Visual Navigation using Deep Reinforcement Learning

Thesis · May 2019

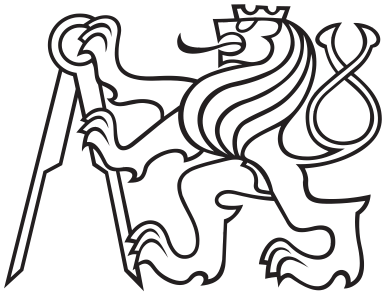
CITATIONS
0

READS
541

1 author:

Jonáš Kulhánek
Czech Technical University in Prague
16 PUBLICATIONS **246** CITATIONS

Czech Technical University in Prague
Faculty of Electrical Engineering
Department of Cybernetics



Visual Navigation using Deep Reinforcement Learning

Bachelor Thesis

Jonáš Kulhánek

Bachelor programme: Open Informatics
Branch of study: Computer and Informatic Science
Supervisor: Ing. Erik Derner

Prague, May 2019

Thesis Supervisor:
Ing. Erik Derner
Department of Control Engineering
Faculty of Electrical Engineering
Czech Technical University in Prague
Prague, Czech Republic

BACHELOR'S THESIS ASSIGNMENT

I. Personal and study details

Student's name: **Kulhánek Jonáš** Personal ID number: **466241**

Faculty / Institute: **Faculty of Electrical Engineering**

Department / Institute: **Department of Cybernetics**

Study program: **Open Informatics**

Branch of study: **Computer and Information Science**

II. Bachelor's thesis details

Bachelor's thesis title in English:

Visual Navigation using Deep Reinforcement Learning

Bachelor's thesis title in Czech:

Vizuální navigace s použitím hlubokého posilovaného učení

Guidelines:

Develop, implement and test a method for a visual navigation task in a 3D environment. The agent percieves the environment through 2D images obtained by a standard RGB camera. The target is also described by an image.
Tasks:
- A deep neural network will be used to navigate the agent in its environment. Consider the method presented in [1] as a possible starting point.
- Use reinforcement learning [2], for instance the Asynchronous Advantage Actor-Critic (A3C) algorithm, to train the agent.
- Consider the use of a recurrent neural network in order to cope with partial observability of the environment.
- Evaluate the method in a suitably chosen scenario.

Bibliography / sources:

[1] Zhu Yuke, Mottaghi Roozbeh, Kolve Eric, Lim Joseph J., Gupta Abhinav, Fei-Fei Li, Farhadi Ali. Target-driven Visual Navigation in Indoor Scenes using Deep Reinforcement Learning. In 2017 IEEE International Conference on Robotics and Automation (ICRA), pp. 3357–3364, Singapore, 2017.
[2] Sutton Richard S., Barto Andrew G. Reinforcement Learning: An Introduction. MIT Press, Cambridge, MA, USA, 2nd edition, 2017.
[3] Goodfellow Ian, Bengio Yoshua, Courville Aaron. Deep Learning. MIT Press, Cambridge, MA, USA, 2016.

Name and workplace of bachelor's thesis supervisor:

Ing. Erik Derner, Department of Control Engineering, FEE

Name and workplace of second bachelor's thesis supervisor or consultant:

Ing. Petr Pošík, Ph.D., Analysis and Interpretation of Biomedical Data, FEE

Date of bachelor's thesis assignment: **07.01.2019** Deadline for bachelor thesis submission: **24.05.2019**

Assignment valid until: **30.09.2020**

Ing. Erik Derner Supervisor's signature	doc. Ing. Tomáš Svoboda, Ph.D. Head of department's signature	prof. Ing. Pavel Ripka, CSc. Dean's signature

III. Assignment receipt

The student acknowledges that the bachelor's thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the bachelor's thesis, the author must state the names of consultants and include a list of references.

.	
Date of assignment receipt	Student's signature

Author statement for undergraduate thesis:
I declare that the presented work was developed independently and that I have listed all sources of information used within it in accordance with the methodical instructions for observing the ethical principles in the preparation of university theses.

Prague, May 6, 2019 .....
signature

Abstract

Deep reinforcement learning (RL) has been successfully applied to a variety of game-like environments. However, the application of deep RL to visual navigation with realistic 3D environments is a challenging task. We propose a novel learning architecture capable of navigating an agent to a target given by an image. To achieve this, we have extended batched advantage actor-critic (A2C) algorithm with auxiliary tasks designed to improve visual navigation performance. We propose three additional auxiliary tasks for the prediction of the depth map, of the observation image segmentation and of the target image segmentation. These tasks enable the use of supervised learning to pre-train a major part of the network and to substantially reduce the number of training steps. The training performance can be further improved by increasing the environment complexity gradually over time. An efficient neural network structure is proposed, which is capable of learning for multiple targets in multiple environments. Our method navigates in continuous state spaces and on the AI2-THOR environment simulator surpasses the performance of state-of-the-art goal-oriented visual navigation methods from the literature.

Keywords: Robot navigation, deep reinforcement learning, actor-critic, auxiliary tasks.

Abstrakt

Hluboké posilované učení bylo aplikované na řadu herních prostředí. Aplikace hlubokého posilovaného učení na vizuální navigaci v realistických prostředích je však náročný úkol. Navrhujeme novou učící architekturu schopnou navigovat agenta k cíli danému obrázkem. K tomu, abychom toho dosáhli, jsme rozšířili batched A2C algoritmus o pomocné moduly, které byly navrženy, aby vylepšily výkon algoritmu aplikovaného na vizuální navigaci. Navrhujeme tři rozšiřující pomocné moduly pro predikci hloubkové mapy a segmentačních masek pozorovaného obrázku a cílového obrázku. Tyto moduly umožňují použít učení s učitelem na předtrénování velké části neuronové sítě, což snižuje počet trénovacích kroků potřebných k naučení algoritmu. Výkon učení může být navíc zlepšen, když se postupně zvyšuje složitost prostředí s časem. Navrhujeme efektivní strukturu neuronové sítě, která je schopná naučit se navigovat do různých cílů v různých prostředích. Naše metoda je schopná navigace ve spojitých prostředích a v prostředí AI2-THOR překonává výkon state-of-the-art metod umožňujících navigaci do zadaného cíle.

Klíčová slova: Robotická navigace, hluboké posilované učení, actor-critic, pomocné moduly.

Preface

For a long time, I was genuinely interested in deep reinforcement learning for its generality and strength. The application to visual navigation was, however, challenging and required a lot of effort to make it work. In the end, I was quite happy with the results. I hope this thesis would help others who are also interested in a similar topic.

This thesis is ultimately based on the results of the research I have conducted on my internship at TU Delft as a member of a larger team of scientists. I would like to namely thank prof. Robert Babuška as well as Erik Derner who have been very helpful and provided me with excellent guidance and support with conducting my research. Thanks also belong to Tim de Bruin with whom I have consulted many choices when designing the method.

This thesis also serves as a base for a paper which has been submitted to the ECMR 2019 conference and is attached in the thesis as Appendix A. The source code of the proposed method is published on a GitHub repository: `https://github.com/jkulhanek/a2cat-vn-pytorch`. It depends on another repository that I have developed to make the implementation of new RL algorithms easier. This repository can be found at the following link: `https://github.com/jkulhanek/deep-rl-pytorch`.

I hope you enjoy reading.
Jonáš Kulhánek
Prague, May 6, 2019

Contents

List of Figures

List of Tables

List of Acronyms

A2C advantage actor-critic vii, 17, 18

A3C asynchronous advantage actor-critic 8, 18

AI artificial intelligence 1

DNN deep neural network 7, 16, 41

DP dynamic programming 7, 13, 14

DQN deep Q networks 16

DRL deep reinforcement learning 7, 8, 16

LSTM long short-term memory 7, 11, 22, 24

MCTS Monte Carlo tree search 7

MDP Markov decision process 6, 12, 13

MSE mean squared error 10, 16, 17, 26

POMDP partially observable Markov decision process 6, 13

RL reinforcement learning 6, 7, 12, 16

RNN recurrent neural network 11

SLAM simultaneous localization and mapping problem 7

SVM support vector machine 7

TD temporal difference 14, 15

Chapter 1

Introduction

Ultimately, the goal of artificial intelligence (AI) is to be able to replace humans on every task. There is still a long way to go from now. Although we have seen AI achieving human-level performance or even outperforming humans on several tasks (Silver et al., 2017; Mnih et al., 2013, 2015), there are still many problems that resist any attempt to solve them. Navigation is the essential skill that needs to be mastered before a generation of robots can help humans with their everyday chores. It is the very prerequisite that would enable a wide range of applications in robotics. However, as of now, none of the known methods have been able to approach human-level perception and navigation. The problem is that navigation using visual input is a complex task requiring vision, recognition, localization, memorization, planning and other skills. Although we have developed methods good enough for reproducing most of the needed skills, joining them together shows itself to be increasingly difficult. Eventually, the missing piece of the puzzle is generalization. Current methods can operate quite reliably if the map of the environment is given beforehand. If there is no prior knowledge, we still struggle to solve the navigation task. The goal of this research is to change that and move us a bit towards achieving the human-level performance in this domain.

Reinforcement learning is a general tool to solve any problem involving agents interacting with an environment. Any problem could be formalized in this way. This is not a coincidence. In fact, any problem that we want to solve involves us as agents interacting with the universe as the environment. This might be the reason why human brain uses similar algorithms as we use to solve reinforcement learning problems (Law and Gold, 2009; Sigala and Logothetis, 2002). If we ever want to achieve true artificial general intel-

ligence, our algorithms would need to solve any problem that is posed to them just like humans. Reinforcement learning is, therefore, our best tool to achieve that.

With the recent rise of computational power our computation capabilities increased. It allowed us to take the idea of multi-layer perceptron further. We created deep neural networks and applied them to several large-scale domains. The idea of a deep neural network is also not too distant from the way the human brain works. The visual cortex in the human brain could be viewed as one large deep neural network. Deep neural networks were applied to several domains including object recognition (He et al., 2016; Krizhevsky et al., 2012), object detection and even machine translation or voice recognition (van den Oord et al., 2016). When applied to vision tasks the neural network was able to develop similar convolutional filters as engineers used to handcraft before the advent of deep learning.

A combination of these two, called deep reinforcement learning, has been successfully applied to a variety of problems including playing the game of Go (Silver et al., 2017), chess, playing Atari games (Mnih et al., 2013) or 3D games (Jaderberg et al., 2016; Espeholt et al., 2018). It even went as far as writing programs to draw images (Ganin et al., 2018). Thus, it seems an intriguing idea to apply deep reinforcement learning to visual navigation tasks. Visual navigation involves vision and therefore could benefit from the application of deep learning. On the other hand, the navigation problem is best formalized as a reinforcement learning problem. We also have an agent or a robot moving in an environment. This agent is supposed to navigate somewhere and receives the reward upon reaching the destination. Another motivation for using deep reinforcement learning might be perhaps the fact that humans can do that and they are the best at it.

However, the application of deep reinforcement learning to visual navigation is very challenging. The real-world environment is far more complex than a game-like environment. The complexity of the visual input calls for a deep neural network. These networks should be comparable in size to those used in He et al. (2016); Krizhevsky et al. (2012). Training these networks seems to be an idea difficult enough on its own. For training the agent, we need a good 3D environment simulator. This simulator should resemble the real-world environment as close as possible and be fast enough to be of any use for the training. As for now, there is no such simulator. Another problem arising when applying deep reinforcement learning to the visual navigation problems is the sparsity of the

reward. The agent does not know if he goes in the right way for a long time. It knows it went in the right direction only after he reaches the destination. Some clever strategies must be used to help the agent find the correct way.

We wanted our agent to be as general as possible. Therefore the agent uses only one monocular camera, and an image specifies the target (an image it will see from the target position). The goal of the agent is to move from its current location to the target by applying a sequence of actions, based on the camera observations only. We focus on the case when the environment is initially unknown, i.e., no explicit map is available. The learning algorithm is based on the batched version of advantage actor-critic algorithm (Wu et al., 2017), extended with auxiliary tasks to help the agent learn useful features in the absence of informative rewards. During the training of the deep network, we use depth maps and image segmentations as inputs to auxiliary tasks. In addition, we propose a method to pre-train the neural network before the reinforcement learning algorithm is applied. The algorithm could be sped up by transfer learning from one environment to another and by gradually increasing the environment complexity. Finally, to address the partial observability problem, we propose a novel neural network architecture that is both efficient and compact. We evaluate our method in realistic indoor environments similar to Zhu et al. (2017) and Mirowski et al. (2016).

Chapter 2

Related Work

2.1 Classical Methods

There has been a lot of work done in visual navigation. Several methods use an explicit map of the environment. However, the map does not need to be complete or precise. At least a generic map, which does not contain any details about the environment, must be provided. One of these methods (Kim and Nevatia, 1999) uses edge detection to reconstruct the detailed map. The planning algorithm is able to respond to symbolic commands, e.g. "at the first conter, turn left", expressed in a formal language. Similarly, in Oriolo et al. (1995), only a generic map is provided. An agent then observes the environment and uses fuzzy logic to incorporate the local map into the global map of the environment. The A* algorithm is used for path planning.

Other methods try to avoid the problem of having a map of the environment a priori by reconstructing the map on the fly. The navigation then occurs in the reconstructed map. Wooden (2006) reconstruct the map from visual sensory input by detecting known features. The A* algorithm is then run on this reconstructed map to find a path to the destination. Tomono (2006) use images only, to reconstruct a map of an indoor environment. Some methods, e.g. the work by Kidono et al. (2002), rely on human guidance in the map reconstruction phase. Saeedi et al. (2006) extract scene features from visual input and their 3D positions are calculated. Then their trajectories are calculated as the agent moves.

We will now take a look at the methods that neither need a map of the environment beforehand nor do they reconstruct it on the fly, e.g. the work by Lenser and Veloso

(2003), and by Remazeilles et al. (2004). Remazeilles et al. (2004) use images of the environment in the training phase. The agent has a task to navigate to a given image. It relies on feature extraction and matching to match similar images pairwise and then find the shortest path in the resulting graph. Potential field methods are used to navigate between two consecutive images on the path. Lenser and Veloso (2003) use the visual input for obstacle avoidance. Bonin-Font et al. (2008) give a survey of methods used in visual navigation.

2.2 Reinforcement Learning

The problem of navigating an agent in an unknown environment is well suited for reinforcement learning (RL) methods. Therefore, recent successes of RL and their application to the problem of navigation and motor control will be summarized in this section.

Many methods use RL-based techniques for collision avoidance and motor control. An obstacle avoidance method (Michels et al., 2005) tries to drive a remote controlled car in unstructured outdoor environments using monocular camera input only. It uses a combination of supervised learning algorithm with an RL agent. This RL agent has the distance to the closest obstacle in different parts of the input image as its input and uses the PEGASUS algorithm (Ng and Jordan, 2000) to learn the correct policy. The authors showed that a system trained on synthetic data was able to perform well in real-world environments. Another method (Kim et al., 2004) uses a similar combination of supervised learning and an RL agent to control the flight of a helicopter autonomously. The authors used the Kalman filter for predicting the position, speed, and rotation of the helicopter from sensory inputs. The PEGASUS algorithm was also used (Ng and Jordan, 2000) on the data collected from a real flight of an expert to learn the agent's policy. Both of these approaches share the same PEGASUS method (Ng and Jordan, 2000) for finding the optimal policy of the RL agent. This method transforms any Markov decision process (MDP) or partially observable Markov decision process (POMDP) to a deterministic POMDP (such a POMDP where the state resulting from taking a given action in a certain state is not random).

The paper by Peters and Schaal (2008) addresses the problem of learning complex motor skills with human-like limbs. The authors compare several policy-gradient methods

for finding the optimal control policy. Another reinforcement learning method (Kohl and Stone, 2004) uses gradient methods to find a policy to control a four-legged robot. It optimizes the agent for fastest forward motion.

A map reconstruction method (Kollar and Roy, 2008) focuses on simultaneous localization and mapping problem (SLAM). It automates the data collection for map reconstruction. It tries to optimize the agent's trajectory to collect optimal data for map construction. It uses a policy search using dynamic programming (DP) (Bagnell et al., 2004) to find the optimal policy. The support vector machine (SVM) is used as a one-step policy to map high-dimensional inputs to actions.

2.3 Deep Reinforcement Learning

Recently, DNNs were also used as function approximators in RL. The method combining RL with deep learning is called deep reinforcement learning (DRL). A method called deep Q-learning (Mnih et al., 2013) successfully applied DRL to several Atari games, achieving or surpassing human level on several of them. A deep convolutional neural network was used as the function approximator for the state-value function, feeding the raw image input without using any hand-crafted features. An *experience replay buffer* was used to stabilize the convergence of the learning algorithm by decoupling strongly correlated consecutive frames of the game. The Q-learning algorithm was used for off-policy training of the function approximator. Benefits of DRL were demonstrated in the AlphaGo method (Silver et al., 2016) by surpassing the human-level performance on the game of Go. The authors used a convolutional DNN in combination with Monte Carlo tree search (MCTS). Expert knowledge was used to initialize the algorithm. This method was then improved by AlphaGo Zero (Silver et al., 2017), which used only the experience collected by playing against itself. This method achieved the best performance on the game of Go, defeating AlphaGo.

