Efficient navigation using a scalable, biologically inspired spatial representation

Brent Komer (bjkomer@uwaterloo.ca)
Chris Eliasmith (celiasmith@uwaterloo.ca)
Centre for Theoretical Neuroscience, University of Waterloo
Waterloo, ON, Canada, N2L 3G1

Abstract

We present several experiments demonstrating the efficiency and scalability of a biologically inspired spatial representation on navigation tasks using artificial neural networks. Specifically, we demonstrate that encoding coordinates with Spatial Semantic Pointers (SSPs) outperforms six other proposed encoding methods when training a neural network to navigate to arbitrary goals in a 2D environment. The SSP representation naturally generalizes to larger spaces, as there is no definition of a boundary required (unlike most other methods). Additionally, we show how this navigational policy can be integrated into a larger system that combines memory retrieval and self-localization to produce a behavioural agent capable of finding cued goal objects. We further demonstrate that explicitly incorporating a hexagonal grid cell-like structure in the generation of SSPs can improve performance. This biologically inspired spatial representation has been shown to be able to produce spiking neural models of spatial cognition. The link between SSPs and higher level cognition allows models using this representation to be seamlessly integrated into larger neural models to elicit complex behaviour.

Keywords: Semantic Pointer Architecture; spatial semantic pointer; spatial representation; navigation; policy learning

Introduction

An important part of solving any task involves having an effective representation. All animals are embodied in space and interact with other animals and objects around them. In order for an animal to act intelligently and survive in this world their brains must be able to form useful representations of the space around them. The same information merely presented in a different manner to a neural network can produce profoundly different results (Bengio, Courville, & Vincent, 2013).

A proposed method for encoding spatial information in a cognitive system is Spatial Semantic Pointers (SSPs) (Komer, Stewart, Voelker, & Eliasmith, 2019; Lu, Voelker, Komer, & Eliasmith, 2019). The key question we investigate in this work is whether this encoding designed for spiking neural models of spatial cognition is also an effective representation for artificial neural networks in general. We compare the effectiveness of SSPs to various other commonly used representations in machine learning for solving spatial tasks.

We begin with an overview of the mathematical formulation of SSPs along with an improvement to this formulation inspired by grid cells. We then show how encoding spatial information using SSPs provides an excellent representation for a neural network to learn a goal-finding policy. Finally, we demonstrate how the learned policy can be incorporated into a larger system to produce behaviour which solves the navigation task.

Spatial Semantic Pointers

Often when building cognitive models it is useful to be able to construct complex relations from simpler components. Vector Symbolic Architectures (VSAs) are one class of approaches that define algebras over high dimensional vector spaces, where each vector in the space can represent a particular concept. VSAs essentially allow vectors to be used as 'slots' and 'fillers'; where a slot is some property (e.g. colour) and the filler is the value of that property (e.g. red). An operator is required to perform the binding between the two vectors, in our case we use circular convolution (denoted by $\circledast$) (Plate, 1995).

The Semantic Pointer Architecture (SPA) (Eliasmith, 2013) proposes a means of implementing VSAs for explaining cognitive behaviour in biologically plausible spiking networks. Inspiration for SSPs comes from models of serial working memory, where position in a list is encoded by an 'index' vector bound to itself a number of times equal to the position (Choo & Eliasmith, 2010). A vector B can be repeatedly bound with itself $k-1$ times as follows:

$$B^k = \underbrace{B \circledast B \circledast \ldots \circledast B}_{B \text{ appears } k \text{ times}}. \tag{1}$$

While the equation above applies to positive integer values of k, the definition can be expanded to allow k to be any real value, by using the fact that convolution in the time domain becomes multiplication in the frequency domain. This operation, which we call 'fractional binding', is defined as follows:

$$B^k = \mathcal{F}^{-1}\left\{\mathcal{F}\{B\}^k\right\}, \quad k \in \mathbb{R}, \tag{2}$$

where $\mathcal{F}\{\cdot\}$ is the Fourier transform, and $\mathcal{F}\{B\}^k$ is an element-wise exponentiation of a complex vector—analogous to exponentiation using fractional powers (e.g., $b^{2.5}$)—permitting k to be real. In this work we choose B to be a unitary vector, meaning its length does not change with multiple bindings to itself and its inverse is equal to its approximate inverse.

