@Subaru

Enterprise IT & Software Technologies

☐
POSTED BY

SUBARU KOKUBUN
POSTED ON

APRIL 10, 2018
POSTED UNDER

CLOUD, INTELLIGENCE

End-to-End Example for Distributed TensorFlow

Current computing technologies for AI (by giant Google, Microsoft, etc) is focusing more practical IT infrastructures or services, which enables to run high-throughput and massive workloads with cloud infrastructure and device-acceleration (GPU or TPU) integrated. In such a situation, the programming for distributed computing is an important piece of technologies.

In this blog I want to show several fundamentals and options for running distributed training with popular TensorFlow framework, and here (in this post) we begin the basic end-to-end example for your first start with programming, running, and evaluation.
There are several ideas to run distributed training with tensorflow (like Horovod, etc), but here I show the regular Distributed TensorFlow with standard `MonitoredTrainingSession`.
In the next post I'll show other (advanced) options based on this example.

Here I use only pure tensorflow library without Keras or other helper functions, and you can soon run this code on your computing environment in house or cloud infrastructures.
I don't go so far into topology or programming patterns for distributed computing ideas itself (I focus on the tutorial for your first understanding), but I hope this post helps you to start your distributed computing by popular tensorflow.

Preparing your multiple machines (cluster)

Before starting, you must prepare your multiple machines with python and TensorFlow installed and configured (GPU-accelerated, etc).

Here I don't explain how to setup your environment, but you can also use **Google Cloud Machine Learning Engine (Cloud ML)** or **Azure Batch AI**, which significantly simplifies your provisioning rather than the basic cloud infrastructure (IaaS) like Google Compute VM, Amazon EC2, or Azure VM. (Google Cloud ML is fully-managed and Azure Batch AI is IaaS based. For both Google Cloud ML and Azure Batch AI, you need to specify the following cluster spec and start the following `MonitoredTrainingSession` in your code.)

In this post I used Azure Batch AI for the following samples and here (https://tsmatz.wordpress.com/2018/01/30/azure-batch-ai-how-it-works/) is useful resource for your first start of Azure Batch AI. In the practical execution, you can also use infini-band network for inter-node communications.

Topology (Brief Overview)

There exist a lot of resources (articles) explaining the topology and programming of Distributed TensorFlow, but let me summarize brief concepts before starting. (You need to know for running your code.)

Distributed TensorFlow consists of 3 types of computing nodes, called "**parameter** node", "**worker** node", and "**master** node".

Computing session runs on multiple (or single) worker nodes. Computed parameters are kept by the parameter node and shared with workers. (You can also run multiple parameter nodes for each parameter blocks, which enables high-throughput IOs for writing and reading parameters. If you don't specify ps (=parameter server) tasks for variables, the round-robin strategy over all ps tasks is applied.)
One of workers (among worker nodes) must be master node (chief) and master coordinates all workloads in each worker nodes.

Programming Sample

Because I want to focus only on our concerns, here I use simple graph (neural network) with well-known MNIST (hand-writing digits) and I referred here (https://github.com/tsmatsuz/tensorflow-mnist-batch-read-and-train-tutorial/blob/master/mnist_tf.py) (non-distributed training example) which only use standard tensorflow functions without any other helper classes or functions.
Now I modified this original code for Distributed TensorFlow and the following is our complete code for distributed training.
The highlighted line is modified for Distributed TensorFlow. Please compare the following distributed one with the original non-distributed one. (Here we use asynchronous training, with which each nodes has independent training loop.)

```
from __future__ import absolute_import
from __future__ import division
from __future__ import print_function

import sys
import argparse
import math

import tensorflow as tf

FLAGS = None
batch_size = 100

cluster = None
server = None
is_chief = False
num_workers = 2

class _MasterNodeHook(tf.train.SessionRunHook):
  def begin(self):
    # start without checkpoint
    self._step = -1

def main(_):
  with tf.device(tf.train.replica_device_setter(
    worker_device="/job:%s/task:%d" % (FLAGS.job_name, FLAGS.task_index),
    cluster=cluster)):

    ###
    ### Training
    ###

    #
    # read training data
    #

    # image - 784 (=28 x 28) elements of grey-scaled integer value [0, 1]
    # label - digit (0, 1, ..., 9)
    train_queue = tf.train.string_input_producer(
      [FLAGS.train_file],
      num_epochs = 2) # data is repeated and it raises OutOfRange when data i
    train_reader = tf.TFRecordReader()
    _, train_serialized_exam = train_reader.read(train_queue)
    train_exam = tf.parse_single_example(
      train_serialized_exam,
      features={
        'image_raw': tf.FixedLenFeature([], tf.string),
        'label': tf.FixedLenFeature([], tf.int64)
      })
    train_image = tf.decode_raw(train_exam['image_raw'], tf.uint8)
    train_image.set_shape([784])
    train_image = tf.cast(train_image, tf.float32) * (1. / 255)
    train_label = tf.cast(train_exam['label'], tf.int32)
    train_batch_image, train_batch_label = tf.train.batch(
      [train_image, train_label],
      batch_size=batch_size)

