Graph Model of Computation

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

c = a + b

d = b + 1

e = c \times d

Diagram:
- Node 'c = a + b'
- Node 'd = b + 1'
- Node 'e = c \times d'
- Edges: 'a' to 'c', 'b' to 'c', 'c' to 'e', 'd' to 'e'
e = (a + b) \times (b + 1)

c = a + b

d = b + 1

e = c \times d

a = 2 \text{ and } b = 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 \(1\)  
\(\Rightarrow c\) 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

=> 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.**

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

Diagram:
- $a = 2$
- $b = 1$
- $c = 3$
- $d = 2$
- $e = 6$

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

Equations:
- $e = c \times d$
- $e = 6$
- $c = a + b$
- $c = 3$
- $d = b + 1$
- $d = 2$
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!

overhead

overhead

stupid
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

overhead

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

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

Reverse Mode AD in 1970 MA thesis
tf.Graph

Class **Graph**

Defined in [tensorflow/python/framework/ops.py](tensorflow/python/framework/ops.py).

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

tf.gradients

```python
gradients(
y,  
x,  
grad_ys=None,  
name='gradients',  
colocate_gradients_with_ops=False,  
gate_gradients=False,  
aggregation_method=None
)
```

Defined in `tensorflow/python/ops/grads_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
Data management: how to Store DNN Experiments?

**IS THERE A REPRODUCIBILITY CRISIS?**

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

1,576 researchers surveyed

©nature
Anecdote 1: It took my lab one man month to figure out how replicate training of AlexNet to reported numbers. Numerous published tutorials fail to replicate.
Anecdote 1: It took my lab one man month to figure out how replicate training of AlexNet 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 because a whole bunch of critical training parameters were extremely nonstandard and nonrobust, and nonreported.
Data management: how to Store DNN Experiments?

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

Anecdote 1: It took my lab one man month to figure out how to replicate training of AlexNet to reported numbers. Numerous published tutorials fail to replicate.
Data management: how to Store DNN Experiments?

Problem: Store results of a training session so that:

- Trajectory of training can easily be queried

![Diagram showing trajectory of training](image-url)
Problem: Store results of a training session so that:

- Trajectory of training can easily be queried
Problem: Store results of a training session so that:

- Trajectory of training can easily be queried

- Evaluations of trained model on test datasets can easily be indexed back to training parameters
Problem: Store results of a training session so that:

- Trajectory of training can easily be queried
- Evaluations of trained model on test datasets can easily be indexed back to training parameters
- Training can be continued from any point:
  - and if from the end re-indexed into the saved location
Problem: Store results of a training session so that:

- Trajectory of training can easily be queried
- Evaluations of trained model on test datasets can easily be indexed back to training parameters
- Training can be continued from any point:
  - and if from the end re-indexed into the saved location
Problem: Store results of a training session so that:

- Trajectory of training can easily be queried
- Evaluations of trained model on test datasets can easily be indexed back to training parameters
- Training can be continued from any point:
  - and if from the end re-indexed into the saved location
  - if from an intermediate point, with some model structure changes, saved into a new location
Data management: how to Store DNN Experiments?
**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

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

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

nesting “.” operator

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

Queries: “subdictionaries” that describe what to match on to subselect documents
Collections are what you can query *over*.
Data management: how to Store DNN Experiments?

MongoDB documents are good* for recording science experiments:

- expressive of idea of nested groupings of parameters

```python
params['validation_params'] = {
    'topn_val': {
        'data_params': {
            # ImageNet data provider arguments
            'func': ImageNetDataProvider,
            'data_path': self.Config.data_path,
            'group': 'val',
            'crop_size': self.Config.crop_size,
            # TFRecords (super class) data provider arguments
            'file_pattern': 'validation*.tfrecords',
            'batch_size': self.Config.batch_size,
            'shuffle': False,
            'shuffle_seed': self.Config.seed,
            'file_read_func': self.subselect_tfrecords,
            'n_threads': 4,
        },
        'queue_params': {
            'queue_type': 'fifo',
            'batch_size': self.Config.batch_size,
            'seed': self.Config.seed,
            'capacity': self.Config.batch_size * 10,
            'min_after_dequeue': self.Config.batch_size * 5,
        },
        'targets': {
            'fun': self.in_top_k,
        },
        'num_steps': self.Config.val_steps,
        'agg_func': lambda x: {k: np.mean(v) for k, v in x.items()},
        'online_agg_func': self.online_agg,
    }
}
```

- easy to extend to new type without thinking about a schema

*other DB formats, like Postgres, might be even better
Data management: how to Store DNN Experiments?