This representation is extended to multiple spatial dimensions by repeating Equation 2 with a different semantic

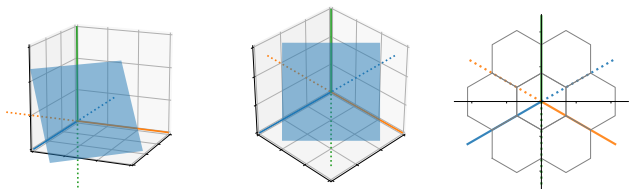

Figure 1: Projection to Hexagonal Coordinates. Left: Plane in 3D. Middle: Perspective aligned to the normal vector of the plane. Right: Coordinate system of the plane with principal axes of the 3D system overlaid.

pointer for each dimension (referred to as 'axis vectors') and then binding all of the resulting vectors together. For the 2D case, the SSP that represents the point (x,y) is defined as the vector resulting from the function:

$$S(x,y) = X^x \circledast Y^y, \tag{3}$$

where X and Y are fixed unitary semantic pointers, x and y are reals, and we are using fractional binding as defined by Equation 2. In the language of 'slots' and 'fillers', SSPs can be seen as a continuous slot that can be filled with an object to represent an object at a location.

SSPs have been shown to be a powerful model of spatial cognition. They have the capability to represent arbitrary objects at any location in continuous space. They can also represent collections of objects and regions of space. Many spatial operations can be performed on a memory containing SSPs, such as translating items in memory, querying object relations and mental image scanning (Komer et al., 2019; Lu et al., 2019). SSPs can be implemented efficiently in a spiking neural network using the methods of the Neural Engineering Framework (Eliasmith & Anderson, 2003).

An Improved Spatial Representation

In the initial introduction of SSPs, the vectors corresponding to the x and y axes were chosen randomly and independently. While this method performed well in many applications, there is still room for improvement. In this section we demonstrate how changing the generation method of the axis vectors can lead to improvements in the representation.

Inspiration from Grid Cells

Neurons in the medial entorhinal cortex have been shown to produce hexagonally symmetric firing across spatial locations (Hafting, Fyhn, Molden, Moser, & Moser, 2005).

One way to interpret a hexagonal coordinate system is as a standard 3D coordinate system projected onto a 2D plane. By choosing the plane to have a normal vector of $\begin{bmatrix}1 & 1 & 1\end{bmatrix}$, movement along any of the principal axes in the 3D system corresponds to moving along the 120 degree apart principal axes in the hexagonal coordinate system. A visualization of this relation is shown in Figure 1.

In the original SSP formulation, two high-dimensional vectors, X and Y, are chosen to define the encoding. In the modified formulation for a hexagonal coordinate system, three high-dimensional unitary vectors, X, Y, and Z, are used. The convolutional exponents for each of these vectors are determined by transforming a given 2D coordinate into a 3D coordinate constrained to lie on a 2D plane. This transformation is carried out by multiplying the 2D coordinate by a 2x3 matrix. This matrix is formed by stacking two orthogonal unit vectors that lie on the plane. The hexagonal SSP formulation is shown in Equation 4.

$$\begin{aligned} S_3(x,y) &= X^{x'} \circledast Y^{y'} \circledast Z^{z'} \\ \begin{bmatrix}x' & y' & z'\end{bmatrix}^T &= \begin{bmatrix}x & y\end{bmatrix}^T A_{2x3} \end{aligned} \tag{4}$$

It is important to note that there is no additional computational cost associated with defining the representation using three axis vectors, as an equivalent two axis vector system can be constructed and used instead. The axis vectors of this equivalent system are defined in Equation 5.

$$\begin{aligned} X &= S_3(1,0) \\ Y &= S_3(0,1) \end{aligned} \tag{5}$$

These vectors can be thought of as spanning a 2D subspace of the 3D space being represented. Since the original three axis vectors are chosen randomly, this operation can also be interpreted as a change in the distribution where possible axis vectors are sampled from and the new X and Y vectors are no longer independent.

One way to measure the effectiveness of an encoding method is to compare the representation of a location to all other locations. Ideally if the representations are similar, the locations should be similar (i.e. nearby) and if the representations are different, the corresponding locations should be different. Similarity is measured as the cosine distance between two vectors. A comparison of the similarity heatmaps generated from the two approaches encoding the coordinate (0, 0) are is shown in Figure 2.

