



# Agenda

1. Concepts: Fundamentals
  - i. Supervised learning
  - ii. Artificial Neural Networks
  - iii. Error backpropagation
2. Code example: 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. Code examples: Distributed training
  - i. MNIST classification: Epoch distributed
  - ii. Distributing training data
5. The “Deep Learning on Supercomputers” tutorial
  - i. Introduction
  - ii. Job configuration
  - iii. Running code samples on JUWELS
6. Conclusion

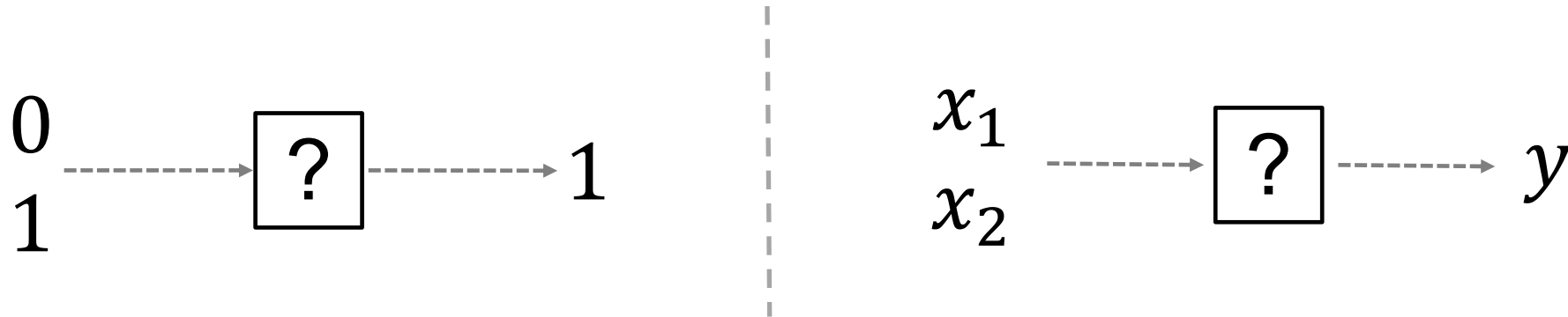
# Agenda

1. Concepts: Fundamentals
  - i. Supervised learning
  - ii. Artificial Neural Networks
  - iii. Error backpropagation
2. Code example: 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. Code examples: Distributed training
  - i. MNIST classification: Epoch distributed
  - ii. Distributing training data
5. The “Deep Learning on Supercomputers” tutorial
  - i. Introduction
  - ii. Job configuration
  - iii. Running code samples on JUWELS
6. Conclusion

