Tensorflow Estimators

Mai Giménez

When everything
is possible,
the most probable outcome
is chaos.

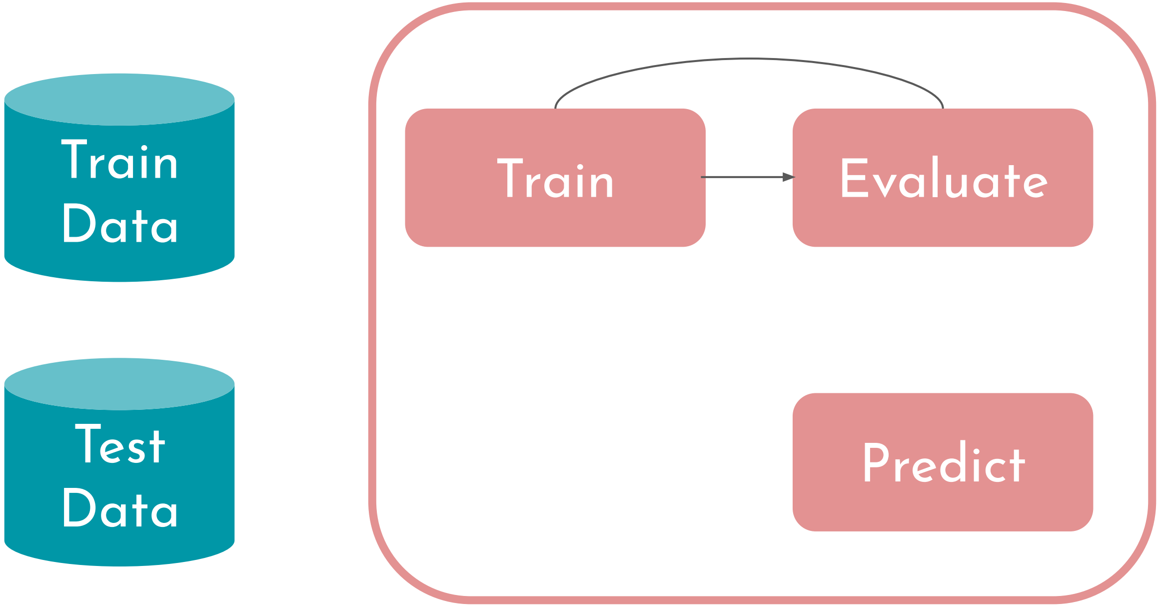
```python
def train(x_train, y_train, vocab_processor, x_dev, y_dev):
    # Training
    # ==================================================

    with tf.Graph().as_default():
        session_conf = tf.ConfigProto(
          allow_soft_placement=FLAGS.allow_soft_placement,
          log_device_placement=FLAGS.log_device_placement)
        sess = tf.Session(config=session_conf)
        with sess.as_default():
            cnn = TextCNN(
                sequence_length=x_train.shape[1],
                num_classes=y_train.shape[1],
                vocab_size=len(vocab_processor.vocabulary_),
                embedding_size=FLAGS.embedding_dim,
                filter_sizes=list(map(int, FLAGS.filter_sizes.split(","))),
                num_filters=FLAGS.num_filters,
                l2_reg_lambda=FLAGS.l2_reg_lambda)
```

```python
class LSTMOCR(object):
    def __init__(self, mode):
        self.mode = mode
        # image
        self.inputs = tf.placeholder(tf.float32, [None, FLAGS.image_height, FLAGS.image_width, FLAGS.image_channel])
        # SparseTensor required by ctc_loss op
        self.labels = tf.sparse_placeholder(tf.int32)
        # 1d array of size [batch_size]
        # self.seq_len = tf.placeholder(tf.int32, [None])
        # l2
        self._extra_train_ops = []

    def build_graph(self):
        self._build_model()
        self._build_train_op()

        self.merged_summay = tf.summary.merge_all()

    def _build_model(self):
        filters = [1, 64, 128, 128, FLAGS.out_channels]
        strides = [1, 2]

        feature_h = FLAGS.image_height
        feature_w = FLAGS.image_width

        count_ = 0
        min_size = min(FLAGS.image_height, FLAGS.image_width)
        while min_size > 1:
            min_size = (min_size + 1) // 2
            count_ += 1
        assert (FLAGS.cnn_count <= count_, "FLAGS.cnn_count should be <= {}!".format(count_))

        # CNN part
```

