

Introduction to ML: Part II

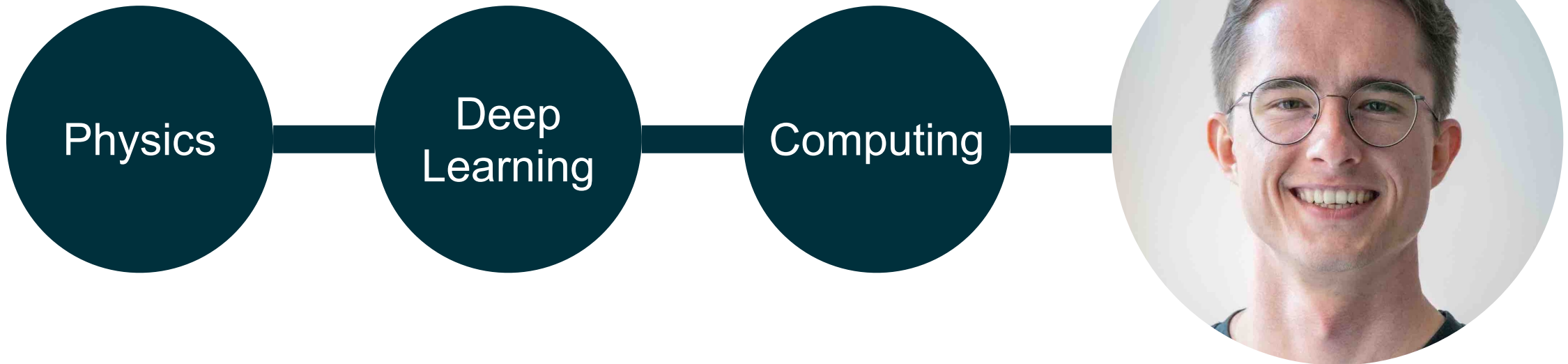
Dennis Noll

ML4FP School

June 1st, 2026

About Me

- 3rd year postdoc @ Stanford University
- Since >9 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

\geq 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

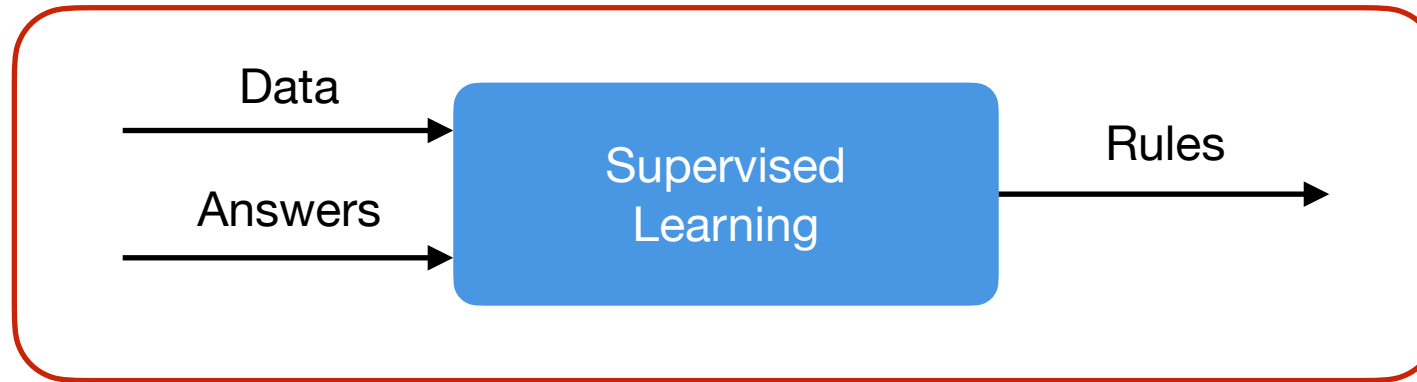
C Use occasionally

D Use regularly

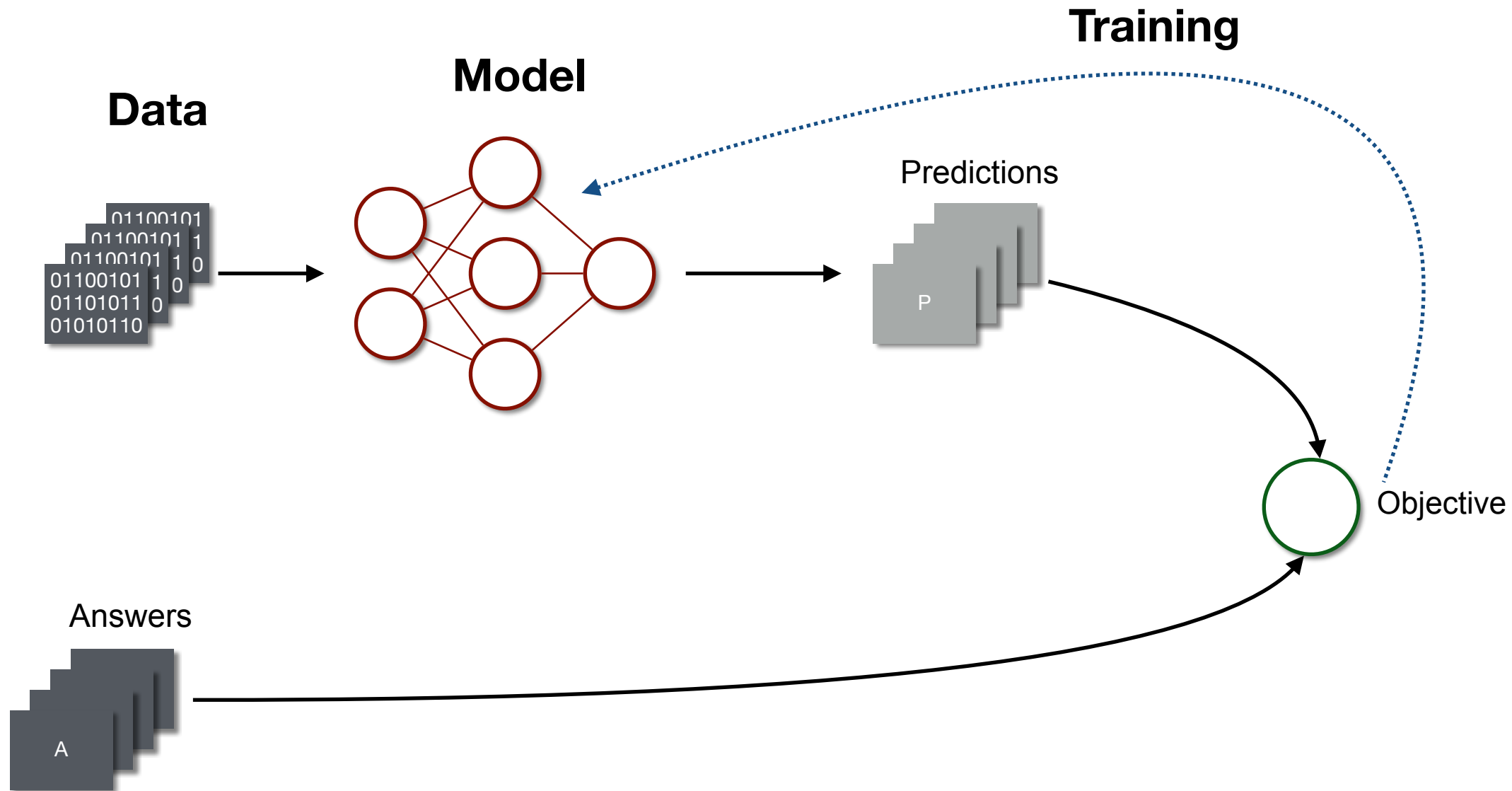
Please Ask!

If you don't understand it there is another person in the room that also doesn't understand it. Do them a favor!

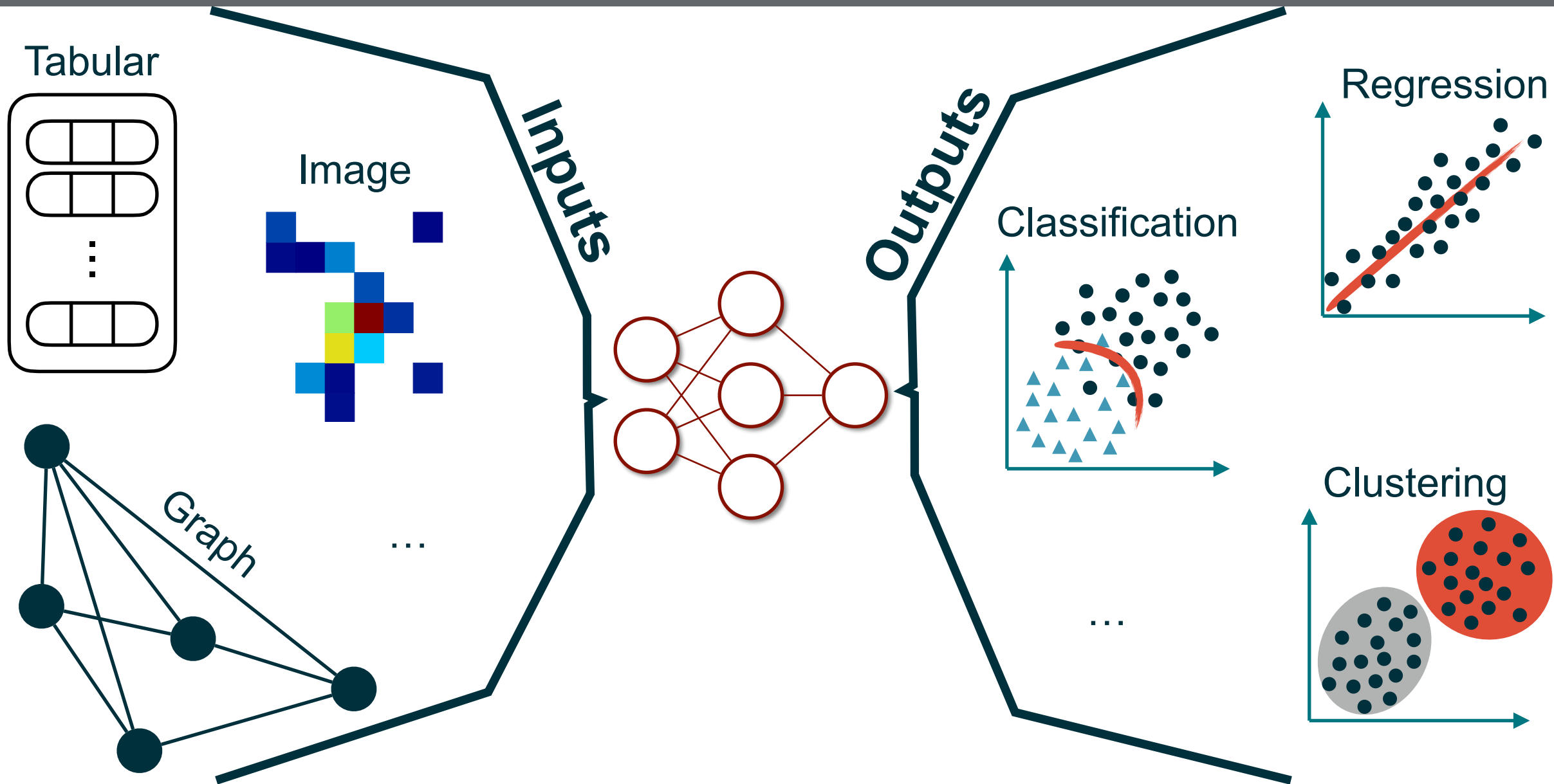
Types of Machine Learning



Supervised Machine Learning in a Nutshell



Data (Recap)

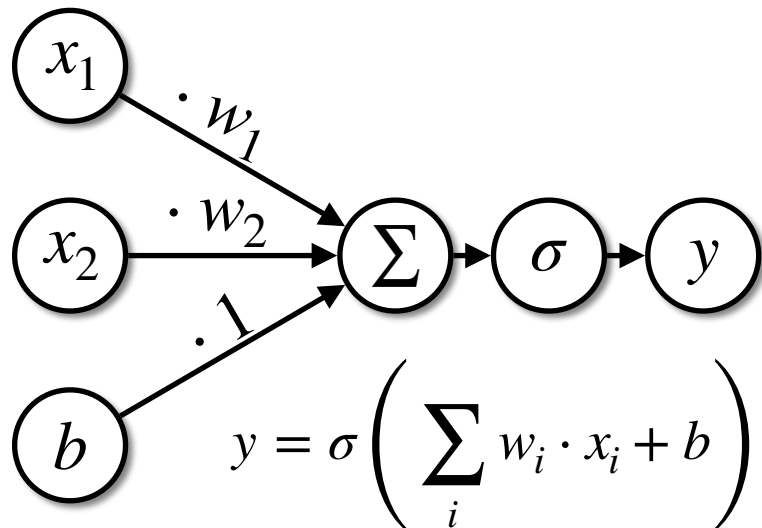


Model (Recap)

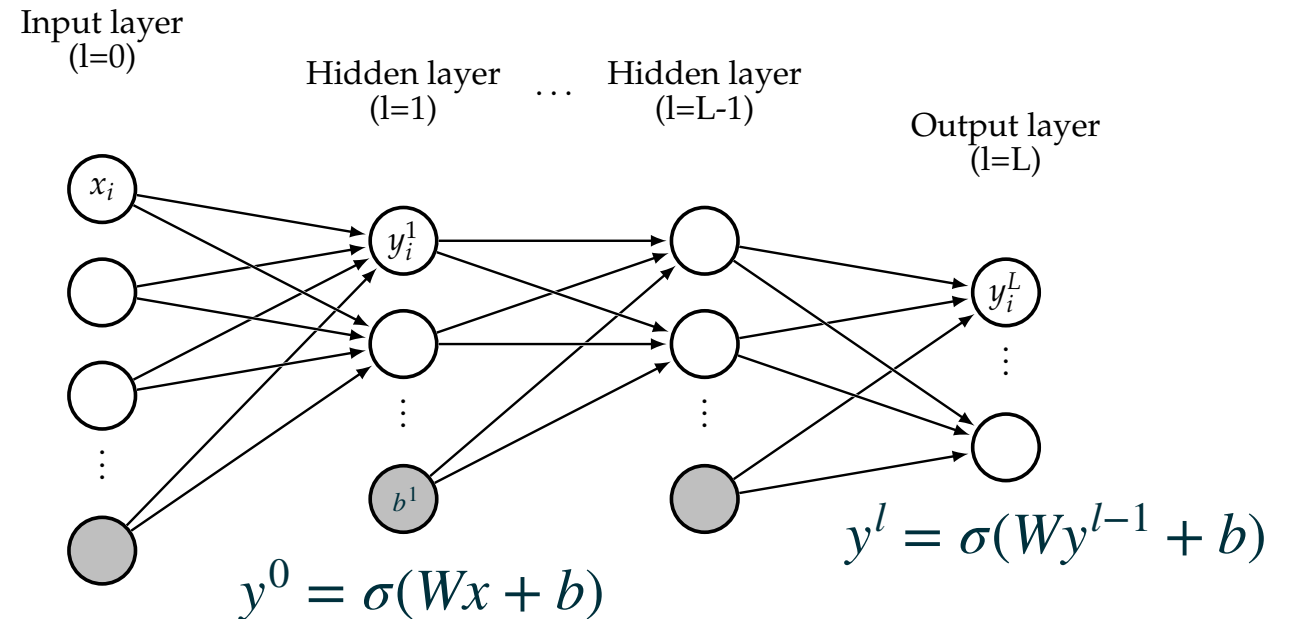
- **Node (Perceptron):** Granular unit with parameters $\theta = (W, b)$
- **Neural Network:**
 - Connected layers of multiple nodes
 - **Width:** number of nodes per layer
 - **Depth:** number of layers holding weights

can approximate many functions!

