CS375 / Psych 249:
Large-Scale Neural Network Models for Neuroscience

Lecture 5: Technical Tools II (Tensorflow, Optimization, and TFUtils)
2018.10.08

Daniel Yamins
Stanford Neurosciences Institute
Stanford Artificial Intelligence Laboratory
Departments of Psychology and Computer Science
Stanford University
NB: This lecture should be read in conjunction with the tutorial at:

Tensorflow is a graph computation language.
\[ e = (a + b) \times (b + 1) \]
Graph Model of Computation

e = (a+b)(b+1)

decomposes into 5 “nodes”:

c = a+b

d = b+1

e = c*d

It’s a Directed Acyclic Graph (DAG)
e = (a + b) \cdot (b + 1)

decomposes into 5 “nodes”:

\begin{align*}
  c &= a + b \\
  d &= b + 1 \\
  e &= c \cdot d
\end{align*}

Ok let’s see what happens if a = 2 and b = 1.
\[ e = (a+b) \cdot (b+1) \]

decomposes into 5 “nodes”:

- \( c = a + b \)
- \( d = b + 1 \)
- \( e = c \cdot d \)

Ok let’s see what happens if \( a = 2 \) and \( b = 1 \).
So what’s the tensors in the data here?

[IPYNB: constants]
Tensor Constants

\[ \text{one} = \text{tf.constant}(1) \]
one = \texttt{tf.constant(1)}

two = one + one
Tensor Constants

one = tf.constant(1)

two = one + one

+ matrices and tensors with shapes just like in NumPy.
So what’s the tensors in the data here?

[IPYNB: operations]
x = tf.range(-10, 10, .1)

y = tf.sin(x)

plt.plot(sess.run(x), sess.run(y))
x = tf.range(-10, 10, .1)
y = tf.sin(x)
z = y**2 + 10
w = tf.log(z)

plt.plot(sess.run(x), sess.run(w))
Tensor Operations

Element-wise multiplication:

\[ m1 \times m2 \quad m1.shape == m2.shape \]
(NumPy broadcasting rules apply)

Matrix multiplication:

\[ \text{tf.matmul}(m1, m2) \]
\[ m1.shape[1] == m2.shape[0] \]
Convolution:

\[
\text{blurred} = \text{tf.nn.conv2d}(\text{imtensor}, \text{filterarray}, \text{strides}=[1, 1, 1, 1], \text{padding}='\text{VALID}')
\]
So what’s the tensors in the data here?

[IPYNB: tensor variables]
eks = tf.get_variable('x',
                     shape=(),
                     dtype=tf.float32)

why = eks**2

sess.run(why, feed_dict={eks: 8})

> 64.0
So what’s the tensors in the data here?

[IPYNB: constructing hinge loss]
Tensor Variables

Having constructed variables

```
neural_data.shape -> (batch_size, num_neurons)
category_labels.shape -> (batch_size, num_categories)
weights.shape -> (num_neurons, num_categories)
bias.shape -> (num_categories,)
```

We then make hinge loss like so:

```
margins = tf.matmul(neural_data, weights) + bias
hinge_loss = tf.maximum(0., 1. - category_labels * margins)
```
\[ \frac{\partial}{\partial a} (a + b) = \frac{\partial a}{\partial a} + \frac{\partial b}{\partial a} \]

\[ \frac{\partial}{\partial u} u \cdot v = u \cdot \frac{\partial v}{\partial u} + \frac{\partial u}{\partial u} \cdot v = v \]

\( a \) changes at rate 1

\( \Rightarrow c \) changes at rate 1
Graph Model of Computation

\[
\frac{\partial}{\partial a} (a + b) = \frac{\partial a}{\partial a} + \frac{\partial b}{\partial a}
\]

\[
\frac{\partial}{\partial u} u \cdot v = u \cdot \frac{\partial v}{\partial u} + \frac{\partial u}{\partial u} \cdot v = v
\]

a changes at rate 1
=> c changes at rate 1

c changes at rate 1
=> e changes at rate 2

\[
\frac{\partial e}{\partial a} = 1 \cdot 2 = 2
\]
Graph Model of Computation

\[ \frac{\partial}{\partial a} (a + b) = \frac{\partial a}{\partial a} + \frac{\partial b}{\partial a} \]

