HELMHOLTZAI | ARTIFICIAL INTELLIGENCE COOPERATION UNIT

TOWARDS EXASCALE-READY ASTROPHYSICS

Tutorial: AI for astrophysicists

Stefan Kesselheim

Forschungszentrum Jülich & University of Cologne / 2024-09-26

What is Machine Learning?

mitchell1997machine

"A computer program is said to learn from experience E with respect to some class of tasks T, and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E."

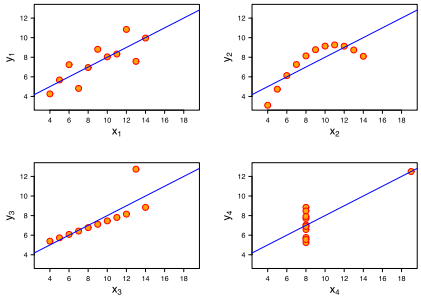
versus

"We require a function $f \in \mathcal{F} : f(x)$ that maps a space of data $x \in \mathcal{X}$ into a different space $y \in \mathcal{Y}$. This function should be useful in some sense. We find it with the help of some data $x_i \in \mathcal{X}$. Finding it, we call Machine Learning. "

Linear Regression

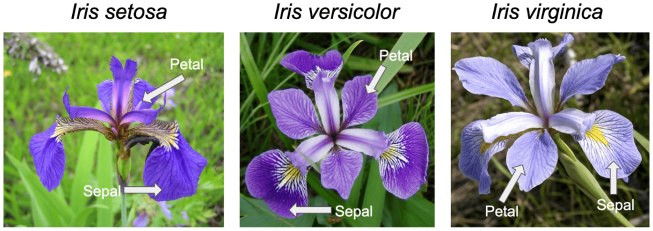
- Linear regression is the simplest example of Machine Learning
- $\mathcal{F}$ is spanned by $f(x; a, b) = a \cdot x + b$
- Data points $x_i \in \mathbb{R}$, labels $y_i \in \mathbb{R}$
- Determine (a, b) from solving the least squares problem:

$$(a, b) = \arg\min \sum_i (a \cdot x_i + b - y_i)^2$$

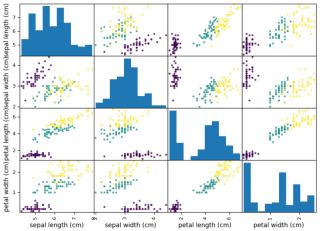


Source: Wikipedia (Users Schutz, Avenue)

Classification

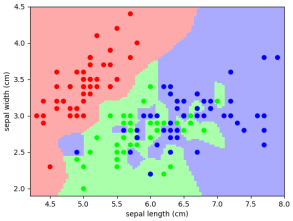


Source:
https://www.embedded-robotics.com/iris-dataset-classification/



Source: https://www.dotnetlovers.com

- The Iris (German: Schwertlilie) dataset lists properties of flowers, and their species. [**fisher36a**].
- Data $x_i \in \mathbb{R}^4$: Width and length of petal, width and length of sepal.
- Labels $y_i \in \{0, 1, 2\}$: Three different flower species.
- Useful functions can (but do not have to be): $f : \mathbb{R}^4 \to \{0, 1, 2\}$.

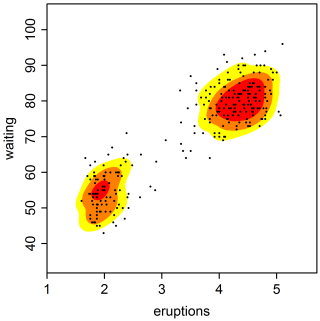


Source: Scipy Lecture Notes

Density Estimation

- Data $\vec{x}_i \in \mathbb{R}^2$: Waiting time between eruptions, eruption duration
- Machine Learning task: approximate the probability density $f(\vec{x})$
- One solution: Kernel density estimation:

$$f(\vec{x}) \approx \hat{f}(\vec{x}) = \frac{1}{n} \sum_{i=0}^{n-1} K(\vec{x} - \vec{x}_i)$$



