BERKELEY LAB
Bringing Science Solutions to the World

U.S. DEPARTMENT OF ENERGY
Office of Science

Introduction to ML: Part II

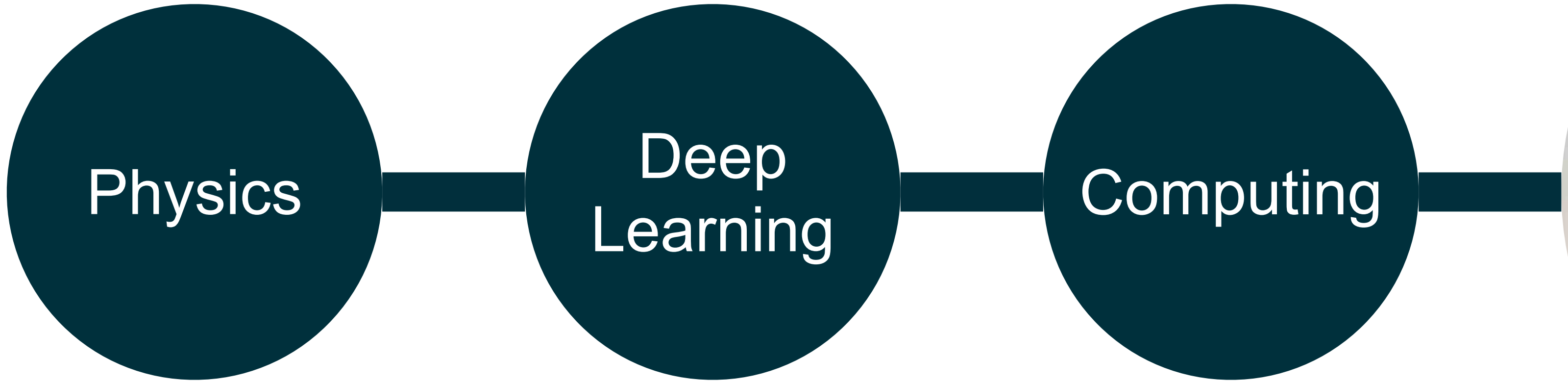
Dennis Noll

ML4FP School 2024

August 12, 2024

About Me

- Postdoc at Lawrence Berkeley National Lab
- Since >8 years working in data analysis @ LHC (CMS & ATLAS)



About you

What is your data science experience?
(Courses, Projects, …)

A No experience

B 1-2 years

C 3-4 years

D ≧ 5 years

About you

The data I am usually using is …

A O(MB)

B O(GB)

C O(TB)

D O(PB)

About you

What is the structure of your data?

A Rectangular (List, …)

B Geometric (Picture, …)

C Point Cloud (Set, Graph, …)

D Other / Don’t know

About you

I have used Advanced Machine Learning Models **(CNN, GNN, …)**

A What are CNN or GNN?

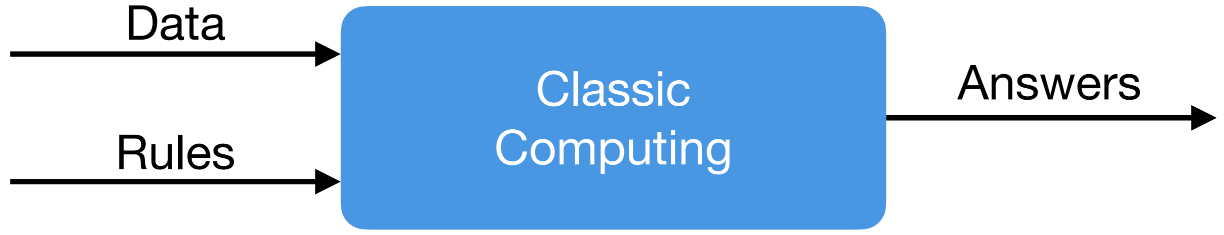
B Tried it out

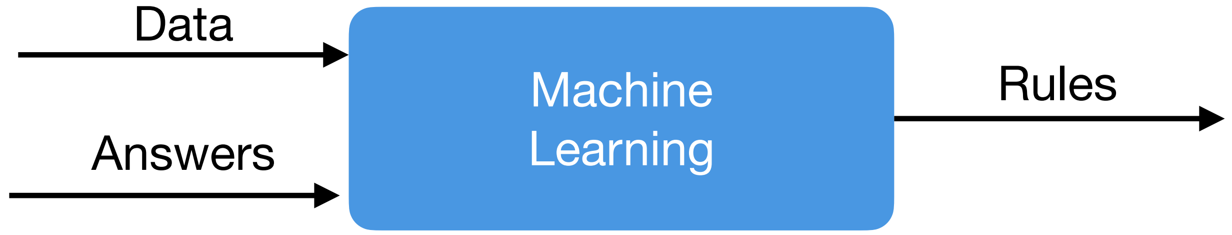
C Use occasionally

D Use regularly

From Classic Computing to Machine Learning

Data:	1001010110001 1010110101101 0110101011101 1010101101011 0101101011110 1001010110001 1010110101101 0110101011101 1010101101011 0101101011110	0101101010111 0110101011000 1101011010111 0101101011010 1101011110100 0101101010111 0110101011000 1101011010111 0101101011010 1101011110100	0110101001010 1100010110110 1110110101011 0101101011010 1111010101101 0110101001010 1100010110110 1110110101011 0101101011010 1111010101101
Answers:	Rock	Paper	Scissors

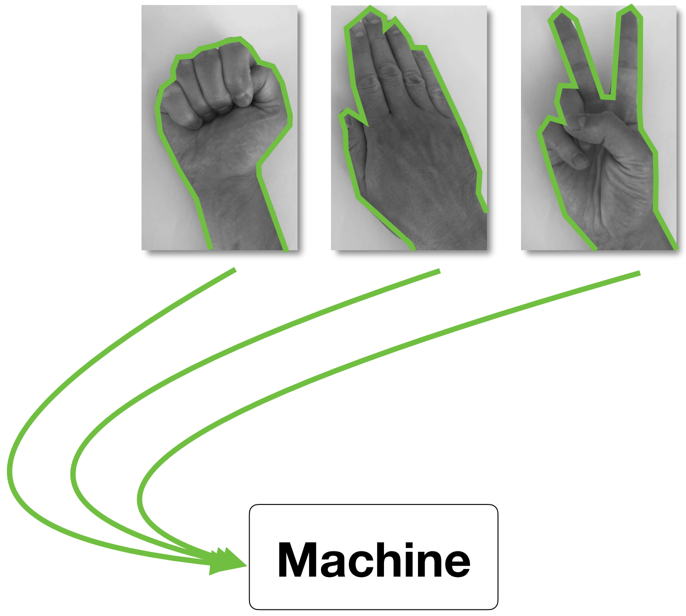




From Machine Learning to Deep Learning

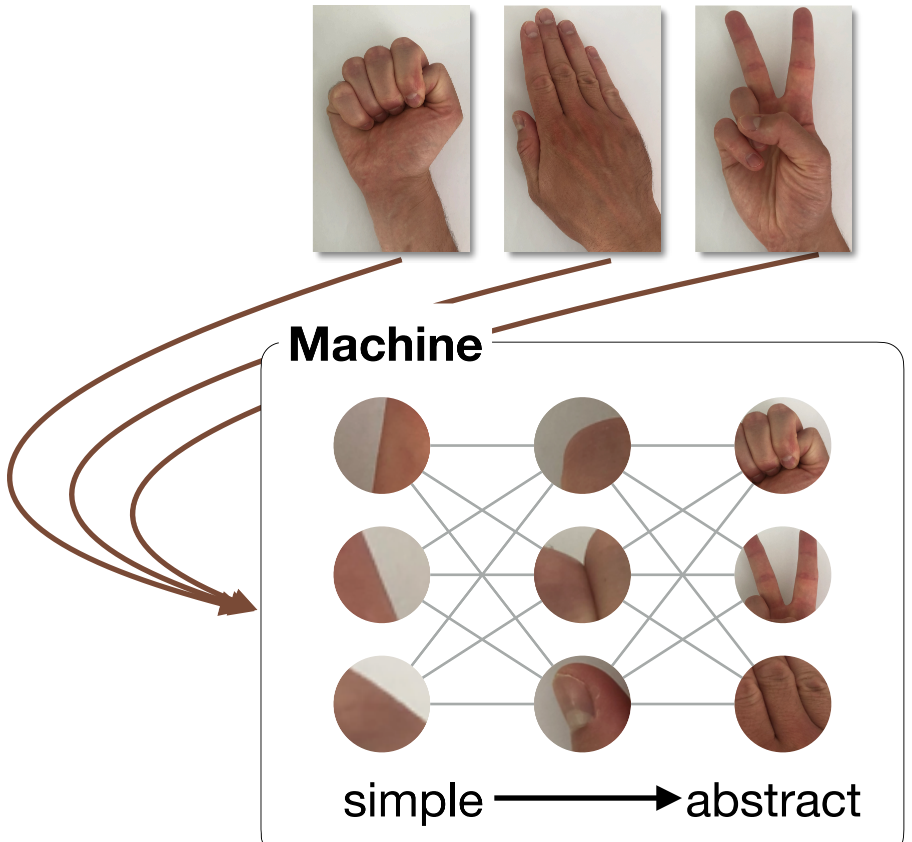
Classic Machine Learning

- Hand-engineered features
  - time consuming
  - not stable
  - not scalable

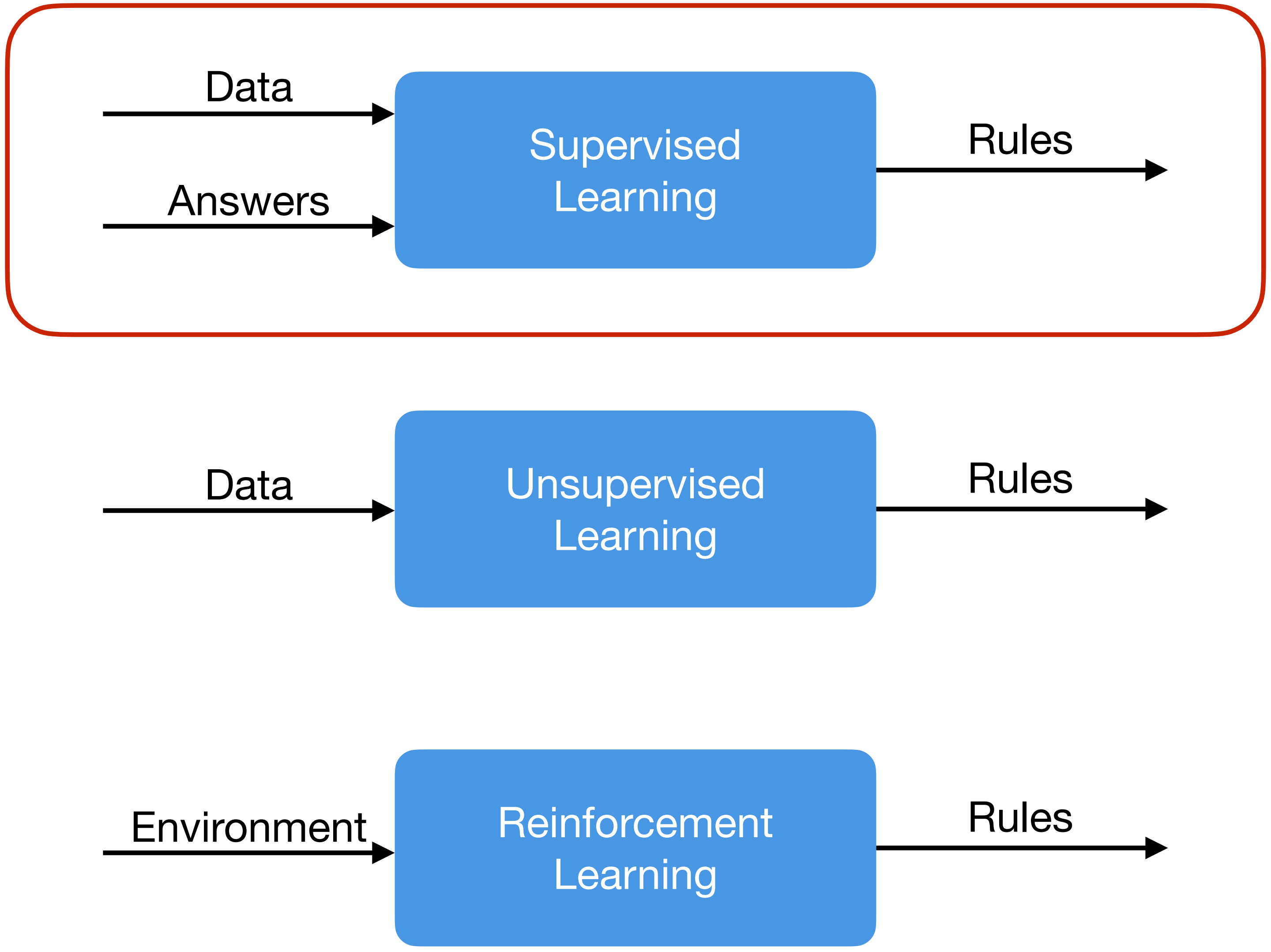


Deep Learning

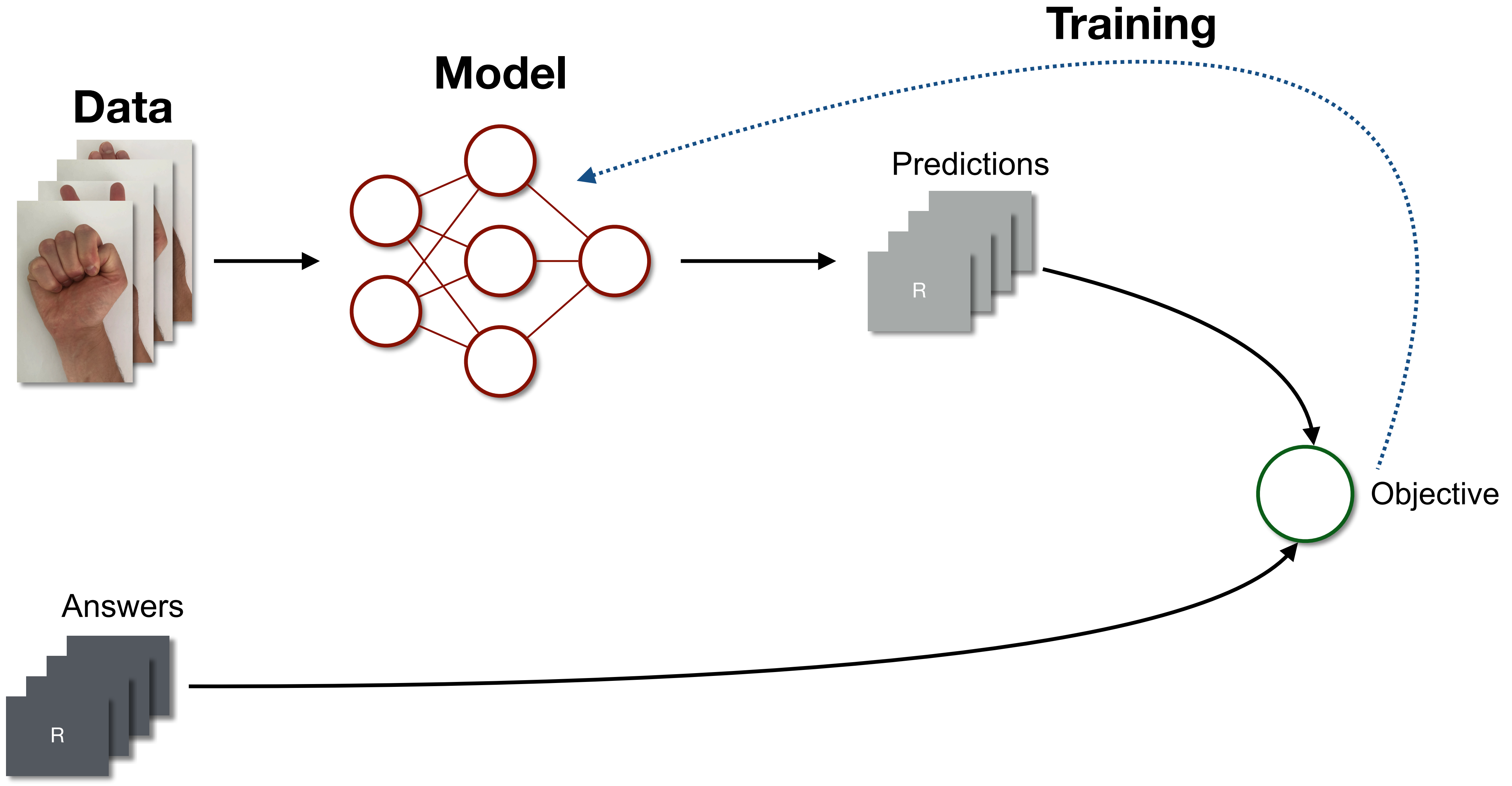
- Machine extracts features
  - fast
  - stable
  - well scalable



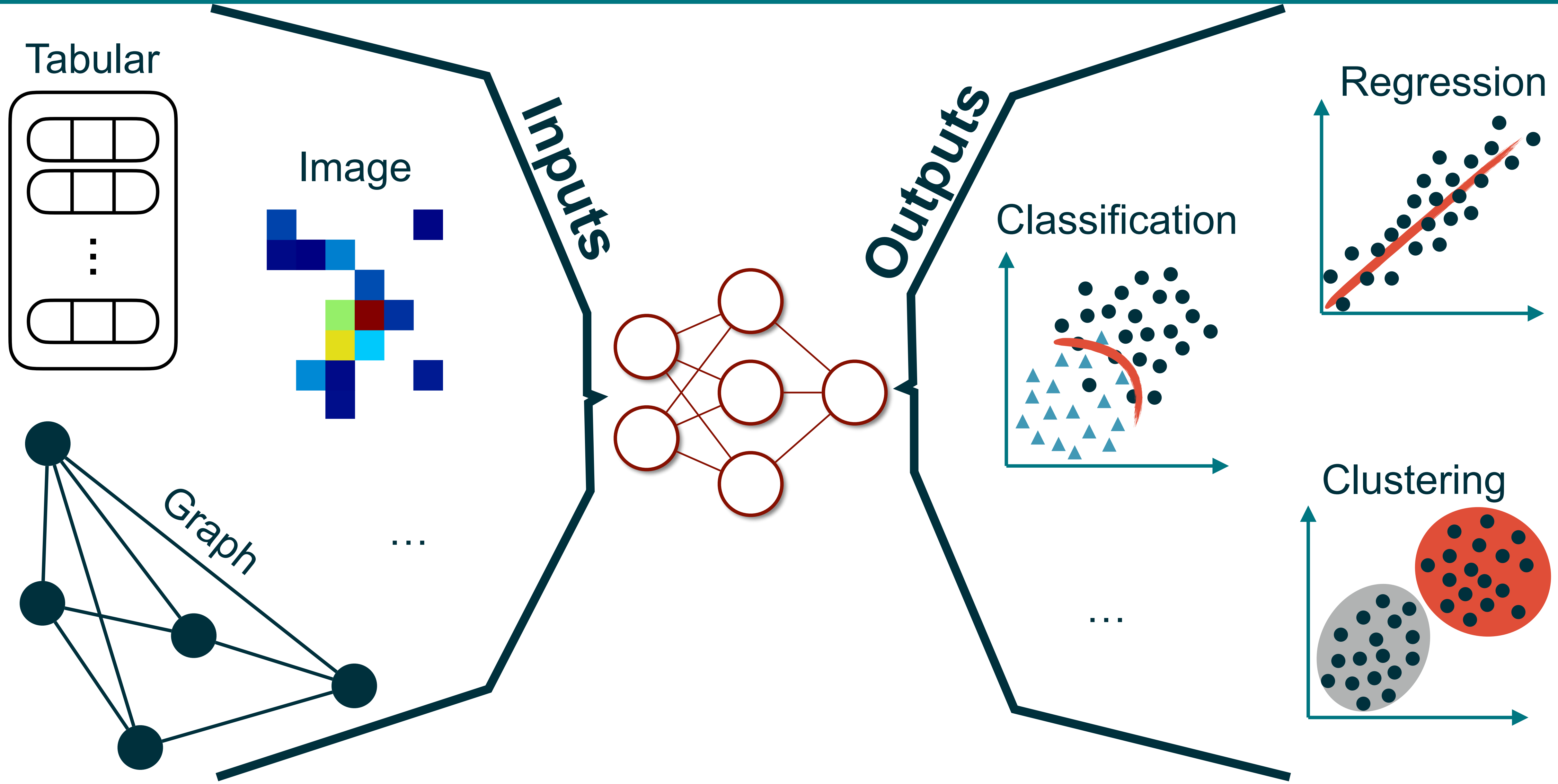
Types of Machine Learning



Supervised Machine Learning in a Nutshell



Data (Recap)

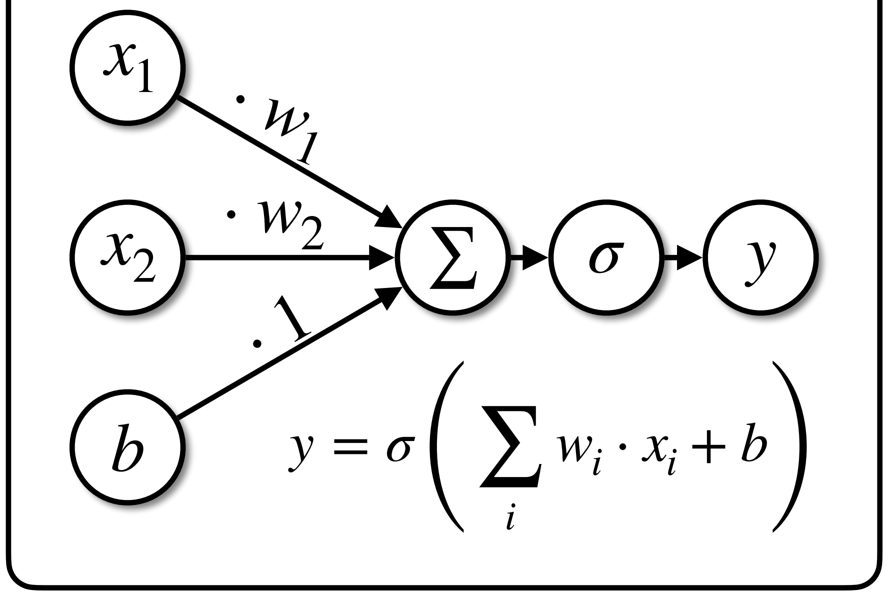

Tabular
Image
Graph
...
Inputs
Outputs
Classification
Regression
Clustering
...

