

---


# Analyzing Population Codes in Mice using Spiking Neural Networks

---

**Olav Førland**

Harvard University

olavfoerland@g.harvard.edu

 Analyzing Population Codes in Mice using Spiking Neural Networks

## Abstract

This study investigates how population codes in the mouse visual cortex (V1) encode visual stimuli and explores methods for disentangling these representations using Artificial Neural Networks (ANNs) and Spiking Neural Networks (SNNs). A single layer perceptron efficiently disentangles the representations, revealing task-relevant low-dimensional structures in the neural code. In contrast, an SNN trained with Spike-Timing-Dependent Plasticity (STDP) shows limited disentanglement, potentially reflecting the unsupervised nature of the learning rule or suboptimal hyperparameters.

## 1 Introduction

The visual cortex encodes information through the collective activity of millions of neurons, forming a population code. While this code has been the focus of long-standing theories, its structure remains elusive. At each side of the spectrum are the efficient coding hypothesis [1] [4] and robust coding hypothesis [15] [14]. The former posits that neural codes maximize information transmission by reducing input correlations, resulting in sparse and high-dimensional representations. The latter suggests that much of the neural encoding is redundant, leading the neural activity to rather reside in low-dimensional subspaces.

The authors of [21] suggest that the truth lies somewhere in the middle. They recorded the activity of approximately 10,000 neurons in the mouse primary visual cortex (V1) in response to thousands of natural images. Their findings indicate that population responses are governed by a trade-off between efficiency and smoothness. While the responses were high-dimensional, they exhibited structured correlations, following a power-law scaling that balanced the need to maximize information transmission with the requirement to maintain smooth, differentiable mappings. This enables robust and generalizable neural representations.

The authors of [2] extend this by examining how population codes in the brain support learning from few examples. They used a biologically plausible delta rule to investigate how population codes shape inductive biases and enable sample-efficient learning.

However, on discriminating mice vs birds, they found that the population code had an entangled representation, complicating the ability of simple readout mechanisms to generalize. Continuing down this road, I aim to analyze how different learning rules, such as those implemented by an Artificial Neural Network (ANN) and a Spiking Neural Network (SNN), can disentangle these representations and enable efficient learning. To achieve this, I first reproduce key results from [2], focusing on the population codes in mice when presented with images of birds and mice. I then simulate downstream task neurons using ANNs and SNNs, record their responses to each population code in the training set, and analyze the resulting kernels.

## 2 Background

This section presents the necessary background for understanding the models I have used. 2.1 briefly present the single-layer perceptron, and 2.2 gives a thorough review of spiking neural networks.

### 2.1 Artificial Neural Networks (ANNs)

As ANNs have been covered extensively in class, I will only for completeness give a high-level overview of the single-layer perceptron relevant for this project. The single-layer perceptron is a simple computational model where input data is mapped directly to an output through a weighted linear transformation followed by a nonlinear activation function. Mathematically, the output  $\mathbf{y}$  can be expressed as:

$$\mathbf{y} = \phi(\mathbf{W}\mathbf{x} + \mathbf{b}),$$

where  $\mathbf{x}$  represents the input vector,  $\mathbf{W}$  is the weight matrix,  $\mathbf{b}$  is the bias vector, and  $\phi$  is the activation function (e.g., ReLU or sigmoid). A single-layer perceptron can learn to classify linearly separable data and serves as a useful model for studying basic neural representations.

### 2.2 Spiking Neural Networks (SNNs)

Spiking Neural Networks are often referred to as the third generation of neural networks [11], distinguished by their biologically inspired computation and energy efficiency [22]. Unlike traditional ANNs, where information is represented as continuous scalar values and processed in discrete steps, SNNs emulate the behavior of biological neurons by communicating through discrete spikes, or action potentials, transmitted in continuous time.

This subsection presents the theory behind SNNs. 2.2.1 overviews how neurons are structured in these networks, relating it to their biological inspiration. 2.2.2 overviews the most popular models of how the neurons learn. 2.2.3 reviews two different data encoding schemes, namely rate encoding and temporal encoding, and 2.2.4 describes two different learning rules, spike-based backpropagation and spike timing-dependent plasticity.

