Introduction to Distributed TensorFlow

Reedbush Hackathon @ The University of Tokyo

2019-01-24

目次

データ並列

データ並列の基本的な考え方

正則化項 + 損失の最小化問題:

$$\min_{x} \lambda R(w) + L(w) \text{ where } L(w) = \sum_{i} \ell_i(w)$$

損失項の勾配はデータごとに並列で計算できる

$$w_{k+1} = w_k - \alpha \left[\nabla_w R(w_k) + \sum_{i} \nabla_w \ell_i(w_k) \right]$$

Parameter Server 方式での役割分担

Parameter Server

- 最新の weight w を保持して必要なときに master や worker に共有

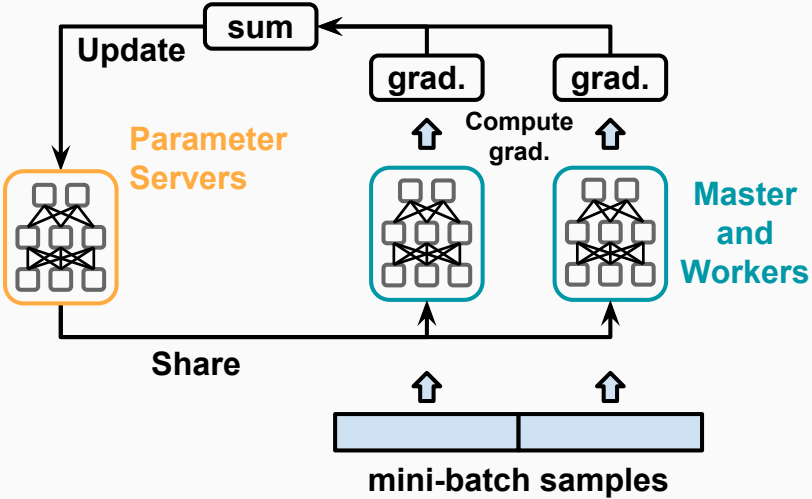
Worker

- それぞれ異なるデータを使って勾配を計算

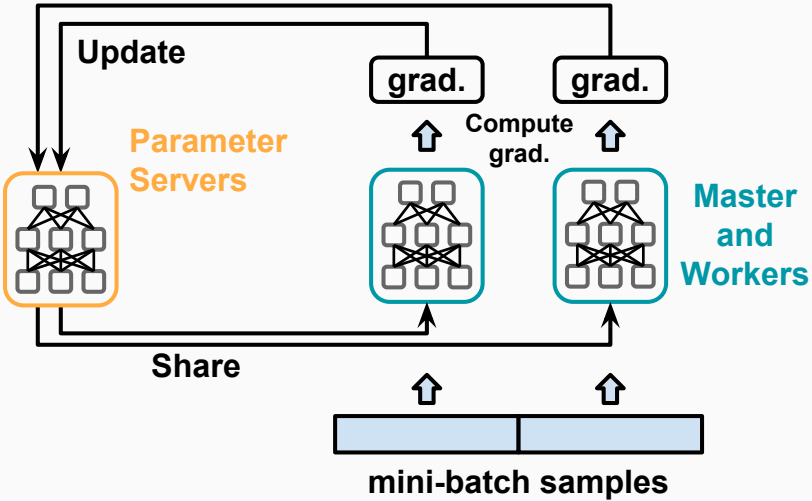
Master

- Worker とだいたい同じ
- モデルの保存や初期化など実装上必要な一部の処理

データ並列 + 同期型 + Parameter Server



データ並列 + 非同期型 + Parameter Server



それぞれの長所

同期型

- 収束性が良く学習が上手くいきやすい
  - Stale gradients の問題が発生しない

非同期型

- 対故障性
  - Worker が落ちても影響が小さく学習が継続可能
- 待ち合わせが無いので高スループット
  - 遅い worker に足を引っ張られない

All Reduce での役割分担

Worker

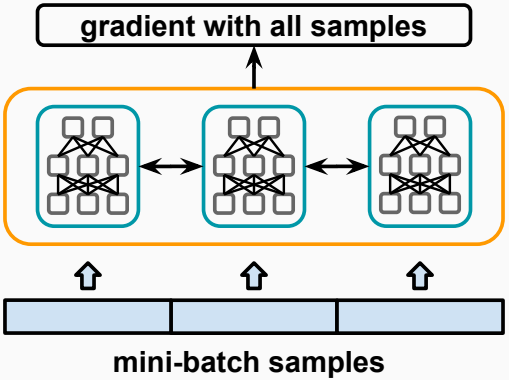
- それぞれ異なるデータを使って勾配を計算
- Parameter server は無いので各 worker が weight w のコピーを保持

