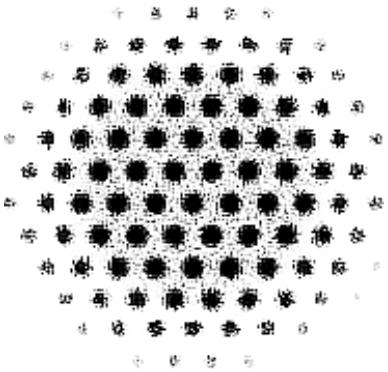
ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

GRID CELLS

THE EMERGENCE OF GRID CELL REPRESENTATIONS IN AUTO-ENCODING AND RECURRENT NEURAL NETWORKS



FLAVIO SCHNEIDER

BACHELOR THESIS

ADVISED BY XUN ZOU, PROF. ANGELIKA STEGER

SPRING 2020

Abstract

A lot of neurons in the mammalian hippocampus have been found to be responsible for the encoding of location. Grid cells are the most recently discovered, but their functionality is still not completely understood. One of the leading hypothesis is that grid cells represent an internal map used during navigation. As a way to better understand the tessellating pattern generated by grid cells, we train many different machine learning models to perform path integration and auto-encoding, from which quasi-periodic patterns resembling grid cells emerge. We find that applying noise during path-integration forces a more robust representation, and that regularization imposes a more sparse representation, both factors can have a metabolic analogous such as neuronal interference and energy efficiency, and are crucial for the emergence of those patterns. Furthermore, similarly to recent work, we observe that place cells facilitate the emergence of quasi-periodic patterns during auto-encoding even without path integration, suggesting that grid cells might have a more important role in the encoding of position than with navigation as previously thought.

Contents

Introduction 1

How does the brain build an internal representation of space? This is a question that philosophers, physicians, epistemologists, and many other scientists have long sought to answer. During the past half-century, thanks to the joint work of physicians and computational neuroscientists, much has been discovered. With the increasing technological advances, physicians are now able to inspect the activity of the brain with higher precision, where computational neuroscientists can exploit the insurgence of machine learning and more computational power to build better models. Models that are in turn used to better understand the discoveries of physicians.

In our case we will explore the brain from the computational perspective, more precisely we will try to understand a specific type of neuron found in the mammalian brain, a grid cell. Grid cells are believed to be one of the main building blocks of our internal representation of space, and a very important component needed to perform spatial navigation [1] .

1: Our ability to go from point A to point B even if those points are not directly connected by a straight line.

Before going into the specifics of grid cells, it's helpful to start with a general anatomical overview of all the different types of neurons surrounding grid cells, how those were discovered, and what is their function.

1.1 Place Cells

In 1971 neuroscientists John O'Keefe and John Dostrovsky were working on the hippocampus of rats, a region of the mammalian brain known to be extremely important for learning and memory. It was previously observed that patient and rats with a removed or damaged hippocampus had an impaired episodic memory[2] and the ability to navigate an environment. Suggesting that the hippocampus plays a central role in our internal representation of space. Not much before 1971, the only way to study the brain was to find patients with damaged regions or to remove/inhibit certain regions from the brains of rats and see how their function would change. However, in O'Keefe's lab, the first experiments with manually placed electrodes were possible, capable of reading the firing of *single neurons* in moving rats.

2: The memory used to memorize events, we use it to answer who, what, where, and why questions.

In [1] O'Keefe et al., they listened to 76 hippocampal neurons in 23 different rats. In many, they found previously known neurons that fired with the highest frequency when the rat was aroused by something: orienting, sniffing, walking, sitting, grooming, and some that fired unpredictably. But in 8 of those neurons, they found something surprising, each one of them was firing the most only when the rat was around a certain location in the testing platform. About half of those fired the most when the rat was around a certain location *and* some other constraint was also satisfied (*e.g.* arousal or facing in a certain direction). This was

[1]: O'Keefe et al. (1971), 'The hippocampus as a spatial map: preliminary evidence from unit activity in the freely-moving rat.'

the first time that they found individual neurons firing selectively for a single location, a surprising fact considering that the hippocampus is not directly connected to any sensory input or motor outputs. Those newly discovered cells were named *place cells*.

After the discovery, O'Keefe went on a long scientific journey to understand place cells and the role of the hippocampus. In 1978, he published one of the most influential books on the subject [2] , which described a broad range of inherent topics. Starting from the past philosophical ideas of space (relative *vs* absolute), going to the psychological observations of how human and animals solve problems in space, and concluding in a broad chapter summarizing the discovery of place cells with detailed documentation on hippocampal anatomy. The last chapter contains a lot of speculation on how place cells are formed and how the hippocampus might build a cognitive map. One of the ideas is that rats can use the *theta rhythm*, when moving, to measure distance. Theta rhythm is a name given to waves of activation in the hippocampus observed to have activity oscillating at a steady frequency of around $6 - 10Hz$, hypothesized to be used as a metric of time for spatial navigation. The theory still was missing many things, one of them is that to navigate space you need a sense of direction, or a frame of reference, a problem that the book did not settle.

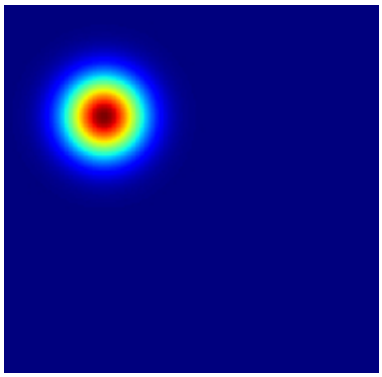


Figure 1.1: Reproduced firing pattern of a single place cell.

[2]: O'keefe et al. (1978), *The hippocampus as a cognitive map*

1.2 Head-Direction Cells

Around 10 years later, neuroscientist Jeffry Taube conducted an experiment [3] in a region of the hippocampus close to where place cells were discovered. By recording the activity of 239 cells, it was observed that 61 of them only fired when the rat was pointing in a certain direction with respect to the testing platform. Those cells were named *head-direction cells*. Head-direction cells are narrowly tuned to the orientation of the head with respect to the vertical axis (*i.e.* they are independent of the head tilt [4]) and do not depend directly on any environmental input. Similarly to how a compass works [5] , head-direction cells depend on the vestibular system, part of the inner ear that helps us balance. Like place cells, they provide a sparse representation of some measure, which in this case it's the orientation. Again, it was not understood how the mechanism responsible for the configuration of those cells worked, but many experiments were conducted in the following years. In 1998, 8 years after the discovery, Taube published a paper[6] to summarize the results of many experiments on head direction cells, and their possible role in our internal representation of space.

[3]: Taube et al. (1990), 'Head-direction cells recorded from the postsubiculum in freely moving rats. I. Description and quantitative analysis'

[4]: Raudies et al. (2015), 'Head direction is coded more strongly than movement direction in a population of entorhinal neurons'

[5]: Blair et al. (1996), 'Visual and vestibular influences on head-direction cells in the anterior thalamus of the rat.'

[6]: Taube (1998), 'Head direction cells and the neurophysiological basis for a sense of direction'

1.3 Egocentric, Allocentric, and Path Integration

At the time, since the discovery of place cells, all studies have been conducted with the underlying assumption that we navigate space with an *egocentric* frame of reference, updated through external inputs. Or, in simpler words, that every time we move we observe what is around

us and we can deduce our new position based on some specific points called landmarks, always referencing the positioning of an object with respect to a frame of reference centered on the self. In his book[2] O'Keefe, had previously suggested that another type of internal representation could be possible, but that idea was mostly ignored until 1996, when [7] McNaughton et al. suggested a theory based on an *allocentric* model. An allocentric model is similar to an internal GPS where the frame of reference is not centered on the person but distributed[3] and the information about the position of an object is encoded with respect to the position of other objects. Our representation of space is then updated through a process called *path integration*, where instead of changing our position completely based on environmental landmarks, we only use landmarks to infer the initial position and then each subsequent step updates our position in the map with self-motion cues. This theory could explain how we are able to abstract space without referring to any landmarks, for example in a dark room when the lights are turned off. Head-direction cells are the first clear component of this internal map, as they provide an internal frame of reference for the head orientation which can be manipulated by changing environmental cues but does not directly depend on it. Place cells are somewhere in-between as they can be initialized with environmental cues *i.e.* landmark navigation, but when those are removed the remapping doesn't always take place, but instead, the internal path is used. This hypothesis raises many questions, how, given head-direction cells and place cells position, is the brain able to perform path integration to continuously update the current position? Where is the map actually stored? How are the internal reference frames updated? And how can we use those reference frames to enable long-range navigation?

[7]: McNaughton et al. (1996), 'Deciphering the hippocampal polyglot: the hippocampus as a path integration system.'

3: Similarly to the difference between right (egocentric) and east (allocentric).

1.4 Grid Cells

In 2005 grid cells were discovered [8] . It was observed that 211 cells in the entorhinal cortex of 14 rats were exhibiting a striking spatial organization. In each of those neurons when the rat moved around in the testing platform, tessellating triangular patterns spanning the entire arena were recorded. This discovery could answer one of the previous questions, raising the possibility that grid cells are the permanent internal map, updated through path integration, that does not depend on environmental cues.

In this thesis, we will try to understand this most recent discovery on grid cells, by training different machine learning models to perform spatial navigation and auto-encoding of the current position.

[8]: Hafting et al. (2005), 'Microstructure of a spatial map in the entorhinal cortex'

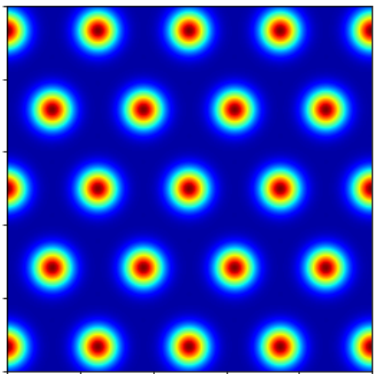


Figure 1.2: Reproduced firing pattern of a single grid cell.

Models 2

In this chapter we will summarize the models used in the experiments and previous work, describing the functionality, purpose, and tradeoffs. Mainly we will work with artificial neural networks, and try to imitate the behavior of real neurons. Artificial neural networks (ANN) using back-propagation are very different from how the brain works as they only model broadly the firing frequency of a real neuron, providing a rough sketch of the brain's connectivity. The main reason we use ANNs is not to have a model as similar as possible to the brain, but to have a theoretical model that can automatically learn the solution to a problem when different constraints are applied and enough data is given. We expect that if the brain has evolved to use a certain representation, we might also find the same representation in an artificial neural network when the same constraints are applied. The application of artificial neural networks as neural information processing models has been explored [9] and already applied in the study of place cells and grid cells [10, 11] .

[9]: Van Gerven et al. (2017), 'Artificial neural networks as models of neural information processing'

[10]: Cueva et al. (2018), 'Emergence of grid-like representations by training recurrent neural networks to perform spatial localization'
[11]: Banino et al. (2018), 'Vector-based navigation using grid-like representations in artificial agents'

More precisely we will consider linear ANNs to model the encoding and decoding of information, and different sequence-to-sequence models such as recurrent neural networks (RNNs) and the similar variants long-short term memory (LSTMs) and gated recurrent units (GRUs) [12–14] . Sequence-to-sequence models will be used to handle the encoding of time-dependent information, where an input sequence is provided and an output sequence is generated, and each item of the sequence represents a different time-step. For example, an agent walking in a room knows some information about its position at each time-step, and given this information it can learn to generate new information depending on the data it's trained on. This will be particularly useful to simulate information processing during path integration.

[12]: Rumelhart et al. (1986), 'Learning representations by back-propagating errors'
[13]: Hochreiter et al. (1997), 'Long short-term memory'
[14]: Chung et al. (2014), 'Empirical evaluation of gated recurrent neural networks on sequence modeling'

2.1 Fully Connected Linear

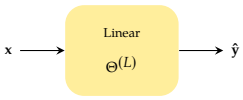


Figure 2.1: Fully connected linear model block.

Fully connected linear layers can be used as encoders if trained with a high dimensional representation as the input state and a lower dimensional representation as the output state. Similarly we can use linear layers as decoders if the input state has a lower dimension to the output state. More formally, given the input batch $\mathbf{x} \in \mathbb{R}^{B \times IN}$, then output prediction batch $\hat{\mathbf{y}} \in \mathbb{R}^{B \times OUT}$ is computed as:

$$\hat{\mathbf{y}} = \mathbf{x}\mathbf{W} + \mathbf{b} \tag{2.1}$$

where $\Theta^{(L)} = (\mathbf{W}, \mathbf{b})$ are trainable parameters with $\mathbf{W} \in \mathbb{R}^{IN \times OUT}$, $\mathbf{b} \in \mathbb{R}^{1 \times OUT}$ that can be adjusted using back-propagation [15] . We will always use B as the batch-size, IN as the size of the input vectors, and OUT as the size of the output vectors. If $IN > OUT$ then we will have an encoder, else a decoder.

[15]: Hecht-Nielsen (1992), 'Theory of the backpropagation neural network'

2.2 RNN Cell

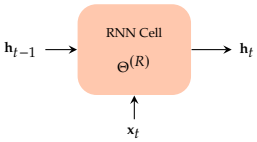


Figure 2.2: RNN Cell block.