MongoDB documents are good for recording science experiments:

- expressive of idea of nested groupings of parameters
- easy to extend to new type without thinking about a schema

Basic idea: store data from training and testing DNNs as mongo documents.

*other DB formats, like Postgres, might be even better*
Data management: how to Store DNN Experiments?

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:

\[(host, port, db\_name, coll\_name, exp\_id)\]
Data management: how to Store DNN Experiments?

Experiments have *addresses* in the Database

i & ii. connection **host** and **port** (what machine the database is running on)
Data management: how to Store DNN Experiments?

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

```python
In [4]: conn['erroropt-vgg'].collection_names()
Out[4]:
[u'alexnet_imnet_i2.files',
 u'system.indexes',
 u'alexnet_imnet_i2.chunks',
 u'alexnet_imnet_i2_method2.files',
 u'alexnet_imnet_i2_method2.chunks']
```
Data management: how to Store DNN Experiments?

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

```
In [4]: conn['erroropt-vgg'].collection_names()
Out[4]:
[ u'alexnet_imnet_i2.files',
  u'system.indexes',
  u'alexnet_imnet_i2chunks',
  u'alexnet_imnet_i2_method2.files',
  u'alexnet_imnet_i2_method2.chunks']
```

v. the experiment id ("exp_id")

```
In [5]: conn['erroropt-vgg']['alexnet_imnet_i2_method2.files'].distinct('exp_id')
Out[5]:
[ u'trainval1',
  u'trainval0',
  u'trainval2',
  u'trainval3',
  u'trainval4',
  u'trainval5',
  u'trainval6',
  u'trainval7']
```
Data management: how to Store DNN Experiments?

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”)

there are 109 training points in this trajectory

```python
In [7]: conn['erroropt-vgg']['alexnet_imnet_i2_method2.files'].find({'exp_id': 'trainval0'}).count()
Out[7]: 109
```

```
(host, port, db_name, coll_name, exp_id) = node1-neuroailabcluster, 27017, erroropt-vgg, alexnet_imnet_i2_method2, trainval0
```
Data management: how to Store DNN Experiments?

Get a record:

```python
In [8]: r = conn['erroropt-vgg']['alexnet_imnet_i2_method2.files'].find({'exp_id': 'trainval0'})[0]
```
Data management: how to Store DNN Experiments?

Get a record:

```python
In [8]: r = conn['erroropt-vgg']['alexnet_imnet_i2_method2.files'].find({'exp_id': 'trainval0'})[0]
```

What's in it?

```python
In [9]: r.keys()
Out[9]:
[u'_saver_write_version',
 u'chunkSize',
 u'filename',
 u'_saver_num_data_files',
 u'saved_filters',
 u'duration',
 u'length',
 u'params',
 u'uploadDate',
 u'step',
 u'exp_id',
 u'_id',
 u'validation_results',
 u'md5']
```
Data management: how to Store DNN Experiments?

Ok — lots of stuff:

```python
In [10]: r['params']
Out[10]:
{u'inter_op_parallelism_threads': 40,
 u'learning_rate_params': {u'decay_rate': 0.95,
 u'decay_steps': 16108,
 u'func': {u'modname': u'tensorflow.python.training.learning_rate_decay',
 u'objname': u'exponential_decay',
 u'source_path': u'/usr/local/lib/python2.7/dist-packages/tensorflow/python/training/learning_rate_decay.py',
 u'version': None},
 u'learning_rate': 0.0001,
 u'staircase': True},
 u'load_params': {},
 u'log_device_placement': False,
 u'loss_params': {u'agg_func': {u'modname': u'tensorflow.python.ops.math_ops',
 u'objname': u'reduce_mean',
 u'source_path': u'/usr/local/lib/python2.7/dist-packages/tensorflow/python/ops/math_ops.py',
 u'version': None},
 u'loss_func_kwargs': {u'BUFFER_SIZE': 10000,
 u'labels': [u'TypeError', u'TypeError', u'TypeError'],
 u'logits': u'TypeError'},
 u'loss_per_case_func': {u'active_branch': u'master',
 u'active_branch_in_origin': True,
 u'clean': False,
 u'commit': u'8863be5b87f939e6dec8ba0a25a3cd259419ca8c8',
 u'commit_in_log': True,
 u'git_dir': u'/mnt/fs0/eliwang/erroropt/.git',
 u'modname': u'pyfunc_buffer',
 u'objname': u'combined_method2_loss',
 u'remote_urls': [u'https://github.com/neurotablab/erroropt.git'],
 u'source_path': u'/mnt/fs0/eliwang/erroropt/pyfunc_buffer.py',
 u'targets': [u'train_labels', u'val_labels', u'val_dprime_labels'],
 u'model_params': {u'cfg_final': {u'conv1': {u'conv': {u'activation': u'relu',
 u'bias': 0,
 u'init': u'xavier',
 u'input': u'random_shuffle_queue_DequeueMany:2',
 u'kernel_size': [11, 11]},
}
Data management: how to Store DNN Experiments?

Ok — lots of stuff:

