Deep Learning on Supercomputers

An introduction

Fahad Khalid [ f.khalid@fz-juelich.de ]. Simulation Laboratory Neuroscience, Jülich Supercomputing Centre

November 29, 2019 I Introduction to the Programming and Usage of Supercomputing resources at Jülich
Agenda

   i. Supervised learning
   ii. Artificial Neural Networks
   iii. Error backpropagation

2. Code examples: Training with one GPU
   i. Handwritten digit recognition
   ii. The MNIST dataset
   iii. Implementation with tf.keras

3. Concepts: Distributed training
   i. Why use distributed training?
   ii. Model parallelism
   iii. Data parallelism
   iv. Gradient aggregation

4. Pit stop: Parallel computing
   i. Shared memory vs. Distributed memory
   ii. The hpc4neuro Python library

5. Back to distributed training: Examples
   i. MNIST classification: Epoch distributed
   ii. Handling custom data
   iii. Distributing training data

6. The “Deep Learning on Supercomputers” tutorial
   i. Introduction
   ii. Running code samples on JUWELS
   iii. Job configuration

7. Conclusion
Agenda

   i. Supervised learning
   ii. Artificial Neural Networks
   iii. Error backpropagation

2. Code examples: Training with one GPU
   i. Handwritten digit recognition
   ii. The MNIST dataset
   iii. Implementation with tf.keras

3. Concepts: Distributed training
   i. Why use distributed training?
   ii. Model parallelism
   iii. Data parallelism
   iv. Gradient aggregation

4. Pit stop: Parallel computing
   i. Shared memory vs. Distributed memory
   ii. The hpc4neuro Python library

5. Back to distributed training: Examples
   i. MNIST classification: Epoch distributed
   ii. Handling custom data
   iii. Distributing training data

6. The “Deep Learning on Supercomputers” tutorial
   i. Introduction
   ii. Running code samples on JUWELS
   iii. Job configuration

7. Conclusion
Concepts

Supervised learning (1)

<table>
<thead>
<tr>
<th>$x_1$</th>
<th>$x_2$</th>
<th>$y$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

$0 \rightarrow \square \rightarrow 1$

$x_1 \rightarrow \square \rightarrow y$

$\square \rightarrow x_1$

$\square \rightarrow x_2$
Concepts

Artificial Neural Networks

\[ w_0 = 0.1 \quad b = -1.5 \quad \varphi(w \bar{x} + b) \]

\[ w_1 = 1.0 \]

\[ x_0 \quad x_1 \]

\[ y \]
Error backpropagation (1)

Instance \( \{(x_0, x_1), \dot{y}\} \)

\[
\begin{align*}
\dot{y} - y &= -0.5 \\
\varphi(.) &= \phi(x, \varphi(.)) \\
\varphi(.) &= \varphi(y) \\
x_0 &\rightarrow 0.1 \\
x_1 &\rightarrow -0.5 \\
&\rightarrow 1.0 \\
&\rightarrow -1.0 \\
&\rightarrow 0.2 \\
&\rightarrow -0.5
\end{align*}
\]
Concepts

Error backpropagation (2)

Instance $\{(x_0, x_1), \hat{y}\}$

Error $\hat{y} - y$
Concepts

Instance 
\{(x_0, x_1), \hat{y}\}

Error backpropagation (3)

\[ x_0 \rightarrow 1.0 \rightarrow -1.5 \rightarrow \varphi(.) \rightarrow \hat{y} \]

\[ x_1 \rightarrow 1.0 \rightarrow -0.5 \rightarrow \varphi(.) \rightarrow \hat{y} \]

\[ \hat{y} - y \]

Error
   i. Supervised learning
   ii. Artificial Neural Networks
   iii. Error backpropagation

2. Code examples: Training with one GPU
   i. Handwritten digit recognition
   ii. The MNIST dataset
   iii. Implementation with tf.keras

3. Concepts: Distributed training
   i. Why use distributed training?
   ii. Model parallelism
   iii. Data parallelism
   iv. Gradient aggregation

