TensorFlow 2.0 Beta is available **Learn more** (/beta/)

Estimators

This document introduces **`tf.estimator`** (https://www.tensorflow.org/api_docs/python/tf/estimator)—a high-level TensorFlow API that greatly simplifies machine learning programming. Estimators encapsulate the following actions:

- training
- evaluation
- prediction
- export for serving

You may either use the pre-made Estimators we provide or write your own custom Estimators. All Estimators—whether pre-made or custom—are classes based on the **`tf.estimator.Estimator`** (https://www.tensorflow.org/api_docs/python/tf/estimator/Estimator) class.

For a quick example try Estimator tutorials (https://www.tensorflow.org/tutorials/estimators/linear). To see each sub-topic in depth, see the Estimator guides (/guide/premade_estimators). For an overview of the API design, see our white paper here (https://arxiv.org/abs/1708.02637).

Note: TensorFlow also includes a deprecated **`Estimator`** class at **`tf.contrib.learn.Estimator`** (https://www.tensorflow.org/api_docs/python/tf/contrib/learn/Estimator), which you should not use.

Advantages of Estimators

Estimators provide the following benefits:

- You can run Estimator-based models on a local host or on a distributed multi-server environment without changing your model. Furthermore, you can run Estimator-based models on CPUs, GPUs, or TPUs without recoding your model.
- Estimators simplify sharing implementations between model developers.
- You can develop a state of the art model with high-level intuitive code. In short, it is generally much easier to create models with Estimators than with the low-level TensorFlow APIs.
- Estimators are themselves built on `tf.keras.layers` (https://www.tensorflow.org/api_docs/python/tf/keras/layers), which simplifies customization.
- Estimators build the graph for you.
- Estimators provide a safe distributed training loop that controls how and when to:
    - build the graph
    - initialize variables
    - load data
    - handle exceptions
    - create checkpoint files and recover from failures
    - save summaries for TensorBoard

When writing an application with Estimators, you must separate the data input pipeline from the model. This separation simplifies experiments with different data sets.

Pre-made Estimators

Pre-made Estimators enable you to work at a much higher conceptual level than the base TensorFlow APIs. You no longer have to worry about creating the computational graph or sessions since Estimators handle all the "plumbing" for you. That is, pre-made Estimators create and manage `tf.Graph` (https://www.tensorflow.org/api_docs/python/tf/Graph) and `tf.Session` (https://www.tensorflow.org/api_docs/python/tf/Session) objects for you. Furthermore, pre-made Estimators let you experiment with different model architectures by making only minimal code changes. `tf.estimator.DNNClassifier` (https://www.tensorflow.org/api_docs/python/tf/estimator/DNNClassifier), for example, is a pre-made Estimator class that trains classification models based on dense, feed-forward neural networks.

Structure of a pre-made Estimators program

A TensorFlow program relying on a pre-made Estimator typically consists of the following four steps:

1. **Write one or more dataset importing functions.** For example, you might create one function to import the training set and another function to import the test set. Each dataset importing function must return two objects:

   - a dictionary in which the keys are feature names and the values are Tensors (or SparseTensors) containing the corresponding feature data
   - a Tensor containing one or more labels

   For example, the following code illustrates the basic skeleton for an input function:

   ```
   def input_fn(dataset):
      ...  # manipulate dataset, extracting the feature dict and the
      return feature_dict, label
   ```

   (See Importing Data (https://www.tensorflow.org/guide/datasets) for full details.)

2. **Define the feature columns.** Each `tf.feature_column` (https://www.tensorflow.org/api_docs/python/tf/feature_column) identifies a feature name, its type, and any input pre-processing. For example, the following snippet creates three feature columns that hold integer or floating-point data. The first two feature columns simply identify the feature's name and type. The third feature column also specifies a lambda the program will invoke to scale the raw data:

```
# Define three numeric feature columns.
population = tf.feature_column.numeric_column('population')
crime_rate = tf.feature_column.numeric_column('crime_rate')
median_education = tf.feature_column.numeric_column('median_educa
                    normalizer_fn=lambda x: x - global_education_
```

3. **Instantiate the relevant pre-made Estimator.** For example, here's a sample instantiation of a pre-made Estimator named `LinearClassifier`:

```
# Instantiate an estimator, passing the feature columns.
estimator = tf.estimator.LinearClassifier(
    feature_columns=[population, crime_rate, median_education])
```

4. **Call a training, evaluation, or inference method.** For example, all Estimators provide a `train` method, which trains a model.

```
# `input_fn` is the function created in Step 1
estimator.train(input_fn=my_training_set, steps=2000)
```

Benefits of pre-made Estimators

Pre-made Estimators encode best practices, providing the following benefits:

- Best practices for determining where different parts of the computational graph should run, implementing strategies on a single machine or on a cluster.
- Best practices for event (summary) writing and universally useful summaries.

If you don't use pre-made Estimators, you must implement the preceding features yourself.

Custom Estimators

The heart of every Estimator—whether pre-made or custom—is its **model function**, which is a method that builds graphs for training, evaluation, and prediction. When you are using a pre-made Estimator, someone else has already implemented the model function. When relying on a custom Estimator, you must write the model function yourself. A companion document (https://www.tensorflow.org/guide/custom_estimators) explains how to write the model function.

Recommended workflow

We recommend the following workflow:

1. Assuming a suitable pre-made Estimator exists, use it to build your first model and use its results to establish a baseline.
2. Build and test your overall pipeline, including the integrity and reliability of your data with this pre-made Estimator.
3. If suitable alternative pre-made Estimators are available, run experiments to determine which pre-made Estimator produces the best results.
4. Possibly, further improve your model by building your own custom Estimator.

Creating Estimators from Keras models

You can convert existing Keras models to Estimators. Doing so enables your Keras model to access Estimator's strengths, such as distributed training. Call **`tf.keras.estimator.model_to_estimator`** (https://www.tensorflow.org/api_docs/python/tf/keras/estimator/model_to_estimator) as in the following sample:

```
# Instantiate a Keras inception v3 model.
keras_inception_v3 = tf.keras.applications.inception_v3.InceptionV3(weights=None

# Compile model with the optimizer, loss, and metrics you'd like to train with.
keras_inception_v3.compile(optimizer=tf.keras.optimizers.SGD(lr=0.0001, momentum
                          loss='categorical_crossentropy',
                          metric='accuracy')

# Create an Estimator from the compiled Keras model. Note the initial model
# state of the keras model is preserved in the created Estimator.
est_inception_v3 = tf.keras.estimator.model_to_estimator(keras_model=keras_incep

# Treat the derived Estimator as you would with any other Estimator.
# First, recover the input name(s) of Keras model, so we can use them as the
# feature column name(s) of the Estimator input function:
keras_inception_v3.input_names  # print out: ['input_1']

# Once we have the input name(s), we can create the input function, for example,
# for input(s) in the format of numpy ndarray:
train_input_fn = tf.estimator.inputs.numpy_input_fn(
    x={"input_1": train_data},
    y=train_labels,
    num_epochs=1,
    shuffle=False)

# To train, we call Estimator's train function:
est_inception_v3.train(input_fn=train_input_fn, steps=2000)
```

Note that the names of feature columns and labels of a keras estimator come from the corresponding compiled keras model. For example, the input key names for `train_input_fn` above can be obtained from `keras_inception_v3.input_names`, and similarly, the predicted output names can be obtained from `keras_inception_v3.output_names`.

For more details, please refer to the documentation for `tf.keras.estimator.model_to_estimator` (https://www.tensorflow.org/api_docs/python/tf/keras/estimator/model_to_estimator).