Linköping University | Department of Computer and Information Science
Master’s thesis, 30 ECTS | Datateknik
YYYY | LIU-IDA/LITH-EX-A--YYYY/001--SE

Learning to Search for Targets

– A Deep Reinforcement Learning Approach for Unknown Environments

---

Inlärd sökning efter mål

Oskar Lundin

Supervisor : Sourabh Balgi
Examiner : Jose M. Peña

External supervisor : Fredrik Bissmarck

Linköpings universitet
SE–581 83 Linköping
+46 13 28 10 00 , www.liu.se

Abstract

The abstract resides in file `Abstract.tex`. Here you should write a short summary of your work.

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Pellentesque in massa suscipit, congue massa in, pharetra lacus. Donec nec felis tempor, suscipit metus molestie, consectetur orci. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Curabitur fermentum, augue non ullamcorper tempus, ex urna suscipit lorem, eu consectetur ligula orci quis ex. Phasellus imperdiet dolor at luctus tempor. Curabitur nisi enim, porta ut gravida nec, feugiat fermentum purus. Donec hendrerit justo metus. In ultrices malesuada erat id scelerisque. Sed sapien nisi, feugiat in ligula vitae, condimentum accumsan nisi. Nunc sit amet est leo. Quisque hendrerit, libero ut viverra aliquet, neque mi vestibulum mauris, a tincidunt nulla lacus vitae nunc. Cras eros ex, tincidunt ac porta et, vulputate ut lectus. Curabitur ultricies faucibus turpis, ac placerat sem sollicitudin at. Ut libero odio, eleifend in urna non, varius imperdiet diam. Aenean lacinia dapibus mauris. Sed posuere imperdiet ipsum a fermentum.

Nulla lobortis enim ac magna rhoncus, nec condimentum erat aliquam. Nullam laoreet interdum lacus, ac rutrum eros dictum vel. Cras lobortis egestas lectus, id varius turpis rhoncus et. Nam vitae auctor ligula, et fermentum turpis. Morbi neque tellus, dignissim a cursus sed, tempus eu sapien. Morbi volutpat convallis mauris, a euismod dui egestas sit amet. Nullam a volutpat mauris. Fusce sed ipsum lectus. In feugiat, velit eu fermentum efficitur, mi ex eleifend ante, eget scelerisque sem turpis nec augue.

Vestibulum posuere nibh ut iaculis semper. Ut diam justo, interdum quis felis ac, posuere fermentum ex. Fusce tincidunt vel nunc non semper. Sed ultrices suscipit dui, vel lacinia lorem euismod quis. Etiam pellentesque vitae sem eu bibendum. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Pellentesque scelerisque congue ullamcorper. Sed vehicula sodales velit a scelerisque. Pellentesque dignissim lectus ipsum, quis consectetur tellus rhoncus a.

Nunc placerat ut lectus vel ornare. Sed nec dictum enim. Donec imperdiet, ipsum ut facilisis blandit, lacus nisi maximus ex, sed semper nisl metus eget leo. Nunc efficitur risus ac risus placerat, vel ullamcorper felis interdum. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos himenaeos. Duis vitae felis vel nibh sodales fringilla. Donec semper eleifend sem quis ornare. Proin et leo ut dolor consectetur vehicula. Lorem ipsum dolor sit amet, consectetur adipiscing elit.

Nunc dignissim interdum orci, sit amet pretium nibh consectetur sagittis. Aenean a eros id risus aliquam placerat nec ut lectus. Curabitur at quam in nisi sodales imperdiet in at erat. Praesent euismod pulvinar imperdiet. Nam auctor mattis nisi in efficitur. Quisque non cursus ipsum, consequat vehicula justo. Fusce varius metus et nulla rutrum scelerisque. Praesent molestie elementum nulla a consequat. In at facilisis nisi, convallis molestie sapien. Cras id ullamcorper purus. Sed at lectus sit amet dolor finibus suscipit vel et purus. Sed odio ipsum, dictum vel justo sit amet, interdum dictum justo. Quisque euismod quam magna, at dignissim eros varius in. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas.

Acknowledgments

Acknowledgments.tex

Contents

List of Figures

Notation

x	variable
X	random variable
$\vec{x}$	vector
$\mathbb{X}$	set

1 Introduction

In this thesis project, the problem of searching for targets in unknown but familiar environments is addressed. This chapter presents the motivation behind the project, the research questions that are addressed, and the delimitations.

1.1 Motivation

The ability to visually search for things in an environment is fundamental to intelligent behaviour. We humans are constantly looking for things, be it be it the right book in the bookshelf, a certain keyword in an article or blueberries in the forest. In many cases, it is important that this search is strategic, efficient, and fast. Animals need to quickly identify predators, and drivers need to be able to search for pedestrians crossing the road they are driving on.

An intelligent searcher should be able to

Automating the task of searching is of great interest

While searching for targets is often seemingly effortless to humans, it is a complex process. How humans and animals search for things has been extensively studied in neuroscience and neurobiology [14, 44, 43].

Applications such as helping robots and search and rescue mean that it is of great interest to automate visual search. In the computer vision field, there has been several attempts to mimic the way humans search in machines []. Most attempts focus on fully observable scenes where the target is in view and the task is to localize it (object localization). However, in many real-world visual search scenarios the field-of-view is limited. This means that the search process is split into two steps: directing the field of view (covert attention), and locating targets within the view (overt attention). Much work has been focused on latter, locating targets within the field of view [].

When only a fraction of the environment is visible, where to move the field of view becomes an important decision. The characteristics of the searched environment can often be used to find targets quicker. For example, if one is foraging for blueberries it makes sense to search the ground rather than the trees. Similarly, if one is searching a satellite image for boats it is reasonable to focus on ocean shores. If you see a railroad track or the wake of a boat you can usually follow it to find a vehicle. The exact characteristics of the environment need not be constant - forests with blueberries can vary greatly in appearance and boats can be found in all of the seven seas. In many cases, the environment is familiar in that it has characteristics that are similar to previously seen environments. Humans are able to generalize in such cases.

Manually creating search algorithms for such tasks is problematic. The appearance and distribution of targets in an environment varies greatly, and may be subtle. The visual richness of the environment itself is another problem. How can you identify useful hints from the environment to guide covert attention? Doing so manually can be labour intensive, especially if a searcing system should be deployed in many different environments. If one could instead learn the underlying from a limited set of sample environments and generalize to unseen similar environments this problem would be circumvented.

Deep reinforcement learning is an approach for how to act. It has been applied to a number of problems with sucess...

1.2 Aim

The aim of this thesis is to investigate how an intelligent agent that learns to search for targets can be implemented with deep reinforcement learning. Such an agent should learn the characteristics of the environments it is trained on and utilize this knowledge to effectively search for targets in unseen environments. Specifically, we consider scenarious where the agent can only observe a small portion of its environment at any given time. The agent has to actively choose where to look in order to gain new information about the environment.

We are looking for the following properties:

- The agent should prioritize regions where the probability of finding a target is high according to previous experience.
- The agent should utilize information from previously visited regions to decide where to move.
- The agent should be able to find multiple targets while minimizing its path length.
- The agent should avoid looking at the same region twice.

Our contributions are as follows:

- ...

1.3 Research questions

This thesis will address the following questions:

1. How can a learning agent that learns to intelligently search for targets be implemented?
2. How does the learning agent compare to random walk, exhaustive search, a human searcher?
3. How well does the learning agent generalize to unseen but familiar environments?

1.4 Delimitations

This thesis will be focused on the behavioral aspects of the presented problem. We do not focus on difficult detection problems, but rather efficient actions. For this reason, targets will deliberately be made easy to detect. For simplicity, we make the assumption that the searched environment is static. The appearance of the environment and the location of the targets does not change from one observation to the next.

2 Theory

This chapter introduces relevant theory.

2.1 Background

2.1.1 Active Vision

Much of past and present research in machine perception involves a passive observer. Images are passively sampled and perceived. Animal perception, however, is active. We do not only see things, but look for them. One might ask why this is the case, if there is any advantage that an active observer has over a passive one. Aloimonos and Weiss (1988) [2] introduce the paradigm called *active vision*, and prove that an active observer can solve several basic vision problems in a more efficient way than a passive one.