Another method (Sallab et al., 2017) focused on autonomous driving. It used deep attention reinforcement learning to extract the features needed for the control task. Long short-term memory (LSTM) (Hochreiter and Schmidhuber, 1997) was employed to deal with partial observability of the environment. The network was trained end-to-end using the deep Q-learning algorithm.

2.4 Deep Reinforcement Learning for Visual Navigation

The application of DRL to visual navigation suffers from high dimensionality of the observation space. Observation space dimensionality can be reduced by using hand-crafted features, or by using learned features trained on either the training dataset or on a completely different one, e.g. ResNet (He et al., 2016) features, automatically extracted from the image (Zhu et al., 2017; Bruce et al., 2017). A different, proposed in Wu et al. (2018), uses image segmentations and depth maps as inputs to the agent. It was trained and evaluated on houses from SUNCG dataset (Song et al., 2017) and the trained agent was able to find multiple targets specified as a separate input to the agent.

Raw high-dimensional input images can also be used directly for navigation (Jaderberg et al., 2016; Mirowski et al., 2016). These two methods extend the asynchronous advantage actor-critic (A3C) algorithm (Mnih et al., 2016) with auxiliary tasks to stabilize the training and make it more efficient when the reward is sparse. They, however, use the DeepMind Lab (Beattie et al., 2016) game simulator which is much simpler than realistic simulators (Kolve et al., 2017; Wu et al., 2018; Song et al., 2017). The only method that relies on visual input only in a realistic indoor environment was presented by Zhu et al. (2017). However, it was applied to AI2-THOR 3D environment simulator (Kolve et al., 2017) which contains small single-room environments, and the action space discretized the environments into a simple grid worlds.

In our approach, the agent learns to navigate based on the observed raw images only, as opposed to Zhu et al. (2017), which uses ResNet features. It was applied to realistic indoor environments similar to Zhu et al. (2017) and Mirowski et al. (2016). The input to our agent is only the image input from the environment (we also utilize depth maps and image segmentations, but only for training).

Chapter 3

Theoretical Background

This chapter begins with a concise introduction to deep learning. We discuss multilayer-perceptrons and extend the idea to recurrent neural networks. Next, the reinforcement learning is described. We will give our formalization of the problem. Value-based methods, as well as policy-gradient methods, are discussed. The chapter is concluded with a short overview of selected actor-critic methods.

3.1 Deep Learning

Deep learning represents a method for function approximation. First, we will introduce the feed-forward neural networks, which can be viewed as directed acyclic graphs. The idea will be taken further to the case of recurrent neural networks.

3.1.1 Multilayer Perceptron

The deep feed-forward network (Goodfellow et al., 2016), also called multilayer perceptron, is a machine learning model used for function approximation. The original function $f(x)$ is approximated by a deep feed-forward network represented by a function $\hat{f}(x, \theta)$, parametrized by $\theta \in \mathbb{R}^n$. This reduces the problem of searching for a function in the space of all functions to the search in $\mathbb{R}^n$. The space of all functions obtained by changing parameters θ is called the *hypothesis space*. If the function f lies in the hypothesis space, it can be approximated with zero error. The best-approximating function is chosen such

that it minimizes the *empirical risk* given by

$$R(\theta|\mathcal{D}) \triangleq \frac{1}{|\mathcal{D}|} \sum_{(x,y)\in\mathcal{D}} J(y - \hat{f}(x,\theta)) \,, \tag{3.1}$$

where J is the loss function, e.g. the mean squared error (MSE), and $\mathcal{D}$ is the training set.

The function $\hat{f}(x,\theta)$ is constructed by alternating affine functions in the form $Ax + b$ with non-linear functions. Some examples of non-linear functions are tanh, sigmoid, ReLU. The resulting differentiable function is then optimized using stochastic gradient methods (Robbins and Monro, 1951). The recent rise of computational power allowed for big functions with lots of parameters to be optimized efficiently.

We often view the network as being composed of several layers. A layer is composed of an affine function followed by a non-linear function. This non-linear function is called the activation function. The layers are stacked on top of each other – the output of each layer is the input to the next layer. Let $l_1, l_2, \ldots, l_m$ be the network layers. Then the neural network is given by

$$\hat{f}(x,\theta) \equiv l_n \circ \ldots \circ l_2 \circ l_1 \,, \tag{3.2}$$

where $l_i(X) = \text{nonlin}_i(\theta_{i,weight}X + \theta_{i,bias})$ for a non-linear function nonlin_i. Softmax non-linear function is defined as

$$\sigma(x)_i = \frac{\mathrm{e}^{x_i}}{\sum_{j=1}^{n} \mathrm{e}^{x_j}} \,, \tag{3.3}$$

and the ReLU function is defined as

$$\text{ReLU}(x)_i = \max\{0, x_i\} \,, \tag{3.4}$$

where both the ReLU and the softmax functions maps from $\mathbb{R}^n$ to $\mathbb{R}^n$, indexed by subscript.

A layer is not always connected to the next layer in sequence. Instead, we view the network as a directed acyclic graph where the layers represent the nodes and the connections between the layers represent the edges. We can allow for a single node to have more than one incoming as well as outgoing edges. In this case, the input is simply the concatenation of the input vectors. This graph-view of the deep neural networks will be especially useful later when describing the neural network layout.

3.1.2 Recurrent Neural Networks

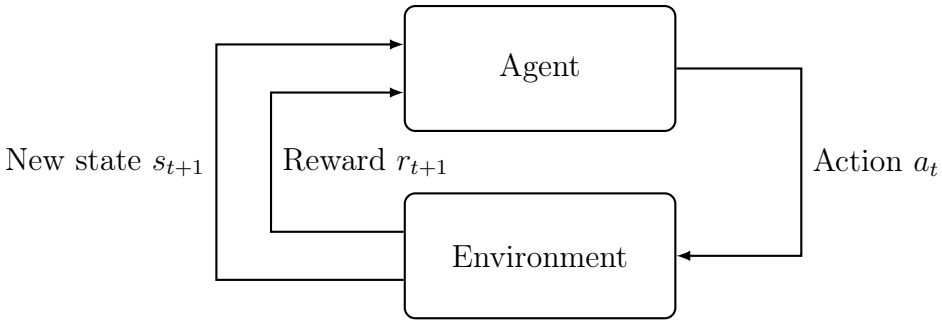
Feed-forward neural networks are great tools when dealing with all sorts of problems, especially vision. However, it would be useful for the neural network to have a memory. In feed-forward neural networks, knowledge is embedded in network parameters. Each network evaluation is independent on another. For some tasks, however, it would be useful to let the network remember the previous evaluations. A motivation could be translation problems where the next word depends on the previous words. We would like to enable the network to build a compact representation of all previous experience in the form of a state. To achieve this, we relax the acyclic property of our neural network graph. By allowing the network to use the outputs of previous evaluations, the gradients propagate back in time and the network is allowed to use the past experience. The set of outputs of the previous evaluation that are passed to the next one is called a state. Note that the state is not a set of all neural network outputs. Usually, it is a secondary output of a hidden layer. Neural networks which reuse the outputs from previous evaluations are generally called recurrent neural networks (RNNs) (Sherstinsky, 2018).

When the gradient is propagated backward in time, there is a problem with gradient decay. Clever network architectures must be used to solve this problem. One example of such an architecture is LSTM (Hochreiter and Schmidhuber, 1997).

3.2 Reinforcement Learning

In this section, we provide a brief introduction to the concepts used in our method. The notation used in this section follows Sutton and Barto (2018), where random values are denoted by capital letters whereas lower letters are used for their observed values[1]. For brevity, we will mostly describe the discrete space case. The equations could, however, be extended to the continuous case simply by replacing sums with integrals and by replacing probability mass functions with probability density functions.

Figure 3.1: Reinforcement learning (RL) process (Sutton and Barto, 2018).



3.2.1 Formalization

A broad range of problems can be described as RL problems. These problems include a decision maker called *agent* which tries to achieve a given goal. The agent interacts with the *environment* throughout a specified set of *actions* and receives a real-valued quantity called *reward*. The reward describes how well was the agent able to achieve the predefined goal.

More specifically, the agent and the environment interact at discrete time steps $t \in \mathbb{N}$. At each time step t, the agent observes the state $S_t \in \mathcal{S}$ of the environment and picks one of the possible actions $A_t \in \mathcal{A}$. The environment then changes its state to state $S_{t+1} \in \mathcal{S}$ and the agent receives reward $R_{t+1} \in \mathbb{R}$. The process is visualized in Figure 3.1. The agent starts in a state S_0 sampled from a distribution over a set of initial states. Then it proceeds until it reaches a *terminal state*. This process is called an *episode*[2]. Each episode is independent of another. An example of an episode is a single play of a game. The experience the agent collects in a single episode is called the *trajectory*. The trajectory is defined as

$$\zeta = S_0, A_0, R_1, S_1, A_1, R_2, \dots . \tag{3.5}$$

We call the problem an MDP when the probability distribution over the next state S_{t+1}, obtained by following action $a_t \in \mathcal{A}$ in the state $s_t \in \mathcal{S}$, depends only on the state s_t and the action a_t, i.e. the following property holds in the discrete state-space case:

$$\begin{aligned} &P(S_{t+1} = s_{t+1} | S_t = s_t, A_t = a_t) = \\ &P(S_{t+1} = s_{t+1} | S_t = s_t, A_t = a_t, \dots, S_1 = s_1, A_1 = a_1) \,. \end{aligned} \tag{3.6}$$

[1]For the sake of simplicity, we write $S \in \mathcal{S}$ to denote that the observed value of the random variable S belongs to the set $\mathcal{S}$, as used in Sutton and Barto (2018).

[2]Sometimes called a trial

If the agent does not have access to the true state, but receives only an observation of the true state which depends on but does not fully determine the true state, we call the problem a POMDP. For the moment, we will only consider the MDP case. Later, we will extend the algorithms used to the POMDP case.

An agent can be represented by a probability distribution π over the set of possible actions, conditioned on the state[3]. This distribution is called a *policy*. Let the *discounted cumulative reward* be defined as

$$G_t \triangleq \sum_{i=0}^{\infty} \gamma^i R_{t+i+1}\,, \tag{3.7}$$

where γ is greater than 0 and less than or equal to 1.

The agent tries to maximize the expected discounted cumulative reward: $\mathbb{E}\big[G_0\big]$.

3.2.2 Dynamic programming

If the probability distribution over the next states $P(S_{t+1} = s_{t+1}|S_t = s_t, A_t = a_t)$ is available, we can employ DP methods. The solution of finding an optimal policy $\pi_\star$, such that it maximizes $\mathbb{E}\big[G_0\big]$, can be found by solving the Bellman optimality equations (Bellman, 1957). Let the optimal state-action value function $q_\star : (\mathcal{S}, \mathcal{A}) \to \mathbb{R}$ be defined as

$$q_\star(s, a) = \mathbb{E}\big[R_{t+1} + \gamma \max_{a'} q_\star(S_{t+1}, a')|S_t = s, A_t = a\big]\,. \tag{3.8}$$

This equation gives us the expected discounted cumulative reward we would get by taking action a in state s and following the optimal policy $\pi_\star$ from there. Optimal policy for the discrete action space case can be represented as probability mass function $\pi_\star$ which can be obtained from $q_\star$, as shown in (3.9). For more detail, please refer to Sutton and Barto

[3]Sometimes, we will use the same symbol for either the probability density function of the policy if the action space is continuous or for the probability mass function of the policy if the action space is discrete.

(2018).

$$\pi_\star(a|s) = \begin{cases} 0 & \text{if } a \notin \arg\max_{a'} q_\star(s,a')\,, \\ \delta_a & \text{otherwise}\,, \end{cases} \tag{3.9}$$

$$\text{where} \sum_{a \in \arg\max_{a'} q_\star(s,a')} \delta_a = 1\,.$$

Most of the times, however, the model is not available to us or the state space is too large for constructing the state-value function[4].

3.2.3 Model-Free Methods

Model-free methods do not require the model, but only the experience collected from the interaction between the agent and the environment. We can split them into policy gradient methods (Sutton et al., 2000) and value-based methods. Value-based methods use an approximation of the value-function or the state-value function. These algorithms include SARSA (Sutton and Barto, 2018) or Q-learning.

Value-Based Methods

Q-learning (Watkins and Dayan, 1992) is an *off-policy* algorithm – the policy used to select the action is different than the policy that is being optimized. To ensure exploration the algorithm uses the ϵ-greedy policy, which selects a random action with probability ϵ and follows the policy π otherwise. In each step, the algorithm performs the temporal difference (TD) (Sutton and Barto, 2018) learning step to update its state-value function. It means that the state-value function is not updated with full DP update described in (3.8) but only with its sample. We cannot compute the full update because we do not have access to the model – probability distribution over next states (3.6). Note that in this case, the state-value function is random for two reasons. It depends on the initialization (bootstrapping is used) and the updates to the state-value function are just Monte Carlo approximations of the real updates we would get if we could evaluate the full state-value updates according to (3.8). It makes the algorithm a combination of a DP method where bootstrapping is used and the Monte Carlo method where only samples from the DP

[4]For example in the continuous space case

updates are used to update the state-value function. For more details please refer to the description of TD methods in Sutton's Introduction to RL (Sutton and Barto, 2018), the Q-learning algorithm convergence proof (Watkins and Dayan, 1992) and a lecture on temporal-difference methods (Gašić, 2017).

Policy-Gradient Methods

Instead of constructing either the state-value function or the value-function, the *policy gradient* methods model and optimize the policy directly. Let the reward function for the policy π be defined as

$$J_\pi \triangleq \mathbb{E}_{s\sim d_\pi(s), a\sim\pi_\theta(a|s)}\left[q_\pi(a,s)\right], \tag{3.10}$$

where $q_\pi(a,s)$ is the state-value function for π and d_π is the stationary distribution of Markov chain for π. In the discrete case, it is defined as

$$d_\pi(s) = \lim_{t\to\infty} P\{S_t = s\}. \tag{3.11}$$

The initial state $S_0 \sim \mathbb{U}_{\text{init}}$ is sampled from the uniform probability distribution over a set of initial states.[5]

Let the policy π_θ be parametrized by a set of parameters θ. By using the *policy gradient theorem* (Sutton and Barto, 2018), the gradient is obtained as

$$\frac{\partial J_{\pi_\theta}}{\partial\theta} = \mathbb{E}_{s\sim d_{\pi_\theta}(s), a\sim\pi_\theta(a|s)}\left[q_{\pi_\theta}(a,s)\frac{\partial}{\partial\theta}\log\pi_\theta(a|s)\right]. \tag{3.12}$$

3.2.4 Deep Reinforcement Learning

For complex reinforcement learning problems, neither the policy π nor the value-function can be represented explicitly. When the dimension of the state space is too big or the state space is continuous, we have to use approximators for either the policy or the value-function. The use of approximators can also be motivated for cases where there is a concept of similarity between states – for example in case of images. A metric measuring the similarity between states can be defined in the state space. It is beneficial if the approximator translates the metric from the state space to actions or values. This concept of metric translation vaguely means that when two states are similar, so should be the

[5] Other distributions over the set of initial states than uniform could be used.

output of the approximator used.

Deep neural networks are a great choice for such an approximator. They not only learn the function they are approximating, but they also learn the correct way to translate the similarity metric from the state space. An example of this is when the input to the network is an image of an animal. We would like to teach the algorithm to act in a specific way if the animal in the picture is a cat. If we did not use any approximators, the original space would have a high number of dimensions. The algorithm would be incapable of generalization. We would need to use all possible images of all possible animals to train it correctly. When a neural network with a suitable architecture is used as a function approximator, it can learn the common features describing the concept of a cat. A desired property of the neural network is to yield a similar output for similar animals. It can, therefore, generalize well to previously unseen images. Use of DNNs as function approximators in RL is generally called deep reinforcement learning (DRL).

Deep Q Networks

In Q-learning, we aim to find the function $q : (\mathcal{S} \times \mathcal{A}) \to \mathbb{R}$ mapping from the state-action space to real numbers. We can instead reformulate the problem as finding a vector θ such that $\hat{q}(s, a|\theta) \approx q(a, s)$, where $s \in S$, $a \in \mathcal{A}$ and $\hat{q}$ is a parametrized approximator of q. This extension allows to apply the algorithm to continuous or large state spaces where it would not be possible to enumerate all the states. We can apply the Bellman optimality equation (3.8) for $\hat{q}$ to obtain

$$\hat{q}(s, a|\theta) = \mathbb{E}\big[R_{t+1} + \gamma \max_{a'} \hat{q}(S_{t+1}, a'|\theta)|S_t = s, A_t = a\big]. \tag{3.13}$$

We want to minimize the MSE criterion. The loss function is given by

$$J(\theta) = \mathbb{E}\big[\big(R_{t+1} + \gamma \max_{a'} \hat{q}(S_{t+1}, a'|\theta) - \hat{q}(S_t, A_t|\theta)\big)^2\big]. \tag{3.14}$$

Fortunately, this equation is differentiable with respect to θ, and thus we can employ gradient-based methods to find the optimal value for θ. This algorithm is called deep Q networks (DQN) (Mnih et al., 2013).

3.3 Actor-Critic Methods

Actor-critic algorithms are policy gradient methods. They are suitable for continuous state spaces (Grondman et al., 2012). They use two functions: the *actor* and the *critic*. The actor is the function approximator of the policy and the critic is the function approximator of either the value-function or the state-value function.

