Investigating memory: an intersection of neuroscience and artificial intelligence

Andrea Banino

A dissertation submitted in partial fulfillment of the requirements for the degree of
Doctor of Philosophy
of
University College London.

Centre for Mathematics, Physics and Engineering in the Life Sciences and Experimental Biology
University College London

May 4, 2020
I, Andrea Banino, confirm that the work presented in this thesis is my own. Where information has been derived from other sources, I confirm that this has been indicated in the work.
Abstract

In recent years, deep neural networks have enjoyed tremendous successes in a variety of challenging tasks. Despite these breakthroughs, there remain key areas in which humans are still strikingly superior: the ability to learn in a one-shot fashion (episodic memory) and spatial navigation being two core examples. Fortuitously, these areas are topics in neuroscience that have deep theoretical and empirical foundations. Consequently, in this body of work we drew on this opportunity to develop neuroscience-inspired architectures that support navigation and episodic memory and in so doing, we also provided new neuroscientific insights. Specifically we identified architectural constraints in neural network models that allowed the emergence of spatial representation that resemble the ones found in the mammalian brain (e.g. place cells, grid cells, head direction cells). Grid cells in particular are believed to provide multi-scale periodic representation that functions as a metric for coding space which is critical to plan direct trajectories to goals. To test this hypothesis we used our artificial agent to show that emergent grid-like representations furnish it with a Euclidean spatial metric and associated vector operations, providing a foundation for proficient navigation. As such, our results supported neuroscientific theories that see grid cells as critical for vector-based navigation. In a second line of work we focused on episodic memory and in particular on the role of the hippocampus in generalisation. We employed a classic associative inference task from the human neuroscience literature - the paired associative inference task (PAI) - to carefully probe the reasoning capacity of existing memory-augmented neural networks. Surprisingly, we found that current architectures struggle to reason over long distance associations. Consequently we developed a new memory architecture
inspired by research on how the hippocampus supports generalisation. This new architecture was capable of solving the PAI tasks, as well as other challenging machine learning tasks. To sum up, in our work we showed a considerable potential for synergy between neuroscience and deep learning, whereby the latter can be used as a tool to validate theories from the former. At the same time we also showed that neuroscience can be used to inspire artificial architectures capable of solving hard memory tasks, where traditional methods have failed.
Acknowledgements

First of all I would like to thank DeepMind for giving me the chance and the scholarship to pursue this PhD. I want to personally thank Demis for this and for being a continuous source of inspiration. To Caswell, my supervisor, thanks for all the neuroscientific insights, the guidance and the kindness throughout these years. Also a big thanks to Charles, for teaching me a lot of machine learning and for always giving me a different point of view on things, this has been key for my professional development. To my dear friend and mentor Dharsh, it is difficult to find the right words to say how much grateful I am, this would have simply been impossible without you, thanks for believing in me!

I want to thank my wife, by walking side by side we made another step together. To my son Lorenzo, without saying a single word you already gave me so many lessons. My mum, sorry if getting here has not been straightforward, but thanks for always being present and supportive. And finally, to my dad... well, you are not here to see this, but I know you would have been over the moon. I’m not sure if I will ever write something else so... this is not much, but it is dedicated to you, and to your smile. Grazie papà.
Publications Arising

The following publications were produced from work undertaken as part of this thesis:


Impact Statement

The goal of general artificial intelligence is to design new computational systems that can solve a wide variety of tasks with similar or better performance than humans. In contrast, systems neuroscience has the goal of explaining how behaviour emerges from brain activity. In this work we tried to merge these two fields to develop new neuroscience-inspired artificial models with the aim of providing new neuroscientific insights. In particular we focused our attention on the domains of episodic memory and spatial navigation, as these areas have received intensive investigation from neuroscience and, are at this time, of central interest for designing artificial agents.

In the first part of the work we used deep-learning to investigate the role of grid cells in supporting vector based navigation, that is calculate the direct route between two places. In particular we developed a new deep learning architecture to perform the task of self localisation to investigate whether grid cells emerged in the network as a consequence of performing the task. Indeed the “grid units” that spontaneously emerged in the network were remarkably similar to the ones found in the mammalian brains. We then embedded this system into a deep reinforcement learning agent that was able to find the shortest path to a goal even in complex mazes with random blockages. None of the the other agents we tested, all lacking “grid-like” units, demonstrated the same performance. This work was impactful for the neuroscience community as it was an example of how deep learning could be used as tool to validate neuroscientific theories. It seems likely that a similar approach could be used by researchers interested in limb control. They could train a neural network to control a robotic arm the way that the brain controls a living
arm, and then run experiments on the artificial system to generate further insights into the living one. We believe that this aspect makes our approach a general purpose neuroscience tool. At the same time our model is the first artificial agent that solves complex navigation tasks, like finding shortcuts in unexplored mazes. So by solving such complicated tasks we strengthen the idea that by taking inspiration from neuroscience we can design deep learning systems with the ability to tackle problems closer to the ones that require high level cognitive functions.

In the second part of this work we employed a classic associative inference task from the memory-based reasoning neuroscience literature in order to more carefully probe the reasoning capacity of existing neural network architectures. In particular the associative inference task was used to capture the essence of reasoning, that is the appreciation of distant relationships among elements distributed across multiple facts or memories. In our analysis we found that current artificial neural network architectures struggle to reason over long distance associations. Consequently we designed a new neural network architecture endowed with the capacity to perform multistep reasoning. Our new model was able to solve the associative inference task and was also state of the art in a challenging question and answering language task. The architecture presented in this work has been patented with potential uses in commercial artificial intelligence systems.
Contents

1 Introduction 15

1.1 Cognition 15
1.2 Understanding complex systems 17
1.3 Artificial neural networks 18
  1.3.1 A brief description of Artificial neural networks and back-
       propagation 19
  1.3.2 Feed-Forward and Recurrent Networks 21
  1.3.3 Frameworks used to train Neural Networks 22
  1.3.4 The longstanding relation between Neural Networks and
       Neuroscience 24
1.4 Attention 27
1.5 Memory 29
  1.5.1 Working Memory 30
  1.5.2 Episodic Memory 33
  1.5.3 An hybrid approach: Neural network augmented with ex-
       ternal memory 34
1.6 Using ANN to model Cognition 35

2 Understanding Spatial Navigation using Deep Learning 37

2.1 A Brief Outline of the Neural Mechanisms of Self-Localisation 38
2.2 Methods 41
  2.2.1 Supervised learning - Path integration experiments and set-
       tings 42
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.2.2 Deep Reinforcement Learning - Environments and Tasks</td>
<td>46</td>
</tr>
<tr>
<td>2.2.3 Deep Reinforcement Learning Agent Architectures</td>
<td>51</td>
</tr>
<tr>
<td>2.2.4 Deep Reinforcement Learning - Training Algorithms and Regimes</td>
<td>59</td>
</tr>
<tr>
<td>2.2.5 Deep Reinforcement Learning - Control experiments</td>
<td>61</td>
</tr>
<tr>
<td>2.2.6 Neuroscience-based analyses of network units</td>
<td>62</td>
</tr>
<tr>
<td>2.3 Results</td>
<td>68</td>
</tr>
<tr>
<td>2.3.1 Supervised Learning - Path Integration in a Square Arena</td>
<td>68</td>
</tr>
<tr>
<td>2.3.2 Deep Reinforcement Learning - Path Integration in the Virtual Morris Watermaze</td>
<td>72</td>
</tr>
<tr>
<td>2.3.3 Deep Reinforcement Learning - Vector Based Navigation in the Virtual Morris Watermaze</td>
<td>74</td>
</tr>
<tr>
<td>2.3.4 Comparison of grid cell agent with other agents in challenging, procedurally-generated multi-room environments</td>
<td>78</td>
</tr>
<tr>
<td>2.3.5 Deep Reinforcement Learning - Shortcut</td>
<td>80</td>
</tr>
<tr>
<td>2.4 Discussion</td>
<td>83</td>
</tr>
<tr>
<td>3 Modelling inferential reasoning with Memory Augmented Deep Neural Networks</td>
<td>86</td>
</tr>
<tr>
<td>3.1 Introduction</td>
<td>86</td>
</tr>
<tr>
<td>3.1.1 MEMO: a Deep Neural Network Model for the Flexible Recombination of Episodic Memories</td>
<td>88</td>
</tr>
<tr>
<td>3.2 Methods</td>
<td>90</td>
</tr>
<tr>
<td>3.2.1 Recapitulating End-to-End Memory Networks</td>
<td>91</td>
</tr>
<tr>
<td>3.2.2 MEMO</td>
<td>93</td>
</tr>
<tr>
<td>3.2.3 The Halting Policy</td>
<td>95</td>
</tr>
<tr>
<td>3.2.4 Baselines</td>
<td>96</td>
</tr>
<tr>
<td>3.2.5 Training details</td>
<td>98</td>
</tr>
<tr>
<td>3.2.6 Paired associative inference task</td>
<td>98</td>
</tr>
<tr>
<td>3.2.7 Shortest path task</td>
<td>101</td>
</tr>
<tr>
<td>3.2.8 bAbi - Training and evaluation details</td>
<td>103</td>
</tr>
</tbody>
</table>
3.3 Results ................................................. 105  
  3.3.1 Paired associative inference ................. 105  
  3.3.2 Shortest path on randomly generated graphs . 110  
  3.3.3 Question answering on the bAbI tasks .......... 112  
3.4 Discussion ........................................ 113  

4 General Conclusions 116  
  4.1 Concept learning and manipulation ............. 116  
  4.2 What can artificial neural networks teach us? ...... 119  

Appendices 121  
A Appendix for Chapter 2 121  
B Appendix for Chapter 3 127  
  B.1 Paired Associative inference ................... 128  
     B.1.1 Attention weights analysis ................. 128  
     B.1.2 Adaptive computation ...................... 129  
  B.2 bAbI ............................................. 129  
     B.2.1 Task-wise results ......................... 129  
  B.3 MEMO complexity analysis ..................... 129  
  B.4 MEMO training details and hyper-parameters .... 130  
  B.5 Baselines hyper-parameters ................... 131  
     B.5.1 DNC ..................................... 131  
     B.5.2 Universal transformer .................... 132  
  B.6 ACT description ................................ 132  

Bibliography 134
List of Figures

1.1 Feed-forward Artificial Neural Networks ........................................... 19
1.2 Recurrent Neural Network................................................................. 21

2.1 Examples of paths used to train the grid network. ............................ 42
2.2 Network architecture in the supervised learning experiment. ............... 44
2.3 DeepMind Lab. ................................................................................. 47
2.4 Architecture of the grid cell agent. ...................................................... 51
2.5 Architecture of the A3C and place cell agent. .................................... 54
2.6 Architecture of the place cell prediction agent and of the NavMemNet agent. ................................................................. 56
2.7 Emergence of Enthorinal Grid Cells in a Deep Neural Network Trained to Path Integrate. ...................................................... 69
2.8 Characterization of grid-like units in Square environment and Circular environment. .................................................. 70
2.9 Characterisation of grid-like representations and robustness of performance for the grid cell agent in the square “land maze” environment. .................................................. 72
2.10 One-shot open field navigation to a hidden goal. ............................... 75
2.11 Navigation in complex environmentsl ................................................. 80
2.12 Flexible use of short-cuts. ................................................................. 82

3.1 MEMO architecture ........................................................................... 91
3.2 Paired associative inference. ............................................................... 98
3.3 Example of Graph. ............................................................................. 101
### List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.4</td>
<td>bAbI task.</td>
<td>103</td>
</tr>
<tr>
<td>3.5</td>
<td>Number of Hops</td>
<td>107</td>
</tr>
<tr>
<td>3.6</td>
<td>Analysis of length 3 PAI task.</td>
<td>109</td>
</tr>
<tr>
<td>3.7</td>
<td>Weights analysis of an inference query in the length 3 PAI task.</td>
<td>110</td>
</tr>
<tr>
<td>A.1</td>
<td>Linear layer spatial activity maps from the supervised learning experiment.</td>
<td>122</td>
</tr>
<tr>
<td>A.2</td>
<td>Grid-like units did not emerge in the linear layer when dropout was not applied.</td>
<td>123</td>
</tr>
<tr>
<td>A.3</td>
<td>Robustness of grid cell agent and performance of other agents.</td>
<td>124</td>
</tr>
<tr>
<td>B.1</td>
<td>Attention weights analysis of length 3 PAI task</td>
<td>128</td>
</tr>
<tr>
<td>B.2</td>
<td>Comparison between MEMO + REINFORCE and MEMO + ACT on length 3 PAI task.</td>
<td>129</td>
</tr>
</tbody>
</table>
# List of Tables

3.1 Test results on Inference queries ........................................ 105
3.2 Test results - length 3: A-B-C ........................................... 106
3.3 Test results - length 4: A-B-C-D ........................................ 106
3.4 Test results - length 4: A-B-C-D-E ..................................... 107
3.5 PAI - Ablations - sequence of length 3: A-B-C ....................... 110
3.6 Undirected graph - shortest path ........................................ 111
3.7 Undirected graph - second node analysis ............................... 112
3.8 bAbI - joint training ..................................................... 112
3.9 bAbI - Ablations .......................................................... 113

A.1 Supervised learning hyperparameters. ................................. 126
A.2 Hyperparameters of all the agents presented. ......................... 126

B.1 bAbI Results - average over 5 hyper-parameters with lower loss on 
   the validation set ....................................................... 130
B.2 Fixed hyper-parameters used across tasks ............................ 131
B.3 Range of hyper-parameters used in sweeps ........................... 131
B.4 Hyperparameters used on all tasks trained with DNC. ............... 131
B.5 Hyperparameters ranges used to search over with DNC. .......... 132
B.6 Hyperparameters used for all experiments for UT. .................. 132
B.7 Hyperparameters ranges used to search over with UT. .............. 132
Chapter 1

Introduction

1.1 Cognition

Our world is inherently complex, and all forms of life are constantly competing against each other for scarce resources [1]. Despite the astonishing complexity of life on earth, virtually all creatures are stuck in the same loop. First they observe a state of the world, then they process the information to learn about that state, next they decide which action to take to maximise some measure of success, and finally they observe the new world state consequent to that action. Then this loop repeats, tirelessly.

Cognition can be seen as the characterisation of this loop, and in this vein it could be defined as the mechanism that processes sensory information to perform goal directed behaviours. This definition is intriguing, because it does not exclude any creature, instead it sees cognition as a continuum where the possession of particular cognitive ability depends on the information processing capacity of an agents, being it biological or artificial. One important term in the definition is information, which is a non physical quantity [2], and so this implies that its content can be handled by different physical substrates (e.g. brains, computers), with the same resulting outcome. This is particularly important in the context of this work, where we will take the endeavour of using recent techniques developed in artificial intelligence to investigate how the brain supports navigation and inferential reasoning, two key aspects of animal cognition. However, before diving into the details of our
The idea to equate the mind to an information processing system lies at the centre of the cognitive shift that happens in the field of psychology in the 1950s, and that gave rise to the field of cognitive science [3]. At the beginning of that decades, the ideas of Norbert Wiener’s on cybernetics [4] were gaining popularity, but it was in the 1956 that a collection of ideas and findings started to shape the study of cognition in ways that are still relevant today. In particular, in that year Claude Shannon and John McCarthy edited a book that investigated the possibility of designing a machine that could simulate a brain [5]. Also, Marvin Minsky started to circulate a report that laid the foundations of the field of artificial intelligence [6]. Moreover, at about this time, the term "cognitive strategies” first appeared in the book "A Study of Thinking” from Jerry Bruner, Jackie Goodenough and George Austin [7]. In concert, critical work, showing the computational limits of the human mind, was published by George Miller [8]. Finally, Noam Chomsky applied similar ideas in linguistics, effectively revolutionising our understating of language [9].

All these approaches were influenced by the work of Kurt Gödel, Alan Turing, Alonzo Church and John von Neumann in the 1940s and 1950s, which defined the basic theories of computation that eventually gave rise to the invention of digital computers. Nowadays, the computer metaphor referred to as the von Neumann computer architecture seems outdated, but the idea that the brain is a computer is not. Indeed, a computer is just a device that can compute many different computable functions. Our brain is one such device, so brains are literally computers. These consideration are not merely superficial, instead they emphasise that, to understand the link between brain and behaviour, we need an approach that can described both the representations and computations capabilities of the mind (the functions), and how these are grounded into the structures and functions of the brain (i.e. the device). Only by following this path we can aim for a general theory of brain function. However, given the complexity and the breath of this venture, it rapidly became evident that a framework was required to validate the computational models. It was the British computer scientist and psychologist David Marr, who defined perhaps
1.2. Understanding complex systems

It is widely accepted that brains constitute one of the most complex and adaptive machine discovered by evolution [10]. They are characterised by various forms of nonlinear activation [11, 12, 13], they can adapt quickly [14, 15], and they can perform massive amount of parallel computations [16], at different time scales [17]. Interestingly, all these features effortlessly dance together to give rise to cognition, in the forms of perception, memory, attention, learning, language, emotions and motivations, to mention a few. Understanding how these aspects of the mind relate to behaviour will likely require a different approach from the reductionist one which is too often used in neuroscience [18, 19, 20], as reducing the problem to a mere description of the brain physiology could only give a partial view of the whole picture.

This point was clearly made by Marr [21], who proposed three different levels of understanding for complex system:

- level 1, the computation, this defines the goal achieved by the system - e.g. navigating from A to B;
- level 2, the algorithm, these are the rules implemented by the system to achieve the goal - e.g. calculating the shortest path between the 2 points;
- level 3, the implementation, this is the physical substrate - e.g. how neural firing supports the calculation of the shortest path.

The intuition behind this scheme is that by only looking at the implementation it could be too difficult, if not impossible, to deduce the algorithm that the brain implements to successfully achieve a specific goal. In particular, Marr was objecting that by only looking at neurophysiological data it could, at best, be possible to describe some properties of the neurons. But, on its own, a mere description of the
1.3 Artificial neural networks

1.3. Artificial neural networks

The brain is a neural network formed by many units, the neurons, connected to each other by synapses. Typically biological neurons have four main components, the cell, the body, the axons and the dendrites. They come with different morphologies and functions and are modulated by a specific set of neurotransmitters. Each neuron is part of a complex network of neurons with recursive and inhibitory connections. This is just to mention few properties of biological neurons to highlight the fact that they are more complex than artificial neural networks (ANNs), which instead are a highly simplified abstraction of their biological counterparts. ANNs are composed of computational units (akin to neurons) connected to each other by weights (akin to synapses) which define the strength of these connections. In this section we will first describe two classes of ANN, feed forward and recurrent networks, then the algorithm used to train them, and the possible frameworks under which ANNs can
1.3. Artificial neural networks

Figure 1.1: Feed-forward Artificial Neural Networks

a. The equations for the forward pass in an ANNs with two hidden layers and one output layer. For each layer in the network, each unit calculate the weighted sum of all the incoming connections from the layer below (biases are omitted for simplicity). Then the weighted sum is fed through a non-linear activation function $f(.)$, such as the sigmoid function or the rectified linear unit (ReLU) b. The equations for the backwards pass. At each layer the error derivative with respect to each unit is computed by calculating a weighted sum of the error derivatives with respect to all the connections entering that unit from the layer above. This is the error derivative with respect to the output, $\frac{\partial E}{\partial y}$, which is then converted to the error derivative with respect to the input, $\frac{\partial E}{\partial o}$, by multiplying it by the gradient of $f(o)$, $\frac{\partial y}{\partial o}$.

be trained. Finally, will introduce the history of the field, and highlight how deep learning and reinforcement learning share roots with neuroscience.

1.3.1 A brief description of Artificial neural networks and backpropagation

Artificial neural networks are formed by a large set of units connected together by learnt weights. The units composing an ANN are normally of three different categories. Firstly, input units receive the external information that needs to be processed; then, hidden units perform the computations, and finally the output units compute the answer (see Fig. 1.1a). Originally ANN where composed of one, or maximum two layers of hidden units, these are the so called shallow networks. The problem with shallow networks is that the functions they can approximate are extremely simple and so researchers had long been interested in finding a solution to
1.3. Artificial neural networks

train very deep networks that could discover good hidden features needed to model complex input-output problems, such as visual and speech recognition. However, by the 1980s hand-engineering features had become the standard in many fields of artificial intelligence, since experts knew from empirical studies which features were important to solve a task. Nevertheless, hand-engineering successful features requires a lot of knowledge and practice, and makes it difficult to scale these systems to real world tasks, due to the difficulty of knowing a large set of good features.

Interestingly, since the early days of artificial intelligence research the aim of researchers has been to replace hand-engineered features with learnt ones. Despite its simplicity, it almost took another twenty years before several groups of independent researchers found the solution to train very deep neural networks (DNNs), the backpropagation algorithm [22, 23, 24, 25]. Backpropagation is a general purpose learning algorithm based on stochastic gradient descent that requires two central elements. The first is an objective function, that is, a smooth mathematical formulation used to evaluate the quality of the solution. The second is the need for the layers in the network to be smooth functions of their inputs and internal weights. The smoothness of both components is required because backpropagation is nothing else than a practical application of the chain rule of derivatives, hence the need for the functions to be differentiable [26]. The reason for the name backpropagation comes from the fact that the algorithm calculates the gradients going backwards down the network. Gradients are the multi-variable derivative of the loss function with respect to all the network parameters. Specifically, it starts with the calculation of the errors between the objective function and each of the output units, then it proceeds backwards down through all the layers to the input units (see Fig. 1.1 for an example of forward and backward pass with the corresponding equations). The key aspects is that the calculation performed for one layer are reused in the layer before to allow an efficient flow of the errors with respect to the weights of each layers. Critically, each parameter (weight) is adjusted in relation to its contribution in reducing the error (see Fig. 1.1b). The idea behind this gradient descent technique is very general and can be seen as a way to optimise the behaviour of the network in
1.3. Artificial neural networks

The recurrent network receives an input $x_t$ at time $t$, which is then fed through a set of parameters $U$ into the hidden layer, $h$. The hidden activation $h_t$ are calculated as $f(Ux_t + Wh_{t-1})$, where $W$ is another set of parameters, $f(.)$ is the activation function. In this way the network has a memory of the computations performed at the previous time steps. Finally, the output, $o_t$, is calculated as $g(Vh_t)$, where $g(.)$ is the activation function and $V$ a set of parameters. The same $U, W, V$ are used at each time step (biases are omitted from this figure for clarity).

Relation to its experience. The technique can also be extended to the reinforcement learning domain (see below).

1.3.2 Feed-Forward and Recurrent Networks

There are essentially two types of ANN: feedforward (see Fig. 1.1a) and recurrent networks (see Fig. 1.2). In the former, an input of fixed dimension (e.g. an image) is presented to the input units, which then send their activation values through the first hidden layer. Each hidden unit then calculates a weighted sum of its inputs, where the weights define the strength of connections between the units. Weights can be both positive and negative, resembling, respectively, excitatory and inhibitory synaptic connections. Then the activation value of each unit is typically passed through a non-linear function before being sent as input to the next layer. Finally, the last layer computes the output, e.g. the probability that the input image corresponds to a certain class. The purpose of the non-linear activation function is to allow the network to learn to partition the solution space in ways that allow a complex input-output mapping. This aspect is essential to model high-dimensional data, such as images, video, audio. Finally, non-linear activation have also practical implications, mainly allowing a better propagation of gradients in deep networks due to their...
1.3. Artificial neural networks

The second type, recurrent neural networks (RNNs), are normally used for tasks that have sequential, rather than fixed size inputs. This aspect is particularly important as many real world tasks, like speech and language recognition, are sequential in nature. RNNs are called recurrent because the same network is applied to each element of the input sequence, and they employ recurrent connections that allow the hidden state to have a memory of the computation performed for each previous item in the sequence. To train RNNs, the networks are unfolded in time and each output is considered as a discrete time step; the gradients are then calculated using backpropagation-through-time, which provides a means to deal with the problem of temporal credit assignment [28, 29, 30] (see section 1.5.1 for more details on RNNs and their relation to working memory).

1.3.3 Frameworks used to train Neural Networks

One final distinction that helps organising ideas in the field of ANNs is the learning paradigms used to train them. The most commonly used form of machine learning is supervised learning, where the training data come as a collection of input-label pairs \((x, y)\), and the goal of the algorithm is to produce a mapping \(f(x)\), that makes a prediction, \(\hat{y}\). Then an objective function that measure the error between \(y\) and \(\hat{y}\) is computed, and the parameters of \(f(x)\) are modified to reduce this error. The final goal of such systems is to correctly classify the class labels for unseen inputs, that is, \(f(x)\) should be able to generalise from the training data to unseen data. One classic example of supervised learning is image classification, where the goal is to take as input an image, \(x\), and produce as output, \(\hat{y}\), the probability that the image belong to a certain class, \(y\). There are several forms that \(f(x)\) can take, such as decision trees, support vector machines, random forest, but for the scope of this work only ANNs will be considered. Supervised learning has been the most successful method in AI over the last few years, nonetheless collecting labelled data is expensive, and it is well know that most of human learning is unsupervised [26].

To overcome these limitations, one option is to employ unsupervised learning, which uses unlabelled examples to discover structure in the data. The assump-
tion made in this case is that it is possible to find a low-dimensional manifold that summarises the structure shared by the data. In the ANNs space, one common example of unsupervised learning technique is represented by autoencoders [31, 32]. These networks are essentially trying to learn to copy their inputs to their outputs. Typically a number of constraints are imposed, namely limiting the number of hidden units, such that autoencoders can learn to compress the input into a lower-dimensional code which summarises the structure in the data. This code is normally referred to as the latent-space representation. Autoencoders are often described as generative models, as they can use the learnt density representation to generate new samples [33]. Among others, one influential model in this class, is the variational autoencoder (VAE) [34], which uses an encoder to infer the latent-space representation and a decoder to reconstruct the inputs and generate new samples.

