HPC.NRW

NATIONALES HOCHLEISTUNGS RECHNEN

EDIH Rheinland

WEST AI KI-Servicezentrum

# PPCES: Machine and Deep Learning

## Introduction to Deep Learning

DAS KOMPETENZNETZWERK FÜR HOCHLEISTUNGSRECHNEN.

# Agenda – Friday, March 15

HPC.NRW

| Zeit | Thema | Berichtende/r |
|---|---|---|
| 09:00 – 10:30 | Introduction to AI, Machine and Preprocessing Techniques – Part 1 | Jannis Klinkenberg (RWTH) |
| 10:30 – 11:00 | Break | |
| 11:00 – 12:30 | Introduction to AI, Machine and Preprocessing Techniques – Part 2<br>Execution Options for ML / DL Software<br>Lab: Machine Learning with scikit-learn | Jannis Klinkenberg (RWTH) |
| 12:30 – 14:00 | Lunch Break | |
| 14:00 – 14:45 | **Introduction to Deep Learning + Example** | Jannis Klinkenberg (RWTH) |
| 14:45 – 15:30 | **Introduction to Distributed Deep Learning + Example** | Jannis Klinkenberg (RWTH) |
| 15:30 – 16:00 | Break | |
| 16:00 – 17:00 | Lab: Deep Learning with PyTorch | Jannis Klinkenberg (RWTH) |

**Acknowledgements / Contributors:**

- Georg Zitzlsberger, NVIDIA (formerly IT4I Supercomputing Center, Ostrava)
- Dominik Viehhauser and Radita Liem, IT Center RWTH Aachen University

# Deep Learning – History

- **Timeline**
  - **1950:** First perceptron model
    - By Frank Rosenblatt
    - Limited to single layer and single class
  - **1989:** Theoretical Framework for Back-Propagation
    - By Yann le Cun
  - **2012:** Dawn of Deep Neural Networks with AlexNet
  - **2018:** First Large Language Models (LLM) such as GPT
  - **2022:** ChatGPT

