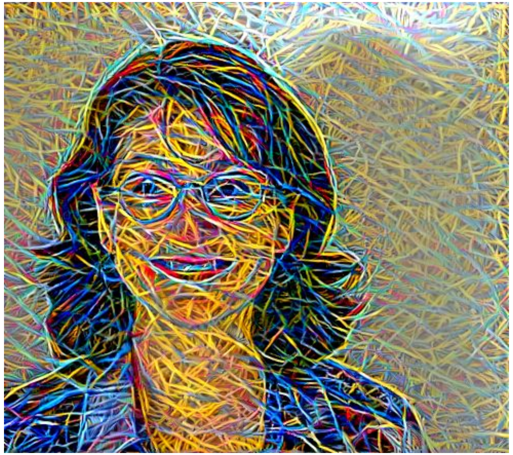
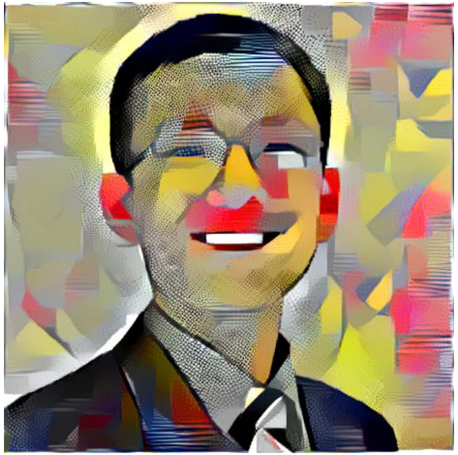
TensorFlow on Cloud ML

January 12, 2017
AI Frontiers

Amy Unruh, Eli Bixby, Yufeng Guo

Your guides





Amy
amyu@google.com
@amygdala

Eli
elibixby@google.com
@eli_bixby

Yufeng
yfg@google.com
@YufengG

Welcome and Logistics

Slides: **http://bit.ly/tf-workshop-slides**
Alternate link:
https://storage.googleapis.com/amy-jo/talks/tf-workshop.pdf

GitHub: https://github.com/amygdala/tensorflow-workshop

Agenda

- **Intro**
  - Setup time, Introduction to Tensorflow and Cloud ML
- Warmup: **XOR**
- **Wide and Deep**
  - Use the “tf.learn” API to jointly train a wide linear model and a deep feed-forward neural network.
- **Word2vec**
  - Custom Estimators, learning and using word embeddings, and the embeddings visualizer
- **Transfer learning** and **online prediction**
  - learn your own image classifications by bootstrapping the *Inception v3* model, then use the Cloud ML API for prediction

http://bit.ly/tf-workshop-slides

bit.ly/tensorflow-workshop

TensorFlow on Cloud ML Workshop Setup

- For a cloud-based setup, follow these instructions: http://bit.ly/aifrontiers-cloudml-install

If you’re done, visit playground.tensorflow.org

Or to set up on your laptop:

- Clone or download this repo: https://github.com/amygdala/tensorflow-workshop

- Follow the installation instructions in `INSTALL.md`.
  You can run the workshop exercises in a **Docker** container, or alternately install and use a **virtual environment**.

Workshop Setup addendum

- If you set up a docker container yesterday, then **in the container**, do:
  - `cd`
  - `python download_git_repo.py`

What’s TensorFlow?

(and why is it so great for ML?)

TensorFlow

- Open source Machine Learning library
- Especially useful for **Deep Learning**
- For research **and** production
- **Apache 2.0** license
- tensorflow.org

Open Source Models

github.com/tensorflow/models

Inception

An Alaskan Malamute (left) and a Siberian Husky (right). Images from Wikipedia.

https://research.googleblog.com/2016/08/improving-inception-and-image.html

Show and Tell

https://research.googleblog.com/2016/09/show-and-tell-image-captioning-open.html

Text Summarization

Original text

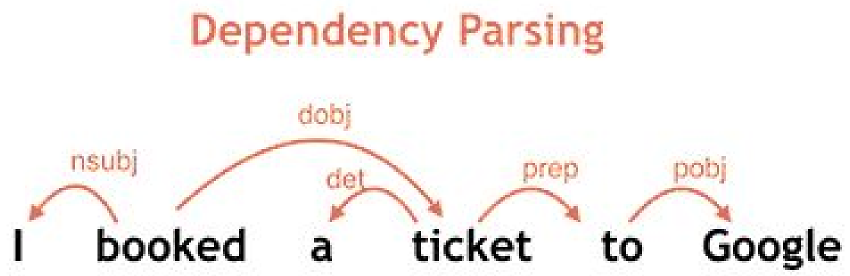
- *Alice and Bob took the train to visit the zoo. They saw a **baby giraffe, a lion, and a flock of colorful tropical birds**.*

Abstractive summary

- *Alice and Bob visited the zoo and saw **animals and birds**.*

https://research.googleblog.com/2016/08/text-summarization-with-tensorflow.html

Parsey McParseface



https://research.googleblog.com/2016/05/announcing-syntaxnet-worlds-most.html

What’s TensorFlow?

A multidimensional array.

TensorFlow

A graph of operations.

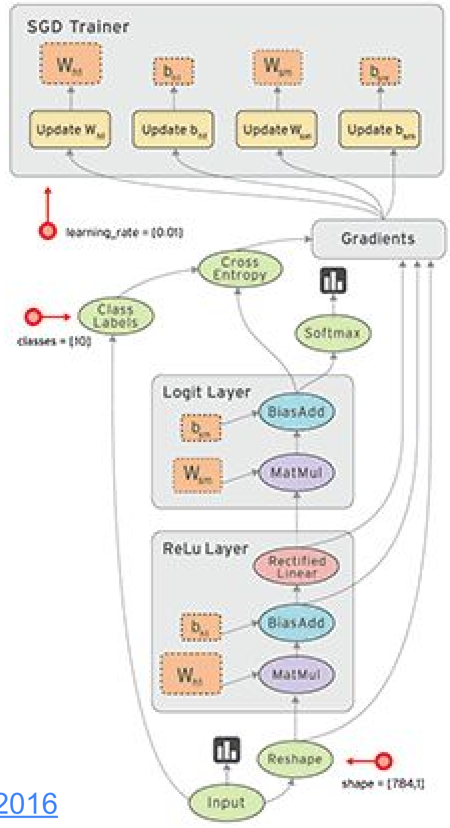
TensorFlow

A multidimensional array.

A graph of operations.

Operates over **tensors**: *n-dimensional arrays*
Using a **flow graph**: *data flow computation framework*

- Flexible, intuitive construction
- automatic differentiation
- Support for threads, queues, and asynchronous computation; [distributed runtime]
- Train on CPUs, GPUs, ...and coming soon, **TPUS**...
- Run wherever you like



https://cloudplatform.googleblog.com/2016/05/Google-supercharges-machine-learning-tasks-with-custom-chip.html

Tensors - generalized matrices

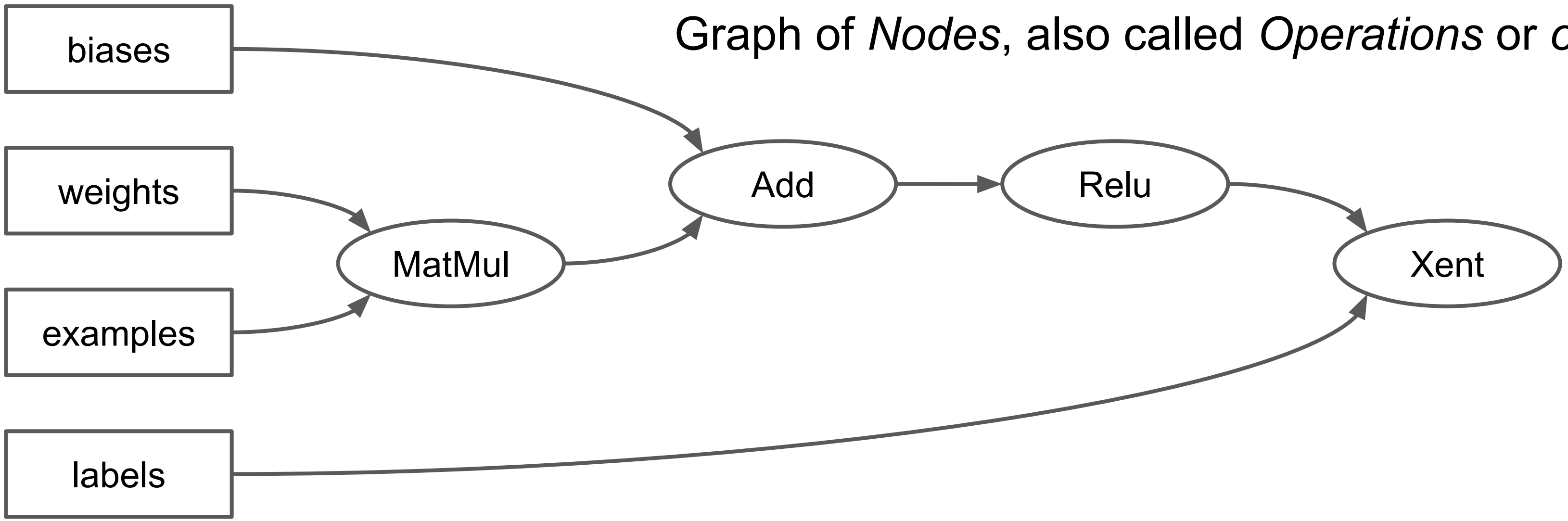
Tensors have a Shape that's described with a *vector*.

[10000, 256, 256, 3]

- 10000 Images
- Each Image has 256 Rows
- Each Row has 256 Pixels
- Each Pixel has 3 channels (RGB)

