Karlsruher Institut für Technologie

Institut für Experimentelle Teilchenphysik

Use of GPU resources for the Training of Neural Networks

Tim Voigtländer, Roger Wolf, Lars Sowa
tim.voigtlaender@student.kit.edu

Karlsruher Institut für Technologie (KIT) - Institut für experimentelle Teilchenphysik (ETP)

Will a GPU speed up your Machine Learning application?

It's complicated

- GPU speed up training immensely by parallelizing demanding tasks
- Some overhead is necessary to transfer data
- Individual computations are also a bit slower
- Usually a GPU replaces multiple CPU cores

➤ **Hard to determine if an ML application will be sped up**

Rule of thumb: GPUs help most for

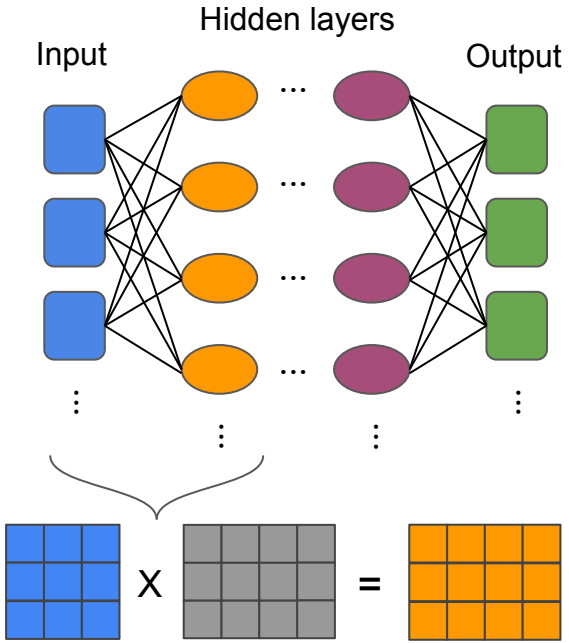
- Bigger networks
- Less complex networks

There are many hyperparameters to tune

The only sure way to know, is to try it out

Example: Event Discrimination via NN

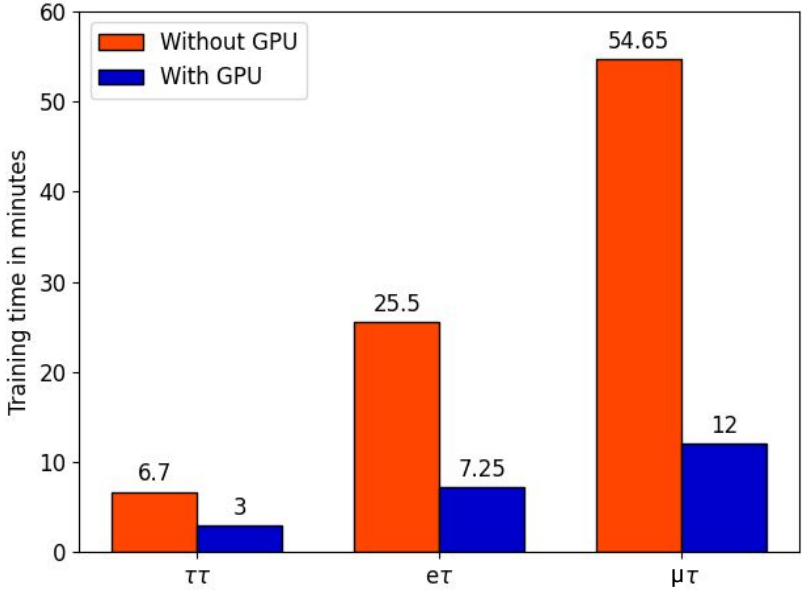
- **To differentiate between signal and background, Neural Networks are trained**
  - **22** to **28** physical variables are used as inputs
  - **5** classes are used as outputs
    - **4** are composed of various backgrounds
    - The signal for a specific group of mass hypotheses
  - The network is comparably simple
    - Feed forward fully connected network
    - **3** hidden layers with **512** nodes each
  - ➤ Around **45,000** parameters