Master

- Worker とだいたい同じ
- モデルの保存や初期化など実装上必要な一部の処理

データ並列 + 同期型 + All Reduce

- Weight w は各 worker がコピーを持つ
- Worker 同士で通信して勾配の情報を交換
- それぞれ w を更新する



モデル並列

モデル並列の考え方

- データ単位ではなくモデルそのもの (ニューラルネットで言えばネットワークの部分) で並列化できる部分を分割して worker に割り当てる
- まだ発展途上で定石が定まっている状態ではないので創意工夫を凝らしてやっていく形になる
- Mesh TensorFlow という専用のライブラリも開発されているが事例などはまだほとんど無い段階
  - https://github.com/tensorflow/mesh

TensorFlow での実装

TensorFlow で分散学習を実装する時の選択肢

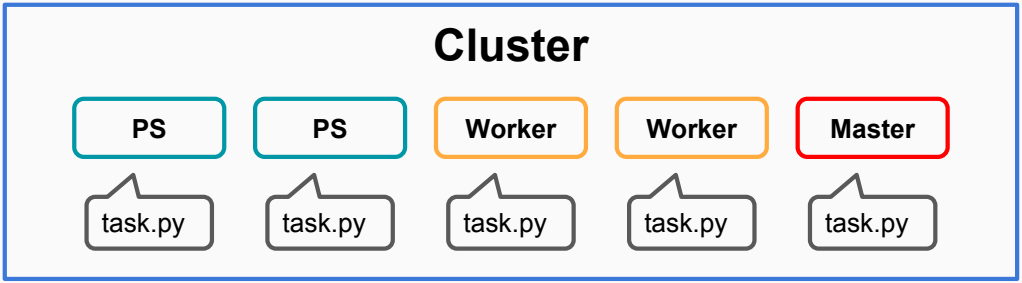
TensorFlow の標準の機能

- TensorFlow の operation を任意のデバイスに配置する形で実装
- 自由度が高く頑張れば大抵の構成は作れる
- データ並列のよくある構成は高レベル API で分散処理を意識せずにやってもらうこともできる

Horovod

- Uber が開発した OSS
- データ並列 + All Reduce の形式が前提
- Reedbush ではこちらの方が実績もあるのでおそらく良い

TensorFlow の標準機能で実装するときのお作法



- クラスタの全ノードは同じスクリプトを実行
- スクリプト内で master, worker, ps の分岐を記述
  - 自分がどれなのかという情報は環境変数に入れておく

環境変数に入れておく情報の例

```
{
  "cluster": {
    "ps": ["ps-408715a125-0:2222", "ps-408715a125-1:2222"],
    "worker": ["worker-408715a125-0:2222", "worker-408715a125-1:2222"],
    "master": ["master-408715a125-0:2222"]
  },
  "task": {
    "index": 0,
    "type": "master",
    ...
  },
  ...
}
```

TensorFlow 側の処理

```
tf_conf = json.loads(os.environ.get("TF_CONFIG", "{}"))
# TF_CONFIG からクラスタ構成の情報を取り出して使う
server = tf.train.Server(
    tf_conf.get("cluster", None), job_name=tf_conf["task"]["type"],
    task_index=tf_conf["task"]["index"]
)
# TF_CONFIG から自分の役割が Master、Worker、PS のどれか調べて分岐させる
if tf_conf["task"]["type"] == "ps":
    server.join()  # Parameter Server はこの処理をするだけ
else:
    # MasterとWorkerの処理を実行する
    ...
```

高レベル APIs (1/2)

自力で書き切ることもできるが、高レベル API を使うことで分散処理部分を含め面倒な部分を自動でやってもらえる仕組みが存在する

Estimator

- 分散学習の機能は最も充実している
- Horovod との併用も可能

Keras

- Keras 自体に分散学習の機能は無いので Horovod を使う

高レベル APIs (2/2)

tf.keras

- TensorFlow に特化して独自機能が追加された Keras
- まだ標準機能で multi-node の分散学習には対応していない
  - https://github.com/tensorflow/tensorflow/issues/23664
  - tf.keras のモデルを Estimator に変換する機能があるので現状はそれを使用する形になる
- Horovod とも最近併用可能に
  - https://github.com/uber/horovod/pull/513
- 将来的には Estimator より tf.keras が主流になる可能性もある