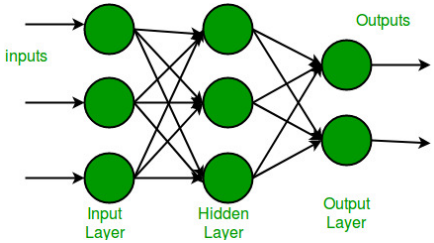
Source: Wikipedia Old Faithful Geyser Kernel Density Estimate User: Drleft

The ingredients of deep learning

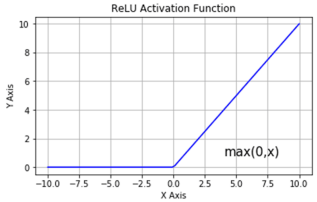
1. Powerful function families
2. Robust learning mechanism
3. Large datasets
4. Accessible Compute
5. Flexible programming models to combine all this.

The Multi-Layer Perceptron

- Input: $\vec{x}$
- Hidden Layer: $\vec{z}'_0 = \theta_0 \cdot \vec{x}$
- Activation: $\vec{z}_0 = f(\vec{z}'_0)$
- Output Layer: $\vec{y}_0 = \theta_1 \cdot \vec{x}$



https://www.geeksforgeeks.org/multi-layer-perceptron-learning-in-tensorflow/



https://www.nomidl.com/deep-learning/what-is-relu-and-sigmoid-activation-function/

Stochastic Gradient Descent

Supervised deep learning

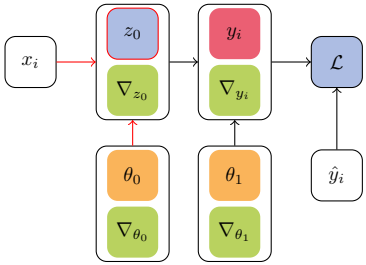
- Neural Network with input x output y, weights θ

$$f : y = f(x; \theta) .$$

- Define a loss $\mathcal{L}(y, \dots)$
- Define dataset x_i in batches of size b, with ground truth $\hat{y}_i$
- Initialize weights θ randomly
- Repeat until convergence
  1. Draw batch: $x_{k \cdot b} \cdots x_{k \cdot b + b - 1}$
  2. Calculate batch loss: $Q = \frac{1}{b} \sum_{\text{batch}} \mathcal{L}(x_i f())$
  3. Calculate batch loss gradient: $\nabla_\theta Q$
  4. Update weights: $\theta^{i+1} = \theta^i + \eta \nabla_\theta Q$
  5. Evaluate Model

The Backpropagation Algorithm

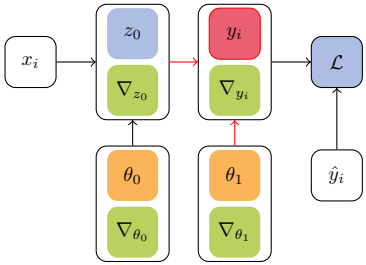
How to obtain the gradient



1. $z_0 = f_0\left(x_i; \theta_0\right)$
2. $y_i = f_1\left(z_1; \theta_1\right)$
3. $\mathcal{L} = L\left(y_i, \hat{y}_i\right)$
4. $\nabla_{y_i} = \partial / \partial y_i \, L\left(y_i, \hat{y}_i\right)$
5. $\nabla_{\theta_1} = \partial / \partial \theta_1 \, f_1\left(z_1; \theta_1\right) \cdot \nabla_{y_i}$
6. $\nabla_{z_1} = \partial / \partial z_1 \, f_1\left(z_0; \theta_1\right) \cdot \nabla_{y_i}$
7. $\nabla_{\theta_0} = \partial / \partial \theta_0 \, f_0\left(x_i; \theta_0\right) \cdot \nabla_{z_1}$