\[ \frac{\partial}{\partial u} u \cdot v = u \cdot \frac{\partial v}{\partial u} + \frac{\partial u}{\partial u} \cdot v = v \]

a changes at rate 1

=> c changes at rate 1

c changes at rate 1

=> e changes at rate 2

\[ \frac{\partial e}{\partial a} = 1 \cdot 2 = 2 \]

effect of n1 on n2: sum over all paths between n1 and n2, multiply across edges

\[ \frac{\partial e}{\partial b} = 1 \cdot 2 + 1 \cdot 3 = 6 \]
How to prevent combinatorial explosion?
Graph Model of Computation

How to prevent combinatorial explosion? **Factor and merge.**

\[
\frac{\partial Z}{\partial X} = \alpha \delta + \alpha \epsilon + \alpha \zeta + \beta \delta + \beta \epsilon + \beta \zeta + \gamma \delta + \gamma \epsilon + \gamma \zeta
\]
How to prevent combinatorial explosion? **Factor and merge.**

\[
\frac{\partial Z}{\partial X} = (\alpha + \beta + \gamma) \cdot (\delta + \epsilon + \zeta)
\]

Method 1: start at an input, move up; sum all paths as you go.
Graph Model of Computation

How to prevent combinatorial explosion? **Factor and merge.**

\[
\frac{\partial Z}{\partial X} = (\alpha + \beta + \gamma) \cdot (\delta + \epsilon + \zeta)
\]

Method 1: start at an input, move up; sum all paths as you go.
How to prevent combinatorial explosion? **Factor and merge.**

\[
\frac{\partial Z}{\partial X} = (\alpha + \beta + \gamma) \cdot (\delta + \epsilon + \zeta)
\]

Method 2: start at the output, move down; sum all paths as you go.
Graph Model of Computation

How to prevent combinatorial explosion? **Factor and merge.**

\[
\frac{\partial Z}{\partial X} = (\alpha + \beta + \gamma) \cdot (\delta + \epsilon + \zeta)
\]

Method 2: start at the output, move down; sum all paths as you go.

Reverse-Mode Differentiation \((\frac{\partial Z}{\partial \delta})\)
Graph Model of Computation

Goal: calculate \( \frac{\partial e}{\partial b} \)
Graph Model of Computation

Goal: calculate \( \frac{\partial e}{\partial b} \)

Forward differentiation
Graph Model of Computation

Goal: calculate $\frac{\partial e}{\partial b}$

Forward Differentiation

Reverse Differentiation
Graph Model of Computation

Goal: calculate $\frac{\partial e}{\partial b}$

Forward differentiation

Reverse Differentiation
Graph Model of Computation

Goal: calculate $\frac{\partial e}{\partial b}$

Get: $\frac{\partial e}{\partial b}$ and $\frac{\partial e}{\partial a}$

Forward differentiation

Reverse Differentiation

Useful!
Graph Model of Computation

Goal: calculate $\frac{\partial e}{\partial b}$

Get: $\frac{\partial e}{\partial b}$ and $\frac{\partial e}{\partial a}$

The utility of backpropagation.

Reverse Differentiation

useful!

overhead

Geoff Hinton

David Rumelhart
Graph Model of Computation

Goal: calculate $\frac{\partial e}{\partial b}$

Get: $\frac{\partial e}{\partial b}$ and $\frac{\partial e}{\partial a}$

The utility of Backpropagation.

Reverse Mode AD in 1970 MA thesis

Seppo Linnainmaa • 3rd
Independent Computer Software Professional
VTT • University of Helsinki
Helsinki Area, Finland • 72 pp

Useful!
tf.Graph

Class **Graph**

Defined in  [tf.Operation objects](https://www.tensorflow.org/api_docs/python/tf/Operation) and  [tf.Tensor objects](https://www.tensorflow.org/api_docs/python/tf/Tensor), which represent units of computation; and  [tf.Tensor objects](https://www.tensorflow.org/api_docs/python/tf/Tensor), which represent the units of data that flow between operations.
Graph Model of Computation

tf.gradients

```python
def gradients(
    ys,
    xs,
    grad_ys=None,
    name='gradients',
    colocate_gradients_with_ops=False,
    gate_gradients=False,
    aggregation_method=None
)
```

Defined in `tensorflow/python/ops/gradients_impl.py`.

