Machine Learning

Lesson 2

School on Data Science in Fundamental Physics, IGFAE/USC, Spain

Dr. Pietro Vischia
pietro.vischia@cern.ch
@pietrovischia
https://vischia.github.io/

Universidad de Oviedo

ICTEA

Supported by project RYC2021- 033305-I funded by

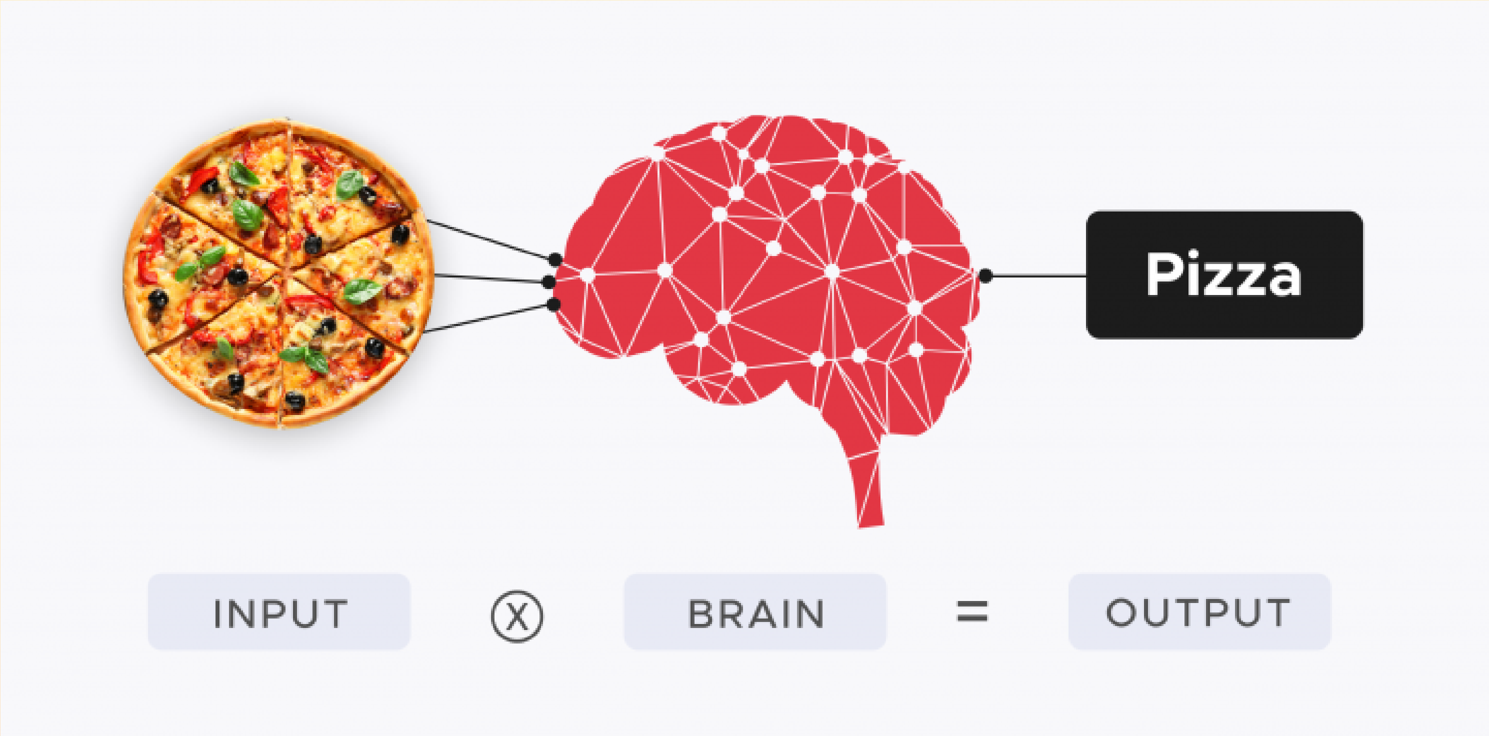
Gobierno de España, Ministerio de Ciencia e Innovación, Agencia Estatal de Investigación

If you are reading this as a web page: have fun! If you are reading this as a PDF: please visit

https://www.hep.uniovi.es/vischia/persistent/2024-06-03to07_MachineLearningAtDataScienceSchoolIGFAE_vischia_2.html

to get the version with working animations

Brain activity...

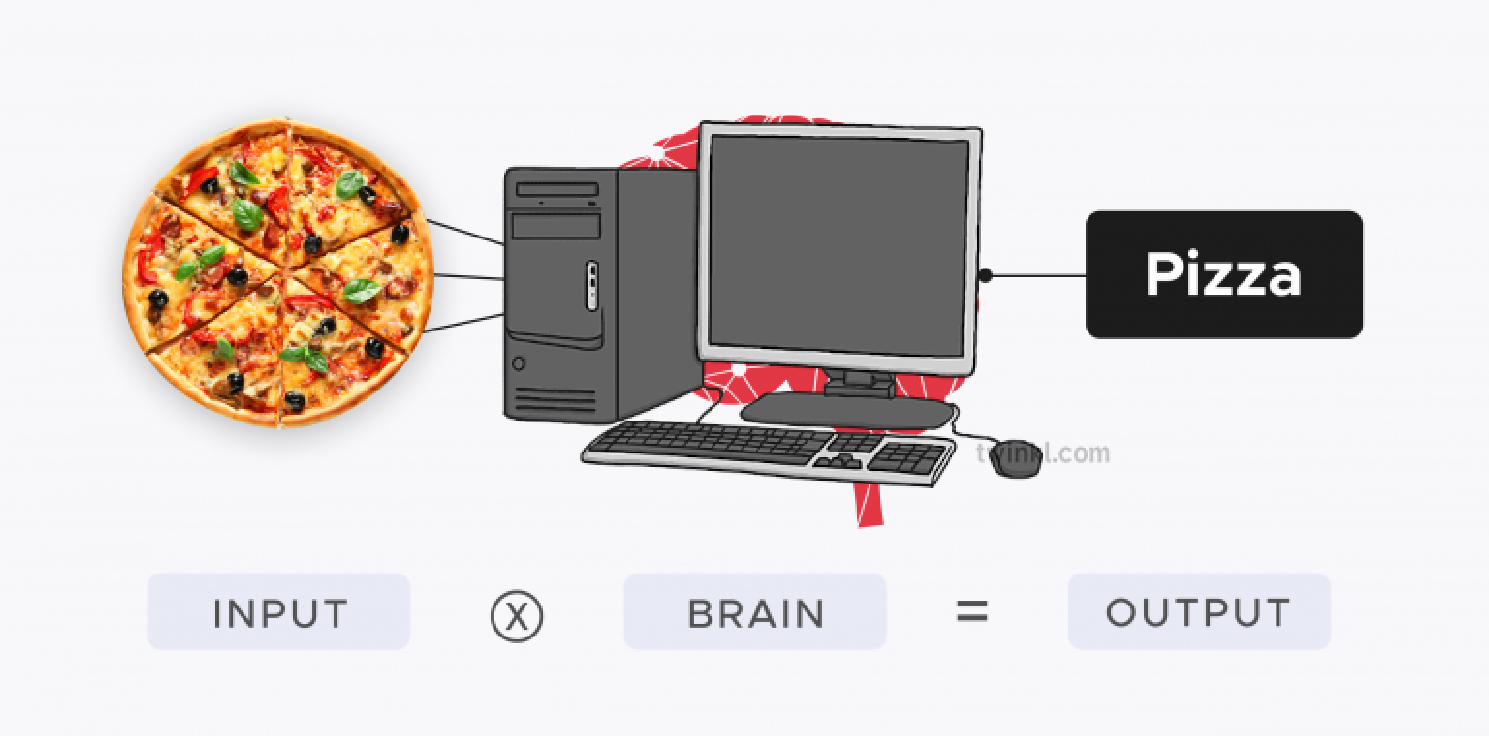


—
Image from zomato

...approximated...

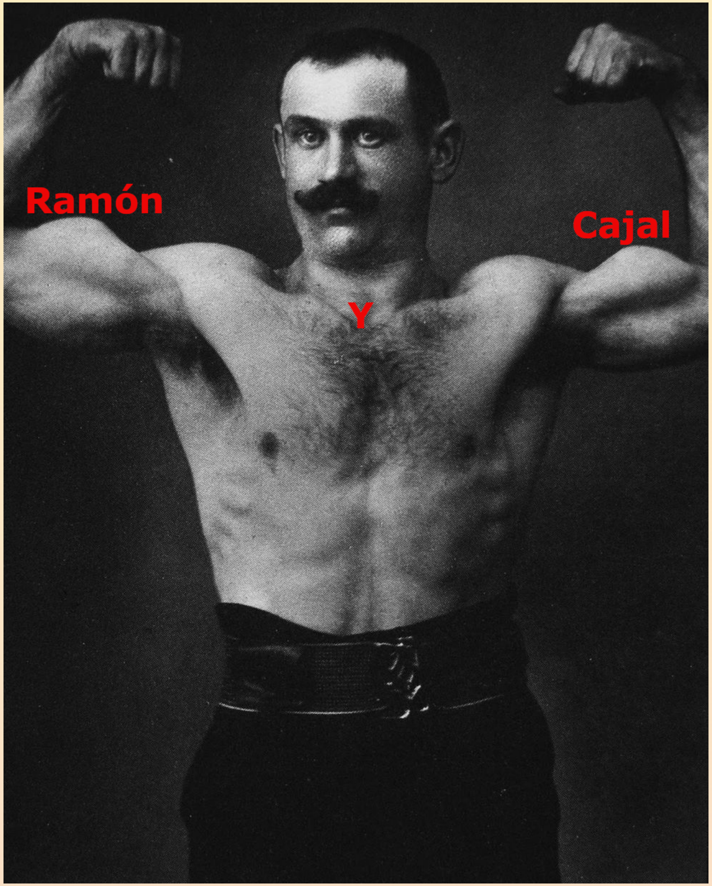
—
Image from zomato, elaborated with image from pixabay

...using computers



—
Image from zomato, elaborated with image from twinkl

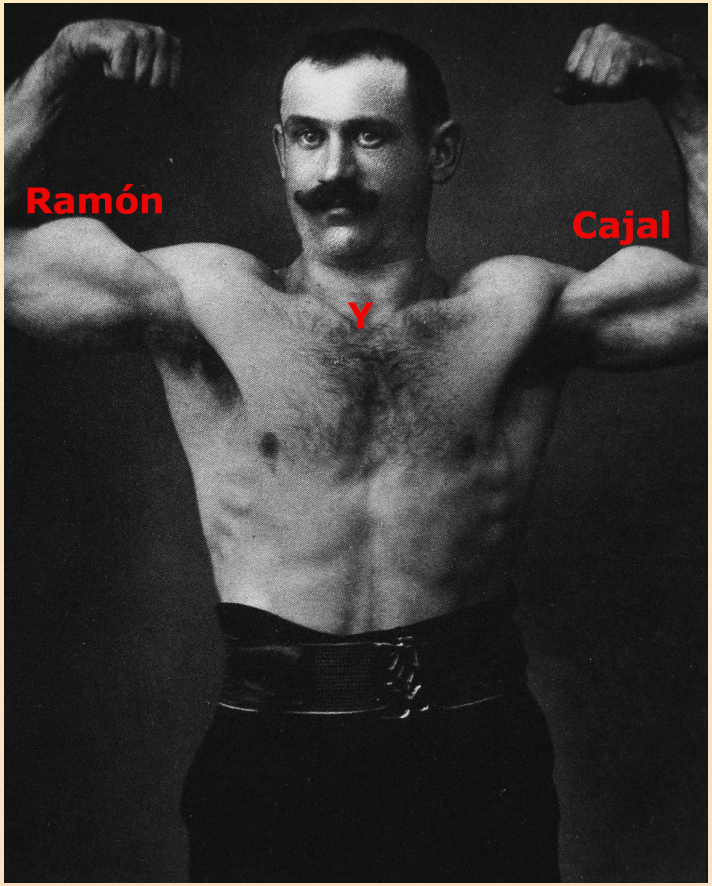
Santiago Ramón y Cajal



- "The Spanish father of culturism"

—
Images from menshealth and The Nobel Prize

Santiago Ramón y Cajal

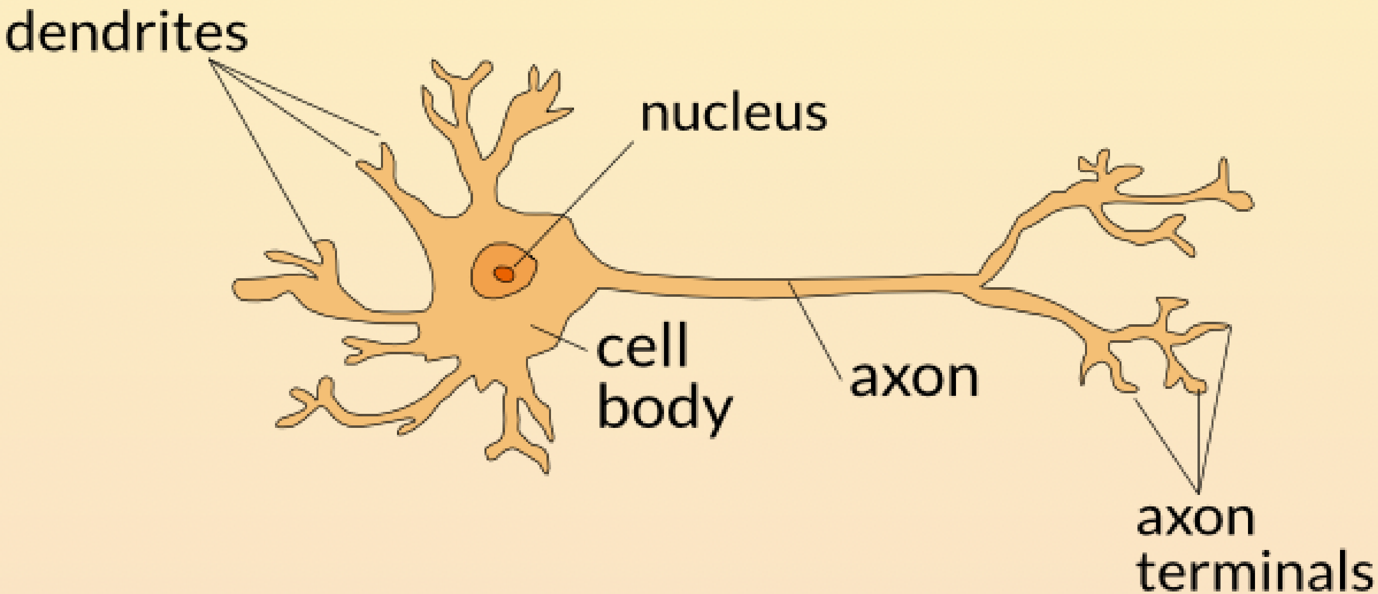


- "The Spanish father of culturism"
- The 1906 Nobel Prize in Medicine

—

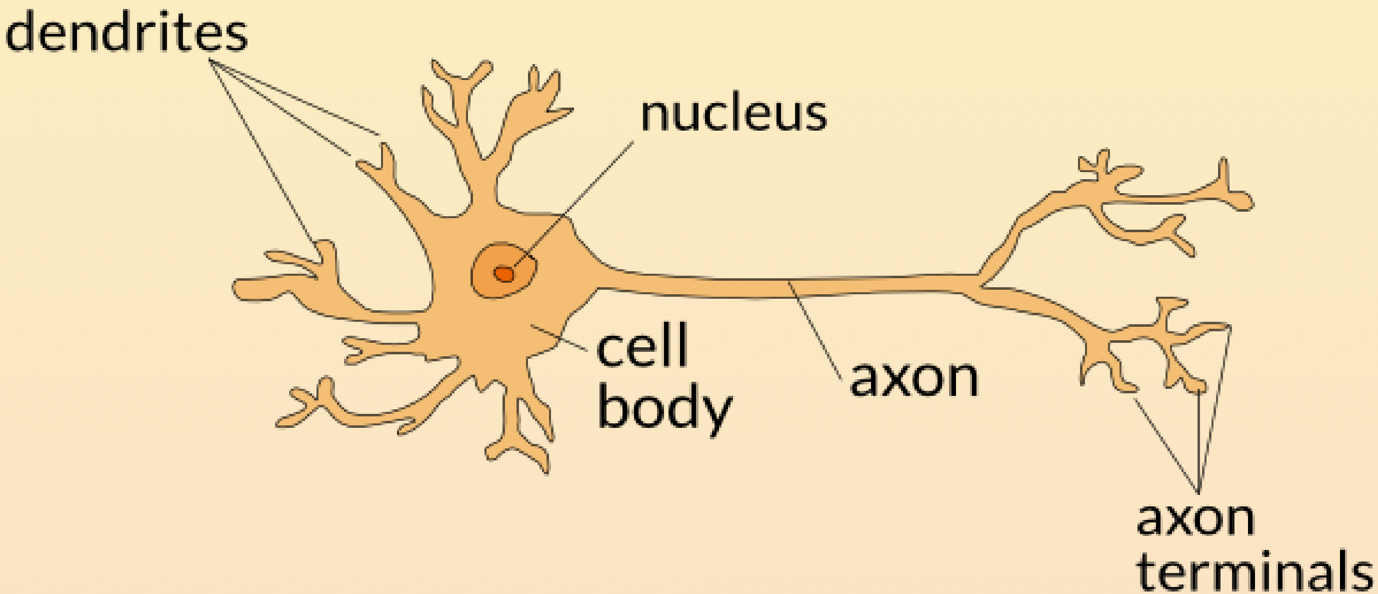
Images from menshealth and The Nobel Prize

Real neurons



—
Image from https://appliedgo.net/perceptron/

Real neurons



$$I = C\frac{dV}{dt} + G_{Na}m^3h(V - V_{Na}) + G_K n^4(V - V_K) + G_L(V - V_L)$$

—
Image from https://appliedgo.net/perceptron/

Computationally heavy

—
Animation from giphy.com

Simplified Neurons

Bulletin of Mathematical Biology Vol. 52, No. 1/2, pp. 99–115, 1990.
Printed in Great Britain.

0092–8240/90$3.00 + 0.00
Pergamon Press plc
Society for Mathematical Biology

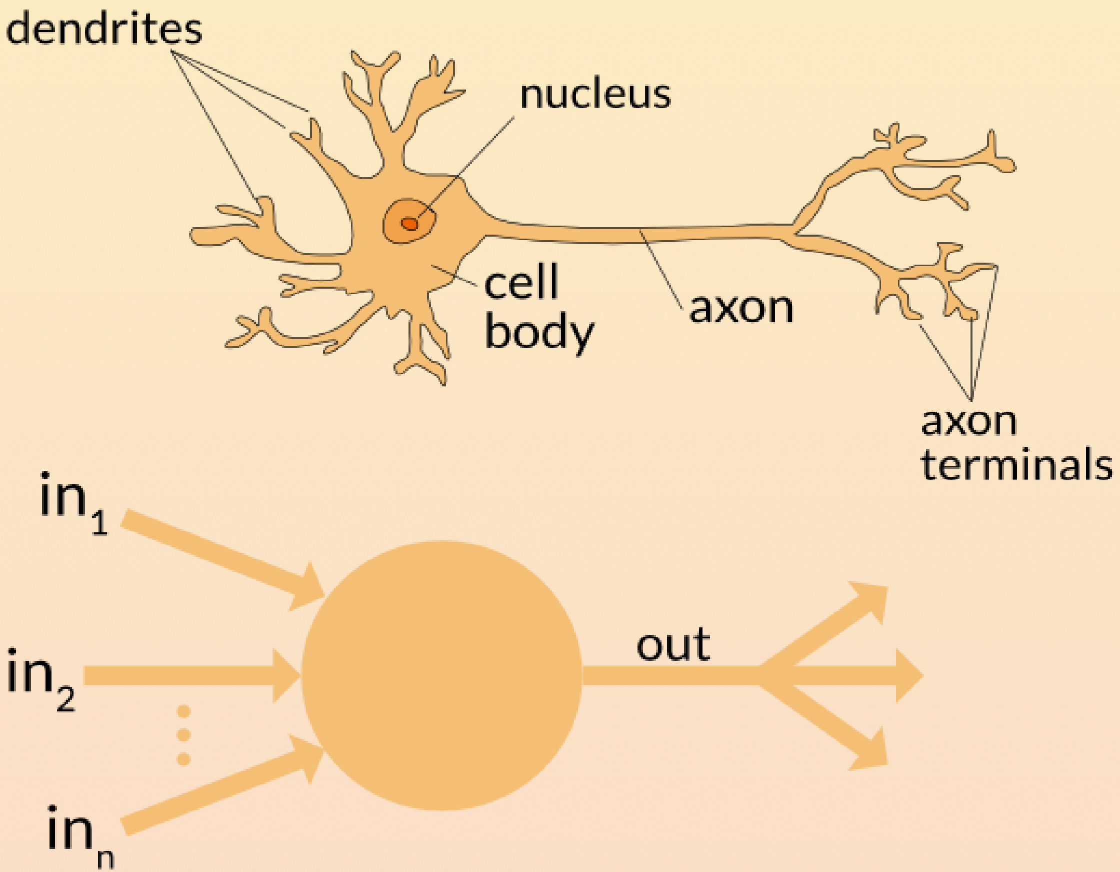
A LOGICAL CALCULUS OF THE IDEAS IMMANENT IN NERVOUS ACTIVITY*

■ WARREN S. MCCULLOCH AND WALTER PITTS
University of Illinois, College of Medicine,
Department of Psychiatry at the Illinois Neuropsychiatric Institute,
University of Chicago, Chicago, U.S.A.

Because of the "all-or-none" character of nervous activity, neural events and the relations among them can be treated by means of propositional logic. It is found that the behavior of every net can be described in these terms, with the addition of more complicated logical means for nets containing circles; and that for any logical expression satisfying certain conditions, one can find a net behaving in the fashion it describes. It is shown that many particular choices among possible neurophysiological assumptions are equivalent, in the sense that for every net behaving under one assumption, there exists another net which behaves under the other and gives the same results, although perhaps not in the same time. Various applications of the calculus are discussed.

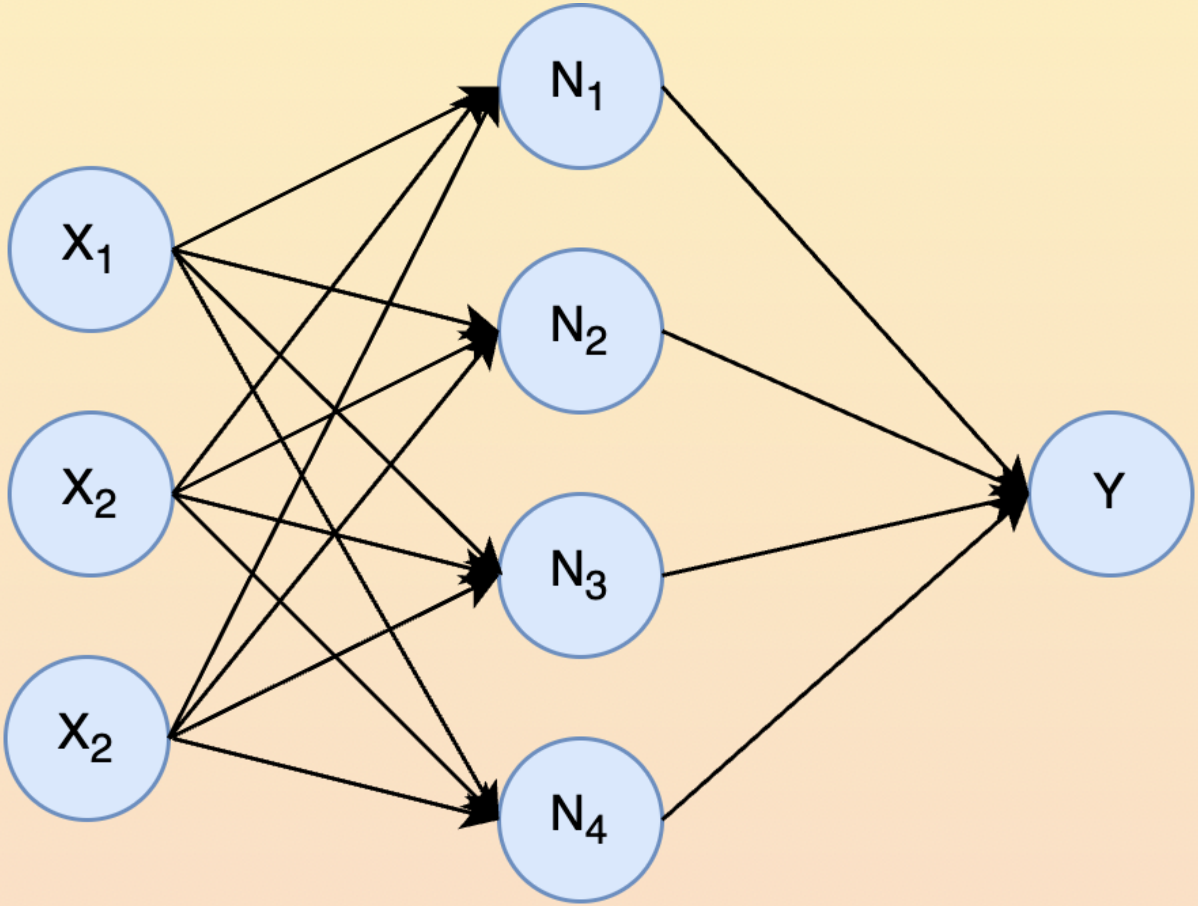
Perceptrons

$$y = f\left(b_i + \sum w_i x_i\right)$$



—
Image from https://appliedgo.net/perceptron/

Artificial Neural Networks



—
Illustration (c) P. Vischia, book in preparation

Perceptron step by step

- Linear combination of the inputs

$$\sum_j w_j x_j$$

- Activation function (imitates the activation of real neurons, where a voltage peak, a spike, is generated when the injected potential passes a threshold)

$$g\left(w_0 + \sum_j w_j x_j\right)$$

- Bias term, an order-zero term in the inputs (translation)

$$\hat{y} = g\left(w_0 + \sum_j w_j x_j\right)$$

- In matrix form, a row-column product

$$\hat{y} = g\left(w_0 + \mathbf{X}^T \mathbf{W}\right)$$

Activation Function

- Originally, the activation function was linear

$$g(z) = 1 \text{ if } z = w_0 + \sum w_i x_i >= 0$$
$$g(z) = -1 \text{ if } z = w_0 + \sum w_i x_i < 0$$

- Activation function is the only chance of estimating nonlinear functions
- Otherwise, a neural network would be just a fancier linear regression model

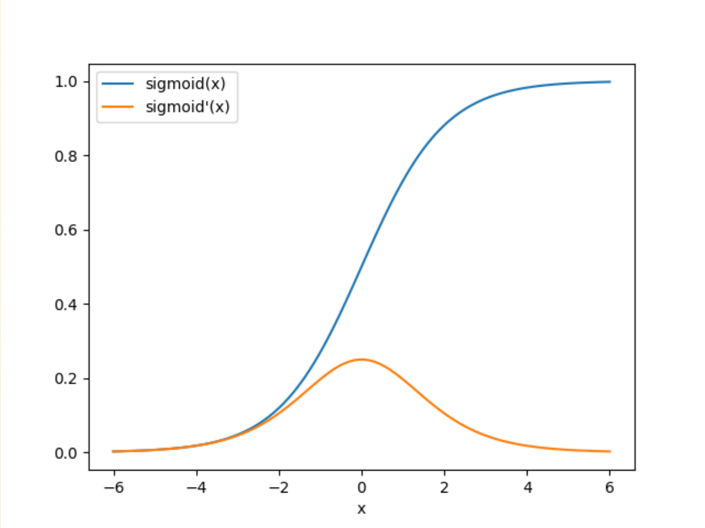
$$\hat{y} = g(1 + \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}^T \begin{bmatrix} 3 \\ 2 \end{bmatrix}) = g(1 + 3x_1 + 2x_2)$$

Popular activation functions

Sigmoid

$$g(z) = \frac{1}{1+e^{-z}}$$

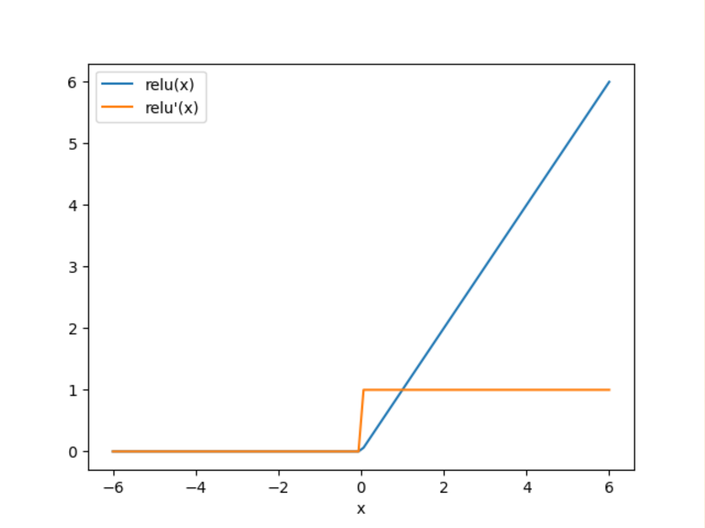
$$g'(z) = g(z)\Big(1 - g(z)\Big)$$



Rectified Linear Unit (ReLU)

$$g(z) = max(0, z)$$

$g'(z) = 1$ if $z > 0, 0$ otherwise



—

Illustrations (c) P. Vischia, book in preparation

Multidimensional outputs

- $\hat{y}_1$, $\hat{y}_2$, each one with the same formula as a single neuron$\rightarrow$ just with an additional index

$$\hat{y}_i = g(w_{0,i} + \sum_j w_{j,i} x_j)$$

Neural network with one internal layer

- Between input and hidden layer: $\mathbf{W}^{(1)}$
- Between hidden layer and output layer: $\mathbf{W}^{(2)}$
- Output of the hidden layer neurons:

$$z_i = g(w_{0,i}^{(1)} + \sum_j w_{j,i}^{(1)} x_j)$$

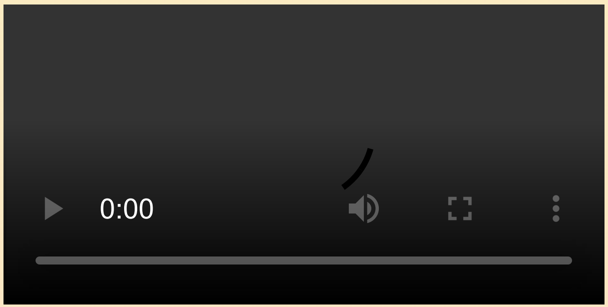
- Output of the network

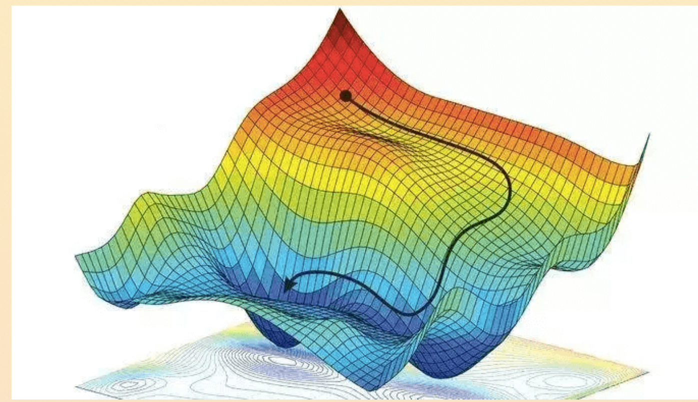
$$\hat{y}_i = g(w_{0,i}^{(2)} + \sum_j w_{j,i}^{(2)} z_j)$$

- The generalization to multiple outputs $\hat{y}_i$ is also trivial

Gradient Descent

- Search for the "minimum error" as a function of the values of the training paramters (weights)

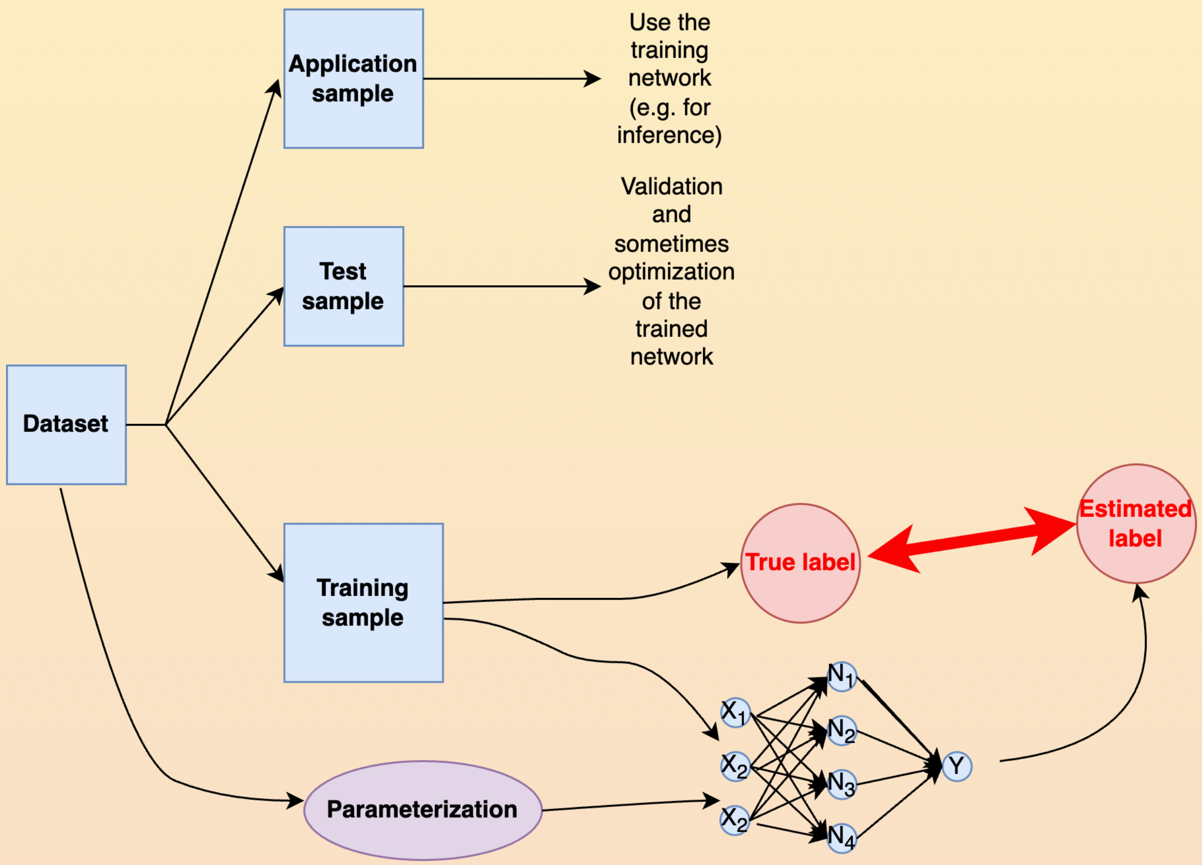




—

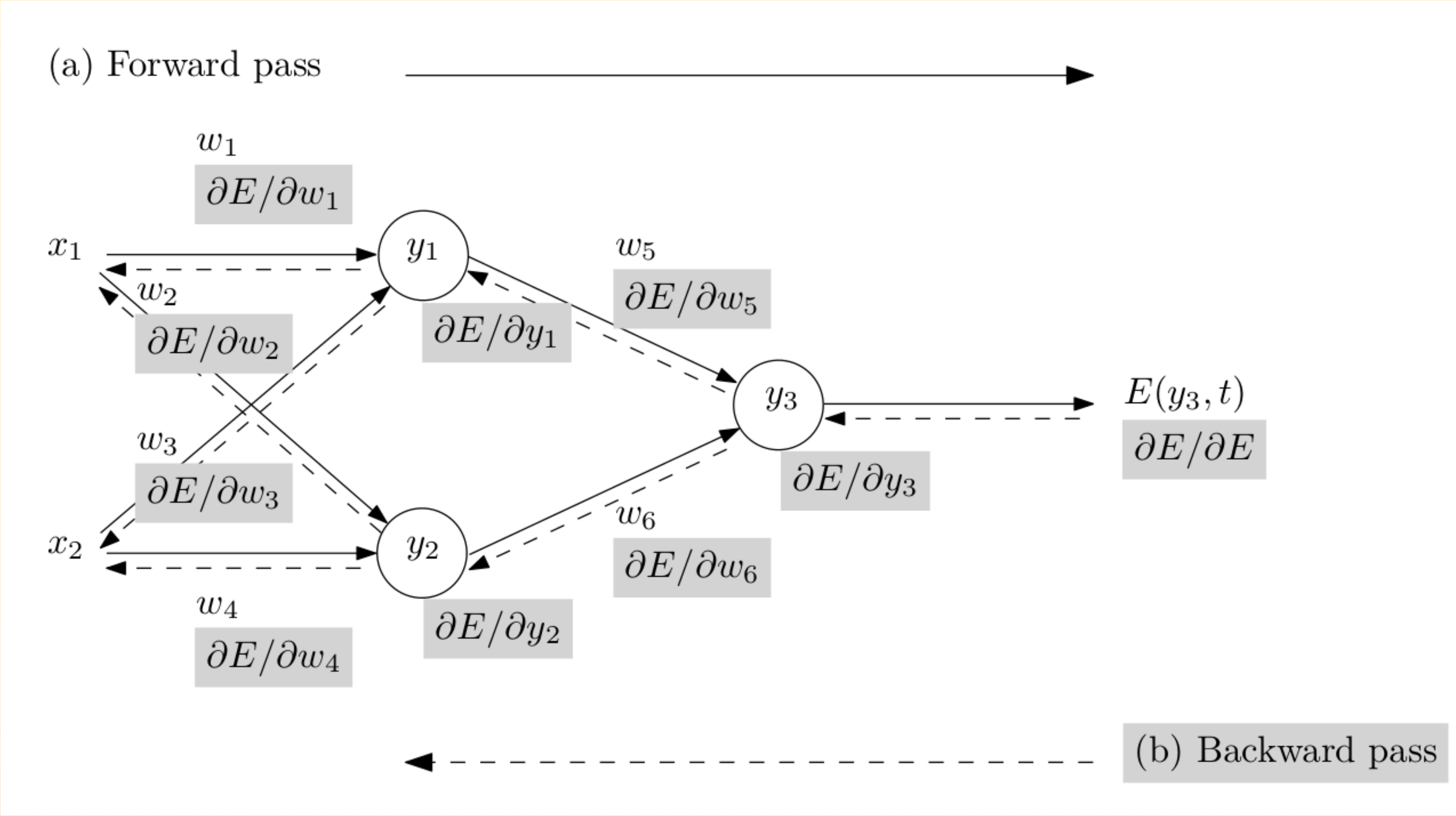
Top illustration from easyai.tech, bottom one from the MODE White Paper

Learning



—
Illustration (c) P. Vischia, book in preparation

Backpropagation



—

Image from Güneş Baydin et al, JMLR 18 (2018) 1--43

Backpropagation

- Empirical Loss Function
- Cost function
- Empirical risk

$$\mathbf{J}(\mathbf{W}) = \frac{1}{n}\sum_{i=1}^{n}\mathcal{L}(f(x^{(i)};\mathbf{W}), y^{*(i)})$$

- Minimized by:

$$\mathbf{W}^{0} = argmin_{\mathcal{W}}\mathbf{J}(\mathbf{W}) = argmin_{\mathcal{W}}\mathbf{J}(\mathbf{W}) = \frac{1}{n}\sum_{i=1}^{n}\mathcal{L}(f(x^{(i)};\mathbf{W}), y^{*(i)})$$

Loss function comes from inference

- Decision-theoretic approach (C.P. Robert, "The Bayesian Choice")
  - $\mathcal{X}$: observation space
  - Θ: parameter space
  - $\mathcal{D}$: decision (action) space
- Statistical inference take a decision $d \in \mathcal{D}$ related to parameter $\theta \in \Theta$ based on observation $x \in \mathcal{X}$, under $f(x|\theta)$
  - Typically, d consists in estimating $h(\theta)$ accurately

$$U(\theta, d) = \mathbb{E}_{\theta,d}\Big[U(r)\Big]$$

Loss function comes from inference

- Loss function: $L(\theta, d) = -U(\theta, d)$
  - Represents intuitively the loss or error in which you incur when you make a bad decision (a bad estimation of the target function)
  - Lower bound at 0: avoids "infinite utility" paradoxes (St. Petersburg paradox, martingale-based stragegies)
- Generally impossible to uniformly minimize in d the loss for θ unknown
  - Need for a practical prescription to use the loss function as a comparison criterion in practice

Frequentist loss, Bayesian loss

- Frequentist loss (risk) is integrated (averaged) on $\mathcal{X}$: $R(\theta, \delta) = \mathbb{E}_\theta\left[L(\theta, \delta(x))\right]$
  - $\delta(\cdot)$ is an \textbf{estimator} of θ (e.g. MLE)
  - Compare estimators, find the best estimator based on long-run performance for all values of unknown θ
  - Issues: based on long run performance (not optimal for x_{obs}); repeatability of the experiment; no total ordering on the set of estimators
- Bayesian loss: is integrated on Θ: $\rho(\pi, d|x) = \mathbb{E}^\pi\left[L(\theta, d)|x\right]$
  - π is the prior distribution
  - Posterior expected loss averages the error over the posterior distribution of θ conditional on x_{obs}
  - Can use the conditionality because x_{obs} is known!
  - Can also integrate the frequentist risk; integrated risk $r(\pi, \delta) = \mathbb{E}^\pi\left[R(\theta, \delta)\right]$ averaged over θ according to π (total ordering)

