Machine learning for neural decoding

Joshua I. Glaser^{1,2,6,7*}, Raeed H. Chowdhury^{3,4}, Matthew G. Perich^{3,4}, Lee E. Miller²⁻⁴, and Konrad P. Kording²⁻⁷

- 1. Interdepartmental Neuroscience Program, Northwestern University, Chicago, IL, USA
- 2. Department of Physical Medicine and Rehabilitation, Northwestern University and Shirley Ryan Ability Lab, Chicago, IL, USA
- 3. Department of Physiology, Northwestern University, Chicago, IL, USA
- 4. Department of Biomedical Engineering, Northwestern University, Chicago, IL, USA
- 5. Department of Applied Mathematics, Northwestern University, Chicago, IL, USA
- 6. Department of Neuroscience, University of Pennsylvania, Philadelphia, IL, USA
- 7. Department of Bioengineering, University of Pennsylvania, Philadelphia, IL, USA
- * Contact: joshglaser88@gmail.com

Abstract:

Despite rapid advances in machine learning tools, the majority of neural decoding approaches still use traditional methods. Improving the performance of neural decoding algorithms allows us to better understand the information contained in a neural population, and can help advance engineering applications such as brain machine interfaces. Here, we apply modern machine learning techniques, including neural networks and gradient boosting, to decode from spiking activity in 1) motor cortex, 2) somatosensory cortex, and 3) hippocampus. We compare the predictive ability of these modern methods with traditional decoding methods such as Wiener and Kalman filters. Modern methods, in particular neural networks and ensembles, significantly outperformed the traditional approaches. For instance, for all of the three brain areas, an LSTM decoder explained over 40% of the unexplained variance from a Wiener filter. These results suggest that modern machine learning techniques should become the standard methodology for neural decoding. We provide a tutorial and code to facilitate wider implementation of these methods.

Introduction:

Neural decoding uses activity recorded from the brain to make predictions about variables in the outside world. For example, researchers predict movements based on activity in motor cortex [1, 2], predict decisions based on activity in prefrontal and parietal cortices [3, 4], and predict locations based on activity in the hippocampus [5, 6]. There are two primary purposes of decoding. First, it is an increasingly critical tool for understanding how neural signals relate to the outside world. It can be used to determine how much information the brain contains about an external variable (e.g., sensation or movement) [7-9], and how this information differs across brain areas [10-12], experimental conditions [13, 14], disease states [15], and more. Second, it is useful in engineering contexts, such as for brain machine interfaces (BMIs), where signals from motor cortex are used to control computer cursors [1], robotic arms [16], and muscles [2]. Decoding is a central tool for neural data analysis.

When predicting a continuous variable, decoding is simply a regression problem and when predicting a discrete variable, decoding is simply a classification problem. Thus, there are many methods that can be used for neural decoding. However, despite the recent advances in machine learning techniques, it is still common to use traditional methods such as linear regression. Using modern machine learning tools for neural decoding would likely significantly boost performance, and might allow deeper insights into neural function.

Here, we first give a brief tutorial so that readers can get started with using standard machine learning methods for decoding. We provide companion code so that readers can easily use a variety of decoding methods. Next, we compare the performance of many different machine learning methods to decode information from neural spiking activity. We predict movement velocities from macaque motor cortex and

sensorimotor cortex, and locations in space from rat hippocampus. In all brain regions, modern methods, in particular neural networks and ensembles, led to the highest accuracy decoding, even for limited amounts of data.

Tutorial for getting started with using machine learning for decoding:

Code

We have made Python code available at https://github.com/KordingLab/Neural_Decoding, which accompanies the tutorial below. This includes code that will correctly format the neural and output data for decoding, a tutorial for hyperparameter optimization, and examples of using many different decoders. We go into more detail on these topics below.

General framework for decoding

The decoding problem we are considering can be summarized as follows. We have N neurons whose spiking activity is recorded for a period of time, T (Fig. 1a). While we focus here on spiking neurons, the same methods could be used with other forms of neural data, such as the BOLD activity of N voxels, or the power in particular frequency bands of N LFP or EEG signals. We have also recorded outputs that we are trying to predict over that same time period (Fig. 1a). Here, we focus on output variables that are continuous (e.g., velocity, position), rather than discrete (e.g., choice). However, the general framework is very similar for discrete output variables.

The first choice we need to make is to decide the temporal resolution, R, for decoding. That is, do we want to make a prediction every 50ms, 100ms, etc? We need to put the input and output into bins of length R (Fig. 1a). It is common (although not necessary) to use the same bin size for the neural data and output data, and we do so here. Thus, we will have approximately T/R total bins of neural activity and outputs. Within each bin, we compute the average activity of all neurons and the average value of the output.

Next, we need to choose the time period of neural activity used to predict a given output. In the simplest case, the activity from all neurons in a given time bin would be used to predict the output in that same time bin. However, it is often the case that we want the neural data to precede the output (e.g., in the case of movements) or follow the decoder output (e.g., in the case of sensation). Plus, we often want to use neural data from more than one bin (e.g., using 500 ms of preceding neural data to predict a movement in the current 50 ms bin). In the following, we use the nomenclature that B time bins of neural activity are being used to predict a given output. For example, if we use one bin preceding the output, one concurrent bin, and one following bin, then B=3 (Fig. 1a). Note that when multiple bins of neural data are used to predict an output (B>1), then overlapping neural data will be used to predict different output times (Fig. 1a).

When multiple bins of neural data are used to predict an output, then we will need to exclude some output bins. For instance, if we are using one bin of neural data preceding the output, then we cannot predict the first output bin, and if we are using one bin of neural data following the output, then we cannot predict the final output bin (Fig. 1a). Thus, we will be predicting K total output bins, where K is less than the total number of bins (T/R). To summarize, our decoders will be predicting each of these K outputs using B surrounding bins of activity from N neurons.

Below, we describe how to format the neural data and output variables for use in different types of decoders.

Non-recurrent decoders: For many "non-recurrent" decoders, we are just solving a standard machine learning regression problem. We have $N \times B$ features (the firing rates of each neuron in each relevant time bin) that are used to predict each output (Fig. 1b). If there is a single output that is being predicted, it can be put in a vector, \mathbf{Y} , of length K. Note that for many decoders, if there are multiple outputs, each is independently decoded. If multiple outputs are being simultaneously predicted, which can occur with neural network decoders, the outputs can be put in a matrix \mathbf{Y} , that has K rows and K columns, where K is the

number of outputs being predicted. The input covariate matrix, X, has $N \times B$ columns (one for each feature) and K rows (corresponding to each output being predicted). This is now the format of a standard regression problem. Linear regression simply finds a linear combination of these features that predicts the output. More sophisticated forms of regression use nonlinear combinations of features for predictions. In general, this format is beneficial because there are many machine learning regression techniques that can easily be substituted for one another. We provide code for a Wiener filter (linear regression), a Wiener cascade (a linear-nonlinear model), support vector regression, XGBoost (gradient boosted trees), and feedforward neural networks (see Methods). We test the performance of these decoders in Results.

Recurrent neural network decoders: When using recurrent neural networks (RNNs) for decoding, we need to put the inputs in a different format. Recurrent neural networks explicitly model temporal transitions across time (Fig. 1c). In the non-recurrent decoders, there were N x B features that were equivalently used for prediction, regardless of the time bin they came from. However, with a recurrent decoder, at each time bin, N features (the firing rates of all neurons in that time bin) are used for predicting the hidden state of the system at that time. Along with being a function of the N features, the hidden state at a time bin is also a function of the hidden state at the previous time bin (Fig. 1c). After transitioning through all B bins, the hidden state in this final bin is used to predict the output. This architecture allows the decoder to take advantage of temporal structure in the data, and allowing it (via its hidden state) to integrate the effect of neural inputs over an extended period of time. For use in this type of decoder, the input can be formatted as a 3-dimensional matrix of size $K \times N \times B$ (Fig. 1c). That is, for each row (corresponding to the output that is predicted), there will be N features (2^{nd} matrix dimension) over B bins (3^{rd} matrix dimension) used for prediction. Within this format, different types of RNNs, including those more sophisticated than the standard RNN shown in Fig. 1c, can be easily switched for one another. We provide code for a standard recurrent network, a gated recurrent unit (GRU) network, and a long short-term memory (LSTM) network. In Results, we test the performance of these decoders.