Bajcsy (1988) [**bajcsy_1988**] defines active vision, and perception in general, as a problem of intelligent data acquisition. An active observer needs to define and measure parameters and errors from its scene and feed then back to control the data acquisition process. Bajcsy states that one of the difficulties of this problem is that they are scene and context dependent. A thorough understanding of the data acquisition parameters and the goal of the visual processing is needed. One view lacks information that may be present with multiple views. Multiple views also add the time dimension into the problem.

In a re-visitation of active perception, Bajcsy, Aloimonos and Tsotsos (2018) [**bajcsy_aloimonos_tsotsos_2018**] stress that despite recent successes in robotics, artificial intelligence and computer vision, an intelligent agent must include active perception:

> An agent is an active perceiver if it knows why it wishes to sense, and then chooses what to perceive, and determines how, when and where to achieve that perception

[**bajcsy_aloimonos_tsotsos_2018**]

2.1.2 Visual Search

The perceptual task of searching for something in a visual environment is usually referred to as *visual search*. The searched object or feature is the *target*, and the other objects or features in the environment are the *distractors*. This task has been studied extensively in psychology and neuroscience.

Wolfe (2021) [43] describes a model of visual search

Eckstein (2011) [14] reviews efforts from various subfields and identifies a set of mechanisms used to achieve efficient visual search. Knowledge about the target, distractor, background statistical properties, location probabilities, contextual cues, rewards and target prevalence are all identified as useful. This is motivated with evidence from psychology as well as neural correlates.

Visual search is not always instant, and can in fact often be slow. This is in part due to processing: our visual system cannot process the entire visual field and

Wolfe and Horowitz (2017) [**wolfe_horowitz_2017**] identify and measure a set of factors that guide attention in visual search. One of these is bottom-up guidance, in which some visual properties of the scene draw more attention than others. Another is top-down guidance, which is user driven and directed to objects with known features of desired targets. Scene guidance is also identified, in which attributes of the scene guide attention to areas likely to contain targets.

These works ground the task considered in this project in psychology.

2.1.3 Reinforcement Learning

Reinforcement learning (RL) [40] is a subfield of machine learning concerned with learning from interaction how to achieve a goal. This section introduces the fundamental concepts of RL.

2.1.3.1 Partially Observable Markov Decision Processes

The problem of learning from interaction to achieve some goal is often framed as a Markov decision process (MDP). A learning *agent* interacts continually with its *environment*. The agent takes the *state* of the environment as input, and select an *action* to take. This action updates the state of the environment and gives the agent a scalar *reward*. It is assumed that the next state and reward depend only on the previous state and the action taken. This is referred to as the *Markov* property. [23]

In an MDP, the agent can perceive the state of the environment with full certainty. For many problems, including the one we consider here, this is not the case. The agent can only perceive a partial representation of the environment's state. Such a process is referred to as a partially observable Markov decision process (POMDP). A POMDP is formally defined as a 7-tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \Omega, \mathcal{O}, \gamma \rangle$, where

- $\mathcal{S}$ is a finite set of states,
- $\mathcal{A}$ is a finite set of actions,
- $\mathcal{T} : \mathcal{S} \times \mathcal{A} \to \Pi(\mathcal{S})$ is a state-transition function,
- $\mathcal{R} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is a reward function,
- Ω is a finite set of observations,
- $\mathcal{O} : \mathcal{S} \times \mathcal{A} \to \Pi(\Omega)$ is an observation function, and
- $\gamma \in [0, 1]$ is a discount factor.

Assume that the environment is in state $s_t \in \mathcal{S}$, and the agent selects action $a_t \in \mathcal{A}$. Then, $T(s_t, a_t, s_{t+1})$ is the probability of ending in state s_{t+1} and $r_t = R(s_t, a_t)$ is the expected reward gained by the agent. The agent also receives an observation $o_t \in \Omega$ with probability $\mathcal{O}(s_{t+1}, a_t, o_t)$. [23] Figure 2.1.3.1 illustrates the interaction between agent and environment.

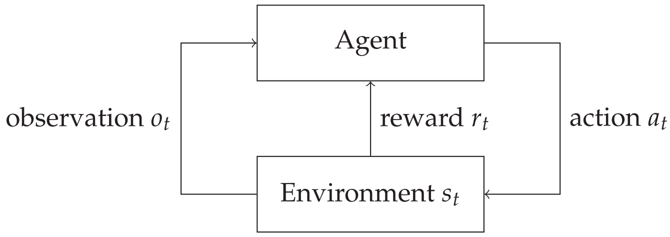

Figure 2.1: Partially observable Markov decision process.

The agent and environment interact over a sequence of discrete time steps $t = 0, 1, \dots, T$, giving rise to an *episode* of length T. At each time step t, the goal of the agent is to select the action that maximizes the expected *discounted return*:

$$\mathbb{E}\left[\sum_{k=0}^{T} \gamma^{k-t-1} r_k\right]$$

Since the agent receives partial observations of the environment's state, it has to act under uncertainty. Planning in a POMDP is undecidable, and solving them is often computationally intractable. Approximate solutions are more common, where the agent usually maintains an internal *belief state* [23] which is acts on. The belief state summarizes the agent's previous experience and is therefore dependent on the previous belief state, observation and action. It does not need to summarize the whole history, but generally only the information that helps the agent maximize the expected reward. From here on we will use the belief state and the environment state s interchangably.

2.1.3.2 Policies and Value Functions

The behaviour of the agent is described by its *policy*. A policy π is a mapping from perceived environment states to actions. Policies are often stochastic and specify probabilities for each action, with $\pi(a|s)$ denoting the probability of taking action a in state s. [40]

Most RL solutions methods also approximate a *value function*. A value function v_π estimates how good it is to be in a state. The value function $v_\pi(s)$ is the expected (discounted) return when starting at state s and following policy π until the end of the episode. There are two common alternative value functions: The *quality function* $q_\pi(s, a)$ gives the value of state s under policy π where a is the first action taken. Given a quality function, *action-value* methods choose the action greedily at every state as $arg\, max q_\pi(s, a)$. The *advantage function* $a_\pi(s, a)$ instead represents the relative advantage of actions, $a_\pi = q_\pi - v_\pi$. [40]

For problems with large state and action spaces, it is common to represent value functions with *function approximation*. In such cases, it is common to encounter states that have never been encountered before. This makes it important that the estimated value function can generalize from seen to unseen states. With examples from the true value function, an approximation can be made with supervised learning methods. We write $\hat{v}(s, w) \approx v_\pi(s)$ for the approximate value of state s with the some weight vector $w \in \mathbb{R}^d$. [40]

An alternative to action-value methods is to approximate the policy itself. *Policy gradient methods* [39] learn a parametrized policy that select actions without a value function. We denote a parametrized policy as $\pi(a|s, \theta)$ with $\theta \in \mathbb{R}^{d'}$ as the parameters to the policy. The policy parameters are usually learned based on the gradient of some performance measure $J(\theta)$. As long as $\pi(a|s, \theta)$ is differentiable with respect to its parameters, the parameters can be updated with *gradient ascent* in J:

$$\theta_{t+1} = \theta_t + \alpha \nabla \hat{J(\theta_t)})$$

Advantages of policy parametrization over action-value methods include stronger convergence guarantees [39] and more flexibility in parametrization [40]. In practice, value functions are often still used to learn the policy parameter, but they are not needed for action selection. Such methods are called *actor-critic* methods, with actor referring to the learned policy and critic referring to the learned value function. In these cases, there might also be some overlap between the weights w of the value function estimate and θ of the policy estimate.

2.1.3.3 Challenges in Reinforcement Learning

- Exploration and Exploitation
- Sparse Rewards
- Credit Assignment [27]

For some tasks, like certain video games, the objective is simply to maximize the score obtained. In this case there is an inherent reward signal and the agent achieves its task simply by maximizing this inherent signal. Other times, we have a task we want the agent to solve and have to design a reward signal around that task. Designing rewards is not straightforward and can often have unintended effects [40]. Special care has to be taken to ensure that the reward incentivizes the desired behaviour.

Here, the problem of sparse rewards also comes into play.