4. Pit stop: Parallel computing
   i. Shared memory vs. Distributed memory
   ii. The hpc4neuro Python library

5. Back to distributed training: Examples
   i. MNIST classification: Epoch distributed
   ii. Handling custom data
   iii. Distributing training data

6. The “Deep Learning on Supercomputers” tutorial
   i. Introduction
   ii. Running code samples on JUWELS
   iii. Job configuration

7. Conclusion
Examples

Handwritten digit recognition

$f(image)$
Examples

Handwritten digit recognition

Artificial Neural Network

7
Examples

Modified National Institute of Standards and Technology (MNIST) database

- Image dimensions: 28×28
- Each pixel $p \in [0, 255]$
- 60,000 training examples
- 10,000 test examples

Source: Modified version of this. License.
Examples

A basic network for classification

Image

Flatten

weights

512

weights

Output

Member of the Helmholtz Association

29 November 2019

Page 13
Examples

MNIST classification with tf.keras: Code

1. `import` tensorflow as tf
2. mnist = tf.keras.datasets.mnist
3. (x_train, y_train), (x_test, y_test) = mnist.load_data()
4. x_train, x_test = x_train / 255.0, x_test / 255.0
5. model = tf.keras.models.Sequential([tf.keras.layers.Flatten(),
                                           tf.keras.layers.Dense(512, activation=tf.nn.relu),
                                           tf.keras.layers.Dense(10, activation=tf.nn.softmax)]
                                       )
6. optimizer = tf.keras.optimizers.Adam()
7. epochs = 4
8. model.compile(
        optimizer=optimizer,
        loss='sparse_categorical_crossentropy',
        metrics=['accuracy']
    
)
9. model.fit(
      x=x_train,
      y=y_train,
      batch_size=32,
      epochs=epochs
    )

dl_on_supercomputers/course_material/examples/mnist_single_gpu.py
Examples

MNIST classification with tf.keras: Output

$ python -u mnist_single_gpu.py

Epoch 1/4
60000/60000 [==============================] - 8s 131us/sample - loss: 0.2006 - acc: 0.9404
Epoch 2/4
60000/60000 [==============================] - 7s 120us/sample - loss: 0.0815 - acc: 0.9749
Epoch 3/4
60000/60000 [==============================] - 7s 120us/sample - loss: 0.0548 - acc: 0.9831
Epoch 4/4
60000/60000 [==============================] - 7s 119us/sample - loss: 0.0376 - acc: 0.9879

Test loss: 0.06436453427168308
Test accuracy: 0.9805

1. score = model.evaluate(x=x_test, y=y_test, verbose=0)
2. print(f'Test loss: {score[0]}')
3. print(f'Test accuracy: {score[1]}')
Agenda

   i. Supervised learning
   ii. Artificial Neural Networks
   iii. Error backpropagation

2. Code examples: Training with one GPU
   i. Handwritten digit recognition
   ii. The MNIST dataset
   iii. Implementation with tf.keras

3. Concepts: Distributed training
   i. Why use distributed training?
   ii. Model parallelism
   iii. Data parallelism
   iv. Gradient aggregation

4. Pit stop: Parallel computing
   i. Shared memory vs. Distributed memory
   ii. The hpc4neuro Python library

5. Back to distributed training: Examples
   i. MNIST classification: Epoch distributed
   ii. Handling custom data
   iii. Distributing training data

6. The “Deep Learning on Supercomputers” tutorial
   i. Introduction
   ii. Running code samples on JUWELS
   iii. Job configuration

7. Conclusion
Distributed training

Motivation

1. Train faster
   i. Large dataset size
      a. Distribute epochs
      b. Distribute training/validation data
   ii. Compute intensive model