#### 2.2.1 Biological Inspiration

The basic structure of SNNs is directly inspired by biological neurons, which consist of dendrites, a soma, an axon, and synapses [22]. In SNNs, neurons receive input through synapses connected to their dendrites. These inputs, represented as spikes, are aggregated in the soma, where a membrane potential is computed. If the membrane potential surpasses a certain threshold, the neuron fires a spike, which is transmitted to other neurons through its axon. The potential then Figure 1 compares a biological neuron with a neuron in an SNN.

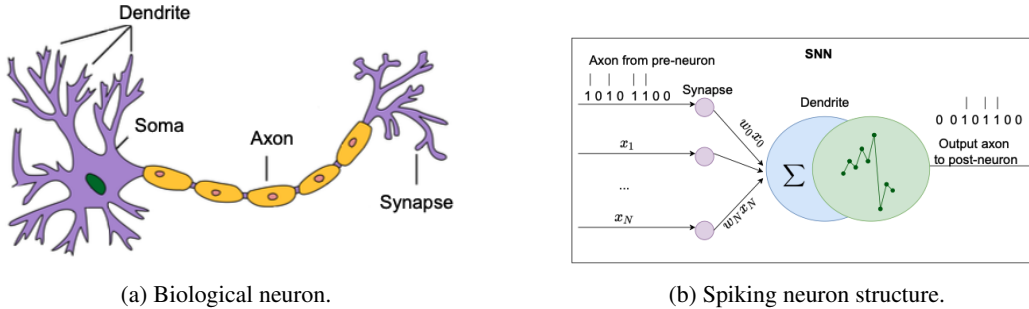


Figure 1: Comparison of a biological neuron and spiking neuron (adopted from [22]).

The membrane potential is the electric potential difference between the inside and the outside of a neuron. It arises due to the uneven distribution of ions across the cell membrane, where ions are constantly moving in and out of the cell through the membrane. The current flow caused by this

is mainly ascribed to the  $\text{Na}^+$ ,  $\text{K}^+$ , and  $\text{Cl}^-$  ions [10]. The resulting membrane potential can be mathematically modeled using the Goldman-Hodgkin-Katz equation:

$$v_m = \frac{RT}{F} \ln \left( \frac{P_K[\text{K}^+]_{\text{out}} + P_{\text{Na}}[\text{Na}^+]_{\text{out}} + P_{\text{Cl}}[\text{Cl}^-]_{\text{in}}}{P_K[\text{K}^+]_{\text{in}} + P_{\text{Na}}[\text{Na}^+]_{\text{in}} + P_{\text{Cl}}[\text{Cl}^-]_{\text{out}}} \right),$$

where  $P_K, P_{\text{Na}}, P_{\text{Cl}}$  are the membrane permeabilities of potassium, sodium, and chloride ions, respectively.  $R$  is the universal gas constant,  $T$  is the absolute temperature, and  $F$  is the Faraday constant. Also,  $[A]_{\text{out}}$  and  $[A]_{\text{in}}$  are the concentrations of ion  $A$  outside and inside the cell, respectively [18]. For a typical neuron at rest,  $P_K = 1, P_{\text{Na}} = 0.04, P_{\text{Cl}} = 0.45$ , while at the peak of neuronal action potential,  $P_K = 1, P_{\text{Na}} = 12, P_{\text{Cl}} = 0.45$ . This leads to resting membrane potential  $E_m = -70.15[\text{mV}]$  and action potential  $v_{\text{peak}} = 38.43[\text{mV}]$  [22].

## 2.2.2 Spiking Neuron Models

There are several ways of modeling the interactions of the neurons in SNNs. The different approaches aim to model the change in potential using differential equations. [7] compares several models. I recite the three most important for this study, namely the Hodgkin-Huxley (HH), Leaky Integrate-and-Fire (LIF), and Izhikevich models.

