Data Parallel Deep Learning with Tensorflow and Keras

Georg Zitzlsberger, IT4Innovations

29.06.2023

Univerza v Ljubljani

TECHNISCHE UNIVERSITÄT WIEN

CINECA

VSB TECHNICAL UNIVERSITY OF OSTRAVA | IT4INNOVATIONS NATIONAL SUPERCOMPUTING CENTER

Co-funded by the Erasmus+ Programme of the European Union

This project has been funded with support from the European Commission.
This publication [communication] reflects the views only of the author, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

Tensorflow and Keras

Tensorflow and Keras

Georg Zitzlsberger, IT4Innovations

- Tensorflow 2.0 introduced end of 2019, including Keras
- Latest version: 2.12.0 (March 2023)
- APIs for C++, Java and **Python**
- The "biggest" community, but also lot's of changes
- Applicable to a wide range of user types:
  - Developer
  - Researcher
  - Industry or academia
- Enhanced versions are available from different vendors (via PIP):
  - Intel CPUs intel-tensorflow
  - AMD CPUs tensorflow-rocm
  - NVIDIA tensorflow-gpu

TensorFlow

Keras

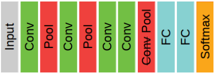
Georg Zitzlsberger, IT4Innovations

- Keras offers two ways to build a model:
  - Sequential model with `tf.keras.Sequential`
  - Functional API with `tf.keras.Model`
- Most used operations/layers already exist in the Keras API, e.g.:
  - `tf.keras.layers.Conv2D` or `tf.keras.layers.Conv3D`
  - `tf.keras.layers.Dense`
  - `tf.keras.layers.LSTM`
  - ...
- The models expect data in the following formats (`channels_last`):
  - `[batch, spatial_dims…, channels]`, e.g. 2D images:
    `[10, 256, 256, 3]` (10 per batch, 256x256 images with 3 color channels)
  - `[batch, time_step, spatial_dims…, channels]`, e.g. time series of images:
    `[10, 25, 64, 32, 1]` (10 per batch, series of 25 64x32 images with 1 color channel)

Building a Sequential Model

Georg Zitzlsberger, IT4Innovations

- Very easy with least amount of code
- Only sequential models, no forks/joins!
- Example:




```python
from keras import Sequential
from keras.layers import Conv2D, MaxPooling2D,
  BatchNormalization, ZeroPadding2D, Dropout,
  Activation, Flatten, Dense

def alexnet(n_classes=5):
    model = Sequential()
    model.add(Conv2D(64, 11, strides=4))
    model.add(ZeroPadding2D(2))
    model.add(Activation('relu'))
    model.add(MaxPooling2D(pool_size=3, strides=2))
    model.add(Conv2D(192, 5))
    model.add(ZeroPadding2D(2))
    model.add(Activation('relu'))
    model.add(MaxPooling2D(pool_size=3, strides=2))
    model.add(Conv2D(384, 3))
    model.add(ZeroPadding2D(1))
    model.add(Activation('relu'))
    model.add(Conv2D(256, 3))
    model.add(ZeroPadding2D(1))
    model.add(Activation('relu'))
    model.add(MaxPooling2D(pool_size=3, strides=2))

    model.add(Flatten())
    model.add(Dropout(0.5))
    model.add(Dense(4096,
                    input_shape=(6 * 6 * 256, )))
    model.add(Activation('relu'))
    model.add(Dropout(0.5))
    model.add(Dense(4096))
    model.add(Activation('relu'))
    model.add(Dense(n_classes))
    model.add(Activation('softmax'))
    return model

if __name__ == '__main__':
    amodel = alexnet(10)
    amodel.summary()
```

- Requires definition of input and model
- Allows forks/joins (not shown here)
- Example:

```python
from keras import Sequential
from keras.layers import Conv2D, MaxPooling2D,
  BatchNormalization, ZeroPadding2D, Dropout,
  Activation, Flatten, Dense

def alexnet(n_classes=5):
    inp = tf.keras.Input(shape=[256,256,3],
                         dtype=tf.float32)

    conv1 = Conv2D(64, 11, strides=4))(inp)
    pad1 = ZeroPadding2D(2)(conv1)
    act1 = Activation('relu')(pad1)
    pool1 = MaxPooling2D(pool_size=3,
                         strides=2)(act1)
    conv2 = Conv2D(192, 5)(pool1)
    pad2 = ZeroPadding2D(2)(conv2)
    act2 = Activation('relu')(pad2)
    pool2 = MaxPooling2D(pool_size=3,
                         strides=2)(act2)
    conv3 = Conv2D(384, 3)(pool2)
    pad3 = ZeroPadding2D(1)(conv3)
    act3 = Activation('relu')(pad3)
    conv4 = Conv2D(256, 3)(act3)
    pad4 = ZeroPadding2D(1)(conv4)
    act4 = Activation('relu')(pad4)
    pool4 = MaxPooling2D(pool_size=3,
                         strides=2)(act4)
    flat = Flatten()(pool4)
    drop1 = Dropout(0.5)(flat)
    dense1 = Dense(4096,
                   input_shape=(6 * 6 * 256, ))(drop1)
    act5 = Activation('relu')(dense1)
    drop2 = Dropout(0.5)(act5)
    dense2 = Dense(4096)(drop2)
    act6 = Activation('relu')(dense2)
    dense3 = Dense(n_classes)(act6)
    act7 = Activation('softmax')(dense3)

    return tf.keras.Model(inp, act7)

if __name__ == '__main__':
    amodel = alexnet(10)
    amodel.summary()
```