Decoders with additional information: While the focus of this tutorial is on decoders that fit into standard machine learning frameworks, we want to briefly mention two other commonly used decoders. The Kalman filter and its variants have frequently been used in the brain computer interface field for decoding movements [17-19]. Bayesian decoders, such as a Naïve Bayes decoder, have been used for decoding an animal's location from hippocampus activity [5, 20, 21]. Both types of decoder explicitly use additional information beyond neural activity to predict the output variables. The Kalman filter uses information about how kinematics (including the output variables position and velocity) transition from one time bin to another. Bayesian decoders can also use transition information, as well as prior information about the probability distribution of the output variables. We provide code for these decoders and test their performance in *Results*. More details about these decoders are in *Methods*.

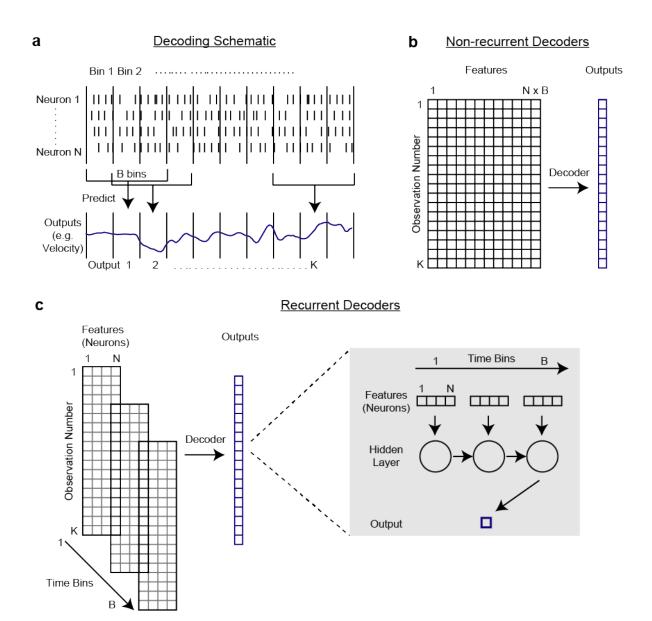


Figure 1: Decoding Schematic

a) To decode (predict) the output in a given time bin, we used the firing rates of all *N* neurons in *B* time bins. In this schematic, *N*=4 and *B*=3 (one bin preceding the output, one concurrent bin, and one following bin). Here, we show a single output being predicted. In our data, we predicted two outputs from each brain region (x and y components of velocity predicted from motor and somatosensory cortex, and x and y components of position predicted from hippocampus). For each region, the number of neurons and time bins used for decoding are described in *Methods*. **b)** For the non-recurrent decoders (Wiener Filter, Wiener Cascade, Support Vector Regression, XGBoost, and Feedforward Neural Network), this is a standard machine learning regression problem where *N* x *B* features (the firing rates of each neuron in each relevant time bin) are used to predict the output. **c)** To predict outputs with the recurrent decoders (simple recurrent neural network, GRUs, LSTMs) we used *N* features, with temporal connections across *B* bins. A schematic of a recurrent neural network predicting a single output is on the right.

Testing decoder performance

To test the performance of a decoder, we need a goodness of fit metric. We used a variant of R^2 that estimates the explained variance (see *Methods* below). It is important to test the decoder performance on a separate, held-out, dataset because a decoder might be trained to overfit a given dataset. That is, the decoders might fit to noise in the training dataset, and thus be unable to provide good predictions on new datasets. To this end, the available data should be split into separate "training" and "testing" datasets. A common method that

builds upon this idea is cross-validation. In 10-fold cross-validation, for example, the dataset is split into 10 sets. The decoder is trained on 9 of the sets, and performance is tested on the final set. This is done 10 times, so that each set is tested once. The performance on all test sets is generally averaged together to determine the overall performance.

Simplicity/complexity tradeoff of decoders

Like with all models, there are tradeoffs between the complexity and simplicity of decoders. Complex decoders have many parameters, so they can provide a good fit to almost any dataset. However, more complex models run a risk of overfitting to the dataset they are trained on (since they have sufficient parameters to fit to noise). This is a particular challenge on small datasets. Note that complex methods generally have strategies for reducing overfitting, such as regularization [22], which penalizes model complexity. Still, with limited or noisy data, simpler decoders can be helpful. Additionally, decoders with built in assumptions (priors) that bias or constrain the solutions can be helpful, provided that those assumptions are correct. In *Results*, we test a variety of decoders using different dataset sizes.

Hyperparameter optimization

All our decoders have fitted parameters that link the neural activity to the output variables, e.g., the weights that are fit in linear regression. Many decoders also have "hyperparameters", which relate to the design of the decoder itself. For example, neural networks can be designed to have any number of hidden units. Thus, the user needs to set the number of hidden units (the hyperparameter) before training the decoder. Often decoders have multiple hyperparameters, and different hyperparameter values can sometimes lead to greatly different performance. Thus, it is important to choose a decoder's hyperparameters carefully. When using a decoder that has hyperparameters, you should take the following steps:

- 1. The data should be split into three separate sets, a training set, testing set, and validation set.
- 2. Find the optimal hyperparameters based on validation set performance. That is, fit the decoder on the training set using many different combinations of hyperparameters, and see which hyperparameters lead to the greatest performance on the validation set. Simple methods for searching through hyperparameters are grid search and random search [23]. There are also more effective methods (e.g., [24, 25]) that can intelligently search through hyperparameters based on the performance of previously tested hyperparameters.
- 3. Using the optimal hyperparameters found in step 2, test the performance of the decoder on the test set.

Research Methods:

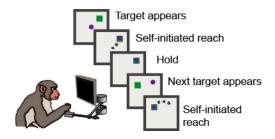
Tasks and brain regions:

Decoding movement velocity from the motor cortex and somatosensory cortex: In our "random-target" experiment [14], monkeys moved a planar manipulandum that controlled a cursor on the screen (Fig. 2a). The monkeys continuously reached to newly presented targets, with a brief hold period between reaches. After training, the monkeys were surgically implanted with 96-channel Utah electrode arrays (Blackrock Microsystems, Salt Lake City, UT) to record the extracellular activity of cortical neurons. In one experiment [14], we recorded from both primary motor cortex (M1) and dorsal premotor cortex (PMd) and combined neurons from both areas. The recording from motor cortex was 21 minutes, and contained 164 neurons. The mean and median firing rates, respectively, were 6.7 and 3.4 spikes / sec. In another experiment we recorded from area 2 of primary somatosensory cortex (S1) [26]. The recording from S1 was 51 minutes, and contained 52 neurons. The mean and median firing rates, respectively, were 9.3 and 6.3 spikes / sec. From both brain regions, we aimed to predict the x and y components of movement velocity.

Decoding position from the hippocampus: We used a dataset from CRCNS (Collaborative Research in Computational Neuroscience), in which rats chased rewards on a square platform (Fig. 2b) and extracellular recordings were made from layer CA1 of dorsal hippocampus (HC) [27, 28]. More specifically, we used dataset "hc2" and session "ec014.333". The recording from HC was 93 minutes, and contained 58 neurons. We did not use the final 20% of the recording, in which the rat had limited movement. We excluded neurons

with fewer than 100 spikes over the duration of the experiment, resulting in 46 neurons. These neurons had mean and median firing rates, respectively, of 1.7 and 0.2 spikes / sec. We aimed to predict the x and y position of the rat.

a Task for motor and somatosensory cortex



b <u>Task for hippocampus</u>



Figure 2: Tasks and Decoding Schematic

a) In the task for decoding from motor and somatosensory cortices, monkeys moved a planar manipulandum that controlled a cursor on the screen. The monkeys continuously reached to new targets as they were presented, with a brief hold period between reaches. **b)** In the task for decoding from hippocampus, rats chased rewards on a square platform.

