THE UNIVERSITY OF CHICAGO

UNIVERSAL NEURAL MEMORY ARCHITECTURES: MULTIGRID CONNECTIVITY, DOMAIN-AGNOSTIC GEOMETRY, AND LOCAL OPERATORS

A DISSERTATION SUBMITTED TO
THE FACULTY OF THE DIVISION OF THE PHYSICAL SCIENCES
IN CANDIDACY FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

DEPARTMENT OF COMPUTER SCIENCE

BY
TRI QUOC HUYNH

CHICAGO, ILLINOIS
JUNE 2021

to my family

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

First and foremost, I would like to express immense gratitude to my principal advisor, Professor Michael Maire, without whom this dissertation would not be able to materialize. From his dedicated guidance to seemingly endless ideas, Professor Maire's support has been the source of energy that shapes the research agenda and the content of this dissertation. While the technical support is astounding, what makes Professor Maire even more admirable is his kind understanding, and the mental support he always displays towards students. It is this kindness that walks us through the challenges in the quest of advancing human knowledge.

Equally important is my co-advisor, Professor Matthew R. Walter, to whom I am deeply thankful. Professor Walter has been involved in every step of shaping the research agenda, spent countless hours in advising, providing ideas, technical support, and helping with paper writing and conference reviewing process. His robotic expertise and research direction bring interdisciplinary knowledge and help driving the pursued research topics.

Thanks to Professor Risi Kondor, who has been part of both the candidacy and dissertation committees alongside Professor Maire and Professor Walter. Professor Kondor's time, commitment and comments have been invaluable in completing this dissertation.

I am also thankful to my former advisor, Professor Gordon Kindlmann. It is Professor Kindlmann who offered me the opportunity to be part of the vibrant and intellectually stimulating community at the University of Chicago. Professor Kindlmann's guidance in the initial phase has provided the foundation for my following development in the graduate program. It is his flexibility and understanding that offer me the opportunity to pursue my new research endeavors.

It has been an exciting time in the Computer Science Department, where we are witnessing a major multi-year growth in all aspects of the department. I feel fortunate to be part of this historic moment. Thanks to the faculty and the staff for always bringing us the best accommodation, resources, and support throughout this monumental period.

Thanks to the fellow graduate students in Professor Maire's group and other friends in the Computer Science Department as well as the Toyota Technological Institute at Chicago for making

the graduate study a memorable and rewarding journey.

Thanks to the Vietnamese student group for making it feel like home, even though we are thousands of miles away from family.

Last but not least, I reserve my deepest gratitude to my family, who have been the source of motivation for all my endeavors. Your love and sacrifices are the most beautiful of all.

ABSTRACT

While memory is fundamental in enabling intelligence, the development of neural memory architectures has largely fallen behind compared to the recent flourish of time-independent neural models. In this thesis, we contribute to the advancement of this field by proposing a novel neural memory model, Multigrid Neural Memory, in which we study in detail three orthogonal dimensions in building the system: architectural design, domain-agnostic processing, and local operators.

First, we introduce a radical new approach to endowing neural networks with access to long-term and large-scale memory. Architecting networks with internal multigrid structure and connectivity, while distributing memory cells alongside computation throughout this topology, we observe that coherent memory subsystems emerge as a result of training. Our design both drastically differs from and is far simpler than prior efforts, such as the recently proposed Differentiable Neural Computer (DNC) [Graves et al., 2016], which uses intricately crafted controllers to connect neural networks to external memory banks. Our hierarchical spatial organization, parameterized convolutionally, permits efficient instantiation of large-capacity memories. Our multigrid topology provides short internal routing pathways, allowing convolutional networks to efficiently approximate the behavior of fully connected networks. Such networks have an implicit capacity for internal attention; augmented with memory, they learn to read and write specific memory locations in a dynamic data-dependent manner. We demonstrate these capabilities on synthetic exploration and mapping tasks, where our network is able to self-organize and retain long-term memory for trajectories of thousands of time steps, outperforming the DNC. On tasks without any notion of spatial geometry: sorting, associative recall, and question answering, our design functions as a truly generic memory and yields excellent results.

Second, we introduce a novel processing scheme that helps enabling domain-agnostic neural architectures. The domain-agnostic property represents the ability to handle data regardless of its nature, either in 1D, 2D, or higher dimensional forms. This property is enforced through the transformations between different spaces enabled by Hilbert curve and positional encoding, which preserve data locality in the original space. Data is then simultaneously and complementarily

processed in multiple sub-spaces. The experiments in tasks involving 1D, 2D, or a combination of both data domains show the effectiveness and genericity of the proposed method.

Third, we further investigate the effect of various new local operators that do not exist in the original multigrid neural memory architecture. Particularly, we study different variants of comparison operators: self-comparison, cross-scale comparison, split-channel comparison, and spatial comparison. Among those operators, experimental results show that split-channel comparison exhibits consistent improvements, especially in settings where there are multiple sources of information.

CHAPTER 1

INTRODUCTION

Memory is one of the two fundamental components defining and enabling intelligence, alongside processing capability. All of our cognitive abilities rely on memory in some form, either long-term or short-term. There has been research [Conway et al., 2003] showing links between working memory capacity and the general intelligence. With its utmost role in human intelligence, memory is a key enabler of artificial general intelligence (AGI).

Recent years have seen tremendous progress in pattern recognition by means of novel advances in deep neural networks. However, most of them either process independent information irrespective of time, perform markov decision process (MDP) with fully observable states, or sequential problems with relatively limited time range or memory storage. While recognition capabilities enjoy dramatic breakthroughs, the development of learnable memory subsystems falls far behind. Even though we enjoy a long history of efforts in endowing AI agents with memorization capacity to reason over time-dependent problems, none has been successful to demonstrate its capability to enable cognitive power on data of size and time scales commensurate with human brain or traditional computers.

Besides, in the contemporary landscape of neural network architectures, most existing models are particularly designed for a certain task or domain. Certain systems are specifically designed for and excel at handling a unique type of data, such as an image sequence (e.g., Convolutions [LeCun et al., 2010] and convolutional LSTMs [Xingjian et al., 2015]), while others are particular effective for other data types (e.g., fully-connected networks [Dauphin et al., 2017, Borovykh et al., 2019] and LSTMs [Hochreiter and Schmidhuber, 1997]). This problem-specific nature of current systems also causes problematic architectural designs in situations where multiple data sources of different types are present, e.g., visual question answering (VQA), in which both 1D and 2D data must be jointly processed. In these situations, for memory processing, most current approaches compromise by using one memory model to process both data types, at the expense of sub-optimal performance for the data type not best suited to this architectural choice. On the other hand, some data sources may not have a clear boundary to be categorized into one specific type, e.g., 1D or 2D. For instance,

although DNA sequences can be viewed in linearized representations, some of their properties depend on their folding structures in space. Anjum et al. [2019] finds that representing DNA sequences in a 2D form brings notably higher accuracy in predicting enhancers.

Furthermore, while convolutional and fully connected networks have been the norm in neural network architectures, the fundamental operators in these networks are only based upon the multiplication between inputs and the learned weights. There are other local operators that are untapped by these traditional models, especially the multiplication between input tensors themselves. Recent architectures [Vaswani et al., 2017, Bello et al., 2019] in fact suggest that these new local operators could bring new processing capability as well as boosting networks' performance. However, the new operators utilized in these architectures are usually confounded with other architectural choices, making it hard to judge their effect in isolation. Also, they are specifically tied to the corresponding architectures, their effect in other settings are not fully investigated.

The aforementioned problems and observations give rise to the three major sets of investigations that we address in this thesis: the architectural design, domain-agnostic processing, and local operators of a novel neural memory model, Multigrid Neural Memory. In particular, the contributions are as follows:

- **Multigrid Neural Memory Architecture:** We introduce a radical new neural memory architecture, termed *multigrid neural memory*, to endow neural networks with long-term, large-scale memory. The proposed architecture departs from all previous attempts in this space. Instead of relying on an external memory storage that is separated from the main computing units, we distribute memory alongside computation throughout the entire network architecture. Parameterized convolutionally, the network allows for efficient instantiation of large scale memory, while the co-location of memory and computation allows for the emergence of novel capabilities. Distinct from all other strategies which craft a controller based on hand-designed attention mechanisms to read from and write to the external memory, we organize memory inside a multigrid hierarchy. This hierarchical topology enables exponentially fast pathways to connect neurons in different network layers, and can effectively approximate the behavior

of fully connected networks. This property translates to the implicit attentional capability of the architecture. A diverse set of experiments are carried out to discover and validate the superior performance of the proposed design. The network is able to self-organize and retain full memory acquired over trajectories of thousands of time-steps. On tasks decoupled from any notion of spatial geometry, such as algorithmic tasks or question answering, the network functions as a truly generic architecture and yields superior performance.

- **Domain-agnostic Processing:** We propose a novel domain-independent processing scheme. The proposed technique should be able to effectively handle data of arbitrary nature, regardless of their underlying structures being in 1D, 2D, or higher dimensional spaces. This universal processing capability is realized through clever transformations among different spaces enabled by Hilbert curve, in addition to positional encodings that help to effectively preserve the locality structure of data in its original space. Data is seamlessly processed and unified in multiple spaces internally, while guaranteeing that the processing in different spatial structures is complementary to each other. We investigate the effect of the proposed method on tasks involving 1D, 2D, and a combination of data in both domains. The experimental results indicate the effectiveness of the method, in which it either achieves comparable or superior performance in all cases compared to specific architectures suitable for the corresponding underlying data domains.

- **Local Operators:** We investigate the effect of comparison operators on the proposed multi-grid neural memory model. Specifically, we investigate four different variants of comparison operators, including self-comparison, cross-scale comparison, spatial comparison, and split-channel comparison. The diverse experiments support the following findings: Not all comparison operators are helpful; Among these operators, split-channel comparison shows consistent improvement in the investigated tasks, and is particularly helpful in cases where there is an interaction of multiple sources of information.

CHAPTER 2

MULTIGRID NEURAL MEMORY

2.1 Introduction

Memory, in the form of generic, high-capacity, long-term storage, is likely to play a critical role in expanding neural networks to new application domains. A neural memory subsystem with such properties could be a transformative technology—pushing neural networks within grasp of tasks traditionally associated with general intelligence and an extended sequence of reasoning steps. Development of architectures for integrating memory units with neural networks spans a good portion of the history of neural networks themselves (e.g., from LSTMs [Hochreiter and Schmidhuber, 1997] to the recent Neural Turing Machines (NTMs) [Graves et al., 2014]). Yet, while useful, none has elevated neural networks to be capable of learning from and processing data on size and time scales commensurate with traditional computing systems. Recent successes of deep neural networks, though dramatic, are focused on tasks, such as visual perception or natural language translation, with relatively short latency— e.g., hundreds of steps, often the depth of the network itself.

We present a network architecture that allows memory subsystems to emerge as a byproduct of training simple components. Though our networks appear structurally uniform, they learn to behave like coherent large-scale memories, internally coordinating a strategy for directing reads and writes to specific layers and spatial locations therein. As an analogy, a convolutional neural network (CNN), tasked with classifying images, may coordinate and specialize its internal layers for extracting useful visual representations. Our networks, trained for a task requiring long-term memory, do the same with respect to memory: they self-organize their internal layers into a large-scale memory store. We accomplish this using only generic components; our networks are comprised of LSTM cells and convolutional operations. Yet, our networks learn to master tasks that are beyond the abilities of traditional CNNs or LSTMs.

Multigrid organization, imposed on both spatial layout and connectivity, is the design principle

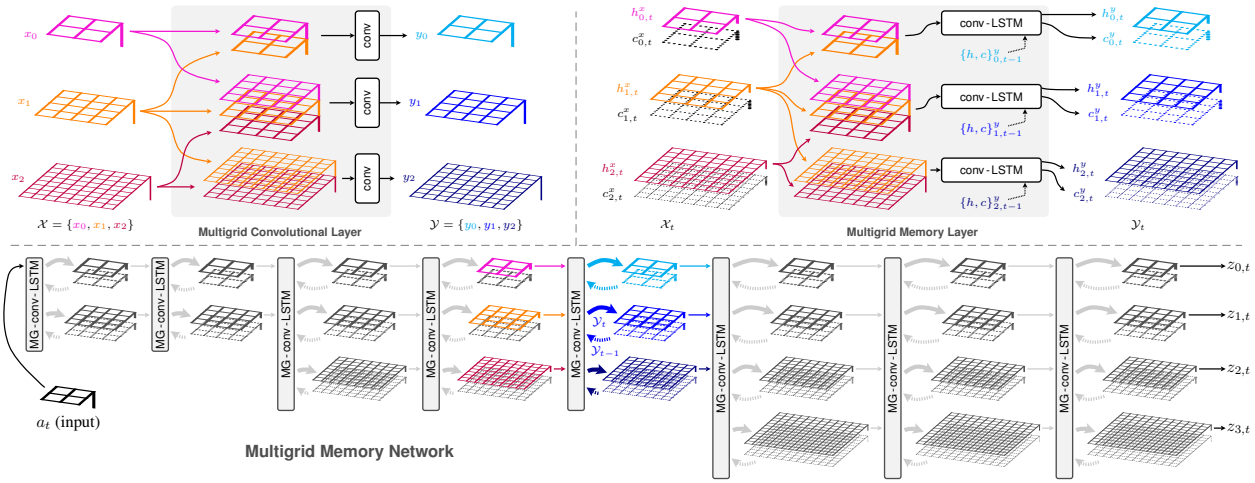


Figure 2.1: **Multigrid memory architecture.** ***Top Left:*** A multigrid convolutional layer [Ke et al., 2017] transforms input pyramid $\mathcal{X}$, containing activation tensors $\{x_0, x_1, x_2\}$, into output pyramid $\mathcal{Y}$ via learned filter sets that act across the concatenated representations of neighboring spatial scales. ***Top Right:*** We design an analogous variant of the convolutional LSTM [Xingjian et al., 2015], in which $\mathcal{X}$ and $\mathcal{Y}$ are indexed by time and encapsulate LSTM internals, *i.e.*, memory cells (c) and hidden states (h). ***Bottom:*** Connecting many such layers, both in sequence and across time, yields a multigrid mesh capable of routing input a_t into a much larger memory space, updating a distributed memory representation, and providing multiple read-out pathways (*i.e.*, $z_{0,t}$, $z_{1,t}$, $z_{2,t}$, $z_{3,t}$, or any combination thereof).

that endows networks with this qualitatively new capacity for forming self-organized memory subsystems. Compared to almost all existing networks, a multigrid wiring pattern provides an exponentially more efficient routing topology among local components embedded within it. Ke et al. [2017] implement a multigrid variant of CNNs, demonstrating that efficient routing capacity enables the network to learn tasks that require attentional behavior. We distribute memory cells throughout such a network, and observe that this implicit capacity for attention translates into an implicit capacity for attention over memory read and write locations. Learned parameters govern how information flows through the network, what memory cells to update, and how to update them.

Our design philosophy starkly contrasts with recent neural memory architectures, including NTMs and the subsequent Differentiable Neural Computer (DNC) [Graves et al., 2016]. These prior approaches isolate memory in an external storage bank, accessed via explicit addressing modes driven by custom hand-crafted controllers; they graft a von Neumann memory model onto a neural

network. Instead, we intertwine memory units throughout the interior of a deep network. Memory is a first-class citizen, rather than a separate data store accessed via a special controller. We introduce a new kind of network layer—a multigrid memory layer—and use it as a stackable building block to create deep memory networks. Contrasting with simpler LSTMs, our memory is truly deep; accessing an arbitrary memory location requires passing through several layers. Figure 2.1 provides a visualization; we defer the full details to Section 2.3.

There are major benefits to our design strategy, in particular:

- ***Distributed, co-located memory and compute.*** Our memory layers incorporate convolutional and LSTM components. Stacking such layers, we create not only a memory network, but also a generalization of both CNNs and LSTMs. Our memory networks are standard networks with additional capabilities. Section 2.4 shows they can learn tasks that require performing classification alongside storage and recall.

  This unification also opens a design space for connecting our memory networks to each other, as well as standard networks. Within a larger system, we could easily plug the internal state of our memory into a standard CNN—essentially granting that CNN read-only memory access. Sections 2.3 and 2.4 develop and experimentally validate two such memory interface approaches.

- ***Scalability.*** Distributing storage over a multigrid hierarchy allows us to instantiate large amounts of memory while remaining parameter-efficient. The low-level mechanism underlying memory access is convolution, and we inherit the parameter-sharing efficiencies of CNNs. Our filters act across a spatially organized collection of memory cells, rather than the spatial extent of an image. Increasing feature channels per memory cell costs parameters, but adding more cells incurs no such cost, decoupling memory size from parameter count. Connecting memory layers across spatial pyramid levels allows for growing memory spatial extent exponentially with network depth, while guaranteeing there is a pathway between the network input and every memory unit.

- ***Simplicity: implicit addressing, emergent subsystems.*** Our networks, once trained, behave like memory subsystems—this is an emergent phenomenon. Our design contains no explicit address

calculation unit, no controller, and no attention mask computation. We take well known building blocks (i.e., convolution and LSTMs), wrap them in a multigrid wiring pattern, and achieve capabilities superior to those of the DNC, a far more complex design.

A diverse array of synthetic tasks serves as our experimental testbed. Mapping and localization, an inherently spatial task with relevance to robotics, is one focus. However, we avoid only experimenting with tasks naturally fit to the architecturally-induced biases of our memory networks. We also train them to perform algorithmic tasks, as well as natural language processing (NLP) tasks, previously used in analyzing the capabilities of NTMs and DNCs. Throughout all settings, DNC accuracy serves as a baseline. We observe significant advantages for multigrid memory, including:

- ***Long-term retention.*** On spatial mapping tasks, our network correctly remembers observations of an external environment collected over paths thousands of time steps long. Visualizing internal memory unit activations reveals an interpretable representation and algorithmic strategy our network learns for solving the problem. The DNC, in contrast, fails to master these tasks.
- ***Generality.*** On tasks decoupled from any notion of spatial geometry, such as associative recall or sorting (algorithmic), or question answering (NLP), our memory networks prove equally or more capable than DNCs.

Section 2.4 further elaborates on experimental results. Section 2.5 discusses implications: multigrid connectivity is a groundbreaking design principle, as it allows qualitatively novel behaviors (attention) and subsystems (memory stores) to emerge from training simple components.

2.2 Related Work

An extensive history of work seeks to grant neural networks the ability to read and write memory [Das et al., 1992, 1993, Mozer and Das, 1993, Zeng et al., 1994, Hölldobler et al., 1997]. Das et al. [1992] propose a neural pushdown automaton, which performs differential push and pop operations on external memory. Schmidhuber [1992] uses two feedforward networks: one

produces context-dependent weights for the second, whose weights may change quickly and can be used as a form of memory. Schmidhuber [1993] proposes memory addressing in the form of a "self-referential" recurrent neural network that modifies its own weights.

Recurrent Long Short-Term Memory networks (LSTMs) [Hochreiter and Schmidhuber, 1997] have enabled significant progress on a variety of sequential prediction tasks, including machine translation [Sutskever et al., 2014], speech recognition [Graves et al., 2013], and image captioning [Donahue et al., 2017]. LSTMs are Turing-complete [Siegelmann and Sontag, 1995] and are, in principle, capable of context-dependent storage and retrieval over long time periods [Hermans and Schrauwen, 2013]. However, capacity for long-term read-write is sensitive to the training procedure [Collins et al., 2017] and is limited in practice.

Grid LSTMs [Kalchbrenner et al., 2015] arrange LSTM cells in a 2D or 3D grid, placing recurrent links along all axes of the grid. This sense of grid differs from our usage of multigrid, as the latter refers to links across a multiscale spatial layout. In Kalchbrenner et al. [2015]'s terminology, our multigrid memory networks are not Grid LSTMs, but are a variant of Stacked LSTMs [Graves et al., 2013].

To improve the long-term read-write abilities of recurrent networks, several modifications have been proposed. These include differentiable attention mechanisms [Graves, 2013, Bahdanau et al., 2014, Mnih et al., 2014, Xu et al., 2015] that provide a form of content-based memory addressing, pointer networks [Vinyals et al., 2015] that "point to" rather than blend inputs, and architectures that enforce independence among neurons within each layer [Li et al., 2018].

A number of methods augment the short- and long-term memory internal to recurrent networks with external "working" memory, in order to realize differentiable programming architectures that can learn to model and execute various programs [Graves et al., 2014, 2016, Weston et al., 2015b, Sukhbaatar et al., 2015, Joulin and Mikolov, 2015, Reed and de Freitas, 2015, Grefenstette et al., 2015, Kurach et al., 2015]. Unlike our approach, these methods explicitly decouple memory from computation, mimicking a standard computer architecture. A neural controller (analogous to a CPU) interfaces with specialized external memory (e.g., random-access memory or tapes).

The Neural Turing Machine (NTM) augments neural networks with a hand-designed attention mechanism to read from and write to external memory in a differentiable fashion. This enables the NTM to learn to perform various algorithmic tasks, including copying, sorting, and associative recall. The Differential Neural Computer [Graves et al., 2016] improves upon the NTM with support for dynamic memory allocation and additional memory addressing modes. Without a sparsifying approximation, DNC runtime grows quadratically with memory due to the need to maintain the temporal link matrix. Our architecture has no such overhead, nor does it require maintaining any auxiliary state.

Other methods enhance recurrent layers with differentiable forms of a restricted class of memory structures, including stacks, queues, and dequeues [Grefenstette et al., 2015, Joulin and Mikolov, 2015]. Gemici et al. [2017] augment structured dynamic models for temporal processes with various external memory architectures [Graves et al., 2014, 2016, Santoro et al., 2016].

Similar memory-explicit architectures have been proposed for deep reinforcement learning (RL) tasks. While deep RL has been applied to several challenging domains [Mnih et al., 2015, Hausknecht and Stone, 2015, Levine et al., 2016], most approaches reason over short-term state representations, which limits their ability to deal with partial observability inherent in many tasks. Several methods augment deep RL architectures with external memory to facilitate long-term reasoning. Oh et al. [2016] maintain a fixed number of recent states in memory and then read from the memory using a soft attention operation. Parisotto and Salakhutdinov [2018] propose a specialized write operator, together with a hand-designed 2D memory structure, both specifically crafted for navigation in maze-like environments.

Rather than learn when to write to memory (e.g., as done by NTM and DNC), Pritzel et al. [2017] continuously write the experience of an RL agent to a dictionary-like memory module queried in a key-based fashion (permitting large memories). Building on this framework, Fraccaro et al. [2018] augment a generative temporal model with a specialized form of spatial memory that exploits privileged information, including an explicit representation of the agent's position.