```
In [11]: r['params'].keys()
Out[11]:
[u'learning_rate_params',
 'inter_op_parallelism_threads',
 'optimizer_params',
 'load_params',
 'save_params',
 'model_params',
 'validation_params',
 'loss_params',
 'train_params',
 'log_device_placement']
```
Ok — lots of stuff, including info about **how the records are being saved**:
Data management: how to Store DNN Experiments?

Ok — lots of stuff, including info about the configuration of the model:

```
In [17]: r['params']['model_params']['cfg_final'].keys()
Out[17]: ['u'fc6', 'u'fc7', 'u'fc8', 'u'conv3', 'u'conv2', 'u'conv1', 'u'conv5', 'u'conv4']

In [18]: r['params']['model_params']['cfg_final']['conv3']
Out[18]:
{u'conv': {u'activation': u'relu',
           u'bias': 0,
           u'init': u'xavier',
           u'input': u'conv2/pool:0',
           u'kernel_size': [3, 3],
           u'num_filters': 384,
           u'padding': u'SAME',
           u'seed': 0,
           u'stddev': 0.01,
           u'stride': 1,
           u'type': u'conv',
           u'weight_decay': 0.0}}
```
Data management: how to Store DNN Experiments?

Ok — lots of stuff, including info about *how* the model is trained:
Data management: how to Store DNN Experiments?

Ok — lots of stuff, including info about performance at each record:

```
In [25]: r['validation_results']['topn_val']['l2']
Out[25]: 23.55299181219859
```
Data management: how to Store DNN Experiments?

Let’s put this together

Zero in on a collection:

```
In [38]: dbname = 'erroropt-vgg'
In [39]: collname = 'alexnet_imnet_i2_method2.files'
```
Data management: how to Store DNN Experiments?

Let’s put this together

Zero in on a collection:

What experiments are here?
Let’s put this together

Zero in on a collection:

What experiments are here?

Let’s focus on the results of experiment “trainval6” in particular, records where validation results exist.
Data management: how to Store DNN Experiments?

Let’s put this together

Zero in on a collection:

What experiments are here?

Let’s focus on the results of experiment “trainval6”

in particular, records where validation results exist
Data management: how to Store DNN Experiments?

Let’s focus on the results of experiment “trainval6”

```
In [43]: query = {'exp_id': 'trainval6', 'validation_results': {'$exists': True}}
```
Data management: how to Store DNN Experiments?

Let’s focus on the results of experiment “trainval6”

```python
In [43]: query = {'exp_id': 'trainval6', 'validation_results': {'$exists': True}}
```

Query all the results sorted by “step”:

```python
In [51]: results = [r['validation_results']['topn_val']['l2'] for r in conn[dbname][collname].find(query).sort('step')]
In [52]: results
Out[52]:
[2.0659278026389547, 1.0428932198868655, 0.7319157273430057, 0.5910100716772089, 0.4542647562713749, 0.41023308641695366, 0.3697211358907601, 0.3590482558766105, 0.32728324129128206]
```
Data management: how to Store DNN Experiments?

Let’s focus on the results of experiment “trainval6”

```python
In [43]: query = {'exp_id': 'trainval6', 'validation_results': {'$exists': True}}
```

Query all the results

```python
In [51]: results = [r['validation_results']['topk'] for r in collection.find(query)]
In [52]: results
Out[52]:
[2.0659278026389547,
  1.0428932198868655,
  0.7319157273430057,
  0.591010716772089,
  0.4542647562713749,
  0.41023308641695366,
  0.369721358907601,
  0.3590482558766105,
  0.32728324129128206]
```

```python
In [9]: plt.plot(results)
Out[9]: [<matplotlib.lines.Line2D at 0x10a0ab410>]
```
Calling `train_from_params` is what puts records in the database to begin with.
TFUtils: train_from_params

```python
def train_from_params(save_params, model_params, train_params, loss_params=None, learning_rate_params=None, optimizer_params=None, validation_params=None, log_device_placement=False, load_params=None, dont_run=False, skip_check=False, inter_op_parallelism_threads=40,):

    ...

