Using Machine Learning to Identify Functional Properties of Complex Systems: from Neurons to Networks

Daniel Zdeblick

A dissertation
submitted in partial fulfillment of the
requirements for the degree of

Doctor of Philosophy

University of Washington

2023

Reading Committee:

Eric T. Shea-Brown, Chair

Michael A. Buice

Daniela M. Witten

Program Authorized to Offer Degree:
Department of Electrical and Computer Engineering

University of Washington

Abstract

Using Machine Learning to Identify Functional Properties of Complex Systems: from Neurons to Networks

Daniel Zdeblick

Chair of the Supervisory Committee:
Eric T. Shea-Brown
Department of Applied Mathematics

The human brain may be the most complicated thing in the discovered universe. Now is an exciting time, when the capabilities of computing and neural recording technologies provide unprecedented opportunity to use advanced machine learning models to begin understanding it. The two main goals of such models are to successfully predict neural data withheld from training and to provide scientific insight into the functioning of the neural system. To this end, I introduce two model designs, each of which achieves these goals by incorporating a scientific hypothesis about the brain into the structure of the model.

First, in Chapter 2, I train models of individual neurons with the assumption that these models are well described by a finite number of cell-types. A hierarchical model that uses the expectation-maximization algorithm to simultaneously learn parameters associated with individual neurons and a description of the cell-types allows parameters associated with each neuron to borrow strength from other neurons' data. I use simulated data to show that this allows the hierarchical model to recover true model parameters and cell type identities better than single-cell models that make no assumption about cell types and are clustered after being fit independently. I apply this hierarchical model to recordings from 634 neurons and show that, compared to the naive approach described above, it yields better predictions of held out data for the overwhelming majority of neurons and discovers cell types that are more robust to the exclusion of different neurons from the training data. These discovered

cell types also relate to available information about the gene-expression, morphology, and location of each neuron.

Then, in Chapter 3, I "joint-train" neural networks to simultaneously predict the activity of a neural population and perform an auxiliary task, hypothesized to be related the the function of those neurons. I find analytic conditions for when the inclusion of an auxiliary task can improve prediction of neural responses in a linear network, and derive an expression for the degree of improvement, which reveals that the most useful auxiliary tasks are those most predictable from the neural data. I then show that this result is born out in deep, nonlinear, structured networks via numerical analyses with simulated data. These analyses also reveal that joint-training allows the model to precisely and accurately associate simulated neurons with a subset of model units. Finally, I use further theory and numerical analysis with the linear model to show how a derived condition on network structure necessary for an auxiliary task to yield improved neural response prediction is modified by nonlinearities, constraints on network parameters, and finite training time, providing insight into how the numerical results differ from those predicted by our theory.

Overall, this work provides insights into how we might incorporate specific hypotheses about brains into statistical models in order to best explain neural data and gain insight into the functioning of neural systems.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

I would first like to acknowledge that I have undertaken this work on the traditional land of the first people of Seattle, the Duwamish People past and present and honor with gratitude the land itself and the Duwamish Tribe.

I thank the University of Washington, the Allen Institute for Brain Science, the city of Seattle, and all those who work behind the scenes to keep them running, from countless department and Allen Institute staff to the bus drivers (especially of the 44) and trash collectors, and all the other people who so often escape notice but without whom this would not be such a nice place to live and work.

I want to thank everyone in the computational neuroscience community at UW and the Allen Institute for providing a stimulating environment to learn and work in, with countless discussions that shaped how I think about the brain, neuroscience, and academia in general. I'll start with my reading committee for this thesis, who have been my core advising team for most of my time at UW, Eric Shea-Brown, Michael Buice, and Daniela Witten. Thank you all for the time and effort you have invested into my progress, and your patience with me when that progress was slower or more chaotic than expected. Thank you Eric for being an honest and supportive mentor to me from my first days at UW until the very end, engaging me with my first research project at UW, helping me find and construct a plan for my thesis work, and navigating many smaller yet stressful moments along the way. Thank you Michael for all your "higher-order-bits" of wisdom (especially the highest-order-bit of focusing on higher-order-bits!), as well as your very low-order bits of more technical advice on everything from the math and programming to paper design and crafting educational materials. And thank you Daniela for holding me to a high standard for my research and writing, and helping me meet that standard with meticulous attention to the details of my models and incredibly thorough feedback on my presentation. I also thank my other committee members, Amy

Orsborne and Andrea Stocco, for providing valuable direction during my generals exam and committee meetings to help get me over the finish line. I also want to thank Bing Brunton for introducing me to the world of graduate research and helping me though the painful process of settling into graduate life and finding a thesis topic. I'd also like to thank others who advised me earlier for my qualifying exam: Mehran Mesbahi, Howard Chizeck, also for first trusting me with admission to this program, and Adrienne Fairhall, also for advising me throughout my program on a wide range of technical and non-technical issues, and, with Eric, creating and fostering an incredible hub for our research community in the Computational Neuroscience Center. I also want to thank Jessica Huszar for all the behind the scenes work to keep this center functioning, as well as all those who have enriched it with their ideas and personality even if just when stopping by for journal club, especially Kameron Harris (special thanks for tons of specific help on many of my early projects), Gabrielle Gutierrez and Helena Liu (special thanks for co-hosting journal club with me), Satpreet Singh and Joe Christianson (special thanks for that cool project we worked on together), Stefano Recanatesi, Leenoy Meshulam, Hannah Choi, Doris Voina, Julien Bloch, Iris Shi, Alison Duffy, Yoni Browning, Hengji Wang, Shashank Sharma, Scott Sterrett, David Bell, Matt Farrell, Ali Weber, Timothy Moore, Zach McNulty, Merav Stern, Braden Brinkman, Fereshteh Lagzi, Faeze Aminmansoor, Hanson Mo, John Ferre, Patrick Zhang, Po-Chen Kuo, Praveen Venkatesh, Preston Jiang, Ryan Raut, Ziyu Lu, Suzanne Lewis, Megan Morrison, Pierre Karashchuk, Joel Zylberberg, and Edgar Walker. I also want to thank all the excellent teachers I've had at UW, which include Andrea, Howard, Mehran, and Eric, as well as Mari Ostendorf and Jim Ritcey for trusting me with teaching responsibilities as a TA and allowing me to hone my teaching skills. From the ECE department I also want to thank Eldridge Alcantara for feedback on my teaching methods and Jen Huberman for helping me navigate all the degree timelines and requirements.

At the Allen, I first want to thank everyone who works to make the Summer Workshops on the Dynamic Brain possible, from the instructors and TAs to the course coordinators and Friday Harbor Labs staff, as well as the other students, who made it an incredible experience

for me as a student, introducing me to the Allen, team science, and git, and as a TA, giving me valuable experience as a mentor and teacher. I especially want to thank Michael, Saskia de Vries, Forrest Collman, Luke Campagnola, Shawn Olsen, and Marina Garrett for being incredible leaders for the course every year I was involved. I also want to thank Michael's group at the Allen, Koosha Khalvati, David Wyrick, Chase King, Matthew Bull, Anamika Agrawal, Amelia Johnson, and Sami Johnson, for your support and unique perspectives on my work and your own.

I also want to acknowledge the many individuals who have given me strength, enrichment, and joy in a more specifically personal and emotional sense during my time at UW. First I want to thank the producers of online content that have taught me so much about the world, the art of explanation, and myself, or just provided me an outlet to relax, especially Timothy Snyder, William Spaniel, 3Blue1Brown, Veritasium, IANW, Jomez, John MacDonald, Real Science, and Liverpool FC. I want to thank the UW counseling center, especially Scott Shiebler and all the group facilitators and members over the years. I want to thank my friends who remind me that there's a world outside of grad school and that it's ok to have fun sometimes, even if I take a month to respond to a text or take my turn in a board game. And of course I am so grateful to my family, who have been there for me in a uniquely specific way throughout the whole journey. Thank you Mom, Dad, Grace, Lyn, Sandy, Walt, and Maddie for loving me unconditionally and reminding me who I even am.

I want to finish by acknowledging a group of people that I simply could not have made it through the last 650 days of my program without. I want to thank the whole community of people supporting Ukraine, there and around the world, via physical assistance and cultural diplomacy, in their jobs, volunteer organizations, and individual acts of advocacy. I especially want to thank the community in Seattle, especially those who organize and participate in the Ukrainian Students United events. I especially want to thank all my classmates in Ukrainian language class for being "мої люли" and above all my teachers Sofiia Fedzhora and Nataliia Kovtoniuk for introducing me to this community and giving me giving me the possibility and inspiration to become part of it, while patiently answering all my inane questions. Дякую

всім, я не зміг зробити без вас. Finally, I want to thank Volodymyr Zelenskyy for inspiring the world with honesty, determination, and hope, and to all of those who risk their lives every day fighting for the principles of freedom, justice, and democracy, so that the rest of us don't have to.

DEDICATION

Слава Україні.

Героям слава.

Chapter 1

INTRODUCTION

1.1 Using computational models to understand brains

The primary goal of computational neuroscience is to use mathematical techniques to advance our understanding of nervous systems. Like other sciences, this is accomplished largely through the use of *models* that can explain data while providing a relatively simple insight into how something works. In this context, a model is a system of equations with parameters that are learned from (fitted to) data in order to allow the model to capture the structure of the data as well as possible. The choice of what equations to use corresponds to a set of assumptions about the system producing the data, with stricter assumptions generally yielding a simpler model that cannot capture the data as accurately, but is easier to understand. We prefer models that are as simple as possible, and thus yield the clearest understanding, but can still accurately predict the behavior of the system we wish to understand.

The early use of such models to describe neural activity focused on understanding individual neurons, and achieved unprecedented success with the Hodgkin-Huxley model in 1952 [43]. This work used an electrical circuit model to explain the dynamics that produce action potentials, or spikes, the primary means by which neurons communicate with each other. While it only described this process taking place in a single spatially homogeneous region and in response to a single electrical current injection, it represented an unprecedented level of biological realism, providing the field with new understanding about the mechanisms driving neural dynamics and setting a high bar for how accurately models can predict neural activity. Since then, there have been many new single-cell models, which generally fall into two categories. There are models that use the electrical circuit approach, but consider neurons as having more complex spatial extent, both in terms of heterogeneous local dynamics and complicated morphology, which is modelled by many connected compartments that influence each other's dynamics. Such models have helped us better understand how the complex and

diverse physical structure of different neurons contributes to their activity and functioning. Likewise, some Hodgkin-Huxley-style models incorporate additional variables not present in the original work in order to better understand different types of dynamics, like bursting and adaptation (see [42] for a review of these more detailed Hodgkin-Huxley-style models). The other category of work seeks to find simpler models than Hodgkin and Huxley's, which sacrifice some accuracy in order to produce a single-cell model that is easier to fit to data and/or understand using dynamical systems analysis [6, 9, 13, 20, 28, 35, 44, 52, 59, 64, 70].

This latter category has been especially useful for modelling populations of many neurons interacting with each other, using the simplicity of these single-cell models to compensate for the diversity of single cells and the complexity of network structure, thereby allowing for easier analysis of the functioning of these neural circuits. Even so, finding appropriately complex models is difficult, and as a result much early work focused on the involvement of relatively simple neural circuits in producing observed behaviors that are tightly controlled by the experimenter [15], or theoretical behavior that is presumed to be optimal [4, 58]. While this makes it easier to propose a simpler model that can explain the data, it can be difficult to integrate many such models of different neural circuits and different behaviors into a cohesive explanation. This observation was made as early as 1973 [63], and the field has since come a long way in terms of producing models that explain a much wider range of more naturalistic behaviors. However, it remains very important for those who are creating new models to consider how their work fits into the bigger picture.

1.2 A framework for computational neuroscience

In general terms, computational neuroscience poses meta-problem, where many individuals work on separate, small problems, and the solutions need to be woven together. In such situations, it is necessary to have a framework - some structure that specifies what the big-picture solution will look like and how the narrower solutions can fit together to build towards it. In 1982, David Marr proposed three levels of analysis as the first such framework for understanding complex information processing systems like brains [57], and, although many of the ideas have evolved, they are still useful today. Marr's framework stipulates that a complete explanation of how an information processing system produces a given behavior

must describe the system at three levels of abstraction, as well as how each more abstract level arises from the one below it (see Figure 1.1). These levels are:

1. **Computation**, wherein we must describe the behavior in terms of an input to the system and the output it produces as a function of that input;
2. **Algorithms**, wherein we must describe how the input is *represented* by the system, and what sequence of operations transforms it into a representation of the output; and
3. **Implementation**, wherein we must describe the physical components of the system and how they interact.

It is important to note that the input and output do not necessarily need to be directly observed in the behavior of the animal. For example, we might wish to describe a neural circuit like the hippocampus, that receives no direct inputs from external senses and sends no direct outputs to motor neurons. In this case, both the input and output might be variables that are related to the behavior of the animal, but not directly observable from it.

Since Marr proposed this framework, many have argued that the desired understanding required at the middle level is not appropriate. Brains perform computations through an inherently continuous process that cannot, in general, be broken down into a sequence of discrete operations like the classical algorithms of computer science. In some cases, the best we can do may be to describe the architecture of the neural circuit and the rules by which it changes to learn an accurate transformation from the input to the output, i.e. a *learning algorithm* [71]. Even in cases where we might be able to understand the behavior of a neural circuit as an algorithm performing a computation, it is still useful to understand how such an algorithm can arise from the learning algorithm responsible for producing it.

Evaluating the extent to which specific architectures and learning rules can model a specific neural circuit has become increasingly possible and popular in recent years, due to the increase in computing power that allows us to train a special class of models, *artificial neural networks* (ANNs). These models loosely mimic the structure and learning processes of neural circuits, defining the activity of each of its "units" as some function g of a linear combination of unit activities and inputs to the model. In the supervised learning context, a computation is defined by a data set D_{train} of training samples of input-output (x_i, z_i) pairs, and the ANN learns to generate accurate predictions of what the output associated with a

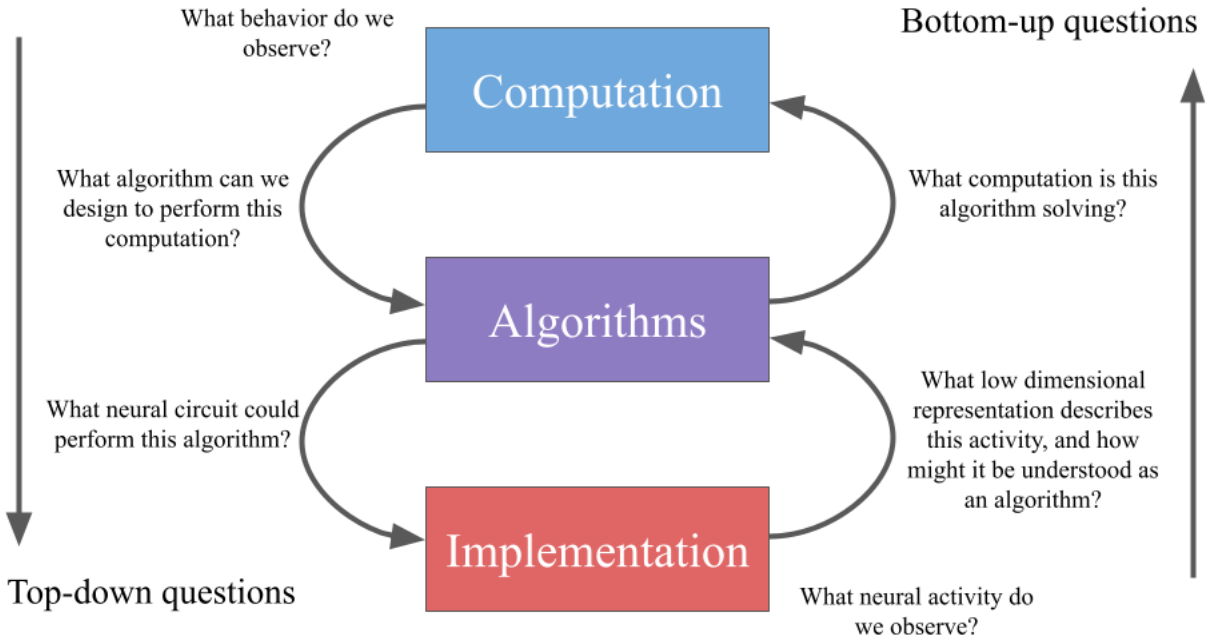


Figure 1.1: **A framework for understanding the brain: Marr's three levels of abstraction.** Understanding a given neural system at each level of abstraction is required for full understanding, and can be obtained through two paths, or some combination of them. In the top-down path, behavioral data is used to inform abstract models that make coarse predictions about neural implementation. In the bottom-up path, neural data is used to inform mechanistic circuit models that can be analyzed or reduced to gain insight into abstract variables they are representing, and how those variables are manipulated to perform a computation.

given input will be $(\tilde{z}_i(x_i;\theta))$ by adjusting parameters, θ, of the network. ANNs choose these adjustments in order to minimize a cost function, which describes how accurately the model predicts the input from the output for some appropriate loss function L, where lower values correspond to more accurate predictions:

$$C = \sum_{i \in D_{\text{train}}} L(z_i, \tilde{z}_i(x_i;\theta)). \tag{1.1}$$

Alongside the increase in computing power, there has also been dramatic improvement in neural recording technology, increasing both the quantity and quality of data that can be collected during an experiment. Some experiments sample neural circuits with incredible density, recording simultaneously from over 10,000 neurons at a time [91], allowing us to better evaluate how well a given model characterizes the functioning of that circuit at the computational and algorithmic levels. Other experiments [1, 37] accurately measure the rich dynamics of large numbers of individual neurons with incredible temporal resolution, allowing us to create a more comprehensive and accurate account of the fundamental physical components that make up the brain's implementation.

1.3 The big question:

Marr's framework provides a substrate for comparing models of similar phenomena and merging models of different phenomena, by setting goals for what any given model should attempt to explain. What it does not do, however, is make recommendations about how to actually design such models. This then brings us the the big question that drives this work:

> How should we define models that will harness the power of modern computers and neural datasets to accurately predict neural data and help us understand the brain in terms of implementation, algorithms, and computations?

In this work, I focus on the assumptions we make when we define models of neural activity. In this context, we are ultimately interested in models that predict the activity of N neurons in a circuit, $y_t \in \mathbb{R}^N$, measured at many times t, in response to inputs from outide the circuit. This can be achieved by minimizing a cost function identical to (1.1), except that the term to be predicted is y_t instead of z_i. While the specific choices of L and g also constitute assumptions, we focus on those that consist of constraints placed on θ. Every additional constraint placed on the model parameters θ limits how well the dataset used to train the model can be explained, but it does so more when the assumption is inaccurate. What's more, when the amount of data available to train a model $|D_{\text{train}}|$ is small (relative to the expressive power of the model), then the model can *overfit* to the training data, predicting that data very well but failing to generalize to data that was held-out from training. In

this case, additional, sufficiently accurate assumptions can prevent overfitting and raise the generalization performance of a model, while making the model easier to interpret.

It is important to recognize that these model assumptions reflect *scientific hypotheses* about the structure of the data, either implicitly or explicitly. This promotes a level of caution when choosing model assumptions, but also allows us to test the hypothesis underlying a specific assumption by looking for an improvement in generalization performance when it is included in a model, compared to when is is not. This view supports a cycle, where hypotheses are incorporated as model assumptions, and then the results of applying that model to data are used to generate new hypotheses that can be incorporated into the next generation of model assumptions. This process is, on the one hand, "bottom-up" research that seeks to ground explanations of neural systems in models of neural activity, and attempts to explain the systems in more abstract terms by analyzing these models and reducing them to craft a more abstract explanation of the neural activity. On the other hand, this process can also have a flavor of "top-down" research that grounds explanations in models of behavior, and postulates how such models might be implemented by neural circuitry. Linderman and Gershman [55] specifically advocate for assumptions that achieve this by encoding hypotheses that describe the brain at the computational level. Both top-down and bottom-up research are valuable on their own, but Marr's levels provide a framework wherein they can contribute to a unified understanding of a neural system through different approaches. Moreover, Linderman and Gersman highlight a methodology for achieving this unification in a statistically rigorous way.

Consider, as an example of this process, a "vanilla" recurrent neural network, a simple form of ANN that feeds the model input and all unit activities from the previous time-step into the generation of each unit's current activity. Suppose that we want to use this model to predict the activity of a neural circuit, and ultimately understand its functioning at a more abstract level. For simplicity, let's also assume that we're associating each of N of the model units directly with a recorded neuron (I'll have more to say on this assumption later):

$$\begin{aligned} \tilde{y}_t(x_t;\theta) &= [u_t]_{1,\ldots,N}, \\ u_t &= g(\theta_x x_t + \theta_u u_{t-1}). \end{aligned} \tag{1.2}$$

Now suppose we want to test the hypothesis that having more synapses between neurons is costly for the brain. Because we've associated neurons directly with model units, we might incorporate this hypothesis into our model assumptions by constraining our matrix of connections between neurons θ_u to be sparse by including an ℓ_1 penalty, $\sum_{r,c} |[\theta_u]_{r,c}|$, in our cost function. The addition of this penalty necessarily means that the training data will not be predicted as well, since it is best predicted by the cost function that only optimizes for those predictions. However, this penalty might result in better prediction of validation data that was held out from training, in which case we might conclude that, indeed, maintaining synapses is costly for brains.

However, there are limits to how precisely we can test a specific hypothesis. In this example we interpreted the ℓ_1 penalty as reflecting the costliness of synapses, but this cost might be metabolic in nature, or maybe due to improved performance of the circuit when it has fewer synapses. Worse yet, as far as understanding synapses is concerned, another model assumption may be having a harmful effect on predictions that the ℓ_1 penalty is ameliorating. For example, real neurons are far more complex than these artificial units, but a large number of artificial units in the ANN might be accurately representing the dynamics of a single neuron. Beniaguev, Segev, and London [8] addressed this specific problem and estimated that on the order of 100-1000 units are required to accurately model a specific kind of neuron, and work by Jones and Kording [48] demonstrated that units whose connections are constrained to mimic a single dendritic tree are capable of solving machine learning problems as well as when they are not constrained. In our setup, the ℓ_1 penalty might then be understood as grouping together artificial units that are working together to model a single neuron, while limiting between-neuron interactions to a small fraction of these units (that might be understood as corresponding to synapses) and limiting within-neuron interactions of some units (modelling the dendritic tree) to have a tree-like arrangement. Most likely, it would be very difficult to find such a pattern, and conclude anything beyond the claim that ℓ_1's

general utility in avoiding overfitting extends to this problem. In general, one must be careful in interpreting model assumptions as specific hypotheses, due to such interactions between different assumptions and limits on how specifically a given assumption actually describes the hypothesis we are interested in. This is a key reason why evaluating models for their interpretability should be tested on data where *ground truth* is known about whatever it is that we would like to conclude from our interpretation, which in this case is the physical structure of the neural circuit.

Our example also highlights the importance of a specific type of model assumption about the relationship between terms in the model and components in the brain. Such assumptions are encoded in the structure of how the final prediction of neural activity is made, and limit how well one can explain the functioning of neural circuits at the level of implementation. For the greatest understanding of how the brain's implementation relates to its functioning, we would like to identify each functionally distinct component of the brain with a single variable in our model, and be able to understand the interaction between model variables in terms of dynamics that occur within and between neurons. In practice, this is very difficult to achieve with recordings from real neural circuits due to the high number of unmeasured variables that affect the system, or put another way, the high amount of noise. It is also not always obvious what a "functionally distinct component" of a neural system is. While the neuron doctrine posits that the neuron is this fundamental component, much research has highlighted cases where sub-components of a single neuron can play functionally distinct roles [42]. On the other hand, much experimental [66] and theoretical [23, 29, 40, 72] work argues that individual neurons are often too small a component to be considered functionally distinct, as the activity of large populations of neurons tends to be low-dimensional, such that there is redundancy among individual neurons in representing abstract variables. This redundancy is shown to also arise in ANNs trained to perform tasks, and is understood to allow for better generalization performance in some settings. Therefore, many modelling approaches [25] seek to make use of this observation and predict a dimension-reduced representation of the activity of a neural circuit, instead of individual recordings.

In this work, I do not model any components smaller than a neuron, and presume that a more precise correspondence between neurons and model units will generally facilitate

interpretation of the neural system, provided that this correspondence is accurate. However, we must contend with the reality that the level of precision that is accurately achievable may be limited by the quality and dimensionality of available data.

First motivating hypothesis: neurons can be categorized into types Generally, studies that seek to understand the brain at the level of implementation model data from single neurons in isolation or pairs connected by a synapse. While neurons are incredibly complicated and are present in larger brains in incredible diversity, experiments suggest that much may be gained by attempting to understand the diversity of neurons using a cell-types framework. The central idea behind all conceptualizations of cell types is that there are relatively few types and each neuron is more similar to other neurons of the same type than neurons of a different type. Different research has focused on different aspects of individual neurons that may be described this way, such as their morphology [37], gene expression [79, 80, 84], electrical properties [22, 81], and connectivity with other neurons of specific types [47]. However, all these studies share this common framework, and thus use an unsupervised clustering approach, which generally takes features of data collected from each neuron (or pair of neurons in the case of clustering by connectivity [47]), and seeks to discover an arrangement of clusters that best explains the distribution of these features across all neurons. Usually, these features are not directly related to models that predict neural activity, but recent work [81] has made this connection, using the fitted parameters of such models as the features of each neuron to be clustered. Such work opens a door towards using research into neural cell types to help us understand the activity of neural circuits in terms of the implementation details of their constituent neurons. In chapter 2, I propose a statistical framework for such functional-model based cell-types research, and ask how we can most accurately discover cell types within it.

Second motivating hypothesis: neural circuits perform computations Other research that seeks to understand the functioning of neural circuits at an algorithmic or computational level records data from a large number of neurons, and uses a linear combination of model terms to predict their measured activity. One such approach that

has gained traction in recent years is training an ANN to predict the output of a computation z_i, then using the responses of the trained units to predict the responses of recorded neurons y_i to some input x_i [62, 88]. Such work seeks to identify an input-output relationship (computation) that explains neural activity. In chapter 3, I propose a general framework that describes such approaches and ask how we can obtain the best models within that framework. By "best," I mean predicting the activity of each neuron correctly and by using a small and appropriate subset of units in the ANN.

In each chapter, the "fundamental ingredient" that I introduce consists of identifying the hypothesis driving an area of research (in chapter 2, that individual neurons belong to cell types; in chapter 3, that neural circuits can be understood as performing a specific computation), and incorporating that hypothesis into a predictive model of neural activity as an assumption. In both cases, the primary goal is using theoretical tools and simulated data to understand what exactly the improved performance of such a model allows us to learn about our driving hypothesis. In chapter 2, I focus on determining how the discovered clusters of putative cell types depend on the true distribution of the underlying data. In chapter 3, I focus on understanding which putative computations are most useful for improving predicted neural activity and how well individual neurons can be understood as playing specific roles in that computation.

In both chapters, there must be a baseline model that the hypothesis-motivated assumption of interest is added to. The remainder of this introduction focuses, in general terms, on introducing those baseline models and describing how our assumption modifies them. Because the focus of this work is on understanding the improvement that comes from including these hypothesis-motivated assumptions, I choose baseline models that are both simple and common in the literature, allowing for a deeper understanding of the effect the assumptions have and effective contextualization of this work within the broader field, both in terms of the significance of performance improvements and interpretation of elements of the baseline model. Accordingly, in chapter 2, I use a single-neuron model that is relatively expressive to seek understanding at the implementational level, and, in chapter 3, I use a network model with many, much simpler units to seek understanding at Marr's computational level.

As described earlier, this trend (whereby simpler single-neuron models are used to model larger networks) is prevalent in the literature and is sensible in that it leads to models that are as simple as possible and still explain the phenomenon of interest. However, ultimate understanding of the brain will consist of explanations that bridge Marr's three levels; in the concluding chapter, I will explore how the models I have presented might be combined to seek integrated explanations of how specific cell types work together in a neural circuit to perform a computation.

1.4 Baseline Models

In chapter 2, the baseline model consists of a Generalized Linear Model (GLM) predicting single-cell neural activity, and a Gaussian Mixture Model (GMM) predicting the fitted parameters of those GLMs from a cluster distribution.

The GLM is a standard extension of linear regression that can be understood as a single-layer ANN with a single output unit, which in our case predicts the activity of a single recorded neuron y_t. Compared to the generic ANN, the nonlinearity, g, and loss function $L(y_t, \tilde{y}_t)$ are restricted to a subset of such possible pairs, allowing for a convenient guarantee that the corresponding loss function is convex, and thus has a single local minimum. In contrast, ANNs in general may have many local minima, making finding the global one more difficult for an optimizer.

In computational neuroscience, the GLM refers to a specific choice of g and L, as well as what the input to the GLM represents [14, 64]. Firstly, it is assumed that the neural data y_t takes the form of how many times a neuron spiked in the tth time bin, since these spikes or action potentials are the fundamental unit of communication between neurons. Accordingly, this output is modelled with a Poisson distribution, i.e.

$$L(y_t, \tilde{y}_t) = -y_t \log \tilde{y}_t + \tilde{y}_t \propto -\log \frac{\tilde{y}_t^{y_t} e^{-\tilde{y}_t}}{y_t!}. \tag{1.3}$$

Likewise $g(x) = e^x$ is the corresponding canonical nonlinearity that allows for the key convexity property. The input is composed of the stimulus incident on the neuron and the previous spiking activity from the last several time bins:

$$\tilde{y}_t = e^{[x_t,...,x_{t-d_x},y_{t-1},...,y_{t-d_y},1]\theta}, \tag{1.4}$$

where θ is the column vector of parameters learned during training. This means that each input feature $(x_t, ..., y_{t-d_y}, 1)$ has a corresponding element of θ that learns how much to scale it by to produce an good prediction of the output, $\tilde{y}_t$ (the 1 feature allows the model to learn a constant offset).

Compared to the simple ANN units we considered before, this model allows for some interesting dynamics by including feedback from the spiking history, and guarantees that we will find the globally optimal parameter estimate $\hat{\theta}$. However, it is still quite simple yet expressive [87], compatible with more general ANN architectures, and widely used to study single neuron activity (see [65] for a review), making it a good choice of baseline model – even if it is not the best model of neural activity according to many metrics [2].

The GMM is likewise a standard choice for unsupervised clustering across many domains and in neuroscience. In this context, it attempts to explain the distribution of GLM parameter estimates $\hat{\theta}_n$ across many different neurons n, using a mixture of Gaussians distribution. This problem can be summarized by its cost function:

$$C = -\sum_n \log[\sum_{k=1}^{K} \pi_k f(\theta_n; \mu_k, \Sigma_k)], \tag{1.5}$$

where π_k, μ_k, and Σ_k are the weight, mean, and covariance matrix of the kth cluster, and f denotes the probability density function of a multivariate normal distribution. It is worth noting that while the other optimization problems introduced here can be solved by methods that descend the gradient of the corresponding cost function, this cluster structure in this cost function allows it to be most effectively solved by the expectation-maximization algorithm (EM).

In chapter 3, the baseline model consists of training a feedforward ANN to perform a computation, followed by a linear regression of the trained unit activities onto neural recordings. A feedforward ANN has its artificial units arranged in a sequence of L layers, and the activities of the units in each layer are determined as follows:

$$
\begin{aligned}
u_i^{(1)} &= g_1(\theta^{(1)}[x_i, 1]), \\
u_i^{(l)} &= g_l(\theta^{(l)}[u_i^{(l-1)}, 1]) \text{ if } l > 1, \\
\tilde{z}_i &= g_z(\theta_z[u_i^{(L)}, 1]).
\end{aligned} \tag{1.6}
$$

Here, $u_i^{(l)}$ is the vector of unit activities in the lth layer in response to the ith sample, $\theta^{(l)}$ is the learned matrix of weights connecting that layer to the previous one, and the 1 appended to the input to each layer allows the network to learn constant offset terms. As before, g is a fixed nonlinearity, which we here assume may be different for each layer. We will also use a specific feedforward architecture, the convolutional neural network (CNN), for a baseline model. CNN's are used with inputs that have some spatiotemporal structure like an image, sound, or video, and arranges each layer correspondingly, such that each has many units that respond homogeneously to local features in a specific spatiotemporal region of the previous layer. These groups of homogeneous units are referred to as channels, where layer l has C_l different channels that derive their activites from all channels in the previous layer. All this generally amounts to placing a certain restriction on each $\theta^{(l)}$ for $l \leq L_C$, where L_C is the number of layers in which this spatiotemporal structure is preserved ($L \geq L_C \geq 1$). This restriction can be written as follows:

$$
\begin{aligned}
u_i^{(1,c,d1,d2)} &= g_1(\theta^{(1,c)}[x_i^{(d1-s,d2-s)}, ..., x_i^{(d1+s,d2+s)}, 1]), \\
u_i^{(l,c,d1,d2)} &= g_l(\theta^{(l,c)}[u_i^{(l-1,1,d1-s,d2-s)}, ..., u_i^{(l-1,C_{l-1},d1+s,d2+s)}, 1]) \text{ if } L_C \geq l > 1, \\
u_i^{(l)} &= g_l(\theta^{(l)}[u_i^{(l-1)}, 1]) \text{ if } l > L_C, \\
\tilde{z}_i &= g_z(\theta_z[u_i^{(L)}, 1]),
\end{aligned} \tag{1.7}
$$

where for simplicity and consistency with the baseline model I use, there are only two spatial dimensions (indexed by $d1$ and $d2$) and one value of s that determines the spatiotemporal extent of the weights associated with each unit. The key element of CNNs is that many different units in the same layer l and *channel*, c, i.e. $u^{(l,c,d1,d2)}$ for many different values of $d1$ and $d2$, have their activities determined by the same matrix of parameters $\theta^{(l,c)}$, which is often called a *filter*.

Another feature common in CNNs is that the nonlinearies g often make reductions across the spatiotemporal dimensions, the most common of which is simply taking the maximum of all unit activities in a given region. These are known as "pooling" operations, and ensure that unit activites are not affected by small translations of the input in the spatiotemporal dimensions; if there are pooling operations in several subsequent layers, then the deeper layers will be invariant to larger translations.

It is this sharing of parameters and reduction across spatiotemporal dimensions that gives CNNs their ability to make accurate predictions on held out data from a smaller number of training samples than is possible for fully-connected feedforward networks. Consistent with the theme of this work, this benefit only arises when the assumptions underlying these constraints on θ and g are appropriate, namely that the data is arranged spatiotemporally and possesses invariances along those spatiotemporal dimensions.

Note that in (1.6) and (1.7) we have used an increasing number of superscripts to index our input x, units u, and parameters θ, corresponding with the increased structure of the model. However, keep in mind that these are just indexing a single vector (or matrix in the case of θ) to help us stay organized. So when we want to predict neural responses y_i to stimuli x_i using the responses of our trained units to the same x_i, it is entirely consistent to consider them as a single vector $u_i(x_i; \hat{\theta})$, no matter how the units are arranged. The baseline model I use for this prediction is linear regression:

$$\tilde{y}_i = \theta_y u_i(x_i; \hat{\theta}), \tag{1.8}$$

where $\hat{\theta}$ has already been estimated in training the ANN, and θ_y is optimized to minimize the mean-squared error between the true and predicted neural responses:

$$L(y_i, \tilde{y}_i) = (y_i - \tilde{y}_i)^2 \tag{1.9}$$

Both of the baseline models above consist of two sequential stages of model fitting, each with their own cost function, and in both cases this implies an assumption that the results of the first stage of fitting are appropriate for feeding into the second. In chapter 2, this assumption is that the parameters of single-cell models can be well described by a cluster

distribution. In chapter 3, it is that a model trained to perform the chosen task will also be able to predict neural activity. In each chapter, I properly incorporate the assumption into an integrated model of neural activity, that simultaneously optimizes both of the constituent cost functions. This is achieved using a novel adaptation of the EM algorithm in chapter 2, and the auxiliary task framework from machine learning in chapter 3. I show that we can use these integrated models to more accurately predict data held out from training and more reliably interpret the fitted parameters (relative to these baseline models). I conclude by discussing future potential for combining the models presented in chapters 2 and 3, as well as other assumptions that might be similarly incorporated into models of neural activity.

Chapter 2

MODELING FUNCTIONAL CELL TYPES IN SPIKE TRAIN DATA

Abstract

A major goal of computational neuroscience is to build accurate models of the activity of neurons that can be used to interpret their function in circuits. Here, we explore using *functional cell types* to refine single-cell models by grouping them into functionally relevant classes. Formally, we define a hierarchical generative model for cell types, single-cell parameters, and neural responses, and then derive an expectation-maximization algorithm with variational inference that maximizes the likelihood of the neural recordings. We apply this "simultaneous" method to estimate cell types and fit single-cell models from simulated data, and find that it accurately recovers the ground truth parameters. We then apply our approach to *in vitro* neural recordings from neurons in mouse primary visual cortex, and find that it yields improved prediction of single-cell activity. We demonstrate that the discovered cell-type clusters are well separated and generalizable, and thus amenable to interpretation. We then compare discovered cluster memberships with locational, morphological, and transcriptomic data. Our findings reveal the potential to improve models of neural responses by explicitly allowing for shared functional properties across neurons.

Publication Note

An early version of this work was presented at the NeurIPS 2019 Neuro↔AI Workshop [89], and this Chapter was published in PLOS Computational Biology [90].

Author Summary

Computational neuroscience has its roots in fitting and interpreting predictive models of the activity of individual neurons. In recent years, more attention has focused on models of how ensembles of neurons work together to perform computations. However, fitting these more

complex models to data is challenging, limiting our ability to use them to understand real neural systems. One idea that has improved our understanding of populations of neurons is *cell types*, where neurons of the same type have similar properties. While the idea of cell types is old, recent work has focused on functional cell types, where the properties of interest are derived from fitted predictive models of the activity of single neurons. In this work, we develop a method that *simultaneously* fits a predictive model of each neuron's activity and groups neurons into functional cell types. Compared to existing techniques, this method produces more accurate models of single-cell neural activity and better groupings of neurons into types. This method can thus contribute the use of cell types in better understanding the components of neural systems based on our increasingly rich observations of their functional responses.

2.1 Introduction

A primary goal of computational neuroscience is to formulate simplifying assumptions on neural structure and activity that allow for models that are both tractable to fit to data and useful for understanding neural systems. One such simplifying assumption that is gaining traction is that all neurons in the brain belong to a one of a finite number of *cell types*, and that neurons of the same type have similar properties. Many studies have sought to cluster neurons into putative cell types according to properties of their morphology [37], gene expression [79, 80, 84], electrophysiology [22, 81], and connectivity [47].

We are interested here in *functional cell types* that group neurons with similar properties of functional output, which we consider to be their spiking response. Clearly, there is a strong relationship between this notion and cell types defined on more specific properties of electrophysiological responses, and potentially transcriptomic and morphological types as well. However, this functional view of cell types only imputes value to differences between neurons that are useful in predicting their spiking responses to stimuli. The work of Teeter et al. [81] took an important step towards identifying functional cell types from electrophysiology data with a sequential approach. Specifically, these authors fit *functional models of single-neuron responses*, and then clustered the resulting parameter estimates. By using functional model parameters as features to be clustered, these authors explicitly relate functional cell types to

prediction of neural responses. Like the authors of that work, we believe that this relationship is crucial to identifying cell types that will help us understand the brain's function.

The present work has two primary goals: to use the idea of functional cell types to improve parameter estimates for predictive models of individual neural responses, and to discover the best possible grouping of cells into types. To meet these twin goals, we develop an approach that *simultaneously* estimates single-cell model parameters and cell types (clusters of those parameters). All of the previous studies on discovering cell types cited above, except [47], take a *sequential approach* to defining cell types: they extract features of interest from each neuron's data, and then cluster these features using an unsupervised clustering algorithm. Unlike these sequential approaches, our method allows estimates of each individual neuron's parameters to "borrow strength" from other neurons' data. As we will show, our simultaneous method leads to more accurate single-cell models and cell-type clusterings than a matched sequential approach. Even in real-world situations where there are no "ground truth" cell types, the simultaneous method yields improved prediction of single-cell responses and clusters that are more robust to the exclusion of different neurons from training, providing validation for this approach to discovering cell types.

In greater detail, we define a hierarchical generative model of functional cell types, single-cell parameters, and neural responses. In our model, each neuron belongs to one of several possible (unknown) functional cell types; the distribution of parameters for that neuron's response model is governed by the (unknown) cell type to which it belongs. This is a mixture model in which each sample is the entire spike train from a single neuron, and the distribution of this sample, conditional on its cluster membership, requires marginalizing out parameters of the response model. To fit this hierarchical model, we adapt the expectation-maximization (EM) algorithm [27] to simultaneously estimate the parameters for each neuron's response model and the functional cell types, in order to provide a good fit to the spiking response of the recorded neurons.

We apply our method to simulated data, as well as to the Allen Cell Types Database collected by the Allen Institute for Brain Science [1] (also used in [81]). We verify that our approach accurately recovers the ground truth model underlying a simulated dataset, and demonstrate that it discovers robust and interpretable cell types that improve prediction of

neural recordings. In particular, we show that the benefits of applying our method to predict neural activity depend on the training data in a way that is consistent with the notion of "borrowing strength"; the improvement is greatest when the training dataset includes more neurons.

For comparison with the simultaneous method, we use a version of the sequential method with the same model structure, but where single-cell parameters are first fit, then clustered into cell types. This comparison reveals the potential improvement of borrowing strength between neurons.

2.2 Methods

We begin by formalizing the goals of this work, and then detail two approaches to meet them: a "sequential method," based on approaches pervading existing cell-types research, and a "simultaneous method," our proposed improvement. We then adapt the EM algorithm for our model in order to estimate the parameters of the model from data, and use the Bayesian information criterion (BIC) to select hyperparameters. All code used to carry out the analyses in this paper is available at `https://github.com/zdeblick/ClusteredGLMs`.

Throughout, we use bold symbols (e.g. $\mathbf{x}_i$) to denote vectors where the tth element is denoted e.g. $x_i(t)$, and capital Latin letters to denote natural number constants (N, K, T_i). We use $f(\mathbf{z}; \boldsymbol{\mu}_z, \Sigma_z)$ to denote the probability density function of a multivariate Gaussian distribution, and a hat (e.g. $\hat{\boldsymbol{\beta}}_i$) to denote an estimate.

2.2.1 Goals

In this work we seek a clustering of neurons into cell types that best explains their functional (spiking) responses. We consider a dataset of spiking responses to stimuli $x_i(t), y_i(t), t \in \{1, ..., T_i\}, i \in \{1, ..., N\}$, where N is the number of neurons, $y_i(t)$ is the number of spikes that the i^{th} neuron fires in the t^{th} time bin, $x_i(t)$ is the value of the stimulus to that neuron in that time bin, and T_i is the duration of that neuron's recording (in time bins). Our goals can then be formalized as estimating two quantities from our dataset:

1. For each neuron, $i, \ldots, N$, parameters $\hat{\boldsymbol{\beta}}_i$ of a predictive model for the ith neuron's response, $P(\mathbf{y}_i|\mathbf{x}_i, \hat{\boldsymbol{\beta}}_i)$.

2. Functional cell-type assignment of each neuron into one of K types, $\hat{k}_i \in \{1, ..., K\}$, that best capture the distribution of $\hat{\boldsymbol{\beta}}$ across all neurons.

We compare two approaches for achieving these goals: a "sequential method" of first estimating $\hat{\boldsymbol{\beta}}_1, \ldots, \hat{\boldsymbol{\beta}}_N$, and then clustering these point estimates into cell-type assignments; and a "simultaneous method" that uses an expectation-maximization algorithm to estimate both $\hat{\boldsymbol{\beta}}_i$ and $\hat{k}_i$ in tandem. The sequential method is motivated by the approach of Teeter et al. [81], although implementation details differ.

2.2.2 Sequential Method

The sequential method defines functional cell types in two steps. In the first step, the data $\mathbf{x}_i, \mathbf{y}_i$ for the ith neuron are fit with a single-cell model, which we describe below, that is parameterized by some vector of parameters $\boldsymbol{\beta}_i$. In the second step, the estimated parameters $\hat{\boldsymbol{\beta}}_i$ are clustered with a Gaussian mixture model. See Algorithm 1.

Single-Cell Model

The goal of the single-cell model is to predict $\mathbf{y}_i$ from $\mathbf{x}_i$, using some learned vector of parameters $\boldsymbol{\beta}_i$. We assume the conditional independence of time bins in order to decompose this probability as follows:

$$P_{SC}(\mathbf{y}_i|\mathbf{x}_i; \boldsymbol{\beta}_i) = \prod_{t=1}^{T_i} P_{SC}(y_i(t)|y_i(1), ..., y_i(t-1), x_i(1), ..., x_i(t); \boldsymbol{\beta}_i). \tag{2.1}$$

Here, we use P_{SC} to denote the probability distribution of spiking for a single-cell. While there are many options for models of single-cell spiking that parameterize this probability, in this work we use the generalized linear model (GLM) [86], illustrated in Fig 2.1A. Specifically, we fit a Poisson GLM on a transformed set of covariates:

$$y_i(t) \sim Poisson\left(\exp\left[\sum_{\tau=0}^{T^{\text{stim}}-1} \beta_i^{\text{stim}}(\tau+1)\tilde{x}_i(t-\tau d^{\text{stim}}) + \sum_{\tau=1}^{T^{\text{self}}} \beta_i^{\text{self}}(\tau) y_i(t-\tau) + \beta_i^0\right]\right). \tag{2.2}$$

In (2.2):

- $\boldsymbol{\beta}_i^{\text{stim}} \in \mathbb{R}^{T^{\text{stim}}}$ is called the *stimulus filter* for the ith neuron, as it filters the stimulus provided to that neuron.

- $\tilde{x}_i(t) = \sum_{s=0}^{d^{\text{stim}}-1} x_i(t-s)$ is the pre-filtered stimulus, using a rectangular filter of length d^{stim}. Accordingly, note that the stimulus filter's coefficients are effectively spaced d^{stim} time bins apart. The stimuli we will consider vary much more slowly than the timescale of spiking, and this feature reduces the dimensionality of $\boldsymbol{\beta}_i^{\text{stim}}$ to combat overfitting by effectively downsampling the stimulus. As such, we refer to d^{stim} as the stimulus downsampling factor.

- $\boldsymbol{\beta}_i^{\text{self}} \in \mathbb{R}^{T^{\text{self}}}$ is called the *self-interaction filter* for the ith neuron, as it filters that neuron's own history of recent spiking activity.

- β_i^0 is the offset term for the ith neuron.

- Collectively, $\boldsymbol{\beta}_i \equiv [\boldsymbol{\beta}_i^{\text{stim}}, \boldsymbol{\beta}_i^{\text{self}}, \beta_i^0]^\top$, yielding a total of $\dim(\boldsymbol{\beta}_i) = T^{\text{stim}} + T^{\text{self}} + 1$ parameters of the single-cell model. We use these superscripts with many symbols throughout this document, clarifying the component(s) of $\boldsymbol{\beta}_i$ with which they are associated (the absence of a superscript corresponds to all of $\boldsymbol{\beta}_i$).

- We take $x_i(\tau) = y_i(\tau) = 0$ for $\tau < 1$. That is, we zero-pad our data.

When we maximize the corresponding log-likelihood with respect to $\boldsymbol{\beta}_i$, we include ℓ_2 regularization on all parameters except the offset:

$$\hat{\boldsymbol{\beta}}_i(\lambda^{\text{stim}}, \lambda^{\text{self}}) = \arg\max_{\boldsymbol{\beta}} \left\{ \frac{1}{T_i^{\text{train}}} \log P_{SC}(\mathbf{y}_i^{\text{train}} | \mathbf{x}_i^{\text{train}}; \boldsymbol{\beta}) - \frac{1}{2}\lambda^{\text{stim}} ||\boldsymbol{\beta}^{\text{stim}}||_2^2 - \frac{1}{2}\lambda^{\text{self}} ||\boldsymbol{\beta}^{\text{self}}||_2^2 \right\}. \tag{2.3}$$

In (2.3), $||\cdot||_2$ denotes the ℓ_2-norm of a vector, and the superscript "train" indicates that we are using a training set to fit the parameters $\boldsymbol{\beta}_i$. This optimization problem is convex. We solve it with the trust-region Newton-conjugate-gradient algorithm (`trust-ncg` in the `scikit-learn minimize` function [68]).

To select the regularization hyperparameters λ^{stim} and λ^{self}, we use cross-validation. That is, each neuron's data is partitioned into L equally-sized bins of adjacent time-points, $(\boldsymbol{x}_i^1, ..., \boldsymbol{x}_i^L)$ and $(\boldsymbol{y}_i^1, ..., \boldsymbol{y}_i^L)$. For a logarithmically spaced grid of choices of $\lambda^{\text{stim}} \in [10^{-7}, 1], \lambda^{\text{self}} \in [10^{-7}, 1]$, we then compute the cross-validated log-likelihood, averaged over all data points from all N neurons:

$$VALL(\lambda^{\text{stim}}, \lambda^{\text{self}}) = \sum_{i=1}^{N} \frac{1}{NT_i} \sum_{l=1}^{L} \log P_{SC}(\mathbf{y}_i^l | \mathbf{x}_i^l; \hat{\boldsymbol{\beta}}_i(\lambda^{\text{stim}}, \lambda^{\text{self}})). \tag{2.4}$$

In (2.4), $\hat{\boldsymbol{\beta}}_i(\lambda^{\text{stim}}, \lambda^{\text{self}})$ is computed using (2.3), where the other partitions of the data $((\boldsymbol{x}_i^1, ..., \boldsymbol{x}_i^{l-1}, \boldsymbol{x}_i^{l+1}, ..., \boldsymbol{x}_i^L)$ and $(\boldsymbol{y}_i^1, ..., \boldsymbol{y}_i^{l-1}, \boldsymbol{y}_i^{l+1}, ..., \boldsymbol{y}_i^L))$ are used for training.

The $\lambda^{\text{stim}}, \lambda^{\text{self}}$ that maximize this quantity are selected and used to fit $\hat{\boldsymbol{\beta}}_i$ to all of the data used for cross-validation (i.e. not including test data).

We choose a Poisson GLM in (2.2) because its log likelihood is convex with respect to its parameters $\boldsymbol{\beta}_i$, simplifying estimation. Additionally, the GLM has been widely used to model single-cell responses (see [65] for a review), and is able to produce a wide variety of spiking dynamics observed in neural data [87]. The GLM parameters are also relatively amenable to interpretation: $\beta_i^{\text{self}}(\tau)$ (or, respectively, $\beta_i^{\text{stim}}(\tau + 1)$) determines how a spike (the stimulus) that happened τ time bins in the past affects the probability that the neuron will spike in the present. Very negative values of $\beta_i^{\text{self}}(\tau)\mathbf{y}_i(t - \tau)$ or $\beta_i^{\text{stim}}(\tau + 1)\tilde{x}_i(t - \tau d^{\text{stim}})$ suppress spiking at time t; very positive values promote spiking.

Cluster Model

After fitting the single-cell model (2.3), we cluster the fitted self-interaction filters, $\{\hat{\boldsymbol{\beta}}_1^{\text{self}}, ..., \hat{\boldsymbol{\beta}}_N^{\text{self}}\}$, to produce cell-type assignments for each neuron. See Section S2.2.1 for consideration of the case where all of $\{\hat{\boldsymbol{\beta}}_1, ..., \hat{\boldsymbol{\beta}}_N\}$, including the stimulus filters, are clustered.

