

# TOOLING

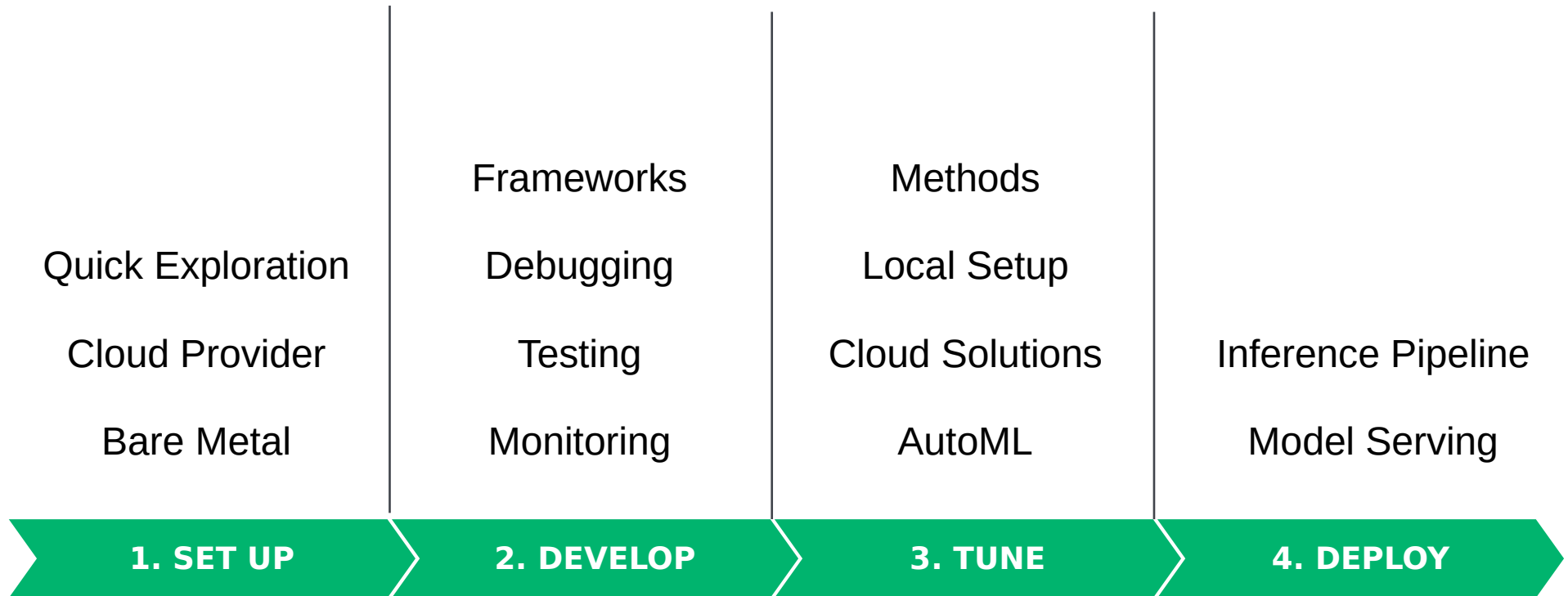
## *Deep Learning & AI*

**Dr. Denis Krompaß**

Co-Founder creaidAI

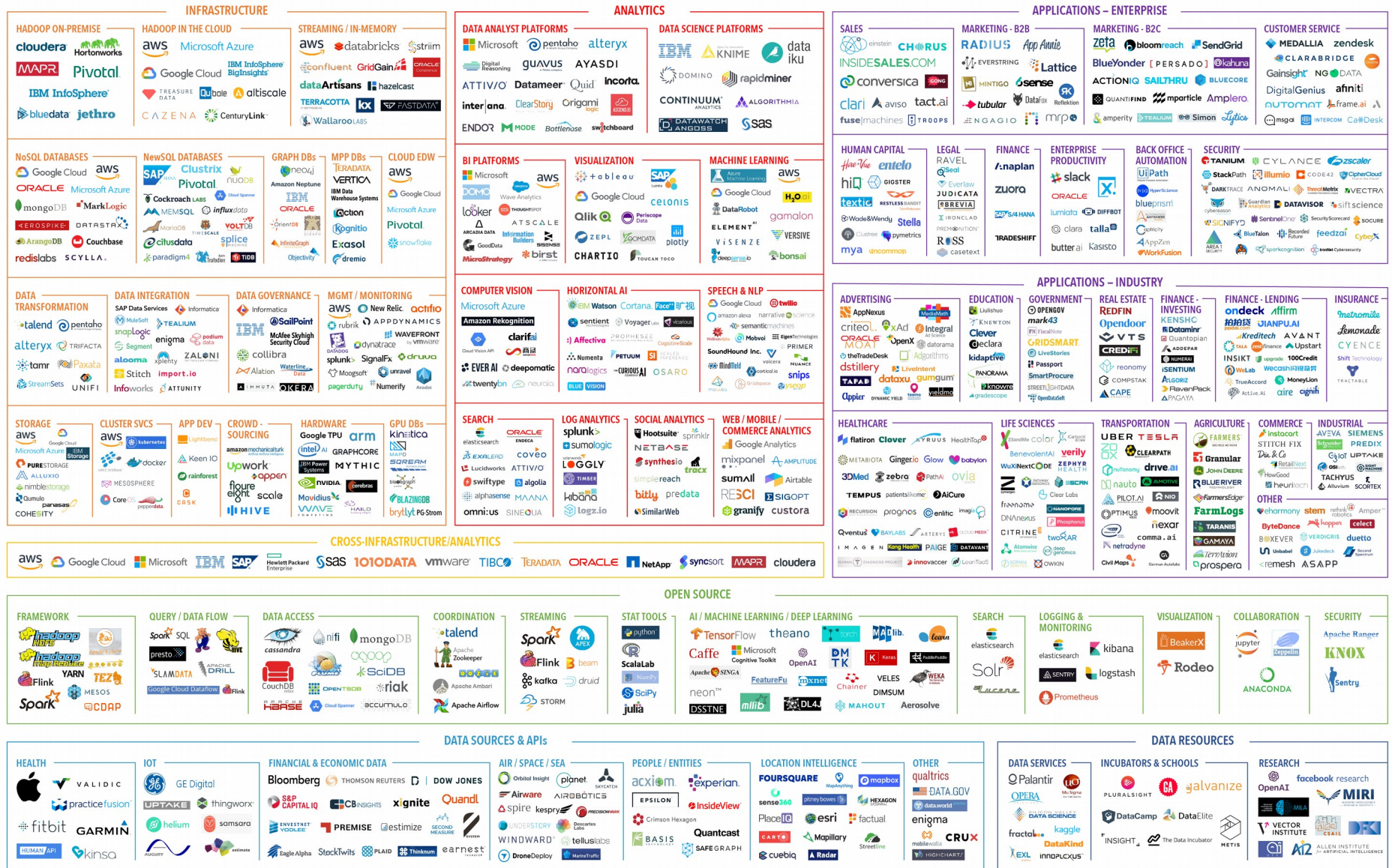
Senior Key Expert Deep Learning at Siemens

# Lecture Overview

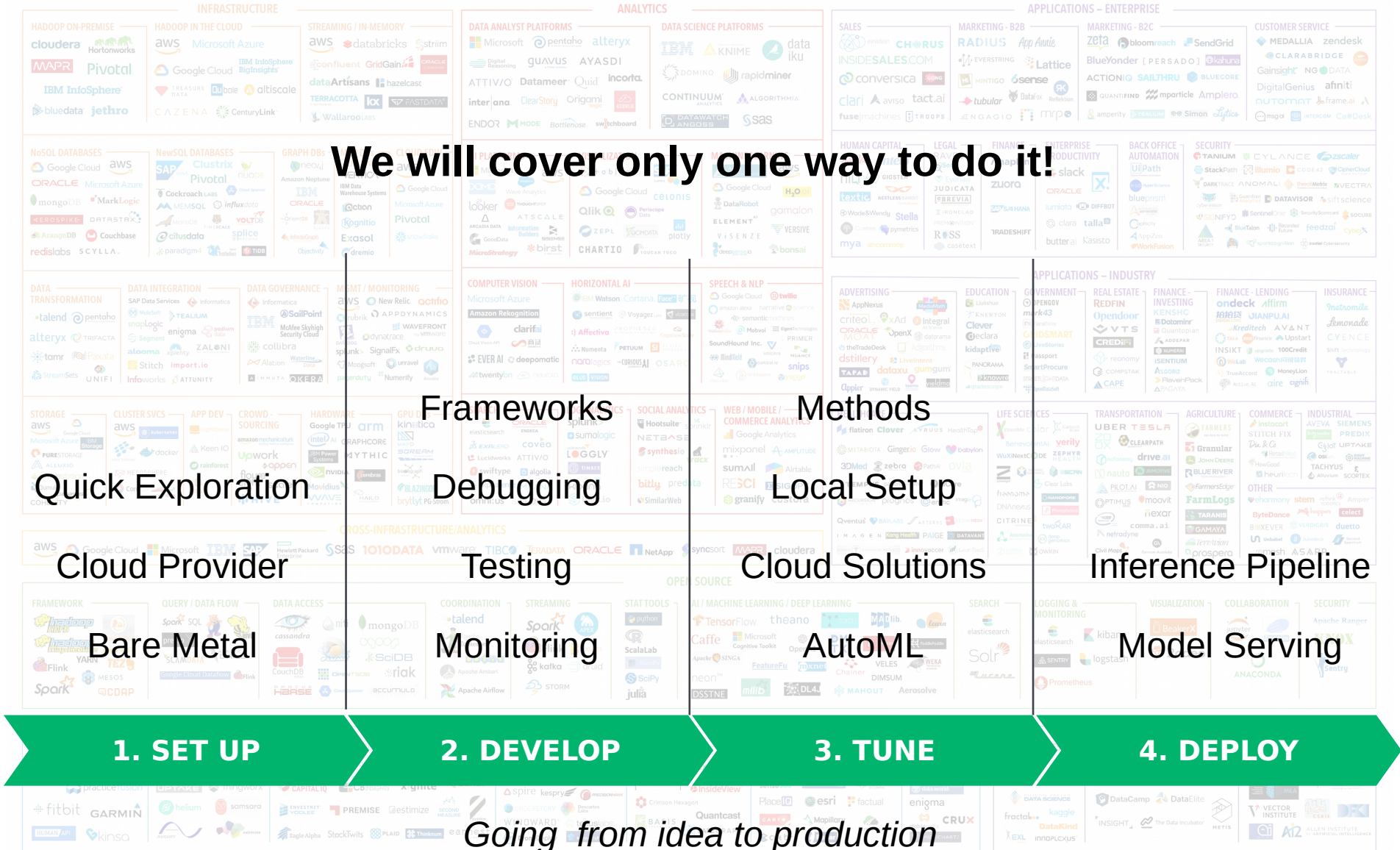


*Going from idea to production*

## BIG DATA & AI LANDSCAPE 2018







**SETUP**

# Google Colab

The screenshot displays the Google Colaboratory web interface. At the top, a dark grey header contains the Colab logo and the text "Welcome to Colaboratory!". Below this, a navigation bar with orange tabs includes "EXAMPLES", "RECENT", "GOOGLE DRIVE", "GITHUB", and "UPLOAD". The "RECENT" tab is active, showing a table of notebooks. The table has columns for "Title", "First opened", and "Last opened". A single notebook titled "Hello, Colaboratory" is listed, with "10 minutes ago" for the first opened time and "0 minutes ago" for the last opened time. To the right of the table is a share icon. Below the table, there are two buttons: "NEW PYTHON 3 NOTEBOOK" and "NEW PYTHON 2 NOTEBOOK", with a "CANCEL" button to the right. The background of the interface is a blurred view of the "Getting Started" page, which includes a list of links and a section on "Highlighted Features".

**Welcome to Colaboratory!**

Colaboratory is a free Jupyter notebook environment that

**Getting Started**

- [Overview of Colaboratory](#)
- [Loading and saving data: Local files, Drive, Sheets, Google Cloud Storage](#)
- [Importing libraries and installing dependencies](#)
- [Using Google Cloud BigQuery](#)
- [Forms, Charts, Markdown, & Widgets](#)
- [TensorFlow with GPU](#)
- [TensorFlow with TPU](#)
- [Machine Learning Crash Course: Intro to Pandas & First Steps with](#)
- [Using Colab with GitHub](#)

**Highlighted Features**

**Seedbank**

Looking for Colab notebooks to learn from? Check out [Seedbank](#), a place


**TensorFlow execution**

Colaboratory allows you to execute TensorFlow code in your browser with

$$\begin{bmatrix} 1. & 1. & 1. \\ 1. & 1. & 1. \end{bmatrix} + \begin{bmatrix} 1. & 2. & 3. \\ 4. & 5. & 6. \end{bmatrix} = \begin{bmatrix} 2. & 3. & 4. \\ 5. & 6. & 7. \end{bmatrix}$$

```
[ ] import tensorflow as tf
```

Filter notebooks

Title	First opened	Last opened
 Hello, Colaboratory	10 minutes ago	0 minutes ago

NEW PYTHON 3 NOTEBOOK

NEW PYTHON 2 NOTEBOOK

CANCEL

<https://colab.research.google.com>

colab\_example.ipynb

File Edit View Insert Runtime Tools Help

CODE TEXT CELL CELL

```
[3] import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
import tensorflow as tf

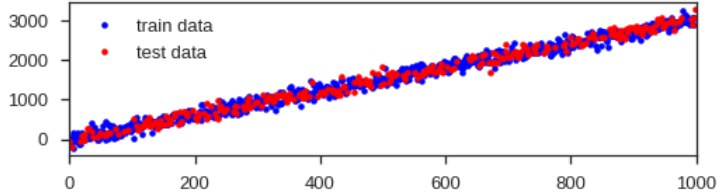
sns.set_context("talk")
sns.set_style("ticks")

[33] # Generate some random data generated by a random linear function.
size = 1000
x = np.arange(size).astype(np.float32)
noise = np.random.normal(0, size // 10, size=(size,))
w = np.random.uniform(-5, 5)
b = np.random.uniform(-5, 5)
y = w * x + b + noise

# Split the data into training and tests set.
example_idcs = np.random.permutation(np.arange(len(x)))
split_idx = int(size * 0.7)
x_train = x[example_idcs[:split_idx]]
x_test = x[example_idcs[split_idx:]]
y_train = y[example_idcs[:split_idx]]
y_test = y[example_idcs[split_idx:]]

# Plot the training and test examples.
plt.figure(figsize=(8, 2))
plt.plot(x_train, y_train, '.', color='b', label='train data')
plt.plot(x_test, y_test, '.', color='r', label='test data')
plt.xlim(0, size)
plt.legend()
```

<matplotlib.legend.Legend at 0x7fdba1dcd9e8>



```
# Define the inputs.
input_x = tf.placeholder(tf.float32, [None])
input_y = tf.placeholder(tf.float32, [None])

# Normalize the inputs.
mean_x = tf.constant(np.mean(x_train))
std_x = tf.constant(np.std(x_train))
input_x_normed = (input_x - mean_x) / std_x
input_x_normed = tf.expand_dims(input_x_normed, axis=-1)
```

# Google Colab

## Notebook settings

Runtime type

Python 3

Hardware accelerator

GPU

☐ Omit code cell output when saving this notebook

CANCEL

SAVE

<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
- ...