3.3.1 Advantage Actor-Critic Algorithms (A2C)

To reduce the variance, the advantage actor-critic (A2C) uses the *advantage* function instead of the state-value function. Let the state-value function be approximated by $v_{\theta_v} \colon \mathcal{S} \to \mathbb{R}$, parameterized by θ_v. We use a stochastic policy $\pi_{\theta_p}(a|s)$ which is a probability distribution over the set of possible actions, conditioned on the state $s \in \mathcal{S}$, and parameterized by θ_p. Let the bootstrapped n-step return G_t^n be defined as

$$G_t^n = R_{t+1} + \gamma R_{t+2} + \ldots + \gamma^{n-1} R_{t+n} + \gamma^n \mathbb{1}_{\mathrm{c}} v_{\theta_v}(S_{t+n}) \,, \tag{3.15}$$

where $\mathbb{1}_{\mathrm{c}}$ is zero if the episode ended during the rollout and one otherwise and $n \geq 1$. The actor is updated similarly to REINFORCE algorithm (Williams, 1992) with advantage estimates from the critic. The gradient of the actor's loss function J_p from Sutton et al. (2000) is given by

$$\frac{\partial J_p}{\partial \theta_p} = -\sum_{i=t}^{t+\ell} \frac{\partial}{\partial \theta_p} \log \pi_{\theta_p}(a_i|s_i)(g_i^{t+\ell-i+1} - v_{\theta_v}(s_i)) \,. \tag{3.16}$$

The term $g_i^{t+\ell-i+1} - v_{\theta_v}(s_i)$ is referred to as the advantage function. The critic is updated using n-step temporal difference learning: the MSE between the bootstrapped n-step return and the critic output is computed and a gradient descent update is applied. The gradient of the critic's loss function J_v is

$$\frac{\partial J_v}{\partial \theta_v} = \sum_{i=t}^{t+\ell} \frac{\partial}{\partial \theta_v} \frac{1}{2} \big(g_i^{t+\ell-i+1} - v_{\theta_v}(s_i)\big)^2 \,. \tag{3.17}$$

To ensure exploration, the negative entropy of the actor is added to the total loss. The negative entropy of the actor in state s is defined as

$$H^{-}(s, \theta_p) = \sum_{a \in \mathcal{A}} \pi_{\theta_p}(a|s) \log \pi_{\theta_p}(a|s) \tag{3.18}$$

and its gradient on the rollout data is

$$\frac{\partial J_e}{\partial \theta_p} = \sum_{i=t}^{t+\ell} \sum_{a \in \mathcal{A}} \frac{\partial}{\partial \theta_p} \pi_{\theta_p}(a|s_i) \log \pi_{\theta_p}(a|s_i)\,. \tag{3.19}$$

Note that the above setting differs from the one given in Sutton and Barto (2018), which uses n-step forward view to compute the return G. When DNNs are used as the function approximators for the actor and critic, it is beneficial to optimize on multiple time-steps in a single batch. We therefore use only the rollout data to optimize all time-steps in the rollout at once. The estimated returns are a mixture of returns with different length for each state, which was proven to have the error reduction property in the discrete RL setting (Watkins and Dayan, 1992; Gurvits et al., 1994).

As the critic and actor can share knowledge about the environment, they can share some of their parameters, which leads to an improved learning performance. For example, when using neural networks for visual tasks, the bottom-most convolutional layers used in both the actor and the critic need to learn the same convolutional filters. It is therefore beneficial to share their parameters as there are fewer to train, the hypothesis space has fewer dimensions and the search for an optimal solution is simpler.

3.3.2 Asynchronous Advantage Actor-Critic Algorithm (A3C)

A3C – first proposed by Mnih et al. (2016) – differs from original A2C in the sense that there are up to k agents, each doing the same rollout-optimization steps as described in Section 3.3.1. Each agent has its copy of the environment and its copy of parameters θ. At the beginning of each rollout, each agent updates its parameters with the parameters from a centralized parameter store and the optimization after each rollout is done with respect to the centralized parameters. The process is visualized in Figure 3.2. Each agent starts from a different initial state and sometimes with a slightly modified environment. Due to this diversity having multiple agents spawned in this way have stabilizing effects

on the learning. As was argued in Mnih et al. (2016), this serves the role of experience replay buffer used in Mnih et al. (2013).

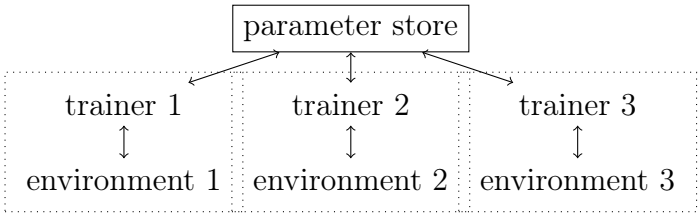


Figure 3.2: **A3C training configuration.** Each trainer has its own copy of parameters. At the beginning of the optimization step, it pulls the newest parameters from the centralized parameter store. The gradients are then applied directly to shared parameters.

However, having multiple agents causes problems when running on GPUs. We can no longer utilize batch updates, which makes the algorithm less efficient. As multiple agents update the same parameters at the same time, the conflicts need to be resolved. Two techniques are generally applied. Either one of the conflicting updates is discarded, which decreases the performance (on terms of the number of frames processed), or a locking mechanism must be introduced, also decreasing the number of processed frames per second.

3.3.3 Batched Advantage Actor-Critic (A2C)

Batched advantage actor-critic (Wu et al., 2017) tries to improve the training performance of A3C algorithm. There are k different environment simulator instances. At each time step, k actions are sampled by the actor, one for each simulator. The rollouts collected from the environments are used to optimize the actor and the critic in a single batch. This training process is visualized in Figure 3.3. Having multiple environment simulators has the same stabilizing effect on the training process as A3C.

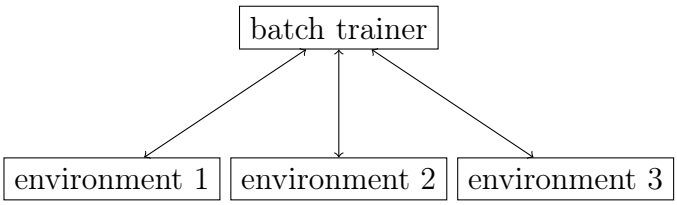


Figure 3.3: **A2C training process.** There are multiple environment simulator instances, but a single trainer with only one set of parameters. The steps in all environments are performed in parallel and the collected data are used to compute the gradients in a batch.

3.3.4 Off-policy Critic Updates

Collecting observations can be costly, especially when the environment simulator has to simulate physics and render 3D scenes. For an algorithm to be efficient, it needs to learn as much as possible from the experiences collected so far. To improve the data efficiency and the stability of the algorithm, a memory of past experiences called the *experience replay buffer* is used. It keeps the last n_e experiences, i.e., observations, actions, rewards, and terminals[6]. At each learning step, a sequence of experiences is sampled from the buffer and it is used to compute the bootstrapped n-step returns (3.15) to optimize the critic.

3.3.5 UNREAL Auxiliary Tasks

Deep RL algorithms are commonly enhanced with auxiliary tasks in order to improve their learning performance. For instance, Jaderberg et al. (2016) extended the A3C algorithm with two auxiliary tasks, *reward prediction* and *pixel control*. The former predicts the sign of the reward based on past four observations and the latter uses an additional pseudo-reward function to learn a policy that maximizes the absolute pixel change. The batched A2C or A3C can be enhanced in the same way; more details are given in Section 4.3.

[6] A terminal is the indicator of the episode ending in a particular time step.

Chapter 4

Proposed Learning Architecture

Our method extends batched A2C algorithm with UNREAL auxiliary tasks and additional auxiliary tasks for visual navigation. We call the method A2CAT-VN, which is an abbreviation of A2C with Auxiliary Tasks for Visual Navigation. We have made its implementation[1] as well as a framework implementing several deep RL algorithms[2] publicly available on GitHub.

First, we will describe the neural network used in our method. Different ways to solve the partial observability problem are discussed. Then, we will describe UNREAL auxiliary tasks (Jaderberg et al., 2016) and our additional auxiliary tasks for visual navigation. The chapter ends with our approach to simplify the training in large environments.

4.1 Neural Network

The deep neural network used in our method consists of several modules: convolutional base, LSTM, actor, critic, and auxiliary tasks, see Figure 4.1. In the sequel, we explain the individual blocks one by one.

The convolutional base is depicted in Figure 4.2. Its inputs are the observed image and the target image, each entering into a separate stream of two convolutional layers with shared weight parameters. The outputs of the second layer are concatenated and passed to additional two convolutional layers, followed by a single fully-connected linear layer.

Each of these layers is followed by the ReLU activation function. Unlike He et al.

[1] `https://github.com/jkulhanek/a2cat-vn-pytorch`
[2] `https://github.com/jkulhanek/deep-rl-pytorch`

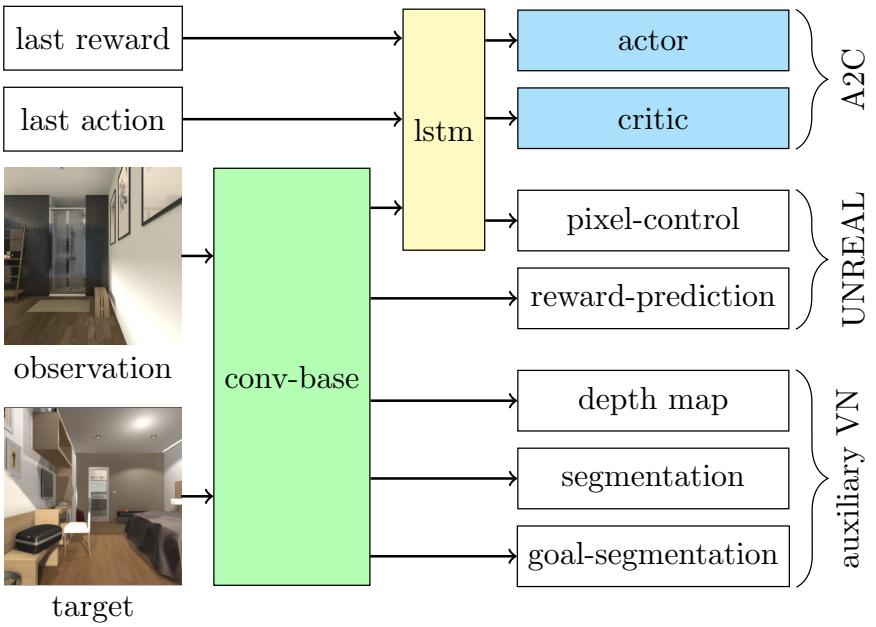


Figure 4.1: A2CAT-VN neural network architecture.

(2016), we do not employ maxpool layers. Instead, the images are down-sampled by using stride only, as suggested in Springenberg et al. (2014). The convolutional base features are merged with the previous action and the previous reward and passed to the LSTM layer (Hochreiter and Schmidhuber, 1997).

The previous action is encoded using one-hot encoding and the reward is clipped to the interval $[-1, 1]$. LSTM features are used as the input for both the actor and the critic, as well as for the pixel control auxiliary task. Let $\phi(x)$ be the LSTM features of an input x (LSTM features are computed from the convolutional features and therefore are a function of the input). Note that the input is composed of the image observation, the target image, the last action and the last reward, as well as the previous LSTM state. The critic is an affine transformation of the LSTM features and the actor is the result of the softmax function applied to an affine transformation of the LSTM features.

4.2 Resolving Partial Observability

The partial observability of the environment does not allow the agent to uniquely distinguish which state it occupies based on a sole observation. Using previous observations can, however, greatly improve its ability to navigate in the environment. For example,

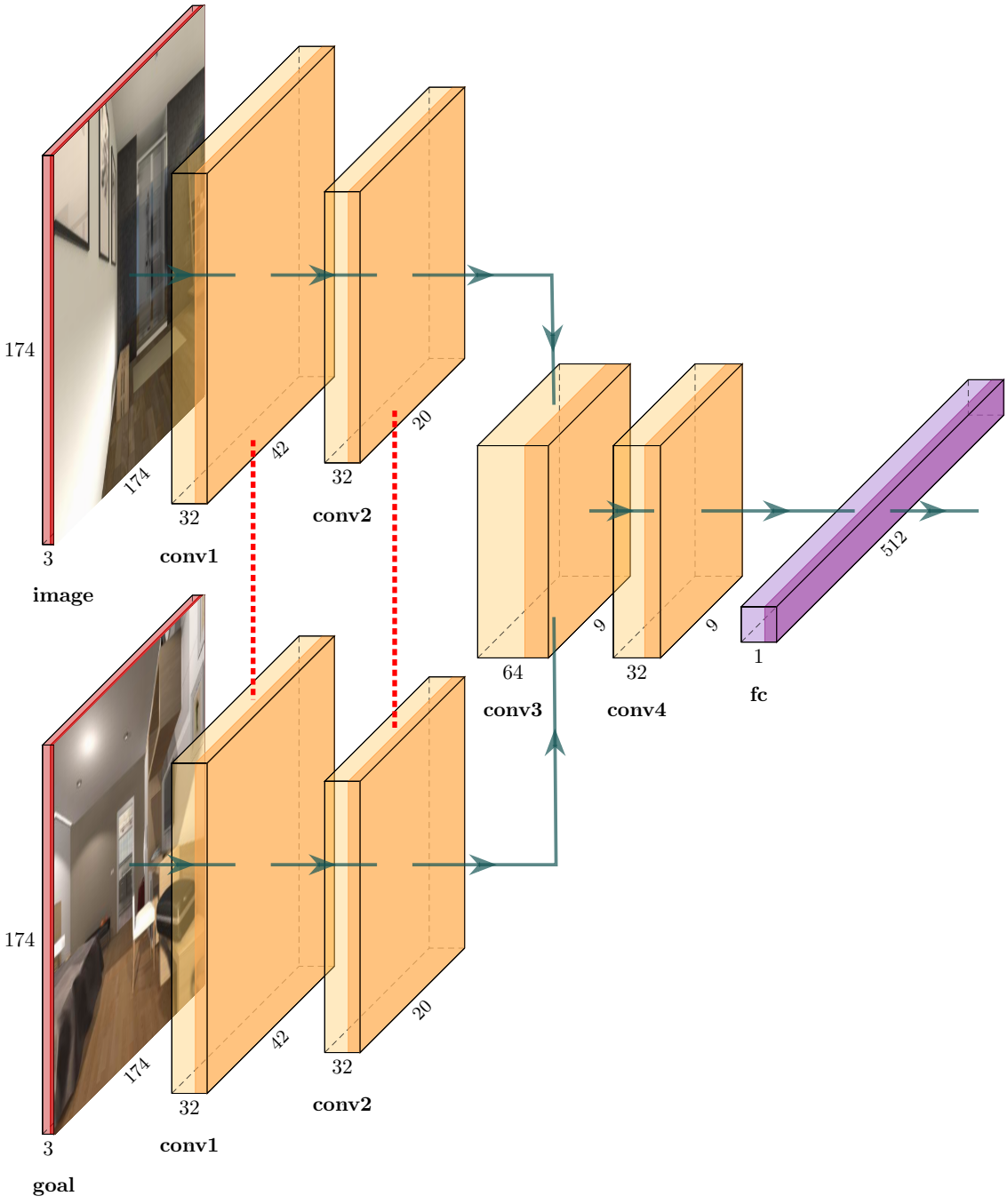


Figure 4.2: **Convolutional base neural network structure.** The feature size after applying each layer is shown in the picture.

if the agent faces a wall, it can instead look at the previous observation and the action taken. Mnih et al. (2013) and Zhu et al. (2017) used past four frames, fed into the network instead of single image input. Jaderberg et al. (2016) used an LSTM Hochreiter and Schmidhuber (1997) instead. We have used the latter in our approach, as we have experimentally found that it was superior to using the past four frames. Past four frames were not enough to capture the complex experience the agent collected when exploring the environment and more frames lead to unmanageable increase of the parameter space size and memory requirements.

4.3 UNREAL Auxiliary Tasks

4.3.1 Reward Prediction

The goal of the agent is to maximize the cumulative reward. It proves beneficial to train the network to predict whether a given state leads to a positive reward or not since it helps the network to build useful features to recognize potentially fruitful states. The agent learns to predict the next reward based on the past three observations (Jaderberg et al., 2016; Munk et al., 2016)[3]. First, a sequence of experiences is sampled from the experience replay buffer such that there is a fixed ratio between the sequences ending with zero reward and the sequences ending with non-zero reward. The output of the fourth convolutional layer computed from all three past observations is merged into a single vector. An additional linear layer and the softmax function are applied to output probabilities of the reward being positive, negative or zero. This new network is then trained using the cross-entropy loss.

4.3.2 Pixel Control

The pixel control task is defined via an additional pseudo-reward function in order to maximize the absolute pixel change. Using this reward, an additional policy is trained that shares most of its parameters with the A2C actor and critic. This policy must be trained using an off-policy RL algorithm since it uses the data sampled from the experience replay buffer generated by the actor. Jaderberg et al. (2016) used the n-step Q-learning

[3] Also here LSTM could be employed, however, we prefer to use the original method from the literature.

loss (Mnih et al., 2013) to update the policy. The observation images are downsized, converted to gray scale, and the absolute difference between two consecutive observations is computed and used as pseudo-rewards for Q-learning (Mnih et al., 2016).

