Yoshua Bengio, and Geoffrey Hinton (2015). "Deep learning". In: *nature* 521.7553, pp. 436–444 (cit. on p. 129).

LeCun, Yann, Bernhard Boser, John S. Denker, Donnie Henderson, Richard E. Howard, Wayne Hubbard, and Lawrence D. Jackel (1989). "Backpropagation applied to handwritten zip code recognition". In: *Neural computation* 1.4, pp. 541–551 (cit. on p. 18).

Lecun, Yann, L Eon Bottou, Yoshua Bengio, and Patrick Haaner (1998). "Gradient-Based Learning Applied to Document Recognition". In: *Proceedings of the IEEE* 86.11, pp. 2278–2324 (cit. on pp. 30, 32, 66, 82, 105, 131).

Levine, Sergey, Chelsea Finn, Trevor Darrell, and Pieter Abbeel (2016a). "End-to-end training of deep visuomotor policies". In: *The Journal of Machine Learning Research* 17.1, pp. 1334–1373 (cit. on pp. 3, 4).

Levine, Sergey, Chelsea Finn, Trevor Darrell, and Pieter Abbeel (2016b). "End-to-End Training of Deep Visuomotor Policies". In: *Journal of Machine Learning Research* 17, pp. 1–40 (cit. on p. 64).

Li, Zhuwen, Qifeng Chen, and Vladlen Koltun (2018). "Combinatorial optimization with graph convolutional networks and guided tree search". In: *Advances in Neural Information Processing Systems*, pp. 539–548 (cit. on p. 100).

Liang, Eric, Richard Liaw, Robert Nishihara, Philipp Moritz, Roy Fox, Ken Goldberg, Joseph Gonzalez, Michael Jordan, and Ion Stoica (2018). "RLlib: Abstractions for distributed reinforcement learning". In: *International Conference on Machine Learning*. PMLR, pp. 3053–3062 (cit. on pp. 44, 130, 134).

Liang, Xiaodan, Liang Lin, Yunchao Wei, Xiaohui Shen, Jianchao Yang, and Shuicheng Yan (2017). "Proposal-free network for instance-level object segmentation". In: *IEEE transactions on pattern analysis and machine intelligence* 40.12, pp. 2978–2991 (cit. on p. 87).

Lillicrap, Timothy P, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra (2015a). "Continuous Control with Deep Reinforcement Learning". In: *arxiv pre-print 1509.02971* (cit. on pp. 64, 65).

Lillicrap, Timothy P, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra (2015b). "Continuous control with deep reinforcement learning". In: *arXiv preprint arXiv:1509.02971* (cit. on p. 136).

Linietsky, Juan, Ariel Manzur, and Rémi Verschelde (2014). *Godot*. https://github.com/godotengine/godot (cit. on pp. 34, 133).

Lipovetsky, Stan (2015). "Analytical closed-form solution for binary logit regression by categorical predictors". In: *Journal of applied statistics* 42.1, pp. 37–49 (cit. on p. 11).

Littman, Michael L, Anthony R Cassandra, and Leslie Pack Kaelbling (1995). "Learning policies for partially observable environments: Scaling up". In: *Machine Learning Proceedings 1995*. Elsevier, pp. 362–370 (cit. on p. 57).

Liu, Chenxi, Liang-Chieh Chen, Florian Schroff, Hartwig Adam, Wei Hua, Alan Yuille, and Li Fei-Fei (2019). "Auto-DeepLab: Hierarchical Neural Architecture Search for Semantic Image Segmentation". In: *CVPR* (cit. on p. 31).

Liu, Chenxi, Barret Zoph, Maxim Neumann, Jonathon Shlens, Wei Hua, Li-Jia Li, Li Fei-Fei, Alan Yuille, Jonathan Huang, and Kevin Murphy (2018). "Progressive neural architecture search". In: *Proceedings of the European conference on computer vision (ECCV)*, pp. 19–34 (cit. on p. 30).

Liu, Liyuan, Haoming Jiang, Pengcheng He, Weizhu Chen, Xiaodong Liu, Jianfeng Gao, and Jiawei Han (2021). *On the Variance of the Adaptive Learning Rate and Beyond*. arXiv: 1908.03265 [cs.LG] (cit. on p. 29).

Liu, Xiaodong, Kevin Duh, Liyuan Liu, and Jianfeng Gao (2020). "Very deep transformers for neural machine translation". In: *arXiv preprint arXiv:2008.07772* (cit. on p. 31).

Loshchilov, Ilya and Frank Hutter (2019). *Decoupled Weight Decay Regularization*. arXiv: 1711.05101 [cs.LG] (cit. on p. 29).

Marza, Pierre, Laetitia Matignon, Olivier Simonin, and Christian Wolf (2021). *Teaching Agents how to Map: Spatial Reasoning for Multi-Object Navigation*. arXiv: 2107.06011 [cs.CV] (cit. on p. 144).