Reward shaping is a technique to tackle... [26]

2.1.4 Deep Learning

Deep learning is a family of techniques in which hypothesis are represented as computation graphs with tunable weights. The computation graphs are inspired by biological neurons in the brain and are referred to as *neural networks*. Deep neural networks consist of *nodes* arranged in *layers*: one input layer, zero or more hidden layers and one output layer. Each layer receives an input *representation* [6] from the previous layer and outputs a transformed representation to the next layer. Given some input, a neural network optimizes its output representation with regard to some objective. Usually, a loss function $\mathcal{L}$ is minimized by updating the weights $\mathbf{w}$ of the network with some variant of *gradient descent* with learning rate α:

$$\mathbf{w} \leftarrow \mathbf{w} - \alpha \nabla_{\mathbf{w}} \mathcal{L}(\mathbf{w}) \tag{2.1}$$

The only requirement on the functions computed by each node is that it is differentiable. As long as this holds, layers can be stacked arbitrarily and the gradients can be computed with the chain rule. This way, errors in the output can be passed back through the network (*back-propagation*) and used to update the weights. [37, 16]

The architecture of a neural network imposes some bias onto the learning that its expected to be useful for generalizing to unseen samples. We now describe three neural network architectures that will be used in this work.

2.1.4.1 Feedforward Neural Network

A feedforward neural network, also known as a multi-layer perceptron (MLP) [16], only has connections in one direction. Each node in the network receives inputs from its predecessors and outputs the result of a function of those inputs. The output y of each node is usually computed by taking the weighted sum of its inputs x and applying some non-linear function

$$y_j = g_j(\mathbf{w}_j^T \mathbf{x}), \tag{2.2}$$

where y_j is the output of node j, g_j is a non-linear *activation function*, $\mathbf{w}_j$ is the vector of weights leading into node j, and $\mathbf{x}$ is the vector of inputs to the node. By convention, each layer also has some *bias* that allows the total weighted input to g_j to be non-zero even when the outputs from the previous layer are zero. The bias is included as an extra input x_0 fixed to 1, and an extra tunable weight $w_{0,j}$. The non-linearity ensures that a network with at least two layers can approximate any continuous function. [37]

2.1.4.2 Convolutional Neural Network

Convolutional Neural Networks (CNNs) contain spatially local connections. They have patterns of weights, called *kernels*, that are replicated across units in each layer. With some input vector $\mathbf{x}$ of size n and a vector kernel $\mathbf{k}$ of size l, the (discrete) convolution operation $\mathbf{z} = \mathbf{x} * \mathbf{k}$ is defined as

$$z_i = \sum_{j=1}^{l} k_j x_{j+1-\frac{l+1}{s}}, \tag{2.3}$$

where s is the *stride*. This operations can be generalized up to more than one dimension, such as 2 dimensions for images and 3 dimensions for volumes. With multiple input channels, kernels are stacked into a *filter*. The outputs of each kernel are then summed over, giving one output channel per filter.

There are several advantages to using CNNs for structured input data where neighboring values are correlated. Kernels are smaller than the input, which means that fewer parameters have to be stored. These *sparse interactions* give CNNs reduced memory requirements, as well as improved statistical and computational efficiency.

Furthermore, the same parameters are also used for more than one function in the CNN. *Parameter sharing* across input locations mean that layers in a CNN have *equivariance* to translation. The output of one kernel is the same regardless of the input location. This property of CNNs is useful for images where similar features may be useful regardless of their location in the input. [16]

2.1.4.3 Recurrent Neural Network

Recurrent neural networks (RNNs) extend feedforward networks by allowing cycles in the computation graph. Each cycle has a delay so that some *hidden state* from the previous computation is used as input to the current computation. A recurrent layer with input $\mathbf{x}_t$, output $\mathbf{y}_t$ and hidden state $\mathbf{z}_t$ is defined by

$$\begin{aligned} \mathbf{z}_t &= f_{\mathbf{w}}(\mathbf{z}_{t-1}, \mathbf{x}_t) \\ \mathbf{y}_t &= g_y(\mathbf{W}_{z,y}, \mathbf{z}_t), \end{aligned} \tag{2.4}$$

where $f_{\mathbf{w}}$ is the update process for the hidden state and g_y is the activation function for the hidden layer. This model can be turned into a feedforward network over a sequence of input vectors $\mathbf{x}_1, \ldots \mathbf{x}_T$ and observed outputs $\mathbf{y}_1, \ldots, \mathbf{y}_T$ by *unrolling* it for T steps. The weights are shared across all time steps. This means that RNNs can operate on inputs of arbitrary lengths. The hidden state is used as a summary of all previous items in the sequence. Thus, RNNs make a Markov assumption. [37]

In practice, conventional RNNs struggle with learning long-term dependencies. During back-propagation, gradients can tend to zero for long sequences, something known as the vanishing gradient problem [16]. An architecture that addresses this issue is long short-term memory (LSTM) [22]. LSTMs include a *memory cell* c in the hidden state that is copied from time step to time step, and three soft *gating units* that govern the information flow in the hidden state update process f. This makes LSTMs particularly useful for learning over long sequences.

2.2 Related Work

In this section, related work is presented.

2.2.1 Deep Reinforcement Learning

As mentioned in Section 2.1.3.2, policies and value functions are often approximated. Neural networks have good properties for function approximation and have been used for RL with success. One early example is TD-Gammon [41], a neural network trained with RL that reached expert Backgammon performance in the 1995.

More recently, the successes of deep learning have bled over into the field of RL. In 2015, Mnih et al. [33] extend [32] and introduce DQN, which combines deep neural networks with RL and gives birth to the field of deep reinforcement learning (deep RL). DQN approximates the quality function q with a CNN, and select actions greedily using only visual input. To incorporate some memory, images from the 4 previous time steps are stacked and used as input to the neural network. The input is fed through three convolutional layers: 32 8x8 filters with stride 4, followed by 64 4x4 filters with stride 2, followed by 64 filters of size 32x32 with stride 1. Between each convolutional layer is a ReLU activation function. Then, there is a hidden fully connected layer with a ReLU activation function. The output layer has one output for each valid action.

The DQN architecture sparked great interest and several modifications. Hausknecht and Stone (2017) [18] investigate the effects of adding recurrency to a DQN in order to tackle POMDPs. They use the same convolutional network as [33], but only use the most recent frame as input and replace the hidden layer with a recurrent LSTM. It is found that the agent is able to integrate information over time and achieves comparable performance to the original DQN agent.

2.2.2 Proximal Policy Optimization

Proximal policy optimization algorithms… [38].

Algorithm 1 Proximal Policy Optimization

for iteration $= 1, 2, \ldots$ **do**
  for actor $= 1, 2, \ldots, N$ **do**
    Run policy $\pi_{\theta_{\text{old}}}$ for T timesteps
    Compute advantage estimates $\hat{A}_1, \ldots \hat{A}_T$
  end for
  Optimize surrogate L wrt θ, with K epochs and minibatch size $M \leqslant NT$
  $\theta_{\text{old}} \leftarrow \theta$
end for

2.2.3 Object Detection

In computer vision, the *object detection* problem

A similar problem can be found in the computer vision literature under *object detection*. The goal of object detection is to, given an input image, detect instances of semantic objects in it. This includes assigning a bounding box to the objects, and classifying the object. The input image is usually passively sampled, and the whole scene is visible at once.

Caicedo and Lazebnik (2015) [8] propose to use deep reinforcement learning for active object localization in images where the object to be localized is fully visible. An agent is trained to successively improve a bounding box using translating and scaling transformations. They use a reward signal that is proportional to how well the current box covers the target object.

An action that improves the region is rewarded with +1, and given a punishment of -1 otherwise. Without this quantization, the difference was small enough to confuse the agent. Binary rewards communicate more clearly which transformations keep the object inside the box and which take the box away from the target. When there is no action that improves the bounding box, the agent may select a trigger action (which would be the only action that does not give a negative reward) which resets the box. This way the agent may select additional bounding boxes. Each trigger modifies the environment by marking it so that the agent may learn to not select the same region twice. This is referred to as an inhibition-of-return mechanism, and is widely used in visual attention models [**[16] in caicedo_active_2015**]. This method has a few shortcomings for the problem considered in this project. The object may not be visible in the initial frame so the agent cannot act in the same way.