General Decoding methods:

Decoding movement velocity from the motor cortex and somatosensory cortex: We predicted the average velocity (x and y components) in 50 ms bins. Neural spike trains used for decoding were also put into 50 ms bins. In motor cortex, we used 700 ms of neural activity (13 bins before and the concurrent bin) to predict the current movement velocity, as a primary interest in the field is investigating how motor cortex affects movement. In somatosensory cortex, we used 650 ms surrounding the movement (6 bins before, the concurrent bin, and 6 bins after), as neural activity has been shown both preceding and following movement [29]. Note that the bins used for decoding differed from above for the Kalman filter and Naïve Bayes decoders (see *Specific Decoders* below). Also, when determining how performance varied as a function of bin size (Fig. 8), we used a slightly different amount of neural data, in order to have a quantity that was divisible by many bin sizes. For motor cortex, we used 600 ms of neural activity prior to and including the current bin. For somatosensory cortex, we used 600 ms of neural activity centered on the current bin.

Decoding position from the hippocampus: We aimed to predict the position (x and y coordinates) of the rat in 200 ms bins. Neural spike trains used for decoding were also put into 200 ms bins. We used 2 seconds of surrounding neural activity (4 bins before, the concurrent bin, and 5 bins after) to predict the current position. Note that the bins differed from above for the Kalman filter (see *Specific Decoders* below). When determining how performance varied as a function of bin size (Fig. 8), we used 2 seconds of neural activity, centered on the current bin.

Scoring Metric: To determine the goodness of fit, we used
$$R^2 = 1 - \frac{\sum_{i} (\hat{y}_i - y_i)^2}{\sum_{i} (y_i - \overline{y})^2}$$
, where \hat{y}_i are the

predicted values, y_i are the true values and \overline{y} is the mean value. This formulation of R^2 (which is the fraction of variance accounted for, rather than the squared Pearson's correlation coefficient [30]) can be negative on the test set due to overfitting on the training set. The reported R^2 values are the average across the x and y components of velocity or position.

Preprocessing: The training input was normalized (z-scored). The training output was zero-centered (mean subtracted), except in support vector regression, where the output was z-scored. The validation/testing inputs and outputs were preprocessed using the preprocessing parameters from the training set.

Cross-validation: When determining the R^2 for every method (Fig. 3), we used 10 fold cross-validation. For each fold, we split the data into a training set (80% of data), a contiguous validation set (10% of data), and a contiguous testing set (10% of data). For each fold, decoders were trained to minimize the mean squared error between the predicted and true velocities/positions of the training data. We found the algorithm hyperparameters that led to the highest R^2 on the validation set using Bayesian optimization [24]. That is, we fit many models on the training set with different hyperparameters and calculated the R^2 on the validation set. Then, using the hyperparameters that led to the highest validation set R^2 , we calculated the R^2 value on the testing set. Error bars on the test set R^2 values were computed across cross-validation folds. Because the training sets on different folds were overlapping, computing the SEM as σ/\sqrt{n} (where σ is the standard deviation and n is the number of folds) would have underestimated the size of the error bars [31]. We thus

calculated the SEM as $\sigma * \sqrt{\frac{1}{n} + \frac{1}{n-1}}$, which takes into account that the estimates across folds are not independent [31].

Bootstrapping: When determining how performance scaled as a function of data size (Figs. 5,6), we used single test and validation sets, and varied amounts of training data that directly preceded the validation set. We did not do this on 10 cross-validation folds due to long run-times. The test and validation sets were 5 minutes long for motor and somatosensory cortices, and 7.5 minutes for hippocampus. To get error bars, we resampled from the test set. Because of the high correlation between temporally adjacent samples, we didn't resample randomly from all examples (which would create highly correlated resamples). Instead, we separated the test set into 20 temporally distinct subsets, S_1 - S_{20} (i.e., S_1 is from t=1 to t=T/20, S_2 is from t=T/20 to t=2T/20, etc., where T is the end time), to ensure that the subsets were more nearly independent of each other. We then resampled combinations of these 20 subsets (e.g., S_5 , S_{13} , ... S_2) 1000 times to get confidence intervals of R^2 values.

Specific Decoders:

Wiener Filter: The Wiener filter uses multiple linear regression to predict the output from multiple time bins of every neurons' spikes. That is, the output is assumed to be a linear mapping of the number of spikes in the relevant time bins from every neuron (Fig. 1a,b). We used separate models to predict the x and y components of the kinematics.

Wiener Cascade: The Wiener cascade (also known as a linear-nonlinear model) fits a linear regression (the Wiener filter) followed by a fitted static nonlinearity (e.g., [32]). This allows for a nonlinear relationship between the input and the output, and assumes that this nonlinearity is purely a function of the linear output. Here, as in the Wiener Filter, the input was neurons' spike rates over relevant time bins. The nonlinear component was a polynomial with degree determined on the validation set. Separate models were used to predict the x and y components of the kinematics.

Support Vector Regression: In support vector machine regression (SVR) [33], the inputs are projected into a higher-dimensional space using a nonlinear kernel, and then linearly mapped from this space to the output to minimize an objective function [33]. Here, we used standard support vector regression (SVR) with a radial basis function kernel to predict the kinematics from the neurons' spike rates in each bin. We set hyperparameters for the penalty of the error term and the maximum number of iterations. Separate models were used to predict the x and y components of the kinematics.

XGBoost: XGBoost (Extreme Gradient Boosting) [34] is an implementation of gradient boosted trees. Tree-based methods sequentially split the input space into many discrete parts (visualized as branches on a tree for each split), in order to assign each final "leaf" (a portion of input space that is not split any more) a value in output space [35]. We fit many regression trees, which are trees that predict continuous output values. "Gradient boosting" refers to fitting each subsequent regression tree to the residuals of the previous fit. Here, we used XGBoost to predict the kinematics from the neurons' spike rates in each bin. We set hyperparameters for the maximum depth of the tree, number of trees, and learning rate. We fit separate models to predict the x and y components.

Feedforward Neural Network: A feedforward neural net connects the inputs to sequential layers of hidden units, which then connect to the output. Each layer connects to the next (e.g., the input layer to the first hidden layer, or the first to second hidden layers) via linear mappings followed by nonlinearities. Note that the Wiener cascade is a special case of a neural network with no hidden layers. Using the Keras library [36], we created a fully connected (dense) feedforward neural network with 2 hidden layers and rectified linear unit [37] activations after each hidden layer. We required the number of hidden units in each layer to be the same. We set hyperparameters for the number of hidden units in the layers, amount of dropout [38], and number of training epochs. We used the Adam algorithm [39] as the optimization routine. This neural network, and all neural networks below had two output units. That is, the same network predicted the x and y components together, rather than separately. The input was still the number of spikes in each bin from every neuron. Note that we refer to feedforward neural networks as a "modern" technique, despite their having been used for many decades, due to their current resurgence and the modern methods for training.

Simple RNN: In a standard recurrent neural network (RNN), the hidden state is a linear combination of the inputs and the previous hidden state. This hidden state is then run through an output nonlinearity, and linearly mapped to the output. RNNs, unlike feedforward neural networks, allow temporal changes in the system to be modeled explicitly. Here, using the Keras library [36], we created a neural network architecture in which the spiking inputs from all neurons were fed into a standard recurrent neural network (Fig. 1c). The units from this recurrent layer were fed through rectified linear unit nonlinearities, and fully connected to an output layer with two units (x and y velocity or position components). We set hyperparameters for the number of units, amount of dropout, and number of training epochs. We used RMSprop [40] as the optimization routine.

Gated Recurrent Unit: Gated recurrent units (GRUs) [41] are a more complex type of recurrent neural network. It has gated units, which in practice allow for better learning of long-term dependencies. Almost all implementation methods were the same as for the simple RNN, except Gated Recurrent Units were used instead. An implementation difference is that the units from the recurrent layers were fed through hyperbolic tangent (tanh) activations (as is standard for GRUs) rather than rectified linear unit activations.

