Efficient decoding of large-scale neural population responses with Gaussian-process multiclass regression

C. Daniel Greenidge Benjamin Scholl Jacob L. Yates
Jonathan W. Pillow

May 2020

Abstract

Neural decoding methods provide a powerful tool for quantifying the information content of neural population codes and the limits imposed by correlations in neural activity. However, standard decoding methods are prone to overfitting and scale poorly to high-dimensional settings. Here, we introduce a novel decoding method to overcome these limitations. Our approach, the Gaussian process multi-class decoder (GPMD), is well-suited to decoding a continuous low-dimensional variable from high-dimensional population activity, and provides a platform for assessing the importance of correlations in neural population codes. The GPMD is a multinomial logistic regression model with a Gaussian process prior over the decoding weights. The prior includes hyperparameters that govern the smoothness of each neuron's decoding weights, allowing automatic pruning of uninformative neurons during inference. We provide a variational inference method for fitting the GPMD to data, which scales to hundreds or thousands of neurons and performs well even in datasets with more neurons than trials. We apply the GPMD to recordings from primary visual cortex in three different species: monkey, ferret, and mouse. Our decoder achieves state-of-the-art accuracy on all three datasets, and substantially outperforms independent Bayesian decoding, showing that knowledge of the correlation structure is essential for optimal decoding in all three species.

1 Introduction

Since Zohary, Shadlen, and Newsome's landmark demonstration of correlated activity in a population of MT neurons (Zohary et al., 1994), computational neuroscience has been seeking to elucidate the role that correlations play in the population code (Averbeck et al., 2006; Bartolo et al., 2020; Ecker et al., 2011; Kanitscheider et al., 2015; Kohn et al., 2016; Moreno-Bote et al., 2014; Nirenberg & Latham, 2003; Schneidman et al., 2003). A common strategy for evaluating the role that correlations play in a particular population's code is to compare the accuracy of two decoders trained on that population's stimulus-response data: a correlation-blind decoder, and a correlation-aware decoder (Berens et al., 2012; Graf et al., 2011; Nirenberg & Latham, 2003; Stringer et al., 2021). If the correlation-aware decoder performs better, then one may conclude that downstream regions must take correlations into account to optimally read out information from the upstream population code.

This strategy is an effective way to investigate the scientific question, but existing work is plagued by a number of statistical issues, which we aim to address in this paper. First, as neural datasets have increased in dimensionality, regularization has become a prerequisite for good decoding performance, making it difficult to compare correlation-blind decoders—which are often unregularized—and correlation-aware decoders, which are almost always regularized. Second, conventional correlation-aware decoders struggle to scale computationally to modern datasets containing tens or hundreds of thousands of neurons.

To address these shortcomings, we develop a suite of three new decoders with a common regularization strategy based on Gaussian Processes (GPs). First, we introduce two correlation-blind decoders that apply Bayesian decoding to an independent encoding model: the GP Poisson Independent Decoder (GPPID), which assumes independent Poisson encoding noise; and the GP Gaussian Independent Decoder (GPGID), which assumes independent Gaussian encoding noise. Both of these decoders place a Gaussian process prior over the neural tuning curves. (For each neuron, its tuning curve is its mean response as a function of the stimulus variable.) The GPPID model can be used when the neural responses are encoded by non-negative integers (e.g., spike counts), whereas the GPGID model can be used when the neural responses are real numbers (e.g., calcium imaging). We emphasize that both of these decoders are insensitive to correlations in neural activity, because they rely on independence assumptions.

We then introduce a third decoder, which is correlation-aware, the Gaussian Process Multiclass Decoder (GPMD), which is a multinomial logistic regression model that uses a GP prior to regularize its weights. This decoder, which learns a direct linear mapping from high-dimensional neural activity patterns to the log-probability of the stimulus, is the only one of the three that can take into account neural correlations. However, the three decoders have a similar number of parameters—equal to the number of neurons times the number of stimulus categories—and rely on a common regularization method, making it straightforward to compare them.

We compared our decoders to a variety of previously proposed decoding methods: first, multinomial logistic regression regularized using an elastic-net penalty (GLMNET, see Zou and Hastie [2005]); second, the empirical linear decoder (ELD), a decoder trained using support vector machines (Graf et al., 2011); and third, the "super-neuron" decoder (SND), a recently proposed decoder trained using least squares regression and a bank of nonlinear target functions (Stringer et al., 2021). All three of these decoders are linear, correlation-aware classifiers. For completeness, we also compared our decoders to unregularized, correlation-blind Poisson and Gaussian independent decoders (PID/GID).

We benchmarked all these decoders on three real-world datasets from primary visual cortex (V1), recorded from monkey (Graf et al., 2011), ferret, and mouse (Stringer et al., 2021). We found that our regularized correlation-blind decoders (GPPID and GPGID) could match and even exceed the performance of some of the correlation-aware decoders. However, none of these decoders did as well as our our proposed correlation-aware decoder, the GPMD, which achieved state-of-the-art performance on all datasets. These results indicate that knowledge of the correlation structure is crucial for reading out stimulus information from V1 populations in all three species. For ease of use, our decoders conform to the scikit-learn interface and are released as a Python package at https://github.com/cdgreenidge/gdec.

2 The neural decoding problem

In this paper, we consider the problem of a decoding a low-dimensional stimulus variable (i.e., the orientation of a sinusoidal grating) from a high-dimensional neural activity pattern (i.e., a vector of spike counts). We assume the stimulus belongs to one of K discrete bins or classes, formally making this a classification problem. However, the regression problem can be approximated by making K large, so that the grid of stimulus values becomes arbitrarily fine.

Figure 1 illustrates the problem setup for the V1 datasets we examined. The visual stimulus for each individual trial is a drifting sinusoidal grating with an orientation θ_k selected from a set of discrete orientations $\{\theta_1, \dots, \theta_K\}$ that evenly divide the interval $[0, 2\pi]$. The stimulus variable to be decoded is thus a categorical variable $y \in \{1, \dots, K\}$.

We consider the neural population response to be a vector $\mathbf{x} \in \mathbb{R}^D$, where D indicates the number of neurons in the dataset. We obtained this response vector by summing each neuron's spikes (monkey) or two-photon calcium fluorescence (ferret and mouse) over some time window after stimulus presentation. Figure 1B shows orientation tuning curves from three example neurons from each dataset. The monkey datasets (left) contained between $D = 68$ and $D = 147$ neurons, with $K = 72$ discrete stimulus orientations (spaced every 5 degrees), and 50 trials per orientation for $T = 3600$ trials (Graf et al., 2011). The ferret dataset (middle) contained $D = 784$ neurons, with $K = 180$ discrete stimuli (spaced every 2 degrees) and 11 trials per orientation for a total of $T = 1991$ trials, with the $0°/360°$ orientation sampled twice. Finally, the mouse datasets (right) contained between $D = 11311$ and $D = 20616$ neurons. The stimuli for this experiment were sampled uniformly in $[0, 2\pi]$, and we subsequently discretized them into $K = 180$ bins (Stringer et al., 2021). Each bin contained between 12 and 42 trials for a total of between $T = 4282$ and $T = 4469$ trials, depending on the dataset.

In each case, we collected the population response vectors $\mathbf{x}$ and the discretized

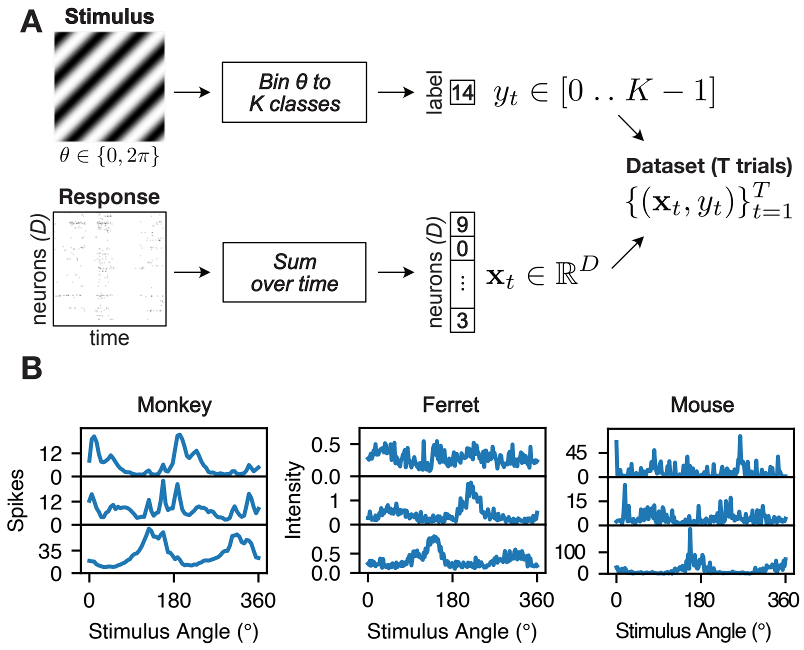

Figure 1: **A**: Decoding task diagram. The animal is presented a grating drifting at an angle $\theta \in [0, 2\pi]$. Responses are recorded from primary visual cortex and summed over time into a feature vector, $\mathbf{x}$. The stimulus is binned to an integer class label, y. We use linear decoders to predict y from $\mathbf{x}$. **B**: Randomly selected tuning curves from each animal. The two calcium datasets (ferret and mouse) are noisier and have many more neurons, increasing the importance of regularization.

stimuli y into a classification dataset $\mathcal{D} = \{(\mathbf{x}_t, y_t)\}_{t=1}^T$. Full details on these datasets and their preprocessing procedures can be found in Appendix C.

The decoders we consider are all linear classifiers, meaning that they are defined by a set of linear decoding weights and an intercept term. Their common form is:

$$\hat{y} = \underset{k \in \{1,\dots,K\}}{\arg\max}\; \mathbf{w}_k^\top \mathbf{x} + b_k, \tag{1}$$

where $\mathbf{w}_k \in \mathbb{R}^D$ is a set of decoding weights, and b_k is an intercept term for stimulus class k. Note that an explicit intercept term is not strictly necessary, since it can be included in the weights if a 1-valued entry is appended to $\mathbf{x}$. To obtain the stimulus estimate $\hat{y}$, we compute the dot product between the neural response vector $\mathbf{x}$ and the weights for each class, and select the class in $\{1, \dots, K\}$ for which this dot product is maximal. The full set of parameters for a decoding model is thus the set of decoding weights for each class, which can be written as a $D \times K$ matrix $W = [\mathbf{w}_1, \dots, \mathbf{w}_K]$, and, optionally, a k-dimensional intercept vector $\mathbf{b} = [b_1, \dots, b_k]^\top$. The only difference between the decoding methods we will consider is the procedure for training these weights from data.

We note that decoding with linear classifiers is optimal for so-called "exponential-family" probabilistic population codes with linear sufficient statistics (Beck et al., 2007; Ma et al., 2006). Although we could certainly expand our study to consider decoding with nonlinear classifiers, previous analyses of two of the V1 datasets we used showed no benefit from adding nonlinear classification (Graf et al., 2011; Stringer et al., 2021).

3 Review of existing decoders

Here we describe previously proposed neural decoding methods, which we will compare to the Gaussian Process based decoding methods we introduce in Section 4.

3.1 Correlation-blind decoders

First, we introduce two independent or "correlation-blind" decoders, the first assuming Poisson noise, and the second assuming Gaussian noise. Both decoders make use of Bayes' rule to obtain a posterior distribution over the stimulus under an independent encoding model, an approach commonly known as "naïve Bayes." The encoding models underlying these decoders assume that neural responses are conditionally independent given the stimulus, making them unable to take correlations into account during decoding.

3.1.1 The Poisson Independent Decoder (PID)

The Poisson independent decoder relies on an independent Poisson encoding model of neural responses, which assumes that each neuron's spike count follows an independent Poisson distribution with its mean determined by the stimulus (Abbott, 1994; Földiák, 1993). The encoding model describes x_d, the response of neuron d, as:

$$P(x_d \mid y = k) = \text{Poiss}(x; \lambda_{dk}) = \frac{1}{x_d!}(\lambda_{dk})^{x_d} e^{-\lambda_{dk}} \tag{2}$$

where λ_{dk} is the mean response of neuron d to stimulus k. Under the conditional independence assumption, the joint distribution of the population response is simply the product of the single-neuron encoding distributions:

$$P(\mathbf{x} \mid y = k) = \prod_{d=1}^{D} \text{Poiss}(x_d; \lambda_{dk}), \tag{3}$$

where D is the total number of neurons.

