BUILDING A FOUNDATION MODEL FOR NEUROSCIENCE

A Dissertation
Presented to
The Academic Faculty

By

Mehdi Azabou

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in the
Machine Learning Program
School of Electrical and Computer Engineering

Georgia Institute of Technology

December 2024

BUILDING A FOUNDATION MODEL FOR NEUROSCIENCE

Thesis committee:

Dr. Eva L. Dyer
School of Biomedical Engineering
Georgia Institute of Technology

Dr. Blake Richards
School of Computer Science and Montreal Neurological Institute
McGill University

Dr. Chethan Pandarinath
School of Biomedical Engineering
Georgia Institute of Technology

Dr. Hannah Choi
School of Mathematics
Georgia Institute of Technology

Dr. Anqi Wu
School of Computational Science and Engineering
Georgia Institute of Technology

Date approved: August 20, 2024

ACKNOWLEDGMENTS

First and foremost, I would like to thank my advisor, Eva Dyer. You have helped me grow both as a researcher and as a person, I am very grateful to you for having guided me through this, I couldn't have asked for a better advisor. When I joined the lab, I told you I wanted to learn more about neuroscience and research, and the outcome surpassed my expectation. You have thought me so much, you have pushed me in so many ways, encouraged me to think critically and think outside of the box. I am forever grateful.

To all my lab mates, Henry, Vinam, Divyansha, Zihao, Jingyun, Venky, Zijing, Michael, Carolina, Daniel, Santosh, and Ian, I am grateful to have you part of my journey, thank you for our many scientific -and sometimes not scientific- discussions. My work would not have been possible without you. For those who I have helped mentor, I am grateful that I was part of your journey, and I hope I was successful at sharing the joy of research with you.

To all my collaborates, thank you for giving me the opportunity to learn from you and work with you. Getting to learn from people with diverse perspectives has been a treasured experience. Thank you to Shantanu Thakoor, Michal Valko, Petar Veličković, Blake Richards, Guillaume Lajoie, Matthew Perich, Patrick Mineault, Ximeng Mao, Cole Hurwitz, Yizi Zhang, Liam Paninski, Aidan Schneider, Keith Hengen, and Nauman Ahad. Special thanks to Moahmmad Gheshlaghi Azar who taught me many things that I still carry with me today, thank you for the research you exposed me to, and for the advice you've given me over the years. I would like to also thank my thesis committee, Chethan Pandarinath, Anqi Wu, Blake Richards and Hannah Choi, thank you for pushing me to think more critically about my work.

Perhaps most importantly, I want to thank my family for their support. Thank you Fehmi, Walid, Aicha, Mom, and Dad. Thank you Mom for instilling a love for

math and science in me when I was younger, I wouldn't be here without you. Thank you to all my friends who have supported me throughout the years. Thank you Killian for pushing me to go after my dreams, and inspiring me to do things that can make a difference. Thank you Maks Sorokin for our endless discussions, we started at Georgia Tech at a similar time, and I am happy that we got to share this experience together. Special thanks to you and Anna for being my family away from home.

TABLE OF CONTENTS

Acknowledgments iii

List of Tables x

List of Figures xii

Summary xv

Chapter 1: Introduction 1

1.1 Motivation 1

1.2 Contributions 3

1.2.1 A unified scalable framework for neural population decoding . 3

1.2.2 A self-supervised approach for multi-timescale behavior representation learning 4

1.2.3 Learning invariances in neural population activity 5

Chapter 2: A Unified, Scalable Framework for Neural Population Decoding 7

2.1 Introduction 7

2.2 Approach 9

2.2.1 Tokenizing neural population activity 9

2.2.2 Building the latent space 12

2.2.3 Encoding relative timing information 12
2.2.4 Querying the latent space 13
2.2.5 Unit identification in new sessions 14
2.3 Application to monkey motor cortical datasets and movement decoding tasks 15
2.3.1 Datasets and experiment setup 15
2.3.2 Testing the model on single sessions 16
2.3.3 `POYO-mp`: Building a large pretrained across-animal, multi-session model 17
2.3.4 Transferring to new sessions 19
2.3.5 `POYO-1`: A multi-lab, multi-task model for neural decoding . . 22
2.4 Application to human decoding of hand writing 24
2.5 Application to mouse multiregion decoding of visual stimuli and behavior 24
2.6 Related Work 25
2.7 Discussion 28

Chapter 3: A self-supervised approach for multi-timescale behavior representation learning 30

3.1 Introduction 30
3.2 Method 33
3.2.1 Problem setup 33
3.2.2 Histogram of Actions (`HoA`): A novel objective for predicting future 33
3.2.3 Multi-timescale bootstrapping in a temporally-diverse architecture 35
3.2.4 Putting it all together 37

3.3 Experiments . . . 38
3.3.1 Simulated Quadrupeds Experiment . . . 38
3.3.2 Experiments on Mouse Triplet Dataset . . . 40
3.3.3 Results . . . 42
3.3.4 Ablations . . . 44
3.3.5 Experiments on Fruit Fly Groups Dataset . . . 45
3.4 Related Work . . . 47
3.4.1 Animal behavior analysis . . . 47
3.4.2 Representation learning for sequential data . . . 48
3.5 Discussion . . . 49

Chapter 4: Learning invariances in neural population activity 52

4.1 Introduction . . . 52
4.2 Mine Your Own vieW (`MYOW`) . . . 55
4.2.1 Combining augmented and mined views through cascaded predictors . . . 55
4.2.2 How to mine views . . . 56
4.2.3 Memory and computational requirements . . . 58
4.3 Experiments . . . 58
4.3.1 Image datasets: Comparisons and ablations . . . 59
4.3.2 Neural datasets: Identifying classes of augmentations . . . 61
4.3.3 Neural datasets: Examining and comparing representation quality 63
4.4 Related Work . . . 67
4.5 Conclusion . . . 69

Chapter 5: Afterword 71

Chapter 6: Appendix 72

6.1 A unified scalable framework for neural population decoding 72

6.1.1 Additional Model Details 72

6.1.2 Datasets 79

6.1.3 Additional Results 82

6.2 A self-supervised approach for multi-timescale behavior representation learning 85

6.2.1 Experimental details: Simulated Quadrupeds 85

6.2.2 Experimental details: Mouse Triplet 88

6.3 Learning invariances in neural population activity 93

6.3.1 Algorithm 93

6.3.2 Mining: Implementation details 93

6.3.3 Experimental setup: Image datasets 95

6.3.4 Experimental details: Neural data 97

6.3.5 Is `MYOW` worth the extra computational load? 101

6.3.6 What makes for good mined views? 102

6.3.7 Ablation on the projector 104

6.3.8 Ablation on the class of transformations 105

6.3.9 Gaining insights into across-sample prediction 106

6.3.10 Augmentations for spiking neural data 107

6.3.11 Visualization of the latent neural space 109

References 110

LIST OF TABLES

2.1 Datasets used to train POYO 15

2.2 Behavioral decoding results across neural recordings from two nonhuman primates performing two different tasks 18

2.3 Behavioral decoding results for datasets from the Neural Latents Benchmark 21

3.1 Linear readouts of robot behavior 39

3.2 Linear evaluation of various models trained on the mouse triplet dataset from MABe 2022 41

3.3 Ablations of BAMS on the Mouse Triplet Dataset 44

3.4 Linear readouts of fly behavior 46

4.1 Accuracy (in %) for classification on CIFAR-10, CIFAR-100 and Tiny Imagenet. 60

4.2 Accuracy (in %) for different classes of transformations 60

4.3 *Accuracy (in %) in the prediction of reach direction from spiking neural activity.* 63

4.4 F1-score (in %) in the prediction of arousal state from spiking neural activity. 66

6.1 Hyperparameters used for training all `POYO` single-session models . . 83

6.2 Unit re-identification results 84

6.3 Linear readouts of mouse behavior 92

6.4 *TS2Vec Linear readouts of mouse behavior.* 92

6.5 Training `BYOL` with adjusted batch size and number of epochs 101

6.6 Mining in online versus. target space in MYOW 102

6.7 Comparing different projector architectures for incorporating mined views 105

6.8 Comparing different projector architectures for incorporating mined views 106

6.9 Ablation of the class of transformations used for mined views 106

6.11 How augmentations impact our ability to decode sleep and wake states accurately 108

6.10 How augmentations impact our ability to decode movements accurately 108

LIST OF FIGURES

2.1 Overview of POYO 11

2.2 Building a multi-session model spanning multiple animals and tasks . 17

2.3 POYO Scaling curves. 19

2.4 Sample and compute efficiency for training, unit identification, and fine-tuning approaches 20

2.5 Scaling up to more tasks and diverse neural and behavioral recording conditions 23

2.6 Multi-session and transfer for decoding imagined letters into text . . . 24

2.7 Results on datasets from the Allen Institute for Brain Science's Neuropixel Visual Survey. 25

3.1 Architecture of Bootstrap Across Multiple Scales (BAMS) 32

3.2 Visualization of the short-term and long-term windows used to build multi-scale similarity in BAMS 36

3.3 Quadrupeds walking on procedurally generated map. 38

3.4 *Multi-Agent Behavior (MABe) - Mouse Triplets Challenge.* (A) Keypoint tracking approaches are used to extract keypoints from many positions on the mouse body in a video. (B) Methods are evaluated across 13 different tasks. 40

3.5 *Sample frame from the Fruit Fly Groups Dataset.* 45

4.1 Overview of MYOW 54

4.2 Examples of mined views from MNIST and CIFAR-10 61

4.3 (A) Sketch of a primate performing a reach task with sample joystick trajectories depicting the center-out reach movement. (B) Increase in decoding accuracy of BYOL and MYOW as we progressively introduce new augmentations. (C) Visualization of the learned representations obtained for the two primates Chewie and Mihi when different SSL methods are applied (embeddings are obtained via t-SNE). This shows how MYOW reveals the underlying structure of the task, as clusters are organized in a circle. 64

4.4 Visualizations of raw and latent spaces (using t-SNE) of 12 hour recordings from mouse (CA1) during free behavior, including sleep and wake. One variable of interest is the arousal state (REM, nREM, Wake). . . 66

6.1 Diversity of neural recordings used to train POYO 79

6.2 Evolution of the unit embeddings with unit-identification on sessions from two different animals. 84

6.3 Single-session model performance for a wide range of values for different hyperparameters 85

6.4 Visualization of different simulated environments 86

6.5 Visualization of the short-term and long-term embeddings in BAMS . 87

6.6 Prediction target for a sample of the MABe Mouse Triplet dataset. . 89

6.7 Visualization of histograms of future actions, for two random action features 90

6.8 Linear evaluation protocol used in BAMS 91

6.9 Visualization of augmentations used for neural activity in MYOW . . 95

6.10 Reach direction prediction task. (a) Sketch of primate performing reaching task. (b) Illustration depicting how the accuracy and δ-accuracy are computed. 100

6.11 Accuracy under linear evaluation, CIFAR10, ResNet18 102

6.12 Examples of views mined by MYOW 103

6.13 Mining class accuracy during training 104

6.14 Mining class confusion matrices at different stages of learning 105

6.15 Understanding predictive learning when augmentations are too local . 107

6.16 Visualization of the learned representation 109

SUMMARY

The brain's complexity enables its remarkable functions, yet this very complexity makes it hard to understand. Current methodologies for recording brain activity often provide narrow views of the brain's function, limited by the constraints of current recording technology and the structured nature of the standard neuroscience experiment. This fragmentation of datasets has hampered the development of robust and comprehensive computational models of brain function that generalize across diverse conditions, tasks, and individuals.

Our work is motivated by the need for a large-scale foundation model in neuroscience– one that can go beyond the limitations of single-dataset approaches and offer a fuller, more comprehensive picture of brain function. In this thesis, we propose novel methodologies and frameworks aimed at addressing the challenges of building such a model. We discuss three main contributions. The first contribution is towards building *scalable and unified approaches* for training on diverse neural datasets. The second contribution aims to develop *self-supervised methods for understanding dynamics at multiple timescales*, which is a key challenge in dealing with a system that is modulated by short- and long-term dynamics. The third contribution of the thesis is to develop *methods for building invariances in neural data* to further our understanding of the brain.

This thesis pushes the boundaries of brain modeling, offering new methodologies for integrating heterogeneous data, improving neural decoding, and building robust, generalizable models. By bridging the gap between isolated datasets, we aim to advance our understanding of the brain and open new avenues for comparing neural activity across individuals and improving brain-machine interfaces.

CHAPTER 1
INTRODUCTION

1.1 Motivation

Understanding the brain is one of the most profound and challenging scientific endeavors of our time. The brain is not only the seat of consciousness and identity but also the organ that enables all human experiences, from basic survival instincts to the most complex thoughts and emotions. Deciphering its mechanisms holds the key to enhancing cognitive abilities, addressing a myriad of neurological and psychiatric disorders, and creating advanced artificial intelligence systems. Despite significant advancements in neuroscience, our understanding of the brain's networks and dynamic processes remains incomplete. Advancing our understanding of the brain is crucial for both scientific progress and the betterment of human health and society.

The brain is a complex system composed of billions of neurons across multiple regions and is constantly involved in diverse and complex tasks. When recording from the brain, however, we do not observe it fully, we observe it through a slice along both spatial —few neurons at a time— and task —a specific task and context— dimensions. Indeed, existing invasive recording devices only record a few dozen to a few thousand neurons, and recordings are usually conducted under structured and meticulously designed experiments where one or a few brain regions are targeted and where a subject is usually given a repetitive simple task. Thus, a single dataset captures a narrow slice of the brain, and thus we are limited in insights we can derive. Most existing computational models are trained on individual datasets, which limits their ability to generalize across different conditions and subjects. This one-dataset-at-a-time approach often results in models that are overly specialized and lack the robustness

needed to capture the brain's complexity. We need an approach that can leverage multiple datasets, and stitch together all these different slices, to build a fuller picture of the brain that truly reflects its complex multifaceted nature.

To advance our understanding of the brain, we need a foundation model, a large-scale generalist model that is trained on multiple diverse datasets. Integrating diverse datasets allows for the exploration of common patterns and unique variations across different brain states, tasks, and individuals. This holistic approach can lead to the development of models that not only perform better in individual tasks but also exhibit transfer capabilities across various domains and populations. Such integration poses significant technical challenges, that we strive to address in this thesis. First, because recording devices only record a random small number of neurons, each recorded population is going to be virtually unique from session to session and from subject to subject. This explains why most current approaches are built to operate on a single population of neurons at a time. Across days and across individuals, the set of neurons that are being recorded changes, rendering a model trained on a particular population unreliable. Second, labels are not always possible to collect, especially in unconstrained, free-behavior settings. This requires self-supervised learning methods to learn representations from a wider range of datasets.

In this thesis, we propose novel methodologies and frameworks that facilitate the integration of multiple datasets into unified models. Our work focuses on developing scalable algorithms that can efficiently handle large and heterogeneous data, employing advanced techniques in machine learning and neuroscience. We demonstrate the effectiveness of our approach through rigorous experiments, showcasing its ability to enhance model performance and uncover new insights into brain function. By bridging the gap between isolated datasets, we aim to push the boundaries of what is possible in brain modeling, opening new avenues for research and application.

1.2 Contributions

This thesis has three contribution chapters, each addressing a unique challenge in building models of the brain and behavior. Collectively, these contributions move us further towards establishing a foundation model for neuroscience.

1.2.1 A unified scalable framework for neural population decoding

Current neural population decoding models are typically built to operate on a single population of neurons at a time, because across days and across individuals, the set of neurons that are being recorded changes. First, this limits the use of such models in brain-computer interface applications, since frequent and extensive calibration is required. Second, this goes against a fundamental principal in deep learning, where substantial amounts of data are usually needed to train expressive models that perform well.

In Chapter 2, we propose a novel framework that addresses these core challenges. POYO is a transformer model that can take in an arbitrary number of neurons, in any order, to build a scalable model that is trained a large collection of datasets. To demonstrate the flexibility and scalability of our multi-session training framework, we apply it to a wide range of motor, visual, and decision making tasks from electrophysiology datasets collected in monkeys, humans, and mice. In particular, we train three large models, the first being from 7 nonhuman primates (27,373 units) engaging in four different motor tasks across three laboratories, the second combining human and monkey motor cortical datasets (9,789 monkey units + 1,920 human units) in diverse tasks, and the third from 58 mice (99,180 units) engaged in visual tasks during recordings to multiple regions in the cortex, deep, and midbrain structures. In these cases, we demonstrate that our pretrained models can be rapidly adapted to new, unseen sessions, enabling few-shot performance across animals and species

with minimal labels. Our results highlight the power of scale and demonstrate how pretraining on large datasets can benefit neural data analysis.

1.2.2 A self-supervised approach for multi-timescale behavior representation learning

Brain and behavior are intimately tied, to understand the brain, requires that we also understand behavior. We do this, for example, with brain-machine interfaces in Chapter 2 where we train models to decode visual and motor tasks and thus infer the link between brain and behavior. But this only tells us what happens at a sub-second timescale, and only for lower-level brain functions, in reality, the brain is involved in much more complex behavior that is modulated at different time scales. For such time scales, we usually do not have labels: we are able to record videos of the subject, track their position, the position of their limbs, or their eye gaze, but we cannot say much about their mental state, or the task they are engaging in, and how they are going about solving that task, at least without further analyzing the observed behavior.

To build a generalist foundation model that can give a full picture of the brain functions, we cannot limit ourselves to only datasets that have labels, we need to figure out how we can work with neural datasets for which we do not have clean behavior labels, datasets that might be recorded in brain regions responsible for higher order processes, and that might not directly correlate with behavior at the sub-second level.

Chapter 3 deals with learning multi-timescale representations of behavior. In this chapter, we develop a multi-task representation learning model for animal behavior that combines two novel components: (i) an action-prediction objective that aims to predict the distribution of actions over future timesteps, and (ii) a multi-scale architecture that builds separate latent spaces to accommodate short- and long-term dynamics. After demonstrating the ability of the method to build representations of both local and global dynamics in robots in varying environments and terrains, we apply our method to the MABe 2022 Multi-Agent Behavior challenge, where our

model ranks first overall on both mice and fly benchmarks. In all of these cases, we show that our model can build representations that capture the many different factors that drive behavior and solve a wide range of downstream tasks.

1.2.3 Learning invariances in neural population activity

Interpretability is important to consider when building tools that are aimed at understanding the brain. Sometimes, the goal is not to decode behavior, but to understand the dynamics underlying the brain state of an individual during a given task. Typically, this is done through dimensionality reduction, or more generally representation learning. Traditional dimensionality techniques aim to capture the main directions of variability in the neural population activity, but these dimensions can be noisy: the electrodes in the brain can drift over time, or the subject can be engaged in auxiliary tasks that are irrelevant to the study. To build representations that are invariant to these potential sources of noise, we leverage self-supervised techniques that use augmentations to control which desired invariances are built into the representation.

Chapter 4 introduces Mine Your Own vieW (MYOW), a self-supervised learning approach that builds representations by maximizing the similarity between different "views" of a sample. Like in other domains, designing good augmentations generally requires a significant level of domain knowledge. With MYOW, we introduce a set of augmentations designed for neural population activity, as well as, a new strategy that looks within the dataset to find similar views of the brain state. The idea behind our approach is to actively mine views, finding samples that are neighbors in the representation space of the network, and then predict, from one sample's latent representation, the representation of a nearby sample. After showing the promise of MYOW on benchmarks used in computer vision, we highlight the power of this idea in two novel applications in neuroscience. When tested on multi-unit neural recordings, we find that MYOW outperforms other self-supervised approaches in all examples (in

some cases by more than 10%), and often surpasses the supervised baseline. With MYOW, we show that it is possible to harness the diversity of the data to build rich views and leverage self-supervision in new domains to build more insights about the data.

CHAPTER 2
A UNIFIED, SCALABLE FRAMEWORK FOR NEURAL POPULATION DECODING

2.1 Introduction

Recent advances in machine learning, particularly in the context of large-scale pre-trained models like GPT [1, 2, 3, 4], have showcased the immense potential of scaling up, both the terms of the size of datasets and models [5, 6]. Similarly, in neuroscience, there is a growing need for a foundational model that can bridge the gaps between diverse datasets, experiments, and individuals, allowing for a more holistic understanding of brain function and information processing [7]. The development of such a model would allow researchers to uncover underlying patterns and interactions within neural populations [8], and potentially allow for more robust decoding of brain states.

However, creating a large-scale neural decoding model that can effectively combine spiking datasets from various sources is a complex challenge [9]. One of the central challenges is the lack of a shared "vocabulary" in neural recordings. Unlike the case for text—wherein every document written in a given language shares a basic lexicon for tokenization—there is no one-to-one correspondence between neurons in different individuals. As such, every recording from a different individual involves a unique set of neurons that cannot be easily aligned with another set. An additional core challenge lies in the inherent variability of neuron sets observed across different days [10]. Even when monitoring the same individual, variability in electrode/tissue interface can lead to distinct neuron sets which, despite advanced sorting methods, can lead to inconsistencies in input channels across sessions [11, 12, 9]. Overall, this lack of correspondence across recordings complicates the integration of information from

different experiments and individuals, ultimately hampering efforts to construct a unified perspective on population-level interactions and dynamics in the brain.

In response to these challenges, we propose a new framework for large-scale training on neural spiking data called `POYO` (**P**re-training **O**n man**Y** neur**O**ns).[1] This framework is designed to enable scalable and efficient training across multiple sessions of neural recordings, even when spanning different sets of neurons with no known correspondence. Our approach centers around a novel tokenization scheme that transforms individual neural action potentials, or "spikes", into discrete tokens, thereby preserving the neural code's finest temporal structure while simultaneously enhancing computational efficiency. The resulting tokenization not only allows for a more effective representation of neural activity but also paves the way for training on larger volumes of data. We combine this input tokenization method with an architecture that builds on the PerceiverIO [13] to compress the input spikes into a latent space and learns interactions across spikes in time and across neurons.

We evaluate the performance of our proposed approach on data from over 158 sessions from open electrophysiology datasets from seven non-human primates (NHPs), spanning over 27,373 units and 100 hours of recordings. We demonstrate that through pretraining on large amounts of data, we can transfer with very few samples (few-shot learning) and thus improve overall brain decoding performance. Our work not only presents an innovative framework for training large models on neuroscience datasets, but also offers insights into the scaling laws that govern decoding from neural populations. By enabling the development of large pretrained models for neural decoding, our approach advances the field of brain-machine interfaces and other decoding applications.

The main contributions of this work include:

- *A framework for large-scale training on neural recordings:* We present a novel

[1] Poyo is the exclamation used by Kirby, who has an insatiable appetite. Similarly, `POYO` consumes spikes from many neural datasets and combines them into one unified model.

framework for training transformer models end-to-end on multi-session and across-individual electrophysiology datasets derived from neural populations, enabling efficient and effective decoding from a diverse range of neural recordings.

- *Innovative spike-based tokenization strategies:* We introduce a fundamentally different way to tokenize neural population activity. Our approach tokenizes individual spikes (events) across neural populations, preserving fine temporal structure and enhancing computational efficiency by adopting a sparse representation of the data.

- *Pre-trained models for neural decoding:* We build two large pretrained models (`POYO-1`, `POYO-mp`) that can be fine-tuned on new sessions and across recordings from different animals and new behavioral tasks. We will make the weights and code available, and provide both pretrained models as a resource to the community.

2.2 Approach

The transformer architecture [14], originally introduced in the context of natural language processing (NLP), has shown remarkable flexibility and effectiveness in various domains, especially in the presence of large and diverse datasets [15, 3]. In this work, we explore how to leverage this versatility in the neural data domain.

2.2.1 Tokenizing neural population activity

Neurons communicate asynchronously using electrical impulses called spikes. The timing and frequency of spikes encode signals that convey information about the external world and coordinate the internal dialogue within the brain. In many neuroscience experiments, neural activity is recorded from the brain through multiple electrodes, and then processed [16] to extract the spiking events for a set of neural

"units" [17]. The resulting data are multi-variate event-based sequences that are typically very sparse relative to the number of recorded time points.

Neurons and their spikes are of course not independent of one another. Rather, they are part of a complex, interwoven tapestry of neural activity, with each spike contributing to a collective dialogue across billions of neurons. The true challenge and opportunity lie in interpreting these spikes not in isolation, but in the context of this broader "conversation."

When trying to scale up and train on many sessions of data, we are faced with the challenge of having recordings of neural conversations with completely different set of "speakers" for which we don't know the identity or their functional tuning (or what they respond to). This is because each time we record from the brain, we are tuning into a conversation between a new set of speakers. However, as with language, there is some reason to think that neurons are ultimately communicating on similar topics, e.g. sensations from the external world, internal states of the body, muscle commands, etc. In other words, the lexicon may be different, but everyone is talking about similar subjects. Despite the variability of our data, our aim is to decipher these "conversations" in a way that generalizes to new neural datasets without fixed or known correspondence across their inputs.

The neural tokenizer. Based upon these motivations, we propose a novel approach for tokenizing neural population activity where *each spike is represented as a token* (see Figure 4.1). In this case, each token can be defined in terms of *which unit it came from* (via a learnable embedding) and the time that the spike event was detected. By representing our data in this way, we never define or fix the expected number of units, and the model can ingest populations of arbitrary size, and thus can be trained across many datasets. At the same time, this approach also avoids having to specify a specific time resolution for the neural code, as is the case with binning, and gets rid

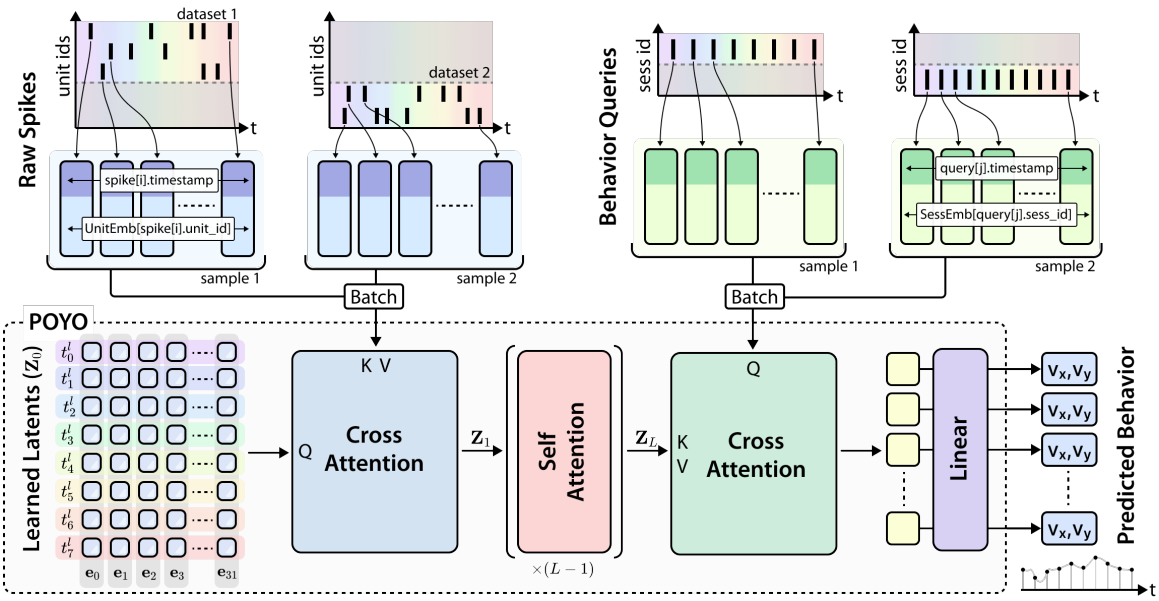


Figure 2.1: *Overview of POYO.* Input spike tokens are compressed into a smaller set of latent tokens which are processed through multiple self-attention blocks, finally time-varying outputs are predicted by querying the latent space. All tokens in this model are assigned a timestamp, which are used towards rotary position encoding.

of the accumulation of a lot of sparse tokens containing no events.

More concretely, we assign each unit a unique identifier and a corresponding D-dimensional learnable embedding. Let UnitEmbed$(\cdot)$ denote a lookup table that associates each unit to its unit embedding. Akin to word embeddings that encode semantic meaning and relationship between words, the unit embedding space encodes something about the "speaker's" identity and role in neural computation. A unit fires a sequence of spikes in a context window of time $[0, T]$. Each spike will be represented by a token characterized by $(\mathbf{x}_i, t_i)$, where

$$\mathbf{x}_i = \text{UnitEmbed}(\text{spike i}'\text{s unit})$$

is the learned embedding associated with the unit that emitted the spike and t_i is the event timestamp.

Collectively, and for an arbitrary set of units, we combine all the spikes into a

sequence of length M. The neural population activity is represented by $[\mathbf{x}_1, \ldots, \mathbf{x}_M]$ and their corresponding times $[t_1, \ldots, t_M]$. While the size of our context window T stays fixed, M will vary depending on the number of units in the population and their firing rates (see Figure 4.1). Note that all spikes from a specific unit have the same unit embedding, and only differ in their timing.

2.2.2 Building the latent space

Compressing the input sequence. Rather than processing the input sequences with self-attention, which is quadratic in sequence length and can be expensive for long sequences, we use a perceiver encoder module [18] that summarizes the input sequence into a shortened "latent" sequence. Let's consider a sequence of M input tokens $\mathbf{X} = [\mathbf{x}_1, \ldots, \mathbf{x}_M]$ and a sequence of learned latent tokens $\mathbf{Z}_0 = [\mathbf{z}_{0,1}, \ldots, \mathbf{z}_{0,N}]$, where $\mathbf{z}_{0,i} \in \mathbb{R}^D$ and $N \ll M$. We use cross-attention to pull information from the input sequence into a compressed latent sequence. The queries $\mathbf{Q} = \mathbf{W}_q\mathbf{Z}_0$ are a projection of the learned latent tokens $\mathbf{Z}_0$ and the keys and values are a projection of the input tokens: $\mathbf{K} = \mathbf{W}_k\mathbf{X}$ and $\mathbf{V} = \mathbf{W}_v\mathbf{X}$, respectively. We use the standard transformer block with pre-normalization layers and feed-forward nets.

Attention in the latent space. Following the compression of the input sequence through cross-attention, we apply multiple self-attention blocks on the latent token sequence, which now costs $O(N^2)$ instead of $O(M^2)$ with $N \ll M$. Let $\mathbf{Z}_l$ denote the embedding of the latent tokens at the lth layer. We denote the final latent sequence obtained after L layers as $\mathbf{Z}_L = [\mathbf{z}_{L,1}, \ldots, \mathbf{z}_{L,N}]$.