A RNN cell takes two inputs at each timestep, the first is the previous hidden state which represents the memory of the cell, and the second is the input vector which represents some value that can be trained to modify the previous hidden state. The new memory is then returned as a new hidden state. More formally, given the input $\mathbf{x}_t \in \mathbb{R}^{B \times IN}$ and the previous hidden state $\mathbf{h}_{t-1} \in \mathbb{R}^{B \times H}$ batches, the current hidden state batch $\mathbf{h}_t \in \mathbb{R}^{B \times H}$ is computed as:

$$\mathbf{h}_t = \sigma(\mathbf{x}_t \mathbf{W}_{IN} + \mathbf{h}_{t-1} \mathbf{W}_H + \mathbf{b}) \tag{2.2}$$
$$= \sigma\left([\mathbf{x}_t, \mathbf{h}_{t-1}]\mathbf{W} + \mathbf{b}\right) \tag{2.3}$$

where $\Theta^{(R)} = (\mathbf{W}, \mathbf{b})$ are trainable parameters with $\mathbf{W} \in \mathbb{R}^{(IN+H) \times H}$, $\mathbf{b} \in \mathbb{R}^{1 \times H}$, and σ is a sigmoid activation function applied element-wise on the input vector.

$$\sigma(\mathbf{z}) := \frac{e^{\mathbf{z}}}{e^{\mathbf{z}} + 1} \tag{2.4}$$

The idea of RNN cells, is that we can chain many cells together[1] by connecting the hidden state of the previous cell to the next cell, such that the current state depends on both the previous state and the current input [12]. One important part of this process is that the trainable parameters are shared between all RNN cells, so that each cell uses the same rule to change the hidden state memory in different time-steps. Linking many RNN cells together is useful if we want to encode sequence information. By initializing the first hidden layer with the initial memory (either randomly, to zero, or in some more clever way), we can then feed as input, at each time-step, the value in the sequence that we want the model to be trained on, depending on the output hidden sequence. After we trained the model, the hidden layer memory will contain some H dimensional representation, it's common practice to connect this hidden state to a linear encoder/decoder to extract the desired information and train the model on that information instead of the hidden layer directly.

1: Reason why we are talking about time-steps and not just inputs/outputs.

The problem with this initial version of RNNs is that when linking many cells together the training becomes unstable in what is called vanishing/exploding gradient problem [16, 17] . The idea is that with this architecture, when back-propagating the error to correct the trainable parameters, we multiply subsequent gradients because of the chain rule. If we link together many cells (*i.e.* use a long sequence length) and the

[16]: Hochreiter (1998), 'The vanishing gradient problem during learning recurrent neural nets and problem solutions'
[17]: Pascanu et al. (2013), 'On the difficulty of training recurrent neural networks'

gradient has small values < 1 the gradient will vanish, similarly if they have large values > 1 the gradient will explode. In both cases, this will disrupt the training of the network. There are some workarounds in [17] but they only partially solve the problem, it's thus common practice to use architectures that are more complicated but don't suffer from this problem, such as LSTMs and GRUs when training models with medium or long sequences.

2.3 LSTM Cell

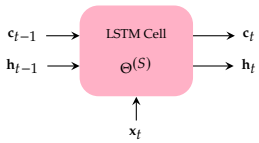


Figure 2.3: LSTM Cell block.

LSTM cells [13] are similar to RNNs, they use a more complex architecture which is able to learn long sequences. To do so they use an additional state, called the cell state, which is responsible for storing memory for long sequences without being disrupted. The general idea is that LSTM contain different gates, two of which are the input and forget gates, trained to add or remove information to the cell state depending on what information should be persisted for many time-steps and on which information should be discarded. The hidden state only contains outputs that are necessary for the next time-step, and is controlled by one gate called the output gate. More formally, given the input $\mathbf{x}_t \in \mathbb{R}^{B \times IN}$, the previous hidden state $\mathbf{h}_{t-1} \in \mathbb{R}^{B \times H}$, and the previous cell state $\mathbf{c}_{t-1} \in \mathbb{R}^{B \times C}$ batches, then current hidden and cell state batches are computed as:

$$\mathbf{i}_t := \sigma([\mathbf{x}_t, \mathbf{h}_{t-1}]\mathbf{W}_I + \mathbf{b}_I) \tag{2.5}$$
$$\mathbf{f}_t := \sigma([\mathbf{x}_t, \mathbf{h}_{t-1}]\mathbf{W}_F + \mathbf{b}_F) \tag{2.6}$$
$$\mathbf{g}_t := \tanh([\mathbf{x}_t, \mathbf{h}_{t-1}]\mathbf{W}_G + \mathbf{b}_G) \tag{2.7}$$
$$\mathbf{o}_t := \sigma([\mathbf{x}_t, \mathbf{h}_{t-1}]\mathbf{W}_O + \mathbf{b}_O) \tag{2.8}$$
$$\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \mathbf{g}_t \tag{2.9}$$
$$\mathbf{h}_t = \mathbf{o}_t \odot \tanh(\mathbf{c}_t) \tag{2.10}$$

[2] where $\Theta^{(S)} = (\mathbf{W}_I, \mathbf{b}_I, \mathbf{W}_F, \mathbf{b}_F, \mathbf{W}_G, \mathbf{b}_G, \mathbf{W}_O, \mathbf{b}_O)$ are trainable parameters and $\mathbf{i}_t, \mathbf{f}_t, \mathbf{g}_t, \mathbf{o}_t$ are intermediate gates called input, forget, modulation, and output gate.

2: $\odot$ is the product applied element-wise, also called Hadamart product.

The downside of LSTMs is that if we want to inspect its memory we now have two different states instead of only a single one as in RNNs. The following more recent model will merge the best of both worlds, long-sequence training and single memory state, in what is called a gated recurrent unit (GRU).

2.4 GRU Cell

GRU are similar to LSTM, but instead of using the input/forget gate to update the cell state, they use only an update gate to add and remove information from the hidden state, and a reset gate that decides how much

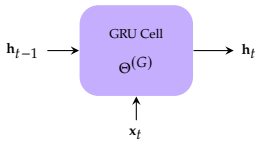


Figure 2.4: GRU Cell block.

of the previous information should be removed. The update gate only tells how much information should flow to the hidden layer, where the reset gate how much information should be provided for the computation of the new hidden layer in what is called the new gate. Given the input $\mathbf{x}_t \in \mathbb{R}^{B \times IN}$ and the previous hidden state $\mathbf{h}_{t-1} \in \mathbb{R}^{B \times H}$ batches, then current hidden state batch $\mathbf{h}_t \in \mathbb{R}^{B \times H}$ is computed as:

$$\mathbf{r}_t := \sigma([\mathbf{x}_t, \mathbf{h}_{t-1}]\mathbf{W}_R + \mathbf{b}_R) \tag{2.11}$$

$$\mathbf{z}_t := \sigma([\mathbf{x}_t, \mathbf{h}_{t-1}]\mathbf{W}_Z + \mathbf{b}_Z) \tag{2.12}$$

$$\mathbf{n}_t := \tanh\left(\mathbf{x}_t\mathbf{W}_{N1} + \mathbf{b}_{N1} + \mathbf{r}_t \odot (\mathbf{h}_{t-1}\mathbf{W}_{N2} + \mathbf{b}_{N2})\right) \tag{2.13}$$

$$\mathbf{h}_t = (1 - \mathbf{z}_t) \odot \mathbf{n}_t + \mathbf{z_t} \odot \mathbf{h}_{t-1} \tag{2.14}$$

where $\Theta^{(G)} = (\mathbf{W}_R, \mathbf{b}_R, \mathbf{W}_Z, \mathbf{b}_Z, \mathbf{W}_{N1}, \mathbf{b}_{N1}, \mathbf{W}_{N2}, \mathbf{b}_{N2})$ are trainable parameters and $\mathbf{r}_t, \mathbf{z}_t, \mathbf{n}_t$ are intermediate gates called reset, update, and new gate.

Now with GRUs, we have what we wanted with the initial RNNs: long-sequence training and a single memory to inspect. GRUs also use fewer operations than LSTMs and thus are also slightly faster to train.

3 Previous Work

We will focus on some questions raised by [10, 11, 18] on the emergence of grid patterns during path integration using recurrent and linear models.

3.1 Cueva et al.

In [10] Cueva *et al.* a continuous-time recurrent neural network CTRNN[1] is trained to perform spatial navigation. They generate short random paths that start from the center of a virtual arena and train the CTRNN given the current velocity and angle as input to predict the new Cartesian position in the next time-step by decoding data from the hidden layer. Arenas with different shapes are selected such as squares, triangles, and hexagons, finding that square grid patterns appear in the hidden layer when training in the square arena, and triangular grid patterns in the triangular and hexagonal arena (similar to real grid cells). Furthermore, they also find band-like cells, which are cells that fire in bands of activation in the arena, and border-like cells, which are cells that fire only when the agent is close to the border of the arena. One important point is that for some unknown reason applying noise and regularization on the hidden layer is crucial for the emergence of grid patterns in the hidden layer, conjecturing that those constraints impose a sparse coding that manifests itself as grid cells. This is the first time that grid patterns emerge during path integration in a computational model.

[10]: Cueva et al. (2018), 'Emergence of grid-like representations by training recurrent neural networks to perform spatial localization'

1: A variant of a RNN that is trained with the Euler method instead of back-propagation and gradient descent.

3.2 Banino et al.

In [11] Banino *et al.* a LSTM is trained to to perform spatial navigation. The paper is split into two parts, the first that trains the LSTM for navigation, and the second that trains a reinforcement learning agent to use this positional encoding as the representation of a virtual complex environment with different walls, where the goal is to find rewards inside of the environment. We will focus on the first part, which can find triangular grid patterns in the hidden layer of the LSTM in square arenas (recall that [10] is only able to find grid patterns when the environment has a particular shape). Differently from [10] the LSTM is trained with paths that start in different positions and decodes the position into both place cells, instead of using Cartesian coordinates, and head direction cells. To summarize the model, the initial position is encoded into the initial hidden layer $\mathbf{h}_0$ using an encoder, then at each time-step, they provide the velocity and facing angle of the agent as input $\mathbf{x}_t$. The network is then trained to project the 128-dimensional hidden layer to a higher dimensional 512-dimensional representation which is subsequently decoded by two different networks, one to a place cell representation, and another to

[11]: Banino et al. (2018), 'Vector-based navigation using grid-like representations in artificial agents'

head-direction cell representation. By inspecting the 512-dimensional layer 129 units exhibit pattern resembling the ones generated by grid cells, some band-like cells, and some head direction cells. Another important point specified in [11] , is that regularization on the hidden layer in the form of dropout [19] is necessary for the grid pattern to appear.

[11]: Banino et al. (2018), 'Vector-based navigation using grid-like representations in artificial agents'

[19]: Srivastava et al. (2014), 'Dropout: a simple way to prevent neural networks from overfitting'

3.3 Dordek et al.

In [18] Dordek *et al.* a single layer neural network is trained to perform principal component analysis (PCA) [20] to encode the position of a moving agent. A random path of l points is generated, each step encoded into p place cells, then PCA is applied on the $l \times p$ representation. Initially, square grid patterns appear, but if a non-negativity constraint is applied on the network, triangular grid patterns emerge.

[18]: Dordek et al. (2016), 'Extracting grid cell characteristics from place cell inputs using non-negative principal component analysis'

[20]: Wold et al. (1987), 'Principal component analysis'

3.4 Questions

The two previous papers raise some questions on the nature of grid patterns. Is path integration really necessary for the emergence of grid patterns? If not, then it's possible that triangular grid patterns have nothing to do with path integration, but only with the encoding of information. A more recent result from [21] Sorscher *et al.* explains how the formation of grid patterns emerges mathematically when the non-negativity constraint is applied and additional assumptions on the structure of the pattern are made.

[21]: Sorscher et al. (2019), 'A unified theory for the origin of grid cells through the lens of pattern formation'

A second question is, why is regularization and noise so important when path integration is applied to recurrent models? Both [10, 11] claim that without perturbing the hidden layer with noise and dropout respectively grid patterns do not appear.

4 Experiments

In this section, we will describe the technical details of each model. In some models we will try to reproduce previous work, in others we will try to remove unnecessary parts of previous works in order to try to isolate the components responsible for path integration, or even for the emergence of grid patterns. Reproducing previous models should give us a more clear understanding of how grid cells emerge, but also give us the freedom to tune different parameters that are often presented in a single configuration. Lastly, we will build some new models to try to answer previous questions.

This section provides a detailed mathematical overview of the experiments, the actual implementation containing with all the hyperparameters is programmed in python using Jupyter notebooks, and can be found in at github.com/flavioschneider/bachelorthesis. All the machine learning models are implemented in PyTorch, a popular machine learning library which provides a good balance between customizability, simplicity, and speed. Furthermore, we decided to run the python Jupyter notebooks on Google Colab, which provides free access to GPUs to dramatically speed up the training of the PyTorch machine learning models.