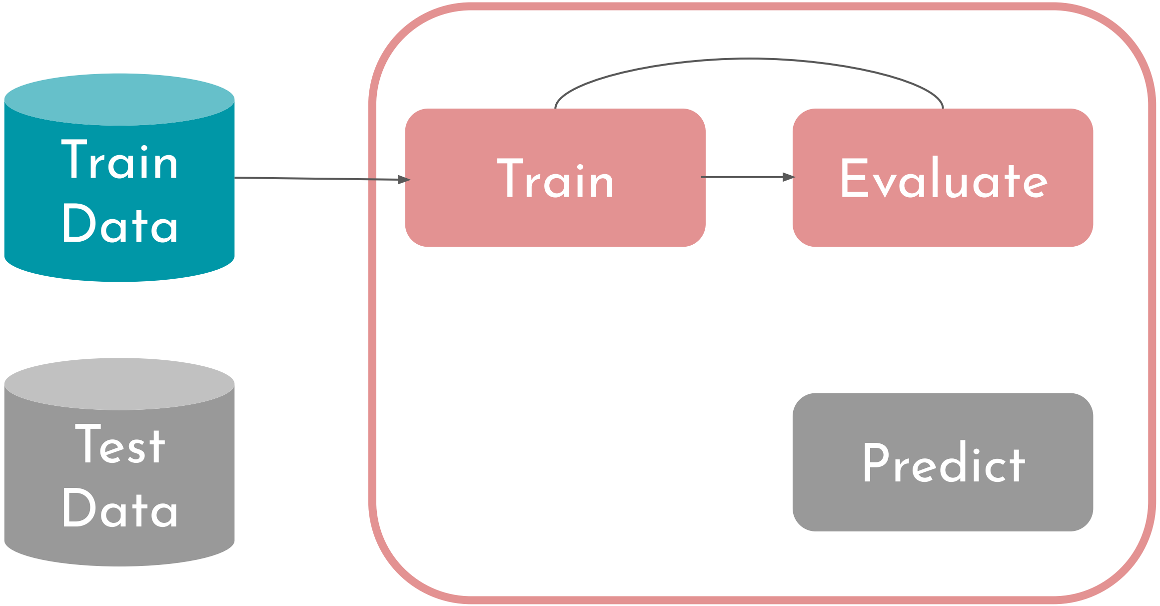
```python
def net(data_path, input_image):
    layers = (
        'conv1_1', 'relu1_1', 'conv1_2', 'relu1_2', 'pool1',

        'conv2_1', 'relu2_1', 'conv2_2', 'relu2_2', 'pool2',

        'conv3_1', 'relu3_1', 'conv3_2', 'relu3_2', 'conv3_3',
        'relu3_3', 'conv3_4', 'relu3_4', 'pool3',

        'conv4_1', 'relu4_1', 'conv4_2', 'relu4_2', 'conv4_3',
        'relu4_3', 'conv4_4', 'relu4_4', 'pool4',

        'conv5_1', 'relu5_1', 'conv5_2', 'relu5_2', 'conv5_3',
        'relu5_3', 'conv5_4', 'relu5_4'
    )

    data = scipy.io.loadmat(data_path)
    mean = data['normalization'][0][0][0]
    mean_pixel = np.mean(mean, axis=(0, 1))
    weights = data['layers'][0]

    net = {}
    current = input_image
    for i, name in enumerate(layers):
        kind = name[:4]
        if kind == 'conv':
            kernels, bias = weights[i][0][0][0][0]
            # matconvnet: weights are [width, height, in_channels, out_channels]
            # tensorflow: weights are [height, width, in_channels, out_channels]
            kernels = np.transpose(kernels, (1, 0, 2, 3))
            bias = bias.reshape(-1)
```


This is fine. I'm okay with the events that are unfolding currently.