# Concepts

## Supervised learning (1)

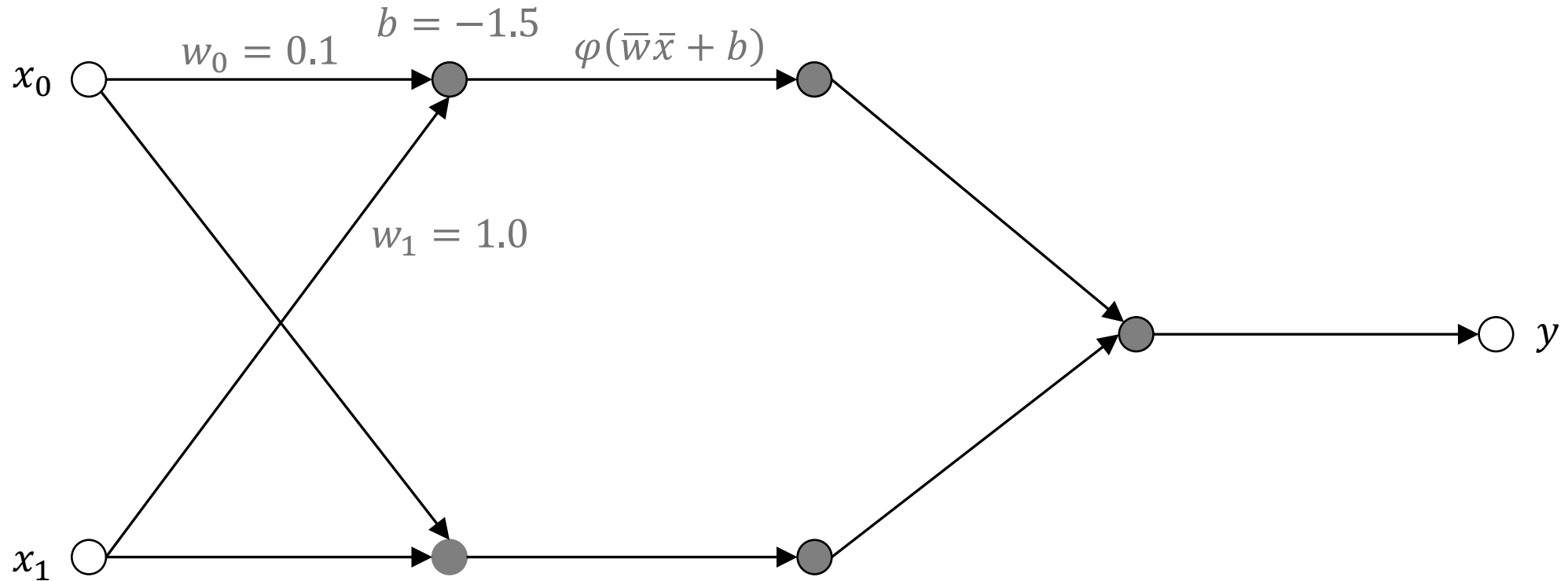
$x_1$	$x_2$	$y$
0	0	0
0	1	1
1	0	1
1	1	0





# Concepts

## Artificial Neural Networks

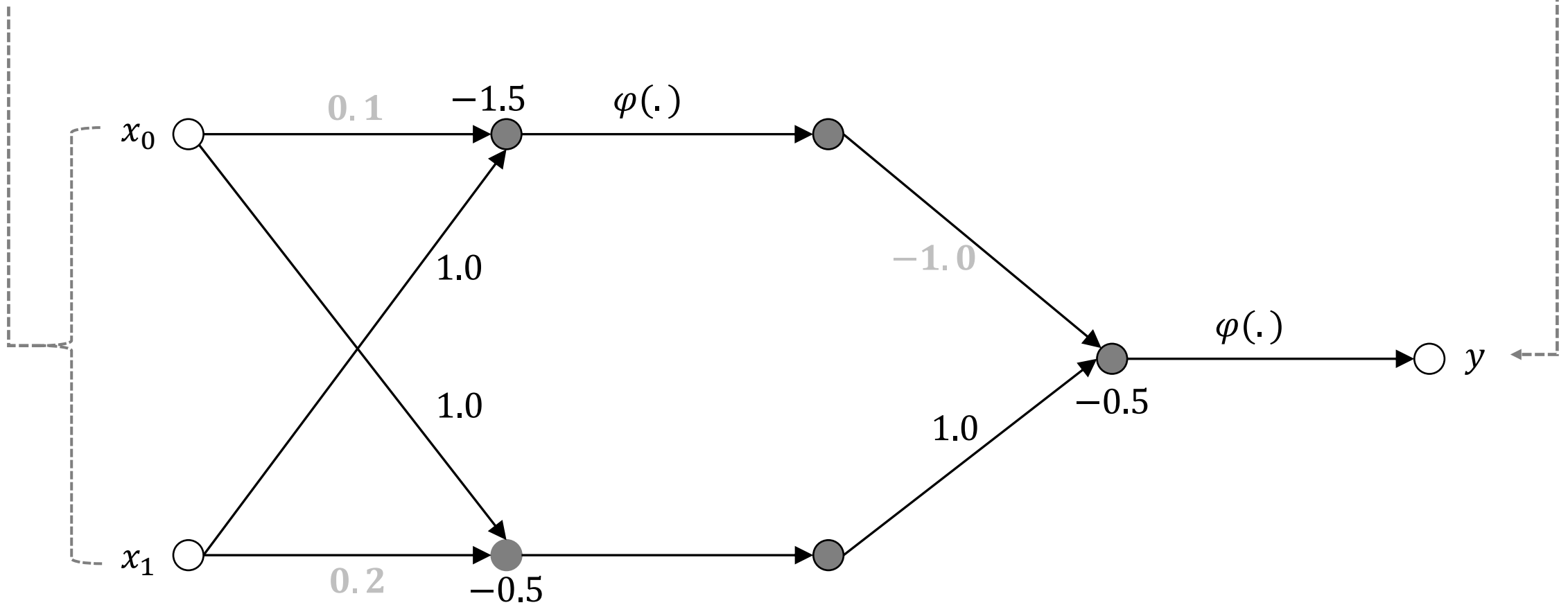


# Concepts

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

Error backpropagation (1)

Error  
 $\hat{y} - y$

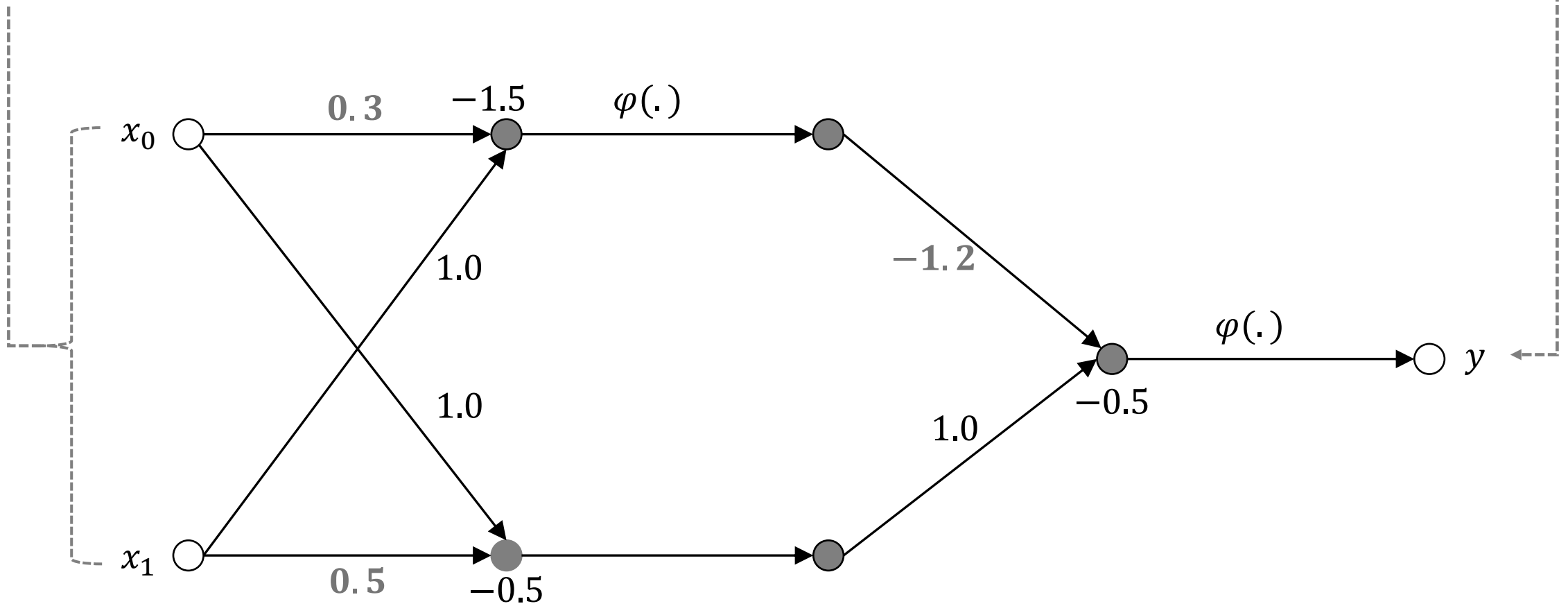


# Concepts

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

Error backpropagation (2)

