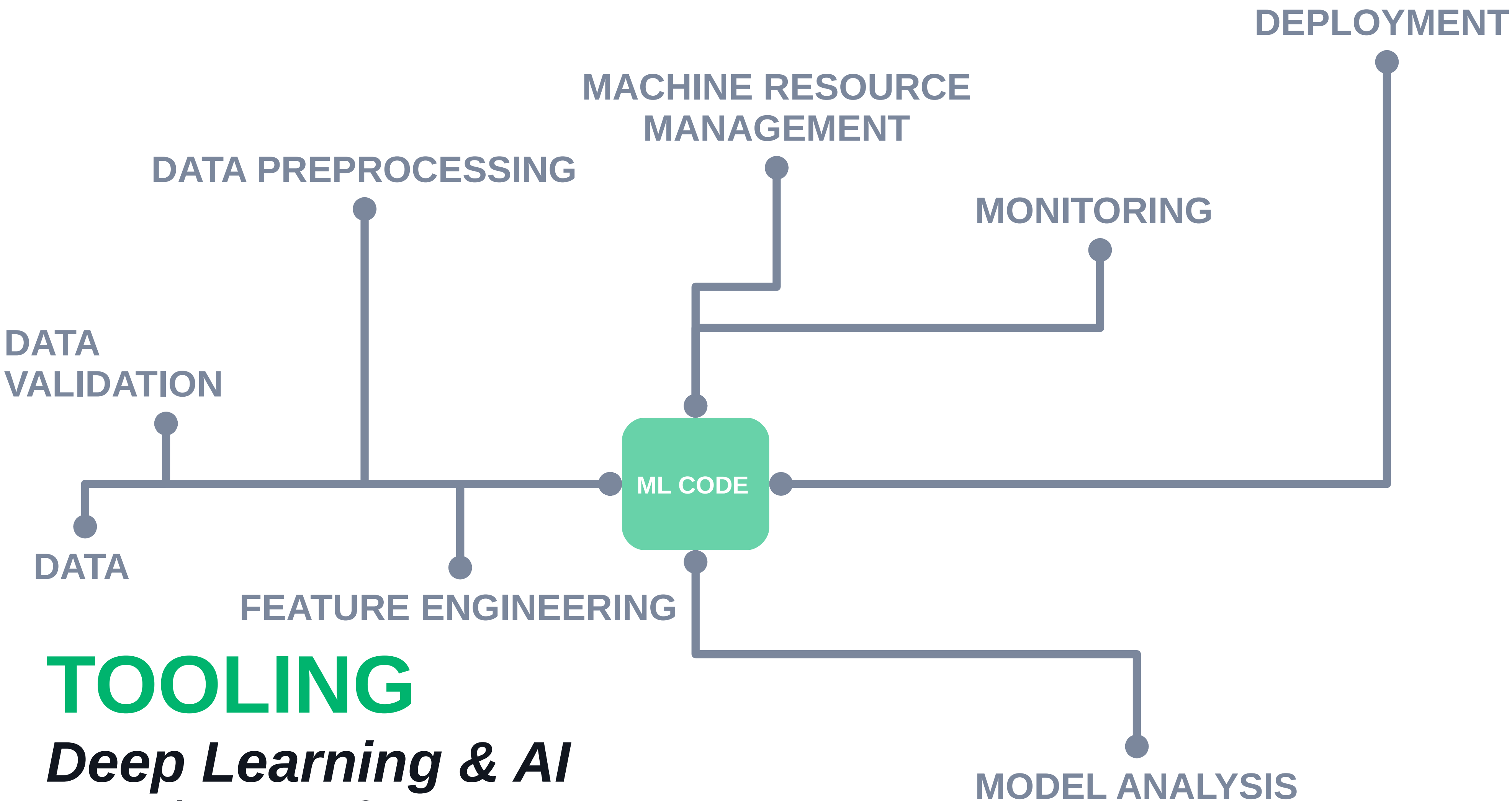


TOOLING

Deep Learning & AI

Dr. Denis Krompaß
Co-Founder creaidAI
Senior Key Expert Deep Learning at Siemens

Lecture Overview

1. SET UP	2. DEVELOP	3. TUNE	4. DEPLOY
	Frameworks	Methods	
Quick Exploration	Debugging	Local Setup	
Cloud Provider	Testing	Cloud Solutions	Inference Pipeline
Bare Metal	Monitoring	AutoML	Model Serving

Going from idea to production

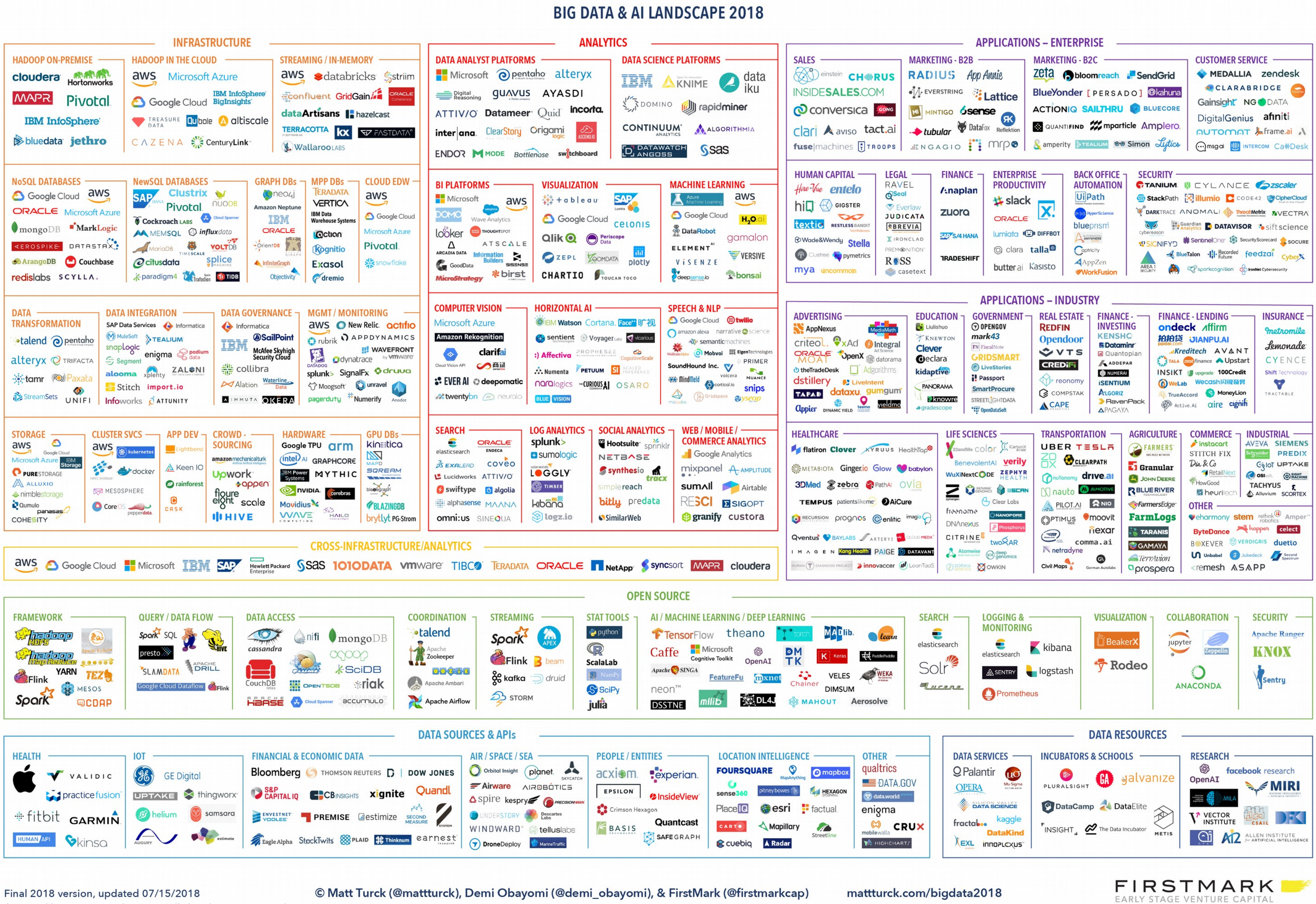


Image from: http://mattturck.com/bigdata2018/

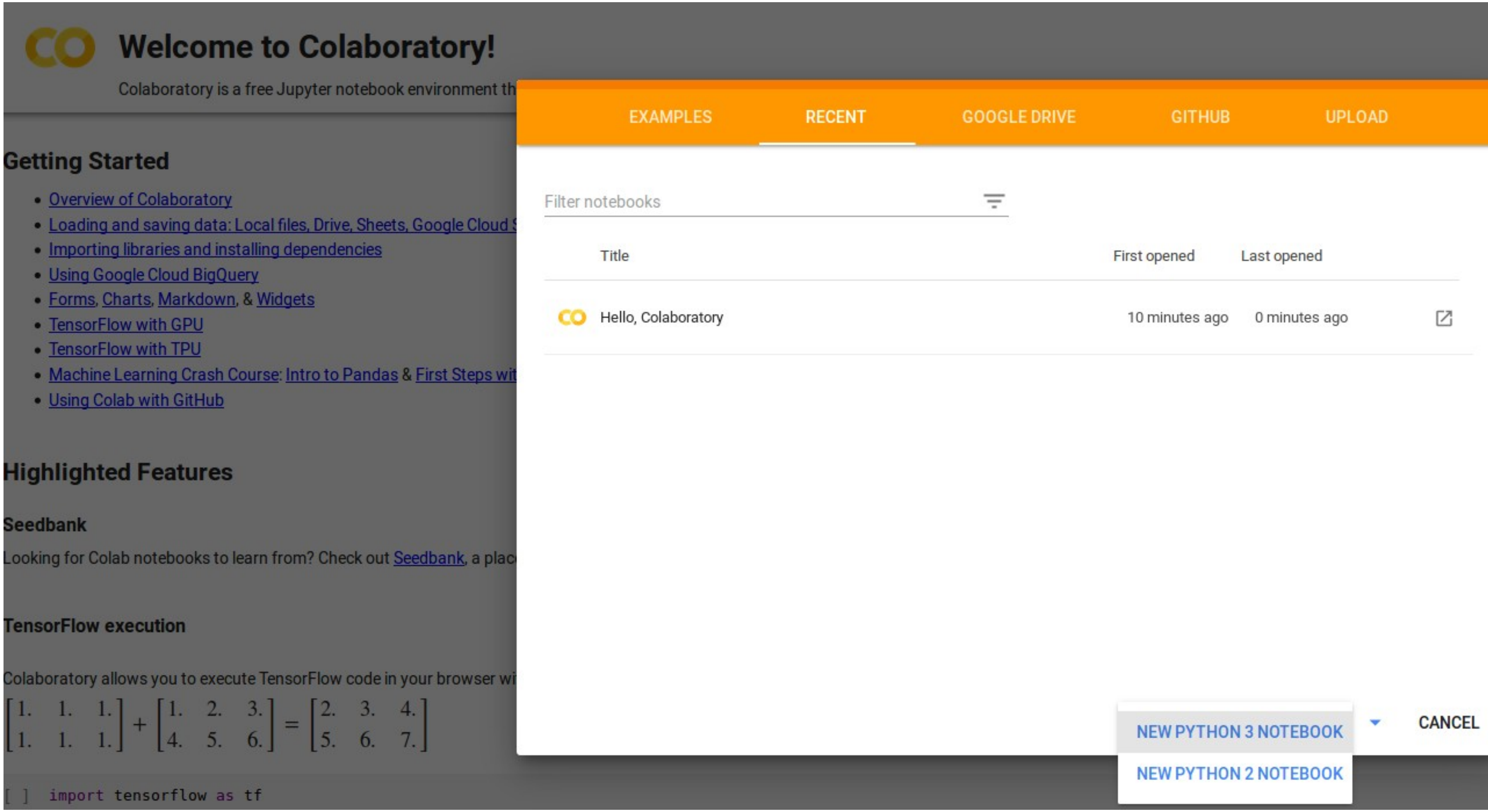
We will cover only one way to do it!

1. SET UP	2. DEVELOP	3. TUNE	4. DEPLOY
Quick Exploration	Frameworks	Methods	Inference Pipeline
Cloud Provider	Debugging	Local Setup	Model Serving
Bare Metal	Testing	Cloud Solutions	
	Monitoring	AutoML	

Going from idea to production

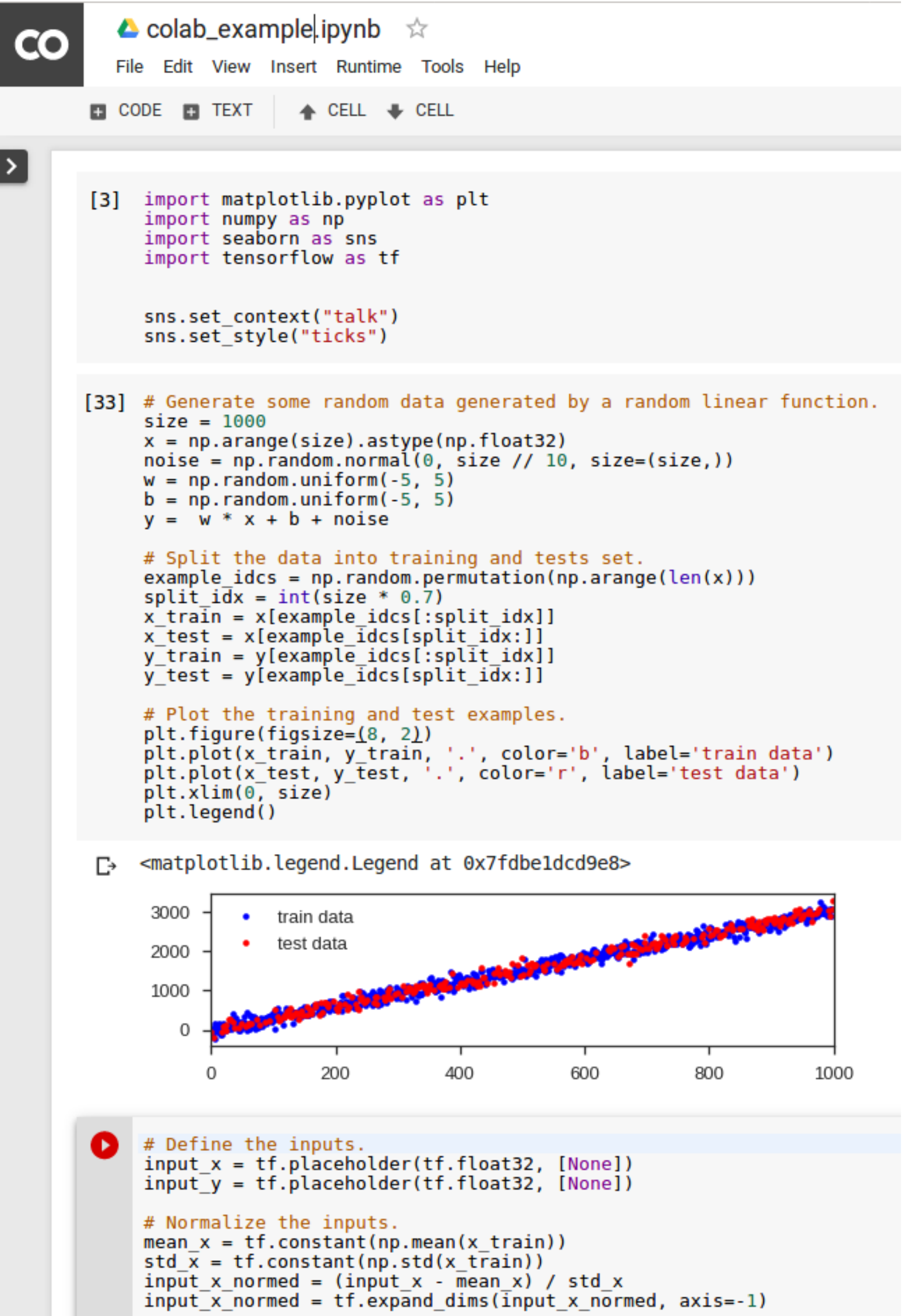
SETUP

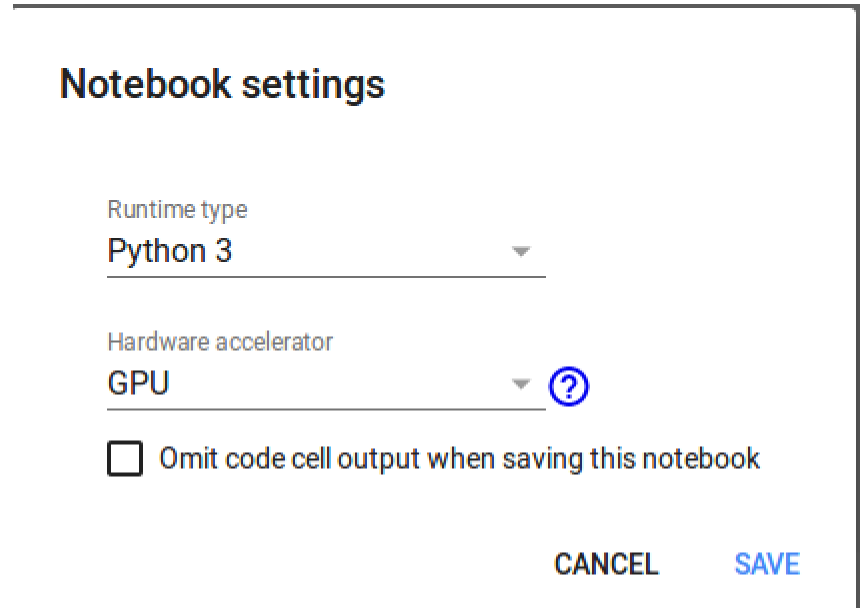
Google Colab



https://colab.research.google.com

Google Colab





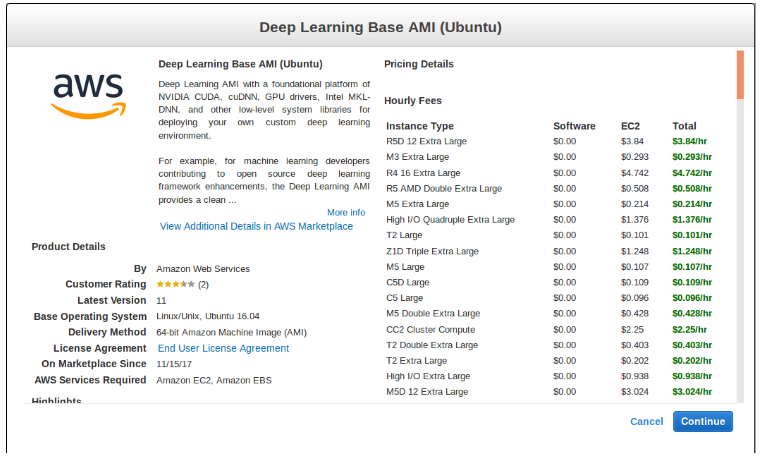
https://colab.research.google.com

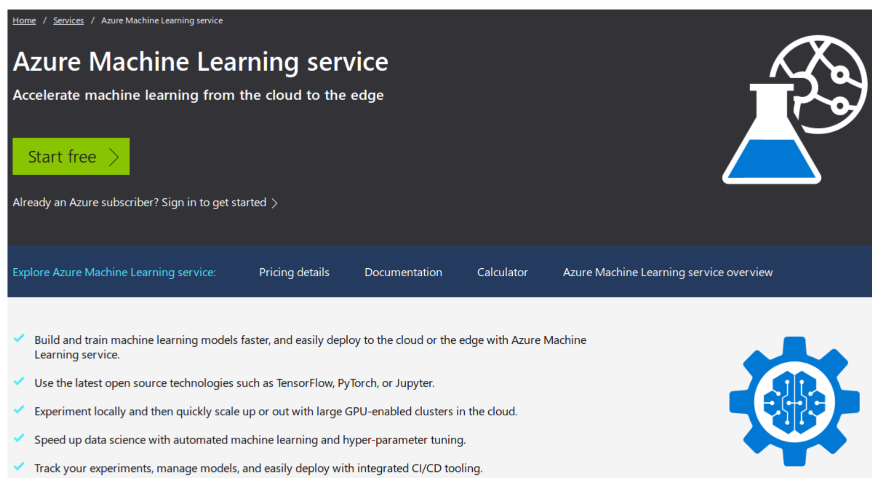
Cloud Provider

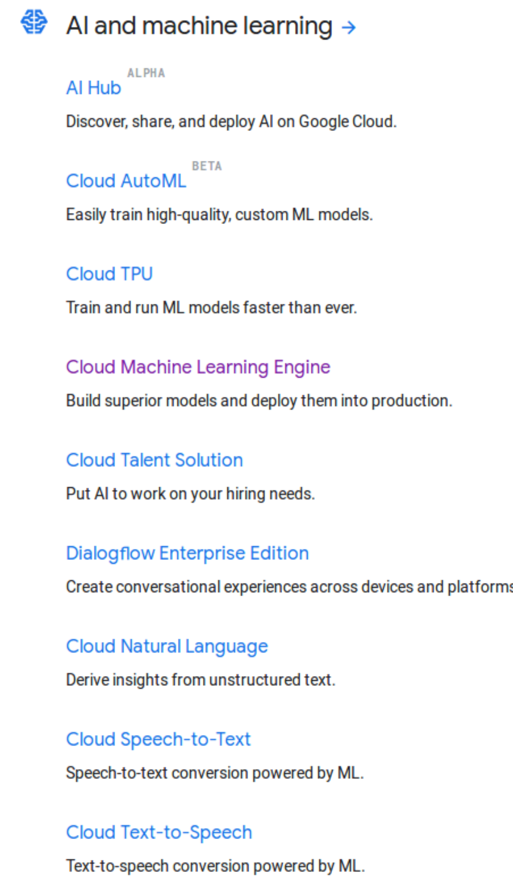
Just the big names, there are more

Offering

- Ready to use environments
- APIs / Libraries for scalable execution
- Pre-build services
- ...







