|  |
| --- |
|  |
|  | import functools  import json |
|  | import os |
|  | import tensorflow as tf |
|  |  |
|  | from object\_detection import trainer |
|  | from object\_detection.builders import dataset\_builder |
|  | from object\_detection.builders import model\_builder |
|  | from object\_detection.utils import config\_util |
|  | from object\_detection.utils import dataset\_util |
|  |  |
|  | tf.logging.set\_verbosity(tf.logging.INFO) |
|  |  |
|  | flags = tf.app.flags |
|  | flags.DEFINE\_string('master', '', 'Name of the TensorFlow master to use.') |
|  | flags.DEFINE\_integer('task', 0, 'task id') |
|  | flags.DEFINE\_integer('num\_clones', 1, 'Number of clones to deploy per worker.') |
|  | flags.DEFINE\_boolean('clone\_on\_cpu', False, |
|  | 'Force clones to be deployed on CPU. Note that even if ' |
|  | 'set to False (allowing ops to run on gpu), some ops may ' |
|  | 'still be run on the CPU if they have no GPU kernel.') |
|  | flags.DEFINE\_integer('worker\_replicas', 1, 'Number of worker+trainer ' |
|  | 'replicas.') |
|  | flags.DEFINE\_integer('ps\_tasks', 0, |
|  | 'Number of parameter server tasks. If None, does not use ' |
|  | 'a parameter server.') |
|  | flags.DEFINE\_string('train\_dir', '', |
|  | 'Directory to save the checkpoints and training summaries.') |
|  |  |
|  | flags.DEFINE\_string('pipeline\_config\_path', '', |
|  | 'Path to a pipeline\_pb2.TrainEvalPipelineConfig config ' |
|  | 'file. If provided, other configs are ignored') |
|  |  |
|  | flags.DEFINE\_string('train\_config\_path', '', |
|  | 'Path to a train\_pb2.TrainConfig config file.') |
|  | flags.DEFINE\_string('input\_config\_path', '', |
|  | 'Path to an input\_reader\_pb2.InputReader config file.') |
|  | flags.DEFINE\_string('model\_config\_path', '', |
|  | 'Path to a model\_pb2.DetectionModel config file.') |
|  |  |
|  | FLAGS = flags.FLAGS |
|  |  |
|  |  |
|  | def main(\_): |
|  | assert FLAGS.train\_dir, '`train\_dir` is missing.' |
|  | if FLAGS.task == 0: tf.gfile.MakeDirs(FLAGS.train\_dir) |
|  | if FLAGS.pipeline\_config\_path: |
|  | configs = config\_util.get\_configs\_from\_pipeline\_file( |
|  | FLAGS.pipeline\_config\_path) |
|  | if FLAGS.task == 0: |
|  | tf.gfile.Copy(FLAGS.pipeline\_config\_path, |
|  | os.path.join(FLAGS.train\_dir, 'pipeline.config'), |
|  | overwrite=True) |
|  | else: |
|  | configs = config\_util.get\_configs\_from\_multiple\_files( |
|  | model\_config\_path=FLAGS.model\_config\_path, |
|  | train\_config\_path=FLAGS.train\_config\_path, |
|  | train\_input\_config\_path=FLAGS.input\_config\_path) |
|  | if FLAGS.task == 0: |
|  | for name, config in [('model.config', FLAGS.model\_config\_path), |
|  | ('train.config', FLAGS.train\_config\_path), |
|  | ('input.config', FLAGS.input\_config\_path)]: |
|  | tf.gfile.Copy(config, os.path.join(FLAGS.train\_dir, name), |
|  | overwrite=True) |
|  |  |
|  | model\_config = configs['model'] |
|  | train\_config = configs['train\_config'] |
|  | input\_config = configs['train\_input\_config'] |
|  |  |
|  | model\_fn = functools.partial( |
|  | model\_builder.build, |
|  | model\_config=model\_config, |
|  | is\_training=True) |
|  |  |
|  | def get\_next(config): |
|  | return dataset\_util.make\_initializable\_iterator( |
|  | dataset\_builder.build( |
|  | config, num\_workers=FLAGS.worker\_replicas, |
|  | worker\_index=FLAGS.task)).get\_next() |
|  |  |
|  | create\_input\_dict\_fn = functools.partial(get\_next, input\_config) |
|  |  |
|  | env = json.loads(os.environ.get('TF\_CONFIG', '{}')) |
|  | cluster\_data = env.get('cluster', None) |
|  | cluster = tf.train.ClusterSpec(cluster\_data) if cluster\_data else None |
|  | task\_data = env.get('task', None) or {'type': 'master', 'index': 0} |
|  | task\_info = type('TaskSpec', (object,), task\_data) |
|  |  |
|  | # Parameters for a single worker. |
|  | ps\_tasks = 0 |
|  | worker\_replicas = 1 |
|  | worker\_job\_name = 'lonely\_worker' |
|  | task = 0 |
|  | is\_chief = True |
|  | master = '' |
|  |  |
|  | if cluster\_data and 'worker' in cluster\_data: |
|  | # Number of total worker replicas include "worker"s and the "master". |
|  | worker\_replicas = len(cluster\_data['worker']) + 1 |
|  | if cluster\_data and 'ps' in cluster\_data: |
|  | ps\_tasks = len(cluster\_data['ps']) |
|  |  |
|  | if worker\_replicas > 1 and ps\_tasks < 1: |
|  | raise ValueError('At least 1 ps task is needed for distributed training.') |
|  |  |
|  | if worker\_replicas >= 1 and ps\_tasks > 0: |
|  | # Set up distributed training. |
|  | server = tf.train.Server(tf.train.ClusterSpec(cluster), protocol='grpc', |
|  | job\_name=task\_info.type, |
|  | task\_index=task\_info.index) |
|  | if task\_info.type == 'ps': |
|  | server.join() |
|  | return |
|  |  |
|  | worker\_job\_name = '%s/task:%d' % (task\_info.type, task\_info.index) |
|  | task = task\_info.index |
|  | is\_chief = (task\_info.type == 'master') |
|  | master = server.target |
|  |  |
|  | trainer.train(create\_input\_dict\_fn, model\_fn, train\_config, master, task, |
|  | FLAGS.num\_clones, worker\_replicas, FLAGS.clone\_on\_cpu, ps\_tasks, |
|  | worker\_job\_name, is\_chief, FLAGS.train\_dir) |
|  |  |
|  |  |
|  | if \_\_name\_\_ == '\_\_main\_\_': |
|  | tf.app.run() |