Model (Recap)

- **Node (Perceptron)**: Granular unit with parameters $\theta = (W, b)$
- **Neural Network:**
  - Connected layers of multiple nodes
  - **Width**: number of nodes per layer
  - **Depth**: number of layers holding weights

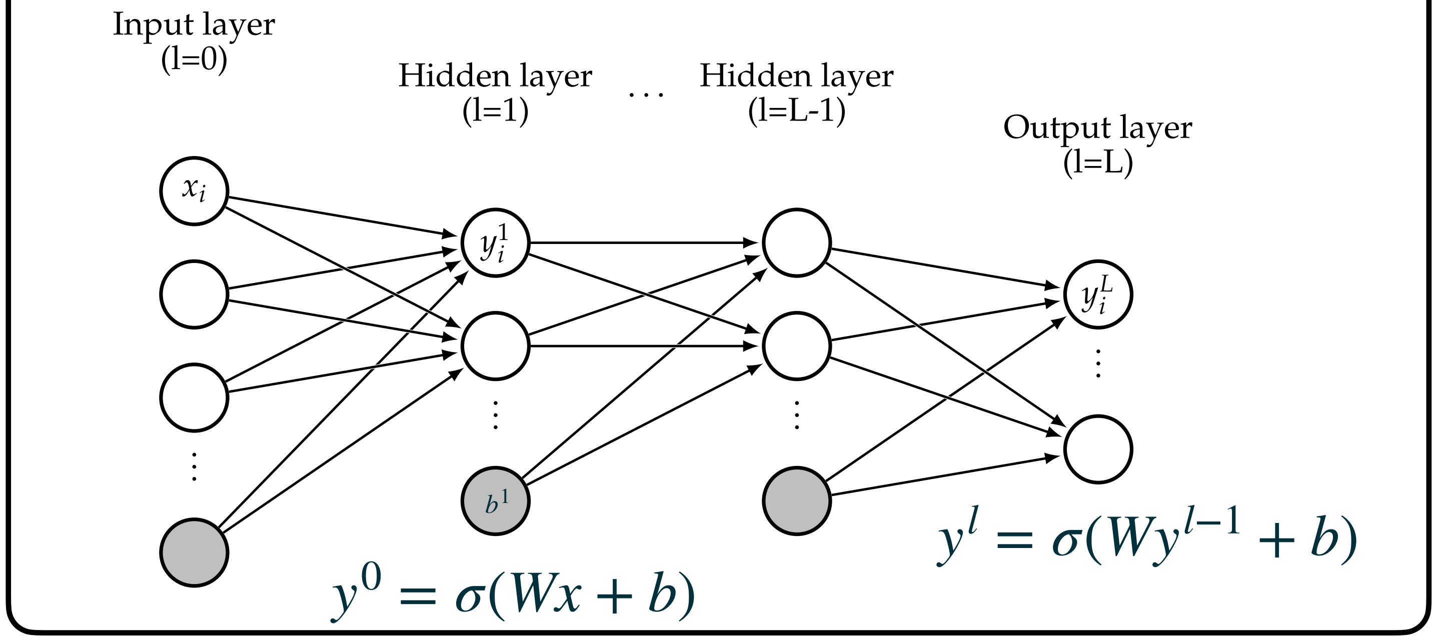
can approximate many functions!

Node



$$y = \sigma\left(\sum_i w_i \cdot x_i + b\right)$$

Neural Network



$$y^l = \sigma(Wy^{l-1} + b)$$

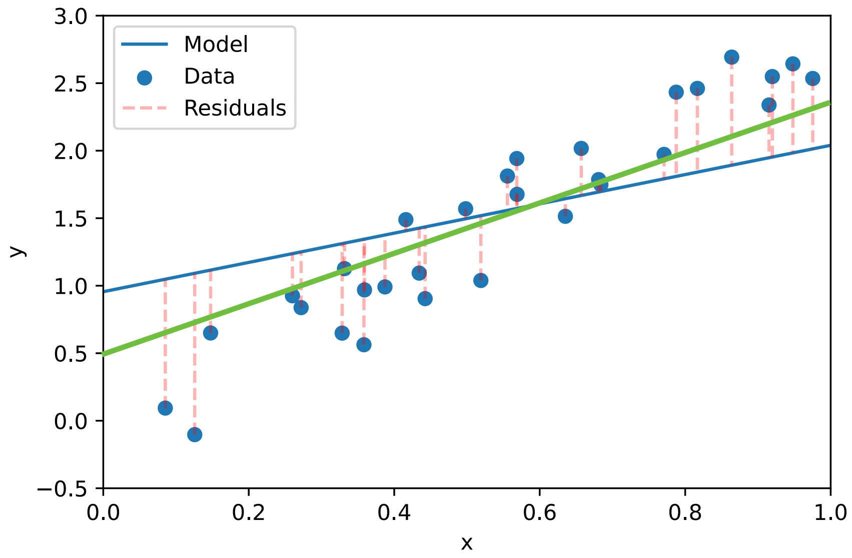
$$y^0 = \sigma(Wx + b)$$

Learning (Recap)

Objective Function

- Does parametrized model approximate the truth?
- e.g. MSE for Regression:

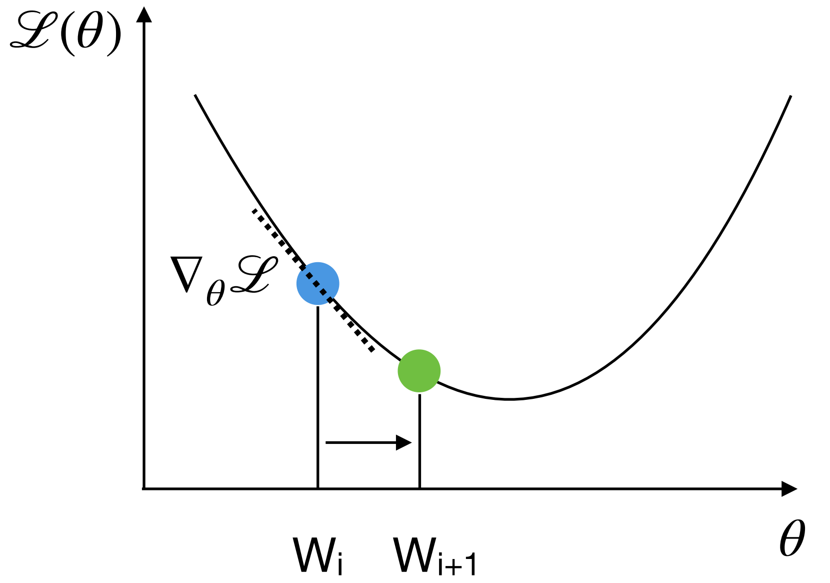
$$\mathscr{L} = \frac{1}{n} \sum_{i}^{n} (y_i^{true} - y_i^{pred}(\theta))^2$$



Parameter Update

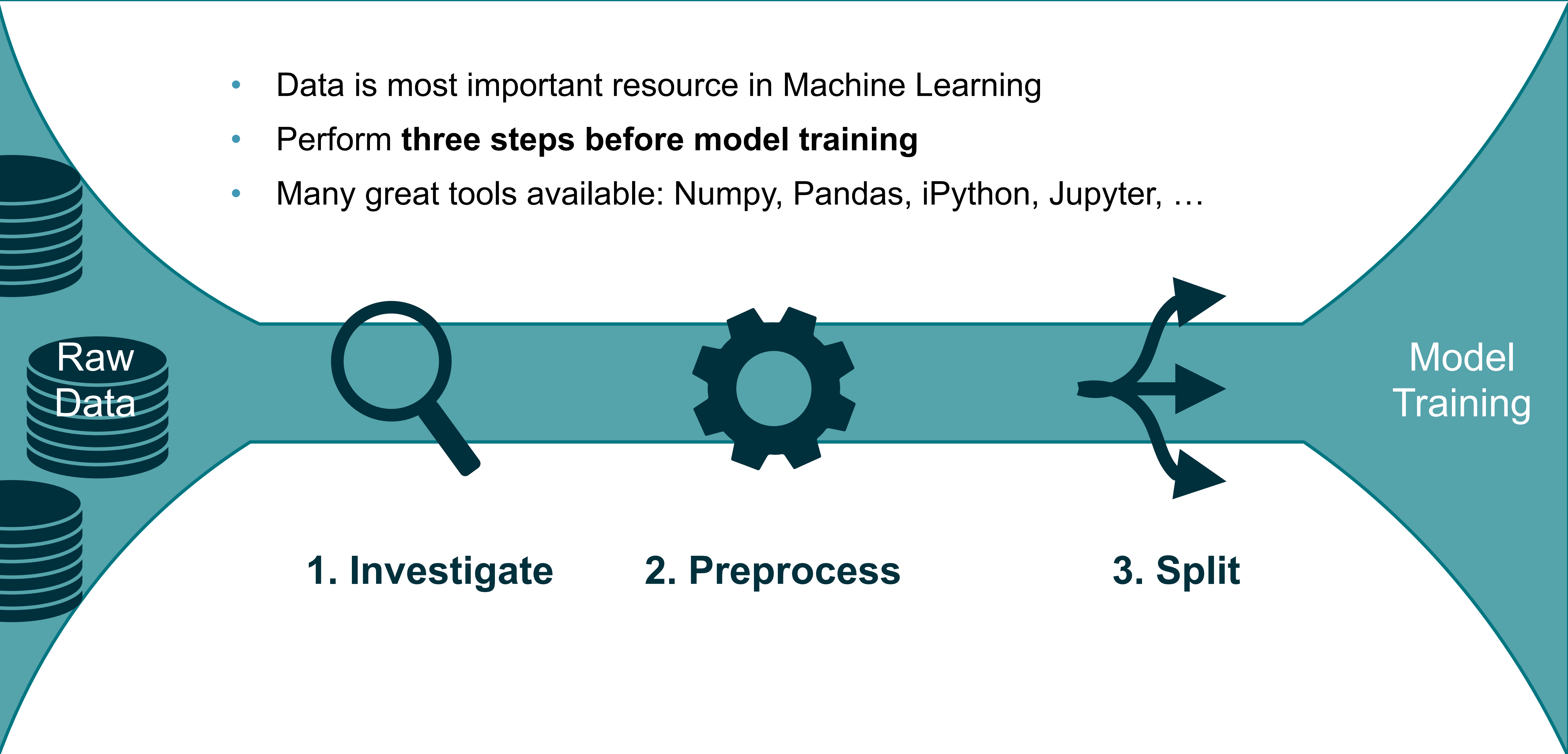
- Update parameters θ to minimize objective function
- Uses gradient descent:

$$\theta \rightarrow \theta - \underset{\text{Step size (learning rate)}}{\alpha} \underset{\text{Gradient}}{\nabla_\theta \mathscr{L}}$$



Data (NEW!)

- Data is most important resource in Machine Learning
- Perform **three steps before model training**
- Many great tools available: Numpy, Pandas, iPython, Jupyter, …



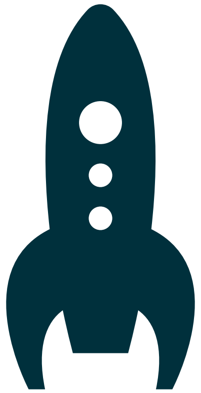
Data Investigation: Know your data!

Understand

- Origin (e.g. experiment)
- Structure (Tabular, …)
- Amount

Explore

- Numerical domain
- Trends
- Mean, Variance, Corr, …



Assess

- Outliers?
- Missing values?
- Bias in training data?

Time well spent: One more hour here might save you weeks afterwards!

Data Preprocessing: Scaling

- Typical “active” range of activation functions is (-1, 1)
- Scale your data to fit this numerical domain
- Can enable, stabilize, and accelerate training process

Min-max Scaling

$$z = \frac{x - \max}{\max - \min}$$

- **Use for**:
  - Uniform data
  - Few / no outliers

Log Scaling

$$z = \log(x)$$

- **Use for**:
  - Diff. orders of magnitude
  - Some heavy outliers

Z Score

$$z = \frac{x - \mu}{\sigma}$$

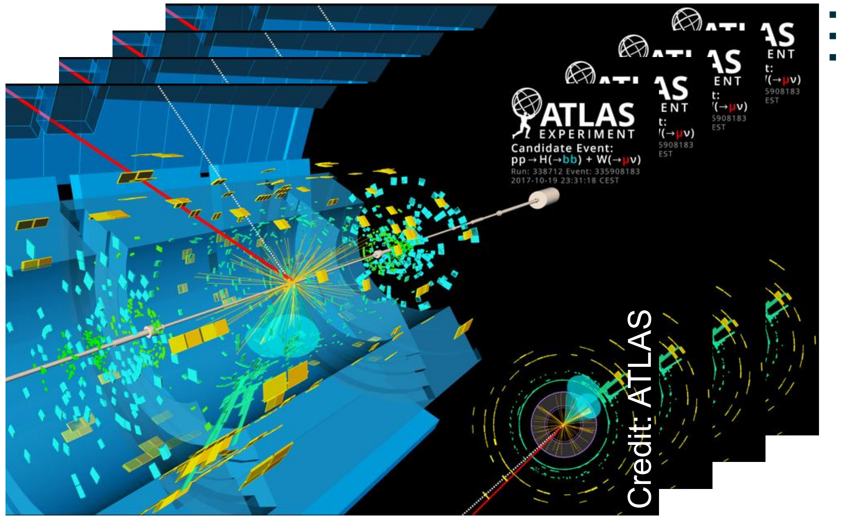
- **Use for**:
  - Gaussian shaped
  - Few outliers

Data Preprocessing: Missing Values

- Recorded data often has missing values (e.g. limited amount of jets in HEP event)
- Strategies:
  - **Delete Rows / Columns**: Easy, but can loose a lot of important information
  - **Fill values with mean / mode / interpolation**: May not be applicable
  - **Fill values with categorical value (e.g. 0, -1)**: Involves educated guess

Use what works!

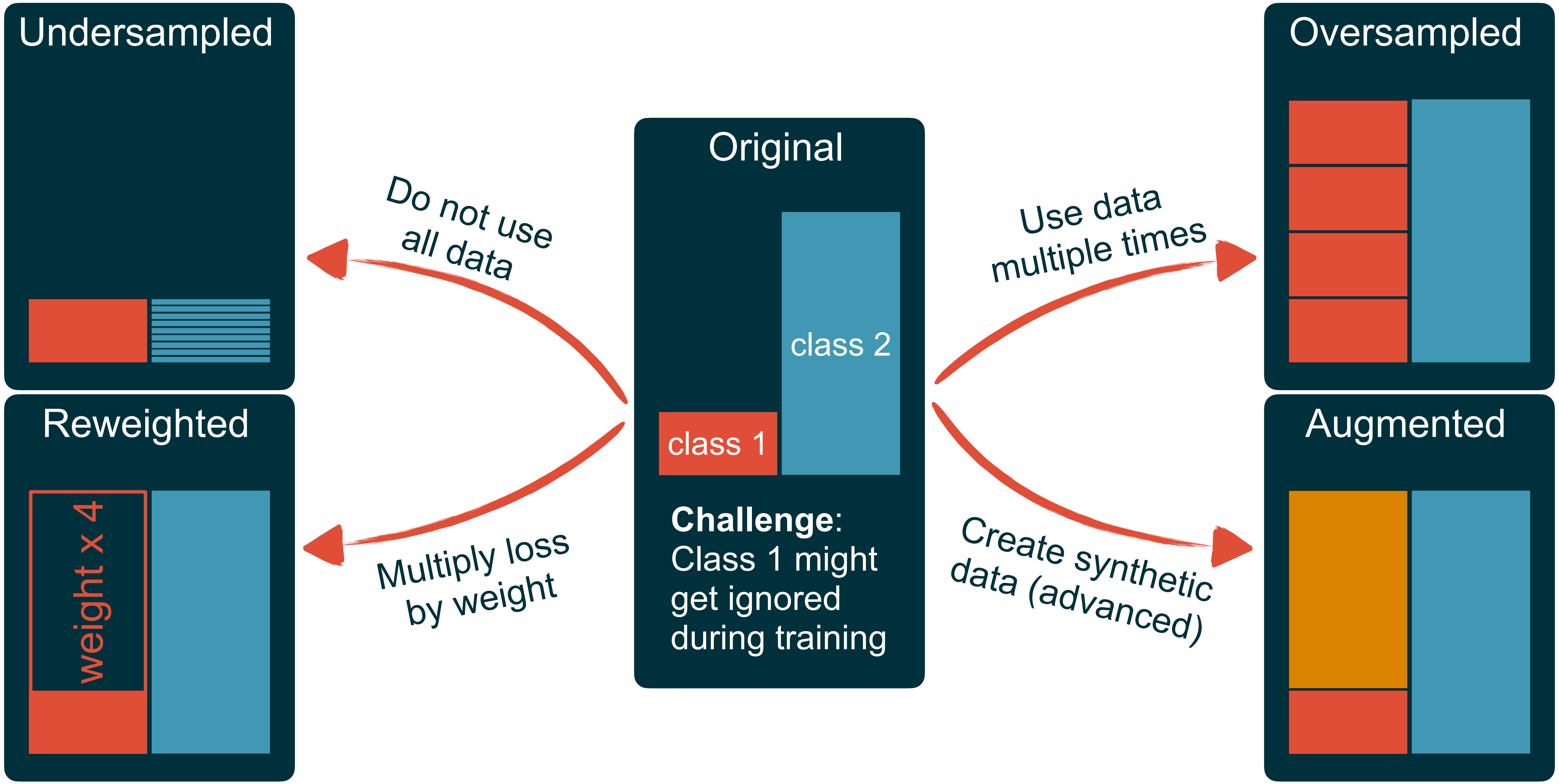
Detected Events



Training Data

	Jet 1 p_T	Jet 1 η	Jet 2 p_T	Jet 2 η
Event #1	100	1.5	80	0.2
Event #2	250	2.1	180	1.1
Event #3	180	0.3	-	-
Event #4	200	0.8	-	-
Event #5	170	1.2	100	0.1
Event #6	210	0.5	-	-