## Deep Learning Base AMI (Ubuntu)

Deep Learning AMI with a foundational platform of NVIDIA CUDA, cuDNN, GPU drivers, Intel MKL-DNN, and other low-level system libraries for deploying your own custom deep learning environment.

For example, for machine learning developers contributing to open source deep learning framework enhancements, the Deep Learning AMI provides a clean ...

[More info](#)

[View Additional Details in AWS Marketplace](#)

## Deep Learning Base AMI (Ubuntu)

### Pricing Details

#### Hourly Fees

Instance Type	Software	EC2	Total
R5D 12 Extra Large	\$0.00	\$3.84	<b>\$3.84/hr</b>
M3 Extra Large	\$0.00	\$0.293	<b>\$0.293/hr</b>
R4 16 Extra Large	\$0.00	\$4.742	<b>\$4.742/hr</b>
R5 AMD Double Extra Large	\$0.00	\$0.508	<b>\$0.508/hr</b>
M5 Extra Large	\$0.00	\$0.214	<b>\$0.214/hr</b>
High I/O Quadruple Extra Large	\$0.00	\$1.376	<b>\$1.376/hr</b>
T2 Large	\$0.00	\$0.101	<b>\$0.101/hr</b>
T2D Triple Extra Large	\$0.00	\$1.248	<b>\$1.248/hr</b>
M5 Large	\$0.00	\$0.107	<b>\$0.107/hr</b>
C5D Large	\$0.00	\$0.109	<b>\$0.109/hr</b>
C5 Large	\$0.00	\$0.096	<b>\$0.096/hr</b>
M5 Double Extra Large	\$0.00	\$0.428	<b>\$0.428/hr</b>
CC2 Cluster Compute	\$0.00	\$2.25	<b>\$2.25/hr</b>
T2 Double Extra Large	\$0.00	\$0.403	<b>\$0.403/hr</b>
T2 Extra Large	\$0.00	\$0.202	<b>\$0.202/hr</b>
High I/O Extra Large	\$0.00	\$0.938	<b>\$0.938/hr</b>
M5D 12 Extra Large	\$0.00	\$3.024	<b>\$3.024/hr</b>

**Product Details**

By Amazon Web Services

Customer Rating **★★★★★** (2)

Latest Version **11**

Base Operating System **Linux/Unix, Ubuntu 16.04**

Delivery Method **64-bit Amazon Machine Image (AMI)**

License Agreement **[End User License Agreement](#)**

On Marketplace Since **11/15/17**

AWS Services Required **Amazon EC2, Amazon EBS**

**LinkInTitle**

**Cancl**

**Continue**

[Home](#) / [Services](#) / [Azure Machine Learning service](#)

# Azure Machine Learning service



Accelerate machine learning from the cloud to the edge

[Start free >](#)

Already an Azure subscriber? [Sign in to get started >](#)

[Explore Azure Machine Learning service](#)
[Pricing details](#)
[Documentation](#)
[Calculator](#)
[Azure Machine Learning service overview](#)

- ✓ Build and train machine learning models faster, and easily deploy to the cloud or the edge with Azure Machine Learning service.
- ✓ Use the latest open source technologies such as TensorFlow, PyTorch, or Jupyter.
- ✓ Experiment locally and then quickly scale up or out with large GPU-enabled clusters in the cloud.
- ✓ Speed up data science with automated machine learning and hyper-parameter tuning.
- ✓ Track your experiments, manage models, and easily deploy with integrated CI/CD tooling.



- AI and machine learning →

ALPHA

AI Hub

Discover, share, and deploy AI on Google Cloud.

BETA

## Cloud AutoML

Easily train high-quality, custom ML models.

Cloud TPU

Train and run ML models faster than ever.

## Cloud Machine Learning Engine

Build superior models and deploy them into production.

## Cloud Talent Solution

Put AI to work on your hiring needs.

Dialogflow Enterprise Edition

Create conversational experiences across devices and platforms.

## Cloud Natural Language

Derive insights from unstructured text.

## Cloud Speech-to-Text

Speech-to-text conversion powered by ML.

## Cloud Text-to-Speech

Text-to-speech conversion powered by ML.

# Bare Metal

```
$ pip install tensorflow
```

```
$ conda install tensorflow
```

# Bare Metal

\$ pip install tensorflow

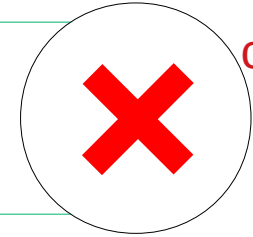
```
>>> import tensorflow as tf
>>> tf.__version__
'1.12.0'
>>> tf.Session()
2018-11-25 13:46:01.280605: I tensorflow/core/platform/cpu_feature_guard.cc:141]
Your CPU supports instructions that this TensorFlow binary was not compiled to
use: AVX2 FMA
```

\$ 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
```



cuda toolkit  
+  
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>

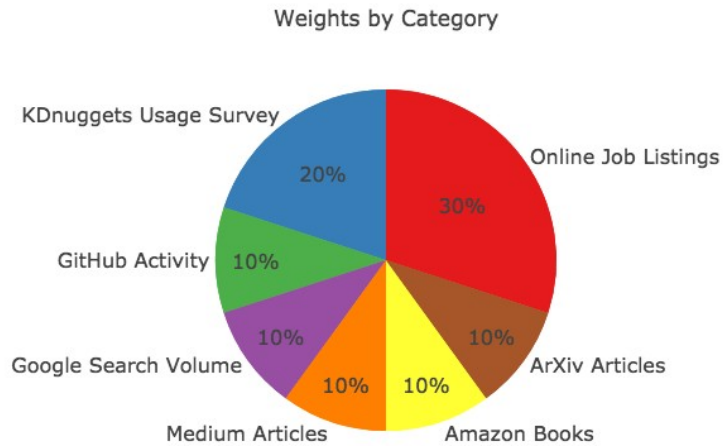
**DEVELOPMENT**

# Frameworks

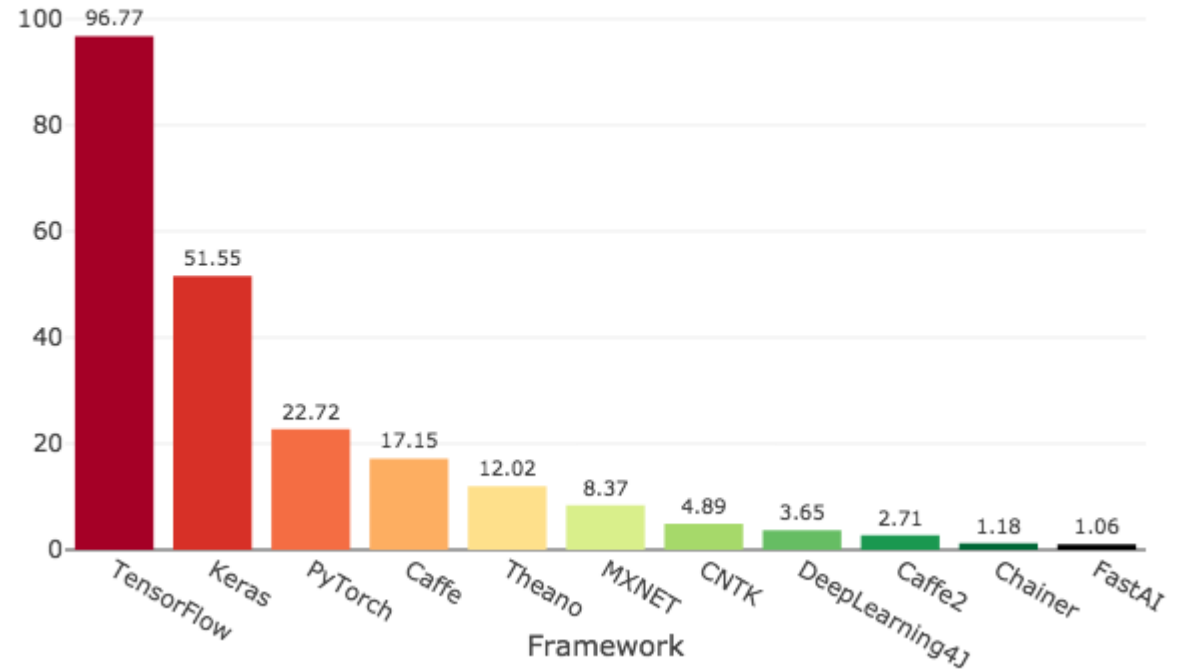


*And many more ...*

# Frameworks



Deep Learning Framework Power Scores 2018



# Before we start...

[https://github.com/dekromp/deep\\_learning\\_and\\_ai\\_tooling\\_lecture](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/1n2RIMdmv1p25Xy5thJUhkKGvjtV-dkAIsUXP-AL4ffl/preview>

**Don't use them for writing your machine learning code!**

**Notebooks are great for plotting stuff.**

# Beware!

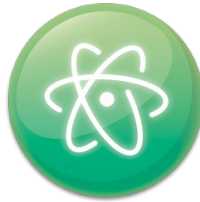
- Find a good text editor and get familiar with it:



Visual Studio  
Code



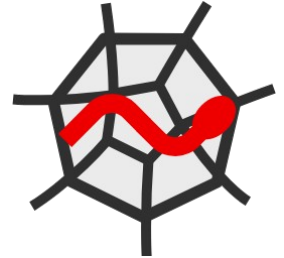
Sublime



Atom



PyCharm



Spyder

***And many more ...***

# Data Scientist are Software Developers

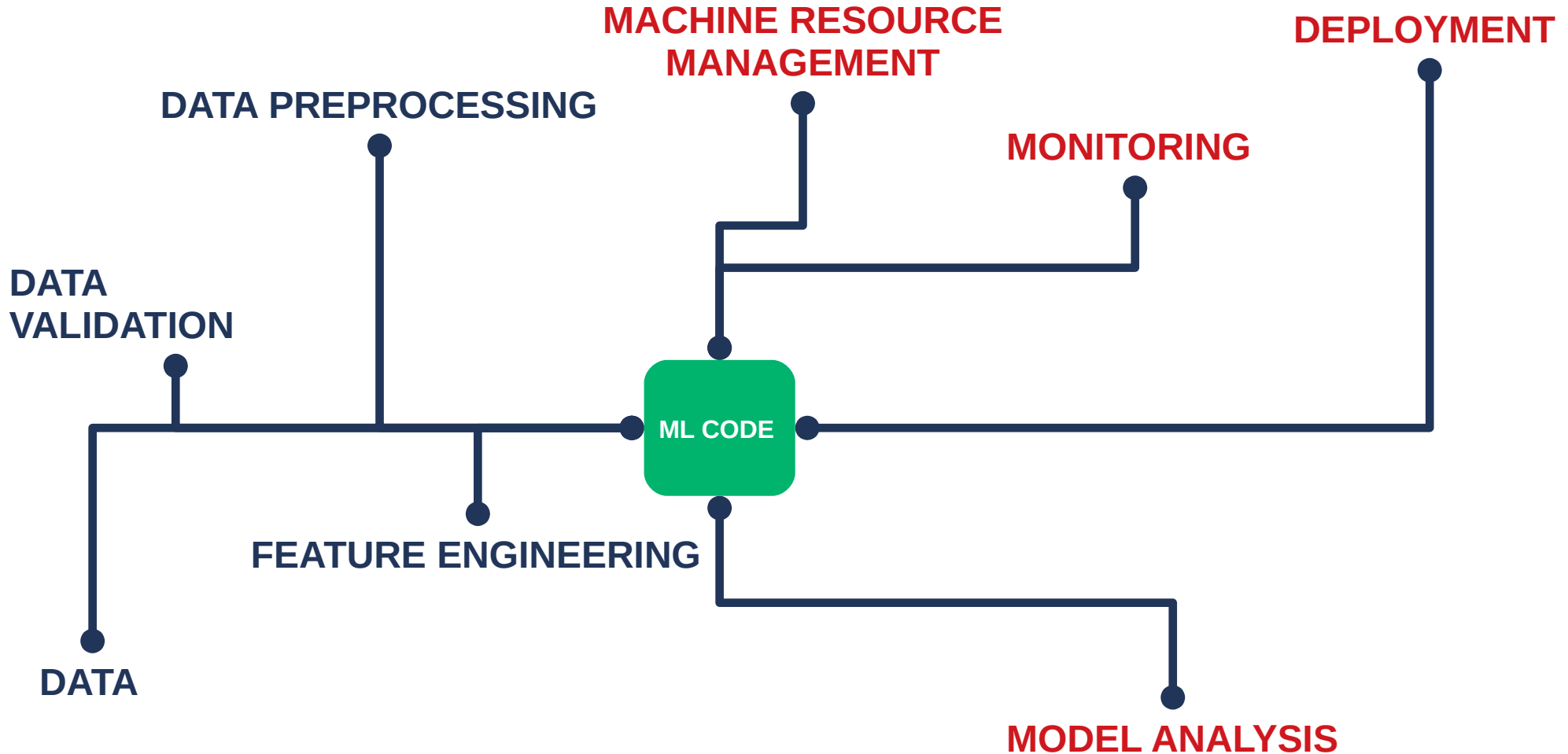
- Get familiar with coding guidelines (Python: PEP 8)
- Document your code (PEP 257, NumPy Style, ...)
- Write tests!!! (e.g. Unit-tests with pytest)
- Modularize your code.

