Continuous Spatial and Temporal Representations in Machine Vision

by

Thomas Lu

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Mathematics
in
Computer Science

Waterloo, Ontario, Canada, 2021

Examining Committee Membership

The following served on the Examining Committee for this thesis. The decision of the Examining Committee is by majority vote.

Supervisor:	Chris Eliasmith Professor, Dept. of Philosophy and Systems Design Engineering Cross-Appointed to Dept. of Computer Science University of Waterloo
Internal Member:	Justin Wan Professor, Dept. of Computer Science University of Waterloo
Internal-External Member:	Bryan Tripp Associate Professor, Dept. of Systems Design Engineering University of Waterloo

Author's Declaration

Abstract

This thesis explores continuous spatial and temporal representations in machine vision. For spatial representations, we explore the Spatial Semantic Pointer as a biologically plausible representation of continuous space its use in performing spatial memory and reasoning tasks. We show that SSPs can be used to encode visual images into high dimensional memory vectors. These vectors can be used to store, retrieve, and manipulate spatial information, as well as perform search and scanning tasks within the vector algebra space. We also demonstrate the psychological plausibility of these representations by qualitatively reproducing Kosslyn's famous map scanning experiment.

For temporal representations, we extend the original 1D Legendre Memory Unit to take multi-dimensional input signals and compare its ability to store temporal information against the Long Short-Term Memory Unit on the task of video action recognition. We show that the multi-dimensional LMU is able to match the LSTM in representing visual data over time. In particular, we demonstrate that the LMU is able to achieve much better performance when the total number of parameters is limited and that the LMU architecture allows it to continue operating at with fewer parameters than the LSTM.

Acknowledgements

The writing of this thesis would not have been possible without the support of the many people who helped me along the way. First and foremost I would like to thank my supervisor, Chris Eliasmith, for his guidance and support through my research. I would also like to thank all the members of the CNRG lab for all their help in discussions and brainstorming. Finally I would like to thank Justin Wan and Bryan Tripp for taking the time to be readers for this thesis.

Dedication

This thesis is dedicated to my family and friends for their unending love and support.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

In order to tackle the ever expanding amount of data being generated by the current fast-paced, data-driven world, artificial intelligence and neural networks need efficient ways to represent and compress data about the world around us. With robots and computers moving about in the world, from self driving cars to robotic vacuums and lawn mowers, being able to capture and reason about space becomes all the more important. They need to be able to build an internal map of where they are, where goals and obstacles are in relation to them, and be able to plan accordingly. In order to be able to trust the implementation of these artificial systems, we need to know that they can think and reason well.

Similarly, as points of data become streams of data, being able to capture temporal information over extended time periods becomes increasingly vital. Machines may need to connect current data to data from minutes, hours, days, even weeks ago or longer. Can a home care device understand the difference and danger between someone sitting still for a few minutes and someone sitting still for a few hours. Can a security system recognize and respond differently to an accidental bump and a full blown fight? Humans are inherently dynamic creatures and our machines need to be able to adapt to those dynamics.

Space and time share the fact that in our best physical theories, both are represented as continuous values. Whether or not they are, at their most basic level, continuous is a matter for theoretical physics. Nevertheless, extremely successful physical theories, including our best theories of motion (Newtonian and otherwise) are cast in terms of continuous space and time. So it should be unsurprising that our theories of how intelligent system represent space and time assume continuity as well. However, this has not generally been the case in the past, especially in the context of contemporary machine learning and neural networks. Because these approaches have been implemented on discrete computers, they largely assume that time and space are discrete. In this thesis we examine representations of space and time that are continuous and apply them to problems in machine intelligence.

For spatial representations, we apply Spatial Semantic Pointers in spatial reasoning tasks by building a model that can manipulate space and answer queries. Our goal is to demonstrate that Spatial Semantic Pointers are a biologically plausible representation for reasoning with continuous spatial data. For temporal representations we implement a Legendre Memory Unit model on a video action recognition task and comparing its performance against the long short-term memory model. Our goal is to demonstrate that Legendre Memory Units can be used to represent visual data over time and can compete with state of the art models in processing video data.

We begin by providing a background on the problems at hand and discuss the representations being proposed as the solutions. We look at what the representations are and how they work as well as their mathematical properties. For each representation we detail the experiments and criteria used to evaluate them. We then describe the specific elements of the architectures being used in each experiment. Following that we provide the results of the experiments and discuss the implications. Finally we discuss the overall findings and identify future work.

Chapter 2

Background

Representing space and performing spatial reasoning have been long standing challenges for artificial neural networks and machine learning models [3, 12]. However, spatial cognition has been studied in depth within natural cognition. These studies have lead to proposals suggesting that mental representations of space are stored in a continuous metric space and can be manipulated like physical images [21]. Such representations are believed to be shifted, scanned, and queried for spatial relational information. In a psychological study, Kosslyn showed participants maps of landmarks and asked them to mentally scan from one landmark to another. He demonstrated that the time it took to perform the scan was proportional to the actual distance on the map. Similarly, Fink and Pinker showed participants a series of dots on an image, then showed an arrow after removing the dots, and asked participants which dot the arrow pointed to [9]. The response time for this experiment was also proportional to the distance between the arrows and the dots, reinforcing the idea that space is stored as a traversable metric space in the memory as opposed to a list of objects and locations, for example.

While there have been many debates over the exact form of representation in the mind, Komer [19] has recently proposed a biologically inspired representation with strong potential for use in artificial network models called Spatial Semantic Pointers. Spatial Semantic Pointers work by storing spatial information in a high-dimensional vector space. Objects and properties can be assigned to high-dimensional vectors and bound to these spatial pointers within the same metric space. This allows a visual memory to be represented within a metric space that is constant in size of vector regardless of the number of objects contained. In this thesis we examine how these representations may be able to account for the kinds of metric space mental processing posited by Kosslyn.

Likewise, capturing information over long periods of time is another long standing challenge for artificial neural networks [26]. With modern datastreams providing huge amounts of data in real time, it has become necessary for neural networks to be able to remember incoming data over long periods of time. Recurrent neural networks, as well as more recent developments and variations such as Long Short-Term Memory Networks (LSTMs) [15] and Gated Recurrent Units (GRUs) [8] try to solve the problem by allowing neural networks to "remember" information over multiple time steps. However, these networks generally only perform well in the short term and often fail over extended time steps [22]. Furthermore, current models have difficulty capturing context clues and understanding actions that occur over multiple time steps [20].

Once again, there has been a new, recently proposed biologically plausible representation called the Legendre Memory Unit [34]. The Legendre Memory Unit attempts to solve the problem of representing temporal information by projecting the input signal into the Legendre polynomial space. In this way, the temporal information is optimally represented within the Legendre basis space without issue of forgetting information over long time spans. In this thesis, we examine how this novel representation of continuous time can improve the performance of a deep neural network on a video processing task.

2.1 Spatial Representation

Contemporary artificial intelligence models have become proficient at answering specific questions about specific objects and representing objects and their properties. However that proficiency does not transfer well into understanding relationships between objects and their properties [18]. Efficient methods of representation are a vital part of the computing pipeline and there is a crucial dearth in efficient representations of object relationships.

The traditional method of storing objects and their relationships is through databases and graphs, using symbols and logical connectors. Haugeland dubbed this approach as

Good Old Fashioned AI (GOFAI) [14]. Databases rely on storing properties of each object separately and computing their relationships through a series logical operations. Each new property results in at least one new field for one hot encoding representations. Graphs scale up quadratically in size and complexity as the number of objects and the relationships being considered increase [24], with an increasing amount of redundant connections. Small changes in relationships being considered can result in an entire new set of connections being required.

In either case, the size of the representations and the number and complexity of operations scale up incredibly fast quadratically with the amount of data and complexity of relationships. Modifying the kind of relationships being considered leads to increased complexity in the data stored or the complexity of logical operations required [18]. Consider that people do not have to check whether every single object in the room is red to determine which objects on a desk are red, nor does it need to check every single object for its location to determine which object is on top of another object. In artificial intelligence, this "frame problem" examines enabling logic based reasoning architectures to isolate the relevant information required for a task without having to examine the entire corpus [28]. These traditional methods of representing objects and their relationships quickly becomes insufficient as artificial intelligence systems are asked to to answer increasingly complex questions about the world.

2.1.1 Vector Symbolic Architecture

An alternative method for representing relationships that tries to overcome some of the disadvantages of the traditional representations is the Vector Symbolic Architecture (VSA) [10]. In a VSA, information is stored as a single fixed-length high-dimensional vector. Regardless of the kind of information represented and how many objects are included, the size of the representation does not change. The encoding and manipulation of information is done through algebras defined on these high-dimensional vector spaces. Using this representation, properties of objects can be bound mathematically to the respective objects and encoded within the same vector space. Information is stored and manipulated across a large high-dimensional space and relationships between concepts is described by their relationships and distances within this algebra. Compared to traditional representations, where specific bits represent specific properties or units of information, VSAs distribute the information across all bits [11], similar to how biological systems distribute its representation and processing of information across many neurons within a neural network (Distributed Information Processing in Biological and Computational Systems). In this work we leverage these advantages for spatial representation.

2.1.2 Semantic Pointer Architecture

The Semantic Pointer Architecture (SPA) uses a specific form of VSA that specifies the methods required to perform complex cognitive functions within a biologically plausible cognitive model. The high-dimensional vectors or 'Semantic Pointers' represent large populations of neurons, and all computations are done within this population of neurons. In particular, the SPA, following holographic reduced representations [27], defines three basic operations within the algebra space [1]:

1. Superposition: For any two vectors v and w, they can be superimposed by an addition operation to create a new vector that is similar to both of the original vectors.

$$u = v + w$$

2. Binding: For any two vectors v and w, they can be bound together by a circular convolution operation to create a new vector representing the bound vectors that is dissimilar to both of the original vectors.

$$u = v \circledast w$$

3. Unbinding: For a vector u representing the original vectors v and w bound together, one of the original vectors can be unbinded from the new vector by a circular convolution with the approximate inverse to recover a vector that is similar to the other original vector.

$$v \approx u \circledast w^{-1}$$

Additionally, in order to interpret the these high-dimensional vectors, the SPA defines a similarity function, cosine similarity, which represents how close two semantic pointer vectors are within this algebra.

$$Similarity(v, w) = \frac{v \cdot w}{\|v\| \|w\|}$$

In this thesis, we examine the use of *Spatial* Semantic Pointers for performing spatial memory and reasoning tasks. Spatial Semantic Pointers (SSPs) extend the SPA to allow for the encoding of continuous spatial information into high-dimensional metric spaces. The implementation of this representation is described in section 3.1, Spatial Semantic Pointers.

2.2 Temporal Representation

2.2.1 Recurrent Neural Networks

Current state-of-the-art methods[1] of representing information over time and performing reasoning generally involve some variation of Recurrent Neural Networks (RNNs). RNNs are neural networks that represent temporal information in a hidden internal state vector that is recurrently connected for each time step. Specifically, at each time step, the recurrent neural network will take the hidden internal state vector generated in the previous time step and update it with new input information. This updated hidden internal state vector is then used to generate the required output and also recurrently passed to the next time step, repeating the process (see Figure 2.1). In general, an RNN unit is mathematically described by the following equations:

$$h(t) = f(x(t), h(t-1)) \tag{2.1}$$

$$y(t) = g(h(t)) \tag{2.2}$$

[1]Recently, transformers have been gaining popularity as an alternative to RNNs for processing sequential data, especially within the field of Natural Language Processing [33]. However, the vast majority of work in the video action recognition field focuses on convolutional networks or RNNs. Very recently, there are some transformer models being proposed for video processing, but they have yet to reach state-of-the-art performance [23]. Thus, we focus on comparisons to LSTMs, although LMUs have been shown to outperform transformers in some cases as well [6].