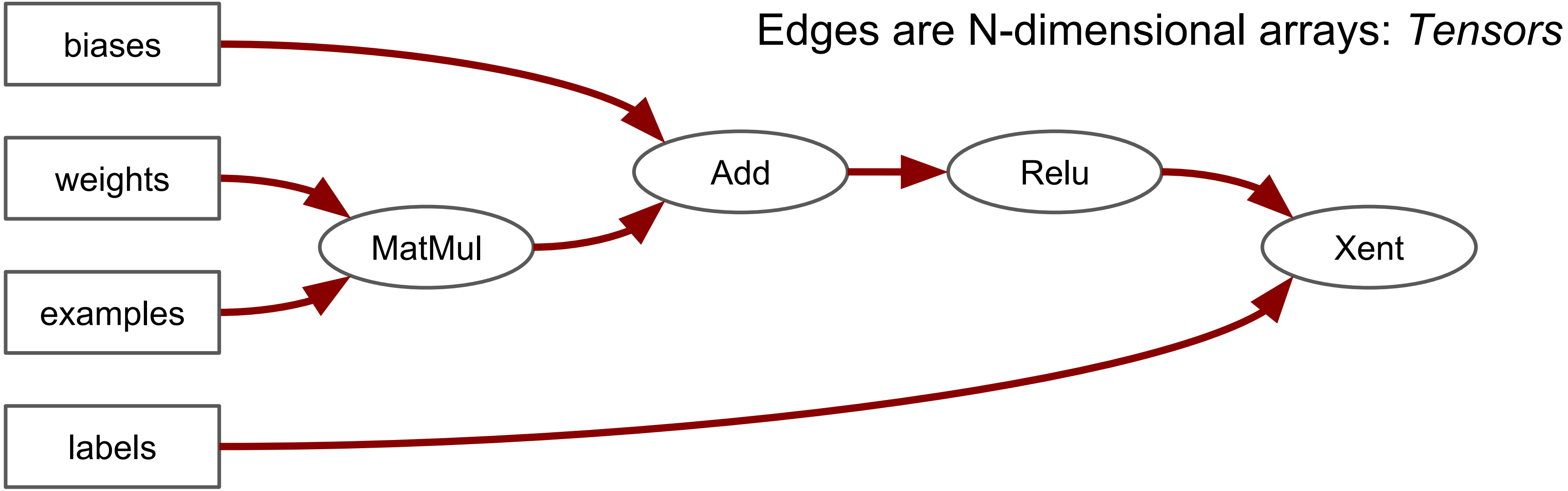
Computation is a dataflow graph

Graph of *Nodes*, also called *Operations* or *ops.*



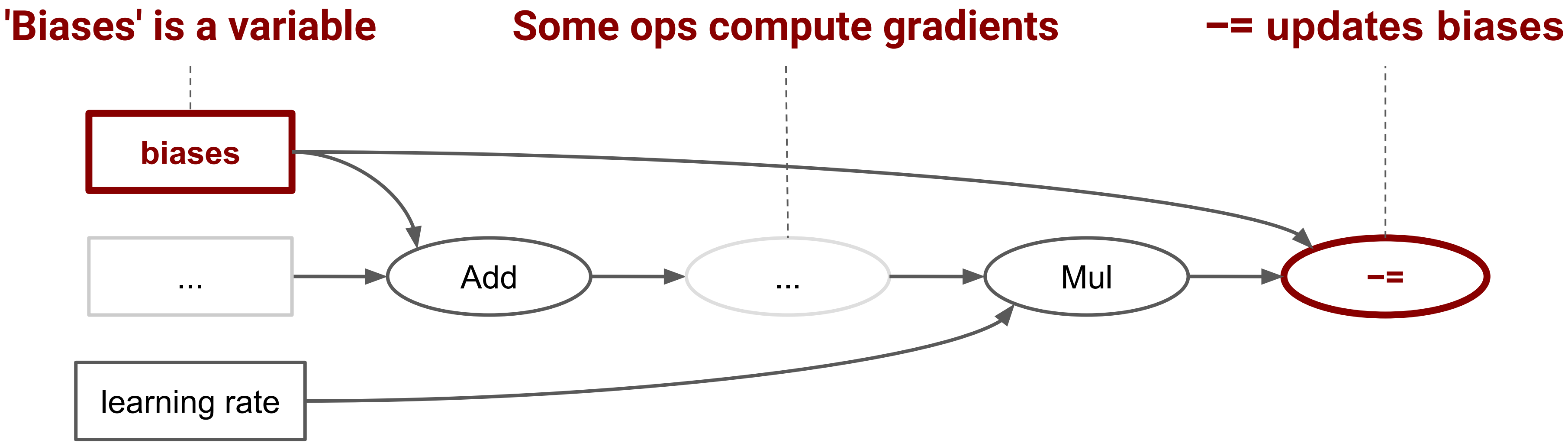
Computation is a dataflow graph

with tensors



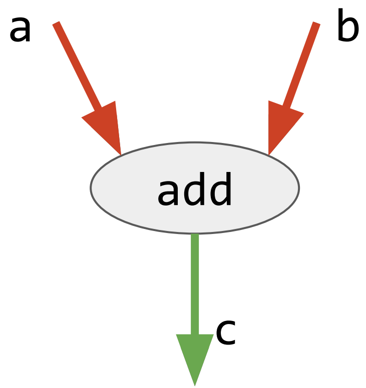
Computation is a dataflow graph

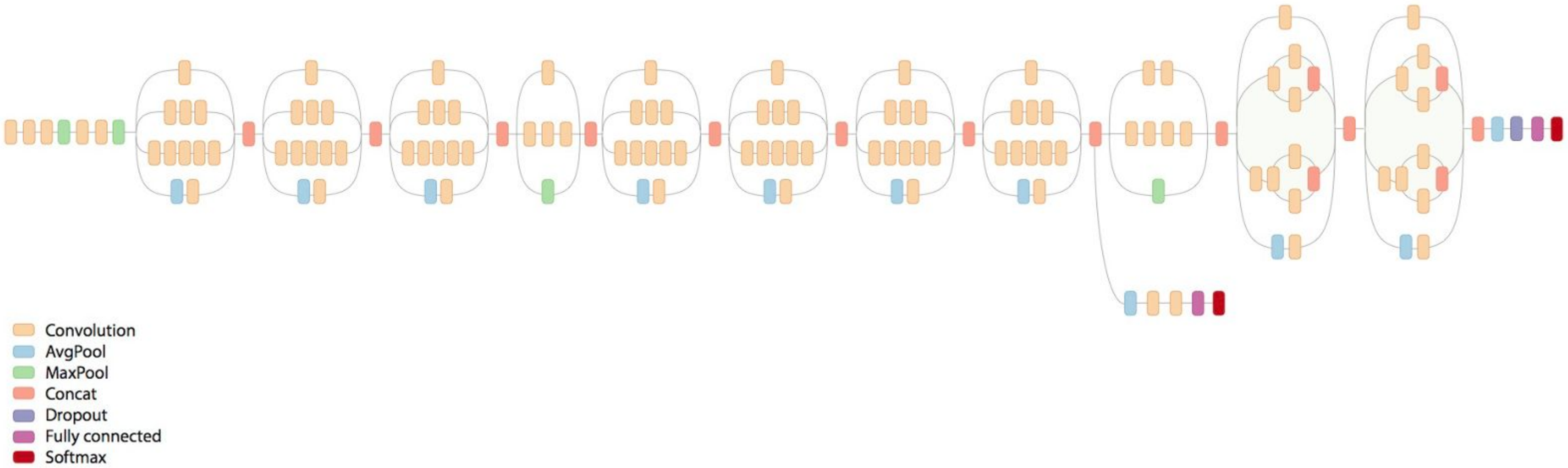
with state



Build a graph; *then* run it.

```
...
c = tf.add(a, b)

...

session = tf.Session()
value_of_c = session.run(c, {a=1, b=2})
```

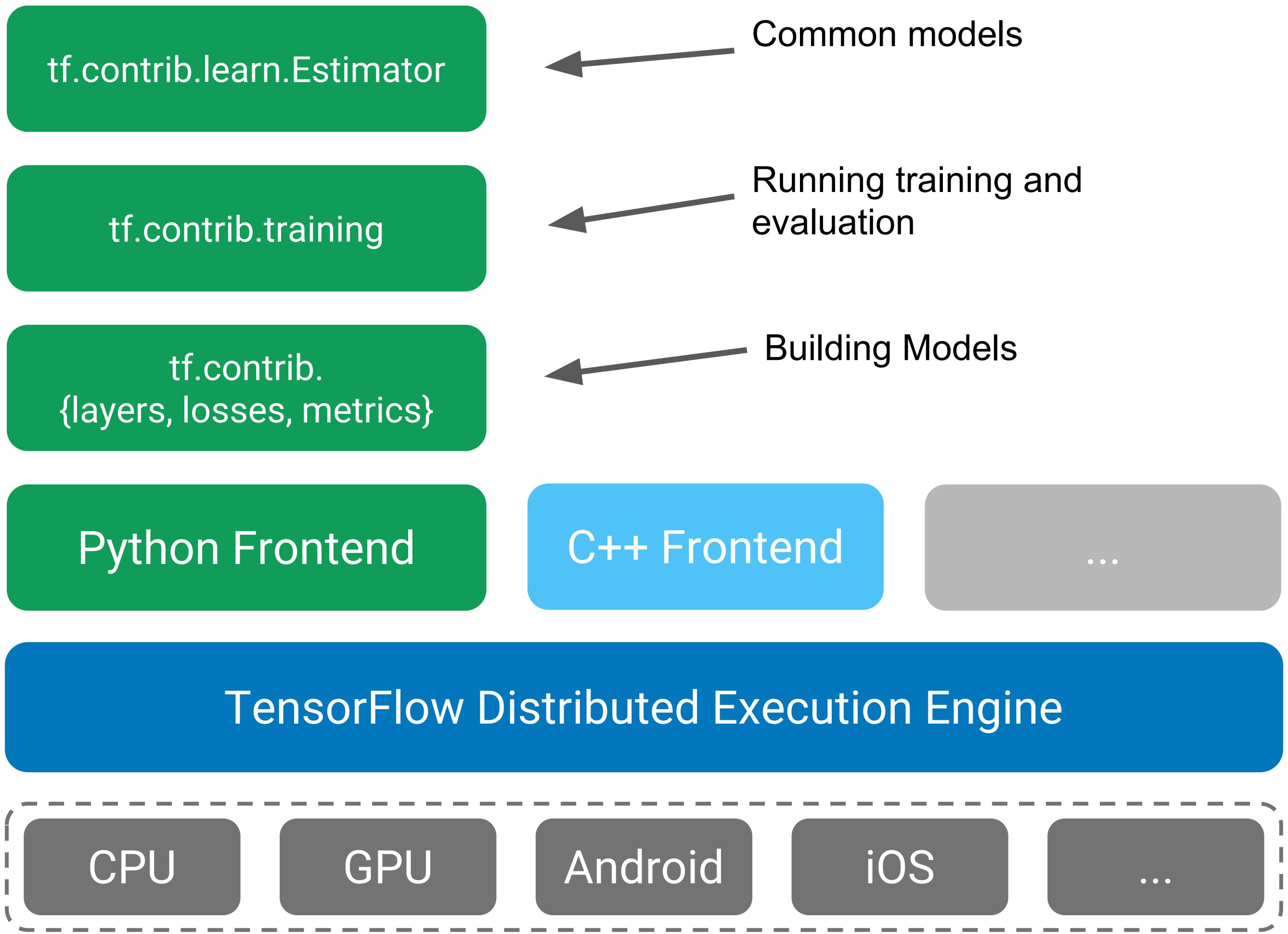




From: http://googleresearch.blogspot.com/2016_03_01_archive.html

http://bit.ly/tf-workshop-slides

bit.ly/tensorflow-workshop


tf.contrib.learn.Estimator
Common models
tf.contrib.training
Running training and evaluation
tf.contrib.
{layers, losses, metrics}
Building Models
Python Frontend
C++ Frontend
...
TensorFlow Distributed Execution Engine
CPU
GPU
Android
iOS
...

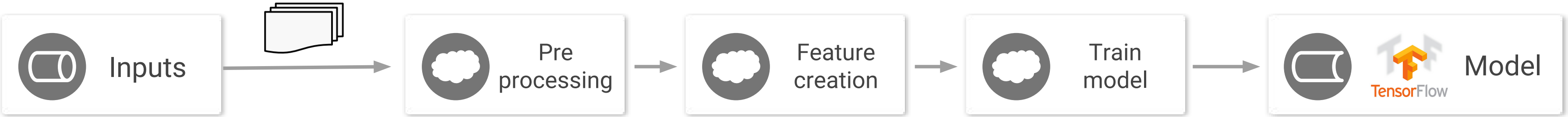
TensorFlow API Documentation:
https://www.tensorflow.org/api_docs/

Cloud ML: Scaling TensorFlow

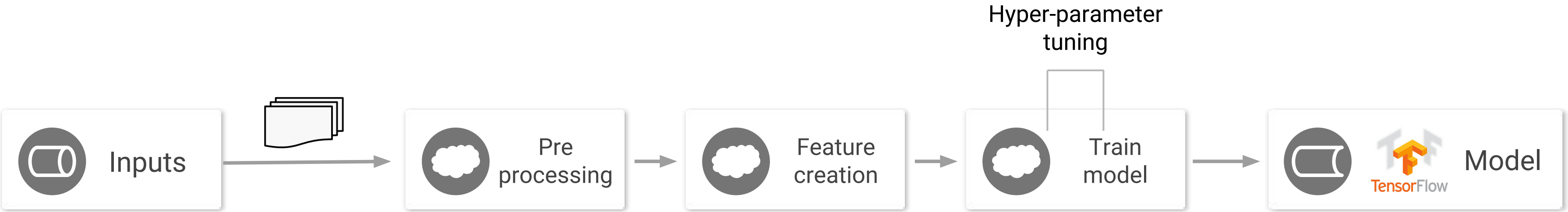
Many machine-learning frameworks can handle toy problems

As your data size increases, batching and distribution become important

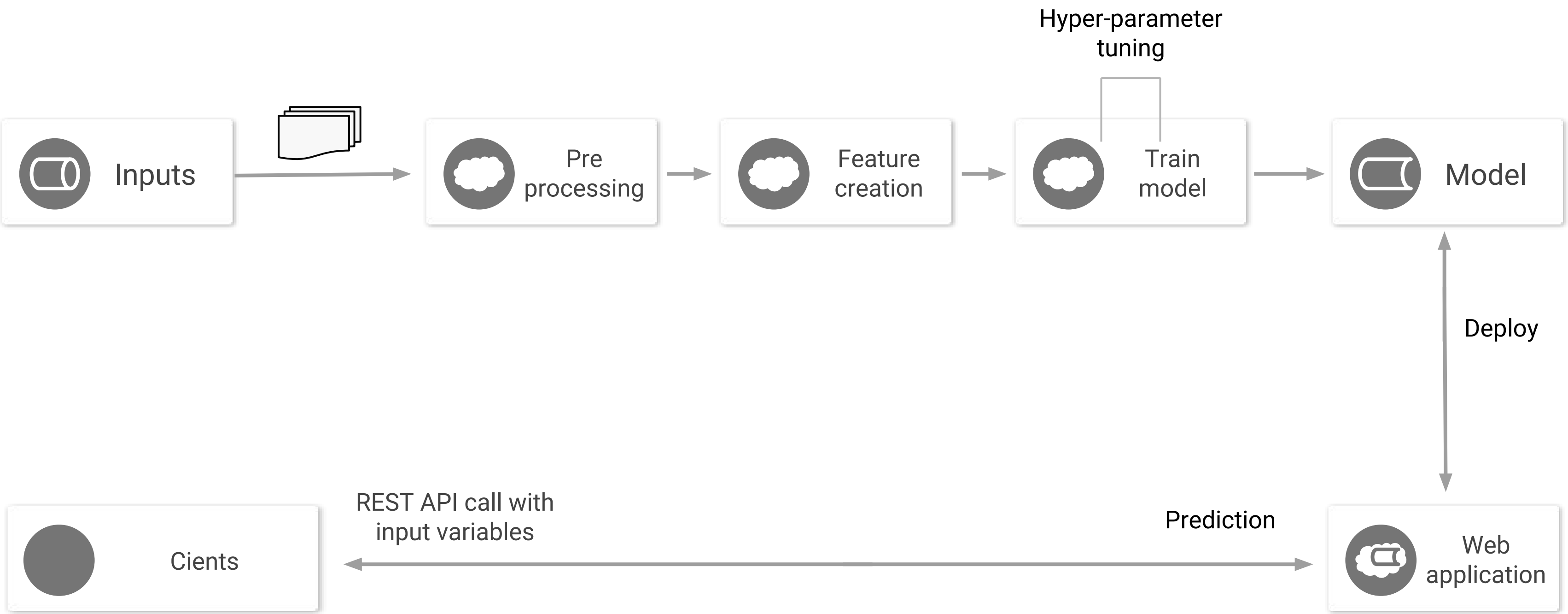
Input necessary transformations



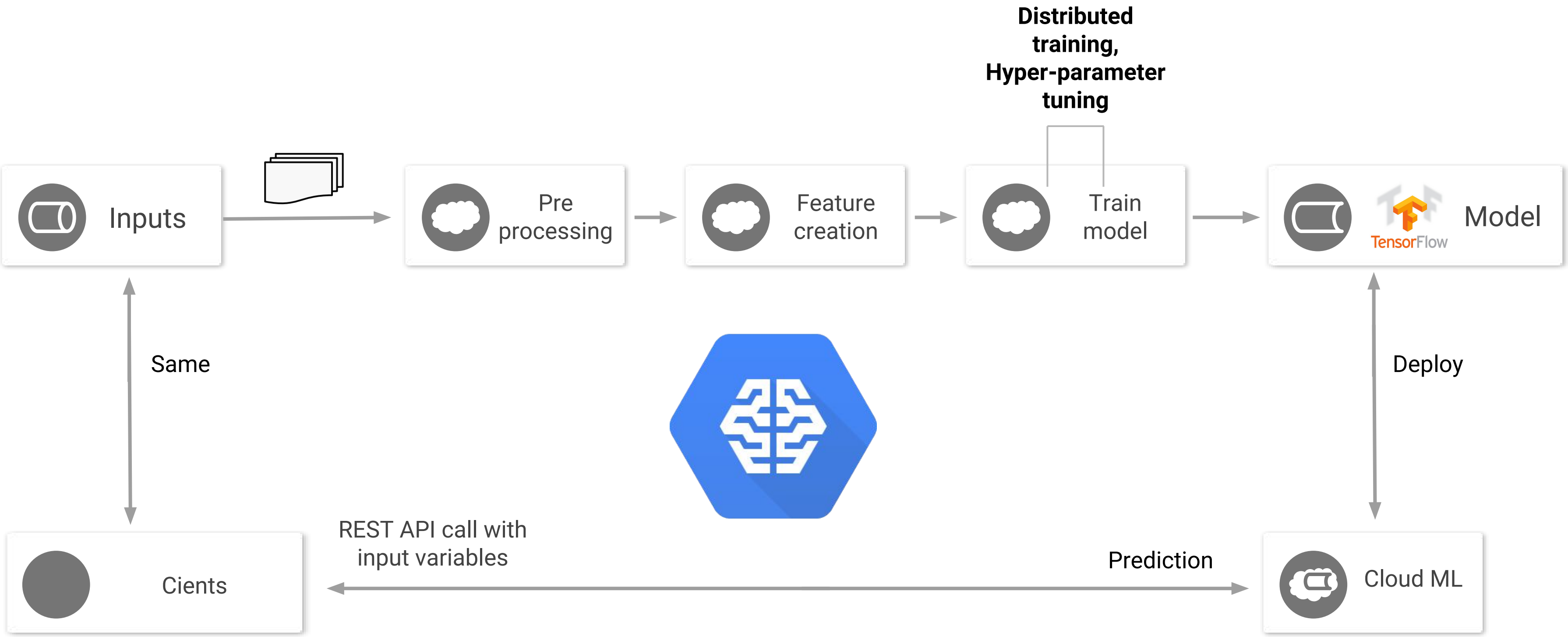
Hyperparameter tuning might be nice



Need to autoscale prediction code



Cloud machine learning—repeatable, scalable, tuned



A first look at some code:
Creating and running a TensorFlow graph
to learn **XOR**