InceptionResnetV2 has around **56,000,000** Parameters

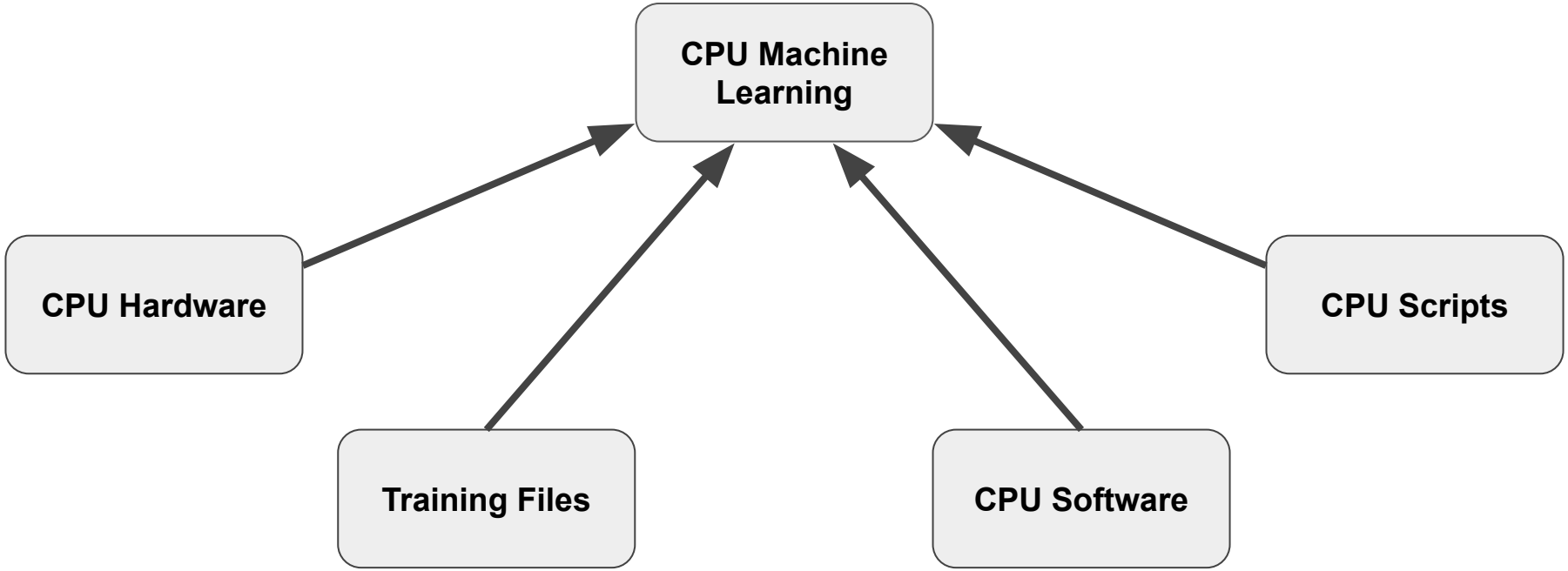
Example: CPU vs GPU

- **Part of Master thesis:**
  - Adapt training for GPU
  - Find ideal Hyperparameters
  - Compare with CPU
- **CPU only**
  - 12 CPU cores
  - Average runtime: 29 minutes
- **CPU + GPU**
  - 2 CPU cores + A100 GPU
  - Average runtime: 7.5 Minutes



➤ **Inclusion of GPU leads to a significant speedup while using fewer CPU resources**