```

```
    #
    # define training graph
    #

    # define input
    plchd_image = tf.placeholder(
      dtype=tf.float32,
      shape=(batch_size, 784))
    plchd_label = tf.placeholder(
      dtype=tf.int32,
      shape=(batch_size))

    # define network and inference
    # (simple 2 fully connected hidden layer : 784-&gt;128-&gt;64-&gt;10)
    with tf.name_scope('hidden1'):
      weights = tf.Variable(
        tf.truncated_normal(
          [784, 128],
          stddev=1.0 / math.sqrt(float(784))),
        name='weights')
      biases = tf.Variable(
        tf.zeros([128]),
        name='biases')
      hidden1 = tf.nn.relu(tf.matmul(plchd_image, weights) + biases)
    with tf.name_scope('hidden2'):
      weights = tf.Variable(
        tf.truncated_normal(
          [128, 64],
          stddev=1.0 / math.sqrt(float(128))),
        name='weights')
      biases = tf.Variable(
        tf.zeros([64]),
        name='biases')
      hidden2 = tf.nn.relu(tf.matmul(hidden1, weights) + biases)
    with tf.name_scope('softmax_linear'):
      weights = tf.Variable(
        tf.truncated_normal(
          [64, 10],
          stddev=1.0 / math.sqrt(float(64))),
      name='weights')
      biases = tf.Variable(
        tf.zeros([10]),
        name='biases')
      logits = tf.matmul(hidden2, weights) + biases

    # define optimization
    global_step = tf.train.create_global_step() # start without checkpoint
    optimizer = tf.train.GradientDescentOptimizer(
      learning_rate=0.07)
    loss = tf.losses.sparse_softmax_cross_entropy(
      labels=plchd_label,
      logits=logits)
    train_op = optimizer.minimize(
      loss=loss,
      global_step=global_step)

    #
```

```
    # run session
    #

    if is_chief:
      hooks = [_MasterNodeHook()]
    else:
      hooks = []
    with tf.train.MonitoredTrainingSession(
      master=server.target,
      checkpoint_dir=FLAGS.out_dir,
      hooks=hooks,
      is_chief=is_chief) as sess:

      # when data is over, OutOfRangeError occurs and ends with MonitoredSess

      step = 0
      array_image, array_label = sess.run(
        [train_batch_image, train_batch_label])
      while not sess.should_stop():
        feed_dict = {
          plchd_image: array_image,
          plchd_label: array_label
        }
        _, loss_value, array_image, array_label = sess.run(
          [train_op, loss, train_batch_image, train_batch_label],
          feed_dict=feed_dict)
        step += 1
        if step % 100 == 0:
          print("Worker: Step %d (Loss: %.2f)" % (step, loss_value))

    print('training finished')

if __name__ == '__main__':
  parser = argparse.ArgumentParser()
  parser.add_argument(
    '--train_file',
    type=str,
    default='/home/demouser/train.tfrecords',
    help='File path for the training data.')
  parser.add_argument(
    '--out_dir',
    type=str,
    default='/home/demouser/out',
    help='Dir path for the model and checkpoint output.')
  parser.add_argument(
    '--job_name',
    type=str,
    required=True,
    help='job name (parameter or worker) for cluster')
  parser.add_argument(
    '--task_index',
    type=int,
    required=True,
    help='index number in job for cluster')
  FLAGS, unparsed = parser.parse_known_args()

  # start server
```

```
    cluster = tf.train.ClusterSpec({
      'ps': ['10.0.0.6:2222'],
      'worker': [
        '10.0.0.4:2222',
        '10.0.0.5:2222'
      ]})
    server = tf.train.Server(
      cluster,
      job_name=FLAGS.job_name,
      task_index=FLAGS.task_index)
    if FLAGS.job_name == "ps":
      server.join()
    elif FLAGS.job_name == "worker":
      is_chief = (FLAGS.task_index == 0)
      tf.app.run(main=main, argv=[sys.argv[0]] + unparsed)
```