Mirowski, Piotr, Matt Grimes, Mateusz Malinowski, Karl Moritz Hermann, Keith Anderson, Denis Teplyashin, Karen Simonyan, Andrew Zisserman, Raia Hadsell, et al. (2018a). "Learning to navigate in cities without a map". In: *Advances in Neural Information Processing Systems* 31, pp. 2419–2430 (cit. on pp. 82, 98).

Mirowski, Piotr, Matt Grimes, Mateusz Malinowski, Karl Moritz Hermann, Keith Anderson, Denis Teplyashin, Karen Simonyan, Andrew Zisserman, Raia Hadsell, et al. (2018b). "Learning to navigate in cities without a map". In: *Advances in Neural Information Processing Systems* 31, pp. 2419–2430 (cit. on p. 118).

Mirowski, Piotr, Razvan Pascanu, Fabio Viola, Hubert Soyer, Andrew J. Ballard, Andrea Banino, Misha Denil, Ross Goroshin, Laurent Sifre, Koray Kavukcuoglu, Dharshan Kumaran, and Raia Hadsell (2016). "Learning to navigate in complex environments". In: (cit. on pp. 3, 4, 114, 118).

Mirowski, Piotr, Razvan Pascanu, Fabio Viola, Hubert Soyer, Andrew J. Ballard, Andrea Banino, Misha Denil, Ross Goroshin, Laurent Sifre, Koray Kavukcuoglu, Dharshan Kumaran, and Raia Hadsell (2017). "Learning to Navigate in Complex Environments". In: *ICLR* (cit. on pp. 64, 67, 81, 82, 85, 98, 131).

Mishkin, Dmytro, Alexey Dosovitskiy, and Vladlen Koltun (2019). "Benchmarking classic and learned navigation in complex 3d environments". In: *arXiv preprint arXiv:1901.10915* (cit. on p. 51).

Mnih, Volodymyr, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu (2016a). "Asynchronous methods for deep reinforcement learning". In: *International conference on machine learning*. PMLR, pp. 1928–1937 (cit. on pp. 36, 89, 135).

Mnih, Volodymyr, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu (2016b). "Asynchronous Methods for Deep Reinforcement Learning". In: URL: http://arxiv.org/abs/1602.01783 (cit. on p. 37).

Mnih, Volodymyr, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu (2016c). "Asynchronous Methods for Deep Reinforcement Learning". In: URL: http://arxiv.org/abs/1602.01783 (cit. on p. 72).

Mnih, Volodymyr, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller (2013). "Playing atari with deep reinforcement learning". In: *arXiv preprint arXiv:1312.5602* (cit. on pp. 114, 135).

Mnih, Volodymyr, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen

King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis (2015b). "Human-level control through deep reinforcement learning". In: *Nature* (cit. on pp. 66, 73, 106, 131).

Mnih, Volodymyr, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis (2015a). "Human-level control through deep reinforcement learning". In: *Nature* 518.7540, pp. 529–533 (cit. on pp. 3, 118).

Mnih, Volodymyr, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. (2015c). "Human-level control through deep reinforcement learning". In: *nature* 518.7540, pp. 529–533 (cit. on pp. 32, 39, 40, 82, 98).

Moravec, Hans P. (1989). "Sensor fusion in certainty grids for mobile robots". In: *Sensor devices and systems for robotics*. Springer, pp. 253–276 (cit. on p. 81).

Moré, Jorge J (1978). "The Levenberg-Marquardt algorithm: implementation and theory". In: *Numerical analysis*. Springer, pp. 105–116 (cit. on p. 29).

Moritz, Philipp, Robert Nishihara, Stephanie Wang, Alexey Tumanov, Richard Liaw, Eric Liang, Melih Elibol, Zongheng Yang, William Paul, and Michael I Jordan (2017). "Ray: A Distributed Framework for Emerging AI Applications". In: *USENIX Symposium on Operating Systems Design and Implementation* (cit. on p. 68).

Müller, M., V. Casser, J. Lahoud, N. Smith, and B. Ghanem (Aug. 2017). "Sim4CV: A Photo-Realistic Simulator for Computer Vision Applications". In: *arXiv*. arXiv: 1708.05869 [cs.CV] (cit. on p. 67).

Mur-Artal, Raúl and Juan D Tardós (2014). "Fast relocalisation and loop closing in keyframe-based SLAM". In: *2014 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, pp. 846–853 (cit. on p. 51).

Mur-Artal, Raul and Juan D Tardós (2017). "Orb-slam2: An open-source slam system for monocular, stereo, and rgb-d cameras". In: *IEEE transactions on robotics* 33.5, pp. 1255–1262 (cit. on p. 51).

Nesterov, Yurii (1983). "A method for unconstrained convex minimization problem with the rate of convergence O (1/kˆ 2)". In: *Doklady an ussr*. Vol. 269, pp. 543–547 (cit. on p. 30).

Neverova, Natalia, Christian Wolf, Griffin Lacey, Lex Fridman, Deepak Chandra, Brandon Barbello, and Graham Taylor (2016). "Learning human identity from motion patterns". In: *IEEE Access* 4, pp. 1810–1820 (cit. on p. 25).