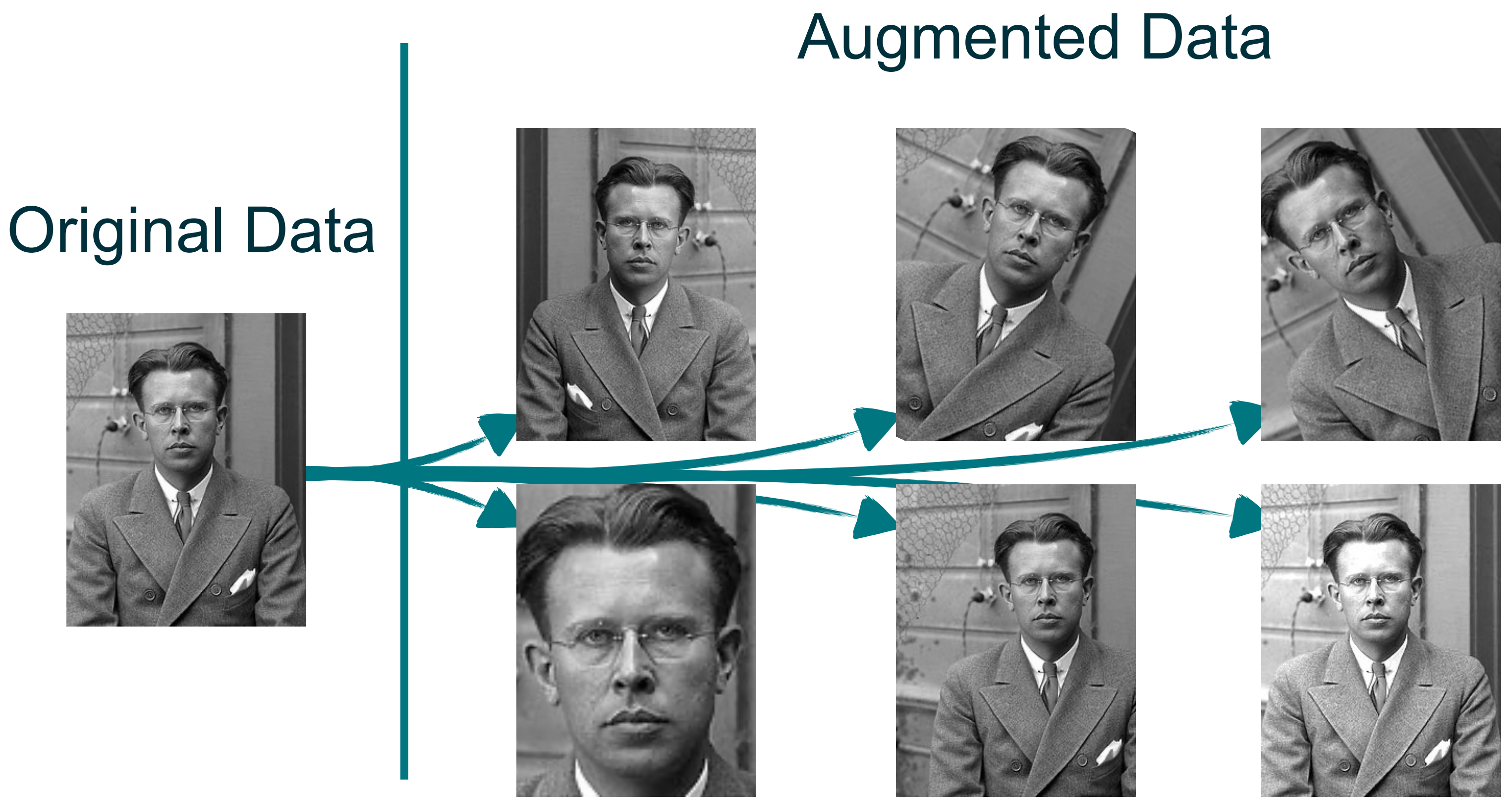
Data Preprocessing: Class Imbalance



Challenge: Class 1 might get ignored during training

Data Preprocessing: Augmentation

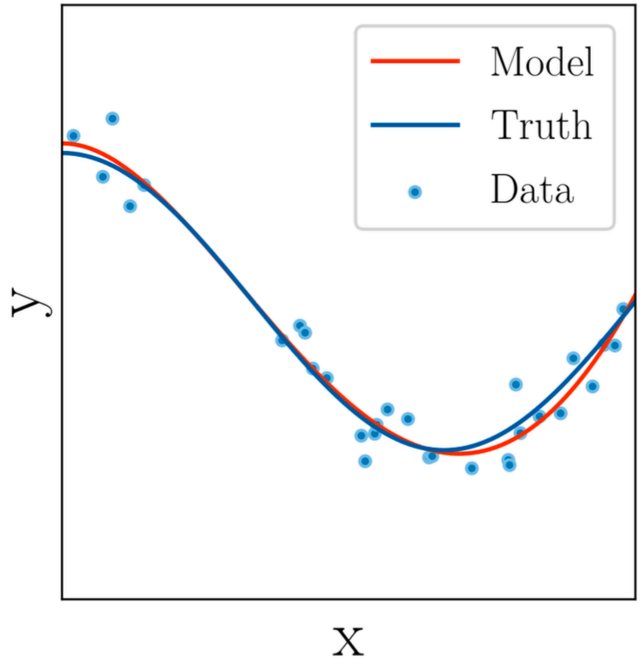
- Create synthetic samples by modifying samples in existing dataset by small changes
- Need to have understanding of meaningful symmetries
- E.g. in physics: Shift measurement within uncertainties, rotate cosmic showers, …
- Other methods: High-dimensional interpolation (e.g. kNN-based augmentation SMOTE), …

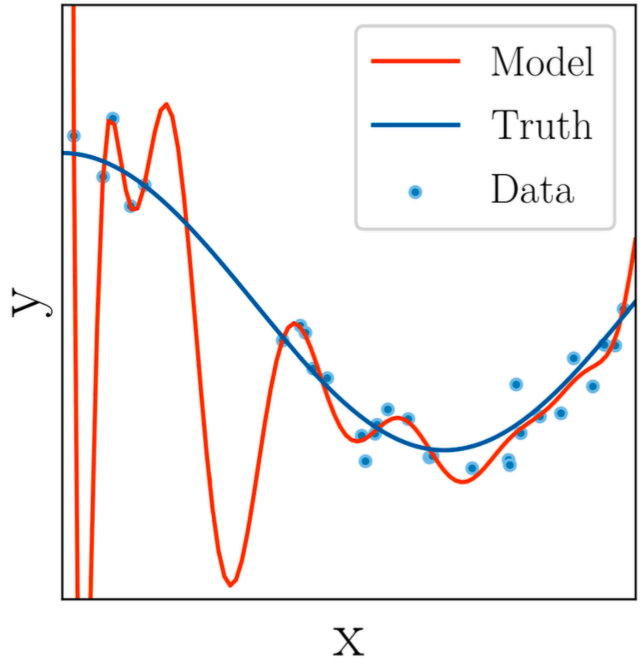


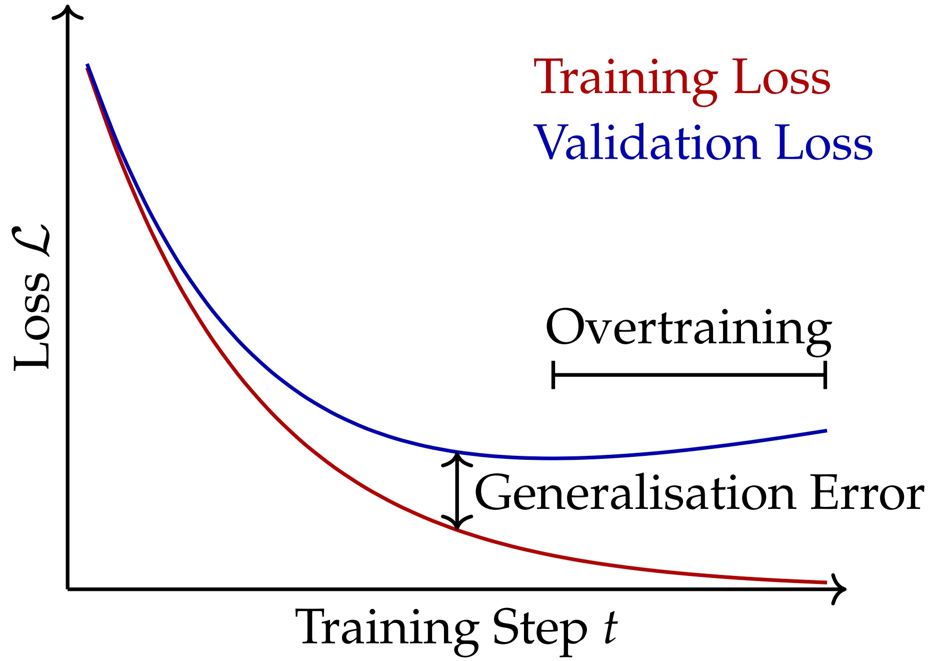
[1]

Data Splitting: Motivation

- A complex model can be fitted to any function if trained long enough
- It might perform great on training data but not generalize (**overtraining**)
- Need measure to prevent this!



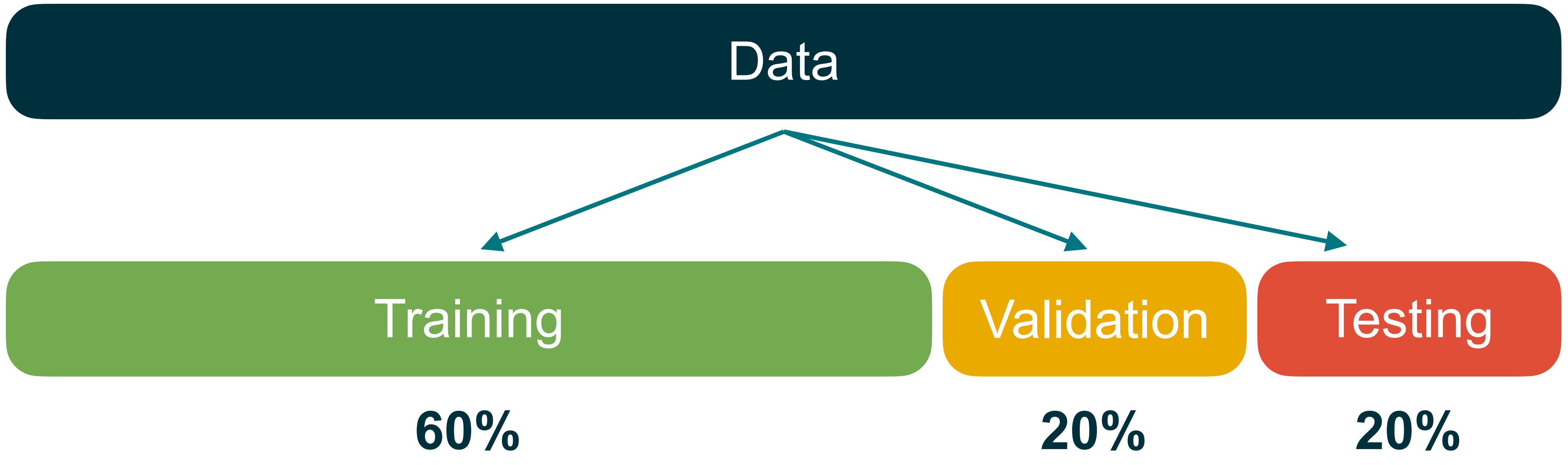




[2]

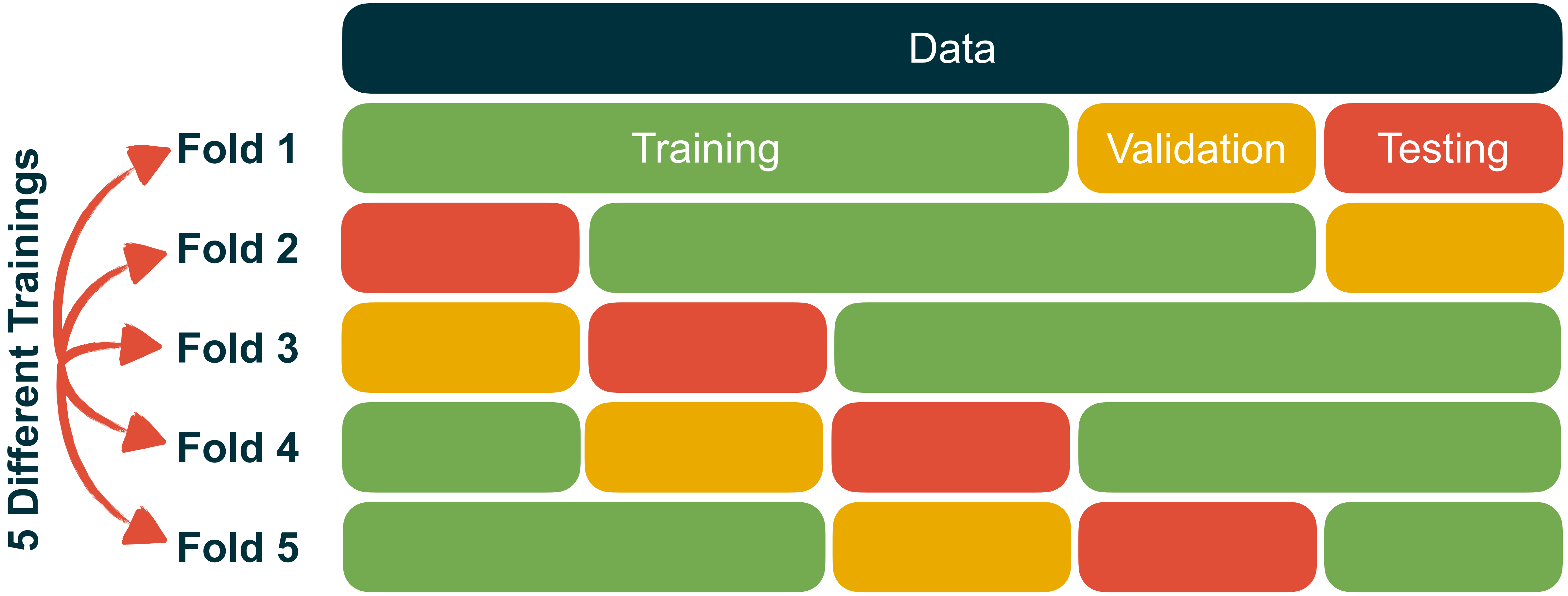
Data Splitting: Train-Val-Test Split

- Split data into three parts:
  - Training set: Train parameters of the model
  - Validation set: Monitor and tune training procedure (e.g. learning rate)
  - Test set: Estimate final performance, use only once!
- Pro-tip: Use deterministic splitting via event identifier (e.g. event number from simulation)



Data Splitting: Cross Validation

- Do not want to loose measured data due to splitting
- Would be better if we could use all measured data in final evaluation
- Solution: Use **rotating cross-validation to train multiple different (independent) models**!



Introduction to Machine Learning: Part II - Take Away

- Know your data (an hour here can save you weeks!)
- Preprocess your data (fix outliers, missing values, normalize, augment)
- Split your data (train val test) before you do anything else!

Model (NEW!)

- **Theory**: A one-layer perceptron can approximate any function with arbitrary precision
- **Reality**: Shallow neural networks often hard to train, advanced architectures much better!
- **Use type of model according to data (type, structure, symmetry, and complexity)**

Feed Forward
CNN
ResNET
RNN
Graph
LSTM
Transformer

Before: Simple Architectures

Now: Advanced Architectures

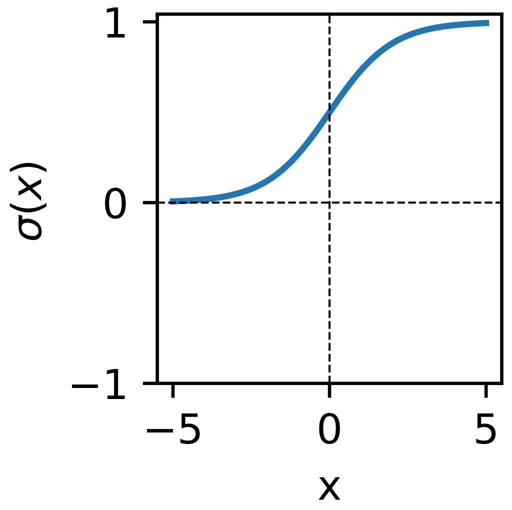
- More resource/parameter efficient
- Easier to train
- Converge faster

Activation Functions

- Activation functions bring non-linearity to models
- Many different possibilities enable complex inner network representations

Logistic function

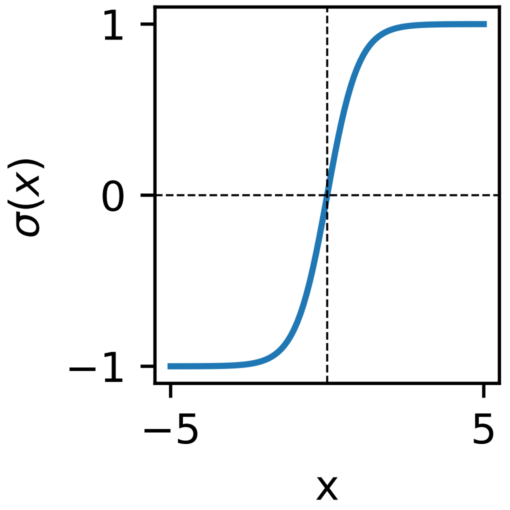
$$\sigma(x) = \frac{1}{1+e^{-x}}$$



Good for probabilities!

Tanh

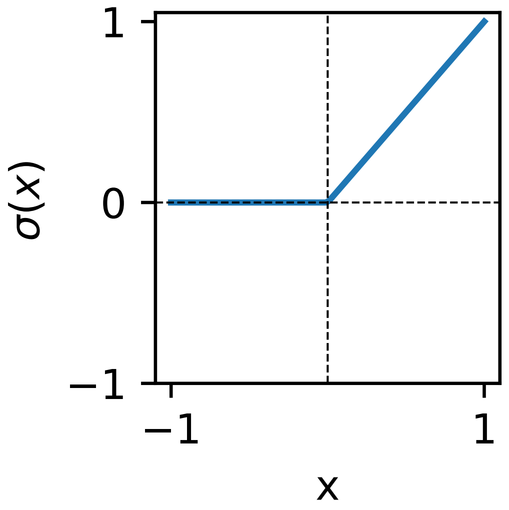
$$\sigma(x) = \frac{e^{2x}-1}{e^{2x}+1}$$



Positive & negative outputs!

ReLU

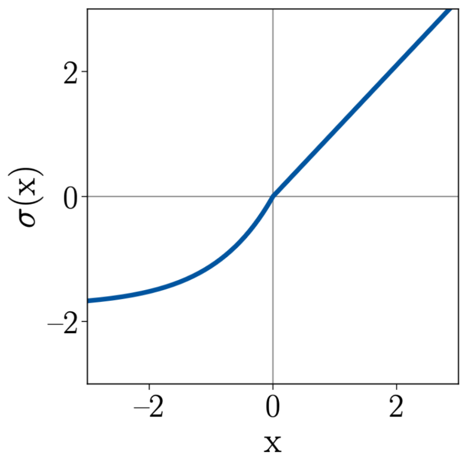
$$\sigma(x) = \max(0,x)$$



Very easy and fast!

