7-17-2018

# Holographic Generative Memory: Neurally Inspired One-Shot Learning with Memory Augmented Neural Networks

Dillon R. Graham
drg7604@rit.edu

# Holographic Generative Memory: Neurally Inspired One-Shot Learning with Memory Augmented Neural Networks

Dillon R. Graham

# Holographic Generative Memory: Neurally Inspired One-Shot Learning with Memory Augmented Neural Networks

Dillon R. Graham

July 17th, 2018

A Thesis Submitted
in Partial Fulfillment
of the Requirements for the Degree of
Master of Science
in
Computer Engineering

R·I·T | Kate Gleason
*College of* ENGINEERING

*Department of Computer Engineering*

# Holographic Generative Memory: Neurally Inspired One-Shot Learning with Memory Augmented Neural Networks

Dillon R. Graham

**Committee Approval:**

---

Dr. Dhireesha Kudithipudi *Advisor* Date
Professor

---

Dr. Ray Ptucha Date
Assistant Professor

---

Dr. Christopher Kanan Date
Assistant Professor

# Acknowledgments

Thank you to my advisor, Dr. Dhireesha Kudithipudi. You showed incredible patience and grace in supporting my sometimes unconventional ideas and thought process. I am genuinely grateful for your guidance and the freedom you allowed me to pursue my work. Many thanks to my committee members Dr. Christopher Kanan and Dr. Ray Ptucha. Your suggestions on presenting my work were very constructive and helped me consider it from an entirely new perspective. Thank you, Dr. Kanan, for indulging my frequent conversations about AI.

I would also like to thank Sandia National Laboratories and the folks at the Critical Skills Master's Program for making my graduate education possible. Thank you to John Mareda for providing me with this opportunity, enjoy your retirement!

Thank you to all my labmates in the Nu.AI Lab. I enjoyed our many conversations and appreciated your tolerance of my lengthy monologues on chairs as compositional semantic objects. To Dr. Ernest Fokoue, who helped me through my many mathematical questions, thank you for graciously providing your time to assist me. Your enthusiasm for statistics and compassion for students is much appreciated.

Thank you to Dr. Tom Caudell for teaching me both biological and artificial neural networks. Without your instruction, I wouldn't be working in this field. Thank you to Dr. Rich Compeau for supporting a mediocre circuits student with some big ideas. A special thank you to Dr. Murat Okandan for hosting a brown-bag lunch that got me started on this path, and his ongoing efforts to support neural-inspired computing.

My wife, Jessica has been a constant source of encouragement and support. She has been incredibly understanding while I've more or less lived in my office for the last two years. Thank you for being awesome and reminding me to take care of the little things like eating and sleeping. Lastly, thank you to my wonderful children Leila and Brian. Thank you for understanding the many long days and nights I had to spend studying. You are my constant source of hope and inspiration.

*I would like to dedicate this thesis to my Dad. He encouraged me to learn everything, from art to engineering. Thank you, Dad. You are the best man I know.*

# Abstract

Humans quickly parse and categorize stimuli by combining perceptual information and previously learned knowledge. We are capable of learning new information quickly with only a few observations, and sometimes even a single observation. This one-shot learning (OSL) capability is still very difficult to realize in machine learning models. Novelty is commonly thought to be the primary driver for OSL. However, neuroscience literature shows that biological OSL mechanisms are guided by uncertainty, rather than novelty, motivating us to explore this idea for machine learning.

In this work, we investigate OSL for neural networks using more robust compositional knowledge representations and a biologically inspired uncertainty mechanism to modulate the rate of learning. We introduce several new neural network models that combine Holographic Reduced Representation (HRR) and Variational Autoencoders. Extending these new models culminates in the Holographic Generative Memory (HGMEM) model.

HGMEM is a novel unsupervised memory augmented neural network. It offers solutions to many of the practical drawbacks associated with HRRs while also providing storage, recall, and generation of latent compositional knowledge representations. Uncertainty is measured as a native part of HGMEM operation by applying trained probabilistic dropout to fully-connected layers. During training, the learning rate is modulated using these uncertainty measurements in a manner inspired by our motivating neuroscience mechanism for OSL. Model performance is demonstrated on several image datasets with experiments that reflect our theoretical approach.

# Contents

# List of Figures

# List of Tables

# Chapter 1

## Introduction

Humans are capable of quickly parsing and categorizing stimuli. We accomplish this by combining perceptual information and knowledge previously internalized through learning. In the vision domain alone, humans can easily distinguish between tens of thousands of object classes, often with minimal attention. Our brains learn new classes of objects with ease and are even capable of imagining new classes based on existing knowledge.

We learn incrementally by default, where knowledge is gained slowly over time, usually with trial and error acting as our instructor. However, in some circumstances, we demonstrate a sudden ability to learn new information very quickly, even from a single observation. This one-shot learning (OSL) capability present in humans is still something we find very difficult to implement in machine learning.

Computer vision tasks like image classification or object detection have often been used to study OSL. This is a natural choice since images offer a readily interpretable source of data. Reducing training requirements is also attractive in a field where current high-performing Machine Learning (ML) systems require thousands or millions of training samples [3] to converge.

Minimal data tends to cause overfitting in ML models. Overfitting can sometimes be alleviated by techniques like data augmentation and regularization, but learning is still slow since numerous weight updates are required with gradient descent train-

ing. Additionally, neural networks often suffer from the classic *catastrophic forgetting* problem due to parameters being shared for all tasks and data domains. Rapidly learning from new data samples while retaining the ability to generalize from common samples (i.e., OSL) remains a difficult problem in ML, and it is the focus of an expanding area of research.

Tasks with few labeled samples or imbalanced data have traditionally been difficult to approach with ML, but many valuable real-world tasks take this form. For instance, in medical imaging there are often many samples from a large population, but relatively few positive labeled examples available, leading to an imbalanced data problem. Fraud detection or other financial applications offer similar challenges, a large amount of samples, but very few examples of the target class. Aside from practical applications, OSL is likely to be a core capability for future advancements in AI. It is difficult to imagine systems that can perform in areas like lifelong learning or abstract reasoning without the ability to learn in this way. Future AI systems for complex tasks like these will need a way to leverage past knowledge for rapid learning without losing previously learned knowledge in the process.

Following the success of deep learning, much of the contemporary study on OSL for ML has focused on vision tasks. Li et al. examined using probabilistic models to learn object categories from a limited number of images by taking advantage of existing knowledge of similar objects [4]. Li further explores this topic in other works with an additional focus on transfer learning by grouping existing techniques into transfer through prior parameters, transfer through shared features or parts, and transfer through contextual information [5]. Work on OSL limited only to visual information is useful for understanding shared knowledge of visually perceived objects, but it does not necessarily provide an exhaustive basis for approaching the OSL problem in general. These earlier approaches also primarily rely on parametric models, which carry the speed and knowledge retention drawbacks that come with shared parameters.

Non-parametric models offer a method to avoid catastrophic forgetting while also rapidly making use of novel data. In some cases, only a metric is chosen, and training is completely avoided [6]. An extension of this approach is using a learnable metric [7], which provides some additional flexibility. More recently, models that combine parametric and non-parametric techniques have emerged that make use of attention mechanisms and memory augmentation [8, 9].

Neural networks augmented with memory components provide one potential path toward effectively using previously learned knowledge, but the implementation of that memory can vary between methods. Much of the recent interest in this area originates from the Neural Turing Machine (NTM) [10], where explicit read and write operations are defined in a manner inspired by standard computer memory. Subsequent models building off NTMs focus on accessing memory via content rather than location [11] or using Gaussian approaches for the memory component [12, 13].

In the 1990s and early 2000s several connectionist data representation architectures were developed that offer alternative ways to represent learned knowledge. They were intended to help model features of human cognition like composing complex representations by combining simpler representations. For example, we create a sentence by combining simpler items (words), which are in-turn composed of even simpler items (letters). Each item is interpretable individually, but they also form a new interpretable item in composition. Psychologist Ross W. Gayler [14] coined the term Vector Symbolic Architecture (VSA) to describe this class of connectionist network.

Different VSA approaches offer various levels of compatibility with modern neural networks. In VSAs, some variant of Smolensky's Tensor Product (TP) [15] is typically used for associating representations. Unfortunately, this can lead to a combinatorial explosion problem, since binding two $N$-dimensional vectors with the TP results in an $N \times N$ square matrix, with dimensionality continuing to increase as more bindings are added. Many VSAs based on Kanerva's sparse distributed memory work [16] use

binary valued vectors with various techniques to avoid this dimensionality problem. However, binary vectors can limit neural network compatibility for these VSAs.

Holographic Reduced Representation (HRR) [17] is another type of VSA that offers a framework for distributed representation using real-number valued vectors rather than binary. In HRRs a lossy compression of the full TP is used to perform association operations. This approach maintains fixed-width vector representations at the expense of added noise. Since HRR vectors are real-valued and do not increase in dimensionality they are directly compatible with modern neural network architectures.

One of the recurring themes in OSL literature is learning about a new object class by building on the knowledge of previously learned objects. With deep learning this often takes the form of using pre-trained networks to extract features (e.g., transfer learning). Using a network that was pre-trained on a large dataset allows us to gather more useful features from a smaller but related dataset (e.g., both sets are natural images). We are building on the pre-existing knowledge in the network to compose representations of new objects, but only in a general sense. The network does not make use of abstract reasoning, analogy, or any of the other tools typically associated with human memory. It only has a set of parameters that have been successfully biased toward a certain type of data via lengthy training.

When training via gradient descent, we are essentially storing knowledge in a distributed manner across network parameters. While this type of representation is obviously effective for many tasks, alternative forms of representation can provide more explicit access to knowledge content. If we consider OSL as a knowledge-based problem, looking at alternative ways to represent knowledge in ML systems is worthwhile. Interpretable and composable memories are a core feature of human knowledge representation, but these features are simply difficult to implement with knowledge distributed across network weights.

Framing the OSL problem as a knowledge-based one can also be illustrative. In a very simplistic view, a learning system must have some way to represent knowledge that distills and compresses useful features for re-application in other tasks. Transfer learning does provide some capacity in this regard, but current techniques tend to focus only on extracting better features rather than exploring how those features can be built into more robust knowledge representations. Memory augmented networks move us closer to better knowledge representations via trainable memory components, but memory content is often uninterpretable outside of the network.

VSAs provide representational advantages that are difficult to reproduce with standard neural networks, like encoding structure and information simultaneously in a single vector, holistic processing, and concept encoding. Despite these advantages, VSAs have remained a niche research topic. We suspect that this may be due to their origin in more niche cognitive science style applications common in the 1990s, but less common in the era of deep learning. Additionally, they can be difficult to work with from a practical standpoint, sometimes requiring extensive programmatic bookkeeping to track vectors and associations. Consequently, integrating VSAs with modern neural network architectures has been an infrequently explored topic.

Aside from knowledge representation concerns for OSL, some attention should also be paid to the mechanisms that assist with learning in a low data setting. Machine learning research typically focuses on the idea that input novelty is the primary driver for OSL. However, in the study of neurobiology and human behavior, [1] found that causal uncertainty plays a more significant role in OSL. Specifically, when humans observe a high degree of uncertainty present in cause and consequence relationships, we switch from incremental learning mode to OSL mode to help resolve that uncertainty. The authors make a critical observation in this study, that may prove useful in machine learning. They show that novelty detection is not the primary driver for this learning modulation mechanism, and even back this empirically using ablation

experiments that separate the impact of novelty and uncertainty. This observation suggests that we may be overlooking a crucial OSL component by focusing on the novelty-based mechanisms dominant in ML, and not considering uncertainty.

The mechanisms for OSL in our primary exemplary model (the brain) are still only minimally understood, so it is not surprising that translating this capability into machines is particularly tricky. One reasonable route for mitigating this difficulty is using neuroscience as inspiration for improving our machine learning models. These two fields rarely overlap outside of attempts to model brain function computationally, but we believe that some cross-domain knowledge of this sort can provide benefits when developing ML models. Using biology for inspiration allows us to leverage insights from both fields, and potentially notice conceptual correspondences that would have been missed otherwise.

Examining OSL from this perspective suggests that current ML approaches lack some fundamental characteristics that contribute to this feature in biology. Developing a way to implement these characteristics for ML models provides an alternative approach for OSL. Encoding learned knowledge with a representation that supports intrepretability and compasability provides tools that a model may leverage when exposed to new data. Using uncertainty measurements to modulate learning allows a model to react quickly to data based on confidence in prediction, rather than just novelty of an input. Implementing these traits in ML models increases their effectiveness in low or imbalanced data settings. In this work, we introduce a neurally inspired OSL mechanism for modulating learning rate in neural networks and a new approach to knowledge representation using learned HRR components.

Our OSL mechanism is driven by predictive uncertainty measured as a normal part of network operation using a trained probabilistic dropout technique known as Concrete Dropout [18]. Uncertainty is used to scale learning rate in a manner inspired by the neuroscience model of the problem, allowing the network to dynamically ramp

up learning rate when uncertainty is high and decrease it as uncertainty is resolved.

We develop new methods for neural network knowledge representation that implement a trainable "full-stack" HRR system. This Holographic Generative Memory (HGMEM) model is capable of learning all necessary HRR components in an unsupervised manner through a generative memory model designed to learn, store, and recall data representations. Developing HGMEM required several novel components that can be used as stand-alone models, which we also present here. We demonstrate model performance on several image datasets with experiments that reflect our theoretical approach.

# Chapter 2

## Background

### 2.1 Data Representations

Data representation has remained a subject of interest since the earliest days of computational research. Symbolic representation was a natural choice for translating mathematical expressions into computer programs. Symbolic processing [19] remained the dominant approach for AI until the 1980s when the rise of connectionist neural networks became a source of contention in the research community.

Over time, advances in neural networks and ML shifted focus to vector or tensor representations rather than symbols for many applications. Today, the use of *local representations* and *distributed representations* are a central part of modern AI, but the reason for their use is often overlooked. Examining the properties of data representation helps inform us better understand how interpretable and composable representations can be constructed. Ferrone et al. explore the differences between symbolic, distributed, and distributional representations at length [20], and we adapt some of their examples in this section to help explain representation properties of interest.

Many problems we encounter in AI involve a set of entities to represent and a network of simple computational elements (e.g., neurons). The most straight-forward representation to use is a simple mapping of each entity to an element with a one-

to-one relationship, otherwise known as a local representation. This is common in one-hot binary vectors we often use in classification, where element position maps directly to some concept (e.g. a class label, a word).

For an example of a local representation, consider a set of symbols $\mathcal{D}$, where the $i$-th unit vector represents the $i$-th symbol. In this example, $\mathcal{D}$ consists of:

$$\begin{aligned}
\text{man} &\rightarrow \mathbf{e}_1(1\,0\,0\,0\,0\,0\,0)^T \\
\text{woman} &\rightarrow \mathbf{e}_2(0\,1\,0\,0\,0\,0\,0)^T \\
\text{train} &\rightarrow \mathbf{e}_2(0\,0\,1\,0\,0\,0\,0)^T \\
\text{waits} &\rightarrow \mathbf{e}_3(0\,0\,0\,1\,0\,0\,0)^T \\
\text{for} &\rightarrow \mathbf{e}_4(0\,0\,0\,0\,1\,0\,0)^T \\
\text{a} &\rightarrow \mathbf{e}_5(0\,0\,0\,0\,0\,1\,0)^T
\end{aligned}$$

Typically, in local representations a sequence of symbols, $\mathbf{s}$ is created using a sequence of vectors, or a bag-of-symbols. Consider the sequence, "a man waits for a train". Using a sequence of vectors to construct $\mathbf{s}$ from $\mathcal{D}$ can be done as follows:

$$\text{a man waits for a train} \rightarrow \mathbf{s} = \begin{pmatrix}
0\,0\,0\,0\,0\,1\,0 \\
1\,0\,0\,0\,0\,0\,0 \\
0\,0\,0\,1\,0\,0\,0 \\
0\,0\,0\,0\,1\,0\,0 \\
0\,0\,0\,0\,0\,1\,0 \\
0\,0\,1\,0\,0\,0\,0 \\
0\,0\,0\,0\,0\,0\,0
\end{pmatrix}$$

Constructing the same sequence with a bag-of-symbols approach can be done using the weighted sum of symbol vectors to create a single vector, as follows:

$$\text{a man waits for a train} \rightarrow \mathbf{s} = \begin{pmatrix} 1 \\ 0 \\ 1 \\ 1 \\ 1 \\ 2 \\ 0 \end{pmatrix}$$

Hinton et al. proposed the idea of distributed representations in 1986 [21], and this idea helped fuel the success of modern neural networks and deep learning. In distributed representations, a concept is represented using **multiple** units (e.g., neurons), with the value and position of each unit contributing. Each unit can also contribute to the representation of multiple concepts.

We use distributed representations often in neural networks and deep learning. In these networks, we store learned information as a distributed representation within vectors or tensors of real number values (e.g., weight parameters). While distributed representations of this type are now conventional, they are only one way to approach the problem. Other related, but less explored techniques provide more explicit control in representation building (e.g., HRRs).

In a general sense, the goal is to represent a complex item using many other items combined with some set of rules or operations. Plate [22] provided an analogy to help explain this concept using grayscale images, which we paraphrase here as an example.

Consider a grayscale image, where each unit (i.e., pixel) is a scalar value from 0 to 255. All grayscale images use this same set of units, and each unit is interpretable as a pixel with some shade of gray. Given this setup, knowing the value of individual pixels provides little or no information about objects in an image. However, composing these elements by distributing them in a specific way forms an image we can interpret. We are using a set of simple, interpretable elements and some compositional rules (e.g., pixel index position) to build more complex representations.

Definitions for composable and interpretable can vary depending on the type of representation used, so it is worth commenting on these properties more directly. These concepts have their roots in symbolic representation, which has a long history in computer science. Symbols are also a crucial part of how human beings think and communicate.

A composable representation is one that can be combined to form more complex representations using a set of strong combinatory rules. Components in a compositional representation remain individually interpretable (e.g., pixels in an image). Multiple compositions can also be combined to form new representations (e.g., two images superimposed).

An interpretable representation is one that allows us to understand or read meaning from it. Often this understanding is direct, like viewing an image or word. With distributed representations, this property also refers to directly decoding meaning using the basic combinatory rules defined by the framework. Distributed representations used in deep learning are generally considered uninterpretable from this standpoint. This limitation leads to the familiar "black-box" criticism since we cannot interpret the representation in the hidden layers by definition.

These two properties are essential for human communication. We use many symbols during a conversation. For example, sounds are composed into words, and words into sentences, letting us effectively communicate complex ideas with a hierarchy of compositional representations. Conversations only work because we compose symbols using combinatory rules understood by both the speaker and listener [23]. This strong relation between symbolic representation and language helps explain why researchers in natural language processing (NLP) still often rely on this method while it has become less prevalent in other domains.

Neural networks with composable and interpretable distributed representations may lead to novel and substantially different deep learning models, but this potential

has seldom been investigated. These representations were designed for cognitive style tasks like context-based attention, associative memory, and concept encoding. Cognitive tasks are also an area where traditional neural networks often don't perform well. Combining these ideas may add new avenues for learning.