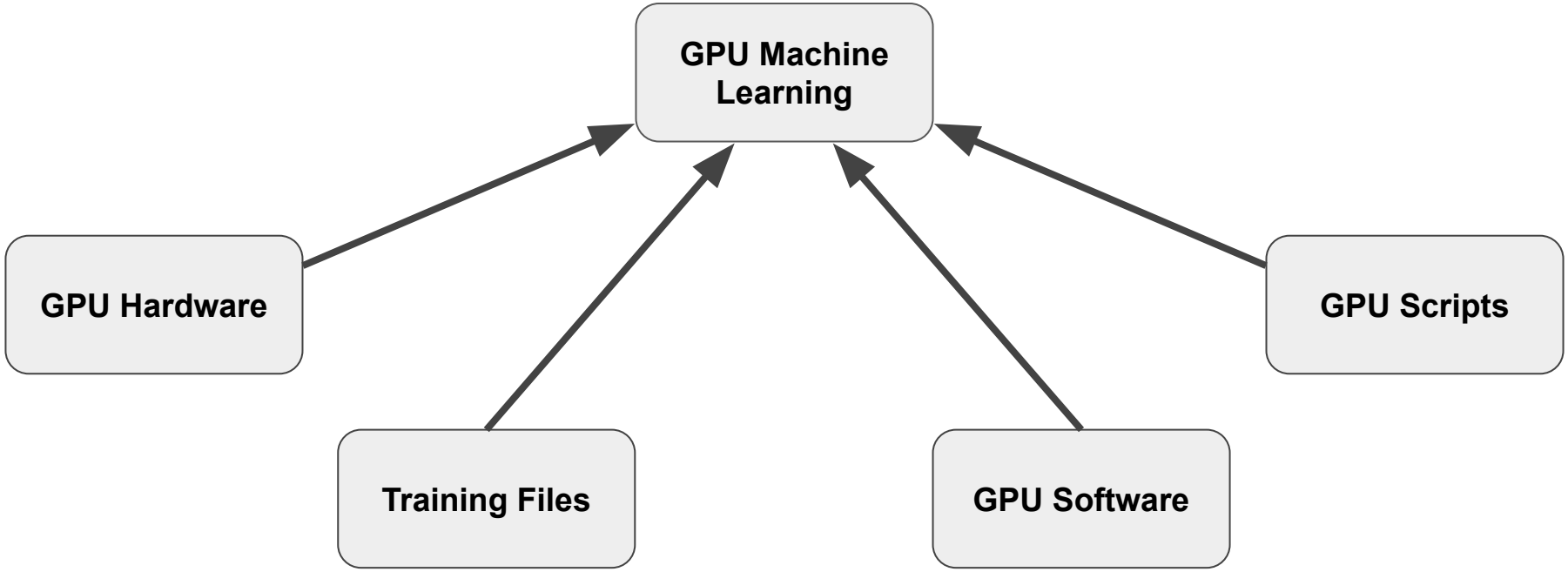
What so we need for CPU training



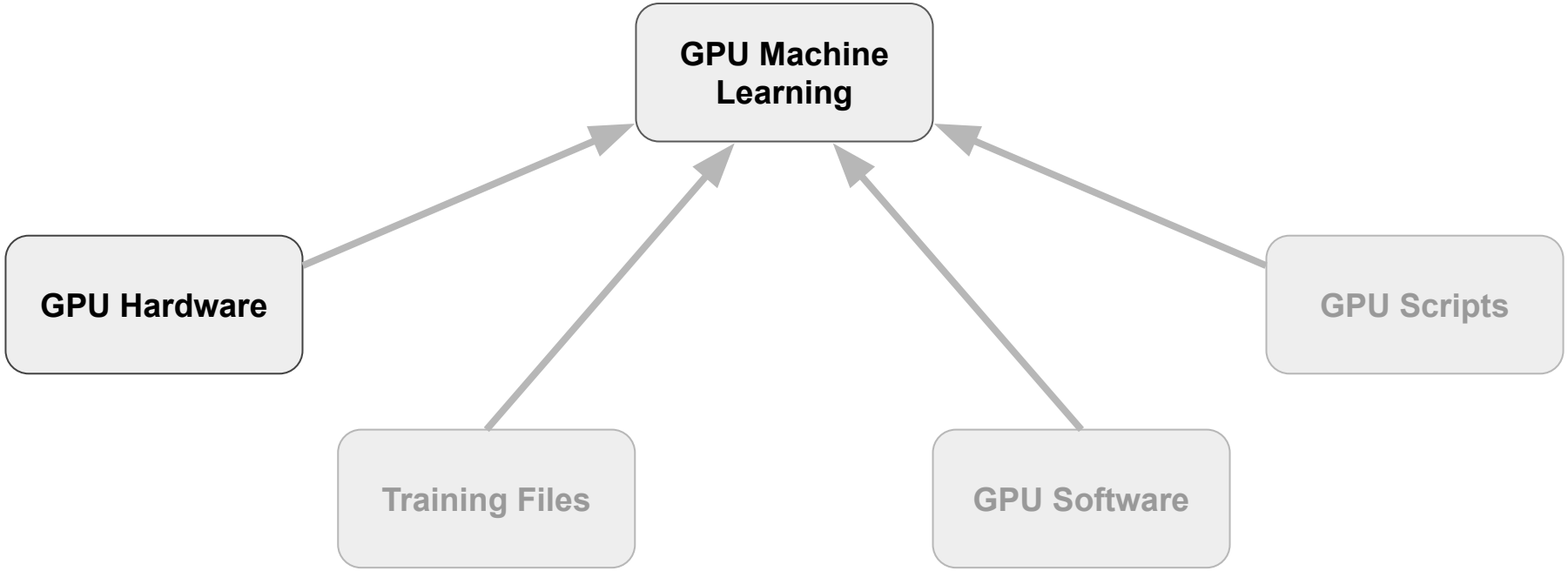
How a training on local CPU might look like (executable.sh)

```
#/bin/bash

# Source all necessary software

source /work/name/some/local/environment/setup.sh


# Run the training

python3 /work/name/some/local/files/mnist_training.py
    --files /ceph/name/some/local/files/training_data
```

What so we need for GPU training



What so we need for GPU training



What GPUs can you use?

The “Deepthought” machine

- Local development machine
- Ubuntu 18.04
- 4 GTX TITAN X

The TOpAS cluster

- Remotely usable through the ETP batch system
- CentOS7
- Multiple machines with
  - V100, V100S, A100
- 56 GPUs in total

What GPUs can you use?

The “Deepthought” machine

- Local development machine
- Ubuntu 18.04
- 4 GTX TITAN X

The TOpAS cluster

- Remotely usable through the ETP batch system
- CentOS7
- Multiple machines with
  - V100, V100S, A100
- 56 GPUs in total

What is in the Jobfile for a CPU job

```
# The Executable that is run in the Job

Executable = ./executable.sh

...

# The hardware requirements

RequestCPUs = 1         # 1 CPU core

RequestMemory = 2000      # 2 GB of memory

request_disk = 2000000   # 2 GB of disk space

...
```

HTCondor Talk by Matthias Schnepf

Corresponding GitLab repo

What is in the Jobfile for a GPU job on TOpAS

```
...
## Additional Arguments in submission.jdl
# Request GPU resources
# Can also be >1 to request more GPUs
request_GPUs = 1
# Mark as remote job
+RemoteJob = True
```

What is in the Jobfile for a GPU job on TOpAS

```
...
## Additional Arguments in submission.jdl
# Request GPU resources
# Can also be >1 to request more GPUs
request_GPUs = 1
# Mark as remote job
+RemoteJob = True
```

What is a Remote Job?

Remote jobs do not have access to `/ceph`, `/work` and `/home`

What is a Remote Job?

