An Introduction
Speaker: Fahad Khalid [ f.khalid@fz-juelich.de ]

May 21, 2019  |  Simulation Laboratory Neuroscience, Jülich Supercomputing Center
Agenda

• Concepts
  • The supervised classification learning problem
  • Artificial Neural Networks
  • Error backpropagation
• Code examples with Keras and Tensorflow
  • Handwritten digit classification and MNIST
  • Distributed training for MNIST
• The how and why of distributed training
• “The Deep Learning on Supercomputers” tutorial
**Agenda**

- Concepts
  - The supervised classification learning problem
  - Artificial Neural Networks
  - Error backpropagation
- Code examples with Keras and Tensorflow
  - Handwritten digit classification and MNIST
  - Distributed training for MNIST
- The how and why of distributed training
- “The Deep Learning on Supercomputers” tutorial
Supervised classification learning

\[ x_0, y_0 \\
\vdots \\
\vdots \\
\vdots \\
\vdots \\
x_n,0 \quad x_n,1 \quad y_n \]
Concepts

Error backpropagation (1)

\[ x_0 \]
\[ w_0 = 0.1 \quad b = -1.5 \quad \varphi(w_1 x_1 + b) \]
\[ w_1 = 1.0 \]

\[ x_1 \]

\[ y \]
Concepts

Error backpropagation (2)

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

Error \( \hat{y} - y \)

\[ x_0 \rightarrow 0.1 \rightarrow -1.5 \rightarrow \varphi(\cdot) \rightarrow -1.0 \rightarrow 1.0 \rightarrow -0.5 \rightarrow y \]

\[ x_1 \rightarrow 0.2 \rightarrow -0.5 \rightarrow \varphi(\cdot) \rightarrow -0.5 \rightarrow 1.0 \rightarrow -1.0 \rightarrow 1.0 \rightarrow -0.5 \rightarrow \]

\( (x_0, x_1) \) and \( \hat{y} \) are the instance, and the error is \( \hat{y} - y \). The nodes represent the values calculated during the backpropagation process.
Instance \((x_0, x_1), \hat{y}\)

Error backpropagation (3)

\[
\begin{align*}
\hat{y} &= y - y \\

\phi(\cdot) &= -1.5 \\
1.0 &= 1.0 \\
-0.5 &= -0.5 \\
0.3 &= 0.3 \\
0.5 &= 0.5
\end{align*}
\]
Error backpropagation (4)

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

Error \( \hat{y} - y \)

\[
\begin{align*}
\phi(.), & \\
0.5 & \rightarrow 1.0 & \rightarrow \phi(.) & \rightarrow 1.0 & \rightarrow -1.5 & \rightarrow \phi(.) & \rightarrow -1.5 \\
& \rightarrow 1.0 & \rightarrow -2.0 & \rightarrow 1.0 & \rightarrow -0.5 & \rightarrow \phi(.) & \rightarrow y
\end{align*}
\]
Agenda

• Concepts
  • The supervised classification learning problem
  • Artificial Neural Networks
  • Error backpropagation

• Code examples with Keras and Tensorflow
  • Handwritten digit classification and MNIST
  • Distributed training for MNIST

• The how and why of distributed training
• “The Deep Learning on Supercomputers” tutorial
Examples

Handwritten character recognition

\[ f(image) \]
Examples

Handwritten character recognition

Artificial Neural Network
Examples

Modified National Institute of Standards and Technology (MNIST) database

- Image dimensions: $28 \times 28$
- 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

Flattened Image

weights

512

weights

Output
Examples

MNIST classification with Keras and Tensorflow: Code

```python
import tensorflow as tf

mnist = tf.keras.datasets.mnist

(x_train, y_train), (x_test, y_test) = mnist.load_data()

x_train, x_test = x_train / 255.0, x_test / 255.0

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)
])

optimizer = tf.keras.optimizers.Adam()

epochs = 4

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

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

model.evaluate(x=x_test, y=y_test)
```
Examples

MNIST classification with Keras and Tensorflow: Output

```
$ python -u mnist_simplest.py
Epoch 1/4
60000/60000 [==================================] - 8s 129us/sample - loss: 0.2041 - acc: 0.9395
Epoch 2/4
60000/60000 [==================================] - 8s 125us/sample - loss: 0.0823 - acc: 0.9745
Epoch 3/4
60000/60000 [==================================] - 8s 125us/sample - loss: 0.0539 - acc: 0.9831
Epoch 4/4
60000/60000 [==================================] - 8s 125us/sample - loss: 0.0379 - acc: 0.9877
Test loss: 0.06125587912490591
Test accuracy: 0.9817
```
Examples