## 2.2 Holographic Reduced Representation (HRR)

Holographic Reduced Representation (HRR) [17] is a form of distributed representation that uses fixed-length vectors with real number elements to represent data. HRRs provide a framework to simultaneously encode structural relationships and data from multiple $n$-dimensional vectors into a single $n$-dimensional vector. Encoding both structure and data in a single representation allows HRRs to effectively support tasks which require compositionality and a high degree of systematicity.

One interesting property of HRRs is the support for *holistic* transformation and mapping operations that do not require any decomposition into member components. Holistic processing is a useful property when building HRRs into more common neural network architectures. Without the explicit need to decompose an HRR representation, we can maintain HRR properties by placing constraints on network components during normal operations (e.g., normalizing vectors, HRR specific prior distributions). Previous work has examined systematic transformations of HRRs using typical gradient descent methods [24, 25, 26]. These techniques help characterize how HRRs may be used for commonplace neural network tasks.

HRRs offer many advantages for more complex cognitive style tasks, but in this work, we are primarily concerned with three of the more fundamental properties, encoding, decoding, and memory trace composition. Encoding consists of creating a memory trace by systematically combining two vectors. Decoding retrieves a target vector from a trace using a single vector associated with the target during encoding. Trace composition adds multiple traces together, while still supporting decoding of

components within.

Matrix-based associative memories often use some form of the outer product as an encoding operation. One drawback to this approach is that dimensionality grows as associations are added, quickly becoming unmanageable. Convolution (aperiodic or periodic) is an alternative encoding operation that can be regarded as a compression of the outer product [17]. As a type of convolution memory model, HRR is designed to make use of this compressed association for memory encoding, decoding, and composition.

Using aperiodic convolution still increases vector dimensionality as more associations are encoded, but to a lesser degree than the outer product. This expanding dimensionality problem can be entirely avoided using circular convolution, a conventional operation in signal processing. Circular convolution maintains fixed-width vector representations at the expense of compression noise in decoded reconstructions.

### 2.2.1 HRR Encoding

Encoding or associating two HRR vectors with circular convolution is central to much of the work we present in later sections. Circular convolution creates an $n$ dimensional trace, $\mathbf{t}$ by binding an input vector, $\mathbf{x}$ to a cue vector, $\mathbf{c}$ with the circular convolution operator $\circledast$, as follows:

$$\mathbf{t} = \mathbf{c} \circledast \mathbf{x} \rightarrow t_j = \sum_{k=0}^{n-1} c_k t_{j-k} \tag{2.1}$$

$$\text{for } j = 0 \text{ to } n-1 \quad \text{(subscripts are modulo-n)}$$

example $n = 3$ :

$$t_0 = c_0 x_0 + c_2 x_1 + c_1 x_2$$

$$t_1 = c_1 x_0 + c_0 x_1 + c_2 x_2$$

$$t_2 = c_2 x_0 + c_1 x_1 + c_0 x_2$$

### 2.2.2 HRR Decoding

Circular correlation is regarded as an approximate inverse of circular convolution [22], and it is used to reconstruct an $n$ dimensional vector, $\tilde{\mathbf{x}}$ which is a noisy reconstruction of $\mathbf{x}$. Reconstruction in this manner functions in a similar way to circular convolution, but operating on $\mathbf{t}$ and $\mathbf{c}$ vectors with the circular correlation operator $\mathbf{c}\circledcirc$, as follows:

$$\tilde{\mathbf{x}} = \mathbf{c}\circledcirc\mathbf{t} \rightarrow \tilde{x}_j = \sum_{k=0}^{n-1} c_k t_{k+j} \tag{2.2}$$

$$\text{for } j = 0 \text{ to } n-1 \quad \text{(subscripts are modulo-n)}$$

example $n = 3$ :

$$\tilde{x}_0 = c_0 x_0 + c_1 t_1 + c_2 t_2$$

$$\tilde{x}_1 = c_2 x_0 + c_0 t_1 + c_1 t_2$$

$$\tilde{x}_2 = c_1 x_0 + c_2 t_1 + c_0 t_2$$

### 2.2.3 HRR Distributions and Composition

Successfully decoding traces with circular correlation requires some conditions on the distribution of vectors used. In the base case, a sufficient condition is ensuring elements of each $n$-dimensional vector are i.i.d. with a mean of 0 and a variance of $1/n$. This is usually performed by creating HRR vectors (e.g., cues) by drawing from a random normal distribution, $\mathcal{N}(0, 1/n)$.

Compositionality with HRRs is also dependent on the distributional conditions above, since trace composition is performed using element-wise vector addition. Multiple encoded associations (traces) are added together to produce a vector representing all the individual associations in aggregate. For example, consider a compositional trace created using two data vectors $\mathbf{x}_1, \mathbf{x}_2$ and two cue vectors $\mathbf{c}_1, \mathbf{c}_2$, as follows:

$$\begin{aligned} \mathbf{t}_{1,2} &= (\mathbf{c}_1 \circledast \mathbf{x}_1) + (\mathbf{c}_2 \circledast \mathbf{x}_2) \\ &= \mathbf{t}_1 + \mathbf{t}_2 \end{aligned} \tag{2.3}$$

This composition can subsequently be decoded to retrieve a specific vector using circular correlation, for instance we can retrieve $\tilde{\mathbf{x}}_1$ using $\mathbf{c}_1$,

$$\begin{aligned} \tilde{\mathbf{x}}_1 &= \mathbf{c}_1 \circledcirc \mathbf{t}_{1,2} \\ &= \mathbf{c}_1 \circledcirc \mathbf{t}_1 + \mathbf{c}_1 \circledcirc \mathbf{t}_2 \\ &= \mathbf{c}_1 \circledcirc (\mathbf{c}_1 \circledast \mathbf{x}_1) + \mathbf{c}_1 \circledcirc (\mathbf{c}_2 \circledast \mathbf{x}_2) \end{aligned} \tag{2.4}$$

Decoding compositional traces in this way is possible due to the random nature of component vectors. Given the distributional conditions, there is a high probability that $\mathbf{c}_1$ has a low correlation with components in the second term $\mathbf{c}_2, \mathbf{x}_2$ $\mathbf{c}_1$, leaving

them disregarded as irrelevant noise. The remaining term follows normal decoding rules, resulting in a recognizable but distorted reconstruction $\tilde{\mathbf{x}}_1$.

Adding traces to a composition also increases reconstruction noise, so fidelity acts as a practical limit to the number of stored traces in a single vector. The value of $n$ also directly impacts this behavior, with larger vectors increasing composition capacity and reducing noise. Techniques like chunking were developed to specifically address this limitation [22], but we do not detail them here since they were not necessary in our work.

### 2.2.4 Frequency Domain HRRs

Convolution based memories offer an additional computational advantage, as they can be computed using Fast Fourier Transforms (FFTs). Encoding with FFTs requires $O(n \log n)$ time to compute, so they offer an attractive alternative to the base method in Eqn. 2.1 which takes $O(n^2)$ time. Circular convolution can be computer with one transform, an element-wise vector multiplication, and an inverse transform as follows,

$$\mathbf{a} \circledast \mathbf{b} = \mathcal{F}^{-1}(\mathcal{F}(\mathbf{a}) \odot \mathcal{F}(\mathbf{b})) \tag{2.5}$$

where $\odot$ is element-wise vector multiplication, $\mathcal{F}$ is a discrete Fourier transform, and $\mathcal{F}^{-1}$ is its inverse transform.

Decoding with circular correlation in the frequency domain is similar to encoding,

$$\mathbf{a} \circledcirc \mathbf{b} = \mathcal{F}^{-1}(\mathcal{F}(\bar{\mathbf{a}}) \odot \mathcal{F}(\mathbf{b})) \tag{2.6}$$

where $\bar{\mathbf{a}}$ is the complex conjugate of $\mathbf{a}$.

HRRs can be implemented entirely in the frequency domain using complex-valued vectors exclusively. This approach is not common in practice since using real-valued

vectors maintains compatibility with other ML techniques. Most often vectors are initialized and manipulated in the spatial domain, but FFT operations are used for encoding and decoding to speed up computation.

Computing HRRs in the frequency domain introduces some additional distributional considerations. In the frequency domain the exact inverse of a vector has elements with magnitudes equal to the reciprocal of original elements. The approximate inverse has the same magnitudes as the original. When the magnitude of all *frequency* components is 1, the exact inverse is equal to the approximate inverse. This class of vectors is referred to as *unitary vectors* [22].

Calculating the exact inverse for frequency-based HRRs can be computationally unstable. Unitary vectors allow us to avoid this by using the approximate inverse instead, which is still sufficient for decoding. Since we often start with spatial domain vectors, understanding their relation to unitary vectors is helpful.

Using the standard $\mathcal{N}(0, 1/n)$ initialization keeps vectors close to unitary when converted to a frequency representation, so it is a valid technique when using FFTs. However, a unitary vector converted to the spatial domain will have elements distributed as $\mathcal{N}(1/n, 1/n)$. This distribution offers an alternative way to initialize spatial vectors when compatibility with FFT operations is a prime concern.

## 2.3 Variational AutoEncoders (VAEs)

Variational Auto-Encoders (VAEs) are a family of models based on the Auto-Encoding Variational Bayes (AEVB) algorithm originally proposed by Kingma, et al. [27]. Conceptually, VAEs can be considered as a marriage of neural networks and directed probabilistic models derived from variational inference methods. Literature on VAEs can often be challenging for many readers due to mixed nomenclature from neural network and probabilistic model paradigms. We will attempt to distill and describe the relevant VAE methods here, and focus more on the neural network perspective.

For comparison purposes, consider a generic "vanilla" autoencoder design pattern with an encoder model and a decoder model. Both encoder and decoder are some arbitrary neural network. The encoder transforms data samples $x$ into a latent space representation $z$. The decoder attempts to reconstruct $x$ from $z$. We denote network parameters (i.e. weights and biases) as $\phi$ for the encoder, and $\theta$ for the decoder.

Training is unsupervised, optimizing $\phi$ and $\theta$ through standard gradient descent techniques based on reconstruction error. It's important to note that with a large enough latent space the model could potentially just memorize data samples, leading to overfitting. This behavior is usually discouraged by making the dimension of the latent space much smaller than the input dimension, leading to the common "hourglass" shape seen in autoencoder block diagrams. Smaller latent dimensions create an information "bottleneck" that encourages efficient compression of useful information. This compression helps improve the ability to generalize on unseen data since useful features for reconstruction are prioritized over redundant or overly-specific features.

In a very general sense, autoencoders can be described by simply defining an encoder, decoder, and a loss function. This simplified view is a good starting point to understand VAEs from a neural network perspective, which is essential for understanding work presented in later sections. Consider a VAE encoder denoted as $q_\phi(z|x)$, with neural network parameters $\phi$, as seen in our generic example. Similarly, the decoder is $p_\theta(x|z)$ with parameters $\theta$. However, in a VAE the latent space for $z$ is stochastic, which is why the encoder and decoder are notated as probability distributions.

Given that the objective of an autoencoder is usually to model that data $x$, we want to find the probability distribution of the data $p(x)$. Using standard probability rules, we can try to find this using a joint variable $z$, and marginalizing it out of the

joint distribution,

$$p(x) = \int p(x|z)p(z)dz \tag{2.7}$$

Calculating this integral directly is typically intractable for real-world data, so another technique is needed. The idea behind VAEs is to infer $p(z)$ with observable data using $p(z|x)$. In other words, we want to find likely values for $z$ given the data available. Unfortunately, the true distribution $p(z|x)$ is also not known by default since $z$ is a latent variable. Variational inference allows us to approach modeling $p(z|x)$ as an optimization problem using a simpler distribution (e.g., Gaussian). Most often we assume $p(z)$ to be a Gaussian prior, which allows drawing samples characterized by only two parameters, mean $\mu$ and variance $\sigma$. For explanatory purposes, we assume a Gaussian prior throughout the rest of this section.

In a VAE, we use this distributional assumption about $p(z)$ in the encoder $q_\phi(z|x)$. Encoding can also be regarded as projecting $x$ into the latent variable space $z$. The encoder uses neural network layers to produce values for $\mu$ and $\sigma$. Location-scale operations with these parameters are used to transform random samples from $p(z)$ into approximations of samples from the true distribution $p(z|x)$.

The decoder $p_\theta(x|z)$ consists of neural network layers that take $z$ as an input, and produces reconstructions of $x$ as output. A VAE decoder is also commonly referred to as a generator, because it generates an $x$ from $z$. Aside from the probabilistic elements, it is easy to see how a VAE fits into the autoencoder paradigm, leaving the loss function as the last model component to review.

VAE loss usually consists of a reconstruction error term, and one or more regularization terms. In the base case with one latent variable $z$, we can describe the loss function as,

$$\mathcal{L} = \mathbb{E}_{q_\phi(z|x)}\left[p_\theta(x|z)\right] - D_{KL}(q_\phi(z|x) \parallel p_\theta(z)) \tag{2.8}$$

where the first term is reconstruction loss in the form of expected negative log likelihood, and the second term is a regularizer based on Kullback-Leibler (KL) divergence, which plays a crucial role in VAE optimization.

KL divergence measures the information lost when the approximated distribution $q_\phi(z|x)$ is used to represent the assumed true distribution $p(z)$. When the encoder produces $z$ samples with a distribution that diverges from $p(z)$, a penalty is imposed in the loss. Without this regularization, the model can just learn to place each sample in a different region of the latent space, causing poor generalization.

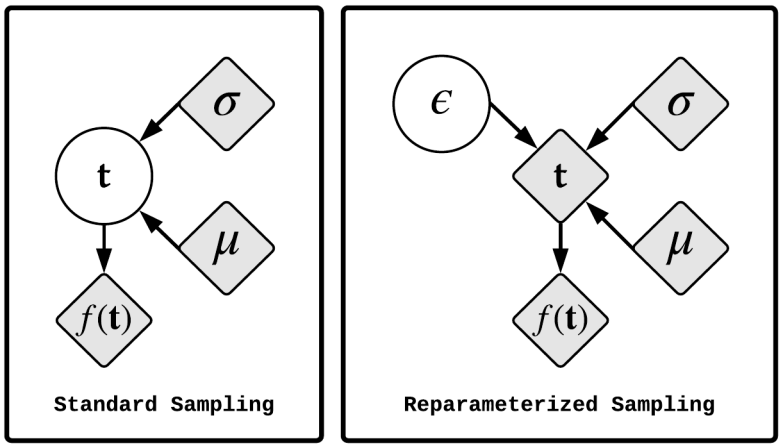



**Figure 2.1:** Graphical sketch of the reparameterization trick used in VAEs. Provides a differentiable sampling process usable in neural network architectures. Analogous to sampling from "location-scale" distributions. Shaded diamonds indicate deterministic nodes, and circles represent random variables.

One final consideration for VAEs is the method used to make the model training with gradient descent. Classic statistical sampling is not differentiable, meaning gradients cannot flow through this operation during backpropagation. VAEs are reliant on sampling, so an alternative sampling process was needed to make the model trainable via standard methods. The *reparameterization trick* [27] was developed to address this problem by modifying the sampling process to make it differentiable.

For example, let the prior distribution $p_\theta(\mathbf{z}) \sim \mathcal{N}(\boldsymbol{\mu}_x, \boldsymbol{\Sigma}_x)$ be a normal Gaussian

distribution. A neural network takes inputs $\mathbf{x}$ and produces $\boldsymbol{\mu}_x, \boldsymbol{\Sigma}_x$ values used to parameterize $p_\theta(\mathbf{z})$. Rather than sampling directly from $p_\theta(\mathbf{z})$, we can sample in a differentiable way by re-working the process to only use deterministic operations within the network.

To make sampling differentiable, stochastic operations must be moved outside of the network (i.e., gradient flow). We use a distribution $\boldsymbol{\epsilon} = \mathcal{N}(\mathbf{0}, \mathbf{1})$ which generates samples completely independent of network activity. Since samples from $\boldsymbol{\epsilon}$ are Gaussian, they can be transformed into the distribution parameterized by $\boldsymbol{\mu}_x, \boldsymbol{\Sigma}_x$ using simple deterministic operations that are differentiable. This process is the reparameterization trick illustrated in Fig. 2.1, and described mathematically as,

$$\mathbf{z} = \boldsymbol{\mu}_x + \boldsymbol{\Sigma}_x^{1/2} \odot \boldsymbol{\epsilon} \tag{2.9}$$

## 2.4 Neuroscience Perspective on One-Shot Learning (OSL)

In their 2015 paper Lee et al. [1] explore the processes that govern OSL in biological intelligence from a neuroscience perspective. This work was the primary inspiration for our approach to developing a new OSL method for machine intelligence. While our work is concerned with machine intelligence, biology provides a rich source of inspiration. Describing some of the mechanisms that contribute to biological OSL will be illustrative when discussing our OSL system in later chapters.

This section will provide a brief summary of key ideas from Lee's work. From a neuroscience perspective, our treatment of these ideas will be very shallow, as they are only intended to provide a general conceptual framework. Any references to biological mechanisms or experiments in this section are drawn from Lee's paper unless otherwise noted, and readers desiring a more in-depth biological analysis should refer to that work.

In a general sense, there are at least two very distinct learning strategies employed

in our brains. When presented with a stimulus, we attempt to identify and understand the causal relationship between that stimulus and its consequence. The most common way we acquire knowledge of this type is through trial and error or incremental learning. However, in some cases we need to learn this relationship rapidly, perhaps from a single example, and that is where OSL comes into play.

Given these two general learning modes, an obvious unknown is the mechanism by which our brains switch between incremental and one-shot learning. Lee proposes that the amount of causal uncertainty between cause and consequence facilitates this switch. In a simplified view, when a stimulus invokes a high degree of uncertainty in the causal relationship, a higher learning rate is applied.

Experimental evidence with human subjects showed that there are two regions of the brain primarily contributing to switching between learning modes, the ventrolateral prefrontal cortex (vlPFC), and the hippocampus (HPC). It has often been hypothesized that the vlPFC helps guide a control system the brain uses to determine which items to remember or forget during learning. The hippocampus is a fascinating and complex part of our brains with details and theoretical properties well beyond the scope of this work. For our purposes, the hippocampus is of interest due to its critical role in consolidating information and forming memories.

Lee performed behavioral experiments with human participants that involved collecting fMRI data during a visual OSL task. Computational models and fMRI data were compared to determine areas of the brain associated with OSL and the role of factors like novelty and causal uncertainty in modulating learning.

Experimental results showed activity that was positively correlated with novelty in brain areas like the dorsal parts of the prefrontal cortex, inferior parietal lobe, middle temporal gyrus, and Caudate. A negative correlation for novelty was observed with activity in the fusiform gyrus extending to the parahippocampal gyrus. Some of these findings have been adapted from the original paper and provided in Fig.

