

A guide for fitting generalized linear models in Tensorflow

Shih-Yi Tseng

Harvey Lab, Harvard Medical School

Generalized linear model (GLM)

- Flexible generalization of ordinary linear regression
 - Response variable y follows a specific distribution (*noise model*)
 - A function of the mean response variable (*link function*) varies linearly with the predictors X

$$f(\mu) = Xw, y \sim \text{Dist}(\mu, \dots)$$

Model type	Distribution	Link function	Mean function (inverse function, activation function)
Linear regression	Normal (Gaussian)	Identity $f(\mu) = \mu$	Linear $\mu = Xw$
Poisson regression	Poisson	Log $f(\mu) = \log(\mu)$	Exponential $\mu = \exp(Xw)$
Logistic regression	Bernoulli	Logit $f(\mu) = \log(\mu/(1 - \mu))$	Sigmoid $\mu = \exp(Xw) / (1 + \exp(Xw))$

Fitting GLM in Tensorflow

- Why Tensorflow?
 - GPU acceleration for massive parallelization
- A typical session with simultaneous recording
 - 200-300 neurons per imaging plane
 - ~30,000 timepoints
 - ~1000 predictors (after basis expansion)

$$f(Y) = XW \leftarrow 1000 \times 250$$

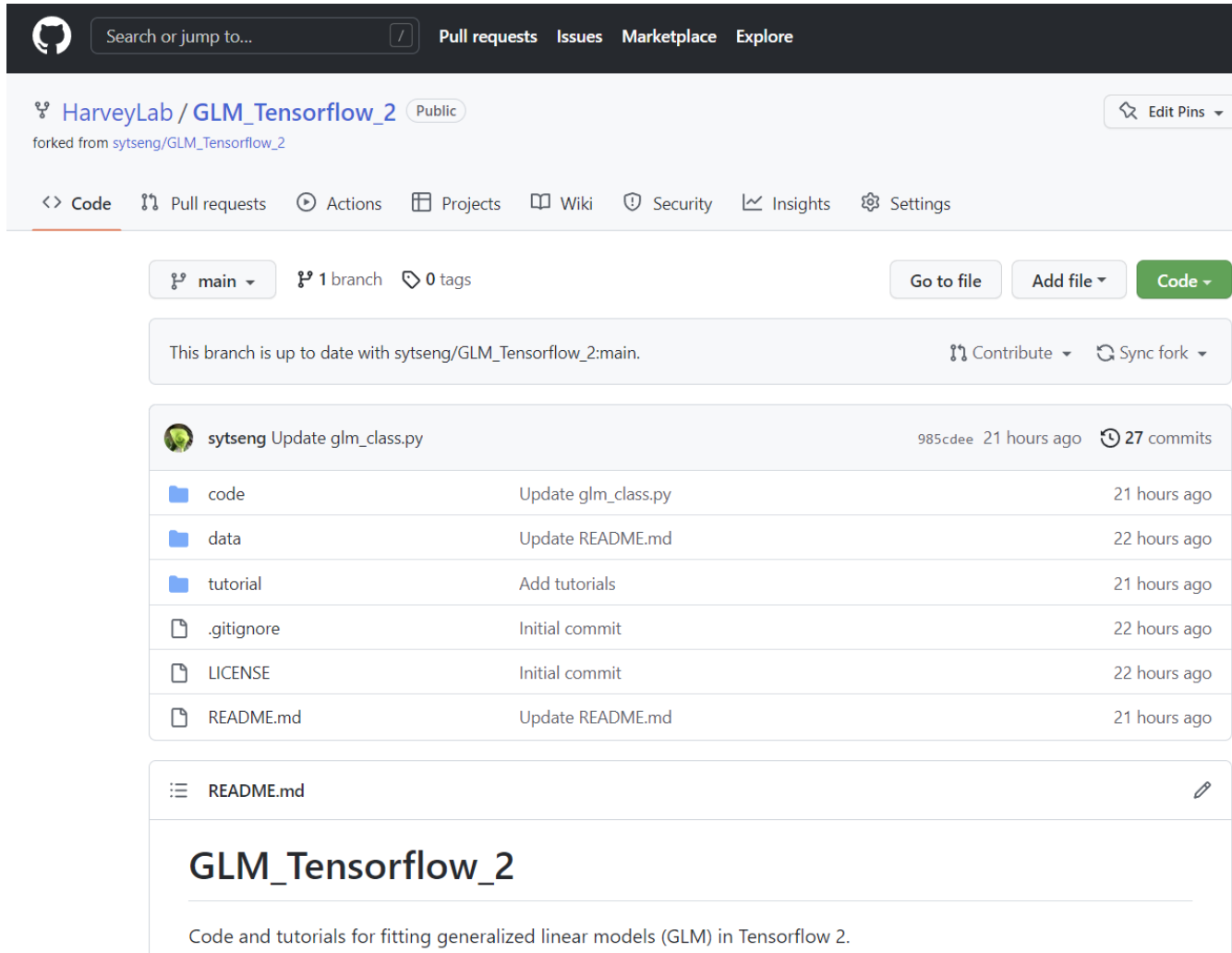
↑
30,000 × 250 ↖
30,000 × 1000

- Formalized as a multi-output, one-layer feedforward neural network
- Combine all outputs for a single loss
- Gradient descent optimization

Fitting takes 5-10 min for Poisson GLMs (1-2 sec per neuron!)
with 5-fold cross validation over >10 regularization values

Github repo for GLM in Tensorflow

https://github.com/HarveyLab/GLM_Tensorflow_2



Search or jump to... Pull requests Issues Marketplace Explore

HarveyLab / GLM_Tensorflow_2 Public

forked from sytseng/GLM_Tensorflow_2

<> Code Pull requests Actions Projects Wiki Security Insights Settings

main 1 branch 0 tags

Go to file Add file Code

This branch is up to date with sytseng/GLM_Tensorflow_2:main. Contribute Sync fork

sytseng Update glm_class.py 985cdee 21 hours ago 27 commits

code	Update glm_class.py	21 hours ago
data	Update README.md	22 hours ago
tutorial	Add tutorials	21 hours ago
.gitignore	Initial commit	22 hours ago
LICENSE	Initial commit	22 hours ago
README.md	Update README.md	21 hours ago

README.md

GLM_Tensorflow_2

Code and tutorials for fitting generalized linear models (GLM) in Tensorflow 2.

Tutorials for fitting GLM in Tensorflow

- Two tutorials
 - `Tutorial_for_using_GLM_class.ipynb`
 - `Tutorial_for_fitting_neural_calcium_imaging_data_with_GLM.ipynb`

GLM class

- Written in Python, object-oriented
- Can be loaded as a module

```
import glm_class as glm
```

- Initialization

```
model = glm.GLM() or model = glm.GLM_CV()
```

- Four class methods

```
model.fit()  
model.select_model()  
model.predict()  
model.evaluate()
```

Model initialization

```
model = glm.GLM(activation = 'linear', loss_type = 'gaussian',  
                regularization = 'elastic_net', lambda_series = 10.0 ** np.linspace(3, -6, 10),  
                l1_ratio = 0., smooth_strength = 0.,  
                optimizer = 'adam', learning_rate = 1e-2, momentum = 0.5,  
                min_iter_per_lambda = 100, max_iter_per_lambda = 10**4,  
                num_iter_check = 100, convergence_tol = 1e-6)
```

- Four types of input arguments
 - Model type:
 - *activation*: 'linear', 'exp', 'sigmoid', 'relu', 'softplus'
 - *loss_type*: 'gaussian', 'poisson', 'binominal'
 - Regularization:
 - *regularization*: 'elastic_net', 'group_lasso'
 - *lambda_series*: list of regularization strengths in descending order
 - *l1_ratio*: 0 for L2/ridge, 1 for L1/Lasso, or btw 0-1 for elastic net
 - *smooth_strength*: add smoothness penalty
 - Optimization: *optimizer* ('adam', 'sgdm'), *learning_rate*, *momentum*
 - Convergence: *min_iter_per_lambda*, *max_iter_per_lambda*, *num_iter_check*, *convergence_tol*