A new head is attached to the output of LSTM. This head consists of deconvolutional layers – upsampling the low-dimensional features back to the size of the downsampled observations. For each action, there is a different output in the last layer to output the Q-function for each pixel. The dueling DQN technique (Wang et al., 2015) is used to improve the performance of the pixel control network. The pixel control network used in our method can be seen in Figure 4.3.

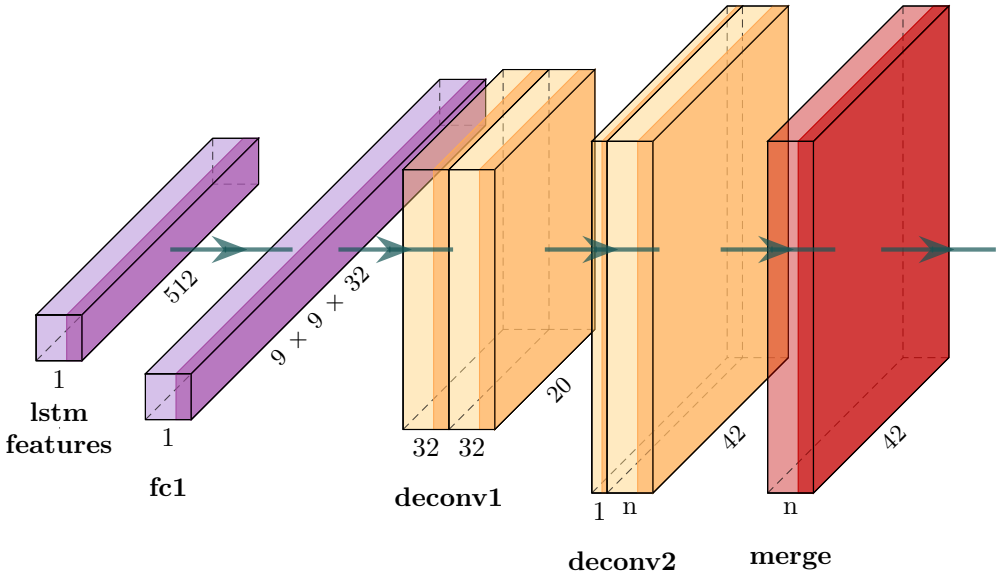


Figure 4.3: **Pixel control network structure.** The feature size after applying each layer is shown in the picture.

4.4 Additional Auxiliary Tasks for Visual Navigation

Motivated by Lange and Riedmiller (2010) and Mirowski et al. (2016), we introduced additional auxiliary tasks that are specific to visual navigation. They were designed to enhance the training process as well as to help the network generalize. We train the model to predict the depth map, image segmentation of the observation and image segmentation of the target. For the image segmentations we map the object-type to the RGB colour space and maximize the distances between each colour in the HSB colour space. The input

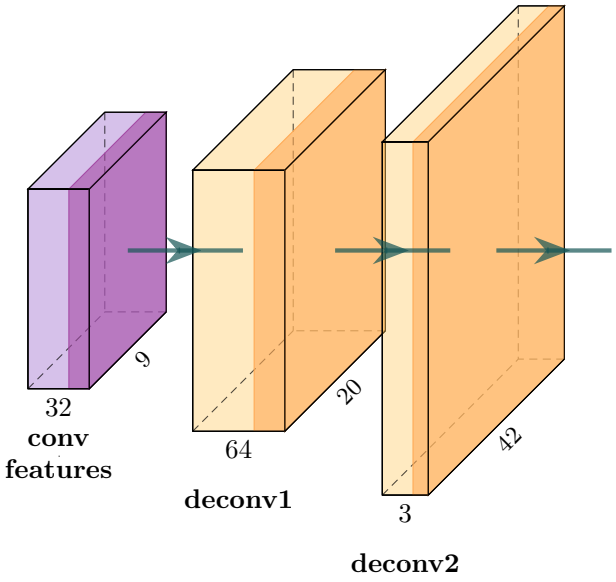


Figure 4.4: Visual navigation auxiliary task network – observation image segmentation and target segmentation prediction. The feature size after applying each layer is shown in the picture.

is passed through a narrow part of the network in autoencoder fashion to improve the quality of features in the shared part of the network. This gives the actor and the critic good features in bottom-most layers with a compact representation of all information needed to reconstruct depth map and image segmentations. These bottom-most layers would otherwise be difficult to train since the network is deep and the loss is noisy due to the imprecise target values computed using the RL algorithm. The image segmentation for the target ensures the network pays attention to what the target is. Otherwise it would be difficult for the network to take the target input into account.

For each visual navigation auxiliary task there is a network attached to the last convolutional layer consisting of deconvolutional layers. The network architecture for the observation image segmentation and the target image segmentation can be seen in Figure 4.4. For the depth map prediction the structure of the network is the same but the intermediate deconvolutional layer has only 32 filters and the last layer has a single channel. True features (the image segmentations for observation and the target and the depth map) are downsampled to a smaller size. The MSE is computed between the outputs of the networks and the true features.

The additional auxiliary tasks for visual navigation also allow for the use of supervised learning to initialize the network with good features in the bottom-most part of the

network since these are the least dependent on the policy. It is costly to render a 3D scene, but it is cheap to pre-compute a dataset of observations taken from the environment and use it for supervised training.

4.5 Environment Complexity

The training of the agent might be hard especially when the environment is large and the initial state is far from the target. To make the task easier for the agent we first sample the initial states closer to the target and gradually increase the distance between the initial state and the target. Let $\tau \in [0, 1]$ be the environment complexity. We define the maximal sampling distance $\mathrm{dmax}_E \colon [0, 1] \to \mathbb{R}$ of an environment E as follows:

$$\mathrm{dmax}_E(\tau) = \tau \max_{s_1, s_2} \{\mathrm{dist}(s_1, s_2)\}\,, \tag{4.1}$$

where $\mathrm{dist}(\cdot, \cdot)$ measures the distance between any two states of the given environment E. Any distance measure can be used, e.g. the Euclidean distance between the corresponding agent positions in the environment.

The initial state s_0 is sampled from the uniform probability distribution over the set of possible initial states closer to any target than $\mathrm{dmax}_E(\tau)$:

$$\mathbb{U}(\{s_1 | s_1 \in \mathcal{S}_{start}, s_2 \in \mathcal{S}_{target}, \mathrm{dist}(s_1, s_2) \leq \mathrm{dmax}_E(\tau)\})\,, \tag{4.2}$$

where the set of target states is denoted by $\mathcal{S}_{target}$. The environment complexity τ starts at a low value, e.g. 0.3, and gradually increases during the training to 1.0.

Chapter 5

Experiments

We have experimentally evaluated the performance of our method A2CAT-VN, using the average episode length and the average episode undiscounted return as performance metrics. The averages are computed Monte Carlo estimates based on 100 rollouts. The randomness comes from the initial state, the non-deterministic behaviour of the environment and the stochasticity of the actor.

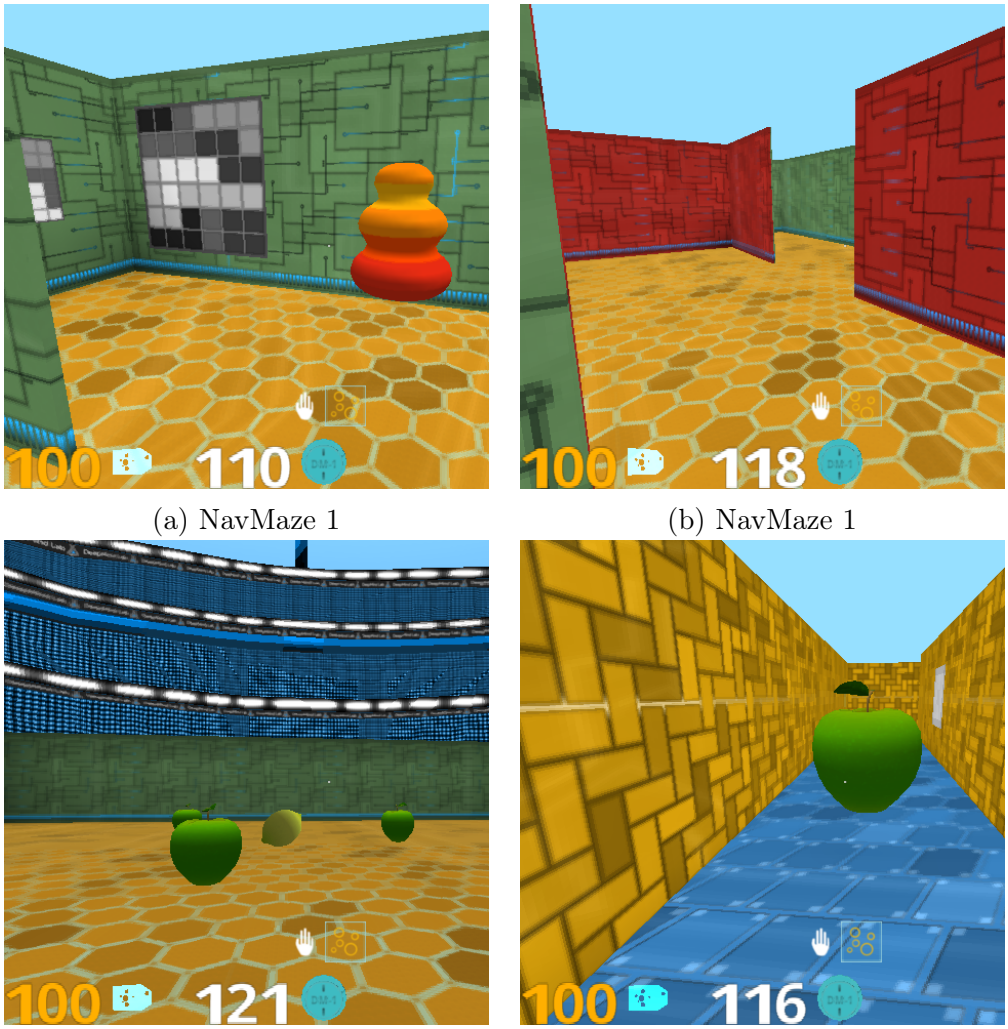
First, the environment simulators used in our experiments will be discussed. We will describe the training configuration. Next, the description of experiments and their results is given.

5.1 Environment Simulators

We have employed three different 3D environment simulators suitable for visual navigation tasks: DeepMind Lab, AI2-THOR, and House3D with SUNCG.

5.1.1 DeepMind Lab

DeepMind Lab (Beattie et al., 2016) is a 3D framework which allows an agent to move and collect objects in synthetic environments. It is fast and highly optimized for training AI agents and the set of allowed actions is customizable. Figure 5.1 shows examples of images from this environment simulator. We used it to compare the proposed algorithm with alternatives from the literature and to pre-train the agent's network for other environments, which sped up the training process.



(a) NavMaze 1 (b) NavMaze 1

(c) SeekAvoid (d) NavMaze 2

Figure 5.1: Sample images from DeepMind Lab framework (Beattie et al., 2016).

5.1.2 AI2-THOR

AI2-THOR (Kolve et al., 2017) is a photo-realistic interactive framework with high-quality indoor images (see Figure 5.2). Most of the environments are a single room and are dynamic, i.e., at the beginning of the episode, various objects can be placed at random positions. The agent moves on a grid: an action moves the agent to a neighboring point on the grid or rotates the agent by 90°. This does not allow for a good generalization since the agent can memorize the finite (and small) number of observations it can possibly receive. Therefore, we have modified the implementation of the AI2-THOR 3D simulator to use continuous space. We have extended the set of possible actions by adding a rotation by an arbitrary angle and a movement by an arbitrary distance. We have also changed the way collisions are handled – when an action would lead to a collision, instead of leaving the agent at the original location, we execute a part of the action until the collision occurs.

(a) bedroom (b) bathroom

(c) kitchen (d) living room

Figure 5.2: Sample images from AI2-THOR framework (Kolve et al., 2017).

5.1.3 House3D with SUNCG

House3D (Wu et al., 2018) is a 3D framework allowing to use the environments from the SUNCG dataset (Song et al., 2017). The SUNCG dataset consists of over 45 000 indoor environments, most of them being two-storey houses and studios. House3D is highly optimized for AI agents training and runs fast on GPUs. Apart from RGB output rendering, it also supports depth map and image segmentation rendering. Illustrative images from this environment simulator are shown in Figure 5.3. The set of actions can be customized in a similar way as in the DeepMind Lab environment simulator.

Figure 5.3: SUNCG (Song et al., 2017) images from the House3D (Wu et al., 2018) framework.

5.2 Action Space

In each of our experiments we used actions from the following set: forward, backward, left, right, rotate-left, rotate-right, tilt-up, tilt-down. The forward and backward actions move the agent in the direction it is currently facing. The left and right actions move the agent in perpendicular directions to the direction it is facing. The rotate-left and rotate-right actions rotate the agent by 30 degrees[1] counter-clockwise and clockwise respectively and the tilt-up and tilt-down actions tilt the agent's camera up or down by 30 degrees.

In real-world environments, the actuators would rarely be able to move the agent precisely. To simulate such a setting, Gaussian noise is added to the position and rotation of the agent after taking an action. More specifically, let $\tilde{s} = (\tilde{x_1}, \tilde{x_2}, \tilde{\theta}, \sigma)$ be the agent's position, horizontal rotation, and tilt of the camera after taking an action before we added the noise. Then the agent's final position and rotation is $s = (x_1, x_2, \theta, \sigma)$, with $x_1 \sim \mathcal{N}(\tilde{x_1}, 0.02^2)$, $x_2 \sim \mathcal{N}(\tilde{x_2}, 0.02^2)$ and $\theta \sim \mathcal{N}(\tilde{\theta}, 2^2)$.

5.3 Training

The reward can be assigned to the agent using different schemes. In our work we give the agent reward one if it reaches the target and zero otherwise. In the training phase, we compute the total gradient as the weighted sum of all the partial gradients: the actor, the critic, the entropy loss, the off-policy critic and the auxiliary tasks. The gradient is clipped so its $l2$-norm does not exceed 0.5 and the RMSprop optimizer is used to optimize the weights. In all experiments we used two Tesla K40 GPUs (10GB each) – one GPU was dedicated for the environment simulator instances and the other for the agent. The parameters used in our method are given in Table 5.1, where f denotes the number of frames processed so far and $f_{\max}$ is the maximum number of frames to be processed during training. Some parameters were chosen to be the same as in Jaderberg et al. (2016) and in Wu et al. (2017), others were chosen experimentally.

[1] One experiment uses 90 ° angles.

Table 5.1: Method parameters.

name	value
discount factor (γ)	0.99
maximum episode length	900
maximum rollout length	20
maximum number of frames ($f_{\max}$)	$4 \cdot 10^7$
number of environment simulator instances	16
replay buffer size	2 000
optimizer	RMSprop
RMSprop alpha	0.99
RMSprop epsilon	10^{-5}
learning rate	$7 \cdot 10^{-4} \frac{f_{\max} - f}{f_{\max}}$
max. gradient norm	0.5
entropy gradient weight	0.001
actor weight	1.0
critic weight	0.5
off-policy critic weight	1.0
pixel control weight	0.05
reward prediction weight	1.0
depth map prediction weight	0.1
observation image segmentation prediction weight	0.1
target segmentation prediction weight	0.1
pixel control discount factor	0.9
pixel control downsize factor	4
auxiliary VN downsize factor	4
pre-training optimizer	Adam
pre-training total epochs	30
pre-training dataset size	$2 \cdot 10^5$

5.4 Partial Observability

We compared two different approaches to resolve the partial observability problem. One approach used by Zhu et al. (2017) and Mnih et al. (2013) concatenates the past four frames as the input to the agent. The other approach (Jaderberg et al., 2016) uses the LSTM network (Hochreiter and Schmidhuber, 1997). We tested both methods on the DeepMind Lab environment simulator because of its great speed and relative simplicity. The allowed actions were forward, backward, left, right, rotate-left, rotate-right. We did not use any noise and the distance by which the actions moved the agent were 0.3 meters for actions forward, backward, left, right. The input to the agent was a single RGB image with the resolution of 84×84 pixels. The network structure based on Jaderberg et al. (2016) was similar in both cases except in the frame concatenation version, where the LSTM was replaced by a linear layer. Both networks used the UNREAL auxiliary tasks (Jaderberg et al., 2016). The algorithms were trained on the DeepMind Lab SeekAvoid environment. The results can be seen in Figure 5.4. The experiment clearly shows that LSTM outperforms the frame concatenation method.