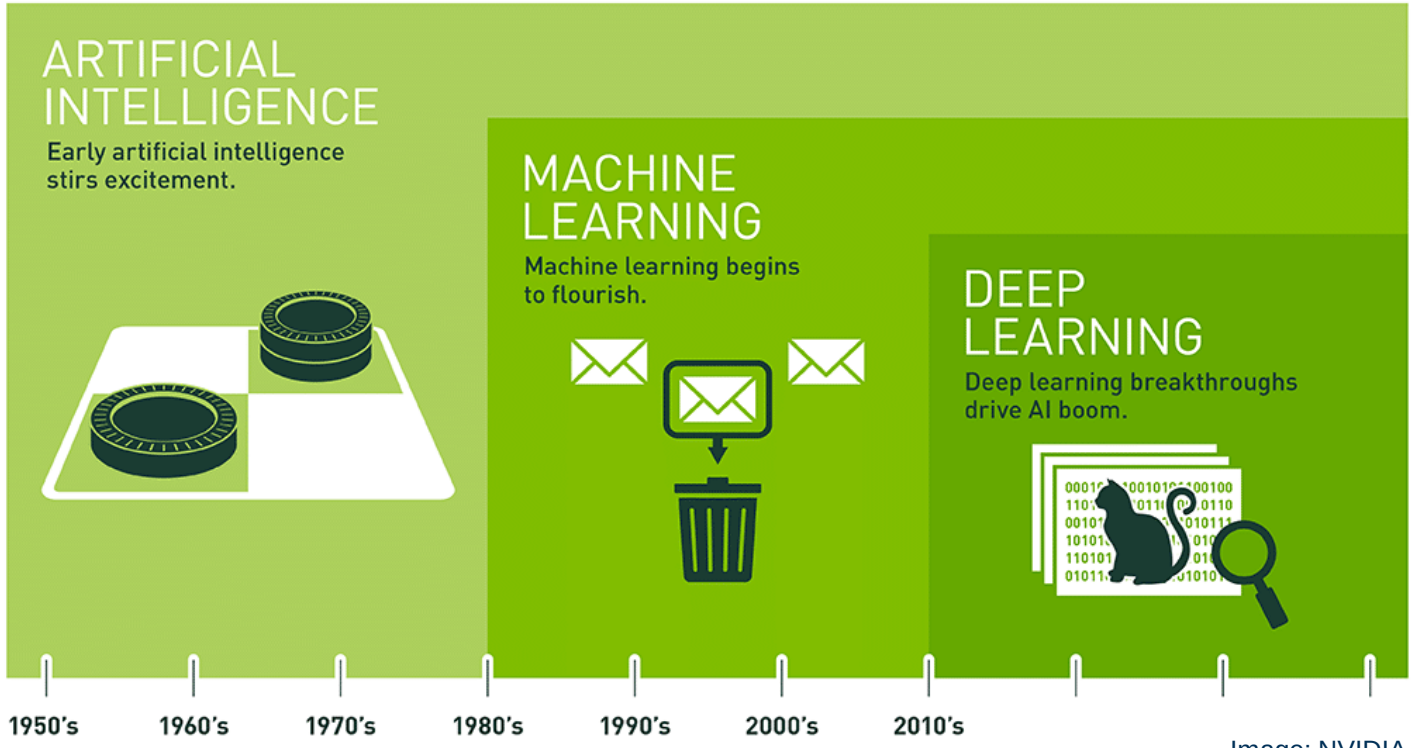



Image: NVIDIA

# Machine Learning vs. Deep Learning

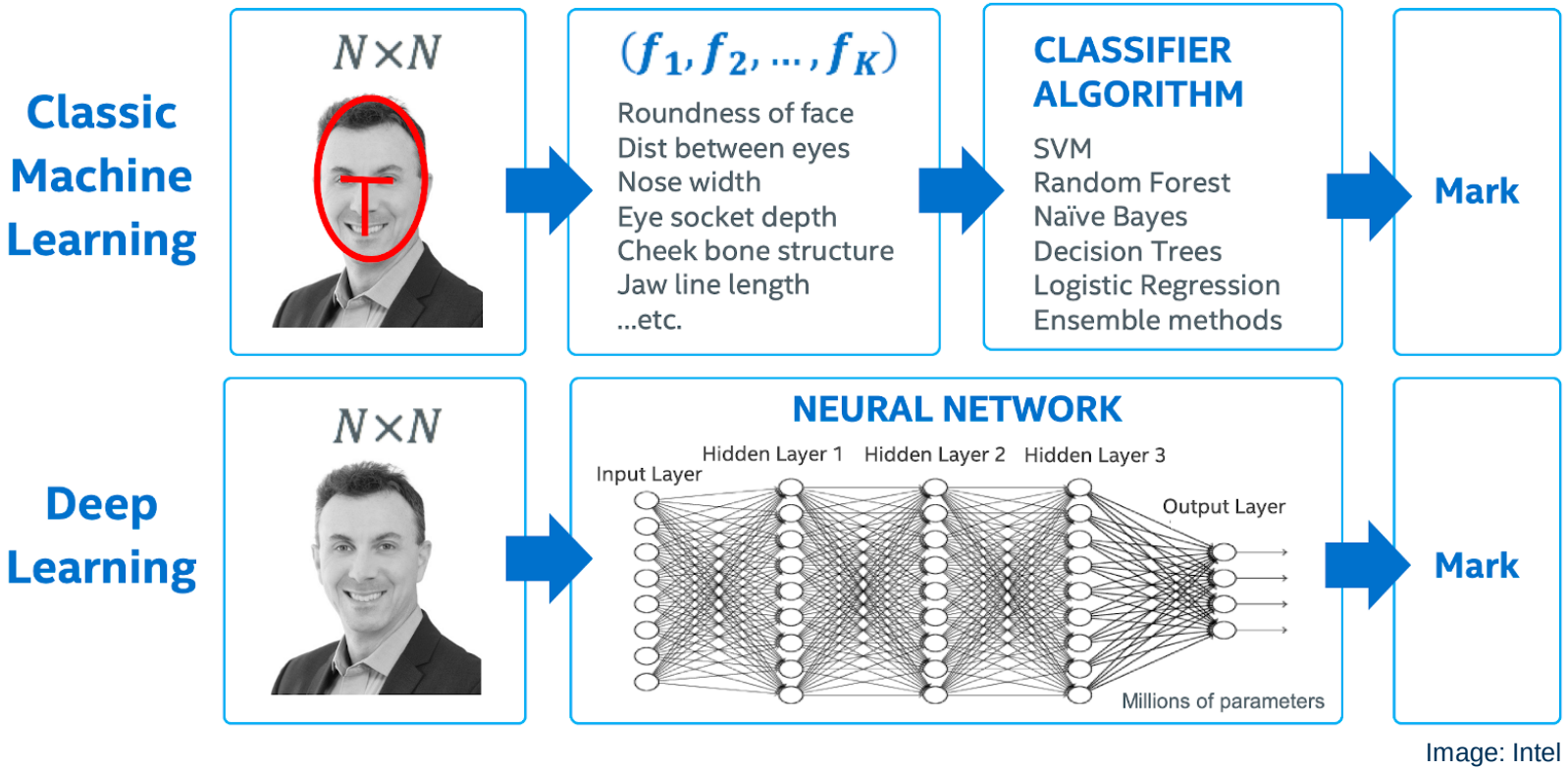



Image: Intel

- **ML:** Feature Engineering
  - Takes a lot of time
  - Usually harder if confronted with unstructured data
- **DL:** Data Engineering
  - Features are discovered automatically from data
  - Extract features at multiple levels of abstraction
  - Performance improves with more data

**Rem. Challenges:** Data cleaning, exploration, hyper-parameter tuning

# Machine Learning vs. Deep Learning

HPC.NRW

- **Pro Machine Learning**
  - Full control over feature space
  - Good for well-defined tasks and structured data
  - Preferable when working with less data
    - Smaller number of samples / features
    - Better generalization than DL

