**NB:** This lecture should be read in conjunction with the tutorial at:

TensorFlow is a graph computation language.
e = (a + b) * (b + 1)
Graph Model of Computation

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

decomposes into 5 "nodes":

c = a + b

d = b + 1

e = c \times d

It's a Directed Acyclic Graph (DAG)
\[ e = (a+b) \times (b+1) \]

decomposes into 5 “nodes”:

\[ c = a + b \]
\[ d = b + 1 \]
\[ e = c \times d \]

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

decomposes into 5 “nodes”:

\[ c = a+b \]
\[ d = b+1 \]
\[ e = c \times d \]

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

[IPYNB: constants]
one = tf.constant(1)
one = tf.constant(1)

two = one + one
Tensor Constants

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

\[
\text{two} = \text{one} + \text{one}
\]

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

[IPYNB: operations]
Tensor Operations

```python
x = tf.range(-10, 10, .1)
y = tf.sin(x)

plt.plot(sess.run(x), sess.run(y))
```
Tensor Operations

```python
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:

\[ m_1 \times m_2 \quad m_1.\text{shape} == m_2.\text{shape} \]
(NumPy broadcasting rules apply)

Matrix multiplication:

\[ \text{tf.matmul}(m_1, m_2) \]
\[ m_1.\text{shape}[1] == m_2.\text{shape}[0] \]
Convolution:

```python
blurred = tf.nn.conv2d(imtensor, filterarray, strides=[1, 1, 1, 1], padding='VALID')
```
Graph Model of Computation

So what’s the tensors in the data here?

[IPYNB: tensor variables]
Tensor Variables

```python
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]
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

\[
\Rightarrow \ c \ changes \ at \ rate \ 1
\]
\[
\frac{\partial}{\partial a} (a + b) = \frac{\partial a}{\partial a} + \frac{\partial b}{\partial a}
\]

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

a changes at rate 1

\[\Rightarrow\text{c changes at rate 1}\]

c changes at rate 1

\[\Rightarrow\text{e changes at rate 2}\]

\[
\frac{\partial e}{\partial a} = 1 \cdot 2 = 2
\]
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
\implies c changes at rate 1

c changes at rate 1
\implies e changes at rate 2

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

effect of \( n_1 \) on \( n_2 \): sum over all paths between \( n_1 \) and \( n_2 \), multiply across edges

\[
\frac{\partial e}{\partial b} = 1 \cdot 2 + 1 \cdot 3 = 6
\]
Graph Model of Computation

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

How to prevent combinatorial explosion?
Graph Model of Computation

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

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

Goal

\( \frac{\partial e}{\partial b} = 5 \)

\( \frac{\partial c}{\partial b} = 1 \)

\( \frac{\partial b}{\partial b} = 1 \)

\( \frac{\partial a}{\partial b} = 0 \)

\( \frac{\partial d}{\partial b} = 1 \)

\( \frac{\partial e}{\partial e} = 1 \)

\( \frac{\partial e}{\partial d} = 3 \)

\( \frac{\partial e}{\partial a} = 2 \)

\( \frac{\partial c}{\partial c} = 2 \)

\( \frac{\partial c}{\partial d} = 3 \)

\( \frac{\partial d}{\partial c} = 1 \)

\( \frac{\partial d}{\partial d} = 1 \)

\( e = c \times d \quad e = 6 \)

\( c = a + b \quad c = 3 \)

\( d = b + 1 \quad d = 2 \)

\( a = 2 \)

\( b = 1 \)
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!  
overhead

goal

stupid

overhead

$e = c \ast d$
$e = 6$

$c = a + b$
$c = 3$

$\frac{\partial e}{\partial c} = 2$
$\frac{\partial e}{\partial d} = 3$

$\frac{\partial c}{\partial c} = 1$
$\frac{\partial c}{\partial a} = 1$

$\frac{\partial b}{\partial b} = 1$
$\frac{\partial d}{\partial b} = 1$

$\frac{\partial d}{\partial d} = 2$
$\frac{\partial d}{\partial c} = 1$

$\frac{\partial e}{\partial e} = 1$
$\frac{\partial e}{\partial c} = 2$
$\frac{\partial e}{\partial d} = 3$

$\frac{\partial e}{\partial a} = 2$
$\frac{\partial e}{\partial b} = 5$

$\frac{\partial e}{\partial b} = 0$

$\frac{\partial e}{\partial b} = 5$
$\frac{\partial e}{\partial b} = 3$

$\frac{\partial e}{\partial b} = 1$
$\frac{\partial e}{\partial b} = 1$

$\frac{\partial e}{\partial b} = 1$
$\frac{\partial e}{\partial b} = 1$

$\frac{\partial e}{\partial b} = 1$
$\frac{\partial e}{\partial b} = 1$
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

Geoff Hinton

David Rumelhart

useful!
Graph Model of Computation

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

Get: \( \frac{\partial e}{\partial b} \) and \( \frac{\partial e}{\partial a} \)

Reverse Mode AD in 1970 MA thesis

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

The utility of backpropagation.