ANNs and Bayesian networks

- Standard ANN training essentially is a frequentist MLE
  - NN weights: true, unknown values
  - Data: random variable
- Bayesian networks treat weights ω as random (latent) variables, and condition on the observed data
  - Obtain $p(\omega|data)$ starting from prior belief $\pi(\omega)$ and likelihood $p(data|\omega)$
  - Predictions obtained as expectation values, $E_p[f] = \int f(\omega)p(\omega|data)d\omega$, averaging f weighting by the posterior
  - Marginalization leads to essentially learning the generative model (the pdfs), leading to interpretability

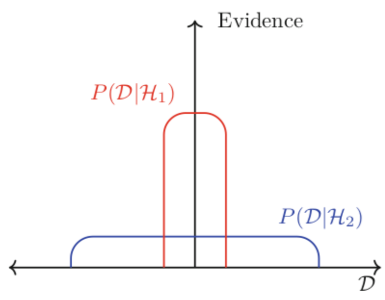


Fig. 3.4 Graphical illustration of how the evidence plays a role in investigating different model hypotheses. The simple model $\mathcal{H}_1$ is able to predict a small range of data with greater strength, while the more complex model $\mathcal{H}_2$ is able to represent a larger range of data, though with lower probability. Adapted from [45, 46]

—

Image from 10.1007/978-3-030-42553-1

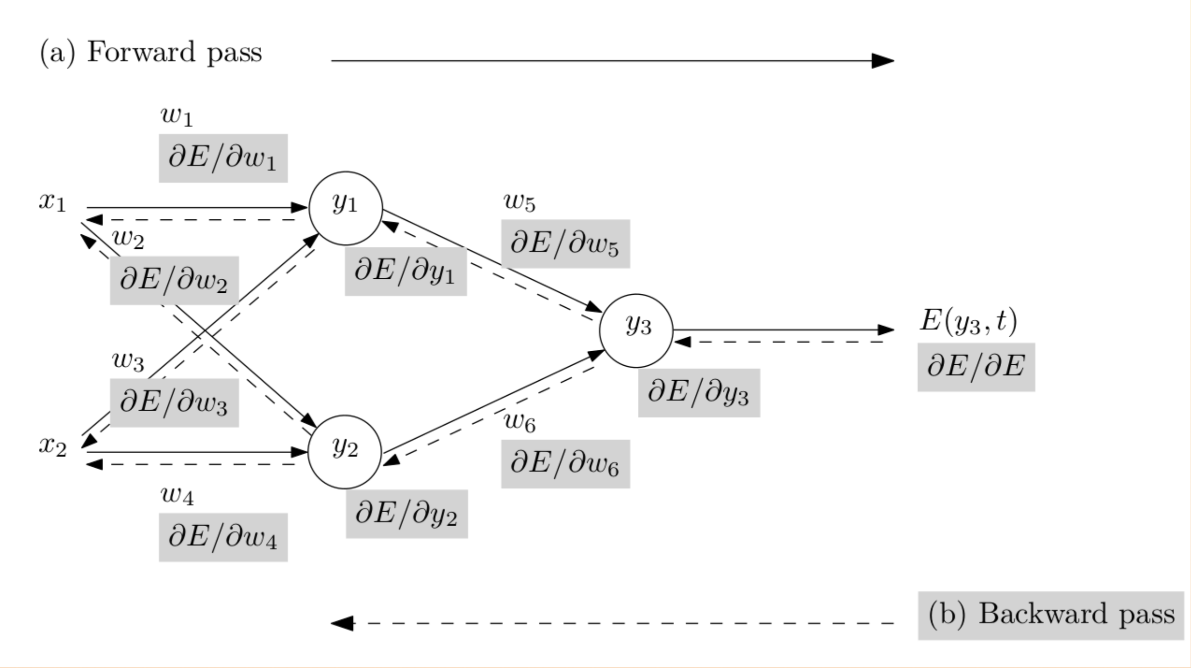
Training loop

- Initialize the weights (for instance $w \sim Gaus(0, \sigma^2)$)
- Loop until convergence
  - Compute network output (forward step)
  - Compute gradients $\frac{\partial \mathbf{J}(\mathbf{W})}{\partial \mathbf{W}}$
  - Update the weights according to the learning rate $\mathbf{W} \leftarrow \mathbf{W} + \eta \frac{\partial \mathbf{J}(\mathbf{W})}{\partial \mathbf{W}}$
  - Return weights

Backpropagation

$$\mathbf{J}(\mathbf{W}) = \frac{1}{n}\sum_{i=1}^{n}\mathcal{L}(f(x^{(i)};\mathbf{W}), y^{*(i)}), \qquad \mathbf{W}^0 = argmin_{\mathcal{W}}\mathbf{J}(\mathbf{W})$$

$$\mathbf{W} \leftarrow \mathbf{W} + \eta\frac{\partial \mathbf{J}(\mathbf{W})}{\partial \mathbf{W}}$$



—

Image from Güneş Baydin et al, JMLR 18 (2018) 1--43

Jacobian and Hessian

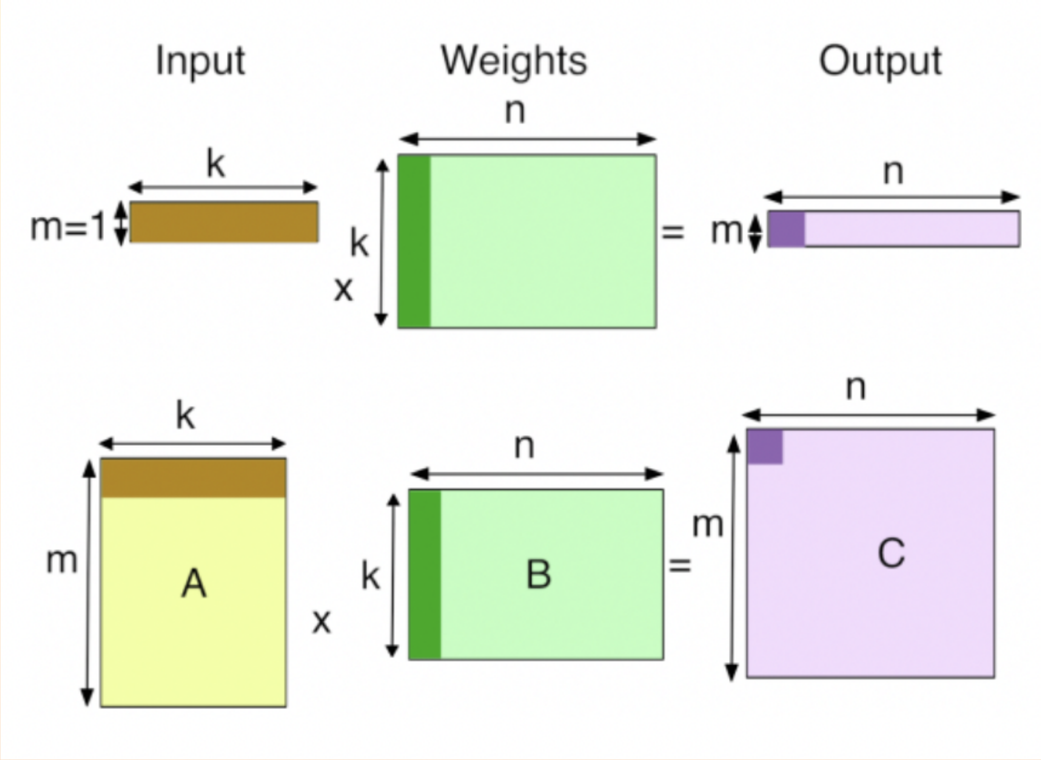
$$\mathbf{J}_{ij} = \frac{\partial f_i}{\partial x_j} \qquad \mathbf{J}(\mathbf{x}) = \begin{bmatrix} \frac{\partial f_1}{\partial x_1}(\mathbf{x}) & \cdots & \frac{\partial f_1}{\partial x_n}(\mathbf{x}) \\ \vdots & & \vdots \\ \frac{\partial f_m}{\partial x_1}(\mathbf{x}) & \ldots & \frac{\partial f_m}{\partial x_n}(\mathbf{x}) \end{bmatrix}$$

$$\mathbf{H}_{ij} = \frac{\partial^2 f}{\partial x_i \partial x_j} \qquad \mathbf{H}(\mathbf{x}) = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2}(\mathbf{x}) & \frac{\partial^2 f}{\partial x_1 \partial x_2}(\mathbf{x}) & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n}(\mathbf{x}) \\ \frac{\partial^2 f}{\partial x_2 \partial x_1}(\mathbf{x}) & \frac{\partial^2 f}{\partial x_2^2}(\mathbf{x}) & \cdots & \frac{\partial^2 f}{\partial x_2 \partial x_n}(\mathbf{x}) \\ \vdots & \vdots & & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1}(\mathbf{x}) & \frac{\partial^2 f}{\partial x_n \partial x_2}(\mathbf{x}) & \cdots & \frac{\partial^2 f}{\partial x_n^2}(\mathbf{x}) \end{bmatrix}$$

- $\mathbf{H}(f(\mathbf{x})) = \mathbf{J}(\nabla f(\mathbf{x}))$
  (describes local curvature)

Matrix multiplication

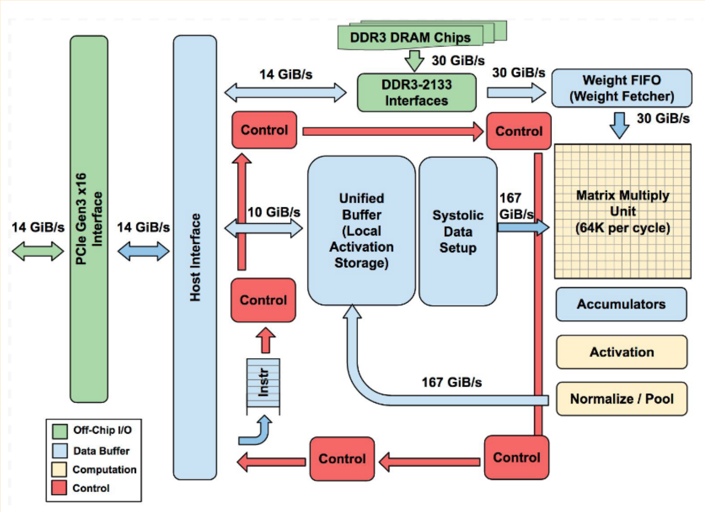
- Neural network weights expressable as matrices
- Generalize matrix calculus to tensors (tensorflow)
- Optimize for efficient tensor calculus (e.g. GPU$\rightarrow$TPU, computational tricks)



—
Figure by Gilles Louppe

Example: Google's TPUs

- Systolic flow
  - Hide four-stage process within the matrix multiplication operation
  - E.g. decoupled access/execution when reading weights
  - Trick flow control into thinking inputs are read and update results at once



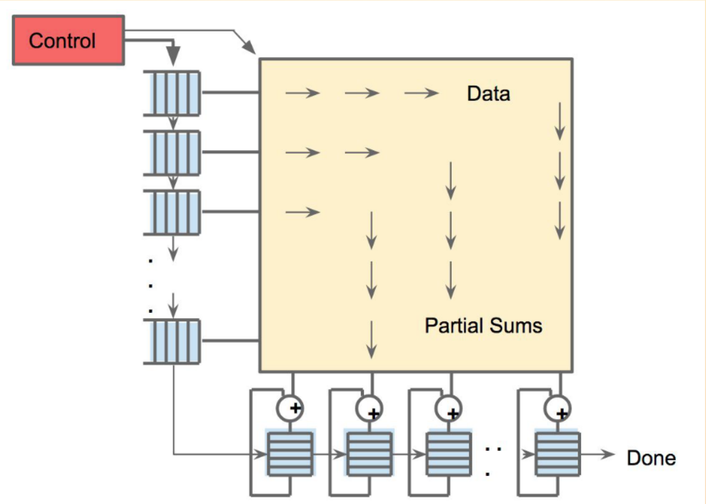
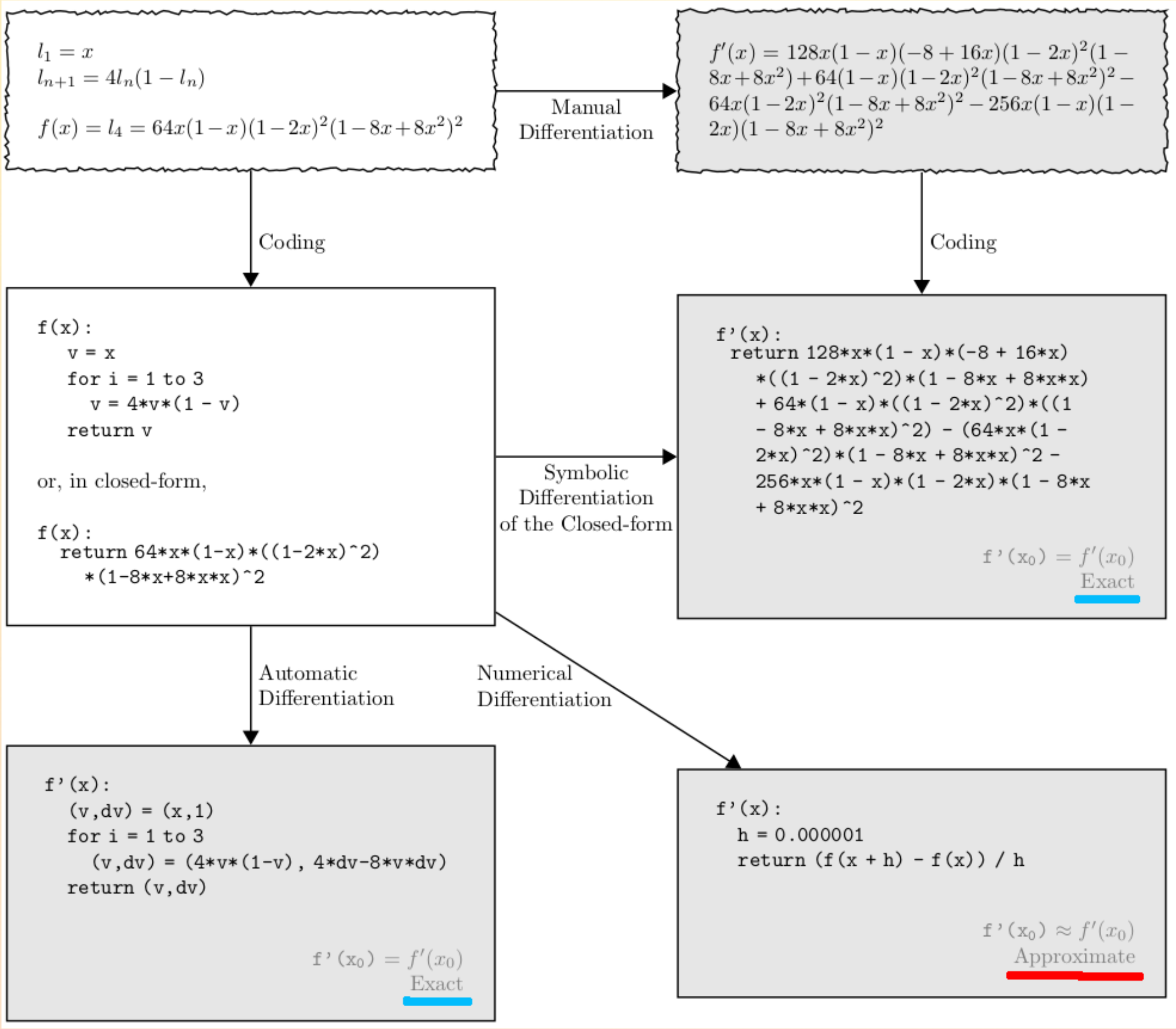


Figure 4. Systolic data flow of the Matrix Multiply Unit. Software has the illusion that each 256B input is read at once, and they instantly update one location of each of 256 accumulator RAMs.

—
Images from 1704.04760

Derive



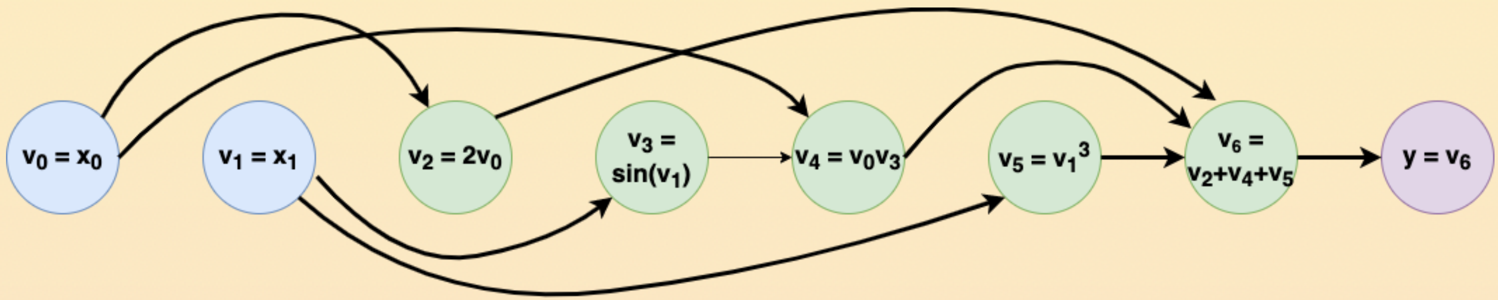
—
Image from Güneş Baydin et al, JMLR 18 (2018) 1--43

Automatic differentiation has many names

- Automatic differentiation
- Algorithmic differentiation
- AD
- Autodiff
- Algodiff
- Autograd

Automatic differentiation

$$z(x, y) = 2x + x\,sin(y) + y^3$$



Forward mode

- To the extreme, $f : \mathbb{R} \to \mathbb{R}^m$
- Evaluates $\left(\frac{\partial f_1}{\partial x}, \ldots, \frac{\partial f_m}{\partial x}\right)$

Reverse mode

- To the extreme, $f : \mathbb{R}^n \to \mathbb{R}$
- Evaluate $\nabla f(\mathbf{x})\left(\frac{\partial f}{\partial x_1}, \ldots, \frac{\partial f}{\partial x_n}\right)$

- Computational cost of calculating $\mathbf{J}_f(\mathbf{x})$ for $f : \mathbb{R}^n \to \mathbb{R}^m$ in $\mathbb{R}^n \times \mathbb{R}^m$

$$\mathcal{O}(n\ \text{time}(f))$$

$$\mathcal{O}(m\ \text{time}(f))$$

Forward and reverse (==backprop) modes

Primal: independent to dependent

Adjoint (derivatives): dependent to independent

$$y(\mathbf{x}) = 2x_0 + x_0\, sin(x_1) + x_1^3$$

***Fwd Primal Trace* Atomic operation**	**Value in $(1,2)$**	***Fwd Tangent Trace (set $\dot{x}_0 = 1$ to compute $\frac{\partial y}{\partial x_0}$)* Atomic operation**	**Value in $(1,2)$**
$v_0 = x_0$ $v_1 = x_1$	1 2	$\dot{v}_0 = \dot{x}_0$ $\dot{v}_1 = \dot{x}_1$	1 0
$v_2 = 2v_0$ $v_3 = sin(v_1)$ $v_4 = v_0 v_3$ $v_5 = v_1^3$ $v_6 = v_2 + v_4 + v_5$	2 0.9093 0.9093 8 10.9093	$\dot{v}_2 = 2\dot{v}_0$ $\dot{v}_3 = \dot{v}_1 cos(v_1)$ $\dot{v}_4 = \dot{v}_0 v_3 + v_0 \dot{v}_3$ $\dot{v}_5 = 3\dot{v}_1 v_1^2$ $\dot{v}_6 = \dot{v}_2 + \dot{v}_4 + \dot{v}_5$	2×1 0×-0.41 $1 \times 0.9093 + 1 \times 0$ $3 \times 0 \times 4$ $2 + 0.9093 + 0$
$y = v_6$	10.9093	$\dot{y} = \dot{v}_6$	2.9093

***Fwd Primal Trace* Atomic operation**	**Value in $(1,2)$**	***Rev Adjoint Trace (set $\bar{y} = 1$ to compute $\frac{\partial v}{\partial y}$)* Atomic operation**	**Value in $(1,2)$**
$v_0 = x_0$ $v_1 = x_1$	1 2	$\bar{x}_0 = \bar{v}_0$ $\bar{x}_1 = \bar{v}_1$	2.9093 11.5839
$v_2 = 2v_0$ $v_3 = sin(v_1)$ $v_4 = v_0 v_3$ $v_5 = v_1^3$ $v_6 = v_2 + v_4 + v_5$	2 0.9093 0.9093 8 10.9093	$\bar{v}_0 = \bar{v}_0 + \bar{v}_2 \partial v_2 / \partial v_0$ $\bar{v}_0 = \bar{v}_4 \partial v_4 / \partial v_0$ $\bar{v}_1 = \bar{v}_1 + \bar{v}_3 \partial v_3 / \partial v_1$ $\bar{v}_1 = \bar{v}_5 \partial v_5 / \partial v_1$ $\bar{v}_2 = \bar{v}_6 \partial v_6 / \partial v_2$ $\bar{v}_3 = \bar{v}_4 \partial v_4 / \partial v_3$ $\bar{v}_4 = \bar{v}_6 \partial v_6 / \partial v_4$ $\bar{v}_5 = \bar{v}_6 \partial v_6 / \partial v_5$	$\bar{v}_0 + \bar{v}_2 \times 2 = 2.9093$ $\bar{v}_4 \times v_3 = 0.9093$ $\bar{v}_1 + \bar{v}_3 \times cos(v_1) = 11.5839$ $\bar{v}_5 \times 3v_1^2 = 12$ $\bar{v}_6 \times 1 = 1$ $\bar{v}_4 \times v_0 = 1$ $\bar{v}_6 \times 1 = 1$ $\bar{v}_6 \times 1 = 1$
$y = v_6$	10.9093	$\bar{v}_6 = \bar{y}$	1

Designed to be simple in software

```
import torch, math
x0 = torch.tensor(1., requires_grad=True)
x1 = torch.tensor(2., requires_grad=True)
p = 2*x0 + x0*torch.sin(x1) + x1**3
print(p)
p.backward()
print(x0.grad, x1.grad)
```

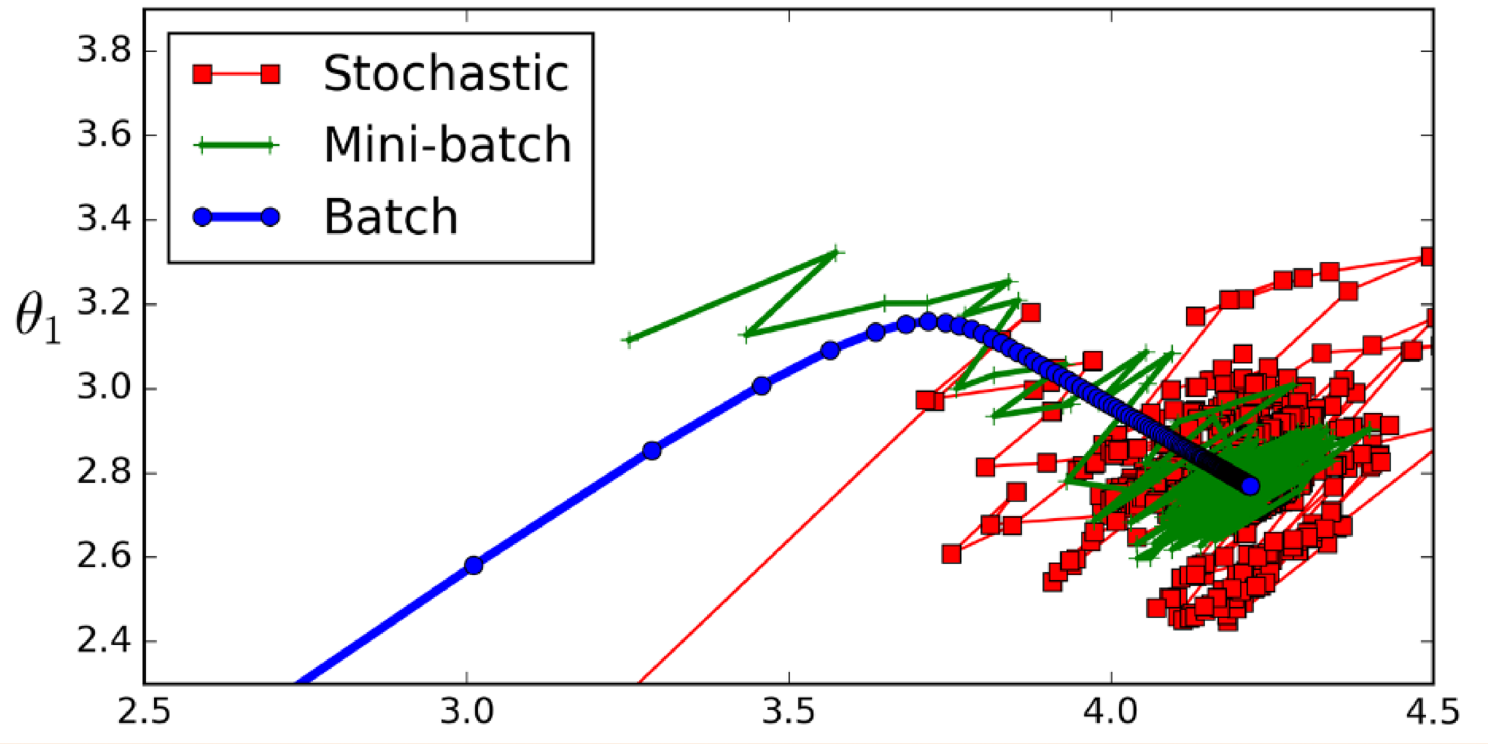
yielding

```
Primal: tensor(10.9093, grad_fn=<AddBackward0>)
Adjoint: tensor(2.9093) tensor(11.5839)
```

- `Torch` (and similar software) will correctly differentiate only when the atomic operations are supported within it
  - Common operations are overloaded (`__mul__` rewritten by `torch._mult_`)
  - Operations from libraries (`math.sin()`) must be replaced by their differentiation-aware equivalents (`torch.sin()`)

Sampling scheme

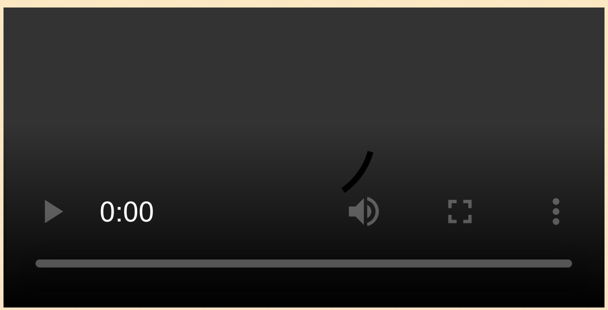
- Batch: compute on the whole training set (for large sets becomes too costly)
- Stochastic: compute on one sample (large noise, difficult to converge)
- Mini-batch: use a relatively small sample of data (tradeoff)



—
Illustration from W. Verbeke

Descent strategies

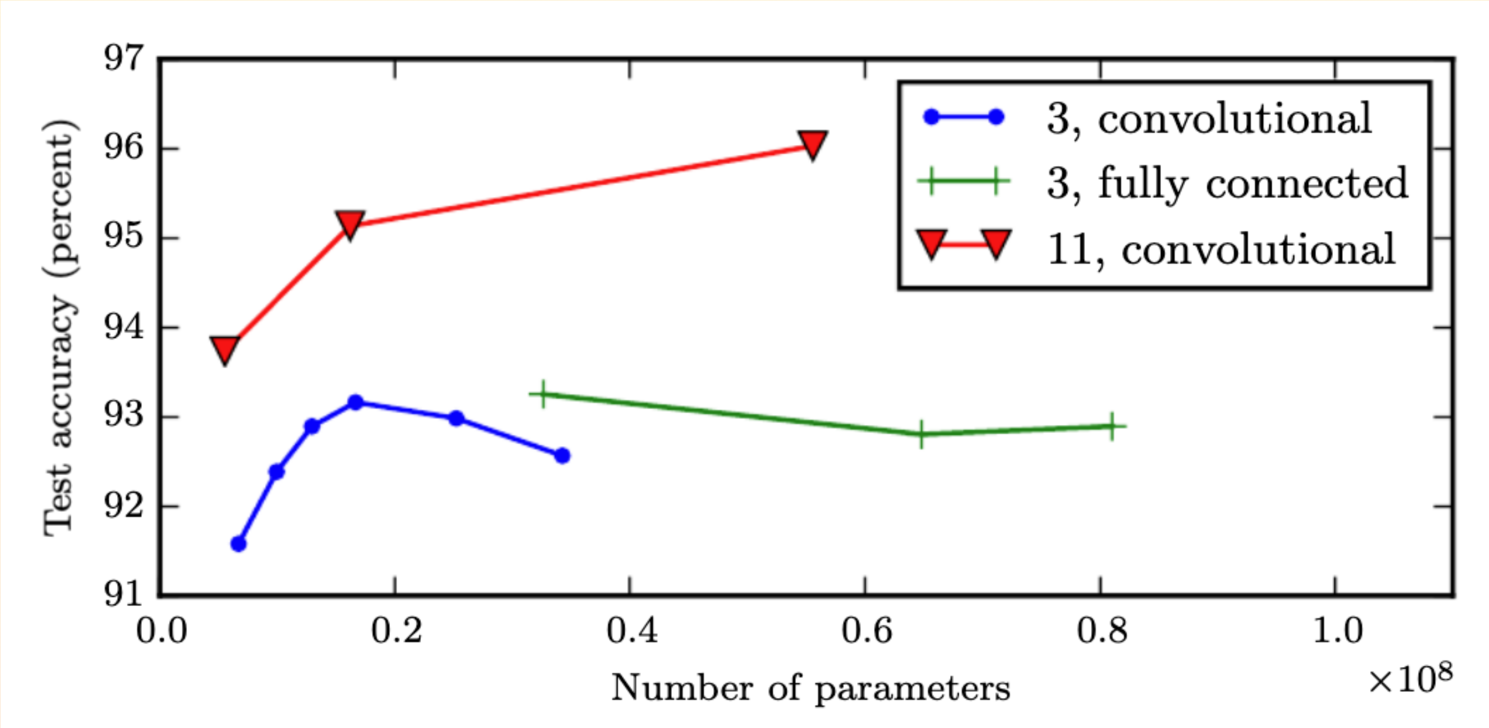
- Mostly nonconvex optimization: very complicated problem, convergence in general not guaranteed
- Nesterov momentum: big jumps followed by correction seem to help!
- Adaptive moments: gradient steps decrease when getting closer to the minimum (avoids overshooting)



—
Top illustration from easyai.tech, bottom one from the MODE White Paper

Number of parameters

- Empirical studies: increasing number of parameters doesn't help beyond a certain point

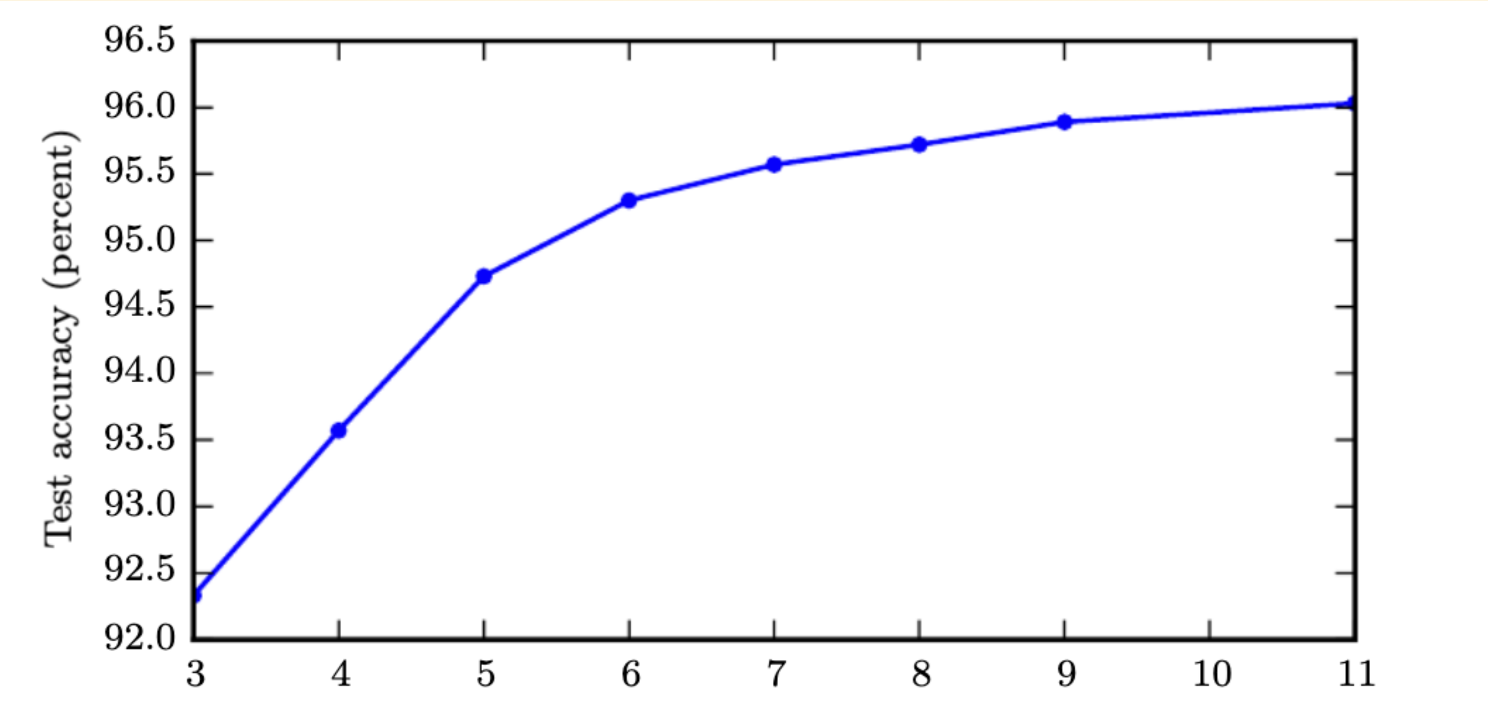


—

Images from Goodfellow, Bengio, Courville, 2016

Depth

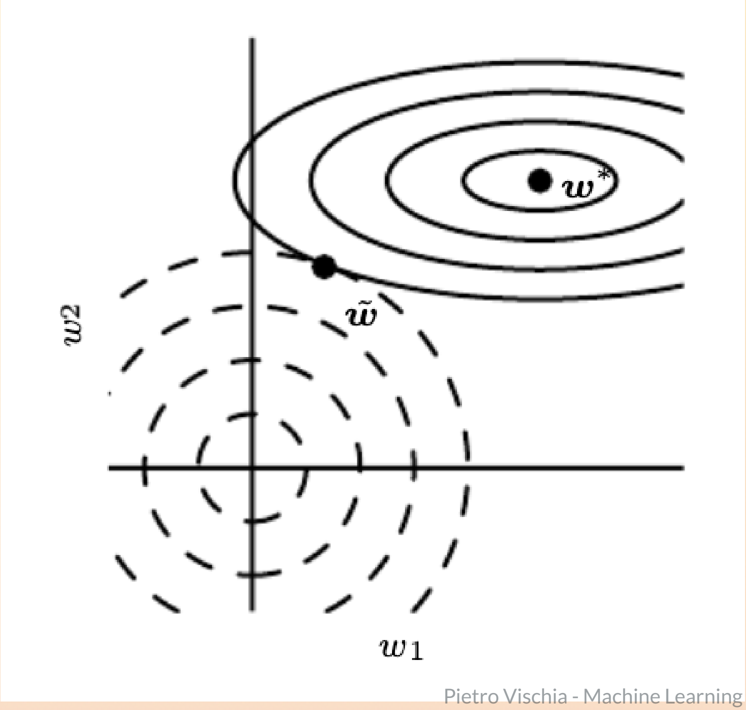
- Empirical studies: increasing depth tends to always result in some improvement



Regularization: weight decay

- Another way of regularizing is via weight decay (L^2 regularization)
    - Tradeoff between good fitting (small MSE) and small norm (smaller slope, or fewer features with large weights)
    - In statistics, "ridge regression", "Tikhonov regularization"

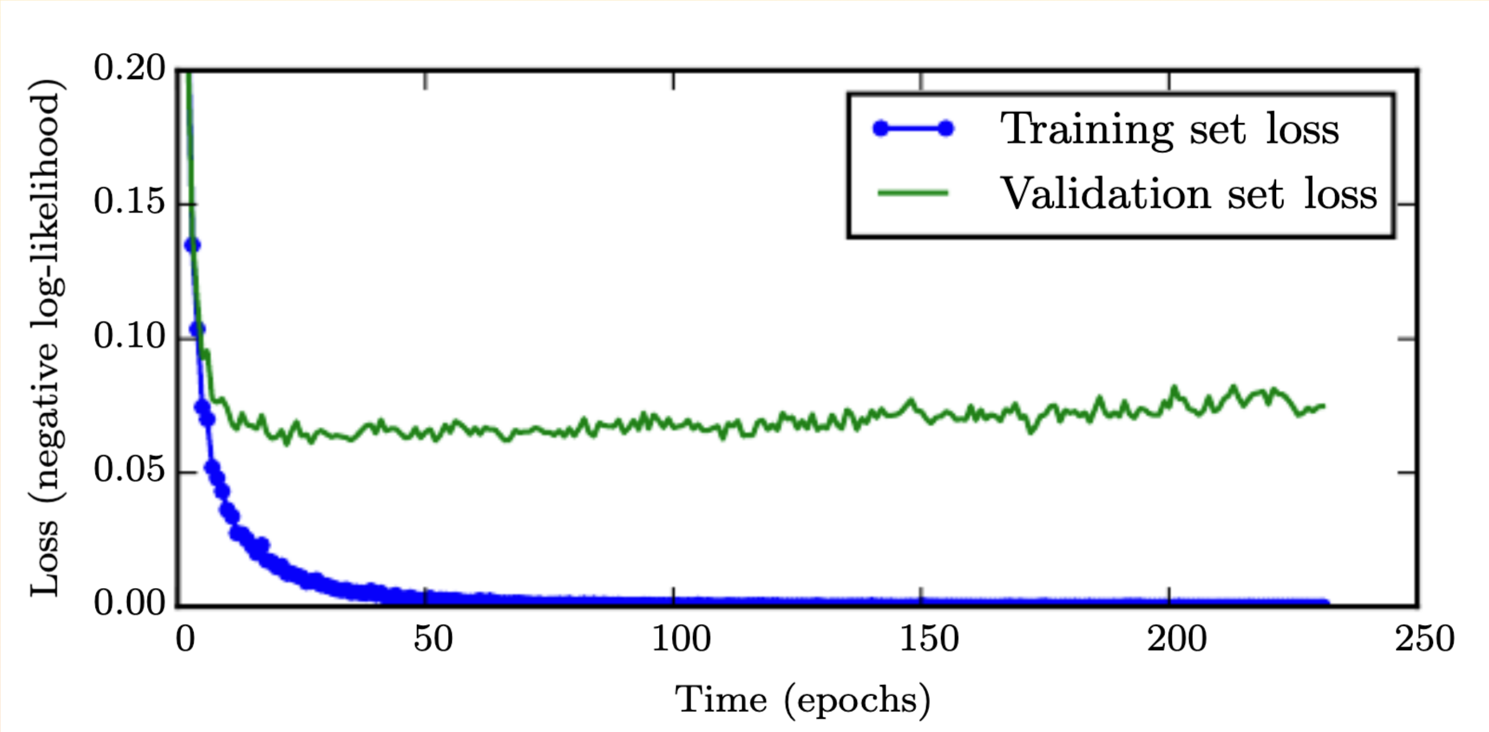
$$J(\mathbf{w}) = MSE_{train} + \lambda \mathbf{w}^T \mathbf{w}$$



—
Images from Goodfellow, Bengio, Courville, 2016

Early stopping...