The Backpropagation Algorithm

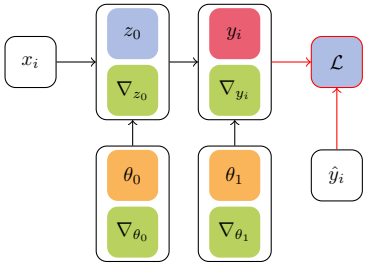
How to obtain the gradient



1. $z_0 = f_0\left(x_i; \theta_0\right)$
2. $y_i = f_1\left(z_1; \theta_1\right)$
3. $\mathcal{L} = L\left(y_i, \hat{y}_i\right)$
4. $\nabla_{y_i} = \partial / \partial y_i \, L\left(y_i, \hat{y}_i\right)$
5. $\nabla_{\theta_1} = \partial / \partial \theta_1 \, f_1\left(z_1; \theta_1\right) \cdot \nabla_{y_i}$
6. $\nabla_{z_1} = \partial / \partial z_1 \, f_1\left(z_0; \theta_1\right) \cdot \nabla_{y_i}$
7. $\nabla_{\theta_0} = \partial / \partial \theta_0 \, f_0\left(x_i; \theta_0\right) \cdot \nabla_{z_1}$

The Backpropagation Algorithm

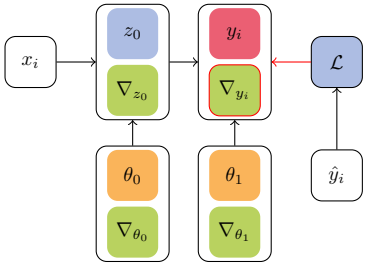
How to obtain the gradient



1. $z_0 = f_0\left(x_i; \theta_0\right)$
2. $y_i = f_1\left(z_1; \theta_1\right)$
3. $\mathcal{L} = L\left(y_i, \hat{y}_i\right)$
4. $\nabla_{y_i} = \partial / \partial y_i \, L\left(y_i, \hat{y}_i\right)$
5. $\nabla_{\theta_1} = \partial / \partial \theta_1 \, f_1\left(z_1; \theta_1\right) \cdot \nabla_{y_i}$
6. $\nabla_{z_1} = \partial / \partial z_1 \, f_1\left(z_0; \theta_1\right) \cdot \nabla_{y_i}$
7. $\nabla_{\theta_0} = \partial / \partial \theta_0 \, f_0\left(x_i; \theta_0\right) \cdot \nabla_{z_1}$

The Backpropagation Algorithm

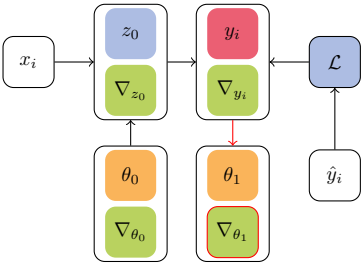
How to obtain the gradient



1. $z_0 = f_0\left(x_i; \theta_0\right)$
2. $y_i = f_1\left(z_1; \theta_1\right)$
3. $\mathcal{L} = L\left(y_i, \hat{y}_i\right)$
4. $\nabla_{y_i} = \partial / \partial y_i \, L\left(y_i, \hat{y}_i\right)$
5. $\nabla_{\theta_1} = \partial / \partial \theta_1 \, f_1\left(z_1; \theta_1\right) \cdot \nabla_{y_i}$
6. $\nabla_{z_1} = \partial / \partial z_1 \, f_1\left(z_0; \theta_1\right) \cdot \nabla_{y_i}$
7. $\nabla_{\theta_0} = \partial / \partial \theta_0 \, f_0\left(x_i; \theta_0\right) \cdot \nabla_{z_1}$

