Leibniz-Rechenzentrum
der Bayerischen Akademie der Wissenschaften

Introduction to the LRZ AI Infrastructure

08.04.2021 | PD Dr. Juan J. Durillo

Agenda

LRZ Systems Offer

`lxlogin1-4.lrz.de`
`lxlogin10.lrz.de`
`lxlogin10.lrz.de`

- CoolMUC-2
- Teramem
- CoolMUC-3
- IvyMUC

Linux Cluster

`datalab2.srv.lrz.de`

- DGX-1 P100
- 6 V100 GPU Nodes
- HPE Intel Skylake + 2 NvidiaTesla P100
- DGX-1 V100

LRZ AI Systems

`cc.lrz.de`

CC (Compute Cloud)

DSS (Data Science Storage)

Tape Archive and Backup

LRZ Systems Offer

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

- CoolMUC-2
- Teramem
- CoolMUC-3
- IvyMUC

Linux Cluster

Designed and Configured for AI

- DGX-1 P100
- 6 V100 GPU Nodes
- HPE Intel Skylake + 2 NvidiaTesla P100
- DGX-1 V100

LRZ AI Systems

Flexible system that copes with almost any workload

CC (Compute Cloud)

DSS (Data Science Storage)

Tape Archive and Backup

Resources Overview

	DGX-1 P100 Architecture	DGX-1 V100 Architecture	HPE Intel Skylake + Nvidia Node	V100 GPU Nodes
Number of Nodes	1	1	1	3
Cores per node	80	80	64	40
Memory per node	512 GB DDR4	512 GB DDR4	2TB DDR4	724 GB DDR4
GPUs per node	8 Nvidia Tesla P100	8 Nvidia Tesla V100	4 Nvidia Tesla P100	2 Nvidia Tesla V100
Memory per GPU	16 GB	16 GB	16GB	16 GB
CUDA / Tensor Cores per GPU	3584 / --	5120 / 640	3584 / --	5120 / 640
SLURM Partition	dgx-1-p100	dgx-1-v100	hpe-p100	gpu-v100
DNS Name	dgx-001.srv.lrz.de	dgx-002.srv.lrz.de	p100-001.cloud.lrz.de	Gpu-00{1-3}.cloud.lrz.de

Hands on – Accessing LRZ System

- Who can access the system?
  - Users with a Linux Cluster account …
  - who explicitly request access explaining intended used (why? how?)
    - you will be invited to a DSS container that will be used as your $HOME
    - submitting a service request ticket
- 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
```

#2.1 Create an enroot container

```
$ enroot create ubuntu.sqsh
```

#2.1.1 Start/run an application within the container

```
$ enroot start ubuntu
```

#2.2 Create and start enroot container in a single step

```
$ enroot start ubuntu.sqsh
```

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

LRZ AI System – On running Interactive Containerized Applications

lrz

Executes in the login node datalab2

Executes in the allocated resource

- Get resources allocated

```
$ salloc -p dgx-1-p100 --ntasks=8 --gres=gpu:8
```

Indicate the number of GPUs access is required

SLURM partition

number of process to run

- Submit containerized job

mounting outside folders inside the container

the container image imported

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

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

```
$ srun --container-mounts=./data-test:/mnt/data-test --container-image=/home/juanjo/ubuntu.sqsh bash
```

- Start a jupyter notebook on an interactive application (the container must provide jupyter)

