Leibniz-Rechenzentrum
der Bayerischen Akademie der Wissenschaften

Introduction to the LRZ AI Infrastructure

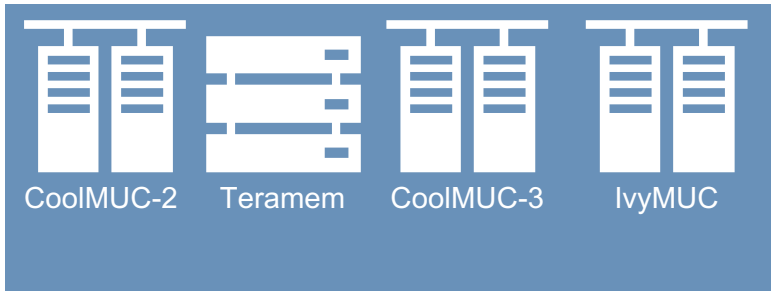
08.10.2020 | PD Dr. Juan J. Durillo

Agenda

- Introduction to the LRZ AI System
- Introduction to Machine Learning Training
- Horovod: an Example of Distributed Learning
- Wrap-Up

LRZ Systems Offer

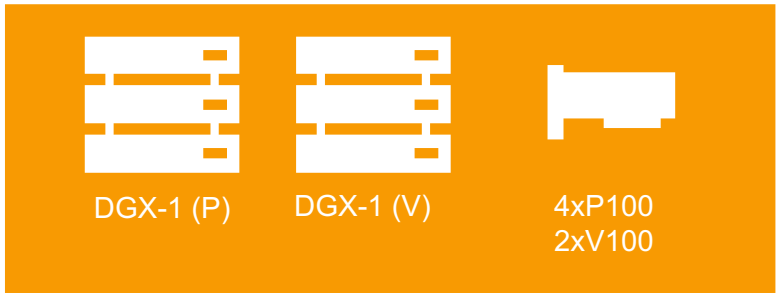
DSS
(Data Science Storage)



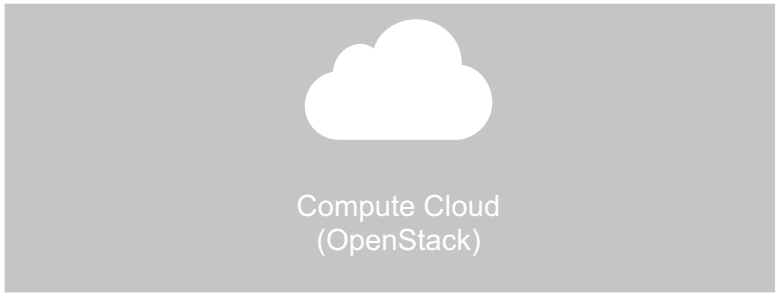
lxlogin8.lrz.de

lxlogin[1-4].lrz.de lxlogin10.lrz.de

https://www.rstudio.lrz.de



https://datalab.srv.lrz.de



https://cc.lrz.de

AI Systems

(Multi-purpose cluster systems might be used for AI workloads as well, but have a different focus)

Resources Overview

	DGX-1 P100 Architecture	DGX-1 V100 Architecture	V100 GPU Nodes	
Number of Nodes	1	1	1	Compute Cloud
Cores per node	80	80	40	Flavor dep.
Memory per node	512 GB DDR4	512 GB DDR4	724 GB DDR4	Flavor dep.
GPUs per node	8 Nvidia Tesla P100	8 Nvidia Tesla V100	2 Nvidia Tesla V100	Flavor dep.
Memory per GPU	16 GB	16 GB	16 GB	
CUDA / Tensor Cores per GPU	3584 / --	5120 / 640	5120 / 640	
SLURM Partition	dgx	?	gpu	

Hands on – Accessing LRZ System

- Who can access the system?
  - Users with a valid Linux Cluster account,
  - who explicitly request access explaining intended used (what? how?)
- A single login node datalab2.srv.lrz.de accessible via ssh

```
ssh -Y datalab2.srv.lrz.de -l xxyyyzz
```

- From the login node, jobs are submitted to the hardware described at the beginning of this course using SLURM
- A couple of handy SLURM commands

```
$ squeue
```

```
$ sinfo
```

```
$ salloc /
scancel
```

```
$ srun
```

LRZ AI System – A container based solution

- Containerized applications with `enroot`, a rootless container runtime by Nvidia
- Slightly different workflow than with `Docker`

#1 Import a container from a valid repository
```
$ enroot import docker://ubuntu
```

#2 Create an enroot container
```
$ enroot create ubuntu.sqsh
```

#3 Start/run an application within the container
```
$ enroot start ubuntu
```

- It should be noticed than the workflow in the AI System consist in submitting jobs that run containerized within an enroot defined container

LRZ AI System – On running Interactive Containerized Applications

- Get resources allocated

```
$ salloc -p dgx --ntasks=8 --gres=gpu:8
```

Indicate that access to the GPU resources is required

resource queue

horovod will start n tasks in the machine (one per GPU probably)