Our Example Model

Georg Zitzlsberger, IT4Innovations

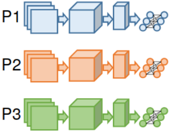
In the demonstration later we use a simple model with the following layers (from input to output):

- Flatten the 2D input via
  `tf.keras.layers.Flatten`
- Dense hidden layer with 128 neurons/units and Rectified Linear Unit (ReLU) activation via
  `tf.keras.layers.Dense`
- Dense layer as output with 10 neurons/units and softmax activation via
  `tf.keras.layers.Dense`

Parallelism

Difference Data vs. Model Parallelism

Georg Zitzlsberger, IT4Innovations



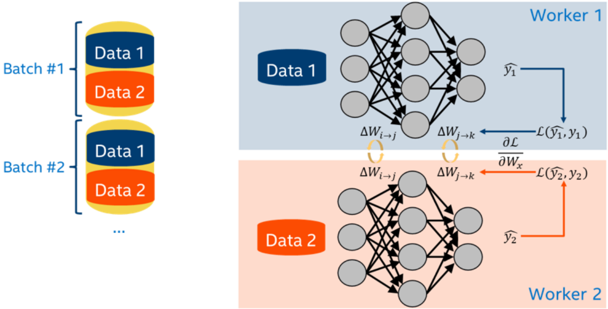
- Network layers assigned to different workers
- Every worker trains with the same data
- Activations are exchanged (requires large I/O bandwidth)
- **Enables bigger models**

- All workers see the same network
- Every worker trians with different data
- Gradints (weights) are exchanged (averaging to common model)
- Side effect: "sharp" minima
- **Enables faster training**

(Images: Ben-Nun, et al.)

Data Parallel Distributed Training

Georg Zitzlsberger, IT4Innovations

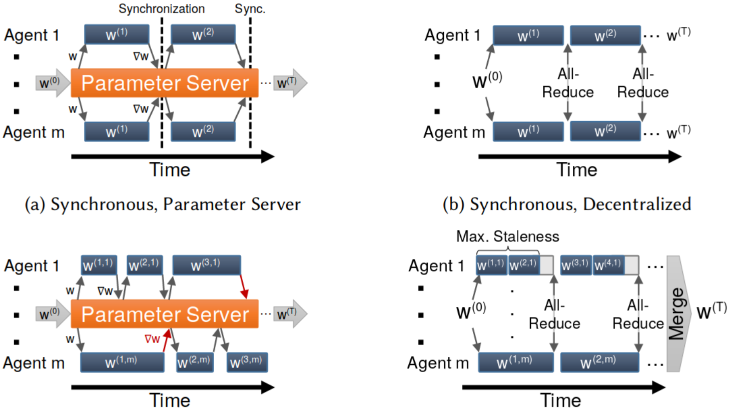


- Batch size limits parallelism
- Scaling batch size requires scaling of learning rate (linearly)

Distributed Training: Model Consistency

Georg Zitzlsberger, IT4Innovations

SCtrain | SUPERCOMPUTING KNOWLEDGE PARTNERSHIP



(a) Synchronous, Parameter Server

(b) Synchronous, Decentralized

(c) Asynchronous, Parameter Server

(d) Stale-Synchronous, Decentralized

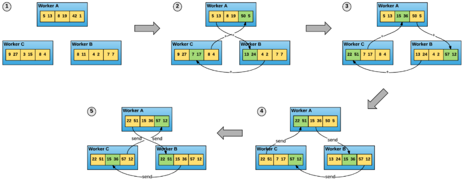
(Image: Ben-Nun, et al.)

Horovod

Horovod

Georg Zitzlsberger, IT4Innovations

- Developed by Uber Engineering
- Part of Michelangelo
  (Uber's Machine Learning Platform)
- Aimed at and demonstrated for large scale
- Uses MPI based collective communication
  (synchronous & decentralized)
- Only small code modifications needed
- Supports the most common frameworks:
  - Tensorflow (1.x & 2.0) + Keras
  - Pytorch
  - MXNet



(Images: Uber)

- Horovod comes with a wrapper horovodrun, e.g.:

  ```
  $ horovodrun -np 4 -H server1:2,server2:2 python train.py
  ```

- Different back-ends are possible: MPI, Gloo, NCCL, oneCCL, etc.

