Modern machine learning far outperforms GLMs at predicting spikes

Ari S. Benjamin[1], Hugo L. Fernandes[2], Tucker Tomlinson[3], Pavan Ramkumar[2,4], Chris VerSteeg[1], Lee Miller[1,2,3], Konrad Paul Kording[1,2,3]

1. *Department of Biomedical Engineering, Northwestern University,, Evanston, IL, 60208, USA*
2. *Department of Physical Medicine and Rehabilitation, Northwestern University and Rehabilitation Institute of Chicago, Chicago, IL, 60611, USA*
3. *Department of Physiology, Northwestern University, Chicago, IL, 60611, USA*
4. *Department of Neurobiology, Northwestern University, Evanston, IL, 60208, USA*

Contact: Ari Benjamin, aribenjamin2014@u.northwestern.edu

Abstract

Neuroscience has long focused on finding encoding models that effectively ask "what predicts neural spiking?" and generalized linear models (GLMs) are a typical approach. Modern machine learning techniques have the potential to perform better. Here we directly compared GLMs to three leading methods: feedforward neural networks, gradient boosted trees, and stacked ensembles that combine the predictions of several methods. We predicted spike counts in macaque motor (M1) and somatosensory (S1) cortices from reaching kinematics, and in rat hippocampal cells from open field location and orientation. In general, the modern methods produced far better spike predictions and were less sensitive to the preprocessing of features. XGBoost and the ensemble were the best-performing methods and worked well even on neural data with very low spike rates. This overall performance suggests that tuning curves built with GLMs are at times inaccurate and can be easily improved upon. Our publicly shared code uses standard packages and can be quickly applied to other datasets. Encoding models built with machine learning techniques more accurately predict spikes and can offer meaningful benchmarks for simpler models.

Introduction

A central tool of neuroscience is the tuning curve, which maps stimulus to neural response. The tuning curve asks what information in the external world a neuron encodes in its spikes. For a tuning curve to be meaningful it is important that it accurately predicts the neural response. Often, however, methods are chosen that sacrifice accuracy for simplicity. Predictive methods for tuning curves should instead be evaluated primarily by their ability to describe neural activity accurately.

A common predictive model is the Generalized Linear Model (GLM), occasionally referred to as a linear-nonlinear Poisson (LNP) cascade (*1-4*). The GLM performs a nonlinear operation upon a linear combination of the input features, which are often called external covariates. Typical covariates are stimulus features, movement vectors, or the animal's location. The nonlinear operation on the weighted sum of covariates is usually held fixed, though it can be learned (*5, 6*), and the linear weights of the combined inputs are chosen to maximize the agreement between the model fit and the neural recordings. This optimization problem of choosing weights is often convex and can be solved with efficient algorithms (*7*). The assumption of Poisson firing statistics can often be loosened (*8*) allowing the modeling of a broad range of neural responses. Due to its ease of use, perceived interpretability, and flexibility, the GLM has become a popular model of neural spiking.

The GLM's central assumption of linearity in feature space may hold in certain cases (*8, 9*), but in general, neural responses can be very nonlinear (*5, 10*). When a neuron responds nonlinearly to stimulus features, it is common practice to mathematically transform the features to obtain a new set that meets the linearity requirements of the GLM and yields better spike

predictions. The new features may be any function of the original features and may include cross-interactions. In keeping with the machine learning literature, we call this step *feature engineering*. The precise form of feature engineering is rarely rigorously determined and often falls to the researcher's intuition. Given the infinite space of possible engineered features, it is unlikely that any guess would yield a set that is truly linear with respect to inputs. Incorrect guesses yield suboptimal predictions, and it is therefore important that GLMs be compared with nonlinear models that can express more complex stimulus–response relationships.

Machine learning (ML) methods for regression have improved dramatically since the invention of the GLM. Many ML methods require little feature engineering and do not need to assume linearity. Top performing methods, (as judged by the frequency of winning solutions on Kaggle, a ML competition website (*11*)) include neural networks (*12*), gradient boosted trees (*13*), and ensemble techniques. Many neuroscientists are unaware that these methods are now relatively easy to implement in a few lines of code in a scripting language such as Python. This ease of use is enabled by machine learning packages that are supported by large groups of scientists, such as scikit-learn (*14*), Keras (*15*), Theano (*16*), and XGBoost (*13*). Applications of modern ML to spike prediction remain rare, though some inroads have been made with neural networks (*17-20*). The greatly increased predictive power of modern ML methods is now very accessible and could improve the state of the art in encoding models across neuroscience.

Here we applied the standard ML methods of artificial neural networks, gradient boosted trees, and ensemble methods to the task of spike prediction, and evaluated their performance alongside a GLM. We compared the methods on recordings from three separate brain areas. These areas differed greatly in the effect size of covariates and typical spike rates, and thus served to evaluate the strengths of these methods across different conditions. For neurons from each area we found that the advanced ML methods could more accurately predict spiking than the GLM. The stacked ensemble and XGBoost were consistently the highest-scoring of the methods tested. We provide our implementing code in an accessible format so that all neuroscientists may easily test and compare these methods on other datasets.

Methods

Data

We tested our methods at predicting spikes for neurons in the macaque primary motor cortex, the macaque primary somatosensory cortex, and the rat hippocampus.

The macaque motor cortex data consisted of previously published electrophysiological recordings from 82 neurons in the primary motor cortex (M1) (*21*). The neurons were sorted from recordings made during a two-dimensional center-out reaching task with eight targets. In this task the monkey grasped the handle of a planar manipulandum that controlled a cursor on a computer screen and simultaneously measured the hand location and velocity (Fig. 1). After training, an electrode array was implanted in the arm area of area 4 on the precentral gyrus. Spikes were discriminated using offline sorter (Plexon, Inc), counted and collected in 50-ms bins. The neural recordings used here were taken in a single session lasting around 13 minutes.

The macaque primary somatosensory cortex (S1) data was recorded during a two-dimensional random-pursuit reaching task and was previously unpublished. In this task, the monkey gripped the handle of the same manipulandum. The monkey was rewarded for bringing the cursor to a series of randomly positioned targets appearing on the screen. After training, an electrode array was implanted in the arm area of area 2 on the postcentral gyrus, which receives a mix of cutaneous and proprioceptive afferents. Spikes were processed as for M1. The data used for this publication derives from a single recording session lasting 51 minutes.

As for M1 (described in results), we processed the hand position, velocity, and acceleration accompanying the S1 recordings in an attempt to obtain linearized features. We extracted six features for the models: the hand speed, the sine and cosine of velocity direction, the distance of the hand from the center of the workspace, and the sine and cosine of the angle of the hand position with respect the workspace center. Cells in the arm area of S1 have been shown to have approximately sinusoidal tuning curves relating to movement direction (*22*), and the features were chosen accordingly. The features $(x, y, \dot{x}, \dot{y}, \ddot{x}, \ddot{y})$ were also tested but were not found to improve the performance of the GLM.