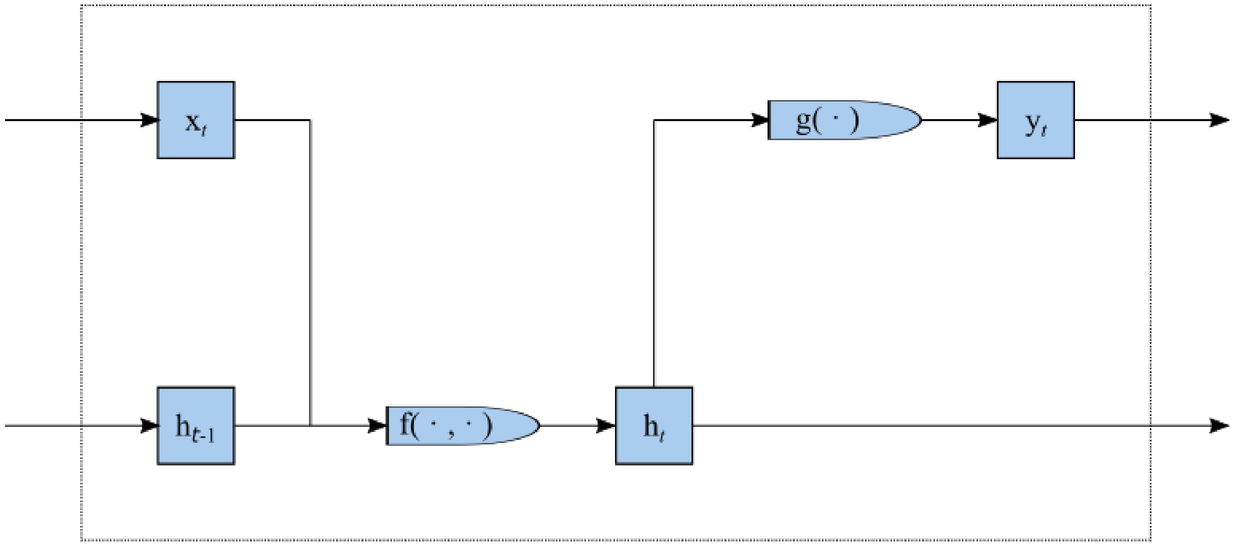

Figure 2.1: Dataflow diagram for a Recurrent Neural Network unit. At each timestep, the cell takes the current input signal x_t as well as the previous time step output of h_{t-1} to produce a new hidden vector h_t using a function f. h_t is then fed through a function g to produce a final output y_t. The f and g functions depend on the specific implementation of the RNN.

where $x(t)$ and $y(t)$ are the inputs and outputs of the system respectively, and $f(\cdot,\cdot)$ and $g(\cdot)$ are neural network layers that depend on the specific implementation of the RNN, see Figure 2.1. In a basic RNN, these would be dense layers followed by activation functions. In more complex architectures like the LSTM or GRU these functions would include input and output activations gates or calculations of additional internal states.

As with other neural networks, RNNs are trained through backpropagation and gradient descent. At each training step, the neural network seeks to reduce the error between the network's output and the correct output by minimizing the network's loss function. This is done by calculating the gradient of the loss function, by propagating the error backwards from the output of the network back through all the layers of the network, and by modifying each layer's weights accordingly.

However, because Recurrent Neural Networks have a recurrent connection from one layer's output to a previous layer's inputs, that loss gets propagated backwards through each time step. The danger is that certain gradients will increase in magnitude with every propagation while other gradients will decrease in magnitude with every propagation. This is already an issue in training very large deep neural networks where the gradients are propagated through a large number of layers, and becomes exacerbated by the recurrent backpropogation in an RNN. Naturally, as the number of time steps increases, the magnitudes of these gradients will either explode towards infinity or vanish towards zero. While variations like the LSTM models seek to alleviate the severity of these issues, pushing memory capacity up to around 1000 timesteps [22], they are not able to completely eliminate the problem of vanishing and exploding gradients.

As the world becomes increasingly digitized, data streams are becoming larger in volume as well as higher in frequency. Neural networks are required to process vast amounts of data at a near constant rate. The problem of vanishing and exploding gradients becomes impossible to ignore as the number of timesteps trends towards infinity. In this thesis, we examine a new approach to encoding temporal information within an RNN, the Legendre Memory Unit. This architecture attempts to circumvent the issue of vanishing and exploding gradients by optimally encoding temporal information within the Legendre basis domain as described in Section 3.2, Legendre Memory Units.

2.2.2 Convolutional Neural Networks

Convolutional Neural Networks (CNNs) are the most commonly used neural network architecture for analysing visual imagery [32]. More recently, however, they have been used broadly to process temporal data, such as speech [2]. CNNs use convolutional filters and kernels to convert a feature map with some number of input channels into a new feature map with a new set of feature channels. Often they are used to compress large images into a smaller set of features to process. However, convolutional layers can also be applied to the time axis of a data set [31], compressing the temporal information into a set of features in the time dimension. Successive sets of convolution can compress a long set of temporal data into a short set of features for further processing.

Using CNNs in the time domain grants all the advantages of CNNs in the visual domain.

The convolutional layers in the temporal domain are very good at extracting features and relationships within the size of the convolutional kernel. However, the compression is a function of size of the convolutional kernel. Unlike the RNN, which updates a fixed internal layer and outputs a fixed sized vector regardless of how many time steps are processed, convolutional layers have outputs that scale in size with the inputs. While the RNN does not scale well into large time steps due to backpropogation and gradient descent challenges, the CNN does not scale well due to output sizes scaling with the input. An N-fold increase in number of time steps will result in an N-fold increase in amount of features output by the CNN that need to be processed by the rest of the neural network.

In order to deal with converting longer and longer time length data sets into features, CNNs need to be incredibly large or stacked incredibly high. Either way, the sizes of the networks become excessive. Alternatively, the CNNs can be used as preprocessing and the output features in the time domain are then further fed through RNNs, but this leads to the same issues vanilla RNNs have as time steps tend towards infinity. As a result, in this thesis we combine LMUs with CNNs to process time varying visual data, i.e., video. We describe this experiment and results in Section 5, Temporal Representation with Legendre Memory Units.

Chapter 3

Methods

3.1 Spatial Semantic Pointers

In order to represent information in continuous space with high dimensional vectors, we employ a generalization of the Vector Semantic Architectures (VSAs) used by the Semantic Pointer Architecture (SPA). This generalization allows us to process continuous spaces via a method called fractional binding that constructs Spatial Semantic Pointers as proposed by Komer et al. [19]. In order to bind vectors together, the SPA makes use of circular convolution as proposed by Plate [27], which is an element-wise product of vectors in Fourier space. So, if we repeatedly bind a vector to itself, we can write:

$$B^k = \underbrace{B \circledast B \circledast B \circledast ... \circledast B \circledast B}_{\text{Binding } k \ B\text{s together}}. \tag{3.1}$$

However, as originally defined, 'k' can only take on integer values. The natural extension of this product into continuous space is an element-wise exponentiation of vectors in Fourier space. Supposing B is a fixed d-dimensional vector, fractional binding B by a real valued scalar k is defined by expressing the binding in the complex domain:

$$B^k = \mathcal{F}^{-1}\left\{\mathcal{F}\left\{B\right\}^k\right\}, \quad k \in \mathbb{R}, \tag{3.2}$$

Here $\mathcal{F}\{\cdot\}$ is the Fourier transform and $\mathcal{F}^{-1}\{\cdot\}$ is the inverse Fourier transform, and exponentiation of a vector is done element-wise. This extends the domain of k from the original binding definition from the integers to fractions and real values. An important mathematical property of this method is that we have similar algebraic properties as regular exponentiation. In particular, products of exponents:

$$B^{k_1} \circledast B^{k_2} = B^{k_1+k_2}. \tag{3.3}$$

Associativity:

$$A \circledast B = B \circledast A. \tag{3.4}$$

And the property of unbinding:

$$A \circledast B \circledast B^{-1} \approx A. \tag{3.5}$$

Here, the inverse refers to the approximate inverse used for circular convolution. Specifically, for a vector $v = (v_0, v_1, v_2, ..., v_{n-1}, v_n)$:

$$v^{-1} = (v_0, v_n, v_{n-1}, ..., v_2, v_1). \tag{3.6}$$

We can further extend this representation to allow for the mapping of continuous spaces, $\mathbb{R}$, to high dimensional vectors, $\mathbb{R}^d$. This, in turn, allows for the mapping of 2-D coordinate spaces, $(x, y) \in \mathbb{R}^2$, to high dimensional vectors, $S(x, y)$, by binding two such vectors together, each representing one part of a coordinate:

$$S(x, y) = X^x \circledast Y^y, \tag{3.7}$$

where X and Y are arbitrary high dimensional vectors representing each axis. Similarly, this method can be extended to represent 3D space or any other multi-dimensional space.

A set of multiple points, P, can then be summed together in order to create a SSP memory of multiple locations in space within a single vector:

$$M = \sum_{(x,y)\in P} S(x,y). \tag{3.8}$$

Furthermore, objects can be assigned locations in space by binding a vector OBJ with the SSP representing the point in space $S(x,y)$:

$$M = OBJ \circledast S(x,y). \tag{3.9}$$

In order to represent a set of m objects in this space together as a single memory vector, we can super impose the SSP vectors with summation:

$$M = \sum_{i=1}^{m} OBJ_i \circledast S(x_i,y_i). \tag{3.10}$$

Returning to the property of unbinding, we can extract pointers from memory by unbinding from the memory either a location, to extract the object, or an object, to extract the location. As the SSP space is high dimensional and each individual pointer is nearly orthogonal, the output will be approximately what was bound with the input:

$$M \circledast S(x_k,y_k)^{-1} \approx OBJ_k, \tag{3.11}$$

where OBJ_k is any of the m objects originally encoded into M in Equation 3.10. This representation of points in space can be further extended to continuous regions represented by some infinite set of points R, (e.g., a rectangle or a circle). We can represent a region in SSP space by taking the integral:

$$S(R) = \int_{(x,y)\in R} X^x \circledast Y^y \, dx\, dy. \tag{3.12}$$

This allows us to not only encode information to specific regions in space, but also decode information from specific regions. Furthermore, using the algebraic properties from above, we can manipulate regions in SSP space. For example, we can shift the region vector in space by multiplying the vector by the SSP vector representing the shift. For example, to shift some region, R, by a in the x direction and b in the y direction, we take the convolution:

$$S(R_2) = S(R) \circledast S(a, b). \tag{3.13}$$

Another property resulting from this method is the ability to directly calculate the vector representing rectangular regions using Euler's formula. Specifically, using the property that:

$$\int_a^b e^{ikx} di = \begin{cases} \frac{i(e^{ika} - e^{ikb})}{k} & \text{for } k \neq 0 \\ b - a & \text{for } k = 0 \end{cases} \tag{3.14}$$

Let X' be the fourier space representation of an SSP axis vector X and θ be a vector of the respective angles of X'. The i-th element of the integral of X' from a to b is given by:

$$\int_a^b X_i'^x dx = \begin{cases} \frac{i(X_i'^b - X_i'^a)}{\theta_i} & \text{for } \theta_i \neq 0 \\ b - a & \text{for } \theta_i = 0 \end{cases} \tag{3.15}$$

And subsequently, the integral of a region is given by the product of the integrals of each axis:

$$\int_c^d \int_a^b X^x \circledast Y^y dx dy = (\int_a^b X^x dx) \circledast (\int_c^d Y^y dy). \tag{3.16}$$

See Figure 3.1 for an example of using these techniques to shift a rectangular region to a new location with a single convolution.