In the original formulation there are clear noise patterns aligned with the axes, visible in both a single choice of axis vectors as well as averaging across many choices. For the hexagonal formulation, there is less distinct structure in the noise pattern and only when averaged across many choices of axis vectors does a faint hexagonally symmetric pattern arise. When comparing the heatmaps to an ideal Gaussian fit, the original formulation results in an RMSE of 0.043 while the hexagonal formulation has an RMSE of 0.019.

Navigation

The first experiment consists of generating a policy to solve a navigation task. The primary goal is to assess the effectiveness of the representation for a behavioural policy. Since an optimal policy can be computed for this task, supervised learning can be used to train a network to match this desired policy. This allows training to converge much faster

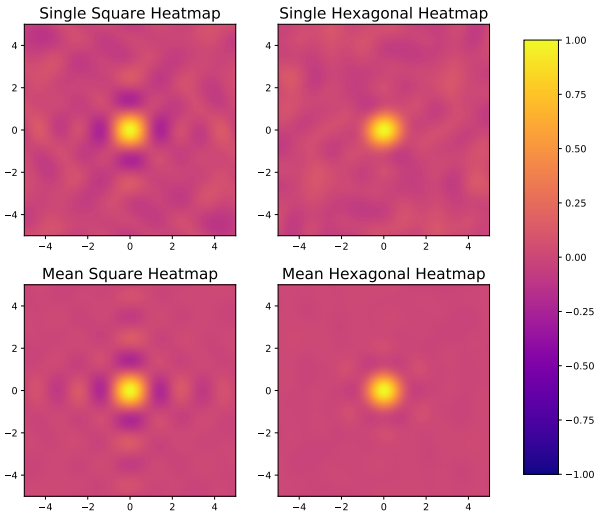

Figure 2: Similarity Heatmap of Different SSP Formulations. Top: Heatmaps for a single randomly generated SSP. Bottom: Mean heatmaps across 50 randomly generated SSPs. Left: Original formulation using two independent unitary vectors. Right: Hexagonal formulation using three independent unitary vectors projected onto a 2D space.

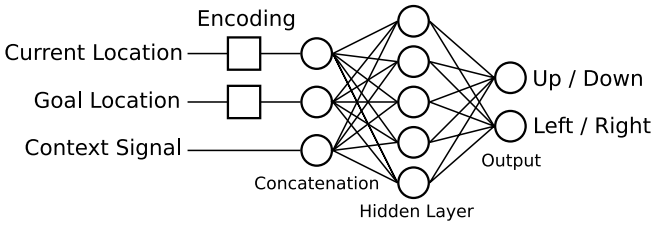

Figure 3: Navigation Policy Diagram

and avoids many of the difficulties in parameter sensitivity and reproducability with reinforcement learning approaches.

Policy Network

The policy network produces a mapping from states to actions. In this case, the state consists of the location of the agent, the location of the current goal, and a context signal indicating which of several mazes the agent is currently exploring. The output of the network is a two dimensional vector corresponding to the direction the agent should move from the current location to get to the goal. A diagram of the network structure is shown in Figure 3.

Optimal Policy

The optimal policy for a given goal location is computed using a modified version of Dijkstra's search algorithm (Dijkstra et al., 1959). The space is discretized into a 64 by 64 array. For each element in the array, a continuous direction is computed indicating the direction to move in order to optimally reach the goal. Optimality is defined as traversing the least total distance and only moving through free space and not obstacles.

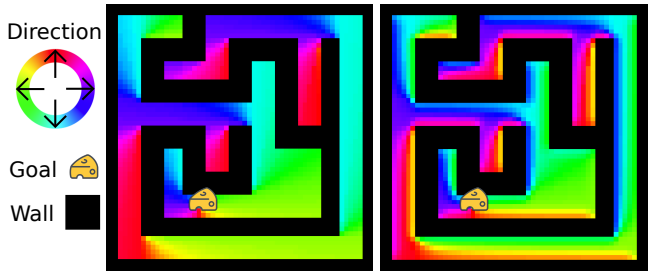

Figure 4: Ground Truth Optimal Policy. Left: Policy computed using search algorithm. Right: Policy with additional bias added to avoid contact with walls.