See the guide: Training > Gradient Computation

Constructs symbolic partial derivatives of sum of `ys` w.r.t. `x in xs`
Graph Model of Computation
Graph Model of Computation

So what’s the tensors in the data here?

[IPYNB: gradients]
Derivatives

\[ y = x^{**2} \]

\[ d = \text{tf.gradients}(y, x) \]
Derivatives

\[ y = x^{**2} \]

\[ d = \text{tf.gradients}(y, x) \]
\[ y = 2x^3 - 5x^2 + 3x - 1 \]
Derivatives

\[ y = x^{**.5} \]
\[ z = \text{tf.log}(\text{tf.exp}(\text{tf.sin}(y) + \text{tf.cos}(2*y)) + 1) \]
\[ w = z / (y + \text{tf.cos}(x\_arr)) \]
Derivatives

\[ z = x^{**2} + y^{**2} \]
\[ z = x^{**2} + y^{**2} \times x \]
Derivatives

\[ z = tf.cosh((3 + tf.cos(x+2*y)**3)**.5 * tf.sin(y-x)) \]
Module: tf.data

Defined in tensorflow/data/__init__.py.

tf.data.Dataset  API for input pipelines.

See Importing Data for an overview.

Classes

- **class Dataset**: Represents a potentially large set of elements.
- **class FixedLengthRecordDataset**: A Dataset of fixed-length records from one or more binary files.
- **class Iterator**: Represents the state of iterating through a Dataset.
- **class TFRecordDataset**: A Dataset comprising records from one or more TFRecord files.
- **class TextLineDataset**: A Dataset comprising lines from one or more text files.
Creating an iterator

Once you have built a `Dataset` to represent your input data, the next step is to create an `Iterator` to access elements from that dataset. The `tf.data` API currently supports the following iterators, in increasing level of sophistication:

Reading input data

Consuming NumPy arrays

If all of your input data fit in memory, the simplest way to create a `Dataset` from them is to convert them to `tf.Tensor` objects and use `Dataset.from_tensor_slices()`.

Consuming TFRecord data

The `tf.data` API supports a variety of file formats so that you can process large datasets that do not fit in memory. For example, the TFRecord file format is a simple record-oriented binary format that many TensorFlow applications use for training data. The `tf.data.TFRecordDataset` class enables you to stream over the contents of one or more TFRecord files as part of an input pipeline.

Consuming text data  

Consuming CSV data
Dataset objects

Preprocessing data with `Dataset.map()`

Decoding image data and resizing it

When training a neural network on real-world image data, it is often necessary to convert images of different sizes to a common size, so that they may be batched into a fixed size.

Applying arbitrary Python logic with `tf.py_func()`

For performance reasons, we encourage you to use TensorFlow operations for preprocessing your data whenever possible. However, it is sometimes useful to call upon external Python libraries when parsing your input data. To do so, invoke the `tf.py_func()` operation in a `Dataset.map()` transformation.

Batching dataset elements
Dataset objects
Model constructor functions

output = model_func(input_data, args ...)

output is a graph node that can be run

you can put whatever kind of complex structure in there
output = model_func(input_data, args ...)

output is a graph node that can be run

you can put whatever kind of complex structure in there

[IPYNB: model constructor functions]
Model constructor functions

\[
\text{output} = \text{model_func}(\text{input_data}, \text{args} \ldots)
\]

output is a graph node that can be run

you can put whatever kind of complex structure in there

![Diagram of neural network structure]

high-level description of result of constructing \texttt{mnist} model
\[ \text{batch_loss} = \text{loss_function}(\text{output}, \text{labels}) \]
Loss functions

\[
\text{batch_loss} = \text{loss_function}(\text{output}, \text{labels})
\]

\[
\text{mean_loss} = \text{tf.reduce_mean}(\text{batch_loss})
\]
Dataset objects

[IPYNB: optimizers and loss functions]
\[
\text{opt} = \text{optimizer}(\text{loss}, \text{learning_rate}).\text{minimize}
\]
Optimizers

\[
\text{opt} = \text{optimizer}(\text{loss, learning_rate}).\text{minimize}
\]
IS THERE A REPRODUCIBILITY CRISIS?