Node



Neural Network

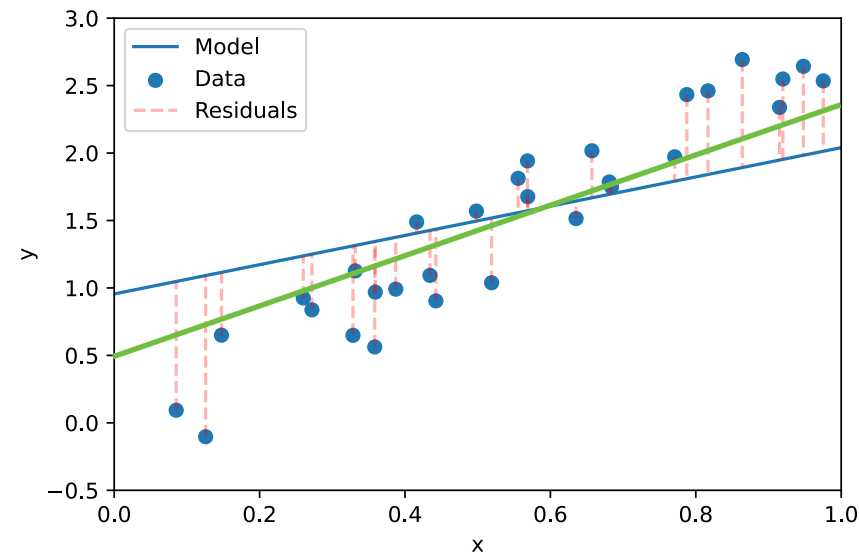


Learning (Recap)

Objective Function

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

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



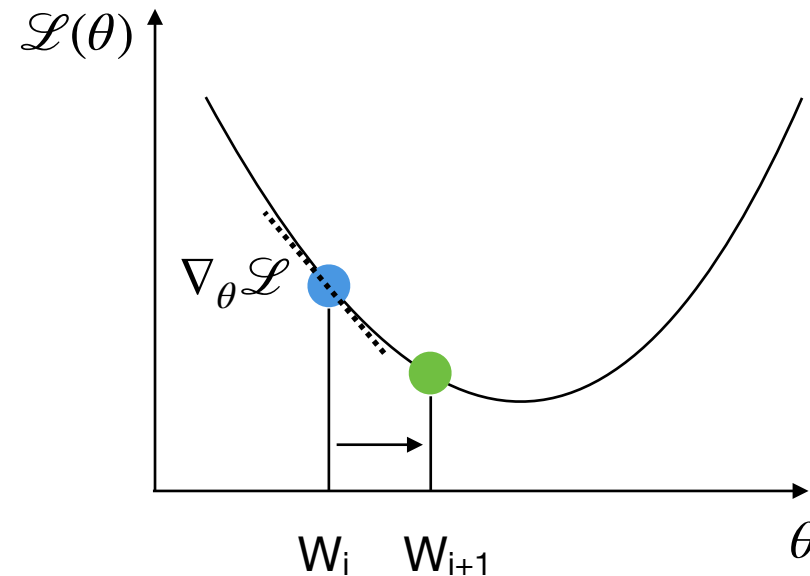
Parameter Update

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

$$\theta \rightarrow \theta - \alpha \nabla_{\theta} \mathcal{L}$$

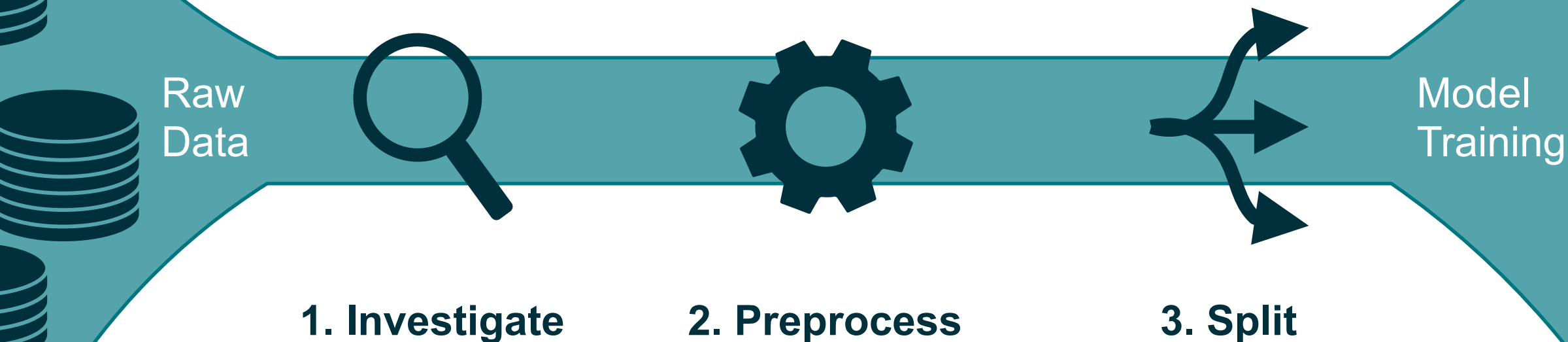
Step size
(learning rate)

Gradient



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!



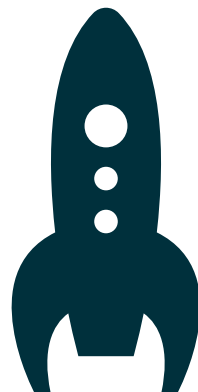
Understand

- 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

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

- **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

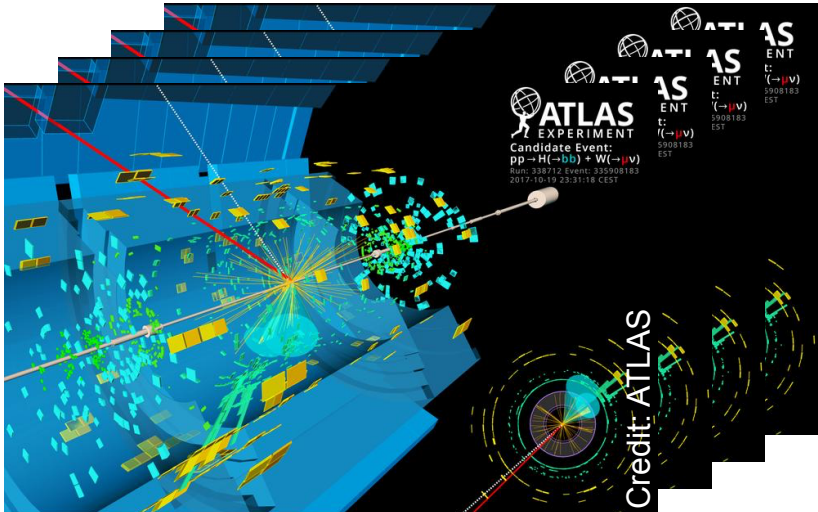


Data Preprocessing: Missing Values

- Recorded data often has missing values (e.g. limited amount of jets in HEP event)
- Strategies:
 - Delete Rows / Columns:** Easy, but can lose 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 jagged (different length) arrays:** Requires advanced techniques

Use what works!

Detected Events

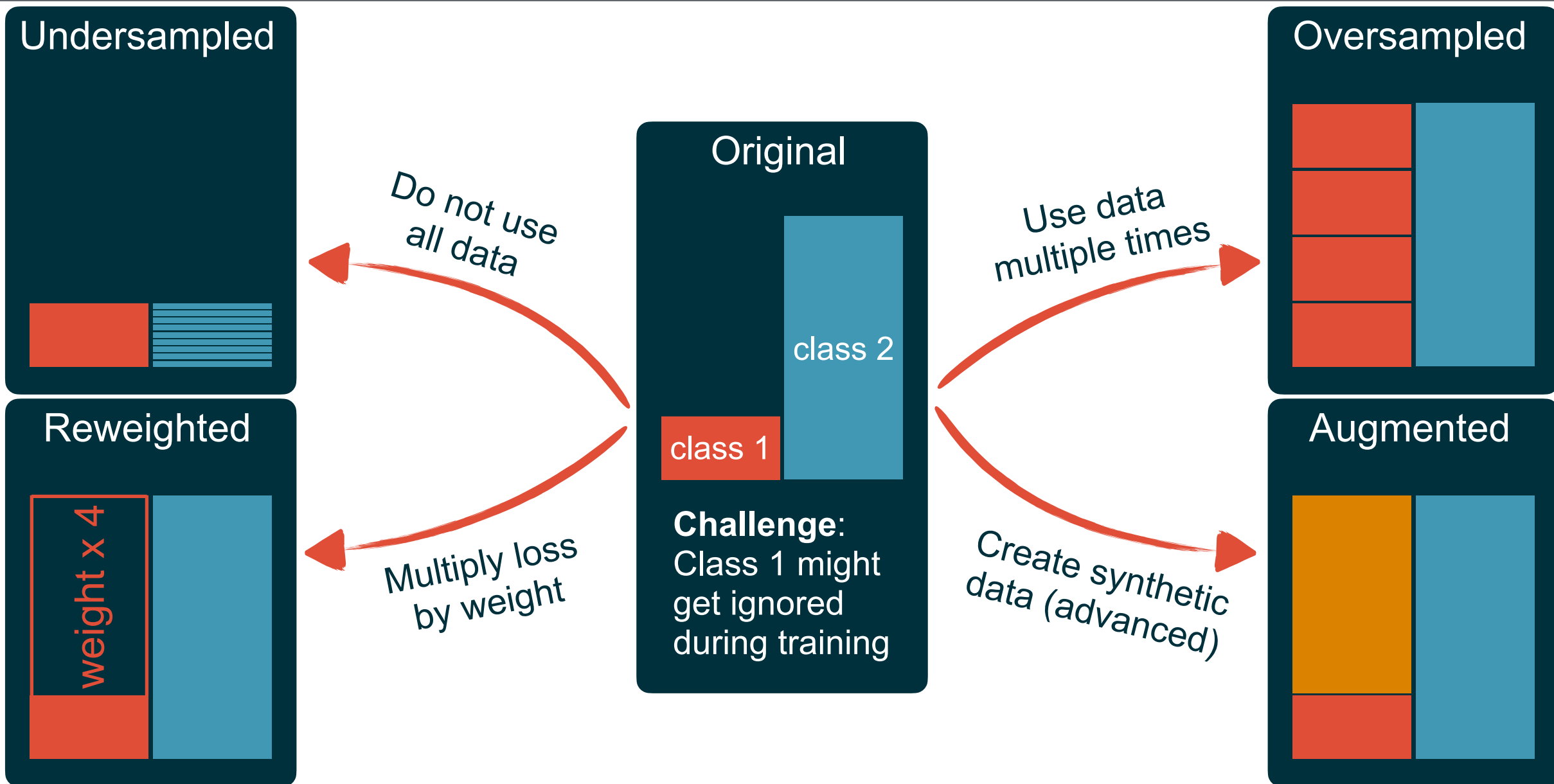


Training Data

	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	-	-



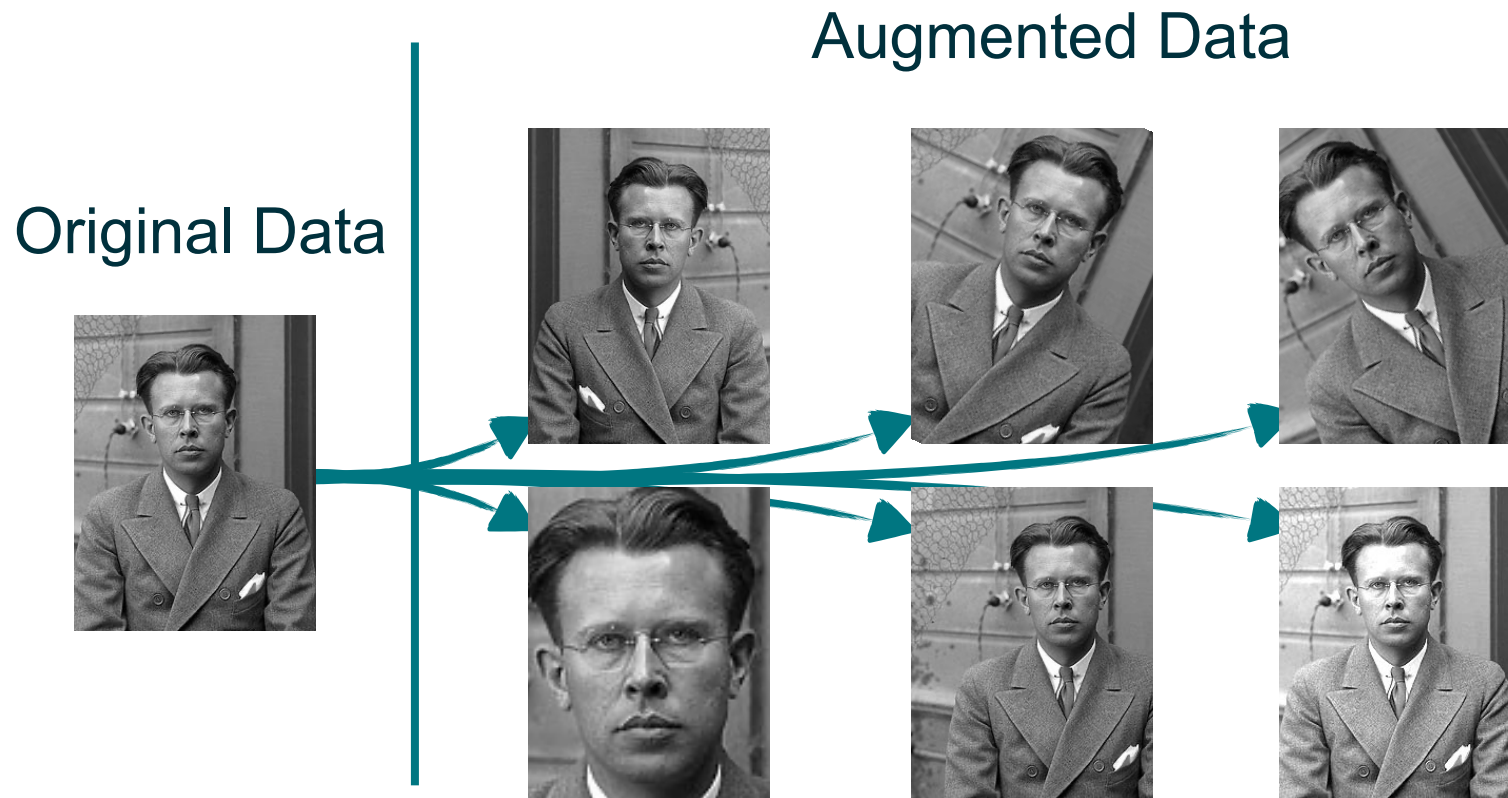
Data Preprocessing: Class Imbalance





Data Preprocessing: Augmentation

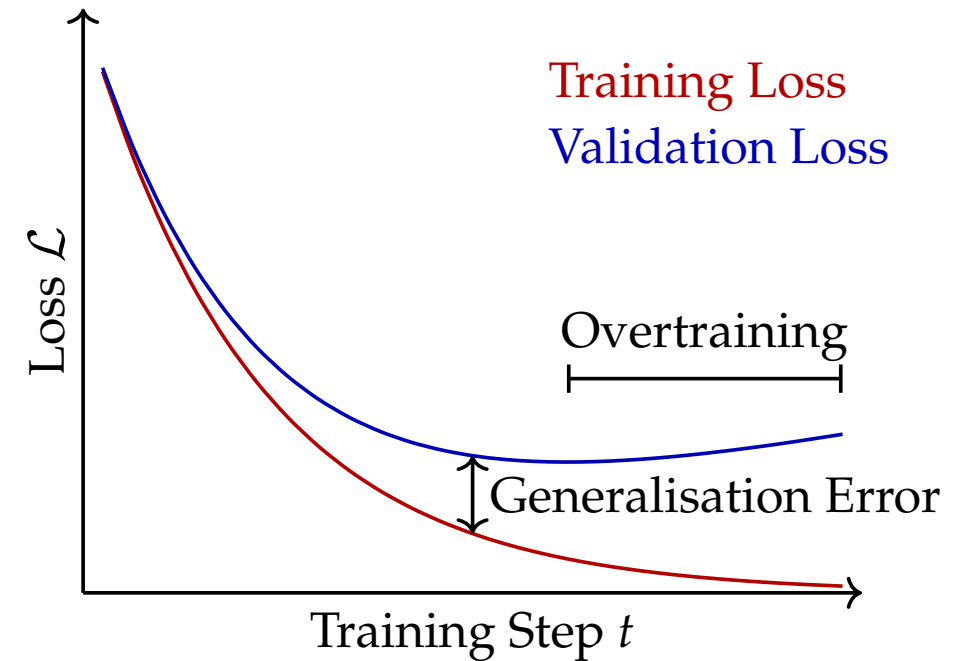
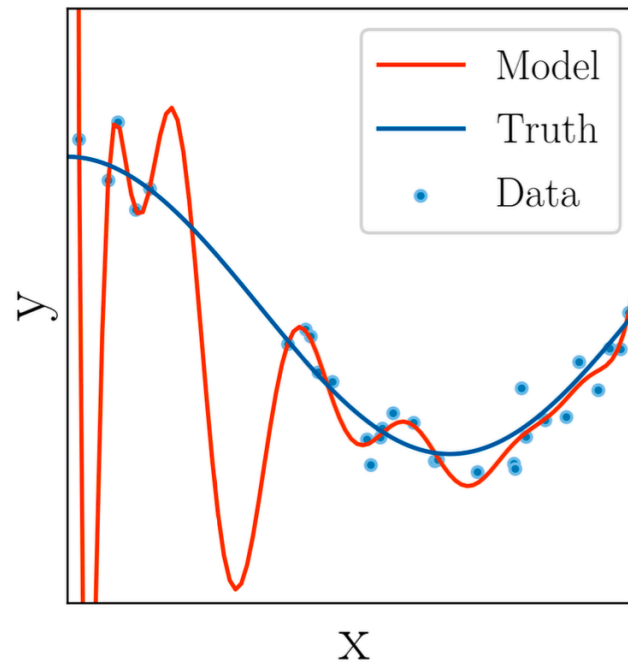
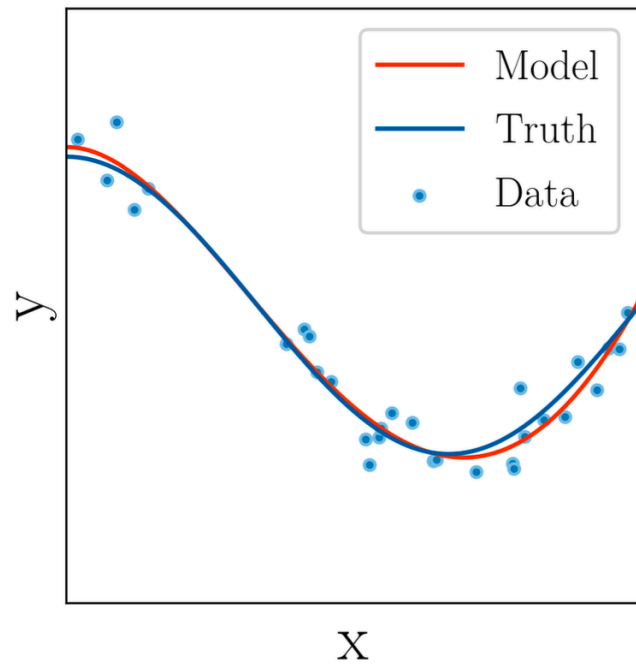
- Create synthetic samples by modifying samples in existing dataset by small changes
- Need to have understanding of meaningful symmetries
- E.g. in physics: Shift measurement within uncertainties, rotate cosmic showers, ...
- Other methods: High-dimensional interpolation (e.g. kNN-based augmentation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