Traversing the least distance involves paths that cut corners very tightly and stay near walls. This is not ideal behaviour in the real world, where there is noise involved in motion, and the agent takes up physical space. To account for this, an additional bias is added to avoid contact with walls and to traverse closer to the center of hallways. This bias is created by applying an energy function to all obstacles that pushes the optimal policy direction away from the obstacle, similar to a potential field planner (Latombe, 2012). The energy function chosen was a Gaussian with a σ of 0.25 and a magnitude of 0.75.

Figure 4 shows a visualization of the optimal policy computed for a given goal location in a maze. The colour of each pixel indicates the direction an agent should move from that location in order to reach a specific goal. Black indicates impassable obstacles.

Other Encoding Methods

In order to gauge the effectiveness of the SSP encoding, it must be compared to other commonly used methods. In this work we compare against Radial Basis Functions (RBF), One-Hot encoding, Tile-coding (Albus, 1975), learning the encoding, and directly using 2D coordinates with no encoding. An additional comparison is made with a random encoding to create a baseline for all of these methods.

One parameter common to all encoding methods is the dimensionality of the encoding. For comparisons, this parameter is held constant across all methods and set to a value of 256.

For RBF, the basis function used is a 2D Gaussian with a σ of 0.75. This value was chosen by performing a parameter sweep on the task and choosing the result that produced the lowest error. The locations of the Gaussians are chosen uniformly at random within the bounds of the environment. For one-hot encoding, the environment is discretized into 256 evenly sized square bins, resulting in a 16 by 16 grid. For tile-coding the environment is discretized into four overlapping 8 by 8 grids with random offsets. These values were chosen to maintain a total dimensionality of 256. For the learned encoding, an additional layer with 256 neurons is added to the network. This layer is applied to both the current location and the goal location with weights shared, to ensure a consistent

encoding. The random encoding uses the input coordinates as the seed to generate a random 256D unit vector.

Experiments

In all experiments the context signal is set to a 256D random unit vector unique to each maze layout. When concatenated with the 256D encoding of the current location and 256D encoding of the goal location, the total size of the input vector is 768 (or 260 for the no encoding case). The hidden layer size is set to 256 neurons. A ReLU activation function is used between the input layer and the hidden layer. The output is a linear readout from the hidden layer.

The network is implemented in PyTorch (Paszke et al., n.d.). It is trained using the Adam optimizer (Kingma & Ba, 2015) with a learning rate of 0.001 for 50 epochs on a training dataset of 100,000 data points. Initial experiments were run for 500 epochs, but performance on a validation dataset plateaued by 50 epochs, indicating that any additional training would only result in overfitting. All results shown are from an independently generated test dataset of 100,000 data points.

One task parameter that was explored is the number of unique environments that the network must learn to navigate. The more environments, the greater the difficulty of the task as the network is fixed in size and the same learned weights are used in every environment. Experiments were performed using 10, 25, and 50 different environments. Each environment layout is generated randomly following one of two distributions from mazelab (Zuo, 2018). The 'maze' generation method creates environments with winding narrow hallways (example in Figure 11) and the 'blocks' generation method creates environments with open space and scattered obstacles (example in Figure 10). A mix of styles is used to prevent overfitting to a particular kind of environment. All environments are generated such that a path exists from any free space to any other free space.

Performance is measured by comparing the output of the network to the optimal policy. Root Mean Squared Error (RMSE) is calculated based on the difference is angle between the movement direction computed by the network and the optimal direction. The angle between two directions can be different depending on whether a clockwise or counter-clockwise rotation is used to align the directions. In terms of error, we are always interested in whichever rotation produces the smallest angle. A general way to obtain the smallest angular distance between two angles is shown in Equation 6.

$$\theta_e = \min(|\theta_p - \theta_t - 2\pi|, |\theta_p - \theta_t|, |\theta_p - \theta_t + 2\pi|) \quad (6)$$

Where θ_e is the error, θ_p is the predicted angle, and θ_t is the target angle. Both the predicted and target angle are constrained to be between $-\pi$ and π. This equation enforces that the error is within the same range. This equation can be vectorized in Python to compute the error for all data points efficiently.