    Main training interface function.
```
def train_from_params(save_params,
    model_params,
    train_params,
    loss_params=None,
    learning_rate_params=None,
    optimizer_params=None,
    validation_params=None,
    log_device_placement=False,
    load_params=None,
    dont_run=False,
    skip_check=False,
    inter_op_parallelism_threads=40,
):  

......

Main training interface function.

def alexnet_model(inputs, train=True, norm=True, **kwargs):

......

AlexNet model definition as defined in the paper:
https://papers.nips.cc/paper/4824-imagenet-classification-
def train_from_params(save_params, model_params, train_params, loss_params=None, learning_rate_params=None, optimizer_params=None, validation_params=None, log_device_placement=False, load_params=None, dont_run=False, skip_check=False, inter_op_parallelism_threads=40,):

......

Main training interface function.

params['train_params'] = {
    'data_params': {
        # ImageNet data provider arguments
        'func': ImageNetDataProvider,
        'data_path': self.Config.data_path,
        'group': 'train',
        'crop_size': self.Config.crop_size,
        # TFRecords (super class) data provider
        'file_pattern': 'train*.tfrecords',
        'batch_size': self.Config.batch_size,
        'shuffle': False,
        'shuffle_seed': self.Config.seed,
        'file_read_func': self.subselect_tfr,
        'n_threads': 4,
    },
    ...
    ...
    ...
    
    'targets': {
        'func': self.return_outputs,
        'targets': [],
    },
    'num_steps': self.Config.train_steps,
    'thres_loss': self.Config.thres_loss,
    'validate_first': False,
}
TFUtils: train_from_params

```python
def train_from_params(save_params, model_params, train_params, loss_params=None, learning_rate_params=None, optimizer_params=None, validation_params=None, log_device_placement=False, load_params=None, dont_run=False, skip_check=False, inter_op_parallelism_threads=40,):

    params['loss_params'] = {
        'targets': ['labels'],
        'agg_func': tf.reduce_mean,
        'loss_per_case_func': EDIT_YOUR_LOSS_PER_CASE_FUNC_HERE,
        'loss_per_case_func_params': {_outputs: 'outputs',
                                     '_targets_all': 'inputs'},
        'loss_func_kwargs': {};
    }
```

Interface function.
```python
params['learning_rate_params'] = {
    'func': tf.train.exponential_decay,
    'learning_rate': 0.01,
    'decay_steps': ImageNetDataProvider.N_TRAIN /
    'decay_rate': 0.95,
    'staircase': True,
}
```

Interface function.
TFUtils: train_from_params

```python
params['optimizer_params'] = {
    'func': optimizer.ClipOptimizer,
    'optimizer_class': tf.train.AdamOptimizer,
    'clip': False,
    'momentum': .9,
}
```
Running code with these specifications leads to a trajectory being saved in the database:
Running code with these specifications leads to a trajectory being saved in the database:

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

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

Running with `load_params` equal to old `save_params`, `new save_params`, and partially overlapping model:
TFUtils: train_from_params

But let's say you want to test:

... that's what the purpose of test_from_params is
def test_from_params(load_params,
    model_params,
    validation_params,
    log_device_placement=False,
    save_params=None,
    dont_run=False,
    skip_check=False,
    inter_op_parallelism_threads=40,
):
    
    ....

    params['load_params'] = {
        'host': 'localhost',
        'port': 24444,
        'dbname': 'imagenet',
        'collname': 'alexnet',
        'exp_id': self.Config.exp_id,
        'do_restore': True,
        'query': {'step': self.Config.extraction_step} \ 
            if self.Config.extraction_step is not None else None,
    }
TFUtils: test_from_params

def test_from_params(load_params,
    model_params,
    validation_params,
    log_device_placement=False,
    save_params=None,
    dont_run=False,
    skip_check=False,
    inter_op_parallelism_threads=40,
):

    ...
params
    load_params,
    model_params,
    validation_params,
    log_device_placement=False,
    save_params=None,
    dont_run=False,
    skip_check=False,
    inter_op_parallelism_threads=40,
    ):

    params[validation_params] = {
        'valid0': {
            'data_params': {
                # ImageNet data provider arguments
                func': NeuralDataProvider,
                'data_path': self.Config.data_path,
                'crop_size': self.Config.crop_size,
                # TFRecords (super class) data provider arguments
                'file_pattern': '*.tfrecords',
                'batch_size': self.Config.batch_size,
                'shuffle': False,
                'shuffle_seed': self.Config.seed,
                'n_threads': 1,
            },
        },
    },
TFUtils: `test_from_params`