Neverova, Natalia, Christian Wolf, Graham Taylor, and Florian Nebout (2015). "Moddrop: adaptive multi-modal gesture recognition". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 38.8, pp. 1692–1706 (cit. on p. 105).

Nguyen, Khanh, Debadeepta Dey, Chris Brockett, and Bill Dolan (2019). "Vision-based navigation with language-based assistance via imitation learning with

indirect intervention". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 12527–12537 (cit. on p. 145).

Novikov, Alexey A., Dimitrios Lenis, David Major, Jiri Hladůvka, Maria Wimmer, and Katja Bühler (2018). *Fully Convolutional Architectures for Multi-Class Segmentation in Chest Radiographs*. arXiv: 1701.08816 [cs.CV] (cit. on p. 31).

Oh, Junhyuk, Valliappa Chockalingam, Satinder Singh, and Honglak Lee (2016). *Control of Memory, Active Perception, and Action in Minecraft*. arXiv: 1605.09128 [cs.AI] (cit. on p. 59).

O'Keefe, John and Jonathan Dostrovsky (1971). "The hippocampus as a spatial map: preliminary evidence from unit activity in the freely-moving rat." In: *Brain research* (cit. on pp. 56, 83).

Olah, Chris (2015). *Understanding LSTM Networks*. URL: https://colah.github.io/posts/2015-08-Understanding-LSTMs/ (visited on 11/30/2021) (cit. on p. 24).

OpenAI (2018). *OpenAI Five*. https://blog.openai.com/openai-five/ (cit. on pp. 3, 67, 132).

Paine, Tom Le, Caglar Gulcehre, Bobak Shahriari, Misha Denil, Matt Hoffman, Hubert Soyer, Richard Tanburn, Steven Kapturowski, Neil Rabinowitz, Duncan Williams, et al. (2019). "Making efficient use of demonstrations to solve hard exploration problems". In: *arXiv preprint arXiv:1909.01387* (cit. on p. 122).

Parisotto, Emilio and Ruslan Salakhutdinov (2017a). "Neural map: Structured memory for deep reinforcement learning". In: *arXiv preprint arXiv:1702.08360* (cit. on pp. 5, 58, 118).

Parisotto, Emilio and Ruslan Salakhutdinov (2017b). "Neural Map: Structured Memory for Deep Reinforcement Learning". In: *arxiv pre-print 1702.08360* (cit. on p. 71).

Parisotto, Emilio and Ruslan Salakhutdinov (2018). "Neural Map: Structured Memory for Deep Reinforcement Learning". In: (cit. on pp. 81, 83, 84, 87, 88, 90, 99).

Parisotto, Emilio, Francis Song, Jack Rae, Razvan Pascanu, Caglar Gulcehre, Siddhant Jayakumar, Max Jaderberg, Raphael Lopez Kaufman, Aidan Clark, Seb Noury, et al. (2020). "Stabilizing transformers for reinforcement learning". In: *International Conference on Machine Learning*. PMLR, pp. 7487–7498 (cit. on p. 144).

Park, Daniel S, Yu Zhang, Ye Jia, Wei Han, Chung-Cheng Chiu, Bo Li, Yonghui Wu, and Quoc V Le (2020). "Improved noisy student training for automatic speech recognition". In: *arXiv preprint arXiv:2005.09629* (cit. on p. 31).

Paszke, Adam, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer (2017). "Automatic differentiation in pytorch". In: (cit. on p. 89).

Paszke, Adam, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga,

Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala (2019). "PyTorch: An Imperative Style, High-Performance Deep Learning Library". In: *Advances in Neural Information Processing Systems 32*. Ed. by H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett. Curran Associates, Inc., pp. 8024–8035. URL: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf (cit. on pp. 2, 44, 107).

Pathak, Deepak, Pulkit Agrawal, Alexei A Efros, and Trevor Darrell (2017). "Curiosity-driven exploration by self-supervised prediction". In: *International conference on machine learning*. PMLR, pp. 2778–2787 (cit. on p. 136).

Perlin, Ken (1985). "An image synthesizer". In: *ACM Siggraph Computer Graphics* 19.3, pp. 287–296 (cit. on p. 119).

Petrenko, Aleksei, Zhehui Huang, Tushar Kumar, Gaurav Sukhatme, and Vladlen Koltun (2020). "Sample Factory: Egocentric 3D Control from Pixels at 100000 FPS with Asynchronous Reinforcement Learning". In: *ICML* (cit. on pp. 45, 134).

Pham, Hieu, Zihang Dai, Qizhe Xie, and Quoc V Le (2021). "Meta pseudo labels". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 11557–11568 (cit. on p. 30).

Pire, Taihú, Thomas Fischer, Gastón Castro, Pablo De Cristóforis, Javier Civera, and Julio Jacobo Berlles (2017). "S-ptam: Stereo parallel tracking and mapping". In: *Robotics and Autonomous Systems* 93, pp. 27–42 (cit. on p. 51).

Puterman, Martin L (1990). "Markov decision processes". In: *Handbooks in operations research and management science* 2, pp. 331–434 (cit. on p. 13).