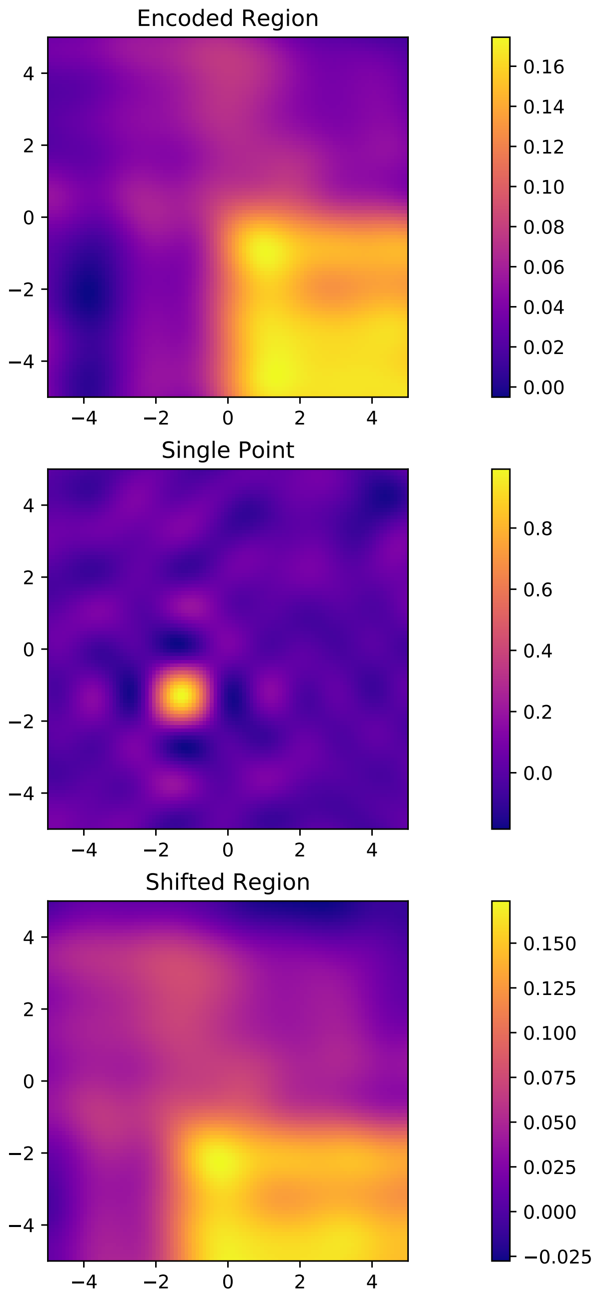

Figure 3.1: Region Shift Example Demonstration of shifting a rectangular region (top panel), to a point encoded as an SSP (middle panel), resulting in a new region vector shifted in relation to the point (bottom panel).

3.2 Legendre Memory Units

The current method that is standard for learning temporal dependencies is some version of an RNN. In particular, LSTMs are a popular choice used in a variety of machine learning applications including action recognition [7], machine translation [38], and anomaly detection [25].

However typical RNNs are prone to vanishing and exploding gradients, which cause instabilities as the length of time sequence increases. The success of LSTMs is largely due to their being less susceptible to these gradient problems. However, they are typically good only up to time series windows of about 500-1000 datapoints [22]. While the LSTM may work for simpler applications, modern data streams are trending towards being continuous in nature, making larger sample windows critical, and demanding architectures that are able to cope with much longer time windows.

The Legendre Memory Unit, LMU, proposed by Voelker [34] seeks to break this upper limit for T. Taking inspiration from the way biological neurons encode temporal information, the LMU is a recurrent architecture that optimally encodes continuous time signals onto an orthogonal basis over time. Unlike traditional RNNs, the LMU is able to maintain stability even as $T \to \infty$ due to its encoding matrices having been optimally derived from the Legendre basis.

The Legendre Memory Unit encodes temporal information within a memory cell, using encoding matrices derived from the ordinary differential equation:

$$\theta \dot{m}(t) = Am(t) + Bu(t), \tag{3.17}$$

where $m(t)$ represents the $d-$dimensional memory state vector at time t and $u(t)$ represents a real valued input signal at time t and θ represents the length of the sliding window of time. A and B are the state-space matrices used to optimally project the encoded matrices into the Legendre polynomial space, making m a sliding window of u projected into $d-1$ degrees of Legendre polynomials. The A and B matrices are defined as:

$$A = [a]_{ij} \in \mathbb{R}^{d \times d}, \quad a_{ij} = (2i+1) \begin{cases} -1 & i < j \\ (-1)^{i-k+1} & i \geq j \end{cases} \tag{3.18}$$

$$B = [b]_i \in \mathbb{R}^{d \times 1}, \quad b_i = (2i+1)(-1)^i, \tag{3.19}$$

for $i, j \in [0, d-1]$

Given m, we can recover the original signal u approximated via the first $d-1$ degrees of Legendre polynomials:

$$u(t-\theta') \approx \sum_{i=0}^{d-1} P_i(\frac{\theta'}{\theta}) m_i(t), \quad 0 \leq \theta' \leq \theta, \tag{3.20}$$

where $P_n(x)$ is a scaled n-th degree Legendre polynomial defined as:

$$P_n(x) = (-1)^n \sum_{k=0}^{n} \binom{n}{k} \binom{n+k}{k} (-x)^k. \tag{3.21}$$

The discretized version of the LMU is given by:

$$m_t = A' m_{t-1} + B' u_t, \tag{3.22}$$

where A' and B' are discretized versions of the above A and B matrices in equations (3.18) and (3.19) obtained using an ODE solver. In particular, solving with Euler's method for sufficiently small Δt:

$$A' = (\Delta t / \theta) A + I, \tag{3.23}$$

$$B' = (\Delta t / \theta) B. \tag{3.24}$$

Note that we will be referring to the discretized versions of the Legendre Memory Unit for the implementation in this thesis.

The full Legendre Memory Unit neural network layer takes an input x_t representing the signal value at timestep t and calculates the memory vector m as well as a hidden state h.

At each time step the input signal x_t as well as the memory and hidden states of the previous time steps are encoded into an input signal u_t for memory.

$$u_t = e_x^\intercal x_t + e_h^\intercal h_{t-1} + e_m^\intercal m_{t-1}, \tag{3.25}$$

where e_x, e_h, and e_m are learned encoder vectors.

The memory vector m is updated with the new input u_t using the A and B matrices:

$$m_t = Am_{t-1} + Bu_t. \tag{3.26}$$

The hidden state is then calculated using the current input x_t, the current memory vector mt, and the previous hidden vector h_{t-1}:

$$h_t = f(W_x^\intercal x_t + W_h^\intercal h_{t-1} + W_m^\intercal m_t), \tag{3.27}$$

where W_x, W_h, and W_m are learned weight kernels and f is a nonlinearity, in this case we use tanh.

See Figure 3.2 for the full dataflow of an LMU layer.

3.2.1 Multi-dimensional LMUs

The original Legendre Memory Unit proposed by Voelker encoded inputs into one dimensional inputs through time. However, for our experiment, the LMU is required to process multiple features at each timestep and remember multiple features at each timestep. Stacking a separate memory unit for each input feature and processing each feature separately is a possible solution, but realistically not every feature needs to be remembered and features need to be considered in the context of other features to determine what the model needs to remember.

Our solution for memorizing a d_x-dimensional input signal was to extend the dimension of the internal encoders and decoders by d_u, the new size for the internal feature memory u_t.

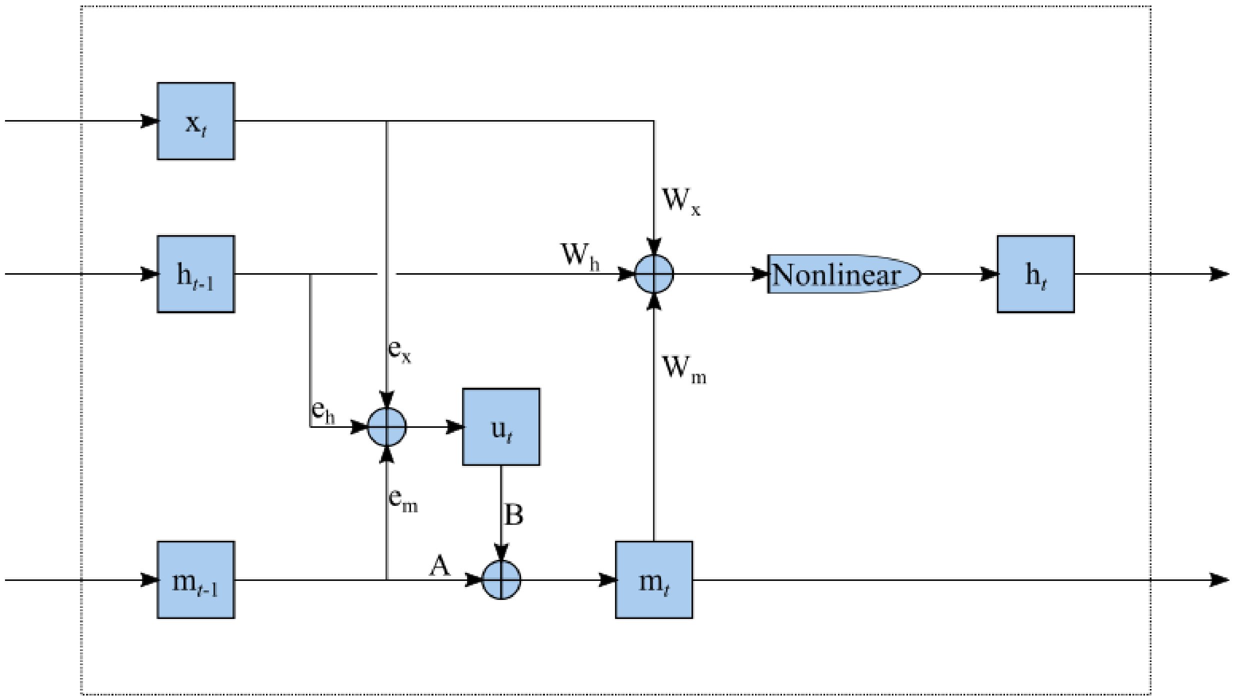

Figure 3.2: Dataflow diagram of an LMU cell. Squares represent variables, matrices along an arrow represent matrix multiplication, and $\bigoplus$ represents summation of inputs. At each timestep, the cell takes the current input x_t as well as the previous time step outputs of h_{t-1} and m_{t-1}. e_x, e_h, e_m are encoder vectors used to produce the intermediate input signal vector u_t. A and B are the state-space matrices used to optimally project the input vectors into the the output m_t. W_x, W_h, W_m are weight kernels used to produce the input to the nonlinear function (in this case tanh is used) to produce the output h_t.

Variable	Original Dimensions	New Dimensions
x	$d_x \times 1$	$d_x \times 1$
u	1×1	$d_u \times 1$
h	$d_h \times 1$	$d_h \times 1$
m	$d_o \times 1$	$d_o \times d_u$
e_x	$d_x \times 1$	$d_x \times d_u$
e_h	$d_h \times 1$	$d_h \times d_u$
e_m	$d_o \times 1$	$d_o \times d_u \times d_u$
W_x	$d_x \times d_h$	$d_x \times d_h$
W_h	$d_h \times d_h$	$d_h \times d_h$
W_m	$d_o \times d_h$	$d_o \times d_u \times d_h$

Table 3.1: Vector Dimensionality Changes for Multi-Dimensional LMUs

Now, rather than memorizing a single value at each timestep, the LMU instead encodes the d_x-dimensional input signals into a d_u-dimensional signal for memory. The size of the LMU memory state m_t is extended in a new dimension by d_u to accommodate the increased dimensionality of u. And the sizes of the recurrent connections are also increased by a factor of d_u. Encoding the new vector u into memory is done with the same A and B vectors in parallel across all d_u units.

Table 3.1 shows the dimensionality changes of the different vectors. Note that d_o refers to the order of the memory vector, that is the number of degrees of Legendre polynomials used in the approximation. The crucial difference in the new memory vector size is the product of the number of memory units and the order of the memory.

3.2.2 Gates

Because not every timestep is necessarily important for memorization, we explored adding an input gate architecture to the LMU. This is inspired by the use of gates in the LSTM, which increases the quality of the information being stored into the memory state by selectively storing only the information that passes through the gate. To do this, an additional vector i_t is calculated at each timestep consistent with the standard input gates found in the LSTM:

$$i_t = \sigma_s(W_x^\intercal x_t + W_h^\intercal h_t + b), \tag{3.28}$$

where $\sigma_s(x)$ is the sigmoid function:

$$\sigma_s(x) = \frac{1}{1 + e^{-x}}. \tag{3.29}$$

This input gate is applied to the memory state, modifying the memory state formula to:

$$m_t = Am_{t-1} + i_t \circ Bu_t, \tag{3.30}$$

where $\circ$ is the element-wise product. Here i_t is either in R^{d_u}, where each dimension of u gets a separate gate applied in parallel across the orthogonal time dimensions in memory, or in $R^{d_u \times d_o}$, where each dimension of u and each dimension of time gets its own gate applied.