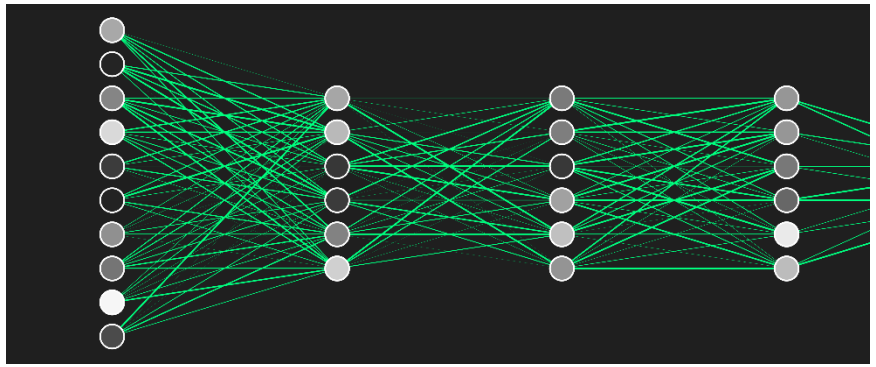
- **Pro Deep Learning**
  - Tackle complex problem spaces w/o feature engineering
  - Ensemble of networks possible
  - Suitable for large amounts of data
  - Easy to use across multiple nodes / GPUs

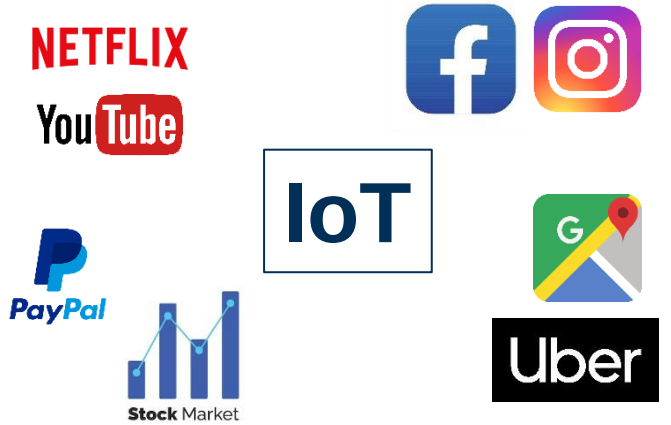
# Why has Deep Learning become so popular now?

HPC.NRW

Image: NVIDIA

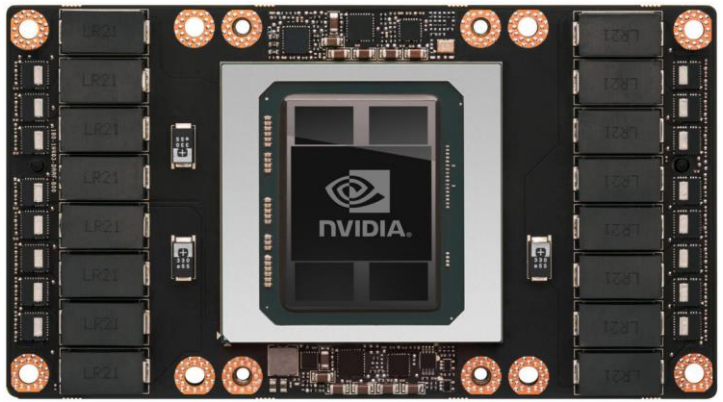

Image: towardsdatascience.com





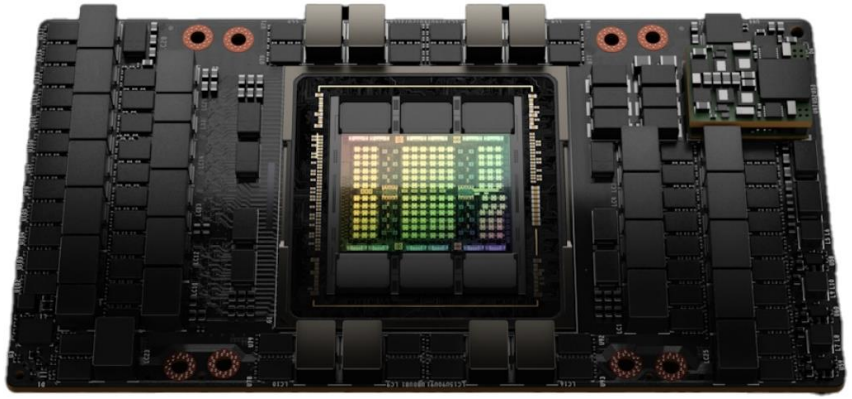

- **Reason 1:** Hardware
  - Modern system are fast enough
  - Enough memory capacity / speed
- **Reason 2:** Advanced Methodologies
  - Ongoing DL development
  - Deep networks / LLMs / Text and Speech Recognition
- **Reason 3:** Big Data
  - Huge amounts of data collected
  - DL much better in processing and analysing

# NVIDIA Tesla P100 / V100 / H100




- Peak Performance
  - 4.7 TFLOPS (double)
  - 9.3 TFLOPS (single)
  - 18.7 TFLOPS (half)
- 3584 CUDA cores
- Memory
  - 16 GB HBM2
  - 732 GB/s BW
- SXM: 300 Watts