The Backpropagation Algorithm

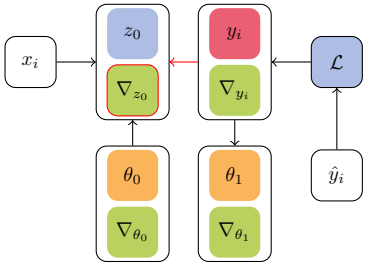
How to obtain the gradient



1. $z_0 = f_0\left(x_i; \theta_0\right)$
2. $y_i = f_1\left(z_1; \theta_1\right)$
3. $\mathcal{L} = L\left(y_i, \hat{y}_i\right)$
4. $\nabla_{y_i} = \partial / \partial y_i \, L\left(y_i, \hat{y}_i\right)$
5. $\nabla_{\theta_1} = \partial / \partial \theta_1 \, f_1\left(z_1; \theta_1\right) \cdot \nabla_{y_i}$
6. $\nabla_{z_1} = \partial / \partial z_1 \, f_1\left(z_0; \theta_1\right) \cdot \nabla_{y_i}$
7. $\nabla_{\theta_0} = \partial / \partial \theta_0 \, f_0\left(x_i; \theta_0\right) \cdot \nabla_{z_1}$

The Backpropagation Algorithm

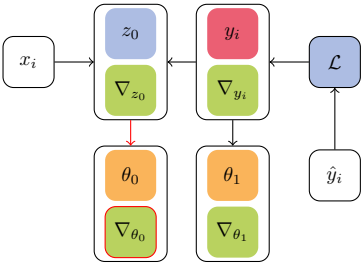
How to obtain the gradient



1. $z_0 = f_0\left(x_i; \theta_0\right)$
2. $y_i = f_1\left(z_1; \theta_1\right)$
3. $\mathcal{L} = L\left(y_i, \hat{y}_i\right)$
4. $\nabla_{y_i} = \partial / \partial y_i \, L\left(y_i, \hat{y}_i\right)$
5. $\nabla_{\theta_1} = \partial / \partial \theta_1 \, f_1\left(z_1; \theta_1\right) \cdot \nabla_{y_i}$
6. $\nabla_{z_1} = \partial / \partial z_1 \, f_1\left(z_0; \theta_1\right) \cdot \nabla_{y_i}$
7. $\nabla_{\theta_0} = \partial / \partial \theta_0 \, f_0\left(x_i; \theta_0\right) \cdot \nabla_{z_1}$