Qian, Ning (1999). "On the momentum term in gradient descent learning algorithms". In: *Neural networks* 12.1, pp. 145–151 (cit. on p. 30).

Rae, Jack W, Jonathan J Hunt, Tim Harley, Ivo Danihelka, Andrew Senior, Greg Wayne, Alex Graves, and Timothy P Lillicrap (2016). "Scaling memory-augmented neural networks with sparse reads and writes". In: *arXiv preprint arXiv:1610.09027* (cit. on p. 25).

Raffel, Colin, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu (2019). "Exploring the limits of transfer learning with a unified text-to-text transformer". In: *arXiv preprint arXiv:1910.10683* (cit. on p. 31).

Raposo, David, Adam Santoro, David Barrett, Razvan Pascanu, Timothy Lillicrap, and Peter Battaglia (2017). "Discovering objects and their relations from entangled scene representations". In: *arXiv preprint arXiv:1702.05068* (cit. on p. 26).

Ravanelli, Mirco, Philemon Brakel, Maurizio Omologo, and Yoshua Bengio (2018). "Light gated recurrent units for speech recognition". In: *IEEE Transactions on Emerging Topics in Computational Intelligence* 2.2, pp. 92–102 (cit. on p. 31).

Reddi, Sashank J., Satyen Kale, and Sanjiv Kumar (2018). "On the Convergence of Adam and Beyond". In: *International Conference on Learning Representations*. URL: https://openreview.net/forum?id=ryQu7f-RZ (cit. on p. 30).

Redmon, Joseph and Ali Farhadi (2018). *YOLOv3: An Incremental Improvement*. arXiv: 1804.02767 [cs.CV] (cit. on p. 31).

Remolina, Emilio and Benjamin Kuipers (2004). "Towards a general theory of topological maps". In: *Artif. Intell.* 152, pp. 47–104 (cit. on p. 98).

Ren, Shaoqing, Kaiming He, Ross Girshick, and Jian Sun (2015). "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks". In: ed. by C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett (cit. on p. 82).

Ribeiro, Maria Isabel (2004). "Kalman and extended kalman filters: Concept, derivation and properties". In: *Institute for Systems and Robotics* 43, p. 46 (cit. on p. 51).

Rosenblatt, Frank (1958). "The perceptron: a probabilistic model for information storage and organization in the brain." In: *Psychological review* 65.6, p. 386 (cit. on p. 16).

Roumimper, Nick (2017). "Mesh Navigation Through Jumping". MA thesis. Utrecht University (cit. on p. 115).

Ruder, Sebastian (2017). *An overview of gradient descent optimization algorithms*. arXiv: 1609.04747 [cs.LG] (cit. on p. 30).

Rummelhard, Lukas, Amaury Negre, and Christian Laugier (2015). "Conditional monte carlo dense occupancy tracker". In: *2015 IEEE 18th International Conference on Intelligent Transportation Systems*. IEEE, pp. 2485–2490 (cit. on p. 81).

Sadek, Assem, Guillaume Bono, Boris Chidlovskii, and Christian Wolf (2021). *An in-depth experimental study of sensor usage and visual reasoning of robots navigating in real environments*. arXiv: 2111.14666 [cs.AI] (cit. on pp. 48, 144).

Salimans, Tim, Jonathan Ho, Xi Chen, Szymon Sidor, and Ilya Sutskever (2017). "Evolution strategies as a scalable alternative to reinforcement learning". In: *arXiv preprint arXiv:1703.03864* (cit. on p. 136).

Savinov, Nikolay, Alexey Dosovitskiy, and Vladlen Koltun (2018a). "Semi-parametric topological memory for navigation". In: *arXiv preprint arXiv:1803.00653* (cit. on pp. 5, 118).

Savinov, Nikolay, Alexey Dosovitskiy, and Vladlen Koltun (2018b). "Semi-parametric topological memory for navigation". In: *International Conference on Learning Representations*. URL: https://openreview.net/forum?id=SygwwGbRW (cit. on pp. 61, 96, 99).

Savva, Manolis, Angel X Chang, Alexey Dosovitskiy, Thomas Funkhouser, and Vladlen Koltun (2017). "MINOS: Multimodal Indoor Simulator for Navigation in Complex Environments". In: *arxiv pre-print 1712.03931* (cit. on pp. 65, 66).

Savva, Manolis, Abhishek Kadian, Oleksandr Maksymets, Yili Zhao, Erik Wijmans, Bhavana Jain, Julian Straub, Jia Liu, Vladlen Koltun, Jitendra Malik, Devi Parikh,

and Dhruv Batra (2019a). "Habitat: A Platform for Embodied AI Research". In: (cit. on pp. 1, 2, 46–48, 100, 107, 116, 132).

Savva, Manolis, Abhishek Kadian, Oleksandr Maksymets, Yili Zhao, Erik Wijmans, Bhavana Jain, Julian Straub, Jia Liu, Vladlen Koltun, Jitendra Malik, et al. (2019b). "Habitat: A platform for embodied ai research". In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 9339–9347 (cit. on pp. 61, 66, 88, 93).