Though we experiment with RL, our memory implementation contrasts with this past work. Our

multigrid memory architecture jointly couples computation with memory read and write operations, and learns how to use a generic memory structure rather than one specialized to a particular task.

2.3 Multigrid Memory Architectures

A common approach to endowing neural networks with long-term memory builds memory addressing upon explicit attention mechanisms. Such attention mechanisms, independent of memory, are hugely influential in natural language processing [Vaswani et al., 2017]. NTMs [Graves et al., 2014] and DNCs [Graves et al., 2016] address memory by explicitly computing a soft attention mask over memory locations. This leads to a design reliant on an external memory controller, which produces and then applies that mask when reading from or writing to a separate memory bank.

We craft a memory network without such strict division into modules. Instead, we propose a structurally uniform architecture that generalizes modern convolutional and recurrent designs by embedding memory cells within the feed-forward computational flow of a deep network. Convolutional neural networks and LSTMs (specifically, the convolutional LSTM variety [Xingjian et al., 2015]) exist as strict subsets of the full connection set comprising our multigrid memory network. We even encapsulate modern residual networks [He et al., 2016]. Though omitted from diagrams (e.g., Figure 2.1) for the sake of clarity, we utilize residual connections linking the inputs of subsequent layers across the depth (not time) dimension of our memory networks.

In our design, memory addressing is implicit rather than explicit. We build upon an implicit capacity for attentional behavior inherent in a specific kind of network architecture. Ke et al. [2017] propose a multigrid variant of both standard CNNs and residual networks (ResNets). While their primary experiments concern image classification, they also present a striking result on a synthetic image-to-image transformation task: multigrid CNNs (and multigrid ResNets) are capable of learning to emulate attentional behavior. Their analysis reveals that the multigrid connection structure is both essential to and sufficient for enabling this phenomenon.

The underlying cause is that bi-directional connections across a scale-space hierarchy (Figure 2.1, left) create exponentially shorter signalling pathways between units at different locations

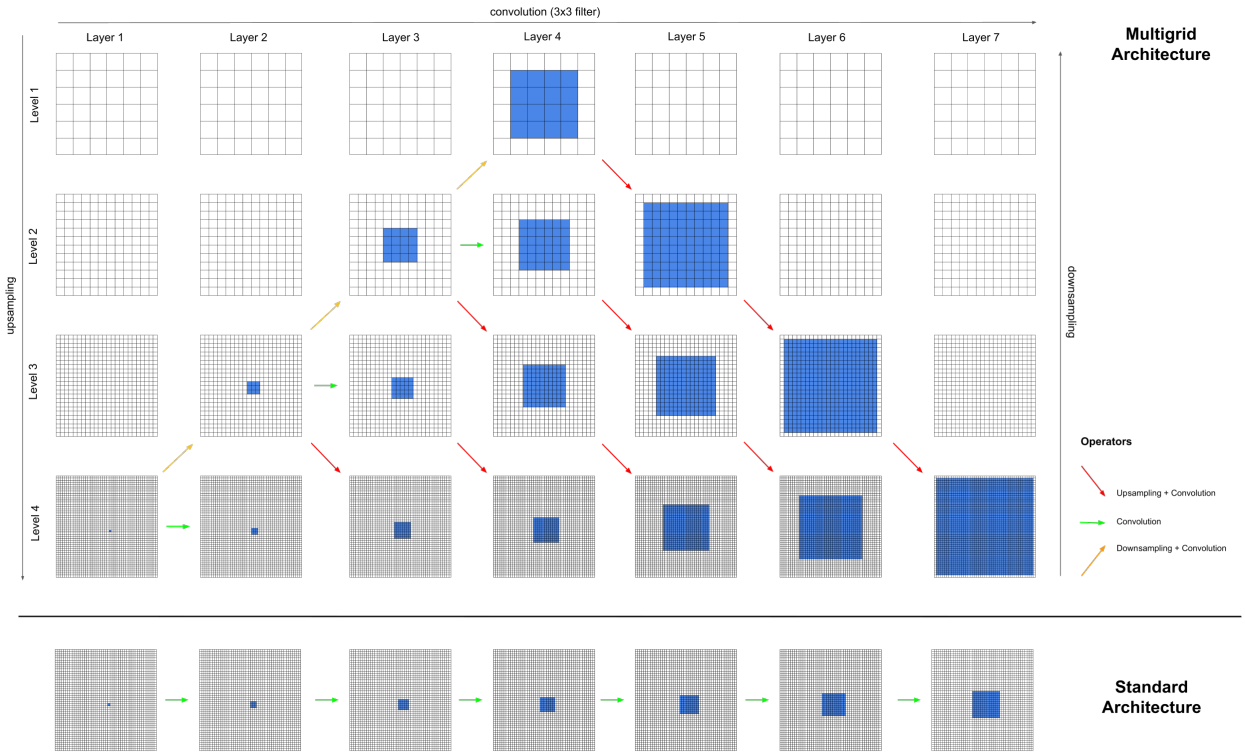


Figure 2.2: **Information routing.** ***Top:*** Paths depicting information flow in a multigrid architecture. Progressing from one layer to the next, information flows between grids at the same level (via convolution, green), as well as to adjacent grids at higher resolution (via upsampling and convolution, red) and lower resolution (via downsampling and convolution, orange). Information from a sample location $(26, 26)$ (blue) of the source grid at [layer 1, level 4] can be propagated to all locations rendered in blue in subsequent layers and levels, following the indicated paths (among others). Information quickly flows from finer levels to coarser levels, and then to any location in just a few layers. Receptive field size grows exponentially with depth. In practice, the routing strategy is emergent—routing is determined by the learned network parameters (convolutional filters). Multigrid connectivity endows the network with the potential to quickly route from any spatial location to any other location just a few layers deeper. ***Bottom:*** Information flow in a standard architecture. Without multigrid connections, information from the same source location is propagated much more slowly across network layers. Receptive fields expand at a constant rate with depth, compared to the multigrid network's exponential growth.

on the spatial grid. Specifically, coarse-to-fine and fine-to-coarse connections between pyramids in subsequent layers allow a signal to hop up pyramid levels and back down again (and vice-versa). As a consequence, pathways connect any neuron in a given layer with every neuron located only $\mathcal{O}(\log(S))$ layers deeper, where S is the spatial extent (diameter) of the highest-resolution grid, as further analyzed in Section 2.3.1. In a standard convolutional network, this takes $\mathcal{O}(S)$ layers. These shorter pathways enable our convolutional architecture to approximate the behavior of a fully-connected network.

By replacing convolutional layers with convolutional LSTMs [Xingjian et al., 2015], we convert the inherent attentional capacity of multigrid CNNs into an inherent capacity for distributed memory addressing. Grid levels no longer correspond to operations on a multiresolution image representation, but instead correspond to accessing smaller or larger storage banks within a distributed memory hierarchy. Dynamic routing across scale space (in the multigrid CNN) now corresponds to dynamic routing into different regions of memory, according to a learned strategy.

2.3.1 Information Routing

Proposition 1: For the setup in Figure 2.2, suppose that the convolutional kernel size is 3×3, and upsampling is $2\times$ nearest-neighbor sampling. Consider location $(1, 1)$ of the source grid at [layer 1, level 1]. For a target grid at [layer m, level n], where $m \geq n$, the information from the source location can be routed to any location (i, j), where $1 \leq i, j \leq (m - n + 2) \cdot 2^{n-1} - 1$.

Proof of Proposition 1: Induction proof on level n.

- For level $n = 1$: Each convolution of size 3×3 can direct information from a location (i, j) at layer k to any of its immediate neighbors (i', j') where $i - 1 \leq i' \leq i + 1$, $j - 1 \leq j' \leq j + 1$ in layer $k + 1$. Therefore, convolutional operations can direct information from location $(1, 1)$ in layer 1 to any locations (i', j') in layer $k = m$ where $1 \leq i'$, $j' \leq m = (m - 1 + 2) \cdot 2^0 - 1 = (m - n + 2) \cdot 2^{n-1} - 1$.

- Assume the proposition is true for level n ($\forall m \geq n$), we show that it is true for level $n + 1$.

Consider any layer $m+1$ in level $n+1$, where $m+1 \geq n+1$:

We have, $m+1 \geq n+1 \Rightarrow m \geq n$. Therefore, we have that at [layer m, level n], the information from the source location can be routed to any location (i, j), where $1 \leq i, j \leq (m-n+2) \cdot 2^{n-1} - 1$. Now, consider the path from [layer m, level n] to [layer $m+1$, level $n+1$]. This path involves the upsampling followed by a convolution operator, as illustrated in Figure 2.2.

Nearest-neighbor upsampling directly transfers information from index i to $2 \cdot i$ and $2 \cdot i - 1$, and j to $2 \cdot j$ and $2 \cdot j - 1$ by definition. For simplicity, first consider index i separately. By transferring to $2 \cdot i$, information from location $1 \leq i \leq (m-n+2) \cdot 2^{n-1} - 1$ in level n will be transferred to all even indices in $[2, ((m-n+2) \cdot 2^{n-1} - 1) \cdot 2]$ at level $n+1$. By transferring to $2 \cdot i - 1$, information from location $1 \leq i \leq (m-n+2) \cdot 2^{n-1} - 1$ in level n will be transferred to all odd indices in $[1, ((m-n+2) \cdot 2^{n-1} - 1) \cdot 2 - 1]$ at level $n+1$. Together, with $2 \cdot i$ and $2 \cdot i - 1$ transferring, the nearest-neighbor upsampling transfers information from location $1 \leq i \leq (m-n+2) \cdot 2^{n-1} - 1$ in level n to all indices in $[1, ((m-n+2) \cdot 2^{n-1} - 1) \cdot 2]$ at level $n+1$.

Furthermore, the following convolution operator with 3×3 kernel size can continue to transfer information from $[1, ((m-n+2) \cdot 2^{n-1} - 1) \cdot 2]$ to $[1, ((m-n+2) \cdot 2^{n-1} - 1) \cdot 2 + 1]$ at level $n+1$. We have $((m-n+2) \cdot 2^{n-1} - 1) \cdot 2 + 1 = (m+1-(n+1)+2) \cdot 2^n - 1$. Taking together indices i and j, information from location (i, j) where $1 \leq i, j \leq (m-n+2) \cdot 2^{n-1} - 1$ in level n can be transferred to (i', j') in level $n+1$, where $1 \leq i', j' \leq (m+1-(n+1)+2) \cdot 2^n - 1$.

■

2.3.2 Multigrid Memory Layer

Figure 2.1 diagrams both the multigrid convolutional layer of Ke et al. [2017] and our corresponding multigrid memory (MG-conv-LSTM) layer. Activations at a particular depth in our network consist of a pyramid $\mathcal{X}_t = \{(h^x_{j,t}, c^x_{j,t})\}$, where j indexes the pyramid level, t indexes time, and x is the layer. h^x and c^x denote the hidden state and memory cell contents of a convolutional LSTM [Xingjian

et al., 2015], respectively. Following the construction of Ke et al. [2017], states h^x at neighboring scales are resized and concatenated, with the resulting tensors fed as inputs to the corresponding scale-specific convolutional LSTM units in the next multigrid layer. The state associated with a conv-LSTM unit at a particular layer and level $(h^y_{j,t}, c^y_{j,t})$ is computed from memory: $h^y_{j,t-1}$ and $c^y_{j,t-1}$, and input tensor: $\uparrow h^x_{j-1,t} \oplus h^x_{j,t} \oplus \downarrow h^x_{j+1,t}$, where $\uparrow$, $\downarrow$, and $\oplus$ denote upsampling, downsampling, and concatenation. Specifically, a multigrid memory layer (Figure 2.1, top right) operates as:

$$
\begin{aligned}
H^x_{j,t} &:= (\uparrow h^x_{j-1,t}) \oplus (h^x_{j,t}) \oplus (\downarrow h^x_{j+1,t}) \\
i_{j,t} &:= \sigma(W^{xi}_j * H^x_{j,t} + W^{hi}_j * h^y_{j,t-1} + W^{ci}_j \circ c^y_{j,t-1} + b^i_j) \\
f_{j,t} &:= \sigma(W^{xf}_j * H^x_{j,t} + W^{hf}_j * h^y_{j,t-1} + W^{cf}_j \circ c^y_{j,t-1} + b^f_j) \\
c^y_{j,t} &:= f_{j,t} \circ c^y_{j,t-1} + i_{j,t} \circ \tanh(W^{xc}_j * H^x_{j,t} + W^{hc}_j * h^y_{j,t-1} + b^c_j) \\
o^y_{j,t} &:= \sigma(W^{xo}_j * H^x_{j,t} + W^{ho}_j * h^y_{j,t-1} + W^{co}_j \circ c^y_{j,t} + b^o_j) \\
h^y_{j,t} &:= o^y_{j,t} \circ \tanh(c^y_{j,t})
\end{aligned}
$$

Superscripts denote variable roles (e.g., layer x or y, and/or a particular parameter subtype for weights or biases). Subscripts index pyramid level j and time t, $*$ denotes convolution, and $\circ$ the Hadamard product. Computation resembles Xingjian et al. [2015], with additional input tensor assembly, and repetition over output pyramid levels j. If a particular input pyramid level is not present in the architecture, it is dropped from the concatenation in the first step. Like Ke et al. [2017], downsampling ($\downarrow$) includes max-pooling. We utilize a two-dimensional memory geometry, and change resolution by a factor of two in each spatial dimension when moving up or down a pyramid level.

Connecting many such memory layers yields a memory network or distributed memory mesh, as shown in the bottom diagram of Figure 2.1. Note that a single time increment (from $t-1$ to t) consists of running an entire forward pass of the network, propagating the input signal a_t to the deepest layer z_t. Though not drawn here, we also incorporate batch normalization layers and

residual connections along grids of corresponding resolution (i.e., from $h^x_{j,t}$ to $h^y_{j,t}$). These details mirror Ke et al. [2017]. The convolutional nature of the multigrid memory architecture, together with its routing capability provides parameter-efficient implicit addressing of a scalable memory space.

2.3.3 Memory Interfaces

As our multigrid memory networks are multigrid CNNs plus internal memory units, we are able to connect them to other neural network modules as freely and flexibly as one can do with CNNs. Figure 2.3 diagrams a few such interface architectures, which we experimentally explore in Section 2.4.

In Figure 2.3 (top), multiple "threads", two readers and one writer, simultaneously access a shared multigrid memory. The memory itself is located within the writer network (blue), which is structured as a deep multigrid convolutional-LSTM. The reader networks (red and orange), are merely multigrid CNNs, containing no internal storage, but observing the hidden state of the multigrid memory network.

Figure 2.3 (bottom) diagrams a deep multigrid analogue of a standard paired recurrent encoder and decoder. This design substantially expands the amount of memory that can be manipulated when learning sequence-to-sequence tasks.

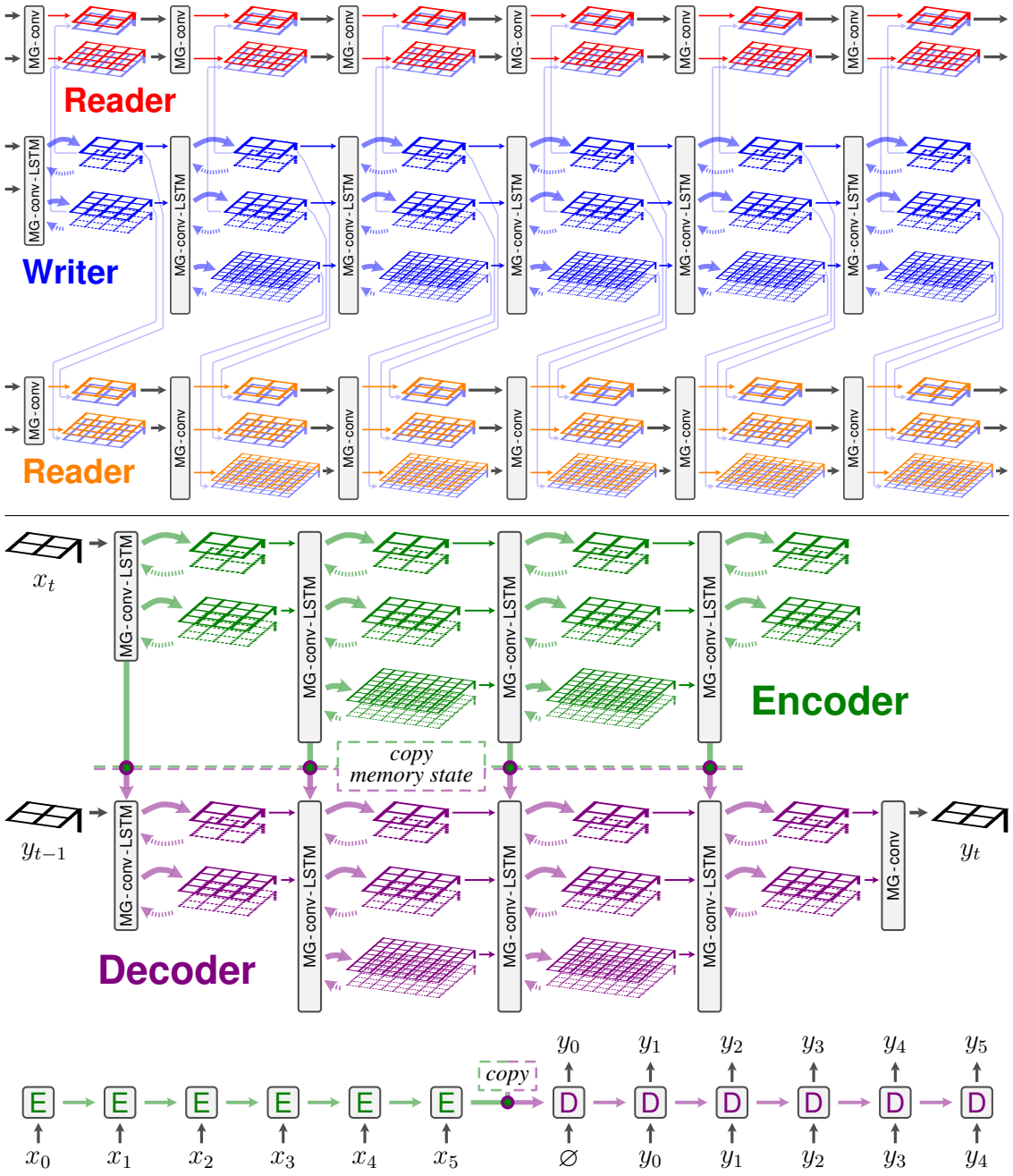


Figure 2.3: **Memory interfaces.** ***Top:*** Multiple readers (red, orange) and a single writer simultaneously manipulate a multigrid memory. Readers are multigrid CNNs; each convolutional layer views the hidden state of the corresponding grid in memory by concatenating it as an additional input. ***Bottom:*** Distinct encoder and decoder networks, each structured as a deep multigrid memory mesh, cooperate to perform a sequence-to-sequence task. We initialize the memory pyramid (LSTM internals) of each decoder layer by copying it from the corresponding encoder layer.

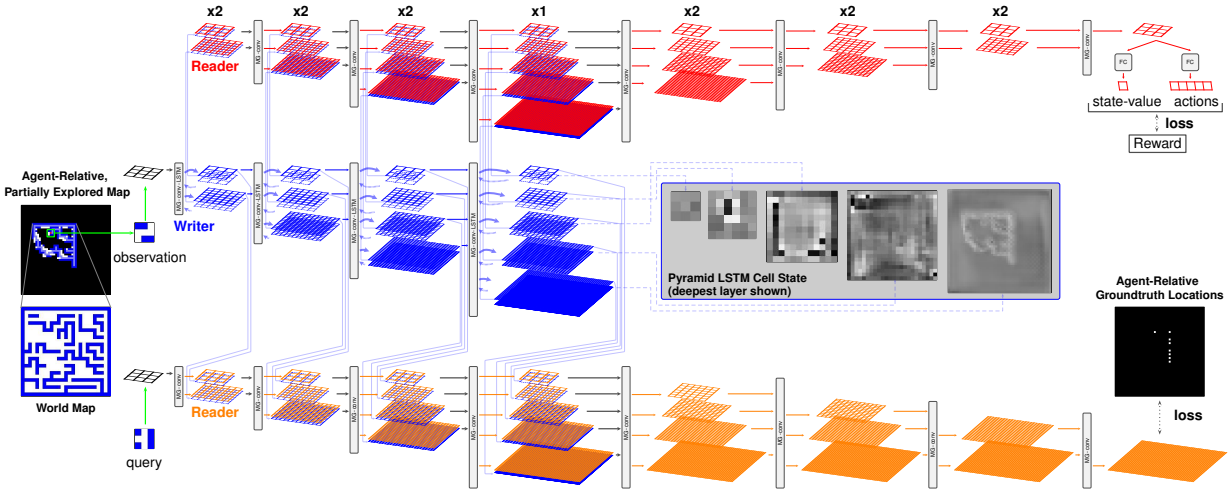


Figure 2.4: **Mapping, localization, and exploration.** An agent is comprised of a deep multigrid memory, and two deep multigrid CNNs (query and policy subnetworks), which have memory read access. Navigating a maze, the agent makes a local observation at each time step, and chooses a next action, receiving reward for exploring unseen areas. Given a random local patch, the query subnet must report all previously observed maze locations whose local observations match that patch. Subnet colors reflect those in Figure 2.3.

2.4 Experiments

We evaluate our multigrid neural memory architecture on a diverse set of domains. We begin with a reinforcement learning-based navigation task, in which memory provides a representation of the environment (i.e., a map). To demonstrate the generalizability of our memory architecture on domains decoupled from spatial geometry, we also consider various algorithmic and NLP tasks previously used to evaluate the performance of NTMs and DNCs.

2.4.1 Mapping & Localization

We first consider a navigation problem, in which an agent explores a priori unknown environments with access to only observations of its immediate surroundings. Effective navigation requires maintaining a consistent representation of the environment (i.e., a map). Using memory as a form of map, an agent must learn where and when to perform write and read operations as it moves, while retaining the map over long time periods. This task mimics partially observable spatial navigation scenarios considered by memory-based deep reinforcement learning (RL) frameworks.