Warmup Lab: Learning to Learn XOR

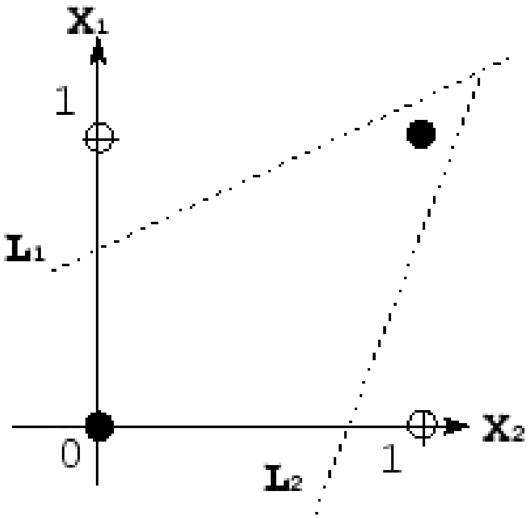
Workshop section: `xor`

XOR: A Minimal Training Example

- Simple
- No need to do data manipulation
- Can’t learn with a single linear regression
- Can use TensorFlow’s standard gradient descent tools to learn it.
- XOR cannot be learned without artificial non-linearity, and at least one hidden layer

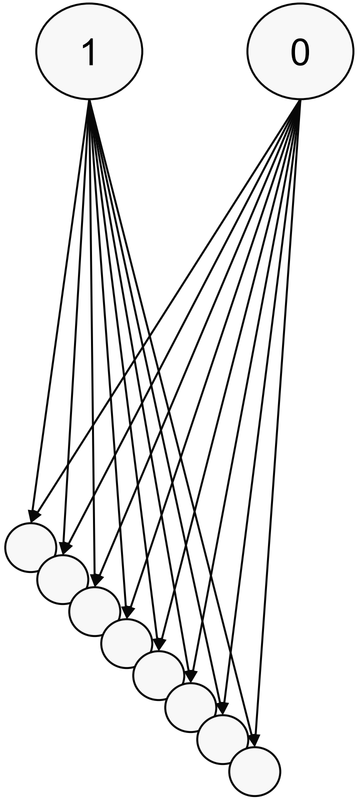
X_1	X_2	Y
0	0	0
0	1	1
1	0	1
1	1	0

$$Y = X_1 \oplus X_2$$



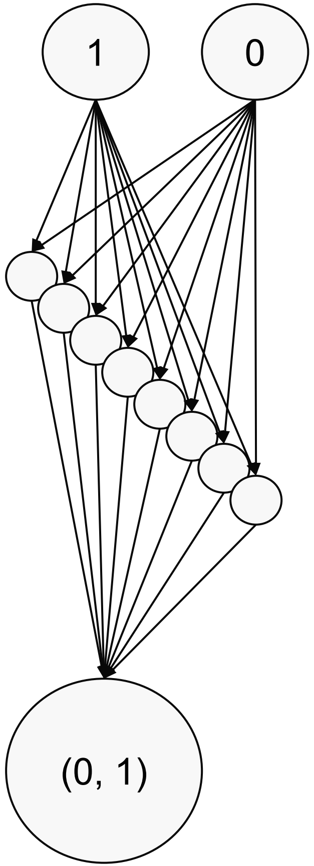
XOR: Building The Graph

```
def make_graph(features,
               labels,
               num_hidden=8):
  hidden_weights = tf.Variable(
       tf.truncated_normal(
           [2, num_hidden],
           stddev=1/math.sqrt(2)))
  # Shape [4, num_hidden]
  hidden_activations = tf.nn.relu(
       tf.matmul(features, hidden_weights))
```



XOR: Building The Graph

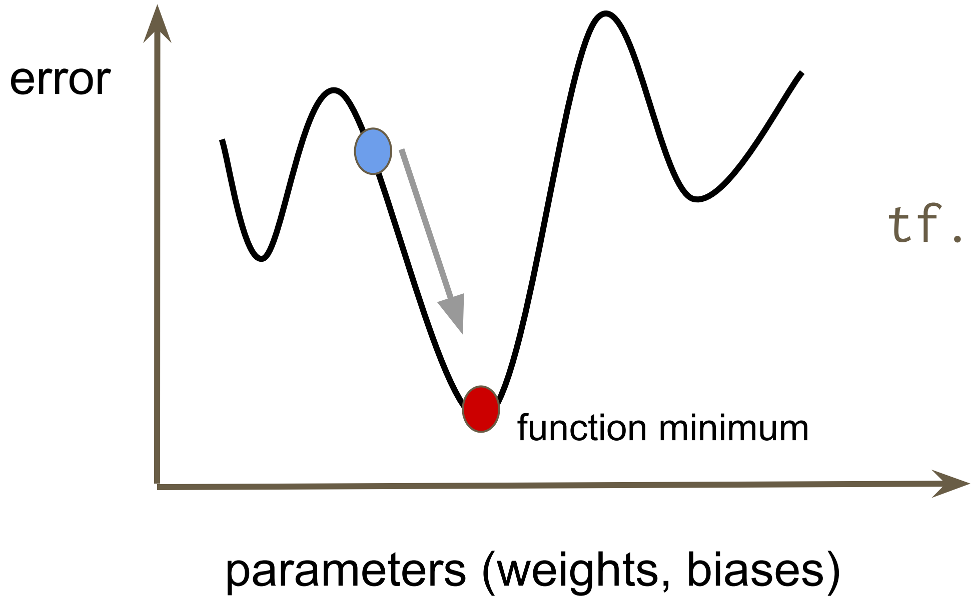
```
output_weights = tf.Variable(tf.truncated_normal(
    [num_hidden, 1],
    stddev=1/math.sqrt(num_hidden)
))

# Shape [4, 1]
logits = tf.matmul(hidden_activations,
output_weights)

# Shape [4]
predictions = tf.sigmoid(tf.squeeze(logits))
```



http://bit.ly/tf-workshop-slides

bit.ly/tensorflow-workshop

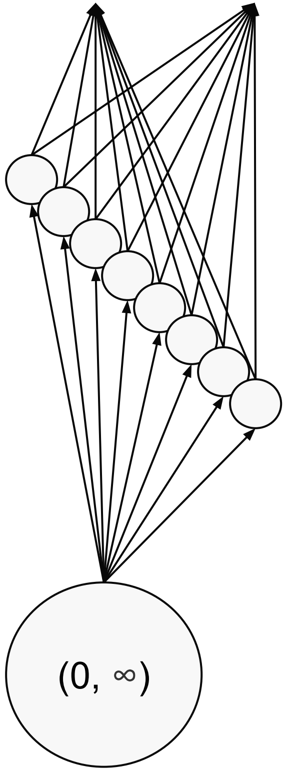
Minimize loss: optimizers



`tf.train.GradientDescentOptimizer`

XOR: Building The Graph

```
loss = tf.reduce_mean(
tf.square(predictions - tf.to_float(labels)))

gs = tf.Variable(0, trainable=False)
train_op = tf.train.GradientDescentOptimizer(
    0.2
).minimize(loss, global_step=gs)
```



Notebook Interlude

XOR: Building The Graph *For Real*

```
xy, y_ = # numpy truth table
graph = tf.Graph()
with graph.as_default():
  features = tf.placeholder(tf.float32, shape=[4, 2])
  labels = tf.placeholder(tf.int32, shape=[4])

  train_op, loss, gs = make_graph(features, labels)
  init = tf.global_variables_initializer()
```

XOR: Running The Graph

```
with tf.Session(graph=graph) as sess:
  while step < num_steps:
    _, step, loss_value = sess.run(
        [train_op, gs, loss],
        feed_dict={features: xy, labels: y_}
    )
```

Common TF NN pattern:

- Create the **model (inference) graph**
- Define the **loss function**
- specify the **optimizer** and learning rate
  → training step op
- In a training loop, call
  `sess.run([train_step,..], feed_dict={....})`
  where `feed_dict` maps inputs to placeholder values

Monitoring With TensorBoard: `xor_summaries`

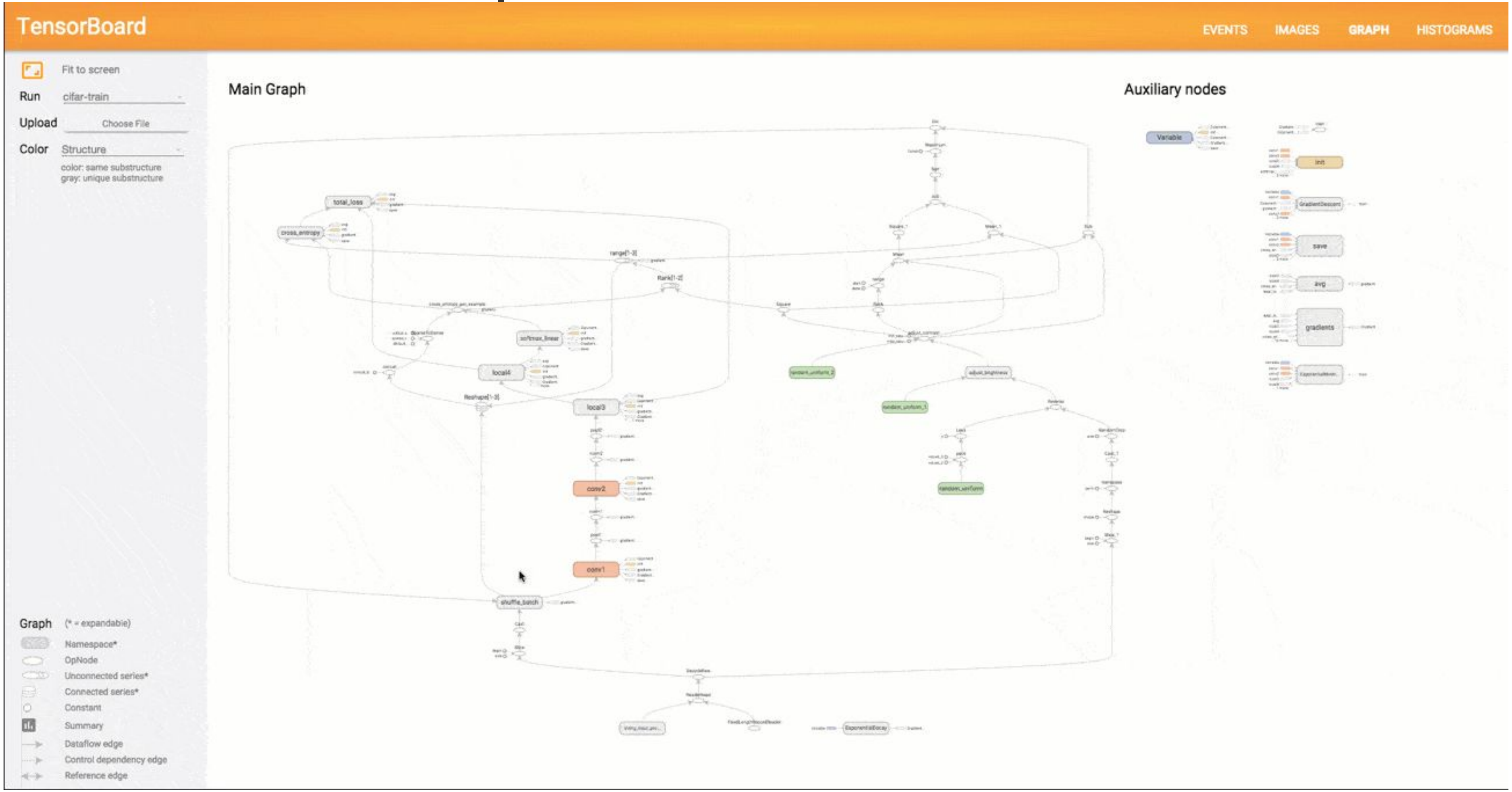
Getting information about your models and training runs:

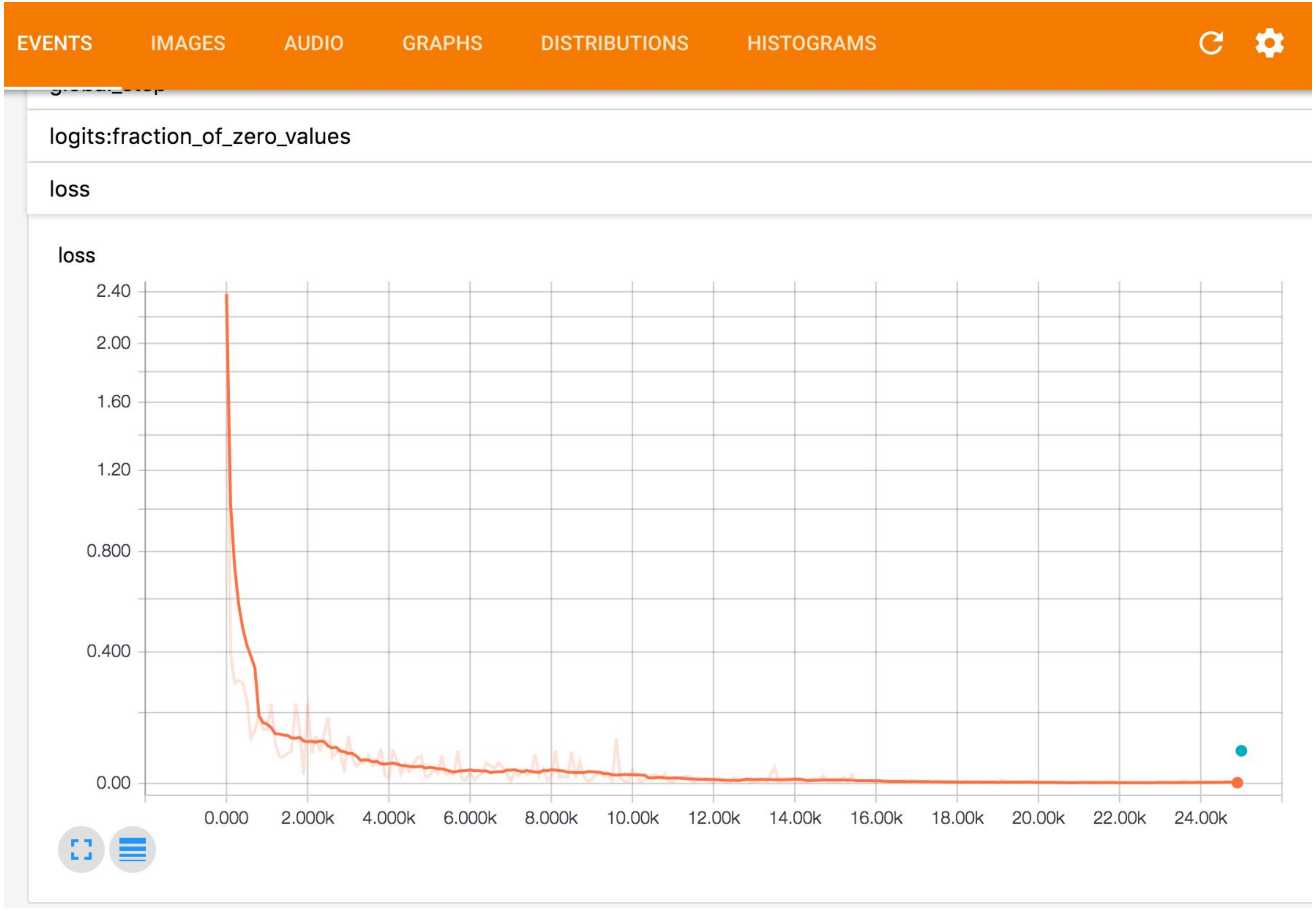
Introducing **TensorBoard**

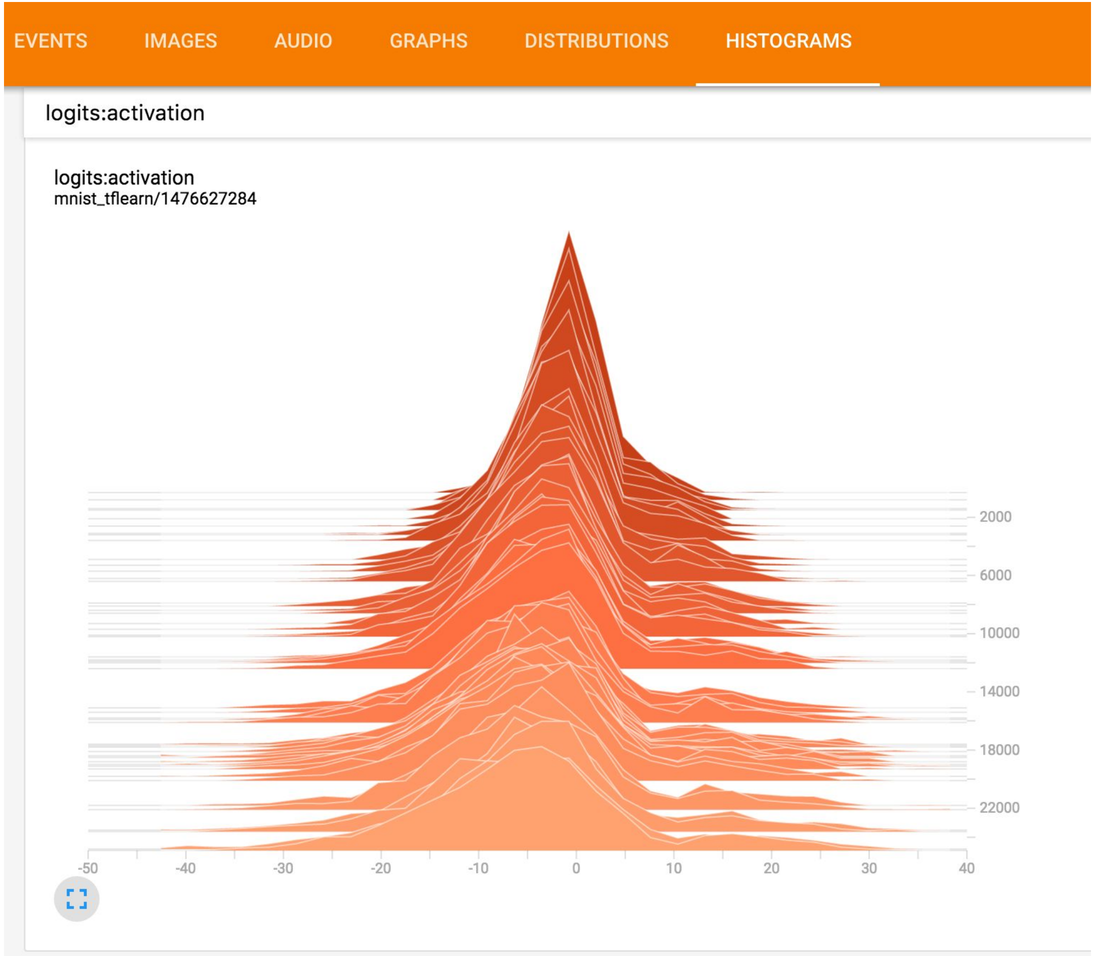
```
tensorboard --logdir=<logdir>
```

(reads subdirs recursively! Great for multiple runs)

Tensorboard: Graph Visualization




EVENTS
IMAGES
AUDIO
GRAPHS
DISTRIBUTIONS
HISTOGRAMS
logits:fraction_of_zero_values
loss
loss
2.40
2.00
1.60
1.20
0.800
0.400
0.00
0.000
2.000k
4.000k
6.000k
8.000k
10.00k
12.00k
14.00k
16.00k
18.00k
20.00k
22.00k
24.00k


EVENTS
IMAGES
AUDIO
GRAPHS
DISTRIBUTIONS
HISTOGRAMS
logits:activation
logits:activation
mnist_tflearn/1476627284
2000
6000
10000
14000
18000
22000
-50
-40
-30
-20
-10
0
10
20
30
40

Related concepts / resources

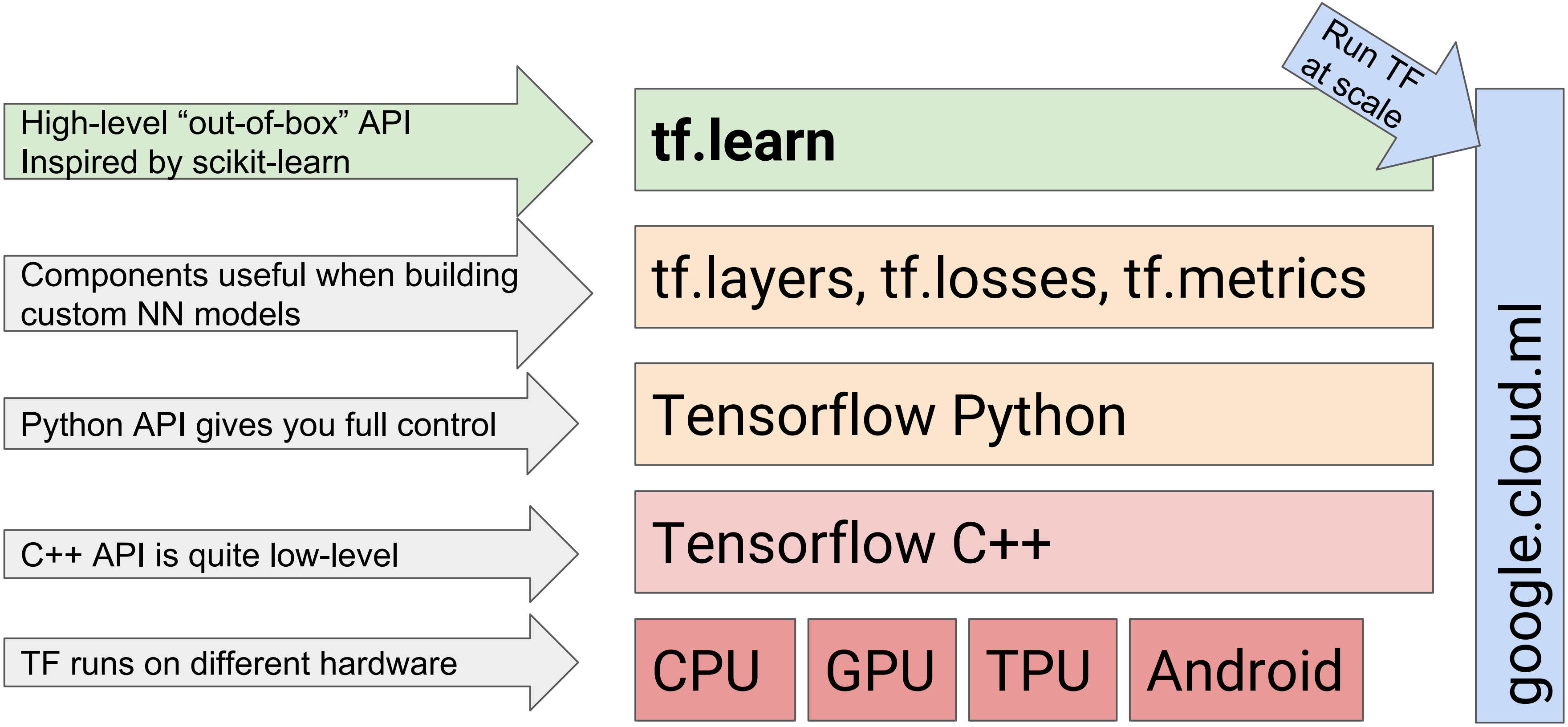
- Softmax Function: http://bit.ly/softmax
- Loss Function: http://bit.ly/loss-fn
- Gradient Descent Overview: http://bit.ly/gradient-descent
- Training, Testing, & Cross Validation: http://bit.ly/ml-eval

Break (10 min)

Up next: Using the TensorFlow High-level APIs

“Wide and Deep”: Using the TensorFlow High-level APIs

TensorFlow toolkit hierarchy



http://scikit-learn.org/

Many levels of abstraction

Choose the right one for you:

- Layers, losses, metrics
- Training/Eval loop functions
- Estimator (BaseEstimator)
  - Any model you want, but must separate input from the rest of the model
- Predefined estimators
  - `LinearClassifier`, `DNNClassifier`, `DNNRegressor`, ... **`DNNLinearCombinedClassifier`**
  - Limited configuration options: feature columns, metrics.

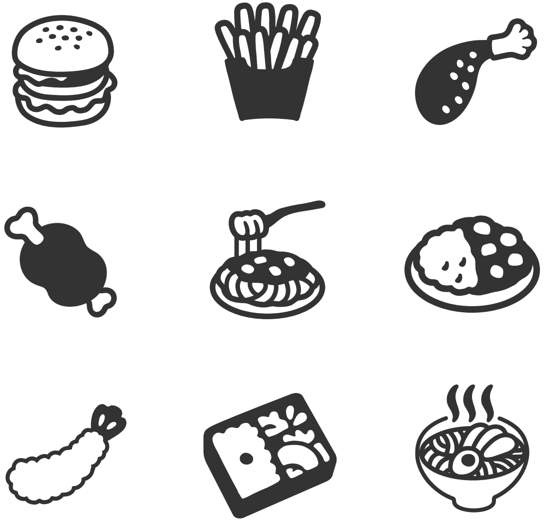
Typical structure

- Load data
- Set up feature columns
- Create your model
- Run the training loop (fit the model)
- Evaluate your model's accuracy (and other metrics)
- (optional) Predict new examples

tf.learn high-level structure

```
# data already loaded into 'data_sets'
feature_columns = tf.contrib.learn.infer_real_valued_columns_from_input(
    data_sets.train.images)
model = tf.contrib.learn.DNNClassifier(
        [layer2_hidden_units, layer1_hidden_units],
        feature_columns=feature_columns,
        n_classes=NUM_CLASSES
)
model.fit(x=data_sets.train.images, y=data_sets.train.labels)
model.evaluate(x=data_sets.eval.images, y=data_sets.eval.labels)
model.predict(x=some_new_images)
```

Motivation - a "magical" food app

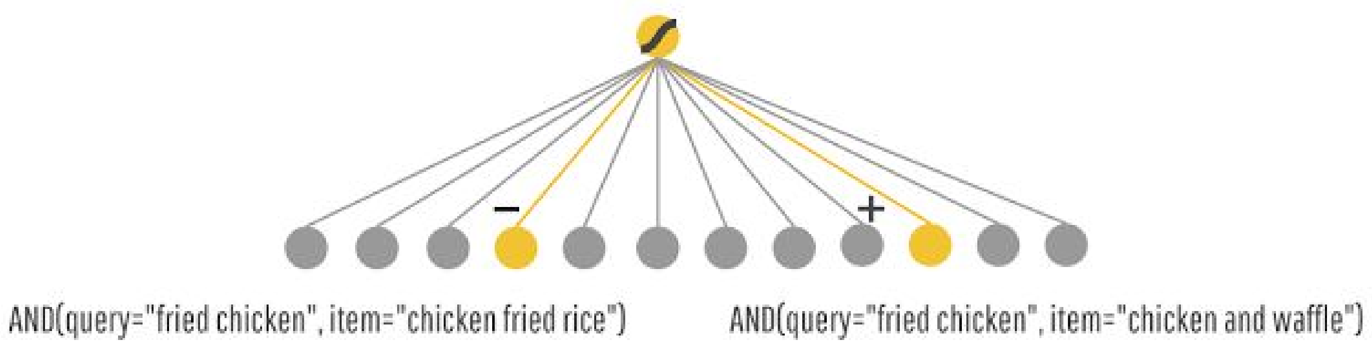


Launch and Iterate!

- Naive character matching
- Say "Fried chicken"
- Get "Chicken Fried Rice"
- Oops. Now what?
- Machine learning to the rescue!

v2.0: memorize all the things

- Train a linear TF model

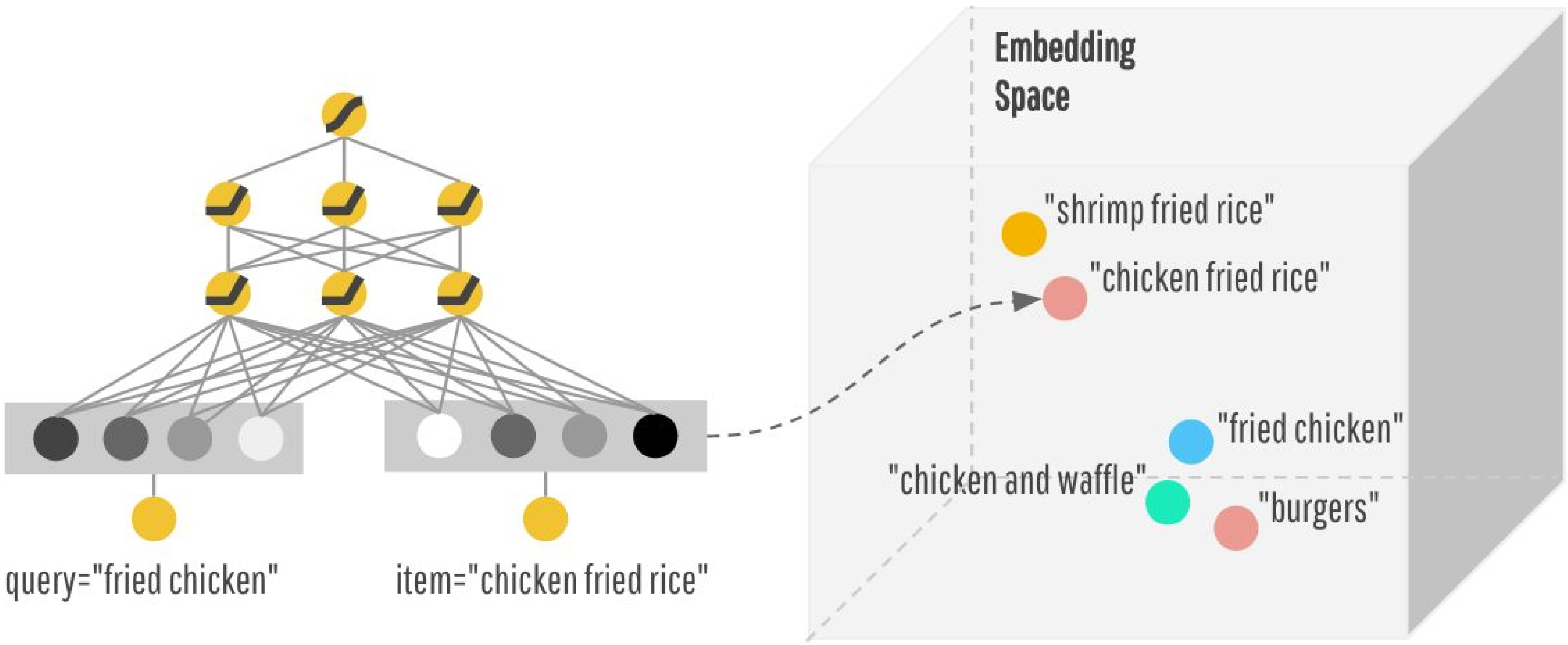


- Your app is gaining traction!