Long Short Term Memory Network: Like the GRU, the long short term memory (LSTM) network [42] is a more complex recurrent neural network with gated units that further improve the capture of long-term dependencies. The LSTM has more parameters than the GRU. All implementation methods were the same as for GRUs, except LSTM units were used instead.

Ensemble: Ensemble techniques combine the predictions from several methods, and thus have the potential to leverage their different benefits. We used the predictions from all decoders except the Kalman filter and Naïve Bayes decoders (which have different formats). We combined the predictions from the above 8 methods using a feedforward neural network. That is, the 8 methods' predictions were provided as input into a feedforward neural network that we trained to predict the true output. This was done separately for the x and y components of the position or velocity.

Kalman Filter: Our Kalman filter for neural decoding was based on [17]. In the Kalman filter, the hidden state at time t is a linear function of the hidden state at time t-1, plus a matrix characterizing the uncertainty. For neural decoding, the hidden state is the kinematics (x and y components of position, velocity, and acceleration). Note that even though we only aim to predict position or velocity, all kinematics are included because this allows for better prediction. More formally,

$$y_t = Ay_{t-1} + w$$

where y_t and y_{t-1} are 6 x 1 vectors, A is a 6 x 6 matrix, and w is sampled from a normal distribution, N(0, W), with mean 0 and covariance W. W is the 6 x 6 uncertainty matrix.

The observation (measurement) at time t^* is a linear function of the hidden state at time t (plus noise). For neural decoding, the measurement is the neural activity. Note that we allowed a lag between the neural data and predicted kinematics, which is why we use t^* for the time of the neural activity. More formally,

$$x_{t*} = Hy_t + q$$

where x_{t*} is an $N \times 1$ vector, H is an $N \times 6$ matrix, and q is sampled from a normal distribution, N(0,Q), with mean 0 and covariance Q. Q is the $N \times N$ measurement noise matrix.

During training, A, H, W, and Q are empirically fit on the training set using maximum likelihood estimation. When making predictions, to update the estimated hidden state at a given time point, the updates derived from the current measurement and the previous hidden states are combined. During this combination, the noise matrices give a higher weight to the less uncertain information. See [17] or our code for the update equations (note that x and y have different notation in [17]).

We had one hyperparameter which differed from the standard implementation [17]. We divided the noise matrix associated with the transition in kinematic states, W, by the hyperparameter scalar C, which allowed weighting the neural evidence and kinematic transitions differently. The rationale for this addition is that neurons have temporal correlations, which make it desirable to have a parameter that allows changing the weight of the new neural evidence. The introduction of this parameter made a big difference for the hippocampus dataset (Fig. S1). We also allowed for a lag between the neural data and predicted kinematics. The lag and hyperparameter were determined based on validation set performance.

Naïve Bayes: The Naïve Bayes decoder is a type of Bayesian decoder that determines the probabilities of different outcomes, and it then predicts the most probable. Briefly, it fits an encoding model to each neuron, makes conditional independence assumptions about neurons, and then uses Bayes' rule to create a decoding model from the encoding models. This probabilistic framework can incorporate prior information about the variables.

We used a Naïve Bayes decoder similar to the one implemented in [5]. We first fit an encoding model (tuning curve) using the output variables. Let $f_i(s)$ be the value of the tuning curve (the expected number of spikes) for neuron i at the output variables s. Note that s is a vector containing the two output variables we are predicting (x and y positions/velocities). We assume the number of recorded spikes in the given bin, r_i , is generated from the tuning curve with Poisson statistics:

$$P(r_i|\mathbf{s}) = \frac{\exp\left[-f_i(\mathbf{s})\right]f_i(\mathbf{s})^{r_i}}{r_i!}$$

We also assume that all the neurons' spike counts are conditionally independent given the output variables, so that:

$$P(r|s) \propto \prod_{i} P(r_{i}|s)$$

where r is a vector with the spike counts of all neurons. Bayes' rule can then be used to determine the likelihood of the output variables given the spike counts of all neurons:

$$P(s|r) \propto P(r|s)P(s)$$

where P(s) is the probability distribution of the output variables. To help with temporal continuity of decoding, we want our probabilistic model to include how the output variables at one time step depend on the output variables at the previous time step: $P(s_t|s_{t-1})$. Thus, we can more generally write, using Bayes' rule as before:

$$P(s_t|r_{t*},s_{t-1}) \propto P(r_{t*}|s_t)P(s_{t-1}|s_t)P(s_t)$$

Note that we use r_{t*} rather than r_t because we use neural responses from multiple time bins to predict the current output variables. The above formula assumes that r_{t*} and s_{t-1} are independent, conditioned on s_t . The final decoded stimulus in a time bin is: $\operatorname{argmax}_{s_t} P(s_t | r_{t*}, s_{t-1})$.

 $P(s_{t-1}|s_t)$ was determined as follows. Let Δs be the Euclidean distance in s from one time step to the next. We fit $P(\Delta s)$ as a Gaussian using data from the training set. $P(s_{t-1}|s_t)$ was approximated as $P(\Delta s_t)$. That is, the probability of going from one output state to another was only based on the distance between the output states, not the output state itself.

Additionally, including P(s) based on the distribution of output variables in the training set did not improve performance on the validation set. This could be because the probability distribution differed between the training and validation/testing sets, or because the distribution of output variables was approximately uniform in our tasks. Thus, we simply used a uniform prior.

In our calculations, we discretize s into a 100 x 100 grid going from the minimum to maximum of the output variables. When increasing the decoding resolution of the output variables, we did not see a meaningful change in decoding accuracy.

Our tuning curves had the format of a Poisson generalized quadratic model [43], which improved the performance over generalized linear models on validation datasets.

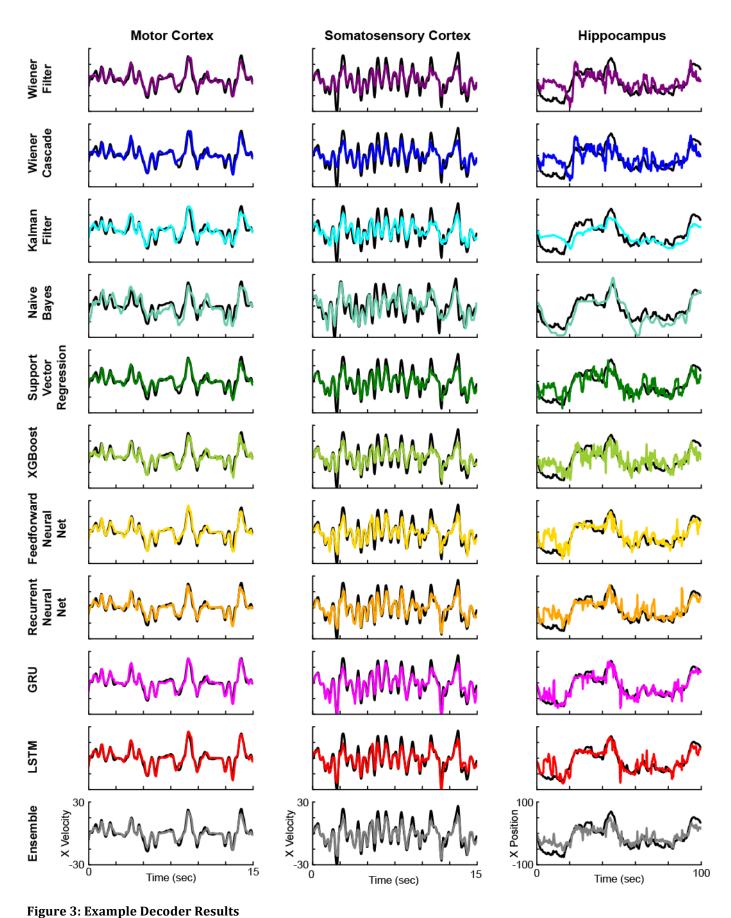
On the hippocampus dataset, we used the total number of spikes over the same time interval as we used for the other decoders (4 bins before, the concurrent bin, and 5 bins after). Note that using a single time bin of spikes led to very poor performance. On the motor cortex and somatosensory cortex datasets, the naïve Bayes decoder gave very poor performance regardless of the bins used. We ultimately used bins that gave the best performance on a validation set: 2 bins before and the concurrent bin for the motor cortex dataset; 1 bin before, the concurrent bin, and 1 bin after for the somatosensory cortex dataset.