**Hodgkin-Huxley (HH) Model** is one of the first spiking models, and captures the generation of action potentials by modeling ion channel dynamics through differential equations [6]:

$$C_m \frac{dv_m}{dt} = I_{\text{ion}} + I_{\text{syn}}(t)$$

$$I_{\text{ion}}(t) = G_K n^4 (v_m - E_K) + G_{\text{Na}} m^3 h (v_m - E_{\text{Na}}) + G_L (v_m - E_L)$$

where  $C_m$  is the membrane capacitance,  $v_m$  is the membrane potential,  $I_{\text{ion}}$  represents ionic currents, and  $I_{\text{syn}}(t)$  denotes synaptic input currents. Furthermore,  $G_K$  and  $G_{\text{Na}}$  represent the conductance of the K and Na ion,  $E_K$  and  $E_{\text{Na}}$  the reversal potential of the K and Na ion, while  $G_L$  and  $E_L$  represent the leak conductance and reversal potential.  $n, m$ , and  $h$  are constants that are further described by differential equations found in [6] and [22]. The model is biologically plausible, but demands large computational resources.

**Leaky Integrate-and-Fire (LIF) Model** is a simpler alternative, representing membrane potential dynamics with:

$$C_m \frac{dv_m}{dt} = -G_L (v_m - E_L) + I_{\text{syn}}(t)$$

$$v_m \geq v_\theta \implies v_m \leftarrow v_{\text{peak}} \text{ then } v_m \leftarrow v_{\text{reset}}$$

where  $v_\theta$  is the threshold voltage,  $v_{\text{peak}}$  is the action potential, and  $v_{\text{reset}}$  is the resetting membrane potential. When the voltage  $v_m$  crosses the threshold  $v_\theta$ , the neuron emits a spike and the voltage is reset to zero for a refractory period  $\tau_{\text{ref}}$ . This model is computationally efficient and widely used in large-scale simulations [22].

**Izhikevich Model** combines the biological realism of HH with the efficiency of LIF neurons [9]. It uses:

$$C_m \frac{dv_m}{dt} = k(v_m - E_L)(v_m - v_t) - u + I_{\text{syn}}(t),$$

$$\frac{du(t)}{dt} = a(b(v_m - E_m) - u)$$

with the spike reset

$$v_m \geq v_\theta \implies \begin{cases} v_m \leftarrow c \\ u \leftarrow u + d \end{cases}$$

where  $u$  is a membrane recovery variable, accounting for the activation of  $\text{K}^+$  ionic currents and inactivation of  $\text{Na}^+$  ionic currents, and  $v_t$  is the instantaneous threshold potential.  $a, b, k, V_r, V_t$  are tunable parameters [22]. The Izhikevich model is capable of reproducing diverse spiking patterns with minimal computational cost using parameters in [8].

### 2.2.3 Spike Encoding

Information is represented as spike trains in SNNs. There are two principal encoding schemes, namely rate encoding and temporal encoding. In rate encoding scheme, the number of number of spikes within a time-interval encodes information, while in temporal encoding, their timing additionally conveys information.

**Rate Encoding** represents information by the average spike rate over a time window. For a spike train with  $n$  spikes in time  $T$ , the firing rate  $r$  is given by:

$$r = \frac{n}{T}.$$

This method is robust to noise and simplifies the analysis but sacrifices temporal precision.

**Temporal Encoding** encodes information in the exact timing of spikes. For instance, in latency coding, the delay of the first spike relative to a reference carries the input value. Temporal encoding is sparser and can represent higher-dimensional information, but it is more sensitive to temporal noise.

### 2.2.4 Learning Mechanisms

Due to the non-differentiable nature of the spike trains, SNNs cannot be trained directly by gradient descent as with ANNs. Instead, learning algorithms need to be adopted to the event-driven nature of the information. Two approaches are spike-based backpropagation and spike-timing-dependent plasticity (STDP).

**Spike-Based Backpropagation** adapts traditional backpropagation to SNNs by approximating gradients of the non-differentiable spike function using surrogate gradients. For example, the loss function  $L$  for a set of target spike trains  $s$  and predicted spike trains  $\hat{s}$  can be defined as:

$$L = \frac{1}{2} \int_0^T (\alpha \times (s(t) - \hat{s}(t)))^2 dt.$$

where  $\alpha$  is a normalized smooth temporal convolution kernel, and the spike trains are represented as  $s(t) = \sum_{t_k < t} \delta(t - t_k)$  [23]. The gradient of the spike train can then be approxiated with a function of the membrane potential, e.g.

$$\frac{\partial s}{\partial w} \approx \frac{\partial \sigma(v_m)}{\partial w} = \sigma'(v_m) \frac{\partial v_m}{\partial w} \approx \sigma'(v_m)(\epsilon \times s)$$

where  $\sigma(x) = \frac{x}{1+|x|}$  is a fast sigmoid.