- 7% Don’t know
- 52% Yes, a significant crisis
- 38% Yes, a slight crisis
- 3% No, there is no crisis

1,576 researchers surveyed

©nature
Data management: how to Store DNN Experiments?

IS THERE A REPRODUCIBILITY CRISIS?

Anecdote 1: It took my lab two person-months to figure out how replicate training of recent paper* to reported numbers. Numerous published tutorials fail to replicate.

Data management: how to Store DNN Experiments?

**IS THERE A REPRODUCIBILITY CRISIS?**

Anecdote 1: It took my lab two person-months to figure out how replicate training of recent paper* to reported numbers. Numerous published tutorials fail to replicate.

Anecdote 2: Facebook AI Research Labs (FAIR) had a team spend one month trying to replicate a recent DeepMind result.** It took so long because a whole bunch of critical training parameters were extremely nonstandard, nonrobust, and **nonreported**.


TFUtils package

Utilities for working with TensorFlow

Installation

```
pip install git+https://github.com/neuroailab/tfutils.git
```

Documentation

http://neuroailab.stanford.edu/tfutils/index.html

License

MIT

http://github.com/neuroailab/tfutils
TFUtils

http://neuroailab.stanford.edu/tfutils

TFUtils's documentation

TFUtils is a Python utility package designed for coordinating neural network experiments using Tensorflow.

- constructing, running and monitoring neural network models
- facilitating multi-model and multi-gpu training
- retrieving data from data sources
- interfacing with common databases

TFUtils is compatible with: Python 2.7.

Contents

Package Reference

- tfutils
  - tfutils.base
  - tfutils.data
  - tfutils.error
  - tfutils.model
  - tfutils.optimizer
  - tfutils.utils
tfutils/train.py
contains routines for network training and validation

tfutils/test.py
contains routines for network testing

tfutils/imagenet_data.py
contains routines for efficiently working with data from ImageNet

tfutils/model_tool.py
contains convenience functions for building common models
Concept of training trajectory

Each point is a model at different stage of training.
Concept of training trajectory

Each point is a model at different stage of training.
TFUtils

Concept of training trajectory

Each point is a model at different stage of training.

Purpose of **TFUtils**: make it easy to store data so that trajectory can be easily:

1. reproduced,
2. continued, and
3. branched
MongoDB 3.4
Your Database Evolved
[IPYNB: mongo database]
Documents: are essentially nested dictionaries of values

```python
In [1]: x = {'a': 1, 'b': 1}
```
Documents: are essentially nested dictionaries of values

```
In [1]: x = {'a': 1, 'b': 1}
```

Collections: are groups of documents over which queries can be run

```
In [1]: x = {'a': 1, 'b': 1}
In [2]: y = {'a': 1, 'b': 2}
In [3]: z = {'a': 2, 'b': 1}
In [4]: w = {'a': 2, 'b': {'b1': 10, 'b2': 3}}
```
Documents: are essentially nested dictionaries of values

```
In [1]: x = {'a': 1, 'b': 1}
```

Collections: are groups of documents over which queries can be run

```
In [1]: x = {'a': 1, 'b': 1}
In [2]: y = {'a': 1, 'b': 2}
In [3]: z = {'a': 2, 'b': 1}
In [4]: w = {'a': 2, 'b': {'b1': 10, 'b2': 3}}
```

Queries: “subdictionaries” that describe what to match on to subselect documents

```
In [5]: query = {'a': 1}
```
Documents: are essentially nested dictionaries of values

\[ \text{In [1]: } x = \{ 'a': 1, 'b': 1 \} \]

Collections: are groups of documents over which queries can be run

\[ \text{In [1]: } x = \{ 'a': 1, 'b': 1 \} \]
\[ \text{In [2]: } y = \{ 'a': 1, 'b': 2 \} \]
\[ \text{In [3]: } z = \{ 'a': 2, 'b': 1 \} \]
\[ \text{In [4]: } w = \{ 'a': 2, 'b': \{ 'b1': 10, 'b2': 3 \} \} \]