SELU

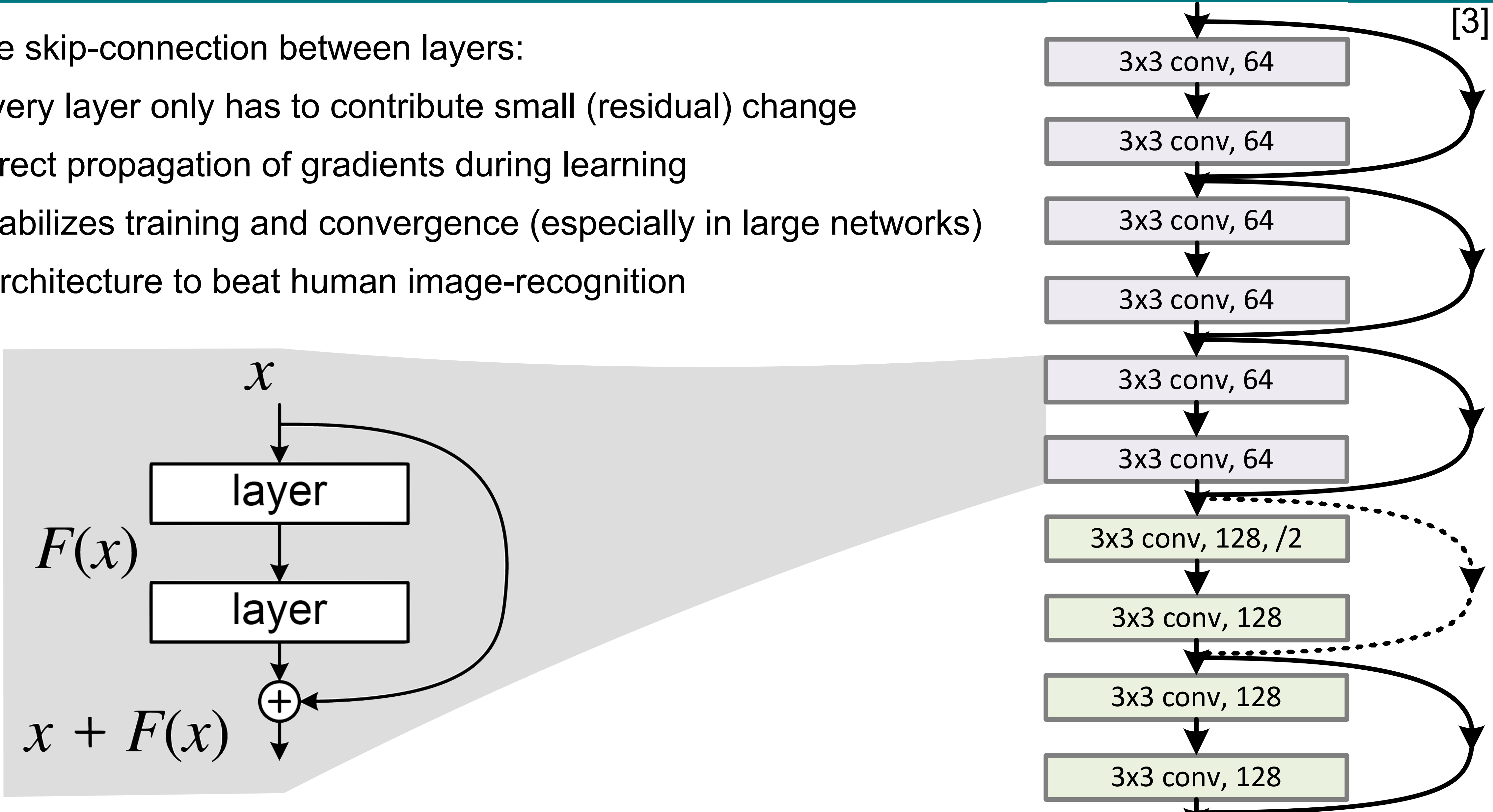
$$\sigma(x) = \begin{cases} \lambda x & |x > 0 \\ \lambda\alpha(e^x - 1) & |x \le 0 \end{cases}$$



Self-normalizing!

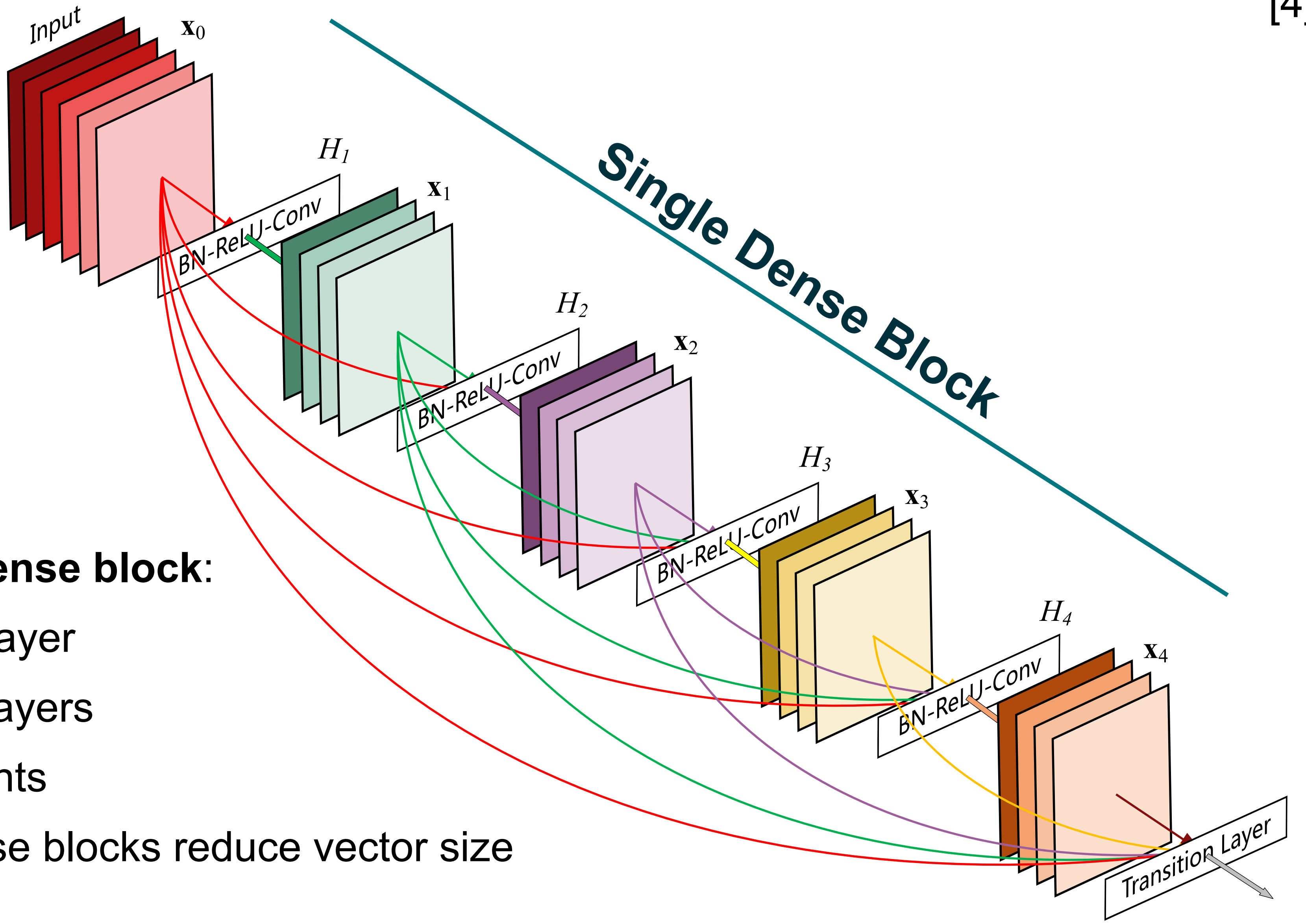
Residual Neural Network

- Include skip-connection between layers:
  - Every layer only has to contribute small (residual) change
  - Direct propagation of gradients during learning
  - Stabilizes training and convergence (especially in large networks)
- First architecture to beat human image-recognition



Densely Connected Networks

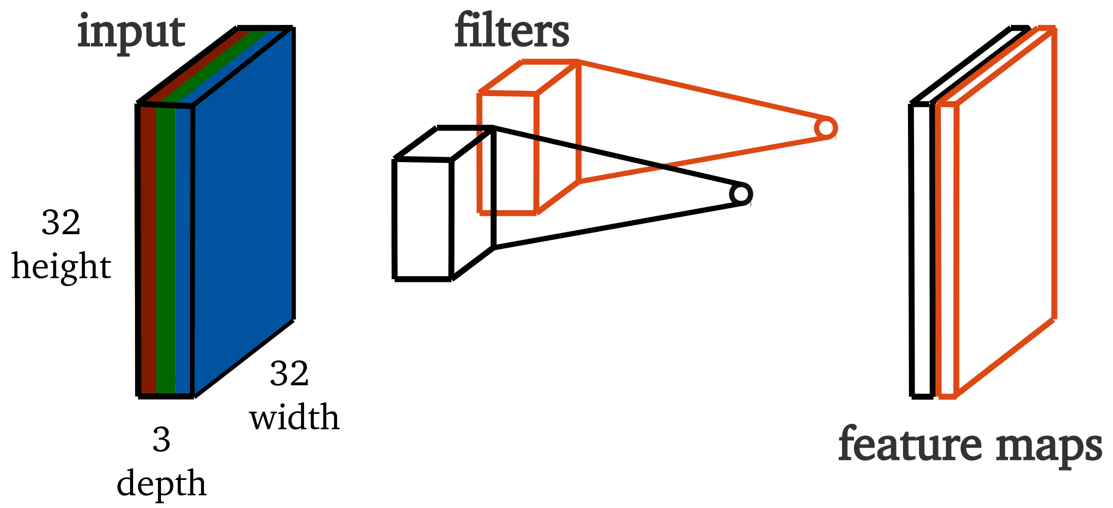
[4]



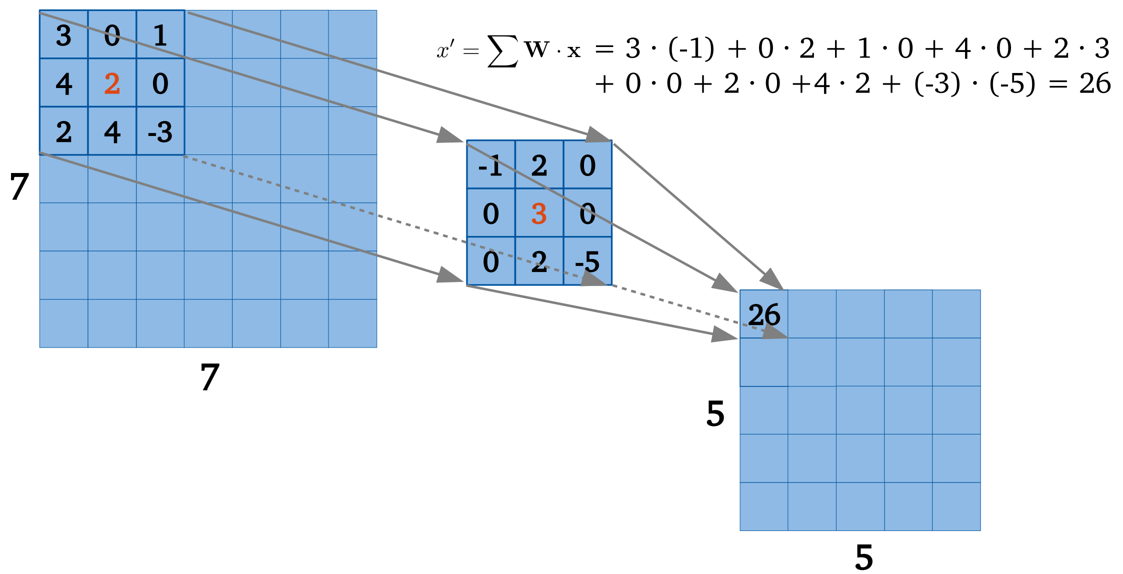
- Apply shortcut to all layers in **dense block**:
  - Reuse features from each layer
  - Combine features from all layers
  - Easy propagation of gradients
- Transition Layers between dense blocks reduce vector size

Convolutional Neural Network (1/2)

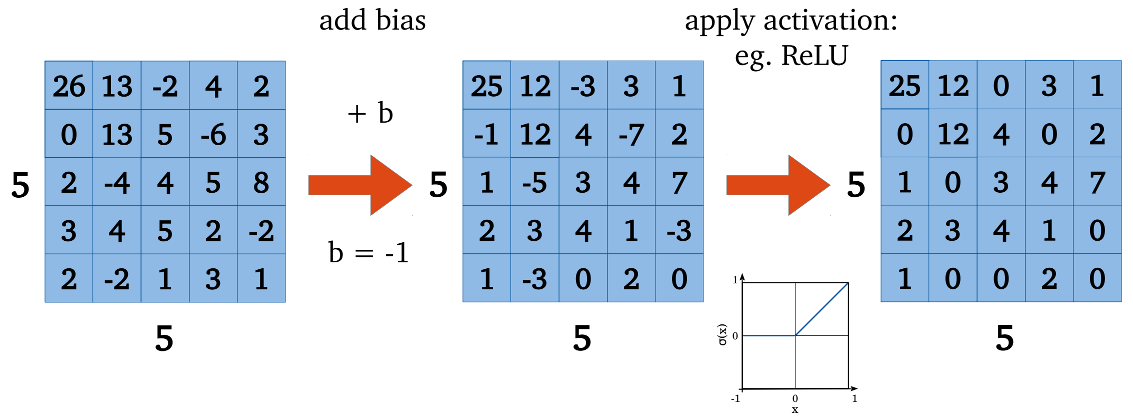
- For structured, geometrical data (e.g. images)
- Instead of weights now have 'filters':
  - Slided over data (translational invariant)
  - Each filter extracts a feature



Step by step:



$$x' = \sum \mathrm{W} \cdot \mathrm{x} = 3 \cdot (-1) + 0 \cdot 2 + 1 \cdot 0 + 4 \cdot 0 + 2 \cdot 3 + 0 \cdot 0 + 2 \cdot 0 + 4 \cdot 2 + (-3) \cdot (-5) = 26$$



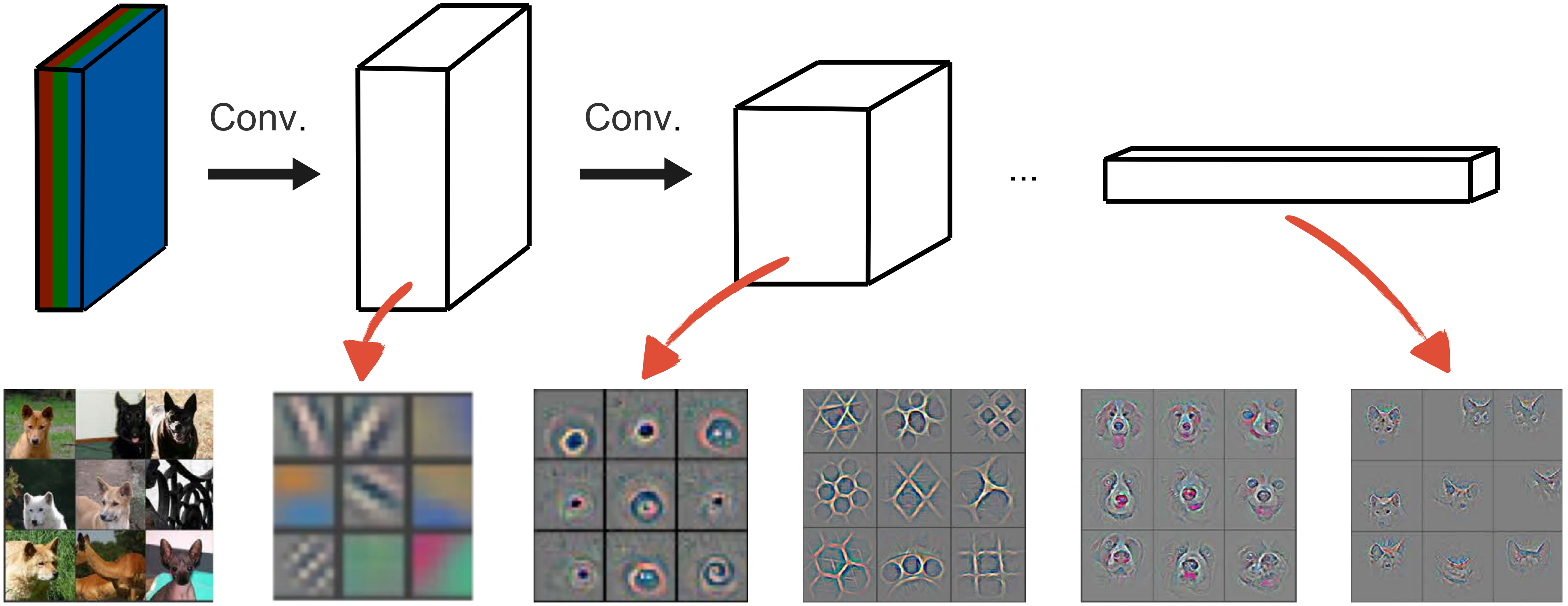
$$\begin{bmatrix} 26 & 13 & -2 & 4 & 2 \\ 0 & 13 & 5 & -6 & 3 \\ 2 & -4 & 4 & 5 & 8 \\ 3 & 4 & 5 & 2 & -2 \\ 2 & -2 & 1 & 3 & 1 \end{bmatrix} \xrightarrow[b\,=\,-1]{+\,b} \begin{bmatrix} 25 & 12 & -3 & 3 & 1 \\ -1 & 12 & 4 & -7 & 2 \\ 1 & -5 & 3 & 4 & 7 \\ 2 & 3 & 4 & 1 & -3 \\ 1 & -3 & 0 & 2 & 0 \end{bmatrix} \xrightarrow{} \begin{bmatrix} 25 & 12 & 0 & 3 & 1 \\ 0 & 12 & 4 & 0 & 2 \\ 1 & 0 & 3 & 4 & 7 \\ 2 & 3 & 4 & 1 & 0 \\ 1 & 0 & 0 & 2 & 0 \end{bmatrix}$$

[2]

Convolutional Neural Network (2/2)

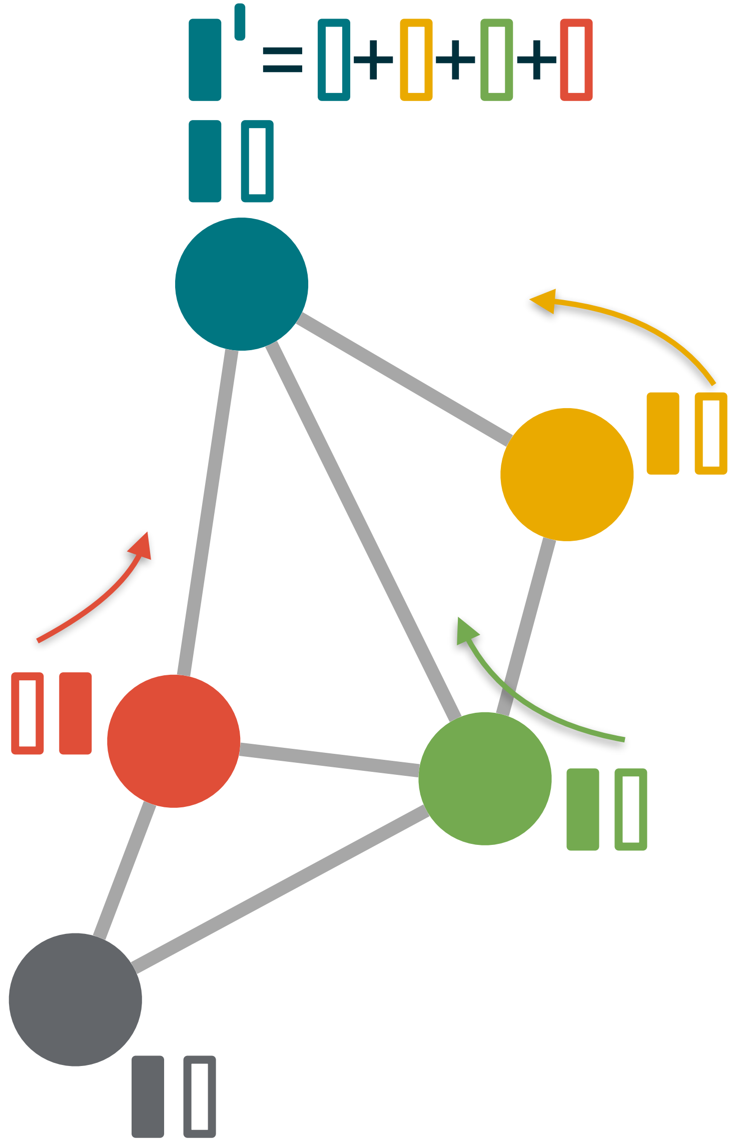
[5]

- Multiple layers of filters extract more-and-more abstract features
- Usually have pyramidal shape: Decrease spatial extent & increase feature space



Learning on Graphs (1/2)

- **Graph:**
  - Nodes: Have features
  - Edges: Connect nodes, can have features
- **Learning by updating each node:**
  - Embed neighbors
  - Aggregate embebbings (permutation invariant.)
  - Embed aggregations