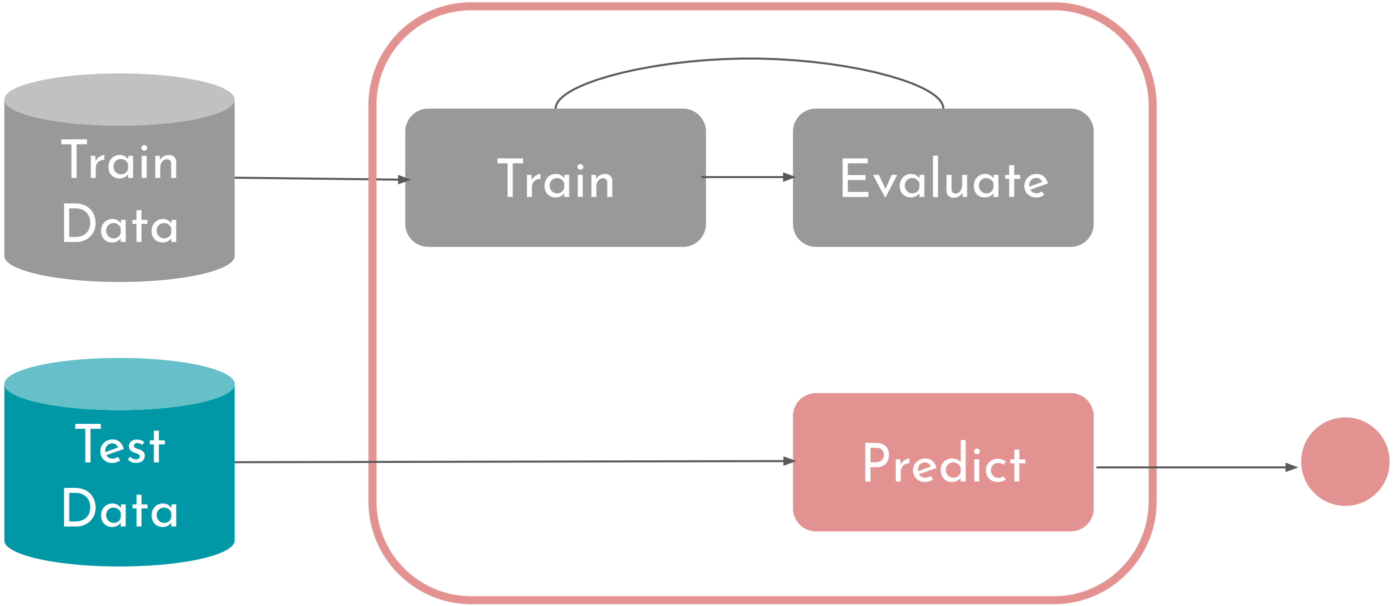
Model



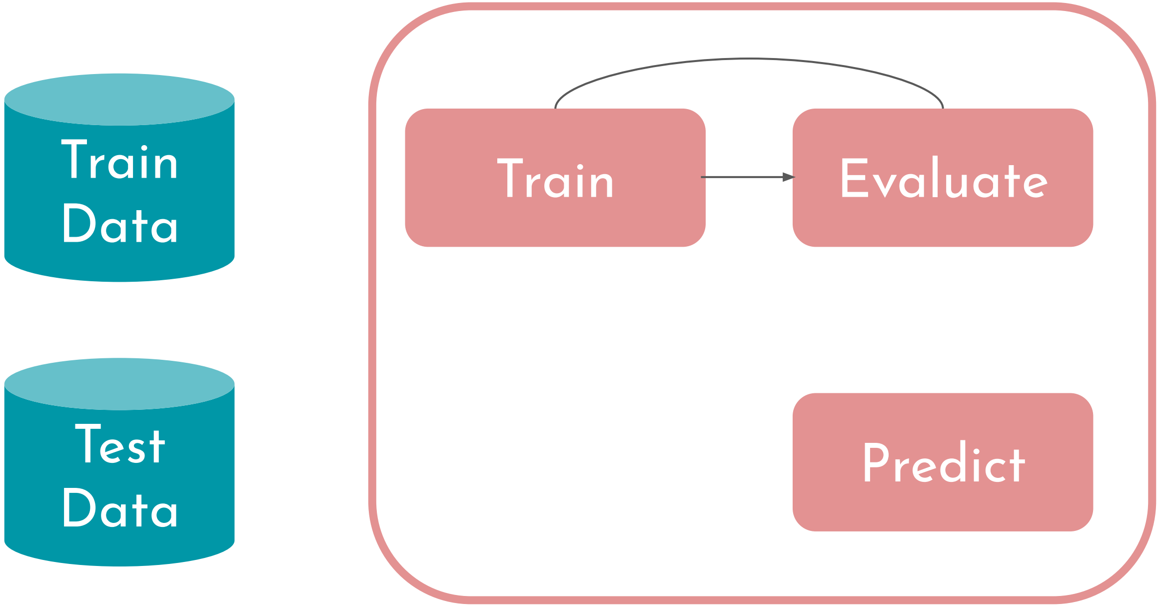
Training



Inference

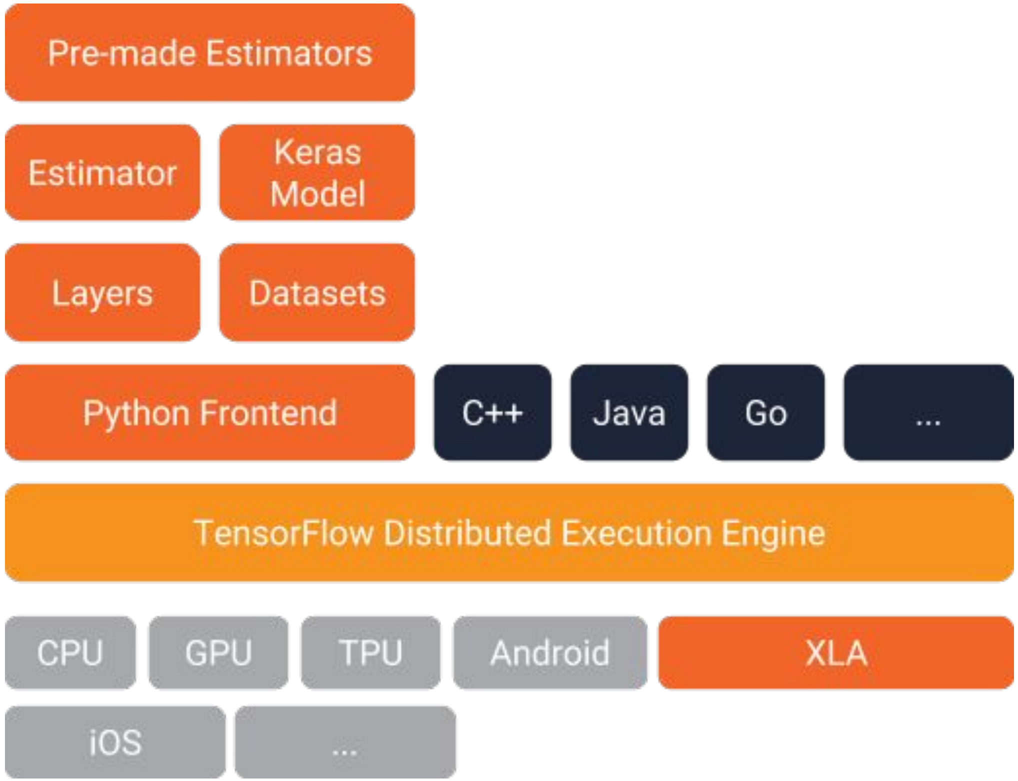


Model



Estimators

Steering users towards good practices.



Pre-made Estimators
Estimator
Keras Model
Layers
Datasets
Python Frontend
C++
Java
Go
...
TensorFlow Distributed Execution Engine
CPU
GPU
TPU
Android
XLA
iOS
...

Estimators

Estimators represents a complete model.
The Estimator API provides methods to train the model, to judge the model's accuracy, and to generate predictions.