Remote jobs do not have access to `/ceph`, `/work` and `/home`

Neither

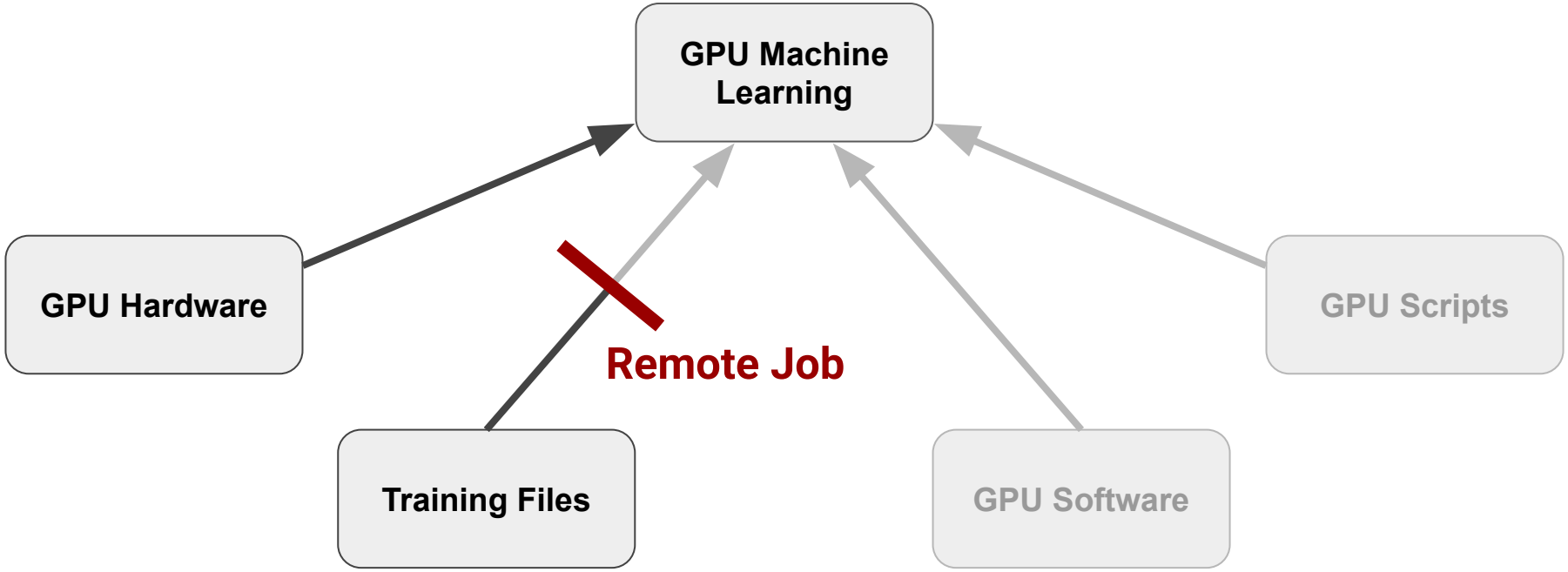
```
source /work/name/some/local/environment/setup.sh
```

nor

```
python3 /work/name/some/local/files/mnist_training.py
    --files /ceph/name/some/local/files/training_data
```

will work!

What so we need for GPU training on TOpAS



How to transfer Files to Remote Batch Jobs

Small files (Together < 1 GB):

- Through HTCondor internal transfer

Shown in GitLab examples

How to transfer small files

```
...
## Additional Arguments in submission.jdl
# List of files to transfer into job
transfer_input_files = mnist_training.py
# List of files to transfer out of job
transfer_output_files = trained_model.pt
# Paths can be adjusted with transfer_output_remaps
```

How to transfer Files to Remote Batch Jobs

Small files (Together < 1 GB):

- Through HTCondor internal transfer

Shown in GitLab examples

Big Files:

- Transfer with Grid protocols (SRM, XRootD, etc.)
  - Directly from `/ceph` via `/ceph/srv`
    - Can be slow
  - Through intermediate Grid storage
    - Often requires VOMS proxy

Shown in GitLab examples

How to transfer big files

1. **Place files on accessible storage before the job starts**
   a. `/ceph/srv` works for ETP and TOpAS
   b. Remote storage like NRG

How to transfer big files

1. **Place files on accessible storage before the job starts**
   a. `/ceph/srv` works for ETP and TOpAS
   b. Remote storage like NRG
2. **Copy files into the job with Grid Protocols at runtime** (like `xrdcp`)

Files on
`/ceph/srv/name/path/to/dataset`
can be found as
`root://ceph-node-a.etp.kit.edu://name/path/to/dataset`

Documentation of XRootD

How to transfer Files to Remote Batch Jobs

Small files (Together < 1 GB):