Bare Metal

```
$ pip install tensorflow
```

```
$ conda install tensorflow
```

Bare Metal

$ pip install tensorflow

```
>>> import tensorflow as tf
>>> tf.__version__
'1.12.0'
>>> tf.Session()
2018-11-25 13:46:01.280605: I tensorflow/core/platform/cpu_feature_guard.cc:141]
Your CPU supports instructions that this TensorFlow binary was not compiled to
use: AVX2 FMA
```

$ conda install tensorflow

```
>>> import tensorflow as tf
>>> tf.__version__
'1.12.0'
>>> tf.Session()
2018-11-25 13:58:51.813242: I tensorflow/core/platform/cpu_feature_guard.cc:141]
Your CPU supports instructions that this TensorFlow binary was not compiled to
use: SSE4.1 SSE4.2 AVX AVX2 FMA
```

Bare Metal

$ pip install tensorflow-gpu

cudatoolkit
+
cudnn

```
>>> import tensorflow as tf
…
ImportError: libcublas.so.9.0: cannot open shared object file: No such file or directory
…
```

$ conda install tensorflow-gpu

```
>>> import tensorflow as tf
>>> tf.__version__
'1.12.0'
>>> tf.Session()
2018-11-25 14:13:29.490165: I tensorflow/core/platform/cpu_feature_guard.cc:141]
Your CPU supports instructions that this TensorFlow binary was not compiled to
use: SSE4.1 SSE4.2 AVX AVX2 FMA
2018-11-25 14:13:29.615067: I
tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:964] successful NUMA node
read from SysFS had negative value (-1), but there must be at least one NUMA node,
so returning NUMA node zero
2018-11-25 14:13:29.615760: I
tensorflow/core/common_runtime/gpu/gpu_device.cc:1432] Found device 0 with
properties:
name: GeForce GTX 1050 major: 6 minor: 1 memoryClockRate(GHz): 1.493
```

Bare Metal

https://www.tensorflow.org/install/source

DEVELOPMENT

Frameworks

DL4J

TensorFlow

mxnet

PyTorch

Keras

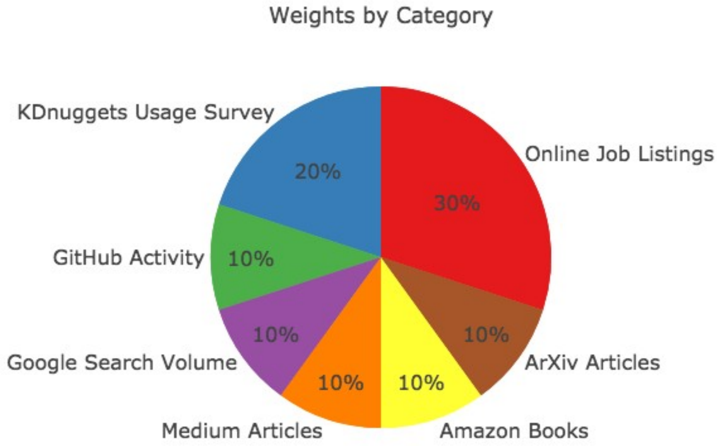
fast.ai

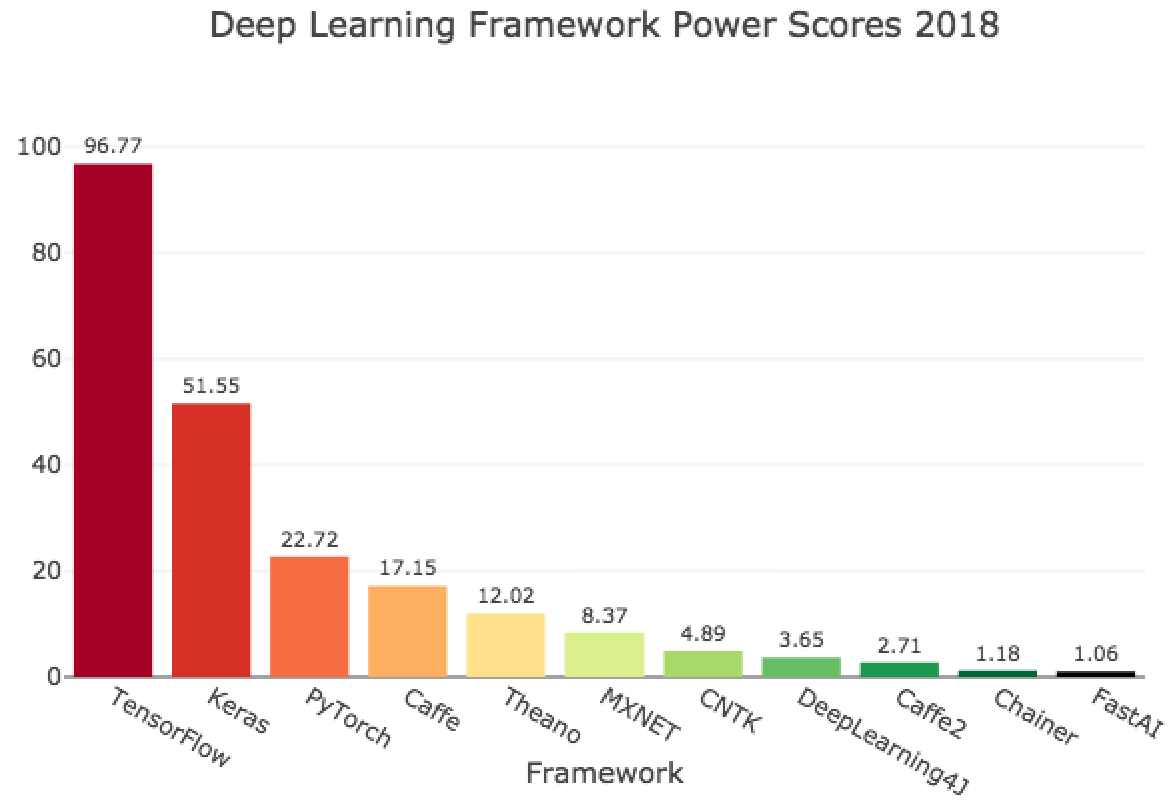
Microsoft
Cognitive Toolkit (CNTK)

Caffe2

And many more ...

Frameworks





Full article: https://towardsdatascience.com/deep-learning-framework-power-scores-2018-23607ddf297a

Before we start…

https://github.com/dekromp/deep_learning_and_ai_tooling_lecture

Beware!

- Notebooks seem convenient, but there are many pitfalls!
  - Hidden states can lead to nasty bugs
    - Reproducibility is difficult
    - Newcomers get easily confused
  - Notebooks encourage bad habits

Nice slide deck that shows the pitfalls of notebooks:
https://docs.google.com/presentation/d/1n2RlMdmv1p25Xy5thJUhkKGvjtV-dkAIsUXP-AL4ffI/preview

Don’t use them for writing your machine learning code!

Notebooks are great for plotting stuff.

Beware!

- Find a good text editor and get familiar with it:

Visual Studio Code

Sublime

Atom

PyCharm

Spyder

And many more ...

Data Scientist are Software Developers

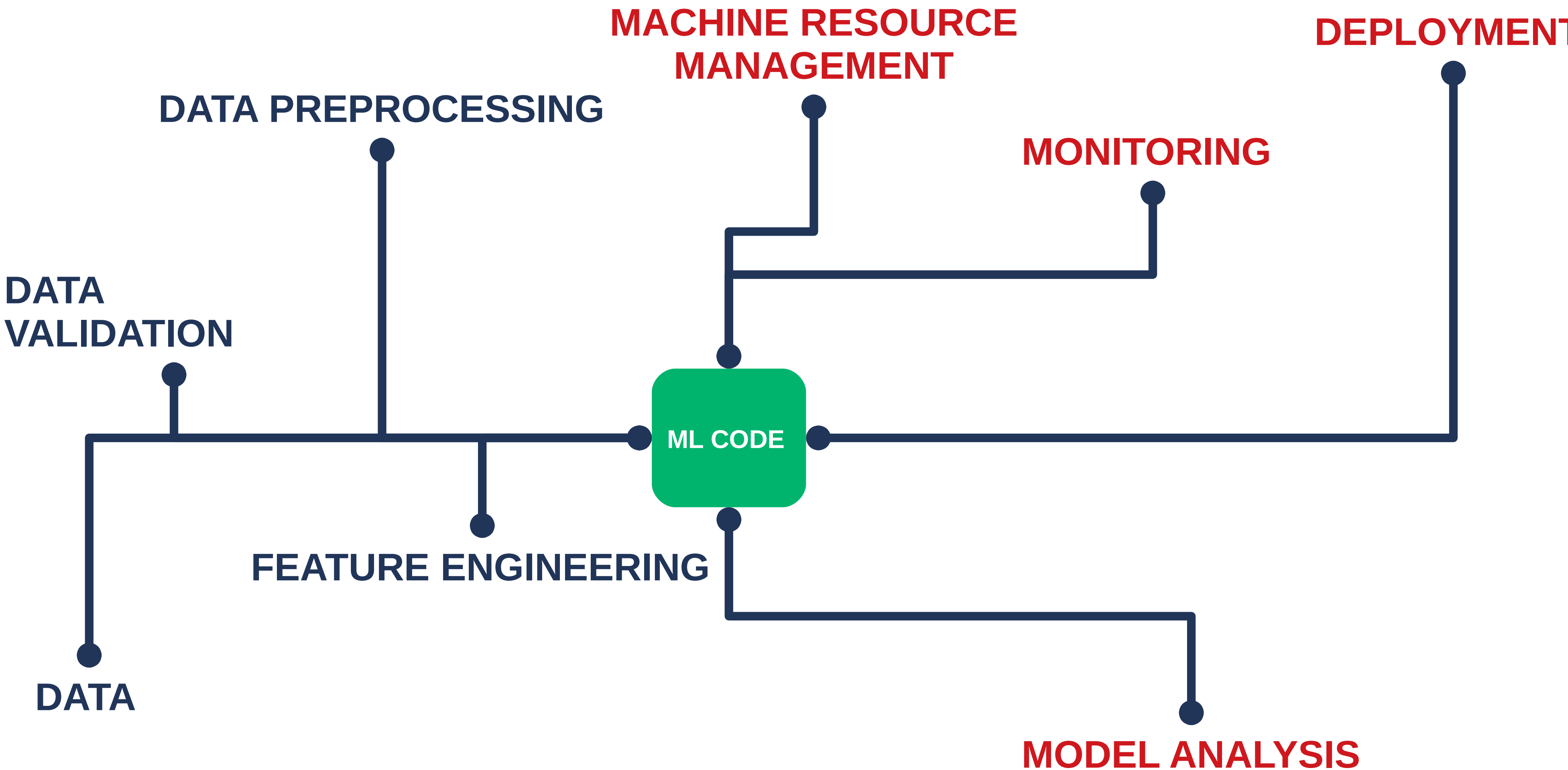
- Get familiar with coding guidelines (Python: PEP 8)
- Document your code (PEP 257, NumPy Style, ...)
- Write tests!!! (e.g. Unit-tests with pytest)
- Modularize your code.

```python
def f(x, y):
    xtxi = np.linalg.pinv(np.dot(x.T, x))
    xty = np.dot(x.T, y)
    w = np.dot(xtxi, xty)
    return w
```

```python
def f(x, y):
    xtxi = np.linalg.pinv(np.dot(x.T, x))
    xty = np.dot(x.T, y)
    w = np.dot(xtxi, xty)
    return w


def fit_linear(x, y):
    """Compute the parameters of a linear regression model in closed form.

    Parameters
    ----------
    x : :class:`numpy.ndarray`
        The feature data.
    y : :class:`numpy.ndarray`
        The target data.

    Returns
    -------
    w : :class:`numpy.ndarray`
        The parameters of the linear regression model.

    """
    # Compute the pseudo-inverse of the covariance matrix.
    xtxi = np.linalg.pinv(np.dot(x.T, x))

    # Compute the parameters of the linear model using the closed form solution
    # w = (XtX)^-1 * Xt * y
    xty = np.dot(x.T, y)
    w = np.dot(xtxi, xty)

    return w
```

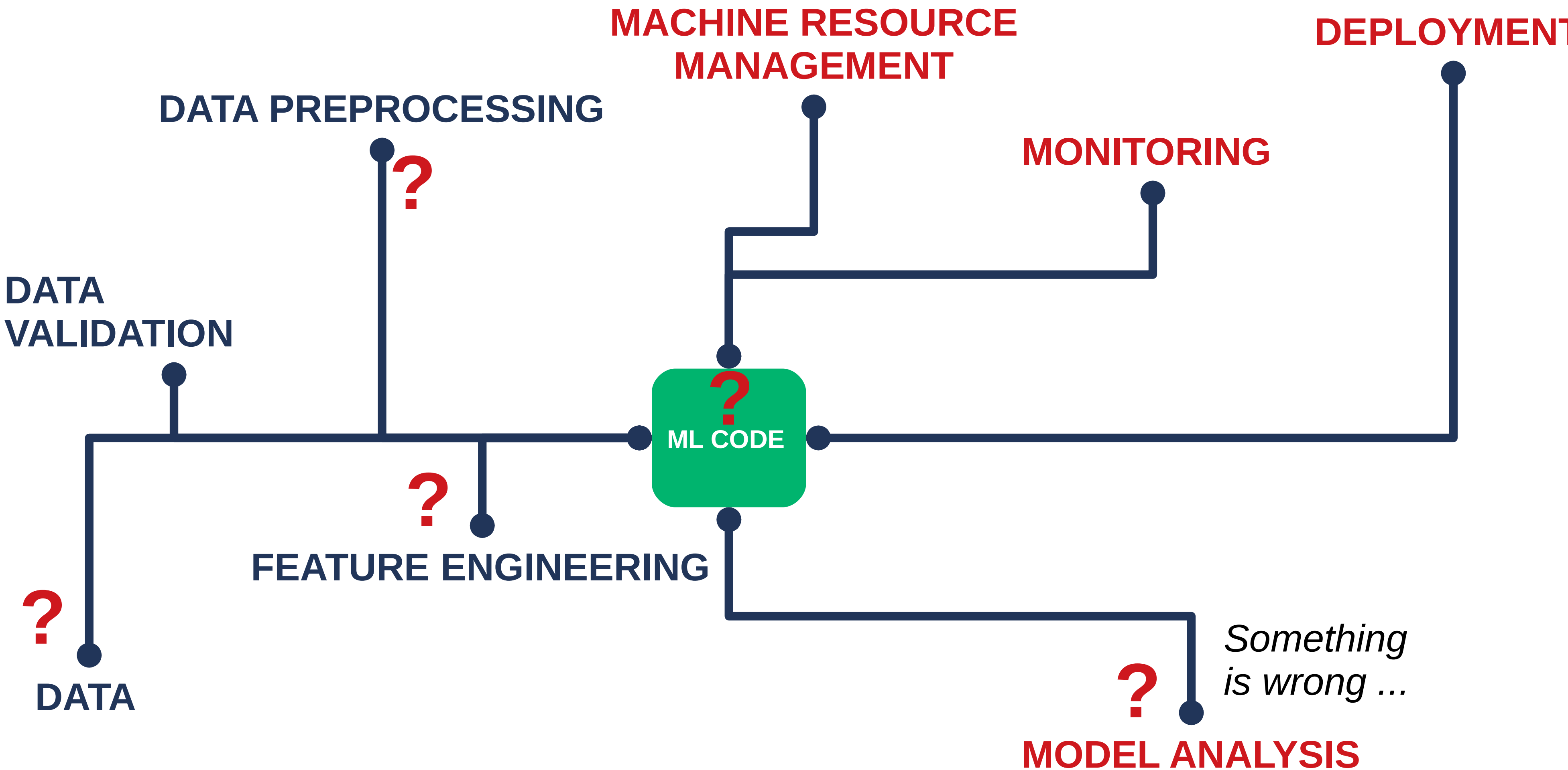
https://github.com/dekromp/deep_learning_and_ai_tooling_lecture/tree/master/tooling_lecture/code_style

Why tests?



MACHINE RESOURCE MANAGEMENT
DEPLOYMENT
DATA PREPROCESSING
MONITORING
DATA VALIDATION
ML CODE
FEATURE ENGINEERING
DATA
MODEL ANALYSIS

Why tests?


MACHINE RESOURCE MANAGEMENT
DEPLOYMENT
DATA PREPROCESSING
MONITORING
DATA VALIDATION
?
ML CODE
?
FEATURE ENGINEERING
?
DATA
Something is wrong ...
?
MODEL ANALYSIS

Testing with pytest

```python
import numpy as np
from numpy.testing import assert_array_almost_equal

from .documented_function_example import fit_linear


np.random.seed(123456)


def test_fit_linear():
    """The fit_linear function from the slides."""
    # Generate a random linear regression model on random data.
    x = np.random.randn(100, 3)
    true_w = np.array([0.3, -0.21, 0.8])
    y = np.dot(x, true_w)

    # Use our function to compute the parameters.
    w = fit_linear(x, y)

    # Should be the same as the true w.
    assert_array_almost_equal(true_w, w)
```

pytest

https://docs.pytest.org/en/latest/contents.html

- code_style
  - __init__.py
  - documented_function_example.py
  - documented_function_example_test.py

```
(dev-p27) [redacted]/code_style(master)$ py.test doc
umented_function_example_test.py
====================================== test session starts ======================================
platform linux2 -- Python 2.7.15, pytest-3.10.0, py-1.7.0, pluggy-0.8.0
rootdir: /home/denis/creaidAI/development/side_projects/lecture/code_style, inifile:
plugins: pep8-1.0.6, cov-2.6.0
collected 1 item

documented_function_example_test.py .                                                     [100%]

=================================== 1 passed in 0.05 seconds ====================================
```

https://github.com/dekromp/deep_learning_and_ai_tooling_lecture/tree/master/tooling_lecture/code_style

Summary

Data Scientist do **not** have a license to write ‘spaghetti code’

In fact, your code (and data) needs to be clean, structured and **better tested** as ‘*regular*’ software code.

The hidden technical debt in machine learning systems. Sculley et al (Google). Neural Information Processing Systems (NIPS) 2015.
https://papers.nips.cc/paper/5656-hidden-technical-debt-in-machine-learning-systems.pdhttp:/martin.zinkevich.org/rules_of_ml/rules_of_ml

Now we can start.

Quick Refresher

```python
"""Simple example script that shows basic operations of tensorflow."""
import numpy as np
import tensorflow as tf


# Fix the random seeds to make the computations reproducable.
tf.set_random_seed(12345)
np.random.seed(12321)

# Create an placeholder for feeding inputs in the graph.
input_x = tf.placeholder(tf.float32, [None, 3], name='features')

# Create a variable.
w = tf.get_variable(
    'weights', [3, 1], initializer=tf.glorot_uniform_initializer())

# Perform some computation steps.
output = tf.matmul(input_x, w)
output = tf.reshape(output, [-1])  # Flatten the outputs.

# Generate some random input data.
x = np.random.randn(5, 3)

# Execute the graph on some random data.
with tf.Session() as session:
    # Boilerplate code that initializes all variables in the graph (just w).
    session.run(tf.global_variables_initializer())
    output_value = session.run(output, feed_dict={input_x: x})
    print('Output: %s' % str(output_value))
    # Output: [ 1.382279  -0.9660325 -0.5551475  0.1781615 -1.5802894]
```

!!!