Data Splitting: Motivation

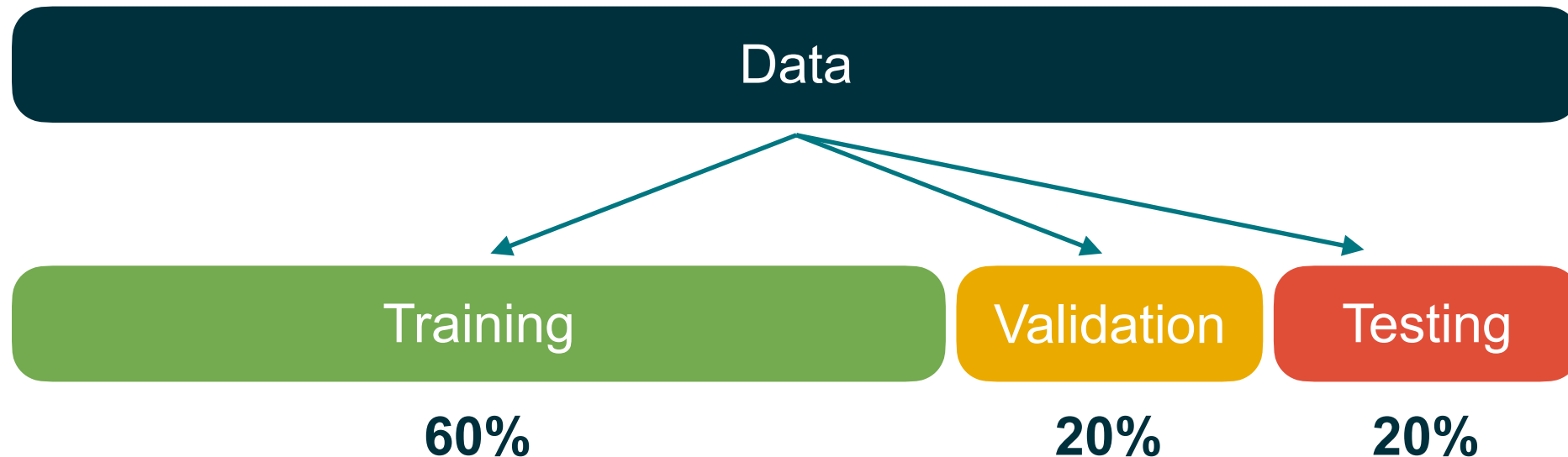
- A complex model can be fitted to any function if trained long enough
- It might perform great on training data but not generalize (**overtraining**)
- Need measure to prevent this!





Data Splitting: Train-Val-Test Split

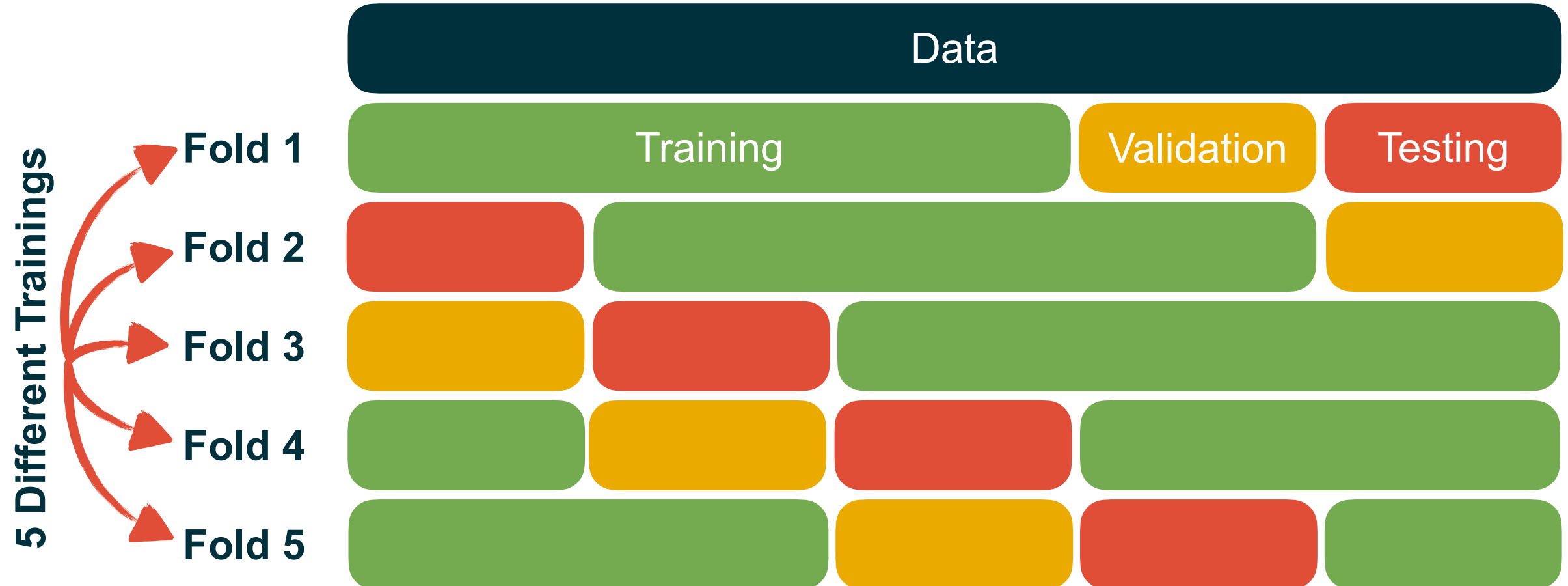
- Split data into three parts:
 - **Training set**: Train parameters of the model
 - **Validation set**: Monitor and tune training procedure (e.g. learning rate)
 - **Test set**: Estimate final performance, use only once!
- Pro-tip: Use deterministic splitting via event identifier (e.g. event number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





Data Splitting: Cross Validation

- Do not want to lose 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)**

An iceberg floating in the ocean. The tip of the iceberg, which is above the water line, is labeled 'Feed Forward'. The much larger part of the iceberg, which is submerged below the water line, is labeled 'Now: Advanced Architectures'. This submerged part contains several labels: 'ResNET', 'CNN', 'RNN', 'Graph', 'LSTM', and 'Transformer'. The water surface is a dark blue line, and the sky is a light blue gradient.

Feed Forward

Before: Simple Architectures

ResNET

CNN

RNN

Graph

LSTM

Transformer

Now: Advanced Architectures

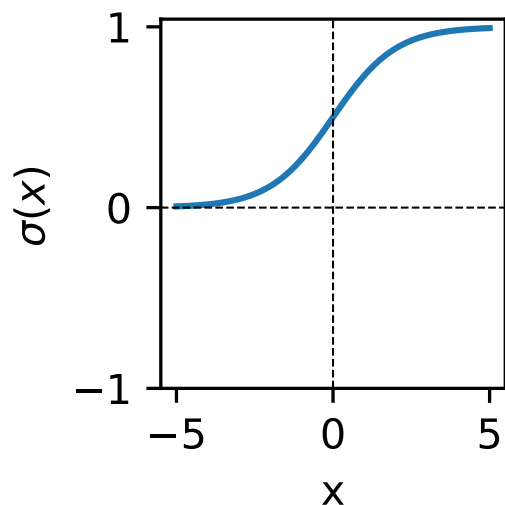
- More resource/parameter efficient
- Easier to train
- Converge faster

Activation Functions

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

Logistic function

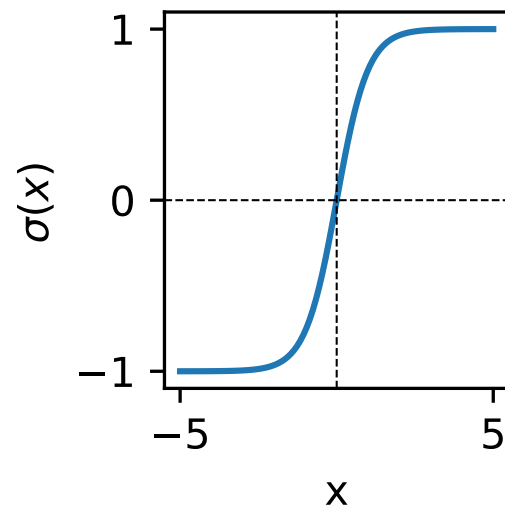
$$\sigma(x) = \frac{1}{1 + e^{-x}}$$



Good for probabilities!

Tanh

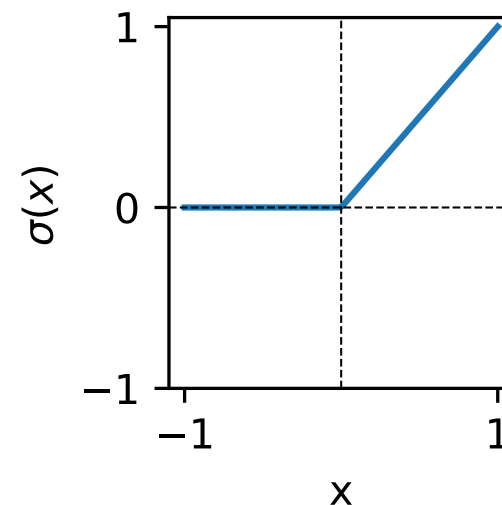
$$\sigma(x) = \frac{e^{2x} - 1}{e^{2x} + 1}$$



Positive & negative outputs!

ReLU

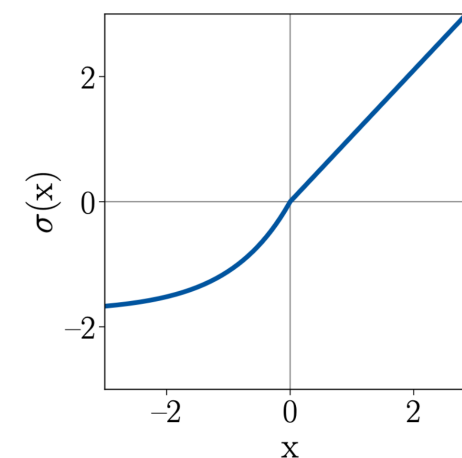
$$\sigma(x) = \max(0, x)$$



Very easy and fast!

SELU

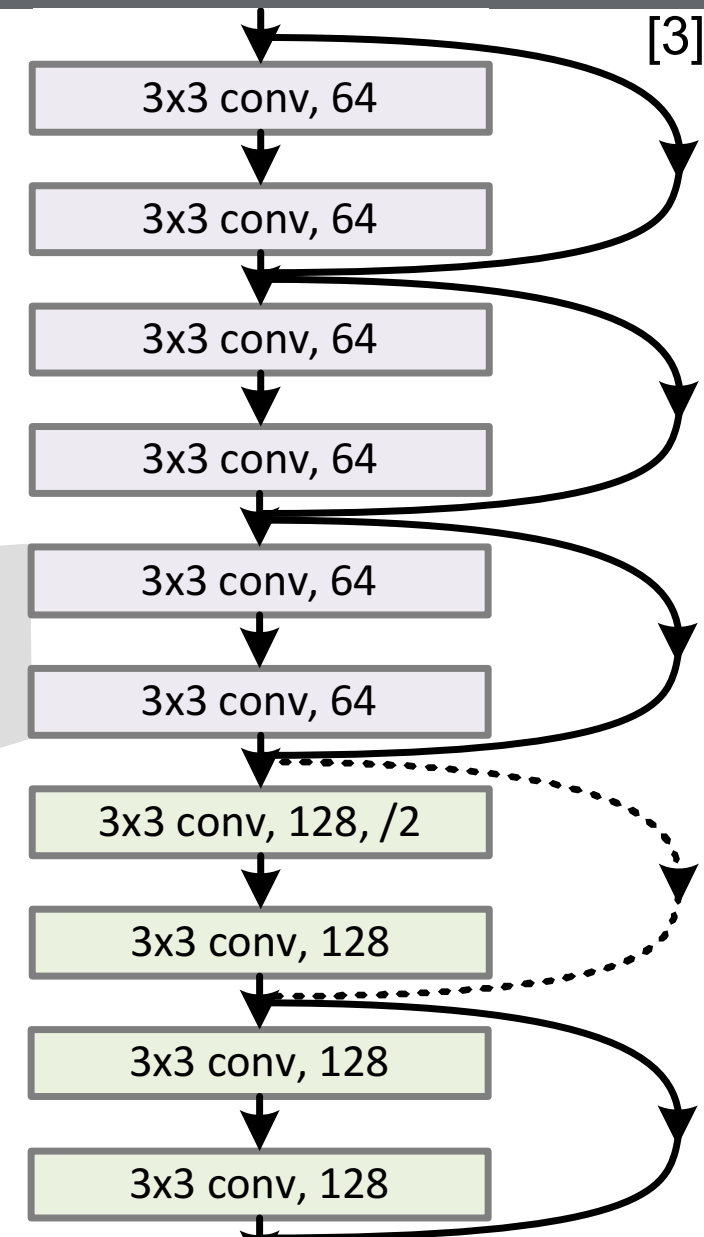
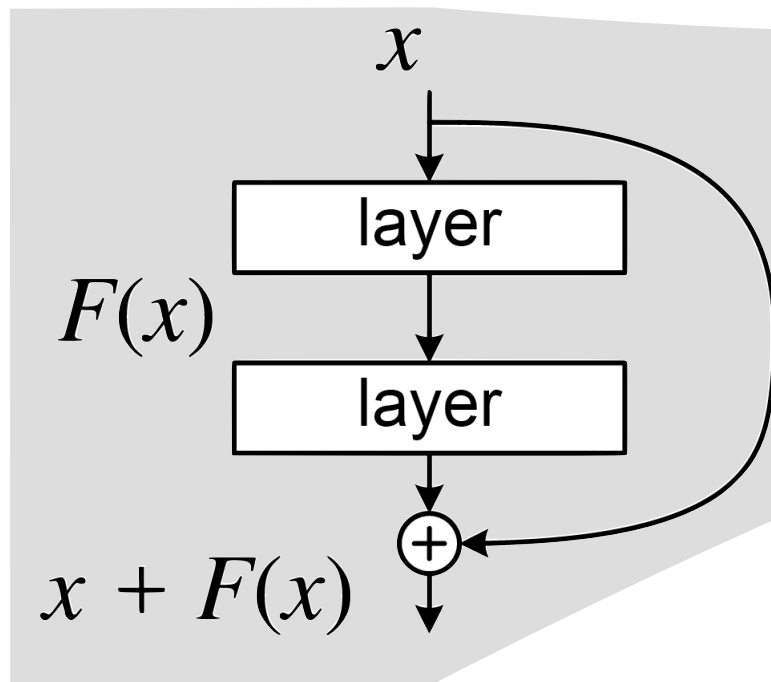
$$\sigma(x) = \begin{cases} \lambda x & |x > 0 \\ \lambda \alpha (e^x - 1) & |x \leq 0 \end{cases}$$



Self-normalizing!