4.1 Path Generation

Many models will be trained to do path integration, to do so we need a way to generate realistic trajectories inside a bounded arena. More specifically we will simulate, with our artificial agent, trajectories that are as similar as possible to rats [22] . The idea is that the agent will start at a location drawn uniformly at random $\mathbf{p}(0) \in [0, 1]^2$ in the arena, with velocity $v = 0$, rotational velocity $\omega = 0$, and a random facing vector $\boldsymbol{\alpha}(0) \in \mathbb{R}^2$. Where $\mathbf{p}(t) = [p_x(t)\ p_y(t)]^T$, $\boldsymbol{\alpha}(t) = [\alpha_x(t)\ \alpha_y(t)]^T$ are the position and facing direction vector at timestep t. The updated position at each timestep will be governed by the following equation:

$$p_x(t+1) = p_x(t) + \cos(\alpha_x(t)) \cdot v(t) \cdot \tau \tag{4.1}$$
$$p_y(t+1) = p_y(t) + \sin(\alpha_y(t)) \cdot v(t) \cdot \tau \tag{4.2}$$

[22]: Raudies et al. (2012), 'Modeling boundary vector cell firing given optic flow as a cue'

And the following equation for the angle:

$$\alpha_x(t+1) = \alpha_x(t) + w(t) \cdot \tau \tag{4.3}$$
$$\alpha_y(t+1) = \alpha_y(t) + w(t) \cdot \tau \tag{4.4}$$

Where $\tau = 0.05$ is a time constant. For the update of the velocity v and angular velocity ω we will consider two different cases:

Far from the border at each timestep, we will sample a new random velocity from a Rayleigh distribution[1]

$$\mathbb{P}(x;\delta) = \frac{x}{\delta^2}\exp\left(\frac{-x^2}{2\delta^2}\right) \tag{4.5}$$

centered at $\delta = 0.13$, and a random rotational velocity from a Gaussian distribution with mean $\mu = 0$ and standard deviation $\sigma = 5.7$ (330 degrees). This will give the agent a varying speed and direction.

Close to the border at each timestep, the speed is reduced with $v(t+1) = \frac{3}{4}v(t)$, and the angle is mirrored with respect to the wall. This makes sure that the agent never crosses the wall of the arena.

1: Looks like a Gaussian but only on the positive side, on the negative side, there is no tail and goes smoothly to zero. The idea is that the agent can have bursts where the velocity is higher but no negative velocity.

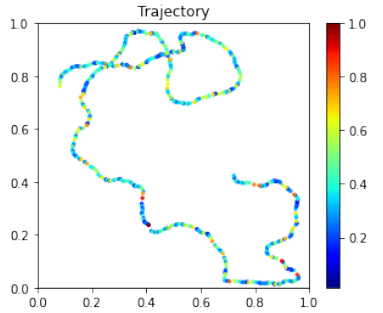


Figure 4.1: Generated path with 500 steps, blue is low velocity, and red is high velocity.

4.2 RNN Model

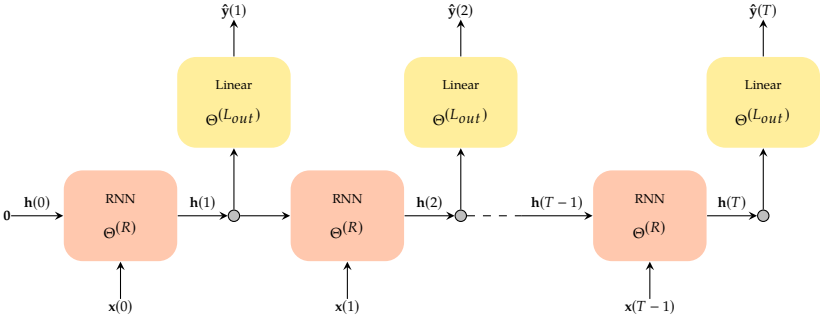


Figure 4.2: Unrolled Model.

Similarly to [10] , we train a recurrent neural network (RNN) to do path integration for $T = 500$ steps. At each time-step $t \in \{0, \dots, T-1\}$, using the previous path generation algorithm, we feed as input to the recurrent model the angle and velocity $\mathbf{x(t)} = [\alpha_x(t)\, \alpha_y(t)\, v(t)]^T \in \mathbb{R}^3$ at the current position. The hidden layer, $\mathbf{h}(t) \in \mathbb{R}^H$, at time-step $t = 0$ is initialized to the zero vector $\mathbf{0} \in \mathbb{R}^H$. The RNN will return at each time-step a hidden layer in $\mathbb{R}^H$ that will be transfomred using a linear layer to $\mathbb{R}^2$. Then the predicted output $\hat{\mathbf{y}}(t) \in \mathbb{R}^2$ will be trained using back-propagation with stochastic gradient descent to return the coordinates of the new position at time-step t, using the mean squared error between the predicted and correct position.

[10]: Cueva et al. (2018), 'Emergence of grid-like representations by training recurrent neural networks to perform spatial localization'

Loss function

$$L(\Theta;\mathbf{x},\mathbf{y}) := \frac{1}{T\cdot B\cdot 2}\sum_{t=1}^{T}\sum_{b=1}^{B}\sum_{i=1}^{2}\left(\hat{\mathbf{y}}_i(\Theta;t,b,\mathbf{x}) - \mathbf{y}_i(t,b)\right)^2 \tag{4.6}$$

Where $B = 64$ is the batch size, *i.e.* we will use batches of 64 different paths at the time to train the model. Where the goal will be to find the parameters $\Theta = (\Theta^{(R)}, \Theta^{(L_{out})})$ that minimize the loss function.

Goal

$$\Theta^* := \arg\min_{\Theta} L(\Theta;\mathbf{x},\mathbf{y}) \tag{4.7}$$

Note that we don't have a different model for each time-step as the weights $\Theta^{(R)}$ of the RNN are shared, similarly for the output linear layer parametrized by $\Theta^{(L_{out})}$. To visualize this idea in a more compact form from now on we will use the following recurrent diagram.

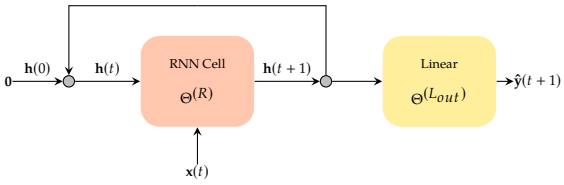


Figure 4.3: Compact Model. It's the exact same model but the time-steps are not unrolled.

This model is not able to do path integration as there are 2 issues. First [10] is not training the model with back-propagation and gradient descent, but with an hessian-free algorithm [23] . Hessian-free optimization is used to train an RNN without having the vanishing and exploding gradient problem [16] , to do so it uses an approximation of a second-order method. Second-order methods are theoretically better to train neural networks, however evaluating the Hessian matrix is usually intractable for large neural networks as the size grows with the square of the number of parameters in Θ. Hessian-free optimization uses a clever trick to avoid evaluating the hessian without giving up the advantages of second-order optimization. The problem with this method is that it is much more complicated to evaluate, and with LSTMs or GRUs [14, 24] , it's possible to train long-sequence recurrent models using first-order optimization without having vanishing/exploding gradients. Thus the idea is that we will use a slightly more complicated recurrent model (LSTMs or GRUs instead of RNNs) in favor of a less complicated optimization algorithm (stochastic gradient descent instead of hessian-free optimization).

[23]: Martens (2010), 'Deep learning via hessian-free optimization.'

[16]: Hochreiter (1998), 'The vanishing gradient problem during learning recurrent neural nets and problem solutions'

[24]: Gers et al. (1999), 'Learning to forget: Continual prediction with LSTM'
[14]: Chung et al. (2014), 'Empirical evaluation of gated recurrent neural networks on sequence modeling'

The second issue is that [10] uses paths that always start from the center of the arena, and thus setting the initial hidden layer $\mathbf{h}(0)$ to the zero vector (which is supposed to encode the initial position), doesn't give any information about the starting position to our model (which starts from a random position).

4.3 GRU Model

To fix the previous problems we will take inspiration from [11] . To solve the first issue we will use, as mentioned before, a GRU. GRUs are recurrent models very similar to RNN, the advantage is that (like LSTMs) they use an additional internal gate to avoid the vanishing/exploding gradient problem. However, differently from LSTMs, which are used in [11], GRUs are more lightweight as they use a smaller number of operations and parameters. The performance is usually almost the same depending on the application. To solve the second problem we will add an additional linear layer at the beginning that takes the coordinates of the initial position and is trained to encode it into the initial hidden layer.

[11]: Banino et al. (2018), 'Vector-based navigation using grid-like representations in artificial agents'

We will train this model the same way as the previous one but this time $\Theta := (\Theta^{(L_{in})}, \Theta^{(G)}, \Theta^{(L_{out})})$, the goal and loss function will also be the same.

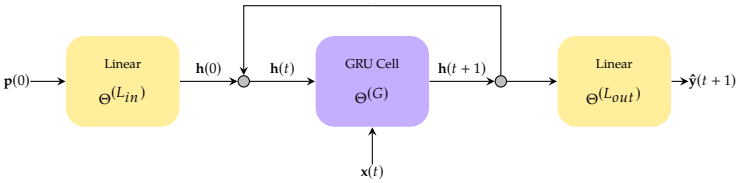


Figure 4.4: Model with GRU path integration and initial position encoding.

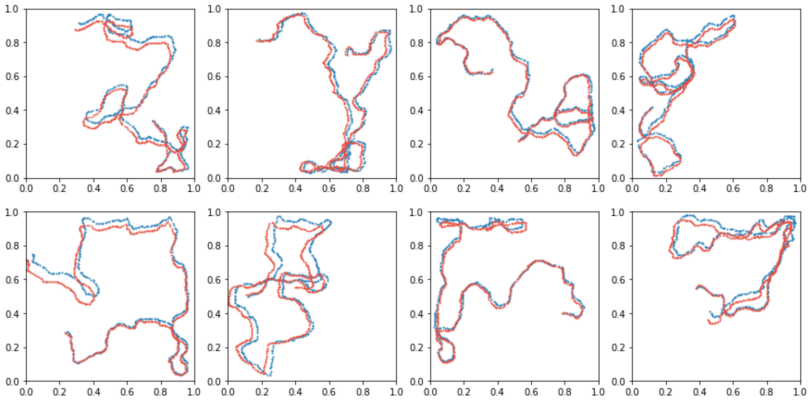


Figure 4.5: Path integration predictions: blue the real path, red the predicted path.

Path integration works correctly with a stable mean squared error of 0.002 after only 50 epochs. The predicted path usually starts out very precisely and then as the number of steps increases it shifts slightly from the correct position. This is expected since the model has access only to the initial starting point and errors accumulate.

The idea is that the model *must* use the hidden layer H-dimensional representation to store the current location in order to successfully path integrate. If $\mathbf{h}$ didn't encode the current position then the model would have no way of evaluating the new coordinates correctly using only the current angle and velocity. The goal of building this path integration mechanism is to then inspect this hidden layer, which mathematically is just a H-dimensional vector, and check how the model encodes the position. Note that $\mathbf{h}$ must store the position but it could also store other types of information that the model learns to be useful for path integration.

4.4 Hidden Layer Inspection

Inspection Given vectors $\mathbf{h}_1, \ldots, \mathbf{h}_n$ and vectors $\mathbf{p}_1, \ldots, \mathbf{p}_n$ where $\mathbf{h}_i$ is the H-dimensional hidden layer vector that must encode the 2-dimensional position $\mathbf{p}_i$, we will return H matrices (images) displaying the activation of each individual unit of the hidden vectors. To do so, we discretize the arena into a grid of $m \times m$ cells $C_{u,v} = \left\{\mathbf{p} \in \mathbb{R}^2 \mid p_x \in \left(\frac{u}{m}, \frac{u+1}{m}\right] \wedge p_y \in \left(\frac{v}{m}, \frac{v+1}{m}\right]\right\}$, with $0 \leq u, v < m$. Then the activation of cell u, v at unit $j \in \{1, \ldots, H\}$ is given by:

$$A_{u,v,j} = \frac{1}{|\{i \mid \mathbf{p}_i \in C_{u,v}\}|} \sum_{i:\mathbf{p}_i \in C_{u,v}} |h_{i,j}| \tag{4.8}$$

more simply, we average the activation of the absolute value of each cell for each unit[2] . This is necessary since the paths are generated randomly

2: This process is analog to placing an electrode to listen for one neuron in the brain and inspecting the firing as the agent walks around. Instead of listening to neurons for action potentials, we listen to H units in the hidden layer for their average activation.

and we want to cover the entire grid of the arena by picking n large enough. We use the absolute value of each hidden unit as it provides a better visualization for the hidden layer activation.

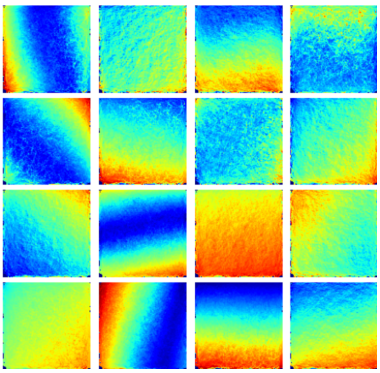


Figure 4.6: Visualization of 12 out of $H = 48$ hidden layer units. The grid has been discretized with a resolution of $m = 100$.