Reproducibility is a big issue in ML

https://github.com/dekromp/deep_learning_and_ai_tooling_lecture/tree/master/tooling_lecture/refresher

Quick Refresher

```python
"""Simple example script that shows basic operations of tensorflow."""
import numpy as np
import tensorflow as tf


# Fix the random seeds to make the computations reproducable.
tf.set_random_seed(12345)
np.random.seed(12321)

# Create an placeholder for feeding inputs in the graph.
input_x = tf.placeholder(tf.float32, [None, 3], name='features')

# Create a variable.
w = tf.get_variable(
    'weights', [3, 1], initializer=tf.glorot_uniform_initializer())

# Perform some computation steps.
output = tf.matmul(input_x, w)
output = tf.reshape(output, [-1])  # Flatten the outputs.

# Create a target placeholder and define the loss computation.
input_y = tf.placeholder(tf.float32, [None], name='target')
# Mean squared error.
loss = tf.reduce_mean(tf.square(output - input_y))

# Define the update operation (stochastic gradient descent).
update_op = tf.assign(w, w - 0.01 * tf.gradients(loss, w)[0])

# Generate some random training data.
x = np.random.randn(100, 3)
unknown_w = np.array([0.3, -0.21, 0.8])
y = np.dot(x, unknown_w)

# Execute the graph on some random data.
batch_size = 8
num_epochs = 15
with tf.Session() as session:
    # Boilerplate code that initializes all variables in the graph (just w).
    session.run(tf.global_variables_initializer())
    for epoch in range(num_epochs):  # Train for 15 epochs.
        # Shuffle the training data.
        shuffle_idx = np.random.permutation(np.arange(len(x)))
        x = x[shuffle_idx]
        y = y[shuffle_idx]

        # Train the model on batches of data with SGD.
        epoch_losses = []
        for i in range(0, len(x), batch_size):
            batch_loss, _ = session.run(
                [loss, update_op],
                feed_dict={input_x: x[i: i + batch_size],
                           input_y: y[i: i + batch_size]})
            epoch_losses += [batch_loss]

        print('Epoch %d; TrainLoss: %.4f' % (epoch + 1, np.mean(epoch_losses)))

    print('Found parameters: %s' % str(w.eval().reshape(-1)))
    print('True parameters: %s' % str(unknown_w))
```

https://github.com/dekromp/deep_learning_and_ai_tooling_lecture/tree/master/tooling_lecture/refresher

Quick Refresher

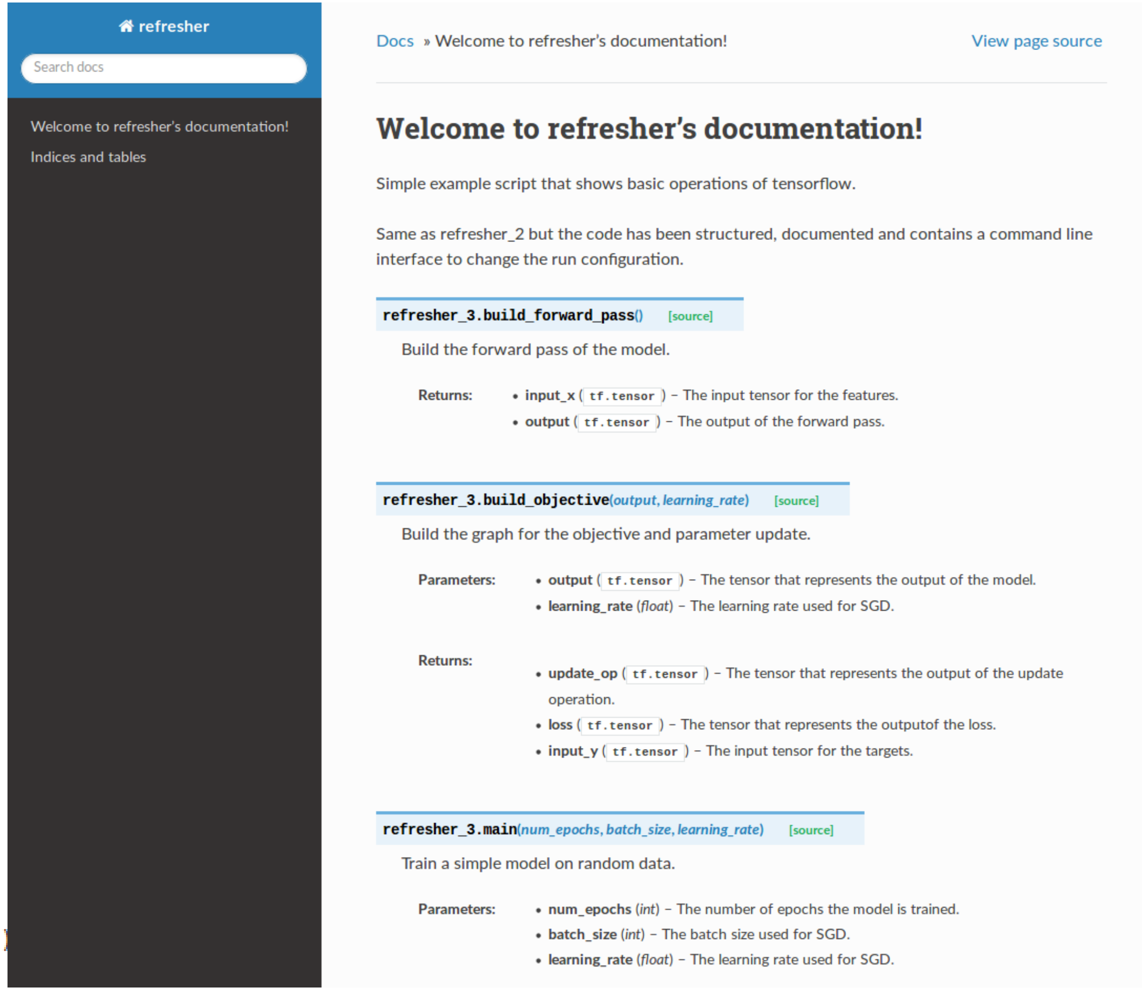
```python
"""Simple example script that shows basic operations of tensorflow."""
import numpy as np
import tensorflow as tf


# Fix the random seeds to make the computations reproducable.
tf.set_random_seed(12345)
np.random.seed(12321)

# Create an placeholder for feeding inputs in the graph.
input_x = tf.placeholder(tf.float32, [None, 3], name='features')

# Create a variable.
w = tf.get_variable(
    'weights', [3, 1], initializer=tf.glorot_uniform_initializer())

# Perform some computation steps.
output = tf.matmul(input_x, w)
output = tf.reshape(output, [-1])  # Flatten the outputs.

# Create a target placeholder and define the loss computation.
input_y = tf.placeholder(tf.float32, [None], name='target')
# Mean squared error.
loss = tf.reduce_mean(tf.square(output - input_y))

# Define the update operation (stochastic gradient descent).
update_op = tf.assign(w, w - 0.01 * tf.gradients(loss, w)[0])

# Generate some random training data.
x = np.random.randn(100, 3)
unknown_w = np.array([0.3, -0.21, 0.8])
y = np.dot(x, unknown_w)

# Execute the graph on some random data.
batch_size = 8
num_epochs = 15
with tf.Session() as session:
    # Boilerplate code that initializes all variables in the graph (just w).
    session.run(tf.global_variables_initializer())
    for epoch in range(num_epochs):  # Train for 15 epochs.
        # Shuffle the training data.
        shuffle_idx = np.random.permutation(np.arange(len(x)))
        x = x[shuffle_idx]
        y = y[shuffle_idx]

        # Train the model on batches of data with SGD.
        epoch_losses = []
        for i in range(0, len(x), batch_size):
            batch_loss, _ = session.run(
                [loss, update_op],
                feed_dict={input_x: x[i: i + batch_size],
                           input_y: y[i: i + batch_size]})
            epoch_losses += [batch_loss]

        print('Epoch %d; TrainLoss: %.4f' % (epoch + 1, np.mean(epoch_losses)))

    print('Found parameters: %s' % str(w.eval().reshape(-1)))
    print('True parameters: %s' % str(unknown_w))
```

```python
# Fix the random seeds to make the computations reproducable.
tf.set_random_seed(12345)
np.random.seed(12321)

# Constants of the experiments.
unknown_true_w = np.array([0.3, -0.21, 0.8])


def main(num_epochs, batch_size, learning_rate):
    """Train a simple model on random data.

    Parameters
    ----------
    num_epochs : int
        The number of epochs the model is trained.
    batch_size : int
        The batch size used for SGD.
    learning_rate : float
        The learning rate used for SGD.

    """
    # Generate some random training data.
    x = np.random.randn(100, 3)
    y = np.dot(x, unknown_true_w)

    # Build forward pass.
    input_x, output = build_forward_pass()
    # Build the update op with respect to the objective.
    update_op, loss, input_y = build_objective(output, learning_rate)
    # Fit the model on the input data.
    inputs = (input_x, input_y)
    data = (x, y)
    train_model(inputs, data, loss, update_op, batch_size, num_epochs)
```

https://github.com/dekromp/deep_learning_and_ai_tooling_lecture/tree/master/tooling_lecture/refresher

Quick Refresher

```python
# Fix the random seeds to make the computations reproducable.
tf.set_random_seed(12345)
np.random.seed(12321)

# Constants of the experiments.
unknown_true_w = np.array([0.3, -0.21, 0.8])


def main(num_epochs, batch_size, learning_rate):
    """Train a simple model on random data.

    Parameters
    ----------
    num_epochs : int
        The number of epochs the model is trained.
    batch_size : int
        The batch size used for SGD.
    learning_rate : float
        The learning rate used for SGD.

    """
    # Generate some random training data.
    x = np.random.randn(100, 3)
    y = np.dot(x, unknown_true_w)

    # Build forward pass.
    input_x, output = build_forward_pass()
    # Build the update op with respect to the objective.
    update_op, loss, input_y = build_objective(output, learning_rate)
    # Fit the model on the input data.
    inputs = (input_x, input_y)
    data = (x, y)
    train_model(inputs, data, loss, update_op, batch_size, num_epochs)
```



refresher

Search docs

Welcome to refresher's documentation!

Indices and tables

Docs » Welcome to refresher's documentation! View page source

Welcome to refresher's documentation!

Simple example script that shows basic operations of tensorflow.

Same as refresher_2 but the code has been structured, documented and contains a command line interface to change the run configuration.

refresher_3.build_forward_pass() [source]

Build the forward pass of the model.

Returns:
- **input_x** (`tf.tensor`) – The input tensor for the features.
- **output** (`tf.tensor`) – The output of the forward pass.

refresher_3.build_objective(*output, learning_rate*) [source]

Build the graph for the objective and parameter update.

Parameters:
- **output** (`tf.tensor`) – The tensor that represents the output of the model.
- **learning_rate** (*float*) – The learning rate used for SGD.

Returns:
- **update_op** (`tf.tensor`) – The tensor that represents the output of the update operation.
- **loss** (`tf.tensor`) – The tensor that represents the outputof the loss.
- **input_y** (`tf.tensor`) – The input tensor for the targets.

refresher_3.main(*num_epochs, batch_size, learning_rate*) [source]

Train a simple model on random data.

Parameters:
- **num_epochs** (*int*) – The number of epochs the model is trained.
- **batch_size** (*int*) – The batch size used for SGD.
- **learning_rate** (*float*) – The learning rate used for SGD.

Nice tutorial:
https://medium.com/@eikonomega/getting-started-with-sphinx-autodoc-part-1-2cebbbca5365

Debugging

Debugging Tensorflow can be intimidating...

```
Traceback (most recent call last):
  File "/home/denis/miniconda2/envs/dev-p36/lib/python3.6/site-packages/tensorflow/python/client/session.py", line 1334, in _do_call
    return fn(*args)
  File "/home/denis/miniconda2/envs/dev-p36/lib/python3.6/site-packages/tensorflow/python/client/session.py", line 1319, in _run_fn
    options, feed_dict, fetch_list, target_list, run_metadata)
  File "/home/denis/miniconda2/envs/dev-p36/lib/python3.6/site-packages/tensorflow/python/client/session.py", line 1407, in _call_tf_sessionrun
    run_metadata)
tensorflow.python.framework.errors_impl.InvalidArgumentError: Matrix size-incompatible: In[0]: [110,1], In[1]: [11,10]
         [[{{node MatMul}} = MatMul[T=DT_FLOAT, transpose_a=false, transpose_b=false, _device="/job:localhost/replica:0/task:0/device:CPU:0"](ArithmeticOptimizer/SimplifyAggregation_Mul_add, ArithmeticOptimizer/SimplifyAggregation_Mul_add_1)]]

During handling of the above exception, another exception occurred:

Traceback (most recent call last):
  File "nightmare_error_message.py", line 10, in <module>
    session.run(c, feed_dict={x: np.random.randn(11, 10)})
  File "/home/denis/miniconda2/envs/dev-p36/lib/python3.6/site-packages/tensorflow/python/client/session.py", line 929, in run
    run_metadata_ptr)
  File "/home/denis/miniconda2/envs/dev-p36/lib/python3.6/site-packages/tensorflow/python/client/session.py", line 1152, in _run
    feed_dict_tensor, options, run_metadata)
  File "/home/denis/miniconda2/envs/dev-p36/lib/python3.6/site-packages/tensorflow/python/client/session.py", line 1328, in _do_run
    run_metadata)
  File "/home/denis/miniconda2/envs/dev-p36/lib/python3.6/site-packages/tensorflow/python/client/session.py", line 1348, in _do_call
    raise type(e)(node_def, op, message)
tensorflow.python.framework.errors_impl.InvalidArgumentError: Matrix size-incompatible: In[0]: [110,1], In[1]: [11,10]
         [[node MatMul (defined at nightmare_error_message.py:8)  = MatMul[T=DT_FLOAT, transpose_a=false, transpose_b=false, _device="/job:localhost/replica:0/task:0/device:CPU:0"](ArithmeticOptimizer/SimplifyAggregation_Mul_add, ArithmeticOptimizer/SimplifyAggregation_Mul_add_1)]]

Caused by op 'MatMul', defined at:
  File "nightmare_error_message.py", line 8, in <module>
    c = tf.matmul(x1 + x1, x + x)
  File "/home/denis/miniconda2/envs/dev-p36/lib/python3.6/site-packages/tensorflow/python/ops/math_ops.py", line 2057, in matmul
    a, b, transpose_a=transpose_a, transpose_b=transpose_b, name=name)
  File "/home/denis/miniconda2/envs/dev-p36/lib/python3.6/site-packages/tensorflow/python/ops/gen_math_ops.py", line 4560, in mat_mul
    name=name)
  File "/home/denis/miniconda2/envs/dev-p36/lib/python3.6/site-packages/tensorflow/python/framework/op_def_library.py", line 787, in _apply_op_helper
    op_def=op_def)
  File "/home/denis/miniconda2/envs/dev-p36/lib/python3.6/site-packages/tensorflow/python/util/deprecation.py", line 488, in new_func
    return func(*args, **kwargs)
  File "/home/denis/miniconda2/envs/dev-p36/lib/python3.6/site-packages/tensorflow/python/framework/ops.py", line 3274, in create_op
    op_def=op_def)
  File "/home/denis/miniconda2/envs/dev-p36/lib/python3.6/site-packages/tensorflow/python/framework/ops.py", line 1770, in __init__
    self._traceback = tf_stack.extract_stack()

InvalidArgumentError (see above for traceback): Matrix size-incompatible: In[0]: [110,1], In[1]: [11,10]
         [[node MatMul (defined at nightmare_error_message.py:8)  = MatMul[T=DT_FLOAT, transpose_a=false, transpose_b=false, _device="/job:localhost/replica:0/task:0/device:CPU:0"](ArithmeticOptimizer/SimplifyAggregation_Mul_add, ArithmeticOptimizer/SimplifyAggregation_Mul_add_1)]]
```

```python
1  """A simple script that produces some error message from tensorflow."""
2  import numpy as np
3  import tensorflow as tf
4
5
6  x = tf.placeholder(tf.float32, [None, 10])
7  x1 = tf.reshape(x, [-1, 1])
8  c = tf.matmul(x1 + x1, x + x)
9  with tf.Session() as session:
10     session.run(c, feed_dict={x: np.random.randn(11, 10)})
```

https://github.com/dekromp/deep_learning_and_ai_tooling_lecture/tree/master/tooling_lecture/debug_tensorflow

Debugging

If you get used to it, the errors contain a lot of valuable information.

```
Traceback (most recent call last):
  File "/home/denis/miniconda2/envs/dev-p36/lib/python3.6/site-packages/tensorflow/python/client/session.py", line 1334, in _do_call
    return fn(*args)
  File "/home/denis/miniconda2/envs/dev-p36/lib/python3.6/site-packages/tensorflow/python/client/session.py", line 1319, in _run_fn
    options, feed_dict, fetch_list, target_list, run_metadata)
  File "/home/denis/miniconda2/envs/dev-p36/lib/python3.6/site-packages/tensorflow/python/client/session.py", line 1407, in _call_tf_sessionrun
    run_metadata)
tensorflow.python.framework.errors_impl.InvalidArgumentError: Matrix size-incompatible: In[0]: [110,1], In[1]: [11,10]
         [[{{node MatMul}} = MatMul[T=DT_FLOAT, transpose_a=false, transpose_b=false, _device="/job:localhost/replica:0/task:0/device:CPU:0"](ArithmeticOptimizer/SimplifyAggregation_Mul_add, ArithmeticOptimizer/SimplifyAggregation_Mul_add_1)]]

During handling of the above exception, another exception occurred:

Traceback (most recent call last):
  File "nightmare_error_message.py", line 10, in <module>
    session.run(c, feed_dict={x: np.random.randn(11, 10)})
  File "/home/denis/miniconda2/envs/dev-p36/lib/python3.6/site-packages/tensorflow/python/client/session.py", line 929, in run
    run_metadata_ptr)
  File "/home/denis/miniconda2/envs/dev-p36/lib/python3.6/site-packages/tensorflow/python/client/session.py", line 1152, in _run
    feed_dict_tensor, options, run_metadata)
  File "/home/denis/miniconda2/envs/dev-p36/lib/python3.6/site-packages/tensorflow/python/client/session.py", line 1328, in _do_run
    run_metadata)
  File "/home/denis/miniconda2/envs/dev-p36/lib/python3.6/site-packages/tensorflow/python/client/session.py", line 1348, in _do_call
    raise type(e)(node_def, op, message)
tensorflow.python.framework.errors_impl.InvalidArgumentError: Matrix size-incompatible: In[0]: [110,1], In[1]: [11,10]
         [[node MatMul (defined at nightmare_error_message.py:8)  = MatMul[T=DT_FLOAT, transpose_a=false, transpose_b=false, _device="/job:localhost/replica:0/task:0/device:CPU:0"](ArithmeticOptimizer/SimplifyAggregation_Mul_add, ArithmeticOptimizer/SimplifyAggregation_Mul_add_1)]]

Caused by op 'MatMul', defined at:
  File "nightmare_error_message.py", line 8, in <module>
    c = tf.matmul(x1 + x1, x + x)
  File "/home/denis/miniconda2/envs/dev-p36/lib/python3.6/site-packages/tensorflow/python/ops/math_ops.py", line 2057, in matmul
    a, b, transpose_a=transpose_a, transpose_b=transpose_b, name=name)
  File "/home/denis/miniconda2/envs/dev-p36/lib/python3.6/site-packages/tensorflow/python/ops/gen_math_ops.py", line 4560, in mat_mul
    name=name)
  File "/home/denis/miniconda2/envs/dev-p36/lib/python3.6/site-packages/tensorflow/python/framework/op_def_library.py", line 787, in _apply_op_helper
    op_def=op_def)
  File "/home/denis/miniconda2/envs/dev-p36/lib/python3.6/site-packages/tensorflow/python/util/deprecation.py", line 488, in new_func
    return func(*args, **kwargs)
  File "/home/denis/miniconda2/envs/dev-p36/lib/python3.6/site-packages/tensorflow/python/framework/ops.py", line 3274, in create_op
    op_def=op_def)
  File "/home/denis/miniconda2/envs/dev-p36/lib/python3.6/site-packages/tensorflow/python/framework/ops.py", line 1770, in __init__
    self._traceback = tf_stack.extract_stack()

InvalidArgumentError (see above for traceback): Matrix size-incompatible: In[0]: [110,1], In[1]: [11,10]
         [[node MatMul (defined at nightmare_error_message.py:8)  = MatMul[T=DT_FLOAT, transpose_a=false, transpose_b=false, _device="/job:localhost/replica:0/task:0/device:CPU:0"](ArithmeticOptimizer/SimplifyAggregation_Mul_add, ArithmeticOptimizer/SimplifyAggregation_Mul_add_1)]]
```

```python
"""A simple script that produces some error message from tensorflow."""
import numpy as np
import tensorflow as tf


x = tf.placeholder(tf.float32, [None, 10])
x1 = tf.reshape(x, [-1, 1])
c = tf.matmul(x1 + x1, x + x)
with tf.Session() as session:
    session.run(c, feed_dict={x: np.random.randn(11, 10)})
```

https://github.com/dekromp/deep_learning_and_ai_tooling_lecture/tree/master/tooling_lecture/debug_tensorflow

Debugging

- You can improve the readability of the graph by grouping tensors and variables into **scopes**.

```python
"""A simple script shows how scopes work."""
import numpy as np
import tensorflow as tf


x = tf.placeholder(tf.float32, [None, 10])

print('No scope is used:')
w1 = tf.get_variable(
    'v1', dtype=np.float32, initializer=np.ones((10, 3), dtype=np.float32))
h1 = tf.matmul(x, w1)

w2 = tf.get_variable(
    'v2', dtype=np.float32, initializer=np.ones((3, 10), dtype=np.float32))
h2 = tf.matmul(h1, w2)

for tensor in [x, w1, w2, h1, h2]:
    print(tensor)

print('\nScope is used:')
with tf.variable_scope('first_block'):
    w1 = tf.get_variable(
        'v1', dtype=np.float32, initializer=np.ones((10, 3), dtype=np.float32))
    h1 = tf.matmul(x, w1)

with tf.variable_scope('second_block'):
    w2 = tf.get_variable(
        'v2', dtype=np.float32, initializer=np.ones((3, 10), dtype=np.float32))
    h2 = tf.matmul(h1, w2)
for tensor in [x, w1, w2, h1, h2]:
    print(tensor)
```

```
No scope is used:
Tensor("Placeholder:0", shape=(?, 10), dtype=float32)
<tf.Variable 'v1:0' shape=(10, 3) dtype=float32_ref>
<tf.Variable 'v2:0' shape=(3, 10) dtype=float32_ref>
Tensor("MatMul:0", shape=(?, 3), dtype=float32)
Tensor("MatMul_1:0", shape=(?, 10), dtype=float32)

Scope is used:
Tensor("Placeholder:0", shape=(?, 10), dtype=float32)
<tf.Variable 'first_block/v1:0' shape=(10, 3) dtype=float32_re
<tf.Variable 'second_block/v2:0' shape=(3, 10) dtype=float32_r
Tensor("first_block/MatMul:0", shape=(?, 3), dtype=float32)
Tensor("second_block/MatMul:0", shape=(?, 10), dtype=float32)
```

https://github.com/dekromp/deep_learning_and_ai_tooling_lecture/tree/master/tooling_lecture/debug_tensorflow