Error  
 $\hat{y} - y$

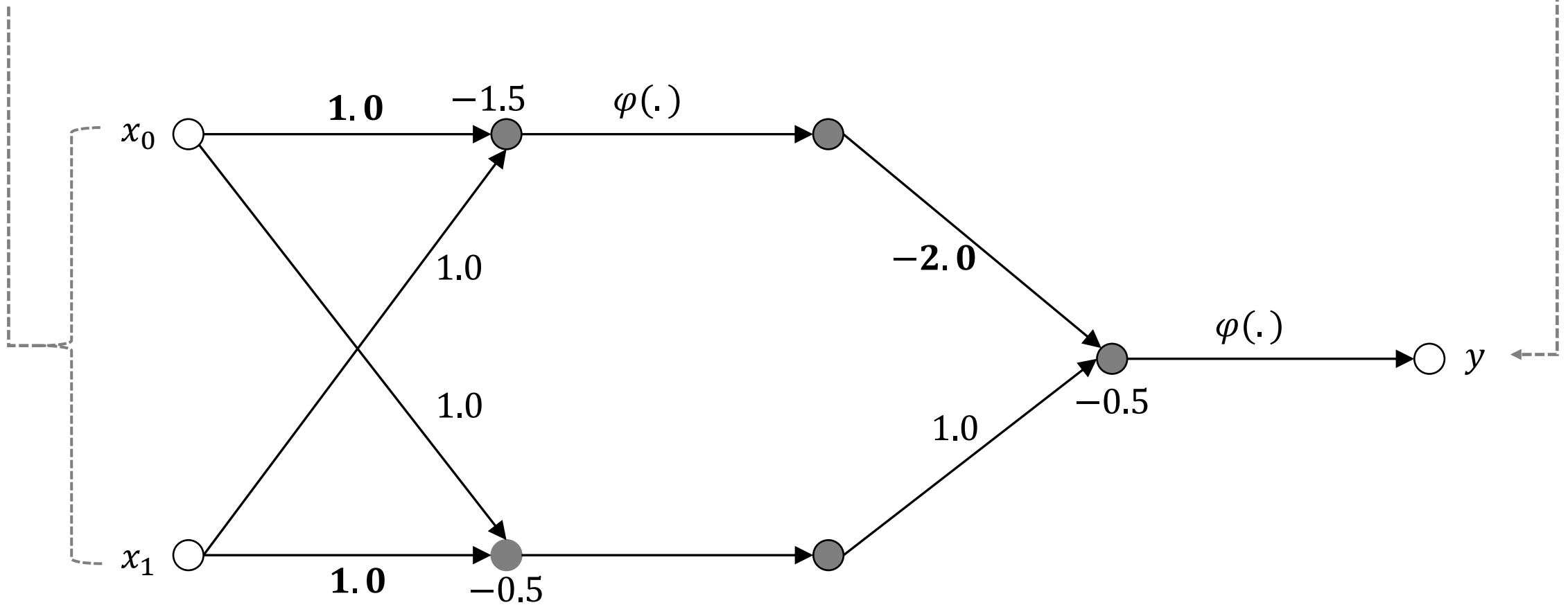


# Concepts

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

Error backpropagation (3)

Error  
 $\hat{y} - y$

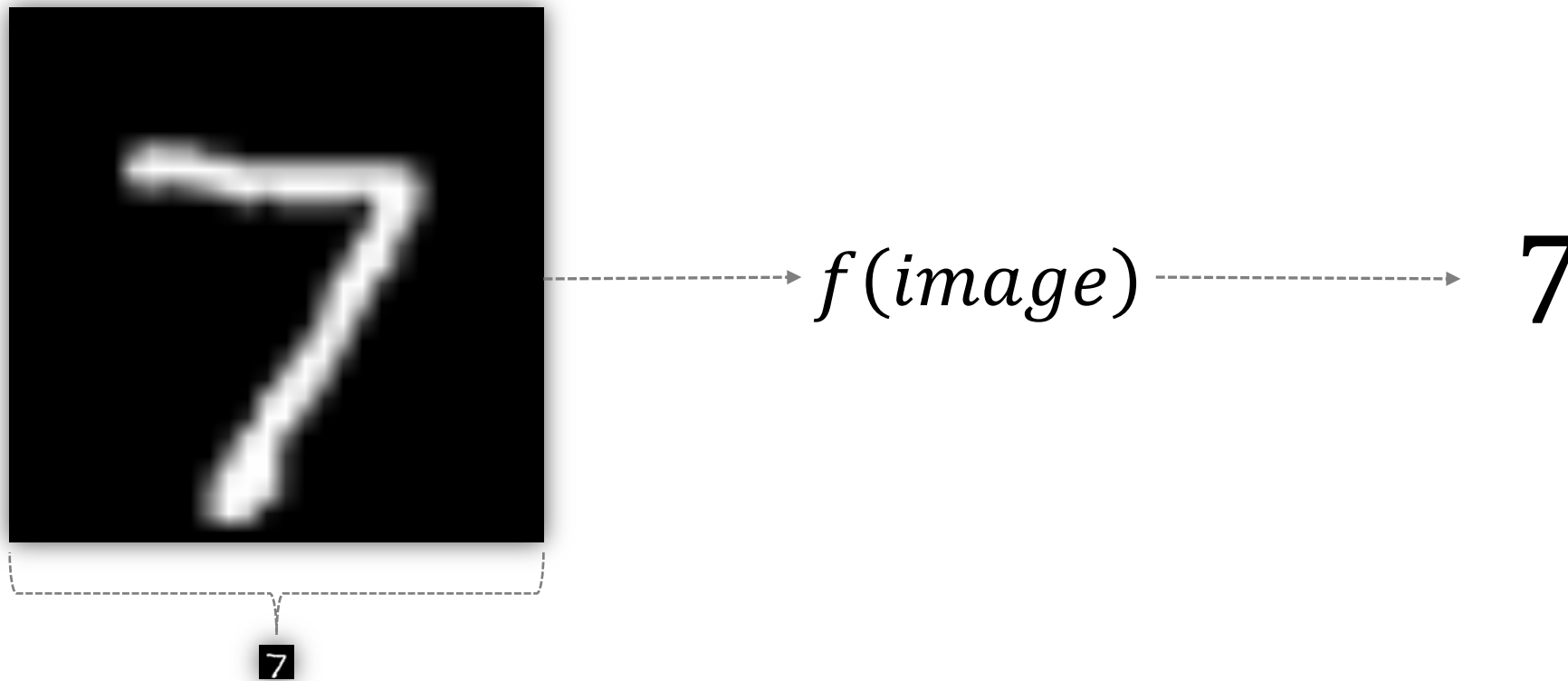


# Agenda

1. Concepts: Fundamentals
  - i. Supervised learning
  - ii. Artificial Neural Networks
  - iii. Error backpropagation
2. Code example: 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. Code examples: Distributed training
  - i. MNIST classification: Epoch distributed
  - ii. Distributing training data
5. The “Deep Learning on Supercomputers” tutorial
  - i. Introduction
  - ii. Job configuration
  - iii. Running code samples on JUWELS
6. Conclusion