A separate field is active object search, which is perhaps most closely related to the problem we consider in this work. In active object search,

A similar work by Ghesu et al. (2016) [**ghesu_artificial_2016**] present an agent for anatomical landmark detection trained with DRL. Different from [8] is that the entire scene is not visible at once. The agent sees a limited region of interest in an image, with its center representing the current position of the agent. The actions available to the agent translate the view up, down, left and aright. A reward is given to the agent that is equal to the supervised relative distance-change to the landmark after each action. Three datasets of 891 anatomical images are used. The agent starts at random positions in the image close to the target landmark and is tasked with moving to the target location. While achieving strong results (90% success rate), the scenes and targets are all drawn from a distribution with low variance. Most real-world search tasks exhibit larger variance than anatomical images of the human body.

[42]. . .

Chen and Gupta [10] use a spatial memory for context reasoning in object detection. . .

2.2.4 Visual Attention

[28]

[31]

Soft attention (Bahdanau [4]). . .

Partially observable processes, such as the one we consider in this work, can be seens as hard attention problems. By taking actions, the hard attention can be redirected.

2.2.5 Coverage Path Planning

[15]

[25]

2.2.6 Visual Navigation

The task we consider bears resemblance to visual navigation [46].

Mnih et al. (2016) [30] use a recurrent policy with only RGB images to navigate in a labyrinth. 3D labyrinths are randomly generated, and an agent is tasked with finding objects in them. The same architecture as in [33] is used, but with 256 LSTM cells after the final hidden layer.

Mirowski et al. (2017) [29]...

Henriques and Vedaldi (2018) [20] use a spatial memory...

Gupta et al. (2019) [17] use a latent spatial memory. The They also use a planner that can plan paths given partial information of the environment. This allows the agent to take appearance of visited locations into account when deciding where to look next. The RGB observation is fed through an encoder network that. . . Planning in this fashion s

Dhiman et al. (2019) [13] critically investigate deep RL for navigation. They ask whether DRL algorithms are inherently able to gather and exploit environmental information for during navigation. Experimentally, they find that an agent is able to exploit environment information when trained and tested on the same map. However, when trained and tested on differen maps, it cannot do so succesfully. They further find that, with a single decision point whose correct.

Chaplot et al. (2020) [9] build on the idea of an explicit memory by including environment semantics…

Zhu et al. (2016) [48] create a model for target-driven visual navigation in indooor scenes with DRL. An observer is given a partial image of its scene as well as an image of the target object, and is tasked with navigating to the object n the scene with a minimal number of steps. The agent moves forwards, backwards, and turns left and right at constant step lengths. They use a reward signal with a small time penalty to incentivize task completion in few steps. They compare their approach to random walk and the shortest path and achieve promising results. This setup is quite similar to the one considered in this report, but the authors make a few assumptions that we do not. They a set of 32 scenes, each of which contain a fixed number of object instances. They focus on learning spatial relationships between objects in these specific scenes, and have scene-specific layers to achieve this. Thus, while they show that they can adapt a trained network to a new scene, their approach is unable to zero-shot generalize to new scenes.

A similar work by Ye et al. (2018) [45] integrates an object recognition module with a deep reinforcement learning based visual navigation module. They experiment with a set of reward functions and find that constant time penalizing rewards can be problematic and lead to slow convergence. Their experiments make the same assumptions as [**zhu_target_driven**] - the scenes and targets used during testing have all been seen during training.

2.2.7 Memory for Deep Reinforcement Learning

1. Frame stacking
2. Recurrent networks
3. Explicit memories [34, 35].

Oh et al. [34] use a differentiable retrieval memory.

Parisotto and Salakhutdinov [35] propose a structured memory…

2.2.8 Inductive Biases, Overfitting and Generalization in Deep Reinforcement Learning

Kirk et al. (2021) [24] survey generalization in deep RL.

While deep neural networks have proved to be effective function approximators for RL, they are also prone to *overfitting*. High-capacity models trained over a long time may memorize the distribution seen during training rather than general patterns. While studied in supervised learning, overfitting is generally been neglected in deep RL. Training and evaluation stages are typically not separated. Instead, the final return on the training environments is used as a measure of agent performance.

Zhang et al. (2018) [47] study overfitting and generalization in deep RL. With experiments, they show that RL agents are capable of memorizing training data, even when completely random. When the number of training samples exceeds the capacity of the agent, they overfit to them. When exposed to new but statistically similar environments during testing, test performance could vary significantly despite consistent training performance. The authors argue that good generalization requires that the *inductive bias* of the algorithms is compatible with the bias of the problems. The inductive bias refers to a priori algorithmic preferences, like neural network architecture. When comparing MLPs with CNNs, they find that MLPs tend to be better at fitting the training data are worse at generalizing. When rewards are spatially invariant, CNNs generalize much better than MLPs. The authors advocate for carefully designed testing protocols for detecting overfitting. The effectiveness of stochastic-based evaluation depends on the properties of the task. Agents could still learn to overfit to random training data. For this reason, they recommend isolation of statistically tied training and test sets.

In a similar spirit, Cobbe et al. (2019) [12] construct distinct training and test sets to measure generalization in RL. They find that agents can overfit to surprisingly large training sets, and that deep convolutional architectures can improve generalization. Methods from supervised learning, like L2 regularization, dropout, data augmentation and batch normalization are also shown to aid with generalization.

Many current deep RL agents do not optimize the true objective that they are evaluated against, but rather a handcrafted objective that incorporates biases to simplify learning. Stronger biases can lead to faster learning, while weaker biases potentially lead to more general agents. Hessel et al. (2019) [21] investigate the trade-off between generality and performance from the perspective of inductive biases. Through experimentation with common reward sculpting techniques, they find that learned solutions are competitive with domain heuristics like handcrafted objectives. Learned solutions also seem to be better at generalizing to unseen domains. For this reason, they argue for removing biases determined with domain knowledge in future research.

Cobbe et al. (2020) [11] introduce a benchmark for sample efficiency and generalization in RL. They make use of procedural generalization to decide many parameters of the initial state of the environment. This forces agents to learn policies that are robust variation and avoid overfitting. To evaluate sample efficiency of agents in the benchmark, they train and test on the full distribution of states. To evaluate generalization, they fix the number of training samples and then test on held out levels. When an episode ends, a new sample is drawn from the training set. Agents may train for arbitrarily many time steps. The number of training samples required to generalize is dependent on the particulars and difficulty of the environment. The authors choose the training set size to be near the region when generalization begins to take effect. Empirically they find that larger model architectures improve both sample efficiency and generalization. Agents strongly overfit to small training sets and need many samples to generalize. Interestingly, training performance improves as the training set grows past a certain threshold. The authors attribute this to the implicit curriculum of the distribution of levels.

2.2.9 Evaluation of Deep Reinforcement Learning Agents

A problem in state-of-the art RL is reproducibility. There is often non-determinism, both in the methods and environments used. Furthermore, many methods have intrincic variance which can make published results difficult to interpret. This has meant that reproducing state-of-the-art deep RL results is difficult.

Henderson et al. [19] discuss this problem from multiple perspectives. Through experimental analysis, they show that:

- In policy gradient methods, hyperparameters and the choice of network architecture for policy and value function approximation can affect performance significantly. They find that ReLU activations tend to perform best across environments and algorithms. For PPO, the use of large networks may require changing other hyperparameters like learning rate.

- Rescaling rewards can have a large effect, although it is difficult to predict how.

- Variance between random seeds in stochastic environments affects performance of algorithms, and give learning curves that do not fall within the same distribution. This suggests that selecting the top N trials or average over a small number of trials N can be misleading. They suggest to compare performance over many different random seeds.

- For certain environments, learning curves can indicate successful optimization but the learned behaviour may not be satisfactory. It is therefore important to not only show returns, but also demonstrations of the learned policy in action.

- Implementation differences that are not reflected in publications can have a dramatic impact on performance. It is therefore necessary to enumerate implementation details and package codebases with publications. Performance of baseline experiments should also match original baseline publication code.

Due to the unstable nature of RL algorithms, it is often inadequate to just report average return. Henderson et al. [19] propose to include confidence intervals when reporting results. Confidence bounds with sample bootstrapping is used to show that PPO is among the more stable algorithms. Various other significance tests…

Finally, they make the point that more emphasis should be placed on applying RL algorithms to real-world tasks. Benchmarks environments like ALE [**arcade**] often have no clear winner. It could be more useful to propose a set of tasks that an algorithm could be used for than to show performance on fictional tasks.