- Train until the validation set loss starts increasing, and pick the model corresponding to the minimum validation loss

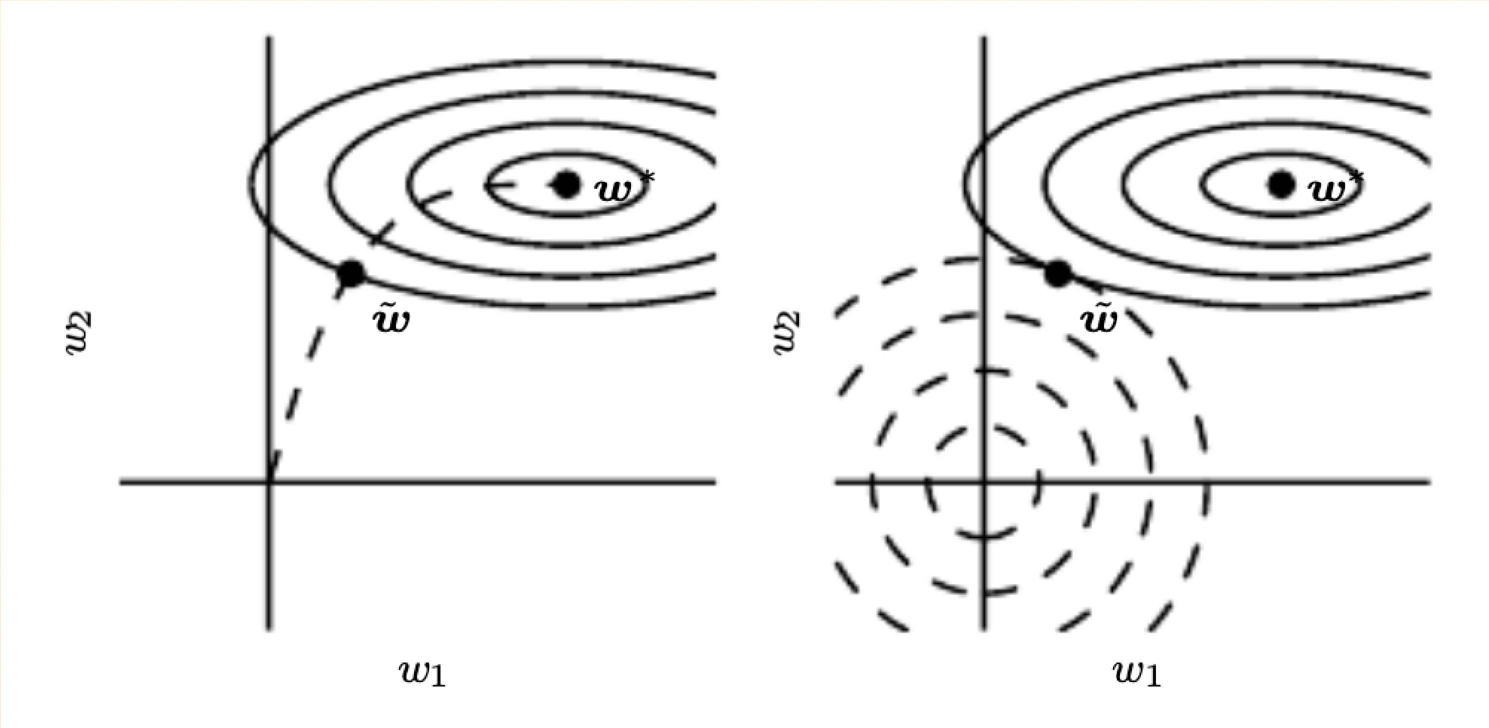


—
Images from Goodfellow, Bengio, Courville, 2016

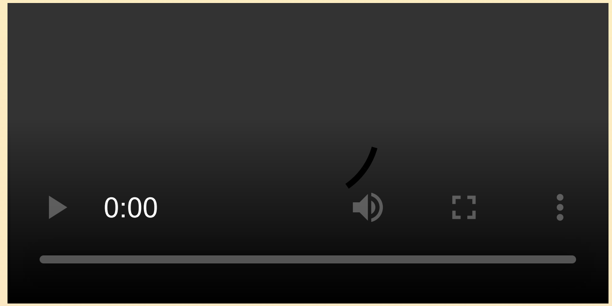
...is a form of regularisation

- Early stopping limits the reachable phase space, and is therefore analogous to L2 regularization (weight decay)

$$J(\mathbf{w}) = MSE_{train} + \lambda \mathbf{w}^T \mathbf{w}$$

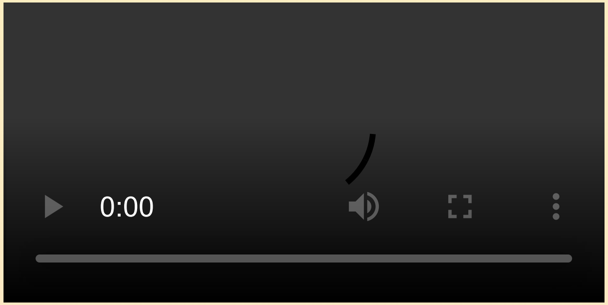


Impressive results



—
Videos from YouTube, autonomous driving

Impressive results



- Busco colaboraciones para aplicaciones médicas de inteligencia artificial

—
Video de YouTube, cancer research

Differentiable Programming

Execute differentiable functions (programs) via automatic differentiation

Yann LeCun
January 5, 2018 ·

OK, Deep Learning has outlived its usefulness as a buzz-phrase.
Deep Learning est mort. Vive Differentiable Programming!

Yeah, Differentiable Programming is little more than a rebranding of the modern collection Deep Learning techniques, the same way Deep Learning was a rebranding of the modern incarnations of neural nets with more than two layers.

But the important point is that people are now building a new kind of software by assembling networks of parameterized functional blocks and by training them from examples using some form of gradient-based optimization.

An increasingly large number of people are defining the networks procedurally in a data-dependent way (with loops and conditionals), allowing them to change dynamically as a function of the input data fed to them. It's really very much like a regular progam, except it's parameterized, automatically differentiated, and trainable/optimizable. Dynamic networks have become increasingly popular (particularly for NLP), thanks to deep learning frameworks that can handle them such as PyTorch and Chainer (note: our old deep learning framework Lush could handle a particular kind of dynamic nets called Graph Transformer Networks, back in 1994. It was needed for text recognition).

People are now actively working on compilers for imperative differentiable programming languages. This is a very exciting avenue for the development of learning-based AI.

Important note: this won't be sufficient to take us to "true" AI. Other concepts will be needed for that, such as what I used to call predictive learning and now decided to call Imputative Learning. More on this later....

1.8K 186 Comments 464 Shares

—
Screenshot of Yann LeCun's facebook post

Many ways of inserting our biases

- Regularization corresponds to inserting our bias into the algorithm
  - "I know that the solution should not wiggle", "I know that the curvature must not be too large"
- Two models A and B performing the same classification task
  - $\hat{y}^{(A)} = f(\mathbf{w}^{(A)}, \mathbf{x})$
  - $\hat{y}^{(B)} = f(\mathbf{w}^{(B)}, \mathbf{x})$
- If the inputs distributions are somehow different, but we know (or want that) the output are related, we can assume that the weights should be similar

$$J(\mathbf{w}) = MSE_{train} + \lambda ||\mathbf{w}^{(A)} - \mathbf{w}^{(B)}||_2^2$$

Parameter sharing

- Simply require parameters are equal
  - If they are equal, you can store only one number in memory (sometimes dramatic memory footprint reduction)

—
Images from goodhousekeeping.com

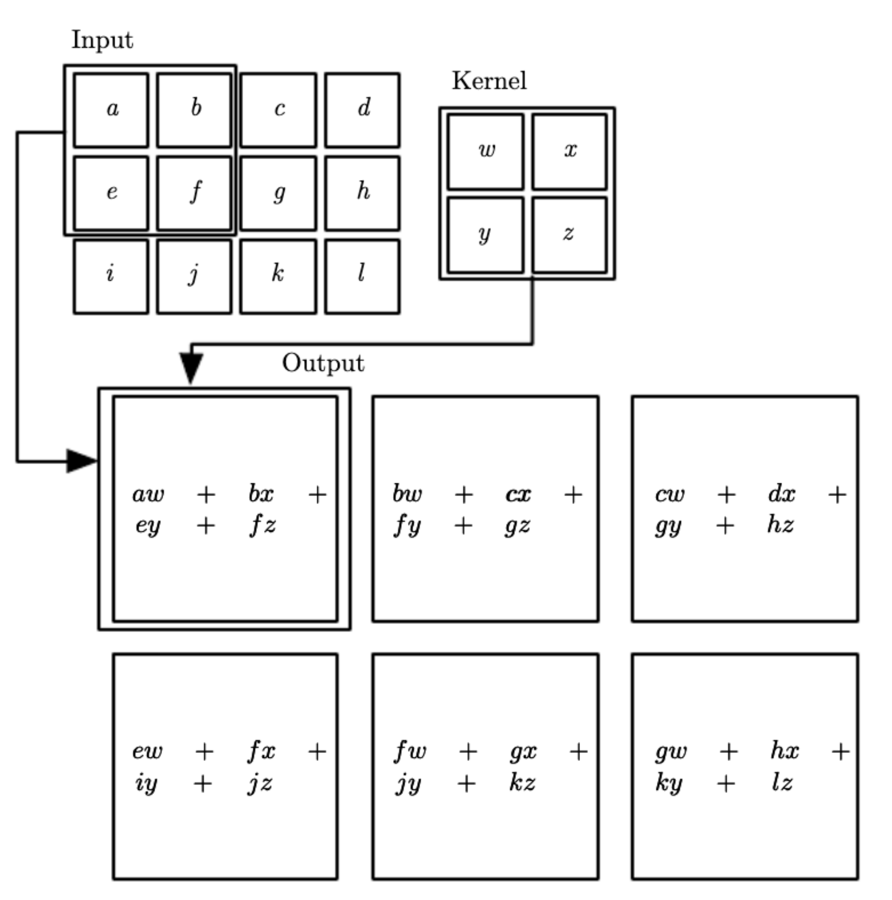
Convolution: a form of averaging

$$s(t) = \int x(a)w(t-a)da$$

- When discretized, integral becomes a sum
  - x input
  - w kernel: specifies how far does the averaging goes
  - s feature map

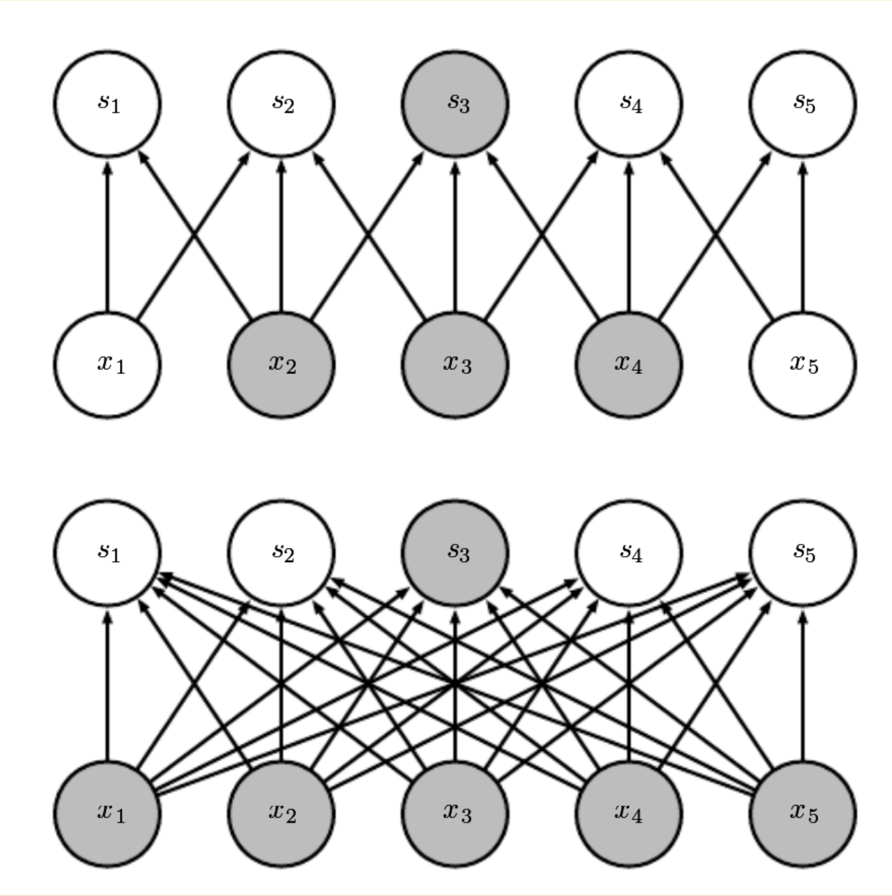
Convolution: a form of averaging

$$S(i,j) = (K * I)(i,j) = \sum_m \sum_n I(i-m, j-n)K(m,n)$$



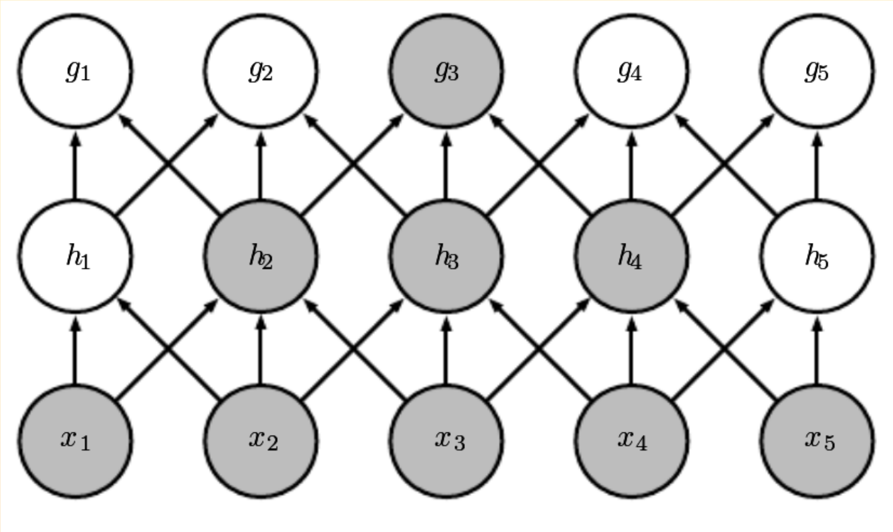
—
Images from Goodfellow, Bengio, Courville, 2016

Receptive field



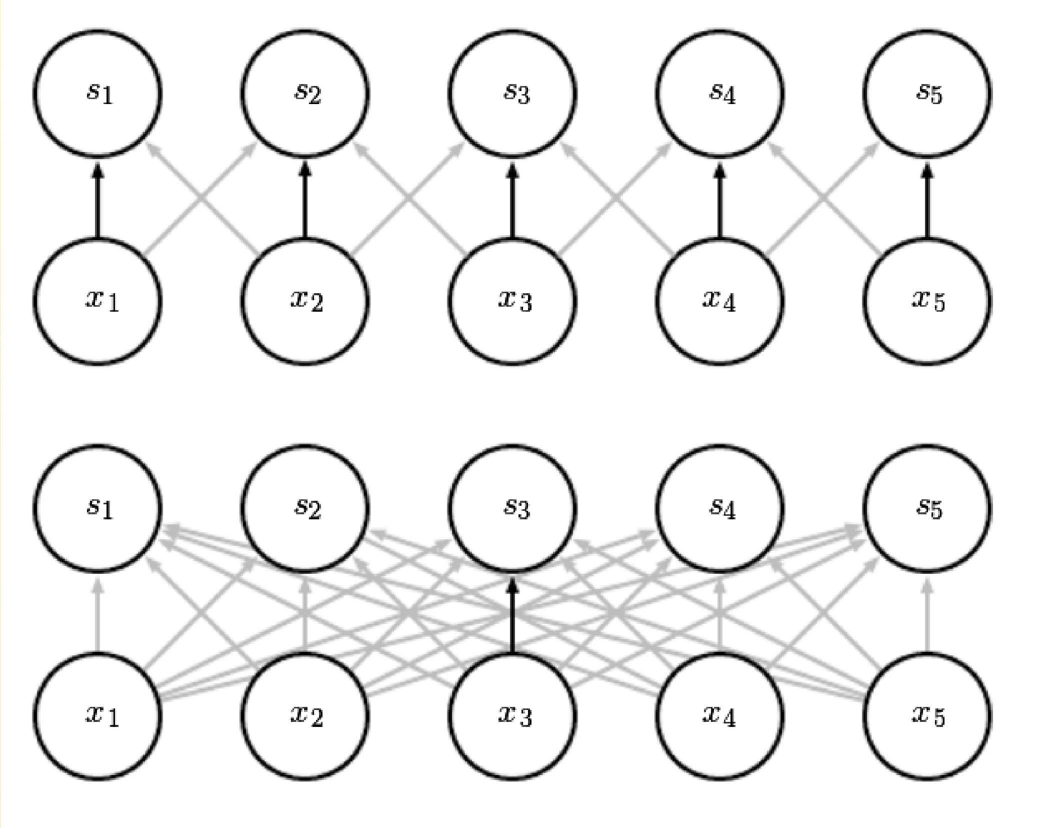
—
Images from Goodfellow, Bengio, Courville, 2016

Receptive field: deeper = larger



—
Images from Goodfellow, Bengio, Courville, 2016

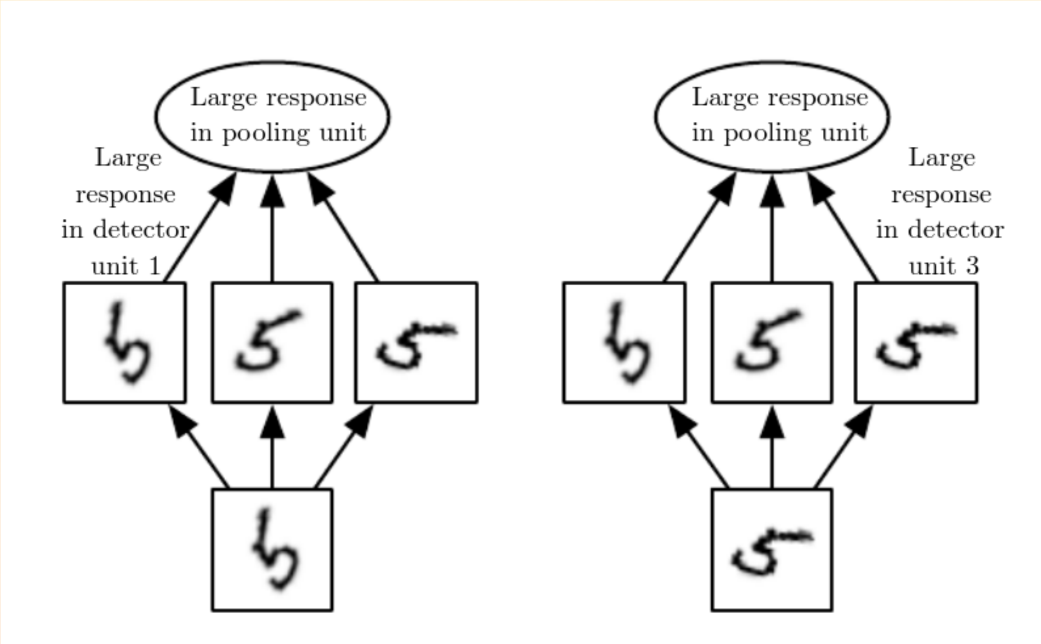
Parameter sharing



—
Images from Goodfellow, Bengio, Courville, 2016

Convolutional network

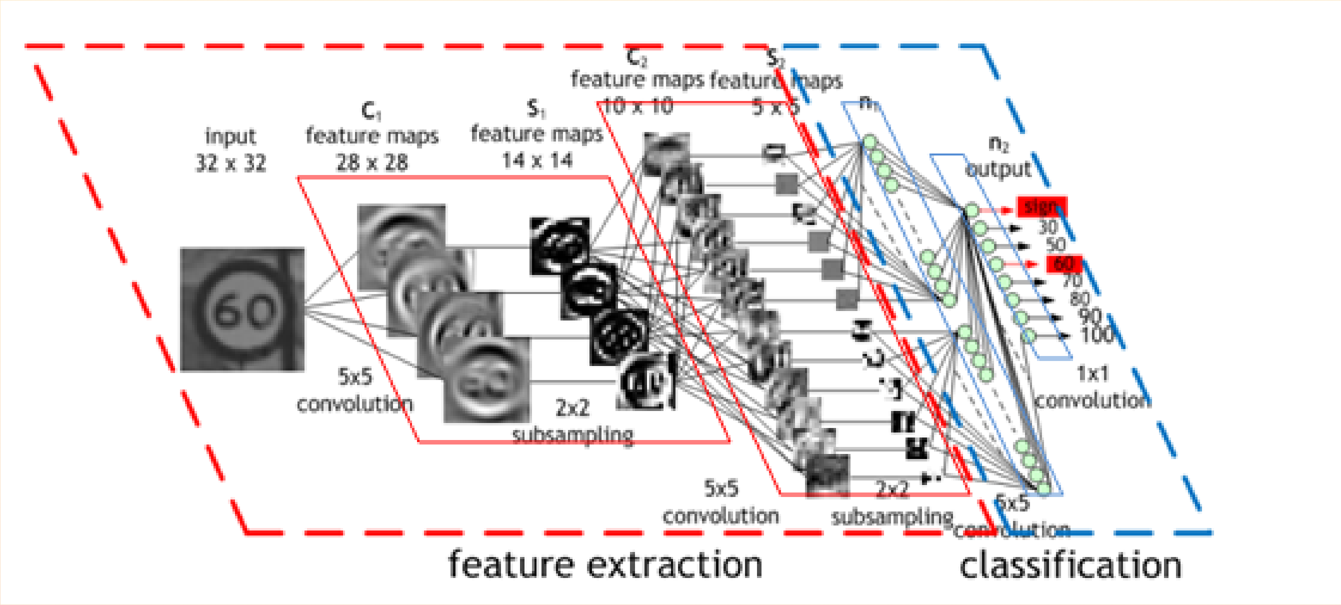
- Convolution $\longrightarrow$ nonlinear activation $\longrightarrow$ pooling
- Pooling: replace output at a location with a summary statistic
    - e.g., max pooling = report the maximum output in a neighbourhood
    - Helps with invariance for translations



—

Images from Goodfellow, Bengio, Courville, 2016

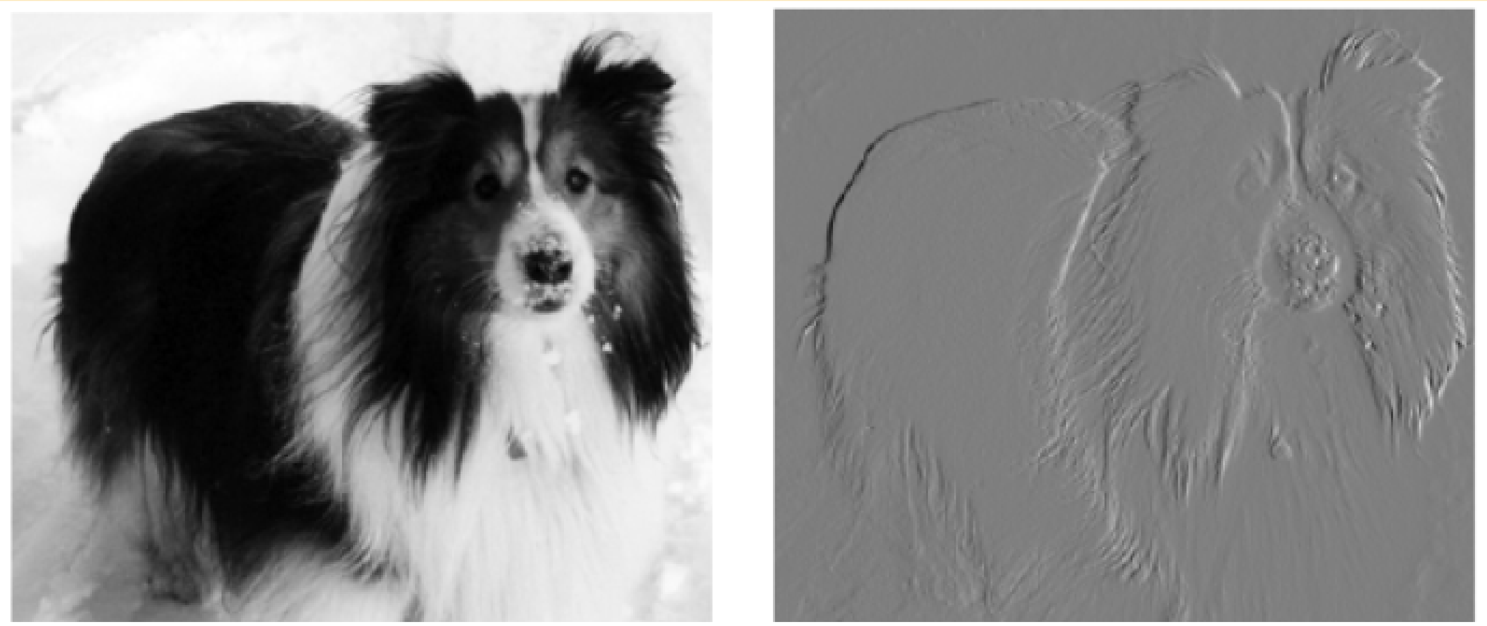
Convolutional networks



—

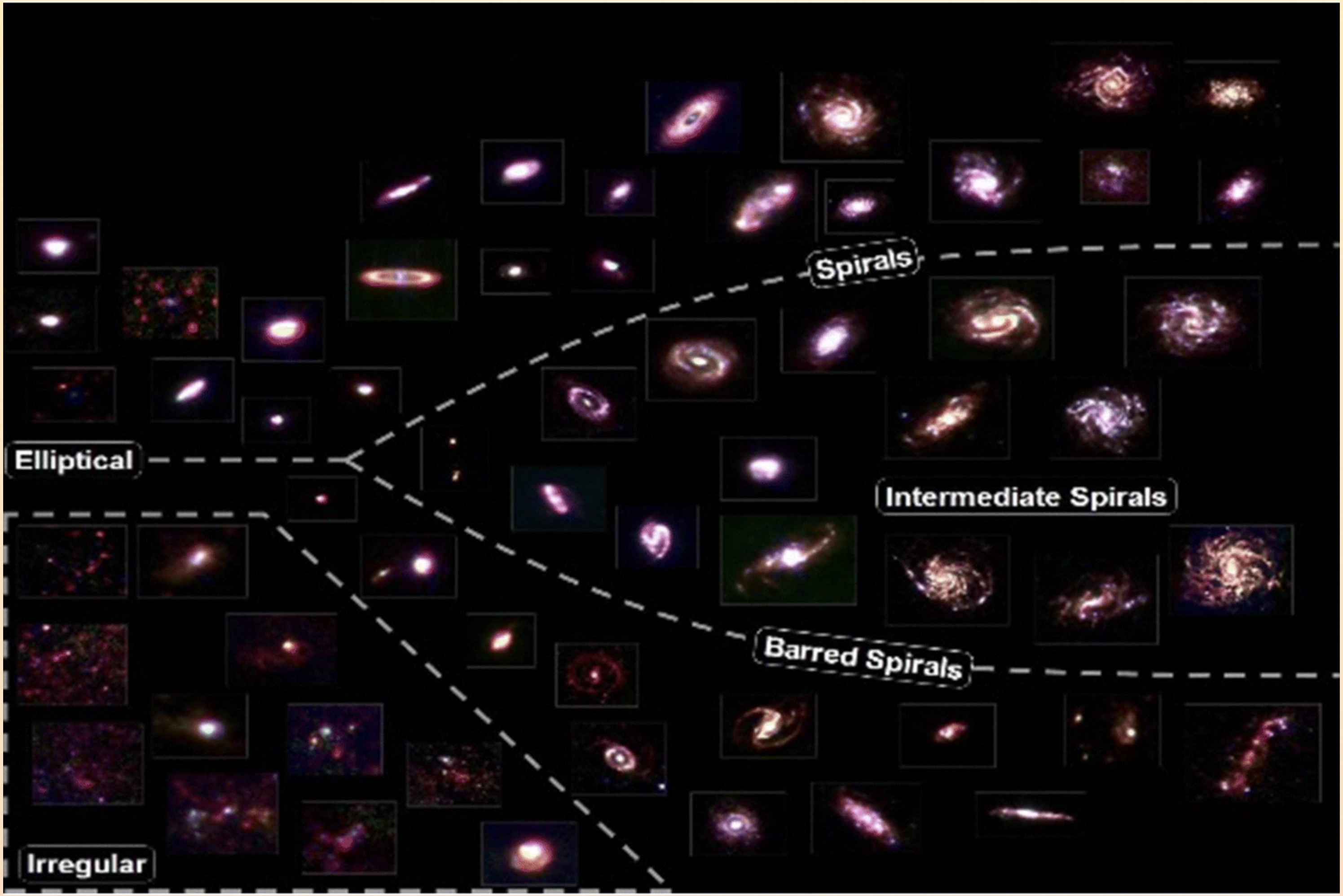
Images from tue.nl and Yann LeCun's LeNet

Intermediate representations



—
Images from Goodfellow, Bengio, Courville, 2016

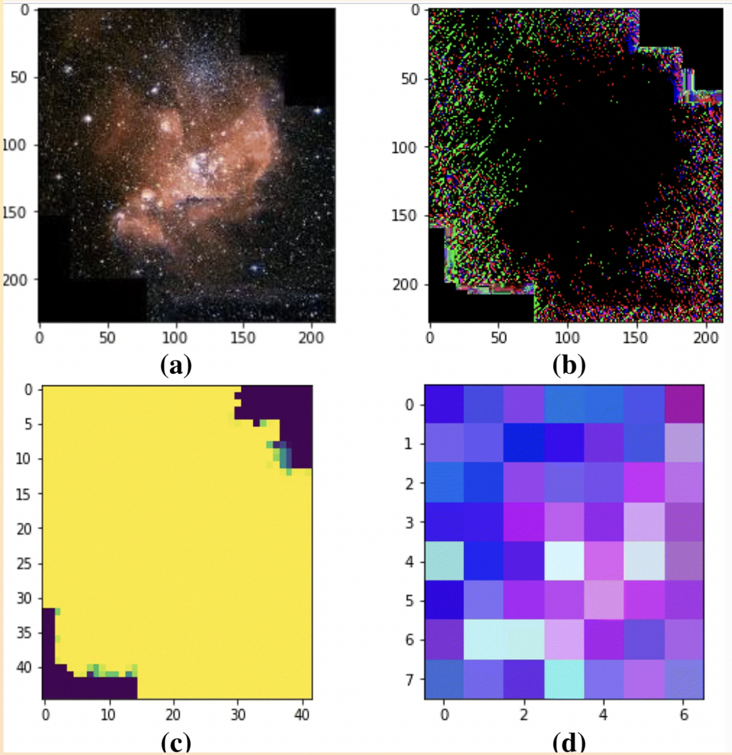
Morphology of galaxies

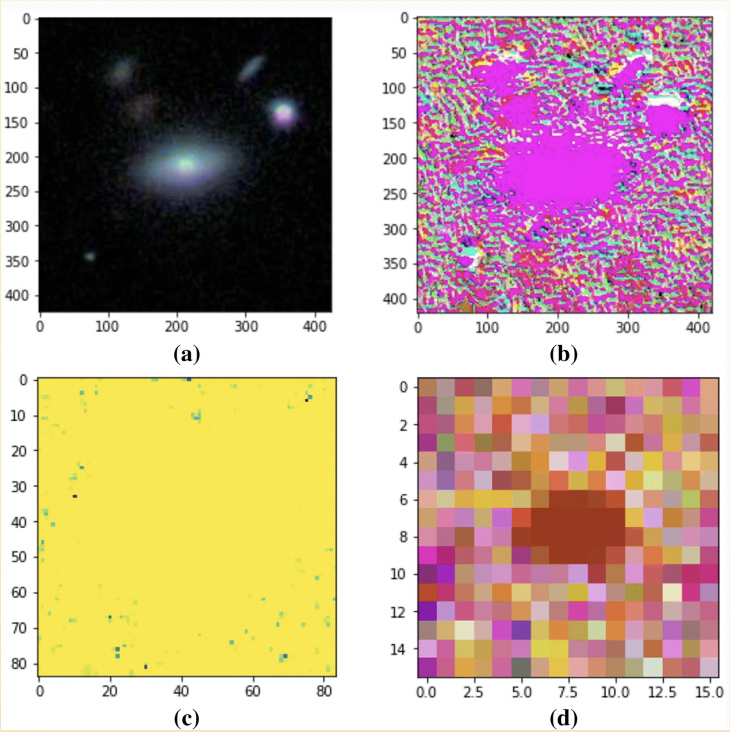


—

Image from Earth Science Informatics volume 13, pages 601–617 (2020)

Representations of galaxies...

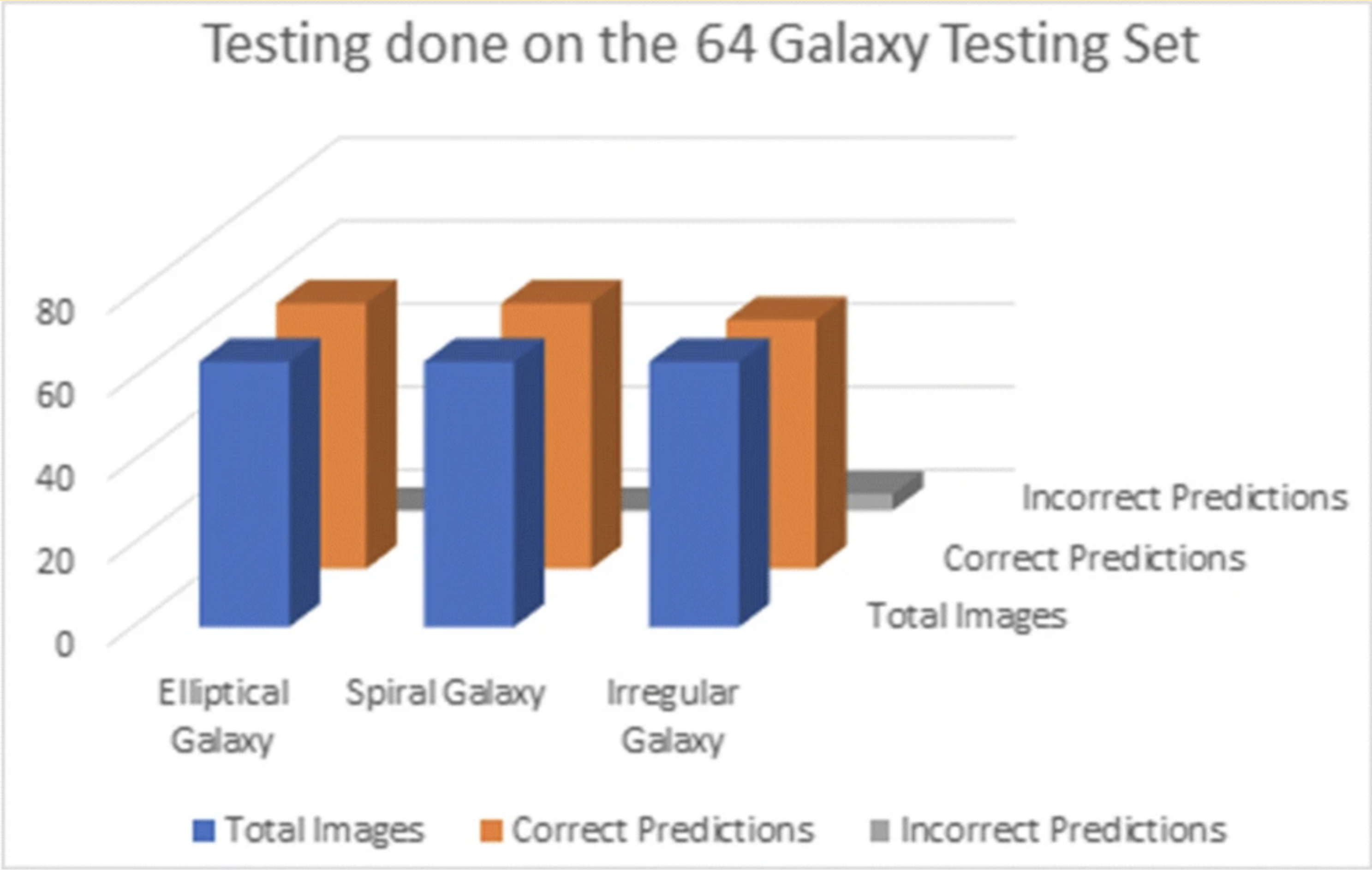




—

Image from Earth Science Informatics volume 13, pages 601–617 (2020)

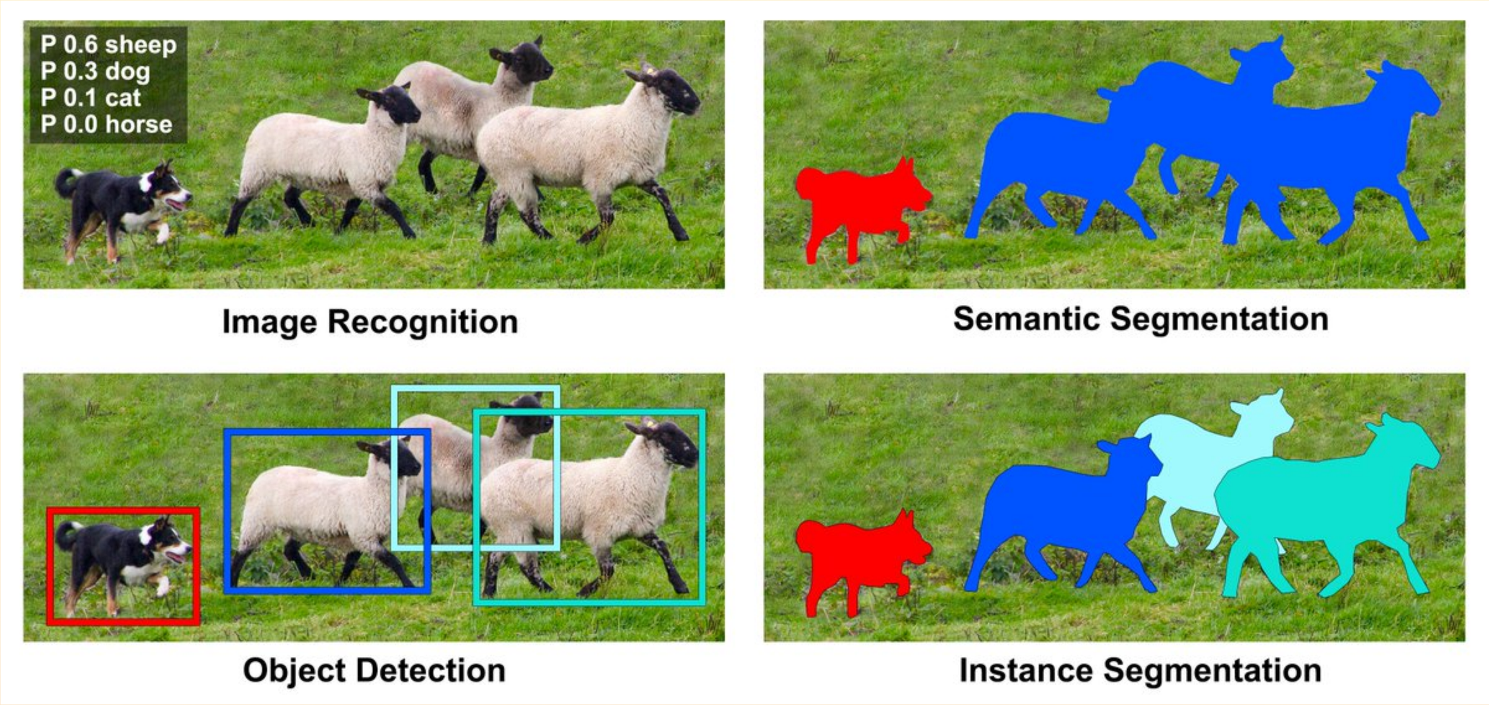
...work pretty well



—

Image from Earth Science Informatics volume 13, pages 601–617 (2020)

Semantic representations



—

Image from ai-pool.com

What about time?

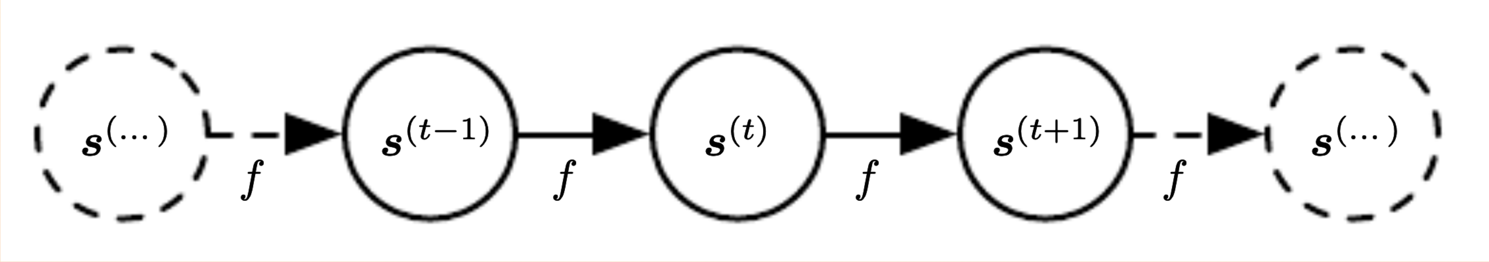
- Convolutional network: process grid of values (e.g. images)
- Recurrent networks: process a sequence of values indicised by a "time" component
  - Language is a sequence
- Parameter sharing crucial to generalize:
  - lengths unseen in training
  - different positions in the sentences
- Without parameter sharing, a network would have to learn all the language rules at each step of the sequence
  - Very impractical
- Both scale very well (thanks to parameter sharing)

Convolutional networks for sequences?