Distributed training with Horovod: Code (1)

```python
import math
import tensorflow as tf
import horovod.tensorflow.keras as hvd
from tensorflow.python.keras import backend as K

mnist = tf.keras.datasets.mnist

(x_train, y_train), (x_test, y_test) = mnist.load_data()

x_train, x_test = x_train / 255.0, x_test / 255.0

cfg = tf.ConfigProto()
cfg.gpu_options.visible_device_list = str(hvd.local_rank())
K.set_session(tf.Session(config=cfg))
```
Examples

Distributed training with Horovod: Code (2)

```python
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)
])

optimizer = tf.keras.optimizers.Adam()

optimizer = hvd.DistributedOptimizer(optimizer)

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

Distributed training with Horovod: Code (3)

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

epochs = int(math.ceil(4.0 / hvd.size()))

model.fit(
    x=x_train,
    y=y_train,
    batch_size=32,
    epochs=epochs,
    verbose=1,
    callbacks=callbacks
)

model.evaluate(x=x_test, y=y_test)
Examples

Distributed training with Horovod: Output

```
$ mpirun -np 1 python -u mnist_simplest_horovod.py
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
Epoch 1/4
60000/60000-------------------------------] - 4s 68us/sample - loss: 0.1997 - acc: 0.9414
Epoch 2/4
60000/60000-------------------------------] - 4s 65us/sample - loss: 0.0800 - acc: 0.9754
Epoch 3/4
60000/60000-------------------------------] - 4s 64us/sample - loss: 0.0526 - acc: 0.9833
Epoch 4/4
60000/60000-------------------------------] - 4s 64us/sample - loss: 0.0343 - acc: 0.9888
Test loss: 0.06574100296042161
Test accuracy: 0.9789
```
Distributed training

The broad categories

• Why use distributed training at all?
  • Speedup the training process
    • compute intensive models, large datasets
• Data parallel vs. Model parallel
• How to benefit from the data parallel distribution method?
  1. Reduce the number of epochs
  2. Reduce the number of training examples per rank
  3. Increase the effective batch size
Agenda

• Concepts
  • The supervised classification learning problem
  • Artificial Neural Networks
  • Error backpropagation
• Code examples with Keras and Tensorflow
  • Handwritten digit classification and MNIST
  • Distributed training for MNIST
• The how and why of distributed training
• “The Deep Learning on Supercomputers” tutorial
Tutorial

What is it about?

- Showcases well-known deep learning frameworks
  - Keras
  - Tensorflow
  - PyTorch
  - Horovod
  - Caffe
- Describes how to use these on the supercomputers at the JSC
  - Environment setup, code samples, data …

  gitlab.version.fz-juelich.de/khalid1/ml_dl_on_supercomputers
Tutorial

Getting started with ML/DL on Supercomputers

This repository is intended to serve as a tutorial for anyone interested in utilizing the supercomputers available at the JSC for ML/DL related projects. It is assumed that the reader is proficient in one or more of the following frameworks:

- Tensorflow
- Keras
- PyTorch
- Caffe
- Horovod

Note: This tutorial is by no means intended as an introduction to ML/DL, or to any of the above mentioned frameworks. If you are interested in educational resources for beginners, please visit this page.

Note: This tutorial does not support JUWELS at the moment. We hope to include the steps for JUWELS soon.

Table of contents

1. A word regarding the code samples
   a. Choose code to suit your needs; also download datasets
Table of contents

1. A word regarding the code samples
2. Changes made to support loading of pre-downloaded datasets
3. Applying for user accounts on supercomputers
   - 3.1. JURECA and JUWELS
   - 3.2. JURON
4. Logging on to the supercomputers
   - 4.1. JURECA
   - 4.2. JURON
5. Cloning the repository
   - 5.1. JURECA
   - 5.2. JURON
6. Running a sample
   - 6.1. JURECA
   - 6.2. JURON
7. Python 2 support
8. Distributed training
9. Credits
$ tree -L 1
.
  caffe
  datasets
  horovod
  keras
  pytorch
  README.md
  tensorflow
  utils

7 directories, 1 file
Tutorial

Keras samples

The following Keras samples are included:

1. **mnist.py**: A simple MNIST processing example with only the essential Horvod code for distributed training.
2. **mnist_advanced.py**: This sample is primarily the same as **mnist.py**. However, a few more advanced Horvod features are used.

PyTorch samples

The following PyTorch samples are included:

1. **mnist.py**: Demonstrates distributed training using Horvod with PyTorch. A simple convolutional neural network is trained on the MNIST dataset.
2. **synthetic_benchmark.py**: A benchmark that can be used to measure performance of PyTorch with Horvod without using any external dataset.

Note: The job scripts for JURECA are prefixed with `j` for these samples, so that these scripts do not appear in the directory listing. The reason for doing this is that our testing revealed issues with multi-node training. As soon as the issue has been resolved, we'll make the scripts available.

Tensorflow samples

The following Tensorflow samples are included:

1. **mnist.py**: Demonstrates distributed training using Horvod with the low-level Tensorflow API. A simple convolutional neural network is trained on the MNIST dataset.
2. **mnist_estimator.py**: Demonstrates distributed training using Horvod with the high-level Estimator API in Tensorflow. A simple
Running a sample (1)