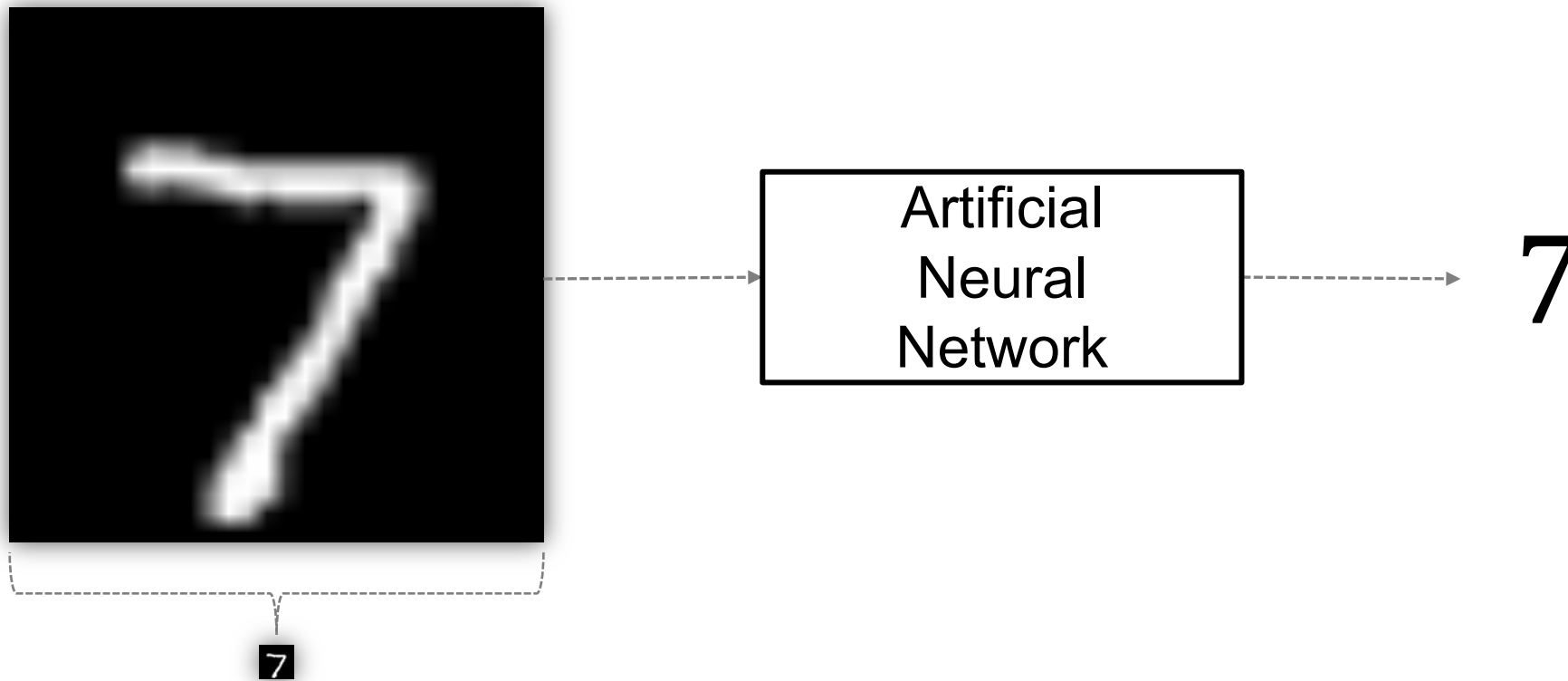
# Examples

## Handwritten digit recognition



# Examples

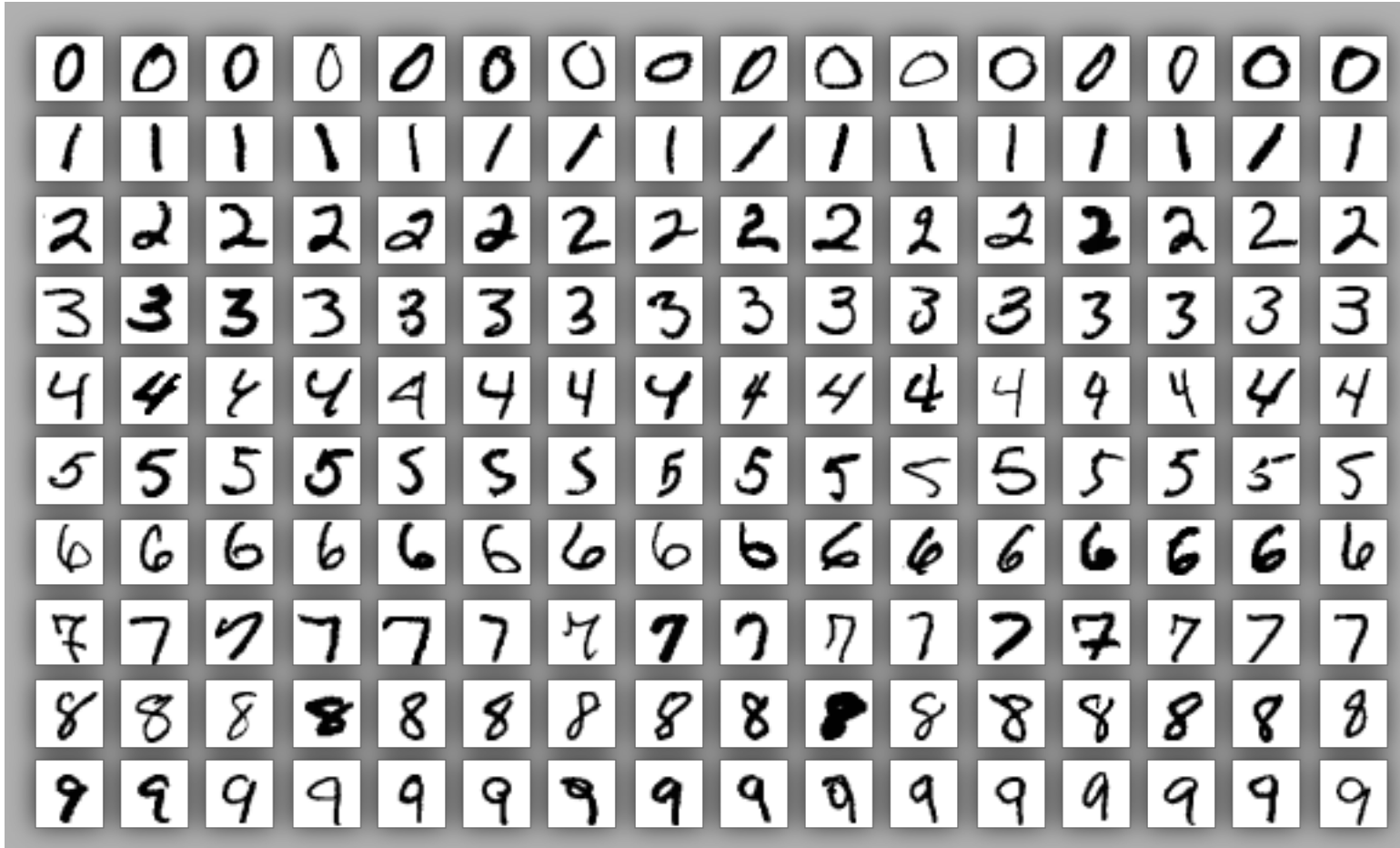
## Handwritten digit recognition





# Examples

Modified National Institute of Standards and Technology (MNIST) database