Problem: Your users are bored!

- Too many 🍗 & waffles
- Show me similar, but different food
- Your users are picky 🙊

v3.0: More generalized recommendations for all



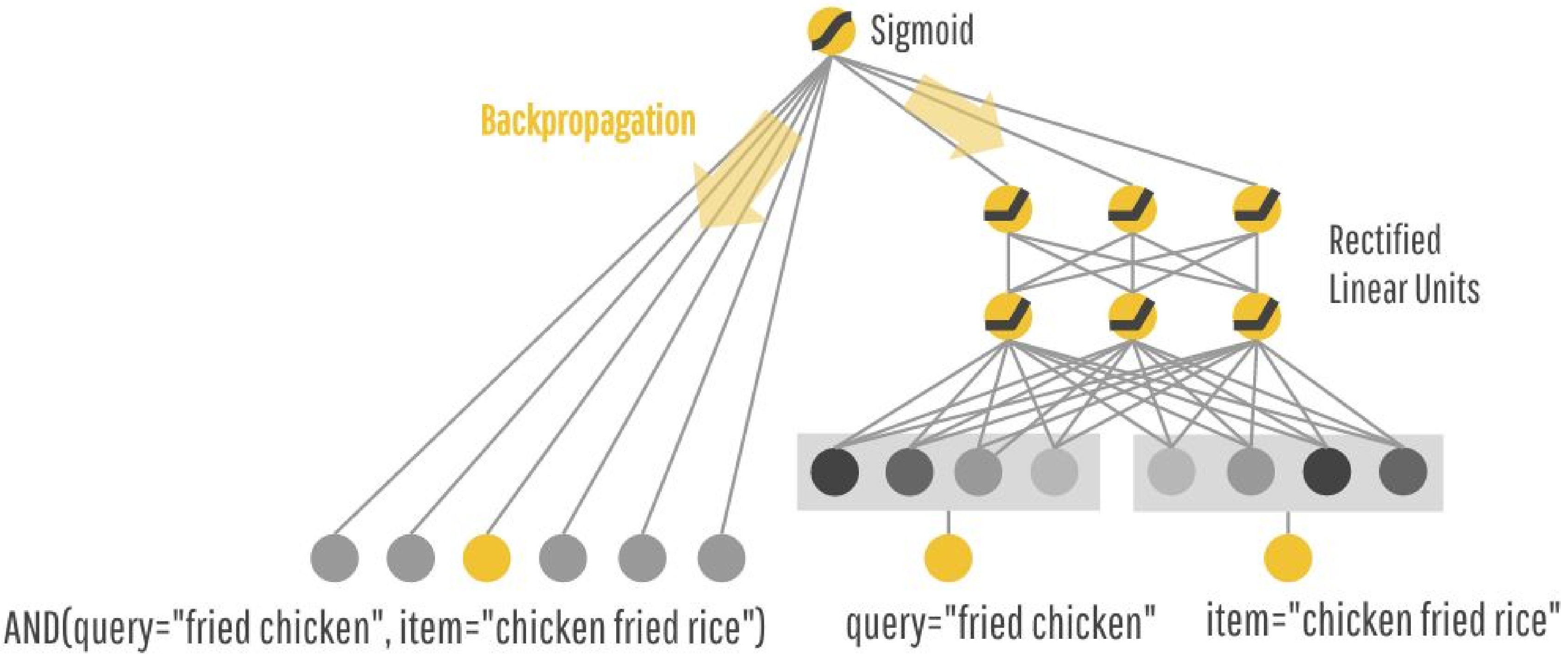
No good deed goes unpunished

- Some recommendations are "too general"
    - Irrelevant dishes are being sent
- Your users are still picky 😞

No good deed goes unpunished

- 2 types of requests: specific and general
- "iced decaf latte with nonfat milk" != "hot latte with whole milk"
- “seafood” or “italian food” or “fast food”
- How to balance this?

v4.0: Why not both?



Lab: Wide and Deep: Using TensorFlow's high-level APIs

Workshop section:

`wide_n_deep`

Meet our dataset: US Census Data

- Just as exciting as chicken and waffles
- **Task**: predict the probability that the individual has an annual income of over 50,000 dollars
- Over 32k training examples
- Was extracted from the 1994 US Census by Barry Becker.

Meet our dataset: US Census Data

Column Name	Type	Description
age	Continuous	The age of the individual
workclass	Categorical	The type of employer the individual has (government, military, private, etc.).
fnlwgt	Continuous	The number of people the census takers believe that observation represents (sample weight). This variable will not be used.
education	Categorical	The highest level of education achieved for that individual.
education_num	Continuous	The highest level of education in numerical form.
marital_status	Categorical	Marital status of the individual.

Meet our dataset: US Census Data

Column Name	Type	Description
occupation	Categorical	The occupation of the individual.
relationship	Categorical	Wife, Own-child, Husband, Not-in-family, Other-relative, Unmarried.
race	Categorical	White, Asian-Pac-Islander, Amer-Indian-Eskimo, Other, Black.
gender	Categorical	Female, Male.
capital_gain	Continuous	Capital gains recorded.
capital_loss	Continuous	Capital Losses recorded.

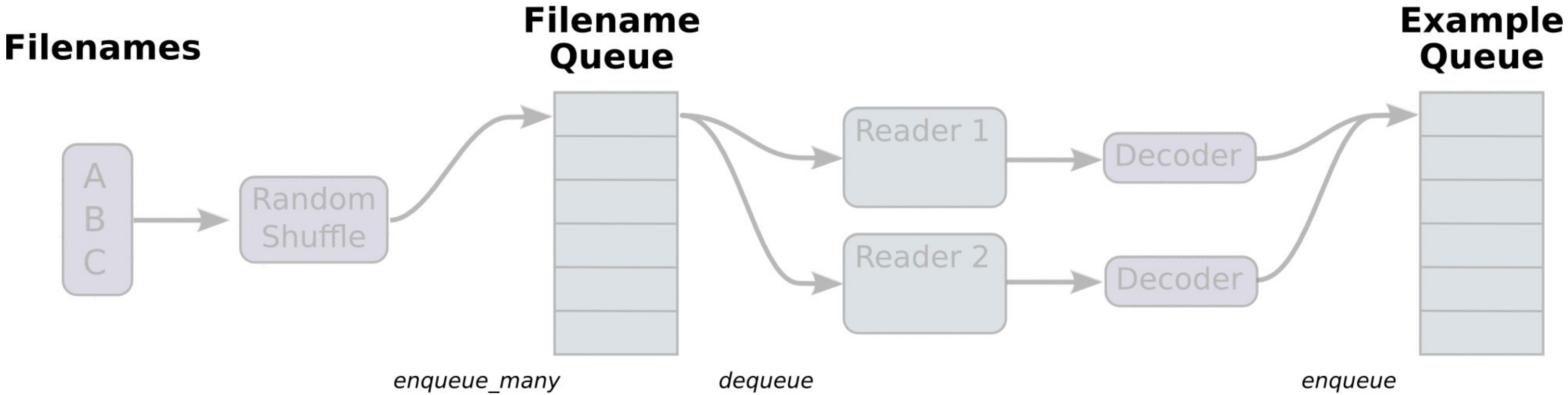
Meet our dataset: US Census Data

Column Name	Type	Description
hours_per_week	Continuous	Hours worked per week.
native_country	Categorical	Country of origin of the individual.
income_bracket	**Categorical**	**">50K" or "<=50K", meaning whether the person makes more than $50,000 annually.**

Typical structure

- **Load data**
- Set up feature columns
- Create your model
- Run the training loop (fit the model)
- Evaluate your model's accuracy (and other metrics)
- (optional) Predict new examples

(File) Queues In TensorFlow



File Queues

```
filename_queue = tf.train.string_input_producer([filename])
reader = tf.TextLineReader()
key, value = reader.read_up_to(filename_queue,
  num_records=BATCH_SIZE)
record_defaults = [[0], [" "], [0], [" "], [0],
                   [" "], [" "], [" "], [" "], [" "],
                   [0], [0], [0], [" "], [" "]]
columns = tf.decode_csv(value, record_defaults=record_defaults)
```

Input data format

```
features = {
  'hours_per_week': array([16, 45, 50], dtype=int32),

  'relationship': SparseTensorValue(indices=array([[0, 0],[1, 0],[2, 0]]), values=array(['
Not-in-family', ' Husband', ' Not-in-family'], dtype=object), shape=array([3, 1])),

  'gender': SparseTensorValue(indices=array([[0, 0],[1, 0],[2, 0]]), values=array([' Female', '
Male', ' Female'], dtype=object), shape=array([3, 1])),

  'age': array([49, 52, 31], dtype=int32)
...
}
labels = [0 1 1]
```

Input data format

```
features, income_bracket = dict(zip(COLUMNS, columns[:-1])), columns[-1]
# for sparse tensors
for feature_name in CATEGORICAL_COLUMNS:
  features[feature_name] = tf.expand_dims(features[feature_name], -1)
# convert ">50K" => 1 and "<=50K" => 0
income_int = tf.to_int32(tf.equal(income_bracket, " >50K"))
return features, income_int
```

Typical structure

- Load data
- **Set up feature columns**
- Create your model
- Run the training loop (fit the model)
- Evaluate your model's accuracy (and other metrics)
- (optional) Predict new examples

Feature columns

```
# Sparse base columns.
gender = tf.contrib.layers.sparse_column_with_keys(
    column_name="gender", keys=["female", "male"])
education = tf.contrib.layers.sparse_column_with_hash_bucket(
    "education", hash_bucket_size=1000)
...
# Continuous base columns.
age = tf.contrib.layers.real_valued_column("age")
...
```

Feature columns continued

```
# Transformations.
age_buckets = tf.contrib.layers.bucketized_column(
    age, boundaries=[18, 25, 30, 35, 40, 45, 50, 55, 60, 65])
education_occupation = tf.contrib.layers.crossed_column(
    [education, occupation], hash_bucket_size=int(1e4))
...
# embeddings for deep learning
tf.contrib.layers.embedding_column(workclass, dimension=8)
```

Typical structure

- Load data
- Set up feature columns
- **Create your model**
- Run the training loop (fit the model)
- Evaluate your model's accuracy (and other metrics)
- (optional) Predict new examples

Make the model (Estimator)

```
m = tf.contrib.learn.DNNLinearCombinedClassifier(
     model_dir=model_dir,
     linear_feature_columns=wide_columns,
     dnn_feature_columns=deep_columns,
     dnn_hidden_units=[100, 70, 50, 25])
```

Typical structure

- Load data
- Set up feature columns
- Create your model
- **Run the training loop (fit the model)**
- **Evaluate your model's accuracy (and other metrics)**
- (optional) Predict new examples

Fit and Evaluate

```
m.fit(input_fn=generate_input_fn(train_file),
      steps=1000)
results = m.evaluate(
     input_fn=generate_input_fn(test_file),
     steps=1)
print('Accuracy: %s' % results['accuracy'])
```

Checkpointing and reloading a trained model

- Everything is stored in the `model_dir` folder

```
m = tf.contrib.learn.DNNLinearCombinedClassifier(
      model_dir=model_dir,
      linear_feature_columns=wide_columns,
      dnn_feature_columns=deep_columns,
      dnn_hidden_units=[100, 70, 50, 25])
```

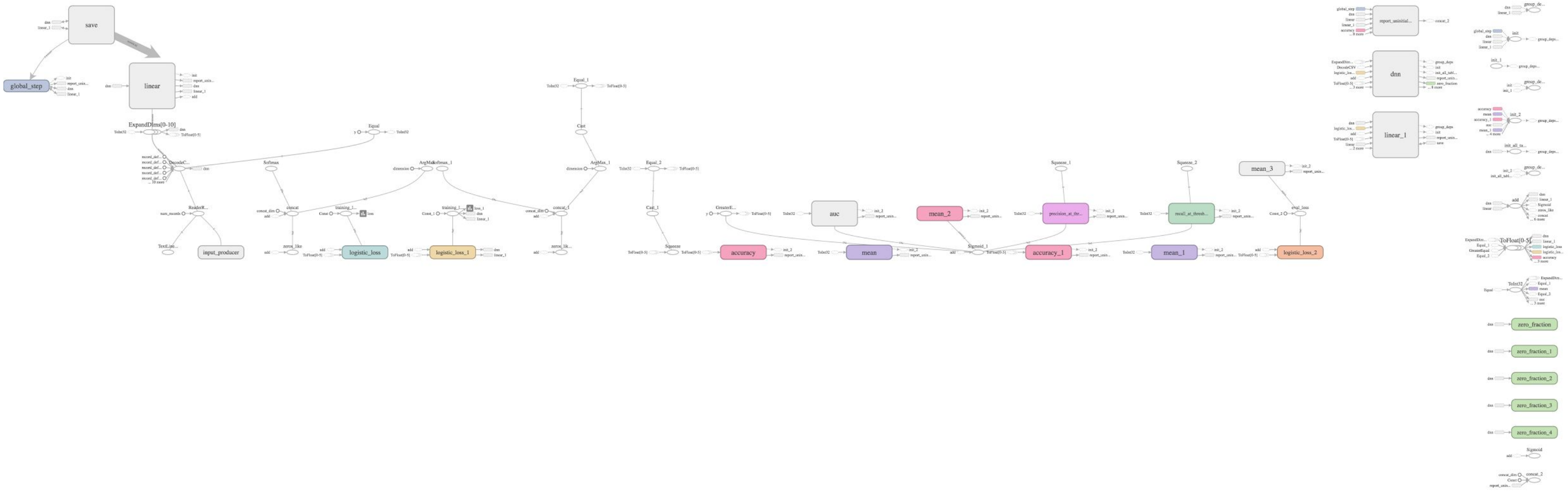
- If you run multiple `fit` operations on the same `Estimator` and supply the same directory, training will resume where it left off.