```
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
```

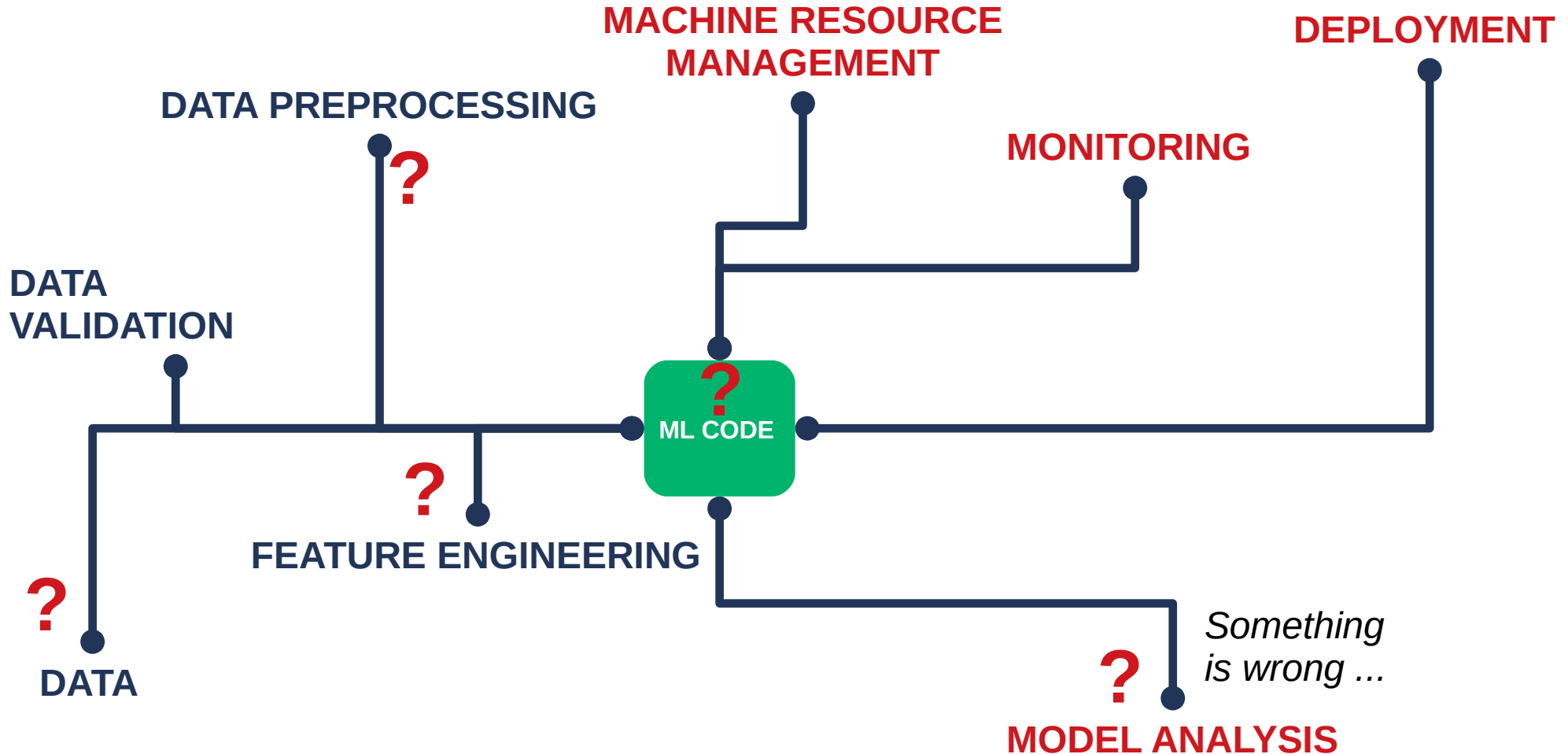
```
def f(x, y):  
    xtxi = np.linalg.pinv(np.dot(x.T, x))  
    xty = np.dot(x.T, y)  
    w = np.dot(xtxi, xty)  
    return w
```

```
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  
    #  $w = (X^T X)^{-1} * X^T * y$   
    xty = np.dot(x.T, y)  
    w = np.dot(xtxi, xty)  
  
    return w
```

# Why tests?



# Why tests?



# Testing with pytest

```
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 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) |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 =====
```

# 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.

Now we can start.

# Quick Refresher

```
1  """Simple example script that shows basic operations of tensorflow."""
2  import numpy as np
3  import tensorflow as tf
4
5
6  # Fix the random seeds to make the computations reproducible.
7  tf.set_random_seed(12345)
8  np.random.seed(12321)
9
10 # Create an placeholder for feeding inputs in the graph.
11 input_x = tf.placeholder(tf.float32, [None, 3], name='features')
12
13 # Create a variable.
14 w = tf.get_variable(
15     'weights', [3, 1], initializer=tf.glorot_uniform_initializer())
16
17 # Perform some computation steps.
18 output = tf.matmul(input_x, w)
19 output = tf.reshape(output, [-1]) # Flatten the outputs.
20
21 # Generate some random input data.
22 x = np.random.randn(5, 3)
23
24 # Execute the graph on some random data.
25 with tf.Session() as session:
26     # Boilerplate code that initializes all variables in the graph (just w).
27     session.run(tf.global_variables_initializer())
28     output_value = session.run(output, feed_dict={input_x: x})
29     print('Output: %s' % str(output_value))
30     # Output: [ 1.382279 -0.9660325 -0.5551475  0.1781615 -1.5802894]
31
```

!!!



Reproducibility is a big issue in ML

# Quick Refresher

```
"""Simple example script that shows basic operations of tensorflow."""
2 import numpy as np
3 import tensorflow as tf
4
5
6 # Fix the random seeds to make the computations reproducible.
7 tf.set_random_seed(12345)
8 np.random.seed(12321)
9
10 # Create an placeholder for feeding inputs in the graph.
11 input_x = tf.placeholder(tf.float32, [None, 3], name='features')
12
13 # Create a variable.
14 w = tf.get_variable(
15     'weights', [3, 1], initializer=tf.glorot_uniform_initializer())
16
17 # Perform some computation steps.
18 output = tf.matmul(input_x, w)
19 output = tf.reshape(output, [-1]) # Flatten the outputs.
20
21 # Create a target placeholder and define the loss computation.
22 input_y = tf.placeholder(tf.float32, [None], name='target')
23 # Mean squared error.
24 loss = tf.reduce_mean(tf.square(output - input_y))
25
26 # Define the update operation (stochastic gradient descent).
27 update_op = tf.assign(w, w - 0.01 * tf.gradients(loss, w)[0])
28
29 # Generate some random training data.
30 x = np.random.randn(100, 3)
31 unknown_w = np.array([0.3, -0.21, 0.8])
32 y = np.dot(x, unknown_w)
33
34 # Execute the graph on some random data.
35 batch_size = 8
36 num_epochs = 15
37 with tf.Session() as session:
38     # Boilerplate code that initializes all variables in the graph (just w).
39     session.run(tf.global_variables_initializer())
40     for epoch in range(num_epochs): # Train for 15 epochs.
41         # Shuffle the training data.
42         shuffle_idx = np.random.permutation(np.arange(len(x)))
43         x = x[shuffle_idx]
44         y = y[shuffle_idx]
45
46         # Train the model on batches of data with SGD.
47         epoch_losses = []
48         for i in range(0, len(x), batch_size):
49             batch_loss, _ = session.run(
50                 [loss, update_op],
51                 feed_dict={input_x: x[i: i + batch_size],
52                           input_y: y[i: i + batch_size]})
53             epoch_losses += [batch_loss]
54
55         print('Epoch %d; TrainLoss: %.4f' % (epoch + 1, np.mean(epoch_losses)))
56
57 print('Found parameters: %s' % str(w.eval().reshape(-1)))
58 print('True parameters: %s' % str(unknown_w))
59
```

```

1 """Simple example script that shows basic operations of tensorflow."""
2 import numpy as np
3 import tensorflow as tf
4
5
6 # Fix the random seeds to make the computations reproducible.
7 tf.set_random_seed(12345)
8 np.random.seed(12321)
9
10 # Create an placeholder for feeding inputs in the graph.
11 input_x = tf.placeholder(tf.float32, [None, 3], name='features')
12
13 # Create a variable.
14 w = tf.get_variable(
15     'weights', [3, 1], initializer=tf.glorot_uniform_initializer())
16
17 # Perform some computation steps.
18 output = tf.matmul(input_x, w)
19 output = tf.reshape(output, [-1]) # Flatten the outputs.
20
21 # Create a target placeholder and define the loss computation.
22 input_y = tf.placeholder(tf.float32, [None], name='target')
23 # Mean squared error.
24 loss = tf.reduce_mean(tf.square(output - input_y))
25
26 # Define the update operation (stochastic gradient descent).
27 update_op = tf.assign(w, w - 0.01 * tf.gradients(loss, w)[0])
28
29 # Generate some random training data.
30 x = np.random.randn(100, 3)
31 unknown_w = np.array([0.3, -0.21, 0.8])
32 y = np.dot(x, unknown_w)
33
34 # Execute the graph on some random data.
35 batch_size = 8
36 num_epochs = 15
37 with tf.Session() as session:
38     # Boilerplate code that initializes all variables in the graph (just w).
39     session.run(tf.global_variables_initializer())
40     for epoch in range(num_epochs): # Train for 15 epochs.
41         # Shuffle the training data.
42         shuffle_idx = np.random.permutation(np.arange(len(x)))
43         x = x[shuffle_idx]
44         y = y[shuffle_idx]
45
46         # Train the model on batches of data with SGD.
47         epoch_losses = []
48         for i in range(0, len(x), batch_size):
49             batch_loss, _ = session.run(
50                 [loss, update_op],
51                 feed_dict={input_x: x[i: i + batch_size],
52                             input_y: y[i: i + batch_size]})
53             epoch_losses += [batch_loss]
54
55         print('Epoch %d; TrainLoss: %.4f' % (epoch + 1, np.mean(epoch_losses)))
56
57 print('Found parameters: %s' % str(w.eval().reshape(-1)))
58 print('True parameters: %s' % str(unknown_w))
59

```

# Quick Refresher

```

12 # Fix the random seeds to make the computations reproducible.
13 tf.set_random_seed(12345)
14 np.random.seed(12321)
15
16 # Constants of the experiments.
17 unknown_true_w = np.array([0.3, -0.21, 0.8])
18
19
20 def main(num_epochs, batch_size, learning_rate):
21     """Train a simple model on random data.
22
23     Parameters
24     -----
25     num_epochs : int
26         The number of epochs the model is trained.
27     batch_size : int
28         The batch size used for SGD.
29     learning_rate : float
30         The learning rate used for SGD.
31
32     """
33     # Generate some random training data.
34     x = np.random.randn(100, 3)
35     y = np.dot(x, unknown_true_w)
36
37     # Build forward pass.
38     input_x, output = build_forward_pass()
39     # Build the update op with respect to the objective.
40     update_op, loss, input_y = build_objective(output, learning_rate)
41     # Fit the model on the input data.
42     inputs = (input_x, input_y)
43     data = (x, y)
44     train_model(inputs, data, loss, update_op, batch_size, num_epochs)
45

```

# Quick Refresher

```
# Fix the random seeds to make the computations reproducible.
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)
```

Nice tutorial:

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



[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 output of 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.

# 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 matmul
  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)]]
```

```
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)})
```

# 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
    op, 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 matmul
    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)]]
```

```
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)})
```

# Debugging

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

```
1  """A simple script shows how scopes work."""
2  import numpy as np
3  import tensorflow as tf
4
5
6  x = tf.placeholder(tf.float32, [None, 10])
7
8  print('No scope is used:')
9  w1 = tf.get_variable(
10     'w1', dtype=np.float32, initializer=np.ones((10, 3), dtype=np.float32))
11  h1 = tf.matmul(x, w1)
12
13  w2 = tf.get_variable(
14     'w2', dtype=np.float32, initializer=np.ones((3, 10), dtype=np.float32))
15  h2 = tf.matmul(h1, w2)
16
17  for tensor in [x, w1, w2, h1, h2]:
18     print(tensor)
19
20  print('\nScope is used:')
21  with tf.variable_scope('first_block'):
22     w1 = tf.get_variable(
23         'w1', dtype=np.float32, initializer=np.ones((10, 3), dtype=np.float32))
24     h1 = tf.matmul(x, w1)
25
26  with tf.variable_scope('second_block'):
27     w2 = tf.get_variable(
28         'w2', dtype=np.float32, initializer=np.ones((3, 10), dtype=np.float32))
29     h2 = tf.matmul(h1, w2)
30  for tensor in [x, w1, w2, h1, h2]:
31     print(tensor)
```



```
No scope is used:
Tensor("Placeholder:0", shape=(?, 10), dtype=float32)
<tf.Variable 'w1:0' shape=(10, 3) dtype=float32_ref>
<tf.Variable 'w2:0' 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/w1:0' shape=(10, 3) dtype=float32_ref>
<tf.Variable 'second_block/w2:0' shape=(3, 10) dtype=float32_ref>
Tensor("first_block/MatMul:0", shape=(?, 3), dtype=float32)
Tensor("second_block/MatMul:0", shape=(?, 10), dtype=float32)
```

# Debugging

- Something seems to be wrong...