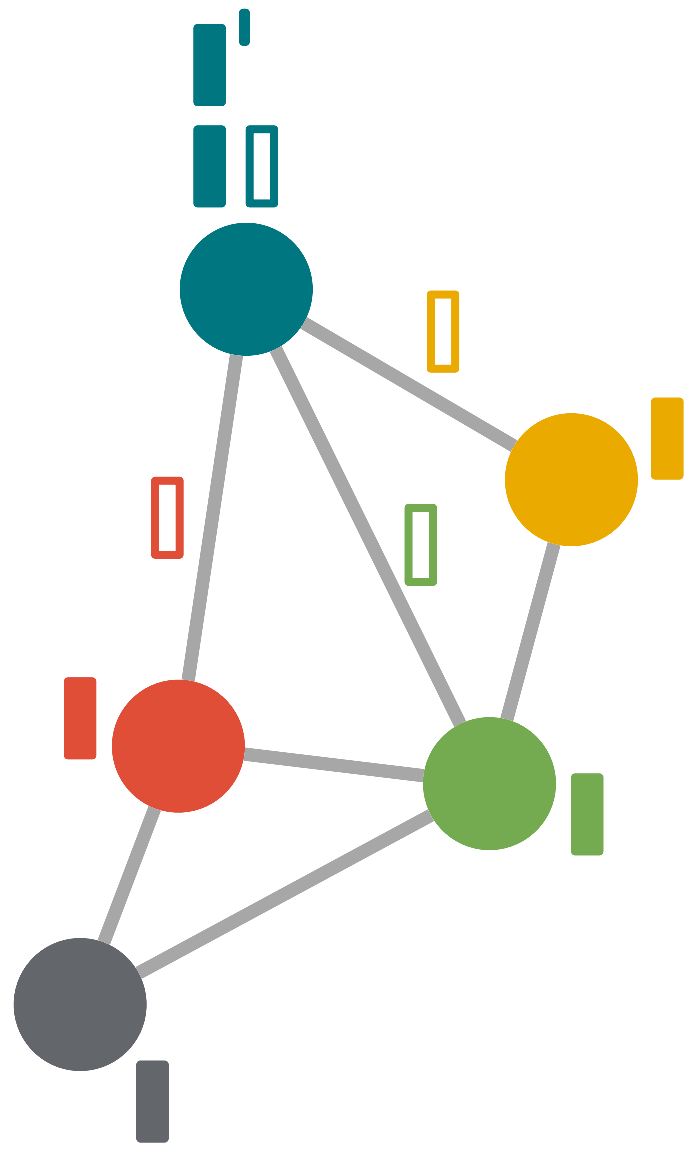
[6]

Learning on Graphs (2/2)

- **Graph:**
  - Nodes: Have features 🟦🟨🟥🟩
  - Edges: Connect nodes, can have features —
- **Learning by updating each node:**
  - Embed edges ▯ = ϕ(🟨) - Isotropic

    ▯ = ϕ(🟨, 🟦) - Anisotropic
  - Aggregate embebbings $\oplus$ ▯,▯,▯
  - Embed aggregations 🟦' = ψ(🟦, $\oplus$ ▯,▯,▯)

$\boxed{\text{ϕ and ψ can be DNNs!}}$ $\boxed{\text{use multiple rounds k}}$

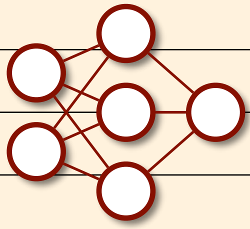


[6]

Introduction to Machine Learning: Part II - Take Away

- Know your data (an hour here can save you weeks!)
- Preprocess your data (fix outliers, missing values, normalize, augment)
- Split your data (train val test) before you do anything else!

- Use an appropriate architecture: many different options
- My personal start: 3 layers, 256 nodes, ReLU activation

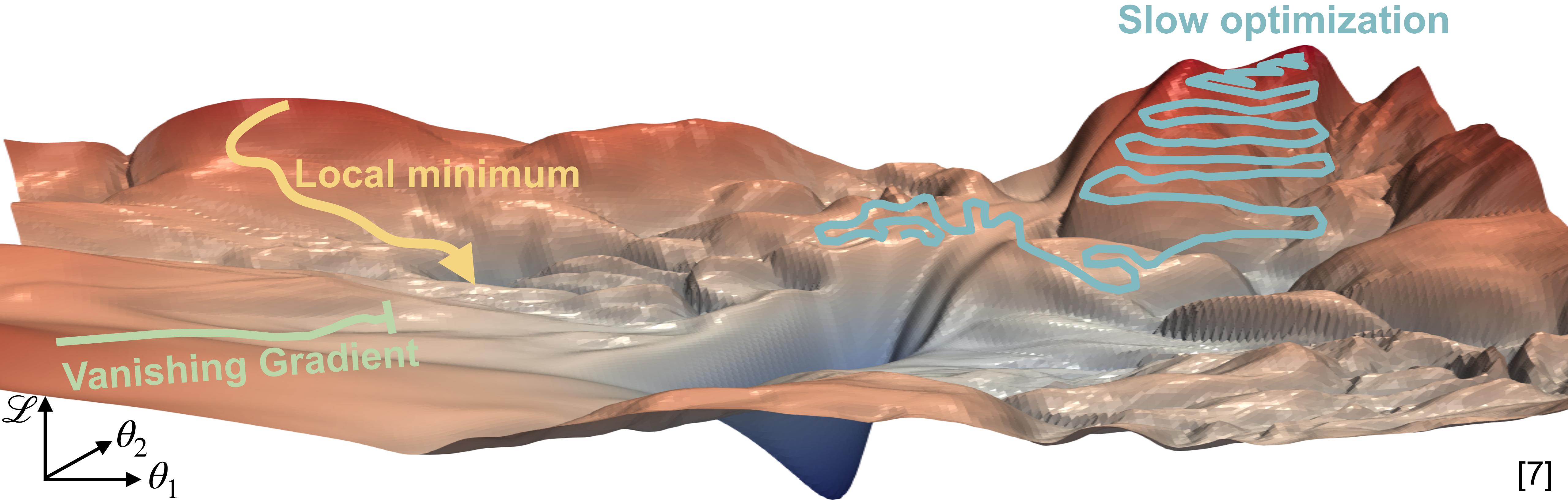


Training (NEW!)

- Loss landscape can look very complicated (e.g. local minima)
- At each step only evaluate loss $\mathscr{L}$ and gradient
- Many possible failure modes (- - -)

Reminder:

Gradient Descent $\theta \rightarrow \theta - \alpha \nabla_\theta \mathscr{L}$



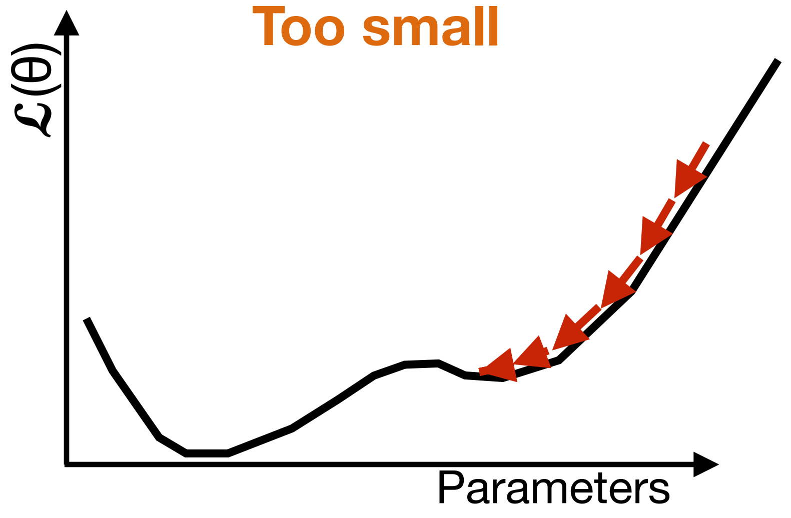
[7]

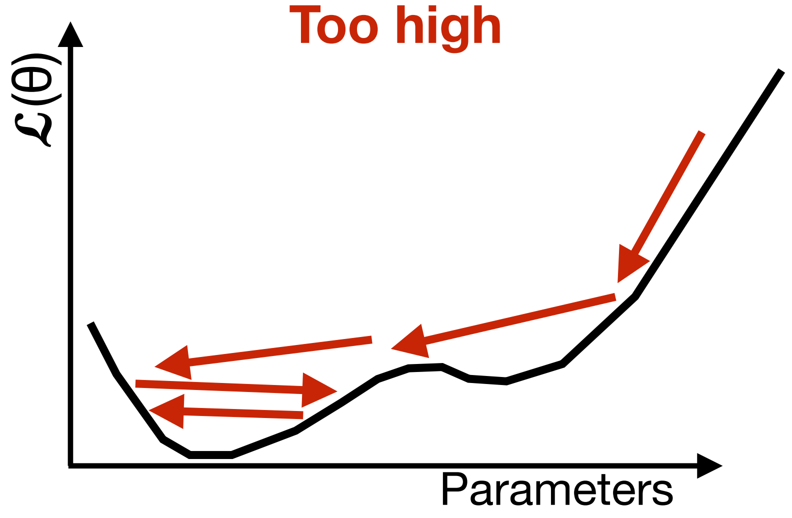
Learning rate

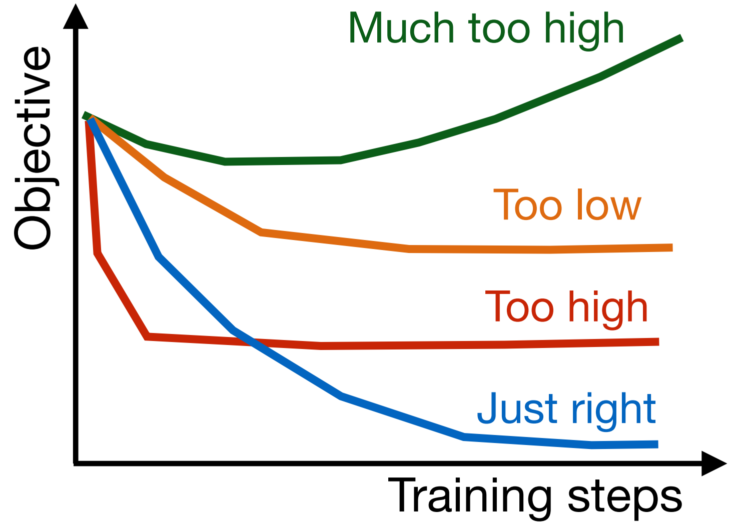
- Want training to converge smoothly and avoid local minima
- Learning rate α instrumental for success
- Can decrease learning rate during training:
  - e.g. exponential with steps or on-plateau

Reminder:

Gradient Descent $\theta \rightarrow \theta - \alpha \nabla_\theta \mathscr{L}$







Stochastic Gradient Descent (SGD)

- Until now: Calculation of loss and gradient based on whole dataset
- New idea: Approximate loss and gradient on subset of dataset (mini-batch)

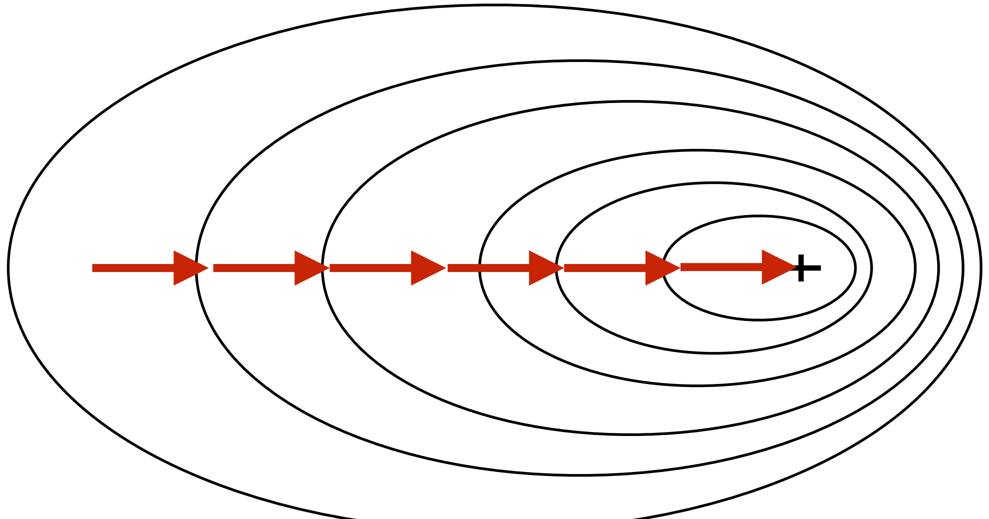
Pro

- More parameter updates
- Stochasticity helps escape local minima

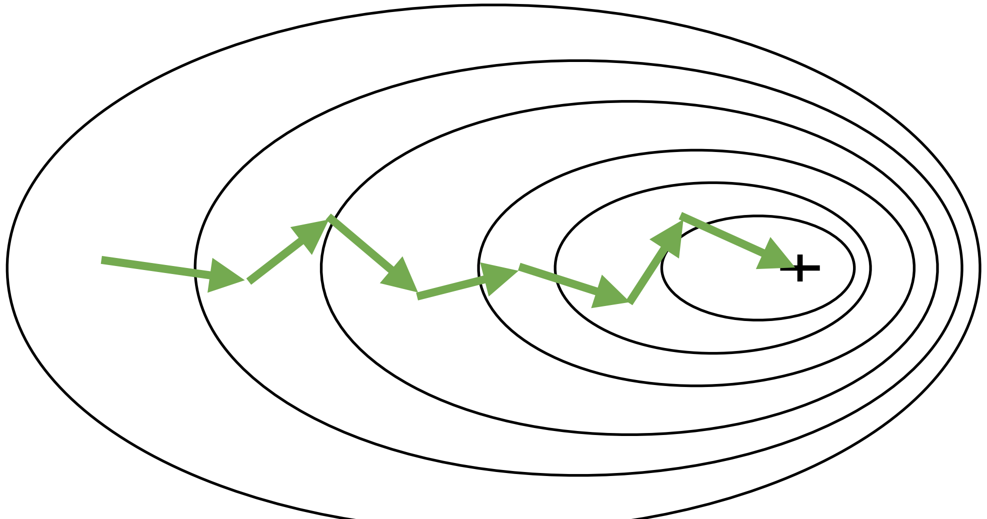
Contra

- Gradient not exact (however in practice good enough)

Gradient Descent:



Stochastic Gradient Descent:



Advanced Optimization Algorithms

Slide inspired by M. Rieger [8]

Momentum

Maintain velocity or previous updates: stable

$$\Delta\theta_t = m_t = \gamma \cdot m_{t-1} + (1-\gamma) \cdot \alpha \cdot \frac{d\mathcal{L}}{d\theta}$$

Adagrad

Remember past gradients and adapt $\alpha \rightarrow \alpha_t$: adaptive

$$\alpha_t = \frac{\alpha}{\sqrt{v_t} + \epsilon} \qquad v_t = \sum_{\tau=1}^{t} \left(\frac{\partial\mathcal{L}}{\partial\theta_\tau}\right)^2$$

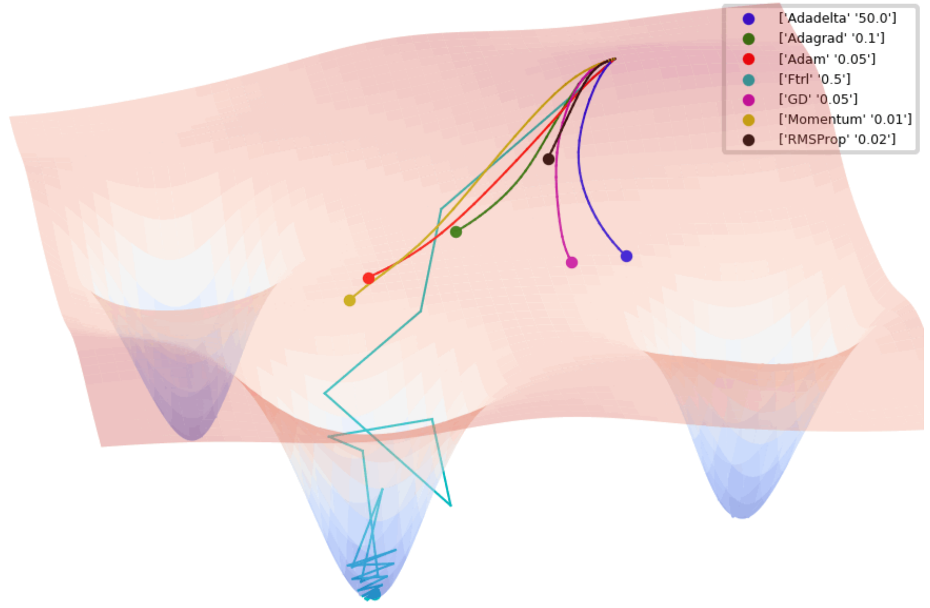
RMSprob

Decay memory of past gradients: good to train longer

$$\alpha_t = \frac{\alpha}{\sqrt{v_t} + \epsilon} \qquad v_t = \beta \cdot v_{t-1} + (1-\beta) \cdot \left(\frac{\partial\mathcal{L}}{\partial\theta_t}\right)^2$$

Reminder:

Gradient Descent $\theta_{t+1} \rightarrow \theta_t - \Delta\theta_t$



[9]

Adam

Combines Momentum and RMSprob:

$$\Delta\theta_t = \alpha \cdot \frac{m_t}{\sqrt{v_t} + \epsilon} \qquad m_t = \frac{1}{1-\gamma^t}\left[\gamma \cdot m_{t-1} + (1-\gamma) \cdot \frac{\partial\mathcal{L}}{\partial\theta_t}\right] \qquad v_t = \frac{1}{1-\beta^t}\left[\beta v_{t-1} + (1-\beta) \cdot \left(\frac{\partial\mathcal{L}}{\partial\theta_t}\right)^2\right]$$

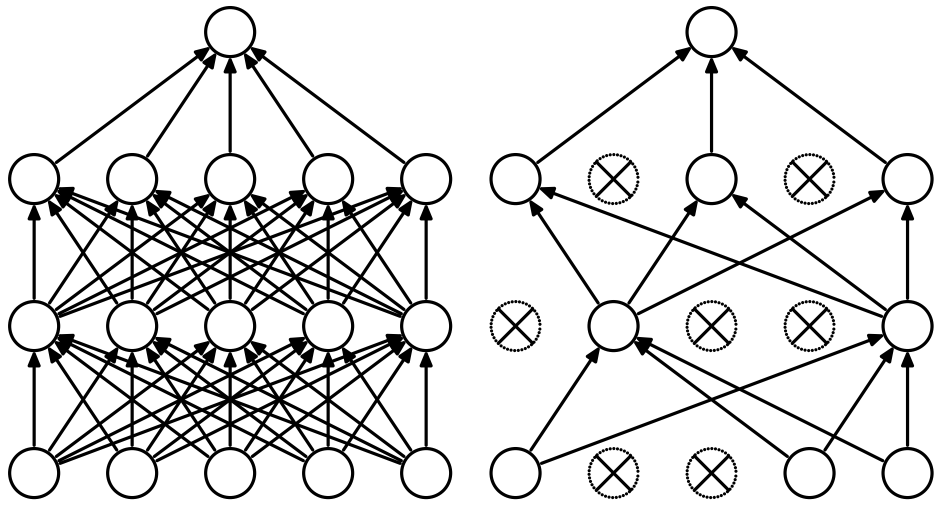
Regularization

[10,11]

- Regularization methods can prevent overtraining
  - **More data**: generally best but not always possible
  - **Early stopping**: stop training at minimum of validation loss

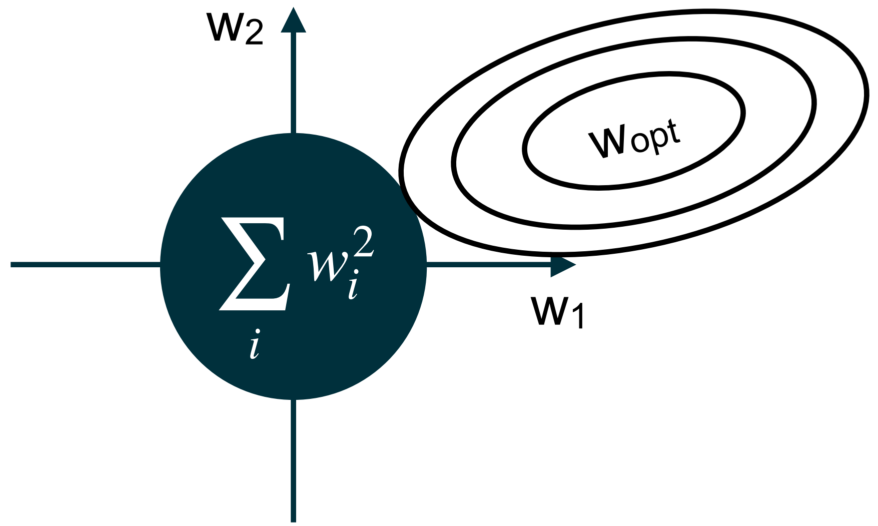
Dropout

- Randomly disable nodes during training
- Effectively creates ensemble of models