To cluster the coefficient estimates, we fit a Gaussian mixture model (GMM) with

weights π_k, means $\boldsymbol{\mu}_k^{\text{self}}$, and covariances Σ_k^{self}, $k \in \{1, ..., K\}$. The weights must sum to 1 ($\sum_{k=1}^{K} \pi_k = 1$), and additionally we assume that the covariance matrices are diagonal. We fit the GMM with the `GaussianMixture` class from `scikit-learn` [68], enforcing diagonal covariance matrices (`covariance_type='diag'`), initializing the optimizer with the solution of K-means clustering (`init_params='kmeans'`), and choosing the best optimization of those obtained from 20 random initializations (`n_init=20`). Collectively, we refer to these parameters of the clustering as $\Omega_K \equiv \{\pi_1, ...\pi_K, \boldsymbol{\mu}_1^{\text{self}}, ..., \boldsymbol{\mu}_K^{\text{self}}, \Sigma_1^{\text{self}}, ..., \Sigma_K^{\text{self}}\}$. Thus we write the clustering as:

$$\hat{\Omega}_K \leftarrow \arg\max_{\Omega_K} \sum_{i=1}^{N} \log \sum_{k=1}^{K} \pi_k f(\hat{\boldsymbol{\beta}}_i^{\text{self}}; \boldsymbol{\mu}_k^{\text{self}}, \Sigma_k^{\text{self}}). \tag{2.5}$$

Once we have fit the GMM to the parameter estimates $\{\hat{\boldsymbol{\beta}}_1^{\text{self}}, ..., \hat{\boldsymbol{\beta}}_N^{\text{self}}\}$, the maximum likelihood estimate (MLE) for the cell type of each neuron is

$$\hat{k}_i \leftarrow \arg\max_{k} \hat{\pi}_k f(\hat{\boldsymbol{\beta}}_i^{\text{self}}; \hat{\boldsymbol{\mu}}_k^{\text{self}}, \hat{\Sigma}_k^{\text{self}}),\ i = 1, ..., N. \tag{2.6}$$

2.2.3 Simultaneous Method

In the simultaneous method, we define a generative model for the data $x_i(t), y_i(t)$ over all neurons, given that there are K classes. In this model, the response to stimuli will be determined by latent variables, which we denote by $\boldsymbol{\beta}_i = [\boldsymbol{\beta}_i^{\text{stim}}, \boldsymbol{\beta}_i^{\text{self}}, \beta_i^0]^\top$, that are distributed according to a Gaussian distribution, $f(\boldsymbol{\beta}; \boldsymbol{\mu}_k, \Sigma_k)$, given membership in class k. Class membership is given by the categorical distribution with parameters $\pi_1, ..., \pi_K$.

We take $\{\mathbf{x}_1, ..., \mathbf{x}_N\}$ as fixed inputs, and let $\Omega_K = \{\pi_1, ..., \pi_K, \boldsymbol{\mu}_1^{\text{self}}, ..., \boldsymbol{\mu}_K^{\text{self}}, \Sigma_1^{\text{self}}, ..., \Sigma_K^{\text{self}}\}$ denote the set of all parameters of the generative model with K classes. We use the same symbols for the simultaneous and sequential methods to emphasize their shared structure and facilitate comparisons.

We write the joint likelihood of a combination of latent variables and data for the ith neuron as

$$P_{\text{joint}}(k, \boldsymbol{\beta}, \mathbf{y}_i | \mathbf{x}_i; \Omega_K) = P(\mathbf{y}_i | k, \boldsymbol{\beta}, \mathbf{x}_i; \Omega_K) P(\boldsymbol{\beta} | k, \mathbf{x}_i; \Omega_K) P(k | \mathbf{x}_i; \Omega_K). \tag{2.7}$$

Data: $x_i(t), y_i(t),\ t = 1, ..., T_i,\ i = 1, ..., N,\ K,\ d^{\text{stim}}$

Result: $\hat{\Omega}_K \equiv \{\hat{\pi}_k, \hat{\boldsymbol{\mu}}_k^{\text{self}}, \hat{\Sigma}_k^{\text{self}}\},\ k = 1, ..., K, \hat{\boldsymbol{\beta}}_i, \hat{k}_i,\ i = 1, ...N$

`/* Step 1: Estimate response model parameters` $\boldsymbol{\beta}_1, ..., \boldsymbol{\beta}_N$ `*/`

for *a logarithmically spaced grid of* $\lambda^{\text{stim}}, \lambda^{\text{self}}$ **do**

  $VALL(\lambda^{\text{stim}}, \lambda^{\text{self}}) \leftarrow 0$

  for $i = 1, ..., N$ **do**

    Partition the ith neuron's data into L equally-sized bins of adjacent time points, $(\boldsymbol{x}_i^1, ..., \boldsymbol{x}_i^L)$ and $(\boldsymbol{y}_i^1, ..., \boldsymbol{y}_i^L)$

    for *l=1,...,L* **do**

      $\hat{\boldsymbol{\beta}}_i(\lambda^{\text{stim}}, \lambda^{\text{self}}) \leftarrow$

      $\arg\max_{\boldsymbol{\beta}} \left\{ \frac{L}{(L-1)T_i} \sum_{l' \neq l} \log P_{SC}(\mathbf{y}_i^{l'} | \mathbf{x}_i^{l'}; \boldsymbol{\beta}) - \frac{1}{2}\lambda^{\text{stim}} ||\boldsymbol{\beta}^{\text{stim}}||_2^2 - \frac{1}{2}\lambda^{\text{self}} ||\boldsymbol{\beta}^{\text{self}}||_2^2 \right\}.$

      $VALL(\lambda^{\text{stim}}, \lambda^{\text{self}}) \leftarrow$

      $VALL(\lambda^{\text{stim}}, \lambda^{\text{self}}) + \frac{1}{NT_i} \log P_{SC}(\mathbf{y}_i^l | \mathbf{x}_i^l; \hat{\boldsymbol{\beta}}_i(\lambda^{\text{stim}}, \lambda^{\text{self}}))$.

    end

  end

end

$\hat{\lambda}^{\text{stim}}, \hat{\lambda}^{\text{self}} \leftarrow \arg\max_{\lambda^{\text{stim}}, \lambda^{\text{self}}} VALL(\lambda^{\text{stim}}, \lambda^{\text{self}})$.

$\hat{\boldsymbol{\beta}}_i \leftarrow \arg\max_{\boldsymbol{\beta}} \log P_{SC}(\mathbf{y}_i | \mathbf{x}_i; \boldsymbol{\beta}) + \frac{1}{2}\hat{\lambda}^{\text{stim}} ||\boldsymbol{\beta}^{\text{stim}}||_2^2 + \frac{1}{2}\hat{\lambda}^{\text{self}} ||\boldsymbol{\beta}^{\text{self}}||_2^2,\ \ i = 1, ..., N.$

`/* Step 2: Cluster estimated` $\boldsymbol{\beta}_i^{\text{self}}$ `into cell types` k_i`. */`

$\hat{\Omega}_K \leftarrow \arg\max_{\Omega_K} \prod_{i=1}^N \sum_{k=1}^K \pi_k f(\hat{\boldsymbol{\beta}}_i^{\text{self}}; \boldsymbol{\mu}_k^{\text{self}}, \Sigma_k^{\text{self}})$ (Fit a standard GMM)

$\hat{k}_i \leftarrow \arg\max_k \hat{\pi}_k f(\hat{\boldsymbol{\beta}}_i^{\text{self}}; \hat{\boldsymbol{\mu}}_k^{\text{self}}, \hat{\Sigma}_k^{\text{self}}),\ i = 1, ..., N$

Algorithm 1: Sequential method.

We then make the following assumptions:

1. $P(\mathbf{y}_i | k, \boldsymbol{\beta}, \mathbf{x}_i; \Omega_K) \equiv P_{SC}(\mathbf{y}_i | \mathbf{x}_i; \boldsymbol{\beta})$, where P_{SC} was defined in (2.1) and (2.2). Thus, the spiking response of the ith neuron given its stimulus and single-cell parameters $\boldsymbol{\beta}_i$ is conditionally independent of k_i and is not a function of Ω_K.

2. $P(\boldsymbol{\beta} | k, \mathbf{x}_i; \Omega_K) \equiv f(\boldsymbol{\beta}; \boldsymbol{\mu}_k, \Sigma_k)$, so that the single-cell parameters for the ith neuron are

independent of $\mathbf{x}_i$.

3. The matrix Σ_k is diagonal for $k = 1, ..., K$, so that the elements of $\boldsymbol{\beta}_i$ are mutually independent, given k_i.

4. $P(k|\mathbf{x}_i; \Omega_K) = P(k; \Omega_K) \equiv \pi_k$. Thus the cluster label for the ith neuron is independent of $\mathbf{x}_i, \boldsymbol{\mu}_k$, and Σ_k.

With these assumptions, we re-write (2.7) as

$$P_{\text{joint}}(k, \boldsymbol{\beta}, \mathbf{y}_i|\mathbf{x}_i; \Omega_K) = P_{SC}(\mathbf{y}_i|\mathbf{x}_i; \boldsymbol{\beta}) f(\boldsymbol{\beta}; \boldsymbol{\mu}_k, \Sigma_k)\pi_k. \tag{2.8}$$

Recalling that we used superscripts to represent the components of $\boldsymbol{\beta}_i \equiv [\boldsymbol{\beta}_i^{\text{stim}}, \boldsymbol{\beta}_i^{\text{self}}, \beta_i^0]^\top$, we denote the components of the cluster means as $\boldsymbol{\mu}_k = [\boldsymbol{\mu}_k^{\text{stim}}, \boldsymbol{\mu}_k^{\text{self}}, \mu_k^0]^\top$, and likewise for the covariances, with $\Sigma_k^{\text{stim}}, \Sigma_k^{\text{self}}, (\sigma_k^0)^2$ diagonally stacked to form Σ_k (recall that Σ_k is diagonal).

As with the sequential method, we assume that $\boldsymbol{\beta}_i^{\text{stim}}$ and β_i^0 are independent of the cell type k_i. (See Section S2.2.1 for the case where all parameters are cell-type-dependent.) Specifically:

- $\boldsymbol{\mu}_1^{\text{stim}} = ... = \boldsymbol{\mu}_K^{\text{stim}} = 0$, $\Sigma_1^{\text{stim}} = ... = \Sigma_K^{\text{stim}} = (1/\lambda^{\text{stim}}) * I$. This amounts to applying ℓ_2 regularization to the stimulus filters, as in the sequential method.

- The prior for β_i^0, $f(\beta_i^0; \mu_k^0, (\sigma_k^0)^2)$, is flat. That is, no regularization is applied to the offset term; we can think of this as taking $(\sigma_k^0)^2 \to \infty$.

Thus (2.8) may be further factorized as

$$P_{\text{joint}}(k, \boldsymbol{\beta}, \mathbf{y}_i|\mathbf{x}_i; \Omega_K) \propto P_{SC}(\mathbf{y}_i|\mathbf{x}_i; \boldsymbol{\beta}) f(\boldsymbol{\beta}_i^{\text{stim}}; 0, (1/\lambda^{\text{stim}}) * I) f(\boldsymbol{\beta}_i^{\text{self}}; \boldsymbol{\mu}_k^{\text{self}}, \Sigma_k^{\text{self}})\pi_k. \tag{2.9}$$

This hierarchical model has two *hyperparameters*, K and λ^{stim}, a set of *global parameters* describing the cell types, Ω_K, and *latent variables* for the ith neuron, k_i and $\boldsymbol{\beta}_i$. Note that there is no λ^{self} hyperparameter for ℓ_2 regularization of $\boldsymbol{\beta}_i^{\text{self}}$ with a Gaussian prior, as the the clustering induces regularization on these parameters.

To obtain the likelihood of the data, we marginalize over the latent variables:

$$P(\mathbf{y}_i|\mathbf{x}_i;\Omega_K) = \sum_{k=1}^{K} \int P_{\text{joint}}(k, \boldsymbol{\beta}, \mathbf{y}_i|\mathbf{x}_i;\Omega_K) d\boldsymbol{\beta}. \tag{2.10}$$

We further assume that the spiking activity of each neuron is independent, and so the likelihood of an entire dataset is simply a product over neurons:

$$P(\mathbf{y}_1, ..., \mathbf{y}_N|\mathbf{x}_1, ..., \mathbf{x}_N;\Omega_K) = \prod_{i=1}^{N} P(\mathbf{y}_i|\mathbf{x}_i;\Omega_K). \tag{2.11}$$

EM Algorithm

We adapt the expectation-maximization (EM) algorithm [27] for the generative model in (2.9) to find the MLE of Ω_K,

$$\Omega_K^* = \arg\max_{\Omega_K} P(\mathbf{y}_1, ..., \mathbf{y}_N|\mathbf{x}_1, ..., \mathbf{x}_N;\Omega_K) \tag{2.12}$$

(see (2.11)). Each $\mathbf{y}_i$ and $\mathbf{x}_i$ correspond to a single independent neuron (each $\mathbf{y}_i$ is a sample, in common EM language), with associated latent variables k_i and $\boldsymbol{\beta}_i$; the global parameters of our model are Ω_K.

We outline the two steps of the EM algorithm here, and unpack them in detail in Section S2.1:

- E-step: for $i = 1, ..., N$, we compute the posterior distribution over latent variables $k_i, \boldsymbol{\beta}_i$, given the data and the current estimate of global parameters $\hat{\Omega}_K$. We call this $Q_i(k, \boldsymbol{\beta})$:

$$\begin{aligned} Q_i(k, \boldsymbol{\beta}) &\equiv P(k, \boldsymbol{\beta}|\mathbf{x}_i, \mathbf{y}_i; \hat{\Omega}_K) \\ &= \frac{P_{\text{joint}}(k, \boldsymbol{\beta}, \mathbf{y}_i|\mathbf{x}_i; \hat{\Omega}_K)}{P(\mathbf{y}_i|\mathbf{x}_i; \hat{\Omega}_K)} = \frac{P_{\text{joint}}(k, \boldsymbol{\beta}, \mathbf{y}_i|\mathbf{x}_i; \hat{\Omega}_K)}{\sum_{k=1}^{K} \int P_{\text{joint}}(k, \boldsymbol{\beta}, \mathbf{y}_i|\mathbf{x}_i; \hat{\Omega}_K) d\boldsymbol{\beta}}. \end{aligned} \tag{2.13}$$

  Due to the complexity of the denominator of the right hand side of (2.13), we cannot compute the right-hand side exactly. Therefore we employ a weighted Gaussian approximation for $P_{\text{joint}}(k, \boldsymbol{\beta}, \mathbf{y}_i|\mathbf{x}_i; \hat{\Omega}_K)$:

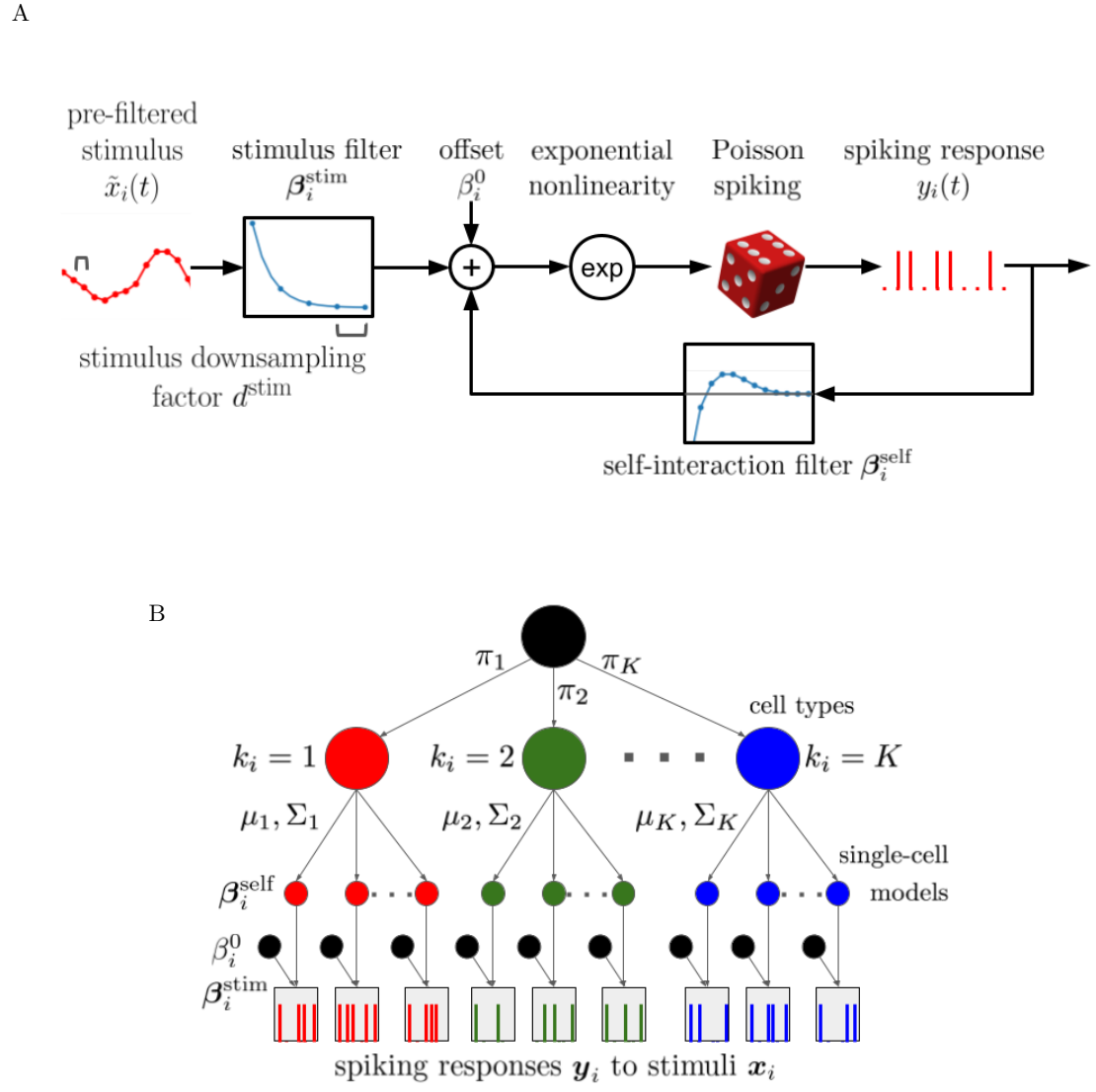


Figure 2.1: **Diagrams of models fitted to spiking data.** A: The Poisson GLM (2.2) used to model the spiking response of a single neuron. B: The generative model (2.11) for the simultaneous method.

$$P_{\text{joint}}(k, \boldsymbol{\beta}, \mathbf{y}_i | \mathbf{x}_i; \hat{\Omega}_K) \approx Z_{i,k} f(\boldsymbol{\beta}; \boldsymbol{m}_{i,k}, c_{i,k}). \tag{2.14}$$

This approximation allows us to simplify (2.13) as follows:

$$\begin{aligned} &\frac{P_{\text{joint}}(k, \boldsymbol{\beta}, \mathbf{y}_i | \mathbf{x}_i; \hat{\Omega}_K)}{\sum_{k=1}^{K} \int P_{\text{joint}}(k, \boldsymbol{\beta}, \mathbf{y}_i | \mathbf{x}_i; \hat{\Omega}_K) d\boldsymbol{\beta}} \\ &\qquad \approx \frac{Z_{i,k} f(\boldsymbol{\beta}; \boldsymbol{m}_{i,k}, c_{i,k})}{\sum_{k=1}^{K} \int Z_{i,k} f(\boldsymbol{\beta}; \boldsymbol{m}_{i,k}, c_{i,k}) d\boldsymbol{\beta}} = \frac{Z_{i,k}}{\sum_{k=1}^{K} Z_{i,k}} f(\boldsymbol{\beta}; \boldsymbol{m}_{i,k}, c_{i,k}). \end{aligned} \tag{2.15}$$

The normalized weights $\frac{Z_{i,k}}{\sum_{k=1}^{K} Z_{i,k}}$ appear repeatedly in what follows, so we denote them

$$\tilde{Z}_{i,k} \equiv \frac{Z_{i,k}}{\sum_{k=1}^{K} Z_{i,k}}. \tag{2.16}$$

This, along with (2.13) and (2.15), allows us to write

$$Q_i(k, \boldsymbol{\beta}) \approx \tilde{Z}_{i,k} f(\boldsymbol{\beta}; \boldsymbol{m}_{i,k}, c_{i,k}). \tag{2.17}$$

Our E-step now consists of finding the optimal $Z_{i,k}, \boldsymbol{m}_{i,k}, c_{i,k}$ to make the approximation in (2.17) as good as possible (more detail in Section S2.1.1). We note that many other approximations may be suitable here, such as mean field variational inference [11], but have chosen this one for its simplicity and the low computational cost of the resulting algorithm (see Section S2.1.3 for details).

- M-step: update the estimate of the global parameters $\hat{\Omega}_K$ by maximizing an approximation to a lower bound on $\log(P(\mathbf{y}_1, ..., \mathbf{y}_N | \mathbf{x}_1, ..., \mathbf{x}_N, \hat{\Omega}_K))$:

$$\hat{\Omega}_K = \arg\max_{\Omega_K} \sum_{i=1}^{N} \sum_{k=1}^{K} \int \tilde{Z}_{i,k} f(\boldsymbol{\beta}; \boldsymbol{m}_{i,k}, c_{i,k}) \log P_{\text{joint}}(k, \boldsymbol{\beta}, \mathbf{y}_i | \mathbf{x}_i; \Omega_K) d\boldsymbol{\beta}. \tag{2.18}$$

Latent Variable Estimates Once the EM algorithm has converged to a final point estimate of Ω_K, which we denote Ω_K^*, we may also want to compute point estimates of k_i and $\boldsymbol{\beta}_i$. While it is tempting to estimate both simultaneously as $(\hat{k}_i, \hat{\boldsymbol{\beta}}_i) = \arg\max_{k,\boldsymbol{\beta}} P_{\text{joint}}(k, \boldsymbol{\beta}, \mathbf{y}_i | \mathbf{x}_i; \Omega_K^*)$, this approach will tend to assign neurons to clusters with smaller estimated variances.

Instead, we estimate k_i for each neuron by maximizing the likelihood with $\boldsymbol{\beta}_i$ marginalized out:

$$\begin{aligned} \hat{k}_i &= \arg\max_k \int P_{\text{joint}}(k, \boldsymbol{\beta}, \mathbf{y}_i | \mathbf{x}_i; \Omega_K^*) d\boldsymbol{\beta} \approx \arg\max_k \int Z_{i,k} f(\boldsymbol{\beta}; \boldsymbol{m}_{i,k}, c_{i,k}) d\boldsymbol{\beta} \\ &= \arg\max_k Z_{i,k}. \end{aligned} \tag{2.19}$$

After estimating k_i, we estimate $\hat{\boldsymbol{\beta}}_i$ by maximizing the likelihood with respect to it,

$$\begin{aligned} \hat{\boldsymbol{\beta}}_i &= \arg\max_{\boldsymbol{\beta}} P_{\text{joint}}(\hat{k}_i, \boldsymbol{\beta}, \mathbf{y}_i | \mathbf{x}_i; \Omega_K^*) \approx \arg\max_{\boldsymbol{\beta}} Z_{i,\hat{k}_i} f(\boldsymbol{\beta}; \boldsymbol{m}_{i,\hat{k}_i}, c_{i,\hat{k}_i}) \\ &= \boldsymbol{m}_{i,\hat{k}_i}. \end{aligned} \tag{2.20}$$

Summary of EM algorithm The overall EM algorithm is spelled out in Algorithm 2. We perform this algorithm 20 times from different random initializations and use the results from the optimization with the lowest loss (see Section 2.2.4). This choice mirrors that used for the sequential method (see Section 2.2.2). We also note that the initialization of the cluster parameters ($\Omega_K \leftarrow$ GMM fit to $\hat{\boldsymbol{\beta}}_1, ..., \hat{\boldsymbol{\beta}}_N$) uses the same settings as for the sequential method, except that only 10 random initializations are used.

2.2.4 Model Selection

Since the true value of K is generally unknown, we consider two separate criteria for estimating $\hat{K}$, as well as the other hyperparameter $\hat{\lambda}^{\text{stim}}$ for the simultaneous method: Bayesian information criterion (BIC) and cross-validation log-likelihood on held-out neurons (CVLL, considered in Section S2.5.1). For the simultaneous method, both criteria are applied to the approximate marginal log-likelihood for the hierarchical model, $\text{LL}_i \equiv \log \sum_{k=1}^K Z_{i,k} \approx \log P(\boldsymbol{y}_i | \boldsymbol{x}_i; \hat{\Omega}_k)$, as computed in (S2.3). For the sequential method, both criteria are applied

Data: $\mathbf{x}_i(t), \mathbf{y}_i(t)$ for $t = 1, ..., T_i,\ i = 1, ..., N,\ K,\ d^{\text{stim}}$

Result: $\hat{\pi}_k, \hat{\boldsymbol{\mu}}_k, \hat{\Sigma}_k$ for $k = 1, ..., K, \hat{k}_i, \hat{\boldsymbol{\beta}}_i$ for $i = 1, ..., N$

For $i \in \{1, \ldots, N\}$, $\hat{\boldsymbol{\beta}}_i \leftarrow \arg\max_{\boldsymbol{\beta}} P(\mathbf{y}_i|\mathbf{x}_i, \boldsymbol{\beta})$, using (2.2)

$\Omega_K \leftarrow$ GMM fit to $\hat{\boldsymbol{\beta}}_1, ..., \hat{\boldsymbol{\beta}}_N$

repeat

`/* E-Step: approximate distributions over latent variables` $k_i, \boldsymbol{\beta}_i$ `*/`

for $i = 1, ..., N$ **do**

$\boldsymbol{m}_{i,k} = \arg\max_{\boldsymbol{\beta}} \log P_{\text{joint}}(k, \boldsymbol{\beta}, \mathbf{y}_i|\mathbf{x}_i; \Omega_K)\ k = 1, ..., K$

$c_{i,k}^{-1} = -\text{diag}(\nabla^2_{\boldsymbol{\beta}} \log P_{\text{joint}}(k, \boldsymbol{\beta}, \mathbf{y}_i|\mathbf{x}_i; \Omega_K))\big|_{\boldsymbol{\beta}=\boldsymbol{m}_{i,k}}\ k = 1, ..., K$

$Z_{i,k} \leftarrow P_{\text{joint}}(k, \boldsymbol{\beta}, \mathbf{y}_i|\mathbf{x}_i; \Omega_K)\big|_{\boldsymbol{\beta}=\boldsymbol{m}_{i,k}} \sqrt{(2\pi)^{\dim(\boldsymbol{\beta})}|c_{i,k}|}\ k = 1, ..., K$

$\tilde{Z}_{i,k} \leftarrow Z_{i,k} / \sum_{k'=1}^{K} Z_{i,k'}\ k = 1, ..., K$

end

`/* M-Step: update estimates of global parameters` $\hat{\pi}_k, \hat{\boldsymbol{\mu}}_k, \hat{\Sigma}_k$ `*/`

for $k = 1, ..., K$ **do**

$\hat{\pi}_k \leftarrow \frac{1}{N} \sum_{i=1}^{N} \tilde{Z}_{i,k}$

$\hat{\boldsymbol{\mu}}_k^{\text{self}} \leftarrow \frac{\sum_{i=1}^{N} \tilde{Z}_{i,k} \boldsymbol{m}_{i,k}^{\text{self}}}{\sum_{i=1}^{N} \tilde{Z}_{i,k}}$

$\hat{\Sigma}_k^{\text{self}} \leftarrow \frac{\sum_{i=1}^{N} \tilde{Z}_{i,k} \text{diag}(c_{i,k}^{\text{self}} + \boldsymbol{m}_{i,k}^{\text{self}} \boldsymbol{m}_{i,k}^{\text{self}\top} - \hat{\boldsymbol{\mu}}_k^{\text{self}} \hat{\boldsymbol{\mu}}_k^{\text{self}\top})}{\sum_{i=1}^{N} \tilde{Z}_{i,k}}$

end

until *convergence*;

$\hat{k}_i \leftarrow \arg\max_k \tilde{Z}_{i,k},\ i = 1, ..., N$

$\hat{\boldsymbol{\beta}}_i \leftarrow \boldsymbol{m}_{i,\hat{k}_i},\ i = 1, ..., N$

Algorithm 2: EM algorithm for simultaneous model. See Sections S2.1.1 and S2.1.2 for detailed derivations of each step.

to the GMM log-likelihood, $\text{LL}_i \equiv \log \sum_{k=1}^{K} \hat{\pi}_k f(\hat{\boldsymbol{\beta}}_i; \hat{\boldsymbol{\mu}}_k, \hat{\Sigma}_k)$. (The λ hyperparameters were already selected using (2.4).) Both BIC and CVLL are evaluated for a range of K (and λ^{stim} for the simultaneous method), to select the optimal set of hyperparameters.

We use the following heuristic for BIC:

$$BIC = \sum_{i=1}^{N} \mathrm{LL}_i - \frac{\mathrm{dof}(\hat{\Omega}_K)}{2} \log(N). \tag{2.21}$$

Above, $\mathrm{dof}(\hat{\Omega}_K)$ is the number of degrees of freedom in $\hat{\Omega}_K$, equal to $\mathrm{dof}(\{\hat{\pi}_k\}) + \mathrm{dof}(\{\hat{\Sigma}_k\}) + \mathrm{dof}(\{\hat{\boldsymbol{\mu}}_k\}) = K - 1 + K * T^{\mathrm{self}} + K * T^{\mathrm{self}} = K * (2 * T^{\mathrm{self}} + 1) - 1$ (recall that $\sum_{k=1}^{K} \hat{\pi}_k = 1$ and that each $\hat{\Sigma}_k$ is diagonal).

We note here that these model selection criteria assume a specific form of model, and thus do not admit direct comparisons between the simultaneous and sequential methods, or with the alternative model we consider in Section S2.2. All plots displaying model criteria are normalized so that the best value attained is at 0 to facilitate judging the scale of the plot. All direct comparisons between methods must therefore rely on the evaluation metrics we consider in the following Section.

2.2.5 Evaluation Metrics

Throughout the results, we use several metrics to assess how accurately each method estimates clustering and parameters.

To assess clustering, we use the adjusted Rand score (ARS) (see [60] 25.1.2.2), which compares two labelings of a set of neurons. The ARS is symmetric in its inputs, equals 1 if the two labelings agree perfectly, and equals 0 if the two labelings are as similar as expected by chance. In Section 2.3.1, we compute ARS to compare estimated clusters to ground truth labels of simulated data. In Section 2.3.2, we use ARS to compare two sets of estimated clusters, obtained using two different sets of training neurons. This gives us a measure of how robust the clustering is to the inclusion or exclusion of subsets of neurons from training.

To assess accuracy of parameter estimation, we use the root-mean-squared error (RMS), $\sqrt{\frac{1}{T^{\mathrm{stim}}} \sum_{t=1}^{T^{\mathrm{stim}}} (\beta^{\mathrm{stim}}(t) - \hat{\beta}^{\mathrm{stim}}(t))^2}$, when ground truth is available. For the self-interaction filter, we exclude coefficients whose true value is less than -4, as parameter inaccuracies in this regime have a minimal effect on the likelihood.

When the ground truth is not available, we use two separate metrics to evaluate GLM parameter estimates for a given neuron based on how well they explain held-out test data. We use the average negative log-likelihood (ANLL) to evaluate the performance of each GLM

on test data, in terms of the same quantity that was used to train it,

$$ANLL_i \equiv -\frac{1}{T_i^{\text{test}}} \log(P_{SC}(\mathbf{y}_i^{\text{test}}|\mathbf{x}_i^{\text{test}}; \hat{\boldsymbol{\beta}}_i)). \tag{2.22}$$

We also compute the explained variance ratio (EV_{ratio}), following [81]. To compute EV_{ratio}, we require that the test data consists of neural responses to repeated presentations of a single stimulus waveform. Calculating the EV_{ratio} for the parameter estimate of a given neuron then consists of three steps:

1. *Simulate the response of a GLM with the estimated parameters to the test stimulus.* Because our model is stochastic, we simulate a large number (3000) of responses by repeatedly sampling from (2.2), and average the simulated spike trains to produce the *average model prediction.*

2. *Smooth the spiking response to each presentation of the test stimulus, as well as the average model prediction, with a Gaussian kernel with a standard deviation of 10ms.* We refer to the neuron's smoothed response to the jth of P presentations of the test stimulus as $stPSTH_j$, which is an abbreviation for *single peristimulus time histogram.* We average these quantities to obtain the smoothed average response, defined as $PSTH_D \equiv \frac{1}{P}\sum_{j=1}^{P} stPSTH_j$. We let $PSTH_M$ denote the smoothed average model prediction.

3. *Define EV_{ratio} as the ratio of the trial averaged explained variance between $PSTH_M$ and $stPSTH_j$ to that between $PSTH_D$ and $stPSTH_j$.* That is,

$$EV_{ratio} = \frac{\sum_{j=1}^{P} EV(stPSTH_j, PSTH_M)}{\sum_{j=1}^{P} EV(stPSTH_j, PSTH_D)}, \tag{2.23}$$

where

$$EV(PSTH_1, PSTH_2) = \frac{var(PSTH_1) + var(PSTH_2) - var(PSTH_1 - PSTH_2)}{var(PSTH_1) + var(PSTH_2)}. \tag{2.24}$$

Note that (2.24) will equal 1 if $PSTH_1 = PSTH_2$, and will equal zero if $PSTH_1$ and $PSTH_2$ are independent; it can be seen as the scaled covariance across time points between the $PSTH_1$ and $PSTH_2$. A very high value of EV_{ratio} (near 1) would thus indicate that the average model prediction covaries across time with the individual trial responses almost as well as the trial-averaged response (and thus that the model fits the data well); a very low value (near 0) would indicate that the average model prediction has low covariance with the individual trial responses.

2.3 Results

In Section 2.2, we detailed two methods to identify cell types from neural spiking responses, the sequential approach and our simultaneous approach. The sequential approach consists of individually tuning the parameters of a generalized linear model (GLM, Section 2.2.2) to fit each neuron's responses and then clustering those parameters (Section 2.2.2). The simultaneous approach makes use of a hierarchical probabilistic framework to simultaneously estimate both the GLM parameters and their cluster labels (Section 2.2.3). Crucially, the simultaneous approach "borrows strength" from other neurons' data, allowing for improved estimates of the GLM parameters, in addition to improved estimates of cell types.

In this section, we first demonstrate that the simultaneous method recovers the ground truth parameters used to generate simulated data (Section 2.3.1). We then apply it to the Allen Cell Types Database collected by the Allen Institute for Brain Science [1] (Section 2.3.2). Specifically, we model the spiking response of chemically-isolated neurons in mouse primary visual cortex to an injected pink noise current. This dataset provides an excellent benchmark for these methods, as it contains large amounts of high-quality data collected from many different neurons.

Importantly, these neurons were chosen from a variety of different transgenic lines in an attempt to sample a diverse set of cells that will be useful for characterizing cell types. To this end, the transgenic line of each recorded neuron has been made available in the dataset, as well as information about the cell's location and morphology. This enables us to compare the putative cell types we extract to these other recorded properties of neurons, (Section 2.3.2, see [81] for a similar analysis on the same data).

We demonstrate that our novel simultaneous method produces single-cell models that generally predict spiking responses in the Allen Cell Types Database better than individually fitted models. We further show that the choice of metric used to quantify this improvement leads to different implications regarding which neurons' models are most improved, and how the improvement scales with the number of neurons and amount of data used per neuron. We demonstrate that the clusters of Allen Cell Types Database neurons discovered by our simultaneous method have properties that generally make them amenable to interpretation. We also interpret the discovered clusters by comparing membership in each with the available information about each cell's transgenic line, location, and morphology. For all of these analyses, we use the sequential method and its individually fit GLMs for comparison.

Throughout, we fix the dimensions of $\boldsymbol{\beta}_i^{\text{stim}}$ and $\boldsymbol{\beta}_i^{\text{self}}$ to $T^{\text{stim}} = 10$ and $T^{\text{self}} = 20$, respectively, and use a downsampling factor of $d^{stim} = 5$ for stimulus filters. We use 2 ms time bins, so this corresponds to filtering the last 40 ms of spiking history and the last 100 ms of stimulus.

2.3.1 Application to simulated datasets shows that the simultaneous method accurately recovers ground truth parameters

First, we compare how well each method recovers the true parameters of a generative model from simulated data. To create these data, we use the same stimuli that were presented in the Allen Cell Types Database, and simulate responses of GLMs with identical, fixed stimulus filters $\boldsymbol{\beta}_i^{\text{stim}}$ and offsets $\beta_i^0 = -5$, and with self-interaction filters $\boldsymbol{\beta}_i^{\text{self}}$ sampled from a GMM with equal cluster sizes $\pi_1 = \cdots = \pi_K$ (for simplicity we just simulate 40 neurons per cluster) and fixed $\boldsymbol{\mu}_k^{\text{self}}$ (see Fig 2.2A,B for fixed parameters, Section S2.4 for more details). We vary the number of clusters K and the within-cluster variance σ^2 ($\sigma^2 I = \Sigma_k^{\text{self}},\ \forall k \in \{1, ..., K\}$).

We fit the simultaneous and sequential methods to the data, and then plot several accuracy metrics for $\sigma \in [10^{-2}, 10^{-5/6}]$. We use an "oracle" approach to determine the hyperparameters for both methods, using the K that was used to generate the data, and λ hyperparameters that most accurately recover $\boldsymbol{\beta}$ (by RMS, see Section 2.2.5) on 10 held-out "oracle" datasets. Because only the sequential method applies ℓ_2 regularization to self-interaction filters, those

estimates particularly are biased towards 0 (for example, see Fig 2.2B), especially for very negative filter coefficients. However, the exact magnitude of such negative coefficients does not affect the GLM's spiking likelihood much, so we exclude these coefficient estimates from measures of self-interaction filter accuracy (see Section 2.2.5 for details).

Our simultaneous method outperforms the sequential method in terms of clustering accuracy (Fig 2.2C), recovery of single-cell parameters β_i^{stim} (Fig 2.2D) and β_i^{self} (Fig 2.2E), and estimation of within-cluster variance (Fig 2.2F). Note that as $\sigma \to 0$, the estimated cluster spread saturates, reflecting the width of the posterior distribution of the clustered GLM parameters $\boldsymbol{\beta}_i^{\text{self}}$. This saturating value is about $\sqrt{40}$ times lower for the simultaneous method because it essentially pools the data across all 40 simulated neurons in each cluster to estimate the posterior for a single one. These results demonstrate that, by borrowing strength across neurons, the simultaneous method provides better estimates of single-cell model parameters and cell types.

To determine how well each method can recover the true K, we evaluate the Bayesian information criterion (BIC) for both methods (Fig 2.3, see Section S2.5.1 for validation loss on held-out neurons). Here we use oracle to select the λ hyperparameters, but fit models with $K = 1, ..., 8$ and select the value of K with the highest BIC (see Section 2.2.4). We see in Fig 2.3 that the simultaneous method is better able to recover the true number of clusters.

2.3.2 Application to neural data shows that the simultaneous method is a useful tool for modeling and understanding real neural data

To evaluate the simultaneous method on neural data, we apply Algorithm 2 to spiking data recorded from 634 cells in mouse primary visual cortex (Allen Cell Types Database [1], region 'VISp'). These spikes are in response to repeated current injections of pink noise stimuli. Across the entire dataset, only two specific instantiations of pink noise are used, which we refer to as "Noise 1" and "Noise 2"; we selected only cells that received at least three presentations of both, and were labeled with a transgenic cre-line. In order to evaluate how well our method generalizes to new stimuli, all fitting and model selection was done on Noise 1 stimuli, and Noise 2 stimuli were withheld for testing. In applying Algorithm 1, we

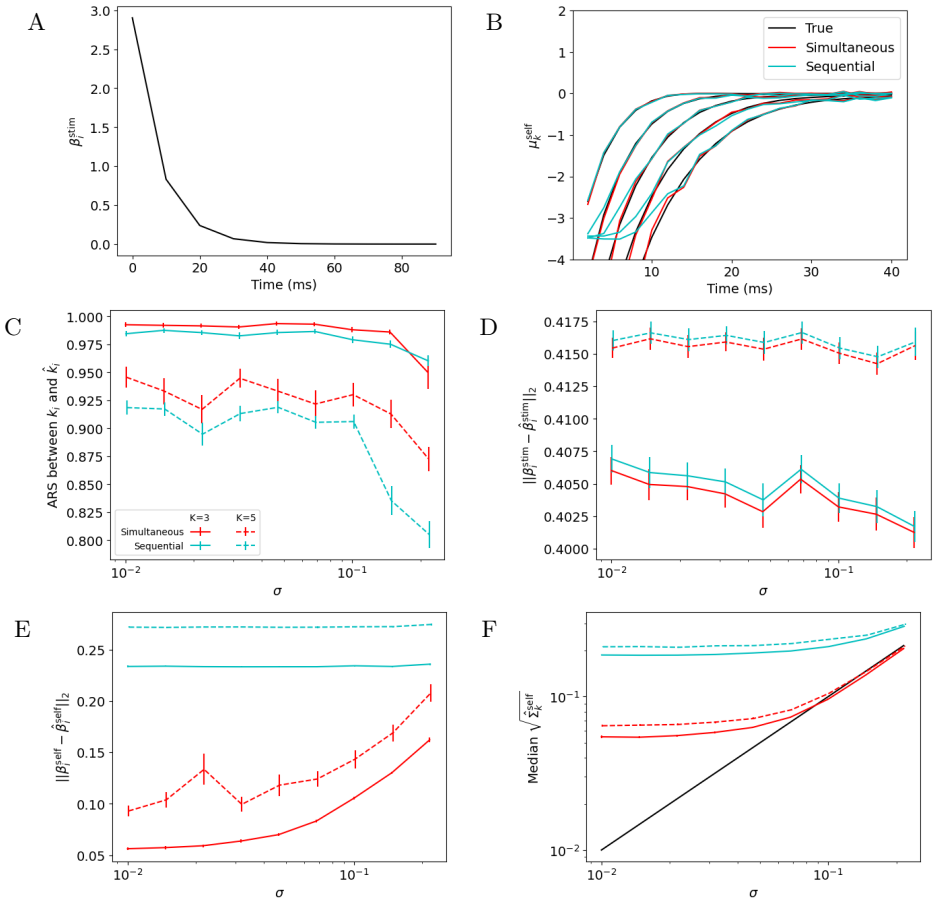


Figure 2.2: **Performance of sequential and simultaneous methods on simulated data.**

A: The true stimulus filter $\boldsymbol{\beta}_i^{\text{stim}}$ for each simulated neuron.

B: The true cluster means $\boldsymbol{\mu}_k^{\text{self}}$ used to generate simulated datasets, and those estimated by the sequential and simultaneous methods, $\hat{\boldsymbol{\mu}}_k^{\text{self}}$, fit with the correct $K = 5$.

C-F: Mean ± SEM over 50 simulated datasets of accuracy measures, as a function of σ and shown for both $K = 3$ and $K = 5$. When $K = 3$, the three clusters with leftmost $\boldsymbol{\mu}_k^{\text{self}}$ in panel B are used. For each condition (K and σ) and for each measure of accuracy, the simultaneous method's performance is statistically significantly better than that of the sequential method, except for ARS with $K = 3$ and $\sigma = 10^{-5/6}$ (evaluated using the Wilcoxon signed rank test: uncorrected P-value$<$0.002).

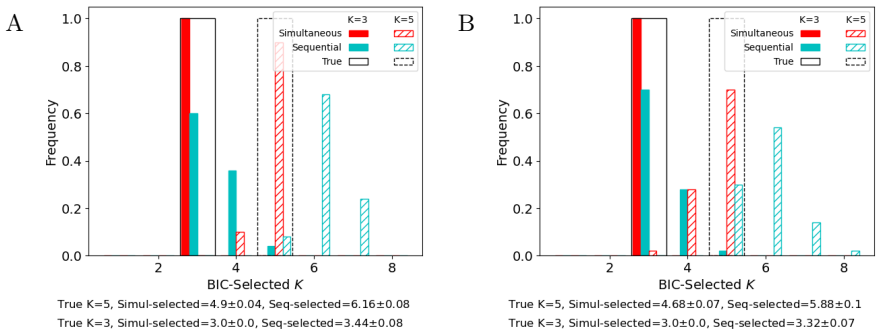


Figure 2.3: **Model selection of $\hat{K}$.** Frequencies of $\hat{K}$ estimated via Bayesian information criterion over 50 simulated datasets with the same μ_k as in Fig 2.2, and $\sigma = 10^{-2}$ (A) or $10^{-5/6}$ (B), the maximum value that does not result in degenerate simulations. Black lines indicate the true value of K. The summary below each plot reports the mean ± SEM of the estimated value of $\hat{K}$ across the 50 datasets, for each value of K and each method.

partitioned each neuron's data into equally-sized bins of adjacent time points so that each element of the partition contains a separate presentation of the Noise 1 stimulus.

We used BIC to perform model selection with the simultaneous method, selecting $\hat{K} = 12$ (Fig 2.4A, see Section S2.5.1 for model selection using cross-validation on held-out neurons). The discovered cell types are distinct, have smoothed self-interaction filters, and suggest different computational roles for some of the types, as their self-interaction filters have qualitatively different shapes (Fig 2.4B).

We compare these results to those obtained from individually fitting GLM models for each neuron and then clustering those models' self-interaction filters $\boldsymbol{\beta}_i^{\text{self}}$ (the sequential method). The GLM models are fitted with an ℓ_2 regularization, with hyperparameters selected using cross-validation, where individual presentations of the stimulus are used to define the partition of time bins (see (2.4)). Autocorrelations of the residuals (after subtracting the smoothed trial average) indicated very low temporal correlations and the trial correlations of residuals were low for the majority of cells. We find that the estimated $\hat{K}$ is much higher, at least 19 (Fig 2.4C). The resulting cluster centers for the $K = 12$ (same K as Fig 2.4B) are all bunched on top of each other near 0 and have higher variances (Fig 2.4D). We hypothesize that this

difference arises from the simultaneous method's ability to borrow strength across different neurons in the same cluster, pulling their parameters closer together toward an improved estimate of their center. By contrast, the sequential method weakly pulls all parameter estimates into a region near 0 with its standard ℓ_2 penalty, and then fits highly overlapping clusters that are densely packed to fill this region. The cluster weights, π_k, shown in the legend, also support this description: the simultaneous method's clusters have much more variable weights (<0.03-0.2), as well as shapes, compared to the sequential method's (weights 0.06-0.13).

Generalization performance demonstrates improved parameter estimates

In order to demonstrate that the differences in parameter estimates between the simultaneous and sequential methods reflect meaningful differences in the descriptions of the underlying biological system, we show that the simultaneous method generalizes better to held-out data. We examine two types of generalization to assess how well each method accomplishes each goal (Section 2.2.1): namely, generalization to held-out time bins and to held-out neurons. Without knowledge of ground truth single-cell parameters or cell types, this is the best assessment we can make of each method's accuracy. To accomplish this, we partition the neurons into four sets, using each in turn for evaluation after fitting and performing model selection with BIC on the other three, in addition to withholding Noise 2 responses from training.

The simultaneous method discovers individual parameters for each neuron $\hat{\boldsymbol{\beta}}_i$ (2.20) that allow for better prediction of the held-out responses to Noise 2 than those found by the sequential method (2.3). We measure this using each fitted GLM's average negative log-likelihood (ANLL) of the held-out data ((2.22), Fig 2.5A), as well as its explained variance ratio, (EV_{ratio}, how well the mean smoothed model prediction captures the variability in smoothed spike trains across trials, relative to the true cross-trial mean, see (2.23), Fig 2.5C). Likewise, we use ARS to measure stability of the cluster assignments $\hat{k}_i$ when each of the folds of neurons are held-out from training, and find that the simultaneous method generally finds more stable clusters (positive values in the cells of Fig 2.5F).

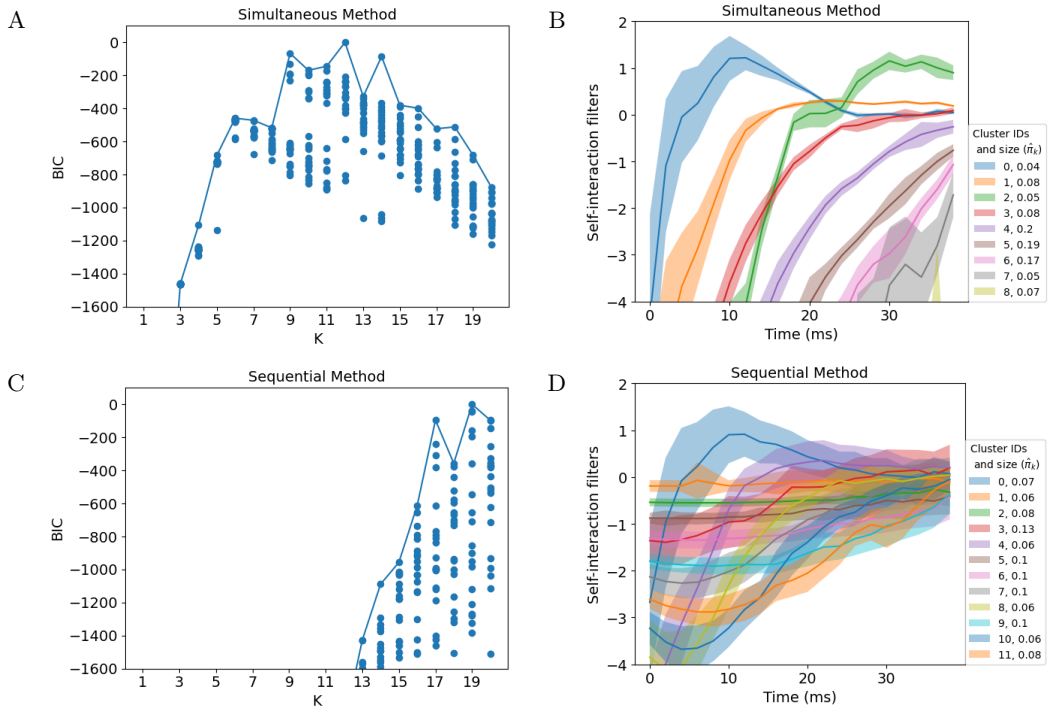


Figure 2.4: **Allen Cell Types Database: The simultaneous method explains the data with a smaller number of less overlapping clusters.** A: BIC for the simultaneous method over a range of K; BIC determines $\hat{K} = 12$ as optimal. Each dot reports the result of running Algorithm 2 from a different random initialization. B: Cluster centers $\boldsymbol{\mu}_k$ of self-interaction filters fit to data using the simultaneous method; shaded region is $\pm\sqrt{diag(\Sigma_k)}$. Only the 9 clusters with at least 20 neurons are shown from the model with BIC-selected $K = 12$. C: By contrast, the standard BIC of a GMM fit to individually fitted self-interaction filters (the sequential method) suggests an optimal $\hat{K}$ of at least 19. Each dot reports the result of performing the GMM fit, (2.5), from a different random initialization. D: The clusters of self-interaction filters with at least 20 neurons found by the sequential approach with $K = 12$ are bunched up closer to the origin, such that the clusters overlap significantly.

Next, we restricted our analysis to subsets of training trials and subsets of neurons. Recall that each presentation of the Noise 1 stimulus was applied at least three times; we subset

the trials so that exactly one, two, or three presentations of Noise 1 was retained per neuron. Furthermore, earlier we split the neurons into four folds and fit the model on three of the four folds; here we instead fit the model on only one, two, or three of the four folds. Finally, we used all Noise 2 presentations for all of the remaining folds as test data.

We find that the relative improvement of the simultaneous method is greater when data from more neurons is used (Fig 2.5B, D, and F, although not significantly in F). This has a simple interpretation: providing more neurons allows for more borrowing of strength to improve parameter estimates, and thus improves all generalization measures.