2. Increase the effective batch size

3. Use a dataset with very large instances

4. Use a very large model
Distributed training

Model Parallel

• Other methods
  • Layer Pipelining
  • Hybrid Parallelism

Distributed training

Data Parallel

Distributed training

Gradient averaging in data parallel training

<table>
<thead>
<tr>
<th>Initialization</th>
<th>$G_0$</th>
<th>$G_1$</th>
<th>...</th>
<th>$G_{n-1}$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td></td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Iteration 1</th>
<th>$G_0$</th>
<th>$G_1$</th>
<th>...</th>
<th>$G_{n-1}$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0.1</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td></td>
<td>0.0</td>
<td>0.2</td>
<td>0.9</td>
<td>0.7</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Updated Model</th>
<th>$G_0$</th>
<th>$G_1$</th>
<th>...</th>
<th>$G_{n-1}$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0.1</td>
<td>0.4</td>
<td>0.1</td>
<td>0.1</td>
</tr>
<tr>
<td></td>
<td>0.3</td>
<td>0.5</td>
<td>0.3</td>
<td>0.3</td>
</tr>
</tbody>
</table>

Average gradients across all GPUs
Agenda

   i. Supervised learning
   ii. Artificial Neural Networks
   iii. Error backpropagation

2. Code examples: Training with one GPU
   i. Handwritten digit recognition
   ii. The MNIST dataset
   iii. Implementation with tf.keras

3. Concepts: Distributed training
   i. Why use distributed training?
   ii. Model parallelism
   iii. Data parallelism
   iv. Gradient aggregation

4. Pit stop: Parallel computing
   i. Shared memory vs. Distributed memory
   ii. The hpc4neuro Python library

5. Back to distributed training: Examples
   i. MNIST classification: Epoch distributed
   ii. Handling custom data
   iii. Distributing training data

6. The “Deep Learning on Supercomputers” tutorial
   i. Introduction
   ii. Running code samples on JUWELS
   iii. Job configuration

7. Conclusion
Parallel computing: Very briefly

Shared memory vs. Distributed memory

Threading: OpenMP, Intel TBB, etc.

Message passing: MPI, NCCL, etc.
import os

def get_filenames(path):
    return os.listdir(path)

designation = get_filenames('.

print(f'Filenames: {filenames}')

python -m hpc4neuro.examples.distribution.sequential_filenames

Filenames: ['errors.py', '__init__.py', 'utils', 'tutorials', '__pycache__']

mpirun -np 4 python -m hpc4neuro.examples.distribution.static_filenames_decorator

0 -- Filenames: ['errors.py', '__pycache__']
1 -- Filenames: ['__init__.py']
2 -- Filenames: ['utils']
3 -- Filenames: ['tutorials']

hpc4neuro/examples/sequential_filenames.py

hpc4neuro/examples/static_filenames_decorator.py
```python
1. import os
2. filenames = os.listdir('.')
3. print(f'Filenames: {filenames}')
```

```python
1. import os
2. from mpi4py import MPI
3. from hpc4neuro.distribution import DataDistributor
4. dist_decorator = DataDistributor(MPI.COMM_WORLD, shutdown_on_error=True)
5. get_rank_local_filenames = dist_decorator(os.listdir)
6. filenames = get_rank_local_filenames('.')
7. print(f'{MPI.COMM_WORLD.Get_rank()} -- Filenames: {filenames}')
```

hpc4neuro/examples/dynamic_filenames_decorator.py
Agenda

   i. Supervised learning
   ii. Artificial Neural Networks
   iii. Error backpropagation

2. Code examples: Training with one GPU
   i. Handwritten digit recognition
   ii. The MNIST dataset
   iii. Implementation with tf.keras

3. Concepts: Distributed training
   i. Why use distributed training?
   ii. Model parallelism
   iii. Data parallelism
   iv. Gradient aggregation

4. Pit stop: Parallel computing
   i. Shared memory vs. Distributed memory
   ii. The hpc4neuro Python library

5. Back to distributed training: Examples
   i. MNIST classification: Epoch distributed
   ii. Handling custom data
   iii. Distributing training data

6. The “Deep Learning on Supercomputers” tutorial
   i. Introduction
   ii. Running code samples on JUWELS
   iii. Job configuration

7. Conclusion
Back to distributed training: Examples

Distributed training with Horovod: Code (1)