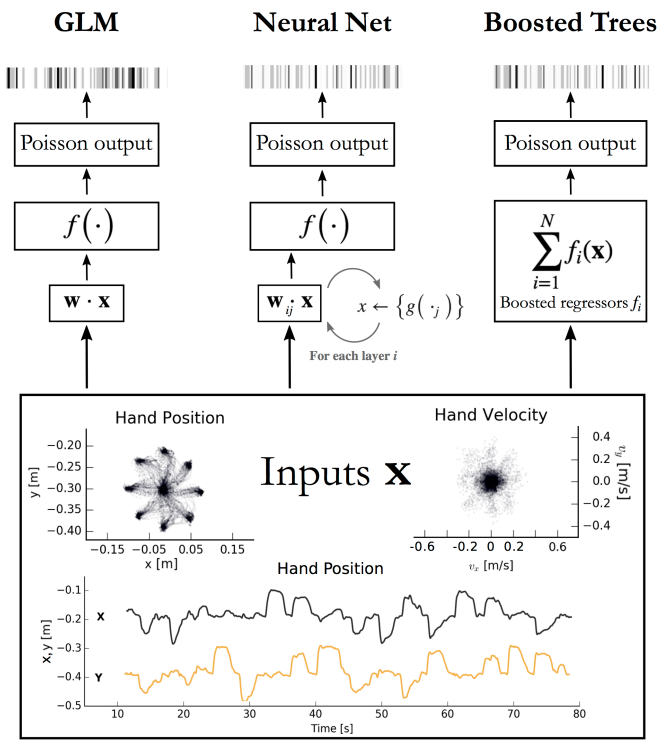

Figure 1: Encoding models aim to predict spikes, top, from input data, bottom. The inputs displayed are the position and velocity signals from the M1 dataset (*21*) but could represent any set of external covariates. The GLM takes a linear combination of the inputs, applies an exponential function *f*, and produces a Poisson spike probability that can be used to generate spikes (left). The feedforward neural network (center) does the same when the number of hidden layers $i = 0$. With $i \geq 1$ hidden layers, the process repeats; each of the *j* nodes in layer *i* computes a nonlinear function *g* of a linear combination of the previous layer. The vector of outputs from all *j* nodes is then fed as input to the nodes in the next layer, or to the final exponential *f* on the final iteration. Boosted trees (right) return the sum of N functions of the original inputs. Each of the f_i is built to minimize the residual error of the sum of the previous $f_{0:i-1}$.

The third dataset consists of recordings from 58 neurons in the CA1 region of the rat dorsal hippocampus during a single 93 minute free foraging experiment, previously published and made available online (*23, 24*). Position data from two head-mounted LEDs provided position and heading direction inputs. Once again we binned inputs and spikes in 50ms bins. Since many neurons in the dorsal hippocampus are responsive to the location of the rat, we processed the 2D position data into a list of squared distances from a 5x5 grid of place fields that tile the workspace. Each position feature thus has the form

$$p_{ij} = \frac{1}{2}\left(x(t) - \mu_{ij}\right)^T \Sigma_{ij}^{-1}\left(x(t) - \mu_{ij}\right),$$

where $\mu_{i,j}$ is the center of place field *i*, *j*≤5 and Σ_{ij} is a covariance matrix chosen for the uniformity of tiling. An exponentiated linear combination of the p_{ij} (as is performed in the GLM) evaluates to a single Gaussian centered anywhere between the place fields. The inclusion of the p_{ij} as features thus transforms the standard representation of cell-specific place fields (*25*) into the mathematical formulation of a GLM. The final set of features included the p_{ij} as well as the rat speed and head orientation.

Generalized Linear Model

The Poisson generalized linear model is a multivariate regression model that describes the instantaneous firing rate as a nonlinear function of a linear combination of input features (see e.g. (*26, 27*) for review, (*28, 29*) for usage). Here, we took the form of the nonlinearity *f* to be exponential, as has been found to be successful in previous applications of GLMs to similar data (*30*). After the nonlinearity, spiking is generated as a Poisson process, in which the probability of firing in any instant is independent of firing history. The general form of the GLM is depicted Figure 1. We implemented the GLM using elastic-net regularization, using the open-source Python package pyglmnet (*31*). The regularization path was optimized separately on a single neuron in each dataset on a validation set not used for scoring.

Neural Network

Neural networks are well-known for their success at supervised learning tasks. More comprehensive reviews can be found elsewhere (*12*). Here, we implemented a simple feedforward neural network. A model that takes predicted spike history as input, such as a recurrent neural network, would likely increase predictive power. We omit such architectures to be able to establish the methods' relative power when trained on the same information.

We point out that a neural network with no hidden layers is equivalent in mathematical form to a GLM (Fig. 1). For multilayer networks, one can write each hidden layer of *n* nodes as simply *n* GLMs, each taking the output of the previous layer as inputs (noting that the weights of each are chosen to maximize only the final objective function, and that the intermediate nonlinearities need not be the same as the output

nonlinearity). A feedforward neural network is thus a generalization, or repeated application of a GLM.

The networks were implemented with the open-source neural network library Keras, running Theano as the backend (*15, 16*). The network contained two hidden layers (or one for S1), dense connections, rectified linear activation, and a final exponentiation. To help avoid overfitting, we allowed dropout on the first layer, and an elastic-net or max-norm regularization upon the weights (but not the bias term) of the network (*32*). The networks were trained to maximize the Poisson likelihood of the neural response. We optimized over four hyperparameters: the number of nodes in the first and second hidden layers (if present), the dropout, and the regularization parameters. Optimization was performed on only a subset of the data from a single neuron in each dataset, using Bayesian optimization (*33*) in an open-source Python implementation (*34*).

Gradient Boosted Trees

A popular method in many machine learning competitions is that of gradient boosted trees. Here we describe the general operation of XGBoost, an open-source implementation that is efficient and highly scalable, works on sparse data, and easy to implement out-of-the-box (*13*).

XGBoost trains many sequential models to minimize the residual error of the sum of previous model. Each model is a decision tree, or more specifically a classification and regression tree (CART) (*35*). Training a decision tree amounts to determining a series of rule-based splits on the input to classify output. The CART algorithm generalizes this to regression by taking continuously-valued weights on each of the leaves of the decision tree.