The results of varying the number of stimulus presentations per neuron are more ambiguous: ARS benefits the most from the use of the simultaneous method rather than the sequential method when there are fewer presentations, ANLL when there are more, and effects for EV_{ratio} are not significant. These discrepancies can be linked to the observation that different populations of neurons are responsible for the improvement of each metric between methods (Fig 2.5E). Neurons with fewer spikes in their response will naturally have a good ANLL (see Fig 2.5A; it is easy to predict zero spikes) and bad EV_{ratio} (see Fig 2.5C; the inter-spike intervals are very high, so relative jitter in predicted spikes hurts more). Therefore, the simultaneous method improves the EV_{ratio} of these neurons the most, while it improves the ANLL of those with many spikes the most, as in each case those neurons leave the most room for improvement. Given that different populations of neurons are responsible for changes in ANLL and EV_{ratio}, it is no surprise that these metrics scale differently with the number of presentations used for training. One potential reason for this is that one of these populations may have a less variable response to the repeated stimulus, yielding a narrower posterior over GLM parameters, than the other. ARS is not computed on a neuron-by-neuron basis, so we cannot easily attribute its difference in scaling to a different population of neurons, but it is clearly assessing performance in a very different way than ANLL and EV_{ratio}. Together, these results show that the choice of evaluation metric can drastically affect conclusions about model performance and how it scales with dataset sizes, suggesting that the best practice is to consider many metrics, as we do here.

Additional details about Fig 2.5 are provided in Section S2.3.

Figure 2.5: **Allen Cell Types Database generalization performance: the simultaneous method produces single-cell models and clusterings that generalize better, especially when fitted to more neurons.** Additional details about this figure are available in Section C in S2.3.

A: ANLL (lower is better) for each held-out neuron's single-cell model, evaluated on responses to the test stimulus (Noise 2), using the MAP $\hat{\boldsymbol{\beta}}_i$ (2.20) of the simultaneous method with (hyper)parameters $K, \lambda^{\text{stim}}, \hat{\Omega}_K$ estimated from the training neurons, versus those found by the sequential method. Color encodes number of spikes for each neuron in the evaluation data.

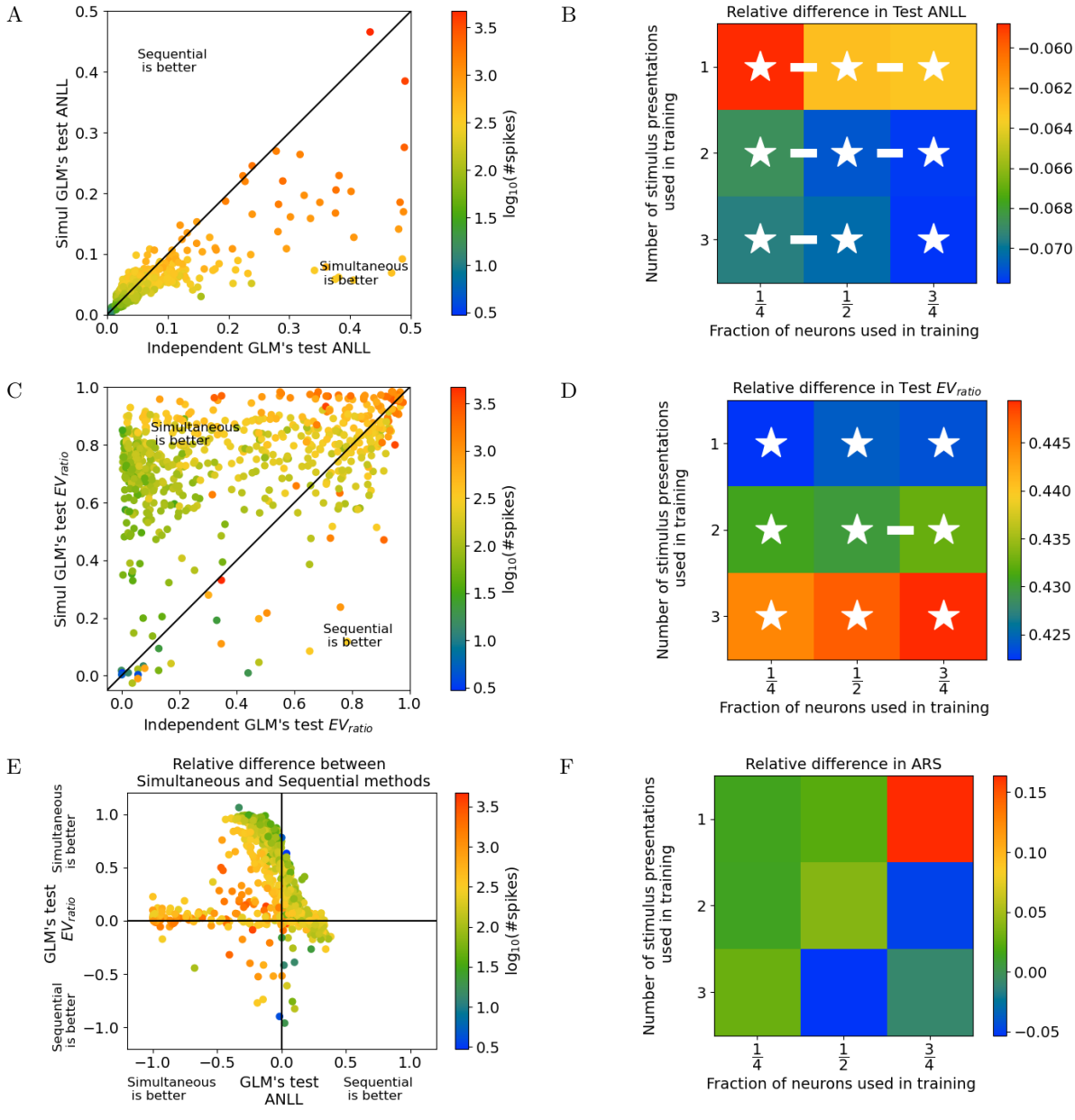
B: Median relative difference between methods in ANLL of held-out neurons, evaluated on responses to the test stimulus, as a function of how many neurons and how much data from each were used in training (more negative values indicate that the simultaneous method is better). White asterisks indicate a significant relative difference; white bars indicate adjacent cases of training data subselection where the relative differences were significantly different. Differences pooled across all vertically (horizontally) adjacent conditions showed a significant, $p = 3 \times 10^{-4}$, ($p = 5 \times 10^{-26}$) trend, with more presentations (neurons) yielding greater improvement by the simultaneous method.

C: Same analysis as A, but with EV_{ratio} (see (2.23); higher values indicate that the simultaneous method is better).

D: Same analysis as B, but with EV_{ratio}. Pooled vertical differences showed no significant trend ($p > 0.1$); horizontal differences showed a significant ($p = 5 \times 10^{-3}$) trend, with more neurons yielding greater improvement by the simultaneous method.

E: Relative differences between methods of EV_{ratio} (shown in C) versus ANLL (shown in A); color encodes number of spikes. Neurons with many (few) spikes show only improved ANLL (EV_{ratio}) in the simultaneous method.

F: Same analysis as B, but for the similarity of cluster assignments $\hat{k}_i$ between model fits with different held-out neurons, measured by ARS (more positive values indicate that the simultaneous method is better). Pooled vertical differences showed a significant ($p = 5 \times 10^{-2}$) trend, with fewer presentations yielding greater improvement by the simultaneous method; horizontal differences showed no significant trend ($p > 0.1$).


A
Sequential is better
Simultaneous is better
Simul GLM's test ANLL
Independent GLM's test ANLL
log10(#spikes)
B
Relative difference in Test ANLL
Number of stimulus presentations used in training
Fraction of neurons used in training
C
Simultaneous is better
Sequential is better
Simul GLM's test EVratio
Independent GLM's test EVratio
log10(#spikes)
D
Relative difference in Test EVratio
Number of stimulus presentations used in training
Fraction of neurons used in training
E
Relative difference between Simultaneous and Sequential methods
GLM's test EVratio
GLM's test ANLL
Simultaneous is better
Sequential is better
log10(#spikes)
F
Relative difference in ARS
Number of stimulus presentations used in training
Fraction of neurons used in training

Comparison to metadata suggests relationships between discovered cell types and measured genetic and anatomical properties of neurons

The Allen Cell Types Database contains limited morphological, locational, and transcriptomic information about each cell in addition to the electrophysiological recordings. We will now investigate whether the electrophysiological cell types that we discover are related to these metadata, in the same spirit as [79] and [81]. Note, however, that these metadata do not constitute a "ground truth" for functional cell types as we have defined them: they merely provide different dimensions along which neurons can be clustered; any similarities (or lack thereof) between discovered types and the metadata do not suggest better or worse performance of the clustering algorithm. However, as none of the metadata is supplied to either clustering algorithm, any similarities that do exist may provide insights into how functional properties of cells relate to morphological, transcriptomic, or location factors.

Many metadata labels belonged to certain clusters discovered by the simultaneous method much more or less often than expected by chance: namely, dendrite types, most transgenic lines, and some cortical layers (Fig 2.6A). Dendrite types and especially cortical layer reflect a quantization of an inherently continuous variable, and accordingly these plots display a horizontal gradient, which is to be expected if each cluster ID has some spread in distribution over the underlying continuous variable. It is worth noting that many columns (attributes) display a vertical gradient (cluster IDs are ordered by the mean values of their estimated self-interaction filters so that $\sum_{t=1}^{T^{\text{self}}} \hat{\mu}_1^{\text{self}}(t) > \cdots > \sum_{t=1}^{T^{\text{self}}} \hat{\mu}_K^{\text{self}}(t)$, and used in Figs 2.4B and D and 2.6). This is consistent with the reality that our notion of cell types is also a quantization of an inherently continuous space. Both such quantizations, however, can be useful - for example, certain layers (2/3, 4, and 6b) and transgenic lines (Pvalb, Tlx3, and many others to a lesser extent) are strongly overrepresented in a very small number of clusters and strongly underrepresented in the others. These results suggest that there are meaningful relationships between the functional cell types discovered by our method and these genetic and anatomical factors, beyond what would be expected if cells were uniformly distributed throughout the continuous spaces of functional parameters, dendrite density, depth, and transgenic expression.

Results obtained using the sequential method's clusters are generally similar, albeit reflecting differences in cluster structure seen in Fig 2.4. Because the cluster means are relatively bunched, the ordering of their indices, $\sum_{t=1}^{T^{\text{self}}} \hat{\mu}_1^{\text{self}}(t) > \cdots > \sum_{t=1}^{T^{\text{self}}} \hat{\mu}_K^{\text{self}}(t)$, is messier than that of the simultaneous method, limiting our ability to observe the vertical gradients described above.

We see from Fig 2.6 that certain attributes tend to be characterized by a small number of clusters in both the simultaneous and sequential methods. For example, neurons in the Pvalb cre line tend to be assigned to clusters 0 and 1 by the simultaneous method, and to clusters 0 and 4 by the sequential method.

2.4 Discussion

In this work, we leverage a hierarchical probabilistic framework to advance our ability to identify cell types from neural responses and improve our models of individual neurons. We find that, even applied to relatively noiseless *in vitro* recordings, our method provides substantial gains over independently fit single-cell models, in terms of its ability to predict the response to a held-out stimulus. We demonstrated that these gains increase as our method is applied to datasets of increasingly many neurons, and highlighted the importance of using multiple evaluation metrics. Compared to clustering individually fitted neuron models, our method discovers cell types that are more robust to the exclusion of different groups of neurons from training, are more amenable to interpretation, and reveal trends of scientific interest in terms of correlations between cluster membership and other information available about each neuron.

We argue that these improvements stem from the simultaneous method's ability to "borrow strength" between the spike trains of different neurons. That is, while both approaches can be thought of as applying regularization to the estimation of single-cell parameters for each neuron, $\hat{\boldsymbol{\beta}}_i$, in the simultaneous case this regularization is informed by the spiking responses of other neurons, bringing more data to bear on the estimation problem.

Compared with using sequential approaches that separately fit single neuron models and then cluster their parameters, using this hierarchical generative model framework requires more advanced statistical methods for parameter estimation and model selection.

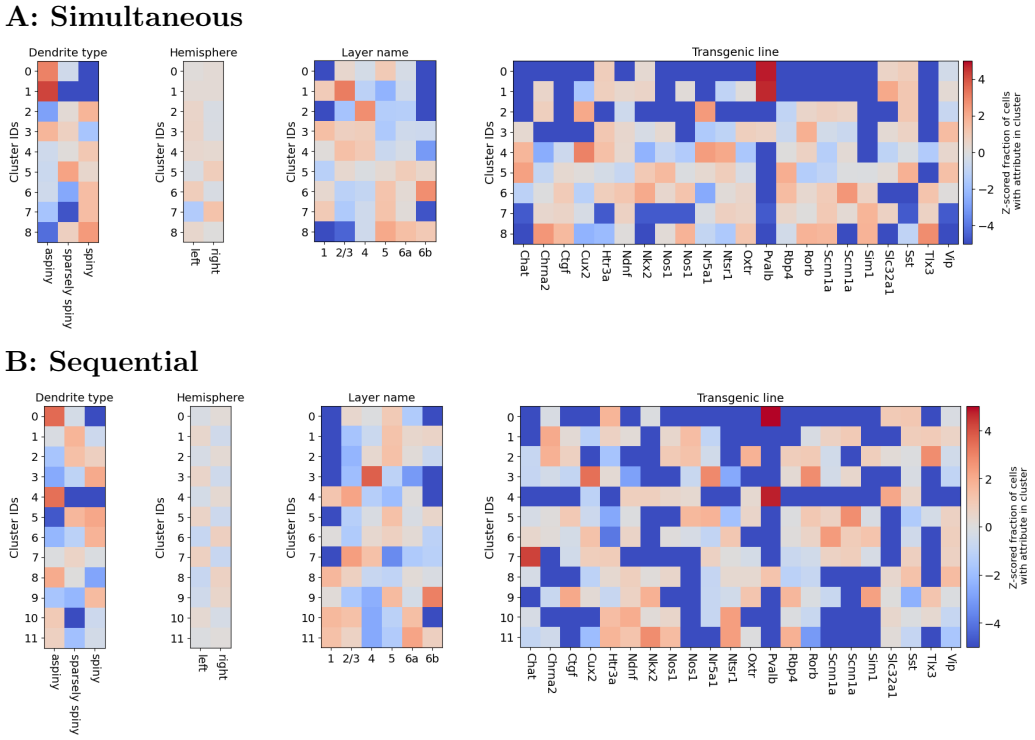


Figure 2.6: **Allen Cell Types Database metadata is related to discovered cell types.** Z-scored fraction of cells with an attribute in each cluster. Cluster identities in panel A (B) are the same as in Fig 2.4B (D), obtained using the simultaneous (sequential) method (fitted with BIC-selected $K = 12$ clusters, showing only clusters with at least 20 neurons). Attributes are spiny or aspiny dendrites, location (hemisphere and cortical layer), and Cre line. Z-scores are calculated as $Z_i^{(a)} = (\hat{p}_i^{(a)} - \hat{p}_i)/\sqrt{\hat{p}_i^{(a)}(1-\hat{p}_i^{(a)})/N^{(a)} + \hat{p}_i(1-\hat{p}_i)/N}$, where $\hat{p}_i$ is the empirical probability that a cell is in cluster i and $\hat{p}_i^{(a)}$ is the empirical probability that a cell with attribute a is in cluster i, N is the number of cells, and $N^{(a)}$ is the number of cells with attribute a.

However, appropriate choices of models and approximations allow for tractable and improved parameter estimation. We make particular choices for the single-cell response model (GLM), which parameters are related to cell type (self-interaction filters, $\boldsymbol{\beta}_i^{\text{self}}$), and how (normal distribution), but our algorithm can generalize to any choices for these. However, these choices, along with the choice to use a Gaussian approximation for variational inference,

allow for a simpler, faster algorithm that can make use of off-the-shelf convex optimizers.

Overall, our method provides an unsupervised approach to the categorization of dynamical properties that differ among neurons. This approach complements well-established dynamical categories such as "Type I" and "Type II" neurons, as discussed in e.g. [31, 45, 73] that are established based on mathematical properties of their underlying differential equations models: specifically, the bifurcation that leads to a neuron's spiking. An interesting avenue for future work would be to compare these categories with what we find with the present approach.

Hierarchical generative models can easily be modified to incorporate multimodal data (as in [47]), so long as appropriate distributions for each modality can be specified. Thus our method could be used to identify multimodal cell types in terms of their transcriptomic and/or morphological properties as well as their functional ones, as an alternate approach to previous work ([34, 36]). Indeed, our approach can be easily applied to clustering problems outside of neuroscience, whenever there exists a cluster structure among individual entities, and each entity generates many samples of data, requiring only a change of how the data is described by a GLM.

That our approach provides the greatest performance improvements with data from many neurons may make it well-suited for application to *in vivo* recordings of brain activity. Modern recording technologies allow experimenters to simultaneously measure the activity of hundreds to thousands of neurons at a time. However, the noise inherent to the data and the effect of unmeasured inputs on *in vivo* activity limit the accuracy of fitted parameters of single-cell neural dynamical models. In this work, our simultaneous method improves the accuracy of fitted parameters, suggesting that it may be able to overcome these challenges.

Applying this algorithm to *in vivo* neural recordings is an exciting avenue for future work. This would require expanding the framework to include connections between observed neurons and noisy inputs from unobserved ones. There has been much research on cell-type-specific connectivity suggesting that certain types are more likely to synapse on each other, and affect each other in stereotyped ways [10, 12, 21, 46, 78]. Including aspects of connectivity and/or noisy inputs with the cell-type-specific parameters $\boldsymbol{\beta}$ may thus lead to improved estimates of network effects on neural activity. For example, Jonas and Kording use a similar

generative model and simultaneous approach that predicts connectivity between cells as well as data about each cell's position [47]. Although their method was applied to data about measured anatomical connectivity, such approaches could be modified to work with inferred connectivity instead. Given the long-term goal of identifying functional cell types from *in vivo* data, using this method to identify cell types according to inferred intrinsic electrophysiological parameters together with inferred connectivity parameters is a promising direction.

Such network models with functional cell types that such an algorithm may produce can complement the growing body of theoretical literature regarding such networks [3, 29]. Such work provides ideas about how to interpret cell-type-specific properties and interactions that our method may discover in the context of a neural circuit; in return, our method's discovery of cell-type structure in neural data would highlight the biological relevance of such theoretical work.

2.5 Chapter Acknowledgements

We would like to acknowledge funding from NIH R01DA047869 and a Simons Investigator Award in Mathematical Modeling of Living Systems to Daniela Witten, and NIH R01EB026908 to Daniela Witten and Michael A. Buice. We thank the Allen Institute founder, Paul G. Allen, for his vision, encouragement, and support.

Chapter S2

SUPPLEMENTARY INFORMATION FOR MODELING FUNCTIONAL CELL TYPES IN SPIKE TRAIN DATA

Code Availability All code used in this study is available at `https://github.com/zdeblick/ClusteredGLMs`.

S2.1 Extended EM Methods

Here we provide a more detailed account of how our adapted EM algorithm fits our hierarchical generative model (2.11) to data.

S2.1.1 E-Step

Our expectation (E) step consists of finding the $Z_{i,k}, \boldsymbol{m}_{i,k}, c_{i,k}$ that best approximate the posterior distribution over the latent variables, $Q_i(k, \boldsymbol{\beta})$ (2.17).

For fixed i and k, we first set $\boldsymbol{m}_{i,k}$ so that the modes of the left and right hand sides of (2.14) are equal. To do this, we apply the trust-region Newton-conjugate-gradient algorithm to solve the convex optimization problem

$$\boldsymbol{m}_{i,k} = \arg\max_{\boldsymbol{\beta}} \log P_{\text{joint}}(k, \boldsymbol{\beta}, \mathbf{y}_i | \mathbf{x}_i; \hat{\Omega}_K). \tag{S2.1}$$

Next, so that the Hessians of the logs of the left and right hand sides of (2.14) are equal, we compute

$$\tilde{c}_{i,k}^{-1} = -\nabla^2_{\boldsymbol{\beta}} \log P_{\text{joint}}(k, \boldsymbol{\beta}, \mathbf{y}_i | \mathbf{x}_i; \hat{\Omega}_K) \Big|_{\boldsymbol{\beta}=\boldsymbol{m}_{i,k}}. \tag{S2.2}$$

The Hessian of a GLM log-likelihood function is often used during optimization (see [60] 9.3.2 for details), and we make use of the `statsmodels` package to efficiently compute it. We further approximate $\tilde{c}_{i,k}^{-1}$ by retaining only its diagonal elements, i.e. $c_{i,k}^{-1} = \text{diag}(\tilde{c}_{i,k}^{-1})$.

Finally, noting that $f(\boldsymbol{\beta}; \boldsymbol{m}_{i,k}, c_{i,k})\Big|_{\boldsymbol{\beta}=\boldsymbol{m}_{i,k}} = \frac{1}{\sqrt{(2\pi)^{T^{\text{stim}}+T^{\text{self}}+1}|c_{i,k}|}}$, we take

$$Z_{i,k} = P_{\text{joint}}(k, \boldsymbol{\beta}, \mathbf{y}_i|\mathbf{x}_i; \hat{\Omega}_K)\Big|_{\boldsymbol{\beta}=\boldsymbol{m}_{i,k}} \sqrt{(2\pi)^{T^{\text{stim}}+T^{\text{self}}+1}|c_{i,k}|}, \tag{S2.3}$$

where $|c_{i,k}|$ denotes the determinant, so that the left and right hand sides of (2.14) are equal when $\boldsymbol{\beta} = \boldsymbol{m}_{i,k}$. Collectively, (S2.1), (S2.2), and (S2.3) are known as the Laplace approximation (see [60] 8.4.1).

This concludes the set of operations needed to perform the E-step.

S2.1.2 M-Step

The maximization (M) step consists of finding Ω_K that maximizes the lower bound on the log-likelihood, (2.18). In light of (2.8), the bound being maximized in the M-step factors as follows:

$$\begin{aligned}
&\sum_{i=1}^{N}\sum_{k=1}^{K}\int \tilde{Z}_{i,k} f(\boldsymbol{\beta}; \boldsymbol{m}_{i,k}, c_{i,k}) \log P_{\text{joint}}(k, \boldsymbol{\beta}, \mathbf{y}_i|\mathbf{x}_i; \Omega_K) d\boldsymbol{\beta} \\
= &\sum_{i=1}^{N}\sum_{k=1}^{K}\int \tilde{Z}_{i,k} f(\boldsymbol{\beta}; \boldsymbol{m}_{i,k}, c_{i,k}) [\log P_{\text{SC}}(\mathbf{y}_i|\mathbf{x}_i, \boldsymbol{\beta}) + \log \pi_k + \log f(\boldsymbol{\beta}; \boldsymbol{\mu}_k, \Sigma_k)] d\boldsymbol{\beta}.
\end{aligned}$$

Absorbing into C terms that do not depend on the optimization variables Ω_K, this equals

$$\sum_{i=1}^{N}\sum_{k=1}^{K}\int \tilde{Z}_{i,k} f(\boldsymbol{\beta}; \boldsymbol{m}_{i,k}, c_{i,k}) [\log \pi_k + \log f(\boldsymbol{\beta}; \boldsymbol{\mu}_k, \Sigma_k)] d\boldsymbol{\beta} + C.$$

Rearranging the sums and integrals, this equals

$$\sum_{k=1}^{K}\int [\sum_{i=1}^{N} \tilde{Z}_{i,k} f(\boldsymbol{\beta}; \boldsymbol{m}_{i,k}, c_{i,k})] [\log \pi_k + \log f(\boldsymbol{\beta}; \boldsymbol{\mu}_k, \Sigma_k)] d\boldsymbol{\beta} + C.$$

Distributing the product and using $\int f(\boldsymbol{\beta}; \boldsymbol{m}_{i,k}, c_{i,k}) d\boldsymbol{\beta} = 1$ to simplify the first term, this equals

$$\sum_{k=1}^{K}[\sum_{i=1}^{N} \tilde{Z}_{i,k}] \log \pi_k + \sum_{k=1}^{K}\int [\sum_{i=1}^{N} \tilde{Z}_{i,k} f(\boldsymbol{\beta}; \boldsymbol{m}_{i,k}, c_{i,k})] \log f(\boldsymbol{\beta}; \boldsymbol{\mu}_k, \Sigma_k) d\boldsymbol{\beta} + C. \tag{S2.4}$$

We maximize the first term in (S2.4), $\sum_{k=1}^{K}[\sum_{i=1}^{N} \tilde{Z}_{i,k}] \log \pi_k$, with respect to $\pi_1, ..., \pi_K$, subject to $\pi_k \geq 0, k = 1, ..., K$ and $\sum_{k=1}^{K} \pi_k = 1$. Algebraic manipulation reveals that

$$\hat{\pi}_k = \frac{1}{N}\sum_{i=1}^{N} \tilde{Z}_{i,k}. \tag{S2.5}$$

Each summand in the second term in (S2.4), $\int[\sum_{i=1}^{N} \tilde{Z}_{i,k} f(\boldsymbol{\beta}; \boldsymbol{m}_{i,k}, c_{i,k})] \log f(\boldsymbol{\beta}; \boldsymbol{\mu}_k, \Sigma_k) d\boldsymbol{\beta}$, is maximized independently with respect to $\boldsymbol{\mu}_k$ and Σ_k. We use the probability density function for a multivariate Gaussian to rewrite this term as

$$\int \left[\sum_{i=1}^{N} \tilde{Z}_{i,k} f(\boldsymbol{\beta}; \boldsymbol{m}_{i,k}, c_{i,k})\right] \left[-\frac{(\boldsymbol{\beta}-\boldsymbol{\mu}_k)^\top \Sigma_k^{-1}(\boldsymbol{\beta}-\boldsymbol{\mu}_k) - (T^{\text{stim}} + T^{\text{self}} + 1)\log(2\pi) + \log(|\Sigma_k^{-1}|)}{2}\right] d\boldsymbol{\beta}. \tag{S2.6}$$

Differentiating (S2.6) with respect to $\boldsymbol{\mu}_k$ reveals that

$$\begin{aligned} 0 &= \int \left[\sum_{i=1}^{N} \tilde{Z}_{i,k} f(\boldsymbol{\beta}; \boldsymbol{m}_{i,k}, c_{i,k})\right] \Sigma_k^{-1}(\boldsymbol{\beta}-\boldsymbol{\mu}_k) d\boldsymbol{\beta} \\ &= \sum_{i=1}^{N} \tilde{Z}_{i,k} \Sigma_k^{-1} \left[\int f(\boldsymbol{\beta}; \boldsymbol{m}_{i,k}, c_{i,k}) \boldsymbol{\beta} d\boldsymbol{\beta} - \int f(\boldsymbol{\beta}; \boldsymbol{m}_{i,k}, c_{i,k}) \boldsymbol{\mu}_k d\boldsymbol{\beta}\right] \\ &= \sum_{i=1}^{N} \tilde{Z}_{i,k} \Sigma_k^{-1} \left[\boldsymbol{m}_{i,k} - \boldsymbol{\mu}_k\right]. \end{aligned}$$

This gives us

$$\hat{\boldsymbol{\mu}}_k = \frac{\sum_{i=1}^{N} \tilde{Z}_{i,k} \boldsymbol{m}_{i,k}}{\sum_{i=1}^{N} \tilde{Z}_{i,k}}. \tag{S2.7}$$

To maximize (S2.6) with respect to Σ_k, we note that it is concave in Σ_k^{-1}. We can then solve for Σ_k by setting the derivative with respect to Σ_k^{-1} equal to zero:

$$0 = \int \sum_{i=1}^{N} \tilde{Z}_{i,k} f(\boldsymbol{\beta}; \boldsymbol{m}_{i,k}, c_{i,k}) \left[-\frac{1}{2}(\boldsymbol{\beta} - \boldsymbol{\mu}_k)(\boldsymbol{\beta} - \boldsymbol{\mu}_k)^\top + \frac{1}{2}\Sigma_k \right] d\boldsymbol{\beta},$$

making use of the fact that $\frac{d}{dX}\log(|X|) = (X^{-1})^\top$, and that $\Sigma_k = \Sigma_k^\top$. That is,

$$\Sigma_k = \frac{\sum_{i=1}^{N} \tilde{Z}_{i,k} \mathbb{E}\left[(\boldsymbol{\beta} - \boldsymbol{\mu}_k)(\boldsymbol{\beta} - \boldsymbol{\mu}_k)^\top\right]}{\sum_{i=1}^{N} \tilde{Z}_{i,k}},$$

where the expectation is with respect to $\boldsymbol{\beta} \sim \mathcal{N}(\boldsymbol{m}_{i,k}, c_{i,k})$.

We can decompose this expectation as

$$\begin{aligned}
&\mathbb{E}_{\boldsymbol{\beta}\sim\mathcal{N}(\boldsymbol{m}_{i,k},c_{i,k})}\left[(\boldsymbol{\beta} - \boldsymbol{\mu}_k)(\boldsymbol{\beta} - \boldsymbol{\mu}_k)^\top\right] \\
= \;& \mathbb{E}_{\boldsymbol{\beta}\sim\mathcal{N}(\boldsymbol{m}_{i,k},c_{i,k})}\left[(\boldsymbol{\beta} - \boldsymbol{m}_{i,k} + \boldsymbol{m}_{i,k} - \boldsymbol{\mu}_k)(\boldsymbol{\beta} - \boldsymbol{m}_{i,k} + \boldsymbol{m}_{i,k} - \boldsymbol{\mu}_k)^\top\right] \\
= \;& \mathbb{E}_{\boldsymbol{\beta}\sim\mathcal{N}(\boldsymbol{m}_{i,k},c_{i,k})}\big[(\boldsymbol{\beta} - \boldsymbol{m}_{i,k})(\boldsymbol{\beta} - \boldsymbol{m}_{i,k})^\top + (\boldsymbol{m}_{i,k} - \boldsymbol{\mu}_k)(\boldsymbol{m}_{i,k} - \boldsymbol{\mu}_k)^\top \\
& + (\boldsymbol{\beta} - \boldsymbol{m}_{i,k})(\boldsymbol{m}_{i,k} - \boldsymbol{\mu}_k)^\top + (\boldsymbol{m}_{i,k} - \boldsymbol{\mu}_k)(\boldsymbol{\beta} - \boldsymbol{m}_{i,k})^\top\big] \\
= \;& c_{i,k} + (\boldsymbol{m}_{i,k} - \boldsymbol{\mu}_k)(\boldsymbol{m}_{i,k} - \boldsymbol{\mu}_k)^\top.
\end{aligned}$$

Plugging this expression back in for the expectation, we have

$$\begin{aligned}
\Sigma_k &= \frac{\sum_{i=1}^{N} \tilde{Z}_{i,k}(c_{i,k} + \boldsymbol{m}_{i,k}\boldsymbol{m}_{i,k}^\top + \boldsymbol{\mu}_k\boldsymbol{\mu}_k^\top - \boldsymbol{m}_{i,k}\boldsymbol{\mu}_k^\top - \boldsymbol{\mu}_k\boldsymbol{m}_{i,k}^\top)}{\sum_{i=1}^{N} \tilde{Z}_{i,k}} \\
&= \frac{\sum_{i=1}^{N} \tilde{Z}_{i,k}(c_{i,k} + \boldsymbol{m}_{i,k}\boldsymbol{m}_{i,k}^\top - \boldsymbol{\mu}_k\boldsymbol{\mu}_k^\top)}{\sum_{i=1}^{N} \tilde{Z}_{i,k}},
\end{aligned}$$

where in the last step we make use of (S2.7) to simplify the last two terms in the numerator. Finally, subjecting this expression to the constraint that Σ_k must be diagonal, we have:

$$\hat{\Sigma}_k = \frac{\sum_{i=1}^{N} \tilde{Z}_{i,k}\mathrm{diag}(c_{i,k} + \boldsymbol{m}_{i,k}\boldsymbol{m}_{i,k}^\top - \hat{\boldsymbol{\mu}}_k\hat{\boldsymbol{\mu}}_k^\top)}{\sum_{i=1}^{N} \tilde{Z}_{i,k}}. \tag{S2.8}$$

In the main manuscript, the solutions (S2.7) and (S2.8) are applied only to $\boldsymbol{\mu}^{\mathrm{self}}, \Sigma^{\mathrm{self}}$, as the other components are held fixed during EM.

S2.1.3 Computational complexity, runtime information, and job load

Each iteration of the E step must perform $N \times K$ convex gradient descent sub-optimizations, for every neuron and every cluster, through a space of $T^{\mathrm{stim}} + T^{\mathrm{self}} + 1$ parameters on T data points. This operation is the most costly, as the M step is a simple calculation.

In practice, using a warm start for these convex gradient descent sub-optimizations, where the results of last iteration's sub-optimizations are used to initialize those of this iteration, dramatically reduces the runtime of subsequent iterations. Concretely, for Case B applied to $T = 20,000$ time bins of Allen Cell Types Database data, each sub-optimization took about 40ms after the warm start had kicked in.

The number of iterations necessary for the EM algorithm to converge is difficult to predict theoretically. In practice, the relative change in loss ($\frac{\text{Loss(iter}-1)-\text{Loss(iter)}}{\text{Loss(iter)}}$, where $\text{Loss} \equiv \sum_{i=1}^{N} \text{LL}_i$, with LL_i defined in Section 2.2.4) reached 10^{-7} in at most 55 iterations for $K = 20$ applied to all $N = 634$ Allen Cell Types Database neurons. For smaller values of K, fewer iterations are necessary.

The total runtime for a single initialization of the EM algorithm on Allen Cell Types Database data with $K = 20$ was around 5 hours or less for Case A (for a single choice of λ^{stim}) and 21 hours or less for Case B. All optimizations were performed on individual compute nodes with at least 32 cores and 256G of memory. We performed separate optimizations in parallel on separate nodes to produce different hyperparameter combinations and random initializations. For Case A with the Allen Cell Types Database data, we varied $K \in \{1, ..., 20\}$, $\lambda^{\text{stim}} \in \{10^{-9}, 10^{-8\frac{2}{3}}, ..., 10^{-1}\}$, and ran the EM algorithm for 20 different random initializations, yielding a total of $20 \times 26 \times 20 = 14000$ separate runs.

S2.2 Alternative Model: all parameters depend on cell-type

Here, we consider an alternate formulation of both sequential and simultaneous methods, where instead of just $\boldsymbol{\beta}_i^{\text{self}}$ being related to cell-type, as considered in the main manuscript, we consider the case where all of $\boldsymbol{\beta}_i$ is related to cell-type. We call the former, in main, case A, and the latter, detailed here, case B.

We note here that providing analysis of the relative benefits of case A and case B is not a major objective of this work. Rather, we seek to demonstrate only that there are options for how one chooses to define a hierarchical model within our general framework, and each may have its own benefits and drawbacks, depending on the line of inquiry. As the rest of this section will elaborate, this option is equally present for both the simultaneous and sequential methods.

We focus on case A in the main primarily because it provides a simpler model than B: it selects fewer clusters, lower within-cluster variances (Σ_k), and by its definition is a lower dimensional model. All of these factors give case A an advantage over case B in terms of interpretability. Additionally, case A performs better than B in terms of model selection on simulated data (compare Figs 2.3, S2.8A, and S2.8B with Figs S2.2, S2.8C, and S2.8D). Finally, the shared variables in case A, the self-interaction filters, are more strongly associated with intrinsic neural dynamics, whereas, at least in the IVSCC dataset, the stimulus-filters may depend more on other factors, such as the impedance of the electrode-neuron interface used to administer the stimulus. This, along with the fact that case A is lower dimensional, may account for any relatively better performance by case A. However we also acknowledge that there are cases where case B may be preferable, such as when one specifically wishes to distinguish the stimulus processing properties of neurons.

S2.2.1 Alternative Methods

Below we detail how the sequential and simultaneous methods change for case B. For both, the main difference is that instead of fitting a cluster model with means $\boldsymbol{\mu}_k^{\text{self}}$ and covariance matrices Σ_k^{self} that describes the distribution of $\boldsymbol{\beta}_i^{\text{self}}$, they now fit one with means $\boldsymbol{\mu}_k$ and covariance matrices Σ_k that describes the distribution of $\boldsymbol{\beta}_i$. For convenience, we will refer to the appropriate components of $\boldsymbol{\mu}_k$ and Σ_k using superscripts, e.g. $\boldsymbol{\mu}_k^{\text{stim}}$, just as we do with $\boldsymbol{\beta}_i$.

Sequential Method

Because, in the sequential method, the fitting of single cell models does not depend on cell-types, that first stage is identical between cases A and B. For the second stage, clustering, the only difference is that GMM clustering is performed on all of $\hat{\boldsymbol{\beta}}_i$, instead of just $\hat{\boldsymbol{\beta}}_i^{\text{self}}$:

$$\hat{\Omega}_K \leftarrow \arg\max_{\Omega_K} \sum_{i=1}^{N} \log \sum_{k=1}^{K} \pi_k f(\hat{\boldsymbol{\beta}}_i; \boldsymbol{\mu}_k, \Sigma_k) \tag{S2.9}$$

$$\hat{k}_i \leftarrow \arg\max_{k} \hat{\pi}_k f(\hat{\boldsymbol{\beta}}_i; \hat{\boldsymbol{\mu}}_k, \hat{\Sigma}_k),\ i = 1, ..., N \tag{S2.10}$$

Algorithm 1 is thus unchanged except for changing the last two lines accordingly.

The computation of the BIC also changes accordingly:

$$BIC \equiv \sum_{i=1}^{N} \log \left[\sum_{k=1}^{K} \hat{\pi}_k f(\hat{\boldsymbol{\beta}}_i; \hat{\boldsymbol{\mu}}_k, \hat{\Sigma}_k) \right] - \frac{K - 1 + 2K\dim(\boldsymbol{\beta})}{2} \log(N) \tag{S2.11}$$

Simultaneous Method

For the simultaneous method, case B is much simpler, as all parameters of the single cell models use the learned cluster structure as priors, so there is no longer a need for ℓ_2 regularization.

The joint probability of the data and parameters simplifies instead to:

$$P_{\text{joint}}(k, \boldsymbol{\beta}, \mathbf{y}_i | \mathbf{x}_i; \Omega_K) \propto P_{SC}(\mathbf{y}_i | \mathbf{x}_i; \boldsymbol{\beta}) f(\boldsymbol{\beta}_i^{\text{stim}}; \boldsymbol{\mu}_k^{\text{stim}}, \Sigma_k^{\text{stim}}) f(\boldsymbol{\beta}_i^{\text{self}}; \boldsymbol{\mu}_k^{\text{self}}, \Sigma_k^{\text{self}}) f(\beta_i^0; \mu_k^0, \Sigma_k^0) \pi_k. \tag{S2.12}$$

With this adapted P_{joint} the results derived in the main manuscript, i.e. (2.12)-(2.21), all apply to case B, without making the exceptions for constrained cluster means and covariances, as was done for S2.7,S2.8. Algorithm 2 is thus unchanged except that all of $\boldsymbol{\mu}_k$ and Σ_k are updated in the M-step instead of just $\boldsymbol{\mu}_k^{\text{self}}$ and Σ_k^{self}. For (2.21), in case B we have $\text{dof}(\hat{\Omega}_K) = K * (2 * (T^{\text{self}} + T^{\text{stim}} + 1) + 1) - 1$, reflecting the change in number of free parameters.

S2.2.2 Results on Simulated Data

Here we show the equivalent results to those presented in Figs 2.2 and 2.3, but for the alternative model where all parameters depend on cell-type. For these results, the simulated dataset was sampled from GLMs whose true $\boldsymbol{\mu}^{\text{stim}}$ and μ^0 were clustered by cell-type, in addition to $\boldsymbol{\mu}^{\text{self}}$ (Fig S2.1A and S2.1B).

Fig S2.1C-S2.1F show very similar results for parameter recovery to those in Fig 2.2, the only salient difference being that the recovery of $\boldsymbol{\beta}_i^{\text{stim}}$ is also much better in the simultaneous method, and for lower σ. This is to be expected now that the true values of these parameters

depend on sigma, and their estimates in the simultaneous method make use of cell-type specific priors.

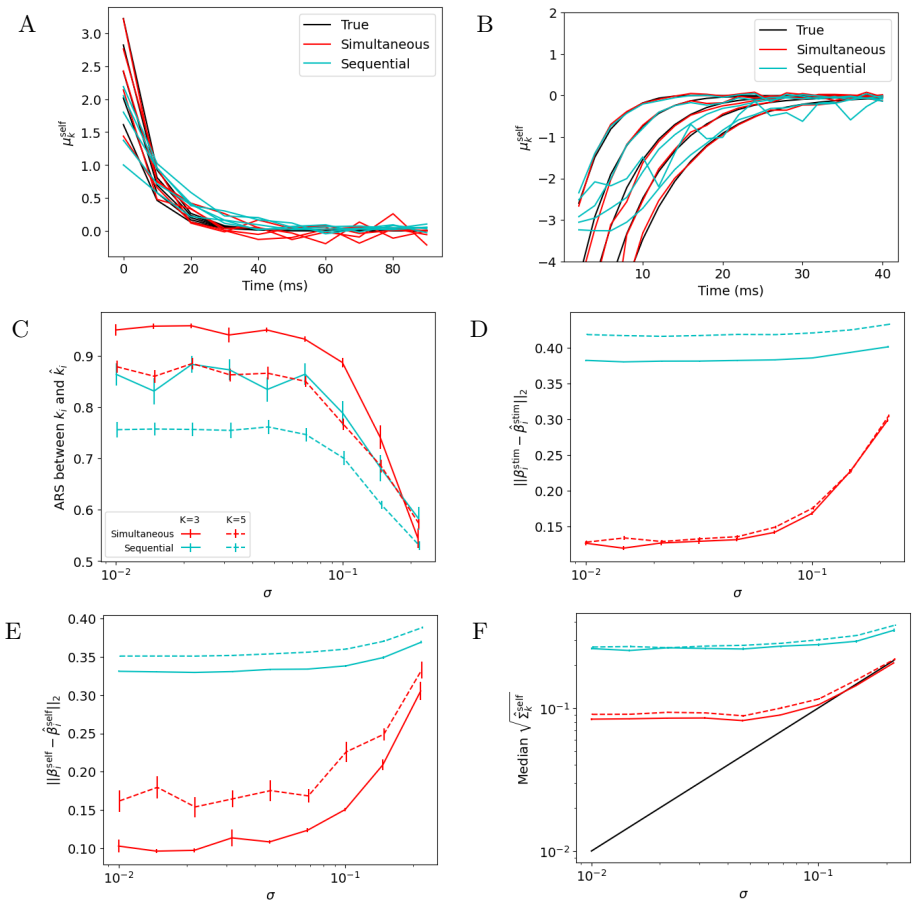


Figure S2.1: **Performance of simultaneous and sequential methods with all parameters shared on data simulated from a fixed cluster structure.**

A, B: The true cluster means $\boldsymbol{\mu}_k^{\text{stim}}, \boldsymbol{\mu}_k^{\text{self}}$ used to generate simulated datasets and those estimated by the sequential and simultaneous methods, fit with the correct $K = 5$.

C-F: same plots as Fig 2.2, but with the alternative model. In all metrics and conditions except ARS with $K = 3$ and the two highest values of σ, the simultaneous method is significantly better.

Model selection, however, has markedly worse performance with the all-parameters-shared model, and we can no longer say that the simultaneous method is better (Fig S2.2). Since

the $\boldsymbol{\mu}^{\text{self}}$ are the same in all cases we consider, we can interpret this result by saying that for our choice of Ω_K, the additional differences between clusters in $\boldsymbol{\mu}^{\text{stim}}$ and μ^0 do not make it easier enough to separate the clusters to overcome the increased complexity penalty in BIC with increased dof$(\hat{\Omega}_K)$.

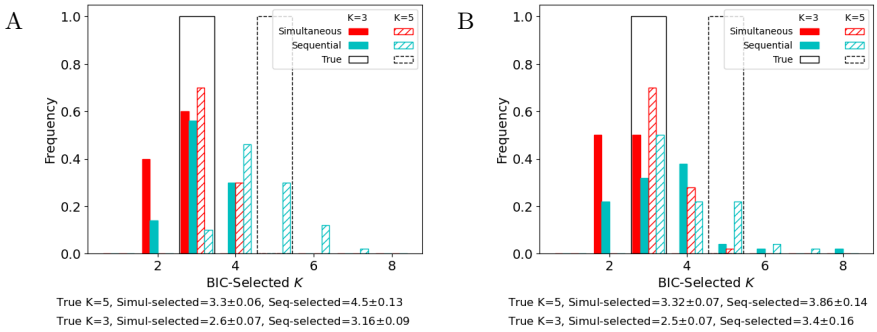


Figure S2.2: **Model selection of $\hat{K}$ using BIC with the alternative model.** Frequencies of $\hat{K}$ estimated via BIC over 50 simulated datasets with the same μ_k as in Fig S2.1, and $\sigma = 10^{-2}$ (A) or $10^{-5/6}$ (B), the maximum value that does not result in degenerate simulations. Black lines indicate true K. Summary below plots gives the mean ± SEM of estimated $\hat{K}$ across the 50 datasets for each case and each method.

S2.2.3 Results on Allen Cell Types Database Data

Here we show results like those shown in Figs 2.4,2.5, and 2.6, but using the alternate model where all parameters are cell-type-dependent (case B).

In Figs S2.3, S2.4, and S2.5, we compare between simultaneous and sequential methods the discovered cluster structure, the generalization performance of fitted models, and the link between cluster labels and metadata, respectively, resulting from fitting to the Allen Cell Types Database data.

To assess how similar the cluster assignments discovered by this alternate method are to those presented in main, we show the confusion matrix in Fig S2.6A for the simultaneous method, and in Fig S2.6B for the sequential method. For completeness, we also show the other 4 pairs of comparisons between different clusterings. In each panel, the rows and columns

are ordered independently to maximize the sum along the main diagonal. This is achieved using the Hungarian algorithm, implemented in `scipy.optimize.linear_sum_assignment`. Afterwards the values are normalized by the geometric mean of the row- and column-sums, so that the plot is not dominated by the larger clusters.

All four combinations of method and case produce very different clusterings, but the two results of the sequential method produce the most similar results. This might be expected from the fact that these use identical estimates of self-interaction filters $\hat{\boldsymbol{\beta}}_i^{\text{self}}$, which are used in both clusterings. The fact that there is such good agreement may be taken to mean that the other GLM parameters do not convey much useful information about cell type, as the heavy overlap between clusters in the stimulus filter dimensions would suggest (see Fig S2.3F).

The greater change is seen for the simultaneous method, both in the clustering assignments (see Fig S2.6A), and in the increased spread of clusters along the self-interaction filter dimension ($\sqrt{\Sigma_k^{\text{self}}}$, shaded bars in Fig S2.3B). The latter may be explained by the fact that here we are clustering in a higher-dimensional space, and therefore clusters will generally spread wider along an existing dimension to account for the extra variability added in the new dimensions. This explanation presumes that the new dimensions provide new variability that is different enough from that in the existing dimensions to meaningfully change cluster assignments, as is indeed the case (see Fig S2.6A). Why the variability in stimulus filter dimensions is different from in self-interaction filter dimensions is a difficult question to answer rigorously, but we propose that the self-interaction filters may truly be more purely related to intrinsic dynamics, whereas the stimulus filter may also be accounting for elements of the electrode-neuron interface. These elements may include electrical properties of the interface and the electrode placement in relation to the morphology of the neuron.

The fact that a similar pattern does not emerge for the sequential method may be a result of the fact that the discovered clusters overlap heavily in the stimulus filter dimensions, as shown in Fig S2.3F, whereas there are distinct differences between clusters for the simultaneous method (Fig S2.3E). This pattern is expected from our general observation that the simultaneous method can use borrowed strength between neurons to collapse clusters and increase their separation. As such, the addition of stimulus filters to the clustering does

not change the cluster assignments very much (see Fig S2.6B), and thus the positions of the clusters in the self-interaction filter dimensions do not change substantially.

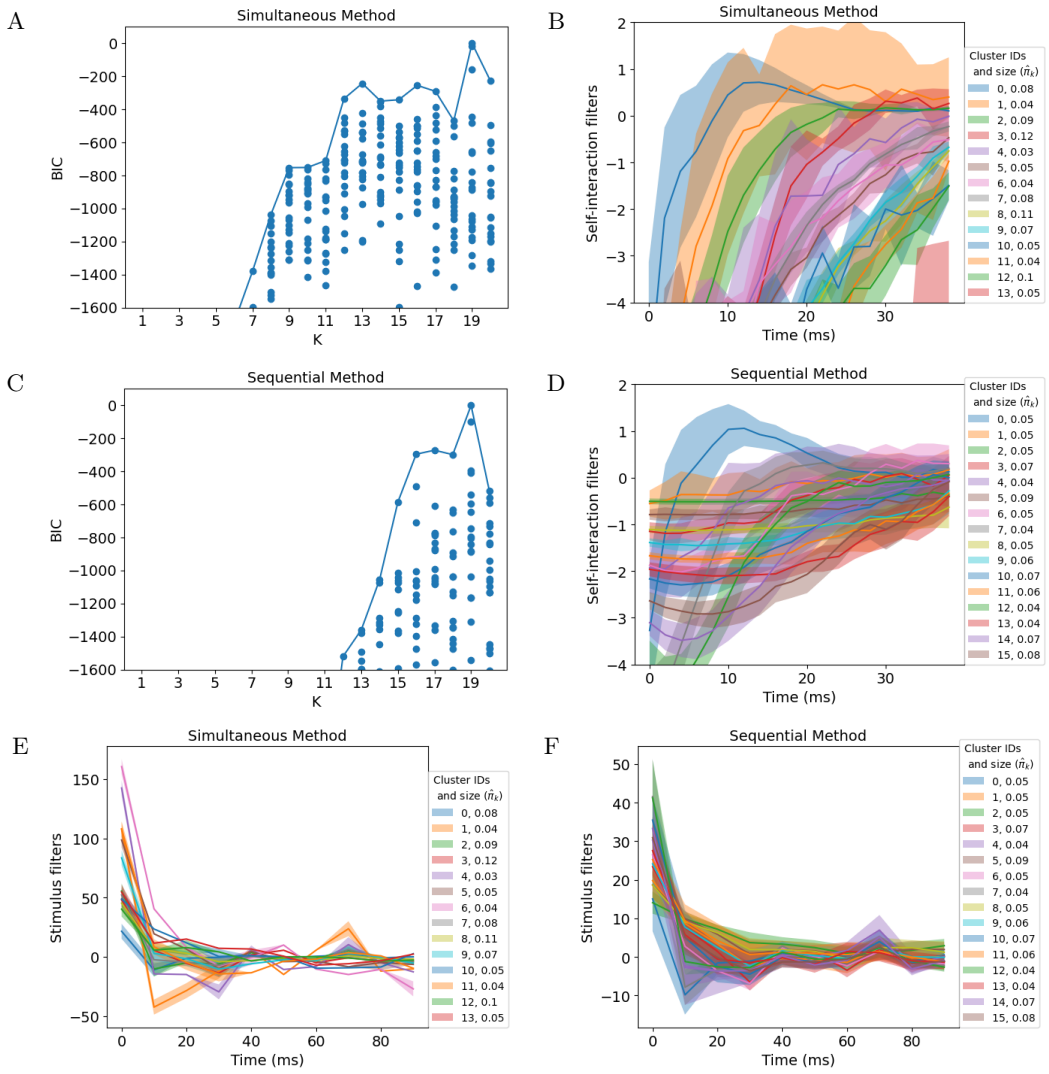


Figure S2.3: **Allen Cell Types Database Dataset, Alternative Model.** A-D: Same analysis as in Fig 2.4, but with the alternative model (case B) where all parameters are cell-type dependent. Here, $\hat{K} = 19$ is selected by BIC for both methods. In addition, panels E and F show comparable information to B and D, but for the stimulus filters.