- Could "link" the steps of the sequence via the convolution
- Use the same kernel at each time step
- Shallow: it links only neighbouring time steps

Recurrent network

- Use the same parameter at the same step, $s^{(t)} = f(s^{(t-1)}, \theta)$
  - Very deep structure

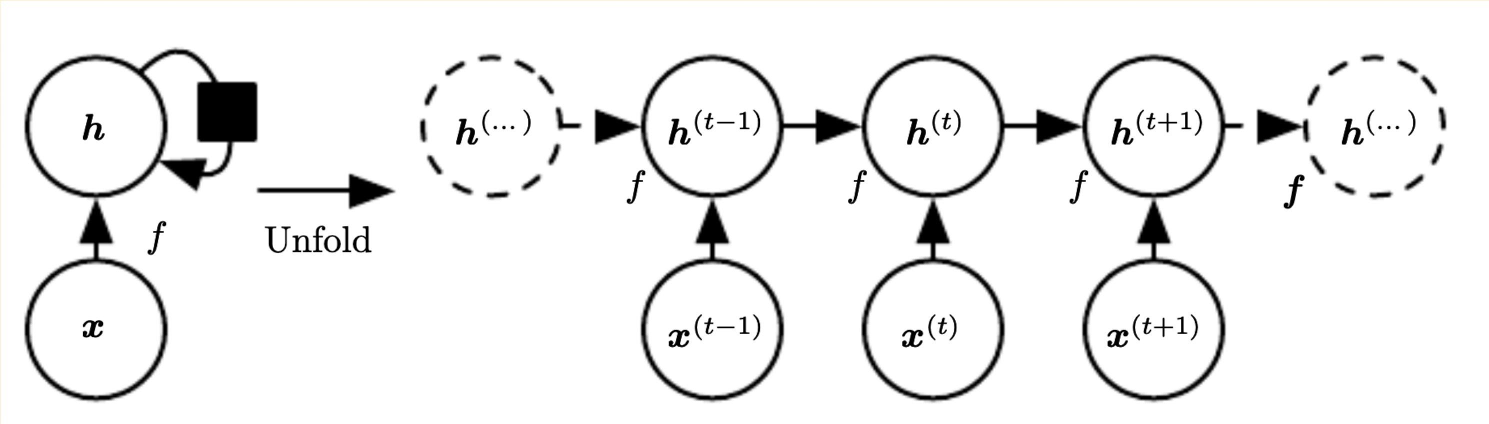


—

Images from Goodfellow, Bengio, Courville, 2016

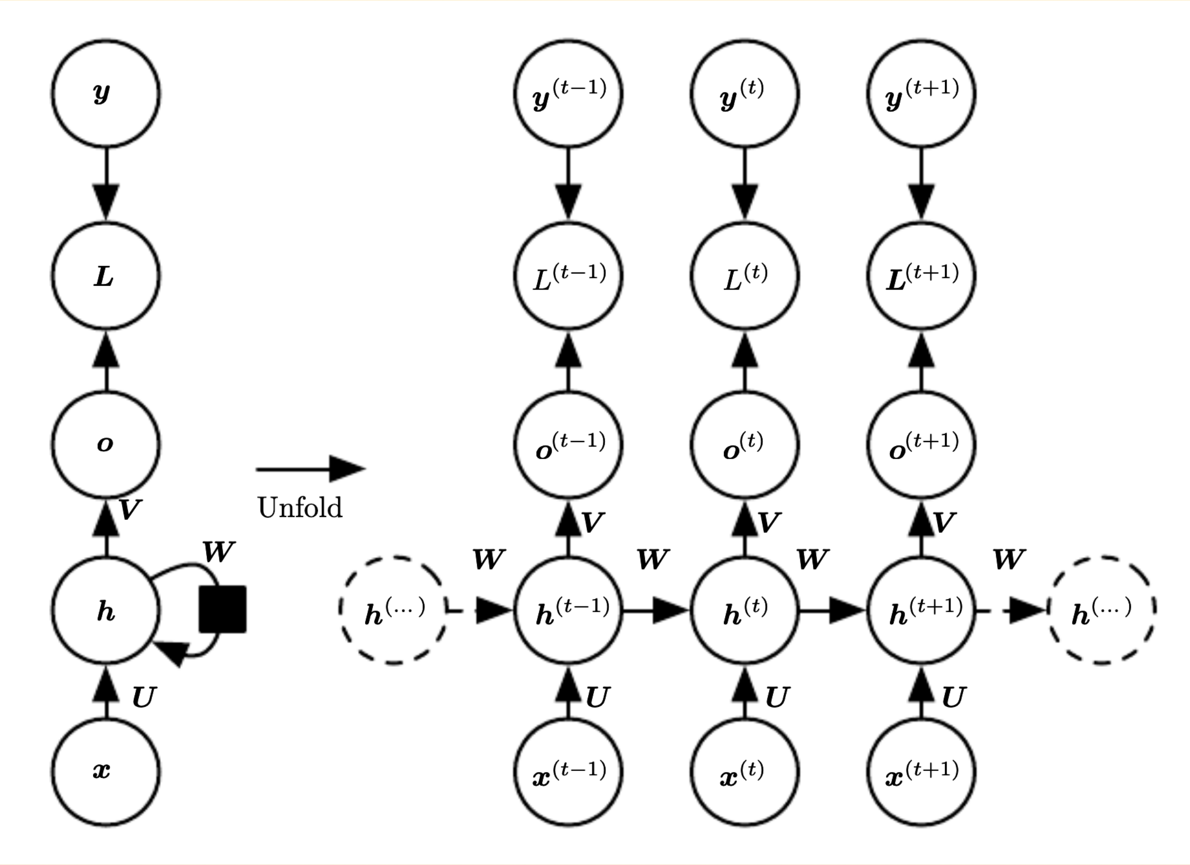
Unfold the graph

$$s^{(3)} = f(s^{(2)}, \theta) = f(f(s^{(1)}, \theta), \theta)$$



Vast zoology

= An output at each time step, recurrent connections between hidden units

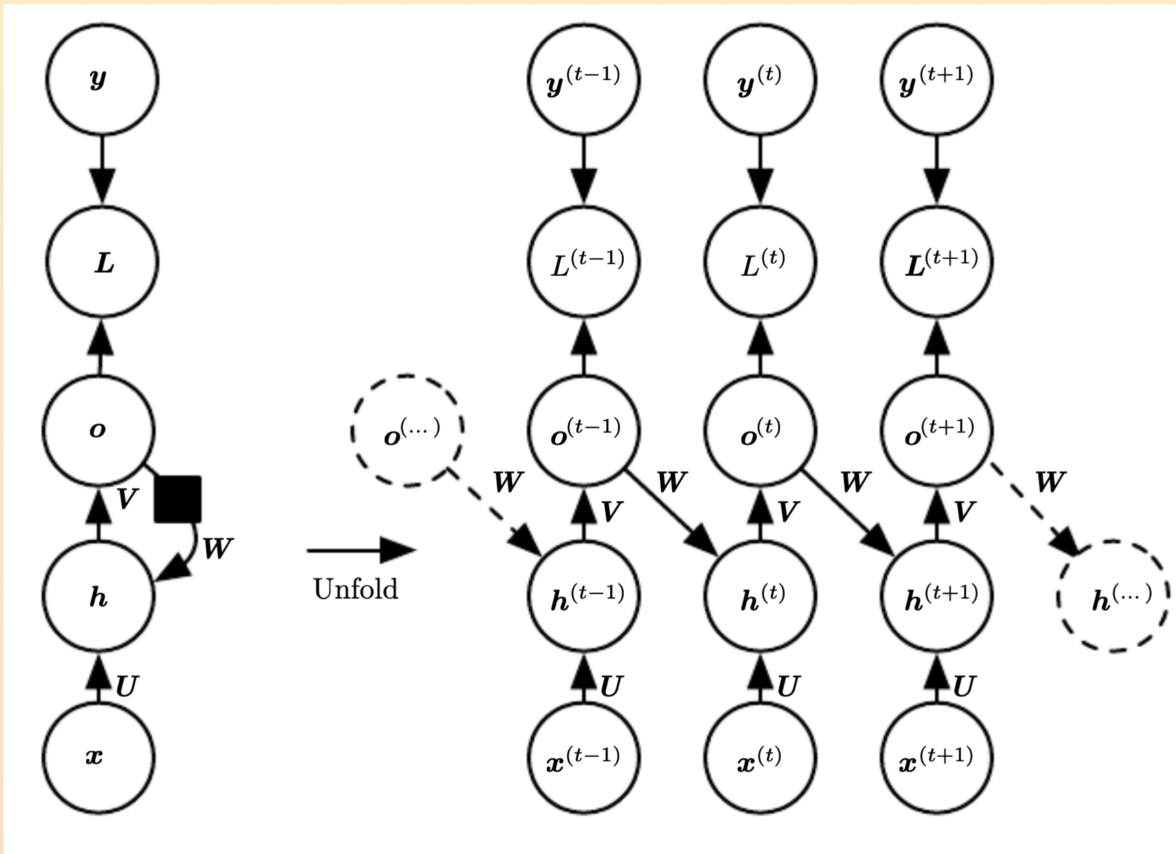


—

Images from Goodfellow, Bengio, Courville, 2016

Vast zoology

- An output at each time step, recurrent connections only from the output at one time step to the hidden units at the next time step

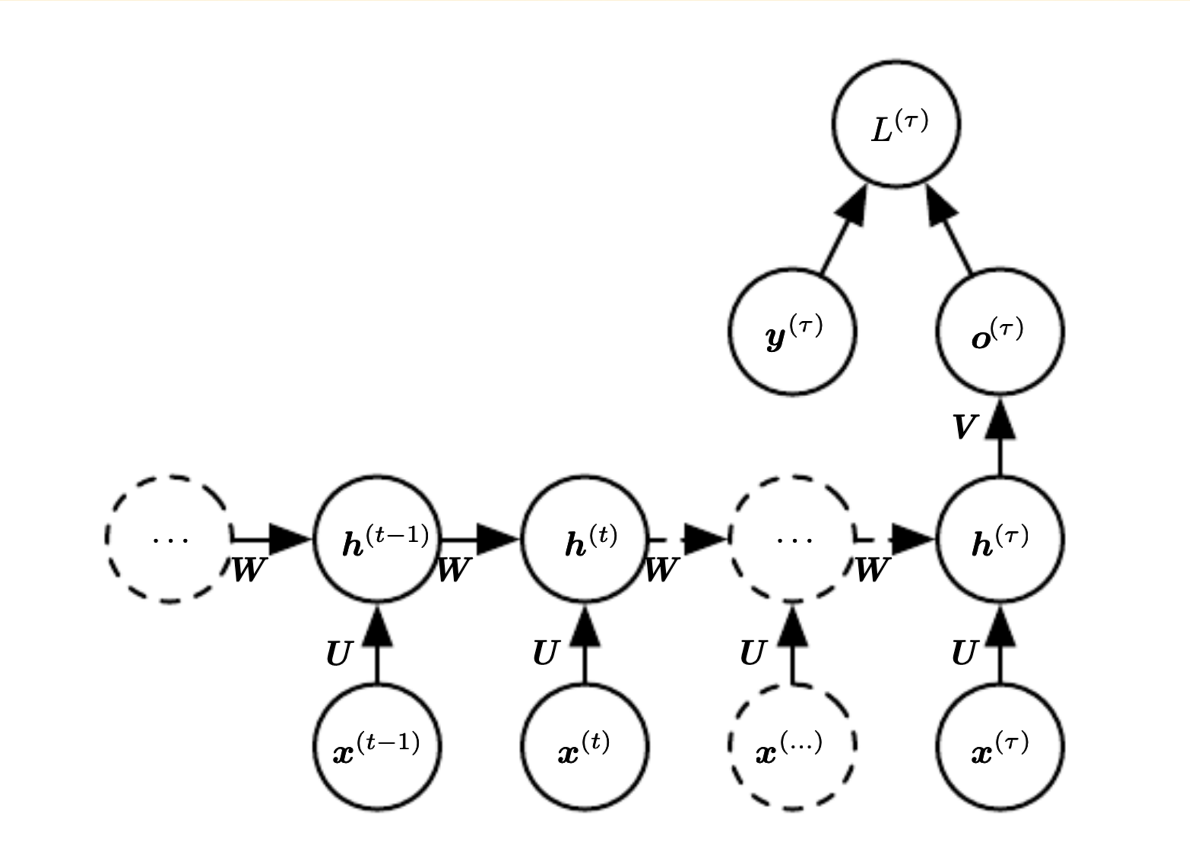


—

Images from Goodfellow, Bengio, Courville, 2016

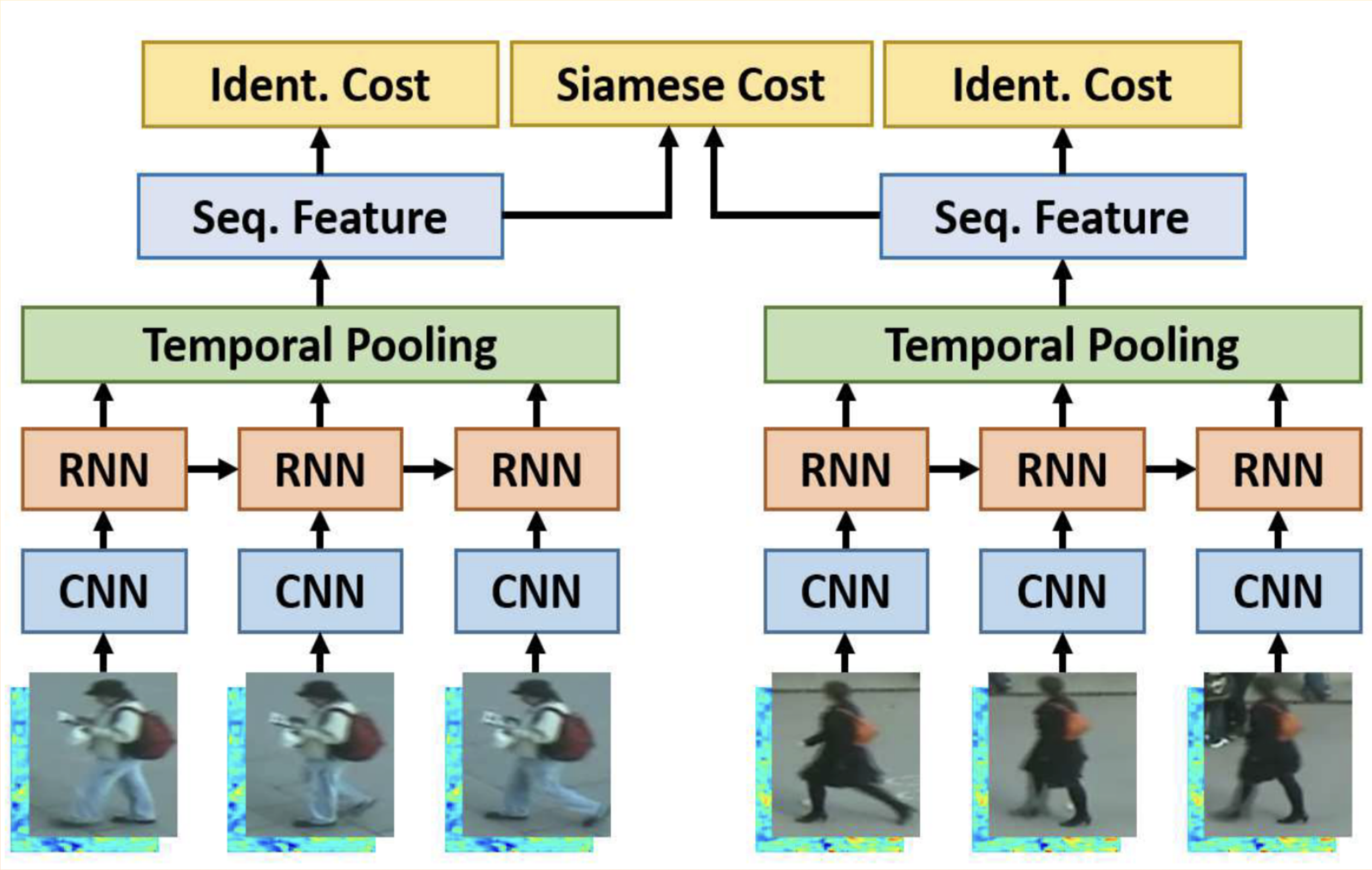
Vast zoology

- Recurrent connections between hidden units, that read an entire sequence and then produce a single output



—
Images from Goodfellow, Bengio, Courville, 2016

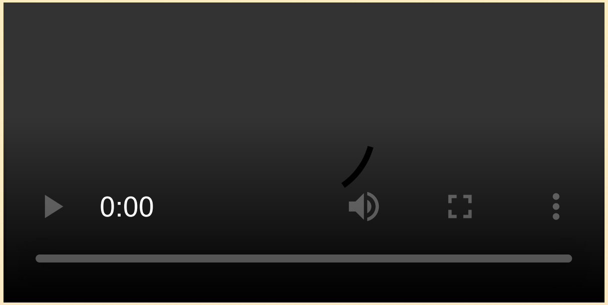
Sequences of images

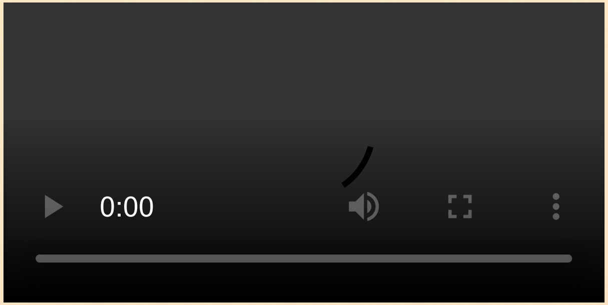


—

Picture from 10.1109/CVPR.2016.148

Real-time segmentation

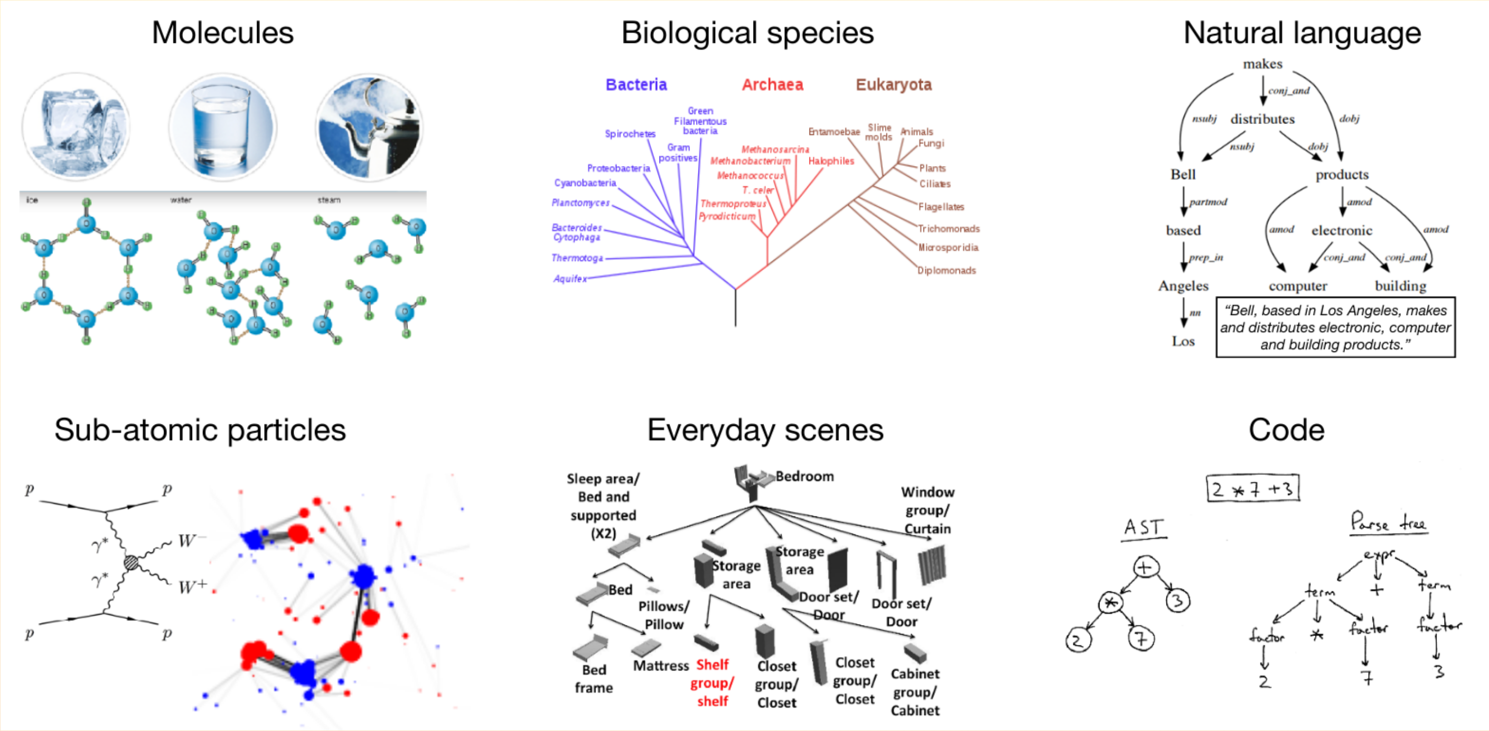




—

Videos from YouTube, autonomous driving and YouTube, cancer research

Graphs Represent Structure



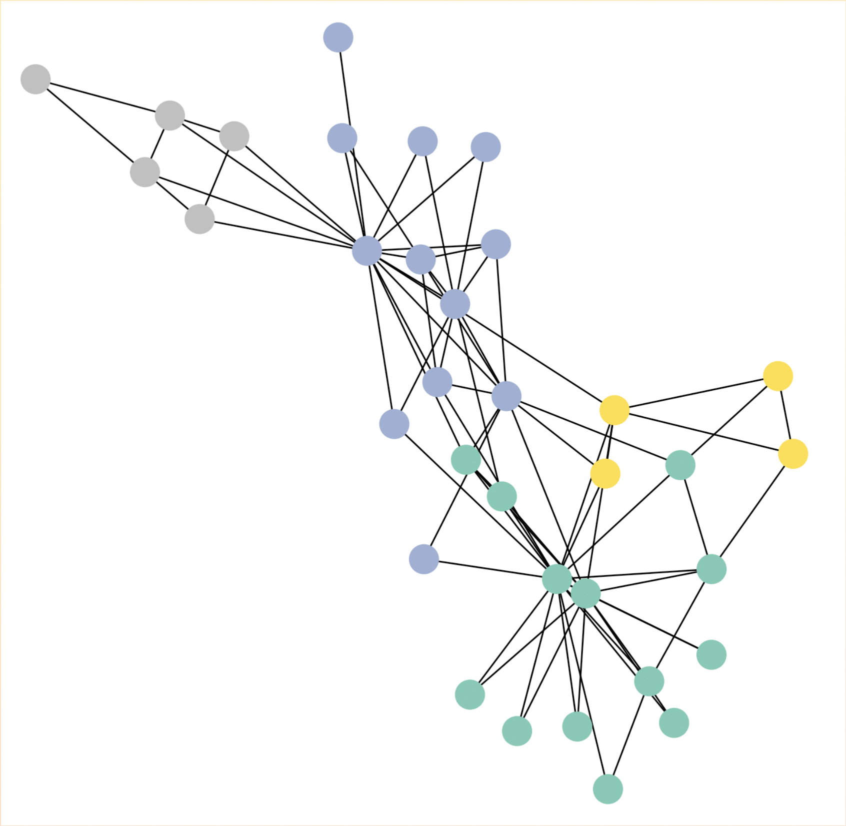
—

Image from Peter Battaglia

Graph networks

- Represent data as point clouds

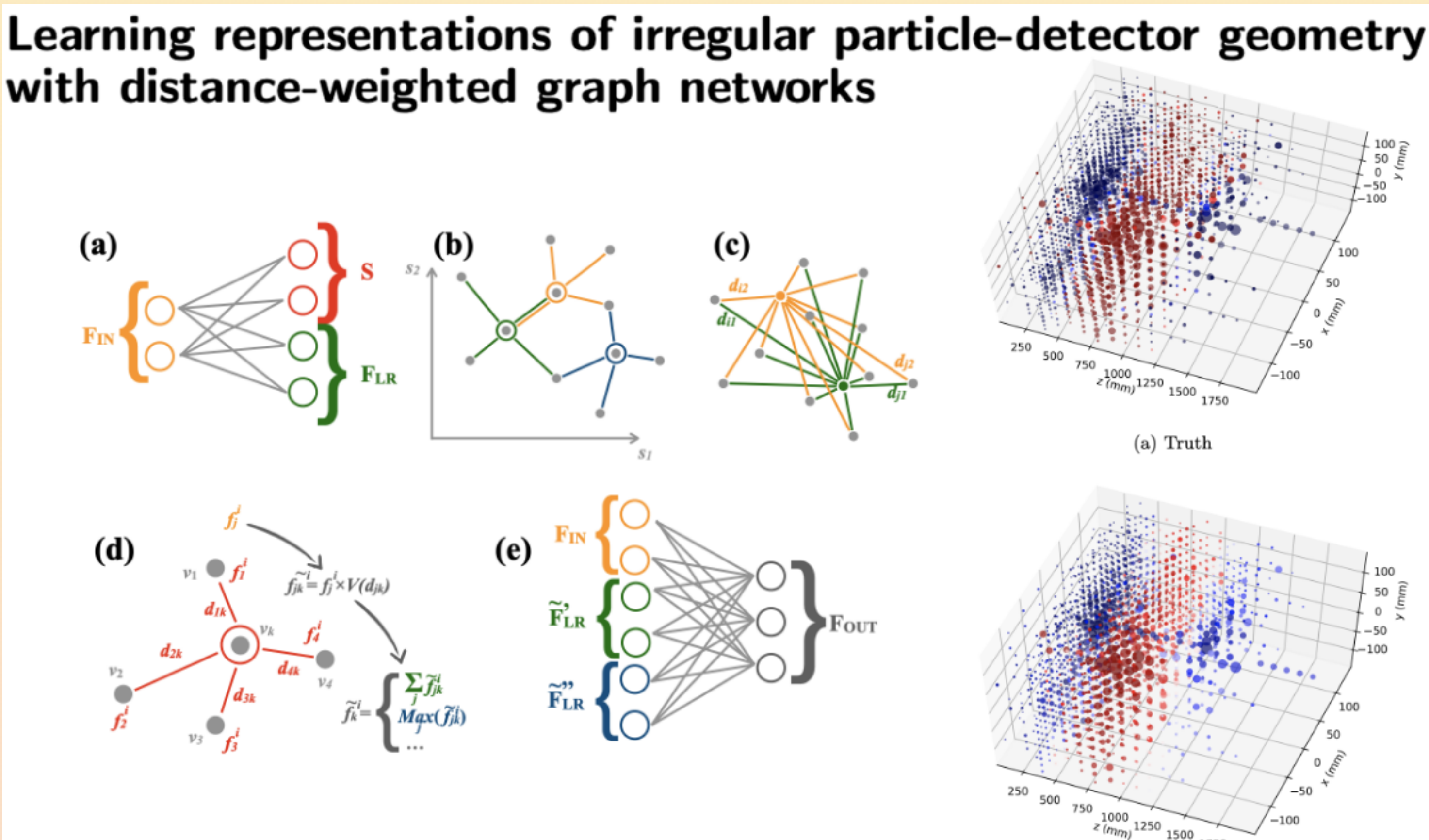
- Connect data points with weight-dependent connections
- Train the network to find which weights are strongest
  - Learng the connectivity structure of the data



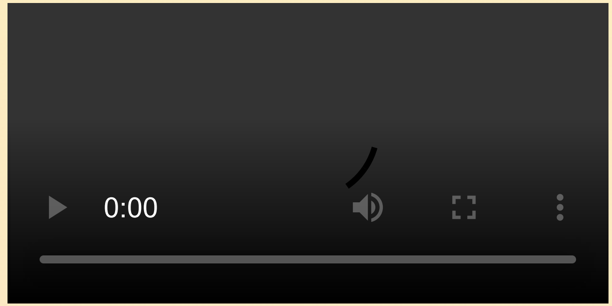
—
Image by me

CMS High-granularity calorimeter

- 6 million cells with $\sim 3mm$ spatial resolution, over $600m^2$ of sensors
- Non-projective geometry



Graphs for water simulation



—

Video by Peter Battaglia's team

Plug the Physics into the AI

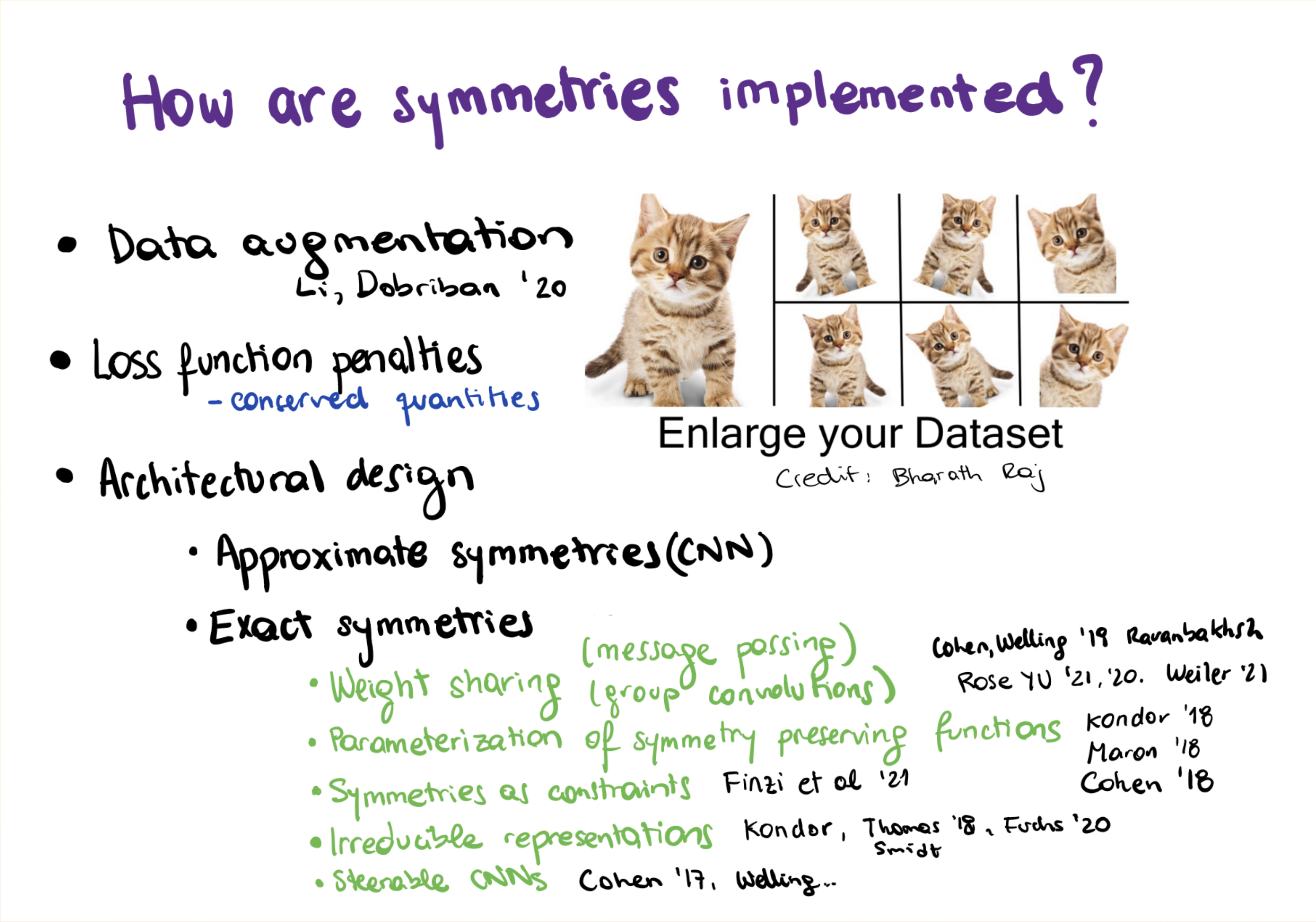


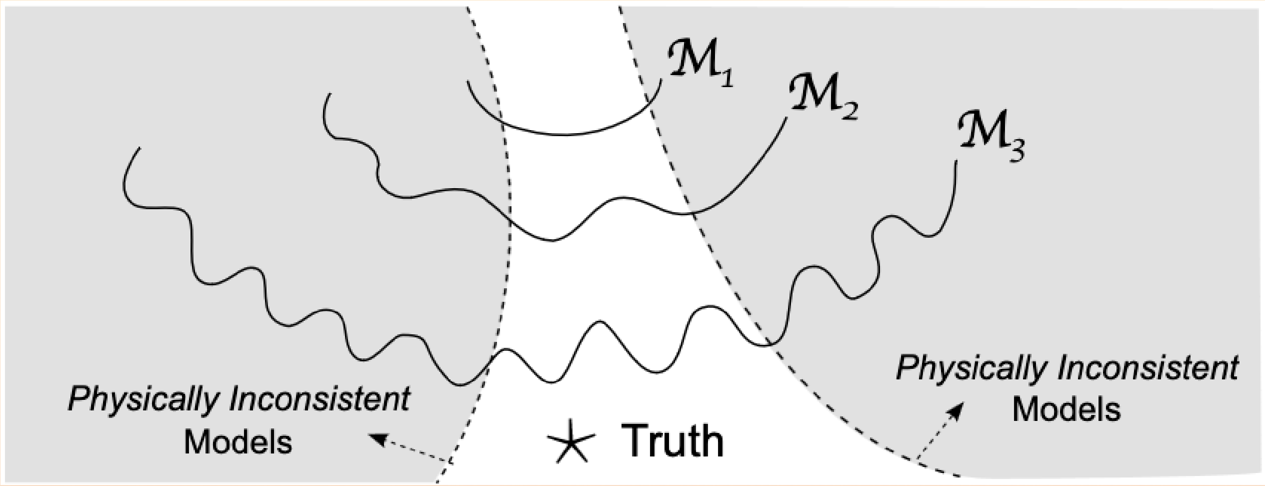
Illustration by Soledad Villar

Plug the physics into the AI: constraints

$$\hat{y} = f(\mathbf{x}, \theta)$$

- Encode physics knowledge (e.g. inconsistency of models) inside the loss function as a penalty term

$$\mathbf{J}(\mathbf{w}) = Loss(y, \hat{y}) + \lambda||\mathbf{w}||_2^2 + \gamma\Omega(\hat{y}, \Phi)$$



—

Figure from 10.1109/TKDE.2017.2720168

Plug the physics into the AI: network structure

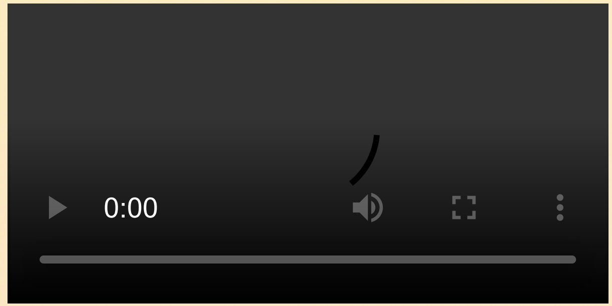
- Equivariance under group transformation can e.g. enforced by convolutional layers
- Some implementations available in pytorch

$$
\begin{tikzcd}
L_{V_1}(\mathcal{X}_1) \arrow[r, "\mathbb{T}_g"] \arrow[d, "\phi"] & L_{V_1}(\mathcal{X}_1) \arrow[d, "\phi"] \\
L_{V_2}(\mathcal{X}_2) \arrow[r, "\mathbb{T}'_g"] & L_{V_2}(\mathcal{X}_2)
\end{tikzcd}
$$

—
Figure from 10.1109/TKDE.2017.2720168

Plug the Physics into the AI

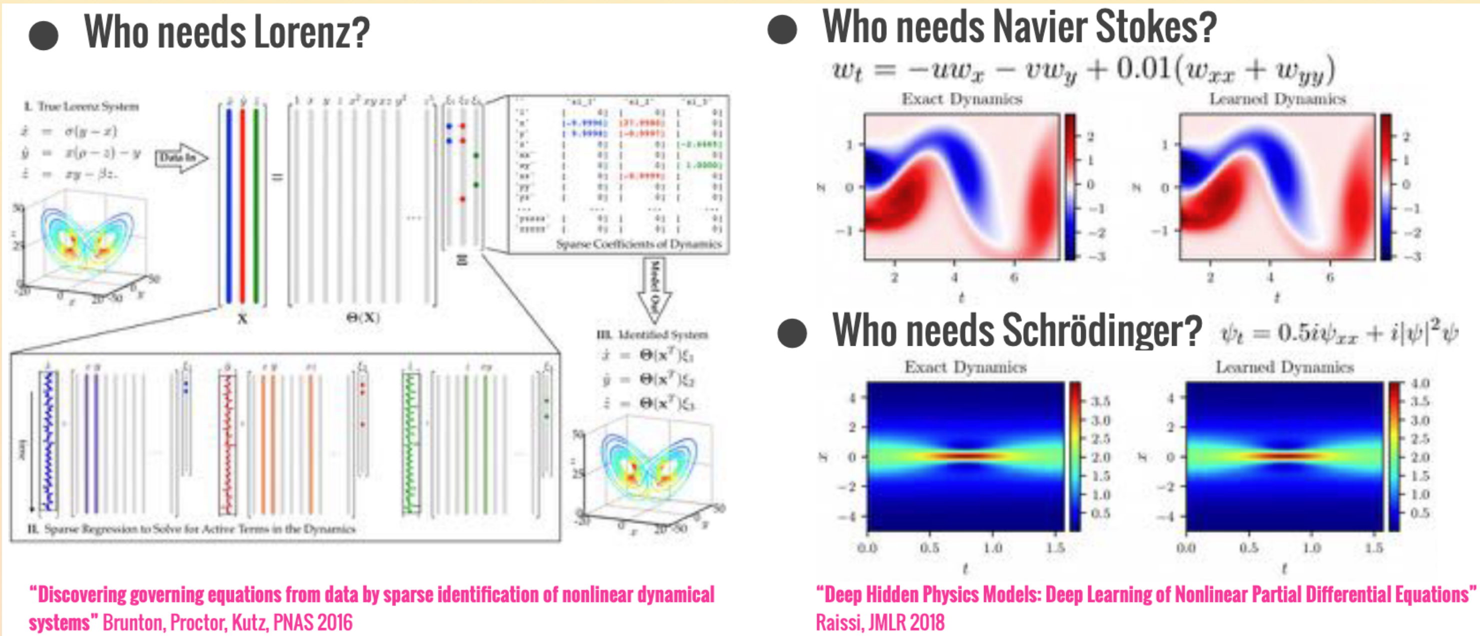
- Physics-aware differential equations solving



—
Animation from 2202.06988

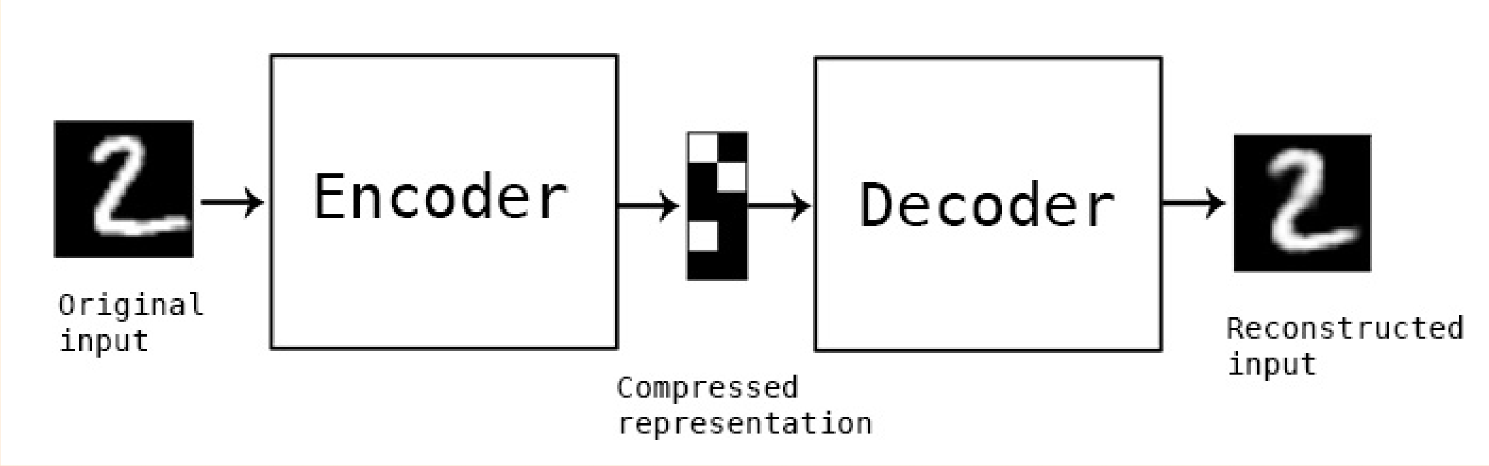
Plug the Physics into the AI

- Several ODE problems now solvable via neural networks



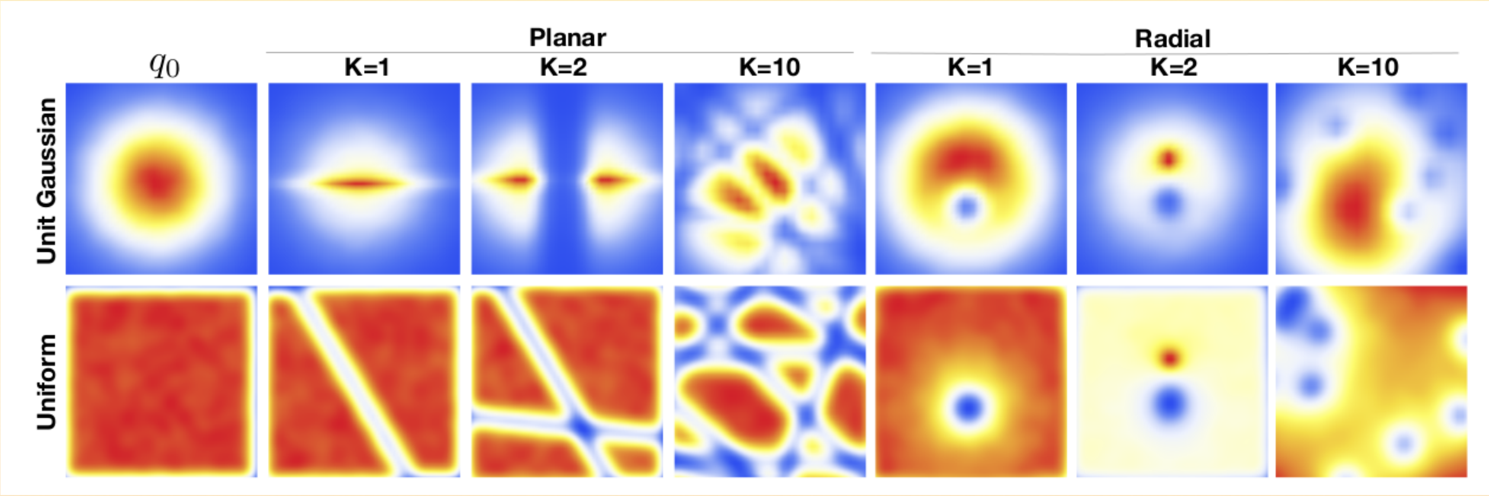
Autoencoders

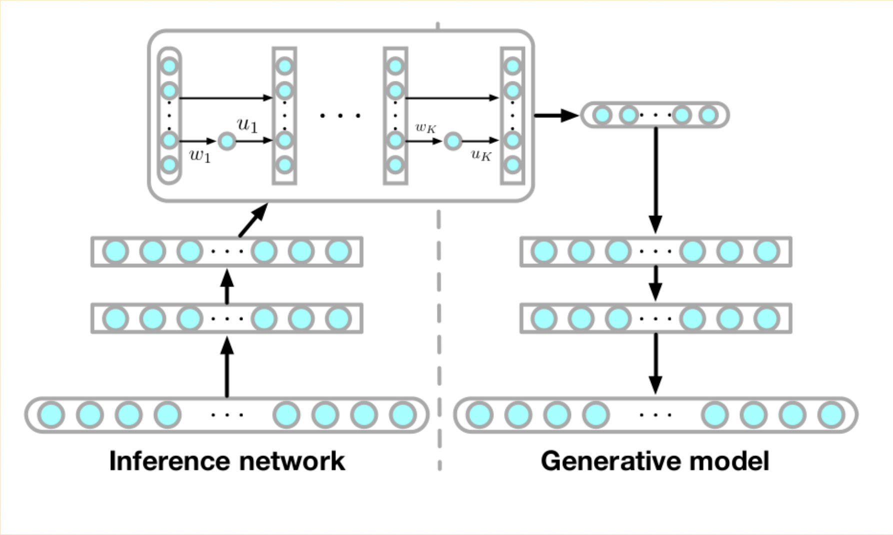
- Learn the data itself passing by a lower-dimensional intermediate representations
    - Capture data generation features into a lower-dimensional space
- Can use for anomaly detection
- Can sample from the latent space to obtain random samples (generative AI)



—
Figure from...