Bayes' theorem lets us derive the probability over stimuli value given an observed response vector $\mathbf{x}$:

$$P(y = k \mid \mathbf{x}) = \frac{P(\mathbf{x} \mid y = k)P(y = k)}{\sum_{k'=1}^{K} P(\mathbf{x} \mid y = k')P(y = k')}. \tag{4}$$

If there are equal numbers of trials per class in the training dataset, the prior probabilities $P(y = k)$ are equal for all k, and cancel, leaving the prediction rule

$$P(y = k \mid \mathbf{x}) = \frac{\prod_{d=1}^{D} \mathrm{Poiss}(x_d; \lambda_{dk})}{\sum_{k'=1}^{K} \prod_{d=1}^{D} \mathrm{Poiss}(x_d; \lambda_{dk'})}. \tag{5}$$

To fit the model, one has to estimate the parameters $\{\lambda_{dk}\}$ across all the stimuli for each neuron. Collected into a vector $\boldsymbol{\lambda}_d = (\lambda_{d1}, \ldots, \lambda_{dK})^T$, these are known as the tuning curve. The maximum likelihood estimate for λ_{dk} is given by the mean spike count for each neuron-stimulus combination:

$$\hat{\lambda}_{dk} = \frac{1}{|\mathcal{A}_k|} \sum_{(\mathbf{x},y)\in\mathcal{A}_k} x_d \tag{6}$$

where $\mathcal{A}_k = \{(\mathbf{x}_t, y_t) \in \mathcal{D} \mid y_t = k\}$ is the set of all elements of the dataset $\mathcal{D} = \{(\mathbf{x}_n, y_n)\}_{t=1}^{T}$ associated with a particular stimulus $y = k$, and $|\mathcal{A}_k|$ is the number of elements in $\mathcal{A}_k$.

Assuming the prior class probabilities $P(y = k)$ are equal, the log of the class-conditional probability (eq. 5), also known as the log posterior over classes, can be written:

$$\log P(y = k \mid \mathbf{x}) = \sum_{d=1}^{D} x_d(\log \lambda_{dk}) - \sum_{d=1}^{D} \lambda_{dk} + c \tag{7}$$

where c is a constant we ignore because it does not depend on the class. This shows that the PID decoder is a linear classifier (eq. 1), with weights $\hat{W}$ and intercepts $\hat{\mathbf{b}}$ given by

$$\hat{W}_{dk} = \log \hat{\lambda}_{dk} \tag{8}$$

$$\hat{b}_d = -\sum_{k=1}^{K} \hat{\lambda}_{dk}. \tag{9}$$

See Appendix A for a detailed derivation.

3.1.2 The Gaussian Independent Decoder (GID)

The Poisson independent decoder described above can only applied to nonnegative integer data, such as spike counts. For real-valued data such as calcium fluorescence, intracellularly-recorded membrane potential, local field potential, or fMRI

BOLD signals, it is common to use a Gaussian encoding model. This model describes x_d, the response of neuron d, as:

$$P(x_d \mid y = k) = \mathcal{N}(x_d; \mu_{dk}, \sigma_d^2) = \frac{1}{\sqrt{2\pi\sigma_d^2}} \exp\left(-\frac{(x_d - \mu_{dk})^2}{2\sigma_d^2}\right), \tag{10}$$

where μ_{dk} is the mean response of neuron d to stimulus k, and σ_d^2 is the noise variance for neuron d. Unlike a typical Gaussian naïve Bayes decoder, we restrict the noise variance to be constant across stimulus classes, though, as usual, it can vary across neurons. With this restriction, the decoder becomes a linear classifier, like the other decoders we consider. If the noise variance were allowed to vary across stimulus classes, the decoder would be a quadratic classifier (see Appendix A.)

To fit the model, we compute maximum likelihood estimates of the encoding distribution parameters for each neuron, which are given by the class-conditional empirical means $\hat{\mu}_{dk}$, and the empirical variances σ_d^2, for each d-th neuron:

$$\hat{\mu}_{dk} = \frac{1}{|\mathcal{A}_k|} \sum_{(\mathbf{x},y)\in\mathcal{A}_k} x_d \tag{11}$$

$$\hat{\sigma}_d^2 = \frac{1}{|\mathcal{D}|} \sum_{(\mathbf{x},y)\in\mathcal{D}} (x_d - \hat{\mu}_d)^2, \tag{12}$$

As before, $\mathcal{D} = \{(\mathbf{x}_t, y_t)\}_{t=1}^T$ is the dataset, and $\mathcal{A}_k = \{\mathbf{x} \in \mathcal{D} \mid y_t = k\}$ is the set of all neural response vectors for a particular stimulus $y = k$.

Decoding stimuli under this encoding model follows from Bayes' rule in the same manner as in the Poisson independent decoder (eq. 5), but using the Gaussian encoding distribution instead of the Poisson. After some algebra, we can see that the Gaussian independent decoder (GID) is a a linear classifier (eq. 1) with weights W and intercepts $\mathbf{b}$ given by

$$W_{dk} = \frac{\mu_{dk}}{\sigma_d^2} \tag{13}$$

$$b_d = -\sum_{k=1}^{K} \frac{\mu_{dk}^2}{2\sigma_d^2} \tag{14}$$

See Appendix A for a detailed derivation.

3.2 Correlation-aware decoders

Here we review three previously-proposed decoders that take into account the structure of neural correlations when determining a classification boundary, and are therefore "correlation-aware." Unlike the two naïve Bayes decoders described above, which resulted from applying Bayes' rule to an encoding model, these directly model the posterior probability over stimuli given a vector of neural activity. All three decoders are multiclass linear classifiers, but they are trained with different loss functions and regularization methods.

3.2.1 Multinomial logistic regression with an elastic-net penalty (GLM-NET)

The multinomial logistic regression model is a generalization of binary logistic regression to the multiple-class setting. It assumes that the log probability of the stimulus given the response is an affine function—that is, a linear transform plus a constant—of the neural response vector (Bishop, 2006). The conditional probability of the stimulus belonging to class k given the neural response vector $\mathbf{x}$ can be written:

$$P(y = k \mid \mathbf{x}) = \frac{1}{Z} e^{\mathbf{w}_k^\top \mathbf{x} + b_k}, \qquad Z = \sum_{k=1}^{K} e^{\mathbf{w}_k^\top \mathbf{x} + b_k} \tag{15}$$

where $\mathbf{w}_k$ is a vector of the decoding weights for class k, b_k is the constant offset for class k, and the Z is the normalizing constant.

The model parameters consist of the weights $W = (\mathbf{w}_1, \dots, \mathbf{w}_K)$ and offsets $\mathbf{b} = (b_1, \dots, b_K)^\top$, and can be fit by maximum likelihood. The log-likelihood function given the dataset $\mathcal{D} = \{(\mathbf{x}_t, y_t)\}_{t=1}^{T}$ can be written:

$$\mathcal{L}(W, \mathbf{b}) = \log P(\mathcal{D} \mid W, \mathbf{b}) = \sum_{t=1}^{T} \mathbf{y}_t^\top (W^\top \mathbf{x}_t + \mathbf{b}) - \log\left(\mathbf{1}^\top \exp(W^\top \mathbf{x}_t + \mathbf{b})\right), \tag{16}$$

where $\mathbf{y}_t$ is the one-hot vector representation of the stimulus class $y_t \in \{1, \dots, K\}$ on trial t—that is, a vector of all zeros except for a one in the entry corresponding to the stimulus class—and $\mathbf{1}$ is a length-D vector of ones.

The maximum-likelihood estimator (MLE) tends to perform poorly in settings with limited amounts of data, and may not exist for small datasets. In fact, the MLE is not defined when the number of trials T is smaller than the number of identifiable parameters in the weight matrix W, i.e. when $T < D(K-1)$. Even in settings where the MLE does exist, it may overfit, yielding poor generalization performance.

A popular solution to this problem is to regularize the MLE with the elastic-net penalty, which combines ℓ_1 ("lasso") and ℓ_2 ("ridge") penalties to induce parameter sparsity and shrinkage (Friedman et al., 2010). The elastic-net estimator is obtained by maximizing the log-likelihood minus the regularization penalty:

$$(\hat{W}, \hat{\mathbf{b}}) = \underset{W, \mathbf{b}}{\arg\max}\ \mathcal{L}(W, \mathbf{b}) - \gamma_1\left(\gamma_2 ||W||_{\ell_1} + (1 - \gamma_2)||W||_{\ell_2}^2\right), \tag{17}$$

Here, γ_1 is a hyperparameter determining the strength of the regularization, and γ_2 is a hyperparameter that controls the balance between the ℓ_1 penalty, which encourages W to be sparse, and the ℓ_2 penalty, which encourages W to have a small magnitude. For our decoding tasks, we found that including the ℓ_1 penalty always diminished cross-validated performance, so we fixed $\gamma_2 = 0$. We then set γ_1 using a five-step logarithmic grid search from 10^{-4} to 10, evaluated with three-fold cross validation.

3.2.2 The Empirical Linear Decoder (ELD)

The Empirical Linear Decoder (ELD), introduced by Graf et al. (2011), is similar to multinomial logistic regression in that it models the log probability of the stimulus class as an affine function of the neural response vector (eq. 15). However, instead of using standard likelihood-based methods to fit the model, the authors constructed an inference method based on support vector machines (SVMs).

Their key observation was that the log-likelihood ratio for adjacent stimulus classes is an affine function of the response vector, with weights given by the difference of the two classes' decoding weight vectors. For example, for stimulus classes one and two we have:

$$\log \frac{P(y=2|\mathbf{x})}{P(y=1|\mathbf{x})} = \log \frac{e^{\mathbf{w}_2^\top \mathbf{x} + b_2}}{e^{\mathbf{w}_1^\top \mathbf{x} + b_1}} = (\mathbf{w}_2 - \mathbf{w}_1)^\top \mathbf{x} + (b_2 - b_1) \triangleq \mathbf{v}_2^\top \mathbf{x} + c_2 \tag{18}$$

Here, we have defined $\mathbf{v}_2$ to be the difference vector $(\mathbf{w}_2 - \mathbf{w}_1)$ and c_2 to be the difference scalar $b_2 - b_1$.

We see that discriminating class two from class one under the multinomial logistic regression model is equivalent to solving a linear binary classification task with weights $\mathbf{v}_2$ and offset c_2. The authors proposed estimating $\mathbf{v}_2$ and c_2 using an SVM trained on the data from classes one and two. They then used the same approach to estimate the weights for all subsequent pairs of adjacent classes. That is, they estimated the difference weights $\mathbf{v}_{k+1}$ using an SVM trained on data from classes $k-1$ and k, for $k = 2, \dots, K$.

To recover the weights of the multinomial logistic regression model from the SVM weights, Graf et al. (2011) used the recursions:

$$\begin{aligned} \hat{\mathbf{w}}_1 &= \mathbf{0} \\ \hat{\mathbf{w}}_k &= \hat{\mathbf{w}}_{k-1} + \alpha_k \hat{\mathbf{v}}_k \end{aligned} \tag{19}$$

and

$$\begin{aligned} \hat{b}_1 &= 0 \\ \hat{b}_k &= \hat{b}_{k-1} + \alpha_k \hat{c}_k \end{aligned} \tag{20}$$

for $k = 2, \dots, K$. Here the weights for class one can be set to zero without loss of generality. The constant α_k, which scales the contribution of the SVM weights $\mathbf{v}_k$ and c_k, is necessary because SVMs only recover $\mathbf{v}_k$ and c_k up to a multiplicative constant. We were unable to determine how the authors set these scaling constants, so we fit them by maximizing the log likelihood of the data under the multinomial logistic regression model (eq. 16).

3.2.3 The Super Neuron Decoder (SND)

The Super Neuron Decoder (SND), introduced by Stringer et al. (2021), is a third approach for training a linear classifier on multi-class data. It optimizes a set of decoding weights using penalized least-squares regression and a set of nonlinear super-neuron response functions. The super-neuron response functions encode the tuning curves of a population of narrowly-selective downstream "super-neurons", containing one super-neuron for each stimulus orientation. Each super-neuron responds maximally to a single orientation, making the population response on each trial a narrow bump of activity centered on the correct stimulus.

Formally, the SND seeks a matrix of weights W that maps the response vector $\mathbf{x}$ to a target vector $\mathbf{h} \in \mathbb{R}^K$ on each trial. The target vector $\mathbf{h}$ contains the responses of the super-neuron population. The super-neurons have tuning curves parameterized by the von Mises probability density function, which is appropriate since the stimulus variable is periodic.

The i-th super-neuron has a preferred orientation of $\theta_i = \frac{2\pi}{K}(i-1)$, so its tuning curve is given by:

$$f_i(\theta) = \exp\left(\frac{\cos(\theta_i - \theta) - 1}{0.1}\right) \tag{21}$$

The target vector for the k-th stimulus class is therefore

$$\mathbf{h}_k = \left(f_1(\theta_k), \dots, f_K(\theta_k)\right)^\top \tag{22}$$

where $\theta_k = \frac{2\pi}{K}(k-1)$ is the stimulus angle associated with stimulus class $k \in \{1, \dots, K\}$.

Stringer et al. (2021) trained the model weights W by linearly regressing the observed neural responses onto the target vectors. To penalize large weight values, they included an ℓ_2 ("ridge") regularization:

$$\hat{W} = \arg\min_W \left(\sum_{t=1}^{\top} ||\mathbf{h}_{y_t} - W^\top \mathbf{x}_t||^2\right) + \gamma ||W||^2_{\ell_2}, \tag{23}$$

The term $||\mathbf{h}_{y_t} - W^\top \mathbf{x}_t||^2$ is the squared error between the correct target vector $\mathbf{h}_{y_t}$ and the output of the linear decoding weights $W^\top \mathbf{x}_t$ on trial t. The term $\gamma||W||^2_{\ell_2}$ is the squared ℓ_2 penalty on the decoding weights with regularization strength γ, which the authors fixed at $\gamma = 1.0$. Intuitively, this training procedure seeks weights W that make the linearly transformed population response $W^\top \mathbf{x}$ match the super-neuron population response $\mathbf{h}$ as closely as possible in a least-squares sense.