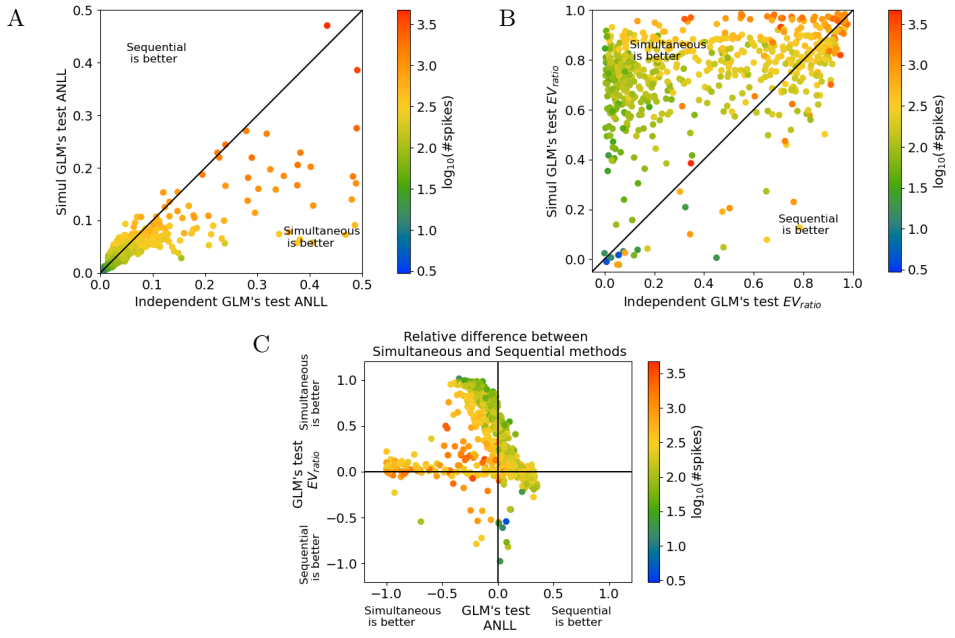


Figure S2.4: **Allen Cell Types Database Generalization Performance, Alternative Model.** Same analysis as Fig 2.5A, 2.5C, and 2.5E, but with the alternative model (case B) where all parameters are cell-type dependent. Results here are comparable to those in the main manuscript.

S2.3 Additional Details for Figs 2.5 and S2.4

The following descriptions apply to both Figs 2.5 and S2.4, except for panels B, D, and F, which do not exist in Fig S2.4, as these analyses have not been performed. The results in Fig 2.5 are from models where only the self-interaction filters are shared ("Case A" - see Sections 2.2.2 and 2.2.3), whereas those in Fig S2.4 are from models where all parameters are shared ("Case B" - see Sections S2.2.1 and S2.2.1).

To generate panels A, C, and E, we split the neurons into four folds, and applied both simultaneous and sequential approaches and performed model selection on three of the four folds using the Noise 1 stimulus. We then fit GLM models for each neuron in the test fold using the Noise 1 stimulus with the results of training (selected hyperparameters, as well as the fitted hierarchical model for the simultaneous method). Then, we use the Noise 2

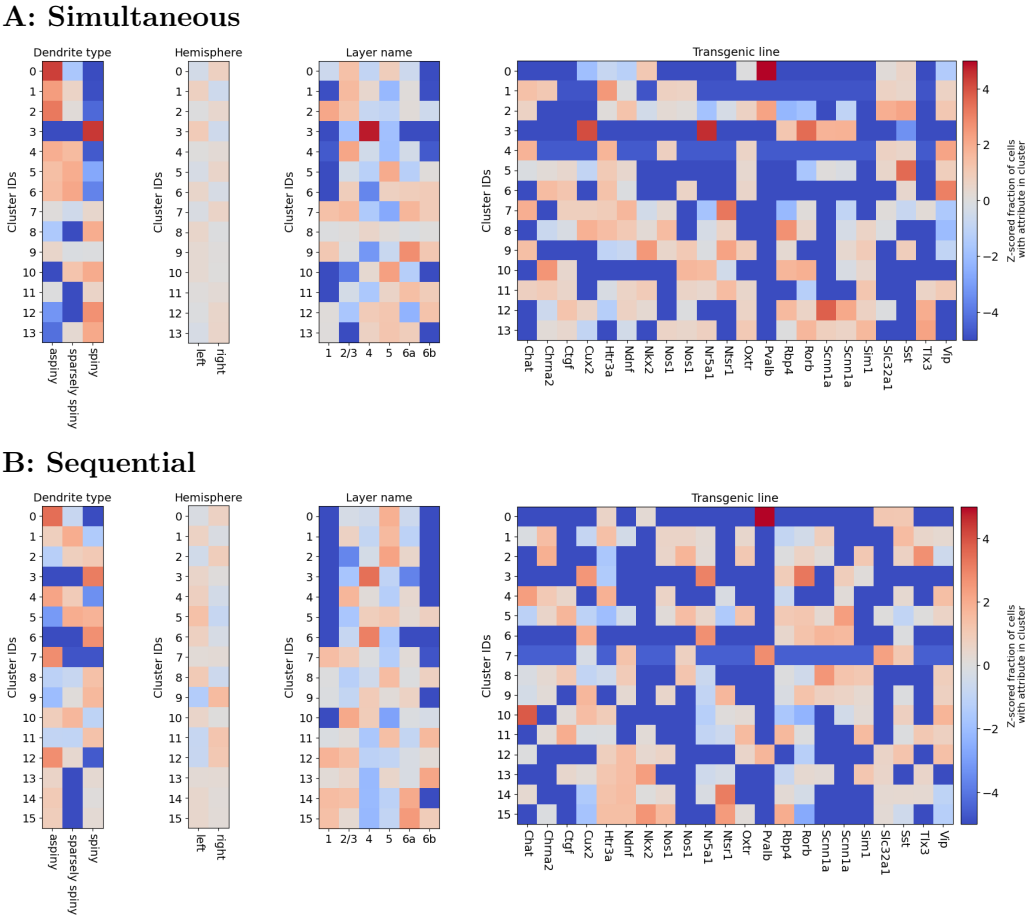


Figure S2.5: **Allen Cell Types Database Metadata, alternate model.** Same analysis as Fig 2.6, but with the alternative model (case B) where all parameters are cell-type dependent. Results here are comparable to those in the main manuscript.

stimulus to evaluate those test neurons. We repeat this process for each choice of test fold to evaluate all neurons.

To generate panels B, D, and F:

- We fix all hyperparameters to their values selected when using all presentations of Noise 1 to all neurons.

- We split the neurons into four folds, and applied both simultaneous and sequential

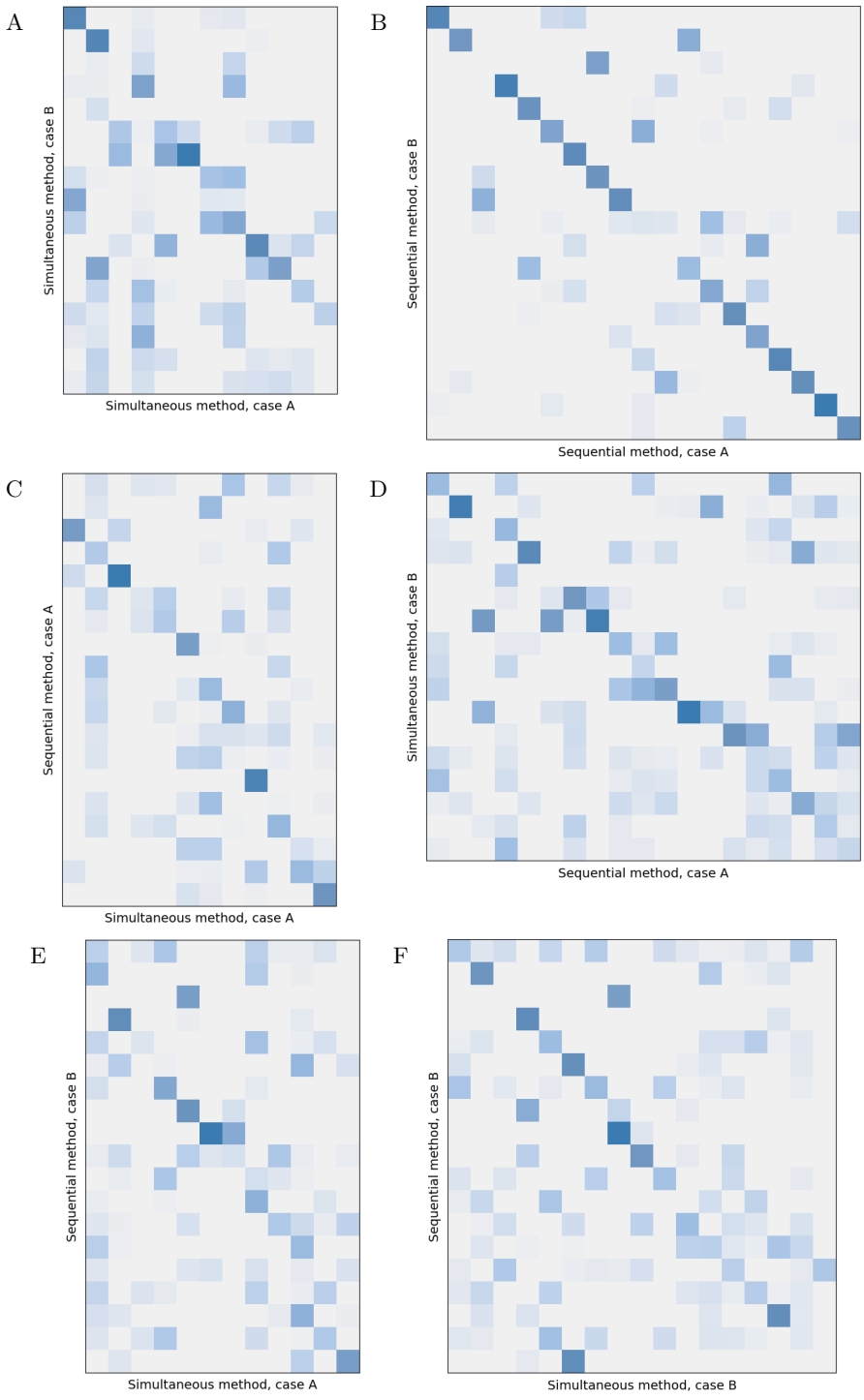


Figure S2.6: **Similarity of clusterings from the four method-case combinations.** Each panel shows the confusion matrix of the clusterings from two of the four method-case combinations.

approaches to subsets of the folds: either one fold, or two folds, or three folds. These are the columns of panels B, D, and F. We also used only one, two, or three presentations of the Noise 1 stimulus: these are the rows of panels B, D, and F. We then estimate the GLM parameters and cluster assignments for all neurons in the test fold(s), using all presentations of the Noise 1 stimulus with the fixed hyperparameters and the results of training (the fitted hierarchical model for the simultaneous method or the fitted GMM for the sequential method). We then evaluated ANLL and EV_{ratio} on all presentations of Noise 2 to these test neurons. This process is then repeated for each possible division of the folds into training and testing, and ANLL and EV_{ratio} for each neuron are each averaged over all divisions in which that neuron was in the test fold(s).

For instance, to obtain the top left square of panel B, we repeatedly applied both simultaneous and sequential approaches to 1/4 of the neurons (one fold) using only one presentation of the Noise 1 stimulus. Then, for the remaining 3/4 of the neurons (three folds), we estimated their GLM parameters using all presentations of Noise 1 (along with the fitted hierarchical model for the simultaneous method), and finally evaluated them using all presentations of the Noise 2 stimulus.

- In each cell of panels B and D, we define a *sample* as the relative difference (simultaneous minus sequential, divided by their sum) in ANLL and EV_{ratio}, respectively, of a single neuron.

- In each cell of panel F, we define a *sample* as the relative difference (simultaneous minus sequential, divided by their sum) in ARS of cluster assignments between a pair of different divisions of the neurons into training and testing.

- In each cell, the color is the median across samples, and we place an asterisk if the simultaneous method is significantly better (negative relative difference for panel B, positive for D and F), i.e. if a one-sided Wilcoxon signed-rank test of the samples produces an uncorrected p-value less than 0.001.

- Between each pair of adjacent cells, except horizontal pairs in panel F, we computed a

two-sided Wilcoxon signed-rank test on the paired differences between their samples. We placed a dash if the p-value is less than 0.001.

- The p-values reported in the caption for trends in horizontal and vertical differences, except for the horizontal trend in panel F, are calculated by a one-sided Wilcoxon signed-rank test on the paired differences between samples, aggregated across all six pairs of either horizontally or vertically adjacent cells. P-values greater than 0.1 are reported as "not significant."

- Between each pair of horizontally adjacent cells in panel F, samples cannot be paired, so we used a two-sided, two-sample t-test between the samples from the left and right cells to measure significance, and placed a dash if the p-value is less than 0.001.

- Likewise, the p-value for the horizontal trend in panel F is computed by a one-sided two-sample t-test between the aggregated samples from the six leftmost cells and the six rightmost cells (it was greater than 0.1 and thus labeled "not significant").

S2.4 Further Details for Simulated Data

This section details our process for simulating datasets from hierarchical models in both cases A and B. These datasets were used to generate Figs 2.2 and 2.3 and Figs S2.1 and S2.2 respectively.

Instead of truly sampling the cell-types, k_i, from $\{\pi_k\}$ we simply assign an equal number (40) of neurons to each class. To set the filters, we use functions that parameterize the stimulus filters as a decaying exponential,

$$g(t; a_{stim}, \tau_{stim}) \equiv a_{stim} e^{-\frac{t}{\tau_{stim}}}, \tag{S2.13}$$

and the self-interaction filters as the sum of a negative decaying exponential and a Gaussian bump,

$$h(t; t_{ref}, \tau_{ref}, a_{ISI}, \mu_{ISI}, \sigma_{ISI}) \equiv -e^{-\frac{t-t_{ref}}{\tau_{ref}}} + a_{ISI} e^{-\frac{(t-\mu_{ISI})^2}{2*\sigma_{ISI}^2}}. \tag{S2.14}$$

In both cases A and B, $\boldsymbol{\mu}_k^{\text{self}}$ are determined with $t_{ref}, \tau_{ref}, a_{ISI}, \mu_{ISI}, \sigma_{ISI}$ that are set to uniformly spaced values for each k. In case A, a_{stim}, τ_{stim}, and β_k^0 are each fixed to a single value for all neurons, whereas in case B they are also set to uniformly spaced values for each $\boldsymbol{\mu}_k$. All relevant constants that parameterize this spacing are enumerated in Table S2.1.

Evenly spaced params for $\boldsymbol{\mu}_k^{\text{self}}$ in both cases A and B

Parameter	Value for $\boldsymbol{\mu}_1$	Increment	
t_{ref}	2	1.75	
τ_{ref}	2	0.5	
a_{ISI}	0.2	-0.05	
μ_{ISI}	3	2	
σ_{ISI}	3	0.25	

Evenly spaced for $\boldsymbol{\mu}_k$ in case B, fixed to one value for all $\boldsymbol{\beta}_i$ in case A

Parameter	Value for $\boldsymbol{\mu}_1$ (case B)	Increment (case B)	Value for all $\boldsymbol{\beta}_i$ (case A)
a_{stim}	0.5	0.125	0.9
τ_{stim}	4	0	4
μ_k^0	-4.5	0.25	-5

Table S2.1: **Spacing of the parameters used to simulate datasets for cases A and B.**

We also used the same stimulus downsampling factor, $d^{\text{stim}} = 5$, in our simulations. In order that its value does not affect the simulation much, we compute $g(t), t \in \{1, .., T^{\text{stim}} * d^{\text{stim}}\}$, and then sum the τth block of d^{stim} values, $\sum_{t=(\tau-1)*d^{\text{stim}}}^{\tau*d^{\text{stim}}} g(t)$ to assign to $\beta_i^{\text{stim}}(\tau)$ (case A) or $\mu_i^{\text{stim}}(\tau)$ (case B).

Once all $\boldsymbol{\mu}_k$ have been set, $\boldsymbol{\beta}_i$ are sampled from $f(\boldsymbol{\beta}_i; \boldsymbol{\mu}_k, \sigma^2 * I)$, where they have not already been determined (case A). Once all of $\boldsymbol{\beta}_i$ has been determined, the spike train for the ith neuron is then simulated by sampling from the GLM's distribution for $y_i(1)$, then $y_i(2)$, all the way up to $y_i(T_i)$ (using eq 2.2).

When $\sigma > 10^{-5/6}$ is used, the simulation becomes unstable, because some $\boldsymbol{\beta}_i^{\text{self}}(\tau)$ values become high enough to cause runaway feedback, whereby the number of spikes in bins spaced

τ apart diverge to infinity with increasing time. This critical value of $\boldsymbol{\beta}_i^{\text{self}}(\tau)$ is around $-[\beta_i^0 + \max_t x_i(t) \sum_{t=1}^{T^{\text{stim}}} \beta_i^{\text{stim}}(t)]$, such that a single spike τ time bins ago can raise the spike rate above 1 if it occurs during an extended period of high stimulus values. To prevent this scenario, we impose $y(t) \in \{0, 1\}$ while simulating (as in [87]), and do not consider such high values of σ that would lead to such severe model misspecification that the true model parameters cannot be recovered. It is worth noting that this problem can arise even for filters fitted to real neural data because of model misspecification - specifically, there is a maximum number of times any neuron can spike in a given time bin because of its absolute refractory period, whereas the (Poisson) GLM makes no such assumption.

S2.4.1 Simulating spike trains from fitted models

Here, we consider the spike trains generated from GLM models whose parameters are taken from fits to the Allen Cell Types Database, and models fitted to those spike trains. This analysis provides an extension of the simulation studies in Sections 2.3.1 and S2.2.2, asking how accurate the simultaneous method is in the parameter regime appropriate for the Allen Cell Types Database.

As discussed in Section S2.4, certain GLM parameter regimes will lead to severe model misspecification because of the truncation of Poisson-generated spike counts to at most 1 spike per bin. Therefore we first select the five clusters from those fitted and displayed in Fig S2.3B and S2.3F that we expect to produce a sufficient number of spikes, while requiring the fewest instances where the truncation of the Poisson distribution affects the simulated data. To do this, we compute the maximum spiking rate that could arise from simulating a GLM with the kth cluster mean as parameters, following an isolated spike:

$$\log(\text{rate}_{\max}) = \max_{\tau} \hat{\mu}_k^{\text{self}}(\tau) + \hat{\mu}_k^0 + \max_t x(t) \sum_{t=1}^{T^{\text{stim}}} \hat{\mu}_k^{\text{stim}}(t).$$

As discussed in Section 2.3.1, when this quantity is above 0 the spike rate τ time bins after a spike will be greater than 1, leading to model misspecification. We thus choose the five clusters with the smallest $\log(\text{rate}_{\max})$, provided it is two standard deviations (determined using the fitted $\hat{\Sigma}_k$) above -2. The results of fitting the sequential and simultaneous methods

to the data simulated from these clusters is shown in Fig S2.7, and generally shows that, while imperfect, the simultaneous method outperforms the sequential method. It also appears that the major mistake the simultaneous method makes in this instance, merging two clusters with very similar self-interaction filters, is a reasonable one.

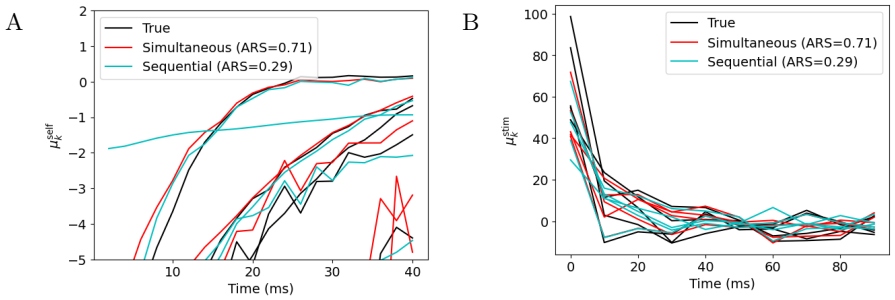


Figure S2.7: **Simulated data from cluster models with fitted parameters.** The simultaneous method adequately recovers the true cluster structure of simulated data. Here, the true cluster structure is that fitted using the simultaneous method.

S2.5 Model Selection

S2.5.1 Alternative Model Selection: Validation Log-Likelihood

In addition to using BIC to perform model selection, we also investigate using the validation log-likelihood (VLL) on held out neurons (see section 2.2.4 detailed method).

For our simulated data, we use the exact same models that the BIC analysis was applied to (Figs 2.3 and S2.2), but evaluate the model log-likelihood, averaged over a new validation set of 10 neurons per true cluster. Unlike with BIC, this VLL increases monotonically with K towards an asymptote for both the simultaneous and sequential methods.

For each of the 50 simulated datasets, we selected the lowest K whose VLL was within one standard error of the maximum, where this standard error was measured over the 50 datasets, after subtracting off each dataset's VLL for $K = 1$. This last step improves performance because each dataset has an offset in VLL that results from the spike density in the validation neurons (compare Figs S2.9 and S2.10 for an illustration of this effect in

the Allen Cell Types Database data). Overall, this approach can be thought of as a variant of the one-standard-error (1SE) rule [41]. These results are shown in Fig S2.8, and can be compared to those obtained via BIC in Figs 2.3 and S2.2. For case A, BIC is better, but for case B, the results are more mixed. Across all cases, BIC tends to select a lower $\hat{K}$ than VLL.

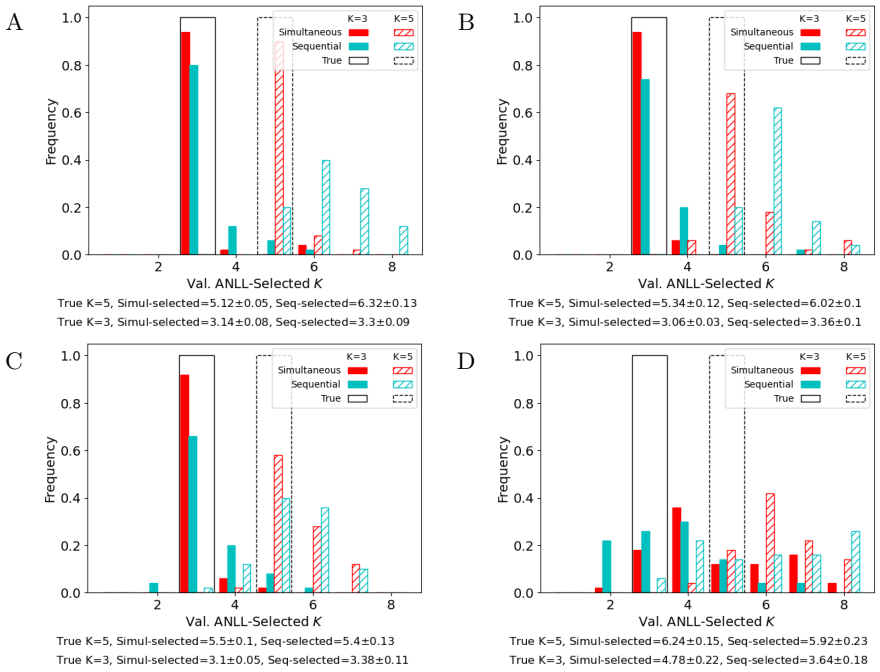


Figure S2.8: **Model selection of $\hat{K}$ using validation loss.** Frequencies of $\hat{K}$ estimated via the loss on held out neurons over 50 simulated datasets with the same μ_k as in Fig 2.2 (case A, panels A and B) or Fig S2.1 (case B, panels C and D), and $\sigma = 10^{-2}$ (A and C) or $10^{-5/6}$ (B and D), the maximum value that does not result in degenerate simulations. Black lines indicate true K. Summary below plots gives the mean ± SEM of estimated $\hat{K}$ across the 50 datasets for each case and each method.

For real data, where the true distribution of neurons is unknown, we must use cross-validated log-likelihood (CVLL) to get a metric that is not biased by our choice of test neurons. To compute CVLL, we randomly partition the neurons into L equally-sized sets $A_1, ..., A_L$. Each set A_l in turn is held out while the parameters are fit using K clusters to

the remaining neurons, and then the validation loss is simply $\frac{1}{|A_l|}\sum_{i\in A_l}\mathrm{LL}_i$ (see Section 2.2.4 for our definition of LL_i). CVLL is then evaluated as the average validation loss over the L different sets.

When this approach is applied to the Allen Cell Types Database data, we observe the same issue with monotonic improvement in VLL with K (Fig S2.9). Additionally, the simultaneous method yields VLLs for each validation fold that are offset far away from one another (Fig S2.9A and S2.9B). These offsets stem from large differences between the spike trains of individual neurons: some (with many spikes) yield a much higher ANLL, while others (with few spikes) yield a low ANLL (see Figs 2.5 and S2.4 for an illustration of this phenomenon). Since the ANLL of all the validation cells factors into the validation loglikelihood (VLL) in the simultaneous method, a random partition of the neurons into different validation sets produces some sets with higher VLL and some with lower VLL.

To account for the offsets between these curves, we first shift them by subtracting off their values at $K = 1$ before using the standard 1SE rule to select $\hat{K}$ (Fig S2.10). That is, we pick the lowest K whose CVLL is within one standard error of the maximum, where the standard error is computed over the shifted VLLs of the different partitions of the neurons into training and validation.

Just as with the simulated datasets, CVLL selects a much higher $\hat{K}$ than BIC for each case (compare to Fig 2.4A and 2.4C and S2.3A and S2.3C). It is worth noting that as the VLL curves have not fully plateaued, repeating this analysis with a greater range of K would likely result in a higher selected $\hat{K}$.

S2.5.2 Generalization Performance of Model Selection Metrics

We have now shown two different approaches to model selection and how they differ, both in method and results on simulated and real data. For the simulated data, as we know the true K, we are able to assess how the accuracy of selected $\hat{K}$. For the Allen Cell Types Database data, we cannot do this directly as there is no ground truth for K, but we can attempt a similar approach to that used in Figs 2.5 and S2.4, using the same measures of performance on predicting responses to the test stimulus of neurons that were held out from fitting the

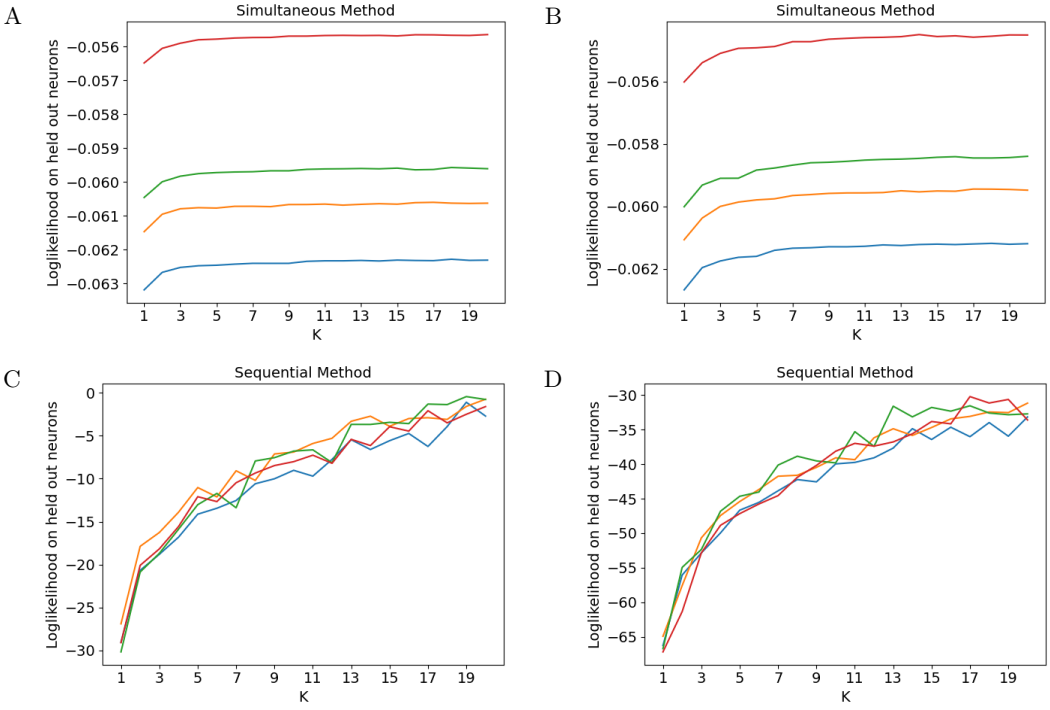


Figure S2.9: **Validation Log-Likelihood on Allen Cell Types Database Dataset.** Each color represents a specific split of the neurons into training and testing. All log-likelihoods increase monotonically with K. The different folds of the simultaneous method are heavily offset relative to one another as a result of the variety of spike counts in each fold's validation set (see Figs 2.5 and S2.4).

A: Case A, Simultaneous

B: Case B, Simultaneous

C: Case A, Sequential

D: Case B, Sequential

cluster models. Just as we argued that improvements in these metrics implied improved estimation of model parameters, $\boldsymbol{\beta}_i$, we can look for differences in these metrics, evaluated on test neurons, with respect to the K used to fit the simultaneous method, and compare them to model selection criteria computed on the train/validation neurons. The $\boldsymbol{\beta}_i$ estimated by

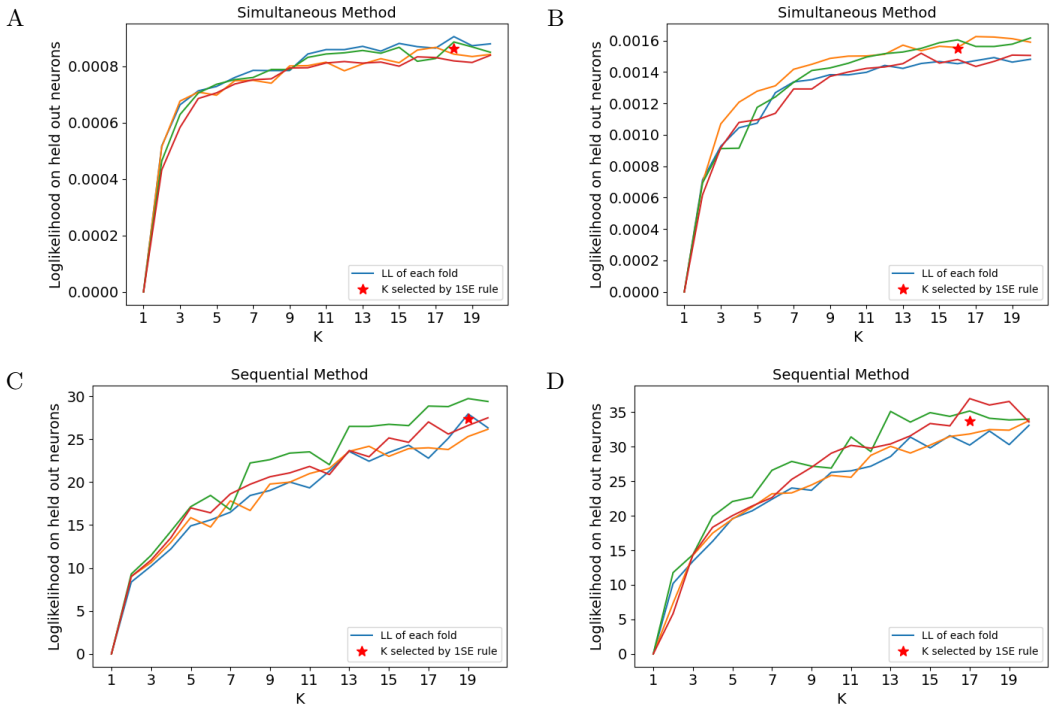


Figure S2.10: **Validation Log-Likelihood on Allen Cell Types Database Dataset.** Same colors as Fig S2.9. The One-Standard-Error rule (1SE) applied to the Cross-Validated Loss (CVL) selects very high $\hat{K}$ for all methods (red stars). To account for the large offsets shown in Fig S2.9, each VL curve is first shifted (as shown here) so that its value for $K = 1$ is 0 before 1SE is applied.

A: Case A, Simultaneous

B: Case B, Simultaneous

C: Case A, Sequential

D: Case B, Sequential

the sequential approach do not depend on K, so this analysis is limited to the sequential method.

However, these metrics do not show sufficient dependence on K to warrant such an analysis (Fig S2.12). With the possible exceptions of $K \in \{1, 2\}$, there is very little difference

between the two cases, A and B, or between fitted K, relative to the variation between neurons.

S2.6 Cluster Distribution for Sequential Method with BIC-selected K

In Fig 2.4, we showed the cluster distributions for $K = 12$ for both sequential and simultaneous methods to facilitate comparison between the two. However, $K = 19$ is the BIC-selected value for the sequential method, so, for completeness, we show those clusters in Fig S2.11. The results are much like those for $K = 12$, with highly overlapping clusters bunched near 0.

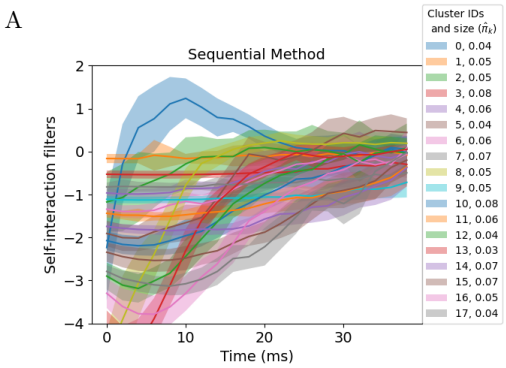


Figure S2.11: **Clusters discovered by the sequential method with $K = 19$ are highly overlapping and tightly bunched near 0.**

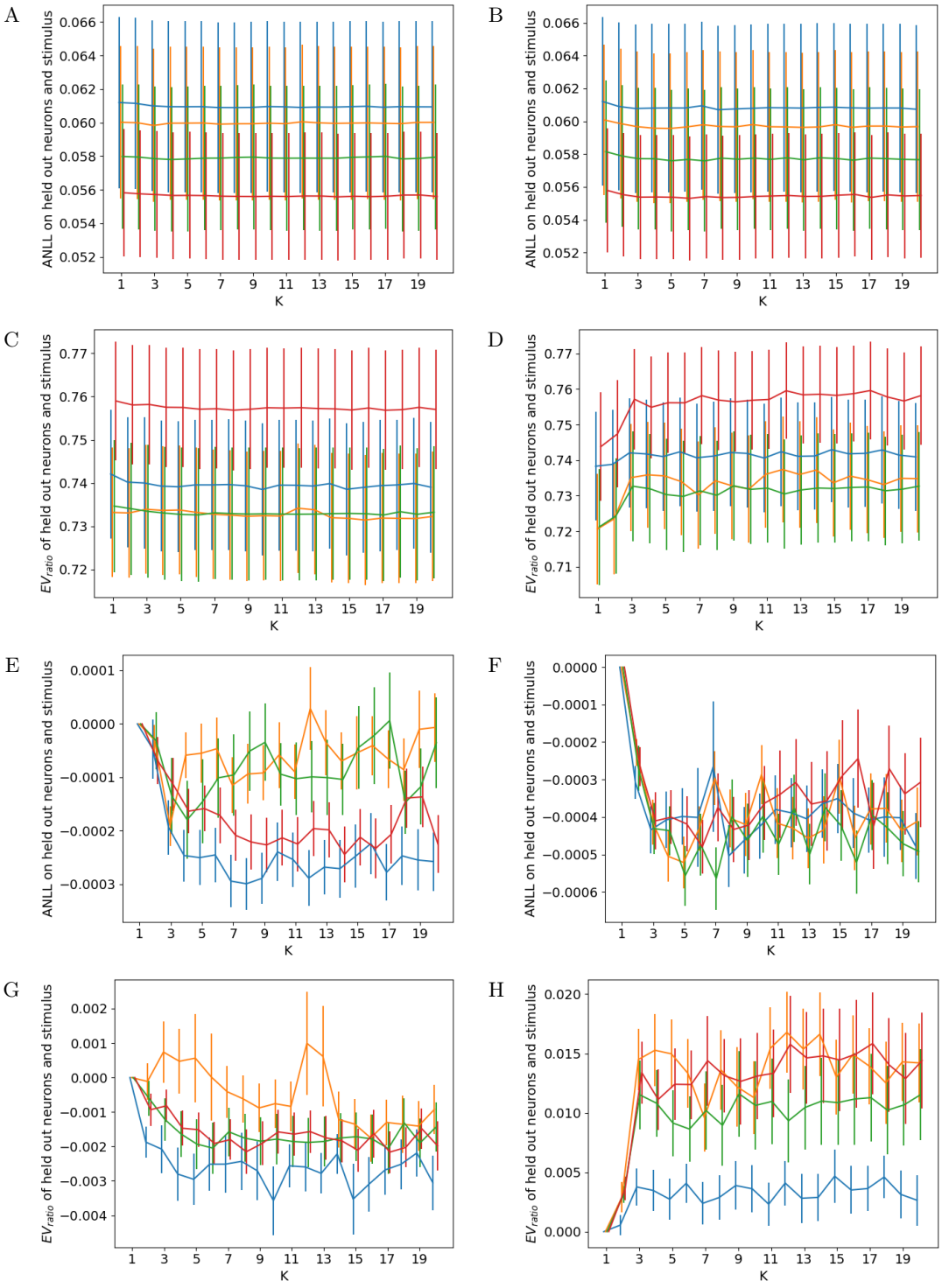
Figure S2.12: **Model Comparison using CVL on Allen Cell Types Database Dataset.** Validation metrics for each fold are shown, with the same colors as in Fig S2.9. A-D: Error bars are one SE across neurons in the validation fold. E-H: The curves are shifted so that their values at $K = 1$ are 0, to facilitate judging their (lack of) change w.r.t. K. Error bars are one SE across validation neurons of the shifted metrics.

A,E: Case A, ANLL

B,F: Case B, ANLL

C,G: Case A, EV_{ratio}

D,H: Case B, EV_{ratio}


A
B
C
D
E
F
G
H
ANLL on held out neurons and stimulus
EV_{ratio} of held out neurons and stimulus
K

Chapter 3

USING JOINT-TRAINING ON NEURAL DATA AND AUXILIARY TASKS TO IMPROVE MODELS OF NEURAL CIRCUITS

Abstract

Recent years have seen many successful attempts to capture properties of specific brain circuits with artificial neural networks (ANNs) that are trained to perform a task that is hypothesized to be relevant to the circuit at hand. Here, we analyze the "joint-training" approach, where an ANN is trained to learn a computation and model data from a neural circuit simultaneously, treating the computation as an auxiliary task. Conceptually, the aim of the approach is to impose task demands that will encourage the development of features that better predict neural responses. We first conduct a theoretical study of joint-training, in the context of linear networks. This yields analytic conditions under which the auxiliary task will improve performance in predicting neural responses, most notably that the model must possess a dimensionality bottleneck, and allows us to describe auxiliary tasks that will yield greatest possible improvement. We then conduct numerical studies of simple nonlinear networks to test the applicability of our analytical results. Specifically, we numerically fit ANNs to simulated deep, nonlinear neural circuits, demonstrating several insights from our analysis and allowing us to investigate how appropriately joint-trained models allocate units to predict the activity of specific neurons. Finally, we return to the dimensionality bottleneck condition and use further numerical simulations and theory to explain how it is modified by the presence of nonlinearites, constraints on model parameters, and finite training time.

3.1 Introduction

In computational neuroscience, the "computation" or function of a neural circuit is characterized in terms of what input it is provided and what output it produces. In Marr's foundational three levels of analysis [57], it is the most abstract level at which we must be

able to describe a system in order to truly understand it.

With increases in computing power, recent years have seen many attempts to leverage machine learning to investigate computations in the brain. In the supervised context, a computation is defined by a dataset of (x_i, z_i) sample pairs, and learning consists of fitting a model on some training data, $i \in D_{\text{train}}$, to predict z_i from x_i, and evaluating it on other data, $i \in D_{\text{val}}$. The model generally consists of many "units," arranged in some network architecture, connected by weights whose values are tuned during training. Recent work has sought to use the unit activities of such "task-trained" networks, trained with a suitable choice of input and output, for comparison with [24, 49] or prediction of [62, 88] neural responses, y_i, providing a "top-down" approach to analyzing neural circuits by specifying a computation and investigating how well the features learned to perform that computation relate to neural activity. On the other hand, the ready availability of software packages for modern machine learning methods has also allowed for more complex "response-trained" models that predict neural activity from input as an example of a "bottom-up" approach to understand the brain. Both top-down and bottom-up uses of neural networks are valuable; the work of Cadena et. al. [16] specifically focuses on comparing these two related approaches. However, both task-training and response-training have their own shortcomings. Response-training often results in models that have limited generalizability and interpretability. Task-training relies on the assumption that the representation an ANN learns to solve a task is closely related to the representation of the neural circuit. However, in most relevant settings, the solution space for a computation is very large, such that the two representations will have limited similarity, especially at the level of individual features.

In this work, we consider training networks in a joint "bottom-up" and "top-down" manner, which we refer to as "joint-training". Specifically, we consider a cost function for training that consists of a term for the neural response loss and another for the loss of predicting the output of a chosen computation. These two terms are balanced by a hyperparameter, β, that controls how much each term affects the overall cost function. At the $\beta \to 0$ extreme, the computation term disappears and the cost function reduces back to response-training; at the $\beta \to \infty$ extreme, the computation dominates the features learned by the network, which are then used for neural data prediction, equivalent to task-training. Intermediate values of β

compromise between the two approaches; we treat the "task-trained" term as an auxiliary task (see [75] section 7 for a review) for the "response-trained" term, selecting the β that minimizes prediction error of held out responses (see Figure 3.1 for a graphical representation of this framework).

Other work [32, 54] has investigated using such a joint cost function, but with the distinction of focusing on using the addition of neural data to task-training to improve how well a trained model can perform a desired computation. Indeed, the joint-training approach can be seen as an example of multi-task learning [18, 75], where the present work differs from [32, 54] primarily in which task is considered to be of interest, and which plays an auxiliary role.

In the present context, unlike in [32, 54] or in generic multi-class learning (see [26, 51] for theoretical analyses similar to ours in other transfer learning paradigms), we consider the neural data, y_i, to be a *given*, as it is the main quantity we are trying to understand, while the auxiliary task, z_i, represents a *hypothesis* that this computation is related to the functioning of the neural circuit. This allows for us to test multiple candidate z_i, and select one that best allows us to predict the neural data. Here we ask the question of whether a given task will improve neural prediction and which will yield the greatest improvement.

We also wish to understand specific neurons as playing a role in the chosen computation, and joint-training provides a greater opportunity to do this by encouraging sparsity during training in the map between model units and recorded neurons. This mapping then allows the researcher to make targeted hypotheses about the role of specific neurons in terms of the chosen architecture and auxiliary task, to the extent that this mapping precisely and accurately links a recorded neuron to a subset of units in the model. While a maximally precise one-unit to one-neuron map may not be possible in practice, incorporating regularization and thresholding into network training should allow joint-training to learn a more precise map than task-training by encouraging the model to learn features that can be sparsely mapped onto the neural recordings during training, as opposed to attempting sparse selection of features from an already trained model. We ask the question of how the precision of this map, as well as its accuracy and bias (to the extend that such things can be meaningfully determined), is related to the prediction of held-out neural activity across the spectrum of

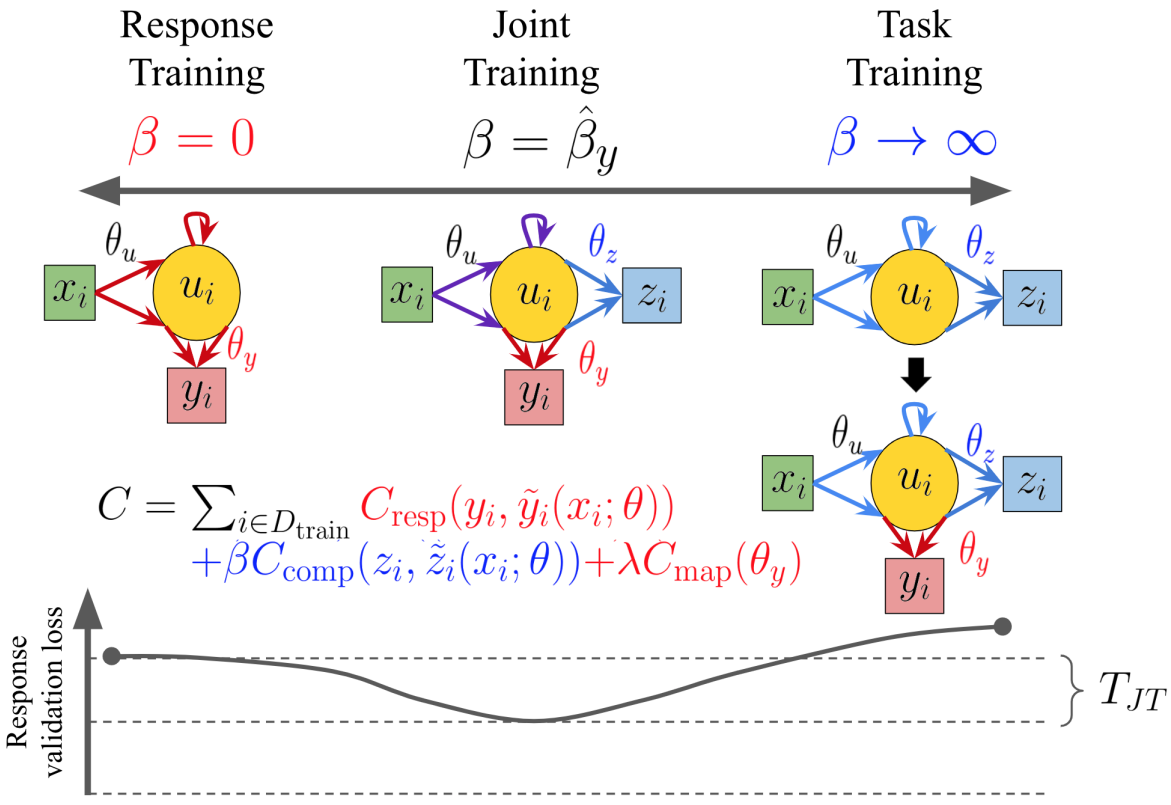


Figure 3.1: **Unifying framework for response-training, joint-training, and task-training, where a hypothesized computation functions as an auxiliary task weighted by the hyperparameter β.** Top: The three modeling paradigms, each corresponding to a specific choice of β. Also shown is the cost function, C, which is minimized with respect to parameters of the model $\theta = \{\theta_u, \theta_y, \theta_z\}$. This figure introduces a color scheme that is respected by all diagramatic figures in this chapter, where green denotes a stimulus (x_i), yellow denotes model units (u_i), red denotes neural data (y_i, boxes) or parameters fitted solely to that output (arrows), blue denotes an auxiliary task (z_i, boxes) or parameters fitted solely to that output (arrows), and purple arrows denote parameters fitted jointly to both neural and auxiliary task outputs. Bottom: We quantify the relative improvement obtained by joint-training over response-training as T_{JT}. We will explore how this quantity depends on the relationship between the given data and chosen auxiliary task.

response-, joint-, and task-training. To this end, we will employ a range of assumptions about how precision is obtained during learning, including both extremes of *maximally sparse* and *dense* (minimally sparse) maps. While the minimally sparse extreme in particular may not be achievable in many situations, we will make use of it as an endpoint of this spectrum that is particularly amenable to a variety analyses.

3.1.1 A hypothetical application of joint-training to neural data

Data from any of the above studies could serve for an application of joint-training, but for clarity and consistency with our analyses (in which we consider data in the form of (x_i, y_i, z_i) triplets), we consider a concrete hypothetical example of applying joint-training to neural data from a visual change detection experiment performed by the Allen Institute [38] in Figure 3.2. In this example, the input x_i consists of a pair of flashed images that were shown to a head-fixed mouse, and the neural data y_i is the mean calcium signal during the second flash. This neural data is the primary target for modelling, and several auxiliary tasks are tested, consisting of output labels z_i associated with the same inputs x_i shown to the mouse. One such task would be the one that the mouse was trained to perform, change detection, but others, which one might hypothesize that the mouse learned during training, before training, or through evolution, would be considered as well. For example such auxiliary tasks could be determining the identity of the second flashed image, or whether it is an image of a predator.

We expect, due to the results of our subsequent analyses, that the auxiliary task that is most predictable from the neural data will be the most useful as an auxiliary task in improving neural prediction. This relationship relies on the assumption that our data consists of (x_i, y_i, z_i) triplets, as opposed to separate neural and auxiliary task datasets with similar inputs. This is in contrast to much work which seeks to leverage the power of huge online datasets, like ImageNet or CIFAR, to provide a greater boost in neural data prediction. While our framework can certainly accommodate this nuance, we exclude it from our analysis for the sake of cleaner results, and consider how it would affect our results in the discussion.

In this example, one might choose a conventional feedforward Convolutional Neural Network (CNN), such as VGG16, for a model, with the two image flashes represented as

different channels in the input layer. The auxiliary task would be predicted from the deepest layer, while the neural recordings would be predicted using the units in every hidden layer. If the model can learn to predict each neuron's activity from a sparse subset of the units, we might be able associate individual neurons with specific stages of processing, corresponding to depth in the VGG16 model, to the extent that specific neurons have distinct roles that roughly correspond to those of units at a specific depth in a CNN. For example, one might expect to see primary visual cortex neurons predicted from earlier layers, and prefrontal cortex neurons from later ones. One might also expect to see differential advantages of different auxiliary tasks for neurons in different brain regions, as these stages of processing might be well described as a sequence of computations. For example, neurons in primary visual cortex might be better helped by a low-level auxiliary task like image identification, while those in prefrontal cortex might be better helped by a higher-level, goal-oriented one like change detection.

3.1.2 Outline of analyses

Under our framework, we analyze the performance of joint-training along three dimensions: improvement in prediction of neural activity, appropriateness of the selected auxiliary task, and quality of the mapping between units and neurons. In Section 3.2, we define our framework and in Section 3.3, we analytically investigate a simple linear model, deriving analytical expressions for performance metrics. Our two main results are:

1. Joint-training cannot improve prediction of held out neural responses unless the model architecture contains a dimensionality bottleneck.
2. The better we can predict the task outputs from the neural data, the more joint-training improves prediction of held out neural responses.

Then, in Section 3.4, we apply joint-training numerically to simulated data, using a three-layer CNN model to assess performance in a more structured, nonlinear network. Here we use our analytical results from Section 3.3 as a lens to better understand these numerical results. We can also investigate more phenomena that are only meaningful to study in more complex networks, such as the mapping between units and neurons. In Section 3.5, we qualify and

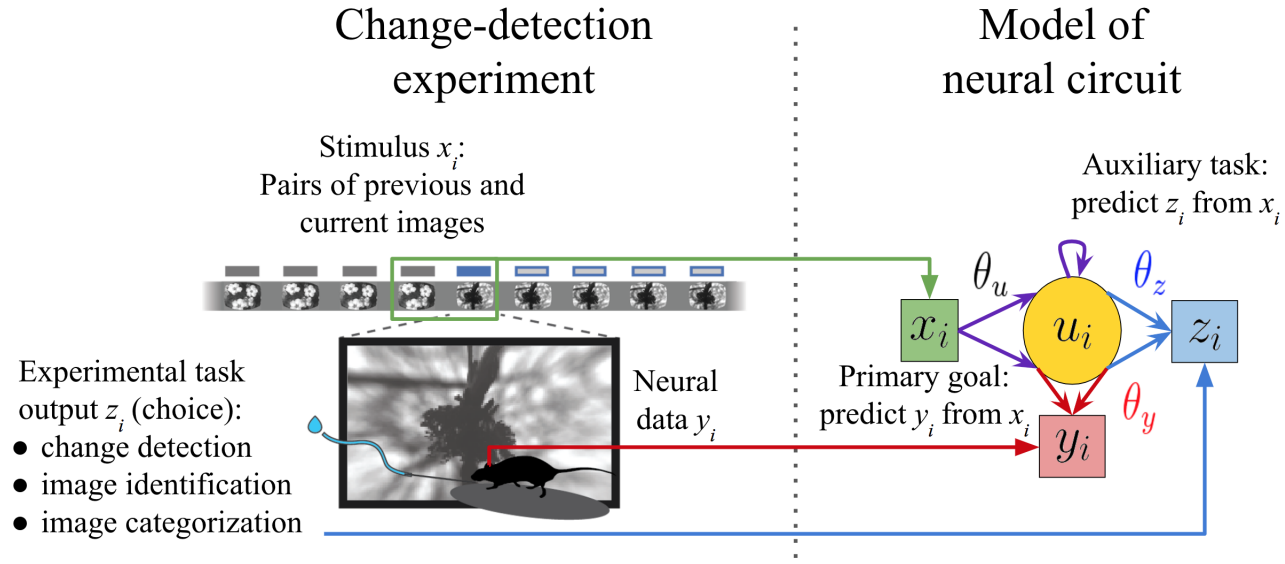


Figure 3.2: **A hypothetical application of joint-training to neural data.** Data is collected from an animal observing a series of image flashes and reporting a detected change to receive a water reward. The primary goal is to predict the average activity of every recorded neuron during a flash from the two most recent image flashes. To accomplish this, we make use of an auxiliary task, a function of the flashed images that we think may be related to the functioning of the recorded neurons.

address the fact that our first main result from the linear theory does not seem to hold in the numerical, nonlinear case. We use theoretical and numerical techniques to demonstrate how nonlinearities, constraints on model parameters, and finite training time affect the determination of when a dimensionality bottleneck is present. We conclude with a discussion of possible interpretations of our results, limitations and extensions of our method, and the potential for its application to neural data.