Visualization We observe that many units activate in gradients, a few activate only at borders, and many seem to contain random activation noise. No cells activate in triangular grid patterns or more structured representations. This is expected, both [10, 11] claim that regularization of the hidden layer is necessary for structured patterns to appear, [10] uses L2 regularization and noise in order for a grid pattern to appear, where [11] uses dropout [19] on the hidden layer, which acts as a form of noise.

We hypothesize that the network, as it is currently, could use precise numerical values to store the position inside the hidden state. To prevent it from using precise numerical representation to store the position, we will apply random noise to the hidden layer vector at each time-step during training[3] . This will force the model to discard precise numerical representation of the position, which would be disrupted by noise, in favor of more robust and structured representations.

3: Neurons are noisy by nature as there is a lot of interference due to the electrical activity of the brain, we will replicate this metabolic noise programmatically.

To better visualize what is the effect of noise, that we will apply to the next model, we first plot the value of the inner product between all pairs of activation matrices of the current model, then we will analyze the differences.

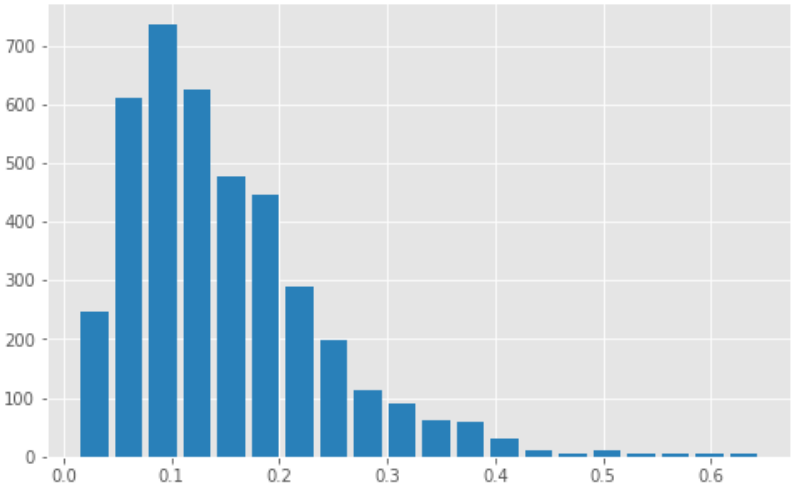


Figure 4.7: Inner product values on the x-axis and number of pairs in the y-axis.

The mean inner product between any two representations is 0.09, suggesting that most units have a low similarity between each other. The low similarity is good if each unit encodes a robust representation, however in this case the representations are not robust but very granulated and encode broad areas of locations.

4.5 GRU Noisy Model

This model will be analog to the previous one but will add Gaussian noise on the recurrent hidden layer: $\mathbf{h}(t) \leftarrow \mathbf{h}(t) + \mathbf{n}(t)$, where $n(t)_j \sim \mathcal{N}(0, \sigma^2)$ for $j \in \{1, \ldots, H\}$.

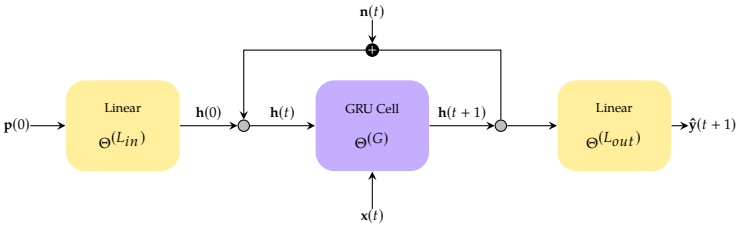
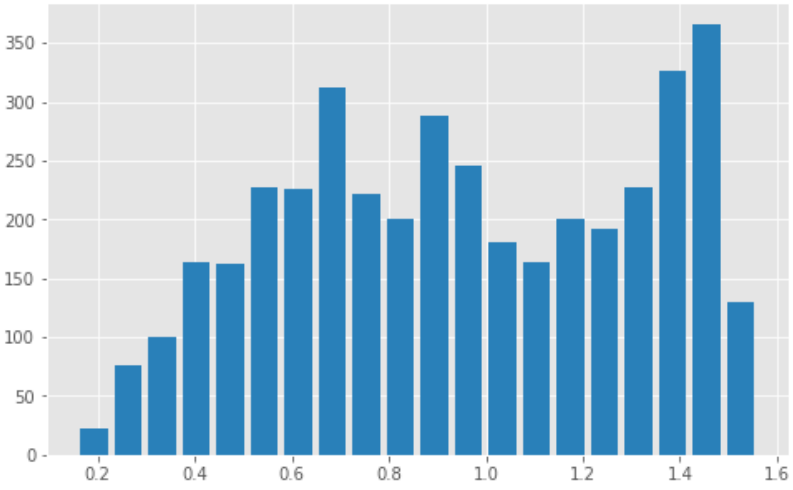


Figure 4.8: Model GRU with Gaussian noise applied on the recurrent connection at each timestep.

The agent is still able to path integrate in a very similar way to the noise-less model, with a mean squared error of 0.007 after 300 epoch if $\sigma = 0.1$ noise is applied. The model takes more to train as we increase the amount of noise. Overall the path is not as precise, but this is expected as we are injecting noise which forces coarser representations.

As expected, by perturbing the hidden layer at each time-step, we get a more robust representation formed of lines of activation which are more uniform (less noisy). It seems that the model uses the intersection of many lines of activation to encode a single position. If we test this model with higher values of noise path integration performance decreases, if we test this model with a lower level of noise we get activation regions between the previous and the current model.



The inner product between units suggests that this representation tend to be more alike (overlap), as the inner product pairs have a higher mean, and that each unit is firing more than before. We can see that noise does indeed provide a more robust representation, but also many units have a high inner product between each other and hence redundant, which is not optimal.

One thing that is not accurate with the anatomy of a real neuron is that many units are active on almost the entire arena, and using the negative regions of activation are able to encode the position. Metabolically activating a neuron in almost all positions is not realistic as it's a waste of energy. The next model will try to fix this problem by penalizing high activation rates, it's possible that a combination of noise and penalization for the activation of a unit may lead to the emergence of triangular grid patterns.

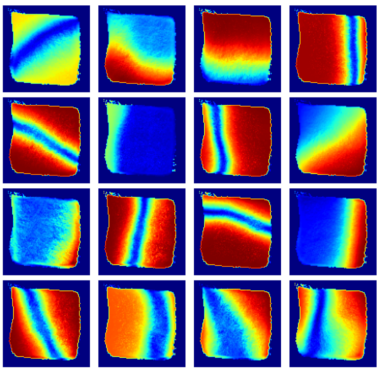


Figure 4.9: Visualization of 16 out of $H = 48$ hidden layer units. With noise $\sigma = 0.1$. More than this amount of noise makes the model unstable to train.

Figure 4.11: Inner product pairs of the noisy model with $\sigma = 0.1$.

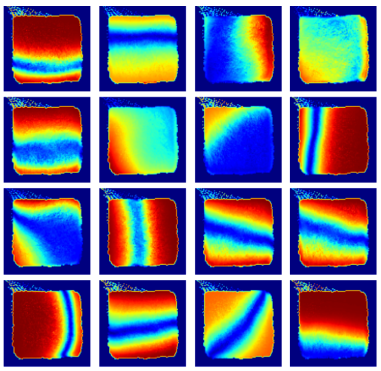


Figure 4.10: Visualization of 16 out of $H = 64$ hidden layer units. With noise $\sigma = 0.05$.

4.6 GRU Noisy with Regularization

To penalize the hidden layer for a high level of activation, we will apply L2 regularization on it. We will keep the same model as before, and change the optimization goal as follows:

$$\Theta^* := \arg\min_{\Theta} L(\Theta; \mathbf{x}, \mathbf{y}) + \frac{\lambda}{T \cdot B \cdot H} \sum_{t=1}^{T} \sum_{b=1}^{B} \|\mathbf{h}(t; b)\|_2^2 \tag{4.9}$$

for some regularization parameter λ. This will force the activation of the entire hidden layer to be as small as possible, thus activating one unit in the entire arena to encode the location will be penalized.

Results: again the agent is able to path integrate. Interestingly, by forcing the model to use smaller activations spots with L2 regularization on the hidden layer, we get spotted representations, some exhibiting structured triangular grid patterns. Some units still fire on straight lines, but none activate on almost the entire arena. It seems that the triangular grid patterns emerge as the intersection of many lines as suggested in [25] .

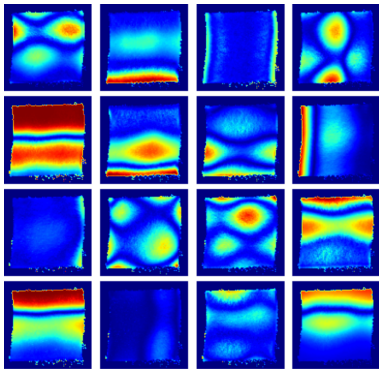


Figure 4.12: Visualization of 16 out of $H = 64$ hidden layer units. With noise $\sigma = 0.1$ and L2 penalization $\lambda = 0.1$.

[25]: Mathis et al. (2015), 'Probable nature of higher-dimensional symmetries underlying mammalian grid-cell activity patterns'

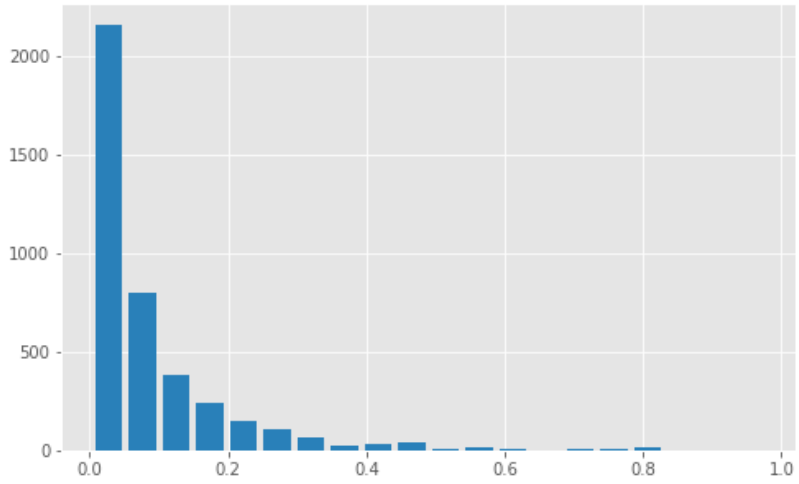


Figure 4.13: Inner product pairs of the noisy model with regularization.

The inner product histogram shows that most units have an inner product close to 0, suggesting that most units encode unique information as there is a low similarity between almost all pairs.

4.7 Place Cells Encoding

In the previous models, we have always provided the position as 2D Cartesian coordinates, however, as we have seen, the brain is hypothesized to use place cells to encode the location. It's thus a good idea to also provide the position to our model as if it is encoded by place cells as in [11]. It's possible that place cells may play an important part in the emergence of grid cells which is not there when using Cartesian coordinates.

Encoding: we will use a testing platform arena in the range $[0,1]^2$ and then plot the position in the arena (x, y) with respect to the firing frequency of the place cell, where red is high and blue is low. The region in which the cell fires the most (*i.e.* green to red) is called *place field*.

Each place cell can be characterized by the position in the arena *i.e.* the mean $\boldsymbol{\mu} \in [0,1]^2$ of the 2D normal distribution and it's standard deviation $\sigma \in \mathbb{R}$. More formally, $\forall \mathbf{x} \in [0,1]^2$:

$$f(\mathbf{x}; \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \frac{1}{\sqrt{(2\pi)^2 \det(\boldsymbol{\Sigma})}} \exp\left(-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu})\right) \tag{4.10}$$

where the covariance matrix is $\boldsymbol{\Sigma} \coloneqq \text{diag}(\sigma, \sigma)$.

By sampling the means uniformly at random we can generate as many place cells as we want. Let P be the number of generated place cells, then we can encode any 2D position $\mathbf{p} \in \mathbb{R}^2$ as a P-dimensional vector $\tilde{\mathbf{p}} \in \mathbb{R}^P$, with $\tilde{p}_k \coloneqq f(\mathbf{p}; \boldsymbol{\mu}_k, \boldsymbol{\Sigma})$ where $\boldsymbol{\mu}_k$ is the center of the k-th place cell for $k \in \{1, \dots, P\}$.

As a rule of thumb, the lower the number of place cells P, the higher we must pick σ in order to cover the entire arena, and vice-versa. This is because if we pick a small σ and a small P, then we might pick a position that is not close to any of the centers μ_k, and thus providing a bad encoding to the model.

A natural question to ask is, what standard deviation can best approximate place cells? And how is the position of the place cell selected by the brain? It turns out that the standard deviation, *i.e.* the place field size ranges from $350cm^2$ to about $1700cm^2$ [26] depending on the different hippocampal regions. The position, however, depends on other inputs and can also radically change in new environments in a process called remapping[27, 28] , which is still not completely understood.