- Through HTCondor internal transfer

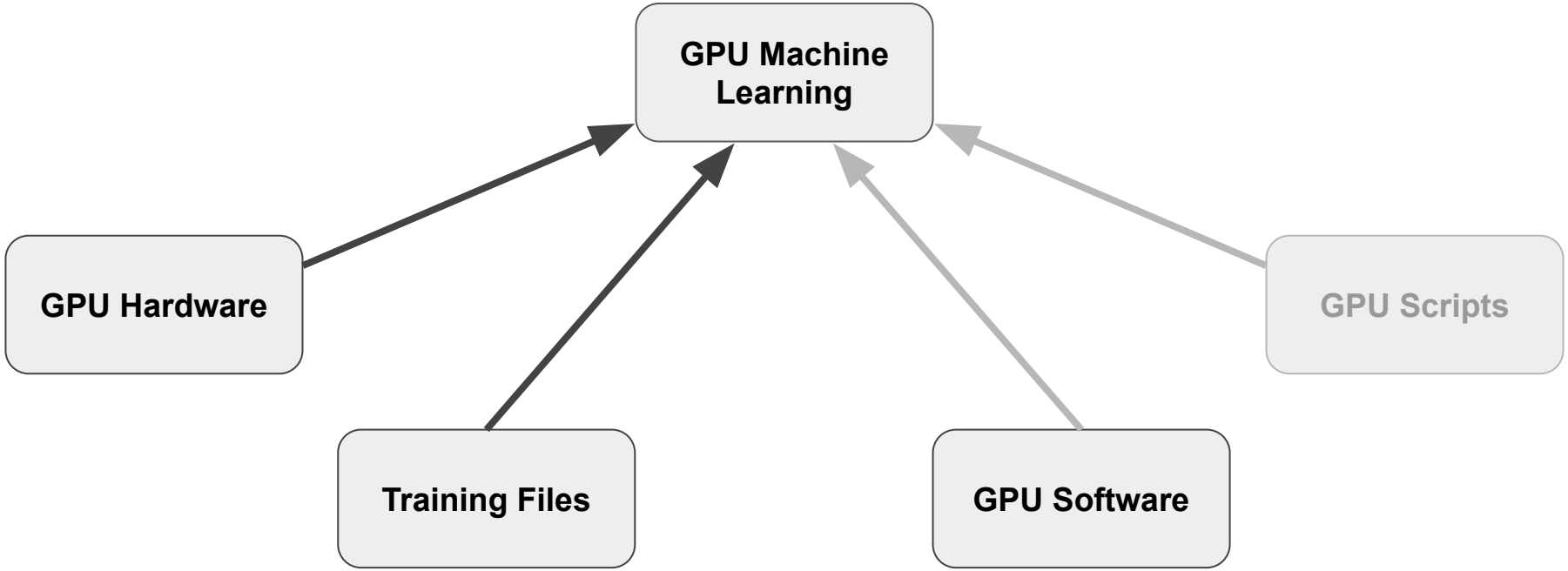
Shown in GitLab examples

Big Files:

- Transfer with Grid protocols (SRM, XRootD, etc.)
  - Directly from `/ceph` via `/ceph/srv`
    - Can be slow
  - Through intermediate Grid storage
    - Often requires VOMS proxy
- Some data types can be streamed
- Subject to caching on TOpAS

Shown in GitLab examples

What so we need for GPU training on TOpAS



Requirements for your GPU Environment

Set-up has to be:

- **GPU capable**

CUDA + **cuDNN** + **GPU versions of software**

- **Portable**

Send with job | **Get from CVMFS** | **Get from Web**

GPU Environment Examples: LCG Stack

```
source /cvmfs/.../LCG_101cuda/.../setup.sh
HOME=$_CONDOR_JOB_IWD
python3 -m venv venv
source venv/bin/activate
python3 -m pip install pip --upgrade
python3 -m pip install torch==1.9.0+cu111
    torchvision==0.10.0+cu111 torchaudio==0.9.0
    -f https://download.pytorch.org/whl/torch_stable.html
```

List of available LCG stacks

Source base environment from CUDA LCG stacks

GPU Environment Examples: LCG Stack

```
source /cvmfs/.../LCG_101cuda/.../setup.sh
HOME=$_CONDOR_JOB_IWD
python3 -m venv venv
source venv/bin/activate
python3 -m pip install pip --upgrade
python3 -m pip install torch==1.9.0+cu111
    torchvision==0.10.0+cu111 torchaudio==0.9.0
    -f https://download.pytorch.org/whl/torch_stable.html
```

List of available LCG stacks

Set `$HOME` to writable directory. `$_CONDOR_JOB_IWD` is the job start directory

GPU Environment Examples: LCG Stack

```
source /cvmfs/.../LCG_101cuda/.../setup.sh
HOME=$_CONDOR_JOB_IWD
python3 -m venv venv
source venv/bin/activate
python3 -m pip install pip --upgrade
python3 -m pip install torch==1.9.0+cu111
    torchvision==0.10.0+cu111 torchaudio==0.9.0
    -f https://download.pytorch.org/whl/torch_stable.html
```

List of available LCG stacks

Set up local `venv` to install missing modules

GPU Environment Examples: LCG Stack

```
source /cvmfs/.../LCG_101cuda/.../setup.sh
HOME=$_CONDOR_JOB_IWD
python3 -m venv venv
source venv/bin/activate
python3 -m pip install pip --upgrade
python3 -m pip install torch==1.9.0+cu111
    torchvision==0.10.0+cu111 torchaudio==0.9.0
    -f https://download.pytorch.org/whl/torch_stable.html
```

List of available LCG stacks

Install modules compatible with GPU Hardware and LCG stack libraries

GPU Environment Examples: LCG Stack

```
source /cvmfs/.../LCG_101cuda/.../setup.sh
HOME=$_CONDOR_JOB_IWD
python3 -m venv venv
source venv/bin/activate
python3 -m p
python3 -m p
    torchvision==0.10.0+cu111 torchaudio==0.9.0
    -f https://download.pytorch.org/whl/torch_stable.html
```

List of available LCG stacks

Newest available CUDA version is 11.2

Install modules compatible with GPU Hardware and LCG stack libraries

GPU Environment Examples: Anaconda

```
HOME=$_CONDOR_JOB_IWD
wget https://repo.anaconda.com/miniconda/
    Miniconda3-latest-Linux-x86_64.sh
chmod +x Miniconda3-latest-Linux-x86_64.sh
./Miniconda3-latest-Linux-x86_64.sh -b -p $(pwd)/miniconda
source miniconda/bin/activate
conda install pytorch torchvision torchaudio
    cudatoolkit=11.3 -c pytorch -y
```

Set `$HOME` to writable directory. `$_CONDOR_JOB_IWD` is the job start directory

GPU Environment Examples: Anaconda

```
HOME=$_CONDOR_JOB_IWD
wget https://repo.anaconda.com/miniconda/
    Miniconda3-latest-Linux-x86_64.sh
chmod +x Miniconda3-latest-Linux-x86_64.sh
./Miniconda3-latest-Linux-x86_64.sh -b -p $(pwd)/miniconda
source miniconda/bin/activate
conda install pytorch torchvision torchaudio
    cudatoolkit=11.3 -c pytorch -y
```