2.2.3 Encoding relative timing information

To incorporate timing information, we leverage rotary position encoding (RoPE) [19] across all attention layers in the architecture. Unlike traditional position encoding methods that inject absolute position information into the token's embedding, RoPE

applies a rotation operation in the query, key embedding space, allowing each token in a sequence to attend to others based on its relative positional information.

Recall that each input token i has an associated timestamp t_i. We will also assign a timestamp to each of the latent tokens: we divide the latent tokens into groups of equal size and spread each group uniformly over the context window $[0, T]$. This method allows us to capture temporal relationships between the latent tokens and the input tokens, enabling a temporal understanding of the encoded sequence. By distributing the latent tokens evenly within the context window $[0, T]$, we create a structured temporal representation, which preserves and propagates temporal information all the way through the model.

2.2.4 Querying the latent space

Having built a latent space which can encode and model any population of units, we now want a flexible way to readout behavioral variables. The output of the latent encoder is the latent sequence of size N, while the desired output can be any arbitrary sequence of length P. Let us consider the task of hand velocity decoding for example, in the context window $[0, T]$, the length of the output sequence will depend on the sampling frequency. Since we aspire to train on datasets sourced from various labs, the sampling rate can differ significantly. We thus need a flexible mechanism for predicting outputs of varying lengths and querying from the neural activity at specific points in time.

We define a sequence of output tokens $\mathbf{Y}_0 = [\mathbf{y}_{0,1}, \cdots, \mathbf{y}_{0,P}]$, where P is the number of output time points, which can change across sequences. Each output token is defined by $(\mathbf{y}_{0,i}, t_i^{out})$. Initially all the output tokens within a session are set to the same learned embedding $\mathbf{y}_{0,i} = \mathbf{y}_0 \in \mathbb{R}^D \ \forall i$. The output $\mathbf{Y}$ is obtained by querying the latent space through cross-attention.

Since we use rotary embeddings, and both the latent and output tokens have

assigned timestamps, the querying mechanism can leverage the relative position of latent and output tokens to extract the temporally relevant context that enable prediction of the behavioral variable of interest.

Session embeddings. To account for variability of the experimental setups in real world settings, we propose to define a learnable session embedding which captures the hidden experimental variables. This information is injected into the output query $\mathbf{y}_0$, where again we use a lookup table to register each new session we encounter. This produces a set of output tokens of the dimension of the number of queries which we then pass through an MLP to generate the behavioral variables of the desired dimension (e.g., 2D for hand velocities in a planar movement task).

2.2.5 Unit identification in new sessions

Our design of the unit embedding space allows our model to learn latent information about the units it encounters, as well as capture the relationship between units in the population. Given a new recording with unidentified units, we can transfer our model by mapping these new units into the unit embedding space. To do this, we introduce an approach that we call "unit identification", which leverages gradient descent to learn the embeddings of new units. In this approach, we freeze all existing weights of the model and simply add new rows to the UnitEmbed($\cdot$) lookup table for each of the new units, as well as a new session embedding. Notably, the bulk of our model which maps the neural population activity to behavior is unchanged and is simply transferred to the new dataset. In our experiments, we find that this approach is surprisingly effective and allows us to rapidly integrate new datasets into the same underlying model.

2.3 Application to monkey motor cortical datasets and movement decoding tasks

In a first application of our approach, we curated a multi-lab dataset with electrophysiological recordings from motor cortical regions, where neural population activity has been extensively studied [10], and deep learning tools and benchmarks like the Neural Latents Benchmark (NLB) [24] have recently been established.

2.3.1 Datasets and experiment setup

One of the key advantages of our approach is its ability to scale to handle large amounts of neural data, including sessions from different numbers of neurons, across different tasks and recording setups, and from different animals. Thus we set out to build a diverse dataset large enough to test our approach. We curated a multi-lab dataset with electrophysiological recordings from motor cortical regions, where neural population activity has been extensively studied [10], and deep learning tools and benchmarks have recently been established [24]. In total, we aggregated 178 sessions worth of data, spanning 29,453 units from the primary motor (M1), premotor (PMd), and primary somatosensory (S1) regions in the cortex of 9 nonhuman primates (see Table 2.1). We place this in the context of standard analyses within a single lab or paper which typically involve 10's of sessions and a few hundred neurons.

All of these neural recordings were collected while the animals performed various

Table 2.1: *Datasets used to train POYO.* CO: Center-Out, RT: Random Target.

Study	**Regions**	**# Indiv**	**# Sess**	**# Units**	**# In**	**# Out**	**Tasks**
Perich et al. [20]	M1, PMd	4	117	11,557	143M	20M	CO, RT
Churchland et al. [21]	M1	2	9	1,728	706M	87M	CO
Makin et al. [22]	M1, S1	2	47	14,899	123M	15M	RT
Flint et al. [23]	M1	1	5	957	7.9M	0.3M	CO
NLB-Maze [24]	M1	1	1	182	3.6M	6.8M	Maze
NLB-RTT [24]	M1	1	1	130	1.5M	2.8M	RT

motor tasks that vary in their inherent complexity (see Figure 2.2A-B). The center-out (CO) task is relatively stereotyped, with the animal making a reach to one of eight targets after receiving a go cue, and then returning to the center. In contrast, the random target (RT) task is significantly more complex. The animals make continuous and self-paced movements with new targets appearing in succession at random locations in the workspace. In addition to the greater exploration of the workspace and heterogeneity in movements, this task allows individual to plan their next movement while finishing the execution of the current movement leading to greater complexity in neural dynamics. Other sources of variability that we find across labs include the: choice of pre-processing algorithms (spike sorting or threshold crossing analysis), type of controller (manipulandum, touch screen), and sampling rate when recording behavior (100Hz to 1kHz). We do not re-process the data or attempt to standardize it across labs or tasks. Further details on the datasets are provided in subsection 6.1.2.

Experiment setup. Throughout all of our experiments, we use a context window of $1s$ and do not segment data into trials during training. We only use the trial structure when reporting the decoding performance, in particular, center-out sessions are evaluated during the reaching movement. We train the model with $N = 512$ latent tokens and a dimension $D = 128$. We use the LAMB optimizer [25], and employ a cosine decay of the learning rate at the end of training. For every session, we holdout 20% of the trials for testing, and 10% for validation. We use 1-GPU and 8-GPU setups for single-session and multi-session models, respectively.

2.3.2 Testing the model on single sessions

To first investigate the performance of our architecture when trained on a single session, we trained single-session models on 100 different recording sessions acquired from three nonhuman primates (Monkey C, Monkey M, Monkey Ja), each containing

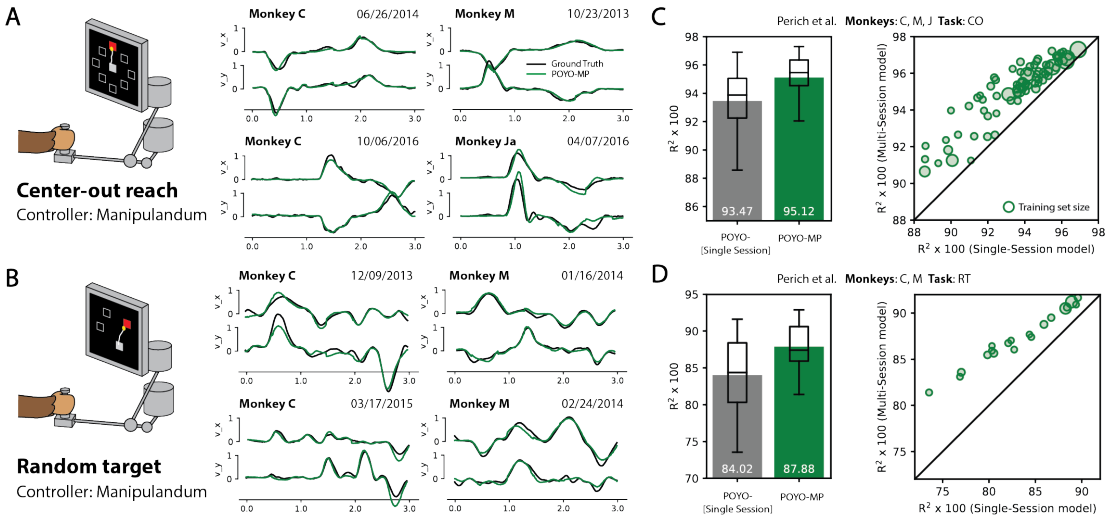


Figure 2.2: *Building a multi-session model spanning multiple animals and tasks.* (A) Center-out reach and (B) Random target task [20], along with examples of true and predicted behavior (x,y velocities). In (C-D), we show the decoding performance for single-session models (gray) and the `POYO-mp` multi-session model (green).

anywhere from 10 to 106 minutes of data [26]. In all of these recordings, the same behavioral task setup, behavioral recording apparatus (10 ms temporal resolution), and spike sorting procedure was used.

Across all 100 single-session models, we obtained an average R2 of 0.9347 on CO and 0.8402 for RT sessions. When we compared these single-session results with existing models for neural decoding, including a Wiener Filter, GRU and MLP [27], we found that our approach consistently outperforms these baselines, with even greater improvements observed on the RT task. Additionally, we found that our single-session models are stable for a wide range of hyperparameters.

2.3.3 `POYO-mp`: Building a large pretrained across-animal, multi-session model

To investigate the question of how training with more sessions of data can improve brain decoding, we trained a large model (`POYO-mp`, 24 layers) on all 100 sessions that we studied in our single-session analysis. In total, we used 9,789 units and 4,367 neuron-hours (number of neurons $\times$ amount of time recorded) to train our `POYO-mp` model.

This model achieves an average test R^2 of 0.9512 on the center-out tasks and 0.8738 on the random target tasks, which is a marked improvement over the single-session average. When we compared the performance of our multi-session model to single-session models head-to-head (see Figure 2.2C-D), we found that across the board, multi-session training improves over single-sessions, and we observed even greater improvements for datasets with fewer trials (indicated by the size of the circles in the plot). On RT sessions, we observe an even bigger improvement over the single session models. These results suggest that there are significant benefits in joint-training across multiple sessions, especially when decoding more complex behaviors like the random target task.

Scaling analysis. By enabling multi-session training, we can start to ask questions about the extent to which having more data or more parameters can improve decoding (Figure 2.3). Thus, we studied the improvements obtained for three different depths of models, with L=6 (3.8M), L=14 (7.4M), and L=26 (13M) layers (parameters). For both CO and RT tasks, we find that even at the same depth (L=6), our 1 multi-session model shows an improvement over the average performance of 100 single-session models. We see further improvements with increasing the model depth, with the RT task

Table 2.2: *Behavioral decoding results across neural recordings from two nonhuman primates performing two different tasks.* All the baselines and the single-session model are trained from scratch, while `POYO-mp` and `POYO-1` are pretrained. The standard deviation is reported over the sessions. The number of sessions in each dataset is contained in $(\cdot)$ and the top performing models in each category are indicated in boldface (first) and underlined (second).

		Same animal, New day	*New animal*	
	Method	Monkey C - CO (2)	Monkey T - CO (6)	Monkey T - RT (6)
From scratch	Wiener Filter	0.8860 ± 0.0149	0.6387 ± 0.0283	0.5922 ± 0.0901
	GRU	0.9308 ± 0.0257	0.8041 ± 0.0232	0.6967 ± 0.1011
	MLP	<u>0.9498</u> ± 0.0119	<u>0.8577</u> ± 0.0242	<u>0.7467</u> ± 0.0771
	`POYO`-[Single-session]	**0.9682** ± 0.0111	**0.9194** ± 0.0185	**0.7800** ± 0.0702
Pre-trained	`POYO-mp` + Unit ID	0.9675 ± 0.0079	0.9012 ± 0.0271	0.7759 ± 0.0471
	`POYO-mp` + Finetune	**0.9708** ± 0.0116	**0.9379** ± 0.0193	<u>0.8105</u> ± 0.0561
	`POYO-1` + Unit ID	0.9677 ± 0.0096	0.9028 ± 0.0248	0.7788 ± 0.0548
	`POYO-1` + Finetune	<u>0.9683</u> ± 0.0118	<u>0.9364</u> ± 0.0132	**0.8145** ± 0.0496

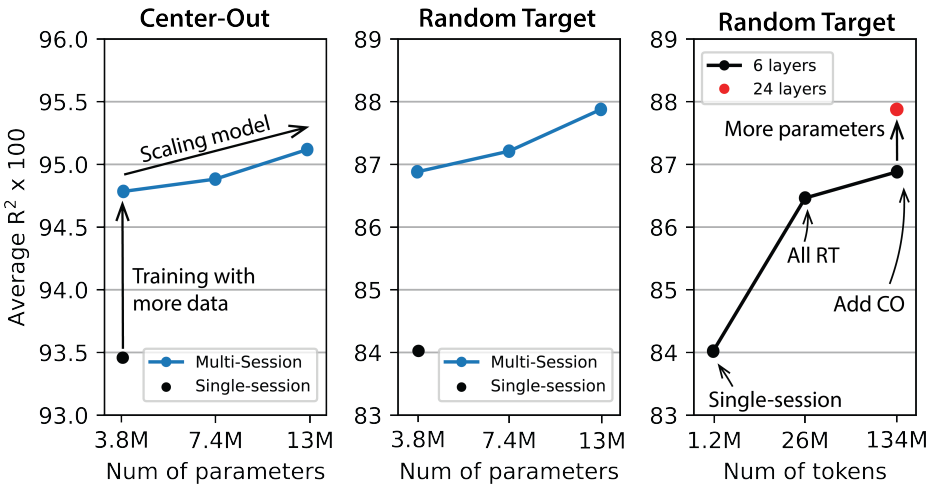


Figure 2.3: *Scaling curves.* R^2 is evaluated and averaged over the test splits on both CO and RT tasks. The single-session performance is the mean over 100 different models.

benefiting even further from scaling. When we fix the model depth to L=6, and study how training jointly with more data contributes to improvement in performance, we find an improvement in performance when comparing single-session training (average 1.2M tokens) to training jointly on all RT sessions (26M tokens). When we also train jointly with other CO sessions (134M tokens in total), and increase model depth to help in accommodating the more growing and diverse training set, we continue seeing improvement in the decoding performance on RT sessions, accumulating to 4% over our single-session baseline.

2.3.4 Transferring to new sessions

After pretraining, we can then test on new sessions with unknown neurons using either the (i) *unit identification* approach we described in subsection 2.2.5, or (ii) full *finetuning* of the model weights. Recall that when we use a unit identification approach, we freeze the weights of the model and only learn new unit and session embeddings.

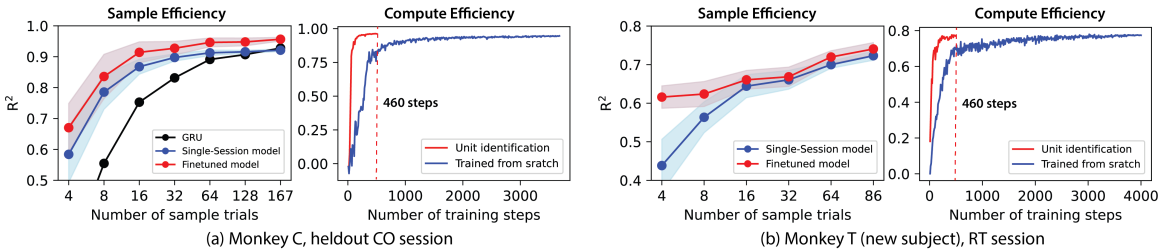


Figure 2.4: *Sample and compute efficiency for training, unit identification, and finetuning approaches.* On the left, we show the sample and compute efficiency for a heldout CO session. On the right, we plot the sample and compute efficiency for a new animal not seen during training.

Results on held out sessions from the same animals. We first tested the unit identification approach on held-out sessions from Monkey C that are unseen during training (Table 2.2, Left). In this case, we don't have correspondence between units in the training and the testing conditions. Surprisingly, we find that we can achieve comparable performance with unit identification on the pretrained model (0.9675) with that of the single-session models trained fully from scratch (0.9682). With further finetuning of the rest of the model's weights, we can improve the accuracy further (0.9708) to go beyond the accuracy of the single-session models. This highlights the robustness of our model and its flexibility to accommodate fresh data with even a simple input mapping.

We compare with the single session model and a GRU baseline (Figure 2.4A) with our multi-session model trained on the same number of trials. Both our single-session and finetuning approach achieve good performance scaling as we increase the number of samples, with the finetuning maintaining a gap over the single session models over the entire sampling space.

Results on a new animal performing the same tasks. Next we tested our model on 12 sessions from a completely new animal (Monkey T) that was not included in the training of the model (Table 2.2). When we applied our unit identification approach on the multi-session model (`POYO-mp` + Unit ID), we see a bit of a dip in overall accuracy from the single-session model (`POYO`-[Single-session]); however, when

Table 2.3: *Behavioral decoding results for datasets from the Neural Latents Benchmark [24].* Best performing model is in bold and second best model is underlined.

	Method	NLB-Maze	NLB-RTT
From scratch	Wiener Filter	0.7485	0.5438
	GRU	0.8887	0.5951
	MLP	0.8794	**0.6953**
	AutoLFADS + Linear [28]	<u>0.9062</u>	0.5931
	NDT + Linear [29]	0.8929	0.5895
	NDT-Sup [30]	0.8708	0.4621
	EIT [30]	0.8791	0.4691
	`POYO-`[Single-session]	**0.9470**	<u>0.6850</u>
Pretrained	`POYO-mp` + Unit ID	0.8962	0.7107
	`POYO-mp` + Finetune	<u>0.9466</u>	<u>0.7318</u>
	`POYO-1` + Unit ID	0.9329	0.7294
	`POYO-1` + Finetune	**0.9482**	**0.7378**

we fine-tune the weights of the model further, we achieve an accuracy of 0.9379 on CO, which is a significant improvement over all of the other baselines. For the RT task, the single session model is 0.7569 and with unit identification we achieve 0.7669 and with full fine-tuning we get up to 0.7916 (Figure 2.4B). These results are promising and show that we can use our pretrained model on new animals with only a few minutes of labeled data.

Performance on the Neural Latents Benchmark (NLB). To understand how well our pre-trained model performs on data collected from new animals performing novel tasks with different equipment (example: touch screen vs. manipulandum), we applied our pretrained model to the MC-Maze (Monkey L) and MC-RTT (Monkey I) datasets from the NLB (Table 2.3)[24]. The NLB serves as a benchmark for neural representation learning and decoding and thus we can include other single-session baselines to our comparisons, including self-supervised models AutoLFADS [28] and NDT [29] which produce denoised firing rate estimates over which we fit a linear layer (+ Linear), a supervised variant of NDT [30] (NDT-Sup), and EIT [30]. Both datasets contain single sessions from new animals performing movement tasks that we haven't seen during training.

On the NLB-Maze dataset, we obtain a R2 of 0.8952 after unit identification, which is competitive with the baselines. These results are surprising since we do not modify the model's weights, and yet our pretrained model yields competitive results on a dataset collected under very different conditions. When we finetune the model, we boost the performance even further establishing a 4.4% gap over the best baseline. Similar trends can be observed for the RTT task (Monkey I), with even larger (2%) improvement after finetuning.

2.3.5 `POYO-1`: A multi-lab, multi-task model for neural decoding

Given the impressive transfer results of `POYO-mp` to datasets from different labs, we ask whether we can use our approach to build a model that spans even more diverse recording setups that we expect to encounter when trying to unify data from many sources. In total, we used datasets from seven nonhuman primates spanning three different labs, with a total of 27,373 units and 16,473 neuron-hours for training our model. We call this pretrained multi-lab, multi-task model `POYO-1`.

Even in light of the high amounts of variability across these different datasets, `POYO-1` provides consistent improvements over the single-session models (Figure 2.5C). When tested on a number of different transfer tasks (Table 2.2, Table 2.3), we again find that the unit identification and finetuning approaches provide effective strategies for adapting our pretrained model to new datasets. We obtain notable performance on the NLB-Maze, where we find that we obtain a R^2 of 0.9329 with only unit identification remapping, an almost 4% improvement over the unit identification result for our `POYO-mp` pretrained model, suggesting that we cover more of the task space with `POYO-1`.

When comparing the `POYO-1` model with `POYO-mp` model, it is clear that both methods have their strengths. `POYO-mp` excels on the datasets sourced from the same lab, with 0.8788 on RT sessions compared to `POYO-1`'s 0.8664. On the other hand,

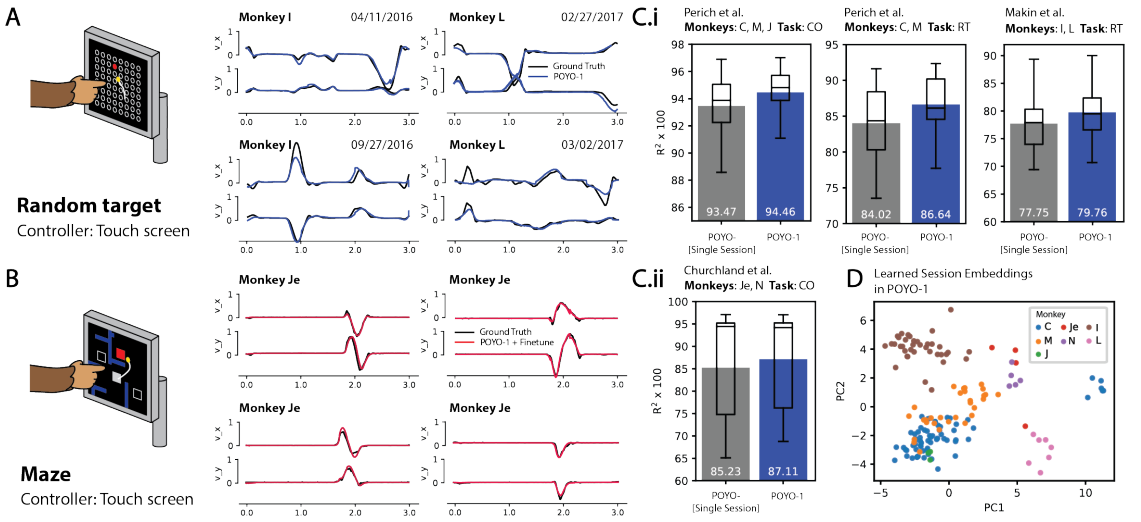


Figure 2.5: *Scaling up to more tasks and diverse neural and behavioral recording conditions.* (A) Random target task from Makin et al. included in training of `POYO-1` and (B) Maze task from NLB heldout for transfer testing. In (C) decoding accuracy for the single-session (gray) and the `POYO-1` model (blue). (D) PCA projection of the learned session embeddings.

both models exhibit great transfer capabilities, with `POYO-1` having the edge especially when using unit identification, indicating its ability to generalize better across diverse experimental conditions. This flexibility and scalability make these methods promising tools for future research in analyzing neural data from diverse sources.

Visualizing the session embedding space. We visualized the learned task embeddings to see if the model learns relationships between sessions seen during training (Figure 2.5D), even though it was not explicitly trained with this information. This analysis revealed clusters corresponding to different data sources and tasks, suggesting that our model has not only learned to identify patterns within each session but also recognized overarching structures across sessions. In particular, we find that the datasets collected in the Miller Lab (C,M,J) used to train the `POYO-mp` model are mapped into similar regions of the session latent space, and (I and L) sessions are mapped to their own distinct clusters.

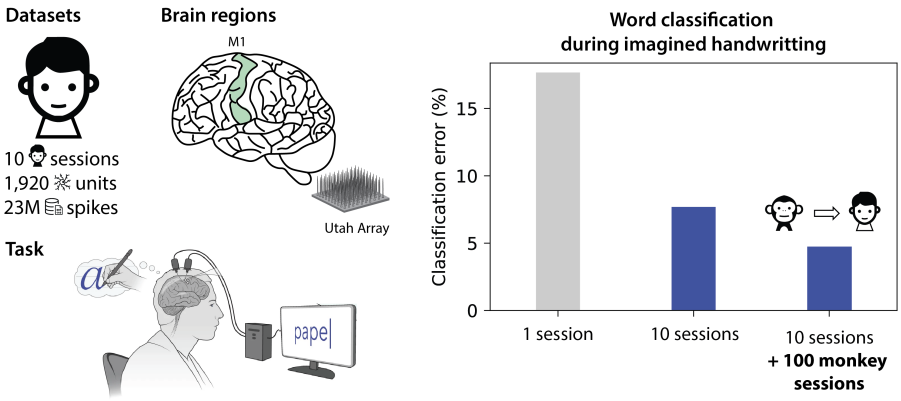


Figure 2.6: Multi-session and transfer for decoding imagined letters into text. The decoding error rate is plotted as we vary the amount of pretraining data provided to the model.

2.4 Application to human decoding of hand writing

In an impressive feat, we have successfully demonstrated that it is possible to pretrain on neural recordings from monkeys performing real hand movements, and transfer the model to decode imagined handwriting from human electrophysiological recordings (Figure 2.6). In a challenging decoding task, a human paraplegic subject imagines writing single characters—the 26 letters of the alphabet and 6 control characters [31] —while neural recordings are measured from their motor cortex. In our analysis, we show that it is possible to transfer the monkey decoder model to effectively decode the human data and drive down the error rate by an average of 12.8%. This proof-of-concept showcases the utility of our approach for transfer in important clinical applications.

2.5 Application to mouse multiregion decoding of visual stimuli and behavior

To further demonstrate the flexibility of our approach, we next applied it to the Allen Institute Neuropixels Survey (AINS), which contains high-density, simultaneous

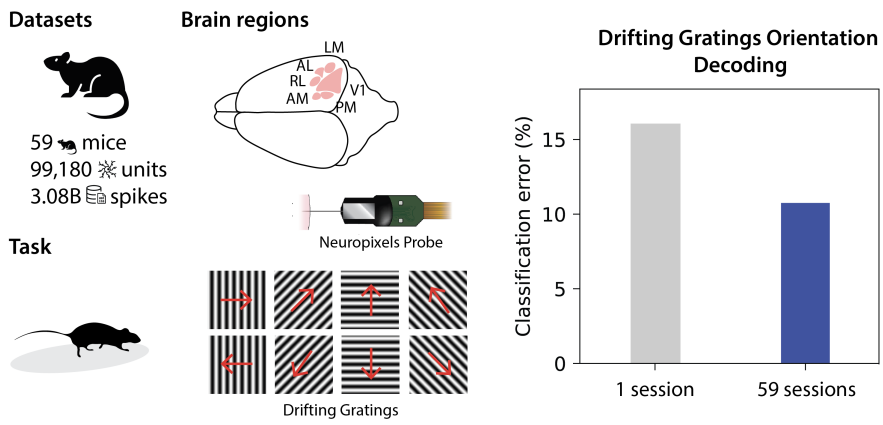


Figure 2.7: Results on datasets from the Allen Institute for Brain Science's Neuropixel Visual Survey.

electrophysiological recordings from a wide range of different visual cortical areas that extend across the visual hierarchy as well as thalamus, hippocampus, and midbrain regions [32]. This dataset consists of 3.08B spikes, 99,180 units, across 59 animals. When trained to perform a 8-class classification task to predict the orientation of drifting grating visual stimuli, we successfully demonstrate that our model works well and outperforms single session models (Figure 4.4). This result highlights the flexibility of our encoder-decoder framework (moving from a dense prediction task to a sparse sequence-level task) and its ability to work well on recordings spanning multiple brain areas.

2.6 Related Work

Transformer architectures for time-series data. Transformers have emerged as a powerful model for processing time-series data due to their ability to capture long-range dependencies [33, 34]. The key to their success lies in the self-attention mechanism [14], which allows the model to weigh the importance of each time step in the input sequence when producing an output. A typical approach in applying transformers to time-series data involves discretizing the continuous time domain into

discrete bins [34], and treating the binned data as a sequence of tokens. Each bin is then linearly projected into a high-dimensional embedding, which is input to the transformer model. The Perceiver framework [18, 13] moves away from the idea of patches, by directly processing the input bytes, using cross-attention layers to reduce the complexity of attention. In addition to regular timeseries, there has been interest in applying transformers to model irregular timeseries including event stream and point process data [35, 36, 37]. However, many of these models work on univariate event streams and when they extend to multivariate cases, they assume fixed and known input channels.

Transformers applied to neural data. With the recent advances in transformers for sequence modeling in many different time-series, transformers have also recently found successful applications in building representations of neural population activity [29, 30, 38]. In these models, the spiking activity is first binned and then the estimated bin counts are tokenized. In the neural data transformer (NDT) model [29], the firing rates of all neurons in the population are embedded jointly in one token (time step or bin). In the embedded interaction transformer (EIT) [39], neurons are considered independently in one stage of processing and at a population level in a second stage, and thus the whole dataset is tokenized over both neurons and time. In the spatiotemporal (STNDT) model [38], two different tokenizations are also considered, one in space and one in time, and two representations are learned jointly for both tokenizations. In all cases, binned data are used and the models are trained on a single session and fixed set of neurons.

In the THP, for example, the time embedding is given by a sinusoidal function of the event timestamps, which allows the model to capture periodic patterns in the data. Event types are represented by one-hot vectors, which are transformed into embeddings via a trainable embedding matrix. The time and event type embeddings are then combined to form the input to the transformer, which uses self-attention to

compute a representation of the event sequence. The output of the transformer is then used to model the 'conditional intensity function', which predicts the likelihood of future events given the past history. This approach has been shown to achieve state-of-the-art performance on a range of predictive tasks involving event data streams, demonstrating the power of transformers in this domain [37, 40, 36].

Tokenization for time-series data. In most applications of transformers to time-series data, each channel or time-series is divided into uniform bins. This scheme results in a sequence of tokens, where each token corresponds to a vector or single scalar representing the time-series data. This can be performed over a sliding and overlapping window or by creating a non-overlapping set of bins that tile the full sequence. Once the time-series is tokenized as a temporal sequence, then a wide range of position embeddings that are used in language can be employed. Recently, more advanced tokenization schemes have been developed to better capture the structure and dependencies within time-series data. TokenLearner [41] learns tokens in video data. A-VIT [42] considers adaptive tokenization for images, and multiple works now consider more efficient attention mechanisms through token pruning [43].

Representation learning and decoding for neural recordings. Representation learning from neural time-series data has become an increasingly important area of research in neuroscience [44, 29, 45, 39]. In LFADS, the aim is to predict neural activity from a sequential autoencoder [44]; AutoLFADS extends this approach [28] to add regularization to avoid overfitting and enable hyperparameter optimization. Pi-VAE proposes a semi-supervised generative VAE model for neural activity [46]. In MYOW [45] and Swap-VAE [39], they learn contrastive representations of neural population firing rates through dropout and temporal augmentations. Approaches for multi-modal contrastive learning have also been proposed [47], where behavior and neural recordings can be jointly embedded. In contrast to our work, all of these methods use fixed input representations that don't allow for multi-session training.