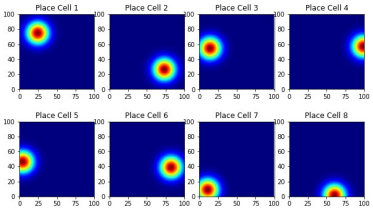


Figure 4.14: 8 place cells modeled with normal distributions parametrized by $\boldsymbol{\mu}$ uniform at random and $\sigma = 0.01$

[26]: Neher et al. (2017), 'From grid cells to place cells with realistic field sizes'

[27]: Fyhn et al. (2007), 'Hippocampal remapping and grid realignment in entorhinal cortex'
[28]: Colgin et al. (2008), 'Understanding memory through hippocampal remapping'

4.8 Decoder Place Cells Model

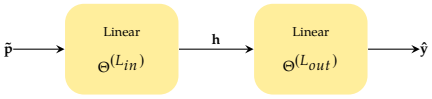


Figure 4.15: Model decoder from place cells position encoding to Cartesian position.

This model will take the position $\tilde{\mathbf{p}}$ encoded in using the previous place cell encoding, then project it using a linear layer to a H-dimensional representation $\mathbf{h}$, and finally transform it again to a 2-dimensional representation of the initial Cartesian coordinates. Again we will minimize the means squared error between $\hat{\mathbf{y}}$ and $\mathbf{y} \coloneqq \mathbf{p}$. As always we will inspect the hidden layer activation. Note that in this case, we will not use path integration, instead, we'll pick positions uniformly at random in the arena (in $[0,1]^2$) to train the model. The idea is to test whether using the place cell encoding as input might influence the appearance of grid patterns and if quasi-periodic patterns emerge without path integration. Furthermore, this model will be used to decode the place cell position

into Cartesian coordinates as it's not trivial to manually build a function to do so.

The model is able to reconstruct the position properly with a mean squared error of 1e-4, for example, the position $(0.8, 0.3)$ encoded with place cells and then decoded with the model is returned as $(0.8048, 0.3045)$. By inspecting the hidden layer we observe that encoding the position using the place cell representation produces a more spotted representation, but not exactly the triangular pattern we are looking for. It's interesting that in the previous model without noise and regularization spots of activation never appeared, but in this model, by just encoding the position using place cells we always get spots of activation.

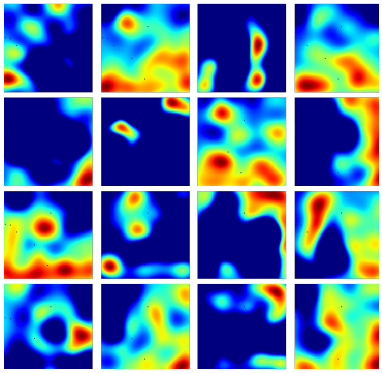


Figure 4.16: Visualization of 16 random out of 64 hidden layer units.

4.9 Auto-encoder Place Cells Model

This model will have the same structure as the previous decoder, however this time we will decode the position to the place cell representation, *i.e.* minimize the mean squared error between $\hat{\mathbf{y}}$ and $\mathbf{y} := \tilde{\mathbf{p}}$ instead of $\mathbf{y} := \mathbf{p}$. The idea is to see how auto-encoding the place cell position differs from decoding the position from place cells to the 2-dimensional Cartesian representation.

We observe that quasi-periodic patterns appear by auto-encoding the place cell position to a H-dimensional representation. Suggesting that completely encoding the position into a place cell representation has a big influence on the hidden layer.

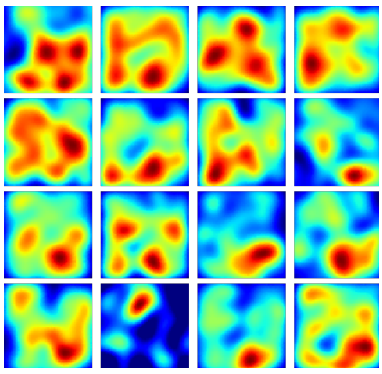


Figure 4.17: Visualization of 16 random out of 48 hidden layer units.

4.10 GRU Place Cells Dropout Model

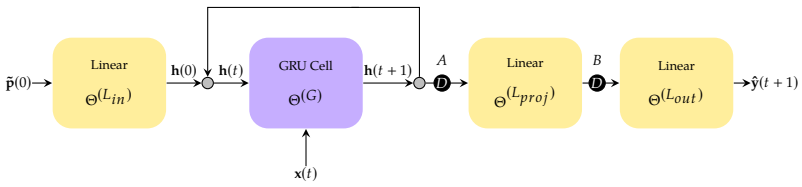


Figure 4.18: Model GRU with dropout.

In this model, we will use a very similar model as the previous path-integrator, but instead of decoding the position to Cartesian coordinates, we will train the model to encode the position to the place cell format, similarly to [11]. We will then use the model from Section 4.8 to decode the 2-dimensional position to visualize the integrated path.

In [11], dropout applied to A is necessary for grid pattern to appear, we will try with different configurations where dropout is not applied, applied on A, applied on B, or both. As always we will inspect the hidden layer $\mathbf{h}$ and in this case also the projection layer $\hat{\mathbf{h}}$ *i.e.* the hidden state between the projection linear layer and the output linear layer.

We observe that random gradients of activation appear when no dropout is applied in both hidden states $\mathbf{h}$ and $\hat{\mathbf{h}}$, if however we apply dropout in A and inspect the 128-dimensional hidden layer, blobs of activation appear. Those blobs are not very structured and mostly cover a coarse

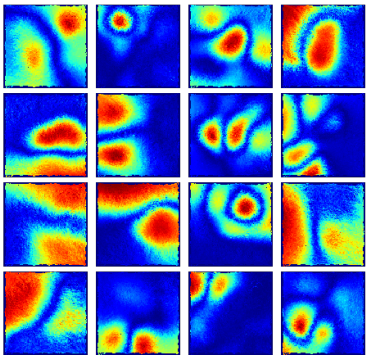


Figure 4.19: Visualization of 16 random out of 128 hidden layer units when dropout is applied on A.

region. Dropout is a form of regularisation, during training it removes data from the hidden layer randomly with probability 0.5, by doing so it forces the model to store more robust representation distributed between all units instead of having a few units containing all the information that would be otherwise sometimes removed. Furthermore, dropout is also a form of noise since it's randomly perturbing some units and changing the contained information. Applying dropout could be thus analogous to manually injecting noise and L1 regularization on the hidden layer as with the previous path-integration model.

If dropout is applied on A and B, then by inspecting the 512-dimensional projection layer many structured patterns appear, in the image random units are visualized but if the units are carefully selected some resemble grid patterns. We used 512 units to reproduce the work from [11], however it works similarly with 128 units.

Lastly, we observe that if dropout is applied only to B, then the visualization of B are analog to when dropout is applied on both A and B, suggesting that dropout is a key factor for the emergence of quasi-grid patterns on different layers. We observe that as a rule of thumb, to obtain spotted patterns on layer $\star$ we must apply dropout on $\star$, and that the closer is layer $\star$ to the final place cell output layer the more spotted representations we get.

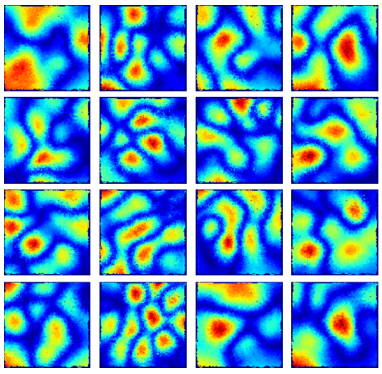


Figure 4.20: Visualization of 16 random out of 512 projection layer units when dropout is applied on both A and B.

4.11 Auto-encoder GRU Place Cells Model

In the previous model, we get similar results to [11], however from the test we have done on auto-encoding we observed that path integration seems to be not necessary to get quasi-periodic patterns. To test this hypothesis we will take the same model as Figure 4.18 and train it to encode random positions that do not follow each other, thus removing the path integration part. We will provide a place cell encoded initial random position $\tilde{\mathbf{p}}(0)$ and at each time-step provide $\mathbf{x}(t) = \mathbf{0}$ for all t. As before we apply dropout on A and B, and expect to get as output $\mathbf{y}(t) = \tilde{\mathbf{p}}(0)$ by training the network using the mean squared error loss between $\mathbf{y}(t)$ and $\hat{\mathbf{y}}(t)$. Note that we have visualized the last hidden layer but all hidden layers have a similar (almost equal) representation.

Interestingly, we get very similar representation with and without path-integration.

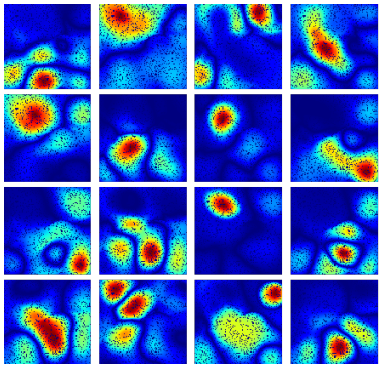


Figure 4.21: Visualization of 16 random out of 128 hidden layer units when dropout is applied on both A and B.

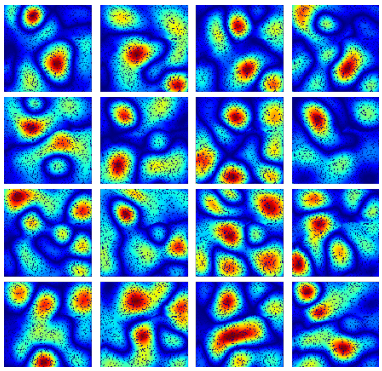


Figure 4.22: Visualization of 16 random out of 128 projection layer units when dropout is applied on both A and B.

5 Discussion

5.1 Noise and Regularization

We find, similarly to [10] that noise and regularization are both necessary for gird patterns to appear when using a Cartesian positional encoding. To investigate this more in-depth why this happens we apply different perturbations to the models. By inspecting the hidden layer we observe that applying noise forces a more robust representation on the hidden layer, however, this more robust representation uses broad areas of activation which is not realistic if compared to neurons in the brain. If neurons had to fire on half or more of the arena a lot of energy would be wasted, thus to enforce this constraint, regularization is needed. By forcing the hidden layer neurons to fire the least possible, without giving up the ability to encode the correct position, we observe that sparse blobs of activation appear. More precisely, those blobs of activation often get arranged into a grid pattern. This model must be carefully tuned for grid patterns to show up, we think that providing the position as Cartesian coordinates is not the favorable encoding of position for this robust and metabolically cheap representation to appear.

5.2 The Idea of Position

The most common way to encode the position is Cartesian coordinates, however different coordinate system such as polar, cylindrical, or homogeneous exist. Each system has its advantages and disadvantage depending on the type of use, even if at the end all represent the same thing. We hypothesize that using groups of place cells to encode the position might have different advantages, some of which might be noise resistance and sparsity, as perturbing the activation values of one place cell doesn't dramatically change the current position. Some disadvantages are that the precision is fixed by the activation and that more than 2 scalars are needed for good encoding. The idea of using place cell to store information doesn't necessarily have to be linked with location, it could be seen as the continuous analog of storing one-hot encoded data, which could be also generalized to more than 2 dimensional data.

5.3 Path Integration

In the introduction, we have described how grid cells might be some kind of map that the brain uses to update the current position in an allocentric manner. This is a common idea, but it's possible that grid cells are not a map, but only an intermediate to code convert the position from many place cells and different positions. In [21] grid cells are explained under the lens of pattern formation. The emergence of grid patterns from auto-encoding suggests that grid cells might not be the map, but a

[21]: Sorscher et al. (2019), 'A unified theory for the origin of grid cells through the lens of pattern formation'

compressed representation of place cells, raising the hypothesis that the brain might generate the encoding found in grid cells for other purposes that are different from the encoding of location. If this hypothesis is confirmed, then we might predict that non-position coding grid cells could be possibly found in the brain when an input similar to place cell encoding is provided.

5.4 Place Cells as General Encoding

Place cells could be used to encode any information that lives on a 2-dimensional plane with 2D Gaussians, this idea could be easily generalized to more dimensions by having for example a 3d space with 3d Gaussians. If other neurons use such representation is hard to know, but it's expected that if this type of encoding exists somewhere else, only information that has a one-to-one mapping with the world can be easily seen using electrodes, such as place cells, and hence we hypothesize that other types of similar encoding neurons would be harder to be found in the brain even if present. It would be interesting to investigate how encoding information similarly to place cells compares to other encoding methods, as it's similar to one-hot encoding but can also work with continuous and higher dimensional information.