**Spike-Timing-Dependent Plasticity (STDP)** STDP is a biologically inspired, unsupervised learning rule that adjusts synaptic weights based on the relative timing of pre- and post-synaptic spikes [16]. If a pre-synaptic spike arrives before a post-synaptic spike, their connection is strengthened. Otherwise, it is weakened. The update rule is:

$$\Delta w = \begin{cases} A_+ \exp(\frac{\Delta t}{\tau_+}) & \text{if } \Delta t < 0, \\ -A_- \exp(-\frac{\Delta t}{\tau_-}) & \text{if } \Delta t \geq 0, \end{cases}$$

where  $\Delta t$  is the time of the presynaptic spike minus the time of the postsynaptic spike, and  $A_+$ ,  $A_-$  and  $\tau_+$ ,  $\tau_-$  are constants.

## 3 Methodology

This section presents my approach. 3.1 formally presents the problem, while 3.2 presents the data and code repositories I have used. 3.3 and 3.3.1 detail the ANN and SNN implementations, respectively. Finally, 3.4 outlines my validation strategies. 3.2

### 3.1 Problem Setup

I continue on the work in [2] using the recorded responses of 10,000 neurons in a mouse visual cortex (V1) in response to 2800 images [20]. Formally, I consider a population of  $N$  neurons and  $P$  patterns. The responses of the neuron population,  $\mathbf{r}^\mu = [r_1^\mu, r_2^\mu, \dots, r_N^\mu]$ , vary with the input stimuli. These responses define the population code.

In [2], a readout neuron learns from  $P$  stimulus-response examples, which are given by  $\mathcal{D} = \{\theta^\mu, y(\theta^\mu)\}_{\mu=1}^P$ . The authors learn the readout weights with the biological plausible delta-rule,

$$\Delta w_j = \eta \sum_{\mu} r_j(\theta^\mu)(y(\theta^\mu) - \mathbf{r}(\theta^\mu) \cdot \mathbf{w})$$

where  $\eta$  is the learning rate and  $\mathbf{w}$  are the learned weights. I will restrict the analysis to the responses when presented with mice or birds.

To start, I will reproduce the results from [2], which examine population-level dynamics in the mouse brain in response to images of birds and mice. Specifically, I will replicate the disentangled representation of V1 shown in Figure 3D of [2], which demonstrates how these representations hinder the performance of linear readouts. Following this, I will investigate two learning frameworks: (i) how/whether a one-layer Artificial Neural Network (ANN) disentangles representations, and (ii) how/whether a biologically plausible Spiking Neural Network (SNN) achieves the same. The aim is to simulate and analyze how downstream neurons process and disentangle the population code from response neurons.

### 3.2 Data and Code

This study utilizes the publicly available dataset described in [20] to reproduce the population-level dynamics in mouse V1 neurons. Specifically, the analysis focuses on neural responses to images of birds and mice, as explored in Figure 3D of [2]. The dataset consists of spike counts derived from deconvolved and z-scored calcium fluorescence traces, recorded from approximately 10,000 neurons in the visual cortex. Stimuli were drawn from 15 ethologically relevant ImageNet categories, with preprocessing and category information obtained from the repository at [19].

For this project, only the bird and mouse image categories were analyzed, consistent with the preprocessing pipeline outlined in [19]. Additionally, the open-source repository [13] provided crucial guidance for reproducing key representations and analyses from [2]. These resources ensured reproducibility and accuracy throughout the study.

All code developed for this project is available at this Github Repository.

### 3.3 Implementation of Artificial Neural Network

To explore how a one-layer Artificial Neural Network (ANN) disentangles neural representations of visual stimuli, I employ a simple feedforward architecture. This network comprises a single hidden layer with ReLU activation functions and a softmax output layer. The Adam optimizer is used to train the network on the population codes from [20], focusing on classifying birds and mice.

The kernel is analyzed by recording activations from the hidden layer as each population code is passed through the network. This is repeated for varying numbers of hidden neurons. I estimate the benefit of additional neurons computing the margin on the top two principal components using Support Vector Machines (SVM). I implement both ANN and SVM using Scikit Learn's *MLPClassifier* and *SVC*, respectively [12]. The aim is to investigate how the kernel changes under a different learning rule compared to [2].