1. `import` tensorflow as `tf`
2. `import` math
3. `import` horovod.tensorflow.keras as `hvd`
4. `from` tensorflow.python.keras `import` backend as `K`
5. `hvd.init()`
6. `config = tf.ConfigProto()`  
7. `config.gpu_options.visible_device_list = str(hvd.local_rank())`
8. `K.set_session(tf.Session(config=config))`

Single GPU

Distributed

`dl_on_supercomputers/course_material/examples/mnist_epoch_distributed.py`
Back to distributed training: Examples

Distributed training with Horovod: Code (2)

2. `mnist = tf.keras.datasets.mnist`
3. `(x_train, y_train), (x_test, y_test) = mnist.load_data()`
4. `x_train, x_test = x_train / 255.0, x_test / 255.0`
5. `model = tf.keras.models.Sequential([`  
   `tf.keras.layers.Flatten(),  
   tf.keras.layers.Dense(512, activation=tf.nn.relu),  
   tf.keras.layers.Dense(10, activation=tf.nn.softmax)  
])`
6. `optimizer = tf.keras.optimizers.Adam()`
7. `epochs = 4`

9. `mnist = tf.keras.datasets.mnist`
10. `(x_train, y_train), (x_test, y_test) = mnist.load_data()`
11. `x_train, x_test = x_train / 255.0, x_test / 255.0`
12. `model = tf.keras.models.Sequential([`  
    `tf.keras.layers.Flatten(),  
    tf.keras.layers.Dense(512, activation=tf.nn.relu),  
    tf.keras.layers.Dense(10, activation=tf.nn.softmax)  
])`
13. `optimizer = tf.keras.optimizers.Adam()`
14. `optimizer = hvd.DistributedOptimizer(optimizer)`
15. `epochs = int(math.ceil(4.0 / hvd.size()))`
Back to distributed training: Examples

Distributed training with Horovod: Code (3)

8. model.compile(
    optimizer=optimizer,
    loss='sparse_categorical_crossentropy',
    metrics=['accuracy']
)

9. model.fit(
    x=x_train,
    y=y_train,
    batch_size=32,
    epochs=epochs
)

16. model.compile(
    optimizer=optimizer,
    loss='sparse_categorical_crossentropy',
    metrics=['accuracy']
)

17. callbacks = [hvd.callbacks.BroadcastGlobalVariablesCallback(0)]

18. model.fit(
    x=x_train,
    y=y_train,
    batch_size=32,
    epochs=epochs,
    callbacks=callbacks
)

Single GPU

Distributed
Back to distributed training: Examples

Distributed training with Horovod: Output

$ mpirun -np 1 python -u mnist_epoch_distributed.py

Epoch 1/4
60000/60000 [==============================] - 8s 125us/sample - loss: 0.2004 - acc: 0.9410

Epoch 2/4
60000/60000 [==============================] - 7s 121us/sample - loss: 0.0786 - acc: 0.9763

Epoch 3/4
60000/60000 [==============================] - 7s 121us/sample - loss: 0.0519 - acc: 0.9836

Epoch 4/4
60000/60000 [==============================] - 7s 121us/sample - loss: 0.0374 - acc: 0.9879

Test loss: 0.0761936013394734
Test accuracy: 0.9773
Training with custom data I/O: Code snippet

```python
1. def get_filenames(path):
2.     absolute_path = os.path.join(os.path.abspath(f'{path}/x'))
3.     return os.listdir(absolute_path)
4. 
5. def main():
6.     data_dir = 'data/mnist/partitioned'
7.     train_filenames = get_filenames(f'{data_dir}/train')
8.     test_filenames = get_filenames(f'{data_dir}/test')
9.     x_train, y_train = load_dataset(f'{data_dir}/train', train_filenames)
10.    x_test, y_test = load_dataset(f'{data_dir}/test', test_filenames)
11.    x_train, x_test = x_train / 255.0, x_test / 255.0
```
Back to distributed training: Examples