A similar work by Agarwal et al. (2022) [1] criticises the heavy use of point estimates of aggregate performance. They advocate for a set of performance metrics that take uncertainty in results into account. …

Anderson et al. (2018) [3] discuss problem statements and evaluation measures for embodied navigation agents, and make a set of recommendations. A navigation agent should be equipped with a special action that indicates that it has concluded the episode. The agent should be evaluated at the time this action is made, and not at some more favorable time step. Proximity to a goal should be measured using geodesic distance, the shortest distance in the environment. They recomnend success weighted by (normalized inverse) path length (SPL) as the primary measure of navigation performance. With N test episodes, SPL is computed as

$$\frac{1}{N}\sum_{i=1}^{N} S_i \frac{l_i}{\max(p_i, l_i)} \tag{2.5}$$

where S_i is a binary indicator of success, l_i is the shortest path distance from the agent's starting position to the goal, and p_i is the length of the path actually taken in the episode. If 50% of test episode are successful and the agent takes the optimal path in all of them, its SPL is 0.5. By measuring SPL of human subjects, what is a good score can be calibrated. Finally, they emphasize the importance of memory mechanisms that support the construction of rich internal representations of the agent's environment. Simple agents that are purely reactive and act on the sensory input at the current time step only work for simple tasks. Agumentations like reucrrent update mechanisms add more potential. More advanced memory mechanisms can be important for better navigation. The nature of the internal representation is central to the study of embodied navigation.

Batra et al. (2020) [5] do something similar…

3 Method

In this chapter, the method used is described. Section 3.1 formalizes the problem solved. Section 3.2 details the environment used to evaluate solutions. Section 3.4 describes the baseline learning method. Section 3.3 describes the approach used to solve the problem with a learning agent. Section 3.5 describes the experiments conducted to answer research questions 2 and 3.

3.1 Problem Statement

We can now formally define the problem of searching for targets in unknown environments. Searched environment contains a scene with a set of targets. The scene is described by a Euclidean space. In the scene, there are N targets, each described by a subspace. At any given time the agent observes a subspace of the scene, which we call the view. This observation is given in the form of an image. Through a finite set of actions, the agent can transform its view. With a final trigger action, the agent can indicate that there is a target in the view. The goal of the agent is to bring all targets into the view and indicate that they have been found. This is to be done with a minimal number of actions. This corresponds to changing the field of perception.

We denote the task by $\langle \mathcal{M}, \mathcal{T}_0 \rangle$, where $\mathcal{M}$ is a POMDP and $\mathcal{T}_0$ is the probability distribution on the initial states.

We focus on search in two dimensions, although all methods should scale to three dimensions.

3.2 Environment

To train an test an agent for the problem, we use three environments of varying difficulty. In each environment there is a scene with a background of distractors and a foreground of targets. The scenes are drawn from some unknown distribution. The background and foreground are assumed to be correlated. This means that by looking at the background, an agent should sometimes be able to deduce a suitable action. All scenes are assumed to be static in that the actions of the agent do not affect their appearance.

The scenes of each environment are discretized into a grid. We use the same action space, reward signal for all environments. The action space is

$$\mathcal{A} = \{\texttt{UP}, \texttt{DOWN}, \texttt{LEFT}, \texttt{RIGHT}, \texttt{TRIGGER}\},$$

where `UP`, `DOWN`, `LEFT`, and `RIGHT` translate the view and `TRIGGER` indicates that a target is in view.

We experiment with two rewards signals. The first reward signal is defined as

$$\mathcal{R}_1(s_t, a_t) = \begin{cases} 10 & \text{if } a_t = \texttt{TRIGGER} \text{ and a target is in view,} \\ -1 & \text{otherwise.} \end{cases}$$

We argue that R_1 provides a suitable inductive bias for the task at hand. Early experiments show that a constant reward of $r_t = -1$ that simply incentivizes the agent to complete the episode as quickly as possible don't converge for large state spaces. The reward for finding a target speeds up training. Targets should be triggered when in view, but triggers when targets are out of view should be penalized. The constant penalty of -1 in all other cases assures that the agent is rewarded for quick episode completion. This reward signal is maximized when the targets are found as quickly as possible.

The second reward signal, similar to that of [**...**, 8], is defined as

$$\mathcal{R}_2(s_t, a_t) = \begin{cases} 10 & \text{if } a_t = \texttt{TRIGGER} \text{ and a target is in view,} \\ 1 & \text{if } a_t \text{ moves the view closer to the nearest target, and} \\ -1 & \text{otherwise.} \end{cases}$$

R_2 uses the supervised distance between targets and the agent which is available during training time. We hypothesize that this reward speeds up training time and may help the agent pick out correlations between scene and target probability quicker. However, it can never yield policies that search optimally nor exhaustively as these will not be rewarded. It will also not learn to take the shortest paths in the general case, as selecting waypoints greedily does not

At each time step, the agent receives an observation that includes

- an RGB image $x_t \in \mathbb{R}^{3 \times W \times H}$ of the currently visible region of the scene, and
- the current position of the agent $p_t \in \mathbb{R}^{W \times H}$.

For the position, we assume the presense of some oracle. In many realistic scenarios this is the case (GPS, pan/tilt, etc.). If how each action moves the agent is well-defined, we do not need the position at all. We can use relative positions instead of absolute ones. Some of the baselines do not use the position.

The episode is terminated when all targets have been found, or when 1000 time steps have passed. Terminating episodes early this way is common to speed up training.

Averaged over all possible samples, the probability of targets should be uniform over the scene.

3.2.1 Gaussian Environment

The first environment is the simplest environment. The scene is a two-dimensional discrete Euclidean space. The appearance of the scene is given by a 256x256 RGB image. The agent observes a 64x64 sub-image at each time step. In the image there are three Gaussian kernels with random positions. The height of the kernel is indicated by a higher intensity in the blue channel. Targets are 1x1 pixels in the red channel. The locations of the targets are randomized weighted by the height of the Gaussian kernels. This means that the more intense the blue channel, the higher the probability of a target. The idea with this environment is to test that the method learns what we want it to learn. It is easy to determine whether the agent acts well in this environment. Our feeling is that this is something that previous similar works has not done.

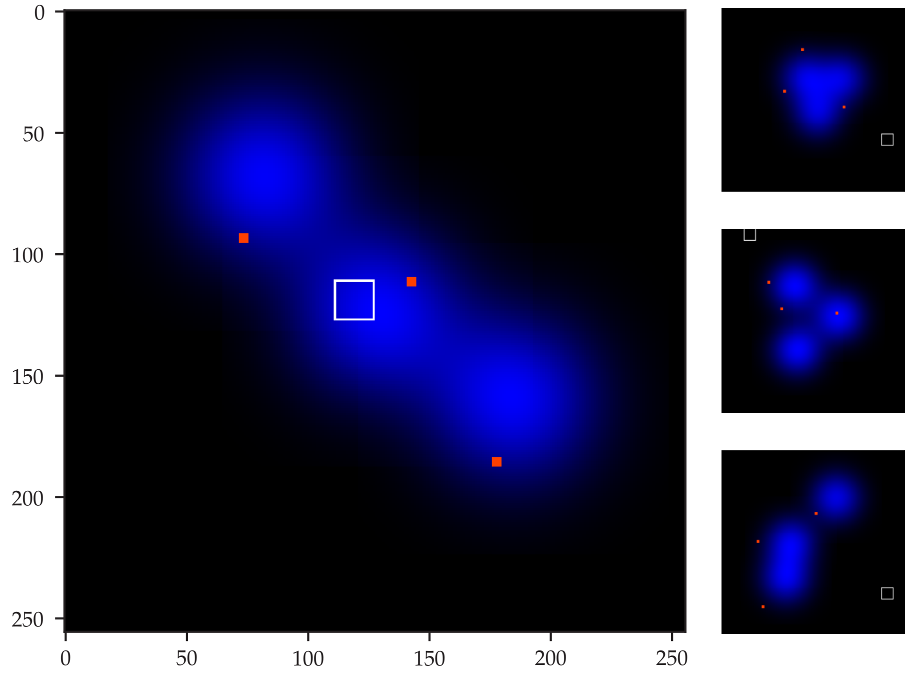

Figure 3.1: Four samples of the first environment. There are three gaussian kernels in the environment, whose height is visualized with the blue channel. There are three targets in the environment, whose location is sampled from the distribution defined by the sum of the three gaussian kernels.