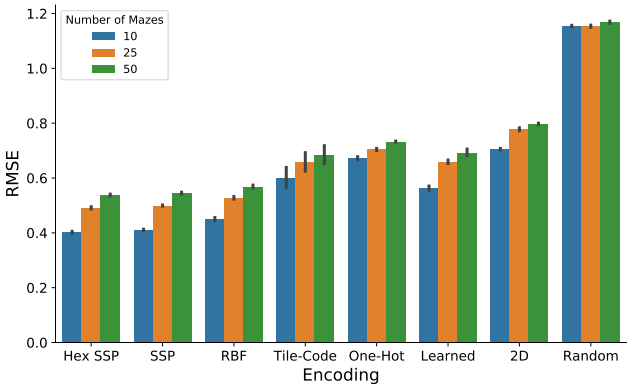

Figure 5: Comparison of Encoding Methods

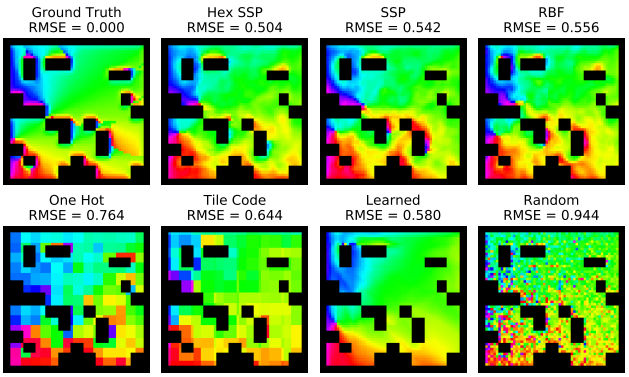

Figure 6: Policy Visualization. The colour of each pixel corresponds to the output direction computed by the policy for an agent at the location of the pixel. Shown is one goal location in one particular maze for all encoding methods. The image is created using a mix of training and test samples to tile the space.

A comparison of the different encoding methods is shown in Figure 5. 'Hex SSP' refers to the formulation in Equation 4. While the SSP encodings performed the best overall, some of the other methods are still comparable on this task. For the 50 maze case, Hex SSP produced an RMSE of 0.538 (SD 0.004), SSP 0.545 (SD 0.004), and the next closest encoding was RBF with 0.568 (SD 0.007). The differences in mean performance are all significant; Student's t-test between Hex SSP and SSP (p=0.02079), between SSP and RBF (p=0.00021). However, the SSP method becomes uniquely effective when generalizability is taken into account, as detailed in the next section.

Generalization

A natural advantage of using a SSP encoding of space is there are no explicitly defined limits of the space that need be represented. When using a method such as Radial Basis Functions or Tile-Coding, boundaries must be chosen for the space to be represented. Any region outside of the boundary cannot

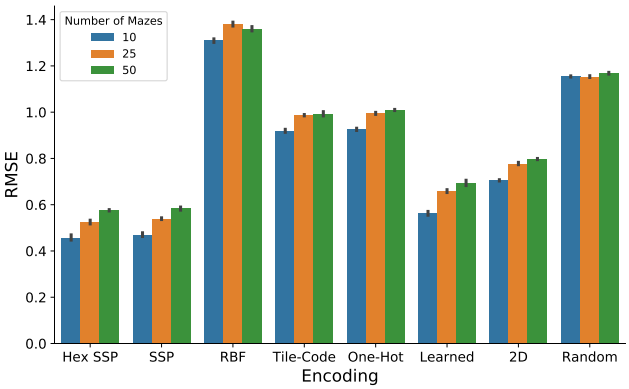

Figure 7: Performance Outside of Bounds

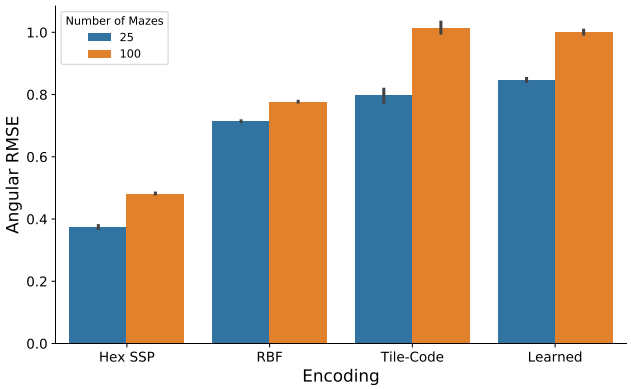

Figure 8: Performance on Larger Task

be encoded effectively without changing the representation to include additional basis functions or tiles. In contrast, SSPs can represent any location and the dimensionality of the SSP does not have to increase to incorporate more space.