Class methods

Fitting (with training data)

```
model.fit(X, Y, [initial_w0, initial_w,  
               feature_group_size, verbose])
```

A list containing the size of each group
For group_lasso or smoothness penalty

Model selection (with validation data)

```
model.select_model(X_val, Y_val, [min_lambda, make_fig])
```

Evaluation (with training, validation, and test data)

```
frac_dev_expl, dev_model, dev_null, dev_expl  
= model.evaluate(X, Y, [make_fig])
```

Prediction (with training, validation, and test data)

```
Y_pred = model.predict(X)
```


Model attributes

- Loss & lambda trace

`model.loss_trace` and `model.lambda_trace`

- Selected weights & intercepts

`model.selected_w` and `model.selected_w0`

- Selected lambda & lambda indices

`model.selected_lambda` and `model.selected_lambda_ind`

GLM with cross validation

- Model initialization

```
model_cv = glm.GLM_CV(n_folds = 5, auto_split = True, split_by_group = True, split_random_state = 42,  
    activation = 'linear', loss_type = 'gaussian',  
    regularization = 'elastic_net', lambda_series = 10.0 ** np.linspace(3, -6, 10),  
    l1_ratio = 0., smooth_strength = 0.,  
    optimizer = 'adam', learning_rate = 1e-2, momentum = 0.5,  
    min_iter_per_lambda = 100, max_iter_per_lambda = 10**4,  
    num_iter_check = 100, convergence_tol = 1e-6)
```

- Additional input arguments
 - *n_folds*: number of CV folds
 - *auto_split*: perform CV split automatically
 - *split_by_group*: split CV folds according to a third-party group provided during fitting time
 - *split_random_state*: Numpy random state for splitting

GLM with cross validation

- Model fitting

Train and validation indices for each fold
if *auto_split = False*

Third party group info
if *split_by_group = True*

```
model_cv.fit(X, Y, [train_idx, val_idx, group_idx,  
                  initial_w0, initial_w,  
                  feature_group_size, verbose])
```

- Model selection (no validation data required)

```
model_cv.select_model([se_fraction, min_lambda, make_fig])
```

Controls the tolerance of choosing models with
smallest deviance vs. larger regularization;
0 for model with minimal deviance, 1 for 1SE rule

- Same methods for model evaluation and prediction

Useful utility functions

- Null deviance (avg for each response, or for every datapoint)

```
null_dev = null_deviance(y, loss_type = 'poisson')  
null_dev_pt = pointwise_null_deviance(y, loss_type = 'poisson')
```

- Deviance (avg for each response, or for every datapoint)

```
frac_dev_expl, d_model, d_null = deviance(y_pred, y_true, loss_type = 'poisson')  
dev_pt = pointwise_deviance(y_true, y_pred, loss_type = 'poisson')
```

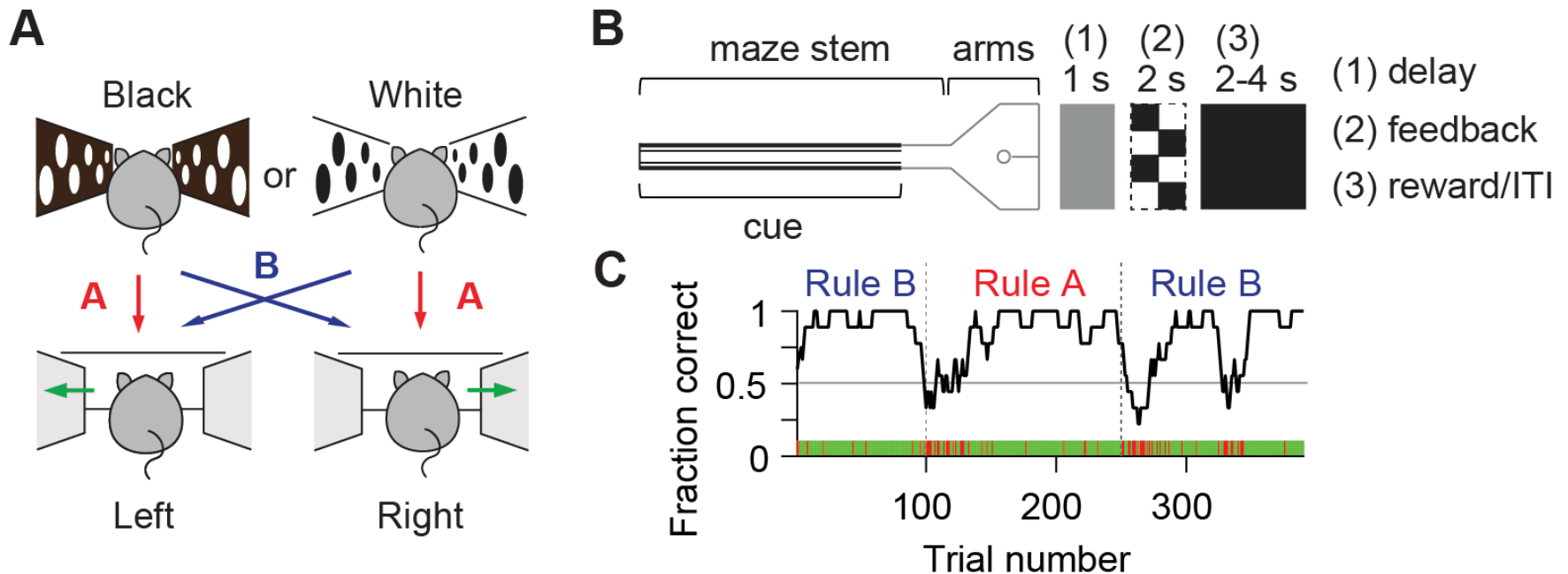
- Make prediction given design matrix, weights and intercepts, and activation function

```
prediction = make_prediction(X, w, w0, activation = 'exp')
```