3.2.2 Terrain Environment

The second environment is intended to look like realistic terrain. The environment has a two-dimensional scene whose appearance is given by a 512x512 image. Gradient noise is generated and used as a height map. The height map determines the color of the terrain. The height also correlates to the probability of targets. Specifically, targets are located between shores and mountain bases. This environment simulates a UAV search-and-rescue scenario.

3.2.3 Realistic Environment

The third environment is a three-dimensional version of the second one.

3.3 Approach

Due to the advantages described in Section **??**, we will limit our approaches to policy gradients.

We postulate that an effective searcher should be able to:

- Search the scene exhaustively, while avoiding visiting the same location multiple times.
- Learn a probability distribution of targets and its correlation with the appearance of the scene.

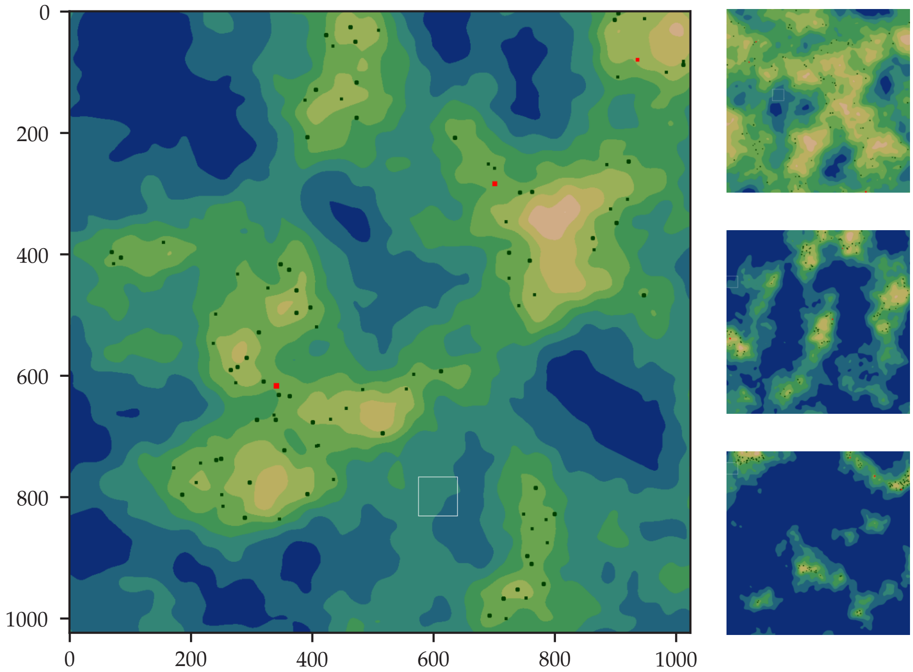

Figure 3.2: Second environment. Terrain seen from above.

- Remember the appearance of previously visited areas in order to prioritize where to go next.

The agent is trained with reinforcement learning using Proximal Policy Optimization [**proxim**] with function approximation using neural networks.

3.3.1 Architectures

The architecture of the neural network is presented in Figure X. The network is split into three main section: the feature extraction, the shared network, and the policy and value heads. In order to be able to use the same extractor for all environments we resize the RGB image to 64×64 pixels.

3.4 Baselines

We compare our architecture to three different baselines. As [29], we use a

3.5 Experiments

The agent was trained using the algorithm described in Section 3.3 for 100 million time steps in all three environments using PPO, PPG and A2C. Hyperparameters are tuned with random search separately for each environment. For all experiments, the average return per episode is reported together with the theoretically optimal reward (obtained with an optimal path). The agent was trained and tested on the full distribution of environments.

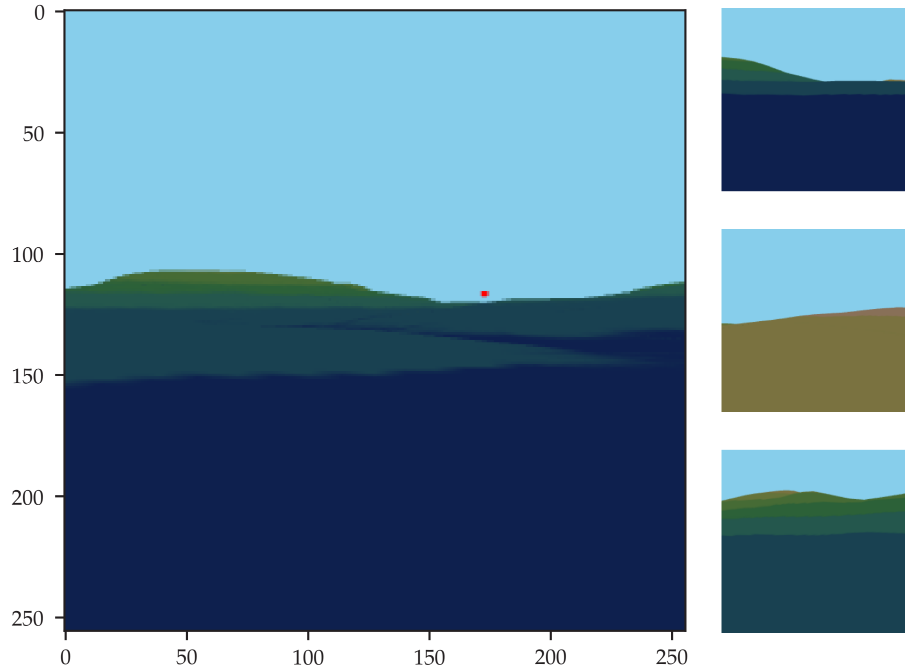

Figure 3.3: Second environment. Terrain seen from a pan-tilt-zoom camera.

We compare the two reward signals on the first and second environment. . .

The baselines and our approach was. . .

Additionally, experiments to eveluate the generalization capability of the agent were conducted. These were conducted on the procedurally generated terrain environment following the approach suggested in [**procgen**]. During training, the seed pool size was fixed to various sizes to limit the training set size. The agent was trained for varying number of timesteps and then tested on the full distribution of environments. This way, we can get a sense of how much data and simulation is required to use the approach for real-world tasks.

We compare the approach to random walk, exhaustive search and a human searcher with prior knowledge of the characteristics of the searched environments.

All experiments are conducted on an Intel Core i9-10900X CPU and an NVIDIA GeForce RTX 2080 Ti GPU.

For each experiment, we report the mean return and episode length over time during training. As per [1], we report results across multiple seeds. We also use their recommended. . .

During testing, we increase the maximum episode length from 1000 time steps to 5000 time steps. This is to

3.6 Implementation

The environment is implemented with Gym [7]. The agent is implemented and RL algorithms are implemented with PyTorch [36] for automatic differentiation of the computation graphs.

Proximal policy optimization was implemented following the official implementation by OpenAI. Some necessary modifications were made to allow for recurrent policies.

4 Results

When it comes to hyperparameters, we find that letting the number of rollout steps be substantially lower than the episode length we achieve much more stable training results. Furthermore, increasing the number of weights in the neural network made it more difficult to train.

We find that the hyperparameters from [procgen] perform well, especially when the number of environments is large.

Also, proximal policy optimization was unstable without reward normalization.

This coupled with a sparse reward signal led to many cases where the agent converged towards a poor local optimum (or perhaps never converged at all).

5 Discussion

This chapter contains the following sub-headings.

5.1 Results

5.2 Method

It is worth considering whether using a learning agent like this is suitable for this task. One could imagine that it is possible to compute an optimal strategy for certain environments. However, this quickly falls apart. The dynamics of environments can vary considerably which may drastically affect how a manual approach is implemented.

Another thing worth discussing is the possibility of combining manual search method with reinforcement learning. One could imagine combining a frontier based approach with a learning approach.