To illustrate this point, experiments were conducted where the encoding is designed for a particular region of space, but the network learns from a larger region that goes outside of these bounds. In this experiment the encodings were designed for a space of half the width and half the height of the entire maze. For SSPs, the only available parameter to change is the scaling of the coordinates used as the convolutional power. The scaling that is used in all previous experiments was increased by a factor of two, to indicate being chosen for a smaller space. As can be seen in Figure 7, the SSP encoding method outperforms all other methods by a large margin.

To further test the scalability of this method, an additional experiment is performed using significantly larger environments and a larger network. To keep the optimal policy computation tractable, sets of smaller environments from the previous experiments are tiled together to form the larger environments. The same pre-computed optimal policy can be used for training within each sub-environment. Experiments are conducted on a square tiling of 25 and 100 environments. The encoding dimension is set to 1024 and network hidden size to 2048. Training is conducted on a single large environment, so the context signal is not required. Results are shown in Figure 8. Parameters for RBF and Tile Code are scaled to account for the larger environment size, without scaling performance is extremely poor. The learned encoding method receives normalized input coordinates. The Hex SSP method requires no parameter change and its definition remains that same as the smaller environment case. Visualization of the policy learned for one sub-environment of the 25 tiling case is shown in Figure 9.

Integrated System

A policy for generating actions is just one component of a navigation system. In this section we demonstrate how this component can fit into a larger circuit that allows an agent to navigate towards semantically specified goals. Crucially this system contains a memory for storing remembered objects and their locations, a method of retrieving the location of an object from memory, a method of estimating the agent's current location given its visible surroundings, and choosing motor actions based on estimated current location and estimated goal location. A diagram of this system is shown in Figure 10.

Memory

The agent's memory is implemented as a single Semantic Pointer containing the summation of all goal objects and their locations as an SSP.

$$M = \sum_{i=1}^{m} OBJ_i \circledast S_i, \tag{7}$$

The resulting vector M is then normalized to maintain unit length.

The estimated location of a given goal object $\hat{S}_{goal}$ is retrieved by performing a circular convolution between the memory M and the inverse of the Semantic Pointer corresponding to the goal OBJ_{goal}^{-1}. The result of the circular convolution can also be passed through a cleanup function f to remove noise from the result.

$$\hat{S}_{goal} = f(M \circledast OBJ_{goal}^{-1}), \tag{8}$$

In this work, f is a denoising autoencoder trained on pairs of clean and noisy SSPs. The network will be most effective when the noise used in training matches the true distribution that the network will encounter. To facilitate this, a series of memories are created as in Equation 7 with random unit vectors as objects and SSPs encoding random 2D points within an environment. To retrieve a noisy vector, a query is performed to get the location of a random object in this memory. A database of 100,000 such pairs of noisy and clean vectors was created. The network has a single hidden layer with a ReLU activation function. The hidden layer is set to be the same size as the input. The loss function used during training

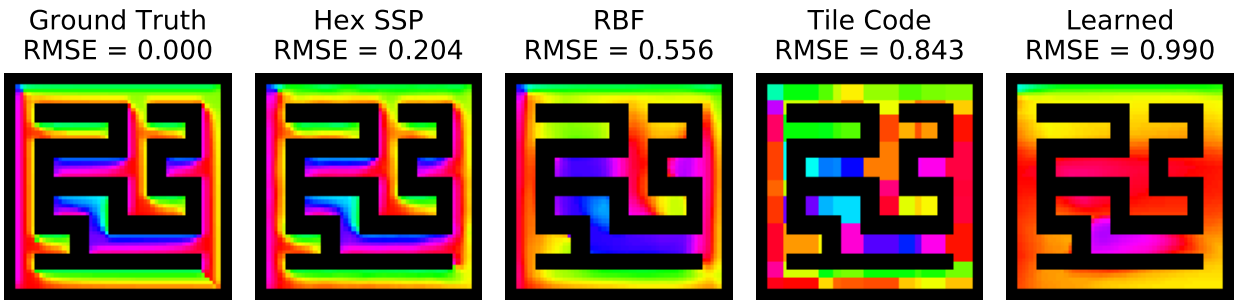

Figure 9: Policy Visualization on Larger Task

is the cosine distance between the vector produced by the network from the noisy input and the true target. Since SSPs are unit length and defined by their direction, this loss function works well on this problem.