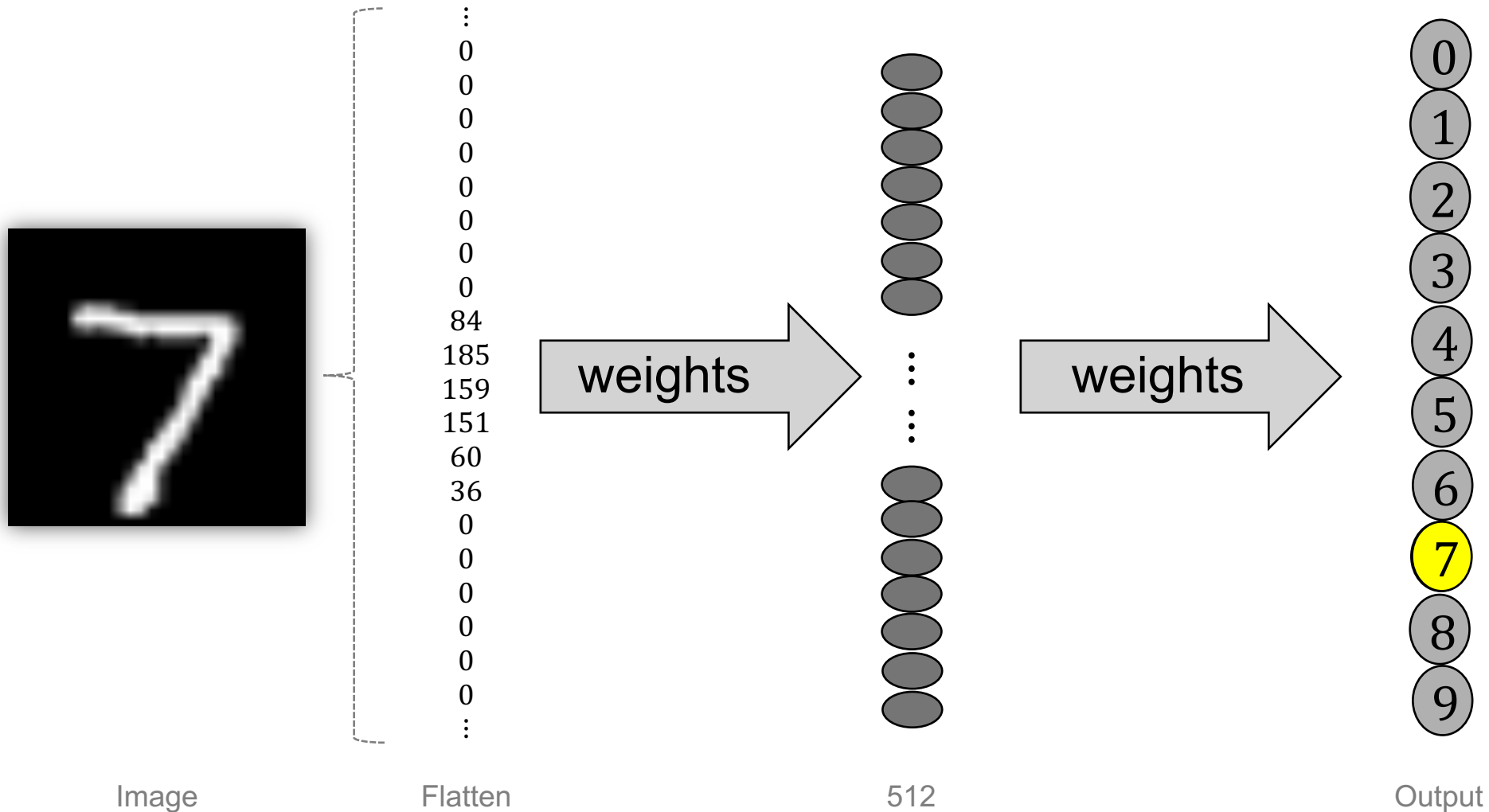


- Image dimensions: 28×28
- Each pixel  $p \in [0, 255]$
- 60,000 training examples
- 10,000 test examples

Source: Modified version of [this](#). [License](#).

