This document covers the usage of the TPUEstimator API with Cloud TPU. TPUEstimator simplifies running models on a Cloud TPU by handling numerous low-level, hardware-specific details.

Models written using TPUEstimator work across CPUs, GPUs, single TPU devices, and whole TPU pods, generally with no code changes. TPUEstimator also makes it easier to achieve maximum performance by automatically performing some optimizations on your behalf.

To learn how machine learning workloads operate on TPU hardware in general, read the System Architecture (/tpu/docs/system-architecture#software_architecture) documentation.

At a high-level, the standard TensorFlow Estimator API (https://www.tensorflow.org/api_docs/python/tf/estimator/Estimator) provides:

- `Estimator.train()` - train a model on a given input for a fixed number of steps.
- `Estimator.evaluate()` - evaluate the model on a test set.
- `Estimator.predict()` - run inference using the trained model.
- `Estimator.export_savedmodel()` - export your model for serving.

In addition, `Estimator` includes default behavior common to training jobs, such as saving and restoring checkpoints, creating summaries for TensorBoard, etc.

`Estimator` requires you to write a `model_fn` and an `input_fn` that correspond to the model and input portions of your TensorFlow graph.

The `TPUEstimator` (https://www.tensorflow.org/api_docs/python/tf/contrib/tpu/TPUEstimator) wraps the computation (the `model_fn`) and distributes it to all available Cloud TPU cores. The learning rate must be tuned with the batch size.

- The `input_fn` function models the input pipeline running on the remote host CPU. Use `tf.data` to program the input ops as described in the programmer's guide (https://www.tensorflow.org/programmers_guide/datasets). Each invocation handles input of the global batch onto one device. The shard batch size is retrieved from `params['batch_size']`. Pro tip: return a dataset instead of tensors for optimal performance.
- The `model_fn` function models the computation being replicated and distributed to the TPUs. The computation should contain only ops supported by Cloud TPU. TensorFlow ops (/tpu/docs/tensorflow-ops) includes the list of available ops.

The following code demonstrates training a MNIST model using `TPUEstimator`:

The following section covers the new concepts introduced in the above sample, to help you use Cloud TPU effectively.

TPUEstimator uses an in-graph replication (https://www.tensorflow.org/deploy/distributed) approach to running TensorFlow programs. In-graph (single-session) replication differs from the between-graph (multi-session) replication training typically used in distributed TensorFlow. The major differences include:

1. In TPUEstimator, the TensorFlow session master is not local. Your Python program creates a single graph that is replicated across all of the cores in the Cloud TPU. A typical configuration sets the TensorFlow session master to be the first worker.
2. The input pipeline is placed on remote hosts (instead of local) to ensure that training examples can be fed to the Cloud TPU as fast as possible. A dataset (`tf.data`) is required.

3. Cloud TPU workers operate synchronously, with each worker performing the same step at the same time.

We recommend that you port a small, simple model first and test end-to-end behavior. Doing so helps solidify your familiarity with the basic concepts of `TPUEstimator`. When your simple model runs, gradually add more functionality.

See the tutorials (/tpu/docs/tutorials/) for a set of sample models and instructions for running them with Cloud TPU. Additional models are available on GitHub (https://github.com/tensorflow/tpu).

To convert your code from `tf.estimator.Estimator` class to use `tf.contrib.tpu.TPUEstimator`, change the following:

- Change `tf.estimator.RunConfig` to `tf.contrib.tpu.RunConfig`.
- Set `TPUConfig` (part of the `tf.contrib.tpu.RunConfig`) to specify the `iterations_per_loop`. `iterations_per_loop` is the number of iterations to run on the Cloud TPU for one `session.run` call (per training loop).

Cloud TPU runs a specified number of iterations of the training loop before returning to the host. No checkpoints or summaries are saved until all Cloud TPU iterations are run.

- In `model_fn`, use `tf.contrib.tpu.CrossShardOptimizer` to wrap your optimizer. For example:

- Change `tf.estimator.Estimator` to `tf.contrib.tpu.TPUEstimator`.

The default `RunConfig` saves summaries for TensorBoard every 100 steps and writes checkpoints every 10 minutes.

There are two reasons:

1. Your application code runs on the client while the TPU computation is executed on the `worker`. Input pipeline ops must be placed on the remote worker for good performance. Only `tf.data` (Dataset) supports this.
2. In order to amortize the TPU launch cost, the model training step is wrapped in a `tf.while_loop`, such that one `Session.run` actually runs many iterations for a single training loop. Currently only `tf.data` can be wrapped by a `tf.while_loop`.

You can profile model training performance using the profiler (/tpu/docs/cloud-tpu-tools#profile_tab) provided for TensorBoard.