```
$ jupyter notebook --ip=0.0.0.0 --allow-root
```

LRZ AI System – On running Batch Containerized Applications

- Batch jobs are also possible
- Create batch script defining the job (e.g., script.sbatch)

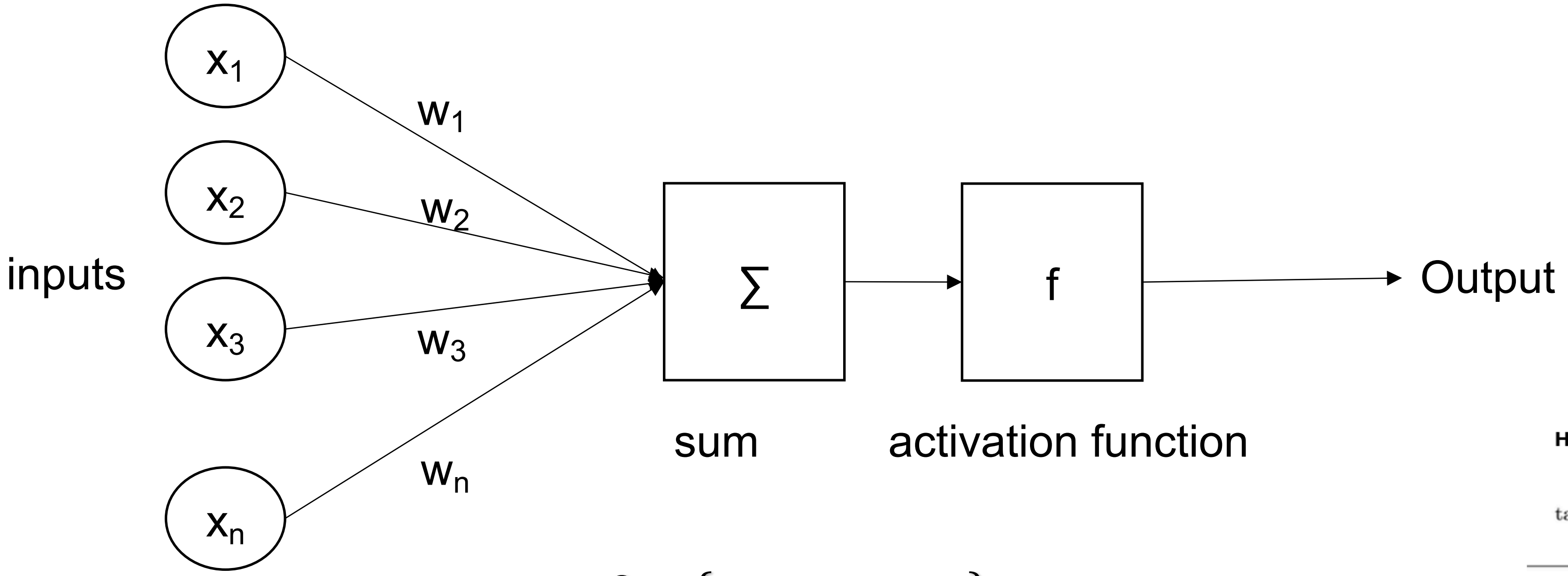
```
#!/bin/bash
#SBATCH -N 1
#SBATCH -p dgx
#SBATCH --gres=gpu:8
#SBATCH --ntasks=8
#SBATCH -o enroot_test.out
#SBATCH -e enroot_test.err

srun --container-mounts=./data-test:/mnt/data-test --container-image='horovod/horovod+0.16.4-tf1.12.0-torch1.1.0-mxnet1.4.1-py3.5' \
    python script.py --epochs 55 --batch-size 512
```

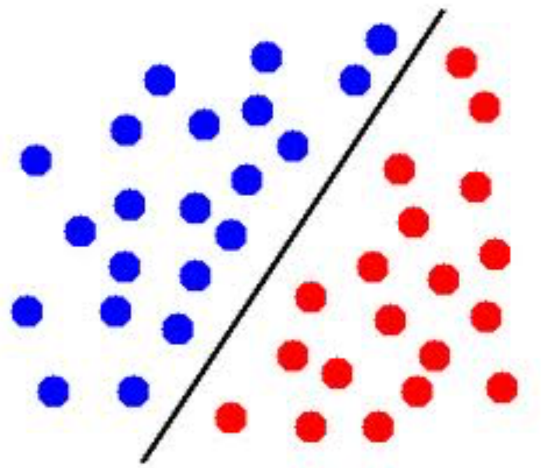
- Submit with sbatch