#### 3.3.1 Implementation of Spiking Neural Network

The Spiking Neural Network (SNN) implementation also utilizes a single hidden layer, fixed at 128 neurons, to examine the effect of varying the number of training examples. Population codes from [musegutta] are rate-encoded, and the network architecture is adapted from [5], employing LIF neurons and Spike-Timing-Dependent Plasticity (STDP) for training. Simulations and training are conducted using the Brian2 simulator [17]. The model used is based on an implementation of [5], the

code of which can be found in this Google Colab. The aim is to provide insight into how biologically plausible learning rules disentangle neural representations under varying conditions, complementing the investigations performed with ANNs.

### 3.4 Validation

Similar to [2], I use the cumulative power distribution  $C(k)$  as a metric to quantify the alignment between the population code's kernel and the learning task. This metric, originally introduced in [3], measures the proportion of a target function's power contained in the top  $k$  eigenmodes of the kernel. Mathematically, it is defined as:

$$C(k) = \frac{\sum_{l=1}^k v_l^2}{\sum_{l=1}^{\infty} v_l^2},$$

where  $v_l^2$  represents the eigenvalue power of the kernel's  $l$ -th mode. A rapidly rising  $C(k)$  indicates strong alignment between the kernel and the task, as more power is concentrated in the top modes. Tasks with high  $C(k)$  values can be learned more efficiently, requiring fewer training samples compared to tasks with lower  $C(k)$  [2].

To compute the kernel, I follow the methods discussed in class, using the recorded population codes as input. Spectral decomposition is applied to extract the eigenvectors and eigenvalues of the kernel, enabling the computation of  $C(k)$ . Additionally, I randomly project the high-dimensional population codes into 3D for visualization, as explored through the problem sets in class, to visualize the structure of the neural representations.

## 4 Results

This section presents the results. 4.1 reproduces the results of [2], while 4.2 and ?? present the results of the ANN kernel and SNN kernel, respectively.

### 4.1 Entangled Representation in Neural Response

Figure 2 show the reproduced results from Figure 3D in [13]. As can be seen from both the two principal eigenvectors (2a) and the power of the kernel (2b), V1 has an entangled representation. Therefore, linear readouts do not work well on discrimination tasks.

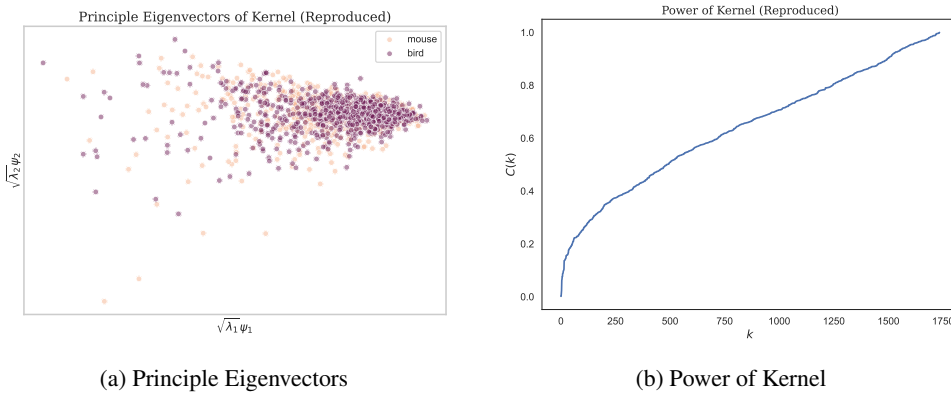


Figure 2: Comparison of the two principle eigenvectors and power of kernel, reproduced from [2].

### 4.2 ANN Kernel

Despite the entangled representation, the population code in V1 encodes the information of the image displayed very well. Figure 3 shows the same plots for the kernel of a one-layer ANN. The

intermediate neurons separates the two classes perfectly regardless of the number of neurons in the hidden layer (3a, 3b, 3c, 3d). This is explained by the power of the kernel, as the ANN manages to encode the most important information in the principle eigenvectors (3e, 3f, 3g, 3h). Furthermore, Table 1 shows the SVM margin for each of the cases when fitting it to the two principle eigenvectors. This is also unaffected by the number of neurons, reinforcing the results of Figure 3.