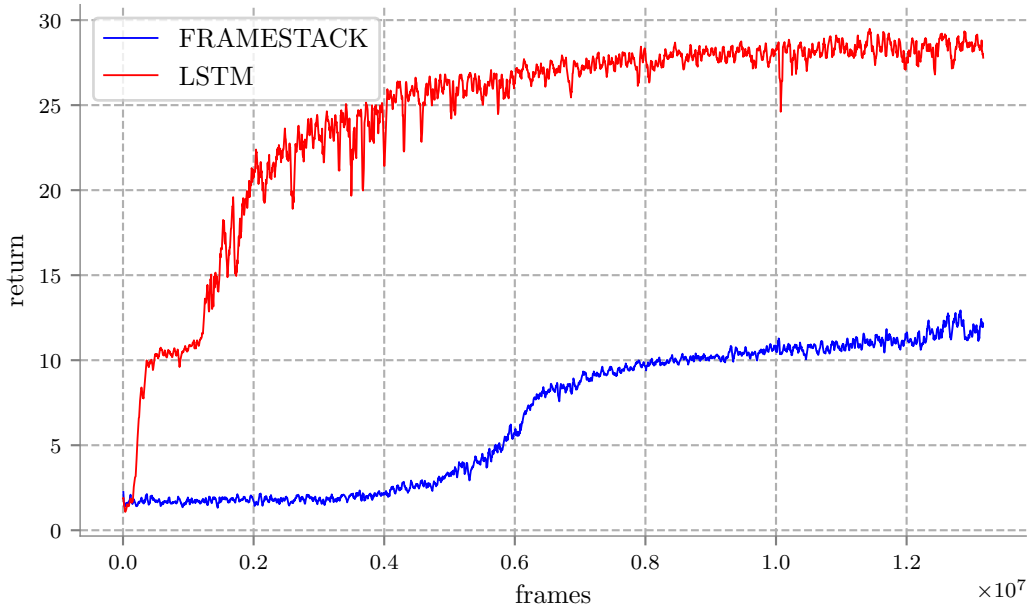


Figure 5.4: Comparison of LSTM model (LSTM) with frame-concatenation model (FRAMESTACK) trained using deterministic UNREAL. Plot shows average return curves during training on DeepMind Lab (Beattie et al., 2016) environment called SeekAvoid.

5.5 AI2-THOR

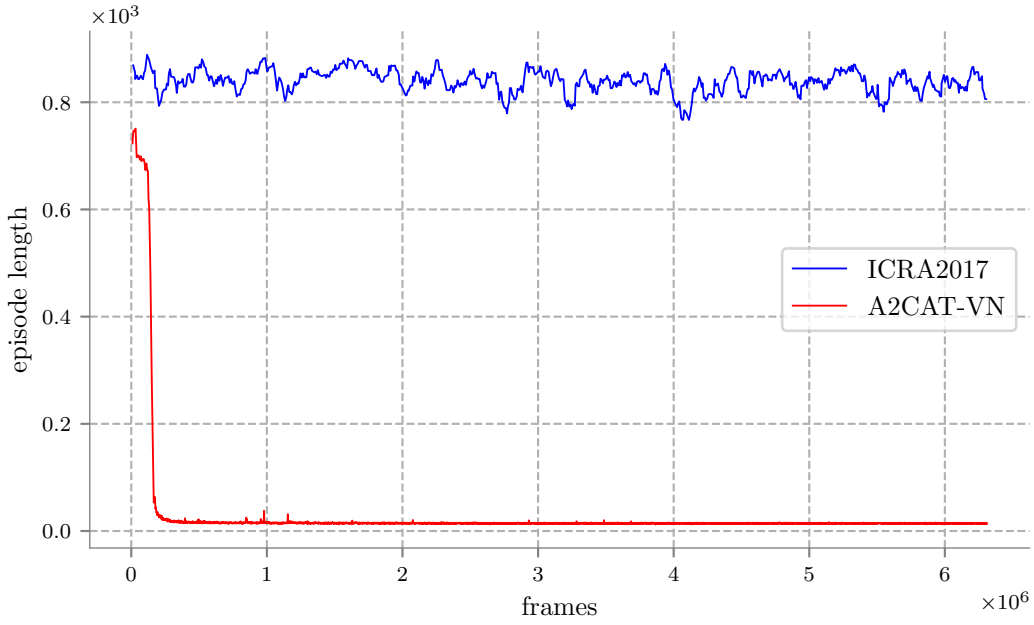


Figure 5.5: Comparison of our model (A2CAT-VN) with target driven visual navigation paper (Zhu et al., 2017) (ICRA2017) trained on four environments from AI2-THOR. Plot shows the average episode length curves during training.

We have trained our algorithm on four environments from AI2-THOR environment simulator with multiple targets. We have used the same set of actions as Zhu et al. (2017) – rotate-left, rotate-right, forward, and backward. Actions forward and backward moved the agent in the direction is was facing by either 0.33 m or −0.33 m. Actions rotate-left and roatate-right rotated the agent by ±90°. No noise was applied. This allowed us to compare our method to Zhu et al. (2017) and also to cache the observations since it turned the problem into an instance of a grid world. The resolution of the input images was 174 × 174 pixels. We used 16 environment simulator instances in parallel for our algorithm each using different environment or different target. We did not use any pre-training nor did we increase the environment complexity. Our method is compared to Zhu et al. (2017). We used their own code for fair comparison, however, the environments we chose for this experiment were bigger and more difficult to navigate than those used in Zhu et al. (2017). The training of our algorithm took roughly 1 day, but we had to stop the training of Zhu et al. (2017) after three days with unsatisfactory results – their implementation did not allow to speed up the training by using GPUs. The results

can be seen in Figure 5.5. Our method A2CAT-VN reached the optimal solution after approximatelly $5 \cdot 10^5$ frames, whereas the method described in Zhu et al. (2017) was not able to move closer to the optimal solution after $7 \cdot 10^6$ frames.

5.6 Continuous AI2-THOR

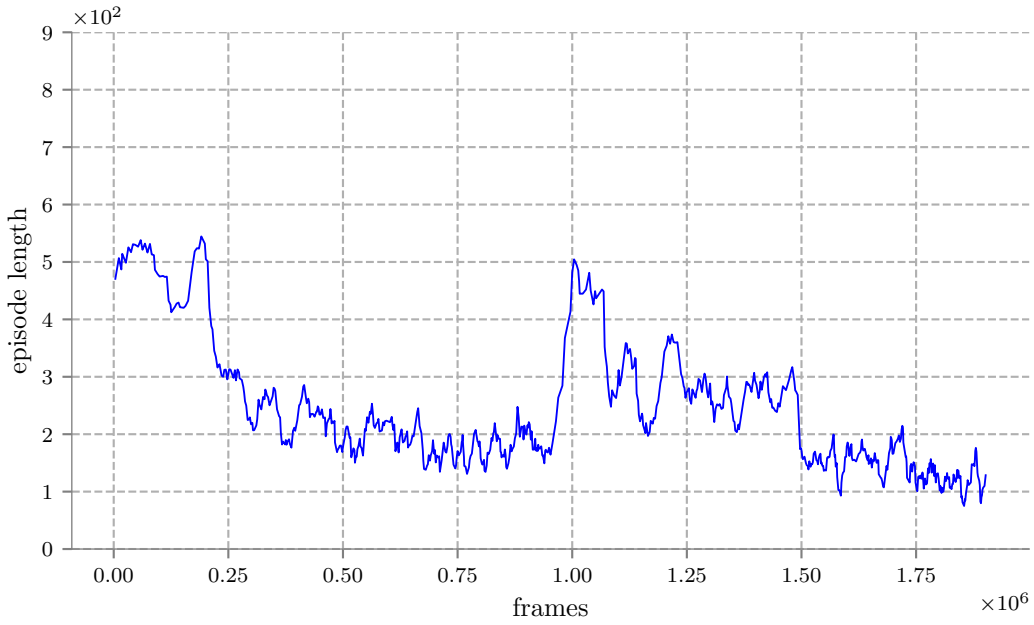


Figure 5.6: This plot shows the average episode length curve while training our algorithm A2CAT-VN in a single environment from AI2-THOR framework (Kolve et al., 2017) with multiple randomly placed targets.

We trained our agent on our modified version of the AI2-THOR environment. We used the full set of actions as described in Section 5.2: forward, backward, left, right, rotate-left, rotate-right, tilt-up, tilt-down. The forward and backward actions moved the agent by 0.5 and -0.2 meters respectively and the left and right actions moved the agent by 0.35 meters. Due to performance issues, however, the noise was only applied in the direction of the movement and no noise was applied in case of tilt-up and tilt-down actions. The agent was trained on a single bedroom environment with multiple targets specified by images. We used 16 environment simulator instances in parallel, each having a different target. The target object was placed randomly to different positions in the environments and the agent was trained to get to close proximity of the target object (1 meter). The resolution of the input images was 174×174 pixels. We did not use pre-training nor did we increase

the environment complexity. The results can be seen in Figure 5.6. The training took 4 days. The AI2-THOR 3D environment simulator was too slow for further experiments. The results show the ability of the agent to navigate in non-static environments and find dynamically placed objects.

5.7 Auxiliary Tasks

We compared our method (A2CAT-VN) with the batched A2C extended with the original two UNREAL auxiliary tasks. Single agent was trained on 16 houses from the SUNCG dataset (Song et al., 2017) using House3D environment simulator. We used the same actions as those described in Section 5.6 except for the tilt-up and tilt-down actions. Inspired by Wu et al. (2018), the agent was trained to find a selected room in the house. The room was given to the agent in the form of an observation taken in a room of the same type. For example, if the target room is the bedroom, the agent is supposed to find any bedroom. The resolution of the input images was 174×174 pixels. We pre-trained our neural network using the data collected from a subset of all houses from SUNCG dataset. The number of images we used for pre-training was approximatelly 20 000 and we trained our network for 30 epochs using the Adam optimizer. The pre-training took roughly one hour. For the full training, we linearly increased the environment complexity from 0.3 at the time step 5M to 1.0 a the time step 10M. The training took rougly two days. The training curves for the average episode length can be seen in Figure 5.7. Our algorithm A2CAT-VN converged much faster with additional auxiliary tasks for visual navigation enabled, reaching the average episode length 200 in $\approx 3 \cdot 10^6$ frames whereas without the additional tasks the training took $\approx 8 \cdot 10^6$ frames to get to the same level.

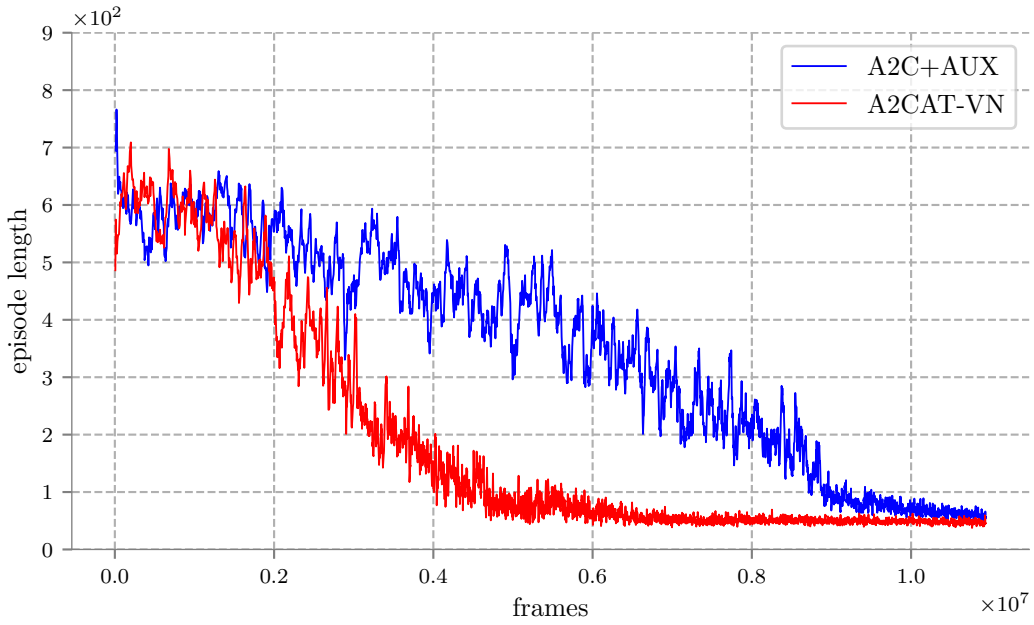


Figure 5.7: Comparison of our method (A2CAT-VN) with the A2C algorithm extended with UNREAL auxiliary tasks (Jaderberg et al., 2016). The training was performed on a set of 16 environments from SUNCG dataset (Song et al., 2017). The plot shows the average episode length curve.

Chapter 6

Conclusions & Future Work

In this thesis, a novel learning architecture A2CAT-VN for visual navigation in indoor environments has been proposed. It is based on a compact deep neural network capable of fast learning over multiple realistic environments, using the batched A2C algorithm extended with auxiliary tasks. By using the target image as an input, the method enables the agent to locate arbitrary goals, as long as their images have been seen during the training phase.

The method was demonstrated on AI2-THOR and House3D environment simulators. First, we have shown that the basic batched A2C algorithm benefits from the addition of the UNREAL auxiliary tasks (Jaderberg et al., 2016). Further performance gain was achieved by employing our novel auxiliary tasks specifically designed to improve the performance of the algorithm when applied to visual navigation. These tasks included depth map, observation image segmentation, and target image segmentation prediction. After the training, our tasks did not require any additional input to the agent except for the observation image. A large part of the DNN could be pre-trained with these auxiliary tasks, reducing the training costs. Additionally, we have proposed a way to improve the training by incrementally increasing the environment complexity.

When applied to AI2-THOR environments, our method was able to converge at least an order of magnitude faster than an alternative state-of-the-art method (Zhu et al., 2017), which also allows for using multiple targets and was demonstrated in indoor environments, similarly to our method. The auxiliary tasks introduced were shown to reduce the number of frames needed to train the agent by the factor of two. Our method was able to achieve satisfactory results when applied to dynamically changing environments where the goal

was to find randomly positioned objects.

Future research should investigate the potential effects of using supervised pre-training of additional auxiliary tasks for visual navigation on the training performance. The network could also be forced to output the input image and, therefore, to build a compact representation of the observation.

We would also like to explore the application of our method to more 3D environments (perhaps outdoor environments) and potentially apply it to real-world environments. An efficient 3D environment simulator needs to be developed, which would allow us to change the lighting conditions and other details, and would be realistic both in terms of the physics used and the image quality.

Another line of research needs to be conducted on the ability of the method to generalize to unseen targets. Also, we believe the ability of the agent to deal with unseen environments might outline a critical area for future research.

Bibliography

J. A. Bagnell, Sham M Kakade, Jeff G. Schneider, and Andrew Y. Ng. Policy search by dynamic programming. In S. Thrun, L. K. Saul, and B. Schölkopf, editors, *Advances in Neural Information Processing Systems 16*, pages 831–838. MIT Press, 2004. URL http://papers.nips.cc/paper/2378-policy-search-by-dynamic-programming.pdf.

André M.S. Barreto, Doina Precup, and Joelle Pineau. Practical kernel-based reinforcement learning. *Journal of Machine Learning Research*, 17(67):1–70, 2016. URL http://jmlr.org/papers/v17/13-134.html.

Charles Beattie, 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. DeepMind Lab, 2016.

Richard Bellman. *Dynamic Programming*. Princeton University Press, Princeton, NJ, USA, 1 edition, 1957. URL http://books.google.com/books?id=fyVtp3EMxasC&pg=PR5&dq=dynamic+programming+richard+e+bellman&client=firefox-a#v=onepage&q=dynamic%20programming%20richard%20e%20bellman&f=false.

Francisco Bonin-Font, Alberto Ortiz, and Gabriel Oliver. Visual navigation for mobile robots: A survey. *Journal of Intelligent and Robotic Systems*, 53(3):263, May 2008. ISSN 1573-0409. doi: 10.1007/s10846-008-9235-4.

Jake Bruce, Niko Suenderhauf, Piotr Mirowski, Raia Hadsell, and Michael Milford. One-shot reinforcement learning for robot navigation with interactive replay, 2017.

Lasse Espeholt, Hubert Soyer, Remi Munos, Karen Simonyan, Volodymir Mnih, Tom Ward, Yotam Doron, Vlad Firoiu, Tim Harley, Iain Dunning, Shane Legg, and Koray Kavukcuoglu. IMPALA: Scalable distributed deep-RL with importance weighted actor-learner architectures, 2018.

Yaroslav Ganin, Tejas Kulkarni, Igor Babuschkin, S. M. Ali Eslami, and Oriol Vinyals. Synthesizing programs for images using reinforced adversarial learning, 2018.

Milica Gašić. Lecture notes on temporal-difference methods. URL http://mi.eng.cam.ac.uk/~mg436/LectureSlides/MLSALT7/L3.pdf, 2017.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

I. Grondman, L. Buşoniu, G.A.D. Lopes, and R. Babuška. A survey of actor-critic reinforcement learning: Standard and natural policy gradients. *IEEE Transactions on Systems, Man, and Cybernetics. Part C: Applications and Reviews*, 42(6):1291–1307, 2012.

Leonid Gurvits, L. J. Lin, and Stephen José Hanson. Incremental learning of evaluation functions for absorbing Markov chains: New methods and theorems. *preprint*, 1994.

Tatsunori Hashimoto, Yi Sun, and Tommi Jaakkola. From random walks to distances on unweighted graphs. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 3429–3437. Curran Associates, Inc., 2015. URL `http://papers.nips.cc/paper/6009-from-random-walks-to-distances-on-unweighted-graphs.pdf`.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.

Geoffrey Hinton, Nitish Srivastava, and Kevin Swersky. Lecture notes on neural networks for machine learning. URL `http://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf`, 2014.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997. doi: 10.1162/neco.1997.9.8.1735.

Max Jaderberg, Volodymyr Mnih, Wojciech Marian Czarnecki, Tom Schaul, Joel Z Leibo, David Silver, and Koray Kavukcuoglu. Reinforcement learning with unsupervised auxiliary tasks, 2016.

Alan F. Karr. *Probability*. Springer Texts in Statistics. Springer New York, 2012. ISBN 9781461208914. URL `https://books.google.cz/books?id=34XuBwAAQBAJ`.