For any predictive model $\hat{y}^{(1)} = f_1(\boldsymbol{x_i})$ and true response y_i, we can define a loss function $l\left(\hat{y}^{(1)}, y_i\right)$ between the prediction and the response. The objective to be minimized during training is then simply the sum of the loss over each training example *i*, plus some regularizing function Ω that biases towards simple models.

$$L = \sum_i l(\hat{y}_i^{(1)}, y_i) + \Omega(f_1)$$

After minimizing L for a single tree, XGBoost constructs a second tree $f_2(\boldsymbol{x_i})$ that approximates the residual. The objective to be minimized is thus the total loss L between the true response y_i and the sum of the predictions given by the first tree and the one to be trained.

$$L = \sum_i l(\hat{y}_i^{(1)} + f_2(\boldsymbol{x}_i), y_i) + \Omega(f_2)$$

This process is continued sequentially for a predetermined number of trees, each trained to approximate the residual of the sum of previous trees. In this manner XGBoost is designed to progressively decrease the total loss with each additional tree. At the end of training, new predictions are given by the sum of the outputs of all trees.

$$\hat{y} = \sum_{k=1}^{N} f_k(\boldsymbol{x})$$

In practice, it is simpler to choose the functions f_k via gradient boosting, which minimizes a second order approximation of the loss function (*36*).

XGBoost offers several additional parameters to optimize performance and prevent overfitting. Many of these describe the training criteria for each tree. We optimized some of these parameters for a single neuron in each dataset using Bayesian optimization (again over a validation set different from the final test set). These parameters included the number of trees to train, the maximum depth of each decision tree, and the minimum weight allowed on each decision leaf, the data subsampling ratio, and the minimum gain required to create a new decision branch.

Random Forests

Random forests train multiple parallel decision trees on the features-to-spikes regression problem (not sequentially on the remaining residual, as in XGBoost) and averages their outputs (*37*). The variance on each decision tree is increased by training on a sample of the data drawn with replacement (i.e., bootstrapped inputs) and by choosing new splits using only a random subset of the available features. Random forests are implemented in Scikit-learn (*14*). We introduced this method only to increase the power of the ensemble (see below). Their performance alone is displayed in Supplementary Figure 1. It should be noted that the Scikit-learn implementation currently only minimizes the mean-squared error of the output, which is not properly applicable to Poisson processes and may cause poor performance. Despite this drawback their presence still improves the ensemble scores.

Ensemble Method

It is common machine learning practice to create ensembles of several trained models. Different algorithms may learn different characteristics of the data, make different types of errors, or generalize differently to new examples. Ensemble methods allow for the successes of different algorithms to be combined. Here we implemented *stacking*, in which the output of several models is taken as the input set of a new model (*38*). After training the GLM, neural network, random forest, and XGBoost on the features of each dataset, we trained an additional instance of XGBoost using the spike predictions of the previous methods as input. The outputs of this 'second stage' XGBoost are the predictions of the ensemble.

Scoring and Cross-Validation

Each of the three methods was scored with the pseudo-R^2 score, a scoring function applicable to Poisson processes (*39*). Note that a standard R^2 score assumes Gaussian noise and cannot be applied here. The pseudo-R^2 was calculated as

$$R_M^2 = 1 - \frac{\log L(y) - \log L(\hat{y})}{\log L(y) - \log L(\bar{y})} = \frac{\log L(\hat{y}) - \log L(\bar{y})}{\log L(y) - \log L(\bar{y})}$$

Here $L(y)$ is the log likelihood of the true output, $L(\hat{y})$ is the log likelihood of the predicted output, and $L(\bar{y})$ is the null log likelihood, which here is the log likelihood of the data under the mean firing rate alone. The pseudo-R^2 can be interpreted as the fraction of the maximum potential log-likelihood gain (relative to the null model) achieved by the tested model (*39*). The score can also be seen as related to the ratio of deviances of the tested model and the null model. It takes a value of 0 when the data is as likely under the tested model as the null model, and a value of 1 when the tested model perfectly describes the data. It is empirically a lower value than a standard R^2 when both are applicable (*40*). The null model can also be taken to be a model other than the mean firing rate (e.g. the GLM) to directly compare two methods, in which case we refer to the score as the 'comparative pseudo-R^2'. The comparative pseudo-R^2 is referred to elsewhere as the 'relative pseudo-R^2', renamed here to avoid confusion with the difference of two standard pseudo-R^2 scores measured against the mean (*29*).

As many methods are prone to overfitting the training data, we used 8-fold cross-validation (CV) when assigning a final score to the models. Briefly, the input and spike data were randomly segmented, discontinuously in time, into eight equal partitions. The methods were trained on seven partitions and tested on the eighth, and this was repeated until all segments served as the test partition once. The mean and variance of the eight scores are then recorded for the final score.

Cross-validation for ensemble methods requires extra care to ensure that there is no leak of information from the validation set into the training set. The training set for the ensemble must contain predictions from methods that were themselves not trained on the validation set. This rules out using simple *k*-fold CV with all methods trained on the same folds. Instead, we used the following nested CV scheme to train and score the ensemble. The data were split into $p=8$ folds, each of which contained a training set and a test set for the ensemble. On each fold standard *k*-fold CV is run on just the training set with each first stage method (GLM, etc.) such that we obtain predictions for all training data. The ensemble's test set is then obtained from the predictions of the first stage methods trained on the entire training set. This ensures that the ensemble's test set was never used for training any method. The process is repeated for each of the *p* folds and the mean and variance of the *p* scores of the ensemble's predictions are recorded.

Results

We applied modern machine learning methods to predict spike counts in three brain regions and compared the quality of the predictions to those of a GLM. Our primary analysis centered on neural recordings from the macaque primary motor cortex (M1) during reaching (Fig. 1). Analyses of data from macaque S1 and from rat hippocampus indicate how these methods compare beyond M1. On each of the three datasets we trained a GLM and compared it to the performance of a feedforward neural network, XGBoost (a gradient boosted trees implementation), and an ensemble method. The ensemble was inspired by ML competition strategies and was an additional instance of XGBoost trained on the predictions of all three methods plus a random forest regressor. Together, these methods allowed us to compare the performance of traditional GLMs with modern methods. The resulting code implementing these methods can be used by any electrophysiology lab to compare these machine

