Lecture 4: Technical Tools I — Working with Neural Data
2018.10.03
NB: This lecture should be read in conjunction with the tutorial at:

What is Data?

What is a model (of neural or behavioral data)?
What is Data?

What is a model (of neural or behavioral data)?

What is (neural or behavioral) data?
What is Data?

**Def’n:** Data is a set of linked tensors.
**Def’n:** Data is a set of linked tensors.

Vectors $= \text{list of numbers}$

$$v = [0.4, 0.2, -1.144, \ldots, 9.5]$$
What is Data?

**Def’n:** Data is a set of linked tensors.

**Vectors** $= \text{list of numbers}$

$$v = [0.4, 0.2, -1.144, \ldots, 9.5]$$

$$v = [v_0, v_1, v_2, \ldots, v_{N-1}]$$

“components”
Def’n: Data is a set of linked tensors.

Vectors = list of numbers

\[ \mathbf{v} = [0.4, 0.2, -1.144, \ldots, 9.5] \]

\[ \mathbf{v} = [v_0, v_1, v_2, \ldots, v_{N-1}] \]

\[ N = "\text{"dimension" of the data} = \text{length of the list} \]
What is Data?

**Def’n:** Data is a set of linked tensors.

**Matrices** = 2-dimensional array of numbers

\[
M = \begin{bmatrix}
1.1 & -0.3 & -0.22 & 0.64 & 1.9 \\
9.1 & -0.91 & 0.31 & 0.441 & 10.3 \\
::: & ::: & ::: & ::: & ::: \\
3.2 & -0.131 & 1.5 & \ldots & 0.3
\end{bmatrix}
\]
Def’n: Data is a set of linked tensors.

Matrices = 2-dimensional array of numbers

\[
M = \begin{bmatrix}
    x_{11} & x_{12} & x_{13} & \cdots & x_{1n} \\
    x_{21} & x_{22} & x_{23} & \cdots & x_{2n} \\
    \vdots & \vdots & \vdots & \ddots & \vdots \\
    x_{m1} & x_{m2} & x_{m3} & \cdots & x_{mn}
\end{bmatrix}
\]

\[(m, n) = “\text{shape}” \text{ of the data.}\]
**Def’n:** Data is a set of linked tensors.

<table>
<thead>
<tr>
<th>Vector</th>
<th>Matrix</th>
<th>Data Cube</th>
</tr>
</thead>
<tbody>
<tr>
<td>'t'</td>
<td>3 1 4 1</td>
<td></td>
</tr>
<tr>
<td>'e'</td>
<td>5 9 2 6</td>
<td></td>
</tr>
<tr>
<td>'n'</td>
<td>5 3 5 8</td>
<td></td>
</tr>
<tr>
<td>'s'</td>
<td>9 7 9 3</td>
<td></td>
</tr>
<tr>
<td>'o'</td>
<td>2 3 8 4</td>
<td></td>
</tr>
<tr>
<td>'r'</td>
<td>6 2 6 4</td>
<td></td>
</tr>
</tbody>
</table>

$D$ = dimension of the tensor

$(n_1, \ldots, n_D) = \text{shape}$
What is Data?

**Def’n:** Data is a set of linked tensors.

Color Image = typically, a 3 Tensor

two bears.jpg
Def’n: Data is a set of linked tensors.

Color Image = typically, a 3 Tensor

two bears.jpg

[IPYNB: “Loading Images as Python Arrays”]
**Def’n:** Data is a set of linked tensors.

Color Image = typically, a 3 Tensor

3-Tensor with shape (194, 260, 3)
- rows
- cols
- color channels
Def’n: Data is a set of linked tensors.
Def’n: Data is a set of linked tensors.

Let’s explain what this means in terms of a real example.
**Ventral Cortex Data:** from multi-array electrophysiology in macaques


Multi-array electrophysiology in macaque V4 and IT.