- Submit containerized job

mounting outside folders inside the container

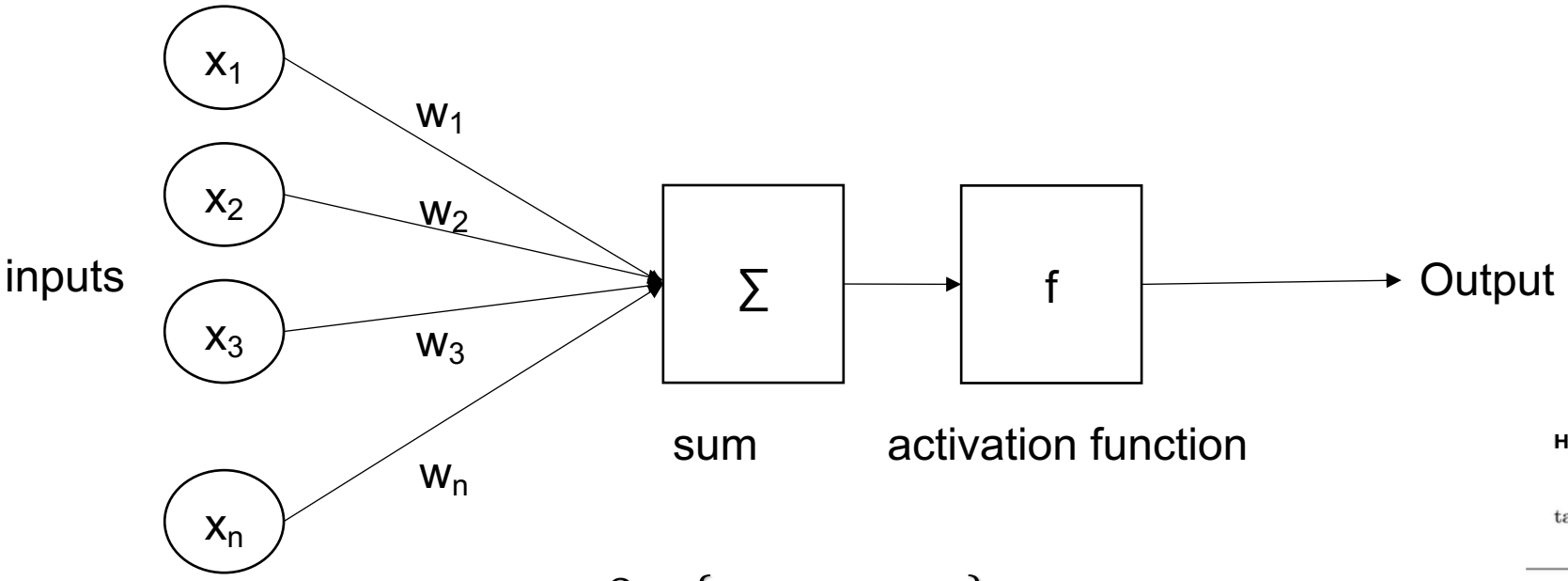
the container previously created

```
$ srun --pty enroot start --mount ./data-test:/mnt/data-test ubuntu bash
```

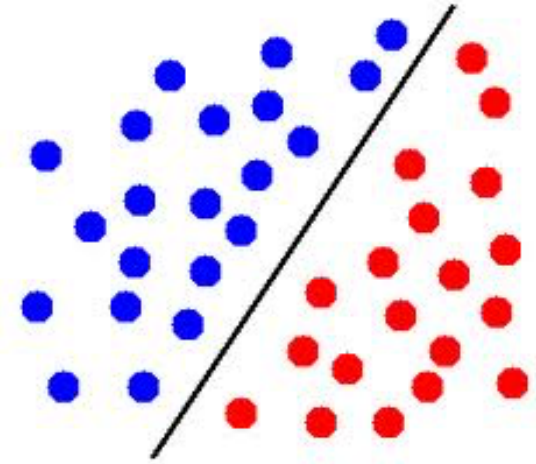
- Meet the pyxis plugin: container creating and job submission in a single step

```
$ srun --container-mounts=./data-test:/mnt/data-test --container-name=horovod
--container-image='horovod/horovod+0.16.4-tf1.12.0-torch1.1.0-mxnet1.4.1-
py3.5' bash
```