Multi-session training and alignment. The idea of decoding across multiple sessions has been explored in previous work [44, 48, 49, 50]. In many of these works, an initial baseline representation is formed on one day and alignment-based approaches are used to transfer a model trained on one session across recording days [51, 52, 12, 53, 54]. A subset of these methods [44, 49] can be trained on many sessions jointly, but rely on the assumption of shared dynamics or structure of a single task to achieve alignment. To the best of our knowledge, our work is the first to demonstrate multi-session transfer across subjects performing different tasks, and the first to demonstrate scaling across different data and model sizes.

2.7 Discussion

In this chapter, we introduce a novel framework for training transformers on large multi-session, multi-task neural activity datasets. To tackle the challenges of training on such large heterogeneous sources, we introduce a novel spike-level tokenization scheme and architecture that enables the model to learn from populations with varying numbers of neurons. We show that training a single unified model on multiple recordings is possible, and find that it leads to improved decoding performance. Finally, we build two large pretrained models that can be efficiently fine-tuned on new datasets, and make them available as a resource to the community.

In contrast to models trained on a single dataset, the pretrained models that we have developed provide a potential way to compare and contrast datasets, and also understand common motifs of activity and dynamics that may be shared across different sessions, tasks, and individuals. Thus, it will be critical to develop tools to probe the patterns and motifs learned by such models and characterize the neural mechanisms underlying different tasks and computations. In particular, we look to understand how spike tokens are grouped across different latent tokens and how the dynamics of the population are modeled in this latent space. Additionally, our proposed

unit embedding space allows us to map units into a high-dimensional space; thus understanding how unit projections are organized might help reveal the similarities between different neurons and the nature of their interactions. Similarly, we can analyse the session embeddings to glean insights into inter-session and across-animal differences.

Our work shows how pretraining on diverse data, including datasets from animals performing different tasks and across different laboratories, can all help to improve our ability to decode from novel and unseen neural datasets. Already, our results demonstrate the positive effect of scale for neural data analysis. However, to scale this approach further and integrate even more diverse brain regions and tasks, it will be critical to move toward a self-supervised objective. Thus, our current architecture and multi-session framework could be also extended to self-supervised tasks like generative next-event prediction or masked modeling to allow for even larger datasets to be ingested.

This framework has the potential to advance neuroscience research in several ways. By enabling the development of large pretrained models that can be fine-tuned for various downstream tasks, our work can accelerate progress in brain-machine interfaces and disease modeling applications. The ability to quickly and effectively decode neural activity from diverse datasets and modalities can have significant implications for improving our understanding of the brain and developing new therapeutic interventions.

CHAPTER 3
A SELF-SUPERVISED APPROACH FOR MULTI-TIMESCALE BEHAVIOR REPRESENTATION LEARNING

3.1 Introduction

Behavior is shaped by various factors operating across different timescales. Immediate motivations can drive moment-to-moment interactions, while long-term experiences or even the time of day can influence behavior on broader scales. Analyzing these dynamics, particularly in complex and naturalistic contexts [55, 56], has now become a critical component in many modern studies in neuroscience [56], cognitive science, and in social behavior and decision making [57, 58, 59, 60]. Additionally, monitoring and tracking systems now allow for modeling of multi-agent interactions [61, 62, 63] and social behaviors [58, 64], providing valuable insights into dynamics across many individuals.

In order to learn latent factors that may influence behavioral patterns, a promising solution is to build models of behavior in a unsupervised manner [65, 66]. Unsupervised models are of particular interest in this domain as it becomes hard to identify complex behaviors which can be composed of many "syllables" of movement [67], and are thus hard and tedious to annotate. Recent work in this direction build such representations using generative modeling and reconstruction-based objectives, typically by performing open loop [68, 69, 70] or closed loop [69] prediction of observations or actions multiple timesteps into the future.

However, when using a reconstruction or prediction objective to analyze behavior, future actions become hard to predict and models can start to become myopic, focusing only on short-range interactions in the data [67]. To circumvent this overly local

learning of dynamics, there have been a number of efforts to build models of long-term behavioral style [71, 72], where instance-level learning methods are used to extract a single representation for an entire sequence. However, these models then lose their ability to provide time-varying representations that capture the dynamic nature of different behaviors. It is still an outstanding challenge to build representations that can capture both short-term behavioral dynamics along with longer-term trends and global structure.

In this chapter, we develop a new self-supervised approach for learning multiscale representations of behavior. Our method consists of two core innovations: (i) a novel action-prediction approach that aims to predict the *distribution of actions* over future timesteps, without modeling exactly when each action is taken, and (ii) a novel multi-scale architecture that builds *separate latent spaces to accommodate short- and long-term dynamics*. We combine both of these innovations and show that our approach can capture both long-term and short-term attributes of behavior and work flexibly to solve a variety of different downstream tasks.

To test our approach, we utilize behavioral datasets that contain multiple tasks that vary in complexity and contain distinct multi-timescale dynamics. Using NVIDIA's Isaac Gym [73], we generate a synthetic dataset of the multi-limb kinematics from quadruped robots by varying the robot's morphological properties and the environment's terrain type and difficulty. Using this robot behavior data, we demonstrate that our method can effectively build dynamical models of behavior that accurately elicit both the robot and environment properties, without any explicit training signal encouraging the learning of this information.

Having established that our approach can successfully predict the complex behavior of an artificial creature, we apply it to two multi-agent behavior benchmarks [74] and challenges with multiple tasks that vary in their frame-level (local) vs. sequence-level (global) labels and properties. On the mouse triplet benchmark, we rank first overall

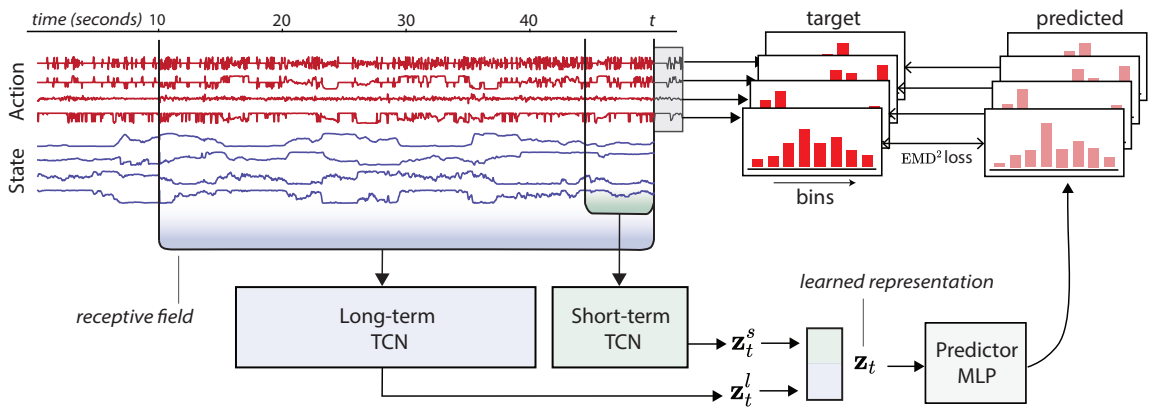


Figure 3.1: *Architecture of Bootstrap Across Multiple Scales (BAMS)*. BAMS uses two temporal convolutional networks with two latent spaces, each with their own receptive field sizes. The model is trained on a novel learning objective that consists in predicting future action distributions instead of future action sequences. In this figure, we use sample data from the MABe dataset, only a subset of the channels are shown.

on the leaderboard [1] (averaged across 13 tasks), first on all of the 4 global tasks, and are in the top-3 on all the 9 frame-level local subtasks. In one of the global tasks (decoding the strain of the mouse), we achieve impressive performance over the other methods, with a 10% gap over the next best performing method. On the fruit fly groups benchmark, we also rank first overall [2] (averaged across 50 tasks), and outperform other methods on both average frame-level and sequence-level subtasks. Our results demonstrate that our approach can provide representations that can be used to decode meaningful information from behavior that spans many timescales (longer approach interactions, grooming, etc.) as well as global attributes like the time of day or the strain of the mouse.

[1] aicrowd.com/challenges/multi-agent-behavior-challenge-2022/problems/mabe-2022-mouse-triplets/leaderboards?post_challenge=true

[2] aicrowd.com/challenges/multi-agent-behavior-challenge-2022/problems/mabe-2022-fruit-fly-groups/leaderboards?post_challenge=true

3.2 Method

Our approach addresses two critical challenges of modeling naturalistic behavior (Figure 4.1). In subsection 3.2.2, we introduce a novel distributional-relaxation of the reconstruction-based learning objective. In subsection 3.2.3, we introduce an architecture and self-supervised learning objectives that support the learning of behavior at different levels of temporal granularity.

3.2.1 Problem setup

We assume a fixed dataset of D trajectories, each comprised of a sequence of observations $\mathbf{x}_t$ and/or actions $\mathbf{y}_t$. Where actions are not explicitly provided, in many cases we can infer actions based on the difference between consecutive observations. Our goal is to learn, for each timestep, behavioral representations $\mathbf{z}_t$ that capture both global-information such as the strain of the mouse or the time of day, as well as temporally-localised representations such as the activity each mouse is engaged in at a given point in time. As obtaining labeled datasets for realistically-useful scales of agent population and diversity of behavior is impractical, we aim to learn representations in an unsupervised manner.

3.2.2 Histogram of Actions (`HoA`): A novel objective for predicting future

Modeling behavior dynamics can be done by training a model to predict future actions. This reconstruction-based objective becomes challenging when behavior is complex and non-stereotyped. Let us consider the example of a mouse scanning the room, rotating its head from one side to the other. It is possible to extrapolate the trajectory of the head over a few milliseconds, but prediction quickly becomes impossible, not because this particular behavior is complex but because any temporal misalignment in the prediction leads to increasing errors.

We propose to predict the distribution of future actions rather than their sequence. The motivation behind this distributional-relaxation of the reconstruction objective lies in *blurring the exact temporal unfolding of the actions* while preserving their behavioral fingerprint.

Predicting histograms of future actions. Let $\mathbf{y}_t \in \mathbb{R}^N$ be the action vector at time t. Each feature in the action vector can, for example, represent the linear velocity of a joint or the angular velocity of the head. Given observations $[\mathbf{x}_0, \ldots, \mathbf{x}_t]$ of the behavior at timesteps 0 through t, the objective is to predict the distribution of future actions over the next L timesteps. For each i-th element of the action vector, we compute a one-dimensional normalized histogram of the values it takes between timesteps $t+1$ and $t+L$. We pre-partition the space of action values into K equally spaced bins, resulting in a K-dimensional histogram that we denotes as $\mathbf{h}_{t,i}$, for all keypoints $1 \leq i \leq N$.

We introduce a predictor g that, given the extracted representation $\mathbf{z}_t$, predicts all feature-wise histograms of future actions. The predictor is a multi-layer perceptron (MLP) with an output space in $\mathbb{R}^{N \times K}$. The output is split into N vectors, which are normalized using the softmax operator. We obtain $[\hat{\mathbf{h}}_{t,1}, \ldots, \hat{\mathbf{h}}_{t,n}]$, each estimating the histogram of the i-th action feature following timestep t.

EMD^2 loss for histograms. To measure the loss between the predicted and target histograms, we use the Discrete Wasserstein distance, also known as the Earth Mover's Distance (EMD). This distance is obtained by solving an optimal transport problem that consists in moving mass from one distribution to the other while incurring the lowest transport cost. In our case, the cost of moving mass from one bin to another is equal to the number of steps between the two bins.

Because our histogram has equally-sized bins, the EMD is equivalent to the Mallows distance which has a closed-form solution [75, 76]. In particular we use EMD^2 which

has been shown to be easier to optimize and converge faster [77]. The loss is defined as follows:

$$\mathcal{D}_{\mathrm{EMD}^2}(\mathbf{h}_{t,i}, \hat{\mathbf{h}}_{t,i}) = \sum_{k=1}^{K} (\mathrm{CDF}_k(\mathbf{h}_{t,i}) - \mathrm{CDF}_k(\widehat{\mathbf{h}}_{t,i}))^2, \tag{3.1}$$

where $\mathrm{CDF}_k(\mathbf{h})$ is the k-th element of the cumulative distribution function of $\mathbf{h}$.

The total loss is obtained by summing over all features of the action vector, which leaves us with the following loss at time t:

$$\mathcal{L}_t = \sum_{i=1}^{N} \mathcal{D}_{\mathrm{EMD}^2}(\mathbf{h}_{t,i}, \widehat{\mathbf{h}}_{t,i}). \tag{3.2}$$

3.2.3 Multi-timescale bootstrapping in a temporally-diverse architecture

In order to form richer and multi-scale representations of behavior, we use a self-supervised learning objective. We introduce a new approach using latent predictive losses to build representations across different scales while preserving the granularity in each. We achieve this by explicitly separating the short-term and long-term representations, then bootstrapping within each representation space. This approach enables us to learn from otherwise incompatible representation learning objectives [78, 79].

Our goal is to capture and separate short-term and long-term dynamics in two different spaces. We use the Temporal Convolutional Network (TCN) [80] as our building block. The TCN produces a representation at time t that only depends on the past observations [81].

We design an architecture that separates the different timescales by using two TCN encoders: A **short-term encoder**, f_s, that captures short-term dynamics and targets momentary behaviors such as drinking, running or chasing; A **long-term encoder**, f_l, that captures long-term dynamics and targets longstanding factors that modulate

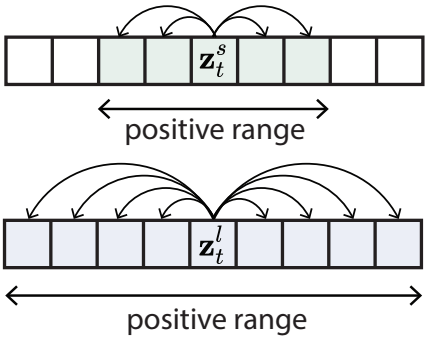


Figure 3.2: *Visualization of the short-term and long-term windows used to build multi-scale similarity in BAMS.* Positives are selected within a window. The window is small for short-term embeddings and can be as large as the entire sequence for long-term embeddings.

behavior (strain of mouse, time of day). Architecturally, the difference between the two is that we increase the number of layers and use larger dilation rates [82] for the long-term encoder, thus effectively covering a larger receptive field (more history) in the input sequence. All feature embeddings extracted by the TCNs are concatenated, to produce the final embedding, $\mathbf{z}_t = \mathbf{concat}[\mathbf{z}_t^s, \mathbf{z}_t^l]$.

We draw inspiration from recent work [83, 84, 85] that uses latent bootstrapping to learn a latent space where "positive" views are mapped close to each other. Unlike other contrastive methods [86], bootstrapping does not require negative examples.

In the context of temporal representation learning, a common assumption is that points that are nearby in time are positive views of each other and can be constrained to lie nearby in the latent space [86, 87]. In our case, we can bootstrap and find positive views at both the short-term and also at a more long-term scale, as illustrated in Figure 3.2.

Bootstrapping short-term representations. We randomly select samples, both future or past, that are within a small window Δ of the current timestep t. In other words, $\delta \in [-\Delta, \Delta]$. Bootstrapping involves using a shallow network to predict the representation of one view from the other [83]. We use a predictor q_s that takes in the

short-term embedding $\mathbf{z}_t^s$ and learns to regress $\mathbf{z}_{t+\delta}^s$ using the loss:

$$\mathcal{L}_{r,short} = \left\| \frac{q_s(\mathbf{z}_t^s)}{\|q_s(\mathbf{z}_t^s)\|_2} - \text{sg}\left[\frac{\mathbf{z}_{t+\delta}^s}{\|\mathbf{z}_{t+\delta}^s\|_2} \right] \right\|_2^2, \tag{3.3}$$

where sg[·] denotes the stop gradient operator. Unlike [83], we do not use an exponential moving average of the model, but simply increase the learning rate of the predictor as in [84].

Bootstrapping long-term representations. For long-term behavior embeddings which should be stable at the level of a sequence, we sample any other time point in the same sequence, i.e. $t' \in [0, T]$. We use a similar setup for the long-term behavior embedding, where predictor q_l is trained over longer time periods or in the limit, over the entire sequence.

$$\mathcal{L}_{r,long} = \left\| \frac{q_l(\mathbf{z}_t^l)}{\|q_l(\mathbf{z}_t^l)\|_2} - \text{sg}\left[\frac{\mathbf{z}_{t'}^l}{\|\mathbf{z}_{t'}^l\|_2} \right] \right\|_2^2 \tag{3.4}$$

3.2.4 Putting it all together

Finally, we optimize the proposed multi-task architecture with a combined loss:

$$\mathcal{L} = \mathcal{L}_t + \alpha(\mathcal{L}_{r,short} + \mathcal{L}_{r,long}) \tag{3.5}$$

where α is a scalar that is used to weigh the contribution of the short- and long-term contrastive losses. In practice, we find that we simply need to choose α that re-scales the contrastive losses to the same order of magnitude as the `HoA` prediction loss.

3.3 Experiments

3.3.1 Simulated Quadrupeds Experiment

A synthetic dataset of simulated legged robots

To test our model's ability to separate behavioral factors that vary in complexity and contain distinct multi-timescale dynamics, we introduce a new dataset generated from a heterogeneous population of quadrupeds traversing different terrains. Simulation enables access to information that is generally inaccessible or hard to acquire in a real-world setting and provides ground-truth information about the agent and the world state.

Agents. We use advanced robotic systems [88] that imitates 4-legged creatures capable of various locomotion skills. These robots are trained to walk on challenging terrains using reinforcement learning [89]. We use two robots that differ by their morphology, ANYmal B and ANYmal C. To create heterogeneity in the population, we randomize the body mass of the robot as well as the target traversal velocity. We track a set of 24 proprioceptive features including linear and angular velocities of the

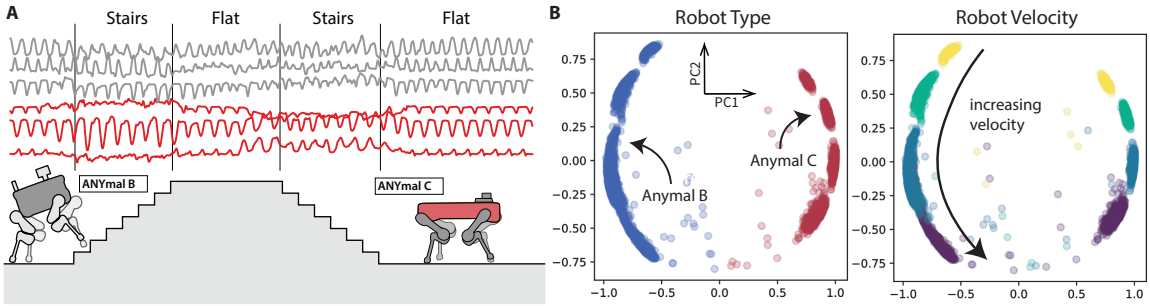


Figure 3.3: *Quadrupeds walking on procedurally generated map.* (A) Illustration showing the two robots (ANYmal B and ANYmal C) walking on the procedurally generated map, with segments of different terrain types. As the robots traverse different terrains, the velocity of their joints is tracked and visualized. (B) Long-term embeddings learned by BAMS. On the left, we overlay the labels for the type of robot and on the right we overlay the velocity of the robot; we observe a clear organization of the latents in terms of their velocity and robot type.

Table 3.1: *Linear readouts of robot behavior.* For each task, we report the linear decoding performance on sequence-level and frame-level tasks. Tasks marked with * are classification tasks for which the F1-score is reported, for the remaining tasks, we report the mean-squared error.

	Sequence-level Tasks		*Frame-level Tasks*		
Model	Robot Type* (↑)	Linear velocity (↓)	Terrain type* (↑)	Terrain slope (↓)	Terrain difficulty (↓)
PCA	99.72	0.069	08.83	0.037	0.790
TCN	99.93	0.102	33.03	0.037	0.080
BAMS	99.96	0.038	**39.89**	**0.033**	**0.078**
↳ short-term	**100.0**	0.094	34.86	0.036	0.079
↳ long-term	99.88	**0.020**	32.39	0.036	**0.078**

robots' joints.

Procedurally generated environments. Using NVIDIA's Isaac Gym [88] simulation environment, we procedurally generate maps composed of multiple segments of different terrains types (Figure 3.3). We consider five different terrains including flat surfaces, pits, hills, ascending and descending stairs. We also vary the roughness and slope of the terrain to control the difficulty of terrain traversal.

Experimental setup. We collect 5182 trajectories of robots walking through terrains. We record for 3 minutes at a frequency of 50Hz. For evaluation, we split the dataset into train and test sets (80/20 split) and use multi-task probes that correspond to different long-term and short-term behavioral factors. More details can be found in subsection 6.2.1.

Results

Results in Table 3.1 suggest that our model performs well on these diverse prediction tasks. A major advantage of our method is the separation of the short-term and long-term dynamics, which enables us to more clearly identify the multi-scale factors. While some of the tasks are represented best in the mixed model, we find that the linear velocity is more decodable in the long-term embedding, while terrain type is more decodable in the short-term embedding. We further analyze the formed

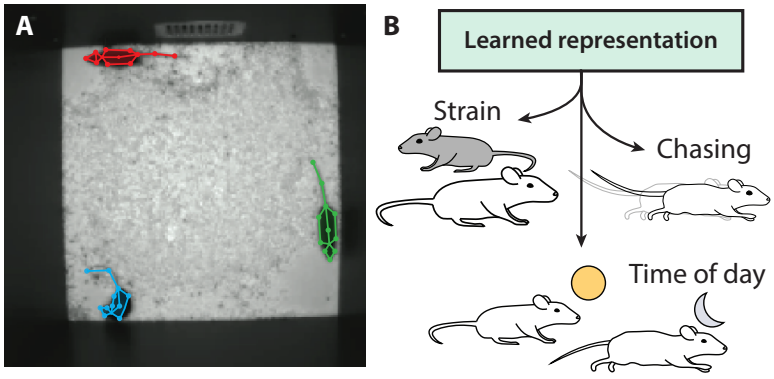


Figure 3.4: *Multi-Agent Behavior (MABe) - Mouse Triplets Challenge.* (A) Keypoint tracking approaches are used to extract keypoints from many positions on the mouse body in a video. (B) Methods are evaluated across 13 different tasks.

representations by visualizing the embeddings in the different spaces. In Figure 3.3-B, we visualize the long-term embedding space and find that our model is able to capture the main factors of variance in the dataset, corresponding to the robot type and the velocity at which the robots are moving. This suggest that in the absence of labels, the learned embedding can provide valuable insights into how the recorded population is distributed without the need for annotations. In subsection 6.2.1, we visualize the extracted embeddings of a single sequence over time. We find that the long-term embedding is more stable and smooth, while the short term embeddings reveal different blocks of behavior that change more frequently.

3.3.2 Experiments on Mouse Triplet Dataset

Dataset description. The mouse triplet dataset [74] is part of the Multi-Agent Behavior Challenge (MABe 2022), hosted at CVPR 2022. This large-scale dataset was introduced to address the lack of standardized benchmarks for representation learning of animal behavior. It consists of a set of trajectories from three mice interacting in an open-field arena. A total of 5336 one-minute clips, recorded from a top-view camera, were curated and processed to track twelve anatomically defined keypoints on each mouse, as shown in Figure 3.4.

Table 3.2: *Linear readouts of mouse behavior.* We report the BAMS against the top performing models in the MABe 2022 challenge. All numbers are reported in [90] except for TS2Vec and T-BYOL, which we produce. The scores show the performance of the linear readouts across 13 different tasks. Mean-squared error (MSE) is used in the case of tasks 1 and 2, since they are continuously labeled. In the rest of the subtasks, which are binary (yes/no), F1-scores are used. The best-performing models are those with low MSE scores and high F1-scores, are highlighted in bold.

	Sequence-level subtasks				*Frame-level subtasks*								
Model	Day (↓)	Time (↓)	Strain	Lights	Approach	Chase	Close	Contact	Huddle	O/E	O/G	O/O	Watching
PCA	0.09416	0.09445	51.60	54.65	0.86	0.14	49.27	37.87	12.71	0.21	0.60	0.53	6.65
TVAE	0.09403	0.09442	52.98	56.80	1.07	0.45	59.33	44.77	21.96	0.27	0.83	0.62	10.20
T-BYOL	0.09362	0.09373	60.95	65.05	1.68	0.72	62.48	48.19	18.52	0.35	0.96	0.82	17.77
T-BERT	0.09262	0.09276	78.63	68.84	1.80	0.87	70.22	55.84	30.24	0.51	1.40	1.12	17.27
TS2Vec	0.09380	0.09422	57.12	65.60	1.29	0.66	59.53	46.13	24.74	0.35	1.09	0.74	12.37
T-Perceiver	0.09322	0.09323	69.81	69.68	1.57	1.27	60.84	47.81	28.32	0.41	1.16	0.86	16.42
T-GPT	0.09269	0.09384	64.45	65.39	1.73	0.64	69.05	55.78	23.80	0.46	1.12	1.05	17.86
T-PointNet	0.09275	0.09320	66.01	67.15	2.56	**4.57**	**70.68**	**55.96**	21.23	**0.84**	**2.79**	**2.32**	15.61
BAMS	**0.09094**	**0.08989**	**88.23**	**72.00**	**2.74**	1.89	67.22	53.43	**31.43**	0.59	1.61	1.57	**18.15**
↳ short-term	0.09288	0.09294	61.33	66.34	1.80	1.15	66.58	52.60	25.34	0.39	1.09	0.98	16.74
↳ long-term	0.09174	0.09037	86.49	70.91	2.10	1.06	61.99	49.12	29.09	0.45	1.32	1.08	13.98

As part of this benchmark, a set of 13 common behavior analysis tasks were identified, and are used to evaluate the performance of representation learning methods. Over the course of these sequences, the mice might exhibit individual and social behaviors. Some might unfold at the frame level, like chasing or being chased, others at the sequence level, like light cycles affecting the behavior of the mice or mouse strains that inherently differentiate behavior.

Integrating features across multiple animals. We process the trajectory data to extract 36 features characterizing each mouse individually, including head orientation, body velocity and joint angles. We construct the short-term TCN and long-term TCN encoders to have representation of size 32 each, and receptive fields approximated to be 60 and 1200 frames respectively. We compute the histogram of actions over 1 second, i.e. $L = 30$ and use $K = 32$ bins. We build representations for each mouse independently, which means that at time t, and for each frame t, we produce embeddings $\mathbf{z}_{t,1}$, $\mathbf{z}_{t,2}$ and $\mathbf{z}_{t,3}$, for each mouse respectively and concatentate the embeddings for all three individuals to build a joint embedding for evaluation. The model is trained for 500 epochs using the Adam optimizer with a learning rate of 10^{-3}. More details can be

found in subsection 6.2.2.

Interaction loss. To learn additional features that are useful in predicting animal interactions for multi-agent settings, we introduce a simple auxiliary loss to predict the distances between the trio at time t. Our input features do not include any information about the global position of the mice in the arena, so the model can only rely on the inherent behavior and movement of each individual mouse to draw conclusions about their proximity. Thus, we build a network h that takes in the embeddings of two mice i and j and predicts the distance $d_{i,j}$ between them as follows,

$$\mathcal{L}_{\text{aux}} = \|h(\mathbf{z}_{t,i}, \mathbf{z}_{t,j}) - \mathrm{d}_{i,j}\|_2^2. \tag{3.6}$$

This penalty is added to the loss in Equation Equation 4.1 to encourage learning of shared features across the different individual embeddings.

Evaluation protocol. To evaluate the performance of the model in detecting frame- and sequence-level behaviors, we compute representations $\mathbf{z}_{t,1}$, $\mathbf{z}_{t,2}$ and $\mathbf{z}_{t,3}$ for each mouse respectively, which we then aggregate into a single mouse triplet embedding using two different pooling strategies. First, we apply average pooling to get $\mathbf{z}_{t,\text{avg}}$. Second, we apply max pooling and min pooling, then compute the difference to get $\mathbf{z}_{t,\text{minmax}} = \mathbf{z}_{t,\text{max}} - \mathbf{z}_{t,\text{min}}$. Both aggregated embeddings are concatenated into a 128-dim embedding for each frame in the sequence. Evaluation of each one of the 13 tasks, is performed by training a linear layer on top of the frozen representations, producing a final F1 score or a mean squared error depending on the task.

3.3.3 Results

We compare our model against PCA, trajectory VAE (TVAE) [69], TS2Vec [86], BYOL [83], and the top performing models in the MABe2022 challenge [74] which are adapted respectively from Perceiver [18], GPT [15], PointNet [91] and BERT [92].

All methods are trained using some form of reconstruction-based objective, with both T-Perceiver and T-BERT using masked modeling. Both T-BERT and T-PointNet also supplement their training with a contrastive learning objective using positives from the same sequence. It is also important to note that the training set labels from two tasks (Lights and Chase) are made publicly available, and are used as additional supervision in the T-Perceiver and T-BERT models. We do not use any supervision for BAMS.

Our model achieves a new state-of-the-art result on the MABe Multi-Agent Behavior 2022 - Mouse Triplets Challenge, as can be seen in Table 3.2. We rank first overall based upon our performance on all tasks, and show impressive boosts in performance on global sequence-level tasks where we find a 22% improvement in the Strain task and 5% improvement in the Light task. In the frame-level tasks, we remain competitive with other approaches, and rank 1st on 3 out of 9 of the frame-level subtasks; this is in contrast to the other top performing model that explicitly models the interaction between mice by introducing hand-crafted pairwise features. This is outside the scope of this work, as we do not focus on social interactions beyond predicting the distance between mice.

We observe that our proposed method results in significant gaps on sequence-level tasks. In particular, we observe a marked improvement on the prediction of strain over the second-place model at 78.63%, with BAMS yielding a 10% improvement in accuracy at 88.23%. These big improvements suggest that we might have identified and addressed a critical problem in behavior modeling. In the next section, we conduct a series of ablations to further dissect our model's performance.

Multi-timescale embedding separation also enables us to probe our model for timescale-specific features. In Table 3.2, we report the decoding performance with short-term embeddings and long-term embeddings respectively. We find that sequence-level behavioral factors are better revealed in the long-term space, while the frame-

Table 3.3: *Ablations on the Mouse Triplet Dataset.* We report the average sequence-level MSE and F1-score, and the average frame-level F1-score.

Hist. of actions	Bootstrapping	Multi-timescale	Seq MSE	Seq F1	Frame F1
✓	✓	✓	**0.090415**	**80.12**	**19.85**
	✓	✓	0.093100	68.30	19.46
✓		✓	0.090717	78.11	19.45
✓	✓		0.092483	73.37	19.52

level factors are more distinct in the short-term space. That being said, decoding performance is still best when using both timescales.