Code: Python code for all methods is available at https://github.com/KordingLab/Neural_Decoding

Results

We investigated how the choice of machine learning technique affects decoding performance using a number of common machine learning methods. These ranged from historical linear techniques (e.g., the Wiener filter) to modern machine learning techniques (e.g., neural networks and ensembles of techniques). We tested the performance of these techniques across datasets from motor cortex, somatosensory cortex, and hippocampus (Fig. 2).

We aimed to understand the performance of the methods when fit to neural data. First, in order to get a qualitative impression of the performance, we plotted the output of each decoding method for each of the three datasets (Fig. 3). In these examples, the modern methods, such as the LSTM and ensemble, appeared to outperform traditional methods. We next quantitatively compared the methods' performances (Fig. 4), which confirmed these results. In particular, neural networks and the ensemble led to the best performance, while the Wiener or Kalman Filter led to the worst performance. In fact, the LSTM decoder explained over 40% of the unexplained variance from a Wiener filter (R²'s of 0.88, 0.86, 0.62 vs. 0.78, 0.75, 0.35). Interestingly, while the Naïve Bayes decoder performed relatively well when predicting position in the hippocampus dataset (mean R² just slightly less than the neural networks), it performed very poorly when predicting hand velocities in the other two datasets. Another interesting finding is that the feedforward neural network did almost as well as the LSTM in all brain areas. Across cases, the ensemble method added a reliable, but small increase to the explained variance. Overall, modern machine learning methods led to significant increases in predictive power.



Example decoding results from motor cortex (left), somatosensory cortex (middle), and hippocampus (right), for all eleven methods (top to bottom). Ground truth traces are in black, while decoder results are in various colors.

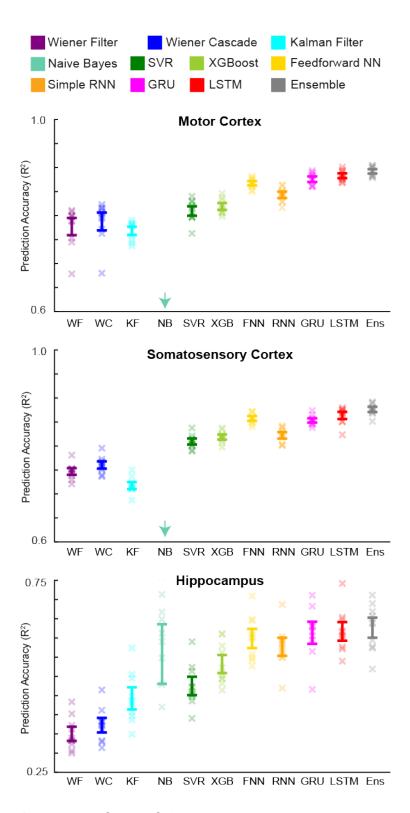


Figure 4: Decoder Result Summary

 R^2 values are reported for all decoders (different colors) for each brain area (top to bottom). Error bars represent the mean +/- SEM across cross-validation folds. X's represent the R^2 values of each cross-validation fold. The NB decoder had mean R^2 values of 0.26 and 0.36 (below the minimum y-axis value) for the motor and somatosensory cortex datasets, respectively. Note the different y-axis limits for the hippocampus dataset in this and all subsequent figures.

We chose a representative subset of the ten methods to pursue further questions about particular aspects of neural data analysis: the feedforward neural network and LSTM (two modern methods that worked particularly well), along with the Wiener and Kalman filters. The improved predictive performance of the modern methods is likely due to their greater complexity. However, this greater complexity may make these methods unsuitable for smaller amounts of data. Thus, we tested performance with varying amounts of training data. With only 2 minutes of data for motor and somatosensory cortices, and 15 minutes of hippocampus data, both modern methods outperformed both traditional methods (Fig. 5,6). When decreasing the amount of training data further, to only 1 minute for motor and somatosensory cortices and 7.5 minutes for hippocampus data, the Kalman filter performance was sometimes comparable to the modern methods, but the modern methods significantly outperformed the Wiener Filter (Fig. 6). Thus, even for limited recording times, modern machine learning methods can yield significant gains in decoding performance.

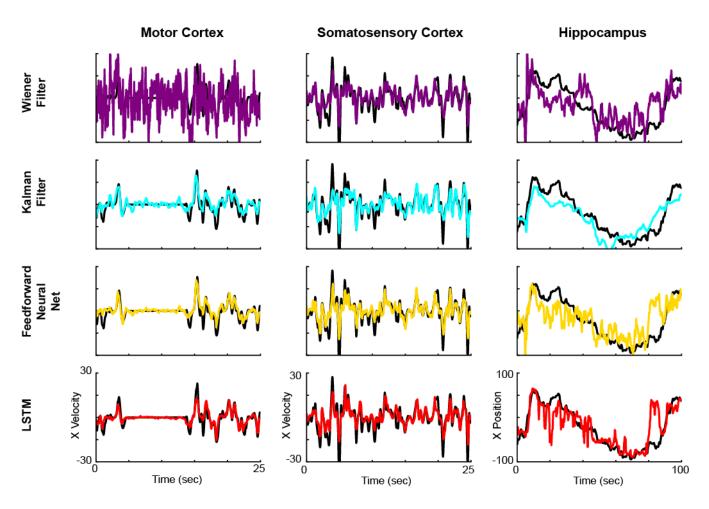


Figure 5: Example results with limited training data

Using only 2 minutes of training data for motor cortex and somatosensory cortex, and 15 minutes of training data for hippocampus, we trained two traditional methods (Wiener filter and Kalman filter), and two modern methods (feedforward neural network and LSTM). Example decoding results are shown from motor cortex (left), somatosensory cortex (middle), and hippocampus (right), for these methods (top to bottom). Ground truth traces are in black, while decoder results are in the same colors as previous figures.

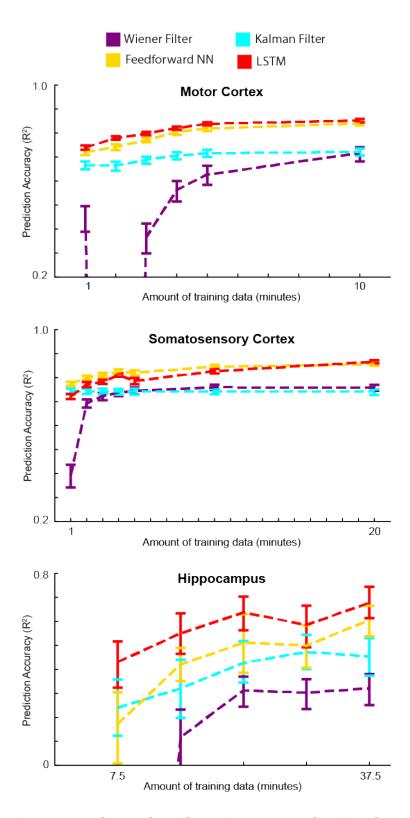
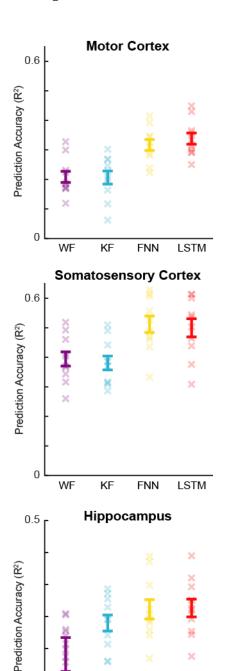


Figure 6: Decoder results with varying amounts of training data

Using varying amounts of training data, we trained two traditional methods (Wiener filter and Kalman filter), and two modern methods (feedforward neural network and LSTM). R^2 values are reported for these decoders (different colors) for each brain area (top to bottom). Error bars are 68% confidence intervals (meant to approximate the SEM) produced via bootstrapping, as we used a single test set. Values with negative R^2 s were not shown.