- Peak Performance
  - 7.8 TFLOPS (double)
  - 15.7 TFLOPS (single)
  - 125 TFLOPS (half, Tensor)
- Cores: 5120 CUDA + 620 T
- Memory
  - 32 GB HBM2
  - 900 GB/s BW
- SXM: 300 Watts



- Peak Performance
  - 34 TFLOPS (double)
  - 67 TFLOPS (single)
  - 989.4 TFLOPS (half, Tensor)
  - 1979 TFLOPS (half, Tensor, sparse)
- Cores: 16896 CUDA + 528 T
- Memory
  - 80 GB HBM3
  - 3,35 TB/s BW
- SXM: 700 Watts

Images: NVIDIA

# NVIDIA DGX Systems

- **Example: DGX-2**
  - 16x V100 GPUs
  - Peak: 2000 TFLOPS (half)
  - CUDA Cores: 81,920
  - Tensor cores: 10,240
  - 10,000 Watts

- **Example: DGX H100**
  - 8x H100 GPUs
  - Peak: 32000 TFLOPS (FP8)
  - CUDA Cores: 135,168
  - Tensor cores: 4,224
  - 11,300 Watts

Images: NVIDIA

# CPU vs. GPU: Architectural Advantages

HPC.NRW

- **Note:** Deep Learning frameworks typically work on both GPU and CPU

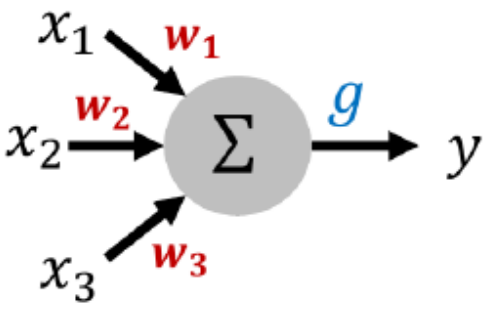
- **Pro CPU**
  - More memory for larger models
  - Easier I/O and setup
  - Some operations / layers can only be executed on the CPU

- **Pro GPU**
  - Very efficient if operation / layers are supported
  - Divide I/O and training between CPU and GPU
  - Best for deep networks

# Artificial Neural Networks

HPC.NRW

- Inspired by biology: Tries to mimic neurons of human brain
- Inputs → Processing → Output