Invertible networks





—
Images from Rezende and Mohamed, 1505.05770

Solve inverse problems ("unfolding")

- Correct detector observation noise to recover source distribution

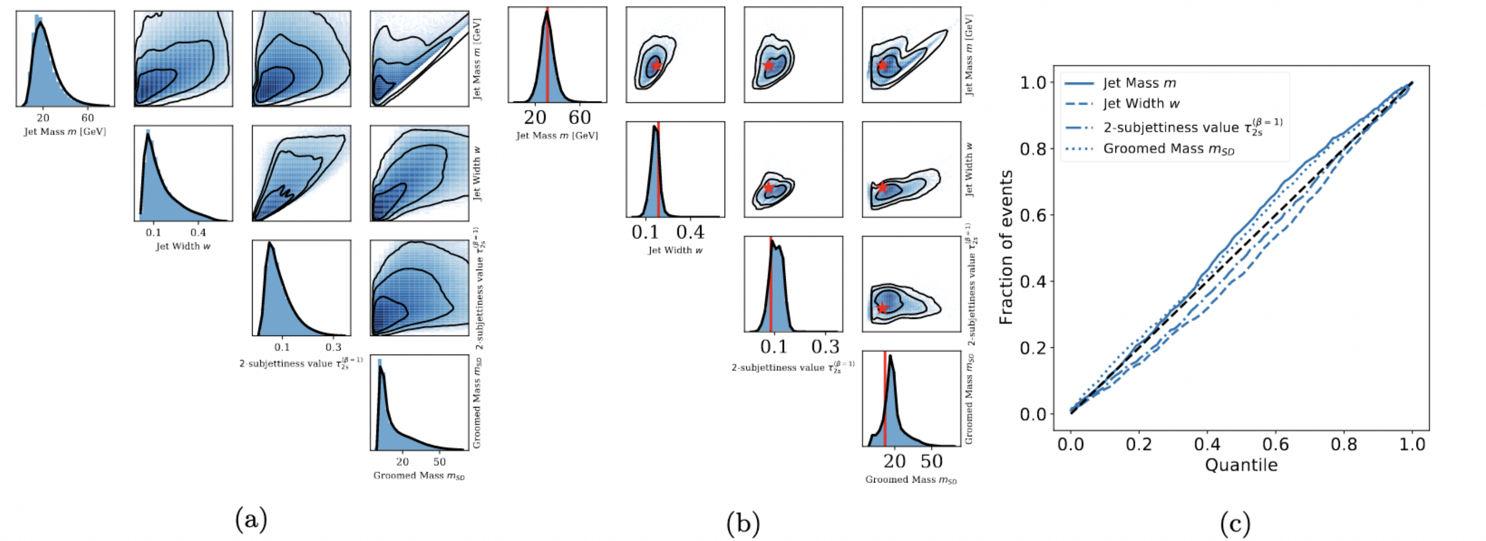
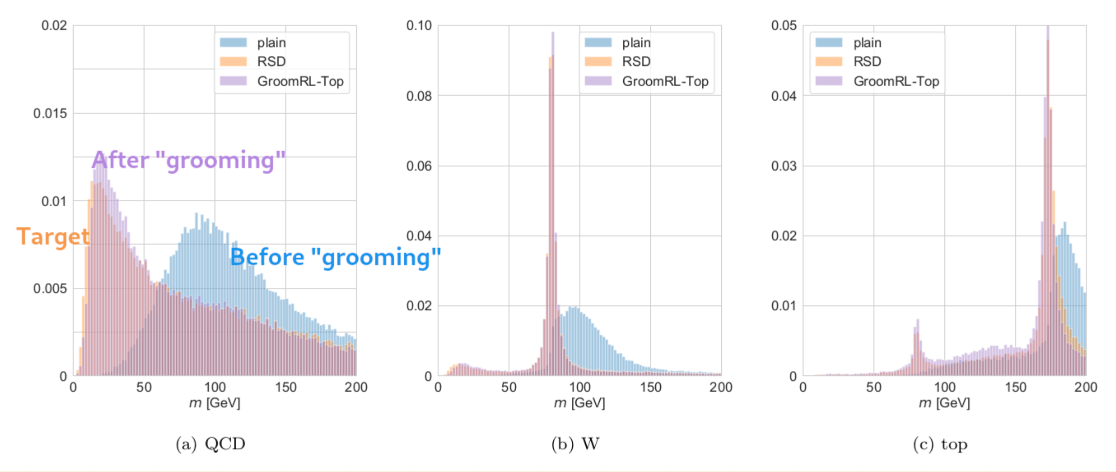


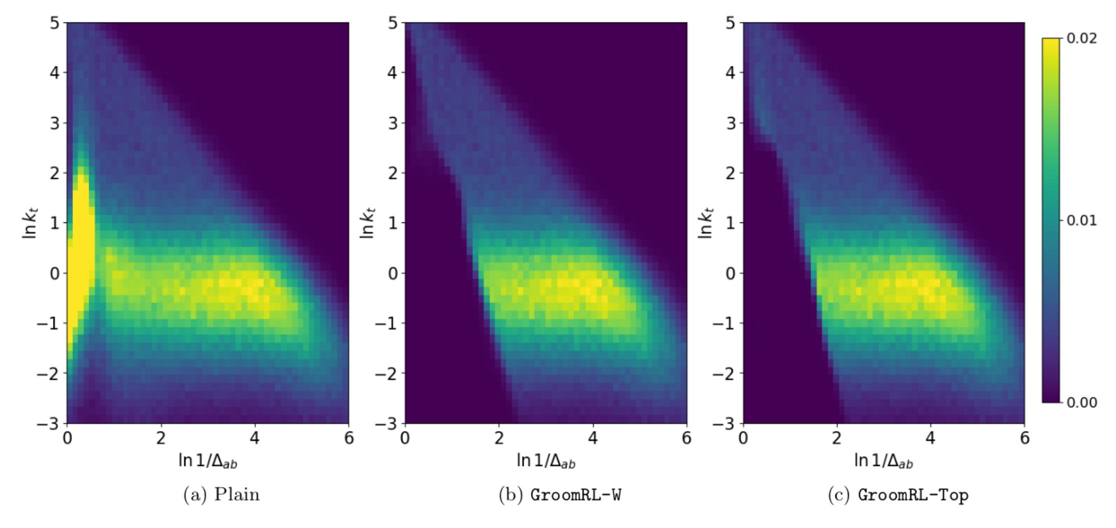
Figure 5: Neural Empirical Bayes for detector correction in collider physics. (a) The source distribution $p(\mathbf{x})$ is shown in blue against the estimated source distribution $q_\theta(\mathbf{x})$ in black. (b) Posterior distribution obtained with rejection sampling, with generating source sample $\mathbf{x}$ indicated in red. (c) Calibration curves for each jet property obtained with rejection sampling on 10000 observations. In (a) and (b), contours represent the 68-95-99.7% levels.

—
Images from 2011.05836

Interpretability



(a) QCD (b) W (c) top



(a) Plain (b) GroomRL-W (c) GroomRL-Top

—
Images from 1903.09644

Encode sequences

- One-hot encoding for unordered sequences
    - Works e.g. for text

Feature (Color)
Red
Green
Yellow
Green
Red

One Hot Encoding

One Hot Encoded Vector	Red	Green	Yellow
[1,00]	1	0	0
[0,1,0]	0	1	0
[0,0,1]	0	0	1
[0,1,0]	0	1	0
[1,00]	1	0	0

SCALER Topics

—
Picture from scaler.com

Encode sequences

- "Yellow" $[0, 0, 1]$ can be predicted as "0 for red and 0 for green"
    - One-hot-encoded features highly correlated ("multicollinearity")
- Dummy variable trap
    - Drop one of the "dimensions"

Feature (Color)
Red
Green
Yellow
Green
Red

One Hot Encoding →

Red	Green
1	0
0	1
0	0
0	1
1	0

Yellow Column dropped to avoid
the Dummy Variable Trap

SCALER Topics

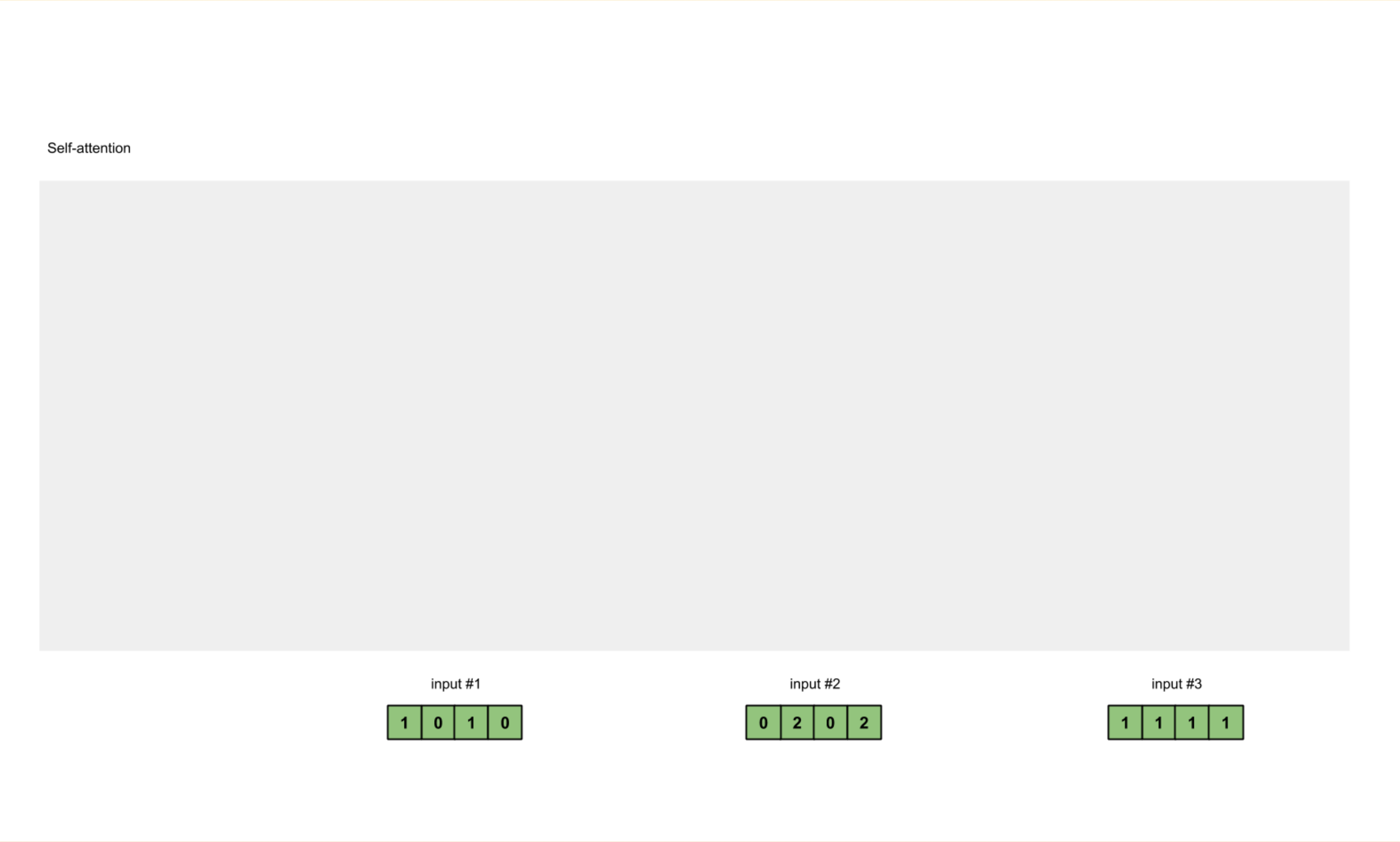
—
Picture from scaler.com

Self-attention

- Capture dependencies and relationships within inputs
  - Mostly in natural language processing and computer vision
- N inputs, N outputs
  - Allow inputs to interact with eacho other and find out which ones to pay attention to
  - Output is an aggregate of interactions and attention scores
- Useful for:
  - Long-range dependencies: understand complex patterns and dependencies
  - Contextual understanding: assign appropriate weights to important elements in the sequence
  - Parallel computation: can be computed in parallel $\rightarrow$ efficient and scalable for large datasets.

Self-attention

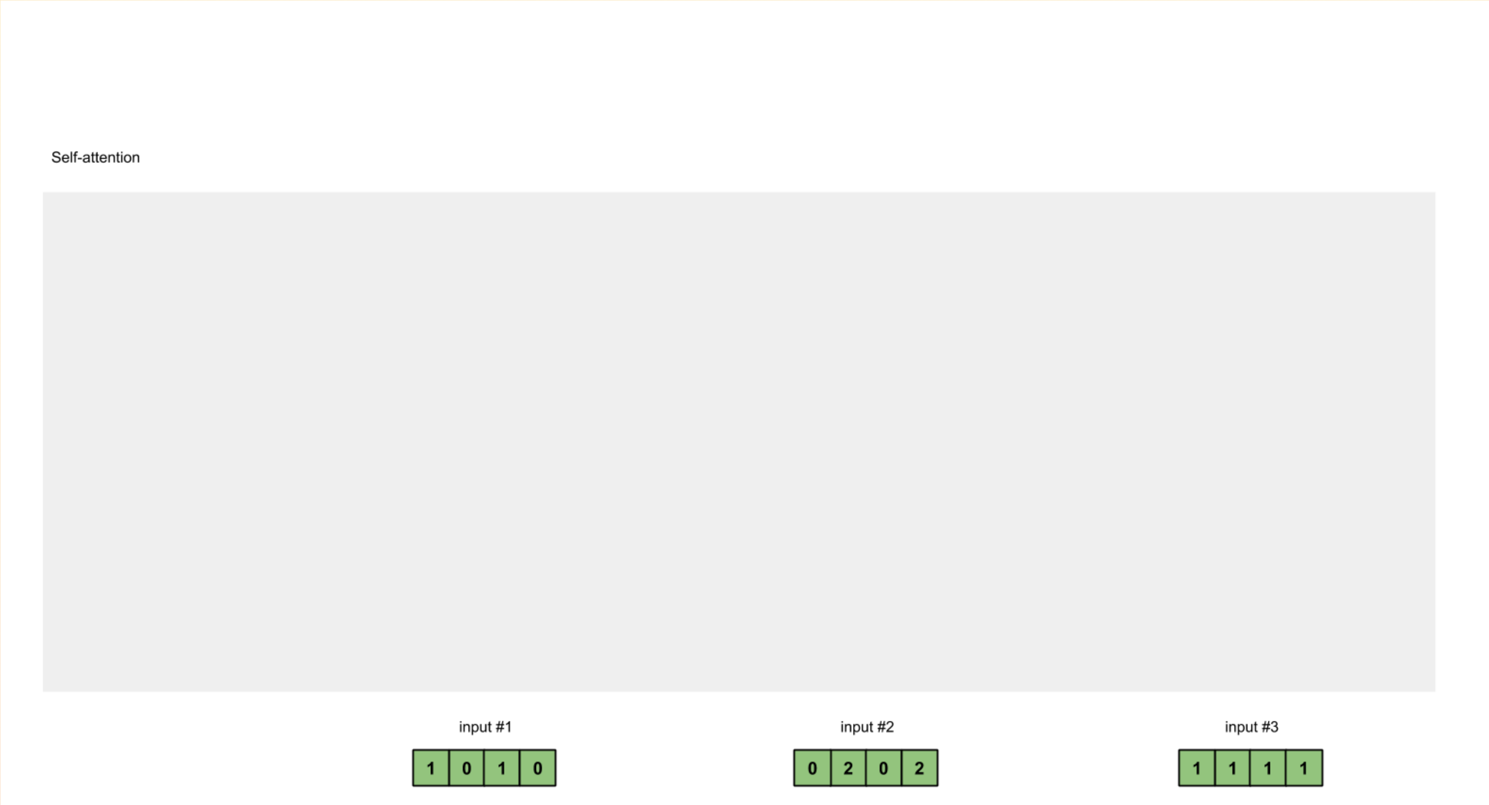
- Inputs (green) must be represented as: key (orange), query (red), value (purple)
  - Initially, by random reweighting of inputs themselves



—
Animation from towardsdatascience

Self-attention

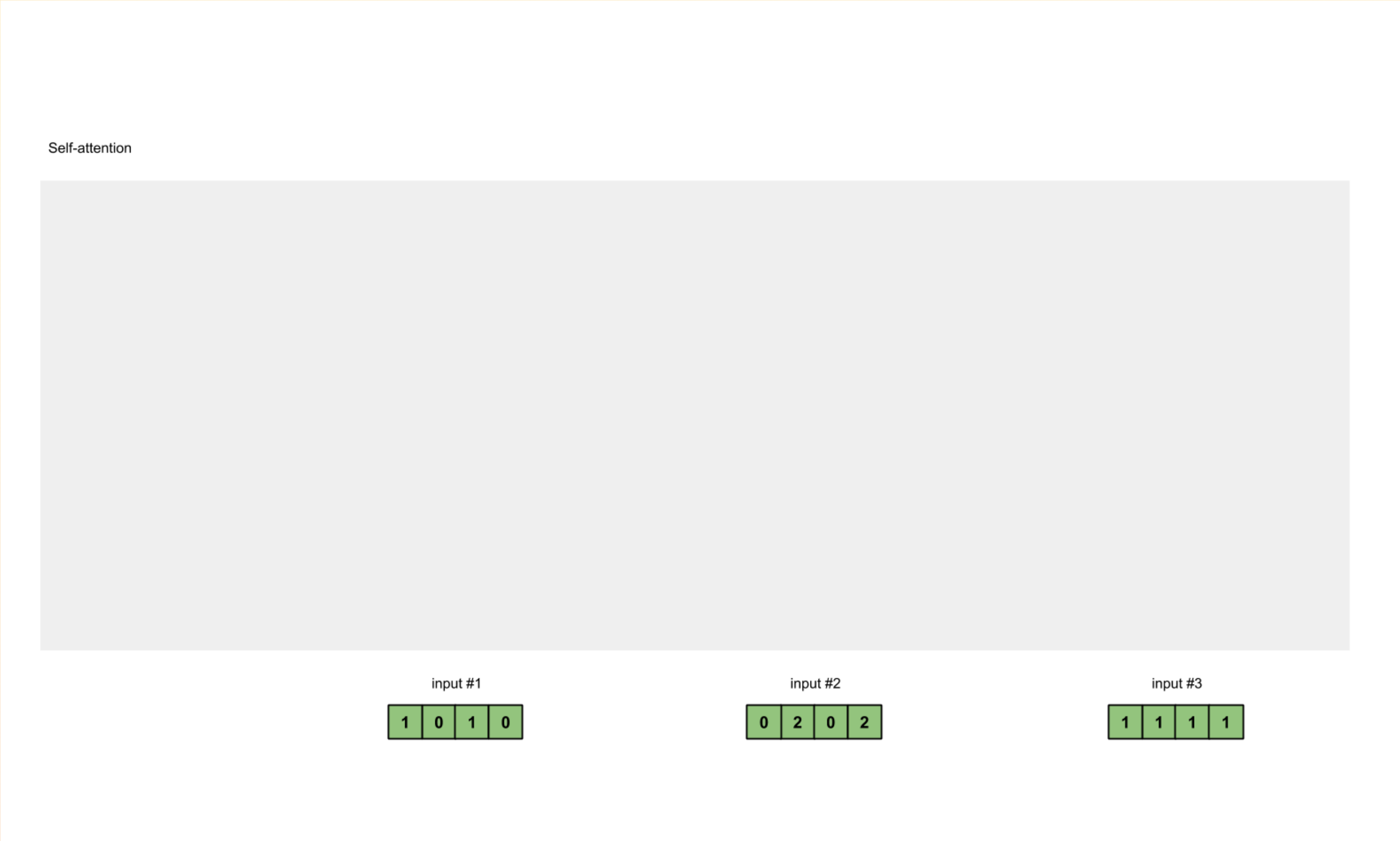
- Calculate attention score
  - Multiply (dot product) each query with all keys
  - For each query: N keys $\rightarrow N$ attention scores



—
Animation from towardsdatascience

Self-attention

- Activation function (softmax) of attention scores



—
Animation from towardsdatascience

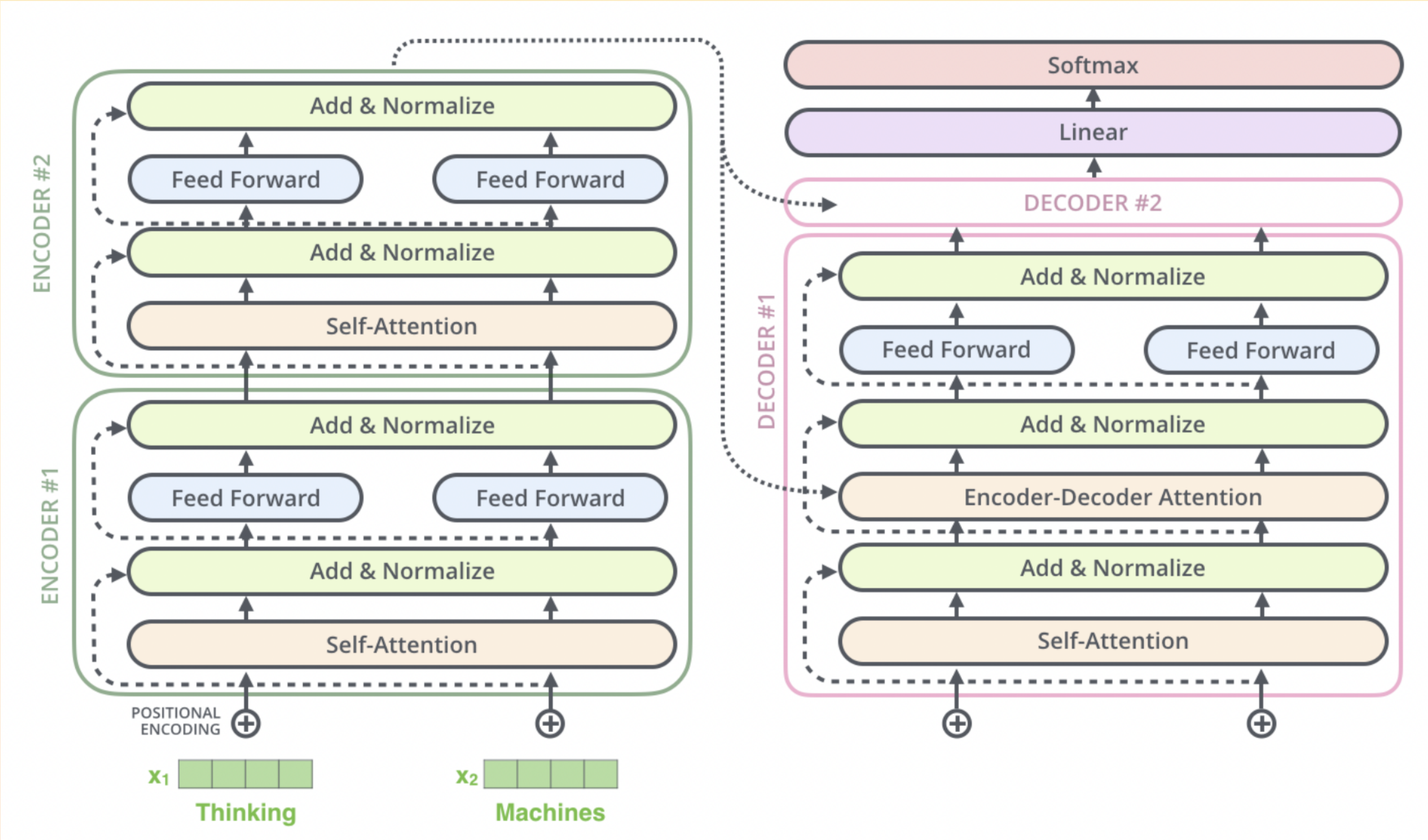
Self-attention

- Calculate alignment vectors (yellow), i.e. weighted values
  - Multiply each attention score (blue) by its value (purple)
- Sum alignment vectors to get input for output 1, repeat for 2 and 3

—
Animation from towardsdatascience

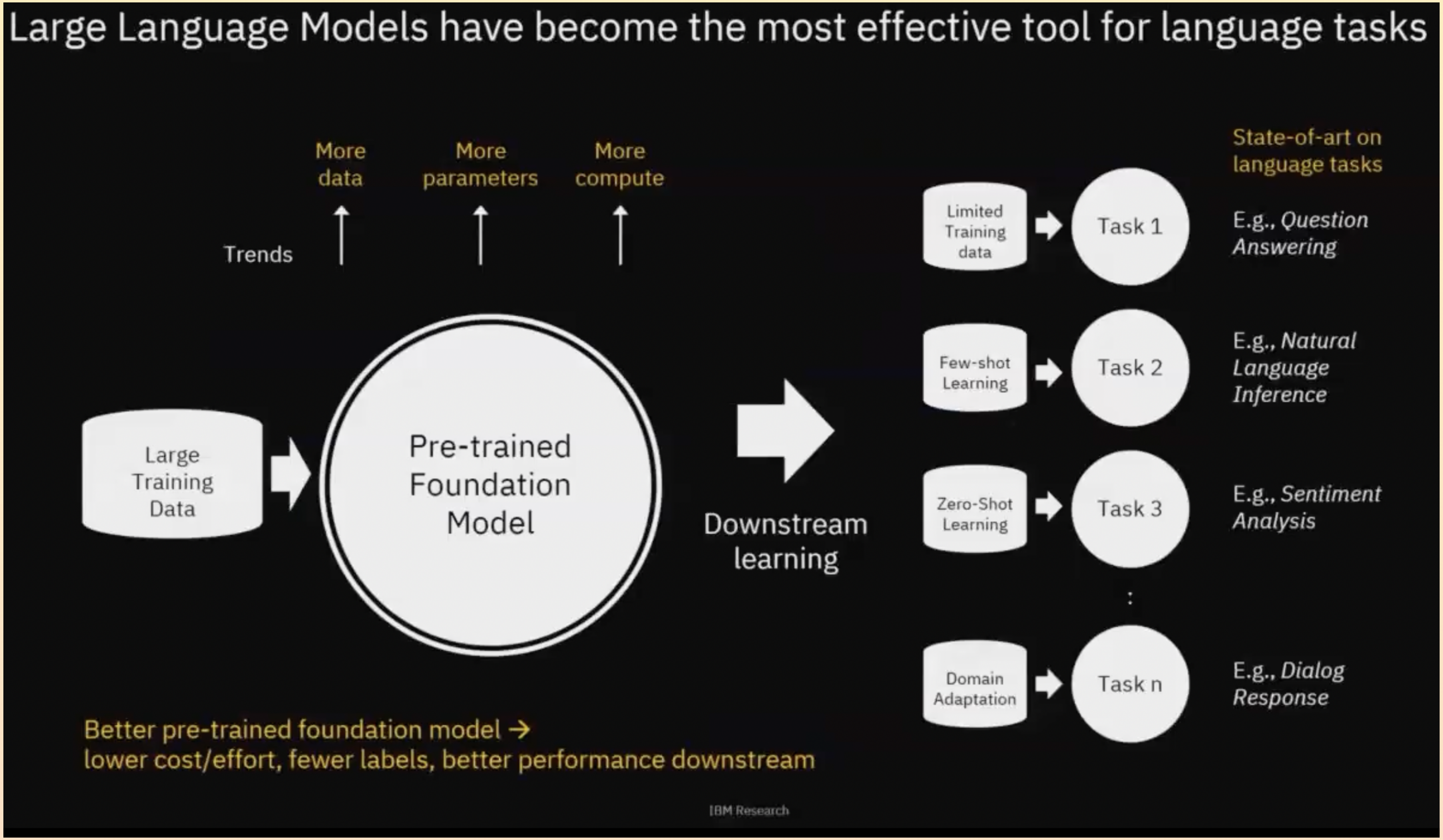
Transformers

- The engine behind GPT3



—
Image by Jay Alammar

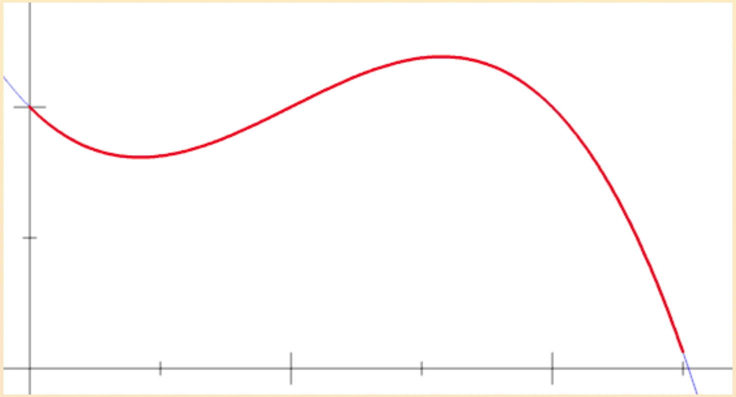
Foundation Models



—
Image by John R. Smith (IBM Research)

Translate Problems into Solutions

- Symbolic integration: find the analytic formula for the area of the curve

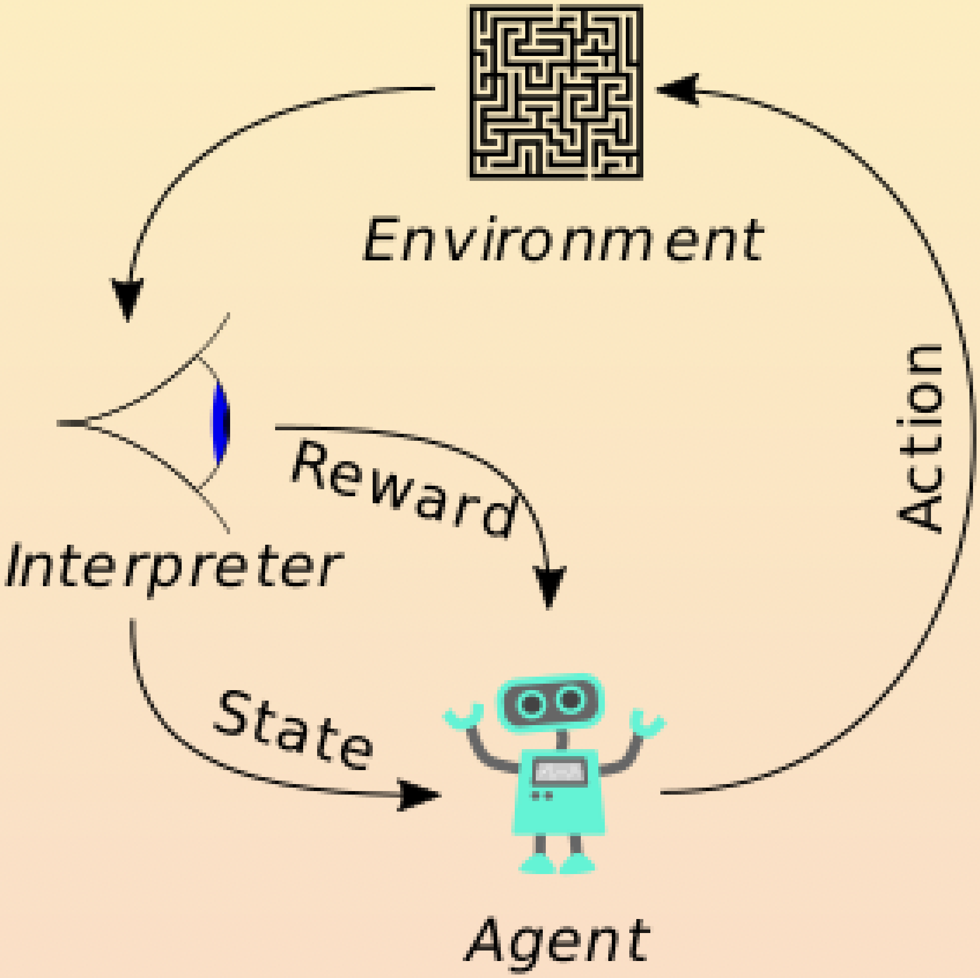


	Integration (BWD)	ODE (order 1)	ODE (order 2)
Mathematica (30s)	84.0	77.2	61.6
Matlab	65.2	-	-
Maple	67.4	-	-
Beam size 1	98.4	81.2	40.8
Beam size 10	99.6	94.0	73.2
Beam size 50	99.6	97.0	81.0

—

Table by Lampe, Charton (2019), image by Julia Inozemtseva

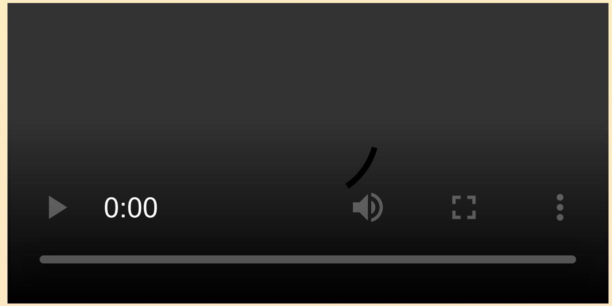
Reinforcement learning



—

Image by Megajuice on Wikimedia

From Videogames...



—
Image from YouTube video

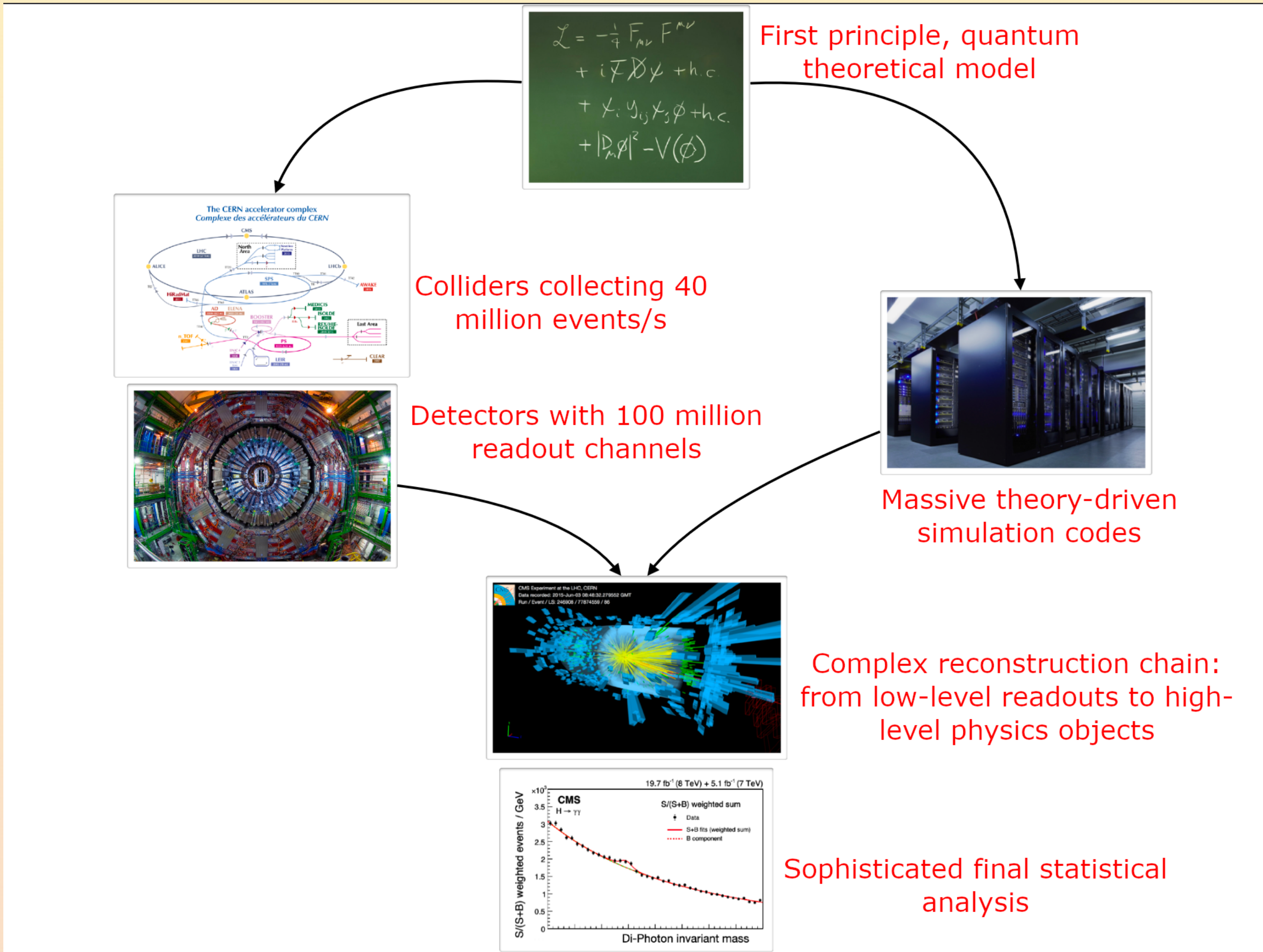
...to Physics

- Reward models consistent with the observed quark properties

charges	$\mathcal{Q} = \left(\begin{array}{c\|ccc\|ccc\|ccc\|c\|c} & Q_1 & Q_2 & Q_3 & u_1 & u_2 & u_3 & d_1 & d_2 & d_3 & H & \phi \\ \hline q & 6 & 4 & 3 & -2 & 2 & 4 & -3 & -1 & -1 & 1 & 1 \end{array}\right)$
$\mathcal{O}(1)$ coeff.	$(a_{ij}) \simeq \begin{pmatrix} -1.975 & 1.284 & -1.219 \\ 1.875 & -1.802 & -0.639 \\ 0.592 & 1.772 & 0.982 \end{pmatrix}$ $(b_{ij}) \simeq \begin{pmatrix} -1.349 & 1.042 & 1.200 \\ 1.632 & 0.830 & -1.758 \\ -1.259 & -1.085 & 1.949 \end{pmatrix}$
VEV, Value	$v_1 \simeq 0.224\,, \qquad \mathcal{V}(\mathcal{Q}) \simeq -0.598$
charges	$\mathcal{Q} = \left(\begin{array}{ccc\|ccc\|ccc\|c\|c} Q_1 & Q_2 & Q_3 & u_1 & u_2 & u_3 & d_1 & d_2 & d_3 & H & \phi \\ \hline 1 & 2 & 0 & -1 & -3 & 1 & -3 & -5 & -4 & 1 & 1 \end{array}\right)$
$\mathcal{O}(1)$ coeff.	$(a_{ij}) \simeq \begin{pmatrix} -0.601 & 1.996 & 0.537 \\ -0.976 & -1.498 & -1.156 \\ 1.513 & 1.565 & 0.982 \end{pmatrix}$ $(b_{ij}) \simeq \begin{pmatrix} 0.740 & -1.581 & -1.664 \\ -1.199 & -1.383 & 0.542 \\ 0.968 & 0.679 & -1.153 \end{pmatrix}$
VEV, value	$v_1 \simeq 0.158\,, \qquad \mathcal{V}(\mathcal{Q}) \simeq -0.621$