Besides limited recording times, neural data is often limited in the number of recorded neurons. Thus, we compared methods using a subset of only 10 neurons. For motor and somatosensory data, despite a general decrease in performance for all decoding methods, the modern methods significantly outperformed the traditional methods (Fig. 7). For the hippocampus dataset, no method predicted well (mean $R^2 < 0.25$) with only 10 neurons. This is likely because 10 sparsely firing neurons (median firing of HC neurons was ~ 0.2 spikes / sec) did not contain enough information about the entire space of positions. However, in most scenarios, with limited neurons and for limited recorded times, modern machine learning methods can be advantageous.



-0.1

KF

FNN

LSTM

Figure 7: Decoder results with fewer neurons

Using only 10 neurons, we trained two traditional methods (Wiener filter and Kalman filter), and two modern methods (feedforward neural network and LSTM). We used the same testing set as in Fig. 6, and the largest training set from Fig. 6. R² values are reported for these decoders (different colors) for each brain area (top to bottom). Error bars represent the mean +/- SEM of multiple repetitions with different subsets of 10 neurons. X's represent the R² values of each repetition.

While we have so far only tested decoding accuracy using a set bin size, different decoding applications may require different temporal resolutions. It is therefore also important to compare decoding accuracy with varying bin sizes. For the motor and somatosensory cortex datasets, we tested bin sizes from 10-100 ms. For the hippocampus dataset, we tested bin sizes from 30-400 ms. Surprisingly, for all methods besides the Kalman filter, decoding performance remained consistent across all the tested bin sizes (Fig. 8). For the Kalman filter, decoding performance increased as bin sizes became larger, likely because the Kalman filter used a single bin of neural data to make predictions, while the other methods used many bins. Importantly, for all bin sizes, the modern methods led to increased decoding accuracy. Thus, modern machine learning methods remain advantageous regardless of the temporal resolution.

All our previous results used hyperparameter optimization. While we strongly encourage a thorough hyperparameter optimization, a user with limited time might not be able to do this, or might just do a limited hyperparameter search. Thus, it is helpful to know how sensitive results may be to varying hyperparameters. We tested the performance of the feedforward neural network while varying two hyperparameters: the number of units and the dropout rate. We held the third hyperparameter, the number of training epochs, constant at 10. We found that the performance of the neural network was generally robust to large changes in the hyperparameters (Fig. 9). As an example, for the somatosensory cortex dataset, the peak performance of the neural network was R^2 =0.86 with 1000 units and 0 dropout, and virtually the same (R^2 =0.84) with 300 units and 30% dropout. Even when using limited data, neural network performance was robust to hyperparameter changes. For instance, when training the somatosensory cortex dataset with 1 minute of training data, the peak performance was R^2 =0.77 with 700 units and 20% dropout. A network with 300 units and 30% dropout had R^2 =0.75. Note that the hippocampus dataset, in particular when using limited training data, did have greater variability, emphasizing the importance of hyperparameter optimization on sparse datasets. However, for most datasets, researchers should not be concerned that slightly non-optimal hyperparameters will lead to largely degraded performance.

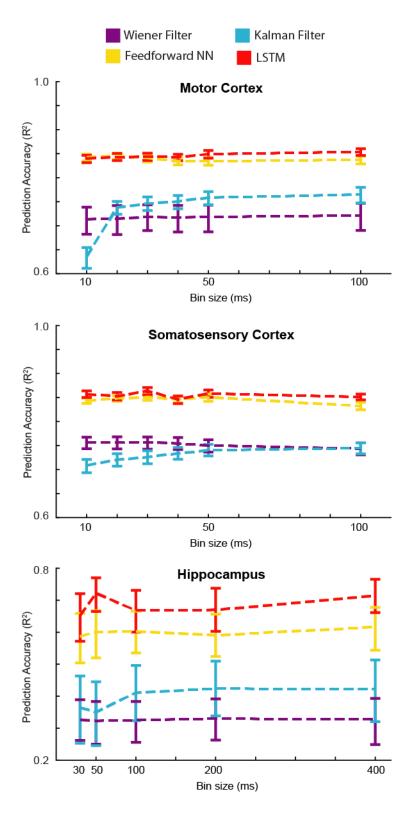


Figure 8: Decoder results with different bin sizes

Using varying bin sizes, we trained two traditional methods (Wiener filter and Kalman filter), and two modern methods (feedforward neural network and LSTM). We used the same testing set as in Fig. 6, and the largest training set from Fig. 6. R² values are reported for these decoders (different colors) for each brain area (top to bottom). Error bars are 68% confidence intervals (meant to approximate the SEM) produced via bootstrapping, as we used a single test set.

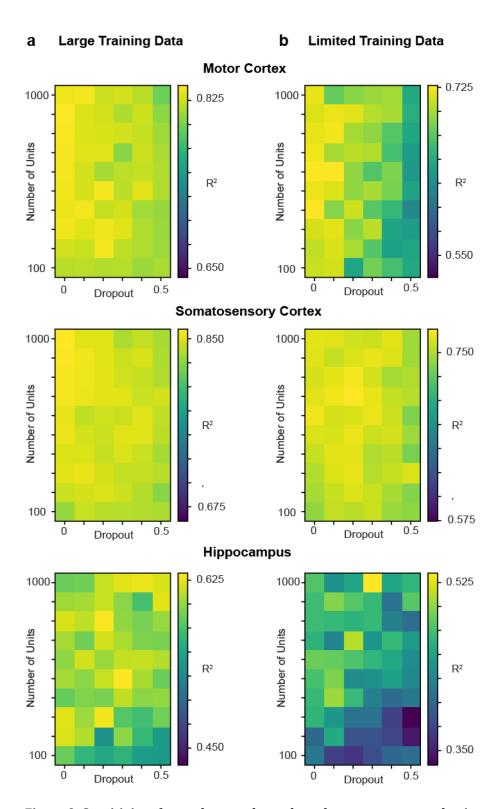


Figure 9: Sensitivity of neural network results to hyperparameter selection

In a feedforward neural network, we varied the number of hidden units per layer (in increments of 100) and the proportion of dropout (in increments of 0.1), and evaluated the decoder's performance on all three datasets (top to bottom). As described in *Methods*, the neural network had two hidden layers, each with the same number of hidden units. The number of training epochs was kept constant at 10. The colors show the R^2 on the test set, and each panel's colors were put in the range: [maximum R^2 – 0.2 , maximum R^2]. a) We used a large amount of training data (the maximum amount used in Fig. 6), which was 10, 20, and 37.5 minutes of data for the motor cortex, somatosensory cortex, and hippocampus datasets, respectively. b) Same results for a limited amount of training data: 1, 1, and 15 minutes of data for the motor cortex, somatosensory cortex, and hippocampus datasets, respectively.

Discussion:

We tested the performance of a large number of decoding techniques on three different neural decoding problems and provided a tutorial on their use. We found that, across datasets, neural networks outperformed traditional methods. An ensemble method provided only minor additional predictive power. The enhanced performance of neural networks even persisted for small datasets with as little as one minute of training data or as few as 10 neurons. Moreover, neural networks were robust to decoding at many different temporal resolutions, and to a wide range of hyperparameters.

We find it particularly interesting that the neural network methods worked so well with limited data, which is counter to the common perception. We believe the explanation is simply the size of the networks. For instance, our networks have on the order of 10^5 parameters, while common networks for image classification (e.g., [44]) can have on the order of 10^8 parameters. Thus, the reasonable size of our networks (hundreds of hidden units) likely allowed for excellent prediction with limited data [45]. Moreover, the fact that the tasks we used had a low-dimensional structure, and therefore the neural data was also likely low dimensional [46], might allow high decoding performance with limited data.