```
[khalid1@r106 ~]$ jutil env activate -p cslns -A slns
[khalid1@r106 ~]$ cd $PROJECT/khalid1
[khalid1@r106 khalid1]$ module use /usr/local/software/jureca/OtherStages
[khalid1@r106 khalid1]$ module load Stages/2018b
    Preparing the environment for use of requested stage ( 2018b ).
```

Due to MODULEPATH changes, the following have been reloaded:

1) StdEnv

The following have been reloaded with a version change:

1) GCCcore/.8.3.0 => GCCcore/.7.3.0  
2) binutils/.2.32 => binutils/.2.31.1
Running a sample (2)

[khalid1@jr106 khalid1]$ module load git-lfs/2.6.1

[khalid1@jr106 khalid1]$ git-lfs install
Git LFS initialized.

[khalid1@jr106 khalid1]$ git-lfs clone https://gitlab.version.fz-juelich.de/khalid1/ml_dl_on_supercomputers.git
Cloning into 'ml_dl_on_supercomputers'...
remote: Enumerating objects: 190, done.
remote: Counting objects: 100% (190/190), done.
remote: Compressing objects: 100% (90/90), done.
remote: Total 190 (delta 100), reused 182 (delta 92)
Receiving objects: 100% (190/190), 48.93 KiB | 0 bytes/s, done.
Resolving deltas: 100% (100/100), done.
Tutorial

Running a sample (3)

[khalid1@jr106 khalid1]$ cd ml_dl_on_supercomputers/horovod/keras/

[khalid1@jr106 keras]$ sbatch submit_job_jureca_python3.sh
Submitted batch job 7056695

[khalid1@jr106 keras]$ squeue -u $USER

<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>7056695</td>
<td>develgpu</td>
<td>HOROVO</td>
<td>khalid1</td>
<td>CF</td>
<td>0:04</td>
<td>2 jrc[0002-0003]</td>
<td></td>
</tr>
</tbody>
</table>
Tutorial

Running a sample (4)

```
[khalid@jr103 keras]$ tail -f output_7056362.out
...
54016/60000 [=============================================>...] - ETA: 0s - loss: 0.0291 - acc: 0.9906
54400/60000 [=============================================>...] - ETA: 0s - loss: 0.0291 - acc: 0.9906
54784/60000 [=============================================>...] - ETA: 0s - loss: 0.0291 - acc: 0.9906
55168/60000 [=============================================>...] - ETA: 0s - loss: 0.0290 - acc: 0.9906
55552/60000 [=============================================>...] - ETA: 0s - loss: 0.0290 - acc: 0.9907
55936/60000 [=============================================>...] - ETA: 0s - loss: 0.0291 - acc: 0.9907
56320/60000 [=============================================>...] - ETA: 0s - loss: 0.0291 - acc: 0.9907
56704/60000 [=============================================>...] - ETA: 0s - loss: 0.0299 - acc: 0.9907
57088/60000 [=============================================>...] - ETA: 0s - loss: 0.0291 - acc: 0.9906
57472/60000 [=============================================>...] - ETA: 0s - loss: 0.0289 - acc: 0.9907
57856/60000 [=============================================>...] - ETA: 0s - loss: 0.0290 - acc: 0.9907
58240/60000 [=============================================>...] - ETA: 0s - loss: 0.0290 - acc: 0.9907
58624/60000 [=============================================>...] - ETA: 0s - loss: 0.0289 - acc: 0.9907
59008/60000 [=============================================>...] - ETA: 0s - loss: 0.0288 - acc: 0.9907
59392/60000 [=============================================>...] - ETA: 0s - loss: 0.0288 - acc: 0.9907
59776/60000 [=============================================>...] - ETA: 0s - loss: 0.0288 - acc: 0.9907
60000/60000 [=============================================>] - 10s 159us/step - loss: 0.0288 - acc: 0.9907 - val_loss: 0.0297 - val_acc: 0.9920
Test loss: 0.029660221893041943
Test accuracy: 0.992
```
The SLURM job script (1)

```
#SBATCH --nodes=2
#SBATCH --ntasks=4
#SBATCH --ntasks-per-node=2
#SBATCH --output=output_%j.out
#SBATCH --error=error_%j.err
#SBATCH --time=00:10:00
#SBATCH --job-name=HOROVOOD_KERAS_MNIST
#SBATCH --gres=gpu:2 --partition=develgpus
#SBATCH --mail-user=<your email address here>
#SBATCH --mail-type=ALL
```
The SLURM job script (2)