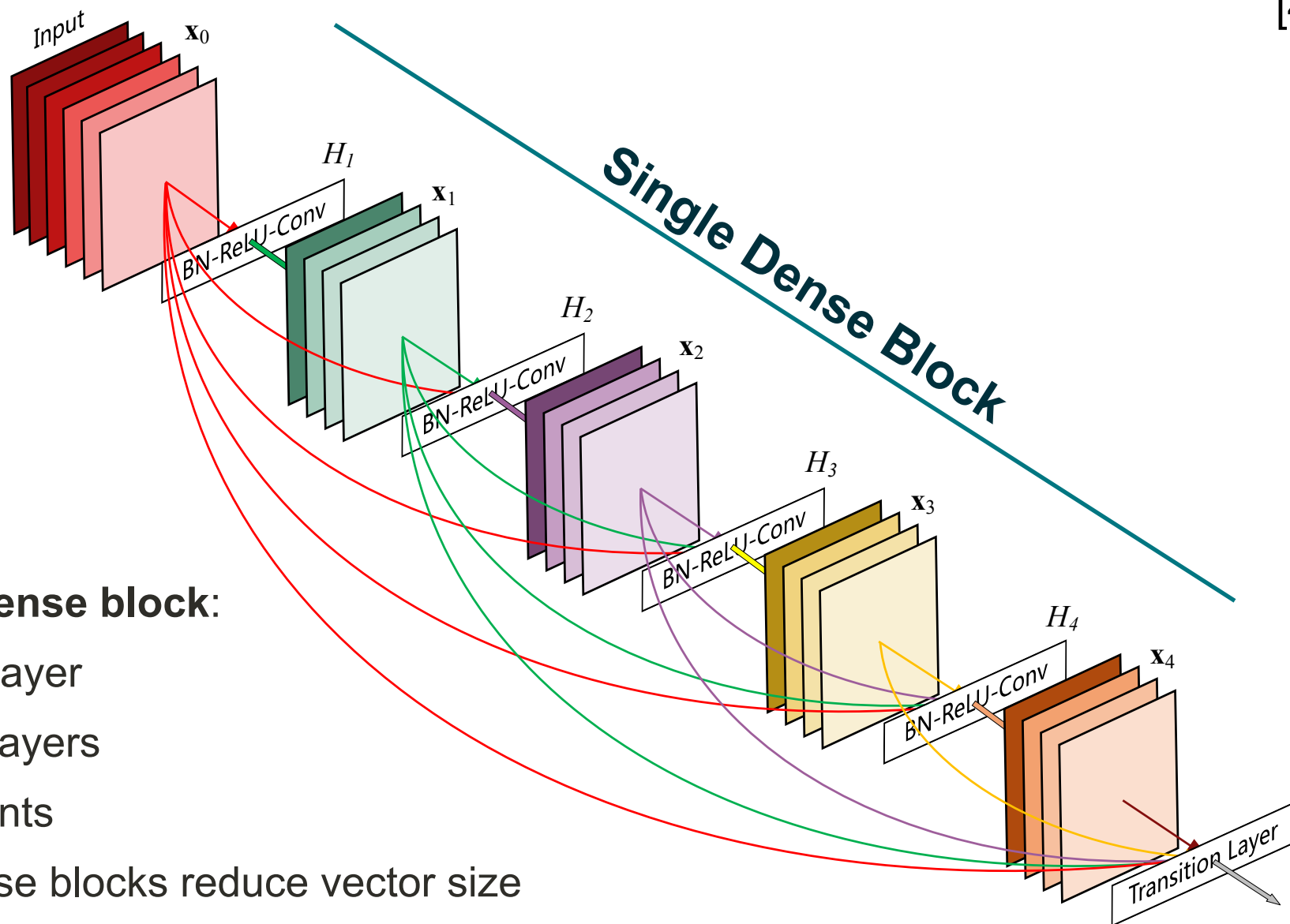
Residual Neural Network

- 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

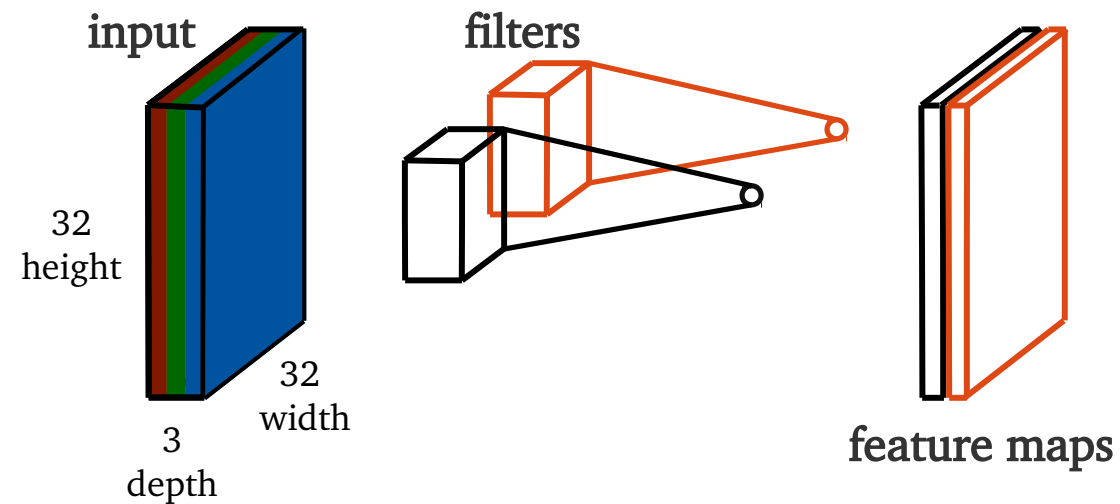
[4]



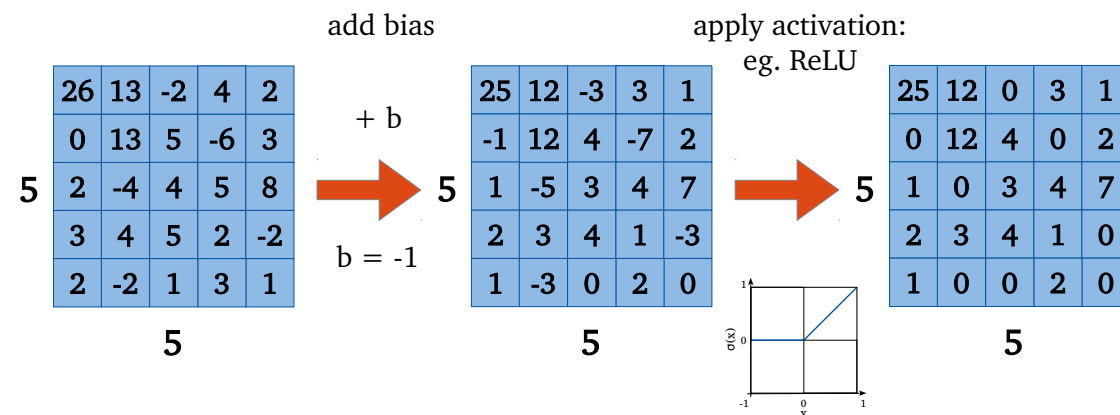
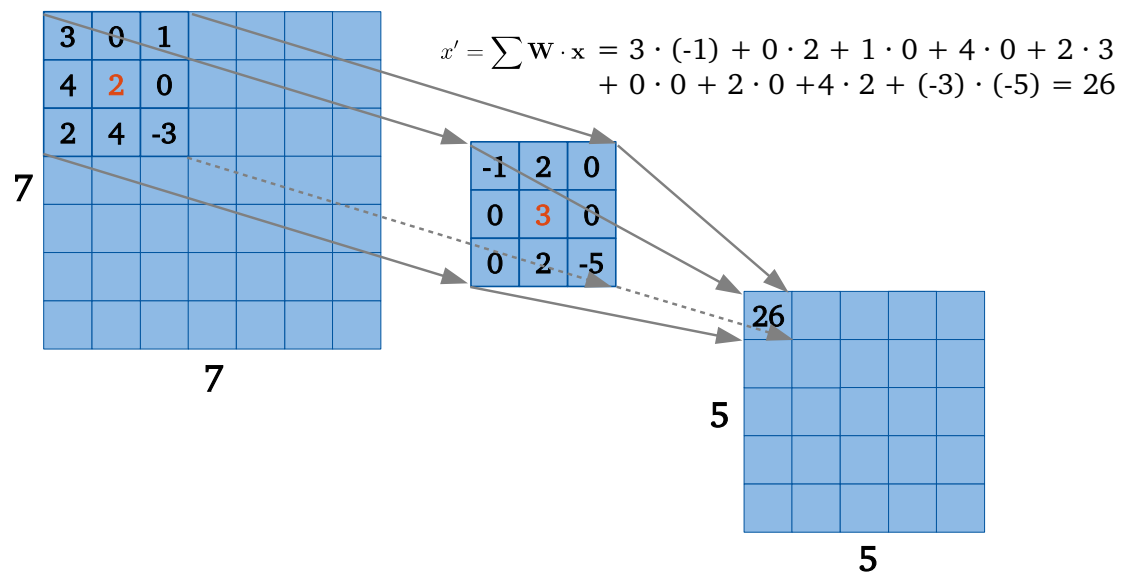
- Apply shortcut to all layers in **dense block**:
 - Reuse features from each layer
 - Combine features from all layers
 - Easy propagation of gradients
- Transition Layers between dense blocks reduce vector size

Convolutional Neural Network (1/2)

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



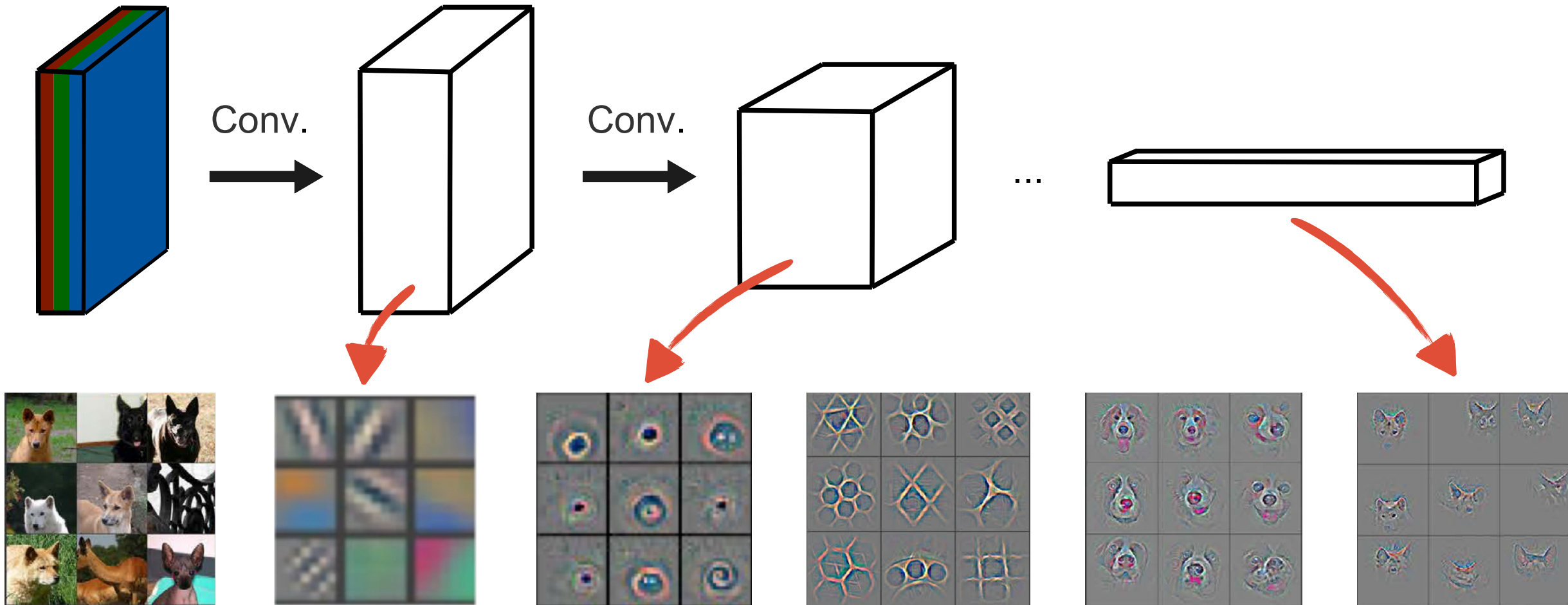
Step by step:








Convolutional Neural Network (2/2)

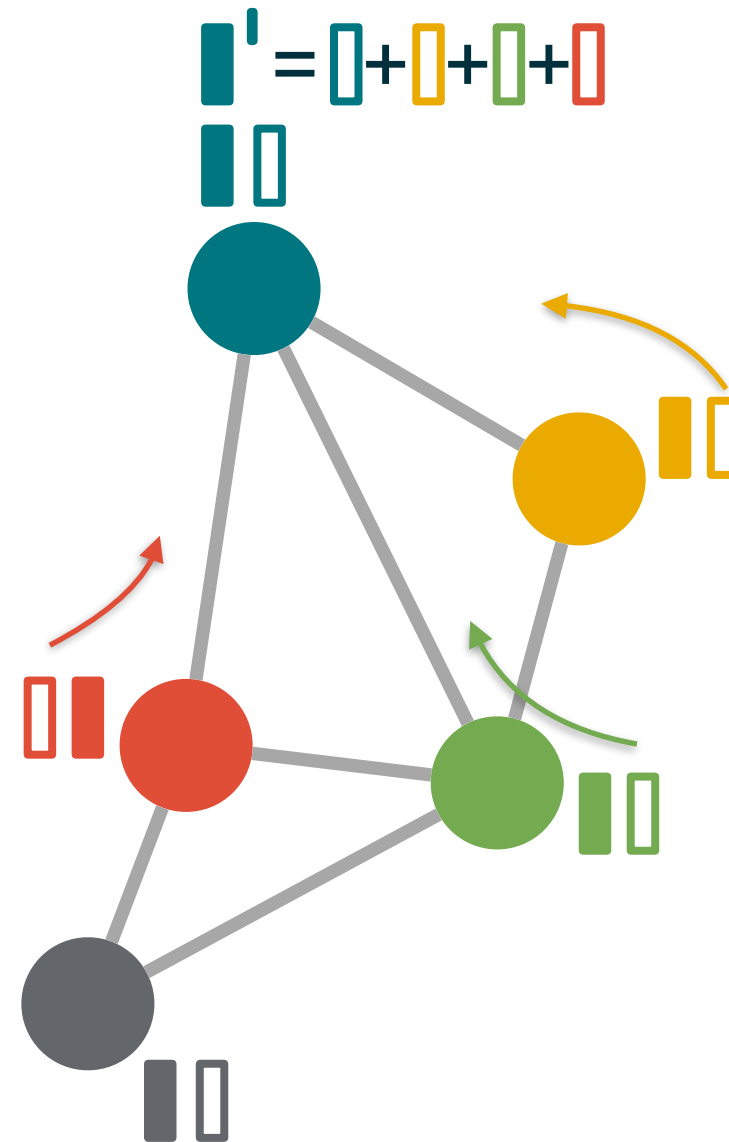
- Multiple layers of filters extract more-and-more abstract features
- Usually have pyramidal shape: Decrease spatial extent & increase feature space

[5]





Learning on Graphs (1/2)