3.2 The Joint-Training Approach

We begin by defining the joint-training approach in technical terms as a four-step process and using those terms to restate our research questions more precisely.

3.2.1 Model Definition.

Given many samples of a stimulus $x_i \in \mathbb{R}^{S\times 1}$, and neural response data, $y_i \in \mathbb{R}^{M\times 1}$, we *choose an auxiliary task*, defined by the output, $z_i \in \mathbb{R}^{Q\times 1}$, corresponding to each stimulus input. We consider a single partition of these data into training, validation, and testing sets ($i \in D_{\text{train}}$, $i \in D_{\text{val}}$, or $i \in D_{\text{test}}$). We also specify a network architecture with parameters $\theta = \{\theta_u, \theta_z, \theta_y\}$, defined by

$$u_i = F(x_i; \theta_u), \tag{3.1}$$

$$\tilde{z}_i(x_i; \theta) = G(u_i; \theta_z), \tag{3.2}$$

$$\tilde{y}_i(x_i; \theta) = \theta_y u_i, \tag{3.3}$$

where we call u_i the unit activities of the network. Here we consider only feedforward architectures, although the framework could be easily extended to recurrent networks. Here, we also assume a linear relationship between unit activities and neural responses, as (3.3) specifies, although this could also be relaxed.

Finally, we define appropriate cost functions for the neural responses and auxiliary task, $C_{\text{resp}}(y_i, \tilde{y}_i)$ and $C_{\text{comp}}(z_i, \tilde{z}_i)$ respectively. Optionally, we may also impose regularization on θ_y, the linear map between units and neurons, via a cost function $C_{\text{map}}(\theta_y)$ that pushes θ_y, for example, towards a sparser map. A diagram of this setup can be found in Figure 3.1.

3.2.2 Optimizing Network Parameters.

We then minimize the joint cost function on training data with respect to the model parameters:

$$\hat{\theta}_{\beta,\lambda} = \arg \min_{\theta=\{\theta_u,\theta_y,\theta_z\}} \sum_{i\in D_{\text{train}}} C_{\text{resp}}(y_i, \tilde{y}_i(x_i;\theta)) + \beta C_{\text{comp}}(z_i, \tilde{z}_i(x_i;\theta)) + \lambda C_{\text{map}}(\theta_y), \tag{3.4}$$

where $\beta \in [0, \infty)$ and $\lambda \in [0, \infty)$ are hyperparameters that control how much weight to give to each of the three cost functions. If there is no regularization on θ_y, we do not include the third term.

3.2.3 Selecting Optimal Hyperparameters.

Since we are primarily interested in modeling neural activity, we then select the β and, if present, λ, that minimize neural response loss of the validation data:

$$\hat{\beta}_y, \hat{\lambda} = \arg\min_{\beta,\lambda} C^{\text{val}}_{\text{resp}}(\beta, \lambda), \tag{3.5}$$

$$C^{\text{val}}_{\text{resp}}(\beta, \lambda) \equiv \sum_{i \in D_{\text{val}}} C_{\text{resp}}(y_i, \tilde{y}_i(x_i; \hat{\theta}_{\beta,\lambda})). \tag{3.6}$$

3.2.4 Evaluation.

To get a nicely scaled measure of how successful joint-training ($\beta = \hat{\beta}_y$) was on this data with this architecture and auxiliary task, we evaluate the improvement in neural response validation loss relative to that produced by an entirely response-trained network ($\beta = 0$):

$$T_{JT} \equiv \frac{C^{\text{val}}_{\text{resp}}(0, \hat{\lambda}) - C^{\text{val}}_{\text{resp}}(\hat{\beta}_y, \hat{\lambda})}{C^{\text{val}}_{\text{resp}}(0, \hat{\lambda})}. \tag{3.7}$$

For comparison, we can also evaluate the corresponding relative improvement obtained by the task-training approach as:

$$T_{TT} \equiv \frac{C^{\text{val}}_{\text{resp}}(0, \hat{\lambda}) - C^{\text{val}}_{\text{resp}}(\infty, \hat{\lambda})}{C^{\text{val}}_{\text{resp}}(0, \hat{\lambda})}. \tag{3.8}$$

Note that, in the task-training limit, $\beta \to \infty$, only θ_y is influenced by the neural response term in the cost function (3.4).

These improvement metrics are 1 if the corresponding method reduces $C^{\text{val}}_{\text{resp}}$ all the way to zero, 0 if it produces identical $C^{\text{val}}_{\text{resp}}$ to response-training, and negative if it is worse than response-training (note that this is impossible by construction for joint-training as hyperparameter optimization would set $\hat{\beta}_y = 0$ in the worst case).

3.2.5 Questions.

We can now define the three main questions we ask of the joint-training approach in terms of this relative improvement T_{JT}:

1. When does joint-training improve neural response prediction? Specifically, for what data, auxiliary task, z_i, and model properties, is $T_{JT} > 0$?

2. What auxiliary task(s) does joint-training select? That is, for what auxiliary task, $\{z_i | i \in D_{\text{train}}\}$, is T_{JT} maximized?
3. What quality of map from units to recorded neurons, θ_y, does joint-training discover? That is, how few nonzero elements does θ_y have, and do they link neurons to appropriate model units, to the extent that appropriateness can be established?

In this work, we consider two relatively simple network models fitted to data with a known distribution, so that we can quantify the performance of joint-training as accurately as possible. In Section 3.3, both the data-generating network and the joint-trained model have a fully-connected, two-layer structure with no nonlinearities. This simple system allows us to perform the above procedure for joint-training analytically, so that we may gain a better theoretical understanding of the method. However, the simple structure of this network means that all artificial units in the model are interchangeable, as any permutation of the units could be accompanied by the same permutation of the rows of θ_u and columns of θ_y and θ_z. Therefore we cannot use this model to address our third question of joint-training. In Section 3.4, we use a convolutional neural network for both data generation and modeling. This system allows us to numerically demonstrate joint-training's performance with respect to question 3, and test how well the analytical results from the first case generalize to a more complicated structure with nonlinearites.

3.3 Analytical Treatment of a Simple Linear Network

Here we analyze a simple linear network with a dense θ_y map predicting each neuron from all units and derive precise conditions for when joint-training will provide an improvement, addressing our first motivating question. We will then define a special case of this network that has a fixed, maximally sparse map ($\theta_y = I$), and use it to derive an expression for T_{JT}, allowing us to more directly address our second question, and demonstrating a continuity of results across models with different assumptions about the precision of the unit-to-neuron map.

3.3.1 Model Definition.

We define this simple two-layer model with joint cost function (schematized in Figure 3.3):

$$\begin{aligned}\hat{\theta}_u, \hat{\theta}_y, \hat{\theta}_z &= \arg\min_{\theta_u,\theta_y,\theta_z} \sum_{i=1}^{P} \|y_i - \theta_y\theta_u x_i\|_F^2 + \beta\|z_i - \theta_z\theta_u x_i\|_F^2, \\ &= \arg\min_{\theta_u,\theta_y,\theta_z} \left\| \begin{bmatrix} Y \\ \sqrt{\beta}Z \end{bmatrix} - \begin{bmatrix} \theta_y \\ \sqrt{\beta}\theta_z \end{bmatrix} \theta_u X \right\|_F^2 \end{aligned} \tag{3.9}$$

where, in the terms defined in Section 3.2

- $P = |D_{\text{train}}|$ is the number of training samples;
- $\theta_u \in \mathbb{R}^{N\times S}$ are the weights of the first layer, mapping the S-dimensional stimulus onto the N hidden units ($u_i = F(x_i;\theta_u) \equiv \theta_u x_i$, see (3.2));
- $\theta_y \in \mathbb{R}^{M\times N}$ are the weights predicting the M neural responses from the N hidden units ($\tilde{y}_i = \theta_y u_i$, see (3.3));
- $\theta_z \in \mathbb{R}^{Q\times N}$ are the weights predicting the Q-dimensional auxiliary task from the N hidden units ($\tilde{z}_i = G(u_i;\theta_z) \equiv \theta_z u_i$, see (3.3));
- $X \equiv [x_1, ..., x_P]$;
- $Y \equiv [y_1, ..., y_P]$;
- $Z \equiv [z_1, ..., z_P]$;
- and there is no regularization term λC_{map}.

We restrict our focus to cases where the number of samples P is greater than the input dimension S, so that the solution is overdetermined. Otherwise, both (3.9) and (3.10) will have multiple solutions that correspond to different predictions for validation data. If $S > P$, then the first term in (3.10) will have a solution space of dimension $(S-P)M$, with every solution in this space potentially producing a different prediction for validation data. Such underdetermined cases therefore might or might not result in a successful auxiliary task, depending on details of the optimization routine that determine which solution is found.

We have posed this as one problem consisting of two tasks (neural responses and auxiliary task) to be solved jointly, but it may be reducible to two separate linear regression problems:

$$\begin{aligned}\hat{\theta}_{xy}, \hat{\theta}_{xz} &= \arg\min_{\theta_{xy},\theta_{xz}} \sum_{i=1}^{P} \|y_i - \theta_{xy} x_i\|_F^2 + \beta \|z_i - \theta_{xz} x_i\|_F^2, \\ &= \arg\min_{\theta_{xy},\theta_{xz}} \left\| \begin{bmatrix} Y \\ \sqrt{\beta} Z \end{bmatrix} - \begin{bmatrix} \theta_{xy} \\ \sqrt{\beta}\theta_{xz} \end{bmatrix} X \right\|_F^2 . \end{aligned} \tag{3.10}$$

Note that the first term can be minimized solely with respect to θ_{xy}, while the second can be minimized solely with respect to θ_{xz}. If we can indeed reduce (3.9) to (3.10) by finding a $\hat{\theta}_u, \hat{\theta}_y, \hat{\theta}_z$ such that $\hat{\theta}_y \hat{\theta}_u = \hat{\theta}_{xy}$ and $\hat{\theta}_z \hat{\theta}_u = \hat{\theta}_{xz}$, then we can show that all global minima of the joint cost function (3.9) are translatable to a global minimum of the two separate linear regressions (3.10), since the added constraints of the former can only increase the value of the cost function. This means that the performance of solutions to the two problems will be the same on validation data as well as the training data, and there cannot be a helpful effect provided by the auxiliary task. We will consider two cases where this reduction is possible and show that it is impossible in the remaining case (see Figure 3.3 for a visualization of these reductions). For this last case, we then determine how the performance on validation data is affected by joint-training.

Case 1: $N \geq M + Q$

In this case, we can obtain a solution to (3.9) from the solution to (3.10) via:

$$\hat{\theta}_u = \begin{bmatrix} \hat{\theta}_{xy} \\ 0 \\ \hat{\theta}_{xz} \end{bmatrix}; \quad \hat{\theta}_y = \begin{bmatrix} I_M & 0 & 0 \end{bmatrix}; \quad \hat{\theta}_z = \begin{bmatrix} 0 & 0 & I_Q \end{bmatrix}. \tag{3.11}$$

This reduction (or any orthogonal transformation thereof) allows for M hidden units to learn a representation of the neural output, and Q different hidden units to learn the auxiliary task output (see Figure 3.3A).

Case 2: $N \geq S$

In this case, we can obtain a solution to (3.9) from the solution to (3.10) via:

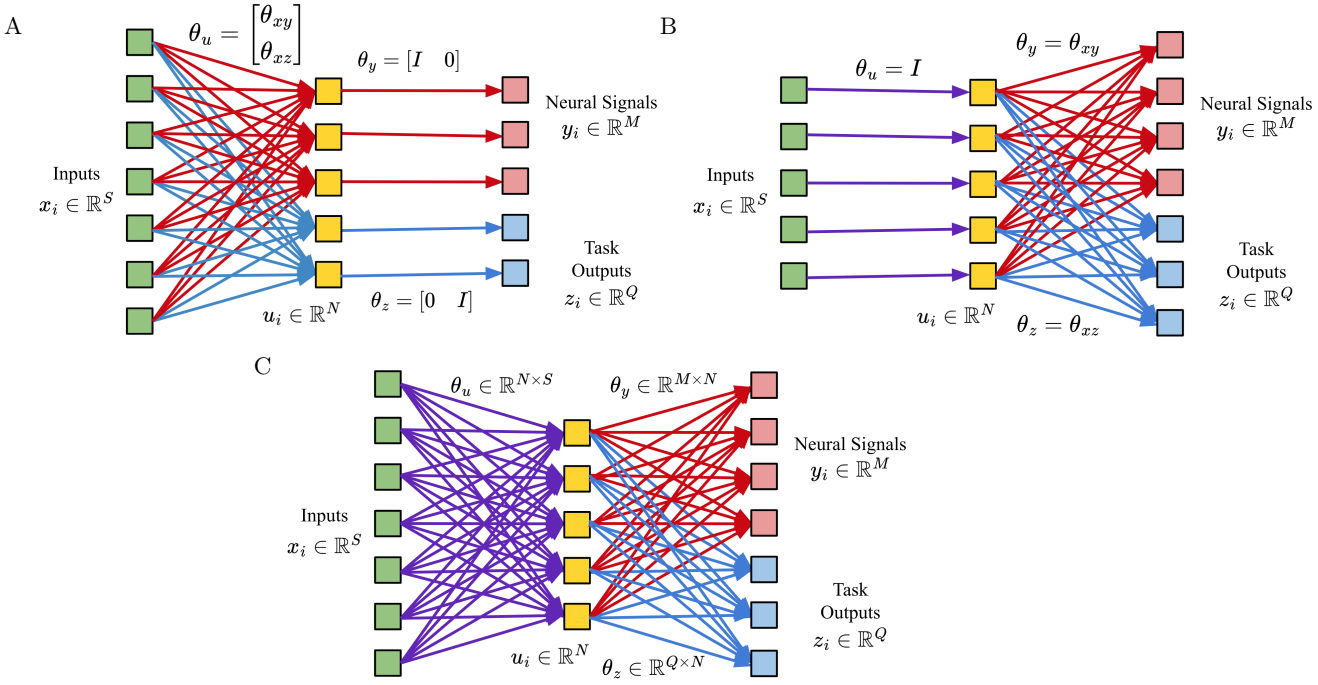


Figure 3.3: **When the linear network does not contain a "bottleneck," it can be reduced to two separate problems.** Each panel shows the model used in Section 3.3, a specific architecture of the form shown in Figure 3.1 and outlined in Section 3.2. See (3.9).

A: When there are too few output dimensions, separate features of the input for the two problems are learned in the first layer.

B: When there are too few input dimensions, all input features are preserved, and the two problems are solved separately in the second layer.

C: When there is a bottleneck, input features that would be used by one of the outputs must be discarded, yielding an inseparable problem.

$$\hat{\theta}_u = \begin{bmatrix} I_S \\ 0 \end{bmatrix}; \quad \hat{\theta}_y = \begin{bmatrix} \hat{\theta}_{xy} & 0 \end{bmatrix}; \quad \hat{\theta}_z = \begin{bmatrix} \hat{\theta}_{xz} & 0 \end{bmatrix}. \tag{3.12}$$

This reduction (or any orthogonal transformation thereof) allows for S of the hidden units to perfectly represent the input, such that the second layer consists of two separate linear regressions from the full input onto the two separate outputs (see Figure 3.3B).

Remaining case

Similarly, we can show that the only remaining case, $\min(M+Q, S) > N$ does not admit this reduction, as the matrix product that predicts the outputs Y and Z is of different rank for (3.9) and (3.10). That is, in the general case where all parameter matrices are full rank, $\text{rank}\left(\begin{bmatrix} \theta_y \\ \sqrt{\beta}\theta_z \end{bmatrix} \theta_u X\right) = N$ and $\text{rank}\left(\begin{bmatrix} \theta_{xy} \\ \theta_{xz} \end{bmatrix} X\right) = \min(M+Q, S)$ (recall we have assumed $P > S$). If, in general, the two models make different predictions, it is impossible to systematically reduce one into the other (see Figure 3.3C).

We can therefore say that $N \geq \min(M+Q, S)$ is a necessary and sufficient condition for being able to reduce (3.9) to (3.10), given that both problems are overdetermined and matrices of model parameters are full rank. However, to fully answer our first question, when joint-training with an auxiliary task will improve performance of the primary task, we must determine when in this remaining case we will actually see improvement.

To that end, we make the following assumptions about the true distribution of the data, $p(x_i, y_i, z_i)$:

$$\begin{aligned} x_i &\sim \mathcal{N}(0, \sigma_x^2 I), \\ y_i &= \theta_{xy}^* x_i + \xi_{y,i}, \\ z_i &= \theta_{xz}^* x_i + \xi_{z,i}, \end{aligned}$$

where $\xi_{y,i} \sim \mathcal{N}(0, \sigma_y^2 I)$ and $\xi_{z,i} \sim \mathcal{N}(0, \sigma_z^2 I)$ are independent Gaussian random variables. A diagram of this process may be found in Figure 3.4B, and contrasted with the model (Figure 3.3C). Note that we may have $\theta_{xz}^* = \theta_z \theta_u, \theta_{xy}^* = \theta_y \theta_u$ for some θ_z, θ_y, and θ_u, in which case

we say that the auxiliary task (θ^*_{xz}) and neural data (θ^*_{xy}) are "compatible," otherwise we say they are "incompatible." We will also use these terms to describe whether specific rows of θ^*_{xz} admit this description.

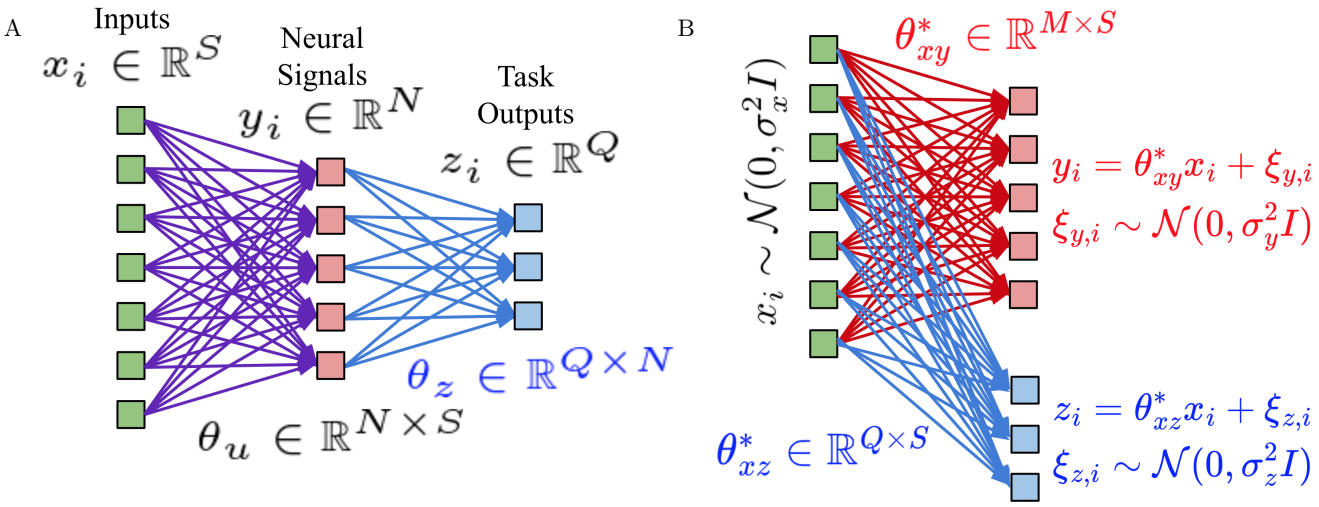


Figure 3.4: **Diagrams of the model and data generator used to derive an expression for T_{JT} (3.15).** A: The simplified model used to derive our main result, which assumes a perfect correspondence between model units and recorded neurons.

B: The process used to generate data throughout Section 3.3 (See (S3.1)). Note that this process may or may not match the model assumptions, depending on whether θ^*_{xz} and θ^*_{xy} are "compatible".

In Section S3.1.3, we simulate data and fit networks with $S = 30$, $N = 10$, and $Q \in \{1, 4, 8\}$.

3.3.2 Optimizing Network Parameters and Selecting Optimal Hyperparameters.

We refer to Sections S3.1.1 and S3.1.2 for the analytical treatment of steps 2 and 3, which use the stated model and assumptions to derive quantities describing how much the inclusion of an auxiliary task improves prediction of held out neural activity.

3.3.3 Evaluation.

Necessary condition for a lack of improvement in the general model Under this general model, we can compute how rapidly prediction of held-out responses improves as we increase β from zero (response-training). If the rate of change in this validation loss is negative, then joint-training will yield better performance than response-training.

As derived in Section S3.1.2, this change in validation loss can be approximated as:

$$\frac{d}{d\beta} C^{val}_{resp}(\beta)\bigg|_{\beta=0} = \frac{-2S\sigma_y^2}{P-S-1} ||\overline{\theta}_z \overline{\theta}_y^\top||_F^2, \tag{3.13}$$

where $\overline{\theta}_y$ and $\overline{\theta}_z$ are fixed matrix approximations for θ_y and θ_z, rather than results of optimization $\hat{\theta}_y$ and $\hat{\theta}_z$ (see Sections S3.1.1 and S3.2 for more details).

Thus, when $P > S$, there will be an improvement (decrease) in validation loss relative to $\beta = 0$ (response-training), unless at least one of the following is true:

1. the number of training samples, $P \to \infty$;
2. the noise in neural recordings $\sigma_y \to 0$;
3. the mapping between neural activity and auxiliary task output, $\overline{\theta}_z \overline{\theta}_y^\top \to 0$; or
4. the dimension of the hidden layer provides a "bottleneck," $N \geq \min(M+Q, S)$ (recall Section 3.3.1).

All of these are conditions when one would not think that including a task in joint-training would improve neural response validation loss: if the neural data is noiseless or there is infinite training data, then neural responses can be predicted as well as possible, if the auxiliary task is incompatible with the neural data, it will not be helpful, and if the network can learn the two tasks separately, then it will do so. While none of these points are counter-intuitive, the exact impact each term has on the cost function is not obvious a priori. This analytical approach allows us to derive such relationships, and thus come to a deeper understanding of joint-training.

The fact that (3.13) is always non-positive is not trivial, as one might imagine that the gradual addition of a completely inappropriate auxiliary task would immediately start hurting prediction of neural responses. The solution suggests that this does not happen because in such a case, $\overline{\theta}_z \overline{\theta}_y^\top \to 0$, reflecting the lack of information about the neural responses in the

auxiliary task, and preventing an effect at $\beta = 0$.

However, the implications of (3.13) are limited in multiple ways. Firstly, even if $\frac{d}{d\beta} C^{val}_{resp}(\beta)\Big|_{\beta=0} = 0$, we have not proven that $C^{val}_{resp}(\beta)$ won't dip below $C^{val}_{resp}(0)$ for a higher value of β, and thus (3.13) being negative is a sufficient, but not necessary, condition for joint-training to produce improvement. Secondly, we cannot make any claims about the model selected by joint-training because we cannot easily compute the optimal value of β. This prevents us from using this model to address our second motivating question about what auxiliary tasks lead to optimal performance. In the next section, we investigate a simplified version of the model that will leave these problems behind. We will still rely on the approximation that the second layer of weights is set to its expected value as $P \to \infty$, and thus employ numerical evaluation to measure its effect in Section S3.1.3.

The final, "bottleneck," condition was derived in such a way that it is a necessary and sufficient condition for joint-training to yield no improvement, but it relies on set of model assumptions (linear, fully connected layers) that are generally violated in practice (we will see an example of this in Section 3.4 where $T_{JT} >> 0$ despite the absence of a bottleneck, as defined above). In Section 3.5, we explore relaxations of these assumptions and take a deep dive into the dynamics of joint-training to understand how early stopping of the optimization routine can lead to improved performance in the full model (3.9) despite the presence of a bottleneck.

Improvement in neural data prediction for a simplified model, and its dependence on auxiliary task Here, we turn our attention to our second motivating question - what choice of auxiliary task will allow for the best prediction of neural data under joint-training? To this end, we make a further model simplification within the remaining case where improvement is possible:

$$\begin{aligned} N &= M \\ \theta_y &= I_N \end{aligned} \tag{3.14}$$

This assumes that our model contains one unit for each recorded neuron, and a perfect

correspondence between model units and recorded neurons has been established (see Figure 3.4A for a diagram). This perfect correspondence, however, is a convenience that we can assume without loss of generality, provided θ_y is square and fixed to some full-rank matrix. In this context, it is a model assumption that θ_y is not learned during training but rather held fixed, whereas before we employed an approximation with similar effect when deriving (3.13).

In Section S3.1.2, we approximate the relative improvement in neural response validation loss (see (3.7)) as:

$$T_{JT} = \frac{\frac{S\sigma_y^2}{N(P-1)} tr[\overline{\theta}_z^\top \overline{\theta}_z]^2}{3\sigma_y^2 tr[\overline{\theta}_z^\top \overline{\theta}_z \overline{\theta}_z^\top \overline{\theta}_z] + \sigma_z^2 tr[\overline{\theta}_z^\top \overline{\theta}_z] + \frac{P-S-1}{S}\sigma_x^2 tr[(\theta^*_{xz} - \overline{\theta}_z\theta^*_{xy})^\top \overline{\theta}_z \overline{\theta}_z^\top (\theta^*_{xz} - \overline{\theta}_z\theta^*_{xy})]} \quad (3.15)$$

First note that, as expected and by construction, T_{JT} is non-negative (in the least-improvement case, joint-training could select $\beta = 0$), and approaches zero (no improvement) if **and only if** at least one the following conditions hold, while all other scalar quantities in (3.15) remain finite and nonzero:

1. the number of training samples, $P \to \infty$;
2. the noise in neural recordings $\sigma_y \to 0$;
3. the spread of the input $\sigma_x \to \infty$;
4. the noise in the auxiliary task output $\sigma_z \to \infty$; or
5. the mapping between neural activity and auxiliary task output, $\overline{\theta}_z \to 0$

The last point in this list is modified slightly from the last set of conditions, reflecting the change in model, and the two before it are new, although they effectively quantify the second point from the first set of conditions: σ_y must approach zero, relative to σ_x or σ_z. The last point requires some additional insight, as in that limit, T_{JT} is undefined; however, if $\theta_z = 0$, then the auxiliary task output will not effect how the rest of the network learns, which is effectively response-training. Note also that $N \to \infty$ is not another such condition because we have assumed that $N < S$, and thus we cannot have $N \to \infty$ without $S \to \infty$. Equation 3.15 provides an analytic expression for whether the inclusion of a task will improve the performance of a neural response model and quantifies the extent of the improvement. Compared to the conditions for zero-improvement we were able to obtain from the full model,

via (3.13), these are more refined and are sufficient conditions, as well as necessary.

We now consider how T_{JT} varies with the choice of auxiliary task, z_i. Under our assumptions about data generation, there are two factors that influence the values z_i will take: the noise added to it, σ_z^2, and the linear transformation from the input, θ_{xz}^*, which effects the optimal transformation from neural activity, $\overline{\theta}_z$ that we use to approximate the results of model fitting.

Note that T_{JT} increases with decreasing $\sigma_x^2 tr[(\theta_{xz}^* - \overline{\theta}_z\theta_{xy}^*)^\top \overline{\theta}_z \overline{\theta}_z^\top (\theta_{xz}^* - \overline{\theta}_z\theta_{xy}^*)]$. This term is closely related to how well one can predict the auxiliary task output directly from the neural activity (see Figure 3.4B):

$$\begin{aligned}\sigma_x^2 tr[(\theta_{xz}^* - \overline{\theta}_z\theta_{xy}^*)^\top \overline{\theta}_z \overline{\theta}_z^\top (\theta_{xz}^* - \overline{\theta}_z\theta_{xy}^*)] &= \mathbb{E}_i[tr[x_i^\top (\theta_{xz}^* - \overline{\theta}_z\theta_{xy}^*)^\top \overline{\theta}_z \overline{\theta}_z^\top (\theta_{xz}^* - \overline{\theta}_z\theta_{xy}^*)x_i]] \\ &= \mathbb{E}_i[tr[(z_i^* - \overline{\theta}_z y_i^*)^\top \overline{\theta}_z \overline{\theta}_z^\top (z_i^* - \overline{\theta}_z y_i^*)]], \qquad (3.16)\end{aligned}$$

where y^* and z^* denote the true responses and auxiliary task output before noise is added ($y_i^* = \theta_{xy}^* x_i$ and $z_i^* = \theta_{xz}^* x_i$). Crucially, when θ_{xz}^* and θ_{xy}^* are compatible, (as we have defined it, i.e. $\theta_{xz}^* = \overline{\theta}_z\theta_{xy}^*$), (3.16) disappears. Thus T_{JT} is maximized when the neural response problem and the auxiliary task are *compatible*.

T_{JT} also increases monotonically with decreasing noise in the auxiliary task output, σ_z^2, and increasing $tr[\overline{\theta}_z^\top \overline{\theta}_z]$ (note that, in general, $tr[A^\top A]^2 \geq tr[A^\top A A^\top A]$). Recalling that we have defined $\overline{\theta}_z = \theta_{xz}^* \theta_{xy}^{*+}$, the latter term, $tr[\overline{\theta}_z^\top \overline{\theta}_z]$, can be increased either by stretching θ_{xz}^* while holding θ_{xy}^* fixed (resulting in $\overline{\theta}_z \leftarrow a\overline{\theta}_z$, scaling all eigenvalues of $\overline{\theta}_z^\top \overline{\theta}_z$ by a^2), or by adding dimensions to θ_{xz}^* that are compatible with θ_{xy}^* (resulting in the introduction of new, positive eigenvalues to $\overline{\theta}_z^\top \overline{\theta}_z$). We can thus conclude that T_{JT} is maximized by an auxiliary task output that has as many dimensions, Q, as possible, all of which are represented in the neural data, and has as high a signal (scaling of θ_{xz}^*) to noise (σ_z^2) ratio as possible.

We can also derive the approximate relative improvement in neural response validation losses obtained by task-training, T_{TT} (see (3.8) for a definition and (S3.9) for a derivation), as:

$$T_{TT} \approx \frac{S}{P-1}\left(1 - \frac{\sigma_z^2 tr[(\overline{\theta}_z \overline{\theta}_z^\top)^{-1}]}{N\sigma_y^2}\right) - \frac{P-S-1}{P-1}\frac{\sigma_x^2}{N\sigma_y^2} tr[(\overline{\theta}_z^+ \theta_{xz}^* - \theta_{xy}^*)^\top (\overline{\theta}_z^+ \theta_{xz}^* - \theta_{xy}^*)] \qquad (3.17)$$

This quantity, unlike T_{JT}, can be negative, i.e. task-training can be worse at predicting held out neural responses than a purely response-trained model. However, T_{TT} obeys the same relationships as T_{JT} with many of the parameters, i.e. it increases with respect to σ_y^2 and N and decreases with respect to σ_x^2 and σ_z^2.

The most important difference is that T_{TT} decreases monotonically with increasing dimension of the auxiliary task, Q, (as both $tr[(\overline{\theta}_z^\top \overline{\theta}_z)^{-1}]$ and $tr[(\overline{\theta}_z^+ \theta_{xz}^* - \theta_{xy}^*)^\top (\overline{\theta}_z^+ \theta_{xz}^* - \theta_{xy}^*)]$ increase with increasing Q), whether the dimensions added to θ_{xz}^* are incompatible or compatible with θ_{xy}^*. This means that if T_{TT} were used to pick the most appropriate auxiliary task, z_i, it would be biased towards lower-dimensional outputs. However, it would pick one that is compatible (eliminating the second term) and has the lowest possible SNR (lowest σ_z^2, greatest scaling of θ_{xz}^*). In the discussion, we address the implications of this distinction between joint- and task-training.

To obtain (3.15) and (3.17), we made use of several approximations, so in Section S3.1.3 we perform joint-training numerically with instances of this simplified network ((3.9), assuming (3.14)), and show that the analytical results are borne out. These analytical results also depend on a very simple model, so in the next Section, we perform joint-training numerically with more complicated models and show that in this case too, the auxiliary task that yields greatest T_{JT} tends to be the one that is best predictable from the neural responses. This setup will also allow us to address our third motivating question regarding the quality of fitted unit-to-neuron maps θ_y. However, we will see that the bottleneck condition derived in Section 3.3.1 appears to be violated, with a network whose input dimensionality (S) exceeds that of a hidden layer (N) allowing substantial improvement in validation loss through joint training ($T_{JT} > 0$), a discrepancy we will address in Section 3.5.

3.4 Numerical Treatment of More Complex Networks with Simulated Data

In the previous Section, we used a very simple network model for the sake of theoretical analysis. Namely, its architecture was linear, shallow, fully-connected, and (for the simplified model) perfectly matched to the neural data (there were an equal number of neurons and units, and the map between them was fixed). Here, we relax these assumptions and show numerically that improved prediction of neural responses is still determined by how well the

auxiliary task output can be predicted from those responses, thereby confirming a central prediction of (3.15) from our theoretical analysis above.

Additionally, because last Section's network model possessed one hidden layer only, and its layers were fully-connected, there were no meaningful difference between all the possible mappings θ_y from artificial units to recorded neurons. This is evidenced by the fact that a network with any full-rank θ_y can be transformed into one with any other θ_y through an appropriate linear transformation of all the network parameters. Here, we use a more structured architecture to show that joint-training tends to find a mapping that is optimal at the layer-to-layer resolution, while task-trained networks cannot do so because their unit's activities are fixed before they have access to the neural responses.

3.4.1 Model Definition.

Just as in Section 3.3, we have a joint probability distribution over inputs, neural activities, and task outputs, as well as a model we fit to P samples from that distribution. Here, we define a multi-layer, nonlinear, convolutional, ANN for both the data distribution and the joint-training model. We will call this data-generating ANN the "data-generator," with "simulated neurons," whose activities, y_i, will be supplied to our joint-training model as if they were neural recordings. We assume that the specific architecture of the joint-training model (entailing layer sizes, nonlinearities, and convolutional structure; F, in the terms of Section 3.2) is correctly specified to match that of the data-generator. We consider data generated by an ANN that has already been trained on a task, which we refer to as the "true" task. This task may or may not be the same as the auxiliary task, z_i, used in fitting the joint-trained model (see Figure 3.5).

In order to robustly demonstrate our results, we use a low-dimensional input, x_i, so that we can easily simulate many networks with many different experimental conditions. For this we choose the `digits` dataset of labeled images of handwritten digits, (accessed via `sklearn.datasets.load_digits` [68], see Figure 3.5 for an example sample) and specify a feedforward, three-layer, convolutional ANN architecture used to both simulate neural data and fit the joint-training model (see Figure 3.5 and Table 3.1). For both the true and

auxiliary tasks, using the labels ($d_i \in \{0, ..., 9\}$) provided in the `digits` dataset, we consider four potential task outputs:

- digit classification ($z_i = d_i$),
- parity classification ($z_i = \mathbb{1}(d_i \in \{0, 2, 4, 6, 8\})$),
- "loop" classification ($z_i = \mathbb{1}(d_i \in \{0, 6, 8, 9\})$), and
- random network for the data generator, ℓ_2 regularization for the model,

where $\mathbb{1}$ denotes the indicator function, which is 1 if its argument is true and otherwise 0.

While the parity classification task is not associated with any relationship to the shape of the handwritten digits, loop classification focuses on the presence of a topological feature (a closed loop) in the form of a digit. As the last option, we consider the absence of a true or auxiliary task as control cases: for the data generator, we use an untrained ANN with parameters sampled from a Gaussian distribution in place of a true task; for the joint-training model, we replace the auxiliary task with standard ℓ_2 regularization ($C_{\text{comp}} = \frac{1}{2}||\theta_u||_F^2$).

Layer	Input	1 (conv1)	2_a (conv2)	2_b (mp)	3 (fc1)	Output (fc2)
Dimensionality	48	384	256	64	128	10, 2, or 0
channels×rows×columns	$1 \times 8 \times 6$	$16 \times 6 \times 4$	$32 \times 4 \times 2$	$32 \times 2 \times 1$		
Nonlinearity	N/A	ReLU	ReLU	2×2 MaxPool	ReLU	Softmax

Table 3.1: **CNN architecture used for both the data generator and joint-training model with the `digits` dataset.** The activities of Layers 1-3 are concatenated and "recorded" from the data generator as y_i; in the model they are referred to as the "unit activities," which are mapped onto y_i via θ_y. In our analysis, units/neurons from layers 2_a and 2_b are grouped together.

Just as in Section 3.3, we use the mean squared error loss for our (simulated) neural responses:

$$C_{resp} = \frac{1}{2P} \sum_{i=1}^{P} ||y_i - \tilde{y}_i||_F^2. \tag{3.18}$$

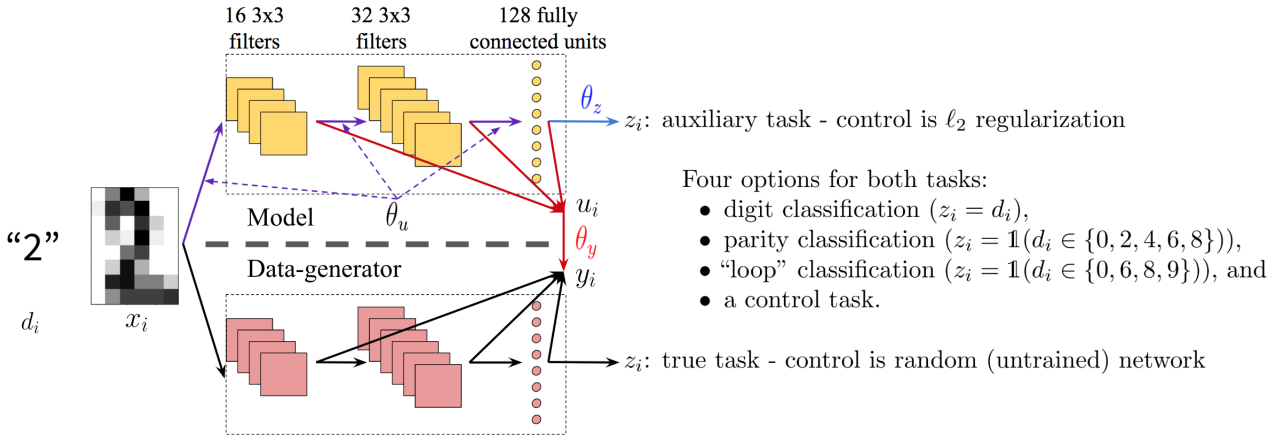


Figure 3.5: **Auxiliary tasks and nonlinear CNN architecture used for generating data and joint-training in Section 3.4.** The `digits` dataset consists of $x_i \in \mathbb{R}^{8\times 6}$ greyscale images and $d_i \in [0, ..., 9]$ labels of digit identity. We use this convolutional architecture for both the data generator (bottom half) and the joint-training model (top half) - see Table 3.1 for details. Before generating data, the data-generator is trained on one of four "true" tasks; one of four auxiliary tasks, which may or may not correspond to the true task, is used for joint-training the model. The model is a specific instance of the general formulation presented in Section 3.2 and Figure 3.1.

We do have a θ_y matrix that learns the mapping from units to simulated neurons (3.3). In Section 3.4.5, we will consider models with minimal (paralleling the full (3.9) version of our linear model) and variable precision in the units-to-neurons map, demonstrating continuity of the main results that follow across this precision spectrum and introducing some new results that are unique to those models.

Here, we focus on a maximally sparse θ_y, with one nonzero element in each row and column. Doing so provides the greatest consistency with the simplified model ((3.14)) we used to derive our main results, (3.15) and (3.17) and the most straightforward answer to our third motivating question regarding the quality of the units-to-neurons map. Even though we will only assess the accuracy and bias of θ_y at a *layer-to-layer* resolution, constraining θ_y to be maximally sparse facilitates this. To push θ_y towards maximal sparsity, we constrain

the elements of θ_y to be non-negative, and use a penalty $C_{map}(\theta_y)$ (with hyperparameter λ) during the first half of training:

$$C_{map}(M) = \sum_{r,c} M_{r,c} \left(\sum_{r' \neq r} M_{r',c} + \sum_{c' \neq c} M_{r,c'} \right). \tag{3.19}$$

Note that this penalty is zero when, for each row and for each column of M, there is only one nonzero element. Note that this assumes that the same unit will not be used to predict multiple neurons, an assumption that is not necessary to achieve maximal sparsity, but may be desirable. During the second half of training, we replace this regularization with hard threshholding on θ_y, which we denote $\theta_y \leftarrow H(\theta_y)$, setting all elements to zero, except for the greatest in each row:

$$H(M)_{r,c} = \begin{cases} M_{r,c} & \text{if } M_{r,c} > M_{r',c} \; \forall r' \neq r \\ 0 & \text{otherwise} \end{cases} \tag{3.20}$$

3.4.2 Optimizing Model Parameters

We use `pytorch` [67] with the `Adam` optimizer to minimize our joint-cost function with respect to the network parameters θ. To maintain an appropriate learning rate for each parameter group (θ_u, θ_z, and θ_y), we divide the base learning rate (10^{-3}) by the summed weight of the terms of the joint cost function that affect that parameter group. That is, the learning rate for θ_u is $10^{-3}/(1+\beta)$, that for θ_y is 10^{-3}, and that for θ_z is $10^{-3}/\beta$. Optimization proceeds for the first half of training with regularization (3.19), and then switches to hard threshholding (3.20) for the second half. Learning rates are further halved if the loss has not decreased in 20 epochs (not including the transition between regularization and thresholding), down to a minimum rate of $10^{-4}/(1+\beta)$. Unless otherwise noted, the network is trained for 300 epochs in total.

3.4.3 Selecting Optimal Hyperparameters

We train the model (Step 2) for a grid of logarithmically spaced $\beta \in [10^{-5}, 10^{5}]$ and $\lambda \in [10^{-6}, 10^{-2}]$ on a training set of 1024 samples, then evaluate the response loss C_{resp}^{val} on

the remaining 773 samples. In order to reduce the effect of randomness inherent in fitting neural networks, we train 40 separate networks on the same simulated data with the same hyperparameters, and average C_{resp}^{val} across them. We select the β, λ pair that produces the lowest C_{resp}^{val}, and use them to compute T_{JT}, according to (3.7). While in practice, one might want to select as a single final model the one with the lowest loss, the selection of hyperparameters should be more robust when the average loss (over the 40 separate networks) is used, since taking the minimum would be much more prone to noise. A single final model could still be chosen as the one with the lowest $C_{resp}^{val}(\hat{\beta}_y, \hat{\lambda})$ (using the $\hat{\beta}_y, \hat{\lambda}$ selected using the averaged loss).

3.4.4 Evaluation

We perform joint-training for all 16 combinations of true and auxiliary task. For each true task, the same generated data is used for all four auxiliary tasks. We first ask whether the trends in neural response prediction improvement, T_{JT}, match what we would expect from our analytical treatment in Section 3.3, i.e. (3.15). Namely, we ask whether we observe improvement, and for which auxiliary task that imporvement is maximized. We then evaluate the learned mapping from model units to simulated neurons, $\hat{\theta}_y$, in terms of how accurately and fairly it links neurons from a given layer of the data generator with units in the corresponding layer of the model.

When is T_{JT} positive?

In our analytical treatment, T_{JT} is non-negative by construction, approaching 0 in specific circumstances. In the present empirical context, we generally find a positive T_{JT}, as can be seen in the y-axis of Figure 3.6A. By contrast, task-training produces predictions that are worse than those of the response trained model ($T_{TT} < 0$ in Figure 3.6B).

We note that extrapolating our derived dimensionality bottleneck condition to this context, using the layer sizes in Table 3.1, one would conclude that joint-training should not produce positive T_{JT}. We will address this apparent discrepancy in Section 3.5, but presently address our second question, which auxiliary task maximizes this improvement.

Which auxiliary task maximizes T_{JT}?

From our analysis in Section 3.3, we expect T_{JT} to be inversely related to the loss of predicting the output z_i directly from the simulated neural recordings y_i (see (3.16)). To compute this quantity from our generated data, we use logistic regression with ℓ_2 regularization, using the same training-validation split as is used for the model.

Because, as described in Section 3.3.3 above, the term describing this loss of predicting z_i from y_i in (3.15), $tr[(\theta^*_{xz}-\overline{\theta}_z\theta^*_{xy})^\top\overline{\theta}_z\overline{\theta}_z^\top(\theta^*_{xz}-\overline{\theta}_z\theta^*_{xy})]$, increases linearly with the dimensionality of the auxiliary task output, Q, (if we assume that each dimension added to the auxiliary task is equally incompatible with the neural data), we consider the scaled negative validation loss of predicting z_i directly from y_i,

$$\text{scaled negative loss} = \frac{1}{Q|D_{\text{val}}|}\sum_{i\in D_{\text{val}}}\log(P(z_i|y_i)), \tag{3.21}$$

where $|D_{\text{val}}|$ is the number of samples in the validation set, and $P(z_i|y_i)$ is the prediction of the logistic regressor described above.

Figure 3.6A shows that indeed this quantity correlates very well with T_{JT}, although neither corresponds with whether the true and auxiliary tasks match. Each solid line shows the T_{JT} values obtained for a single dataset of simulated neural responses, using three different auxiliary tasks. The upward trend with the negative rescaled loss (3.21) for each of these is consistent with our analytical result using a simple linear network (3.15), interpreted via (3.16). Performance using the "control auxiliary task," which is not really an auxiliary task but just ℓ_2 regularization, is encoded in the height of the dashed line, whose color, like that of the solid lines, encodes the simulated neural dataset (note that it is not meaningful to predict this task using the simulated neural data). We did not address this in our linear theory, but expect that this control task should provide some improvement, since ℓ_2 is generally useful regularization, and that it should be most useful, relative to the non-control auxiliary tasks, for the simulated neural data taken from the untrained network. This is indeed the case, although the full digits classification task is still more useful than the control task, for simulated neural data from the untrained network.

By contrast, task-training in this instance produces worse response predictions than

response-training (negative T_{TT}), generally yields better performance from the control task than the non-control tasks, and does not demonstrate such a clear relationship between the accuracy of these predictions and how well the auxiliary task can be predicted from the neural data (Figure 3.6B). Worse performance than joint-training is to be expected, since by construction joint-training selects models with the most improvement, and the weaker trend is consistent with task-training's preference for lower-dimensional tasks derived using the linear theory (3.17).

We claim that the result that the auxiliary tasks that yield higher T_{JT} do not correspond to the true tasks (where the line and fill color match in Figure 3.6) is due to the fact that every network trained to classify a subset of digits is actually learning to perform full digit classification. This is directly evidenced by the x-axis position of the blue dots in Figure 3.6. We will return to this point later in Section 3.4.4, after we have presented more results that allow for a more complete treatment of this subject.

Does joint-training correctly map simulated neurons to units in the appropriate layer?

Here, we investigate properties of the mapping between model units and simulated neurons, $\hat{\theta}_y$, discovered by training with a joint cost function, and show that models that achieve greater improvement in C^{val}_{resp} (joint-training, by construction, achieves the greatest improvement) also lead to better such mappings.

Generally, we desire a mapping $\hat{\theta}_y$ that allows for improved model interpretability, such that one can attempt to understand the role of populations of recorded neurons in terms of populations defined by the model architecture. Here, we evaluate how well $\hat{\theta}_y$ corresponds with ground truth at the resolution of layers in our feedforward structure (with layers 2a and 2b considered as one layer), as invariances introduced by the model architecture make measurement at any higher resolution much more complicated. Specifically, we will evaluate layer-wise *accuracy*, measuring how often neurons from a given layer of the data-generator are mapped to units in the corresponding layer of the model, and layer-wise *bias*, measuring how much neurons tend to be mapped to a layer in the model that is deeper or shallower

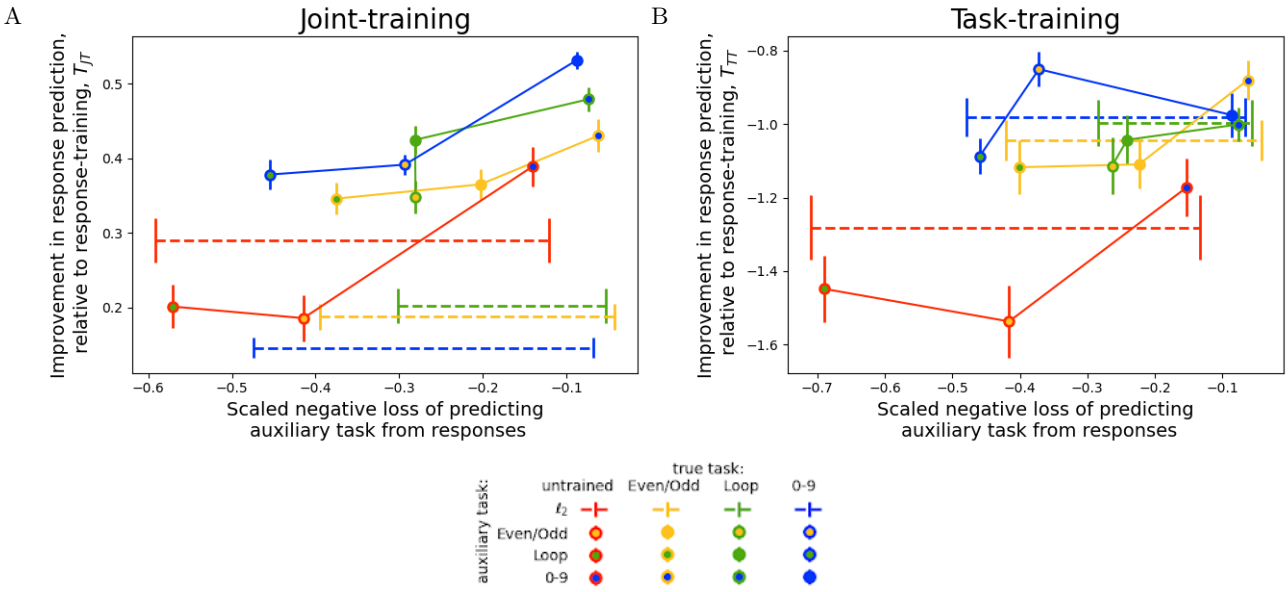


Figure 3.6: **Joint-training provides better prediction of simulated neural responses and selects higher-dimensional auxiliary tasks, relative to task-training.** Each panel shows how relative improvement over response-training afforded by joint-training (panel A, T_{JT}, (3.7)) or task-training (panel B, T_{TT}, (3.8)) is related to how well the task outputs can be predicted directly from the neural responses (3.21). Line (fill) color encodes the true (auxiliary) task used to train the data generator (fit the model). Simulated data for a given true task is identical across trials and auxiliary tasks, so each solid line joins predictions of the same simulated data. When ℓ_2 regularization is used in place of an auxiliary task, a dashed horizontal line is used (whose color encodes true task), since there is no task to predict. Error bars indicate the mean ± SEM over 40 model fits.

than their corresponding layer.