The decoding rule chooses the class label corresponding to the maximum of the linearly weighted responses. (This is the same decoding rule as in the other decoders we have considered):

$$\hat{y} = \arg\max_{k \in \{1, \dots, K\}} (W^\top \mathbf{x})_k, \tag{24}$$

Here $(W^\top \mathbf{x})_k$ is the k-th element of the transformed response vector $W^\top \mathbf{x}$. In other words, the predicted stimuli value is the preferred orientation of the maximally responding super-neuron.

4 Proposed methods: GP-regularized decoders

In this section, we first introduce two correlation-blind decoders regularized with Gaussian process (GP) priors: the GP-regularized Gaussian Independent Decoder (GPGID) and the GP-regularized Poisson Independent Decoder (GPPID). Like the GID and PID, these decoders use independent Gaussian and Poisson encoding models, but they also add GP priors to induce smoothness in the neural tuning curve estimates. Next, we introduce a correlation-aware decoder, the Gaussian Process Multiclass Decoder (GPMD), which adds a GP prior to multinomial logistic regression for the same purpose.

4.1 The GP-regularized Poisson Independent Decoder (GPPID)

When doing inference in the PID decoder (section 3.1.1), it is necessary to estimate each $\hat{\lambda}_{dk}$, the mean spike count for the d-th neuron and the k-th stimulus. The mean spike counts for every stimulus form a tuning curve for each neuron $\boldsymbol{\lambda}_d = (\lambda_{d1}, \dots, \lambda_{dK})^T$. The maximum likelihood estimator for each entry in the tuning curve is simply the empirical mean of the spike counts for the d-th neuron under the k-th stimulus. However, the empirical mean estimates are noisy, especially when the number of trials for each stimulus is small, which can limit the PID decoder's performance.

In principle, we could compensate for the noise by recording more trials for each stimulus, but this is expensive, particularly if the stimulus grid has a fine resolution. Instead, we propose to reduce error in the tuning curve estimates by exploiting our prior knowledge that tuning curves tend to be smooth with respect to orientation. We incorporate this knowledge into the model by placing an independent Gaussian process prior over the log tuning curve of each neuron (Park et al., 2014; Rad & Paninski, 2010).

The resulting GP-regularized PID model is given by

$$\log \boldsymbol{\lambda}_d \sim GP(\mathbf{0}, \kappa_{\theta_d}) \tag{25}$$

$$x_d \mid y = k \sim \text{Poiss}(\lambda_d[k]) \tag{26}$$

where x_d is the spike response of neuron d, and $\lambda_d[k]$ is the kth element of the tuning curve $\boldsymbol{\lambda}_d$. Here, the log tuning curve has a Gaussian Process prior with zero mean, and a covariance function $\kappa(\cdot,\cdot)$ with hyperparameters θ_d.

We choose κ to be the radial basis (RBF) or "Gaussian" covariance function:

$$\kappa(j,k) = \rho_d^2 \exp\left(-\frac{d(j,k)^2}{2\ell_d^2}\right) \tag{27}$$

where $d(j,k)$ denotes the distance between stimulus classes j and k, and the hyperparameters $\theta_d = \{\rho_d, \ell_d\}$ are the marginal variance ρ_d and length scale ℓ_d of the log tuning curve for neuron d. Because our stimuli lie on a circle with circumference K, the number of classes, we choose d to be the circular distance function

$$d(x, y) = \left| \left((x - y - \frac{K}{2}) \bmod K \right) - \frac{K}{2} \right| \tag{28}$$

We will write the Gaussian process kernel matrix using the overloaded notation $\kappa_\theta(\mathbf{u}, \mathbf{v})$, where $\mathbf{u}$ and $\mathbf{v}$ are arbitrary vectors in $\mathbb{R}^n$ and $\mathbb{R}^m$, respectively. This notation denotes an $n \times m$ kernel matrix, with elements given by $\kappa_\theta(\mathbf{u}, \mathbf{v})_{ij} = \kappa_\theta(u_i, v_j)$. An important instance of the kernel matrix is the kernel matrix of the tuning curve. Because of the kernel function's lengthscale parameter, the distance between each stimuli value can be rescaled arbitrarily, so we define the tuning curve kernel matrix to be $\kappa_\theta(\mathbf{r}, \mathbf{r})$, where $\mathbf{r} = (1, \dots, K)^\top$.

Note that the neuron-specific hyperparameters θ_d permit tuning curves to differ in amplitude and smoothness, so different neurons can be regularized differently. For neurons with non-existent tuning or exceptionally noisy responses, the inference procedure will set the amplitude ρ_d to zero or the length scale ℓ_d to infinity, making the tuning curve flat (see fig. 5). Such neurons are effectively pruned from the dataset, since flat tuning curves make no contribution to decoding. This effect is known as automatic relevance determination (MacKay, 1992; Neal, 1996), and it eliminates the need to manually filter out noisy or untuned neurons. Automatic preprocessing in this manner is critical when working with large datasets.

To fit the GPPID model to spike count data, we employ a two-step procedure known as empirical Bayes (Bishop, 2006). For each neuron, we first compute a point estimate of the tuning curve hyperparameters by maximizing the model evidence, and then find the *maximum-a-posteriori* (MAP) estimate of the tuning curve using the previously estimated hyperparameters. The model evidence for neuron d is the marginal probability of $\mathbf{X}_{*d}$, the d-th column of the spike count matrix $\mathbf{X} = (\mathbf{x}_1, \dots, \mathbf{x}_T)^\top$, given the hyperparameters:

$$p(\mathbf{X}_{*d} \mid \theta_d) = \log \int p(\mathbf{X}_{*d} \mid \boldsymbol{\lambda}_d, \theta_d) p(\boldsymbol{\lambda}_d \mid \theta_d)\, d\boldsymbol{\lambda}_d \tag{29}$$

$$= \int \exp\left(\log \mathcal{N}(\log \boldsymbol{\lambda}_d; \mathbf{0}, \kappa_\theta(\mathbf{r}, \mathbf{r})) + \sum_{t=1}^{T} \log \operatorname{Poiss}(\lambda_d[y_t]) \right) d\boldsymbol{\lambda}_d. \tag{30}$$

This integral is intractable, so we approximate it using a Laplace's method (Bishop, 2006). We define $h(\boldsymbol{\lambda}_d) = \log \mathcal{N}(\log \boldsymbol{\lambda}_d; \mathbf{0}, \kappa_\theta(\mathbf{r}, \mathbf{r})) + \sum_{n=1}^{N} \log \operatorname{Poiss}(\lambda_d[y_n])$ to be the sum of log-prior and log-likelihood, and H to be the Hessian matrix of h evaluated at its maximizer, the MAP estimate $\boldsymbol{\lambda}_d^*$. The integral's approximate value is then given by:

$$\log p(\mathbf{X}_{*d} \mid \theta_d) \approx -\frac{n}{2} \log(2\pi) - \frac{1}{2} \log |H| - h(\boldsymbol{\lambda}_d^*). \tag{31}$$

We compute point estimates of the model hyperparameters $\hat{\theta}_d$ by optimizing this approximation with respect to θ_d using the Nelder-Mead algorithm (Nelder & Mead, 1965).

Once we have estimated the hyperparameters $\hat{\theta}_d$ for neuron d, we estimate the neuron's tuning curve by computing the MAP estimate of $\boldsymbol{\lambda}_d$ under the model given by equation 25:

$$\hat{\boldsymbol{\lambda}}_d = \arg\max_{\lambda_d} \left(\log \mathcal{N}(\log \boldsymbol{\lambda}_d; \mathbf{0}, \kappa_\theta(\mathbf{r}, \mathbf{r})) + \sum_{t=1}^{T} \log \text{Poiss}(x_t \mid \lambda_d[y_t]) \right) \tag{32}$$

where $\text{Poiss}(x_t \mid \lambda_d[y_t])$ is the probability of spike count x_t given the firing rate $\lambda_d[y_t]$, under a Poisson distribution (eq. 2).

To accelerate the optimization procedure, we use a Fourier-domain representation of the covariance function based on the Karhunen-Loéve expansion. (See Appendix B for details.) Since the procedure described above can be performed independently for each neuron, fitting the GPPID model is fully parallelizeable across neurons.

4.2 The GP-regularized Gaussian Independent Decoder (GPGID)

To perform inference in the GID (section 3.1.2), it is necessary to estimate the class-conditional mean activity μ_{kd} and the variance σ_d^2 for each neuron. For each d-th neuron, the mean activities form a tuning curve, the vector $\boldsymbol{\mu}_d = (\mu_{1d}, \ldots, \mu_{Kd})^T$. We estimate the tuning curve using the same approach as the PID (section 4.1), but with a Gaussian likelihood instead of a Poisson likelihood. The model is as follows:

$$\boldsymbol{\mu}_d \sim GP(\mathbf{0}, \kappa_{\theta_d}) \tag{33}$$

$$\mathbf{x}_d \mid y = k \sim \mathcal{N}(\mu_d[k]), \sigma_d^2), \tag{34}$$

Note that each neuron has three hyperparameters: the GP prior's marginal variance ρ_d and length scale ℓ_d, and the likelihood's observation noise variance σ_d^2. As in the GID's model (section 3.1.2), we assume that the observation noise variance is constant over classes. This restriction ensures that the classification boundary is linear (see Appendix A).

To fit the model, we use empirical Bayes, as we did for inference in the GPPID. The first step is to compute a point estimate of each neuron's hyperparameters by maximizing the model evidence. Before proceeding, we need to introduce some notation: the vector $\boldsymbol{\mu}_d[\mathbf{y}]$ is the vector whose t-th element is given by $(\boldsymbol{\mu}_d)_t = (\boldsymbol{\mu}_d)_{y_t}$. Intuitively, this vector contains the values from the d-th neuron's tuning curve corresponding to each stimulus in the dataset. With this notation, the evidence maximization can be written as:

$$\hat{\theta}_d, \hat{\sigma}_d^2 = \underset{\theta_d, \sigma_d^2}{\arg\max}\ \log \int p(\mathbf{X}_{*d} \mid \boldsymbol{\mu}_d, \theta_d) p(\boldsymbol{\mu}_d \mid \theta_d)\, d\boldsymbol{\mu}_d \tag{35}$$

$$= \underset{\theta_d, \sigma_d^2}{\arg\max} \int \mathcal{N}(\boldsymbol{\mu}_d[\mathbf{y}]; \mathbf{0}, \kappa_\theta(\mathbf{y}, \mathbf{y})) \prod_{t=1}^{T} \mathcal{N}(x_t[d]); \mu_d[y_t], \sigma_d^2)\, d\boldsymbol{\mu}_d \tag{36}$$

$$= \underset{\theta_d, \sigma_d^2}{\arg\max}\ -\frac{1}{2}\mathbf{X}_{*d}^\top (\kappa_\theta(\mathbf{y}, \mathbf{y}) + \sigma_d^2 I)^{-1} \mathbf{X}_{*d} - \frac{1}{2} \log |\kappa_\theta(\mathbf{y}, \mathbf{y}) + \sigma_d^2 I| - \frac{T}{2} \log 2\pi \tag{37}$$

Since the objective function can be expressed analytically, we perform the maximization using a trust-region Newton method.

Then, for each neuron, we compute the MAP estimate of the tuning curve:

$$\hat{\boldsymbol{\mu}}_d = \underset{\mu_d}{\arg\max} \left(\log \mathcal{N}(\boldsymbol{\mu}_d[\mathbf{y}]; \mathbf{0}, \kappa_\theta(\mathbf{y}, \mathbf{y})) + \sum_{t=1}^{T} \log \mathcal{N}(x_t[d]; \mu_d[y_t], \sigma_d^2) \right) \tag{38}$$

The solution to this problem can be expressed analytically (Rasmussen & Williams, 2006):

$$\hat{\boldsymbol{\mu}}_d = \kappa_{\theta_d}(\mathbf{r}, \mathbf{y})[\kappa_{\theta_d}(\mathbf{y}, \mathbf{y}) + \sigma_d^2 I]^{-1} \mathbf{X}_{*d} \tag{39}$$

However, for scalability, we use an equivalent procedure leveraging the same spectral weight representation as in the GPPID. (See Appendix B for details.)

4.3 The Gaussian Process Multiclass Decoder (GPMD)

In this section, we introduce the the Gaussian Process Multiclass Decoder (GPMD), which is multinomial logistic regression with a GP prior placed over the weights for each neuron (see fig. 2). As in section 3.2.1, the multinomial logistic regression model can be written:

$$P(y = k \mid \mathbf{x}) = \frac{1}{Z} e^{\mathbf{w}_k^\top \mathbf{x} + b_k}, \qquad Z = \sum_{k=1}^{K} e^{\mathbf{w}_k^\top \mathbf{x} + b_k} \tag{40}$$

where $\mathbf{w}_k$ is a vector containing the decoding weights for class k, b_k is the offset for class k, and Z is the normalizing constant.