Now I pick up and explain about our Distributed TensorFlow example.

First, we start `tf.train.Server` in each nodes to communicate each other with gRPC protocol. As you can see below, each server is having corresponding each role (later we set `job_name` and `task_index` using command line options), and here I assume one parameter node (10.0.0.6) and two worker nodes (10.0.0.4 and 10.0.0.5), in which 10.0.0.4 is master (chief) node.

```
def main(_):
  with tf.device(tf.train.replica_device_setter(
    worker_device="/job:%s/task:%d" % (FLAGS.job_name, FLAGS.task_index),
    cluster=cluster)):
    ...

if __name__ == '__main__':
  ...

  # start server
  cluster = tf.train.ClusterSpec({
    'ps': ['10.0.0.6:2222'],
    'worker': [
      '10.0.0.4:2222',
      '10.0.0.5:2222'
    ]})
  server = tf.train.Server(
    cluster,
    job_name=FLAGS.job_name,
    task_index=FLAGS.task_index)
  if FLAGS.job_name == "ps":
    server.join()
  elif FLAGS.job_name == "worker":
    is_chief = (FLAGS.task_index == 0)
    tf.app.run(main=main, argv=[sys.argv[0]] + unparsed)
```

In this example, we're assigning each one task (role) for each node (computing machine), but you can also assign one task for each devices (CPUs or GPUs) on the single machine as follows.

```
...
worker_device = "/job:%s/task:%d/gpu:%d" % (FLAGS.job_name, FLAGS.task_index, 0
with tf.device(tf.train.replica_device_setter(
  worker_device=worker_device,
  ps_device="/job:ps/cpu:0",
  cluster=cluster)):
  ...
```

In this example, we set 2 as `num_epochs` for reading data (train.tfrecords). Therefore, after data is read (iterated) twice by cycle, `OutOfRangeError` occurs and session (`MonitoredTrainingSession`) is correctly closed.

```
...
train_queue = tf.train.string_input_producer(
  [FLAGS.train_file],
  num_epochs = 2)
...
```

After the graph is constructed, we run the distributed session using `tf.train.MonitoredTrainingSession`. This session monitors each tasks in each nodes and it's the core component for distributed computation. (You can also use `tf.train.Supervisor` for distributed running, but `tf.train.MonitoredTrainingSession` is recommended for the current version.)
In the session we read `train_batch_image` and `train_batch_label` (each 100 rows of features and labels) sequentially and run `train_op` and estimate `loss` .

I note that the [original sample (https://github.com/tsmatsuz/tensorflow-mnist-batch-read-and-train-tutorial/blob/master/mnist_tf.py)](https://github.com/tsmatsuz/tensorflow-mnist-batch-read-and-train-tutorial/blob/master/mnist_tf.py) (non-distributed one) manually started queue runner (`QueueRunner`) for batch-reading. But in our example, `tf.train.MonitoredSession` launches queue thread in the background, and we don't need to start queue runner explicitly.

```
...
with tf.train.MonitoredTrainingSession(
  master=server.target,
  checkpoint_dir=FLAGS.out_dir,
  hooks=hooks,
  is_chief=is_chief) as sess:

  step = 0
  array_image, array_label = sess.run(
    [train_batch_image, train_batch_label])
  while not sess.should_stop():
    feed_dict = {
      plchd_image: array_image,
      plchd_label: array_label
    }
    _, loss_value, array_image, array_label = sess.run(
      [train_op, loss, train_batch_image, train_batch_label],
      feed_dict=feed_dict)
    step += 1
    if step % 100 == 0:
      print("Worker: Step %d (Loss: %.2f)" % (step, loss_value))
...
```

In asynchronous approach, you can also implement to detect the completion of each nodes automatically with enqueue/dequeue mechanism and you can proceed the consequent steps as follows.