About 300 total sites

Ha Hong
Jim DiCarlo
The Ventral Visual Pathway

5760 images

64 objects

8 categories

uncorrelated photo backgrounds

Animals  Boats  Cars  Chairs  Faces  Fruits  Planes  Tables

Pose, position, scale, and background variation

Low variation

Medium variation

High variation

... 640 images

... 2560 images

... 2560 images
The Ventral Visual Pathway

About 300 total sites
So what’s the tensors in the data here?

HDF-5 data format

**h5py** python package
So what’s the tensors in the data here?

https://cran.r-project.org/web/packages/h5/vignettes/h5-Intro.html

**h5 - An Object Oriented Interface to HDF5**

Mario Annau

2017-08-30

**Introduction**

The Hierarchical Data Format 5 (HDF5) is a binary data format and API created by the (HDF-Group 1997–2016) to better meet ever-increasing data storage demands of the scientific computing community. HDF5 files store homogeneous, multidimensional data sets organized in groups similar to the folder structure of a file system. As a self-describing file format HDF5 objects can be annotated with meta data using attributes. Compared to R’s integrated binary format HDF5 has various advantages.

- **Language Independence** HDF5 is implemented in C and includes APIs for a wide range of programming languages like e.g. C++, Fortran, Python and Matlab.
- **Partial I/O** HDF5 files support direct access to parts of the file without first parsing the entire contents, thus can process data sets not fitting into memory.
- **Optimization** Access performance to parts of the HDF5 file can be further tuned by specifying the memory layout. The defined chunks can be cached in memory to further improve access times for subsequent queries.
So what’s the tensors in the data here?

[IPYNB: Ventral stream neural data]
Neural Data

So what’s the tensors in the data here?

1. The Stimuli (the images)
   
   \[ \text{shape} = (5760, 256, 256) \]
   
   stimulus number rows cols (no color channels since gray-scale images)

2. The Stimulus Metadata (the images)

   \[ \text{shape} = (5760, \# \text{ of attributes}) \]

3. The “Raw” Neural Data
   
   \[ \text{shape} = (5760, 296, 10, \sim 50) \]

   stimulus number neurons timebins repetitions of same image

   + various derived data tensors (time, trial averaging)
Names of dimensions informally create links between tensors.

Neural Data
Data Slice 2 = $F(data\_slice\_1)$

Data Slice 2 = $F_{\text{params}}(data\_slice\_1)$

Data Slice 2 = $F_{\text{params}}(data\_slice\_1) + \text{Noise}$

What is a Model?
Main Goals of Class

Learn how to

1) Build (high-dimensional) models of/for neural & behavioral data

\[
\text{output} = F_{[\text{params}]}(\text{input})
\]

formulate \( F \) mathematically
on a computer
Main Goals of Class

Learn how to

1) Build (high-dimensional) models of/for neural & behavioral data

\[ \text{output} = F_{[\text{params}]}(\text{input}) \]

(formulate \( F \) mathematically on a computer)

2) Train such models

\[ \text{data\_slice\_2} = F_{[\text{params}]}(\text{data\_slice\_1}) \]

determine \text{params}

(form from pair of linked data tensors)
Main Goals of Class

Learn how to

1) Build (high-dimensional) models of/for neural & behavioral data

\[ \text{output} = F_{[\text{params}]}(\text{input}) \]

formulate \( F \) mathematically on a computer

2) Train such models

\[ \text{data}_\text{slice}_2 = F_{[\text{params}]}(\text{data}_\text{slice}_1) \]

determine \( \text{params} \) from pair of linked data tensors

3) Evaluate and compare such models

\[ \text{new}_\text{data}_\text{slice}_2 = F_{[\text{params}]}(\text{new}_\text{data}_\text{slice}_1) \]

??
Simple Correlation Analysis

So what can the data tell us?