One final class of methods is represented by reinforcement learning (RL), which can be seen as a control-theoretic framework for agents interacting with an environment. The goal of the agent is to learn a policy that, for any given state, produces an action which maximises the sum of expected discounted rewards [35]. RL has been recently combined with deep neural networks (DNNs), giving rise to the field of Deep Reinforcement Learning [36, 37]. Interestingly, neuroscience and psychology were critical for the emergence of the RL field [38], as this framework was derived from findings in the animal learning literature, where learning is driven by the maximisation of a certain measure of progress, like in supervised and unsupervised learning. In particular, one aspect of RL, Temporal Difference learning, was inspired by research on dopamine neurons in the ventral tegmental area and substantia nigra. These studies showed that neurons in those areas track the reward prediction error, that is the difference between the reward at any given time step and the future reward received [39, 40], and when a stimulus reliably predict a future reward then the calculated TD error is used to associate the stimulus with the future reward (second-order conditioning) [41]. Temporal difference learning lies at the core of recent advances in RL, such as the ability of deep reinforcement learning algorithms to beat professional human players in challenging board games.
Interestingly not just RL, but also ANNs have strong ties with work done in psychology and neuroscience, so in the next section we will review how the field of deep learning is directly connected with research done in cognitive science.

1.3.4 The longstanding relation between Neural Networks and Neuroscience

As above mentioned, at the birth of the modern computer age, the field of AI was tightly connected with psychology and neuroscience. The terms artificial neural networks itself points explicitly to neuroscience, and it was used to show how logical operations in electrical circuits could be used to model biological neurons [44]. Subsequently, Donald Hebb proposed an unsupervised learning rule to update the weights of ANNs, whereby the connections between neurons are changed based on their concurrent activation [45]. From equation 1.1 it is possible to see that the connection between neurons \( i \) and \( j \), \( w_{i,j} \), is updated as a function of the current output, \( x \), of both \( i \) and \( j \), weighted by a learning rate coefficient \( \lambda \):

\[
 w_{i,j}^{t+1} = w_{i,j}^t + \lambda x_i^t x_j^t 
\]  

(1.1)

It was roughly a decade later that single layer perceptrons, the basis of modern ANNs, were introduced together with the rule to train it via supervisory feedback [46]:

\[
 w_{i}^{t+1} = w_{i}^t + r \cdot (d_j^t - y_j^t)x_{j,i}, 
\]  

(1.2)

for all units \( 0 \leq i \leq n \), where:

- \( w_i \) is the \( i \)th value in the weight vector, to be multiplied by the value of the \( i \)th input feature
- \( r \) is the learning rate
- \( d_j \) is the desired output for the input \( x_j \)
- \( y_j \) is the current output
1.3. Artificial neural networks

- $x_{i,j}$ is the value of the $i$th feature of the $j$th training input vector
- $n$ is the number of units in the layer

At that time AI research was dominated by the classical theory that saw cognition as a series of manipulations made to symbolic representations similar to the ones performed by computers [47]. More recently, 'symbolic AI' has been challenged by the alternative view that the brain uses distributed representations which are processed in parallel. In particular, the parallel distributed processing (PDP) movement suggested that cognition could be explained by flexible and distributed connections between the units of ANNs, which are learnt in an iterative fashion with the goal of maximising an objective function [48]. Connectionist models based on simple ANNs were successfully used to model several aspect of cognition such as, language [e.g. 30, 25, 49], attention [e.g. 50, 51], motor control [52], memory [e.g. 53, 54], vision [e.g. 55, 56, 57] and neurospychology [58]. Interestingly, not only were these models able to mimic the input-output pattern observed in experiments, but the behaviour of the artificial units resembled biological neurons [59].

The issue about representations is at the centre of the debate between logic and neural network based AI models for understanding cognition. The former builds on the naive idea that information is stored in local representations, that is each neuron is coding for a specific pattern. The latter assumes information to be stored in a distributed fashion, specifically patterns are represented by the activity of several neurons distributed across large parts of the cortex. It has been shown that one of the main advantages of distributed representations is their ability to generalise beyond the training examples much more easily than local ones [60, 61]. Distributed representation, unlike symbols, are likely to be more robust to damage or silencing of units [62]. Moreover, the simple rules learnt by distributed representations can be reused exponentially often as a function of the number of layers in the network [63]. These aspects of connectionist models lie at the heart of current deep learning techniques that are the de-facto standard in most of modern machine learning applications. For instance, current models of natural language processing are typically based on the idea of learnt word vectors [64]; that is, in the first layer of the
model each word is transformed into a vector, then the following layers learn how
to convert this vector into a set of probabilities to predict the following word in a
sentence. It has been proven that these vectors learn to code each word as a set of
distinct features that together represent the word itself [65]. Critically, these fea-
tures were not present in the input of the network, instead the learning procedure
found them to be a good basis set to factorise the input space in ways that maximise
the final objective. Distributed vector representation, such as these, are now the
most used embeddings in language modelling [66, 67, 68].

Developments in experimental and theoretical neuroscience has also shaped
progress in AI. For example, convolutional neural networks (CNNs), one of the
most successful AI architecture of recent years[69], were directly inspired by ex-
periments showing how visual input is filtered and pooled in the mammalian visual
system, forming a cascade of subsequent layers [see 70, for a thorough explanation].
Furthermore, by analysing the activation of modern deep learning architectures, it
has been shown that they replicate the hierarchical organisation of mammals vi-
sual cortex [71], where successive non-linear units transform the raw input into a
progressively more abstract set of representations that capture the relevant factors
of variations in the data by leaving out the noise [72, 70]. Thanks to the advent of
more powerful hardware, mainly GPU, new ways of pre-training very deep network
[73], large datasets [74], and new regularisation technique [75], CNNs are now used
widely used for processing images [e.g. 76], video [e.g. 77], speech [e.g. 78], and
audio [e.g. 79].

Neuroscience continues to provide a source of inspiration for AI researchers
looking to develop more capable models, different aspects of cognition such as
vision, language, attention, memory and imagination, being particularly fertile
ground. However, a thorough review of all these topics is outside of the scope
of this work. Instead, we focus on recent attention and memory models, as these
will serve as base knowledge for the following chapters.
1.4 Attention

Neural networks are generally characterised as universal function approximators – they can estimate any arbitrary functions with any arbitrary precision [80]. However, it is well understood that this is the case only in the limit of infinite capacity, while in practice there are limitations due to the number of hidden units that one can train. To overcome these limitations, and so increase the expressivity of ANNs, one possibility is to use attention mechanisms [81], which mirror the ability of brains to focus only on a subset of the inputs while ignoring the rest. Attention is an area of active research in cognitive science, and the predominant view is that it can be divided into three functional components: alerting, orienting, and executive attention. Alerting is the ability of maintaining active vigilance during the performance of a certain task. Orienting is the capability to prioritise a specific sensory input. Finally, brains are normally able to process different streams of inputs, but when interference arises, then executive attention mediates these conflicts. Executive attention is assumed to reside in higher parts of the brain, like prefrontal cortex (PfC), and it is thought to be the only one involving awareness [82].

The current deep learning exploration of attention started by taking inspiration from human visual attention, and in particular from our ability to focus on a specific region of the visual field with “high resolution”, while perceiving the surrounding image in “low resolution”, and iteratively adjust the high resolution focal point to do better inference [83]. This idea of paying attention only to a subset of the sensory input has been applied in deep neural networks by employing a learnt vector of importance weights that it is used to mask the input (this vector is normally called head). This attention mechanism was introduced in machine translation, to help reducing the long range dependencies issue faced by sequence-to-sequence (S2S) models [84]. In this framework an input sequence in a certain language is fed into a decoder that generates a context vector used by a decoder to translate the original sequence. However, if the input sentence is too long, then the context vector tends to forget the earlier parts of the sequence once it has processed the entire the sequence. Attention creates a set of shortcut connections between the context vector and the
relevant part of the inputs, even if these occur several steps back in the sequence [67]. Subsequently, similar forms of attention were then employed in networks performing vision tasks. Here the attention was used to extract glimpses that were particularly useful to solve the task [85]. Similarly attention was used to perform image-to-caption generation [86], and in the context of generative modelling in both variational autoencoders [87] and adversarial training [88].

More recently, the most prominent use of attention in ANNs has been ‘self-attention’, where elements of a single sequence are compared to each other to generate representations of the sequence itself [89]. This approach has been extended to fully feed forward networks in a model called 'Transformer' [90]. Here the combination of multiple self-attention heads with layer normalisation [91] and residual connections [92], resulted in state-of-the-art performance across a large spectrum of language task, such as next word prediction [68], and text generation [93]. Also, self-attention was recently successfully implemented in multi-agent reinforcement learning [94]. The main advantage of the ‘Transformer’ model resides in its ability to select relevant information from the inputs by performing an highly parallel all-to-all comparison between the elements of the sequence. Nevertheless, the ability of self-attention to model long-range dependencies has been recently been questioned [95], as its quadratic complexity in the input length, make the model almost intractable for long sequences. An alternative attention schema is represented by dynamic lightweight convolution that, by reusing the same weights for each contextual element independently from the sequence time-step, reduce the amount of computational complexity while maintaining the same performance as self-attention [96].

All the aforementioned forms of attention can be seen as a subset of a large class of models that go under the umbrella of multiplicative interactions. These frameworks have also been recently proposed as a means to fuse multiple streams of data and to perform conditional computation [97]. Under this view, multiplicative interactions can be seen as a first attempt to model executive attention, but more work is needed in this direction.
1.5 Memory

Learning, a central topic in modern AI, can be defined as the process of acquiring knowledge about the world, or in other words, the process of building memories. Decades of psychological studies have sought to characterise memory and understand that processes that contribute to it. In neuroscience, a common distinction can be made between short-term memory, working memory and long-term memory [98].

Short-term memory is the storage of a limited amount of readily available information for a brief delay. For instance, to understand the meaning of a sentence, it is important to keep in mind the beginning of it while processing the rest; likewise, to perform a subtraction, it could be important to remember the carrying-over number. These are examples of task supported by short-term memory, which it is believed to store pointers to knowledge that resides in other part of the brain, rather than complete concepts [99].

Short-term memory forms the basis for the working memory system, a mental workspace where information from short-term memory and long-term memory is retrieved and manipulated to support complex tasks like reasoning, learning and decision-making. Working memory is believed to depended on the interactions between a central ”executive controller” and three distinct, domain specific, buffers: the phonological loop, the visual sketch pad and the episodic buffer [100, 101].

Finally, long-term memory is a collection of systems that support the ability to retain information over long periods of time. Long-term memory can be further split into explicit and implicit memory. The former can be intentionally retrieved, and can either be related to personal events – episodic memory – or to general facts about the world – semantic memory [102, 103, 104]. In contrast, implicit memory (or non-declarative memory) is accessed without awareness, and it is reflected through performance rather than explicit recollection. One form of implicit memory is called procedural and enables us to carry out commonly learned tasks without consciously thinking about them, like writing or riding a bicycle. Implicit memory can also come about from priming, the process by which a past experience increases the accuracy or quickness of a response. For instance, if a word has been heard very recently, or many more times than another one, then that
1.5. Memory

A word is more likely to be retrieved more quickly. Another form of implicit memory is conditioning, where the association of one thing with another is unconsciously learned [105].

Episodic memory is often defined as the capacity to remember specific events with just a single exposure (i.e. one-shot); experiences are stored within the medial temporal lobe, and the hippocampal system in particular, by remembering what, where and when an event happened [106]. The retrieval of these episodes takes place through so called "mental time travel" [103] allowing an individual to go back in time and re-live a specific moment but equally can be used to travel forward to plan for future events. For instance, one could remember a chat with a friend the night before at the pub, and agreeing to help painting the friend’s new flat (travel back); this memory can then be used to decided what to wear and what tools are appropriate to help painting (travel forward).

The accumulation of separate events in memory is thought to be the main input to a consolidation process that gradually extract relevant facts and features into a more general representation, the above mentioned semantic memory (even though consolidation is not restricted to this cf. [107]). Albeit the precise nature of the interactions between episodic and semantic memory remains unclear, it has been observed that patients with amnesia, related to damages in the hippocampus, struggle to form new general knowledge, even if they maintain the previously acquired one [103, 106], thus confirming the need of episodic memory to form new knowledge (even though it is not clear cut as there has been some debate on what memories can become semantic [e.g. 108]).

1.5.1 Working Memory

Recurrent neural networks (RNNs) are a class of models that essentially address as similar problem to those solved by working memory - processing sequential data. (see 1.3.1 for more details). The initial work in this area was performed by Hopfield, who designed RNNs formed of fully connected nodes, with symmetric weights and binary outputs (either 0,1 or -1,1). When presented with a new pattern the network runs for a certain number of steps, defined by the following update rule:
\[ s_i = \begin{cases} 
+1 & \text{if } \sum_j w_{i,j} s_j > \theta_i, \\
-1 & \text{otherwise.} 
\end{cases} \] (1.3)

where

- \( w_{ij} \) is the strength of the connection weight from unit \( j \) to unit \( i \) (the weight of the connection),

- \( s_i \) is the state of unit \( i \), \( s_j \) is the state of unit \( j \),

- \( \theta_i \) is the threshold of unit \( i \)

In this process the nodes of the network get updated and the process eventually converges to a stable state and at that time a pattern is read-out. This networks are useful to recover stored pattern from degraded inputs [109]. Subsequently, RNNs for supervised learning were introduced by Jordan [110], this was effectively a feed forward network with a single hidden layer but was equipped with state units. This latter units, which were self-connected, fed back the value of the output units at the following time step, thus making the network recurrent. Interestingly, this way of using the output as new input is also at the basis of modern machine translation architectures [84]. Finally, Elman [30] introduced the equivalent of modern RNNs, where each hidden unit had a single recurrent connection. Interestingly, some theoretical work on this class of models proved that fixed-sized RNNs, when equipped with nonlinear activation function, can simulate a universal Turing machine [111].

Despite the potential of recurrent neural networks, learning long-range dependencies with such models has proven to be difficult due the problems of vanishing and exploding gradients [112, 113]. These problems are related to gradient based learning, where the value of a weight in the network is adjusted based on its contribution to the network’s output (see section 1.3.1 for details on the backpropagation learning rule). If a change in the weight’s value causes very small change in the network’s output then the network is unable to learn the parameter effectively, this is what is referred to as vanishing gradient. On the opposite, exploding gradient is a problem due to large error gradients which accumulate over time and result in very
1.5. Memory

large updates to the neural network weights, thus resulting in saturation of the network. In the context of recurrent networks, both problems have been mitigated with the Long Short-Term Memory (LSTM) [114] network. LSTMs replace the units in the hidden layer of traditional RNNs with a memory cell, which contains a set of gated nodes. These self-connected nodes can learn to create paths through time that ensure that gradients flow without vanishing or exploding. In its initial version, the weights of the self-connected nodes in the memory cell were fixed, this limitation was addressed by making these weights conditioned on the hidden state produced by network at the previous time step [115], thus making LSTMs more powerful in dealing with longer sequences. Alongside LSTMs, a further advancement was made with the Bidirectional Recurrent Neural Network [116], which is able to access both past and future elements in the sequence to determine the output. The use of LSTMs and bidirectional recurrent neural networks has been extremely successful in several application, such as phoneme classification [117], handwriting recognition [118], speech recognition [119], machine translation [84], image captioning [86] and syntactic parsing [120]. More recently, a simplified version of the LSTM, the gated recurrent unit [121], uses a single gate to decided when to forget and to update the content of the cell. This architecture has shown performance that in most cases matches the LSTM framework, so rising doubts about which elements of the architecture are needed. However subsequent investigations [122, 123] of several variants of the LSTM and gated recurrent unit reached the conclusion that learning to forget is essential, but did not find a conclusive answer which architectural components are universally necessary, leaving this research question open to future investigations.

A further line of work that is worth mentioning in connection with the neuroscience literature is the so-called fast weights [124], which take inspiration from short term potentiation of biological neurons [125, 15]. With fast weights each weight of a NNs is augmented with another more plastic one that is generally trained with the Hebbian learning rule. This new weight can grow and decay rapidly as a function of the current inputs, and so store traces of the recent past that are useful
to solve sequential memory tasks. More recently, two concurrent papers [126, 127] augmented RNNs with fast Hebbian weights, allowing the networks to attend to the recent past with better performance than traditional LSTMs, without the need to explicitly store copied of the neural activation.

1.5.2 Episodic Memory

Several aspects of episodic memory have been introduced to overcome two main limitations of NNs, catastrophic interference and slow learning [128]. Catastrophic interference occurs when learning of new information cause the network to abruptly forget previously learned information [129, 130, 131]. This problem is related to the fact that using correlated data to train NNs generates interference between new and old knowledge. In particular, any gradient descent technique, like backpropagation, assumes samples to be uncorrelated, otherwise error estimates will be biased, resulting in the weights being updated all together in the same, presumably, wrong direction. However, the issue of correlated data arises often in situations where the data used to train NNs are collected online and then used in batches (e.g. time series, reinforcement learning). To overcome this limitation, one possible solution is to store the data in an episodic buffer, and then re-sample the batches such that data are uncorrelated. Episodic buffers in RL have been used since the early days of NNs [132], and they have been a key ingredient in the success of the deep Q-network (DQN), which learned to play the vast majority of the Atari 2600 games directly from pixels at a super human level of performance [36]. Specifically, the buffer used in DQN, called experience replay, was used to store games’ states as they were experienced under the current policy. These states were then re-sampled offline, and gradually used to adjust the parameter of the NNs towards a policy that better approximates the optimal Q-value. The use of experience replay was critical to both increasing data efficiency and avoiding correlated data. The same technique has then been extended by prioritising the replay of highly rewarding trajectories [133], and the construction of fictitious trajectories, where an agent was rewarded for reaching an imagined goal [134].

The second problem, slow learning, is again related to the parametric nature of
NNs where the optimal value of each connection is a function of the values of all the others. This interdependence is especially critical at the beginning of training, when there is virtually no structure in the weights ensemble, and so the error signal coming from the gradients calculation is weak and noisy. In this scenario, it has been shown that adjusting the behaviour of an agent by using single experiences stored in an episodic like memory, a process called episodic control [135], is beneficial in the case of complex tasks with high inferential noise and especially at the beginning of learning, when the statistical knowledge provided by the parametric system has not set in yet [136]. The idea of episodic control has been used recently in deep reinforcement learning [137, 138], where an agent was furnished with an episodic buffer to store visual observations together with the actions taken and the associated reward. The agent was then trained to select a novel action based on the similarity between the current visual observation and the one stored in memory, by critically taking into account the reward observed before. By using such a training regime the agent was able to substantially increase data efficiency and reach much higher scores on specific Atari games which had proven to be hard for simple reactive algorithms like DQN. This success was mainly due to the ability of the algorithm to perform one-shot learning [138], in a vein similar to the one supported by the hippocampus [135]. A similar approach was also investigated in the domain of supervised learning, where a network was trained to classify images rapidly from only few examples, by employing an episodic buffer where a set of common representations for each ImageNet class were stored, and then used to match new instances using cosine similarity [139].

1.5.3 An hybrid approach: Neural network augmented with external memory

In recent years there has been increasing interest in the development of models that combine ANNs controller with an episodic memory storage, an approach aligned with the neuroscience of memory explained above. For instance, the Neural Turing Machine employed a recurrent neural networks (RNNs) to learn how to write and read from an external memory matrix [140]. To overcome some initial limitations,
like the interference of items in memory, Neural Turing Machine was extended with the inclusion of memory usage statistics, in the so called Differential Neural Computer [141]. A more recent extension incorporated sparsity into the Differential Neural Computer [142], allowing the model to perform similarly to the original Differential Neural Computer, but employing a fraction of the computation. However, despite its ability to solve complex reasoning tasks, Differential Neural Computer proved to be sometime difficult to train. Another architecture, termed Memory Networks, avoided some of these limitations by using a read-only memory storage that scaled with the size of the inputs [143, 144], and so avoiding the problem of learning to write. This approach was also successfully used to tackle some complex navigation task in DRL [145]. Since these initial architectures were introduced, a number of alternative have been developed. For instance, the Dynamic Memory Network [146] shares some similarities with Memory Networks, but instead of using a pre-filled memory buffer it works with sequential inputs, thus making the method more general. The Working Memory Network [147] is again heavily based on Memory Networks, but it integrates a working memory buffer and the ability to explicitly perform relational reasoning using a RelationNet [148]. Finally, the Recurrent Entity Network [149] has analogies with the DNC, but by employing a parallel architecture it exploits simultaneous updates across several memory locations. However, all these models are based on the same differentiable attention read-out function [67], or analogous variants [150] that, despite several success, suffers from non trivial scaling issue, thus making this approach difficult to generalise to useful scenarios.

### 1.6 Using ANN to model Cognition

Throughout this chapter we have seen how ANNs have been developed and used to model several aspect of cognition. We described how, in the last ten years, there has been a renewed interests for these models in the machine learning community. The field of DL evolved from the one of parallel distributed processing (PDP), and from there it went as far as improving state-of-the-art models in a wide range of subjects, from visual [76] and speech recognition [67] to hard scientific problems, such as
1.6. Using ANN to model Cognition

protein folding [151] and genomics [152].

On these premises we argue that it is time for a renaissance of the use of ANNs as a general tool to understanding the link between brain and behaviour. It would be elegant to find a single equation that can explain how the brain functions, but this is probably a naive hope. Indeed, the brain is not a single neural network that implements the same optimisation algorithm, but it is composed of several modules interacting to each other [153, 154]. So if the aim is to understand the function, the algorithm and, up to a certain extent, the mechanisms through which these different modules give rise to cognition, then ANNs should be our choice as a modelling tool. The reason is threefold. First, ANNs do not require any domain specific knowledge. ANNs are able to discover the structure in the data by only using domain-general biases derived from properties of the learning procedure and network architecture. The approach is so general that even the learning procedure can be learnt [155], as well as the architecture [156]. Second, ANNs represent a systemic approach that keep all the levels of analysis into account simultaneously [157], whereas other models (e.g. probabilistic [158]) often take a top-down approach. Finally, ANNs do not make any assumption regarding the agent that generates the data, they are only concerned with mimicking the input-output behaviour observed experimentally. This is especially interesting now, considering the huge growth in both data generated by experimental neuroscience and the capability of hardware needed to process it.

The aforementioned points will be more generally expanded in the discussion, but to experimentally motivate our argument we will devote the next two chapters to implement this idea in the domains of spatial navigation and inferential reasoning, two critical aspects of animal cognition. Chapter 2 will outline how DNNs can be used to model spatial memory, and in particular to characterise the role of grid cells [159] in supporting spatial navigation. Then, Chapter 3 will focus on episodic memory, and in particular how, by looking at the hippocampus, we can gain inspiration to design DNNs augmented with external memory that can successfully perform hard inferential reasoning tasks [160].
Chapter 2

Understanding Spatial Navigation using Deep Learning

The work presented in this chapter was previously published as [161]. The current text is based on the published manuscript but has been expanded and elaborated upon.
2.1 A Brief Outline of the Neural Mechanisms of Self-Localisation

Navigation is generally defined as the process of calculating and following a trajectory from one point to another [162], a mechanism known to be supported by two distinct strategies. The first involves local navigation, and happens when the goal to be reached is directly perceivable from the current position. The second requires wayfinding and is used when a goal lies beyond the immediate limit of perception - hence it requires the recollection of known routes or, more generally, a cognitive map [163]. Wayfinding necessarily requires some degree of self-localisation - the ability to place oneself within an environment.

Interestingly, the last fifty years of neuroscience research have identified neural populations in the hippocampal formation of the mammalian brain which encode information about the position and orientation of an animal relative to the environment. These consist of place cells, which are active when an animal traverses a specific location within the environment, irrespective of orientation [164, 165, 166]; head direction cells which fire when the animal’s head is at a specific orientation [167]; grid cells, which present periodic firing fields arranged in a triangular array [159], border cells (or boundary vector cells) which fires when an animal is at specific combination of distance and allocentric orientation from a boundary [168, 169], and egocentric boundary vector cells [170].

Place cells, identified in areas CA1 and CA3 of the hippocampus [164], exhibit varied firing locations and field shapes. Each cell only fires an action potentials when the animals head is within a certain region of the environment, which is called the cells place field [171]. As a population, place cells provide a sparse activity vector that contains enough information to represent the animal’s current location [172]. Place cells are established almost immediately after an animal enters a new environment [172], and they remain stable even after a long delay [173]. However, recent studies using optical imaging techniques, tracking thousands of CA1 pyramidal cells place fields over weeks, revealed that place coding was dynamic. In particular, each day the ensemble representation of the environment involved a
unique subset of cells, with an overlap of about 15% / 25% between any two of these subsets. Other studies shown that place cell firing vectors seem to be effectively decorrelated between different environments, a process called remapping - with individual cells changing their firing locations and rates[174, 175, 176]. Theoretical studies have described remapping in terms of pattern separation, a process that likely depends on dentate gyrus and CA3 [177, 128, 178, 179, 180, 181, 182] (see 3.1 for a thorough explanation of pattern separation).

Head direction cells signal the azimuthal angle of an animal’s head via directionally tuned receptive fields - being maximally active when the animal’s head occupies a certain allocentric orientation spectrum (≈ 100°). These cells have been discovered outside of the hippocampus, in the presubiculum [167], thalamic nuclei [183] and mammillary bodies [184] as well as in the entorhinal cortex [185]. Importantly, between environments these cells maintain their relative angular offset, hence two cell that fire ≈ 45° apart in one environment will do the same in another, even though their absolute firing fields have shifted.