In order to come up with appropriately scaled metrics that describe $\hat{\theta}_y$ at layer-to-layer resolution, we measure the fraction of neurons in layer i of the data-generator that are mapped to units in layer j in the model as:

$$W_{i,j} = \frac{1}{|L_i|} \sum_{n \in L_i} \sum_{u \in L_j} \mathbb{1}([\hat{\theta}_y]_{n,u} \neq 0), \tag{3.22}$$

where L_i is the set of indices for units/neurons in the ith layer of the network architecture (in this case, $i \in \{1, 2, 3\}$), and $\mathbb{1}$ is the indicator function. See Table 3.1 for the layer sizes, $|L_i|$. Using this measure, we define the class-balanced, layer-wise accuracy of $\hat{\theta}_y$:

$$\text{accuracy} = \frac{\sum_i \frac{1}{|L_i|} W_{i,i}}{\sum_{i,j} \frac{1}{|L_j|} W_{i,j}}. \tag{3.23}$$

Note that unless there is exactly one non-zero element of $\hat{\theta}_y$ per row and column, this is not the standard class-balanced classification accuracy. However, it is always an appropriately normalized accuracy metric (between 0 and 1) in terms of the density of connections between layers of the data-generator and model. Similarly, we define the class-balanced layer-wise bias as:

$$\text{bias} = \frac{\sum_{i<j} \frac{1}{|L_j|} W_{i,j} - \sum_{i>j} \frac{1}{|L_j|} W_{i,j}}{\sum_{i \neq j} \frac{1}{|L_j|} W_{i,j}}. \tag{3.24}$$

Note that bias is positive when $\hat{\theta}_y$ contains too many connections from too deep in the model to too shallow in the data-generator, and vice-versa (ranging from -1 to 1). See Figure 3.7 for a graphical representation of these metrics.

We find that, across all true and auxiliary tasks and hyperparameter settings, the layer-wise accuracy and bias of $\hat{\theta}_y$ both show dramatic improvement with increasing ability to predict held out neural activity (Figure 3.8A and D, respectively). We quantify the latter, which is represented by the x-axis in these panels, similarly to T_{JT} and T_{TT}:

$$T \equiv \frac{C^{val}_{resp}(0, \hat{\lambda}) - C^{val}_{resp}}{C^{val}_{resp}(0, \hat{\lambda})}. \tag{3.25}$$

Like T_{TT} (and unlike T_{JT}) this value can be positive or negative, depending on whether the model in question predicts held out neural responses better or worse than response-training.

While layer-wise accuracy improves monotonically with T, the absolute value of layer-wise bias initially increases (worsens) with increasing T as the mapping becomes more structured than one would expect by chance (the dashed red line in Figure 3.8A). It is only right around $T = 0$, when there is actually an improvement in C^{val}_{resp}, that the bias begins to decrease. This

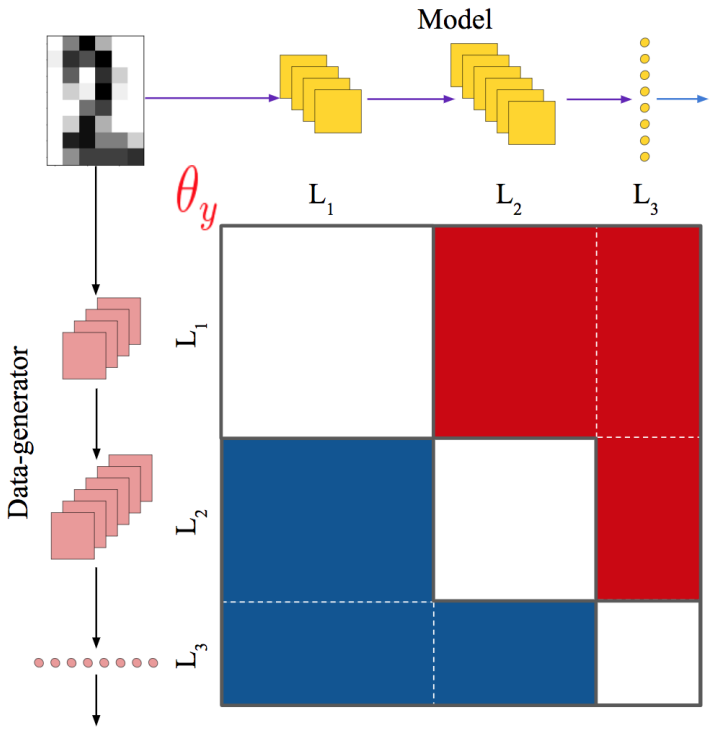


Figure 3.7: **Diagram of layer-wise accuracy** (3.23) **and bias** (3.24) **metrics for the fitted units-to-neurons map, $\hat{\theta}_y$.** The matrix shown represents a fitted $\hat{\theta}_y$, and each block corresponds to connections from a specific layer of the model to a specific layer of the data-generator. The non-zero values of $\hat{\theta}_y$ in the i, jth block are counted and normalized to measure the fraction of units in the ith layer of the model mapped to the jth layer of the data-generator (see (3.22)). Accuracy is the class-balanced sum of these values from diagonal blocks, normalized to $[0, 1]$ (grey, see (3.23)). Bias is the difference between the class-balanced sum over the red blocks and that over the blue blocks, normalized to $[-1, 1]$ (see (3.24)).

relationship between quality of the mapping $\hat{\theta}_y$ and improvement in neural prediction loss T, means that the process of joint-training effectively finds models whose structure corresponds as closely as possible with that of the neural data by choosing the $\hat{\beta}_y$ that produce the largest T. While this is a sensible result, it is by no means trivial - in general there are many, meaningfully different solutions to network training that will produce similarly accurate

predictions of held out neural data, but joint-training is able to find those that best align with the true structure of the data-generator. We will see in Section 3.4.5 that indeed such trends are not always guaranteed, as making networks sparser can lead to better prediction of held out neural activity, but only up to a point.

We can also ask directly what quality of mappings are produced by joint-training by looking only at the models with selected hyperparameters. When we restrict our view to these, we see that the trends in accuracy and bias better match what we would expect from the relationship between true and auxiliary task, rather than the pattern we observed for T_{JT} (Figure 3.8B and E). That is, accuracy is high and bias is of low magnitude when the true and auxiliary tasks match. We also note that, when the auxiliary task is digit classification ($[0-9]$), the bias is from too deep in the model to too shallow in the data-generator, and otherwise it is the opposite, when it differs greatly from no bias. When ℓ_2 regularization is used in place of an auxiliary task, in particular, there is a very strong bias from too shallow in the model to too deep in the data generator.

In contrast, the units-to-neurons map discovered by task-training obtains only chance levels of accuracy (Figure 3.8C) and is biased towards using model units in early layers (Figure 3.8F). This can be understood as a result of the task-trained network learning its representation without the influence of the representation present in the neural data. Thus the two representations of the input should diverge as the depth from the shared input increases, making units in earlier layers of the model more useful for predicting neural activity.

Why is the true task used for training the data-generator not always the best auxiliary task?

We are now prepared to ask why those selected computations that are best represented (and thus produce the highest T_{JT}) in the simulated neural data are not always the same as the "true" task used to train the data-generator. While a thorough treatment of this phenomenon is beyond the scope of this work, we propose that it arises because of specific relationships among the tasks. That is, the digit classification task ($[0-9]$) best captures the structure of the input, in such a way that it might be discovered by an unsupervised clustering algorithm.

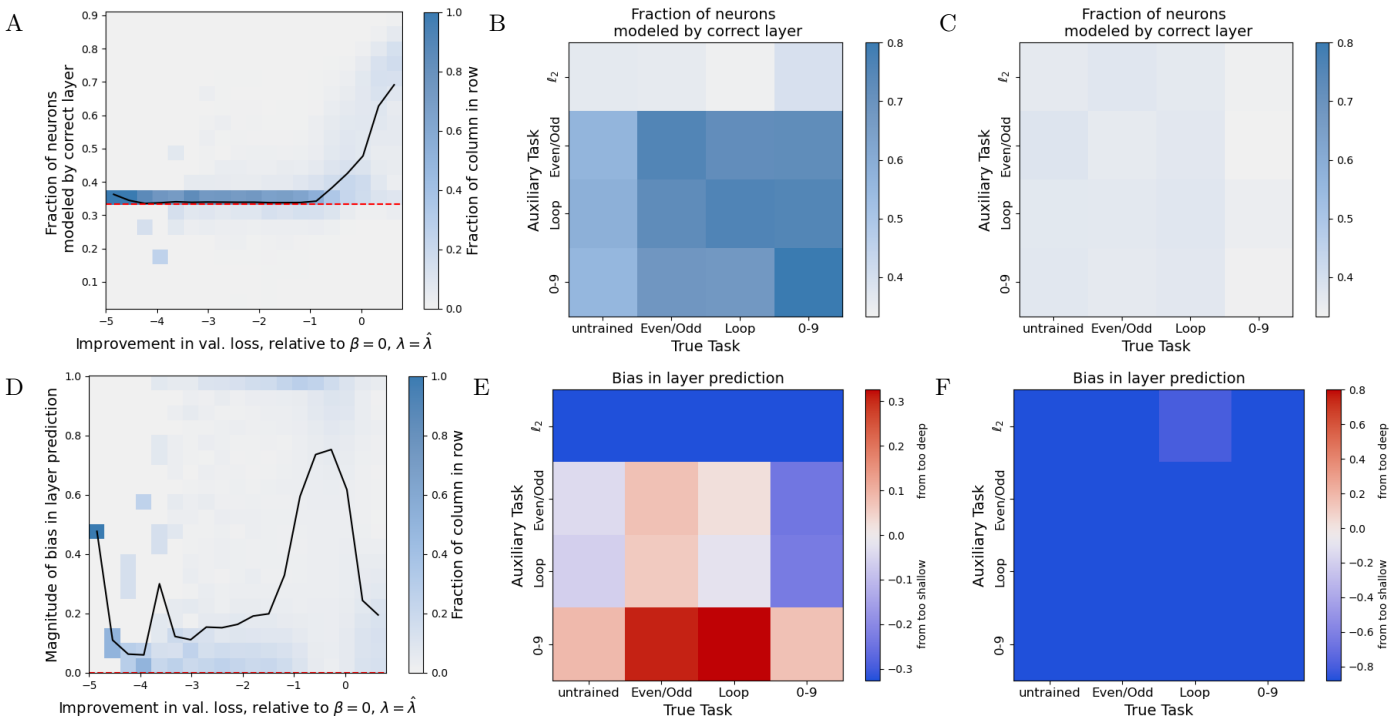


Figure 3.8: **Joint-training discovers best map between model units and simulated neurons when neural response loss is most improved.**

A: Heatmap across different true and auxiliary tasks, hyperparameters, and model fits, of the layer-wise accuracy of the map between model units and simulated neurons, θ_y, (3.23) versus the improvement in loss of held out neural data, C^{val}_{resp}, relative to the baseline, $C^{val}_{resp}(0, \hat{\lambda})$. Columns are normalized to sum to one, and the solid black line shows the mean of data in each column. The dashed red line shows chance levels (1/3, since there are 3 layers).

B: Layer-prediction accuracy, averaged over models fit with the optimal hyperparameters, for each true and auxiliary task.

D: Same as panel A, except showing the absolute value of bias in layer prediction (3.24) on the y-axis. The dashed red line here is at 0. Note that joint-training selects hyperparameters that maximize C^{val}_{resp}, and thus find maps between units and neurons that are as accurate as possible and have low bias.

E: Layer-prediction bias, averaged over models fit with the optimal hyperparameters, for each true and auxiliary task. Red (blue) color of a cell denotes a $\hat{\theta}_y$ with too many connections from units deeper (shallower) in the model than the neuron they map to, in the red (blue) blocks in Figure 3.7.

C and F: Same as B and E, but using $\beta \to \infty$ (task-training).

The other two task outputs (parity and loop classification), however, are not as natural a division of the inputs into classes, but can be easily computed from the outputs of digit classification.

This means that a natural way for a feedforward network to learn the parity (or loop) task may be to learn digit classification in its lower layers, and then trivially compute parity (or loopiness) in the final layer. Not only would this explain why digit classification is so well represented in networks trained on a different task (blue dots are the farthest right in Figure 3.6), but also the layer bias obtained by joint-training (Figure 3.8E). When one network that is trained on digit classification is matched with another that is trained on parity (or loop) classification, there will be high similarity between the last layer in the former and the penultimate layer of the latter, because in both that is where the digit classification outputs are best represented. This will produce the pattern of layer bias observed in Figure 3.8E, in which a model trained with full digit classification as an auxiliary task is biased towards using units layers that are too deep to predict data from a model trained on parity classification, and vice versa.

It is also worth noting that ℓ_2 regularization uniformly pulls all weights of the model towards 0, such that supporting a large signal deeper in the network incurs penalties from more layers. This leads to the regularized model using its lower layers to model simulated neurons from all layers, resulting in the observed bias.

3.4.5 Models of the digits dataset with different precision of the units-to-neurons map

To this point, we have enforced maximal sparsity of our units-to-neurons map, θ_y, using a regularization (3.19) and thresholding (3.20) scheme. We have done this to present a clean set of results from a model most similar to the one in which we derived our main result, (3.15), (which assumed a perfect correspondence between units and neurons – see (3.14), Figure 3.4A), that admits simple measurement of the quality of the units-to-neurons map. However, in practice such precision may not be attainable or desirable, so here we explore results from models with less precise units-to-neurons mappings.

We will demonstrate that generally, the main results using the maximally sparse map carry over to when a dense map or a variable-sparsity map is used. However, prediction of simulated neural data using the dense map shows less improvement under auxiliary tasks, compared to the control task, ℓ_2 regularization. The variable-sparsity map also allows us to show that joint-training selects models with maps that are relatively sparse (in terms of associating each simulated neuron with fewer model units), as well as accurate and unbiased, compared to response-training and task-training.

Minimally sparse (dense) units-to-neurons map θ_y

First, we will consider the least sparse mapping, where all units are mapped onto all neurons, and no sparsity at all is promoted in θ_y. This corresponds more closely to our unsimplified linear model (3.9), which we found theoretically to have similar conditions for when joint-training would result in improvement in response prediction (compare (3.13) and associated discussion with (3.15)). Here, we explore numerically how similarly they perform in terms of which auxiliary tasks produce the most improvement, and how improvement is related to the quality of the units-to-neurons map, θ_y.

Which auxiliary task maximizes T_{JT}? Figure 3.9 shows the same results as those in 3.6, relating the improvement in neural prediction afforded by an auxiliary task to how predictable it is from the neural data, but for models with a minimally sparse θ_y. The main result derived in (3.15) and observed in Figure 3.6A, that an auxiliary task yields greater improvement under joint-training when it is better predictable from the responses, is also generally observed in Figure 3.9A (although notably it does not hold when the true task is the even/odd task). Likewise, this trend does not hold when task-training is used (Figure 3.9B), with higher-dimensional auxiliary tasks generally yielding lower improvement, despite higher predictability, as expected from (3.17) and observed in Figure 3.6B. As with the results using the maximally sparse units-to-neurons map (Figure 3.6), the "control" auxiliary task, ℓ_2 regularization of θ_u, yields the greatest (least) improvement under joint (task) -training, when simulated neurons are recorded from a random, untrained network.

The key difference here is that with the dense map, the improvement joint-training

produces with the non-control tasks is much lower, and specifically lower than when ℓ_2 regularization is used. These decreases in relative performance can be understood in terms of the dimensionality bottleneck criterion: when a maximally sparse mapping is used, neurons predicted by model units that are deeper in the network than a bottleneck where information in the stimulus must be discarded will admit improvement in their responses due to joint-training. However, when a dense mapping is used, as it is here, model units in the layers before the bottleneck can be recruited to predict all responses, to some extent eliminating what would be a bottleneck if the model units deeper in the network were required. Since the bottleneck is necessary for an auxiliary task but not ℓ_2 regularization to yield improvement, this explanation can account for the observed shift.

While task-training produced worse performance than response-training with the maximally sparse map (negative T_{TT} in Figure 3.6B), now it is astronomically worse, in relative terms (note the 10^7 scale of the y-axis in Figure 3.9B). This large scale demonstrates the level of overfitting that is possible using all task-trained features together to predict neural activity.

Does joint-training correctly map simulated neurons to units in the appropriate layer? To answer this question, because θ_y is not sparse in this case, we first adjust our layer-to-layer connectivity score accordingly (see (3.22):

$$W_{i,j} = \frac{1}{|L_i|} \sum_{n \in L_i} \sum_{u \in L_j} [\hat{\theta}_y]_{n,u}. \tag{3.26}$$

Layer-wise accuracy and bias metrics for this case are defined as before (see (3.23), (3.24)), but using this new connectivity score which takes the magnitude of connections into account, as opposed to just their existence (as before in (3.22)). Note that this is not a trivial difference, as differences in the magnitude of $[\hat{\theta}_y]_{n,u}$ may reflect differences in the scale of the nth neuron's activity and/or the uth unit's activity, and thus the resulting metrics may be harder to interpret.

Indeed, we see that joint-training's bias towards using units from too deep in the network (Figure 3.10E) does not match our expectation from Figure 3.8E, but this may be due

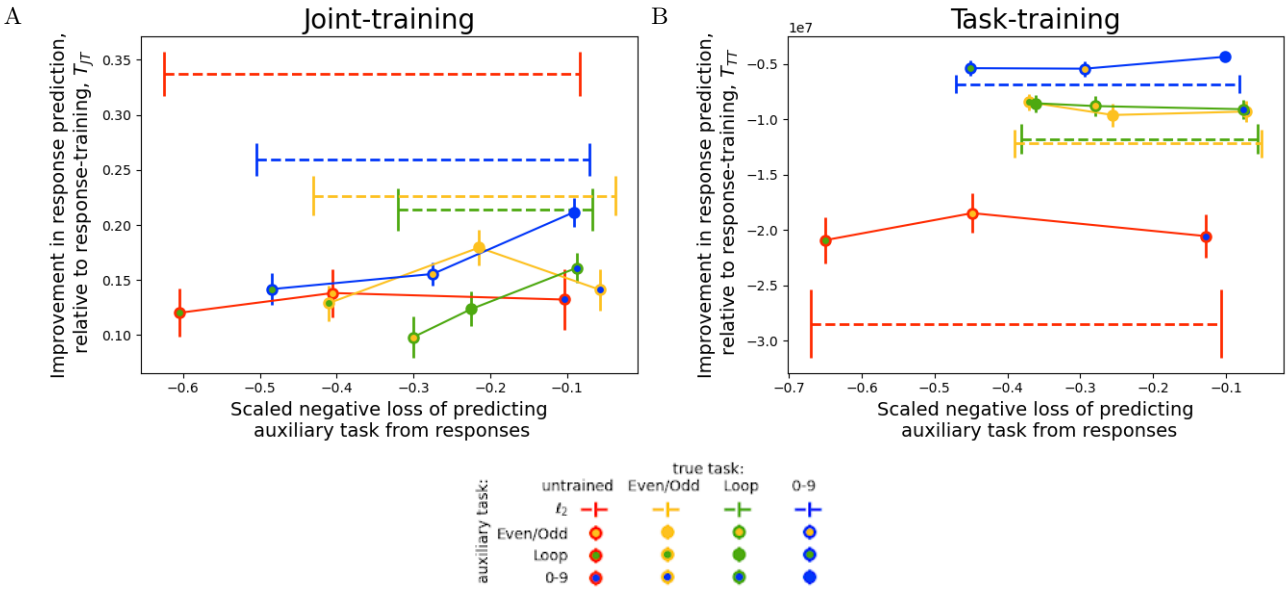


Figure 3.9: **Joint-training provides less improvement with a dense units-to-neurons map, but that still corresponds with predictability of auxiliary task from responses.**

Same results as Figure 3.8, but without any constraints placed on the units-to-neurons map θ_y. The main trend is upheld, that joint-training, more than task-training, is greatest when the auxiliary task is best predicted from the responses. However, here the resulting values of T_{JT} produced by all auxiliary tasks are much lower, especially relative to the control, ℓ_2 regularization.

smaller activities of those units or the fact that their activities represent features of the input aggregated across greater portions of the image. This is consistent with the difference in bias produced by task-training between the two models: whereas the maximally sparse map apparently uses units that are too shallow (Figure 3.8F), the dense map uses units that are too deep (Figure 3.10F). The difference in metric for the dense map may also account for the generally lower layer-wise accuracies depicted in Figure 3.10A, 3.10B, and 3.10C, compared to the corresponding panels in Figure 3.8.

Despite these differences, we see that the main result from Figure 3.8 is upheld: model fits

that attain greater improvement in prediction of responses also obtain unit-to-neuron maps that are more accurate (Figure 3.10A) and less biased (Figure 3.10D) at the layer-to-layer resolution.

Variable-sparsity units-to-neurons map θ_y using ℓ_1 regularization

Now that we have investigated models with units-to-neuron maps at both extremes of the spectrum from lowest sparsity (dense θ_y, Figures 3.9 and 3.10) to highest (maximally sparse θ_y, Figures 3.6 and 3.8), we consider an intermediate level of sparsity, obtained by using standard ℓ_1 regularization and thresholding. This is achieved using the scheme described in Section 3.4.1, but with

$$C_{map}(M) = \sum_{r,c} |M_{r,c}|, \tag{3.27}$$

$$H(M)_{r,c} = \begin{cases} M_{r,c} & \text{if } M_{r,c} > 10^{-6} \\ 0 & \text{otherwise} \end{cases}. \tag{3.28}$$

Which auxiliary task maximizes T_{JT}? As expected, the results for models with intermediate sparsity in θ_y interpolate between those using the dense and maximally sparse θ_y. Most importantly, the main trend of better response prediction by joint-training associated with greater predictability of the auxiliary task from responses is preserved (Figure 3.11A, even clearer than the trend in Figure 3.6A). When ℓ_1 regularization is used to sparsify the units-to-neurons map, the best auxiliary tasks yield greater improvements than ℓ_2 regularization of θ_u, except when the data generator was not trained on a task. The results for task-training (Figure 3.11B), relative to those using maximally sparse and dense maps, are intermediate, in scale of T_{JT}, relative positions of results for the different datasets (line colors), and selected auxiliary task for each dataset.

Does joint-training correctly map simulated neurons to units in the appropriate layer? Because the ℓ_1 penalty promotes a high degree of sparsity, we can use the original formulation of layer-to-layer connectivity (3.22) to define our accuracy and bias metrics for

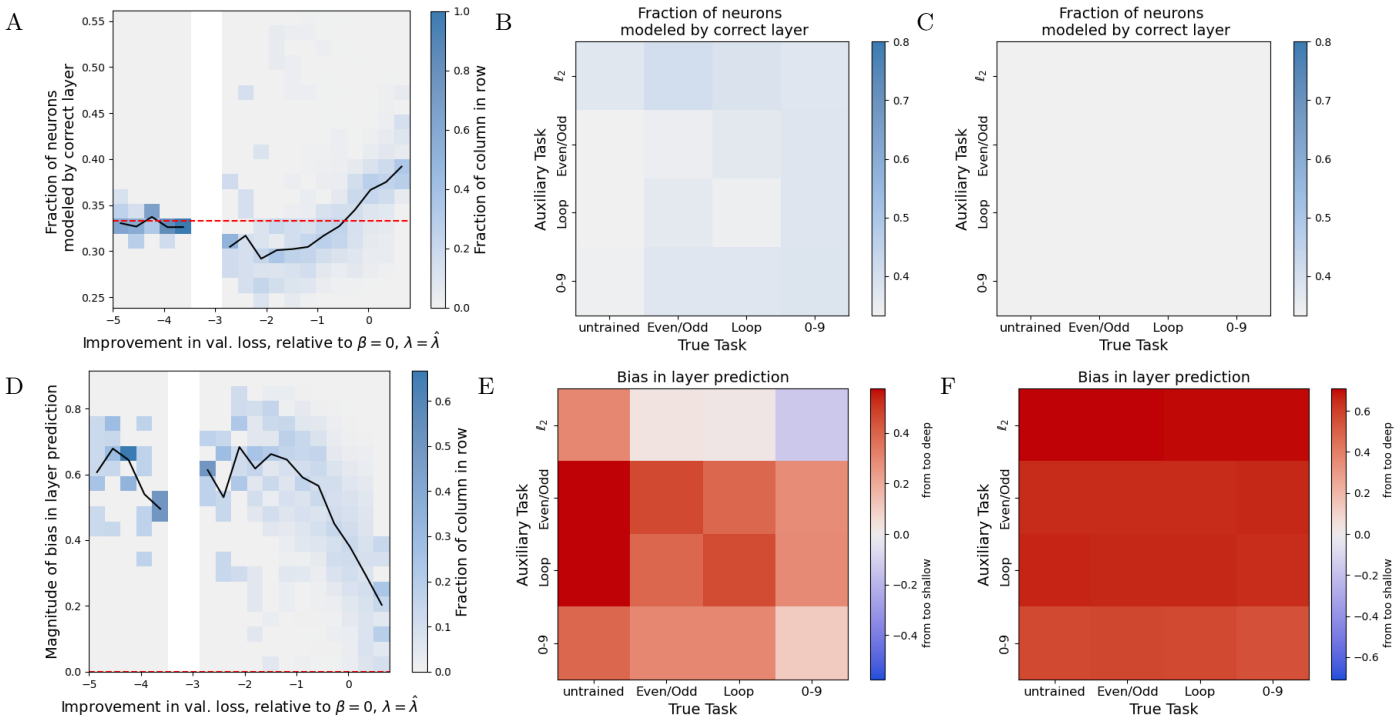


Figure 3.10: **Joint-training discovers best dense map between model units and simulated neurons when neural response loss is most improved.** Same results as Figure 3.8, except using an unconstrained θ_y, and a correspondingly different method for computing layer-wise metrics (3.26). The main trend, that accuracy and bias improve with improving response prediction, is shared by both models. However, the level of accuracy achieved and the specific pattern of bias as a function of true and auxiliary task is very different, possibly due to the necessarily less precise formulation of these metrics.

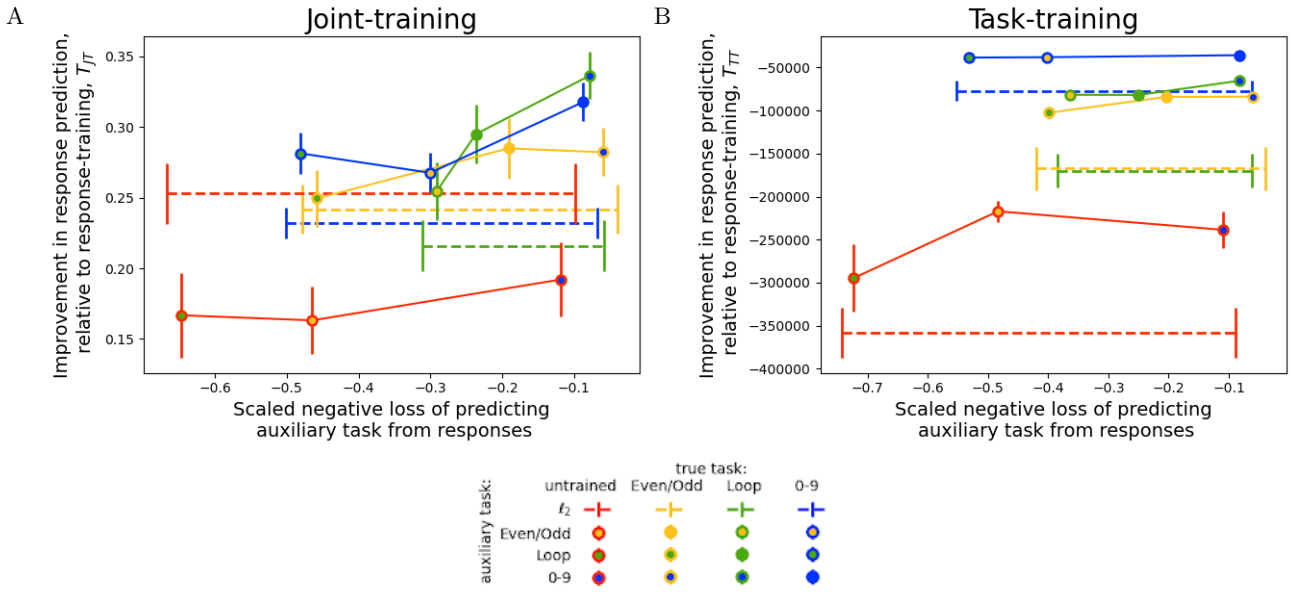


Figure 3.11: **Joint-training provides intermediate improvement with a variable sparsity map, but T_{JT} still corresponds with predictability of auxiliary task from responses.** Same results as Figure 3.8, but using ℓ_1 regularization and thresholding to promote sparsity in the units-to-neurons map θ_y. The main trend is upheld, that the improvement in predictions of held-out neural data afforded by joint-training is greatest when the auxiliary task is best predicted from the responses, unlike with task-training. The improvement afforded by auxiliary tasks, relative to the control, is more consistent with the desired relationship (greater improvement from the auxiliary tasks, compared to ℓ_2 regularization, except when simulated neural recordings are from an untrained network) than using either maximally or minimally sparse maps (compare to Figures 3.6A and 3.9A respectively).

$\hat{\theta}_y$, consistently with those used in Figure 3.8. Additionally, because the level of sparsity achieved varies across model fits, we can investigate how the level of sparsity achieved using ℓ_1 corresponds with improvement in response prediction (Figure 3.12G). As with the maximally sparse and dense maps, we observe that the layer-wise bias decreases as response predictions improve past the performance of response-training (Figure 3.12D), but layer-wise

accuracy is too low to discern a trend (Figure 3.12A). Similarly, better response predictions correspond with more sparse mappings, but only up to a point - the most sparse maps result in predictions slightly worse than those made by response-trained networks, but the best response predictions are made using maps that are almost as sparse.

While some features of how layer-wise bias is related to true and auxiliary task (Figure 3.12E and F) were present when a maximally sparse mapping was enforced (Figure 3.8E and F), the overall patterns are quite different. This difference may be related to the lower scale of bias associated with the layer-wise accuracy that is closer to chance levels (larger denominator in (3.24)). It may also be that an uneven allocation of nonzero elements of $\hat{\theta}_y$ across layers of the model and/or data-generator may be skewing the results, similarly to what we observed with the dense units-to-neurons map.

Takeaways from comparing models with different levels of sparsity in their units-to-neurons map

Comparing all these numerical results across the three different levels of sparsity in the units-to-neurons map we investigated (maximally, minimally, and variably sparse) highlights the important role that constraints on this map may play. The main result that auxiliary tasks better predictable from neural data lead joint-training to make better predictions of held out neural data holds across each of these cases, although this trend clearer in some cases than others. However, the unregularized, dense map resulted in much worse performance in terms of improved prediction of held out responses (lower T_{JT} in Figure 3.10 than Figure 3.8 or 3.12) and selection of an appropriate auxiliary task (highest T_{JT} for every dataset (line color) in Figure 3.10 is ℓ_2 regularization (dashed line), whereas in Figure 3.8 and 3.12 this is only the case for data from an untrained (random) network). In Section 3.5, especially Section 3.5.1, we will further explore the importance of constraints on the units-to-neurons map and seek to understand it in terms of our analytical results from Section 3.3.1 regarding dimensionality bottlenecks.

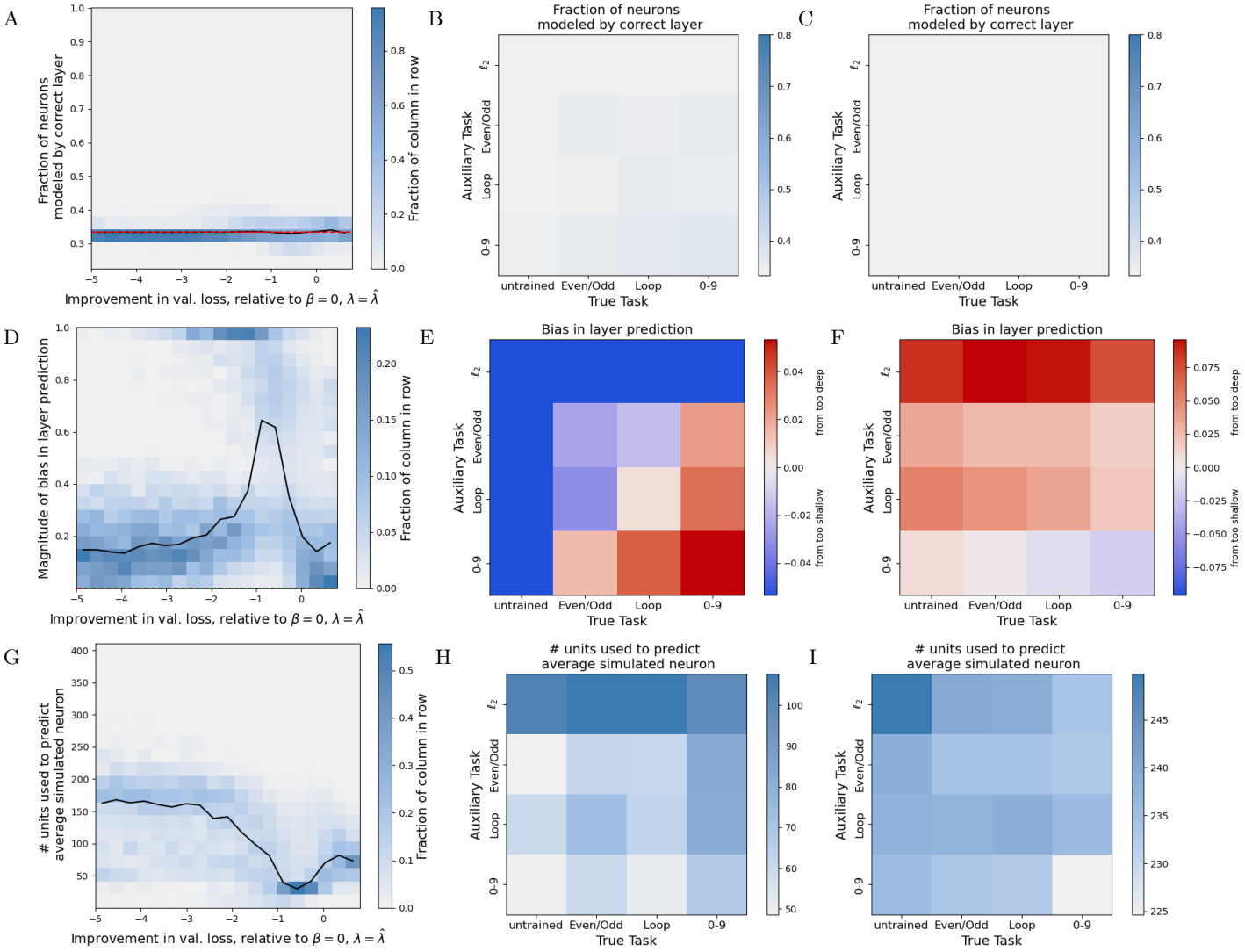


Figure 3.12: **Joint-training discovers best ℓ_1-sparsified map between model units and simulated neurons when neural response loss is most improved.**

Same results as Figure 3.8, except using ℓ_1 regularization (3.27) and thresholding (3.28) to constrain θ_y. Additionally, panels G, H, and I show the level of sparsity achieved by all fits, joint-training, and task-training respectively, in the same style as the other rows of this figure. Differences in layer-wise accuracy are too small to observe a trend, but greater improvement in response prediction corresponds with unit-to-neuron maps that are sparser (up to a point) and less biased.

3.5 Is a dimensionality bottleneck necessary for joint-training to yield improved response prediction?

Here, we consider in more detail the "bottleneck" condition, $N \geq \min(M + Q, S)$ (recall Figure 3.3), which stipulates that an auxiliary task will not provide improved prediction of neural responses unless the dimensionality of the hidden layer is less than that of both the input and combined output (see Section 3.3.1). We first note that this result can easily be extended to a network with more hidden layers, in which case N will be replaced by the smallest dimensionality of any hidden layer. To see this, we can simply consider θ_u to be the matrix product of the network layers up to this narrowest layer, and θ_y and θ_z to be the product of subsequent layers leading to each respective task. This assumes that only the last layer is used to predict both outputs, as we will do for much of this section. If instead earlier layers are used to predict recordings, and there are no restrictions on which unit(s) can be used to predict a given neuron, than this amounts to replacing N with the width of the shallowest layer used to predict recordings.

When we consider deep, nonlinear networks, this bottleneck condition is violated by most standard architectures, including the one we use in Section 3.4.5 (see Table 3.1), and yet, as we have seen, there is still the potential for joint-training to produce a significant improvement in predictive accuracy. We consider three potential reasons why a feedforward network model can effectively possess a bottleneck, even when a hidden layer has more units than the input or combined outputs:

- Nonlinearities can provide an information bottleneck through noninvertible transformations.

- Constraints on model parameters induced by architecture or regularization can induce a bottleneck by requiring some level of redundancy among units in a layer in terms of how they represent the input.

- Early stopping of training can provide an effective bottleneck because modes of the input-output cross-covariance matrix with the largest singular values are learned first.

In Section 3.5.1, we explore the extent to which each of these three factors allowed for successful joint-training in our simulations with the digits dataset. In Sections 3.5.2 and 3.5.3, we restrict our focus to the third explanation, learning dynamics, and undertake a more thorough analysis using simulations and theory respectively.

3.5.1 How do nonlinearities, regularization, and early stopping affect joint-training on the digits dataset?

At first glance, the results of Section 3.4, namely the large improvements in response prediction through joint-training, seem inconsistent with the bottleneck condition, as the number of stimulus dimensions ($6 \times 8 = 48$) is less than the dimensionality of the smallest hidden layer (64 units in layer 2_b). Therefore we ask the natural question: which of the ways in which this CNN differs from a linear, fully-connected network give rise to this discrepancy?

For greatest consistency with the model we used to derive the bottleneck condition, we consider first a version of our CNN with a dense units-to-neurons map (no regularization term $C_{\text{map}}(\theta_y)$) and where only $M = 41$ neurons (about 5% of the total number) are recorded, so that both bottleneck conditions are violated (Figure 3.13A, but see also Figure 3.13B, where a maximally sparse map is used to predict all recorded neurons, and Figure 3.13C and D, where we consider a range of recording densities with the maximally sparse map). We use full digit classification as both the true and auxiliary task ($Q = 10$). We record the T_{JT} values obtained for a range of learning durations to test how much of the improvement comes from earlier stopping. For comparison, we also perform this analysis for a linearized version of our CNN architecture, where the ReLU nonlinearity is replaced by the identity transformation, and the MaxPool layer is replaced by an AveragePool layer of the same dimension, in order to test how much of the improvement is due to information discarded by the nonlinearities.

The difference achieved by joint-training in improvement between the dense map (y-scale of Figure 3.13A, also of Figure 3.6A) and the maximally sparse map (y-scale of Figure 3.13B, also of Figure 3.9A) may be understood in terms of a constraint on parameters promoting a bottleneck. This is easiest to understand for the linearized model: when a dense map is used, the first layer will be capable, alone, of making a prediction for neural activity that

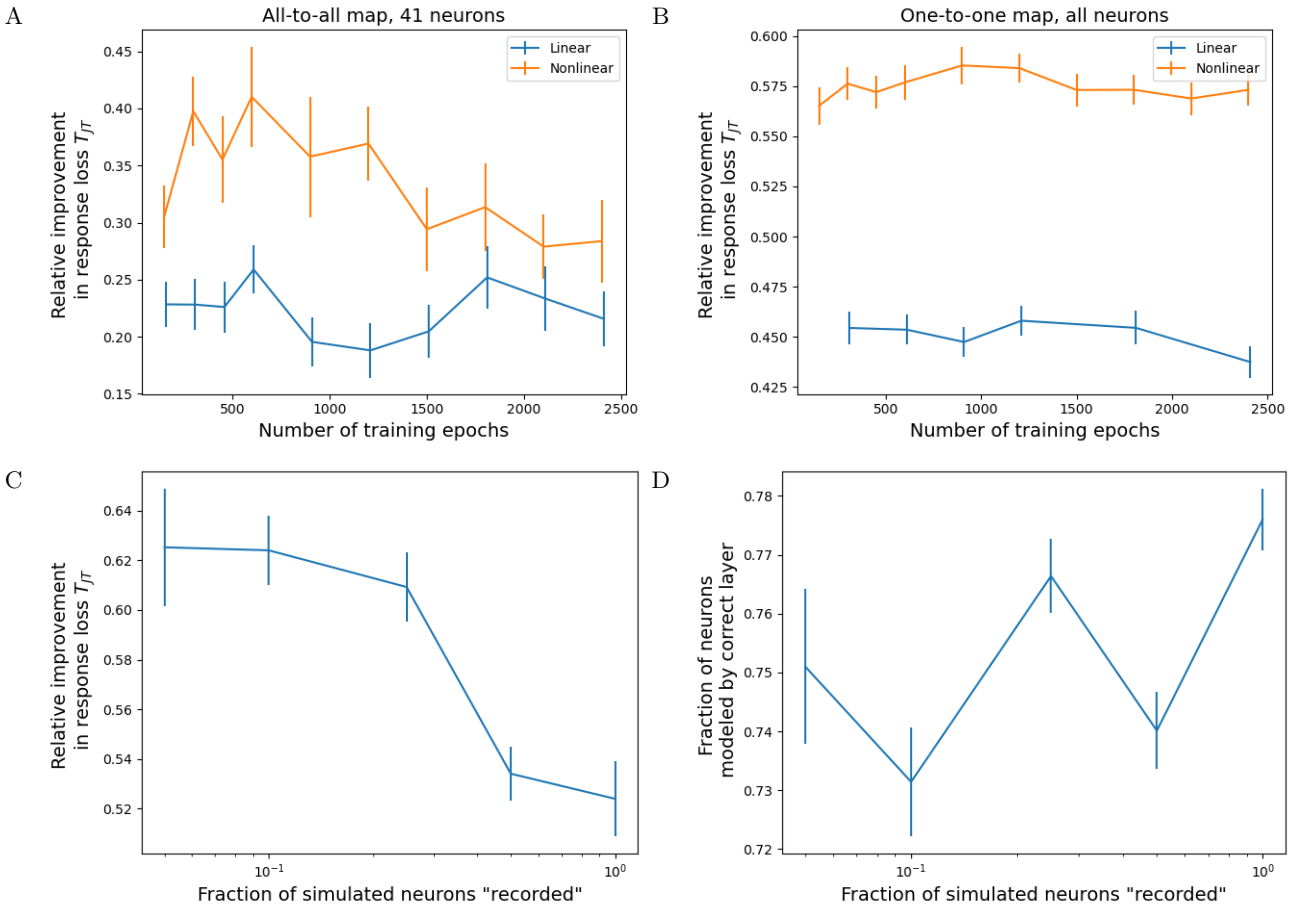


Figure 3.13: **Nonlinearites, early stopping, and constraints on model parameters at least partially explain improvement in a network with no dimensionality bottleneck.** Panels A and B show how nonlinearity (line color) and shorter training time (x-axis) generally increase T_{JT} (y-axis) for two different versions of the CNN model used in Section 3.4. Panels C and D then highlight the specific effects of varying the number of simulated neural responses. A: Dense map between units and neurons, with only 5% of neurons ($M = 41$) supplied as data. B: A maximally sparse map between units and neurons (see (3.19) and (3.20)), with all neurons used results in higher T_{JT} overall. C: Relative improvement in response validation loss T_{JT} ((3.7), y-axis) is greater, when fewer simulated neurons are provided as input to nonlinear networks with maximally sparse unit-to-neuron maps. D: Layer-wise accuracy of mapping between model units and simulated neurons ((3.23), y-axis) does not show as clear a trend for these same networks. All panels show the mean ± SEM over 40 fits.

uses any combination of units in the model. Thus the constraint of maximal sparsity in the units-to-neurons map makes it possible for the architecture of the deeper layers of the network to introduce a bottleneck. In this architecture, the pooling layer takes inputs from four units and reduces them to one unit, effectively throwing away three units worth of information. Even though the resulting number of units (64) is greater than the number of input dimensions (48), the way the convolutional layers feed signals into the pooling layer prevents the network from creating the redundancy that would be necessary to preserve all of the stimulus information in this layer.

We also see that, indeed, the linear network (orange lines in Figure 3.13A and B) and longer training time (greater x-axis values) generally result in less improvement (lower T_{JT}), but that T_{JT} does not approach zero. The fact that T_{JT} does not approach zero for later stopping with the unregularized linear network (blue line in Figure 3.13A) may be due to finite training time or other factors related to the optimization process we have not considered.

We also observe in Figure 3.13C that in general a smaller number of recorded neurons M leads to greater relative improvement in predictions of their activity, although there is not a clear trend for changes in layer-wise accuracy (Figure 3.13D). As we proposed above, our example CNN has a bottleneck built into the architecture at the pooling layer. Thus, as long as any recorded neurons come from the output of the pooling layer or deeper, this bottleneck will remain. Otherwise, we might expect a steep drop in T_{JT} when then the number of recorded neurons falls below a certain number.

Our analysis of the full model (3.9) predicts the rate of improvement in predicting held-out neural responses with β from $\beta = 0$, and this formula (3.13) suggests that adding recorded neurons should always yield greater initial improvement, by appending columns to the argument of the Frobenius norm. Thus, extrapolating the result in (3.13) to attempt to predict T_{JT} does not account for this trend.

However, we might also employ our result for T_{JT} with the maximally sparse map (3.15) by considering a reduction of the number of recorded neurons M without changing the number of hidden units N. After this reduction, there are $N - M$ hidden units that are not associated with a recorded neuron, and are thus free to represent any $N - M$ dimensions of the auxiliary task. Suppose that they learn to represent dimensions of the task that are

not compatible with the neural data, as all compatible dimensions can be learned from the other units that are associated with recorded neurons. We could then effectively reduce this case back to our simplified model by removing these $N - M$ hidden units and at the same time removing the $N - M$ dimensions of the auxiliary task they represent. Since these dimensions were incompatible with the neural data, this will reduce the rightmost term in the denominator of (3.15) more than the reduction in N and correspondingly in $tr[\overline{\theta}_z^\top \overline{\theta}_z]$, amounting to a reduction in T_{JT}. This explanation of course assumes that there are incompatible dimensions of the auxiliary task to remove. This implies that the improvement in T_{JT} that results from a removal of neurons from y_i should be smaller when the auxiliary computation is well chosen, effectively making it harder to select an appropriate computation. Here, we have only investigated this effect when both the true and auxiliary task are digit classification, but it is of interest to future work whether or not this prediction holds across a variety of auxiliary tasks.

So far, our analysis has allowed us to make claims about to what extent each specific factor (architecture, nonlinearities, training time) contributes to improvement under joint-training. We have already provided some commentary for how parameter constraints from architecture and regularization might introduce a bottleneck by preventing all the information in the stimulus from reaching a certain depth in the network. Nonlinearities accomplish this in a more straightforward way when they are noninvertible by mapping many input values to the same output value, discarding information about which of those input values produced a given output. Training time is considerably trickier to understand without further analyis, but fortunately is quite amenable to analysis using the linear model (3.9) from Section 3.3 which we used to derive the bottleneck result. In Section 3.5.2, we will numerically fit this model to simulated data, isolating the effect of training time and painting a clearer picture of what the *learning trajectories* of joint-training are that allow for improved prediction of neural responses. Then, in Section 3.5.3, we will use a theoretical treatment of these learning dynamics to help understand *why* joint-training produces such dynamics.

3.5.2 Simulations with linear networks reveal positive T_{JT} up until late in training, due to slow growth in effective network dimensionality

To assess whether the absence of a dimensionality bottleneck ($N \geq \min(S, M+Q)$) in a fully-connected, linear network performing joint-training prevents performance improvements, and how this depends on when training is stopped, we simulated learning in many instances of (3.9).

Model Setup

In order to test many different cases with and without bottlenecks, we sampled the shape of the network and data as uniformly distributed random variables. Specifically, where $\mathcal{U}(a, b)$ denotes a random variable that takes on one of the integers between a and b (inclusive) with equal probability:

- $P \sim \mathcal{U}(35, 50)$,
- $S \sim \mathcal{U}(10, 41)$,
- $N \sim \mathcal{U}(10, 32)$,
- $M \sim \mathcal{U}(5, 20)$,
- $Q \sim \mathcal{U}(5, 20)$.

Because of the multiplicity of solutions discussed in Section 3.3, we then excluded combinations with $P \leq \max(S, M+Q)$, yielding a total of 131 dimensionality combinations with a bottleneck ($N < \min(S, M+Q)$) and 170 without $N \geq \min(S, M+Q)$.

For each of these combinations, we then randomly generated the matrix of P samples of inputs X by sampling each element from the standard normal distribution ($\sigma_x = 1$). We likewise generated the matrices that produce the outputs, θ^*_{xy} and θ^*_{xz}, by sampling each element from the standard normal distribution, and then used these to generate the outputs, Y and Z, with standard normal noise added ($\sigma_y = \sigma_z = 1$, see (S3.1)). We also generated 2000 samples of validation data using the same θ^*_{xy}, θ^*_{xz}, and σ values.

In order to investigate how any results depend on the depth of the network, for each of these 301 dimensionality combinations and corresponding datasets, we consider three networks with one, two, or four hidden layers of equal dimension N. Note that this will allow

us to perform paired comparisons across depth.

We fit each network to its training data using (3.9) for 20000 epochs using `pytorch` [67] with a learning rate of $10^{-3}/(1+\beta)$. This learning rate scaling accounts for the combined weight of the response- and task-related terms in the joint cost function, and was also employed in our numerical analysis with the digits dataset (see Section 3.4.2). Every 100 epochs, we recorded the validation loss on the 2000 held out (x, y) samples and the effective network dimensionality. The latter is the effective rank [74] of the matrix product of all hidden layers of the network, i.e., the matrix that produces the representation in the deepest hidden layer from the input.

Results

As expected, we see that early on in training, there is no difference between networks with and without a dimensionality bottleneck in terms of improvement in validation loss (Figure 3.14A). However, a clear difference emerges by the end of training, where all networks that lack a bottleneck produce a very small relative improvement ($< 10^{-3}$), while only a few with the bottleneck produce such a low improvement (Figure 3.14B). These few are not unexpected, as there are other criteria that can lead to vanishing improvement (see (3.13), section 3.3.3). The temporal evolution of this separation for each network depth is shown in Figure 3.14C. The main feature to note is that the two cases do not separate much until after validation loss has reached a minimum (around 2000 epochs, see Figure 3.14D). This is important, since it implies that a reasonable early-stopping criteria might result in joint-training whether or not the network contains a dimensionality bottleneck.

However, there are other notable features of the time course, especially the negative overshoot just before 10000 epochs for depth 1 and 2 networks, and the greater final separation achieved by shallower networks. To better understand these features of the learning trajectory, we compare it to the evolution of $\hat{\beta}$, the value of β that is selected by joint-training to minimize the prediction loss of held out neural responses, (Figure 3.14E) and the effective network dimensionality (Figure 3.14F) at each epoch. We note that the timing of the overshoot in T_{JT} aligns well with variation in $\hat{\beta}_y$ and that the dependence of asymptotic T_{JT} on depth is

consistent with the asymptotic depth-dependence of network dimensionality as a fraction of the dimensionality of the data, $\min(S, M+Q)$. This correspondence is logical, since when the effective network dimensionality is much less than the data dimensionality, there is effectively a bottleneck in the network, whether it is imposed by the shape of the network or learning dynamics.

However, so far we have not addressed why the effective dimensionality or the estimated β might evolve in such ways, or why learning trajectory should relate to these measured quantities. In order to provide intuition for what is going on here, we delve deeper into a theoretical exploration of these relationships in the next Section.

3.5.3 Learning dynamics of joint-training

Here, we conduct a final analysis to help explain how the slow increase in network dimensionality and particular timecourse of the estimated hyperparameter $\hat{\beta}_y$ observed in joint-training may arise from the structure of the data, and why this allows for positive T_{JT} up until a certain point in training. To achieve this, we study the dynamics of learning across epochs. We focus on the eigenvalues of the cross-covariance matrix between the input X and the output R, where

$$R = \begin{bmatrix} Y \\ \sqrt{\beta} Z \end{bmatrix}. \tag{3.29}$$

We will seek a solution in terms of the cross-covariance matrices between the input and each individual task output (primary or auxiliary):

$$\begin{aligned} \Sigma_{YX} &= YX^\top = U_{YX} S_{YX} V_{YX}^\top \\ \Sigma_{ZX} &= ZX^\top = U_{ZX} S_{ZX} V_{ZX}^\top \end{aligned} \tag{3.30}$$

Both U and V matrices are unitary, and the S matrices are nonzero only on the diagonal elements; we refer to the ith element as $s_{yx,i}$ or $s_{zx,i}$.