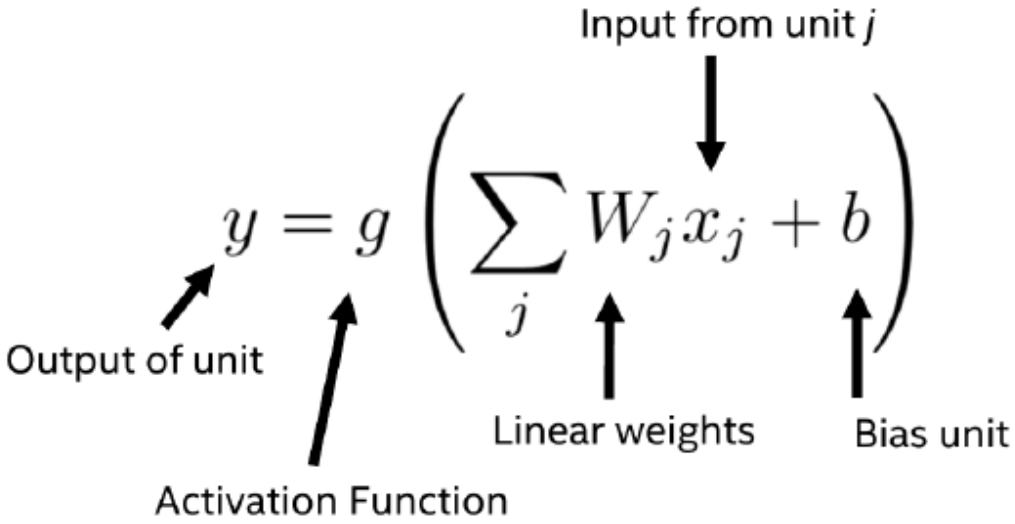




Image: Intel

# Activation Functions

HPC.NRW

- Add non-linear component
- ReLU currently most popular

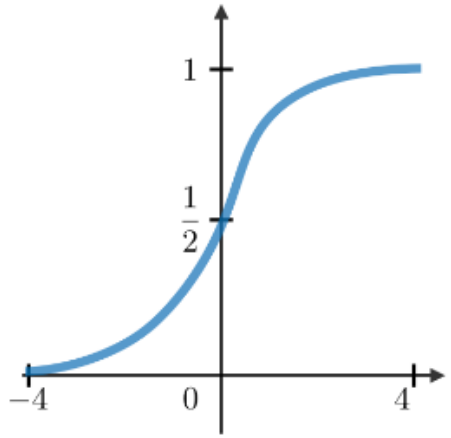
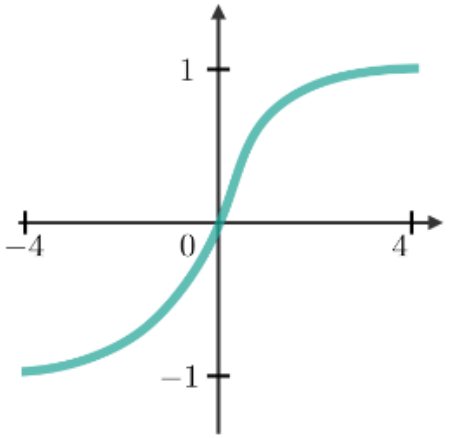
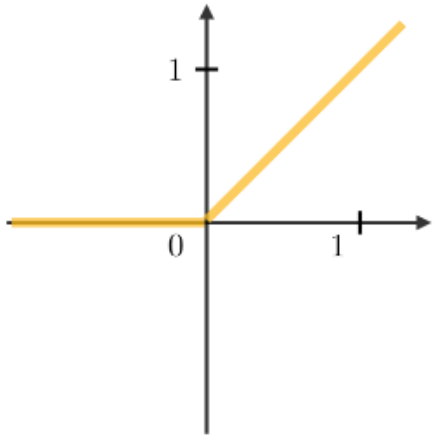
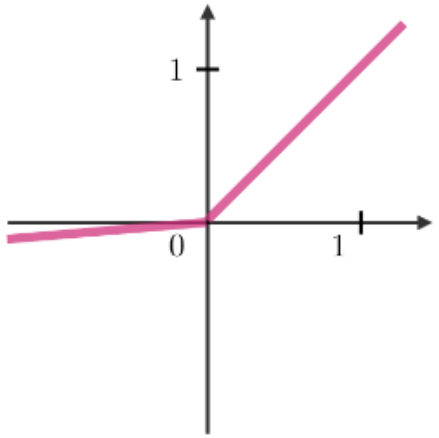
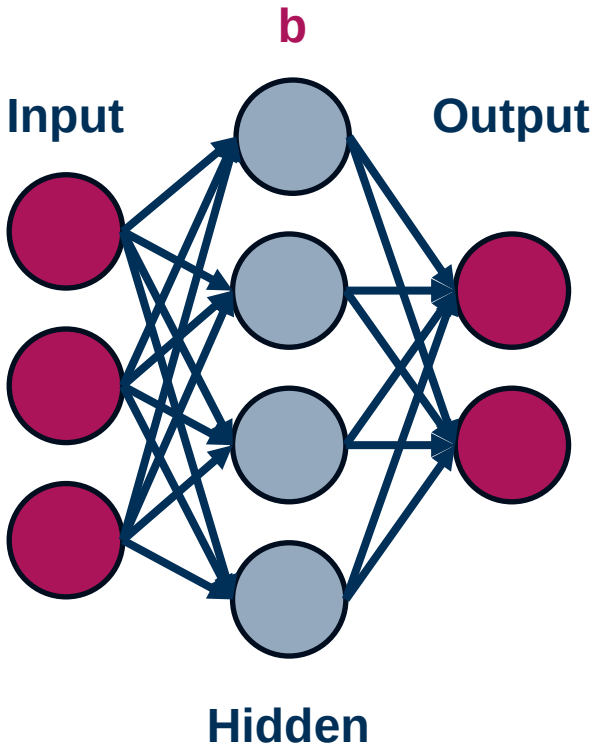
| Sigmoid | Tanh | ReLU | Leaky ReLU |
|---|---|---|---|
| $g(z) = \frac{1}{1+e^{-z}}$ | $g(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$ | $g(z) = \max(0, z)$ | $g(z) = \max(\epsilon z, z)$ with $\epsilon \ll 1$ |
|  |  |  |  |



Image: Afshine Amidi and Shervine Amidi

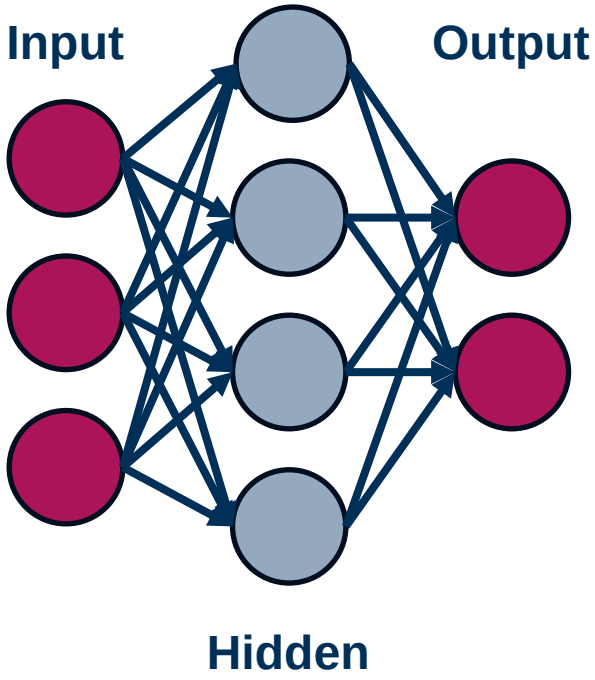
– **Example:** Fully connected NN




– Input and outputs can have different sizes
– Output usually combined with *softmax* function (probabilistic)