—
Table from 2103.04759

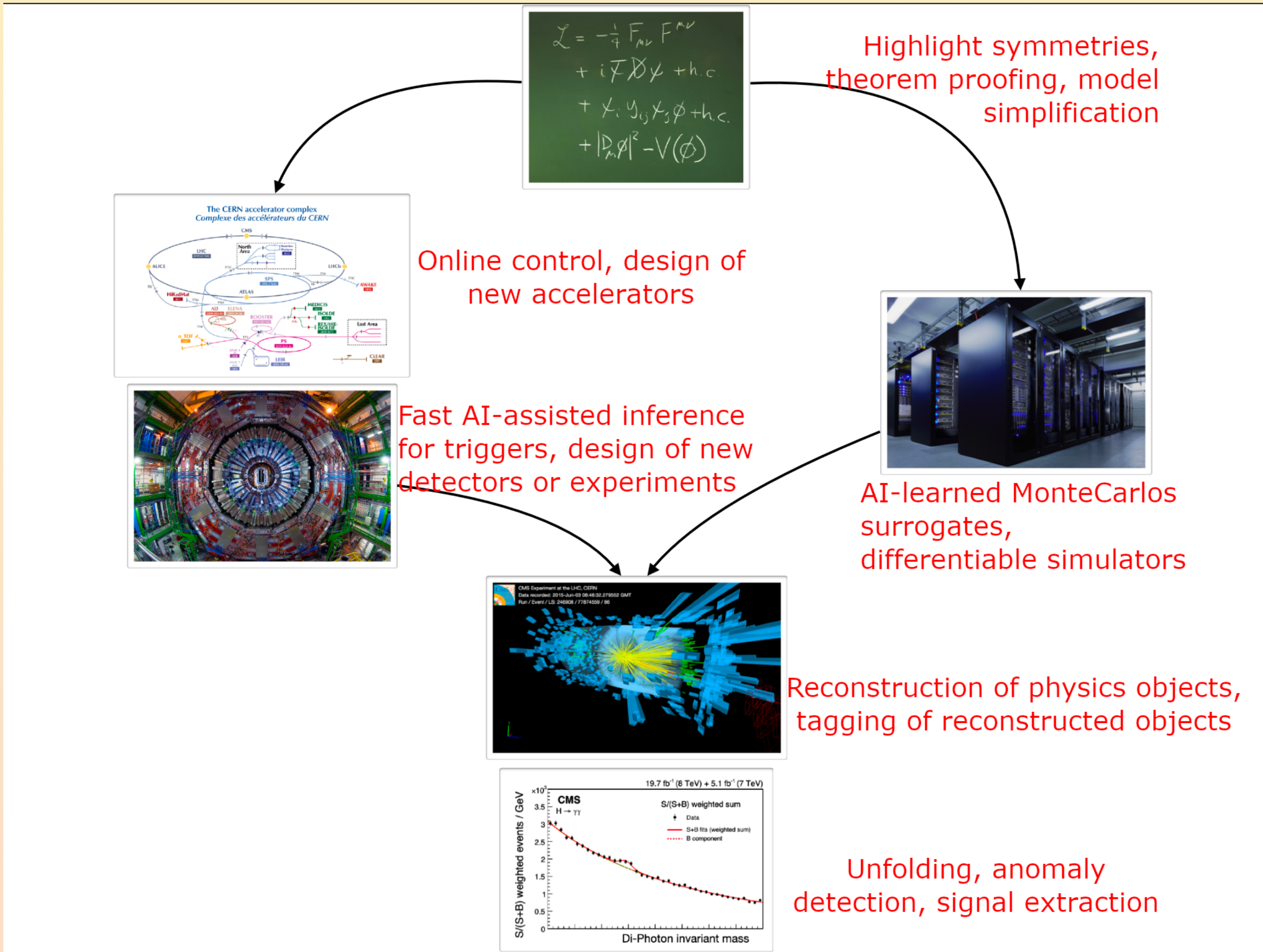
What do we do



—

Illustration reedited from original by Gregor Kasieczka

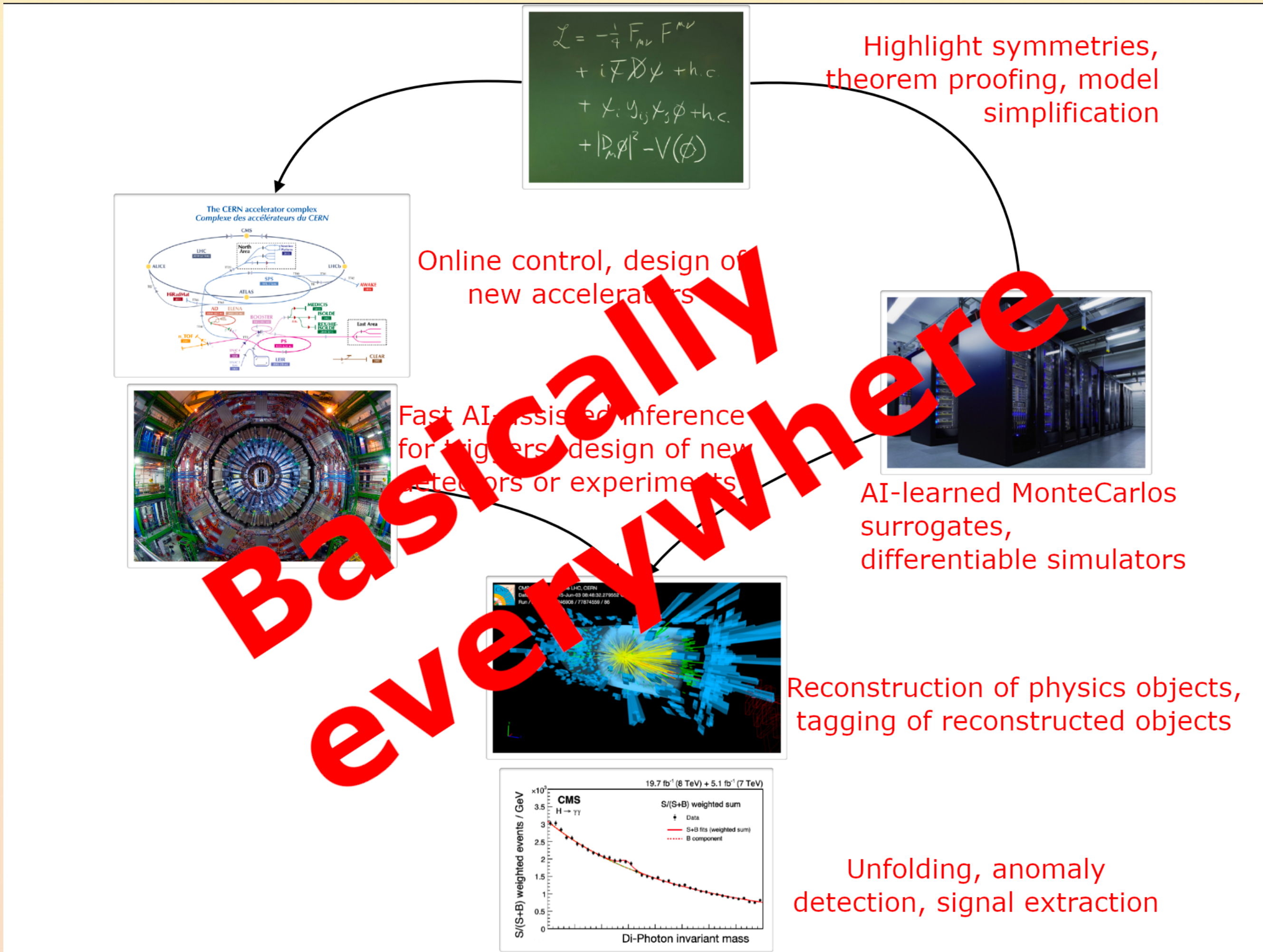
Where we can plug AI



—

Illustration reelaborated with some additions from original by Gregor Kasieczka

Where we can plug AI



Most theory papers are symbolic

- AI-assisted theorem proofing

Lean [de Moura et al., 2015] Coq [Barras et al., 1997] Isabelle [Nipkow et al., 2002]

https://machine-learning-for-theorem-proving.github.io/ (NeurIPS 2023)

- LLMs to solve mathematical problems

Article | Open access | Published: 14 December 2023

Mathematical discoveries from program search with large language models

Bernardino Romera-Paredes, Mohammadamin Barekatain, Alexander Novikov, Matej Balog, M. Pawan

- Simplify polylogarithms (no classical algorithm available, LLMs 91% success!)

Dutch: naamsveranderingsdocumentenbriefgeheel

$$f(x) = 9\left(-\mathrm{Li}_3(x) - \mathrm{Li}_3\left(\frac{2ix}{-i+\sqrt{3}}\right) - \mathrm{Li}_3\left(-\frac{2ix}{i+\sqrt{3}}\right)\right) + 4\left(-\mathrm{Li}_3(x) + \mathrm{Li}_3\left(\frac{x}{x+1}\right) + \mathrm{Li}_3(x+1) - \mathrm{Li}_2(-x)\ln(x+1)\right) - 4\left(\mathrm{Li}_2(x+1)\ln(x+1) + \frac{1}{6}\ln^3(x+1) + \frac{1}{2}\ln(-x)\ln^2(x+1)\right)$$

translate

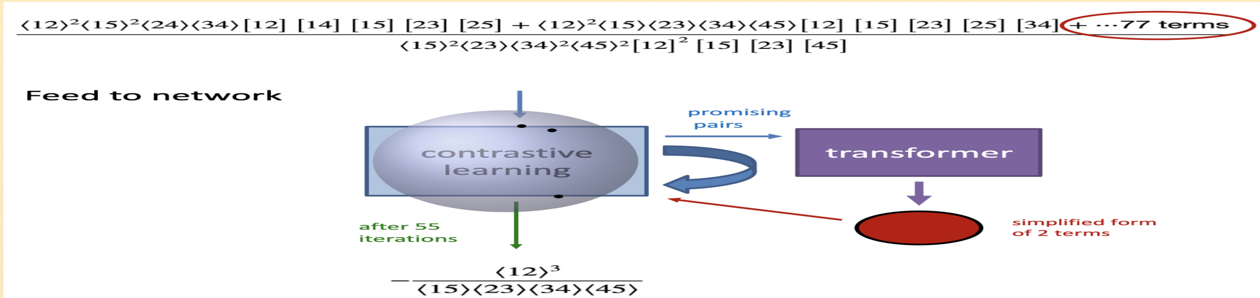
English: dossier

$$f(x) = -\mathrm{Li}_3(x^3) - \mathrm{Li}_3(x^2) + 4\zeta_3$$

—
Examples from Matthew Schwartz

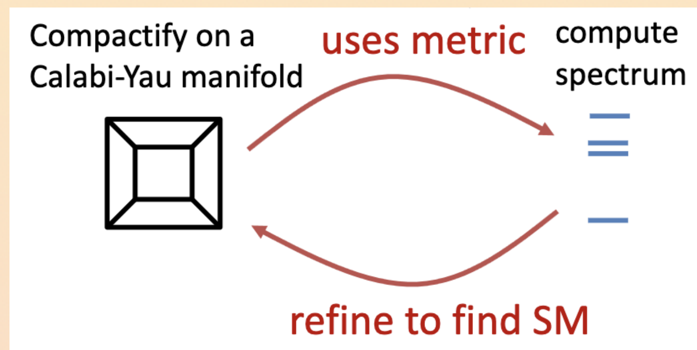
Most theory papers are symbolic

- 5-point MHV amplitude w/ Feynman diagrams: from 1990 tokens to 27 tokens



Solve string theory 😜

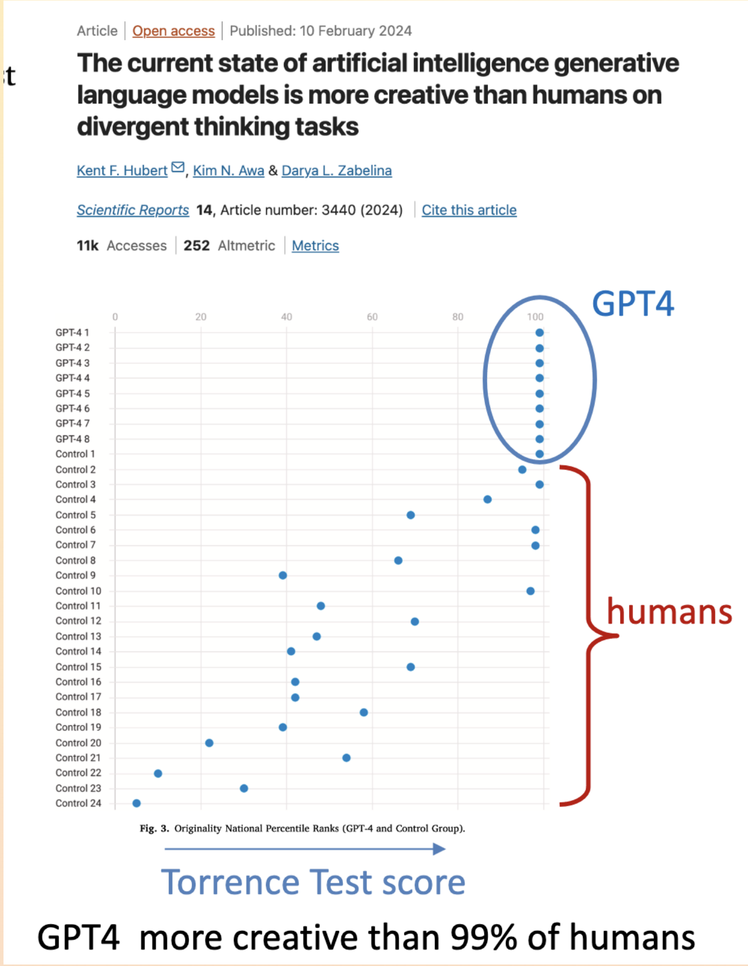
- Find nontrivial Calaby-Yau metrics (1910.08605)
- Look for fixed points of metric flows (2310.19870)
- Predict rank of gauge group (1707.00655, prediction later proven)



—

Beyond symbolic manipulation

- Can AI find interesting questions?
- Can AI models teach themselves to be good physicists using data?
- **If AI understands physics (can calculate everything) but we do not, do we consider it an acceptable "understanding"?**



Accelerate accelerators

- Daily operation and control have huge impact on resources and efficiency
  - Beam scheduling: changing supercycle requires 20-100 clicks (2-25min) about 60 times/day
  - 15% of the yearly cost of SPS fixed target cycle employed for "waste" cycles to mitigate hysteresis problems
- What if we could make them fully automatic (like e.g. Space telescopes)?

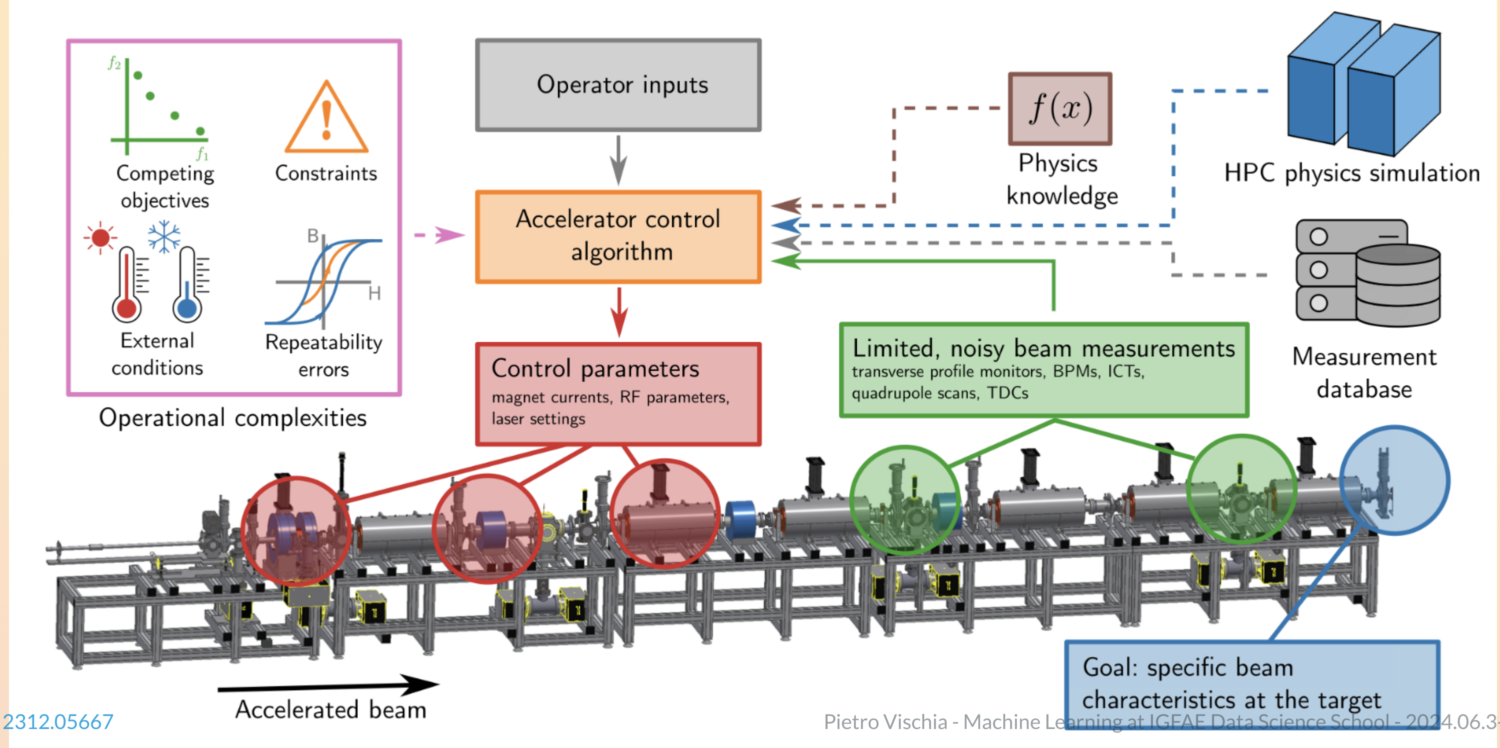
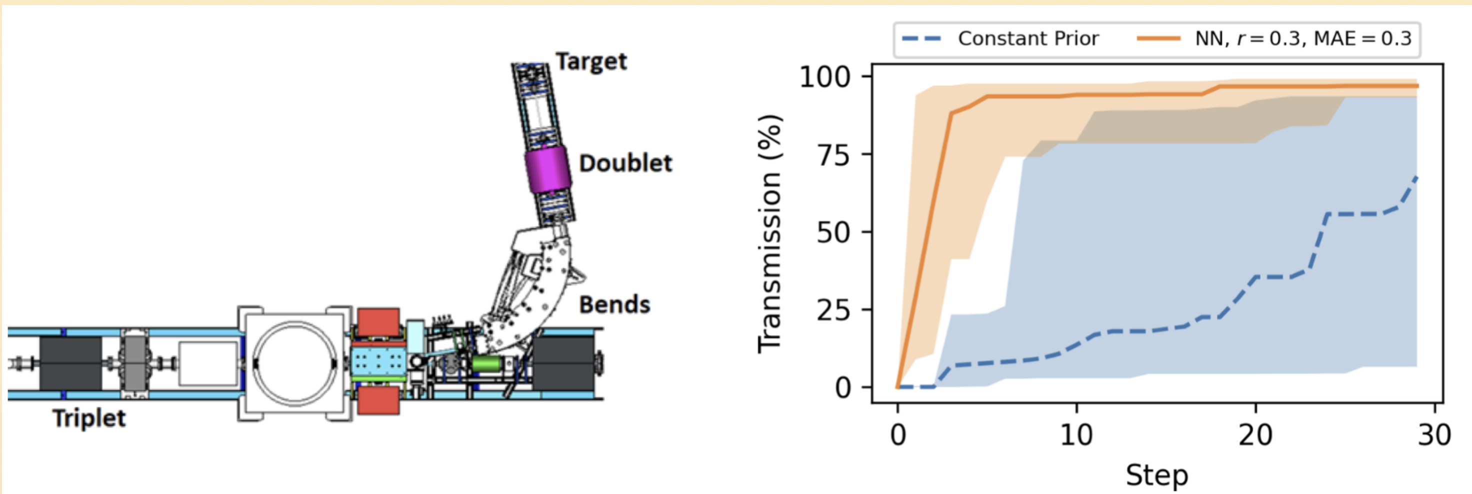


Image from 2312.05667

Accelerate accelerators

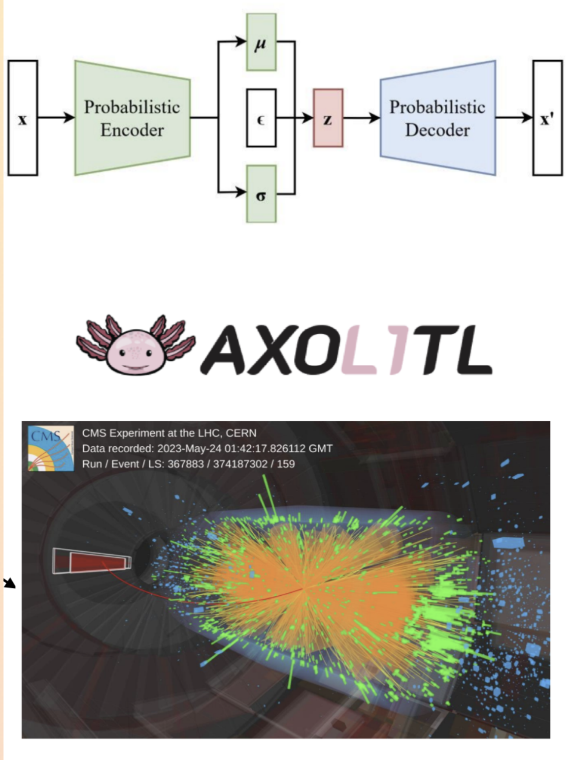
- Hierarchical, AI-controlled autonomous systems
- Optimize trasmission to target in a system with 5 DoF, using Bayesian Optimization

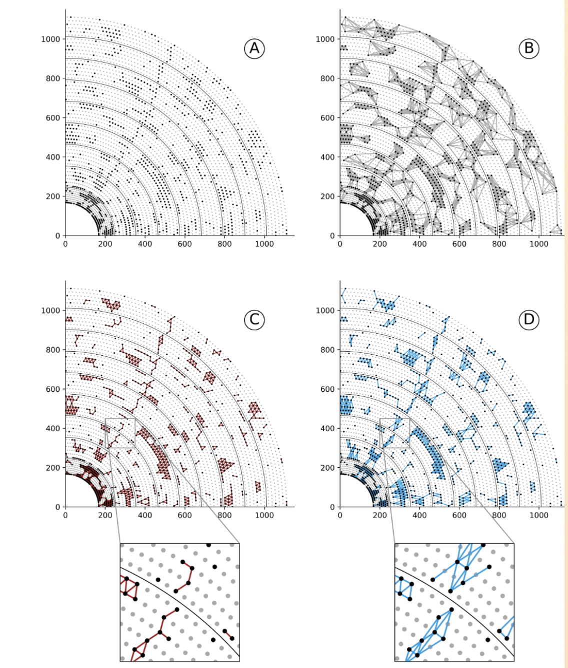


Courtesy T. Boltz et al, arXiv:2403.03225

Trigger

- See talks by A. Zabi and S. Folgueras
- Pack AI models into the L1 trigger ⟶ improve selection criteria
  - At ICTEA!
- Can do e.g. anomaly detection, and online graph building

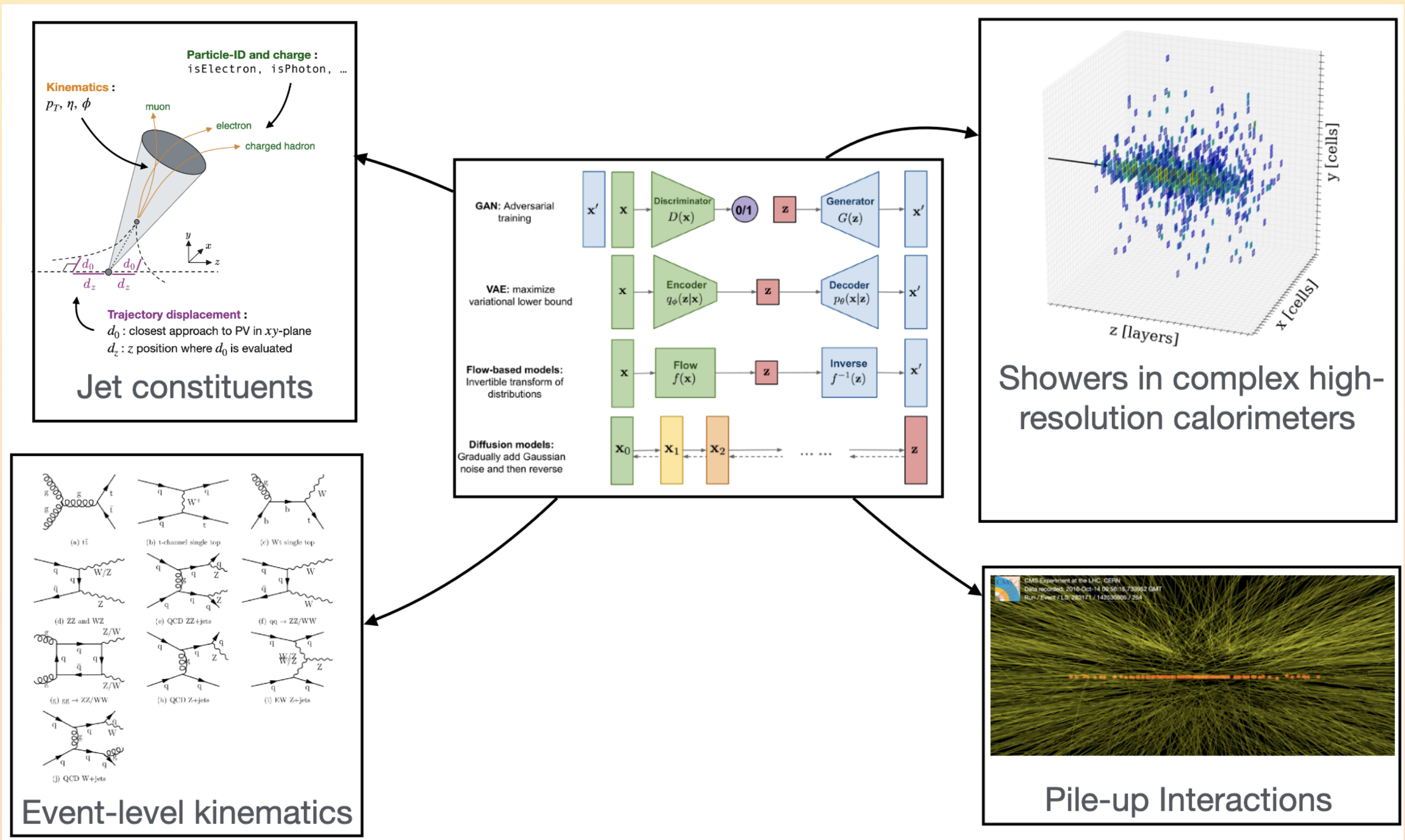




Neu et al 2307.07289

Simulations: the problem

- Monte Carlo simulations are very costly
- The more data we collect, the more simulated events we need

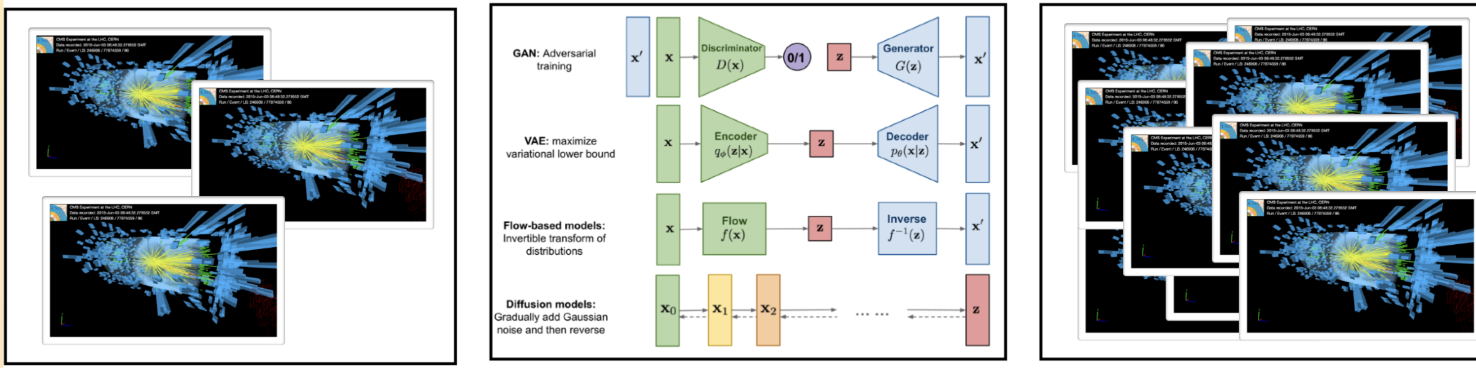


Simulation: two solutions

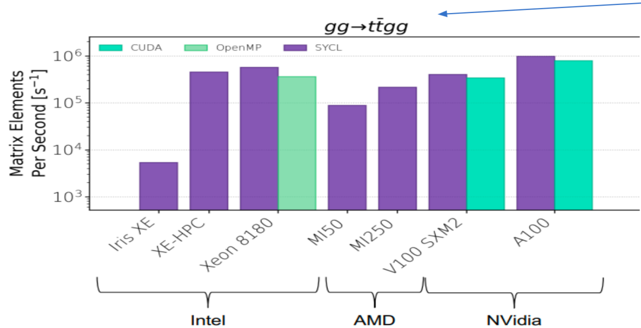
1. Use classical simulation or collider data as input

2. Train generative surrogate

3. Oversample



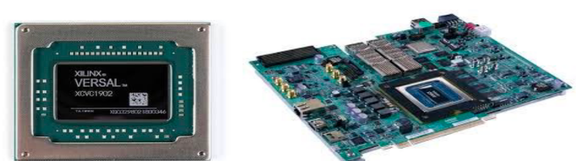
- Very recently, Madgraph5_aMC@NLO authors deployed a version of their code that can run on GPUs.
- This version significantly improves computation times (see this talk).



Talk by C. Vico Villalba

So our idea is: can we do this on hardware based accelerators?

- **FPGAs** are:
  - **Highly** parallelizable
  - In some cases not as fast as GPU.
  - But less power consuming.
  - Hardware based! really versatile.

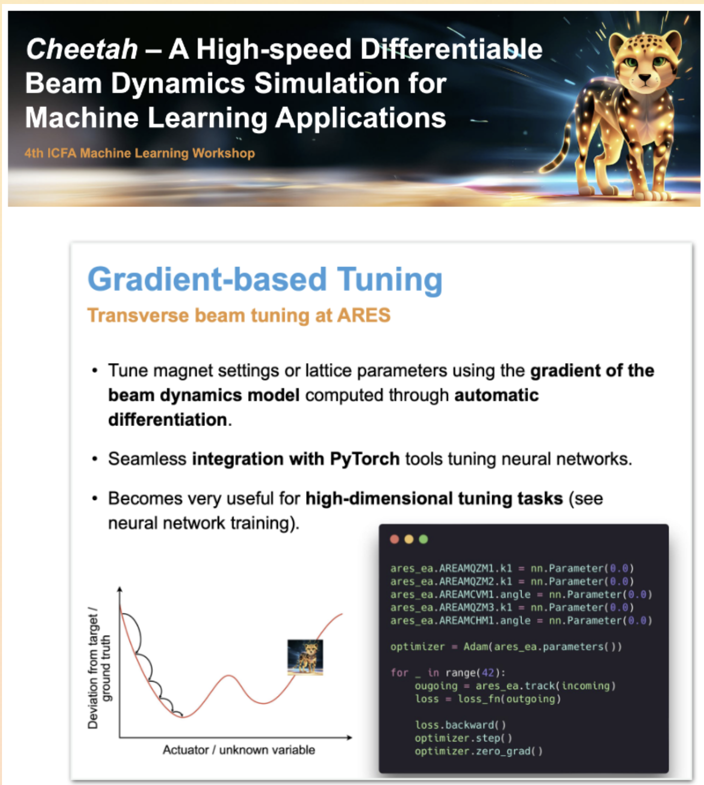


See talk by P. Leguina.

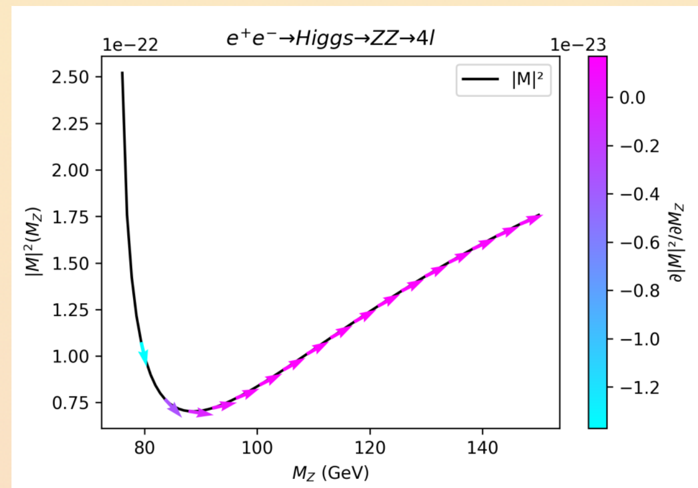
Simulation: long term solution

- Make everything differentiable, exploiting differentiable programming

Cheetah



MadJax
(differentiable matrix element computation)



—
2401.05815, 2203.00057

Towards full differentiability

- Matrix Element: differentiable (MadJax)
- Integration in Madgraph: multi-channel integration speed up (MadNIS)
- Parton shower: mostly differentiable (2208.02274, and recent work by Kagan+Heinrich)
- Detector simulation: GATE/GEANT4 numerically differentiable (in small ranges) (2202.05551)
- Operator overloading for GEANT4 (2405.07944)

AD in GEANT4

- Probabilistic generator $f : \Theta \times \Omega \to Y, \qquad (\theta, \omega) \mapsto y$
- Find optimal inputs θ to maximize output y
    - Need $\frac{\partial y}{\partial \theta}$, pathwise derivative $\frac{\partial}{\partial \theta} f(\theta, \omega)$
        - **Derivgrind**: insert AD logic into the program (a sort of debugger): cannot support tricky cases
        - **CoDiPack**: operator overloading (e.g. replace `double` type): can run out of memory when storing the real-arithmetic evaluation graph (tape)
        - **Clad**: compiler-based source transformation tools: could use smaller tapes, more advanced optimization

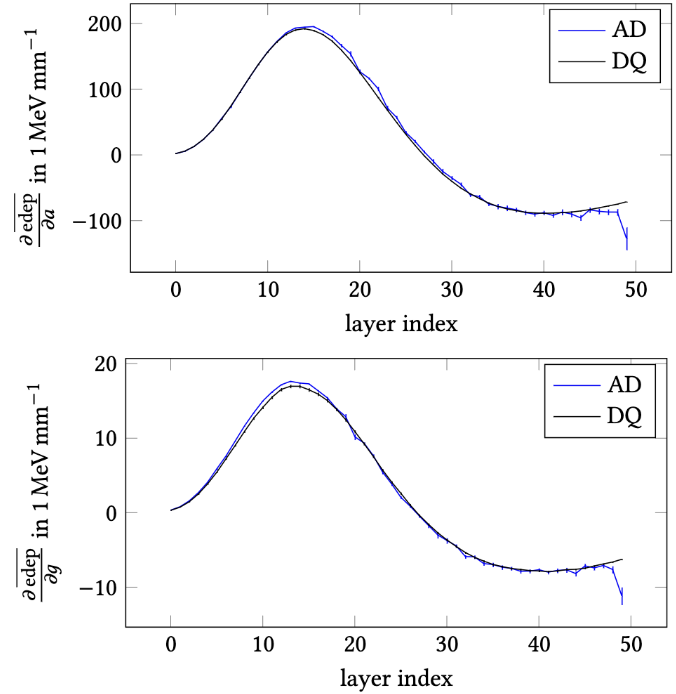
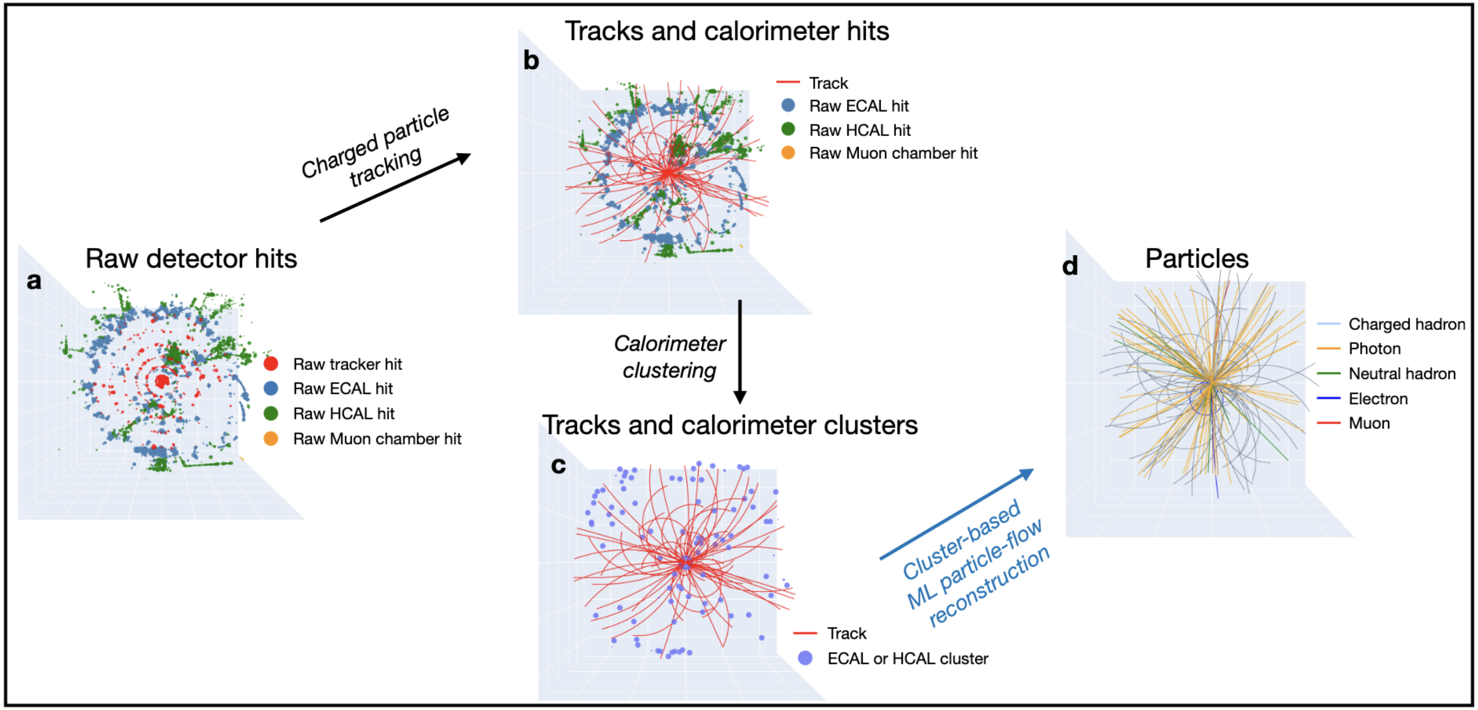


Figure 10: Algorithmic derivative of the edep with respect to the absorber thickness *a* (top) and gap thickness *g* (bottom).

—
Figure from 2405.07944

Reconstruction...