Localization

The agent does not inherently know its own location within its cognitive map, it must estimate its location from its surroundings. In this simplified 2D world, the agent's visual system consists of a series of distance measures to nearby walls, effectively forming a picture of the visible geometry of its surroundings. A context signal is also provided to the network to help disambiguate geometry shared across environments. For example, many environments may contain a long hallway which would return the same sensor readings. The context signal is meant to represent additional information an agent would have in the real world, such as colours, textures, or other cues that would signal which environment the agent is currently in. The output of the localization subsystem is a SSP corresponding to the agent's estimated current location.

The localization subsystem is also implemented as a neural network and trained in a supervised fashion. A database of 100,000 samples of locations within an environment and corresponding distance measurements and SSP encoding are used for training. For this problem, the best results were achieved using a loss function that is a summation of the mean squared error and cosine distance.

Policy

The goal representation from the memory subsystem is combined with the agent location from the localization subsystem and fed into the policy network along with the context signal. This is the same network as depicted in Figure 3 except that no explicit encoding step is required as the outputs of the memory and localization subsystems are already SSPs.

Results

A series of trials is conducted with different goals being cued on each trial. A trial ends when the agent successfully reaches the cued goal object, or after 1000 time steps (100 seconds) have elapsed without reaching the goal.

At the beginning of a trial the agent is placed in a random location in a randomly selected environment. A single

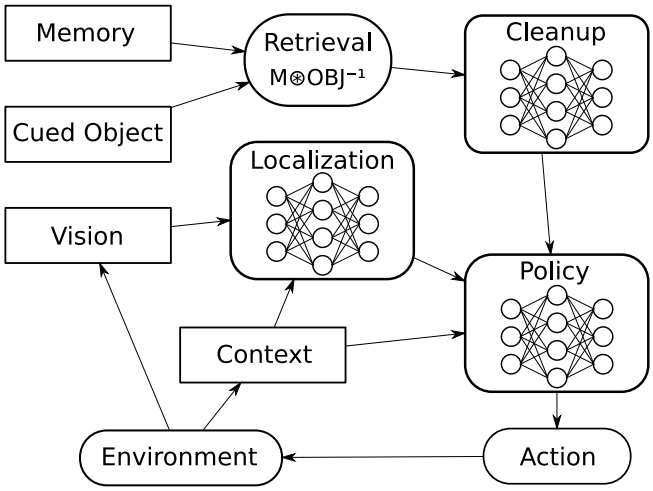

Figure 10: Full Navigation Task Diagram

256D memory is initialized to contain a set of possible goal objects bound with their location SSPs. At each time step it is given a 256D context signal corresponding to its current environment, a 256D semantic pointer corresponding to the goal object, and 36 sensor measurements corresponding to the distance to obstacles at an array of angles from the agent evenly covering 360 degrees. Every time step the network will output a 2D action corresponding to the x and y velocity the agent will move with. Gaussian noise is added to these motor commands. A set of trajectories from a particular start location and multiple cued goals is shown in Figure 11.

Discussion

We have demonstrated that Spatial Semantic Pointers provide an excellent means of encoding continuous valued location information in a neural network. This biologically inspired representation permits the use of continuous structures, as opposed to standard discrete data structures, including VSAs, typically used in cognitive models. The use of circular convolution as the binding operation defining this representation leads to nice integration with the SPA, providing a natural method of linking to spiking neural models of higher level cognition.

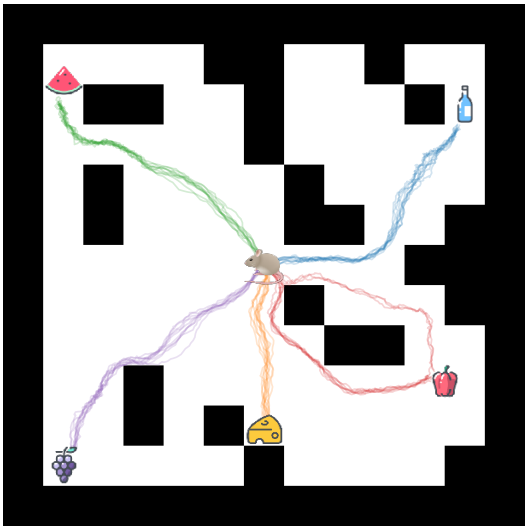

Figure 11: Goal-Finding Behaviour. This example shows the behaviour of the agent depending on which food item it is told to find. 10 trials for each goal are overlaid on the environment. The colour of the trajectory corresponds to which goal was cued. On each trial in this example, the agents starts at the location indicated by the mouse icon.