```bash
module use /usr/local/software/jureca/OtherStages
module load Stages/2018b
module load GCC/7.3.0
module load MVAPICH2/2.3-GDR
module load TensorFlow/1.12.0-GPU-Python-3.6.6
module load Keras/2.2.4-GPU-Python-3.6.6
module load Horovod/0.15.2-GPU-Python-3.6.6

srun python -u mnist.py
```
### Status of the relevant modules

**JURECA**

<table>
<thead>
<tr>
<th>Module</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Tensorflow</td>
<td>PASS</td>
</tr>
<tr>
<td>2. Keras</td>
<td>PASS</td>
</tr>
<tr>
<td>3. PyTorch</td>
<td>PASS</td>
</tr>
<tr>
<td>4. Caffe</td>
<td>PASS</td>
</tr>
<tr>
<td>5. Horovod + Keras (all samples)</td>
<td>PASS</td>
</tr>
<tr>
<td>6. Horovod + Tensorflow (all samples)</td>
<td>PASS</td>
</tr>
<tr>
<td>7. Horovod + PyTorch (synthetic_benchmark)</td>
<td>PASS</td>
</tr>
<tr>
<td>8. Horovod + PyTorch (mnist)</td>
<td>FAIL</td>
</tr>
</tbody>
</table>
Conclusion

Summary

- The supercomputers can be utilized for data parallel training with relative ease
- Keras, Tensorflow, PyTorch, Horovod and Caffe are available as system-wide modules on JURECA
- Following the tutorial is a great way to get started

Support

- All SC related issues: sc@fz-juelich.de
- Tutorial: Please write to me directly

Resources

- Tutorial: https://gitlab.version.fz-juelich.de/khalid1/ml_dl_on_supercomputers#getting-started-with-mldl-on-supercomputers
- Courses at the JSC: https://www.fz-juelich.de/ias/jsc/EN/Expertise/Workshops/Courses/courses_node.html?cms_qts=944518_list%253DstartDate_dt%252Bdesc

Thank you!
import tensorflow as tf

# Reference to the MNIST dataset
mnist = tf.keras.datasets.mnist

# Tuples in the (input, label) format for train and test sets
(x_train, y_train), (x_test, y_test) = mnist.load_data()

# Normalize input samples
x_train, x_test = x_train / 255.0, x_test / 255.0

# Define the model, i.e., the network
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)]
)

# Optimizer
optimizer = tf.keras.optimizers.Adam()

# Compile the model
model.compile(optimizers.Adam(),
               loss='sparse_categorical_crossentropy',
               metrics=['accuracy'])

# No. of epochs
epochs = 4

# Train the model using the training set
model.fit(x=x_train,
           y=y_train,
           batch_size=32,
           epochs=epochs,
           verbose=1)

# Test the model on the test set
score = model.evaluate(x=x_test,
                        y=y_test,
                        verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])

# Reference to the JÜLICH Forschungszentrum
JULICH
import math
import tensorflow as tf
import horovod.tensorflow.keras as hvd
from tensorflow.python.keras import backend as K

# Horovod: initialize Horovod.
hvd.init()

# Horovod: pin GPU to be used to process local rank (one GPU per process)
config = tf.ConfigProto()
config.gpu_options.visible_device_list = str(hvd.local_rank())
K.set_session(tf.Session(config=config))

# Reference to the MNIST dataset
mnist = tf.keras.datasets.mnist

# Tuples in the (input, label) format for train and test sets
(x_train, y_train), (x_test, y_test) = mnist.load_data()

# Normalize input samples
x_train, x_test = x_train / 255.0, x_test / 255.0

# Define the model, i.e., the network
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)
])

# Optimizer
optimizer = tf.keras.optimizers.Adam()

# Horovod: add Horovod Distributed Optimizer.
optimizer = hvd.DistributedOptimizer(optimizer)

# Compile the model
model.compile(
    optimizer=optimizer,
    loss='sparse_categorical_crossentropy',
    metrics=['accuracy'])

# Horovod: adjust number of epochs based on number of GPUs.
epochs = int(math.ceil(4.0 / hvd.size()))

# Training callbacks
callbacks = [
    # Horovod: broadcast initial variable states from rank 0 to all other processes.
    # This is necessary to ensure consistent initialization of all workers when
    # training is started with random weights or restored from a checkpoint.
    hvd.callbacks.BroadcastGlobalVariablesCallback(0)
]

# Train the model using the training set
model.fit(
    x=x_train,
    y=y_train,
    batch_size=32,
    epochs=epochs,
    verbose=1,
    callbacks=callbacks,
)

# Test the model on the test set
score = model.evaluate(x=x_test,
                       y=y_test,
                       verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])

mnist_simplest_horovod.py