Grid cells, first identified in medial entorhinal cortex (mEC) and predominately in layer II, are distinguished by an interesting periodic pattern of activation composed of multiple firing fields arranged in a triangular lattice tiling the whole environment. Some of these cells also present a specific directional tuning reminiscent of head direction cells [185]. Grid cells are clustered in modules where all cells share the same orientation and scale, being essentially a translated version of each other. Grid scale, defined from the periodicity of pattern, increases in discrete steps, following a geometric progression along the mEC dorso-ventral axis [186, 187].

Empirical data from grid cell recordings have provided fertile ground for computational models seeking to characterise the neural activity. In particular models have centred around two possible mechanisms, oscillatory interference [188, 189, 190, 191] and continuous attractors [192, 193, 194, 195]. In turn, these models which initially focused on the mechanism by which self-motion might be integrated to update allocentric location, have lead to suggestions that grid cells provide a Euclidean spatial metric framework. This suggests that grid
cell networks support the calculation of goal-directed vectors, enabling animals to follow direct routes to a remembered goal, a process known as vector-based navigation\[196, 197, 198\]. Theoretically, the advantage of decomposing spatial location into a multi-scale periodic code, as provided by grid cells, is that the relative position of two points can be retrieved by examining the difference in the code at the level of each scale — combining the modulus remainders to return the true vector\[196, 198\] (see Fig.2.10a). However, despite the obvious utility of such a framework, experimental evidence for the direct involvement of grid representations in goal-directed navigation is still lacking.

The work in this chapter aimed to address this point in two stages. First we trained a deep recurrent neural network to perform a path integration task to investigate whether grid cells could emerge as a consequence of minimising the objective of self-localisation. The network was trained in a virtual square arena of 2.2m\(\times\)2.2m, using simulated trajectories modelled on those of foraging rodents. The network was required to update its estimate of location and head direction based on translational and angular velocity signals, mirroring those available to the mammalian brain \[199, 200, 201\] (see Methods and Fig.2.7a&b). The network used was a Long Short-Term Memory (LSTM) \[114\], which as explained in section 1.5.1 is a model particularly well suited to deal with sequential data of the kind we are interested here. The LSTM received velocity inputs and it was trained using backpropagation through time, allowing the network to dynamically combine current input signals with activity patterns reflecting past events (see Fig.2.7a). Importantly, the network was subject to regularisation, in particular dropout \[202\] and gradient clipping (see Methods 2.2.1.7). The vector of activities in the place and head direction units, corresponding to the current position, was provided as a supervised training signal at each time step (see Methods), following evidence that in mammals, place and head direction representations exist in close anatomical proximity to entorhinal grid cells \[185\] and emerge in rodent pups prior to the appearance of mature grid cells \[203, 204\]. Equally, in adult rodents, entorhinal grid cells are known to project to the hippocampus and appear to contribute to the neural activity of place cells.
Next, to develop an agent with the potential for vector-based navigation, we incorporated the “grid network” described above, into a larger architecture that was trained using deep RL (Fig.2.10d). As before, the grid network was trained using supervised learning but, to better approximate the information available to navigating mammals, it now received velocity signals perturbed with random noise (see methods for details) as well as visual input. Experimental evidence suggests that place cell input to grid cells corrects for drift and anchors grids to environmental cues [186]. To parallel this, visual input was processed by a "vision module" consisting of a convolutional network that produced place and head direction cell activity patterns which were provided as input to the grid network 5% of the time – akin to a moving animal making occasional, imperfect observations of salient environmental cues [206] (see Methods, Fig.2.10b&c). The output of the linear layer of the grid network, corresponding to the agent’s current location, was provided as input to another LSTM (“policy LSTM”) that predicted the sequence of actions to take to reach the desired goal. Additionally, whenever the agent reached the goal, the ”goal grid code” — activity in the linear layer — was subsequently provided to the policy LSTM during navigation as an additional input, such that the ”policy LSTM” had all the information needed to perform vector based navigation.

2.2 Methods

In this section we present the main ideas and models used to develop the grid cells agent. First we present the details of the supervised learning experiments, by detailing the inputs, the network architectures, the objective function and the regularisation techniques. We then define the Deep Reinforcement learning experiments by presenting the environments used, the grid cells agent architecture, the comparison agents, the general training algorithm and the control experiments. Finally we present the neuroscience-based analyses used to characterise the artificial cells.
2.2 Methods

2.2.1 Supervised learning - Path integration experiments and settings

Figure 2.1: Examples of paths used to train the grid network.

a. Examples of 100 paths chained together to show environment coverage. b. Example of three paths of 100 steps each used in a mini-batch

2.2.1.1 Simplified 2D environment.

Simulated rat trajectories of duration $T$ were generated in square and circular environments with walls of length $L$ (diameter in the circular case). Fig. 2.1 show some examples. The simulated rat started at a uniformly sampled location and facing angle within the enclosure. A rat-like motion model[207] was used to obtain trajectories that uniformly covered the whole environment by avoiding walls. Table A.1 displays the motion model parameters.

2.2.1.2 Ground truth place cell distribution.

Place cell activations, $\vec{c} \in [0, 1]^N$, for a given position $\vec{x} \in \mathbb{R}^2$, were simulated by the posterior probability of each component of a mixture of two-dimensional isotropic Gaussians,

$$
c_i = \frac{e^{-\frac{||\vec{x} - \vec{\mu}_i^{(c)}||^2}{2\sigma^{(c)}_i^2}}}{\sum_{j=1}^{N} e^{-\frac{||\vec{x} - \vec{\mu}_j^{(c)}||^2}{2\sigma^{(c)}_j^2}}},$$  \hspace{1cm} (2.1)

where $\vec{\mu}_i^{(c)} \in \mathbb{R}^2$, the place cell centres, are $N$ two-dimensional vectors chosen uniformly at random before training, and $\sigma^{(c)}$, the place cell scale, is a positive scalar fixed for each experiment.
2.2. Methods

2.2.1.3 Ground truth head-direction cell distribution.

Head-direction cell activations, $\vec{h} \in [0,1]^M$, for a given facing angle $\varphi$ were represented by the posterior probability of each component of a mixture of Von Mises distributions with concentration parameter $\kappa^{(h)}$,

$$h_i = \frac{e^{\kappa^{(h)} \cos(\varphi - \mu_i^{(h)})}}{\sum_{j=1}^{M} e^{\kappa^{(h)} \cos(\varphi - \mu_j^{(h)})}},$$  \hspace{1cm} (2.2)

where the $M$ head direction centres $\mu_i^{(h)} \in [-\pi, \pi]$, were chosen uniformly at random before training, and $\kappa^{(h)}$ the concentration parameter is a positive scalar fixed for each experiment.

2.2.1.4 Supervised learning inputs.

In the supervised setup the grid cell network receives, at each step $t$, the egocentric linear velocity $\vec{u}_t \in \mathbb{R}$, and the sine and cosine of the angular velocity of its head direction $\dot{\varphi}$:

$$\dot{\varphi}_t = (\theta_t - \theta_{t-1}) + (\phi_t - \phi_{t-1})$$  \hspace{1cm} (2.3)

$$\vec{u}_{t+1} = v_t \begin{bmatrix} \cos(\theta_t - \phi_t) \\ \sin(\theta_t - \phi_t) \end{bmatrix}$$  \hspace{1cm} (2.4)

To clarify, in this motion model the head of the artificial agent was fixed, that is the agent always faces the direction of travel.

2.2.1.5 Grid cell network architecture

The grid cell network architecture (see Fig.2.2) consists of three layers: a recurrent layer, a linear layer, and an output layer. The single recurrent layer is an LSTM (long short-term memory [114]) that projects to place and head direction units via the linear layer. The linear layer was implemented because it permits the addition of dropout[202]. With Dropout, some number of layer units, in our case 50% (otherwise differently stated), are randomly silenced during the training phase. Dropout has the effect of making the training process noisy, forcing units within a layer to
2.2. Methods

The recurrent layer of the grid cell network is an LSTM with 128 hidden units. The recurrent layer receives as input the vector \([v_t, \sin(\dot{\phi}_t), \cos(\dot{\phi}_t)]\). The initial cell state and hidden state of the LSTM, \(\vec{c}_0\) and \(\vec{m}_0\) respectively, were initialised by computing a linear transformation of the ground truth place cell activity \(\vec{c}_0\) and head-direction cell activity \(\vec{h}_0\) at time 0. The output of the LSTM is followed by a linear layer on which dropout is applied. The output of the linear layer, \(\vec{g}_t\), is linearly transformed and passed to two softmax functions that calculate the predicted head direction cell activity, \(\vec{z}_t\), and place cell activity, \(\vec{y}_t\), respectively. We found evidence of grid-like and head direction-like units in the linear layer activations \(\vec{g}_t\).

\[
L(\vec{y}, \vec{z}, \vec{c}, \vec{h}) = -\sum_{i=1}^{N} c_i \log(y_i) - \sum_{j=1}^{M} h_j \log(z_j)
\]

Figure 2.2: Network architecture in the supervised learning experiment.
time 0:

\[ \vec{l}_0 = W^{(cp)} \vec{c}_0 + W^{(cd)} \vec{h}_0 \]  
\[ \vec{m}_0 = W^{(hp)} \vec{c}_0 + W^{(hd)} \vec{h}_0 \]  

The parameters of these two linear transformations \((W^{(cp)}, W^{(cd)}, W^{(hp)}, \text{and } W^{(hd)})\) were optimised during training. The output of the LSTM, \(\vec{m}_t\) is then used to produce predictions of the place cells \(\vec{y}_t\) and head direction cells \(\vec{z}_t\) by means of a linear decoder network.

The linear decoder consists of three sets of weights and biases: first, weights and biases that map from the LSTM hidden state \(\vec{m}_t\) to the linear layer activations \(\vec{g}_t \in \mathbb{R}^{512}\). The other two sets of weights map from the linear layer activations \(\vec{g}_t\) to the predicted head directions, \(\vec{z}_t\), and predicted place cells, \(\vec{y}_t\), respectively via softmax functions[208]. Dropout [202] with drop probability 0.5 was applied to each \(\vec{g}_t\) unit, which means that at each time step in the sequence 50% units in the linear layer were randomly silenced. Note that there is no intermediary non-linearity in the linear decoder.

### 2.2.1.6 Supervised learning loss.

The grid cell network is trained to predict the place and head-direction cell ensemble activations, \(\vec{c}_t\) and \(\vec{h}_t\) respectively, at each time step \(t\). During training, the network was trained in a single environment where the place cell centres were constant throughout. The parameters of the grid cell network were trained by minimising the cross-entropy between the network place cell predictions, \(\vec{y}\), and the synthetic place-cells targets, \(\vec{c}\), and the cross-entropy between head-direction predictions, \(\vec{z}\), and their targets, \(\vec{h}\):

\[ L(\vec{y}, \vec{z}, \vec{c}, \vec{h}) = -\sum_{i=1}^{N} c_i \log(y_i) - \sum_{j=1}^{M} h_j \log(z_j), \]  

(2.7)

Gradients of (2.7) with respect to the network parameters were calculated using backpropagation through time[209], unrolling the network into blocks of 100 time
steps. The network parameters were updated using stochastic-gradient descent (RMSProp[210]), with weight decay[211] for the weights incident upon the linear layer activations. Hyperparameter values used for training were listed in Table A.1.

2.2.1.7 Gradient clipping
In our simulations gradient clipping was used for parameters projecting from the dropout linear layer, $g_t$, to the place and head-direction cell predictions $\vec{y}_t$ and $\vec{z}_t$. Gradient clipping clips each element of the gradient vector to lie in a given interval $[-g_c, g_c]$. Gradient clipping is an important tool for optimisation in deep and recurrent artificial neural networks where it helps to prevent exploding gradients[113]. Gradient clipping also introduces distortions into the weight updates which help to avoid local minima. Together with the fact that all other local signals used to compute updates in biological networks were bounded, this implies that weight update magnitudes will be clipped at every layer, and thus cannot grow unchecked as they do in artificial networks that often have unbounded backward passes.

2.2.1.8 Generalisation test
While the training of the network was performed in noise-free conditions, the hyperparameters of the network were optimised to maximise performance in conditions where noise was present (testing condition). Specifically two different noise conditions were considered. In the first case Gaussian noise $\epsilon$, with $\mu = 0$ and $\sigma = 0.3$ was applied to the velocities inputs, $\vec{u}_t$ and $\phi_t$. Whereas in the second case, Gaussian noise $\epsilon$, with $\mu = 0$ and $\sigma = 0.3$ was applied to the ground truth place and head-direction cells at time 0: $\vec{c}_0$, $\vec{h}_0$. We then optimized the parameters of the network with the objective of minimising the testing error in both noise conditions (see Fig.2.2).

2.2.2 Deep Reinforcement Learning - Environments and Tasks
We assessed the performance of agents on three environments seen by the agent from a first-person perspective in the DeepMind Lab [212] platform. DeepMind Lab is a first-person 3D game platform that provides a suite of challenging 3D navigation and puzzle-solving tasks for learning agents. Its primary purpose is to
2.2. Methods

Figure 2.3: DeepMind Lab.

a. Example of the Goal-Driven environment where the goal is shown. b. Example of the Goal-Doors environment where a closed door (in black) on the left is show, and an open door is shown on the right.

DeepMind Lab. act as a testbed for research in artificial intelligence, especially deep reinforcement learning. It was chosen for its flexibility in designing testing environments and also because it is open-source and so it could make replicability easier. Fig. 2.3 shows some frames from the agent view.

2.2.2.1 Custom Environment: Square Arena

This comprised a 10×10 square arena - which corresponds to 2.5×2.5 meters assuming an agent speed of 15 cm/s (Fig. 2.10b, c). The arena contained a single, coloured, intra-arena cue whose position and colour changed each episode — as did the texture of the floor, the texture of the walls and the goal location. As in the goal-driven and goal-door environments described below, there were a set of distal cues (i.e. buildings) that paralleled the design of virtual reality environments used in human experiments[213]. These distal cues were rendered at infinity — so as to provide directional but not distance information — and their configuration was consistent across episodes. At the start of each episode the agent (described below) started in a random location and was required to explore in order to find an unmarked goal, paralleling the task of rodents in the classic Morris water maze. The agent always started in the central section of the maze (i.e. 1.5×1.5 meters) of the environment. Noise in the velocity input $\vec{u}_t$ was applied throughout training and testing (i.e. Gaussian noise $\varepsilon$, with $\mu = 0$ and $\sigma = 0.01$). The action space is discrete (six actions) but affords fine-grained motor control (i.e. the agent could ro-
2.2. Methods

tate in small increments, accelerate forward/backward/sideways, or effect rotational acceleration while moving).

2.2.2.2 DeepMind Lab Environments: Goal-Driven and Goal-Doors

Goal-driven and Goal-Doors are complex, visually-rich multi-room environments (see Fig. 2.11a-d). Mazes were formed within an $11 \times 11$ grid, corresponding to $2.7 \times 2.7$ meters, (see below for definition of larger $11 \times 17$ mazes). Mazes were procedurally generated at the beginning of each episode; thus, the layout, wall textures, landmarks (i.e. intra-maze cues on walls) and goal location were different for each episode but consistent within an episode. Distal cues, in the form of buildings rendered at infinity, were as described for the square arena (see above and see Fig. 2.3a).

The critical difference between goal-driven and goal-doors tasks is that the latter had the additional challenge of stochastic doors within the maze (see Fig. 2.3b). Specifically, the state of the doors (i.e. open or closed) randomly changed during an episode each time the agent reached the goal. This meant that the optimal path to the goal from a given location changed during an episode – requiring the agent to recompute trajectories.

In both tasks the agent starts at a random location within the maze and its task is to explore to find the goal. The goal in both levels was always represented by the same object (see Fig. 2.11c). After getting to the goal the agent received a reward of 10 points after which it was teleported to a new random location within the maze. In both levels, episodes lasted a fixed duration of 5,400 environment steps (90 seconds).

2.2.2.3 Generalisation on larger environments.

We tested the ability of agents trained on the standard environment ($11 \times 11$) to generalise to larger environments ($11 \times 17$, corresponding to $2.7 \times 4.25$ meters). The procedural generation and composition of these environments was done as with the standard environments. Each agent was trained in the $11 \times 11$ goal-driven maze for a total of $10^9$ environment steps, and the best performing replica (i.e. highest asymptotic performance averaged over 100 episodes in $11 \times 11$) was selected for
evaluation in the larger maze. Note that the weights of the agent were frozen during evaluation on the larger maze. Evaluation was over 100 episodes of fixed duration 12,600 environment steps (210 seconds).

2.2.2.4 Probe maze to test for shortcut behaviour

To test the agent's ability to follow novel, goal-directed routes, we designed a probe environments inspired by mazes designed to test the shortcut abilities of rodents and utilised here to test the capacities of agents previously trained in the “goal doors” environments.

The first maze used to test shortcut behaviour was a linearized version of Tolman's sunburst maze[163] (Fig. 2.12a). The maze was used to determined if the agent was able to follow an accurate heading towards the goal when a path became available. The maze was size $13 \times 13$ and contained 5 evenly spaced corridors, each of which had a door at the end closest to the start position of the agent. The agent always started on one side of the corridors with the same heading orientation (North; see Fig 2.12a) and the goal was always placed in the same location on the other side of the corridors. Until the agent reached the goal the first time only one door was open (door 5, Fig. 2.12a), but after that all the doors were opened for the remainder of the episode. After reaching the goal, the agent was teleported to the original position with the same heading orientation. This maze was used to test the shortcut capabilities of agents that had been previously trained in the “goal doors” environment. All the agents were tested in the maze for 100 episodes, each one lasting for a fixed duration of 5,400 environment steps (90 seconds).

The second maze, termed double E-maze(Fig. 2.12e), was designed to test the agents abilities to traverse a previously unvisited section of an environment. The maze was size $12 \times 13$ and was formed of 2 symmetric sides each one with 3 corridors. The goal location was always on the bottom right or left, and the location was randomised over episodes. During training, the left and right corridors closest to the bottom (i.e. those providing the shortest paths to the goals) were always closed from both sides to avoid any exploration down these corridors (see 2.12e&f). This ensured any subsequent shortcut behaviour had to traverse unexplored space. Of the
remaining corridors, at any time, on each side only one was accessible (top or middle, randomly determined). Each time the agent reached the goal, the doors were randomly configured again (with the same constraints). The agent always started in a random location in the central room with a random orientation. At test time, after the agent reached the goal for the first time, all corridors were opened, allowing potential shortcut behaviour (see 2.12g&h). During the test phase, the agent always started in the centre of the central room facing north. Each agent was trained for $1e9$ environment step divided into episodes of 5,400 steps (90 seconds), and subsequently tested for 100 episodes, each one lasting for a fixed duration of 5,400 environment steps (90 seconds).

### 2.2.2.5 Agent Performance

For evaluating agent performance during training (as in Fig. 2.10f, Fig. 2.11e,f) we selected the 30 replicas (out of 60) which had the highest average cumulative reward across 100 episodes. Also we assessed the robustness of the architecture over different initial random seeds and the hyperparameters in Table A.2 by calculating the area under the curve (AUC). To plot the AUC we ran 60 replicas with hyperparameters sampled from the same interval (see Table A.2) and different initial random seeds (see Fig.A.3a-c).
2.2.3 Deep Reinforcement Learning Agent Architectures

2.2.3.1 Architecture for the Grid Cell Agent.

The architecture of the supervised network (grid network, light blue dashed) was incorporated into a larger deep RL network, including a visual module (green dashed) and an actor critic learner (based on A3C [214]; dark blue dashed). In this case the supervised learner does not receive the ground truth $\vec{c}_0$ and $\vec{h}_0$ to signal its initial position, but uses input from the visual module to self-localize after placement at a random position within the environment. Visual module: since experimental evidence suggests that place cell input...
to grid cells functions to correct for drift and anchor grids to environmental cues[206, 186], visual input was processed by a convolutional network to produce place cell (and head direction cell) activity patterns which were used as input to the grid network. The output of the vision module was only provided 5% of the time to the grid network, akin to occasional observations made by behaving animals of salient environmental cues[206].

The agent architecture (see Fig.2.4) was composed of a visual module, of the grid cell network (described above), and of an actor-critic learner[214]. The visual module was a neural network with input consisting of a three channel (RGB) $64 \times 64$ image $\phi \in [-1, 1]^{3 \times 64 \times 64}$. The image was processed by a convolutional neural network (see below for the details of the convolutional neural network), which produced embeddings, $\vec{e}$, which in turn were used as input to a fully connected linear layer trained in a supervised fashion to predict place and head-direction cell ensemble activations, $\vec{c}$ and $\vec{h}$ (as specified above), respectively. The predicted place and head direction cell activity patterns were provided as input to the grid network 5% of the time on average, akin to occasional imperfect observations made by behaving animals of salient environmental cues[206]. Specifically, the output of the convolutional network $\vec{\tilde{e}}$ was then passed through a masking layer which zeroed the units with a probability of 95%. This was done to prevent the network to rely too much on vision and discard the velocity inputs.

The grid cell network of the agent was implemented as in the supervised learning set up except that the LSTM (“GRID LSTM”) was not initialised based upon ground truth place cell activations but rather set to zero. The input to the grid cell network were the two translational velocities, $u$ and $v$, as in DeepMind Lab it was possible to move in a direction different from the facing direction, and the sine and cosine of the angular velocity, $\dot{\phi}$, (these velocities were provided by DeepMind Lab) — and additionally the $\vec{y}$ and $\vec{z}$ output by the vision module. In contrast to the supervised learning case, here the grid cell network had to use $\vec{y}$ and $\vec{z}$ to learn how to reset its internal state each time it was teleported to an arbitrary location in the environment (e.g. after visit to goal). As in the supervised learning experiments described above, the configuration of place fields (i.e. location of place field centres
in the $11 \times 11$ environments, “goal-driven” and “goal-doors”, $10 \times 10$ square arena, and $13 \times 13$ double E) were constant throughout training (i.e. across episodes).

For the actor-critic learner the input was a three channel (RGB) $64 \times 64$ image $\phi_t \in [-1, 1]^{3 \times 64 \times 64}$, which was processed by a convolutional neural network followed by a fully connected layer. The convolutional neural network had four convolutional layers. The first convolutional layer had 16 filters of size $5 \times 5$ with stride 2 and padding 2. The second convolutional layer had 32 filters of size $5 \times 5$ with stride 2 and padding 2. The third convolutional layer had 64 filters of size $5 \times 5$ with stride 2 and padding 2. Finally, the fourth convolutional layer with 128 filters of size $5 \times 5$ with stride 2 and padding 2. All convolutional hidden layers were followed by a rectifier nonlinearity. The last convolution was followed by a fully connected layer with 256 hidden units. The same convolutional neural network was used for the actor-critic learner. The weights of the two network were not shared.

The output of the fully connected layer of the convolutional network $\vec{e}_t$ was then concatenated with the reward $r_t$, the previous action $a_{t-1}$, the current “grid code”, $\vec{g}_t$, goal “grid code”, $\vec{g}_*$(i.e. linear layer activations observed last time the goal was reached) — or zeros if the goal had not yet been reached in the episode. Note we refer to these linear layer activations as “grid codes” for brevity, even though units in this layer comprise also units resembling head direction cells, border cells, and also other non classified units (e.g. see Fig.2.9a). This concatenated input was provided to an LSTM with 256 units. The LSTM had 2 different outputs. The first output, the actor, is a linear layer with 6 units followed by a softmax activation function, that represents a categorical distribution over the agent’s next action. The second output, the critic, is a single linear unit that estimates the value function. Note that we refer to this as the ”policy LSTM" for brevity, even though it also outputs the value function.
2.2.3.2 Place cell agent.

a) The A3C implementation is as described in [214]. b) The place cell agent was provided with the ground-truth place, \( \vec{c}_t \), and head-direction, \( \vec{h}_t \), cell activations (as described above) at each time step. The output of the fully connected layer of the convolutional network \( \vec{c}_t \) was concatenated with the reward \( r_t \), the previous action \( a_{t-1} \), the ground-truth current place code, \( \vec{c}_t \), and current head-direction code, \( \vec{h}_t \) — together with the ground truth goal place code, \( \vec{c}_\ast \), and ground truth head direction code, \( \vec{h}_\ast \), observed the last time the agent reached the goal.

The place cell agent architecture is shown in Figure see Fig.2.5b. In contrast...
to the grid cell agent, the place cell agent used ground truth information: specifically, the ground-truth place, \( \vec{c}_t \), and head-direction, \( \vec{h}_t \), cell activations (as described above). These activity vectors were provided as input to the policy LSTM in an analogous way to the provision of grid codes in the grid cell agent.

Specifically, the output of the fully connected layer of the convolutional network \( \vec{e}_t \) was concatenated with the reward \( r_t \), the previous action \( a_{t-1} \), the ground-truth current place code, \( \vec{c}_t \), and current head-direction code, \( \vec{h}_t \) — together with the ground truth goal place code, \( \vec{c}_* \), and ground truth head direction code, \( \vec{h}_* \), observed last time the goal was reached — or zeros if the goal had not yet been reached in the episode (see Fig.2.5b). The convolutional network had the same architecture described for the grid cells agent.
2.2.3.3 Place cell prediction agent.

Figure 2.6: Architecture of the place cell prediction agent and of the NavMemNet agent.

a) The architecture of the place cell prediction agent is similar to the grid cell agent — having a grid cell network with the same parameters as that of the grid cell agent. The key difference is the nature of the input provided to the policy LSTM. Instead of using grid codes from the linear layer of the grid network $\vec{g}$, we used the predicted place cell population activity vector $\vec{y}$ and the predicted head direction population activity vector $\vec{z}$.
(i.e. the activations present on the output place and head direction unit layers of the grid cell network, corresponding to the current and goal position) as input for the policy LSTM. As in the grid cell agent, the output of the fully connected layer of the convolutional network, $\vec{e}_t$, the reward $r_t$, and the previous action $a_{t-1}$, were also input to the policy LSTM. The convolutional network had the same architecture described for the grid cell agent.

b) NavMemNet agent. The architecture implemented is the one described in[145], specifically FRMQN but the Asynchronous Advantage Actor-Critic (A3C) algorithm was used in place of Q-learning. The convolutional network had the same architecture described for the grid cell agent and the memory was formed of 2 banks (keys and values), each one composed of 1350 slots.