- Intel MPI or OpenMPI can be used:

  ```
  $ mpirun -n 4 -ppn 2 -hosts server1,server2 python train.py
  ```

- Add the following:
  - `hvd.init()`:
    Initializes Horovod (and MPI underneath)
  - `hvd.callbacks.BroadcastGlobalVariablesCallback(0)`:
    Initialize model to start with same copies
  - `hvd.DistributedOptimizer(…)`:
    Wrapper around standard optimizer (SGD, Adam, etc.) to enable distributed weight/gradient updates

- **Note: The same script is executed on all workers!**
  Only let first rank do the I/O (e.g. print to stdout or save snapshots)

- Full documentation can be found here

What needs attention:

- If `tf.data.Dataset` is used, consider `shard(num_shards, index)`, e.g.:
  `my_dataset.shard(hvd.size(), hvd.rank())`
- If training steps are used, instead of number of epochs, adjust the steps, e.g.:
  `training_steps /= hvd.size()` (assuming perfectly balanced training data)
- If training data size is large, avoid loading it at every worker and divide accross workers
  The same script is executed on all workers!
- Scale the learning rate linearly with the number of workers, e.g.:
  `lr *= hvd.size()`
  See Alex Krizhevsky's paper:
  Strictly speaking it should be `lr *= sqrt(hvd.size())`

Native Data Parallel Training

Georg Zitzlsberger, IT4Innovations

SCtrain | SUPERCOMPUTING KNOWLEDGE PARTNERSHIP

Native support of (sync.) data parallel training is also available:

- Tensorflow:
  `tf.distribute.Strategy` with different strategies (MirroredStrategy, TPUStrategy, MultiWorkerMirroredStrategy, CentralStorageStrategy, ParameterServerStrategy)
  Documentation
- PyTorch:
  `torch.distributed` with three backends (GLOO, MPI, NCCL)
  Documentation
  Use `torch.nn.parallel.DistributedDataParallel` (DDP) with **NCCL** backend for multi-node and multi-GPU support.

- Tensorflow/Horovod:
  - Data parallelization is done in optimizer
  - Decomposition of data is done with `tf.data.Dataset.shard`
- PyTorch:
  - Data parallelization is done in model:

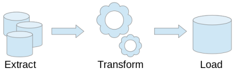
```
import os
import torch.distributed as dist
from torch.nn.parallel import DistributedDataParallel as DDP
...
num_gpus = int(os.environ['OMPI_COMM_WORLD_SIZE'])
rank = int(os.environ['OMPI_COMM_WORLD_RANK'])

dist.init_process_group("nccl", rank=rank, world_size=num_gpus)

model = Model().to(rank) # Move to GPU
ddp_model = DDP(model, device_ids=[rank])
```

  - Data decomposition with `torch.utils.data.distributed.DistributedSampler`, e.g.:

```
from torch.utils.data import DataLoader
from torch.utils.data.distributed import DistributedSampler
...
sampler = DistributedSampler(dataset, num_replicas=num_gpus, rank=rank)
loader = DataLoader(dataset, sampler=sampler)
```

Data Pipeline

- Extract, Transform and Load (ETL) pipeline via `tf.data.Dataset`
- Provides a wide range of functionality to process training/validation data:
  - I/O: files, NumPy, TFRecord/Protocol Buffers, Pandas Data Frames, etc.
  - Split training/validation: Provide a ratio how much of the dataset should be for training.
  - Batch and pad: Build minibatches and pad to ensure balance.
  - Shuffle: Randomize the samples with every training epoch.
  - Cache and Pre-fetch: Optimize access to data.
  - Map and filter: Convert the data to a format needed for training/validation and also filter samples.
  - ...



- The hands-on uses the following training pipeline:
  - Input MNIST training dataset `ds_train` and apply `normalize_img` with `tf.data.experimental.AUTOTUNE` parallel calls, via ▸ `tf.data.Dataset.map`
  - Give every worker/GPU/process an own shard with ▸ `tf.data.Dataset.shard`
  - Cache the data (no repeated normalization/sharding) with ▸ `tf.data.Dataset.cache`
  - Shuffle data entirely (size of full shard) with ▸ `tf.data.Dataset.shuffle`
  - Batch with a batch size of 32 with ▸ `tf.data.Dataset.batch`
  - Prefetch the next elements ▸ `tf.data.Dataset.prefetch` (for the buffer size, use `tf.data.experimental.AUTOTUNE`)
- The validation pipeline `ds_test`, does the same **except** shuffling