Many properties of SSPs are beneficial to machine learning systems operating on spatial data. No explicit boundary of the space being represented needs to be defined, allowing straightforward generalization to larger spaces. There is no discrete binning of the space, allowing a high degree of precision if necessary. The magnitude of the encoded value is fixed and independent of the magnitude of the coordinate being represented, allowing the network to scale to handling larger spaces without requiring larger weights.

Empirical results show superior performance on a navigational policy learning task compared to commonly used encoding methods for spatial information. While performance is only slightly better than other methods when all methods are optimized for a small space being trained over, this difference becomes much larger when the space is expanded. This indicates that the SSP representation is more general and less dependent on parameter settings.

We also show that incorporating hexagonal structure inspired by grid cells into the SSP representation can further improve the results on these spatial tasks. An area of future work is to explore this link to biology in more detail.

Another area of future work is exploring how sophisticated navigation behaviour can be learned in an online fashion and how it can adapt to changing environments. In this work, each subsystem of the navigation network is trained individually to achieve a sub-goal. It would be interesting to see how well a system trained end to end would perform and what the internal representations of that system would look like.

All source code required to reproduce these experiments and generate all figures in this paper are made publicly available on GitHub.[1]

Acknowledgments

This work was supported by CFI and OIT infrastructure funding as well as the Canada Research Chairs program, NSERC Discovery grant 261453, AFOSR grant FA9550-17-1-0026 and NSERC graduate funding.

References

Albus, J. S. (1975). Data storage in the cerebellar model articulation controller (cmac). *Journal of Dynamic Systems, Measurement, and Control*, *97*(3), 228–233.

Bengio, Y., Courville, A., & Vincent, P. (2013). Representation learning: A review and new perspectives. *IEEE transactions on pattern analysis and machine intelligence*, *35*(8), 1798–1828.

Choo, X., & Eliasmith, C. (2010, 08/2010). *A spiking neuron model of serial-order recall.* Portland, OR: Cognitive Science Society.

Dijkstra, E. W., et al. (1959). A note on two problems in connexion with graphs. *Numerische mathematik*, *1*(1), 269–271.

Eliasmith, C. (2013). *How to build a brain: A neural architecture for biological cognition*. Oxford University Press.

Eliasmith, C., & Anderson, C. H. (2003). *Neural engineering: Computation, representation, and dynamics in neurobiological systems*. MIT press.

Hafting, T., Fyhn, M., Molden, S., Moser, M.-B., & Moser, E. I. (2005). Microstructure of a spatial map in the entorhinal cortex. *Nature*, *436*(7052), 801–806.

Kingma, D., & Ba, J. (2015). Adam: A method for stochastic optimization in: Proceedings of the 3rd international conference for learning representations (iclr'15). *San Diego*.

Komer, B., Stewart, T. C., Voelker, A. R., & Eliasmith, C. (2019). A neural representation of continuous space using fractional binding. In *41st annual meeting of the cognitive science society.* Montreal, QC: Cognitive Science Society.

Latombe, J.-C. (2012). *Robot motion planning* (Vol. 124). Springer Science & Business Media.

Lu, T., Voelker, A. R., Komer, B., & Eliasmith, C. (2019). Representing spatial relations with fractional binding. In *41st annual meeting of the cognitive science society.* Montreal, QC: Cognitive Science Society.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., ... Chintala, S. (n.d.). Pytorch: An imperative style, high-performance deep learning library. In *Advances in neural information processing systems 32*.

Plate, T. A. (1995). Holographic reduced representations. *IEEE Transactions on Neural networks*, *6*(3), 623–641.

Zuo, X. (2018). *mazelab: A customizable framework to create maze and gridworld environments.* `https://github.com/zuoxingdong/mazelab`. GitHub.

[1] `https://github.com/ctn-waterloo/cogsci2020-ssp-nav`