```
(dev-p36) (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 TensorFlow
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
```

# Debugging

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

```
6 import numpy as np
7 import tensorflow as tf
8 from tensorflow.python import debug as tf_debug
9
108 ▼ with tf.Session() as session:
109     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.

```

- 201 | | | # Initialize all variables in the graph.
  202 | | | session.run(tf.global_variables_initializer())

```

```

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):

```

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

--- 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
[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
--- Scroll (PgDn/PgUp): 21.69% --- Mouse: ON ---
tfdbg>
```

- That's 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/v: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

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=float32)

Tensor "loss/loss_out:0:DebugIdentity":
dtype: float32
shape: ()
array(nan, dtype=float32)

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

```
73 # Build the loss.
74 with tf.name_scope('loss'):
75     # Flatten the output.
76     output = tf.reshape(output, [-1])
77     # Create an input for the targets
78     input_y = tf.placeholder(tf.float32, [None], name='input_y')
79
80     # Compute the loss (binary cross entropy)
81     epsilon = 1e-7 # for numerical stability.
82     loss = -(tf.multiply(input_y, tf.log(output + epsilon)) +
83             tf.multiply(1 - input_y, tf.log(1 - output + epsilon)))
84     loss = tf.reduce_mean(loss, name='loss_out')
```

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

```
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
```



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

28

Returns

29

30

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.

31

32

33

34

35

36

37

38

```
"""
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)

46

47

48

```
return input_x1, input_x2, output
```

```
(dev-p36)
2018-11-26 14:41:10.734006: I tensorflow/core/platform/cpu_feature_guard.cc:141] Your CPU supports instructions that this TensorFlow
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
w(master)$ python buggy_tensorflow_nan_output.py
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
```

# 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 imperative programming
- Inspired by  PyTorch

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

# Eager Execution Example

```
1  """This script shows a simple example on how eager execution works."""
2  import numpy as np
3  import tensorflow as tf
4
5
6  # Enable eager execution.
7  tf.enable_eager_execution()
8
9  # Make the execution reproducible.
10 tf.set_random_seed(2132)
11 np.random.seed(3423)
12
13 # Generate some random data.
14 x = np.arange(3).reshape(-1, 1).astype(np.float32)
15 w = tf.get_variable(
16     'w', dtype=np.float32, shape=[1, 3],
17     initializer=tf.glorot_uniform_initializer())
18
19 # Interwine python and tensorflow code directly.
20 z = tf.matmul(w, x)
21 if np.sum(x) > 0:
22     h = -tf.nn.sigmoid(z)
23 else:
24     h = tf.nn.sigmoid(z)
25
26 # Evaluate immediately the output without session run.
27 print(h)
28
29 # tf.Tensor([[-0.36252844]], shape=(1, 1), dtype=float32)
30
```

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:

```
# 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]
```

```
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))
```

# 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!

# 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/contrib/eager/python/examples).

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

# Eager Execution

```
1  """This script shows a simple example on how eager execution works."""
2  import numpy as np
3  import tensorflow as tf
4
5
6  # Enable eager execution.
7  tf.enable_eager_execution()
8
9  # Make the execution reproducible.
10 tf.set_random_seed(2132)
11 np.random.seed(3423)
12
13 # Generate some random data.
14 x = np.arange(3).reshape(-1, 1).astype(np.float32)
15 w = tf.get_variable(
16     'w', dtype=np.float32, shape=[1, 3],
17     initializer=tf.glorot_uniform_initializer())
18
19 # Interwine python and tensorflow code directly.
20 z = tf.matmul(w, x)
21 if np.sum(x) > 0:
22     h = -tf.nn.sigmoid(z)
23 else:
24     h = tf.nn.sigmoid(z)
25
26 # Evaluate immediately the output without session run.
27 print(h)
28
29 # tf.Tensor([[-0.36252844]], shape=(1, 1), dtype=float32)
30
```

Mix arrays and tensors directly

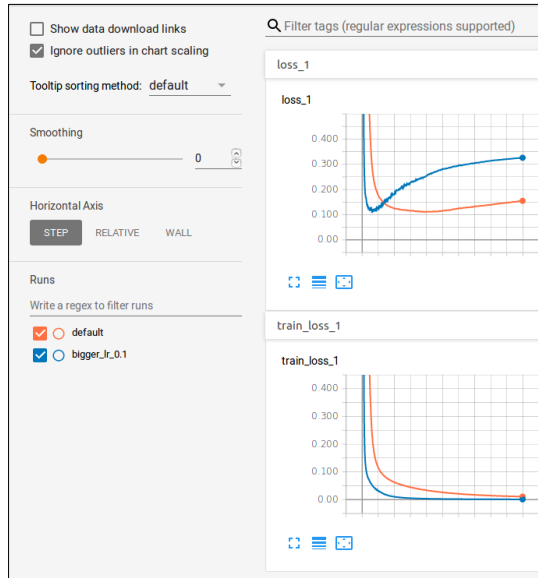
Mix python control flows with Tensorflow  
Only for static graphs

No session run calls required

# Tensorboard

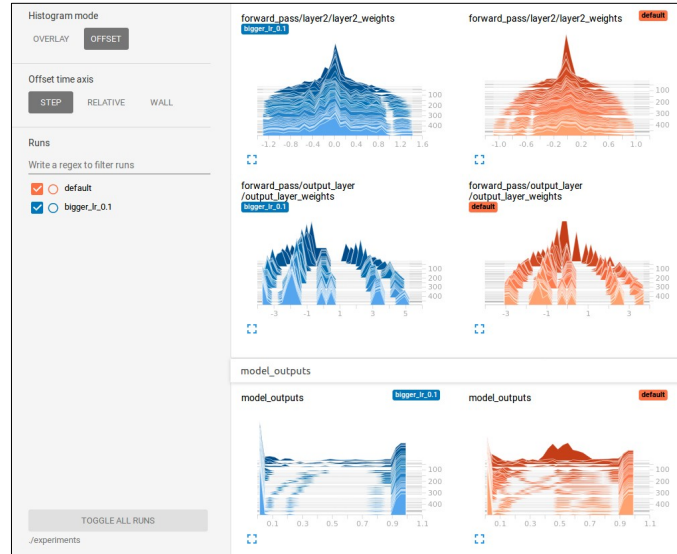
## Scalars

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



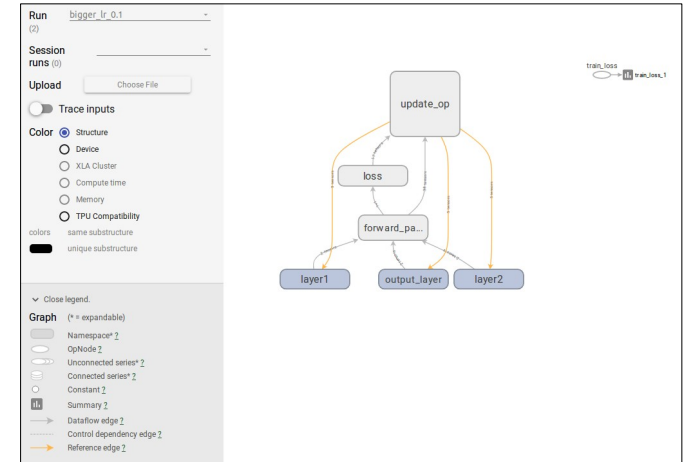
## Histograms

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



## Graph

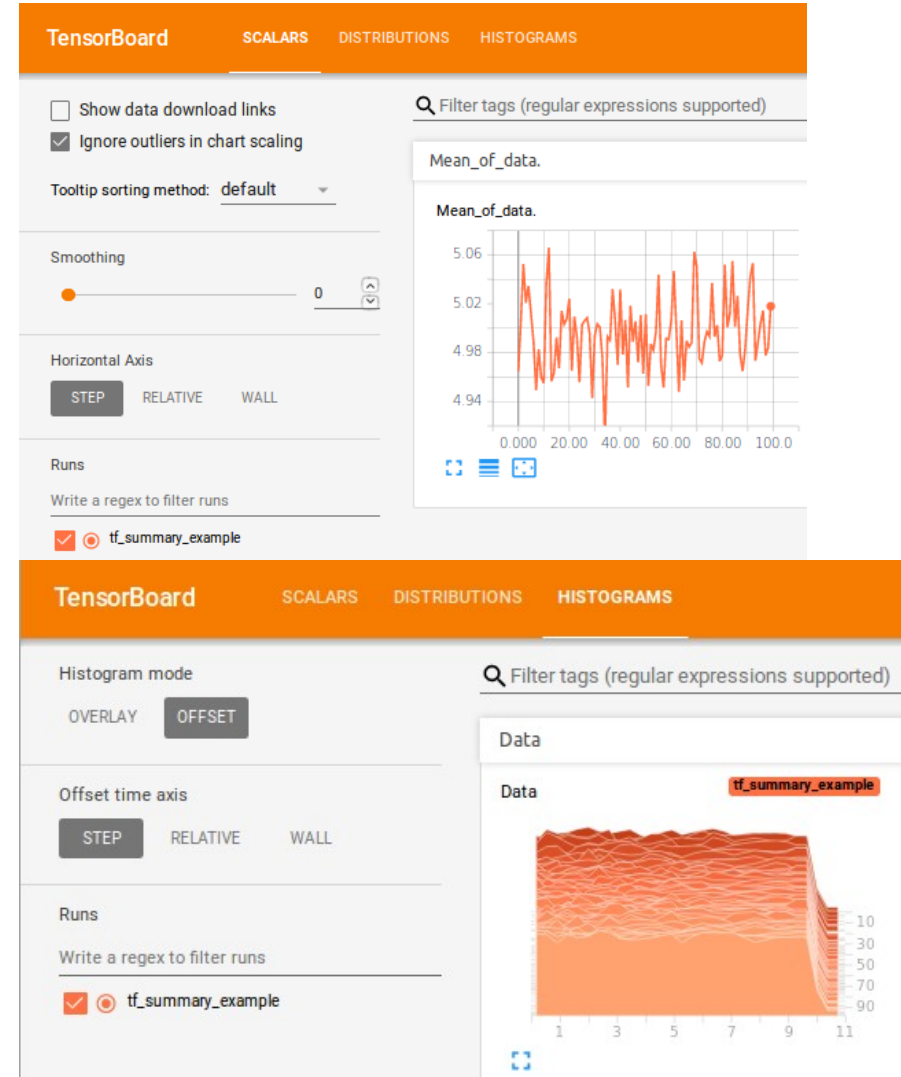
```
260 # Add the graph to the summaries.  
261 summary_writer.add_graph(session.graph)
```



[https://www.tensorflow.org/api\\_docs/python/tf/summary](https://www.tensorflow.org/api_docs/python/tf/summary)

# Tensorboard

```
1  """Minimal example that shows how summaries are used.
2
3  Observe the result in the web browser (localhost:6006) after starting the
4  tensorboard:|
5
6  $ tensorboard --logdir=./experiments/
7  """
8  import numpy as np
9  import tensorflow as tf
10
11
12  np.random.seed(12123)
13
14  # Create a placeholder for our data.
15  input_data = tf.placeholder(tf.float32, [1000, 10])
16
17  # Create some summaries.
18  tf.summary.scalar('Mean of data.', tf.reduce_mean(input_data))
19  tf.summary.histogram('Data', input_data)
20
21  # Merge all summaries. <- tensorflow magic op.
22  all_summaries_op = tf.summary.merge_all()
23
24  # Create a writer for storing the summaries on disk for tensorboard to find.
25  summary_writer = tf.summary.FileWriter('./experiments/tf_summary_example')
26
27  # Let's create some summary events.
28  with tf.Session() as session:
29      for step in range(100):
30          # Generate some random data.
31          data = np.random.uniform(0, 10, size=[1000, 10])
32
33          # Compute the summary values.
34          all_summaries = session.run(
35              all_summaries_op, feed_dict={input_data: data})
36
37          # Write the summaries, dont forget the step ('x-coordinate' in plot).
38          summary_writer.add_summary(all_summaries, step)
39
40
```

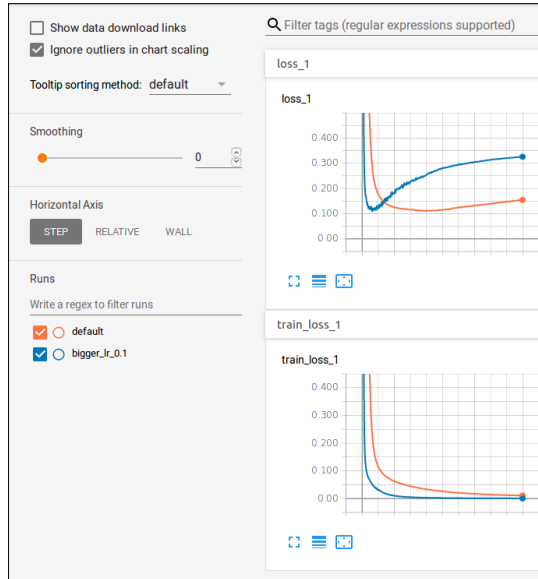


# Tensorboard

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

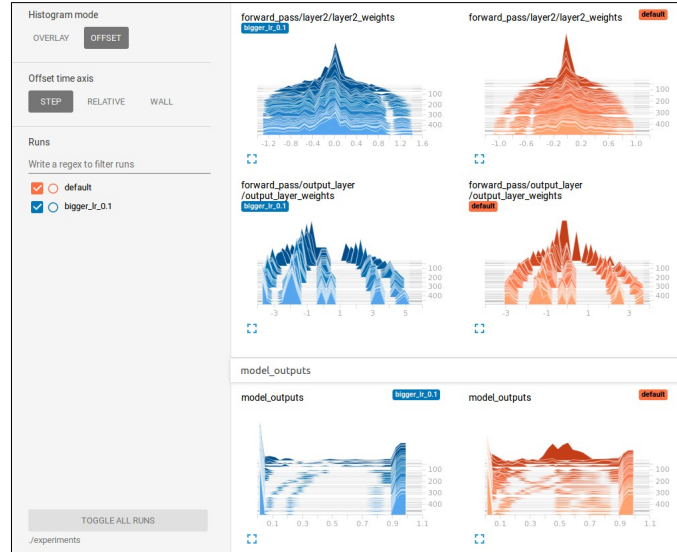
Visualize /  
compare learning  
curves

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



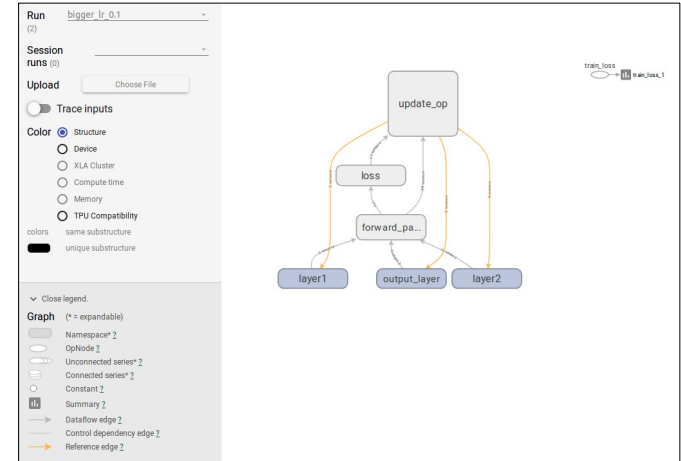
Visualize how parameters  
and outputs are evolving