Queries: “subdictionaries” that describe what to match on to subselect documents

\[ \text{In [5]: } \text{query} = \{ 'a': 1 \} \]
MongoDB

**Documents:** are essentially nested dictionaries of values

```
In [1]: x = {'a': 1, 'b': 1}
```

**Collections:** are groups of documents over which queries can be run

```
In [1]: x = {'a': 1, 'b': 1}
In [2]: y = {'a': 1, 'b': 2}
In [3]: z = {'a': 2, 'b': 1}
In [4]: w = {'a': 2, 'b': {'b1': 10, 'b2': 3}}
```

**Queries:** “subdictionaries” that describe what to match on to subselect documents

```
In [5]: query = {'a': 1}
```
**Documents:** are essentially nested dictionaries of values

```
In [1]: x = {'a': 1, 'b': 1}
```

**Collections:** are groups of documents over which queries can be run

```
In [1]: x = {'a': 1, 'b': 1}
In [2]: y = {'a': 1, 'b': 2}
In [3]: z = {'a': 2, 'b': 1}
In [4]: w = {'a': 2, 'b': {'b1': 10, 'b2': 3}}
```

**Queries:** “subdictionaries” that describe what to match on to subselect documents

```
In [6]: query = {'b.b1': 10}
```
Mongo query language highly expressive:

\[ \land, \lor, \neg, \exists, \leq, \in, \leq, \geq, \text{type}, \text{regex}, \text{array ops}, \ldots \]

nesting "." operator

**Collections:** are groups of documents over which queries can be run

```
In [1]: x = {'a': 1, 'b': 1}
In [2]: y = {'a': 1, 'b': 2}
In [3]: z = {'a': 2, 'b': 1}
In [4]: w = {'a': 2, 'b': {'b1': 10, 'b2': 3}}
```

**Queries:** “subdictionaries” that describe what to match on to subselect documents

```
In [6]: query = {'b.b1': 10}
```
MongoDB

Collections are what you can query *over*.

```python
In [1]: x = {'a': 1, 'b': 1}
In [2]: y = {'a': 1, 'b': 2}
In [3]: z = {'a': 2, 'b': 1}
In [4]: w = {'a': 2, 'b': {'b1': 10, 'b2': 3}}
```
Experiments have *addresses* in the Database

i & ii. connection **host** and **port** (what machine the database is running on)

iii & iv. the **database** and **collection** names

v. the **experiment id** ("exp_id")

Points on trajectory of training of a given network are records with a common address:

\[(host, port, db\_name, coll\_name, exp\_id)\]

different points distinguished by **_id** values
[IPYNB: basic training]
tf_train.train_from_params(save_params=save_params,
model_params=model_params,
train_params=train_params,
loss_params=loss_params,
learning_rate_params=learning_rate_params,
optimizer_params=optimizer_params,
skip_check=True)
tf_train.train_from_params(save_params=save_params,
model_params=model_params,
train_params=train_params,
loss_params=loss_params,
learning_rate_params=learning_rate_params,
optimizer_params=optimizer_params,
skip_check=True)

save_params = {
    'host': 'localhost',
    'port': 29101,
    'dbname': 'cs375_demo_db',
    'collname': 'cs375_demo_coll',
    'exp_id': 'training0',
    'save_valid_freq': 20,
    'save_filters_freq': 200,
    'cache_filters_freq': 100}
tf_train.train_from_params(
    save_params=save_params,
    model_params=model_params,
    train_params=train_params,
    loss_params=loss_params,
    learning_rate_params=learning_rate_params,
    optimizer_params=optimizer_params,
    skip_check=True
)

save_params = {
    'host': 'localhost',
    'port': 29101,
    'dbname': 'cs375_demo_db',
    'collname': 'cs375_demo_coll',
    'exp_id': 'training0',
    'save_valid_freq': 20,
    'save_filters_freq': 200,
    'cache_filters_freq': 100
}

model_params = {'func': model_tool.mnist_tfutils}
tf_train.train_from_params(save_params=save_params,
    model_params=model_params,
    train_params=train_params,
    loss_params=loss_params,
    learning_rate_params=learning_rate_params,
    optimizer_params=optimizer_params,
    skip_check=True)