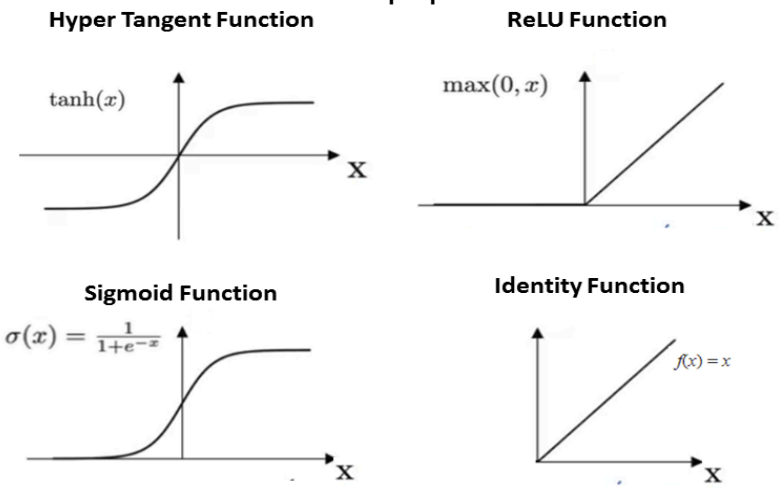
Perceptron – Artificial Neuron



$$\Theta = \{w_1, w_2 \ldots, w_n\}$$

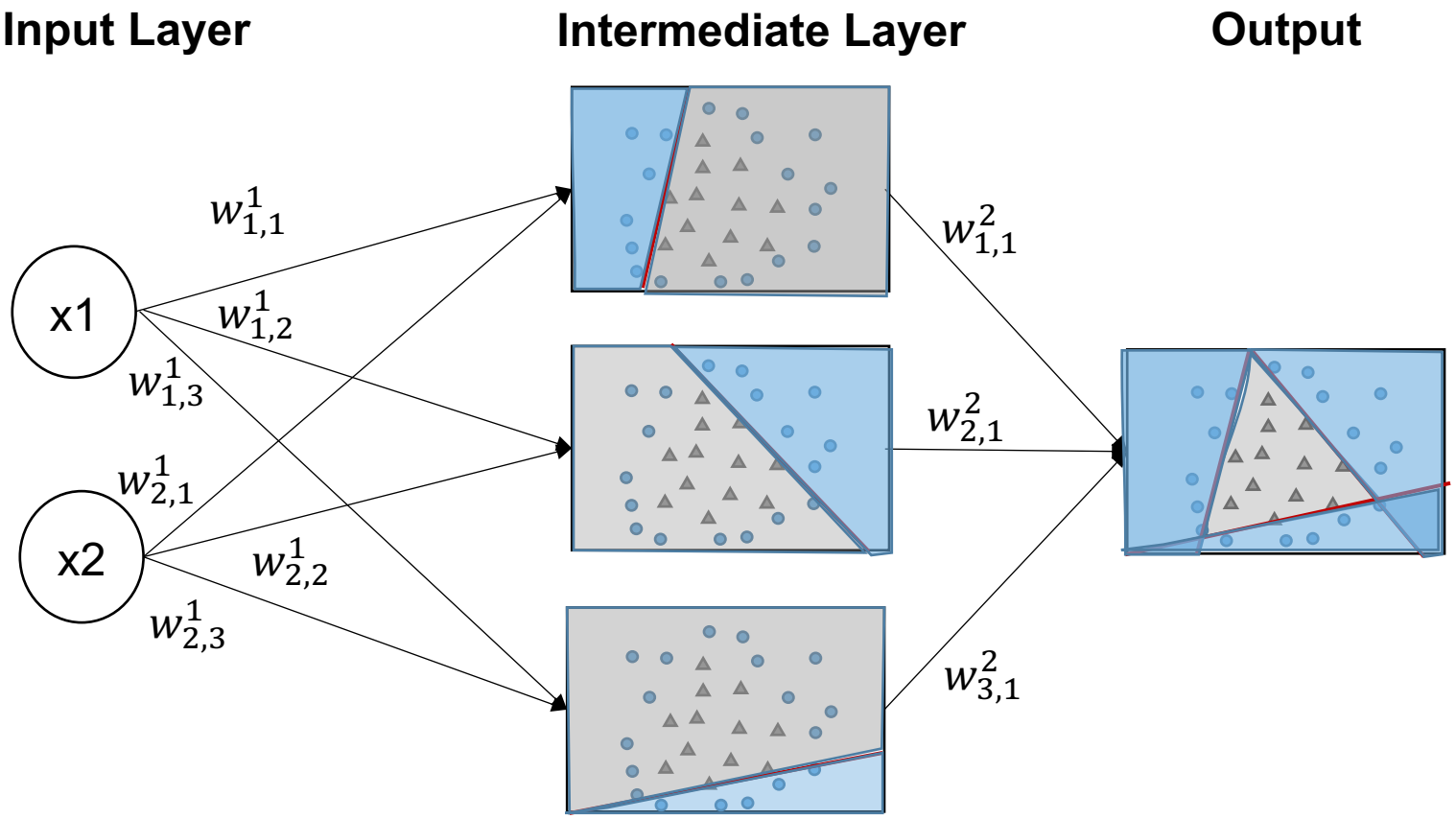


most popular activation functions



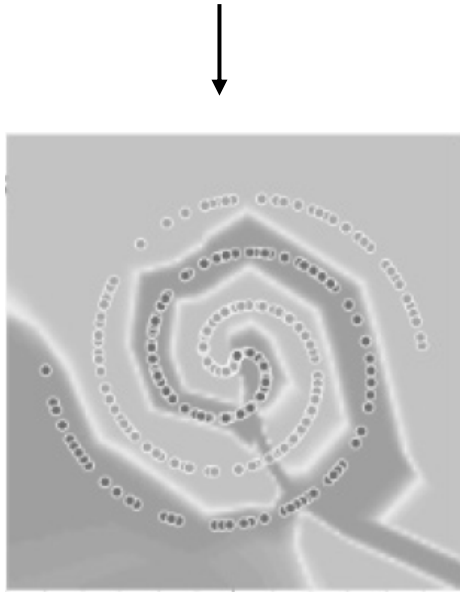
Single artificial neurons work well for linearly separable datasets (indeed output is the activation effect on a linear combination of the input)

Neural Network



$$\Theta = \{w^1_{1,1}, w^1_{1,2}, w^1_{1,3}, w^1_{2,1}, w^1_{2,2}, w^1_{2,3}, w^2_{1,1}, w^2_{2,1}, w^2_{2,3}\}$$