Debugging

- Something seems to be wrong...

```
(dev-p36) [redacted] (master)$ python buggy_tensorflow_nan_output.py
2018-11-26 13:55:42.726381: I tensorflow/core/platform/cpu_feature_guard.cc:141] Your CPU supports instructions that this TensorFl
2018-11-26 13:55:42.729947: I tensorflow/core/common_runtime/process_util.cc:69] Creating new thread pool with default inter op se
rformance.
Epoch 1; TrainLoss: nan
Epoch 2; TrainLoss: nan
Epoch 3; TrainLoss: nan
Epoch 4; TrainLoss: nan
Epoch 5; TrainLoss: nan
Epoch 6; TrainLoss: nan
Epoch 7; TrainLoss: nan
Epoch 8; TrainLoss: nan
Epoch 9; TrainLoss: nan
Epoch 10; TrainLoss: nan
Epoch 11; TrainLoss: nan
Epoch 12; TrainLoss: nan
Epoch 13; TrainLoss: nan
Epoch 14; TrainLoss: nan
Epoch 15; TrainLoss: nan
Epoch 16; TrainLoss: nan
Epoch 17; TrainLoss: nan
Epoch 18; TrainLoss: nan
Epoch 19; TrainLoss: nan
Epoch 20; TrainLoss: nan
Epoch 21; TrainLoss: nan
```

https://github.com/dekromp/deep_learning_and_ai_tooling_lecture/tree/master/tooling_lecture/debug_tensorflow

Debugging

- **Tensorlow Debugger** is a great tool to get to the bottom of this.

```python
import numpy as np
import tensorflow as tf
from tensorflow.python import debug as tf_debug
⋮
    with tf.Session() as session:
        session = tf_debug.LocalCLIDebugWrapperSession(session)
```

- Import it and wrap the session, just execute the code again.

```
--- run-start: run #1: 1 fetch (init); 0 feeds ---------------------------------------------------
| <-- --> | run_info
| run | invoke_stepper | exit |

TTTTTT FFFF DDD  BBBB   GGG
  TT   F    D  D B   B G
  TT   FFF  D  D BBBB  G  GG
  TT   F    D  D B   B G   G
  TT   F    DDD  BBBB   GGG

TensorFlow version: 1.12.0

======================================
Session.run() call #1:

Fetch(es):
  init

Feed dict:
  (Empty)
======================================

Select one of the following commands to proceed ---->
  run:
    Execute the run() call with debug tensor-watching
  run -n:
    Execute the run() call without debug tensor-watching
  run -t <T>:
    Execute run() calls (T - 1) times without debugging, then execute run() once more with debugging and drop back to the CLI
  run -f <filter_name>:
    Keep executing run() calls until a dumped tensor passes a given, registered filter (conditional breakpoint mode)
    Registered filter(s):
        * has_inf_or_nan
  invoke_stepper:
    Use the node-stepper interface, which allows you to interactively step through nodes involved in the graph run() call and inspect/modify their values

For more details, see help..

--- Scroll (PgDn): 0.00% -------------------------------------------------------------------------
tfdbg>
```

- Enter ‘run’ to get to the first session run call.
  - ```
    # Initialize all variables in the graph.
    session.run(tf.global_variables_initializer())
    ```

https://github.com/dekromp/deep_learning_and_ai_tooling_lecture/tree/master/tooling_lecture/debug_tensorflow
    ```

```
--- run-end: run #1: 1 fetch (init); 0 feeds -----------------------------------
| <-- --> | (-1) lt
| list_tensors | node_info | print_tensor | list_inputs | list_outputs | run_info | help |
30 dumped tensor(s):                                                              UP

t (ms)   Size (B) Op type        Tensor name
[0.000]  182      VariableV2     layer1/W:0
[0.005]  258      Const          layer1/W/Initializer/truncated_normal/shape:0
[3.317]  178      VariableV2     layer1/b:0
[3.350]  252      Const          layer1/W/Initializer/truncated_normal/stddev:0
[3.380]  182      VariableV2     layer2/W:0
[3.405]  236      Const          layer1/b/Initializer/Const:0
[3.458]  178      VariableV2     layer2/b:0
[3.497]  258      Const          layer2/W/Initializer/truncated_normal/shape:0
[3.510]  194      VariableV2     output_layer/W:0
[3.583]  190      VariableV2     output_layer/b:0
[3.650]  476      TruncatedNormal layer1/W/Initializer/truncated_normal/TruncatedNormal:0
[3.711]  452      Mul            layer1/W/Initializer/truncated_normal/mul:0
[3.754]  444      Snapshot       layer1/W/Initializer/truncated_normal:0
[3.798]  400      Assign         layer1/W/Assign:0
[3.840]  214      Assign         layer1/b/Assign:0
[3.949]  252      Const          layer2/W/Initializer/truncated_normal/stddev:0
[3.965]  836      TruncatedNormal layer2/W/Initializer/truncated_normal/TruncatedNormal:0
[4.003]  244      Const          layer2/b/Initializer/Const:0
[4.029]  812      Mul            layer2/W/Initializer/truncated_normal/mul:0
[4.082]  804      Snapshot       layer2/W/Initializer/truncated_normal:0
[4.097]  270      Const          output_layer/W/Initializer/truncated_normal/shape:0
[4.136]  760      Assign         layer2/W/Assign:0
[4.191]  222      Assign         layer2/b/Assign:0
[4.211]  264      Const          output_layer/W/Initializer/truncated_normal/stddev:0
[4.247]  334      TruncatedNormal output_layer/W/Initializer/truncated_normal/TruncatedNormal:0
[4.259]  232      Const          output_layer/b/Initializer/Const:0
[4.309]  310      Mul            output_layer/W/Initializer/truncated_normal/mul:0
[4.365]  302      Snapshot       output_layer/W/Initializer/truncated_normal:0
[4.366]  210      Assign         output_layer/b/Assign:0
[4.422]  258      Assign         output_layer/W/Assign:0

                                                                                  DN
--- Scroll (PgDn): 0.00% ------------------------------------------- Mouse: ON --
tfdbg>
```

- Scopes are really useful here, too.
- You can click on the “Tensor name” to show its content.
  - Try layer1/W/Assign:0 which shows the weights.
- Enter ‘run’ again to get to the next session run call.

```
--- run-end: run #2: 2 fetches; 3 feeds ------------------------------------------------------------------------------
| <-- --> | lt
| list_tensors | node_info | print_tensor | list_inputs | list_outputs | run_info | help |
[1.234]  212      Sub                   loss/sub:0                                                              UP
[1.238]  280      Shape                 update_op/gradients/forward_pass/layer2/add_grad/Shape:0
[1.288]  250      Shape                 update_op/gradients/loss/Mul_1_grad/Shape:0
[1.291]  308      BroadcastGradientArgs update_op/gradients/forward_pass/layer2/add_grad/BroadcastGradientArgs:1
[1.337]  300      BroadcastGradientArgs update_op/gradients/forward_pass/layer2/add_grad/BroadcastGradientArgs:0
[1.365]  244      VariableV2            output_layer/W:0
[1.389]  208      VariableV2            layer2/b:0
[1.414]  254      Identity              output_layer/W/read:0
[1.436]  218      Identity              layer2/b/read:0
[1.505]  440      Add                   forward_pass/layer2/add:0
[1.556]  288      Shape                 update_op/gradients/forward_pass/layer2/Maximum_grad/Shape:0
[1.560]  354      GreaterEqual          update_op/gradients/forward_pass/layer2/Maximum_grad/GreaterEqual:0
[1.631]  320      BroadcastGradientArgs update_op/gradients/forward_pass/layer2/Maximum_grad/BroadcastGradientArgs:1
[1.643]  448      Maximum               forward_pass/layer2/Maximum:0
[1.682]  308      BroadcastGradientArgs update_op/gradients/forward_pass/layer2/Maximum_grad/BroadcastGradientArgs:0
[1.694]  274      ShapeN                update_op/gradients/forward_pass/concat_grad/ShapeN:0
[1.741]  274      ShapeN                update_op/gradients/forward_pass/concat_grad/ShapeN:1
[1.760]  592      _MklConcatV2          forward_pass/concat:0
[1.794]  286      ConcatOffset          update_op/gradients/forward_pass/concat_grad/ConcatOffset:1
[1.819]  264      MatMul                forward_pass/output_layer/MatMul:0
[1.839]  286      ConcatOffset          update_op/gradients/forward_pass/concat_grad/ConcatOffset:0
[1.887]  292      Shape                 update_op/gradients/forward_pass/output_layer/add_grad/Shape:0
[1.892]  258      Add                   forward_pass/output_layer/add:0
[1.940]  258      Shape                 update_op/gradients/loss/Reshape_grad/Shape:0
[1.942]  324      BroadcastGradientArgs update_op/gradients/forward_pass/output_layer/add_grad/BroadcastGradientArgs:1
[1.987]  320      BroadcastGradientArgs update_op/gradients/forward_pass/output_layer/add_grad/BroadcastGradientArgs:0
[2.015]  220      _MklReshape           loss/Reshape:0
[2.066]  246      Shape                 update_op/gradients/loss/add_grad/Shape:0
[2.071]  216      _MklSub               loss/sub_1:0
[2.122]  216      _MklAdd               loss/add_1:0
[2.144]  278      BroadcastGradientArgs update_op/gradients/loss/add_grad/BroadcastGradientArgs:1
[2.174]  216      Log                   loss/Log_1:0
[2.192]  270      BroadcastGradientArgs update_op/gradients/loss/add_grad/BroadcastGradientArgs:0
[2.221]  254      Shape                 update_op/gradients/loss/Mul_1_grad/Shape_1:0
[2.235]  274      BroadcastGradientArgs update_op/gradients/loss/sub_1_grad/BroadcastGradientArgs:1          DN
--- Scroll (PgDn/PgUp): 21.69% -------------------------------------------------------------------- Mouse: ON --
tfdbg>
```

- That's are a lot of tensors to inspect. Luckily we used scopes in our code. We can use them to filter this list.
- The reported loss was nan so we will start there.
- Enter 'lt -n loss'

```
--- run-end: run #2: 2 fetches; 3 feeds ---------------------------------------------
| <-- --> | lt -n loss
| list_tensors | node_info | print_tensor | list_inputs | list_outputs | run_info | help |
16 dumped tensor(s):
Node name regex filter: "loss"

t (ms)    Size (B) Op type       Tensor name
[0.016]   188      Const         loss/Const:0
[0.560]   184      Const         loss/add/y:0
[0.623]   184      Const         loss/sub/x:0
[0.749]   204      Const         loss/Reshape/shape:0
[1.234]   212      Sub           loss/sub:0
[2.015]   220      _MklReshape   loss/Reshape:0
[2.071]   216      _MklSub       loss/sub_1:0
[2.122]   216      _MklAdd       loss/add_1:0
[2.174]   216      Log           loss/Log_1:0
[2.356]   212      _MklAdd       loss/add:0
[2.364]   216      Mul           loss/Mul_1:0
[2.408]   212      Log           loss/Log:0
[2.603]   212      Mul           loss/Mul:0
[2.653]   216      Add           loss/add_2:0
[2.707]   212      Neg           loss/Neg:0
[2.806]   190      Mean          loss/loss_out:0

--- Scroll (PgDn): 0.00% ------------------------------------------- Mouse: ON --
tfdbg>
```

```
Tensor "loss/Reshape:0:DebugIdentity":
  dtype: float32
  shape: (8,)

array([ 3.6315949e-04, -5.8758940e-04, -5.9261845e-05,  7.0691298e-05,  1.3850409e-03,  1.0980107e-03,  2.6370457e-03,
        1.0636997e-03], dtype=float32)
```

```
Tensor "loss/Log:0:DebugIdentity":
  dtype: float32
  shape: (8,)

array([-7.920393 ,        nan,        nan, -9.555775 , -6.5819535, -6.814164 , -5.9380584, -6.845908 ], dtype=f
```

```
Tensor "loss/loss_out:0:De
  dtype: float32
  shape: ()

array(nan, dtype=float32)
```

```python
    # Build the loss.
    with tf.name_scope('loss'):
        # Flatten the output.
        output = tf.reshape(output, [-1])
        # Create an input for the targets
        input_y = tf.placeholder(tf.float32, [None], name='input_y')

        # Compute the loss (binary cross entropy)
        epsilon = 1e-7  # for numerical stability.
        loss = -(tf.multiply(input_y, tf.log(output + epsilon)) +
                 tf.multiply(1 - input_y, tf.log(1 - output + epsilon)))
        loss = tf.reduce_mean(loss, name='loss_out')
```

- There are negative values flowing into the log of the binary cross entropy...

https://github.com/dekromp/deep_learning_and_ai_tooling_lecture/tree/master/tooling_lecture/debug_tensorflow

```python
def build_forward_pass():
    """Build the forward pass of the model.

    Returns
    -------
    input_x1 : :class:`tf.tensor`
        The input for the first feature set.
    input_x2 : :class:`tf.tensor`
        The input for the second feature set.
    output : :class:`tf.tensor`
        The output of the model.

    """
    with tf.name_scope('forward_pass'):
        input_x1 = tf.placeholder(tf.float32, [None, 10], name='input_x1')
        input_x2 = tf.placeholder(tf.float32, [None, 20], name='input_x2')
        h1 = dense_layer(input_x1, 'layer1', 5, activation=relu)
        h2 = dense_layer(input_x2, 'layer2', 7, activation=relu)
        h = tf.concat([h1, h2], axis=-1)
        output = dense_layer(h, 'output_layer', 1)

    return input_x1, input_x2, output
```

```python
def build_forward_pass():
    """Build the forward pass of the model.

    Returns
    -------
    input_x1 : :class:`tf.tensor`
        The input for the first feature set.
    input_x2 : :class:`tf.tensor`
        The input for the second feature set.
    output : :class:`tf.tensor`
        The output of the model.

    """
    with tf.name_scope('forward_pass'):
        input_x1 = tf.placeholder(tf.float32, [None, 10], name='input_x1')
        input_x2 = tf.placeholder(tf.float32, [None, 20], name='input_x2')
        h1 = dense_layer(input_x1, 'layer1', 5, activation=relu)
        h2 = dense_layer(input_x2, 'layer2', 7, activation=relu)
        h = tf.concat([h1, h2], axis=-1)
        logits = dense_layer(h, 'output_layer', 1)
        output = sigmoid(logits)

    return input_x1, input_x2, output
```

```
(dev-p36)                                                              w(master)$ python buggy_tensorflow_nan_output.py
2018-11-26 14:41:10.734006: I tensorflow/core/platform/cpu_feature_guard.cc:141] Your CPU supports instructions that this TensorFl
2018-11-26 14:41:10.737544: I tensorflow/core/common_runtime/process_util.cc:69] Creating new thread pool with default inter op se
Epoch 1; TrainLoss: 0.6931
Epoch 2; TrainLoss: 0.6930
Epoch 3; TrainLoss: 0.6928
Epoch 4; TrainLoss: 0.6927
Epoch 5; TrainLoss: 0.6925
Epoch 6; TrainLoss: 0.6922
Epoch 7; TrainLoss: 0.6918
Epoch 8; TrainLoss: 0.6911
Epoch 9; TrainLoss: 0.6900
Epoch 10; TrainLoss: 0.6882
Epoch 11; TrainLoss: 0.6854
Epoch 12; TrainLoss: 0.6811
Epoch 13; TrainLoss: 0.6744
Epoch 14; TrainLoss: 0.6645
Epoch 15; TrainLoss: 0.6501
Epoch 16; TrainLoss: 0.6306
Epoch 17; TrainLoss: 0.6053
Epoch 18; TrainLoss: 0.5750
Epoch 19; TrainLoss: 0.5410
Epoch 20; TrainLoss: 0.5052
Epoch 21; TrainLoss: 0.4685
Epoch 22; TrainLoss: 0.4325
Epoch 23; TrainLoss: 0.3972
Epoch 24; TrainLoss: 0.3640
Epoch 25; TrainLoss: 0.3332
```

https://github.com/dekromp/deep_learning_and_ai_tooling_lecture/tree/master/tooling_lecture/debug_tensorflow

Debugging

Don't use tf.print for debugging your code.

It's tedious to use.
It's adds more code (that you have to remove later).

Tensorflow Debugger works also with tf.keras, tf.estimator …
https://www.tensorflow.org/guide/debugger

Tensorflow Eager Execution

- Tensorflow's (and others) symbolic programming style is:
  - Unintuitive for newcomers
  - Hard to debug (hopefully less hard now)
  - People feel comfortable with imparative programming
- Inspired by PyTorch

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

Eager Execution Example

```python
"""This script shows a simple example on how eager execution works."""
import numpy as np
import tensorflow as tf


# Enable eager execution.
tf.enable_eager_execution()

# Make the execution reproducable.
tf.set_random_seed(2132)
np.random.seed(3423)

# Generate some random data.
x = np.arange(3).reshape(-1, 1).astype(np.float32)
w = tf.get_variable(
    'w', dtype=np.float32, shape=[1, 3],
    initializer=tf.glorot_uniform_initializer())

# Interwine python and tensorflow code directly.
z = tf.matmul(w, x)
if np.sum(x) > 0:
    h = -tf.nn.sigmoid(z)
else:
    h = tf.nn.sigmoid(z)

# Evaluate immediately the output without session run.
print(h)

# tf.Tensor([[-0.36252844]], shape=(1, 1), dtype=float32)
```

Mix arrays and tensors directly

Mix python control flows with Tensorflow

No session run calls required

Eager Execution Example

- There are some other things to consider:

```python
# Train the model on batches of data with SGD.
epoch_losses = []
for i in range(0, len(x1), batch_size):
    # Build the batches.
    batch_x1 = x1[i: i + batch_size]
    batch_x2 = x2[i: i + batch_size]
    batch_y = y[i: i + batch_size]

    # The gradient tape is specific for eager execution. It keeps track
    # of all the computed outputs in the graph which will be used later
    # to compute the gradients. Note that some magic is happening.
    # Every variable initialized with `trainable=True` (default) is
    # automatically watched but other tensors can be watched, too.
    # See https://www.tensorflow.org/api_docs/python/tf/GradientTape.
    with tf.GradientTape() as tape:
        # Compute the forward pass using the batches.
        h1 = dense_layer1(batch_x1)
        h2 = dense_layer2(batch_x2)
        h = tf.concat([h1, h2], axis=-1)
        output = tf.reshape(dense_layer3(h), [-1])
        # Compute the binary cross entropy loss.
        loss = -(tf.multiply(batch_y, tf.log(output)) +
                 tf.multiply(1 - batch_y, tf.log(1 - output)))
        loss = tf.reduce_mean(loss)

    # Compute the gradients and update the variables.
    grads = tape.gradient(loss, all_params)
    for grad, v in zip(grads, all_params):
        tf.assign(v, v - learning_rate * grad)
    epoch_losses += [loss]
```

```python
class DenseLayer(object):
    """Own implementation of a dense layer.