By default, BAMS is pre-trained with all available trajectory data. We also test BAMS in the inductive setting, where we only pre-train using the training split (only 1800 out of 5336) of the dataset. Results are reported in subsection 6.2.2. We find that the performance drop is modest, and that BAMS trained in this setting still beats all other methods.

3.3.4 Ablations

To understand the role of the different proposed components in enabling us to achieve state-of-the-art performance, we conduct a series of ablations on the MABe benchmark that we report in Table 3.3. First, we compare the performance of BAMS when trained with the traditional reconstruction-based objective (multi-step sequential prediction). BAMS without the `HoA` prediction objective, performs comparably with many of the top-entries, though performance is ultimately improved when using this novel learning objective. We note that we had to reduce the number of prediction frames to 10, as the training fails if we go beyond. With our `HoA` objective, we can use 30 frames, which strongly suggests that this loss can be stably applied over longer time horizons when compared to other losses.

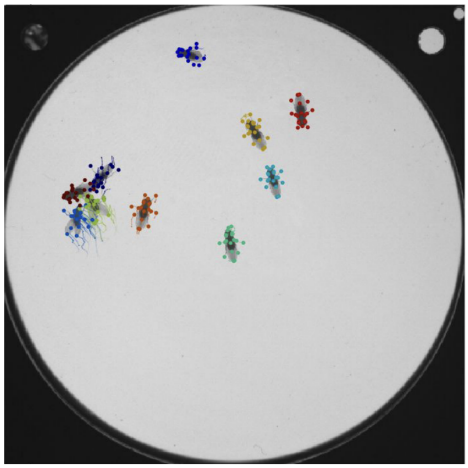


Figure 3.5: *Sample frame from the Fruit Fly Groups Dataset.*

Next, we analyze the role of the multi-timescale bootstrapping in improving the quality of the representation. When removing the bootstrapping objective, we find a 2% drop in the sequence-level averaged F1-score, as well as modest drops in frame-level performance. This suggests that this learning objective may help to resolve global and intermediate-scale features. We also ablate the multi-timescale component, i.e. we perform bootstrapping but in the same space, by using a single TCN that spans the receptive field of the two used originally. This results in sub-optimal performance, which emphasizes the idea that having different objectives in different spaces is critical to prevent interference and provide ideal performance. We report additional ablations in subsection 6.2.2, including an ablation of the interaction module where we find that it is not critical for good performance.

3.3.5 Experiments on Fruit Fly Groups Dataset

Experimental Setup and Tasks

The fruit fly groups dataset is the second dataset in the MABe benchmark. It consists of tracking data of a group of 9 to 11 flies interacting in a small dish (Figure 3.5). Tracking data consists of 19 keypoints on each fly body and wings, and is recorded at a much higher frame rate compared to the mice (150 vs. 30 fps). Precise neural

Table 3.4: *Linear readouts of fly behavior.* We report the average performance of various models on both frame-level, and sequence-level subtasks.

Model	All F1	Seq F1	Frame F1
PCA	42.5	23.0	45.2
TVAE	37.0	22.2	39.0
T-Perceiver	44.8	19.7	48.2
T-GPT	45.8	24.5	48.7
BAMS	**48.2**	**25.4**	**51.1**

activity manipulations are performed on certain neurons which, when activated, induce certain types of behavior including courtship, avoidance and female aggression [74]. Additionally, the groups of flies are differentiated by various genetic mutations and tagged by sex. This along with other behavioral factors provide us with 50 different subtasks, both frame-level and sequence-level, that can be use to evaluate the learned representations.

In this set of experiments, we are interested in testing whether our proposed method generalizes to a dataset from another organism, and whether it can work out-of-the-box with minimal expert intervention. In particular we chose to not extract any features manually and only use the provided data, we use the default hyperparameters found in the previous experiment and do not perform hyperparameter tuning, and finally we do not use an interaction module. Note that we follow the same evaluation procedure established previously.

Results

We compare our model against PCA, TVAE, TS2Vec, and the top performing models in the fly challenge: T-GPT, and T-Perceiver. Our model achieves state-of-the-art performance on the fly dataset, as seen in Table 3.4. BAMS outperforms other models on both frame-level and sequence-level sub-tasks, and we note a significant boost in the average frame-level F1 score. This result further demonstrates the generalizability

of our approach to new datasets and scaling to an even larger numbers of animals.

3.4 Related Work

3.4.1 Animal behavior analysis

Pose estimation and animal tracking. Recently, there has been a recent democratization of automated methods for pose estimation and animal tracking that has made it possible to conduct large scale behavioral studies in many scientific domains. These tools abstract behavior trajectories from video recordings, and facilitate the modeling of behavioral dynamics. Most pipelines for analyzing animal behavior consist of three key steps [93]: (1) pose estimation [94, 95, 96, 97, 98], (2) spatial-temporal feature extraction [99, 70], and (3) quantification and phenotyping of behavior [100, 101, 102]. In our work, we consider the analysis of behavior after pose estimation is performed. However, one could imagine using our multi-timescale bootstrapping approach for representation learning in video analysis, where self-supervised losses have been proposed recently for keypoint discovery [94, 103, 104].

Disentanglement of animal behavior in videos. In recent work [105, 106, 107], two distinct disentangled behaviorial embeddings are learned from video, separating non-behavioral features (context, recording condition, etc.) from the dynamic behavioral factors (pose). This has even been applied to situations with multiple individuals, performing disentanglement on each individual [108]. This is performed by training two encoders, typically variational autoencoders (VAEs) [109], on either multi-view dynamic information or a single image. These encoders create explicitly separated embeddings of behavioral and context features. In comparison to this work, rather than seperating behavior from context, our model considers the explicit separation of behavioral embeddings across multiple timescales, and considers the construction of a global embedding that is consistent over long timescales.

Modeling social behavior. For social and multi-animal datasets, there are a number of other challenges that arise. Simba [110] and MARS [58] have similar overall workflows for detecting keypoints and pose of many animals and classifying social behaviors. More recently, a semi-supervised approach TREBA has been introduced [90] for building behavior embeddings using task programming. TREBA is built on top of the trajectory VAE [69], a variational generative model for learning representations of physical trajectories in space. In our work, we do not consider a reconstruction objective but a future prediction objective, in addition to bootstrapping the behavior representations at different timescales.

3.4.2 Representation learning for sequential data

The self-supervised learning (SSL) framework has gained a lot of popularity recently due to its impressive performance in many domains [92, 111, 112]. Many SSL methods are built based on the concept of *instance-specific* alignment loss: Different views of each datapoint are created based on pre-selected augmentations, and the views that are produced from the same datapoint are treated as positive examples; while the views that are produced from different datapoints are treated as negative examples. While contrastive methods like SimCLR [111] utilize both positive examples and negative examples to guide the learning, BYOL [83] proposes a framework in which augmentations of a sample are brought closer together in the representation space through a predictive regression loss. Recent work [85, 84, 113, 114] applies BYOL to learn representations of sequential data. In such cases, neighboring samples in time are considered to be positive examples of each other, assuming temporal smoothness of the semantics underlying the sequence. The model is trained such that neighboring samples in time are mapped close to each other in the representation space. However, these methods use a single scale to define similarities unlike our method.

Recently, self-supervised methods such as TS2vec [86] learn self-supervised repre-

sentations for sequential data by generating positive sub-sequence views through more complex temporal augmentations that can be integrated at instance or local level. Positive sub-sequences are temporally contrasted with other representations within the same sequence as well as contrasted with representations of other available sequences. However, unlike our method, the same space of representations are used for learning these multi-scale representations. We learn a separate space of representations for different time scales to encourage disentanglement.

The idea of using multi-scale feature extractors can be found in representation learning. In [84], a video representation learning framework, two different encoders process a narrow view and a broad view respectively. The narrow view corresponds to a video clip of a few seconds, while the broad view spans a larger timescale. The objective, however, is different to ours, as the narrow and broad representations are brought closer to each other, in the goal of encoding their mutual information. This strategy is also used in graph representation learning where a local-neighborhood of node is compared to its global neighborhood [115, 116]. Our method differs in that we bootstrap the embeddings at different timescales separately, this is important to maintain the fine granularity specific to each timescale, thus revealing richer information about behavioral dynamics.

3.5 Discussion

Behavior is likely to be driven by a number of factors that can unfold over different timescales. Thus, having ways to model behavior and discover differences in behavioral repertoires or actions at many scales could provide insights into individual differences, and help, for example, detect signatures of cognitive impairment [102]. We make steps towards addressing these needs by proposing BAMS, a novel self-supervised approach that learns representations for behavioral data at different timescales.

Our analysis on realworld datasets centers on two different datasets in a multi-agent

benchmark, both with very different dimensionalities and variable tasks. Despite this, our approach is designed to model a wide range of behaviors, whether it's individual, social, structured or naturalistic. More multi-scale and multi-task benchmarks like MABe are needed in order properly evaluate and guide the development of tools for general behavior analysis.

Currently, our model has a relatively simple way of (optionally) modeling interactions between animals in the multi-agent setting. Despite this, we show really competitive performance in multi-agent analysis without using handcrafted interaction features like T-PointNet or T-BERT [74]. Moving forward, it would be interesting to develop new ways to learn features of multi-animal interactions, especially in open environments where animals might come in and out of frame or get occluded.

Our experiments highlight the truly multi-scale nature of BAMS and show that our method can learn to distinguish global as well as temporally local behaviors. Currently, we seperate short-term from long-term dynamics through a contrastive loss and separation of the information into two latent spaces for each scale. Although this approach appears to be effective, we don't modify the loss explicitly to disentangle the two latent spaces. By providing additional incentives for the model to separate short from long-term dynamics we hope to improve the interpretability of the model.

When extending our model from mice to flies, we found that it was possible to use the same overall model architecture and hyperparameters, despite the major differences in datasets and underlying tasks. Thus, given the robustness of the method, we imagine that it can be utilized in the analysis of other species and even more diverse types of behavior [117, 118]. By combining our modeling approach with methods for self-supervised video keypoint discovery [119], we could further extend BAMS to raw video data without needing the intermediate step of pose estimation. We also look forward to the opportunities provided by simultaneous recordings of the brain and behavior [120, 121] and other multi-modal sources of input that could be leveraged to

further study how behavior unfolds across different timescales.

CHAPTER 4
LEARNING INVARIANCES IN NEURAL POPULATION ACTIVITY

4.1 Introduction

Self-supervised learning (SSL) methods have made impressive advances on a wide range of tasks in vision [122, 123, 124, 111, 83, 125, 126], speech [127], graphs [128, 129], and reinforcement learning [127, 85, 130]. This has been due, in part, to the simple paradigm of instance-level learning, where a representation is learned by maximizing the similarity between different transformed "views" of the same sample (positive examples). Contrastive learning methods compare positive examples to views of other samples (negative examples) and encourage them to have dissimilar representations [124, 111, 127, 131], while more recent methods like `BYOL` [83], `W-MSE` [132], and `BarlowTwins` [133] show how this instance-specific approach can be implemented without the need for negative examples.

Augmentations are a key component of self-supervised methods; they establish the invariances learned by the network and control the richness of the learned representation. Thus, there are many cases where it is useful to *go beyond simple augmentations* to integrate more diverse views into learning [134, 135]. At the same time, it can be challenging to find the right balance between augmentations that both introduce sufficient diversity and preserve the semantics of the original data. This is particularly true in new domains, like brain decoding, where we do not have prior knowledge to guide our search. Here, we ask whether diverse views can be found by *looking within the dataset*. Intuitively, other examples drawn from the dataset have the potential to satisfy both criteria: They offer more diversity, and when chosen correctly, they will share semantic content with the target sample.

In this paper, we introduce **M**ine **Y**our **O**wn vie**W** (`MYOW`), a self-supervised approach for representation learning that looks within the dataset for different samples to use as positive examples for one another. The idea behind our strategy is to *mine views*, or adaptively select other samples that are nearby in the latent space, and then use these mined views as targets for self-supervision. To integrate both mined and augmented views into a unified framework, we introduce a novel, cascaded dual projector architecture that learns to predict across augmented views of the *same sample* in the first part of the network, and then to predict across mined views of *different samples* through a separate projector/predictor that draws from the first projector's output (Figure 4.1).

To first test the method in domains where effective augmentations are well-established, we apply our approach to computer vision benchmarks, including CIFAR-10, CIFAR-100 and Tiny Imagenet. On these benchmark datasets, we show that `MYOW` is competitive with state-of-the-art methods like `BYOL` [83] and `SimCLR` [111] and in many cases, outperforms these methods. After validating our approach in the image domain, we then turn our attention to *brain decoding from multi-neuron recordings*, a novel application of SSL where diverse augmentations are unknown. We apply our approach to neural activities from the brains of non-human primates and rodents, where we show significant improvements over other approaches on two distinct brain decoding tasks (i.e., movement prediction from motor cortex, and sleep stage prediction from V1 and hippocampus). These results suggest that nearest-neighbor prediction can be a very effective tool for self-supervision in new domains where strong augmentations are not already established.

Overall, we make the following contributions:

- In section 4.2, we introduce `MYOW`, a new approach for adaptively finding views from distinct samples in the dataset and using them as positive examples for one another. We introduce a novel cascaded dual projector architecture that

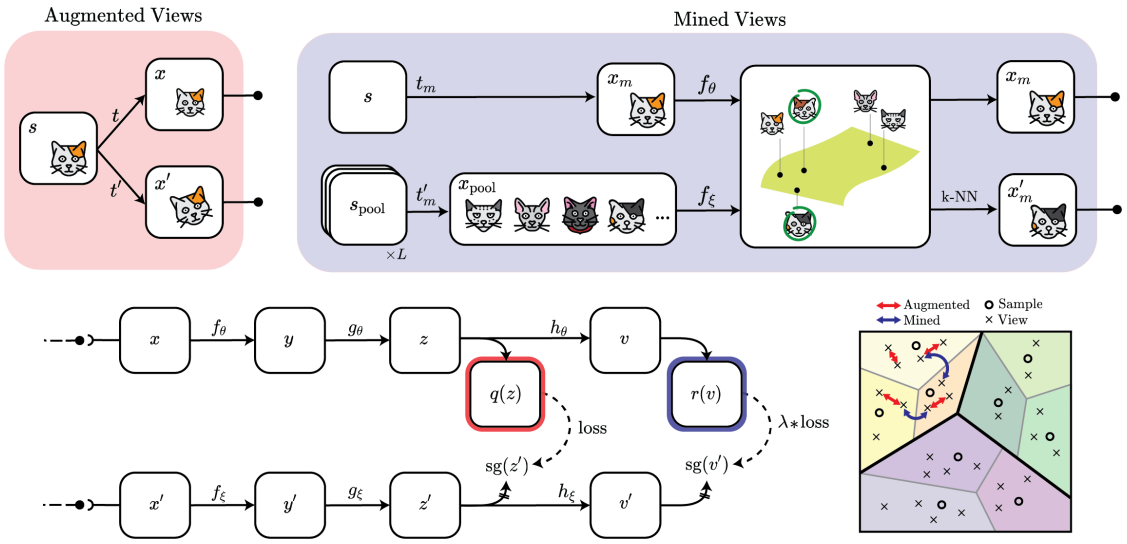


Figure 4.1: *Overview of our approach.* The architecture of the system, shown in the bottom row, consists of two networks, the online network (top) and the target network (below). There exists two sources of views, the augmented views block (top, red) and the mined views block (top, blue). Each type of view is handled by a dedicated predictor of the corresponding color. During mining, mined views are found by computing the k-nearest neighbors of the anchor online representation among the target representations of the pool of candidates. One of the nearest neighbors is randomly selected to be the mined view. On the bottom right, we illustrate the idea behind across-sample prediction and show how the two spaces emphasize different levels of similarity between data points.

builds on `BYOL` to integrate augmented and mined views without the need for negative examples.

- After validating our approach on standard datasets used in computer vision, in subsection 4.3.2, we show how SSL and `MYOW` can be applied to multi-neuron recordings. To the best of our knowledge, this is the first time that SSL has been applied to these types of brain datasets that capture activity at the level of individual neurons. We establish a set of universal augmentations, that can be successfully applied to datasets spanning non-human primate, rat, and mouse.

- In our experiments on neural datasets (subsection 4.3.3), we show that by linking "semantically close" yet temporally separated brain states, `MYOW` yields significant improvement in the decoding of behavior when compared to other

self-supervised approaches. We also observe that in some datasets, the linear readouts from our representation layer provide better decoding performance than supervised methods, suggesting that MYOW can be a powerful tool for reading out information from neural circuits.

4.2 Mine Your Own vieW (MYOW)

In this section, we introduce MYOW, our proposed self-supervised approach for across-sample prediction (see Figure 4.1).

4.2.1 Combining augmented and mined views through cascaded predictors

To build a representation, we will leverage the predictive framework introduced in BYOL [83] which aims to maximize similarity across *augmented views*. Instead of relying solely on instance-level augmentations, MYOW finds *mined views*, or views of different samples that are close in the latent space. We now provide a detailed overview of our method starting with the architecture, and then describing our view mining approach (see subsection 6.3.1 for pseudocode).

View generation. Given a sample $\mathbf{s} \in \mathcal{D}$ from our dataset, we generate two augmented views $\mathbf{x}, \mathbf{x}'$ using transformations t, t' sampled from a set $\mathcal{T}$. A third view $\mathbf{x}_m$ of the same example is also generated, while the mined view $\mathbf{x}'_m$ is of a different sample $\mathbf{s}'$ *selected* from the dataset. The transformations t_m, t'_m to produce these views are sampled from a set $\mathcal{T}_m$ which is not necessarily the same as $\mathcal{T}$. Different heuristics can be designed to mine views; in the next section, we present a simple nearest neighbor strategy, which uses points that are nearby in the representation space of the network to serve as positive examples for each other.

Dual deep architecture. Both types of views are fed through *online* and *target* networks, parameterized by weights θ and ξ, respectively. The encoders produce representations $\mathbf{y} = f_\theta(\mathbf{x})$ and $\mathbf{y}' = f_\xi(\mathbf{x}')$, which are then passed through a projector

to obtain $\mathbf{z} = g_\theta(\mathbf{y})$ and $\mathbf{z}' = g_\xi(\mathbf{y}')$. Mined views are further projected in secondary spaces to obtain $\mathbf{v}_m = h_\theta(\mathbf{z}_m)$ and $\mathbf{v}'_m = h_\xi(\mathbf{z}'_m)$. The projections in the target network act as targets for their respective predictors: q_θ forms predictions across augmented views and r_θ forms predictions across mined views.

Loss function. MYOW learns a representation by minimizing both augmented and mined prediction errors through the following loss:

$$\mathcal{L} = \underbrace{d(q_\theta(\mathbf{z}), \mathbf{z}')}_{\text{Augmentation Loss}} + \lambda \underbrace{d(r_\theta(\mathbf{v}_m), \mathbf{v}'_m)}_{\text{Mining Loss}}, \quad \text{with} \quad d(\mathbf{u}, \boldsymbol{\nu}) = -\frac{\langle \mathbf{u}, \boldsymbol{\nu} \rangle}{\|\mathbf{u}\|_2 \|\boldsymbol{\nu}\|_2}, \tag{4.1}$$

where λ is a weight that regulates the contribution of the mined views in the objective; in practice, λ has an initial linear warmup period of a few epochs. Just as in BYOL, we symmetrize the distance between augmented views by feeding $\mathbf{x}'$ and $\mathbf{x}$ to the online and target network, respectively.

We use the same approach for optimizing the online and target networks as proposed in BYOL. The loss $\mathcal{L}$ is optimized only in terms of θ and ξ is updated according to a moving average of θ. In particular, we update the online and target networks according to the following:

$$\theta \leftarrow \text{optimize}(\theta, \nabla_\theta \mathcal{L}, \eta), \quad \xi \leftarrow \tau\xi + (1-\tau)\theta, \tag{4.2}$$

where $\tau \in [0, 1]$ is a momentum parameter, and η is the learning rate used to optimize the weights of the online network. We point the reader to a discussion of the cost and benefits of different components of this dual network implementation (i.e., stop gradient, predictor, momentum) [136].

4.2.2 How to mine views

Randomized nearest-neighbor selection approach. MYOW adaptively "mines" samples in the dataset that are neighbors in the representation space and uses them

as positive examples. One could imagine many strategies for doing this; we show that a simple random k-nearest neighbor (k-NN) strategy suffices. Specifically, given an anchor sample $\mathbf{s}$, we draw a set of L candidate samples and apply transformations sampled from a set $\mathcal{T}_m$.[1] The anchor sample is passed through the online encoder to obtain its representation $\mathbf{y}_m = f_\theta(\mathbf{x}_m)$, where $\mathbf{x}_m = t_m(\mathbf{s})$ and $t_m \sim \mathcal{T}_m$. The candidate views $\{\mathbf{x}_j\}$ (generated from other samples) are projected in the target encoder's space to obtain $\mathcal{S} = \{f_\xi(\mathbf{x}_j)\}_L$. The k-nearest neighbors of the anchor representation $\mathbf{y}_m$ are computed from this set $\mathcal{S}$ and one of these neighbors is randomly selected as the mined view $\mathbf{x}'_m$.

Controlling stochasticity in mining. There are two main parameters that must be specified for mining, the number of nearest neighbors (k) and the number of samples that are considered as candidates for mining (L). Both of these parameters control the diversity and randomness of which views may be selected. Only a fraction of the dataset (L/N) is used during the mining process, the smaller this fraction gets, the more stochastic the mining becomes: at the end of training, each sample would have seen a large and diverse set of mined views. In the case of the image datasets we study, we are able to use a pool of candidates of size equal to the batch size $L = B = 512$ with $k = 1$. On neural datasets, we find that slightly higher values of k are more favorable, suggesting that more stochasticity is helpful in this case. In all of our experiments, we find that `MYOW` can be effectively trained using $L = B$.

Defining which samples can be selected through mining. When mining views, our algorithm can flexibly accommodate different constraints into our mining procedure. While not necessary in images, when mining in temporal data (like our neural examples), we know that temporally close data points can be selected as augmentations and thus it is useful to restrict the mining candidates to samples that

[1] In general, the set of transformations that we use for mined views, $\mathcal{T}_m$, can be different from the set $\mathcal{T}$ used for augmented views; particularly in the case of images, we empirically find that the use of simpler transformations is more favorable.

are either farther in time from the anchor sample or in entirely different temporal sequences. Further details on our mining procedure can be found in subsection 6.3.2; we note that the same global pool of candidates of size L is used for all samples in a batch.

4.2.3 Memory and computational requirements

In our experiments, the pool of candidates is resampled on-the-fly at each iteration and thus `MYOW` does not require a memory bank. While there is an additional, but negligible (less than 2%), memory overhead due to the k-NN operation, the memory requirements for training `MYOW` are not different from `BYOL` 's when $L \leq B$. This is because augmented and mined views are forwarded sequentially through the network and gradients are accumulated before updating the weights. To reduce the extra computational overhead due to mining, we use the candidates' target representations instead of their online representations and avoid an extra forward pass. We empirically find that mining in either the online or target network leads to similar results (subsection 6.3.6) and thus use this strategy in practice. In this case, `MYOW` requires 1.5x computation time when compared to `BYOL`. When memory is not an issue, computation time can be reduced significantly by feeding in all views at the same time. When using a multi-GPU setup, we distribute the computation of the candidate's representations over all GPUs and then have them broadcast their local pools to each other, effectively building a pool of mining candidates of larger size.

4.3 Experiments

In order to evaluate our approach, we first test it on benchmark datasets used for image recognition in computer vision. After we establish the promise of our approach on images, we then focus our attention on a novel application of SSL to decoding latent variables from neural population activity.

4.3.1 Image datasets: Comparisons and ablations

Experimental setup. To train our model and other SSL approaches on natural images, we follow the procedures reported in previous work [111, 136, 137], both to augment the data and evaluate our models (see subsection 6.3.3). We train the networks for 800 epochs and use a batch size of 512. When mining, we use an equally sized pool of candidates $L = 512$, as well as $k = 1$ and $\lambda = 0.1$. During training we use an SGD optimizer with a learning rate of $\eta = 0.03$ to update the online network, and a moving average momentum of $\tau = 0.996$ for the target network. For all ResNet-18 and ResNet-50 experiments, we train using 1 and 2 GTX 2080Ti GPU(s), respectively. We assess the quality of the representations by following the standard linear evaluation protocol: a linear layer is trained on top of the frozen representation, and the accuracy is reported on the validation set. Models trained on CIFAR-100 are also evaluated on CIFAR-20 which aggregates labels into 20 superclasses.

Results on natural images. In our experiments, we compare `MYOW` with both `BYOL`, and `SimCLR` on CIFAR-10, CIFAR-100 and Tiny ImageNet (Table 4.1). Consistently, `MYOW` yields competitive results with these state-of-the-art methods, and outperforms `BYOL` even when they share the same random seed and the same hyper-parameters. We rule out the possibility that `MYOW` simply benefits from an effectively higher batch size by conducting experiments where the batch size or number of epochs used in `BYOL` is increased by 50% (subsection 6.3.5). More significantly, we find, for the CIFAR-10 experiment, that `MYOW` surpasses `BYOL`'s final accuracy only after 300 epochs, which, in this case, largely justifies the additional computational cost of our approach. When we consider a limited augmentation regime (Table 4.2), we find that `MYOW` increases its gap above `BYOL`. Overall, we find that `MYOW` provides competitive performance on the vision datasets we tested.

Examining mined views. Figure 4.2 highlights examples of views mined during training, where we can see the rich semantic content shared within each pair. Even

Table 4.1: *Accuracy (in %) for classification on CIFAR-10, CIFAR-100 and Tiny Imagenet.* We report the linear evaluation accuracies for different architectures and datasets. For CIFAR-100, we report both accuracies under linear evaluation on CIFAR-100 and CIFAR-20. Results for `SimCLR` are reported from [132].

	ResNet-18				*ResNet-50*		
Method	CIFAR-10	CIFAR-100	CIFAR-20	Tiny ImageNet	CIFAR-10	CIFAR-100	CIFAR-20
`SimCLR` *	91.80	66.83	-	48.84	91.73	-	-
`BYOL`	91.71	66.70	76.90	51.56	92.12	67.87	77.38
`MYOW`	**92.10**	**67.91**	**78.10**	**52.58**	**93.18**	**68.69**	**78.87**

Table 4.2: *Accuracy (in %) for different classes of transformations.* We report the linear evaluation accuracies for `BYOL` and `MYOW` trained on CIFAR-10 using ResNet-18.

	Original	Remove Grayscale	Remove Color	Crop only
`BYOL`	91.71	88.04	87.13	82.10
`MYOW`	92.10	89.16	89.38	84.82

when mined views are not from the same class, we find other semantic similarities shared between the views (see the penultimate column where we select a Dachshund dog and the horse with similar body shape and color through mining). While we do find that the mining process does not always select positive examples from the same class (refer to subsection 6.3.6), the presence of these across-class predictions does not seem to hinder performance.

Ablations. Our architecture integrates mined views through a second cascaded projector/predictor. On both MNIST and CIFAR-10, we performed architecture ablations to study the role of our cascaded architecture compared to a single projector or parallel dual projectors (subsection 6.3.7). Our experiments reveal that all three configurations (cascaded, single, parallel) lead to an improvement over the `BYOL` baseline in CIFAR-10, with the cascaded architecture showing the best performance. We also perform ablations on the class of transformations $\mathcal{T}_m$ used for mined views (subsection 6.3.6), and find that, when training on the CIFAR-10 dataset, the use of minimal to no transformations yields the best result.

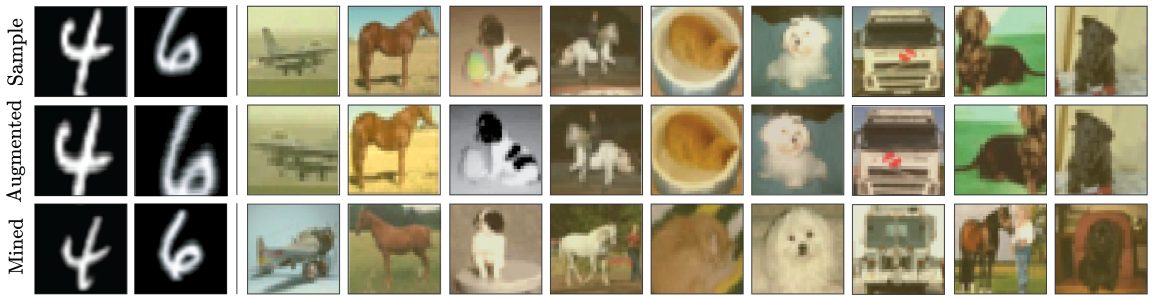


Figure 4.2: *Examples of mined views from MNIST (left) and CIFAR-10 (right).*

4.3.2 Neural datasets: Identifying classes of augmentations

After establishing our method on image datasets, we set out to test our approach on multi-neuron recordings. As this is the first attempt at leveraging a self-supervised learning framework for neural data of this nature, our first goal was to establish simple yet general classes of augmentations that can be utilized in this application.

Neural datasets and decoding tasks. In our experiments, we consider a total of six neural datasets from both non-human primates and rodents.

1) Reaching datasets. The first datasets we will consider are acquired from the primary motor cortex (M1) of two non-human primates (NHPs), Chewie and Mihi, while they repeatedly made reaching movements towards one of eight targets [50]. We call each repetition a "trial". Spiking activity of d single neurons is recorded during each reach trial, Figure 4.3-A shows some instances of the trajectory of the joystick during movement. Data was collected at two different dates (77 days apart for Chewie, 3 days apart for Mihi), resulting in two datasets per primate, each targeting a different set and number of neurons in the same part of M1. The activity of neurons was spike-sorted and binned into 100ms intervals to generate around 1.3k d-dimensional vectors per dataset. To measure representation quality, we will define our target downstream task as the *decoding of the intended reach direction* from the neural activity during the movement.

2) Sleep datasets. The second datasets that we will consider are collected from rodent visual cortex (V1) and hippocampus (CA1) during free behavior over 12 hours [138]. Here, neural activity was binned into 4s intervals to produce firing rates for 42 and 120 single neurons, for a rat and mouse, respectively. To measure the quality of representations learned, we will define our downstream task as the *decoding of the arousal state* of the rodent into one of three classes: rapid eye movement (REM) sleep, non-REM sleep, or wake [139, 138].