3.2.3 LSTMs

In this thesis, we employ the standard LSTM [15] with input and forget gates as a point of comparison to the LMU. The LSTM neural network layer takes an input vector x_t, and maintains a hidden state h_t and a memory state m_t at every time step.

At each time step the input signal x_t as well as the previous hidden state h_{t-1} are encoded into an input signal u_t:

$$u_t = \sigma_t(e_x x_t + e_h h_{t-1} + b), \tag{3.31}$$

where $\sigma_t(x)$ is the hyperbolic tangent function, $\tanh(x)$:

$$\sigma_t(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}. \tag{3.32}$$

Three activation vectors representing the“gates” are calculated using the input vector x_t and the previous hidden state h_t and passed through sigmoids.

$$f_t = \sigma_s(W_{fx}x_t + W_{fh}h_{t-1} + b_f), \tag{3.33}$$

$$i_t = \sigma_s(W_{ix}x_t + W_{ih}h_{t-1} + b_i), \tag{3.34}$$

$$o_t = \sigma_s(W_{ox}x_t + W_{oh}h_{t-1} + b_o). \tag{3.35}$$

The memory cell m_t is calculated by applying the forget gate f_t to the previous memory cell m_{t-1} and the input gate i_t to the current input signal u_t:

$$m_t = f_t \circ m_{t-1} + i_t \circ u_t. \tag{3.36}$$

The hidden state is then calculated by applying the output gate o to the hidden state after being applied to a nonlinearity, in this case tanh:

$$h_t = o_t \circ \sigma_t(c_t). \tag{3.37}$$

Training for both the LMU and LSTM is done using categorical cross-entropy loss with the Adam optimizer [17].

3.3 Summary

This chapter surveys the methods and architectures that we implement in the models we build for our experiments in the following chapters. Specifically we discuss the reasoning behind choosing these methods as well as the underlying mathematical properties of the methods.

In Section 3.1 we describe the mathematics behind the SSP that allow us to encode and decode information to and from the SSP memory vectors. We then the mathematical properties that allow us to manipulate space within this representation, including creating vectors representing continuous regions as well as shift objects around in space.

In Section 3.2 we describe the shortcomings of traditional RRN models and how the LMU aims to overcome these issues. We detail the architecture of the LMU and how we extended to multiple dimensions for our models. We also describe the architecture of the LSTM in comparison to the LMU. Furthermore, we examine a possible extension to the LMU by adding gates, consistent with their use in the LSTM.

Chapter 4

Spatial Reasoning with Spatial Semantic Pointers

4.1 Overview

In this section, we apply Spatial Semantic Pointers in spatial reasoning tasks to demonstrate that SSPs are effective in performing reasoning on continuous spatial data and can also be used as a model for human cognition. We begin by describing the experimental designs, focusing on two spatial relationship tasks involving asking relational questions about a given image map. We then detail each of the elements of the architecture used for each of the experiments and how they integrate together to perform the tasks. Finally, we present the results and discuss the findings.

4.2 Experimental Design

Our experimental design for testing the usefulness of the Spatial Semantic Pointers focuses on examining the representation's ability to support spatial reasoning while remaining biological plausible. For the first experiment, we adopt a task similar to that proposed by

Weiss et al. [37] in which we use the SSP architecture to answer spatial relational queries based on input images. Specifically, we construct a set of input images by randomly placing objects, in this case numerical digits, onto a visual space. We then randomly choose two of the digits placed in each image. One is assigned as the target goal for the model which is not given to the model, the other is assigned as the query object which is given to the model. Additionally, we give the model the rough direction from the given object to the target object. The purpose of the model is to identify the goal object given the query object plus a direction. In our experiment, we use 4 directions as the possible queries, either up and to the right, up and to the left, down and to the right, and down and to the left (see Figure 4.1). We allow the model to answer with a any digit that is in the correct direction in relation to the query.

For the second task, we examine the representation's biological plausibility by using it to perform a visual scanning experiment based on Kosslyn's work [21]. In Kosslyn's map experiment, participants are given a simple map to memorize, after which they are asked to picture the map starting at a given spot and scan towards a target location. Kosslyn found that the time it took participants to scan between locations scaled linearly with the physical distance of the locations on the map, suggesting that spatial representation in the human brain has a metric and scales linearly.

For our experiment, we perform the same task given by Kosslyn by re-using the generated images in the previous task as our maps and the digits as our location objects. The model is given an origin object and a target object as a query, and it is required to extract the location of the origin object and travel from that object to the target object within the SSP representation space. For example, if using Figure 4.1, the model might be asked to scan from the '4' to the '7'. While Kosslyn compared physical distance against time in seconds, we will be comparing pixel distance against time steps, with each movement within the space performed by the model constituting one time step.

Furthermore, as scanning requires repeated operations to move around in memory, this experiment will also serve to investigate the robustness of the model. Outside of cleaning the initially extracted location vectors, all intermediate vectors used for scanning will be generated by the SSP location shifting operations. Completion of the scanning tasks will demonstrate that SSP models can be shifted repeatedly and still behave as expected.

4.3 Architecture

In this section, we go in depth into each of the components of the spatial relationship memory model used to perform the tasks described in the experimental design section. We detail each of the components and their functions as well as describe the flow of information through from the visual cue input to the final response output.

4.3.1 Dataset

Since the focus of the research and experiments is the visual memory of the model and its ability to perform spatial relational queries rather than image classification, we are using simple MNIST digits for our objects. Each digit corresponds to one of ten distinct objects in the memory dictionary.

The images used for the experiments are generated by selecting batches of $k = 2 : 4$ digits randomly selected 28x28 pixel images from the MNIST database and placing them onto a 120×120 pixel visual field (see Figure 4.1). In order to restrict the selected digits for each image to be unique, we sample by uniformly drawing digits from the MNIST database, tossing out any digits already picked, until the required number of objects has been placed in the image.

To ensure objects are not cut off by the edges of the visual field, coordinates for the center of each digit are picked from the inside 96×96 area of the visual field. We also limit the coordinates to not only be too close in order to avoid visual overlap between digits. Furthermore, because the experiment queries are limited to the four diagonal directions, coordinates are selected such that the vertical and horizontal coordinates of the objects are not too close to ensure no ambiguity in direction. We do this by dividing the sample space into $m \times m$ blocks. For each of k objects in a batch, we sample without replacement a row and a column from the $m \times m$ blocks, then coordinates for the objects are sampled from within each block.

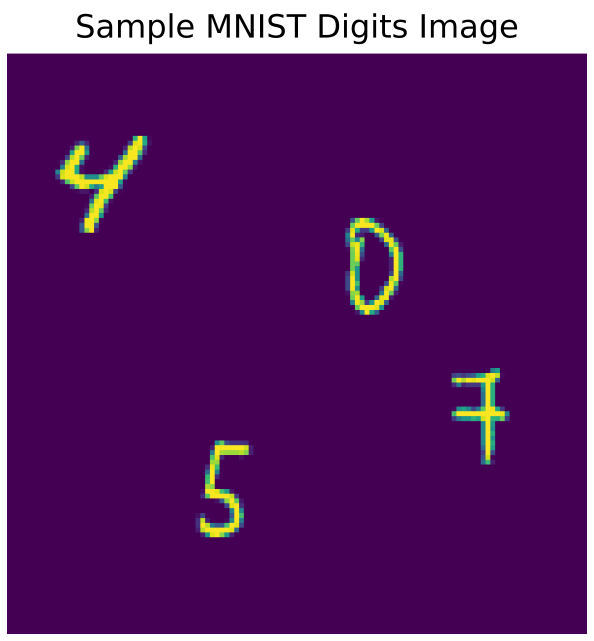

Figure 4.1: Example randomly generated MNIST image. A possible question could be "What is down and to the right of the '0'?", in which case '7' would be the correct response. If the question was instead "What is up and to the left of the '7'?", then both '0' and '4' would both be correct responses.

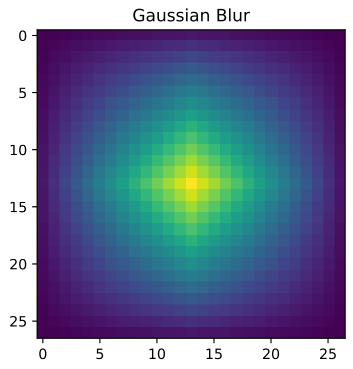

Figure 4.2: Gaussian blur matrix used to blur input images using matrix cross-correlation in order to generate a saliency map.

4.3.2 Visual Attention

We employ the use of visual saliency maps in order to decode the input images into objects for memory. Research has shown that human vision consists of a high resolution fovea at the center of each eye's visual field surrounded by a very low resolution peripheral vision [16]. Capturing visual information consists of presaccadic processing to create a saliency map from which saccadic eye movements are used to attend to various parts of the visual field and snapshot information while saccadic suppression at each step is used to prevent revisiting a target [35].

This system is simulated in our model using a gaussian blur (See Figure 4.2 to generate a saliency map representing the low resolution vision. The highest saliency point in the map at each step is selected for our visual system to observe at each step, with a mask being applied afterwards to prevent revisiting. Observing consists of cropping out the 28x28 block centered at the point of attention and feeding the cropped image to the object classification system representing the high resolution fovea. The attention mask is a simple element wise multiplication mask of 0s where the crops occur and 1s everywhere else. This process is displayed in Figure 4.3.

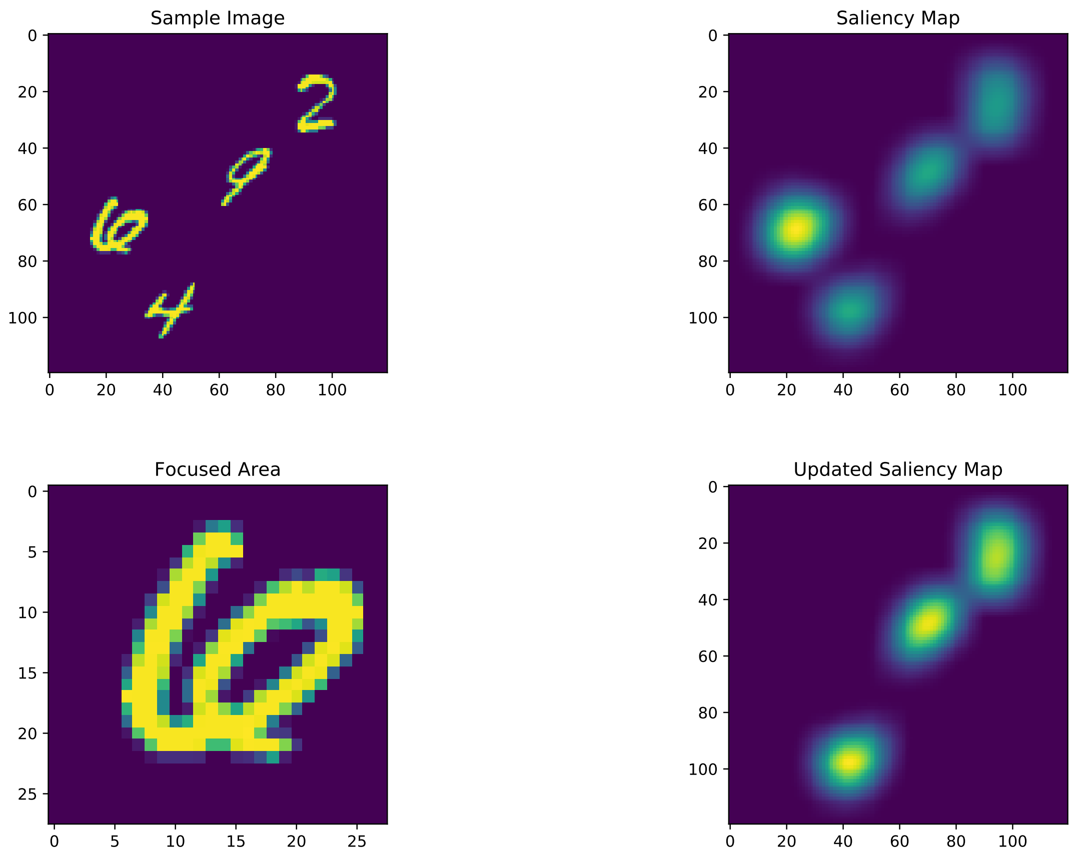

Figure 4.3: Visual Attention Simulation. The top left panel shows an example randomly generated MNIST image given as an input the model. The Gaussian blur from Figure 4.2 is applied to the input to generate a saliency map (top right). The point of highest saliency in the saliency map is chosen to focus on and a high resolution crop is taken of the area (bottom left). In this case, a '6' is observed and encoded into the SSP memory. Finally, a mask is applied to remove the already "observed" object from the saliency map (bottom right) and the process is repeated.