```
'targets': {
    'func': self.return_outputs,
    'targets': self.Config.extraction_targets,
},

'num_steps': self.Config.val_steps,
'agg_func': self.neural_analysis,
'online_agg_func': self.online_agg,
```

Describes what you want to get out of the saved model, once it’s run on your validation data.
Describes what you want to get out of the saved model, once it's run on your validation data.

For each batch in testing:

```python
    batch_outputs = run_model(test batch)
    increment = online_agg_func(batch_outputs)
    step_results.append(increment)

    final_test_results = agg_func(step_results)
```
Running code with these specifications leads to *testing results* being saved in the database:

```python
def test_from_params(load_params,
    model_params,
    validation_params,
    log_device_placement=False,
    save_params=None,
    dont_run=False,
    skip_check=False,
    inter_op_parallelism_threads=40,
    ):

    .....
'num_steps': self.Config.val_steps,
'agg_func': self.neural_analysis,
'online_agg_func': self.online_agg,

```python
def neural_analysis(self, results):
    """
    Performs an analysis of the results from the model on the neural data.
    This analysis includes:
    - saving the conv1 kernels
    - computing a RDM
    - a categorization test
    - and an IT regression.
    ```
In the example code, workhorse is `dldata.metrics.utils.compute_metric_base`
TFUtils: test_from_params

dldata.metrics.utils.compute_metric_base is a unified interface to many different simple machine learning constructs, many of which come from scikit-learn.

```python
def compute_metric_base(F, meta, eval_config, return_splits=False,
                        attach_models=False, dataset=None,
                        attach_predictions=True,
                        knockouts=None, compute_from=None):
```

\[ F = \text{ feature matrix} --- \text{N stimuli} \times \text{M features} --- \text{can come from model, neurons, fMRI, whatever} \]
TFUtils: test_from_params

`dldata.metrics.utils.compute_metric_base` is a unified interface to many different simple machine learning constructs, many of which come from scikit-learn.

```python
def compute_metric_base(F, meta, eval_config, return_splits=False,
                        attach_models=False, dataset=None,
                        attach_predictions=True,
                        knockouts=None, compute_from=None):
```

\[ F = \text{feature matrix} \text{ — } N \text{ stimuli} \times M \text{ features} \text{ — can come from model, neurons, fMRI, whatever} \]

\[ \text{meta} = \text{numpy.rec.array} \text{ data object of length describing metadata for each stimuli, should be thought of as like a “spreadsheet” with one column per label type} \]
Multi-array Electrophysiology Experiment

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
Beyond categorization

plane  Category

f16  Identity
Beyond categorization

Position
Beyond categorization

Size
Beyond categorization

Aspect Ratio and Angle
Beyond categorization

We can quickly assess the scene as a whole.
meta = `numpy.rec.array` data object of length describing metadata for each stimuli, should be thought of as like a “spreadsheet” with one column per label type

For the Neural Representation Benchmark set (5760 images) —>

Column (field) names:

Accessing subsets of records and columns:
meta = numpy.rec.array data object of length describing metadata for each stimuli, should be thought of as like a “spreadsheet” with one column per label type

Querying subsets: e.g. all low variation-stimuli (“var” level == V0) — there are 640

```python
In [19]: meta['var'] == 'V0'
Out[19]: array([ True, True, True, ..., False, False, False], dtype=bool)
In [20]: (meta['var'] == 'V0').sum()
Out[20]: 640
```

V0 Low variation … 640 images
V3 Medium variation … 2560 images
V6 High variation … 2560 images
TFUtils: test_from_params

```
meta = numpy.rec.array data object of length describing metadata for each stimuli, should be thought of as like a "spreadsheet" with one column per label type

Querying subsets: e.g. all low variation-stimuli ("var" level == V0) — there are 640

In [19]: meta['var'] == 'V0'
Out[19]: array([ True,  True,  True, ..., False, False, False], dtype=bool)
In [20]: (meta['var'] == 'V0').sum()
Out[20]: 640

Fancy (database-style) indexing:

In [25]: lowvarfaces = meta[(meta['var'] == 'V0') & (meta['category'] == 'Faces')]
In [26]: lowvarfaces[['category', 'obj', 'ty', 'tz', 'var']][0]
Out[26]: ('Faces', 'face0001', 0., 0., 'V0')
```
TFUtils: test_from_params