Experimental setup. For all datasets, we use multi-layer perceptrons (MLPs) as encoders with a representation size of 64 and 32, for primate and rodent data respectively. We train the networks for 1000 epochs and use a batch size of 512. When mining we use an equally sized pool of candidates $L = 512$, as well as $k = 3$ and $\lambda = 0.1$. During training we update the online network using AdamW with a learning rate of $\eta = 0.02$ for primates and $\eta = 0.001$ for rodents and weight decay of $2 * 10^{-5}$, and use a moving average momentum of $\tau = 0.98$ for the target network. Each dataset is temporally split into (70/10/20%) train/validation/test sets. More details on the datasets and experimental setup can be found in subsection 6.3.4.

Augmentations for spiking neural data. While self-supervised approaches have not been applied to the multi-neuron recordings that we consider, we take cues from other domains (video, graphs), as well as previous work on electroencephalogram (EEG) data [140, 141], to define simple classes of augmentations for our datasets. Specifically, we consider four different types of augmentations: (i) *Temporal Jitter*–stochastic backward or forward prediction of nearby samples within a small window around the sample, (ii) *Dropout*–masking neurons with some probability, and (iii) *Pepper*–sparse additive noise, and (iv) *Noise*–additive Gaussian noise.

We test the inclusion and combination of these different augmentations, first on our `BYOL` backbone which uses augmented views only (Figure 4.3-B). While we find that temporal jitter alone is insufficient to drive learning, when we combine both

jitter and dropout, we see a substantial increase in decoding accuracy and qualitative improvements in the resulting representations. In this case, our baseline SSL method, BYOL, quickly starts to create meaningful predictive relationships between data, as evidenced by our decoding results and qualitative evaluations of the representations (subsection 6.3.10). As we include additional augmentations (*Noise* + *Pepper*), the performance increases further, but by smaller margins than before. In general, we see these same trends observed throughout our remaining primate datasets and in our experiments on rodent (see subsection 6.3.10), suggesting that these classes of transformations are good candidates for building SSL frameworks for neural activity.

After establishing a good set of simple augmentations, we then integrate *mined views* with MYOW (Figure 4.3-B, blue). In this case, we can interpret mined views as *nonlocal brain states* that are not temporally close but can be semantically similar. For instance, in our reaching datasets, MYOW will mine outside of the current reach and look for other samples that it can use to build a more unified picture of the brain states as they evolve. Through combining simple augmentations with nonlocal samples with MYOW, we provide an impressive boost in performance over BYOL on this application.

Table 4.3: *Accuracy (in %) in the prediction of reach direction from spiking neural activity.*

	Chewie-1		Chewie-2		Mihi-1		Mihi-2	
	Acc		Acc		Acc		Acc	
Supervised	63.29	77.22	72.29	81.51	63.64	79.02	61.49	68.44
pi-VAE	65.63	**82.62**	60.60	74.64	62.44	77.12	63.26	77.58
AE	48.40	67.51	46.79	65.84	50.94	68.03	55.19	74.98
RP	59.21	78.10	50.69	60.01	57.78	76.03	53.76	71.34
TS	60.16	78.76	49.48	63.55	59.23	76.98	54.10	71.65
SimCLR	61.36	79.40	51.62	65.01	59.41	77.82	56.29	74.57
BYOL	66.65	78.17	64.56	77.22	72.64	85.14	67.44	**82.17**
MYOW	**70.54**	79.99	**72.33**	**84.81**	**73.40**	**85.58**	**71.80**	81.96

4.3.3 Neural datasets: Examining and comparing representation quality

Next, we will test the representational quality of our model by asking how well relevant behavioral variables can be linearly decoded from the learned representations.

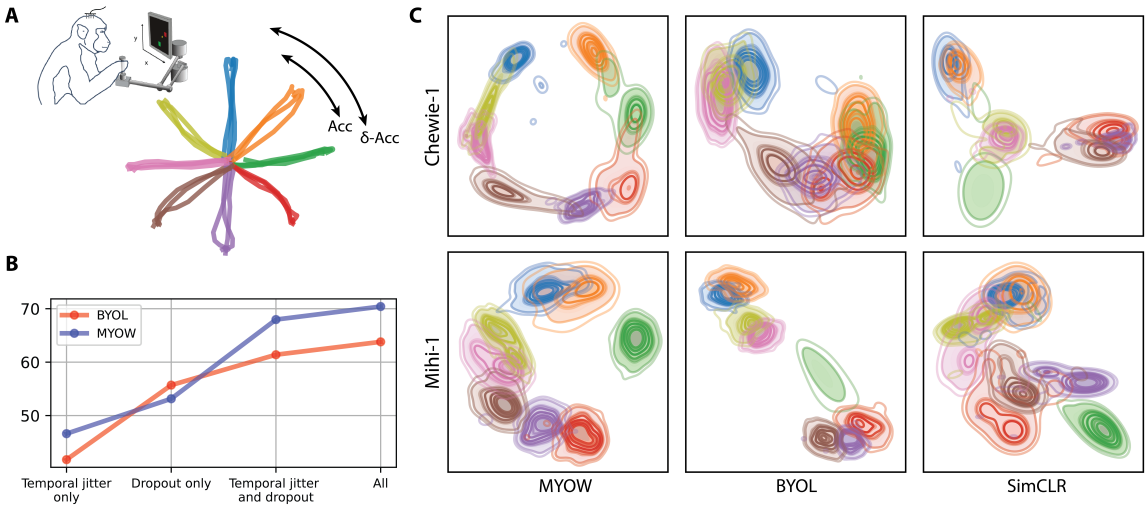


Figure 4.3: (A) Sketch of a primate performing a reach task with sample joystick trajectories depicting the center-out reach movement. (B) Increase in decoding accuracy of `BYOL` and `MYOW` as we progressively introduce new augmentations. (C) Visualization of the learned representations obtained for the two primates Chewie and Mihi when different SSL methods are applied (embeddings are obtained via t-SNE). This shows how `MYOW` reveals the underlying structure of the task, as clusters are organized in a circle.

Decoding movements from the primate brain. In the reaching datasets that we consider here, there is a direct connection between the neural state (brain activity across many neurons) and the underlying movements (behavior). Thus, we wanted to assess the quality of the representations learned from these datasets by asking how well we can predict the reach direction from neural activity. If we have a good representation, we should be able to better separate reach direction from the neural activities. To quantify this, we will use a linear readout to predict the cosine and sine of the reach direction, and report the classification accuracy. We also introduce a slightly relaxed accuracy metric that we call the (akin to Top-k), which has a larger true positive acceptance range, as can be seen in Figure 4.3-A. (see subsection 6.3.4 for a formal definition).

We compare our approach with several self-supervised methods, including state-of-the-art methods `BYOL` and `SimCLR`, as well as two widely used self-supervised tasks

recently applied to EEG data called Relative Positioning (RP) and Temporal Shuffling (TS) [142]. RP trains the network by classifying whether two samples are temporally close, while TS takes in three samples and learns whether they are in the right order or if they are shuffled. In addition to these self-supervised methods, we also train a Multi-layer Perceptron (MLP) classifier (Supervised) using weight regularization and dropout (in nodes in intermediate layers in the network), an autoencoder (AE), and a state-of-the-art supervised approach for generative modeling of neural activity (pi-VAE) that leverages behavioral labels to condition and decompose the latent space [46].

We find that MYOW consistently outperforms other approaches and that contrastive methods that rely on negative examples (SimCLR, RP and TS) fall behind both MYOW and BYOL. We also find that MYOW generalizes to unseen data more readily than others; in some cases, beating supervised approaches by a significant margin, with over 10% on both Mihi datasets. When we consider , our method scores above 80% on all datasets, outperforming the supervised baseline by over 10% on Mihi-2. These results are even more impressive considering that we only tune augmentations and hyperparameters on Chewie-1 and find that MYOW consistently generalizes across time and individuals. We thus show that by integrating diverse views (across trials) through mining into our prediction task, we can more accurately decode movement variables than supervised decoders.

When we visualize the learned representation in Figure 4.3-C, we notice that MYOW organizes representations in a way that is more reflective of the global task structure, placing reach directions in their correct circular order. In contrast, we find that in both individuals, other methods tend to distort the underlying latent structure of the behavior when visualized in low-dimensions (subsection 6.3.11). We conjecture that across-sample predictions (including those across different reach directions), may be responsible for capturing this kind of higher-level structure in the data.

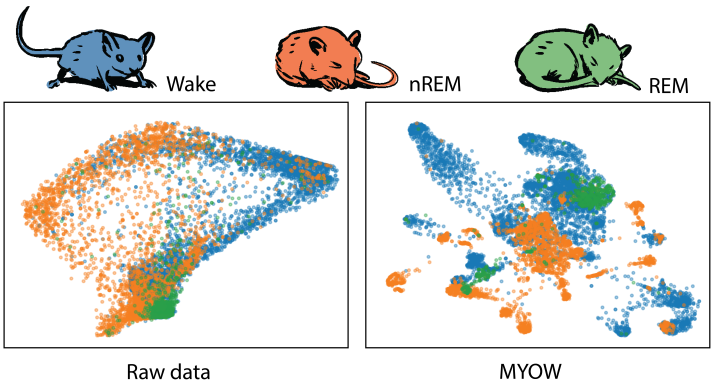


Figure 4.4: Visualizations of raw and latent spaces (using t-SNE) of 12 hour recordings from mouse (CA1) during free behavior, including sleep and wake. One variable of interest is the arousal state (REM, nREM, Wake).

Table 4.4: F1-score (in %) in the prediction of arousal state from spiking neural activity.

	Rat-V1	Mouse-CA1
`Supervised`	86.34	93.01
`pi-VAE`	73.10	82.48
`AE`	34.17	57.73
`RP`	82.93	82.12
`TS`	82.45	81.93
`SimCLR`	81.03	81.94
`BYOL`	85.42	93.24
`MYOW`	**88.01**	**93.70**

Decoding arousal states from the rodent brain during free behavior. Next, we applied `MYOW` to datasets from the rodent cortex and hippocampus, where we test our ability to decode arousal states (REM, nREM, Wake) from the learned representations. Despite the strong class imbalance, the trends are similar to that of our earlier experiments, with `MYOW` providing robust performance, exceeding that of the supervised baseline, and outperforming other self-supervised methods.

In these datasets, the animal is "untethered" and can roam around in its cage without any task or explicit instructions. In these free-behaving conditions, we find a great deal of variability in the latent state beyond the coarse labels that we have access to. When we visualize the representation learned by `MYOW` in Figure 4.4, we find that the network separates different parts of the behavior space, revealing subspaces of neural states that are otherwise unobservable when examining the embeddings of the raw data.

4.4 Related Work

Self-supervised learning. SSL aims to learn representations of unlabeled data that are useful for downstream tasks. While early work utilized proxy tasks for self-supervision [143, 144], instance discrimination-based SSL methods [111, 124, 83, 133] have emerged as the state-of-the-art for representation learning, showing tremendous success and moving towards closing the gap with supervised learning. Conceptually, these approaches treat each instance in the dataset as its own class. A given sample is transformed to create distinct *positive views*, which are encouraged to be close in terms of their representations, while negative pairs are pushed apart. `BYOL` [83], `SimSiam` [136], and more recently `BarlowTwins` [133] move away from the explicit contrastive framework and the reliance on negative samples by employing different strategies that avoid collapse in the representation. The precise mechanisms underlying the success of `BYOL` [83] are still unclear and have been the subject of recent theoretical

and empirical studies [145, 146].

Connections to mining hard negatives in contrastive learning. In contrastive learning, it is a commonly held belief that the use of large numbers of negative examples is necessary to introduce enough "hard negative examples" into learning. Thus, there has been interest in nearest-neighbor sampling and mixing to define *hard negative examples* [147, 148, 149] instead of just relying on larger batch sizes. Interestingly, the mined views in `MYOW` can be considered as *harder positive examples*, but are different from their negative counterpart in that they define a new type of views.

Clustering-based SSL and local aggregation (LA). Clustering-based representation learning methods are different from instance-specific contrastive methods in that they do not compare pairs of samples directly, but do it through the use of prototypes or pseudolabels. DeepCluster [150], for example, uses k-means assignments as pseudolabels for training. LA [151] leverages neighbors to guide learning by defining two sets of neighbors, close and background neighbors, encouraging close neighbors to be nearby while pushing them away from a set of background neighbors. More recently, SwAv [152] simultaneously learns a set of prototype vectors and enforces consistency between cluster assignments of two positive views.

Like many of these methods, we select samples with similar embeddings and use them to adaptively link data samples in the latent space. However, instead of using a small number of prototypes to cluster the representations, we use neighbors in the representation space as positive views for prediction and do not force any kind of explicit clustering. Moreover, because our model is built on `BYOL`, *we do not require negative examples* and also avoid the introduction of more complex distance measures to establish contrast (e.g., close vs. background neighbors).

Applications of SSL in neuroscience and biosignal analysis. Previous work in self-supervised and contrastive learning for sequential data often leverages a slowness

assumption to use nearby samples as positive examples and farther samples as negative examples [127, 153, 154, 155, 156]. Contrastive predictive coding (CPC) [127] further leverages the temporal ordering in sequential data by building an autoregressive (AR)-model that predicts future points given previous observed timesteps. In reinforcement learning, PBL [85] also uses a similar strategy, however, they show similarly to BYOL that negative examples are not needed to learn a good representation.

In [156], the authors test different temporal contrastive methods (RP, TS and CPC) on EEG datasets. They find that, despite the additional complexity afforded by TS and CPC, these approaches perform similarly to RP in their experiments on sleep decoding from the human brain. In [140], they propose a contrastive learning method for EEG that also leverages subject-level information to build representations. Our approach shares similarity with these existing approaches in how we build augmented views for neural data. However, MYOW goes beyond these temporally local predictions to incorporate nonlocal time points as positive examples. We show that non-local predictions across samples can be used to significantly boost performance for our neural datasets, and thus we expect that nearest-neighbor based approaches could also be used to extend these previous applications of SSL in neuroscience.

4.5 Conclusion

This chapter introduces a new method for SSL that integrates diverse across-sample views into learning through a novel cascaded architecture. We show that our approach can be used to learn meaningful representations on a variety of image and neural datasets.

This chapter provides an important first step towards applying self-supervised methods to learn representations of neural activity. For these datasets, we establish general classes of augmentations and study the impact of these augmentations on diverse neural recordings. Our results in this domain are compelling: we typically

obtain better generalization than supervised methods trained with dropout and weight decay. Through the inclusion of temporal structure into our framework and architecture, we may be able to improve this approach even further and capture dynamics over longer timescales.

In our application to spiking neural data, we demonstrate that both dropout and temporal augmentations are necessary for building meaningful representations of different brain states. Similarly in neural circuits, neurons are unable to send direct signals to every other neuron in a downstream population; thus, target areas receiving signals may need to predict future brain states from partial information [157]. Our results suggest that it may be fruitful to try to understand how brains may leverage dropout to build predictive representations, and that a theoretical understanding of SSL might yield insight into these processes.

CHAPTER 5
AFTERWORD

This thesis has presented a series of novel methodologies and frameworks aimed at advancing our understanding of the brain through the integration of diverse and heterogeneous datasets. The main contributions of this work include the development of POYO, a scalable framework for neural population decoding that can generalize across different sessions, subjects, and species; BAMS, a self-supervised approach for multi-timescale behavior representation learning, which successfully captures the complex dynamics of behavior across different contexts; and the introduction of MYOW, a self-supervised learning technique designed to extract robust and invariant representations from neural population activity.

By addressing challenges such as the variability of recorded neural populations and the limitations posed by both labeled and unlabeled data, this thesis contributes to the ongoing efforts to create more comprehensive models of brain function. The approaches presented here offer new tools that can enhance our understanding of the brain.

It is my hope that this thesis contributes to the democratization of AI tools for neuroscientists, providing them with the advanced capabilities needed to explore and uncover new insights in the brain. By making these tools more accessible and effective, we can empower researchers to push the frontiers of science, leading to breakthroughs that enhance our understanding of the brain and, ultimately, improve human health and well-being.

CHAPTER 6

APPENDIX

6.1 A unified scalable framework for neural population decoding

6.1.1 Additional Model Details

Rotary position encoding

Rotary Position Embeddings (RoPE) are used to incorporate relative positional information into transformer models in a memory and compute efficient manner[19]. Unlike absolute position embeddings which are added or appended to the input embeddings, these are directly incorporated in the attention mechanism.

First, we define a 2×2 rotation matrix $\mathbf{R}_{2\times2}(t, T)$ for a given timestamp t and time-period T:

$$\mathbf{R}_{2\times2}(t, T) = \begin{bmatrix} \cos(2\pi t/T) & -\sin(2\pi t/T) \\ \sin(2\pi t/T) & \cos(2\pi t/T) \end{bmatrix}$$

$\mathbf{R}_{2\times2}(t, T)$ provides a time/positional encoding with a time period T. Intuitively, the sinusoidal values in this rotation matrix can help the model learn and distinguish variations in timestamps at a timescale T.

Given the head dimension of the attention block D, we create a $D \times D$ rotary embedding matrix $\mathbf{R}(t)$ for a timestamp t by constructing a block-diagonal matrix of rotation matrices $\mathbf{R}_{2\times2}$, where the time-periods are varied logarithmically between

T_{min} and T_{max}

$$\mathbf{R}(t) = \begin{bmatrix} \mathbf{R}_{2\times2}(t, T_0) & 0 & \cdots & 0 \\ 0 & \mathbf{R}_{2\times2}(t, T_1) & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \mathbf{R}_{2\times2}(t, T_{\frac{D}{2}-1}) \end{bmatrix}$$

Given D, T_{min}, and T_{max}, we generate T_i by the following equation:

$$T_i = T_{min} \left(\frac{T_{max}}{T_{min}} \right)^{2i/D}$$

In our models, T_{min} and T_{max} were set to $1ms$ and $4s$ respectively.

This rotation matrix is incorporated into the attention mechanism by replacing the attention score calculation with the following equation:

$$\mathbf{a}_{ij} = \text{softmax}\big(\left(\mathbf{R}(t_i)\mathbf{q}_i\right)^T \left(\mathbf{R}(t_j)\mathbf{k}_j\right) \big)$$

Since $\mathbf{R}(t_i)^T\mathbf{R}(t_j) = \mathbf{R}(t_j - t_i)$, the attention score $\mathbf{a}_{ij}$ depends only on the relative timing between the query and key tokens. This is what makes RoPE relative in nature.

What is explained above has been introduced in [19], and use successfully in many models [18]. In our experiments, we found an interesting extension of RoPE. In addition to incorporating RoPE in calculating the attention scores, we also incorporate it in the weighted-summation of values inside the attention mechanism. We do this to encode the relative timing between values and queries in the output of the attention block. A direct, but computationally inefficient, way of doing so would be:

$$\mathbf{z}_j \leftarrow \mathbf{z}_j + \sum_i \mathbf{a}_{ij}\mathbf{R}(t_i - t_j)\mathbf{v}_i$$

This is inefficient since it requires us to explicitly compute $\mathbf{R}(t_i - t_j)$ for all N^2

values of $(t_i - t_j)$. We use the fact that $\mathbf{R}(t_i - t_j)$ can be factorized as $\mathbf{R}(-t_j)\mathbf{R}(t_i)$ to reach the following, much more efficient, implementation:

$$\mathbf{z}_j \leftarrow \mathbf{z}_j + \mathbf{R}(-t_j) \sum_i \mathbf{a}_{ij} \mathbf{R}(t_i) \mathbf{v}_i$$

That is, we first apply rotation matrices $\mathbf{R}(t_i)$ to all values $\mathbf{v}_i$ before passing them to as inputs to the attention block. Then we apply the rotation matrices $\mathbf{R}(-t_j)$ to its outputs.

This *value-rotation* mechanism captures relative-timing information more explicitly in the output of the attention block, allowing the feedforward network to take advantage of timing information. In the absence of this mechanism, the only way timing information is added to the latent tokens is through the attention scores. We observed in early experiments that our model performed better with value-rotation enabled for the input cross-attention and all self-attention layers.

As discovered in the Perceiver framework [13], we observed an improvement in performance upon rotating only half of the head dimensions. That is, half of the $\mathbf{R}_{2\times 2}$ matrices in $\mathbf{R}$ were replaced by 2×2 identity matrices.

Time-based context window

We consider an arbitrary window of time $[t, t + T]$, meaning that we do not use the trial structure when training our model. In our experiments, we set T equal to 1 second. At inference time, we use a sliding window of step 500ms and average the predicted output in overlapping segments.

In language models, and more generally transformer-based models, the context window is defined with respect to the number of tokens. The neural data modality is a bit more complex, as the number of spikes scales with the number of units: Across multiple datasets, we record a variable number of units, ranging from tens of units

to hundreds of units. Defining the context window with respect to number of tokens will result in a variable time duration, long for small populations and short for larger ones. Our context window is defined with respect to time (fixed to $1s$), each sequence will have a variable number of tokens (M). When creating batches, we pad to the largest sequence in the batch, and appropriately use an attention mask in the first cross-attention layer. GPU hardware is optimized to process fixed-size tensors, to make computation more efficient, we introduce a distributed training load balancing mechanism, which deals with variable-length input sequences (paragraph 6.1.1).

Further details on tokenization

Delimiters. When extracting the spike data in a window of time $[t, t+T]$, we include delimiters for each unit, indicating to the model that we are tuning into the activity of that unit between times t and $t+T$. This choice is important because: 1) we are using relative position encoding, so the beginning or the end of the window is not clear, and 2) our tokenization only captures activity and not inactivity. If we consider a unit that is inhibited and does not fire in that window of time, the model would be unaware of the presence of that unit, unless we add [START] and [END] tokens. We learn two embedding vectors for these delimiters $\mathbf{x}_{start}$ and $\mathbf{x}_{end}$. For each unit in the population, we add these two tokens with embedding equal to the sum of the delimiter embedding and the unit embedding, and assign each token timestamps t and $t+T$ respectively.

Latent tokens. Recall that we divide the latent tokens into groups of n and spread each group uniformly over the context window. Specifically, we use a total of 256 latent tokens and divide them into groups of 8, this means that there will be 32 tokens that share the same timestamp. We experiment with weight-sharing, where we set the learned latent tokens in the same group to have the same weights ($\mathbf{e}_0, \ldots, \mathbf{e}_{31}$ in

Fig Figure 4.1). This means that tokens in the same group (size 8) will have the same embedding $\mathbf{z}_{0,i} = \mathbf{e}_{(i \bmod 32)}$, but different timestamps ($t_0^l, \ldots, t_7^l$ in Fig Figure 4.1). We believe this is a reasonable inductive-bias to architecturally enforce since, intuitively, we want all sets of latent tokens that share a timestamp to query the spike tokens in the exact same manner, just with a different timestamps. These tokens will evolve independently once the information is pulled from the input space (after the first cross-attention).

Training details

The model is trained using the LAMB optimizer [25] with weight decay. The learning rate is held constant, then decayed towards the end of training (last 25% of epochs), using a cosine decay schedule. Single-session models are trained on a single GPU with a batch size of 128 while large models are trained with 8 GPUs with a total batch size of 1400. Note that we didn't see any benefits in increasing the batch size when training single-session models.

Compute The large models are trained on a machine with an AMD EPYC 7452 32-Core Processor and 8 Nvidia A40 GPUs (48Gb memory), `POYO-mp` was trained for 2 days, and `POYO-1` was trained for 5 days (both for a total of 400 epochs). Single-session models are trained with a single Nvidia GeForce RTX 3090 GPU, and take less than an hour to train. Finetuning models is also done with a single GPU. Unit identification converges very quickly and takes less than a minute on a single GPU or a few minutes on the CPU.

Data augmentation Previous work [45, 156] explores the use of augmentations when training neural population models. In our experiments, we use the unit dropout augmentation which randomly samples a subset of the population. We set a minimum population size of 30 for this augmentation to ensure that it is not too destructive (note

that our model is still trained with any number of units, this limit is only imposed for the augmentation). We did not explore the use of other augmentations, but believe it could be a promising direction to further improve the capabilities of the model.

Loss We train our model using a mean-squared error loss over the hand velocity sequence. We normalize the velocity data using z-scoring at the lab level, this means that all velocity data in datasets from the same lab is rescaled using the same scalar.

Dealing with variable sampling rate. When training on mixtures of datasets from multiple labs, any given batch will contain behavior sequences that have different sampling rates. In a window of 1s, some sequence will have more timepoints over which we will evaluate the MSE compared to other sequences. To avoid the over-representation of samples that simply have a high sampling rate, we evaluate the error on a randomly sampled 100 timepoints (these points are randomly sampled at each pass).

Weight in training mix. While behavior during random target tasks can be more complex and noisy, behavior during center-out reaching tasks is more structured, the monkey usually gets a preparation phase, where the movement can be planned [20]. The latter task has been studied extensively for that reason. Knowing that neural activity is very salient during center-out reaching, we increase the weight of the prediction loss during the reaching segments by a factor of 5. The idea of over-weighing samples of "good quality" is not novel and is commonly used when training language models [15].

Load Balancing We distribute our training over multiple GPUs. Note that, in our experiments, the input sequence length (M) can vary anywhere between 1k and 20k tokens. Because of the variability in the length of the input sequences, we distribute

sequences that have close length to same GPU node.

We now provide further details on our load balancing strategy:

- Each GPU node has a local batch size. A GPU that is assigned large sequences will have a smaller local batch size than one that is assigned shorter sequences.
- We create a number of data buckets equal to the number of GPUs. Each bucket i is assigned sequences that are of length $M \in [M^i_{min}, M^i_{max}]$, and a local batch size B_i.
- We select these parameters in such a way that we effectively utilize all available compute capacity, and minimize the amount of padding used.

Evaluation Details

During training, we train on any 1s segment of the recording, but at evaluation we report the decoding performance using the same evaluation strategy used in previous work [24, 10]: for center-out reaching datasets, we report the decoding score during the reaching movement (only when the trial is completed successfully), for random target datasets that are trialized (hold period followed by 3/4 random reaches), we report the decoding score during the reaching as well, and finally for all other continuous random target datasets, we report the decoding score for all segments. Note that none of the segments used for testing are seen during training.

Fine-tuning Details

When finetuning the model, we perform gradual unfreezing of the weights. For the first few steps, we optimize the unit embeddings and the session embedding only (unit identication) and then unfreeze the rest of the weights. We use the LAMB optimizer with batch size 128, learning rate 10^{-4} and weight decay 10^{-4}. We use the exact same hyper parameters and finetuning process for all experiments we report. This strongly

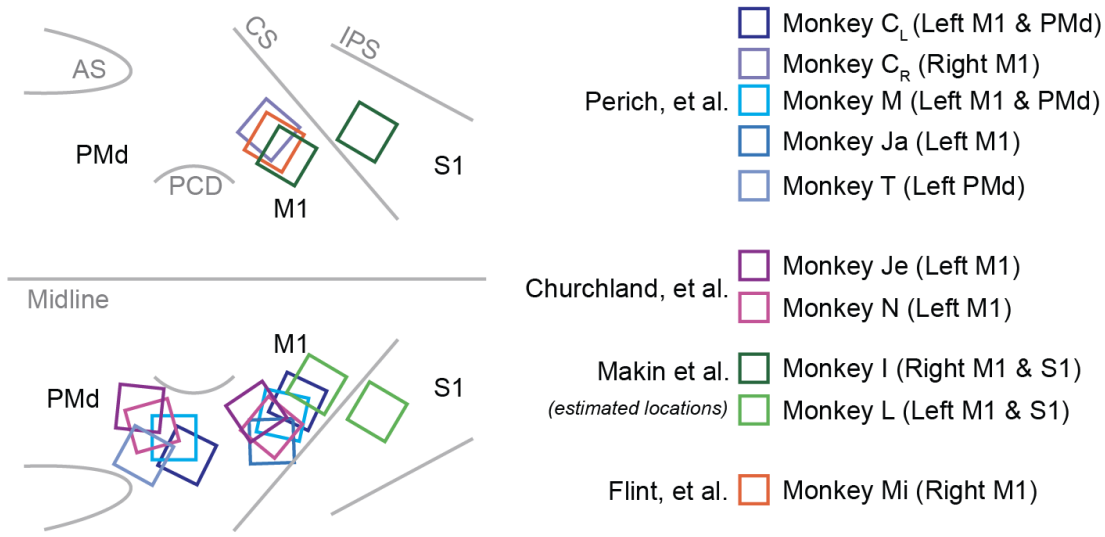


Figure 6.1: Diversity of neural recordings used to train `POYO`. We visualized the array locations for all of the datasets used in our current study. `POYO-mp` is trained on recordings from PMD and M1 (CL, CR, M, Ja) and `POYO-1` is trained on recordings from (CL, CR, M, Ja, I, L, Mi) spanning PMD, M1, and S1 across both hemispheres and in seven animals.

suggests that our method is general, and easy to adapt to new data, without the need for expert neural network training knowledge, or expensive hyperparameter search.

6.1.2 Datasets

To train and evaluate our framework, we curated a large number of open and publicly available datasets that represent a range of neural and behavioral recording platforms.

Datasets used for training

Datasets used to train `POYO-mp`. To build our first large-scale model (`POYO-mp`), we acquired 100 sessions of unique recordings from three non-human primates performing two different movement tasks that were previously studied in four publications [158, 159, 20, 51]. We make these datasets publicly available on the DANDI open repository [26]. In all of these datasets, a nonhuman primate is seated in a primate chair and executes movements using a custom 2-D planar manipulandum to control a cursor on

a computer screen. They performed one of two tasks within a session.

- *Center-out Task* (CO): They performed a standard center-out reaching task involving eight outer targets arranged around a circle with a radius of 8 cm. The monkeys were required to move to the center target, wait for a variable hold period, and then move to one of the outer targets upon receiving an auditory go cue. The task aimed to study neural activity during movement planning, preparation, and execution.

- *Random Target Task* (RT): The setup is similar, the targets are not arranged around a circle but are randomly placed. The monkeys were required to move between 4 targets upon receiving an auditory go cue.

After extensive training, the monkeys were implanted with chronic multi-electrode arrays in the primary motor cortex (M1) and the dorsal premotor cortex (PMd). Neural activity was recorded using a Blackrock Cerebus system, and the data was manually processed offline to identify single neurons. To ensure only independent channels were included, steps were taken to identify and disable potential candidates with high crosstalk and to exclude cells with a high percentage of coincident spikes.

Datasets used to train the multi-lab, multi-session model (`POYO-1`). When training `POYO-1`, we acquire more data as shown in Table 2.1, adding all of the datasets outlined from Churchland et al., [21], Makin et al., [22], and Flint et al., [23]. We still hold out the 12 sessions from Monkey T, 2 sessions from Monkey C, and the two Neural Latents Benchmark (NLB) [24] datasets from training.