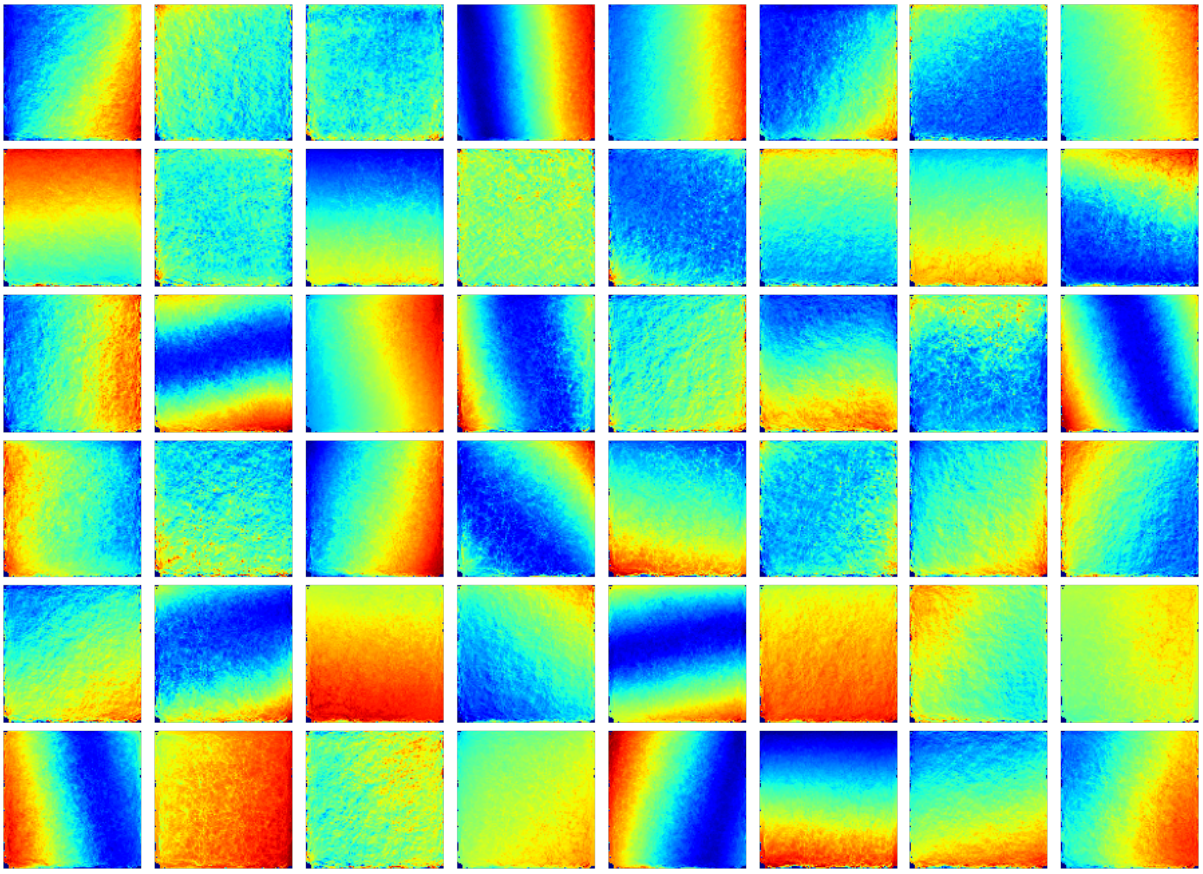
5.5 Future Work

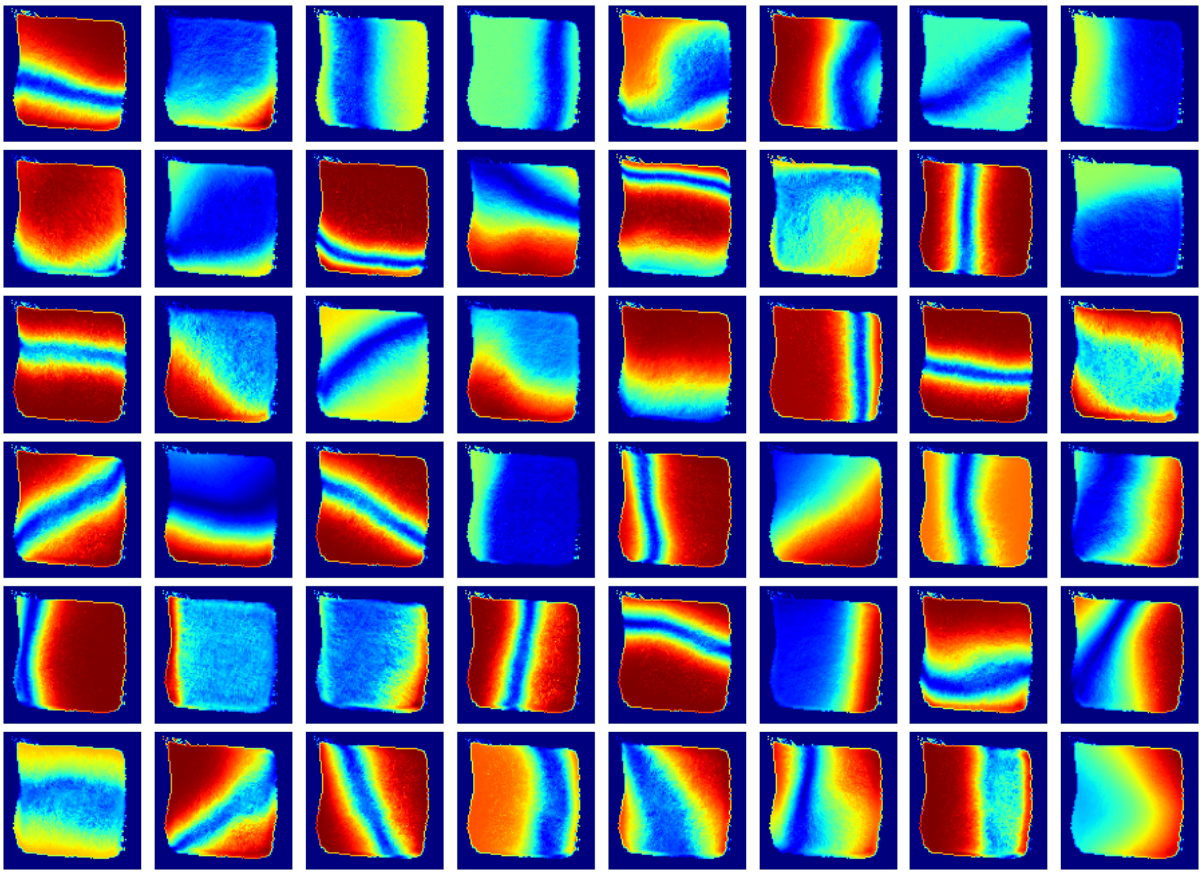
We see that grid cells emerge during path integration and auto-encoding of location, or more generally 2D vectors. It seems that path integration is trying to solve a broader problem that also contains auto-encoding as a subset producing similar representations in both cases. In other words, path integration is not necessary for the emergence of grid cells. It could be interesting to investigate what concrete advantage do grid cells provide when general information is encoded, for example, what happens if one of the many input place cells does not provide information (*i.e.* the place cell dies), how robust is the grid cell representation? Similarly, if one grid cell is removed how well are we able to get back the initial place cell information? Those are questions that can be answered by training machine learning models, on the other end on the neuroscience side, it would be interesting to investigate what other brain structures are reading information from grid cells, and similarly, how are place cells able to encode a single position from environmental cues in the first place?