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

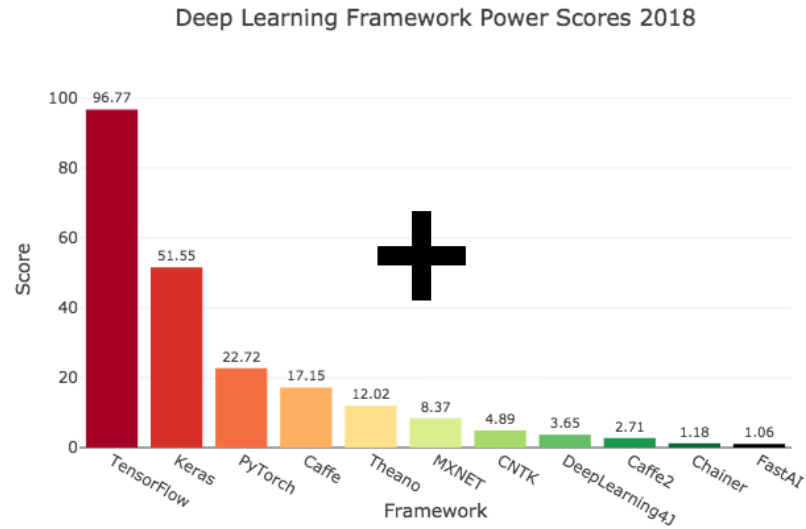


Visualize the  
computation graph  
(Use scopes and names)

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



# Frameworks



# 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
```

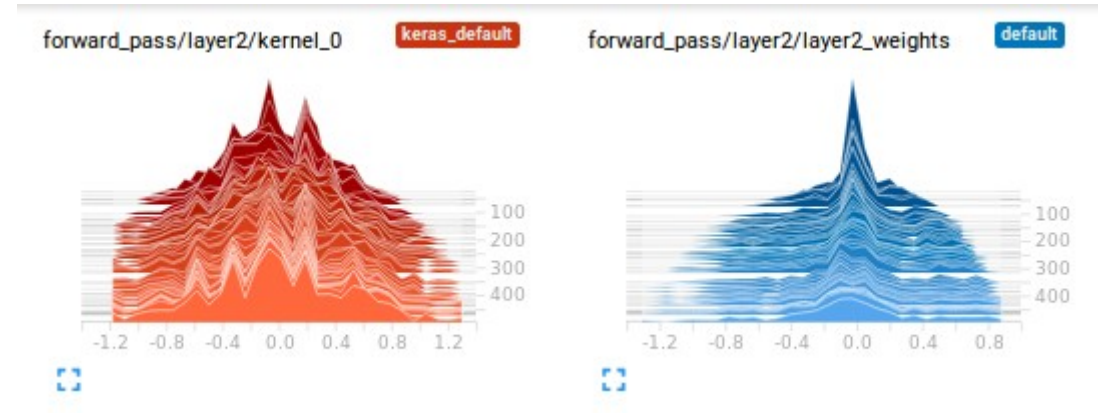
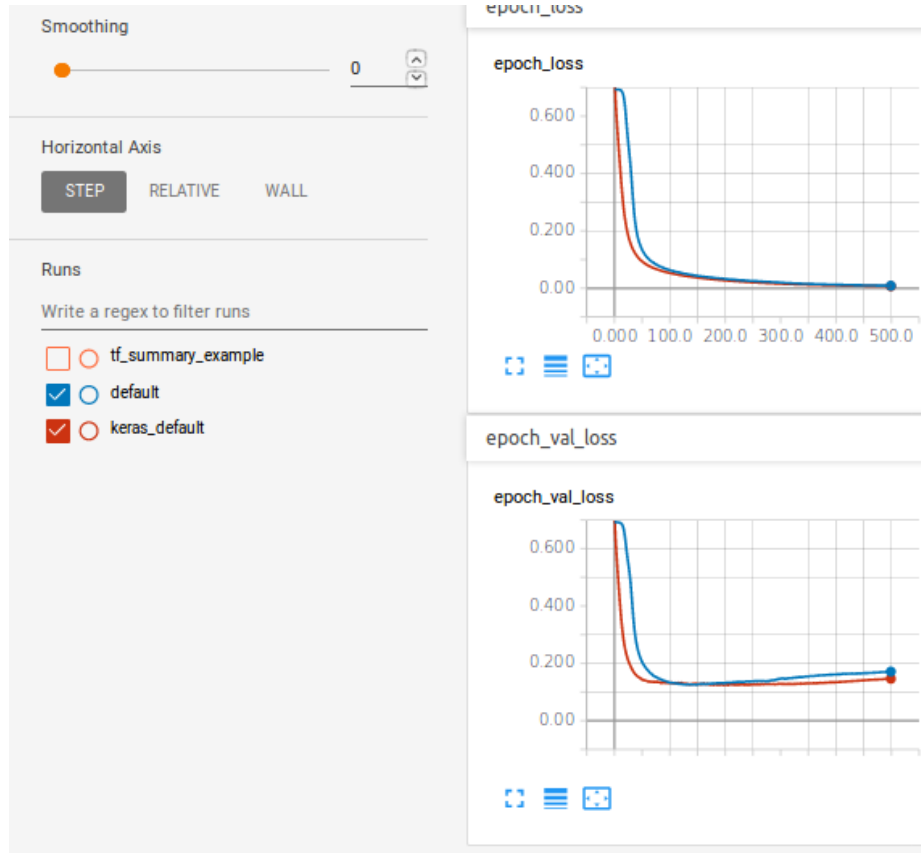
`loss = tf.losses.log_loss(x)`

```
# 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')
```

# Frameworks

*Worry less about best practices*

- Here: Initialization of model parameters

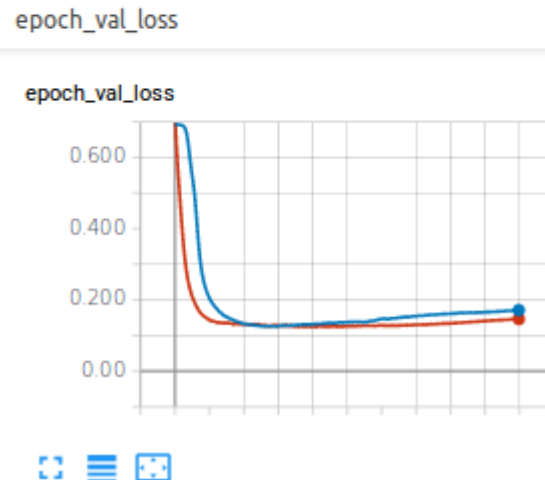
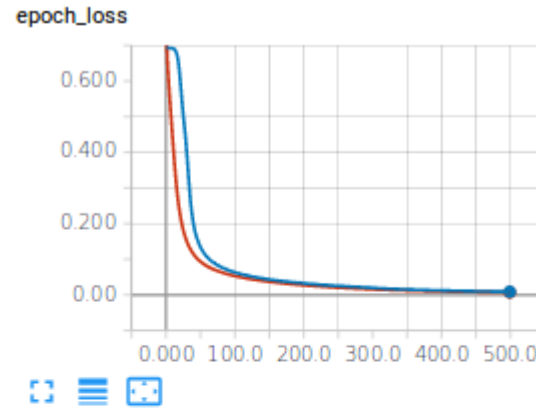


[illegible]

371 lines

# Frameworks

*Lots of convenience*

[illegible]

133 lines

# 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](https://www.tensorflow.org/api_docs/python/tf/keras) or  
<https://keras.io/getting-started/functional-api-guide/>

```
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
```

```
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

☒ Ignore outliers in chart scaling

Tooltip sorting method: default

Smoothing

0

Horizontal Axis

STEP

RELATIVE

WALL

Runs

Write a regex to filter runs

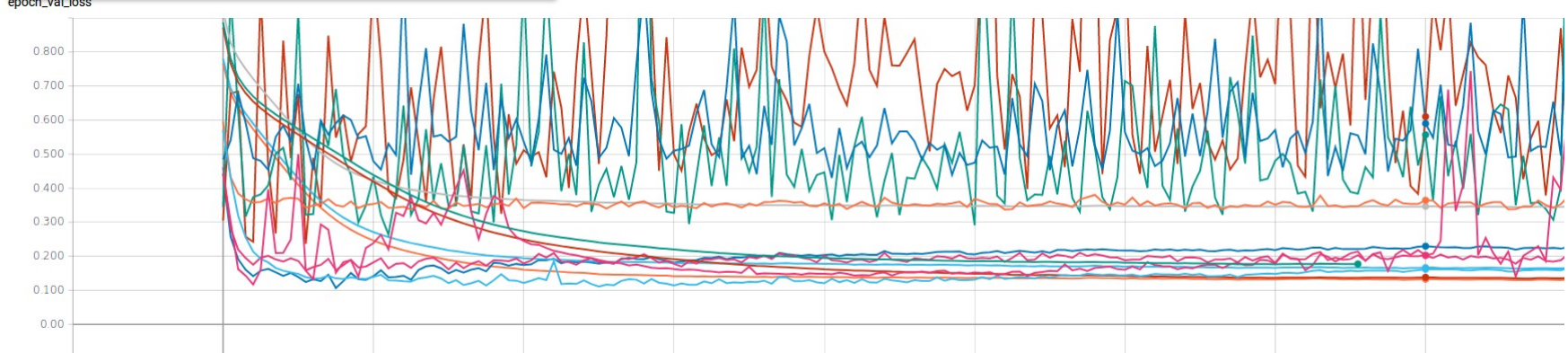
☒

modelV1\_bs[8]\_l2[1e-4]\_lr[1e-2]

☒

modelV1\_bs[8]\_l2[1e-4]\_lr[1e-1]☒☒☒☒☐☒☒☒☒☒☒

epoch_loss	Name	Smoothed	Value	Step	Time	Relative
	modelV1_bs[16]_l2[1e-3]_lr[1e-2]	0.1774	0.1774	151.0	Wed Nov 28, 08:41:45	23s
	modelV1_bs[16]_l2[1e-4]_lr[1]	0.2027	0.2027	160.0	Wed Nov 28, 08:41:13	25s
	modelV1_bs[16]_l2[1e-4]_lr[1e-1]	0.1620	0.1620	160.0	Wed Nov 28, 08:40:39	25s
	modelV1_bs[16]_l2[1e-4]_lr[1e-2]	0.1384	0.1384	160.0	Wed Nov 28, 08:40:06	25s
	modelV1_bs[8]_l2[1e-2]_lr[1]	0.5901	0.5901	160.0	Wed Nov 28, 08:39:28	48s
	modelV1_bs[8]_l2[1e-2]_lr[1e-1]	0.3645	0.3645	160.0	Wed Nov 28, 08:38:26	48s
	modelV1_bs[8]_l2[1e-2]_lr[1e-2]	0.3466	0.3466	160.0	Wed Nov 28, 08:37:23	48s
	modelV1_bs[8]_l2[1e-3]_lr[1]	0.5566	0.5566	160.0	Wed Nov 28, 08:36:19	50s
	modelV1_bs[8]_l2[1e-3]_lr[1e-1]	0.2026	0.2026	160.0	Wed Nov 28, 08:35:14	48s
	modelV1_bs[8]_l2[1e-3]_lr[1e-2]	0.1666	0.1666	160.0	Wed Nov 28, 08:34:10	48s
	modelV1_bs[8]_l2[1e-4]_lr[1]	0.6102	0.6102	160.0	Wed Nov 28, 08:33:06	48s
	modelV1_bs[8]_l2[1e-4]_lr[1e-1]	0.2296	0.2296	160.0	Wed Nov 28, 08:32:05	46s
	modelV1_bs[8]_l2[1e-4]_lr[1e-2]	0.1328	0.1328	160.0	Wed Nov 28, 08:31:06	43s