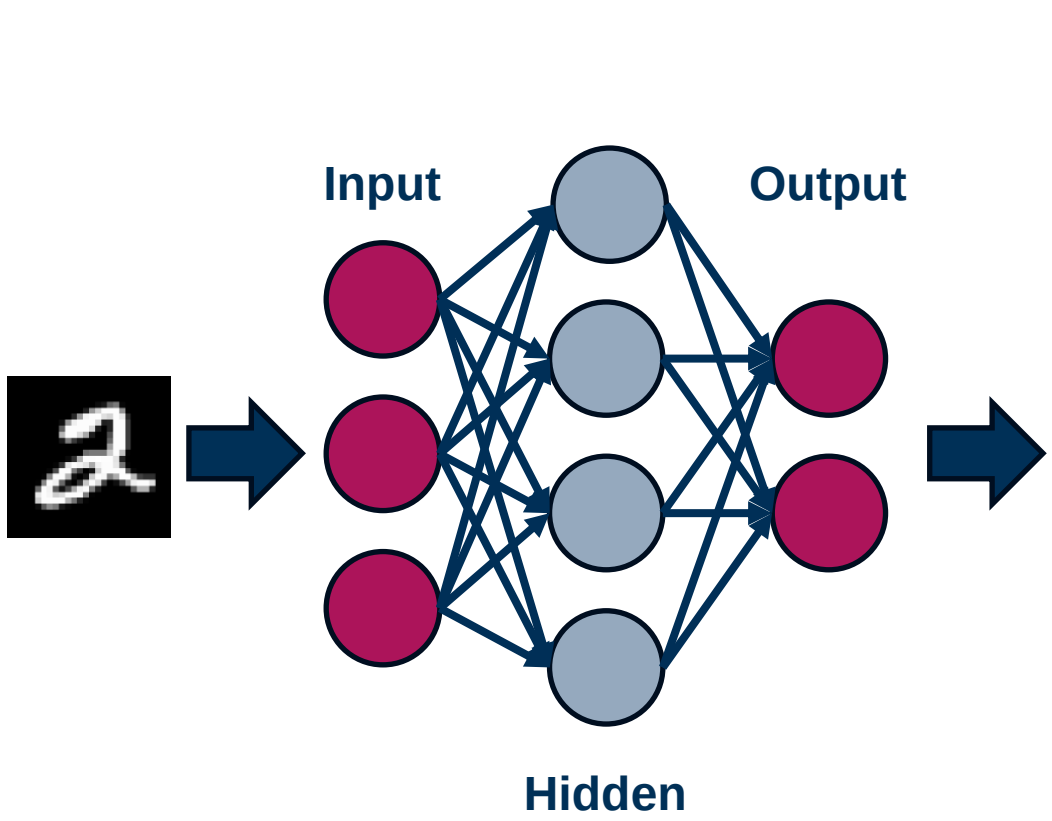
– Multiple hidden layers possible
– Each layer can have different number of neurons
– Typically, one bias parameter per layer

# Deep Neural Networks – Training




1. Initialization (random weights)
2. Get next batch of training data
3. Forward propagation
4. Loss (or cost) calculation
5. Backward propagation
6. Update weights and bias
7. Repeat with step 2

# Deep Neural Networks – Forward Propagation + Loss Calc.

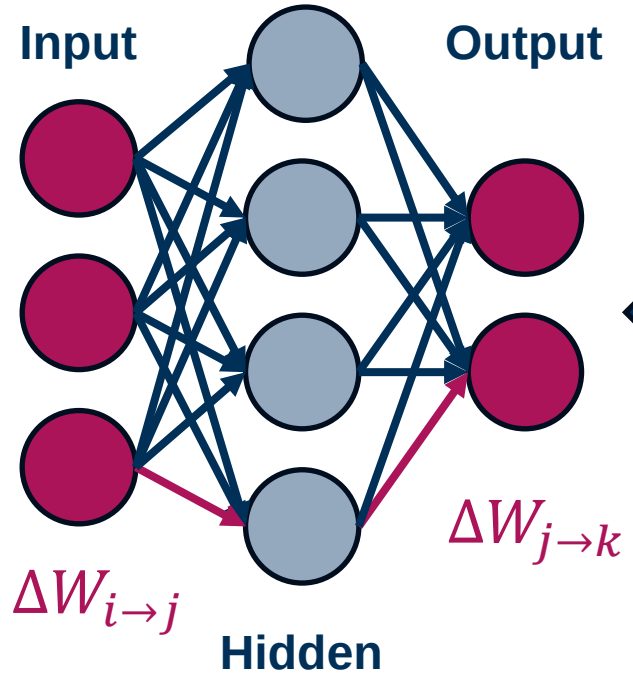



| Digit | Prob. |
|---|---|
| 0 | 0.0 |
| 1 | 0.1 |
| 2 | 0.3 |
| 3 | 0.1 |
| 4 | 0.1 |
| 5 | 0.0 |
| 6 | 0.2 |
| 7 | 0.1 |
| 8 | 0.4 |
| 9 | 0.0 |

| Truth |
|---|
| 0 |
| 0 |
| 1 |
| 0 |
| 0 |
| 0 |
| 0 |
| 0 |
| 0 |
| 0 |

```
loss = func(out, truth)
```