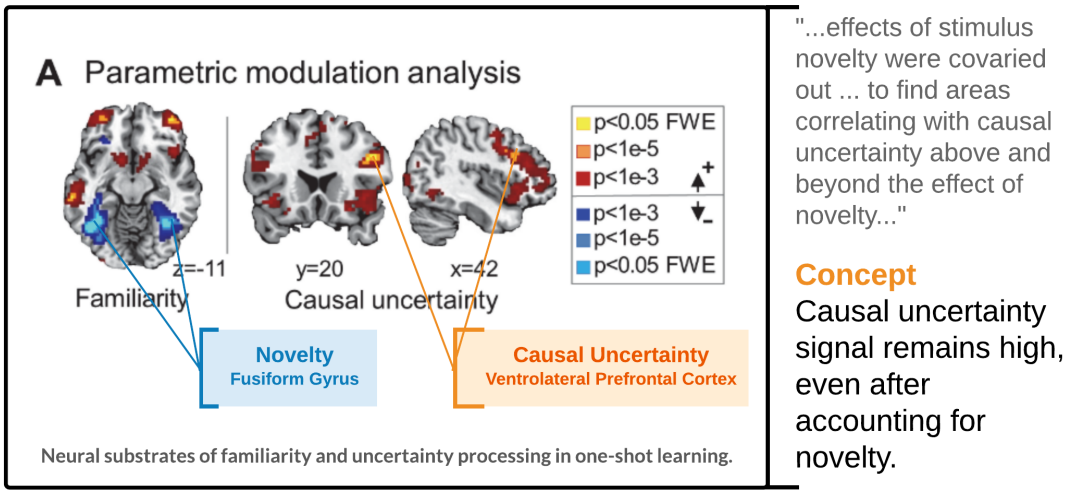



**Figure 2.2:** Familiarity and uncertainty in OSL. Figure adapted from [1].

2.2. Correlations were used to identify activity related to uncertainty processing, beyond the effect of novelty. After accounting for novelty, activity in the vlPFC still correlated with causal uncertainty, supporting the hypothesis that it plays an active role in uncertainty processing.

During events where computational models predicted a high learning rate, participants were expected to implement OSL. Analyzing brain activity during these events showed an increased degree of functional coupling between the vlPFC and hippocampus. The authors speculate that these two regions act in concert during learning, with the vlPFC acting as the "switch," turning OSL on or off when needed as seen in Fig. 2.3. Computational models produced evidence supporting this idea. They showed learning rates that were modulated in a highly nonlinear way due to the vlPFC encoding of causal uncertainty signals.

During incremental learning events coupled activity indicating hippocampus involvement was not observed. Since this coupling behavior was only observed during OSL, it follows that the hippocampus is selectively recruited in the presence of high causal uncertainty. This recruitment is an indicator that OSL "mode" is engaged. When considering the effect of novelty alone, this indicator was not present. Over-

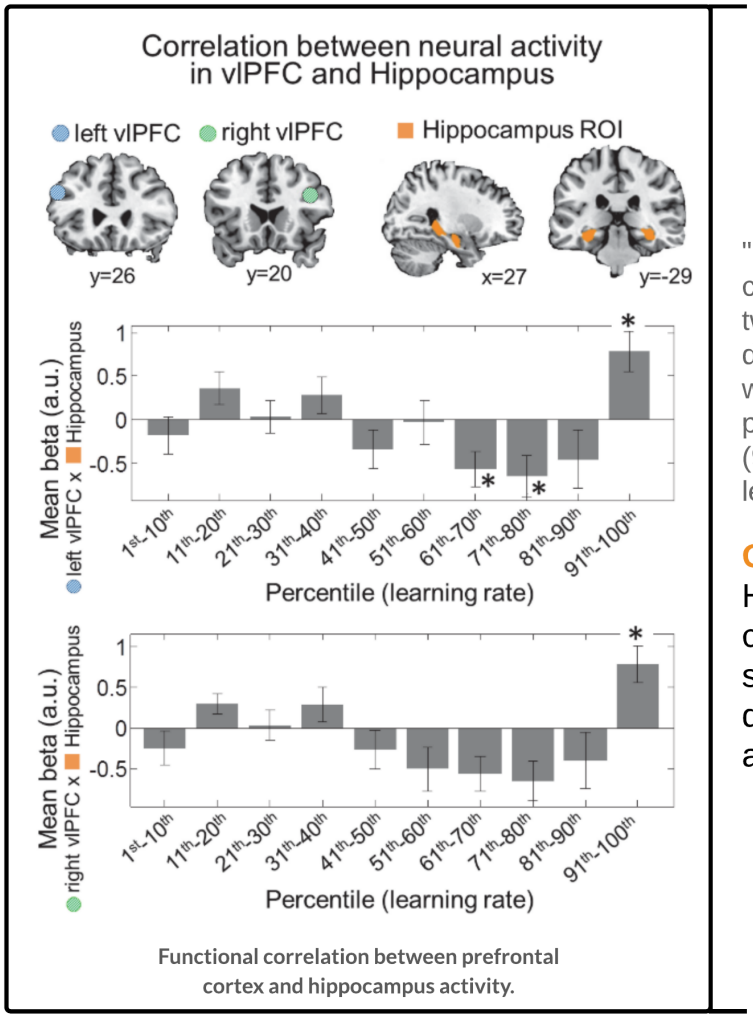



**Figure 2.3:** Correlation between vlPFC and Hippocampus activity. Figure adapted from [1].

all, there is substantial evidence that causal uncertainty rather than novelty is the primary driver for OSL in humans.

In ML we tend to think of OSL as purely novelty-driven, but these experimental results suggest that studying uncertainty-based OSL may be a productive area for ML research. While the computational models used in neuroscience may not directly translate to ML, we can still use them as a basis for developing new approaches.

From a modeling perspective Lee's work offers some methods that are of interest to the work we present in later chapters. The primary model components of interest are

those used to implement uncertainty-based modulation of learning rate. The general concept is that learning rate is not a constant, but a function of uncertainty in the causal relationship between a stimulus and an outcome (causal uncertainty). Using a Bayesian inference model provides a way to estimate causal uncertainty and strength as the variance and mean of a posterior distribution respectively.

In the source work, there are only a small number of possible outcomes, and they are known prior to the experiments. This motivates use of a finite mixture model to infer latent classes. Many of the exact details on the specific model used are less relevant to our work, since ML tasks often have an incompatible set up, but examining their technique for controlling learning rate is informative. Their model computes learning rate using the amount of causal uncertainty for each individual stimulus with a softmax operation, as follows,

$$\gamma_i = \frac{exp(\tau Var(\theta_i|D))}{\Sigma_j exp(\tau Var(\theta_j|D))} \tag{2.10}$$

where $\theta$ is a prior probability, $\theta|D)$ is a posterior conditioned on evidence $D$, and $\tau$ is an inverse temperature parameter controlling the impact of high posterior variance on the learning rate for a stimulus. This approach allows the model to converge as uncertainty is resolved, rather than converging after the same types of events occur repeatedly.

Values in the posterior distribution are denoted $\alpha_i = \lambda_i + x_i$ for the ith stimulus, where $\lambda_i$ is a value from the initial prior and $x_i$ is a salience value. Updating $p(\theta|D)$ is performed using $\gamma_i$ as the learning rate with $\Delta\alpha_i = \pm\gamma_i x_i$, where the sign determined by a correct pairing with an expected outcome.

## 2.5 Uncertainty in Neural Networks

Neural network models do not usually feature a way to capture model uncertainty. There is often a tendency to erroneously interpret outputs as model confidence when probabilistic values are involved. For example, with classification models we might be tempted to interpret a probabilistic predictive output (e.g., softmax layer output) as model confidence. This interpretation is faulty. A model can be uncertain about its predictions even when a high softmax output is observed [28]. Flawed assumptions like these can lead to misinformed design decisions and potentially catastrophic failures. Consider an autonomous vehicle with this type of confidence assumption built into its safety system for instance, the result could be dangerous.

Alternatively, the ability to measure uncertainty in a more principled way adds additional tools for designing and optimizing models. Model uncertainty is a common part of Bayesian probability theory, but Bayesian techniques can be expensive computationally, and therefore impractical for typical neural network applications. There have been some developments bridging the gap to combine Bayesian theory and neural networks, but this line of uncertainty research is still a relatively new. Useful methods have already emerged though, with applications like agent exploration [29, 30] and adversarial example detection [31].

One approach to neural network uncertainty hinges on interpreting dropout [32] as a Bayesian approximation of the Gaussian Process [33]. A theoretical framework to support this interpretation was introduced in the original work [28], and later extended with several variants. This approach is especially advantageous for neural network models in a practical way since dropout is already a common technique readily supported by popular software frameworks and tools. Additionally, it is general enough to apply for nearly any neural network model that can support dropout.

Standard dropout is typically implemented as a discrete mask with some fixed

probability value that determines how the mask is applied to values. Concrete Dropout (CD) [18] is an extension of the dropout-based uncertainty approach that provides a continuous relaxation of the dropout mask. Rather than using a fixed probability value for dropout, CD optimizes this value as part of the usual gradient-based neural network training process. Interpreting dropout from a variational perspective provides the optimization objective from [18] as follows,

$$\hat{\mathcal{L}}_{MC}(\theta) = -\frac{1}{M}\sum_{i\in S}\log p(\mathbf{y}_i|\mathbf{f}^{\boldsymbol{\omega}}(\mathbf{x}_i)) + \frac{1}{N}D_{KL}(q_\theta(\boldsymbol{\omega}) \parallel p(\boldsymbol{\omega})) \tag{2.11}$$

where $\theta$ as parameters to optimize, $N$ the number of data points, $S$ a random set of $M$ data points, $\mathbf{f}^{\boldsymbol{\omega}}(\mathbf{x}_i)$ the network's output on input $\mathbf{x}_i$ when evaluated with weight matrices realization $\boldsymbol{\omega}$, and $p(\mathbf{y}_i|\mathbf{f}^{\boldsymbol{\omega}}(\mathbf{x}_i))$ the model's likelihood. The KL term $D_{KL}(q_\theta(\boldsymbol{\omega}) \parallel p(\boldsymbol{\omega}))$ adds regularization to ensure that the approximate posterior $q_\theta(\boldsymbol{\omega})$ stays near the prior distribution $p(\boldsymbol{\omega})$.

Standard dropout masks use a discrete Bernoulli distribution which is not compatible with the optimization in Eqn.2.11, so the continuous Concrete distribution is used to approximate these discrete random variables. In the one-dimensional case of the Bernoulli random variable $\mathbf{z}$, the Concrete relaxation $\tilde{\mathbf{z}}$ reduces to a simple sigmoid distribution that makes for easy parameterization,

$$\tilde{\mathbf{z}} = \text{sigmoid}\left(\frac{1}{t}\cdot(\log p - \log(1-p) + \log u - \log(1-u))\right) \tag{2.12}$$

with some temperature value $t$ and uniform $u \sim \text{Unif}(0,1)$. This relaxation of dropout masks is how CD is defined.

From an implementation standpoint, CD is advantageous because it provides a way to measure uncertainty during normal neural network operation by making a few simple modifications. This is accomplished by applying a layer wrapping function that implements CD functionality and adds the loss from Eqn.2.11 to the network's

overall loss function.

# Chapter 3

## Autoencoding Cleanup Memory (AECM)

### 3.1 Autoencoding Cleanup Memory Overview

Data reconstructed from convolution memory (e.g., HRR) is inherently noisy due to the compression that takes place during binding. When a task only requires recognition, and not recall, this is not much of an issue. Similarity measures like the vector dot product can be used holistically to test for recognition without actually reconstructing encoded data. However, in tasks that require accurate reconstruction, additional error-correcting auto-associative item memory is needed.

In HRR literature this item memory described as cleanup memory. The purpose of cleanup memory is to produce a clean version of noisy reconstructed vectors it receives as input. It compares inputs to stored items and outputs the best match along with a scalar "strength" value. The strength score measures how closely a cleanup memory output matches the original noiseless data sample with a higher score indicating a close match. The exact implementation of cleanup memory is generally considered unimportant as long as it provides the necessary capabilities. In practice, cleanup memory implementation can have a significant impact on computational cost, especially if it does not scale well with the number of items stored.

In Plate's initial HRR simulations [17] cleanup memory is kept very simple. His basic implementation used a simple array for item storage, by adding a pristine copy

of every data vector observed in an experiment. Cleanup processing then consists of calculating dot products between a noisy query vector and all vectors in item memory. The item producing the highest dot product value (i.e., strength) is output as the closest match.

Although a simple list-based implementation is embarrassingly parallel, it is not practical for larger memories since the number of dot product operations needed for a single recall grows for every vector stored. Note that Plate did not suggest this simple system was a scalable solution in the original literature, only a simple way to perform the small-scale simulations he presented. Using an array or other similar data structure is undoubtedly a very straightforward type of cleanup memory though, so it is useful to consider it as an example.

Hopfield networks can have potential as an auto-associative cleanup memory, but their capacity related to vector size limits their usefulness. Other recurrent neural networks (RNNs) like Long Short-Term Memory (LSTM) [34] would also likely be applicable for cleanup, but to our knowledge an in-depth study on using contemporary RNNs for cleanup remains relatively unexplored. Stewart et al. [35] did investigate cleanup memory for VSAs quite extensively, even focusing on HRRs. Their focus was on spiking neural networks and those techniques are not readily adaptable to the models we develop in this work. Overall, there has been limited research produced investigating cleanup memory for HRRs, especially from a machine learning perspective.

We investigated potential HRR cleanup memory implementations that would be compatible with contemporary neural network architectures. The purpose of this search was to find a cleanup memory that does not require explicit bookkeeping (e.g. lists, lookup-tables, etc.), scales well with larger datasets, and is capable of acting as a plug-and-play component for models we describe in later chapters.

In this chapter we describe our approach to implementing an Autoencoding Cleanup

Memory (AECM) network that learns to remove HRR reconstruction noise with a simple de-noising Autoencdoer (dAE) network. We found that a standard dAE model essentially works off-the-shelf as a cleanup memory where items are stored in a distributed way across network parameters via unsupervised training. This approach is advantageous since it readily adapts to remove noise without requiring explicit storage and costly repeated similarity computations.

## 3.2 AECM Model

The AECM model is a stacked dAE neural network re-purposed to act as a proxy for an explicit cleanup memory. Although dAEs were originally conceived as an intermediate component in deep learning models used to initialize weights with unsupervised pre-training, we found that they are useful as a stand-alone network that makes data representations robust to noise.

In our context, AECMs are used to cleanup the reconstruction noise $\eta$ added when decoding an HRR trace. The basic concept for AECMs can be characterized starting with $\mathbf{c}$, $\mathbf{t}$, and $\mathbf{x}$ which represent cue, trace, and data vectors respectively. Encoding, decoding, and cleanup can be outlined as,

$$\mathbf{t} = \mathbf{c} \circledast \mathbf{x} \tag{3.1}$$

$$\hat{\mathbf{x}}_\eta = \mathbf{c} \circledcirc \mathbf{t} = \mathbf{c}^* \circledast \mathbf{t} \tag{3.2}$$

$$AECM(\hat{\mathbf{x}}_\eta) = \hat{\mathbf{x}} \tag{3.3}$$

where 3.1 shows binding to produce traces, 3.2 is decoding traces into a noisy reconstruction of the input $\hat{\mathbf{x}}_\eta$, and 3.3 is the AECM network removing noise to produce a cleaned up version of the input $\hat{\mathbf{x}}$. The model is unlikely to produce a completely pristine version of $\mathbf{x}$, so $\hat{\mathbf{x}}$ can be considered a very close approximation of the original.

The neural network architecture of the AECM model used in our experiments was a very standard dAE design that contains fully-connected 2D convolutional neural network (CNN) layers for filtering, with a block diagram provided in 3.1. The encoder uses 2D max pooling following each CNN layer to spatially aggregate filter outputs. This aggregation progressively reduces dimensionality, bottlenecking the representation produced at the encoder output layer (i.e. the latent space). Decoding is performed in a similar manner with CNN layers. Dimensionality is progressively expanded back to the original size using simple 2d upsampling that repeats data to expand representation size at a given layer.

This architecture worked very well for our purposes, but it should be noted that the specific architecture used for AECM is largely unimportant in a general sense. Some other network or process could reasonably be substituted given that it follows the same pattern of receiving corrupted input data and learning to produce cleaned data via unsupervised learning. We envision the AECM as a drop-in component for other models, so this flexibility in architecture definition is convenient. For instance, CNN dAEs make sense for image processing tasks, but other types of data would likely require different network architectures for the AECM to be effective.

## 3.3 Example Experiments

In this section we present some basic experiments with image data to demonstrate reconstruction noise removal capability with the AECM model. We performed cleanup experiments on two datasets, MNIST [36] and Fashion-MNIST [37]. Both datasets contain 60,000 training samples and 10,000 testing samples of grayscale 28x28 images, with 10 object classes.

Experiments were performed using HRR reconstructions of every sample as inputs to the AECM model. As an initial step, all samples were pre-processed with l2 normalization to make them more compatible with HRR operations. Reconstructed

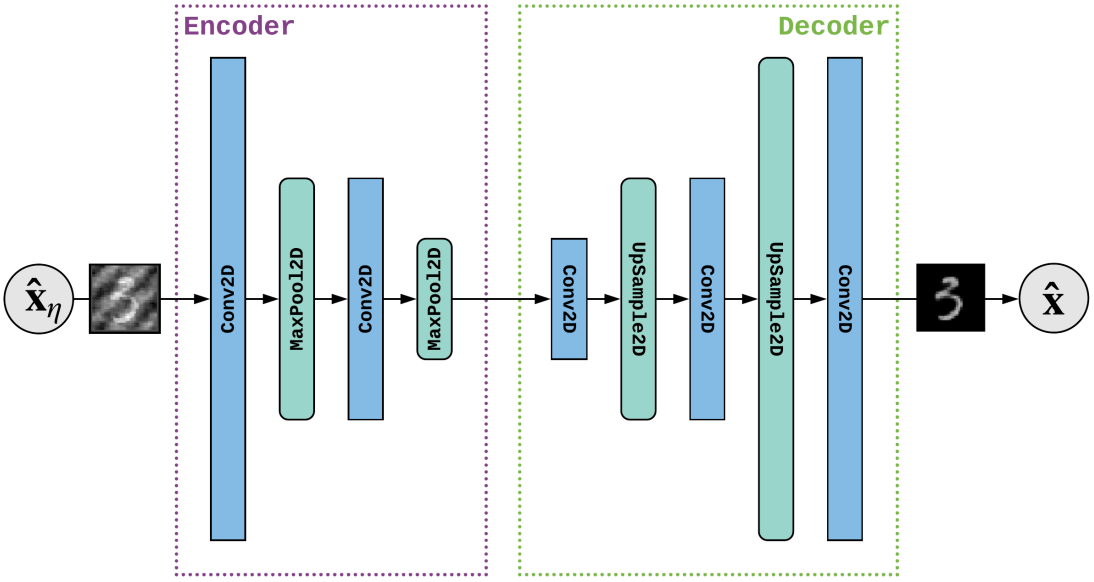



**Figure 3.1:** Autoencoding Cleanup Memory (AECM) example network architecture. $\hat{\mathbf{x}}_\eta$ indicates a noisy reconstruction of $\mathbf{x}$ produced by circular correlation decoding, and $\hat{\mathbf{x}}$ indicates a cleaned reconstruction produced by the AECM network.

samples for model input were then prepared using the basic HRR encoding and decoding process. First, unique cue vectors were generated randomly for every sample by drawing from the unitary HRR distribution $\mathcal{N}(1/n, 1/n)$. Next, all cue, data vector pairs $(\mathbf{c}, \mathbf{x})$ were encoded into trace sets $\mathbf{t}_{train}, \mathbf{t}_{test}$ using circular convolution. Traces were then decoded into the noisy reconstructions $\hat{\mathbf{x}}_{\eta train}, \hat{\mathbf{x}}_{\eta test}$, and used to train the AECM model.