The architecture of the place cell prediction agent (see Fig.2.6a) is similar to the grid cell agent described above: the key difference is the nature of the input provided to the policy LSTM as described below. The place cell prediction agent had a grid cell network — with the same parameters as that of the grid cell agent. However, instead of using grid codes from the linear layer of the grid network $\vec{g}$, as input for the policy LSTM (i.e. as in the grid cell agent), we used the predicted place cell population activity vector $\vec{y}$ and the predicted head direction population activity vector $\vec{z}$ (i.e. the activations present on the output place and head direction unit layers of the grid cell network at each timestep). Specifically, the output of the fully connected layer of the convolutional network, $\vec{e}_t$, was concatenated with the reward $r_t$, the previous action $a_{t-1}$, the current predicted place cell activity vector, $\vec{y}_t$, and the current predicted head direction cell activity vector $\vec{h}_t$ - and the goal predicted place cell activity vector, $\vec{y}_g$, and goal head direction activity vector, $\vec{h}_g$, observed the last time the agent had reached the goal - or zeros if the agent had not yet reached the goal within the episode (see Fig.2.5). The convolutional network had the same architecture described for the grid cell agent.

The critical difference between the place cell agent and the place cell prediction agent (see Fig.2.5b and 2.6a respectively) is that the former used ground truth information (i.e. place and head direction cell activations for current location and goal location) - whereas the latter used the population activity produced across the
output place and head direction cell layers (i.e. for current location and goal location) by the linear layer of the same grid network as utilised by the grid cell agent.

2.2.3.4 Place cell agent with homogeneously sized place fields

We tested agents with fields — modelled as regular 2D Gaussians — having standard deviations of 7.5cm, 25cm, and 75cm bins. The agent with fields of size 7.5cm was found to perform best (highest cumulative reward on the Morris water maze task; see Supplemental Results) and hence was chosen as the primary place cell control agent (see main text for score comparisons).

Place cell agent with heterogeneously sized place fields: to control for differences in the number and area of spatial fields between agents, we also generated two further place cell agents that were explicitly matched to the grid cell agent. Specifically, we used a watershedding algorithm[215] to detect 660 individual grid fields in the grid-like units of the grid cell agent. The distribution of the areas of these fields were found to exhibit 3 peaks — based on a Gaussian fitting procedure — having means equivalent to 2D Gaussians with standard deviations of 8.2cm, 15.0cm, and 21.7cm. Hence we generated a further control agent having 395 place cells of size 8.2cm, 198 of size 15.0cm, and 67 of 21.7cm — 660 place cells in total, the relative numbers reflecting the magnitudes of the Gaussians fit to the distribution. A final control agent was also generated having 256 place cell units in total — the same number of linear layer units as the grid agent — distributed across the same three scales in a similar ratio. Additionally, we note that from a machine learning perspective, the place cell and grid cell agents with the same number of linear layer units are in principle well matched since they are provided with the same input information and have an identical number of parameters.

2.2.3.5 A3C

We implemented the asynchronous advantage actor-critic architecture described in[214] with convolutional network having the same architecture described for the grid cell agent (see Fig.2.5a).
2.2.3.6 Other Agents

We also assessed the performance of two deep RL agents with external memory (see Fig.2.6b), which served to establish the challenging nature of the multi-compartment environments (goal-doors and goal-driven). Agent with external memory have proven very effective in solving navigation tasks and in particular to generalise to unseen environments better than architectures without external memory [145]. First, we implemented a memory network agent (“NavMemNet”) consisting of the FRMQN architecture[145], this architecture is the current state of the art in solving navigation problems like the ones we are interested here. In the version we implemented, instead of Q-learning we used the Asynchronous Advantage Actor-Critic (A3C) algorithm described below. Further, the input to memory was generated as an output from the LSTM controller (see Fig.2.6b), rather than constituting embeddings from the convolutional network (i.e. as in[145]). The convolutional network had the same architecture described for the grid cell agent and the memory was formed of 2 banks (keys and values), each one with 1350 slots.

Second, we implemented a Differentiable Neural Computer (“DNC”) agent which uses content-based retrieval and writes to the most recently used or least recently used memory slots [141]. The architecture was implemented exactly has “NavMemNet”, that is trained with Asynchronous Advantage Actor-Critic (A3C) and the convolutional network had the same architecture described for the grid cell agent and the memory was formed of 1 register with 1350 slots. The version of the “DNC” we implemented is exactly the ones presented in the [141] paper.

2.2.4 Deep Reinforcement Learning - Training Algorithms and Regimes

We assume the standard reinforcement learning setting where an agent interacts with an environment over a number of discrete time steps. As previously defined, at time $t$ the agent receives an observation $o_t$ along with a reward $r_t$ and produces an action $a_t$. The agent’s state $s_t$ is a function of its experience up until time $t$, $s_t = f(o_1, r_1, a_1, ..., o_t, r_t)$ (The specifics of $o_t$ are defined in the architecture section). The $n$-step return $R_{t:t+n}$ at time $t$ is defined as the discounted sum of re-
wards, $R_{t:t+n} = \sum_{i=1}^{n} \gamma^{i-1} r_{t+i}$. The value function is the expected return from state $s$, $V^\pi(s) = \mathbb{E}[R_{t:\infty}|s_t = s, \pi]$, under actions selected according to a policy $\pi(a|s)$.

We used the Asynchronous Advantage Actor-Critic (A3C) algorithm[214], which implements a policy, $\pi(a|s, \theta)$, and an approximation to its value function, $V(s, \theta)$, using a neural network parameterised by $\theta$, which is trained by minimising the following loss function: $L_{A3C} = L_\pi + \alpha L_V + \beta L_H$, where $L_\pi = -\mathbb{E}_{s_t \sim \pi} \left[ \hat{R}_t \right]$, $L_V = \mathbb{E}_{s_t \sim \pi} \left[ \left( \hat{R}_t - V(s_t, \theta) \right)^2 \right]$, $L_H = -\mathbb{E}_{s_t \sim \pi} \left[ H(\pi(\cdot|s_t, \theta)) \right]$. Where $L_H$ is a policy entropy regularisation term. The grid cell network and the vision module were trained with the same loss reported for supervised learning: $L(y, z, c, h) = -\sum_{i=1}^{N} c_i \log(y_i) - \sum_{j=1}^{M} h_j \log(z_j)$.

### 2.2.4.1 Agent training details.

We follow closely the approach of[214]. Each experiment used 32 actor-critic learner threads running on a single CPU machine. All threads applied updates to their gradients every 4 actions (i.e. action repeat of 4) using RMSProp with shared gradient statistics[214]. All the experiments were run for a total of $10^9$ environment steps.

In architectures where the grid cell network and the vision module were present we used a shared buffer [36, 132] where we stored the agents experiences at each time-step, $e_t = (\phi_t, u_t, v_t)$, collected over many episodes. All the 32 actor-critic workers were updating the same shared buffer which had a total size of 20e6 slots. The vision module was trained with mini batches of size 32 frames ($\bar{\phi}$) sampled randomly from the replay buffer. The grid cell network was trained with mini batches of size 10, randomly sample from the buffer, each one comprising a sequence of 100 consecutive observations, $[\bar{\phi}, \bar{u}, \bar{v}]$. These mini batches were firstly forwarded through the vision module to get $\bar{c}$, and $\bar{h}$, which were then passed through a masking layer which masked them to 0 with a probability of 95% (i.e. as described above in section on grid cell architecture). The output of this masking layer was then concatenated with $\bar{u}$, $\bar{v}$, $\sin\bar{\phi}$, $\cos\bar{\phi}$, which were then used as inputs to the grid network, as previously described (see Fig.2.4 for details). Both networks were trained using one single thread, one to train the vision module and another to train
2.2. Methods

the grid network (so in total we used 34 threads). Also, there was no gradient sharing between the actor-critic learners, the vision module and the grid network.

The hyperparameters of the grid cell network were kept fixed across all the simulations and were derived from the best performing network in the supervised learning experiments. For the hyperparameter details of the vision module, the grid network and the actor-critic learner please refer to Table A.2. For each of the agents in this paper, 60 replicas were run with hyperparameters sampled from the same interval (see Table A.2) and different initial random seeds.

2.2.5 Deep Reinforcement Learning - Control experiments

2.2.5.1 Details for lesion experiment

To conduct a lesioning experiment in the agent we trained the grid cell agent with dropout applied on the goal grid code input $\vec{g}_*$. Specifically, every 100 training steps we generated a random mask to silence 20% of the units in the goal grid code ($\vec{g}_*$) - i.e. units were zeroed. This procedure was implemented to ensure that the policy LSTM would become robust through training to receiving a lesioned input (i.e. would not catastrophically fail), and still be able to perform the task.

We then selected the agent with the best performance over 100 episodes, and we computed the grid score [185, 204] (see 2.2.6.3 for details on how grid score was calculated) of all units found in $\vec{g}$. The critical comparison to test the importance of grid-like units to vector-based navigation was as follows. In one condition we ran 100 testing episodes where we silenced the 25% units in $\vec{g}_*$ with the highest grid scores. In the other condition, we ran 100 testing episodes with the same agent with 25% random units in $\vec{g}_*$ silenced. In this second case we ensured head direction cells with a resultant vector length of more than 0.47 were not silenced, to preserve crucial head direction signals. The threshold of 0.47 was obtained by applying the Rayleigh tests of directional uniformity to the binned directional activity maps. A unit was considered to be directionally modulated if the null hypothesis of uniform was rejected at the $= 0.01$ level - corresponding to units with resultant vector length in excess of 0.47. We then compared the performance, and representation of metrics relating to vector-based navigation, of the agents under these two conditions.
2.2.5.2 Details of experiment using "fake" goal grid code

To demonstrate that the goal grid code provided sufficient information to enable the agent to navigate to an arbitrary location we took an agent trained in the square arena, we froze the weights and we ran it in the same square arena for 5,400 steps. Critically, after the 6th time that the agent reaches the goal, we sampled the grid code from a random point that the agent visited in the environment (called fake goal grid code). We then substituted the true goal grid code with this fake goal grid code, to show that this would be sufficient to direct the agent to a location where there was no actual goal.

2.2.6 Neuroscience-based analyses of network units

2.2.6.1 Generation of activity maps

Spatial (ratemaps) and directional activity maps were calculated by taking a fully trained network and collecting data for 500 episodes of 1350 steps each. Then each point in the trajectory was assigned to a specific spatial and directional bin based on its location and direction of facing. Spatial bins were defined as a $32 \times 32$ square grid spanning each environment and directional bins as 20 equal width intervals. Then, for each unit, the mean activity over all the trajectories points assigned to that bin was found. These values were displayed and analysed further without additional smoothing.

2.2.6.2 Inter-trial stability

For each unit the reliability of spatial firing between baseline trials was assessed by calculating the spatial correlation between pairs of rate maps taken at 2 different logging steps in training ($t = 2e5; t' = 3e5$). The total training time was 3e5 so the points were selected with enough time difference to minimise the chances of finding random correlations. The Pearson product moment correlation coefficient was calculated between equivalent bins in the two trials and unvisited bins were excluded from the measure.
2.2. Methods

2.2.6.3 Quantification of spatial activity

**Gridness score and grid scale calculation.** Following [185] and [204] spatial autocorrelograms of ratemaps were used to assess the gridness and grid scale of linear layer units. First, for each unit, the spatial autocorrelogram was calculated as defined in [185]. To calculate gridness[185], a measure of hexagonal periodicity, we followed the ’expanding gridness’ method introduced by [204]. Briefly, a circular annulus centred on the origin of the autocorrelogram was defined, having radius of 8 bins and with the central peak excluded. The annulus was rotated in 30° increments and, at each increment, the Pearson product moment correlation coefficient with the unrotated version of itself found. An interim gridness value was then defined as the highest correlation obtained from rotations of 30, 90 and 150° subtracted from the lowest at 0, 60 and 120°. This process was then repeated, each time expanding the annuls by 2, up to a maximum of 20. Finally, the gridness value was taken as the highest interim score.

Grid scale[185], a simple measure of the wavelength of spatial periodicity, was defined from the autocorrelogram as follows. The six local maxima closest to but excluding the central peak were identified. Grid scale was then calculated as the median distance of these peaks from the origin.

**Directional measures.** Following[216] the degree of directional modulation exhibited by each unit was assessed using the length of the resultant vector of the directional activity map. Vectors corresponding to each bin of the activity map were created:

\[ r_i = \begin{bmatrix} \beta_i \cos \alpha_i \\ \beta_i \sin \alpha_i \end{bmatrix}, \]  

(2.8)

where \( \alpha \) and \( \beta \) are, respectively, the centre and intensity of angular bin \( i \) in the activity map. These vectors were averaged to generate a mean resultant vector:

\[ \vec{r} = \frac{\sum_{i=1}^{N} r_i}{\sum_{i=1}^{N} \beta_i}, \]  

(2.9)

and the length of the resultant vector calculated as the magnitude of \( \vec{r} \). We used 20 angular bins.
Border score. To identify units that were preferentially active adjacent to the edges of the enclosure we adopted a modified version of the border score[169]. For each of the four walls in the square enclosure, the average activation for that wall, $b_i$, was compared to the average centre activity $c$ obtaining a border score for that wall, and the maximum was used as the border-score for the unit:

$$b_s = \max_{i \in \{1, 2, 3, 4\}} \frac{b_i - c}{b_i + c}$$ (2.10)

where $b_i$ is the mean activation for bins within $d_b$ distance from the $i$-th wall and $c$ the average activity for bins further than $d_b$ bins from any wall. In all our experiments 20 by 20 bins were used and $d_b$ took value 3.

Threshold setting for gridness, border score, and directional measures. The hexagonality of the spatial activity map (gridness), directional modulation (length of resultant vector), and propensity to be active against environmental boundaries (border scale) exhibited by units in the linear layer were benchmarked against the 95th percentile of null distributions obtained using a permutation procedure [217, 218] applied to each unit’s ratemap. This shuffling procedure aimed to preserve the local topography of fields within each ratemap while distributing the fields themselves at random[218].

For the gridness measure and border score, null distributions were constructed using a ‘field shuffle’ procedure equivalent to that specified by[218]. Briefly, a watershedding algorithm[215] was applied to the ratemap to segment spatial fields. The peak bin of each field was found and allocated to a random position within the ratemap. Bins around each peak were then incrementally replaced, retaining as far as possible their proximity to the peak bin. This procedure was repeated 100 times for each of the units present in the linear layer and the gridness and border score of the shuffled ratemaps assessed as before. In each case the 95th percentile of the resulting null distribution was found and used as a threshold to determine if that unit exhibited significant grid or border-like activity.

To validate the thresholds obtained using shuffling procedures we calculated alternative null distributions by analysing the grid and border responses of linear units.
from 500 untrained networks. Again, in each case, a grid score and border score for each unit was calculated, these were pooled, and the 95th percentile found. In all cases the thresholds obtained by the first method were found to be most stringent and these were used for all subsequent analyses. The means, over units, of the thresholds obtained were gridness > 0.37 and border score > 0.50. Units exceeding these thresholds were considered to be grid-like and border-like, respectively.

To establish a significance threshold for directional modulation we calculated the length of the resultant vector that would demonstrate statistical significance under a Rayleigh test of directional uniformity at $\alpha = 0.01$. The resultant vector was calculated by first calculating the average activation for 20 directional bins. A threshold length of 0.47 for the resultant vector was obtained. The most stringent of these two thresholds was used.

### 2.2.6.4 Clustering of scale in grid-like units

To determine if grid-like units exhibited a tendency to cluster around specific scales we applied two methods.

First, following [187], to determine if the scales of grid-like units (gridness > 0.37, 129/512 units) followed a continuous or discrete distribution we calculated the 'discreteness measure'[187] of the distribution of their scales. Specifically, scales were binned into a histogram with 13 bins distributed evenly across a range corresponding to scales 10 to 36 spatial bins. ‘Discreteness’ was defined as the standard deviation of the counts in each bin. Again following[187], statistical significance for this value was obtained by comparing it to a null distribution generated from a shuffled version of the same data. Specifically, shuffles were generated as follows: For each unit, a random number was drawn from a flat distribution between -1/2 and +1/2 of the smallest grid scale in this case between -7 and +7 spatial bins. The random number was added to the grid scales, the population was binned as before, and the discreteness score calculated. This procedure was completed 500 times. The discreteness score of the real data was found to exceed that of all the 500 shuffles ($p < 0.002$).

Second, to characterise the number and location of scale clusters, the distri-
bution of scales from grid-like units was fit with Gaussian mixture distributions containing 1 to 8 components. Fits were made using an Expectation-Maximization approach implemented with fitgmdist (Matlab 2016b, Mathworks, MA). The efficiency of fits made with different numbers of components was compared using Bayesian Information Criterion (BIC)[219] the model (3 components) with the lowest BIC score was selected as the most efficient.

2.2.6.5 Multivariate decoding of the representation of metric quantities within the LSTM

A key prediction of the vector-based navigation hypothesis is that grid codes should allow downstream regions to compute a set of metric codes relevant to accurate goal-directed navigation. Specifically, Euclidean distance and allocentric direction to the goal should both be computed by an agent using vector-based navigation (see Fig. 2.10j&k also Fig. 2.11i&k). To test whether the same representations can be found in the grid cell agent, and thereby provide additional evidence that it is indeed using a vector-based navigation strategy, we recorded the activity in the policy LSTM of the grid cell agent while it navigated in the land-maze and goal-driven environments. For each environment and agent, we collected data from 200 separate episodes. In each episode, we recorded data from the time period following the first time the agent reached the goal and was teleported to a new location in the maze. After an initial period to allow self-localization (8 steps), we examined the representation of the metric quantities over the next 12 steps, where the LSTM activity was sampled at 4 even points over those steps. We focussed on this time period because the agent potentially has knowledge of the goal location, but has not yet been able to learn the optimal path to the goal. Thus it is this initial period of time where the computation of the vector-based navigation metrics should be most useful, as this allows accurate navigation right from the start of being teleported to a new location. In the land maze task, we additionally collected the same data from a place cell agent control, and the two lesioned grid cell agents. In the goal driven task, we collected data from the place cell agent and A3C. For each agent, we applied a decoding analysis to the LSTM dictating the policy and value
function. We ran two separate decoding analyses, looking for evidence of each of the two metric codes (i.e. Euclidean distance, allocentric goal direction). For each decoding analysis we trained a L2-regularized (ridge) regression model on all data apart from the first 21 time-steps of each episode. The model was then tested on the four early sampling steps of interest, where accuracy was assessed as the Pearson correlation between the predicted and actual values over the 200 episodes. The penalization parameter was selected by randomly splitting the training data into internal training and validation sets (90% and 10% of the episodes respectively). The optimal parameter was selected from 30 values, evenly spaced on a log scale between 0.001 and 1000, based on the best performance on the validation set. This parameter was then used to train the model on the full training set, and evaluated on the fully independent test set. As the allocentric direction metric is circular, we decomposed the vector into two target variables: the cosine and sine of the polar angle. All reported allocentric decoding results are the average of the cosine and sine results. For the purpose of comparing decoding accuracy across agents, we report the difference in accuracy, along with a 95% bootstrapped confidence interval on this difference, based on 10,000 samples.

2.2.6.6 Statistical reporting

We followed the guidelines outlined by[220]. Specifically reporting effect sizes and confidence intervals. Unless otherwise stated, the effect sizes are calculated using the following formula:

\[
effect size = \frac{\mu_{group 1} - \mu_{group 2}}{\sigma_{pooled}},
\]  

and the \( \sigma_{pooled} \) was calculated accordingly to[221] using:

\[
\sigma_{pooled} = \sqrt{\frac{(N_{group 1} - 1) \times \sigma_{group 1}^2 + (N_{group 2} - 1) \times \sigma_{group 2}^2}{N_{group 1} + N_{group 2} - 2}}
\]
The confidence interval for the effect size was calculated accordingly to [222] using:

\[ ci_{\text{effect size}} = \sqrt{\frac{N_{\text{group 1}} + N_{\text{group 2}}}{N_{\text{group 1}} \times N_{\text{group 2}}} + \frac{\text{effect size}^2}{2 \times (N_{\text{group 1}} + N_{\text{group 2}})}}. \] (2.13)

2.3 Results

In this section we are presenting the results of the supervised learning experiment where the "grid network" developed grid-like units as a solution to optimise the self-location problem posed by the objective function. We then present how the "grid network" has been used in a deep learning agent, the "grid cells agent", to test the idea that grid cells support vector based navigation. Finally we present the performance of the "grid cells agent" in complex multi-compartment environments which showed the ability of this agent to take shortcut through part of the environments previously unvisited.

2.3.1 Supervised Learning - Path Integration in a Square Arena

The "grid network" was able to path integrate accurately in this setting involving foraging behaviour (mean error after 15s trajectory, 16cm vs 91cm for an untrained network, effect size = 2.83; 95% CI [2.80, 2.86], Fig. 2.7b&c). Strikingly, individual units within the linear layer of the network developed stable spatial activity profiles similar to neurons within the entorhinal network [159, 185] (Fig. 2.7d, and Fig. A.1 for the whole population). Specifically, 129 of the 512 linear layer units (25.2%) resembled grid cells, exhibiting significant hexagonal periodicity (gridness [185]) versus a null distribution generated by a conservative fields shuffling procedure (see 2.2.6.3 for details on the shuffling procedure), which resulted in a threshold of 0.37. The scale of the grid-patterns, measured from the spatial autocorrelograms of the activity maps [185], varied between units (range 28cm to 115cm, mean 66cm) and followed a multi-modal distribution, consistent with empirical results from rodent grid cells [186, 187] (Fig. 2.7e). To assess these clusters we fit mixtures of Gaussians, finding the most parsimonious number by minimizing the Bayesian Information Criterion (BIC, in Fig. 2.8 we report the distribution of
2.3. Results

Figure 2.7: Emergence of Enthorinal Grid Cells in a Deep Neural Network Trained to Path Integrate.

a. Schematic of network architecture (see Extended Data Figure 1 for details).  
b. Example trajectory (15s), self-location decoded from place cells resembles actual path (respectively, dark and light-blue).  
c. Accuracy of decoded location before (blue) and after (green) training.  
d. Linear layer units exhibit spatially tuned responses resembling grid, border, and head direction cells. Ratemap shows activity over location (top), spatial autocorrelogram of the ratemap with gridness indicated (middle), polar plot show activity vs. head direction (bottom).  
e. Spatial scale of grid-like units ($n = 129$) is clustered. Distribution is more discrete than chance (effect size = 2.98, 95% CI [0.97, 4.91]) and best fit by a mixture of 3 Gaussians (centres 0.47, 0.70 & 1.06m, ratio=1.49 & 1.51).  
f. Directional tuning of the most strongly directional units ($n = 52$). Lines indicate length and orientation of resultant vector (see Methods), exhibiting a six-fold clustering reminiscent of conjunctive grid cells[213].  
g. Distribution of gridness and directional tuning. Dashed lines indicate 95% confidence interval from null distributions (based on 500 data permutations), 14 (11%) grids exhibit directional modulation (see Methods). Similar results were seen in a circular environment (Extended Data Figure 3).
BIC vs. the number of Gaussians fit). The distribution was best fit by 3 Gaussians (means 47cm, 70cm, and 106cm), indicating the presence of scale clusters with a ratio between neighbouring clusters of approximately 1.5, closely matching theoretical predictions [223] and lying within the range reported for rodents [186, 187] (Fig. 2.7e, Fig. 2.8). Interestingly when the network was trained again in a circular environment the ratio adapted to 1.66. The linear layer also exhibited units resembling head direction cells (10.2%), border cells (8.7%), and a small number of place cells [200] as well as conjunctions of these representations (Fig. 2.7d,f&g, Fig. A.1).

Figure 2.8: Characterization of grid-like units in Square environment and Circular environment.

a) The scale (assessed from the spatial autocorrelogram of the ratemaps) of grid-like units exhibited a tendency to cluster at specific values. The number of distinct scale clusters was assessed by sequentially fitting Gaussian mixture models with 1 to 8 components. In each
2.3. Results

Results were assessed using Bayesian information criterion (BIC). BIC was minimized with three Gaussian components indicating the presence of three distinct scale clusters. b) Spatial stability of units in the linear layer of the supervised network was assessed using spatial correlations — bin-wise Pearson product moment correlation between spatial activity maps (32 spatial bins in each map) generated at 2 different points in training, \( t = 2e5 \) and \( t' = 3e5 \) training steps. That is, \( \frac{2}{3} \) of the way through training and the end of training, respectively. This separation was imposed to minimise the effect of temporal correlations and to provide a conservative test of stability. Grid-like units (gridness > 0.37) blue, directionally modulated units (resultant vector length > 0.47) green. Grid-like units exhibit high spatial stability, while directionally modulated units do not. c) Robustness of the grid representation to starting conditions. The network was retrained 100 times with the same hyperparameters but different random seeds controlling the initialisation of network weights, \( \vec{c} \) and \( \vec{h} \). Populations of grid-like units (gridness > 0.37) were found to appear in all cases, the average proportion of grid-like units being 23% (SD of 2.8%). d) Circular environment: the supervised network was also trained in a circular environment (diameter = 2.2m). As before, units in the linear layer exhibited spatially tuned responses resembling grid, border, and head direction cells. Eight units are shown. Top, ratemap displaying activity binned over location. Middle, spatial autocorrelogram of the ratemap, gridness[185] is indicated above. Bottom, polar plot of activity binned over head direction. e) Spatial scale of grid-like units (n = 56 (21.9%)) is clustered. Distribution is best fit by a mixture of 2 Gaussians (centres 0.58 & 0.96m, ratio = 1.66). f) Distribution of directional tuning for 31 most directionally active units, single line for each unit indicates length and orientation of resultant vector[216] g) Distribution of gridness and directional tuning. Dashed lines indicate 95% confidence interval derived from shuffling procedure (500 permutations), 5 grid units (9%) exhibit significant directional modulation.