We regularize the weight matrix by placing an independent zero-mean Gaussian Process prior on each of its rows:

$$W_{d*} \sim GP(\mathbf{0}, \kappa_{\theta_d}), \tag{41}$$

Here W_{d*} is the d-th row of W, which contains the decoding weights associated with neuron d across all stimuli, and κ_{θ_d} is the RBF covariance function defined in equation 27.

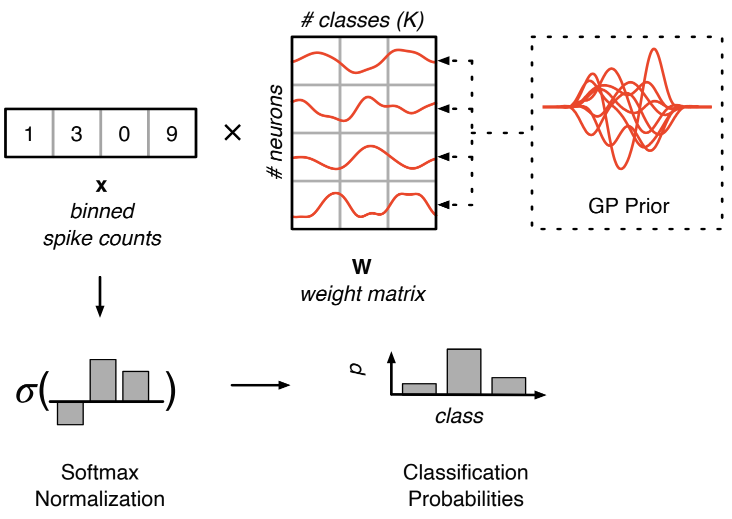

Figure 2: The GPMD model. A smoothing Gaussian Process prior is applied to each row of a logistic regression weight matrix **W** to shrink the weights and to ensure that they vary smoothly across stimuli, mimicking the smoothness of neural tuning curves. The weight matrix is then linearly combined with the neuron's response vector **x** to produce unnormalized scores for each decoding class. A softmax function is used to transform the unnormalized scores into probabilities.

In the GPPID and GPGID, we used a GP prior to formalize our prior knowledge that neuron's tuning curves tend to be smooth. In the GPMD, we cannot apply that prior knowledge directly, since the GPMD's weights have no direct interpretation in terms of tuning curves. Nonetheless, we can motivate our application of GP prior with the following observation: since orientation is a continuous variable, the decoding weights ought to vary smoothly as a function of orientation.

Like previous decoders, the GPMD has neuron-specific hyperparameters, which allow different neurons to have decoding weights with different amplitudes and different amounts of smoothness. This flexibility has two benefits: first, it allows each neuron's weights to adapt to the neuron's response properties, and second, it automatically discards untuned or noisy neurons as described in section 4.1, eliminating the need for manual dataset preprocessing.

To fit the GPMD, we use variational inference to simultaneously learn both a posterior estimate for the weights $\{W, \mathbf{b}\}$ and point estimates for the prior hyperparameters. Specifically, given an approximate posterior family q—which we choose to be mean-field Gaussian—indexed by parameters ϕ and prior hyperparameters θ, we maximize the evidence lower bound (ELBO, see Blei et al. [2017] and Hoffman et al.

[2013]) jointly with respect to ϕ and θ:

$$\hat{\phi}, \hat{\theta} = \underset{q_\phi, \theta}{\arg\max}\, \mathbb{E}_q[\log p(\mathcal{D} \mid W, \mathbf{b})] - D_{\mathrm{KL}}(q \parallel p) \tag{42}$$

To calculate the likelihood term, we draw $M = 3$ samples from the variational posterior $\{(W^{(m)}, \mathbf{b}^{(m)})\}_{m=1}^{M} \sim q$, and use these to compute a Monte Carlo approximation of the expectation:

$$\log p(\mathcal{D} \mid W, \mathbf{b}) \approx \sum_{m=1}^{M} \sum_{t=1}^{T} \frac{1}{Z} e^{\langle \mathbf{w}_{y_t}^{(m)}, \mathbf{x}_t \rangle + b_{y_t}^{(m)}} \tag{43}$$

where Z is the normalizing constant defined in Eq. 40. In principle, we could also calculate this approximation using a subset of the data, or "minibatch" (Hoffman et al., 2013), but our datasets are small enough that this is not necessary, and doing so increases the approximation variance.

The KL-divergence term expands to a sum of KL divergences for each row of the weight matrix, since each row of the matrix is independent of the others. Assuming $q_1, \ldots, q_K$ are the mean-field variational distributions for each row of the weight matrix, and $p_1, \ldots, p_K$ are the associated GP priors, the KL-divergence term reduces to

$$D_{\mathrm{KL}}(q \parallel p) = \sum_{k=1}^{K} D_{\mathrm{KL}}(q_k \parallel p_k) \tag{44}$$

Each of the summands is easy to calculate analytically, since q_k and p_k are both multivariate normal distributions. The approximate variational posterior q_k is a diagonal normal distribution, and the GP prior for each column p_k is the multivariate normal distribution $\mathcal{N}(\mathbf{0}, \kappa_{\theta_k}(\mathbf{r}, \mathbf{r}))$, where, as in the GPPID model, $\mathbf{r} = (0, 1, \ldots, K)^\top$ defines the grid of class labels.

To make predictions, we approximate MAP inference by using the mode of the posterior approximation q as a point estimate of the weights W and $\mathbf{b}$ in the multinomial logistic regression model (eq. 40). Since we are only interested in the effects of the prior, and not in using the posterior to assess prediction uncertainty, this approach suffices.

4.3.1 Scaling GPMD inference

When maximizing the ELBO, there are two scaling concerns: the number of examples (T) and the number of neurons (D). It is well-known that the ELBO can be scaled to huge numbers of examples by estimating the likelihood term using a minibatch approximation (Hoffman et al., 2013). However, even when using a minibatch approximation, the KL-divergence term must be calculated at each gradient step, and from it costs $\sim DK^3$ to evaluate (eq. 44). For large values of D, which are expected in high-dimensional neural datasets, the KL-divergence term evaluation makes stochastic gradient descent far too slow.

We solve this problem by representing $W^\top$ using a basis $\Psi \in \mathbb{C}^{K\times M}$, i.e. $W^\top = \Psi U$. Then, we place an independent normal prior on each entry of U. This allows the KL-divergence term to be evaluated with $\sim DM$ complexity, since it becomes a KL divergence between two diagonal normal distributions. The only difficulty is choosing Ψ such that W turns out to have the desired Gaussian process distribution when U is a standard normal.

It can be shown that the appropriate choice of Ψ is the unitary Fourier basis (see Appendix B). With this basis, the entry U_{id}, the element in the d-th column of U corresponding to Fourier frequency i, must satisfy two conditions. The first condition is conjugacy, $U_{id} = U^*_{-id}$, which ensures that W is real. The second condition is on the distribution, which must be a zero-mean complex normal with variance

$$\mathbb{E}[U_{id}U^*_{id}] = \frac{1}{\sqrt{M}}\mathcal{F}[\kappa_{\theta_d}]\left(\frac{i}{2\pi}\right) \tag{45}$$

This spectral formulation assumes the stimuli lie on $[0, 2\pi]$, but it can be trivially extended to any domain.

5 Results

5.1 Evaluation and performance

We benchmarked each decoder by calculating its mean absolute test error on the monkey (Graf et al., 2011), ferret (this paper), and mouse (Stringer et al., 2021) datasets, using five-fold cross-validation repeated ten times (fig. 3). We examined five monkey datasets, one ferret dataset, and three mouse datasets. Figure 3A reports the average scores for each animal; separate scores are reported in supplementary figure 9. Note that the GID decoder has two variants: the standard formulation, which has a quadratic decision boundary, and the formulation described in section 3.1.2, which has a linear decision boundary.

The rank ordering of the models remained largely consistent across datasets. In general, the correlation-blind decoders (the PID and GID) performed worse than the correlation-aware decoders, which is consistent with previous decoding studies (Graf et al., 2011; Stringer et al., 2021). Their regularized variants (the GPPID and GPGID) performed better, but still did not match the performance of the best correlation-aware decoders.

The GPMD set or matched state-of-the-art performance on all datasets. An important advantage of its Bayesian approach is that hyperparameters are learned automatically, which allowed the GPMD to adapt to the conditions present in different datasets. By contrast, models that set hyperparameters manually exhibited occasional poor performance—for example, the SND's scores on the monkey datasets.

These results could have been be obscured by the choice of error metric. For example, repeating the same benchmark using "proportion correct" instead of mean

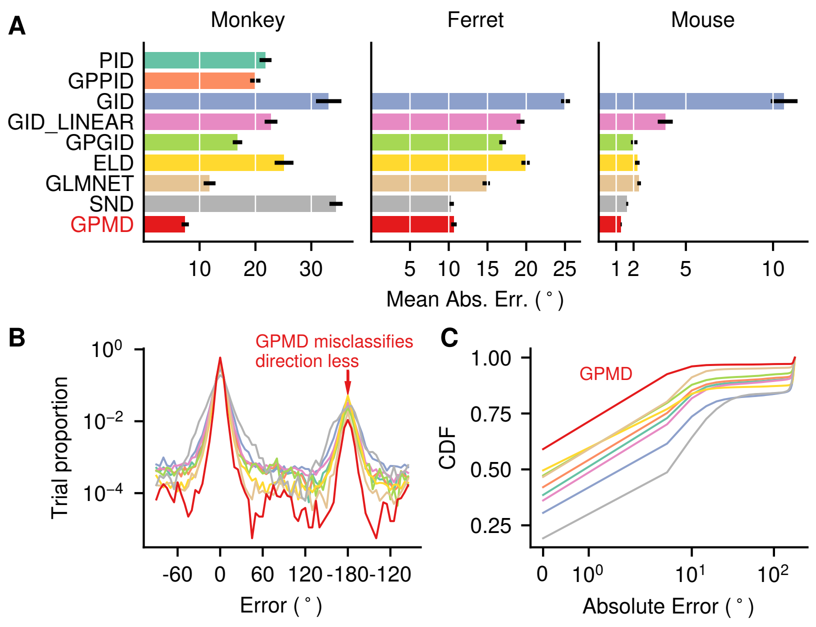

Figure 3: **A**: Test error on monkey (Graf et al., 2011), ferret, and mouse (Stringer et al., 2021) datasets, estimated using five-fold cross-validation repeated ten times. Since there are five monkey and three mouse datasets, we trained the models on each dataset separately and averaged the scores. We did not train the PID and GP-PID decoders on the ferret and mouse datasets, which contain real-valued calcium data, because they assume integer features. Error bars represent the standard error of the mean. **B**: Error distributions on the third monkey dataset. The GPMD has more mass concentrated around 0° error (correct classification) than the other decoders. The most common source of error appears to be misclassifying direction, a 180° error; the GPMD does this less than the other decoders. **C**: Empirical CDF of errors on the third monkey dataset. The GPMD classifies higher fractions of the dataset at lower errors than the other decoders, demonstrating its superior performance.

absolute error improved the performance of the ELD substantially (see supplementary figure 8), qualitatively replicating the results of Graf et al. (2011). To ensure that our results were not artifacts of the error metric, we used the empirical error cumulative distribution function to characterize each decoder's errors in more detail (fig. 3B). Good decoders should classify higher fractions of the dataset at lower errors, producing curves that lie up and to the left. We found that the GPMD outperformed or matched all the other decoders on all the datasets (see supplementary figure 10).

Our results show that both regularization and exploiting correlations improved decoding performance substantially. The regularized correlation-blind decoders, the

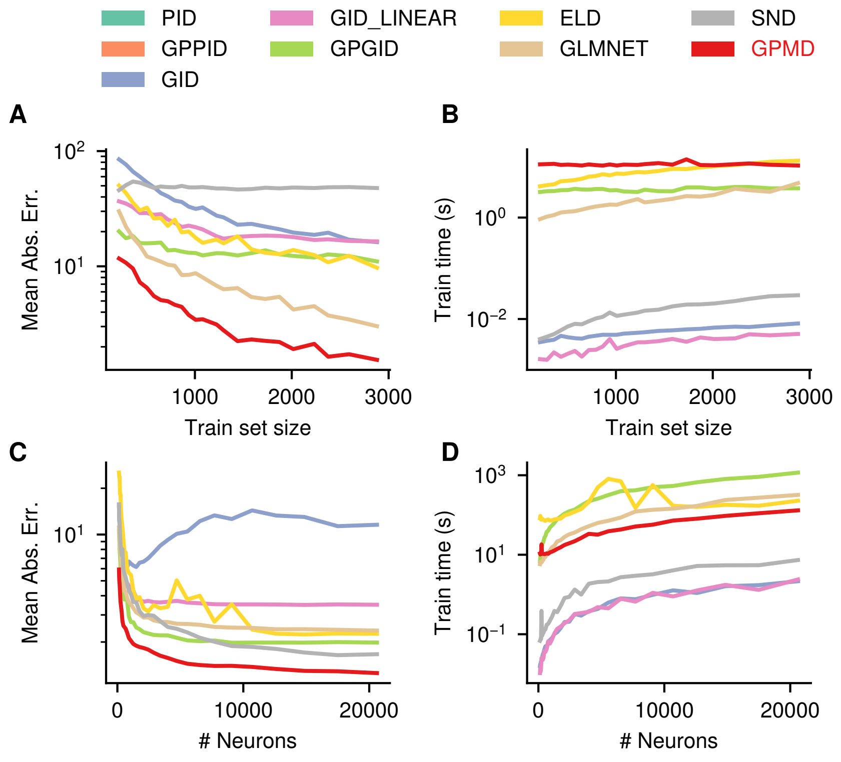

Figure 4: **A**: Cross-validated model performance with varying amounts of training data. We chose random subsets of the third monkey dataset. The GPMD continues to benefit from increasing training data, and does exhibit asymptotic behavior like most of the other models. **B**: Training times for the ablation study in (A). Like all the models, the GPMD shows essentially constant training times. This is because the monkey dataset is small enough that training cost is dominated by constant factors. **C**: Cross-validated model performance with varying amounts of neurons (features). We chose random subsets of the first mouse dataset. The GPMD's careful regularization avoids undesirable double-descent characteristics while achieving state-of-the-art performance for all feature sizes. **D**: Training times for the ablation study in (C). The GPMD is nearly two orders of magnitude faster than logistic regression, a less sophisticated model, and trains in a few minutes on the largest datasets.