save_params = {'host': 'localhost',
    'port': 29101,
    'dbname': 'cs375_demo_db',
    'collname': 'cs375_demo_coll',
    'exp_id': 'training0',
    'save_valid_freq': 20,
    'save_filters_freq': 200,
    'cache_filters_freq': 100}

model_params = {'func': model_tool.mnist_tfutils}

train_params = {}
train_params['data_params'] = {'func': mnist_data.build_data,
    'batch_size': BATCH_SIZE,
    'group': 'train',
    'directory': '/mnt/data/yamins/mnist_data'}

train_params['num_steps'] = 500
tf_train.train_from_params(
    save_params=save_params,
    model_params=model_params,
    train_params=train_params,
    loss_params=loss_params,
    learning_rate_params=learning_rate_params,
    optimizer_params=optimizer_params,
    skip_check=True)

save_params = {
    'host': 'localhost',
    'port': 29101,
    'dbname': 'cs375_demo_db',
    'collname': 'cs375_demo_coll',
    'exp_id': 'training0',
    'save_valid_freq': 20,
    'save_filters_freq': 200,
    'cache_filters_freq': 100
}

model_params = {'func': model_tool.mnist_tfutils}

train_params = {}
train_params['data_params'] = {'func': mnist_data.build_data,
                                'batch_size': BATCH_SIZE,
                                'group': 'train',
                                'directory': '/mnt/data/yamins/mnist_data'}

train_params['num_steps'] = 500

loss_params = {'loss_func': tf.nn.sparse_softmax_cross_entropy_with_logits,
               'agg_func': tf.reduce_mean,
               'pred_targets': 'labels'}
```python
tf_train.train_from_params(
    save_params=save_params,
    model_params=model_params,
    train_params=train_params,
    loss_params=loss_params,
    learning_rate_params=learning_rate_params,
    optimizer_params=optimizer_params,
    skip_check=True)

save_params = {
    'host': 'localhost',
    'port': 29101,
    'dbname': 'cs375_demo_db',
    'collname': 'cs375_demo_coll',
    'exp_id': 'training0',
    'save_valid_freq': 20,
    'save_filters_freq': 200,
    'cache_filters_freq': 100
}

model_params = {'func': model_tool.mnist_tfutils}

train_params = {}
train_params['data_params'] = {
    'func': mnist_data.build_data,
    'batch_size': BATCH_SIZE,
    'group': 'train',
    'directory': '/mnt/data/yamins/mnist_data'
}
train_params['num_steps'] = 500

loss_params = {
    'loss_func': tf.nn.sparse_softmax_cross_entropy_with_logits,
    'agg_func': tf.reduce_mean,
    'pred_targets': 'labels'
}

optimizer_params = {
    'optimizer_class': tf.train.MomentumOptimizer,
    'momentum': 0.9
}

learning_rate_params = {
    'learning_rate': 0.05,
    'decay_steps': SIZE_OF_DATASET // BATCH_SIZE,
    'decay_rate': 0.95,
    'staircase': True
}
```
Running code with these specifications leads to a trajectory being saved in the database:

tf_train.train_from_params(save_params=save_params,
model_params=model_params,
train_params=train_params,
loss_params=loss_params,
learning_rate_params=learning_rate_params,
optimizer_params=optimizer_params,
skip_check=True)
Running code with these specifications leads to a trajectory being saved in the database:

```python
tf_train.train_from_params(save_params=save_params,
model_params=model_params,
train_params=train_params,
loss_params=loss_params,
learning_rate_params=learning_rate_params,
optimizer_params=optimizer_params,
skip_check=True)
```

It's running this kind of graph:
Running code with these specifications leads to a trajectory being saved in the database:

Running with more steps just extends training trajectory from where you left.
Running code with these specifications leads to a trajectory being saved in the database:

```python
tf_train.train_from_params(save_params=save_params,
model_params=model_params,
train_params=train_params,
loss_params=loss_params,
learning_rate_params=learning_rate_params,
optimizer_params=optimizer_params,
skip_check=True)
```

Running with `load_params` equal to old `save_params`, new `save_params`:
Running code with these specifications leads to a trajectory being saved in the database:

```
tf_train.train_from_params(save_params=save_params,
model_params=model_params,
train_params=train_params,
loss_params=loss_params,
learning_rate_params=learning_rate_params,
optimizer_params=optimizer_params,
skip_check=True)
```