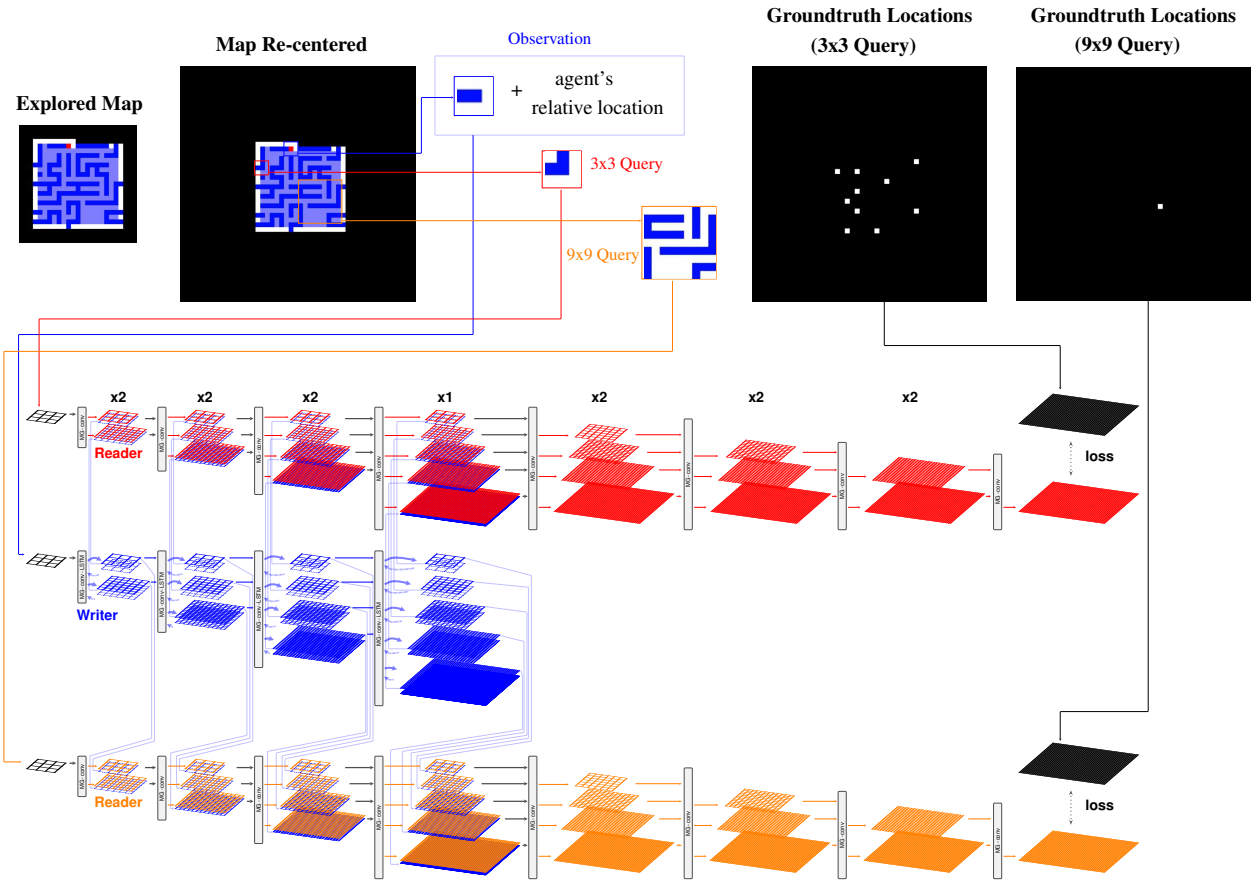


Figure 2.5: **Multigrid memory writer-reader(s) architecture for spatial mapping and localization.** At each time step, the agent moves to a new location and observes the surrounding 3×3 patch. The writer receives this 3×3 observation along with the agent's relative location (with respect to the starting point), updating the memory with this information. Two readers receive randomly chosen 3×3 and 9×9 queries, view the current map memory built by the writer, and infer the possible locations of those queries.

Problem Setup: The agent navigates an unknown $n \times n$ 2D maze and observes only the local $m \times m$ grid ($m \ll n$) centered at the agent's position. It has no knowledge of its absolute position. Actions consist of one-step motion in the four cardinal directions. While navigating, we query the network with a randomly chosen, previously seen, $k \times k$ patch ($m \leq k \ll n$) and ask it to identify every location matching that patch in the explored map. See Figure 2.4.

Multigrid Architecture: We use a deep multigrid network with multigrid memory and multigrid CNN subcomponents (Figure 2.4). Our memory (writer subnet) consists of seven MG-conv-LSTM layers, with pyramid spatial scales progressively increasing from 3×3 to 48×48. The reader,

structured similarly, has an output attached to its deepest 48×48 grid, and is tasked with answering localization queries. Section 2.4.2 experiments with an additional reader network that predicts actions, driving the agent to explore.

In order to understand the network's ability to maintain a "map" of the environment in memory, we first consider a setting in which the agent executes a pre-defined navigation policy, and evaluate its localization performance. We consider different policies (spiraling outwards and a random walk), patch sizes for observation and localization (3×3 or 9×9), as well as different trajectory (path) lengths. Detailed architectures are depicted in Figure 2.5.

At each time step during training, the agent takes a one-step action (e.g., along a spiral trajectory) and observes its 3×3 surroundings. This observation, together with its location relative to the starting point, are fed into the writer, which must learn to update its memory. The agent has no knowledge of its absolute location in the world map. Two random 3×3 and 9×9 patches within the explored map are presented to the agent as queries (some experiments use only 3×3 queries). These queries feed into two readers, each viewing the same memory built by the writer; they must infer which previously seen locations match the query. Since the agent has no knowledge of its absolute location in the world map, the agent builds a map relative to its initial position (*map re-centered* in Figure 2.5) as it navigates.

During training, the writer learns to organize and update memory from localization losses simultaneously backpropagated from the two readers. During inference, only the writer updates the memory at each time step, and the readers simply view (i.e., without modification) the memory to infer the query locations. It is also worth noting that only 3×3 patches are fed into the writer at each time step; the agent never observes a 9×9 patch. However, the agent successfully integrates information from the 3×3 patches into a coherent map memory in order to correctly answer queries much larger than its observations.

We experiment in two regimes: small (8K) memory multigrid and DNC models, calibrated to the maximum trainable DNC size, and larger memory multigrid and convolutional-LSTM variants. We compare against:

- **Differentiable Neural Computer (DNC)** [Graves et al., 2016]: We use the official DNC implementation,[1] with 5 controller heads (4 read heads and 1 write head). For spatial navigation and algorithmic tasks, we use a memory vector of 16 elements, and 500 memory slots (8K total), which is the largest memory size permitted by GPU resource limitations. Controllers are LSTMs, with hidden state sizes chosen to make total parameters comparable to other models in Table 2.1 and Table 2.2. DNC imposes a relatively small cap on the addressable memory due to the quadratic cost of the temporal linkage matrix.[2]
- **Ablated MG**: a multigrid architecture variant including only the finest pyramid scale at each layer.
- **ConvLSTM-deep**: a network made of 23 convolutional-LSTM layers, each on a 48×48 grid, yielding the same total grid count as our 7-layer multigrid network.
- **ConvLSTM-thick**: 7 layers of convolutional-LSTMs acting on 48×48 grids. We set channel counts to the sum of channels distributed across the corresponding pyramid layer of our large multigrid network.

We train each architecture using RMSProp. We search over learning rates in log scale from 10^{-2} to 10^{-4}, and use 10^{-3} for multigrid and ConvLSTM, and 10^{-4} for DNC. We use randomly generated maps for training and testing. Training runs for 8×10^6 steps with batch size 32. Test set size is 5000 maps.

Loss: Given predicted probabilities and the ground-truth location mask (Figure 2.5), we employ a pixel-wise cross-entropy loss as the localization loss. Specifically, letting S be the set of pixels, p_i be the predicted probability at pixel i, and y_i be the binary ground-truth at pixel i, the pixel-wise cross-entropy loss is computed as follows:

$$-\sum_{i \in S} y_i \log(p_i) + (1 - y_i) \log(1 - p_i) \tag{2.1}$$

1. `https://github.com/deepmind/dnc`

2. `https://github.com/deepmind/dnc/blob/master/dnc/addressing.py#L163`

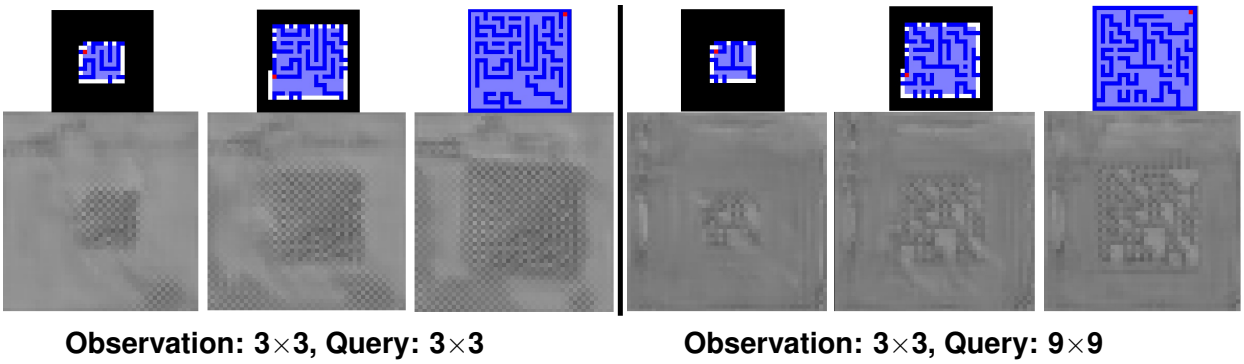


Figure 2.6: **Memory Visualization.** Memory contents (hidden states $\{h_t\}$ on deepest, highest-resolution grid) mirror the map explored with spiral motion (top), vividly showing the interpretable strategy for self-organizing, implicit attentional addressing (reading/writing) of highly specific memory cells when training localization tasks, without having hand-designed attention mechanisms.

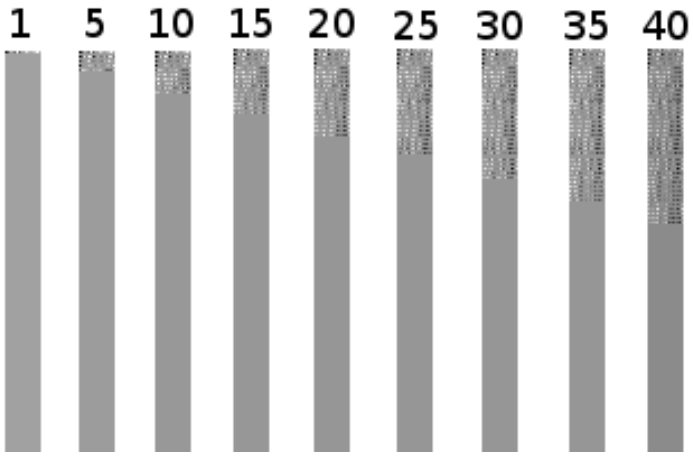


Figure 2.7: **Visualization of DNC memory in mapping task.** Due to its defined addressing mechanism, the DNC always allocates a new continuous memory slot at each time-step. It does not appear to maintain an interpretable structure of the map.

Table 2.1 reports performance in terms of localization accuracy on the test set. For the 25×25 world in which the agent moves in a spiral (i.e., predictable) motion and the observation and query are 3×3, our small multigrid network achieves near perfect precision (99.33%), recall (99.02%), and F-score (99.17%), while all baselines struggle. Both ConvLSTM baselines fail to learn the task; simply stacking convolutional-LSTM units does not work. DNC performs similarly to Ablated MG in terms of precision ($\approx 77.6\%$), at the expense of a significant loss in recall (14.50%). Instead tasking the DNC with the simpler job of localization in a 15×15 world, its performance improves, yet its scores are still around 10% lower than those of multigrid on the more challenging 25×25

Table 2.1: **Mapping and localization.** Our network significantly outperforms the DNC and other baselines. Efficient memory usage, enabled by multigrid connectivity, is essential; the DNC even fails to master smaller 15×15 mazes. Our network retains memory over thousands of time-steps. Our localization subnet, trained on random motion, generalizes to queries for a *policy-driven agent* (last row).

Architecture	Params ($\times 10^6$)	Memory ($\times 10^3$)	World Map	Task Definition FoV	Motion	Query	Path Length	Localization Accuracy Prec.	Recall	F
MG Mem+CNN	0.12	7.99	15×15	3×3	Spiral	3×3	169	**99.79**	**99.88**	**99.83**
DNC	0.75	8.00						91.09	87.67	89.35
MG Mem+CNN	0.17	7.99	25×25				529	**99.33**	**99.02**	**99.17**
DNC	0.68	8.00						77.63	14.50	24.44
MG Mem+CNN	0.65	76.97						**99.99**	**99.97**	**99.98**
Ablated MG	1.40	265.54						77.57	51.27	61.73
ConvLSTM-deep	0.38	626.69						43.42	3.52	6.51
ConvLSTM-thick	1.40	626.69						47.68	1.11	2.16
MG Mem+CNN	0.79	76.97	25×25	3×3	Spiral	9×9	529	97.34	99.50	98.41
	0.65	76.97		3×3	Random	3×3	500	96.83	95.59	96.20
							1500	96.13	91.08	93.54
	0.66	78.12		9×9	Random	9×9	500	92.82	87.60	90.14
	0.65	76.97		3×3	***Policy***	3×3	1000	***95.65***	***90.22***	***92.86***

environment. For 25×25, efficiently addressing a large memory is required; the DNC's explicit attention strategy appears to fall behind our implicit routing mechanism. Trying to compensate by augmenting the DNC with more memory is difficult: without a sparsifying approximation, the DNC's temporal memory linkage matrix incurs a quadratic cost in memory size. Our architecture has no such overhead, nor does it require maintaining auxiliary state. Even limiting multigrid to 8K memory, it has no issue mastering the 25×25 world.

Figure 2.6 visualizes the contents of the deepest and high-resolution LSTM block within the multigrid memory network of an agent moving in a spiral pattern. This memory clearly mirrors the contents of the true map. The network has learned a correct, and incidentally, an interpretable, procedure for addressing and writing memory. For comparison, a visualization of DNC memory in the setting with 15×15 map is provided in Figure 2.7.

In more complex settings for motion type and query size (Table 2.1, bottom) our multigrid network remains accurate. It even generalizes to motions different from those on which it trained, including motion dictated by the learned policy that we describe shortly. Notably, even with the very long trajectory of 1500 time steps, our proposed architecture has no issue retaining a large map

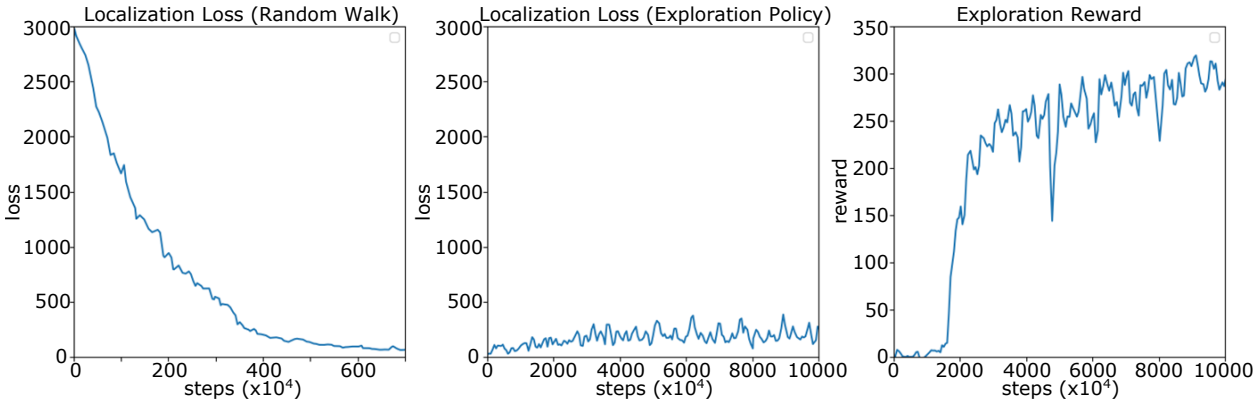


Figure 2.8: **Generalization of localization.** Fixing parameters after training the query subnet on random motion ***(left)***, its localization loss remains low while training the exploration policy ***(middle)***, whose reward improves ***(right)***.

memory.

2.4.2 Joint Exploration, Mapping, and Localization

We now consider a setting in which the agent learns an exploration policy via reinforcement, on top of a fixed mapping and localization network pre-trained with random walk motion. We implement the policy network as another multigrid reader, and leverage the pre-trained mapping and localization capabilities to learn a more effective policy.

We formulate exploration as a reinforcement learning problem: the agent receives a reward of 1 when visiting a new space cell, -1 if it hits a wall, and 0 otherwise. We use a discount factor $\gamma = 0.99$, and train the multigrid policy network using A3C [Mnih et al., 2016].

Figure 2.8 (left) depicts the localization loss while pre-training the mapping and localization subnets. Freezing these subnets, we see that localization remains reliable (Figure 2.8, middle) while reinforcement learning the policy (Figure 2.8, right). The results demonstrate that the learned multigrid memory and query subnets generalize to trajectories that differ from those in their training dataset, as also conveyed in Table 2.1 (last row). Meanwhile, the multigrid policy network is able to utilize memory from the mapping subnet in order to learn an effective exploration policy. See Section 2.4.6 for visualizations of the exploratory behavior.

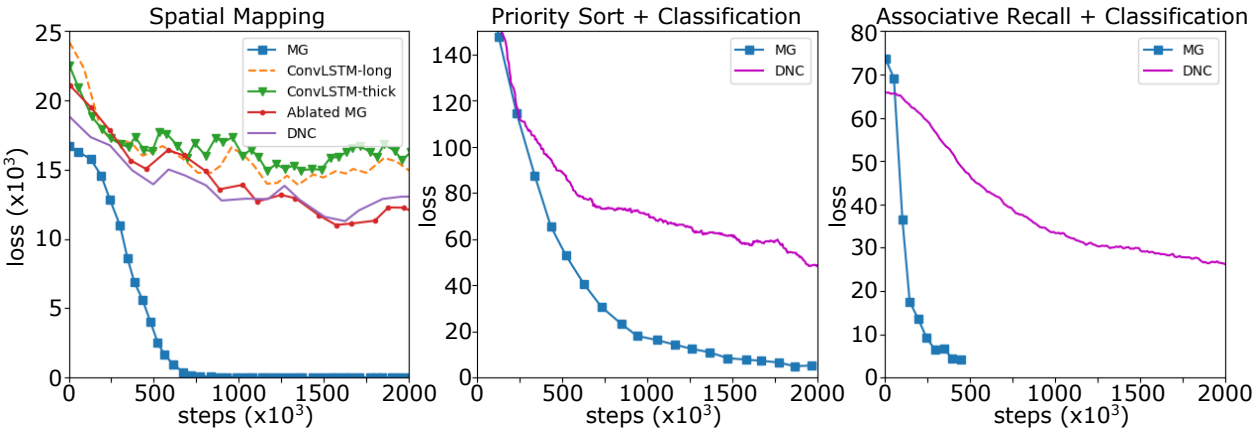


Figure 2.9: **Multigrid memory architectures learn significantly faster.** ***Left***: Maze localization task. ***Middle***: Joint priority sort and classification. ***Right***: Joint associative recall and classification.

Figure 2.10: **MNIST recall.** A random sequence of images followed by a repeat (green), output the *class* of the next image (red).

Table 2.2: **Algorithmic tasks.** Multigrid memory architectures achieve far lower error on the classification variants of priority sort and associative recall, while performing comparably to the DNC on the simpler versions of these tasks. Multigrid memory architectures also remain effective when dealing with long sequences.

	Architecture	Params ($\times 10^6$)	Memory ($\times 10^3$)	Item Size	List Length	Data	Task	Error Rate $\pm\ \sigma$
Standard Sequence	MG Enc+Dec	0.12	7.99	1×9	20	Random Patch	Priority Sort	0.0043±0.0009
	DNC	0.76	8.00					0.0039±0.0016
	MG Enc+Dec	0.29	7.56	28×28	20	MNIST	Priority Sort + Classify	**0.0864±0.0016**
	DNC	1.05	8.00					0.1659±0.0188
	MG Mem+CNN	0.13	7.99	1×9	10	Random Patch	Assoc. Recall	0.0030±0.0002
	DNC	0.76	8.00					0.0044±0.0001
	MG Mem+CNN	0.21	7.56	28×28	10	MNIST	Assoc. Recall + Classify	**0.0736±0.0045**
	DNC	0.90	8.00					0.2016±0.0161
Extended Sequence	MG Enc+Dec	0.89	76.97	1×9	50	Random Patch	Priority Sort	0.0067±0.0001
	MG Mem+CNN	0.65	76.97		20		Assoc. Recall	0.0056±0.0003

2.4.3 Algorithmic Tasks

We test the task-agnostic nature of our multigrid memory architecture by evaluating on a series of algorithmic tasks, closely inspired by those appearing in the original NTM work [Graves et al., 2014]. For each of the following tasks, we consider two variants, increasing in level of difficulty.

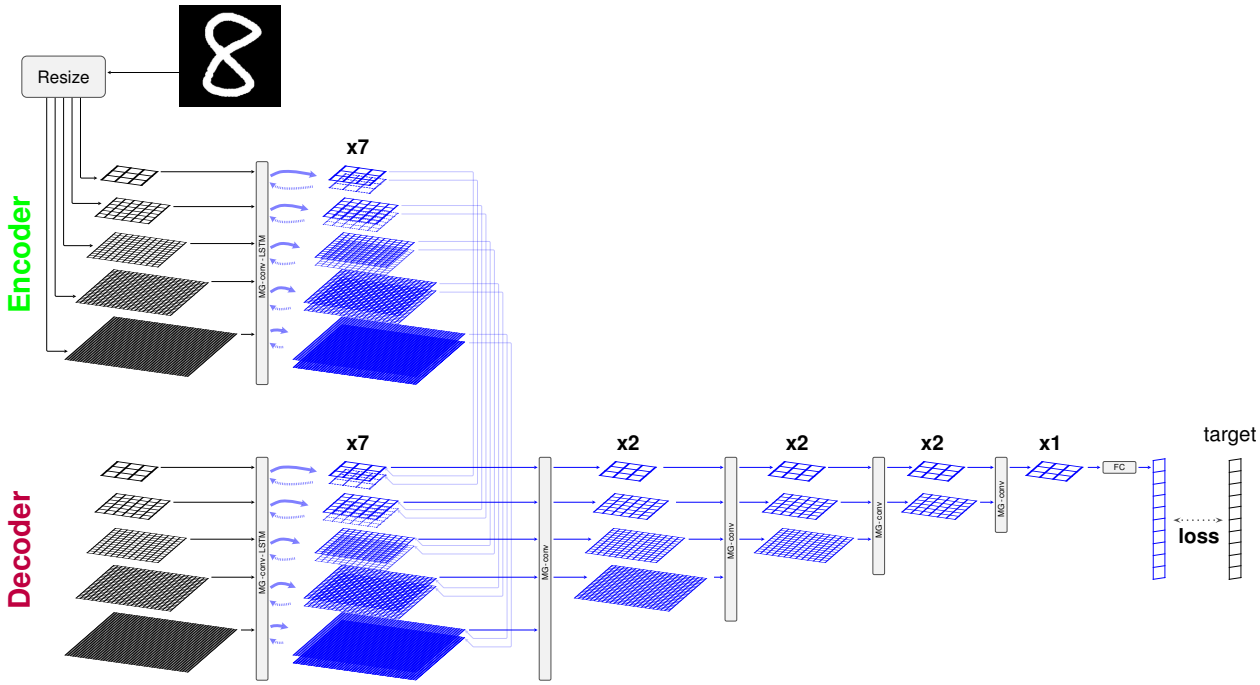


Figure 2.11: **Multigrid memory encoder-decoder architecture for MNIST sorting.** After processing the input sequence, the encoder (top) transfers memory into the decoder, which predicts the sequence of classes of the input digits in sorted order.