Estimators

```
train()
evaluate()
predict()
export_savedmodel
```

Canned Estimators

or

Custom Estimators

Canned Estimators

Commonly used architectures.
Subclasses of tf.estimator.Estimator.
Representation of the whole model.

How to Use a Canned Estimator

1. Create input function(s).
2. Define the model's feature columns.
3. Instantiate an Estimator.
4. Call one of its methods: train, evaluate or predict, passing the corresponding input data.

How to Use a Canned Estimator

1. **Create input function(s).**
2. Define the model's feature columns.
3. Instantiate an Estimator.
4. Call one of its methods: train, evaluate or predict, passing the corresponding input data.

Input Function

```
def train_input_fn(features, labels, batch_size):
    """An input function for training"""
    # Convert the inputs to a Dataset.
    dataset = tf.data.Dataset.from_tensor_slices(
                (dict(features),labels))
    # Shuffle, repeat, and batch the examples.
    return dataset.shuffle(1000).repeat().batch(batch_size)
```

How to Use a Canned Estimator

1. Create input function(s).
2. **Define the model's feature columns.**
3. Instantiate an Estimator.
4. Call one of its methods: train, evaluate or predict, passing the corresponding input data.

Feature columns

```
# Feature columns describe how to use the input.
my_feature_columns = []
for key in train_x.keys():
    my_feature_columns.append(
            tf.feature_column.numeric_column(key=key))
```

How to Use a Canned Estimator

1. Create input function(s).
2. Define the model's feature columns.
3. **Instantiate an Estimator.**
4. Call one of its methods: train, evaluate or predict, passing the corresponding input data.

Instantiate and Estimator

```
# Build a DNN with 2 hidden layers and 10 nodes in each
hidden layer.
classifier = tf.estimator.DNNClassifier(
                 feature_columns=my_feature_columns,
                 # Two hidden layers of 10 nodes each.
                 hidden_units=[10, 10],
                 # The model must choose between 3 classes.
                 n_classes=3)
```

How to Use a Canned Estimator

1. Create input function(s).
2. Define the model's feature columns.
3. Instantiate an Estimator.
4. **Call one of its methods: train, evaluate or predict, passing the corresponding input data.**

Train the model

```
# Train the Model.
classifier.train(
    input_fn=lambda: iris_data.train_input_fn(
                                            train_x, train_y, batch_size),
    steps=train_steps)
```

Evaluate the model

```
# Evaluate the model.
eval_result = classifier.evaluate(
        input_fn=lambda:iris_data.eval_input_fn(
                                              test_x, test_y, batch_size))
print('Test set accuracy:{accuracy:0.3f}'.format(
                                                     **eval_result))
```

Inference using the model

```
# Generate predictions from the model.
expected = ['Setosa', 'Versicolor', 'Virginica']
predict_x = { 'SepalLength': [5.1, 5.9, 6.9],
              'SepalWidth': [3.3, 3.0, 3.1],
              'PetalLength': [1.7, 4.2, 5.4],
              'PetalWidth': [0.5, 1.5, 2.1]}
predictions = classifier.predict(
       input_fn=lambda:iris_data.eval_input_fn(
                                 predict_x, batch_size=batch_size)
```

Canned Estimators
or
Custom Estimators

Custom Estimators

Flexibility on top of good practices.
Instances of tf.estimator.Estimator.
We must write the model.

How to Use a Custom Estimator

1. Create input function(s).
2. Define the model's feature columns.
3. **Create a custom Estimator.**
4. Instantiate an Estimator.
5. Call one of its methods: train, evaluate or predict, passing the corresponding input data.