In this work, we have only covered searches where the view is transformed in the spatial domain. However, the method could be applied to a broader category of problems. For instance, one could imagine a scenario when searching along the time dimension is useful. If we let the actions be translations arbitrary translations along the time dimensions in, say, a long audio or video file, the agent could learn to look for landmark features in such modalities.

5.3 The work in a wider context

While automated search systems have many positive uses, like XXX, there are certainly other use cases that could be considered negative. Mass surveillance, XXX, are both very relevant today.

6 Conclusion

Bibliography

[1] Rishabh Agarwal, Max Schwarzer, Pablo Samuel Castro, Aaron Courville, and Marc G. Bellemare. "Deep Reinforcement Learning at the Edge of the Statistical Precipice". In: *arXiv:2108.13264 [cs, stat]* (Jan. 2022). arXiv: 2108.13264. URL: `http://arxiv.org/abs/2108.13264` (Cited on pages 13, 18).

[2] John Aloimonos, Isaac Weiss, and Amit Bandyopadhyay. "Active vision". In: *International Journal of Computer Vision* 1.4 (Jan. 1988). 705 citations (Crossref) [2022-02-07], pp. 333–356. ISSN: 0920-5691, 1573-1405. DOI: `10/cn4mdc`. URL: `http://link.springer.com/10.1007/BF00133571` (visited on 02/07/2022) (Cited on page 4).

[3] Peter Anderson, Angel Chang, Devendra Singh Chaplot, Alexey Dosovitskiy, Saurabh Gupta, Vladlen Koltun, Jana Kosecka, Jitendra Malik, Roozbeh Mottaghi, Manolis Savva, and Amir R. Zamir. "On Evaluation of Embodied Navigation Agents". In: *arXiv:1807.06757 [cs]* (July 2018). arXiv: 1807.06757. URL: `http://arxiv.org/abs/1807.06757` (Cited on page 13).

[4] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. "Neural Machine Translation by Jointly Learning to Align and Translate". In: *arXiv:1409.0473 [cs, stat]* (May 2016). arXiv: 1409.0473. URL: `http://arxiv.org/abs/1409.0473` (Cited on page 10).

[5] Dhruv Batra, Aaron Gokaslan, Aniruddha Kembhavi, Oleksandr Maksymets, Roozbeh Mottaghi, Manolis Savva, Alexander Toshev, and Erik Wijmans. "ObjectNav Revisited: On Evaluation of Embodied Agents Navigating to Objects". In: *arXiv:2006.13171 [cs]* (Aug. 2020). arXiv: 2006.13171. URL: `http://arxiv.org/abs/2006.13171` (Cited on page 13).

[6] Yoshua Bengio, Aaron Courville, and Pascal Vincent. "Representation Learning: A Review and New Perspectives". In: *arXiv:1206.5538 [cs]* (Apr. 2014). arXiv: 1206.5538. URL: `http://arxiv.org/abs/1206.5538` (Cited on page 7).

[7] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. "OpenAI Gym". In: *arXiv:1606.01540 [cs]* (June 2016). arXiv: 1606.01540. URL: `http://arxiv.org/abs/1606.01540` (Cited on page 18).

[8] Juan C. Caicedo and Svetlana Lazebnik. "Active Object Localization with Deep Reinforcement Learning". In: *arXiv:1511.06015 [cs]* (Nov. 18, 2015). arXiv: `1511.06015`. URL: `http://arxiv.org/abs/1511.06015` (visited on 02/03/2022) (Cited on pages 9, 10, 15).

[9] Devendra Singh Chaplot, Dhiraj Gandhi, Abhinav Gupta, and Ruslan Salakhutdinov. "Object Goal Navigation using Goal-Oriented Semantic Exploration". In: *arXiv:2007.00643 [cs]* (July 2020). arXiv: 2007.00643. URL: `http://arxiv.org/abs/2007.00643` (Cited on page 11).

[10] Xinlei Chen and Abhinav Gupta. "Spatial Memory for Context Reasoning in Object Detection". In: *2017 IEEE International Conference on Computer Vision (ICCV)*. IEEE, Oct. 2017, pp. 4106–4116. ISBN: 978-1-5386-1032-9. DOI: `10.1109/ICCV.2017.440`. URL: `http://ieeexplore.ieee.org/document/8237702/` (Cited on page 10).

[11] Karl Cobbe, Christopher Hesse, Jacob Hilton, and John Schulman. "Leveraging Procedural Generation to Benchmark Reinforcement Learning". In: *arXiv:1912.01588 [cs, stat]* (July 2020). arXiv: 1912.01588. URL: `http://arxiv.org/abs/1912.01588` (Cited on page 12).

[12] Karl Cobbe, Oleg Klimov, Chris Hesse, Taehoon Kim, and John Schulman. "Quantifying Generalization in Reinforcement Learning". In: *arXiv:1812.02341 [cs, stat]* (July 2019). arXiv: 1812.02341. URL: `http://arxiv.org/abs/1812.02341` (Cited on page 12).

[13] Vikas Dhiman, Shurjo Banerjee, Brent Griffin, Jeffrey M. Siskind, and Jason J. Corso. "A Critical Investigation of Deep Reinforcement Learning for Navigation". In: *arXiv:1802.02274 [cs]* (Jan. 2019). arXiv: 1802.02274. URL: `http://arxiv.org/abs/1802.02274` (Cited on page 11).

[14] M. P. Eckstein. "Visual search: A retrospective". In: *Journal of Vision* 11.5 (Dec. 30, 2011). 207 citations (Crossref) [2022-02-28], pp. 14–14. ISSN: 1534-7362. DOI: `10.1167/11.5.14`. URL: `http://jov.arvojournals.org/Article.aspx?doi=10.1167/11.5.14` (visited on 02/22/2022) (Cited on pages 2, 5).

[15] Enric Galceran and Marc Carreras. "A survey on coverage path planning for robotics". In: *Robotics and Autonomous Systems* 61.12 (Dec. 2013), pp. 1258–1276. ISSN: 09218890. DOI: `10/f5j2n5` (Cited on page 10).

[16] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. Adaptive Computation and Machine Learning series. MIT Press, Nov. 2016. ISBN: 978-0-262-03561-3 (Cited on pages 7, 8).

[17] Saurabh Gupta, Varun Tolani, James Davidson, Sergey Levine, Rahul Sukthankar, and Jitendra Malik. "Cognitive Mapping and Planning for Visual Navigation". In: *arXiv:1702.03920 [cs]* (Feb. 2019). arXiv: 1702.03920. URL: `http://arxiv.org/abs/1702.03920` (Cited on page 10).

[18] Matthew Hausknecht and Peter Stone. "Deep Recurrent Q-Learning for Partially Observable MDPs". In: *arXiv:1507.06527 [cs]* (Jan. 2017). arXiv: 1507.06527. URL: `http://arxiv.org/abs/1507.06527` (Cited on page 9).

[19] Peter Henderson, Riashat Islam, Philip Bachman, Joelle Pineau, Doina Precup, and David Meger. "Deep Reinforcement Learning That Matters". In: *Proceedings of the AAAI Conference on Artificial Intelligence* 32.11 (Apr. 2018). ISSN: 2374-3468. URL: `https://ojs.aaai.org/index.php/AAAI/article/view/11694` (Cited on pages 12, 13).

[20] Joao F. Henriques and Andrea Vedaldi. "MapNet: An Allocentric Spatial Memory for Mapping Environments". In: *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*. IEEE, June 2018, pp. 8476–8484. ISBN: 978-1-5386-6420-9. DOI: `10.1109/CVPR.2018.00884`. URL: `https://ieeexplore.ieee.org/document/8578982/` (Cited on page 10).

[21] Matteo Hessel, Hado van Hasselt, Joseph Modayil, and David Silver. "On Inductive Biases in Deep Reinforcement Learning". In: *arXiv:1907.02908 [cs, stat]* (July 2019). arXiv: 1907.02908. URL: `http://arxiv.org/abs/1907.02908` (Cited on page 12).

[22] Sepp Hochreiter and Jürgen Schmidhuber. "Long Short-Term Memory". In: *Neural Computation* 9.8 (Nov. 1997), pp. 1735–1780. ISSN: 0899-7667. DOI: 10.1162/neco.1997.9.8.1735 (Cited on page 8).