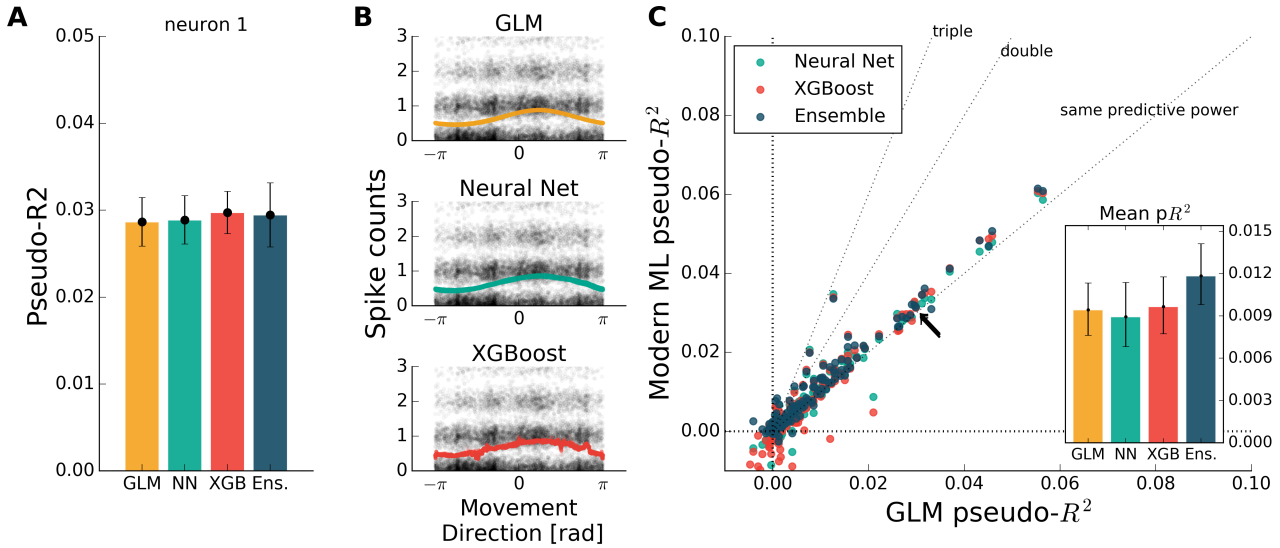

Figure 2: Encoding models for M1 performed similarly when trained on the sine and cosine of hand velocity direction. (a) The pseudo-R^2 for an example neuron was similar for all four methods. On this figure and in Figures 3-5 the example neuron is the same, and is not the neuron for which method hyperparameters were optimized. (b) The tuning curves of the neural net and XGBoost were similar to that of the GLM. The black points are the recorded responses, to which we added y-axis jitter for visualization. The tuning curve of the ensemble method was similar and is omitted here for clarity. (c) Plotting the pseudo-R^2 of modern ML methods vs. that of the GLM indicates that the similarity of methods generalizes across neurons. The single neuron plotted at left is marked with black arrows. The mean scores, inset, indicate the overall success of the methods; error bars represent the 95% bootstrap confidence interval.

learning methods with their own approaches for encoding models.

To test that all methods work reasonably well in a trivial case, we trained each to predict spiking from a simple and well-understood feature. Some neurons in M1 have been described as responding linearly to the exponentiated cosine of movement direction relative to a preferred angle (*41*). We therefore predicted the

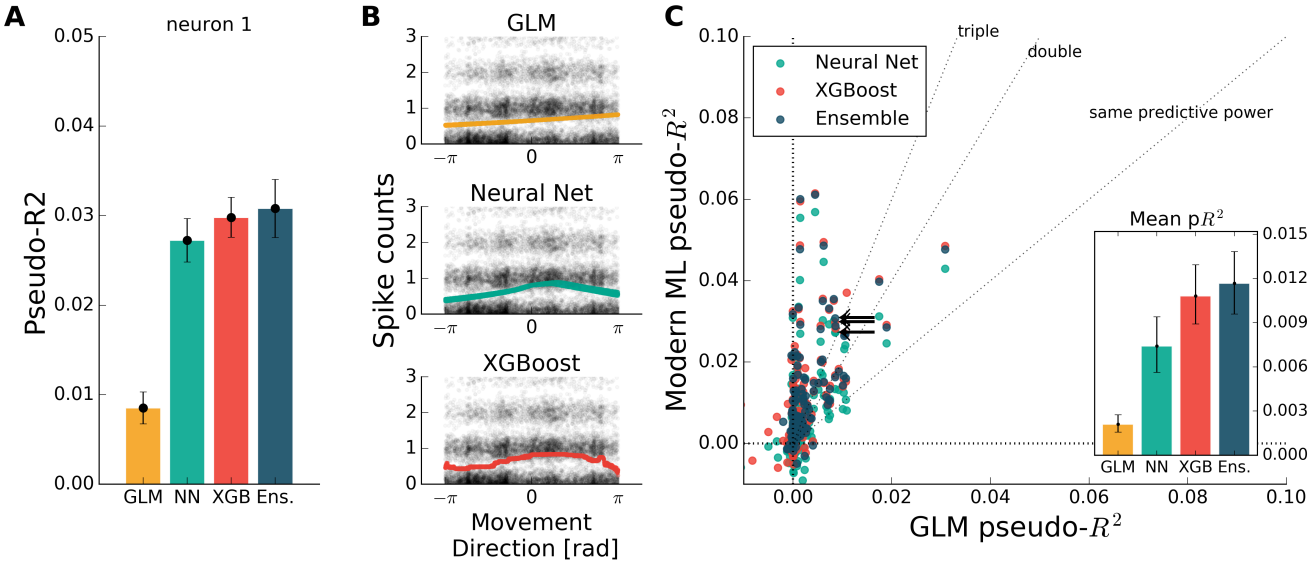

Figure 3: Modern ML models could learn the cosine nonlinearity when trained on only the direction of hand velocity, in radians. (a) For the same example neuron as in Figure 3, the neural net and XGBoost maintained the same predictive power, while the GLM was unable to extract a relationship between direction and spike rate. (b) XGBoost and neural nets displayed reasonable tuning curves, while the GLM reduced to the average spiking rate (with a small slope, in this case). (c) Most neurons in the population were poorly fit by the GLM, while the ML methods achieved the performance levels of Figure 2. The ensemble performed the best of the methods tested.

spiking of M1 neurons from the cosine and sine of the direction of hand movement in the reaching task. (The linear combination of a sine and cosine curve is a phase-shifted cosine, by identity, allowing the GLM to learn the proper preferred direction). We observed that each method identified a similar tuning curve (Fig. 2b), constructed by plotting the predictions of spike rate on the validation set against movement direction. The bulk of the neurons in the dataset were just as well predicted by each of the methods (Fig. 2a, c), though the ensemble was slightly better than the GLM (mean comparative pseudo-R^2, defined in methods, of 0.06 [0.043 – 0.084], 95% bootstrapped confidence interval (CI)). The similar performance suggested that an exponentiated cosine is a nearly optimal approximating function of the neural response to movement direction alone, as was previously known (*42*). This classic example thus illustrated that all methods can in principle estimate tuning curves.

The exact form of the nonlinearity of the neural response to a given feature is rarely known, but this lack of knowledge need not impact our prediction ability. To illustrate the ability of modern machine learning to find the proper nonlinearity, we performed the same analysis as above but omitted the initial cosine feature engineering step. Trained on only the hand velocity direction, in radians, which are likely to be discontinuous at ±π, the modern ML methods very nearly reproduced the predictive power they attained using the engineered feature (Fig. 3a). As expected, the GLM failed at generating a meaningful tuning curve (Fig. 3b). Both trends were consistent across the population of recorded neurons (Fig. 3c). The neural net, XGBoost, and ensemble methods thus perform well without feature engineering and the required prior knowledge or assumptions.