Schaarschmidt, Michael, Alexander Kuhnle, and Kai Fricke (2017). *TensorForce: A TensorFlow library for applied reinforcement learning* (cit. on p. 68).

Schaul, Tom, John Quan, Ioannis Antonoglou, and David Silver (2015). "Prioritized experience replay". In: *arXiv preprint arXiv:1511.05952* (cit. on p. 40).

Schlichtkrull, Michael, Thomas N Kipf, Peter Bloem, Rianne Van Den Berg, Ivan Titov, and Max Welling (2018). "Modeling relational data with graph convolutional networks". In: *European Semantic Web Conference*. Springer, pp. 593–607 (cit. on p. 99).

Schulman, John, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz (2015). "Trust region policy optimization". In: *International conference on machine learning*. PMLR, pp. 1889–1897 (cit. on pp. 38, 50).

Schulman, John, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov (2017a). "Proximal policy optimization algorithms". In: *arXiv preprint arXiv:1707.06347* (cit. on pp. 36–38, 106, 135).

Schulman, John, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov (2017b). "Proximal Policy Optimization Algorithms". In: *arxiv pre-print 1707.06347* (cit. on pp. 45, 59, 65, 106).

Sethian, James A (1996). "A fast marching level set method for monotonically advancing fronts". In: *Proceedings of the National Academy of Sciences* 93.4, pp. 1591–1595 (cit. on p. 53).

Sethian, James A (1999). "Fast marching methods". In: *SIAM review* 41.2, pp. 199–235 (cit. on p. 53).

Shani, Guy, Joelle Pineau, and Robert Kaplow (2013). "A survey of point-based POMDP solvers". In: *Autonomous Agents and Multi-Agent Systems* 27.1, pp. 1–51 (cit. on p. 98).

Shatkay, Hagit and Leslie Pack Kaelbling (1997). "Learning topological maps with weak local odometric information". In: *IJCAI (2)*, pp. 920–929 (cit. on p. 98).

Shen, Jonathan, Ruoming Pang, Ron J Weiss, Mike Schuster, Navdeep Jaitly, Zongheng Yang, Zhifeng Chen, Yu Zhang, Yuxuan Wang, Rj Skerrv-Ryan, et al. (2018). "Natural tts synthesis by conditioning wavenet on mel spectrogram predictions". In: *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, pp. 4779–4783 (cit. on p. 31).

Silver, David, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalch-

brenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis (Jan. 2016a). "Mastering the Game of Go with Deep Neural Networks and Tree Search". In: *Nature* 529.7587, pp. 484–489 (cit. on pp. 66, 131).

Silver, David, Aja Huang, Christopher J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis (2016b). "Mastering the game of Go with deep neural networks and tree search". In: *nature* 529.7587, 484–489 (cit. on pp. 114, 118).

Silver, David, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. (2018). "A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play". In: *Science* 362.6419, pp. 1140–1144 (cit. on pp. 82, 98).

Silver, David, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller (2014). "Deterministic policy gradient algorithms". In: *International conference on machine learning*. PMLR, pp. 387–395 (cit. on p. 45).

Silver, David, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. (2017). "Mastering the game of go without human knowledge". In: *nature* 550.7676, pp. 354–359 (cit. on p. 114).

Simonyan, Karen and Andrew Zisserman (2014). "Very deep convolutional networks for large-scale image recognition". In: *arXiv preprint arXiv:1409.1556* (cit. on p. 19).

Smallwood, Richard D and Edward J Sondik (1973). "The optimal control of partially observable Markov processes over a finite horizon". In: *Operations research* 21.5, pp. 1071–1088 (cit. on p. 98).

Smith, Trey and Reid Simmons (2004). "Heuristic search value iteration for POMDPs". In: *Proceedings of the 20th conference on Uncertainty in artificial intelligence*, pp. 520–527 (cit. on p. 98).

Snook, Greg (2000). "Simplified 3D movement and pathfinding using navigation meshes". In: *Game Programming Gems*. Charles River Media, pp. 288–304 (cit. on pp. 114, 117).

Sreenu, G and MA Saleem Durai (2019). "Intelligent video surveillance: a review through deep learning techniques for crowd analysis". In: *Journal of Big Data* 6.1, pp. 1–27 (cit. on p. 31).

Srinivas, Aravind, Allan Jabri, Pieter Abbeel, Sergey Levine, and Chelsea Finn (2018). *Universal Planning Networks*. arXiv: 1804.00645 [cs.LG] (cit. on pp. 57, 99).

Srivastava, Rupesh Kumar, Klaus Greff, and Jürgen Schmidhuber (2015). "Highway networks". In: *arXiv preprint arXiv:1505.00387* (cit. on p. 19).

Stentz, Anthony (1997). "Optimal and efficient path planning for partially known environments". In: *Intelligent unmanned ground vehicles*. Springer, pp. 203–220 (cit. on p. 51).