# Examples

## A basic network for classification



# 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]}')
```

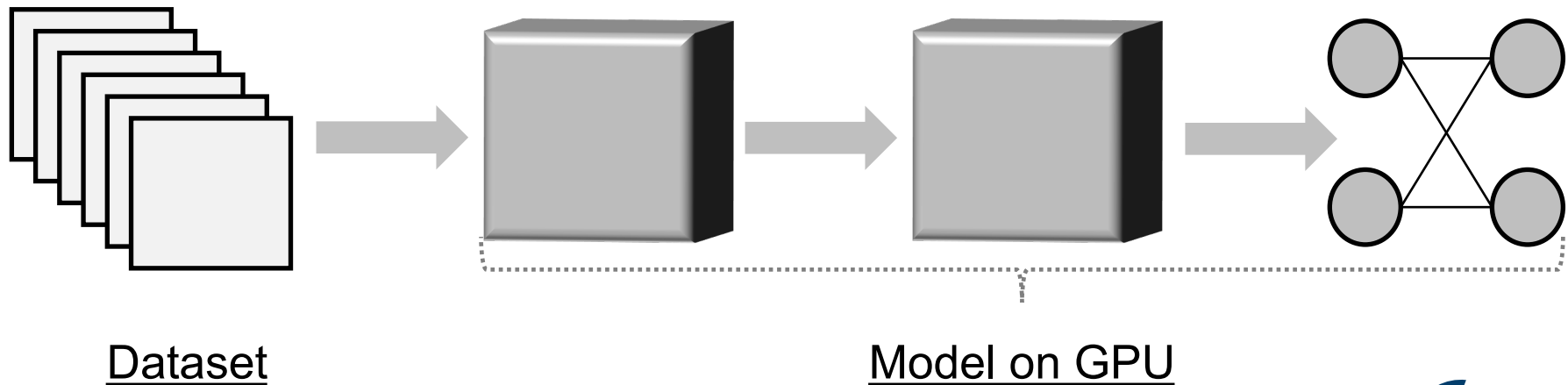
# Agenda

1. Concepts: Fundamentals
  - i. Supervised learning
  - ii. Artificial Neural Networks
  - iii. Error backpropagation
2. Code example: 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. Code examples: Distributed training
  - i. MNIST classification: Epoch distributed
  - ii. Distributing training data
5. The “Deep Learning on Supercomputers” tutorial
  - i. Introduction
  - ii. Job configuration
  - iii. Running code samples on JUWELS
6. Conclusion

# Distributed training

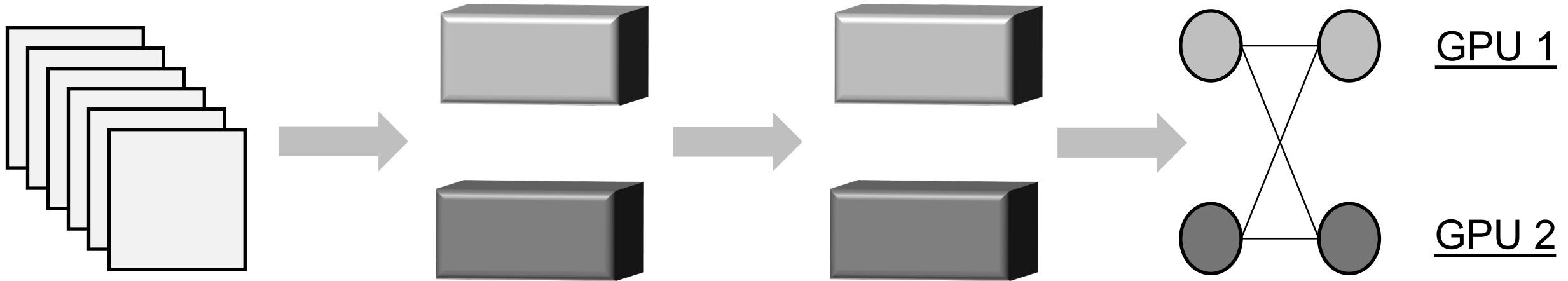
## Motivation

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



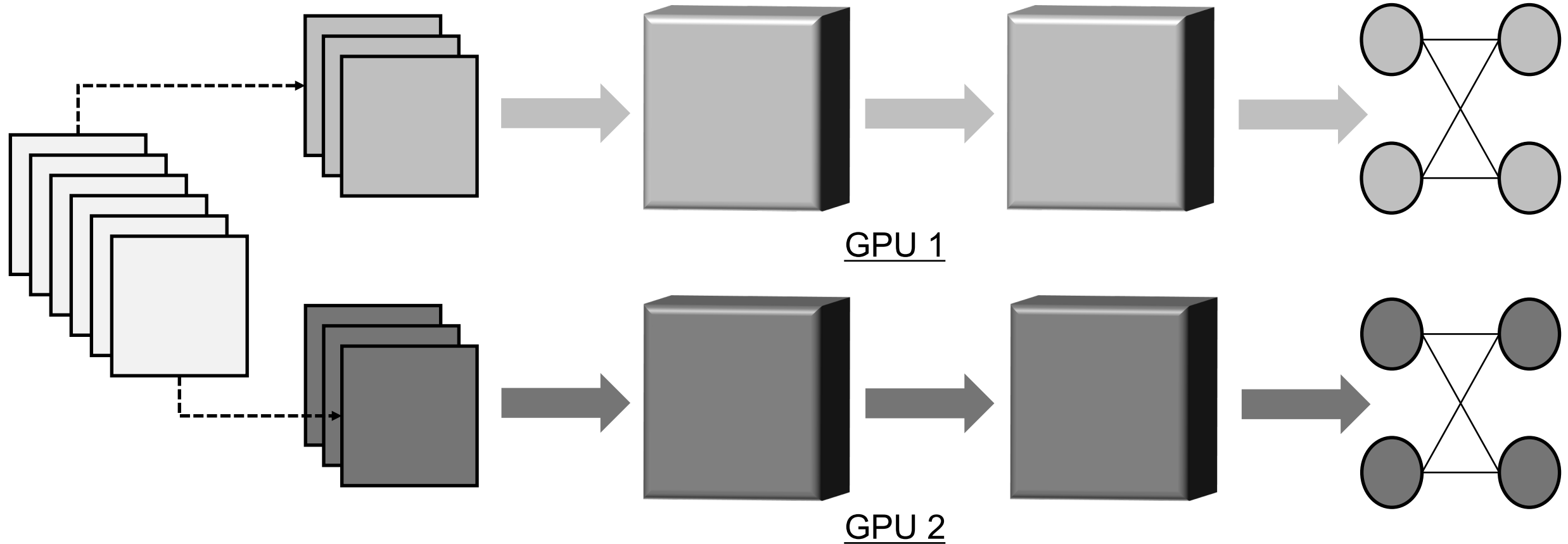
- Other methods
  - Layer Pipelining
  - Hybrid Parallelism