GPPID and GPGID, outperformed their unregularized analogues, the GID and PID. The GLMNET decoder, which is correlation aware, outperformed the correlation-blind GPPID and GPGID. Finally, the SND and GPMD, which are both regularized and correlation-aware, outperformed all other decoders.

The relative impact of these strategies depended on dataset dimensionality. For small datasets, such as the monkey dataset with $\sim$ 150 neurons, both regulariza-

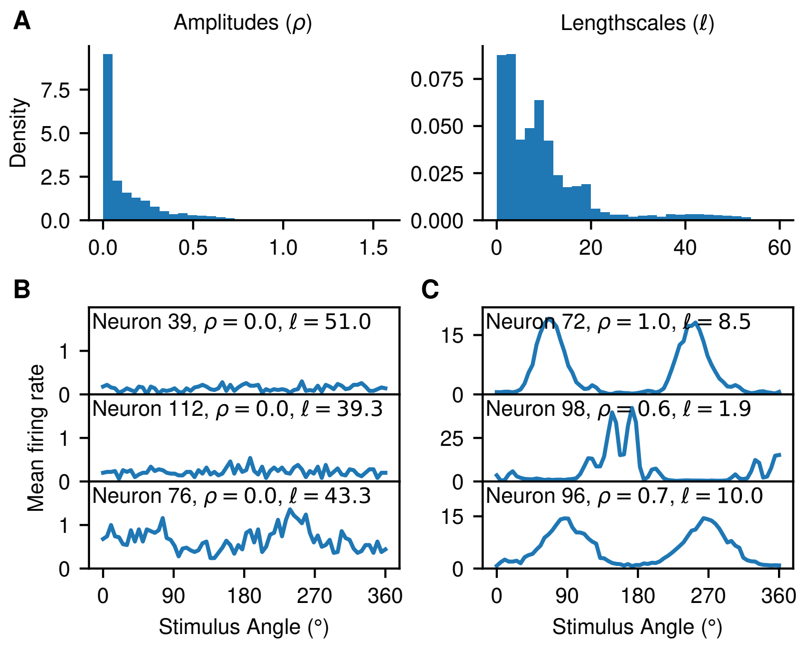

Figure 5: **A**: GPMD hyperparameter distributions on the third monkey dataset. The distribution of amplitudes has a peak near zero, and the distribution of lengthscales has a second mode near 45, indicating an automatic relevance determination (ARD) effect. **B**: The ARD-eliminated neurons have amplitudes near zero and, often, long lengthscales. **C**: The neurons that were not eliminated have positive amplitudes and much shorter lengthscales. Some have simple "Gaussian bump" tuning curves, whereas others have more complex tuning characteristics.

tion and exploiting correlations had a substantial effect. For example, adding regularization to the GID (using the GPGID) decreased its mean absolute error by 16.3 degrees, and exploiting correlations (using the GPMD), decreased error by another 9.4 degrees. For high-dimensional datasets where it was easy to overfit, such as the mouse dataset, which was recorded from $\sim 20,000$ neurons, regularization became the most important strategy. In fact, on the mouse dataset, the regularized correlation-blind GPGID did just as well as some of the correlation-aware decoders.

To characterize the GPMD's performance and training times with respect to dataset size and dimensionality, we performed ablation studies on both the number of training examples and the number of neural features (fig. 4).

The GPMD performed well at all training set sizes (fig. 4A) implying that its inductive biases were well-calibrated—that is, strong enough to permit good performance with few training examples, but flexible enough to allow continued learning with many training examples. We believe the good calibration is due to the flexibil-

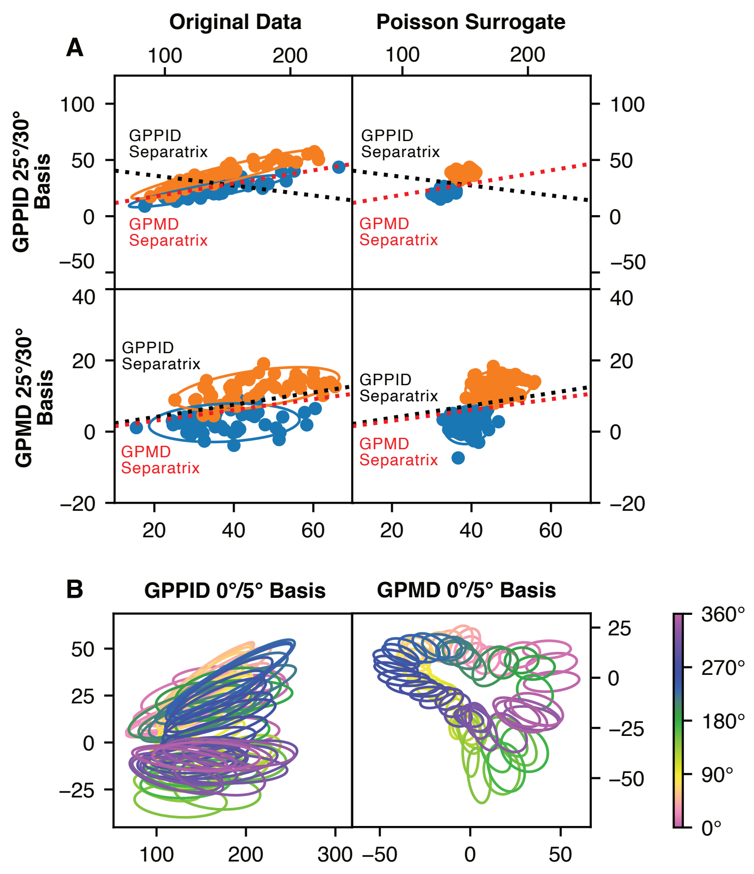

Figure 6: **A**: To investigate the impact of correlations on decoding performance, we projected the 25° and 30° data classes to two dimensions using an orthogonalized bases derived from the 25° and 30° decoding vectors from the correlation-blind GPPID and correlation-aware GPMD. The data displayed significant correlations in the GPPID basis, making the GPPID separatrix a poor decision boundary. The GPPID separatrix only performed well when correlations were removed from the dataset using a Poisson surrogate model. In the GPMD basis, the difference between the correlated and Poisson surrogate datasets was much less pronounced, indicating that the GPMD's projection decorrelated the data somewhat. **B**: Class correlation ellipses plotted using the GPPID and GPMD bases derived from zero- and five-degree decoding vectors. The GPMD basis produced far superior class separation.

ity of the GP prior, which learns the structure present in the neural dataset. Models with stronger inductive biases, such as the GPGID, which assumes independence, or the SND, which has many hard-coded parameters, had difficulty learning from increasing numbers of training examples.

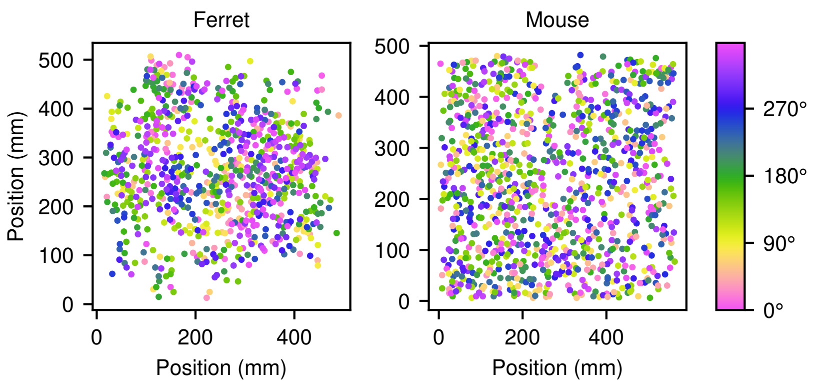

Figure 7: A map of each neuron's position, colored by its preferred angle as calculated by the GPMD decoder. The ferret neurons are clustered into cortical columns according to their direction selectivity, but the mouse neurons are not.

The GPMD also performed well with any number of neural features (fig. 4C). Linear decoders with no or poor regularization, such as the GID and ELD, did not exhibit this property; in fact, their performance became worse as the number of neural features increased from the "classical" to the "interpolating" regime, producing a phenomenon known as the "double descent" error curve (Belkin et al., 2019). Properly regularized models such as the GPGID and GPMD did not display this phenomenon and gave accurate performance estimates for all numbers of neural features.

Thanks to the GPMD's approximate inference, GPU acceleration, and spectral weight representation, it trained quickly, producing fast cross-validated error estimates that exhibited favorable scaling with respect to both observations and neurons (figures 4B and 4D). For the largest dataset with 20,000 neurons, it took 131 +/- 0.82 seconds to train (roughly 20 minutes of wall-clock time) for a ten-fold cross-validation estimate. By comparison, a performance-tuned GLMNET model took 618 +/- 6.40 seconds to train (roughly 1 hour and 45 minutes of wall-clock time) for the same estimate. Given the training time trends shown in the training-set size ablation (fig. 4BB) and neural feature ablation (fig. 4D) studies, we expect the GPMD to handle even larger datasets without difficulty.

Scaling to large datasets was further enhanced by the GPMD's automatic dataset preprocessing. Decoding studies, such as Graf et al. (2011), often select only strongly tuned neurons for decoding, since noisy neurons make it easier for models to overfit. Manual selection rules have two disadvantages: first, they may ignore neurons that look noisy but actually carry information, and second, they can require prohibitive amounts of time if human input is needed (e.g., for choosing initialization points for nonlinear curve fitting).

The GPMD's Bayesian formulation automatically discarded noise neurons by setting their prior amplitudes to zero (fig. 5), a phenomenon known as automatic relevance determination (MacKay, 1992; Neal, 1996). Examples of tuning curves from automatically discarded and automatically retained neurons are shown in figure 5. Some of the automatically retained neurons displayed the bimodal "Gaussian bump" structure commonly sought by manual selection rules. Others displayed more complicated tuning patterns that would likely be ignored by a manual selection rule.

Our implementation of the empirical linear decoder (ELD, see Graf et al. [2011]) replicated the original paper's results only qualitatively, not quantitatively. Our implementation of the ELD did outperform the Poisson Independent Decoder (PID) when using the "proportion correct" error criterion, as in the original paper (see supplementary figure 8). However, it did not achieve the performance reported in the original paper. Because our implementation of the PID, a very simple decoder, also did not match the performance of the PID in Graf et al. (2011), we believe the discrepancy was caused by data preprocessing. We were not able to replicate the data preprocessing steps described in Graf et al. (2011) precisely, since the original code has been lost.

5.2 Scientific implications

Previous studies, such as Graf et al. (2011) and Stringer et al. (2021), have used correlation-blind and correlation-aware decoders to investigate the effects of correlations on decoding performance. In all cases, they have found that correlation-aware decoders outperform correlation-blind decoders. However, the performance difference could be due to the lack of regularization in the correlation-blind decoders. In the sample-poor data regimes typically studied, the ideal weights of correlation-blind decoders are often corrupted by substantial amounts of noise.

Our results show that the performance difference between correlation-blind and correlation-aware decoders is fundamental, and not just a result of regularization. The correlation-aware decoders consistently perform better than even the regularized correlation-blind decoders (fig. 3), though regularization does narrow the gap significantly.

To characterize the effects of correlations on decoding performance, we visualized the decoding separatrices given by a correlation-blind model, the GPPID, and a correlation-aware model, the GPMD, on the third monkey dataset (fig. 6A). To reduce the 147 neural dimensions to two dimensions for visualization, we first selected a model to visualize, and two classes i and j. Then, we formed a two-dimensional basis by orthogonalizing the decoding weight vectors W_{i*} and W_{j*}. Using this basis, we were able to plot the data for classes i and j; the source model's separatrix, which lay in the basis span; and approximate separatrices from other models, which had to be projected onto the basis. We performed this procedure both the GPMD and GPPID models, since each two-dimensional basis could exactly represent only the separatrix from its source weight matrix.

We first wished to determine whether the data deviated significantly from the independent Poisson model assumed by the GPPID. To do this, we generated an uncorrelated dataset using a Poisson distribution that matched the data's empirical mean (referred to in the figure as the "Poisson surrogate" dataset) and plotted both it and the real dataset in the same basis. Compared to the real data, the Poisson surrogate data exhibit much less variance and tilt relative to the basis vectors, showing that the correlations in the real data can significantly affect decoding. However, in the GPMD basis, the differences between the real and surrogate datasets are much less pronounced, implying that the GPMD's weight matrix incorporates a linear transform that "decorrelates" the data somewhat.