As depicted in Figure 2.5, instead of using a manipulandum like in the previous experiments, these new datasets use a touch screen and have different sampling rates for their behavioral outputs. In addition, in the Churchland, et al. datasets, they use threshold crossing based processing rather than applying spike sorting to identify single units. We refer the reader to the listed publications for more details about how

these datasets were collected.

We would like to reiterate that there are major differences in the way the datasets were collected. Yet, in spite of this variability, it is important that our method works without having to standardize the datasets across sites further and apply specialized techniques to one dataset but not other. Thus we did not attempt to homogenize or standardize the data. The only processing that we perform is on the behavior to clip outliers in the behavior based upon times of high acceleration. We use a simple threshold heuristic to avoid training on periods with extremely high acceleration.

This means that:

1. We do not filter units based on a presence ratio, or reject multi-units.

2. We do not re-apply the same spike sorting algorithm on all datasets, and use the data as is. In [21], the spiking events obtained through threshold crossings (i.e. all units are multi-units), and in [22], we have both threshold crossing units and spike sorted isolated single units, and we use both. Each algorithm used in each lab was tuned differently, but we consider this a good example of diversity that a general model needs to deal with.

3. We do not resample or process the velocity timeseries.

Minimizing the amount of processing needed to integrate new datasets into our model is key for more easily scaling to more datasets, and also providing an accessible model that the community at large can use out of the box, without having to adapt to a specific standard.

Datasets for evaluation and fine-tuning

In addition to holding out 20% of the data from each session, we hold out all sessions from Monkey T (6 CO, 6 RT). We also use two datasets from the Neural Latents Benchmark [24], MC-Maze and RTT. The Maze dataset consists of recordings from

primary motor and dorsal premotor cortices of a monkey performing reaches with instructed delays to visually presented targets while avoiding the boundaries of a virtual maze. This dataset offers a variety of behavioral configurations, with each configuration using a different combination of target position, number of virtual barriers, and barrier positions, resulting in a variety of straight and curved reach trajectories. With thousands of trials, the Maze dataset allows for a rich investigation into the structure of population activity, while its instructed delay paradigm enables a clear separation of neural processes related to preparation and execution. On the other hand, the RTT dataset introduces different modeling challenges, as it contains continuous, point-to-point reaches that start and end in various locations, have highly variable lengths, and few repetitions. We report the performance of our models on the same segments of data used to evaluate models in the NLB benchmark.

6.1.3 Additional Results

Comparison with single-session baselines

In our comparisons in Table 2.2, we compared `POYO` against existing baseline models that are commonly used for neural decoding [27]. In particular, we applied a Wiener Filter (WF), Feed Forward Network (MLP), and a Gated Recurrent Unit (GRU). These models predict the behavior at each timestamp t given a history window of neural activity $[t - T, t]$. For the NLB datasets, we also compare with Auto-LFADS [28], an unsupervised approach that can be used to obtain single trial smoothed rate estimates for a population of neurons before applying a linear layer on top of the inferred rates to obtain an estimate of the behavior. While this model isn't designed for decoding per se, it is frequently used for denoising for BMI decoding applications.

For the NLB, we use a 5 ms binning of the neural activity for training models and for evaluation of the behavioral output as this is the finest resolution used in the benchmark. For the rest of the MP datasets, we use a 10 ms binning rate for the

neural data and behavior as this corresponds to the behavior sampling rate for these datasets.

The hyperparameters we used for training our single-session POYO model can be found in Table 6.1.

Table 6.1: Hyperparameters used for training all POYO single-session models

Hyperparameter	**Value**
Embedding Dimension	128
Head Dimension	64
Number of Latents	128
Depth	6
Number of Heads	8
FFN Dropout	0.3
Linear Dropout	0.3
Attention Dropout	0.3
Weight Decay	1e-4
Learning Rate	4e-3
Batch Size	128

Robustness of Unit-identification

Unit-identification, introduced in subsection 2.2.5, can be seen as an approach for *identifying* units from a new dataset. To evaluate the robustness of this process, we ran unit-identification on a session that was already seen during pre-training. We test whether finetuning on this "new" set of units, will map them close to their true embedding (found during pre-training). We use our largest model, POYO-1 , start with randomly initialized unit embeddings and run unit-identification.

We perform this experiment on two different animals. In Figure 6.2, we show the evolution of unit-embeddings during training. To compare the newly calibrated set of units with its pre-trained version, we normalize the unit-embeddings and report the cosine similarity averaged over the set of units. Note that normalization is done because POYO 's first cross-attention layer applies layer norm to these embeddings. We also report the "unit identification accuracy" which we define as the ratio of tuned unit

embeddings that have their true unit embedding as their nearest neighbor. Results for these experiments are present in Table Table 6.2.

Table 6.2: *Unit re-identification results.* Cosine-similarity measure is averaged over all units in a dataset. Accuracy is measured for 1-nearest-neighbor classifier.

Dataset	Num of Units	Unit-Id Accuracy	Unit-Id Cosine-Similarity
Monkey C, CO, 2013/10/03	73	1.000	0.845
Monkey M, CO, 2014/02/03	116	0.966	0.802

Overall, this analysis suggests that the unit-identification approach is robust and reliable for identifying units. We believe that understanding the functional similarities of units that are mapped close to each other in the unit embedding space will be an interesting avenue for future work.

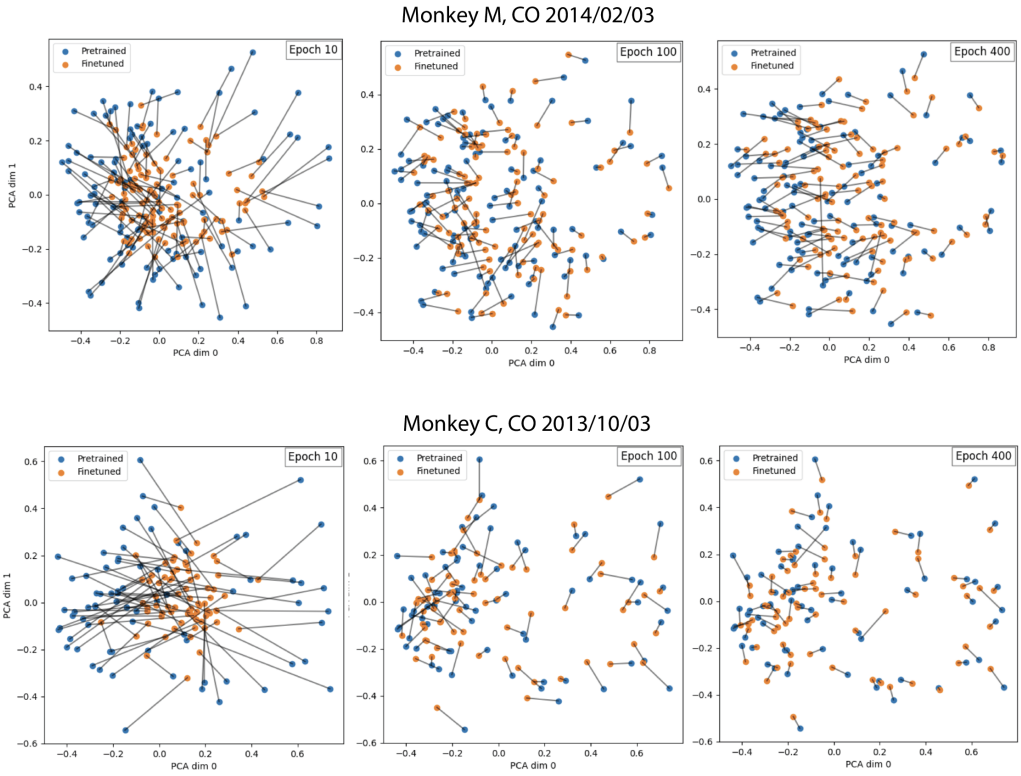


Figure 6.2: Evolution of the unit embeddings with unit-identification on sessions from two different animals. From left to right, the finetuned embeddings (orange) are visualized at 10, 100, and 400 epochs, relative to their true embedding in the pre-trained model (blue).

Hyperparameter sensitivity

When initially training the single-session models, we selected a subset of datasets (out of the 100 sessions) and performed a random hyperparameter search and selected the set of hyperparameters with the best performance on the validation set. These experiments quickly revealed that the model was very stable to a wide range of hyperparameters. In Figure 6.3, we report some examples of this for two sessions. We perform 300 runs for each session, then report the average performance for each hyperparameter.

Given these findings, we selected the best set of hyperparameters for the subset and used the same parameters when training all of the 100 single-session models that we study in Figure 2.2.

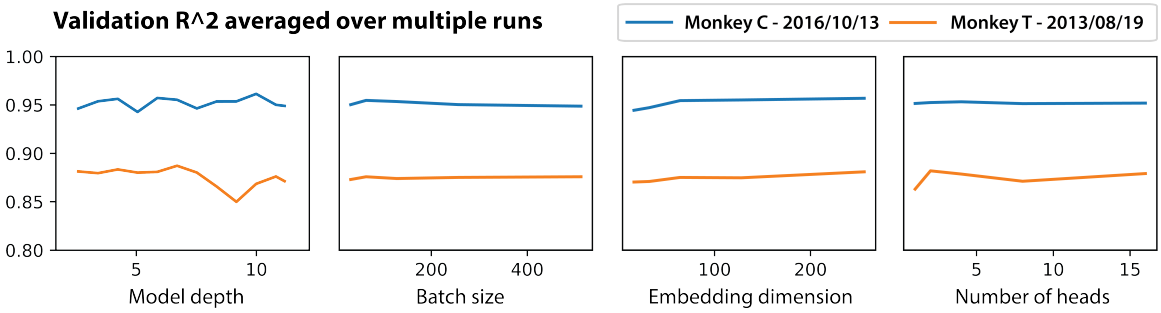


Figure 6.3: Single-session model performance for a wide range of values for different hyperparameters. The R^2 score is reported on the validation sets for both datasets.

6.2 A self-supervised approach for multi-timescale behavior representation learning

6.2.1 Experimental details: Simulated Quadrupeds

Data generation

Simulation details. We record a total of 5182 trajectories. 2756 were generated for robots of type ANYmal B and 2426 sequences were generated for robots of type

ANYmal C. These are quadruped robots, which means that they have four legs. Each leg has 3 degrees of freedoms - hip, shank and thigh. The position and velocities of these degrees of freedom for all 4 legs were recorded. This results in 24 features for each robot. Robots are generated while traversing an procedurally generated environment with different terrain types and traversal difficulty, as show in Figure 6.4. We only keep trajectories that correspond to a successful traversal.

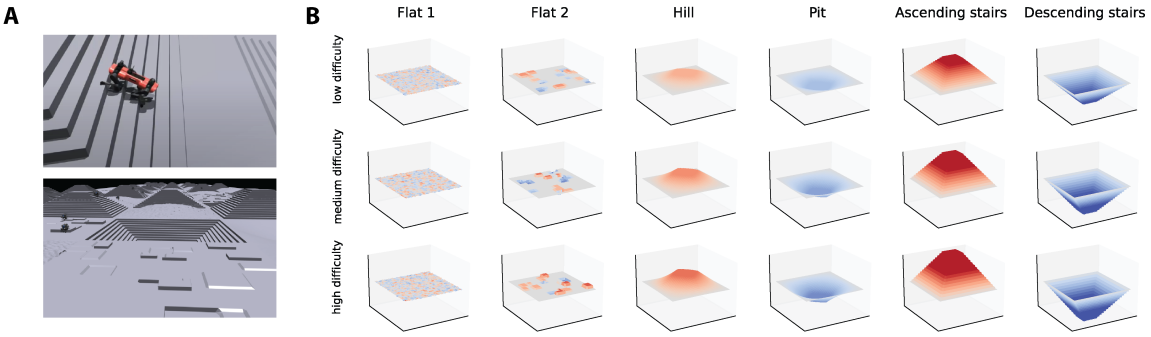


Figure 6.4: *Visualization of different simulated environments.* (A) Screenshots from the simulator showing a robot walking down some stairs, and a view of the terrain landscape. (B) Visualization of the different terrain sections, characterized by a terrain type and different levels of difficulty. Terrain are made more difficult to traverse by either making them more rough or have steeper slopes.

Tasks. To evaluate the representation quality of our model, we use multi-task probes that correspond to different long-term and short-term behavioral factors.

- Robot type: the robot can either be of type "ANYmal B" or "ANYmal C". These robots have the same degrees of freedom and tracked joints but differ by their morphology. This is a sequence-level task.
- Linear velocity: the command of the robot is a constant velocity vector. The amplitude of the velocity dictates how fast the robot is commanded to traverse the environment. A higher velocity would translate into more clumpsy and more risk-taking behavior. This is a sequence-level task.
- Terrain type: the environment is generated with multiple segments of five terrain

types that are categorized as: flat surfaces, pits, hills, ascending and descending stairs. This is a frame-level task.

- Terrain slope: the slope of the surface the robot is walking on. This is a frame-level task.
- Terrain difficulty: the different terrain segments have different difficulty levels based on terrain roughness or steepness of the surface. This is a frame-level task.

Why this dataset. Simulation-based data collection enables access to information that is generally inaccessible or hard to acquire in a real-world setting. Unlike noisy measurements coming from the camera-based feature extractor in the case of the mouse dataset, physics engines do not suffer from the problem of noise. Instead, they provide accurate ground-truth information about the creature and the world state free of charge. Access to such information is at times critical for scrutinizing the capabilities of the learning algorithms.

Visualizing differences between short-term and long-term embeddings

In Figure 6.5, we visualize how the short-term and long-term embeddings evolve over time, for a single sample sequence. We note a clear difference in the smoothness in the two timescales. In the short-term embeddings, we note a clear block structure corresponding to different blocks of behavior that span a few seconds, while in the long-term embeddings the representation is more stable over time. This suggests that, as expected, the bootstrapping objectives are forming representations with different levels of granularity.

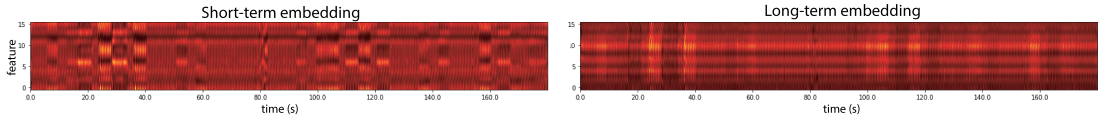


Figure 6.5: *Visualization of the short-term and long-term embeddings.* We visualize for a single sequence how the short-term and long-term embeddings evolve over time.

6.2.2 Experimental details: Mouse Triplet

Feature extraction

Each mouse in the arena is tracked using 12 anatomically defined keypoints. We process these keypoints to extract 36 different features characterizing each mouse individually, similar to [58]. We separate the keypoints into two different areas, the head and the body, for each we extract different measures of displacement, that we express in the frame of the mouse, i.e. these features are invariant to the pose of the mouse relative to the arena. These features include:

- Head linear velocity vector that we express using polar coordinates.
- Head angular velocity denoting the change in the heading direction in the arena.
- Body linear velocity vector that we express using polar coordinates.
- Body angular velocity denoting the change in the direction of the body with respect to the arena.
- Angular and linear velocities of the fore paws and the hind paws.
- Spine length change, depicting the expansion and contraction of the mouse's body.
- Angles formed by the tail with respect to the body.

We normalize all features before training. We also use cosine and sinus of the angles instead of the angles. During training, we did not use any form of augmentation.

Noise in the data. Because of errors in pose estimation and tracking, there are sometimes errors in the tracking data, notably some identity swap issues [74]. To address this, we simply zero out all of the corresponding features and flag the frame as invalid. A binary feature is also add to the input features indicating whether or

not the frame is valid. When predicting future actions, we only compute the error over windows in which at least 80% of the frames are valid.

Difference between Histogram of Actions and previous objectives

Our novel objective consists in predicting the future histogram of actions instead of predicting the future sequence of actions. In Figure 6.6, we show what the target is for a sample from the MABe Mouse Triplet dataset. Note that the time dimension is collapsed, blurring the exact unrolling of the future events, but preserving the set of values that these actions will sweep. Note that the loss (EMD) is applied for each action feature.

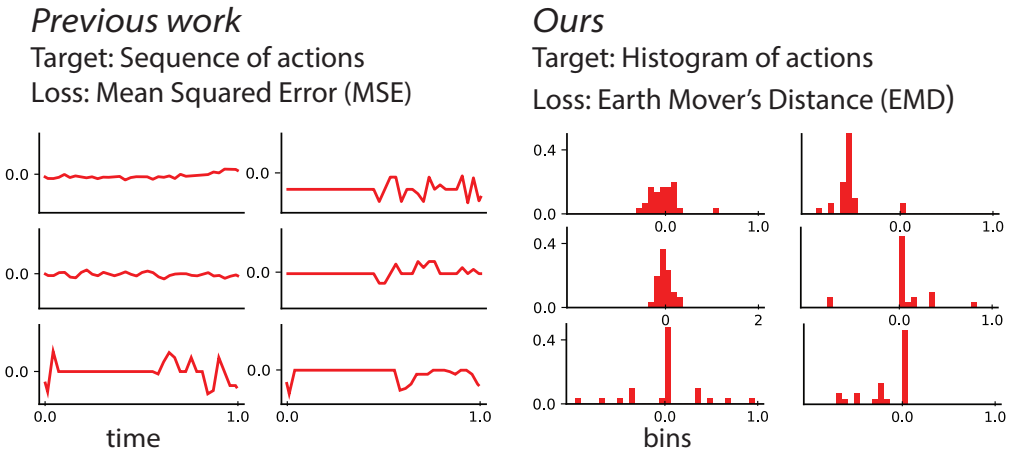


Figure 6.6: Prediction target for a sample of the MABe Mouse Triplet dataset.

We show that by relaxing our future prediction loss to a HoA loss, we benefit in terms of the representations that are learned by the model, especially for sequence-level (global tasks). Thus, in many ways, we show that directly forecasting, which is what many previous approaches have used for representation learning, can actually lead to representations that capture less of the task structure. While the model doesn't predict future timesteps directly, we can visualize the histogram prediction for the model and ground truth (Figure Figure 6.7).

Training details

Architecture. We use two TCNs [80]. Each TCN is built using multiple residual blocks, each residual block is composed of two convolutional layers, and use PReLU activation, dropout and weight normalization. All convolutions are dilated with a rate r, that increases after each residual block. The formula is r^i where i is the index of the residual block. The first TCN is the short-term encoder, which uses 4 blocks with output sizes $[64, 64, 32, 32]$ and a dilation rate $r = 2$. The second TCN is the long-term encoder, which uses 5 blocks with output size $[64, 64, 64, 32, 32]$ and a dilation rate $r = 4$. The output of both encoders are concatenated to form a 64d embedding. The predictor is a multi-layer perceptron (MLP) that has 4 hidden layers.

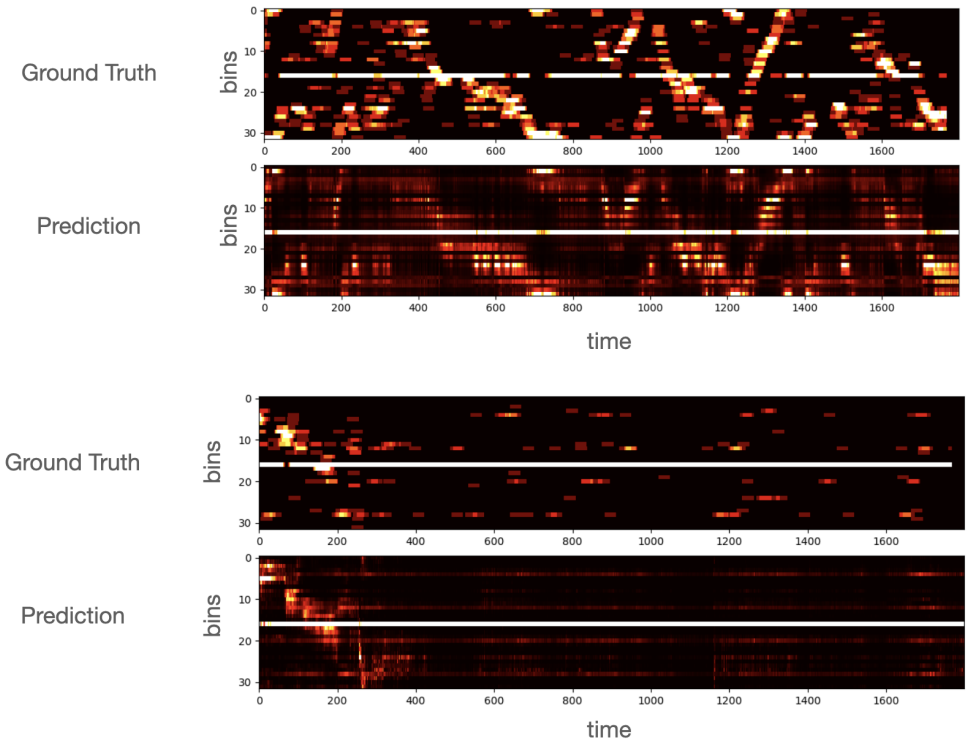


Figure 6.7: Visualization of histograms of future actions, for two random action features. For a timestamp t on the time axis, we show the 32 dimensional histogram of future actions, which is the target of prediction for BAMS.

Training. We train the model for 500 epochs using the Adam optimizer with a learning rate of 10^{-3} and weight decay $4 \cdot 10^{-5}$, we decrease the learning rate to 10^{-4} after 100 epochs. We use a batch size of 96, and compute the future histogram of action prediction error for each timestep t starting at 5 seconds after the start of each sequence, in order to allow the model to aggregate enough context. We set the learning rate of the predictors used for bootstrapping to be 10 times higher than the learning rate used for the rest of the weights.

Evaluation. During the development of the model (Figure 6.8), we test our model on the public test splits, and only look at the performance on the private set after finishing any hyperparameter tuning. We repeat the training and evaluation 5 times and report the average performance

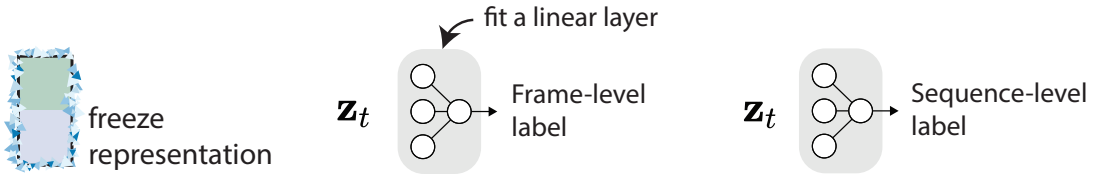


Figure 6.8: Linear evaluation protocol. The model is frozen, and for each task, a single linear layer is trained to predict the corresponding labels.

Additional ablations

In addition to those mentioned in the main text, we perform two additional ablations to BAMS (Table Table 6.3). In our first experiment, we removed the interaction loss from the model, meaning that the dynamics of each mouse are modeled completely independently from each other. The ablated model sees a small drop in performance, but continues to outperform all other methods on average.

BAMS in the inductive setting

The mouse triplet dataset (5336 sequences) has three different sets, a training set (1800 sequences), a private test set and a public test set. During training of the representation learning model, we can either pre-train on all of the available data (transductive setting) or on the training set only (inductive setting). During linear evaluation, the different linear layers are trained using labels from the training set and the performance is reported on the public test set (during the challenge) and then on the private test set (to rank models).

We train BAMS in the inductive setting and report the performance in Table 6.3. We find that even when BAMS is trained with approximately one third of the data, the drop in performance is modest. More importantly, BAMS preserves its ranking compared to other methods, and still achieves state-of-the-art performance.

Table 6.3: *Linear readouts of mouse behavior.* The best-performing models are those with low MSE scores and high F1-scores.

	Sequence-level subtasks				*Frame-level subtasks*								
Model	Day (↓)	Time (↓)	Strain	Lights	Approach	Chase	Close	Contact	Huddle	O/E	O/G	O/O	Watching
PCA	0.09416	0.09445	51.60	54.65	0.86	0.14	49.27	37.87	12.71	0.21	0.60	0.53	6.65
TVAE	0.09403	0.09442	52.98	56.80	1.07	0.45	59.33	44.77	21.96	0.27	0.83	0.62	10.20
T-BERT	0.09262	0.09276	78.63	68.84	1.80	0.87	70.22	55.84	30.24	0.51	1.40	1.12	17.27
TS2Vec	0.09380	0.09422	57.12	65.60	1.29	0.66	59.53	46.13	24.74	0.35	1.09	0.74	12.37
T-Perceiver	0.09322	0.09323	69.81	69.68	1.57	1.27	60.84	47.81	28.32	0.41	1.16	0.86	16.42
T-GPT	0.09269	0.09384	64.45	65.39	1.73	0.64	69.05	55.78	23.80	0.46	1.12	1.05	17.86
T-PointNet	0.09275	0.09320	66.01	67.15	2.56	4.57	70.68	55.96	21.23	0.84	2.79	2.32	15.61
BAMS - no interaction	0.09164	0.09154	83.47	71.23	2.55	2.03	63.63	50.97	31.15	0.58	1.47	1.37	15.10
BAMS - inductive	0.09112	0.09132	83.44	70.39	2.62	1.40	65.98	52.39	31.08	0.60	1.54	1.40	18.14
BAMS - transductive	0.09094	0.08989	88.23	72.00	2.74	1.89	67.22	53.43	31.43	0.59	1.61	1.57	18.15

Table 6.4: *TS2Vec Linear readouts of mouse behavior.*

	Sequence-level subtasks				*Frame-level subtasks*								
Model	Day (↓)	Time (↓)	Strain	Lights	Approach	Chase	Close	Contact	Huddle	O/E	O/G	O/O	Watching
TS2Vec-I	0.09380	0.09422	57.12	65.60	1.29	0.66	59.53	46.13	24.74	0.35	1.09	0.74	12.37
TS2Vec-T	0.09882	1.0252	45.82	46.69	0.72	0.14	45.19	34.93	9.38	0.186	0.38	0.38	05.31
TS2Vec-IT	0.09846	1.01646	46.67	44.28	0.67	0.13	44.56	33.87	9.79	0.178	0.42	0.42	04.58

Notes on TS2Vec experiments

TS2Vec [86] employs two types of contrastive losses to learn representations. The first of these losses is an instance contrastive loss which contrasts a sequence with all other

sequences in a batch which are treated as negative examples, while two subsequences extracted from the same sequence are treated as positive examples. The second loss is a temporal contrastive loss which acts along a single time series. Temporal representations of nearby time points are taken as positive examples, while the rest of the time points within the same sequence are taken as negative examples. The results for the three versions of TS2vec, namely TS2Vec-I, which uses only instance contrastive loss, TS2Vec-T, which uses only temporal contrastive loss,and TS2Vec IT, which uses both instance and temporal contrastive losses, are listed in Table 6.4. Our TS2Vec experiments on the mouse dataset showed that using temporal contrastive loss resulted in worse performance across all tasks as compared to only using instance contrastive loss. For this reason, we only report results for TS2vec that only employs instance contrastive loss.

We note that for both TS2Vec and TS2Vec-IT, we ran into out-of-memory errors when creating instance-level or global contrast. Contrastive learning methods usually incur high computational costs, we find that our method, which doesn't rely on negative examples, can scale better when working with longer sequences and larger datasets.

6.3 Learning invariances in neural population activity

6.3.1 Algorithm

6.3.2 Mining: Implementation details

At a given training iteration, for every sample in the batch, we mine for views in the same pool of candidates of size L. Depending on the type of data, the mining for a given sample can be restricted to a subset of that pool of candidates.

Image datasets. When training `MYOW` on images, we use two different dataloaders. The first is the main dataloader that creates batches of size B, the second dataloader

Algorithm 1: `Mine Your Own vieW - MYOW`

`Dataset` $\mathcal{D}$; `online network` $f_\theta, g_\theta, h_\theta$; `target network` f_ξ, g_ξ, h_ξ; `dual predictors` q_θ, r_θ; `learning rate` η; `momentum` τ; `mining weight` λ; `batch size` B; `pool batch size` L.

$\xi \leftarrow \theta$ **while** *not converging* **do**

  `Augment views Fetch a mini-batch` $\{\mathbf{s}_i\}_B$ `from` $\mathcal{D}$;

  for $i \in \{1...B\}$ *(in parallel)* **do**

    `Draw functions:` $t \sim \mathcal{T}, t' \sim \mathcal{T}$;

    $\mathbf{x}_i = t(\mathbf{s}_i),\ \mathbf{x}'_i = t'(\mathbf{s}_i)$;

    $\mathbf{z}_i = g_\theta(f_\theta(\mathbf{x}_i));\ \mathbf{z}'_i = g_\xi(f_\xi(\mathbf{x}'_i))$;

    $\mathbf{u}_i = g_\xi(f_\xi(\mathbf{x}_i));\ \mathbf{u}'_i = g_\theta(f_\theta(\mathbf{x}'_i))$;

  end

  `Mine views Fetch a mini-batch` $\{\mathbf{c}_j\}_L$ `from` $\mathcal{D}$;

  for $j \in \{1...L\}$ *(in parallel)* **do**

    `Draw function:` $t \sim \mathcal{T}_m$

    $\mathbf{x}_{c,j} = t(\mathbf{c}_j);\ \mathbf{y}'_{c,j} = f_\xi(\mathbf{x}_{c,j})$;

  end

  `Let` $\mathcal{S} = \{\mathbf{y}'_{c,j}\}_{j=1}^{L}$;

  for $i \in \{1...B\}$ *(in parallel)* **do**

    `Draw function:` $t \sim \mathcal{T}_m$;

    $\mathbf{x}_{m,i} = t(\mathbf{s}_i);\ \mathbf{y}_{m,i} = f_\theta(\mathbf{x}_{m,i})$;

    `Find` $\mathcal{N}_k(\mathbf{y}_{m,i})$, `the` k-NNs `of` $\mathbf{y}_{m,i}$ in $\mathcal{S}$;

    `Randomly select` $\mathbf{y}'_{m,i}$ `from` $\mathcal{N}_k(\mathbf{y}_{m,i})$;

    $\boldsymbol{v}_i = h_\theta\left(g_\theta\left(\mathbf{y}_{m,i}\right)\right); \boldsymbol{v}'_i = h_\xi(g_\xi(\mathbf{y}'_{m,i}))$;

  end

  `Update parameters` $\mathcal{L} = \sum_i d(q_\theta(\mathbf{z}_i), \mathbf{z}'_i) + d(q_\theta(\mathbf{u}'_i), \mathbf{u}_i) + \lambda d(r_\theta(\boldsymbol{v}_i), \boldsymbol{v}'_i)$;

  $\theta \leftarrow \text{Optimizer}\theta, \mathcal{L}/B, \eta$;

  $\xi \leftarrow \tau\xi + (1-\tau)\theta$;

end

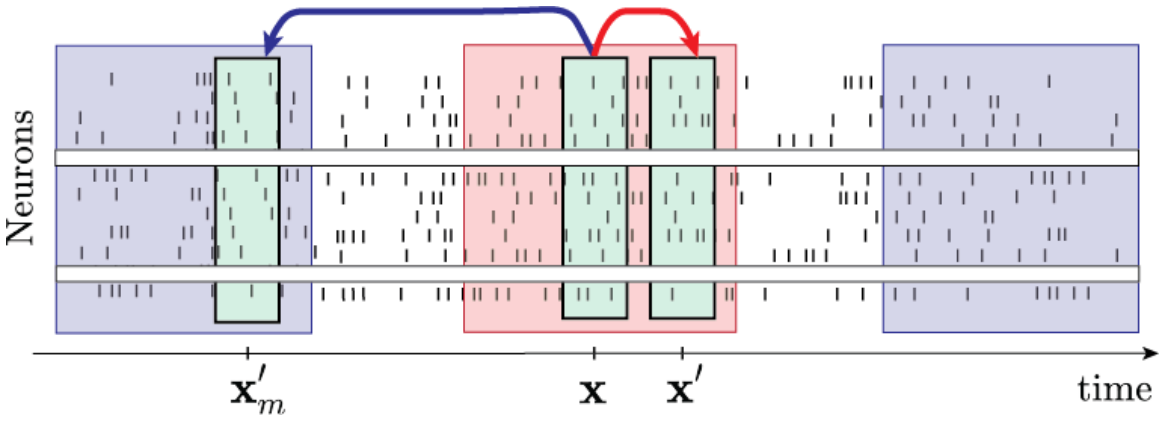


Figure 6.9: *Visualization of augmentations used for neural activity.* Within a small local window around each anchor sample, we consider the nearby samples (red) to be potential positive examples. Outside of a safe zone, we can label more distant samples (blue) as either negative examples (in contrastive learning) or we can also use these points as candidate views to mine from (in `MYOW`). Randomized dropout is illustrated via white bars corresponding to the dropping of the same neurons in all three views.

is independent from the first and is used to sample candidates, and thus has a batch size of L. When $L > B$, the second dataloader consumes the dataset before the end of the training epoch, in this case we simply reset the candidate dataloader as many times as needed.