The Backpropagation Algorithm

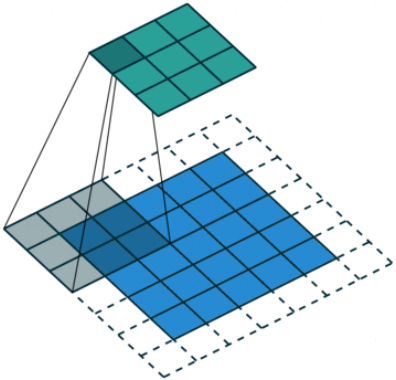
How to obtain the gradient

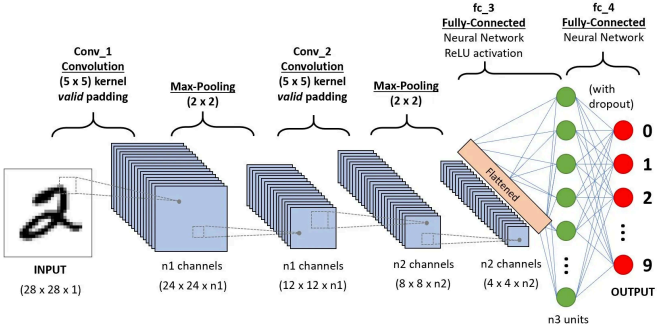


1. $z_0 = f_0\left(x_i; \theta_0\right)$
2. $y_i = f_1\left(z_1; \theta_1\right)$
3. $\mathcal{L} = L\left(y_i, \hat{y}_i\right)$
4. $\nabla_{y_i} = \partial / \partial y_i \, L\left(y_i, \hat{y}_i\right)$
5. $\nabla_{\theta_1} = \partial / \partial \theta_1 \, f_1\left(z_1; \theta_1\right) \cdot \nabla_{y_i}$
6. $\nabla_{z_1} = \partial / \partial z_1 \, f_1\left(z_0; \theta_1\right) \cdot \nabla_{y_i}$
7. $\nabla_{\theta_0} = \partial / \partial \theta_0 \, f_0\left(x_i; \theta_0\right) \cdot \nabla_{z_1}$

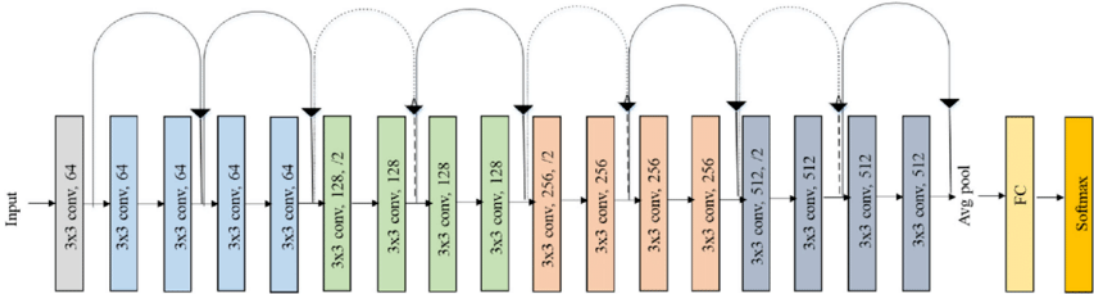
Convolutional Neural Networks

- Building blocks local in pixel space.

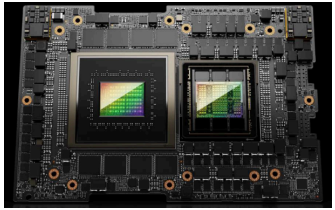




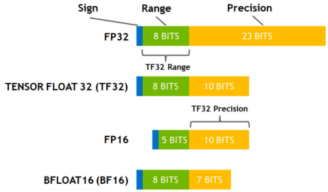
Convolutional Neural Networks



NVidias GH200



https://nvidia.com



https://deeprec.readthedocs.io/en/latest/NVIDIA-TF32.html

Product Specifications

Feature	GH200	GH200 NVL2
CPU core count	72 Arm Neoverse V2 cores	144 Arm Neoverse V2 cores
L1 cache	64KB i-cache + 64KB d-cache	
L2 cache	1MB per core	
L3 cache	114MB	228MB
Base frequency | all-core single instruction, multiple data (SIMD) frequency	3.1GHz | 3.0GHz	
LPDDR5X size	480GB 120GB, 240GB	960GB 240GB, 480GB
Memory bandwidth	Up to 384GB/s Up to 512GB/s	Up to 768GB/s Up to 1024GB/s
PCIe links	Up to 4x PCIe x16 (Gen5)	Up to 8x PCIe x16 (Gen5)