To ascertain how robust these representations were, we retrained the network 100 times, in each instance finding similar proportions of grid-like units (mean 23% SD 2.8%, units with significant gridness scores) and other spatially modulated units (see Fig. 2.8). Conversely, grid-like representations did not emerge in networks without regularisation (dropout and gradient clipping; also see [224], Fig. A.2 as
an example). Therefore, the use of regularisation was critical to the emergence of entorhinal-like representations. Notably, therefore, our results show that grid-like representations reminiscent of those found in the mammalian entorhinal cortex emerge in a generic network trained to path integrate, contrasting with previous approaches using pre-configured grid cells (e.g. [225]; see Discussion). Further our results are consistent with the view that grid cells represent an efficient and robust basis for a location code updated by self-motion cues [159, 193, 196, 226].

2.3.2 Deep Reinforcement Learning - Path Integration in the Virtual Morris Watermaze

![Figure 2.9: Characterisation of grid-like representations and robustness of performance for the grid cell agent in the square “land maze” environment.](image)

a) Spatial activity plots for the 256 linear layer units in the agent exhibit spatial patterns similar to grid, border, and place cells. b) Cumulative reward indexing goal visits per episode (goal = 10 points) when distal cues are removed (dark blue) and when distal cues are present (light blue) — performance is unaffected, hence dark blue largely obscures light blue. Average of 50% best agent replicas (n=32) plotted (see Methods). The gray band displays the 68% confidence interval based on 5000 bootstrapped samples. c) Cumulative reward per episode when no goal code was provide (light blue) and when goal code was provided (dark blue). When no goal code was provided the agent performance fell to that of the baseline deep RL agent (A3C) (100 episodes average score ”no goal code” = 123.22 vs. A3C = 112.06 ,effect size = 0.21, 95% CI [0.18, 0.28]). Average of
50% best agent replicas (n=32) plotted (see Methods). The gray band displays the 68% confidence interval based on 5000 bootstrapped samples. d) After locating the goal for the first time during an episode the agent typically returned directly to it from each new starting position, showing decreased latencies for subsequent visits, paralleling the behaviour exhibited by rodents.

To better understand the advantage conveyed by a grid-like representation, we trained the agent to navigate to an unmarked goal in a simple setting inspired by the classic Morris water maze (Fig. 2b&c; 2.5m × 2.5m square arena; see Methods 2.2.2.1). The agent was trained in episodes to ensure it was able to generalise to arbitrary open field enclosures, each episode consisted of 5,400 steps — corresponding to approximately 90 s in total — after which the goal location, floor texture, and cue location were randomised. An episode started with the agent in a random location, requiring it to first explore in order to find an unmarked goal. Upon reaching the goal the agent was teleported to another random location and continued to navigate with the aim of maximising the number of times it reached the goal before the episode ended. In this setting self-localisation was more challenging. Previously, in the experiment described above, information about the ground truth initial location was provided to initialise the LSTM, here the grid network learned to use visual information to determine the agent’s starting location and to correct for drift resulting from noise introduced to the velocity inputs (see Methods). Despite these differences the grid network continued to self-localize accurately, outputting place cell predictions consistent with the agent’s location (Fig. 2.10e).

Notably, the agent was still able to self localise accurately in this more challenging setting (mean error after 15s trajectory, 12cm vs 88cm for an untrained network, effect size = 2.82; 95% CI [2.79, 2.84], Fig. 2.10e). After locating the goal for the first time during an episode, the agent typically returned directly to it from each new starting position, showing decreased latencies for subsequent visits (average score for 100 episodes: grid cell agent = 289 vs place cell agent = 238, effect size = 1.80, 95% CI [1.63, 1.99], Fig. 2.10h, Fig. 2.9d).

Performance of the grid cell agent was substantially better than that of a con-
trol place cell agent (Fig. 2.10f) with homogeneous place fields tuned to maximize performance. This agent was chosen because place cells provide a robust representation of self-location but are not thought to provide a substrate for long range vector calculations [198]. Further, to additionally control for differences in the number and area of spatial fields between agents, we also generated two place cell agents – incorporating 256 and 660 heterogeneously sized place fields – that were explicitly matched to the grid cell agent (see section 2.2.3.4 in the Methods for details). Again, the performance of the grid cell agent was found to be considerably better than these additional place cell agents (Average score over 100 episodes: grid cell agent = 289 vs. best place agent with 660 heterogeneous fields = 212, effect size = 3.93, 95% CI [3.54, 4.31]; best place agent with 256 heterogeneous fields = 225, effect size = 3.52, 95% CI [3.18, 3.87]).

Finally, we examined the units in the linear layer, again finding a heterogeneous population resembling those found in entorhinal cortex, including 21.4% cells that surpassed the threshold for being considered grid-like units (Fig. 2.10g, Fig 2.9) — paralleling the dependence of mammalian grid cells on self-motion information [201, 227] and spatial cues [159, 186].

2.3.3 Deep Reinforcement Learning - Vector Based Navigation in the Virtual Morris Watermaze

Next we turned to the central aim - testing the hypothesis that grid cells endow agents with the ability to perform vector-based navigation, enabling downstream regions to calculate goal directed vectors by comparing current activity with that of a remembered goal[196, 197, 198]. In the agent, we expect these calculations to be performed by the policy LSTM, which receives the current activity pattern over the linear layer (termed “current grid code”; see Fig. 2.10d and Fig. 2.4) as well as that present the last time the agent reached the goal (termed “goal grid code”), using them to control movement. Hence we performed several manipulations, which yielded four lines of evidence in support of the vector-based navigation hypothesis.

First, to demonstrate that the goal grid code provided sufficient information
2.3. Results

Figure 2.10: One-shot open field navigation to a hidden goal.

a. Schematic of vector-based navigation [198].
b. Overhead view of typical environment (icon indicates agent and facing direction).
c. Agent view of (b).
d. Schematic of Deep RL architecture (see Extended Data Figure 5)
e. Accuracy of self-location decoded from place cell units.
f. Performance of grid cell agent and place cell agent (y-axis shows reward obtained within a single episode, 10 points per goal arrival, gray band displays the 68% confidence interval based on 5000 bootstrapped samples).
g. As before the linear layer develops spatial representations similar to entorhinal cortex. Left to right, 2 grid cells, 1 border cell, and 1 head direction cell.
h. On the first trial of an episode the agent explores to find the goal and subsequently navigates directly to it.
i. After successful navigation, the policy LSTM was supplied with a "fake" goal grid-code, directing the agent to this location where no goal was present.
j & k. Decoding of goal-directed metric codes (i.e. Euclidean distance and direction) from the policy LSTM of grid cell and place cell agents.
The bootstrapped distribution (1000 samples) of correlation coefficients are each displayed with a violin plot overlaid on a Tukey boxplot.
to enable the agent to navigate to an arbitrary location, we substituted it with a "fake" goal grid code sampled randomly from a location in the environment (see Methods 2.2.5.2). The agent followed a direct path to the newly specified location, circling the absent goal (Fig. 2.10i) — similar to rodents in probe trials of the Morris water maze (escape platform removed). As a second test, we trained a grid cell agent without providing the goal grid vector to the policy LSTM, effectively "lesioning" this code. Performance of the grid agent drops to that of the baseline deep RL agent (A3C - a standard deep RL architecture, trained without any grid or place cell input), confirming that the goal grid code is critical for vector based navigation (see Fig. 2.9c for the learning curves showing the effect of lesioning the grid code). Thirdly, to confirm the presence of a goal-directed vector, we attempted to decode the scalar quantities composing the vector from the policy LSTM (the decoding details are presented in section 2.2.6.5). Reasoning that the goal directed vector would be particularly important at the start of a trajectory, so we focused on the initial portion of navigation after the agent had reached the goal and was teleported to a new location (see 2.2.6.5 in the Methods for details on the procedure we used). We found that the policy LSTM of the grid cell agent contained representations of two key components of vector-based navigation (Euclidean distance, and allocentric goal direction), and that both were more strongly present than in the place cell agent (Euclidean distance difference in $r = 0.17; 95\% \text{ CI } [0.11, 0.24]$; Goal direction difference in $r = 0.22; 95\% \text{ CI } [0.18, 0.26]$; Figure 2.10j&k). Notably, a neural representation of goal distance has recently been reported in mammalian hippocampus[228] (also see [229]). To determine the behavioural relevance of these two metric codes, we examined the goal-homing accuracy in each episode over several steps immediately following the period of metric decoding. We found that variation in both Euclidean distance ($r = 0.22, 95\% \text{ CI } [-0.32, -0.09]$) and allocentric goal direction ($r = 0.22, 95\% \text{ CI } [-0.38, -0.15]$) decoding error correlated with subsequent behavioural accuracy. This suggests that stronger metric codes are indeed important for accurate goal-homing behaviour.

Finally, to determine the specific contribution of the grid-like units, we made
targeted lesions to the goal grid code and reexamined performance and representation of the goal directed vector. When 25% of the most grid-like units were silenced (see Methods 2.2.5.1), performance was worse than lesioning 25% of cells in the linear layer at random (average score for 100 episodes: 126.1 vs. 152.5, respectively; effect size = 0.38, 95% CI [0.34, 0.42]). Further, as expected, goal-directed vector codes were more strongly degraded (Euclidean distance: random lesions decoding accuracy $r = 0.45$, top-grid lesions decoding accuracy $r = 0.38$, difference in decoding accuracy = 0.08, 95% CI [0.03, 0.13]). We also performed an additional experiment where the effect of the targeted grid lesion was compared to that of lesioning non-grid units with patchy firing. As explained in 2.2.5.1, the patchy multi-field spatial cells that were non-grid units were chosen amongst the units with a grid score lower than the 0.37 threshold. The units chosen had also a head-direction score lower than 0.47 and the number of spatial fields was in the same range as grid-like units (3 to 13). The number of fields in each ratemap was calculated by applying a watershedding[215] algorithm to their ratemap ignoring fields with area smaller than 4 bins. Our results show that the targeted grid cell lesion had a greater effect than the patchy non-grid cell lesion (average score for 100 episodes: 126.1 vs. 151.7, respectively; effect size = 0.38, 95% CI [0.34, 0.42]). These results support a role for the grid-like units in vector-based navigation, with the relatively mild impact on performance potentially accounted for by the difference in lesioning networks as compared to animals. Specifically, the procedure for lesioning networks differs in important respects from experimental lesions in animals — which bears upon the results observed. Briefly, networks have to be trained in the presence of an incomplete goal grid code and thus have the opportunity to develop a degree of robustness to the lesioning procedure – which would otherwise likely result in a catastrophic performance drop (see Methods 2.2.5.1). This opportunity is not typically afforded to experimental animals. This, therefore, may explain the significant but relatively small performance deficit observed in lesioned networks.
2.3.4 Comparison of grid cell agent with other agents in challenging, procedurally-generated multi-room environments

Our comparison agents for the grid cell agent included an agent specifically designed to use a different representational scheme for space (i.e. place cell agent, see Fig. 2.5b and see Methods 2.2.3.2), and a baseline deep RL agent (A3C [214], see Fig. 2.5a). The place cell agent relates to theoretical models of goal-directed navigation from the neuroscience literature (e.g. [230, 231]). A key difference between grid and place cell based models is that the former are proposed to enable the computation of goal-directed vectors across large-scale spaces [196, 198, 197] and [232], whereas place cell based models are inherently limited in terms of navigational range (i.e. to the largest place field) and do not support route planning across unexplored spaces [198]. First, we tested these three agents in the “goal-driven” maze (see Methods 2.2.2.2). The grid-cell agent exhibited high levels of performance, and over the course of 100 episodes, attained an average score of 346.5, beating both the place cell agent (average score 258.76; contrast effect size = 1.98, 95% CI [1.79, 2.18]) and the A3C agent (average score 137.00; contrast effect size = 14.31, 95% CI [12.91, 15.71]). The grid cell agent showed markedly superior performance compared to the other agents in the “goal-doors” maze (average score over 100 episodes: grid cell agent = 284.30 vs place cell agent = 90.53, effect size = 7.86, 95% CI [7.09, 8.63]; A3C agent = 48.69, effect size = 7.73, 95% CI [6.97, 8.48]). Interestingly, therefore, the enhanced performance of the grid cell agent was particularly evident when it was necessary to recompute trajectories due to changes in the door configuration, highlighting the flexibility of vector-based navigation in exploiting ad hoc short-cuts (Fig. 2.11f).

The grid cell agent exhibited stronger performance than a professional human player in both “goal-driven” (average score: grid cell agent = 346.50 vs. professional human player = 261, effect size = 4.00, 95% CI [3.50, 4.52]) and “goal-doors” (average score: grid cell agent = 284.30 vs. professional human player = 240.5, effect size = 2.49, 95% CI [2.18, 2.81]). The human expert received 10 episodes worth of training in each environment before undergoing 20 episodes of
2.3. Results

This is considerably less training than that experienced by the network. Importantly, however, the mammalian brain has evolved to path integrate and naturally the human expert had a lifetimes worth of relevant navigational experience. Hence, although directly drawing concrete conclusions from relative performance of human and agents is necessarily difficult, providing human-level performance is useful as a broad comparison and represents a commonly used benchmark in similar papers [36].

We also tested the ability of agents trained on the standard environment (11 × 11) to generalise to larger environments (11 × 17, corresponding to 2.7 × 4.25 meters) (see Methods 2.2.2.3). The grid cell agent exhibited strong generalisation performance compared to the control agents (average score over 100 episodes grid cell agent = 366.5 vs place cell agent = 175.7, effect size = 4.60, 95% CI [4.16, 5.06]; A3C agent = 219.4, effect size = 3.78, 95% CI [3.41, 4.15]).

We assessed the performance of two deep RL agents with external memory [145], [141] (see Fig.2.6b). Whilst these agents were trained purely using RL — that is, they did not utilise supervised learning implemented by the grid cell agent — their relatively poor performance illustrates the challenge posed by the environments used (i.e. goal-driven and goal-doors) and show they are not readily solved by the use of external memory alone. Importantly, this also serves to highlight the substantial advantage afforded to agents that can exploit vector-based mechanisms grounded in a grid-cell based Euclidean framework of space — and the potential for future work to examine the combination of such navigational strategies with more memory-intensive approaches. We also compared the grid cell agent with a variation of the place cell agent which used the predicted place cell and head direction cell as input to the Policy LSTM instead of the ground truth information (see Fig.2.6a). This agent exhibited substantially poorer performance than the grid agent.

Further, decoding accuracy for Euclidean distance and goal direction was substantially and significantly higher in the grid cell agent than both the place cell (Euclidean distance difference in r = 0.44; 95% CI [0.37, 0.51]; Goal direction
Figure 2.11: Navigation in complex environments.

a-b, Overhead view of multi-room environments “goal-driven” (a) and “goal-doors” (i.e. with stochastic doors) (b) Goal (*) and agent locations (head icon) are displayed. c-d, Agent views of (a) & (b) showing red goal and closed black door. e-f, Agent training performance curves for (a) & (b), and performance of human expert (dotted line). Performance is mean cumulative reward over 100 episodes. The gray band displays the 68% confidence interval based on 5000 bootstrapped samples. g, Distribution of test performance over 100 episodes, showing ability of agents to generalize to a larger version of goal-doors environment, displayed with a violin plot overlaid on a Tukey boxplot for each agent. h, The value function of the grid cell agent is projected onto an example larger goal doors environment as a heatmap. Dotted lines show the extent of the original training environment. Despite the larger size, the value function clearly approximates Euclidean distance to goal. i, Schematic displaying the key metrics required for vector-based navigation to a goal. j-k, Decoding of vector-based metric codes from the policy LSTM of agents during navigation. The bootstrapped distribution (1000 samples) of correlation coefficients are displayed with a violin plot overlaid on a Tukey boxplot in each case.

difference in $r = 0.52$; 95% CI [0.49, 0.56]) and deep RL (Euclidean distance difference in $r = 0.57$; 95% CI [0.5, 0.63]; Goal direction difference in $r = 0.66$; 95% CI [0.62, 0.70]) control agents (Figure 2.11j&k).

2.3.5 Deep Reinforcement Learning - Shortcut

A core feature of mammalian spatial behaviour is the ability to exploit novel shortcuts and traverse unvisited portions of space [193], a capacity thought to depend on vector-based navigation [193, 198]. To assess this, we examined the ability of
the grid cell agent and comparison agents to use novel shortcuts when they became available in specifically configured probe mazes. First, agents trained in the goal-doors environment were exposed to a linearised version of Tolman’s sunburst maze with no further training. The maze contained 5 evenly spaced corridors, each of which had a door at the end closest to the start position of the agent. The agent always started on one side of the corridors with the same heading orientation (North; see Fig 2.12a) and the goal was always placed in the same location on the other side of the corridors. Until the agent reached the goal the first time only one door was open (door 5, Fig. 2.12a), but after that all the doors were opened for the remainder of the episode. After reaching the goal, the agent was teleported to the original position with the same heading orientation. The grid cell agent, but not comparison agents, was reliably able to exploit shortcuts, preferentially passing through the doorways that offered a direct route towards the goal (Fig.2.12a-c). The average testing score of the grid cell agent was higher than that of the place agent (124.1 vs 60.9, effect size = 1.46, 95% CI [1.32, 1.61]) and of the A3C agent (124.1 vs. 59.7, effect size = 1.51, 95% CI [1.36, 1.66]), see Fig. 2.12d.

Next, to test the agents’ abilities to traverse a previously unvisited section of an environment, we employed the “double-E shortcut” maze (and Fig.2.12e-l). During training, the corridor presenting the shortest route to the goal was closed at both ends, preventing access or observation of the interior. In this simple configuration the grid and place cell agents performed similarly, exceeding the RL control agent (see Fig.2.12i). However, at test, when the doors were opened, the grid cell agent was able to exploit the short-cut corridor, whereas the control agents continued to follow the longer route they had previously learnt (Fig.2.12j-l). In the “double-E shortcut” maze performance does not significantly differ between the grid and place cell agents, but both are significantly better than the A3C control (grid cell agent vs. place cell agent, effect size = 0.27, 95% CI [0.24, 0.29]; grid cell agent vs. A3C agent, effect size = 2.99, 95% CI [2.69, 3.29]; place cell agent vs. A3C agent, effect size = 2.92, 95% CI [2.63, 3.21]). When shortcuts become available in the test phase, the grid cell agent performs significantly better than the place agent...
2.3. Results

(grid cell agent vs. place cell agent, effect size = 1.89, 95% CI [1.69, 2.09]; grid cell agent vs. A3C agent, effect size = 12.77, 95% CI [11.48, 14.07]; place cell agent vs. A3C agent, effect size = 14.87, 95% CI [13.35, 16.38]).

Figure 2.12: Flexible use of short-cuts.

a) Overhead view of the linear sunburst maze in initial configuration, with only door 5 open. Example trajectory from grid cell agent during training (green line, icon indicates start location). b) Test configuration with all doors open: grid cell agent uses the newly available shortcuts (multiple episodes shown). c) Histogram showing proportion of times the agent uses each of the doors during 100 test episodes. The agent shows a clear preference for the shortest paths. d) Performance of grid cell agent and comparison agents during test episodes. e) Example grid cell agent and f) example place cell agent trajectory during training in the double E-maze (corridor 1 doors closed). g-h) in the test phase, with all doors open, the grid cell agent exploits the available shortcut (g), while the place cell agent does not (h). i-j) Performance of agents during training (i) and test (j). k-l, The proportion of times the grid (k) and place (l) cell agents use the doors on the 1st to 3rd corridor during test. The grid cell agent shows a clear preference for available shortcuts, while the place cell
2.4 Discussion

Several theoretical papers argue for a computational role of grid cells in providing an efficient and noise-tolerant representation for space [226, 196, 159, 193]. Here we use a novel approach to provide evidence that supports this view, by demonstrating that grid-like representations can emerge in a generic deep neural network trained to optimise the objective of self localisation. Notably, our work contrasts with previous approaches where grid cells have been hard wired [188, 190, 233, 195, 192], derived through eigendecomposition of place fields [234, 235], or arisen through self organisation in the absence of an objective function [236]. It is worth noting that our experiments were not designed to provide insights into the development of grid cells in the brain - due to the limitations of the training algorithm used (i.e. backpropagation) in terms of biological plausibility (although see [237]). More generally, however, our findings accord with the perspective that the internal representations of individual brain regions such as the entorhinal cortex arise as a consequence of optimizing for specific objective functions (e.g. path integration), providing a parallel to the optimisation process in artificial neural networks [154].

We further show that grid-like representations furnish an agent trained by deep RL with an effective basis to support flexible navigation to distant goals in novel environments, in a fashion that generalises to spaces considerably larger than those experienced during training. We demonstrate that the grid cell agent exhibits behavioural signatures of vector-based navigation, in particular the capacity to take direct short-cut routes to goals through previously unexplored regions of space. Critically, we shown that the network perform this strategy by computing and representing the critical metric variables (e.g. euclidean distance and allocentric heading direction to goal). In doing so, we provide compelling support for theoretical perspectives on grid cells that have been relatively lacking in empirical support: specifically, their function in path integration [159, 193, 188, 196] and naviga-
2.4. Discussion

ation based on the computation of goal-directed vectors across large scale spaces [196, 232, 198, 197]. Moreover, we demonstrate that vector based navigation can be effectively combined with a path-based barrier avoidance strategy enabling the exploitation of optimal routes in complex multi-compartment environments.

Our model also departs from the traditional machine learning way of approaching navigation through simultaneous localisation and mapping (SLAM). Conventional SLAM based techniques typically require extensive experience to construct an accurate map of the environment in order to support navigation [225, 238], an approach that requires considerable data to be collected and is inflexible to modification of the environment. Recently deep reinforcement learning (RL) approaches to navigation have been developed [239, 145, 240]; however, these primarily use reactive route-based navigational strategies, and fail to traverse unexplored space or exploit short-cuts. In contrast, at the start of each trajectory, our model is able to rapidly develop a goal-directed vector, providing agents with the ability to exploit novel short-cuts in complex, novel environments.

Finally it is worth pointing out that in this work we primarily set our focus on grid cells and head direction cells, taking for granted the availability of place cells. We believe that a possible future direction could be to augment the model with an episodic buffer that can be used to store visual memories of visited places. Then the network could be be trained to anticipate, given a set of velocity inputs, the similarity of its current visual input to past visual memories. We believe that this augmented model would provide a framework within which place cells would develop - these in turn would become the target on which the appearance of grid cells depend.

Also, a further limitation stems from the fact that we only explored the domain of spatial navigation, by considering the role of grid and head direction cells in the integration of self-motion. However several recent reviews have provided increasing evidence for the role of the hippocampus and surrounding areas in domains other than spatial reasoning [241, 242, 243, 244]. In particular it has been shown that the humans hippocampus is involved in mapping abstract spaces by cre-
ating links between elements that were not experienced together, a process which is believed to support inferential reasoning [160, 245]. Intriguingly, recent evidence also suggest that grid-like representation are used to navigate in this abstract space in support of non-spatial reasoning [246] and to create links between memories. By building on this evidence the next chapter will investigate inferential reasoning, how this is supported by the hippocampus and how to build a deep learning model with similar capabilities.
Chapter 3

Modelling inferential reasoning with Memory Augmented Deep Neural Networks

The work presented in this chapter was previously published as [247] at the ICLR 2020 conference. The current text is based on the published manuscript but has been expanded and elaborated upon.

3.1 Introduction

During everyday life we often need to make judgements that require combinations of facts which were acquired separately, possibly at different times and places. For instance, imagine walking your daughter to a coding summer camp and encountering another little girl with a woman. You might conclude that the woman is the mother of the little girl. A few weeks later, at a coffee shop near your house, you see the same little girl, this time with a man. Based on these two separated episodes you might infer that there is a relationship between the woman and the man. This flexible recombination of single experiences in novel ways is called inferential reasoning and this task is thought to capture the essence of reasoning, that is the appreciation of distant relationships among elements distributed across multiple facts or memories [248].

Interestingly, there is mounting evidence that the hippocampus is critical to
the building of cognitive maps of abstract spaces that creates links between associations experienced at different times to support the kind of inferential reasoning outlined above. For instance, in one study rats were trained to perform the so-called paired associative inference task (PAI). Here the animals had to learn to link randomly paired objects (e.g. A-B and B-C) and later they were required to infer the indirect relationship between objects that were never been experienced together (A-C). All animals were able to learn the direct associations but only rats with intact hippocampi succeeded in the inference trial [249]. In a following study rats where trained on a set of overlapping choice trials (A vs. B, B vs. C, C vs. D and D vs. E) and then subsequently tested on transitive inference judgements (e.g. B vs. D). As before, all rats learned the direct judgements but only the ones with an intact hippocampi where able to perform the longer range inference tests [250]. These results have also been replicated in humans [160, 251] confirming that the hippocampus is needed for creating the associations and hierarchical relations which form the basis of our complex reasoning skills. However, the involvement of the hippocampus in linking events experienced at different times seems to be in conflict with its role in supporting the ability to distinguish similar events, a crucial aspect of episodic memory [252, 106]. Indeed, several computational models highlight the contribution of the hippocampus in the so called pattern separation process, whereby similar events are stored in orthogonal patterns to avoid interference [253, 177].