# Deep Neural Networks – Backward Propagation




| Digit | Prob. |
|---|---|
| 0 | 0.0 |
| 1 | 0.1 |
| 2 | 0.3 |
| 3 | 0.1 |
| 4 | 0.1 |
| 5 | 0.0 |
| 6 | 0.2 |
| 7 | 0.1 |
| 8 | 0.4 |
| 9 | 0.0 |

| Truth |
|---|
| 0 |
| 0 |
| 1 |
| 0 |
| 0 |
| 0 |
| 0 |
| 0 |
| 0 |
| 0 |

```
loss = func(out, truth)
```

Calculate gradients / partial derivatives

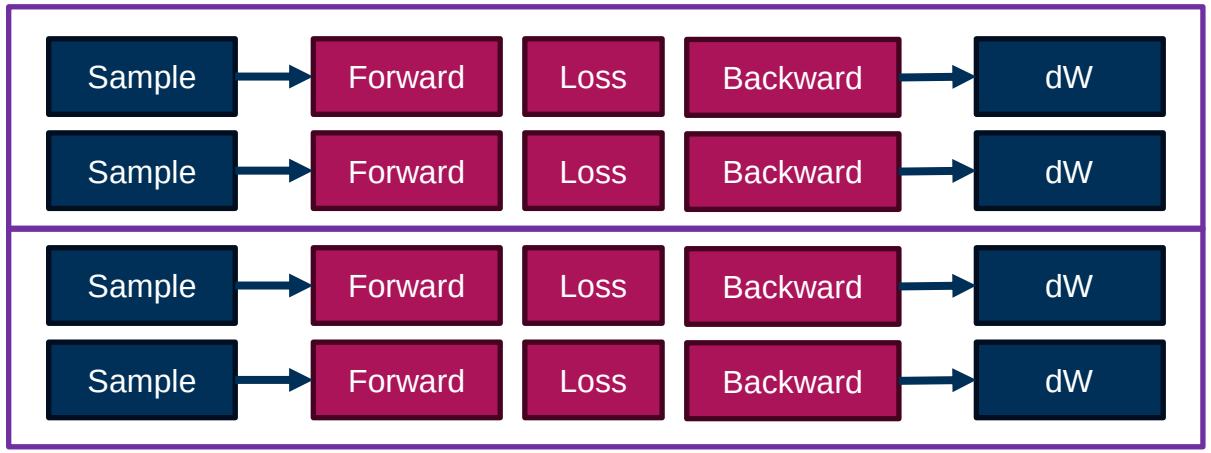
Update individual weights and biases

**Optimizers (SGD, ADAM)**

# Deep Neural Networks – Hyperparameters & Layers

HPC.NRW

- **Batching**
  - Back propagation more expensive
  - Idea: group training data in batches, run parallel forward passes and average weight updates
- **Number and types of layers**
  - Dense / MLP
  - Convolutions (e.g. for images)
  - LSTM (e.g. for time series)
- **Learning rate of optimizers**
  - How far to go into most promising direction
  - Faster convergence vs. stability



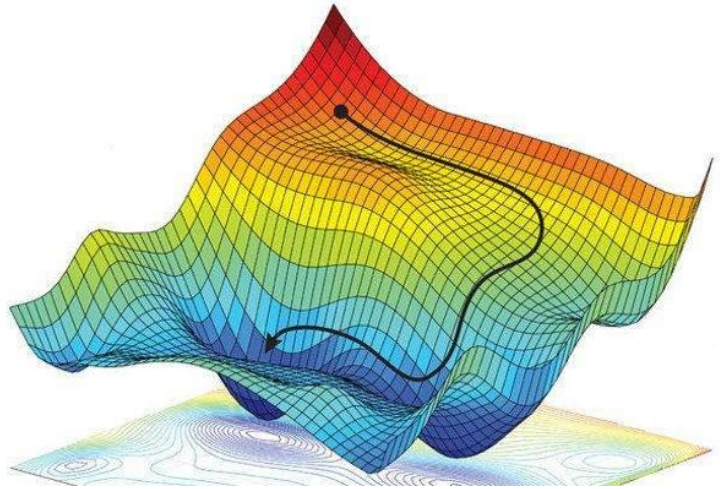



Image: https://easyai.tech/en/ai-definition/gradient-descent/

# Deep Neural Networks – Batch Performance

HPC.NRW

- **Question:** What is a good batch size?
  - Higher batch size = more performance
    - More parallelism that can be exploited
    - Faster training for large datasets
  - But at the same time:
    - Accuracy might decrease
    - More memory needed (limited on GPUs)