    Parameters
    ----------
    layer_name : str
        The name of the layer, used as scope name.
    units : int
        Number of hidden units.
    input_size : int
        The size of the input.
    activation : callable or `None`, optional
        A function that computes an activation.
        If `None` no activation is used.
        Defaults to `None`.
    """

    def __init__(self, layer_name, units, input_size, activation=None):
        self.layer_name = layer_name
        self.activation = activation
        with tf.variable_scope(layer_name):
            self.weights = tf.get_variable(
                'W', dtype=tf.float32,
                shape=[input_size, units], trainable=True,
                initializer=tf.initializers.truncated_normal(
                    stddev=0.01, mean=0.0))
            self.b = tf.get_variable(
                'b', dtype=tf.float32, shape=[units], trainable=True,
                initializer=tf.constant_initializer(0.0))
```

https://github.com/dekromp/deep_learning_and_ai_tooling_lecture/tree/master/tooling_lecture/eager_execution

Eager Execution Debugging

```
import pdb  # noqa

  ⋮

# Set the breakpoint here if debugging mode is on.
if debug:
    pdb.set_trace()  # noqa

# Train the model on batches of data with SGD.
epoch_losses = []
for i in range(0, len(x1), batch_size):
    # Build the batches.
    batch_x1 = x1[i: i + batch_size]
    batch_x2 = x2[i: i + batch_size]
    batch_y = y[i: i + batch_size]

  ⋮
```

```
> /home/denis/creaidAI/development/side_projects/lecture/eager_execution/buggy_eager_tensorflow_nan_output.py(59)main()
-> batch_x2 = x2[i: i + batch_size]
(Pdb) n
> /home/denis/creaidAI/development/side_projects/lecture/eager_execution/buggy_eager_tensorflow_nan_output.py(60)main()
-> batch_y = y[i: i + batch_size]
(Pdb) n
> /home/denis/creaidAI/development/side_projects/lecture/eager_execution/buggy_eager_tensorflow_nan_output.py(69)main()
-> with tf.GradientTape() as tape:
(Pdb) n
> /home/denis/creaidAI/development/side_projects/lecture/eager_execution/buggy_eager_tensorflow_nan_output.py(71)main()
-> h1 = dense_layer1(batch_x1)
(Pdb) n
> /home/denis/creaidAI/development/side_projects/lecture/eager_execution/buggy_eager_tensorflow_nan_output.py(72)main()
-> h2 = dense_layer2(batch_x2)
(Pdb) n
> /home/denis/creaidAI/development/side_projects/lecture/eager_execution/buggy_eager_tensorflow_nan_output.py(73)main()
-> h = tf.concat([h1, h2], axis=-1)
(Pdb) n
> /home/denis/creaidAI/development/side_projects/lecture/eager_execution/buggy_eager_tensorflow_nan_output.py(74)main()
-> output = tf.reshape(dense_layer3(h), [-1])
(Pdb) n
> /home/denis/creaidAI/development/side_projects/lecture/eager_execution/buggy_eager_tensorflow_nan_output.py(76)main()
-> loss = -(tf.multiply(batch_y, tf.log(output)) +
(Pdb) n
> /home/denis/creaidAI/development/side_projects/lecture/eager_execution/buggy_eager_tensorflow_nan_output.py(77)main()
-> tf.multiply(1 - batch_y, tf.log(1 - output)))
(Pdb) n
> /home/denis/creaidAI/development/side_projects/lecture/eager_execution/buggy_eager_tensorflow_nan_output.py(78)main()
-> loss = tf.reduce_mean(loss)
(Pdb) n
> /home/denis/creaidAI/development/side_projects/lecture/eager_execution/buggy_eager_tensorflow_nan_output.py(81)main()
-> grads = tape.gradient(loss, all_params)
(Pdb) p loss
<tf.Tensor: id=103, shape=(), dtype=float32, numpy=nan>
(Pdb)
```

You can use your standard python debugging routine!

https://github.com/dekromp/deep_learning_and_ai_tooling_lecture/tree/master/tooling_lecture/eager_execution

Eager Execution

Write compatible code

The same code written for eager execution will also build a graph during graph execution. Do this by simply running the same code in a new Python session where eager execution is not enabled.

Most TensorFlow operations work during eager execution, but there are some things to keep in mind:

- Use `tf.data` for input processing instead of queues. It's faster and easier.
- Use object-oriented layer APIs—like `tf.keras.layers` and `tf.keras.Model`—since they have explicit storage for variables.
- Most model code works the same during eager and graph execution, but there are exceptions. (For example, dynamic models using Python control flow to change the computation based on inputs.)
- Once eager execution is enabled with `tf.enable_eager_execution`, it cannot be turned off. Start a new Python session to return to graph execution.

It's best to write code for both eager execution *and* graph execution. This gives you eager's interactive experimentation and debuggability with the distributed performance benefits of graph execution.

Write, debug, and iterate in eager execution, then import the model graph for production deployment. Use `tf.train.Checkpoint` to save and restore model variables, this allows movement between eager and graph execution environments. See the examples in: tensorflow/contrib/eager/python/examples.

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

Eager Execution

```python
"""This script shows a simple example on how eager execution works."""
import numpy as np
import tensorflow as tf


# Enable eager execution.
tf.enable_eager_execution()

# Make the execution reproducable.
tf.set_random_seed(2132)
np.random.seed(3423)

# Generate some random data.
x = np.arange(3).reshape(-1, 1).astype(np.float32)
w = tf.get_variable(
    'w', dtype=np.float32, shape=[1, 3],
    initializer=tf.glorot_uniform_initializer())

# Interwine python and tensorflow code directly.
z = tf.matmul(w, x)
if np.sum(x) > 0:
    h = -tf.nn.sigmoid(z)
else:
    h = tf.nn.sigmoid(z)

# Evaluate immediately the output without session run.
print(h)

# tf.Tensor([[-0.36252844]], shape=(1, 1), dtype=float32)

```

Mix arrays and tensors directly

~~Mix python control flows with Tensorflow~~

Only for static graphs

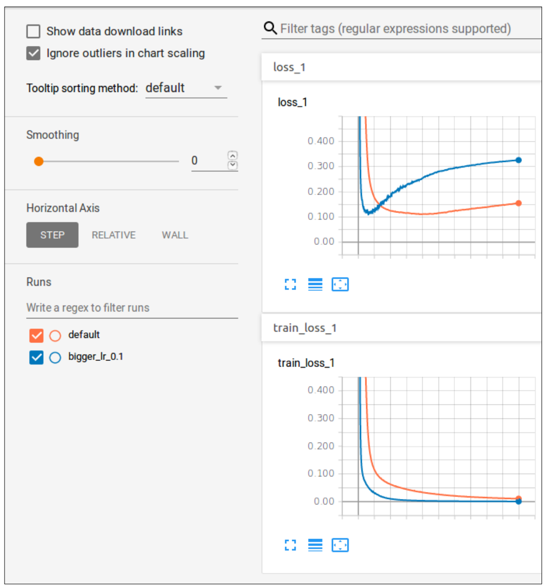
No session run calls required

https://github.com/dekromp/deep_learning_and_ai_tooling_lecture/tree/master/tooling_lecture/eager_execution

Tensorboard

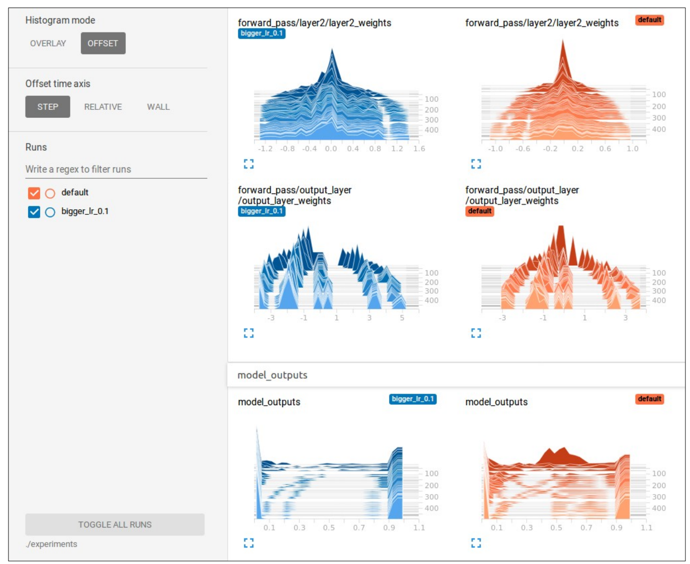
Scalars

```
# Monitor the loss.
tf.summary.scalar('loss', loss)
```



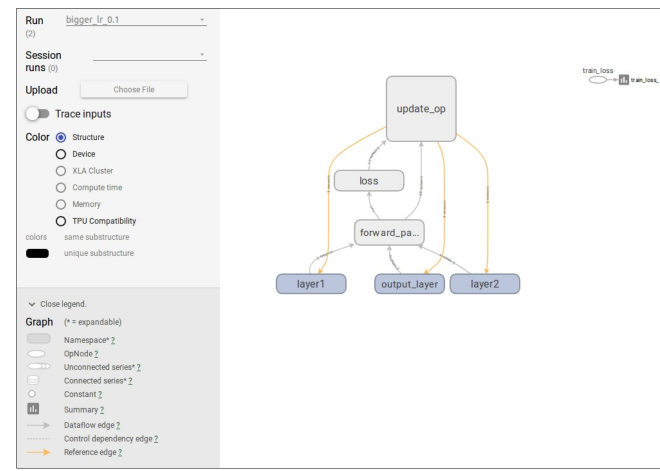
Histograms

```
# Compute a historgram over the outputs.
tf.summary.histogram('model_outputs', output)
```



Graph

```
# Add the graph to the summaries.
summary_writer.add_graph(session.graph)
```

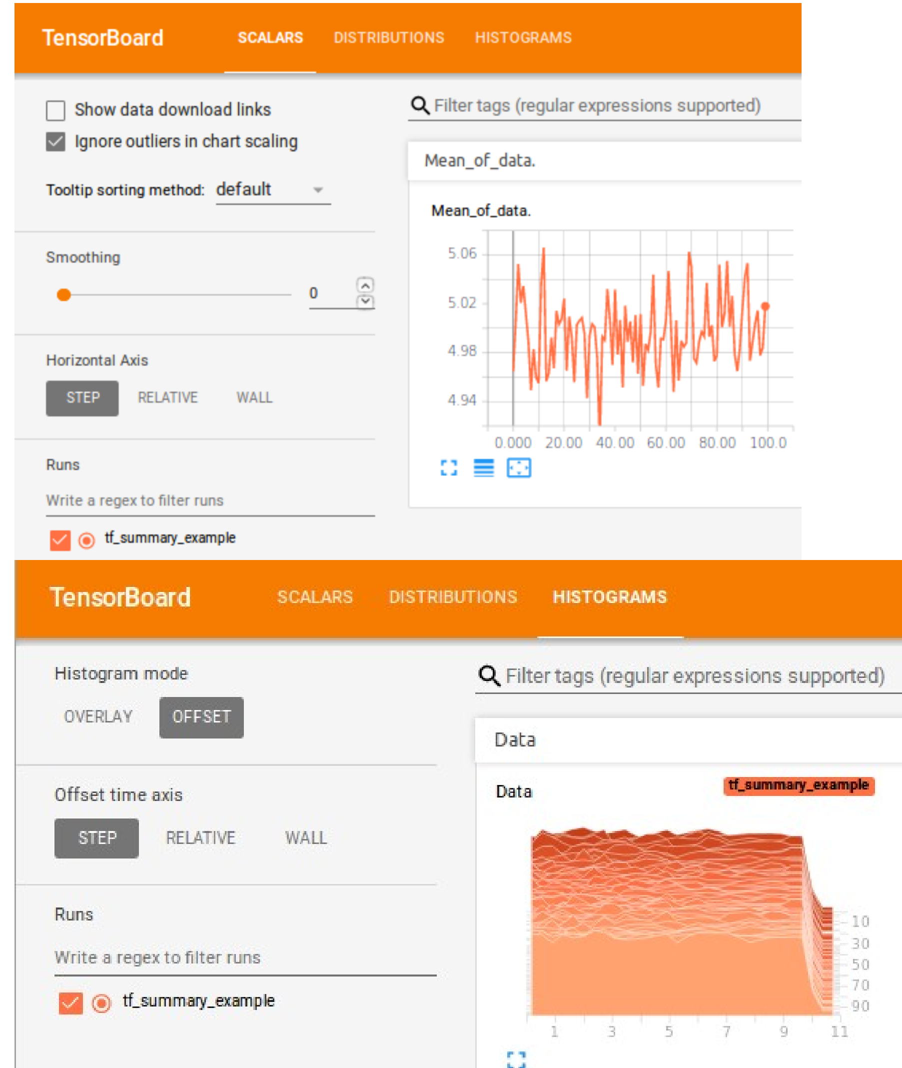


https://www.tensorflow.org/api_docs/python/tf/summary

https://github.com/dekromp/deep_learning_and_ai_tooling_lecture/tree/master/tooling_lecture/tensorboard

Tensorboard

```python
"""Minimal example that shows how summaries are used.

Observe the result in the web browser (localhost:6006) after starting the
tensorboard:

$ tensorboard --logdir=./experiments/
"""
import numpy as np
import tensorflow as tf


np.random.seed(12123)


# Create a placeholder for our data.
input_data = tf.placeholder(tf.float32, [1000, 10])

# Create some summaries.
tf.summary.scalar('Mean of data.', tf.reduce_mean(input_data))
tf.summary.histogram('Data', input_data)

# Merge all summaries. <- tensorflow magic op.
all_summaries_op = tf.summary.merge_all()

# Create a writer for storing the summaries on disk for tensorboard to find.
summary_writer = tf.summary.FileWriter('./experiments/tf_summary_example')

# Let's create some summary events.
with tf.Session() as session:
    for step in range(100):
        # Generate some random data.
        data = np.random.uniform(0, 10, size=[1000, 10])

        # Compute the summary values.
        all_summaries = session.run(
            all_summaries_op, feed_dict={input_data: data})

        # Write the summaries, dont forget the step ('x-coordinate' in plot).
        summary_writer.add_summary(all_summaries, step)
```



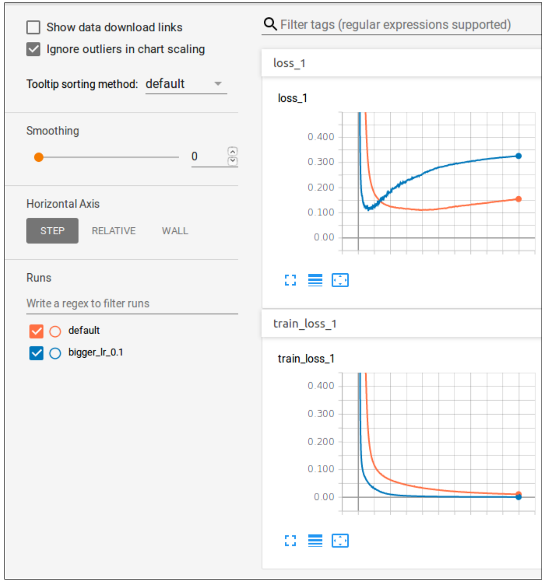
https://github.com/dekromp/deep_learning_and_ai_tooling_lecture/tree/master/tooling_lecture/tensorboard

Tensorboard

Tensorboard is mostly used for **monitoring the training**, not for evaluating the model.
It is an additional tool for debugging.

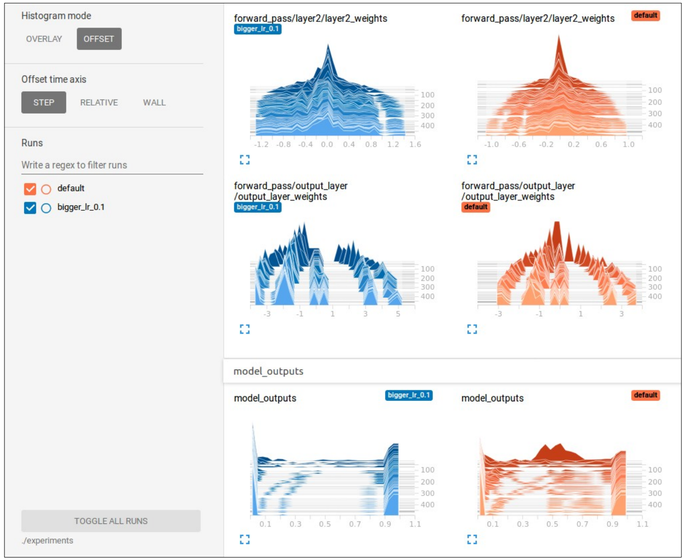
Visualize / compare learning curves

```
# Monitor the loss.
tf.summary.scalar('loss', loss)
```



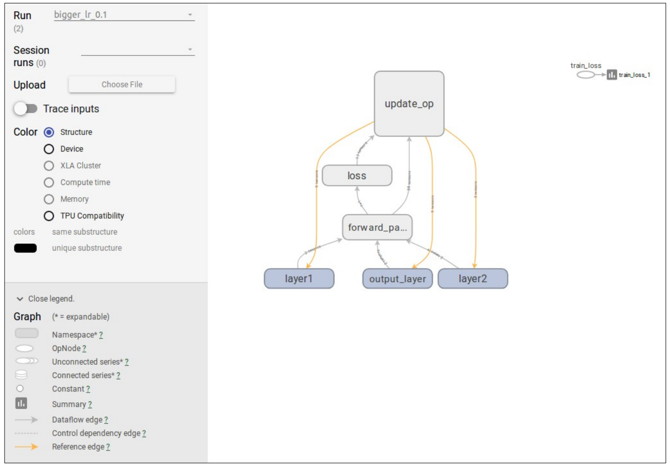
Visualize how parameters and outputs are evolving

```
# Compute a historgram over the outputs.
tf.summary.histogram('model_outputs', output)
```



Visualize the computation graph
(Use scopes and names)

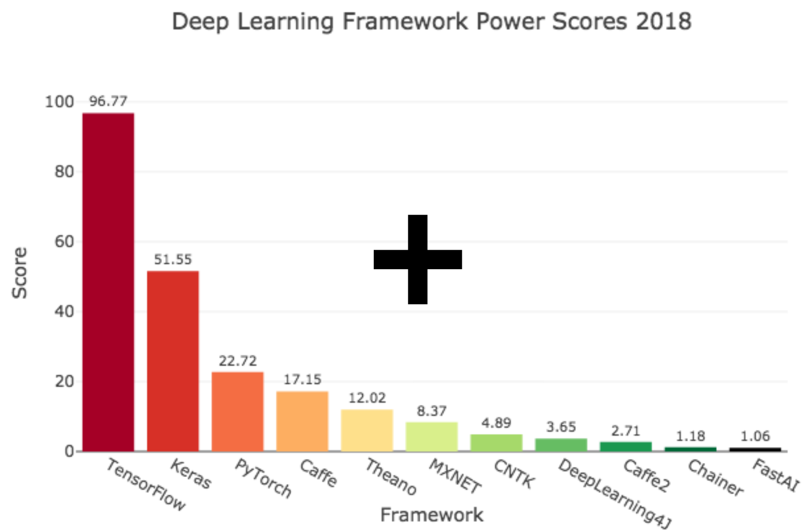
```
# Add the graph to the summaries.
summary_writer.add_graph(session.graph)
```



https://github.com/dekromp/deep_learning_and_ai_tooling_lecture/tree/master/tooling_lecture/tensorboard

Frameworks

TensorFlow



Keras

Frameworks

Don't worry about numerical stability

output = tf.nn.sigmoid(x)

```
def sigmoid(x):
    """Sigmoid activation function.

    Parameters
    ----------
    x : :class:`tf.tensor`
        The input to this op.

    Returns
    -------
    activated : :class:`tf.tensor`
        The activated input.

    """
    # Make sure that the values of x are not too small/big.
    x = tf.clip_by_value(x, -80, 80)

    negative = tf.less(x, 0.0)
    activation = tf.where(
        negative, tf.exp(x) / (1.0 + tf.exp(x)), 1.0 / (1.0 + tf.exp(-x)))
    return activation
```

loss = tf.losses.log_loss(x)

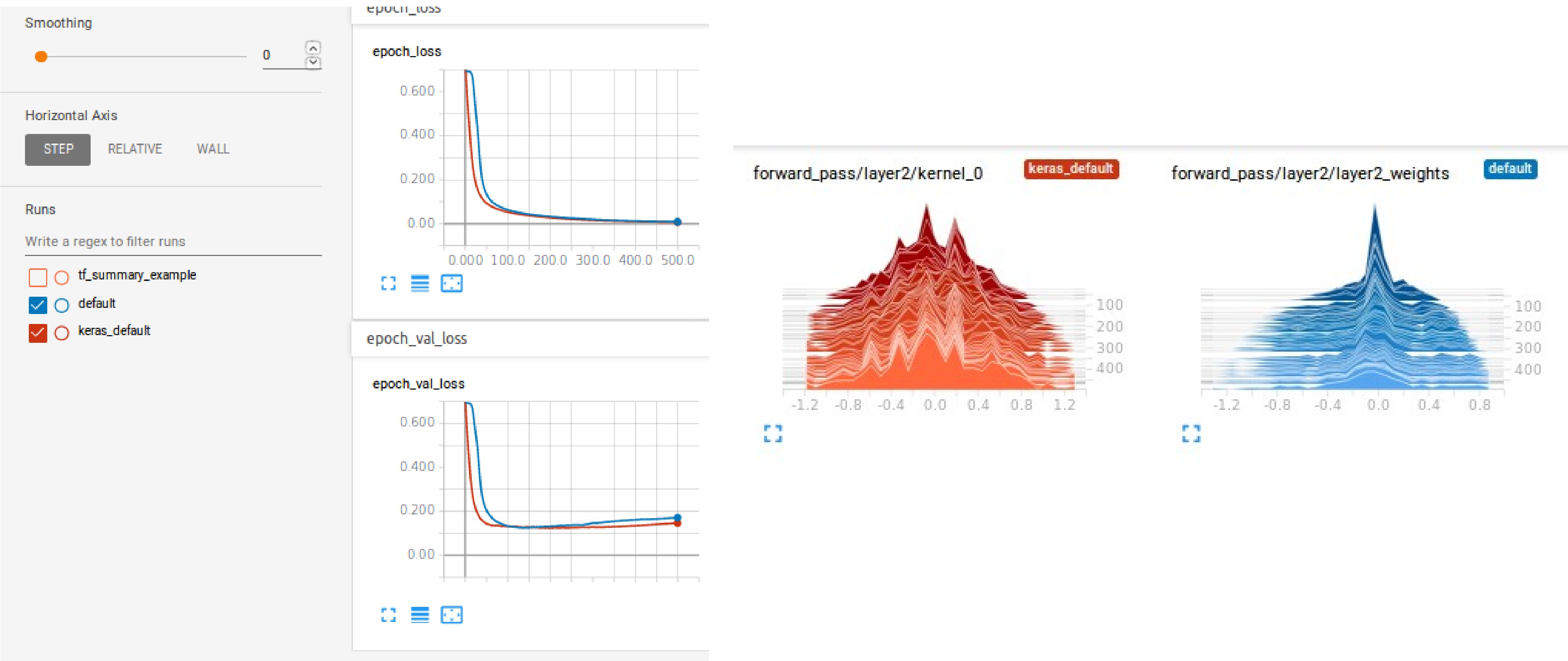
```
# Compute the loss (binary cross entropy)
epsilon = 1e-7  # for numerical stability.
loss = -(tf.multiply(input_y, tf.log(output + epsilon)) +
         tf.multiply(1 - input_y, tf.log(1 - output + epsilon)))
loss = tf.reduce_mean(loss, name='loss_out')
```

https://github.com/dekromp/deep_learning_and_ai_tooling_lecture/tree/master/tooling_lecture/frameworks

Frameworks

Worry less about best practices

- Here: Initialization of model parameters

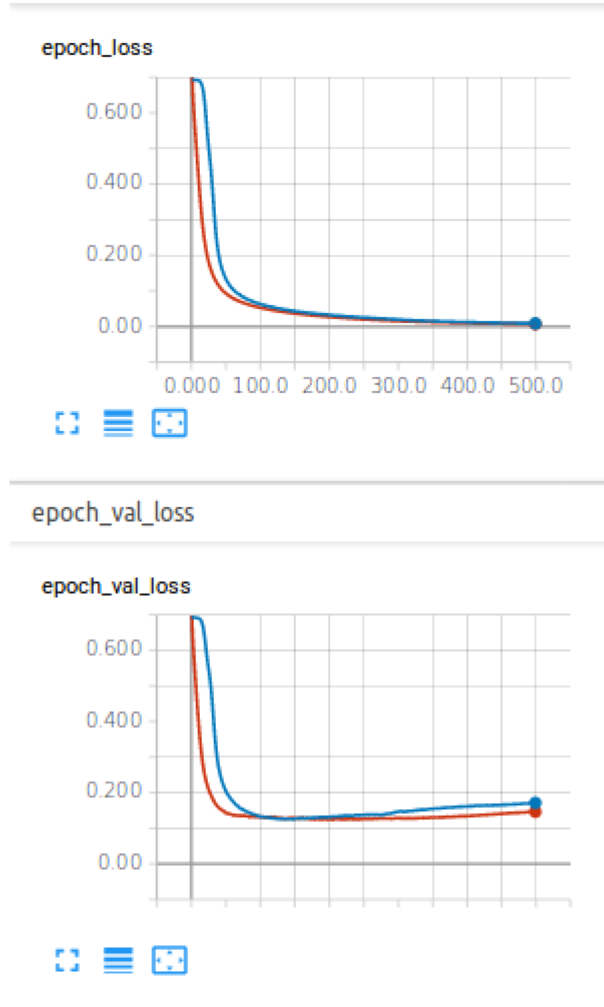


Frameworks

Lots of convenience

Tensorflow from scratch

371 lines



tf.keras + sklearn

133 lines

https://github.com/dekromp/deep_learning_and_ai_tooling_lecture/tree/master/tooling_lecture/frameworks

Frameworks

Lots of convenience

- Keras Model API is a powerful tool for prototyping models quickly.
- Additional features are already implemented (layers, Tensorboard summaries, ...)

https://www.tensorflow.org/api_docs/python/tf/keras or https://keras.io/getting-started/functional-api-guide/

```python
def build_forward_pass():
    """Build the forward pass of the model.