- Even when the data is not linearly separable



(Supervised) Learning

- Data domain $\mathcal{Z}$: $\mathcal{X} \times \mathcal{Y}$

$\mathcal{X}$ → domain of the input data

$\mathcal{Y}$ → set of labels (knowledge)

$\mathcal{X}$: 32 x 32 color images

$\mathcal{Y}$: labels

{truck, car, horse, bird, boat}

Example (CIFAR10 dataset)

- Data Distribution is a probability distribution over a data domain
- Training set $z_1, \ldots, z_n$ from $\mathcal{Z}$ assumed to be drawn from the Data Distribution D
- Validation set $v_1, \ldots, v_m$ from $\mathcal{Z}$ also assumed to be drawn from D
- A machine learning model is a function that given a set of parameters Θ and z from $\mathcal{Z}$ produces a prediction
- The prediction quality is measured by a differentiable non-negative scalar-valued loss function, that we denote $\ell(\Theta; z)$

(Supervised) Learning

- Given Θ we can define the expected loss as: $L(\Theta) = \mathbb{E}_{z\sim D}[\ell(\Theta; z)]$
- Given D, ℓ, and a model with parameter set Θ, we can define learning as:
  "The task of finding parameters Θ that achieve low values of the expected loss, while we are given access to only n training examples"
- The mentioned task before is commonly referred to as *training*
- Empirical average loss given a subset of the training data set S(z_1, …, z_n) as:

$$\hat{L}(\Theta) = \frac{1}{n}\sum_{t=1}^{n}[\ell(\Theta; z_t)]$$

- Usually a proxy function, easier to understand by humans, is used for describing how well the training is performed (e.g., accuracy)

Introduction to Learning

- The dominant algorithms for training neural networks are based on mini-batch stochastic gradient descent (SGD)

- Given an initial point Θ_0 SGD attempt to decrease $\hat{L}$ via the sequence of iterates

$$\Theta_t \leftarrow \Theta_{t-1} - n_t g(\Theta_{t-1}; B_t)$$

$$g(\Theta; B) = \frac{1}{|B|} \sum_{z \epsilon B} \nabla \ell(\Theta; z)$$

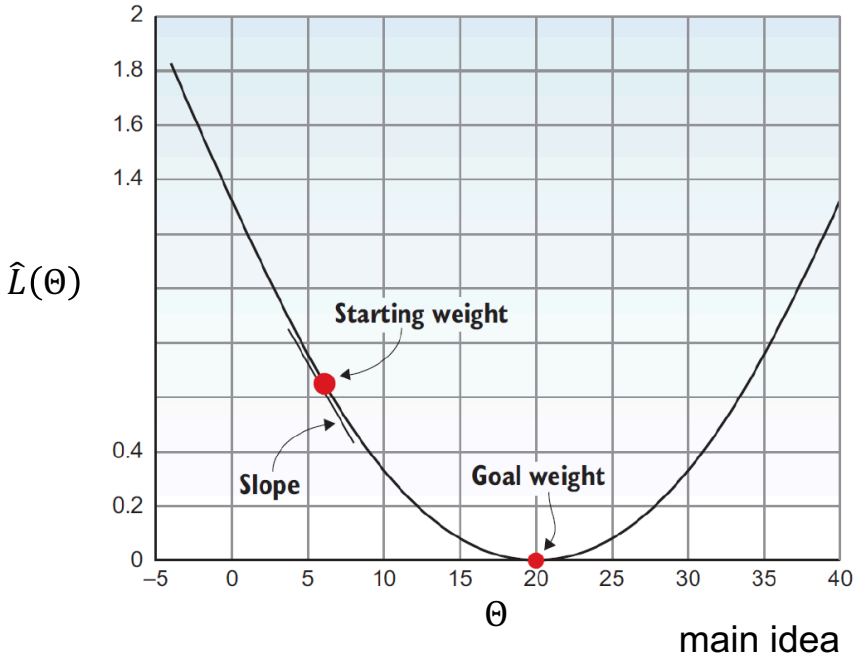
Definitions

B_t: random subset of training examples

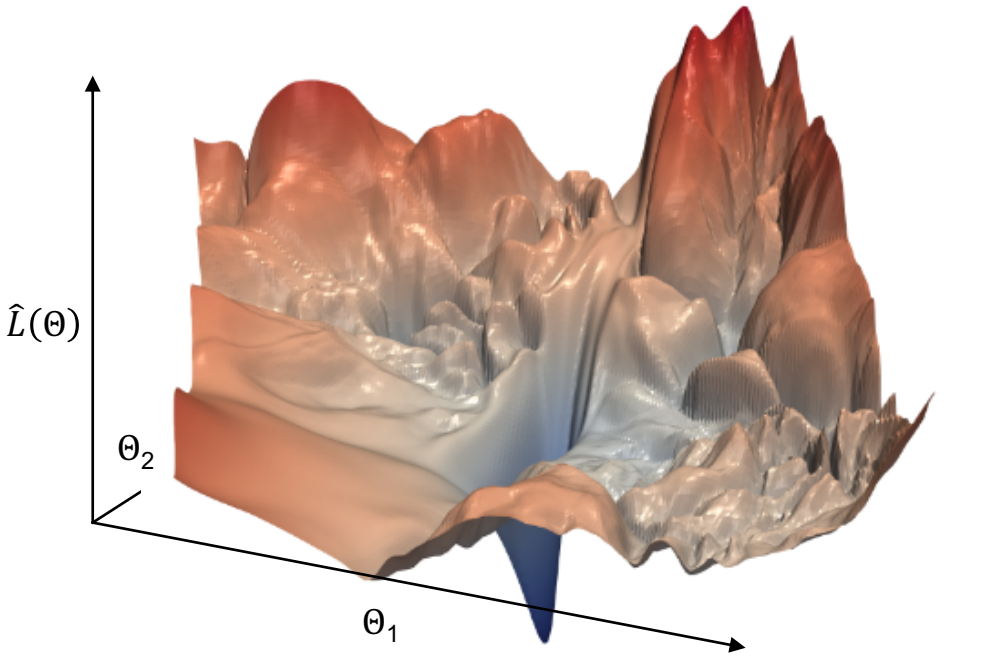
n_t: positive scalar (learning rate)