See the ▸ Tensorflow Dataset Documentation for more information

Train and Visualize with Tensorboard

- Recap - we have:
  - Data pipelines provide training/validation data
  - Model to train
- Select the loss function, e.g.:

```
loss = tf.keras.losses.BinaryCrossentropy()
```

- Optionally, select metrics, e.g.:

```
metric1 = tf.keras.metrics.MeanAbsoluteError()
```

- Select the optimizer to use, e.g.:

```
opt = tf.keras.optimizers.SGD(lr=.001, momentum=0.8)
```

- Compile the model:

```
amodel.compile(optimizer=opt, loss=loss, metrics=[metric1])
```

Train and Visualize with Tensorboard

Georg Zitzlsberger, IT4Innovations

- For Tensorboard, define a callback, e.g.:

```
tensorboard_cb = tf.keras.callbacks.TensorBoard(log_dir="./logs",
                                                histogram_freq=1,
                                                update_freq='batch')
```

- Snapshots needed?

```
save_best_cb = tf.keras.callbacks.ModelCheckpoint(
                                    filepath="./best_weights.hdf5"
                                    monitor='val_loss',
                                    save_best_only=True)
```

- Train...

```
amodel.fit(
    training_ds,
    validation_data = validation_ds,
    epochs = 100,
    callbacks = [save_best_cb, tensorboard_cb],
    verbose=2)
```

Train and Visualize with Tensorboard

Georg Zitzlsberger, IT4Innovations

Train with visualization in Tensorboard using the hands-on:

- Find the epoch loss in Tensorboard in tab *SCALARS*
- Compare the loss with the metric (here: *accuracy*)
- Inspect the model graph in tab *GRAPHS*
- See the change of parameters during training in tab *HISTOGRAMS*

Note: Don't forget to set a reload interval in settings (or reload manually)!

Tensorflow Dataset Recommendations

Tensorflow Dataset Recommendations

Georg Zitzlsberger, IT4Innovations

SCtrain | SUPERCOMPUTING KNOWLEDGE PARTNERSHIP

- Some methods offer multi-threading; try `tf.data.experimental.AUTOTUNE`, e.g.:

```
train_ds = tf.data.Dataset.from_tensor_slices(my_data)
                      .map(my_prepare_func, num_parallel_calls=AUTO))
```

- Caching keeps everything in memory - be carefull where to place it in the pipeline!
- Caching can also be used to use fast NVM/SSD storage, e.g.:

```
train_ds.cache(filename="/mnt/nvmeof/train_ds_{}".format(hvd.rank()))
```

- Use `tf.data.Dataset.map` before `tf.data.Dataset.batch` if map is expensive, vice versa otherwise
- Prefetch at the end of the pipeline

See Tensorflow's ▸ Better performance with the tf.data API

Caching @ IT4Innovations

Georg Zitzlsberger, IT4Innovations

SCtrain | SUPERCOMPUTING KNOWLEDGE PARTNERSHIP

- The *SCRATCH* filesystem is used for projects
- Reloading training/validation data from *SCRATCH* is not efficient:
  - No guaranteed I/O bandwidth
  - Hogging of resources
- Solution: Cache dataset pipelines using
  - ▸ Ramdisk (global with `qsub …-l global_ramdisk=true`)
  - ▸ NVMeoF (Non-Volatile Memory express over Fabric)
- Barbora cluster:
  - Ramdisk with 180GB of 192GB per node on `/mnt/global_ramdisk/`
  - NVMeoF shared with `qsub …-l nvmeof=1TB:shared` on `/mnt/nvmeof/` (max. 10TB)
- Karolina cluster:
  - Ramdisk with approx. 1TB per node (`qnvidia`) on `/mnt/global_ramdisk/`
  - *SCRATCH* can be used for larger sizes (uses SSDs, 730.9 GB/s write, 1198.3 GB/s read)

Hands-on of Multi-node/-GPU Examples using Tensorflow

SCtrain | SUPERCOMPUTING KNOWLEDGE PARTNERSHIP

Hands-on contains:

- Simple Multilayer Perceptron (MLP) model
- Use `tf.data.Dataset` to ingest MNIST data set
- Extend it with Horovod for data parallel training (on multiple GPUs)

Singularity

Singularity

Georg Zitzlsberger, IT4Innovations

- Container system for HPC
- Convert a Docker container to a Singularity image:
  docker2singularity
- Example:

```
$ module load Singularity/3.8.0
$ module load CUDA/11.0.2-GCC-9.3.0
$ singularity exec --nv -B
/scratch/project/open-21-31:/work ↙
     my_container.sif jupyter lab --port 8888
```

- Get ready-to-use images from the NVIDIA GPU Cloud

Thank you for your attention!