Input data distribution for Horovod: Code snippet

```python
1. def main():
2.     initialize_hvd_and_mpi()
3.     is_root = hvd.rank() == 0
4. 
5.     dist_decorator = DataDistributor(mpi_comm=mpi4py.MPI.COMM_WORLD, shutdown_on_error=True)
6.     get_rank_local_filenames = dist_decorator(get_filenames)
7. 
8.     data_dir = 'data/mnist/partitioned'
9.     train_filenames = get_rank_local_filenames(f'{data_dir}/train')
10.    x_train, y_train = load_dataset(f'{data_dir}/train', train_filenames)
11.    x_train = x_train / 255.0
12.    if is_root:
13.        test_filenames = get_filenames(f'{data_dir}/test')
14.        x_test, y_test = load_dataset(f'{data_dir}/test', test_filenames)
15.        x_test = x_test / 255.0
16.    else:
17.        x_test, y_test = None, None
```

```python
1. def get_filenames(path):
2.     absolute_path = os.path.join(os.path.abspath(f'{path}/x'))
3.     return os.listdir(absolute_path)
```
   i. Supervised learning
   ii. Artificial Neural Networks
   iii. Error backpropagation
2. Code examples: Training with one GPU
   i. Handwritten digit recognition
   ii. The MNIST dataset
   iii. Implementation with tf.keras
3. Concepts: Distributed training
   i. Why use distributed training?
   ii. Model parallelism
   iii. Data parallelism
   iv. Gradient aggregation
4. Pit stop: Parallel computing
   i. Shared memory vs. Distributed memory
   ii. The hpc4neuro Python library
5. Back to distributed training: Examples
   i. MNIST classification: Epoch distributed
   ii. Handling custom data
   iii. Distributing training data
6. The “Deep Learning on Supercomputers” tutorial
   i. Introduction
   ii. Running code samples on JUWELS
   iii. Job configuration
7. Conclusion
Getting started with Deep learning on Supercomputers

The tutorial

https://gitlab.version.fz-juelich.de/hpc4ns/dl_on_supercomputers
Getting started with Deep learning on Supercomputers

The tutorial: Logging in to JUWELS

```
{ ~ } » ssh khalid1@juwels.fz-juelich.de
Last login: Tue Nov 26 11:27:09 2019
******************************************************************************
* Welcome to                                                                 *
*       _ _   ___        _______ _     ____                                    *
*      | | | | |
*      / / ____| |   / ___|     Juelich Wizard                *
*    _  | | | | |
*    / /|  _| | |
*   ___| |___| |___ ___) |      European Leadership *
*  | |___| |___| |___ |      Science                               *
*                                                                
******************************************************************************
2019-03-11T12:00+0200
### Known Issues ###
An up-to-date list of known issues on the system is maintained at

https://apps.fz-juelich.de/jsc/hps/juwels/known-issues.html
******************************************************************************
[khalid1@juwels04 ~]$
Getting started with Deep learning on Supercomputers

The tutorial: Environment setup and Repository cloning