Download an Anaconda installer

GPU Environment Examples: Anaconda

```
HOME=$_CONDOR_JOB_IWD
wget https://repo.anaconda.com/miniconda/
    Miniconda3-latest-Linux-x86_64.sh
chmod +x Miniconda3-latest-Linux-x86_64.sh
./Miniconda3-latest-Linux-x86_64.sh -b -p $(pwd)/miniconda
source miniconda/bin/activate
conda install pytorch torchvision torchaudio
    cudatoolkit=11.3 -c pytorch -y
```

Set up Anaconda environment

GPU Environment Examples: Anaconda

```
HOME=$_CONDOR_JOB_IWD
wget https://repo.anaconda.com/miniconda/
    Miniconda3-latest-Linux-x86_64.sh
chmod +x Miniconda3-latest-Linux-x86_64.sh
./Miniconda3-latest-Linux-x86_64.sh -b -p $(pwd)/miniconda
source miniconda/bin/activate
conda install pytorch torchvision torchaudio
    cudatoolkit=11.3 -c pytorch -y
```

Install software compatible with GPU Hardware

GPU Environment Examples: Anaconda

```
HOME=$_CONDOR_JOB_IWD
wget https://repo.anaconda.com/miniconda/
    Miniconda3-latest-Linux-x86_64.sh
chmod +x Miniconda3-latest-Linux-x86_64.sh
./Miniconda3                                    /miniconda
source minic
conda install pytorch torchvision torchaudio
    cudatoolkit=11.3 -c pytorch -y
```

Newest available CUDA version is 11.3

Install software compatible with GPU Hardware

GPU Environment Examples: Docker

Batch jobs at ETP are usually set up to run in a docker container

```
...
Universe = docker
## Docker image used by the job
# Default container
docker_image = mschnepf/slc7-condocker
# Container with CUDA + cuDNN libraries preinstalled
docker_image =
    tvoigtlaender/slc7-condocker-cuda-11.6-cudnn8-devel:base
```

GPU Environment Examples: Docker

Batch jobs at ETP are usually set up to run in a docker container

```
...
Universe = docker
## Docker image used by the job
# Default container
docker_image = mschnepf/slc7-condocker
# Container with CUDA + cuDNN libraries preinstalled
docker_image =
    tvoigtlaender/slc7-condocker-cuda-11.6-cudnn8-devel:base
```

GPU Environment Examples: Docker

```
# CUDA and cuDNN are already installed from the start
HOME=$_CONDOR_JOB_IWD
python3 -m venv venv
source venv/bin/activate
python3 -m pip install pip --upgrade
python3 -m pip install torch==1.9.0+cu111
    torchvision==0.10.0+cu111 torchaudio==0.9.0
    -f https://download.pytorch.org/whl/torch_stable.html
```

Set `$HOME` to writable directory. `$_CONDOR_JOB_IWD` is the job start directory

GPU Environment Examples: Docker

```
# CUDA and cuDNN are already installed from the start
HOME=$_CONDOR_JOB_IWD
python3 -m venv venv
source venv/bin/activate
python3 -m pip install pip --upgrade
python3 -m pip install torch==1.9.0+cu111
    torchvision==0.10.0+cu111 torchaudio==0.9.0
    -f https://download.pytorch.org/whl/torch_stable.html
```

Set up local `venv` to install missing modules

GPU Environment Examples: Docker

```
# CUDA and cuDNN are already installed from the start
HOME=$_CONDOR_JOB_IWD
python3 -m venv venv
source venv/bin/activate
python3 -m pip install pip --upgrade
python3 -m pip install torch==1.9.0+cu111
    torchvision==0.10.0+cu111 torchaudio==0.9.0
    -f https://download.pytorch.org/whl/torch_stable.html
```

Install modules compatible with GPU Hardware and preinstalled libraries

GPU Environment Examples: Docker

```
# CUDA and cuDNN are already installed from the start
HOME=$_CONDOR_JOB_IWD
python3 -m venv venv
source venv/bin/activate
python3 -m p
python3 -m p
    torchvision==0.10.0+cu111 torchaudio==0.9.0
    -f https://download.pytorch.org/whl/torch_stable.html
```