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

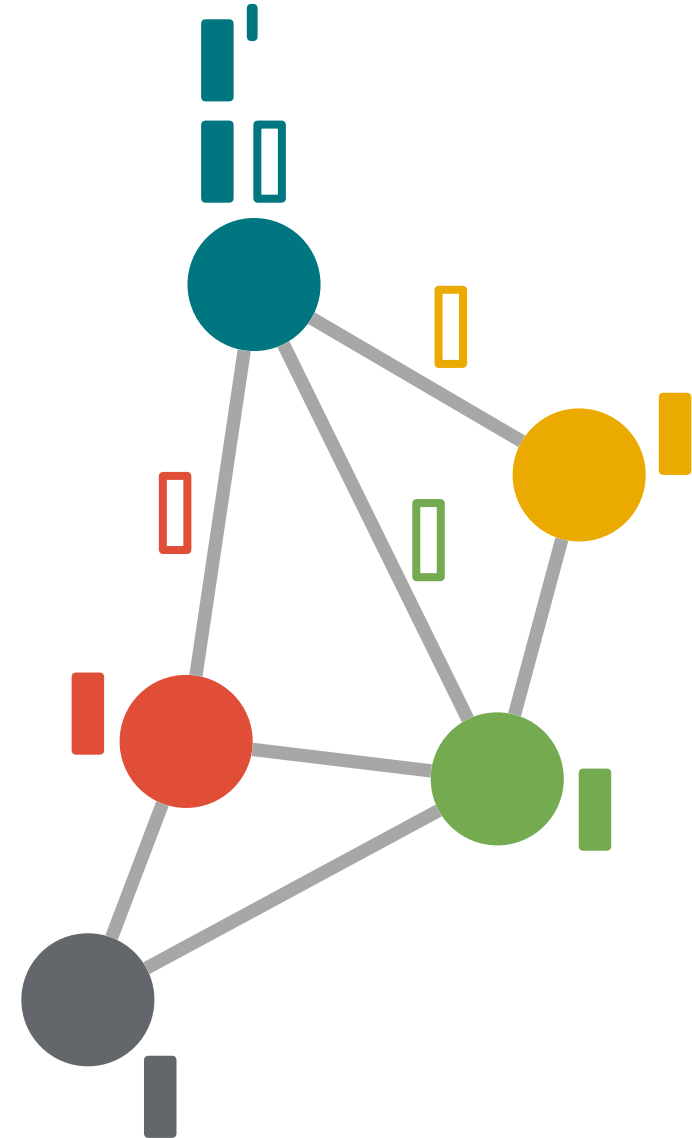


Learning on Graphs (2/2)

- **Graph:**
 - Nodes: Have features 
 - Edges: Connect nodes, can have features 
- **Learning by updating each node:**
 - Embed edges $\square = \phi(\square)$ - Isotropic
 - $\square = \phi(\square, \square)$ - Anisotropic
 - Aggregate embeddings $\oplus \square, \square, \square$
 - Embed aggregations $\square' = \psi(\square, \oplus \square, \square, \square)$

ϕ and ψ can be DNNs!

use multiple rounds

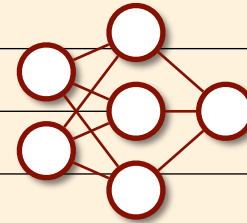


Introduction to Machine Learning: Part II - Take Away

- 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



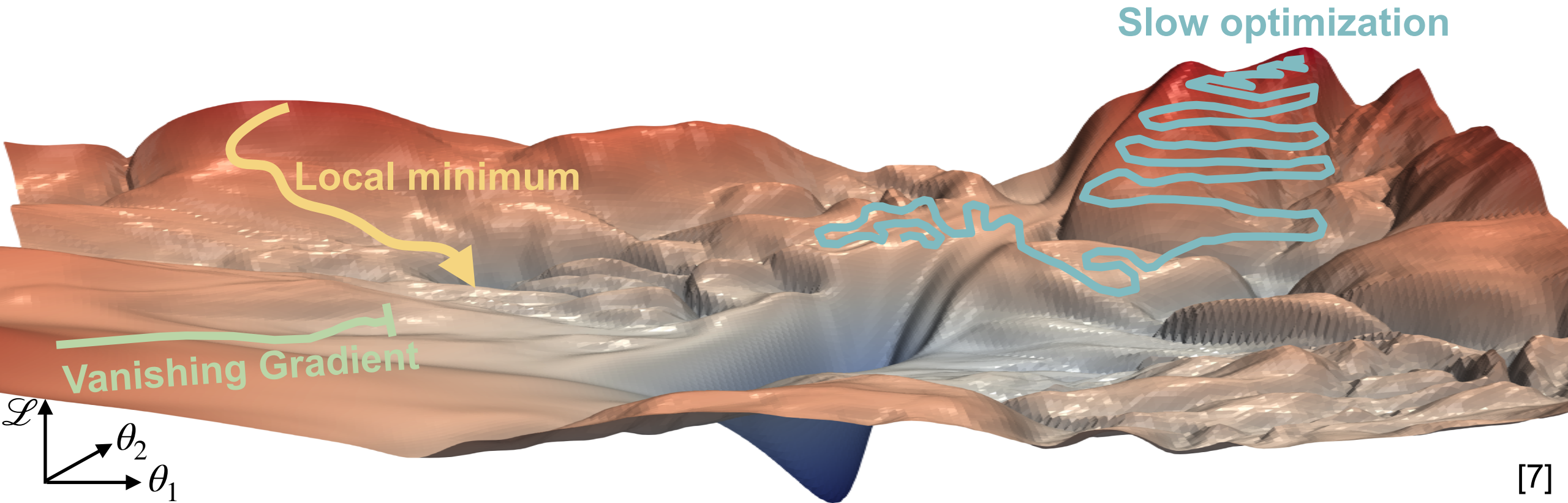
Training (NEW!)

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

Reminder:

Gradient
Descent

$$\theta \rightarrow \theta - \alpha \nabla_{\theta} \mathcal{L}$$



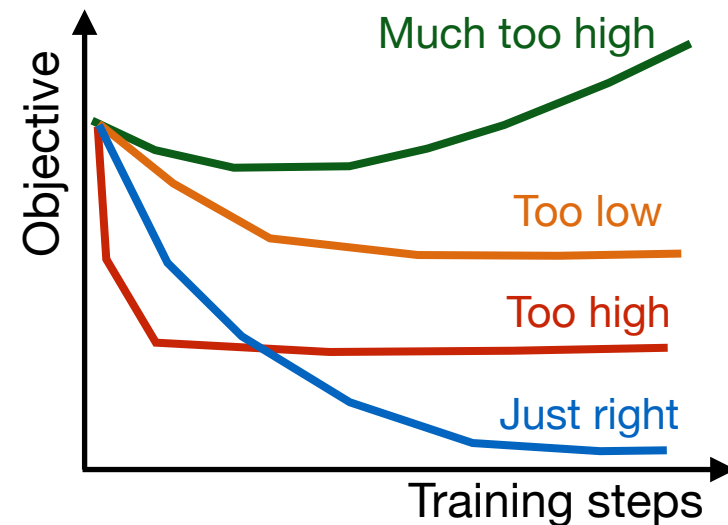
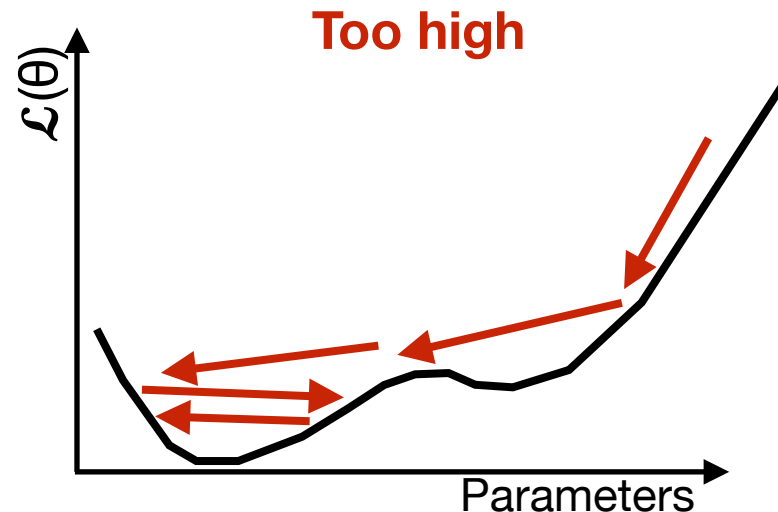
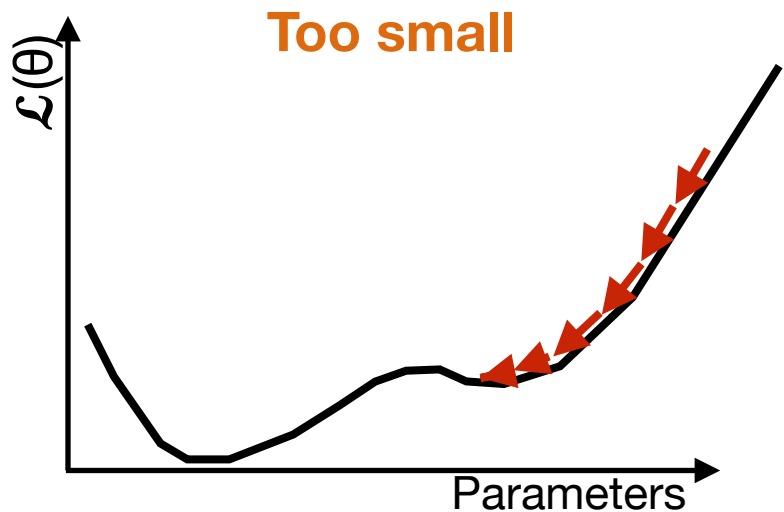
Learning rate

- 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

$$\theta \rightarrow \theta - \alpha \nabla_{\theta} \mathcal{L}$$



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)

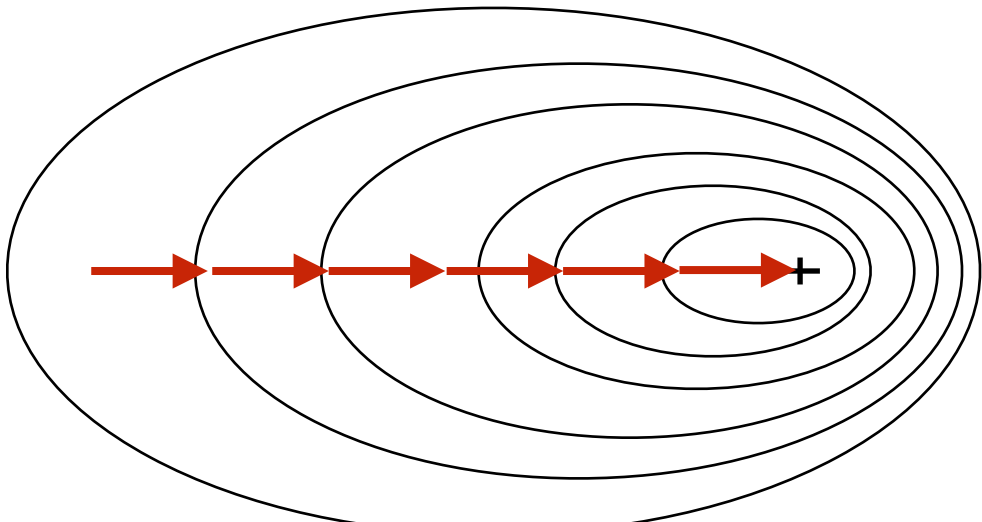
Pro

- More parameter updates
- Stochasticity helps escape local minima

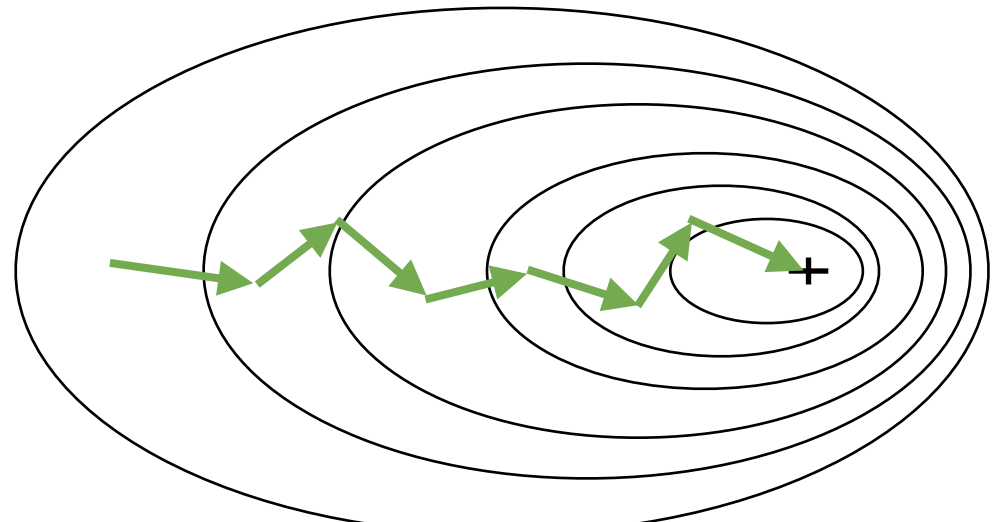
Contra

- 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

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

RMSprop

Decay memory of past gradients: good to train longer

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

Adam

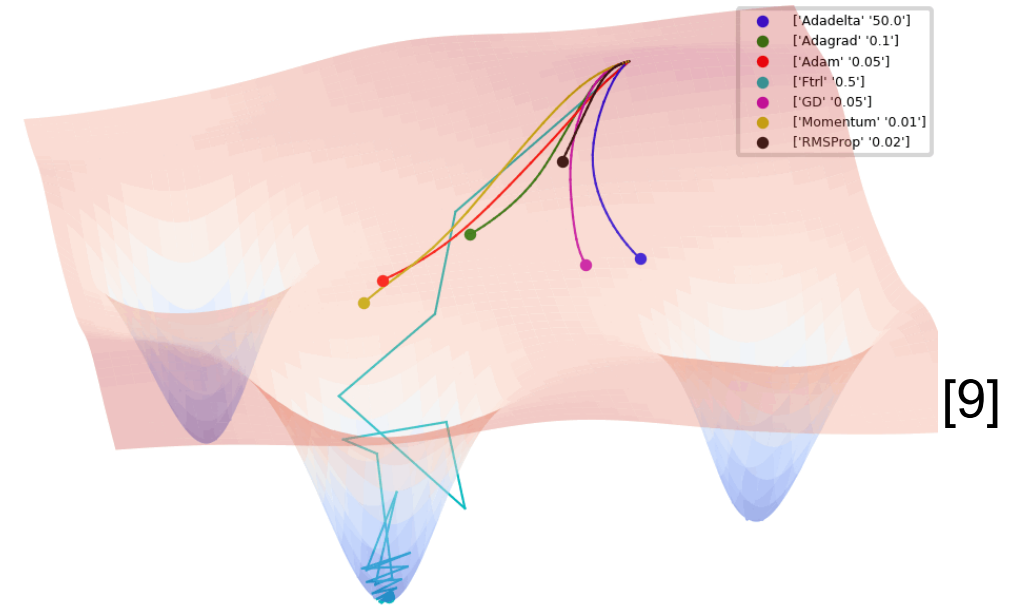
Combines Momentum and RMSprop:

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

Reminder:

Gradient
Descent

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



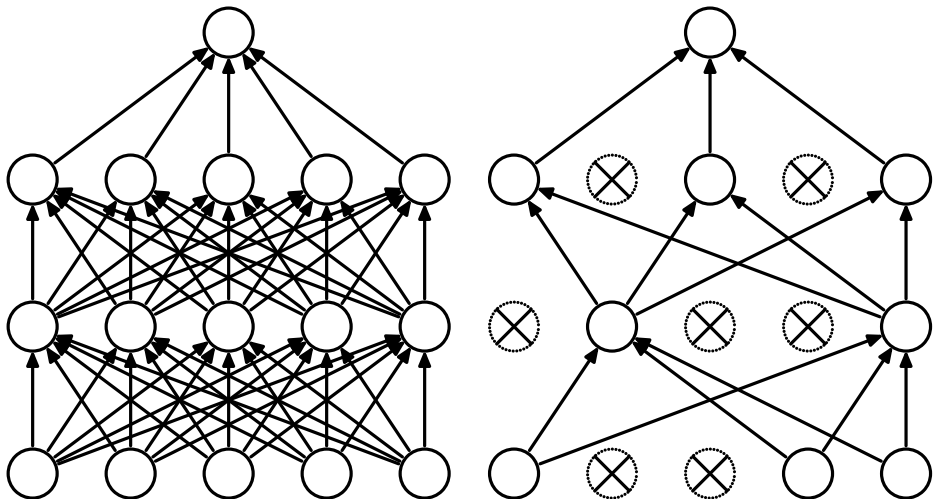
Regularization

[10,11]