```python
meta = numpy.rec.array data object of length describing metadata for each stimuli, should be thought of as like a “spreadsheet” with one column per label type

['obj', #object name
 'category', #object category
 'rxz', 'rxy', 'ryz', #rotational parameters
 'r[xz/xy/yz]_semantic', #semantically consistent rotational parameters,
 'ty', 'tz', #translation
 's', #size
 'bg_id', #id of background image
 'var', #variation level
 '_id', #background + "_" + var_level -- this SHOULD be unique
 'filename', #filename locally -- this is a locally unique identifier
 'id' #hash of last portion of filename -- this is a globally
]
```

The axis labeling here uses the convention that
+\(x\) coming "out of the screen"
+\(z\) is "up" (vertical height) and
+\(y\) is "right" (horizontal extent)
TFUtils: test_from_params

dldata.metrics.utils.compute_metric_base is a unified interface to many different simple machine learning constructs, many of which come from scikit-learn.

```python
def compute_metric_base(F, meta, eval_config, return_splits=False,
                        attach_models=False, dataset=None,
                        attach_predictions=True,
                        knockouts=None, compute_from=None):

    F = feature matrix — N stimuli × M features — can come from model, neurons, fMRI, whatever

    meta = numpy.rec.array data object of length describing metadata for each stimuli,
          like a “spreadsheet” with one meta-data column per label type

    eval_config = dictionary describing which specific type of evaluation to run,
                 referencing back to metadata
```
eval_config = dictionary describing which specific type of evaluation to run.

category_eval_spec = {
    'npc_train': None,
    'npc_test': 2,
    'num_splits': 20,
    'npc_validate': 0,
    'metric_screen': 'classifier',
    'metric_labels': None,
    'metric_kwargs': {'model_type': 'svm.LinearSVC',
                      'model_kwargs': {'C': 5e-3}},
    'labelfunc': 'category',
    'train_q': {'var': ['V3', 'V6']},
    'test_q': {'var': ['V3', 'V6']},
    'split_by': 'obj'
}
TFUtils: test_from_params

eval_config = dictionary describing which specific type of evaluation to run.

category_eval_spec = {
    'npc_train': None,
    'npc_test': 2,  # how many test examples (per "split_by" group) to use
    'num_splits': 20,
    'npc_validate': 0,
    'metric_screen': 'classifier',
    'metric_labels': None,
    'metric_kwargs': {'model_type': 'svm.LinearSVC',
                      'model_kwargs': {'C': 5e-3},
    },
    'labelfunc': 'category',
    'train_q': {'var': ['V3', 'V6']},
    'test_q': {'var': ['V3', 'V6']},
    'split_by': 'obj'
}
TFUtils: test_from_params

eval_config = dictionary describing which specific type of evaluation to run.

category_eval_spec = {
    'npc_train': None,
    'npc_test': 2,
    'num_splits': 20,
    'npc_validate': 0,
    'metric_screen': 'classifier',
    'metric_labels': None,
    'metric_kwargs': {'model_type': 'svm.LinearSVC',
                      'model_kwargs': {'C': 5e-3}
                      },
    'labelfunc': 'category',
    'train_q': {'var': ['V3', 'V6']},
    'test_q': {'var': ['V3', 'V6']},
    'split_by': 'obj'  # which metadata column to stratify your splits by
}
TFUtils: test_from_params

eval_config = dictionary describing which specific type of evaluation to run.

category_eval_spec = {
    'npc_train': None,
    'npc_test': 2,
    'num_splits': 20,
    'npc_validate': 0,
    'metric_screen': 'classifier',
    'metric_labels': None,
    'metric_kwargs': {'model_type': 'svm.LinearSVC',
                      'model_kwargs': {'C': 5e-3}},
    'labelfunc': 'category',
    'train_q': {'var': ['V3', 'V6']},
    'test_q': {'var': ['V3', 'V6']},
    'split_by': 'obj'}
TFUtils: test_from_params

eval_config = dictionary describing which specific type of evaluation to run.

category_eval_spec = {
    'npc_train': None,
    'npc_test': 2,
    'num_splits': 20,
    'npc_validate': 0,
    "metric_screen": 'classifier',  # what type of metric to compute
    'metric_labels': None,
    'metric_kwarg': {
        'model_type': 'svm.LinearSVC',
        'model_kwarg': {'C': 5e-3}
    },
    'labelfunc': 'category',
    'train_q': {'var': ['V3', 'V6']},
    'test_q': {'var': ['V3', 'V6']},
    'split_by': 'obj'
}
**TFUtils: test_from_params**

`eval_config` = dictionary describing which specific type of evaluation to run.