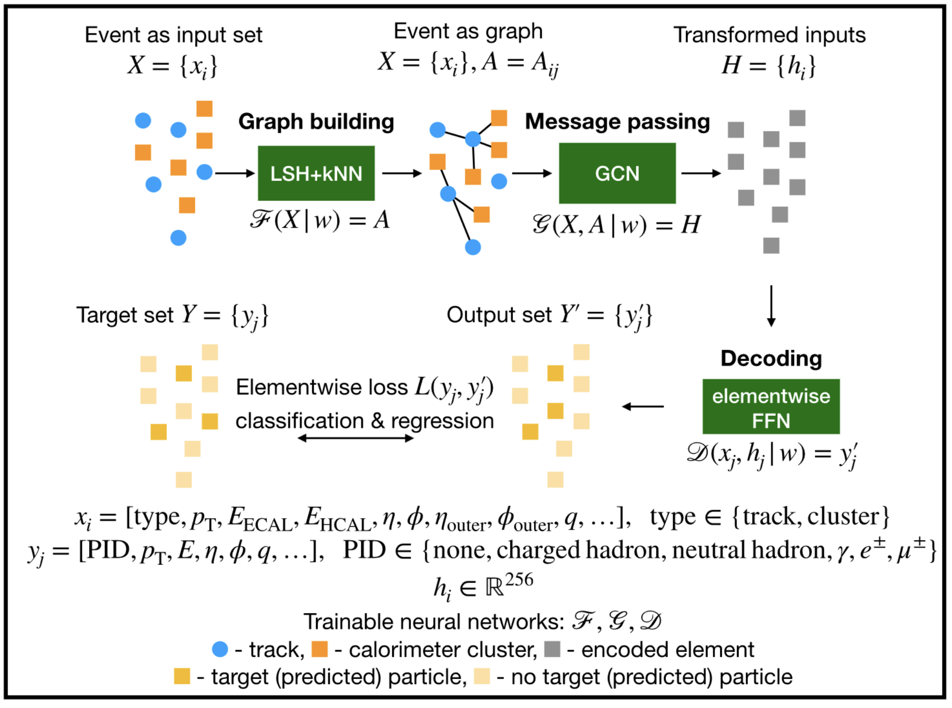


Reconstruction maps low-level detector read-outs

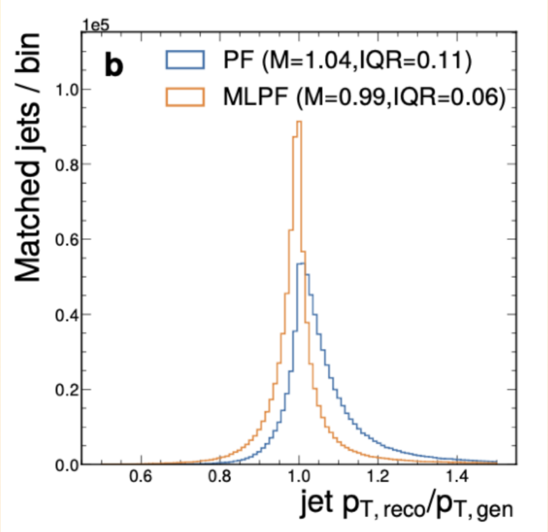
to physical particles

(Which in-turn are the basis of higher-level interpretation)

...with AI

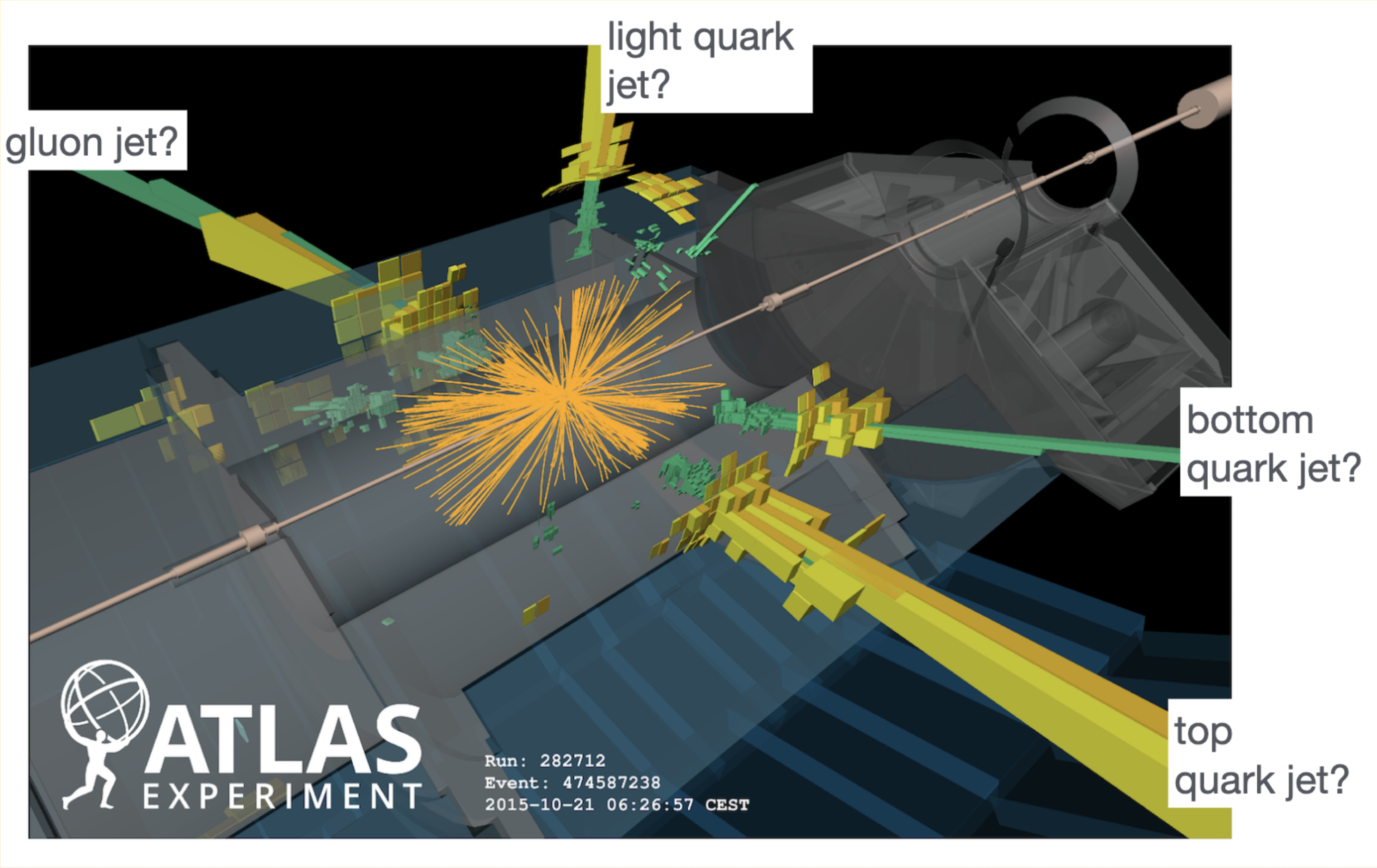


Event as input set $X = \{x_i\}$
Event as graph $X = \{x_i\}, A = A_{ij}$
Transformed inputs $H = \{h_i\}$
Graph building
LSH+kNN
$\mathscr{F}(X|w) = A$
Message passing
GCN
$\mathscr{G}(X, A|w) = H$
Target set $Y = \{y_j\}$
Output set $Y' = \{y'_j\}$
Decoding
elementwise FFN
$\mathscr{D}(x_j, h_j|w) = y'_j$
Elementwise loss $L(y_j, y'_j)$
classification & regression
$x_i = [\text{type}, p_\text{T}, E_\text{ECAL}, E_\text{HCAL}, \eta, \phi, \eta_\text{outer}, \phi_\text{outer}, q, \ldots], \quad \text{type} \in \{\text{track}, \text{cluster}\}$
$y_j = [\text{PID}, p_\text{T}, E, \eta, \phi, q, \ldots], \quad \text{PID} \in \{\text{none}, \text{charged hadron}, \text{neutral hadron}, \gamma, e^\pm, \mu^\pm\}$
$h_i \in \mathbb{R}^{256}$
Trainable neural networks: $\mathscr{F}, \mathscr{G}, \mathscr{D}$
- track, - calorimeter cluster, - encoded element
- target (predicted) particle, - no target (predicted) particle



b
PF (M=1.04,IQR=0.11)
MLPF (M=0.99,IQR=0.06)
Matched jets / bin
1e5
1.0
0.8
0.6
0.4
0.2
0.0
0.6
0.8
1.0
1.2
1.4
jet $p_{T,\text{reco}}/p_{T,\text{gen}}$

Identification...

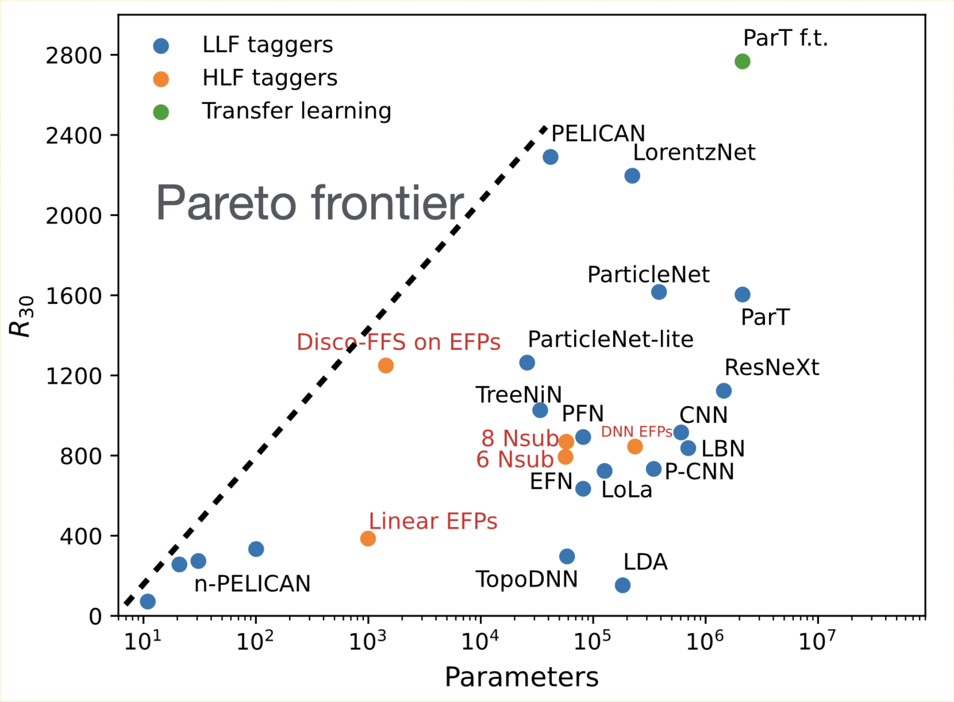


—

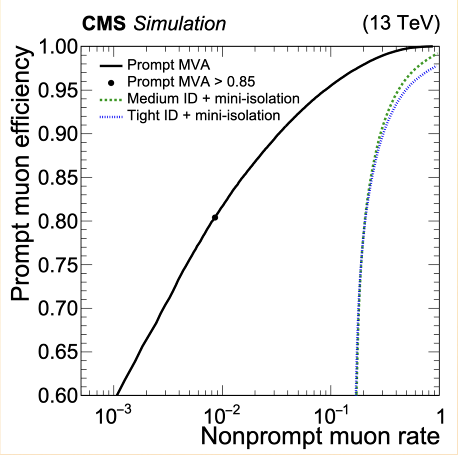
From Gregor Kasieczka's talk at EuCaifCon24

...with AI

Vast landscape of taggers

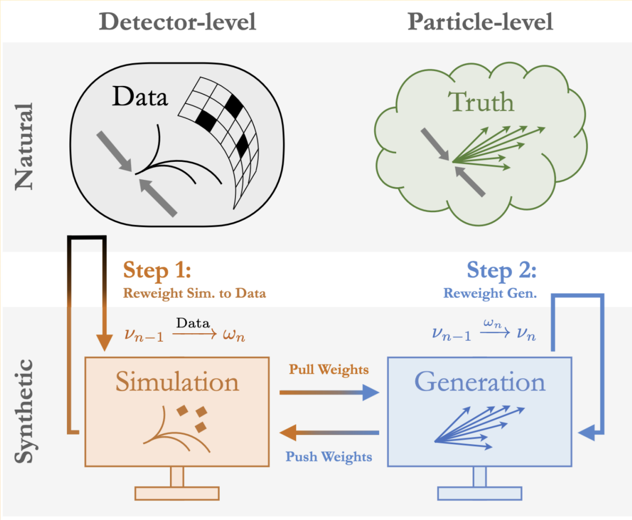


CMS Muon ID: made in ICTEA!

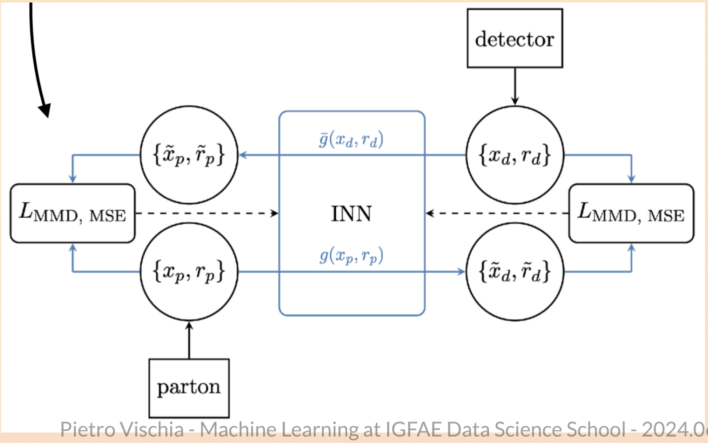


Inference: unfolding

- Use classifiers to learn appropriate weights



- Morph distributions one into the other using diffusion models



—
1911.09107, 2404.XXXXX

Inference: anomaly detection

Gaussian processes)

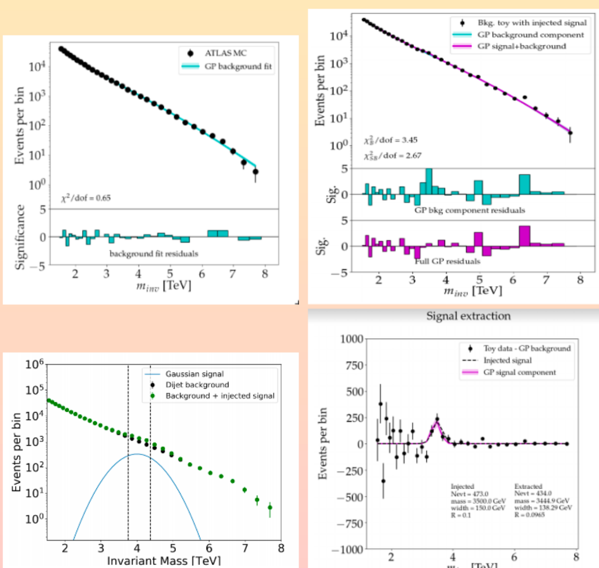
- Multivariate gaussian associated to a set of random variables ($N_{dim} = N_{random\ variables}$)
  - *Kernel* as a similarity measure between bin centers (counts) and a *averaging function*

$$\mu(x) = 0, \tag{9}$$

$$\Sigma_B(x,x') = A\ \exp\left(\frac{d-(x+x')}{2a}\right)\sqrt{\frac{2l(x)l(x')}{l(x)^2+l(x')^2}}\ \exp\left(\frac{-(x-x')^2}{l(x)^2+l(x')^2}\right), \tag{10}$$

$$\Sigma_S(x,x') = C\ \exp\left(-\frac{1}{2}(x-x')^2/k^2\right)\exp\left(-\frac{1}{2}\left((x-m)^2+(x'-m)^2\right)/t^2\right), \tag{11}$$

-
  - Signal is not parameterized
  - Hyperparameters fixed by the B-only fit
- S: residual of B-subtraction

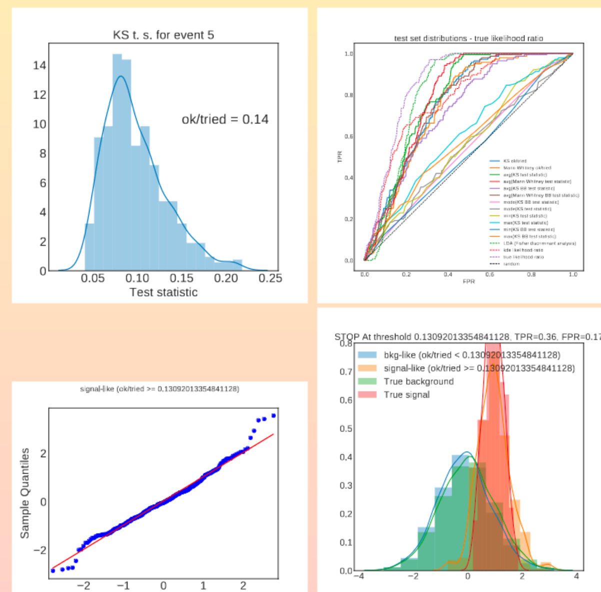


AMVA4NewPhysics deliverable 2.5 public report

Inverse Bagging

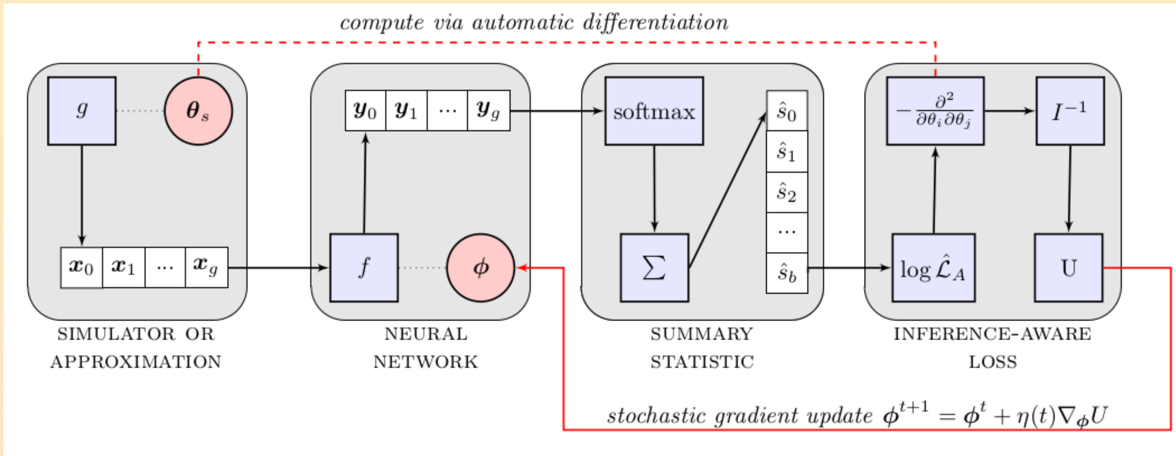
- Data: mixture model with small S
- Classification based on sample properties
  - Compare bootstrapped samples with reference (pure B)
  - Use Metodiev theorem to translate inference into signal fraction
- Validate with LR y LDA
  - Promising results

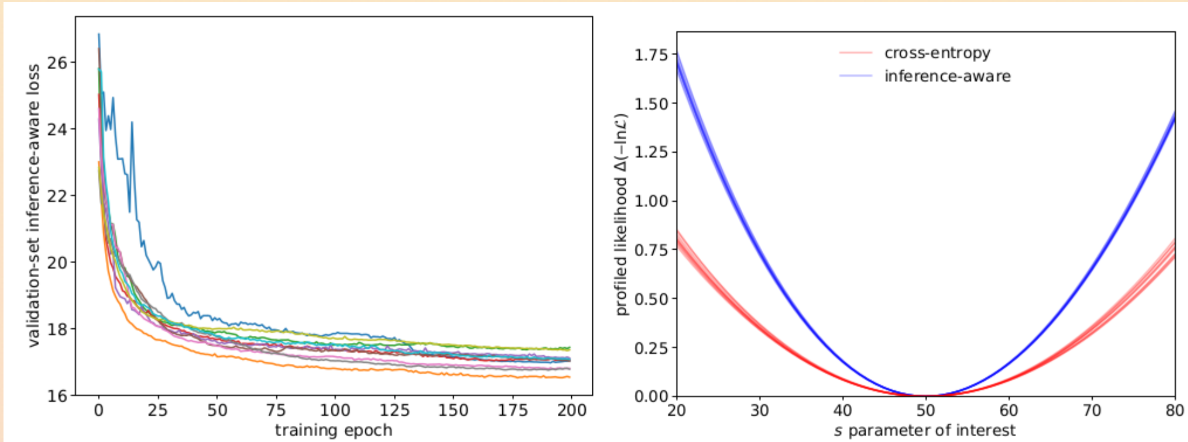
At ICTEA!



Vischia-Dorigo arXiv:1611.08256, doi:10.1051/epjconf/201713711009, and P. Vischia's talk at EMS2019

Go to INFERNO: syst-aware inference opt.





(a) inference-aware training loss

(b) profile-likelihood comparison

Figures from 10.1016/j.cpc.2019.06.007

The likelihood ratio trick

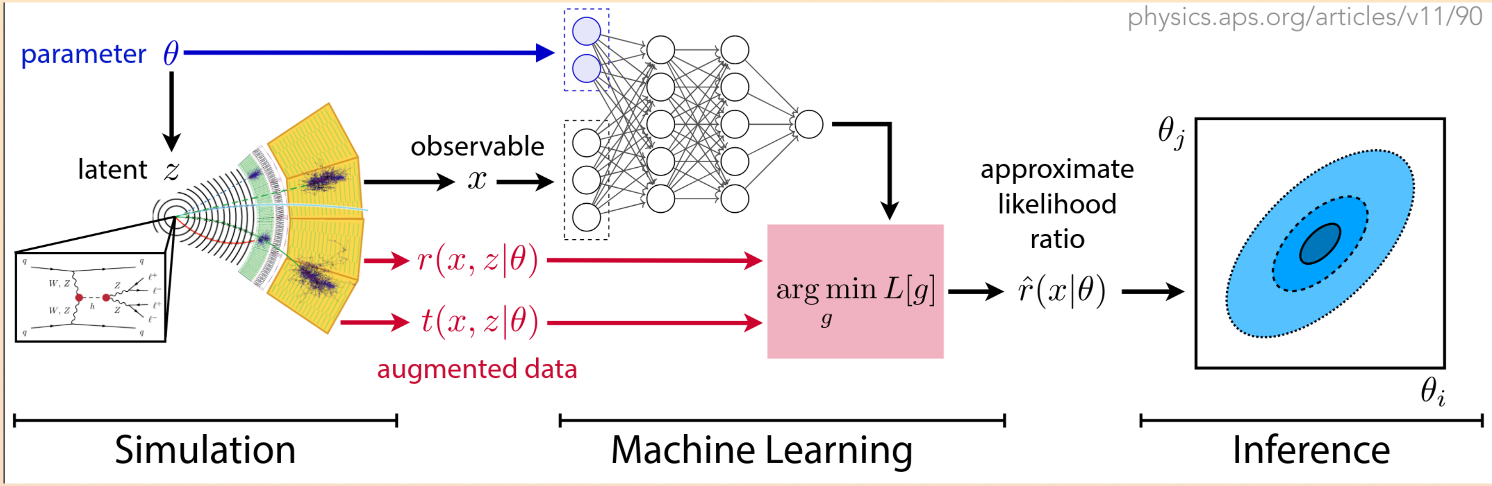
- Train surrogate to discriminate $x_i \sim p(x|\theta_0)$ from $x_i \sim p(x|\theta_1)$
- Cross-entropy

$$L_{\mathrm{XE}} = -\mathbb{E}[1(\theta = \theta_1) \log \hat{s}(x|\theta_0, \theta_1) + 1(\theta = \theta_0) \log(1 - \hat{s}(x|\theta_0, \theta_1))]$$

  - Minimized, $s(x|\theta_0, \theta_1) = p(x|\theta_1)/(p(x|\theta_0) + p(x|\theta_1))$

- Invert, to estimate the likelihood ratio:

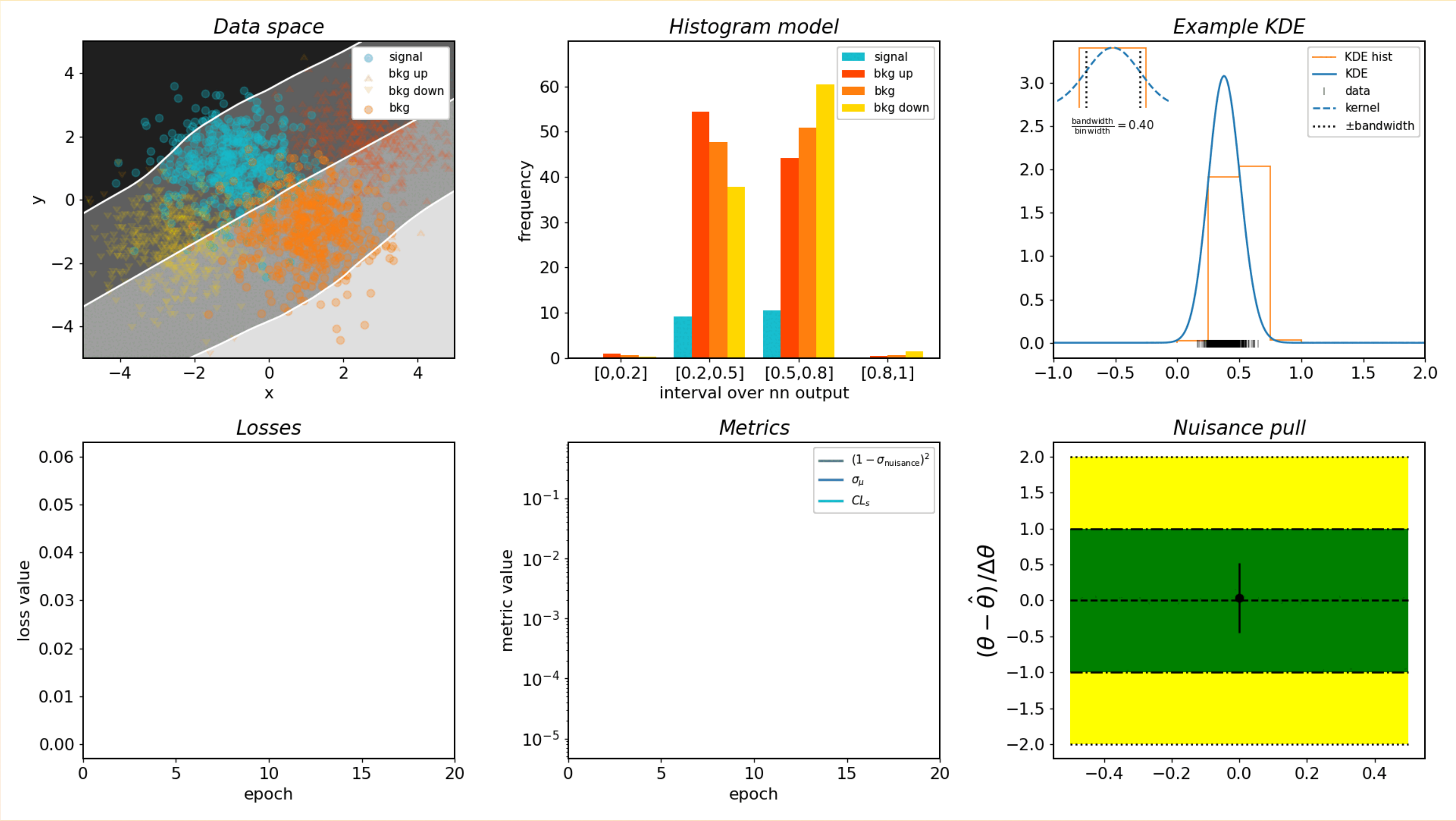
$$\hat{r}(x|\theta_0, \theta_1) = (1 - \hat{s}(x|\theta_0, \theta_1))/\hat{s}(x|\theta_0, \theta_1)$$



—
Figure from 10.1073/pnas.1915980117

Measurement-aware analysis opt.

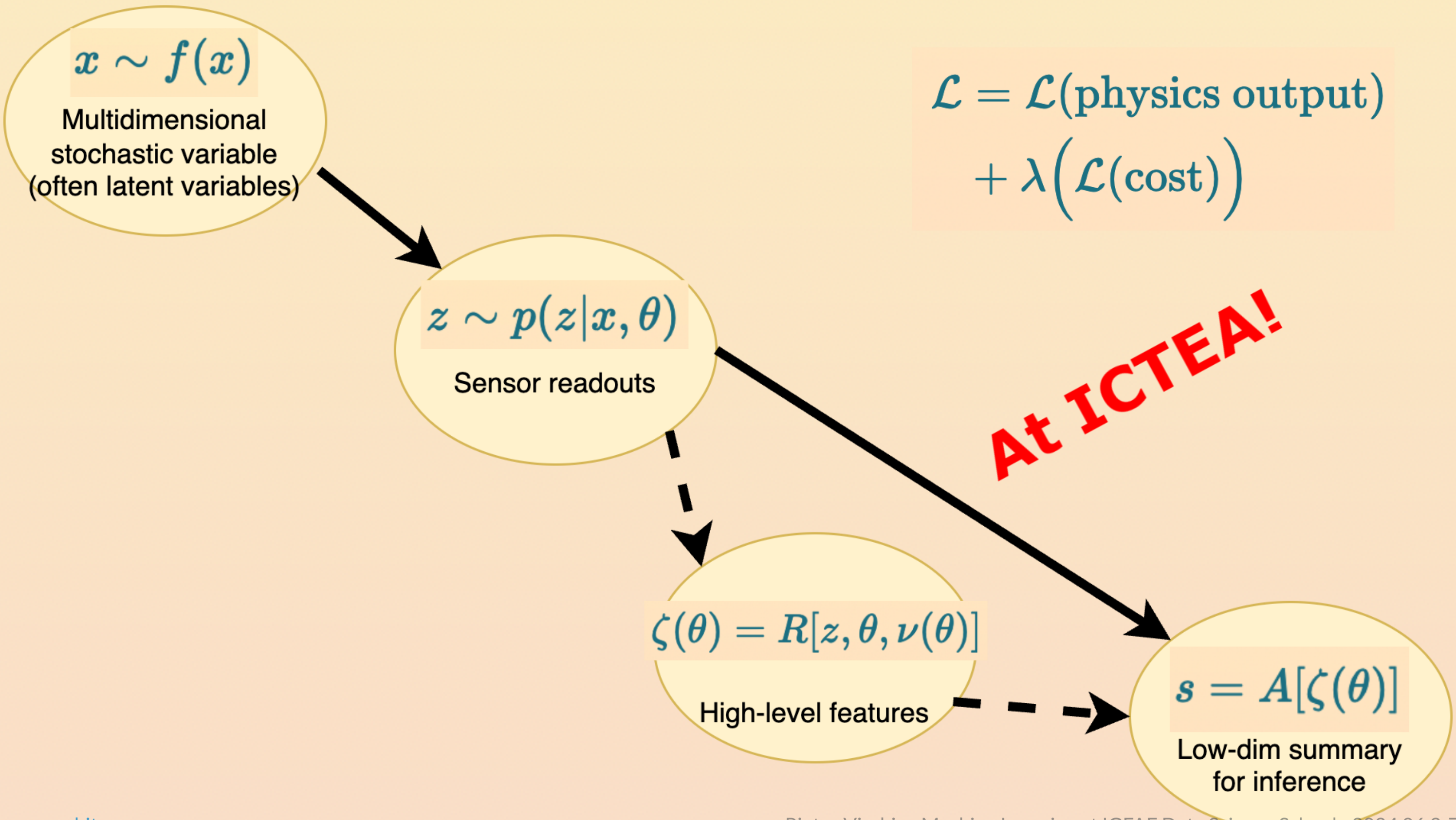
neos



Animation from 2203.05570

Measurement-aware detector opt.!

- Joint optimization of design parameters w.r.t. inference made with data
- MODE White Paper, 10.1016/j.revip.2023.100085 (2203.13818), 117-pages document, physicists + computer scientists



—
Formulas from our white paper

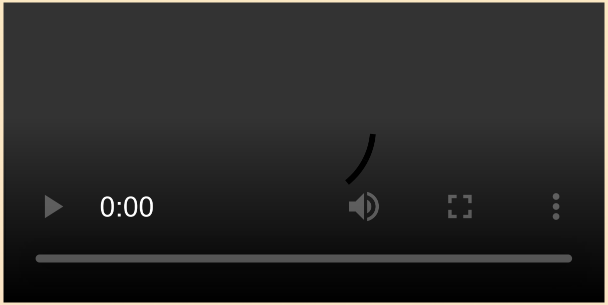
Prototype for muon tomography

TomOpt: Differential optimisation for task- and constraint-aware design of particle detectors in the context of muon tomography

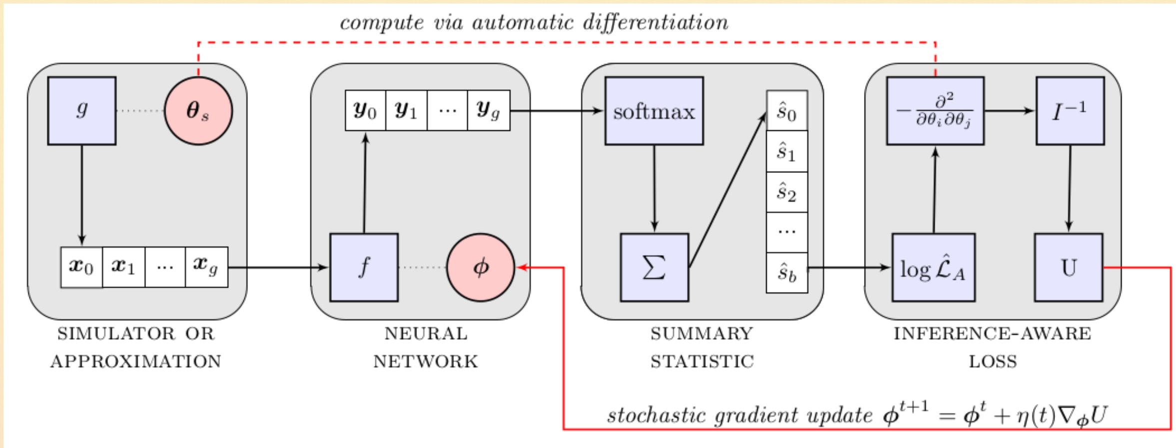
Giles C. Strong, Maxime Lagrange, Aitor Orio, Anna Bordignon, Florian Bury, Tommaso Dorigo, Andrea Giammanco, Mariam Heikal, Jan Kieseler, Max Lamparth, Pablo Martínez Ruíz del Árbol, Federico Nardi, Pietro Vischia, Haitham Zaraket

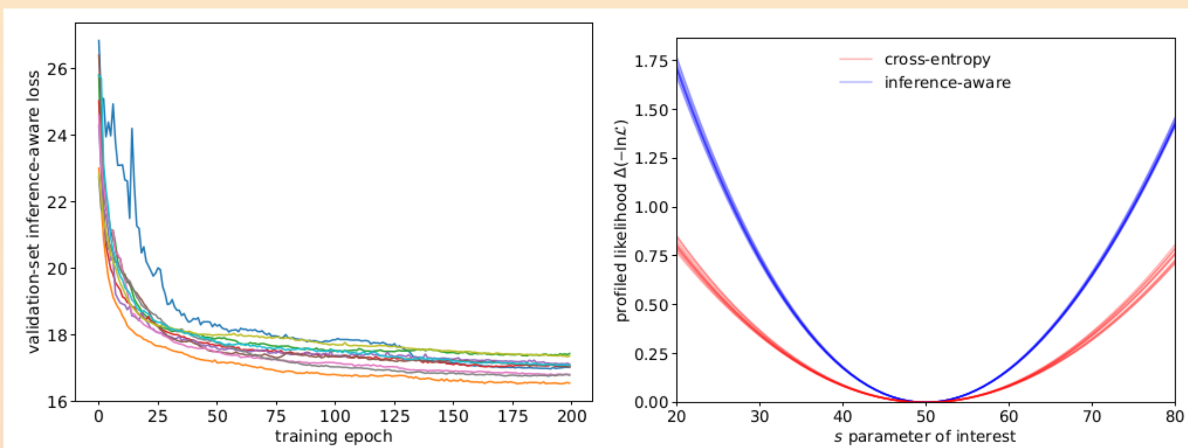
We describe a software package, TomOpt, developed to optimise the geometrical layout and specifications of detectors designed for tomography by scattering of cosmic-ray muons. The software exploits differentiable programming for the modeling of muon interactions with detectors and scanned volumes, the inference of volume properties, and the optimisation cycle performing the loss minimisation. In doing so, we provide the first demonstration of end-to-end-differentiable and inference-aware optimisation of particle physics instruments. We study the performance of the software on a relevant benchmark scenarios and discuss its potential applications.

At ICTEA!



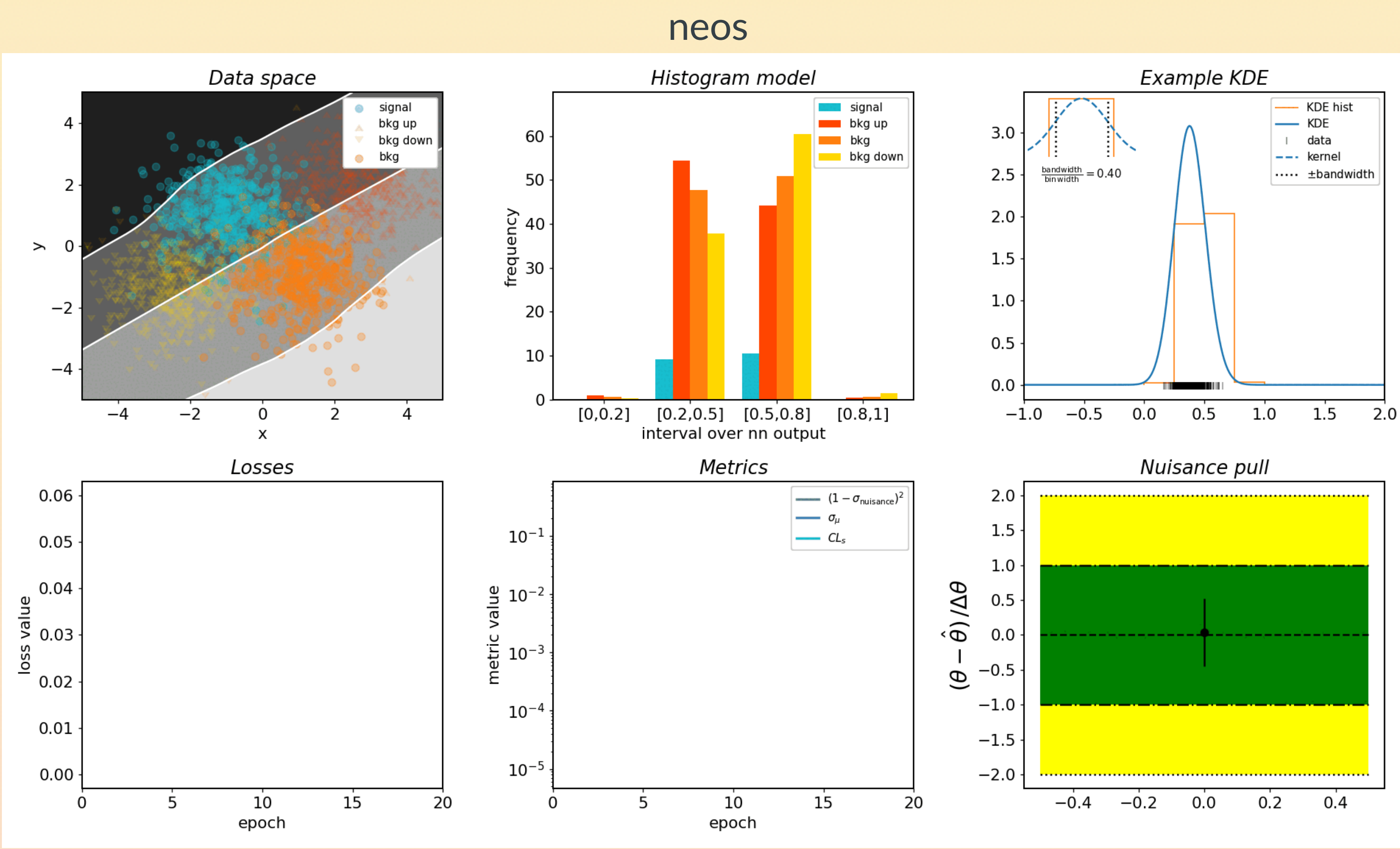
Go to INFERNO: syst-aware inference opt.





(a) inference-aware training loss (b) profile-likelihood comparison

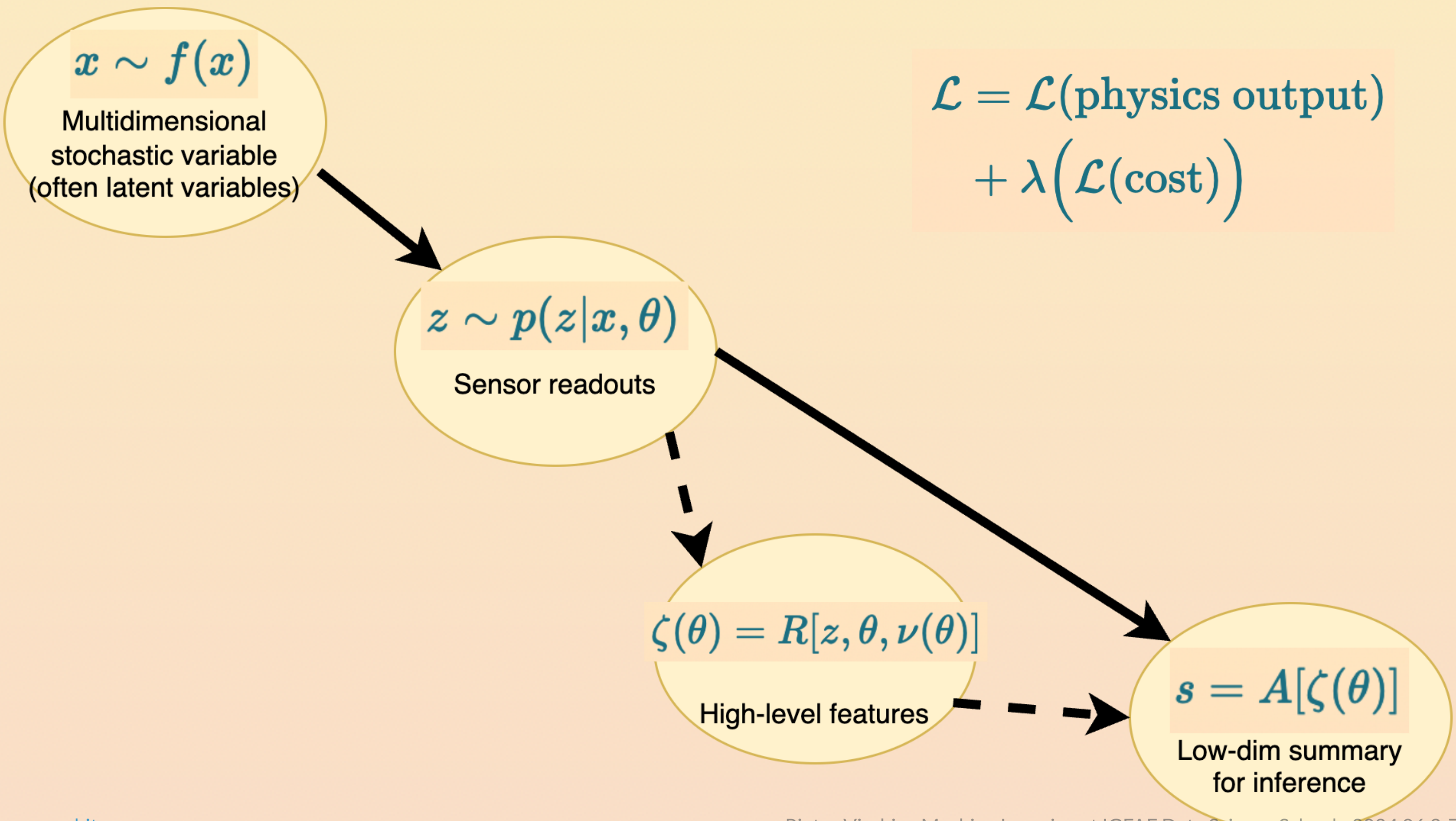
Measurement-aware analysis opt.