Newest available CUDA version is 11.6

Install modules compatible with GPU Hardware and preinstalled libraries

GPU Environment Examples: `/cvmfs/etp.kit.edu`

If you have a stable environment setup, it can be put on

`/cvmfs/etp.kit.edu`

Files there can be **read** from inside the batch job

CVMFS Documentation

GPU Environment Examples: `/cvmfs/etp.kit.edu`

If you have a stable environment setup, it can be put on

`/cvmfs/etp.kit.edu`

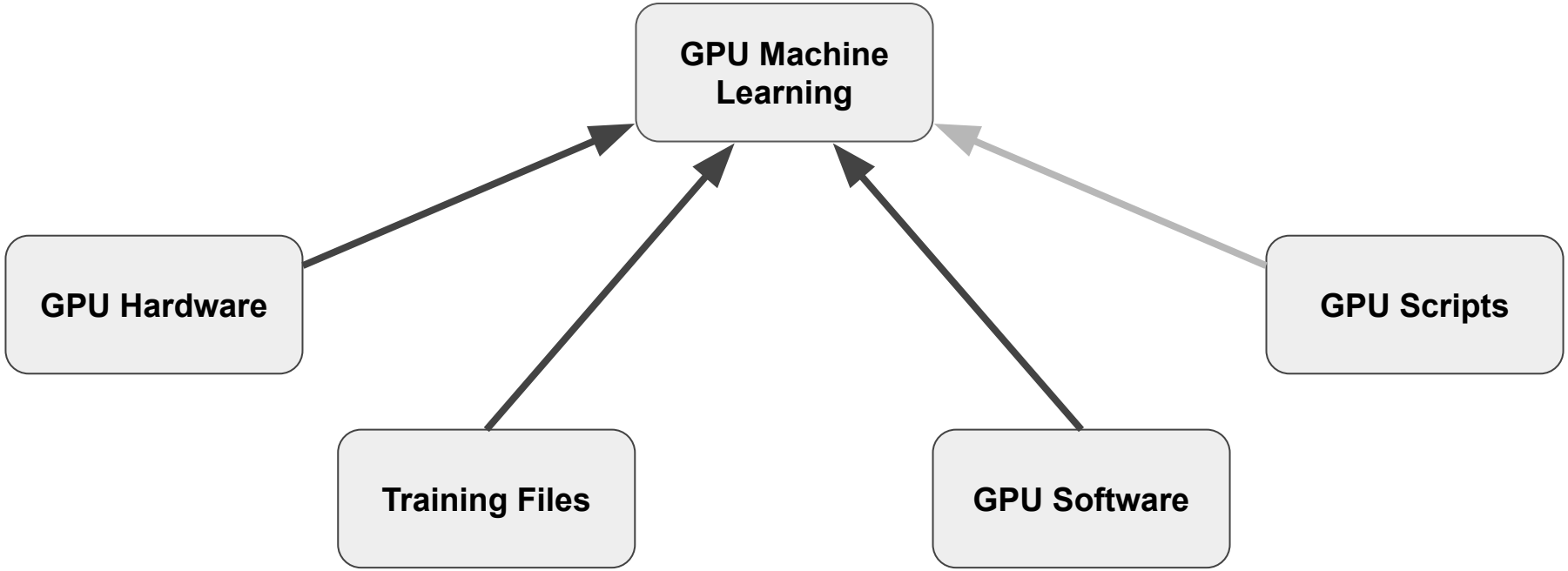
Files there can be **read** from inside the batch job

CVMFS Documentation

The **Anaconda** environment setup placed on **`/cvmfs/etp.kit.edu`** :

```
source
    /cvmfs/etp.kit.edu/GPU_examples/miniconda/bin/activate
    GPU_ML
```

What so we need for GPU training on TOpAS



How to train on GPU: Some Notes

- The example is a shortened version of the GitLab example (Link)
- Everything was tested on fairly new versions (PyTorch **1.9**, Tensorflow **2.4**)
  - Significantly older versions (especially Tensorflow 1) will deviate
  - GPU training with Tensorflow 1 is still possible but requires more work
- Ideally, analyses should not run on old software
- I can provide examples for older versions if necessary

The Training on CPU (`mnist_training.py`)

```
# Define NN model
class Net(torch.nn.Module):
    < Definition of model >
model = Net()



# Other setups
optimizer, scheduler,
train_loader, validation_loader
```

The Training on CPU (`mnist_training.py`)

```
# Define NN model
class Net(torch.nn.Module):
    < Definition of model >
model = Net()



# Other setups
optimizer, scheduler,
train_loader, validation_loader
```

The Training on CPU (`mnist_training.py`)

```
# Run training loop
for epoch in range(1, epochs + 1):
    train(model, ...)
    validation(model, ...)
    scheduler.step()


# Export trained model
torch.save(model.state_dict(), out_dir)
```

The Training on CPU (`mnist_training.py`)

```
# Run training loop
for epoch in range(1, epochs + 1):
    train(model, ...)
    validation(model, ...)
    scheduler.step()


# Export trained model
torch.save(model.state_dict(), out_dir)
```

The Training on CPU (mnist_training.py)

```
# The training step
def train(model, train_loader, optimizer):
    model.train()
    for data, target in train_loader:

        optimizer.zero_grad()
        output = model(data)
        loss = LossFunctional(output, target)
        loss.backward()
        optimizer.step()
```

The Training on CPU (mnist_training.py)

```
# The validation step
def validation(model, validation_loader):
    model.eval()
    loss = 0
    with torch.no_grad():
        for data, target in validation_loader:

            output = model(data)
            loss += LossFunctional(output, target)
```

How to train on GPU with PyTorch

1. **Detect the GPU**
2. Place model on GPU
3. Place data on GPU

Article on GPU usage in PyTorch

How to train on GPU with PyTorch

1. Detect the GPU
2. Place model on GPU
3. Place data on GPU

Article on GPU usage in PyTorch

How to train on GPU with PyTorch

1. Detect the GPU
2. Place model on GPU
3. Place data on GPU

Article on GPU usage in PyTorch

How to train on GPU with PyTorch

1. Detect the GPU
2. Place model on GPU
3. Place data on GPU

Article on GPU usage in PyTorch

→ PyTorch commands utilize the GPU

How to detect the GPU

```
# Detect if GPU are usable
GPU_check = torch.cuda.is_available()

# Detect how many GPUs are usable
num_GPUs = torch.cuda.device_count()

# Set which device to use
device = torch.device("cuda" if GPU_check
   else "cpu")
```