We assume that the singular values of both matrices have been ordered such that the first N_s right singular vectors are equal ($[V_{YX}]_{1,1} = [V_{ZX}]_{1,1}; ...; [V_{YX}]_{S,N_s} = [V_{ZX}]_{S,N_s}$), and

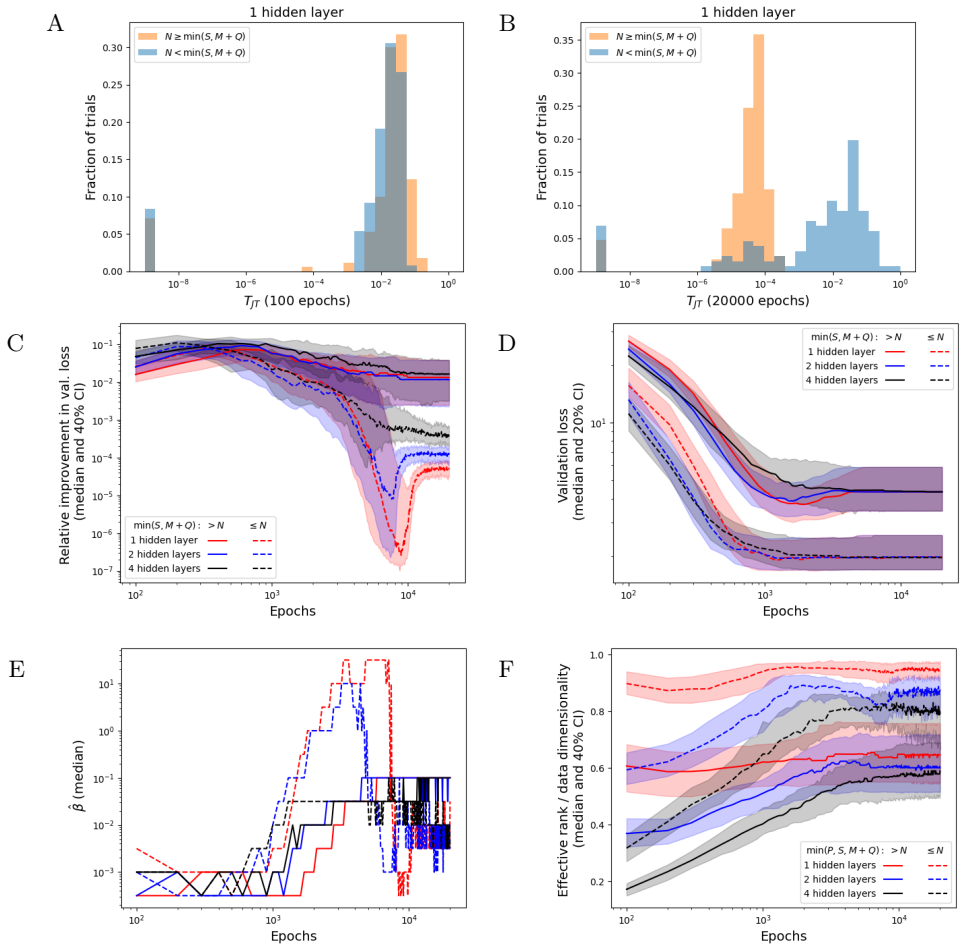


Figure 3.14: **Networks with no bottleneck yield no improvement, but only after excessive training.** A,B: Distribution of T_{JT} values achieved after 100 or 20000 epochs of learning, for networks with (blue) and without (orange) a dimensionality bottleneck. C: Separation in T_{JT} between the bottlenecked (solid) and unbottlenecked (dashed) networks emerges only after about 3000 epochs. D: Validation loss stops improving at around 2000 epochs. This means that a reasonable early stopping criterion would halt training before the bottleneck comes into play. E: Selected value of β during learning - note the complex shape that closely aligns with the time course of T_{JT}. F: Effective dimensionality of the network increases up to saturation over the course of training. For the unbottlenecked case, this is the dimensionality of the data; for the bottlenecked case it is the hidden layer size. Note that the deeper networks do not saturate as fully, corresponding to their higher T_{JT}. Panels B-F each display the median trace across 301 networks for each network depth and type, along with shading an interval between appropriate percentiles.

the remaining right singular vectors are all mutually orthogonal. Generally, we will use the s subscript to denote these "shared" dimensions of various matrices. Note that this represents a strict assumption on the relationship between the two covariance matrices. Because V_{YX} and V_{ZX} contribute M and Q singular vectors respectively that reside in an S-dimensional space, we can observe that $\max(0, M + Q - S) \leq N_s \leq \min(Q, M)$.

We are now ready to write and decompose the cross-covariance matrix learned by joint-training:

$$\begin{aligned}
\Sigma_{RX} &= RX^\top = \begin{bmatrix} YX^\top \\ \sqrt{\beta}ZX^\top \end{bmatrix} = \begin{bmatrix} \Sigma_{YX} \\ \sqrt{\beta}\Sigma_{ZX} \end{bmatrix} \\
\Sigma_{RX}\Sigma_{RX}^\top &= \begin{bmatrix} U_{YX}S_{YX}^2U_{YX}^\top & \sqrt{\beta}U_{YX}S_{YX}V_{YX}^\top V_{ZX}S_{ZX}U_{ZX}^\top \\ \sqrt{\beta}U_{ZX}S_{ZX}V_{ZX}^\top V_{YX}S_{YX}U_{YX}^\top & \beta U_{ZX}S_{ZX}^2U_{ZX}^\top \end{bmatrix} \\
&= \begin{bmatrix} U_{YX}S_{YX}^2U_{YX}^\top & \sqrt{\beta}U_{YXs}S_{YXs}S_{ZXs}U_{ZXs}^\top \\ \sqrt{\beta}U_{ZXs}S_{ZXs}S_{YXs}U_{YXs}^\top & \beta U_{ZX}S_{ZX}^2U_{ZX}^\top \end{bmatrix}
\end{aligned} \tag{3.31}$$

To find the eigenvalues of this matrix, we set

$$\begin{aligned}
0 &= det\left(\begin{bmatrix} U_{YX}(S_{YX}^2 - \lambda I)U_{YX}^\top & \sqrt{\beta}U_{YX}S_{YXs}S_{ZXs}U_{ZX}^\top \\ \sqrt{\beta}U_{ZX}S_{ZXs}S_{YXs}U_{YX}^\top & U_{ZX}(\beta S_{ZX}^2 - \lambda I)U_{ZX}^\top \end{bmatrix}\right) \\
&= det(U_{YX}(S_{YX}^2 - \lambda I)U_{YX}^\top) \times \\
&\quad det(U_{ZX}(\beta S_{ZX}^2 - \lambda I)U_{ZX}^\top - \beta U_{ZX}S_{ZXs}S_{YXs}U_{YX}^\top(U_{YX}(S_{YX}^2 - \lambda I)U_{YX}^\top)^{-1}U_{YX}S_{YXs}S_{ZXs}U_{ZX}^\top) \\
&= det(S_{YX}^2 - \lambda I)det((\beta S_{ZX}^2 - \lambda I) - \beta S_{ZXs}S_{YXs}U_{YX}^\top U_{YX}(S_{YX}^2 - \lambda I)^{-1}U_{YX}^\top U_{YX}S_{YXs}S_{ZXs})] \\
&= det(S_{YX}^2 - \lambda I)det(\beta S_{ZX}^2 - \lambda I - \beta S_{ZXs}^2 S_{YXs}^2(S_{YX}^2 - \lambda I)^{-1})]
\end{aligned} \tag{3.32}$$

The first step makes use of the relationship for block matrices $det\left(\begin{bmatrix} A & B \\ C & D \end{bmatrix}\right) = det(A)det(D - CA^{-1}B)$ (or alternatively $det(D)det(A - BD^{-1}C)$). The argument of the second determinant in (3.32) is a diagonal matrix. The ith element has a value of $\beta s_{zx,i}^2(1 - \frac{s_{yx,i}^2}{s_{yx,i}^2 - \lambda}) - \lambda$ if $i \leq N_s$, and $\beta s_{zx,i}^2 - \lambda$ if $i > N_s$. The determinant will be zero if

any diagonal element is zero. For any $i > N_s$, this happens if $\lambda = \beta s_{zx,i}^2$. For any $i \leq N_s$, this happens if

$$\begin{aligned} 0 &= \beta s_{zx,i}^2 \left(1 - \frac{s_{yx,i}^2}{s_{yx,i}^2 - \lambda}\right) - \lambda \\ 0 &= \beta s_{zx,i}^2((s_{yx,i}^2 - \lambda) - s_{yx,i}^2) - \lambda(s_{yx,i}^2 - \lambda) \\ 0 &= \lambda^2 - (s_{yx,i}^2 + \beta s_{zx,i}^2)\lambda \\ \lambda &= s_{yx,i}^2 + \beta s_{zx,i}^2 \text{ or } 0 \end{aligned} \tag{3.33}$$

By using the alternative determinant simplification in (3.32), we can similarly show that $\lambda = s_{yx,i}^2$ for $i > N_s$ is also a solution.

From here, we will use the substitution $\beta = \frac{\alpha}{1-\alpha}$, since this incorporates the learning rate scaling we use when optimizing with a specific value of β (see Section 3.4.2). Each singular value of Σ_{RX} is just the square root of the corresponding eigenvalue of $\Sigma_{RX}\Sigma_{RX}^\top$, so we have four possible forms for the singular values:

- $\sqrt{(1-\alpha)s_{yx,i}^2 + \alpha s_{zx,i}^2}$
- $s_{yx,i}\sqrt{1-\alpha}$
- $s_{zx,i}\sqrt{\alpha}$
- 0

To understand how a learning trajectory might arise in joint-training, we make use of the main result in work by Saxe, McClelland, and Ganguli [76], that a multi-layer network learns each singular mode of the input-output cross-covariance matrix separately, after a duration of training inversely proportional to the corresponding singular value. Additionally, Saxe, McClelland, and Ganguli found that training happens more quickly in deeper networks, until the finite step size used by the optimizer becomes too big to support approximating learning as continuous. This detail may explain the differences in learning trajectories we observed between networks of different depths (between red, blue, and black curves in Figure 3.14). Moreover Saxe, McClelland, and Ganguli show that, the time course of learning for a single

eigenmode is sigmoidal, with the learned strength of a mode growing from near zero and approaching its corresponding singular value. This means that the model will finish learning larger modes while barely starting to learn smaller ones.

In our setup, there is a range of mode strengths that can be attained after any given amount of training, controlled by the hyperparameter β (equivalently, α). The training set of (x, y, z) data determines the singular vectors of the cross-covariance matrix and the range of corresponding singular values possible. The validation set of (x, y) data determines, for each of these singular vectors, what the optimal singular value is for minimizing the loss of predicting held-out y. At any given point in training, an α is selected that minimizes the aggregated differences between these optimal singular values and the learned mode strengths.

Figure 3.15 depicts how a joint-training learning trajectory would be shaped by an example landscape of the singular values that are optimal and possible as a function of α. Because learning time is inversely proportional to singular value, we can consider a learning trajectory by moving downwards through this landscape, adjusting α as we go to minimize the loss of predicting held out neural data.

The variation in α over time that results from such a landscape not only can recapitulate corresponding trends observed in simulations (Figure 3.14E), but also results for variation in validation loss (Figure 3.14C,D) and network dimensionality (Figure 3.14F) over time. Validation loss, as mentioned, is related to the differences between the estimated and optimal mode strengths, aggregated across all modes. This will generally improve as modes are learned (Figure 3.15A-E), until the network is forced to learn (or avoid learning) modes that it would be better off ignoring (G-H). The latter phase is especially common when $N \geq \min(S, M + Q)$, as in this case all singular vectors of the input-output cross-covariance are learned (including ones like the green mode in the example), whereas a smaller N only allows the first N to be learned.

Network dimensionality for a given value of α at a particular time roughly corresponds to the number of singular value curves above that point. This quantity is guaranteed to increase for a given α, and will generally do so along the joint-training trajectory, although temporary decreases are possible, as changing α can move the trajectory above a curve that it was previously below, without crossing below another curve (e.g. to "avoid" another curve, like

the green one in Figure 3.15G). The exception is that at the beginning of training, measured dimensionality will be high, since all modes have similarly low strength. When the first mode is learned, the dimensionality will decrease to 1. If the trajectory is below N singular value curves, it cannot learn any more modes and training is over.

It is worth noting that the assumption about the relationship between V_{YX} and V_{ZX} in general will not hold. However, even with this strict assumption, many learning trajectories are possible for different singular value landscapes. For the more general case, the singular vectors would change as well as the singular values, and the modes from each task would not be paired off so nicely, but one could draw a similar picture to Figure 3.15. The curves could take on more varied shapes due to interactions between many more modes, and the optimal point on the curve would represent an optimal singular value as well as singular vector. Aggregating differences in estimated mode strengths would also be more complicated, due to the changing singular vectors. Despite all this, the same general principles still apply, and the resulting trajectories should be largely similar to those in our simplified case.

In this Section, we have used theoretical perspectives on training linear networks to explain the detailed learning trajectories we observed in Figure 3.5.2, demonstrating how finite training time can effectively introduce a bottleneck into a model. This, combined with our observations in Section 3.5.1 regarding how nonlinearities and constraints on model parameters may also introduce a bottleneck, provides a thorough qualification of the main idea we derived for a dimensionality bottleneck from our linear model in Section 3.3.1.

3.6 Discussion

In this work, we analyzed joint-training of a neural circuit model as an intermediate approach between response-training and task-training that treats a hypothesized computation as an auxiliary task. We analytically derived conditions under which joint-training improves predictions of held out neural activity and showed that maximizing this improvement selects computations that are most predictable from the neural data. We performed these derivations for networks with both maximal and minimal sparsity of the units-to-neurons map, and also verified their applicability to a nonlinear CNN architecture, under a range of assumptions about the sparsity of the units-to-neurons map. These numerical experiments also allowed

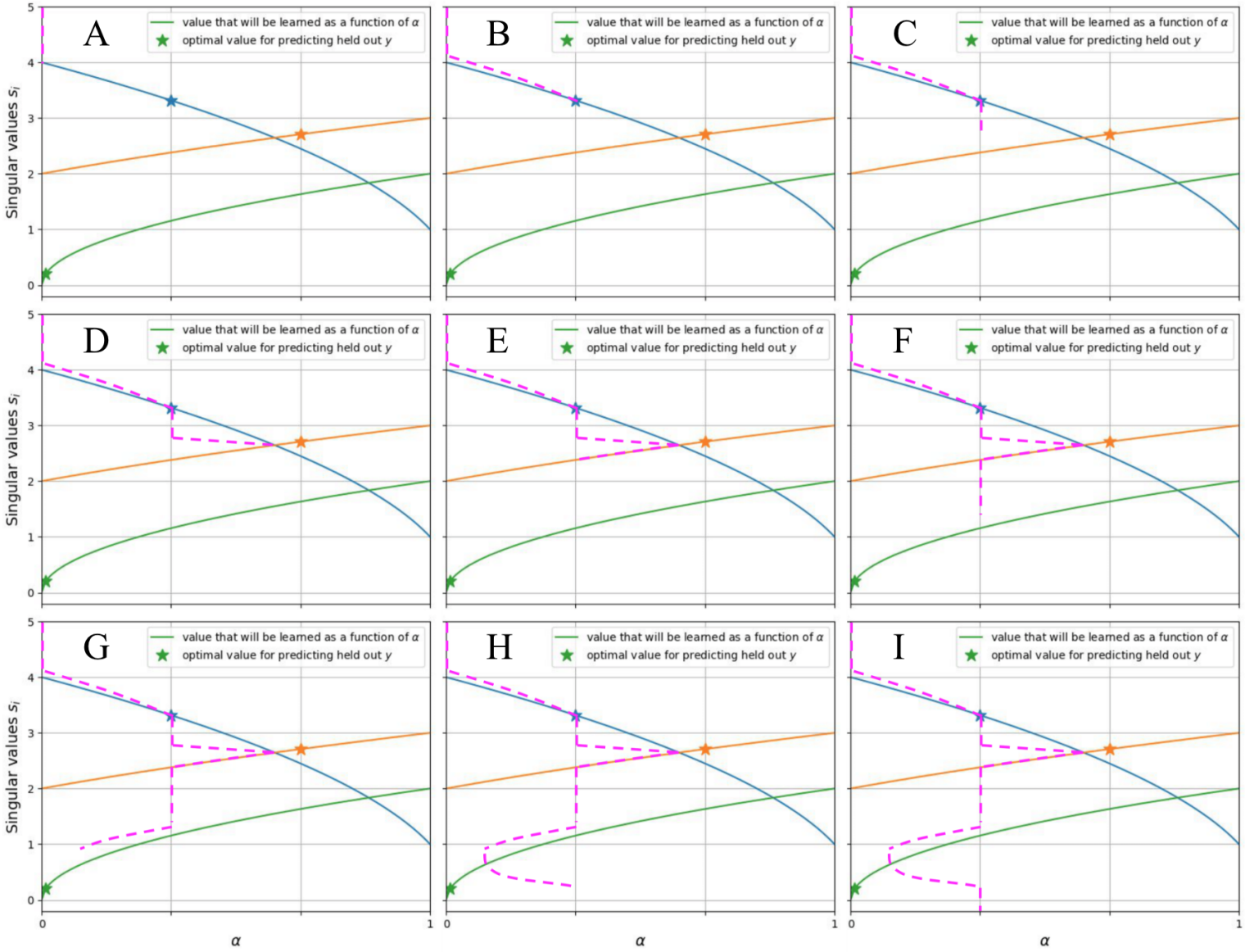

A
B
C
D
E
F
G
H
I
value that will be learned as a function of α
optimal value for predicting held out y
Singular values s_i
α
0
1
2
3
4
5

Figure 3.15: **A hypothetical learning trajectory of joint-training suggests mechanisms that produce the results in Figure 3.14.** Each panel shows the state of the joint-trained model (dashed magenta line) on a landscape of the eigenvalues (y-axis; blue, orange, and green correspond with three eigenmodes) of the cross-covariance matrix between the input X and combined output $\begin{bmatrix} Y \\ \sqrt{\beta}Z \end{bmatrix}$. The value of the eigenvalue as a function of $\alpha = \frac{\beta}{1+\beta}$ (x-axis) is the solid line; the value that corresponds to the best predictor of held out neural data is marked by a star. Joint-training consists of moving downward through this landscape (since the speed at which an eigenmode is learned is proportional to its eigenvalue), adjusting α to the value that best predicts held out neural data by minimizing the aggregated differences between each optimal eigenmode strength (height of each star) and its learned value (0 if above the corresponding solid curve, otherwise the value of that curve at that α). A: Initially, $\alpha = 0$, because the blue eigenmode is learned fastest. B: Once the blue eigenmode has been learned to the correct degree, α increases to keep the learned strength as close as possible to optimal. C: Once the blue eigenmode has been fully learned at a value of α that yields the optimal mode strength, α remains fixed. The orange eigenmode does not yet have an influence because to increase α to a value where it has been learned would have a worse impact on the blue eigenmode. D: Now α accommodates the orange eigenmode, as doing so no longer requires switching to an α where the blue one is not yet learned (i.e where the orange curve is above the blue). E: As the orange eigenmode becomes learned for lower values of α, α decreases accordingly improving the blue eigenmode strength's accuracy more than it hurts the orange one's. F: Having achieved the best balance of the blue and orange eigenmodes, α remains constant. G: In order to avoid a sharp increase in the green eigenmode strength, α decreases to a value where its learned strength is still 0 (above the green curve), near the optimal value (the height of the green star). H: When continuing to decrease α would do more harm to blue and orange combined than help to green, it gives in to learning green and increases to better accommodate blue and orange. I: We have reached the end of training.

us to show that joint-training maps recorded neurons to model units as accurately and (almost) as precisely as possible, allowing for easier interpretation of the model in terms of how specific recorded neurons might contribute to a computation. We demonstrated that improved neural data prediction requires a bottleneck in the network, whose presence depends on nonlinearities and constraints on model parameters imposed by the architecture, regularization terms such as those that constrain the units-to-neurons map, and especially early stopping.

A central result from our analysis is that tasks which are “compatible" with the neural data maximally improve neural response prediction, and we have provided a quantification of the expected improvement. Joint-training and task-training both prefer such tasks, as well as those with relatively low output noise. However, joint-training will tend to select higher-dimensional auxiliary tasks, while task-training will select lower-dimensional ones. While it may be that a lower-dimensional auxiliary task facilitates interpretation, finding one is better achieved by appropriately restricting the space of tasks to search.

In Section 3.4.4, we explored how the application of joint-training to data can lead to interesting questions and answers about the relationships between computations, a domain now known as “taskology” [56]. Indeed, using tasks defined by datasets of paired inputs and outputs to understand neural activity will require understanding the candidate tasks and how they are related to each other. Many studies have shown how using task datasets that are biased or problematic in their selection of samples, labels, or pairings of the two, can lead to undesirable results in models trained on them [19, 33, 82, 92]. Such problems could affect the sort of analysis we carry out here, and, generally, a better understanding of tasks would allow for them to be better used as tools in models of neural activity.

Naturally, one would rely on the intuition of a domain expert in choosing candidate tasks for a given neural dataset, such as the guidance that visually responsive areas of cortex encode information about visual features like contours, frequencies, or object properties. Task-training with these kinds of features has led to extremely high performing neural response models [88]. Alternatively, the conditions we derive can be used in principle as the basis for a differentiable search strategy over a limited space of tasks, potentially providing a means for machine-learning-based identification of computations.

Either way, the researcher requires some amount of intuition in proposing a bounded set of tasks to evaluate that might be related to the function of a particular neural circuit. Indeed, one could level this criticism at regularization in general, especially noting that the results of many neural network studies, including many response-trained models, depend critically on a precisely mixed regularization cocktail, with many different terms each weighed by its own hyperparameter [77]. In contrast, joint-training treats the computation like one regularization term, weighed by a single hyperparameter β. Regularization on the map between neurons and units may be desirable to achieve a more interpretable mapping, as we have shown that it is capable of doing, but this is not necessary for the other improvements in performance. However, all the assumptions that affect model definition regarding any regularization, the architecture, nonlinearities, and even details like early stopping may have an impact on the improvements afforded by joint-training. We have argued that joint-training can only provide any improvement at all when some combination of these factors create a bottleneck in the flow of information through the network. This idea of creating a bottleneck can thus serve as a model design principle for those seeking to use joint-training to improve predictions of neural responses. Joint-training requires less fiddling with assumptions of the researcher to build a useful model, compared to response-training, while providing them greater intuition in doing so.

That being said, there are still numerous other details that the researcher must attend to when applying joint-training, and it is to be expected these challenges should only be greater when using real neural data. Generally, these details are common to all research that uses neural networks: weight initialization, learning rate scheduling, and, especially relevant here, architecture selection. However, the gradient descent performed in joint-training is really just multi-task learning, which is equivalent to regular gradient descent with a specific form of output. This affords joint-training easy access to all existing and future methods for automating this fiddling. For example, one could use joint-training with an architecture learning method [7, 39]. While one could use such an approach with task-training, one would expect that the learned architecture would not resemble true neural structure as closely. Additionally, the framing of the problem as auxiliary task learning also allows for the use of any tools specific to that domain (e.g. [83]) or multi-task learning in general.

Another problem that may arise when applying this method to real world data is that instead of data samples consisting of (x_i, y_i, z_i) triplets, as we assume here, one may have separate neural samples (x_i, y_i) and task samples (x_i, z_i). Indeed this may be desirable, since it allows for an auxiliary task with many more samples than is often possible with neural data collection, potentially allowing for a greater benefit to neural data predictions. This does not affect the training procedure, except that the sum in (3.4) must be split in two, and one may want to rescale the individual cost terms to account for any difference in the number of samples. Any differences between the distributions over the x_i associated with neural data and task would also effect our results, especially when nonlinearities are present. In this case, neural data prediction would be improved when there is alignment between task and neural data in the domain of the (x_i, z_i) task samples (although without data triplets it is not easy to define exactly what is meant by "alignment").

In sum, have demonstrated a theoretical foundation for understanding the performance of joint-training and a concrete example of its general utility with complex nonlinear networks, both under a range of assumptions about the precision of the units-to-neurons map. These results promise to support the development of improved models of neural responses that connect neural activities to principles of computation.

3.7 Chapter Acknowledgements

This work was supported by the Laboratory Directed Research and Development program at Sandia National Laboratories, a multimission laboratory managed and operated by National Technology and Engineering Solutions of Sandia LLC, a wholly owned subsidiary of Honeywell International Inc. for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525. This paper describes objective technical results and analysis. Any subjective views or opinions that might be expressed in the paper do not necessarily represent the views of the U.S. Department of Energy or the United States Government. We thank the Allen Institute founder, Paul G. Allen, for his vision, encouragement, and support.

Chapter S3

ADDITIONAL ANALYSES FOR USING JOINT-TRAINING ON NEURAL DATA AND AUXILIARY TASKS TO IMPROVE MODELS OF NEURAL CIRCUITS

Code Availability All code used in this study is available at `https://github.com/zdeblick/JointTraining`.

S3.1 Derivations and Numerical Verification of Results in Section 3.3

Here we provide the analytical derivations for joint-training that lead up to the main results for the general (3.13) and simplified (3.15) linear models, and numerical verification of the accuracy of the approximations involved in producing the latter. In Section S3.1.1, we derive the parameters obtained by minimizing the loss function for the general model (3.9). In Section S3.1.2, we derive the expected validation loss of predicting neural responses. We use this first to derive the main result for the general model, (3.13), and then to use the simplified model to permit approximations that allow us to derive (3.15).

Notation: Recall from Section 3.3 that our model consists of an S dimensional stimulus, x_i, fed into an N dimensional hidden layer whose unit activities we call u_i, which is used to predict the responses of M neurons, y_i (main task) and a Q dimensional auxiliary task output z_i. The learned parameters of these three linear operations are θ_u, θ_y, and θ_z, respectively, and we assume that the data comes from a true distribution specified by

$$\begin{aligned} x_i &\sim \mathcal{N}(0, \sigma_x^2 I), \\ y_i &= \theta_{xy}^* x_i + \xi_{y,i}, \\ z_i &= \theta_{xz}^* x_i + \xi_{z,i}, \end{aligned} \tag{S3.1}$$

where $\xi_{y,i} \sim \mathcal{N}(0, \sigma_y^2 I)$ and $\xi_{z,i} \sim \mathcal{N}(0, \sigma_z^2 I)$ are independent Gaussian random variables. A diagram of this process may be found in Figure 3.4B, and contrasted with the model (Figure

3.3C).

Here, we introduce the term $Q^* = dim(C(\theta^*_{xz}) \cap C(\theta^*_{xy}))$ as the dimension of the intersection of the column spaces for the matrices that generate the data z_i and y_i, respectively, from x_i ($dim()$ denotes dimension of a subspace, $C()$ denotes the column space of a matrix). If θ^*_{xz} and θ^*_{xy} are compatible, $Q^* = Q$, otherwise $Q^* < Q$. We thus call Q^* the "compatible dimensionality." We also make use of the matrices $\Xi_y \equiv [\xi_{y,1}, ..., \xi_{y,P}]$, and $\Xi_z \equiv [\xi_{z,1}, ..., \xi_{z,P}]$.

S3.1.1 Optimizing Network Parameters

We can find the MLE for θ_u by taking the derivative of the joint cost function (3.9) w.r.t. θ_u and setting it equal to zero:

$$\frac{dC}{d\theta_u} = 0 \quad = \quad 2\sum_{i=1}^{P} \theta_y^\top \theta_y \theta_u x_i x_i^\top - \theta_y^\top y_i x_i^\top + \beta[\theta_z^\top \theta_z \theta_u x_i x_i^\top - \theta_z^\top z_i x_i^\top]. \tag{S3.2}$$

Using matrix notation and dividing by 2, this can be rewritten as

$$0 \quad = \quad \theta_y^\top \theta_y \theta_u X X^\top - \theta_y^\top Y X^\top + \beta[\theta_z^\top \theta_z \theta_u X X^\top - \theta_z^\top Z X^\top],$$

which can be solved with

$$\hat{\theta}_u \quad = \quad (\theta_y^\top \theta_y + \beta \theta_z^\top \theta_z)^{-1} (\theta_y^\top Y X^\top + \beta \theta_z^\top Z X^\top)(X X^\top)^{-1}. \tag{S3.3}$$

Note that this final step relies on the assumptions $P \geq S$ (to invert $XX^\top$; otherwise the solution would require the pseudoinverse), which we have already made, and $M + Q \geq N$ (to invert $\theta_y^\top \theta_y + \beta \theta_z^\top \theta_z$), which we have already identified as a necessary condition for (3.9) to have a distinct solution from (3.10).

Instead of finding the MLE for θ_y and θ_z, in our approximate analysis we simply assume that they are fixed matrices $\overline{\theta}_y$ and $\overline{\theta}_z$. When computing a numerical value for T_{JT} or T_{TT}, as we will in Section S3.1.3, we specifically assume that $\overline{\theta}_y = I_M$ (assuming the number of artificial units, N, equals that of recorded neurons M, and the model is configured such that each unique neuron is associated with a unique unit) and $\overline{\theta}_z \equiv \theta^*_{xz} \theta^{*+}_{xy}$, which is the MLE as

$P \to \infty$ (superscript + denotes the Moore-Penrose pseudoinverse). This approximation is nontrivial, so we derive in Section S3.2 exactly what terms are left out of this approximation, and numerically assess the aggregate effect of all approximations in Section S3.1.3. This specific form of $\overline{\theta}_y$ and $\overline{\theta}_z$ is only valid in our simplified model where $N = M$, but this is the only time we use such a specification to directly compare analytically and numerically derived quantities. We can plug these $\overline{\theta}_y$ and $\overline{\theta}_z$, along with the assumed structure of our data, (S3.1), into (S3.3), obtaining

$$\begin{aligned}\hat{\theta}_u &= (\overline{\theta}_y^\top \overline{\theta}_y + \beta \overline{\theta}_z^\top \overline{\theta}_z)^{-1}(\overline{\theta}_y^\top(\theta^*_{xy} X X^\top + \Xi_y X^\top) + \beta \overline{\theta}_z^\top(\theta^*_{xz} X X^\top + \Xi_z X^\top))(X X^\top)^{-1} \\ &= (\overline{\theta}_y^\top \overline{\theta}_y + \beta \overline{\theta}_z^\top \overline{\theta}_z)^{-1}(\overline{\theta}_y^\top(\theta^*_{xy} + \Xi_y X^+) + \beta \overline{\theta}_z^\top(\theta^*_{xz} + \Xi_z X^+)). \qquad \text{(S3.4)}\end{aligned}$$

S3.1.2 Selecting Optimal Hyperparameters

Given a particular $\hat{\theta}_u$, we can then compute the expected neural response loss on given validation data, $C^{val}_{resp}(\beta)$, by taking the expectation over held-out samples, (x_i, y_i, z_i) for $i \in D_{val}$ (denoted $\mathbb{E}_{i \in D_{val}}$ for simplicity). However, $\hat{\theta}_u$ is also a random variable, as its value depends on the training data, and so we also take the expectation with respect to it (denoted $\mathbb{E}_{\hat{\theta}_u}$, or simply $\mathbb{E}$ when there is no ambiguity). In sum:

$$C^{val}_{resp}(\beta) = \mathbb{E}_{i \in D_{val}, \hat{\theta}_u} \|y_i - \overline{\theta}_y \hat{\theta}_u x_i\|_F^2. \qquad \text{(S3.5)}$$

Recall that we are approximating θ_y as a fixed matrix $\overline{\theta}_y$, which is not learned in training, and thus we do not take the expectation with respect to it. We can plug in our assumption for $y_i = \theta^*_{xy} x_i + \xi_{y,i}$ (S3.1), and reorganize these terms to obtain

$$\begin{aligned}C^{val}_{resp}(\beta) &= \mathbb{E}_{i \in D_{val}, \hat{\theta}_u}[x_i^\top(\theta^*_{xy} - \overline{\theta}_y \hat{\theta}_u)^\top(\theta^*_{xy} - \overline{\theta}_y \hat{\theta}_u) x_i + \xi_{y,i}^\top \xi_{y,i}] \\ &= \mathbb{E}_{i \in D_{val}}[x_i^\top \mathbb{E}_{\hat{\theta}_u}[\overline{\theta}^{*\top}_{xy} \theta^*_{xy} - \hat{\theta}_u^\top \overline{\theta}_y^\top \theta^*_{xy} - \overline{\theta}^{*\top}_{xy} \overline{\theta}_y \hat{\theta}_u + \hat{\theta}_u^\top \overline{\theta}_y^\top \overline{\theta}_y \hat{\theta}_u] x_i + \xi_{y,i}^\top \xi_{y,i}] \\ &= \mathbb{E}_{i \in D_{val}}[x_i^\top(\overline{\theta}^{*\top}_{xy} \theta^*_{xy} - \mathbb{E}[\hat{\theta}_u]^\top \overline{\theta}_y^\top \theta^*_{xy} - \overline{\theta}^{*\top}_{xy} \overline{\theta}_y \mathbb{E}[\hat{\theta}_u] + \mathbb{E}[\hat{\theta}_u^\top \overline{\theta}_y^\top \overline{\theta}_y \hat{\theta}_u]) x_i] + M\sigma_y^2 \\ &= \sigma_x^2 tr[\overline{\theta}^{*\top}_{xy} \theta^*_{xy} - \mathbb{E}[\hat{\theta}_u]^\top \overline{\theta}_y^\top \theta^*_{xy} - \overline{\theta}^{*\top}_{xy} \overline{\theta}_y \mathbb{E}[\hat{\theta}_u] + \mathbb{E}[\hat{\theta}_u^\top \overline{\theta}_y^\top \overline{\theta}_y \hat{\theta}_u]] + M\sigma_y^2. \qquad \text{(S3.6)}\end{aligned}$$

The last step makes use of the relationship $\mathbb{E}[x^\top Ax] = tr[A\text{Var}(x)] + \mathbb{E}(x)^\top A\mathbb{E}(x)$ (equation 328 in [69]). To evaluate this expression, we need only to solve for $\mathbb{E}[\hat{\theta}_u]$ and $\mathbb{E}[\hat{\theta}_u^\top \overline{\theta}_y^\top \overline{\theta}_y \hat{\theta}_u]$. We can easily solve for the first by noting that $\mathbb{E}[X^+] = 0$:

$$\overline{\theta}_u \equiv \mathbb{E}[\hat{\theta}_u] = (\overline{\theta}_y^\top \overline{\theta}_y + \beta \overline{\theta}_z^\top \overline{\theta}_z)^{-1}(\overline{\theta}_y^\top \theta^*_{xy} + \beta \overline{\theta}_z^\top \theta^*_{xz}). \tag{S3.7}$$

The second however, takes a little work:

$$\begin{aligned}
\mathbb{E}[\hat{\theta}_u^\top \overline{\theta}_y^\top \overline{\theta}_y \hat{\theta}_u] &= \mathbb{E}[R^\top (\overline{\theta}_y^\top \overline{\theta}_y + \beta \overline{\theta}_z^\top \overline{\theta}_z)^{-1} \overline{\theta}_y^\top \overline{\theta}_y (\overline{\theta}_y^\top \overline{\theta}_y + \beta \overline{\theta}_z^\top \overline{\theta}_z)^{-1} R], \\
R &\equiv (\overline{\theta}_y^\top (\theta^*_{xy} + \Xi_y X^+) + \beta \overline{\theta}_z^\top (\theta^*_{xz} + \Xi_z X^+))
\end{aligned}$$

Approximately, $R \sim \mathcal{MN}(\overline{\theta}_y^\top \theta^*_{xy} + \beta \overline{\theta}_z^\top \theta^*_{xz}, \sigma_y^2 \overline{\theta}_y^\top \overline{\theta}_y + \beta^2 \sigma_z^2 \overline{\theta}_z^\top \overline{\theta}_z, \frac{1}{(P-S-1)\sigma_x^2} I)$, where $\mathcal{MN}$ denotes the matrix normal distribution [61]. This reduction assumes P is large enough for the central limit theorem to apply, and uses properties of the pseudoinverse of a matrix with zero-mean i.i.d. Gaussian entries, namely that it has independent elements with mean $\mathbb{E}[X^+_{ij}] = 0$ and variance $\mathbb{E}[(X^+_{ij})^2] = \frac{1}{P(P-S-1)\sigma_x^2}$ (see Section S3.2). We can then say

$$\begin{aligned}
\mathbb{E}[\hat{\theta}_u^\top \overline{\theta}_y^\top \overline{\theta}_y \hat{\theta}_u] &= (\overline{\theta}_y^\top \theta^*_{xy} + \beta \overline{\theta}_z^\top \theta^*_{xz})^\top (\overline{\theta}_y^\top \overline{\theta}_y + \beta \overline{\theta}_z^\top \overline{\theta}_z)^{-1} \overline{\theta}_y^\top \overline{\theta}_y (\overline{\theta}_y^\top \overline{\theta}_y + \beta \overline{\theta}_z^\top \overline{\theta}_z)^{-1} (\overline{\theta}_y^\top \theta^*_{xy} + \beta \overline{\theta}_z^\top \theta^*_{xz}) \\
&\quad + \frac{1}{(P-S-1)\sigma_x^2} tr[(\sigma_y^2 \overline{\theta}_y^\top \overline{\theta}_y + \beta^2 \sigma_z^2 \overline{\theta}_z^\top \overline{\theta}_z)(\overline{\theta}_y^\top \overline{\theta}_y + \beta \overline{\theta}_z^\top \overline{\theta}_z)^{-1} \overline{\theta}_y^\top \overline{\theta}_y (\overline{\theta}_y^\top \overline{\theta}_y + \beta \overline{\theta}_z^\top \overline{\theta}_z)^{-1}] I. \\
&= \overline{\theta}_u^\top \overline{\theta}_y^\top \overline{\theta}_y \overline{\theta}_u + \frac{1}{(P-S-1)\sigma_x^2} tr[(\sigma_y^2 \overline{\theta}_y^\top \overline{\theta}_y + \beta^2 \sigma_z^2 \overline{\theta}_z^\top \overline{\theta}_z)(\overline{\theta}_y^\top \overline{\theta}_y + \beta \overline{\theta}_z^\top \overline{\theta}_z)^{-1} \overline{\theta}_y^\top \overline{\theta}_y (\overline{\theta}_y^\top \overline{\theta}_y + \beta \overline{\theta}_z^\top \overline{\theta}_z)^{-1}] I.
\end{aligned} \tag{S3.8}$$

Plugging (S3.7) and (S3.8) back into (S3.6), we get

$$\begin{aligned}
C^{val}_{resp}(\beta) &= \sigma_x^2 tr\big[\overline{\theta}^{*\top}_{xy} \theta^*_{xy} - \overline{\theta}_u^\top \overline{\theta}_y^\top \theta^*_{xy} - \theta^{*\top}_{xy} \overline{\theta}_y \overline{\theta}_u + \overline{\theta}_u^\top \overline{\theta}_y^\top \overline{\theta}_y \overline{\theta}_u \\
&\quad + \frac{1}{(P-S-1)\sigma_x^2} tr[(\sigma_y^2 \overline{\theta}_y^\top \overline{\theta}_y + \beta^2 \sigma_z^2 \overline{\theta}_z^\top \overline{\theta}_z)(\overline{\theta}_y^\top \overline{\theta}_y + \beta \overline{\theta}_z^\top \overline{\theta}_z)^{-1} \overline{\theta}_y^\top \overline{\theta}_y (\overline{\theta}_y^\top \overline{\theta}_y + \beta \overline{\theta}_z^\top \overline{\theta}_z)^{-1}] I\big] + M\sigma_y^2 \\
&= \sigma_x^2 tr[(\theta^*_{xy} - \overline{\theta}_y \overline{\theta}_u)^\top (\theta^*_{xy} - \overline{\theta}_y \overline{\theta}_u)] \\
&\quad + \frac{S}{P-S-1} tr[(\sigma_y^2 \overline{\theta}_y^\top \overline{\theta}_y + \beta^2 \sigma_z^2 \overline{\theta}_z^\top \overline{\theta}_z)(\overline{\theta}_y^\top \overline{\theta}_y + \beta \overline{\theta}_z^\top \overline{\theta}_z)^{-1} \overline{\theta}_y^\top \overline{\theta}_y (\overline{\theta}_y^\top \overline{\theta}_y + \beta \overline{\theta}_z^\top \overline{\theta}_z)^{-1}] + M\sigma_y^2.
\end{aligned} \tag{S3.9}$$

Deriving the slope in validation loss at $\beta = 0$ We can now compute the change in this validation loss with respect to β by moving the derivative inside the trace and applying the chain rule, plugging in for $\overline{\theta}_u$ where necessary:

$$
\begin{aligned}
&\frac{d}{d\beta}C^{val}_{resp}(\beta) = \\
&2\sigma_x^2 tr\left[\left(\overline{\theta}_y\frac{d}{d\beta}[(\overline{\theta}_y^\top\overline{\theta}_y+\beta\overline{\theta}_z^\top\overline{\theta}_z)^{-1}](\overline{\theta}_y^\top\theta^*_{xy}+\beta\overline{\theta}_z^\top\theta^*_{xz})+\overline{\theta}_y(\overline{\theta}_y^\top\overline{\theta}_y+\beta\overline{\theta}_z^\top\overline{\theta}_z)^{-1}\frac{d}{d\beta}[\overline{\theta}_y^\top\theta^*_{xy}+\beta\overline{\theta}_z^\top\theta^*_{xz}]\right)^\top(\overline{\theta}_y\overline{\theta}_u-\theta^*_{xy})\right]+ \\
&\frac{S}{P-S-1}tr\Bigg[\frac{d}{d\beta}[\sigma_y^2\overline{\theta}_y^\top\overline{\theta}_y+\beta^2\sigma_z^2\overline{\theta}_z^\top\overline{\theta}_z](\overline{\theta}_y^\top\overline{\theta}_y+\beta\overline{\theta}_z^\top\overline{\theta}_z)^{-1}\overline{\theta}_y^\top\overline{\theta}_y(\overline{\theta}_y^\top\overline{\theta}_y+\beta\overline{\theta}_z^\top\overline{\theta}_z)^{-1} \\
&+(\sigma_y^2\overline{\theta}_y^\top\overline{\theta}_y+\beta^2\sigma_z^2\overline{\theta}_z^\top\overline{\theta}_z)\frac{d}{d\beta}[(\overline{\theta}_y^\top\overline{\theta}_y+\beta\overline{\theta}_z^\top\overline{\theta}_z)^{-1}]\overline{\theta}_y^\top\overline{\theta}_y(\overline{\theta}_y^\top\overline{\theta}_y+\beta\overline{\theta}_z^\top\overline{\theta}_z)^{-1} \\
&+(\sigma_y^2\overline{\theta}_y^\top\overline{\theta}_y+\beta^2\sigma_z^2\overline{\theta}_z^\top\overline{\theta}_z)(\overline{\theta}_y^\top\overline{\theta}_y+\beta\overline{\theta}_z^\top\overline{\theta}_z)^{-1}\overline{\theta}_y^\top\overline{\theta}_y\frac{d}{d\beta}[(\overline{\theta}_y^\top\overline{\theta}_y+\beta\overline{\theta}_z^\top\overline{\theta}_z)^{-1}]\Bigg], \\
=\;& 2\sigma_x^2 tr\Bigg[\Bigg(-\overline{\theta}_y(\overline{\theta}_y^\top\overline{\theta}_y+\beta\overline{\theta}_z^\top\overline{\theta}_z)^{-1}\overline{\theta}_z^\top\overline{\theta}_z\underbrace{(\overline{\theta}_y^\top\overline{\theta}_y+\beta\overline{\theta}_z^\top\overline{\theta}_z)^{-1}(\overline{\theta}_y^\top\theta^*_{xy}+\beta\overline{\theta}_z^\top\theta^*_{xz})}_{\overline{\theta}_u}+\overline{\theta}_y(\overline{\theta}_y^\top\overline{\theta}_y+\beta\overline{\theta}_z^\top\overline{\theta}_z)^{-1}\overline{\theta}_z^\top\theta^*_{xz}\Bigg)^\top(\overline{\theta}_y\overline{\theta}_u-\theta^*_{xy})\Bigg]+ \\
&\frac{S}{P-S-1}tr\Bigg[2\beta\sigma_z^2\overline{\theta}_z^\top\overline{\theta}_z(\overline{\theta}_y^\top\overline{\theta}_y+\beta\overline{\theta}_z^\top\overline{\theta}_z)^{-1}\overline{\theta}_y^\top\overline{\theta}_y(\overline{\theta}_y^\top\overline{\theta}_y+\beta\overline{\theta}_z^\top\overline{\theta}_z)^{-1} \\
&-(\sigma_y^2\overline{\theta}_y^\top\overline{\theta}_y+\beta^2\sigma_z^2\overline{\theta}_z^\top\overline{\theta}_z)[(\overline{\theta}_y^\top\overline{\theta}_y+\beta\overline{\theta}_z^\top\overline{\theta}_z)^{-1}\overline{\theta}_z^\top\overline{\theta}_z(\overline{\theta}_y^\top\overline{\theta}_y+\beta\overline{\theta}_z^\top\overline{\theta}_z)^{-1}]\overline{\theta}_y^\top\overline{\theta}_y(\overline{\theta}_y^\top\overline{\theta}_y+\beta\overline{\theta}_z^\top\overline{\theta}_z)^{-1} \\
&-(\sigma_y^2\overline{\theta}_y^\top\overline{\theta}_y+\beta^2\sigma_z^2\overline{\theta}_z^\top\overline{\theta}_z)(\overline{\theta}_y^\top\overline{\theta}_y+\beta\overline{\theta}_z^\top\overline{\theta}_z)^{-1}\overline{\theta}_y^\top\overline{\theta}_y[(\overline{\theta}_y^\top\overline{\theta}_y+\beta\overline{\theta}_z^\top\overline{\theta}_z)^{-1}\overline{\theta}_z^\top\overline{\theta}_z(\overline{\theta}_y^\top\overline{\theta}_y+\beta\overline{\theta}_z^\top\overline{\theta}_z)^{-1}]\Bigg],
\end{aligned}
$$

To obtain the above, we have made use of the derivative of an inverse: $\frac{d}{db}A^{-1} = -A^{-1}\frac{dA}{db}A^{-1}$ [69]. The trace is invariant to transposition of its entire argument and cyclic permutations of matrix products, which we can use to show that the last two lines are equivalent. This allows us to add the last three lines together, and we can also substitute in our shorthand $\overline{\theta}_u$ where indicated to obtain a much simpler expression:

$$
\begin{aligned}
=\;& -2\sigma_x^2 tr\left[(\overline{\theta}_z\overline{\theta}_u-\theta^*_{xz})^\top\overline{\theta}_z(\overline{\theta}_y^\top\overline{\theta}_y+\beta\overline{\theta}_z^\top\overline{\theta}_z)^{-1}\overline{\theta}_y^\top(\overline{\theta}_y\overline{\theta}_u-\theta^*_{xy})\right]+ \\
&\frac{S}{P-S-1}tr\Bigg[2[\beta\sigma_z^2-(\sigma_y^2\overline{\theta}_y^\top\overline{\theta}_y+\beta^2\sigma_z^2\overline{\theta}_z^\top\overline{\theta}_z)(\overline{\theta}_y^\top\overline{\theta}_y+\beta\overline{\theta}_z^\top\overline{\theta}_z)^{-1}]\overline{\theta}_z^\top\overline{\theta}_z(\overline{\theta}_y^\top\overline{\theta}_y+\beta\overline{\theta}_z^\top\overline{\theta}_z)^{-1}\overline{\theta}_y^\top\overline{\theta}_y(\overline{\theta}_y^\top\overline{\theta}_y+\beta\overline{\theta}_z^\top\overline{\theta}_z)^{-1}\Bigg].
\end{aligned}
$$

And, finally, we can compute the value of this derivative at $\beta = 0$ (note that $\overline{\theta}_u$ reduces

to $\overline{\theta}_y^+\theta_{xy}^*$). This will indicate whether incorporating the second term in (3.9) improves the validation loss corresponding with the first term:

$$\begin{aligned}\frac{d}{d\beta}C_{resp}^{val}(\beta)\Big|_{\beta=0} &= -2\sigma_x^2 tr[(\overline{\theta}_z\overline{\theta}_y^+\theta_{xy}^* - \theta_{xz}^*)^\top\overline{\theta}_z\overline{\theta}_y^+(\overline{\theta}_y\overline{\theta}_y^+\theta_{xy}^* - \theta_{xy}^*)] + \frac{S}{P-S-1}tr[-2\sigma_y^2\overline{\theta}_z^\top\overline{\theta}_z\overline{\theta}_y^\top\overline{\theta}_y] \\ &= \frac{-2S\sigma_y^2}{P-S-1}||\overline{\theta}_z\overline{\theta}_y^\top||_F^2, \end{aligned} \tag{S3.10}$$

where we made use of a property of the pseudoinverse, $\overline{\theta}_y^+\overline{\theta}_y\overline{\theta}_y^+ = \overline{\theta}_y^+$, to eliminate the first term.