Neural datasets. When training `MYOW` on neural datasets, or temporal datasets in general, we restrict mining for a given sample to candidates that are temporally farther in time, as illustrated in Figure 6.9. Implementation-wise, we use a global pool of candidates of size L for simplicity, then when computing the distance matrix used to determine the k-nearest neighbors, we mask out the undesired correspondences in the matrix.

6.3.3 Experimental setup: Image datasets

Notation Let MLP(i, h, o) be a linear layer with input size i and output size h, followed by batch normalization, rectified linear units (ReLU) and a linear layer of output size o. Like in [83], we use these multi-layer perceptrons (MLPs) of depth 2 for projectors and predictors.

Architecture. We use the CIFAR variant of ResNet-18 as our backbone [111]. The representation $\mathbf{y}$ corresponds to the output of the final average pool layer, which has a feature dimension of 512. We use MLP$(512, 4096, 256)$ for the first projector g_θ and MLP$(256, 4096, 256)$ for its corresponding predictor q_θ. For the pair of projector/predictor (h_θ/r_θ) dedicated to mined views, we use MLP$(256, 4096, 256)$ and MLP$(256, 4096, 256)$, respectively.

Class of transformations. During training, we generate augmented views using the following transformations $(\mathcal{T})$ [136, 137]:

- Random cropping: Images are resized to 32x32 using bicubic interpolation, with random area ratio between 0.2 and 1.0, and a random aspect ratio between 3/4 and 4/3.
- Random horizontal flip: the image is fliped left to right with a probability of 0.5.
- Color jittering: the brightness, contrast, saturation, and hue of the image are randomly changed with strengths of $(0.4, 0.4, 0.4, 0.1)$ with a probability of 0.8.
- Color dropping: the image is converted to gray scale with a probability of 0.1.

When mining, we only use random cropping with a random area ratio between 0.8 and 1.0 to augment views $(\mathcal{T}')$.

Training. We use the SGD optimizer with a learning rate of 0.03, a momentum of 0.9 and weight decay of $5 * 10^{-4}$. After a linear warmup period of 10 epochs, the learning rate is decayed following a cosine decay scheduler. The exponential moving average parameter τ is also decayed from 0.996 to 1. following a cosine decay scheduler. We train `MYOW` for 800 epochs and use a batch size of $B = 512$, as well as a pool batch size of $L = 512$, and $k = 1$. We use a mining weight of $\lambda = 0.1$ linearly ramped-up for 10 epochs. `BYOL` is trained using the same relevant hyperparameters. In our experiments, we use the same random seeds for both `MYOW` and `BYOL`.

Evaluation Protocol: Following the evaluation procedures described in [111, 83],

we train a linear classifier on top of the frozen representation of the encoder network and report the accuracy on the test sets (We use the public train/test split for both CIFAR datasets). The linear layer is trained without augmentations for 200 epochs, with an SGD optimizer with a learning rate of 0.4 decayed by a factor of 10 at 140 and 190 epochs.

6.3.4 Experimental details: Neural data

Application 1: Decoding movements from motor cortex

Details on neural and behavioral datasets in movement decoding task. Neural and behavioral data were collected from two rhesus macaque monkeys (Chewie, Mihi). Both individuals performed a standard delayed center-out movement paradigm (reaching experiment). The subjects were seated in a primate chair and grasped a handle of a custom 2-D planar manipulandum that controlled a computer cursor on a screen. In the first dataset from Chewie, the individual began each trial by moving to a 2 x 2 x 2 cm target in the center of the workspace, and was instructed to hold for 500-1500 ms before another 2 cm target was randomly displayed in one of eight outer positions regularly spaced at a radial distance of 8 cm. For Mihi, this is followed by another variable delay period of 500 to 1500 ms to plan the movement before an auditory 'Go' cue. The sessions with Chewie omitted this instructed delay period and the 'Go' cue was provided when the outer target appeared. Both individuals were required to reach to the target within 1000-1300 ms and hold within it for 500 ms to receive an auditory success tone and a liquid reward.

Both individuals were surgically implanted a 100- electrode array (Blackrock Microsystems, Salt Lake City) in their primary motor cortex (M1). To record the spiking activity of single neural units, threshold crossings of six times the root-mean square (RMS) noise on each of the 96 recording channels are initially recorded. After each session, the neural waveform data was sorted using Offline Sorter (Plexon, Inc,

Dallas, TX) to identify single neurons and discarded all waveforms believed to be multi-unit activity.

Data is only recorded when the primate is performing the reaching task, we note such instance a "trial". We split the trials time-wise, using a 70/10/20 ratio, to obtain our training, validation and test sets. The temporal splits gives us a better estimate of the prospective prediction compared to a random split [160]. The activity of individual neurons was binned (100 ms intervals) to produce firing rates for roughly 150 neurons across two days.

Class of transformations. During training, we generate augmented views and mined views using the following transformations ($\mathcal{T} = \mathcal{T}'$):

- Temporal Jitter: a sample within 200ms is used as a positive example.
- Dropout: mask out neurons with a probability uniformly sampled between 0. and 0.2.
- Noise: add gaussian noise with standard deviation of 1.5, with a probability of 0.5.
- Pepper or Sparse additive noise: increase the firing rate of a neuron by a 1.5 constant with a probability of 0.3. This augmentation is applied on the sample with a probability of 0.5.

Because these datasets correspond to a collection of trials, we restrict mining to candidates that are in different trials from the anchor sample.

Network Architecture. For the encoder, we use an MLP which is 4 blocks deep. Each block consists of a linear layer with output size 64 followed by batch normalization (BN) and rectified linear units (ReLU). The final layer has an output size of 32 and no BN or activation. We don't use projectors, predictor q_θ used for augmented views is MLP$(32, 128, 32)$, and predictor r_θ used for mined views is MLP$(32, 128, 32)$.

Training. We use the AdamW optimizer with a learning rate of 0.02 and weight

decay of $2 * 10^{-5}$. After a linear warmup period of 100 epochs, the learning rate is decayed following a cosine decay scheduler. The exponential moving average parameter τ is also decayed from 0.98 to 1. following a cosine decay scheduler. We train `MYOW` for 1000 epochs and use a batch size of $B = 512$, as well as a pool batch size of $L = 1024$, and $k = 5$. We use a mining weight of $\lambda = 1.$ linearly ramped-up for 10 epochs. `BYOL` is trained using the same relevant hyperparameters.

Reach direction prediction task. The downstream task we use to evaluate the learned representation, is the prediction of the reach direction during movement. There are 8 possible reach direction in total. Unlike most classification tasks, there is an inherent cyclic ordering between the different classes. Thus, we estimate the angles corresponding to each reach direction, and evaluate their cosine and sine. The linear layer outputs a 2d vector $[x, y]$ that predicts $[\cos\theta_r, \sin\theta_r]$. We train the network using a mean-squared error loss. Once the network is trained, to readout out the predicted reach direction label, we use the following formula:

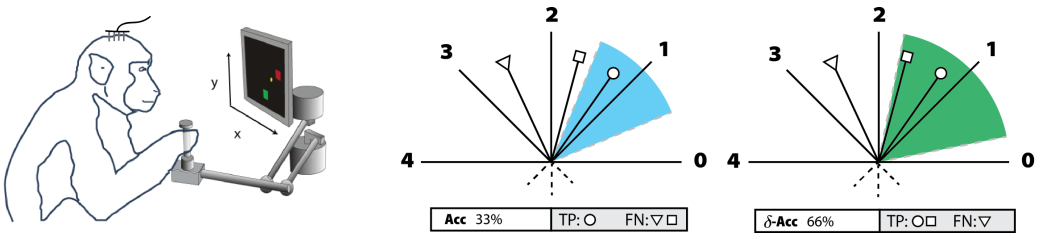
$$l_{\text{predicted}} = \lfloor \frac{4}{\pi}(\text{atan2}(y, x) \bmod 2\pi) \rceil \tag{6.1}$$

Evaluation Procedure. We train a linear classifier on top of the frozen representation of the encoder network and report the accuracy on the test sets. The linear layer is trained for 100 epochs using the AdamW optimizer with a learning rate of 0.01. We sweep over 20 values of the weight decay $\{2^{10}, 2^8, 2^6, \ldots, 2^6, 2^8, 2^{10}\}$ on the valudation set, and report the accuracies of the best validation hyperparameter on the test set.

More specifically, we report two different metrics that are computed over the validation set. The Accuracy is the conventional classification accuracy that is obtained when assigning the predicted reach angle to the closest corresponding reach direction. The second metric, , is obtained when considering that a prediction is a true positive if it is within a slightly larger window around the true reach direction

(an analogy to top-k metrics). (Fig Figure 6.10-b).

$$\text{TP}_{\text{Acc}} = |\frac{4}{\pi}(\text{atan2}(y, x) \bmod 2\pi) - l| < 1 \qquad \text{TP}_{\delta-\text{Acc}} = |\frac{4}{\pi}(\text{atan2}(y, x) \bmod 2\pi) - l| < 1.5$$



(a) (b)

Figure 6.10: *Reach direction prediction task.* (a) Sketch of primate performing reaching task. (b) Illustration depicting how the accuracy and δ-accuracy are computed. The three points have reach direction 1 as their ground truth. TP is true positive and FN is false negative. The highlighted areas correspond to the area a point should fall in to be considered a true positive and be counted towards the corresponding accuracy.

Application 2: Decoding sleep states from rodent cortex

Details on neural and behavioral datasets in arousal state decoding. Extracellular single unit spiking was collected from chronically implanted, freely behaving animals. Tetrode arrays were implanted without drives into mouse CA1 (C57BL/6) and rat V1 (Long Evans). Following recovery, neural data were recorded at 25 kHz continuously during free behavior. Raw data were processed and clustered using standard pipelines. Data was bandpassed (500-10,000 Hz) and clustered using MountainSort [161, 162]. Single units were identified in the clustering output via XGBoost.

Arousal state was scored using standard polysomnographic methods. Local field potentials (LFP) from 8/64 channels were averaged together, lowpassed (250 Hz), and downsampled. Video (15 fps) was processed using a CNN [98] to track animal position and movement. Trained human scorers evaluated the LFP power spectral density and integral of animal movement to evaluate waking, NREM and REM sleep.

We split the 12 hour block of data temporally using an 70/10/20 ratio, to obtain our training, validation and test sets. The activity of individual neurons was binned (4s intervals) to produce firing rates for roughly 40 and 120 neurons from CA1 and V1, respectively.

Training. We use the same hyperparameters as for the monkey datasets, except that the representation size is larger (64), and the temporal augmentations are different. With temporal jitter, we consider any two samples that are at most 12s apart to be positive examples and when mining we restrict the candidates to be at least 30min before or after the anchor sample.

Arousal state prediction task. We train a linear classifier on top of the frozen representation of the encoder network to predict the arousal state.

6.3.5 Is MYOW worth the extra computational load?

In one iteration, MYOW receives 3 batches worth of views, compared to 2 for BYOL. Thus, there is a possibility that MYOW performs better than BYOL simply because of the higher effective batch size used during training. To rule this possibility out, we try both training BYOL for 50% more epochs and training BYOL using a 50% bigger batch size, and report the results in Table 6.5. We show that the improvements we find through with MYOW go beyond extra training time.

Table 6.5: *Training BYOL with adjusted batch size and number of epochs.* We report the linear evaluation accuracies on CIFAR-10 using ResNet-18.

	Batch size	Number of epochs	Accuracy
BYOL	512	800	91.71
BYOL	512	1200	91.75
BYOL	768	800	91.65
MYOW	512	800	92.10

When we examine the accuracy curves during training (Figure 6.11), we find that MYOW surpasses the final accuracy of BYOL after only 300 epochs of training. Thus, in

the case of this dataset, we can justify the extra computational load that comes with using MYOW, as it yields better results early on in training.

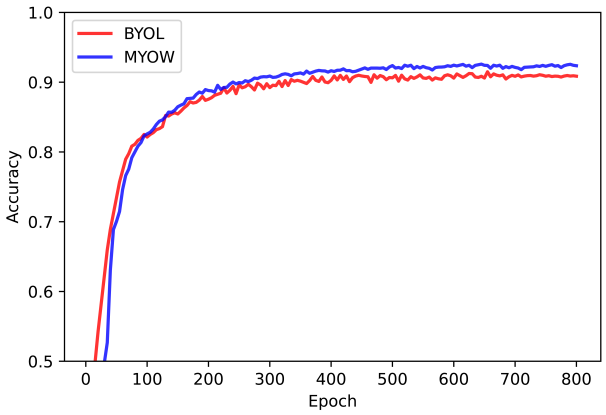


Figure 6.11: *Accuracy under linear evaluation, CIFAR10, ResNet18.* BYOL (bottom), MYOW (top).

6.3.6 What makes for good mined views?

In Table 6.6, we compare the outcomes of using the online representations of the candidates compared to their target representations when looking for the k-nn of the online representation of the anchor sample. We find that both strategies yield similar results while mining in the target is less computationally expensive.

Table 6.6: *Mining in online versus. target space.* We report the linear evaluation accuracies on CIFAR-10 using ResNet-18, as well as an approximation of the computational load factor with BYOL as the baseline.

	Mining in	Computational factor	Accuracy
BYOL	-	1.00	91.71
MYOW	online	1.75	92.13
MYOW	target	1.50	92.10

We analyse the views that are being mined when training MYOW on CIFAR-10. In Figure 6.12, we show a random collection of views paired during mining.

MYOW relies on mining views that are semantically similar, but it is not clear how robust MYOW is to "bad" mined views. While we are not able to give a definitive answer

Figure 6.12: *Examples of views mined by* ***MYOW***. We visualize the views mined by MYOW during training on the CIFAR-10 dataset at epoch 400.

to this question, we find that even when certain mined views have a different class from the anchor samples, MYOW still yields competitive results. In Figure 6.13, we look at the mining class accuracy, defined as the percentage of mined views that share the same class as their anchor samples, and find that the accuracy steadily increases during training and that the relatively low accuracy at the beginning of training does not hinder the performance of MYOW. The mining class accuracy gives us a better understanding of the mining, but it is not a reflection of the goodness of the mining, as we do not know what makes for a good mined view and whether a inter-class mined views could be "good". We also visualize, in Figure 6.14, the mining class confusion matrices at epochs 100 and 700 of training.

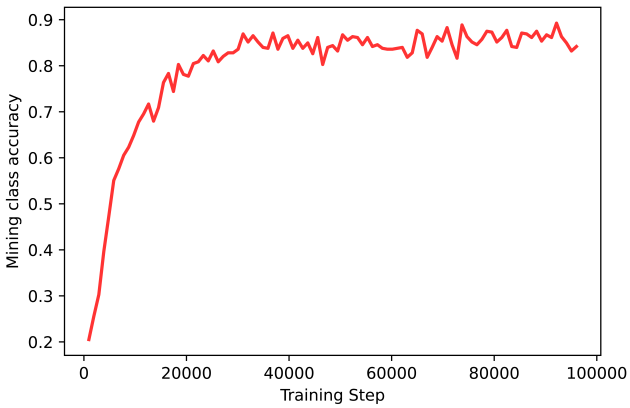


Figure 6.13: *Mining class accuracy during training.* This metric is reported on CIFAR-10 using ResNet-18.

6.3.7 Ablation on the projector

In Table 6.7 and Table 6.8, we report the results of MYOW on the MNIST and CIFAR-10 datasets for different architectures used for incorporating mined views into our objective: cascaded projectors (used in MYOW), parallel projectors and single projector For MNIST, we show the results for two different settings, weak augmentation (Crop only) and strong augmentation (All). Overall, we find that separating the projection

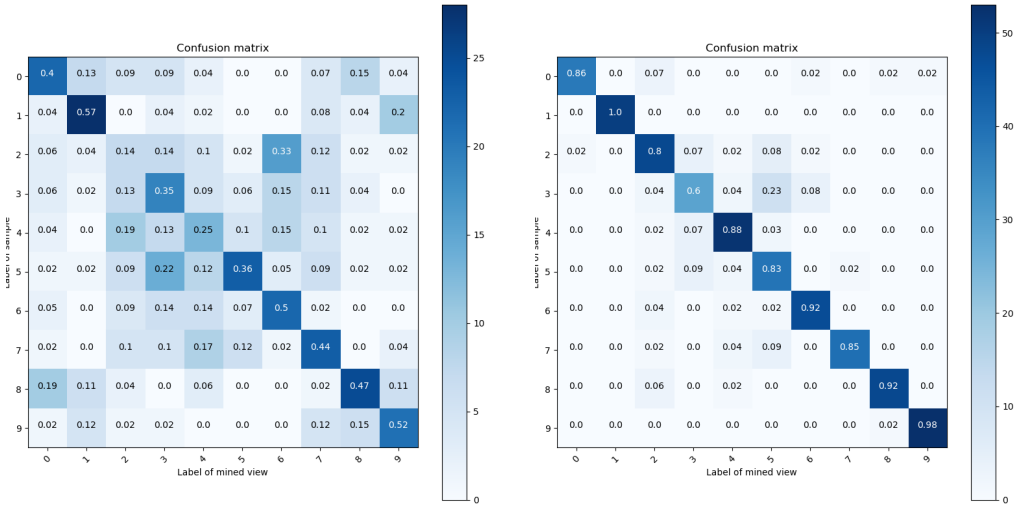


Figure 6.14: *Mining class confusion matrices at different stages of learning.* We compute the confusion matrix at epochs 100 (right) and (700) when training on CIFAR-10.

Table 6.7: *Comparing different projector architectures for incorporating mined views.* MNIST classification accuracy (in %) with `MYOW` for different architectures.

Arch	Dimension	MNIST	
		Crop only	All
Cascaded	16	**99.20**	**99.33**
Cascaded	128	98.09	98.80
Parallel	16	96.33	98.71
Parallel	128	97.75	98.12
Single	16	97.13	97.48
Single	128	98.75	98.31

spaces for augmented and mined views is better, with the cascading yielding the best results.

6.3.8 Ablation on the class of transformations

We study how the choice of the set of transformations used in the mining process, impacts the quality of the representation. In Table 6.9, we report the accuracies under linear evaluation when we use different classes of transformation $\mathcal{T}'$.

Table 6.8: *Comparing different projector architectures for incorporating mined views.* CIFAR-10 classification accuracy (in %) with MYOW for different architectures.

Arch	CIFAR-10
Cascaded projectors	**92.10**
Parallel projectors	92.01
Single projector	91.84

Table 6.9: *Class of transformations for mined views.* We report the accuracies under linear evaluation of MYOW trained on CIFAR-10 using ResNet-18, for different classes of transformation $\mathcal{T}'$

Crop	Flip	Color jitter	CIFAR-10
✓$(0.2 - 1.0)$	✓	✓	91.63
✓$(0.8 - 1.0)$			92.10
			92.08

6.3.9 Gaining insights into across-sample prediction

Based upon our experiments on neural data, we conjectured that the diversity introduced by MYOW makes it possible to learn effectively, even when the augmentations provided to the network are too local to drive learning in BYOL. We thus designed an experiment using the dSprites dataset [163], as it allows control over the generation of data over multiple latent positions.

The dSprites dataset is comprised of a total of 737,280 images. Each image has an associated shape, orientation, scale and 2D position. Each one of these latent variables has a finite number of possible values because of the procedural nature of the dataset. To generate the downsampled training sets used in our experiment, we uniformly sample 50% of the orientation latent values as well as 50% of the scale latent values, and only consider the corresponding images, thus effectively creating holes in the latent manifold. The dataset is further downsampled at a given rate r to generate the train set, the remaining images form the test set. The size of the train set is effectively $0.25 * r$ that of the entire dataset. In our experiment, we generate training sets that are 30%, 15% and 7.5% the size of the dataset.

When we train BYOL and MYOW on a sufficiently dense sampling of the latent positions (30%), we observe that both models can classify on unseen latent positions with nearly 100% accuracy (Figure 6.15). However, when we consider the undersampled condition (7.5%), BYOL fails to generalize to the unseen positions, resulting in a low accuracy of around 60%. In contrast, MYOW maintains a high accuracy of 94% despite the limited training data. These findings suggest that in settings where the data manifold is sparsely sampled, MYOW provides a way to build predictions across different but similar data samples.

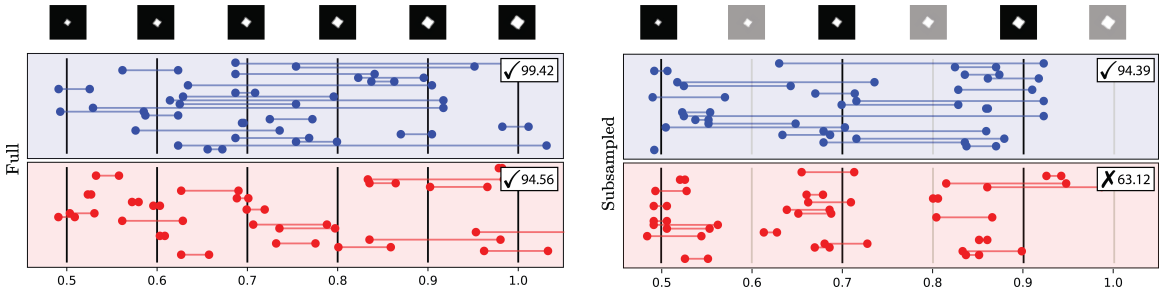


Figure 6.15: *Understanding predictive learning when augmentations are too local.* Each segment represents a pair of views (red for augmented, blue for mined) of the corresponding latent scale (x-axis). The vertical lines represent the original scales of samples pre-augmentation. We examine the case where we have access to the full dataset (left) and when we have only half of the latent positions (3/6) and 7.5% of the remaining samples (right).

6.3.10 Augmentations for spiking neural data

Temporal jitter. As in previous work in temporal contrastive learning [127, 153, 154, 155, 156], we can use nearby samples as positive examples for one another.

Randomized dropout. When working with neural data, we consider randomized dropout [164] as an augmentation. The dropout rate is uniformly sampled between $p_{\min}$ and $p_{\max}$.

Gaussian noise. Random Gaussian noise with mean 0 and standard deviation 1.5 is applied before normalizing the firing rates.

Random pepper. In contrast to dropout, applying random pepper consists of

Table 6.11: *How augmentations impact our ability to decode sleep and wake states accurately.* To understand how different augmentations impact the representations obtained with `BYOL` and `MYOW` for the two datasets labeled Sleep, we computed the F1-score for different classes of augmentations in two brain areas.

	TS	RDrop	F1-score	
			Rat-V1	Mouse-CA1
`BYOL`	✓		68.66	87.73
		✓	79.31	88.84
	✓	✓	85.42	93.24
`MYOW`	✓		72.13	90.01
		✓	85.60	83.33
	✓	✓	88.01	93.70

randomly activating neurons. Similar to the dropout probability, a pepper probability is used to specify the probability of activating a neuron. The activation consists in adding a constant to the firing rate.

In Table 6.10, we show how different augmentations impact neural datasets not detailed in the main text. The findings are echoed through all monkey datasets.

Table 6.10: *How augmentations impact our ability to decode movements accurately.* To understand how different augmentations impact the representations obtained with `BYOL` and `MYOW` for all four datasets, we computed the Accuracy in our reach direction prediction task when we apply a given set of transformations.

	TJ	Drop	Noise	Pepper	Accuracy			
					Chewie-1	Chewie-2	Mihi-1	Mihi-2
`BYOL`	✓				41.75	40.83	43.98	44.10
		✓	✓	✓	55.70	49.37	47.61	43.12
	✓	✓			61.39	56.48	59.53	58.37
	✓	✓	✓	✓	63.80	57.17	59.50	60.82
`MYOW`	✓				46.61	42.91	42.08	44.13
		✓	✓	✓	53.15	46.17	51.44	48.72
	✓	✓			67.97	58.21	68.93	63.90
	✓	✓	✓	✓	70.41	60.95	70.48	64.35

In Table 6.11, we show the impact of both temporal shift and dropout on the performance on rodent datasets. Here, we also find that both components are important to achieving good performance.

6.3.11 Visualization of the latent neural space

In Figure 6.16, we provide the visualizations of the latent spaces for all four monkey reach dataset and can identify a common pattern in the structure uncovered by the different methods.

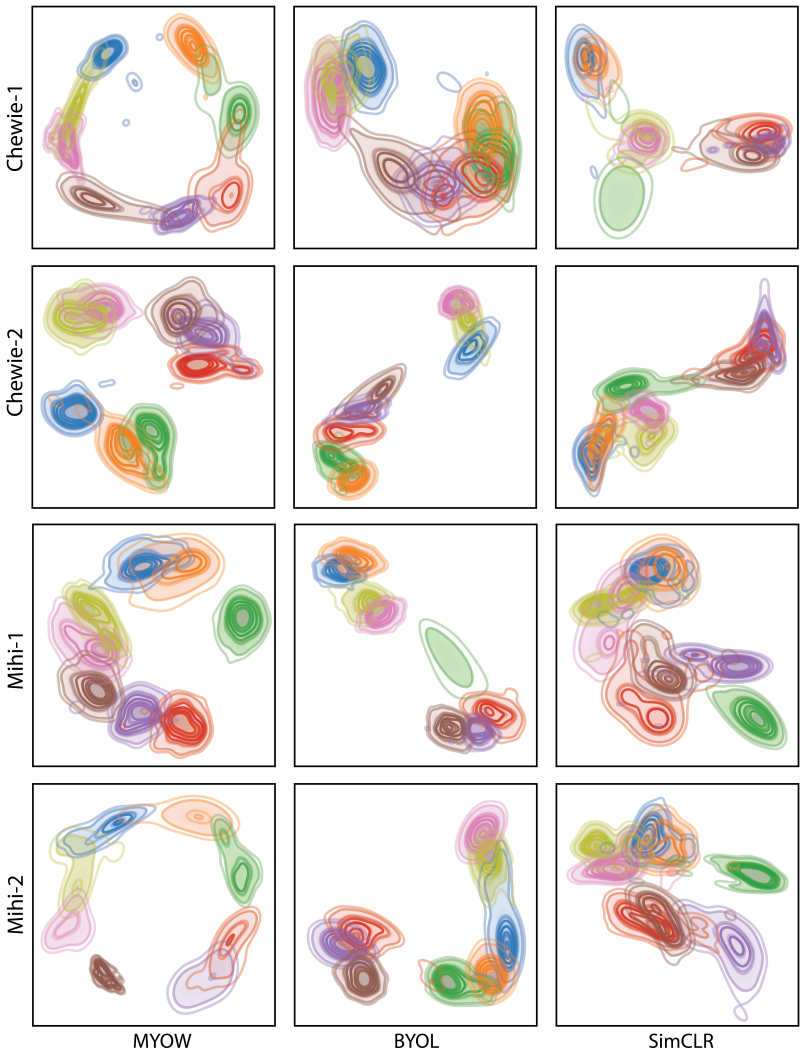


Figure 6.16: *Visualization of the learned representation.* Using t-SNE, we visualize the representation spaces when training `MYOW`, `BYOL` and `SimCLR` on all four monkey reach datasets.

REFERENCES

[1] A. Radford, K. Narasimhan, T. Salimans, I. Sutskever, *et al.*, "Improving language understanding by generative pre-training," 2018.

[2] A. Chowdhery *et al.*, "Palm: Scaling language modeling with pathways," *arXiv preprint arXiv:2204.02311*, 2022.

[3] M. Dehghani *et al.*, "Scaling vision transformers to 22 billion parameters," *arXiv preprint arXiv:2302.05442*, 2023.

[4] A. Radford, J. W. Kim, T. Xu, G. Brockman, C. McLeavey, and I. Sutskever, "Robust speech recognition via large-scale weak supervision," *arXiv preprint arXiv:2212.04356*, 2022.

[5] R. Bommasani *et al.*, "On the opportunities and risks of foundation models," *arXiv preprint arXiv:2108.07258*, 2021.

[6] J. Kaplan *et al.*, "Scaling laws for neural language models," *arXiv preprint arXiv:2001.08361*, 2020.

[7] A. E. Urai, B. Doiron, A. M. Leifer, and A. K. Churchland, "Large-scale neural recordings call for new insights to link brain and behavior," *Nature neuroscience*, vol. 25, no. 1, pp. 11–19, 2022.

[8] C. Hurwitz, N. Kudryashova, A. Onken, and M. H. Hennig, "Building population models for large-scale neural recordings: Opportunities and pitfalls," *Current Opinion in Neurobiology*, vol. 70, pp. 64–73, 2021, Computational Neuroscience.