```
$ sbatch script.sbatch
```

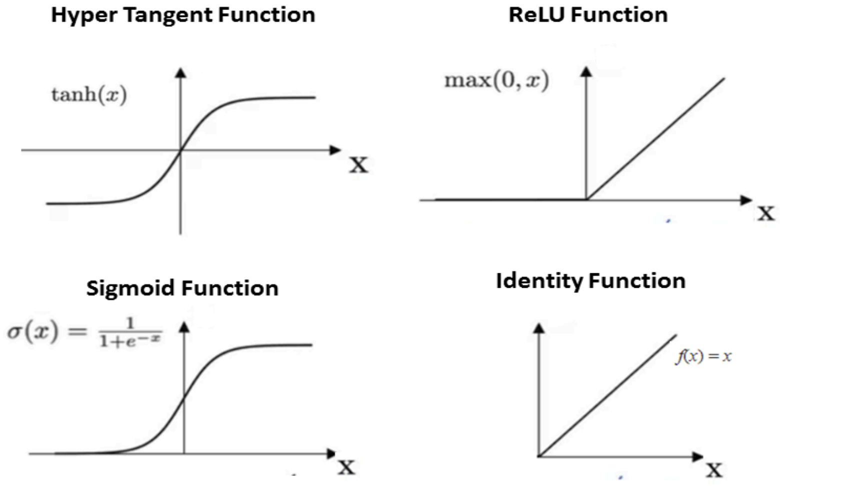
Perceptron – Artificial Neuron



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

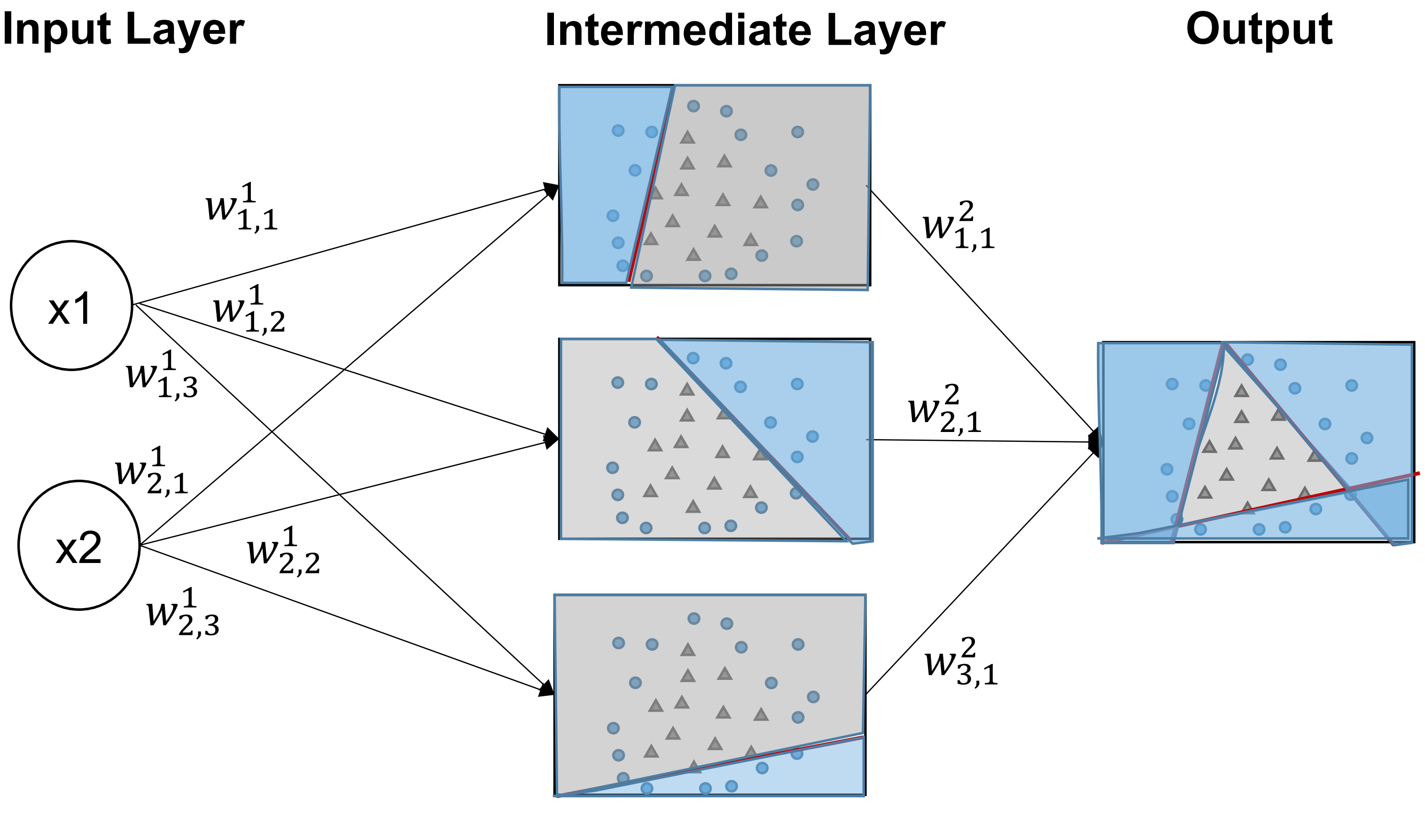


most popular activation functions



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 Z: $X \times \Upsilon$

  X → domain of the input data

  Υ → set of labels (knowledge)

X: 32 x 32 color images

Υ : 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 Z assumed to be drawn from the Data Distribution D