4.3.3 Object Classification

We employ a basic convolutional neural network in order to classify the cropped images from the visual attention module. The architecture of the network is as follows:

1. 2D Convolutional Layer, 32x3x3 filters.
2. Batch Normalization Layer
3. 2D Convolutional Layer, 64x3x3 filters.
4. 2D Max Pool Layer, 2x2
5. Dense Layer, 128 units
6. Dense Layer, 10 units

During training, Dropout layers of 0.2 and 0.4 are used after the Max Pool layer and the first Dense layer respectively. The network is trained for 12 epochs on the MNIST training set, reaching a final test accuracy of 99.12%.

4.3.4 Spatial Semantic Pointer Encoding

For our model, we chose to use 512-dimension semantic pointer vectors to represent our visual memory. Each of the 10 different digits as well as each of two axis vectors used for encoding location is represented by a randomly 512-dimensional SSP vector. Vectors are generated randomly at the beginning of each experiment and stay the same for each sample. No hand picking or crafting of vectors was required for our results.

The identified objects from the previous steps were passed on along with their coordinates. The coordinates are mapped from the 120×120 visual field to a 10×10 continuous space, specifically the intervals of $x \in [-5, 5]$ and $y \in [-5, 5]$ and converted to the corresponding spatial semantic pointer.

We experimented with both point and region representations for the objects. For point representations, the spatial pointers are left as is. For region representations, the points are bound with a 1×1 box around the origin to convert the point to a region vector representing approximately the area of space the object was located. The idea is that it is more intuitive to remember and recall an area where an object is located rather than a single, specific point in space, such as its centre of mass. Mathematically, the query region is more similar to a small subregion compared to a singular point within the region, increasing the likelihood that the object location in the region can be extracted.

Recall that unbinding location from memory returns approximately the original object bound to that location with some noise from the other objects in memory. Similarly, unbinding a region that is close to but not the same as the point to which an object is bound produces an output that is close to but not the same as the object that was bound. Binding the original object to a small region should make the location more similar to the region query, which should in turn make the output more similar to the original object and increase overall accuracy.

Given either the point or square region spatial semantic pointers, the SSPs are then bound through circular convolution with each corresponding object vector and summed together to form a 512-dimensional spatial semantic pointer representing the model's memory of the visual information.

For example, given an image with the following: 3 at (36,60), 9 at (60,12), and 8 at (90, 90), the coordinates are normalized to (-2,0), (0, -4), and (2.5, 2.5) respectively. Now let $THREE, NINE$, and $EIGHT$ represent randomly generated vectors for each digit, and X and Y be the axis vectors for a 512-dimensional SSP vector space. Let $SQUARE$ be 1×1 square region around the origin calculated using equation 3.14. We construct a square region represented memory vector M by binding each digit vector to axis vectors raised to the normalized coordinates and summing them together:

$$\begin{aligned} M =& THREE \circledast SQUARE \circledast X^{-2} \circledast Y^{0} \\ &+ NINE \circledast SQUARE \circledast X^{0} \circledast Y^{-4} \\ &+ EIGHT \circledast SQUARE \circledast X^{2.5} \circledast Y^{2.5} \end{aligned} \tag{4.1}$$

.

4.3.5 Query Generation

The queries are composed of two parts, an object to start from and a direction to look in memory. For each sample image, one digit is randomly selected as the given starting object and a different digit is randomly selected as the target digit. The full query given is the starting object represented as a SSP vector and the direction to the target object, each represented as one-hot encoded vectors. The directions in this case are simplified to four directions encoded as a 2 by 2 vector: Up and Left, Up and Right, Down and Left, Down and Right. For example, in Figure 4.1, we could select '0' as the given digit and '7' as the target digit, and provide the query: Down and to the Right of '0'. This is encoded as:

$$x = [1, 0, 0, 0, 0, 0, 0, 0, 0, 0]$$
$$D = \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix},$$

where x is the encoded given digit and D is the encoded given direction.

4.3.6 Information Extraction

In order to extract the response to a query, first the query must be converted to Spatial Semantic Pointer form. The object in the given query represents the location from which to start the search. Unbinding the object SSP from the memory SSP via inverse convolution gives us the location of the object. The location is then put through a cleanup before being converted to a search region.

In order to convert the Spatial Semantic Pointer from a location to a region, a precomputed 10x10 rectangular SSP representing the quadrant corresponding to the direction given in the query is bound to the SSP. 10x10 is chosen as the maximum distance that needs to be searched in the space in any given direction. Note that a single 10x10 rectangle can be precomputed and shifted along the X and Y axes accordingly to each of the 4 quadrants making this step very fast. (See Figure 4.4)

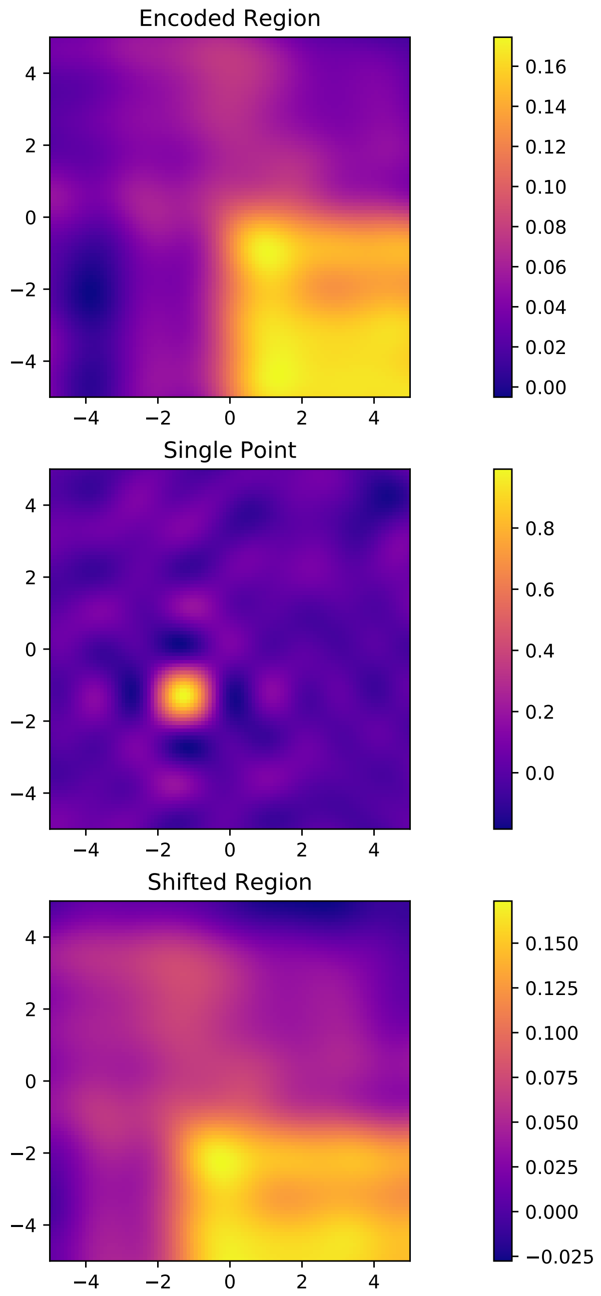

Figure 4.4: Region Shift Example. 10×10 rectangular region representing a down and to the right (top panel) can be shifted by convolution with a single point (middle panel) resulting in the final query region to be searched (bottom panel).

The end result is a rectangular SSP representing the area to be searched. Unbinding this SSP from the memory object gives us a value representing the objects found in this area. Comparing the output with the dictionary provides the query response. The entire data pipeline from initial input image to final output result is shown in Figure 4.5

4.3.7 Location Cleanup

For the above experiment, cleanup on each extracted location vector was performed using a precomputed cleanup dictionary. The visual space is divided evenly into 100x100 coordinates and the SSP vector representing each point is computed and stored in an array. During each experiment, whenever location vectors are extracted, they are converted to a clean vector by picking the SSP vector within the precomputed cleanup dictionary with the highest dot product similarity with the extracted location vector. Since the vectors are precomputed, this is a very quick parallel matrix computation.

4.3.8 Scanning Memory

For the other experiment, replicating Kosslyn's spatial map scanning task, we extended the previous relational query model by adding a means of scanning memory. Specifically, we reuse the previous architecture to generate an SSP memory vector representing the model's memory of the map image. We first unbind the two objects from the memory in order to extract their locations, and use location cleanup to get clean vectors. Using these two location vectors, we can calculate the direction vector V required to travel from the origin (x_o, y_o) to the target (x_t, y_t):

$$V = (X^{x_t} \circledast Y^{y_t}) \circledast (X^{x_o} \circledast Y^{y_o})^{-1}. \tag{4.2}$$

We then normalize this vector, shrinking it to a 0.05 unit step scan vector. Using this scan direction vector, we start at the origin location vector and scan the memory by repeatedly binding the scan direction vector to current location to generate a new location vector and unbinding the new vector from memory to extract or the objects at the location.

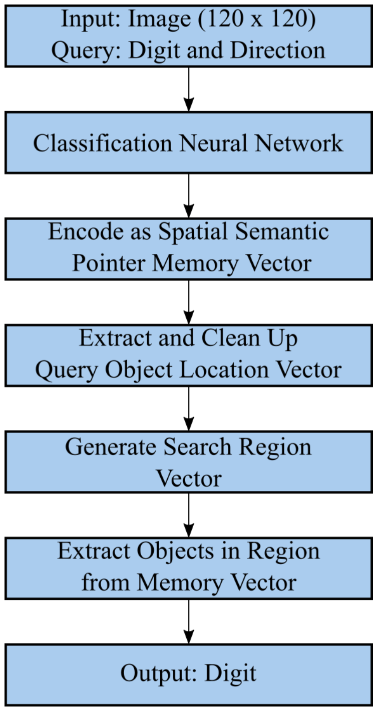
Input: Image (120 x 120)
Query: Digit and Direction
Classification Neural Network
Encode as Spatial Semantic Pointer Memory Vector
Extract and Clean Up Query Object Location Vector
Generate Search Region Vector
Extract Objects in Region from Memory Vector
Output: Digit

Figure 4.5: Dataflow diagram for the Spatial Relation Queries model. Input images are fed through a classification network and encoded as SSP memory vectors. The query digit's location is extracted and cleaned up, then bound with the region representing the direction to produce a region SSP vector representing the query. The image memory is then searched to produce the unknown target digit.

The scan ends when the model reaches the target object. For this we used dot product similarity to determine what objects are extracted from a given location and a similarity threshold of 0.8 was used to determine when the target object has been reached. This process is shown in Figure 4.6

In order to make the most direct comparison between the SSP architecture and Kosslyn's experiment results, we ensure that the objects are classified correctly prior to insertion into memory. This ensures that classification errors are not considered and that only the architectures ability to navigate through space in memory is tested. Thus the MNIST classification neural network is replaced by directly feeding in the correct digit.