A recent line of research [254, 245, 255, 256] sheds light on this tension – i.e. how can separated memories be chained together? In particular, it has been shown that the integration of separated experiences emerges at the point of retrieval through a recurrent mechanism that allows multiple pattern separated codes to interact, and therefore support inference. In particular, a computational model published recently, Recurrency and Episodic MEmory REsults in Generalization (REMERGE) [254, 245], provides two clear principles for how the recurrent mechanism might be implemented in the hippocampal circuit. First, memories remain stored in the hippocampus as separated codes to preserve information about their constituent elements, allowing greater flexibility at the time of retrieval. Second, in
REMERGE the single retrieved memories are recirculated as a new input to the hippocampal circuit, a process that continues until the network has settled into a fixed point. Contingent on the difficulty of a given tasks the number of re-circulation in memory for the network to settle to a fixed point varies: hence computation in REMERGE is variable length. Recent studies employing a paired-associate inference task found empirical support for this account in human behavior [245] and neural data [256].

Based on these findings we set out to create a new neural network architecture, called MEMO, which introduces a new multistep retrieval mechanism that supports the flexible weighting of individual elements in memory. This is achieved through a powerful recurrent attention mechanism which adapts the number of memory retrieval operations based on the difficulty of the task at hand. Both aspects will be described in details in the next section.

### 3.1.1 MEMO: a Deep Neural Network Model for the Flexible Recombination of Episodic Memories

Neural networks augmented with external memory, like the Differential Neural Computer [141] and end to end memory networks [144] (EMN) have shown remarkable abilities to tackle difficult computational and reasoning tasks. In parallel, more powerful attention mechanisms [90, 257] or the use of context [258] have recently allowed traditional neural networks to tackle similar tasks. However, some of these tasks present repetitions and commonalities between the train and the test set that neural networks can exploit to come up with degenerate solutions. To overcome this limitation we introduced a new task, called Paired Associative Inference (PAI - see below), which is derived from the neuroscientific literature [249, 245]. This task is meant to capture the essence of inferential reasoning – i.e. the appreciation of distant relationships among elements distributed across multiple facts or memories. PAI is fully procedurally generated and so it is designed to force neural networks to learn abstractions to solve previously unseen associations.

We then use a version of the PAI task (see 3.2.6 in the methods for details), followed by a task involving finding the shortest path and finally bAbi [259], a
standard machine learning test for text understanding and reasoning (see details in 3.2.8), to investigate what kind of memory representations effectively support memory based reasoning. The end to end memory networks (EMN) and other similar models [144, 148, 147] have used fixed memory representations based on combining word embeddings with a positional encoding transformation. A similar approach has been recently implemented by current state of the art language model [90, 68]. By contrast our approach, called MEMO, retains the full set of facts into memory, and then learns a linear projection paired with a powerful recurrent attention mechanism that enables greater flexibility in the use of these memories. MEMO is based on the same basic structure of the external memory presented in EMN [144]. However, its new architectural components can potentially allow for flexible weighting of individual elements in memory and so supporting the form of the inferential reasoning outlined above.

Next, we tackle the problem of prohibitive computation time. In general, the time required to solve a problem is expected to increase with the complexity of the task. However, most machine learning algorithms do not adapt their computational budget based on task complexity. Indeed, in standard neural networks, the computation grows as a function of the size of the input, instead of the complexity of the problem being learnt. Sometimes the input is padded with a fixed number of extra values to provide greater computation [141], in other cases, input values are systematically dropped to reduce the amount of computation (e.g. frame dropping in reinforcement learning [214]). Critically, these values are normally hand tuned by the experimenter; instead, here we are interested in adapting the amount of compute time to the complexity of the task. One approach to this problem is represented by Adaptive Computation Time (ACT) [260]. ACT is a mechanism for learning a scalar halting probability, called the ponder time, to dynamically modulate the number of computational steps needed for each input. An alternative approach is represented by Adaptive Early Exit Networks [261], which give the network the ability to exit prematurely - i.e. not computing the whole hierarchy of layers - if no more computation is needed. A further approach to conditional computation is
the use of REINFORCE [262] to learn a discrete latent variable which dynamically adjusts the number of computation steps. This has been applied to recurrent neural networks where each layer decides whether or not to activate the next layer [263]. REINFORCE has also been used to learn how many steps to “jump” in sequence, so reducing the total number of processed inputs [264]. This jump technique has also been applied to recurrent neural network without the need of REINFORCE [265].

Instead, to tackle the problem of adaptive computation we drew inspiration from a model of human associative memory called REMERGE [254]. In this model, the content retrieved from memory is recirculated back as the new query, then the difference between the content retrieved at different time steps in the re-circulation process is used to calculate if the network has settled into a fixed point, and if so this process terminates. To implement this principle in a neural network, we were inspired by techniques such as adaptive computation time [260]. In our architecture, the network outputs an action (in the reinforcement learning sense) that indicates whether it wishes to continue computing and querying its memory, or whether it is able to answer the given task. We call this the halting policy as the network learns the termination criteria of a fixed point operator (see 3.1). Like Adaptive computation time [260], the network outputs a probability of halting, but unlike ACT, the binary halting random variable is trained using REINFORCE [262], a policy gradient algorithm. The use of REINFORCE to perform variable amounts of computation has been investigated already [e.g. 266, 267] however our approach differs in that we added an extra term to the REINFORCE loss that, by exploiting the mathematical properties of binary random variables, naturally minimizes the expected number of computation steps. Thus we directly encourage our network to explicitly prefer representations and computation that minimize the amount of required computation.

3.2 Methods

In this section we recapitulate End-to-End Memory Networks [144], we then introduce MEMO and two more baselines used for comparisons: Differential Neural
An example where the network took 3 hops. In MEMO the input sequence \( \{x_0, ..., x_T\} \), is embedded into the memory slots using a series of linear projections. \( T \) is the length of the sequence. \( q_0 \) represent the original query and is used to retrieve a slot from memory \( h_0 \). The retrieved slot is combined with the original query using a residual connection and the result of this operation is a vector that define a proposed answer \( q_1 \). Based on this proposed answer the network decides, using REINFORCE [262] if another query in memory is necessary or the retrieved content is enough to answer, in this latter case the network decides to stop the memory re-circulation and an answer is given \( a \). (see 3.2.2 for the mathematical details)

Computer [141] and Universal Transformer [257]. We then describe the tasks used in the experiments: Paired Associative Inference (PAI), Shortest Path Graph Traversal and bAbI [259] and the training regimes.

### 3.2.1 Recapitulating End-to-End Memory Networks

We begin by describing End-to-End Memory Networks (EMN) [144, EMN], as a reminder of this architecture, to introduce notation and nomenclature, and also as a contrast to our work. We will focus on the multilayer, tied weight variant of EMN as this most closely resembles our architecture.

The set up used for the rest of the paper is as follows: given a set of knowledge inputs \( \{x_i\}_{i=1}^I = \{x_{i1}, x_{i2}, ..., x_{iS}\} \), and a query or question \( q = \{q_1, q_2, ..., q_S\} \in \mathbb{R}^S \), the network must predict the answer \( a \). \( I \) represents the length of the knowledge input sequence, and \( S \) is the length of each input sentence; for instance, in bAbI
[259] (see Methods 3.2.8 and Fig. 3.4 for details on bAbI), \( I \) will be the number of stories and \( S \) is the number of words in each sentence in a story. \( x_{is} \) will be the word in position \( s \) in the sentence, in the \( i \)th story and will be a \( O \)-dimensional one hot vector encoding one of \( O \) possible input words.

EMN embeds each word and sums the resulting vectors:

\[
    k_i = \sum_s l_s \cdot W_k x_{is} \quad (3.1)
\]
\[
    v_i = \sum_s W_v x_{is} \quad (3.2)
\]
\[
    q_0 = W_q q \quad (3.3)
\]

where \( W_k, W_v, W_q \in \mathbb{R}^{d \times O} \) are embedding matrices for the key, values and query, respectively. Also \( l_s \) is a positional encoding column vector (as defined in [144]), 

\[
    \cdot \n\]

represents an element wise multiplication and \( O \) is the size of the vocabulary.

At each step \( t \), EMN calculates the vector of weights over the memory elements \( k_i \) and produces the output. Let \( K \) be the \( I \times d \) matrix formed by taking \( k_i \) as its rows, and similarly \( V \) formed by \( v_i \) as rows, then:

\[
    w_t = \text{softmax}(Kq_t) \quad (3.4)
\]
\[
    q_{t+1} = w_t V + W_{qv} q_t \quad (3.5)
\]
\[
    a_t = \text{softmax}(W_a q_{t+1}) \quad (3.6)
\]

where \( w_t \in \mathbb{R}^I \) are weights over the memory slots, \( W_{qv}, W_a \in \mathbb{R}^{d \times d} \) is a linear mapping relating the query at the previous step to the current one, \( q_{t+1} \) is the query to be used at the next step, and \( a_t \) is the answer (usually only produced right at the end). EMN is trained via a cross entropy loss on \( a_t \) at the final step.
3.2.2 MEMO

MEMO embeds the input differently. First, a common embedding $c_i$, of size $S \times d_c$, is derived for each input matrix $x_i \in \mathbb{R}^{S \times O}$:

$$c_i = x_i W_c$$  \hfill (3.7)

where $W_c \in \mathbb{R}^{O \times d_c}$. Then each of these embeddings is adapted to either be a key or a value. However, contrary to EMN, we do not use hand-coded positional embeddings, instead the words in each sentence and their one-hot encoding in $x_i$, embedded as $c_i$, are combined and then this vector is passed through a linear projection followed by an attention mechanism (explained in detail below). This allows $c_i$ to capture flexibly any part of the input sentence in $x_i$. MEMO uses multiple heads to attend to the memory following [90]. A head is a learnt vector of importance weights that it is used to assigned a retrieval probability to each slot in memory. Each head has a different view of the same common inputs $c_i$. Let $H$ denote the total number of heads, and $h$ index the particular head, then for each $h \in \{1, \ldots, H\}$ we have:

$$k_i^{(h)} = W_k^{(h)} \text{vec}(c_i)$$ \hfill (3.8)

$$v_i^{(h)} = W_v^{(h)} \text{vec}(c_i)$$ \hfill (3.9)

$$q_0^{(h)} = W_q^{(h)} q$$ \hfill (3.10)

where $W_k^{(h)}, W_v^{(h)} \in \mathbb{R}^{d \times d_c}$ and $W_q^{(h)} \in \mathbb{R}^{d \times S}$ are embedding matrices for the key, values and query respectively. $\text{vec}(c)$ means flattening the matrix $c$ into a vector with the same number of elements as the original matrix, and $\text{vec}^{-1}(v)$ is the reverse operation of a vector $v$ into a matrix such that $\text{vec}^{-1}(\text{vec}(c)) = c$. The result is three $d$-dimensional vectors $k_i^{(h)}, v_i^{(h)}$ and $q_0^{(h)}$. Keeping each item separated into memory allow us to learn how to weight each of these items when we perform a memory lookup. This contrasts with the hand-coded positional embeddings used in EMN [144] and updated recently in [90], and proved critical for enabling the flexible recombination of the stored items.
The attention mechanism used by MEMO also differs from that shown above for EMN. Firstly, it uses multi-head attention, instead of just a single head. Secondly, we use DropOut [202] and Layer Normalisation [91] to improve generalisation and learning dynamics. With Dropout, some number of layer units, in our case 50% (otherwise differently stated), are randomly silenced during the training phase. Dropout has the effect of making the training process noisy, forcing units within a layer to probabilistically respond to their inputs. The idea behind this technique is to reduce correlation amongst units in the network, in turn making the model more robust to overfitting. Layer normalisation (LayerNorm) is a technique to normalise the distributions of intermediate layers. It enables smoother gradients, faster training, and better generalisation accuracy.

Let \( K^{(h)} \in \mathbb{R}^{I \times d} \) denote the matrix formed by taking each \( k_i^{(h)} \) as a row, and \( V^{(h)} \) be the matrix formed by taking each \( v_i^{(h)} \) as a row. In contrast, let \( Q_t \in \mathbb{R}^{H \times d} \) be the matrix formed by taking each \( q_t^{(h)} \) as the rows. The attention mechanism then becomes:

\[
h_t^{(h)} = \frac{1}{\sqrt{d}} W_h K^{(h)} q_t^{(h)} \quad (3.11)
\]

\[
w_t^{(h)} = \text{DropOut}(\text{softmax}(h_t^{(h)})) \quad (3.12)
\]

\[
q_t^{(h)} = w_t^{(h)} V^{(h)} \quad (3.13)
\]

\[
Q_{t+1} = \text{LayerNorm} \left( \text{vec}^{-1}(W_q \text{vec}(Q_{t+1})) + Q_t \right) \quad (3.14)
\]

\[
a_t = \text{softmax} \left( W_a \text{DropOut}(\text{relu}(W_q a \text{vec}(Q_{t+1}))) \right) \quad (3.15)
\]

where \( W_h \in \mathbb{R}^{I \times I}, W_q \in \mathbb{R}^{Hd \times Hd} \) are matrices for transforming the logits and queries respectively. \( W_a \in \mathbb{R}^{O \times d_a} \) and \( W_qa \in \mathbb{R}^{d_a \times Hd} \) are the matrices of the output MLP that produces the answer \( a_t \). It is worth noting, that even though our attention mechanism uses some of the feature implemented in [90] – i.e. normalization factor \( \sqrt{d} \) and multiheading, it differs from it because rather than doing self-attention it preserves the query separated from the keys and the values. This aspect is particularly important in terms of computational complexity, in that MEMO is linear with respect to the number of sentences of the input, whereas methods relying on
3.2. Methods

Self-attention [e.g. 257] have quadratic complexity (see Appendix B.3).

3.2.3 The Halting Policy

In the previous sections we described how MEMO can output a sequence of potential answers to an input query, here we describe how to learn the number of computational steps – hops – required to effectively answer a. To make this decision, we collect some information at every step and we use this to create an observation $s_t$. That observation is then processed by gated recurrent units (GRUs) [268] followed by a multilayer perceptron (MLP) which defines a binary policy $\pi(a|s_t, \theta)$ and approximates its value function $V(s_t, \theta)$. The input $s_t$ to this network is formed by the Bhattacharyya distance [269] between the attention weights of the current time steps $W_t$ and the ones at the previous time step, $W_{t-1}$ (both $W_t$ and $W_{t-1}$ are taken after the softmax), and the number of steps taken so far as a one-hot vector $t$. The idea behind the way we build $s_t$ is that if the attention is focused on the same slot of memory for too many consecutive steps then there is no reason to keep querying the memory because the information retrieved will be the same - i.e. the network has settled into a fixed point.

$$z_t = GRU_R(z_{t-1}, d(W_t, W_{t-1}), t) \quad (3.16)$$
$$v_t, \pi_t = MLP_R(z_t) \quad (3.17)$$
$$h_t = \sigma(\pi_t) \quad (3.18)$$

This network is trained using REINFORCE [262]. More concretely, the parameters $\theta$ are adjusted using a $n$-step look ahead values, $\hat{R}_t = \sum_{i=0}^{n-1} \gamma^i r_{t+i} + \gamma^n V(s_{t+n}, \theta)$, where $\gamma$ is the discount factor. The objective function of this network is to minimize: $\mathcal{L}_{\text{Hop-Net}} = - \mathbb{E}_{s_t \sim \pi} [\hat{R}_t]$, $\mathcal{L}_V = \mathbb{E}_{s_t \sim \pi} [(\hat{R}_t - V(s_t, \theta))^2]$ and $\mathcal{L}_{\text{Hop}} = - \mathbb{E}_{s_t \sim \pi} [\pi(\cdot|s_t, \theta)]$. Interestingly, $\mathcal{L}_{\text{Hop}}$ is a term that directly follows from the fact that $\pi$ is a binary policy. Specifically, the expectation of a binary random variable is its probability and the expectation of their sum is the sum of the expectation. Consequently, the new term that we introduce in the loss, $\mathcal{L}_{\text{Hop}}$ allows us to directly minimize the expected number of hops.
This term, similar in motivation to [267] (although differs mathematically), directly encourages our network to explicitly prefer representations and computation that minimise the amount of required computation. It is also worth noting that an argument against using REINFORCE when training discrete random variables is that the variance can be prohibitively high [270]. Interestingly, in the case of a binary halting random variable, the variance is just $p(1 - p)$ where $p$ is the probability of halting and the variance is bounded by $1/4$ which we find is not too large for learning to proceed successfully in practice.

Finally, the reward structure is defined by the answer $a$:

$$r_t = \begin{cases} 
1, & \text{if } \hat{a} = a \\
0, & \text{otherwise}
\end{cases}$$

where $a$ is the target answer associated with the input and $\hat{a}$ is the prediction from the network. The final layer of $MLP_R$ was initialized with $bias_{init}$, in order to increase the chances that $\pi$ produces a probability of 1 (i.e. completes more than one more hop). Finally, we set a maximum number of hops, $N$, that the network could take. If $N$ was reached, the network stopped performing additional hops. Critically, there was no gradient sharing between the hop network and the main MEMO network explained above. All model hyperparameters are reported in Appendix B.4.

### 3.2.4 Baselines

We compared MEMO with two other memory-augmented architectures: End to End Memory Networks (EMN) [144] and Differential Neural Computer [141]. We also compared MEMO with the Universal Transformer [257], the current state of the art model in the bAbI task suite [259].

#### 3.2.4.1 EMN

For EMN please refer to 3.2.1 and the hyper-parameters used are the same as in MEMO.
3.2.4.2 Differential Neural Computer architecture and hyperparameters

The Differentiable Neural Computer is a neural network which uses external memory slots as in MEMO. However, it is more expressive as it adds a controller, which is responsible for taking input in, reading from and writing to memory, and producing output that can be interpreted as an answer. A controller can perform several operations on memory. At every time step, it chooses whether to write to memory or not. If it chooses to write, it can choose to store information at a new, unused location or at a location that already contains information the controller is searching for. This allows the controller to update what is stored at a location. If all the locations in memory are used up, the controller can decide to free locations. As well as writing, the controller can read from multiple locations in memory using a multi-head attention mechanism, similar to the one implemented in MEMO.

In this work, we used the same architecture as described in [141], with exact sizes of each layer specified in Table B.4. We also optimised the hyperparameters searching across the ranges reported in Table B.5.

3.2.4.3 Universal Transformer architecture and hyperparameters

The Universal Transformer is an extension to the Transformer model [90] described in the introduction (see 1.4), which combines the features of the original transformer model with the recurrent connections as in standard recurrent neural networks. Specifically, in the standard Transformer, there is a pre-defined (using an hyper-parameter) stack of attention blocks, where each block is applied to all the input symbols in parallel. Instead, in the Universal Transformer, the attention block is iteratively applied to the input sequence for an arbitrary number of steps.

We used the same architecture as described in [257]. More concretely, we used the implementation and hyperparameters described as 'universal_transformer_small' that is available at https://github.com/tensorflow/tensor2tensor/blob/master/tensor2tensor/models/research/universal_transformer.py. For completeness, we describe the hyperparameters used in Table B.6. As before we optimised hyperparameters, exploring the ranges reported in Table B.7.
3.2.5 Training details

To train the MEMO network we use Adam [271] with polynomial learning rate decay, starting at $l_{start_{memo}}$ value, and batch size always equal to 64 for the PAI and shortest path and 128 for bAbI. In all the three tasks MEMO was trained using a cross entropy loss, and the network had to predict the class ID in the paired associative inference task, the node ID in the shortest path problem, and the word ID in bAbI. The halting policy network parameters were updated using RMSProp[210], with learning rate $l_{halt}$. The other parameters are reported in Table B.2 and B.3.

3.2.6 Paired associative inference task

![Paired associative inference](image)

**Figure 3.2:** Paired associative inference.

The panel on the left illustrates a memory store filled with random pairs of images. The panels to the right illustrate (from left to right) two ’direct’ queries (AB and BC) where no inference is required, and an ’indirect’ query (AC) where inference is required.

In this work we introduced a task, derived from neuroscience [249], to carefully probe the reasoning capacity of neural networks. This task is thought to capture the essence of reasoning the appreciation of distant relationships among elements distributed across multiple facts or memories. This process is formalized in a prototypical task widely used to study the role of the hippocampus in generalization the paired associative inference (PAI) task [249, 245, see Fig.3.2]. Here, two images are randomly associated together. For example, analogous to seeing a little girl with a woman as in the example in the introduction, in the PAI task the agent (human participant or a artificial neural network) would be presented with an image $A$, e.g. a woman, and image $B$, e.g. a girl, side by side. Later, in a separate event, the agent would be exposed to a second pair, the image $B$ again, but this time paired with a new image $C$, e.g. the other person. This is analogous to seeing the little girl a second time with a different person. During test time two types of query can be asked: direct
and indirect queries. Direct queries are a test of episodic memory as the answer relies on retrieving an episode that was experienced. In contrast, indirect queries require inference across multiple episodes. Here the network is presented with CUE, image A, and two possible choices: image C, the MATCH, that was originally paired with B; or another image C′, the LURE, which was paired with B′ – i.e. forming a different triplet A′ – B′ – C′. The right answer, C, can only be produced by appreciating that A and C are linked because they both were paired with B. This is analogous to the insight that the two people walking with the same little girl are likely to have some form of association.

To make this task challenging for a neural network we started from the ImageNet dataset [272]. We created three sets, training, validation, and test which used the images from the respective three sets of ImageNet to avoid any overlapping. All images were embedded using a pre-trained ResNet [92]. We generated 3 distinct datasets with sequences of length three (i.e. A – B – C), four (i.e. A – B – C – D) and five (i.e. A – B – C – D – E) items. Each dataset contains 1e6 training images, 1e5 evaluation images and 2e5 testing images. Each sequence was randomly generated with no repetition in each single dataset.

To explain how the batch was built we take an example with sequences of length, S, being equal to 3. Each batch entry is composed of a memory, a query and a target. In order to create a single entry in the batch we selected N sequences from the pool, with N = 16. First, we created the memory content with all the possible pairwise associations between the items in the sequence, e.g. A₁B₁ and B₁C₁, A₂B₂ and B₂C₂, ..., AₙBₙ and BₙCₙ. For S = 3, this resulted in a memory with 32 rows. Then we generated all the possible queries. Each query consists of 3 images: the cue, the match, and the lure. The cue is an image from the sequence (e.g. A₁), as is the match (e.g. C₁). The lure is an image from the same memory set but from a different sequence (e.g. Cₗ). There are two types of queries - 'direct' and 'indirect'. In 'direct' queries the cue and the match can be found in the same memory slot, so no inference is required. For example, the sequence A₁ - B₁ - C₁ produces the pairs A₁ - B₁ and B₁ - C₁ which are stored in different slots in memory. An example of a direct test trial would be A₁ (cue) - B₁ (match) - B₃ (lure). Therefore, 'direct' queries are a test of episodic memory as the answer relies on retrieving an episode that was experienced. In contrast, 'indirect' queries require inference across multiple episodes. For the previous example sequence, the inference trial would be A₁ (cue) - C₁ (match) - C₃ (lure). The queries are presented to the network as a concatenation of three image
embedding vectors (the cue, the match, and the lure). The cue is always in the first position in the concatenation, but to avoid any degenerate solution, the position of the match and lure are randomised. It is worth noting that the lure image always has the same position in the sequence (e.g. if the match image is a C the lure is also a C) but it is randomly drawn from a different sequence that is also present in the current memory. This way the task can only be solved by appreciating the correct connection between the images, and this needs to be done by avoiding the interference coming for other items in memory. For instance in Fig 3.2 the items $C_1$ and $C_2$ are both coming from the same memory content, but from different slots. For each entry in the batch we generated all possible queries that the current memory store could support and then one was selected at random. Also the batch was balanced, i.e. half of the elements were direct queries and the other half was indirect. The targets that the network needed to predict are the class of the matches.

It is worth mentioning that longer sequences provide more 'direct' queries, but also multiple 'indirect' queries that require different levels of inference, e.g. the sequence $A_n - B_n - C_n - D_n - E_n$ produces the 'indirect' trial $A_1$ (cue) - $C_1$ (target) - $C_3$ (lure) with 'distance' 1 (one pair apart) and $A_1$ (cue) - $E_1$ (target) - $E_3$ (lure) with 'distance' 4 (4 pairs apart). The latter is obviously harder as more inference steps are required to appreciate the overlap between memories. Finally we use the inputs as follows:

- For EMN and MEMO, the memory content and the query are used as direct inputs to the model (see 3.2.2 for details).

- In the case of Differential Neural Computer (section 3.2.4.2), we embed stories and query in the same way as for MEMO. Memory and query are presented in sequence to the model (in that order), followed by blank inputs as pondering steps to provide a final prediction.

- For Universal Transformer, we embed stories and queries in the same way as for MEMO. Then we used the encoder of Universal Transformer with architecture described in Section 3.2.4.3.
3.2.7 Shortest path task

![Figure 3.3: Example of Graph.](image)

a. Illustration of a graph with 6 nodes and $K = 2$. b. Illustration of the shortest path (in purple) between node 1 and node 5. The length of the shortest path is equal to 2.

**Graph generation** In the shortest path experiments, we generate the graph in the same fashion as [141]: the graphs used to train the networks are generated by uniformly sampling a set of two-dimensional points from a unit square, each point corresponding to a node in the graph. For each node, the $K$ nearest neighbours in the square are used as the $K$ outbound connections, with $K$ independently sampled from a uniform range for each node (see an example in Fig. 3.3).

**Graph representation** We represent our task in three parts: a graph description, a query, and the target. The graph description is presented a sequence of tuples of integers that represent a connection between nodes, holding a token for the source node, and another for the destination node. The query is also represented as a tuple of integers, although, in that case, source and destination are simply the beginning and end of the path to find. The target is the sequence of node IDS that constitute the path between source and destination of the query.