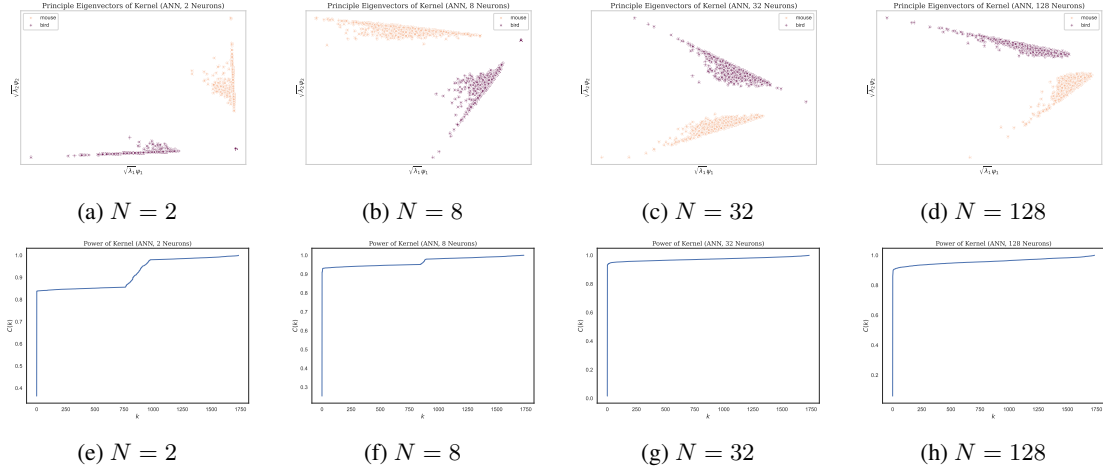


Figure 3: Comparison of the top two eigenvectors and kernel power for different numbers of hidden neurons  $N$ . The top row shows the principal eigenvectors, while the bottom row shows the power of the kernel.

Each row visualizes the kernel resulting from training a SNN with STDP on the population codes from V1 of mice for different number of training examples  $P$ . From the left: Kernel of spike activations; Random 3D Projection of spike activations; Power of Kernel; The top two eigenvectors of the Kernel.

Number of Neurons	Margin
2	0.032
8	0.031
32	0.031
128	0.033

Table 1: Margins for different numbers of hidden neurons.

### 4.3 SNN Kernel

Figure 4 shows the resulting kernel after training SSN with STDP using 10 (4a), 25 (4b), and 50 (4c) training examples. The figure shows a heatmap of the kernel, a random 3D projection of the population code, the cumulative power of the kernel, and the top two eigenvectors. In contrast to the ANN kernel, the SNN kernel does not seem to efficiently separate the two classes. The structure of the kernels seem to change little with increasing number of training examples. The cumulative power is approximately linear, and the points are still entangled.

## 5 Discussion & Conclusion

The results demonstrate that the one-layer Artificial Neural Network (ANN) effectively disentangles the representations in the population code of V1 with a minimal number of hidden neurons. This suggests that the neural responses encode the relevant information about the images displayed to the mice in a way that is linearly separable after minimal processing. Furthermore, the success of the ANN with few hidden neurons implies that the information resides on a low-dimensional manifold,