Ben-Nun, T., and Hoefler, T. Demystifying Parallel and Distributed Deep Learning: An In-Depth Concurrency Analysis. CoRR abs/1802.09941 (Feb. 2018)



# Distributed training

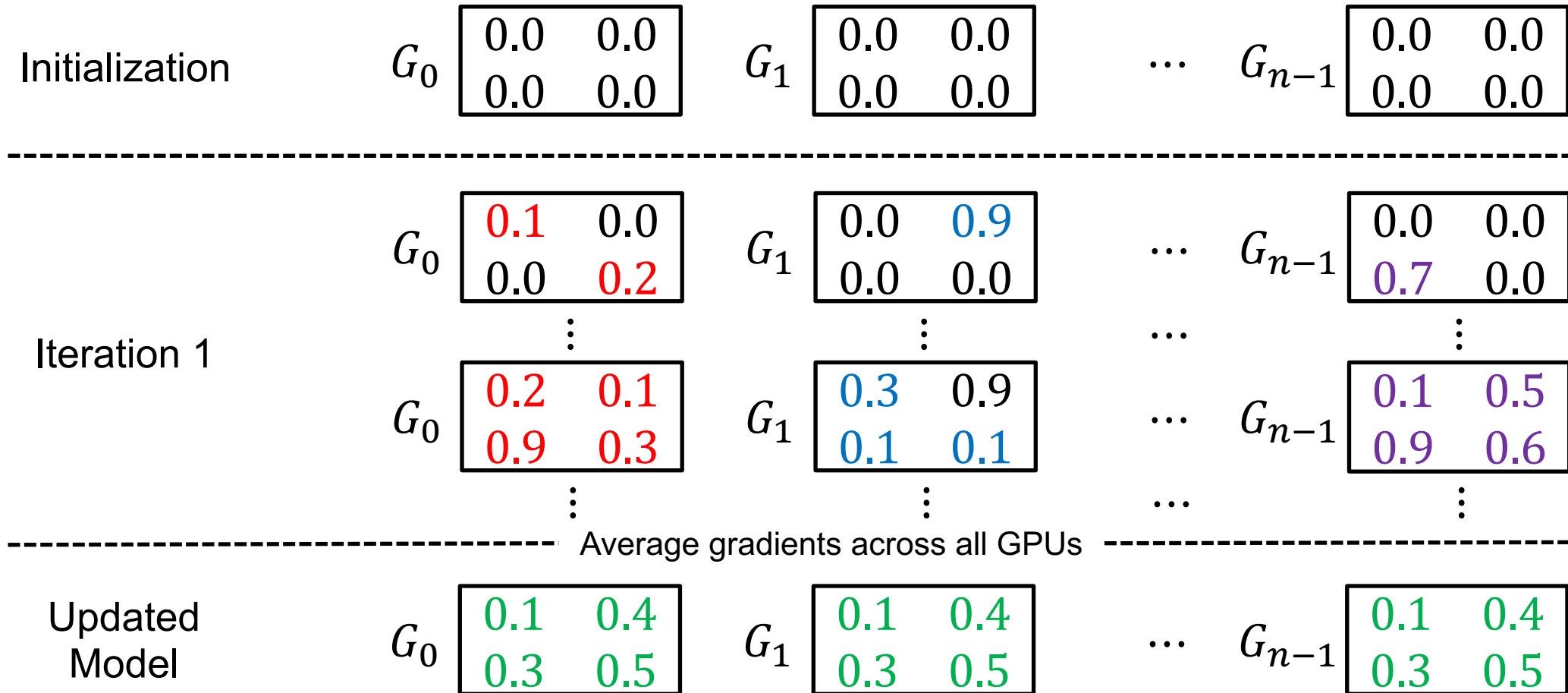
## Data Parallel



Ben-Nun, T., and Hoefler, T. Demystifying Parallel and Distributed Deep Learning: An In-Depth Concurrency Analysis. CoRR abs/1802.09941 (Feb. 2018)

# Distributed training

## Gradient averaging in data parallel training



# Agenda

1. Concepts: Fundamentals
  - 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. Code examples: Distributed training
  - i. MNIST classification: Epoch distributed
  - ii. Distributing training data
5. The “Deep Learning on Supercomputers” tutorial
  - i. Introduction
  - ii. Job configuration
  - iii. Running code samples on JUWELS
6. Conclusion

# Back to distributed training: Examples

## Distributed training with Horovod: Code (1)