When training, we sample a mini-batch of 64 graphs, with associated queries and target paths. Following our description above, queries are represented as a matrix of size $64 \times 2$, targets are of size $64 \times (L - 1)$, and graph descriptions are of size $64 \times M \times 2$, where $L$ is the length of the shortest path, and $M$ is the number of maximum nodes we allow one graph description to have. In our experiments, we fix the upper bound $M$ to be the maximum number of nodes that we have multiplied by the out-degree of the nodes in the graph.

All networks were trained for $2e4$ epochs, each one formed by 100 batch updates.

- For EMN and MEMO, we set the graph description to be the contents of their memory, and we use the query as input. In order to answer the sequence of nodes that
is used as target, we keep the keys $k_i^{(h)}$ and values $v_i^{(h)}$ fixed, and we proceed to use our algorithm as described for each answer, with independent numbers of hops for each one. The model then predict the answers for nodes sequentially: the first node is predicted before the second. However, one important difference between MEMO and EMN is that for EMN we use the ground truth answer of the first node as the query for the second node, whereas for MEMO we used the answer predicted by the model for the first node as the query for the second node. This was done to enhance the performance of EMN while testing the real capabilities of MEMO to sequentially reasoning over multiple steps problems. The weights that are used for each answer are not shared.

• For the Universal Transformer, we also embed the query and graph description as done for EMN and MEMO. After that, we concatenate the embeddings of query and graph description and use the encoder of the Universal Transformer architecture (with specific description in Section 3.2.4.3). We use its output as answer. After providing an answer, that answer is provided as initial query for the following round of hops. The weights that are used for each answer are not shared.

• For Differential Neural Computer, we also embed the query and graph description as done for EMN and MEMO. Since it is naturally a sequential model, the information is presented differently: the tuples of the graph description are presented first, and after that the query tuple is presented. Finally, the pondering steps are used to be able to output the sequence of nodes that constitute the proposed shortest path.

The output of the models was trained using Adam with a cross-entropy loss against all the sampled target sequences. Training is done for a fixed number of steps, detailed in Appendix Section B.4.

For evaluation, we sample a batch of 600 graph descriptions, queries, and targets. We evaluate the mean accuracy over all the nodes of the target path. We report average values and standard deviation over the best 5 hyper parameters we used.

It is worth noting that given this training regime:

• Differential Neural Computer and Universal Transformer have a global view on the problem in order to provide an answer for the second node. This means that, to answer the second node in the path, they can still reason and work backwards from
the end node, and while still having information about the initial node in the path. This makes it intuitive for them to achieve better performance in the second node, as it is closest to the end node of the path, so less reasoning is needed to achieve good performance.

- On the contrary, MEMO has a local view of the problem, the answer to the second node depends on the answer about the first node. Therefore, it cannot exceed chance if the answer to the first node is not correct.

### 3.2.8 bAbI - Training and evaluation details

**a**

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Mary moved to the bathroom.</td>
</tr>
<tr>
<td>2</td>
<td>John went to the hallway.</td>
</tr>
<tr>
<td>3</td>
<td>Where is Mary? bathroom</td>
</tr>
<tr>
<td>4</td>
<td>Daniel went back to the hallway.</td>
</tr>
<tr>
<td>5</td>
<td>Sandra moved to the garden.</td>
</tr>
<tr>
<td>6</td>
<td>Where is Daniel? hallway</td>
</tr>
<tr>
<td>7</td>
<td>John moved to the office.</td>
</tr>
<tr>
<td>8</td>
<td>Sandra journeyed to the bathroom.</td>
</tr>
<tr>
<td>9</td>
<td>Where is Daniel? hallway</td>
</tr>
<tr>
<td>10</td>
<td>Mary moved to the hallway.</td>
</tr>
<tr>
<td>11</td>
<td>Daniel travelled to the office.</td>
</tr>
<tr>
<td>12</td>
<td>Where is Daniel? office</td>
</tr>
<tr>
<td>13</td>
<td>John went back to the garden.</td>
</tr>
<tr>
<td>14</td>
<td>John moved to the bedroom.</td>
</tr>
<tr>
<td>15</td>
<td>Where is Sandra? bathroom</td>
</tr>
</tbody>
</table>

**b**

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Mary moved to the bathroom.</td>
</tr>
<tr>
<td>2</td>
<td>Sandra journeyed to the bedroom.</td>
</tr>
<tr>
<td>3</td>
<td>John went to the kitchen.</td>
</tr>
<tr>
<td>4</td>
<td>Mary took the football there.</td>
</tr>
<tr>
<td>5</td>
<td>How many objects is Mary carrying? one</td>
</tr>
<tr>
<td>6</td>
<td>Sandra went back to the office.</td>
</tr>
<tr>
<td>7</td>
<td>Daniel went back to the office.</td>
</tr>
<tr>
<td>8</td>
<td>How many objects is Mary carrying? one</td>
</tr>
<tr>
<td>9</td>
<td>John moved to the bedroom.</td>
</tr>
<tr>
<td>10</td>
<td>Sandra moved to the garden.</td>
</tr>
<tr>
<td>11</td>
<td>How many objects is Mary carrying? one</td>
</tr>
<tr>
<td>12</td>
<td>Mary travelled to the garden.</td>
</tr>
<tr>
<td>13</td>
<td>Mary went to the hall way.</td>
</tr>
<tr>
<td>14</td>
<td>Sandra journeyed to the bedroom.</td>
</tr>
<tr>
<td>15</td>
<td>Mary dropped the football.</td>
</tr>
<tr>
<td>16</td>
<td>How many objects is Mary carrying? none</td>
</tr>
<tr>
<td>17</td>
<td>Mary got the football there.</td>
</tr>
<tr>
<td>18</td>
<td>Daniel travelled to the garden.</td>
</tr>
<tr>
<td>19</td>
<td>How many objects is Mary carrying? one</td>
</tr>
</tbody>
</table>

**Figure 3.4:** bAbI task.

a. Example of bAbI story from the task called a “single supporting fact”. To answer the query only one supporting sentence is needed, but it is requested to avoid distractions. b. Example of bAbI story from the task called “counting”

One of the main purpose of the research on natural language processing is to design a system that can generically perform a set of question answering problems. Following this spirit, the bAbi (not a acronym) tasks are a synthetic dataset of 20 tasks released by the Facebook AI Research team that help evaluate systems in this domain [259] (see and example of two stories from two different task in Fig. 3.4).
3.2. Methods

For this experiment we used the English Question Answer dataset [259]. We use the training and test datasets that they provide with the following pre-processing:

- All text is converted to lowercase.

- Periods and interrogation marks were ignored.

- Blank spaces are taken as word separation tokens.

- Commas only appear in answers, and they are not ignored. This means that, e.g. for the path finding task, the answer 'n,s' has its own independent label from the answer 'n,w'. This also implies that every input (consisting of 'query' and 'stories') corresponds to a single answer throughout the whole dataset.

- All the questions are stripped out from the text and provided separately (given as "queries" to our system).

At training time, we sample a mini-batch of 128 queries from the test dataset, as well as its corresponding stories (which consist of the text prior to the question). As a result, the queries are a matrix of $128 \times 11$ tokens, and sentences are of size $128 \times 320 \times 11$, where 128 is the batch size, 320 is the max number of stories, and 11 is the max sentence size. We pad with zeros every query and group of stories that do not reach the max sentence and stories size.

- For EMN and MEMO, the memory content and the query are used as direct inputs to the model (see 3.2.2 for details).

- In the case of Differential Neural Computer, we embed stories and queries in the same way as for MEMO. Stories and queries are presented in sequence to the model (in that order), followed by blank inputs as pondering steps to provide a final prediction.

- For Universal Transformer, we embed stories and queries in the same way as for MEMO. Then, we use the encoder of Universal Transformer with architecture described in Section 3.2.4.3. We use its output as the output of the model.

After the mini-batch is sampled, we perform one optimization step using Adam for all the models that we have run in our experiments. The hyper parameters are detailed in Appendix Section B.4. We stop after a fixed number of time-steps, as also detailed in B.4.
3.3 Results

Many of the tasks in bAbI require some notion of temporal context, to account for this in MEMO we added a column vector to the memory store. This vector adds a piece of information to each word embedding about its position in the sentence. This vector is called the positional encoding and after [90] is defined in the following way. Let $t$ be the desired position in an input sentence, $\mathbf{p}_t \in \mathbb{R}^d$ be its corresponding encoding, and $d$ be the encoding dimension (i.e. the number of memory slots in MEMO) Then $f : \mathbb{N} \rightarrow \mathbb{R}^d$ will be the function that produces the positional vector $\mathbf{p}_t$ and it is defined as follows:

$$
\mathbf{p}_t^{(i)} = f(t)^{(i)} := \begin{cases} 
\sin(\omega_k t), & \text{if } i = 2k \\
\cos(\omega_k t), & \text{if } i = 2k + 1
\end{cases}
$$

(3.19)

where: $\omega_k = \frac{1}{\sqrt{10000^2 + k^2}}$

All networks were trained for $2e4$ epochs, each one formed by 100 batch updates. For evaluation, we sample a batch of 10,000 elements from the dataset and compute the forward pass in the same fashion as was done in training. With that, we compute the mean accuracy over those examples, as well as the accuracy per task for each of the 20 tasks of bAbI. We report average values and standard deviation over the best 5 hyper parameters we used.

3.3 Results

In this section we present the results on three different tasks: Paired associative inference, shortest path graph traversal and bAbI question answering.

3.3.1 Paired associative inference

<table>
<thead>
<tr>
<th>Length</th>
<th>EMN</th>
<th>DNC</th>
<th>UT</th>
<th>MEMO</th>
</tr>
</thead>
<tbody>
<tr>
<td>3 items (set: A-B-C - accuracy on A-C)</td>
<td>61.01</td>
<td>96.85</td>
<td>85.60</td>
<td>98.26(0.67)</td>
</tr>
<tr>
<td>4 items (set: A-B-C-D - accuracy on A-D)</td>
<td>48.66</td>
<td>51.56</td>
<td>44.16</td>
<td>97.22(0.13)</td>
</tr>
<tr>
<td>5 items (set: A-B-C-D-E - accuracy on A-E)</td>
<td>45.13</td>
<td>62.61</td>
<td>47.93</td>
<td>84.54(5.72)</td>
</tr>
</tbody>
</table>

Test results for the best 10 hyper-parameters (chosen by validation loss) for MEMO. For EMN and Differential Neural Computer (DNC) and Universal Transformer (UT) results on the best run (chosen by validation loss). The score represent the percentage accuracy on the test set. Chance level is 50%. For MEMO we also report the standard deviation in brackets for the best 10 hyper-parameters (chosen by validation loss).
3.3. Results

Table 3.1 reports the summary of results of our model (MEMO) and the other baselines on the hardest inference query for each of the PAI tasks. On the smaller set - i.e. A-B-C - MEMO was able to achieve the highest accuracy together with Differential Neural Computer, whereas EMN, even with 10 hops, wasn’t able to achieve the same level of accuracy, also Universal Transformer was not able to accurately solve this inference test. For longer sequences -i.e. length 4 and 5 - MEMO was the only architecture which successfully answered the most complex inference queries, with an accuracy of 84.54(SD=5.72), whereas all the other architecture performed almost at chance level.

<table>
<thead>
<tr>
<th>Trial Type</th>
<th>Type</th>
<th>EMN</th>
<th>DNC</th>
<th>UT</th>
<th>MEMO</th>
</tr>
</thead>
<tbody>
<tr>
<td>A-B</td>
<td>98.19</td>
<td>98.58</td>
<td>97.43</td>
<td>99.82(0.30)</td>
<td></td>
</tr>
<tr>
<td>B-C</td>
<td>97.93</td>
<td>99.34</td>
<td>98.28</td>
<td>99.76(0.38)</td>
<td></td>
</tr>
<tr>
<td>A-C</td>
<td>61.01</td>
<td>96.85</td>
<td>85.60</td>
<td>98.26(0.67)</td>
<td></td>
</tr>
</tbody>
</table>

Table 3.2: Test results - length 3: A-B-C

<table>
<thead>
<tr>
<th>Trial Type</th>
<th>Type</th>
<th>EMN</th>
<th>DNC</th>
<th>UT</th>
<th>MEMO</th>
</tr>
</thead>
<tbody>
<tr>
<td>A-B</td>
<td>96.31</td>
<td>94.26</td>
<td>99.32</td>
<td>99.57(0.20)</td>
<td></td>
</tr>
<tr>
<td>B-C</td>
<td>97.57</td>
<td>84.94</td>
<td>88.31</td>
<td>99.33(0.13)</td>
<td></td>
</tr>
<tr>
<td>C-D</td>
<td>96.59</td>
<td>95.68</td>
<td>93.37</td>
<td>99.58(0.13)</td>
<td></td>
</tr>
<tr>
<td>A-C</td>
<td>48.71</td>
<td>49.38</td>
<td>54.87</td>
<td>98.93(0.15)</td>
<td></td>
</tr>
<tr>
<td>B-D</td>
<td>47.42</td>
<td>49.89</td>
<td>51.92</td>
<td>99.14(0.19)</td>
<td></td>
</tr>
<tr>
<td>A-D</td>
<td>48.63</td>
<td>58.63</td>
<td>44.16</td>
<td>97.22(0.13)</td>
<td></td>
</tr>
</tbody>
</table>

Table 3.3: Test results - length 4: A-B-C-D

Tables 3.2, 3.3, 3.4 report the full set of results on both direct and indirect trials for length 3, 4, and 5 respectively.
### 3.3. Results

#### Table 3.4: Test results - length 4: A-B-C-D-E

<table>
<thead>
<tr>
<th>Trial Type</th>
<th>EMN</th>
<th>DNC</th>
<th>UT</th>
<th>MEMO</th>
</tr>
</thead>
<tbody>
<tr>
<td>A-B</td>
<td>95.68</td>
<td>98.88</td>
<td>96.94</td>
<td>99.20(0.43)</td>
</tr>
<tr>
<td>B-C</td>
<td>95.82</td>
<td>94.60</td>
<td>92.63</td>
<td>98.93(0.17)</td>
</tr>
<tr>
<td>C-D</td>
<td>95.43</td>
<td>95.20</td>
<td>89.99</td>
<td>97.27(0.21)</td>
</tr>
<tr>
<td>D-E</td>
<td>95.16</td>
<td>95.98</td>
<td>97.27</td>
<td>95.06(0.12)</td>
</tr>
<tr>
<td>A-C</td>
<td>48.68</td>
<td>48.66</td>
<td>41.85</td>
<td>87.33(0.12)</td>
</tr>
<tr>
<td>B-D</td>
<td>45.75</td>
<td>46.87</td>
<td>39.62</td>
<td>86.65(1.27)</td>
</tr>
<tr>
<td>C-E</td>
<td>49.46</td>
<td>49.51</td>
<td>35.87</td>
<td>87.08(0.92)</td>
</tr>
<tr>
<td>A-D</td>
<td>52.08</td>
<td>50.32</td>
<td>52.38</td>
<td>86.12(0.57)</td>
</tr>
<tr>
<td>B-E</td>
<td>46.69</td>
<td>52.27</td>
<td>43.27</td>
<td>86.37(0.77)</td>
</tr>
<tr>
<td>A-E</td>
<td>48.31</td>
<td>48.79</td>
<td>47.93</td>
<td>84.54(5.72)</td>
</tr>
</tbody>
</table>

Test results for the best 10 hyper-parameters (chosen by validation loss) for MEMO. For EMN and Differential Neural Computer (DNC) and Universal Transformer (UT) results on the best run (chosen by validation loss). The score represent the percentage accuracy on the test set. Chance level is 50%. For MEMO we also report the standard deviation in brackets for the best 10 hyper-parameters (chosen by validation loss).

---

**Figure 3.5: Number of Hops**

The histogram reports the 36 different set of hyper-parameters used for MEMO that solve the inference trials successfully – i.e. accuracy above 95% – out of a total of 48 hyper-parameters run.

To further investigate how these results were achieved, we ran further analyses on the
3.3. Results

length 3 PAI task. Interestingly, to solve this task Differential Neural Computer required 10 pondering steps to solve the inference trials. MEMO instead converged on a median of 3 hops (see Fig. 3.5). To understand how MEMO approached this task we then analysed the attention weights of an inference query, where the goal was to associate a CUE with the MATCH and avoid the interference of the LURE (see 3.2.6 for task details) and we consistently found the same pattern of results, of which an example is presented in 3.6. For clarity we report here the original sequence $A - B - C$, respectively composed by the following class IDs: 611 – 191 – 840 (however this sequence was not directly experienced together by the network, as the two associations $A - B$ and $B - C$ were stored in slot 10 and 25, respectively). As depicted in Figure 3.7, in the first hop MEMO retrieved the memory in slot 10, which contained the CUE, ID 611, and the associated item, ID 191, which form an $A - B$ association. Then in the following hop this slot was partially active, but most of the mass was placed on slot 16, which contained the memory association $B - C$; that is, ID 191 and ID 840, the MATCH. Interestingly, slot 13 which was associated with the LURE, ID 943, saw some probability mass associated with it. Hence, in this second hop MEMO assigned appropriate probability masses to all the slots needed to support a correct inference decision, which was then confirmed in the last hop. This sequence of memories activation is reminiscent of the one predicted by computational model of the hippocampus [254, 255] and observed in neural data [256]. Moreover, another instance of MEMO which used 7 hops, to solve the task with the same level of accuracy, presented a very different pattern of memory activation (see Fig.B.1 in the Appendix B.1.1). This is an indication of the fact that the algorithm used to solve the inference problem depends on how many hops the network takes. This aspect could also be related to knowledge distillation in neural networks [273, 274], whereby many hops are used to initially solved the task (i.e. over-parametrization) and then these are automatically reduced to use less computation (see Fig. 3.6).
3.3. Results

Figure 3.6: Analysis of length 3 PAI task.

a. Evaluation accuracy on the inference trial A-C; b. Number of hops taken during training; c. Distribution of evaluation accuracy obtained by averaging direct queries (A-B and B-C). This was obtained over 100 different hyper-parameters and seeds; d. same as c, but on the inference queries (A-C)

To better understand what parts of MEMO were critical to its performance we also ran a set of ablation experiments, the results of which are presented in Table 3.5. This analysis confirmed that it is the combination of the specific memory representations (i.e. facts kept separated) and the recurrent attention mechanism that support successful inference – i.e. employing these two components individually was not enough. Interestingly, this conclusion was valid only for inference queries, not for direct queries (see Fig. 3.6c,d). Indeed, by definition, direct queries are a pure test of episodic memory and so can be solved with a single memory look-up. Finally, we also compared our adaptive computation mechanism with ACT [260] and we found that, for this task, our method was more data efficient (see Fig. B.2 in Appendix B.1.2).
3.3. Results

Figure 3.7: Weights analysis of an inference query in the length 3 PAI task.

An example of memory content and related inference query is reported in the first column on the left. For clarity we report image class ID. Cue and Match are images from the same sequence e.g. $A_{10} - C_{10}$, where 10 is the slot ID. The lure is an image presented in the same memory store, but associated with a different sequence, e.g. $C_{13}$. The 3 most right columns report the weights associated with the 3 hops used by the network, for each probability mass we report the associated retrieved slot.

Table 3.5: PAI - Ablations - sequence of length 3: A-B-C

<table>
<thead>
<tr>
<th>MEMO Network Architecture</th>
<th>A-C inference trial</th>
</tr>
</thead>
<tbody>
<tr>
<td>Positional encoding as in [90]</td>
<td>Memories kept separated</td>
</tr>
<tr>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

Test results for the best 10 hyper-parameters (chosen by validation loss) for MEMO. The score represent the percentage accuracy on the test set for the A-C inference trial. Chance level is 50%. Standard deviation is reported

✓= not present; ✓= present

3.3.2 Shortest path on randomly generated graphs

To prove the capability of MEMO on a standard test of reasoning [141], we then turn to a set of synthetic reasoning experiments on randomly generated graphs (see 3.2.7 for details). Table 3.6 shows the accuracy of the models on the task related to finding the shortest path.
### 3.3. Results

#### Table 3.6: Undirected graph - shortest path

<table>
<thead>
<tr>
<th>Graph Structure</th>
<th>Prediction of First Node</th>
<th>Prediction of Second Node</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>EMN</td>
<td>UT</td>
</tr>
<tr>
<td>Nodes</td>
<td>Out-degree</td>
<td>Path length</td>
</tr>
<tr>
<td>10</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>20</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>20</td>
<td>5</td>
<td>3</td>
</tr>
</tbody>
</table>

Test results for the best 5 hyper-parameters (chosen by training loss) for MEMO, mean and corresponding standard deviation are reported. For EMN, Universal Transformer (UT), and Differential Neural Computer (DNC) we report results from the best run.

between two nodes. On a small graph with 10 nodes, with a path length of 2 and 2 outgoing edges per node, Differential Neural Computer, the Universal Transformer, and MEMO achieved perfect accuracy in predicting the intermediate shortest path node. However, on more complicated graphs (20 nodes, 3 separated outgoing edges), with a path length of 3, MEMO outperformed EMN in predicting the first node of the path (31.99% vs. 94.40%, with chance level being 20%), and, similarly to Differential Neural Computer, almost completely solved the task. Additionally, MEMO outperformed Differential Neural Computer in more complicated graphs with a high degree of connectivity (5 out-degree), being better by more than 20% at predicting both nodes in the shortest path. This showed the great scalability that MEMO holds - the model was able to iterate and considered more paths as the number of hops increases.

To better compare the performance of MEMO versus EMN we ran additional experiments to test the models in two further conditions:

- The ground truth answer of the first node was used as the query for the second node.
- The answer predicted by the model for the first node was used as the query for the second node.

The results are summarized in Table 3.7. In the case of 20 Nodes with 5 outbound edges, we can see that if we give MEMO the ground truth for node 1 as query for node 2 the performance increases from the one related to the prediction of the first node (85.38%(0.05) vs. 69.20%(0.07)). Interestingly, if we use for EMN the same training regime used for MEMO - i.e. the prediction is used to query the second node - then EMN performs almost at chance level (22.30% vs. chance level of 20%) down from 43.00% obtained when using the ground truth answer.
Table 3.7: Undirected graph - second node analysis

<table>
<thead>
<tr>
<th>Graph Structure</th>
<th>Prediction of First Node</th>
<th>Prediction of Second Node</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nodes</td>
<td>Out-degree Path length</td>
<td>EMN MEMO</td>
</tr>
<tr>
<td>20</td>
<td>5</td>
<td>23.99 69.20(0.07)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>EMN ground truth</td>
</tr>
<tr>
<td></td>
<td></td>
<td>EMN predicted answer</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MEMO ground truth</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MEMO predicted answer</td>
</tr>
<tr>
<td></td>
<td></td>
<td>EMN predicted answer</td>
</tr>
</tbody>
</table>

Comparing results on the second node based on using ground truth or predicted answer of the first node. Test results for the best 5 hyper-parameters (chosen by training loss) for MEMO, mean and corresponding standard deviation are reported. For EMN we report results from the best run.

3.3.3 Question answering on the bAbI tasks

Finally, we turn our attention to the bAbI question answering dataset [259], which consists of 20 different tasks. In particular we trained our model on the joint 10k training set (specifics of training are reported in 3.2.8).

Table 3.8: bAbI - joint training

<table>
<thead>
<tr>
<th>Length</th>
<th>Memory Networks</th>
<th>DNC</th>
<th>Universal Transformer</th>
<th>Memo</th>
</tr>
</thead>
<tbody>
<tr>
<td>10k</td>
<td>4.2 (17/20)</td>
<td>3.8 (18/20)</td>
<td>0.29 (20/20)</td>
<td>0.21 (20/20)</td>
</tr>
</tbody>
</table>

Average error and in parentheses the number of failed tasks (≤ 5% error) out of 20 (lower is better in both cases) on the bAbI dataset. Results are shown for the best run (chosen by validation loss) as standard practice for bAbI. The full set of results is presented in B.2.1 Differential Neural Computer (DNC) results from [141], Universal Transformer results from [257].

Table 3.8 reports the averaged accuracy of our model (MEMO) and the other baselines on bAbI (the accuracy for each single task averaged across the best set of hyper-parameters is reported in Table B.1 in the Appendix B.2.1). In the 10k training regime MEMO was able to solve all tasks, thereby matching the number of tasks solved by [258, 257], but with a lower error (for single task results refer to Appendix B.1).
3.4. Discussion

Table 3.9: bAbI - Ablations

<table>
<thead>
<tr>
<th>MEMO Network Architecture</th>
<th>bAbI 10K</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Number of</td>
</tr>
<tr>
<td></td>
<td>solved tasks</td>
</tr>
<tr>
<td>Positional encoding as in</td>
<td></td>
</tr>
<tr>
<td>[90]</td>
<td></td>
</tr>
<tr>
<td>Memories keep separated</td>
<td></td>
</tr>
<tr>
<td>√</td>
<td>11/20</td>
</tr>
<tr>
<td>×</td>
<td>14/20</td>
</tr>
<tr>
<td>√</td>
<td>17/20</td>
</tr>
<tr>
<td>×</td>
<td>18/20</td>
</tr>
<tr>
<td>×</td>
<td>20/20</td>
</tr>
<tr>
<td>Recurrent attention</td>
<td></td>
</tr>
<tr>
<td>√</td>
<td></td>
</tr>
<tr>
<td>×</td>
<td></td>
</tr>
<tr>
<td>Layernorm</td>
<td></td>
</tr>
<tr>
<td>√</td>
<td></td>
</tr>
<tr>
<td>×</td>
<td></td>
</tr>
</tbody>
</table>

Results for the best run (chosen by validation set) on the bAbI task. The model was trained and tested jointly on all tasks. All tasks received approximately equal training resources. √= not present; ✓ = present

We also ran an extensive set of ablations experiments to understand the contribution of each architectural component to the final performance, the results of which are reported in Table 3.9. As observed previously it was the combination of memory representations and the powerful recurrent attention that was critical to achieve state of the art performance on bAbI. Finally the use of layernorm [91] in the recurrent attention mechanism was critical to achieve a more stable training regime and so better performance.