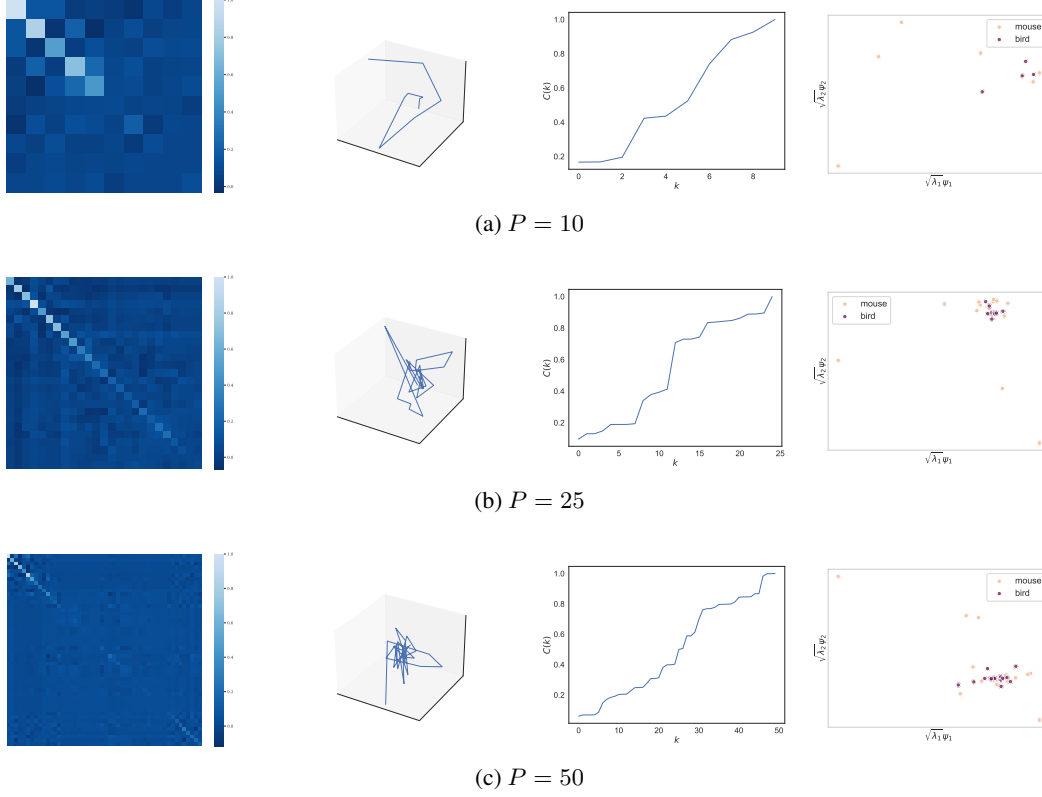


Figure 4: Each row visualizes the kernel resulting from training a SNN with STDP on the population codes from V1 of mice for different number of training examples  $P$ . From the left: Kernel of spike activations; Random 3D Projection of spike activations; Power of Kernel; The top two eigenvectors of the Kernel.

allowing efficient encoding and separation. This interpretation is also consistent with the ability of ANNs to learn simple mappings when the input data is well-structured.

In contrast, the Spiking Neural Network (SNN) does not seem to reveal significant inductive biases in the V1 population code. Increasing the number of training examples  $P$  has a limited effect on improving the kernel’s value for decoding population responses. This may reflect the limitations of the unsupervised Spike-Timing-Dependent Plasticity learning rule, which adjusts synaptic weights based solely on temporal correlations. Since STDP is unsupervised, it could be argued that the information it encodes is already present in the population responses, without introducing additional task-relevant structure. Future work could, therefore, investigate biologically plausible supervised learning rules, such as spike-based backpropagation, to determine whether they can more effectively disentangle representations and reveal task-specific information in the neural code.

A notable limitation of this study is the lack of hyperparameter optimization in the SNN implementation. Due to computational constraints, off-the-shelf hyperparameters from the implementation in [5] were used with minimal tuning. Throughout working on this project, I quickly realized that SNNs have a vast amount of hyperparameters that need to be effectively tuned for it to function properly. Lack of tuning might therefore have affected the SNN analysis much more than it did the ANN analysis. Future work should include thorough hyperparameter optimization to fully evaluate the potential of SNNs in this context.

In sum, this study highlights the ability of a one-layer ANN to efficiently disentangle the population code in V1. In contrast, the SNN trained with STDP showed limited improvement in decoding performance, which I hypothesise is either due to it already being encoded in the population code or suboptimal implementation. Future work addressing these limitations could provide deeper insights into how population codes support efficient and robust learning in the brain.