Feature	GH200	GH200 NVL2
FP64	34 teraFLOPS	68 teraFLOPS
FP64 Tensor Core	67 teraFLOPS	134 teraFLOPS
FP32	67 teraFLOPS	134 teraFLOPS
TF32 Tensor Core	989 teraFLOPS* | 494 teraFLOPS	1979 teraFLOPS* | 990 teraFLOPS
BFLOAT16 Tensor Core	1,979 teraFLOPS* | 990 teraFLOPS	3958 teraFLOPS* | 1979 teraFLOPS
FP16 Tensor Core	1,979 teraFLOPS* | 990 teraFLOPS	3958 teraFLOPS* | 1979 teraFLOPS
FP8 Tensor Core	3,958 teraFLOPS* | 1,979 teraFLOPS	7916 teraFLOPS* | 3958 teraFLOPS
INT8 Tensor Core	3,958 TOPS* | 1,979 TOPS	7916 TOPS* | 3958 TOPS
High-bandwidth memory (HBM) size	96GB HBM3 | 144GB HBM3e	Up to 288GB HBM3e
Memory bandwidth	Up to 4TB/s | Up to 4.9TB/s	Up to 9.8TB/s
NVIDIA NVLink-C2C CPU-to-GPU bandwidth	900 GB/s	900 GB/s
Power	Configurable 450 to 1000W (Memory + CPU + GPU)	Configurable 900W to 2000W (Memory + CPU + GPU)
Thermal solution	Air cooled or liquid cooled	

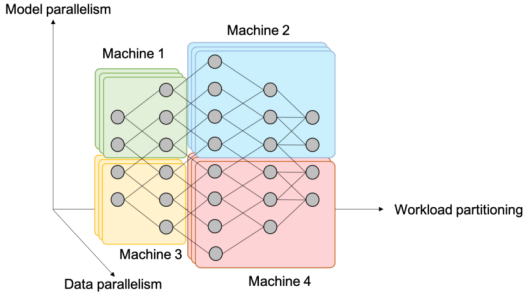
PyTorch

```
model.train()
for batch_idx, (data, target) in enumerate(train_loader):
    data, target = data.to(device), target.to(device)
    optimizer.zero_grad()
    output = model(data)
    loss = loss_fn(output, target)
    loss.backward()
    optimizer.step()
    model.eval()

test_loss = 0
with torch.no_grad():
    for data, target in test_loader:
        data, target = data.to(device), target.to(device)
        output = model(data)
        test_loss += loss_fn(output, target).item()
```

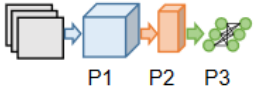
Parallelism in ML

- Model parallelism: Concurrent execution of different parts of the model
- Data parallelism: Compute units perform calculation of different data
- Layer pipelining: Different layers on different compute units.



(a) Data Parallelism

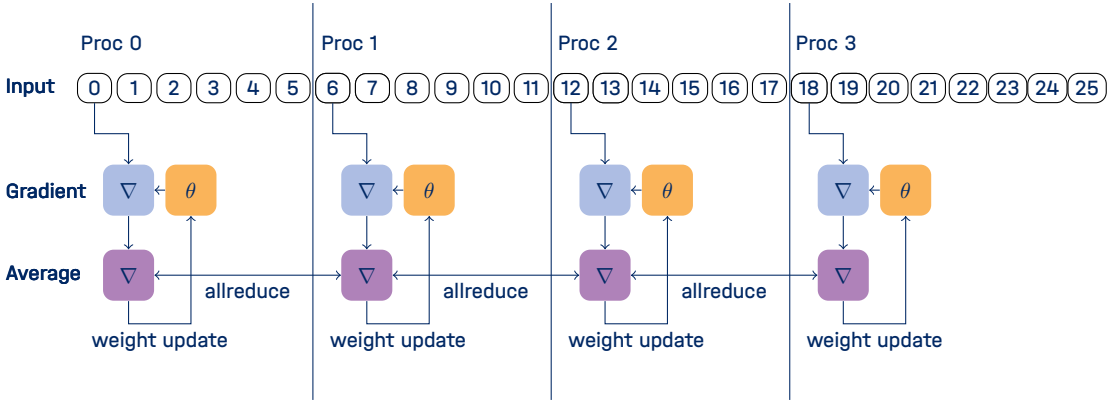
(b) Model Parallelism



(c) Layer Pipelining

[1]Ben-Nun et al., ACM Computing Surveys 52, 2019.