# Hyperparameter Tuning

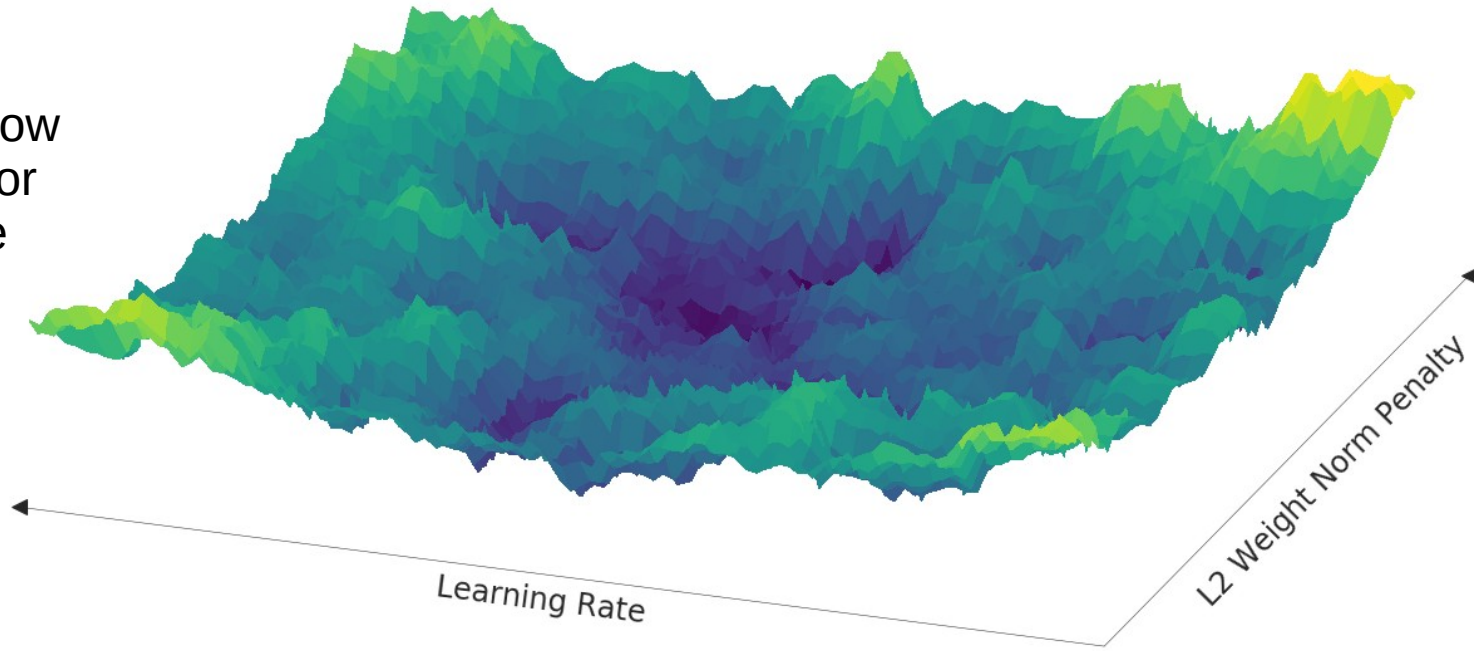
## *General Procedure*

- 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*

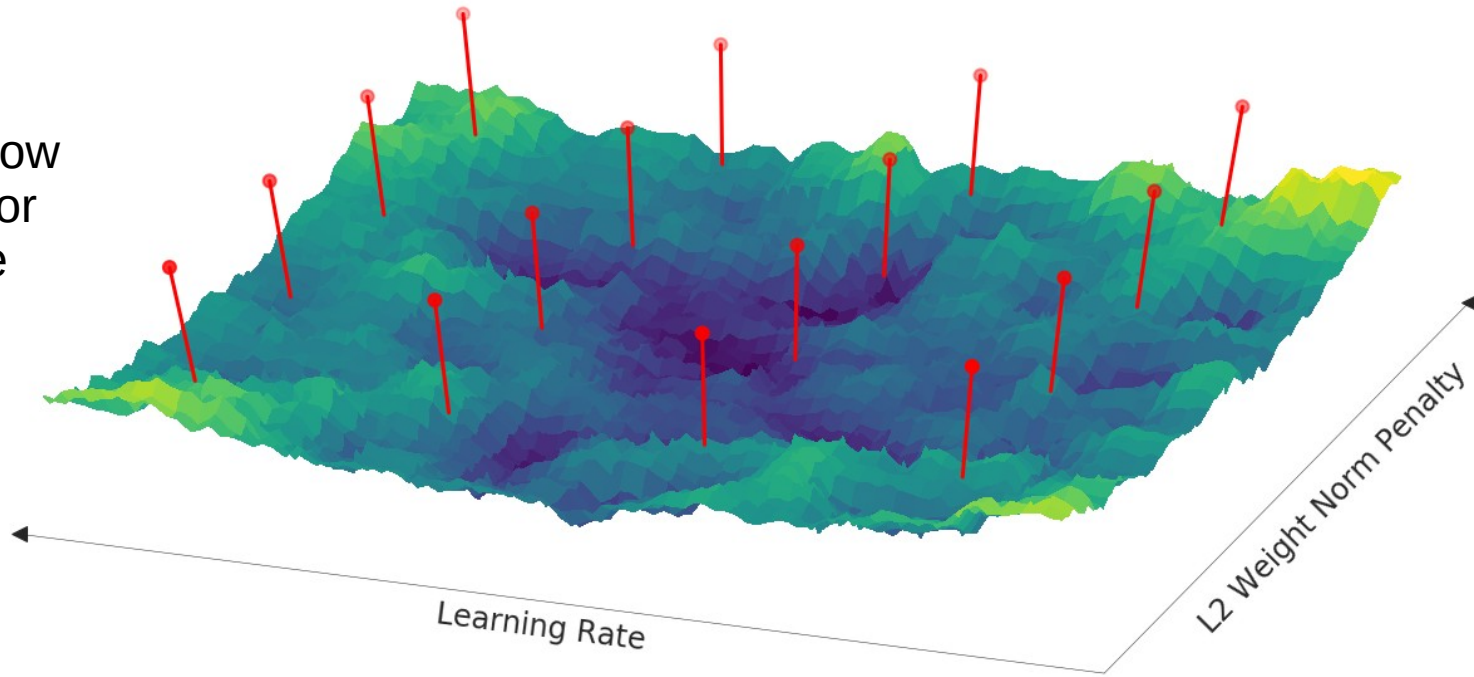
You don't know  
the true error  
landscape



# Hyperparameter Tuning

*Methods - GridSearch*

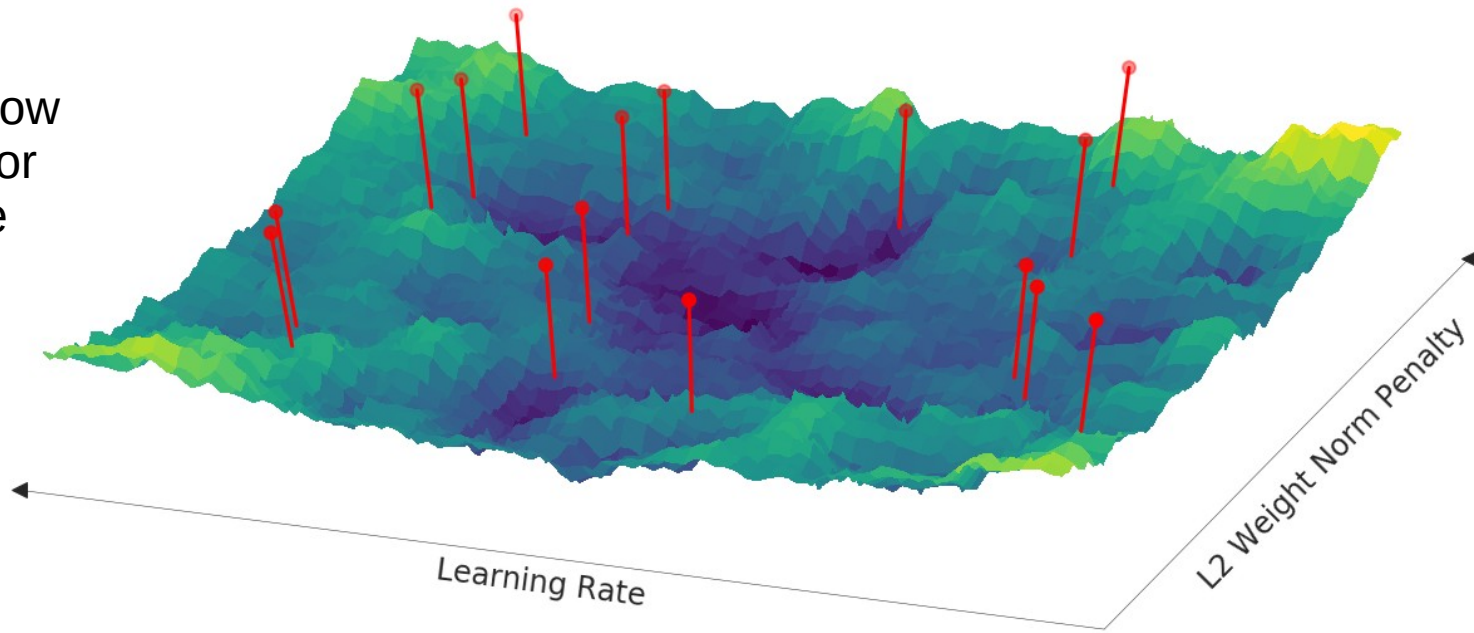
You don't know  
the true error  
landscape



# Hyperparameter Tuning

*Methods – Random Search*

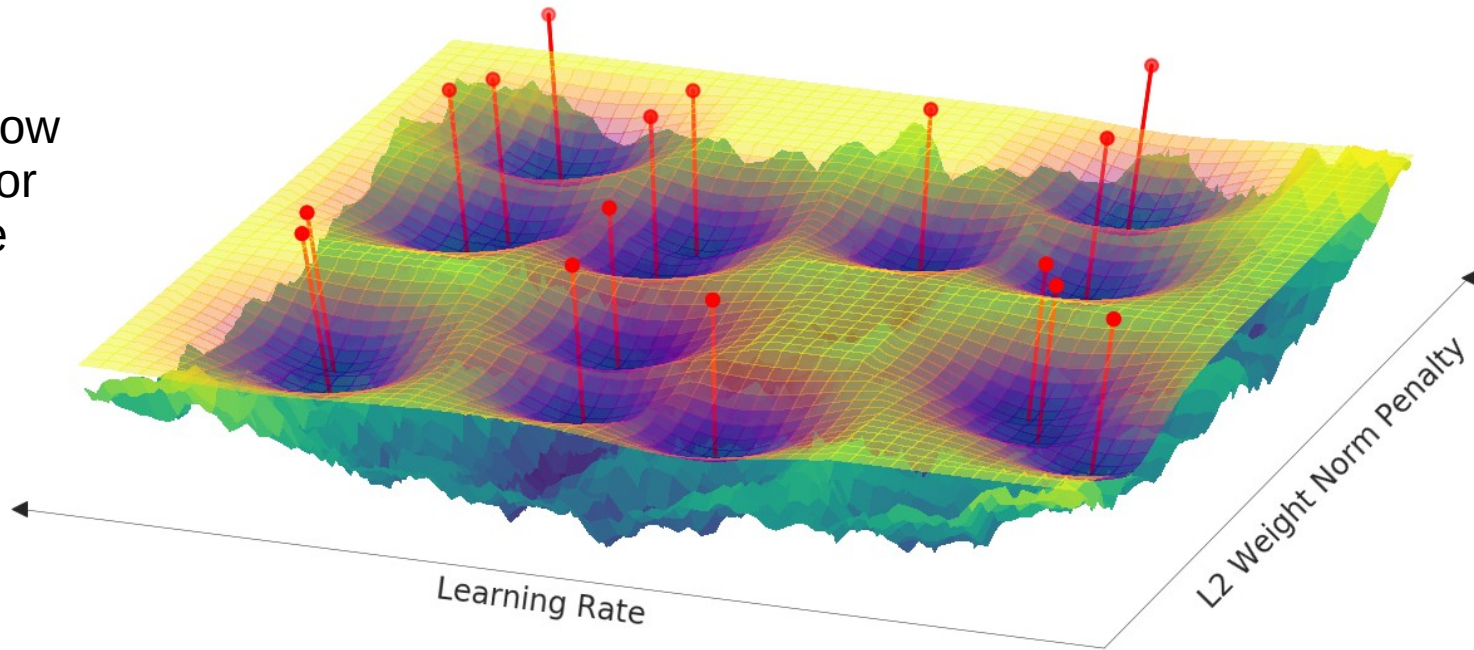
You don't know  
the true error  
landscape



# Hyperparameter Tuning

## *Methods- Bayesian Model Optimization*

You don't know  
the true error  
landscape

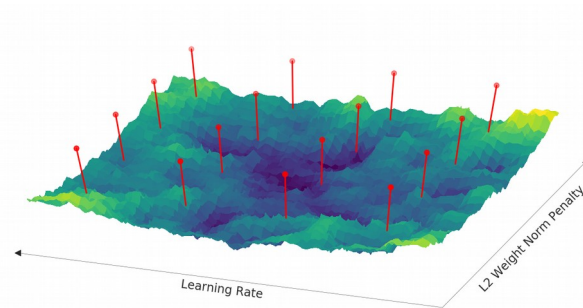


Intuition: Automate hyperparameter search by automatically choosing most promising candidates based on past experience.

# Hyperparameter Tuning

## *The naive way*

```
1 """Simple way to call python scripts."""
2 import os
3 import subprocess
4
5
6 num_epochs = 200
7 batch_size = 8
8 path = './experiments'
9
10 try:
11     for l2_factor in [1e-4, 1e-3, 1e-2]:
12         for learning_rate in [1e-2, 1e-1, 1]:
13             subprocess.call([
14                 'python', 'keras_example.py',
15                 '-ep', str(num_epochs),
16                 '-bs', str(batch_size),
17                 '-lr', str(learning_rate),
18                 '-l2', str(l2_factor),
19                 '-d', os.path.join(
20                     path, 'modelV1_lr[%s]_l2[%s]'
21                     % (learning_rate, l2_factor))]
22             )
23 except subprocess.CalledProcessError as e:
24     print(e)
```



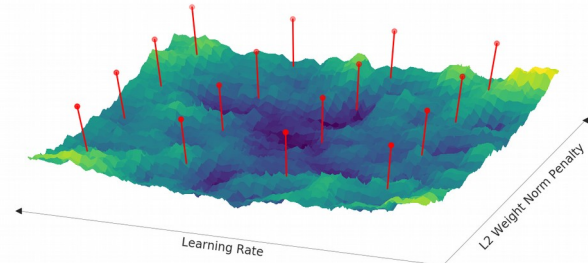
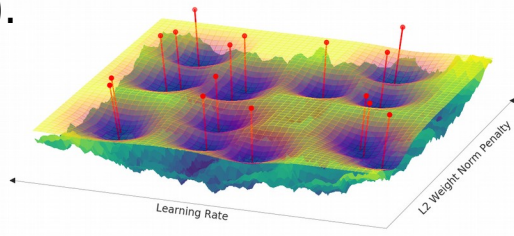
Experiment runs are automated but they are executed sequentially.

```
(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]
```

# 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).



```
1  """Ray can be used to parallelize our grid search implementation."""
2  import ray
3
4  from tooling_lecture.hypertune import keras_example
5
6
7  ray.init()
8
9
10 @ray.remote
11 def keras_example_main(num_epochs, batch_size, l2_factor, learning_rate,
12                       experiment_dir):
13     """A wrapper for the main function of the keras example."""
14     keras_example.main(
15         num_epochs, batch_size, l2_factor, learning_rate, experiment_dir,
16         debug=False)
17
18
19 object_ids = []
20 for i, l2_factor in enumerate([1e-5, 1e-4, 1e-3]):
21     for learning_rate in [1e-2, 1e-1, 1]:
22         object_ids += [keras_example_main.remote(
23             200, 8, l2_factor, learning_rate,
24             experiment_dir=(
25                 './experiments/ray_modelV1_lr[%s]_l2[%s]'
26                 % (learning_rate, l2_factor)))]
27         print(object_ids[-1])
28
29
30 # To make sure that the ray stays alive until all processes are finished.
31 ray.get(object_ids)
```