```python
category_eval_spec = {
    'npc_train': None,
    'npc_test': 2,
    'num_splits': 20,
    'npc_validate': 0,
    'metric_screen': 'classifier',
    'metric_labels': None,
    'metric_kwargs': {'model_type': 'svm.LinearSVC',
                      'model_kwargs': {'C': 5e-3}},
    'labelfunc': 'category',
    'train_q': {'var': ['V3', 'V6']},
    'test_q': {'var': ['V3', 'V6']},
    'split_by': 'obj'
}
```

what specific type of (linear) model to build

`sparse_matrix[something]` = object in the `sklearn` svm module
**sklearn.svm**: Support Vector Machines

The **sklearn.svm** module includes Support Vector Machine algorithms.

**User guide**: See the **Support Vector Machines** section for further details.

### Estimators

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>svm.LinearSVC</code></td>
<td>Linear Support Vector Classification.</td>
</tr>
<tr>
<td><code>svm.LinearSVR</code></td>
<td>Linear Support Vector Regression.</td>
</tr>
<tr>
<td><code>svm.NuSVC</code></td>
<td>Nu-Support Vector Classification.</td>
</tr>
<tr>
<td><code>svm.NuSVR</code></td>
<td>Nu Support Vector Regression.</td>
</tr>
<tr>
<td><code>svm.OneClassSVM</code></td>
<td>Unsupervised Outlier Detection.</td>
</tr>
<tr>
<td><code>svm.SVC</code></td>
<td>C-Support Vector Classification.</td>
</tr>
<tr>
<td><code>svm.SVR</code></td>
<td>Epsilon-Support Vector Regression.</td>
</tr>
<tr>
<td><code>svm.l1_min_c</code></td>
<td>Return the lowest bound for C such that for C in (1_min_C, infinity) the model is guaranteed not to be empty.</td>
</tr>
</tbody>
</table>

### Low-level methods

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>svm.libsvm.cross_validation</code></td>
<td>Binding of the cross-validation routine (low-level routine)</td>
</tr>
<tr>
<td><code>svm.libsvm.decision_function</code></td>
<td>Predict margin (libsvm name for this is predict_values)</td>
</tr>
<tr>
<td><code>svm.libsvm.fit</code></td>
<td>Train the model using libsvm (low-level method)</td>
</tr>
<tr>
<td><code>svm.libsvm.predict</code></td>
<td>Predict target values of X given a model (low-level method)</td>
</tr>
<tr>
<td><code>svm.libsvm.predict_proba</code></td>
<td>Predict probabilities</td>
</tr>
</tbody>
</table>
**TFUtils: test_from_params**

`eval_config` = dictionary describing which specific type of evaluation to run.

```python
category_eval_spec = {
    'npc_train': None,
    'npc_test': 2,
    'num_splits': 20,
    'npc_validate': 0,
    'metric_screen': 'classifier',
    'metric_labels': None,
    'metric_kwargs': {'model_type': 'svm.LinearSVC',
                      'model_kwargs': {'C': 5e-3},
    'labelfunc': 'category',
    'train_q': {'var': ['V3', 'V6']},
    'test_q': {'var': ['V3', 'V6']},
    'split_by': 'obj'
}
```

name of a sklearn object
arguments to sklearn object constructor
eval_config = dictionary describing which specific type of evaluation to run.

category_eval_spec = {
    'npc_train': None,
    'npc_test': 2,
    'num_splits': 20,
    'npc_validate': 0,
    'metric_screen': 'classifier',
    'metric_labels': None,
    'metric_kwargs': {'model_type': 'svm.LinearSVC',
                      'model_kwargs': {'C': 5e-3},
    'labelfunc': 'category',
    'train_q': {'var': ['V3', 'V6']},
    'test_q': {'var': ['V3', 'V6']},
    'split_by': 'obj'
}

what metadata column to use as labels
eval_config = dictionary describing which specific type of evaluation to run.

category_eval_spec = {
    'npc_train': None,
    'npc_test': 2,
    'num_splits': 20,
    'npc_validate': 0,
    'metric_screen': 'classifier',
    'metric_labels': None,
    'metric_kwargs': {'model_type': 'svm.LinearSVC',
                      'model_kwargs': {'C': 5e-3},
    },
    'labelfunc': 'category',
    'train_q': {'var': ['V3', 'V6']},
    'test_q': {'var': ['V3', 'V6']},
    'split_by': 'obj'
}
TFUtils: test_from_params