Deriving T_{JT} using the simplified model To find $\hat{\beta}_y$, can now take the derivative with respect to β, set it to zero, and solve for β. To make this step tractable, we first make use of the simplified model ($\theta_y = I$) and these second order approximations in β:

- $(I+\beta A)^{-1} \approx (I - \beta A + (\beta A)^2)$, and
- $(I+\beta A)^{-2} \approx (I - 2\beta A + 3(\beta A)^2)$

It is worth noting that these approximations are only accurate for small β. However, we have already made an approximation for $\hat{\theta}_z \approx \overline{\theta}_z = \theta_{xz}^*\theta_{xy}^{*+}$, and this term is multiplied by β everywhere it appears in (S3.9). This means that our approximation is already inaccurate for large β, even without employing a second order approximation in β. With these approximations, we can write the second order (in β) approximation of (S3.9) as:

$$
\begin{aligned}
C_{resp}^{val}(\beta) &= O(\beta^3) + \sigma_x^2 tr[\theta_{xy}^{*\top}\theta_{xy}^* - 2\theta_{xy}^{*\top}(I - \beta\overline{\theta}_z^\top\overline{\theta}_z + (\beta\overline{\theta}_z^\top\overline{\theta}_z)^2)(\theta_{xy}^* + \beta\overline{\theta}_z^\top\theta_{xz}^*) \\
&\quad + (\theta_{xy}^* + \beta\overline{\theta}_z^\top\theta_{xz}^*)^\top(I - 2\beta\overline{\theta}_z^\top\overline{\theta}_z + 3(\beta\overline{\theta}_z^\top\overline{\theta}_z)^2)(\theta_{xy}^* + \beta\overline{\theta}_z^\top\theta_{xz}^*)] \\
&\quad + \frac{S\sigma_y^2}{P-S-1} tr[(I + \beta^2\frac{\sigma_z^2}{\sigma_y^2}\overline{\theta}_z^\top\overline{\theta}_z)(I - 2\beta\overline{\theta}_z^\top\overline{\theta}_z + 3(\beta\overline{\theta}_z^\top\overline{\theta}_z)^2)] + N\sigma_y^2 \\
&= O(\beta^3) + \sigma_x^2 tr[\beta^2 B^{*\top}\overline{\theta}_z\overline{\theta}_z^\top\theta_{xz}^* - 2\beta^2 B^{*\top}\overline{\theta}_z\overline{\theta}_z^\top\overline{\theta}_z\theta_{xy}^* + \beta^2\theta_{xy}^{*\top}\overline{\theta}_z^\top\overline{\theta}_z\overline{\theta}_z^\top\overline{\theta}_z\theta_{xy}^*] \\
&\quad + \frac{S\sigma_y^2}{P-S-1}(N - 2\beta tr[\overline{\theta}_z^\top\overline{\theta}_z] + \beta^2(3tr[\overline{\theta}_z^\top\overline{\theta}_z\overline{\theta}_z^\top\overline{\theta}_z] + \frac{\sigma_z^2}{\sigma_y^2}tr[\overline{\theta}_z^\top\overline{\theta}_z])) + N\sigma_y^2 \\
&= O(\beta^3) + \beta^2\sigma_x^2 tr[(\theta_{xz}^* - \overline{\theta}_z\theta_{xy}^*)^\top\overline{\theta}_z\overline{\theta}_z^\top(\theta_{xz}^* - \overline{\theta}_z\theta_{xy}^*)] \\
&\quad + \frac{S\sigma_y^2}{P-S-1}(-2\beta tr[\overline{\theta}_z^\top\overline{\theta}_z] + \beta^2(3tr[\overline{\theta}_z^\top\overline{\theta}_z\overline{\theta}_z^\top\overline{\theta}_z] + \frac{\sigma_z^2}{\sigma_y^2}tr[\overline{\theta}_z^\top\overline{\theta}_z])) + N\sigma_y^2\left(1 + \frac{S}{P-S-1}\right).
\end{aligned}
\tag{S3.11}
$$

Now we can easily find the minimal β, which we write as $\hat{\beta}_y$:

$$
\begin{aligned}
\frac{dC_{resp}^{val}}{d\beta} = 0 &= O(\beta^2) + 2\beta\sigma_x^2 tr[(\theta_{xz}^* - \overline{\theta}_z\theta_{xy}^*)^\top\overline{\theta}_z\overline{\theta}_z^\top(\theta_{xz}^* - \overline{\theta}_z\theta_{xy}^*)] \\
&\quad + \frac{S\sigma_y^2}{P-S-1}(-2tr[\overline{\theta}_z^\top\overline{\theta}_z] + 2\beta(3tr[\overline{\theta}_z^\top\overline{\theta}_z\overline{\theta}_z^\top\overline{\theta}_z] + \frac{\sigma_z^2}{\sigma_y^2}tr[\overline{\theta}_z^\top\overline{\theta}_z]) \\
\hat{\beta}_y &\approx \frac{tr[\overline{\theta}_z^\top\overline{\theta}_z]}{3tr[\overline{\theta}_z^\top\overline{\theta}_z\overline{\theta}_z^\top\overline{\theta}_z] + \frac{\sigma_z^2}{\sigma_y^2}tr[\overline{\theta}_z^\top\overline{\theta}_z] + \frac{P-S-1}{S}\frac{\sigma_x^2}{\sigma_y^2}tr[(\theta_{xz}^* - \overline{\theta}_z\theta_{xy}^*)^\top\overline{\theta}_z\overline{\theta}_z^\top(\theta_{xz}^* - \overline{\theta}_z\theta_{xy}^*)]}
\end{aligned}
\tag{S3.12}
$$

We can then use (S3.11) and (S3.12), to write down the relative improvement in neural response validation loss (3.15).

S3.1.3 Numerical Verification of Analytical Results for the Linear Network

In the above analysis, we made use of many approximations in order to make the derivations tractable and the results simpler and easier to understand. Here, we compare these approximate results to those obtained by performing joint-training numerically on specific instantiations of this network.

For each network, we set $P = 50$, and pick a specific θ_{xy}^* and θ_{xz}^* with dimensions $S = 30$ and $N = 10$, and we vary both the dimensionality of the computation $Q \in 1, 4, 8$ and

the greatest possible compatibility dimension $Q^*_{max} \in 1, 4, 8$. Thus if $Q < Q^*_{max}$, $Q^* = Q$, otherwise $Q^* = Q^*_{max}$. For each unique pair of (Q^*, Q), we pick a single θ^*_{xz} that satisfies the definition of Q^*, given θ^*_{xy}, and then simulate training and validation data using $\sigma_x = 1$, $\sigma_y = 0.3$, and $\sigma_z = 1$.

We then perform the joint-training process numerically. First, we use the SLSQP algorithm to minimize the joint cost function on the training data with respect to θ_u and θ_z. We then repeat this minimization for many training sets sampled from the same distribution, and average the resulting validation loss over these different training sets. We then repeat this process for logarithmically spaced values of β ($\beta \in \{0, 10^{-6}, 10^{-5.5}, ..., 10^{3.5}, 10^4\}$), and use the averaged validation loss to select $\hat{\beta}_y$. We can then calculate T_{JT} and T_{TT} numerically, from the validation losses obtained with $\beta = \hat{\beta}_y$ and $\beta = 10^4$, respectively, along with $\beta = 0$. We then compare these to the analytical approximations of these quantities, (3.15) and (3.17), respectively.

The results of this process for each unique θ^*_{xz} are shown in Figure S3.1A. While there is not a perfect relationship between the numerical and analytical approximations of T_{JT} and T_{TT}, there is a strong positive trend, especially for $Q > 1$. This trend also preserves our main theoretical result from this analysis, that the auxiliary task that produces the highest T_{JT} is the one with the most output dimensions, all of which are compatible with the neural responses (highest $Q*$).

S3.2 Supporting derivations

Here we include several derivations which are necessary for a complete treatment of our analysis in Section 3.3, but are not necessary to understand the basic principles.

Inverse of a Gaussian random matrix: Here, we are interested in the first and second moments of the distribution of R^+, where the elements of $R \in \mathbb{R}^{m \times n}, m + 1 < n$ are i.i.d Gaussian random variables with zero-mean and variance σ^2. Note that if instead $m > n + 1$, we can perform a similar derivation by considering $R^{+\top}$.

First, we note that the elements of R^+ must be i.i.d. because permuting the rows (columns) of R and thus the columns (rows) of R^+ should not affect the distribution of R^+ since it does not affect the distribution of R (its elements are i.i.d.). The mean of the

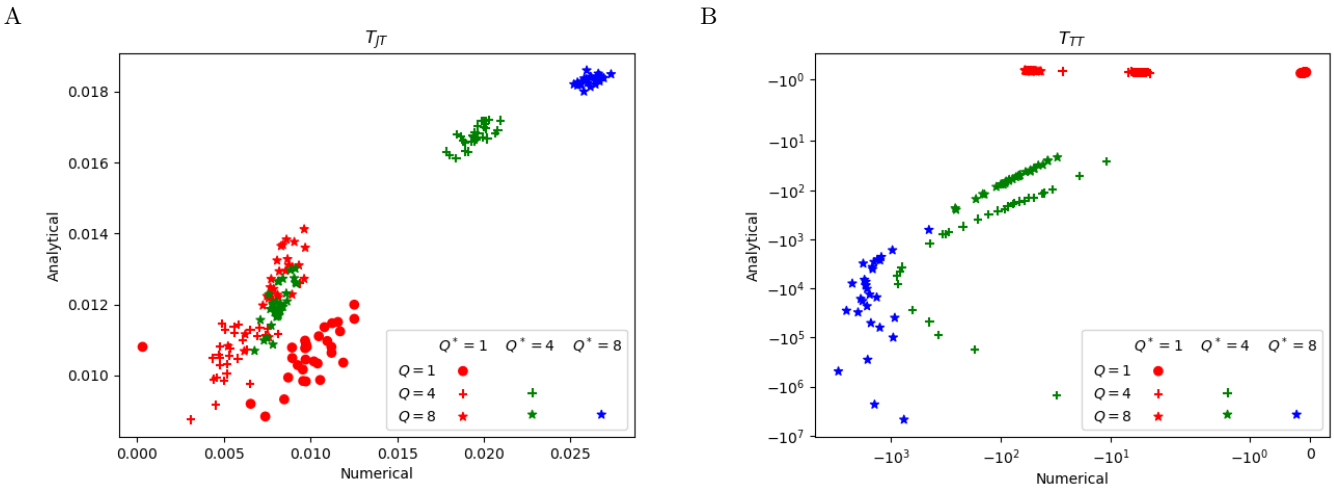


Figure S3.1: **Numerical verification of analytical derivations of T_{JT} (3.15) and T_{TT} (3.17).** Relationship between numerical and analytical derivations of relative improvement over response-training obtained by joint-training (panel A; T_{JT} (3.15)) and task-training (panel B; T_{TT} (3.17)). Agreement is not perfect, but mostly preserves the qualitative conclusions made from (3.15) and (3.17). In particular, T_{JT} is maximized when there is a higher dimensional computation (higher Q), without adding dimensions that are not represented in the neural activity ($Q > Q^*$). Conversely, T_{TT} is maximized for lower dimensional computations. See Section S3.1.3 for simulation details.

distribution of $[R^+]_{ij}$ must be zero because negating R must negate R^+, but this also does not change the distribution of R, since the distribution of each element is symmetric about 0.

To compute the variance of the distribution of $[R^+]_{ij}$, we make use of the Inverse-Wishart distribution. If $m + 1 < n$, then $(RR^\top)^{-1} = (R^{+\top}R^+)$ is an Inverse-Wishart random variable with mean $\frac{1}{(n-m-1)\sigma^2}I$. Each element of this matrix is the expected dot product of two columns of R^+. Off-diagonal elements correspond to the dot product of two different columns, and are accordingly zero, since the elements of R^+ are independent and zero-mean. On-diagonal elements correspond to the sum of n independent, squared elements of R^+. Thus $\mathbb{E}[[R^+]_{ij}] = \frac{1}{n(n-m-1)\sigma^2}$, the diagonal value of the Inverse-Wishart random matrix's mean divided by n.

Validity of the approximation $\hat{\theta}_z \approx \overline{\theta}_z$: If we were instead solving for $\hat{\theta}_z$ by taking the derivative of (3.9) with respect to θ_z and setting it to zero, we would have:

$$\begin{aligned}
\frac{dC}{d\theta_z} = 0 &= 2\beta \sum_{i=1}^{P} \theta_z \theta_u x_i x_i^\top \theta_u^\top - z_i x_i^\top \theta_u^\top, \\
0 &= \theta_z \theta_u X X^\top \theta_u^\top - Z X^\top \theta_u^\top, \\
\hat{\theta}_z &= Z X^\top \theta_u^\top (\theta_u X X^\top \theta_u^\top)^{-1}, \\
\hat{\theta}_z &= (\theta_{xz}^* X + \Xi_z) X^\top \theta_u^\top (\theta_u X X^\top \theta_u^\top)^{-1}, \\
\hat{\theta}_z &= (\theta_{xz}^* \theta_u^+ \theta_u X + \theta_{xz}^* (I - \theta_u^+ \theta_u) X + \Xi_z) X^\top \theta_u^\top (\theta_u X X^\top \theta_u^\top)^{-1}, \\
\hat{\theta}_z &= \theta_{xz}^* \theta_u^+ (\theta_u X X^\top \theta_u^\top)(\theta_u X X^\top \theta_u^\top)^{-1} + E, \\
\hat{\theta}_z &= \theta_{xz}^* \theta_u^+ + E, \qquad \text{(S3.13)}
\end{aligned}$$

where $E = (\theta_{xz}^*(I - \theta_u^+ \theta_u) X + \Xi_z) X^\top \theta_u^\top (\theta_u X X^\top \theta_u^\top)^{-1}$. Using (S3.4), we can see that the error between θ_u and θ_{xy}^* is zero-mean and dominated by $\Xi_y X^+$ for small β, or $(\beta \hat{\theta}_z^\top \Xi_z + \Xi_y) X^+$ when θ_{xz}^* and θ_{xy}^* are compatible. We can then say that the error in our approximation is zero-mean, and has a variance on the order of $\frac{1}{P-S-1}(\frac{\sigma_x^2}{\sigma_y^2} + \frac{\sigma_z^2}{\sigma_x^2})$ when θ_{xz}^* and θ_{xy}^* are compatible. Otherwise, there is also a bias in the approximation error on the order of $\theta_{xz}^*(I - \theta_{xy}^{*+}\theta_{xy}^*)$, as well as additional variance terms. Thus the error is smaller when we have compatibility, $\sigma_z << \sigma_x << \sigma_y$, and $P >> S$.

S3.3 Compute Info

All numerical experiments were performed on a cluster with a heterogeneous collection of computing nodes; the smallest nodes had 32 cores and 256G of memory. We ran two types of jobs on these nodes, those described in Section S3.1.3, and those described in Sections 3.4. Of the former, 180 jobs were run, with a typical job taking about an hour and a half to complete. Of the later, 129600 jobs were run, with a typical job taking about 8 minutes to complete. It is worth noting that the code allows for GPU usage for these later jobs, but it did not provide a speedup in our case, due to the small network size.

Chapter 4

CONCLUSION

In this work, I have used Linderman and Gershman's perspective on incorporating scientific hypotheses into statistical models as assumptions to develop models that build towards an ideal model of neural activity, as described my Marr's three levels of abstraction. I have presented two examples of such hypotheses, that a given population of neurons can be well described as participating in some computation, and that individual neurons belong to one of a finite number of types. For each, I started with a baseline model that describes the basic methodology employed by the field to address that hypothesis, then showed how an improved model that properly incorporates the hypothesis as an assumption leads to fitted models that better predict neural activity and can be more reliably interpreted.

Although the focus here has been on neural systems, it is worth noting that many of these ideas could be widely applicable to many fields that use computational models to understand complex systems. Marr developed his framework for neuroscience, but generally it is desirable to understand complex systems at many levels of abstraction. For example, in weather modelling, it is useful to have detailed mechanistic models that explain how local atmospheric dynamics lead to specific weather patterns at the resolution of hours and kilometers (or better), but it is also useful to understand seasonal and multi-year weather cycles like El Niño. Similarly, if we wish to fully understand a given political system, we must understand the behaviors of individuals in that system, as well as how the system shapes and aggregates those behaviors. Generally, levels of abstraction are connected with the idea of proximality of causation, where more mechanistic levels correspond to understanding more proximal causes, and more abstract levels correspond to understanding more ultimate causes. In neuroscience, this connection is made via the theory of evolution - a computation is the ultimate cause of neural activity because the need to perform that computation provided evolutionary pressure on the circuit. Linderman and Gershman's proposal for incorporating

scientific hypotheses into statistical models as assumptions is also directly applicable to statistical models of any phenomena. Indeed, all of chapter 3 could be rewritten about some entity other than cells, which is modelled via a likelihood function (especially a GLM) and is believed to have unknown types.

Of course, the ultimate goal of computational neuroscience is an integrated model of a brain that can accurately predict neural activity and behavior across a wide range of brain regions and behavioral contexts, and can be understood in terms of Marr's three levels. The two methodologies that I have proposed can be understood as improved tools for defining models that fit neatly into Marr's framework, making them easy to integrate with each other. Here I explore in more detail how one would actually go about integrating models defined in this way, building towards a comprehensive model of the brain. One such hypothetical path is shown in Figure 4.1.

Integrating across levels of abstraction The first type of integration to consider is between two models that describe a system at different levels of abstraction, such as the two models discussed in this work (see, e.g. Figure 4.1 A-D). In general, there will be a range of options in terms of how rich an integration may be, ranging from a single fully integrated model to two models pieced together, like the baseline models we looked at. In this case, a simple option would be to use an ANN with an auxiliary task to predict spiking activity by appending a GLM to the neural prediction end, whose parameters could then be optimized along with the ANN's. After the ANN is fitted, the GLM parameters could be re-fit with the GMM-based cell-types assumption, using the EM algorithm for discovering cell types detailed in chapter 2, while holding the rest of the ANN parameters fixed. A more complete integration might adapt the EM algorithm further to include fitting the ANN parameters, and/or expand the set of parameters that are considered cell-type-dependent to include θ_y, which would determine which ANN unit activities are presented as input to each GLM. A reasonable way to do accomplish the latter in a CNN architecture might be to hypothesize that units of the same channel in the same layer get mapped to neurons of the same cell type, as these units play identical functional roles, but applied to translated regions of the stimulus. Such an approach could be strengthened by defining channels to allow for further

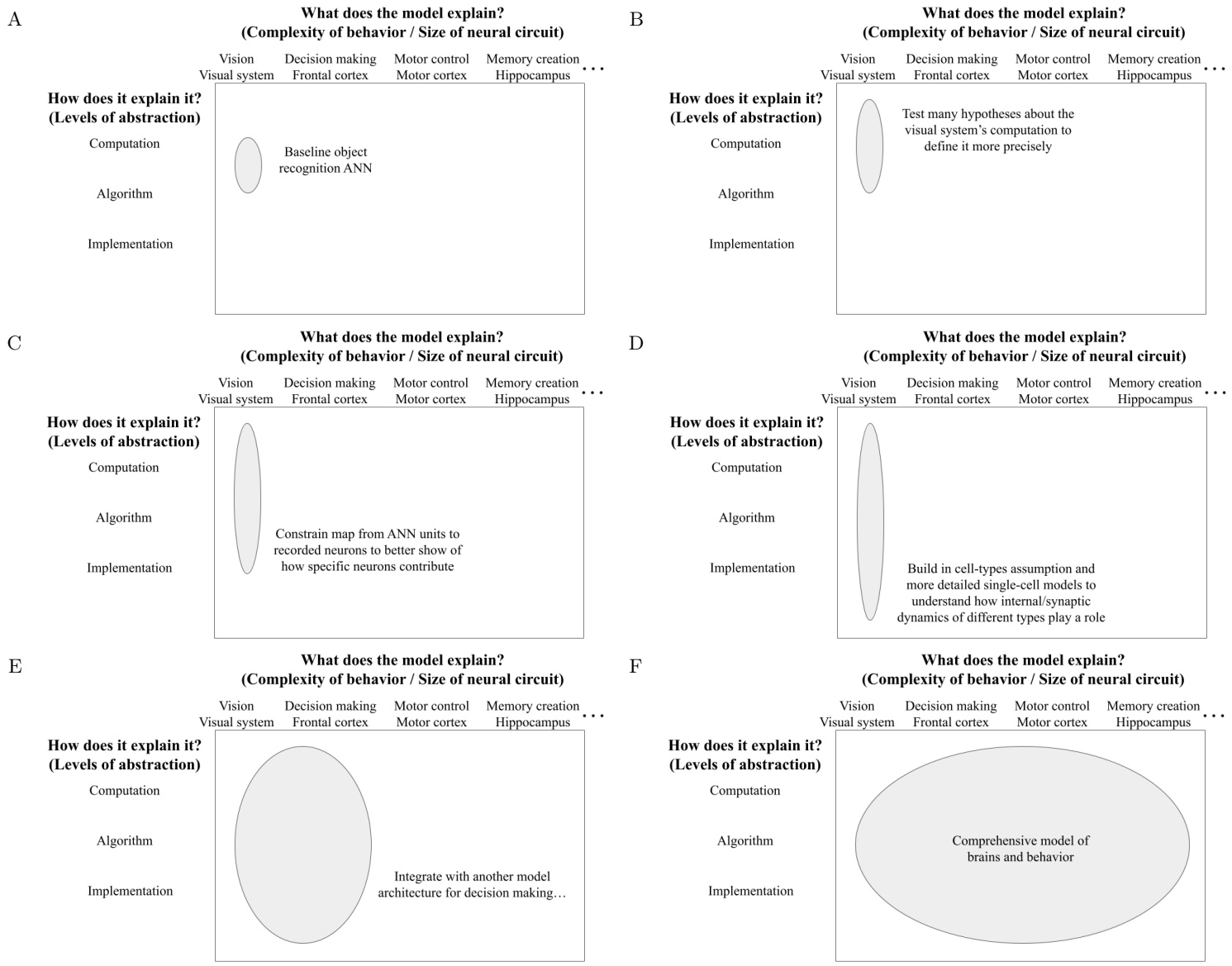


Figure 4.1: **A hypothetical sequence of model integrations, leading toward a whole-brain model.**

invariance, such as rotation of the stimulus (see [30, 85]).

While a more complete integration is preferable in terms of the resulting model, it may become prohibitively costly in terms of the computational resources necessary. In each chapter, the better integrated model takes more computational resources to fit than the baseline

model, but in each case the use of advanced statistical tools and computing techniques makes the improved performance well worth this increased cost. However, fully integrating many such models will eventually lead to intractability, so it may be necessary to use increasingly simplifying heuristics where appropriate (such as how we used variational inference in chapter 2) to keep the time needed to fit a model within reason.

Another prospect for future integration is relying on other modalities of data to reduce the number of parameters that need to be fitted in the model. For example, modern experiments are capable of tracing the exact morphology of neurons, including their dendritic trees and synapses [50], allowing for the data to define the ANN architecture from the connections between neurons and even between specific regions of the same neuron. This could also allow for a data-defined identification of each neuronal recording with an ANN unit, and even contribute to determination of cell types. Other experiments measure gene expression [53] after recording their activity, allowing for an even more accurate and potentially less computationally intensive determination of cell type. While there are limits to how accurately many such modalities of data can be collected from the same experiment, in principle much is possible, and experimental technology is improving every year.

Integrating across brain regions and functional specializations The other type of integration necessary to build towards a model of a whole brain is piecing together models of different brain regions during different behaviors (see e.g. Figure 4.1 E-F). In some ways, such integrations should be more straightforward, provided that each network model explains neural activity in terms of computation and implementation. If the neurons in each brain region have been described using the same basic model of a single neuron, then no integration is necessary; otherwise, it may be desirable to substitute in an appropriate single neuron model that is expressive enough to perform the necessary functions in each network model, for example to maintain a cohesive idea of cell types.

How exactly the network models are connected together may depend on the imputed computations for each region; if the two computations can be understood as two sequential or parallel algorithmic stages of a single, more abstract computation, then that may inform a specific sequential or parallel arrangement. This is consistent with the idea of regional

specialization of function within the brain [5], or even the idea of the cortical microcircuit [17] as a discrete functional unit. Such ideas can be understood as hypotheses about the brain, and could even be tested by making assumptions about the architecture of a neural circuit that spans brain regions or cortical columns. In general, it may not always be possible to cleanly divide computations along anatomical lines, and determining how to connect two network models together may be less straightforward. However, any understanding of each network at the computational and/or algorithmic level should guide this process, as well as the process of describing the function of the bigger circuit computationally and algorithmically. Data that directly measures inter-regional connectivity can also assist with this integration, to the extent that each network model identifies model units with physical neurons at a resolution comparable to the connectivity data or better.

One challenge for the field is that it may be very difficult to understand some set of brain regions without properly incorporating the function performed by another region. In general, it will be easier to gain an integrated understanding of the functioning of many brain regions the better all brain regions are understood at each level of abstraction. Likewise, the more densely individual experiments record from brains and the more complex the set of behaviors they observe or evoke, the better we will be able to test models that seek to explain a wide range of brain activity and/or animal behavior. In turn, as more such experimental data becomes available, it will be more profitable for modellers to attempt to fit and test such integrated models. Such experiments may even allow for well-tested models that span many behaviors or brain regions, but do not explain functioning across Marr's three levels. This could lead to a different path towards an integrated model of a whole brain from the one shown in Figure 4.1.

Regardless of the experimental data available, or how comprehensive a model one seeks to define, using a framework like Marr's three levels of analysis should facilitate future model development and integration with other models. Further, this work suggests, similarly to that of Linderman and Gershman, that a productive way to define models under this framework is by including hypotheses of any level of abstraction about the functioning of a neural circuit as assumptions in statistical models fitted to data, and testing whether those assumptions result in improved prediction of data held out from fitting. In particular, this

work demonstrates the value of using simulation studies and theoretical analysis to establish how reliably the fitted models can be interpreted, and suggests that, in this sense, models with such assumptions are more reliable.

BIBLIOGRAPHY

[1] Overview :: Allen Brain Atlas: Cell Types, June 2023.

[2] Johnatan Aljadeff, Benjamin J. Lansdell, Adrienne L. Fairhall, and David Kleinfeld. Analysis of Neuronal Spike Trains, Deconstructed. *Neuron*, 91(2):221–259, July 2016.

[3] Johnatan Aljadeff, Merav Stern, and Tatyana Sharpee. Transition to Chaos in Random Networks with Cell-Type-Specific Connectivity. *Physical Review Letters*, 114(8), February 2015.

[4] S.-I. Amari. Learning Patterns and Pattern Sequences by Self-Organizing Nets of Threshold Elements. *IEEE Transactions on Computers*, C-21(11):1197–1206, November 1972. Conference Name: IEEE Transactions on Computers.

[5] John R. Anderson. *How can the human mind occur in the physical universe?* Number 3 in Oxford series on cognitive models and architectures. Oxford University Press, New York, New York, first issued as an oxford university press paperback edition, 2010.

[6] Blaise Agüera y Arcas, Adrienne L. Fairhall, and William Bialek. Computation in a Single Neuron: Hodgkin and Huxley Revisited. *Neural Computation*, 15(8):1715–1749, August 2003.

[7] Eric B. Bartlett. Dynamic node architecture learning: An information theoretic approach. *Neural Networks*, 7(1):129–140, January 1994.

[8] David Beniaguev, Idan Segev, and Michael London. Single cortical neurons as deep artificial neural networks. *Neuron*, 109(17):2727–2739, 2021. Publisher: Elsevier.

[9] Michael J. Berry and Markus Meister. Refractoriness and Neural Precision. *Journal of Neuroscience*, 18(6):2200–2211, March 1998. Publisher: Society for Neuroscience Section: ARTICLE.

[10] Tom Binzegger, Rodney J. Douglas, and Kevan A. C. Martin. A Quantitative Map of the Circuit of Cat Primary Visual Cortex. *Journal of Neuroscience*, 24(39):8441–8453, September 2004. Publisher: Society for Neuroscience Section: Behavioral/Systems/Cognitive.

[11] David M. Blei, Alp Kucukelbir, and Jon D. McAuliffe. Variational Inference: A Review for Statisticians. *Journal of the American Statistical Association*, 112(518):859–877, April 2017. Publisher: Taylor & Francis _eprint: https://doi.org/10.1080/01621459.2017.1285773.

[12] Valentino Braitenberg and Almut Schüz. *Cortex: Statistics and Geometry of Neuronal Connectivity*. Springer Science & Business Media, March 2013. Google-Books-ID: yGvvCAAAQBAJ.

[13] Romain Brette and Wulfram Gerstner. Adaptive Exponential Integrate-and-Fire Model as an Effective Description of Neuronal Activity. *Journal of Neurophysiology*, 94(5):3637–3642, November 2005. Publisher: American Physiological Society.

[14] Emery N. Brown, Loren M. Frank, Dengda Tang, Michael C. Quirk, and Matthew A. Wilson. A Statistical Paradigm for Neural Spike Train Decoding Applied to Position Prediction from Ensemble Firing Patterns of Rat Hippocampal Place Cells. *Journal of Neuroscience*, 18(18):7411–7425, September 1998. Publisher: Society for Neuroscience Section: ARTICLE.

[15] Bingni W. Brunton, Matthew M. Botvinick, and Carlos D. Brody. Rats and Humans Can Optimally Accumulate Evidence for Decision-Making. *Science*, 340(6128):95–98, April 2013. Publisher: American Association for the Advancement of Science.

[16] Santiago A. Cadena, George H. Denfield, Edgar Y. Walker, Leon A. Gatys, Andreas S. Tolias, Matthias Bethge, and Alexander S. Ecker. Deep convolutional models improve predictions of macaque V1 responses to natural images. *PLOS Computational Biology*, 15(4):e1006897, April 2019. Publisher: Public Library of Science.

[17] Fioravante Capone, Matteo Paolucci, Federica Assenza, Nicoletta Brunelli, Lorenzo Ricci, Lucia Florio, and Vincenzo Di Lazzaro. Canonical cortical circuits: current evidence and theoretical implications. *Neuroscience and Neuroeconomics*, 5:1–8, April 2016. Publisher: Dove Medical Press _eprint: https://www.tandfonline.com/doi/pdf/10.2147/NAN.S70816.

[18] Rich Caruana. Multitask Learning. *Machine Learning*, 28(1):41–75, July 1997.

[19] Tina Cheuk. Can AI be racist? Color-evasiveness in the application of machine learning to science assessments. *Science Education*, 105(5):825–836, 2021. _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/sce.21671.

[20] E. J. Chichilnisky. A simple white noise analysis of neuronal light responses. *Network: Computation in Neural Systems*, 12(2):199–213, January 2001. Publisher: Taylor & Francis _eprint: https://www.tandfonline.com/doi/pdf/10.1080/net.12.2.199.213.

[21] Marc Colonnier. Synaptic patterns on different cell types in the different laminae of the cat visual cortex. An electron microscope study. *Brain Research*, 9(2):268–287, July 1968.

[22] Diego Contreras. Electrophysiological classes of neocortical neurons. *Neural Networks*, 17(5):633–646, June 2004.

[23] Christopher J. Cueva, Alex Saez, Encarni Marcos, Aldo Genovesio, Mehrdad Jazayeri, Ranulfo Romo, C. Daniel Salzman, Michael N. Shadlen, and Stefano Fusi. Low-dimensional dynamics for working memory and time encoding. *Proceedings of the National Academy of Sciences*, 117(37):23021–23032, September 2020. Publisher: National Academy of Sciences Section: Biological Sciences.

[24] Christopher J Cueva and Xue-Xin Wei. Emergence of grid-like representations by training recurrent neural networks to perform spatial localization. *arXiv preprint arXiv:1803.07770*, 2018.

[25] John P. Cunningham and Byron M. Yu. Dimensionality reduction for large-scale neural recordings. *Nature Neuroscience*, 17(11):1500–1509, November 2014. Number: 11 Publisher: Nature Publishing Group.

[26] Yehuda Dar and Richard G. Baraniuk. Double Double Descent: On Generalization Errors in Transfer Learning between Linear Regression Tasks. *SIAM Journal on Mathematics of Data Science*, 4(4):1447–1472, December 2022. Publisher: Society for Industrial and Applied Mathematics.

[27] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum Likelihood from Incomplete Data Via the EM Algorithm. *Journal of the Royal Statistical Society: Series B (Methodological)*, 39(1):1–22, 1977. _eprint: https://rss.onlinelibrary.wiley.com/doi/pdf/10.1111/j.2517-6161.1977.tb01600.x.

[28] Yi Dong, Stefan Mihalas, Alexander Russell, Ralph Etienne-Cummings, and Ernst Niebur. Estimating Parameters of Generalized Integrate-and-Fire Neurons from the Maximum Likelihood of Spike Trains. *Neural Computation*, 23(11):2833–2867, August 2011. Publisher: MIT Press.

[29] Alexis Dubreuil, Adrian Valente, Manuel Beiran, Francesca Mastrogiuseppe, and Srdjan Ostojic. The role of population structure in computations through neural dynamics. *Nature Neuroscience*, 25(6):783–794, June 2022. Number: 6 Publisher: Nature Publishing Group.

[30] Alexander S. Ecker, Fabian H. Sinz, Emmanouil Froudarakis, Paul G. Fahey, Santiago A. Cadena, Edgar Y. Walker, Erick Cobos, Jacob Reimer, Andreas S. Tolias, and Matthias Bethge. A rotation-equivariant convolutional neural network model of primary visual cortex. In *International Conference on Learning Representations*, September 2018.

[31] G. Bard Ermentrout and David H. Terman. *Mathematical Foundations of Neuroscience*, volume 35 of *Interdisciplinary Applied Mathematics*. Springer, New York, NY, 2010.

[32] Callie Federer, Haoyan Xu, Alona Fyshe, and Joel Zylberberg. Improved object recognition using neural networks trained to mimic the brain's statistical properties. *Neural Networks*, 131:103–114, November 2020.

[33] Jane E. Fountain. The moon, the ghetto and artificial intelligence: Reducing systemic racism in computational algorithms. *Government Information Quarterly*, 39(2):101645, April 2022.

[34] Rohan Gala, Nathan Gouwens, Zizhen Yao, Agata Budzillo, Osnat Penn, Bosiljka Tasic, Gabe Murphy, Hongkui Zeng, and Uygar Sümbül. A coupled autoencoder approach for multi-modal analysis of cell types. In *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

[35] Wulfram Gerstner, Raphael Ritz, and J. Leo van Hemmen. Why spikes? Hebbian learning and retrieval of time-resolved excitation patterns. *Biological Cybernetics*, 69(5):503–515, October 1993.

[36] Nathan W. Gouwens, Staci A. Sorensen, Fahimeh Baftizadeh, Agata Budzillo, Brian R. Lee, Tim Jarsky, Lauren Alfiler, Katherine Baker, Eliza Barkan, Kyla Berry, Darren Bertagnolli, Kris Bickley, Jasmine Bomben, Thomas Braun, Krissy Brouner, Tamara Casper, Kirsten Crichton, Tanya L. Daigle, Rachel Dalley, Rebecca A. de Frates, Nick Dee, Tsega Desta, Samuel Dingman Lee, Nadezhda Dotson, Tom Egdorf, Lauren Ellingwood, Rachel Enstrom, Luke Esposito, Colin Farrell, David Feng, Olivia Fong, Rohan Gala, Clare Gamlin, Amanda Gary, Alexandra Glandon, Jeff Goldy, Melissa Gorham, Lucas Graybuck, Hong Gu, Kristen Hadley, Michael J. Hawrylycz, Alex M. Henry, DiJon Hill, Madie Hupp, Sara Kebede, Tae Kyung Kim, Lisa Kim, Matthew Kroll, Changkyu Lee, Katherine E. Link, Matthew Mallory, Rusty Mann, Michelle Maxwell, Medea McGraw, Delissa McMillen, Alice Mukora, Lindsay Ng, Lydia Ng, Kiet Ngo, Philip R. Nicovich, Aaron Oldre, Daniel Park, Hanchuan Peng, Osnat Penn, Thanh Pham, Alice Pom, Zoran Popović, Lydia Potekhina, Ramkumar Rajanbabu, Shea Ransford, David Reid, Christine Rimorin, Miranda Robertson, Kara Ronellenfitch, Augustin Ruiz, David Sandman, Kimberly Smith, Josef Sulc, Susan M. Sunkin, Aaron Szafer, Michael Tieu, Amy Torkelson, Jessica Trinh, Herman Tung, Wayne Wakeman, Katelyn Ward, Grace Williams, Zhi Zhou, Jonathan T. Ting, Anton Arkhipov, Uygar Sümbül, Ed S. Lein, Christof Koch, Zizhen Yao, Bosiljka Tasic, Jim Berg, Gabe J. Murphy, and Hongkui Zeng. Integrated Morphoelectric and Transcriptomic Classification of Cortical GABAergic Cells. *Cell*, 183(4):935–953.e19, November 2020.

[37] Nathan W. Gouwens, Staci A. Sorensen, Jim Berg, Changkyu Lee, Tim Jarsky, Jonathan Ting, Susan M. Sunkin, David Feng, Costas A. Anastassiou, Eliza Barkan, Kris Bickley, Nicole Blesie, Thomas Braun, Krissy Brouner, Agata Budzillo, Shiella Caldejon, Tamara

Casper, Dan Castelli, Peter Chong, Kirsten Crichton, Christine Cuhaciyan, Tanya L. Daigle, Rachel Dalley, Nick Dee, Tsega Desta, Song-Lin Ding, Samuel Dingman, Alyse Doperalski, Nadezhda Dotson, Tom Egdorf, Michael Fisher, Rebecca A. de Frates, Emma Garren, Marissa Garwood, Amanda Gary, Nathalie Gaudreault, Keith Godfrey, Melissa Gorham, Hong Gu, Caroline Habel, Kristen Hadley, James Harrington, Julie A. Harris, Alex Henry, DiJon Hill, Sam Josephsen, Sara Kebede, Lisa Kim, Matthew Kroll, Brian Lee, Tracy Lemon, Katherine E. Link, Xiaoxiao Liu, Brian Long, Rusty Mann, Medea McGraw, Stefan Mihalas, Alice Mukora, Gabe J. Murphy, Lindsay Ng, Kiet Ngo, Thuc Nghi Nguyen, Philip R. Nicovich, Aaron Oldre, Daniel Park, Sheana Parry, Jed Perkins, Lydia Potekhina, David Reid, Miranda Robertson, David Sandman, Martin Schroedter, Cliff Slaughterbeck, Gilberto Soler-Llavina, Josef Sulc, Aaron Szafer, Bosiljka Tasic, Naz Taskin, Corinne Teeter, Nivretta Thatra, Herman Tung, Wayne Wakeman, Grace Williams, Rob Young, Zhi Zhou, Colin Farrell, Hanchuan Peng, Michael J. Hawrylycz, Ed Lein, Lydia Ng, Anton Arkhipov, Amy Bernard, John W. Phillips, Hongkui Zeng, and Christof Koch. Classification of electrophysiological and morphological neuron types in the mouse visual cortex. *Nature Neuroscience*, 22(7):1182–1195, July 2019. Number: 7 Publisher: Nature Publishing Group.

[38] Peter A. Groblewski, Douglas R. Ollerenshaw, Justin T. Kiggins, Marina E. Garrett, Chris Mochizuki, Linzy Casal, Sissy Cross, Kyla Mace, Jackie Swapp, Sahar Manavi, Derric Williams, Stefan Mihalas, and Shawn R. Olsen. Characterization of Learning, Motivation, and Visual Perception in Five Transgenic Mouse Lines Expressing GCaMP in Distinct Cell Populations. *Frontiers in Behavioral Neuroscience*, 14, 2020.

[39] Qingbei Guo, Xiao-Jun Wu, Josef Kittler, and Zhiquan Feng. Differentiable neural architecture learning for efficient neural networks. *Pattern Recognition*, 126(C), June 2022.

[40] Kameron Decker Harris, Tatiana Dashevskiy, Joshua Mendoza, Alfredo J. Garcia III, Jan-Marino Ramirez, and Eric Shea-Brown. Different roles for inhibition in the rhythm-generating respiratory network. *arXiv:1610.04258 [q-bio]*, October 2016. arXiv: 1610.04258.

[41] Trevor Hastie, Jerome Friedman, and Robert Tibshirani. *The Elements of Statistical Learning*. Springer Series in Statistics. Springer, New York, NY, 2001.

[42] Darrell Haufler, Shinya Ito, Christof Koch, and Anton Arkhipov. Simulations of cortical networks using spatially extended conductance-based neuronal models. *The Journal of Physiology*, 601(15):3123–3139, 2023. _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1113/JP284030.

[43] A. L. Hodgkin and A. F. Huxley. A quantitative description of membrane current and its application to conduction and excitation in nerve. *The Journal of Physiology*, 117(4):500–544, 1952. _eprint: https://physoc.onlinelibrary.wiley.com/doi/pdf/10.1113/jphysiol.1952.sp004764.

[44] E.M. Izhikevich. Simple model of spiking neurons. *IEEE Transactions on Neural Networks*, 14(6):1569–1572, November 2003.

[45] Eugene M. Izhikevich. *Dynamical Systems in Neuroscience: The Geometry of Excitability and Bursting*. MIT press, 2007.

[46] Xiaolong Jiang, Shan Shen, Cathryn R. Cadwell, Philipp Berens, Fabian Sinz, Alexander S. Ecker, Saumil Patel, and Andreas S. Tolias. Principles of connectivity among morphologically defined cell types in adult neocortex. *Science*, 350(6264), November 2015. Publisher: American Association for the Advancement of Science Section: Research Article.

[47] Eric Jonas and Konrad Kording. Automatic discovery of cell types and microcircuitry from neural connectomics. *eLife*, 4, April 2015.

[48] Ilenna Simone Jones and Konrad Paul Kording. Might a single neuron solve interesting machine learning problems through successive computations on its dendritic tree? *Neural Computation*, 33(6):1554–1571, 2021. Publisher: MIT Press One Rogers Street, Cambridge, MA 02142-1209, USA journals-info ....

[49] Ingmar Kanitscheider and Ila Fiete. Training recurrent networks to generate hypotheses about how the brain solves hard navigation problems. *Advances in Neural Information Processing Systems 30*, pages 4529–4538, 2017.

[50] Yoshiyuki Kubota, Jaerin Sohn, and Yasuo Kawaguchi. Large Volume Electron Microscopy and Neural Microcircuit Analysis. *Frontiers in Neural Circuits*, 12, 2018.

[51] Andrew K. Lampinen and Surya Ganguli. An analytic theory of generalization dynamics and transfer learning in deep linear networks, January 2019. arXiv:1809.10374 [cs, stat].

[52] L. LAPIQUE. Recherches quantitatives sur l' excitation electrique des nerfs traitee comme une polarization. *Journal of Physiology and Pathololgy*, 9:620–635, 1907.

[53] Brian R Lee, Agata Budzillo, Kristen Hadley, Jeremy A Miller, Tim Jarsky, Katherine Baker, DiJon Hill, Lisa Kim, Rusty Mann, Lindsay Ng, Aaron Oldre, Ram Rajanbabu, Jessica Trinh, Sara Vargas, Thomas Braun, Rachel A Dalley, Nathan W Gouwens, Brian E Kalmbach, Tae Kyung Kim, Kimberly A Smith, Gilberto Soler-Llavina, Staci Sorensen, Bosiljka Tasic, Jonathan T Ting, Ed Lein, Hongkui Zeng, Gabe J Murphy, and Jim Berg. Scaled, high fidelity electrophysiological, morphological, and transcriptomic cell characterization. *eLife*, 10:e65482, August 2021. Publisher: eLife Sciences Publications, Ltd.

[54] Zhe Li, Wieland Brendel, Edgar Y. Walker, Erick Cobos, Taliah Muhammad, Jacob Reimer, Matthias Bethge, Fabian H. Sinz, Xaq Pitkow, and Andreas S. Tolias. Learning

From Brains How to Regularize Machines. *arXiv:1911.05072 [cs, q-bio]*, November 2019. arXiv: 1911.05072.

[55] Scott W Linderman and Samuel J Gershman. Using computational theory to constrain statistical models of neural data. *Current Opinion in Neurobiology*, 46:14–24, October 2017.

[56] Yao Lu, Soren Pirk, Jan Dlabal, Anthony Brohan, Ankita Pasad, Zhao Chen, Vincent Casser, Anelia Angelova, and Ariel Gordon. Taskology: Utilizing task relations at scale. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8700–8709, 2021.

[57] David Marr. *Vision: a computational investigation into the human representation and processing of visual information*. W.H. Freeman, 1982. Book Title: Vision : a computational investigation into the human representation and processing of visual information Place: San Francisco.

[58] Warren S. McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, December 1943.

[59] Ştefan Mihalaş and Ernst Niebur. A Generalized Linear Integrate-and-Fire Neural Model Produces Diverse Spiking Behaviors. *Neural Computation*, 21(3):704–718, March 2009.

[60] Kevin P. Murphy. *Machine learning: a probabilistic perspective*. Adaptive computation and machine learning series. MIT Press, Cambridge, MA, 2012.

[61] A. K. Gupta Nagar, D. K. *Matrix Variate Distributions*. Chapman and Hall/CRC, New York, October 1999.

[62] Aran Nayebi, Daniel Bear, Jonas Kubilius, Kohitij Kar, Surya Ganguli, David Sussillo, James J. DiCarlo, and Daniel L. K. Yamins. Task-Driven Convolutional Recurrent Models of the Visual System. *arXiv:1807.00053 [cs, q-bio]*, June 2018. arXiv: 1807.00053.

[63] Allen Newell. You can't play 20 questions with nature and win: Projective comments on the papers of this symposium. 1973. Publisher: Carnegie Mellon University, Department of Computer Science Pittsburgh, PA.

[64] Liam Paninski. Maximum likelihood estimation of cascade point-process neural encoding models. *Network: Computation in Neural Systems*, 15(4):243–262, January 2004. Publisher: Taylor & Francis _eprint: https://doi.org/10.1088/0954-898X_15_4_002.

[65] Liam Paninski and JP Cunningham. Neural data science: accelerating the experiment-analysis-theory cycle in large-scale neuroscience. *Current Opinion in Neurobiology*, 50:232–241, June 2018.

[66] Stefano Panzeri, Jakob H. Macke, Joachim Gross, and Christoph Kayser. Neural population coding: combining insights from microscopic and mass signals. *Trends in Cognitive Sciences*, 19(3):162–172, March 2015. Publisher: Elsevier.

[67] Adam Paszke and others. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[68] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Édouard Duchesnay. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, 12:2825-2830, October 2011.

[69] Kaare Brandt Petersen, Michael Syskind Pedersen, et al. The matrix cookbook. *Technical University of Denmark*, 7(15):510, 2008.

[70] Ryan Prenger, Michael C. K. Wu, Stephen V. David, and Jack L. Gallant. Nonlinear V1 responses to natural scenes revealed by neural network analysis. *Neural Networks*, 17(5):663–679, June 2004.

[71] Blake A. Richards, Timothy P. Lillicrap, Philippe Beaudoin, Yoshua Bengio, Rafal Bogacz, Amelia Christensen, Claudia Clopath, Rui Ponte Costa, Archy de Berker, Surya Ganguli, Colleen J. Gillon, Danijar Hafner, Adam Kepecs, Nikolaus Kriegeskorte, Peter Latham, Grace W. Lindsay, Kenneth D. Miller, Richard Naud, Christopher C. Pack, Panayiota Poirazi, Pieter Roelfsema, João Sacramento, Andrew Saxe, Benjamin Scellier, Anna C. Schapiro, Walter Senn, Greg Wayne, Daniel Yamins, Friedemann Zenke, Joel Zylberberg, Denis Therien, and Konrad P. Kording. A deep learning framework for neuroscience. *Nature Neuroscience*, 22(11):1761–1770, November 2019. Number: 11 Publisher: Nature Publishing Group.

[72] Mattia Rigotti, Omri Barak, Melissa R. Warden, Xiao-Jing Wang, Nathaniel D. Daw, Earl K. Miller, and Stefano Fusi. The importance of mixed selectivity in complex cognitive tasks. *Nature*, 497(7451):585–590, May 2013.

[73] John Rinzel and GB Ermentrout. Analysis of neural excitability and oscillations. In C Koch and I Segev, editors, *Methods in neuronal modeling*, pages 251–291. MIT Press, Cambridge MA, 1998.

[74] Olivier Roy and Martin Vetterli. The effective rank: A measure of effective dimensionality. In *2007 15th European signal processing conference*, pages 606–610. IEEE, 2007.

[75] Sebastian Ruder. An Overview of Multi-Task Learning in Deep Neural Networks, June 2017.

[76] Andrew M Saxe, James L McClelland, and Surya Ganguli. Dynamics of learning in deep linear neural networks. In *NIPS Workshop on Deep Learning*, 2013.

[77] Rylan Schaeffer, Mikail Khona, and Ila Rani Fiete. No Free Lunch from Deep Learning in Neuroscience: A Case Study through Models of the Entorhinal-Hippocampal Circuit, August 2022. Pages: 2022.08.07.503109 Section: New Results.

[78] Stephanie C Seeman, Luke Campagnola, Pasha A Davoudian, Alex Hoggarth, Travis A Hage, Alice Bosma-Moody, Christopher A Baker, Jung Hoon Lee, Stefan Mihalas, Corinne Teeter, Andrew L Ko, Jeffrey G Ojemann, Ryder P Gwinn, Daniel L Silbergeld, Charles Cobbs, John Phillips, Ed Lein, Gabe Murphy, Christof Koch, Hongkui Zeng, and Tim Jarsky. Sparse recurrent excitatory connectivity in the microcircuit of the adult mouse and human cortex. *eLife*, 7:e37349, September 2018. Publisher: eLife Sciences Publications, Ltd.

[79] Bosiljka Tasic, Vilas Menon, Thuc Nghi Nguyen, Tae Kyung Kim, Tim Jarsky, Zizhen Yao, Boaz Levi, Lucas T. Gray, Staci A. Sorensen, Tim Dolbeare, Darren Bertagnolli, Jeff Goldy, Nadiya Shapovalova, Sheana Parry, Changkyu Lee, Kimberly Smith, Amy Bernard, Linda Madisen, Susan M. Sunkin, Michael Hawrylycz, Christof Koch, and Hongkui Zeng. Adult mouse cortical cell taxonomy revealed by single cell transcriptomics. *Nature Neuroscience*, 19(2):335–346, February 2016.

[80] Bosiljka Tasic, Zizhen Yao, Lucas T. Graybuck, Kimberly A. Smith, Thuc Nghi Nguyen, Darren Bertagnolli, Jeff Goldy, Emma Garren, Michael N. Economo, Sarada Viswanathan, Osnat Penn, Trygve Bakken, Vilas Menon, Jeremy Miller, Olivia Fong, Karla E. Hirokawa, Kanan Lathia, christine Rimorin, Michael Tieu, Rachael Larsen, Tamara casper, Eliza Barkan, Matthew Kroll, Sheana Parry, Nadiya V. Shapovalova, Daniel Hirschstein, Julie Pendergraft, Heather A. Sullivan, Tae Kyung Kim, Aaron Szafer, Nick Dee, Peter Groblewski, ian Wickersham, Ali cetin, Julie A. Harris, Boaz P. Levi, Susan M. Sunkin, Linda Madisen, Tanya L. Daigle, Loren Looger, Amy Bernard, John Phillips, Ed Lein, Michael Hawrylycz, Karel Svoboda, Allan R. Jones, christof Koch, and Hongkui Zeng. Shared and distinct transcriptomic cell types across neocortical areas. *Nature*, 563(7729):72–78, November 2018.

[81] Corinne Teeter et al. Generalized leaky integrate-and-fire models classify multiple neuron types. *Nature Communications*, 9(1), December 2018.

[82] Jinhua Tian, Hailun Xie, Siyuan Hu, and Jia Liu. Multidimensional Face Representation in a Deep Convolutional Neural Network Reveals the Mechanism Underlying AI Racism. *Frontiers in Computational Neuroscience*, 15, 2021.

[83] Vivien Tran-Thien. Censoring the auxiliary loss gradient to mitigate risks of negative transfer., 2020.

[84] Shreejoy J. Tripathy, Lilah Toker, Brenna Li, Cindy-Lee Crichlow, Dmitry Tebaykin, B. Ogan Mancarci, and Paul Pavlidis. Transcriptomic correlates of neuron electrophysiological diversity. *PLOS Computational Biology*, 13(10):e1005814, October 2017.

[85] Ivan Ustyuzhaninov, Santiago A. Cadena, Emmanouil Froudarakis, Paul G. Fahey, Edgar Y. Walker, Erick Cobos, Jacob Reimer, Fabian H. Sinz, Andreas S. Tolias, Matthias Bethge, and Alexander S. Ecker. Rotation-invariant clustering of functional cell types in primary visual cortex. In *International Conference on Learning Representations*, September 2019.

[86] Jonathan W Pillow, Jonathon Shlens, Liam Paninski, Alexander Sher, Alan M Litke, E.J. Chichilnisky, and Eero Simoncelli. Spatio-temporal correlations and visual signaling in a complete neuronal population. *Nature*, 454:995–9, August 2008.

[87] Alison I. Weber and Jonathan W. Pillow. Capturing the Dynamical Repertoire of Single Neurons with Generalized Linear Models. *Neural Computation*, 29(12):3260–3289, December 2017.

[88] Daniel L Yamins, Ha Hong, Charles Cadieu, and James J DiCarlo. Hierarchical Modular Optimization of Convolutional Networks Achieves Representations Similar to Macaque IT and Human Ventral Stream. In *Advances in Neural Information Processing Systems*, volume 26. Curran Associates, Inc., 2013.

[89] Daniel Zdeblick, Eric Shea-Brown, Daniela Witten, and Michael Buice. Data-Driven Discovery of Functional Cell Types that Improve Models of Neural Activity. Vancouver, BC, September 2019.

[90] Daniel N. Zdeblick, Eric T. Shea-Brown, Daniela M. Witten, and Michael A. Buice. Modeling functional cell types in spike train data. *PLOS Computational Biology*, 19(10):e1011509, October 2023. Publisher: Public Library of Science.

[91] Weijian Zong, Horst A. Obenhaus, Emilie R. Skytøen, Hanna Eneqvist, Nienke L. de Jong, Ruben Vale, Marina R. Jorge, May-Britt Moser, and Edvard I. Moser. Large-scale two-photon calcium imaging in freely moving mice. *Cell*, 185(7):1240–1256.e30, March 2022.

[92] James Zou and Londa Schiebinger. AI can be sexist and racist – it's time to make it fair. *Nature*, 559(7714):324–327, July 2018. Publisher: Nature Publishing Group.