data-parallel gradient descent

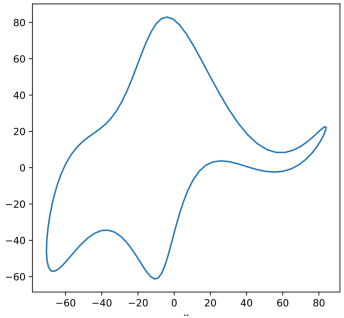


The Bias-Variance Tradeoff

The elephant in the room

Dyson: "Why don't you like my theory?"
Fermi: "How many arbitrary parameters did you use for your calculations?"
Dyson (thinking for a moment): "Four."
Fermi: "I remember my friend Johnny von Neumann used to say, with four parameters I can fit an elephant, and with five I can make him wiggle his trunk."

dyson2004meeting

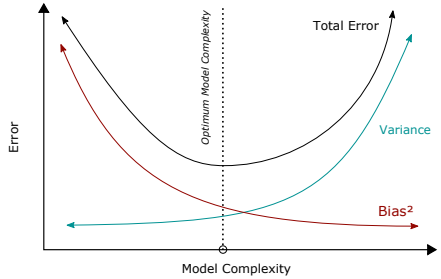


Source: Mayers Elephant [**mayer2010drawing**]

The Bias-Variance Tradeoff

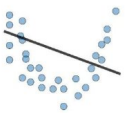
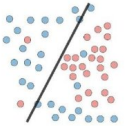
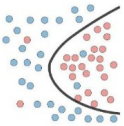
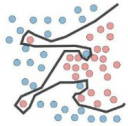
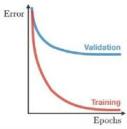
Wikipedia's definition

"The bias error is an error from erroneous assumptions in the learning algorithm. High bias can cause an algorithm to miss the relevant relations between features and target outputs (underfitting)."
"The variance is an error from sensitivity to small fluctuations in the training set. High variance may result from an algorithm modeling the random noise in the training data (overfitting)."



Source: Wikipedia User Bigbossfarin

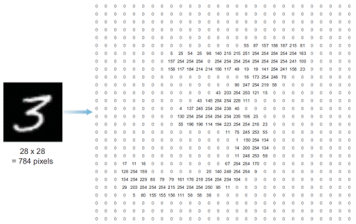
The Bias-Variance Tradeoff

	Underfitting	Just right	Overfitting
Symptoms	- High training error - Training error close to test error - High bias	- Training error slightly lower than test error	- Low training error - Training error much lower than test error - High variance
Regression			
Classification			
Deep learning			
Remedies	- Complexify model - Add more features - Train longer		- Regularize - Get more data

Source: Source could not be traced.

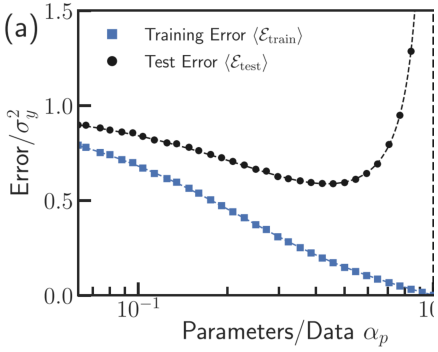
Computer vision lets classical ML fail

1. High-dimensional input makes it difficult to specify features
2. High dataset variability requires large parameter numbers
3. Deep neural networks are challenging to train
4. Conventional wisdom said that high-parametric models will not generalize well



Source: Mayers Elephant [mayer2010drawing]