Kiyosumi Kidono, Jun Miura, and Yoshiaki Shirai. Autonomous visual navigation of a mobile robot using a human-guided experience. *Robotics and Autonomous Systems*, 40 (2):121–130, August 2002. doi: 10.1016/S0921-8890(02)00237-3.

Dongsung Kim and Ramakant Nevatia. Symbolic navigation with a generic map. *Autonomous Robots*, 6(1):69–88, January 1999. ISSN 1573-7527. doi: 10.1023/A:1008824626321.

H. Jin Kim, Michael I. Jordan, Shankar Sastry, and Andrew Y. Ng. Autonomous helicopter flight via reinforcement learning. In S. Thrun, L. K. Saul, and B. Schölkopf, editors, *Advances in Neural Information Processing Systems 16*, pages 799–806. MIT Press, 2004. URL `http://papers.nips.cc/paper/2455-autonomous-helicopter-flight-via-reinforcement-learning.pdf`.

Nate Kohl and Peter Stone. Policy gradient reinforcement learning for fast quadrupedal locomotion. In *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA '04. 2004*, volume 3, pages 2619–2624, April 2004. doi: 10.1109/ROBOT.2004.1307456.

Thomas Kollar and Nicholas Roy. Trajectory optimization using reinforcement learning for map exploration. *The International Journal of Robotics Research*, 27(2):175–196, 2008. doi: 10.1177/0278364907087426.

Eric Kolve, Roozbeh Mottaghi, Daniel Gordon, Yuke Zhu, Abhinav Gupta, and Ali Farhadi. AI2-THOR: An interactive 3D environment for visual AI. *arXiv*, December 2017.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. ImageNet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012. URL `http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf`.

Brenden M. Lake, Tomer D. Ullman, Joshua B. Tenenbaum, and Samuel J. Gershman. Building machines that learn and think like people. *Behavioral and Brain Sciences*, 40, 2017. doi: 10.1017/S0140525X16001837.

S. Lange and M. Riedmiller. Deep auto-encoder neural networks in reinforcement learning. In *The 2010 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8, July 2010. doi: 10.1109/IJCNN.2010.5596468.

Chi-Tat Law and Joshua I. Gold. Reinforcement learning can account for associative and perceptual learning on a visual-decision task. *Nature Neuroscience*, 12, April 2009. doi: 10.1038/nn.2304. Article.

Scott Lenser and Manuela Veloso. Visual sonar: fast obstacle avoidance using monocular vision. In *Proceedings 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003)*, volume 1, pages 886–891, October 2003. doi: 10.1109/IROS.2003.1250741.

Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning, 2015.

Jeff Michels, Ashutosh Saxena, and Andrew Y. Ng. High speed obstacle avoidance using monocular vision and reinforcement learning. In *Proceedings of the 22Nd International Conference on Machine Learning*, ICML '05, pages 593–600, New York, NY, USA, 2005. ACM. ISBN 1-59593-180-5. doi: 10.1145/1102351.1102426.

Piotr Mirowski, Razvan Pascanu, Fabio Viola, Hubert Soyer, Andrew J. Ballard, Andrea Banino, Misha Denil, Ross Goroshin, Laurent Sifre, Koray Kavukcuoglu, Dharshan Kumaran, and Raia Hadsell. Learning to navigate in complex environments, 2016.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing Atari with deep reinforcement learning, 2013.

Volodymyr Mnih, 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. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, February 2015. ISSN 0028-0836. doi: 10.1038/nature14236.

Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pages 1928–1937, 2016.

Jelle Munk, Jens Kober, and Robert Babuska. Learning state representation for deep actor-critic control. In *2016 IEEE 55th Conference on Decision and Control (CDC)*. IEEE, dec 2016. doi: 10.1109/cdc.2016.7798980.

Andrew Y. Ng and Michael Jordan. PEGASUS: A policy search method for large MDPs and POMDPs. In *Proceedings of the Sixteenth Conference on Uncertainty in Artificial Intelligence*, UAI'00, pages 406–415, San Francisco, CA, USA, 2000. Morgan Kaufmann Publishers Inc. ISBN 1-55860-709-9. URL `http://dl.acm.org/citation.cfm?id=2073946.2073994`.

Giuseppe Oriolo, Marilena Vendittelli, and Giovanni Ulivi. On-line map building and navigation for autonomous mobile robots. In *Proceedings of 1995 IEEE International Conference on Robotics and Automation*, volume 3, pages 2900–2906, May 1995. doi: 10.1109/ROBOT.1995.525695.

Śaunak Ormoneit, Dirkand Sen. Kernel-based reinforcement learning. *Machine Learning*, 49(2):161–178, November 2002. ISSN 1573-0565. doi: 10.1023/A:1017928328829.

Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in PyTorch. In *NIPS-W*, 2017.

Jan Peters and Stefan Schaal. Reinforcement learning of motor skills with policy gradients. *Neural networks : the official journal of the International Neural Network Society*, 21 4:682–97, 2008.

Anthony Remazeilles, François. Chaumette, and Patrick Gros. Robot motion control from a visual memory. In *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA '04. 2004*, volume 5, pages 4695–4700, April 2004. doi: 10.1109/ROBOT.2004.1302458.

Martin Riedmiller. Neural fitted Q iteration – first experiences with a data efficient neural reinforcement learning method. In João Gama, Rui Camacho, Pavel B. Brazdil, Alípio Mário Jorge, and Luís Torgo, editors, *Machine Learning: ECML 2005*, pages 317–328, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg. ISBN 978-3-540-31692-3.

Herbert Robbins and Sutton Monro. A stochastic approximation method. *Ann. Math. Statist.*, 22(3):400–407, September 1951. doi: 10.1214/aoms/1177729586.

Parvaneh Saeedi, Peter D. Lawrence, and David G. Lowe. Vision-based 3-D trajectory tracking for unknown environments. *IEEE Transactions on Robotics*, 22(1):119–136, February 2006. ISSN 1552-3098. doi: 10.1109/TRO.2005.858856.

Ahmad Sallab, Mohammed Abdou, Etienne Perot, and Senthil Yogamani. Deep reinforcement learning framework for autonomous driving. *Electronic Imaging*, 2017:70–76, January 2017. doi: 10.2352/ISSN.2470-1173.2017.19.AVM-023.

Alex Sherstinsky. Fundamentals of recurrent neural network (RNN) and long short-term memory (LSTM) network, 2018.

Natasha Sigala and Nikos K. Logothetis. Visual categorization shapes feature selectivity in the primate temporal cortex. *Nature*, 415(6869):318–320, 2002. ISSN 1476-4687. doi: 10.1038/415318a.

David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. Deterministic policy gradient algorithms. In *Proceedings of the 31st International Conference on International Conference on Machine Learning*, volume 32 of *ICML'14*, pages 387–395, Beijing, China, 2014. JMLR.org. URL `http://dl.acm.org/citation.cfm?id=3044805.3044850`.

David Silver, 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 Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489, January 2016. ISSN 0028-0836. doi: 10.1038/nature16961.

David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, L Robert Baker, Matthew Lai, Adrian Bolton, Yutian Chen, Timothy P. Lillicrap, Fan Hui, Laurent Sifre, George van den Driessche, Thore Graepel, and Demis Hassabis. Mastering the game of Go without human knowledge. *Nature*, 550:354–359, October 2017.

Shuran Song, Fisher Yu, Andy Zeng, Angel X Chang, Manolis Savva, and Thomas Funkhouser. Semantic scene completion from a single depth image. *Proceedings of 29th IEEE Conference on Computer Vision and Pattern Recognition*, 2017.

Jost Tobias Springenberg, Alexey Dosovitskiy, Thomas Brox, and Martin Riedmiller. Striving for simplicity: The all convolutional net, 2014.

Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, Upper Saddle River, NJ, USA, adaptive computation and machine learning series edition, November 2018. ISBN 0262039249. URL `https://www.ebook.de/de/product/32966850/richard_s_sutton_andrew_g_barto_reinforcement_learning.html`.

Richard S Sutton, David A McAllester, Satinder P Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. In *Advances in neural information processing systems*, pages 1057–1063, 2000.

Masahiro Tomono. 3-D object map building using dense object models with SIFT-based recognition features. In *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1885–1890, October 2006. doi: 10.1109/IROS.2006.282312.

John N. Tsitsiklis and Benjamin Van Roy. An analysis of temporal-difference learning with function approximation. Technical report, 1997.

Aaron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. WaveNet: A generative model for raw audio, 2016.

Ziyu Wang, Tom Schaul, Matteo Hessel, Hado van Hasselt, Marc Lanctot, and Nando de Freitas. Dueling network architectures for deep reinforcement learning, 2015.

Christopher J. C. H. Watkins and Peter Dayan. Q-learning. In *Machine Learning*, pages 279–292, 1992.

Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8(3):229–256, May 1992. ISSN 1573-0565. doi: 10.1007/BF00992696.

David Wooden. A guide to vision-based map building. *IEEE Robotics Automation Magazine*, 13(2):94–98, June 2006. ISSN 1070-9932. doi: 10.1109/MRA.2006.1638021.

Yi Wu, Yuxin Wu, Georgia Gkioxari, and Yuandong Tian. Building generalizable agents with a realistic and rich 3D environment, 2018.

Yuhuai Wu, Elman Mansimov, Shun Liao, Roger B. Grosse, and Jimmy Ba. Scalable trust-region method for deep reinforcement learning using Kronecker-factored approximation. *CoRR*, abs/1708.05144, 2017.

Linhai Xie, Sen Wang, Andrew Markham, and Niki Trigoni. Towards monocular vision based obstacle avoidance through deep reinforcement learning, 2017.

Yuke Zhu, Roozbeh Mottaghi, Eric Kolve, Joseph J. Lim, Abhinav Gupta, Li Fei-Fei, and Ali Farhadi. Target-driven visual navigation in indoor scenes using deep reinforcement learning. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3357–3364, May 2017. doi: 10.1109/ICRA.2017.7989381.

List of Appendices

Appendix A

Vision-based Navigation Using Deep Reinforcement Learning Paper

Vision-based Navigation Using Deep Reinforcement Learning

Jonáš Kulhánek[1], Erik Derner[2], Tim de Bruin[1], and Robert Babuška[3]

Abstract— **Deep reinforcement learning (RL) has been successfully applied to a variety of game-like environments. However, the application of deep RL to visual navigation with realistic environments is a challenging task. We propose a novel learning architecture capable of navigating an agent to a target given by an image. To achieve this, we have extended the batched A2C algorithm with auxiliary tasks designed to improve visual navigation performance. We propose three additional auxiliary tasks for the prediction of the depth-map, image segmentation and the target image segmentation. These tasks enable the use of supervised learning to pre-train a major part of the network and to substantially reduce the number of training steps. The training performance can be further improved by increasing the environment complexity gradually over time. An efficient neural network structure is proposed, which is capable of learning for multiple targets in multiple environments. Our method navigates in continuous state spaces and on the AI2-THOR environment simulator surpasses the performance of state-of-the-art goal-oriented visual navigation methods from the literature.**

Index Terms— **Robot navigation, deep reinforcement learning, actor-critic, auxiliary tasks.**

I. Introduction

Visual navigation is the problem of navigating an agent in an environment using camera input only. The agent is given a target image (an image it will see from the target position), and its goal is to move from its current position to the target by applying a sequence of actions, based on the camera observations only. We focus on the case when the environment is initially unknown, i.e., no explicit map is available. Such a visual navigation problem can be formalized as a reinforcement learning (RL) problem [1]. Two main challenges in the RL formulation are the dimensionality of the agent's observation space and the fact that the actual state is only partially observable from the images.

Observation space dimensionality can be reduced by using hand-crafted features, or by using learned features trained on either the training dataset or completely different one, e.g., ResNet [2] features automatically extracted from the image [3], [4]. A different method was proposed in [5] that uses image segmentation and a depth map as the inputs to the agent. It was trained and evaluated on houses from SUNCG dataset [6] and the trained agent was able to find multiple targets specified as a separate input to the agent.

Raw high-dimensional input images can also be used directly for navigation [7], [8]. These two methods extend the asynchronous advantage actor-critic (A3C) algorithm with auxiliary tasks to stabilize the training and make it more efficient when the reward is sparse. They, however, use the DeepMind Lab [9] game simulator which is much simpler than realistic simulators [10], [5], [6]. The only method that relies on visual input only in a realistic indoor-scene environment is [3]. However, it was applied to AI2-THOR environment [10] which contains small single-room environments and the action space discretized the environment into a simple grid world.

In our approach, the agent learns to navigate based on the observed raw images only, as opposed to [3], which uses ResNet features. The learning algorithm is based on the batched version of advantage actor-critic (A2C) [11], extended with auxiliary tasks to help the agent to learn useful features also in the absence of informative rewards. During the training of the deep network, we do involve depth-maps and image segmentation as inputs to auxiliary tasks. In addition, we propose a method to pre-train the neural network before the reinforcement learning algorithm is applied. This is accomplished by transfer learning from one environment to another, gradually increasing the environment complexity. Finally, to address the partial observability problem, we propose a novel neural network architecture that is both efficient and compact. We evaluate our method in realistic indoor-scene environments similar to [3] and [8].

II. Preliminaries

A. Formal setting

The visual navigation problem is a partially observable Markov decision process (POMDP). For example, when the agent faces a wall, there are many states yielding the same or very similar image. However, for the ease of notation, we will introduce the problem as an instance of a standard MDP. We will use the state $s_t \in \mathcal{S}$ as if it was available to the agent. Later, we will replace the state by a sequence of past observations $o_1, o_2, \ldots, o_t$.

At the beginning of each learning episode, the agent starts from state s_0 which is uniformly sampled from the set of all possible initial states $\mathcal{S}_{start}$: $s_0 \sim \mathbb{U}(\mathcal{S}_{start})$.[1] At discrete

[1]Jonáš Kulhánek and Tim de Bruin are with Cognitive Robotics, Faculty of 3mE, Delft University of Technology, 2628 CD Delft, The Netherlands jonas.kulhanek@live.com, t.d.debruin@tudelft.nl

[2]Erik Derner is with the Czech Institute of Informatics, Robotics and Cybernetics, Czech Technical University in Prague, 16636 Prague, Czech Republic and with the Department of Control Engineering, Faculty of Electrical Engineering, Czech Technical University in Prague, 16627 Prague, Czech Republic erik.derner@cvut.cz

[3]Robert Babuška is with Cognitive Robotics, Faculty of 3mE, Delft University of Technology, 2628 CD Delft, The Netherlands and with the Czech Institute of Informatics, Robotics and Cybernetics, Czech Technical University in Prague, 16636 Prague, Czech Republic r.babuska@tudelft.nl

This work was supported by the European Regional Development Fund under the project Robotics for Industry 4.0 (reg. no. CZ 02.1.01/0.0/0.0/15_003/0000470).

[1]Other distributions than the uniform one can be used.

time steps $t = 0, 1, 2, \ldots$ the agent executes actions a_t. As a result of each action, the agent moves to the next state s_{t+1} and receives reward r_{t+1}. The experience the agent collected in a single episode is defined as the following sequence:

$$\zeta = s_0, a_0, r_1, s_1, a_1, r_2, \ldots . \quad (1)$$

An episode ends when the agent reaches the target or after a maximum number of time steps has elapsed. For the purpose of learning, the episode is split into equally long *rollouts*, where the last rollout obviously can be shorter. The experience collected in a single rollout of length ℓ is defined as:

$$\zeta_t^\ell = s_t, a_t, r_{t+1}, s_{t+1}, a_{t+1}, r_{t+2}, ..., r_{t+\ell+1}, s_{t+\ell+1} . \quad (2)$$

B. Advantage Actor-Critic Algorithms (A2C)

Actor-critic algorithms are suitable for continuous state spaces [12]. The critic is an approximator of the state-value function: $v_{\theta_v} : \mathcal{S} \to \mathbb{R}$, parameterized by θ_v, while the actor is an approximator of the policy. We use a stochastic policy $\pi_{\theta_p}(a|s)$ which is a probability distribution over the discrete set of possible actions, conditioned on the state $s \in \mathcal{S}$, and parameterized by θ_p. Let the bootstrapped n-step return g_t^n be defined as:

$$g_t^n = r_{t+1} + \gamma r_{t+2} + \ldots + \gamma^{n-1} r_{t+n} + \gamma^n \mathbb{1}_{\mathrm{c}} v_{\theta_v}(s_{t+n}) , \quad (3)$$

where $\mathbb{1}_{\mathrm{c}}$ is zero if the episode ended during the rollout and one otherwise and $n \geq 1$. The actor is updated similarly to REINFORCE algorithm [13] with advantage estimates from the critic. The gradient of the actor's loss function J_p from [14] is given by:

$$\frac{\partial J_p}{\partial \theta_p} = -\sum_{i=t}^{t+\ell} \frac{\partial}{\partial \theta_p} \log \pi_{\theta_p}(a_i|s_i)(g_i^{t+\ell-i+1} - v_{\theta_v}(s_i)) . \quad (4)$$

The term $g_i^{t+\ell-i+1} - v_{\theta_v}(s_i)$ is referred to as the advantage function. The critic is updated using n-step temporal difference learning: the Mean squared error (MSE) between the bootstrapped n-step return and the critic output is computed and a gradient descent update is applied. The gradient of the critic's loss function J_v is:

$$\frac{\partial J_v}{\partial \theta_v} = \sum_{i=t}^{t+\ell} \frac{\partial}{\partial \theta_v} \frac{1}{2} \left(g_i^{t+\ell-i+1} - v_{\theta_v}(s_i)\right)^2 . \quad (5)$$

To ensure exploration, the negative entropy of the actor is added to the total loss. The negative entropy of the actor in state s is defined as:

$$H^-(s, \theta_p) = \sum_{a \in \mathcal{A}} \pi_{\theta_p}(a|s) \log \pi_{\theta_p}(a|s) \quad (6)$$

and its gradient on the rollout data is:

$$\frac{\partial J_e}{\partial \theta_p} = \sum_{i=t}^{t+\ell} \sum_{a \in \mathcal{A}} \frac{\partial}{\partial \theta_p} \pi_{\theta_p}(a|s_i) \log \pi_{\theta_p}(a|s_i) . \quad (7)$$

Note that the above setting differs from the one given in [1], which uses n-step forward view to compute the return g. When DNNs are used as the function approximators for the actor and critic, it is beneficial to optimize on multiple time-steps in a single batch. We therefore use only the rollout data to optimize all time-steps in the rollout at once. The estimated returns are a mixture of returns with different length for each state, which was proven to have the error reduction property in the discrete RL setting [15], [16].