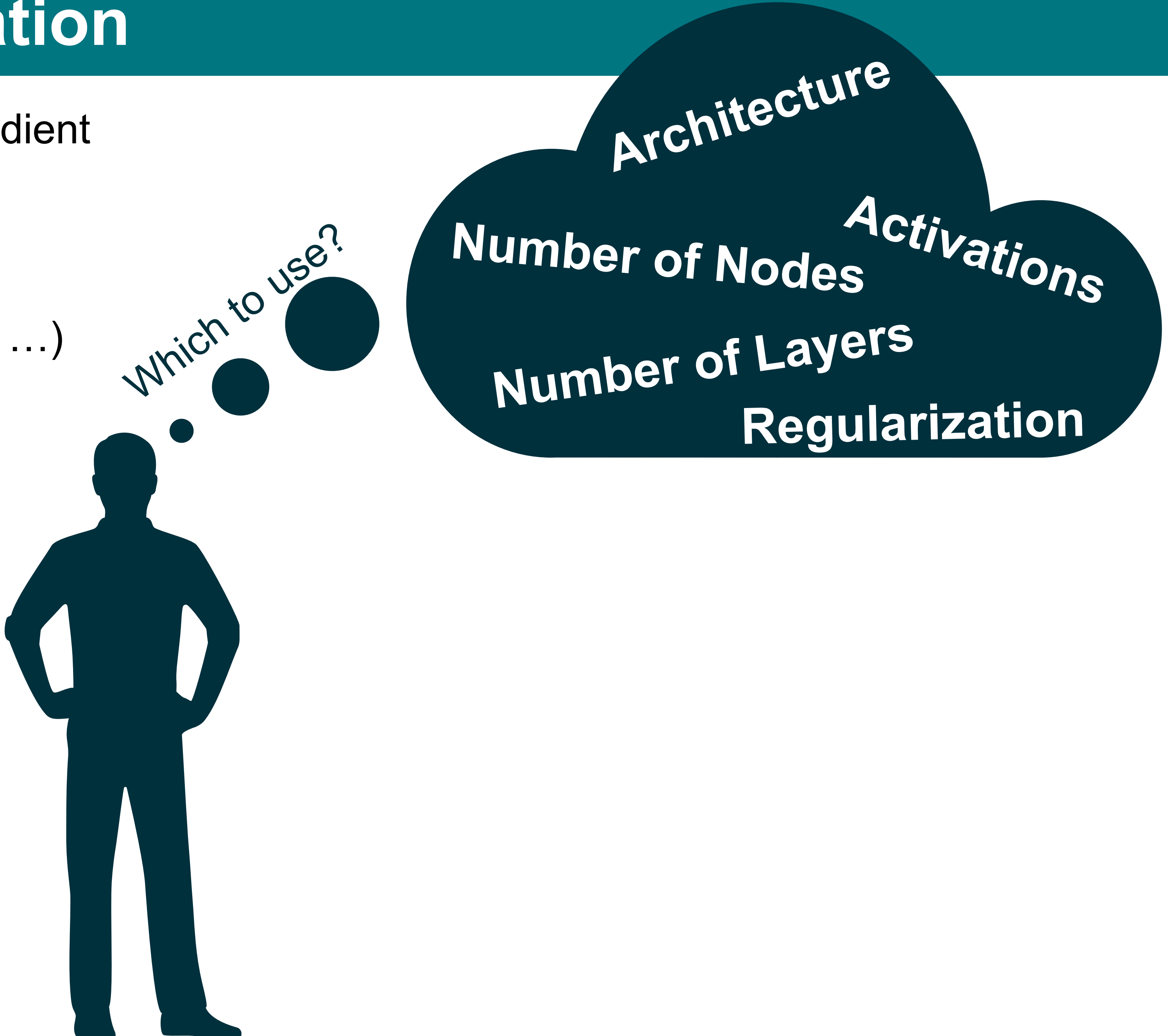
Weight Regularization

- Penalize large weight values (w_i) in $\mathscr{L}$
- Do not want few large volatile weights



Hyperparameter Optimization

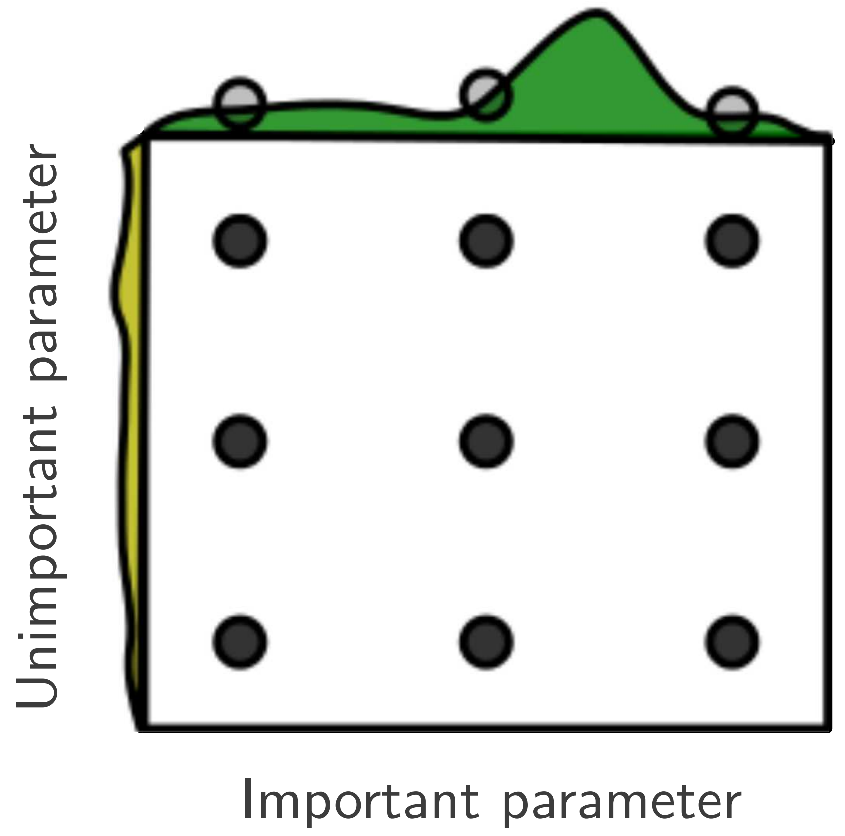
- Hyperparameters (HP) do not have gradient
- For each HP define:
  - Range (min, max, categories)
  - Domain (e.g. log for learning rates, …)
- n-HP-dimensional optimization!



Hyperparameter Optimization - Grid and Random [12]

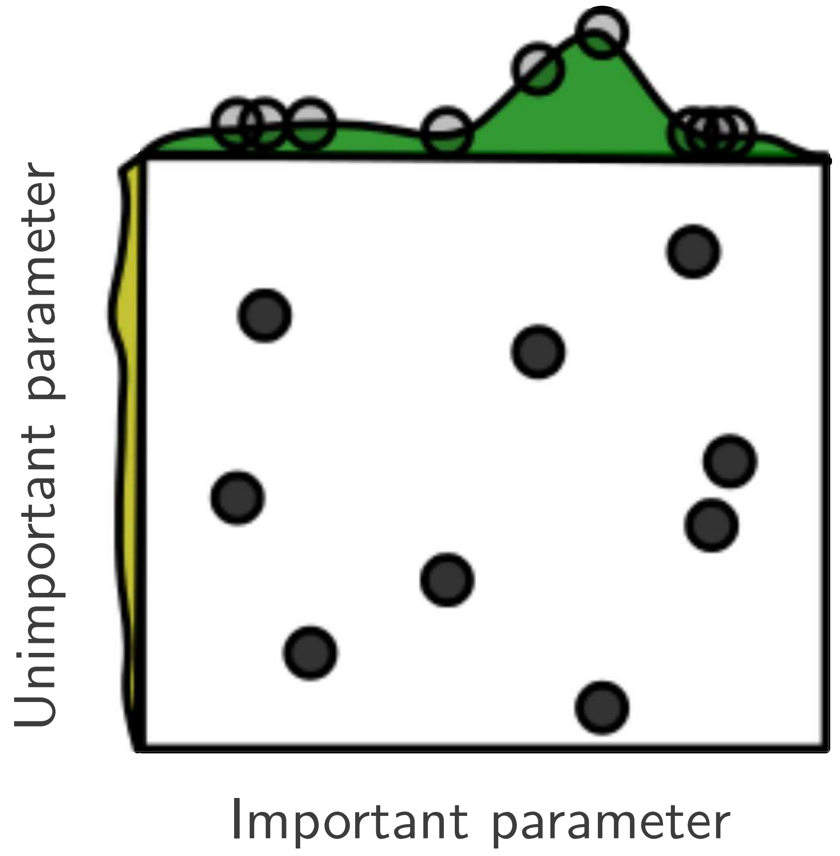
Grid Search

- Test all combinations: **exhaustive**!
- But computationally expensive/inefficient



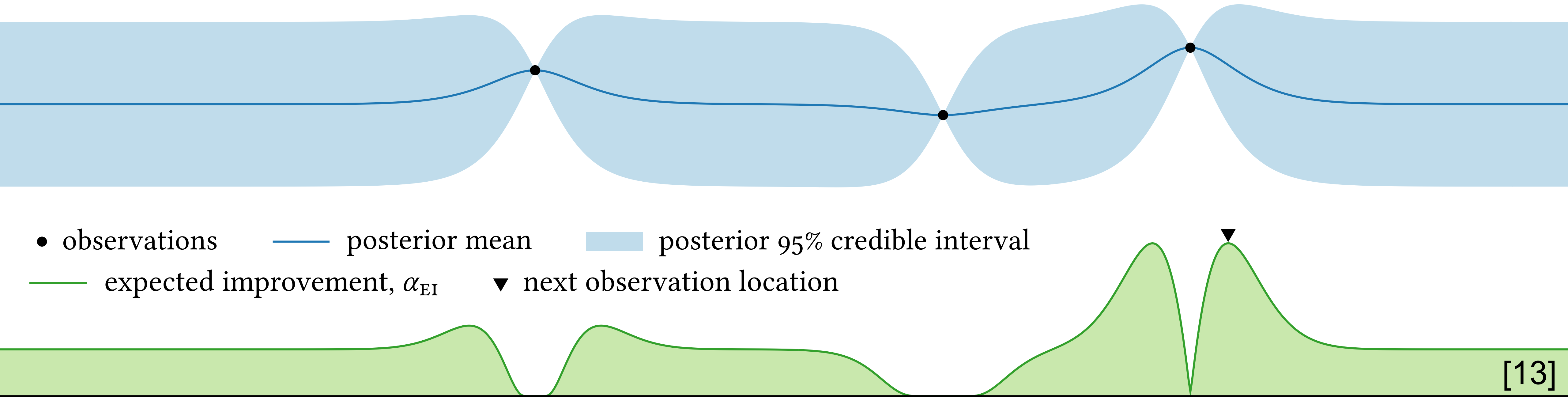
Random Search

- Test random combinations: **efficient**!
- Less systematic and non-deterministic



Hyperparameter Optimization - Bayesian

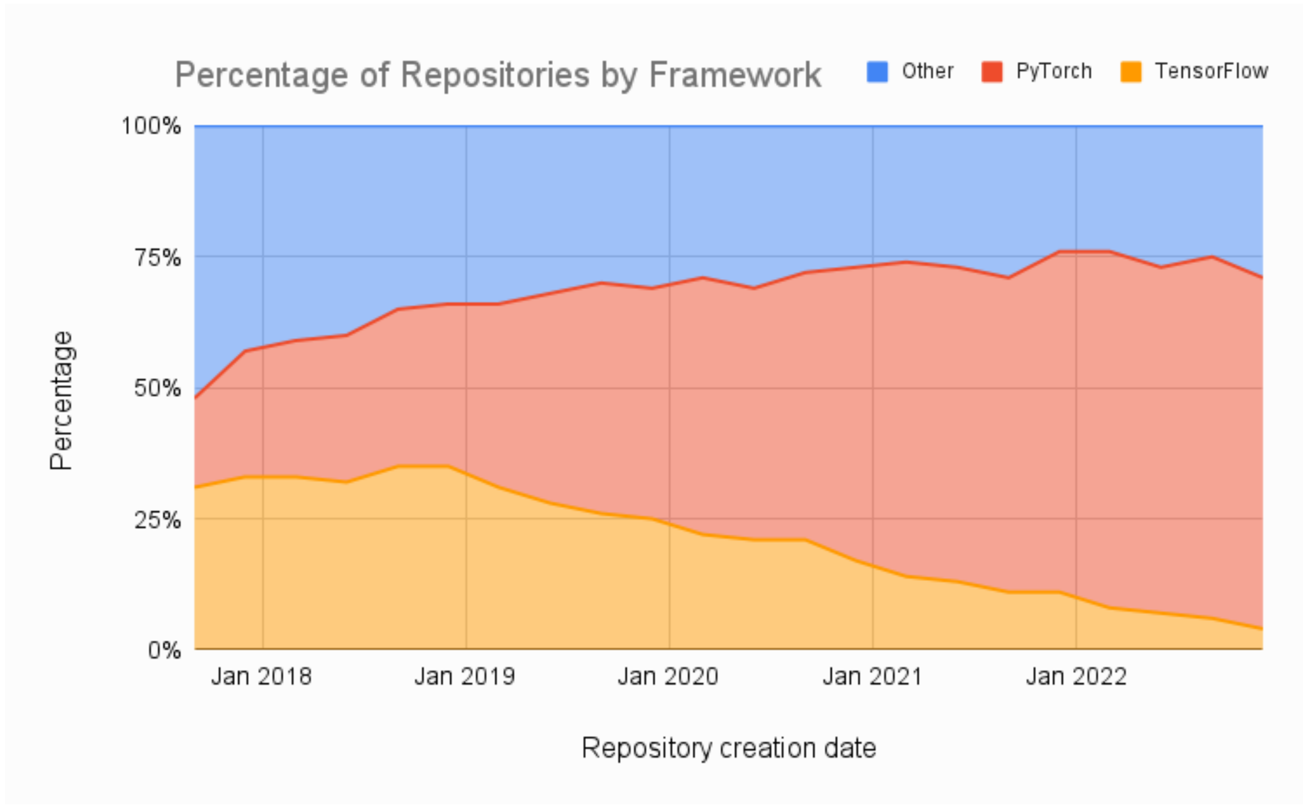
- Model hyperparameter space with **surrogate model** (e.g. Gaussian Processes)
- Use **acquisition function** to predict which hyperparameters to check next
- Can find optimal hyperparameters fast and efficiently!

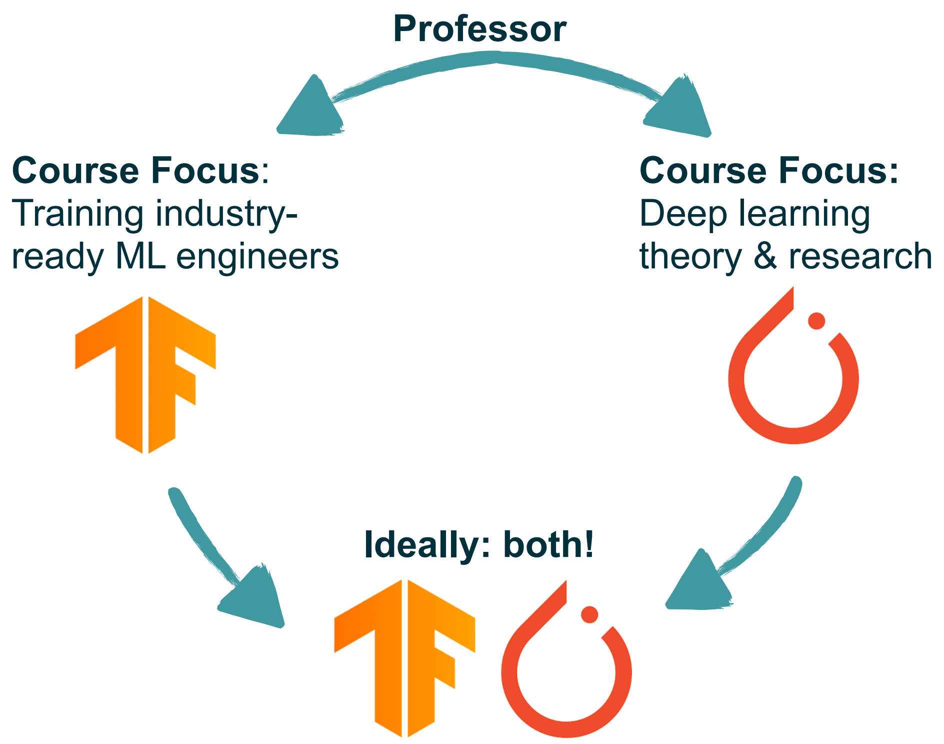


[13]

Deep Learning Software

- Two **popular, easy to use, open-source** software libraries:
  - TensorFlow: End-to-end Deep Learning, industry-ready applications
  - PyTorch: Deep Learning research, Large state-of-the-art models
- Both similar for ML-driven Physics Research

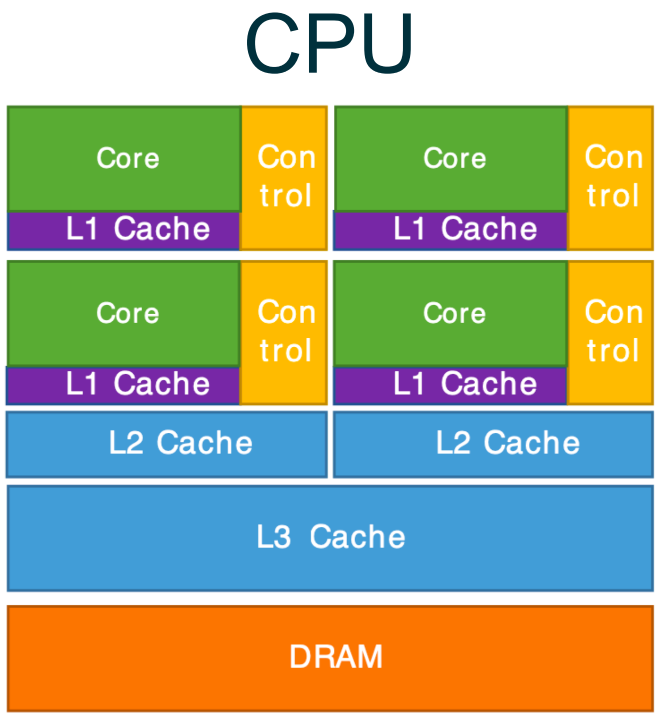


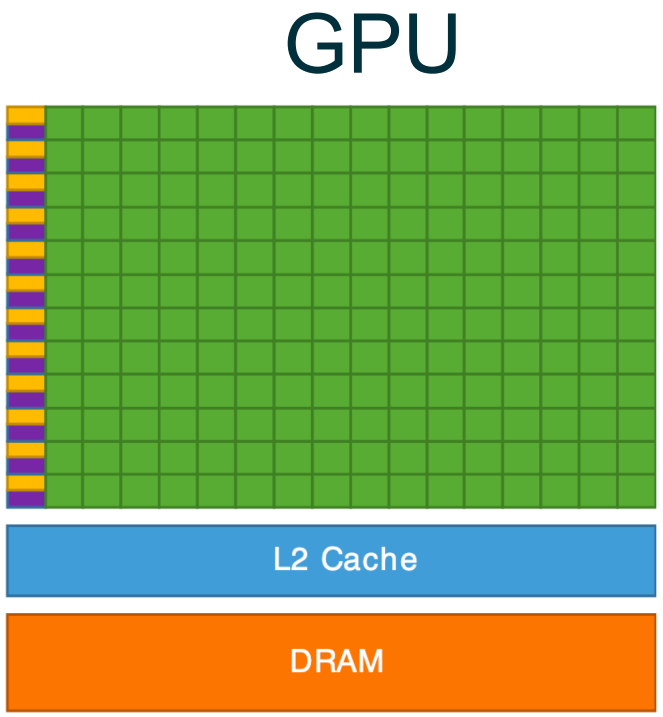

Professor
Course Focus: Training industry-ready ML engineers
Course Focus: Deep learning theory & research
Ideally: both!