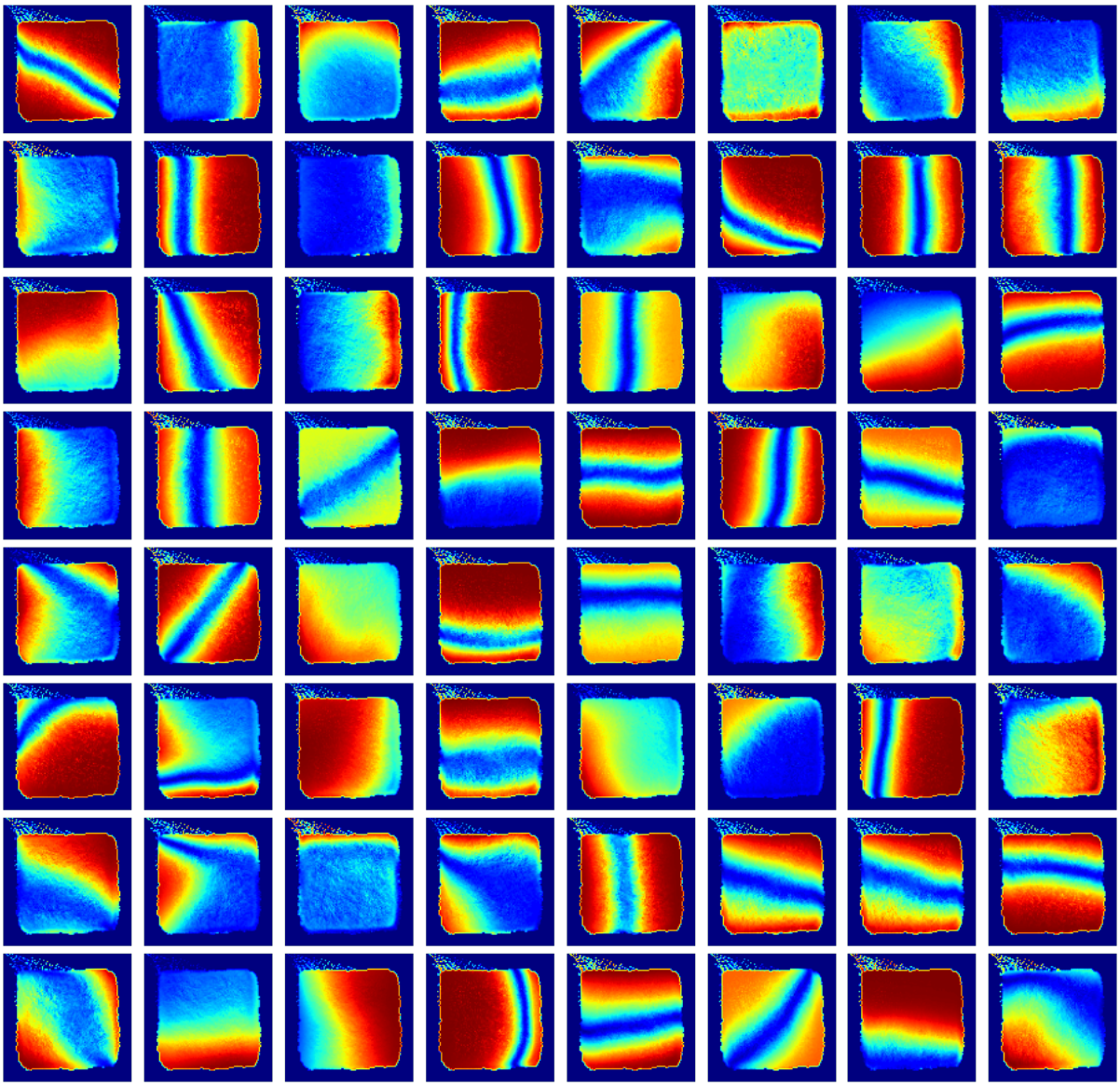
Appendix

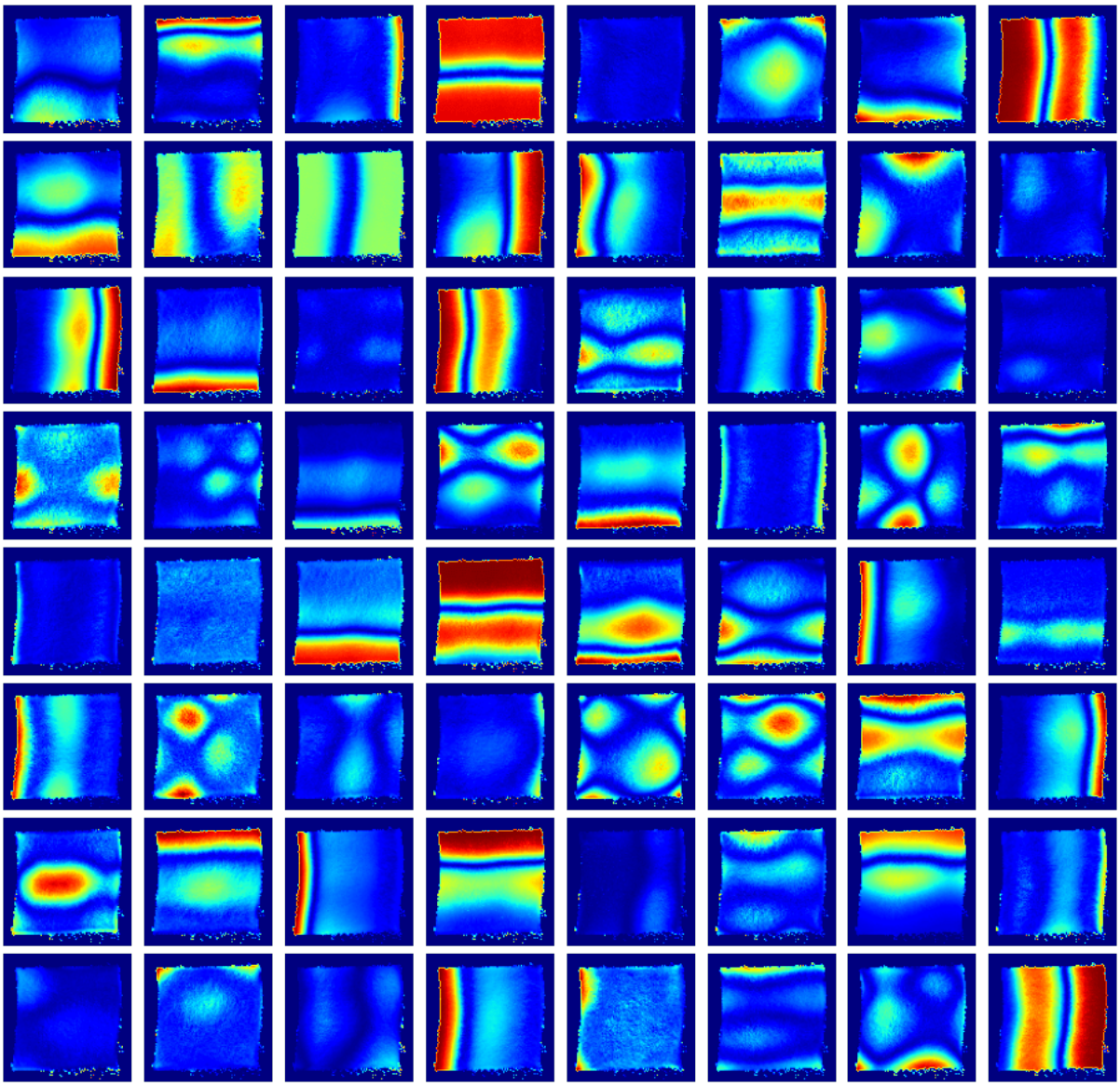
.1 Complete Activation Images

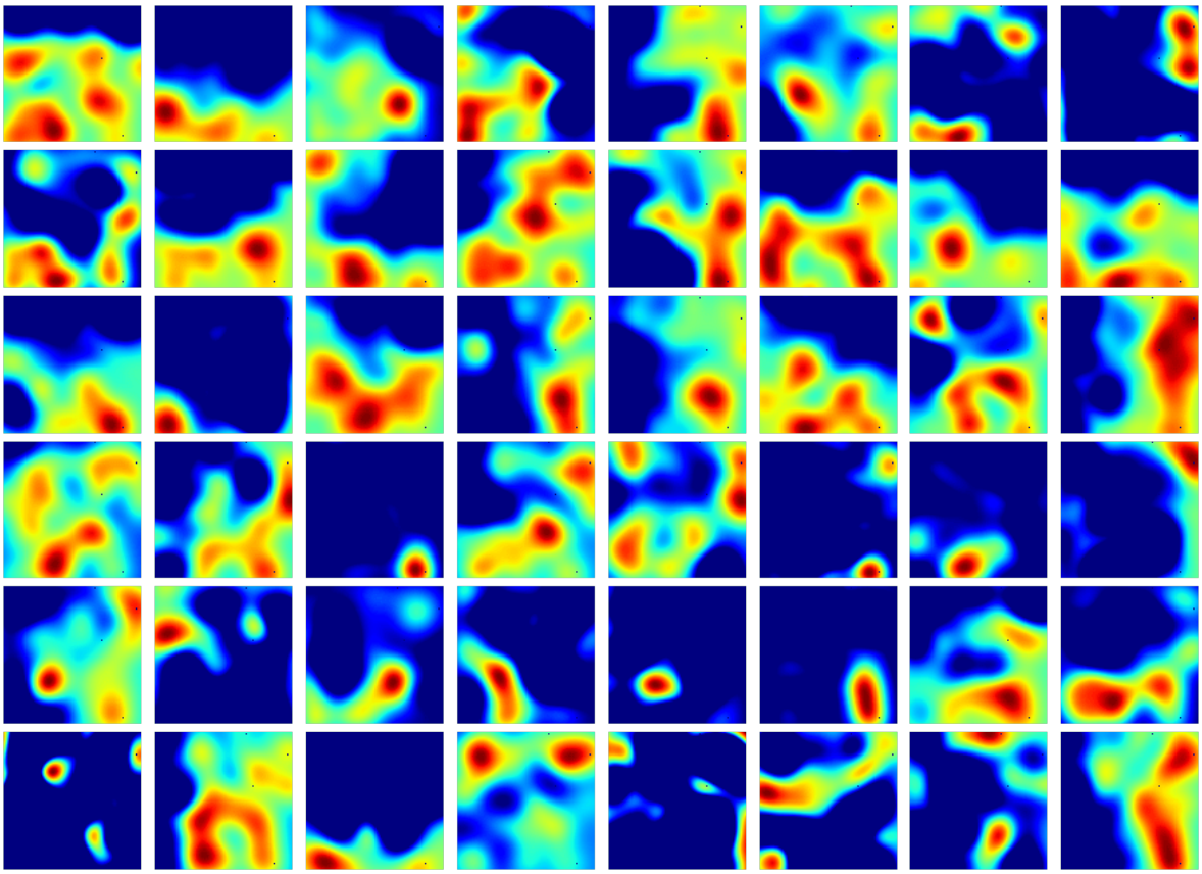
The following are the complete images of the layer activations in the same order they are presented in the thesis.