Stentz, Anthony et al. (1995). "The focussed d^* algorithm for real-time replanning". In: *IJCAI*. Vol. 95, pp. 1652–1659 (cit. on p. 51).

Stooke, Adam and Pieter Abbeel (2018). "Accelerated Methods for Deep Reinforcement Learning". In: *arxiv pre-print 1803.02811* (cit. on pp. 67, 132).

Sun, Chi, Xipeng Qiu, Yige Xu, and Xuanjing Huang (2019). "How to fine-tune bert for text classification?" In: *China National Conference on Chinese Computational Linguistics*. Springer, pp. 194–206 (cit. on p. 31).

Sutton, Richard S. (1988). "Learning to predict by the methods of temporal differences". In: *Machine learning* 3.1, pp. 9–44 (cit. on p. 41).

Szegedy, Christian, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna (2016). "Rethinking the inception architecture for computer vision". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2818–2826 (cit. on p. 30).

Szot, Andrew, Alex Clegg, Eric Undersander, Erik Wijmans, Yili Zhao, John Turner, Noah Maestre, Mustafa Mukadam, Devendra Chaplot, Oleksandr Maksymets, Aaron Gokaslan, Vladimir Vondrus, Sameer Dharur, Franziska Meier, Wojciech Galuba, Angel Chang, Zsolt Kira, Vladlen Koltun, Jitendra Malik, Manolis Savva, and Dhruv Batra (2021). "Habitat 2.0: Training Home Assistants to Rearrange their Habitat". In: *arXiv preprint arXiv:2106.14405* (cit. on pp. 46, 47, 61, 132).

Tamar, Aviv, Yi Wu, Garrett Thomas, Sergey Levine, and Pieter Abbeel (2016). *Value Iteration Networks*. arXiv: 1602.02867 [cs.AI] (cit. on pp. 56, 99).

Tan, Mingxing and Quoc V. Le (2020). *EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks*. arXiv: 1905.11946 [cs.LG] (cit. on p. 19).

Tateno, Keisuke, Federico Tombari, Iro Laina, and Nassir Navab (2017). "Cnn-slam: Real-time dense monocular slam with learned depth prediction". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 6243–6252 (cit. on pp. 51, 80, 84).

Team, Open Ended Learning, Adam Stooke, Anuj Mahajan, Catarina Barros, Charlie Deck, Jakob Bauer, Jakub Sygnowski, Maja Trebacz, Max Jaderberg, Michael Mathieu, et al. (2021). "Open-ended learning leads to generally capable agents". In: *arXiv preprint arXiv:2107.12808* (cit. on p. 145).

Tesauro, Gerald (1994). "TD-Gammon, a self-teaching backgammon program, achieves master-level play". In: *Neural computation* 6.2, pp. 215–219 (cit. on p. 114).

Tesauro, Gerald et al. (1995). "Temporal difference learning and TD-Gammon". In: *Communications of the ACM* 38.3, pp. 58–68 (cit. on p. 3).

Thongtan, Tan and Tanasanee Phienthrakul (2019). "Sentiment classification using document embeddings trained with cosine similarity". In: *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics: Student Research Workshop*, pp. 407–414 (cit. on p. 31).

Thrun, Sebastian (1998). "Learning metric-topological maps for indoor mobile robot navigation". In: *Artificial Intelligence* 99.1, pp. 21–71 (cit. on p. 98).

Todorov, Emanuel, Tom Erez, and Yuval Tassa (2012). "Mujoco: A physics engine for model-based control". In: *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, pp. 5026–5033 (cit. on pp. 46, 132).

Van Der Maaten, Laurens and Geoffrey Hinton (2008). "Visualizing Data using t-SNE". In: *Journal of Machine Learning Research* 9, pp. 2579–2605 (cit. on p. 77).

Van Hasselt, Hado, Arthur Guez, and David Silver (2016). "Deep reinforcement learning with double q-learning". In: *Proceedings of the AAAI conference on artificial intelligence*. Vol. 30. 1 (cit. on p. 40).

Vaswani, Ashish, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin (2017a). "Attention is all you need". In: *Advances in neural information processing systems*, pp. 5998–6008 (cit. on pp. 3, 82).

Vaswani, Ashish, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin (2017b). "Attention is all you need". In: *Advances in neural information processing systems*, pp. 5998–6008 (cit. on pp. 24, 25).

Vaswani, Ashish, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin (2017c). *The Annotated Transformer*. URL: https://nlp.seas.harvard.edu/2018/04/03/attention.html (visited on 11/30/2021) (cit. on p. 24).

Vinyals, Oriol, Igor Babuschkin, Wojciech M Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David H Choi, Richard Powell, Timo Ewalds, Petko Georgiev, et al. (2019). "Grandmaster level in StarCraft II using multi-agent reinforcement learning". In: *Nature* 575.7782, pp. 350–354 (cit. on pp. 3, 132).

Wang, Jane X., Zeb Kurth-Nelson, Dhruva Tirumala, Hubert Soyer, Joel Z. Leibo, Rémi Munos, Charles Blundell, Dharshan Kumaran, and Matthew Botvinick (2016a). "Learning to reinforcement learn". In: *arxiv pre-print 1611.05763* (cit. on pp. 67, 82, 131).