3.4 Discussion

In this work we employed a classic associative inference task from the memory-based reasoning neuroscience literature, the Paired Associative Inference task [249], to more carefully probe the reasoning capacity of existing artificial neural networks. This task is thought to capture the essence of reasoning – the appreciation of distant relationships among elements distributed across multiple facts or memories. Surprisingly, we found that current architectures struggle to reason over long distance associations. Similar results were obtained on a more complex task involving finding the shortest path between nodes in a path. We therefore developed MEMO, an architecture endowed with the capacity to reason over longer distances. This was accomplished with the addition of two novel components. First, it introduces a separation between memories/facts stored in external memory and the items that comprise these facts in external memory. Second, it makes use of an adaptive retrieval mechanism, allowing a variable number of memory hops before the answer is produced.
MEMO was capable of solving our novel reasoning tasks and matched state of the art results in a challenging language task, bAbI (see Methods 3.2.8 and Fig. 3.4 for an example).

One important new feature of this model is that single items remain stored separately in memory and they are not combined together using, for instance, positional embedding techniques. The idea is that preserving the pattern separated items in memory will allow more flexibility at the time of retrieval, as these items can be recombined in novel ways to solve previously unseen queries. This intuition was confirmed by ablation experiments which showed that it was the combination of a powerful attention mechanism and the pattern separated codes that gave rise to generalisation at test time. In contrast, when the items in memory were merged together even attention was not enough to solve the inference trials. On one hand, these results suggest that the role of the hippocampus in creating pattern separated codes to support episodic memory is not in contrast with its role in supporting generalisation [254, 245, 255, 256]. On the other hand, these findings also challenged the current machine learning practice of using a hand coded positional embedding to merge the items in memory - we show this greatly reduced model flexibility.

The work also complements that presented in chapter 2, as it is focused on the more general role of the hippocampus in supporting declarative memory rather than being narrowly focused on navigation. Indeed, as shown in figure 3.7, the attention mechanism implemented in MEMO learned to establish relations between events experienced at different points in time, and it did so by rapidly forming associations between its inputs and reactivated relational memories. This result is in line with the view that the hippocampus is a general relational processing machine that primarily supports declarative memory [275] by creating events based on the relations among the objects that occurred together in a certain context. Events are then chained together as they happen in time to form episodes, and finally relational networks are drawn as links between events and episodes to support inference between events that have not been experienced together [252].

The spatial and declarative views of the hippocampus do not need to be in contrast, on the contrary they might be underlain by a common mechanism [248]. Indeed, navigation can be framed as just a special case where memory is involved, rather than being a distinct process that requires the hippocampus to perform ad hoc computations [248]. This idea of a common mechanism is also more in line with the view that Tolman proposed in his original work [163], where he argued that the ability of humans and other animals to perform
complex inference task is based on an structured organisation of knowledge, which he called a *cognitive map*. Despite what the term *map* might suggests, Tolman did not limit his ideas to spatial reasoning, on the contrary he argued that these maps extend to abstract space and they help to organise memories. However, the particular mechanism by which the hippocampus might support both navigation and relational reasoning remains elusive. Recently, several attempts have been made to bridge this gap. One example is represented by the idea that spatial and non-spatial tasks can be represented as a graph problem for which grid and place cells are seen as a basis set that support an efficient representation of this graph [234]. Another compelling view is given by the recent Tolman-Eichenbaum Machine [276, 277], which is a generative model augmented with memory that by separating the structural representation common across task from the sensory representation specific to each task is able to support rapid generalisation across both spatial and non-spatial domains. MEMO is another instance in this line of thinking whereby the hippocampus is represented by the slot based component of the architecture, which allow for the rapid encoding of single episodes, and the entorhinal cortex is pictured by the recurrent attention mechanism, which is biased towards learning the relations between different episodes to support the rapid inference and generalisation characteristic of human reasoning ability. In particular, casting the learning of relational links as an attentional problem is a unique feature of our model as we believe it could be a more general and scalable approach, this point will be further expanded in the general conclusion.
Chapter 4

General Conclusions

In this work we put forward the argument that artificial neural network are a powerful class of models that can be used to understand and explain the mechanisms that give rise to cognition. We provided two stream of evidence for this, the first used deep reinforcement learning to understand the role of grid cells in spatial navigation, whereas the second investigated inferential reasoning under the lens of memory-augmented neural network. In this final chapter we will again use the artificial neural network framework proposed in the introduction to shed some light on the tension between the spatial and declarative view of the hippocampal-entorhinal functions.

4.1 Concept learning and manipulation

Many believe that the remarkable ability of humans, and some other animals like elephants, canids and cetaceans [278], to learn and perform a wide variety of tasks is based on the ability to create mental models of the reality. These models are then used to efficiently simulate the real world and reason about possible future outcomes to inform decisions [279]. Critically, these mental models are based on abstractions of reality, what we call concepts, which are the essence of intelligence as they support generalisation from limited experience, reasoning, planning, imagination, language, and creativity [280]. Concepts can be seen as the fundamental units of our thoughts that we, as others [e.g. 281, 282, 283], believe should be a key element of artificial agent design.

Interestingly, concepts learning and manipulation can also help to illuminate the tension between the declarative and spatial view of the hippocampus. Indeed, there have been recent proposals to extend the role of place and grid cells beyond their role in spatial navigation to encompass a wide spectrum of cognitive functions [244, 284, 285, 277]. The
4.1. Concept learning and manipulation

Proposal is that place and grid cells provide a metric code for abstract reasoning, and that there is a straightforward overlap from representations of two dimensional physical space to the ones for abstract spaces. Although intriguing, we do not think that such unequivocal mapping exist, and one reason to believe so comes from the research on natural language processing. Language is a very clear example of high level cognitive function, it is based on abstractions, and it is generally considered one of the hallmark of human intelligence. Language has also been an active topic of research in the artificial neural network community [e.g. 84, 90]. One of the most successful frameworks in this domain are recurrent neural networks trained to predict the probability for any word in the vocabulary to be the next word in a sentence. In doing so, the networks normally learns to represent the words as distributed representation that contain information both about an individual word and how words relate to each other [84]. These representations not only contrasts with a sparse code like the place cells one, but also they do not map onto the two-dimensional manifold required to support the hypothesis presented by Bellmund and colleague. Possibly more compelling is the connection with the results we presented in chapter 2, where a very similar network to the one we just describe for language is used to predict which location would be the most active out of all the possible ones in a room (a task akin to next word prediction). In this case the representations developed by the network are indeed the ones found in the spatial literature (see Chapter 2). Hence, using these two examples we can think about what the key differences are between the two tasks that result in the same network producing such different solutions. In both examples we have the same learning rules, the same architectures, and the same objective functions - the main difference is the dataset used, so it is worth exploring this dimension to garner further insights.

For tasks like two-dimensional navigation, it is straightforward to uniformly sample the whole space, and indeed this is how many experiments in the spatial literature are set-up. Interestingly, in the case where the space can be uniformly sampled, then it is efficient to partition it using a sparse code [286], like the one represented by place cells. One aspect of this local code is that it minimises the interference among different instances, but it also has two main drawbacks. First, the dimensionality of the representations needed to map the whole space grows quadratically with the number of items to represent; second, and possibility more critical, it does not carry any information about how instances are related to each other. Interestingly, this is exactly the role that many believe grid cells are optimally
tuned for [e.g. 287, 198]. So for the cases where the space onto which the task is based can be uniformly sampled, then it seems that evolution found a set of factorised representation between the hippocampus and the enthorinal cortex that indeed can be mapped onto a two-dimensional manifold. This line of reasoning would also explain two recent results where place and grid cells have been related to more conceptual tasks (also cf. [288]). The first being a brain imaging study on conceptual learning where subjects learn arbitrary associations along trajectories in abstract space represented by a combination of neck and legs of a bird [246]. In the second study, rats were required to use a joystick to manipulate sound along a continuous frequency axis [289]. In both cases the analysis of neural representation revealed that neurons involved in the representation of the task overlapped with the cell types normally involved in spatial reasoning, like place and grid cells. However a closer look at the experimental methodologies reveal that in the two cases, subject were trained extensively to sample the whole continuous space of neck-legs combination and sound frequencies respectively. Therefore, these results could just be explained by the fact that the designs imposed by the experimenter to reduce the dimensionality of the problem to a few factors of variations made the tasks close enough to spatial reasoning that the same computations are employed.

However if we consider language as our domain of training then the picture is quite different. Language is based on a finite set of of words, from which it is possible to generate an infinite number of meaningful sentences [290]. In other words language is combinatorial, which makes it impossible to uniformly sample the space of all configurations. In this case using a code with capacity $O(n^2)$, like the place cells one, to represent each word would be pointless, as a typical combinatorial space has $O(exp(exp(n)))$ configurations that cannot be identified by that capacity [291]. Instead, one way to represent this space is through distributed embeddings, where each word is represented across a variety of units, and each unit participates in encoding many words. Indeed, this is the coding scheme developed by the LSTM network when trained in the language domain. This code not only has higher capacity, but it also approximates continuity in high dimensional vector space, which allow relationships to be inferred across words, thus supporting interpolation-based generalisation [48]. Critically, this vector space representations does not lie on a two-dimensional manifold required as postulated by several authors [244, 243, 284], but it is higher dimensional, thus showing that an alternative computation from the one supported by place and grid cells
4.2. *What can artificial neural networks teach us?*

We think that from the work we presented we can draw some general conclusions. First, we should finally embrace the idea that the mapping learnt by complex models like biological and artificial neural networks is the result of complicated interactions between the objective functions, the architectures, the learning rules, and the dataset [295]. So, if we hope to understand how such systems work then we need to keep all these aspects into account. For instance, just focusing on the embeddings (or representations) developed by a particular
4.2. What can artificial neural networks teach us?

layer of the network could be misleading, as the structure of these embeddings is a direct consequence of all these components. Consequently we should be very careful in interpreting the structure of these embeddings as a general property of the network, rather we should think of these as a property of the training.

The second point follows directly from the first one. We believe artificial neural networks can really represent a shift from the traditional way of modelling cognitive functions. Most of the work done today is performed using simple interpretable models based on data collected in very controlled experiments. Artificial neural networks instead are complex models that can work in real world domains, sometimes by approximating human level capability. This means that by employing these models we might finally be able to shift away from the now so common reductionist approach in neuroscience [296, 18], and work with real world tasks. However many think that this modelling capability comes with a cost - lack of interpretability. Indeed, one common criticism of artificial neural networks is that they are a black-box [297], that is the input-output mapping they learn is not straightforward to interpret, and so even if they can perform the same tasks as well as us, then we will not be able to understand how they do it. One the contrary, we believe that artificial neural networks are transparent models as their architectures, their objective functions, their learning rules, the data used to train them, and all their weights are easily accessible. The reason why these models are called a black-box is because, as scientists, we are tightly attached to the idea that any model should develop representations that are human interpretable to be useful, otherwise it does not have any explanatory power. We strongly disagree with this view, and we believe that as suggested from the school of ecological psychology [298], and recently beautifully restated [294], we should put less emphasis on representations and more on the ability of neural networks to model our behaviour in naturalistic environments. We believe we are just scratching the surface of what these models can do, as we are still in a low data and computational regimes, but these two aspects are growing and so does the ability of artificial neural network to match biological one.
Appendix A

Appendix for Chapter 2
Figure A.1: Linear layer spatial activity maps from the supervised learning experiment. Spatial activity plots for all 512 units in the linear layer $\vec{g}_t$. Units exhibit spatial activity patterns resembling grid cells, border cells, and place cells — head direction tuning was also present but is not shown.
Figure A.2: Grid-like units did not emerge in the linear layer when dropout was not applied. Linear layer spatial activity maps (n=512) generated from a supervised network trained without dropout. The maps do not exhibit the regular periodic structure diagnostic of grid cells.
Figure A.3: Robustness of grid cell agent and performance of other agents.

a-c) AUC performance gives the robustness to hyperparameters (i.e. learning rate, baseline cost, entropy cost - see Table 2 in Supplementary Methods for details of the range) and seeds (see Methods). For each environment we run 60 agent replicas (see Methods). Light purple is the grid agent, blue is the place cell agent and dark purple is A3C. a) Square arena b) Goal-driven c) Goal Doors. In all cases the grid cell agent shows higher robustness to variations in hyper-parameters and seeds.

d-i Performance of place prediction/NavMemNet/DNC
agents (see Methods) against grid cell agent. Dark blue is the grid cell agent (Extended Data Figure 5), green is the place cell prediction agent (Extended Data Figure 9a), purple is the DNC agent, light blue is the NavMemNet agent (Extended Data Figure 9b). The gray band displays the 68% confidence interval based on 5000 bootstrapped samples. d-f) Performance in goal-driven. g-i) Performance in goal-doors. Note that the performance of the place cell agent (Extended Data Figure 8b, lower panel) is shown in Figure 3.
<table>
<thead>
<tr>
<th>Parameter name</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T$</td>
<td>15</td>
<td>Duration of simulated trajectories (seconds)</td>
</tr>
<tr>
<td>$L$</td>
<td>2.2</td>
<td>Width and height of environment, or diameter for circular environment (meters)</td>
</tr>
<tr>
<td>$d$</td>
<td>0.03</td>
<td>Perimeter region distance to walls (meters)</td>
</tr>
<tr>
<td>$\sigma_v$</td>
<td>0.13</td>
<td>Forward velocity Rayleigh distribution scale (m/sec)</td>
</tr>
<tr>
<td>$\mu_{v}$</td>
<td>0</td>
<td>Rotation velocity Gaussian distribution mean (deg/sec)</td>
</tr>
<tr>
<td>$\sigma_{\phi}$</td>
<td>330</td>
<td>Rotation velocity Gaussian distribution standard deviation (deg/sec)</td>
</tr>
<tr>
<td>$\rho_{\theta}$</td>
<td>0.25</td>
<td>Velocity reduction factor when located in the perimeter</td>
</tr>
<tr>
<td>$\Delta\theta$</td>
<td>90</td>
<td>Change in angle when located in the perimeter (deg)</td>
</tr>
<tr>
<td>$\Delta t$</td>
<td>0.02</td>
<td>Simulation-step time increment (seconds)</td>
</tr>
<tr>
<td>$N$</td>
<td>256</td>
<td>Number of place cells</td>
</tr>
<tr>
<td>$\sigma_c$</td>
<td>0.01</td>
<td>Place cell standard deviation parameter (meters)</td>
</tr>
<tr>
<td>$M$</td>
<td>12</td>
<td>Number of target head direction cells</td>
</tr>
<tr>
<td>$\kappa_{\phi}$</td>
<td>20</td>
<td>Head direction concentration parameter</td>
</tr>
<tr>
<td>$\sigma_v$</td>
<td>10^{-5}</td>
<td>Gradient clipping threshold</td>
</tr>
<tr>
<td>minibatch size</td>
<td>10</td>
<td>Number of trajectories used in the calculation of a stochastic gradient</td>
</tr>
<tr>
<td>trajectory length</td>
<td>100</td>
<td>Number of time steps in the trajectories used for the supervised learning task</td>
</tr>
<tr>
<td>learning rate</td>
<td>10^{-5}</td>
<td>Step size multiplier in the RMSProp algorithm</td>
</tr>
<tr>
<td>momentum</td>
<td>0.9</td>
<td>Momentum parameter of the RMSProp algorithm</td>
</tr>
<tr>
<td>L2 regularisation</td>
<td>10^{-5}</td>
<td>Regularisation parameter for linear layer</td>
</tr>
<tr>
<td>parameter updates</td>
<td>300000</td>
<td>Total number of gradient descent steps taken</td>
</tr>
</tbody>
</table>

**Table A.1:** Supervised learning hyperparameters.

<table>
<thead>
<tr>
<th>Parameter name</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Learning rate</td>
<td>[0.000001, 0.0002]</td>
<td>Step size multiplier in the shared RMSProp algorithm of the actor-critic learner with a break</td>
</tr>
<tr>
<td>Gradient momentum</td>
<td>0.99</td>
<td>Momentum parameter of the shared RMSProp algorithm</td>
</tr>
<tr>
<td>Baseline cost $[\alpha]$</td>
<td>[0.48, 0.52]</td>
<td>Cost applied on the gradient of $v$</td>
</tr>
<tr>
<td>Entropy regularisation $[\beta]$</td>
<td>[0.00006, 0.0001]</td>
<td>Entropy regularization term with respect to the policy parameters</td>
</tr>
<tr>
<td>Discount</td>
<td>0.99</td>
<td>Discount factor gamma used in the value function estimation</td>
</tr>
<tr>
<td>Back-propagation step in the actor-critic learner</td>
<td>100</td>
<td>Number of backpropagation step used to unroll the LSTM</td>
</tr>
<tr>
<td>Action repeat</td>
<td>4</td>
<td>Repeat each action selected by the agent this many times</td>
</tr>
<tr>
<td>Learning rate grid network</td>
<td>0.001</td>
<td>Step size multiplier in the RMSProp algorithm of the supervised learner</td>
</tr>
<tr>
<td>$\sigma_v$</td>
<td>30</td>
<td>Place cell scale</td>
</tr>
<tr>
<td>$M$</td>
<td>12</td>
<td>Number of target head direction cells</td>
</tr>
<tr>
<td>$\kappa_{\phi}$</td>
<td>20</td>
<td>Head direction concentration parameter</td>
</tr>
<tr>
<td>Back-propagation step in the supervised learner</td>
<td>100</td>
<td>Number of time steps in the trajectories used for the supervised learning task</td>
</tr>
<tr>
<td>L2 regularization</td>
<td>0.0001</td>
<td>Regularization parameter for linear layers in bottleneck</td>
</tr>
<tr>
<td>Gradient momentum</td>
<td>0.9</td>
<td>Momentum parameter of the RMSProp algorithm in the supervised learner</td>
</tr>
</tbody>
</table>

**Table A.2:** Hyperparameters of all the agents presented.

Values in square bracket are sampled from a categorial distribution in that range
Appendix B
B.1 Paired Associative inference

B.1.1 Attention weights analysis

**Figure B.1:** Attention weights analysis of length 3 PAI task
Here the network converged to 7 hops. In this case the network uses the first two hops to retrieve the slot where the cue is present and the hops number 3, 4 and 5 to retrieve the slot with the match. The weights are sharp and they focus only on 1 single slot.
B.2. bAbI

B.1.2 Adaptive computation

Figure B.2: Comparison between MEMO + REINFORCE and MEMO + ACT on length 3 PAI task.
MEMO with REINFORCE shows more data efficiency than the one where the adaptive computation is done with ACT.

B.2 bAbI

B.2.1 Task-wise results

B.3 MEMO complexity analysis

In terms of temporal complexity, MEMO has a complexity of $O(n_s \cdot A \cdot N \cdot H \cdot I \cdot S \cdot d)$, where $n_s$ is the number of samples we process with our network, $A$ is the number of answers, $N$ is the upper bound of the number of hops we can take, $H$ is the number of heads used, $I$ is the number of stories, and $S$ is the number of words in each sentence. This is due to the fact that, for each sample, we do the hopping procedure for every answer, taking a number of hops. For each hop we query our memory by interacting with all its slots $I$, for all its size $S \times d$. For all our experiments, all parameters $A, N, H, I, S, d$ are fixed to constants.

Further, it is worth noting that MEMO is linear with respect to the number of sentences of the input, whereas the Universal Transformer has quadratic complexity.

With respect to spatial complexity, MEMO holds information of all the weights constant, apart from all the context information that needs to be used to answer a particular query. Since the context information is the only one that is input dependent, the spatial complexity is in this case $O(I \cdot S \cdot d)$, which is the size of our memory. In all our experiments, such size is fixed.
Table B.1: bAbI Results - average over 5 hyper-parameters with lower loss on the validation set

<table>
<thead>
<tr>
<th>Trial Type</th>
<th>MEMO</th>
<th>MEMO top 5 seeds</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 - Single Supporting Fact</td>
<td>100.00</td>
<td>100.00(0.00)</td>
</tr>
<tr>
<td>2 - Two Supporting Facts</td>
<td>100.00</td>
<td>99.13(1.78)</td>
</tr>
<tr>
<td>3 - Three Supporting Facts</td>
<td>97.05</td>
<td>94.15(6.35)</td>
</tr>
<tr>
<td>4 - Two Arg. Relations</td>
<td>100.00</td>
<td>100.00(0.00)</td>
</tr>
<tr>
<td>5 - Three Arg. Relations</td>
<td>100.00</td>
<td>100.00(0.00)</td>
</tr>
<tr>
<td>6 - Yes/No Questions</td>
<td>100.00</td>
<td>100.00(0.00)</td>
</tr>
<tr>
<td>7 - Counting</td>
<td>100.00</td>
<td>96.69(3.57)</td>
</tr>
<tr>
<td>8 - Lists/Sets</td>
<td>100.00</td>
<td>99.13(1.94)</td>
</tr>
<tr>
<td>9 - Simple Negation</td>
<td>100.00</td>
<td>100.00(0.00)</td>
</tr>
<tr>
<td>10 - Indefinite Knowledge</td>
<td>100.00</td>
<td>99.35(1.44)</td>
</tr>
<tr>
<td>11 - Basic Coreference</td>
<td>100.00</td>
<td>100.00(0.00)</td>
</tr>
<tr>
<td>12 - Conjunction</td>
<td>100.00</td>
<td>100.00(0.00)</td>
</tr>
<tr>
<td>13 - Compound Coref</td>
<td>100.00</td>
<td>100.00(0.00)</td>
</tr>
<tr>
<td>14 - Time Reasoning</td>
<td>100.00</td>
<td>100.00(0.00)</td>
</tr>
<tr>
<td>15 - Basic Deduction</td>
<td>100.00</td>
<td>100.00(0.00)</td>
</tr>
<tr>
<td>16 - Basic Induction</td>
<td>98.75</td>
<td>95.05(5.12)</td>
</tr>
<tr>
<td>17 - Positional Reasoning</td>
<td>100.00</td>
<td>100.00(0.00)</td>
</tr>
<tr>
<td>18 - Size Reasoning</td>
<td>100.00</td>
<td>99.13(1.94)</td>
</tr>
<tr>
<td>19 - Path Finding</td>
<td>100.00</td>
<td>100.00(0.00)</td>
</tr>
<tr>
<td>20 - Agents Motivations</td>
<td>100.00</td>
<td>100.00(0.00)</td>
</tr>
<tr>
<td>Mean error</td>
<td>0.21</td>
<td>0.86(1.11)</td>
</tr>
<tr>
<td>Solved tasks (≥95% accuracy)</td>
<td>20/20</td>
<td>n/a</td>
</tr>
</tbody>
</table>

(Mean and standard deviation of test errors for the best 5 hyperparameters (chosen according the validation loss).)

B.4 MEMO training details and hyper-parameters

The fixed parameters are reported in Table B.2 and the one we sweep over are reported in B.3.
### B.5 Baselines hyper-parameters

#### B.5.1 DNC

<table>
<thead>
<tr>
<th>Parameter name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optimizer algorithm</td>
<td>Adam</td>
</tr>
<tr>
<td>Learning rate</td>
<td>0.0005</td>
</tr>
<tr>
<td>Input embedding size</td>
<td>128</td>
</tr>
<tr>
<td>Input pondering steps</td>
<td>10</td>
</tr>
<tr>
<td>Controller type</td>
<td>LSTM</td>
</tr>
<tr>
<td>Controller hidden size</td>
<td>128</td>
</tr>
<tr>
<td>Memory number of read heads</td>
<td>3</td>
</tr>
<tr>
<td>Memory word size</td>
<td>384</td>
</tr>
<tr>
<td>Memory output size</td>
<td>128</td>
</tr>
</tbody>
</table>

Table B.4: Hyperparameters used on all tasks trained with DNC.
We implement ACT as specified in [260]. Based on our implementation of MEMO, we start by defining the *halting unit* \( h \) as the following:

\[
h_t = \sigma(\pi_t)
\]  

\[\text{B.1}\]
where $\pi_t$ is the binary policy of MEMO. This is slightly different than the original ACT which represents such unit with:

$$h_t = \sigma(W_h s_t + b_h) \quad (B.2)$$

where $W_h$ and $b_h$ are trainable weights and biases respectively, and $s_t$ is the previous observed state. We argue that this slight change increases the fairness of the comparison, for two reasons: firstly, $\pi_t(a|s, \theta)$ depends on $s_t$, but it uses several non-linearities to do so, rather than it being a simple linear projection, so it should enable more powerful representations. Secondly, this makes it much more similar to our model while still being able to evaluate the feasibility of this halting mechanism.

From this point we proceed as in the original work by defining the halting probability:

$$p_t = \begin{cases} 
R & \text{if } t = T \\
 h_t & \text{otherwise} 
\end{cases} \quad (B.3)$$

where

$$T = \min \{ t' : \sum_{t=1}^{t'} h_t \geq 1 - \epsilon \} \quad (B.4)$$

where $\epsilon$, as in [260] is fixed to be 0.01 in all experiments. The reminder $R$ is defined as:

$$R = 1 - \sum_{t=1}^{T-1} h_t \quad (B.5)$$

Finally, the answer provided by MEMO+ACT is defined as:

$$a = \sum_{t=1}^{T} p_t a_t \quad (B.6)$$

where $a_t$ corresponds to the answer that MEMO has provided at hop $t$. 
Bibliography


BIBLIOGRAPHY