Creating a Custom Estimator

```
my_estimator = tf.estimator.Estimator(model_fn=model_fn,
                                      params=params,
                                      config=run_config)
```

Input of a Custom Estimator

```
def model_fn(features, labels, mode, params) :
    input_layer = tf.contrib.layers.embed_sequence(
                          features['sentence'],
                          params['vocab_size'],
                          params['embedding_size'],
                          initializer=params['initializer'])
```

Hidden Layers

```
conv = tf.layers.conv1d(inputs=input_layer,
                        filters=32,
                         kernel_size=3,
                         padding='same',
                         activation=tf.nn.relu,
                         name='conv_1d')
pool = tf.reduce_max(input_tensor=conv, axis=1)
```

Hidden and Output Layers

```
hidden = tf.layers.dense(inputs=pool,
                         units=250,
                         activation=tf.nn.relu)
logits = tf.layers.dense(inputs=hidden,
                         units=2,
                         activation=None)
```

Optimizer

```
optimizer = tf.train.AdamOptimizer()
def _train_op_fn(loss):
    tf.summary.scalar('loss', loss)
    return optimizer.minimize(
                loss=loss,
                global_step=tf.train.get_global_step())
```

Implement train

```
if mode == tf.estimator.ModeKeys.TRAIN:
    return tf.estimator.EstimatorSpec(mode,
                                      loss,
                                       _train_op_fn)
```

Implement evaluation

```
eval_metric_ops = {
    'accuracy': tf.metrics.accuracy(labels, predictions)
    }
if mode == tf.estimator.ModeKeys.EVAL:
    return tf.estimator.EstimatorSpec(mode,
                                      loss,
                                      eval_metric_ops)
```

Estimators

Summary

Datasets

Feeding complex pipelines

How to Use Datasets

1. Import data
2. Manipulate the data
3. Create an Iterator
4. Consume Data from the iterator.

How to Use Datasets

1. **Import data**

   ```
   from_generator
   from_sparse_tensor_slices
   from_tensor_slices
   from_tensors
   from a csv file.
   ```

2. Manipulate the data
3. Create an Iterator
4. Consume Data from the iterator.

Import data

```
dataset = tf.data.Dataset.from_generator(
                                        example_generator_fn,
                                        output_types=output_types,
                                        output_shapes=output_shapes)
```

How to Use Datasets

1. Import data
2. **Manipulate the data**

   `tf.data.Dataset.map`

   `tf.data.Dataset.shuffle`

   `tf.data.Dataset.repeat`

   `tf.data.Dataset.batch`

3. Create an Iterator
4. Consume Data from the iterator.

Manipulate the data

```
# Shuffle, repeat, and batch the examples.
dataset = dataset.shuffle(1000).repeat().batch(batch_size)
```

How to Use Datasets

1. Import data
2. Manipulate the data
3. **Create an Iterator**

   `make_one_shot_iterator` `make_initializable_iterator`
4. Consume Data from the iterator.

Create an iterator

```
iterator = dataset.make_one_shot_iterator()
```

How to Use Datasets

1. Import data
2. Manipulate the data
3. Create an Iterator
4. **Consume Data from the iterator.**

```
iterator.get_next()
```

Datasets

Summary

Tf-Hub

Reusable modules

How to use tf-hub

```
embedded_feature = hub.text_embedding_column(key="sentence",
    module_spec="https://tfhub.dev/google/nnlm-en-dim128/1")
estimator = tf.estimator.DNNClassifier(
                                    hidden_units=[500, 100],
                                    feature_columns=[embedded_feature],
                                    n_classes=2,
                                    optimizer=tf.train.AdagradOptimizer
                                    (learning_rate=0.003))
```

Tf-Hub

Reusable modules

In Summary

Bits of knowledge to take away

Take away

1. Estimators are a good idea.
2. Datasets allow building high performance complex pipelines.
3. There are state-of-the-art modules ready to use tf.hub

Take away

1. **Estimators are a good idea.**
2. Datasets allow building high performance complex pipelines.
3. There are state-of-the-art modules ready to use tf.hub

Take away

1. Estimators are a good idea.
2. **Datasets allow building high performance complex pipelines.**
3. There are state-of-the-art modules ready to use tf.hub

Take away

1. Estimators are a good idea.
2. Datasets allow building high performance complex pipelines.
3. **There are state-of-the-art modules ready to use tf.hub**

Tensorflow Estimators

Mai Gimenez

Twitter: @mai[dot]gimenez
Github: @maigimenez