```python
def compute_metric_base(F, meta, eval_config, return_splits=False,
                        attach_models=False, dataset=None,
                        attach_predictions=True,
                        knockouts=None, compute_from=None):
```

```python
results = compute_metric_base(F, meta, eval_config)
```

About 37% accuracy:

```
In [661]: results.keys()
Out[661]:
['split_results',
 'result_summary',
 'acccbal_loss',
 'dp_sym_loss',
 'acc_loss',
 'dp_lin_loss',
 'dp_sym_loss_stderr',
 'acccbal_loss_stderr',
 'multiacc_loss_stderr',
 'dprime_loss',
 'dprime_loss_stderr',
 'acc_loss_stderr',
 'multiacc_loss',
 'dp_lin_loss_stderr']
```

```
In [665]: 100 * (1 - results['multiacc_loss'])
Out[665]: 36.734693877551017
```

Basic categorization
TFUtils: test_from_params

eval_config = dictionary describing which specific type of evaluation to run.

Here, we’re evaluating within category object identification performance (Faces vs each other).

```python
within_category_identification_spec = {
    'npc_train': None,
    'npc_test': 2,
    'num_splits': 20,
    'npc_validate': 0,
    'metric_screen': 'classifier',
    'metric_labels': None,
    'metric_kwvars': {
        'model_type': 'svm.LinearSVC',
        'model_kwargs': {'C': 5e-3},
    }
}
```

[Images of faces]
TFUtils: test_from_params

eval_config = dictionary describing which specific type of evaluation to run.

Here, we're regressing object horizontal position:

```python
position_estimation_spec =
{'npc_train': None,
 'npc_test': 2,
 'num_splits': 20,
 'npc_validate': 0,
 'metric_screen': 'regression',
 'metric_labels': None,
 'metric_kwargs': {'model_type': 'linear_model.RidgeCV'},
 'labelfunc': 'ty',
 'train_q': {'var': ['V3', 'V6']},
 'test_q': {'var': ['V3', 'V6']},
 'split_by': 'obj'
}
```
TFUtils: test_from_params

eval_config = dictionary describing which specific type of evaluation to run.
TFUtils: test_from_params

eval_config = dictionary describing which specific type of evaluation to run.

Here, we're regressing IT neural responses:

```python
it_reg_eval_spec = {
    'labelfunc': lambda x: (IT_features, None),
    'metric_kwarg': {'model_kwarg': {'n_components': 25, 'scale': False},
                     'model_type': 'pls.PLSRegression'},
    'metric_labels': None,
    'metric_screen': 'regression',
    'npc_test': 10,
    'npc_train': 70,
    'npc_validate': 0,
    'num_splits': 5,
    'split_by': 'obj',
    'test_q': {'var': ['V3', 'V6']},
    'train_q': {'var': ['V3', 'V6']}
}
```
**eval_config** = dictionary describing which specific type of evaluation to run.

### sklearn.cross_decomposition: Cross decomposition

**User guide:** See the Cross decomposition section for further details.

- **cross_decomposition.CCA** ([n_components, ...]) — CCA Canonical Correlation Analysis.
- **cross_decomposition.PLSCanonical** ([...]) — PLSCanonical implements the 2 blocks canonical PLS of the original Wold algorithm [Tenenhaus 1998] p.204, referred as PLSC2A in [Wegelin 2000].
- **cross_decomposition.PLSRegression** ([...]) — PLS regression.
- **cross_decomposition.PLSSVD** ([n_components, ...]) — Partial Least Square SVD.
TFUtils: test_from_params

```python
def compute_metric_base(F, meta, eval_config, return_splits=False, attach_models=False, dataset=None, attach_predictions=True, knockouts=None, compute_from=None):
```

```python
results = compute_metric_base(F, meta, eval_config)
```

About 32% raw explained variance:
The relationship of these tools to encoding & decoding is shown in the diagram. The process involves a visual representation of the stimulus through Neurons, leading to behavioral read-out.

Stimulus representation flows through Neurons to Behavior. The visual representation is very nonlinear, which is presumably why so much brainmeat needs to be devoted to it. Behavioral linking functions are fairly linear.

Category, Location, Size, Pose, and Depth relationships are considered in the diagram.

The diagram includes labels for RGC (Rod and Cone Cells), LGN (Lateral Geniculate Nucleus), V1, V2, V4, PIT, CIT, AIT, and CIT AIT.
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” are produced from the visual representation of the stimulus.
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”
Relationship of these tools to encoding & decoding

~Developmental learning of a visual system

Today’s reinforcement task when you “come into the lab”

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”