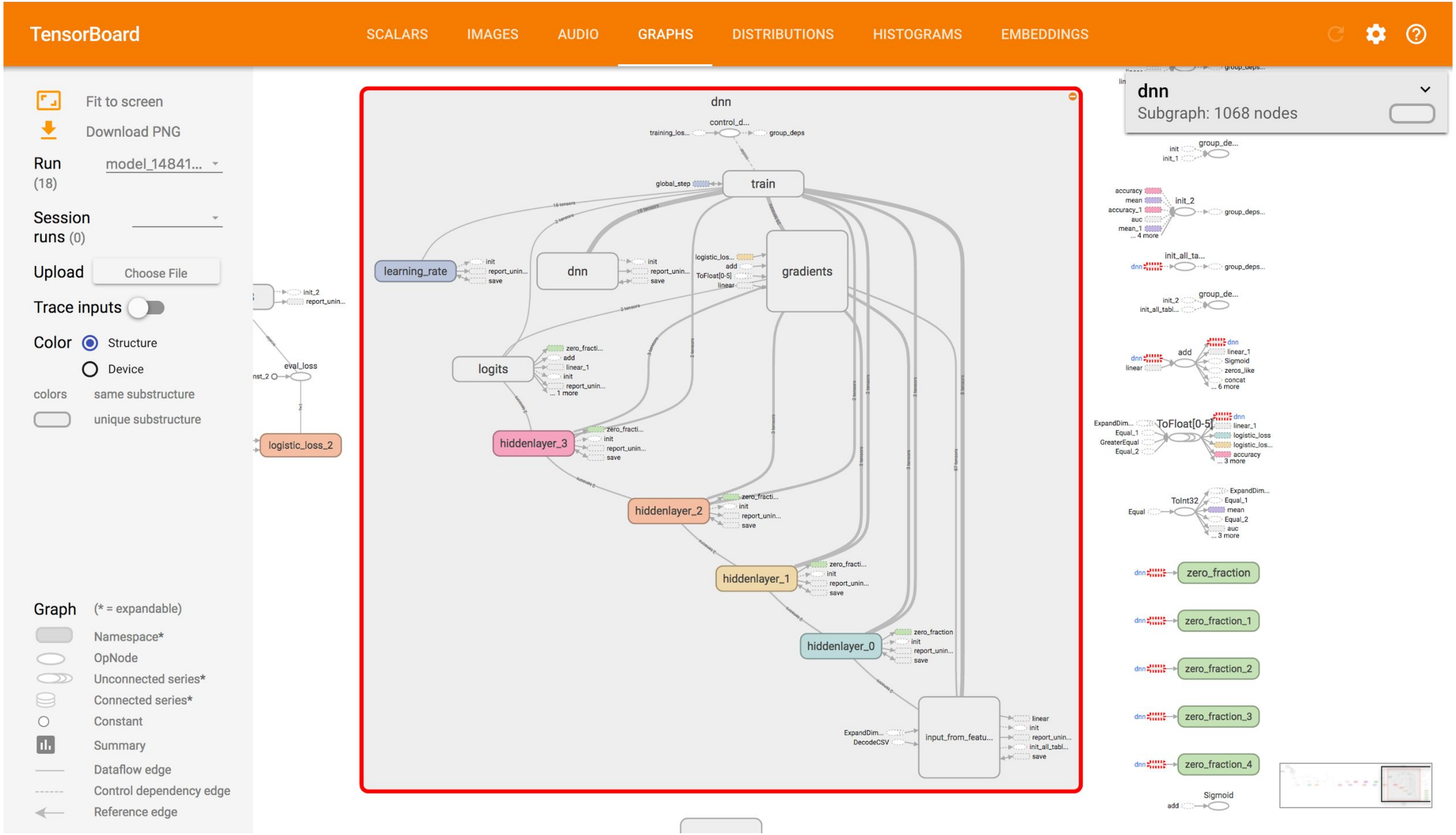
http://bit.ly/tf-workshop-slides

bit.ly/tensorflow-workshop

To the code!

Time for a vision test





TensorBoard
SCALARS
IMAGES
AUDIO
GRAPHS
DISTRIBUTIONS
HISTOGRAMS
EMBEDDINGS
Fit to screen
Download PNG
Run (18)
model_14841...
Session runs (0)
Upload
Choose File
Trace inputs
Color
Structure
Device
colors
same substructure
unique substructure
Graph
(* = expandable)
Namespace*
OpNode
Unconnected series*
Connected series*
Constant
Summary
Dataflow edge
Control dependency edge
Reference edge
dnn
train
learning_rate
gradients
logits
hiddenlayer_3
hiddenlayer_2
hiddenlayer_1
hiddenlayer_0
input_from_featu...
eval_loss
logistic_loss_2
dnn
Subgraph: 1068 nodes
zero_fraction
zero_fraction_1
zero_fraction_2
zero_fraction_3
zero_fraction_4

RECAP: what we’ve done so far

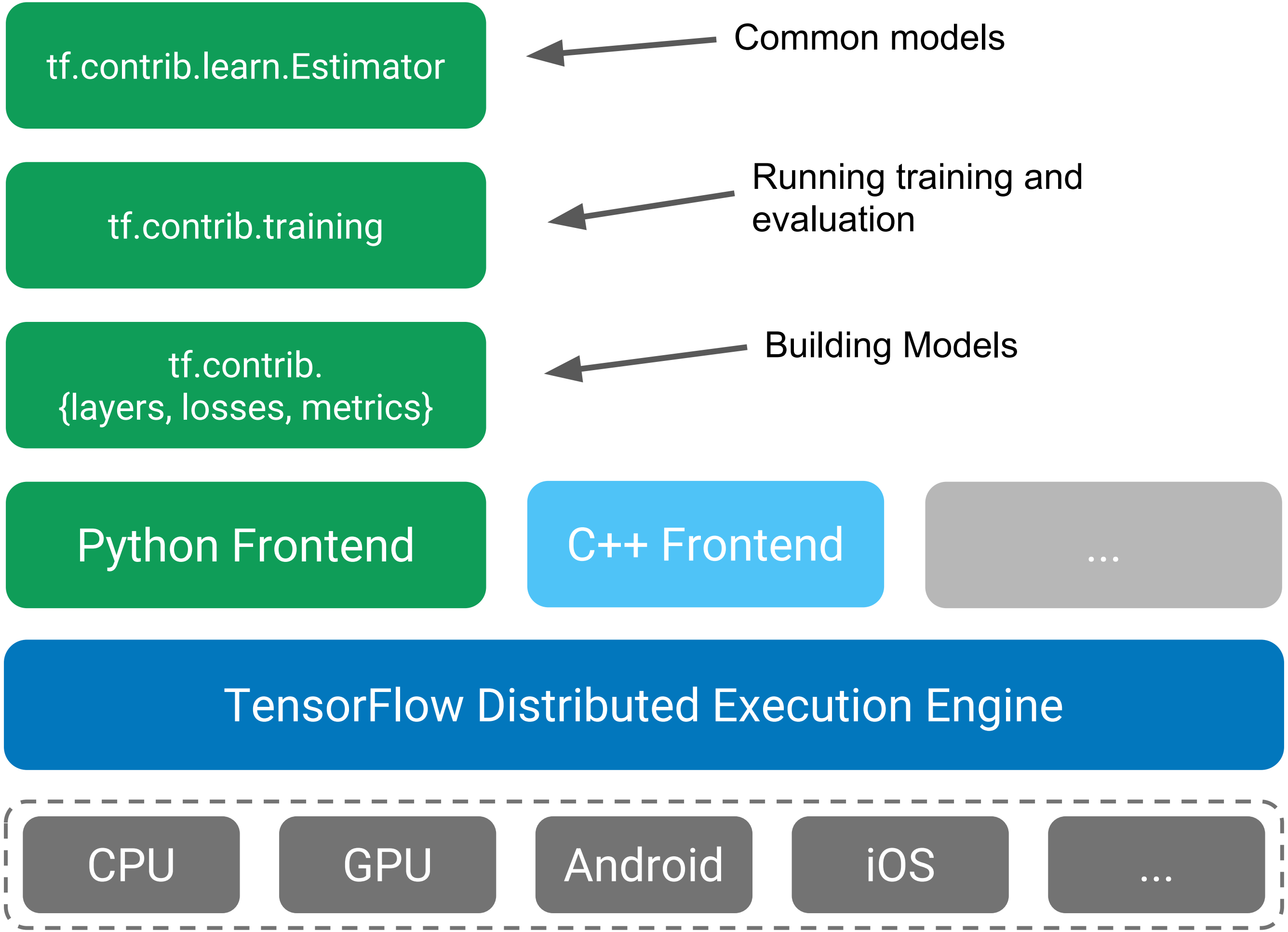
So far we've...

- Learned and used common **patterns for defining and training TensorFlow model graphs**
- Used **TensorFlow's high-level APIs**
- Discussed the merits of **wide linear models** and **deep neural networks**
- Introduced **queues**
- Created summary info and introduced **TensorBoard**

Break (10 min)

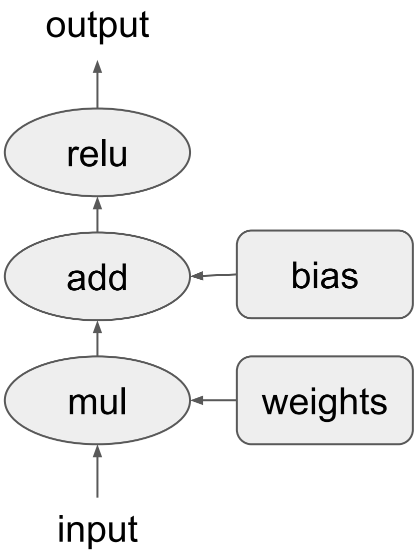
Up next:
Building Word embeddings and Custom Estimators