- Validation set $v_1, \ldots, v_m$ from 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 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

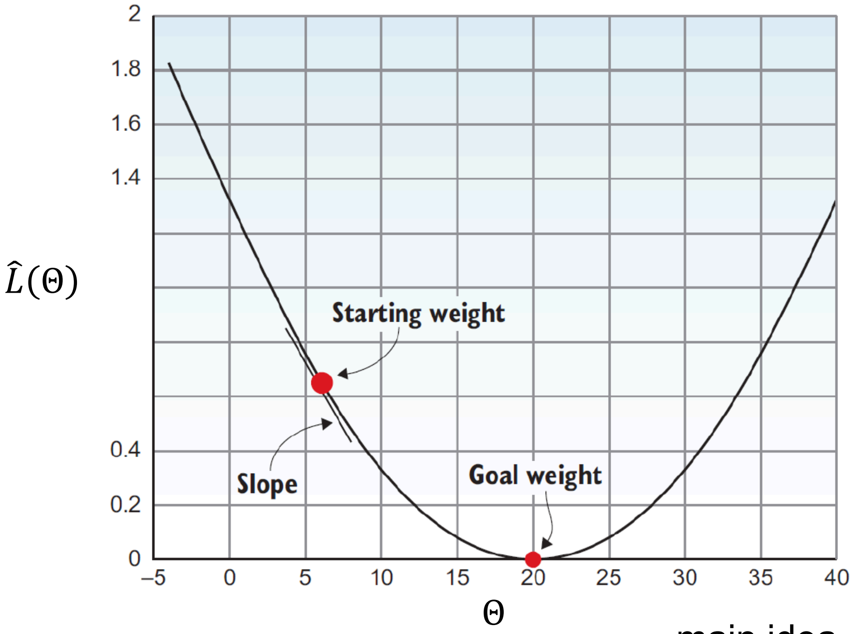
B_t: random subset of training examples

n_t: positive scalar (learning rate)

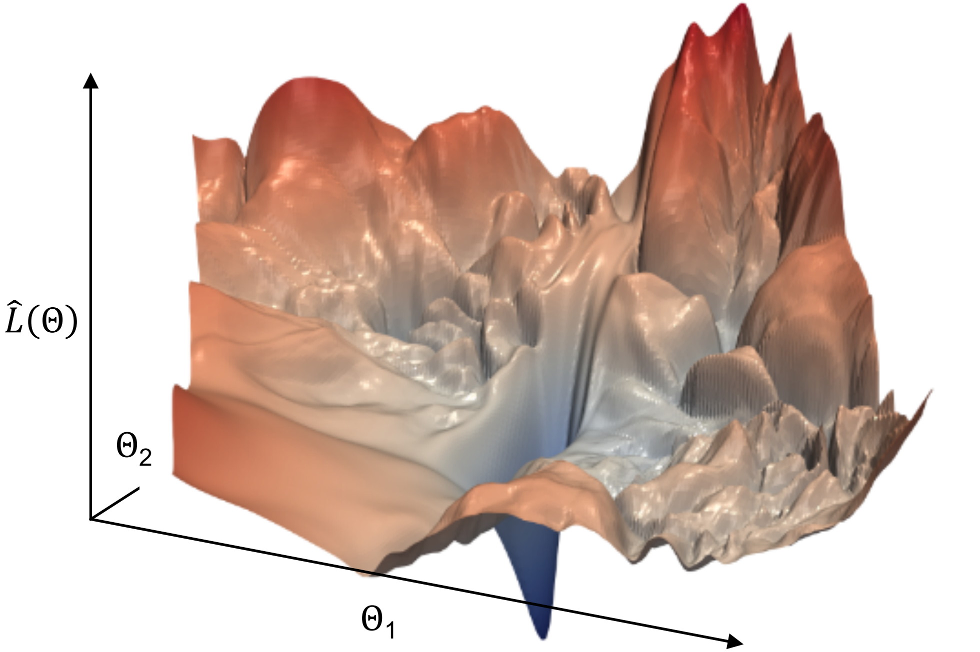
epoch: update the weights after going over all training set

Training Neural Networks

Batch



main idea



how the surface looks like in reality

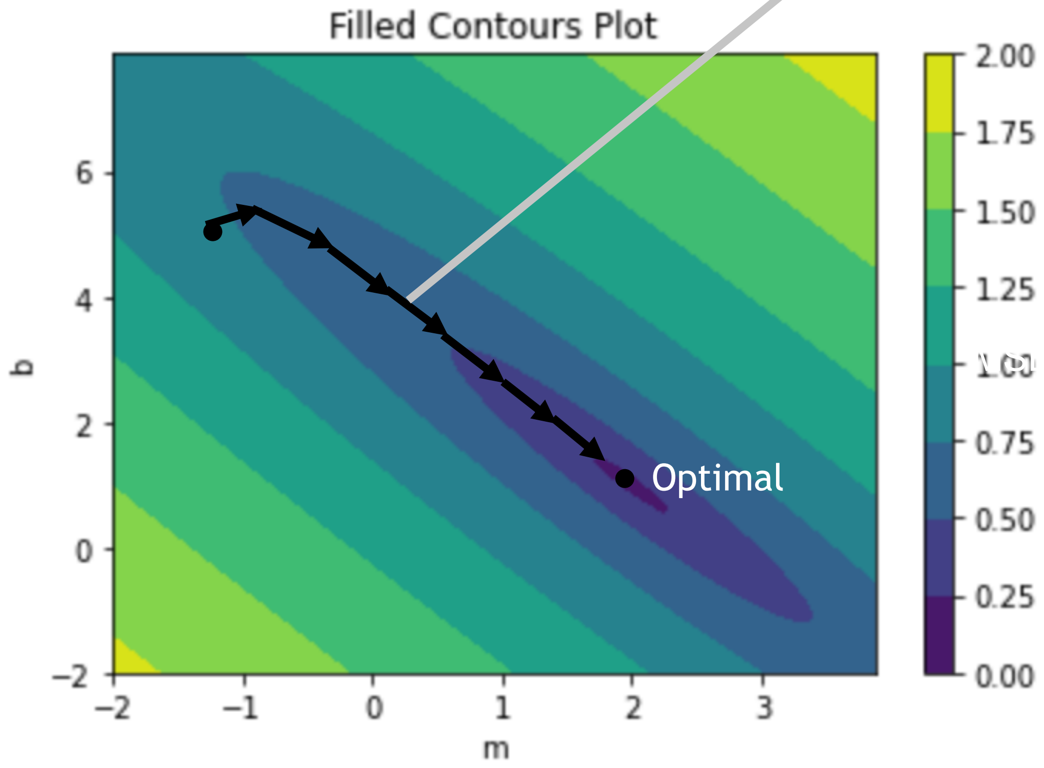
Stochastic Gradient Descent

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

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