It is also intriguing that the feedforward neural network did almost as well as the LSTM and better than the standard RNN, considering the recent attention to treating the brain as a dynamical system [47]. For the motor and somatosensory cortex decoding, it is possible that the highly trained monkeys yielded a stereotyped temporal relationship between neural activity and movement that a feedforward neural network could effectively capture. It would be interesting to compare the performance of feedforward and recurrent neural networks on less constrained behavior.

In order to find the best hyperparameters for the decoding algorithms, we used a Bayesian optimization routine [24] to search the hyperparameter space (see *Methods*). Still, it is possible that some of the decoding algorithms did not use the optimal hyperparameters, which would have lowered overall accuracy. Moreover, for several methods, we did not fit all available hyperparameters. We did this in order to simplify the their use, decrease computational runtime during hyperparameter optimization, and because additional hyperparameters did not appear to improve accuracy. For example, for the neural nets we used dropout but not L1 or L2 regularization, and for XGBoost we used less than half the available hyperparameters designed to avoid overfitting. While our preliminary testing with additional hyperparameters did not appear to significantly change the results, it is possible that we have not achieved optimal performance.

While we have tested standard algorithms on three different datasets, it is possible that the relative performance of algorithms differs on other datasets. However, many datasets in neuroscience share basic properties with those we used. Most are similar in length (tens of minutes to a couple hours), simply because the length of a recording session is usually limited by the patience of both the animal and the experimentalist. Moreover, most variables of interest have similar relevant timescales, where movement, speech, vision, and many other phenomena unfold on a timescale of hundreds of milliseconds to seconds. We thus expect that similar results would be obtained for other datasets. If anything, using datasets of more complex, high-dimensional behaviors would likely increase the benefits of using modern machine learning methods.

We have decoded from spiking data, but it is possible that the problem of decoding from other data modalities is different. One main driver of a difference may be the distinct levels of noise. For example, fMRI signals have far higher noise levels than spikes. As the noise level goes up, linear techniques become more appropriate, which may ultimately lead to a situation where the traditional linear techniques become superior. Applying the same analyses we did here across different data modalities is an important next step.

All our decoding was done "offline," meaning that the decoding occurred after the recording, and was not part of a control loop. This type of decoding is useful for determining how information in a particular brain area relates to an external variable. However, for engineering applications such as BMIs [48, 49], the goal is to decode information (e.g., predict movements) in real time. Our results here may not apply as directly to

online decoding situations, since the subject is ultimately able to adapt to imperfections in the decoder. In that case, even relatively large decoder performance differences may be irrelevant. Plus, there are additional challenges in online applications, such as non-stationary inputs (e.g. due to electrodes shifting in the brain) [18, 50]. Finally, online applications are concerned with computational runtime, which we have not addressed here. In the future, it would be valuable to test modern machine learning techniques for decoding in online applications (as in [50, 51]).

While modern machine learning methods provide an increase in decoding accuracy, it is important to be careful with the scientific interpretation of decoding results. Decoding can tell us how much information a neural population has about a variable *X*. However, high decoding accuracy does not mean that a brain area is directly involved in processing *X*, or that *X* is the purpose of the brain area. For example, with a powerful decoder, it could be possible to accurately classify images based on recordings from the retina, since the retina has information about all visual space. However, this does not mean that the primary purpose of the retina is image classification. Moreover, even if the neural signal comes before the external variable, it does not mean that it is causally involved. For example, information could be in somatosensory cortex prior to movement due to an efference copy from motor cortex. Thus, researchers should constrain interpretations to being about the information in neural populations, and how it may vary across brain regions, experimental conditions, or time intervals.

In this study, we decoded continuous-valued variables. However, these same methods can be used for classification tasks, which often use classic decoders such as logistic regression and support vector machines. While we have not demonstrated the benefit of modern machine learning methods for classification, our available code can easily be modified to perform classification rather than regression.

Neural engineering has a history of developing specialized algorithms meant to increase the performance of decoders [52-54]. However, these algorithms are not typically tested against state of the art machine learning algorithms. Along with this manuscript, we have released a package to perform neural decoding using all the described methods, making it easy to compare with any new algorithm. Our hunch is that it will be hard for specialized algorithms to compete with the standard algorithms developed by the massive machine learning community.

Acknowledgements:

We would like to thank Pavan Ramkumar for help with code development. For funding, JG was supported by NIH F31 EY025532 and NIH T32 HD057845. MP was supported by NIH F31 NS092356 and NIH T32 HD07418. RC was supported by NIH R01 NS095251 and DGE-1324585. LM was supported by NIH R01 NS074044 and NIH R01 NS095251. KK was supported by NIH R01 NS074044, NIH R01 NS063399 and NIH R01 EY021579.

References

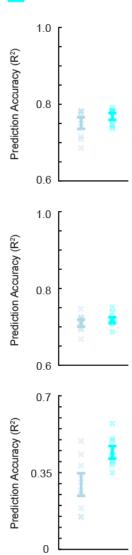
- 1. Serruya MD, Hatsopoulos NG, Paninski L, Fellows MR, Donoghue JP. Brain-machine interface: Instant neural control of a movement signal. Nature. 2002;416(6877):141-2.
- 2. Ethier C, Oby ER, Bauman MJ, Miller LE. Restoration of grasp following paralysis through brain-controlled stimulation of muscles. Nature. 2012;485(7398):368-71. doi: 10.1038/nature10987. PubMed PMID: 22522928; PubMed Central PMCID: PMCPMC3358575.
- 3. Baeg E, Kim Y, Huh K, Mook-Jung I, Kim H, Jung M. Dynamics of population code for working memory in the prefrontal cortex. Neuron. 2003;40(1):177-88.
- 4. Ibos G, Freedman DJ. Sequential sensory and decision processing in posterior parietal cortex. eLife. 2017;6.
- 5. Zhang K, Ginzburg I, McNaughton BL, Sejnowski TJ. Interpreting neuronal population activity by reconstruction: unified framework with application to hippocampal place cells. J Neurophysiol. 1998;79(2):1017-44.
- 6. Davidson TJ, Kloosterman F, Wilson MA. Hippocampal replay of extended experience. Neuron. 2009;63(4):497-507.
- 7. Raposo D, Kaufman MT, Churchland AK. A category-free neural population supports evolving demands during decision-making. Nat Neurosci. 2014;17(12):1784-92.
- 8. Rich EL, Wallis JD. Decoding subjective decisions from orbitofrontal cortex. Nat Neurosci. 2016;19(7):973-80.
- 9. Hung CP, Kreiman G, Poggio T, DiCarlo JJ. Fast readout of object identity from macaque inferior temporal cortex. Science. 2005;310(5749):863-6.
- 10. Quiroga RQ, Snyder LH, Batista AP, Cui H, Andersen RA. Movement intention is better predicted than attention in the posterior parietal cortex. J Neurosci. 2006;26(13):3615-20.
- 11. Hernández A, Nácher V, Luna R, Zainos A, Lemus L, Alvarez M, et al. Decoding a perceptual decision process across cortex. Neuron. 2010;66(2):300-14.
- 12. van der Meer MA, Johnson A, Schmitzer-Torbert NC, Redish AD. Triple dissociation of information processing in dorsal striatum, ventral striatum, and hippocampus on a learned spatial decision task. Neuron. 2010;67(1):25-32.
- 13. Dekleva BM, Ramkumar P, Wanda PA, Kording KP, Miller LE. Uncertainty leads to persistent effects on reach representations in dorsal premotor cortex. eLife. 2016;5:e14316. doi: 10.7554/eLife.14316.
- 14. Glaser JI, Perich MG, Ramkumar P, Miller LE, Kording KP. Population Coding Of Conditional Probability Distributions In Dorsal Premotor Cortex. bioRxiv. 2017:137026.
- 15. Weygandt M, Blecker CR, Schäfer A, Hackmack K, Haynes J-D, Vaitl D, et al. fMRI pattern recognition in obsessive–compulsive disorder. Neuroimage. 2012;60(2):1186-93.
- 16. Collinger JL, Wodlinger B, Downey JE, Wang W, Tyler-Kabara EC, Weber DJ, et al. High-performance neuroprosthetic control by an individual with tetraplegia. The Lancet. 2013;381(9866):557-64.
- 17. Wu W, Black MJ, Gao Y, Serruya M, Shaikhouni A, Donoghue J, et al., editors. Neural decoding of cursor motion using a Kalman filter. Adv Neural Inf Process Syst; 2003.
- 18. Wu W, Hatsopoulos NG. Real-time decoding of nonstationary neural activity in motor cortex. IEEE Trans Neural Syst Rehabil Eng. 2008;16(3):213-22.
- 19. Gilja V, Nuyujukian P, Chestek CA, Cunningham JP, Byron MY, Fan JM, et al. A high-performance neural prosthesis enabled by control algorithm design. Nat Neurosci. 2012;15(12):1752.
- 20. Barbieri R, Wilson MA, Frank LM, Brown EN. An analysis of hippocampal spatio-temporal representations using a Bayesian algorithm for neural spike train decoding. IEEE Trans Neural Syst Rehabil Eng. 2005;13(2):131-6.
- 21. Kloosterman F, Layton SP, Chen Z, Wilson MA. Bayesian decoding using unsorted spikes in the rat hippocampus. J Neurophysiol. 2013;111(1):217-27.
- 22. Friedman J, Hastie T, Tibshirani R. The elements of statistical learning: Springer series in statistics New York; 2001.
- 23. Bergstra J, Bengio Y. Random search for hyper-parameter optimization. Journal of Machine Learning Research. 2012;13(Feb):281-305.