Wang, Qing, Jiechao Xiong, Lei Han, peng sun, Han Liu, and Tong Zhang (2018). "Exponentially Weighted Imitation Learning for Batched Historical Data". In: *Advances in Neural Information Processing Systems*. Ed. by S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett. Vol. 31. Curran Associates, Inc. URL: https://proceedings.neurips.cc/paper/2018/file/4aec1b3435c52abbdf8334ea0e7141e0-Paper.pdf (cit. on p. 136).

Wang, Ranxiao Frances and Elizabeth S. Spelke (2002). "Human Spatial Representation: Insights From Animals". In: *Trends in Cognitive Sciences* 6.9, pp. 376–382 (cit. on p. 95).

Wang, Ziyu, Tom Schaul, Matteo Hessel, Hado Hasselt, Marc Lanctot, and Nando Freitas (2016b). "Dueling network architectures for deep reinforcement learning". In: *International conference on machine learning*. PMLR, pp. 1995–2003 (cit. on p. 40).

Wani, Saim, Shivansh Patel, Unnat Jain, Angel Chang, and Manolis Savva (2020). "MultiON: Benchmarking Semantic Map Memory using Multi-Object Navigation". In: *Advances in Neural Information Processing Systems*. Ed. by H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin. Vol. 33. Curran Associates, Inc., pp. 9700–9712. URL: https://proceedings.neurips.cc/paper/2020/file/6e01383fd96a17ae51cc3e15447e7533-Paper.pdf (cit. on pp. 59, 118, 144).

Watkins, Christopher JCH and Peter Dayan (1992). "Q-learning". In: *Machine learning* 8.3-4, pp. 279–292 (cit. on pp. 14, 15).

Wayne, Greg, Chia-Chun Hung, David Amos, Mehdi Mirza, Arun Ahuja, Agnieszka Grabska-Barwinska, Jack Rae, Piotr Mirowski, Joel Z Leibo, Adam Santoro, et al. (2018a). "Unsupervised predictive memory in a goal-directed agent". In: *arXiv preprint arXiv:1803.10760* (cit. on pp. 83, 84, 87, 99).

Wayne, Greg, Chia-Chun Hung, David Amos, Mehdi Mirza, Arun Ahuja, Agnieszka Grabska-Barwinska, Jack Rae, Piotr Mirowski, Joel Z Leibo, Adam Santoro, et al. (2018b). "Unsupervised predictive memory in a goal-directed agent". In: *arXiv preprint arXiv:1803.10760* (cit. on pp. 118, 132).

Weng, Jiayi, Huayu Chen, Dong Yan, Kaichao You, Alexis Duburcq, Minghao Zhang, Hang Su, and Jun Zhu (2021). "Tianshou: A Highly Modularized Deep Reinforcement Learning Library". In: *arXiv preprint arXiv:2107.14171* (cit. on p. 45).

Wijmans, Erik, Abhishek Kadian, Ari Morcos, Stefan Lee, Irfan Essa, Devi Parikh, Manolis Savva, and Dhruv Batra (2019a). "DD-PPO: Learning near-perfect PointGoal navigators from 2.5 billion frames". In: *arXiv preprint arXiv:1911.00357* (cit. on pp. 3, 45).

Wijmans, Erik, Abhishek Kadian, Ari Morcos, Stefan Lee, Irfan Essa, Devi Parikh, Manolis Savva, and Dhruv Batra (2019b). "DD-PPO: Learning near-perfect pointgoal navigators from 2.5 billion frames". In: *arXiv preprint arXiv:1911.00357* (cit. on pp. 132, 135).

Williams, Richard J. (1992a). "Simple Statistical Gradient Following Algorithms For Connectionist Reinforcement Learning". In: *Machine Learning*, pp. 229–256 (cit. on p. 72).

Williams, Ronald J (1992b). "Simple statistical gradient-following algorithms for connectionist reinforcement learning". In: *Machine learning* 8.3, pp. 229–256 (cit. on pp. 36, 37).

Wögerer, Christian, Harald Bauer, Martijn Rooker, Gerhard Ebenhofer, Alberto Rovetta, Neil Robertson, and Andreas Pichler (2012). "LOCOBOT-low cost toolkit for building robot co-workers in assembly lines". In: *International conference on intelligent robotics and applications*. Springer, pp. 449–459 (cit. on p. 59).

Wolf, Thomas, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, et al. (2019a). "Huggingface's transformers: State-of-the-art natural language processing". In: *arXiv preprint arXiv:1910.03771* (cit. on pp. 24, 144).

Wolf, Thomas, Victor Sanh, Julien Chaumond, and Clement Delangue (2019b). "Transfertransfo: A transfer learning approach for neural network based conversational agents". In: *arXiv preprint arXiv:1901.08149* (cit. on p. 31).

Wu, Yuhuai, Elman Mansimov, Shun Liao, Roger Grosse, and Jimmy Ba (2017). "Scalable trust-region method for deep reinforcement learning using Kronecker-factored approximation". In: *NIPS* (cit. on p. 65).