Reverse Differentiation
Graph Model of Computation

tf.Graph

Class **Graph**

Defined in [tensorflow/python/framework/ops.py](https://www.tensorflow.org/api_docs/python/tf.Graph).

See the guide: Building Graphs > Core graph data structures

A TensorFlow computation, represented as a dataflow graph.

A `Graph` contains a set of `tf.Operation` objects, which represent units of computation; and `tf.Tensor` objects, which represent the units of data that flow between operations.
Graph Model of Computation

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

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

See the guide: \text{Training \textgreater{} Gradient Computation}

Constructs symbolic partial derivatives of sum of `\text{ys}` w.r.t. `\text{x in xs}`
Graph Model of Computation
So what’s the tensors in the data here?

[IPYNB: gradients]
Derivatives

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

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

![Graph showing the function and its derivative](image-url)
Derivatives

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

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

\[ y = 2x^3 - 5x^2 + 3x - 1 \]
Derivatives

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

\[ z = x^{**2} + y^{**2} \]
z = x**2 + y**2 * x
$z = \text{tf.cosh}\left((3 + \text{tf.cos}(x+2*y)^3)^{0.5} \times \text{tf.sin}(y-x)\right)$
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.
Dataset objects

https://www.tensorflow.org/guide/datasets

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
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

[IPYNB:  dataset objects]
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]
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

high-level description of result of constructing \texttt{mnist} model
loss functions

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

mean_loss = tf.reduce_mean(batch_loss)
Dataset objects

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

opt = optimizer(loss, learning_rate).minimize

= effect of calling `sess.run(opt)`, e.g. actually updating parameters
Data management: how to Store DNN Experiments?

**Is there a reproducibility crisis?**

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

1,576 researchers surveyed

©nature
Anecdote 1: It took my lab two person-months to figure out how to 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 to 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

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

tfutils/model_tool.py
contains convenience functions for building common models

tfutils/train.py
contains routines for network training and validation

tfutils/test.py
contains routines for network testing
Concept of training trajectory

each point is a model at different stage of training
Concept of training trajectory

TFUtils

each point is a model at different stage of training
Concept of training trajectory

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

1. reproduced,
2. continued, and
3. branched

each point is a model at different stage of training
MongoDB 3.4
Your Database Evolved

Download Enterprise 3.4
[IPYNB:  mongo database]
**Documents:** are essentially nested dictionaries of values

```
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

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

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

```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}}
```

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

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

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

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

```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}}
```

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

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

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

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}
```
Collections are what you can query *over*
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_tfutil
```python
import tensorflow as tf

def train_from_params(save_params, model_params, train_params, loss_params, learning_rate_params, optimizer_params, skip_check):
    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:

```
import tensorflow as tf

# Define the model
model = tf.keras.models.Sequential([  
    tf.keras.layers.InputLayer(input_shape=(28, 28, 1)),  
    tf.keras.layers.Flatten(),  
    tf.keras.layers.Dense(100, activation='relu'),  
    tf.keras.layers.Dense(32, activation='relu'),  
    tf.keras.layers.Dense(10, activation='softmax'),  
    tf.keras.layers.Dense(1, activation='sigmoid'),
])

# Define the loss function
loss = tf.keras.losses.BinaryCrossentropy(from_logits=True)

# Define the optimizer
optimizer = tf.keras.optimizers.Adam(learning_rate=0.001)

# Compile the model
model.compile(loss=loss, optimizer=optimizer, metrics=['accuracy'])

# Train the model
model.fit(x_train, y_train, epochs=10, batch_size=64)
```

It's running this kind of graph:
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 more steps just extends training trajectory from where you left: 

![Training after 500 steps](image1)

![Training after 1000 steps](image2)
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`:

![Diagram showing training loss over training steps before and after 500 steps.](image)
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`:

- **load_query** to use specific point on trajectory (otherwise last saved point by default)
You can also do online validation:
[IPYNB: online validation]
Fancy aggregation code pattern is (roughly):

```python
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)
```

It's running this kind of graph:
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
```python
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:

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

Out[42]: 2

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

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

In [50]:
record['validation_results']

Out[50]: {'test0': {'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 \[\xrightarrow{representation}\] Neurons \[\xrightarrow{read-out}\] Behavior

Category
Location
Size
Pose
Depth relationships

visual representation

very nonlinear*

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

fairly linear

“behavioral linking functions”

visual representation

visual representation

visual representation
The stuff produced in `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` 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` 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.

Reinforcement task when you “come into the lab”

Location
Size
Pose
Depth relationships

• Developmental learning of a visual system

~Developmental learning of a visual system

visual representation