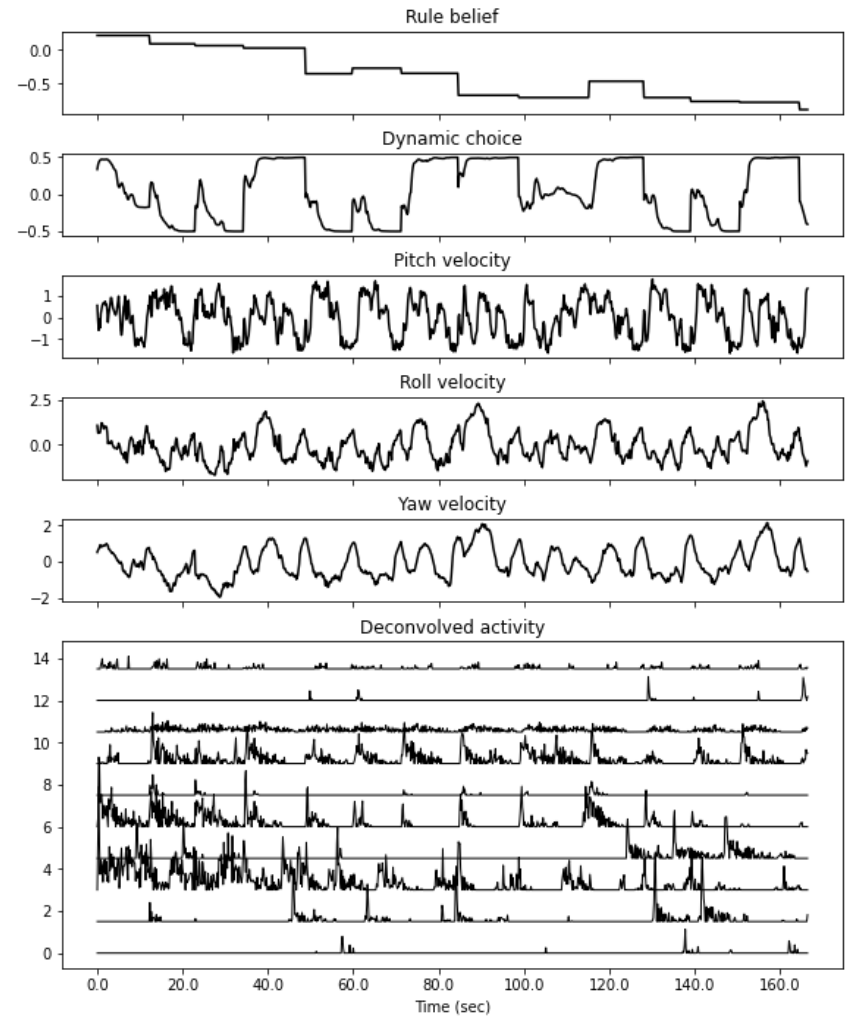
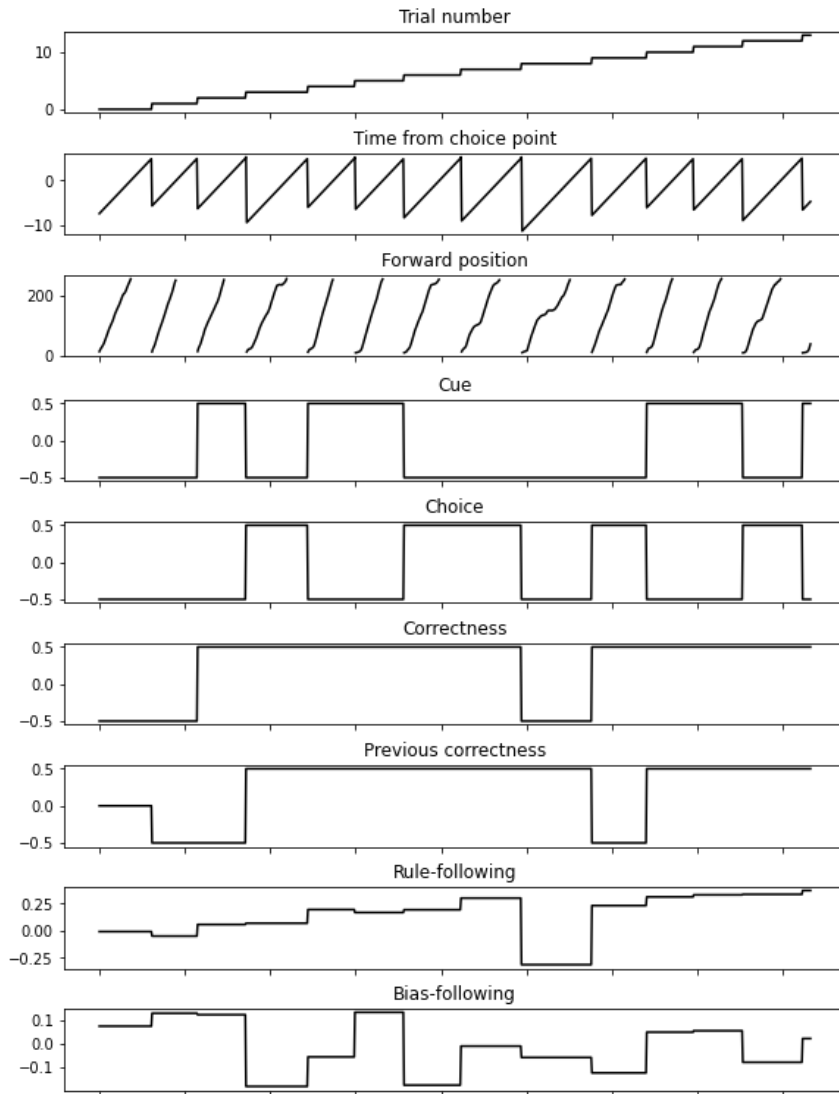
Fitting GLM to calcium imaging data

Tutorial_for_fitting_neural_calcium_imaging_data_with_GLM.ipynb

- Building design matrix (basis expansion)
- Model fitting, selection and evaluation
- Quantification of feature contribution