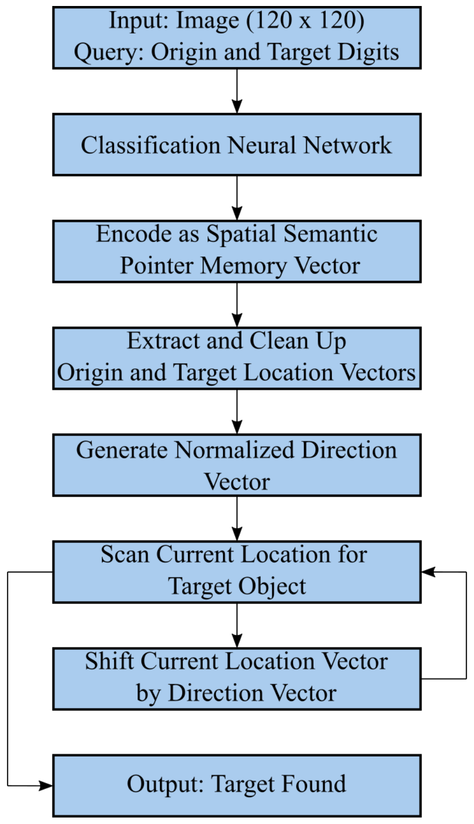
Input: Image (120 x 120)
Query: Origin and Target Digits
Classification Neural Network
Encode as Spatial Semantic Pointer Memory Vector
Extract and Clean Up Origin and Target Location Vectors
Generate Normalized Direction Vector
Scan Current Location for Target Object
Shift Current Location Vector by Direction Vector
Output: Target Found

Figure 4.6: Dataflow diagram for the Image Scanning model. In this model, both the origin and the target digit are known and their locations are extracted from memory. Then the direction vector is produced and the image memory is scanned starting from the origin location until the known target object is found.

	2 Digits	3 Digits	4 Digits	5 Digits
Point Representation Accuracy	98.06%	91.98%	88.26%	78.52%
Standard Error	0.1951%	0.3841%	0.4552%	0.5808%
Region Representation Accuracy	97.98%	92.48%	90.42%	80.26%
Standard Error	0.1990%	0.3729%	0.4162%	0.5629%
Baseline probability	100.00%	71.76%	62.60%	57.84%

Table 4.1: Experiment results for spatial relation queries.

4.4 Results

4.4.1 Spatial Relation Queries

For the spatial relation queries, we ran 5,000 randomly generated experiments for each of 2 to 5 digits for both point and region representations (Table 4.1). Accuracy was based on whether the model can output an object in the specified direction from the query. As unique answers are not forced during the random query generation, it is possible that there could be multiple correct answers for a given query. For example in Figure 4.1, both the digit '0' and the digit '7' would count as acceptable correct answers for the query "What is above and to the right of the digit '5'?"

The baseline performance the models are compared against is the probability of randomly selecting the answer from the other digits in the memory given a digit. For example in the previous example in Figure 4.1, given the query "What is above and to the right of the digit '5'?", the remaining digits are '4', '0', and '7' and there is a 2/3 chance of selecting a correct response out of them. Similarly, the model would be correct guessing any of the remaining digits given the query "What is below and to the right of the digit '4'?". This baseline was chosen as it is the minimum probability if the model can correctly remember the digits in the image but not where they are.

The baseline probability is calculated by dividing the average number of correct answers in each image by one less than the total number of digits in each image. For the 2 digit

	2 Digits	3 Digits	4 Digits	5 Digits
Difference	0.08%	0.50%	2.16%	1.74%
Standard Error	0.2786%	0.5354%	0.6168%	0.8088%
p-value	38.70%	17.52%	0.02311%	1.573%

Table 4.2: p-values for differences between representations.

case, this is trivially 1 correct digit out of 1 other digit, 100%. The baseline probabilities for each case are inherently very high due to the broadness of our queries.

The 2 digit case is not extremely interesting as the baseline probability is 100%. The model is expected to have perfect accuracy if it is able to correctly encode the visual image into the memory and extract the two objects. The results show just that, we see that the model is able to correctly identify the other digit in the image given the query digit about 98% of the time. Given that the classifier itself has a 99.1% accuracy when classifying the digits in the visual image, this result suggests that with just 2 objects, this model meets the expectation of at least being able to store and recall the objects, spatial location aside, with most of the error due to classification error. The accuracies of the two models are close enough that there is no discernible benefit or harm to encoding an object as a point or region with so little outside noise in memory.

Moving onto the 3 to 5 digit cases, it is now not sufficient to just remember the objects but to also correctly encode and extract their locations within the Spatial Semantic Pointer memory. The memory models were able to exceed the 3, 4, and 5 digit experiment baseline probabilities by 20.2%, 25.7%, and 20.7% respectively in the point representation model, and by 20.7%, 27.8%, 22.4% in the region representation case. Beating the baseline by such a wide margin confirms that the model can successfully convert the visual space into memory and perform spatial relational reasoning from the memory.

Outside of the 2 digit trivial case, the region representation outperforms the point representation (Table 4.2), with p-values less than 2% for the 4 and 5 digit cases. This confirms the hypothesis that binding objects to squares that are closer in shape and size to the regions used for queries compared to singular points improves the rate of successful unbinding and extraction from the memory. Mathematically, we know that the closer the original binding SSP is to the query unbinding SSP, the higher the similarity will be between the original target SSP and the output. The findings confirm that this is happening in practice. Conversely, this also suggests that more focused, smaller queries would be easier to answer than the broad, wide area queries asked in this experiment.

While the accuracy does decrease as the number of digits goes up, this is expected as more possible digits means a lower chance of guessing correctly and a higher difficulty in the task. Mathematically, the greater the number of objects in memory, the greater the amount of noise in the output. Decodeability of SSPs, and VSAs in general, tend to drop as a function of objects in memory.

Both models are able to effectively store, recall, and perform spatial reasoning with reasonable accuracy when given visual fields with 4 objects. However, the large drop in accuracy in both models of about 10% when increasing from 4 to 5 digits suggests that the memory models are reaching capacity in terms of what can be readily stored and recalled. This is consistent with human spatial cognition, as human working memory is estimated to have a capacity of about 3-4 items [4].

4.4.2 Image Scanning

For the image scanning task, we ran 100 random trials each with a different randomly generated map, with each map consisting of 4 objects placed on a visual field from which one is chosen as the origin and one is chosen as the target. The maps in this experiment were generated with the same method as the relation query experiment, with the digits representing the landmarks. Each trial also used different randomly generated SSP object and axis vectors to represent different subjects. Pixel distance is calculated based on Euclidean distance between object centers, and each step consists of a single movement and extraction cycle. An experiment ends when the target object is found, which occurs when the extracted object has a similarity value of at least 0.8 with the target object.

The results in Figure 4.7 show a strict linear relationship between object distance and time steps required to scan with a Pearson's R value of 0.9989 and a p-value of 2×10^{-132}. We can see that the model is able to continuously apply small shifts to the location vector without the vector deteriorating. While we know mathematically that location vectors can be shifted without losing accuracy, this experiment shows that the operations are robust against numerical computation errors. Location cleanup was only used on the extracted locations for the origin and target object, the intermediate location vector used in scanning was not cleaned after each shift. This experiment shows that these location vectors still behave as expected after over 100 location shift operations.

Furthermore, we see that none of the trials timed out. Each target object was successfully found, demonstrating that given precise and direct queries, the Spatial Semantic Pointer architecture is able to store and extract spatial information to a high degree of accuracy with four objects in memory. This also reinforces the earlier suggestion that tighter, more focused queries yield higher success rates. In this case, a specific vector direction results in improved performance compared to searching an entire quadrant.

From a biological standpoint, the image scanning experiment further demonstrates the capability of the Spatial Semantic Pointer memory model to capture the cognitive behaviours of human working memory. The SSP model is able to successfully replicate Kosslyn's findings that human spatial memory is a metric space. Kosslyn found that the time for the human mind to scan from object to object scaled linearly with distance on top of a flat reaction time before scanning took place. Our results chart is nearly identical, with the only difference being the intercept is closer to 0. However this can be explained by the time steps only counting time spent scanning and not the flat amount of time required to process the query and produce the relevant vectors before scanning begins. This time is not easily converted into time steps but would make up for the missing flat time spent to match our results with Kosslyn's.

4.5 Discussion

In this chapter we examined the effectiveness of Spatial Semantic Pointers as a biologically plausible representation for performing spatial reasoning tasks. The spatial relation queries experiment showed the architecture is able to perform tasks based on vague imprecise relational queries that encompass large regions of the visual space. The image scanning task demonstrated the representation's robustness while performing precise tasks involving many repeated binding operations to navigate from one portion of the memory to another.

The spatial relation queries experiment showed that the architecture is able perform spatial reasoning within the semantic pointer space to a higher degree of accuracy with up

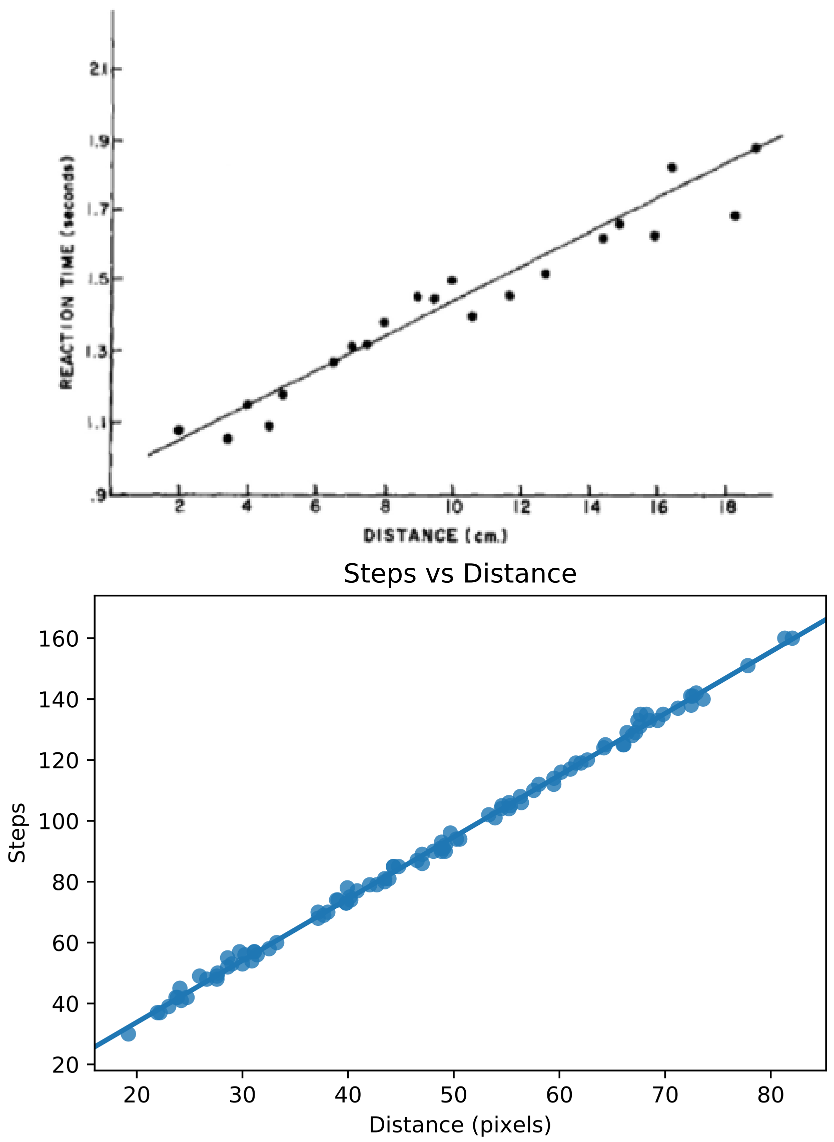

Figure 4.7: Comparison of Kosslyn map scanning results and Image Scanning model. Results from the original mental map scanning experiment performed by Stephen Kosslyn [21] (top panel). Results from our reproduction of the experiment performed by our image scanning model over 100 trials (bottom panel).

to 4 objects, and maintain a reasonable degree of accuracy at 5 objects, consistent with our expectations from human spatial cognition and human working memory. Spatial Semantic Pointers can fit seamlessly into a full computer vision pipeline spanning from processing pixel images with neural network to creating working spatial memory and answering queries based on memory.