Machine learning methods can also take advantage of information contained in combinations of inputs, and should perform better if given more inputs. We verified that this was true for our dataset by training on the four-dimensional set of hand position and velocity $(x, y, \dot{x}, \dot{y})$, which we call the set of original features. All methods gained a significant amount of predictive power with these new features, though the GLM did not nearly match the other methods (Fig 4a, c). This set of neurons thus seemed to encode strongly for position and velocity in a potentially nonlinear fashion captured by machine learning methods.

While some amount of feature engineering can improve the performance of GLMs, it is not always simple to guess the optimal set of processed features. We demonstrated this by training all methods on features that have previously been successful at explaining spike

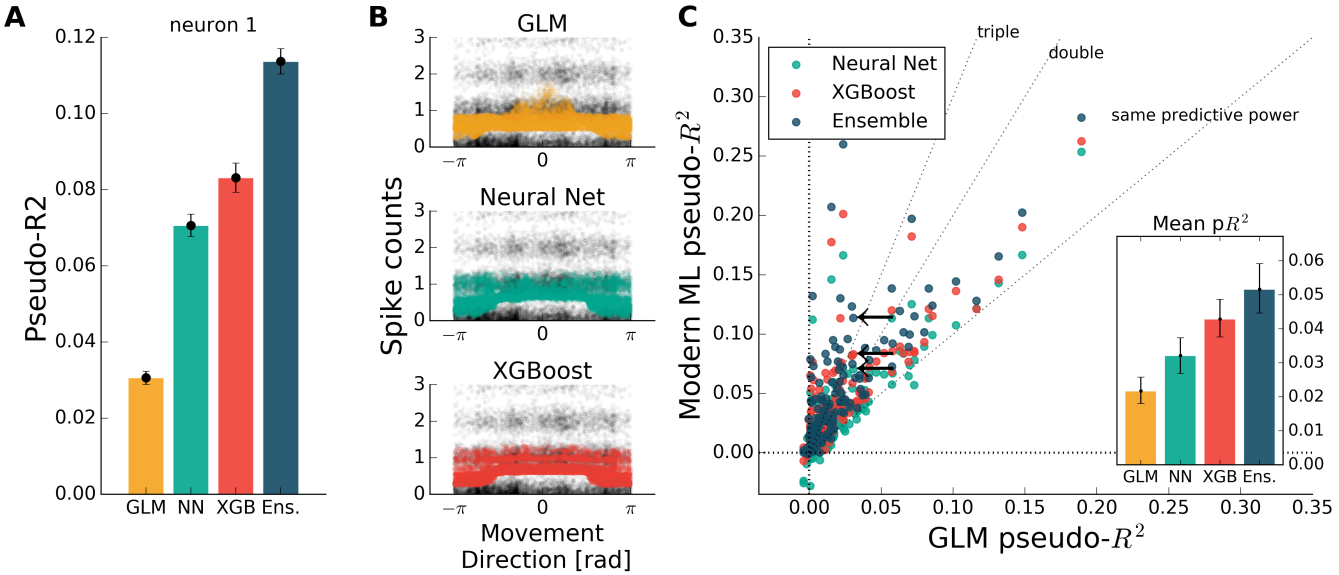

Figure 4: Training on the set of original features $(x, y, \dot{x}, \dot{y})$ increased the predictive power of all methods. Note the change in axes scales from Figures 2-3. (a) For the same example neuron as in Figure 3, all methods gained a significant amount of predictive power, indicating a strong encoding of position and speed or their correlates. The GLM showed less predictive power than the other methods on this feature set. (b) The spike rate in black, with jitter on the y-axis, again overlaid with the predictions of the three methods as a function of velocity direction. The neuron encodes for position and speed, as well, and the projection of the multidimensional tuning curve onto a 1D velocity direction dependence leaves the projected curve diffuse. (c) The ensemble method, neural network, and XGBoost performed consistently better than the GLM across the population. The mean pseudo-R^2 scores show the hierarchy of success across methods.

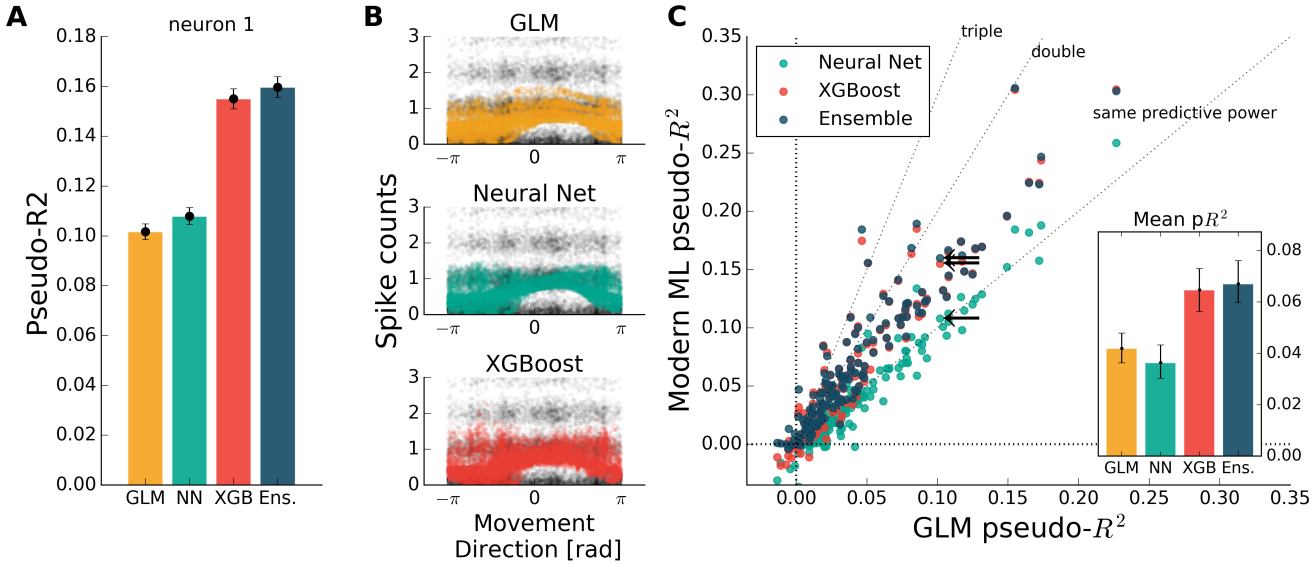

Figure 5: Encoding models for M1 trained on all the original features plus the engineered features show that modern ML methods can outperform the GLM even with standard featuring engineering. (a) For this example neuron, inclusion of the computed features increased the predictive power of the GLM to the level of the neural net. XGBoost and the ensemble method also increased in predictive power. (b) The tuning curves for the example neuron are diffuse when projected onto the movement direction, indicating a high-dimensional dependence. (c) Even with feature engineering, XGBoost and the ensemble consistently achieve pseudo-R^2 scores higher than the GLM, though the neural net does not. The selected neuron at left is marked with black arrows.