TensorFlow: Flexibility vs. Usability



INNOVATION + ASSISTANCE

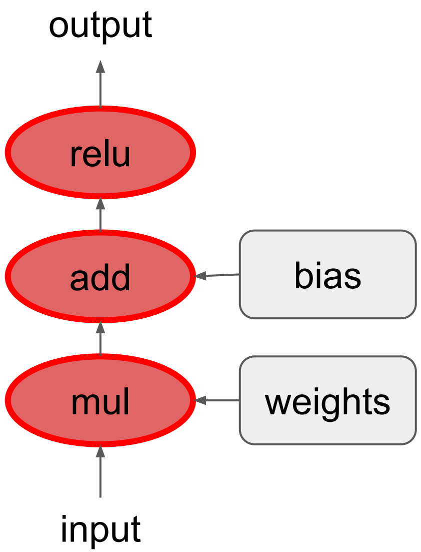
TensorFlow: Flexibility vs. Usability



INNOVATION + ASSISTANCE

TensorFlow: Flexibility vs. Usability

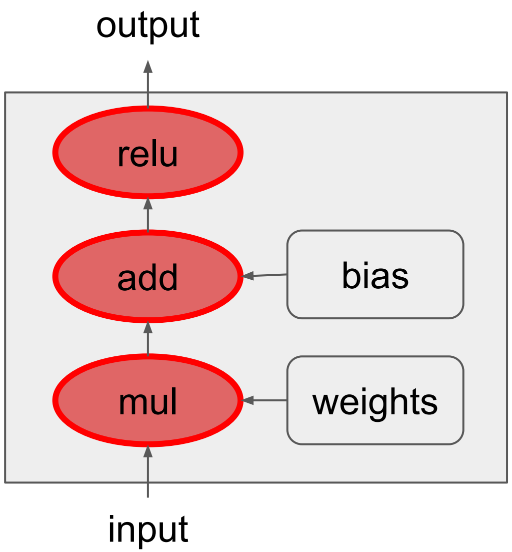
Ops are ***small*** computations

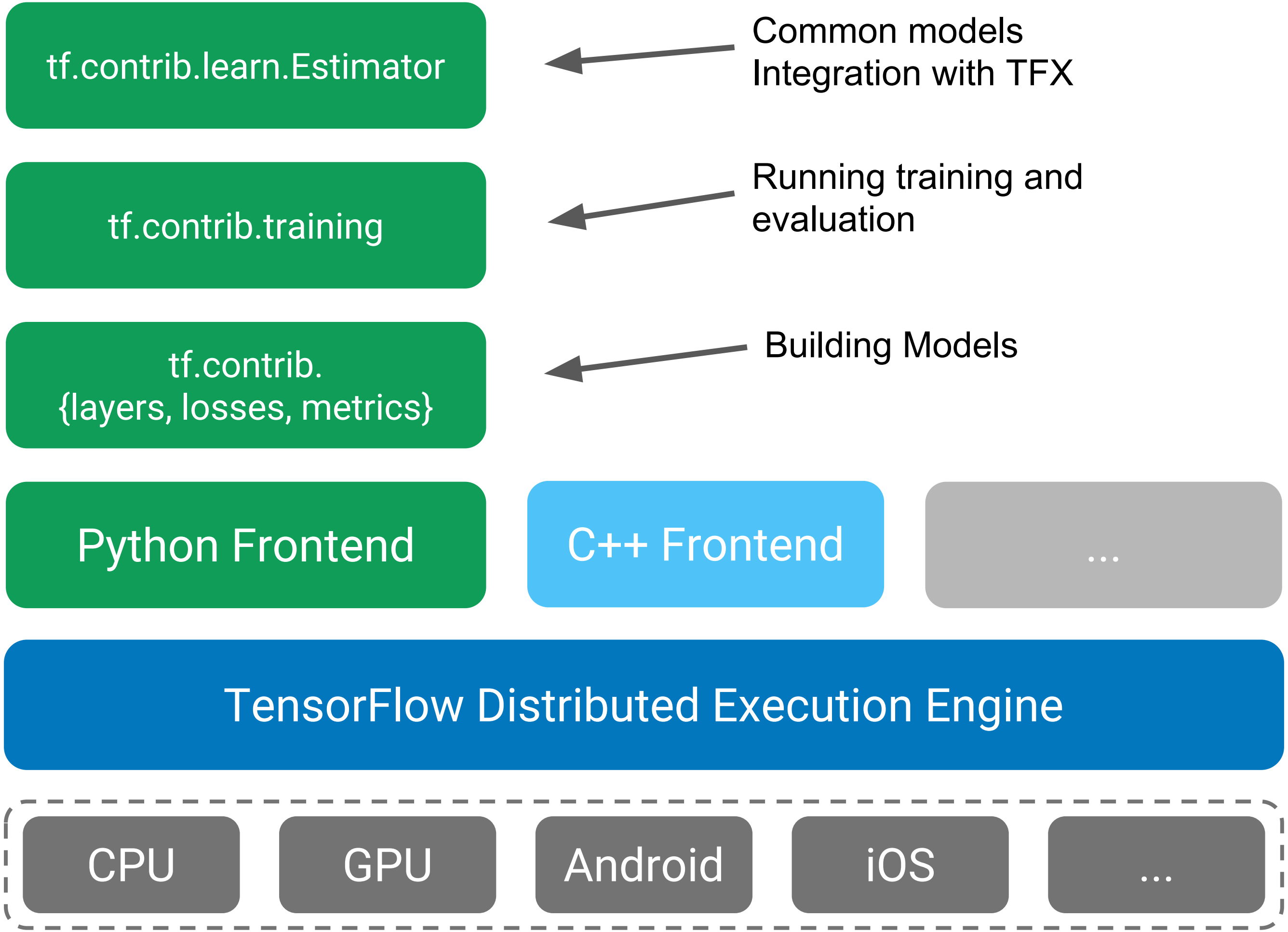


INNOVATION + ASSISTANCE

TensorFlow: Flexibility vs. Usability

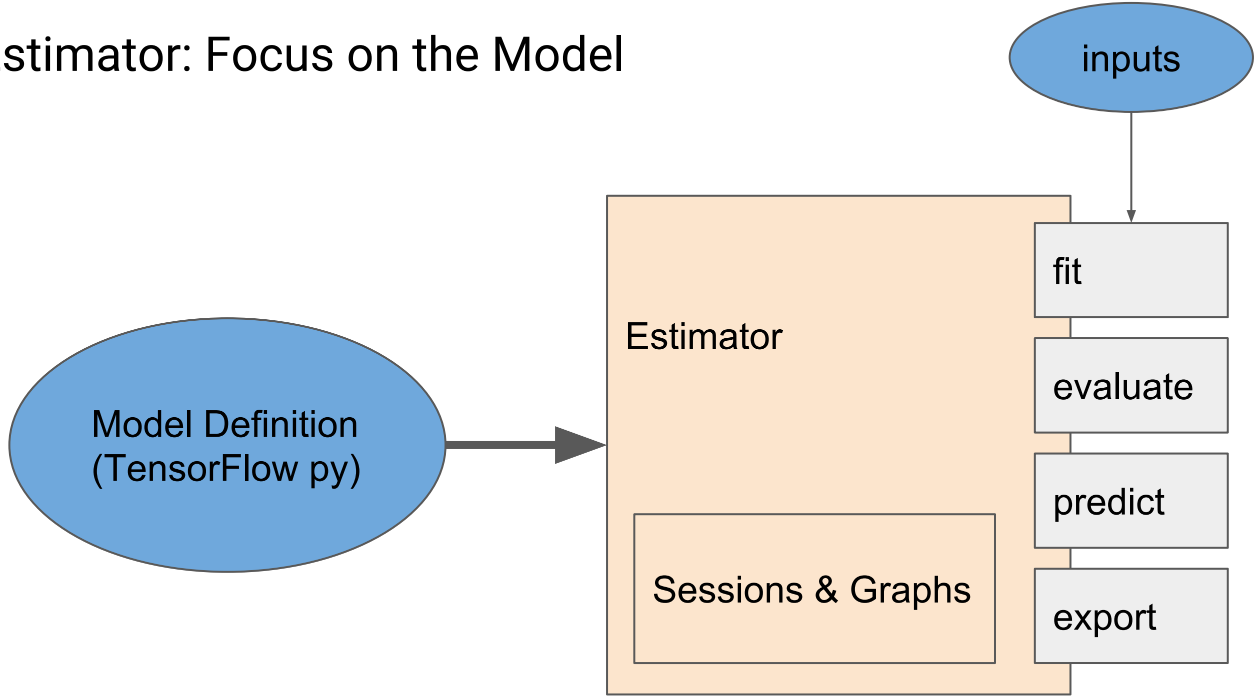
Make ***bigger*** Ops





INNOVATION + ASSISTANCE

Estimator: Focus on the Model



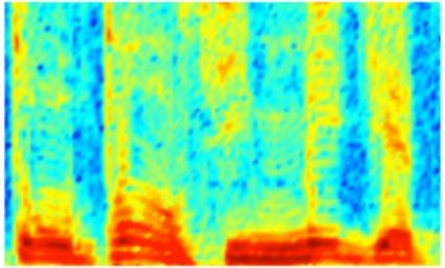
INNOVATION + ASSISTANCE

But, what if there is not a pre-baked Estimator for your model?

Word2vec: Learning and using word embeddings, Building Custom Estimators

What is an embedding?

AUDIO



Audio Spectrogram

DENSE

IMAGES

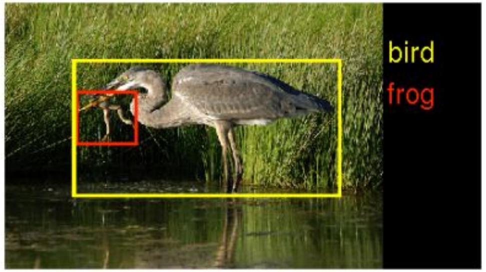


Image pixels

DENSE

TEXT

0	0	0	0.2	0	0.7	0	0	0	...	...

Word, context, or document vectors

SPARSE

Word embeddings

- Word data (and categorical data in general) can't be modeled as dense data

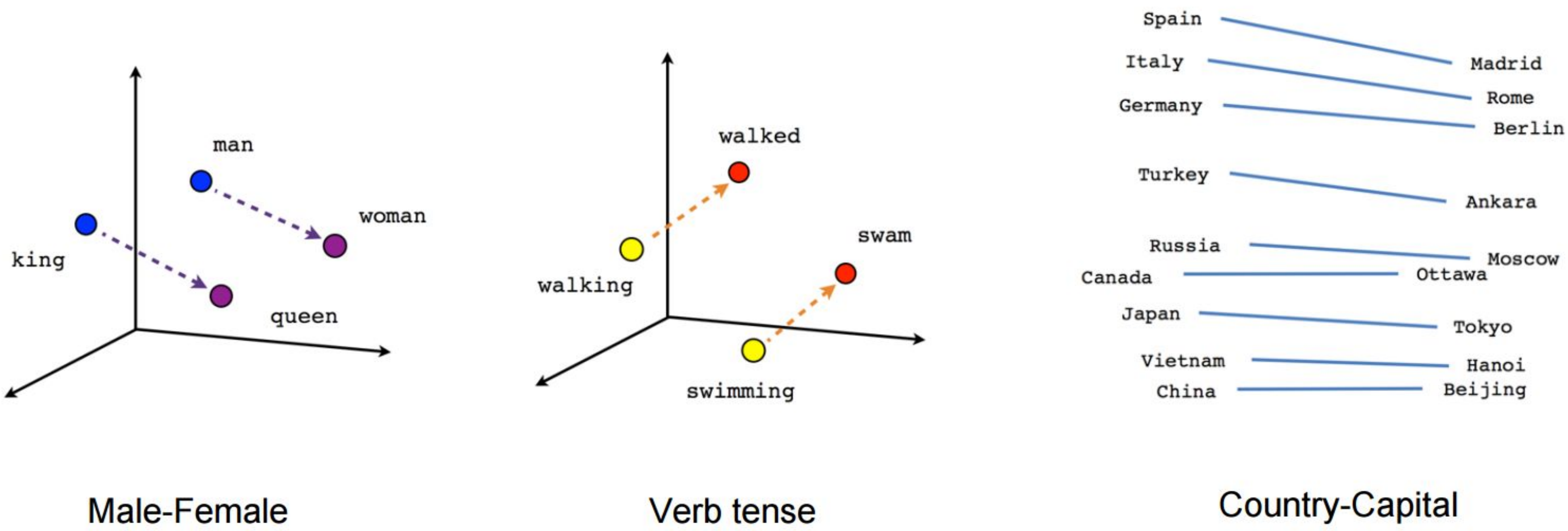
- The **word2vec** model attempts to "compress" sparse "word-indices" into dense "word-vectors."

- These word vectors tend to have neat properties!

NIPS paper: Mikolov et al.: http://bit.ly/word2vec-paper

```
model.nearby([b'cat'])

b'cat'              1.0000
b'cats'             0.6077
b'dog'              0.6030
b'pet'              0.5704
b'dogs'             0.5548
b'kitten'           0.5310
b'toxoplasma'       0.5234
b'kitty'            0.4753
b'avner'            0.4741
b'rat'              0.4641
b'pets'             0.4574
b'rabbit'           0.4501
b'animal'           0.4472
b'puppy'            0.4469
b'veterinarian'     0.4435
b'raccoon'          0.4330
b'squirrel'         0.4310
```

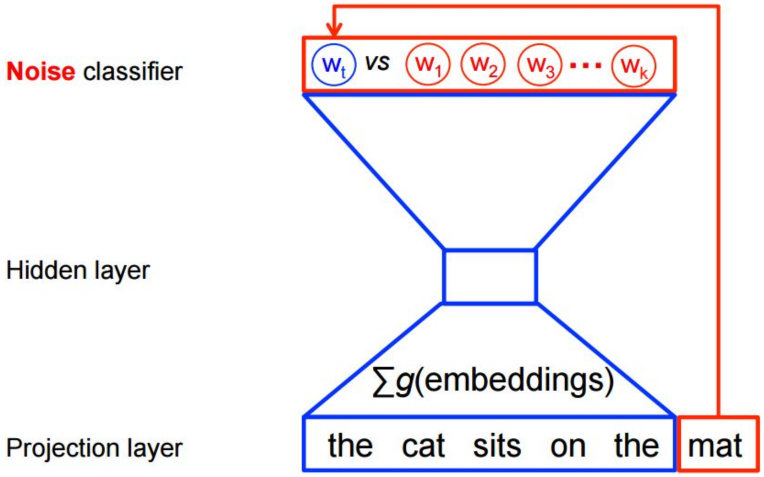
```
model.analogy(b'cat',
b'kitten', b'dog')
Out[1]: b'puppy'
```



https://www.tensorflow.org/versions/r0.8/images/linear-relationships.png

Making word2vec scalable

- Instead of a full probabilistic model… Use logistic regression to discriminate target words from imaginary (noise) words.
- [Noise-contrastive estimation (NCE) loss]
    - tf.nn.**nce_loss**()
    - Scales with number of noise words



https://www.tensorflow.org/versions/r0.8/images/nce-nplm.png

Skip-Gram model
(predict source context-words from target words)

Context/target pairs, window-size of 1 in both directions:

`the quick brown fox jumped over the lazy dog ...` →

`([the, brown], quick), ([quick, fox], brown), ([brown, jumped], fox),` ...

Skip-gram model
(predict source context-words from target words)

Context/target pairs, window-size of 1 in both directions:

`the quick brown fox jumped over the lazy dog ...` →

`([the, brown], quick), ([quick, fox], brown), ([brown, jumped], fox),` ...

Input/output pairs:

`(quick, the), (quick, brown), (brown, quick), (brown, fox),` ...

Typically optimize with stochastic gradient descent (SGD) using minibatches

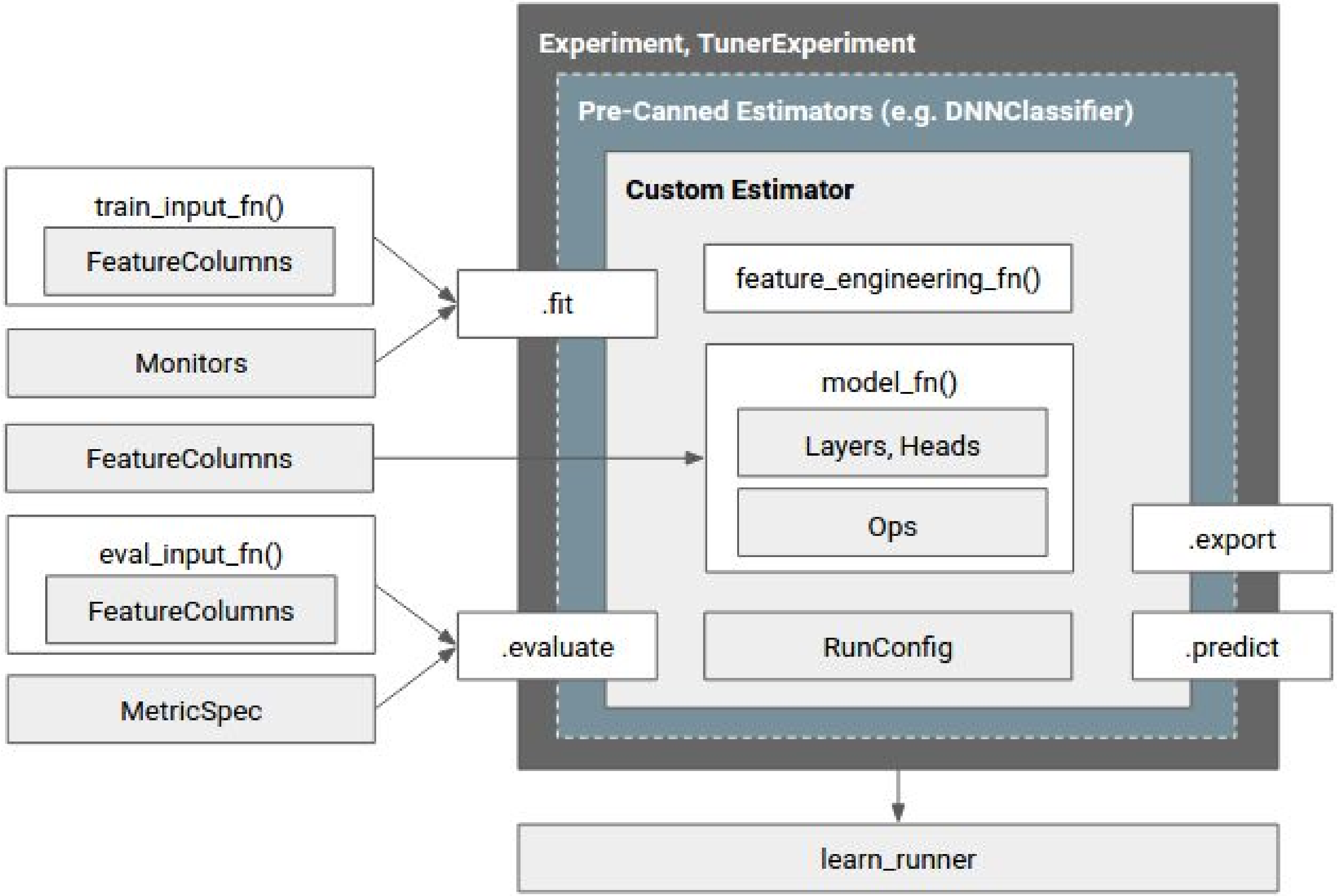
Custom Estimator:

- `model_fn`: Constructs the graph given features, labels, and a “mode”
- `input_fn`: Constructs a graph to read input tensors from files using a `QueueRunner`

Why Is Word2Vec a good example problem?