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

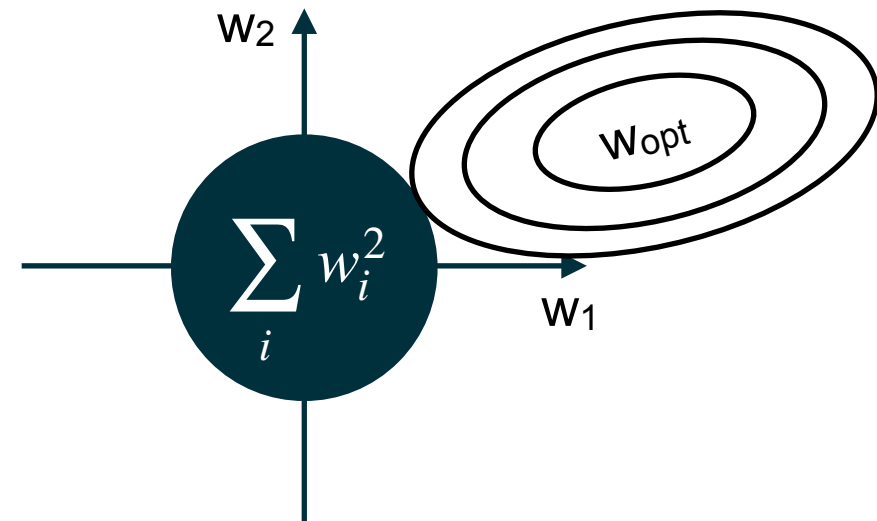
Dropout

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



Weight Regularization

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

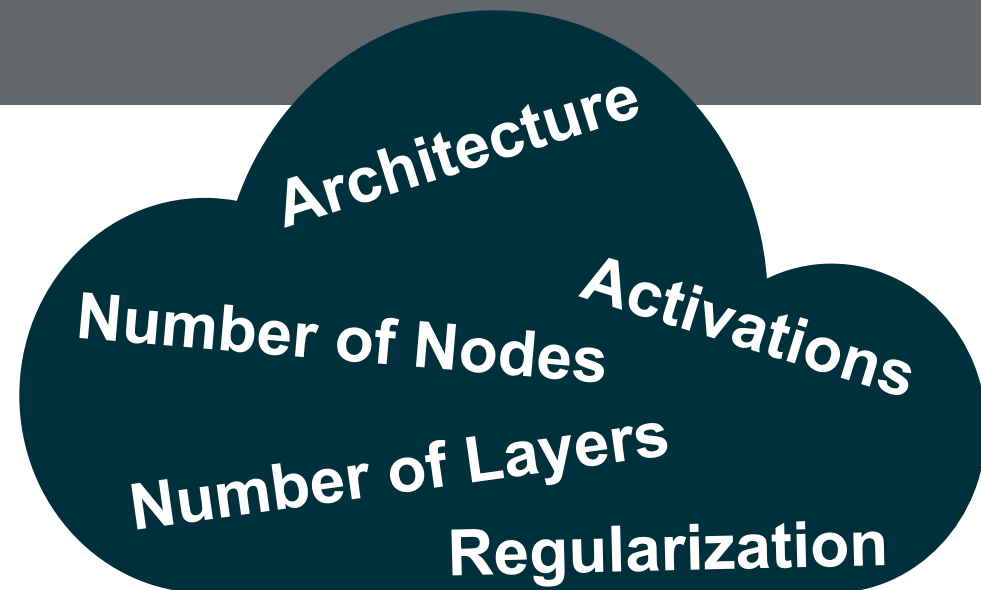


Hyperparameter Optimization

- 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!



Which to use?

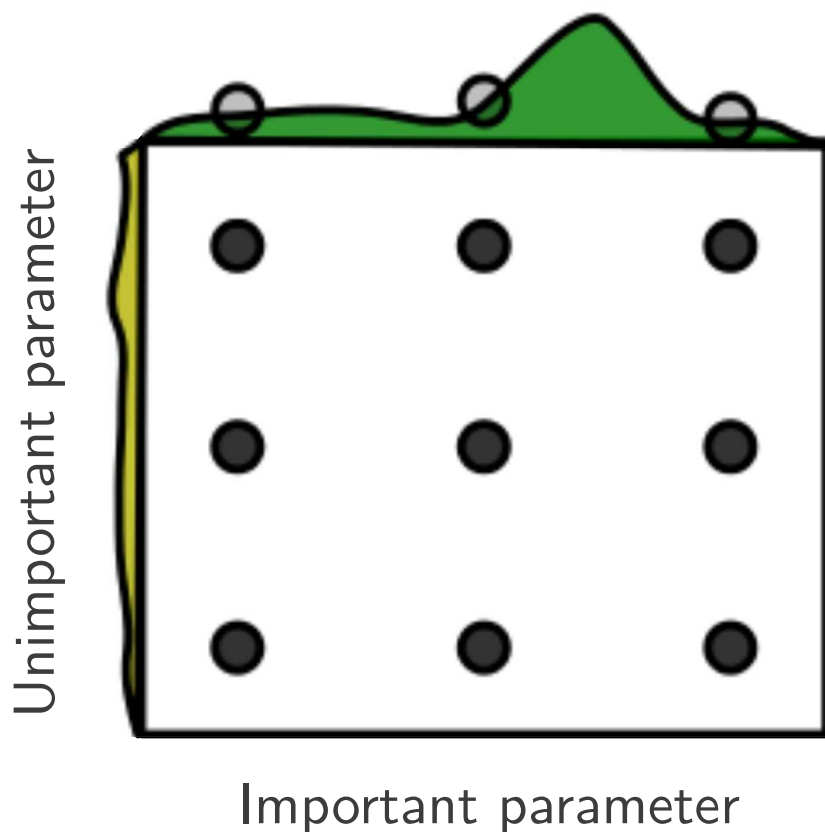


Hyperparameter Optimization - Grid and Random

[12]

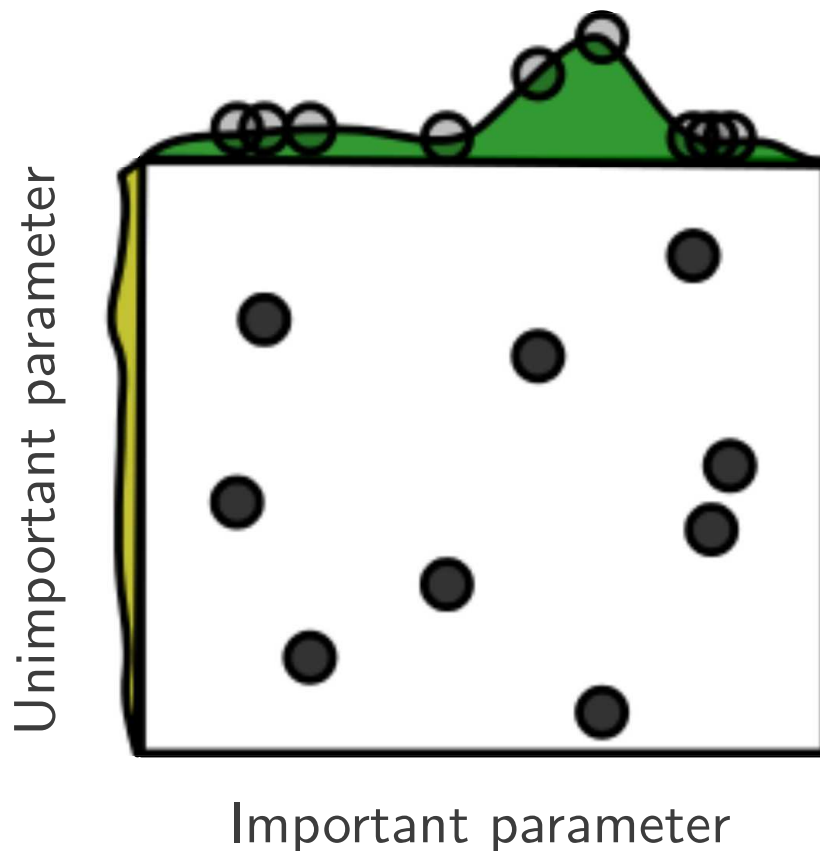
Grid Search

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



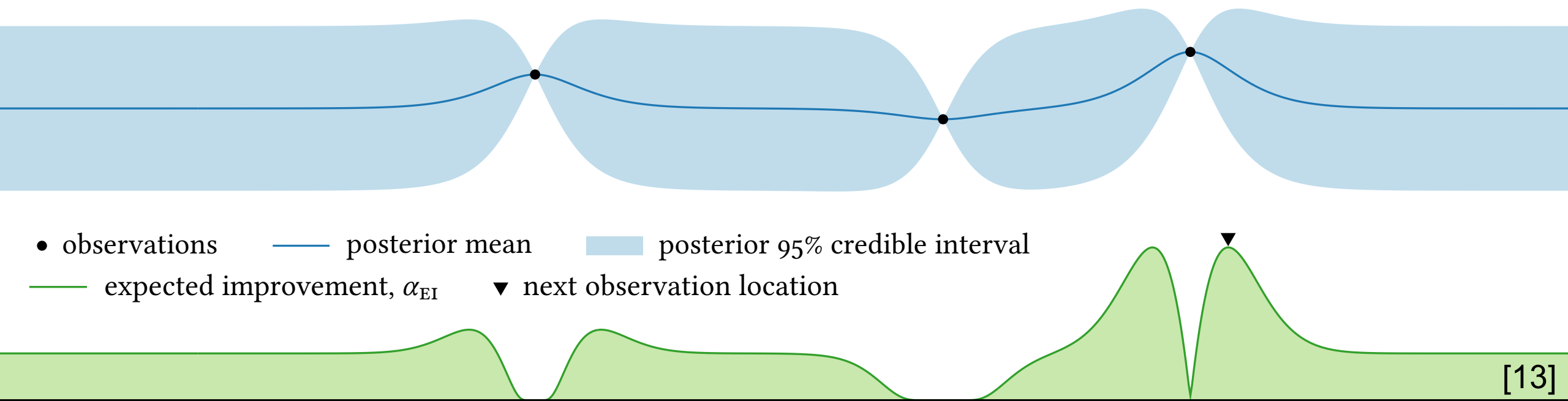
Random Search

- 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!



Deep Learning Software

[14,15,16]

- Three **popular, easy to use, open-source** software libraries
- All suited for ML-driven Physics Research but different pros and cons

TensorFlow

- Developed by Google
- Comprehensive ecosystem
- Large industry applications



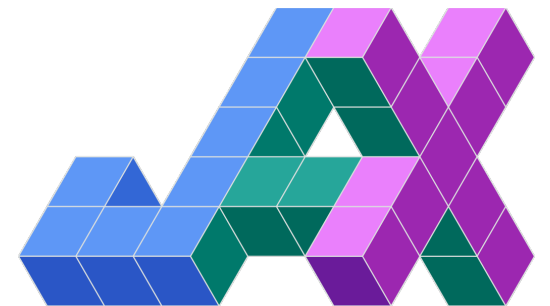
PyTorch

- Developed by Meta
- Fast & flexible prototyping
- ML & Physics Research



JAX

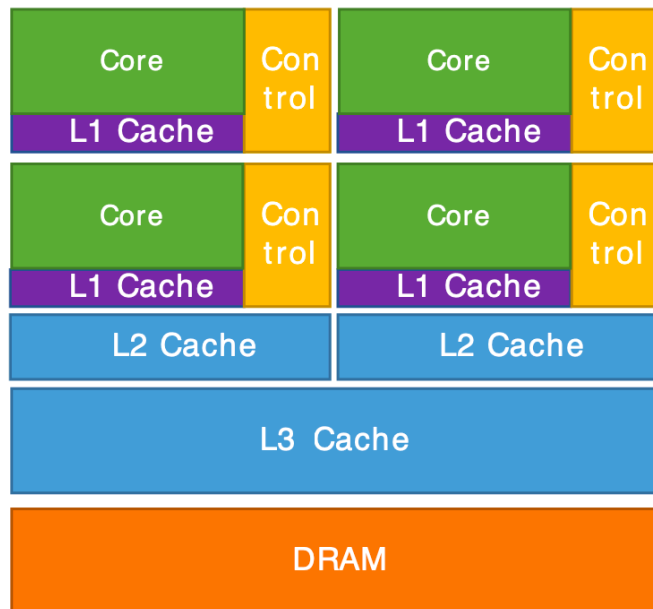
- Developed by Google
- Functional programming
- ML Research & Speed



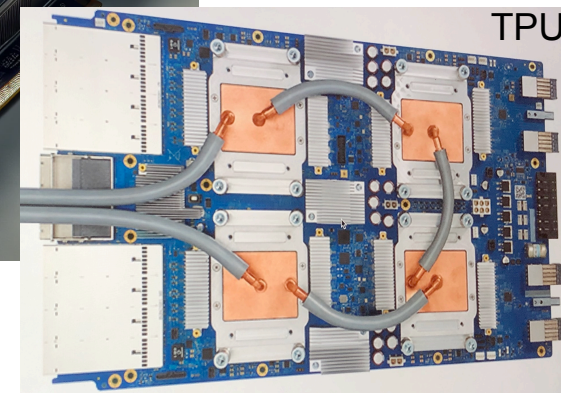
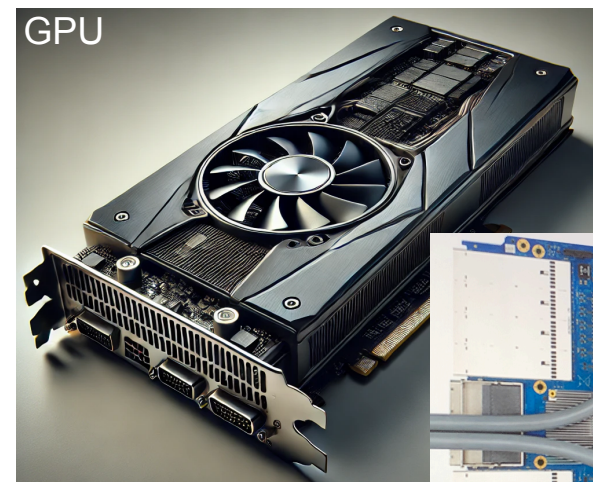
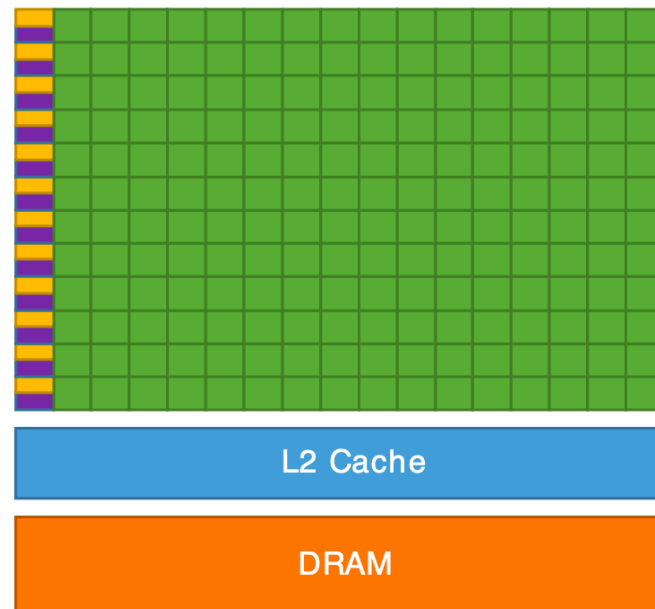
Hardware: GPUs - The Backbone of Machine Learning