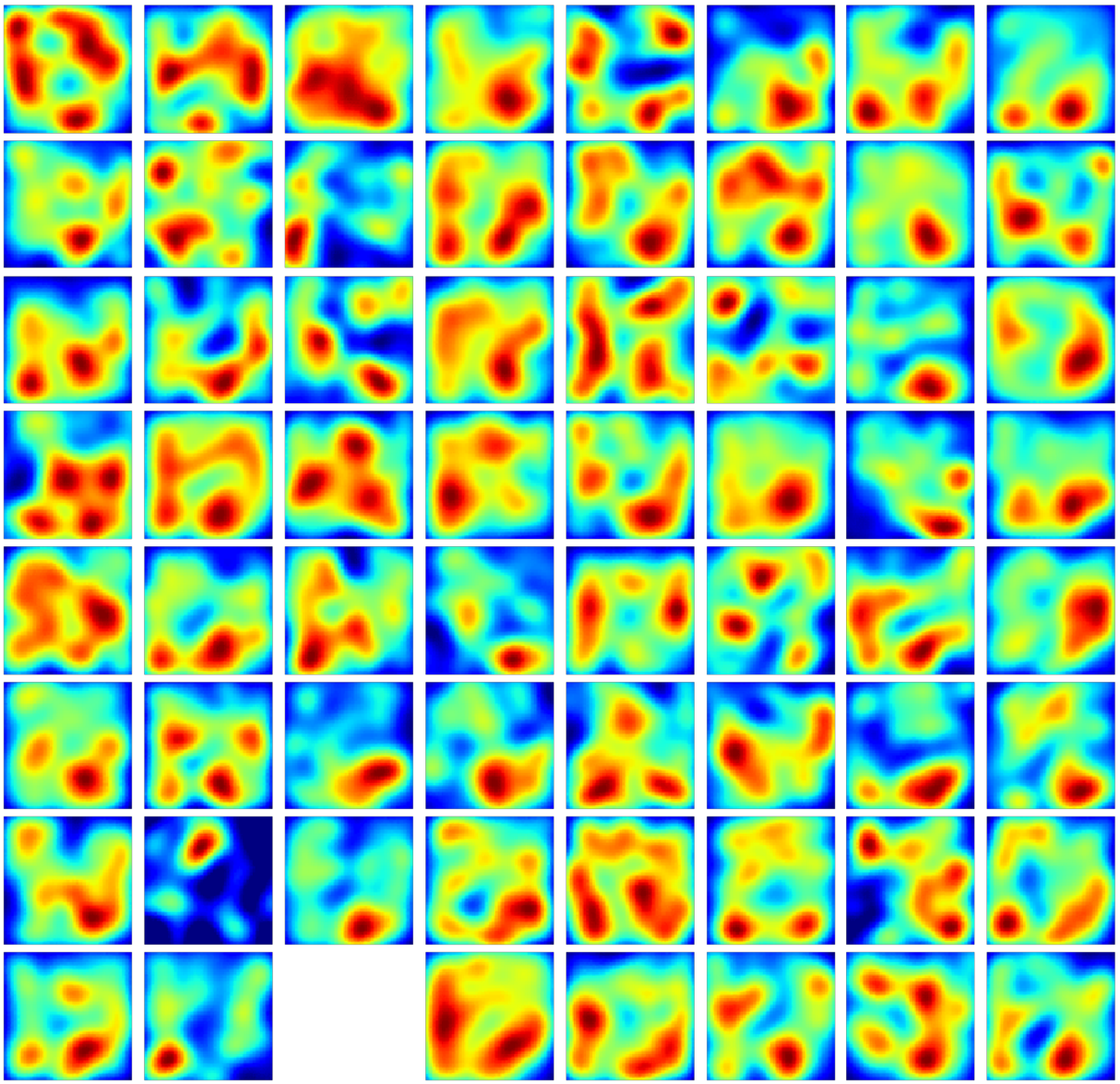


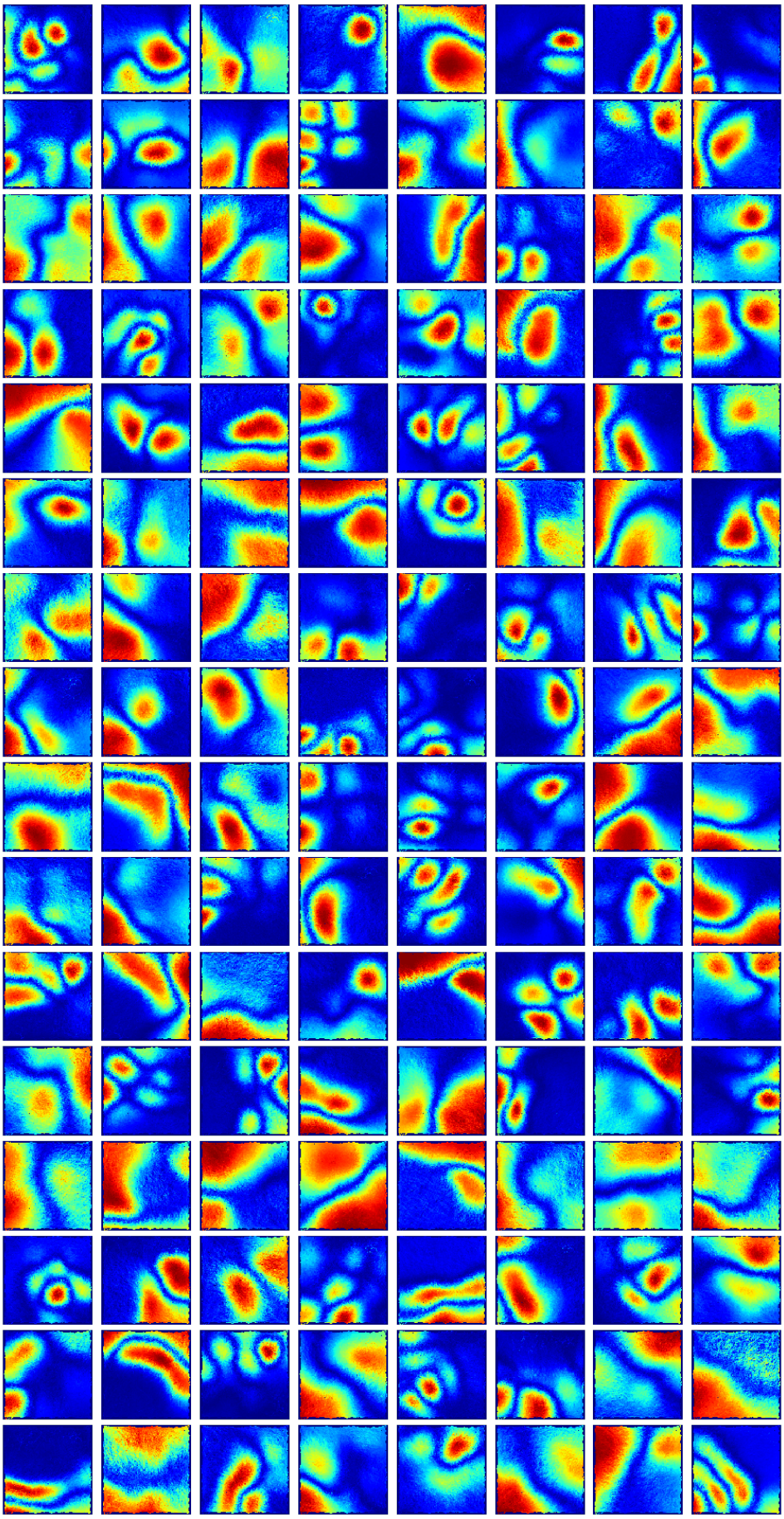


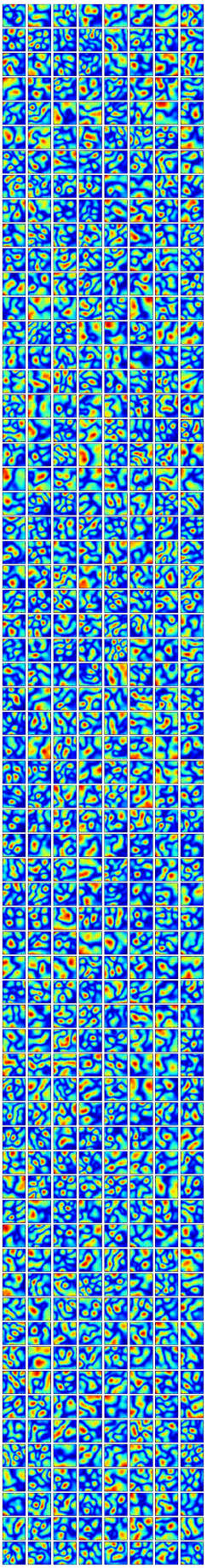


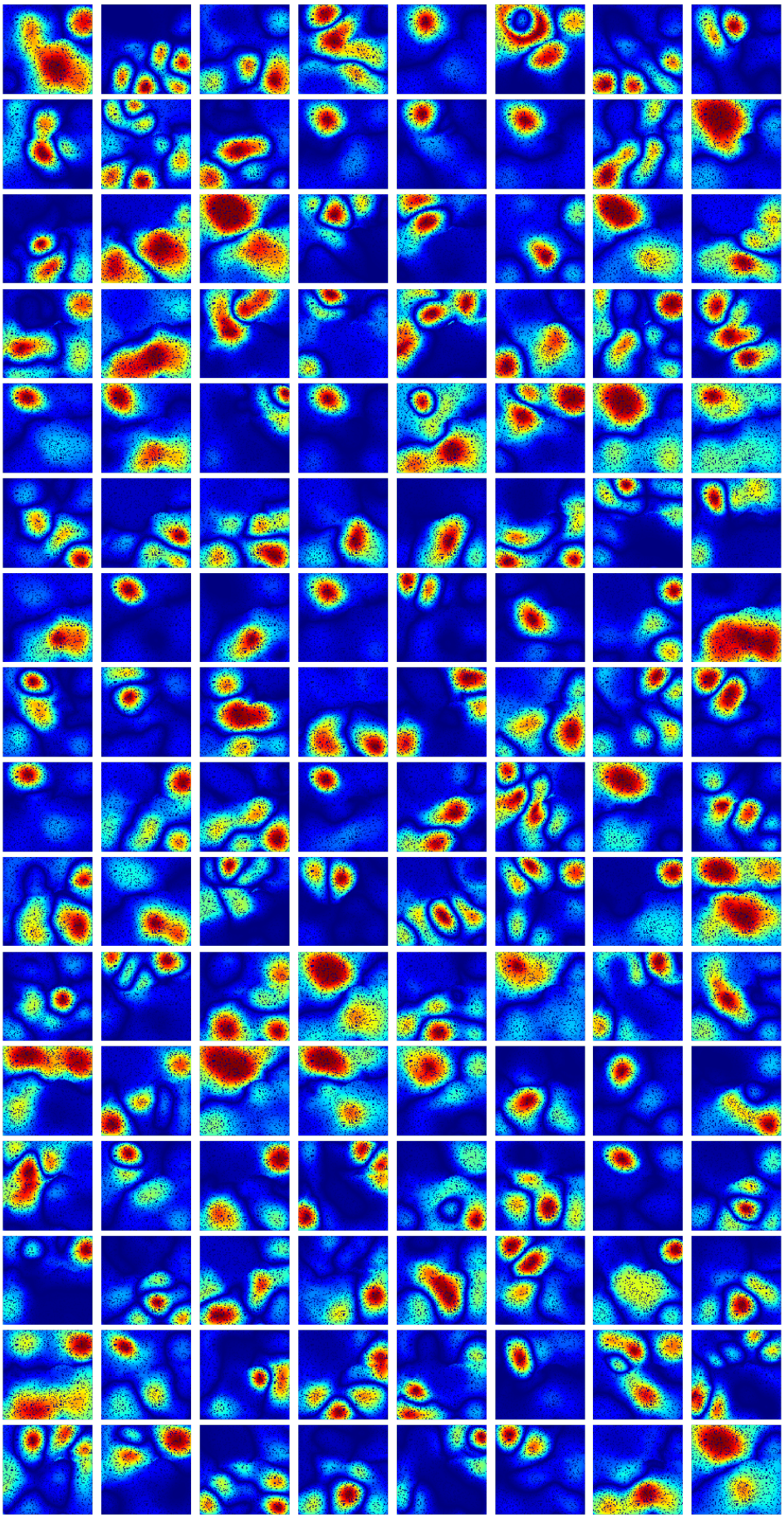


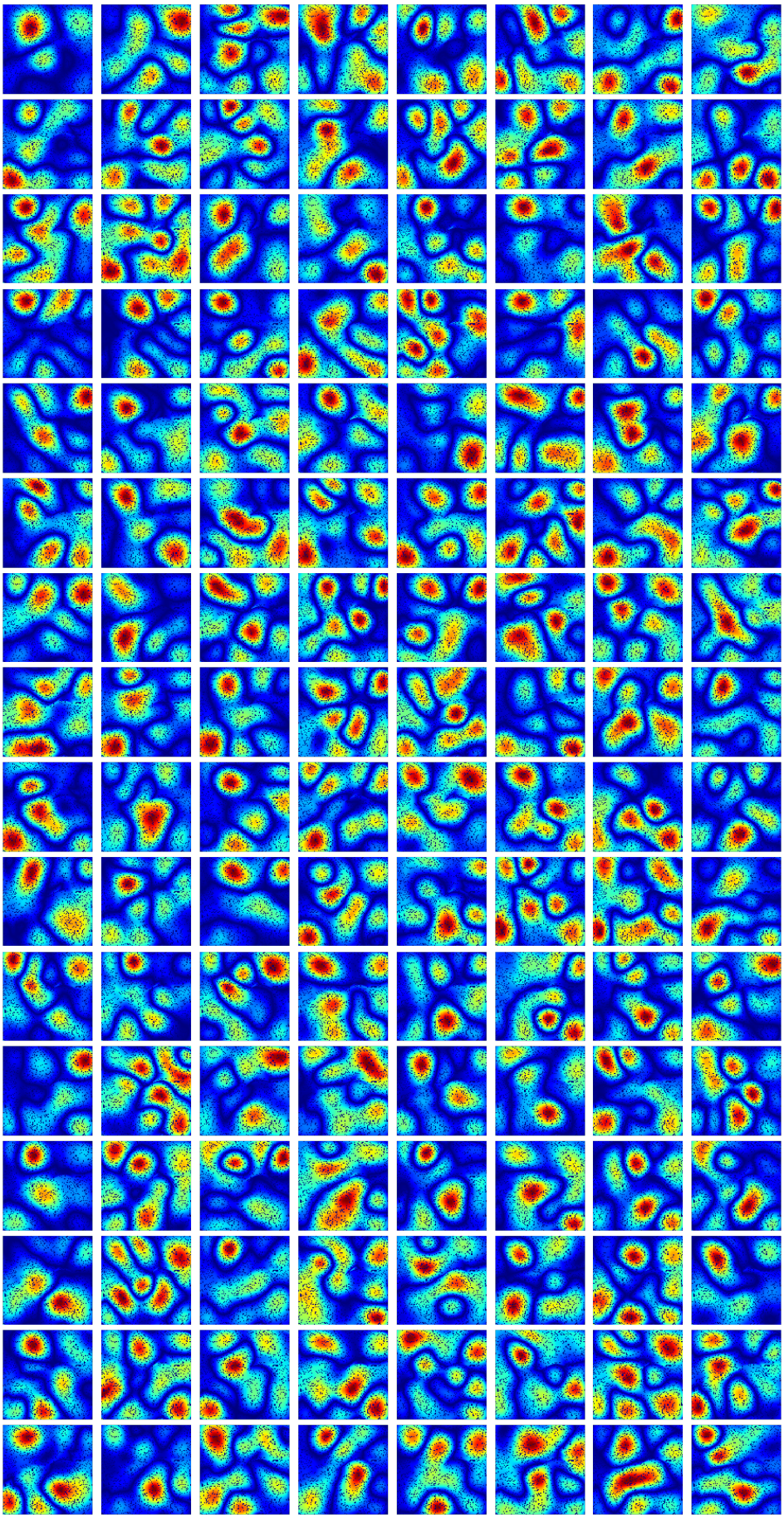












Bibliography

Here are the references in citation order.

[1] John O'Keefe and Jonathan Dostrovsky. 'The hippocampus as a spatial map: preliminary evidence from unit activity in the freely-moving rat.' In: *Brain research* (1971) (cited on page 1).

[2] John O'keefe and Lynn Nadel. *The hippocampus as a cognitive map*. Oxford: Clarendon Press, 1978 (cited on pages 2, 3).

[3] Jeffrey S Taube, Robert U Muller, and James B Ranck. 'Head-direction cells recorded from the postsubiculum in freely moving rats. I. Description and quantitative analysis'. In: *Journal of Neuroscience* 10.2 (1990), pp. 420–435 (cited on page 2).

[4] Florian Raudies et al. 'Head direction is coded more strongly than movement direction in a population of entorhinal neurons'. In: *Brain research* 1621 (2015), pp. 355–367 (cited on page 2).

[5] Hugh T Blair and Patricia E Sharp. 'Visual and vestibular influences on head-direction cells in the anterior thalamus of the rat.' In: *Behavioral neuroscience* 110.4 (1996), p. 643 (cited on page 2).

[6] Jeffrey S Taube. 'Head direction cells and the neurophysiological basis for a sense of direction'. In: *Progress in neurobiology* 55.3 (1998), pp. 225–256 (cited on page 2).

[7] Bruce L McNaughton et al. 'Deciphering the hippocampal polyglot: the hippocampus as a path integration system.' In: *Journal of Experimental Biology* 199.1 (1996), pp. 173–185 (cited on page 3).

[8] Torkel Hafting et al. 'Microstructure of a spatial map in the entorhinal cortex'. In: *Nature* 436.7052 (2005), pp. 801–806 (cited on page 3).

[9] Marcel Van Gerven and Sander Bohte. 'Artificial neural networks as models of neural information processing'. In: *Frontiers in Computational Neuroscience* 11 (2017), p. 114 (cited on page 4).

[10] Christopher J Cueva and Xue-Xin Wei. 'Emergence of grid-like representations by training recurrent neural networks to perform spatial localization'. In: *arXiv preprint arXiv:1803.07770* (2018) (cited on pages 4, 8, 9, 11, 12, 14, 20).

[11] Andrea Banino et al. 'Vector-based navigation using grid-like representations in artificial agents'. In: *Nature* 557.7705 (2018), pp. 429–433 (cited on pages 4, 8, 9, 12, 14, 16, 18, 19).

[12] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. 'Learning representations by back-propagating errors'. In: *nature* 323.6088 (1986), pp. 533–536 (cited on pages 4, 5).

[13] Sepp Hochreiter and Jürgen Schmidhuber. 'Long short-term memory'. In: *Neural computation* 9.8 (1997), pp. 1735–1780 (cited on pages 4, 6).

[14] Junyoung Chung et al. 'Empirical evaluation of gated recurrent neural networks on sequence modeling'. In: *arXiv preprint arXiv:1412.3555* (2014) (cited on pages 4, 12).

[15] Robert Hecht-Nielsen. 'Theory of the backpropagation neural network'. In: *Neural networks for perception*. Elsevier, 1992, pp. 65–93 (cited on page 5).

[16] Sepp Hochreiter. 'The vanishing gradient problem during learning recurrent neural nets and problem solutions'. In: *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems* 6.02 (1998), pp. 107–116 (cited on pages 5, 12).

[17] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. 'On the difficulty of training recurrent neural networks'. In: *International conference on machine learning*. 2013, pp. 1310–1318 (cited on pages 5, 6).

[18] Yedidyah Dordek et al. 'Extracting grid cell characteristics from place cell inputs using non-negative principal component analysis'. In: *Elife* 5 (2016), e10094 (cited on pages 8, 9).

[19] Nitish Srivastava et al. 'Dropout: a simple way to prevent neural networks from overfitting'. In: *The journal of machine learning research* 15.1 (2014), pp. 1929–1958 (cited on pages 9, 14).

[20] Svante Wold, Kim Esbensen, and Paul Geladi. 'Principal component analysis'. In: *Chemometrics and intelligent laboratory systems* 2.1-3 (1987), pp. 37–52 (cited on page 9).

[21] Ben Sorscher et al. 'A unified theory for the origin of grid cells through the lens of pattern formation'. In: *Advances in Neural Information Processing Systems*. 2019, pp. 10003–10013 (cited on pages 9, 20).

[22] Florian Raudies and Michael E Hasselmo. 'Modeling boundary vector cell firing given optic flow as a cue'. In: *PLoS computational biology* 8.6 (2012) (cited on page 10).

[23] James Martens. 'Deep learning via hessian-free optimization.' In: *ICML*. Vol. 27. 2010, pp. 735–742 (cited on page 12).

[24] Felix A Gers, Jürgen Schmidhuber, and Fred Cummins. 'Learning to forget: Continual prediction with LSTM'. In: (1999) (cited on page 12).

[25] Alexander Mathis, Martin B Stemmler, and Andreas VM Herz. 'Probable nature of higher-dimensional symmetries underlying mammalian grid-cell activity patterns'. In: *eLife* 4 (Apr. 2015) (cited on page 16).

[26] Torsten Neher, Amir Hossein Azizi, and Sen Cheng. 'From grid cells to place cells with realistic field sizes'. In: *PloS one* 12.7 (2017) (cited on page 17).

[27] Marianne Fyhn et al. 'Hippocampal remapping and grid realignment in entorhinal cortex'. In: *Nature* 446.7132 (2007), pp. 190–194 (cited on page 17).

[28] Laura Lee Colgin, Edvard I Moser, and May-Britt Moser. 'Understanding memory through hippocampal remapping'. In: *Trends in neurosciences* 31.9 (2008), pp. 469–477 (cited on page 17).