Next, we plotted the class separatrices along with the data. The separatrix given by the GPPID successfully separated the Poisson surrogate data, but failed to separate the real dataset because of correlation-induced distortions. However, as expected, the GPMD separatrix successfully took the data's correlations into account.

To visualize how the entire set of 72 classes related to each other, we plotted each class's correlation ellipse on the basis given by the zero- and five-degree basis vectors from each model (fig. 6B). The GPPID's basis did a poor job of separating the classes, but the GPMD's basis separated them fairly well. In the GPMD's basis, the ellipses from classes 180 degrees apart appear in nearly identical locations, confirming that that the GPMD identified grating angles more precisely than grating drift direction, a phenomenon previously observed in our performance benchmarks (fig. 3C).

Finally, we used each decoder to form a spatial map of each neuron's preferred decoding angle, calculated for the neuron i as the argmax of the weight matrix column $W_{*}i$. We found that the GPMD decoder clustered the ferret neurons into direction-selective columns, but not the mouse neurons (fig. 7). This result is consistent with previous studies on direction selectivity in mouse and ferret visual cortex, which have discovered column structure in ferret visual cortex, but not in mouse (Rochefort et al., 2011).

6 Conclusion

Linear decoders are a natural way to characterize the information in neural populations. While all linear decoders share the same basic classification rule, they make differing assumptions about the neural population that affect both accuracy and parameter inference. Decoders with restrictive assumptions, like correlation-blind decoders based on independent generative models, generally have the worst performance. More accurate linear decoders (e.g., GLMNET) model dependencies across neurons, and the most accurate (e.g., the SND and GPMD) take into account—at least implicitly—dependencies across both neurons and stimuli.

In this paper, we present a suite of new decoders which share a common regularization strategy. The correlation-aware decoder in the suite, the GPMD, explicitly models correlations across neurons and stimuli. We find that it matches or outperforms all other decoders on three real world datasets, from monkey, ferret and mouse. Furthermore, it scales to the very largest datasets using a combination of approximate Bayesian inference, spectral methods, and GPU acceleration.

We investigated the effect of neural correlations on decoding by comparing the performance of the regularized correlation-aware and correlation-blind decoders. We found that the performance gap between correlation-aware and correlation-blind decoders is fundamental—that is, it is not an artifact of the sophisticated regularization schemes commonly used by correlation-aware decoders. This confirmed the results of previous studies. Even with sophisticated regularization, the correlation-blind decoders still performed worse than the correlation-aware decoders. Thus, we may conclude that exploiting neural correlations can significantly improve decoding performance.

Visualizations of the decoding separatrices produced by each decoder indicate that the real datasets differ significantly from the assumptions made by correlation-blind decoders. The correlation-aware decoders discovered low-dimensional subspaces that "decorrelated" the data, making the transformed data match independence assumptions more closely.

Finally, we note that our decoder discovered the cortical column structure (or lack therof) in ferret and mouse visual cortex.

Acknowledgements

This work was supported by grants from the Simons Collaboration on the Global Brain (SCGB AWD543027), the NIH BRAIN initiative (NS104899 and R01EB026946), and a U19 NIH-NINDS BRAIN Initiative Award (5U19NS104648). Jacob L. Yates is supported by the NIH (K99EY032179). Benjamin Scholl is supported by the NIH (K99EY031137) and thanks the Max Planck Society and Max Planck Florida Institute for their generous support. We thank A. B. A. Graf, Al Kohn, M. Jazayeri, and J. A. Movshon for providing the primate datasets; and C. Stringer, M. Michaelos, and M. Pachitariu for providing the publicly available mouse datasets.

References

Abbott, L. F. (1994). Decoding neuronal firing and modelling neural networks. *Quarterly reviews of biophysics*, *27*(3), 291–331. https://doi.org/10.1017/s0033583500003024

Averbeck, B. B., Latham, P. E., & Pouget, A. (2006). Neural correlations, population coding and computation. *Nature Reviews Neuroscience*, *7*(5), 358–366.

Azevedo-Filho, A., & Shachter, R. D. (1994). Laplace's method approximations for probabilistic inference in belief networks with continuous variables. In R. L. de Mantaras & D. Poole (Eds.), *Uncertainty proceedings 1994* (pp. 28–36). Morgan Kaufmann. https://doi.org/10.1016/B978-1-55860-332-5.50009-2

Bartolo, R., Saunders, R. C., Mitz, A. R., & Averbeck, B. B. (2020). Information-limiting correlations in large neural populations. *Journal of Neuroscience*, *40*(8), 1668–1678.

Beck, J., Ma, W., Latham, P., & Pouget, A. (2007). Probabilistic population codes and the exponential family of distributions. *Progress in brain research*, *165*, 509–519.

Belkin, M., Hsu, D., Ma, S., & Mandal, S. (2019). Reconciling modern machine-learning practice and the classical bias-variance trade-off. *Proceedings of the National Academy of Sciences of the United States of America*, *116*(32), 15849–15854. https://doi.org/10.1073/pnas.1903070116

Berens, P., Ecker, A. S., Cotton, R. J., Ma, W. J., Bethge, M., & Tolias, A. S. (2012). A fast and simple population code for orientation in primate V1. *The Journal of neuroscience: the official journal of the Society for Neuroscience*, *32*(31), 10618–10626. https://doi.org/10.1523/JNEUROSCI.1335-12.2012

Bishop, C. M. (2006). *Pattern recognition and machine learning*. Springer.

Blei, D. M., Kucukelbir, A., & McAuliffe, J. D. (2017). Variational inference: A review for statisticians. *Journal of the American Statistical Association*, *112*(518), 859–877. https://doi.org/10.1080/01621459.2017.1285773

Ecker, A. S., Berens, P., Tolias, A. S., & Bethge, M. (2011). The effect of noise correlations in populations of diversely tuned neurons. *The Journal of Neuroscience*, *31*(40), 14272–14283.

Földiák, P. (1993). The 'ideal homunculus': Statistical inference from neural population responses. In F. H. Eeckman & J. M. Bower (Eds.), *Computation and neural systems* (pp. 55–60). Springer.

Friedman, J., Hastie, T., & Tibshirani, R. (2010). Regularization paths for generalized linear models via coordinate descent. *Journal of statistical software*, *33*(1), 1–22.

Graf, A. B. A., Kohn, A., Jazayeri, M., & Movshon, J. A. (2011). Decoding the activity of neuronal populations in macaque primary visual cortex. *Nature neuroscience*, *14*(2), 239–245. https://doi.org/10.1038/nn.2733

Hiner, M. C., Rueden, C. T., & Eliceiri, K. W. (2017). Imagej-matlab: A bidirectional framework for scientific image analysis interoperability. *Bioinformatics*, *33*(4), 629–630.

Hoffman, M. D., Blei, D. M., Wang, C., & Paisley, J. (2013). Stochastic variational inference. *Journal of machine learning research: JMLR*, *14*, 1303–1347.

Kanitscheider, I., Coen-Cagli, R., & Pouget, A. (2015). Origin of information-limiting noise correlations. *Proceedings of the National Academy of Sciences*, *112*(50), E6973–E6982.

Kohn, A., Coen-Cagli, R., Kanitscheider, I., & Pouget, A. (2016). Correlations and neuronal population information. *Annual review of neuroscience*, *39*(1), 237–256. https://doi.org/10.1146/annurev-neuro-070815-013851

Ma, W. J., Beck, J. M., Latham, P. E., & Pouget, A. (2006). Bayesian inference with probabilistic population codes. *Nature Neuroscience*, *9*, 1432–1438.

MacKay, D. J. C. (1992). Bayesian interpolation. *Neural computation*, *4*(3), 415–447. https://doi.org/10.1162/neco.1992.4.3.415

Moreno-Bote, R., Beck, J., Kanitscheider, I., Pitkow, X., Latham, P., & Pouget, A. (2014). Information-limiting correlations. *Nat Neurosci*, *17*(10), 1410–1417. https://doi.org/10.1038/nn.3807

Neal, R. M. (1996). *Bayesian learning for neural networks*. Springer, New York, NY. https://doi.org/https://doi-org.ezproxy.princeton.edu/10.1007/978-1-4612-0745-0

Nelder, J. A., & Mead, R. (1965). A simplex method for function minimization. *Computer Journal*, *7*(4), 308–313. https://doi.org/10.1093/comjnl/7.4.308

Nirenberg, S., & Latham, P. E. (2003). Decoding neuronal spike trains: How important are correlations? *PNAS*, *100*, 7348–7353.

Park, M., Weller, J. P., Horwitz, G. D., & Pillow, J. W. (2014). Bayesian active learning of neural firing rate maps with transformed gaussian process priors. *Neural Computation*, *26*(8), 1519–1541. http://www.mitpressjournals.org/doi/abs/10.1162/NECO_a_00615

Peirce, J. W. (2007). Psychopy—psychophysics software in python. *Journal of neuroscience methods*, *162*(1-2), 8–13.

Pnevmatikakis, E. A., & Giovannucci, A. (2017). Normcorre: An online algorithm for piecewise rigid motion correction of calcium imaging data. *Journal of neuroscience methods*, *291*, 83–94.

Pologruto, T. A., Sabatini, B. L., & Svoboda, K. (2003). Scanimage: Flexible software for operating laser scanning microscopes. *Biomedical engineering online*, *2*(1), 1–9.

Rad, K. R., & Paninski, L. (2010). Efficient, adaptive estimation of two-dimensional firing rate surfaces via gaussian process methods. *Network: Computation in Neural Systems*, *21*(3-4), 142–168.

Rasmussen, C. E., & Williams, C. K. I. (2006). *Gaussian processes for machine learning*. MIT Press.

Rochefort, N. L., Narushima, M., Grienberger, C., Marandi, N., Hill, D. N., & Konnerth, A. (2011). Development of direction selectivity in mouse cortical neurons. *Neuron*, *71*(3), 425–432. https://doi.org/10.1016/j.neuron.2011.06.013

Schneidman, E., Bialek, W., & Berry, M. J. (2003). Synergy, redundancy, and independence in population codes. *J Neurosci*, *23*(37), 11539–11553.

Scholl, B., Wilson, D. E., & Fitzpatrick, D. (2017). Local order within global disorder: Synaptic architecture of visual space. *Neuron*, *96*(5), 1127–1138.

Stringer, C., Michaelos, M., Tsyboulski, D., Lindo, S. E., & Pachitariu, M. (2021). High-precision coding in visual cortex. *Cell*, *184*(10), 2767–2778.e15. https://doi.org/10.1016/j.cell.2021.03.042

Zohary, E., Shadlen, M. N., & Newsome, W. T. (1994). Correlated neuronal discharge rate and its implications for psychophysical performance. *Nature*, *370*(6485), 140–143. https://doi.org/10.1038/370140a0

Zou, H., & Hastie, T. (2005). Regularization and variable selection via the elastic net. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, *67*(2), 301–320.

A Linearity of the PID and GID decoders

Both the GID and the PID decoders, under appropriate assumptions, have linear decision boundaries. To derive both decision boundaries at the same time, let us consider the more general case of a naïve Bayes decoder as described in section 3.1.1, but with an exponential family likelihood (Bishop, 2006). That is, the likelihood of the dth element of the feature vector $\mathbf{x}_d$ can be written as

$$P(\mathbf{x}_d \mid y = k; \eta_{kd}) = h(\mathbf{x}_d)g(\eta_{kd}) \exp(\eta_{kd} u(\mathbf{x}_d)) \tag{46}$$

where η is the natural parameter and the sufficient statistic $u : \mathbb{R} \to \mathbb{R}$ is a function of $\mathbf{x}_d$.