rate in a similar center-out reaching task (*6*). These extra features included the sine and cosine of velocity direction (as in Figure 2), the speed, the radial distance of hand position, and the sine and cosine of position direction. The training set was thus 10-dimensional, though highly redundant, and was aimed at maximizing the predictive power of the GLM. Feature engineering improved the predictive power of all methods to variable degrees, with the GLM improving to the level of the neural network (Fig. 5). XGBoost and the ensemble still predicted spikes better than the GLM (Fig. 5c), with the ensemble scoring on average 1.8 times higher than the GLM (ratio of population means of 1.8 [1.4 – 2.2], 95% bootstrapped CI). The ensemble was significantly better than XGBoost (mean comparative pseudo-R^2 of 0.08 [0.055 – 0.103], 95% bootstrapped CI) and was thus consistently the best predictor. Though standard feature engineering greatly improved the GLM, the ensemble and XGBoost still captured the neural response more accurately.

To ensure that these results are not specific to the motor cortex, we extended the same analyses to primary somatosensory cortex (S1) data. The ensemble was consistently the best predictor across all neurons, scoring almost twice as well as the GLM (ratio of 1.8 [1.2 – 2.2] of population means, 95% bootstrapped CI). XGBoost predicted spikes better than the GLM only for neurons with significant effect sizes for any of the four methods (i.e., with cross-validated pseudo-R^2 scores two standard deviations above 0; mean comparative pseudo-R^2 was 0.002 [0.0006 – 0.0045], 95% bootstrapped CI). Interestingly, the neural network performed worse than all other methods. We speculated that this could be related to the small covariate effect size in the S1 dataset, as we observed similar scores for the neural network on the M1 dataset for regimes of similar effect sizes, as well as on simulated data with GLM structure, small effect size, and similar firing rates (Supp. Fig. 2). We also found that a much smaller network performed better (a single hidden layer with 20 nodes) but that max-norm or elastic-net regularization did not improve the results with the larger network. Neural networks may thus be poor choices for Poisson data with very small covariate effect sizes, though we see no theoretical reason why this should be the case. Overall, on this S1 dataset featuring generally low predictability, the tested methods displayed a range of performances, with the ensemble predicting the data nearly twice as well as the GLM alone.

We asked if the same trends of performance would hold for the rat hippocampus dataset, which was characterized by very low mean firing rates but strong effect sizes. All methods were trained on a list of features representing the rat position and orientation, as

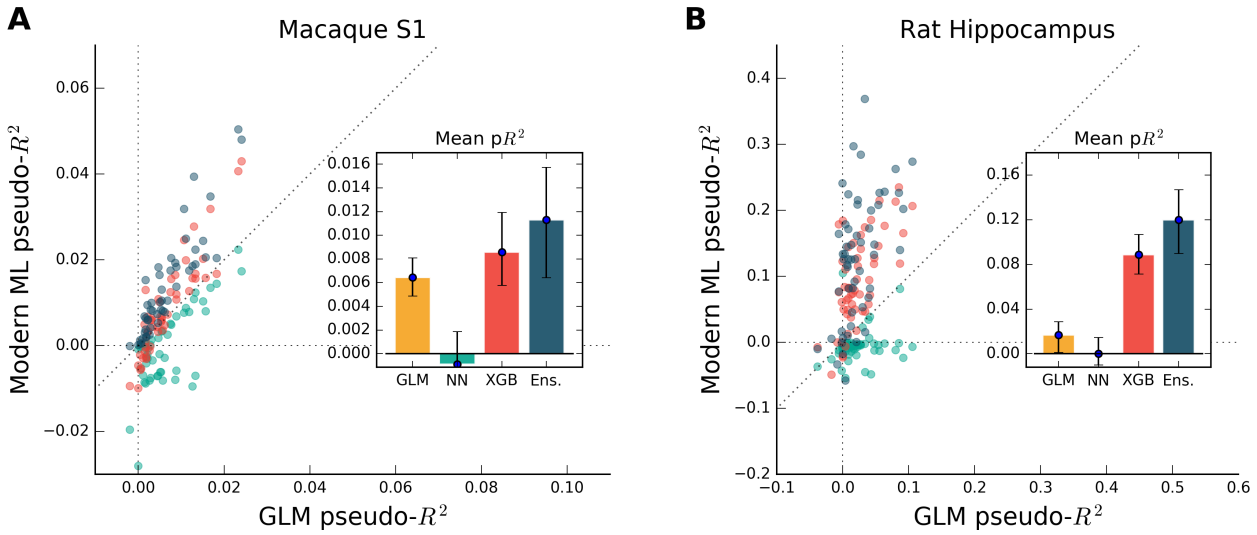

Figure 6: XGBoost and the ensemble method predicted the activity of neurons in S1 and the hippocampus better than a GLM. The diagonal dotted line in both plots is the line of equal predictive power with the GLM. (a) The ensemble predicted firing almost twice as well, on average, as the GLM for all neurons in the S1 dataset. XGBoost was better for neurons with higher effect sizes but poorly predicted neurons that were not predictable by any method. The neural network performed the worst of all methods. (b) Many neurons in the rat hippocampus were described well by XGBoost and the ensemble but poorly by the GLM and the neural network. The poor neural network performance in the hippocampus was due to the low rate of firing of most neurons in the dataset (Supp. Fig. 2). Note the difference in axes; hippocampal cells are generally more predictable than those in S1.

described in methods. We found that many neurons were described much better by XGBoost and the ensemble method than by the GLM (Fig. 6b). On average, the ensemble was almost ten times more predictive than the GLM (ratio of population means of 9.8 [5.4 – 100.0], 95% bootstrapped CI), and many neurons shifted from being completely unpredictable by the GLM (pseudo-R^2 near zero) to very predictable by XGBoost and the ensemble (pseudo-R^2 above 0.2). The neural network performed poorly, here not due to effect size as in S1 but likely due to the very low firing rates of most hippocampal cells (Supp. Fig. 2). Out of the 58 neurons in the dataset, 54 had rates below 1 spikes/second, and it was only on the four high-firing neurons that the neural network achieved pseudo-R^2 scores comparable to the GLM. The relative success of XGBoost was interesting given the failure of the neural network, and supported the general observation that XGBoost can work well with smaller and sparser datasets than those neural networks generally require. Thus for hippocampal cells, a method leveraging decision trees such as XGBoost or the ensemble is able to capture far more structure in the neural response than the GLM or the neural network.