[IPYNB: Simple Correlation Analysis]
Main Goals of Class

Stimulus → Neurons → Behavior

- Representation
- Read-out

Stimulus:
- Category
- Location
- Size
- Pose
- Depth relationships

Neurons:
- Visual representation

Behavior:
Main Goals of Class

Pixel representation

IT neural representation

Behavioral patterns

“animal”
“boat”
“car”

Stimulus representation

Neurons read-out

Behavior

Pixel representation encoding

IT neural representation decoding

Stimulus

Neurons

Behavior

Visual presentation

RGC LGN V1 V2 V4

T(•)

PIT

V2

V4

100ms

Stimulus

Neurons

Behavior

Pixel representation

IT neural representation

Behavioral patterns

“animal”
“boat”
“car”

Stimulus representation

Neurons read-out

Behavior

Pixel representation encoding

IT neural representation decoding

Stimulus

Neurons

Behavior

Visual presentation

RGC LGN V1 V2 V4

T(•)

PIT

V2

V4

100ms

Stimulus

Neurons

Behavior

Visual presentation

RGC LGN V1 V2 V4

T(•)

PIT

V2

V4

100ms

Category

Location

Size

Pose

Depth relationships
Main Goals of Class

Pixel representation

Stimulus

Pixel representation

IT neural representation

Behavioral patterns

“animal”

“boat”

“car”

Both of these things are models

Visual representation

Category

Location

Size

Pose

Depth relationships
Data Slice 2 = $F_{\text{params}}(\text{data\_slice\_1})$
Data Slice 2 = $F_{\text{params}}(\text{data\_slice\_1})$

IT neural representation

Behavioral patterns
“animal”
“boat”
“car”

decoding
Data Slice 2 = $F_{\text{params}}(\text{data\_slice\_1})$

- **Pixel representation**
  - `data\_slice\_1 = stimuli`

- **IT neural representation**
  - `data\_slice\_2 = neural data`

- **Behavioral patterns**
  - “animal”
  - “boat”
  - “car”

**Encoding**

**Decoding**
Sometimes the model has *internal structure* that can be connected to actual structural mechanisms of the real system (e.g. the brain).

**Pixel representation** → **encoding** → **IT neural representation** → **decoding** → **Behavioral patterns**

Data Slice 2 = \( F_{\text{params}}(\text{data_slice}_1) \)

\( F \rightarrow \) convnet architecture
params → filters
Sometimes the model has internal structure that can be connected to actual structural mechanisms of the real system (e.g. the brain).

**Pixel representation**

\[ \text{Data Slice 2} = F_{\text{params}}(\text{data_slice_1}) \]

\[ F \rightarrow \text{shallow “neural net” (single layer)} \]

\[ \text{params} \rightarrow \text{weights + bias} \]

**IT neural representation**

**Behavioral patterns**

- “animal”
- “boat”
- “car”

\[ \text{linear SVM + boundary} \]

**decoding**

neurons in decision areas downstream of visual system
Behavior = Feature space + Simple decision rule
= encoding + decoding
Simple Correlation Analysis

Stimulus \xrightarrow{\text{representation}} \text{Neurons} \xrightarrow{\text{read-out}} \text{Behavior}

Tentative conclusion: Hierarchical representation does “untangling”
Cross-validation, sometimes called rotation estimation, is a model validation technique for assessing how the results of a statistical analysis will generalize to an independent data set.

Cross Validation


“Cross-validation, sometimes called rotation estimation, is a model validation technique for assessing how the results of a statistical analysis will generalize to an independent data set.”

Procedure:
1) estimate model params on training data
2) evaluate performance on testing data
Cross-validation, sometimes called rotation estimation, is a model validation technique for assessing how the results of a statistical analysis will generalize to an independent data set.


Procedure:
1) estimate model params on training data
2) evaluate performance on testing data

Q: Why are we doing this?
Cross Validation


“Cross-validation, sometimes called rotation estimation, is a model validation technique for assessing how the results of a statistical analysis will generalize to an independent data set.”

Procedure:
1) estimate model params on training data
2) evaluate performance on testing data

Q: Why are we doing this?
A: To prevent overfitting (fooling ourselves)
Cross Validation


“Cross-validation, sometimes called rotation estimation, is a model validation technique for assessing how the results of a statistical analysis will generalize to an independent data set.”

Procedure:
1) estimate model params on training data
2) evaluate performance on testing data

Q: Why are we doing this?
A: To prevent overfitting (fooling ourselves)

Q: How do you get the train/test splits?
Q: How do you get the train/test splits?