Priority Sort

Standard Variant. In the first non-visual variant, the network receives a sequence of twenty 9-dimensional vectors, along with their priority. The task is to output the sequence of vectors in order of their priority. Training and testing use randomly generated data. Training takes 2×10^6 steps, batch size 32, and testing uses 5000 sequences. Results are computed over 10 runs. We tune hyperparameters as done for the mapping task.

Architecture: We structure our model as an encoder-decoder architecture (Figure 2.3, bottom). The encoder has the same architecture as the writer used in the spatial navigation tasks (Figure 2.5). For the decoder, the first half of the layers (MG-conv-LSTM) resemble the encoder, while the second half employ MG-conv layers to progressively scale down the output to 3×3.

Loss: We use pixel-wise cross-entropy loss, as described in Section 2.4.1.

As shown in Table 2.2, our network performs equivalently to DNC with equal memory on this task, with both achieving near-perfect performance.

Classification Variant. The second variant extends the priority sort to require recognition capability. The input is a sequence of twenty 28×28 MNIST images [Lecun et al., 1998]. The goal is to output the *class* of the input images in increasing order.

Architecture: Figure 2.11 depicts the encoder-decoder architecture for this variant.

Loss: We use cross-entropy loss over a softmax prediction of the classes. Specifically, letting C be the set of available classes, p_c the softmax output for class c, and y a one-hot vector of the ground-truth label, we compute the loss as:

$$L = -\sum_{c \in C} y_c \log(p_c) \tag{2.2}$$

Table 2.2 reveals that our architecture achieves much lower error rate compared to DNC on this task (priority sort + classification), while also learning faster (Figure 2.9) and with less memory.

Associative Recall.

Standard Variant. In the first formulation, the network receives a sequence of ten 9-element random vectors, followed by a second instance of one of the first nine vectors. The task is to output the vector that immediately followed the query in the input sequence. Training is similar to the sorting task.

Architecture: We demonstrate this capability using the multigrid reader/writer architecture (Figure 2.3, top). We use the same writer architecture that is shown in Figure 2.5. For the reader, after progressing to the finest resolution (48×48) corresponding to the memory in the writer, the second half of MG-conv layers progressively scale down the output to 3×3 to match the expected output size (instead of 48×48 as in the spatial navigation tasks).

Loss: We use pixel-wise cross-entropy loss, as described in Section 2.4.1.

Table 2.2 shows that both DNC and our architecture achieve near-zero error rates.

Classification Variant. In the second variant, the input is a sequence of ten 28×28 randomly chosen MNIST images [Lecun et al., 1998], where the network needs to output the *class* of the

Table 2.3: **Question answering tasks.** Despite using less memory, our multigrid memory architecture surpasses DNC's performance.

Task	Mean $\pm \sigma$	
	MG Mem	DNC
single supporting fact	$\mathbf{0.0 \pm 0.1}$	9.0 ± 12.6
two supporting facts	$\mathbf{34.2 \pm 0.8}$	39.2 ± 20.5
three supporting facts	56.9 ± 2.3	39.6 ± 16.4
two argument relations	$\mathbf{0.0 \pm 0.1}$	0.4 ± 0.7
three argument relations	5.0 ± 2.1	1.5 ± 1.0
yes/no questions	$\mathbf{1.8 \pm 1.0}$	6.9 ± 7.5
counting	$\mathbf{5.4 \pm 1.1}$	9.8 ± 7.0
lists/sets	$\mathbf{4.7 \pm 1.7}$	5.5 ± 5.9
simple negation	$\mathbf{0.8 \pm 1.0}$	7.7 ± 8.3
indefinite knowledge	$\mathbf{7.4 \pm 2.7}$	9.6 ± 11.4
basic coreference	$\mathbf{0.4 \pm 0.5}$	3.3 ± 5.7
conjunction	$\mathbf{0.1 \pm 0.2}$	5.0 ± 6.3
compound coreference	$\mathbf{0.1 \pm 0.2}$	3.1 ± 3.6
time reasoning	24.3 ± 2.0	11.0 ± 7.5
basic deduction	$\mathbf{14.9 \pm 13.8}$	27.2 ± 20.1
basic induction	$\mathbf{51.9 \pm 1.0}$	53.6 ± 1.9
positional reasoning	$\mathbf{25.1 \pm 8.3}$	32.4 ± 8.0
size reasoning	$\mathbf{3.8 \pm 1.9}$	4.2 ± 1.8
path finding	$\mathbf{10.4 \pm 17.8}$	64.6 ± 37.4
agents motivations	0.3 ± 0.4	0.0 ± 0.1
Mean Error (%)	$\mathbf{12.4 \pm 2.1}$	16.7 ± 7.6
Failed Tasks (Error > 5%)	$\mathbf{8.6 \pm 2.5}$	11.2 ± 5.4

image immediately following the query (Figure 2.10).

Architecture: we resize the 28×28 images to five scales from 3×3 to 48×48 and maintain the same five-scale structure for seven layers of the writer. The writer architecture is the same as the encoder architecture in MNIST priority sort task, as depicted in Figure 2.11. The reader for the MNIST variant is similar to the reader in the standard variant, with the final layer followed by a fully connected layer to produce a 10-way prediction vector over MNIST classes.

Loss: we use cross-entropy loss over a softmax prediction of the classes, as detailed in the classification variant of priority sort task.

As shown in Table 2.2 and Figure 2.9, our multigrid memory network performs this task with significantly greater accuracy than the DNC, and also learns in fewer training steps.

To further test the ability of multigrid memory architectures to deal with longer sequences, we experimented with sorting and associative recall with sequence length of 50 and 20, respectively. As can be seen in Table 2.2, multigrid memory architectures remain effective with near-zero error rates.

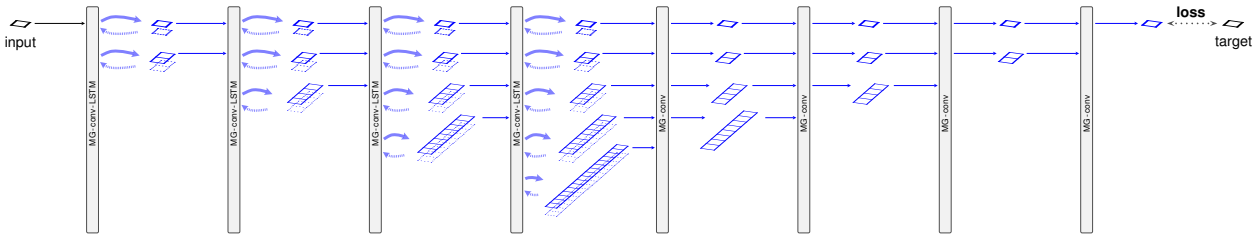


Figure 2.12: **Multigrid memory architecture for question answering.** 1D multigrid architecture is employed for question answering tasks. Input and output are $1 \times 1 \times 159$ tensors representing the word vectors. At each time step, the model receives a word input and generates the next word in the sequence.

The harder variants of both priority sort and associative recall require a combination of memory and a pattern recognition capability. The success of multigrid memory networks (and notable poor performance of DNCs), demonstrates that they are a unique architectural innovation. They are capable of learning to simultaneously perform representational transformations and utilize a large distributed memory store. Furthermore, as Figure 2.9 shows, across all difficult tasks, including mapping and localization, multigrid memory networks train substantially faster and achieve substantially lower loss than all competing methods.

2.4.4 Question Answering

To further investigate the generalizability of our multigrid memory architecture well beyond spatial reasoning, we evaluate its performance on bAbI [Weston et al., 2015a], which consist of 20 question answering tasks corresponding to diverse aspects of natural language understanding. The dataset contains 10000 questions of which 1000 are used for testing and the rest for training.

Architecture: We employ a 1D multigrid memory architecture for question answering tasks, where the spatial dimensions progressively scale from 1×1 to 1×16 through MG-conv-LSTM layers, and gradually scale back to 1×1 through MG-conv layers, as demonstrated in Figure 2.12. Inputs and outputs are $1 \times 1 \times |V|$ tensors representing the word vectors, where V is the set of words in the vocabulary and $|V| = 159$. All 20 question answering tasks are jointly trained, with batch size 1, and sequence-wise normalization. At each time step, the model receives a word input and generates the next word in the sequence. During training, only the losses from words corresponding

to answers are backpropagated, others are masked out, as specified next.

Loss: Let V be the set of words in the vocabulary, and $y \in \{0,1\}^{|V|}$ be a one-hot vector that represents the ground-truth word. For a word sequence S, we define a mask m as:

$$m_i = \begin{cases} 1 & \text{if word } i \text{ in the sequence } S \text{ is an answer} \\ 0 & \text{otherwise} \end{cases} \tag{2.3}$$

Letting $p \in (0,1)^{|V|}$ be the softmax output, we compute the loss for question answering as follows:

$$L = -\sum_{i=1}^{|S|} m_i \sum_{j=1}^{|V|} y_j^i \log(p_j^i) \tag{2.4}$$

DNC Details: DNC memory comprises 256 memory slots, with a 64-element vector for each slot ($16,384$ total). The use of a smaller number of memory slots and batch size allows for the allocation of larger total memory.

Results are shown in Table 2.3. Despite having only a fraction of the memory available to the DNC (2.86K v.s. 16.38K), our architecture outperforms the DNC in terms of both mean error rate (12.4% v.s. 16.7%) and failed tasks (8.6 v.s. 11.2). This result not only demonstrates the adaptability of multigrid memory, but also is a testament to the design's superiority.

2.4.5 Runtime

On spatial mapping (with 15×15 world map), the runtimes for one-step inference with the Multigrid Memory architecture (0.12 M parameters and 8 K memory) and DNC (0.75 M parameters and 8 K memory) are (mean $\pm$ std): 0.018 ± 0.003 seconds and 0.017 ± 0.001 seconds, respectively. These statistics are computed over 10 runs on a NVIDIA Geforce GTX Titan X.

2.4.6 Demos

- Instructions for interpreting the video demos:

  `https://drive.google.com/file/d/18gvQRhNaEbdiV8oNKOsuUXpF75FEHmgG`

- Mapping & localization in spiral trajectory, with 3×3 queries:

  `https://drive.google.com/file/d/1VGPGHqcNXBRdopMx11_wy9XoJS7REXbd`

- Mapping & localization in spiral trajectory, with 3×3 and 9×9 queries:

  `https://drive.google.com/file/d/18lEba0AzpLdAqHhe13Ah3fL2b4YEyAmF`

- Mapping & localization in random trajectory:

  `https://drive.google.com/file/d/19IX93ppGeQ56CqpgvN5MJ2pCl46FjgkO`

- Joint exploration, mapping & localization:

  `https://drive.google.com/file/d/1UdTmxUedRfC-E6b-Kz-1ZqDRnzXV4PMM`

2.5 Conclusion

Multigrid memory represents a groundbreaking approach to augmenting networks with long-term, large-scale storage. A simple principle, multigrid connectivity, underlies our model. Residual networks He et al. [2016], which provide shortcut pathways across depth, have had enormous impact, allowing very deep networks to be trained by facilitating gradient propagation. Multigrid wiring is complementary, improving connectivity across an orthogonal aspect of the network: the spatial dimension. Its impact is equally significant: multigrid wiring exponentially improves internal data routing efficiency, allowing complex behaviors (attention) and coherent memory subsystems to emerge from training simple components. Our results are cause to rethink the prevailing design principles for neural network architectures.

CHAPTER 3

DOMAIN-AGNOSTIC PROCESSING

3.1 Introduction

Data in our world exists in diverse forms with their respective inherent properties. Some lie in 1D space, e.g., natural language, stock price, etc., while others hold their meaning in higher dimensional spaces, e.g., images in 2D, point clouds in 3D, etc. Meanwhile, certain data types do not have a clear boundary about which subspace they reside in, or may express their properties in multiple subspaces. For example, although DNA sequences are treated as a 1D structure in many settings, [Anjum et al., 2019] discovers that the proximity in space of their folding structure may reveal other properties, and shows better prediction accuracy of enhancers when processing the sequences in a particular 2D arrangement.

At the same time, current state of neural network architectures has seen polarized approaches in dealing with different types of data. 2D convolutional operations have been the default choice for processing 2D images, while 1D convolutional filters or fully connected networks are reserved for 1D tasks such as NLP or time series. A similar situation also appears in memory-based systems. Convolutional variants of LSTM and GRU have been shown to be especially effective in dealing with image sequence problems, yet their fully-connected variants are still the default choice for 1D tasks. This polarization leads to the prevalent designs of problem-specific architectures. These designs are conditioned on certain assumption about the data structure, and also contradicts with requirements to build an AGI system. Furthermore, certain problems involve multi-modality. For instance, solving a VQA problem requires a system capable of simultaneously processing images, natural language, and effectively unify information from these sources for inference. Utilizing different processing schemes for separate sources appears to be not ideal, since it may induce difficulty in joining these sources of information. Most contemporary approaches in this domain opt to use a single architectural choice for processing these sources of information, e.g., fully-connected LSTM, at the expense of potentially sub-optimal performance on the data source that is not best

suited to this architecture.

To address the aforementioned problems, in this Chapter, we propose a domain-agnostic processing scheme in which a single processing mechanism is applied to inputs regardless of the nature of data domain. The method is designed to be generic and could be effective in handling either 1D, 2D, or higher dimensional data.

3.2 Related Work

Multi-purpose designs. Certain works attempt to generalize the design to be capable of handling multiple tasks. [Kokkinos, 2017] proposes a UberNet architecture and joinly train a handful of vision tasks. However, the architecture is solely designed for image-based tasks, and only works for non-memory tasks. [Graves et al., 2014] design a neural architecture analogous to a turing machine, where the controllers modeled by neural networks interact with external memory banks in a differentiable fashion. Although this architecture is generic in principle, it is only evaluated on tasks with one-dimensional data structure, and it is not clear how it would perform on tasks without memory requirement. In fact, the results we present in Section 2.4 already show that this line of architectures, as represented by its successor [Graves et al., 2016], performs poorly on 2D input data.

Multi-modality. Certain tasks require simultaneous handling of multiple data sources of different nature. A notable line of works in this space considers problems involving vision and language, e.g., visual question answering [Antol et al., 2015], video question answering [Jang et al., 2017], image captioning [Xu et al., 2015], etc. In these settings, most existing works involve specially designed systems to process images and text separately. For maintaining and unifying memory information from images and text streams, the same fully connected LSTM structure is used in both streams, at the expense of possibly being a suboptimal choice for the image stream.

Spatial mapping. Corcoran et al. [2018] propose a mapping of 3D structures into 2D and 1D based on Hilbert curve for more efficient processing while still preserving representation quality. On the other hand, Yin et al. [2018] employ Hilbert curve to represent 1D DNA sequences in the

form of 2D structures for DNA classification based on 2D convolutions. These works demonstrate the capability to preserve the representational power through Hilbert-based mapping, however the authors only propose and experiment the mapping for the specific data type being considered.

Positional Encodings. In attention-based models [Vaswani et al., 2017, Zhao et al., 2020, Woo et al., 2018, Bello et al., 2019], positional encodings are used to alleviate the permutation-equavariant property of attentional operations, making the network dependent on location and order of inputs. In neural rendering [Mildenhall et al., 2020], similar to Transformers [Vaswani et al., 2017], the sinusoidal mapping of positional information is utilized, however in this case the mapping is used to generate high frequency inputs to help the network more effective at approximating a higher frequency function.

3.3 Domain-Agnostic Processing

2D convolutions have been the gold standard in processing 2D data such as images. Similarly, 3D convolutions are a natural extension to 3D data structures, while 1D data is predominantly processed by fully connected networks or 1D convolutions. Certain architectures rely on hand-designed attention mechanisms to connect information from elements in 1D data. To attain data-agnostic property while maintaining data-specific processing capability, we propose to transform and parallelly process data in different subspaces, while guaranteeing these processings are complementary. For parameter efficiency, we propose to process data entirely based on convolutional operations. The operations are further wrapped within multigrid connections, which are capable of approximating fully connected networks and allow attentional capability to emerge as needed. We defer the details regarding embedding the components inside multigrid connections to Section 3.3.3, and discuss the spatial mapping details in the following.

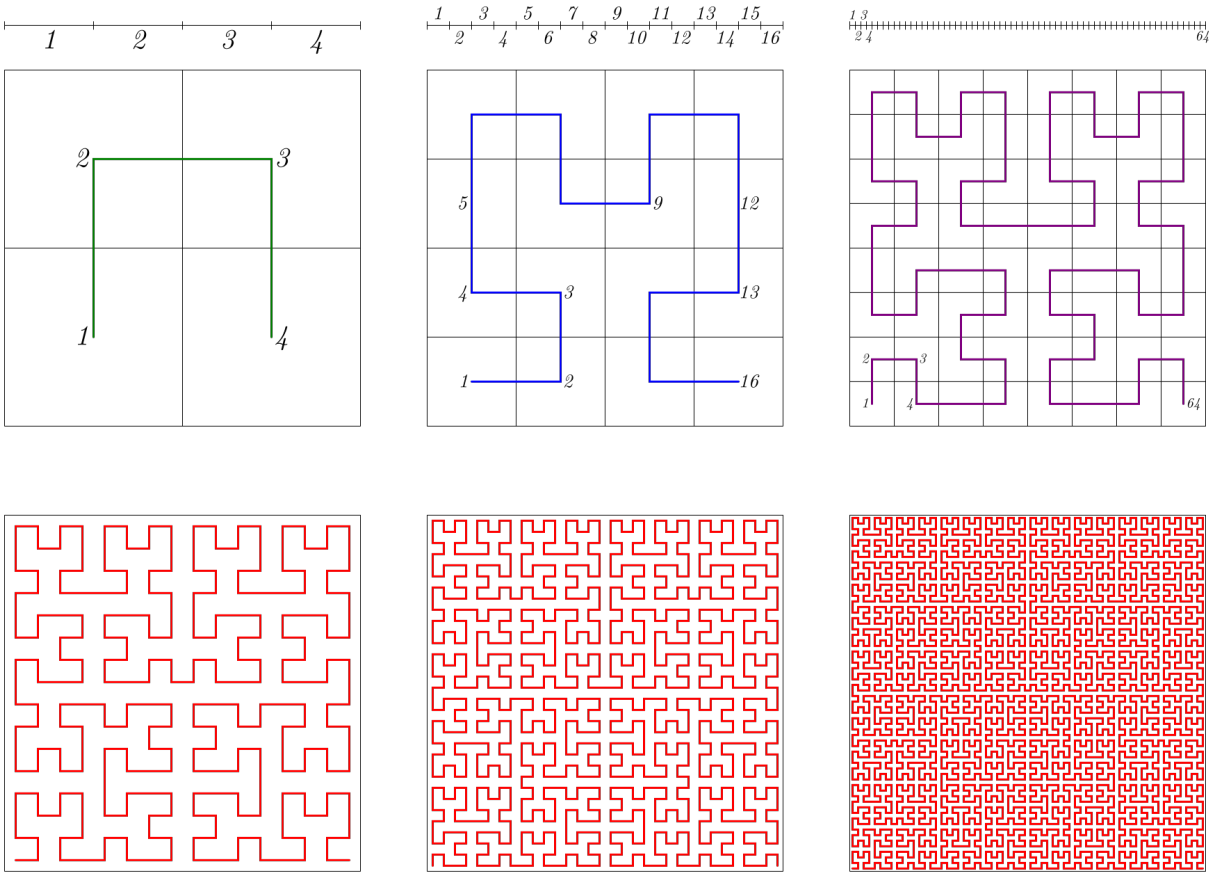


Figure 3.1: **2D Hilbert curve [Wikipedia, b].** ***Top:*** 2D Hilbert curves of first, second, and third orders and their respective 1D arrangements. ***Bottom:*** 2D Hilbert curves of fourth, fifth, and sixth orders.

3.3.1 Hilbert-based Spatial Mapping

Hilbert curve is an instance of space filling curves. These curves define corresponding spatial arrangement of data elements among 1D, 2D, and higher dimensional spaces. In this family, Hilbert curve is known to be superior in preserving data locality when mapping from one space to another. [Corcoran et al., 2018] demonstrate the effectiveness of processing 3D data in 2D and 1D spaces by employing the Hilbert-based spatial mapping. Similarly, Yin et al. [2018] find superior performance when processing 1D DNA sequences in 2D space utilizing Hilbert-based mapping.

Sample 2D Hilbert curves and their corresponding 1D arrangements are given in Figure 3.1, while their generation rules and 3D extension are shown in Figure 3.2. In this work, we propose

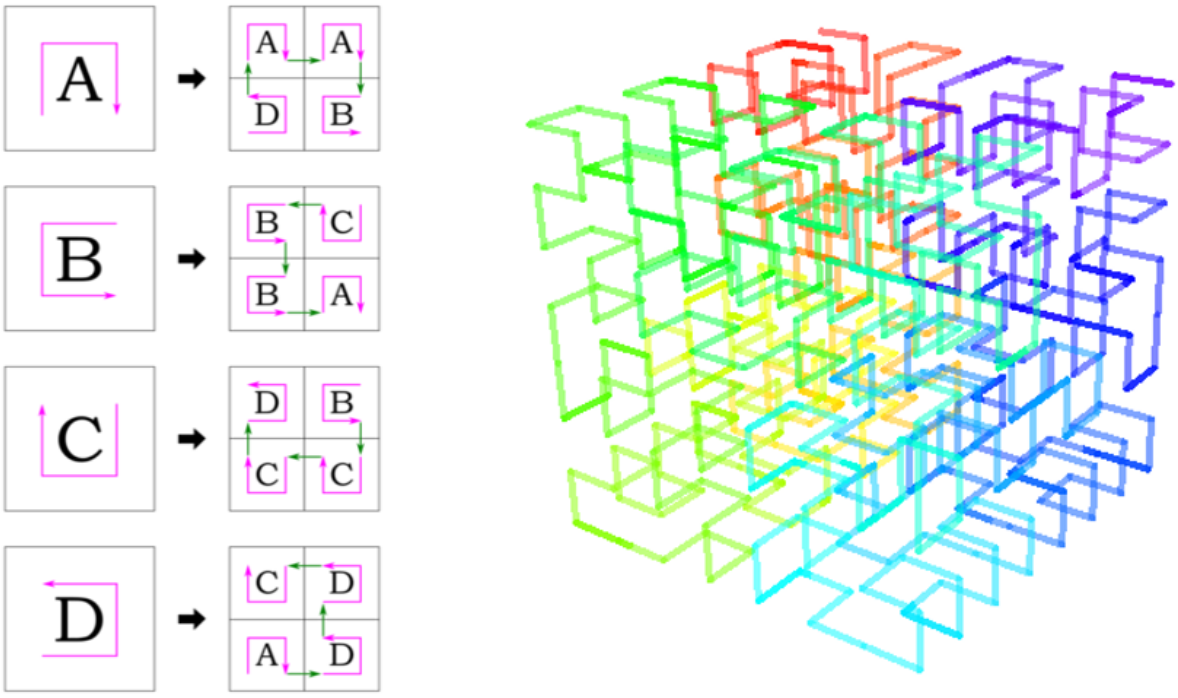


Figure 3.2: **2D Hilbert curve generation rules & 3D Hilbert curve [Wikipedia, a].** ***Left:*** 2D Hilbert curve generation rules. ***Right:*** 3D Hilbert curve.

to transform data into multiple subspaces based on Hilbert curve, and parallelly process and unify them at each stage, while guaranteeing that these processes are complementary. This helps to attain a single processing mechanism regardless of data type, while could also potentially enable emergence of new representational power. While locality is mostly reserved by Hilbert curve in different subspaces, there are still some caveats. We propose to further enhance the process by positional encoding and multigrid connections.

3.3.2 Positional Encoding

Hilbert curve is known to be the best space-filling curve in preserving locality property in different subspaces. However, there are subtle exceptions in certain cases. For example, when arranging 1D data into 2D structure, certain points far away in 1D could be folded and become neighboring in 2D, as can be seen in Figure 3.1. To resolve this situation, we propose to add positional encoding to the data structure. Positional encoding can be fixed or learned. For fixed encoding, we can simply

pad data points with their corresponding location information in the subspace (1D, 2D, or higher dimensions).

The combination of Hilbert curve, positional encoding and multigrid connections allows for maximizing locality preservation as needed. For example, locality property in 1D and 2D conversions is preserved as follows:

- Close in 1D $\rightarrow$ close in 2D: preserved by Hilbert curve.
- Close in 2D $\rightarrow$ close in 1D: preserved by Hilbert curve and multigrid connections. Hilbert curve mostly preserve this property. When certain points being close in 2D but are far in 1D, this is alleviated by multigrid connections, which enable exponentially fast information routing, and in turn help to connect these points seemingly far away in 1D.
- Far in 1D $\rightarrow$ far in 2D: preserved by Hilbert curve and positional encoding. In cases where the data points are far in 1D but close in 2D along the Hilbert curve, positional encoding enable the filters to adapt depending on the location. These adapted filter can learn to not connect these 2D neighbors as needed.
- Far in 2D $\rightarrow$ far in 1D: preserved by Hilbert curve.

There are different choices one can make regarding the type of positional encodings. In this work, we investigate the following options:

- Fixed spatial encoding: The positional encoding is a linear mapping corresponding to the spatial coordinates along the tensor. Particularly, for a tensor of shape $W \times H \times C$ (corresponding to width, height, and channel dimensions), the positional encoding at location (x, y) $(1 \leq x \leq W, 1 \leq y \leq H)$ is $[x/W, y/H]$.
- Fixed sinusoidal encoding: The positional encoding is a non-linear mapping of the spatial coordinates by means of sinusoidal functions. At location (x, y) of the tensor with shape $W \times H \times C$, the sinusoidal positional encoding is characterized as $[sin(\frac{\pi}{2} \cdot \frac{x}{W}), cos(\frac{\pi}{2} \cdot \frac{x}{W}), sin(\frac{\pi}{2} \cdot \frac{y}{H}), cos(\frac{\pi}{2} \cdot \frac{y}{H})]$

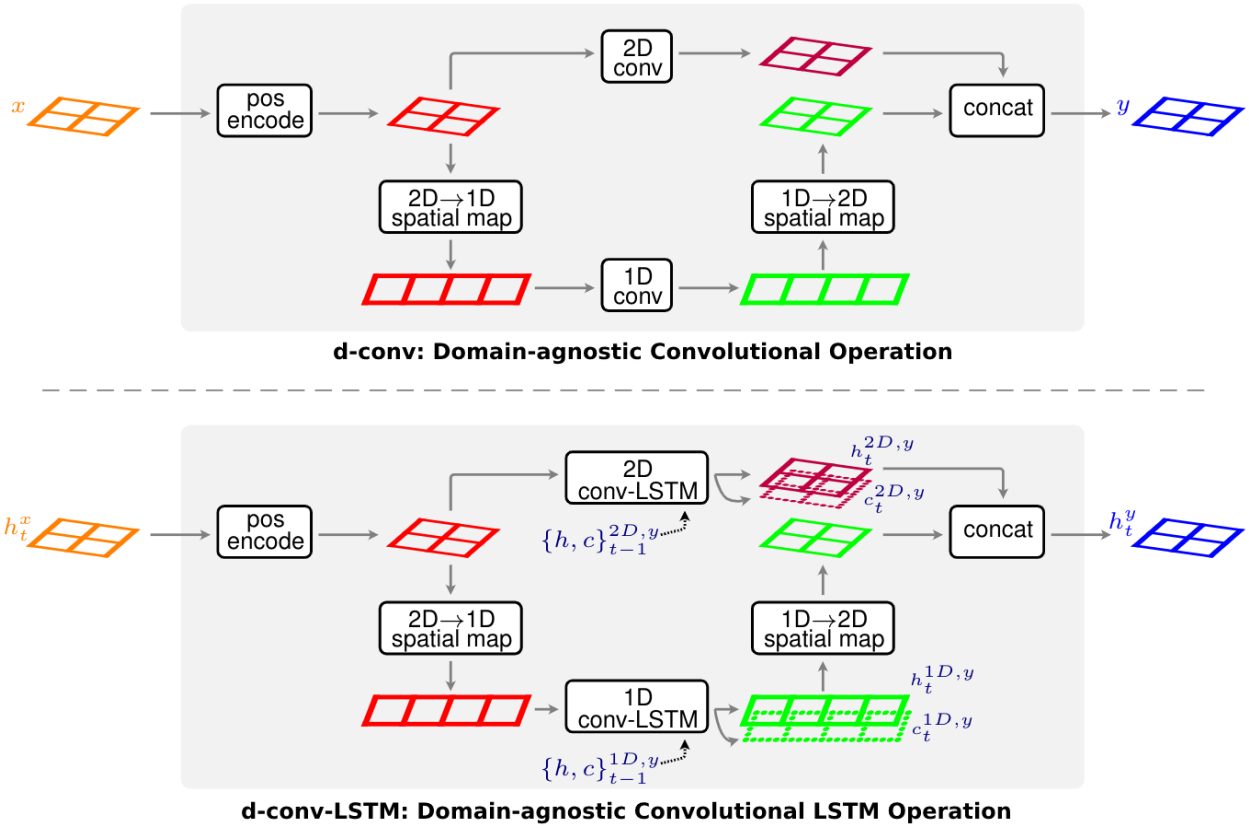


Figure 3.3: **Domain-agnostic convolutional operations.**

- Learned encoding: The positional encoding is not predetermined, but rather learned during training. They are learnable parameters at each location of the input tensor.

3.3.3 Domain-agnostic Processing

With Hilbert curve and positional encoding, we propose to process data the same way regardless of input data structure by transforming and parallelly processing data in multiple subspaces. Figure 3.3 diagrams this process for both feed-forward convolutions and recurrent operations. We denote these domain-agnostic operations as **d-conv** and **d-conv-LSTM** for the versions without and with memory, respectively.

As mentioned in Section 3.3.2, the data locality is preserved by both positional encoding and multigrid connections. Therefore, with the mentioned basic domain-agnostic operations, we further wrap them in a multigrid structure as demonstrated in Figure 3.4, in which **dm-conv** can be either

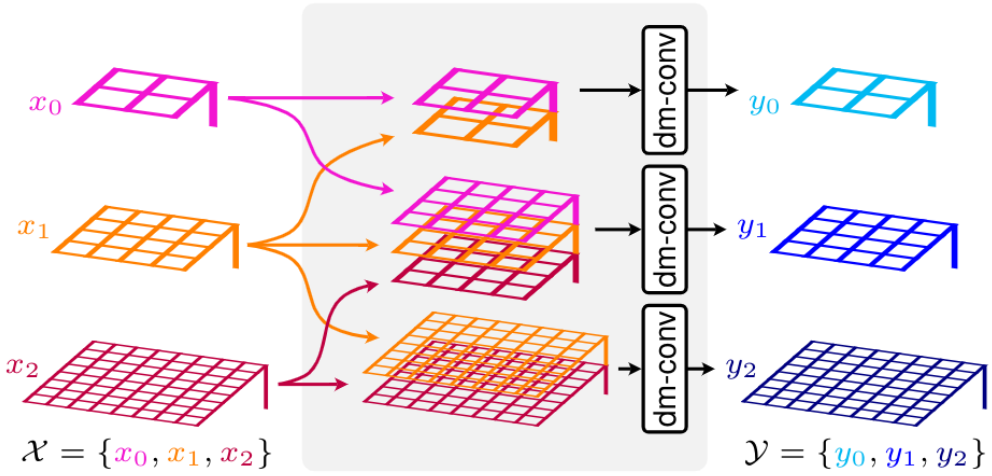


Figure 3.4: **Domain-agnostic multigrid convolutional layer.**

d-conv-LSTM or **d-conv-LSTM**.

3.4 Experiments

3.4.1 Hilbert-based Geometry

First, we investigate the behaviors of Hilbert-based geometry in a single-domain setting. For this purpose, we experiment with different geometric configurations in a pure image recognition task, without memory, and without multigrid connectivity.

Dataset: We utilize CIFAR-10 classification dataset [Krizhevsky, 2009]. The dataset has 10 object classes, with 50000 training images, and 10000 test images.

Architecture: We employ architectures based on ResNet-18 [He et al., 2016].

Training Details: We use a batch size of 128, with a learning rate of 0.1. We train for 200 epochs, in which the learning rate is decayed by a factor of 0.2 every 30 epochs. The models are optimized by SGD with momentum 0.9 and weight decay 0.0005.

Results

1D and 2D Geometries: Figure 3.5 compares the performance of processing methods based on

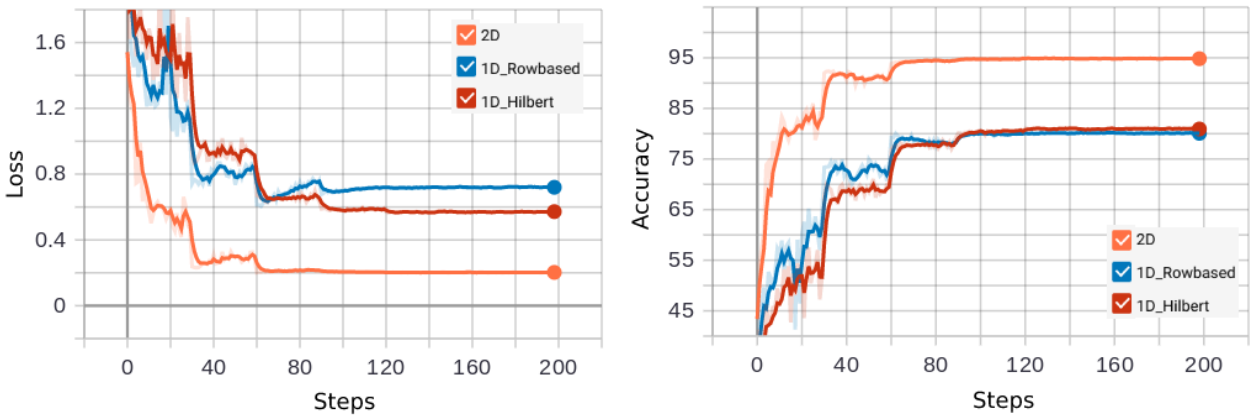


Figure 3.5: **Performance of different geometries in CIFAR-10 image recognition. Left:** Test loss. **Right:** Test accuracy.

Hilbert 1D, row-based 1D (row-based reshaping from 2D to 1D), and 2D geometries. The results suggest that Hilbert 1D is slightly better than row-based 1D, while 2D filters are substantially more effective compared to 1D versions in this setting. The efficacy of 2D filters is expected, as the input domain and the problem setting only involve 2D structured data. On the other hand, it is fairly surprising that the Hilbert 1D alone compares favorably to row-based 1D geometry, as row-based 1D still maintains the natural horizontal patterns in images. Perhaps because of this initial advantage, row-based 1D tends to learn faster at the beginning. However, after the slow start, Hilbert 1D accelerates and eventually exceeds row-based 1D in final performance. This result suggests the potential effectiveness of Hilbert 1D compared to other 1D geometries.

Positional Encoding: Figure 3.6 demonstrates the effect of positional encoding on Hilbert versus row-based 1D. Both methods benefit from positional encoding, however Hilbert 1D notably obtains more advantage with the augmented location information. The improvement brought by positional encoding in Hilbert 1D is almost twice of the improvement seen in row-based 1D. This result validates our hypothesis about the potential effect of location information in Hilbert geometry, as analyzed in Section 3.3.2.

Joint Geometries: Next, we analyze the effect of different strategies in combining multiple 1D geometries. The results are presented in Figure 3.7. The "OneStream" settings are the ones where different 1D geometries are concatenated into a combined input, and jointly processed in a single

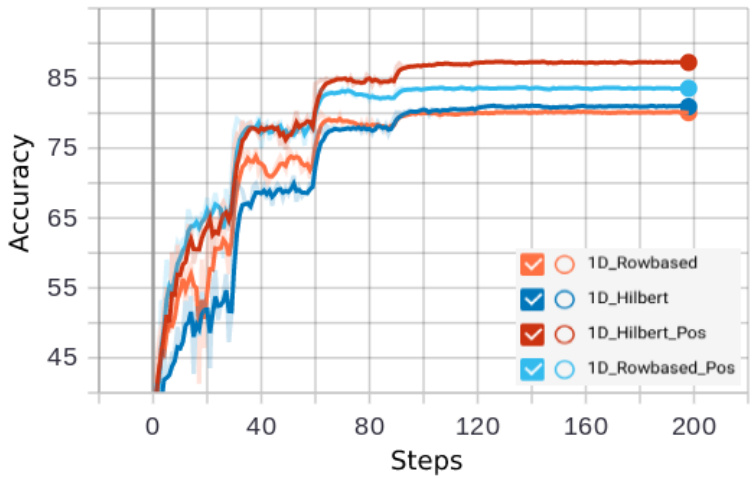


Figure 3.6: **Effect of positional encoding for different 1D geometries in CIFAR-10.**

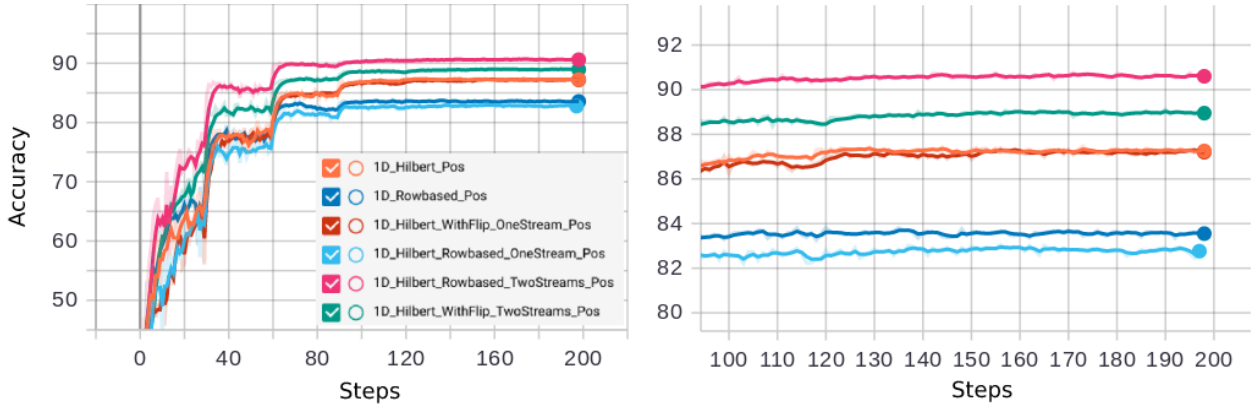


Figure 3.7: **Test accuracy of different settings in jointly processing multiple 1D geometries. Left:** Full view. **Right:** Zoom-in of the last steps.

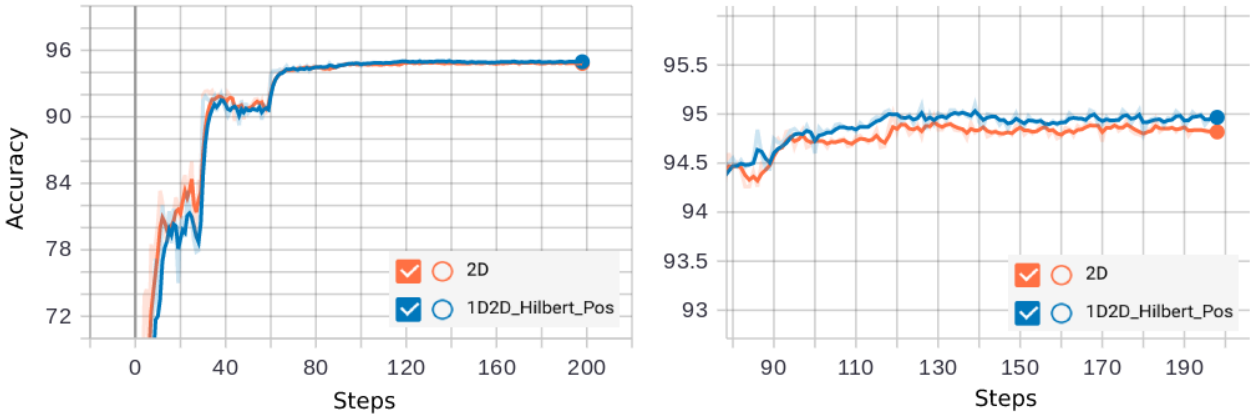


Figure 3.8: **Test accuracy of 2D versus Hilbert-based 1D-2D processing. Left:** Full view. **Right:** Zoom-in of the last steps.

stream of 1D filters. On the other hand, the "TwoStreams" settings are the cases where at each layer, we convert 2D input into corresponding 1D geometries and process each 1D geometry separately with its own 1D filters before converting the outputs back to 2D geometries and concatenating them. Figure 3.7 suggests that combining different geometries using the "OneStream" strategy actually hurts the performance instead of helping. The accuracy of using multiple geometries as input in "OneStream" strategy (e.g., 1D_Hilbert_Rowbased_OneStream_Pos) is consistently lower than the results using corresponding single geometries (e.g., 1D_Hilbert_Pos and 1D_Rowbased_Pos). The "WithFlip" notation means using an additional flipped version of Hilbert curve as another 1D geometry. We hypothesize that the ineffectiveness of the "OneStream" setting is due to the undesirable correlation between unrelated pixels being added to the input in this case. On the other hand, the "TwoStreams" strategy proves to be an effective method for jointly processing in multiple subspaces. Different than the "OneStream" strategy, the "TwoStreams" version brings consistent improvement compared to using a single geometry. This result validates the effectiveness of our designed joint processing scheme as introduced in Section 3.3.3.

Besides joining different 1D geometries, we also experiment with joining 1D and 2D geometries to see how they compare. Figure 3.8 shows the results of regular 2D versus Hilbert 1D-2D processing strategies. We see that while the 2D filters alone seem to learn a bit faster at the beginning, the two methods converge to similar accuracy (in fact, the Hilbert 1D-2D seems to converge to a slightly better accuracy in the end). This result suggests that the proposed 1D-2D processing could be a potential generic and universal processing scheme, as it does not degrade performance when using 1D geometry alongside regular 2D processing. Note that the 2D and Hilbert 1D-2D versions have been calibrated to have similar number of parameters for fair comparison.

Generalization Capability: Figure 3.9 shows the training and testing accuracy of row-based and Hilbert 1D geometries. The results suggest that Hilbert 1D is more generalizable compared to row-based 1D. During training, row-based 1D versions (with and without positional encoding) attain near perfect accuracy, while Hilbert 1D have relatively modest training performance. However, this trend is reversed in performance on the test set, in which Hilbert 1D versions exhibit notably better

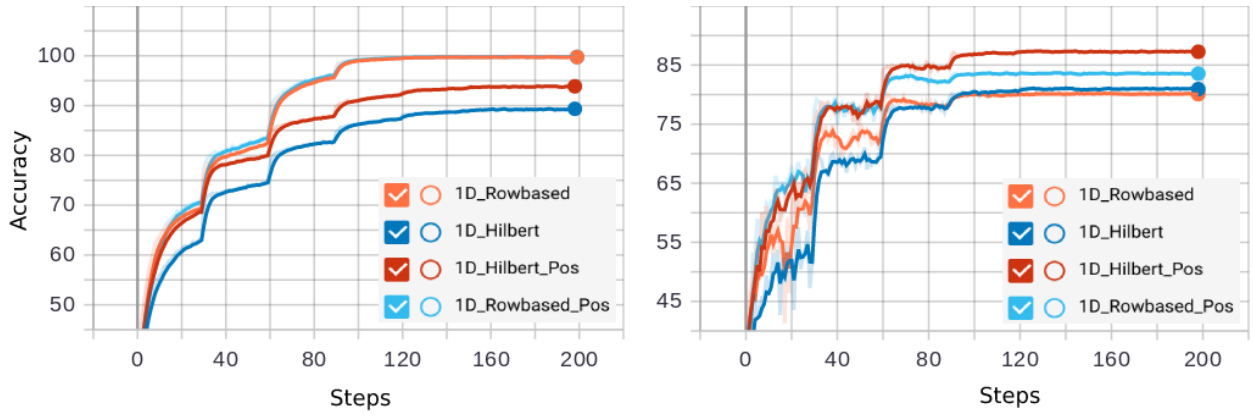


Figure 3.9: **Generalization property of Row-based versus Hilbert 1D geometries. Left:** Train accuracy. **Right:** Test accuracy.

accuracy. The results further reinforce the effectiveness of Hilbert 1D geometry by indicating that row-based 1D tends to be more overfitting compared to Hilbert 1D.