## References

- [1] Joseph J Atick and A Norman Redlich. “Towards a theory of early visual processing”. In: *Neural computation* 2.3 (1990), pp. 308–320.
- [2] Blake Bordelon and Cengiz Pehlevan. “Population codes enable learning from few examples by shaping inductive bias”. In: *Elife* 11 (2022), e78606.
- [3] Abdulkadir Canatar, Blake Bordelon, and Cengiz Pehlevan. “Spectral bias and task-model alignment explain generalization in kernel regression and infinitely wide neural networks”. In: *Nature communications* 12.1 (2021), p. 2914.
- [4] James J DiCarlo, Davide Zoccolan, and Nicole C Rust. “How does the brain solve visual object recognition?” In: *Neuron* 73.3 (2012), pp. 415–434.
- [5] Peter U Diehl and Matthew Cook. “Unsupervised learning of digit recognition using spike-timing-dependent plasticity”. In: *Frontiers in computational neuroscience* 9 (2015), p. 99.
- [6] A.L. Hodgkin and A.F. Huxley. “A quantitative description of membrane current and its application to conduction and excitation in nerve”. In: *The Journal of Physiology* 117 (1952), pp. 500–544. DOI: 10.1113/jphysiol.1952.sp004764.
- [7] Eugene M Izhikevich. “Which model to use for cortical spiking neurons?” In: *IEEE transactions on neural networks* 15.5 (2004), pp. 1063–1070.
- [8] Eugene M. Izhikevich. *Dynamical Systems in Neuroscience: The Geometry of Excitability and Bursting*. Cambridge, MA, USA: MIT Press, 2007.
- [9] Eugene M. Izhikevich. “Simple model of spiking neurons”. In: *IEEE Transactions on Neural Networks* 14.6 (2003), pp. 1569–1572. DOI: 10.1109/TNN.2003.820440.
- [10] Nikola K. Kasabov. *Time-Space, Spiking Neural Networks and Brain-Inspired Artificial Intelligence*. Berlin/Heidelberg, Germany: Springer, 2019.
- [11] Wolfgang Maass. “Networks of spiking neurons: The third generation of neural network models”. In: *Neural Networks* 10.9 (1997), pp. 1659–1671. ISSN: 0893-6080. DOI: 10.1016/S0893-6080(97)00011-7.
- [12] F. Pedregosa et al. “Scikit-learn: Machine Learning in Python”. In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.
- [13] Pehlevan-Group. *Sample\_efficient\_pop\_codes*. [https://github.com/Pehlevan-Group/sample\\_efficient\\_pop\\_codes](https://github.com/Pehlevan-Group/sample_efficient_pop_codes). Accessed: 2024-12-12. 2022.
- [14] Daniel S Reich, Ferenc Mechler, and Jonathan D Victor. “Independent and redundant information in nearby cortical neurons”. In: *Science* 294.5551 (2001), pp. 2566–2568.
- [15] Michael N Shadlen and William T Newsome. “The variable discharge of cortical neurons: implications for connectivity, computation, and information coding”. In: *Journal of neuroscience* 18.10 (1998), pp. 3870–3896.
- [16] Sen Song, Kenneth D Miller, and Larry F Abbott. “Competitive Hebbian learning through spike-timing-dependent synaptic plasticity”. In: *Nature neuroscience* 3.9 (2000), pp. 919–926.
- [17] Marcel Stimberg, Romain Brette, and Dan FM Goodman. “Brian 2, an intuitive and efficient neural simulator”. In: *eLife* 8 (Aug. 2019). Ed. by Frances K Skinner, e47314. ISSN: 2050-084X. DOI: 10.7554/eLife.47314.
- [18] A. Strickholm. “Ionic permeability of K, Na, and Cl in potassium-depolarized nerve. Dependency on pH, cooperative effects, and action of tetrodotoxin”. In: *Biophysical Journal* 35 (1981), pp. 677–697. DOI: 10.1016/S0006-3495(81)84891-8.
- [19] C. Stringer. *MouseLand / stringer-pachitariu-et-al-2018b*. <https://github.com/MouseLand/stringer-pachitariu-et-al-2018b>. Accessed: 2024-12-12. 2018.
- [20] C. Stringer et al. “Recordings of 10,000 neurons in visual cortex in response to 2,800 natural images”. In: *Figshare* (2018). DOI: 10.25378/janelia.6845348.
- [21] Carsen Stringer et al. “High-dimensional geometry of population responses in visual cortex”. In: *Nature* 571.7765 (2019), pp. 361–365.
- [22] Kashu Yamazaki et al. “Spiking neural networks and their applications: A review”. In: *Brain Sciences* 12.7 (2022), p. 863.
- [23] Friedemann Zenke and Surya Ganguli. “Superspike: Supervised learning in multilayer spiking neural networks”. In: *Neural computation* 30.6 (2018), pp. 1514–1541.