1. Leave-\( p \)-out: train on all-but-\( p \), test on \( p \) for all subsets of \( p \) stimuli; average results
Q: How do you get the train/test splits?

1. Leave-$p$-out: train on all-but-$p$, test on $p$ for all subsets of $p$ stimuli; average results

2. $k$-fold: partition data into $k$ equal-sized subsets; train on all but on subset; test on held-out slice; average results
Q: How do you get the train/test splits?

1. Leave-$p$-out: train on all-but-$p$, test on $p$ for all subsets of $p$ stimuli; average results

2. $k$-fold: partition data into $k$ equal-sized subsets; train on all but on subset; test on held-out slice; average results

3. random subsampling: randomly choose training subset and non-overlapping testing subset; average results over many random choices
Q: How do you get the train/test splits?

1. Leave-$p$-out: train on all-but-$p$, test on $p$ for all subsets of $p$ stimuli; average results

2. $k$-fold: partition data into $k$ equal-sized subsets; train on all but on subset; test on held-out slice; average results

3. random subsampling: randomly choose training subset and non-overlapping testing subset; average results over many random choices

key thing in all cases: training/test CANNOT overlap!
Procedure:

```python
splits = get_splits(split arguments)
```

For each split:

- estimate model params on training data
- evaluate performance on testing data
- average results over splits
Cross Validation

[IPYNB:  Getting Splits for Cross-Validation]
The Scikit-Learn project.
The Scikit-Learn project.

The `sklearn.linear_model` module implements generalized linear models. It includes Ridge regression, Lasso and Elastic Net estimators computed with Least Angle Regression and coordinate descent. It also implements Stochastic Gradient Descent related algorithms.

User guide: See the Generalized Linear Models section for further details.

<table>
<thead>
<tr>
<th>Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>linear_model.ARDRidge</code></td>
<td>Bayesian ARD regression.</td>
</tr>
<tr>
<td><code>linear_model.BayesianRidge</code></td>
<td>Bayesian ridge regression.</td>
</tr>
<tr>
<td><code>linear_model.ElasticNet</code></td>
<td>Linear regression with combined L1 and L2 priors as regularizer.</td>
</tr>
<tr>
<td><code>linear_model.ElasticNetCV</code></td>
<td>Elastic Net model with iterative fitting along a regularization path.</td>
</tr>
<tr>
<td><code>linear_model.HuberRegressor</code></td>
<td>Linear regression model that is robust to outliers.</td>
</tr>
<tr>
<td><code>linear_model.Lars</code></td>
<td>Least Angle Regression model a.k.a.</td>
</tr>
<tr>
<td><code>linear_model.LarsCV</code></td>
<td>Cross-validated Least Angle Regression model</td>
</tr>
<tr>
<td><code>linear_model.Lasso</code></td>
<td>Linear Model trained with L1 prior as regularizer (aka the Lasso).</td>
</tr>
<tr>
<td><code>linear_model.LassoCV</code></td>
<td>Lasso linear model with iterative fitting along a regularization path.</td>
</tr>
<tr>
<td><code>linear_model.LassoLars</code></td>
<td>Lasso model fit with Least Angle Regression a.k.a.</td>
</tr>
<tr>
<td><code>linear_model.LassoLarsIC</code></td>
<td>Cross-validated Lasso, using the LARS algorithm</td>
</tr>
<tr>
<td><code>linear_model.LinearRegression</code></td>
<td>Ordinary least squares Linear Regression.</td>
</tr>
<tr>
<td><code>linear_model.LogisticRegression</code></td>
<td>Logistic Regression (aka logit, MaxEnt) classifier.</td>
</tr>
<tr>
<td><code>linear_model.LogisticRegressionCV</code></td>
<td>Logistic Regression CV (aka logit, MaxEnt) classifier.</td>
</tr>
<tr>
<td><code>linear_model.MultiTaskLasso</code></td>
<td>Multi-task Lasso model trained with L1/L2 mixed-norm as regularizer.</td>
</tr>
<tr>
<td><code>linear_model.MultiTaskLassoCV</code></td>
<td>Multi-task L1/L2 Lasso with built-in cross-validation.</td>
</tr>
<tr>
<td><code>linear_model.OrthogonalMatchingPursuit</code></td>
<td>Orthogonal Matching Pursuit model (OMP)</td>
</tr>
<tr>
<td><code>linear_model.OrthogonalMatchingPursuitCV</code></td>
<td>Cross-validated Orthogonal Matching Pursuit model (OMP)</td>
</tr>
<tr>
<td><code>linear_model.PassiveAggressiveClassifier</code></td>
<td>Passive Aggressive Classifier</td>
</tr>
<tr>
<td><code>linear_model.PassiveAggressiveRegressor</code></td>
<td>Passive Aggressive Regressor</td>
</tr>
<tr>
<td><code>linear_model.Perceptron</code></td>
<td>Read more in the User Guide.</td>
</tr>
<tr>
<td><code>linear_model.RANSACRegressor</code></td>
<td>RANSAC (RAndom SAmple Consensus) algorithm.</td>
</tr>
<tr>
<td><code>linear_model.Ridge</code></td>
<td>Linear least squares with I2 regularization.</td>
</tr>
<tr>
<td><code>linear_model.RidgeClassifier</code></td>
<td>Classifier using Ridge regression.</td>
</tr>
<tr>
<td><code>linear_model.RidgeClassifierCV</code></td>
<td>Ridge classifier with built-in cross-validation.</td>
</tr>
<tr>
<td><code>linear_model.RidgeCV</code></td>
<td>Ridge regression with built-in cross-validation.</td>
</tr>
<tr>
<td><code>linear_model.SGDClassifier</code></td>
<td>Linear classifiers (SVM, logistic regression, a.o.) with SGD training.</td>
</tr>
<tr>
<td><code>linear_model.SGDRegressor</code></td>
<td>Linear model fitted by minimizing a regularized empirical loss with SGD.</td>
</tr>
<tr>
<td><code>linear_model.enet_path</code></td>
<td>Compute elastic net path with coordinate descent</td>
</tr>
<tr>
<td><code>linear_model.lars_path</code></td>
<td>Compute Least Angle Regression or Lasso path using LARS algorithm.</td>
</tr>
</tbody>
</table>
The **Scikit-Learn** setup.