Visualizing the training process

Training steps: every time the gradient is updated

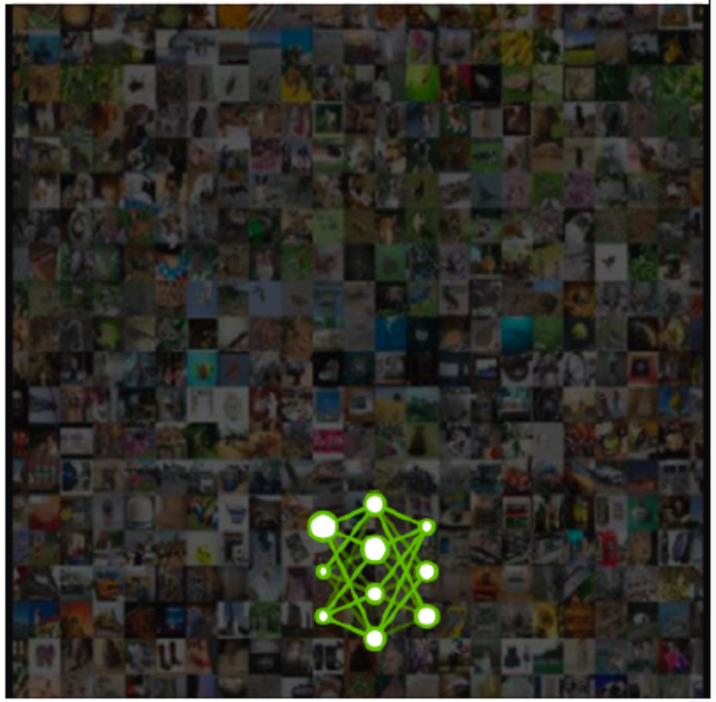


What influences these steps:

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

Models of Increasing complexity

7 Exaflops
60 Million Parameters



2015 - Microsoft ResNet
Superhuman Image Recognition

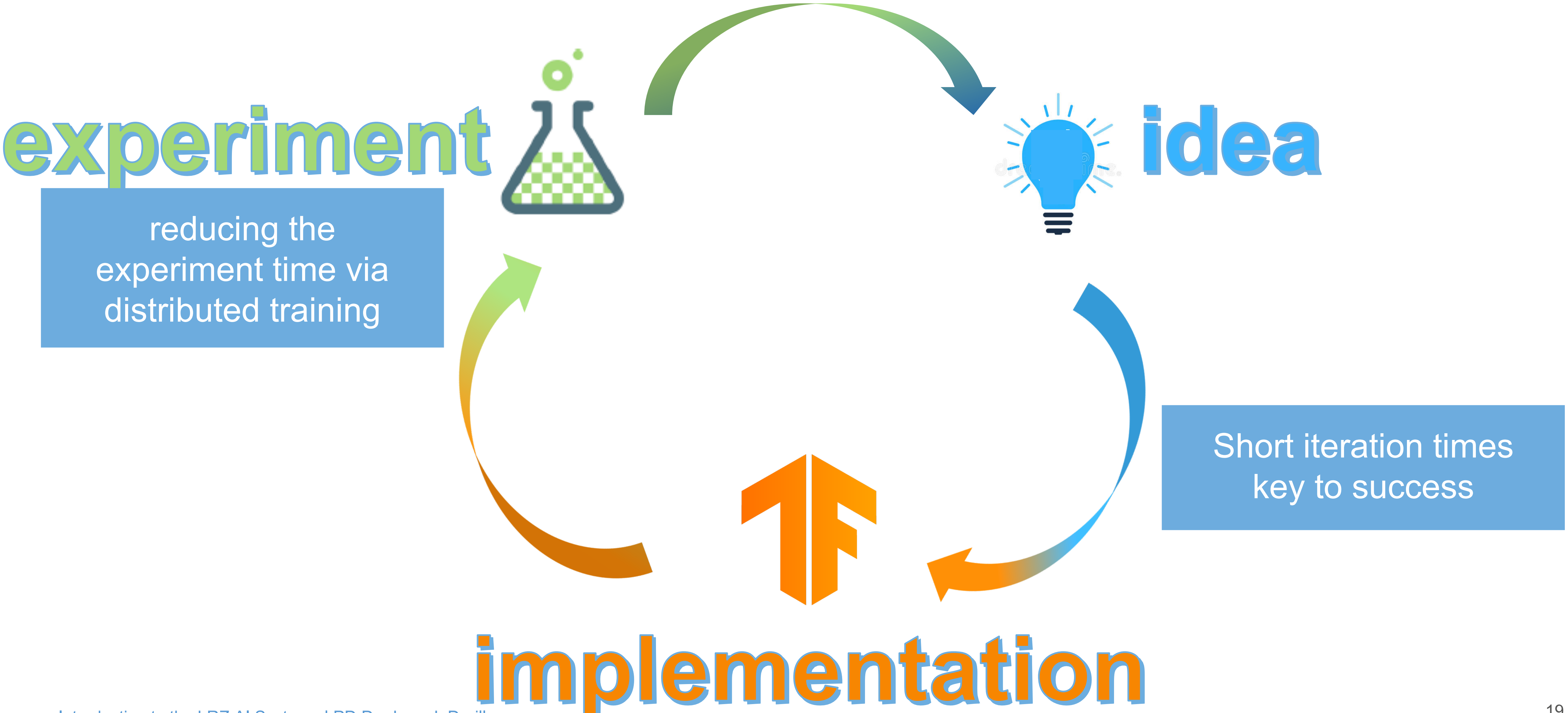
20 Exaflops
300 Million Parameters

2016 - Baidu Deep Speech 2
Superhuman Voice Recognition

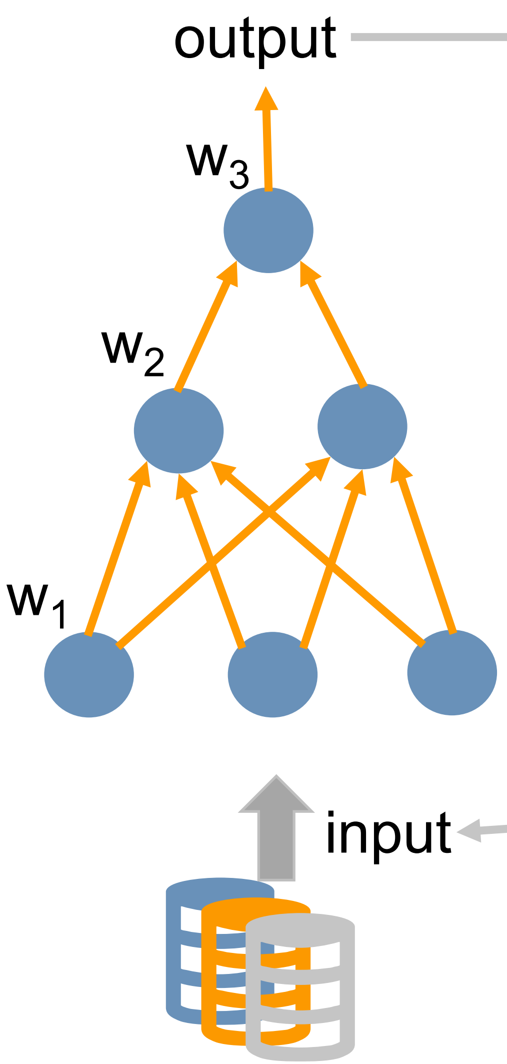
100 Exaflops
8700 Million Parameters

2017 - Google Neural Machine Translation