3.4.2 Domain-agnostic Processing

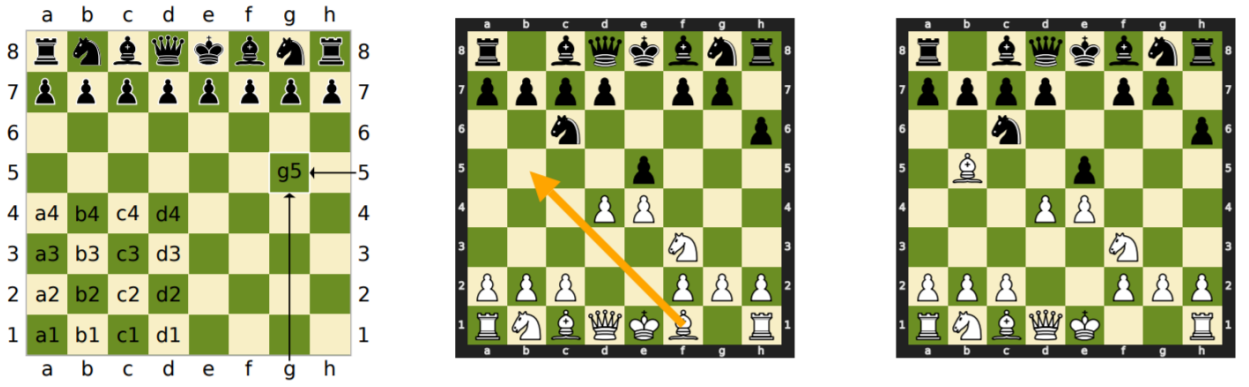
In this section, to investigate the effect of domain-agnostic processing, we study the performance of the proposed methods in contexts involving multiple input domains, or when the nature of the problem requires reasoning in different domain geometries. Also, the proposed structure as a whole is being investigated, with the presence of neural memory and multigrid connectivity.

We will first introduce the tasks being considered, and then present the results of various studies in the following sections.

Tasks

MNIST Recall: The associative recall task on MNIST dataset was introduced in Section 2.4.3. While this task only involves 2D input (MNIST images) at each time step, the associative recall task itself is 1D in nature, in which the sequence of digits along the time dimension is inputted to the model. Therefore, a joint processing in 1D and 2D domains could be beneficial to the task.

Architecture: The architecture being used is also similar to the one described in Section 2.4.3. However, here we utilize a smaller network with the 5-layer writer, and only three spatial scales are



(a) Square naming

(b) Board state before (left) and after (right) the bishop at `f1` is moved to `b5`. UCI notation represents the move as `f1b5`.

Figure 3.10: **Chess notation (courtesy of [Toshniwal et al., 2021]).**

used: 4x4, 8x8, 16x16.

Blindfold Chess: Playing chess blindfolded has been recently proposed as a benchmark for evaluating language models [Toshniwal et al., 2021, Noever et al., 2020]. In this task, the agent needs to predict the next move given the previous moves. It is "blindfolded" as the agent does not have access to the underlying world state of the chess board. Instead, the task is treated as a next token text prediction task, in which the entire input is a series of text tokens, each represents a location in the chess board. A move is a pair of two tokens, the starting and ending location. An example of this setting is provided in Figure 3.10 (courtesy of [Toshniwal et al., 2021]).

While the inputs are only a 1D series of text tokens, mastering this task requires inferring and manipulating the underlying state of the 2D chess board. An architecture effectively handling both 1D and 2D data domains may exhibit advantages in this case. Therefore, we propose to investigate the domain-agnostic processing scheme in this context.

Dataset: We use a small version of the dataset provided by [Toshniwal et al., 2021]. Specifically, the same dev and test sets (15K games each) as in [Toshniwal et al., 2021] are used, while the training set is reduced to include 5K games to accommodate training cost.

Architecture: We employ a six-layer multigrid architecture similar to the one used in question answering task (Figure 2.12), in which the first three layers are multigrid memory layers, while the last three are multigrid convolutional layers.

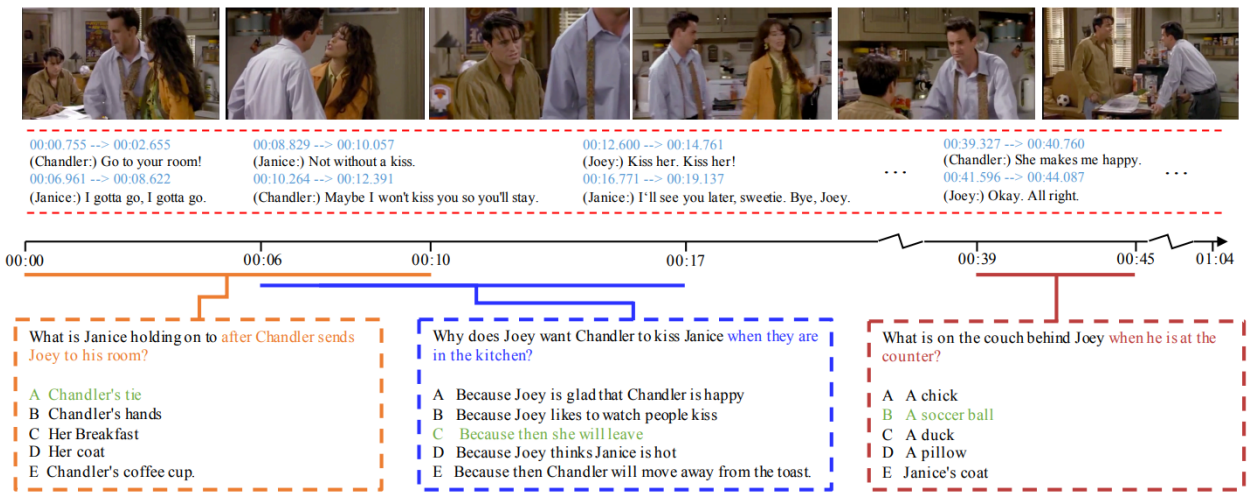


Figure 3.11: **Examples from the TVQA dataset (courtesy of [Lei et al., 2018]).**

Video Question Answering Beyond the single-modal question answering task based solely on one source of textual inputs as introduced in Section 2.4.4, there has been a growing interest in more complex settings where an answer is grounded on multiple sources of information, e.g., images, videos, captions, subtitles [Lei et al., 2018, Liu et al., 2020]. Solving these tasks requires effective manipulation of data across different domains. Therefore, it is a natural fit to investigate the effect of the proposed domain-agnostic processing scheme. In this work, we experiment with the video question answering task proposed by Lei et al. [2018], in which there are four streams of inputs, i.e. the video, subtitle, question, and answer. Some examples of this task is provided in Figure 3.11 (courtesy of Lei et al. [2018]).

Dataset: The TVQA dataset [Lei et al., 2018] contains 152.5K questions from 21.8K video clips of 6 TV shows. Each question has five candidate answers, one of which is the correct answer. From this dataset, 80% is splitted for training, 10% for validation, and 10% for testing.

Architecture: We employ a multigrid based architecture as shown in Figure 3.12. The model has four main components: a textual multigrid portion that processes 1D textual inputs (questions, answers, subtitles), a video frame multigrid processing 2D images, a domain-agnostic multigrid memory module jointly handling the information from both 1D and 2D data streams (video frames, subtitles, questions, and answers) at each time step, and a domain-agnostic convolutional multigrid

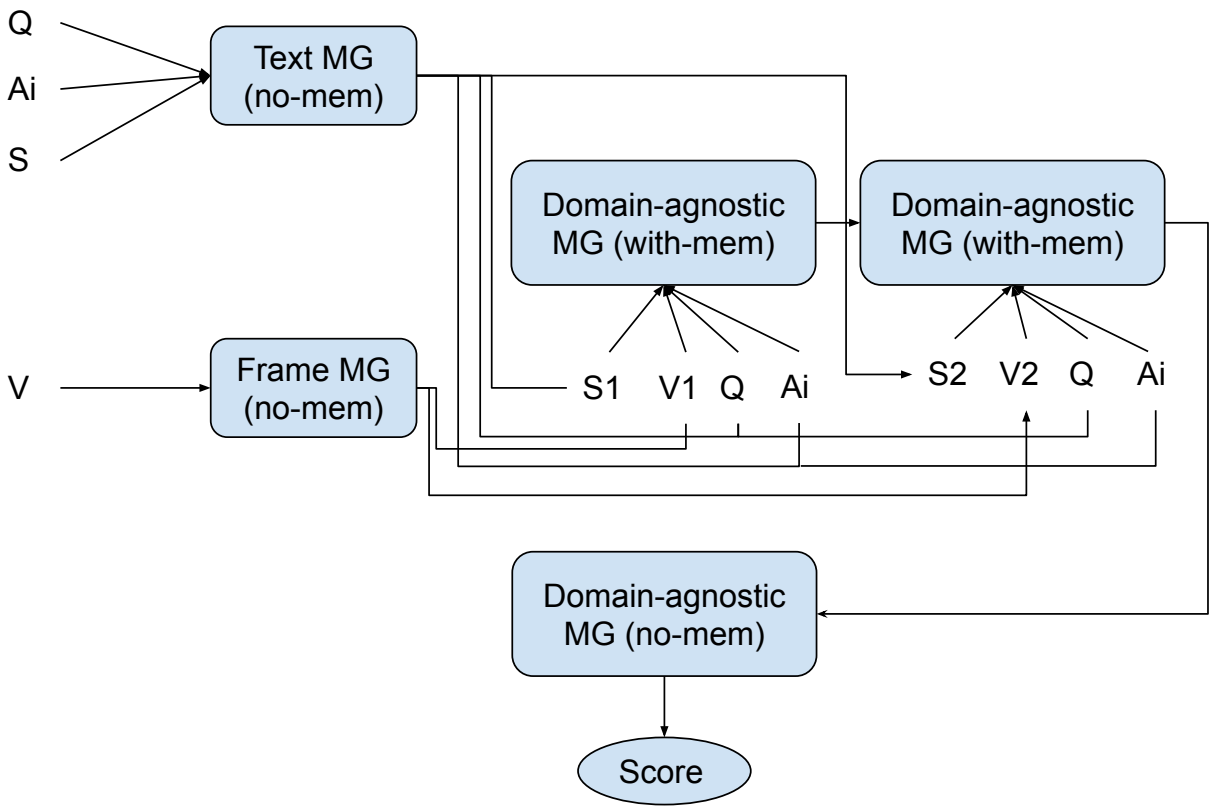


Figure 3.12: **Architecture in video question answering task.** Text MG module processes 1D inputs from questions (Q), answers (A), and subtitles (S). Frame MG handles 2D data from video (V) frames. Domain-agnostic MG (with-mem) is a memory module that jointly processes information from 1D and 2D sources at each time step. Finally, the domain-agnostic MG (no-mem) decodes the outputs from the memory module into the final score for each answer.

module that decodes the output from the memory module to infer the final score for each answer.

Positional Encodings

Rowbased 1D, Hilbert 1D, and 2D filters: Without Positional Encoding

We first investigate the relative performance of different 1D and 2D filters in this setting. Figure 3.13 (left) shows the training loss of regular 2D, row-based 1D, and Hilbert-based 1D filters in MNIST recall task. As we can see, 2D filter is the best fit in this task by converging to the lowest loss, followed very closely by rowbased 1D filter. On the other hand, the pure Hilbert 1D filter does not seem to perform as well. The underlying data structure is 2D images, which makes 2D filters a

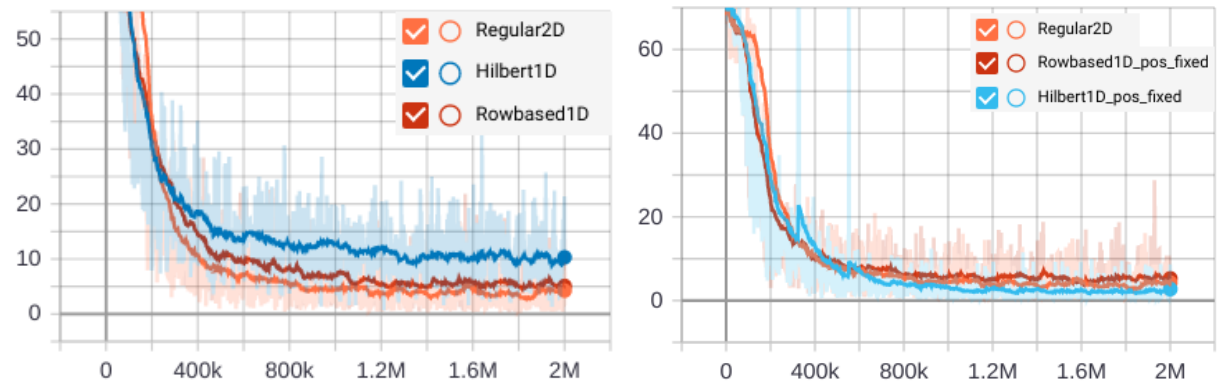


Figure 3.13: **Training loss in MNIST recall task with 2D versus 1D filters. Left:** Without positional encoding. **Right:** With positional encoding.

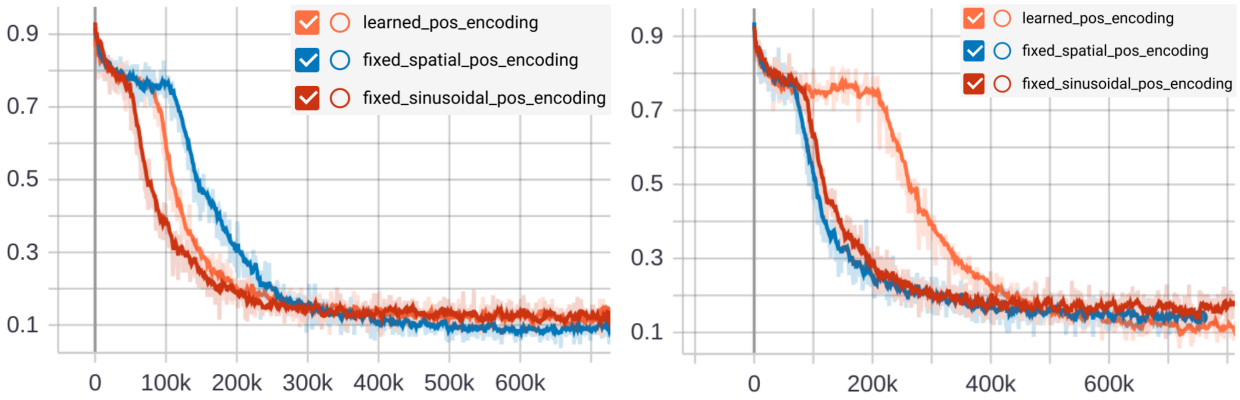


Figure 3.14: **Validation error with different positional encodings in MNIST recall task. Left:** Positional encodings at input layer. **Right:** Positional encodings at all layers.

natural fit. The row-based 1D filter is less optimized, yet still effectively captures the row-based patterns of the images. The Hilbert 1D filter, however, needs to process different patterns along the Hilbert geometry by a single filter, making it less efficient during training.

Rowbased 1D, Hilbert 1D, and 2D filters: With Positional Encoding

Next, we investigate the effect of positional encodings on top of the filters, which is demonstrated in Figure 3.13 (right). While the 1D row-based filter benefits little from the positional encodings, the Hilbert 1D filter improves considerably when accompanying positional encodings. We postulate that this could be due to the fact that the 1D row-based filter operates on a consistent row-based geometry, making it already effective to extract row-based features. Adding positional encodings therefore may not make significant difference. On the other hand, as a single Hilbert 1D filter is not effective

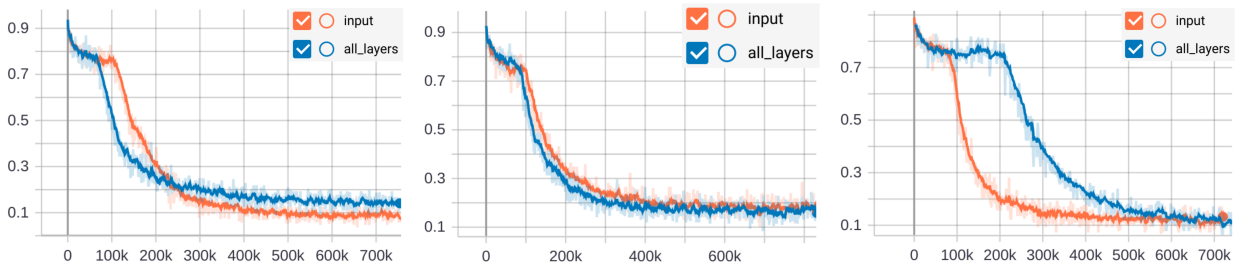


Figure 3.15: **Validation error with positional encodings at input layer versus all layers in MNIST recall task. Left:** Fixed spatial positional encoding. **Middle:** Fixed sinusoidal encoding. **Right:** Learned positional encoding.

at handling various geometries at once, adding positional encodings helps distinguishing different geometry patterns at different locations, and helps to adapt the filter based on these geometry patterns. The fact that Hilbert 1D benefits more with location information is also consistent with our observation in Section 3.4.1. Even more interesting is the fact that the Hilbert 1D filter does not only improve when having access to positional encodings, but it also performs even better than the 2D and row-based 1D filter. This could be due to the fact that while the local geometry of 2D and row-based 1D filters is fixed, Hilbert 1D filter processes varying local geometries, which potentially can exploit more information when being executed effectively. Another factor could be because of the dataset nature, where MNIST digits could be easier to be serialized into a single stroke, making them suitable to be processed by Hilbert 1D geometry.

Positional Encodings: Fixed Spatial, Fixed Sinusoidal, and Learned Encodings

As there are different positional encodings, and different strategies to embed the positional encodings into the model, we further investigate the effect of these strategies.

Figure 3.14 compares the validation error in MNIST recall task when using different positional encodings. The left figure indicates the error rate when the positional encodings are presented at the input layer, while the right figures corresponds to having positional encodings embedded in all layers of the network. The figures suggest that while the speed of convergence could be different depending on the types of positional encoding, they tend to converge to similar performance eventually. Notably, Figure 3.14 (right) suggests that when embedding the positional encodings at all layers, the learned encoding converges considerably slower compared to the fixed encodings.

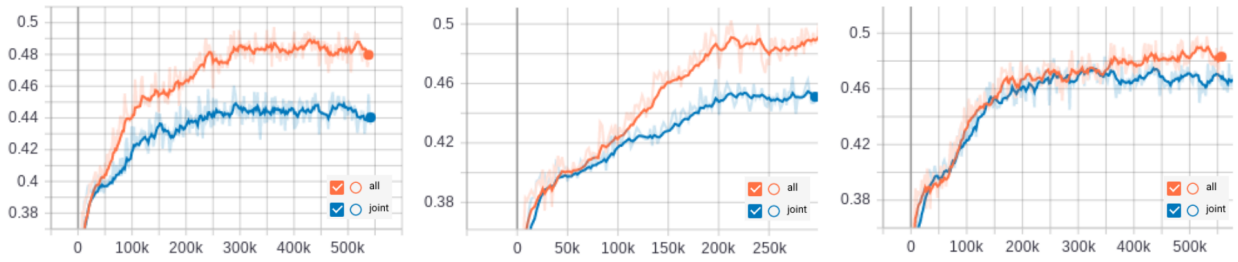


Figure 3.16: **Validation accuracy with hilbert processing at the joint part of different domain streams versus at everywhere in the network (TVQA task). Left:** Without positional encoding. **Middle:** Fixed spatial positional encoding. **Right:** Learned positional encoding.

This could be due to the fact that in case of the learned encoding, the network needs to adjust the encoding at all layers, therefore, it takes some time to tune the encoding and converge.

Figure 3.15 shows the validation error when adding positional encodings at input layer versus all layers. The results suggest that positional encodings at the input layer are sufficient. Adding positional encodings to all layers do not noticeably improve performance. In fact, in case of the fixed spatial encoding, having the encoding at input layer is even better than having at all layers. This could be because the positional information at input layer is sufficient to propagate through the network. Adding more positional encodings at all layers could be redundant, and may even have adverse effect (slow convergence in learned positional encoding, and higher convergence error in fixed spatial positional encoding).

Placement of Hilbert-based Domain-agnostic Processing

To investigate whether the Hilbert-based domain-agnostic processing is only helpful when there are data from different domains or it could be beneficial in more generic cases, we perform ablation studies on TVQA task. Specifically, we perform experiments in two settings: when the Hilbert-based processing only exists in the domain-agnostic MG modules in Figure 3.12, and the Hilbert-based processing is applied to all modules (extending to Text MG and Frame MG in Figure 3.12). Results are shown in Figure 3.16. It is clear that the Hilbert-based processing is effective even when applied to a single source of data domain. Regardless of the type of positional encoding used, having Hilbert-based processing in the whole network consistently outperforms the case where Hilbert

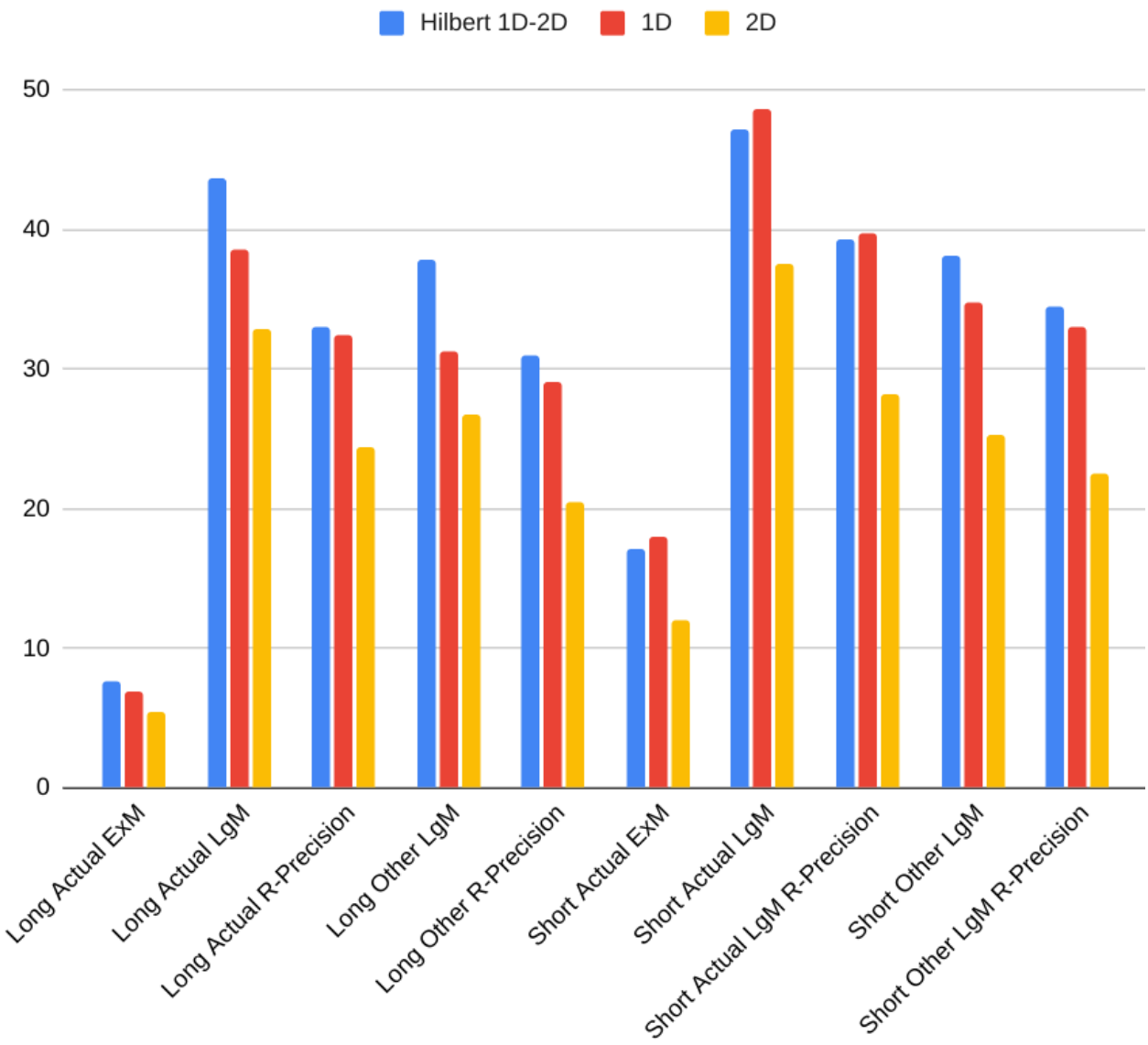


Figure 3.17: **Test accuracy (%) of 1D, 2D, and Hilbert 1D-2D processing in Chess.**

processing only present at the portions involving multiple data domains. The result shows the generality of the proposed processing scheme.

Comparison of 1D, 2D, and Hilbert-based Domain-Agnostic Processing

Finally, we evaluate the performance of 1D, 2D, and Hilbert-based 1D-2D filters on the three considered tasks: MNIST recall, Blindfold Chess, and TVQA.

Figure 3.17 and Table 3.1 show test accuracy of 1D, 2D, and Hilbert 1D-2D filters in the

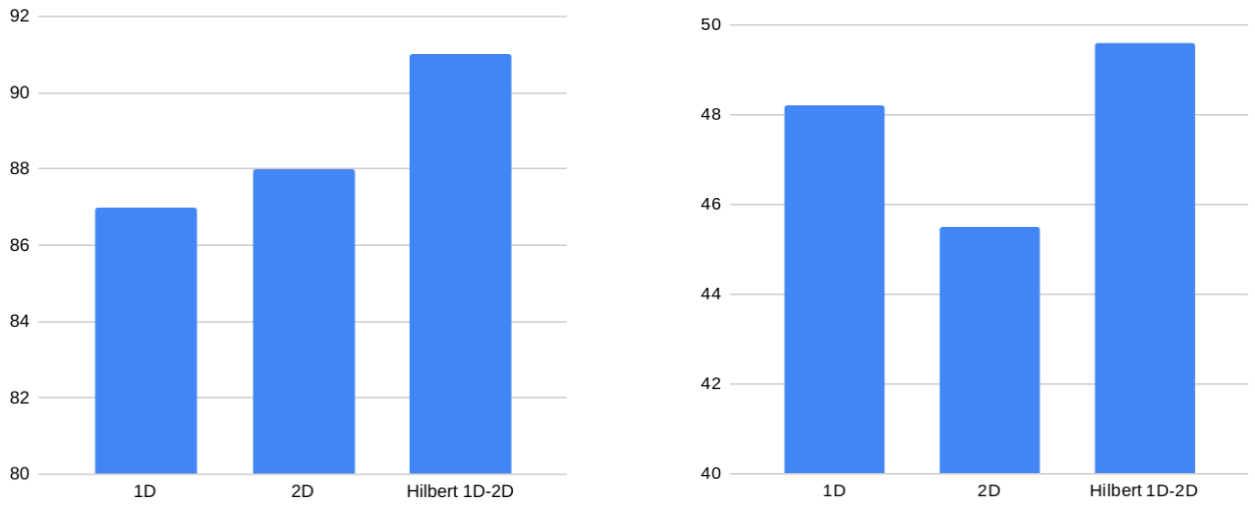


Figure 3.18: **Test accuracy (%) of 1D, 2D, and Hilbert 1D-2D processing. Left:** MNIST recall task. **Right:** TVQA task.

Table 3.1: **Test accuracy (%) of 1D, 2D, and Hilbert 1D-2D processing on the Chess task.**

Metric	Model (#Params)		
	1D (886K)	2D (539K)	Hilbert 1D-2D (528K)
Long Actual ExM	6.9	5.5	**7.6**
Long Actual LgM	38.6	32.9	**43.6**
Long Actual R-Precision	32.4	24.5	**33.1**
Long Other LgM	31.2	26.7	**37.8**
Long Other R-Precision	29.2	20.5	**30.9**
Short Actual ExM	**18.0**	12.0	17.1
Short Actual LgM	**48.6**	37.5	47.1
Short Actual LgM R-Precision	**39.7**	28.2	39.3
Short Other LgM	34.8	25.3	**38.1**
Short Other LgM R-Precision	33.1	22.5	**34.4**

blindfold Chess task. As we can see, with similar number of parameters, Hilbert 1D-2D significantly outperforms 2D filter in all measures. The Hilbert 1D-2D processing also outperforms 1D filter in most measures despite having less parameters. This result proves the consistent effectiveness of the proposed domain-agnostic processing against both 1D and 2D filters.

Figure 3.18 (left) and Table 3.2 (top) compare the test accuracy of 1D, 2D, and Hilbert 1D-2D filters on MNIST recall task. The results also indicate that Hilbert 1D-2D is superior compared to 1D or 2D filters, with test accuracy 91.2% achieved by Hilbert-based processing, while 2D filter gets 88.1%, and 1D filter attains 87.2%. The 2D filter performs better than 1D in this case as 2D processing is naturally a better fit for the underlying data (images).

Table 3.2: **Test accuracy (%) of 1D, 2D, and Hilbert 1D-2D processing on the MNIST recall and TVQA tasks.**

Task	1D	2D	Hilbert 1D-2D
MNIST Recall	87.2	88.1	**91.2**
TVQA	48.2	45.5	**49.6**

Figure 3.18 (right) and Table 3.2 (bottom) indicate test accuracy of different processing schemes in TVQA task. Again, the results are consistent with the observations made in the cases of Chess and MNIST recall tasks. Specifically, the Hilbert 1D-2D processing achieves the best test accuracy (49.6%) compared to 1D (48.2%) and 2D (45.5%). The 1D filter has advantage over 2D here as most data streams (questions, answers, and subtitles) are in 1D domain, combined with the fact that 1D filter setting has more output channels than 2D filter when the number of parameters are calibrated.

3.5 Conclusion

In this chapter, we introduced a novel domain-agnostic processing scheme based on Hilbert spatial mapping, positional encoding, and multigrid connectivity. The input is transformed to different sub-spaces and simultaneously processed in those spaces, regardless of the input domain. The proposed method demonstrates its effectiveness and generality across different data domains, either 1D, 2D, or a combination thereof, and across different tasks of various natures.

CHAPTER 4

LOCAL OPERATORS

4.1 Introduction

In previous chapters, we have discussed choices in architectural design, as well as methods to enable a universal processing approach regardless of input domains in multigrid neural memory. However, there is one aspect that we have taken for granted, and that is the basic building block of the network: local operators. In Chapter 2, we wrap the network inside multigrid connectivities to allow for the emergence of new capabilities, with the basic building blocks being convolutional LSTMs and feedforward convolutional operations. The local operators are based on standard convolutions, with the interaction of a set of weights (filters) and input tensors through multiplication and addition operators.

While the standard operators in convolutional (and fully connected) networks are sufficiently flexible to allow for the approximation of arbitrary functions, they are not the only possible operations. In fact, the types of fundamental operations in standard neural networks are quite limited, and the space of potential new operators that can be explored is fairly large. Recent advances also show the effectiveness of novel architectures that involve new operators that are not present in the standard convolutional or fully connected networks, such as Transformers [Vaswani et al., 2017] or other variations of attention-based models [Zhao et al., 2020, Woo et al., 2018, Bello et al., 2019]. However, while these works have some elements of new local operators, the basic operators are not evaluated in isolation, but oftentimes are part of a more complex module. Furthermore, each work introduces its own variation, eventhough they utilize similar core local operators. These issues make it hard to isolate and evaluate the effect of some new basic local operators.

In this Chapter, we investigate the effect of new local operators in the context of multigrid neural memory models. In particular, we propose and experiment with different versions of comparison operators. The results demonstrate that, while not all new operators are beneficial, the right operator

does offer consistent advantages over standard modules.

4.2 Related Work

While the original convolutional operation is still the most widely used formulation, there have been efforts in extending it to adapt for specific situations. Yu and Koltun [2016] employ dilated convolution to increase the receptive field of the operations. Particularly, instead of performing convolution on the immediate neighboring pixels, the dilated convolution is applied to pixels further away defined by a dilated factor. On the other hand, Dai et al. [2017] introduces deformable convolution, in which the convolution is applied to an irregular sampling grid instead of the regularly spaced grid in the original and dilated convolutions. Although these works extend the formulation of the original convolution to apply to different sampling grid, the fundamental local operators in the convolution are unchanged.

The Transformer [Vaswani et al., 2017] introduces novel architectures based on attention, which include comparison operators alongside fully connected networks. While containing the new comparison operators, they are not evaluated in isolation, but coupled to specific design of the Transformer architectures. Furthermore, the Transformer is specifically targeted for natually language processing domain. Inspired by the success of the Transformer, other works seek to extend the attention ideas into other domains and architectures. Bello et al. [2019] follow the idea of the Transformer and apply it to the image domain, while Zhao et al. [2020] attempts to devise an attention-based module that operates in local neighborhood similar to convolutional operation. However, the effect of the basic comparison operator is also not evaluated in isolation, but is rather coupled with the authors' added designs as a whole. While the architectural designs of the mentioned works include new local comparison operators, these works are not coined or motivated by the investigation of new local operators, but are instead inspired by the notion of the attention idea, the notion of assigning weights for each tensor to focus on certain elements.

Ha et al. [2016] coins the line of works regarding hyper networks, in which the weights of the main networks (regular networks targeting a specific task) are no longer the learnable parameters,

but rather the output activation tensors generated by the hyper networks. The learnable parameters are now instead shifted to be part of the hyper networks. The comparison operator, in a sense, could be regarded as a very localized hyper network, in which one part of the input acts as a filter being applied on another part of the input tensor to generate the desired output activation tensors.

Motivated by the potential effect of new comparison operators in neural networks, in this work we specifically investigate the behaviors of different basic comparison operators in the context of multigrid neural memory. As opposed to Transformer, the investigated comparison operators are local, not a pairwise comparison between all pairs. Different than other variants, we are interested in investigating the basic comparison operators in isolation to study its effect. The studies specifically investigate how these operators behave in the context of multigrid neural memory, which has not been studied in previous works. Furthermore, we investigate different strategies of designing comparison operators, which are not comprehensively covered in other studies.

4.3 Local Operators

4.3.1 Weight-based versus Comparison Operators

Let us consider the operators in a classic neural network, as demonstrated in Figure 4.1. The right figure shows in detail the basic operators carried out internal to the considered neuron. The neuron maintains a set of learned weights w. The inputs x are first multiplied with the corresponding weights. The resulting products are then summed up before passing through a non-linear activation function to generate the output.

While the above operators could allow the network to adjust its weights to approximate a target function, there is not direct interaction between input tensors through the multiplication operators. The multiplication only operates between an input and a weight variable, while the input tensors only interact with each other through summation operators. Therefore, we propose to investigate the effect of direct interaction of tensors through the multiplication operators.

Figure 4.2 shows examples of comparison operators in neural networks. In contrast to the classic

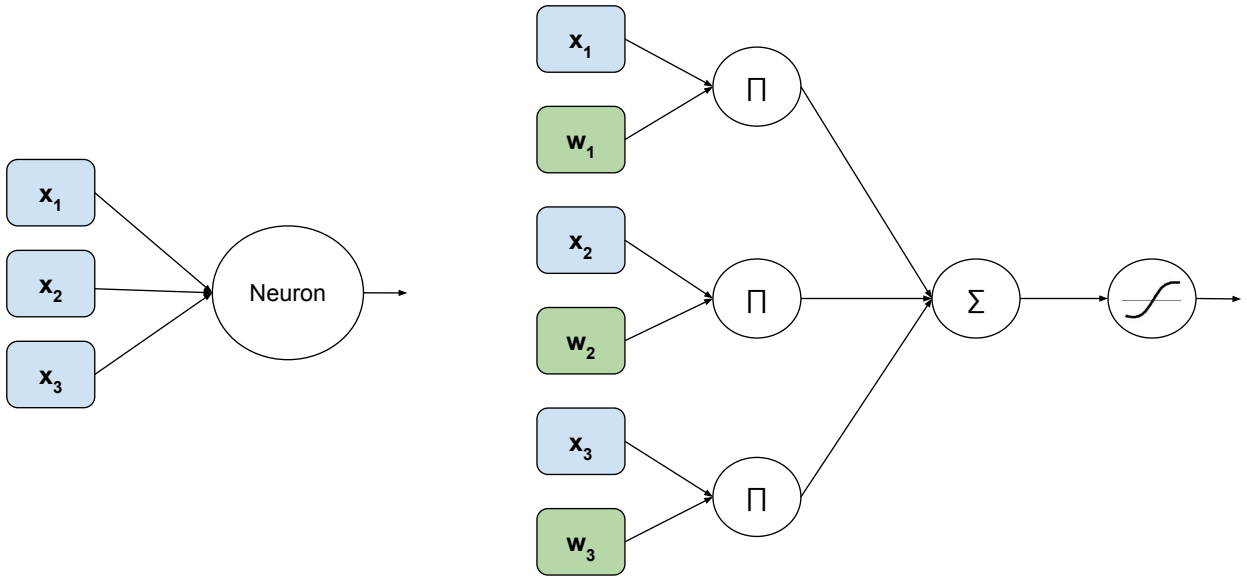


Figure 4.1: **Diagram of operators in a classic neural network. Left:** View of inputs passing through a neuron. **Right:** Expansion of internal operators inside the neuron.

operators in neural network, the comparison operators are based on direct multiplication of input tensors. In Figure 4.2, the left figure shows an example of pairwise comparison of elements in tensor x, while the right figure demonstrates the comparison between two tensors x and y. Notice that the comparison operators themselves do not introduce additional parameters (weights) as in the classic neural networks. While Figure 4.2 shows diagrams with additional operators (summation and non-linear activation) for contrasting with the classic neural network diagram (Figure 4.1), they are just variants on top of the basic pairwise multiplication of tensors. It is this basic multiplication operator on input tensors that is our focus of investigation.

4.3.2 Types of Comparison Operators

We investigate four different comparison operators as demonstrated in Figure 4.3. Each comparison operator is based on the pairwise multiplication of tensors or its derivative (e.g., a dot product). After computing the comparison operators, the results are augmented into the input (by concatenation) before feeding to the next layer of the network. The four settings of comparison operators include the following:

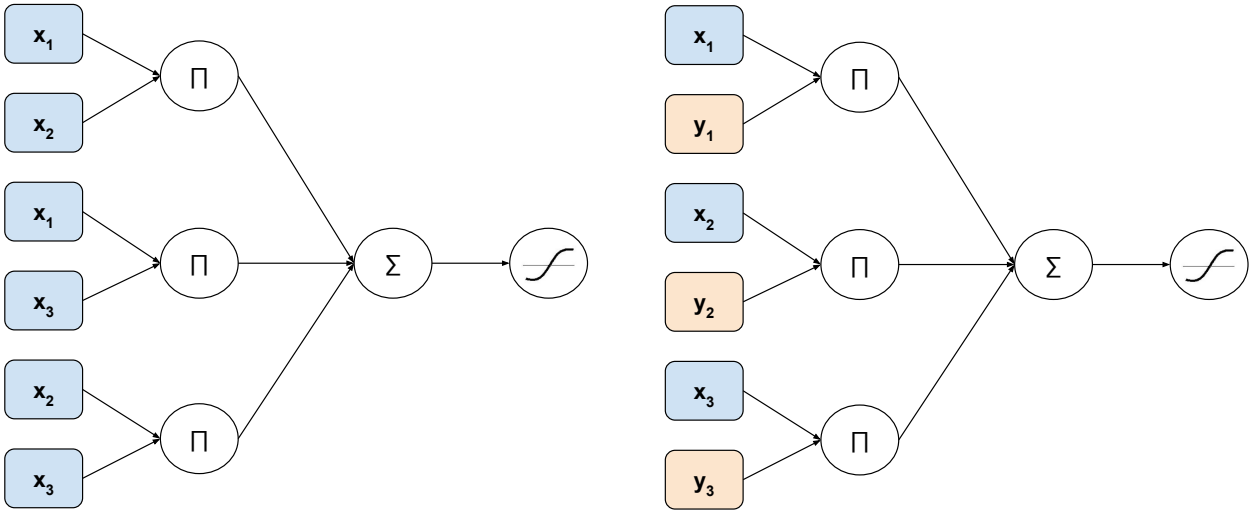


Figure 4.2: **Diagram of comparison operators in neural network. Left:** Pairwise comparison of elements in tensor X. **Right:** Comparison of two tensors X and Y.

- Cross-scale comparison: The comparison operator is applied for each pair of tensors in three neighboring scales of multigrid architectures.
- Split-channel comparison: The tensor is divided into a number of splits along the channel dimension. The splits are then compared against each other in a pairwise manner.
- Spatial comparison: For a tensor, the feature at a spatial location is compared against the features in its immediate neighborhood.
- Self comparison: A tensor is compared against itself.

4.4 Experiments

4.4.1 Effect of Different Types of Comparison Operators

First, we evaluate the effect of different types of comparison operators. Validation accuracy of different operators on the TVQA task are shown in Figure 4.4. Among the four comparison operators, only split-channel comparison demonstrates advantages over the original model. The other comparison configurations (cross-scale comparison, self-comparison, and spatial comparison)

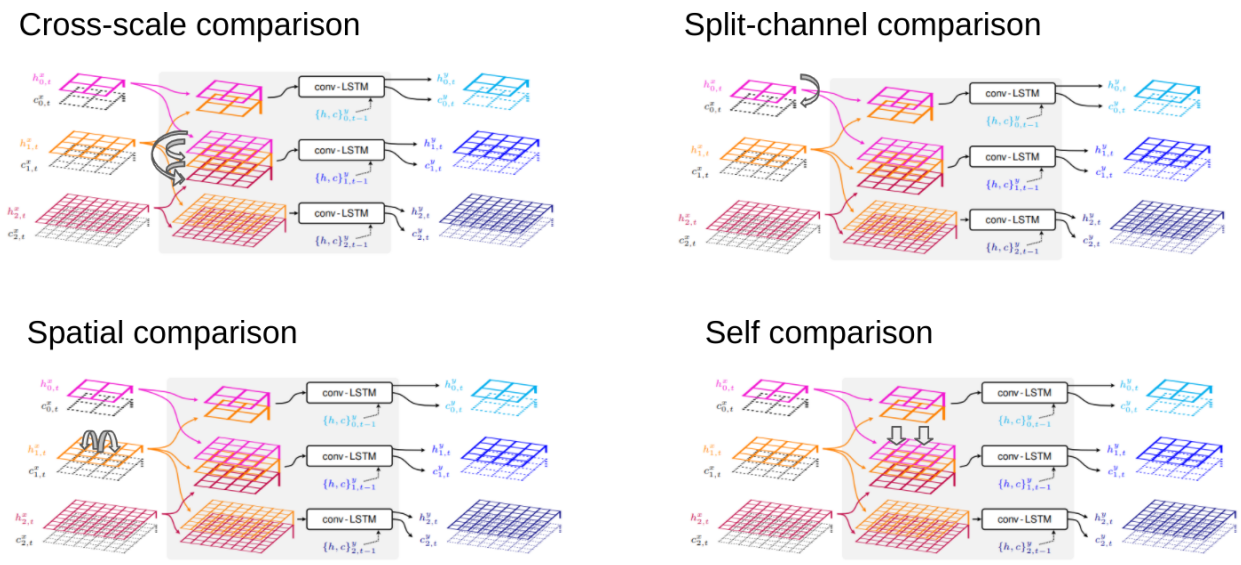


Figure 4.3: **Different comparison operators in multigrid neural architectures.**

do not exhibit beneficial tendency, and could even degrade performance from the original model (without comparison). The order of increasing performance follows as: self-comparison, cross-scale comparison, spatial comparison, no comparison (original), split-channel comparison. This could be because that the channel dimension is most effective for the network to organize information along, as each channel corresponds to a separate filter. Therefore, the network can be effective in adapting the content along the channel dimension to coordinate with, and make good use of the comparison operators.

4.4.2 Placement of Comparison Operator

Next, we investigate the effect of comparison operators on different parts of the network. Figure 4.5 (left) presents the results in various settings in MNIST recall task as follows:

- reader_joint: The comparison operator is placed only at a portion of the reader where it is reading the information from the writer memory.
- reader_full: The comparison operator is placed in all layers of the reader (including the final decoding convolutional layers).

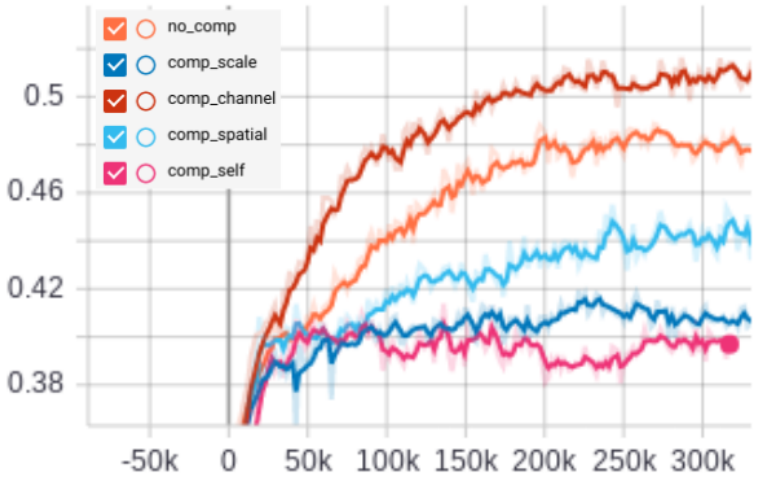


Figure 4.4: **Validation accuracy with different local comparison operators on the TVQA task.**

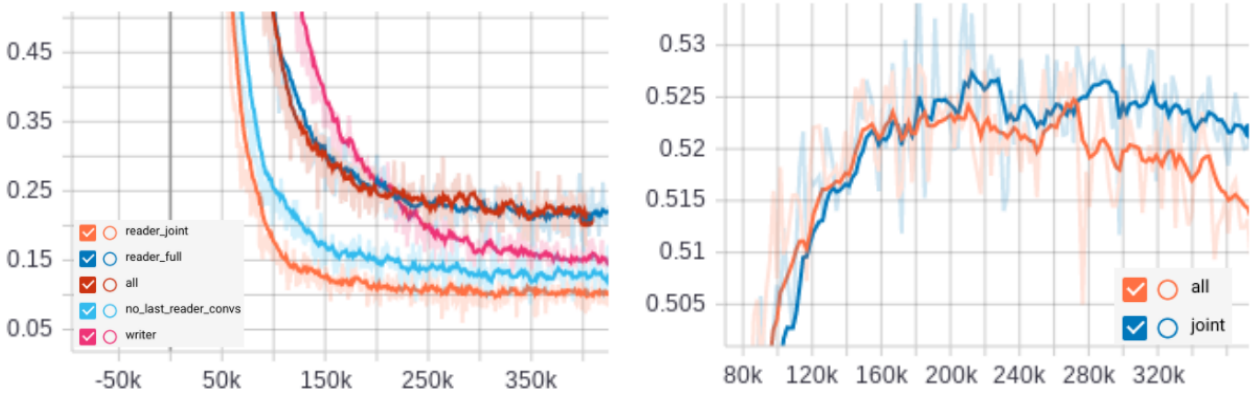


Figure 4.5: **Effect of different placements of the comparison operator. Left:** Validation error rate on the MNIST recall task. **Right:** Validation accuracy on the TVQA task.

- all: The comparison operator is placed every where in the network.
- no_last_reader_convs: The comparison is placed everywhere in the network, except the last decoding convolutional layers in the reader (the layers that do not directly coupled with information from the writer's memory).
- writer: The comparison operator is placed only at the writer portion of the network.

The results suggest that the comparison operator is most effective when dealing with the presence and interaction of multiple sources of information. Particularly, the "reader_joint" setting (The comparison operator is placed at the portion where there is information from both the reader

output and the writer's memory) achieves the best performance, notably better than the worst cases where we extend the comparison to cover the final decoding convolutional layers. It is also worth noting that having the comparison operators covering the writer portion ("writer" and "no_last_reader_convs") has better effect than covering the final decoding convolutional layers in the reader ("reader_full" and "all"). This could be because of the fact that even though the writer does not explicitly have multiple input streams, it has implicit interaction between the input and its internal memory. This result further consolidates the observation that comparison operators are more helpful in cases where there are multiple information streams rather than a single input stream.

To further confirm the above observations made in the MNIST recall task, we perform similar experiments in the TVQA task and investigate the network's behaviors. Figure 4.5 (right) shows the results in this case. Similar to the MNIST recall task, placing the comparison operators only at the portion of the network where there are different streams of data in fact achieves better performance compared to having the comparison operators everywhere. The results suggest that comparing data from different input streams could be essential in enhancing the network's performance.

4.4.3 Number of Splits versus Number of Input Streams

As Section 4.4.2 reveals, it appears that the comparison operators are most effective when dealing with multiple streams of inputs. It is natural to ask whether there is a correlation between the number of input streams and the number of splits in the split-channel comparison? Or which number of splits should be optimal?

Figure 4.6 seeks answers to the above questions by studying the effect of different configurations regarding the number of input streams and the number of channel-wise splits for the TVQA task. From the figure, it is clear that there is a separation in the performance of two classes of configurations: whether the number of splits is the same or different from the number of input streams. The validation accuracy is significantly higher in cases where the number of splits is the same as the number of input streams ("2splits_2streams", "3splits_3streams", and "4splits_4streams" in Figure 4.6). In cases where the number of input streams is different than the number of splits, it

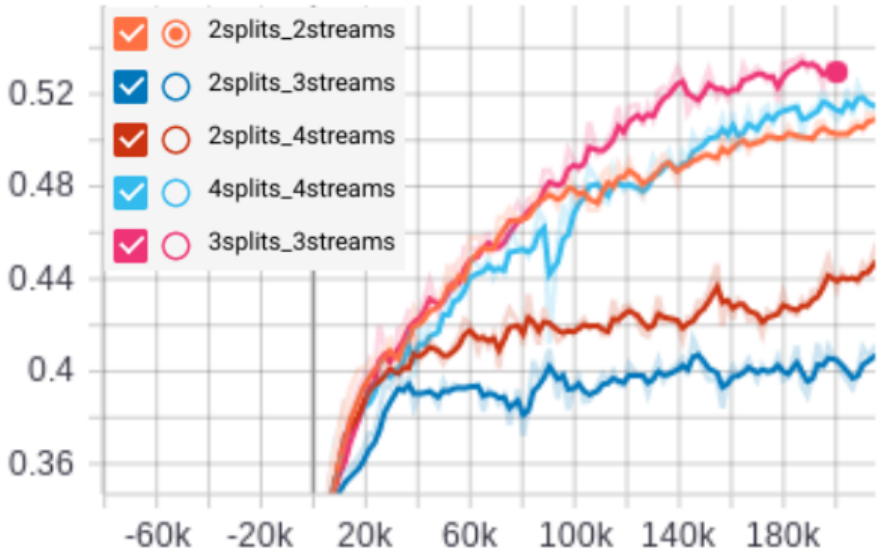


Figure 4.6: **Validation accuracy with different configurations of number of splits and number of data streams on the TVQA task.**

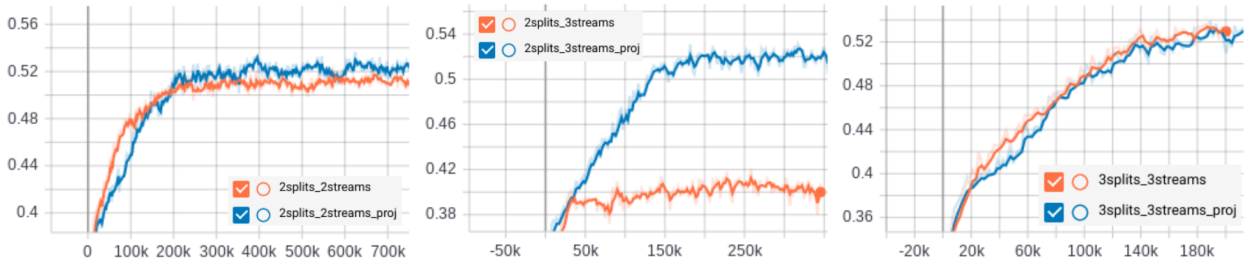


Figure 4.7: **Validation accuracy with and without a projection layer splitting comparison channels on the TVQA task. Left:** 2 splits with 2 input streams. **Middle:** 2 splits with 3 input streams. **Right:** 3 splits with 3 input streams.

appears that having the number of input streams divisible by the number of splits may be preferable ("2splits_4streams" attains strictly better performance than "2splits_3streams" in Figure 4.6).

4.4.4 Effect of The Linear Projection in Channel Splitting

Section 4.4.3 shows that the number of splits should follow the number of data streams for optimal performance. However, the results were obtained by manually splitting the tensor along the channel dimension. In this section, we wonder whether we could lift the restriction regarding the number of splits by allowing the network more freedom to automatically determine how to split the tensor

instead.

Instead of manually splitting the tensor to perform comparison operators, we now inject a linear layer that maps the tensor into different output spaces, and then we perform the comparison operator on those output spaces. This way, the manual explicit splits are replaced by learned mappings instead. Thus, it allows the network to better adapt and organize its feature space in a way that is most effective for the comparison operators.

Figure 4.7 presents the effect of this linear projection layer in various settings regarding the number of splits and number of data streams on the TVQA task. The left and right figures demonstrate the effect in cases where the number of splits are the same as the number of input streams ("2splits_2streams" and "3splits_3streams"). We see that adding the projection layer in this case does not degrade performance, though it only slightly enhances validation accuracy in the "2splits_2streams" setting. However, the middle figure shows very interesting results. Here, we are adding the projection layer on top of the setting where the number of input streams and number of splits are mismatched ("2splits_3streams"). Section 4.4.3 demonstrated that this mismatch results in significant degradation in the network performance. By means of the projection layer instead of manual splitting, we significantly boosts the performance ("2splits_3streams_proj") despite still having a mismatch in the number of splits versus number of data streams. The result proves the effectiveness of the projection layer and the learned splits compared to the manual splits. With this added projection, we are able to break the symmetry requirement between the number of splits and number of data streams. It in turn allows the network to be more generic and flexible, where the model can automatically adapt its feature to best fit the number of splits provided.

4.5 Conclusion

In this chapter, we investigate different new comparison operators as well as their specific configurations in the context of multigrid memory architectures. The results show that the split-channel comparison operator is especially promising, particularly in cases where there are multiple data streams. The study reveals the benefits of novel local operators in architectural designs, and lays

the foundation for future exploration of more novel operators that are not captured in the original multigrid architectures, or in a wider context, convolutional neural networks.

CHAPTER 5

CONCLUSION

In this thesis, we propose a novel neural memory model, Multigrid Neural Memory, and investigate in detail three aspects in building the system: architectural design, domain-agnostic processing, and local operators.

The proposed Multigrid Neural Memory model demonstrates its generic and superior performance in a wide range of tasks. It represents a drastically new way to maintain long-term, large-scale memory, laying the foundation for further advancements in the neural memory space, which is critical in enabling artificial general intelligence.

The proposed domain-agnostic processing scheme shows the potential of having a unified approach to neural memory, in which a generic model could be capable of handling tasks of arbitrary domains.

Finally, the investigated comparison operators demonstrate the potential and effectiveness of new local operators, opening up possibilities for future investigation into other new local operators beyond the current operators in contemporary models.

Together, the three investigated dimensions provide a rather complete picture into the study and effort to develop a universal neural memory model. Besides the introduction of a powerful neural memory model, the presented results reveal valuable insights and lay the foundation for future investigations in this space.

REFERENCES

Monowar Anjum, Ibrahim Tahmid, and M. Rahman. CHilEnPred: CNN model with Hilbert curve representation of DNA sequence for enhancer prediction. 02 2019. doi: 10.1101/552141.

Stanislaw Antol, Aishwarya Agrawal, Jiasen Lu, Margaret Mitchell, Dhruv Batra, C. Lawrence Zitnick, and Devi Parikh. VQA: Visual Question Answering. In *International Conference on Computer Vision (ICCV)*, 2015.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv:1409.0473*, 2014.

Irwan Bello, Barret Zoph, Quoc Le, Ashish Vaswani, and Jonathon Shlens. Attention augmented convolutional networks. In *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 3285–3294, 2019. doi: 10.1109/ICCV.2019.00338.

Anastasia Borovykh, Cornelis W. Oosterlee, and Sander M. Bohté. Generalization in fully-connected neural networks for time series forecasting. *Journal of Computational Science*, 36:101020, 2019. ISSN 1877-7503. doi: https://doi.org/10.1016/j.jocs.2019.07.007. URL `http://www.sciencedirect.com/science/article/pii/S1877750319301838`.

Jasmine Collins, Jascha Sohl-Dickstein, and David Sussillo. Capacity and trainability in recurrent neural networks. *ICLR*, 2017.

Andrew R.A. Conway, Michael J. Kane, and Randall W. Engle. Working memory capacity and its relation to general intelligence. *Trends in Cognitive Sciences*, 7(12):547 – 552, 2003. ISSN 1364-6613. doi: https://doi.org/10.1016/j.tics.2003.10.005. URL `http://www.sciencedirect.com/science/article/pii/S1364661303002833`.

Thomas Corcoran, Rafael Zamora-Resendiz, Xinlian Liu, and Silvia Crivelli. A spatial mapping algorithm with applications in deep learning-based structure classification. *arXiv:1802.02532*, 2018.

Jifeng Dai, Haozhi Qi, Yuwen Xiong, Yi Li, Guodong Zhang, Han Hu, and Yichen Wei. Deformable convolutional networks. In *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 764–773, 2017. doi: 10.1109/ICCV.2017.89.

Sreerupa Das, C. Lee Giles, and Guo-Zheng Sun. Learning context-free grammars: Capabilities and limitations of a recurrent neural network with an external stack memory. In *CogSci*, 1992.

Sreerupa Das, C. Lee Giles, and Guo-Zheng Sun. Using prior knowledge in an NNPDA to learn context-free languages. In *NIPS*, 1993.

Yann N. Dauphin, Angela Fan, Michael Auli, and David Grangier. Language modeling with gated convolutional networks. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ICML'17, page 933–941. JMLR.org, 2017.

Jeff Donahue, Lisa Anne Hendricks, Marcus Rohrbach, Subhashini Venugopalan, Sergio Guadarrama, Kate Saenko, and Trevor Darrell. Long-term recurrent convolutional networks for visual recognition and description. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2017.

Marco Fraccaro, Danilo Jimenez Rezende, Yori Zwols, Alexander Pritzel, S. M. Ali Eslami, and Fabio Viola. Generative temporal models with spatial memory for partially observed environments. *ICML*, 2018.

Mevlana Gemici, Chia-Chun Hung, Adam Santoro, Greg Wayne, Shakir Mohamed, Danilo J Rezende, David Amos, and Timothy Lillicrap. Generative temporal models with memory. *arXiv:1702.04649*, 2017.

Alex Graves. Generating sequences with recurrent neural networks. *arXiv:1308.0850*, 2013.

Alex Graves, Abdel rahman Mohamed, and Geoffrey Hinton. Speech recognition with deep recurrent neural networks. In *IEEE International Conference on Acoustics, Speech and Signal Processing*, 2013.

Alex Graves, Greg Wayne, and Ivo Danihelka. Neural turing machines. *arXiv:1410.5401*, 2014.

Alex Graves, Greg Wayne, Malcolm Reynolds, Tim Harley, Ivo Danihelka, Agnieszka Grabska-Barwińska, Sergio Gómez Colmenarejo, Edward Grefenstette, Tiago Ramalho, John Agapiou, Adrià Puigdomènech Badia, Karl Moritz Hermann, Yori Zwols, Georg Ostrovski, Adam Cain, Helen King, Christopher Summerfield, Phil Blunsom, Koray Kavukcuoglu, and Demis Hassabis. Hybrid computing using a neural network with dynamic external memory. *Nature*, 2016.

Edward Grefenstette, Karl Moritz Hermann, Mustafa Suleyman, and Phil Blunsom. Learning to transduce with unbounded memory. In *NIPS*, 2015.

David Ha, Andrew Dai, and Quoc Le. Hypernetworks. In *Fifth International Conference on Learning Representations (ICLR 2017)*, 2016.

Matthew Hausknecht and Peter Stone. Deep recurrent Q-learning for partially observable MDPs. In *AAAI Fall Symposium*, 2015.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CVPR*, 2016.

Michiel Hermans and Benjamin Schrauwen. Training and analysing deep recurrent neural networks. *NIPS*, 2013.

Sepp Hochreiter and Jurgen Schmidhuber. Long short-term memory. *Neural Computation*, 1997.

Steffen Hölldobler, Yvonne Kalinke, and Helko Lehmann. Designing a counter: Another case study of dynamics and activation landscapes in recurrent networks. In *Annual Conference on Artificial Intelligence*, 1997.

Yunseok Jang, Yale Song, Youngjae Yu, Youngjin Kim, and Gunhee Kim. TGIF-QA: Toward spatio-temporal reasoning in visual question answering. pages 1359–1367, 07 2017. doi: 10.1109/CVPR.2017.149.

Armand Joulin and Tomas Mikolov. Inferring algorithmic patterns with stack-augmented recurrent nets. In *NIPS*, 2015.

Nal Kalchbrenner, Ivo Danihelka, and Alex Graves. Grid long short-term memory. *arXiv:1507.01526*, 2015.

Tsung-Wei Ke, Michael Maire, and Stella X. Yu. Multigrid neural architectures. *CVPR*, 2017.

Iasonas Kokkinos. Ubernet: Training a universal convolutional neural network for low-, mid-, and high-level vision using diverse datasets and limited memory. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5454–5463, 2017. doi: 10.1109/CVPR.2017.579.

Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical report, 2009.

Karol Kurach, Marcin Andrychowicz, and Ilya Sutskever. Neural random-access machines. *arXiv:1511.06392*, 2015.

Yann Lecun, Leon Bottou, Yoshuo Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 1998.

Yann LeCun, Koray Kavukcuoglu, and Clément Farabet. Convolutional networks and applications in vision. *ISCAS*, 2010.

Jie Lei, Licheng Yu, Mohit Bansal, and Tamara L Berg. Tvqa: Localized, compositional video question answering. In *EMNLP*, 2018.

Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. End-to-end training of deep visuomotor policies. *The Journal of Machine Learning Research*, 2016.

Shuai Li, Wanqing Li, Chris Cook, Ce Zhu, and Yanbo Gao. Independently recurrent neural network (indrnn): Building a longer and deeper rnn. *CVPR*, 2018.

J. Liu, Wenhu Chen, Y. Cheng, Zhe Gan, Licheng Yu, Yiming Yang, and Jing jing Liu. Violin: A large-scale dataset for video-and-language inference. *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 10897–10907, 2020.

Ben Mildenhall, Pratul P. Srinivasan, Matthew Tancik, Jonathan T. Barron, Ravi Ramamoorthi, and Ren Ng. Nerf: Representing scenes as neural radiance fields for view synthesis. In *ECCV*, 2020.

Volodymyr Mnih, Nicolas Heess, Alex Graves, et al. Recurrent models of visual attention. In *NIPS*, 2014.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 2015.

Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. *ICML*, 2016.

Michael C. Mozer and Sreerupa Das. A connectionist symbol manipulator that discovers the structure of context-free languages. In *NIPS*, 1993.

David Noever, Matt Ciolino, and Josh Kalin. The chess transformer: Mastering play using generative language models, 2020.

Junhyuk Oh, Valliappa Chockalingam, Satinder Singh, and Honglak Lee. Control of memory, active perception, and action in Minecraft. *arXiv:1605.09128*, 2016.

Emilio Parisotto and Ruslan Salakhutdinov. Neural map: Structured memory for deep reinforcement learning. *ICLR*, 2018.

Alexander Pritzel, Benigno Uria, Sriram Srinivasan, Adria Puigdomenech, Oriol Vinyals, Demis Hassabis, Daan Wierstra, and Charles Blundell. Neural episodic control. *ICML*, 2017.

Scott Reed and Nando de Freitas. Neural programmer-interpreters. *arXiv:1511.06279*, 2015.

Adam Santoro, Sergey Bartunov, Matthew Botvinick, Daan Wierstra, and Timothy Lillicrap. One-shot learning with memory-augmented neural networks. *arXiv:1605.06065*, 2016.

Jürgen Schmidhuber. Learning to control fast-weight memories: An alternative to dynamic recurrent networks. *Neural Computation*, 1992.

Jürgen Schmidhuber. A 'self-referential'weight matrix. In *ICANN*, 1993.

Hava T. Siegelmann and Eduardo D. Sontag. On the computational power of neural nets. *Journal of computer and system sciences*, 1995.

Sainbayar Sukhbaatar, Jason Weston, Rob Fergus, et al. End-to-end memory networks. In *NIPS*, 2015.

Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In *NIPS*, 2014.

Shubham Toshniwal, Sam Wiseman, Karen Livescu, and Kevin Gimpel. Learning Chess Blindfolded: Evaluating Language Models on State Tracking. 2021.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *NeurIPS*, 2017.

Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. Pointer networks. In *NIPS*, 2015.

Jason Weston, Antoine Bordes, Sumit Chopra, Alexander M. Rush, Bart van Merriënboer, Armand Joulin, and Tomas Mikolov. Towards AI complete question answering: A set of prerequisite toy tasks. *arXiv:1502.05698*, 2015a.

Jason Weston, Sumit Chopra, and Antoine Bordes. Memory networks. *ICLR*, 2015b.

Wikipedia. Hilbert curve. https://en.wikipedia.org/wiki/Hilbert_curve, a.

Wikipedia. Space-filling curve. https://en.wikipedia.org/wiki/Space-filling_curve, b.

Sanghyun Woo, Jongchan Park, Joon-Young Lee, and In So Kweon. Cbam: Convolutional block attention module. In Vittorio Ferrari, Martial Hebert, Cristian Sminchisescu, and Yair Weiss, editors, *Computer Vision – ECCV 2018*, pages 3–19, Cham, 2018. Springer International Publishing. ISBN 978-3-030-01234-2.

SHI Xingjian, Zhourong Chen, Hao Wang, Dit-Yan Yeung, Wai-Kin Wong, and Wang-chun Woo. Convolutional LSTM network: A machine learning approach for precipitation nowcasting. *NIPS*, 2015.

Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhutdinov, Richard Zemel, and Yoshua Bengio. Show, attend and tell: Neural image caption generation with visual attention. *arXiv:1502.03044*, 2015.

Bojian Yin, Marleen Balvert, Davide Zambrano, Alexander Schönhuth, and Sander Bohte. An image representation based convolutional network for dna classification. *ICLR*, 2018.

Fisher Yu and Vladlen Koltun. Multi-scale context aggregation by dilated convolutions. *ICLR*, 2016.

Zheng Zeng, Rodney M Goodman, and Padhraic Smyth. Discrete recurrent neural networks for grammatical inference. *IEEE Transactions on Neural Networks*, 1994.

Hengshuang Zhao, Jiaya Jia, and Vladlen Koltun. Exploring self-attention for image recognition. In *CVPR*, 2020.