In addition to the results of the spatial relation queries experiment being consistent with the size of human working memory, the image scanning experiment further reinforces the biological plausibility of the architecture as a model of human cognition. While Vector Symbolic Architectures have been proposed in the past to model human working memory, these experiments show that the Spatial Semantic Pointer architecture in particular can be used to model human spatial cognition.

Chapter 5

Temporal Representation with Legendre Memory Units

5.1 Overview

In this section, we apply our expanded multi-dimensional Legendre Memory Unit to perform video action recognition and compare its effectiveness against similarly sized Long Short-Term Memory models. We introduce the UCF 101 dataset that we use for our experiment as well as the describe the preprocessing performed. We outline the model architecture and training parameters used. At the end, we present the results and discuss the findings.

5.2 Experimental Design

In order to test the LMU's effectiveness at capturing spatial temporal information over time, we chose to implement it in action recognition in videos. The model is required to remember contextual information from visual data across multiple time frames in order to best determine what action is occurring in a given video. Specifically, the model is given video data as images frame by frame and tasked with classifying the action being taken in the video once all the frames are given. The dataflow diagram for the model is shown in Figure 5.1, the specific elements of the model will be explained in the sections following.

To maintain consistency when comparing models, the only change between models is in the Legendre Memory Unit layer within the dataflow diagram. For LMU models, we tested layers of varying sizes of the internal hidden and memory units, as well as a variant of the LMU using gates. For the comparison with LSTM models, the LMU layer is swapped directly with an LSTM layer using default Keras LSTM parameters. The only change was in the number of internal hidden and memory units.

5.3 UCF 101

The UCF 101 dataset published by the University of Central Florida [29] consists of 13320 videos spanning 101 action categories, extending the previous UCF50 dataset. Actions come from a wide variety of categories and include brushing teeth, biking, playing the violin, and taking soccer penalty shots. In addition to the wide variety of actions, each video contains further variations in terms of camera placement, lighting, background clutter, scale, object appearance etc. This simulates real world noisy data as opposed to generated data with standardized background and lighting. The videos in the dataset have also been preprocessed for easy integration into Python saving on computation time. The artificial neural network model needs to be able to remember the visual information and pick out the correct context clues to identify the correct action.

5.4 Preprocessing

Preprocessing was done using Harvey's UCF101 training code [13]. Video from the UCF101 dataset was first extracted into individual frame images and converted into python readable matrices. Each video's length was standardized by selecting 40 frames evenly spaced through each video's time span; videos under 40 frames were not considered. Image size was standardized into 224x224 pixel matrices with 3 colour channels.

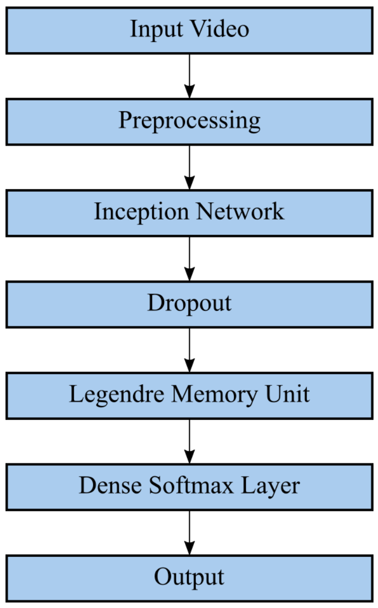

Figure 5.1: Dataflow for LMU action classification model

In order to compare the models purely on their ability to process information across time, the models are given features from the same pre-trained classification networks. Thus neither model has an advantage in image processing and the only comparison is between the LMU and the LSTM parts of the model. For feature generation, we chose two state-of-the-art classification networks, the Inception 2D network and the Inception 3D network.

The Inception 2D network is a state-of-the-art network used for static image classification [30]. For preprocessing purposes, the top classification network was removed and the previous feature layer is used as the output. Each video then becomes 40 time steps of 2048 features which are fed into each of our models.

We also took a look at using the Inception 3D network as preprocessing [5]. While the network was designed to classify whole videos without recurrent networks by using 3D convolutional networks, there is evidence that improvements can be made by feeding the few output timesteps through a final LSTM layer before classification [36]. While this kind of network requires the entire video sequence at once and isn't feasible for real time applications, we thought it would provide another point of comparison between the LMU and LSTM. Feeding in the video data and removing the top classification layer of the network reduces the 40 input frames of video to 4 time steps of 1024 features.

The UCF101 dataset is already pre-split into a training set of 9,537 samples and a testing set of 3,783 samples. We further randomly selected 20% of the training set to be a validation set.

5.5 Legendre Memory Unit Model Training

As shown in Figure 3.2, Dropout was used to prevent overfitting on the data, with differing values applied to both the input features, the hidden memory vector, and the internal LMU memory vector. Redundant connections that did not affect accuracy were also removed. Parameter tuning was performed manually on the size of the hidden vectors, the presence of gates, and the amount of dropout in order to maximize accuracy while minimizing model size.

The InceptionNet preprocessed features are passed through a dropout layer before being input to an LMU cell, returning only the final layer for classification. The output is then fed through a dense softmax layer for classification. All LMU models also have an internal hidden unit dropout parameter of 0.5 and memory unit dropout parameter of 0.05.

5.6 Long Short-Term Memory Model Training

The default tensorflow implementation of the LSTM was used for our comparison model. The architecture of the model is identical to the LMU model, with the only difference being the LMU cell is swapped for an LSTM cell. The InceptionNet preprocessed sequence is fed into a standard LSTM layer with dropout returning only the final value of the sequence. The output from the LSTM layer is then fed through a dense softmax layer for classification. All LSTM models also have an internal dropout parameter of 0.5.

5.7 Results

Multiple models with a variety of parameters were trained on the UCF-101 dataset with identical train, test, validation splits for every model, enforced with a fixed seed for splitting. All models are trained for 1000 epochs, smaller faster training models are also trained for 4000 epochs. Due to computational constraints, we are not able to produce multiple runs for the different models. Both top 1, top 3, and top 5 categorical accuracies were used to determine performance as there is some variance between papers on which metrics to use.

Training time comparisons are based on epochs to train. As different models were trained in different GPU servers with different GPU loads while training, it is not possible to make any comparisons based on wall clock time taken.

5.7.1 3D Preprocessing

The initial models were trained on the output of the Google 3D InceptionNet with only 4 steps of 1024 features. However, it was quickly discovered that because the 3D InceptionNet was already trained for the purpose of action recognition, there was very little difference between the performance of the LSTM and the LMU. In general, regardless of the model placed as the final layer, we saw a top 1 accuracy of about $87 - 88\%$, top 3 accuracy of $95 - 96\%$ and top 5 accuracy of $97 - 98\%$. The original 3D InceptionNet by itself was able to achieve 98.0% top 5 accuracy so it is clear that the LMU and LSTM networks are not performing significant work, especially with just 4 time steps after preprocessing. As such, the results of these initial experiments were not meaningful to the comparison of the LMU and LSTM and not included.

One thing to note is that while both ended up with the same accuracies, the LSTM trained in fewer epochs than the LMU. This could be explained by the 3D InceptionNet being originally already trained to perform action classification on video. As such the bulk of the action classification work is already done by the 3D network. The LSTM architecture by default passes the information forward as is to the dense classification layer compared to the LMU which needs to first learn to decode from the compressed Legendre space.

Model	Hidden Units	Memory Units	Parameters
LSTM128	128	128	1,127,653
LSTM28	28	28	235,553
LSTM13	13	13	108,638

Table 5.1: LSTM Model Parameters for the 40 time step 2D InceptionNet preprocessed Dataset. Parameters refers to the total number of trainable parameters within the model. Note that in the standard LSTM model the Hidden and Memory Units are equal in size.

5.7.2 2D Preprocessing

In order to make a better comparison in performance for learning visual dynamics over time, we switched to training on the output of the Google 2D InceptionNet, which converts the 40 frames of video into 40 time steps of 2048 features. As the 2D inception network was trained to classify images rather than video, all the time dependent action classification work is done by the LMU and LSTM.

For the LSTM models (Table 5.1), we looked at a large model with 128 hidden units and 1.1 million parameters overall and a smaller model with 28 hidden units and 235 thousand units overall. Larger networks did not have a significant impact on accuracy, reducing the size of the network further below the LSTM28 caused the model to fail to learn the task.

For the LMU models (Table 5.2), we looked at a large model with 128 hidden units, 128×4 memory units, and 701 thousand parameters overall, as well as a gated model with 128 hidden units, 64×4 memory units, and 619 thousand parameters. While these have fewer parameters than the LSTM model, larger models did not improve performance and were significantly slower to train so were not included. On the other end, we also examined a smaller models with just 64 hidden units, 32×4 memory units, and just 221 thousand parameters to compare to the smaller LSTM model. Furthermore, we also decreased the parameter count even further with 32 hidden units, 16×4 memory units, and just 106 thousand parameters to test the limits of the architecture.

The final test results after training can be seen in Table 5.3. For the larger models, the LSTM outperformed the LMU in all cases. With 71% accuracy compared to 69% in the Top 1 category and just over 1% improvement in the top 3 and top 5 categories. The

Model	Hidden Units	Memory Units	Memory Order	Theta	Parameters
LMU128	128	128	4	16	701,177
LMUGated128	128	64	4	16	619,321
LMU64	64	32	4	16	221,605
LMU32	32	16	4	16	106,265
LMUGated24	24	12	4	16	106,265

Table 5.2: LMU Model Parameters for the 40 time step 2D InceptionNet preprocessed Dataset. Parameters refers to the total number of trainable parameters within the model. Note that because each memory unit is composed of its own memory vector of length Order, the size of the entire memory vector for the LMU is the product of the Memory Units and Memory Order. Because the LMU encodes its memory with precalculated optimal matrices, the model is able to have much larger memory vectors for similar total parameter counts compared to the LSTM.

Gated LMU performed almost identically to the LMU128 model with half the memory units, only performing slightly worse in top 3 accuracy.

For the models in the 200k parameter range, the LMU outperformed the LSTM across every metric after 1000 epochs, even with fewer parameters. As both models had not flattened out on training accuracies and training loss, and both models were small enough to train quickly, we are able to extend the training to 4000 epochs instead. Both models improved in this case and the LSTM was able to catch up in performance, slightly beating the LMU in the top 5 category by 0.4%, and losing to the LMU in top 3 and top 1 by 0.7% and 1.8% respectively.

For the smallest models with just over 100 thousand parameters, we see the LMU32 significantly outperform the LSTM13 across every metric. Over 10% worse performance in the top 1 after 1000 epochs, 8.5% better in the top 3, and 7.4% better in top 5. These differences continue through 4000 epochs of training, with a 9.9% advantage in top 1 accuracy, 5.7% in top 5, and 3.6%. In fact, the LMU is more accurate across every metric after 1000 epochs than the LSTM after 4000 epochs.

Model	Top 1 Accuracy	Top 3 Accuracy	Top 5 Accuracy	Parameters
LMU128	68.97%	85.22%	89.67%	701,177
LMUGated128	68.99%	84.59%	89.59%	619,321
LMU64	62.27%	80.88%	86.57%	221,605
LMU64-4000	67.67%	83.74%	88.56%	221,605
LMU32	56.02%	75.24%	82.50%	106,265
LMU32-4000	64.18%	79.64%	85.41%	106,265
LMUGated24	48.80%	68.68%	76.81%	103,741
LMUGated24-4000	57.53%	75.72%	83.21%	103,741
LSTM128	71.14%	86.68%	90.81%	1,127,653
LSTM28	55.60%	74.56%	82.00%	235,553
LSTM28-4000	65.90%	83.08%	88.99%	235,553
LSTM13	45.88%	66.77%	75.14%	108,638
LSTM13-4000	54.30%	73.95%	81.81%	108,638

Table 5.3: 2D Models Test Accuracy. Top N Accuracy refers to whether the model predicts the correct category within its top N predictions for each trial. All models were trained for 1000 epochs, unless tagged with '-4000', in which case they were trained for 4000 epochs.