Run and Check Our Model

Now let's start our program with the following command.
As I mentioned earlier, here we're assuming one parameter server (10.0.0.6) and two worker serves (10.0.0.4 and 10.0.0.5).

Parameter Node (10.0.0.6)

```
python mnist_tf_dist.py \
  --job_name=ps \
  --task_index=0
```

Worker Node 1 (10.0.0.4)

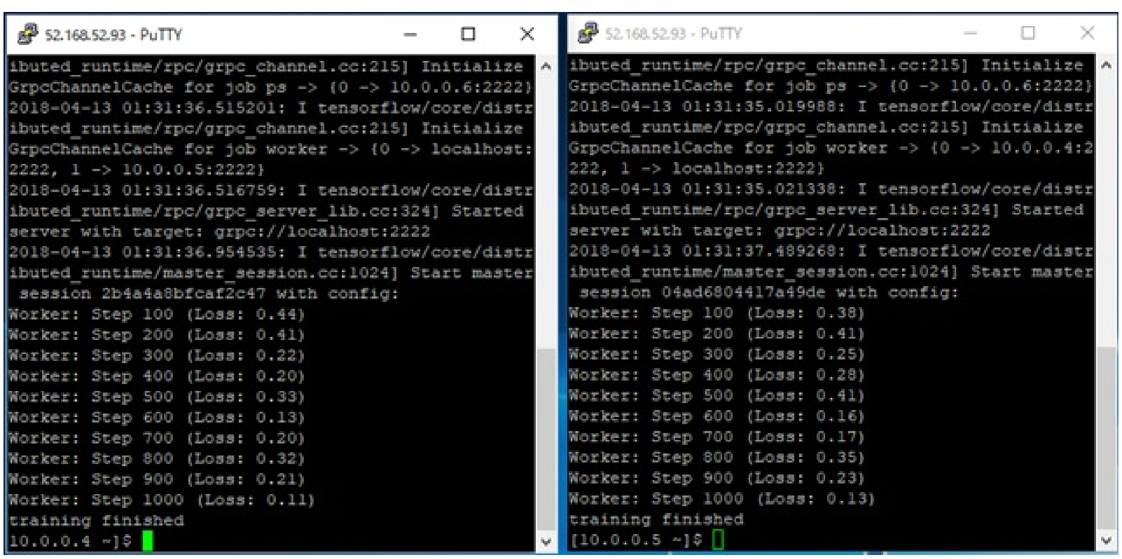
```
python mnist_tf_dist.py \
  --job_name=worker \
  --task_index=0
```

Worker Node 2 (10.0.0.5)

```
python mnist_tf_dist.py \
  --job_name=worker \
  --task_index=1
```

Once the parameter server is started and all workers enter in `MonitoredTrainingSession`, the training (the loop of `train_op`) starts in multiple workers and each nodes respectively (independently) read data on corresponding node.
If you use the shared NFS storage or cloud storage, all nodes can read the same data respectively. (You don't need to deploy data in multiple nodes respectively.)

Output (2 workers – 10.0.0.4 and 10.0.0.5)



(https://netweblog.files.wordpress.com/2018/04/run_output1.jpg)