With this likelihood in mind, we can begin solving for the decision boundary. Our class prediction $\hat{y}$ for a given example $\mathbf{x}$ is

$$\hat{y} = \underset{k \in \{1,\dots,K\}}{\arg\max}\, P(y = k \mid \mathbf{x}) \tag{47}$$

$$= \underset{k \in \{1,\dots,K\}}{\arg\max} \frac{P(\mathbf{x} \mid y = k)P(y = k)}{\sum_{k'=1}^{K} P(\mathbf{x} \mid y = k')P(y = k')} \tag{48}$$

If we assume that the prior probabilities are constant, i.e. $P(y = k) = P(y = k')$ for every $k, k' \in \{1, \dots, K\}$, then this simplifies to

$$= \underset{k \in \{1,\dots,K\}}{\arg\max} \frac{P(\mathbf{x} \mid y = k)}{\sum_{k'=1}^{K} P(\mathbf{x} \mid y = k')} \tag{49}$$

Introducing a log under the argmax, dropping terms that don't depend on k, and substituting $P(\mathbf{x}) = \prod_{d=1}^{D} P(\mathbf{x}_d)$, we simplify further:

$$= \underset{k \in \{1,\dots,K\}}{\arg\max} \log P(\mathbf{x} \mid y = k) - \log \sum_{k'=1}^{K} P(\mathbf{x} \mid y = k') \tag{50}$$

$$= \underset{k \in \{1,\dots,K\}}{\arg\max} \sum_{d=1}^{D} \log P(\mathbf{x}_d \mid y = k) \tag{51}$$

Writing out the exponential family form, we have

$$= \underset{k \in \{1,\dots,K\}}{\arg\max} \sum_{d=1}^{D} \log\left(h(\mathbf{x}_d)g(\eta_{kd}) \exp(\eta_{kd} u(\mathbf{x}_d))\right) \tag{52}$$

$$= \underset{k \in \{1,\dots,K\}}{\arg\max} \sum_{d=1}^{D} \log h(\mathbf{x}_d) + \log g(\eta_{kd}) + \eta_{kd} u(\mathbf{x}_d) \tag{53}$$

$$= \underset{k \in \{1,\dots,K\}}{\arg\max} \sum_{d=1}^{D} \log g(\eta_{kd}) + \eta_{kd} u(\mathbf{x}_d) \tag{54}$$

This will simplify to the form of a linear decoder as long as the sufficient statistic of the exponential distribution $u(\mathbf{x}_d)$ of the form $u(\mathbf{x}_d) = \alpha \mathbf{x}_d$ where α is a scalar. In that case, the entries of the weight matrix are given by the natural parameters

$$W_{kd} = \alpha \eta_{kd} \tag{55}$$

and the entries of the intercept vector $\mathbf{b}$ are given by

$$\mathbf{b}_k = \sum_{d=1}^{D} \log g(\eta_{kd}) \tag{56}$$

Using these definitions, we can write

$$\hat{y} = \underset{k \in \{1,\ldots,K\}}{\arg\max} \, (W\mathbf{x} + \mathbf{b})_k \tag{57}$$

which is the form of a linear decoder.

In the case of the Poisson Independent Decoder, the sufficient statistic is the identity function, the natural parameter of the Poisson distribution is given by $\eta_{kd} = \log \lambda_{kd}$ and $g(\eta_{kd}) = e^{-\lambda_{kd}}$. Thus, for the PID,

$$W_{kd} = \log \lambda_{kd} \tag{58}$$

$$\mathbf{b}_d = -\sum_{k=1}^{K} \lambda_{kd} \tag{59}$$

The case of the Gaussian Independent Decoder is slightly more complicated, since the Gaussian sufficient statistic only takes the proper form if the variance σ^2 can be incorporated into $h(\mathbf{x}_d)$, a term we dropped from the argmax. For this dropping to be valid, we must constrain $\sigma_{kd} = \sigma_{k'd}$ for all $k, k' \in \{1,\ldots,K\}$. If this is true, then the sufficient statistic is $u(\mathbf{x}_d) = \mathbf{x}_d/\sigma_d$, the natural parameter is given by $\eta_{kd} = \mu_{kd}/\sigma_d$, and $g(\eta_{kd}) = \exp\left(-\frac{\mu_{kd}^2}{2\sigma_{kd}^2}\right)$. Thus, for the GID,

$$W_{kd} = \frac{\mu_{kd}}{\sigma_d^2} \tag{60}$$

$$\mathbf{b}_d = -\sum_{k=1}^{K} \frac{\mu_{kd}^2}{2\sigma_d^2} \tag{61}$$

B Spectral GP regression

B.1 With Gaussian noise

In this section we demonstrate how to solve a 1-D GP regression problem in the spectral domain. Consider a regression dataset $\{x_t, y_t\}_{t=1}^{T}$ with $y \in \mathbb{R}$ and, without loss of generality, $x \in [-\pi, \pi]$. Note that we do not require the x values to lie on a grid. We can concatenate the training examples and labels into vectors as follows: $\mathbf{x} = (x_1, \dots, x_t)^\top$ and $\mathbf{y} = (y_1, \dots, y_t)^\top$.

We assume that the values of $\mathbf{y}$ are noisy observations of a zero-mean Gaussian process z where $\mathbf{z}_i = z(\mathbf{x}_i)$, i.e. $\mathbf{y}_i = \mathbf{z}_i + \epsilon$ where $\epsilon \sim \mathcal{N}(0, \sigma^2)$. Our probability model can be written:

$$
\begin{aligned}
p(\mathbf{y}, \mathbf{z}) &= p(\mathbf{y} \mid \mathbf{z}) p(\mathbf{z}) && (62)\\
&= \mathcal{N}(\mathbf{y}; \mu = \mathbf{z}, \Sigma = \sigma^2 I)\mathcal{N}(\mathbf{z}, \mu = \mathbf{0}, \Sigma = K_\theta(\mathbf{x}, \mathbf{x})) && (63)
\end{aligned}
$$

Here $K_\theta(\mathbf{x}, \mathbf{x})$ is the kernel matrix between $\mathbf{x}$ and $\mathbf{x}$. If $\kappa_\theta(\cdot)$ is the stationary kernel function with hyperparameters θ, then $K_\theta(\mathbf{x}, \mathbf{x})_{ij} = \kappa_\theta(\mathbf{x}_i - \mathbf{x}_j)$. To infer the GP hyperparameters using type-II maximum likelihood, we wish to maximize the log evidence given by

$$
\begin{aligned}
\log p(\mathbf{y} \mid \theta) &= \log \int p(\mathbf{y} \mid \mathbf{z}, \theta) p(\mathbf{z} \mid \theta)\, d\mathbf{z} && (64)\\
&= -\frac{1}{2}\mathbf{y}^\top (K + \sigma^2 I)^{-1}\mathbf{y} - \frac{1}{2}\log|K + \sigma^2 I| - \frac{T}{2}\log 2\pi && (65)
\end{aligned}
$$

However, each evaluation of this expression has cost $\sim n^3$, which is intractable for large n. Our goal is to decorrelate $\mathbf{z}$ so that the prior covariance becomes diagonal, dropping the cost to $\sim n$.

To achieve this we will represent the GP $z(x)$ using a Fourier series:

$$
z(x) = \frac{1}{\sqrt{N}} \sum_{k=-N/2+1}^{N/2} F_k \exp(ik\Omega x) \qquad (66)
$$

where $\Omega = 2\pi/N$ and the F_k are the Fourier series coefficients. Our goal is to find the distribution of F_k such that z is a real zero-mean Gaussian with $\mathrm{Cov}[z(x), z(x')] = \kappa_\theta(x, x')$.

To ensure that $z(x)$ is real, we require $F_k = F^*_{-k}$. This requirement can be verified by expanding equation 66 in terms of sines and cosines. To ensure that $z(x)$ is Gaussian, we require that the F_k are independent Gaussian random variables, which implies that they are jointly Gaussian. Since $z(x)$ is a linear combination of the F_ks, this implies it is also Gaussian. To ensure that $z(x)$ is zero mean, we require that each F_k is zero mean. Because expectation is a linear operator, this ensures that $z(x)$ is also zero mean.

The trickiest task is finding the variance of the F_ks that induces the proper GP distribution on z. We can construct an equation to solve for it as follows: given a lag

$\tau = s - t$ between two x values, we have the covariance

$$\kappa(\tau) = \mathbb{E}[z(s)z(t)^*] \tag{67}$$

$$\kappa(\tau) = \frac{1}{N}\mathbb{E}\left[\sum_{k=-N/2+1}^{N/2} F_k \exp(ik\Omega s) \sum_{k'=-N/2+1}^{N/2} F_{k'}^* \exp(-ik'\Omega t)\right] \tag{68}$$

$$\kappa(\tau) = \frac{1}{N}\sum_{k,k'=-N/2+1}^{N/2} \mathbb{E}[F_k F_{k'}^*] \exp(i\Omega(kx - k't)) \tag{69}$$

Because the Fourier coefficients are independent, we have $\mathbb{E}[F_k F_{k'}^*] = 0$ for all $k \neq k'$.

$$\kappa(\tau) = \frac{1}{N}\sum_{k=-N/2+1}^{N/2} \mathbb{E}[F_k F_k^*] \exp(ik\Omega\tau) \tag{70}$$

This is just a Fourier series. Thus, we can use the Fourier coefficient formula to invert the equation and solve for $\mathbb{E}[F_k F_k^*]$:

$$\mathbb{E}[F_k F_k^*] = \frac{1}{2\pi\sqrt{N}} \int_{-\pi}^{\pi} \kappa(\tau) \exp\left(-2\pi i \frac{k}{2\pi}\tau\right) d\tau \tag{71}$$

At this point, we are done. However, many implementations use the Fourier transform of κ rather than the Fourier coefficient expression given above. The equivalence can be derived by extending the bounds of integration to $[-\infty, \infty]$. This is a reasonable approximation as long as $\kappa(\tau)$ is close to zero outside $[-\pi, \pi]$—which is true for the RBF and Matern kernels as long as the lengthscale is short. Extending the bounds of integration, we have

$$\mathbb{E}[F_k F_k^*] \approx \frac{1}{\sqrt{N}} \mathcal{F}[\kappa]\left(\frac{k}{2\pi}\right) \tag{72}$$

For notational simplicity, let $\mathbf{w}$ be the vector of frequency-domain coefficients and $\mathbf{s}$ be the vector of associated covariances. Then the log evidence we wish to maximize is

$$\begin{aligned} p(\mathbf{y} \mid \theta) &= p(\mathbf{y} \mid \mathbf{z}, \theta) p(\mathbf{z} \mid \theta)\, d\mathbf{z} && (73) \\ &= \int \mathcal{N}(\mathbf{y}; \mu = \Psi\mathbf{w}, \Sigma = \sigma^2 I)\mathcal{N}(\mathbf{w}; \mu = \mathbf{0}, \Sigma = \mathrm{diag}(\mathbf{s}))\, d\mathbf{w} && (74) \\ &= -\frac{T}{2}\log(2\pi) - \frac{T}{2}\log(\sigma^2) - \frac{1}{2\sigma^2}\mathbf{y}^\top\mathbf{y} - \frac{1}{2}\log|A| \\ &\quad + \frac{1}{2}\mathbf{b}^\top A^{-1}\mathbf{b} - \frac{1}{2}\log|\mathrm{diag}(\mathbf{s})| && (75) \end{aligned}$$

where $A = \frac{\Psi^\top\Psi}{\sigma^2} + \mathrm{diag}(\mathbf{s})^{-1}$ and $\mathbf{b} = \frac{1}{\sigma^2}\Psi^\top\mathbf{y}$. If Ψ is unitary, which is true if $\mathbf{x}$ lies on a grid, then A will be diagonal, which simplifies the gradient and Hessian calculations somewhat.

B.2 With Poisson noise

Consider the same regression problem as in §B.1, but with Poisson observation noise. We wish to fit the hyperparameters by maximizing the log evidence

$$\log p(\mathbf{y} \mid \theta) = \log \int p(\mathbf{y} \mid \mathbf{z}, \theta) p(\mathbf{z} \mid \theta)\, d\mathbf{z} \tag{76}$$

Since $p(\mathbf{y} \mid \mathbf{z}, \theta) = \prod_{t=1}^{T} \text{Poiss}(\mathbf{y}_t; \theta_t)$ the integral is not analytically tractable.

Define $h(\mathbf{z}) = p(\mathbf{y} \mid \mathbf{z}, \theta)p(\mathbf{z} \mid \theta)$ and its argmax as $\mathbf{z}^*$. Using the Laplace Approximation (Azevedo-Filho & Shachter, 1994), we can approximate the integral in eq. 76 as

$$\log p(y) = h(\mathbf{z}^*) + \frac{N}{2}\log(2\pi) - \frac{1}{2}\log|-\nabla^2 h(\mathbf{z}^*)| \tag{77}$$

Since evaluating this quantity requires finding $\mathbf{z}^*$ via an optimization procedure, it is difficult to maximize it using a derivative-based optimization algorithm, an issue pointed out by Rasmussen and Williams (2006). We use a derivative-free technique, the Nelder-Mead algorithm (Nelder & Mead, 1965).

C Dataset and preprocessing details

For each of the five monkey datasets provided by Graf et al. (2011), we chose the feature ($\{\mathbf{x}\}$) exactly as in Graf et al. (2011). The stimuli grating angles were selected from a five-degree grid, so to get y_t we simply mapped the angles $\{0, 5, 10, \dots, 360\}$ to the integers $\{0, 1, 2, \dots, 72\}$.

Unlike Graf et al., we did not drop noisy neurons from the dataset, since we found it made little to no difference in decoding accuracy (see Figure TODO).

For the three mouse datasets, we chose the feature ($\{\mathbf{x}\}$) vectors exactly as in Stringer et al. (2021). For the class values (y) values, we binned the stimulus angles using 2-degree bins and used the bin index as the class label.