Measurement-aware detector opt.!

- Joint optimization of design parameters w.r.t. inference made with data
- MODE White Paper, 10.1016/j.revip.2023.100085 (2203.13818), 117-page document, physicists + computer scientists

$$\mathcal{L} = \mathcal{L}(\text{physics output}) + \lambda\Big(\mathcal{L}(\text{cost})\Big)$$



—
Formulas from our white paper

Guarantee feasibility within constraints

- Monetary cost
- Case-specific technical constraints

$$\mathcal{L}_{\mathrm{cost}} = c(\theta, \phi)$$

- θ: local, specific to the technology used (e.g. active components material)
- ϕ: global, describing overall detector conception (e.g. number, size, position of detector modules)
- Fixed costs can be added separately to the loss function

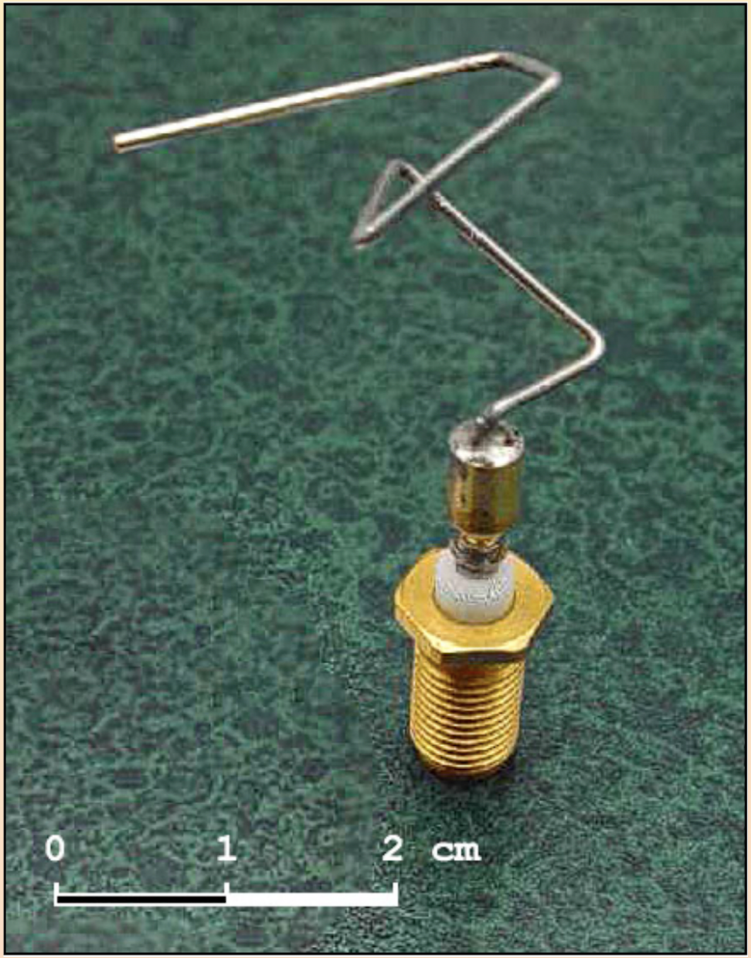
In general

$$\hat{\theta} = arg\ min_\theta \int \underbrace{L[\overbrace{A(\zeta)}^{\text{Depends on z and nuisances}}, \overbrace{c(\theta)}^{\text{Cost of the layout with parameters theta}}]}_{\text{Weight desirable goals while obeying cost constraints}} \overbrace{p(z|x, \theta)}^{\text{Closed form}} f(x) dx dz\ ,$$

—

From our white paper and internal TomOpt presentation

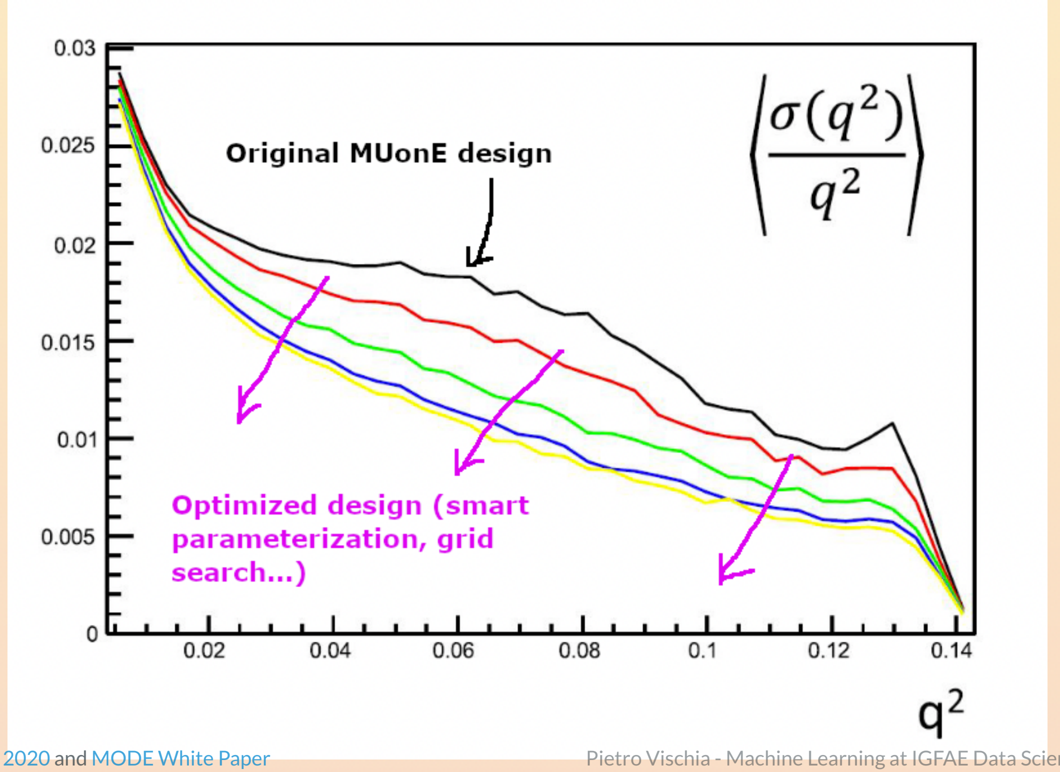
Thrive in asymmetries



—
Photo from doi:10.2514/6.2006-7242

Large gains to be had

- MUonE: proposed 150 GeV muon beam experiment to be built at CERN
  - Measure precisely the q^2 differential cross section in electron-muon scattering
  - 40 tracking stations and a calorimeter
- Dramatic improvement in the resolution on q^2 even from a simple grid search

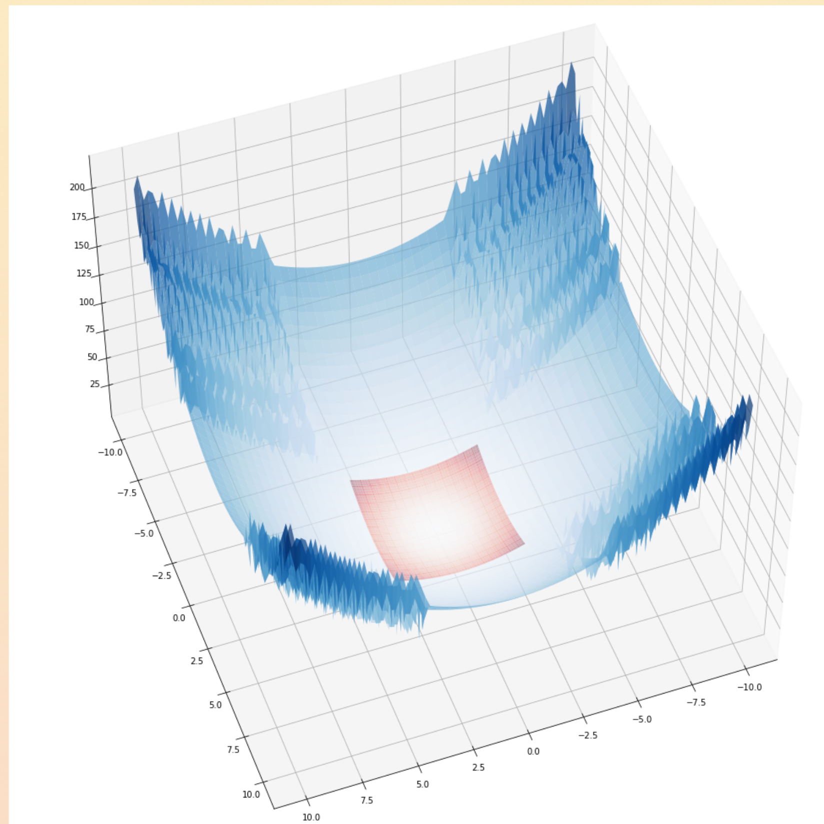


—

Image from Physics Open, 4:100022, 2020 and MODE White Paper

Assist the physicist with a landscape of solutions

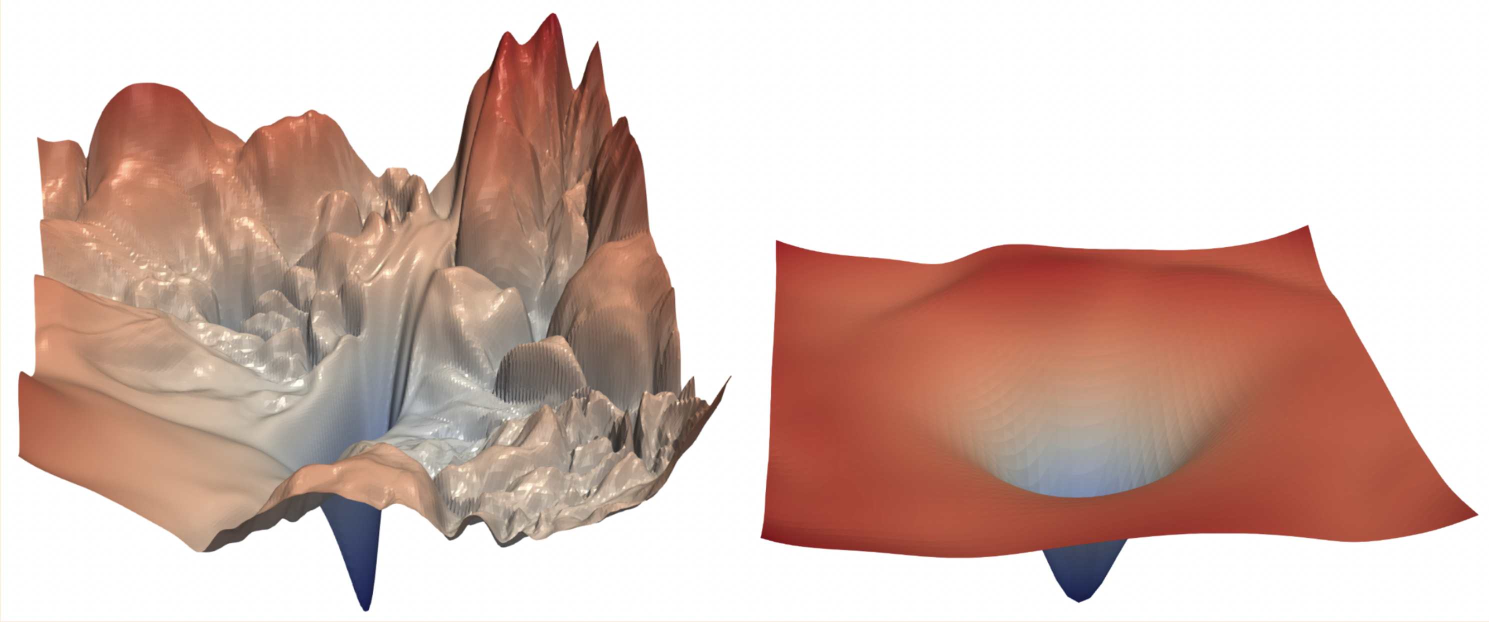
- Cannot parameterize everything
- The optimal solution: unrealistic
- Provide feasible solutions near optimality
- The physicist will fine tune



—
Illustration (c) P. Vischia, book in preparation

How far from optimality?

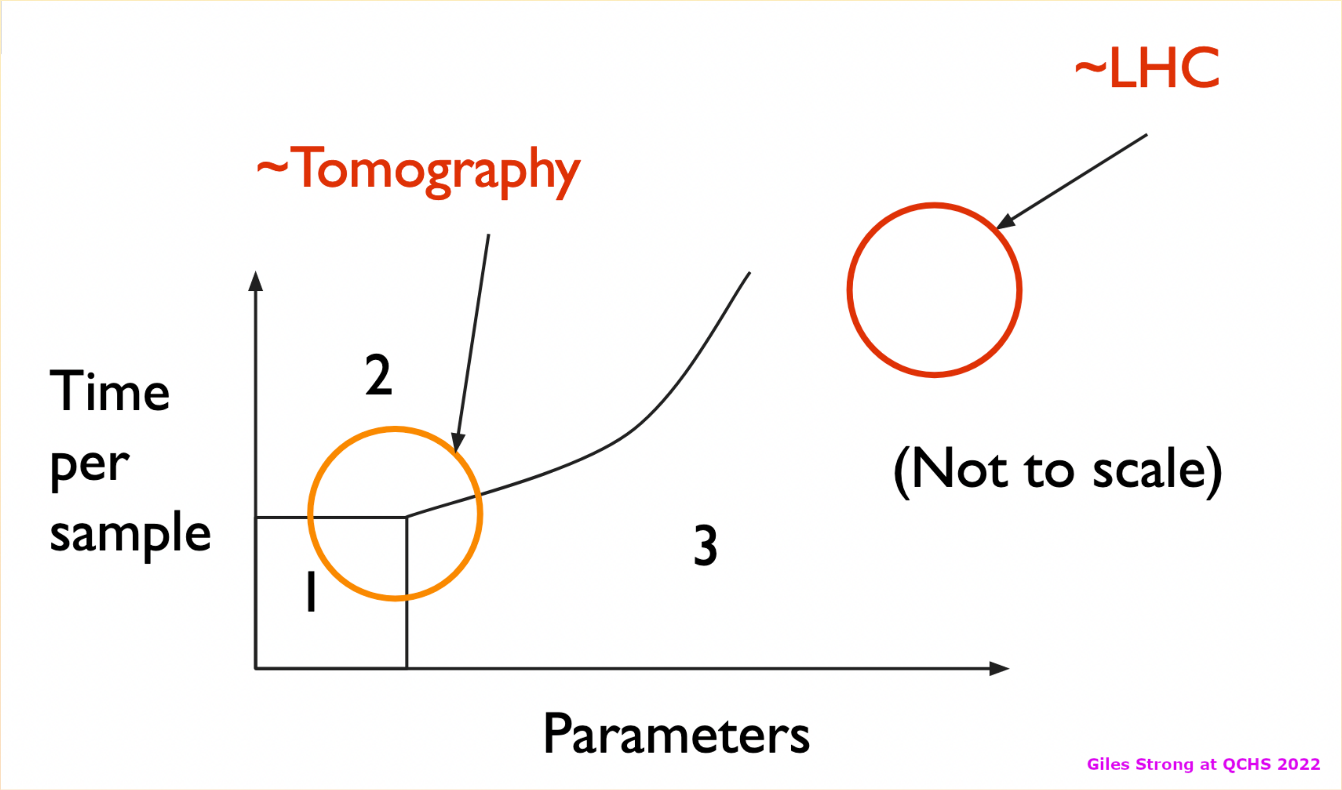
- Can we define in a general way an acceptable increase in loss?
    - Tradeoff performance/cost
- For sure we can regularize the loss landscape to select our scale of interest



—

Image from Li et al (taken as pictorial representation out of its original context)

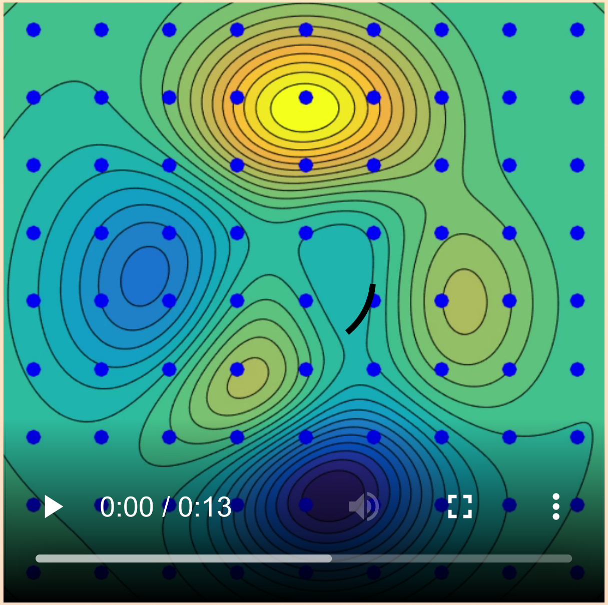
Method of choice depends on scale



1. Grid/random search
2. Bayesian opt, simulated annealing, genetic algos, ...
3. Gradient-based optimization (Newton, BFGS, gradient descent, ...)

Experimental design: present and future

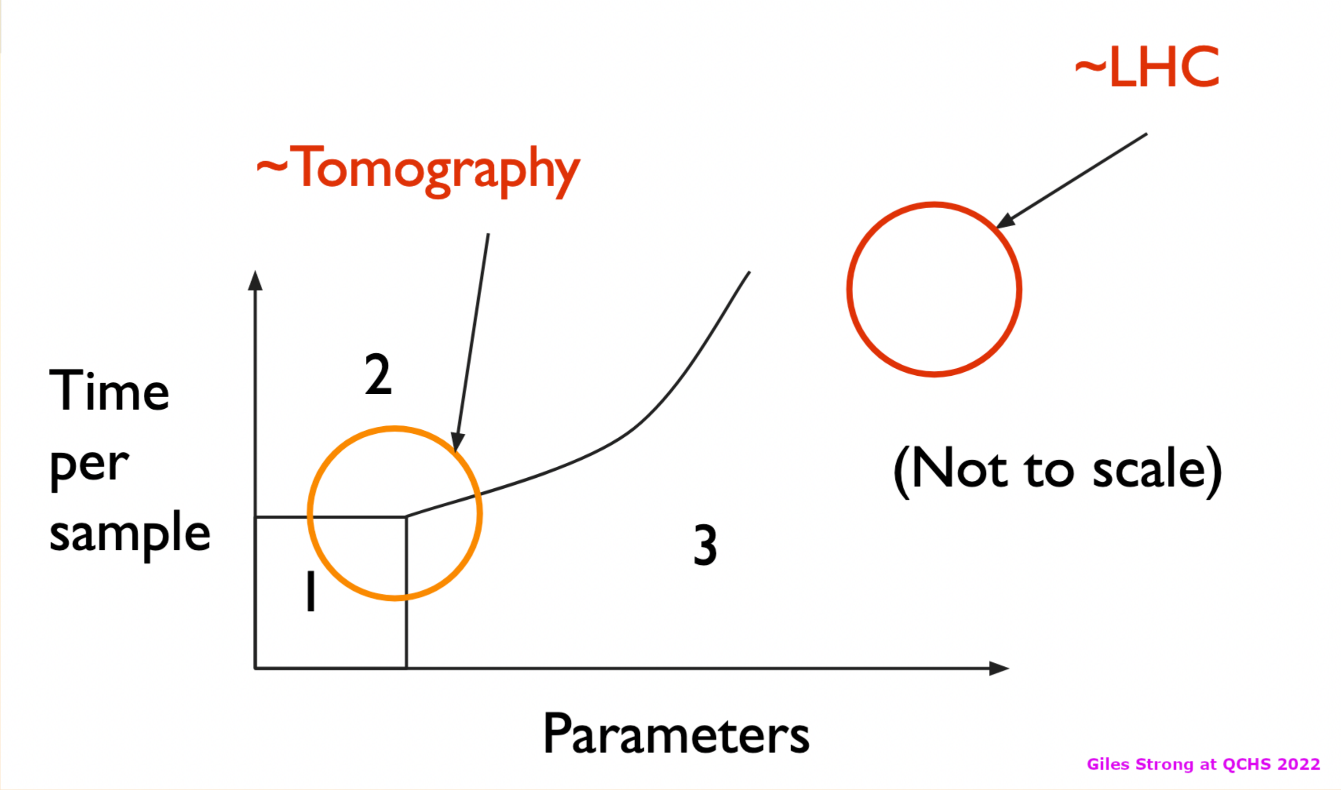
- Gradient descent applied to experiment design works!!!
  - Discreteness and stochasticity mostly solvable or avoidable
- What now?



—

Figure by Gpeyre on wikipedia

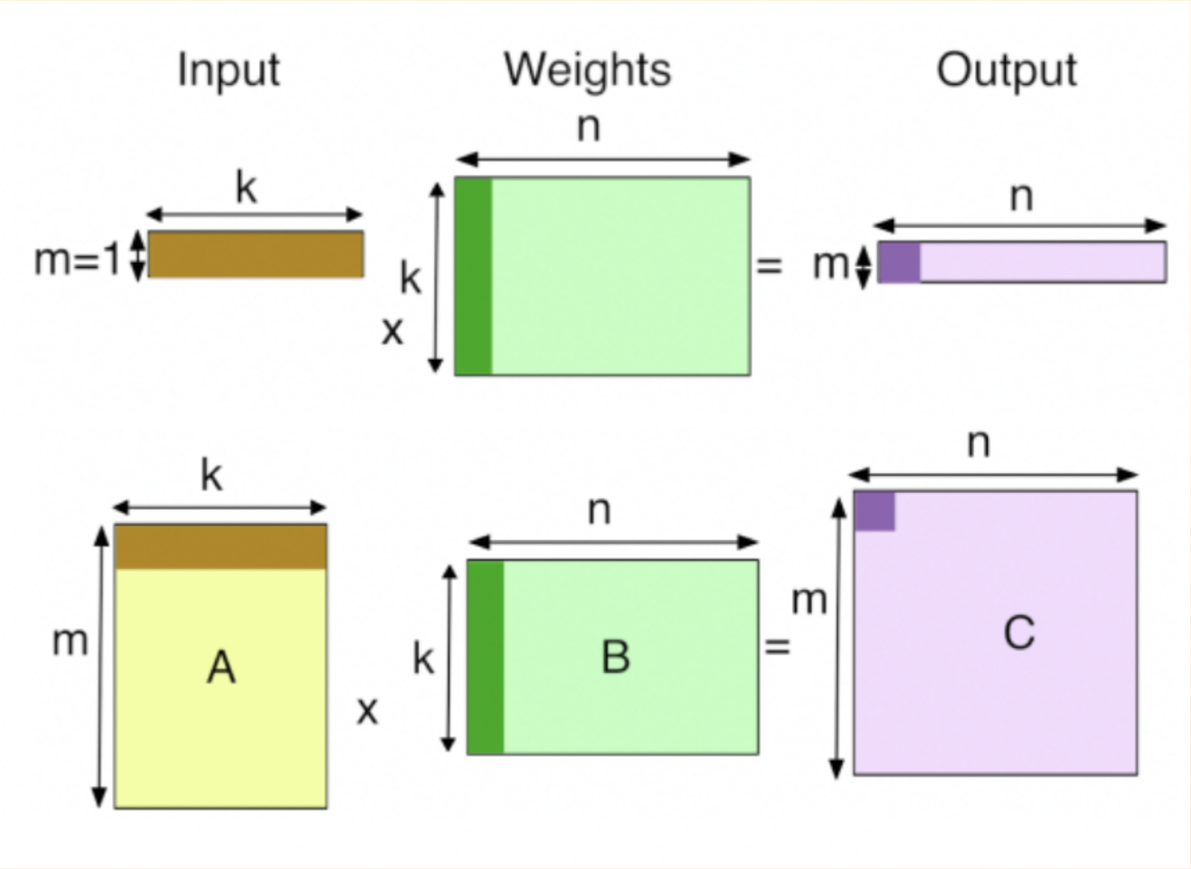
Method of choice depends on scale



1. Grid/random search
2. Bayesian opt, simulated annealing, genetic algos, ...
3. Gradient-based optimization (Newton, BFGS, gradient descent, ...)

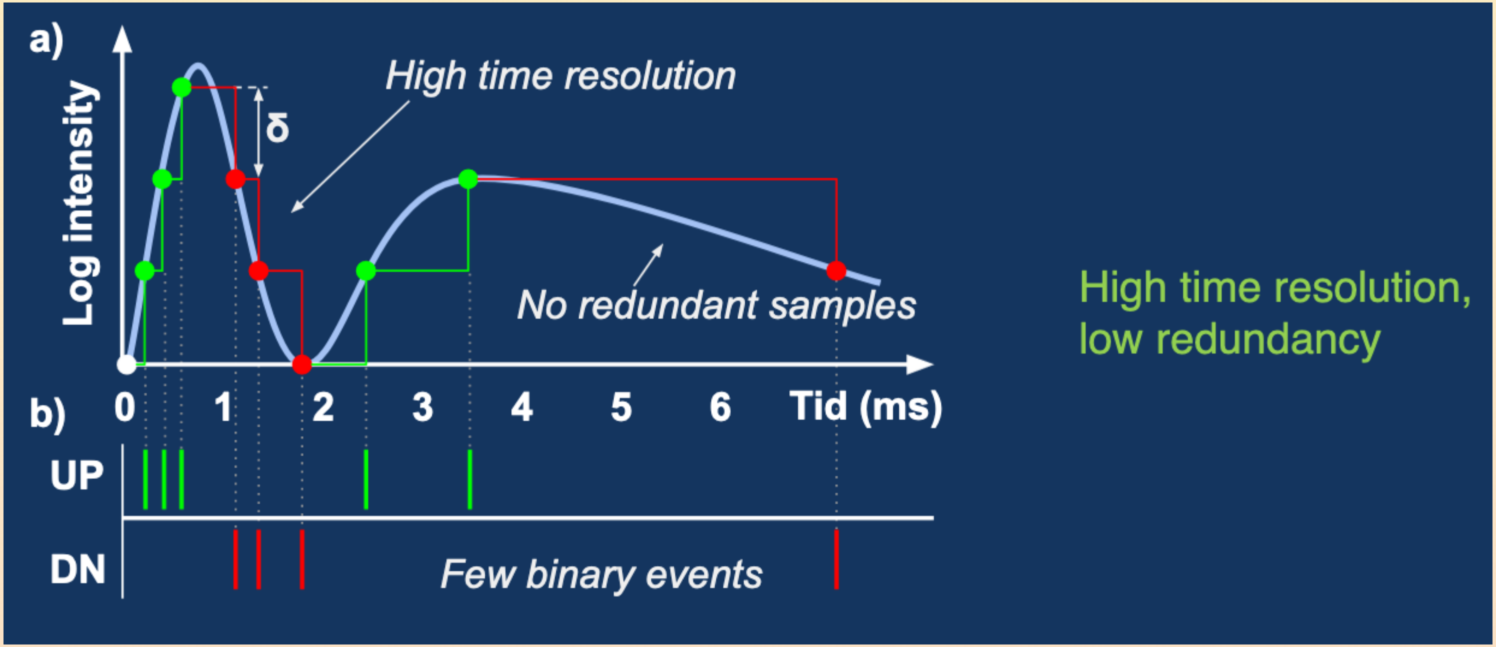
From perceptron-based networks...

- Matrix multiplication



...to spiking neural networks

- Event-driven computations
  - "when a spike occurs, compute something"



—

Figures by Fredrik Sandin

The energy advantage

- Perceptron-based networks: matrix multiplication
    - Sparsity doesn't affect much the throughput and energy consumption
- Spiking neural networks: event-driven computations
    - Sparser inputs require less computations, therefore less time and energy

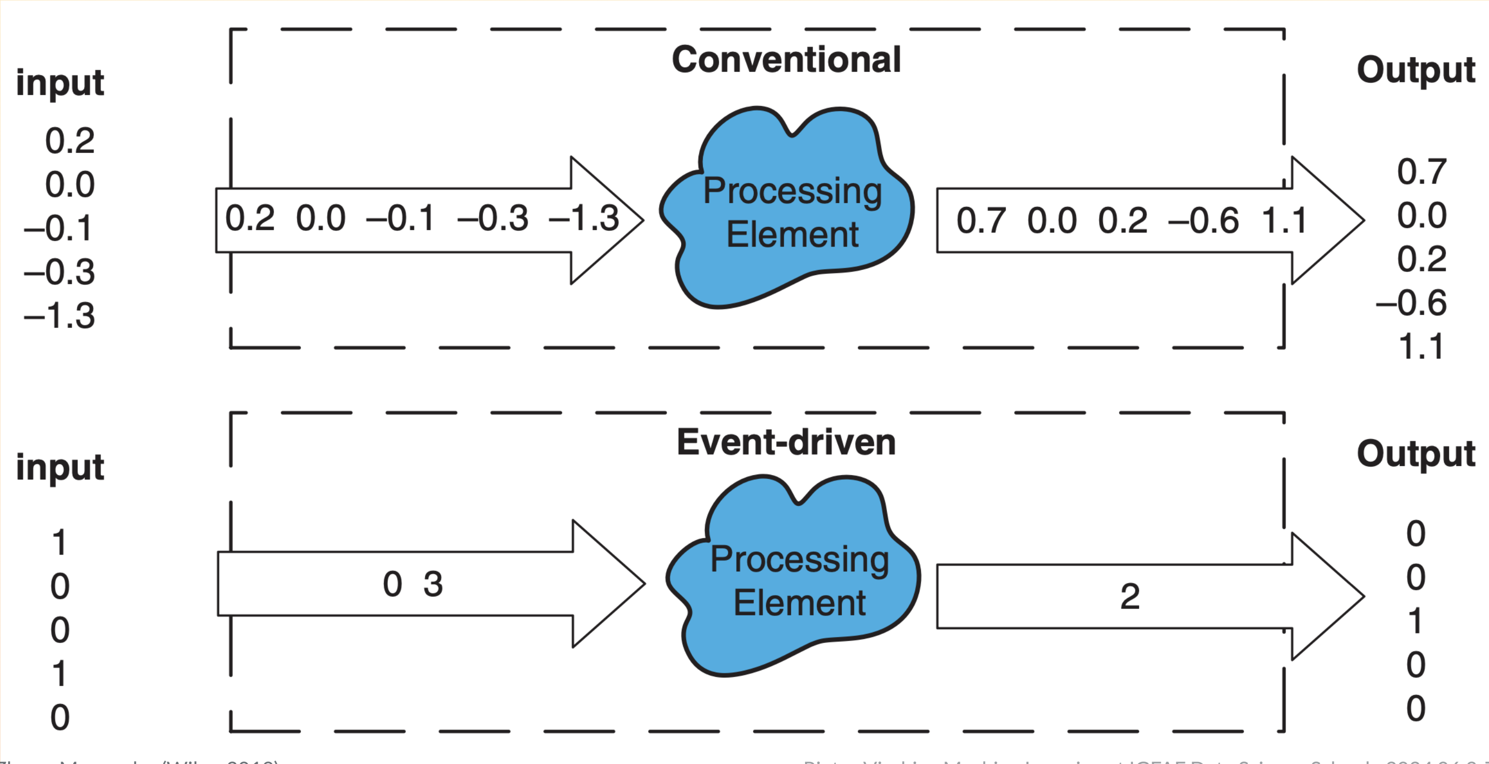
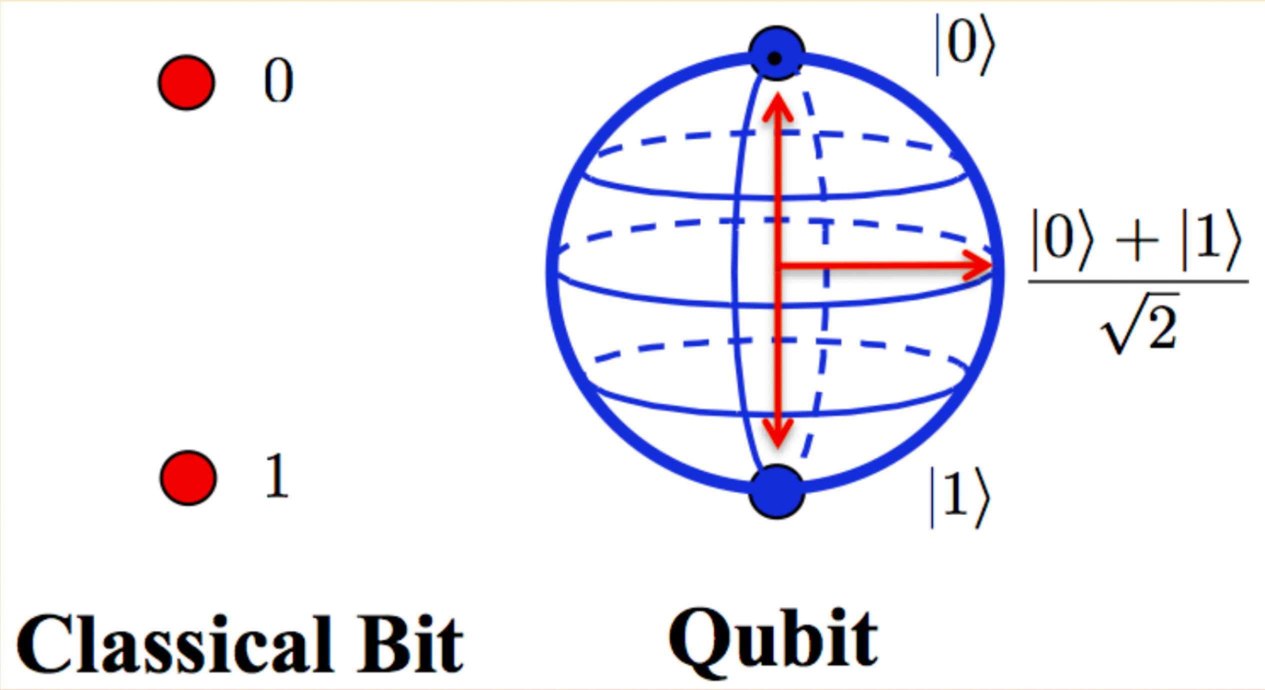


Image from Zheng, Mazumder (Wiley, 2019)

Encode information with Qubits

- Random bit (Bernoulli random variable) whose description is not governed by classical probability theory but by quantum mechanics
- Not only "because it can take real values in $[0, 1]$": complex numbers as coefficients α and β create interference
    - Interference is not reproducible with classical bits

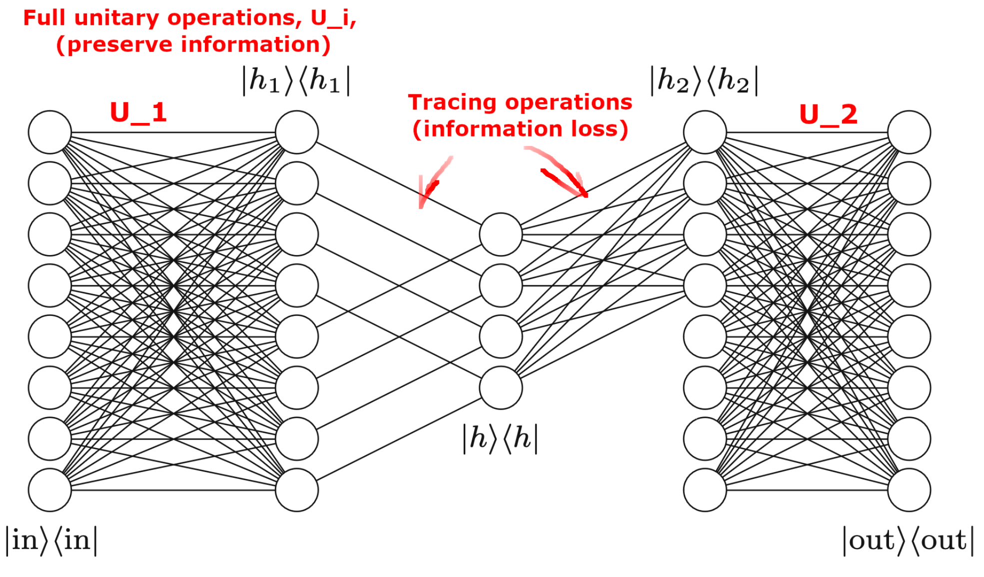


—
Image by prateekvjoshi

Represent neural networks

- Qubit operations can represent rather naturally neural networks

$$|\mathrm{in}\rangle\langle\mathrm{in}| = |q_1, q_2, q_3\rangle\langle q_1, q_2, q_3|.$$



- Gradient descent exploits intrinsic analytic differentiability of quantum circuits

$$\begin{aligned}
\partial_\mu \langle\psi(x,\theta)|\sigma_z|\psi(x,\theta)\rangle &= \langle 0|\ldots\partial_\mu e^{-i\mu\sigma}\ldots\sigma_z\ldots e^{i\mu\sigma}\ldots|0\rangle \\
&\quad + \langle 0|\ldots e^{-i\mu\sigma}\ldots\sigma_z\ldots\partial_\mu e^{i\mu\sigma}\ldots|0\rangle \\
&= \langle 0|\ldots(-i\sigma)e^{-i\mu\sigma}\ldots\sigma_z\ldots e^{i\mu\sigma}\ldots|0\rangle \\
&\quad + \langle 0|\ldots e^{-i\mu\sigma}\ldots\sigma_z\ldots(i\sigma)e^{i\mu\sigma}\ldots|0\rangle \\
&= \langle 0|\ldots(1-i\sigma)e^{-i\mu\sigma}\ldots\sigma_z\ldots(1+i\sigma)e^{i\mu\sigma}\ldots|0\rangle \\
&\quad + \langle 0|\ldots(1+i\sigma)e^{-i\mu\sigma}\ldots\sigma_z\ldots(1-i\sigma)e^{i\mu\sigma}\ldots|0\rangle
\end{aligned}$$

—

Images from M. Schuld, F. Petruccione (2018)

Need for new paradigma

- If you are interested in Neuromorphic computing or Quantum computing, drop me a line!

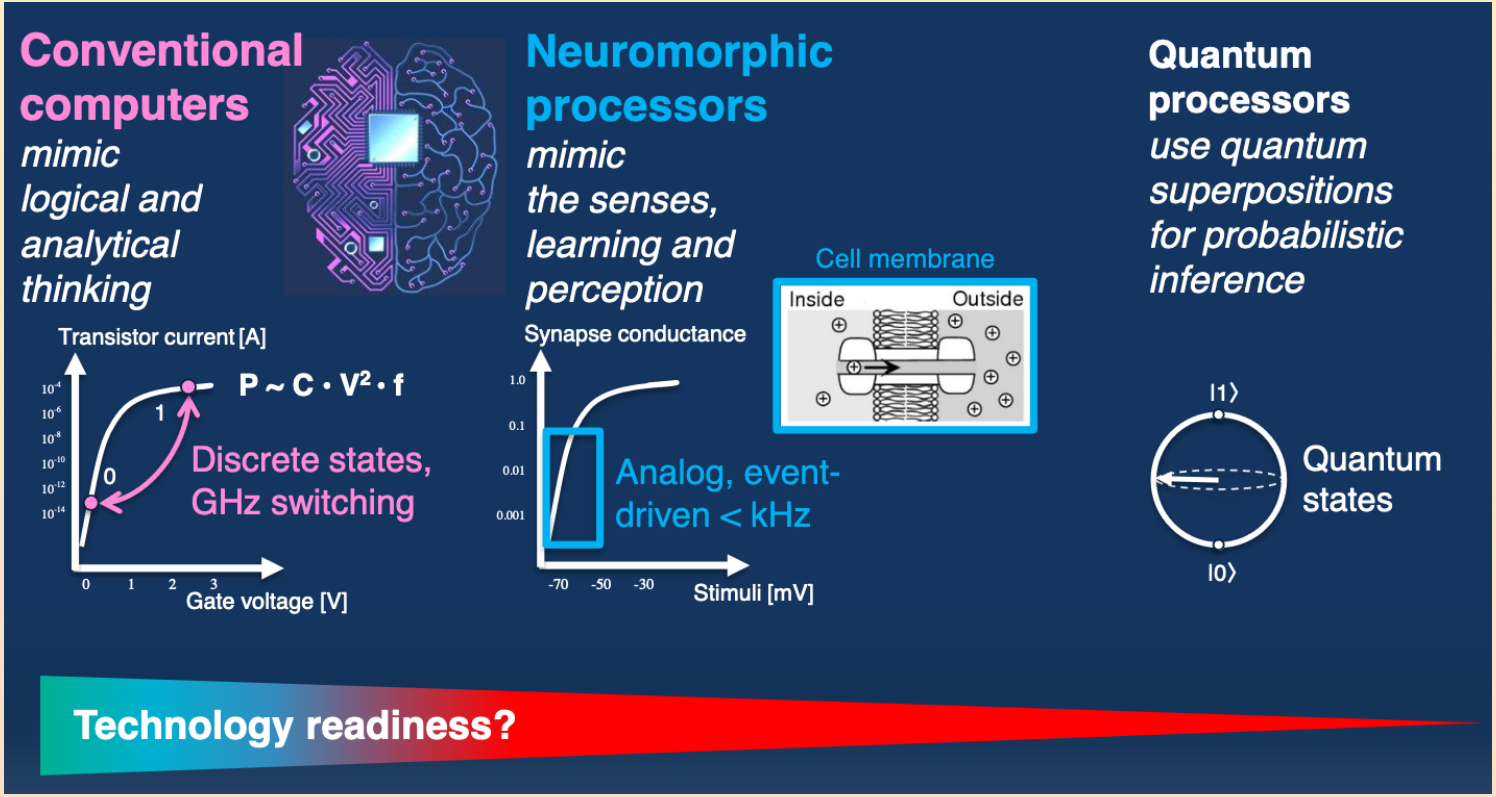


Image by Fredrik Sandin, Lulea University

Thank you!

Photo by me!