```
1. import tensorflow as tf
```

Single GPU

```
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))
```

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
```

Single GPU

```
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()))
```

Distributed

# 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  
)
```

Single GPU

```
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  
)
```

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
```



# Back to distributed training: Examples

## Input data distribution for Horovod: Code snippet

```
1. mnist = tf.keras.datasets.mnist  
2. (x_train, y_train), (x_test, y_test) = mnist.load_data()
```

# Back to distributed training: Examples

## Input data distribution for Horovod: Code snippet

```
1. from mpi4py import MPI
2. from hpc4neuro.distribution import DataDistributor
3.
4. @DataDistributor(MPI.COMM_WORLD, shutdown_on_error=True)
5. def get_filenames(path):
6.     absolute_path = os.path.join(os.path.abspath(f'{path}/x'))
7.     return os.listdir(absolute_path)
8. ...
9.
10. data_dir = 'data/mnist/partitioned'
11. train_filenames = get_filenames(f'{data_dir}/train')
12.
13. x_train, y_train = load_dataset(f'{data_dir}/train', train_filenames)
14.
15. x_train = x_train / 255.0
16. ...
```

```
1. mnist = tf.keras.datasets.mnist
2. (x_train, y_train), (x_test, y_test) = mnist.load_data()
```

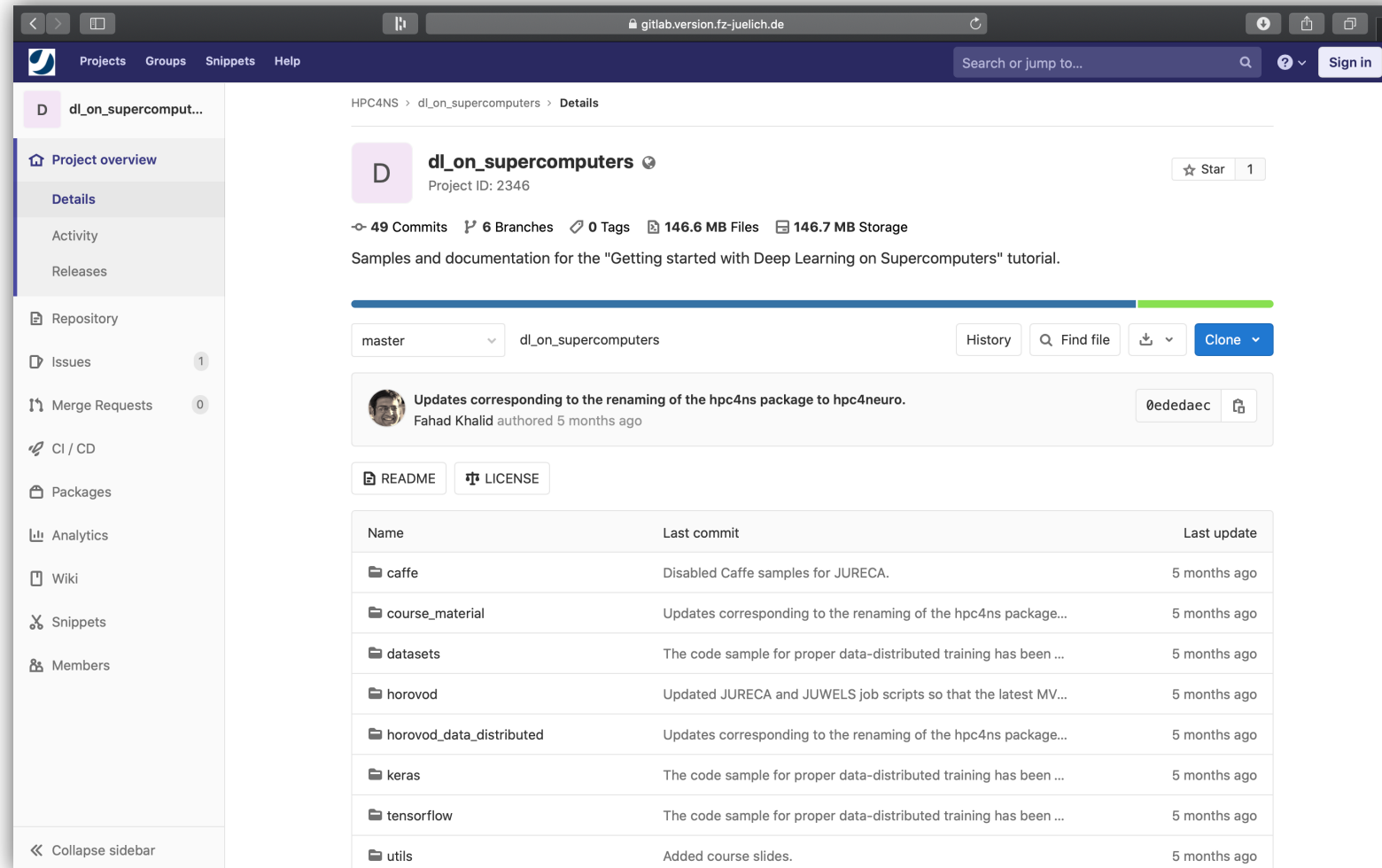
dl\_on\_supercomputers/horovod\_data\_distributed/mnist\_data\_distributed.py

# Agenda

1. Concepts: Fundamentals
  - 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. Code examples: Distributed training
  - i. MNIST classification: Epoch distributed
  - ii. Distributing training data
5. The “Deep Learning on Supercomputers” tutorial
  - i. Introduction
  - ii. Job configuration
  - iii. Running code samples on JUWELS
6. Conclusion

# Getting started with Deep learning on Supercomputers

## The tutorial



The screenshot shows the GitLab web interface for the project 'dl\_on\_supercomputers' on the domain 'gitlab.version.fz-juelich.de'. The left sidebar contains navigation links: Project overview, Details (selected), Activity, Releases, Repository, Issues (1), Merge Requests (0), CI / CD, Packages, Analytics, Wiki, Snippets, and Members. The main content area displays the project details, including the project ID (2346), statistics (49 Commits, 6 Branches, 0 Tags, 146.6 MB Files, 146.7 MB Storage), and a description: 'Samples and documentation for the "Getting started with Deep Learning on Supercomputers" tutorial.' Below this is a commit history table with columns 'Name', 'Last commit', and 'Last update'. The table lists several folders and their corresponding commit messages and dates.

Name	Last commit	Last update
caffe	Disabled Caffe samples for JURECA.	5 months ago
course_material	Updates corresponding to the renaming of the hpc4ns package...	5 months ago
datasets	The code sample for proper data-distributed training has been ...	5 months ago
horovod	Updated JURECA and JUWELS job scripts so that the latest MV...	5 months ago
horovod_data_distributed	Updates corresponding to the renaming of the hpc4ns package...	5 months ago
keras	The code sample for proper data-distributed training has been ...	5 months ago
tensorflow	The code sample for proper data-distributed training has been ...	5 months ago
utils	Added course slides.	5 months ago

[https://gitlab.version.fz-juelich.de/hpc4ns/dl\\_on\\_supercomputers](https://gitlab.version.fz-juelich.de/hpc4ns/dl_on_supercomputers)

# Getting started with Deep learning on Supercomputers

## The tutorial

Name
☞ caffe
☞ course_material
☞ datasets
☞ horovod
☞ horovod_data_distributed
☞ keras
☞ tensorflow
☞ utils
🔴 .gitattributes
🔴 .gitignore
📄 LICENSE
📄 NOTICE
📖 README.md
📄 requirements.txt

## 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 and JUWELS
  - 4.2. JURON
5. Cloning the repository
  - 5.1. JURECA and JUWELS
  - 5.2. JURON
6. Running a sample
  - 6.1. JURECA and JUWELS
  - 6.2. JURON
7. Python 2 support
8. Distributed training
9. Credits

[https://gitlab.version.fz-juelich.de/hpc4ns/dl\\_on\\_supercomputers](https://gitlab.version.fz-juelich.de/hpc4ns/dl_on_supercomputers)

# 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.er
#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
```

# Getting started with Deep learning on Supercomputers

## The tutorial: Starting and monitoring the training job

```
[username@juwels04 juwels]$ cd dl_on_supercomputers/horovod/keras
[username@juwels04 keras]$ sbatch submit_job_juwels.sh
Submitted batch job 1885056
[username@juwels04 keras]$ squeue -u username
      JOBID PARTITION     NAME     USER  ST       TIME  NODES NODELIST(REASON)
      1885056 develgpu  HOROVOD_  username  CF        0:07        2 jwc09n[006,009]
[username@juwels04 keras]$ tail -f output_1885056.out
Using /p/project/cslns/username/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
```



# Agenda

1. Concepts: Fundamentals
  - 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. Code examples: Distributed training
  - i. MNIST classification: Epoch distributed
  - ii. Distributing training data
5. The “Deep Learning on Supercomputers” tutorial
  - i. Introduction
  - ii. Job configuration
  - iii. Running code samples on JUWELS
6. 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](mailto:sc@fz-juelich.de)
- Tutorial and hpc4neuro: [slns@fz-juelich.de](mailto: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](https://gitlab.version.fz-juelich.de/hpc4ns/dl_on_supercomputers)

### 2. The hpc4neuro Python library

- <https://gitlab.version.fz-juelich.de/hpc4ns/hpc4neuro>

### 3. Horovod

- <https://github.com/horovod/horovod>

### 4. The MPI tutorial

- <https://mpitutorial.com/>

### 5. MPI4Py

- <https://mpi4py.readthedocs.io/en/stable/>