Estimator + データ並列

- Estimator に DistributeStrategy を渡す
- クラスタ構成は tf.estimator.RunConfig のコンストラクタが裏で勝手に環境変数から読み込む

```
distribution = tf.contrib.distribute.MirroredStrategy(num_gpus=2)
run_config = tf.estimator.RunConfig(distribute=distribution)
clf = tf.estimator.DNNClassifier(
   feature_columns=[tf.feature_column.numeric_column("x", shape=[4]),
   hidden_units=[10, 20, 10], n_classes=3,
   model_dir="/tmp/iris_model",
   config=run_config
)
clf.train(...)
```

DistributeStrategy (1/2)

tf.distribute.MirroredStrategy

- https://www.tensorflow.org/guide/distribute_strateg
- single-node で multi-GPUs の環境で使う All Reduce
- tf.keras にも対応している
- multi-node でも動くらしいが推奨はされていない
  - https://github.com/tensorflow/tensorflow/issues/23664

DistributeStrategy (2/2)

tf.contrib.distribute.CollectiveAllReduceStrategy

- multi-node で multi-GPUs の環境で使う All Reduce

tf.contrib.distribute.ParameterServerStrategy

- multi-node で multi-GPUs の環境で使う prameter server 方式

補足: DistributeStrategy 以前の書き方

- tf.estimator.train_and_evaluate を使うと parameter server 方式で学習が走る

```
run_config = tf.estimator.RunConfig(distribute=distribution)
clf = tf.estimator.DNNClassifier(
   feature_columns=[tf.feature_column.numeric_column("x", shape=[4]),
   hidden_units=[10, 20, 10], n_classes=3,
   model_dir="/tmp/iris_model",
   config=run_config
)
...
tf.estimator.train_and_evaluate(clf, ...)
```

TensorFlow と Horovod でデータ並列 + All Reduce (1/2)

```
import tensorflow as tf
import horovod.tensorflow as hvd

hvd.init()

# 1 プロセスに 1 GPUを割り当てて他は見えないようにしておく
config = tf.ConfigProto()
config.gpu_options.visible_device_list = str(hvd.local_rank())
...
# Optimizer を分散学習用のクラスで wrap する
optimizer = hvd.DistributedOptimizer(optimizer)
...
```

TensorFlow と Horovod でデータ並列 + All Reduce (2/2)

```
# 初期値を master（rank 0）から他のプロセスに broadcast する hook
hooks = [hvd.BroadcastGlobalVariablesHook(0)]

# モデルの保存は master だけがやれば良い
model_dir = "/your/model/dir" if hvd.rank() == 0 else None
```

- あとは Estimator か tf.train.MonitoredTrainingSession を使って
  いつも通りコードを書けば良い
- 実行するときは mpirun で実行
- Epoch 数、学習率などをプロセス数に応じて調整することが多い
  - Epoch 数はプロセス数に反比例、学習率は比例させるなど

Keras と Horovod でデータ並列 + All Reduce (2/2)

```
import horovod.keras as hvd
import keras
import tensorflow as tf

hvd.init()

# 1 プロセスに 1 GPU を割り当てて他では見えないようにしておく
config = tf.ConfigProto()
config.gpu_options.visible_device_list = str(hvd.local_rank())
K.set_session(tf.Session(config=config))
```

Keras と Horovod でデータ並列 + All Reduce (1/2)

```
# 初期値を master から他のプロセスに broadcast する callback
callbacks = [hvd.callbacks.BroadcastGlobalVariablesCallback(0)]

# master にだけモデル保存の callback を追加
if hvd.rank() == 0:
    callbacks.append(
        keras.callbacks.ModelCheckpoint('./checkpoint-{epoch}.h5')
    )
```

- あとは Keras でいつも通りのコードを書いて mpirun で実行
- Epoch 数、学習率などをプロセス数に応じて調整することが多い
  - Epoch 数はプロセス数に反比例、学習率は比例させるなど

補足: TensorFlow Eager Mode + Horovod

- つい最近 eager mode への対応 PR がマージされた
  - https://github.com/uber/horovod/pull/670
  - TensorFlow v2.0 からは eager mode がデフォルトになるので重要な機能になるかも?

おわりに

- 今はちょうど分散学習周りが変わっている最中なので方法が色々ある
- Reedbush 上でやるなら MPI が前提になっているので
  おそらく Horovod がやりやすい
- モデル並列など凝ったことをやる場合は TensorFlow 標準の機能を
  上手く使う必要がある