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

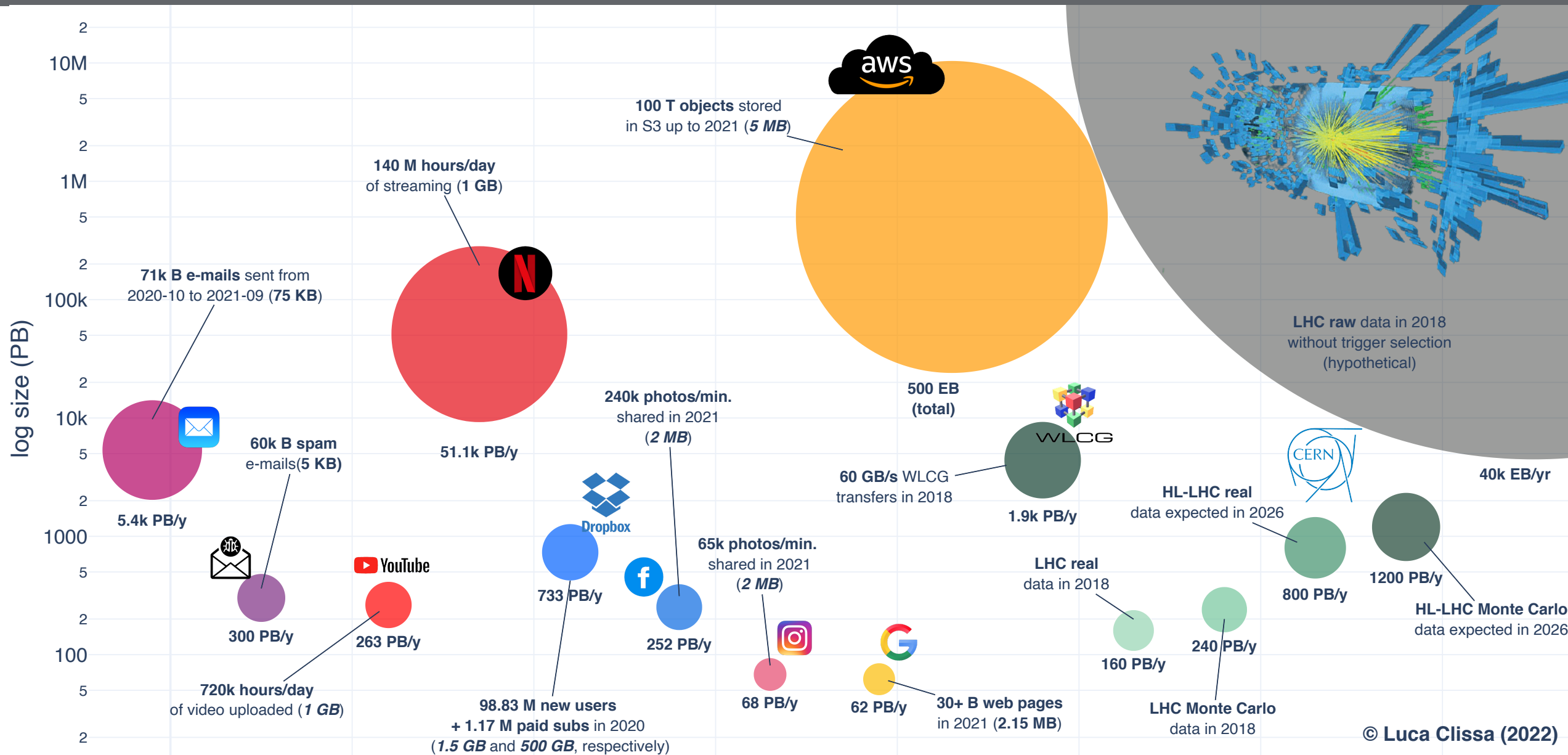
CPU



GPU



How Big is BIG DATA?

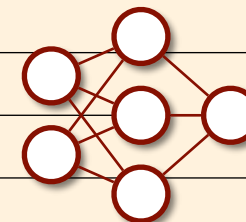


Introduction to Machine Learning: Part II - Take Away

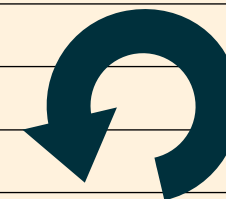
- Know your data (an hour here can save you weeks!)
- Preprocess your data (fix outliers, missing values, normalize, augment)
- Split your data (train val test) before you do anything else!



- 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, ...), GPUs, Big Data
 - Automize every step! (Data Download → Paper Document)

Citations

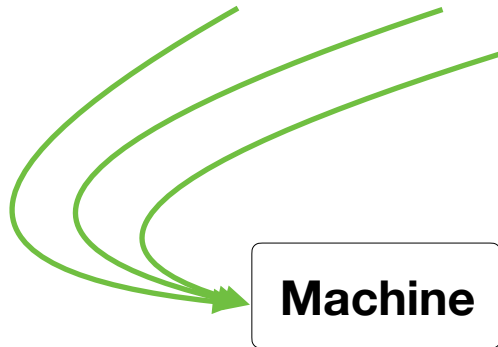
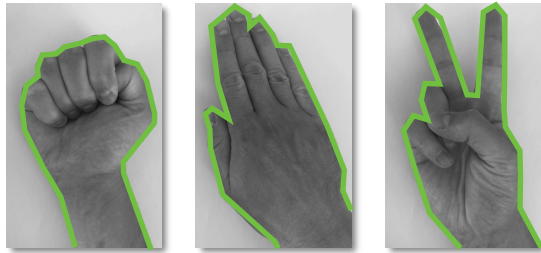
- [1]: **Berkeley Lab History** Berkeley Lab, [Link](#) (accessed 11.08.24)
- [2]: **Deep Learning in Physics Research** Martin Erdmann et al., Lecture (RWTH Aachen University). Apr. 2022. [link](#) (accessed 06.01.23)
- [3]: **Deep Residual Learning for Image Recognition** Kaiming He, Xiangyu Zhang, 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]: **Tensorflow Brand Guidelines**, [link](#) (accessed 03.08.25)
- [15]: **Pytorch logo** Author <https://github.com/soumith>, [link](#) (accessed 03.08.25)
- [16]: **Logo of the Google JAX machine learning framework**, Google LLC <https://github.com/google/jax>, [link](#) (accessed 03.08.25)
- [17]: **Design: GPU vs. CPU** Cornell Virtual Workshop [link](#) (accessed 11.08.24)
- [18]: **Tensor Processing Unit 3.0 (Personal Picture)** Zinskauf, [CC BY-SA 4.0](#), via Wikimedia Commons

Backup

From Machine Learning to Deep Learning

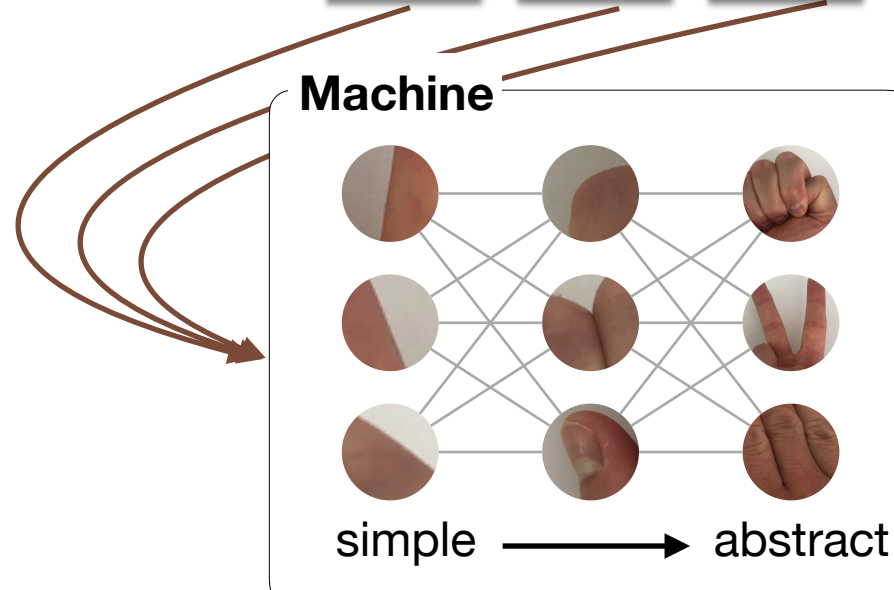
Classic Machine Learning

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



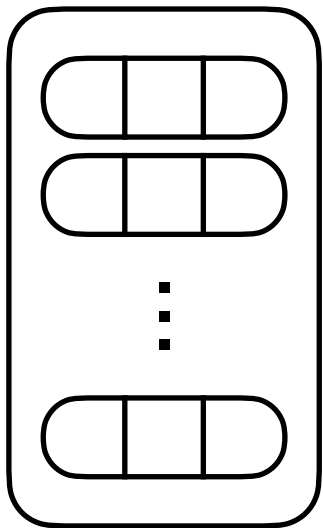
Deep Learning

- Machine extracts features
 - fast
 - stable
 - well scalable



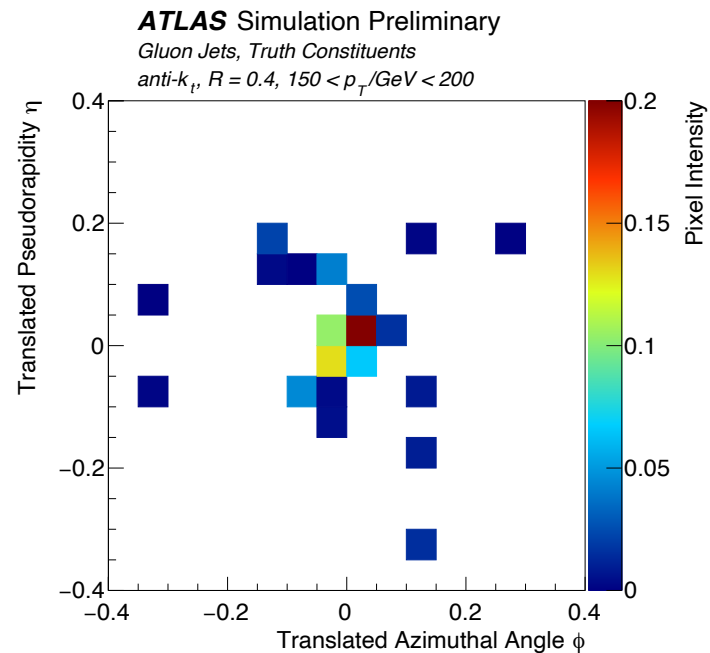
How to represent the data?

As Lists?



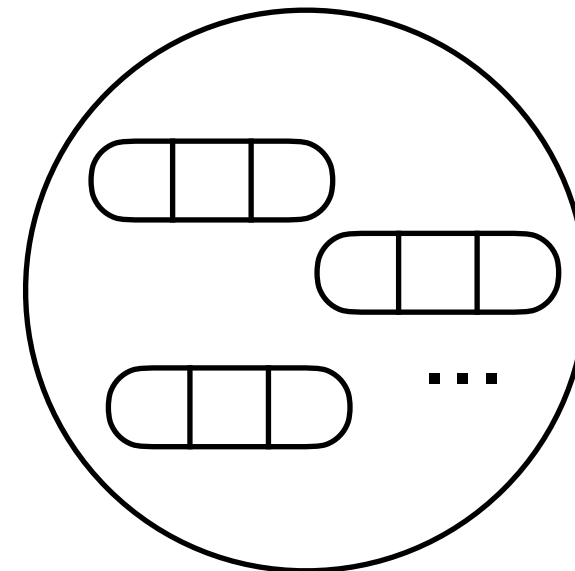
Pro: Easy to use
Con: Which ordering?

As Images?



Pro: Geometric information
Con: Very sparse

As Sets?



Pro: Works with sparse data
Con: No geometric information

As Point Clouds...

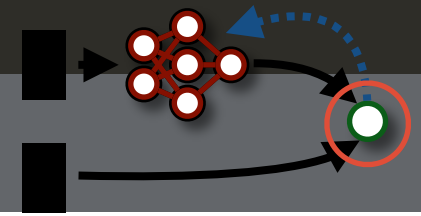
- Unordered set of objects in metric space
- **Why is this nice? Objects can be our detector hits!**

Works with
sparse data

Respects permutation
invariance

Each object can have
position, time, ...



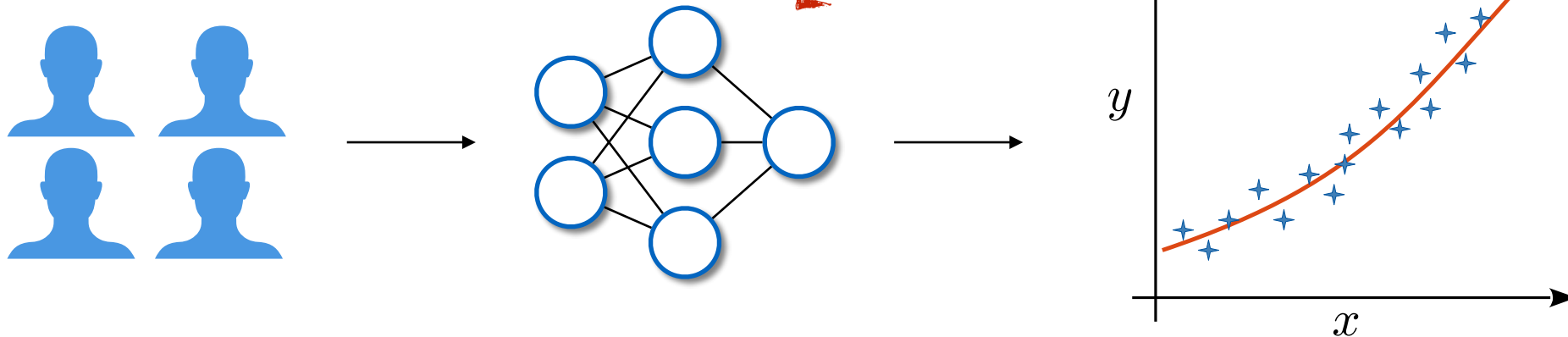


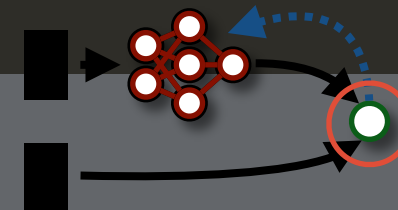
Regression: Predict continuous feature

- 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:

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



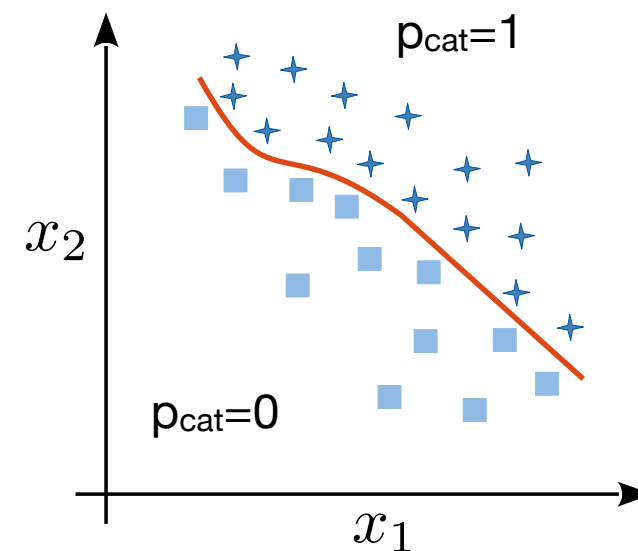
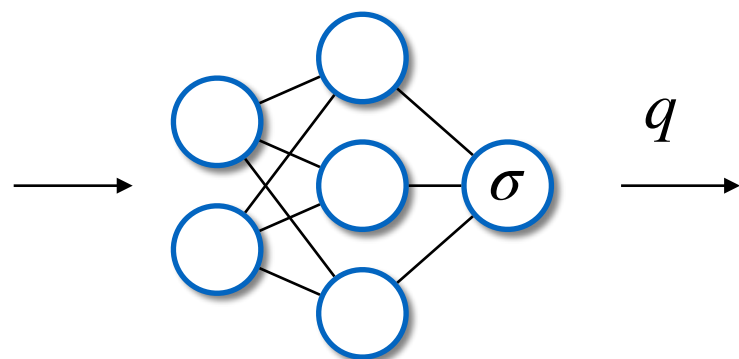


Classification: Predict discrete classes

- 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:

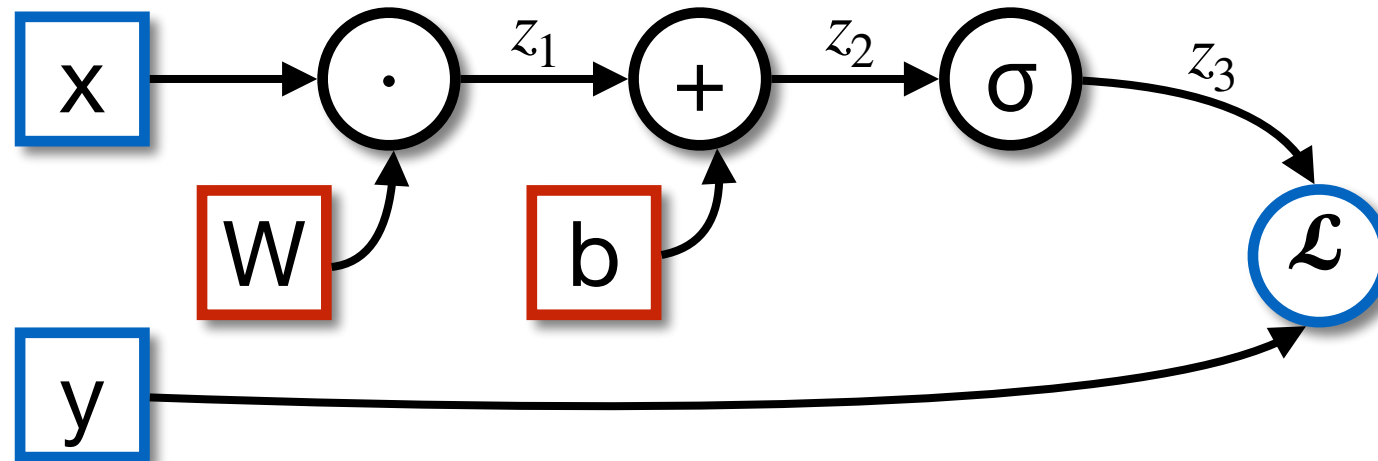
$$\mathcal{L} = -\frac{1}{n} \sum_i^n \sum_j^c p_{ij} \cdot \log(q_{ij})$$