Discussion

We contrasted the performance of GLMs with recent machine learning techniques at the task of predicting spike rates in three brain regions. We found that the tested ML methods predicted spike rates far more accurately than the GLM. Typical feature engineering only partially bridged the performance gap. The ML methods performed comparably well with and without feature engineering, indicating they could serve as convenient performance benchmarks for improving simpler encoding models. The consistently best method was the ensemble, which was an instance of XGBoost stacked on the predictions of the GLM, neural network, XGBoost, and a random forest. The ensemble and XGBoost could fit the data well even for very low spike rates, as in the hippocampus dataset, and for very low covariate effect sizes, as in the S1 dataset. These findings indicate that GLMs are not the best choice as neuroscience's standard method of spike prediction.

The ML methods we have put forward here have been implemented without substantial modification from methods that are already in wide use. We hope that this simple application might spur a wider adoption of these methods in the neurosciences, thereby increasing the power and efficiency of studies involving neural prediction without requiring complicated, application-

specific methods development (e.g. (*43*)). Our methods could also be further optimized by including additional information, such as spike history, covariate history, or the phase relative to the theta cycle (*25, 44*) in the hippocampus. While such steps could be valuable in future studies, they are not necessary for this demonstration of the methods' relative power when operating on a common set of inputs. Further improvements are possible, but researchers may still gain descriptive power over GLMs with simple, out-of-the-box implementations.

The success of a GLM depends on the form of the input features, and as such it might be argued that the GLM underperforms simply because we have selected the wrong sets. This is true, in a technical sense; in principle, one can always find a set of operations that maps the features to a linear regime. (The output of XGBoost, say, or its first several approximating moments.) It is worth asking, however, not just whether different features could improve a GLM but also whether it is necessary or useful to use a GLM in the first place. When determining if a neuron encodes a certain set of features, like muscle forces or body position, one can choose to ask if there is a neural response that is linear with respect to those features, or alternatively if there is any learnable response at all. We posit that the brain is not *a priori* a linear computation engine, and that framing computations in linear space does not necessarily better represent the computations that a neuron 'actually' performs. Choosing to observe the widest possible space of responses may thus be the more prudent decision. Furthermore, we gain little understanding of the neural function by lifting the GLM to the level of the ensemble with feature engineering if there is no prior preference for linearity. It is far easier to stay agnostic to the form of engineered features and use modern ML methods to find an optimal predicting function.

Advanced ML methods are not widely considered to be interpretable, and some may worry that this diminishes their scientific value as encoding models. We can better discuss this issue with a more precise definition of interpretability. Lipton makes the distinction between a method's *post-hoc interpretability*, the ease of justifying its predictions, and *transparency,* the degree to which its operation and internal parameters are human-readable or easily understandable (*45*). A GLM is certainly more transparent than many ML methods due to its algorithmic simplicity. Post-hoc explanations of predictions, on the other hand, are often possible with modern ML methods. It is possible, for example, to visualize the aspects of stimuli that most elicit a predicted response, as has been implemented in previous applications of neural networks to spike prediction (*17, 18*). Post-hoc explanations also include descriptive explanations and justifications by example ("neuron y fired when the stimulus sounded like a human voice"). Work is underway to add such post-hoc explanations to the capabilities of neural networks (*46, 47*). These capabilities for post-hoc justifications could be as scientifically valuable as method transparency if successfully implemented.

Not all types of interpretability are necessary for a given task, and many scientific questions can be answered based on predictive ability alone. Questions of the form, "does feature x contribute to neural activity?", for example, require no method transparency; one can simply ask whether predictive power increases with feature x's inclusion. Advanced ML methods could thus be readily applied to studies of feature importance across the brain (e.g. (*48-50*)). Lack of transparency should thus not generally preclude the use of advanced ML methods in neuroscience.

Though GLMs are considered transparent, it is important to note that a few issues complicate the interpretation of their parameters. Regularization imposes prior distributions on feature weights, introducing a bias that is often left unconsidered. Unaccounted nonlinearity may also cause issues with interpretation. In the extreme case when the neural response to some feature x does not correlate with $\exp(x)$, the feature weights may incorrectly predict no dependence on feature x whatsoever. Feature engineering attempts to resolve this issue, though the engineered features must be guessed if the nonlinearity is unknown. This will leave some ambiguity as to how much the new feature weights simply capture the scaling of the engineering function as opposed to the relative contribution of the feature. Finally, any feature covariance must be acknowledged when examining fitted weights. One may find, for example, that a neuron fires in response to both x and x^3 when the most linearly related feature is $\sin(x)$, which is better approximated as a combination of both terms than by either alone. It would thus be a mistake to interpret the feature weights on x and x^3 as their 'contribution' to firing. These several considerations serve to reduce the interpretability of the parameters of GLMs, and should be remembered when choosing a model for studies of feature importance.

The brain is nonlinear and complex. A systematic description with linear models presents the danger of obscuring its function with an illusion of a simpler form.

The development of a conceptual framework that acknowledges neural complexity could be greatly aided by building tuning curves that capture arbitrary nonlinearity and more accurately describe neural activity.

The code used for this publication is available at https://github.com/KordingLab/spykesML. We invite researchers to adapt it freely for future problems of neural prediction.