5.8 Discussion

The LMU architecture was developed to optimally encode information over a large number of time steps without the issue of vanishing or exploding gradients, so it is not surprising that the LSTM was still able to beat the LMU when it comes to the larger models with just 40 time steps, well within the limits of the LSTM. Perhaps with larger dataset with longer length videos, we will be able to see the advantages of the LMU come into play. Furthermore, while the task involves classifying actions from video, many of the actions are simple enough that a human can identify the action from just a single frame, so it is possible that classification becomes just a matter of remembering a few time steps as opposed to learning the dynamics over the entire time frame.

However, for smaller network sizes the LMU was able to perform much better than the LSTM across the majority of categories. The LMU not only trained faster, showing significantly better performance after 1000 epochs, but also topped out higher with on average better results than the LSTM after 4000 epochs, while requiring 6% fewer parameters overall. The LMU32 model was also able to slightly beat the LSTM28 model after 1000 epochs, showing that the LMU is much more efficient as the parameter count is pushed lower.

The reason for this difference in performance in smaller networks can be seen in the architectures of the two networks. The LMU optimally encodes temporal information using precalculated matrices derived from the Legendre polynomials, whereas the LSTM encodes temporal information using gates, each of which requires learning large matrix multiplications in order to calculate. The result is the LSTM has to spend a significant portion of its parameter budget on the matrices for computing the gates required to construct its memory vector whereas the LMU precomputes those matrices for free. This allows the LMU to have much larger internal vectors for the same number of parameters. Although high end performance is important, real world applications often have restrictions in hardware requirements and running time, creating a need for efficient smaller models.

While the LSTM128 model has 60% more parameters than the LMU128 model, the LMU128 model is able to have the same number of hidden units and a memory vector 4 times larger, 128×4 compared to 128. In similar parameter counts of just over 200 thousand, the LMU is able to have double the hidden units, 64 compared to 28, and over four times the memory units, 32×4 compared to 28. For the 100 thousand parameter models, we have the LSTM able to have just 13 hidden and memory units, while the LMU can have 32 hidden units and 16×4 memory units. This parameter efficiency allows the LMU to do much more with less compared to the LSTM.

The gated LMU has potential, matching the performance of the LMU128 model with a slightly lower parameter count. The difference being the gated LMU had half the memory units while requiring new matrices to produce the input gate. The similar levels of performance suggest that the gate makes up for the missing memory units by allowing the network to be more selective in the information being encoded into the memory. However, the results are close enough that it is impossible to say if the parameter trade off in reducing internal unit sizes is worth it for the addition of gates in the LMU. Similar to how the LSTM gates allow it to remain stable for longer lengths of timesteps compared to the simpler RNNs, the LMU gates may lead to greater performance in longer time step data sets.

Chapter 6

Conclusion

In this thesis we examined the Spatial Semantic Pointer and the Legendre Memory Unit as biologically plausible methods of representing and processing continuous spatial and temporal information respectively within artificial intelligence applications. For spatial representations, we demonstrated that SSPs can be used to represent space in a spatial reasoning model. We showed that we can process visual image maps into an SSP memory vector, store and retrieve spatial information, manipulate space in memory, as well as perform search and scanning tasks. Our SPA model was able to perform the spatial relations reasoning task by extracting spatial information from memory and using that spatial information to locate target objects based on their spatial relationship to the original object.

Furthermore, our model succesfully replicated Kosslyn's famous map scanning experiment by performing in memory scanning from origin object to target object. While Vector Symbolic Architectures have been proposed in the past to model human working memory, these experiments show that Spatial Semantic Pointers in particular can be used to model human spatial cognition.

For temporal representations, we expanded the original LMU architure to be able to have multi-dimensional input signals and multi-dimensional internal memory states. We demonstrated that this new multi-dimensional LMU is able to match the LSTM in representing visual data over time. While the LSTM performed slightly better with high parameter counts, we showed that the LMU is able to outperform the LSTM as the parameter count shrinks. Furthermore, the internal structure of the LMU allows it to obtain much lower total parameter counts even with larger internal state vectors. We also showed that the LMU can be augmented with gates similar to the LSTM, at the cost of an increased number of parameters. While this did not improve accuracy in our experiments, it is possible that these gates will have a larger impact for data with much larger numbers of timesteps.

Our model examined the use of Spatial Semantic Pointers as a continuous representation of space within a neural network model. All spatial reasoning after the initial data preprocessing by neural network was performed using the SSP operations. It is possible to encompass the entire pipeline within a deep neural network model by training a neural network to encode visual spatial information directly into high dimensional SSP vectors and also training the neural network to perform the SSP vector operations required for spatial reasoning.

The dataset used was a simple visual map built from randomly placing MNIST images onto a visual field. This was chosen to ensure that the results were focused on the SSP architecture's ability to process spatial data in memory and not the neural network's ability to pre-process the image. Many object detection neural networks are able to process more complex images into objects and their coordinates. The SSP architecture can easily be added onto these models to enable them to output SSP memory vectors that can be used to perform spatial relations reasoning tasks.

We compared the LMU to the LSTM on the UCF101 dataset with sets of 40 frames extracted from short videos a few seconds long. However, the LMU truly shines over the LSTM when the number of timesteps is much higher, on the order of 1000s or more. Furthermore, the actions within the dataset, such as golfing or typing, can often be identified by a human from just a single frame. As such it is hard to make any comments on the ability of either model to capture context over extended time. Given much more computational power, it would be interesting to train comparison models on much longer videos that are minutes or hours long where robust memory and the ability to capture context would be much more important. For example the detection of irregularities in security tapes where relevant events, such as someone leaving with an item they did not enter with, could occur minutes or hours apart.

References

[1] Introduction to the semantic pointer architecture. https://www.nengo.ai/nengo-spa/user-guide/spa-intro.html.

[2] O. Abdel-Hamid, A. Mohamed, H. Jiang, L. Deng, G. Penn, and D. Yu. Convolutional neural networks for speech recognition. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 22(10):1533–1545, 2014.

[3] S. Antol, A. Agrawal, J. Lu, M. Mitchell, D. Batra, C. L. Zitnick, and D. Parikh. Vqa: Visual question answering. In *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 2425–2433, 2015.

[4] Timothy J. Buschman, Markus Siegel, Jefferson E. Roy, and Earl K. Miller. Neural substrates of cognitive capacity limitations. *Proceedings of the National Academy of Sciences*, 2011.

[5] J. Carreira and A. Zisserman. Quo vadis, action recognition? a new model and the kinetics dataset. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4724–4733, 2017.

[6] Narsimha Chilkuri and Chris Eliasmith. Parallelizing legendre memory unit training, 2021.

[7] Chuankun Li, Pichao Wang, Shuang Wang, Yonghong Hou, and Wanqing Li. Skeleton-based action recognition using lstm and cnn. In *2017 IEEE International Conference on Multimedia Expo Workshops (ICMEW)*, pages 585–590, 2017.

[8] Junyoung Chung, Caglar Gulcehre, Kyunghyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. In *NIPS 2014 Workshop on Deep Learning, December 2014*, 2014.

[9] Ronald A. Finke and Steven Pinker. Spontaneous imagery scanning in mental extrapolation. *Journal of Experimental Psychology: Learning, Memory, and Cognition*, 8(2):142–147, 1982.

[10] R. Gayler. Vector symbolic architectures answer jackendoff's challenges for cognitive neuroscience. *ArXiv*, abs/cs/0412059, 2004.

[11] Ross Gayler and Simon Levy. A distributed basis for analogical mapping. 01 2009.

[12] M. Haldekar, A. Ganesan, and T. Oates. Identifying spatial relations in images using convolutional neural networks. In *2017 International Joint Conference on Neural Networks (IJCNN)*, pages 3593–3600, 2017.

[13] Matt Harvey. Five video classification methods implemented in keras and tensorflow. https://github.com/harvitronix/five-video-classification-methods, Sep 2017.

[14] John Haugeland. *Artificial Intelligence: The Very Idea*. Cambridge: MIT Press, 1985.

[15] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, November 1997.

[16] M. Ibbotson and B. Krekelberg. Visual perception and saccadic eye movements. *Curr Opin Neurobiol*, 21(4):553–558, Aug 2011.

[17] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.

[18] Denis Kleyko. *Pattern Recognition with Vector Symbolic Architectures*. PhD thesis, 03 2016.

[19] Brent Komer, Terrence C. Stewart, Aaron R. Voelker, and Chris Eliasmith. A neural representation of continuous space using fractional binding. In *41st Annual Meeting of the Cognitive Science Society*, Montreal, QC, 2019. Cognitive Science Society.

[20] Yu Kong and Yun Fu. Human action recognition and prediction: A survey. *CoRR*, abs/1806.11230, 2018.

[21] Stephen M. Kosslyn. *Image and Brain.* MIT Press, 1984.

[22] S. Li, W. Li, Chris Cook, C. Zhu, and Y. Gao. Independently recurrent neural network (indrnn): Building a longer and deeper rnn. *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5457–5466, 2018.

[23] Xinyu Li, Yanyi Zhang, Chunhui Liu, Bing Shuai, Yi Zhu, Biagio Brattoli, Hao Chen, Ivan Marsic, and Joseph Tighe. Vidtr: Video transformer without convolutions, 2021.

[24] Giuseppe Liotta, Roberto Tamassia, and Ioannis G. Tollis. *Graph Algorithms and Applications 5.* World Scientific Publishing Company, 2006.

[25] Pankaj Malhotra, Anusha Ramakrishnan, Gaurangi Anand, Lovekesh Vig, Puneet Agarwal, and Gautam Shroff. Lstm-based encoder-decoder for multi-sensor anomaly detection, 2016.

[26] A.K. Pani and G.P. Bhattacharjee. Temporal representation and reasoning in artificial intelligence: A review. *Mathematical and Computer Modelling*, 34(1):55–80, 2001.

[27] Tony A Plate. Holographic reduced representations. *IEEE Transactions on Neural networks*, 6(3):623–641, 1995.

[28] Murray Shanahan. The frame problem. https://plato.stanford.edu/archives/spr2016/entries/frame-problem/, Feb 2004.

[29] Khurram Soomro, Amir Roshan Zamir, and Mubarak Shah. Ucf101: A dataset of 101 human actions classes from videos in the wild, 2012.

[30] C. Szegedy, Wei Liu, Yangqing Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1–9, 2015.

[31] László Tóth. Combining time- and frequency-domain convolution in convolutional neural network-based phone recognition. pages 190–194, 05 2014.

[32] M.V. Valueva, N.N. Nagornov, P.A. Lyakhov, G.V. Valuev, and N.I. Chervyakov. Application of the residue number system to reduce hardware costs of the convolutional neural network implementation. *Mathematics and Computers in Simulation*, 177:232–243, 2020.

[33] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[34] Aaron R. Voelker. *Dynamical Systems in Spiking Neuromorphic Hardware*. Phd thesis, University of Waterloo, 2019.

[35] Wei Wang, Cheng Chen, Yizhou Wang, Tingting Jiang, Fang Fang, and Yuan Yao. Simulating human saccadic scanpaths on natural images. pages 441 – 448, 07 2011.

[36] Xianyuan Wang, Zhenjiang Miao, Ruyi Zhang, and Shanshan Hao. I3d-lstm: A new model for human action recognition. *IOP Conference Series: Materials Science and Engineering*, 569:032035, 08 2019.

[37] Eric Weiss, Brian Cheung, and Bruno Olshausen. A neural architecture for representing and reasoning about spatial relationships. *Proceedings of the International Conference on Learning Representations (ICLR), Workshop Trak,*, 2016.

[38] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Łukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Greg Corrado, Macduff Hughes, and Jeffrey Dean. Google's neural machine translation system: Bridging the gap between human and machine translation, 2016.