epoch: update the weights after going over all training set

Training Neural Networks



main idea



how the surface looks like in reality

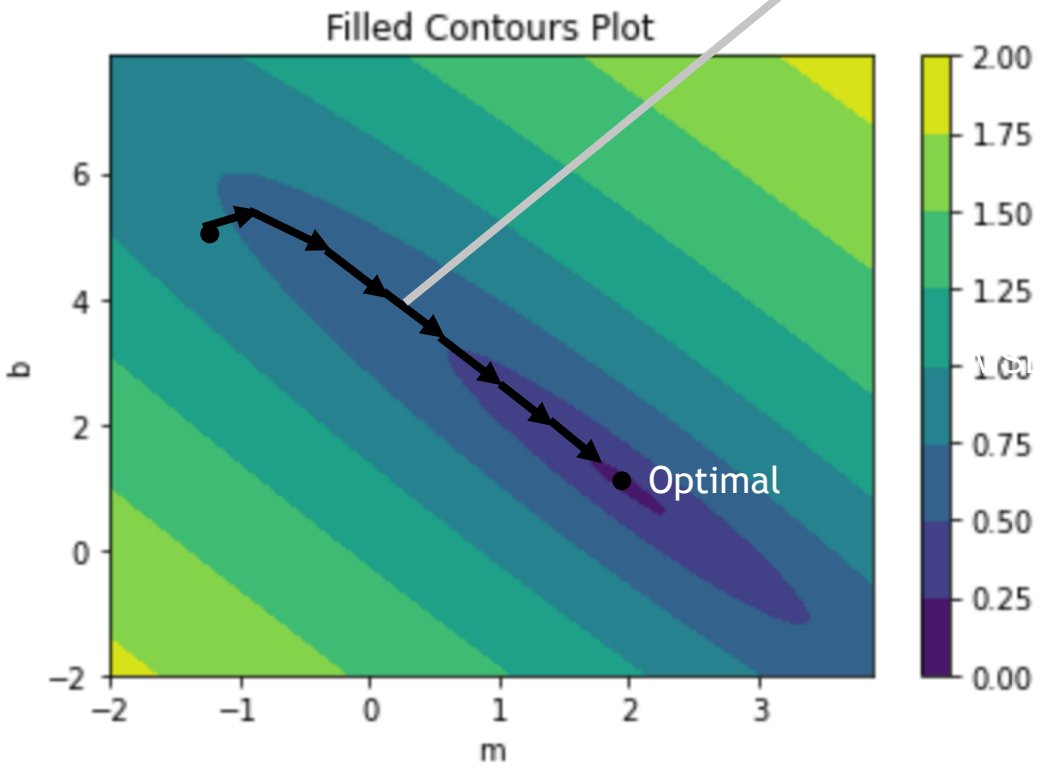
Batch

Stochastic Gradient Descent

$$\Theta_t \leftarrow \Theta_{t-1} - n_t g(\Theta_{t-1}; B_t)$$

$$g(\Theta; B) = \frac{1}{|B|} \sum_{z \epsilon B} \nabla \ell(\Theta; z)$$

Visualizing the training process



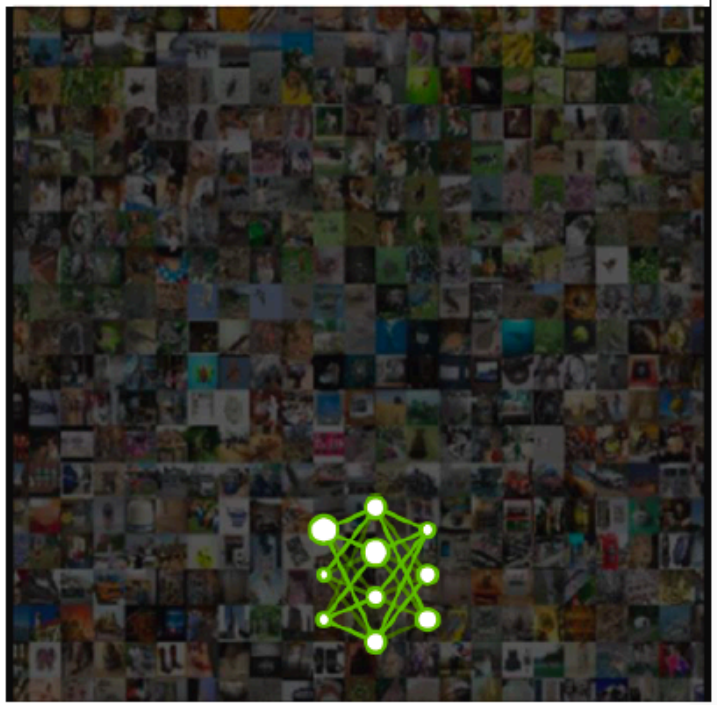
Training steps: every time the gradient is updated