- Complex data input (generate skipgrams).
- Different data input for training and inference (indexed words vs. text words).
- Different computation graph for training and inference.
- Benefits from sophisticated distribution primitives.
- Can use cool new TensorBoard tools! (Embedding Visualizer)

`Experiment` and `learn_runner`


Experiment, TunerExperiment
Pre-Canned Estimators (e.g. DNNClassifier)
Custom Estimator
train_input_fn()
FeatureColumns
Monitors
FeatureColumns
eval_input_fn()
FeatureColumns
MetricSpec
.fit
.evaluate
feature_engineering_fn()
model_fn()
Layers, Heads
Ops
RunConfig
.export
.predict
learn_runner

Experiment

Encapsulates all of the information to run an estimator:

- Training data locations
- Evaluation data locations
- Estimator
- Misc training loop information:
    - How often should we evaluate?
    - What metrics should we use?
    - How much computational power should we spend evaluating?

learn_runner

Runs an Experiment in either single worker or distributed mode depending on clues from the environment:

```
>>> config =
json.loads(os.environ.get('TF_CONFIG'))
```

Two Important Pieces of Information:

- **Where is everyone else?**
  ```
  >>> print(config['cluster'])
  {'master': ['localhost:0'], 'ps':
  ['localhost:1', 'localhost:2'], 'worker':
  ['localhost:3', 'localhost:4']}
  ```
- **Who am I?**
  ```
  >>> print(config['task'])
  {'type': 'worker', 'index': 1}
  ```

Lab: word2vec: learning and using word embeddings

Workshop section:

`word2vec`

Some Code Highlights

Conditional Graph Construction

```
def _model_fn(inputs, context_indices, mode):
  if mode == ModeKeys.INFER:
    sparse_index_tensor = tf.string_split(
        [tf.read_file(vocab_file)], delimiter='\n')
    ...
    reverse_index = tf.contrib.lookup.HashTable(
        tf.contrib.lookup.KeyValueTensorInitializer(
            index_tensor,
            tf.constant(range(vocab_size), dtype=tf.int64)
        ), 0)
    target_indices = reverse_index.lookup(inputs)
  else:
```

Queues to broadcast tensors across batches

```
range_queue = tf.train.range_input_producer(
    num_windows,
    shuffle=False,
    capacity=windows_per_batch * 2,
    num_epochs=num_epochs
)
indices = range_queue.dequeue_many(windows_per_batch)
```

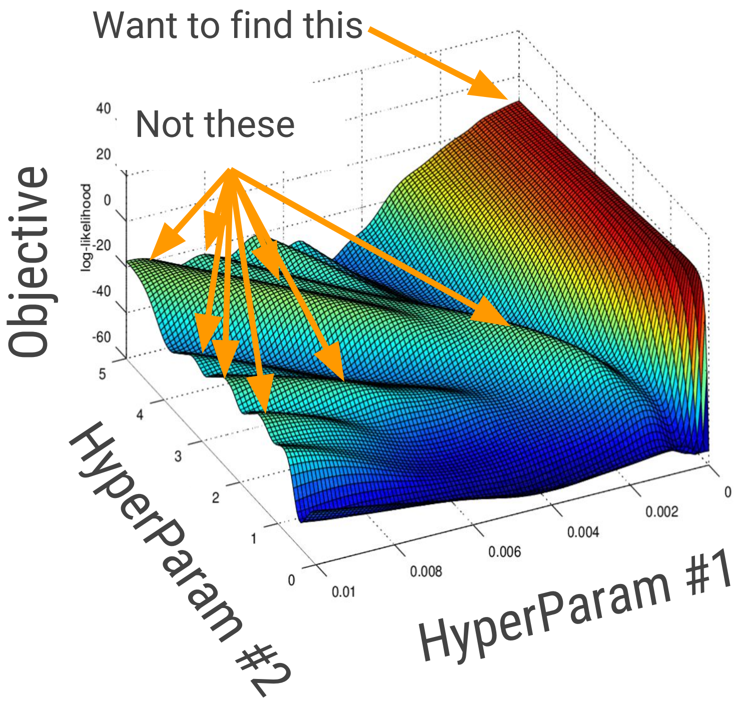
Variable Partitioning

```
with tf.device(tf.train.replica_device_setter()):
  with tf.variable_scope('nce',
                          partitioner=tf.fixed_size_partitioner(
                          num_partitions)):
        embeddings = tf.get_variable(
            'embeddings',
            shape=[vocab_size, embedding_size],
            dtype=tf.float32,
            initializer=tf.random_uniform_initializer(-1.0, 1.0)
        )
```

Hyperparameter Tuning

Automatically tune your model with HyperTune

- Automatic hyperparameter tuning service
- Build better performing models faster and save many hours of manual tuning
- Google-developed search algorithm efficiently finds better hyperparameters for your model/dataset
- One line of code: tf.summary.scalar('training/hptuning/metric', my_metric_tensor)



Break (10 min)

Up next:
Transfer Learning and Online Prediction

Transfer Learning (using the *Inception v3* model) and Online Prediction

Transfer Learning

- The Cloud Vision API is great…
  ...but what if you want to identify more personal/specialized image categories?
- It turns out that we can do this without needing too much data or training time.

Transfer Learning

- We can ‘bootstrap’ an existing model to reduce the effort needed to learn something new.
- we will use an *Inception v3* architecture model trained on ImageNet images:
    - use values generated from its penultimate "bottleneck" layer
    - train a new top layer that can recognize other classes of images.

Demo

Hug?

Don’t Hug?

Okay, how did we do that?

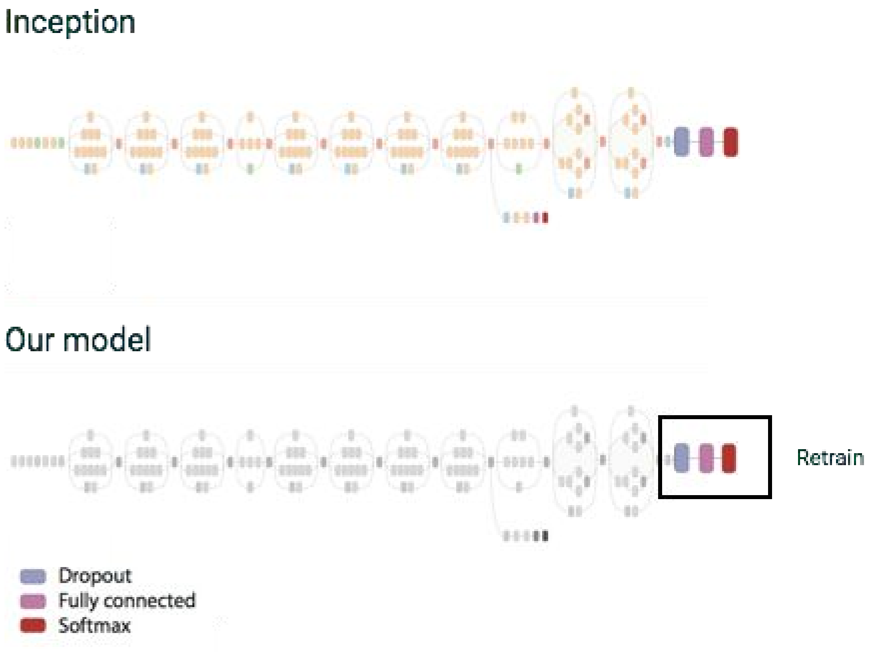
Our steps:

- Do image **pre-processing** using the Inception v3 model to get image *bottleneck embeds*: these express useful high-level image features

- **Train** a small model that sits “on top”, taking the embeds as input

- Tell Cloud ML that we want to use and **serve our trained model**.
- Use the **Cloud ML API for online prediction** with the model

Bootstrapping with the Inception model

- 

Our steps:

- Do image **pre-processing** using the Inception v3 model to get image *bottleneck embeds*: these express useful high-level image features
    - We'll use **Apache Beam (Cloud Dataflow)** for this
- **Train** a small model that sits "on top", taking the embeds as input
    - We'll do this **on Cloud ML**
- Tell Cloud ML that we want to use and **serve our trained model**.
- Use the **Cloud ML API for online prediction** with the model

Lab: Transfer Learning:

Bootstrapping the Inception v3 model to learn whether objects are “huggable” or not

+ making predictions using the Cloud ML API

Workshop section:

`transfer_learning/cloudml`

Generating the embeddings

```
checkpoint_path = (
      'gs://cloud-ml-data/img/flower_photos/inception_v3_2016_08_28.ckpt')

  def restore_from_checkpoint(self, checkpoint_path):
    # Get all variables to restore.
    all_vars = tf.contrib.slim.get_variables_to_restore(
        exclude=['InceptionV3/AuxLogits', 'InceptionV3/Logits', 'global_step'])
    saver = tf.train.Saver(all_vars)
    saver.restore(self.tf_session, checkpoint_path)
```

Generating the embeddings

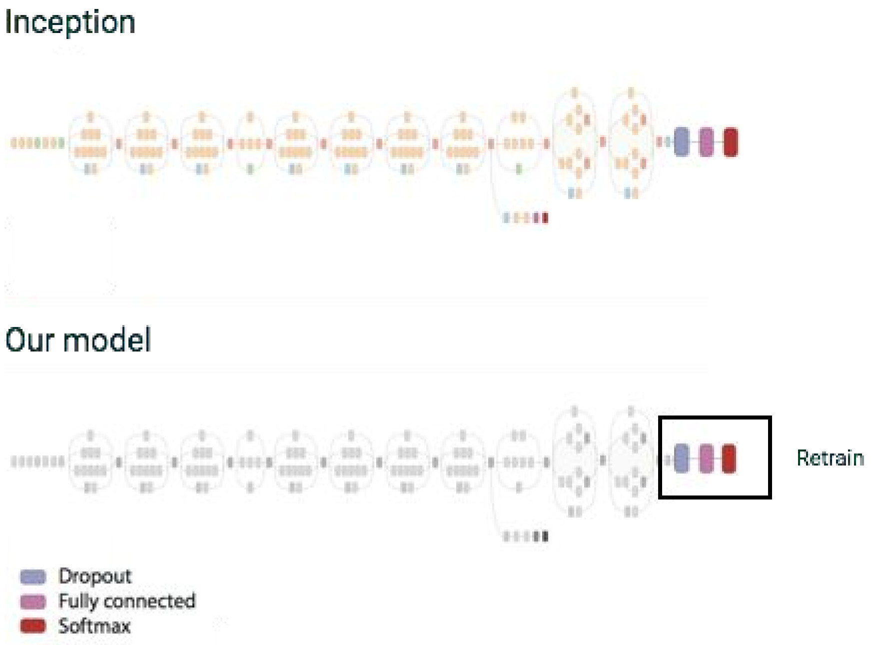
```
def build_graph(self):
  """Returns:
    input_jpeg: A tensor containing raw image bytes as the input layer.
    embedding: The embeddings tensor, that will be materialized later.
  """
  input_jpeg = tf.placeholder(tf.string, shape=None)
  … add some image conversion ops...
  inception_input = munged_image
  # Build Inception layers, which expect a tensor of type float from [-1, 1)
  # and shape [batch_size, height, width, channels].
  with slim.arg_scope(inception.inception_v3_arg_scope()):
    _, end_points = inception.inception_v3(inception_input, is_training=False)
  embedding = end_points['PreLogits']
  return input_jpeg, embedding
```

Generating the embeddings

```
def calculate_embedding(self, batch_image_bytes):
  """Get the embeddings for a given JPEG image.
  Args:
    batch_image_bytes: As if returned from [ff.read() for ff in file_list].
  Returns:
    The Inception embeddings (bottleneck layer output)
  """
  return self.tf_session.run(
      self.embedding, feed_dict={self.input_jpeg: batch_image_bytes})
```

Lab Step 1: Deploy the pre-processing pipeline

Bootstrapping with the Inception model: building our own model

- 

Building our own model

- Build model graph conditionals for train, evaluate, and predict
    - The prediction part will need to know how to generate the embeds
- Using “tf.layers” makes it easy to build the graph

```
def add_final_training_ops(
     self,embeddings, all_labels_count, bottleneck_tensor_size,
     hidden_layer_size=BOTTLENECK_TENSOR_SIZE / 4, dropout_keep_prob=None):
    with tf.name_scope('input'):
      bottleneck_input = tf.placeholder_with_default(
          embeddings, shape=[None, bottleneck_tensor_size],
          name='ReshapeSqueezed')
      bottleneck_with_no_gradient = tf.stop_gradient(bottleneck_input)
      with tf.name_scope('Wx_plus_b'):
        hidden = layers.fully_connected(bottleneck_with_no_gradient,
                                        hidden_layer_size)
        if dropout_keep_prob:
          hidden = tf.nn.dropout(hidden, dropout_keep_prob)
        logits = layers.fully_connected(
            hidden, all_labels_count, activation_fn=None)
    softmax = tf.nn.softmax(logits, name='softmax')
    return softmax, logits
```

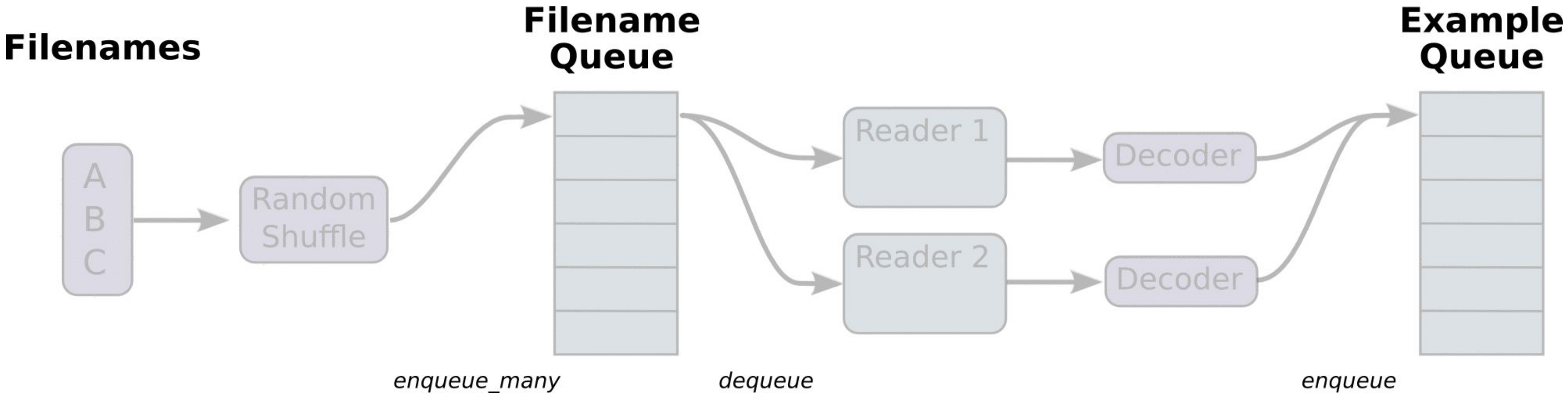
Lab Step 2:
Train our model on Cloud ML

TFRecords and Scalable I/O

What are TFRecords?

- Binary (protobuf) format
- Written and read by TensorFlow via gRPC
- Read in via input_fns which are part of the TensorFlow graph

Queues In TensorFlow!



Lab Step 3: Use the trained model for prediction

Exporting the trained model for prediction

- gcloud beta ml **models create** <model_name>
  gcloud beta ml models list
- gcloud beta ml **versions create** <version_name> \
  --model <model_name> \
  --origin gs://path/to/model"
- gcloud beta ml **versions set-default** <version_name> \
  --model <model_name>

Making online predictions

- gcloud beta ml **predict** --model <model_name> \
  --json-instances <request.json>
- Using the Cloud ML API (and the Google API client libraries)

Putting it all together: back to our demo!

Wrap up and Q&A

Amy
amyu@google.com
@amygdala

Eli
elibixby@google.com
@eli_bixby

Yufeng
yfg@google.com
@YufengG

Where to go for more (an incomplete list..)

- http://tensorflow.org : API, tutorials, resources...
- TensorFlow whitepaper: http://bit.ly/tensorflow-wp
- TensorFlow models: http://bit.ly/tensorflow-models
- Deep Learning Udacity course: http://bit.ly/udacity-tensorflow
- http://www.deeplearningbook.org/
- Neural Networks Demystified (video series): http://bit.ly/nn-demystified
- Gentle Guide to Machine Learning: http://bit.ly/gentle-ml
- Some more TensorFlow tutorials :https://github.com/pkmital/tensorflow_tutorials

Thank you!

Amy
amyu@google.com
@amygdala

Eli
elibixby@google.com
@eli_bixby

Yufeng
yfg@google.com
@YufengG

Thank you!

Le fin