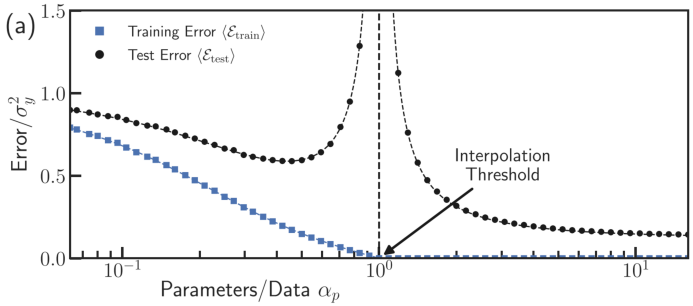
Deep Double Descent



Source: **schaeffer2024double, rocks2022memorizing**

- The classical view: More parameters reduce the error measured on the training set, but harm the generalization to unseen data.
- Training error can be reduced to zero, but at this point, the test error diverges.
- This point is denominated as *Interpolation threshold* because starting at this point, the capacity of a model is sufficient to interpolate the data, e.g. reproduce the values at the training points.

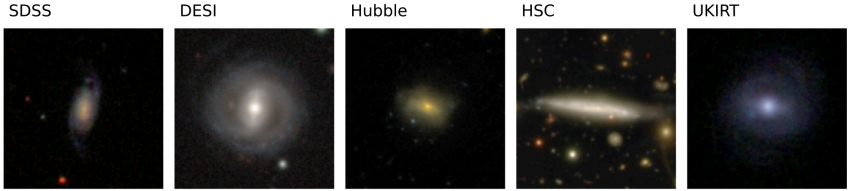
Deep Double Descent



Source: [schaeffer2024double, rocks2022memorizing]

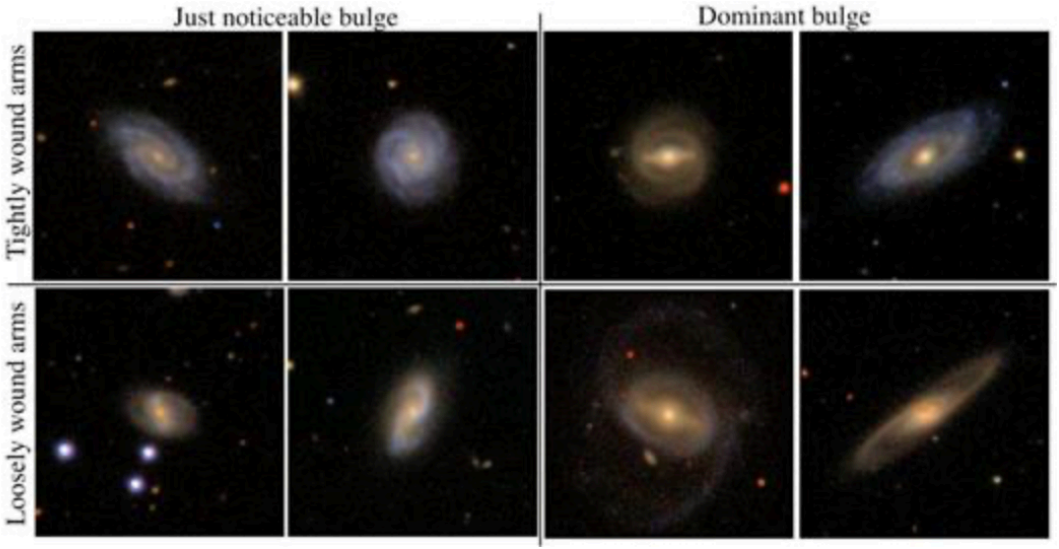
- There is another regime, the overparametrized regime, where different rules apply.

Galaxy Zoo Tutorial



- Train a ResNet18 on the Galaxy Zoo

Galaxy Zoo Tutorial



The Tutorial

1. Setup of a Python Virtual Environment based on the JURECA modules
2. Interactive single-GPU training
3. Multi-GPU training
   - Attention: Unfortunately, I could not figure out how to shard the data in this setup.
   - All processes write stuff on the screen. This is to illustrate what happens under the surface

Please add `--reservation=tera_day3` to `start_interactive_gpu_job.sh` and `run.sbatch`.

https://gitlab.jsc.fz-juelich.de/sdlaml/tutorials/galaxy-classification/-/tree/main

References I