What influences these steps:

- Batch size
- Normalization
- Optimizer
- Learning rate
- Loss function

Models of Increasing complexity

7 Exaflops
60 Million Parameters

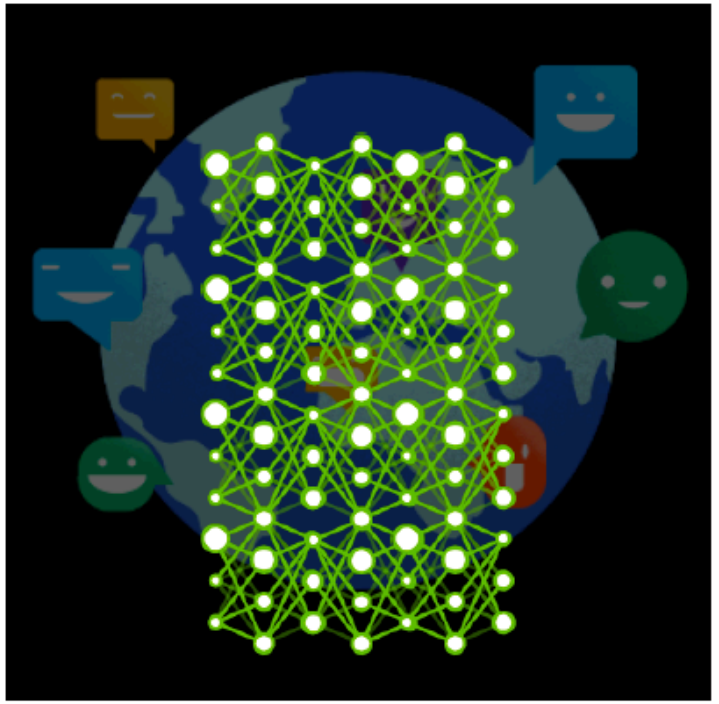


2015 - Microsoft ResNet
Superhuman Image Recognition

20 Exaflops
300 Million Parameters

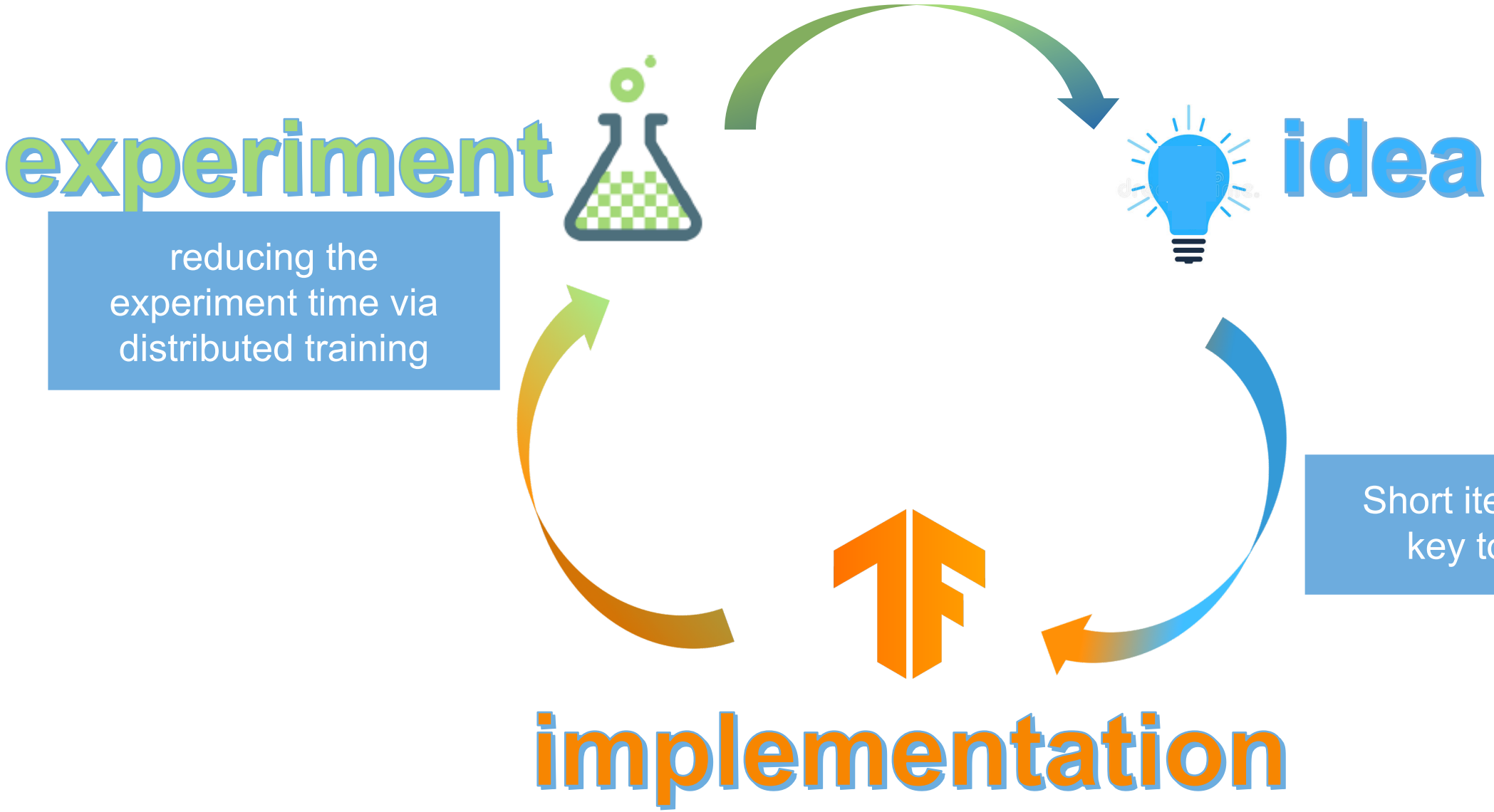
2016 - Baidu Deep Speech 2
Superhuman Voice Recognition