For the ferret dataset, all procedures were performed according to NIH guidelines and approved by the Institutional Animal Care and Use Committee at Max Planck Florida Institute for Neuroscience. Surgical procedures and acute preparations were performed as described in Scholl et al. (2017). To preform calcium imaging of cellular populations, AAV1.Syn.GCaMP6s (UPenn) was injected at multiple depths (total volume 500 nL). Visual stimuli were generated using Psychopy (Peirce, 2007). The monitor was placed 25 cm from the animal, centered in the receptive field locations for the cells of interested. Square-wave drifting gratings (0.10 cycles per degree spatial frequency, 4Hz temporal frequency) were presented at 2 degree increments across the full range of directions (1 second duration, 1 second ISI, 11 trials). Two photon imaging was performed on a Bergamo II microscope (Thorlabs) running Scanimage (Pologruto et al., 2003) (Vidrio Technologies) with 940nm dispersion-compensated excitation provided by an Insight DS+ (Spectraphysics). Power after the objective was 40 mW. Images were collected at 30 Hz using bidirectional scanning with 512x512 pixel resolution. The full field of view was 1 x 1 mm. Raw images were corrected for in-plane motion via a non-rigid motion correction algorithm (Pnevmatikakis & Giovannucci, 2017). Regions of interest were drawn in ImageJ. Mean pixel values for ROIs were computed over the imaging time series and imported into MATLAB (Hiner et al., 2017). $\Delta F/F_o$ was computed by computing F_o with time-averaged median or percentile filter. $\Delta F/F_o$ traces were synchronized to stimulus triggers sent from Psychopy and collected by Spike2. Response amplitudes for each stimulus on each trial was calculated as the sum of the Fourier mean and modulation ($F_0 + F_1$). These values for each neuron were used to generate the feature ($\{\mathbf{x}\}$) vectors. Class values (y) were the stimulus angles presented (at 2-degree increments), using the bin index as the class label

D Supplementary figures

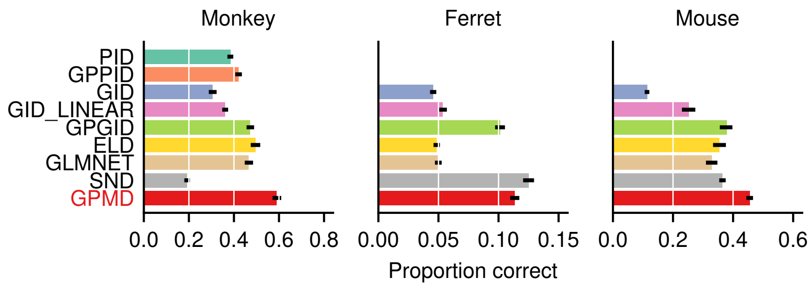

Figure 8: The same benchmark as in Figure 3, but calculated using proportion correct (i.e. proportion with 0° error) instead of mean absolute error. This is the same criterion as in Graf et al., 2011. Using it, we qualitatively replicate the results of Graf et al.

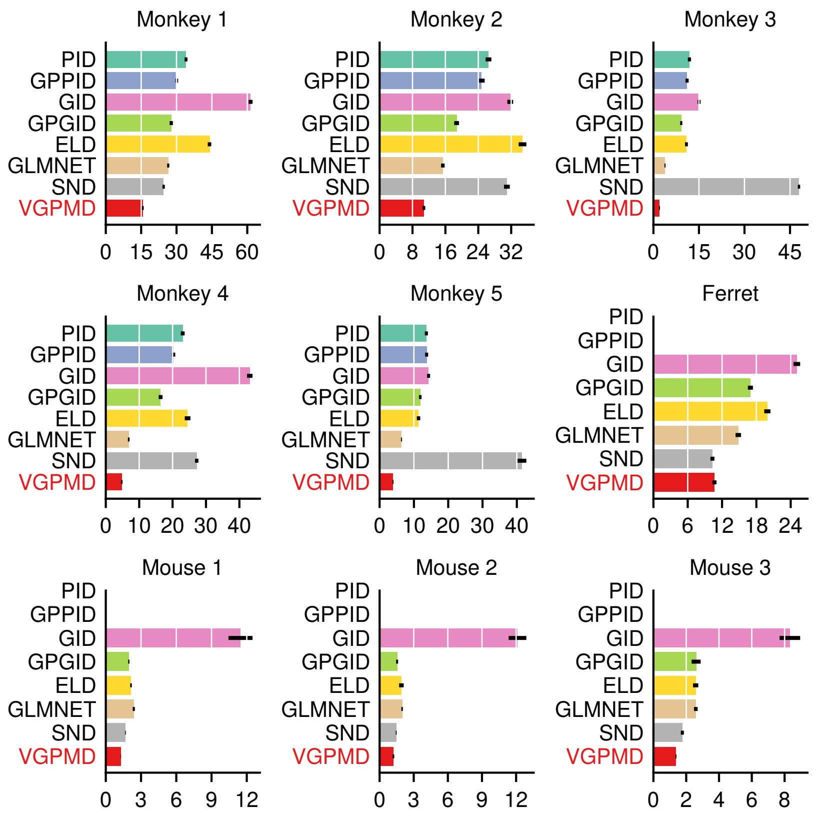

Figure 9: The same benchmark as in Figure 3, but without averaging across animals.

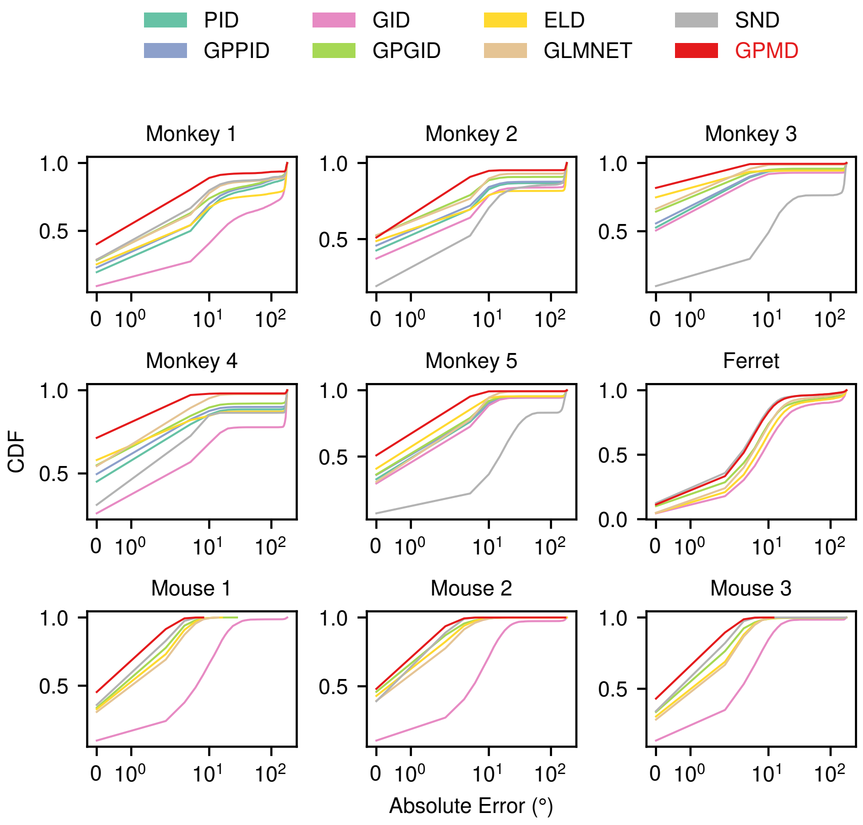

Figure 10: The same benchmark as in Figure 3B, but without averaging across animals.

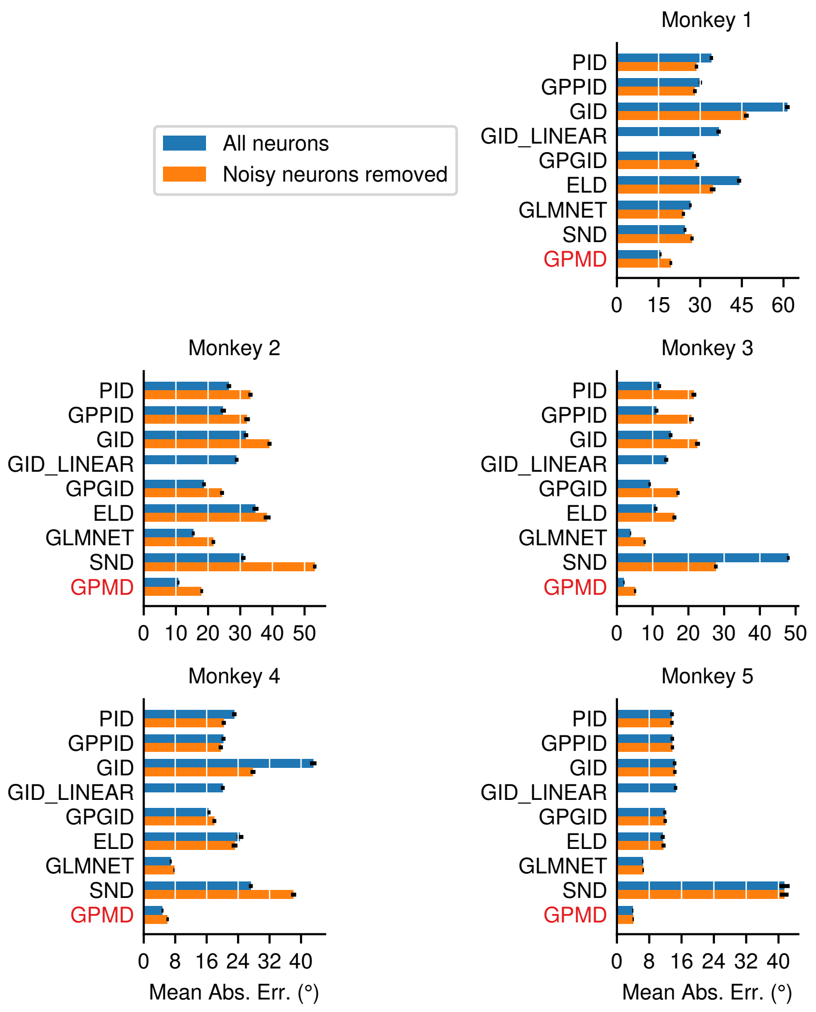

Figure 11: A comparison of model performance on the monkey datasets from Graf et al. (2011), both with and without noisy neurons included. Noisy neurons were dropped using the procedure detailed in the supplementary information of Graf et al. (2011). We thought that this filtering step would make the models perform better (matching the performance reported in Graf et al. (2011)), but it did not. We suspect that the issue lies in the initialization of our nonlinear curve fitting code, but we were not able to compare our implementation with the original, since the original code has been lost.

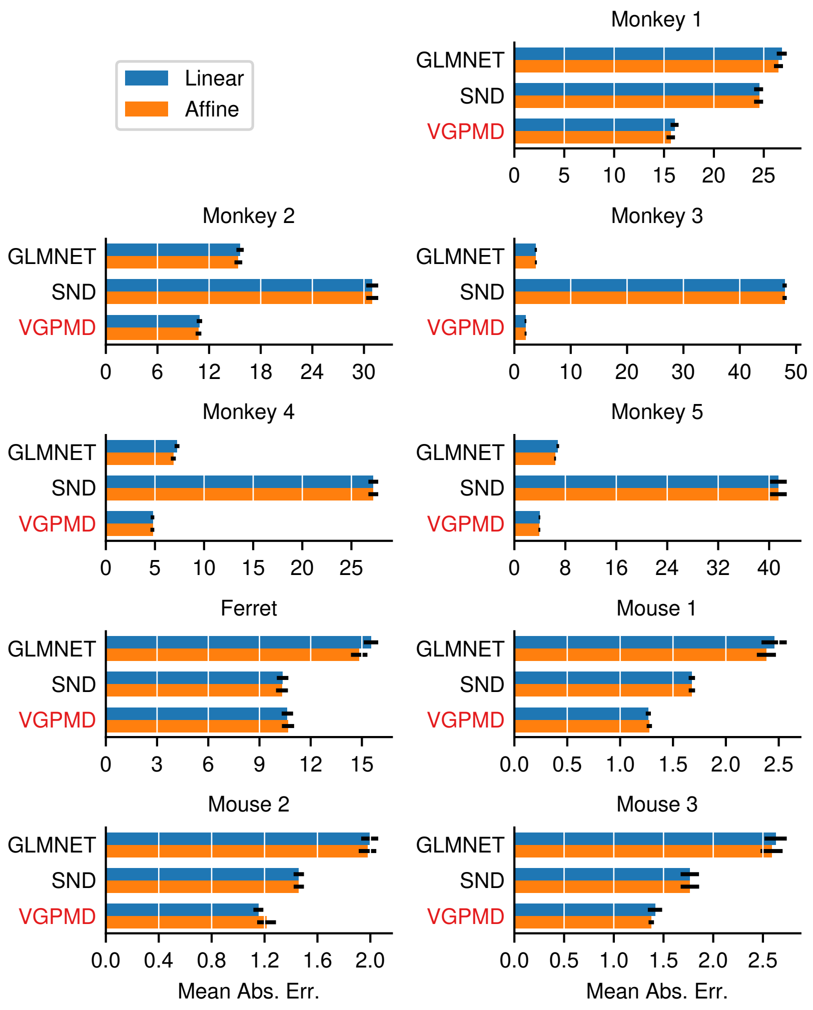

Figure 12: A comparison of model performance using linear ($y = W\mathbf{x}$) and affine ($y = Wx + b$) model formulations. One would expect the affine models to fit the data better, but the difference in performance is negligible. This is likely due to the relatively high dimensionality of the datasets, since separating hyperplanes are quite easy to find in high dimensions whether or not they are restricted to pass through the origin. We expect a larger performance difference on datasets with smaller feature dimensions (< 10).