[9] M. Dabagia, K. P. Kording, and E. L. Dyer, "Aligning latent representations of neural activity," *Nature Biomedical Engineering*, pp. 1–7, 2022.

[10] C. Pandarinath *et al.*, "Latent factors and dynamics in motor cortex and their application to brain–machine interfaces," *Journal of Neuroscience*, vol. 38, no. 44, pp. 9390–9401, 2018.

[11] W. M. Grill, S. E. Norman, and R. V. Bellamkonda, "Implanted neural interfaces: Biochallenges and engineered solutions," *Annual Review of Biomedical Engineering*, vol. 11, pp. 1–24, 2009.

[12] J. Jude, M. Perich, L. Miller, and M. Hennig, "Robust alignment of cross-session recordings of neural population activity by behaviour via unsupervised domain adaptation," in *Proceedings of the 39th International Conference on Machine Learning*, K. Chaudhuri, S. Jegelka, L. Song, C. Szepesvari, G. Niu, and S.

Sabato, Eds., ser. Proceedings of Machine Learning Research, vol. 162, PMLR, 17–23 Jul 2022, pp. 10 462–10 475.

[13] A. Jaegle *et al.*, "Perceiver io: A general architecture for structured inputs & outputs," *arXiv preprint arXiv:2107.14795*, 2021.

[14] A. Vaswani *et al.*, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[15] T. Brown *et al.*, "Language models are few-shot learners," *Advances in neural information processing systems*, vol. 33, pp. 1877–1901, 2020.

[16] H. G. Rey, C. Pedreira, and R. Q. Quiroga, "Past, present and future of spike sorting techniques," *Brain research bulletin*, vol. 119, pp. 106–117, 2015.

[17] M. Pachitariu, N. A. Steinmetz, S. N. Kadir, M. Carandini, and K. D. Harris, "Fast and accurate spike sorting of high-channel count probes with kilosort," *Advances in neural information processing systems*, vol. 29, 2016.

[18] A. Jaegle, F. Gimeno, A. Brock, O. Vinyals, A. Zisserman, and J. Carreira, "Perceiver: General perception with iterative attention," in *International conference on machine learning*, PMLR, 2021, pp. 4651–4664.

[19] J. Su, Y. Lu, S. Pan, A. Murtadha, B. Wen, and Y. Liu, "Roformer: Enhanced transformer with rotary position embedding," *arXiv preprint arXiv:2104.09864*, 2021.

[20] M. G. Perich, J. A. Gallego, and L. E. Miller, "A neural population mechanism for rapid learning," *Neuron*, vol. 100, no. 4, pp. 964–976, 2018.

[21] M. M. Churchland *et al.*, "Neural population dynamics during reaching," *Nature*, vol. 487, no. 7405, p. 51, 2012.

[22] J. G. Makin, J. E. O'Doherty, M. M. Cardoso, and P. N. Sabes, "Superior arm-movement decoding from cortex with a new, unsupervised-learning algorithm," *Journal of neural engineering*, vol. 15, no. 2, p. 026 010, 2018.

[23] R. D. Flint, E. W. Lindberg, L. R. Jordan, L. E. Miller, and M. W. Slutzky, "Accurate decoding of reaching movements from field potentials in the absence of spikes," *Journal of neural engineering*, vol. 9, no. 4, p. 046 006, 2012.

[24] F. Pei *et al.*, "Neural latents benchmark'21: Evaluating latent variable models of neural population activity," *arXiv preprint arXiv:2109.04463*, 2021.

[25] Y. You *et al.*, "Large batch optimization for deep learning: Training bert in 76 minutes," *arXiv preprint arXiv:1904.00962*, 2019.

[26] M. G. Perich, L. E. Miller, M. Azabou, and E. L. Dyer, *Long-term recordings of motor and premotor cortical spiking activity during reaching in monkeys*, DANDI archive, 2023.

[27] J. I. Glaser, A. S. Benjamin, R. H. Chowdhury, M. G. Perich, L. E. Miller, and K. P. Kording, "Machine learning for neural decoding," *Eneuro*, vol. 7, no. 4, 2020.

[28] M. R. Keshtkaran *et al.*, "A large-scale neural network training framework for generalized estimation of single-trial population dynamics," *Nature Methods*, pp. 1–6, 2022.

[29] J. Ye and C. Pandarinath, "Representation learning for neural population activity with neural data transformers," *bioRxiv*, 2021.

[30] R. Liu, M. Azabou, M. Dabagia, J. Xiao, and E. Dyer, "Seeing the forest and the tree: Building representations of both individual and collective dynamics with transformers," *Advances in Neural Information Processing Systems*, vol. 35, pp. 2377–2391, 2022.

[31] F. R. Willett, D. T. Avansino, L. R. Hochberg, J. M. Henderson, and K. V. Shenoy, "High-performance brain-to-text communication via handwriting," *Nature*, vol. 593, no. 7858, pp. 249–254, 2021.

[32] J. H. Siegle *et al.*, "Survey of spiking in the mouse visual system reveals functional hierarchy," *Nature*, vol. 592, no. 7852, pp. 86–92, 2021.

[33] G. Zerveas, S. Jayaraman, D. Patel, A. Bhamidipaty, and C. Eickhoff, "A transformer-based framework for multivariate time series representation learning," in *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, 2021, pp. 2114–2124.

[34] Y. Nie, N. H. Nguyen, P. Sinthong, and J. Kalagnanam, "A time series is worth 64 words: Long-term forecasting with transformers," in *International Conference on Learning Representations*, 2023.

[35] X. Shou, T. Gao, D. Subramanian, D. Bhattacharjya, and K. Bennett, "Influence-aware attention for multivariate temporal point processes," in *2nd Conference on Causal Learning and Reasoning*, 2023.

[36] X. Shou, T. Gao, S. Subramaniam, D. Bhattacharjya, and K. Bennett, "Concurrent multi-label prediction in event streams," in *AAAI Conference on Artificial Intelligence*, 2023.

[37] S. Zuo, H. Jiang, Z. Li, T. Zhao, and H. Zha, "Transformer hawkes process," in *International conference on machine learning*, PMLR, 2020, pp. 11 692–11 702.

[38] T. Le and E. Shlizerman, "Stndt: Modeling neural population activity with a spatiotemporal transformer," *arXiv preprint arXiv:2206.04727*, 2022.

[39] R. Liu *et al.*, "Drop, swap, and generate: A self-supervised approach for generating neural activity," *Advances in neural information processing systems*, vol. 34, pp. 10 587–10 599, 2021.

[40] J. Gou, B. Yu, S. J. Maybank, and D. Tao, "Knowledge distillation: A survey," *International Journal of Computer Vision*, vol. 129, no. 6, pp. 1789–1819, 2021.

[41] M. Ryoo, A. Piergiovanni, A. Arnab, M. Dehghani, and A. Angelova, "Tokenlearner: Adaptive space-time tokenization for videos," *Advances in Neural Information Processing Systems*, vol. 34, pp. 12 786–12 797, 2021.

[42] H. Yin, A. Vahdat, J. Alvarez, A. Mallya, J. Kautz, and P. Molchanov, "Adavit: Adaptive tokens for efficient vision transformer," *arXiv preprint arXiv:2112.07658*, 2021.

[43] X. Liu, T. Wu, and G. Guo, "Adaptive sparse vit: Towards learnable adaptive token pruning by fully exploiting self-attention," *arXiv preprint arXiv:2209.13802*, 2022.

[44] C. Pandarinath *et al.*, "Inferring single-trial neural population dynamics using sequential auto-encoders," *Nature Methods*, p. 1, 2018.

[45] M. Azabou *et al.*, "Mine your own view: Self-supervised learning through across-sample prediction," *arXiv preprint arXiv:2102.10106*, 2021.

[46] D. Zhou and X.-X. Wei, "Learning identifiable and interpretable latent models of high-dimensional neural activity using pi-VAE," *arXiv preprint arXiv:2011.04798*, 2020.

[47] S. M. Peterson, R. P. Rao, and B. W. Brunton, "Learning neural decoders without labels using multiple data streams," *Journal of Neural Engineering*, vol. 19, no. 4, p. 046 032, 2022.

[48] A. Farshchian, J. A. Gallego, J. P. Cohen, Y. Bengio, L. E. Miller, and S. A. Solla, "Adversarial domain adaptation for stable brain-machine interfaces," *arXiv preprint arXiv:1810.00045*, 2018.

[49] S. Schneider, J. H. Lee, and M. W. Mathis, "Learnable latent embeddings for joint behavioural and neural analysis," *Nature*, pp. 1–9, 2023.

[50] E. L. Dyer *et al.*, "A cryptography-based approach for movement decoding," *Nature Biomedical Engineering*, vol. 1, no. 12, pp. 967–976, 2017.

[51] J. A. Gallego, M. G. Perich, R. H. Chowdhury, S. A. Solla, and L. E. Miller, "Long-term stability of cortical population dynamics underlying consistent behavior," *Nature Neuroscience*, vol. 23, no. 2, pp. 260–270, 2020.

[52] A. D. Degenhart *et al.*, "Stabilization of a brain–computer interface via the alignment of low-dimensional spaces of neural activity," *Nature Biomedical Engineering*, pp. 1–14, 2020.

[53] X. Ma, F. Rizzoglio, K. L. Bodkin, E. Perreault, L. E. Miller, and A. Kennedy, "Using adversarial networks to extend brain computer interface decoding accuracy over time," *eLife*, vol. 12, C. Kemere, J. I. Gold, and C. Kemere, Eds., e84296, Aug. 2023.

[54] M. Safaie *et al.*, "Preserved neural population dynamics across animals performing similar behaviour," *bioRxiv*, 2022. eprint: https://www.biorxiv.org/content/early/2022/09/27/2022.09.26.509498.full.pdf.

[55] L. Long *et al.*, "Automatic classification of cichlid behaviors using 3d convolutional residual networks," *Iscience*, vol. 23, no. 10, p. 101 591, 2020.

[56] A. Kennedy, "The what, how, and why of naturalistic behavior," *Current Opinion in Neurobiology*, vol. 74, p. 102 549, 2022.

[57] M. Marks *et al.*, "Deep-learning-based identification, tracking, pose estimation and behaviour classification of interacting primates and mice in complex environments," *Nature Machine Intelligence*, vol. 4, no. 4, pp. 331–340, 2022.

[58] C. Segalin *et al.*, "The mouse action recognition system (mars) software pipeline for automated analysis of social behaviors in mice," *Elife*, vol. 10, e63720, 2021.

[59] O. Sturman *et al.*, "Deep learning-based behavioral analysis reaches human accuracy and is capable of outperforming commercial solutions," *Neuropsychopharmacology*, vol. 45, no. 11, pp. 1942–1952, 2020.

[60] K. Bates, K. N. Le, and H. Lu, "Deep learning for robust and flexible tracking in behavioral studies for c. elegans," *PLOS Computational Biology*, vol. 18, no. 4, e1009942, 2022.

[61] M. Kabra, A. A. Robie, M. Rivera-Alba, S. Branson, and K. Branson, "Jaaba: Interactive machine learning for automatic annotation of animal behavior," *Nature methods*, vol. 10, no. 1, pp. 64–67, 2013.

[62] Z. Chen *et al.*, "Alphatracker: A multi-animal tracking and behavioral analysis tool," *biorxiv*, 2020.

[63] K. Fujii *et al.*, "Learning interaction rules from multi-animal trajectories via augmented behavioral models," *Advances in Neural Information Processing Systems*, vol. 34, pp. 11 108–11 122, 2021.

[64] D. J. Anderson and P. Perona, "Toward a science of computational ethology," *Neuron*, vol. 84, no. 1, pp. 18–31, 2014.

[65] Y. Jia *et al.*, "Selfee, self-supervised features extraction of animal behaviors," *Elife*, vol. 11, e76218, 2022.

[66] Y. Yang *et al.*, "Simper: Simple self-supervised learning of periodic targets," *arXiv preprint arXiv:2210.03115*, 2022.

[67] A. B. Wiltschko *et al.*, "Mapping sub-second structure in mouse behavior," *Neuron*, vol. 88, no. 6, pp. 1121–1135, 2015.

[68] J. J. Sun, A. Kennedy, E. Zhan, D. J. Anderson, Y. Yue, and P. Perona, "Task programming: Learning data efficient behavior representations," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 2876–2885.

[69] J. Co-Reyes, Y. Liu, A. Gupta, B. Eysenbach, P. Abbeel, and S. Levine, "Self-consistent trajectory autoencoder: Hierarchical reinforcement learning with trajectory embeddings," in *International conference on machine learning*, PMLR, 2018, pp. 1009–1018.

[70] X. Chen, J. Xu, R. Zhou, W. Chen, J. Fang, and C. Liu, "Trajvae: A variational autoencoder model for trajectory generation," *Neurocomputing*, vol. 428, pp. 332–339, 2021.

[71] R. McIlroy-Young, Y. Wang, S. Sen, J. Kleinberg, and A. Anderson, "Detecting individual decision-making style: Exploring behavioral stylometry in chess," *Advances in Neural Information Processing Systems*, vol. 34, pp. 24 482–24 497, 2021.

[72] J. Bromley, I. Guyon, Y. LeCun, E. Säckinger, and R. Shah, "Signature verification using a" siamese" time delay neural network," *Advances in neural information processing systems*, vol. 6, 1993.

[73] V. Makoviychuk *et al.*, "Isaac gym: High performance gpu-based physics simulation for robot learning," *arXiv preprint arXiv:2108.10470*, 2021.

[74] J. J. Sun *et al.*, "The mabe22 benchmarks for representation learning of multiagent behavior," *arXiv preprint arXiv:2207.10553*, 2022.

[75] E. Levina and P. Bickel, "The earth mover's distance is the mallows distance: Some insights from statistics," in *Proceedings Eighth IEEE International Conference on Computer Vision. ICCV 2001*, vol. 2, 2001, 251–256 vol.2.

[76] L. Hou, C.-P. Yu, and D. Samaras, "Squared earth mover's distance-based loss for training deep neural networks," *arXiv preprint arXiv:1611.05916*, 2016.

[77] S. Shalev-Shwartz and A. Tewari, "Stochastic methods for l 1 regularized loss minimization," in *Proceedings of the 26th Annual International Conference on Machine Learning*, 2009, pp. 929–936.

[78] T. F. Sterkenburg and P. D. Grünwald, "The no-free-lunch theorems of supervised learning," *Synthese*, vol. 199, no. 3, pp. 9979–10 015, 2021.

[79] Y. Tian, D. Krishnan, and P. Isola, "Contrastive multiview coding," in *European conference on computer vision*, Springer, 2020, pp. 776–794.

[80] S. Bai, J. Z. Kolter, and V. Koltun, *An empirical evaluation of generic convolutional and recurrent networks for sequence modeling*, 2018.

[81] A. v. d. Oord *et al.*, "Wavenet: A generative model for raw audio," *arXiv preprint arXiv:1609.03499*, 2016.

[82] L.-C. Chen, Y. Zhu, G. Papandreou, F. Schroff, and H. Adam, "Encoder-decoder with atrous separable convolution for semantic image segmentation," in *Proceedings of the European conference on computer vision (ECCV)*, 2018, pp. 801–818.

[83] J.-B. Grill *et al.*, "Bootstrap your own latent: A new approach to self-supervised learning," *arXiv preprint arXiv:2006.07733*, 2020.

[84] A. Recasens *et al.*, *Broaden your views for self-supervised video learning*, 2021.

[85] D. Guo *et al.*, "Bootstrap latent-predictive representations for multitask reinforcement learning," *arXiv preprint arXiv:2004.14646*, 2020.

[86] Z. Yue *et al.*, "Ts2vec: Towards universal representation of time series," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 36, 2022, pp. 8980–8987.

[87] M. Azabou, M. Dabagia, R. Liu, C.-H. Lin, K. B. Hengen, and E. L. Dyer, "Using self-supervision and augmentations to build insights into neural coding," *NeurIPS 2021 Workshop on Self-supervised Learning: Theory and Practice*, 2021.

[88] J. Liang, V. Makoviychuk, A. Handa, N. Chentanez, M. Macklin, and D. Fox, *Gpu-accelerated robotic simulation for distributed reinforcement learning*, 2018.

[89] N. Rudin, D. Hoeller, P. Reist, and M. Hutter, "Learning to walk in minutes using massively parallel deep reinforcement learning," in *Conference on Robot Learning*, PMLR, 2022, pp. 91–100.

[90] J. J. Sun, A. Kennedy, E. Zhan, D. J. Anderson, Y. Yue, and P. Perona, "Task programming: Learning data efficient behavior representations," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 2876–2885.

[91] C. R. Qi, H. Su, K. Mo, and L. J. Guibas, "Pointnet: Deep learning on point sets for 3d classification and segmentation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 652–660.

[92] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[93] K. Luxem *et al.*, *Open-source tools for behavioral video analysis: Setup, methods, and development*, 2022.

[94] J. J. Sun *et al.*, "Self-supervised keypoint discovery in behavioral videos," *arXiv preprint arXiv:2112.05121*, 2021.

[95] A. Wu *et al.*, "Deep graph pose: A semi-supervised deep graphical model for improved animal pose tracking," *Advances in Neural Information Processing Systems*, vol. 33, pp. 6040–6052, 2020.

[96] B. J. Forys, D. Xiao, P. Gupta, and T. H. Murphy, "Real-time selective markerless tracking of forepaws of head fixed mice using deep neural networks," *Eneuro*, vol. 7, no. 3, 2020.

[97] T. D. Pereira *et al.*, "Fast animal pose estimation using deep neural networks," *Nature methods*, vol. 16, no. 1, pp. 117–125, 2019.

[98] A. Mathis *et al.*, "Deeplabcut: Markerless pose estimation of user-defined body parts with deep learning," *Nature Neuroscience*, vol. 21, pp. 1281–1289, 2018.

[99] K. Luxem, F. Fuhrmann, J. Kürsch, S. Remy, and P. Bauer, "Identifying behavioral structure from deep variational embeddings of animal motion," *BioRxiv*, 2020.

[100] A. Nair, T. Karigo, B. Yang, S. W. Linderman, D. J. Anderson, and A. Kennedy, "An approximate line attractor in the hypothalamus that encodes an aggressive internal state," *bioRxiv*, 2022. eprint: https://www.biorxiv.org/content/early/2022/04/19/2022.04.19.488776.full.pdf.

[101] J. D. Marshall, T. Li, J. H. Wu, and T. W. Dunn, "Leaving flatland: Advances in 3d behavioral measurement," *Current Opinion in Neurobiology*, vol. 73, p. 102 522, 2022.

[102] A. B. Wiltschko *et al.*, "Revealing the structure of pharmacobehavioral space through motion sequencing," *Nature neuroscience*, vol. 23, no. 11, pp. 1433–1443, 2020.

[103] C. Wan, T. Probst, L. V. Gool, and A. Yao, "Self-supervised 3d hand pose estimation through training by fitting," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 10 853–10 862.

[104] T. Jakab, A. Gupta, H. Bilen, and A. Vedaldi, "Self-supervised learning of interpretable keypoints from unlabelled videos," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 8787–8797.

[105] C. Shi *et al.*, "Learning disentangled behavior embeddings," *Advances in Neural Information Processing Systems*, vol. 34, 2021.

[106] Y. Li and S. Mandt, "Disentangled sequential autoencoder," *arXiv preprint arXiv:1803.02991*, 2018.

[107] R. Villegas, J. Yang, S. Hong, X. Lin, and H. Lee, "Decomposing motion and content for natural video sequence prediction," *arXiv preprint arXiv:1706.08033*, 2017.

[108] J.-T. Hsieh, B. Liu, D.-A. Huang, L. F. Fei-Fei, and J. C. Niebles, "Learning to decompose and disentangle representations for video prediction," *Advances in neural information processing systems*, vol. 31, 2018.

[109] D. P. Kingma, M. Welling, *et al.*, "An introduction to variational autoencoders," *Foundations and Trends® in Machine Learning*, vol. 12, no. 4, pp. 307–392, 2019.

[110] S. R. Nilsson *et al.*, "Simple behavioral analysis (simba)–an open source toolkit for computer classification of complex social behaviors in experimental animals," *BioRxiv*, 2020.

[111] T. Chen, S. Kornblith, M. Norouzi, and G. Hinton, "A simple framework for contrastive learning of visual representations," *arXiv preprint arXiv:2002.05709*, 2020.

[112] Y.-A. Chung and J. Glass, "Speech2vec: A sequence-to-sequence framework for learning word embeddings from speech," *arXiv preprint arXiv:1803.08976*, 2018.

[113] D. Niizumi, D. Takeuchi, Y. Ohishi, N. Harada, and K. Kashino, *Byol for audio: Self-supervised learning for general-purpose audio representation*, 2021.

[114] M. Azabou *et al.*, *Mine your own view: Self-supervised learning through across-sample prediction*, 2021.

[115] K. Hassani and A. H. Khasahmadi, "Contrastive multi-view representation learning on graphs," in *International Conference on Machine Learning*, PMLR, 2020, pp. 4116–4126.

[116] Y. Hu and Y. Zhang, "Graph contrastive learning with local and global mutual information maximization," in *2020 The 8th International Conference on Information Technology: IoT and Smart City*, ser. ICIT 2020, Xi'an, China: Association for Computing Machinery, 2020, pp. 74–78, ISBN: 9781450388559.

[117] M. J. Mendelson *et al.*, "Learning signatures of decision making from many individuals playing the same game," in *2023 11th International IEEE/EMBS Conference on Neural Engineering (NER)*, 2023, pp. 1–5.

[118] Z. V. Johnson *et al.*, "Automated measurement of long-term bower behaviors in lake malawi cichlids using depth sensing and action recognition," *Scientific Reports*, vol. 10, no. 1, p. 20 573, 2020.

[119] J. J. Sun *et al.*, "Bkind-3d: Self-supervised 3d keypoint discovery from multi-view videos," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2023, pp. 9001–9010.

[120] T. I. B. Laboratory *et al.*, "Standardized and reproducible measurement of decision-making in mice," *eLife*, vol. 10, N. Uchida and M. J. Frank, Eds., e63711, May 2021.

[121] D. F. Parks *et al.*, "A non-oscillatory, millisecond-scale embedding of brain state provides insight into behavior," *bioRxiv*, 2023. eprint: https://www.biorxiv.org/content/early/2023/06/27/2023.06.09.544399.full.pdf.

[122] C. Doersch, A. Gupta, and A. A. Efros, "Unsupervised visual representation learning by context prediction," in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 1422–1430.

[123] L. Perez and J. Wang, "The effectiveness of data augmentation in image classification using deep learning," *arXiv preprint arXiv:1712.04621*, 2017.

[124] K. He, H. Fan, Y. Wu, S. Xie, and R. Girshick, "Momentum contrast for unsupervised visual representation learning," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 9729–9738.

[125] Q. Cai, Y. Wang, Y. Pan, T. Yao, and T. Mei, "Joint contrastive learning with infinite possibilities," *arXiv preprint arXiv:2009.14776*, 2020.

[126] J. Song and S. Ermon, "Multi-label contrastive predictive coding," *arXiv preprint arXiv:2007.09852*, 2020.

[127] A. v. d. Oord, Y. Li, and O. Vinyals, "Representation learning with contrastive predictive coding," *arXiv preprint arXiv:1807.03748*, 2018.

[128] P. Veličković, W. Fedus, W. L. Hamilton, P. Liò, Y. Bengio, and R. D. Hjelm, "Deep graph infomax," *arXiv preprint arXiv:1809.10341*, 2018.

[129] Y. Zhu, Y. Xu, F. Yu, Q. Liu, S. Wu, and L. Wang, "Deep graph contrastive representation learning," *arXiv preprint arXiv:2006.04131*, 2020.

[130] M. Schwarzer, A. Anand, R. Goel, R. D. Hjelm, A. Courville, and P. Bachman, "Data-efficient reinforcement learning with momentum predictive representations," *arXiv preprint arXiv:2007.05929*, 2020.

[131] M. Gutmann and A. Hyvärinen, "Noise-contrastive estimation: A new estimation principle for unnormalized statistical models," in *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, JMLR Workshop and Conference Proceedings, 2010, pp. 297–304.

[132] A. Ermolov, A. Siarohin, E. Sangineto, and N. Sebe, "Whitening for self-supervised representation learning," *arXiv preprint arXiv:2007.06346*, 2020.

[133] J. Zbontar, L. Jing, I. Misra, Y. LeCun, and S. Deny, *Barlow twins: Self-supervised learning via redundancy reduction*, 2021. arXiv: 2103.03230 [cs.CV].

[134] X. Wang and G. Qi, "Contrastive learning with stronger augmentations," *CoRR*, vol. abs/2104.07713, 2021. arXiv: 2104.07713.

[135] K. Kong *et al.*, "Flag: Adversarial data augmentation for graph neural networks," *arXiv preprint arXiv:2010.09891*, 2020.

[136] X. Chen and K. He, "Exploring simple siamese representation learning," *arXiv preprint arXiv:2011.10566*, 2020. arXiv: 2011.10566 [cs.CV].

[137] L. Huang, C. Zhang, and H. Zhang, "Self-adaptive training: Bridging the supervised and self-supervised learning," *arXiv preprint arXiv:2101.08732*, 2021. arXiv: 2101.08732 [cs.LG].

[138] Z. Ma, G. G. Turrigiano, R. Wessel, and K. B. Hengen, "Cortical circuit dynamics are homeostatically tuned to criticality in vivo," *Neuron*, 2019.

[139] K. B. Hengen, A. T. Pacheco, J. N. McGregor, S. D. V. Hooser, and G. G. Turrigiano, "Neuronal firing rate homeostasis is inhibited by sleep and promoted by wake," *Cell*, vol. 165, no. 1, pp. 180–191, 2016.

[140] J. Y. Cheng, H. Goh, K. Dogrusoz, O. Tuzel, and E. Azemi, "Subject-aware contrastive learning for biosignals," *arXiv preprint arXiv:2007.04871*, 2020.

[141] P. Sarkar and A. Etemad, "Self-supervised learning for ecg-based emotion recognition," in *ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2020, pp. 3217–3221.

[142] H. Banville, I. Albuquerque, A. Hyvärinen, G. Moffat, D.-A. Engemann, and A. Gramfort, "Self-supervised representation learning from electroencephalography signals," in *2019 IEEE 29th International Workshop on Machine Learning for Signal Processing (MLSP)*, 2019, pp. 1–6.

[143] S. Gidaris, P. Singh, and N. Komodakis, *Unsupervised representation learning by predicting image rotations*, 2018. arXiv: 1803.07728 [cs.CV].

[144] C. Doersch, A. Gupta, and A. A. Efros, *Unsupervised visual representation learning by context prediction*, 2016. arXiv: 1505.05192 [cs.CV].

[145] Y. Tian, L. Yu, X. Chen, and S. Ganguli, *Understanding self-supervised learning with dual deep networks*, 2021. arXiv: 2010.00578 [cs.LG].

[146] P. H. Richemond *et al.*, "Byol works even without batch statistics," *arXiv preprint arXiv:2010.10241*, 2020.

[147] H. Zhang, M. Cisse, Y. N. Dauphin, and D. Lopez-Paz, *Mixup: Beyond empirical risk minimization*, 2018. arXiv: 1710.09412 [cs.LG].

[148] Y. Kalantidis, M. B. Sariyildiz, N. Pion, P. Weinzaepfel, and D. Larlus, *Hard negative mixing for contrastive learning*, 2020. arXiv: 2010.01028 [cs.CV].

[149] J. Robinson, C.-Y. Chuang, S. Sra, and S. Jegelka, *Contrastive learning with hard negative samples*, 2021. arXiv: 2010.04592 [cs.LG].

[150] M. Caron, P. Bojanowski, A. Joulin, and M. Douze, "Deep clustering for unsupervised learning of visual features," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 132–149.

[151] C. Zhuang, A. L. Zhai, and D. Yamins, *Local aggregation for unsupervised learning of visual embeddings*, 2019. arXiv: 1903.12355 [cs.CV].

[152] M. Caron, I. Misra, J. Mairal, P. Goyal, P. Bojanowski, and A. Joulin, "Unsupervised learning of visual features by contrasting cluster assignments," *arXiv preprint arXiv:2006.09882*, 2020.

[153] P. Sermanet *et al.*, "Time-contrastive networks: Self-supervised learning from video," *arXiv preprint arXiv:1704.06888*, 2018. arXiv: 1704.06888 [cs.CV].

[154] D. Dwibedi, J. Tompson, C. Lynch, and P. Sermanet, "Learning actionable representations from visual observations," *arXiv preprint arXiv:1808.00928*, 2019. arXiv: 1808.00928 [cs.CV].

[155] P. H. Le-Khac, G. Healy, and A. F. Smeaton, "Contrastive representation learning: A framework and review," *IEEE Access*, vol. 8, pp. 193 907–193 934, 2020.

[156] H. Banville, O. Chehab, A. Hyvarinen, D. Engemann, and A. Gramfort, "Uncovering the structure of clinical EEG signals with self-supervised learning," *Journal of Neural Engineering*, 2020.

[157] R. P. Rao and D. H. Ballard, "Predictive coding in the visual cortex: A functional interpretation of some extra-classical receptive-field effects," *Nature Neuroscience*, vol. 2, no. 1, pp. 79–87, 1999.

[158] M. G. Perich and L. E. Miller, "Altered tuning in primary motor cortex does not account for behavioral adaptation during force field learning," *Experimental brain research*, vol. 235, no. 9, pp. 2689–2704, 2017.

[159] J. I. Glaser, M. G. Perich, P. Ramkumar, L. E. Miller, and K. P. Kording, "Population coding of conditional probability distributions in dorsal premotor cortex," *Nature communications*, vol. 9, no. 1, p. 1788, 2018.

[160] R. P. Sheridan, "Time-split cross-validation as a method for estimating the goodness of prospective prediction," *Journal of Chemical Information and Modeling*, vol. 53, no. 4, pp. 783–790, 2013.

[161] J. E. Chung *et al.*, "A fully automated approach to spike sorting," *Neuron*, vol. 95, no. 6, 1381–1394.e6, 2017.

[162] A. P. Buccino *et al.*, "Spikeinterface, a unified framework for spike sorting," *bioRxiv*, 2019. eprint: https://www.biorxiv.org/content/early/2019/10/07/796599.full.pdf.

[163] L. Matthey, I. Higgins, D. Hassabis, and A. Lerchner, *Dsprites: Disentanglement testing sprites dataset*, https://github.com/deepmind/dsprites-dataset/, 2017.

[164] X. Bouthillier, K. Konda, P. Vincent, and R. Memisevic, "Dropout as data augmentation," *arXiv preprint arXiv:1506.08700*, 2016. arXiv: 1506.08700 [stat.ML].