100 Exaflops
8700 Million Parameters



2017 - Google Neural Machine Translation
Near Human Language Translation

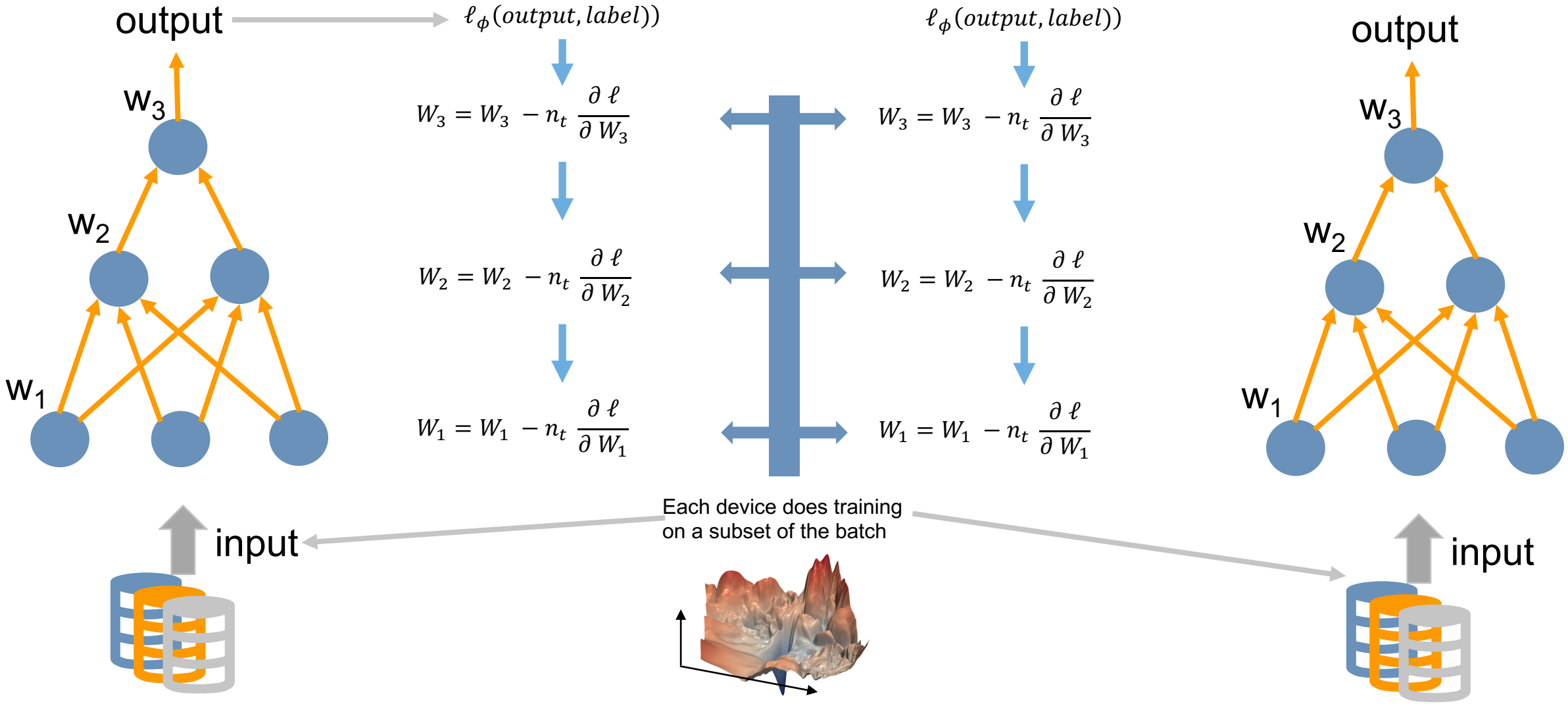
Experimental Science Require Short Iteration Times