    Returns
    -------
    input_x1 : :class:`tf.tensor`
        The input for the first feature set.
    input_x2 : :class:`tf.tensor`
        The input for the second feature set.
    output : :class:`tf.tensor`
        The output of the model.

    """
    with tf.name_scope('forward_pass'):
        input_x1 = tf.keras.Input([10], name='input_x1')
        input_x2 = tf.keras.Input([20], name='input_x2')
        h1 = tf.keras.layers.Dense(
            units=5, activation=tf.nn.relu, name='layer1')(input_x1)
        h2 = tf.keras.layers.Dense(
            units=7, activation=tf.nn.relu, name='layer2')(input_x2)
        h = tf.keras.layers.Concatenate(axis=-1)([h1, h2])
        output = tf.keras.layers.Dense(
            units=1, activation=tf.nn.sigmoid, name='output_layer')(h)

    return input_x1, input_x2, output
```

```python
def main(num_epochs, batch_size, learning_rate, experiment_dir, debug):
    """Train a simple model on random data.

    Parameters
    ----------
    num_epochs : int
        The number of epochs the model is trained.
    batch_size : int
        The batch size used for SGD.
    learning_rate : float
        The learning rate used for SGD.
    experiment_dir : str
        The path to the experiment directory where the summaries will be saved.
    debug : bool
        Whether or not the script is debugged with the tensorflow debugger.

    """
    # Create some random data.
    dataset = load_data()
    x1, x1_val, x2, x2_val, y, y_val = train_test_split(
        *dataset, test_size=0.1)

    # Build forward pass through the network.
    input_x1, input_x2, output = build_forward_pass()

    # Build the model with keras.
    model = tf.keras.Model([input_x1, input_x2], [output])
    # Build loss and the update operations.
    optimizer = tf.keras.optimizers.SGD(lr=learning_rate)
    model.compile(optimizer, loss=tf.keras.losses.binary_crossentropy)

    # Define some callbacks.
    tensorboard_callback = tf.keras.callbacks.TensorBoard(
        experiment_dir, write_graph=True, write_images=True,
        histogram_freq=1)
    callbacks = [tensorboard_callback]

    # Train the model on the data.
    model.fit(x=[x1, x2], y=y, batch_size=batch_size, epochs=num_epochs,
              validation_data=([x1_val, x2_val], y_val), verbose=2,
              callbacks=callbacks)
```

Frameworks

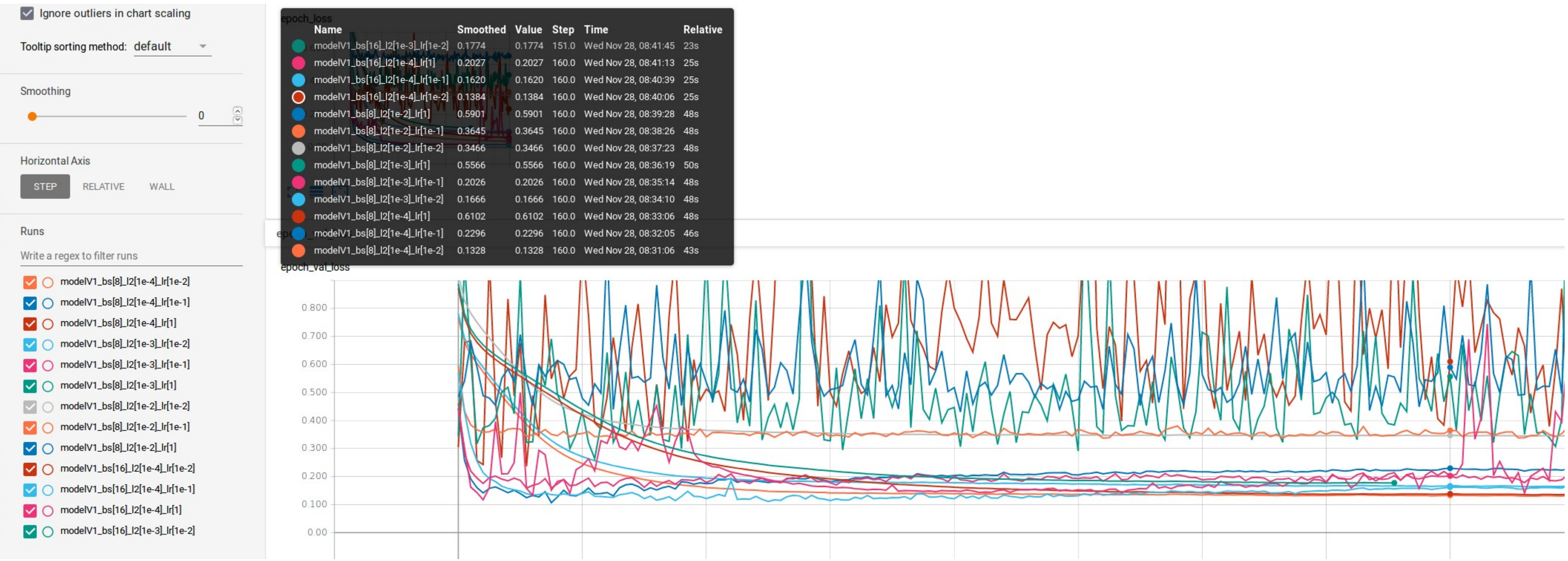
Limits

- Often, a high level framework does not contain all the required features or is not flexible enough:
  - Fall back to Tensorflow
    - Many convenience functions from tf.keras like layers can be reused.
  - Use the framework differently.
    - Maybe you need multiple models? (GANs)
  - Write own extensions for framework.
    - Many things like custom losses, layers and models can be easily implemented.
  - Built-in ways to extend functionalities:

```
output = tf.keras.layers.Lambda(lambda x: tf.expand_dims(x, -1))(input_x)
```

TUNE

Hyperparameter Tuning



https://github.com/dekromp/deep_learning_and_ai_tooling_lecture/tree/master/tooling_lecture/hypertune

Hyperparameter Tuning

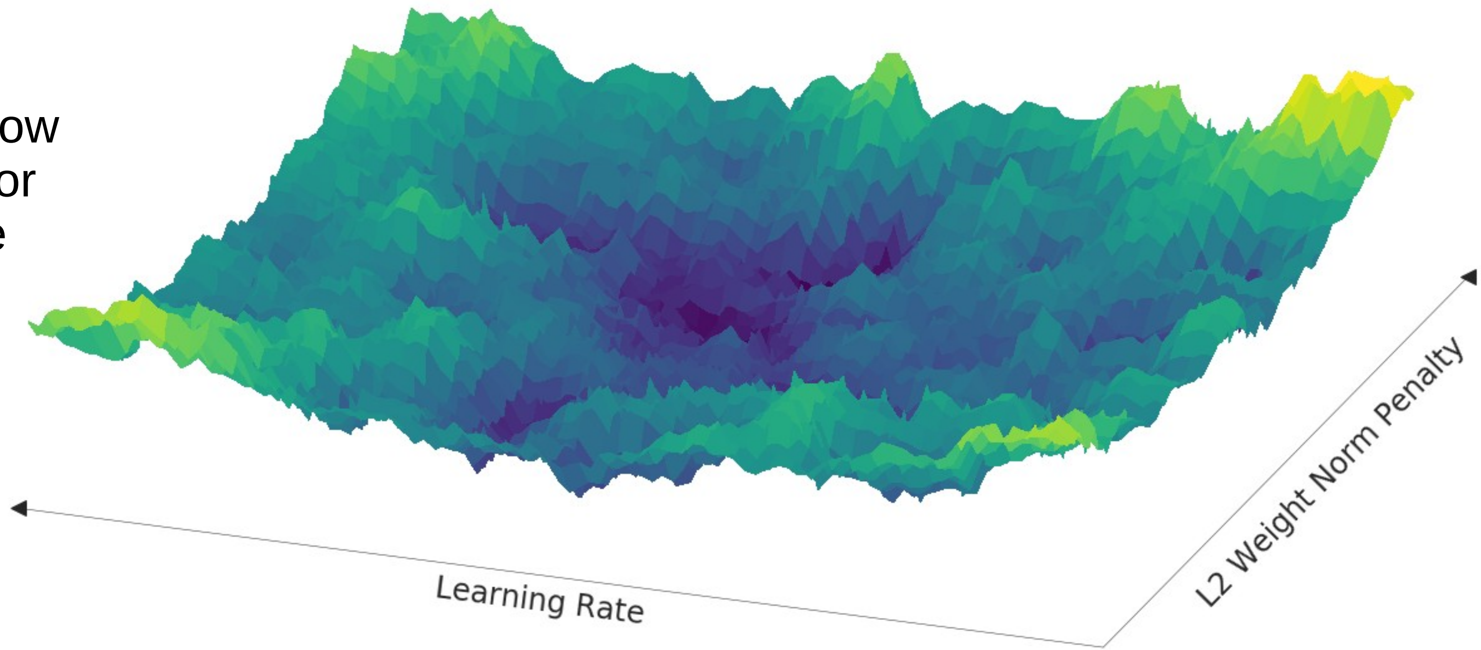
General Procedure

- Get an idea what you are actually after
  - Run a couple of test experiments
  - Measure everything that seems useful to judge the performance manually
  - In the ideal case, find a single measure (could be your own) that frames good models.
- Roll out large scale experiments
  - Use your measures to filter the runs
  - Evaluate the best candidates
- Draw conclusions and repeat/refine

Hyperparameter Tuning

Methods

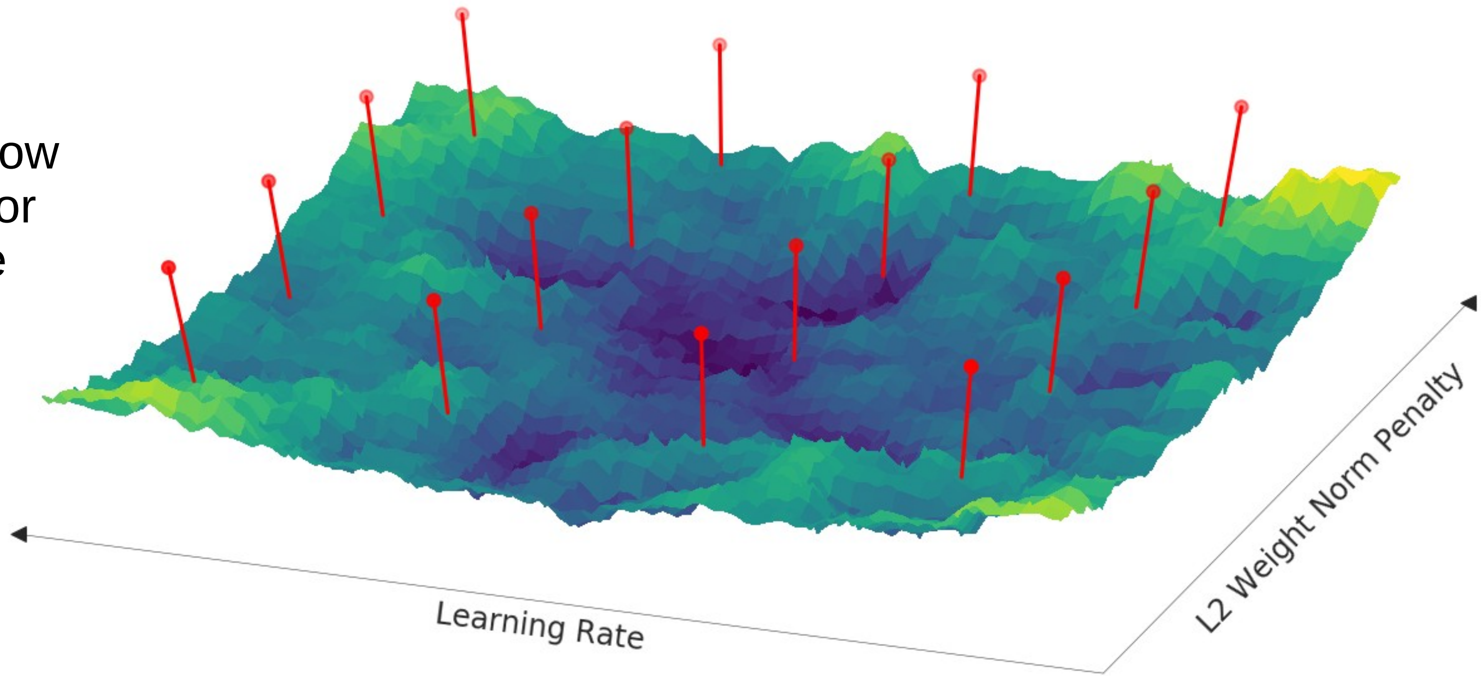
You don't know the true error landscape



Hyperparameter Tuning

Methods - GridSearch

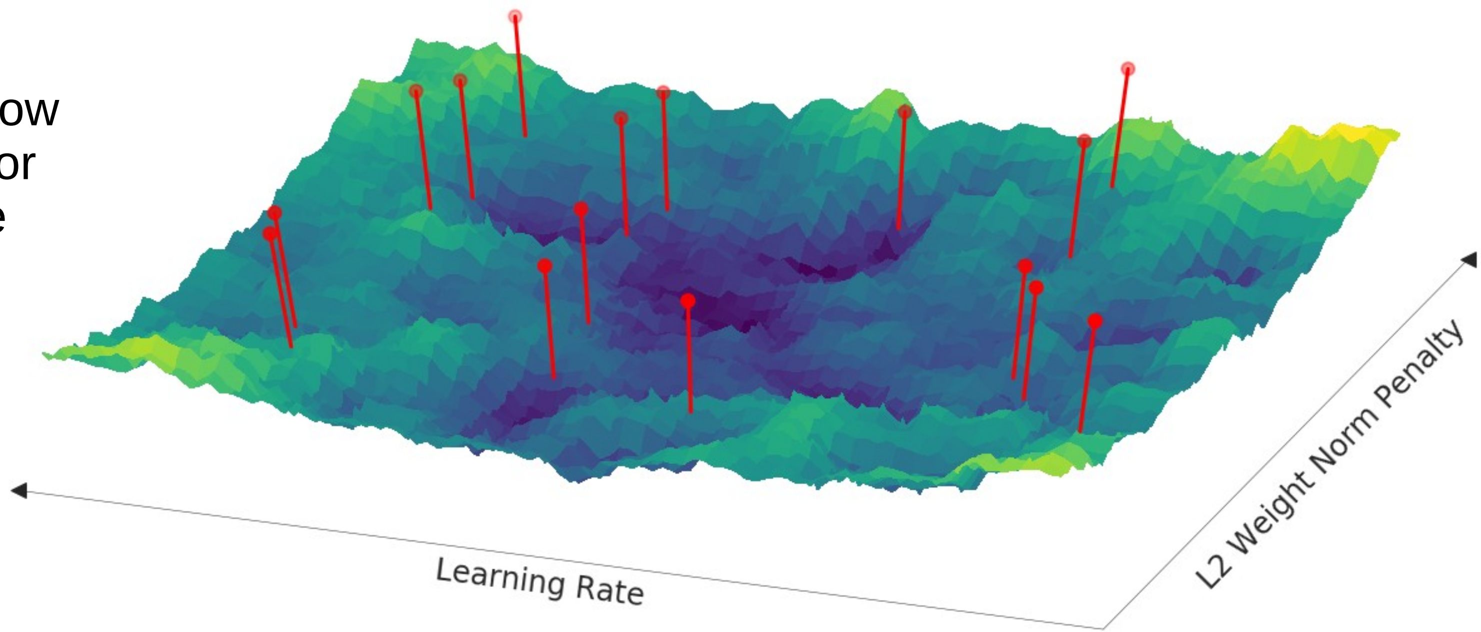
You don't know the true error landscape



Hyperparameter Tuning

Methods – Random Search

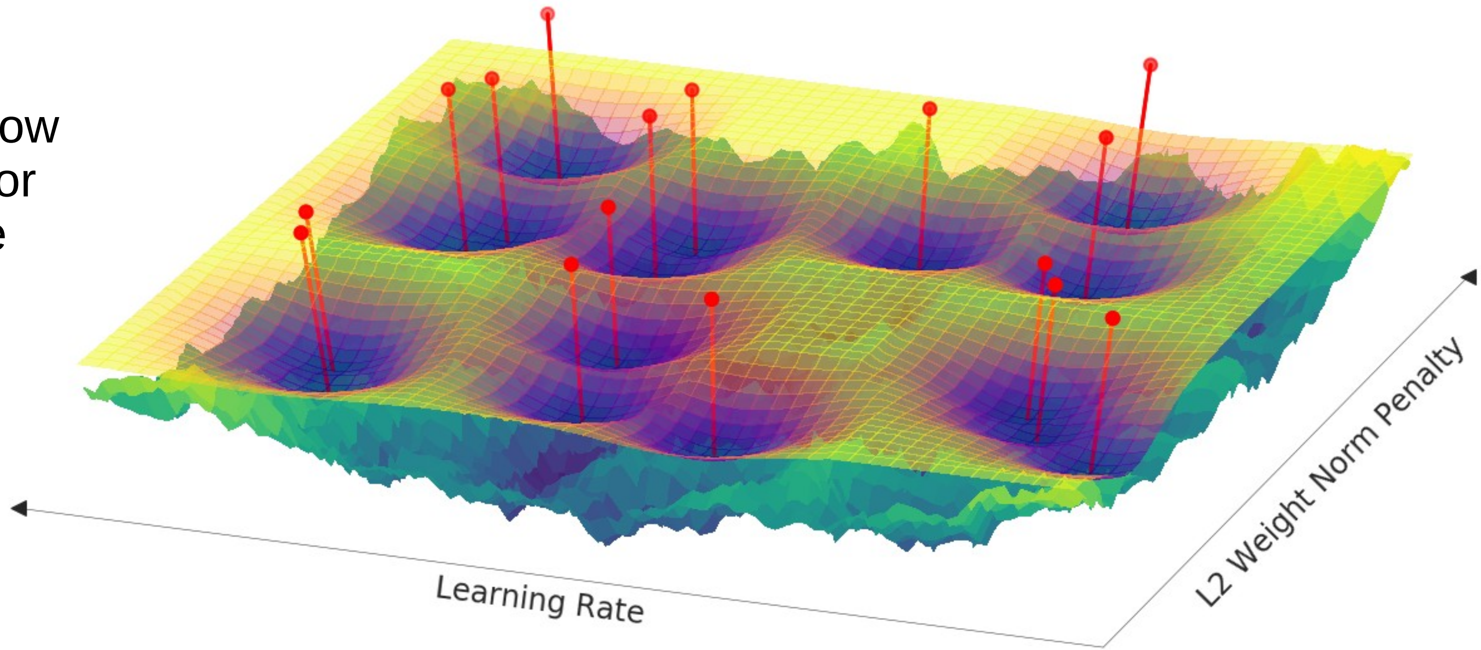
You don't know the true error landscape



Hyperparameter Tuning

Methods- Bayesian Model Optimization

You don't know the true error landscape

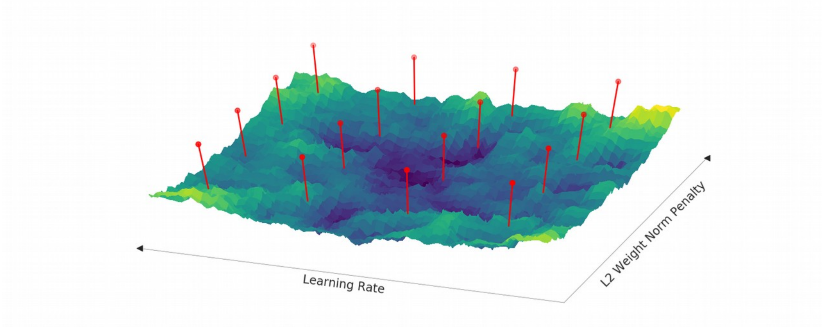


Intuition: Automate hyperparameter search by automatically choosing most promising candidates based on past experience.

Hyperparameter Tuning

The naive way