As the critic and actor can share knowledge about the environment, they can share some of their parameters, which leads to an improved learning performance. For example, when using neural networks for visual tasks, the bottom-most convolutional layers used in both the actor and the critic need to learn the same convolutional filters. It is therefore beneficial to share their parameters as there are fewer to train, the hypothesis space has fewer dimensions and the search for an optimal solution is simpler. The A2C algorithm [17] has been adapted for the use with DNNs by introducing the following two modifications:

1) *Batched Advantage Actor Critic (A2C).* In batched A2C [11], there are k different environments. At each time step, k actions are sampled by the actor, one for each environment. The rollouts collected from the environments are used to optimize the actor and the critic in a single batch. This process can be viewed as having k separate instances of A2C, each updating the same shared parameters. As shown in [17], the use of multiple environments has a stabilizing effect on the training, similarly to using an experience buffer [18].
2) *Off-policy Critic Updates.* Collecting observations can be costly, especially when the environment framework has to simulate physics and render 3D scenes. For an algorithm to be efficient, it needs to learn as much as possible from the experiences collected so far. To improve the data efficiency and the stability of the algorithm, a memory of past experiences called the experience buffer is used. It keeps the last n_e experiences, i.e., observations, actions, rewards, and terminals[2]. At each learning step, a sequence of experiences is sampled from the buffer and it is used to compute the bootstrapped n-step returns (3) and so to train the critic.

C. UNREAL Auxiliary Tasks

Deep RL algorithms are commonly enhanced with auxiliary tasks in order to improve their learning performance. For instance, in [7] the A3C algorithm was extended with two auxiliary tasks, *reward prediction* and *pixel control*. The former predicts the sign of the reward based on past four observations and the latter uses an additional pseudo-reward function to learn a policy that maximizes the absolute pixel change. The batched A2C can be enhanced in the same way; more details are given in Section III-C.

III. Proposed Learning Architecture

Our method extends batched A2C algorithm with UNREAL auxiliary tasks and additional auxiliary tasks for

[2]A terminal is the indicator of the episode ending in a particular time step.

visual navigation. We call the method A2CAT-VN, which is an abbreviation of A2C with Auxiliary Tasks for Visual Navigation. We have made its implementation[3] as well as a framework implementing several deep RL algorithms[4] publicly available on GitHub.

A. Neural Network

The deep neural network used in our method consists of several modules: convolutional base, LSTM, actor, critic and auxiliary tasks, see Fig. 1. In the sequel, we explain the individual blocks one by one.

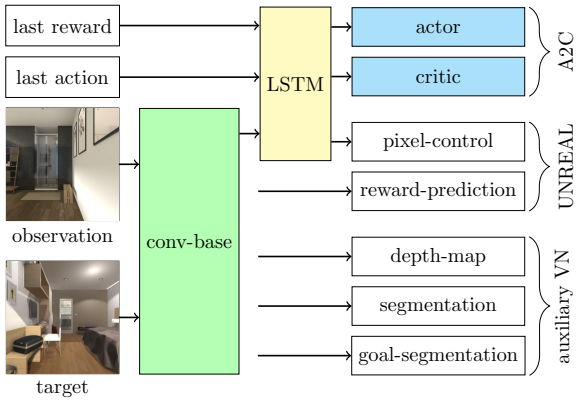


Fig. 1: A2CAT-VN neural network architecture.

The convolutional base is depicted in Fig. 2. Its inputs are the observed image and the target image, each entering into a separate stream of two convolutional layers with shared weight parameters. The outputs of the second layer are concatenated and passed to additional two convolutional layers, followed by a single fully-connected linear layer.

Each of these layers is followed by the ReLU activation function. We do not employ maxpool layers [2], instead, the images are down-sampled by using stride only as suggested in [19]. The convolutional base features are merged with the previous action and the previous reward and passed to the long short-term memory (LSTM) layer [20].

The previous action is encoded using one-hot encoding and the reward is clipped to the interval $[-1, 1]$. LSTM features are used as the input for both the actor and the critic, as well as for the pixel control auxiliary task. Let $\phi(x)$ be the LSTM features of an input x (LSTM features are computed from the convolutional features and therefore are a function of the input). Note that the input is composed of the image observation, the target image, the last action and the last reward, as well as the previous LSTM state. The critic is an affine transformation of the LSTM features and the actor is the result of the softmax function applied to an affine transformation of the LSTM features.

[3]https://github.com/jkulhanek/a2cat-vn-pytorch
[4]https://github.com/jkulhanek/deep-rl-pytorch

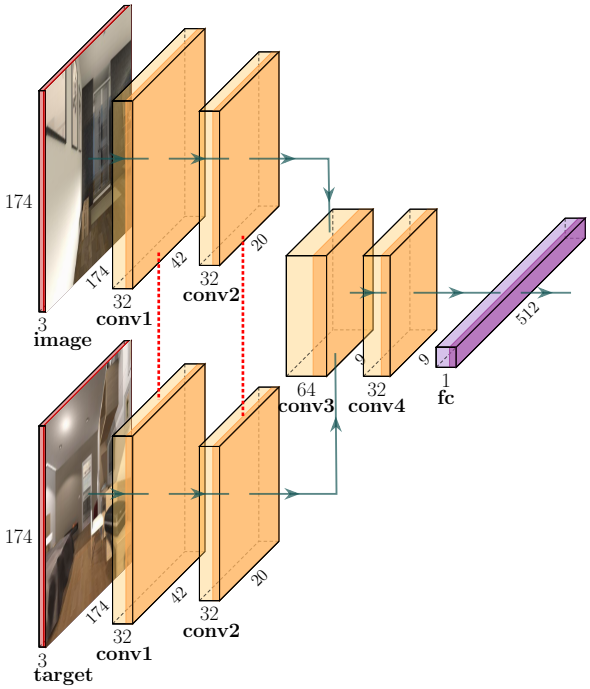


Fig. 2: Convolutional base. The feature size after applying each layer is shown in the picture.

B. Resolving Partial Observability

The partial observability of the environment does not allow the agent to uniquely distinguish which state it occupies based on a sole observation. Using previous observations can, however, greatly improve its ability to navigate in the environment. For example, if the agent faces a wall, it can instead look at the previous observation and the action taken. The authors of [18] and [3] used past four frames, fed into the network instead of single image input. In [7], an LSTM memory [20] was used instead. We have used the latter in our approach, as we have experimentally found that it was superior to using the past four frames. Past four frames were not enough to capture the complex experience the agent collected when exploring the environment and more frames lead to unmanageable increase of the parameter space size and memory requirements.

C. UNREAL Auxiliary Tasks

1) Reward Prediction: The goal of the agent is to maximize the cumulative reward. It proves beneficial to train the network to predict whether a given state leads to a positive reward or not since it helps the network to build useful features to recognize potentially fruitful states. The agent learns to predict the next reward based on the past three observations [7], [21][5]. First, a sequence of experiences is sampled from the experience replay buffer such that there is

[5]Also here LSTM could be employed, however, we prefer to use the original method from the literature.

a fixed ratio between the sequences ending with zero reward and the sequences ending with non-zero reward. The output of the fourth convolutional layer computed from all three past observations is merged into a single vector. An additional linear layer and the softmax function are applied to output probabilities of the reward being positive, negative or zero. This new network is then trained using the cross-entropy loss.

2) Pixel Control: The pixel control task is defined via an additional pseudo-reward function in order to maximize the absolute pixel change. Using this reward, an additional policy is trained that shares most of its parameters with the A2C actor and critic. This policy must be trained using an off-policy RL algorithm since it uses the data sampled from the experience replay buffer generated by the actor. In [7] the n-step Q-learning loss [18] is used to update the policy. The observation images are downsized, converted to gray scale, and the absolute difference between two consecutive observations is computed and used as pseudo-rewards for Q-learning [17].

A new head is attached to the output of LSTM. This head consists of deconvolutional layers – upsampling the low-dimensional features back to the size of the downsampled observations. For each action, there is a different output in the last layer to output the Q-function for each pixel. The dueling DQN technique [22] is used to improve the performance of the pixel control network. The pixel control network used in our method can be seen in Fig. 3.

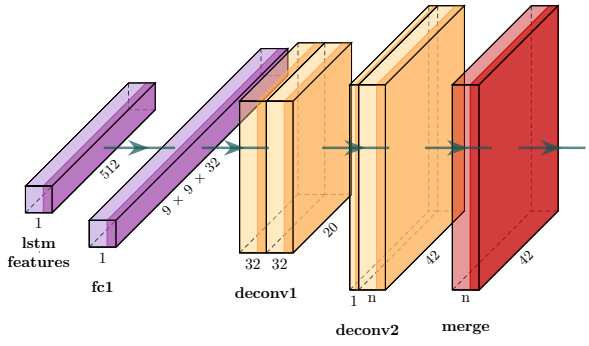


Fig. 3: Pixel control network.

D. Additional Auxiliary Tasks for Visual Navigation

Motivated by [23] and [8], we introduced additional auxiliary tasks that are specific to visual navigation. They were designed to enhance the training process as well as to help the network generalize. We train the model to predict the depth-map, image segmentation of the observation and image segmentation of the target. For the image segmentations we map the object-type to the RGB color space and maximize the distances between each color in the HSB color space. The input is passed through a narrow part of the network in autoencoder fashion to improve the quality of features in the shared part of the network. This gives the actor and the

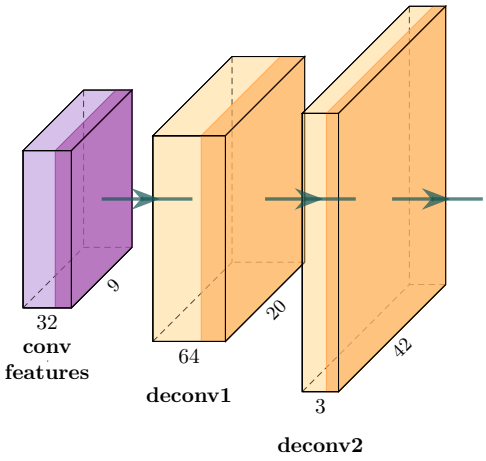


Fig. 4: Visual navigation auxiliary task network – image segmentation and target segmentation prediction.

critic good features in bottom-most layers with a compact representation of all information needed to reconstruct depth-map and image segmentations. These bottom-most layers would otherwise be difficult to train since the network is deep and the loss is noisy due to the imprecise target values computed using the RL algorithm. The image segmentation for the target ensures the network pays attention to what the target is. Otherwise it would be difficult for the network to take the target input into account.

For each visual navigation auxiliary task there is a network attached to the last convolutional layer consisting of deconvolutional layers. The network architecture for the image segmentation and the target image segmentation can be seen in Fig. 4. For the depth-map prediction the structure of the network is the same but the intermediate deconvolutional layer has only 32 filters and the last layer has a single channel. True features (the image segmentations for observation and the target and the depth map) are downsampled to a smaller size. The MSE is computed between the outputs of the networks and the true features.

The additional auxiliary tasks for visual navigation also allow for the use of supervised learning to initialize the network with good features in the bottom-most part of the network since these are the least dependent on the policy. It is costly to render a 3D scene, but it is cheap to pre-compute a data set of observations taken from the scene and use it for supervised training.

E. Environment Complexity

The training of the agent might be hard especially when the environment is large and the initial state is far from the target. To make the task easier for the agent we first sample the initial states closer to the target and gradually increase the distance between the initial state and the target. Let $\tau \in [0, 1]$ be the environment complexity. We define the maximal sampling distance $\mathrm{dmax}_E \colon [0, 1] \to \mathbb{R}$ of an environment E

as follows:

$$\mathrm{dmax}_E(\tau) = \tau \max_{s_1,s_2}\{\mathrm{dist}(s_1,s_2)\}\,, \tag{8}$$

where $\mathrm{dist}(\cdot,\cdot)$ measures the distance between any two states of the given environment E. Any distance measure can be used, e.g., the Euclidean distance between the corresponding agent positions in the environment.

The initial state s_0 is sampled from the uniform probability distribution over the set of possible initial states closer to any target than $\mathrm{dmax}_E(\tau)$:

$$\mathbb{U}(\{s_1 | s_1 \in \mathcal{S}_{start}, s_2 \in \mathcal{S}_{target}, \mathrm{dist}(s_1,s_2) \leq \mathrm{dmax}_E(\tau)\})\,, \tag{9}$$

where the set of target states is denoted by $\mathcal{S}_{target}$. The environment complexity τ starts at a low value, e.g. 0.3, and gradually increases during the training to 1.0.

IV. Experiments

We have experimentally evaluated the performance of our method A2CAT-VN, using the average episode length and the average episode undiscounted return as performance metrics. The averages are computed Monte Carlo estimates based on 100 rollouts. The randomness comes from the initial state, the non-deterministic behavior of the environment and the stochasticity of the actor.

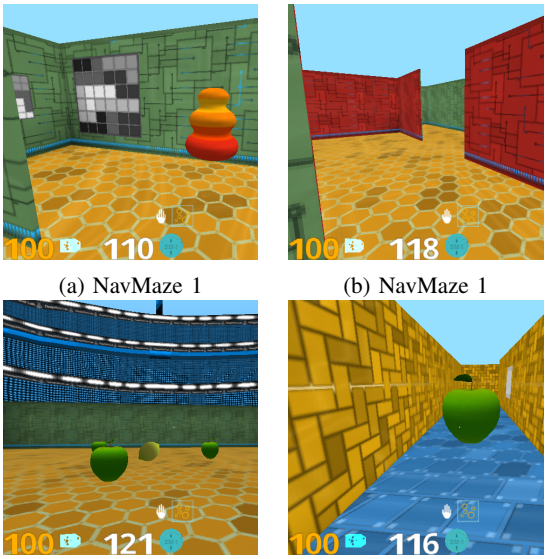
A. Environments

We have employed three different 3D environment simulators suitable for visual navigation tasks.

1) DeepMind Lab [9] is a 3D framework which allows an agent to move and collect objects in synthetic environments. It is fast and highly optimized for training AI agents and the set of allowed actions is customizable. Fig. 5 shows examples of images from this environment. We used it to compare the proposed algorithm with alternatives from the literature and to pre-train the agent's network for other environments, which sped up the training process.

2) AI2-THOR [10] is a photo-realistic interactive framework with high-quality indoor images (see Fig. 6). Most of the environments are a single room and are dynamic, i.e., at the beginning of the episode, various objects can be placed at random positions. The agent moves on a grid: an action moves the agent to a neighboring point on the grid or rotates the agent by 90 °. This does not allow for a good generalization since the agent can memorize the finite (and small) number of observations it can possibly receive. Therefore, we have modified the implementation of the AI2-THOR 3D simulator to use continuous space. We have extended the set of possible actions by adding a rotation by an arbitrary angle and a movement by an arbitrary distance. We have also changed the way collisions are handled – when an action would lead to a collision, instead of leaving the agent at the original location, we execute a part of the action until the collision occurs.

3) House3D with SUNCG [5] is a 3D framework allowing to use the environments from the SUNCG dataset [6]. The



(a) NavMaze 1 (b) NavMaze 1

(c) SeekAvoid (d) NavMaze 2

Fig. 5: Sample images from DeepMind Lab framework [9].

(a) bedroom (b) bathroom

(c) kitchen (d) living room

Fig. 6: Sample images from AI2-THOR framework [10].

SUNCG data set consists of over 45 000 indoor environments, most of them being two-storey houses and studios. House3D is highly optimized for AI agents training and runs fast on GPUs. Apart from RGB output rendering, it also supports depth map and image segmentation rendering. Illustrative images from this environment are shown in Fig. 7. The set of actions can be customized in a similar way as in the DeepMind Lab environment.