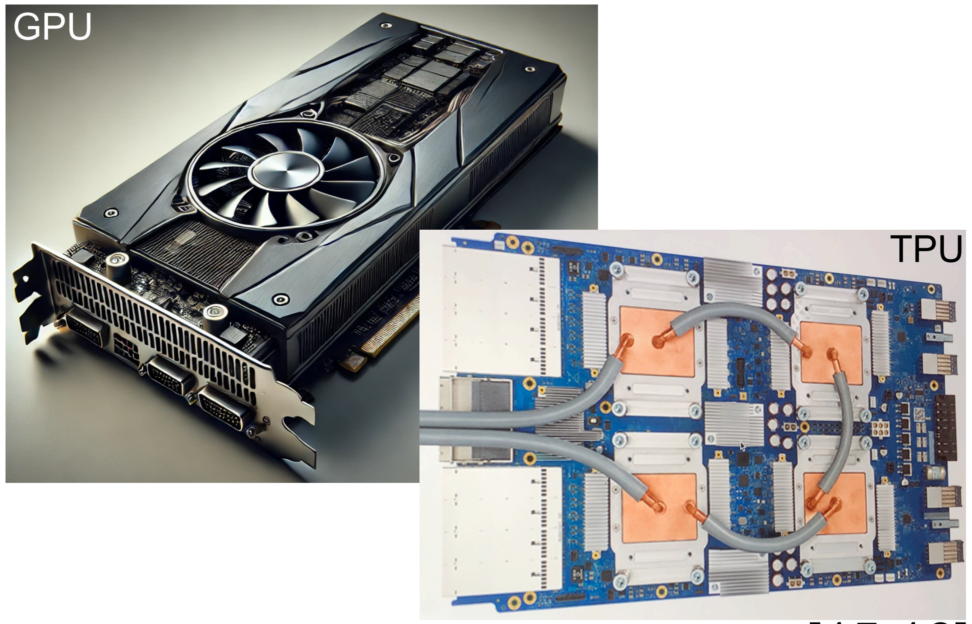
[14]

Hardware: GPUs - The Backbone of Machine Learning

- GPUs originally developed for rendering computer graphics
- GPUs enable highly parallel computations / matrix multiplications
- Other (event more advanced) architectures exist: Tensor Processing Unit (TPU)
- Many computing clusters nowadays offer enormous GPU resources (>7k GPUs on Perlmutter)







[15,16]

Introduction to Machine Learning: Part II - Take Away

- Know your data (an hour here can save you weeks!)
- Preprocess your data (fix outliers, missing values, normalize, augment)
- Split your data (train val test) before you do anything else!

- Use an appropriate architecture: many different options
- My personal start: 3 layers, 256 nodes, ReLU activation

- Use an appropriate optimizer, My personal start: SGD / Adam
- Monitor your training (loss, model predictions, GPU utilization, ...)
- Perform Hyperparameter Optimization

- Pro tips:
  - Use the right software tools (ML library, Lab book, ...)
  - Automize every step! (Data Download → Paper Document)

Citations

[1]: **Berkeley Lab History** Berkeley Lab, Link (accessed 11.08.24)

[2]: **Deep Learning in Physics Research** Martin Erdmann et al., Lecture (RWTH Aachen University). Apr. 2022. link (accessed 06.01.23)

[3]: **Deep Residual Learning for Image Recognition** Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun, 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), held 27-30 June 2016 in Las Vegas, NV. ISSN: 1063-6919, id. 1, eprint arxiv:1512.03385

[4]: **Densely Connected Convolutional Networks** Gao Huang, Zhuang Liu, Laurens van der Maaten, Kilian Q. Weinberger, eprint arXiv:1608.06993

[5]: **Visualizing and Understanding Convolutional Networks** Matthew D Zeiler, Rob Fergus, eprint arXiv:1311.2901

[6]: **Graph Neural Networks for the Travelling Salesman Problem**, Chaitanya K. Joshi et al, INFORMS Annual Meeting, October 22, 2019 (inspired figure)

[7]: **Visualizing the Loss Landscape of Neural Nets** Hao Li, Zheng Xu, Gavin Taylor, Christoph Studer, Tom Goldstein, Advances in Neural Information Processing Systems 31 (NeurIPS), 2018, Link (accessed 11.08.24)

[8]: **Mastering Model Building** Marcel Rieger, Lecture, Deep Learning School "Basic Concepts" ERUM Data Hub, 08/22, Link (accessed 11.08.24)

[9]: **Optimizer Visualization** Jae j-w-yun, GitHub Repository, link (accessed 11.08.24)

[10]: **Dropout: A Simple Way to Prevent Neural Networks from Overfitting** Geoffrey Hinton et al., Journal of Machine Learning Research 15 (2014) 1929-1958

[11]: **The effect of L2-regularization** Julien Harbulot, Personal Website link (accessed 11.08.24, inspired figure)

[12]: **Random Search for Hyper-Parameter Optimization** James Bergstra, Yoshua Bengio, Journal of Machine Learning Research 13 (2012) 281-305, link (accessed 11.08.24)

[13]: **Bayesian Optimization** Roman Garnett, Cambridge University Press, 2023 link (accessed 11.08.24)

[14]: **PyTorch vs TensorFlow in 2023** Ryan O'Connor, Assembly AI Blog, link (accessed 11.08.24)

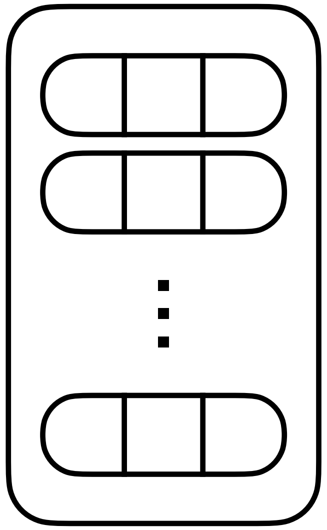
[15]: **Design: GPU vs. CPU** Cornell Virtual Workshop link (accessed 11.08.24)

[16]: **Tensor Processing Unit 3.0 (Personal Picture)** Zinskauf, CC BY-SA 4.0, via Wikimedia Commons

Backup

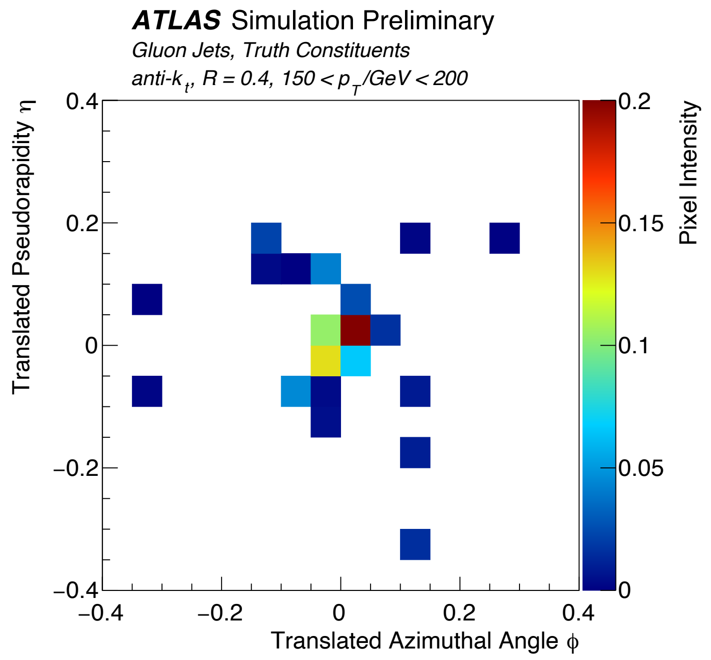
How to represent the data?

As Lists?



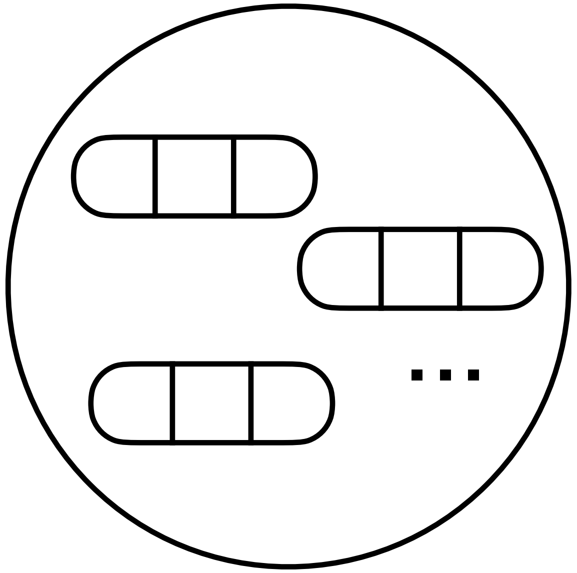
Pro: Easy to use
Con: Which ordering?

As Images?



Pro: Geometric information
Con: Very sparse

As Sets?



Pro: Works with sparse data
Con: No geometric information

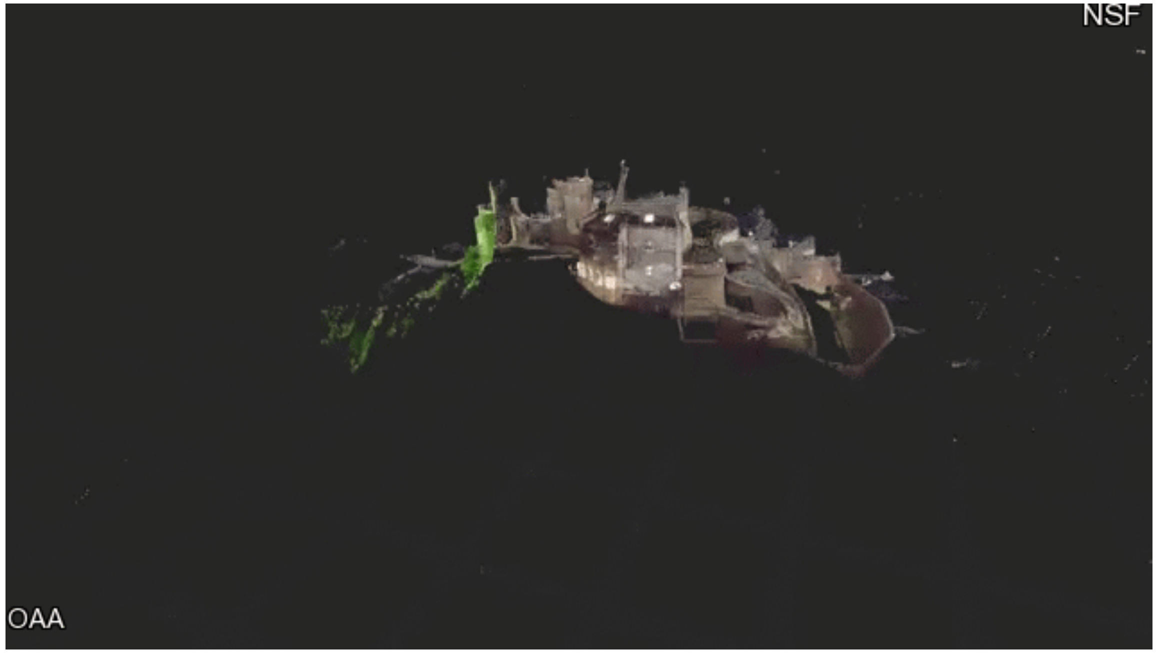
As Point Clouds…

- Unordered set of objects in metric space
- **Why is this nice? Objects can be our detector hits!**

Works with sparse data

Respects permutation invariance

Each object can have position, time, …



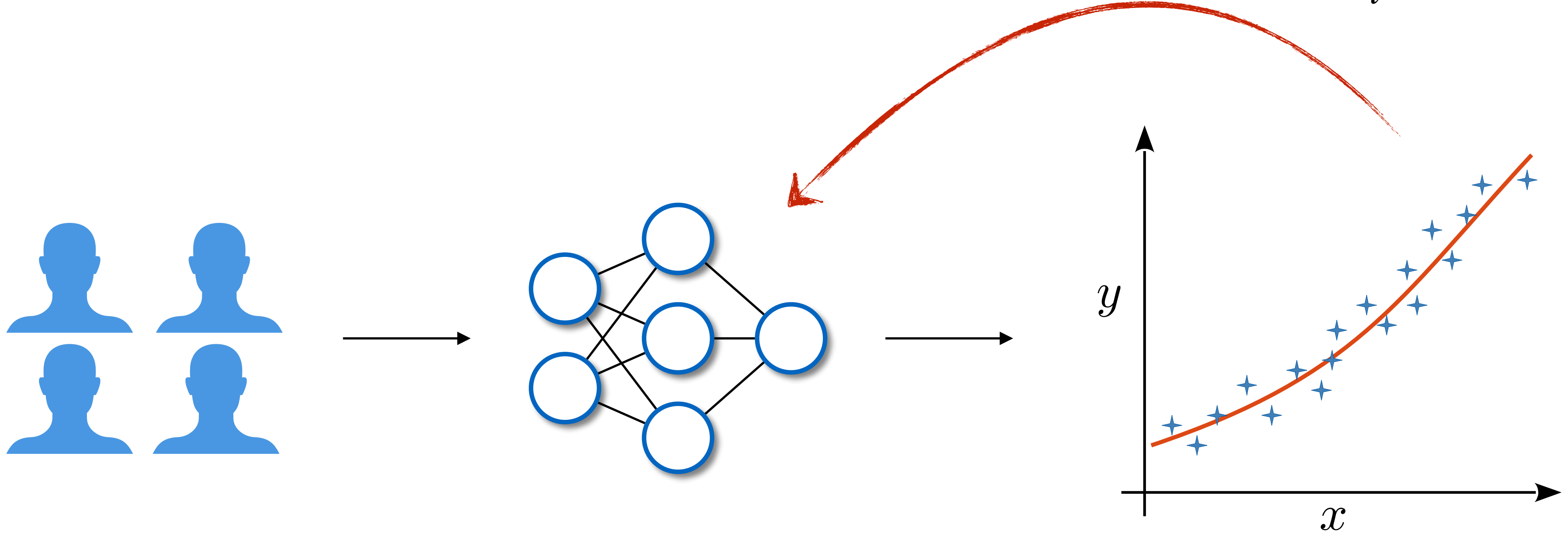
[8, 9]

Regression: Predict continuous feature

- Predict a real number associated with a feature vector
- Example:
  - Prediction: What is the future net income of a student?
  - Input: Grade in course, Participation, Year of study
- Last activation: Linear (no activation)

Mean squared error (MSE) loss:

$$\mathcal{L} = \frac{1}{n}\sum_{i}^{n}(y_i^{true} - y_i^{pred})^2$$

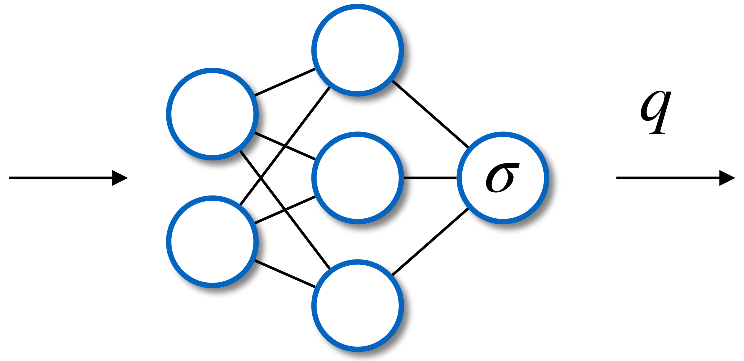


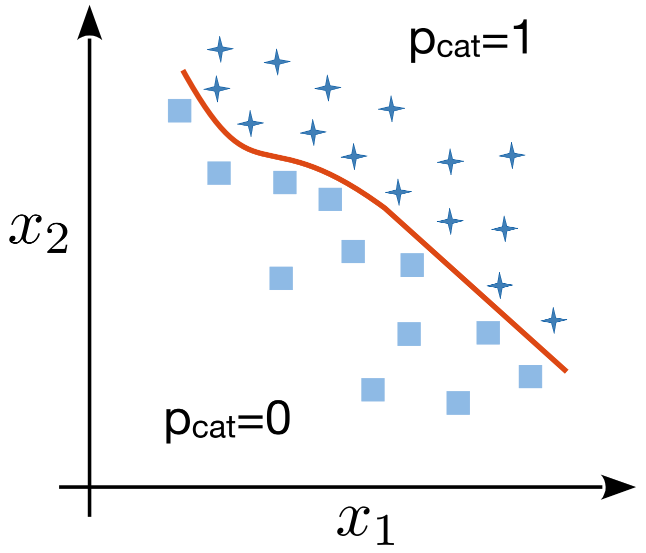
Classification: Predict discrete classes

- Predict a discrete value (label) associated with a feature vector
- Example:
  - Prediction: Does this picture show a cat or a dog?
  - Input: Pixels of image
- Last activation: Sigmoid/softmax
  - Predicted probability $0\,\% \leq q \leq 100\,\%$

Cross-Entropy for c classes:

$$\mathscr{L} = -\frac{1}{n}\sum_{i}^{n}\sum_{j}^{c} p_{ij}\cdot\log(q_{ij})$$





Back-Propagation (Example)

- Each network is a series of (simple) mathematical operations
- Each operation has:
  - Local output (forward pass)
  - Local derivative (backward pass)
- Use chain rule to evaluate derivatives for every parameter

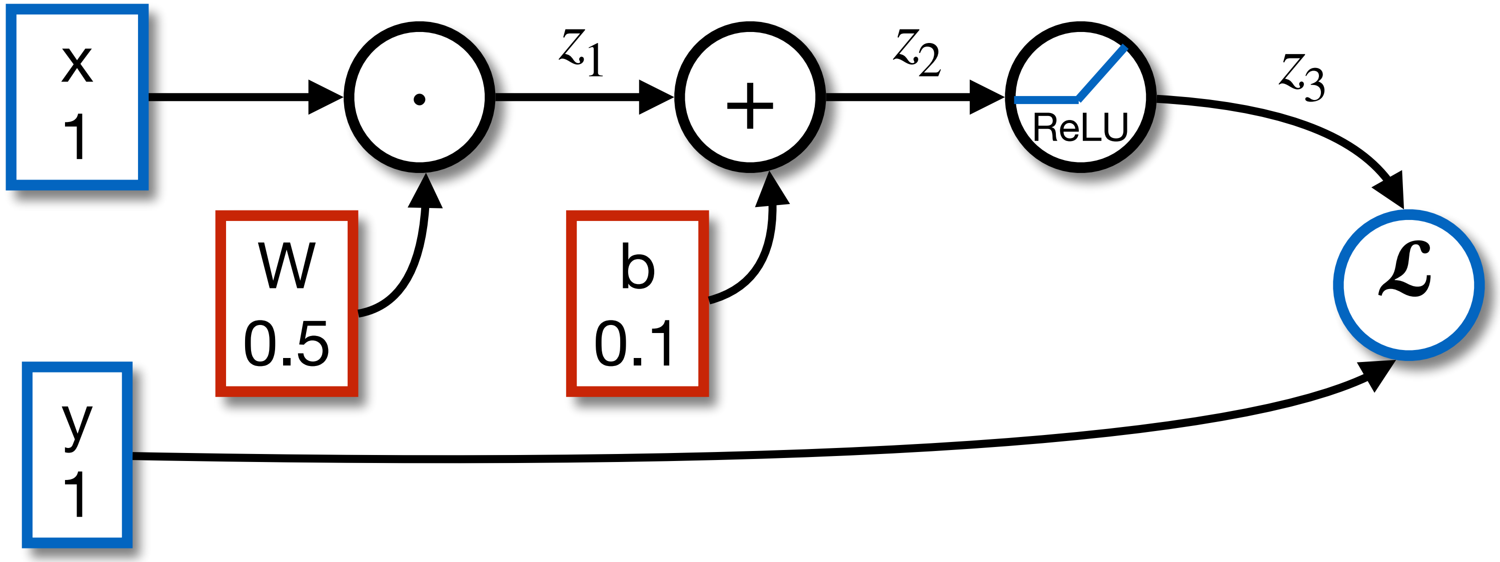
Example: $y^{pred} = z_3 = \sigma(Wx + b)$

$$\begin{tikzcd}
x \arrow[r] & \cdot \arrow[r, "z_1"] & + \arrow[r, "z_2"] & \sigma \arrow[dr, bend left, "z_3"] & \\
& W \arrow[u, bend right] & b \arrow[u, bend right] & & \mathcal{L} \\
y \arrow[urrrr, bend right] & & & &
\end{tikzcd}$$

$$\partial \mathscr{L} / \partial W = \partial \mathscr{L} / \partial z_3 \cdot \partial z_3 / \partial z_2 \cdot \partial z_2 / \partial z_1 \cdot \partial z_1 / \partial W$$