References

1. E. P. Simoncelli, L. Paninski, J. Pillow, O. Schwartz, Characterization of neural responses with stochastic stimuli. *The cognitive neurosciences* **3**, 327-338 (2004).
2. M. C.-K. Wu, S. V. David, J. L. Gallant, Complete functional characterization of sensory neurons by system identification. *Annu. Rev. Neurosci.* **29**, 477-505 (2006).
3. S. Gerwinn, J. H. Macke, M. Bethge, Bayesian inference for generalized linear models for spiking neurons. *Frontiers in Computational Neuroscience* **4**, (2010).
4. J. A. Nelder, R. J. Baker, Generalized linear models. *Encyclopedia of statistical sciences*, (1972).
5. E. Chichilnisky, A simple white noise analysis of neuronal light responses. *Network: Computation in Neural Systems* **12**, 199-213 (2001).
6. L. Paninski, M. R. Fellows, N. G. Hatsopoulos, J. P. Donoghue, Spatiotemporal tuning of motor cortical neurons for hand position and velocity. *Journal of neurophysiology* **91**, 515-532 (2004).
7. L. Paninski, Maximum likelihood estimation of cascade point-process neural encoding models. *Network: Computation in Neural Systems* **15**, 243-262 (2004).
8. J. W. Pillow, L. Paninski, V. J. Uzzell, E. P. Simoncelli, E. Chichilnisky, Prediction and decoding of retinal ganglion cell responses with a probabilistic spiking model. *The Journal of neuroscience* **25**, 11003-11013 (2005).
9. A. P. Georgopoulos, A. B. Schwartz, R. E. Kettner, Neuronal population coding of movement direction. *Science* **233**, 1416-1419 (1986).
10. R. D. R. Van Steveninck, W. Bialek, Real-time performance of a movement-sensitive neuron in the blowfly visual system: coding and information transfer in short spike sequences. *Proceedings of the Royal Society of London B: Biological Sciences* **234**, 379-414 (1988).
11. Kaggle Winner's Blog. (2016).
12. J. Schmidhuber, Deep learning in neural networks: An overview. *Neural Networks* **61**, 85-117 (2015).
13. T. Chen, C. Guestrin, Xgboost: A scalable tree boosting system. *arXiv preprint arXiv:1603.02754*, (2016).
14. F. Pedregosa *et al.*, Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research* **12**, 2825-2830 (2011).
15. F. Chollet, keras. *GitHub repository*, (2015).
16. T. T. D. Team *et al.*, Theano: A Python framework for fast computation of mathematical expressions. *arXiv preprint arXiv:1605.02688*, (2016).
17. B. Lau, G. B. Stanley, Y. Dan, Computational subunits of visual cortical neurons revealed by artificial neural networks. *Proceedings of the National Academy of Sciences* **99**, 8974-8979 (2002).
18. R. Prenger, M. C.-K. Wu, S. V. David, J. L. Gallant, Nonlinear V1 responses to natural scenes revealed by neural network analysis. *Neural Networks* **17**, 663-679 (2004).
19. S. R. Lehky, T. J. Sejnowski, R. Desimone, Predicting responses of nonlinear neurons in monkey striate cortex to complex patterns. *The Journal of neuroscience* **12**, 3568-3581 (1992).
20. L. McIntosh, N. Maheswaranathan, A. Nayebi, S. Ganguli, S. Baccus, in *Advances in Neural Information Processing Systems*. (2016), pp. 1361-1369.
21. I. H. Stevenson *et al.*, Statistical assessment of the stability of neural movement representations. *Journal of neurophysiology* **106**, 764-774 (2011).
22. M. Prud'homme, J. F. Kalaska, Proprioceptive activity in primate primary somatosensory cortex during active arm reaching movements. *Journal of neurophysiology* **72**, 2280-2301 (1994).
23. K. Mizuseki, A. Sirota, E. Pastalkova, G. Buzsáki, Multi-unit recordings from the rat hippocampus made during open field foraging., (2009).
24. K. Mizuseki, A. Sirota, E. Pastalkova, G. Buzsáki, Theta oscillations provide temporal windows for local circuit computation in the entorhinal-hippocampal loop. *Neuron* **64**, 267-280 (2009).
25. E. N. Brown, L. M. Frank, D. Tang, M. C. Quirk, M. A. Wilson, A statistical paradigm for neural spike train decoding applied to position prediction from ensemble firing patterns of rat hippocampal place cells. *The Journal of Neuroscience* **18**, 7411-7425 (1998).
26. J. Aljadeff, B. J. Lansdell, A. L. Fairhall, D. Kleinfeld, Analysis of neuronal spike trains, deconstructed. *Neuron* **91**, 221-259 (2016).
27. O. Schwartz, J. W. Pillow, N. C. Rust, E. P. Simoncelli, Spike-triggered neural characterization. *Journal of Vision* **6**, 13-13 (2006).
28. P. Ramkumar *et al.*, Feature-based attention and spatial selection in frontal eye fields during natural scene search. *Journal of neurophysiology*, jn. 01044.02015 (2016).

29. H. L. Fernandes, I. H. Stevenson, A. N. Phillips, M. A. Segraves, K. P. Kording, Saliency and saccade encoding in the frontal eye field during natural scene search. *Cerebral Cortex* **24**, 3232-3245 (2014).
30. M. Saleh, K. Takahashi, N. G. Hatsopoulos, Encoding of coordinated reach and grasp trajectories in primary motor cortex. *The Journal of Neuroscience* **32**, 1220-1232 (2012).
31. P. Ramkumar *et al.*, Pyglmnet 1.0.1. (2017).
32. N. Srivastava, G. E. Hinton, A. Krizhevsky, I. Sutskever, R. Salakhutdinov, Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research* **15**, 1929-1958 (2014).
33. J. Snoek, H. Larochelle, R. P. Adams, in *Advances in neural information processing systems*. (2012), pp. 2951-2959.
34. BayesianOptimization. *GitHub repository*, (2016).
35. J. H. Friedman, Greedy function approximation: a gradient boosting machine. *Annals of statistics*, 1189-1232 (2001).
36. J. Friedman, T. Hastie, R. Tibshirani, Additive logistic regression: a statistical view of boosting (with discussion and a rejoinder by the authors). *The annals of statistics* **28**, 337-407 (2000).
37. T. K. Ho, The random subspace method for constructing decision forests. *IEEE transactions on pattern analysis and machine intelligence* **20**, 832-844 (1998).
38. D. H. Wolpert, Stacked generalization. *Neural networks* **5**, 241-259 (1992).
39. A. C. Cameron, F. A. Windmeijer, An R-squared measure of goodness of fit for some common nonlinear regression models. *Journal of Econometrics* **77**, 329-342 (1997).
40. T. A. Domencich, D. McFadden, "Urban travel demand-a behavioral analysis," (1975).
41. B. Amirikian, A. P. Georgopulos, Directional tuning profiles of motor cortical cells. *Neuroscience research* **36**, 73-79 (2000).
42. L. Paninski, S. Shoham, M. R. Fellows, N. G. Hatsopoulos, J. P. Donoghue, Superlinear population encoding of dynamic hand trajectory in primary motor cortex. *The Journal of neuroscience* **24**, 8551-8561 (2004).
43. E. A. Corbett, E. J. Perreault, K. P. Körding, Decoding with limited neural data: a mixture of time-warped trajectory models for directional reaches. *Journal of neural engineering* **9**, 036002 (2012).
44. B. McNaughton, C. A. Barnes, J. O'keefe, The contributions of position, direction, and velocity to single unit activity in the hippocampus of freely-moving rats. *Experimental Brain Research* **52**, 41-49 (1983).
45. Z. C. Lipton *et al.*, The Mythos of Model Interpretability. *IEEE Spectrum*, (2016).
46. J. McAuley, J. Leskovec, in *Proceedings of the 7th ACM conference on Recommender systems*. (ACM, 2013), pp. 165-172.
47. K. Simonyan, A. Vedaldi, A. Zisserman, Deep inside convolutional networks: Visualising image classification models and saliency maps. *arXiv preprint arXiv:1312.6034*, (2013).
48. J. Dushanova, J. Donoghue, Neurons in primary motor cortex engaged during action observation. *European Journal of Neuroscience* **31**, 386-398 (2010).
49. E. C. Smith, M. S. Lewicki, Efficient auditory coding. *Nature* **439**, 978-982 (2006).
50. I. Winkler, S. L. Denham, I. Nelken, Modeling the auditory scene: predictive regularity representations and perceptual objects. *Trends in cognitive sciences* **13**, 532-540 (2009).