B. Action Space

In each of our experiments we used actions from the following set: forward, backward, left, right, rotate-left, rotate-right, tilt-up, tilt-down. The forward and backward actions move the agent in the direction it is currently facing. The left and right actions move the agent in perpendicular directions to the direction it is facing. The rotate-left and rotate-right

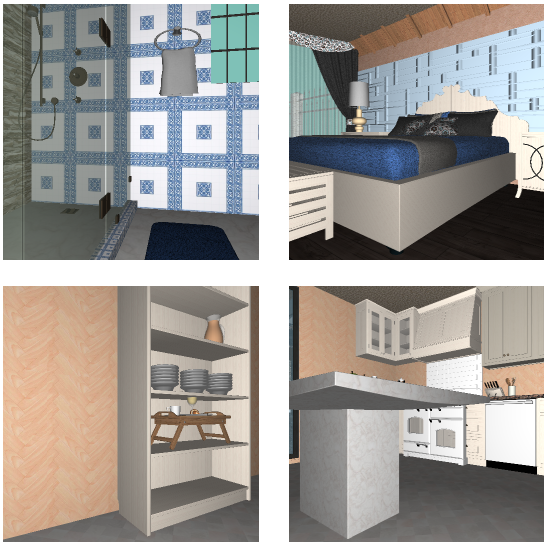


Fig. 7: SUNCG [6] scene images from the House3D [5] framework.

actions rotate the agent by 30 degrees[6] counter-clockwise and clockwise respectively and the tilt-up and tilt-down actions tilt the agent's camera up or down by 30 degrees.

In real-world environments, the actuators would rarely be able to move the agent precisely. To simulate such a setting, Gaussian noise is added to the position and rotation of the agent after taking an action. More specifically, let $\tilde{s} = (\tilde{x_1}, \tilde{x_2}, \tilde{\theta}, \sigma)$ be the agent's position, horizontal rotation, and tilt of the camera after taking an action before we added the noise. Then the agent's final position and rotation is $s = (x_1, x_2, \theta, \sigma)$, with $x_1 \sim \mathcal{N}(\tilde{x_1}, 0.02^2)$, $x_2 \sim \mathcal{N}(\tilde{x_2}, 0.02^2)$ and $\theta \sim \mathcal{N}(\tilde{\theta}, 2^2)$.

C. Training

The reward can be assigned to the agent using different schemes. In our work we give the agent reward one if it reaches the target and zero otherwise. In the training phase, we compute the total gradient as the weighted sum of all the partial gradients: the actor, the critic, the entropy loss, the off-policy critic and the auxiliary tasks. The gradient is clipped so its $l2$-norm does not exceed 0.5 and the RMSprop optimizer is used to optimize the weights. In all experiments we used two Tesla K40 GPUs (10GB each) – one GPU was dedicated for the environments and the other for the agent. The parameters used in our method are given in Table I, where f denotes the number of frames processed so far and $f_{\max}$ is the maximum number of frames to be processed during training. Some parameters were chosen to be the same as in [7], [11], others were chosen experimentally.

D. Partial Observability

We compared two different approaches to resolve the partial observability problem. One approach used by [3], [18] concatenates the past four frames as the input to the agent.

[6]One experiment uses 90 ° angles.

TABLE I: Method parameters

name	value
discount factor (γ)	0.99
maximum episode length	900
maximum rollout length	20
maximum number of frames ($f_{\max}$)	$4 \cdot 10^7$
number of environment instances	16
replay buffer size	2 000
optimizer	RMSprop
RMSprop alpha	0.99
RMSprop epsilon	10^{-5}
learning rate	$7 \cdot 10^{-4} \frac{f_{\max} - f}{f_{\max}}$
max. gradient norm	0.5
entropy gradient weight	0.001
actor weight	1.0
critic weight	0.5
off-policy critic weight	1.0
pixel control weight	0.05
reward prediction weight	1.0
depth-map prediction weight	0.1
image segmentation prediction weight	0.1
target segmentation prediction weight	0.1
pixel control discount factor	0.9
pixel control downsize factor	4
auxiliary VN downsize factor	4
pre-training optimizer	Adam
pre-training total epochs	30
pre-training dataset size	$2 \cdot 10^5$

The other approach [7] uses the LSTM network [20]. We tested both methods on the DeepMind Lab environment because of its great speed and relative simplicity. The allowed actions were forward, backward, left, right, rotate-left, rotate-right. We did not use any noise and the distance by which the actions moved the agent were 0.3 meters for actions forward, backward, left, right. The input to the agent was a single RGB image with the resolution of 84×84 pixels. The network structure based on [7] was similar in both cases except in the frame concatenation version, where the LSTM was replaced by a linear layer. Both networks used the UNREAL auxiliary tasks [7]. The algorithms were trained on the DeepMind Lab SeekAvoid environment. The results can be seen in Fig. 8. The experiment clearly shows that LSTM outperforms the frame concatenation method.

E. AI2-THOR

We have trained our algorithm on four environments from AI2-THOR environment with multiple targets. We have used the same set of actions as [3] – rotate-left, rotate-right, forward, and backward. Actions forward and backward moved the agent in the direction is was facing by either 0.33 m or -0.33 m. Actions rotate-left and roatate-right rotated the agent by $\pm 90°$. No noise was applied. This allowed us to compare our method to [3] and also to cache the observations since it turned the problem into an instance of a grid world. The resolution of the input images was 174×174 pixels. We used 16 environments in parallel for our algorithm each using different scene or different target. We did not use any pre-training nor did we increase the environment complexity. Our method is compared to [3]. We used their own code for

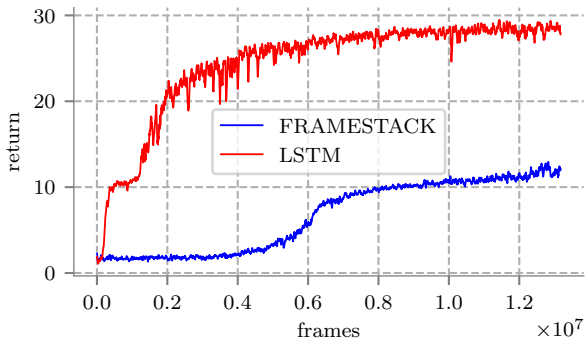


Fig. 8: Comparison of LSTM model (LSTM) with frame-concatenation model (FRAMESTACK) trained using deterministic UNREAL. Plot shows average return curves during training on DeepMind Lab [9] environment called SeekAvoid.

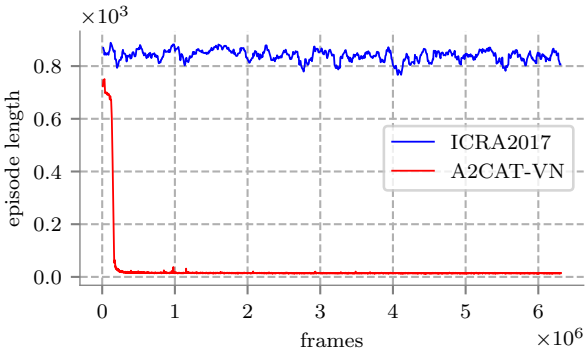


Fig. 9: Comparison of our model (A2CAT-VN) with target driven visual navigation paper [3] (ICRA2017) trained on four environments from AI2-THOR. Plot shows the average episode length curves during training.

fair comparison, however, the environments we chose for this experiment were bigger and more difficult to navigate than those used in [3]. The training of our algorithm took roughly 1 day, but we had to stop the training of [3] after three days with unsatisfactory results – their implementation did not allow to speed up the training by using GPUs. The results can be seen in Fig. 9. Our method A2CAT-VN reached the optimal solution after approximatelly $5 \cdot 10^5$ frames, whereas the method described in [3] was not able to move closer to the optimal solution after $7 \cdot 10^6$ frames.

F. Continuous AI2-THOR

We trained our agent on our modified version of the AI2-THOR environment. We used the full set of actions as described in Section IV-B: forward, backward, left, right, rotate-left, rotate-right, tilt-up, tilt-down. The forward and backward actions moved the agent by 0.5 and -0.2 meters respectively and the left and right actions moved the agent by 0.35 meters. Due to performance issues, however, the noise

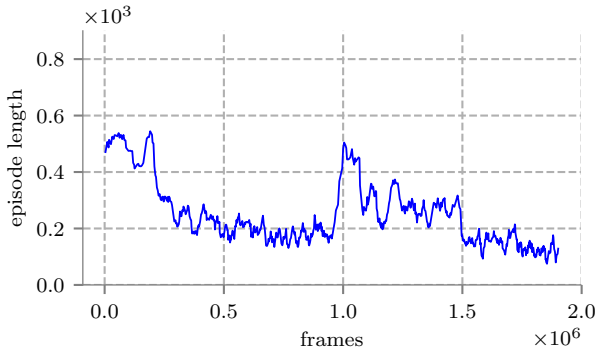


Fig. 10: This plot shows the average episode length curve while training our algorithm A2CAT-VN in a single environment from AI2-THOR framework [10] with multiple randomly placed targets.

was only applied in the direction of the movement and no noise was applied in case of tilt-up and tilt-down actions. The agent was trained on a single bedroom scene with multiple targets specified by images. We used 16 environments in parallel, each having a different target. The target object was placed randomly to different positions in the environments and the agent was trained to get to close proximity of the target object (1 meter). The resolution of the input images was 174×174 pixels. We did not use pre-training nor did we increase the environment complexity. The results can be seen in Fig. 10. The training took 4 days. The AI2-THOR 3D environment simulator was too slow for further experiments. The results show the ability of the agent to navigate in non-static environments and find dynamically placed objects.

G. Auxiliary Tasks

We compared our method (A2CAT-VN) with the batched A2C extended with the original two UNREAL auxiliary tasks. Single agent was trained on 16 houses from the SUNCG dataset [6] using House3D environment simulator. We used the same actions as those described in Section IV-F except for the tilt-up and tilt-down actions. Inspired by [5], the agent was trained to find a selected room in the house. The room was given to the agent in the form of an observation taken in a room of the same type. For example, if the target room is the bedroom, the agent is supposed to find any bedroom. The resolution of the input images was 174×174 pixels. We pre-trained our neural network using the data collected from a subset of all houses from SUNCG dataset. The number of images we used for pre-training was approximatelly $20\,000$ and we trained our network for 30 epochs using the Adam optimizer. The pre-training took roughly one hour. For the full training, we linearly increased the environment complexity from 0.3 at the time step 5M to 1.0 a the time step 10M. The training took rougly two days. The training curves for the average episode length can be seen in Fig. 11. Our algorithm A2CAT-VN converged much faster with additional auxiliary tasks for visual navigation

enabled, reaching the average episode length 200 in $\approx 3 \cdot 10^6$ frames whereas without the additional tasks the training took $\approx 8 \cdot 10^6$ frames to get to the same level.

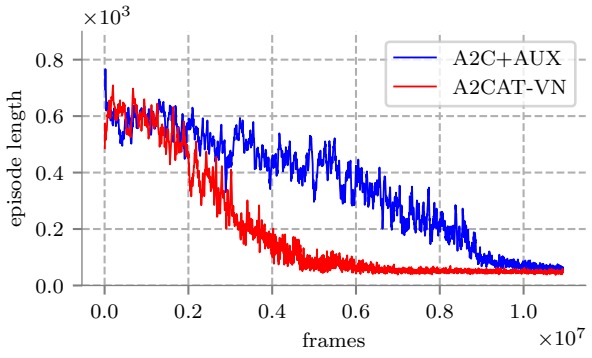


Fig. 11: Comparison of our method (A2CAT-VN) with the A2C algorithm extended with UNREAL auxiliary tasks [7]. The training was performed on a set of 16 environments from SUNCG dataset [6]. The plot shows the average episode length curve.

V. Conclusions & Future Work

We have proposed a novel learning architecture A2CAT-VN for visual navigation in indoor environments. It is based on a compact deep network capable of fast learning over multiple realistic environments, using the batched A2C algorithm extended with novel auxiliary tasks. By using the target image as an input, our method enables the agent to locate arbitrary goals, as long as their images have been used during the training phase.

The method was demonstrated on AI2-THOR and House3D environments. First we have shown that the basic batched A2C algorithm benefits from the addition of the UNREAL auxiliary tasks [7]. Further performance gain was achieved by employing additional auxiliary tasks designed specifically for visual navigation.

When applied to AI2-THOR environment, our method was able to converge at least an order of magnitude faster than an alternative state-of-the-art method [3], which also allows for using multiple targets and was demonstrated in indoor environments, similarly to our method. The auxiliary tasks introduced were shown to reduce the number of frames needed to train the agent by the factor of two and they allowed to use supervised learning to pre-train a part of the network.

Future research should investigate the potential effects of using supervised pre-training of additional auxiliary tasks for visual navigation on the training performance. We would also like to explore the application of our method to more 3D environments (perhaps outdoor environments) and potentially apply it to real-world environments. Another line of research needs to be conducted on the ability of the method to generalize to unseen targets. In addition, we believe the ability of the agent to deal with unseen environments might outline an important area for future research.

References

[1] R. S. Sutton and A. G. Barto, *Introduction to Reinforcement Learning*, 2nd ed. Cambridge, MA, USA: MIT Press, 2017.

[2] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778, 2016.

[3] Y. Zhu, R. Mottaghi, E. Kolve, J. J. Lim, A. Gupta, L. Fei-Fei, and A. Farhadi, "Target-driven visual navigation in indoor scenes using deep reinforcement learning," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, May 2017, pp. 3357–3364.

[4] J. Bruce, N. Suenderhauf, P. Mirowski, R. Hadsell, and M. Milford, "One-shot reinforcement learning for robot navigation with interactive replay," 2017.

[5] Y. Wu, Y. Wu, G. Gkioxari, and Y. Tian, "Building generalizable agents with a realistic and rich 3d environment," 2018.

[6] S. Song, F. Yu, A. Zeng, A. X. Chang, M. Savva, and T. Funkhouser, "Semantic scene completion from a single depth image," *Proceedings of 29th IEEE Conference on Computer Vision and Pattern Recognition*, 2017.

[7] M. Jaderberg, V. Mnih, W. M. Czarnecki, T. Schaul, J. Z. Leibo, D. Silver, and K. Kavukcuoglu, "Reinforcement learning with unsupervised auxiliary tasks," 2016.

[8] P. Mirowski, R. Pascanu, F. Viola, H. Soyer, A. J. Ballard, A. Banino, M. Denil, R. Goroshin, L. Sifre, K. Kavukcuoglu, D. Kumaran, and R. Hadsell, "Learning to navigate in complex environments," 2016.

[9] C. Beattie, J. Z. Leibo, D. Teplyashin, T. Ward, M. Wainwright, H. Küttler, A. Lefrancq, S. Green, V. Valdés, A. Sadik, J. Schrittwieser, K. Anderson, S. York, M. Cant, A. Cain, A. Bolton, S. Gaffney, H. King, D. Hassabis, S. Legg, and S. Petersen, "Deepmind lab," 2016.

[10] E. Kolve, R. Mottaghi, D. Gordon, Y. Zhu, A. Gupta, and A. Farhadi, "AI2-THOR: An Interactive 3D Environment for Visual AI," *arXiv*, Dec 2017.

[11] Y. Wu, E. Mansimov, S. Liao, R. B. Grosse, and J. Ba, "Scalable trust-region method for deep reinforcement learning using kronecker-factored approximation," *CoRR*, vol. abs/1708.05144, 2017.

[12] I. Grondman, L. Buşoniu, G. Lopes, and R. Babuška, "A survey of actor-critic reinforcement learning: Standard and natural policy gradients," *IEEE Transactions on Systems, Man, and Cybernetics. Part C: Applications and Reviews*, vol. 42, no. 6, pp. 1291–1307, 2012.

[13] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Machine Learning*, vol. 8, no. 3, pp. 229–256, May 1992. [Online]. Available: https://doi.org/10.1007/BF00992696

[14] R. S. Sutton, D. A. McAllester, S. P. Singh, and Y. Mansour, "Policy gradient methods for reinforcement learning with function approximation," in *Advances in neural information processing systems*, 2000, pp. 1057–1063.

[15] C. J. C. H. Watkins and P. Dayan, "Q-learning," in *Machine Learning*, 1992, pp. 279–292.

[16] L. Gurvits, L. Lin, and S. Hanson, "Incremental learning of evaluation functions for absorbing markov chains: New methods and theorems," *preprint*, 1994.

[17] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *International conference on machine learning*, 2016, pp. 1928–1937.

[18] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," 2013.

[19] J. T. Springenberg, A. Dosovitskiy, T. Brox, and M. Riedmiller, "Striving for simplicity: The all convolutional net," 2014.

[20] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997. [Online]. Available: https://doi.org/10.1162/neco.1997.9.8.1735

[21] J. Munk, J. Kober, and R. Babuska, "Learning state representation for deep actor-critic control," 12 2016.

[22] Z. Wang, T. Schaul, M. Hessel, H. van Hasselt, M. Lanctot, and N. de Freitas, "Dueling network architectures for deep reinforcement learning," 2015.

[23] S. Lange and M. Riedmiller, "Deep auto-encoder neural networks in reinforcement learning," in *The 2010 International Joint Conference on Neural Networks (IJCNN)*, July 2010, pp. 1–8.

Appendix B

Attached CD Contents

The attached CD includes the following content:

- the content of our method's GitHub repository frozen at the time of handing in the thesis. The repository is also published on `https://github.com/jkulhanek/a2cat-vn-pytorch`.
- the content of the general deep RL repository created as a framework for expriments described in this thesis frozen at the time of handing in the thesis. The repository is also published on `https://github.com/jkulhanek/deep-rl-pytorch`.
- the thesis text in PDF format.
- the submitted conference paper in PDF format.
- the modified version of AI2-THOR environment.