Near Human Language Translation

Experimental Science Require Short Iteration Times



Data Parallelism Training



$\ell_\phi(output, label))$

$$W_3 = W_3 - n_t \frac{\partial \ell}{\partial W_3}$$

$$W_2 = W_2 - n_t \frac{\partial \ell}{\partial W_2}$$

$$W_1 = W_1 - n_t \frac{\partial \ell}{\partial W_1}$$

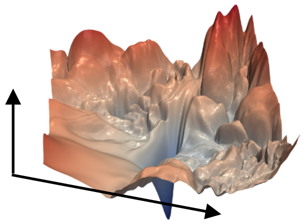
$\ell_\phi(output, label))$

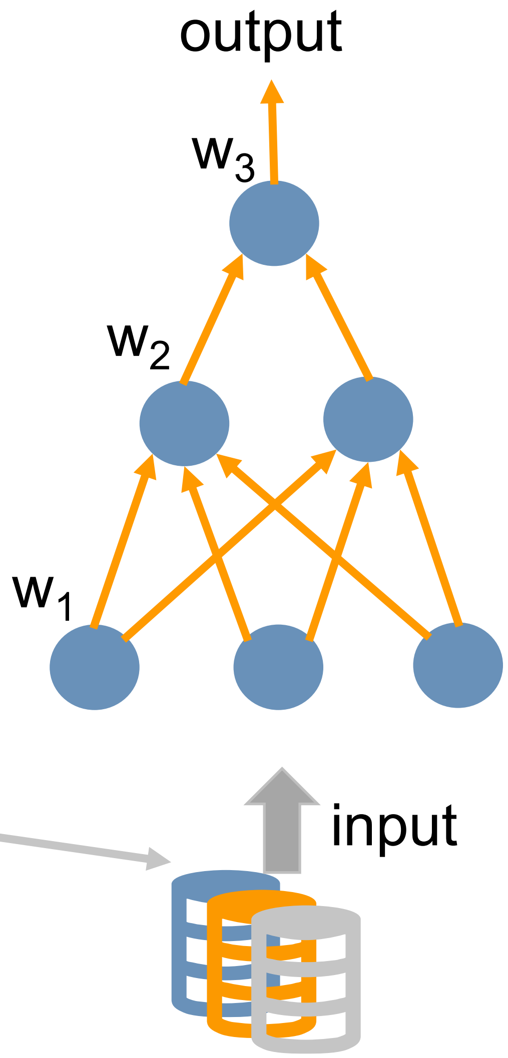
$$W_3 = W_3 - n_t \frac{\partial \ell}{\partial W_3}$$

$$W_2 = W_2 - n_t \frac{\partial \ell}{\partial W_2}$$

$$W_1 = W_1 - n_t \frac{\partial \ell}{\partial W_1}$$