```
[khalid1@juwels04 ~]$ jutil env activate -p cslns -A slns
[khalid1@juwels04 cslns]$ cd $PROJECT/khalid1/juwels
[khalid1@juwels04 juwels]$ module load git-lfs
[khalid1@juwels04 juwels]$ git lfs install
Git LFS initialized.
[khalid1@juwels04 juwels]$ git lfs clone https://gitlab.version.fz-juelich.de/hpc4ns/dl_on_supercomputers.git
Cloning into 'dl_on_supercomputers'...
remote: Enumerating objects: 142, done.
remote: Counting objects: 100% (142/142), done.
remote: Compressing objects: 100% (110/110), done.
remote: Total 434 (delta 46), reused 110 (delta 31)
Receiving objects: 100% (434/434), 103.63 KiB | 0 bytes/s, done.
Resolving deltas: 100% (222/222), done.
[khalid1@juwels04 juwels]$ 0% (43/43), 193 MB | 29 MB/s
[khalid1@juwels04 juwels]$`
```
Getting started with Deep learning on Supercomputers

The tutorial: Starting and monitoring the training job

[khalid1@juwels04 juwels]$ cd dl_on_supercomputers/horovod/keras
[khalid1@juwels04 keras]$ sbatch submit_job_juwels.sh
Submitted batch job 1885056
[khalid1@juwels04 keras]$ squeue -u khalid1

<table>
<thead>
<tr>
<th>JOBID</th>
<th>PARTITION</th>
<th>NAME</th>
<th>USER</th>
<th>ST</th>
<th>TIME</th>
<th>NODES</th>
<th>NODELIST(REASON)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1885056</td>
<td>develgpus</td>
<td>HOROVOD_khalid1</td>
<td>CF</td>
<td>0:07</td>
<td>2</td>
<td>jwc09n[006,009]</td>
<td></td>
</tr>
</tbody>
</table>

[khalid1@juwels04 keras]$ tail -f output_1885056.out
Using /p/project/cslns/khalid1/juwels/dl_on_supercomputers/datasets as the data directory.
x_train shape: (60000, 28, 28, 1)
60000 train samples
10000 test samples
Train on 60000 samples, validate on 10000 samples
Epoch 1/12
128/60000 [..............................] - ETA: 44:55 - loss: 2.3016 - acc: 0.1016
...
60000/60000 [==================================] - 3s 42us/step - loss: 0.0273 - acc: 0.9914
Test loss: 0.02779148665768025
Test accuracy: 0.9911
Getting started with Deep learning on Supercomputers

The tutorial: Job script submit_job_juwels.sh

```
# Slurm job configuration
#SBATCH --nodes=2
#SBATCH --ntasks=8
#SBATCH --ntasks-per-node=4
#SBATCH --output=output_%j.out
#SBATCH --error=error_%j.err
#SBATCH --time=00:10:00
#SBATCH --job-name=HOROVOD_KERAS_MNIST
#SBATCH --gres=gpu:4 --partition=develgpus
#SBATCH --mail-type=ALL
#SBATCH --mail-user=<...>

# Load the required modules
module load GCC/8.3.0
module load TensorFlow/1.13.1-GPU-Python-3.6.8
module load Keras/2.2.4-GPU-Python-3.6.8

# Run the program
srun python -u mnist.py
```
   i. Supervised learning
   ii. Artificial Neural Networks
   iii. Error backpropagation

2. Code examples: Training with one GPU
   i. Handwritten digit recognition
   ii. The MNIST dataset
   iii. Implementation with tf.keras

3. Concepts: Distributed training
   i. Why use distributed training?
   ii. Model parallelism
   iii. Data parallelism
   iv. Gradient aggregation

4. Pit stop: Parallel computing
   i. Shared memory vs. Distributed memory
   ii. The hpc4neuro Python library

5. Back to distributed training: Examples
   i. MNIST classification: Epoch distributed
   ii. Handling custom data
   iii. Distributing training data

6. The “Deep Learning on Supercomputers” tutorial
   i. Introduction
   ii. Running code samples on JUWELS
   iii. Job configuration

7. Conclusion
Conclusion

Summary

Key points
• There is more to distributed training than speedup, e.g., effective batch size can be increased
• The supercomputers can be utilized for data parallel training with relative ease
• Tensorflow and Horovod are already available on JUWELS and JURECA
• Combining the material presented here with the “DL on Supercomputers” tutorial is a good place to start
• A good foundation in parallel programming, especially with MPI, can go a long way

Support
• All SC related issues: sc@fz-juelich.de
• Tutorial and hpc4neuro: slns@fz-juelich.de

Useful resources
• On the next slide

Thank you!
Appendix

Useful links

1. Getting started with Deep Learning on Supercomputers
   • https://gitlab.version.fz-juelich.de/hpc4ns/dl_on_supercomputers#getting-started-with-deep-learning-on-supercomputers

2. The hpc4neuro Python library
   • https://gitlab.version.fz-juelich.de/hpc4ns/hpc4neuro#the-hpc4neuro-library-of-python-utilities

3. Horovod
   • https://github.com/horovod/horovod

4. The MPI tutorial
   • https://mpitutorial.com/

5. MPI4Py