- 24. Snoek J, Larochelle H, Adams RP, editors. Practical bayesian optimization of machine learning algorithms. Adv Neural Inf Process Syst; 2012.
- 25. Hyperopt. http://hyperopt.github.io/hyperopt/.
- 26. Benjamin AS, Fernandes HL, Tomlinson T, Ramkumar P, VerSteeg C, Miller L, et al. Modern machine learning far outperforms GLMs at predicting spikes. bioRxiv. 2017:111450.
- 27. Mizuseki K, Sirota A, Pastalkova E, Buzsáki G. Theta oscillations provide temporal windows for local circuit computation in the entorhinal-hippocampal loop. Neuron. 2009;64(2):267-80.
- 28. Mizuseki K, Sirota A, Pastalkova E, Buzsáki G. Multi-unit recordings from the rat hippocampus made during open field foraging. 2009. doi: http://dx.doi.org/10.6080/K0Z60KZ9.
- 29. London BM, Miller LE. Responses of somatosensory area 2 neurons to actively and passively generated limb movements. J Neurophysiol. 2013;109(6):1505-13.
- 30. Fagg AH, Ojakangas GW, Miller LE, Hatsopoulos NG. Kinetic trajectory decoding using motor cortical ensembles. IEEE Trans Neural Syst Rehabil Eng. 2009;17(5):487-96.
- 31. Nadeau C, Bengio Y, editors. Inference for the generalization error. Adv Neural Inf Process Syst; 2000.
- 32. Pohlmeyer EA, Solla SA, Perreault EJ, Miller LE. Prediction of upper limb muscle activity from motor cortical discharge during reaching. Journal of neural engineering. 2007;4(4):369.
- 33. Chang C-C, Lin C-J. LIBSVM: a library for support vector machines. ACM Transactions on Intelligent Systems and Technology (TIST). 2011;2(3):27.
- 34. Chen T, Guestrin C, editors. Xgboost: A scalable tree boosting system. Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining; 2016: ACM.
- 35. Breiman L. Classification and regression trees: Routledge; 2017.
- 36. Chollet F. Keras. 2015.
- 37. Glorot X, Bordes A, Bengio Y, editors. Deep sparse rectifier neural networks. Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics; 2011.
- 38. Srivastava N, Hinton GE, Krizhevsky A, Sutskever I, Salakhutdinov R. Dropout: a simple way to prevent neural networks from overfitting. Journal of Machine Learning Research. 2014;15(1):1929-58.
- 39. Kingma D, Ba J. Adam: A method for stochastic optimization. arXiv preprint arXiv:14126980. 2014.
- 40. Tieleman T, Hinton G. Lecture 6.5-RmsProp: Divide the gradient by a running average of its recent magnitude. COURSERA: Neural Networks for Machine Learning. 2012.
- 41. Cho K, Van Merriënboer B, Gulcehre C, Bahdanau D, Bougares F, Schwenk H, et al. Learning phrase representations using RNN encoder-decoder for statistical machine translation. arXiv preprint arXiv:14061078. 2014.
- 42. Hochreiter S, Schmidhuber J. Long short-term memory. Neural Comput. 1997;9(8):1735-80.
- 43. Park IM, Archer EW, Priebe N, Pillow JW, editors. Spectral methods for neural characterization using generalized quadratic models. Adv Neural Inf Process Syst; 2013.
- 44. Krizhevsky A, Sutskever I, Hinton GE, editors. Imagenet classification with deep convolutional neural networks. Adv Neural Inf Process Syst; 2012.
- 45. Zhang C, Bengio S, Hardt M, Recht B, Vinyals O. Understanding deep learning requires rethinking generalization. arXiv preprint arXiv:161103530. 2016.
- 46. Gao P, Trautmann E, Byron MY, Santhanam G, Ryu S, Shenoy K, et al. A theory of multineuronal dimensionality, dynamics and measurement. bioRxiv. 2017:214262.
- 47. Shenoy KV, Sahani M, Churchland MM. Cortical control of arm movements: a dynamical systems perspective. Annu Rev Neurosci. 2013;36:337-59. doi: 10.1146/annurev-neuro-062111-150509. PubMed PMID: 23725001.
- 48. Kao JC, Stavisky SD, Sussillo D, Nuyujukian P, Shenoy KV. Information systems opportunities in brain–machine interface decoders. Proceedings of the IEEE. 2014;102(5):666-82.
- 49. Nicolas-Alonso LF, Gomez-Gil J. Brain computer interfaces, a review. Sensors. 2012;12(2):1211-79.
- 50. Sussillo D, Stavisky SD, Kao JC, Ryu SI, Shenoy KV. Making brain–machine interfaces robust to future neural variability. Nature communications. 2016;7:13749.
- 51. Sussillo D, Nuyujukian P, Fan JM, Kao JC, Stavisky SD, Ryu S, et al. A recurrent neural network for closed-loop intracortical brain–machine interface decoders. Journal of neural engineering. 2012;9(2):026027.

- 52. Kao JC, Nuyujukian P, Ryu SI, Churchland MM, Cunningham JP, Shenoy KV. Single-trial dynamics of motor cortex and their applications to brain-machine interfaces. Nature communications. 2015;6.
- 53. Corbett E, Perreault E, Koerding K, editors. Mixture of time-warped trajectory models for movement decoding. Adv Neural Inf Process Syst; 2010.
- 54. Kao JC, Nuyujukian P, Ryu SI, Shenoy KV. A high-performance neural prosthesis incorporating discrete state selection with hidden Markov models. IEEE Trans Biomed Eng. 2017;64(4):935-45.

Kalman Filter (with no extra parameter)

Kalman Filter (with extra parameter)



Supplemental Figure 1. Kalman Filter Versions

 R^2 values are reported for different versions of the Kalman Filter for each brain area (top to bottom). On the left (in light blue), the Kalman Filter is implemented as in [17]. On the right (in cyan), the Kalman Filter is implemented with an extra parameter that scales the noise matrix associated with the transition in kinematic states (see *Methods*). This version with the extra parameter is the one used in the main text. Error bars represent the mean +/- SEM across cross-validation folds. X's represent the R^2 values of each cross-validation fold. Note the different y-axis limits for the hippocampus dataset.