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

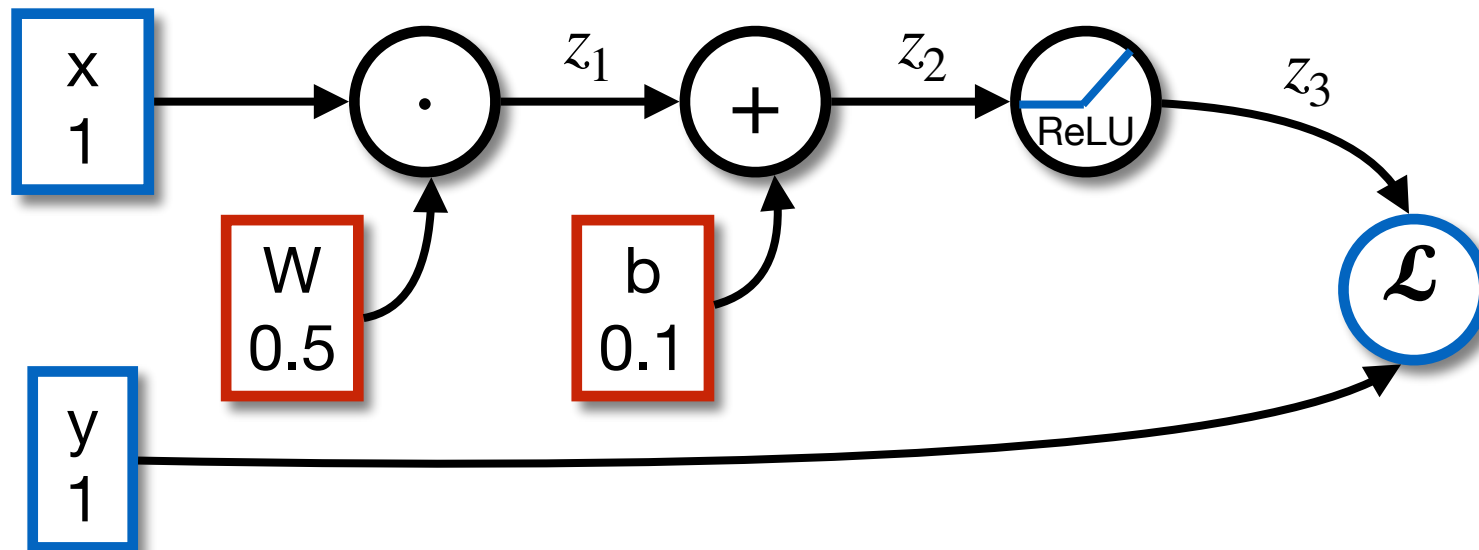
Example: $y^{pred} = z_3 = \sigma(Wx + b)$



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

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

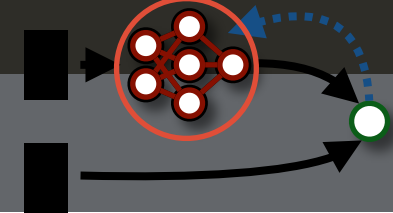
$$\partial \mathcal{L} / \partial z_3 = 2(z_3 - y) = -0.8$$

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

$$\partial z_2 / \partial z_1 = 1$$

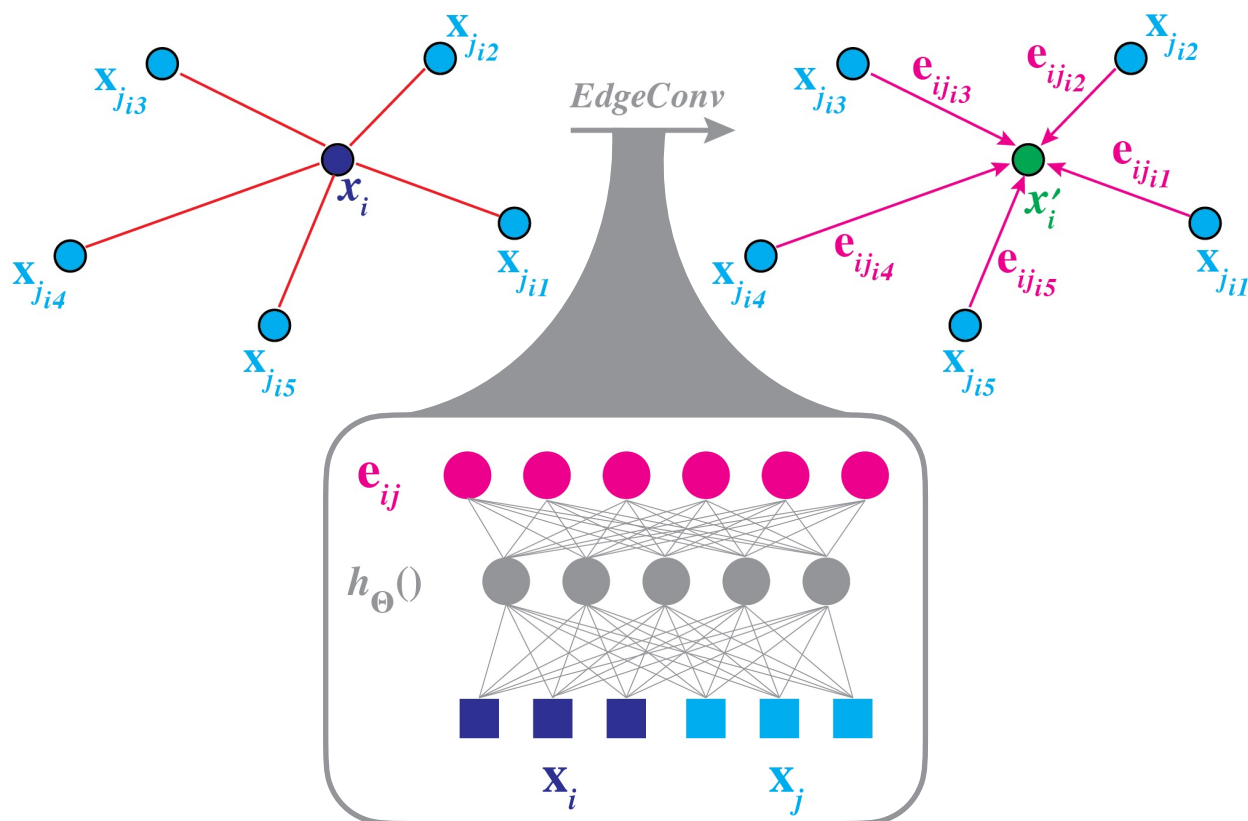
$$\partial z_1 / \partial W = x = 1$$

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

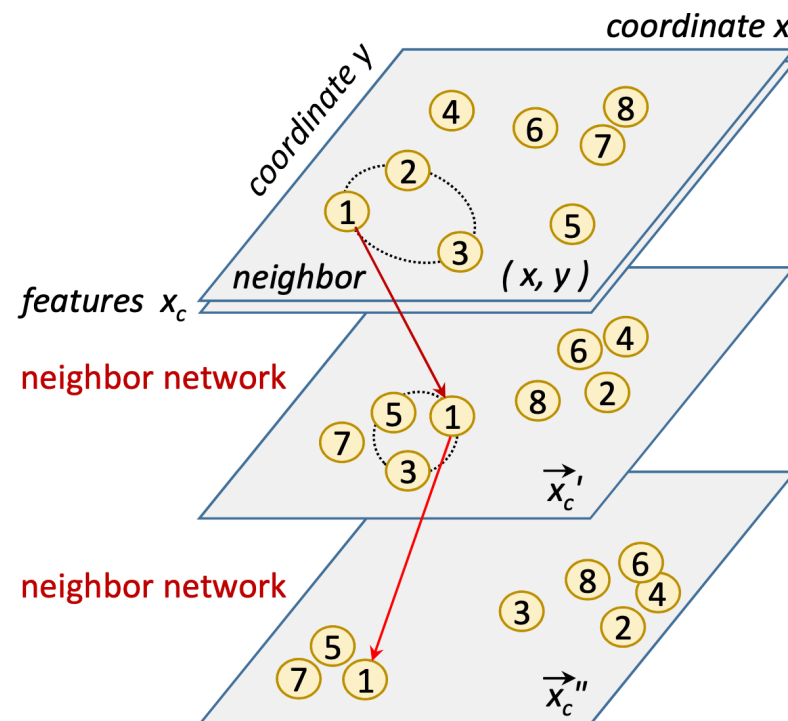


Graph Neural Network: Edge Conv

- 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



Learning on Graphs (Interaction Network)

- **What is a graph:**

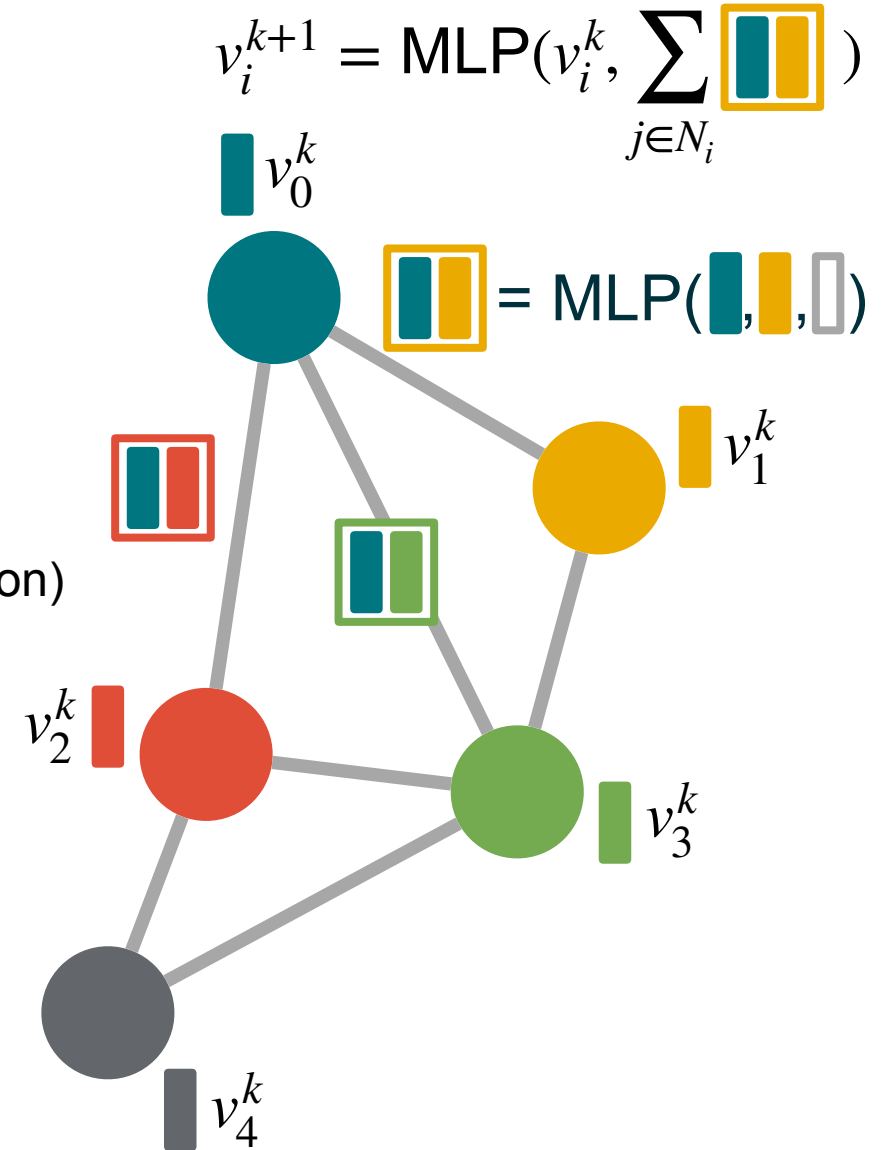
- Nodes: Have features 
- Edges: Connect nodes, can have features 

- **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 embeddings $E_i^{k+1} = \sum_{j \in N_i} e_{ij}^{k+1}$

- Embed aggregations $v_i^{k+1} = \text{MLP}(v_i^k, E_i^{k+1})$

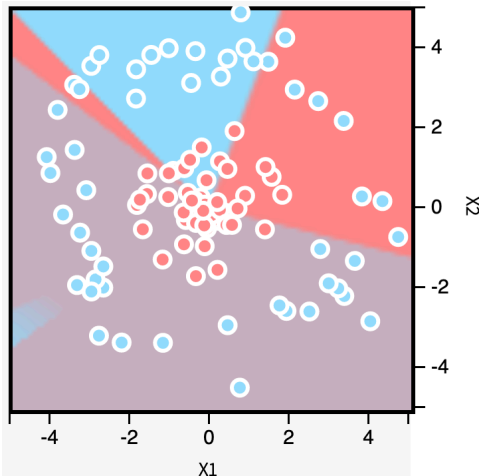


Parameter initialization

- 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: $\sigma^2 = 2/n_{in}$

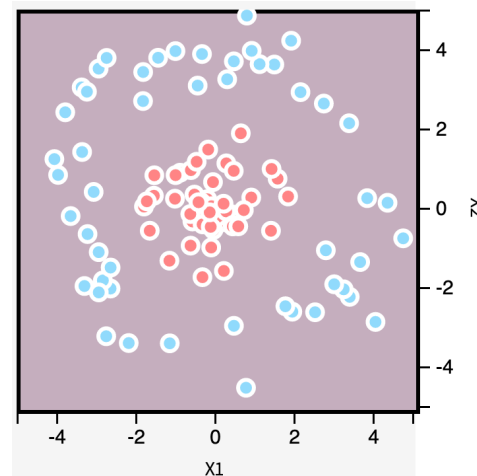
Weights to large

- Exploding Signals:



Weights to small

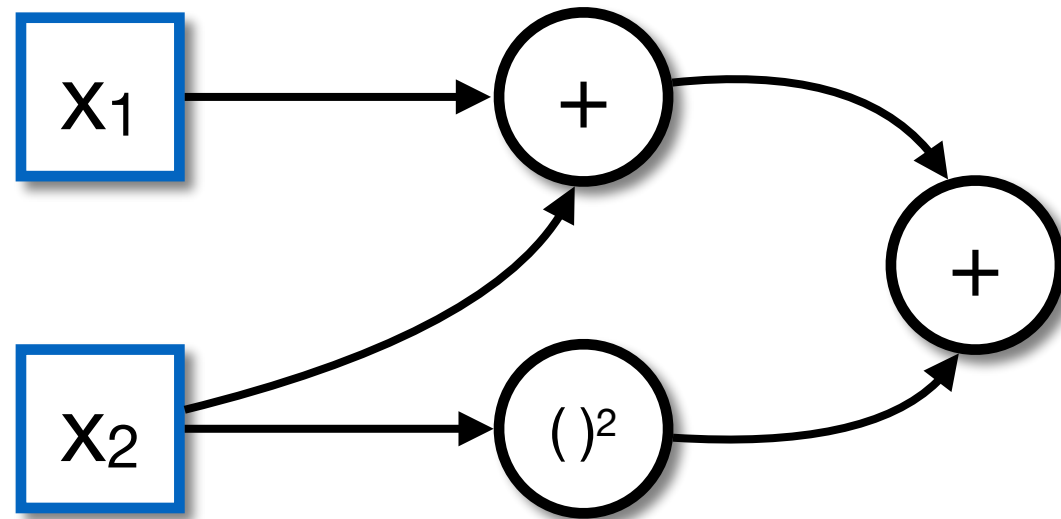
- Vanishing Signals:



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 θ_i
- Pro: Learning rate does not have to be tuned or set specifically
- Con:
 - G is monotonically increasing over number of epochs
 - Therefore learning rate decay to zero