reducing the experiment time via distributed training

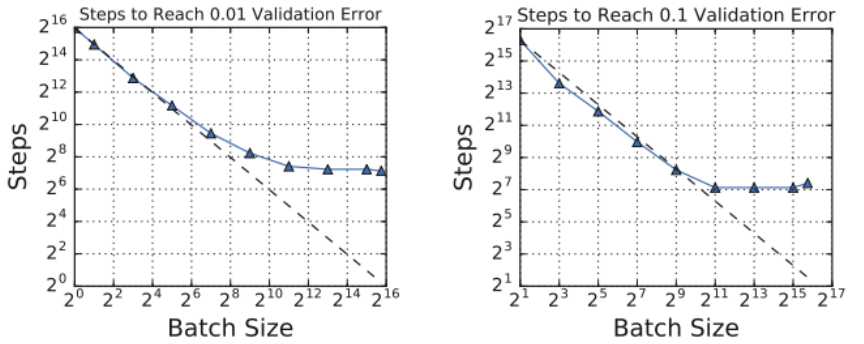
Short iteration times key to success

Data Parallelism Training

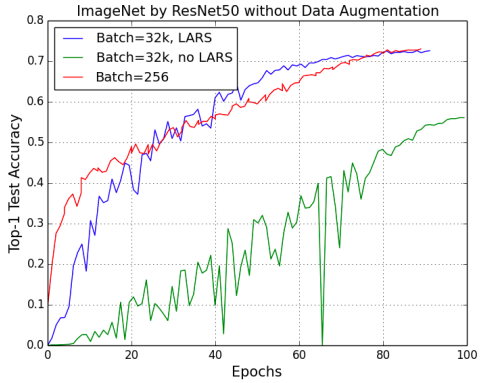


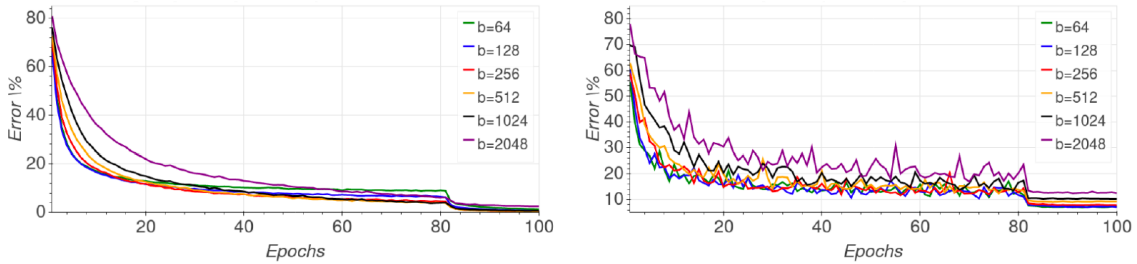
Training with large Batch Sizes

- There are limits

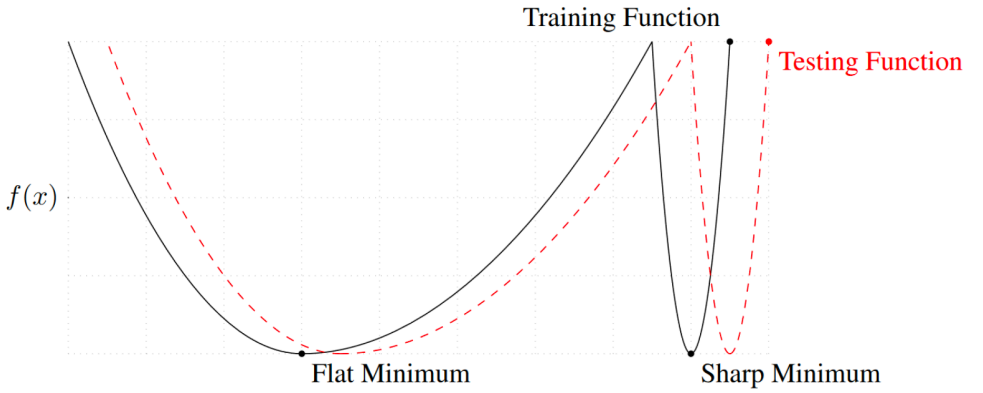


(a) Simple CNN on MNIST (b) Simple CNN on Fashion MNIST





(a) Training error (b) Validation error



Horovod

- Distributed opensource deep learning training framework for TensorFlow, Keras, Pytorch, and Apache MXNet
- Originally developed at UBER
  - Fast. Scale up to hundreds of GPUs with upwards of 90% scaling efficiency
  - Easy. A few lines of codes.
  - Portable. Different frameworks

To run on CPUs:

```
$ pip install horovod
```

To run on GPUs with NCCL:

```
$ HOROVOD_GPU_OPERATIONS=NCCL pip install horovod
```

Horovod

#1 initialization
```
import horovod.tensorflow as hvd
hvd.init()
```

#2 pin resources
```
config = tf.ConfigProto()
config.gpu_options.visible_device_list = str(hvd.local_rank())
```

#3 distributed optimizer
```
opt = tf.train.AdagradOptimizer(0.01 * hvd.size())
opt = hvd.DistributedOptimizer(opt)
```

#4 Broadcast variables from rank 0 to all other processes during initialization
```
hooks = [hvd.BroadcastGlobalVariablesHook(0)]
```

#5 Differentiate among different workers (e.g., check pointing)
```
checkpoint_dir = '/tmp/train_logs' if hvd.rank() == 0 else None
```

A data parallel version in five steps

```
$ horovodrun -np 4 -H localhost:4 python train.py

$ horovodrun -np 16 -H
server1:4,server2:4,server3:4,server4:4 python
train.py
```

An example step by step …

Conclusions and Summary

- Introduction to the LRZ AI Resources
- LRZ AI Resources Software Stack
- Machine Learning Training
- Distributed Training Challenges
- *Horovod*: an Easy Solution for Distributed Training

Feedback…

https://survey.lrz.de/index.php/264821?lang=en