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

# 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:*



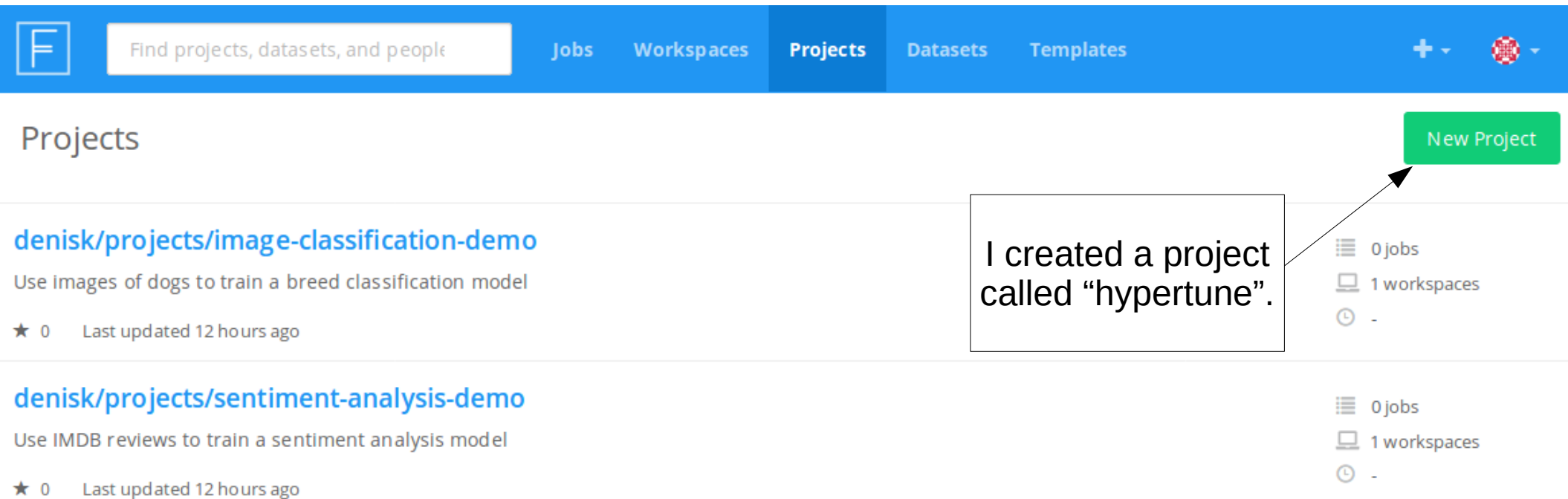
# Hyperparameter Tuning

## *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 syncing)
      - **\$ floyd init <your project name>**
    - 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.
- You can extend this.
- 

# Hyperparameter Tuning

## *Cloud Solutions- Simple Approach*



The screenshot shows the Databricks web interface. At the top is a blue navigation bar with a logo on the left, a search bar containing 'Find projects, datasets, and people', and several tabs: 'Jobs', 'Workspaces', 'Projects' (which is highlighted), 'Datasets', and 'Templates'. On the right side of the navigation bar are a plus icon and a user profile icon. Below the navigation bar, the main heading 'Projects' is displayed. A green 'New Project' button is located in the top right corner. Below this, a list of projects is shown. The first project is 'denisk/projects/image-classification-demo' with the description 'Use images of dogs to train a breed classification model', 0 stars, and 'Last updated 12 hours ago'. The second project is 'denisk/projects/sentiment-analysis-demo' with the description 'Use IMDB reviews to train a sentiment analysis model', 0 stars, and 'Last updated 12 hours ago'. To the right of each project entry, there are statistics: '0 jobs', '1 workspaces', and a clock icon with a dash. A white text box with a black border is positioned over the first project entry, containing the text 'I created a project called "hypertune"'. An arrow points from this text box to the 'New Project' button.

Find projects, datasets, and people

Jobs Workspaces **Projects** Datasets Templates

Projects

[denisk/projects/image-classification-demo](#)  
Use images of dogs to train a breed classification model  
★ 0 Last updated 12 hours ago

[denisk/projects/sentiment-analysis-demo](#)  
Use IMDB reviews to train a sentiment analysis model  
★ 0 Last updated 12 hours ago

**New Project**

I created a project called "hypertune".

0 jobs  
1 workspaces  
-

0 jobs  
1 workspaces  
-

# Hyperparameter Tuning

## Cloud Solutions- Simple Approach

```
(dev-p36) /hypertune(master)$ floyd login
Waiting for login from browser...
Login Successful as denisk
(dev-p36) /hypertune(master)$ floyd init hypertune
Project "hypertune" initialized in current directory
(dev-p36) /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
```



The screenshot shows the Floyd Hub web interface. At the top is a blue navigation bar with the Floyd logo, a search bar, and tabs for Jobs, Workspaces, Projects, Datasets, and Templates. Below the navigation bar are two filter boxes: "Filter by job state..." and "Filter by tags...". On the right, a blue button indicates "1 Running" job. The main content area displays a job card for "denisk/projects/hypertune/2". The job is in a "Running" state, indicated by a blue play button icon. It was submitted 24 seconds ago and has no description. There are buttons for "Shutdown" and "Manage tags". On the right side of the job card, there are icons for "CPU", a timer showing "0:00:24", and a "Cli" icon.

# Hyperparameter Tuning

## Cloud Solutions- Simple Approach

The screenshot shows a cloud workspace interface. At the top, there is a blue header bar with a search bar containing 'Find projects, datasets, and people' and navigation links for 'Jobs', 'Workspaces', 'Projects', 'Datasets', and 'Templates'. Below the header, the breadcrumb path 'denisk / projects / hypertune / 2' is displayed, followed by a green 'Success' button. A secondary navigation bar includes 'Overview', 'Files', 'Input Data', and 'Settings'. The main content area shows a folder 'denisk/projects/hypertune/2/home' with a size of 54.33 MB and a timestamp 'Last updated 3 minutes ago'. To the right are buttons for '+ Create Dataset', 'Download', and 'Deploy'. Below this, a list of files is shown: 'experiments' (folder), 'keras\_example.py' (5.2 KB), 'floyd.yml' (479 B), and '\_\_init\_\_.py' (0 B). A central box labeled 'Uploaded files + new files (results)' has arrows pointing to each of these four items.

denisk / projects / hypertune / 2

Success

Overview Files Input Data Settings

denisk/projects/hypertune/2/home | 54.33 MB | Last updated 3 minutes ago

+ Create Dataset Download Deploy

experiments	
keras_example.py	5.2 KB
floyd.yml	479 B
__init__.py	0 B

Uploaded files  
+  
new files (results)

# 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

**Cloud ML Engine**

**Python SDK**



**SageMaker**

**Python SDK**



**Machine Learning service**

**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.

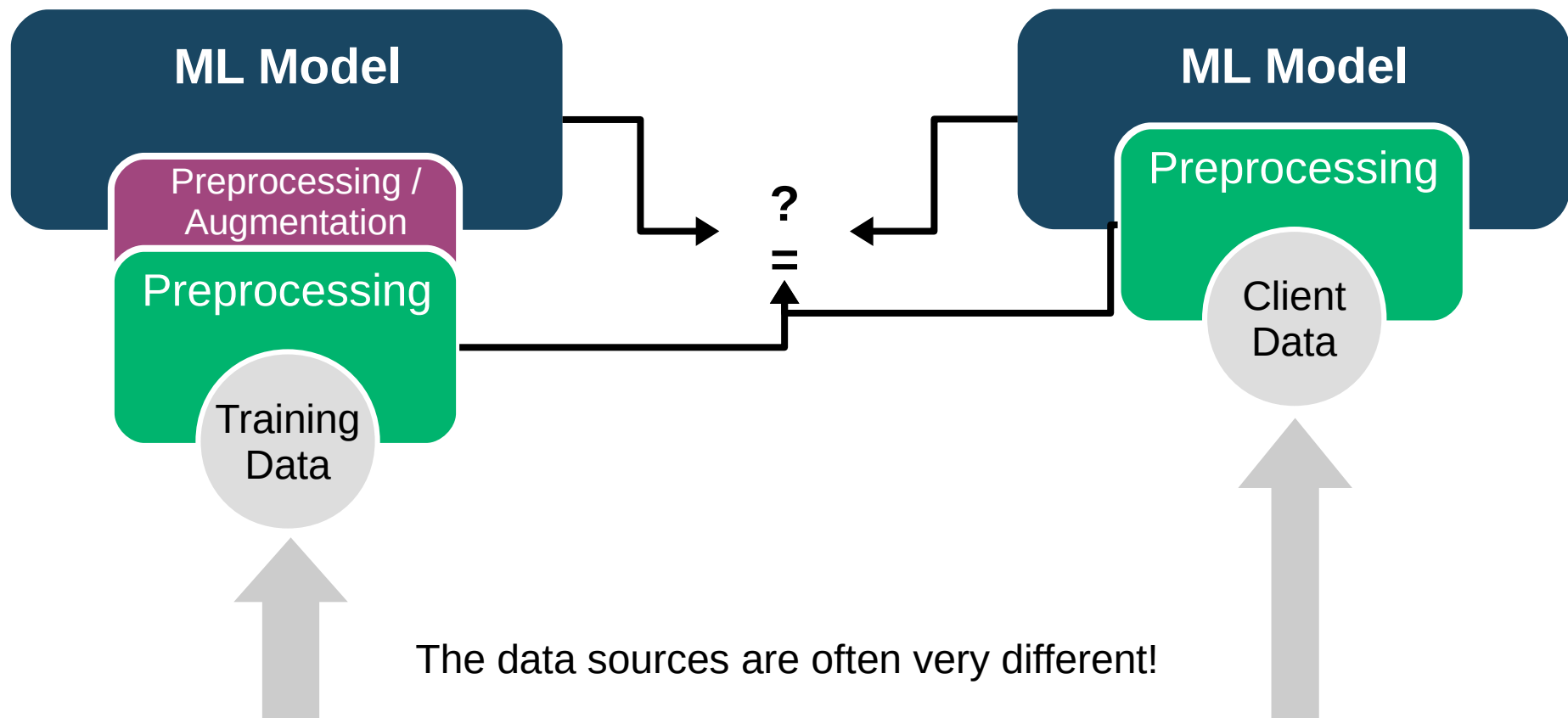
<https://hackernoon.com/a-brief-overview-of-automatic-machine-learning-solutions-automl-2826c7807a2a>