Wu, Zhuofeng, Sinong Wang, Jiatao Gu, Madian Khabsa, Fei Sun, and Hao Ma (2020). "Clear: Contrastive learning for sentence representation". In: *arXiv preprint arXiv:2012.15466* (cit. on p. 31).

Wu, Zonghan, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S Yu (2019). "A comprehensive survey on graph neural networks". In: *arXiv preprint arXiv:1901.00596* (cit. on p. 99).

Wydmuch, Marek, Michał Kempka, and Wojciech Jaśkowski (2018). "ViZDoom Competitions: Playing Doom from Pixels". In: *arXiv preprint arXiv:1809.03470* (cit. on pp. 46, 116, 118, 132).

Xia, Fei, Amir R. Zamir, Zhi-Yang He, Alexander Sax, Jitendra Malik, and Silvio Savarese (2018). "Gibson env: real-world perception for embodied agents". In: *CVPR*. IEEE (cit. on pp. 46, 47, 66, 88, 107, 116, 132, 139).

Xie, Saining, Ross Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He (2017). *Aggregated Residual Transformations for Deep Neural Networks*. arXiv: 1611.05431 [cs.CV] (cit. on p. 19).

Xu, K., J. Li, M. Zhang, S.S. Du, K. Kawarabayashi, and S. Jegelka (2019a). "What Can Neural Networks Reason About?" In: *arxiv preprint 1905.13211* (cit. on p. 102).

Xu, Keyulu, Jingling Li, Mozhi Zhang, Simon S Du, Ken-ichi Kawarabayashi, and Stefanie Jegelka (2019b). "What Can Neural Networks Reason About?" In: *arXiv preprint arXiv:1905.13211* (cit. on p. 100).

Yang, Zhilin, Zihang Dai, Yiming Yang, Jaime Carbonell, Russ R Salakhutdinov, and Quoc V Le (2019). "Xlnet: Generalized autoregressive pretraining for language understanding". In: *Advances in neural information processing systems* 32 (cit. on p. 31).

Zaheer, Manzil, Guru Guruganesh, Kumar Avinava Dubey, Joshua Ainslie, Chris Alberti, Santiago Ontanon, Philip Pham, Anirudh Ravula, Qifan Wang, Li Yang,

et al. (2020). "Big Bird: Transformers for Longer Sequences." In: *NeurIPS* (cit. on p. 31).

Zeiler, Matthew D. (2012). *ADADELTA: An Adaptive Learning Rate Method*. arXiv: 1212.5701 [cs.LG] (cit. on p. 29).

Zeiler, Matthew D and Rob Fergus (2014). "Visualizing and understanding convolutional networks". In: *European conference on computer vision*. Springer, pp. 818–833 (cit. on pp. 18, 21).

Zhang, Chiyuan, Oriol Vinyals, Rémi Munos, and Samy Bengio (2018). "A Study on Overfitting in Deep Reinforcement Learning". In: *arxiv* (cit. on p. 68).

Zhang, Jingwei, Lei Tai, Joschka Boedecker, Wolfram Burgard, and Ming Liu (2017a). "Neural SLAM: Learning to Explore with External Memory". In: *arxiv pre-print 1706.09520* (cit. on p. 58).

Zhang, Jingwei, Lei Tai, Joschka Boedecker, Wolfram Burgard, and Ming Liu (2017b). "Neural Slam: Learning to explore with external memory". In: *arXiv preprint arXiv:1706.09520* (cit. on p. 118).

Zhang, Jingwei, Lei Tai, Ming Liu, Joschka Boedecker, and Wolfram Burgard (2017c). "Neural slam: Learning to explore with external memory". In: *arXiv preprint arXiv:1706.09520* (cit. on pp. 82, 84, 99).

Zhou, Chenhong, Changxing Ding, Xinchao Wang, Zhentai Lu, and Dacheng Tao (2020). "One-pass multi-task networks with cross-task guided attention for brain tumor segmentation". In: *IEEE Transactions on Image Processing* 29, pp. 4516–4529 (cit. on p. 31).

Zhou, Jie, Ganqu Cui, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, Lifeng Wang, Changcheng Li, and Maosong Sun (2018). "Graph neural networks: A review of methods and applications". In: *arXiv preprint arXiv:1812.08434* (cit. on p. 99).

Zhou, Yuyin, Zhe Li, Song Bai, Chong Wang, Xinlei Chen, Mei Han, Elliot Fishman, and Alan L Yuille (2019). "Prior-aware neural network for partially-supervised multi-organ segmentation". In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 10672–10681 (cit. on p. 31).

Zhu, Yuke, Roozbeh Mottaghi, Eric Kolve, Joseph J Lim, Abhinav Gupta, Li Fei-Fei, and Ali Farhadi (2017). "Target-driven visual navigation in indoor scenes using deep reinforcement learning". In: *2017 IEEE international conference on robotics and automation (ICRA)* (cit. on pp. 3, 4, 48, 97, 100).