How to detect the GPU

```
# Detect if GPU are usable
GPU_check = torch.cuda.is_available()

# Detect how many GPUs are usable
num_GPUs = torch.cuda.device_count()

# Set which device to use
device = torch.device("cuda" if GPU_check
   else "cpu")
```

How to detect the GPU

```
# Detect if GPU are usable
GPU_check = torch.cuda.is_available()

# Detect how many GPUs are usable
num_GPUs = torch.cuda.device_count()

# Set which device to use
device = torch.device("cuda" if GPU_check
    else "cpu")
```

The Training on CPU (mnist_training.py)

```
# Define NN model
class Net(torch.nn.Module):
    < Definition of model >
model = Net()



# Other setups
optimizer, scheduler,
train_loader, validation_loader
```

The Training on GPU (`mnist_training.py`)

```
# Define NN model
class Net(torch.nn.Module):
    < Definition of model >
model = Net()
model = model.to(device)

# Other setups
optimizer, scheduler,
train_loader, validation_loader
```

- On CPU
- On Device

The Training on CPU (`mnist_training.py`)

```
# Run training loop
for epoch in range(1, epochs + 1):
    train(model, ...)
    validation(model, ...)
    scheduler.step()


# Export trained model
torch.save(model.state_dict(), out_dir)
```

The Training on CPU (`mnist_training.py`)

```
# Run training loop
for epoch in range(1, epochs + 1):
    train(model, device, ...)
    validation(model, device, ...)
    scheduler.step()


# Export trained model
torch.save(model.state_dict(), out_dir)
```

- On CPU
- On Device

The Training on CPU (mnist_training.py)

```
# The training step
def train(model, train_loader, optimizer):
    model.train()
    for data, target in train_loader:

        optimizer.zero_grad()
        output = model(data)
        loss = LossFunctional(output, target)
        loss.backward()
        optimizer.step()
```

The Training on GPU (`mnist_training.py`)

```
# The training step
def train(model, device, train_loader, optimizer):
    model.train()
    for data, target in train_loader:
        data, target = data.to(device), target.to(device)
        optimizer.zero_grad()
        output = model(data)
        loss = LossFunctional(output, target)
        loss.backward()
        optimizer.step()
```

- On CPU
- On Device

The Training on CPU (mnist_training.py)

```
# The validation step
def validation(model, validation_loader):
    model.eval()
    loss = 0
    with torch.no_grad():
        for data, target in validation_loader:

            output = model(data)
            loss += LossFunctional(output, target)
```

The Training on CPU (mnist_training.py)

```
# The validation step
def validation(model, device, validation_loader):
    model.eval()
    loss = 0
    with torch.no_grad():
        for data, target in validation_loader:
            data, target = data.to(device), target.to(device)
            output = model(data)
            loss += LossFunctional(output, target)
```

- On CPU
- On Device

How to train on GPU with TensorFlow

➔ **TensorFlow commands utilize the GPU by default**

Guide to GPU usage with TensorFlow

How to train on GPU with TensorFlow

➔ **TensorFlow commands utilize the GPU by default**

- **GPUs can still be managed manually**

[Guide to GPU usage with TensorFlow]

```
# Detect which GPUs are available
GPU_list = tensorflow.config.list_physical_devices('GPU')


# Run code on specific device
with tensorflow.device(GPU_list[0]):
    < Some calculation >
```

How to train on GPU with TensorFlow

➔ **TensorFlow commands utilize the GPU by default**

- **GPUs can still be managed manually**

Guide to GPU usage with TensorFlow

```
# Detect which GPUs are available
GPU_list = tensorflow.config.list_physical_devices('GPU')


# Run code on specific device
with tensorflow.device(GPU_list[0]):
    < Some calculation >
```

GPU training on TOpAS: Putting it all together

Submit remote GPU job via `condor_submit submit.jdl`

GPU training on TOpAS: Putting it all together

Submit remote GPU job via `condor_submit submit.jdl`

What should happen in the Executable of the job:

1. Set up GPU capable environment

GPU training on TOpAS: Putting it all together

Submit remote GPU job via `condor_submit submit.jdl`

What should happen in the Executable of the job:

1. Set up GPU capable environment
2. Copy training data into job

GPU training on TOpAS: Putting it all together

Submit remote GPU job via `condor_submit submit.jdl`

What should happen in the Executable of the job:

1. Set up GPU capable environment
2. Copy training data into job
3. Run GPU adapted NN training

GPU training on TOpAS: Putting it all together

Submit remote GPU job via `condor_submit submit.jdl`

What should happen in the Executable of the job:

1. Set up GPU capable environment
2. Copy training data into job
3. Run GPU adapted NN training
4. Return trained model from job

Related Topics

- Partial- and Multi GPU usage
- (GPU)-Job management
- GPU related Hyperparameter tuning
- Framework specific performance optimization
- More about Hardware related improvements for ML
- ML Profiling with Tensorboard

Let us know if you are interested in any of these topics

Collection of Links

- GitLab repo with examples for this talk
- Indico of Matthias Schnepfs HTCondor talk
  - Corresponding GitLab repo
- Documentation of XRootD
- List of available LCG stacks
- CVMFS Documentation
- Article on GPU usage in PyTorch
- Guide to GPU usage with TensorFlow
- TOpAS monitoring

Thank you for your attention