[23] Leslie Pack Kaelbling, Michael L. Littman, and Anthony R. Cassandra. "Planning and acting in partially observable stochastic domains". In: *Artificial Intelligence* 101.1–2 (May 1998), pp. 99–134. ISSN: 00043702. DOI: 10.1016/S0004-3702(98)00023-X (Cited on pages 5, 6).

[24] Robert Kirk, Amy Zhang, Edward Grefenstette, and Tim Rocktäschel. "A Survey of Generalisation in Deep Reinforcement Learning". In: *arXiv:2111.09794 [cs]* (Jan. 2022). arXiv: 2111.09794. URL: http://arxiv.org/abs/2111.09794 (Cited on page 11).

[25] Anirudh Krishna Lakshmanan, Rajesh Elara Mohan, Balakrishnan Ramalingam, Anh Vu Le, Prabahar Veerajagadeshwar, Kamlesh Tiwari, and Muhammad Ilyas. "Complete coverage path planning using reinforcement learning for Tetromino based cleaning and maintenance robot". In: *Automation in Construction* 112 (Apr. 2020), p. 103078. ISSN: 0926-5805. DOI: 10.1016/j.autcon.2020.103078 (Cited on page 10).

[26] Maja J Mataric. "Reward Functions for Accelerated Learning". In: *Machine Learning Proceedings 1994*. Ed. by William W. Cohen and Haym Hirsh. Morgan Kaufmann, Jan. 1994, pp. 181–189. ISBN: 978-1-55860-335-6. DOI: 10.1016/B978-1-55860-335-6.50030-1. URL: https://www.sciencedirect.com/science/article/pii/B9781558603356500301 (Cited on page 7).

[27] Marvin Minsky. "Steps toward Artificial Intelligence". In: *Proceedings of the IRE* 49.1 (Jan. 1961), pp. 8–30. ISSN: 2162-6634. DOI: 10.1109/JRPROC.1961.287775 (Cited on page 7).

[28] Silviu Minut and Sridhar Mahadevan. "A reinforcement learning model of selective visual attention". In: *Proceedings of the fifth international conference on Autonomous agents - AGENTS '01*. ACM Press, 2001, pp. 457–464. ISBN: 978-1-58113-326-4. DOI: 10/dbwckq. URL: http://portal.acm.org/citation.cfm?doid=375735.376414 (Cited on page 10).

[29] 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". In: *arXiv:1611.03673 [cs]* (Jan. 2017). arXiv: 1611.03673. URL: http://arxiv.org/abs/1611.03673 (Cited on pages 10, 17).

[30] Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. "Asynchronous Methods for Deep Reinforcement Learning". In: *arXiv:1602.01783 [cs]* (June 2016). arXiv: 1602.01783. URL: http://arxiv.org/abs/1602.01783 (Cited on page 10).

[31] Volodymyr Mnih, Nicolas Heess, Alex Graves, and Koray Kavukcuoglu. "Recurrent Models of Visual Attention". In: *arXiv:1406.6247 [cs, stat]* (June 2014). arXiv: 1406.6247. URL: http://arxiv.org/abs/1406.6247 (Cited on page 10).

[32] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. "Playing Atari with Deep Reinforcement Learning". In: *arXiv:1312.5602 [cs]* (Dec. 2013). arXiv: 1312.5602. URL: http://arxiv.org/abs/1312.5602 (Cited on page 9).

[33] 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". In: *Nature* 518.75407540 (Feb. 2015), pp. 529–533. ISSN: 1476-4687. DOI: 10.1038/nature14236 (Cited on pages 9, 10).

[34] Junhyuk Oh, Valliappa Chockalingam, Satinder Singh, and Honglak Lee. "Control of Memory, Active Perception, and Action in Minecraft". In: *arXiv:1605.09128 [cs]* (May 2016). arXiv: 1605.09128. URL: http://arxiv.org/abs/1605.09128 (Cited on page 11).

[35] Emilio Parisotto and Ruslan Salakhutdinov. "Neural Map: Structured Memory for Deep Reinforcement Learning". In: *arXiv:1702.08360 [cs]* (Feb. 2017). arXiv: 1702.08360. URL: http://arxiv.org/abs/1702.08360 (Cited on page 11).

[36] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. "PyTorch: An Imperative Style, High-Performance Deep Learning Library". In: (), p. 12 (Cited on page 18).

[37] Stuart J. Russell and Peter Norvig. *Artificial intelligence: a modern approach*. In collab. with Ming-wei Chang, Jacob Devlin, Anca Dragan, David Forsyth, Ian Goodfellow, Jitendra Malik, Vikash Mansinghka, Judea Pearl, and Michael Woolridge. Fourth Edition. Pearson Series in Artificial Intelligence. Hoboken, NJ: Pearson, 2021. 1115 pp. ISBN: 978-0-13-461099-3 (Cited on pages 7, 8).

[38] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. "Proximal Policy Optimization Algorithms". In: *arXiv:1707.06347 [cs]* (Aug. 2017). arXiv: 1707.06347. URL: http://arxiv.org/abs/1707.06347 (Cited on page 9).

[39] Richard S Sutton, David McAllester, Satinder Singh, and Yishay Mansour. "Policy Gradient Methods for Reinforcement Learning with Function Approximation". In: *Advances in Neural Information Processing Systems*. Ed. by S. Solla, T. Leen, and K. Müller. Vol. 12. MIT Press, 1999. URL: https://proceedings.neurips.cc/paper/1999/file/464d828b85b0bed98e80ade0a5c43b0f-Paper.pdf (Cited on pages 6, 7).

[40] Richard S. Sutton and Andrew G. Barto. *Reinforcement learning: an introduction*. Second edition. Adaptive computation and machine learning series. Cambridge, Massachusetts: The MIT Press, 2018. 526 pp. ISBN: 978-0-262-03924-6 (Cited on pages 5–7).

[41] Gerald Tesauro et al. "Temporal difference learning and TD-Gammon". In: *Communications of the ACM* 38.3 (1995), pp. 58–68 (Cited on page 9).

[42] Burak Uzkent, Christopher Yeh, and Stefano Ermon. "Efficient Object Detection in Large Images using Deep Reinforcement Learning". In: *arXiv:1912.03966 [cs]* (Apr. 2020). arXiv: 1912.03966. URL: http://arxiv.org/abs/1912.03966 (Cited on page 10).

[43] Jeremy M. Wolfe. "Guided Search 6.0: An updated model of visual search". In: *Psychonomic Bulletin & Review* 28.4 (Aug. 2021). 29 citations (Crossref) [2022-03-02], pp. 1060–1092. ISSN: 1531-5320. DOI: 10.3758/s13423-020-01859-9 (Cited on pages 2, 5).

[44] Jeremy M. Wolfe. "Visual search". In: *Current biology : CB* 20.8 (Apr. 27, 2010). 64 citations (Crossref) [2022-03-02] Publisher: NIH Public Access, R346. DOI: 10.1016/j.cub.2010.02.016. URL: https://www.ncbi.nlm.nih.gov/labs/pmc/articles/PMC5678963/ (visited on 03/02/2022) (Cited on page 2).

[45] Xin Ye, Zhe Lin, Haoxiang Li, Shibin Zheng, and Yezhou Yang. "Active Object Perceiver: Recognition-Guided Policy Learning for Object Searching on Mobile Robots". In: *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). ISSN: 2153-0866. Oct. 2018, pp. 6857–6863. DOI: 10.1109/IROS.2018.8593720 (Cited on page 11).

[46] Fanyu Zeng, Chen Wang, and Shuzhi Sam Ge. "A Survey on Visual Navigation for Artificial Agents With Deep Reinforcement Learning". In: *IEEE Access* 8 (2020), pp. 135426–135442. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2020.3011438 (Cited on page 10).

[47] Chiyuan Zhang, Oriol Vinyals, Remi Munos, and Samy Bengio. "A Study on Overfitting in Deep Reinforcement Learning". In: *arXiv:1804.06893 [cs, stat]* (Apr. 2018). arXiv: 1804.06893. URL: http://arxiv.org/abs/1804.06893 (Cited on page 11).

[48] 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: *arXiv:1609.05143 [cs]* (Sept. 16, 2016). arXiv: 1609.05143. URL: http://arxiv.org/abs/1609.05143 (visited on 03/14/2022) (Cited on page 11).