Back-Propagation (Example)

$$\partial \mathcal{L} / \partial W = \partial \mathcal{L} / \partial z_3 \cdot \partial z_3 / \partial z_2 \cdot \partial z_2 / \partial z_1 \cdot \partial z_1 / \partial W$$



Forward pass

$z_1 = Wx = 0.5$

$z_2 = z_1 + b = 0.6$

$z_3 = \sigma(z_2) = \text{ReLU}(z_2) = 0.6$

$\mathcal{L}(z_3) = (z_3 - y)^2 = 0.16$

Backward pass

$\partial \mathcal{L} / \partial z_3 = 2(z_3 - y) = -0.8$

$\partial z_3 / \partial z_2 = \partial \sigma(z_2) / \partial z_2 = 1$

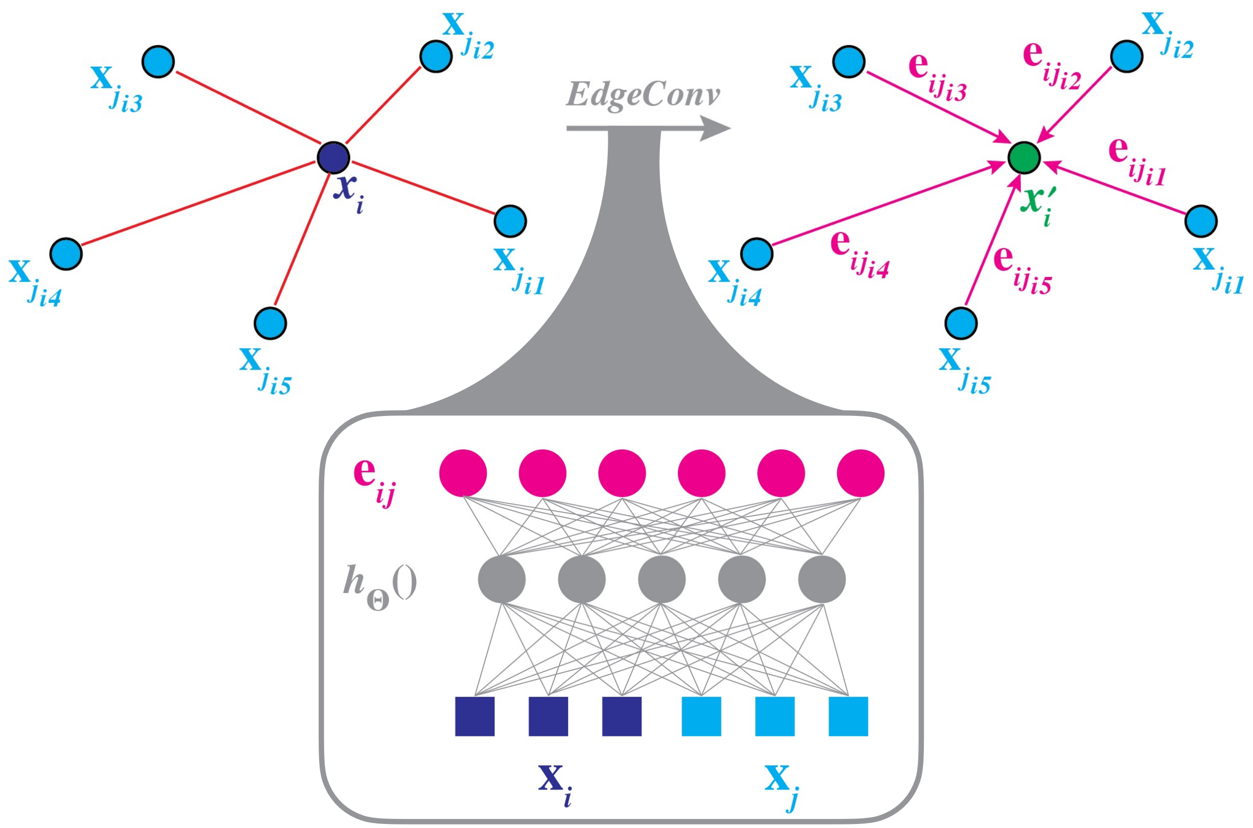
$\partial z_2 / \partial z_1 = 1$

$\partial z_1 / \partial W = x = 1$

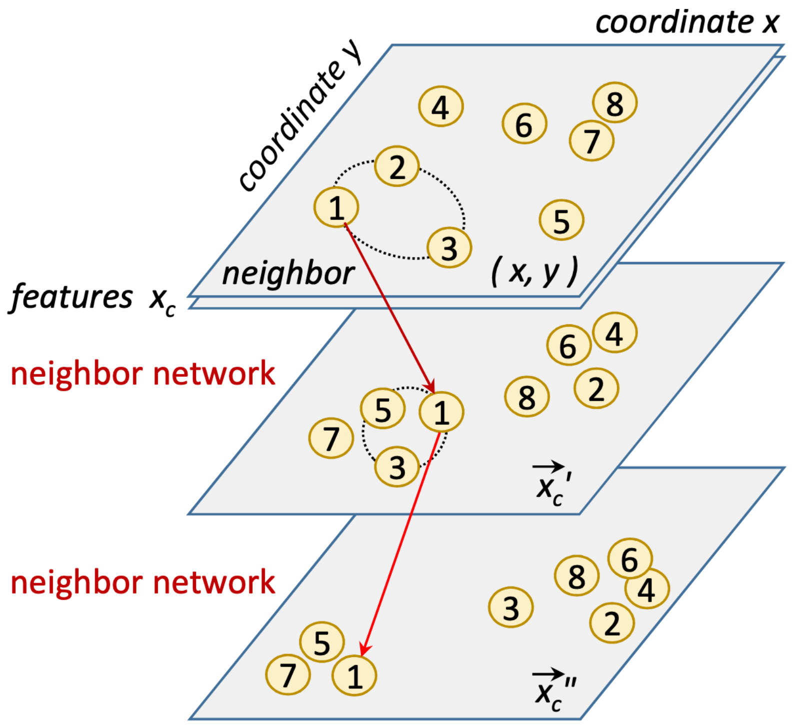
$\Rightarrow \partial \mathcal{L} / \partial W = -0.4 \cdot 1 \cdot 1 \cdot 1 = -0.4$

Graph Neural Network: Edge Conv

- Use for graph-like (unordered) data:
  - Nodes (e.g. people in social network)
  - Edges (e.g. relations between people)
- One possible architecture: EdgeConv



- Steps:
  1. Construct local neighborhood graph
  2. Extract edge features (with DNN)
  3. Symmetric aggregation (sum or max)
  4. Rebuild graph in feature space

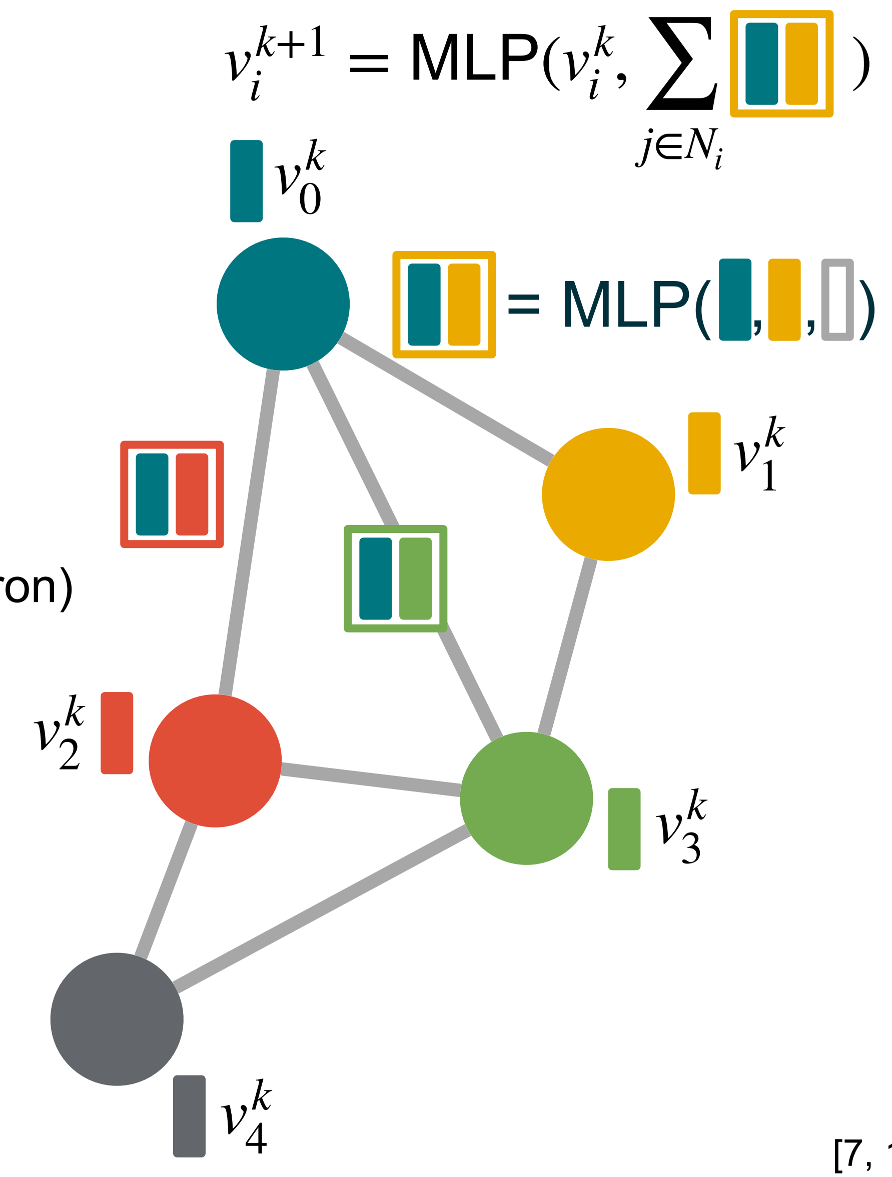


[2, 3]

Learning on Graphs (Interaction Network)

- **What is a graph:**
  - Nodes: Have features 🟦🟨🟥🟩
  - Edges: Connect nodes, can have features —

- **Learning by updating each node (i):**
  - Embed edges $e_{ij}^{k+1} = \text{MLP}(v_i^k, v_j^k, e_{ij}^k)$ (Multilayer Perceptron)
  - Aggregate embebbings $E_i^{k+1} = \sum_{j \in N_i} e_{ij}^{k+1}$
  - Embed aggregations $v_i^{k+1} = \text{MLP}(v_i^k, E_i^{k+1})$



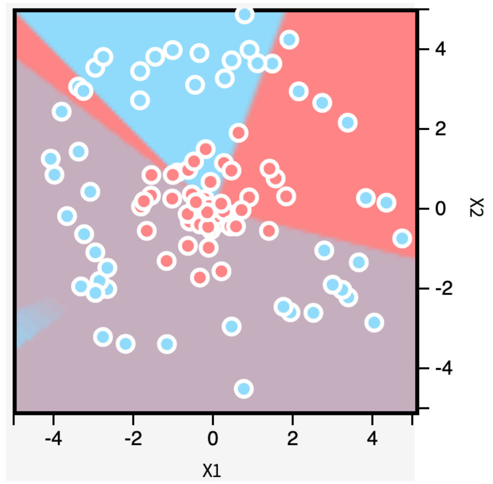
[7, 10]

Parameter initialization

- Initialization of model parameters can be critical for performance
- Choose Gaussian distributed initial weights / break symmetry
- Two standard initializations:
  - Sigmoid, Tanh: $\sigma^2 = 2/(n_{in} + n_{out})$
  - ReLU: $\sigma^2 = 2/n_{in}$

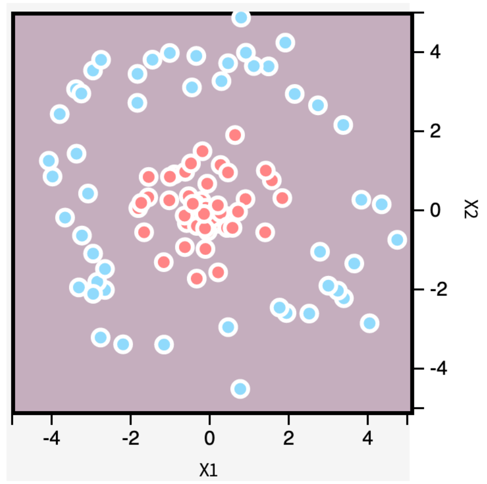
Weights to large

- Exploding Signals:



Weights to small

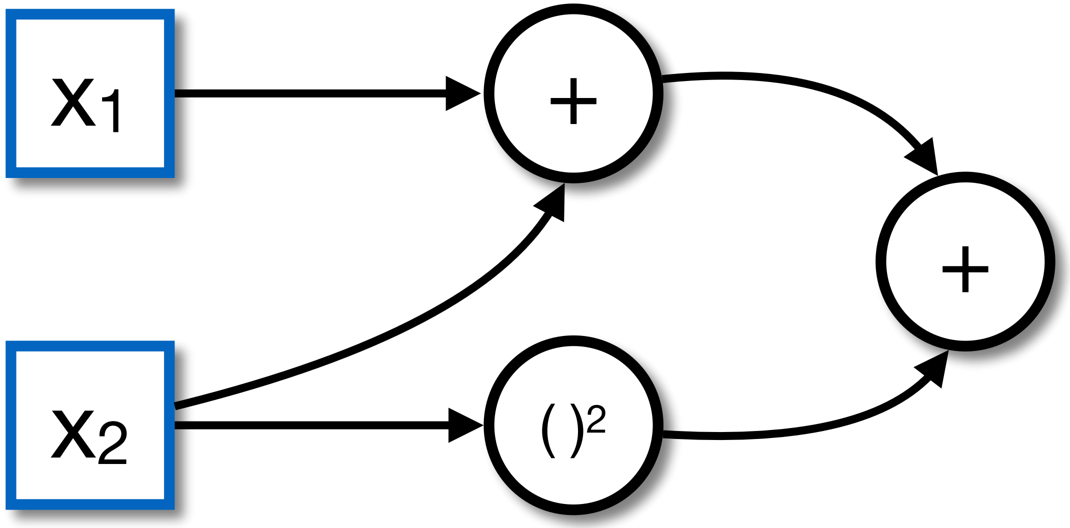
- Vanishing Signals:



Graphs

- Graph = static computing model consisting of
  - Tensors (value placeholders)
  - Structural elements which connect tensors (e.g. tf.Operation)
- Defined by: Inputs, Outputs, Operations and connections

$$f(x_1, x_2) = x_1+x_2+x_2^2$$

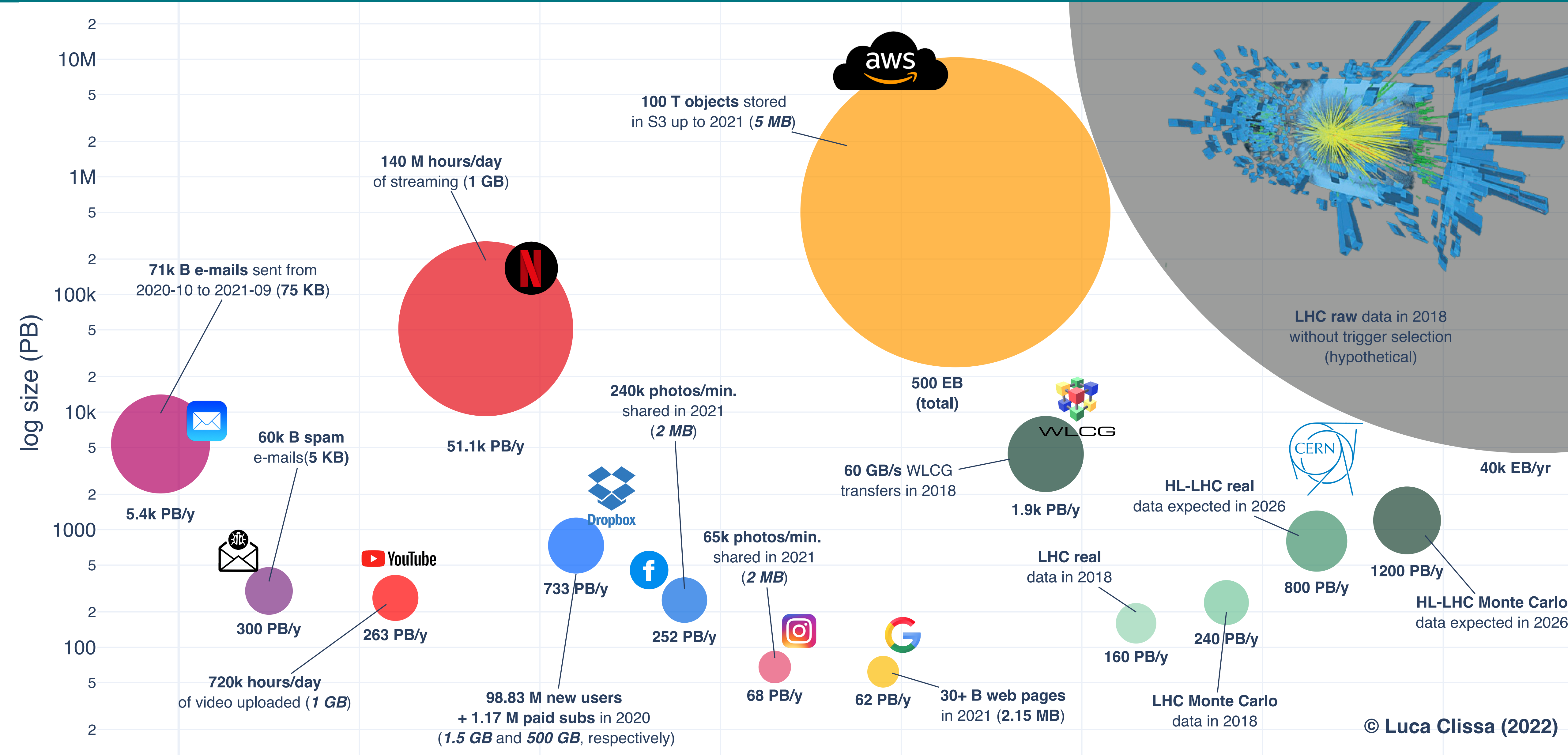


- Graphs can be **optimized** (parallel execution): Super fast!
- Graphs are **portable**: Run on CPU, GPU, TPU, Multiple devices in parallel
- Graphs are **static**: Everybody gets the same results, everywhere

AdaGrad

- Adaptive Learning Rate for every Parameter (i)
  - Smaller updates for parameters associated with frequent modifications
  - Larger updates for parameters associated with infrequent modifications -> Tries a lot in unknown directions!
- How: G is sum of squares of gradients of loss with respect to theta i
- Pro: Learning rate does not have to be tuned of set specifically
- Con:
  - G is monotonically increasing over number of epochs
  - Therefore learning rate decay to zero

How Big is BIG DATA?


log size (PB)
10M
1M
100k
10k
71k B e-mails sent from 2020-10 to 2021-09 (75 KB)
5.4k PB/y
60k B spam e-mails(5 KB)
300 PB/y
720k hours/day of video uploaded (1 GB)
YouTube
263 PB/y
140 M hours/day of streaming (1 GB)
51.1k PB/y
Dropbox
733 PB/y
98.83 M new users + 1.17 M paid subs in 2020 (1.5 GB and 500 GB, respectively)
240k photos/min. shared in 2021 (2 MB)
252 PB/y
65k photos/min. shared in 2021 (2 MB)
68 PB/y
100 T objects stored in S3 up to 2021 (5 MB)
aws
500 EB (total)
62 PB/y
30+ B web pages in 2021 (2.15 MB)
60 GB/s WLCG transfers in 2018
WLCG
1.9k PB/y
LHC real data in 2018
160 PB/y
240 PB/y
LHC Monte Carlo data in 2018
HL-LHC real data expected in 2026
800 PB/y
CERN
1200 PB/y
HL-LHC Monte Carlo data expected in 2026
LHC raw data in 2018 without trigger selection (hypothetical)
40k EB/yr
© Luca Clissa (2022)