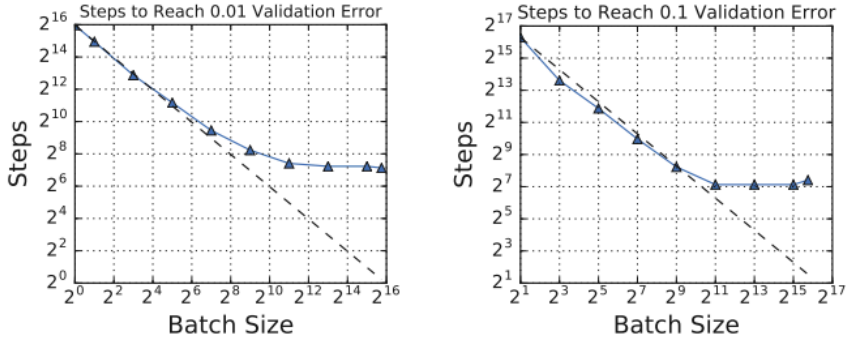
Each device does training on a subset of the batch



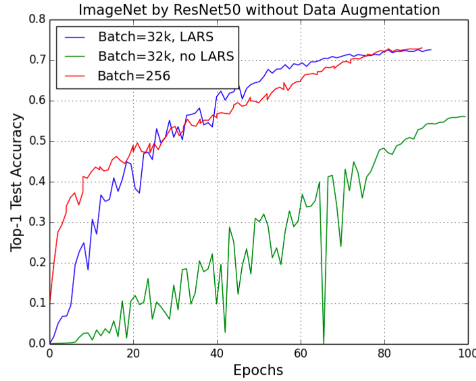


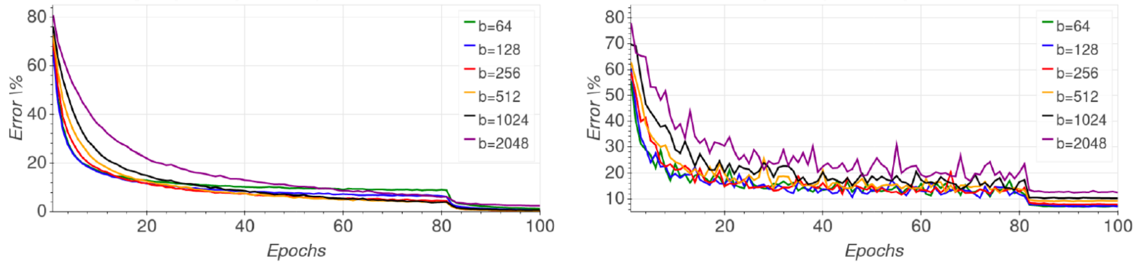
Training with large Batch Sizes

- There are limits

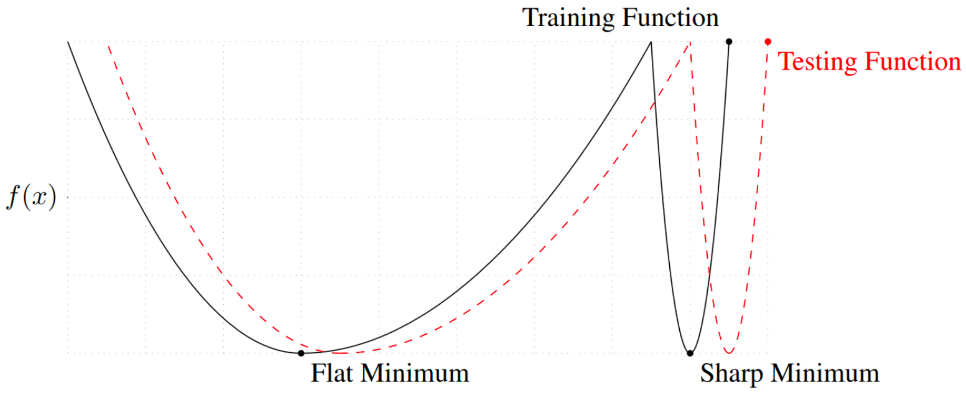


(a) Simple CNN on MNIST (b) Simple CNN on Fashion MNIST





(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

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


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

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

#2 pin resources
```
gpus = tf.config.experimental.list_physical_devices('GPU')
for gpu in gpus:
    tf.config.experimental.set_memory_growth(gpu,True)
if gpus:
    tf.config.experimental.set_visible_devices(gpus[hvd.local_rank()], 'GPU')
```

#3 distributed optimizer
```
opt = #chose your optimizer of preference
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

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

Course Evaluation

Please visit

https://survey.lrz.de/index.php/793144?lang=en
and rate this course.

Your feedback is highly appreciated!

Thank you!