  ➢ Find sweet spot empirically

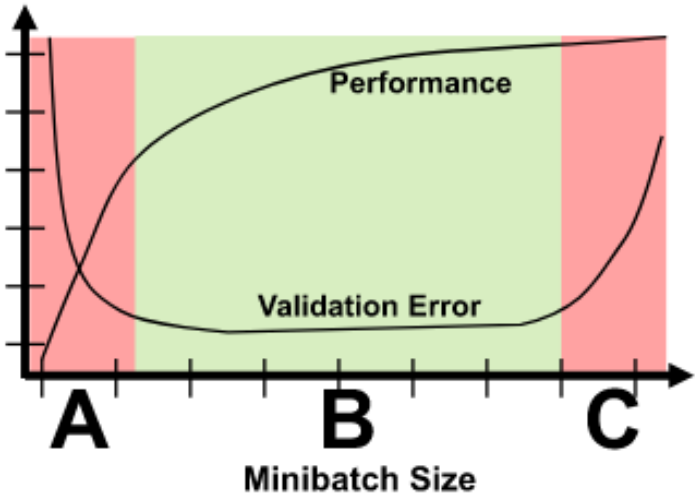



Image: Ben-Nun et al.

# Deep Learning Frameworks – How to get started?

HPC.NRW

- **Different scenarios**
  - Training (time consuming)
  - Inference (using models)

- **Several frameworks available**

TensorFlow

Keras

PyTorch

PyTorch Lightning

HOROVOD

Microsoft CNTK

Caffe2

Colossal-AI

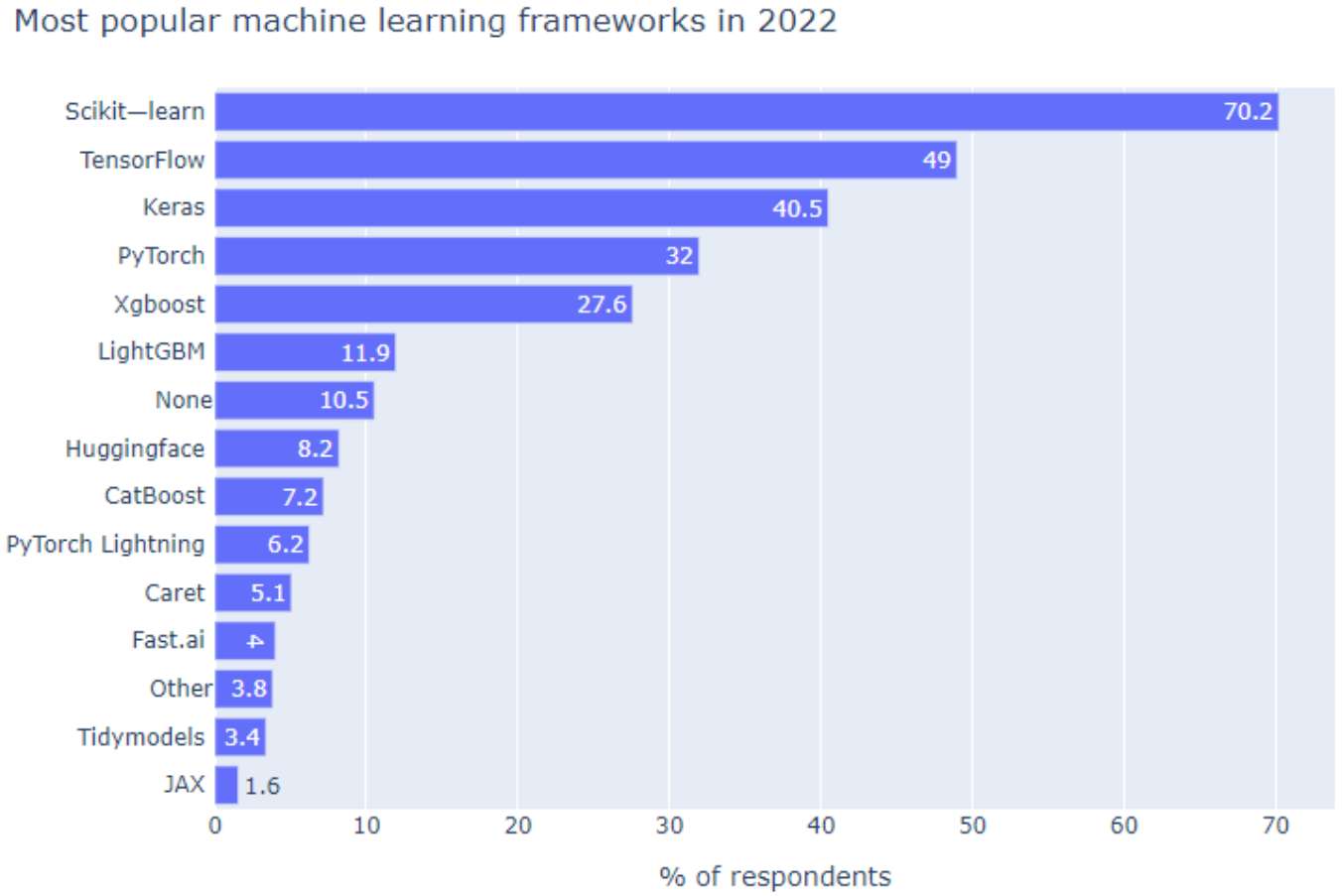



Image: kaggle