Running with `load_params` equal to old `save_params`, new `save_params`:

**load_query** to use specific point on trajectory (otherwise last saved point by default)
You can also do online validation:
[IPYNB: online validation]
for each batch in testing dataset:
  batch outputs = target(test_batch,
                        model(test batch))
  increment = online_agg_func(batch_outputs)
  step_results.append(increment)

final_results = agg_func(step_results)
But let's say you want to test a trained model on some new evaluation task:

... that's what the purpose of `test_from_params` is
load_params = {'host': 'localhost',
               'port': 29101,
               'dbname': 'cs375_demo_db',
               'collname': 'cs375_demo_coll',
               'exp_id': 'training0'}

save_params = {'host': 'localhost',
               'port': 29101,
               'dbname': 'cs375_demo_db',
               'collname': 'cs375_demo_coll',
               'exp_id': 'testing0'}

model_params = {'func': model_tool.mnist_tfutils}

testing_params = {'test0': {'data_params': {'func': mnist_data.build_data,
                                           'batch_size': BATCH_SIZE,
                                           'group': 'test',
                                           'directory': '/mnt/data/yamins/mnist_data'},
                           'num_steps': 10,
                           'targets': {'func': get_predictions_and_labels},
                           'agg_func': aggregate_accuracy}
}

def get_predictions_and_labels(data, logits):
    labels = data['labels']
    predictions = tf.argmax(input=logits, axis=1)
    return {'labels': labels, 'predictions': predictions}

def aggregate_accuracy(batch_results):
    labels = np.concatenate(pluck(batch_results, 'labels'))
    predictions = np.concatenate(pluck(batch_results, 'predictions'))
    correct = float((labels == predictions).sum())
    return {'percent_correct': 100. * correct / len(labels)}
Running code with these specifications leads to a validation record being saved in the database:

```python
tf_test.test_from_params(save_params=save_params,
load_params=load_params,
model_params=model_params,
validation_params=testing_params,
skip_check=True)
```

In [42]:
```python
coll = connection['cs375_demo_db']['cs375_demo_coll.files']
query = {'exp_id': 'testing0'}
coll.find(query).count()
```

Out[42]: 2

In [49]:
```python
record = coll.find(query)[0]
record.keys()
```

Out[49]:
```
[u'validates',
 u'saved_filters',
 u'step',
 u'params',
 u'duration',
 u'exp_id',
 u'_id',
 u'validation_results']
```

In [50]:
```python
record['validation_results']
```

Out[50]:
```
{u'test0': {u'percent_correct': 94.6}}
```
The `validates` key in the validation record indicates which model the testing result is a validation of.
Relationship of these tools to encoding & decoding

Stimulus $\rightarrow$ Neurons $\rightarrow$ Behavior

- Visual representation
- Category
- Location
- Size
- Pose
- Depth relationships

very nonlinear*

*which is presumably why so much brainmeat needs to be devoted to it.

fairly linear

“behavioral linking functions”
The stuff produced in \texttt{train\_from\_params} is very nonlinear*

*which is presumably why so much brainmeat needs to be devoted to it.

fairly linear

“behavioral linking functions”
**Relationship of these tools to encoding & decoding**

The stuff produced in `train_from_params` is very nonlinear*

*which is presumably why so much brainmeat needs to be devoted to it.

The stuff produced in `test_from_params` in the `arg_func` is fairly linear

“behavioral linking functions”
The stuff produced in `train_from_params` is very nonlinear*, which is presumably why so much brainmeat needs to be devoted to it.

The stuff produced in `test_from_params` in the arg_func is fairly linear "behavioral linking functions."

~Developmental learning of a visual system

Reinforcement task when you “come into the lab”

Stimulus representation → Neurons → Behavior

visual representation

Location
Size
Pose
Depth relationships

*Developmental learning of a visual system

Reinforcement task when you “come into the lab”

Stimulus representation → Neurons → Behavior

visual representation

Location
Size
Pose
Depth relationships

* which is presumably why so much brainmeat needs to be devoted to it.

“behavioral linking functions.”