class ModelType(object):

def __init__(self, some_arguments):

def fit(self, train_data, labels):

actually does the parameter estimation
doesn't return anything

def decision_function(self, test_data):

returns "confidence" (continuous value)
for each class for each test data point

shape of return:
num_samples x num_categories

def predict(self, test_data):

returns actual discrete predictions

shape of return:
um_samples
class ModelType(object):

    def __init__(self, some_arguments):

        def decision_function(self, test_data):
            returns "confidence" (continuous value)
            for each class for each test data point
            shape of return:
            num_samples x num_categories

        def predict(self, test_data):
            returns actual discrete predictions
            shape of return:
            num_samples
The **Scikit-Learn** setup.

class ModelType(object):
    
    def __init__(self, some_arguments):

    def fit(self, train_data, labels):
        actually does the parameter estimation
doesn’t return anything

    def decision_function(self, test_data):
        returns "confidence" (continuous value)
        for each class for each test data point
        shape of return:
        num_samples x num_categories

    def predict(self, test_data):
        returns actual discrete predictions
        shape of return:
        num_samples
The **Scikit-Learn** setup.

class ModelType(object):

    def __init__(self, some_arguments):

    def fit(self, train_data, labels):
        actually does the parameter estimation
doesn't return anything

    def decision_function(self, test_data):
        returns "confidence" (continuous value)
for each class for each test data point
shape of return:
    num_samples x num_categories

v
The **Scikit-Learn** setup.

class ModelType(object):

    def __init__(self, some_arguments):

    def fit(self, train_data, labels):
        actually does the parameter estimation
        doesn’t return anything

    def decision_function(self, test_data):
        returns “confidence” (continuous value)
        for each class for each test data point
        shape of return:
        num_samples x num_categories

    def predict(self, test_data):
        returns actual discrete predictions
        shape of return:
        num_samples
Procedure: (using the *Scikit-Learn* setup.)