Identical neural network architectures were used for both datasets and built using the base architecture described in section 3.2. In these experimental models, hidden CNN layers used 32 filters with $3 \times 3$ kernels and ReLu activation functions. The output layer contained a single filter and used a sigmoid activation function to constrain the output values to $(0, 1)$. Models were trained using standard mini-batch style gradient descent methods, with a batch size of 32. Since the network output was sigmoidal, gradients were determined using binary cross-entropy reconstruction loss between pristine input $\mathbf{x}$ and AECM output $\hat{\mathbf{x}}$. All optimization was performed

using Adam [38] with a learning rate of 0.001. No additional effort was made to optimize hyper-parameters, as these experiments are intended to demonstrate basic capability not optimized performance.

### 3.3.1 Experiment Results

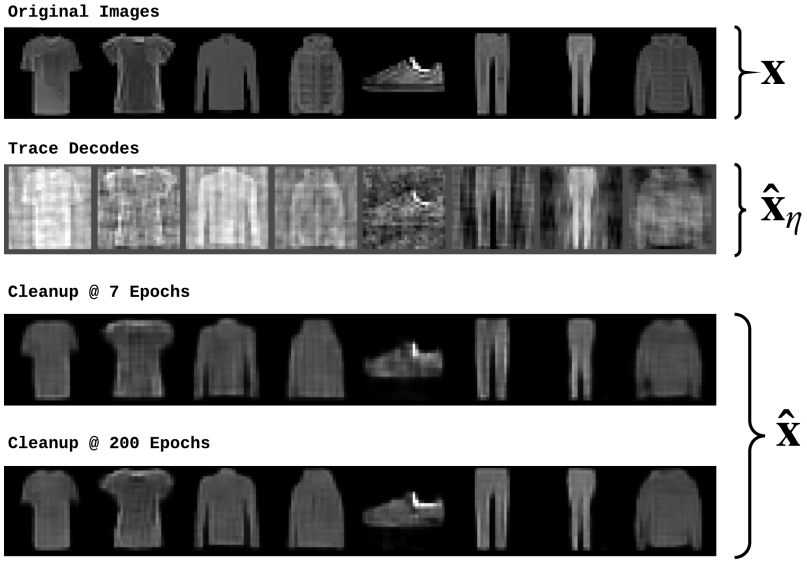



**Figure 3.2:** AECM results on Fashion-MNIST, with eight random samples from test set shown. Original images $\mathbf{x}$ (top row), noisy images decoded with circular correlation $\hat{\mathbf{x}}_\eta$ (second row), and AECM cleanup results $\hat{\mathbf{x}}$ at epoch 7 and epoch 200 for comparison (last two rows).

Examining AECM reconstructed samples from the test set shows a notably reduced noise and sometimes blurry but recognizable high-level image features (e.g. size, shape) on both datasets, even after the first epoch. On subsequent training passes, blur was reduced and finer grain visual details started to become apparent. Example AECM reconstructions at various training passes are provided in Fig. 3.2 for Fashion-MNIST and 3.4 for MNIST. Cleanup samples produced on MNIST are nearly identical to the source data visually and examining samples over multiple epochs shows very little variation as training progresses.

It is evident from the loss plot in Fig. 3.3 that given our experiment parameters,

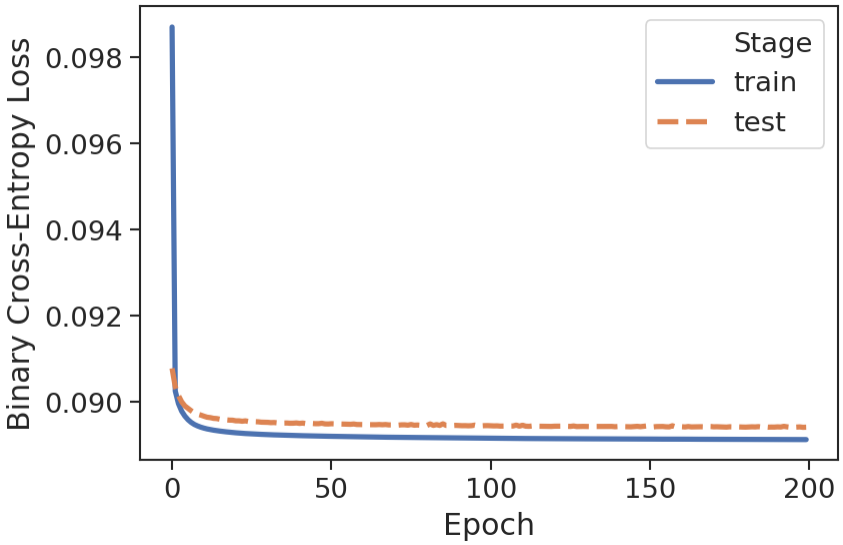



**Figure 3.3:** AECM binary cross-entropy loss while training on Fashion-MNIST over 200 epochs.

the AECM model begins to converge around epoch 80, showing a test loss of $\approx 0.0912$ on Fashion-MNIST. Training on MNIST converges immediately after one epoch, with a test loss of $\approx 0.0484$, and reducing to $\approx 0.0483$ after 200 epochs as shown in Fig. 3.5. This quick convergence reinforces the visually observed result that suggested the AECM learned to effectively cleanup noise on MNIST after only a single training pass.

## 3.4 Remarks on AECM

Cleanup memory is a crucial component that must be considered when applying HRRs. From an engineering standpoint, cleanup memory is one of the first real computational obstacles encountered when actually using HRRs. Anything beyond a toy dataset or data with carefully curated relationships can cause the comparison dot products to quickly become impractical as vectors are added. So, the nature of cleanup memory implementation can have a significant impact on computational cost.

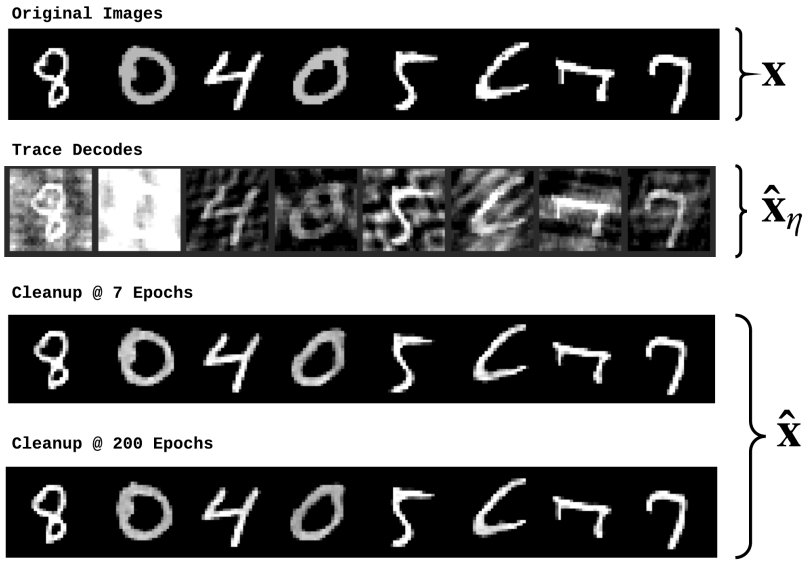



**Figure 3.4:** AECM results on MNIST, with eight random samples from test set shown. Original images $\mathbf{x}$ (top row), noisy images decoded with circular correlation $\hat{\mathbf{x}}_\eta$ (second row), and AECM cleanup results $\hat{\mathbf{x}}$ at epoch 7 and epoch 200 for comparison (last two rows).

Unfortunately, this topic is often glossed over or ignored in all but a few previous works. One of our primary research goals for work we present later was to create HRR models that are practical and usable in contemporary applied machine learning settings. To that end, we needed a solution for cleanup memory that was both flexible and compatible with the larger neural network design paradigm.

Using a dAE neural network architecture in the cleanup memory role seems like an obvious choice when HRR tasks do not require explicit item storage or indexing. Conceptually, it also fits well with the idea of distributed representation since the information needed to de-noise reconstructions is stored across network weights. Despite this seemingly straightforward connection, to our knowledge it's not a technique that has been previously applied to the problem.

Cleaning up noisy vectors with a dAE is obviously not a surprising idea, and we don't consider the AECM to be a new contribution in that sense. However, applying this sort of architecture to the specific problem of cleanup memory does contribute

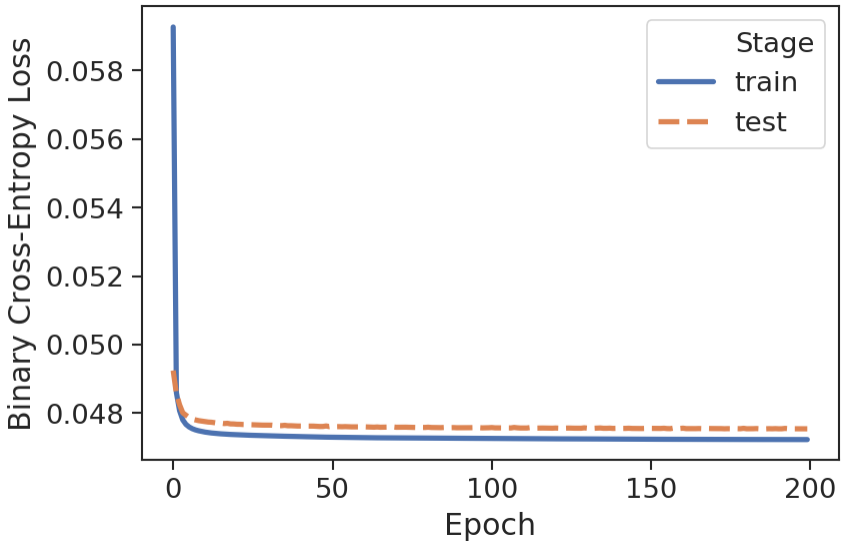



**Figure 3.5:** AECM binary cross-entropy loss while training on MNIST over 200 epochs.

a new and simple solution for an often overlooked problem. We elected to present AECM here as a stand-alone model because it reflects how we use them in the more complex models presented later on, as a drop-in component well-suited to fill the role of cleanup memory.

# Chapter 4

## Holographic Variational Autoencoder (HVAE)

### 4.1 Holographic Variational Autoencoder Overview

HRRs offer some interesting capabilities like content-addressable memory and structural representation. In a broad sense, they are readily compatible with contemporary machine learning since they can be implemented using standard mathematical operations and carry the added benefit of using only real-number valued vectors. Although this compatibility is evident, machine learning research using HRRs is rare.

Research on using HRRs in machine learning is uncommon, but it is not completely unexplored. Plate made an effort to investigate this topic during his original HRR work and he devoted an entire chapter in his 2003 book [22] to using HRRs in systems that learn. That work focused on using HRRs with RNN models available at the time for sequence learning. This line of research was extended in recent years by combining HRRs with more powerful RNN models like LSTM [39]. Applications of learning with HRRs have been studied for knowledge graph embeddings [25] and deep learning network feature representation [40]. More general studies on the subject have focused on learning systematic HRR transformation [24], and extending the theory behind HRRs to add utility [26]. These are all useful additions to HRR knowledge, but in the applied works HRRs are often used in way that is very task specific, or reliant on explicit user knowledge to hand-craft representations.

Developing methods to combine HRRs (or other VSAs) with present-day machine learning techniques will lead to new families of models. Intuitively, adding capabilities like content-addressable memory or compositional structure representation via HRR should augment our machine learning models in interesting ways. Investigations of this nature are very limited though, and therefore the potential value added by this type of augmentation is still mostly unknown. The work presented in this chapter was motivated by the desire to explore this combination in new and potentially useful ways. We take some initial steps toward augmenting modern neural networks with HRRs as a proof of concept while simultaneously mitigating some of the drawbacks native to HRRs.

In the rest of this chapter we will detail a newly developed model, the Holographic Variational Autoencoder (HVAE). HVAE is a novel type of conditional variational autoencoder (CVAE)[41] that combines concepts from VAEs and HRRs. A standard CVAE usually accepts pairs of data vectors and a conditioning variable as input. The conditioning variable is often discrete (e.g., a class label). The encoder learns a conditional latent variable representation from inputs, and that latent variable is used by the decoder to reconstruct data samples in the usual VAE manner. Adding a conditional variable has been used to provide interesting capabilities like producing scaling and rotation transforms in the decoded value by altering the conditioning variable.

The HVAE follows this general design, with some notable differences. It takes pairs of data and HRR cue vectors as inputs and learns a conditional latent representation. However, the latent variable produced by the HVAE is a memory trace in an HRR sense, which we refer to as a latent trace. This is achieved by implementing the decoder as a circular correlation HRR decoding operation rather than the traditional MLP stack. The intuition here is that using an HRR decoding scheme introduces strong constraints that force the model to learn valid HRR latent traces in order to

produce reconstructed outputs. In essence, the model learns to approximate HRR encoding with a stochastic latent space in an unsupervised manner.

To describe how HVAE works, let us first consider model function at a very high level. Our goal is to have a model that receives data $x$ and cues $c$ as inputs, learns a representation for the trace $t$ as a latent variable, and then outputs a reconstructed version of the data, $\hat{x}_\eta$ which approximates a reconstruction produced by HRR encoding and decoding. This can be described with the following relation,

$$\text{HVAE}(\mathbf{c}, \mathbf{x}) = \hat{\mathbf{x}}_\eta \tag{4.1}$$

$$\simeq \mathbf{c} \circledast (\mathbf{c} \circledast \mathbf{x}) = \mathbf{c} \circledast \mathbf{t} \tag{4.2}$$

In Fig. 4.1 we provide several graphical model sketches to help illustrate model function, and we will refer to these in the following sections to describe various model components like the generation or recognition systems.

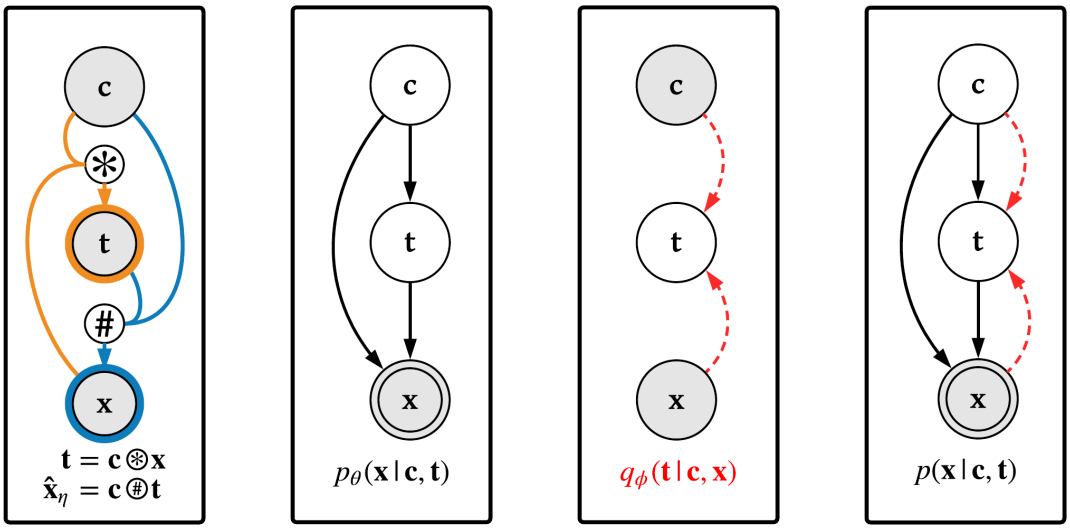
#### 4.1.0.1 Probability Distribution Assumptions

Using HRRs in HVAE necessitates selecting a slightly different form of assumed Gaussian prior to keep representations valid as HRR vectors. Unless otherwise noted, we use a normal Gaussian prior that is the spatial equivalent of the unitary HRR normal distribution $\mathcal{N}_{HRR}$, defined using the HRR vector size $n$,

$$\mathcal{N}_{HRR} = \mathcal{N}(1/n, 1/n) \tag{4.3}$$

#### 4.1.0.2 Encoding Structure as an Alternative Information Bottleneck

In general, autoencoders need some sort of constraint that forces the encoder model to learn a more efficient compression of data. This is often described as creating a




**(a)** HRR Operations **(b)** Generation **(c)** Recognition **(d)** Overall Model

**Figure 4.1:** Graphical sketches of the HVAE model, decomposed for illustration. Red dashed lines indicate approximate inference distributions $q(\cdot|\cdot)$. **4.1a**: Data flow for general deterministic encoding and decoding with HRR operations. Orange lines represent binding cues and data into traces. Blue lines, decoding with cues and traces into data reconstructions. **4.1b**: Generative model parameterized by $\theta$. **4.1c**: Recognition model parameterized by $\phi$. **4.1d**: Overall HVAE graphical model sketch.

compressed or reduced knowledge representation of the input. The idea behind this is that encoding and decoding data with completely independent features is much more difficult than doing so when the features have some type of structure, for instance feature-wise correlation.

Autoencoders usually leverage this concept by keeping the size of the latent space representation $z$ much lower than the size of the input data, which is commonly referred to as an information "bottleneck" constraint. Since autoencoders are trained using reconstruction error and the latent space has limited capacity, the network must learn to ignore redundant features while becoming more sensitive to the features that allow accurate reconstructions (ideally). Without this constraint the network is unlikely to learn any feature correlations and just try to memorize every training sample (i.e., overfitting).

Bottlenecks in VAEs take a slightly different form. They usually do include a dimensional constraint on the latent space, as with basic autoencoders. However, there is an additional bottleneck due to the probabilistic nature of latent space representations. The loss function for VAEs contains a reconstruction error term and a KL divergence term that keeps predicted and prior distributions similar. If we removed the reconstruction error during training and only used KL divergence, the VAE would just use the same unit Gaussian to represent every observed sample, overfitting in a different way. Since we optimize these two terms together, the latent representation can diverge from the prior when it helps describe useful features. Manipulating this distributional bottleneck in the latent space is an interesting line of research explored in models like beta-VAE [42, 43]. In beta-VAE a scalar value $\beta$ is used to modulate the latent information bottleneck by scaling KL divergence.

In the HVAE model we wanted a latent representation that approximates HRR traces, which necessitates a latent dimension equal to the input dimension. Additionally, HRR information capacity is improved as vector size increases [22], so larger vectors are desirable from an HRR viewpoint. Given these considerations, creating a bottleneck by reducing the size of latent representations didn't make sense for HVAEs. However, the need for a bottleneck beyond the distributional one provided by the VAE loss function remained a problem.

Searching for a way to approach the bottleneck problem helped motivate our choice to use HRR operations directly during decoding, so we will try to explain the intuition behind this choice here. Loosely defining the bottleneck needed, let us regard it as some mechanism for compressing information. Ideally, this compression projects data into a space that facilitates learning latent structure and features more efficiently.

HRRs actually provide this type of compression mechanism by default. They are a lossy compression of a higher-dimensional representation by definition [22]. Expand-

ing on that a little further, when we bind HRR vectors using circular convolution it is equivalent to creating a structural relationship between vectors in a high-dimensional vector space using the tensor product [15], and then projecting that representation back down to the original dimension at the expense of added compression noise. To state this yet another way, structural patterns were added to the representation while some of the source data was also removed via corruption.

Although HRR compression isn't quite a direct equivalent to dimensional bottlenecks, it does have similarities worth examining. Compression noise from binding reduces the amount of information available to the network for learning, which is the same general reason for reducing the latent dimension. Reducing dimension is an architecture decision though, so it forces this behavior equally for all data. HRR compression noise is deterministic, making information reduction specific to each data sample. Basically, when data vectors are fixed, the noise produced will always be the same. This is not quite what we needed for a bottleneck since it doesn't help the network generalize, it just partially occludes individual data samples in a fixed way.

Fortunately, using a VAE architecture provides us with a ready source of stochasticity. Latent variables are produced by sampling from a distribution. Therefore, if an input vector is observed by the model multiple times it will produce a similar latent vector each time (ideally). However, it is very unlikely to produce the exact same latent vector due to the stochastic nature of the latent space. Re-visiting HRR compression noise as a type of bottleneck is now more reasonable in this context, since reduced information is no longer completely deterministic.

Consider the HVAE model with input vectors $\mathbf{x}$ and $\mathbf{c}$, along with latent vector $\mathbf{t}$ which exists in a stochastic latent space. For each input pair seen we sample a $\mathbf{t}$ vector based on what the network has learned. Then, we perform circular correlation on $\mathbf{c}$ and $\mathbf{t}$ to reconstruct $\mathbf{x}$. Every time a given $\mathbf{x}$ is reconstructed, the compression noise will corrupt different parts of the data since $\mathbf{t}$ was produced via sampling.

The network is forced to learn more efficient representations, since it must learn to compensate for missing data. This has some obvious relations to the training improvement observed when dropout is used, but for our purposes we will consider it as a replacement constraint for the traditional dimensional bottleneck.

With this alternative constraint in place we can justify trying a latent dimension size equal to the input size, which was required to pursue the goal of generating usable latent HRR traces. The most prominent danger when using a large latent dimension is overfitting. The extra capacity may allow the model to memorize training samples rather than generalizing. Empirical testing will easily demonstrate whether our alternative bottleneck constraint performs as intended, since it will severely overfit otherwise. We suspect that the structure added to representations by HRR operations will actually provide additional data pattern features that the network can leverage during training, but investigating this will be left for future work.

## 4.2 Model Details

### 4.2.1 Generative Model

We begin our more formal HVAE model description by first describing our generative model, also referred to as the decoder. The generative model produces reconstructions of source data $\mathbf{x}$ using a latent trace $\mathbf{t}$ and a conditional cue input $\mathbf{c}$. Since all of the variables are continuous, the generator can be described using the conditionally independent probability distribution shown in Fig. 4.1b as follows,

$$p_\theta(\mathbf{x}|\mathbf{c},\mathbf{t}) = \int_{\mathbf{c},\mathbf{t}} p_\theta(\mathbf{x}|\mathbf{c},\mathbf{t}) p_\theta(\mathbf{t}|\mathbf{c}) p_\theta(\mathbf{c}) d\mathbf{c} d\mathbf{t} \tag{4.4}$$

where $\theta$ represents the neural network parameters (e.g., weights and biases) for the encoder. The prior $p_\theta(\mathbf{c})$ is a $\mathcal{N}_{HRR}$ distribution, $p_\theta(\mathbf{t}|\mathbf{c})$ is the conditional prior distribution that will be refined by the recognition model, and $p_\theta(\mathbf{x}|\mathbf{c},\mathbf{t})$ is a deterministic

function of $\mathbf{c}$ and $\mathbf{t}$ which produces a reconstruction of $\mathbf{x}$ as follows,

$$p_\theta(\mathbf{x}|\mathbf{c},\mathbf{t}) = \mathbf{c} \circledast \mathbf{t} \tag{4.5}$$

where $p_\theta(\mathbf{x}|\mathbf{c},\mathbf{t})$ is obtained through circular correlation of a cue and the latent trace. This operation is one of the core elements that sets the HVAE apart from other VAEs. We do not use any fully-connected layers to reconstruct $\mathbf{x}$, which is the common approach. Instead we use circular correlation to provide this capability deterministically. The intuition here is that we wish to encourage $\mathbf{t}$ to act as a valid trace, even outside of the HVAE model. Therefore, if we use HVAE to generate traces, the goal is that they should still be valid in an HRR sense for use in other systems or with manual HRR operations.

If we add fully connected layers to the decoder trace information is then distributed across those layer parameters, and this is not desirable for our purposes. Without those decoder neural network layers, all learning for trace information is forced into the latent space. This encourages valid HRR traces, since loss is calculated directly from the decoded representation.

### 4.2.2 Recognition Model

The recognition model, sometimes referred to as the inference model, or encoder model is the parameterized conditional posterior distribution $q_\phi(\mathbf{t}|\mathbf{c},\mathbf{x})$ used to approximate the true conditional posterior $p_\theta(\mathbf{t}|\mathbf{c},\mathbf{x})$. An MLP with parameters $\phi$ produces the $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$ values needed to draw samples from the distribution via the reparameterization trick described previously.

### 4.2.3 Training

Training the HVAE model amounts to parameter estimation of a directed graphical model using stochastic gradient variational Bayes (SGVB) [27]. The variational

evidence lower bound (ELBO) is characterized as,

$$\log p_\theta(\mathbf{x}|\mathbf{c},\mathbf{t}) \geq \mathbb{E}_{q_\phi(\mathbf{t}|\mathbf{c},\mathbf{x})}\left[\log p_\theta(\mathbf{x},\mathbf{c},\mathbf{t}) - \log q_\phi(\mathbf{t}|\mathbf{c},\mathbf{x})\right] \tag{4.6}$$

which can be re-written as our overall loss function,

$$\mathcal{L} = \mathbb{E}_{q_\phi(\mathbf{t}|\mathbf{c},\mathbf{x})}\left[\log p_\theta(\mathbf{x}|\mathbf{c},\mathbf{t})\right] - D_{KL}(q_\phi(\mathbf{t}|\mathbf{c},\mathbf{x}) \parallel p_\theta(\mathbf{t}|\mathbf{c})) \tag{4.7}$$

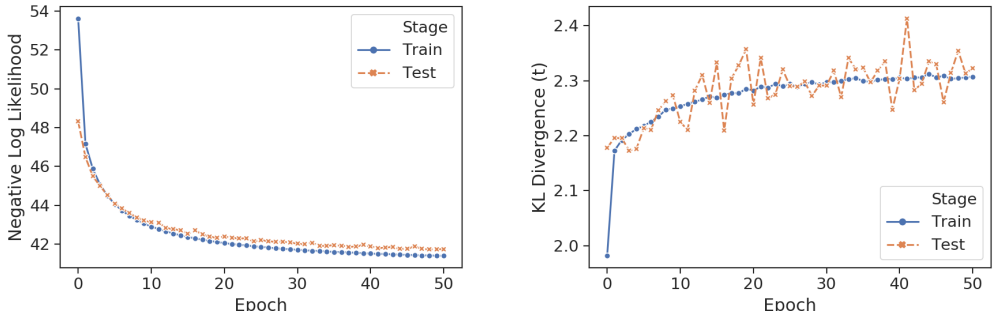
This loss function is the combination of reconstruction error (first term) and a regularizer value (second term), where reconstruction error is measured using negative log-likelihood and regularization is contributed by the Kullback-Leibler (KL) divergence which penalizes approximate posterior distribution divergence from the assumed prior. Since the entire network is differentiable, we can train it with standard gradient descent optimization techniques.

## 4.3 HVAE Experiments

The HVAE model was trained on MNIST to provide a basic demonstration of latent trace learning and data reconstruction. General setup followed the same procedure described in Sec. 3.3. Since HVAE takes cues as a conditional variable input, a cue corresponding to each digit class was generated, so for MNIST ten unique cues were generated and paired with data samples as model inputs. Model reconstructions were also passed through an AECM components for cleanup.

Training was performed for 50 epochs total. Reconstruction loss and KL divergence followed expected patterns demonstrating successful learning of latent traces, as shown in in Fig. 4.2. Visually inspecting samples shows that reconstructions became recognizable after only a few epochs. Samples of raw reconstructions and their

post-cleanup result are provided in Fig. 4.3.




**(a)** MNIST NLL Loss **(b)** MNIST KL Divergence (t)

**Figure 4.2:** Loss plots for training on MNIST over 50 epochs. **4.2a** Negative log likelihood reconstruction loss. **4.2b** KL divergence for latent trace **t**.

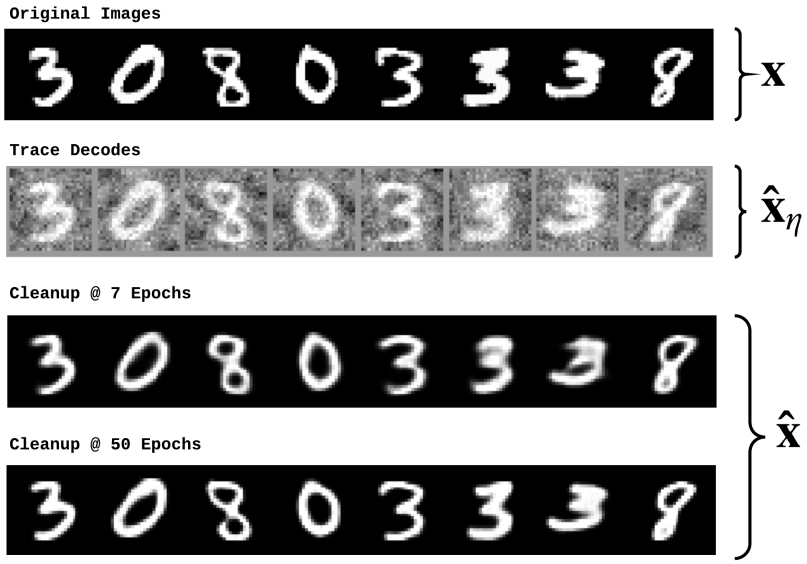



**Figure 4.3:** HVAE results on MNIST, with eight random samples from test set shown. Original images $\mathbf{x}$ (top row), noisy images decoded with circular correlation $\hat{\mathbf{x}}_\eta$ (second row), and AECM cleanup results $\hat{\mathbf{x}}$ at epoch 7 and epoch 50 for comparison (last two rows).

## 4.4 Remarks on HVAE

Adding HRR capability is a natural fit for the VAE design pattern since they are both heavily dependent on assumptions about prior distributions. This combination has

not been explored previously, but we found that these two techniques work well together. Combining these techniques helps mitigate some of the drawbacks associated with each.

One of the primary drawbacks when using VAEs is that the assumed prior distribution used for approximation is often much simpler than the true prior distribution of the data, leading to blurry or inaccurate reconstructions. Fortunately, HRRs provide predictable and well-behaved Gaussian distributions when encoding data into a trace. Using HRR operations with VAEs effectively allows us to impose additional structural constraints on the latent space representation implicitly. Since VAE optimization is dependent on reconstruction error and KL divergence from a prior distribution, adding HRRs assists with both of these objectives.

The base HRR framework can be challenging to work with for practical ML tasks. Management of cues, traces, and cleanup memory requires explicit record keeping, a large number of dot product comparison calculations, or some additional learning system in tasks beyond toy examples. The first two of these options are not scalable, and learning systems for HRRs have been rare historically. We suspect these challenges have discouraged more extensive use of HRRs in the past.

HVAE demonstrates one approach to resolving some HRR implementation barriers by providing a compatible learning system. With latent traces, we can produce HRR components dynamically based on learned input-output relationships. Automating HRR production in this way is much more scalable than the basic approaches. Adding an AECM component described in 3 as a trainable follow-up network even provides an unsupervised cleanup memory.

# Chapter 5

## Holographic Generative Memory (HGMEM)

### 5.1 Holographic Generative Memory Overview

Memory augmentation for neural networks has seen an uptick in interest for tasks where flexible adaptation is critical. Standard gradient-based learning methods show excellent performance when tasks are very specific and a large amount of training data is available. This incremental approach to learning can prove less effective when rapid inference from minimal data is needed, or a task is not narrowly defined.

Standard networks train by re-learning network parameters given whatever data is currently observed. Although trained weight parameters can be considered a type of distributed memory, a network can only use this memory in a limited way. It cannot apply abstract reasoning or value judgments about what information is important to keep and what can be ignored. The network only knows that it needs to adjust weights to optimize for its current data. This limitation leads to classic AI obstacles like abruptly forgetting previously learned information while learning from new data (catastrophic forgetting). Providing neural networks with memory capacity beyond traditional layer weights is one way to address this type of problem.

Interest in augmenting neural networks with memory has seen a notable increase over recent years, but the motivation for using memory augmentation is deeply rooted in the historical notions of what makes AI an interesting problem. The effectiveness

of gradient-based learning methods and the popularity of deep learning has pushed many researchers to focus on incremental optimization for published methods, rather than addressing the more complex but abstract AI problems like conditional behavior, knowledge-based action, or elements of cognition.

Adding interpretable and composable knowledge representation to neural networks moves toward architectures more suited to complex tasks like analogical reasoning or cross-domain transfer learning. We investigate using HRRs for this purpose as a proof of concept that this type of representation is learnable in a more standard neural network architecture. Building a learning system that can generate base HRR components is the first step toward future systems that can make use of the more abstract cognitive capabilities HRRs possess.

The Holographic Generative Memory (HGMEM) model significantly extends the HVAE design presented in Chapter 4. In HGMEM both traces *and* cues are learnable latent variables. Additionally, latent traces are produced using a new generative memory component. We designed this generative memory component to learn an approximation of composing trace vectors via binding and superposition as a byproduct of training. Notably, the HGMEM architecture provides a full-stack HRR system, in the sense that all HRR components are produced by the trainable model.

We denote the HGMEM encoder (recognition model) as $q_\phi$, where any neural network parameters involved are contained in $\boldsymbol{\phi}$. Similarly, the decoder or generative model is $p_\theta$ with parameters $\boldsymbol{\theta}$. Model inputs are data samples, and model outputs are reconstructions of data samples produced via circular correlation with cues and traces. As with HVAE, no additional neural network layers alter the reconstruction output prior to a separately trained AECM module. Finally, loss is calculated using negative log-likelihood as a reconstruction error term, and KL divergence terms for each latent variable to act as regularization. As in previous models, variables are data vectors $\mathbf{x}$, cue vectors $\mathbf{c}$, and trace vectors $\mathbf{t}$, with $\left\{\mathbf{x}, \mathbf{c}, \mathbf{t} \in \mathbb{R}^{N\times 1}\right\}$. Thus far,

this description matches designs we have previously discussed.

Let us examine the first significant place where HGMEM deviates from the previous models. In HVAE, $\mathbf{c}$ is an explicit input used for conditioning while learning the latent traces $\mathbf{t}$. In HGMEM, cues are not an input, but an additional latent variable learned by the network. We still use $\mathbf{c}$ to perform any HRR encoding and decoding needed. However, in HGMEM $\mathbf{c}$ serves a dual purpose. It also acts as an addressing variable used to access content in a memory matrix.

### 5.1.1 Cues in HGMEM

The idea of using an addressing variable to access memory is common in several memory augmented neural network models [9, 13], but the concept lends itself well to HRR operations. In HRRs, cue vectors act as a unique identifier for types of items (e.g., entity, role, concept frame), or a specific instance of a type (e.g., $\text{role}_{gradstudent}$). If we regard the idea of an address generically as a reference to some specific resource, it follows that cues and addresses are very similar in purpose. They both provide a mapping helpful in identifying and accessing specific resources.

This functional similarity to addresses motivated us to use cues for both HRR operations and controlling memory access in our model. This decision carries additional training benefits, since we are forcing the model to optimize for cues that can perform both tasks. Cues must access memory in a useful way, while also correctly decoding traces into data reconstructions. The intuition being that a weighted mapping of content in $M$ is learned based on the value of $\mathbf{c}$. When $\mathbf{x}$ is fed into the network, $\mathbf{c}$ maps to memory content that has previously been useful for reconstructing similar samples.

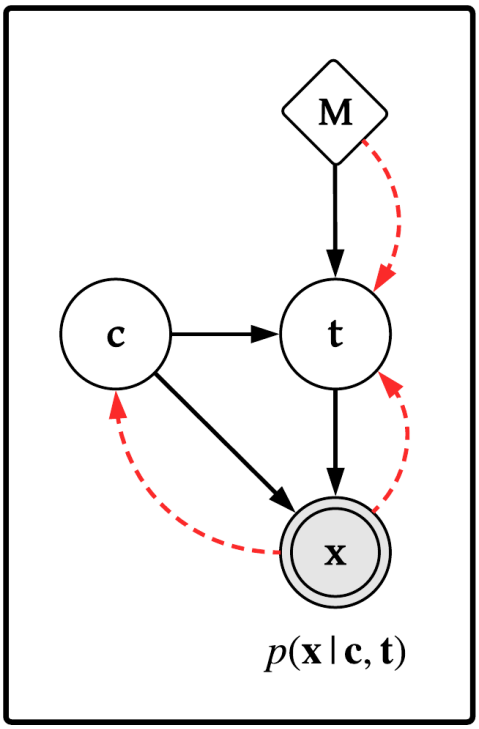



**Figure 5.1:** Graphical sketch of HGMEM model. Red dashed lines indicate approximate inference distributions $q(\cdot|\cdot)$

.

## 5.2 HGMEM Model

### 5.2.1 Generative Model

As with HVAE, the HGMEM model is used to reconstruct $\mathbf{x}$ from latent representations. Since HGEM uses latent representation for both cues and traces, $\mathbf{c}$ and $\mathbf{t}$ are both learned with variational methods. An overall graphical model sketch of HGMEM is shown in Fig.5.1 for reference. First, we define the generative model using joint distribution,

$$p_\theta(\mathbf{x}, \mathbf{c}, \mathbf{t}|\mathbf{M}) = \int_{\mathbf{c},\mathbf{t}} p(\mathbf{x}|\mathbf{c}, \mathbf{t}) p(\mathbf{t}|\mathbf{M}) p(\mathbf{c}) d\mathbf{c} d\mathbf{t} \tag{5.1}$$

where $\mathbf{M}$ is a $K \times N$ matrix with each row initialized randomly using the unitary

HRR normal distribution $\mathcal{N}(1/n, 1/n)$. We do not use an explicit write operation for $\mathbf{M}$, rather it is implemented as a learned memory component used to store latent trace information.

During generation $\mathbf{c}$ is sampled from the prior $p_\theta(\mathbf{c}) \sim \mathcal{N}_{HRR}$, which controls access to $\mathbf{M}$ by acting as an addressing variable. Cue-based addressing transforms $\mathbf{c}$ into the vector $\mathbf{w} \in \mathbb{R}^{K\times 1}$, which is a weighted mapping of content present in the rows of $\mathbf{M}$. This transformation is implemented using a small sub-network $h_A(\mathbf{c})$ as follows,

$$h_{l1}(\mathbf{c}) = f_{l1}(\mathbf{c}^T \cdot \mathbf{W}_{l1} + \boldsymbol{\beta}_{l1}) = \mathbf{b} \in \mathbb{R}^{1\times S} \tag{5.2}$$

$$h_{l2}(\mathbf{b}) = f_{l2}(\mathbf{b} \cdot \mathbf{A}) = \mathbf{w} \in \mathbb{R}^{1\times K} \tag{5.3}$$

and,

$$h_A(\mathbf{c}) = h_{l2}(h_{l1}(\mathbf{c})) = \mathbf{w} \tag{5.4}$$

where these equations describe the two layer addressing sub-network MLP shown in Fig. 5.2c. Subscript $li$ indicates layer index, and layer specific activation functions are $f_{li}(\cdot)$. In $l2$, the layer weights are defined as the addressing weight matrix $\mathbf{A} \in \mathbb{R}^{S\times K}$, where rows are initialized using $\mathcal{N}_{HRR}(1/K, 1/K)$. Note that $l2$ does not contain a bias term, so this layer is equivalent to a dot product followed by an activation function.

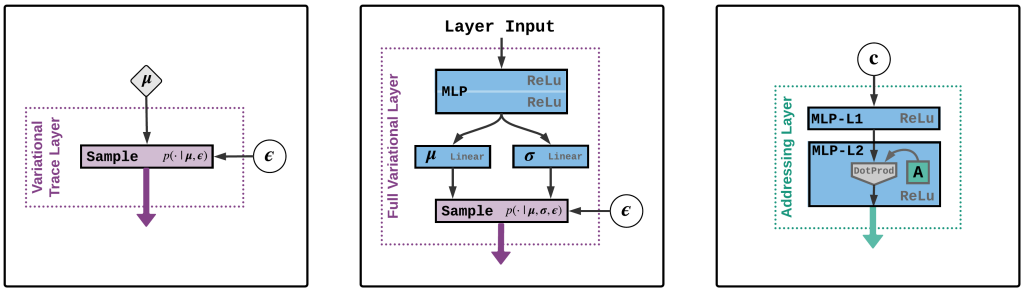
Latent traces are generated using similar variational methods to HVAE. However, we use a memory dependent prior rather than using a $\mathcal{N}_{HRR}$ distribution. Using this alternate prior to generate $\mathbf{t}$ samples is then,

$$p_\theta(\mathbf{t}|\mathbf{M}) = \mathcal{N}(\mathbf{t}|\mathbf{w}^T \cdot \mathbf{M}, 1/\mathbf{n}) \tag{5.5}$$

which indicates that generated samples are parameterized using weighted memory content as the mean, and a fixed unitary HRR variance $1/n$.

Reconstruction of $\mathbf{x}$ with $p_\theta(\mathbf{x}|\mathbf{c},\mathbf{t})$ consists of circular correlation using $\mathbf{c}$ and $\mathbf{t}$, as seen in the HVAE model Eqn.4.5.

### 5.2.2 Recognition Model




**(a)** Variational Trace **(b)** Full Variational **(c)** Addressing

**Figure 5.2:** Breakout diagrams for HGMEM sub-networks. **5.2a** Variational trace layer module used to sample $p_\theta(\mathbf{t}|\mathbf{M})$. **5.2b** Full variational layer module used for portions of the network optimized with KL divergence, $q_\phi(\mathbf{t}|\mathbf{x},\mathbf{M})$ and $q_\phi(\mathbf{c}|\mathbf{x})$. **5.2c** Addressing layer that transforms cues to weighted memory content map.

The HGMEM recognition model uses the factorized approximate posterior distribution,

$$q_\phi(\mathbf{c},\mathbf{t}|\mathbf{x},\mathbf{M}) = \int_{\mathbf{x},\mathbf{M}} q_\phi(\mathbf{t}|\mathbf{x},\mathbf{M}) q_\phi(\mathbf{c}|\mathbf{x}) d\mathbf{x} d\mathbf{M} \tag{5.6}$$

where $q_\phi(\mathbf{c}|\mathbf{x})$ is the parameterized approximate posterior distribution used to learn cues, and $q_\phi(\mathbf{t}|\mathbf{x},\mathbf{M})$ refines the prior distribution $p_\theta(\mathbf{t}|\mathbf{M})$. The overall neural network architecture used to implement HGMEM is illustrated in Fig. 5.3, where this figure demonstrates flow during inference. When generating instead, $\mathbf{c}$ is provided as input to the addressing layer directly, and the variational trace layer output $\mathbf{t}_\theta$ is the generative trace. Decoding proceeds normally using circular correlation with the cue input and generated trace.

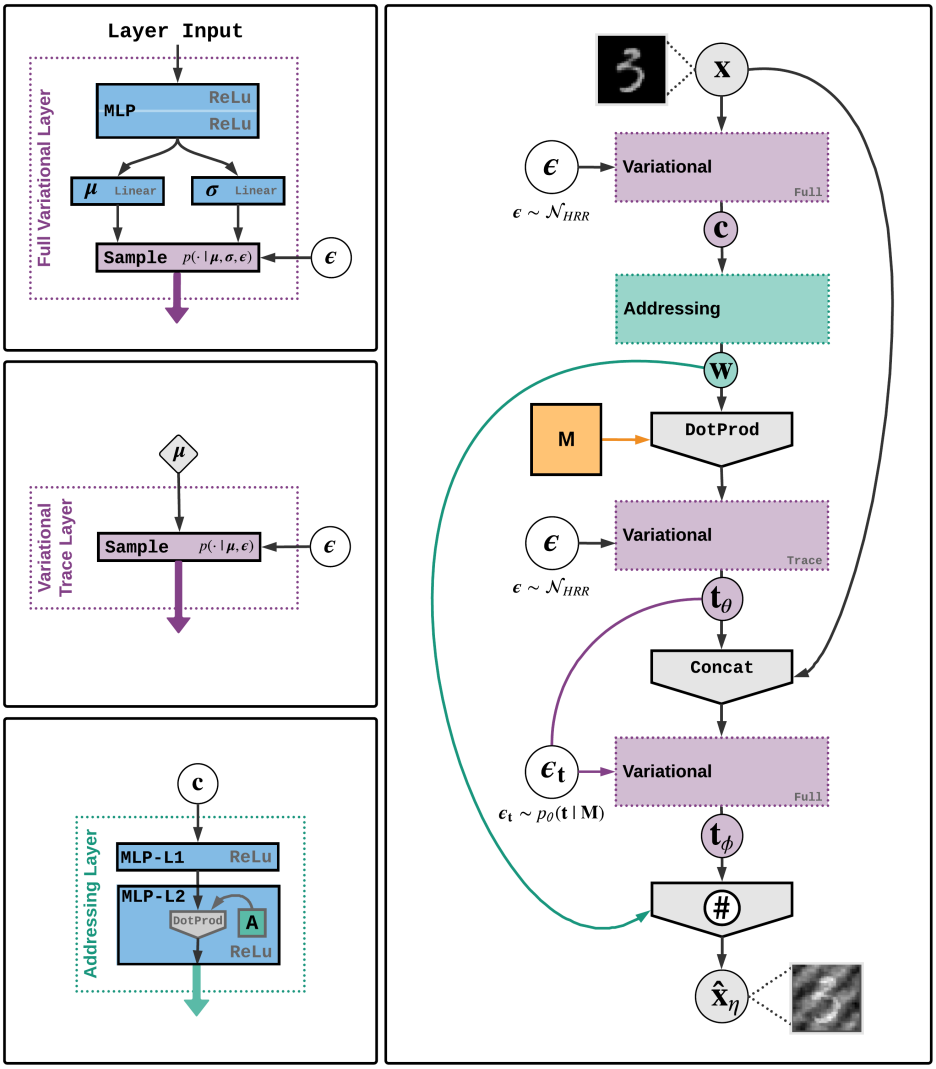



**Figure 5.3:** Full HGMEM neural network architecture implementation. Layers with dotted borders refer to a sub-network, copied here from Fig. 5.2. Diagram represents flow during recognition/inference.

### 5.2.3 Training

The loss function for HGMEM is similar to that used in Eqn. 4.7 for HVAE. Since we use two latent variables, we have a KL divergence term for both. The overall HGMEM loss is as follows,

$$\begin{aligned}\mathcal{L} = \mathbb{E}_{q_\phi(\mathbf{c},\mathbf{t}|\mathbf{x},\mathbf{M})}\left[\log p_\theta(\mathbf{x},\mathbf{c},\mathbf{t}|\mathbf{M})\right] \\ - D_{KL}(q_\phi(\mathbf{c}|\mathbf{x}) \parallel p_\theta(\mathbf{c})) \\ - D_{KL}(q_\phi(\mathbf{t}|\mathbf{x},\mathbf{M}) \parallel p_\theta(\mathbf{t}|\mathbf{M}))\end{aligned} \tag{5.7}$$

## 5.3 HGMEM Experiments

We performed successful preliminary MNIST experiments to test reconstruction with HGMEM, as seen in the previous sections on AECM and HVAE. Decoding results were nearly visually identical to the previously presented examples, so we will omit those here in favor of a more detailed look at components specific to HGMEM. The generative memory component in HGMEM offers some interesting insights into latent trace representation.

### 5.3.1 Examining Latent Traces

In VAEs the latent space is typically some abstract space where information is compressed into more efficient representations. In HGMEM we assume the latent space for $\mathbf{t}$ will take a specific type of representation in the form of traces. Both approaches serve a similar purpose, encoding information into a compressed representation. However, ours imposes some structure in this process via training through circular convolution decoding. Therefore, a visual examination of the latent space was performed to examine if the latent space is actually structured in a manner that we expect, or if it is more akin to traditional VAEs.

We hypothesized that the HGMEM model should learn to use the latent trace memory as an approximation of HRR traces built via superposition. Since we decode only using circular correlation, all trace information is forced into the latent trace

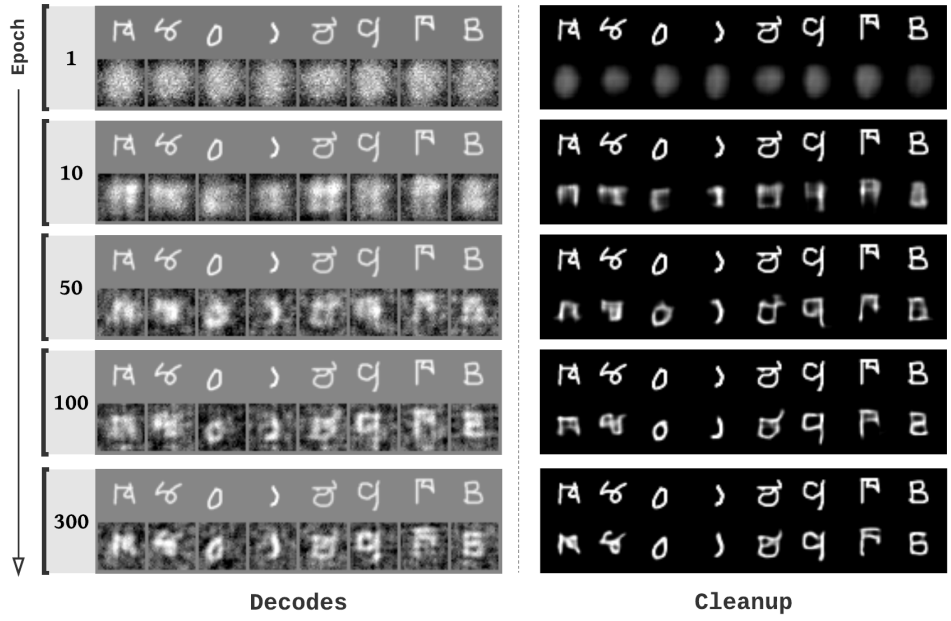



**Figure 5.4:** Example images showing generalization performance for trace decoding (left) and cleanup memory with AECM (right) as training progresses. These images were produced using randomly selected samples from the evaluation set after various epochs. Since samples are not used during training, these examples show reconstruction of in-class (alphabet), but unseen data.

memory representation via training. Our assumption was the visualized random traces should demonstrate structure that indicates superposition of various character components, a sort of multiplexing in the latent space. Circular correlation essentially acts as a selector to retrieve associated data from a trace [25], more specifically, it is designed to perform this function even with traces composed of multiple vectors added together with superposition. With this in mind, it follows that a trace may contain several superimposed character elements but still be decoded successfully with a cue that discriminates well enough.

In a learned memory matrix like that used in HGMEM, we don't have explicit control over how content is store or retrieved. Instead, we guide memory access via training. The training process assists in optimizing efficient storage of information. This is further constrained by decoding only using circular correlation to retrieve

data from memory representations. We also constrain the distribution of the trace representation via KL divergence, encouraging storage in the same basic space, but allowing divergence when necessary. Recall that we uses cues to help address where in memory data is stored, and training refines that process to help similar data samples generate similar cues, leading to similar storage locations. Thus, there are multiple constraints in place which encourage similar data to be stored in a similar location (in a distributional sense).

The intuition here is that constraints encourage memory storage in the same basic space by default, but that space can vary when it is needed to capture data for decoding (e.g., reconstruction loss overcomes the KL penalty). Without some other mechanism, data would likely just be stored in the same space much of the time, leading to highly entangled representations. However, our decoding scheme can actually use cues to access different information stored in the same trace space. When this layered (superimposed) storage still supports decoding, there is no training pressure to move storage elsewhere. Alternatively, when a location does not help with decoding a given sample, training prods the distribution to shift somewhat. We speculate that this leads to a training regime that naturally encourages superposition storage for trace information.

To examine the latent space in this way, we captured visualizations of generated trace decodes using random noise vectors while training progressed. First, we generated 64 random unitary HRR vectors to act as cues, and 64 samples from $\mathcal{N}(0, 1/n)$ as a proxy for data, prior to training. After every epoch, we fed the noise vectors into the generator model and then ran the resulting reconstructions into AECM for cleanup. The goal here was to get some insight into how data is structured in latent trace memory by selecting random elements from it with noise vectors.

Visual examination of our generated samples supported our hypothesis that latent trace memory at least contains an imposed structure from our training method

least from a visual examination. There is obvious structure present, with many substructures that may be reasonably interpreted as superimposed parts of the character data. Additionally, near the beginning of training structures are much more dispersed over the field of 64 samples, which follows since the system has not learned to make use of data for determining storage location yet.

Later in training, we observed structures starting to become more dense within each cell, suggesting further layering of data in that space. Near the end of training the overall field becomes much more sparse, but some individual cells are highly structured. This supports our intuition that training encourages more efficient overlapping storage and that structures tend to cluster together as a result. Fig. 5.5 shows this behavior using some of the sample images produced during our examination.

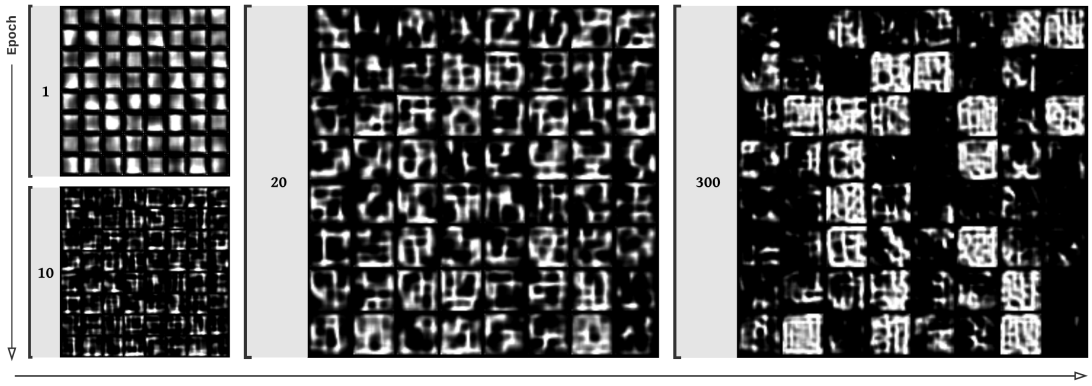



**Figure 5.5:** Visualizing generative latent trace composition structure. Sixty-four random Gaussian noise vectors were created prior to training, then fed as input to the network after various epochs to visualize content stored in latent trace compositions. The space examined shows amorphous shape early in training and progressively learns structure components. As training continues structures are observed to become denser for each sample, and more evenly distributed across all 64 samples. Many structures visually suggest superimposed portions of character data. Toward the end of training structures become more concentrated within individual cells, but sparser across the overall field.

## 5.4 Remarks on HGMEM

Although HGMEM produces reconstructions similar to the HVAE model, the nature of latent trace learning is very different. Using a trainable memory matrix to store information is a familiar approach for memory augmented networks. However, in HGMEM we use this memory component to define the prior distribution when learning latent traces.

Using a memory-dependent prior is key to model function since the KL divergence term for $\mathbf{t}$ penalizes sample divergence from the distribution of memory samples, rather than an arbitrary Gaussian prior. This behavior is observed directly during training, where KL divergence decreases as learned memory content becomes more useful. Essentially, if the memory content does not support reconstruction well, the KL penalty is higher. This value then decreases as the system learns to make better use of memory.

Making latent trace generation dependent on memory encourages richer latent spaces. Enforcing this dependency causes the system to intrinsically optimize for a balance between learned information stored in memory and generative information inferred from variational methods. When building reconstructions the model draws on both sources of information as needed.

HGMEM also represents the first full-stack trainable HRR system, in the sense that it learns to produce cues, traces, and reconstructions all as a product of unsupervised training. This capability is desirable from an HRR practitioner viewpoint since it avoids all of the manual encoding, decoding, and relation logic that we must typically implement explicitly. This model represents a significant step toward bridging the gap between HRRs and modern neural networks.

# Chapter 6

## One-Shot Learning with HGMEM

### 6.1 Overview of OSL with HGMEM

In this section we introduce a neurally inspired OSL mechanism for neural networks that uses uncertainty to modulate learning rate during training. This approach follows some themes from neuroscience source material [1], which shows that causal uncertainty drives OSL in humans. Adapting these ideas to neural networks required a method for obtaining uncertainty measurements during network operation and a strategy for applying those measurements to modulate learning.

In our OSL mechanism we do not use causal uncertainty in the strict sense, but rather an overall predictive uncertainty measurement as a proxy. This is accomplished using Concrete Dropout, which is well-suited to the task since it was specifically designed to provide uncertainty values. This approach was also advantageous because it necessitates variational layers, which we already make extensive use of in HVAE and HGMEM models.

The objective function for CD (Eqn.2.11) is designed to optimize for obtaining a good estimation of epistemic uncertainty via training rather than expensive techniques like a grid-search. Trainable dropout that is part of the network optimization process is also attractive since it allows us to get usable uncertainty measures directly within the network without using an auxiliary training regime to implement learning rate

modulation for OSL.

We use uncertainty measurements to scale a base learning rate dynamically during training. When uncertainty is high the network scales learning rate up in response, but as uncertainty is resolved the rate converges back down toward a natural state determined by network operation. This is advantageous compared to techniques like learning rate scheduling, since the rate is completely adaptive instead of being arbitrarily determined beforehand.

#### 6.1.0.1 Measures for Uncertainty in Neural Networks

When considering uncertainty in the context of neural networks, it is important to first identify what types of uncertainty may be applicable. The value in using two major types of uncertainty has been addressed in the context of deep learning for computer vision [44], where the authors provide a thorough overview of the topic. We will present the relevant ideas here, as they apply to our approach for OSL.

Quantifying uncertainty for neural networks can be considered using two types of uncertainty, epistemic and aleatoric. Epistemic uncertainty refers to the systematic uncertainty present due to factors that are knowable (in principle), but may be unknown in practice due to things like deficient modeling, inaccurate measurements, and obfuscated data. Aleatoric uncertainty is representative of unknowns that can differ every time an experiment is run, like environmental noise independent of the data.

For neural networks epistempic uncertainty amounts to model uncertainty capturing our ignorance about which model (i.e., parameter configuration) is best suited for explaining the data. This is also known as *reducible uncertainty* because it can typically be reduced by providing more data (e.g., via training). This type of uncertainty becomes less useful when a large amount of training data is available, since it is often resolved or explained away in that case. However, for applications where

little training data is available like those used in OSL, this measure can be valuable for recognizing unseen data (e.g., out-of-data examples).

Aleatoric uncertainty relates the uncertainty present due to information not explained directly by the data. It can be further divided into two sub-categories, *heteroscedastic* which is dependent on data, and *homoscedastic* which is task dependant, but not data dependant. Heteroscedastic uncertainty is data dependent because it relates to model inputs, where some may be produce noisier outputs than others. This can be useful for many neural network applications like computer vision. For example, an image of a relatively featureless surface like a wall with little texture would be expected to have a much higher uncertainty than an image of a street scene with a strong depth of field. Homoscedastic uncertainty has less direct application for single-task neural networks since it measures uncertainty we typically cannot reduce in that setting. However, it is useful for multi-task learning problems as measure of task-dependant uncertainty.

Combining both epistemic and aleatoric uncertainty provides a predictive uncertainty value which measures the model's confidence in predicted values, while also accounting for noise it can explain and noise it cannot. When using stochastic regularization techniques like dropout in a network we can allow the model to decrease uncertainty if the dropout probability is not fixed. When a model is able to alter the dropout probability for instance, it can reduce epistemic uncertainty by selecting smaller drop probability values when necessary. This is the general idea behind Concrete Dropout discussed in section 2.5, where drop probability is optimized as part of the standard gradient-based neural network training process.

## 6.2 HGMEM OSL Variant Model

### 6.2.1 Adding Uncertainty to HGMEM

We use Concrete Dropout (CD) as described in section 2.5 to add an uncertainty measurement capability to HGMEM. This is implemented by first wrapping every fully-connected layer with CD functionality, as suggested by the original authors [18]. This alters the overall loss function to include the CD loss term from Eqn. 2.11,

$$\begin{aligned}
\mathcal{L} &= \mathcal{L}_{HGMEM} + \hat{\mathcal{L}}_{MC}(\theta) && (6.1)\\
&= \mathbb{E}_{q_\phi(\mathbf{c},\mathbf{t}|\mathbf{x},\mathbf{M})}\left[\log p_\theta(\mathbf{x},\mathbf{c},\mathbf{t}|\mathbf{M})\right] \\
&\quad - D_{KL}(q_\phi(\mathbf{c}|\mathbf{x}) \parallel p_\theta(\mathbf{c})) \\
&\quad - D_{KL}(q_\phi(\mathbf{t}|\mathbf{x},\mathbf{M}) \parallel p_\theta(\mathbf{t}|\mathbf{M})) \\
&- \quad \frac{1}{M}\sum_{i\in S}\log p(\mathbf{t}_i|\mathbf{f}^{\boldsymbol{\omega}}(\mathbf{x}_i)) \\
&+ \quad \frac{1}{N} D_{KL}(q_\theta(\boldsymbol{\omega}) \parallel p(\boldsymbol{\omega})) && (6.2)
\end{aligned}$$

Actual uncertainty measurements are calculated using the neural network layers in $q_\phi(\mathbf{t}|\mathbf{x},\mathbf{M})$ that produce $\boldsymbol{\mu}_{t_\phi}$ and $\boldsymbol{\sigma}_{t_\phi}$ parameters used when sampling latent traces. This can be approached in several ways depending on design objectives.

Uncertainty can be calculated in a batch-wise manner if sample specific granularity isn't desired, which amounts to calculating the average variance of $\boldsymbol{\mu}_{t_\phi}$ and $\boldsymbol{\sigma}_{t_\phi}$ network outputs across all batch samples, as follows,

$$\mathbf{u}_e = \frac{1}{K}\sum_{k\in K} Var(\boldsymbol{\mu}_k) \tag{6.3}$$
$$\mathbf{u}_a = \frac{1}{K}\sum_{k\in K} Var(\boldsymbol{\sigma}_k)$$
$$\mathbf{u} = \psi[\mathbf{u}_e + \mathbf{u}_a]$$

where $K$ is the batch size, $\mathbf{u}_e$ is epistemic uncertainty, $\mathbf{u}_a$ is aleatoric uncertainty, $\mathbf{u}$ is predictive uncertainty, which includes an importance scalar value $\psi$. In our model, $\psi$ plays a similar role to saliency in [1]. Since the dot product is the most common way to evaluate HRR similarity, we use this to compare original inputs and reconstructions as our importance value,

$$\psi_i = \mathbf{x}_i^T \cdot \hat{\mathbf{x}}_{\eta i} \tag{6.4}$$
$$\tag{6.5}$$

When sample-specific uncertainty is desired, Monte Carlo (MC) style sampling can be used to obtain variance over multiple predictions on the same input and using the average variance for those samples in Eqn. 6.3. This is easily achieved by just repeatedly computing $q_\phi(\mathbf{t}|\mathbf{x}, \mathbf{M})$ for the same $\mathbf{x}$, since the dropout mechanism and stochastic sampling in the network will produce some degree of variance in predictions. When the network is more confident in its prediction for a given input, the sample-specific variance will be lower, leading to decreased uncertainty. This method does provide much more dynamic learning rate adaptation, since uncertainty is determined for every sample before being aggregated for the batch, but it does introduce additional computational costs.

### 6.2.2 Scaling Learning Rate for OSL via Uncertainty

Following the concept presented in [1], our OSL mechanism consists of scaling the learning rate used during training dynamically based on uncertainty observed. This scaling term is denoted $\gamma$, as follows,

$$\gamma = max(\frac{exp(\tau Var(\mathbf{u}|\mathbf{x}))}{\Sigma_j exp(\tau Var(\mathbf{u}|\mathbf{x}))}) \tag{6.6}$$

where $\mathbf{u}$ is the predictive uncertainty from Eqn. 6.4, and $\tau$ is a temperature parameter scaling the impact of the importance value $\psi$.

Prior to calculating network updates in each training batch, we calculate the current learning rate $\alpha$ by scaling the base learning rate $\alpha_0$ with $\gamma$,

$$\alpha = \alpha_0 \exp(\beta\gamma) \tag{6.7}$$

where $\beta$ is an optional hyper-parameter used to tune the learning rate scaling level.

### 6.2.3 Classification Sub-Network

Our OSL mechanism is mostly agnostic to the classification method used, since it modulates overall learning rate and is not tied to a specific classifier implementation. For experiments in this work, we adapt a matching model [8] for classification since it can be added as a sub-network without altering the HGMEM-OSL architecture. This matching model has the general form,

$$\hat{y} = \sum_{i=1}^{k} a(\hat{x}, x_i) y_i \tag{6.8}$$

where $x_i, y_i$ are samples and one-hot labels generated from a support set $S = (x_i, y_i)_{i=1}^{k}$, and $a$ is an attention mechanism. We us simple softmax over cosine distance function

for $a$,

$$a(\hat{x}, x_i) = \frac{\exp\left[c(f(\hat{x}), g(x_i))\right]}{\sum_{j=1}^{k} \exp\left[c(f(\hat{x}), g(x_j))\right]} \tag{6.9}$$

where $c$ is the cosine distance, and $f, g$ are functions used to create embeddings from $\hat{x}, x_i$.

With this general approach, we must choose an embedding strategy that provides useful features for classification. We perform matching classification on cue and data vectors reconstructed from traces. Since cues are used to address learned memory in HGMEM, training encourages generation of similar cues for similar inputs. This native similarity is helpful for classification when features are extracted from cue vectors during embedding. Creating embeddings from cues generated directly by $q_\phi(\mathbf{c}|\mathbf{x})$ is one option, but HRRs offer an additional way to use obtain cues in this context.

We can reconstruct cues from latent traces generated by the model. We haven't previously discussed reconstructing cues, so some explanation is needed. When traces are created using two vectors (e.g., $\mathbf{c}$, $\mathbf{x}$), circular correlation can reconstruct either vector from that trace. We use this property to obtain $\sim \mathbf{c}$, a cue vector reconstructed from $\mathbf{x}$ and $\mathbf{t}$ as follows,

$$\sim \mathbf{c} = \mathbf{x} \circledast \mathbf{t} \tag{6.10}$$

Embedding a reconstructed cue provides additional discriminative information for classification since memory and latent trace layers contribute information through the trace used.

We concatenate reconstructed cues and decoded trace vectors when obtaining embeddings from the functions $f, g$. This combination provides an embedding with features from a direct sample reconstruction $\sim \mathbf{x}_\theta$, and added features from the cue reconstruction $\sim \mathbf{c}_\phi$ to assist with classification. The HGMEM architecture provides

all the necessary components for reconstructing cues and data vectors, so we obtain embeddings using existing HGMEM layers and a small CNN sub-network that extracts features as follows,

$$f(\mathbf{x}) = CNN(concat(\sim \mathbf{x}_\theta, \sim \mathbf{c}_\phi)) \tag{6.11}$$

where,

$$\sim \mathbf{x}_\theta = p_\theta(\mathbf{x}|\mathbf{c}, \mathbf{t})$$

$$\sim \mathbf{c}_\phi = \mathbf{x} \circledast q_\phi(\mathbf{t}|\mathbf{x}, \mathbf{M})$$

and,

$$f(\mathbf{x}) = g(\mathbf{x})$$

where $CNN$ is a convolutional neural network with a stack of four modules, with each module consisting of a 3x3 convolutional layer with 64 filters and 2x2 max pooling. The same CNN sub-network is used for both $f$ and $g$.

During training, embeddings for $\hat{\mathbf{x}}$ and $\mathbf{x}_i$ are created via $f$ and the matching model is used to predict labels. A categorical cross-entropy loss term is added to the overall loss function to enable training of the embedding sub-network during normal network optimization.

## 6.3 HGMEM-OSL Experiments

We performed OSL classification experiments to evaluate whether our HGMEM-OSL model is capable of learning in this setting. These experiments were intended to demonstrate the baseline capability of our approach, so no hyper-parameter tuning or additional algorithmic tricks were used beyond what the base model configuration employs.

### 6.3.1 Omniglot Dataset

The Omniglot dataset [45] is very common in OSL literature. It consists of 1623 handwritten characters from 50 different alphabets, for examples see Fig. 6.1. Only 20 samples for each character type are provided, a relatively small amount compared to the number of characters (i.e., classes). This ratio of samples to classes has led to Omniglot often being referred to as the "transpose" of MNIST, since MNIST contains many samples for only a few classes. Omniglot also serves a similar functional purpose to MNIST for OSL research since it is used as a baseline benchmark set for nearly all OSL techniques.

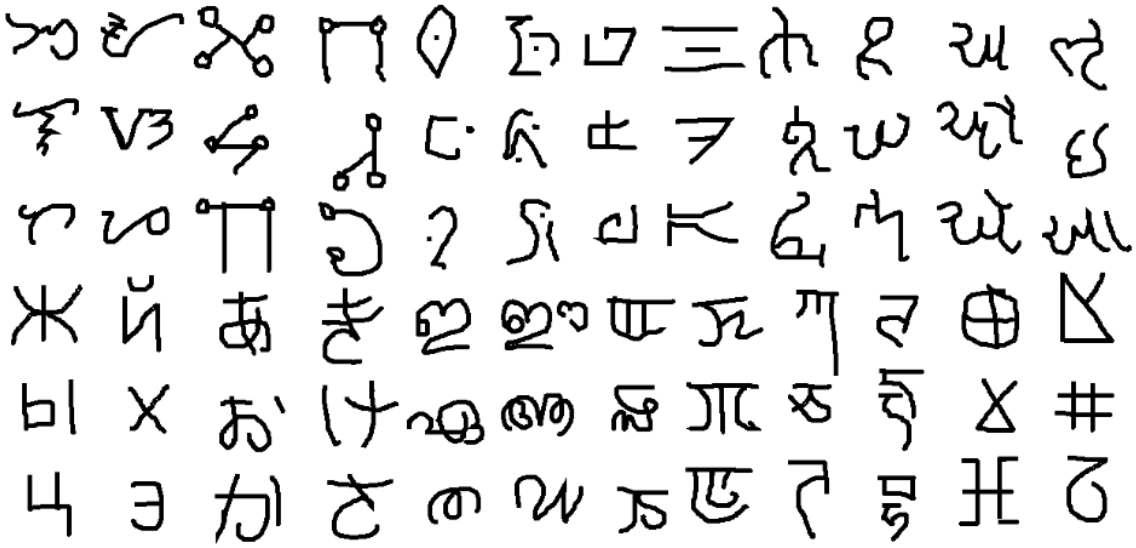


**Figure 6.1:** Character samples from Omniglot dataset. Original figure from [2]

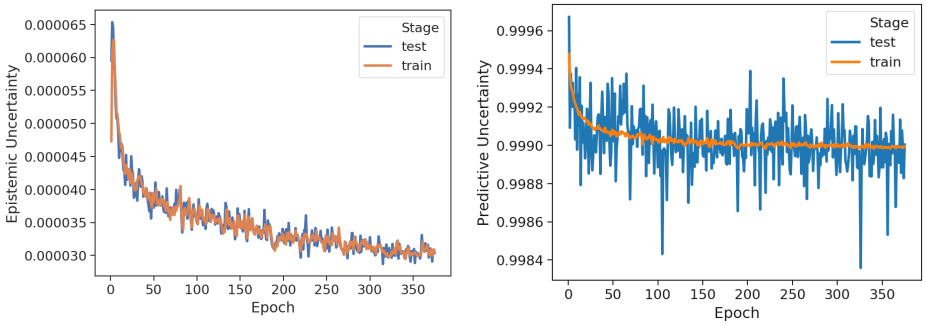
.

Experimental setup for Omniglot varies somewhat in the literature. However, the general form used is $N$-way, $K$-shot tasks. In these tasks we choose $N$ unseen character classes, independent of alphabet. A support set $S$ is generated by choosing $K$ disjoint samples for each of the $N$ classes. The goal is to classify a query sample $x \notin S$, chosen from one of the $N$ classes. Classification is performed by comparing model features produced for $x$ with those produced by samples in $S$ using some similarity metric (e.g., dot product, cosine distance). The class of the support sample

scoring the highest similarity is returned as the predicted class for $x$.

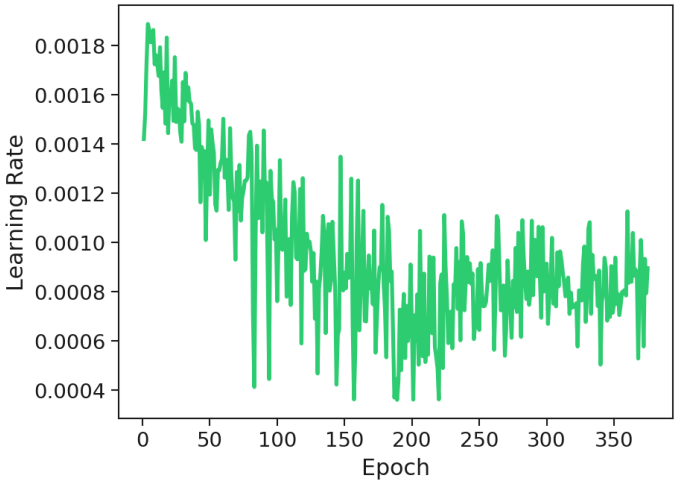
### 6.3.2 Omniglot Classification Results

HGMEM-OSL was evaluated on standard Omniglot classification tasks with two models, one trained and configured for 5-way and another for 20-way. Models were evaluated with 1-shot and 5-shot tests on unseen samples. Some preliminary testing with the OSL mechanism helped select hyper-parameter values that offered consistent learning rate modulation during training, motivating us to use $\tau = 2048.0$ and $\beta = 4.0$ for experiments. No data augmentation was used in any experiments.




**(a)** Epistemic uncertainty.

**(b)** Predictive uncertainty.




**(c)** Modulated learning rate.

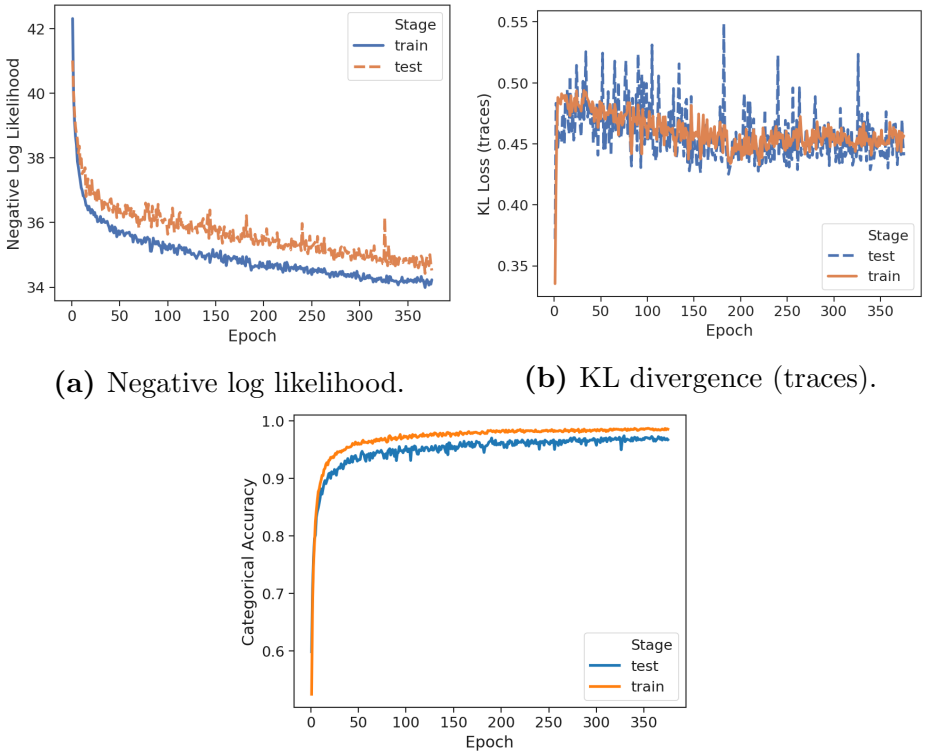
**Figure 6.2:** Omniglot OSL uncertainty and learning rate plots during training of 5-way, 1-shot model. **6.2a**: Epistemic (reducible) uncertainty decreases as training progresses. **6.2b**: Predictive uncertainty decreases as recognizable reconstructions are learned. **6.2c**: Learning rate modulated dynamically by uncertainty measurements.

We observed the learning rate modulation to behave as expected, as shown in Figs. 6.2. Epsitemic uncertainty decreased as training progressed, and aleatoric uncertainty settled near 1 after the network learned to account for it. During early training epochs reconstructions are less recognizable, causing high uncertainty. During training uncertainty reduces quickly until reconstructions are more accurate, at that point uncertainty increases slightly as the network attempts to refine predictions. This was a general pattern we observed in all configurations tested during preliminary tests and this experiment.