```python
"""Simple way to call python scripts."""
import os
import subprocess


num_epochs = 200
batch_size = 8
path = './experiments'
try:
    for l2_factor in [1e-4, 1e-3, 1e-2]:
        for learning_rate in [1e-2, 1e-1, 1]:
            subprocess.call([
                'python', 'keras_example.py',
                '-ep', str(num_epochs),
                '-bs', str(batch_size),
                '-lr', str(learning_rate),
                '-l2', str(l2_factor),
                '-d', os.path.join(
                    path, 'modelV1_lr[%s]_l2[%s]'
                    % (learning_rate, l2_factor))])
except subprocess.CalledProcessError as e:
    print(e)
```



Experiment runs are automated but they are executed sequentially.

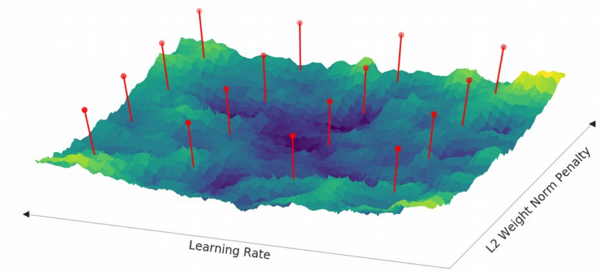
```
(dev-p36)                                          hypertune(master)$ python naive_hypertune.py
keras_example.py -ep 200 -bs 8 -lr 0.01 -l2 0.0001 -d ./experiments/modelV1_lr[0.01]_l2[0.0001]
keras_example.py -ep 200 -bs 8 -lr 0.1 -l2 0.0001 -d ./experiments/modelV1_lr[0.1]_l2[0.0001]
keras_example.py -ep 200 -bs 8 -lr 1 -l2 0.0001 -d ./experiments/modelV1_lr[1]_l2[0.0001]
keras_example.py -ep 200 -bs 8 -lr 0.01 -l2 0.001 -d ./experiments/modelV1_lr[0.01]_l2[0.001]
keras_example.py -ep 200 -bs 8 -lr 0.1 -l2 0.001 -d ./experiments/modelV1_lr[0.1]_l2[0.001]
keras_example.py -ep 200 -bs 8 -lr 1 -l2 0.001 -d ./experiments/modelV1_lr[1]_l2[0.001]
keras_example.py -ep 200 -bs 8 -lr 0.01 -l2 0.01 -d ./experiments/modelV1_lr[0.01]_l2[0.01]
keras_example.py -ep 200 -bs 8 -lr 0.1 -l2 0.01 -d ./experiments/modelV1_lr[0.1]_l2[0.01]
keras_example.py -ep 200 -bs 8 -lr 1 -l2 0.01 -d ./experiments/modelV1_lr[1]_l2[0.01]
```

https://github.com/dekromp/deep_learning_and_ai_tooling_lecture/tree/master/tooling_lecture/hypertune

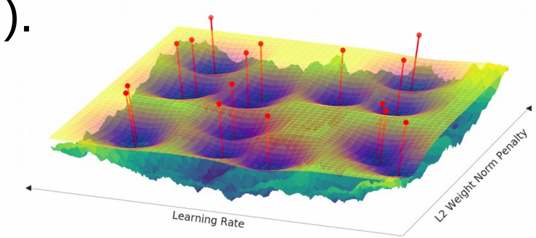
Hyperparameter Tuning

Parallel (distributed) execution

- **ray** is very generic system for parallel and distributed Python
  - Can also be used for distributed execution in computing cluster.
  - Easy to setup (just pip install)
  - **ray.tune** contains implementations for more advanced hyperparameter tuning methods (requires integration into API).




```python
"""Ray can be used to parallelize our grid search implementation."""
import ray

from tooling_lecture.hypertune import keras_example


ray.init()


@ray.remote
def keras_example_main(num_epochs, batch_size, l2_factor, learning_rate,
                       experiment_dir):
    """A wrapper for the main function of the keras example."""
    keras_example.main(
        num_epochs, batch_size, l2_factor, learning_rate, experiment_dir,
        debug=False)


object_ids = []
for i, l2_factor in enumerate([1e-5, 1e-4, 1e-3]):
    for learning_rate in [1e-2, 1e-1, 1]:
        object_ids += [keras_example_main.remote(
            200, 8, l2_factor, learning_rate,
            experiment_dir=(
                './experiments/ray_modelV1_lr[%s]_l2[%s]'
                % (learning_rate, l2_factor)))]
        print(object_ids[-1])


# To make sure that the ray stays alive until all processes are finished.
ray.get(object_ids)
```



https://ray.readthedocs.io/en/latest/index.html

https://github.com/dekromp/deep_learning_and_ai_tooling_lecture/tree/master/tooling_lecture/hypertune

Hyperparameter Tuning

Cloud Solutions

- Cloud solutions have the advantage that you do not have to manage the infrastructure.
- There exists a couple of cloud offerings that enable you to perform hyperparameter tuning on managed infrastructure:

But there are also newcomers:

Hyperparameter Tuning

Cloud Solutions- Simple Approach

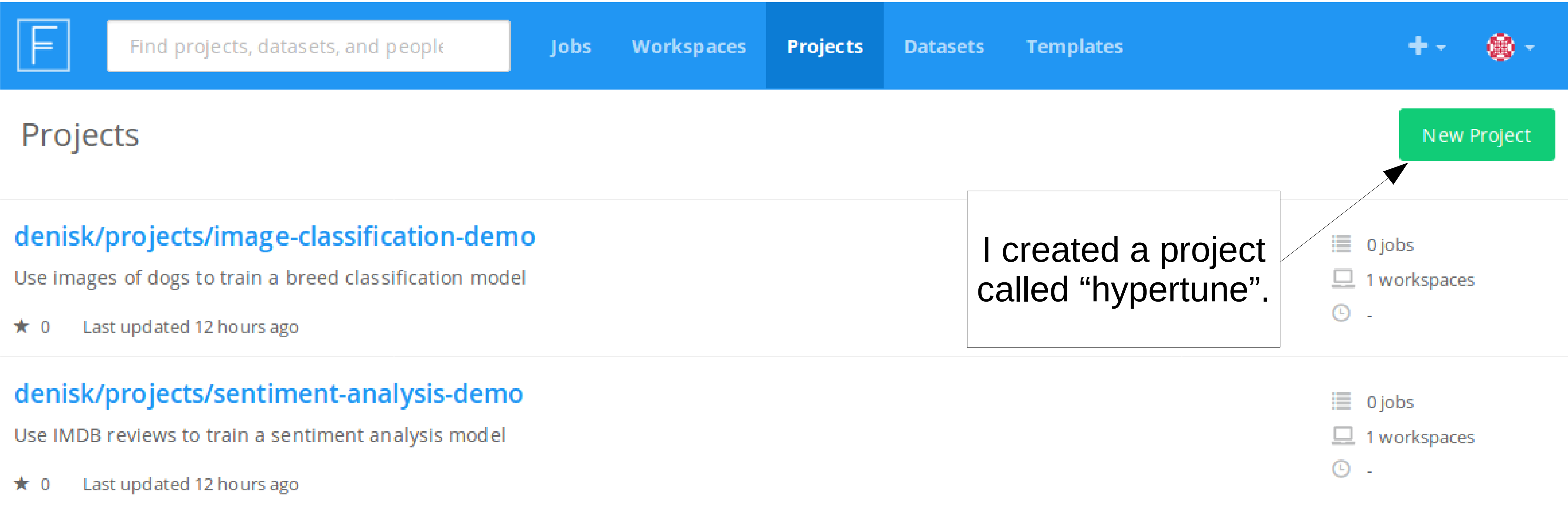
- All cloud solutions offer the possibility to submit an (infinite) amount of jobs and retrieve the results.
- Using for example **FloydHub** for this is fairly easy:
  - Register at floydhub (free)
  - Download cli tool with pip:
    - **$ pip install floyd-cli**
  - Create a project in the web ui.
  - Connect your local files with (you can use .floydignore to exclude files from synching)
    - **$ floyd init <your project name>**

You can extend this.

  - Execute your command with:
    - **$ floyd run –env tensorflow-1.11 “<command>”**
  - Your scripts get uploaded (Max 100 MB) and the command gets scheduled (dockerized).
  - Download the results (via the WebUI or via the CLI)
  - If your job requires data, you should create a dataset with floydhubs API beforehand.

Hyperparameter Tuning

Cloud Solutions- Simple Approach

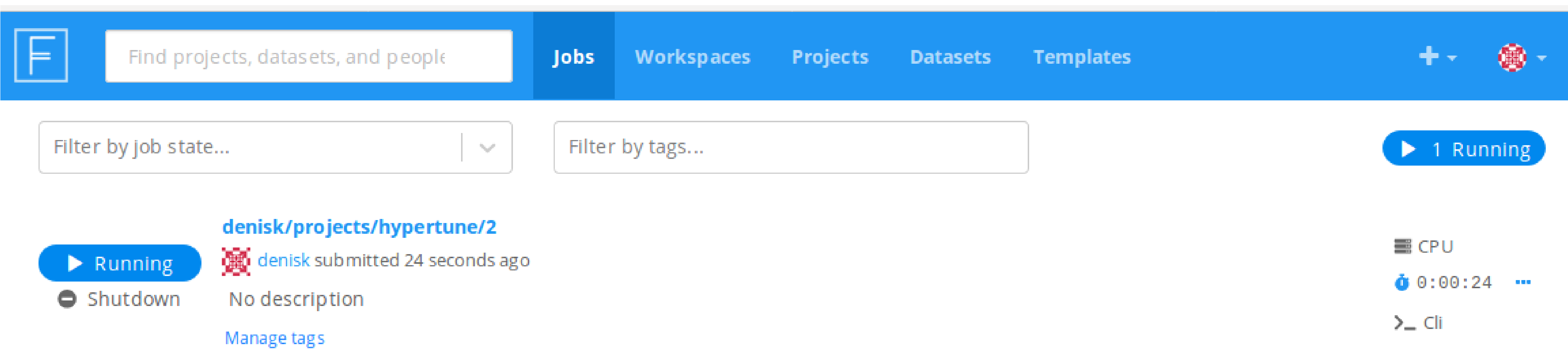


https://github.com/dekromp/deep_learning_and_ai_tooling_lecture/tree/master/tooling_lecture/hypertune

Hyperparameter Tuning

Cloud Solutions- Simple Approach

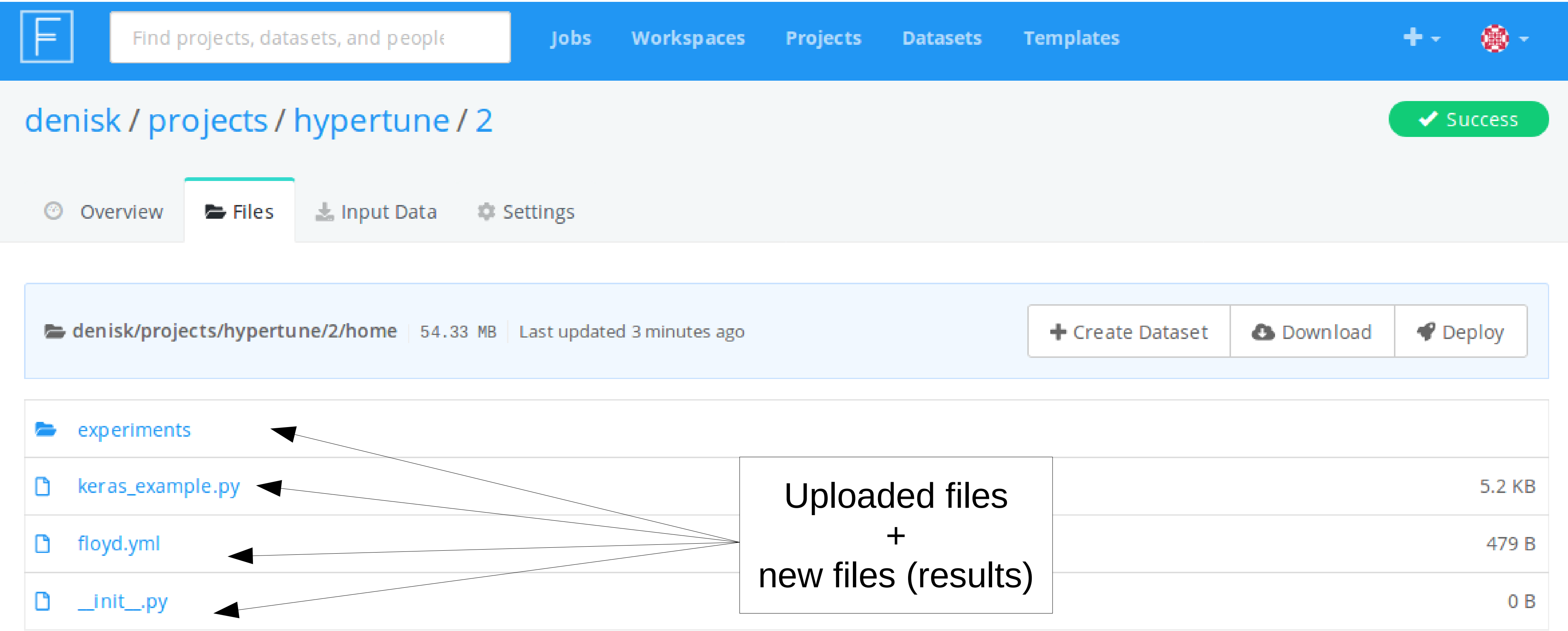
```
(dev-p36) [redacted]/hypertune(master)$ floyd login
Waiting for login from browser...
Login Successful as denisk
(dev-p36) [redacted]/hypertune(master)$ floyd init hypertune
Project "hypertune" initialized in current directory
(dev-p36) [redacted]/hypertune(master)$ floyd run --env tensorflow-1.11 "python keras_example.py -v 2"
Creating project run. Total upload size: 5.9KiB
Syncing code ...
[================================] 7878/7878 - 00:00:00