**DEPLOY**

# Deployment

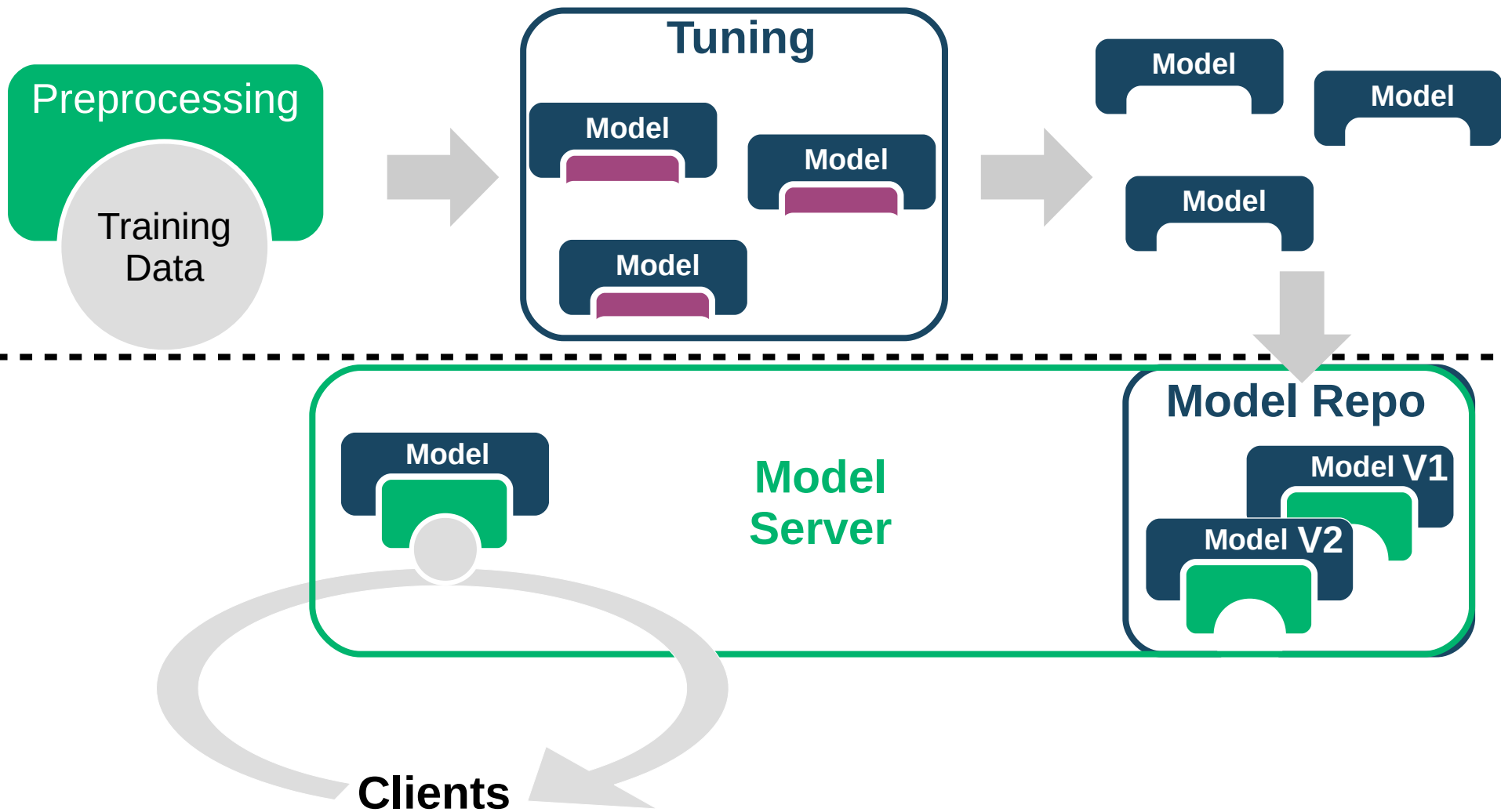
## Model Training

## Model Inference



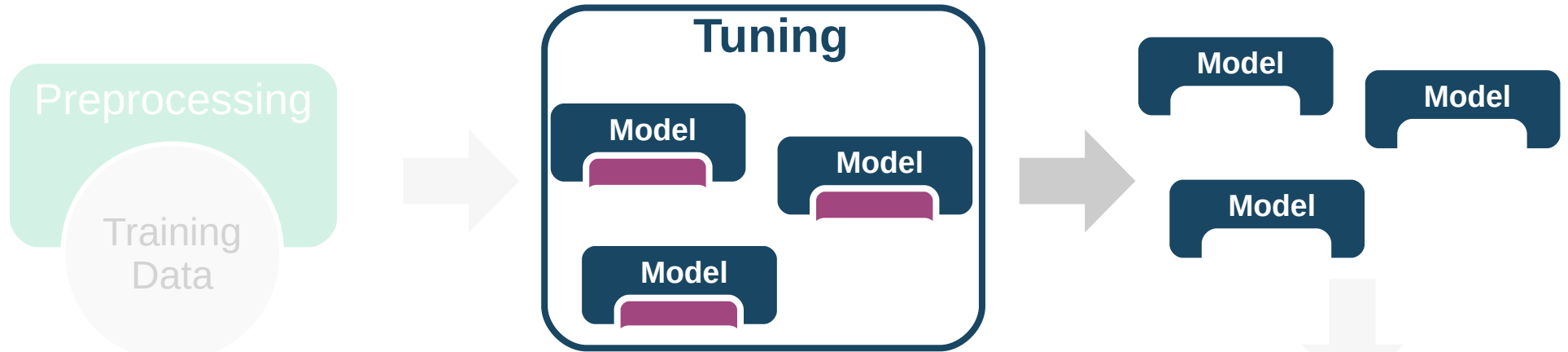
# Deployment

*Model Serving*



# Deployment

## *Model Serving – Tensorflow Serving*



Model

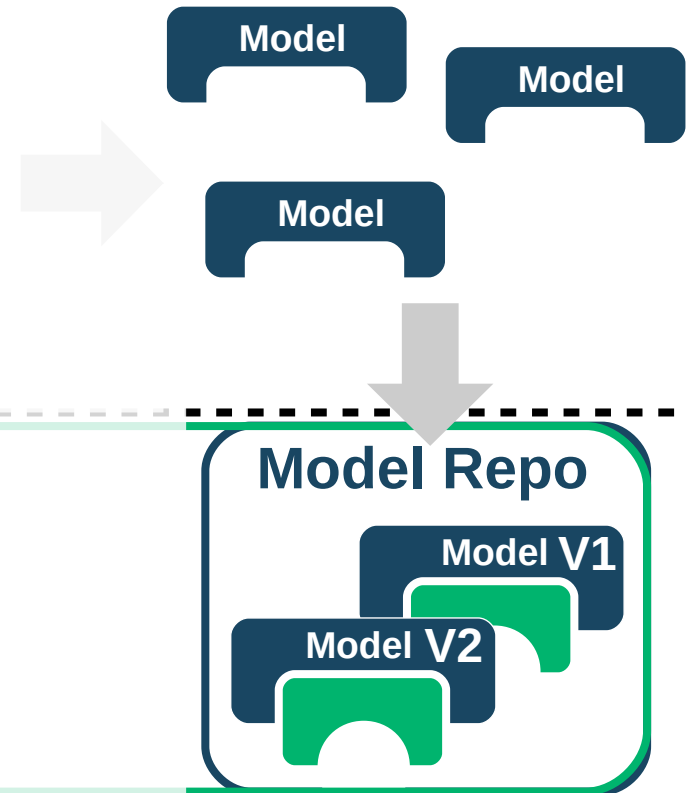
```
52 # For saving the model.  
53 checkpointing_callback = tf.keras.callbacks.ModelCheckpoint(  
54     os.path.join(experiment_dir, 'model.h5'),  
55     monitor='val_loss',  
56     save_best_only=True,  
57     mode='min')  
58 callbacks = [tensorboard_callback, checkpointing_callback]  
59  
60 # Train the model on the data.  
61 model.fit(x=[x1, x2], y=y, batch_size=batch_size, epochs=num_epochs,  
62     validation_data=([x1_val, x2_val], y_val), verbose=2,  
63     callbacks=callbacks)
```

Clients

# Deployment

## Model Serving – Tensorflow Serving

```
1  """Load a keras model and save it in a format understood by tf serving."""
2  import tensorflow as tf
3
4
5  # Load the keras model from disc.
6  model = tf.keras.models.load_model(
7      './experiments/keras_example/model.h5')
8
9  # Set the export path. Tensorflow serving takes the last directory name as the
10 # version of the model.
11 export_path = './production_models/keras_example/1'
12 builder = tf.saved_model.builder.SavedModelBuilder(export_path)
13
14 # Create a signature definition for tf-serving.
15 # We will use the predict API which allows us to have an arbitrary number of
16 # inputs and outputs.
17 model_signature = tf.saved_model.signature_def_utils.build_signature_def(
18     inputs={tensor.name: tf.saved_model.utils.build_tensor_info(tensor)
19             for tensor in model.inputs},
20     outputs={tensor.name: tf.saved_model.utils.build_tensor_info(tensor)
21             for tensor in model.outputs},
22     method_name=tf.saved_model.signature_constants.PREDICT_METHOD_NAME)
23
24 # Serialize the model.
25 with tf.keras.backend.get_session() as session:
26     builder.add_meta_graph_and_variables(
27         session,
28         [tf.saved_model.tag_constants.SERVING], # This is just a tag.
29         signature_def_map={
30             'predict_whatever':
31                 model_signature,
32         })
33
34 # Export the model to the production_models/1 folder.
35 builder.save(as_text=True)
```



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

# Deployment

## *Model Serving – Tensorflow Serving*

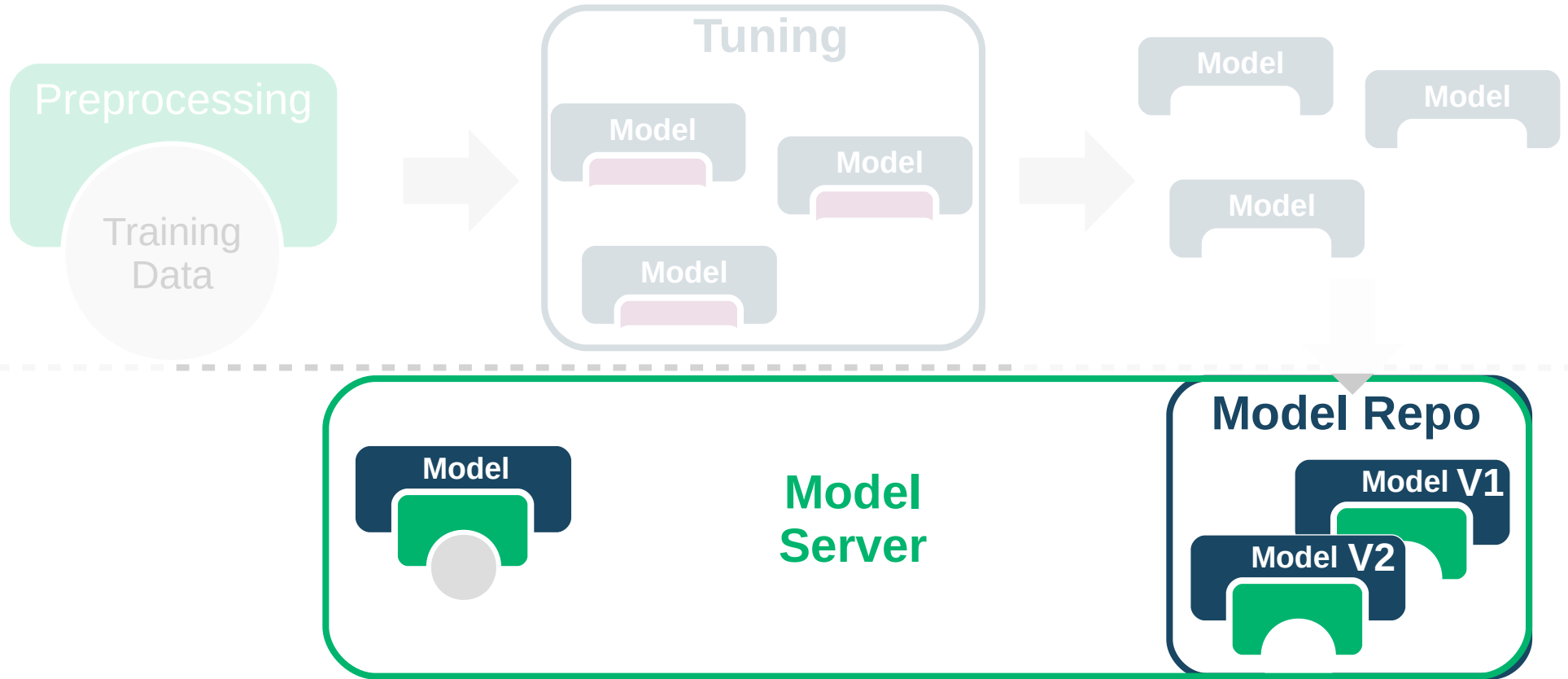
```
(dev-p36) /deployment(master)$ saved_model_cli show --dir=./production_models/keras_example/1 --all
MetaGraphDef with tag-set: 'serve' contains the following SignatureDefs:

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

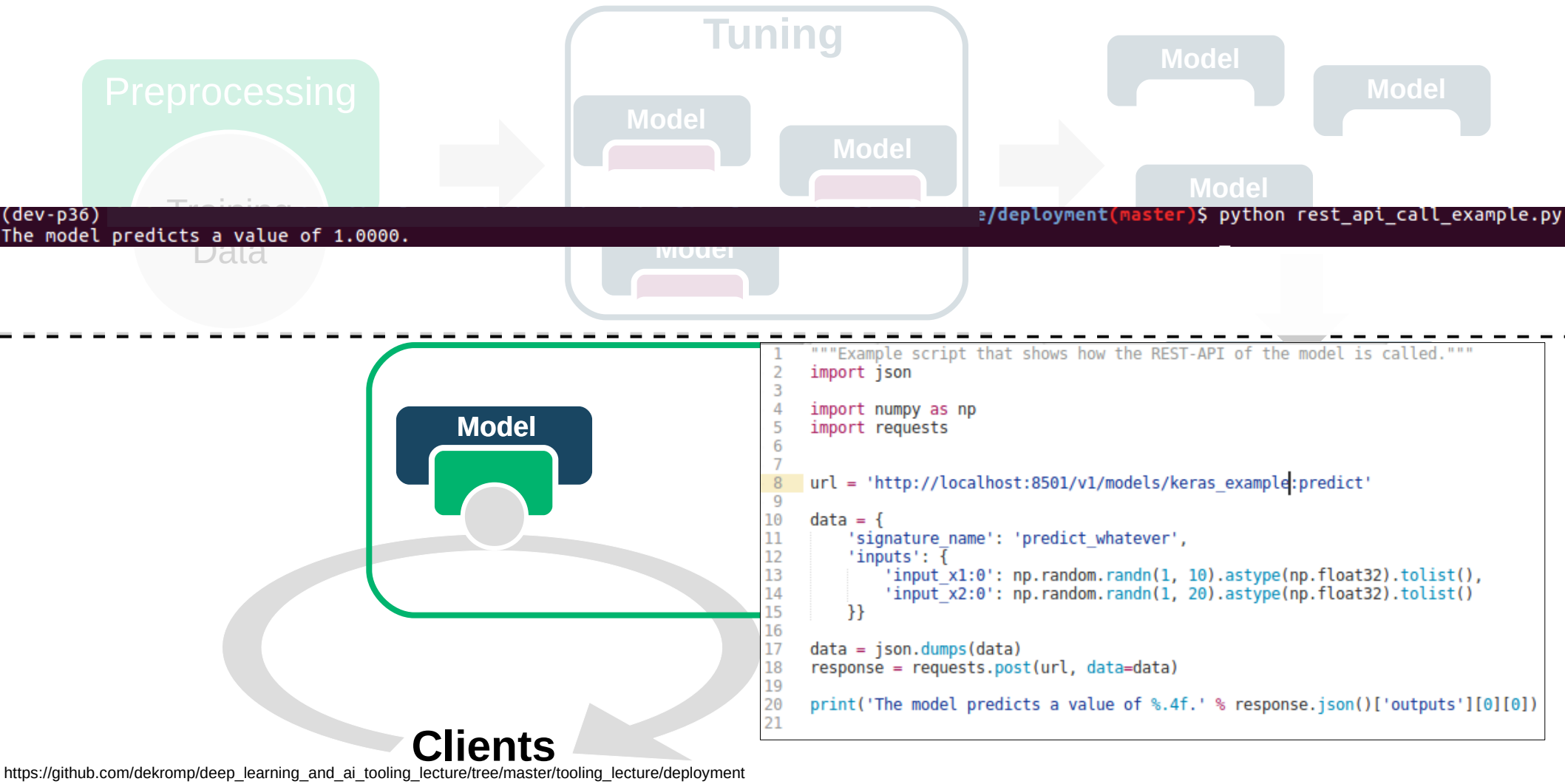
## *Model Serving – Tensorflow Serving*



```
1 #!/bin/bash
2 # Run tensorflow model server and serve our model.
3 sudo docker run -p 8501:8501 -v $(pwd)/production_models:/models/ -e MODEL_NAME=keras_example --rm -t tensorflow/serving
4
```

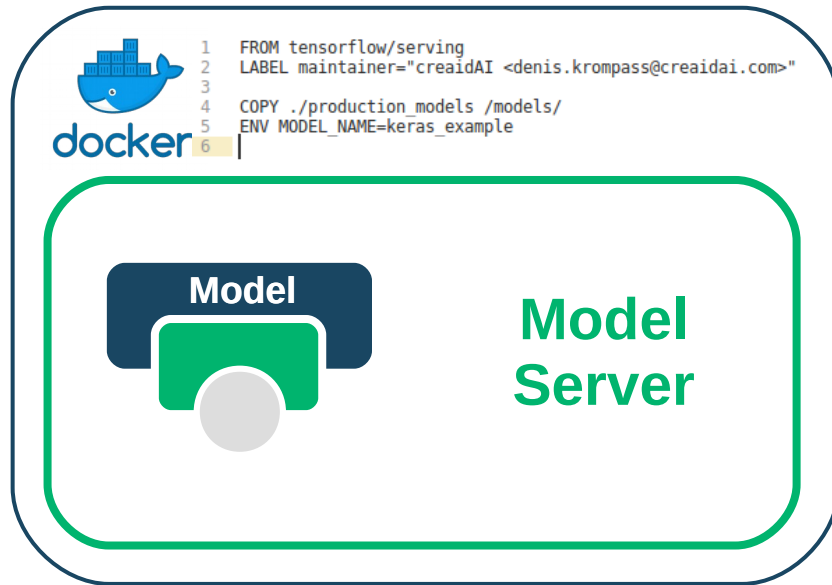
# Deployment

## Model Serving – Tensorflow Serving



# Deployment

## *Model Serving – Scaling up*



# 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!