Reconstruction loss and KL divergence for traces behaved as expected, as shown in Fig. 6.3. KL was higher during early training passes, and then decreased as the system learned to make better use of memory. Accuracy results were lower than expected, when compared to other methods in literature, shown in Table 6.1. This result is not ideal, but it does show our baseline approach is functional for OSL, even without hyper-parameter optimization.

| | | | Classification Accuracy | | | |
|---|---|---|---|---|---|---|
| **Model** | **Aug** | **Tuned** | **5w-1s** | **5w-5s** | **20w-1s** | **20w-5s** |
| **Pixel Distance** [8] | Y | N | 41.70% | 63.20% | 26.70% | 42.60% |
| **MoVAE** [46] | Y | N | 90.90% ± 5.4 | 96.70% ± 2.8 | - | - |
| **Convolutional Siamese Net** [47] | Y | N | - | - | 92.00% | - |
| **MANN-GE** [12] | Y | N | 97.40% | 98.90% | 92.30% | 98.40% |
| **Matching Networks** [8] | Y | Y | 97.90% | 98.70% | 93.50% | 98.70% |
| **Matching Networks** [8] | Y | N | 98.10% | 98.90% | 93.80% | 98.50% |
| **Prototypical Networks** [48] | Y | N | 98.80% | 99.70% | 96.00% | 98.90% |
| **HGMEM-OSL (Ours), Max** | N | N | 97.33% | 99.31% | 91.85% | 97.00% |
| **HGMEM-OSL (Ours), Average** | N | N | 97.00% ± 0.19 | 99.08% ± .09 | 91.07% ± .32% | 96.74% ± .18% |

**Table 6.1:** Omniglot accuracy comparisons to those reported in literature. Results for 5-way 1-shot, 5-way 5-shot, 20-way 1-shot, and 20-way 5-shot classification on Omniglot. Our reported average results are for 20 test replications.




**(a)** Negative log likelihood.

**(b)** KL divergence (traces).

**(c)** Classification Accuracy

**Figure 6.3:** Omniglot OSL loss and accuracy plots plots during training of 5-way, 1-shot model. **6.3a**: Negative log likelihood as reconstruction error loss. **6.3b**: KL divergence for memory dependent latent traces. **6.3c**: Classification accuracy on Omniglot (5-way, 1-shot).

## 6.4 Remarks on OSL with HGMEM

Classification accuracy results did not beat SOTA in our experiments. However, our primary goal in this work was to develop these new techniques and demonstrate some baseline performance as a proof of concept. The accuracy we did achieve during testing is significantly higher than random guessing or simpler systems like pixel distance. Additionally, the learning rate modulation approach worked exactly as designed, which is promising. Overall, these results demonstrate that this approach is capable of OSL, but the specific implementation needs to be optimized in future work.

We suspect that performance was limited by a few factors in the experiments

here. The first is design choice when adding a classification component. HGMEM offers several potential ways to extract features for classification. Reconstructed vectors, cues, memory content, latent traces, and other intermediary activations are all potential candidates to focus on for feature extraction.

After some preliminary testing, we chose to use the reconstructed cue and data vector approach. The reasoning was that cues generated for similar data samples will also be similar since they are trained to access memory in a data dependent way. It seemed reasonable that using cues as a lift for decoded data samples would assist with classification due to implicit cue similarity. Preliminary tests showed reconstructed cues or data vectors alone produced reasonable classification performance, but combining them via concatenation yielded better results.

Using a matching model classification approach was effective, but an alternate classification method is worth investigating. While the HGMEM architecture learned to reconstruct samples very quickly (e.g. 50 to 100 epochs), the matching model sub-network added extensive additional training and testing time since all support samples required reconstruction. This added time limited our reported results in 6.1 somewhat. At reporting time performance increase was small between epochs, but training and testing loss values were still decreasing. With additional training time accuracy could likely have been increased further, but scheduling and resource limitations necessitated an earlier stop for reporting. Exploring alternative ways to leverage the features provided by HGMEM in future work may provide a more efficient classification sub-network.

Extended training and testing time also limited tuning of hyper-parameter values, preventing a thorough exploration of the parameter space via grid search or similar methods. Our OSL mechanism did perform exactly as expected using our chosen hyper-parameter values, with $\tau$ values showing a notable impact on scaling for learning rate. Overall, we found that networks showed higher accuracy values and more

rapid learning with larger $\tau$ values. Since $\tau$ helps to amplify the impact of network uncertainty and reconstruction importance values (i.e., $\psi$), this behavior is expected. We suspect that further experiments on hyper-parameter tuning for our mechanism with a more exhaustive tuning technique may significantly increase performance.

We find these initial OSL outcomes very encouraging, given the performance achieved with minimal tuning and no data augmentation.

# Chapter 7

## Final Remarks

### 7.1 Summary of Work

In this work we demonstrated that neurally inspired techniques for knowledge representation and one-shot learning can be applied to contemporary neural network architectures successfully. The AECM, HVAE, and HGMEM models we've introduced show a path toward interpretable and composable learned knowledge representations that don't require the cumbersome manipulation and indexing associated with historical methods for VSAs. Our OSL mechanism provides a new approach to the problem with uncertainty-based learning rate modulation rather than novelty driven OSL.

Experimental results for autoencoding show that our models learn valid HRR representations in an unsupervised manner while also providing a solution to the cleanup memory problem, reducing practical obstacles that hinder more complex work with HRRs. We demonstrated that using latent HRR representations in a VAE model is also a beneficial combination since the strengths of each technique helps mitigate traditional drawbacks in the other.

OSL experiments demonstrate initial accuracy results approaching, but not exceeding SOTA methods in literature. These results were obtained using very little hyper-parameter tuning or optimization due to computational and time constraints, and given additional time and resources for tuning these results can likely be im-

proved further. We have shown that uncertainty-based OSL is a valid approach with neural networks, and one that offers significant potential for further exploration.

In a more general sense, the work presented here supports the idea that using neuroscience for inspiration in ML research can lead to unexpected insights. We were primarily motivated to pursue the initial exploration of two neurally inspired ideas, uncertainty-based OSL and compositional knowledge representation. In this work we have demonstrated that both of these ideas can be readily adapted to contemporary neural network architectures, and we look forward to expanding on these ideas in future work.

## 7.2 Future Work

The first follow-up to this initial proof of concept work should be a more detailed investigation into optimizing the OSL mechanism for tasks like Omniglot classification (e.g., hyper-parameter tuning). While we were able to show reasonable results, they weren't SOTA. Making these new models and mechanisms functional was a long process, leaving little time for exploring optimization. Further work on optimization should provide a more comprehensive view of how our OSL mechanism compares to others in literature.

Our OSL mechanism was originally designed to work with the HGMEM architecture, but the final technique we developed is actually fairly agnostic to the underlying neural network architecture used. Most architectures capable of supporting concrete dropout that also contain a variational layer should be compatible with our OSL approach. Applying our OSL mechanism to other models from literature and comparing results would provide a good indicator of how generally applicable it is.

Combining HRRs and VAEs is representative of a new family of models with potential for much deeper exploration. In our work we focused on proving models capable of performing the core HRR operations (e.g., encoding, decoding, composi-

tion). HRRs offer many other complex features well beyond these fundamental ones though. Plate's work provides extensive examples using them to represent complex structures like hierarchies or sequences. Additionally, they support representation of more abstract representations like subject-predicate relationships, analogies, concept frames, and more. Our trainable full-stack HRR model provides the groundwork needed to explore implementing these ideas in areas like reinforcement learning or meta-learning.

We only experimented with grayscale images to simplify trace representation during this initial work. Adapting our models to support multi-channel images poses an interesting challenge, since multiple vectors (e.g., channels) are used in a single data sample. Learning a latent trace that encodes information from all channels is a good introductory problem for exploring more advanced HRR structural encoding. Some potential strategies are encoding channel vectors as a sequence structured HRRs, chunking, or using unique channel id cues. Similarly, the same HRR structural encoding techniques can be used to apply our models in other domains like NLP. There are several ways to approach these tasks that are worth investigating in an effort to expand model capability.

## 7.3 Notes on HRR Value

The value added by a full-stack HRR system like we developed with HGMEM may not be readily apparent to readers without practical experience using HRRs. Recall that the HRR framework offers a set of tools to systematically represent structure and data in an interpretable and composable way. This capability is very interesting from an AI standpoint, since compositional knowledge is a cornerstone of human intelligence that we find difficult to replicate in machines.

Providing some context, HRRs were originally developed in an era before deep learning. During this time much of the dialogue in AI was centered on whether

connectionist systems could actually do anything useful, and useful was largely defined by the dominant AI research focus on symbolic processing or cognitive tasks. This environment shaped much of the work done with HRRs, since they were partially developed in response to the criticisms of connectionist systems at the time. This caused HRR applications in literature to often take the form of symbolic or cognitive tasks, which are unfamiliar to many modern ML researchers.

Providing a full-stack HRR model like we do in this work helps showcase the framework in a more modern context that is more compatible with current research trends. Hopefully, this will motivate further work in this direction. An eventual goal might be bringing research back around to the original HRR cognitive task ideas, but with architectures that take full advantage of advances in ML.

From an engineering standpoint, a full-stack HRR model is simply more usable than the basic HRR approach. Nearly all previous HRR works focus on learning a single HRR component in a supervised setting, or they require hand-crafted components relying heavily on a-priori user knowledge (e.g., manually partitioning and combining datasets by class before encoding). A model that automates many of the HRR operations helps make them more generally applicable to a wider range of tasks.

# Bibliography


[1] S. W. Lee, J. P. O'Doherty, and S. Shimojo, "Neural Computations Mediating One-Shot Learning in the Human Brain," *PLoS Biology*, vol. 13, no. 4, pp. 1–36, 2015.

[2] B. M. Lake, R. R. Salakhutdinov, J. Gross, and J. B. Tenenbaum, "One shot learning of simple visual concepts," *Proceedings of the 33rd Annual Conference of the Cognitive Science Society (CogSci 2011)*, vol. 172, pp. 2568–2573, 2011. [Online]. Available: http://palm.mindmodeling.org/cogsci2011/papers/0601/paper0601.pdf

[3] R. C. O'Reilly, D. Wyatte, S. Herd, B. Mingus, and D. J. Jilk, "Recurrent processing during object recognition," *Frontiers in Psychology*, vol. 4, no. APR, pp. 171–180, 2013.

[4] L. Fei-Fei, R. Fergus, and P. Perona, "One-shot learning of object categories," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 28, no. 4, pp. 594–611, 2006.

[5] L. Fei-Fei, "Knowledge transfer in learning to recognize visual objects classes," *Proceedings of the Fifth International Conference ...*, 2006. [Online]. Available: http://www-cs.stanford.edu/groups/vision/documents/Fei-Fei{_}ICDL2006.pdf

[6] C. G. Atkeson, A. W. Moorey, S. Schaalz, A. W. Moore, and S. Schaal, "Locally Weighted Learning," *Artificial Intelligence*, vol. 11, pp. 11–73, 1997. [Online]. Available: http://www.springerlink.com/index/G8280541763Q0223.pdf

[7] J. Goldberger, S. T. Roweis, G. E. Hinton, R. Salakhutdinov, S. T. Roweis, and R. Salakhutdinov, "Neighbourhood Components Analysis," *Advances in Neural Information Processing Systems*, pp. 513–520, 2004. [Online]. Available: http://www.cs.toronto.edu/{~}fritz/absps/nca.pdfhttp://eprints.pascal-network.org/archive/00001570/

[8] O. Vinyals, C. Blundell, T. Lillicrap, K. Kavukcuoglu, and D. Wierstra, "Matching Networks for One Shot Learning," *arXiv*, pp. 1–12, 2016. [Online]. Available: http://arxiv.org/abs/1606.04080

[9] J. Bornschein, A. Mnih, D. Zoran, and D. J. Rezende, "Variational Memory Addressing in Generative Models," 2017. [Online]. Available: http://arxiv.org/abs/1709.07116

[10] A. Graves, G. Wayne, and I. Danihelka, "Neural Turing Machines," pp. 1–26, 2014. [Online]. Available: http://arxiv.org/abs/1410.5401

[11] A. Santoro, S. Bartunov, M. Botvinick, D. Wierstra, and T. Lillicrap, "One-shot Learning with Memory-Augmented Neural Networks," 2016.

[12] H. Tseran and T. Harada, "Memory Augmented Neural Network with Gaussian Embeddings for One-Shot Learning," no. Nips, pp. 1–5, 2017.

[13] Y. Wu, G. Wayne, A. Graves, and T. Lillicrap, "The Kanerva Machine: A Generative Distributed Memory," in *ICLR*, apr 2018, pp. 1–16. [Online]. Available: http://arxiv.org/abs/1804.01756

[14] R. W. Gayler, "Holographic networks are hiking the foothills of analogy," *Neural Computing Surveys*, vol. 2, no. 1, pp. 6–7, 1999.

[15] P. Smolensky, "Tensor product variable binding and the representation of symbolic structures in connectionist systems," *Artificial Intelligence*, vol. 46, no. 1-2, pp. 159–216, 1990.

[16] P. Kanerva, *Sparse Distributed Memory*. MIT Press, 1988.

[17] T. A. Plate, "Holographic Reduced Representations," *IEEE Transactions on Neural Networks*, vol. 6, no. 3, pp. 623–641, 1995.

[18] Y. Gal, J. Hron, and A. Kendall, "Concrete Dropout," 2017. [Online]. Available: http://arxiv.org/abs/1705.07832

[19] A. Newell, J. C. Shaw, and H. A. Simon, "Report on a general problem-solving program," *IFIP Congress*, vol. 256, p. 64, 1959.

[20] L. Ferrone and F. M. Zanzotto, "Symbolic, Distributed and Distributional Representations for Natural Language Processing in the Era of Deep Learning: a Survey," 2017. [Online]. Available: http://arxiv.org/abs/1702.00764

[21] G. E. Hinton, J. L. McClelland, and D. E. Rumelhart, "Distributed representations," *Parallel Distributed Processing*, pp. 77–109, 1986.

[22] T. Plate, *Holographic Reduced Representations: Distributed Representations for Cognitive Structures*. Center for the Study of Language and Inf, 2003.

[23] N. Chomsky, "Aspects of the Theory of Syntax," 1967.

[24] J. Neumann, "Learning the systematic transformation of holographic reduced representations," *Cognitive Systems Research*, vol. 3, no. 2, pp. 227–235, 2002.

[25] M. Nickel, L. Rosasco, and T. Poggio, "Holographic Embeddings of Knowledge Graphs," pp. 1955–1961, 2015. [Online]. Available: http://arxiv.org/abs/1510.04935

[26] M. Kelly, "Advancing the Theory and Utility of Holographic Reduced Representations," *Queen's University, Canada - ProQuest Dissertations and Theses*, 2014.

[27] D. P. Kingma and M. Welling, "Auto-Encoding Variational Bayes," no. Ml, pp. 1–14, 2013. [Online]. Available: http://arxiv.org/abs/1312.6114

[28] Y. Gal and Z. Ghahramani, "Dropout as a Bayesian Approximation: Representing Model Uncertainty in Deep Learning," 2015. [Online]. Available: http://arxiv.org/abs/1506.02142

[29] G. Kahn, A. Villaflor, V. Pong, P. Abbeel, and S. Levine, "Uncertainty-Aware Reinforcement Learning for Collision Avoidance," 2017. [Online]. Available: http://arxiv.org/abs/1702.01182

[30] Y. Gal, R. T. Mcallister, and C. E. Rasmussen, "Improving PILCO with Bayesian Neural Network Dynamics Models," *Data-Efficient Machine Learning Workshop, ICML*, pp. 1–7, 2016. [Online]. Available: http://mlg.eng.cam.ac.uk/yarin/PDFs/DeepPILCO.pdf

[31] Y. Li and Y. Gal, "Dropout Inference in Bayesian Neural Networks with Alpha-divergences," in *International Conference on Machine Learning*, 2017, pp. 2052–2061. [Online]. Available: http://arxiv.org/abs/1703.02914

[32] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A Simple Way to Prevent Neural Networks from Overfitting," *Journal of Machine Learning Research*, vol. 15, pp. 1929–1958, 2014. [Online]. Available: https://www.cs.toronto.edu/{~}hinton/absps/JMLRdropout.pdf

[33] C. E. Rasmussen and C. K. I. Williams, *Gaussian Processes for Machine Learning (Adaptive Computation and Machine Learning)*. The MIT Press, 2005.

[34] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[35] T. C. Stewart, Y. Tang, and C. Eliasmith, "A biologically realistic cleanup memory: Autoassociation in spiking neurons," *Cognitive Systems Research*, vol. 12, no. 2, pp. 84–92, 2011. [Online]. Available: http://dx.doi.org/10.1016/j.cogsys.2010.06.006

[36] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2323, 1998.

[37] H. Xiao, K. Rasul, and R. Vollgraf, "Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms," pp. 1–6, 2017. [Online]. Available: http://arxiv.org/abs/1708.07747

[38] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," pp. 1–15, 2014. [Online]. Available: http://arxiv.org/abs/1412.6980

[39] I. Danihelka, G. Wayne, B. Uria, N. Kalchbrenner, and A. Graves, "Associative Long Short-Term Memory," 2016. [Online]. Available: http://arxiv.org/abs/1602.03032

[40] M. A. Zinkevich and A. Davies, "Holographic Feature Representations of Deep Networks," *Proceedings of UAI*, p. 267, 2017.

[41] K. Sohn, H. Lee, and X. Yan, "Learning structured output representation using deep conditional generative models," in *Advances in Neural Information Processing Systems 28*, C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, Eds. Curran Associates, Inc., 2015, pp. 3483–3491. [Online]. Available: http://papers.nips.cc/paper/5775-learning-structured-output-representation-using-deep-conditional-generative-models.pdf

[42] I. Higgins, L. Matthey, A. Pal, C. Burgess, X. Glorot, M. Botvinick, S. Mohamed, and A. Lerchner, "beta-vae: Learning basic visual concepts with a constrained variational framework," 2016.

[43] C. P. Burgess, I. Higgins, A. Pal, L. Matthey, N. Watters, G. Desjardins, and A. Lerchner, "Understanding disentangling in beta-VAE," no. Nips, 2018. [Online]. Available: http://arxiv.org/abs/1804.03599

[44] A. Kendall and Y. Gal, "What Uncertainties Do We Need in Bayesian Deep Learning for Computer Vision?" mar 2017. [Online]. Available: http://arxiv.org/abs/1703.04977

[45] B. M. Lake, R. Salakhutdinov, and J. B. Tenenbaum, "Human-level concept learning through probabilistic program induction," *Science*, vol. 350, no. 6266, pp. 1332–1338, 2015.

[46] D. C. Mocanu and E. Mocanu, "One-Shot Learning using Mixture of Variational Autoencoders: a Generalization Learning approach," no. July, pp. 10–15, 2018. [Online]. Available: http://arxiv.org/abs/1804.07645

[47] G. Koch, R. Zemel, and R. Salakhutdinov, "Siamese Neural Networks for One-shot Image Recognition." [Online]. Available: http://www.cs.toronto.edu/{~}rsalakhu/papers/oneshot1.pdf

[48] J. Snell, K. Swersky, and R. S. Zemel, "Prototypical Networks for Few-shot Learning," 2017. [Online]. Available: http://arxiv.org/abs/1703.05175