JOB NAME
--------------------------
denisk/projects/hypertune/1

URL to job: https://www.floydhub.com/denisk/projects/hypertune/1

To view logs enter:
   floyd logs denisk/projects/hypertune/1
```



https://github.com/dekromp/deep_learning_and_ai_tooling_lecture/tree/master/tooling_lecture/hypertune

Hyperparameter Tuning

Cloud Solutions- Simple Approach



https://github.com/dekromp/deep_learning_and_ai_tooling_lecture/tree/master/tooling_lecture/hypertune

Hyperparameter Tuning

Cloud Solutions- Advanced Approach

- Cloud solutions have the advantage that you do not have to manage the infrastructure.
- There exists a couple of cloud offerings that enable you to perform hyperparameter tuning on managed infrastructure:

Google Cloud Platform	amazon web services	Azure
Cloud ML Engine	**SageMaker**	**Machine Learning service**
Python SDK	**Python SDK**	**Python SDK**

Better offering but much more labor intensive to get something (custom) running.

AutoML

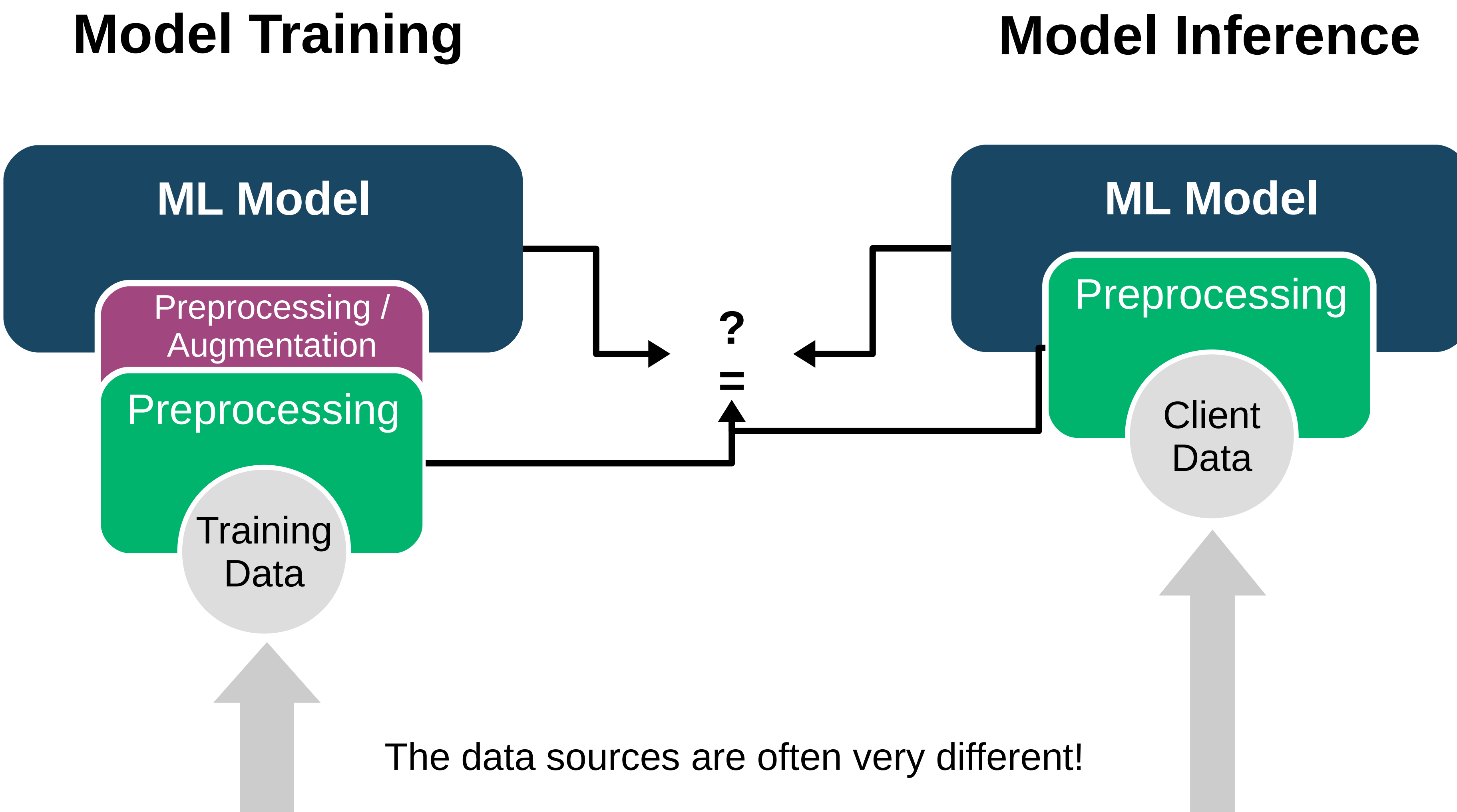
Don’t bet on it just yet.

We are far away from automating Machine Learning.

https://hackernoon.com/a-brief-overview-of-automatic-machine-learning-solutions-automl-2826c7807a2a

DEPLOY

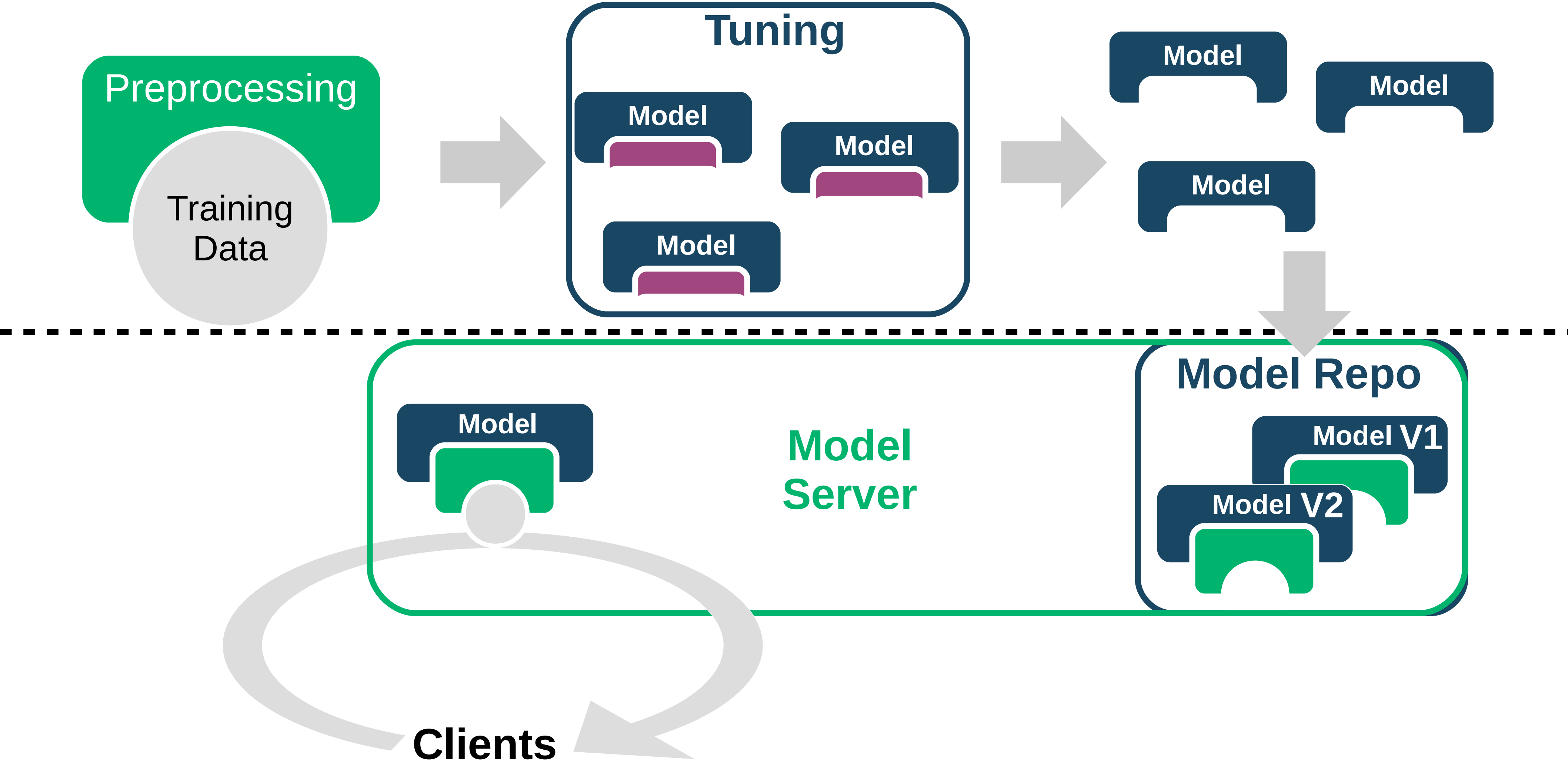
Deployment



The data sources are often very different!

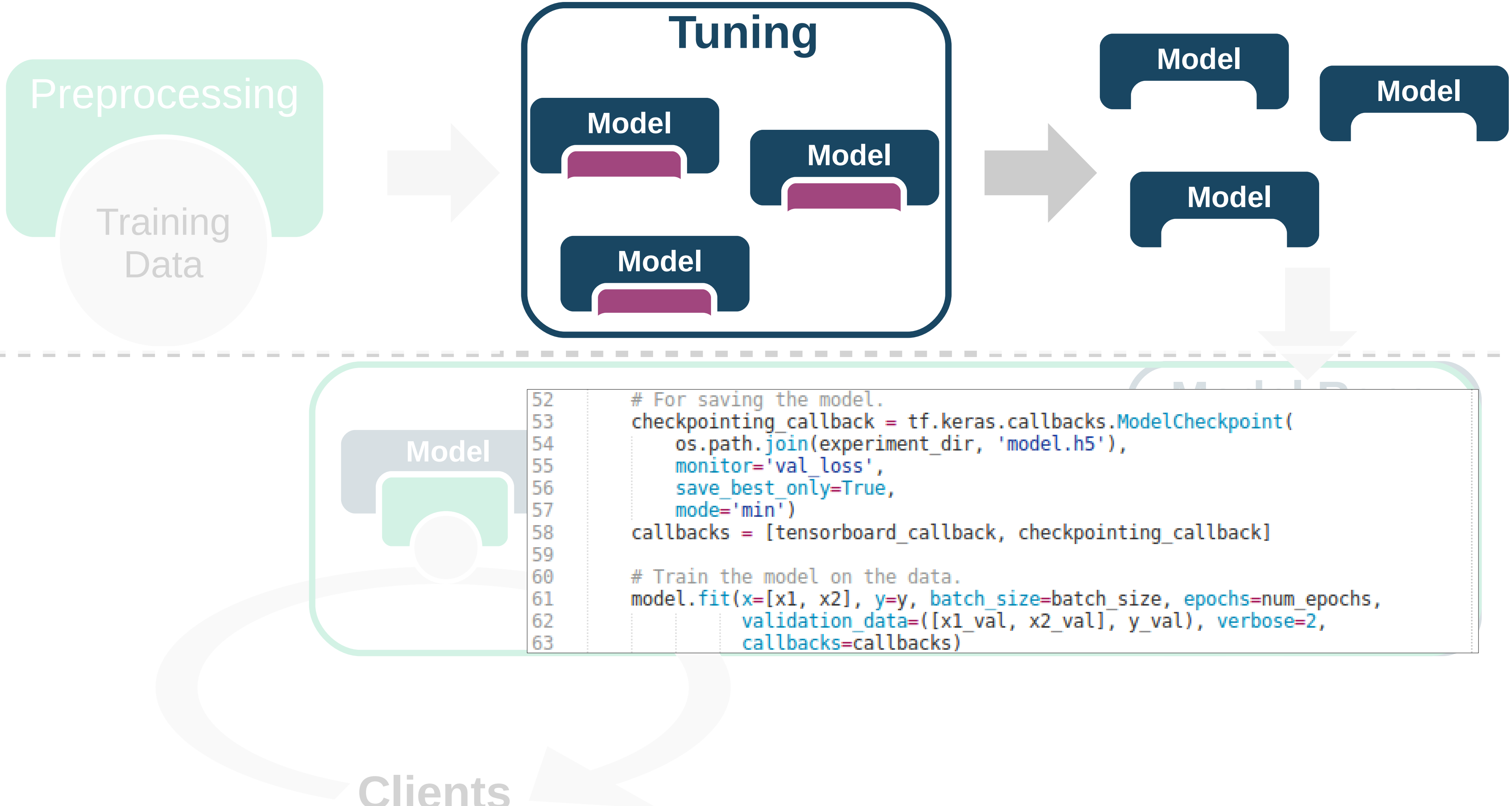
Deployment

Model Serving



Deployment

Model Serving – Tensorflow Serving

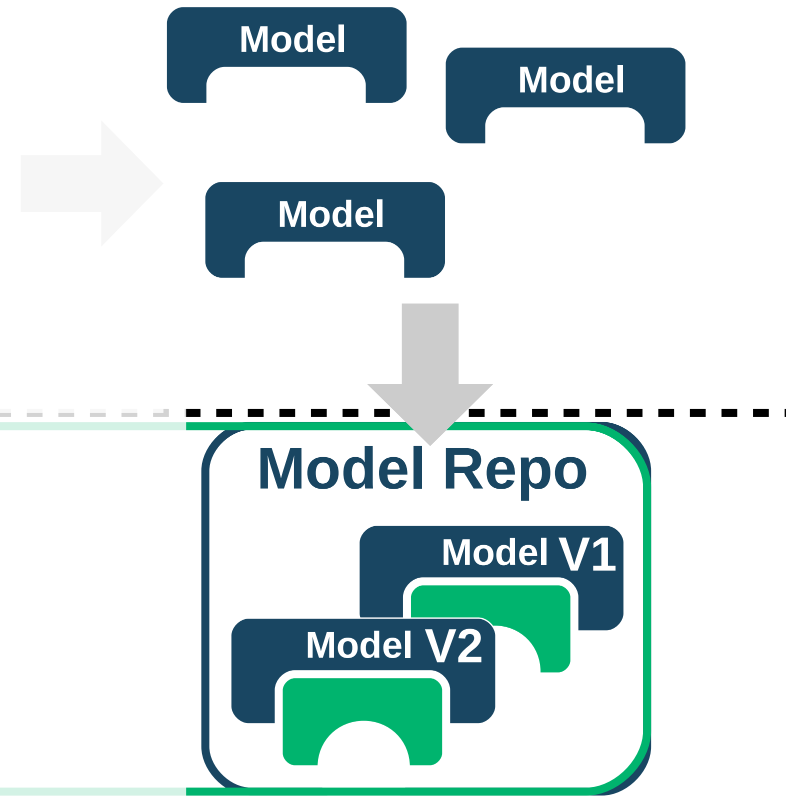



```python
    # For saving the model.
    checkpointing_callback = tf.keras.callbacks.ModelCheckpoint(
        os.path.join(experiment_dir, 'model.h5'),
        monitor='val_loss',
        save_best_only=True,
        mode='min')
    callbacks = [tensorboard_callback, checkpointing_callback]

    # Train the model on the data.
    model.fit(x=[x1, x2], y=y, batch_size=batch_size, epochs=num_epochs,
              validation_data=([x1_val, x2_val], y_val), verbose=2,
              callbacks=callbacks)
```

Deployment

Model Serving – Tensorflow Serving

```python
"""Load a keras model and save it in a format understood by tf serving."""
import tensorflow as tf


# Load the keras model from disc.
model = tf.keras.models.load_model(
    './experiments/keras_example/model.h5')

# Set the export path. Tensorflow serving takes the last directory name as the
# version of the model.
export_path = './production_models/keras_example/1'
builder = tf.saved_model.builder.SavedModelBuilder(export_path)

# Create a signature definition for tfserving.
# We will use the predict API which allows us to have an arbitrary number of
# inputs and outputs.
model_signature = tf.saved_model.signature_def_utils.build_signature_def(
    inputs={tensor.name: tf.saved_model.utils.build_tensor_info(tensor)
            for tensor in model.inputs},
    outputs={tensor.name: tf.saved_model.utils.build_tensor_info(tensor)
             for tensor in model.outputs},
    method_name=tf.saved_model.signature_constants.PREDICT_METHOD_NAME)

# Serialize the model.
with tf.keras.backend.get_session() as session:
    builder.add_meta_graph_and_variables(
        session,
        [tf.saved_model.tag_constants.SERVING],  # This is just a tag.
        signature_def_map={
            'predict_whatever':
                model_signature,
        })

    # Export the model to the production_models/1 folder.
    builder.save(as_text=True)
```



Note: In our simple case, we do not have any data preprocessing.

https://github.com/dekromp/deep_learning_and_ai_tooling_lecture/tree/master/tooling_lecture/deployment

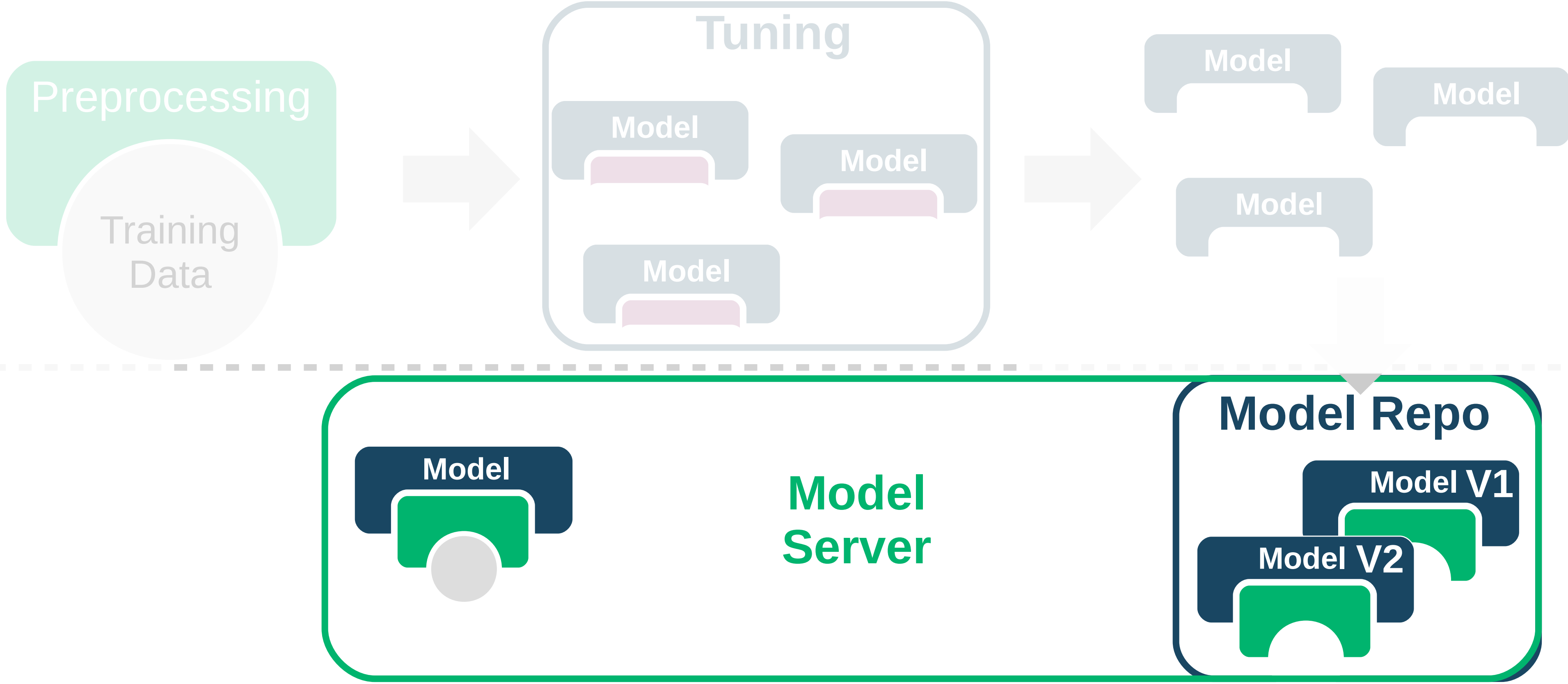
Deployment

Model Serving – Tensorflow Serving

```
(dev-p36) [redacted]/deployment(master)$ saved_model_cli show --dir=./production_models/keras_example/1 --all

MetaGraphDef with tag-set: 'serve' contains the following SignatureDefs:

signature_def['predict_whatever']:
  The given SavedModel SignatureDef contains the following input(s):
    inputs['input_x1:0'] tensor_info:
        dtype: DT_FLOAT
        shape: (-1, 10)
        name: input_x1:0
    inputs['input_x2:0'] tensor_info:
        dtype: DT_FLOAT
        shape: (-1, 20)
        name: input_x2:0
  The given SavedModel SignatureDef contains the following output(s):
    outputs['output_layer/Sigmoid:0'] tensor_info:
        dtype: DT_FLOAT
        shape: (-1, 1)
        name: output_layer/Sigmoid:0
  Method name is: tensorflow/serving/predict
```

Deployment

Model Serving – Tensorflow Serving

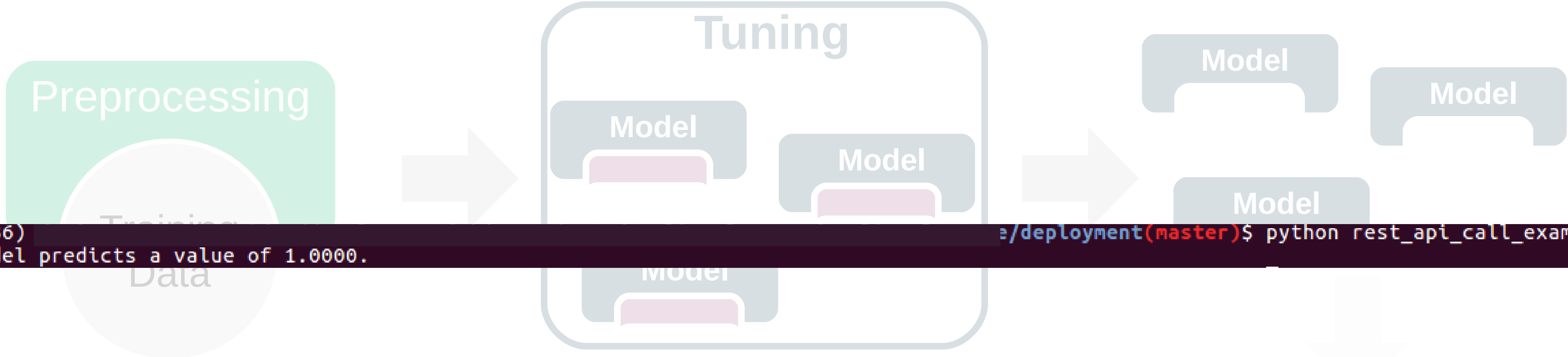



```bash
#!/bin/bash
# Run tensorflow model server and serve our model.
sudo docker run -p 8501:8501 -v $(pwd)/production_models:/models/ -e MODEL_NAME=keras_example --rm -t tensorflow/serving
```

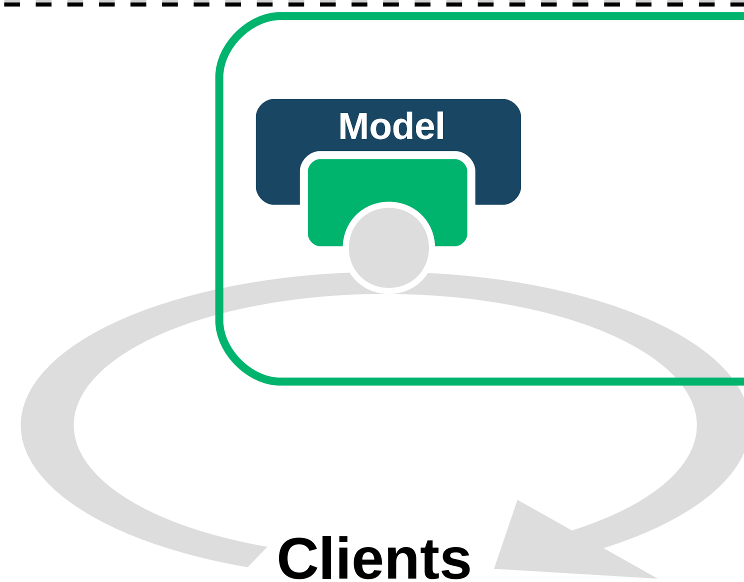
https://github.com/dekromp/deep_learning_and_ai_tooling_lecture/tree/master/tooling_lecture/deployment

Deployment

Model Serving – Tensorflow Serving




```
(dev-p36) [...]/deployment(master)$ python rest_api_call_example.py
The model predicts a value of 1.0000.
```

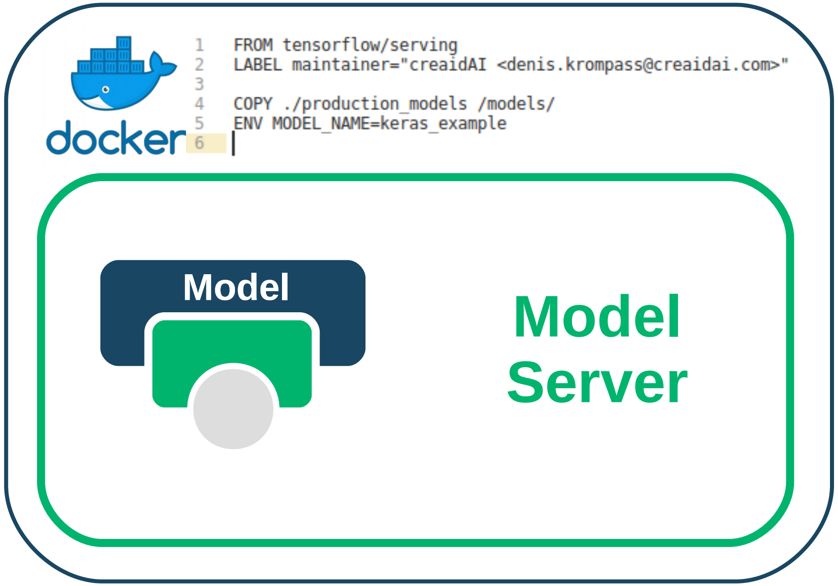



```python
"""Example script that shows how the REST-API of the model is called."""
import json

import numpy as np
import requests


url = 'http://localhost:8501/v1/models/keras_example:predict'

data = {
    'signature_name': 'predict_whatever',
    'inputs': {
        'input_x1:0': np.random.randn(1, 10).astype(np.float32).tolist(),
        'input_x2:0': np.random.randn(1, 20).astype(np.float32).tolist()
    }}

data = json.dumps(data)
response = requests.post(url, data=data)

print('The model predicts a value of %.4f.' % response.json()['outputs'][0][0])
```

https://github.com/dekromp/deep_learning_and_ai_tooling_lecture/tree/master/tooling_lecture/deployment

Deployment

Model Serving – Scaling up



kubernetes

Google Cloud Platform

amazon web services

Azure

Things that we did not cover

Data Validation

Pre-Trained Model Repositories

Serverless Deployment

Model Validation

Model Evaluation

Remote Debugging

Versioning

Embedded Deployment

Online Learning

Distributed Model Training

Thanks!