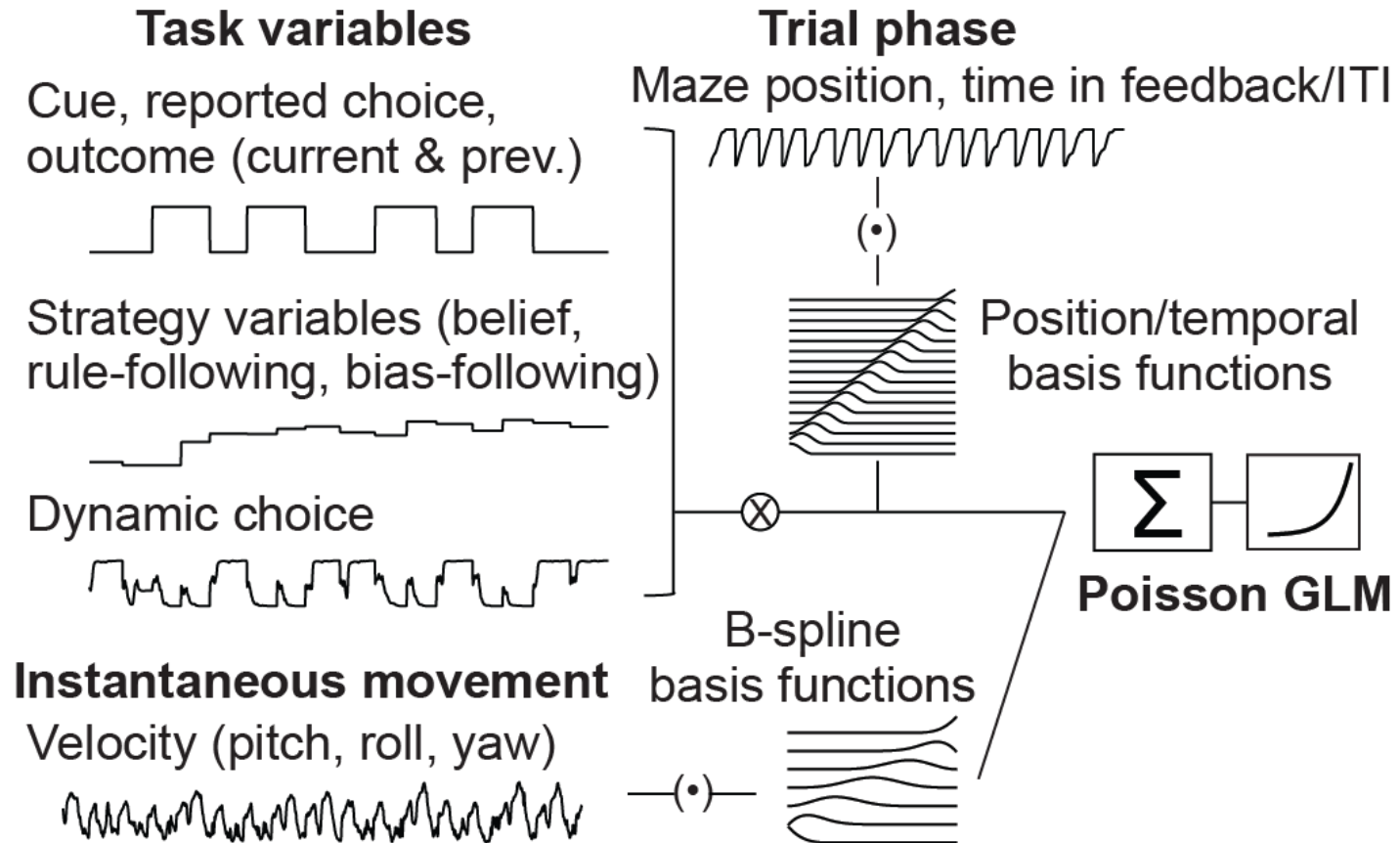


Predictors vs. neural activity



100 RSC neurons

Model schematic



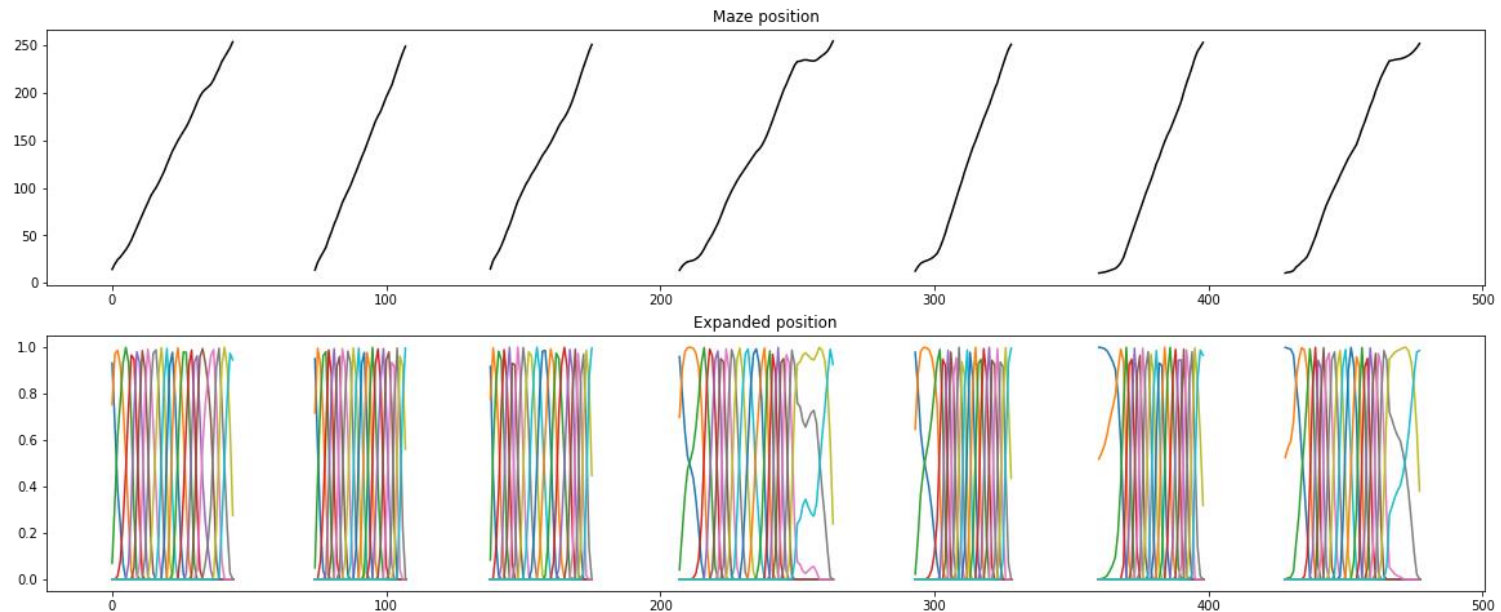
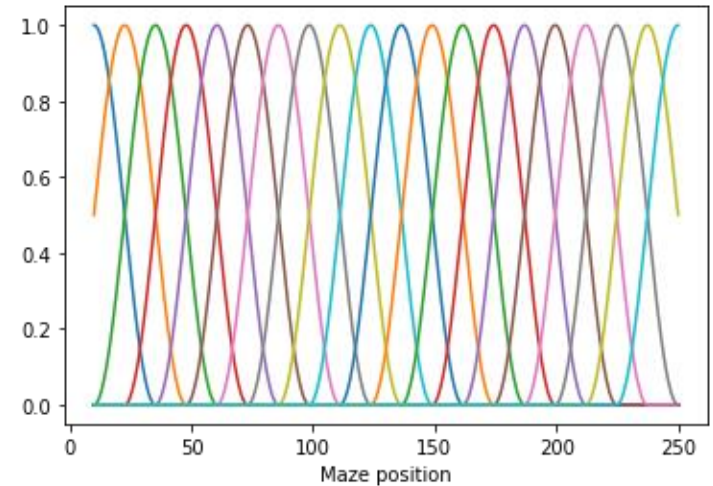
Position basis expansion

“Place field-like” tuning

Raised cosine bumps

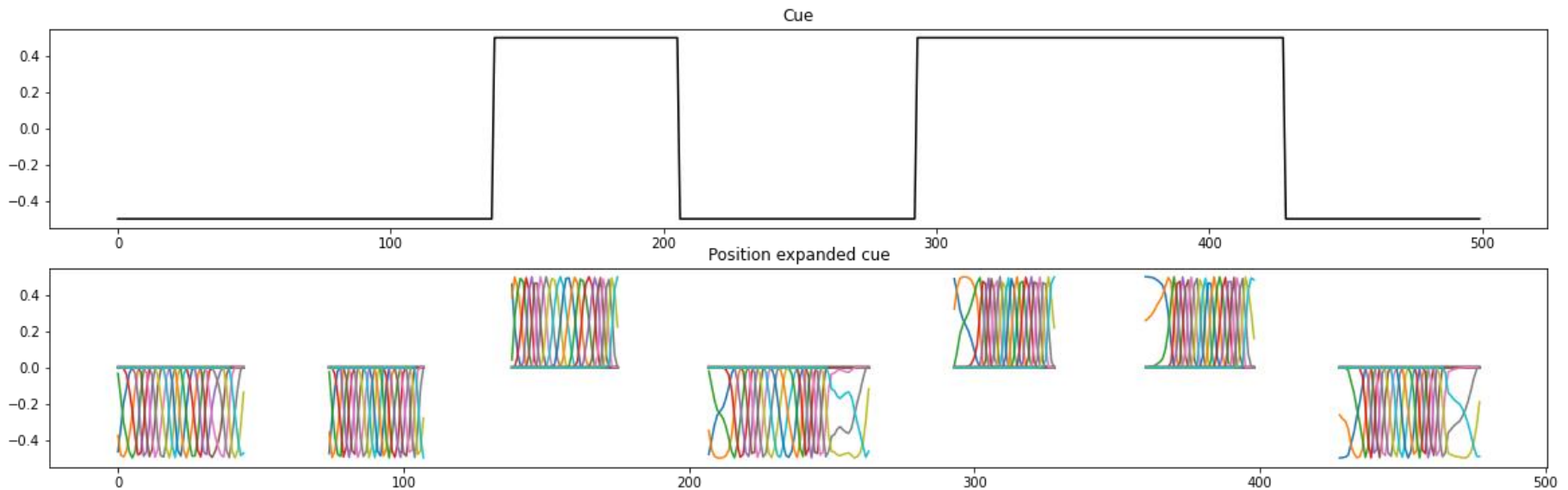
$$f_i(x) = \begin{cases} \frac{1}{2}\cos(ax - \varphi_i) + \frac{1}{2} & \text{if } \varphi_i - \pi < ax \leq \varphi_i + \pi \\ 0 & \text{otherwise} \end{cases}$$

20 position bases spanning the maze



Load task variables onto position basis functions

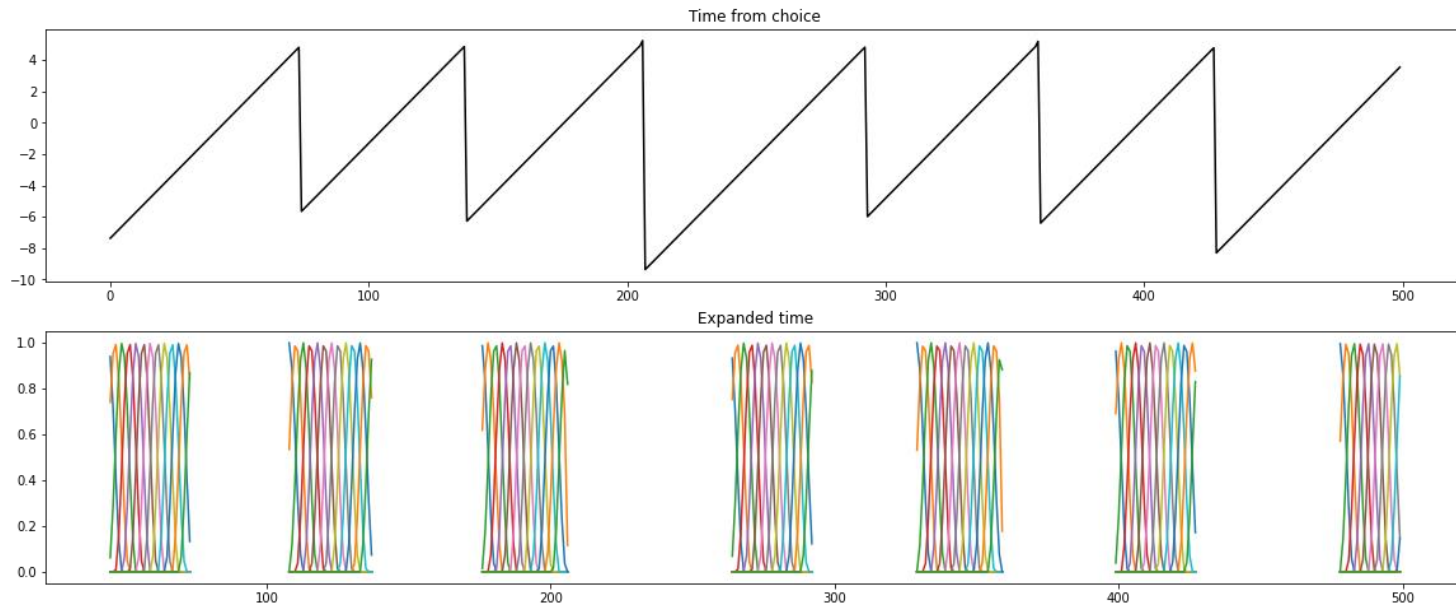
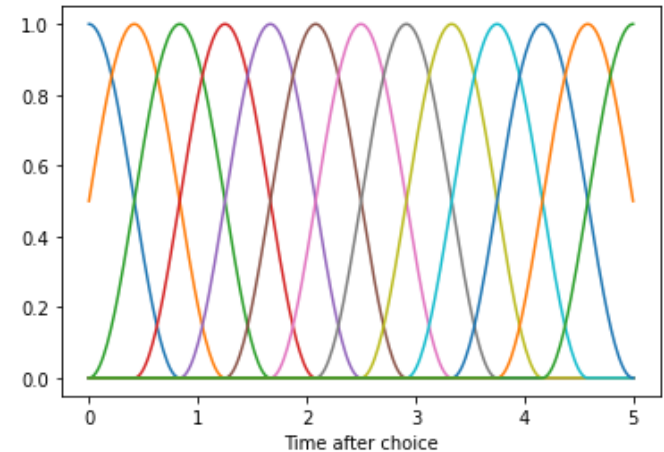
Simply take the **product** of the time series of task variables and each of the position basis functions, i.e. create **interaction terms**



Temporal basis expansion

For time elapsed in feedback/ITI,
follow the same principle to create
temporal basis functions and load task
variables onto them

13 temporal bases spanning feedback/ITI



B-spline expansion for velocities

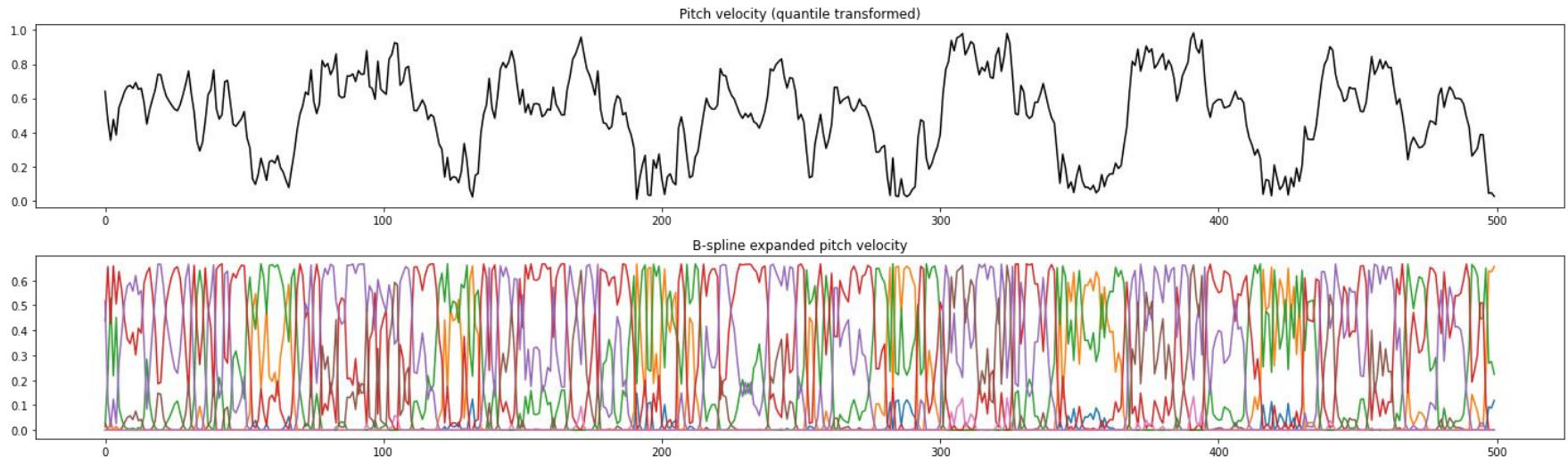
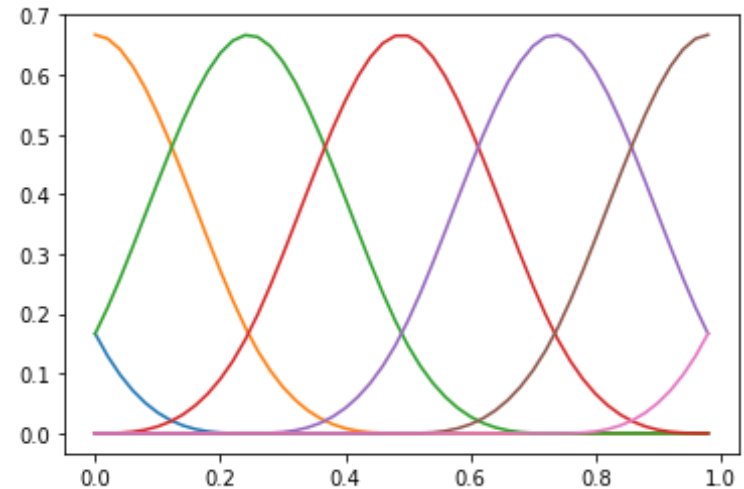
For movement variables, we assume *consistent relationship* btw neural activity and velocities (in maze vs. ITI)

Non-linear velocity tuning:

B-spline expansion (degree 3 polynomial)

Quantile transformation: easy comparison

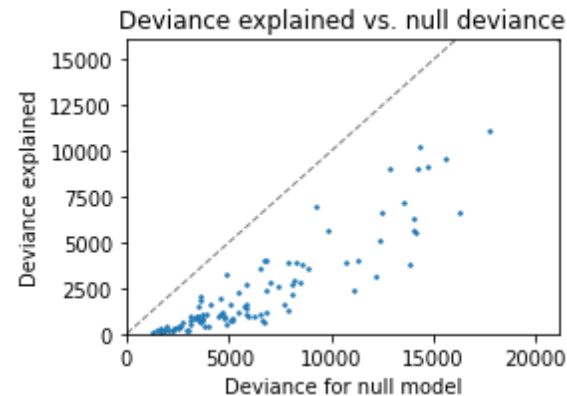
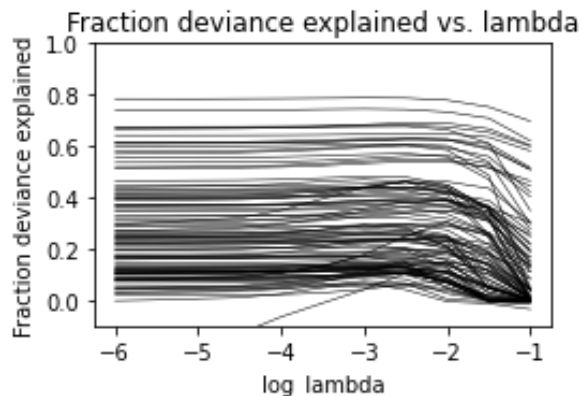
7 b-splines spanning the quantile range



Fitting

- Poisson GLM, 5-fold cross validation
 - $Y = \exp(XW + W_0)$
 - Multiply deconvolved activity by 10 -> mimic spike count
- Regularization: group lasso
 - L2 for features belong to the same basis expanded groups
 - L1 between basis expanded groups
- Evaluation

$$\text{fraction deviance explained} = 1 - \frac{\text{model deviance}}{\text{null deviance}}$$

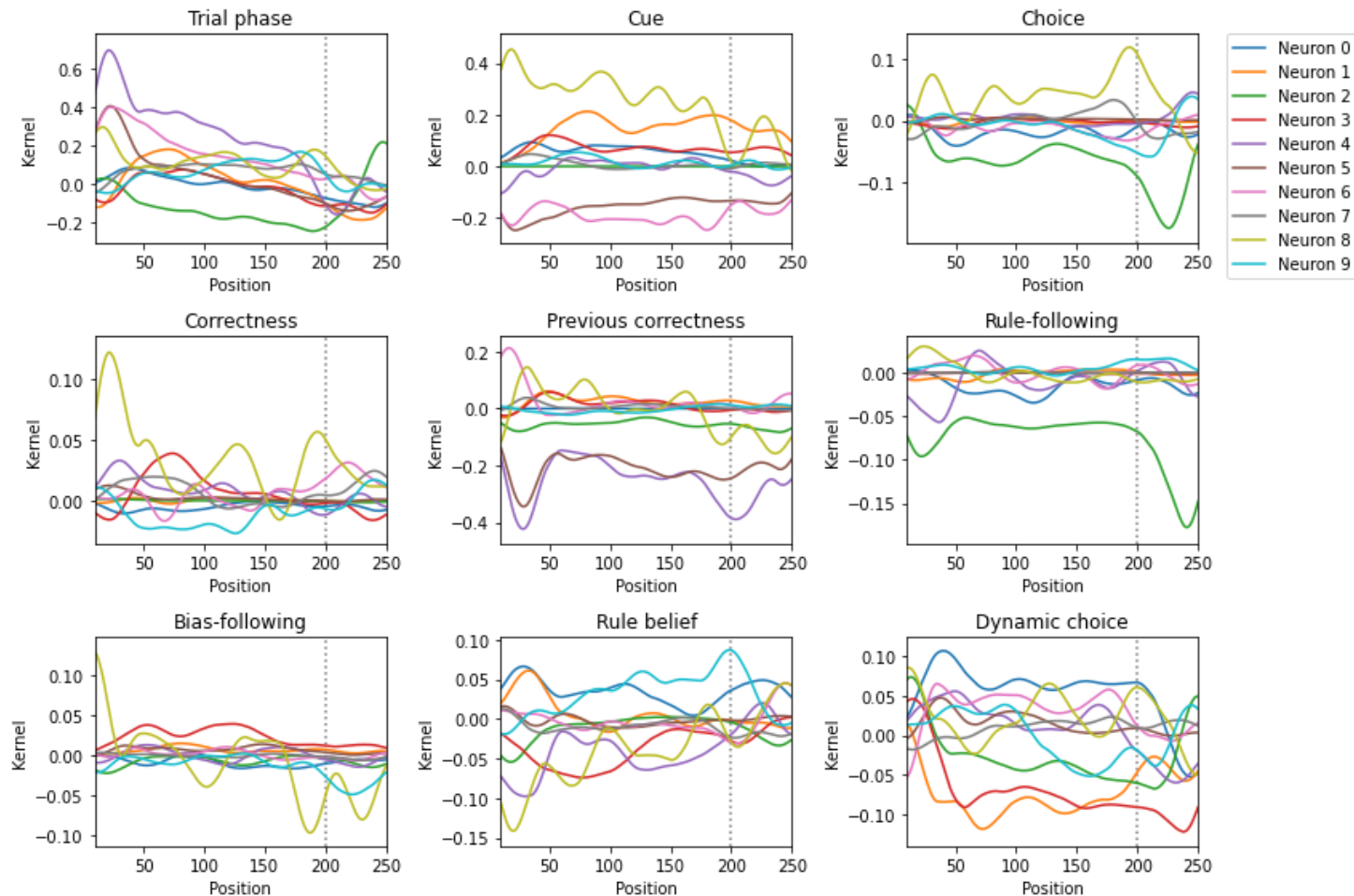


Quantification of feature contribution

- Model weights (kernels)
- Fraction null deviance or fraction explained deviance through model breakdown
- Fraction null deviance through model comparison with re-fitting

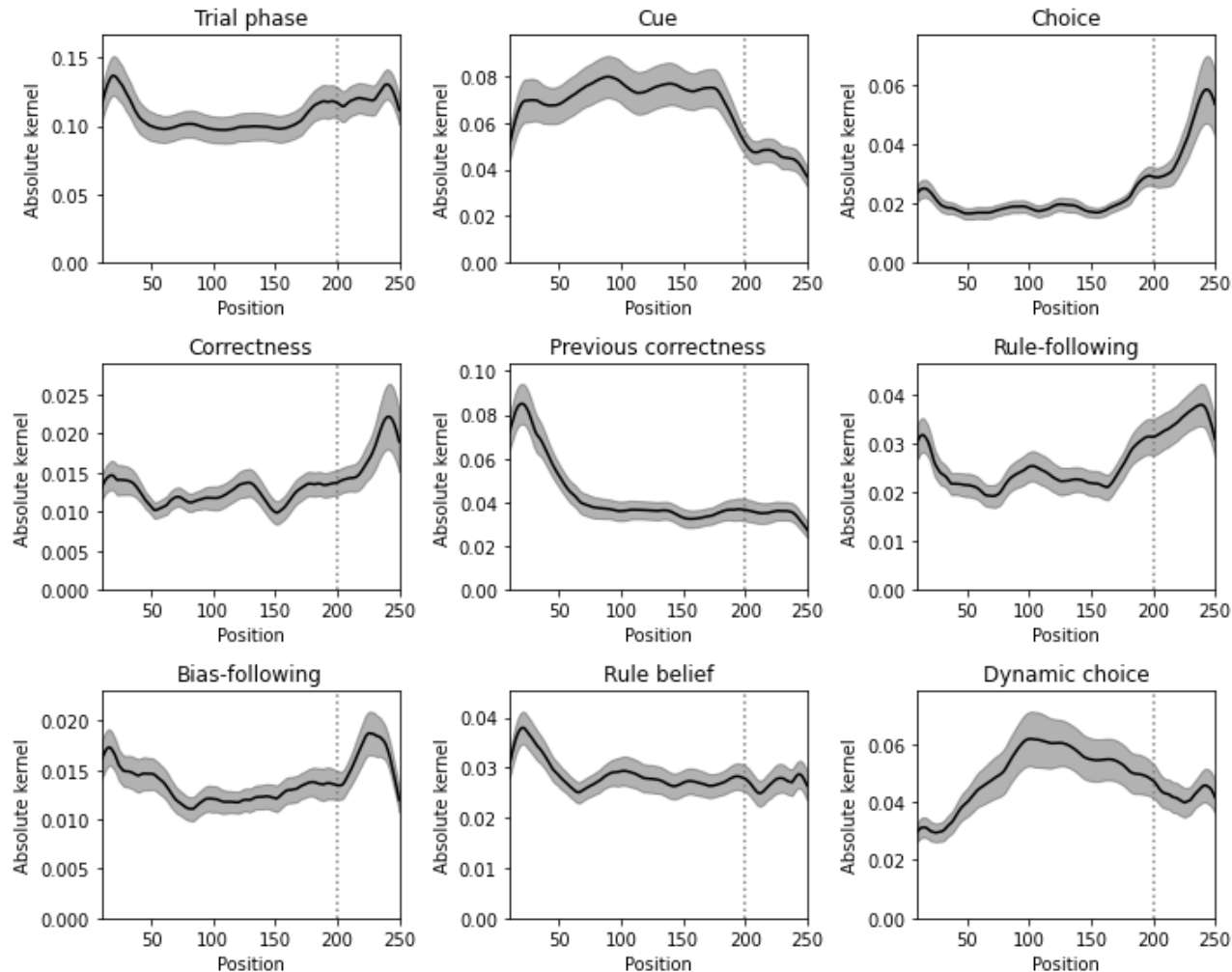
Model weights (kernels)

- Weight vector \times basis functions \rightarrow sum up into kernels



Model weights (kernels)

- Mean absolute kernels: strength of tuning



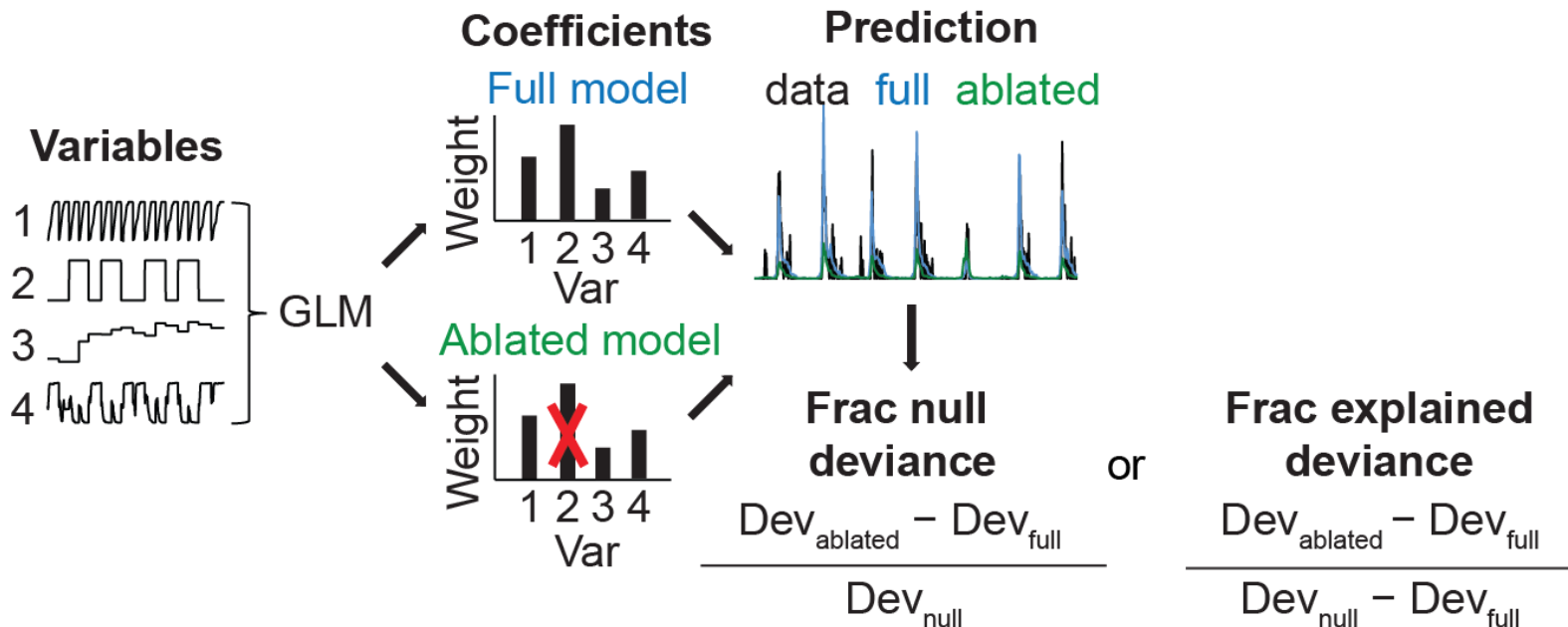
Model weights (kernels)

- Multiplicative factor on top of baseline activity
 - $y = \exp(w_0 + w_1x_1 + w_2x_2 + \dots)$
 - Exponentiate to obtain “gain”
- Informative of directionality (signed)
- Robust to scaling of y
- Hard to interpret the magnitude and compare across variables
 - Better with “fraction null deviance” or “fraction explained deviance”

Model breakdown

“Model breakdown”

Quantification of variable contribution (without re-fitting)

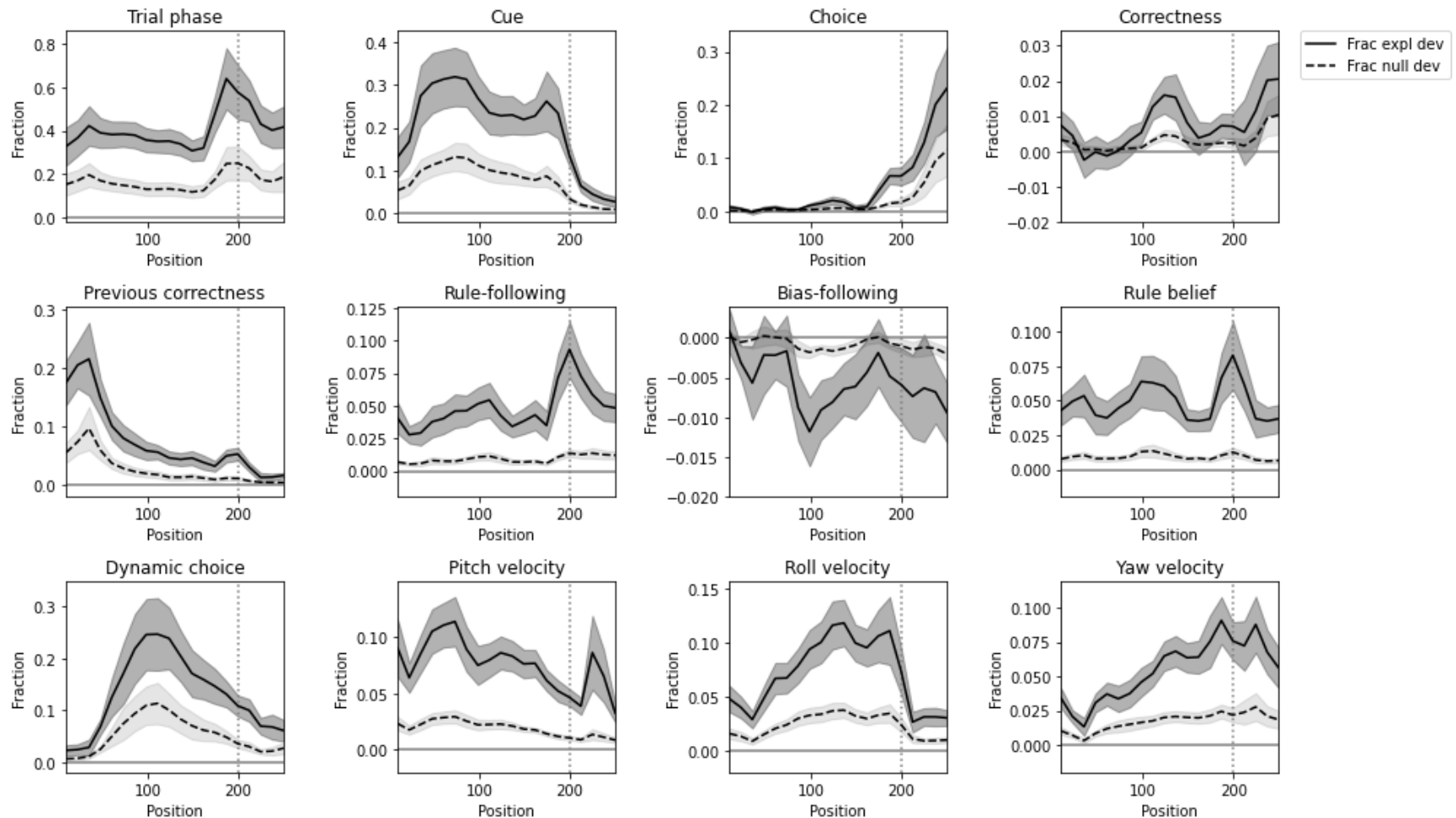


Zero weights = zero variables (must be zero-centered) = shuffle variables

Evaluate on CV held-out data (85%) instead of test data (15%)

Model breakdown

Evaluate fraction null deviance or fraction explained deviance at each position (or time) bin



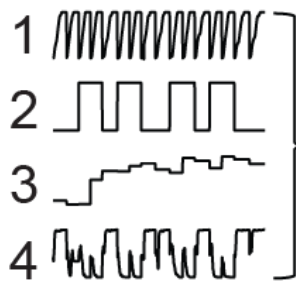
Model comparison (with re-fitting)

“Model comparison”

Quantification of variable contribution (with re-fitting)

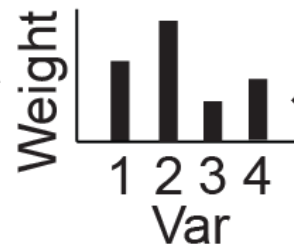
Variables

Full model

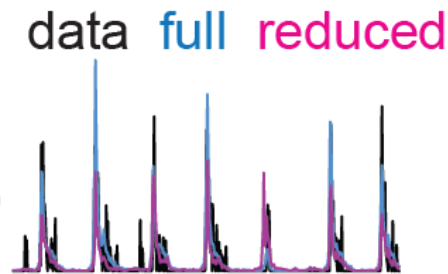


GLM →

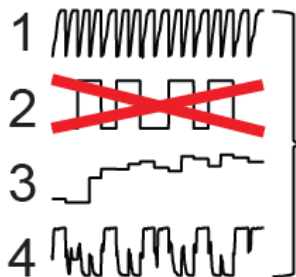
Coefficients



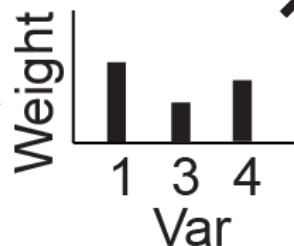
Prediction



Reduced model



GLM →



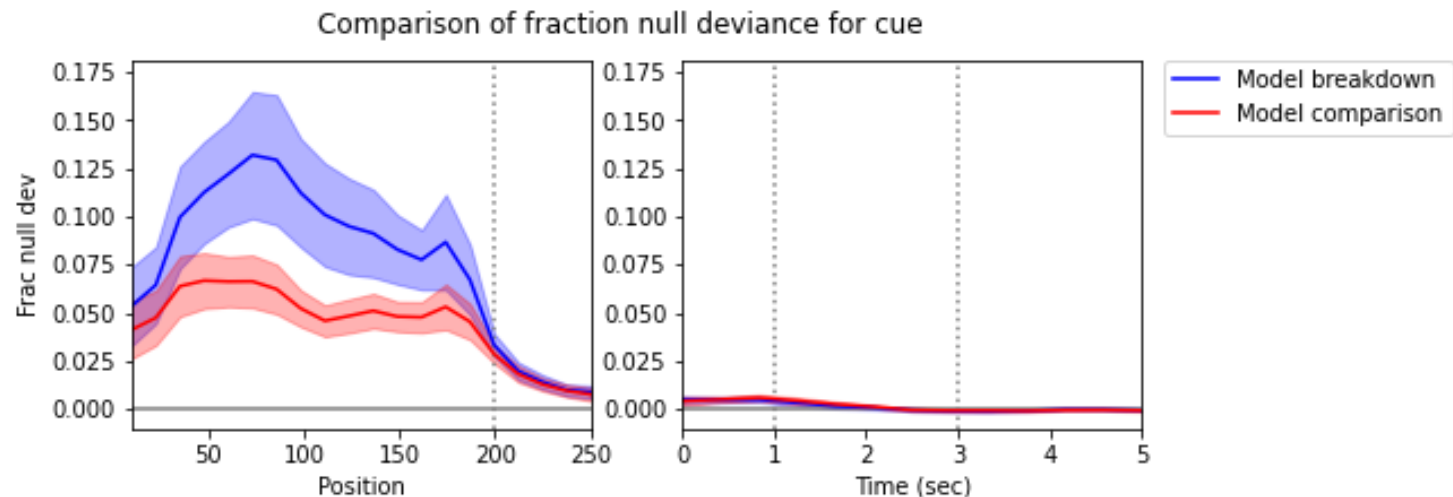
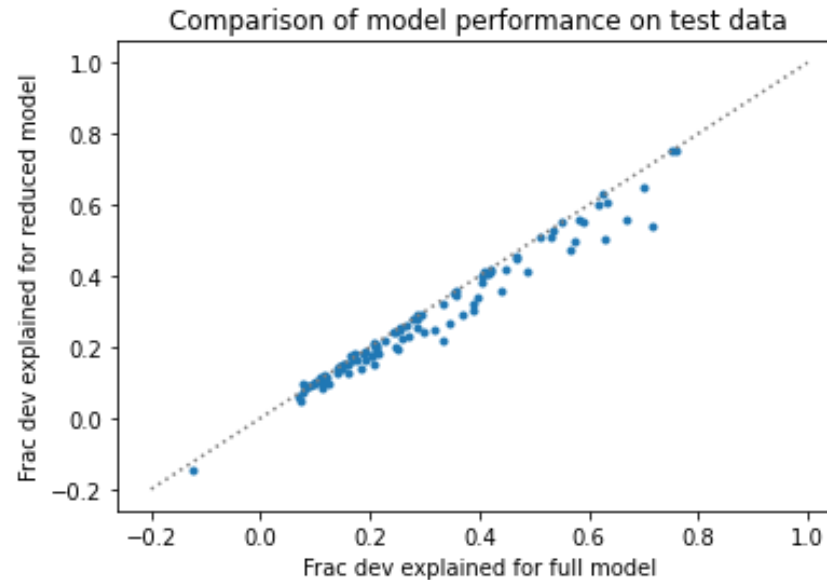
Frac null deviance

$$\frac{\text{Dev}_{\text{reduced}} - \text{Dev}_{\text{full}}}{\text{Dev}_{\text{null}}}$$

Remove variable = shuffle variable

Model comparison (with re-fitting)

For cue



Model comparison vs. breakdown

Model breakdown (fraction null deviance and fraction explained deviance)

- The metrics are easy to interpret and compare across variables
- Does not tell you the directionality of tuning
- Does not require re-fitting the model (fast to do)
- Not a linear breakdown for Poisson GLMs, but linear for Gaussian
- Can be less biased in assigning contribution of correlated variables, if the model is not terribly mis-specified

Model comparison (fraction null deviance)

- The metric is easy to interpret and compare across variables
- Does not tell you the directionality of tuning
- Require re-fitting of a model for each variable (computationally demanding)
- Extract the "unique" portion of deviance explained, can be biased (and underestimate the contribution of a variable) when there are correlated variables and you don't know which is actually contributing to response

References

- Tseng, S.-Y., Chettih, S.N., Arlt, C., Barroso-Luque, R., and Harvey, C.D. (2022). Shared and specialized coding across posterior cortical areas for dynamic navigation decisions. *Neuron* 110, 2484–2502.e16. [\[link\]](#)
- Minderer, M., Brown, K.D., and Harvey, C.D. (2019). The spatial structure of neural encoding in mouse posterior cortex during navigation. *Neuron* 102, 232–248.e11. [\[link\]](#)