As you can see in our source code, we're setting `checkpoint_dir` in session. By this setting, the checkpoint data (tensor objects, variables, etc) are all saved in the directory and you can restore and restart your training later again. You can use gathered checkpoint data as you need. (retrain, test, etc) For example, when you copy these checkpoint data in the specific local folder, you can test the model's accuracy using the following local (non-distributed) python code. (Note that the meta is saved in master node, but the index and data is saved in parameter node separately. You must gather these data in one location for running the following code.)

```
from __future__ import absolute_import
from __future__ import division
from __future__ import print_function

import sys
import argparse
import math

import tensorflow as tf

FLAGS = None
batch_size = 100

def main(_):
  #
  # define graph (to be restored !)
  #

  # define input
  plchd_image = tf.placeholder(
    dtype=tf.float32,
    shape=(batch_size, 784))
  plchd_label = tf.placeholder(
    dtype=tf.int32,
    shape=(batch_size))

  # define network and inference
  with tf.name_scope('hidden1'):
    weights = tf.Variable(
      tf.truncated_normal(
        [784, 128],
        stddev=1.0 / math.sqrt(float(784))),
      name='weights')
    biases = tf.Variable(
      tf.zeros([128]),
      name='biases')
    hidden1 = tf.nn.relu(tf.matmul(plchd_image, weights) + biases)
  with tf.name_scope('hidden2'):
    weights = tf.Variable(
      tf.truncated_normal(
        [128, 64],
        stddev=1.0 / math.sqrt(float(128))),
      name='weights')
    biases = tf.Variable(
      tf.zeros([64]),
      name='biases')
    hidden2 = tf.nn.relu(tf.matmul(hidden1, weights) + biases)
  with tf.name_scope('softmax_linear'):
```

```
      weights = tf.Variable(
        tf.truncated_normal(
          [64, 10],
          stddev=1.0 / math.sqrt(float(64))),
      name='weights')
      biases = tf.Variable(
        tf.zeros([10]),
        name='biases')
      logits = tf.matmul(hidden2, weights) + biases

    #
    # Restore and Testing
    #

    ckpt = tf.train.get_checkpoint_state(FLAGS.out_dir)
    idex = int(ckpt.model_checkpoint_path.split('/')[-1].split('-')[-1])

    saver = tf.train.Saver()

    with tf.Session() as sess:
      # restore graph
      saver.restore(sess, ckpt.model_checkpoint_path)
      graph = tf.get_default_graph()

      # add to graph - read test data
      test_queue = tf.train.string_input_producer(
        [FLAGS.test_file],
        num_epochs = 1) # when data is over, it raises OutOfRange
      test_reader = tf.TFRecordReader()
      _, test_serialized_exam = test_reader.read(test_queue)
      test_exam = tf.parse_single_example(
        test_serialized_exam,
        features={
          'image_raw': tf.FixedLenFeature([], tf.string),
          'label': tf.FixedLenFeature([], tf.int64)
        })
      test_image = tf.decode_raw(test_exam['image_raw'], tf.uint8)
      test_image.set_shape([784])
      test_image = tf.cast(test_image, tf.float32) * (1. / 255)
      test_label = tf.cast(test_exam['label'], tf.int32)
      test_batch_image, test_batch_label = tf.train.batch(
        [test_image, test_label],
        batch_size=batch_size)

      # add to graph - test (evaluate) graph
      array_correct = tf.nn.in_top_k(logits, plchd_label, 1)
      test_op = tf.reduce_sum(tf.cast(array_correct, tf.int32))

      # run
      sess.run(tf.initialize_local_variables())
      coord = tf.train.Coordinator()
      threads = tf.train.start_queue_runners(sess=sess, coord=coord) # for dat
      num_test = 0
      num_true = 0
      array_image, array_label = sess.run(
        [test_batch_image, test_batch_label])
      try:
```

```
      while True:
        feed_dict = {
          plchd_image: array_image,
          plchd_label: array_label
        }
        batch_num_true, array_image, array_label = sess.run(
          [test_op, test_batch_image, test_batch_label],
          feed_dict=feed_dict)
        num_true += batch_num_true
        num_test += batch_size
    except tf.errors.OutOfRangeError:
      print('Scoring done !')
    precision = float(num_true) / num_test
    print('Accuracy: %0.04f (Num of samples: %d)' %
      (precision, num_test))

if __name__ == '__main__':
  parser = argparse.ArgumentParser()
  parser.add_argument(
    '--test_file',
    type=str,
    default='/home/demouser/test.tfrecords',
    help='File path for the test data.')
  parser.add_argument(
    '--out_dir',
    type=str,
    default='/home/demouser/out',
    help='Dir path for the model and checkpoint output.')
  FLAGS, unparsed = parser.parse_known_args()

  tf.app.run(main=main, argv=[sys.argv[0]] + unparsed)
```

Next I'll show Distributed TensorFlow using Estimator or Experiment. (Recent samples are using these high-level APIs.)