splits = get_splits(split arguments)
Procedure: (using the **Scikit-Learn** setup.)

splits = get_splits(split arguments)

For each split:
Procedure: (using the Scikit-Learn setup.)

splits = get_splits(split arguments)

For each split:

model = ModelType(args)
model.fit(train_data, train_labels)
result = compare(prediction, test_labels)

average results over splits

Cross Validation
Procedure: (using the **Scikit-Learn** setup.)

```
splits = get_splits(split arguments)

For each split:

   model = ModelType(args)
   model.fit(train_data, train_labels)

   prediction = model.predict(test_data)
   result = compare(prediction, test_labels)
```
Procedure: (using the **Scikit-Learn** setup.)

```python
splits = get_splits(split arguments)

For each split:

    model = ModelType(args)
    model.fit(train_data, train_labels)
    prediction = model.predict(test_data)
    result = compare(prediction, test_labels)

average results over splits
```
Procedure: (using the Scikit-Learn setup.)

splits = get_splits(split arguments)

For each split:

model = ModelType(args)
model.fit(train_data, train_labels)

prediction = model.predict(test_data)
result = compare(prediction, test_labels)

average results over splits
Cross Validation

[IPYNB: The Train-Test Procedure]
Cross Validation

OVERFITTING
[IPYNB: Regularization]
(1) training and testing are both poor at very high levels of regularization (low C)

(2) training performance increases monotonically as regularization decreases, converging to ceiling levels at very high values of C

(3) overfitting (gap between green and blue) increases as regularization decreases

(4) **most important**: there is an *optimal* level of regularization for test performance
...of course when you actually do this you **must** cross validate your choice of C, since it is a free parameter of the model.

[IPYNB: Cross-Validating your regularization parameter]
Reliability

Neuron 10

Responses for neuron 10 Variation V0 images

Responses for neuron 10 Variation V3 images

Responses for neuron 10 Variation V6 images

1 std error due to trial noise

Neuron 105

Responses for neuron 105 Variation V0 images

Responses for neuron 105 Variation V3 images

Responses for neuron 105 Variation V6 images
[IPYNB: Reliability]
num_splits $\sim 10 \times$ num_trials is good enough to get a reasonable estimate for reliability:

\[ \text{error bars} = \text{SEM of reliability estimate} \]
Interestingly, reliability patterns are different for different neurons:
The Prophecy Formula

for normally distributed independent measure, correlation between averages of two length-\(k\) samples is:

\[
\rho_k = \frac{\sigma_T^2}{\sigma_T^2 + a^2/k}
\]

\(\sigma_T\) = true variability of data

\(a\) = error std for 1 sample

\(\rho_k = \text{corr}(X_k/k, X_k'/k)\)
The Prophecy Formula

\[ \rho_k = \frac{\sigma_T^2}{\sigma_T^2 + a^2/k} \]

\[ = \frac{k\sigma_T^2}{k\sigma_T^2 + a^2} \]

\[ = \frac{k\sigma_T^2}{k\sigma_T^2 - \sigma_T^2 + (\sigma_T^2 + a^2)} \]

\[ = \frac{k\sigma_T^2}{(k - 1)\sigma_T^2 + (\sigma_T^2 + a^2)} \]

\[ = \frac{k\sigma_T^2}{\sigma_T^2 + a^2} \]

\[ = \frac{k\sigma_T^2}{(k - 1)\sigma_T^2 + a^2 + 1} \]

\[ = \frac{k\rho_1}{(k - 1)\rho_1 + 1} \]
The Prophecy Formula

\[
spearman-brown(\rho, k) = \frac{k \cdot \rho}{1 + (k - 1)\rho}
\]

\(\rho\) = correlation for some number of trials

\(k\) = multiple of original number of trials for which you want to estimate correlation
Reliability

Spearman-brown estimates are pretty good . . .

. . . so neural noise (in these whitened signals) must be pretty gaussian
In IT neural sample, corrected reliabilities at ~50 trials: