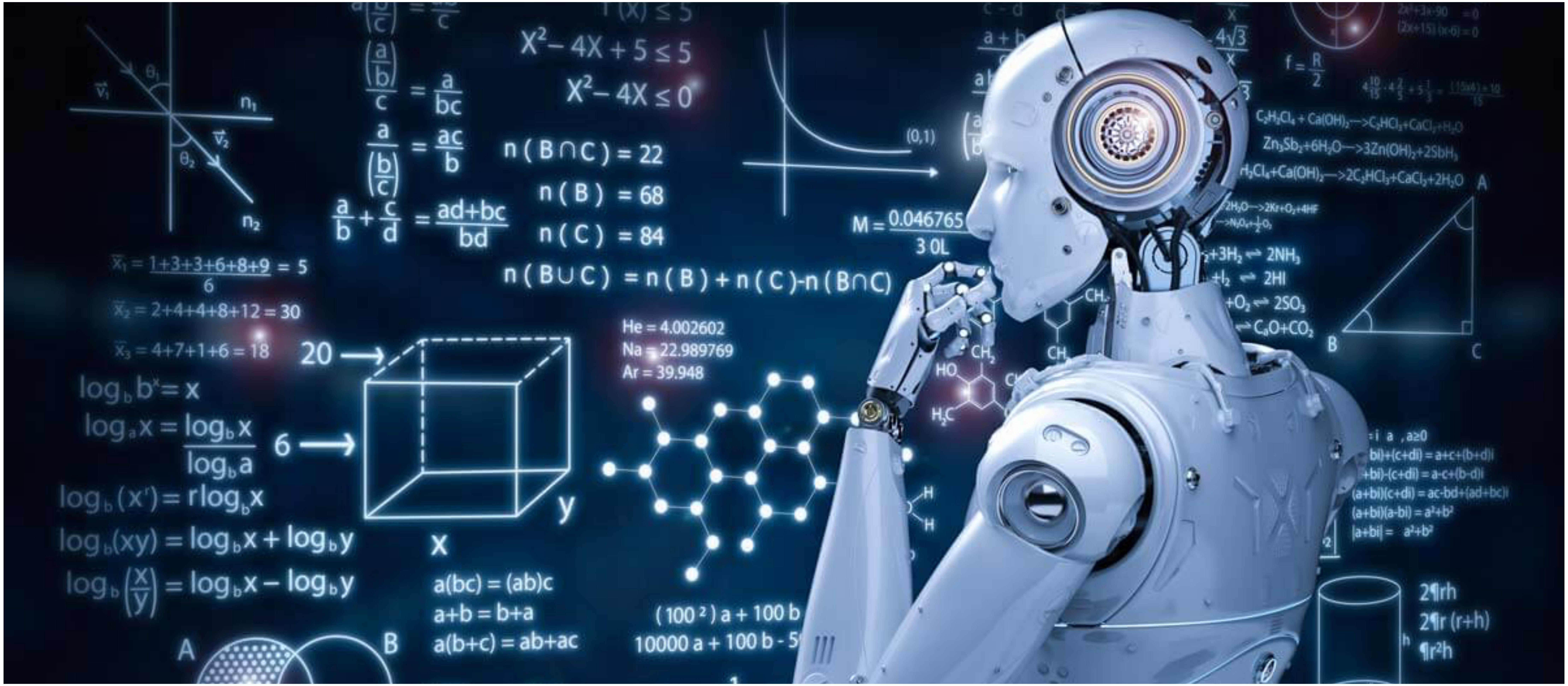
What is Machine Learning and why it is relevant for research?

Maurizio Pierini

CERN

European Research Council

CERN



What is Machine Learning ?

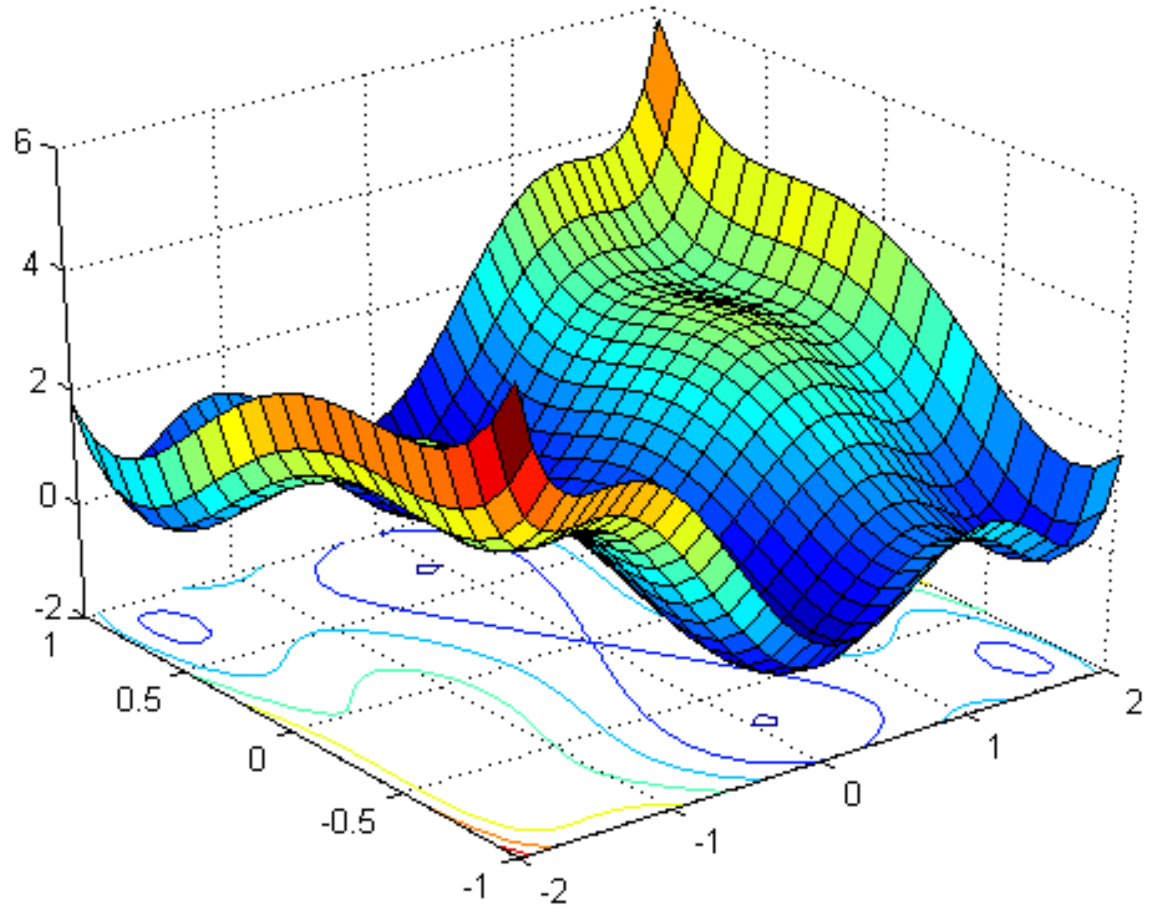
European
Research
Council

A definition (Wikipedia)

Machine learning (ML) is the scientific study of algorithms and statistical models that computer systems use to progressively improve their performance on a specific task. Machine learning algorithms build a mathematical model of sample data, known as "training data", in order to make predictions or decisions without being explicitly programmed to perform the task.

TRUE ANSWER

TRAINING DATA



LOSS FUNCTIONS

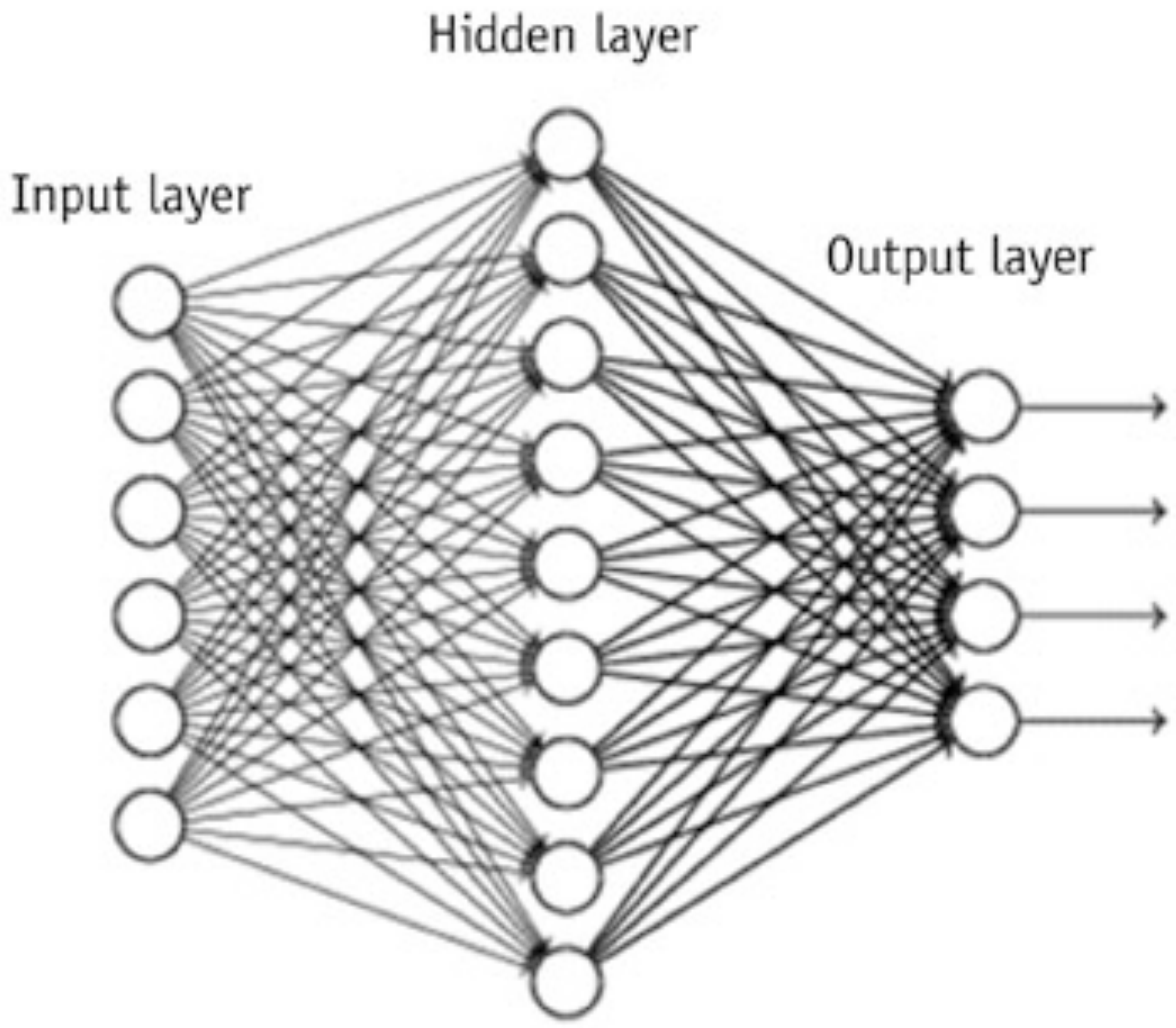
The name of the game is finding the algorithm setting (its parameter values) that minimise the loss, i.e. the error made in prediction

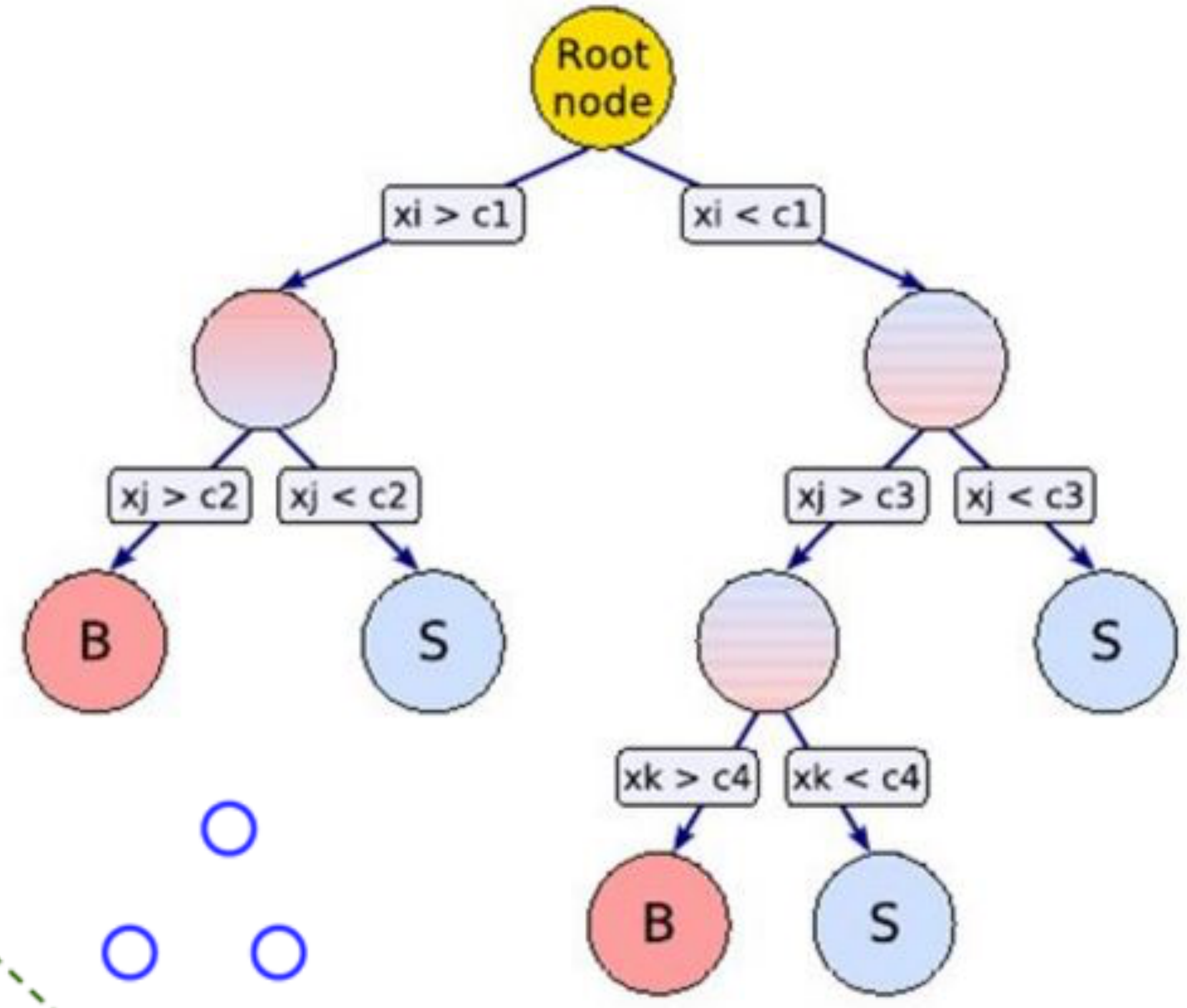
ALGORITHM

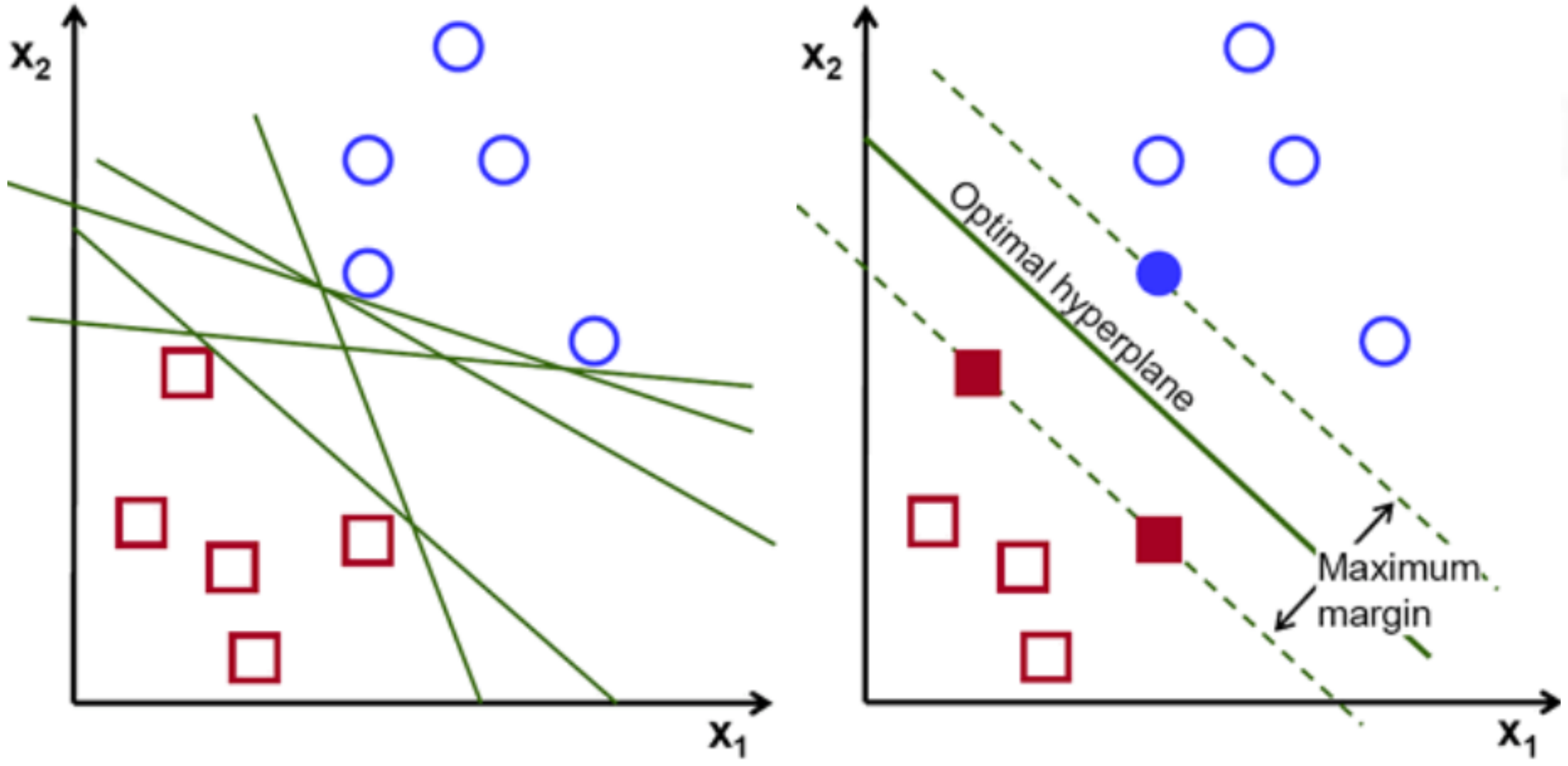
PREDICTION

Many flavors of ML

- Different ML algorithms had their moment of glory
  - (Shallow) neural networks dominated in the 80's
  - Alternatives emerged in the 90's
    - Support vector machine
    - Boosting of decision trees

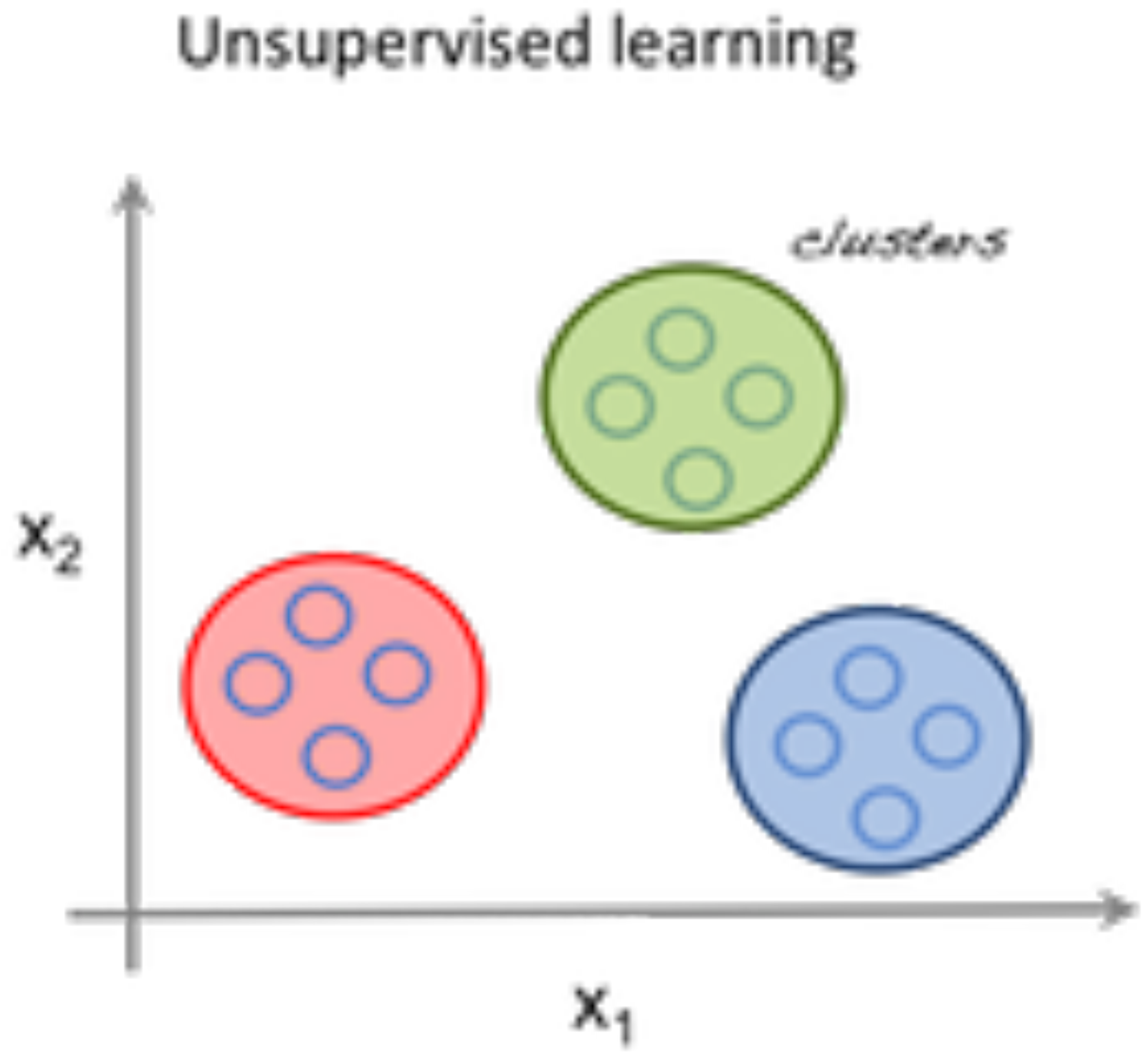






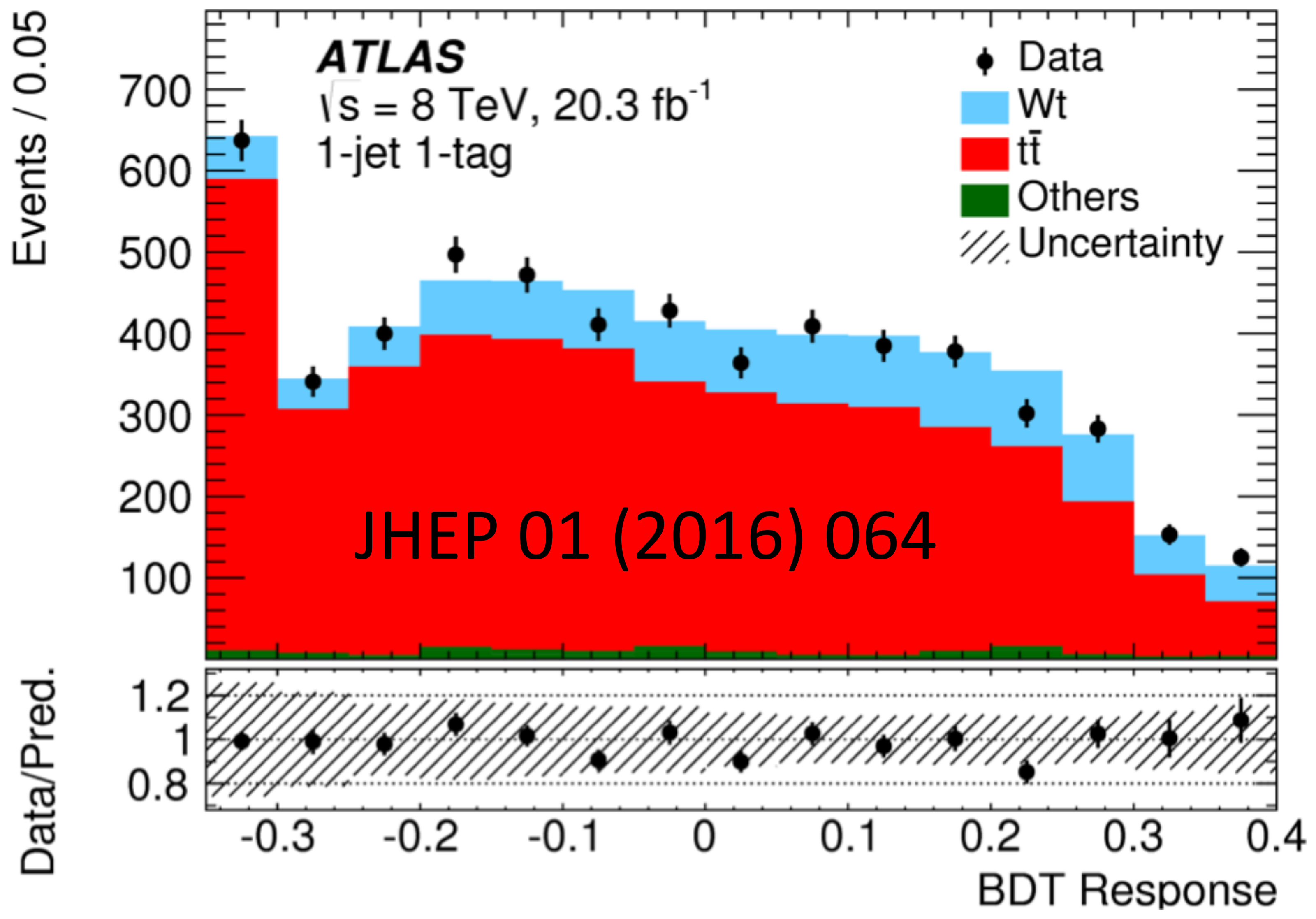
A two-steps process

- ***Learning:*** *train the algorithm on a provided dataset*
  - ***Supervised:*** *the dataset X comes with the right answer y (right class in a classification problem). The algorithm learns the function*
  - ***Unsupervised:*** *the dataset X comes with no label. The algorithm learns structures in the data (e.g., alike events in a clustering algorithm)*
  - ***Reinforcement:*** *learn a series of actions and develop a decision-taking algorithm, based on some action/reward model*
- ***Inference:*** *once trained, the model can be applied to other datasets*

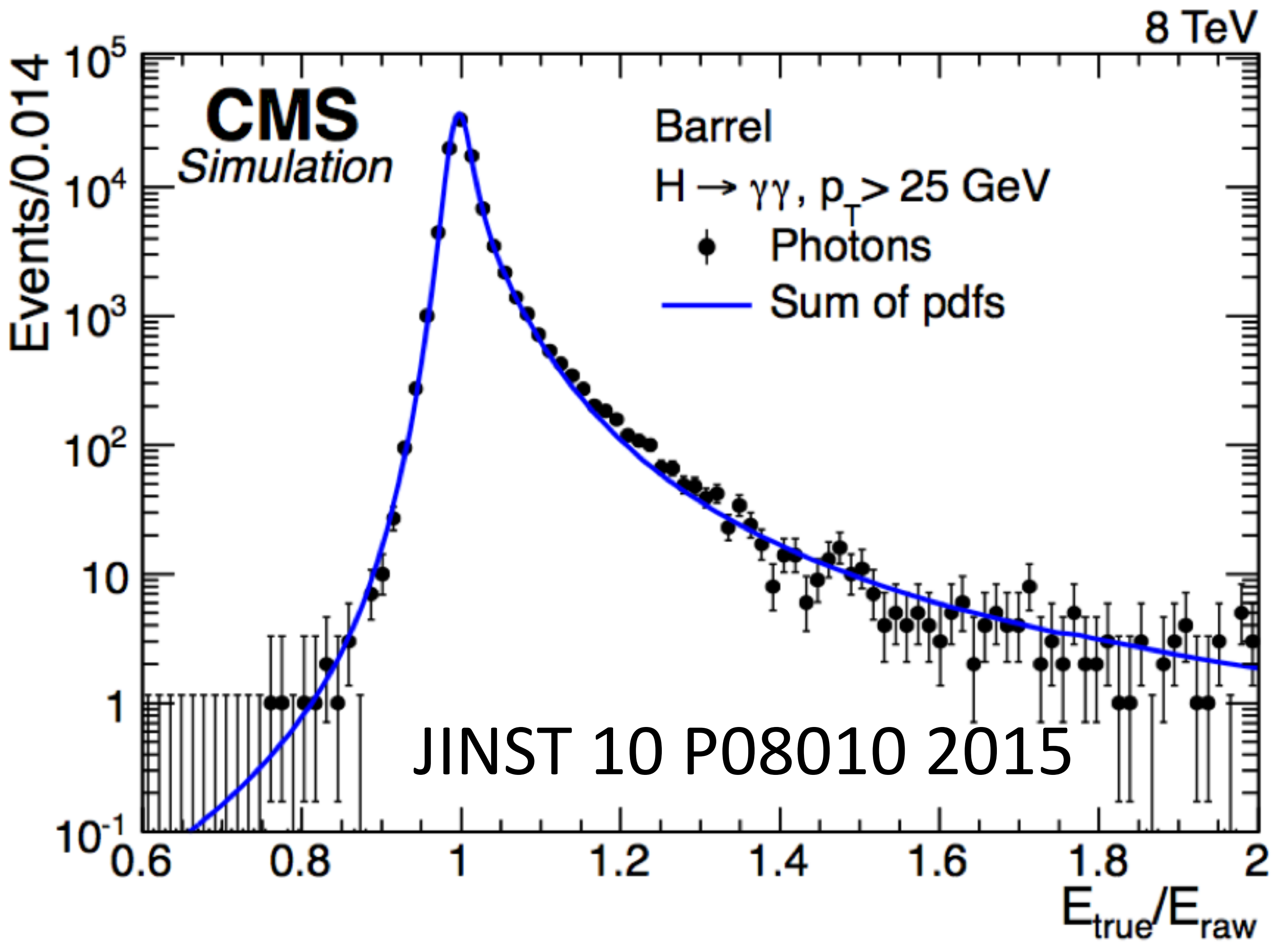


Machine Learning in HEP

- *Classification:*
  - *given an image, identify the object represented*
  - *in particle physics, given a particle shower, identify the particle kind*

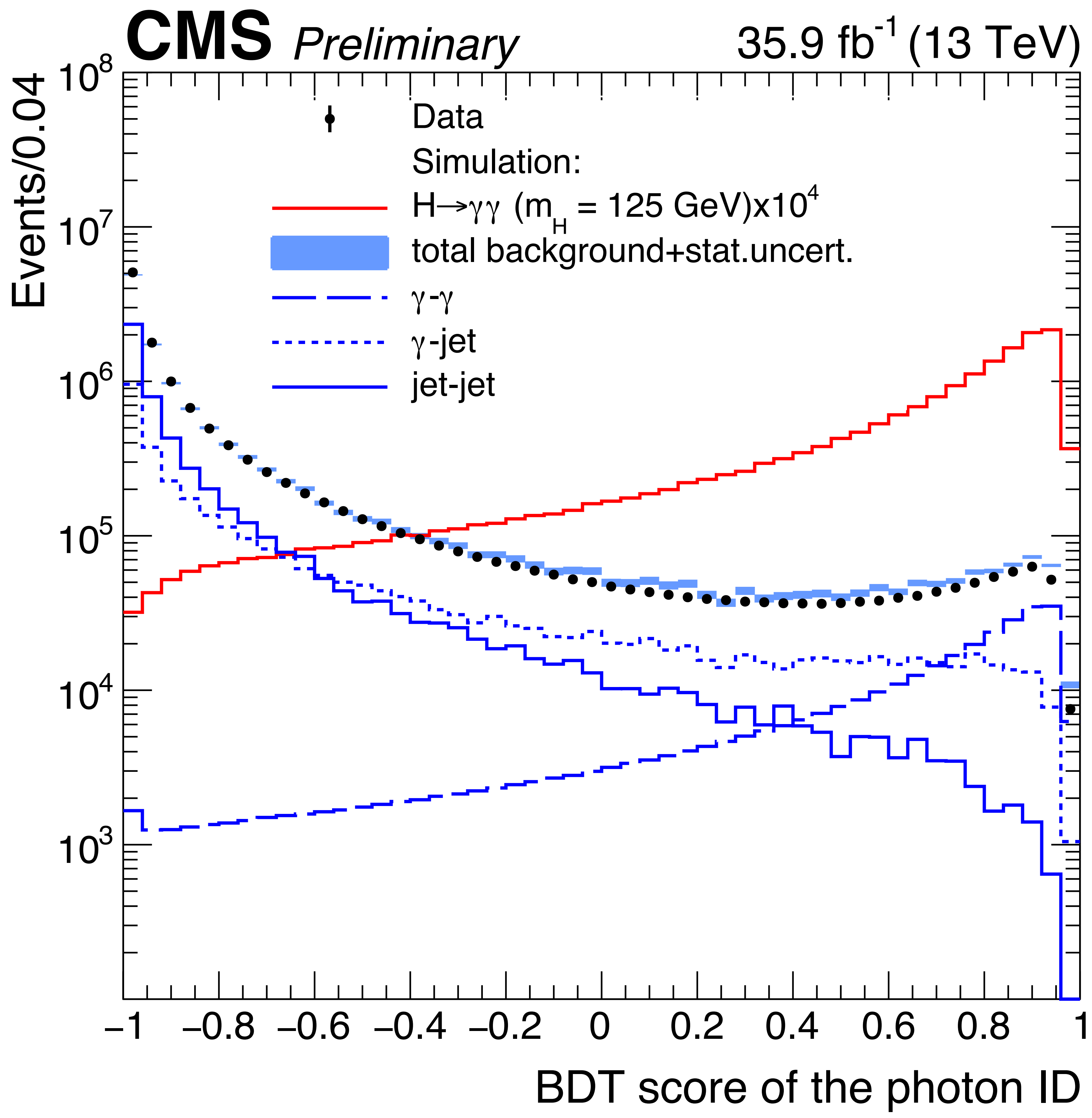


- *Regression:*
  - *given a set of quantities x, learn some function f(x)*
  - *in particle physics, given a particle shower, learn its energy*



Machine Learning in HEP

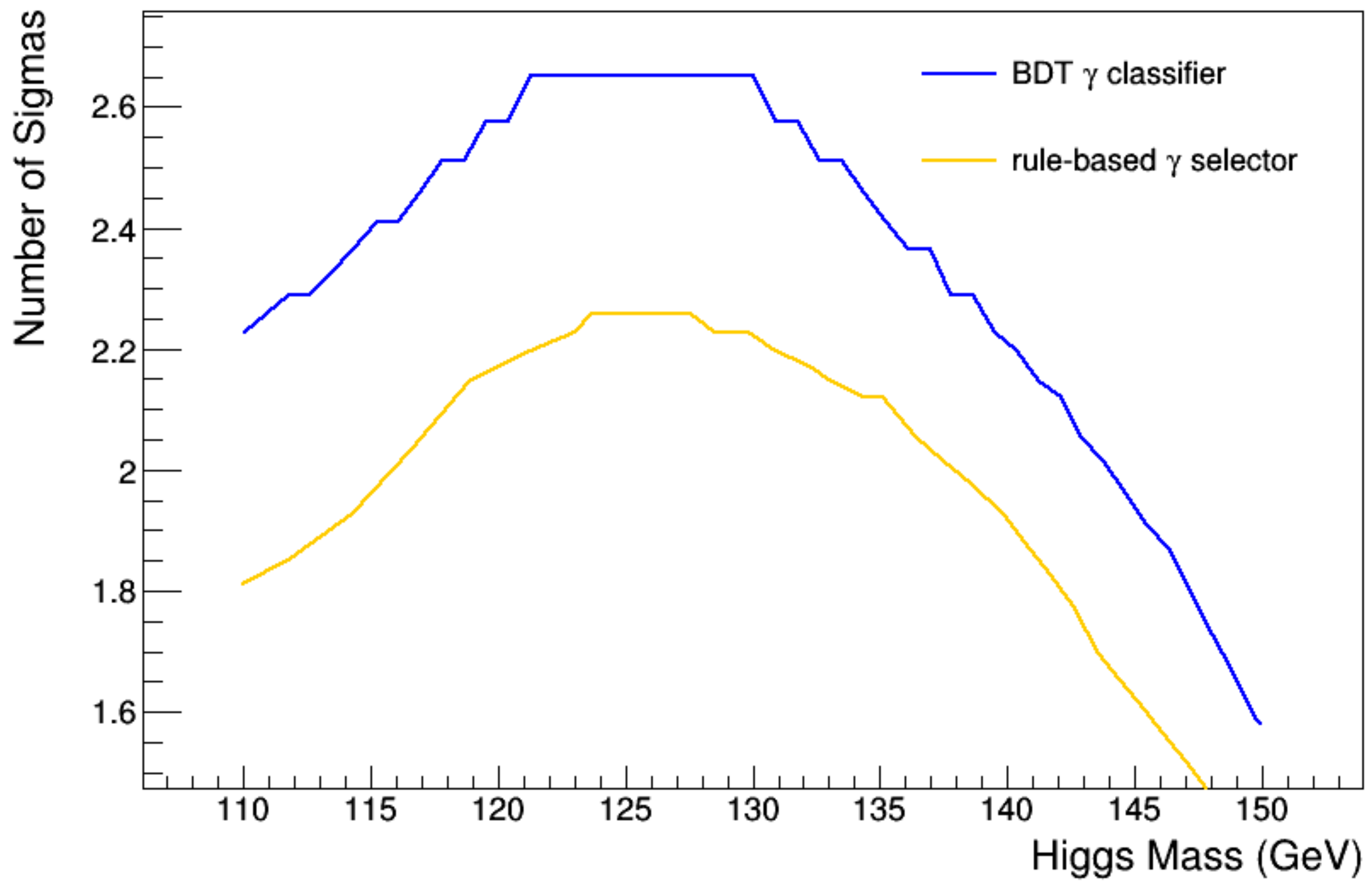
- *Classification:*
  - *identify a particle & reject fakes*
  - *identify signal events & reject background*
- *Regression:*
  - *Measure energy of a particle*
- *Up to now, these task mainly solved with BDTs*
  - *moved to Deep Learning for analysis-specific tasks*
  - *same will happen for centralised tasks (eventually)*

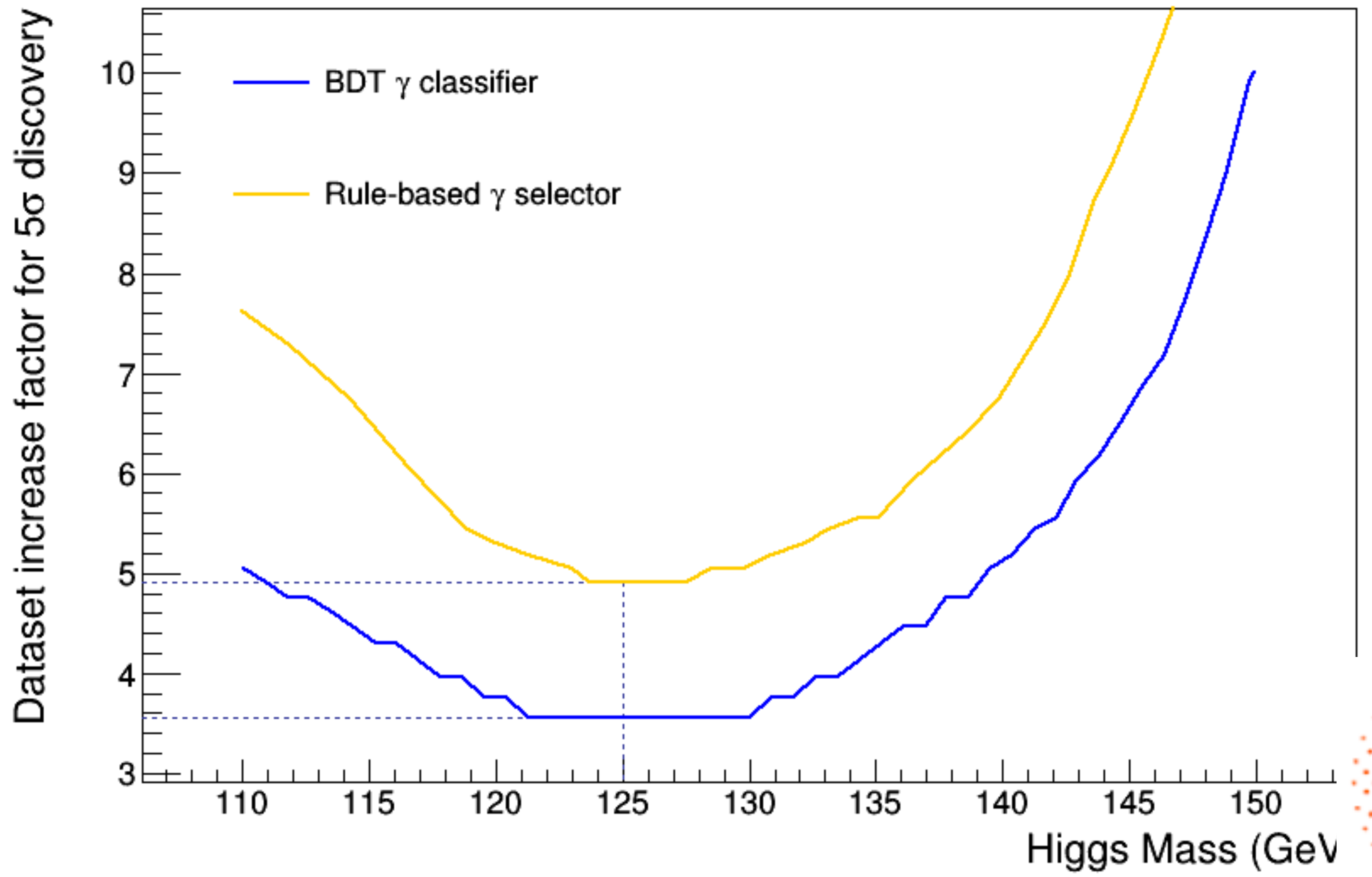


Centralised task (in online or offline reconstruction)
Analysis-specific task (by users on local computing infrastructures)

Machine Learning in HEP

- *Long tradition*
  - *Neural networks used at LEP and the Tevatron*
  - *Boosted Decision Trees introduced by MiniNooNE and heavy used at BaBar*
  - ***BDTs ported to LHC and very useful on Higgs discovery***
  - *Now Deep Learning is opening up many new possibilities*

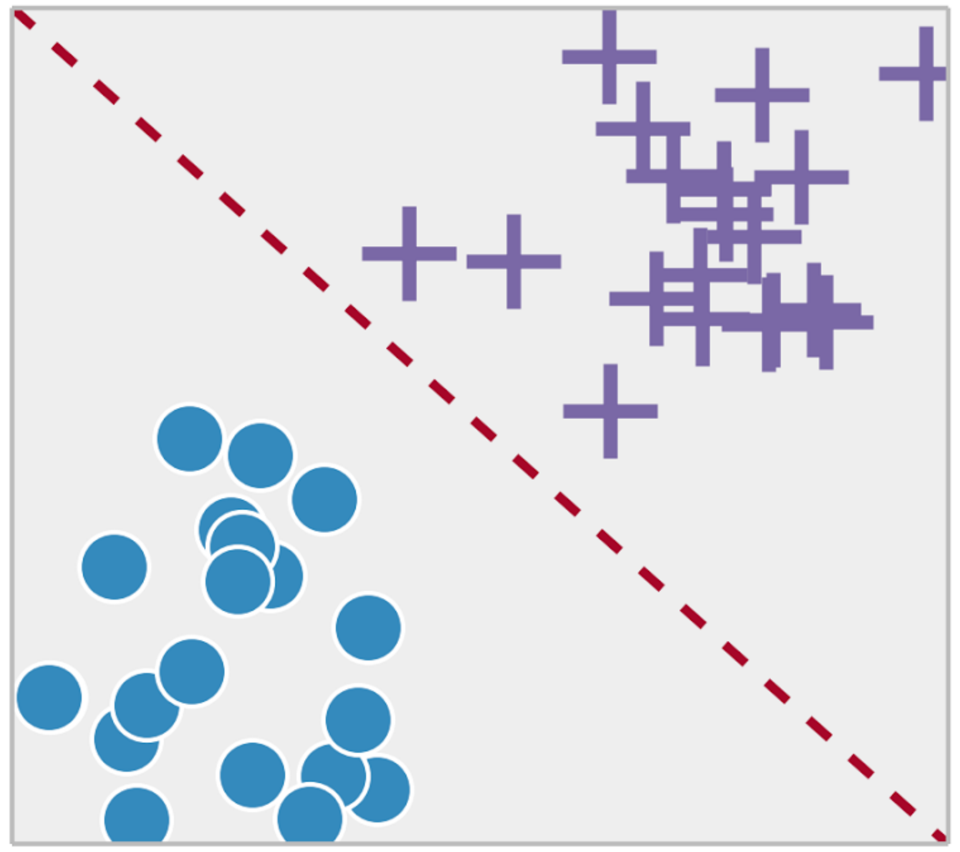


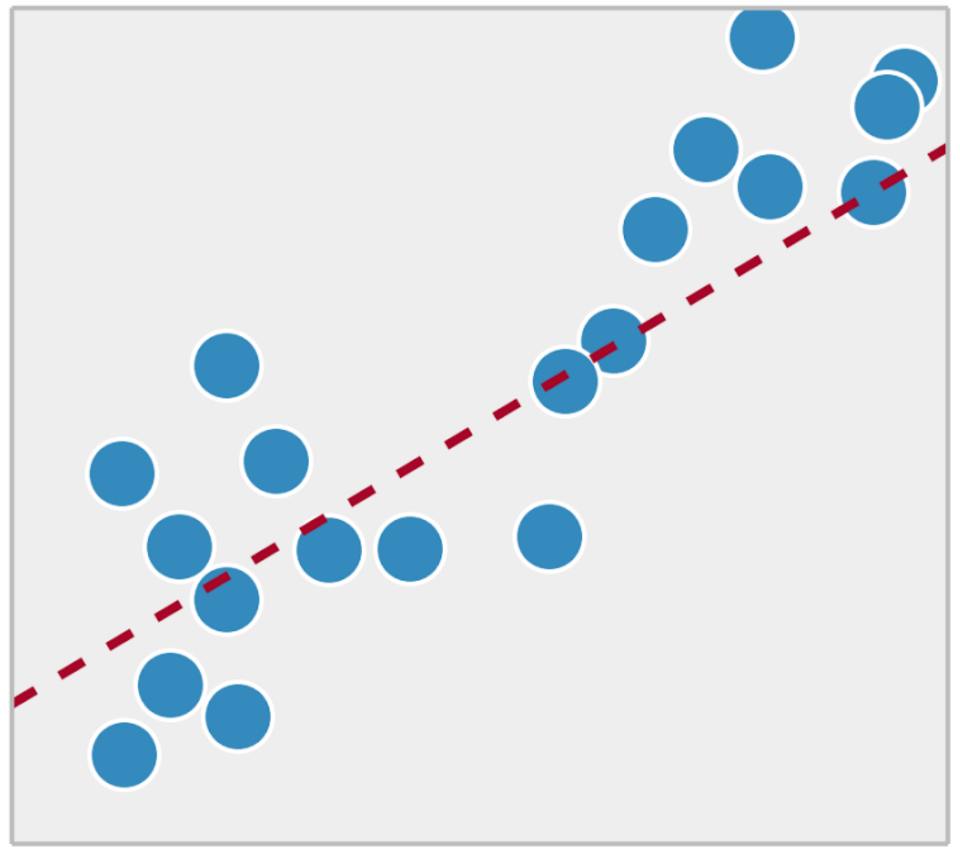


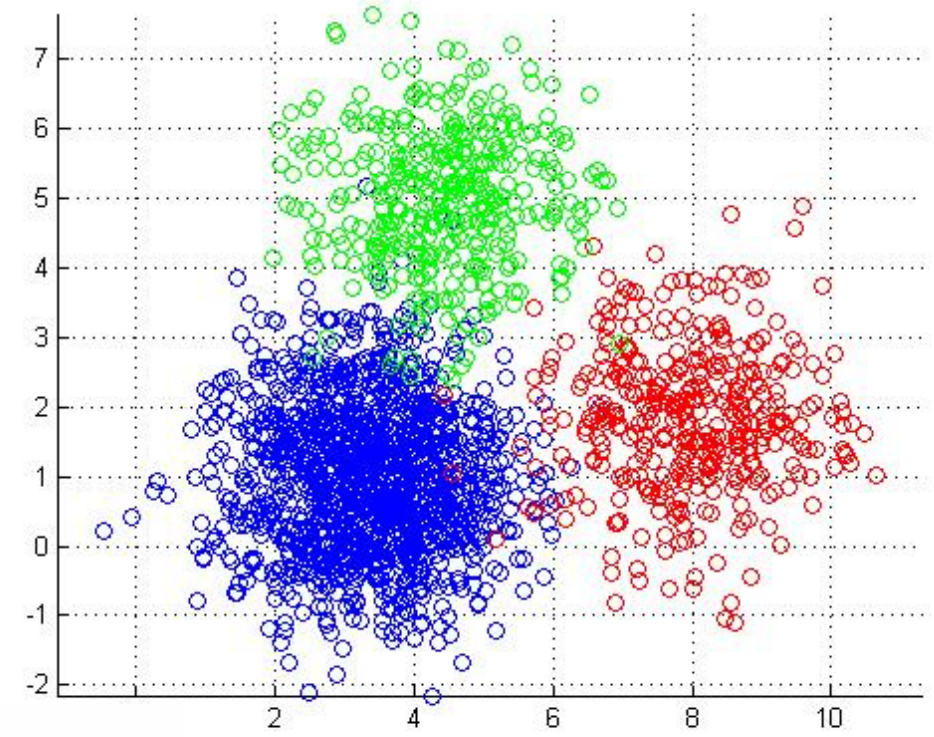
European Research Council

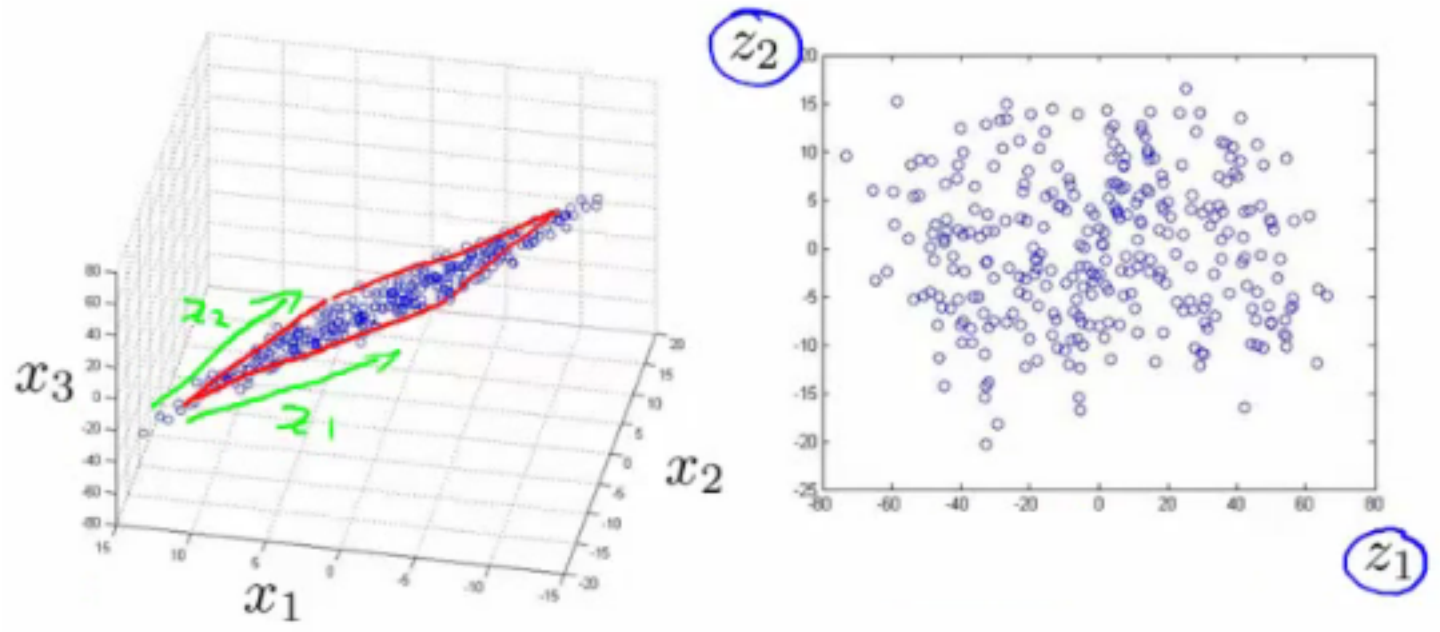
Typical problems

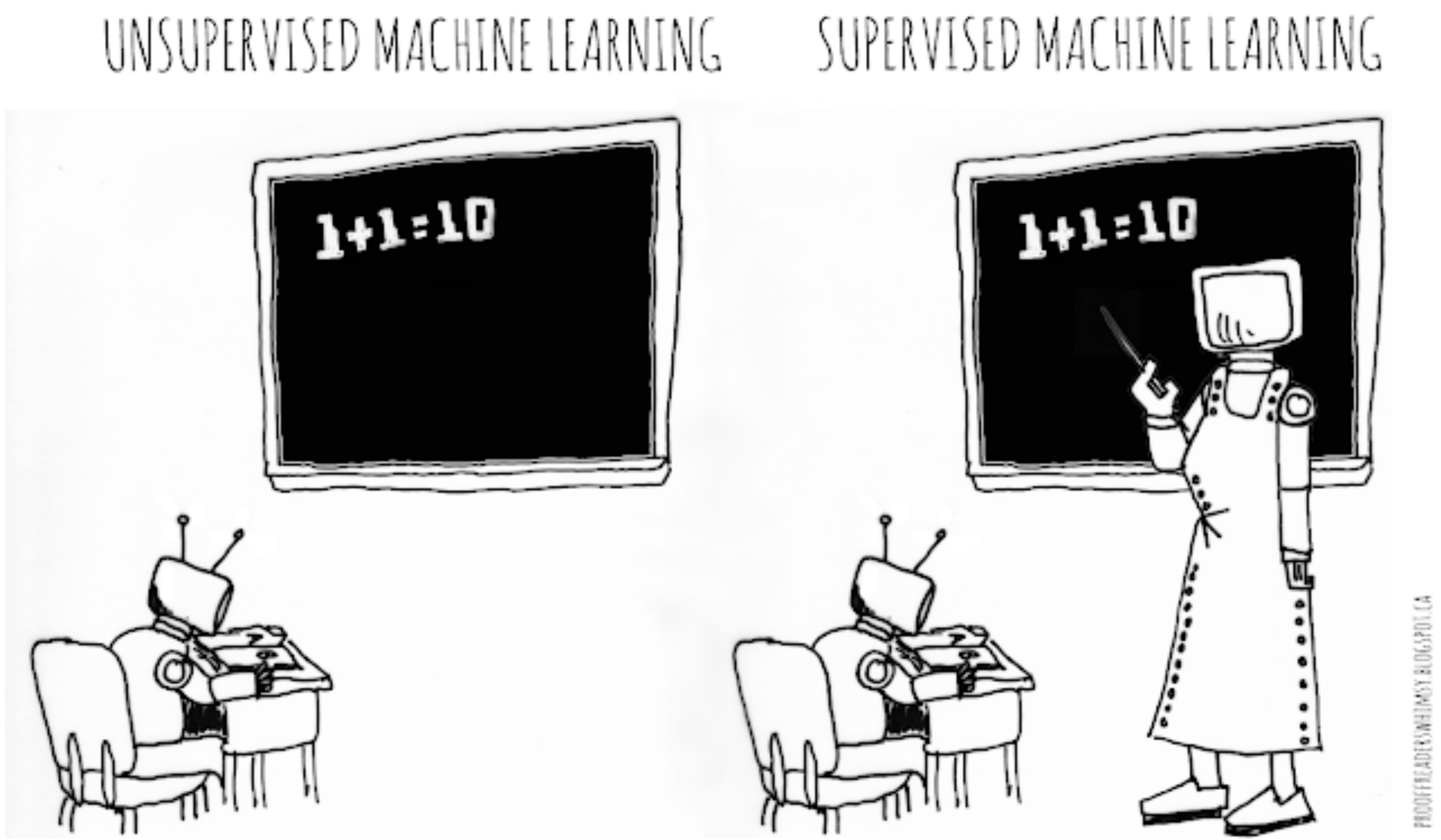
- ***Classification***: *associate a given element of a dataset to one of N exclusive classes*
- ***Regression***: *determine a continuous value y from a set of inputs x*
- ***Clustering***: *group elements of a dataset because of their similarity according to some learned metric*
- ***Dimensionality reduction***: *find the k quantities of the N inputs (with k<N) that incorporate the relevant information (e.g., principal component analysis)*







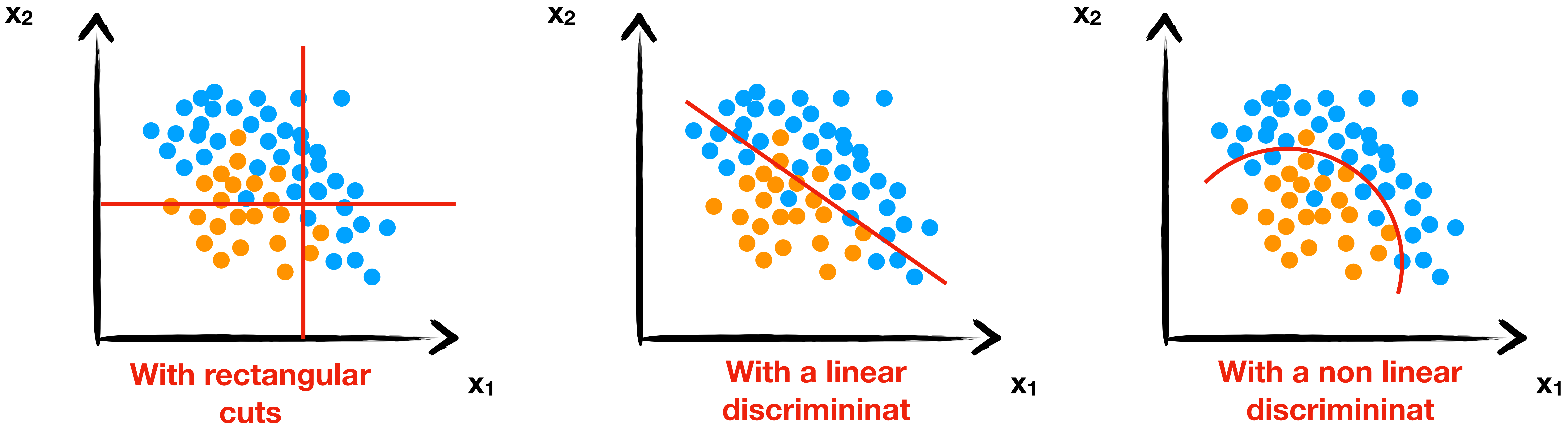




Supervised Learning

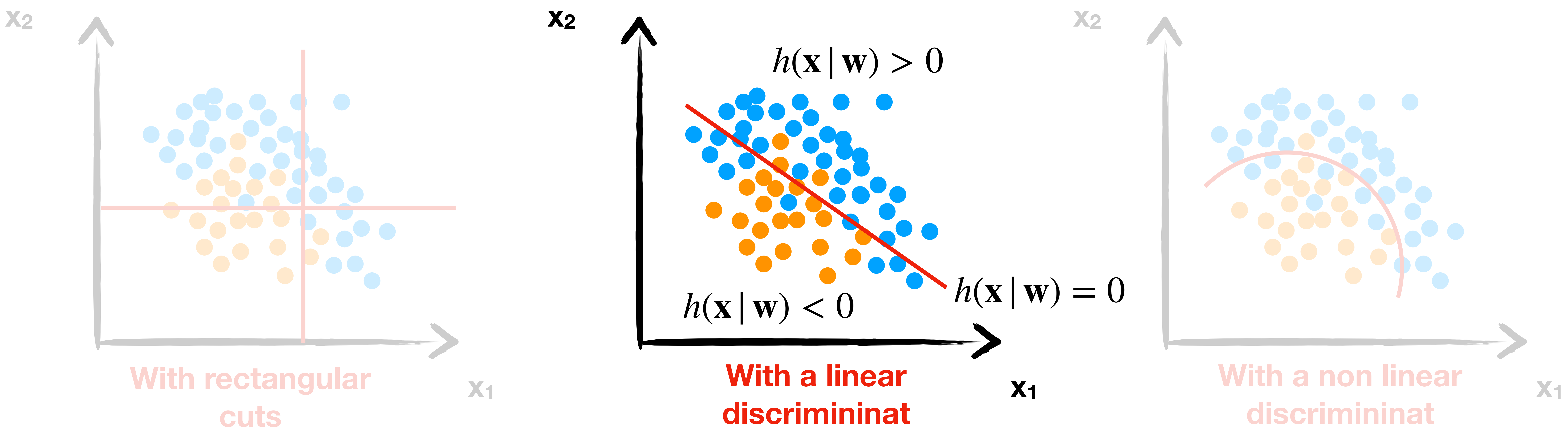
A simple example: S vs B selection

- *Define a selection to separate the signal from the background*



A simple example: S vs B selection

- *Define a selection to separate the signal from the background*



- *Define a decision boundary which gives optimal separation*

$$h(\mathbf{x}\,|\,\mathbf{w}) = \mathbf{w}^{\mathrm{T}}\mathbf{x} = 0$$

(Signed) distance between x and the boundary plane

Logistic Regression

- Give as input pairs of inputs and outputs:

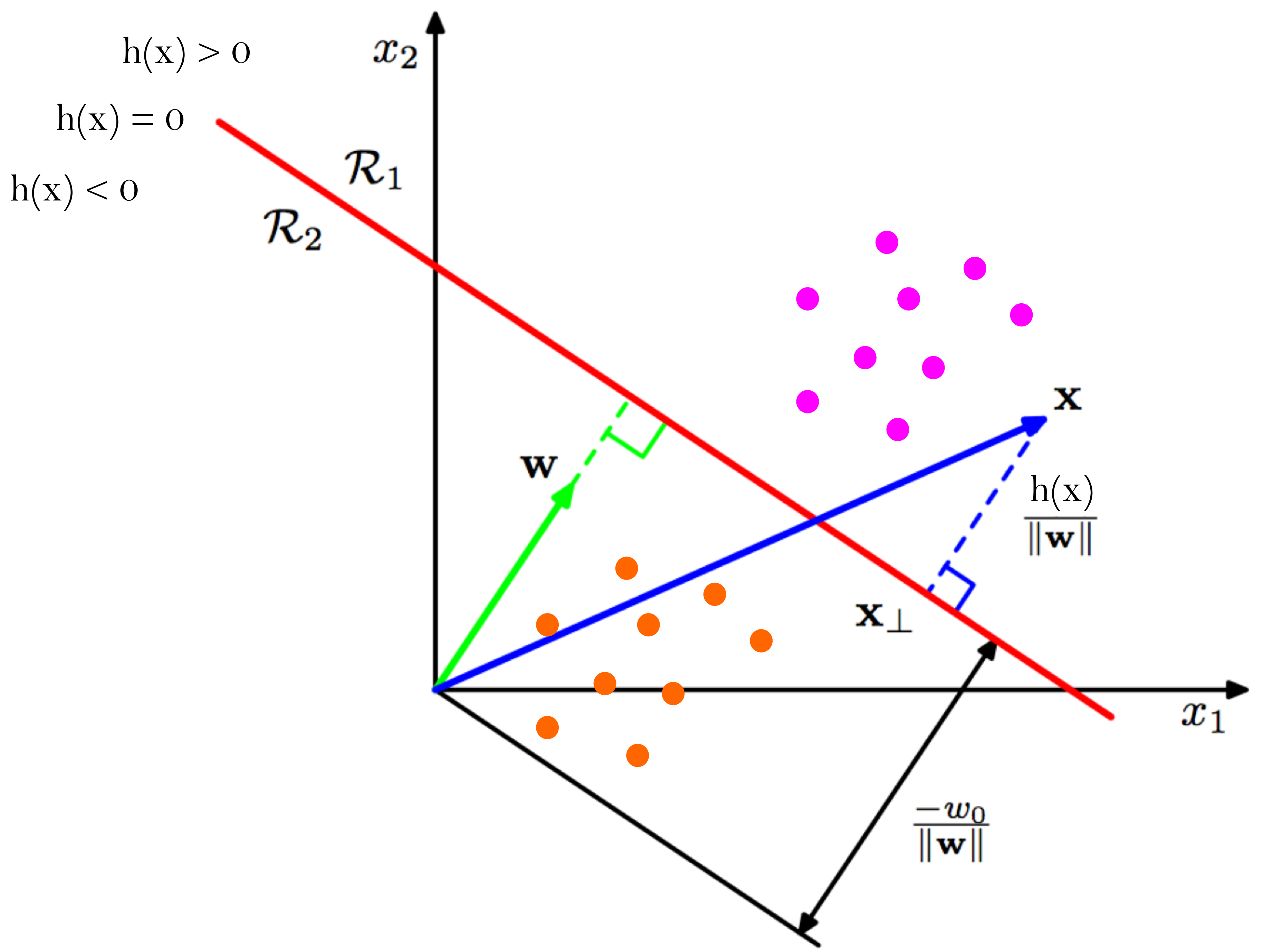
$$x_i \in \mathbb{R}^n \qquad y_i = \{0,1\}$$

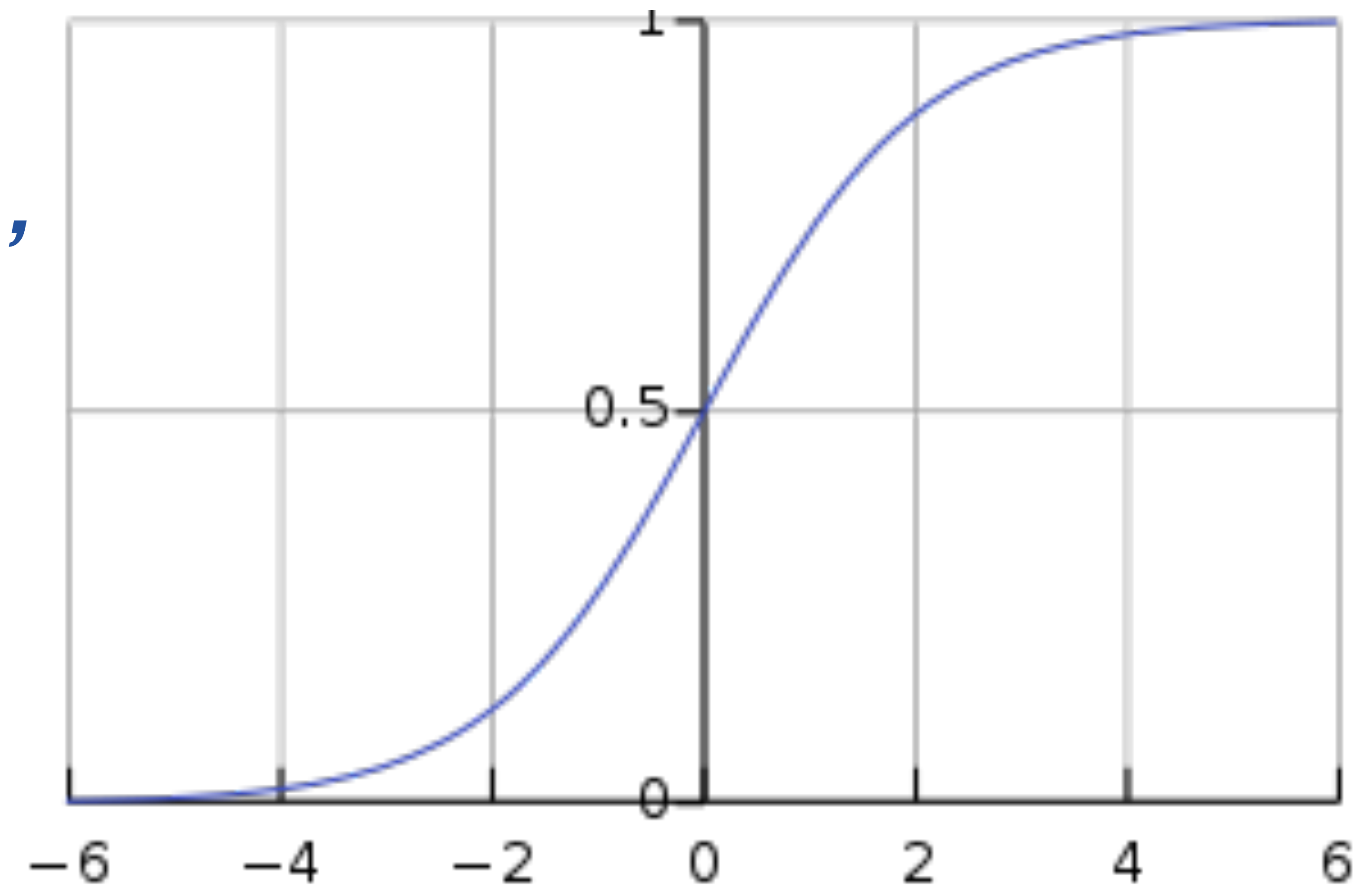
- Model the probability of x to be signal (y=1) as

$$p(y=1\,|\,x) = \frac{1}{1+e^{-\mathbf{w}^T\mathbf{x}}}$$

  - The larger (and positive) the distance the closer p to 1
  - The larger (and negative) the distance, the closer p to 0

- We can choose the plane such that we maximise the probability of the signal and minimise that of the background





Bernoulli's problem

- Bernoulli's problem: probability of a process that can give 1 or 0
- The corresponding likelihood is (as usual) the product of the probabilities across the events
- Maximizing the likelihood corresponds to minimizing the -logL
- Minimizing the -logL corresponds to minimizing the binary cross entropy
- How do we minimise it?

$$\mathscr{L} = \prod_i p_i^{x_i}(1-p_i)^{1-x_i}$$

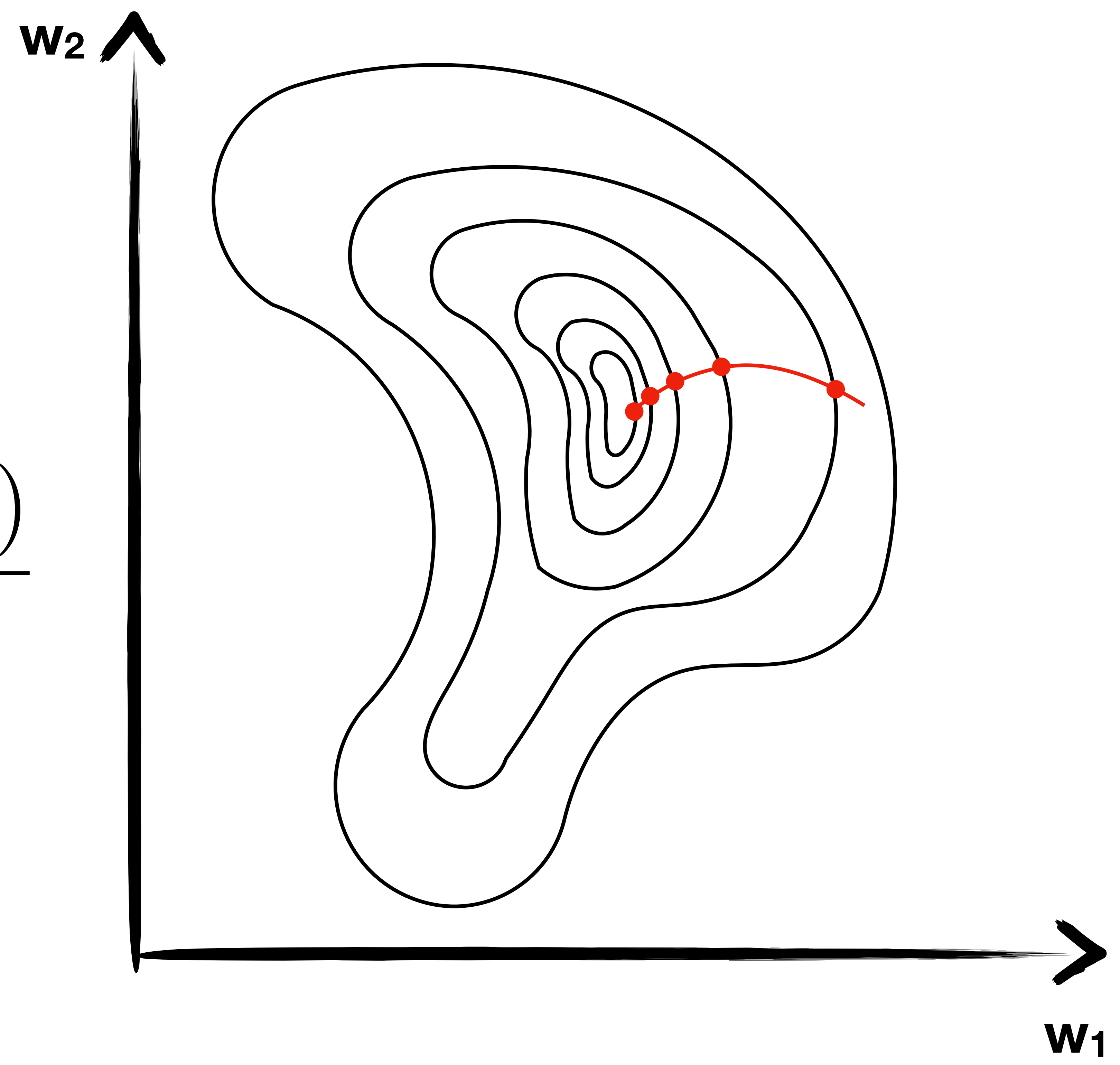
$$-\log \mathscr{L} = -\log[\prod_i p_i^{x_i}(1-p_i)^{1-x_i}]$$

$$= -\sum_i \left[x_i \log p_i + (1-x_i)\log(1-p_i)\right]$$

European Research Council

Gradient Descent

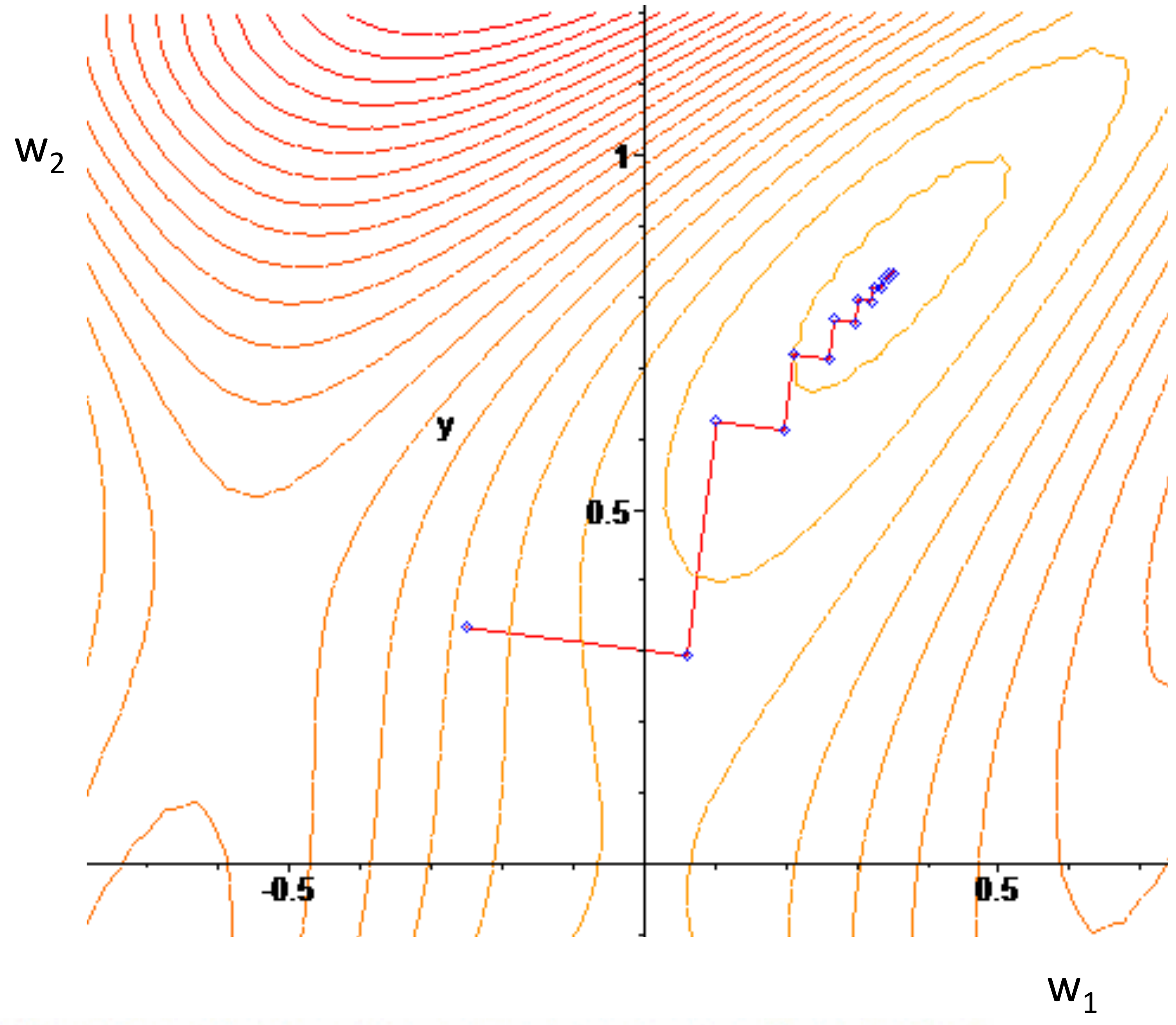
- *Gradient Descent is a popular minimisation algorithm*
    - *Start from a random point*
    - *Compute the gradient wrt the model parameters* $\frac{\partial L(\mathbf{w})}{\partial \mathbf{w}}$
    - *Make a step of size η (the **learning rate**) towards the gradient direction*
    - *Update the parameters of the mode accordingly*
    - *Effective, but computationally expensive (gradient over entire dataset)*

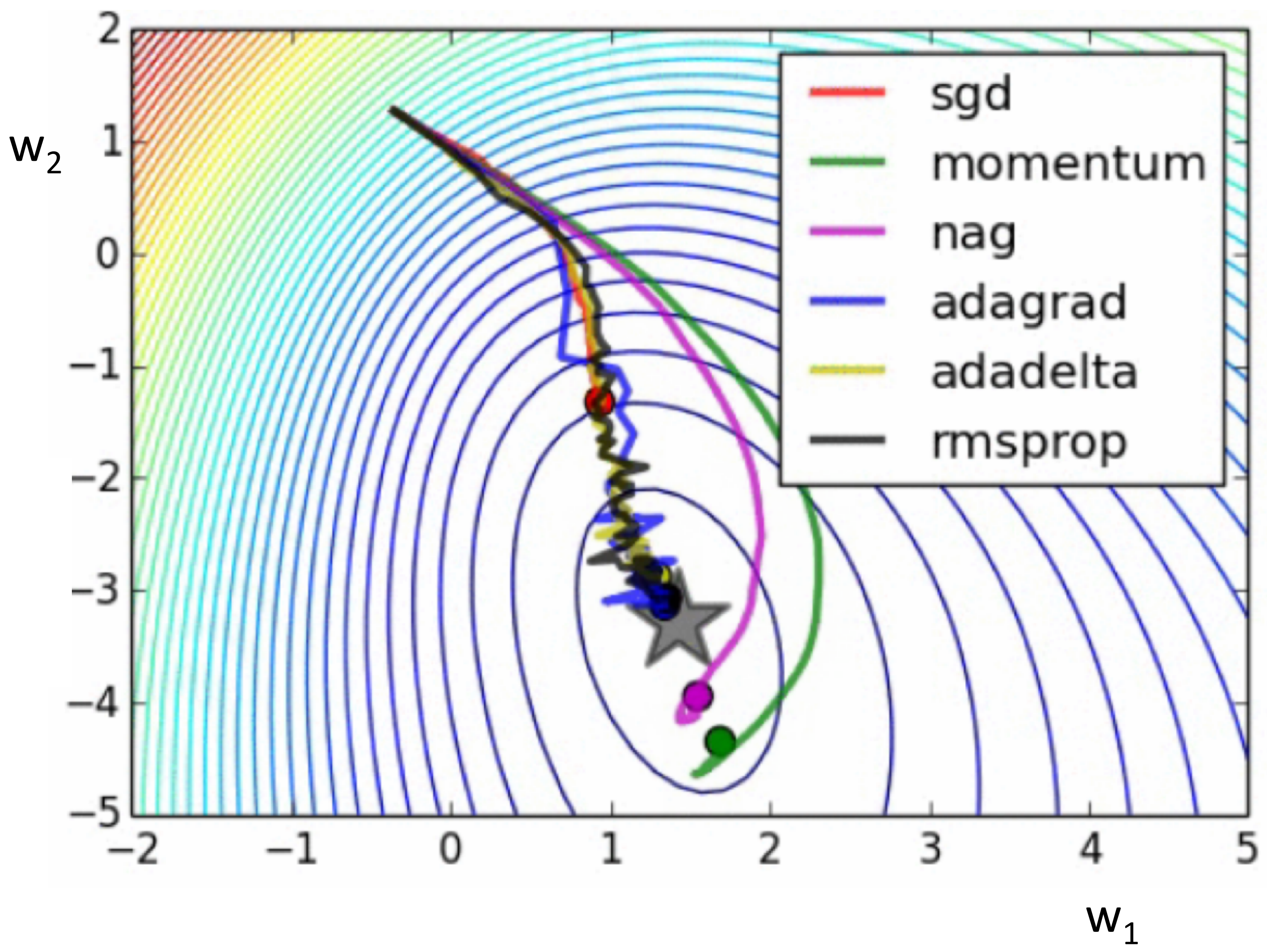


$$\mathbf{w}' \leftarrow \mathbf{w} - \eta \frac{\partial L(\mathbf{w})}{\partial \mathbf{w}}$$

Stochastic Gradient Descent

- Make the minimisation more computationally efficient
  - Compute gradient on a small batch of events (faster & parallelizable, but noisy)
  - Average over the batches to reduce noise
  - BEWARE: better scalability come at the cost of (sometimes) not converging
  - Many recipes exist to help convergence, by playing with the algorithm setup (e.g., adapting learning rate)





Example: regression & MSE

- *Given a set of points, find the curve that goes through them*

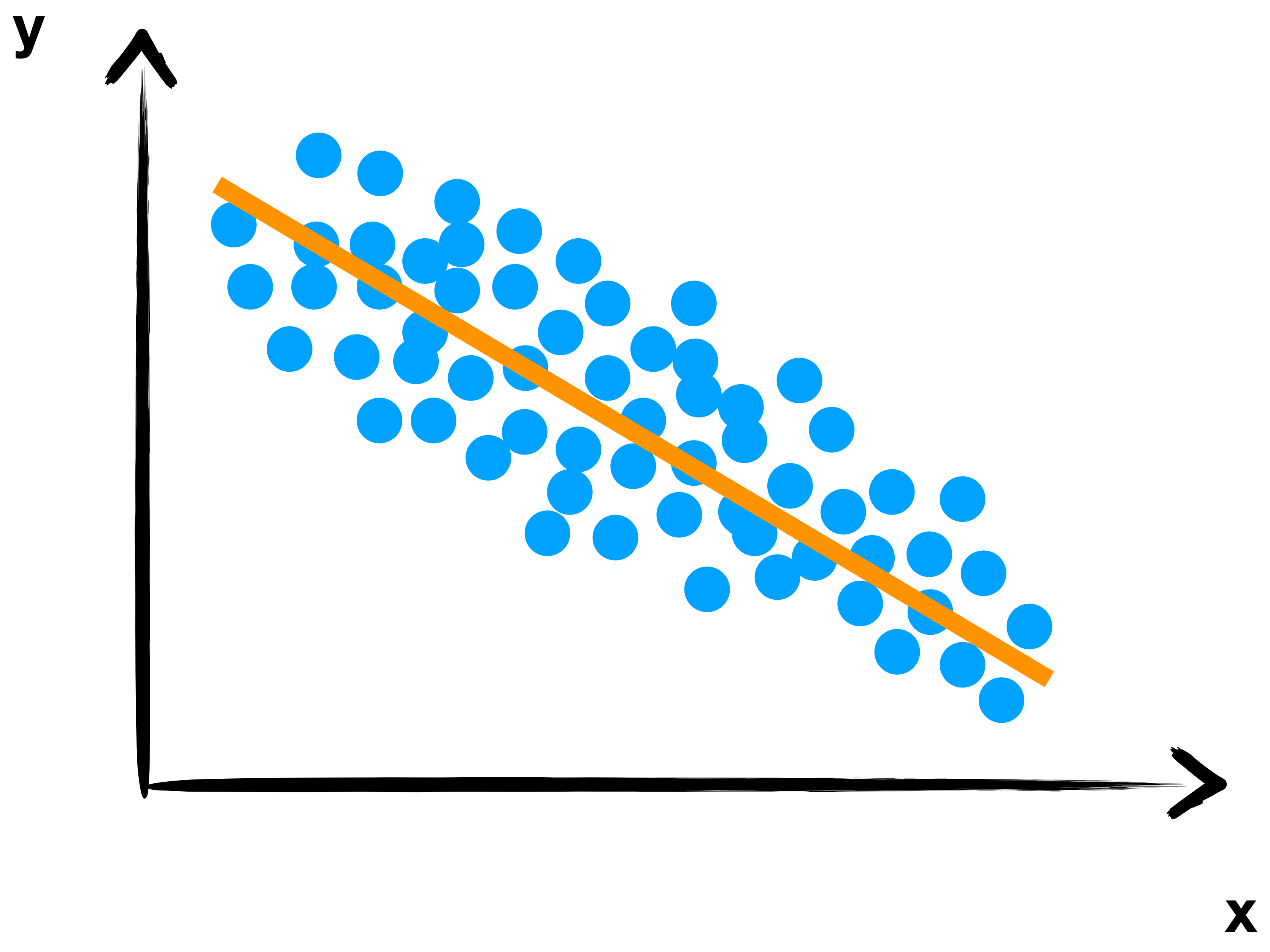
  - *Can be a linear model*

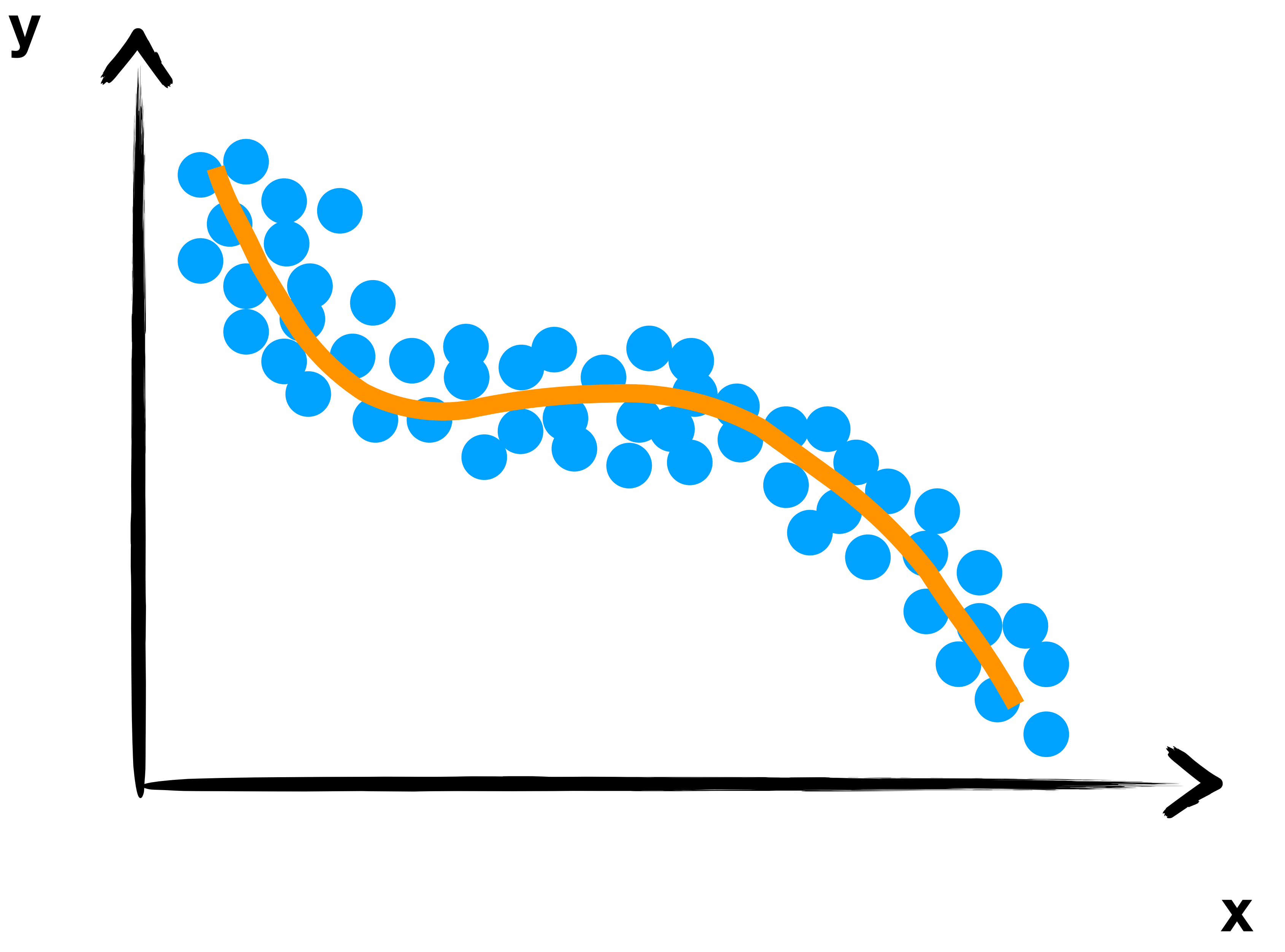
$$y_i = a x_i + b$$

  - *Can be a linear function of non-linear kernel of the x. For instance, a polynomial basis*

$$y_i = a\,\phi(x_i) + b$$

New feature, “engineered” from the input features





Example: regression & MSE

- Take some model (e.g., linear)

$$h(x_i \mid a, b) = a x_i + b$$

- Consider the case of a Gaussian dispersion of y around the expected value

$$y_i = h(x_i) + e_i \qquad p(e_i) = \frac{1}{\sqrt{2\pi\sigma}} e^{-\frac{e_i^2}{2\sigma^2}}$$

- Assume that the resolution σ is fixed

- Write down the likelihood

$$\mathcal{L} = \prod_i \frac{1}{\sqrt{2\pi\sigma}} e^{-\frac{e_i^2}{2\sigma^2}} = \prod_i \frac{1}{\sqrt{2\pi\sigma}} e^{-\frac{(y_i - h(x_i))^2}{2\sigma^2}}$$

Example: regression & MSE

- *The maximisation of this likelihood corresponds to the minimisation of the mean square error (MSE)*

$$argmin[-2\log\mathscr{L}] = argmin\Big[-2\log\big[\prod_i \frac{1}{\sqrt{2\pi\sigma}} e^{-\frac{(y_i-h(x_i))^2}{2\sigma^2}}\big]\Big]$$

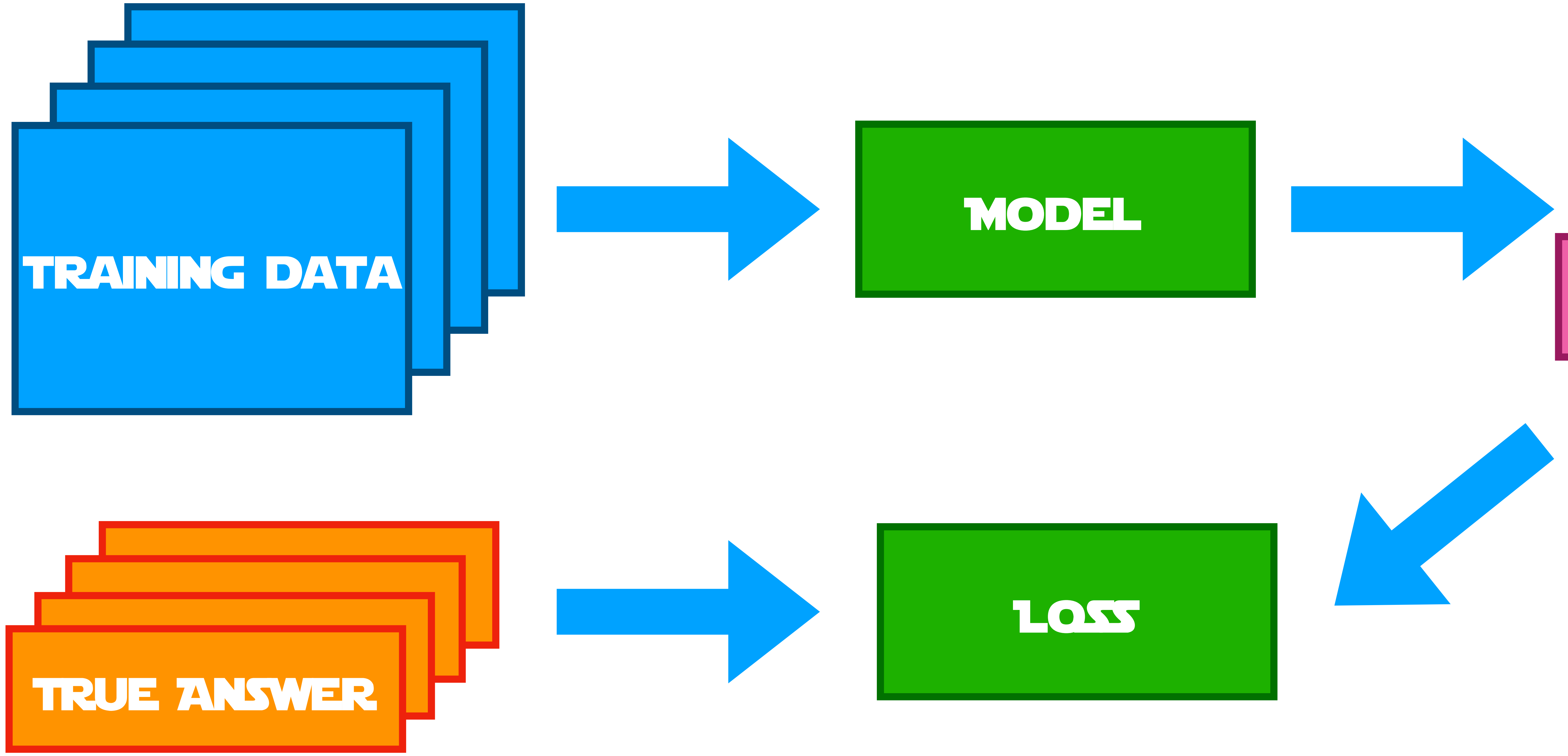
$$= argmin\Big[\sum_i \frac{(y_i-h(x_i))^2}{\sigma^2}\Big] = argmin\Big[\sum_i (y_i-h(x_i))^2\Big] = MSE$$

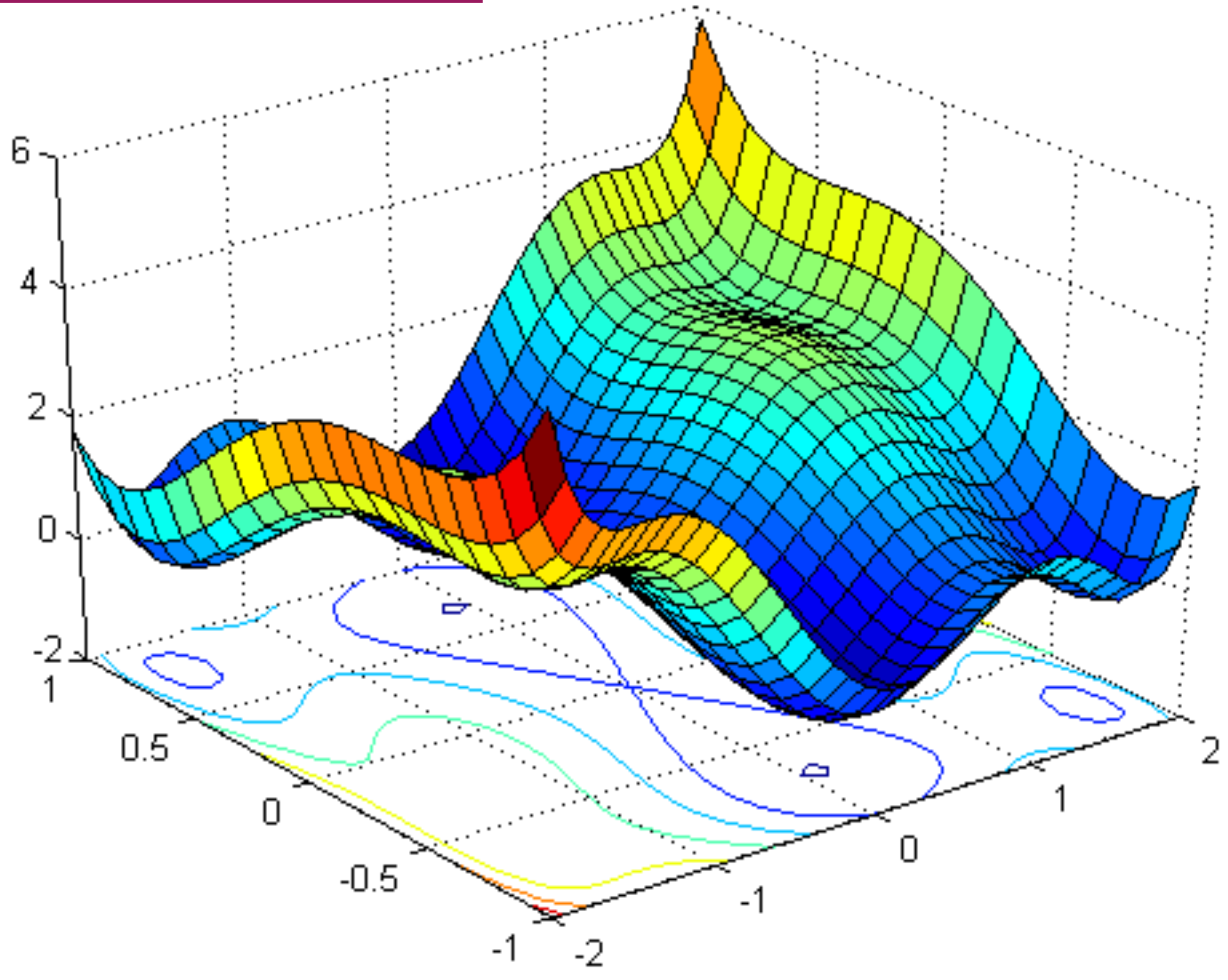
- *MSE is the most popular loss function when dealing with continuous outputs. We will use it a few times in the next days*

- ***BE AWARE OF THE UNDERLYING ASSUMPTION:*** *if you are using MSE, you are implicitly assuming that your y are Gaussian distributed, with fixed RMS*

- *What if the RMS is not a constant?*

Supervised Learning in a nutshell





- A training dataset **x**
- A target *y*
- A model to go from **x** to *y*
- A loss function quantifying how wrong the model is
- A minimisation algorithm to find the model *h* that corresponds to the minimal loss

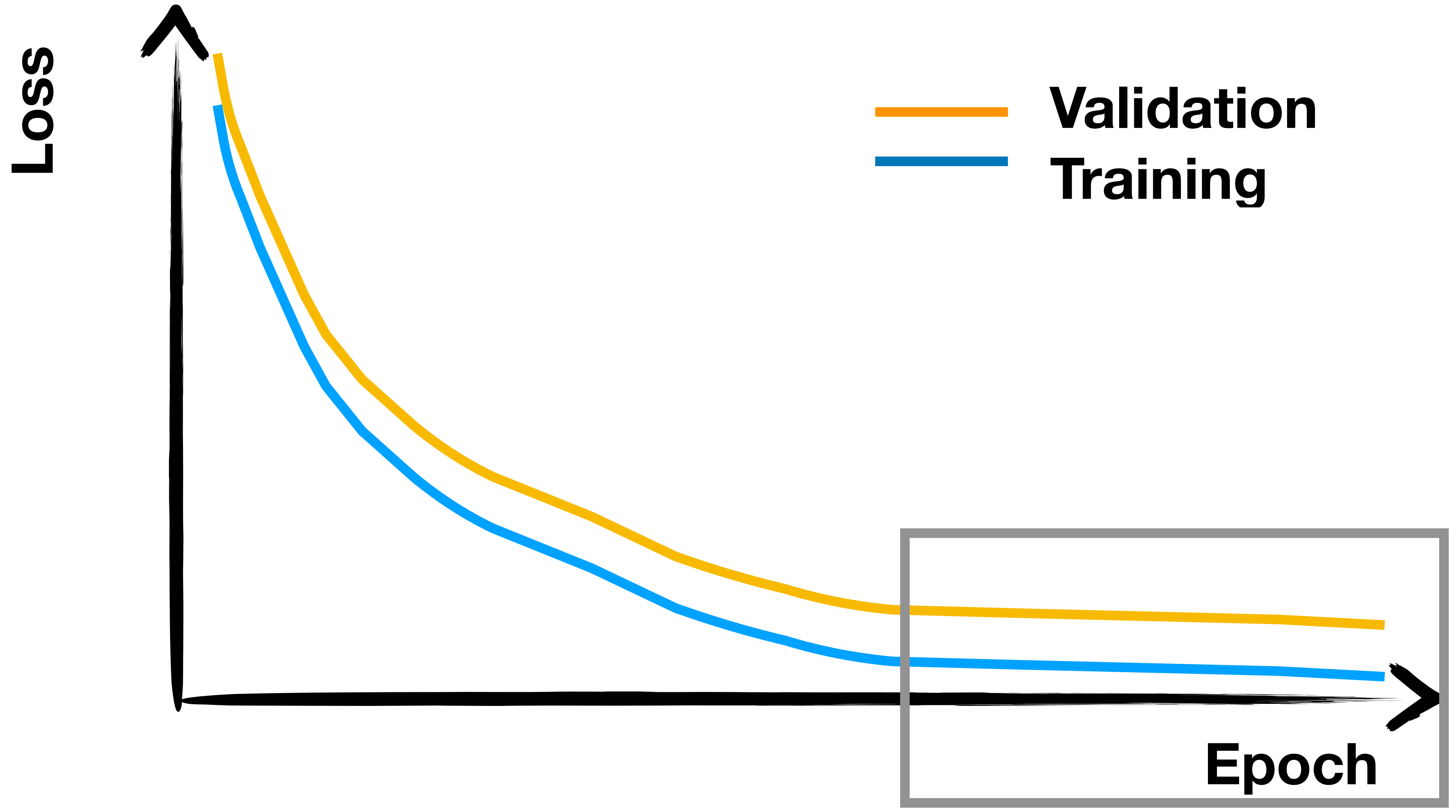
Training in practice

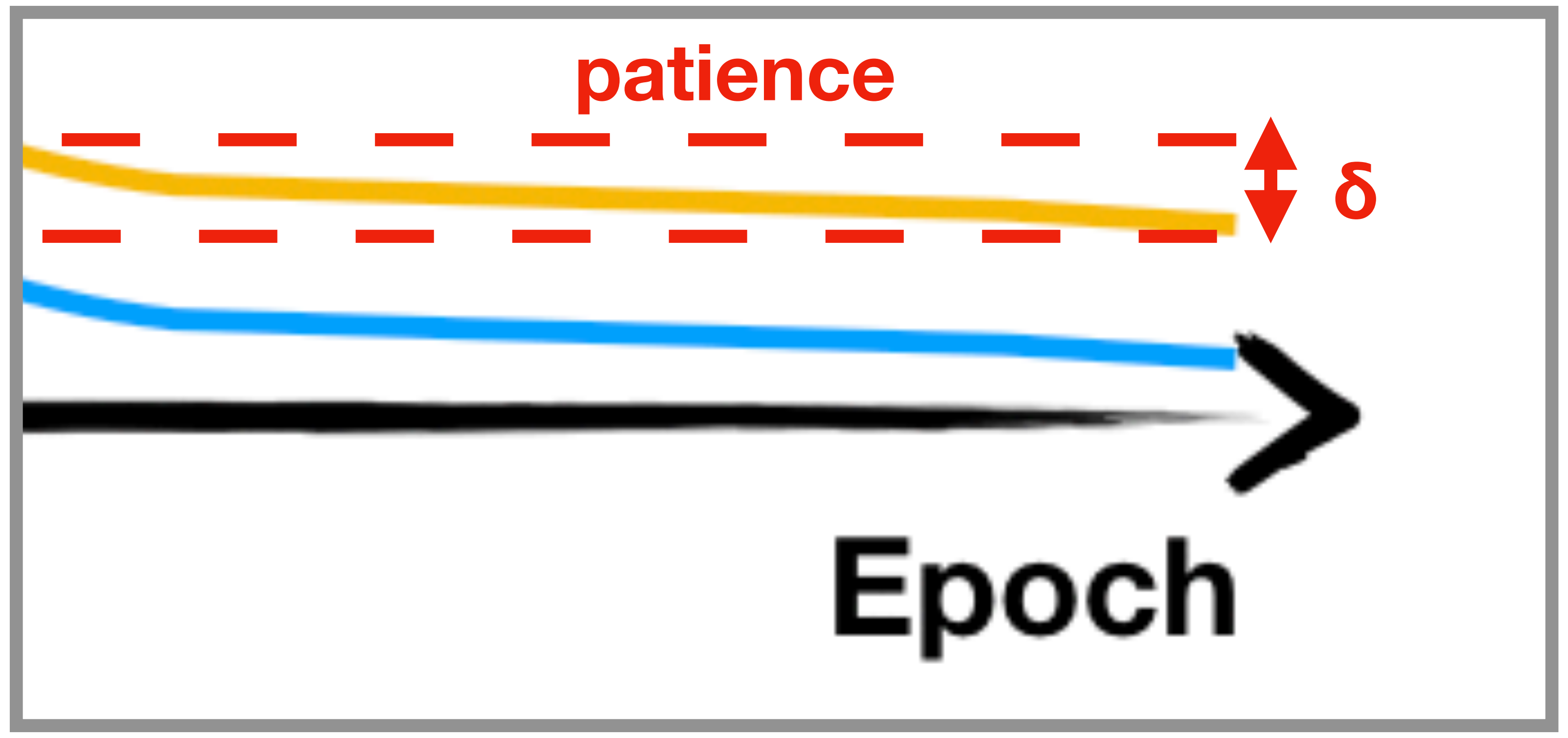
- *Split your sample in three:*
  - *Training: the biggest chunk, where you learn from*
  - *Validation: an auxiliary dataset to verify generalization and prevent overtraining*
  - *Test: the dataset for the final independent check*

Training	Validation	Test

Training in practice

- Train across multiple epochs
  - 1 epoch = going once through the full dataset
- Use small batches (64, 128, etc)
- Check your training history
  - on the training data (training loss)
  - and the validation ones (validation loss)
- Use an objective algorithm to stop (e.g., early stopping)





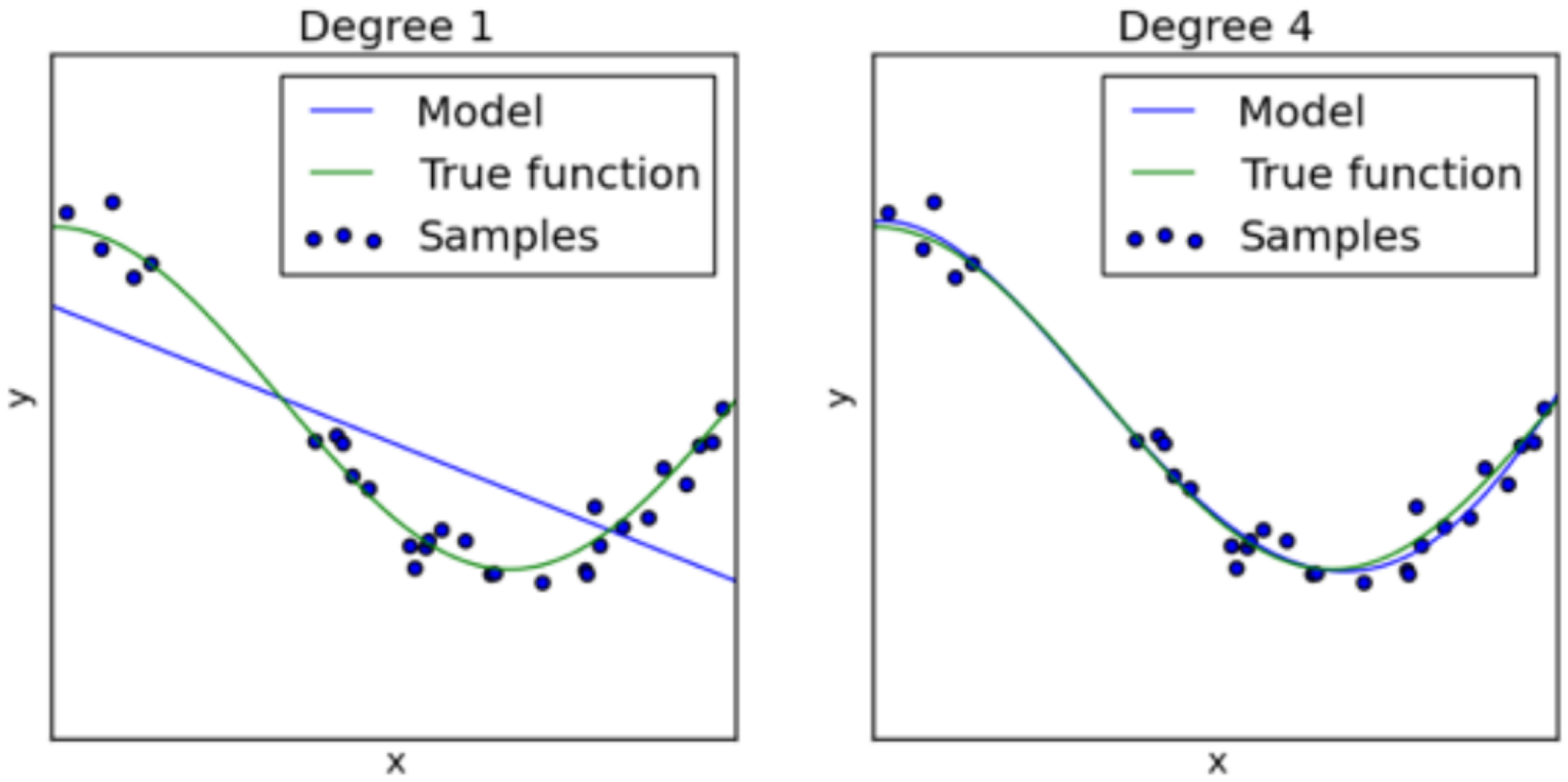
EARLY TOPPING: stop the train if the validation loss didn't change more than δ in the last n epochs (patience)

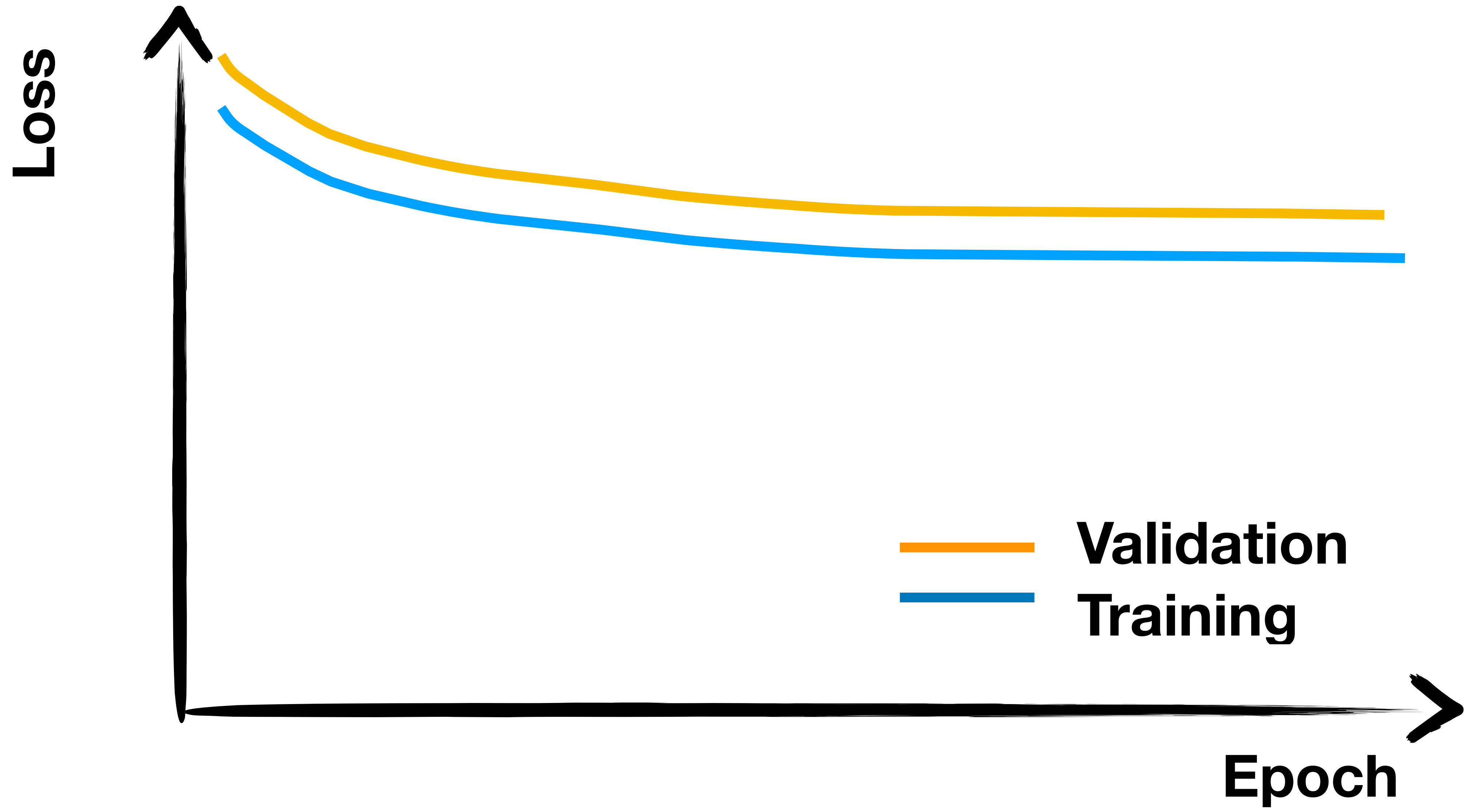
What can go wrong: underfitting

- *If your model has not enough flexibility, it will not be able to describe the data*

- *The training and validation loss will be close, but their value will not decrease*

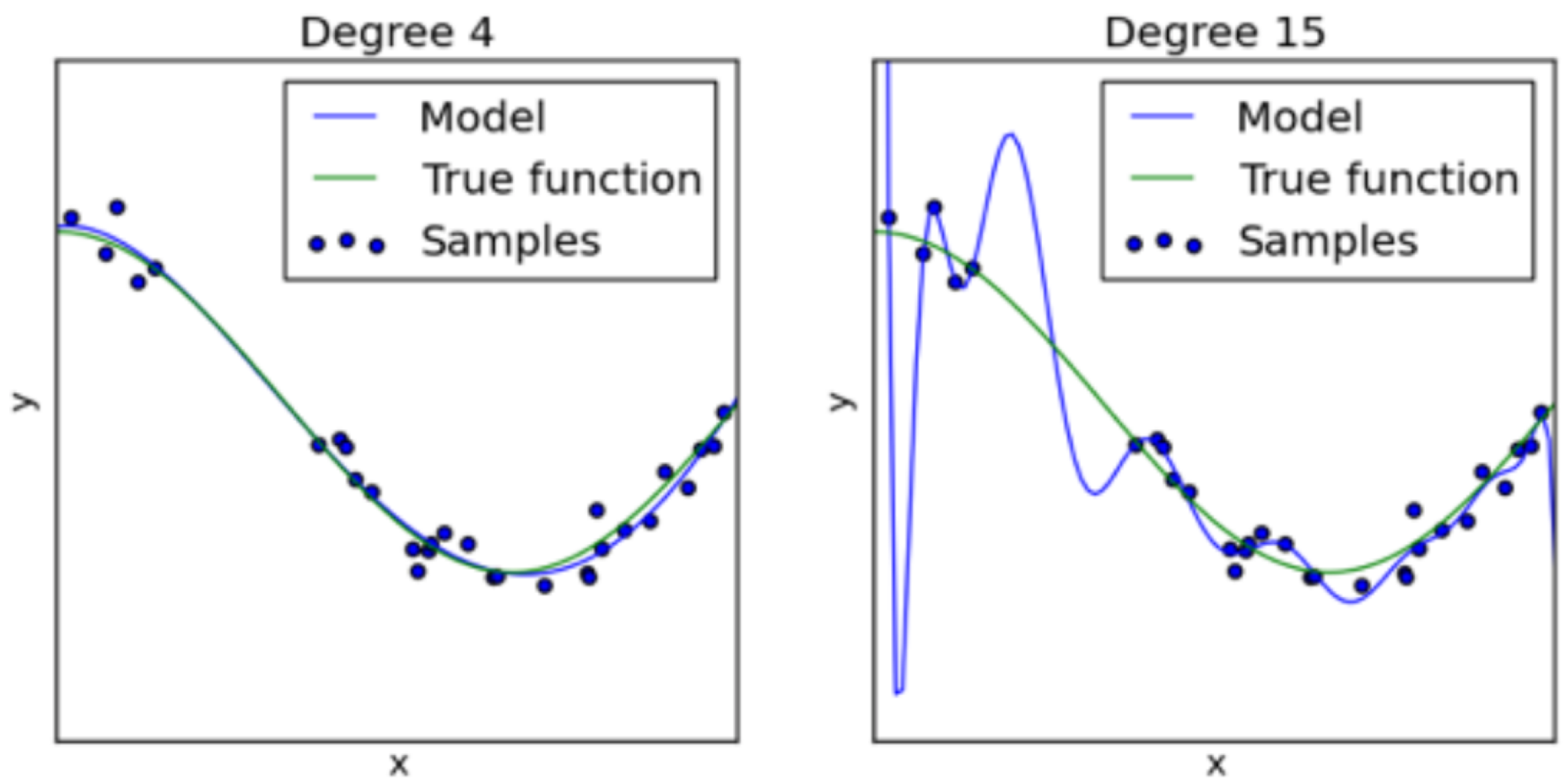
- *The model is said to be underfitting, or being* ***biased***

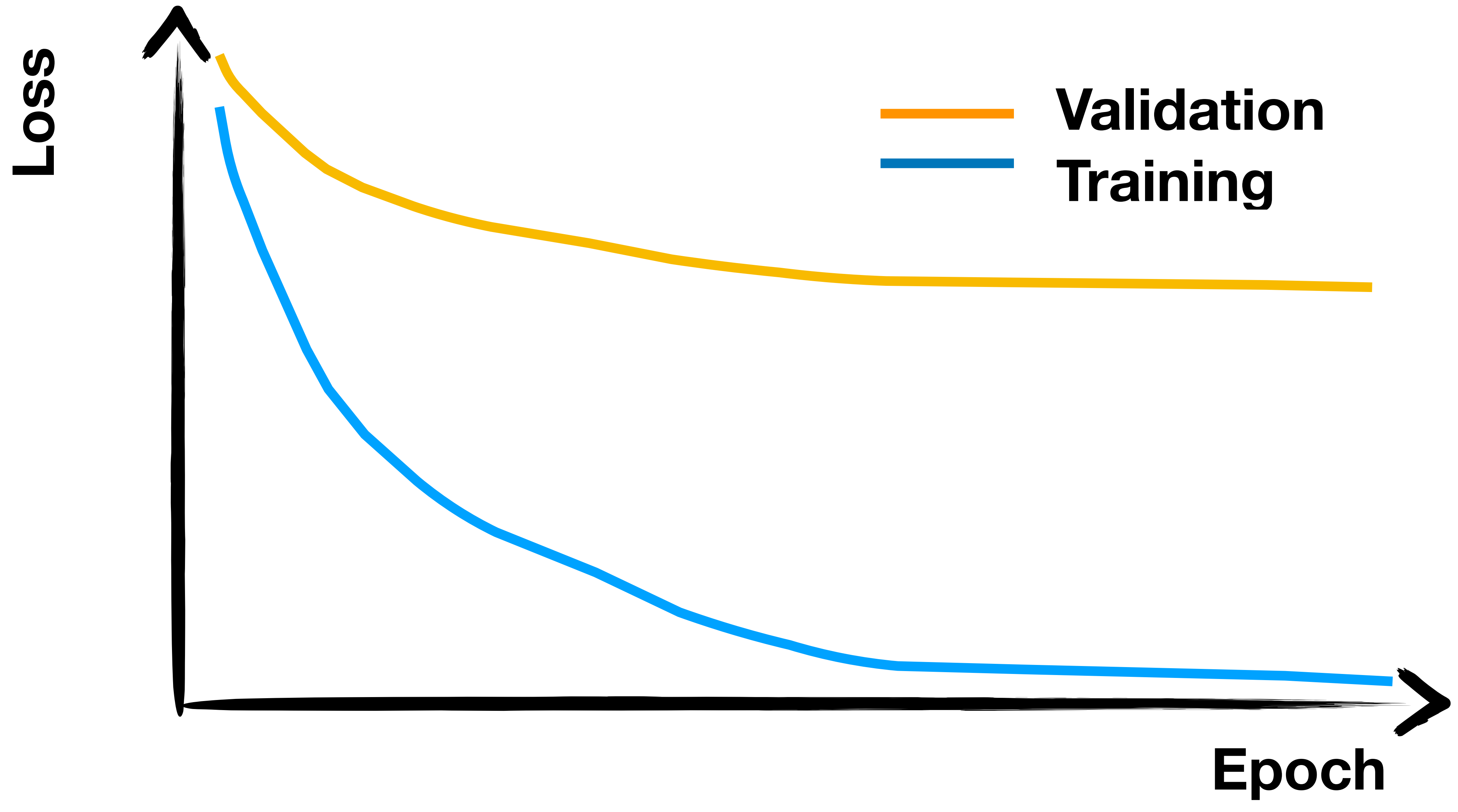




What can go wrong: overfitting

- *Your model can learn too much of your training dataset*
  - *e.g., its statistical fluctuations*
- *Such an overfitted model would not generalise*
- *So, its description of the validation dataset will be bad (i.e., **the mode doesn't generalise**)*
- *This is typically highlighted by a divergence of the training and validation loss*



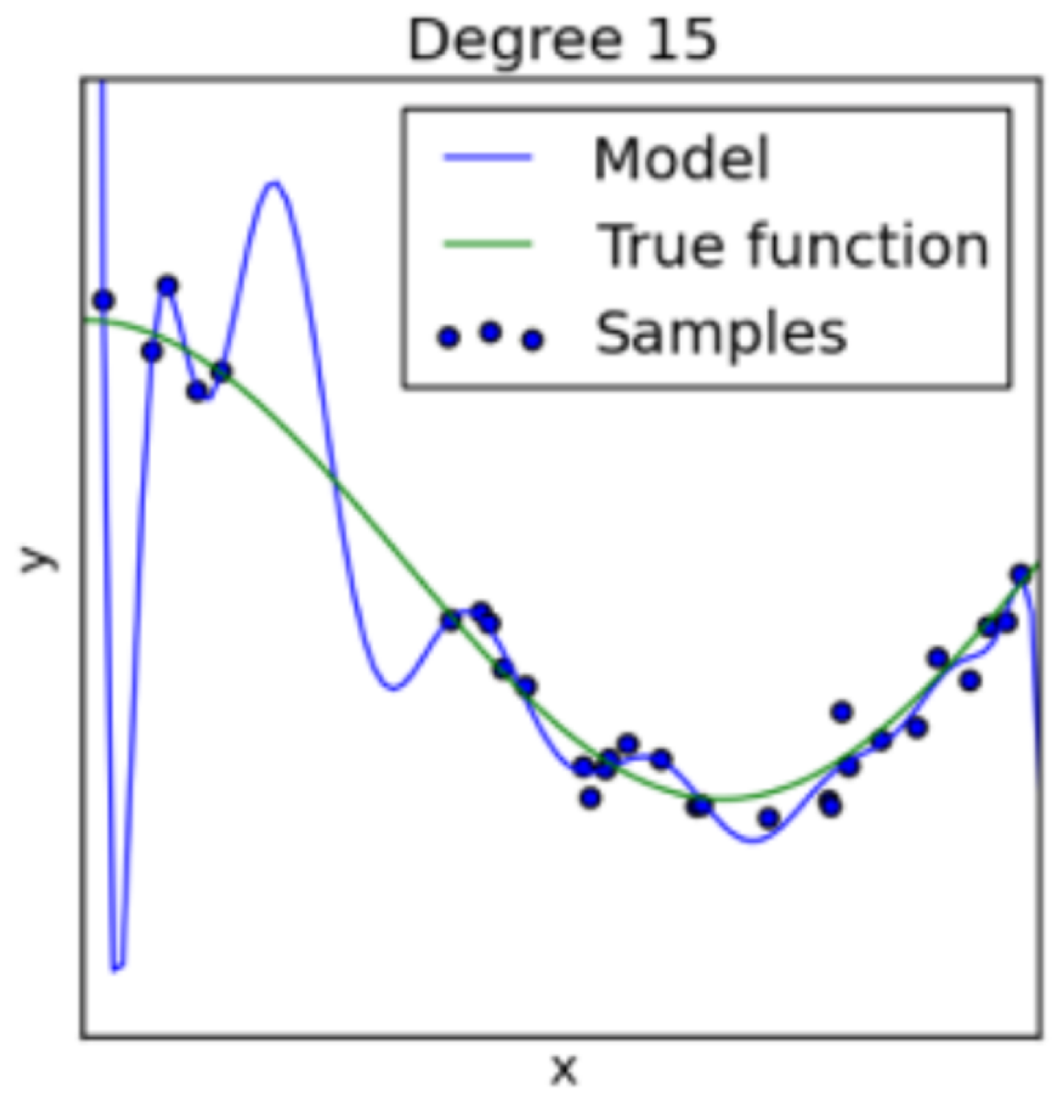


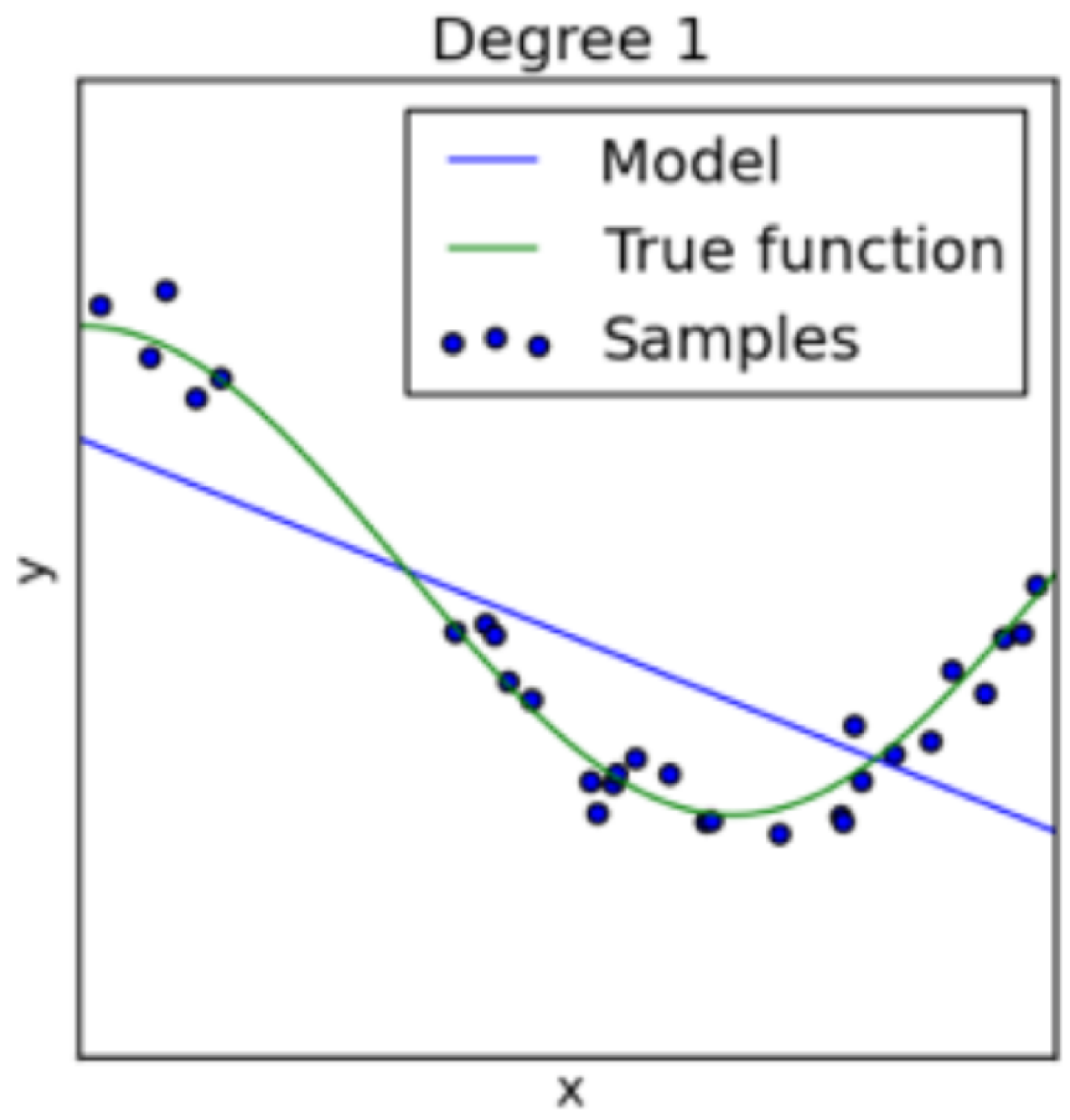
The Bias vs Variance tradeoff

- ***A model would underfit if too simple:*** *it will not be able to model the mean value*

- ***A model would overfit if too complex:*** *it will reproduce the mean value, but it will underestimate the variance of the data*

- *The* ***generalization error*** *is the error made going from the training sample to another sample (e.g., the test sample)*





European Research Council

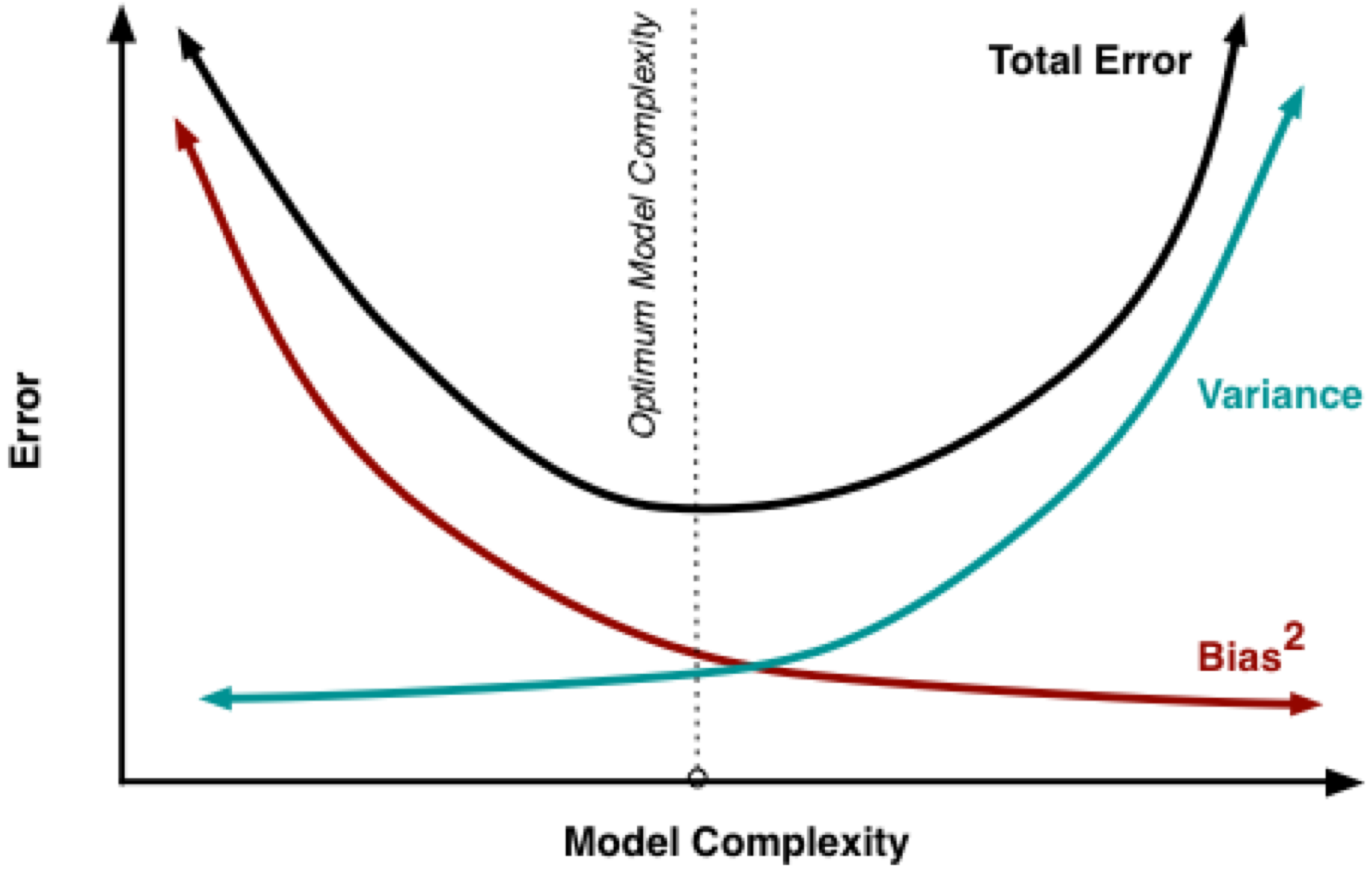
The Bias vs Variance tradeoff

- *Generalization error can be written as the sum of three terms:*
  - *The intrinsic statistical noise in the data*
  - *the bias wrt the mean*
  - *the variance of the prediction around the mean*

$$E[(y-h(x))^2] = E[(y-\bar{y})^2] + (\bar{y}-\bar{h}(x))^2 + E[(h(x)-\bar{h}(x))^2]$$

Noise **Bias Squared** **Variance**

The Bias vs Variance tradeoff



Regularization

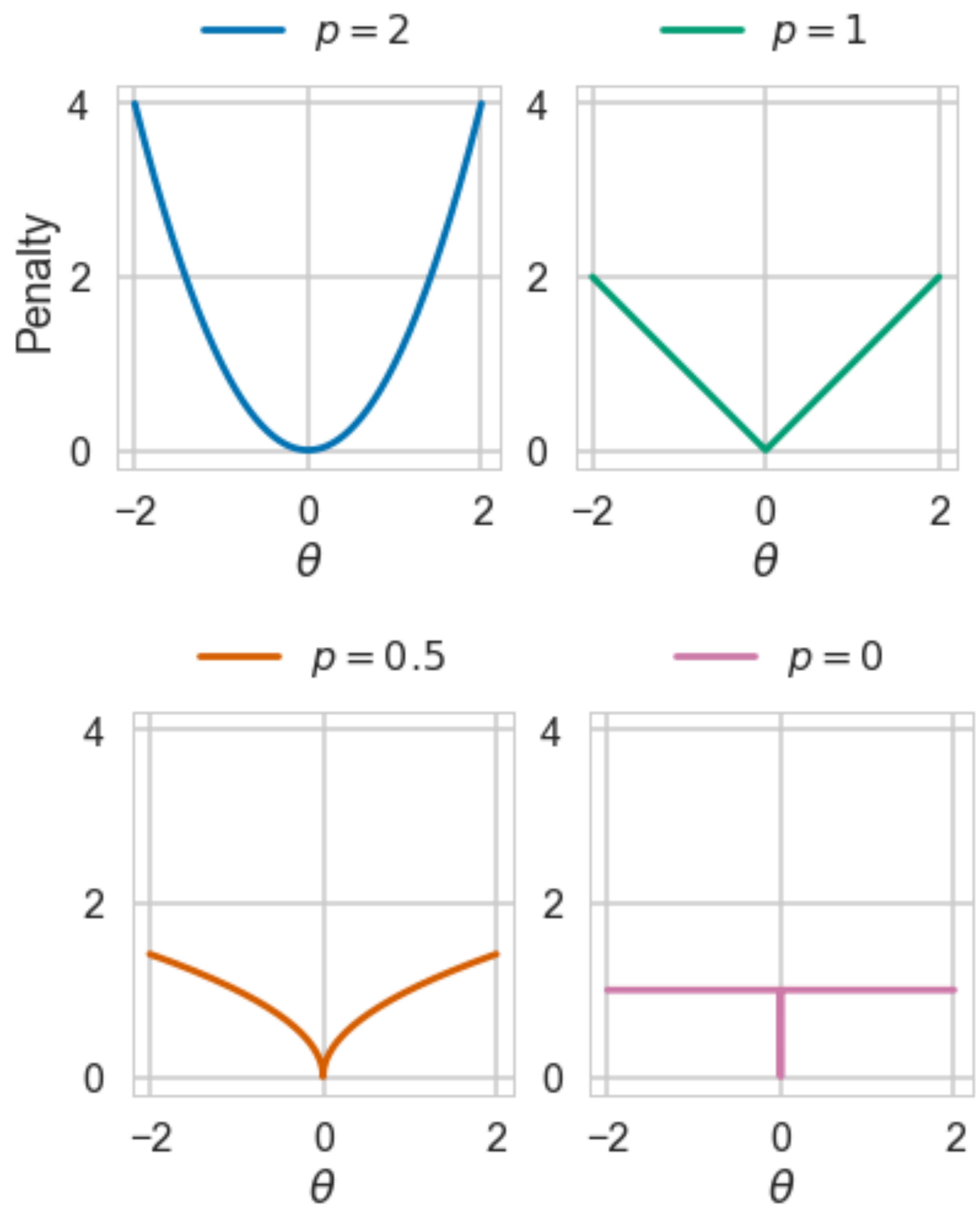
- *Model complexity can be "optimized" when minimizing the loss*
  - *A modified loss is introduced, with a penalty term attache to each model parameter*

$$L_{reg} = L + \Omega(w)$$

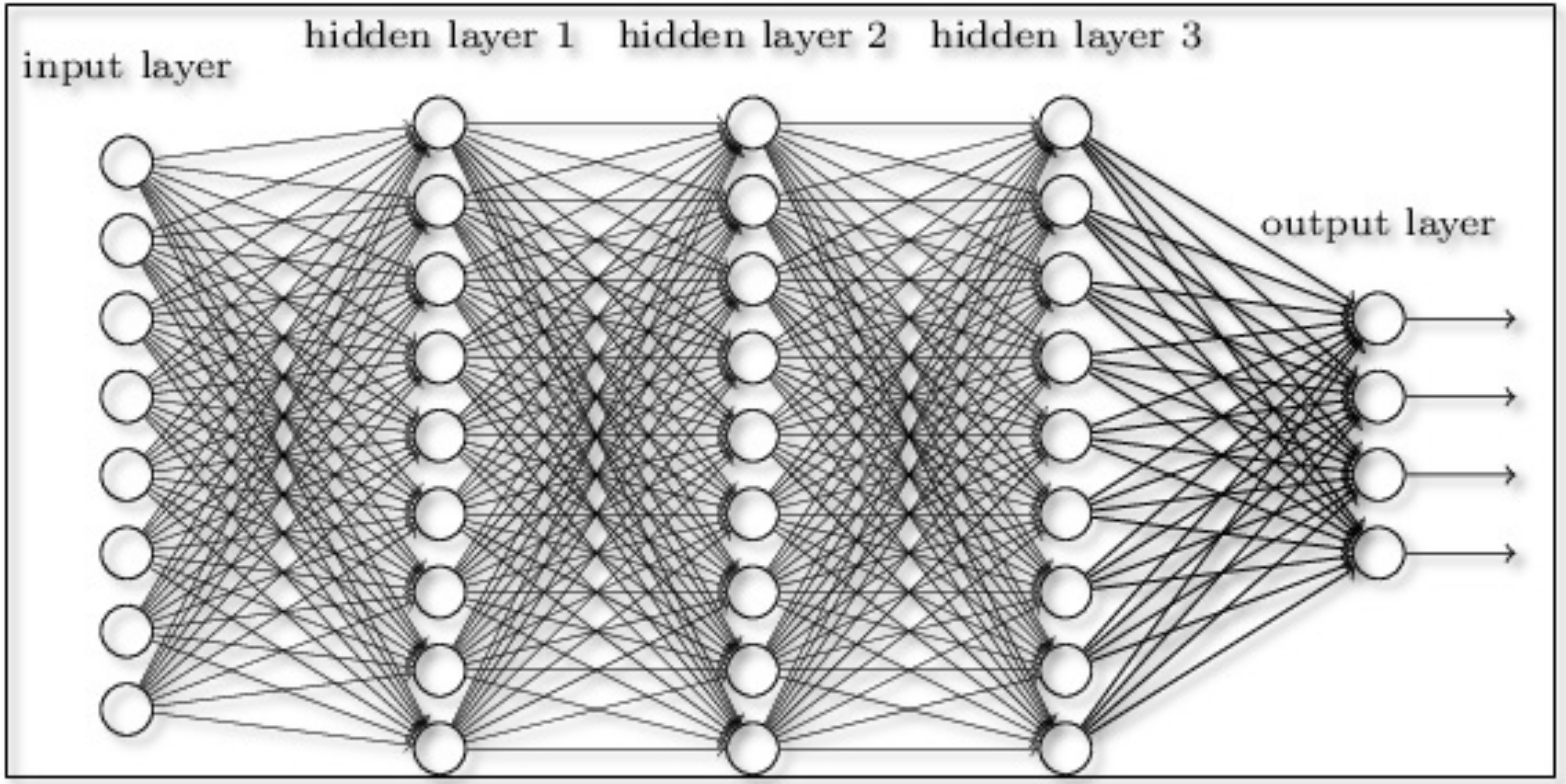
- *For instance, Lp regularisation*

$$L_p = \|\mathbf{w}\|^p = \sum_i |w_i|^p$$

- *The minimisation is a tradeoff between:*
  - *pushing down the 1st term by taking advantage of the parameters*
  - *pushing down the 2nd term by switching off the parameters*



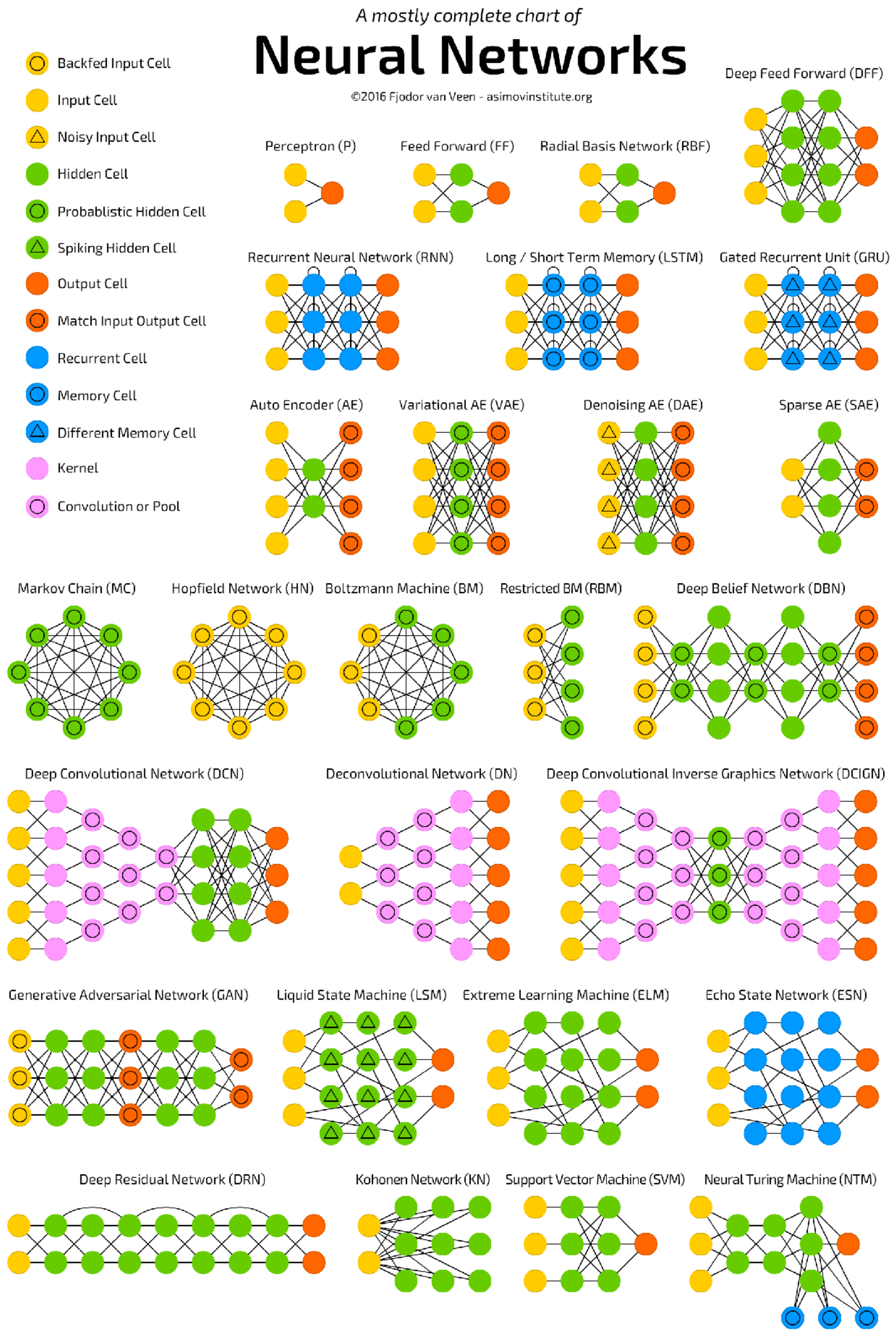
https://openreview.net/pdf?id=H1Y8hhg0b



Deep Learning

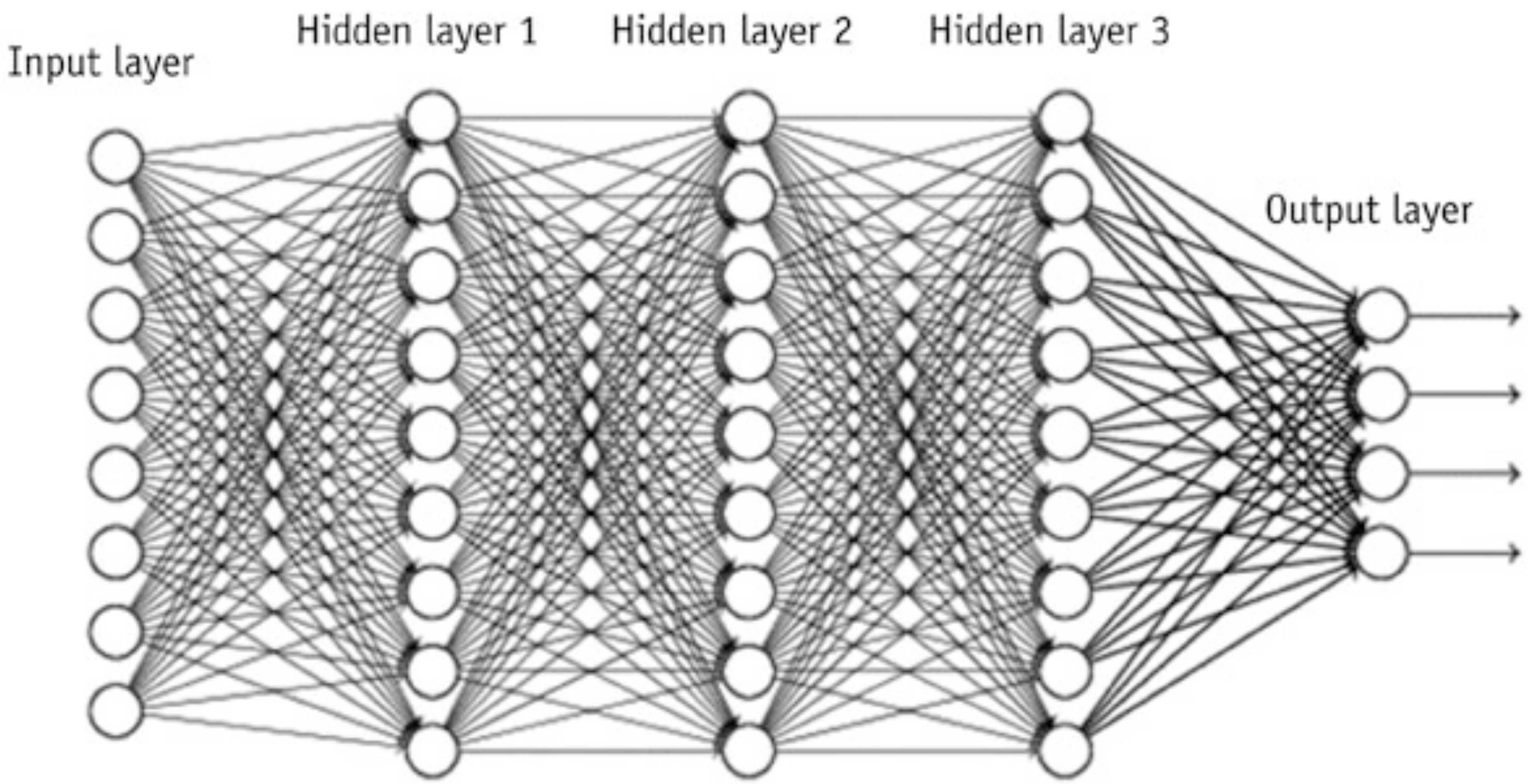
Neural Networks in a nutshell

- NNs are (as of today) the best ML solution on the market
- NNs are usually structured in nodes connected by edges
    - each node performs a math operation on the input
    - edges determine the flow of neuron's inputs & outputs



Deep Neural Networks

- *Deep neural networks are those with >1 inner layer*
- *Thanks to GPUs, it is now possible to train them efficiently, which boosted the revival of neural networks in the years 2000*
- *In addition, new architectures emerged, which better exploit the new computing power*



Large-scale Deep Unsupervised Learning using Graphics Processors

Rajat Raina RAJATR@CS.STANFORD.EDU
Anand Madhavan MANAND@STANFORD.EDU
Andrew Y. Ng ANG@CS.STANFORD.EDU
Computer Science Department, Stanford University, Stanford CA 94305 USA

European Research Council

What is DL used for

Image processing

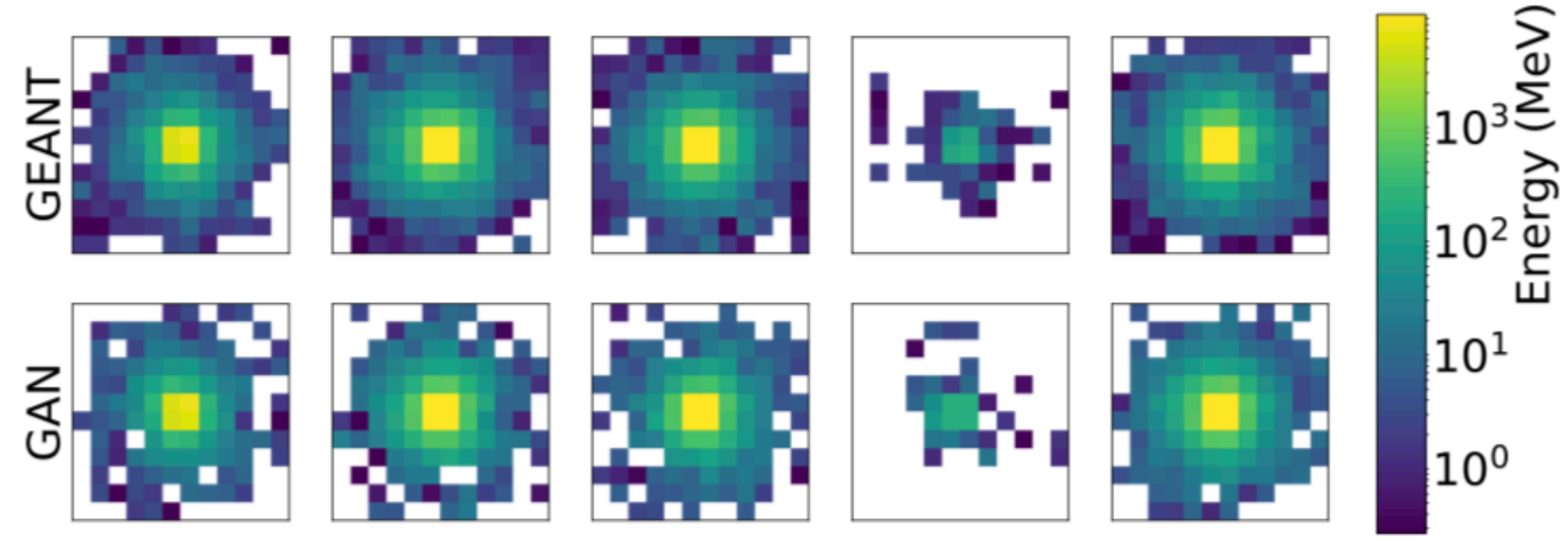
text/sound processing

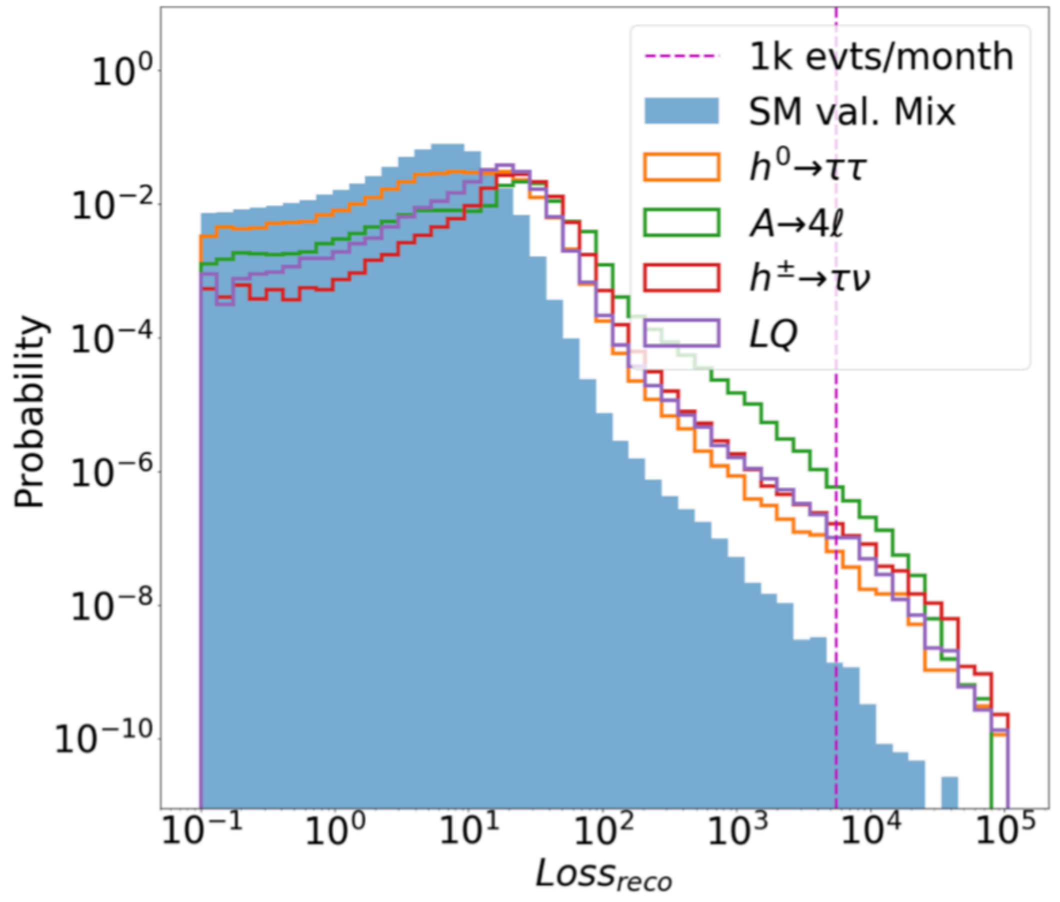
Reinforcement Learning

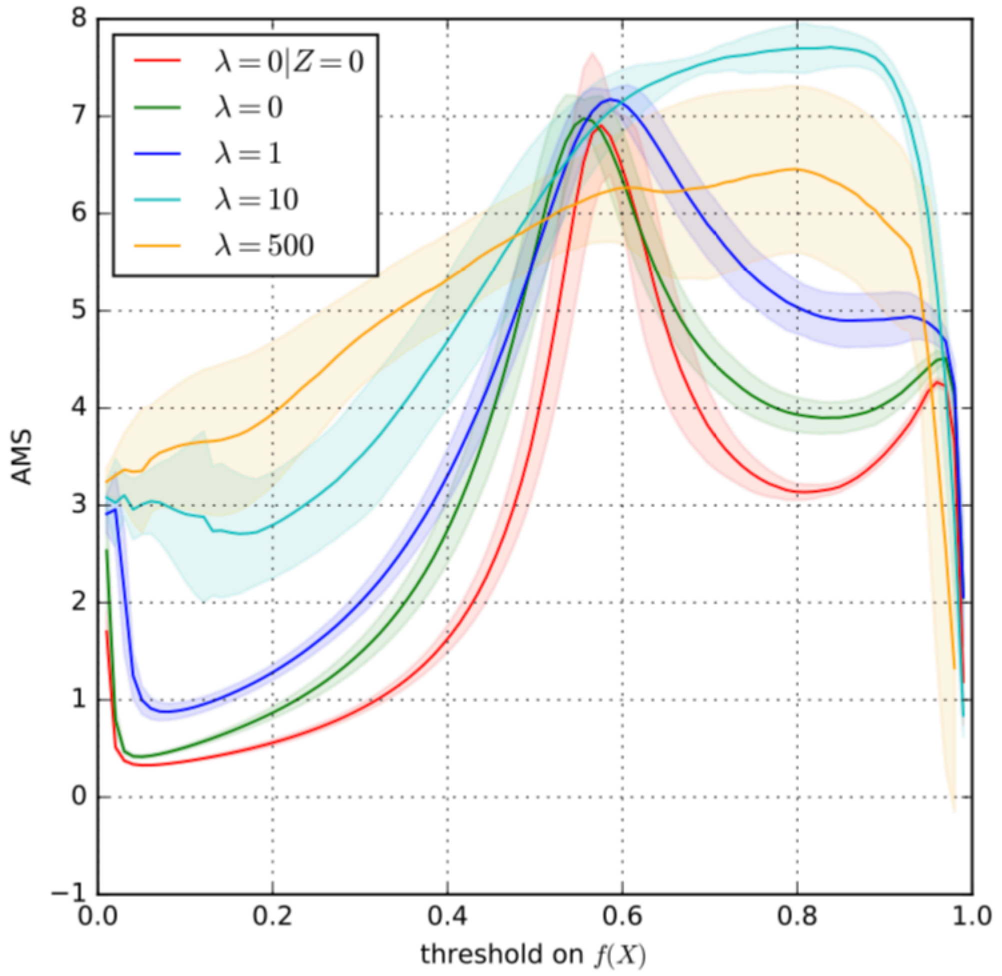
Clustering

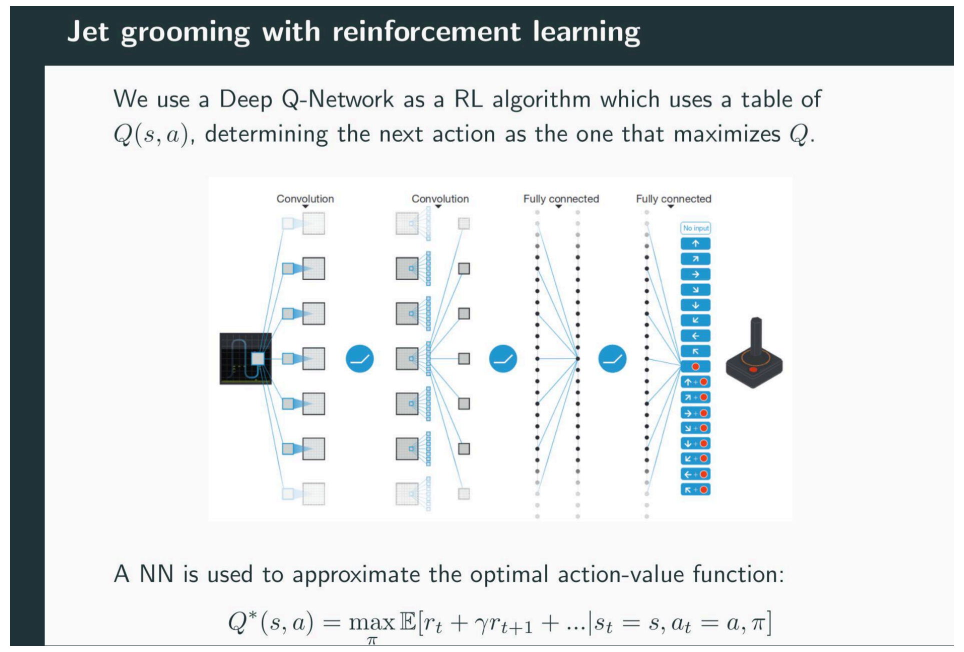
DL, HEP, and new opportunities

- *Event Generation with generative models*
- *Anomaly Detection to search for new Physics*
- *Adversarial training for systematics*
- *Reinforcement learning for jet grooming*
- ...



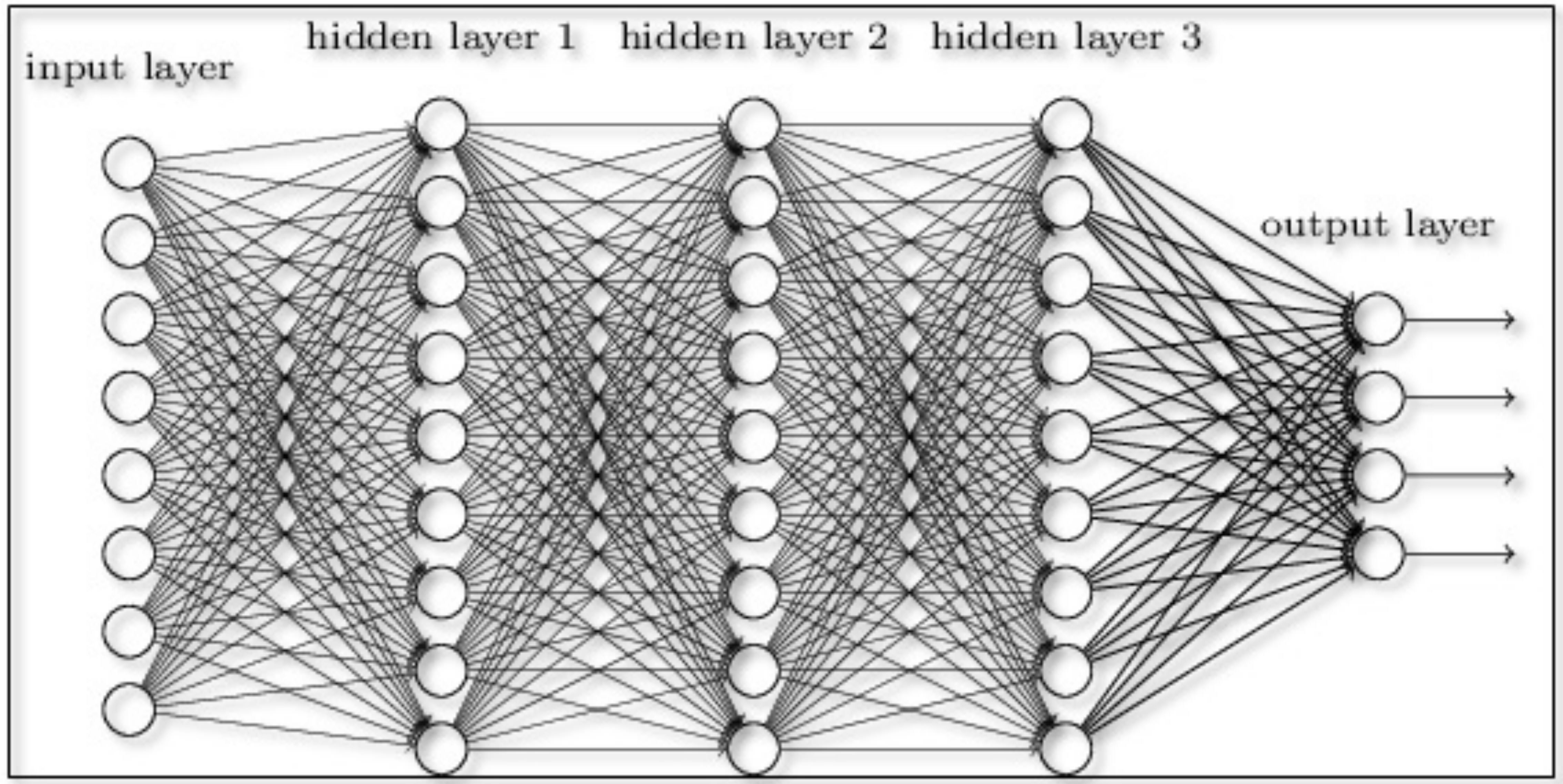






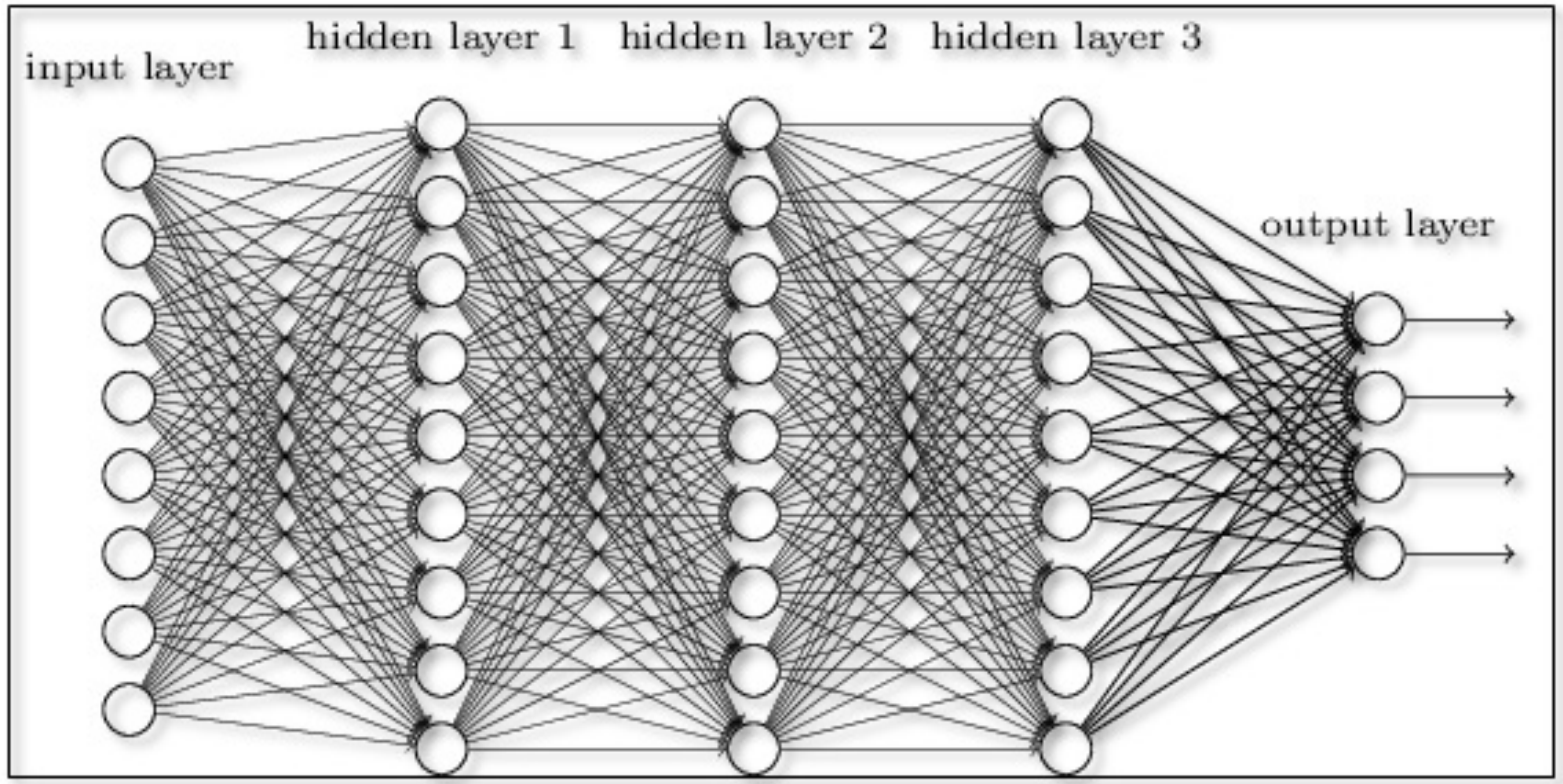
Feed-Forward NNs

- Feed-forward neural networks have hierarchical structures:
  - inputs enter from the left and flow to the right
  - no closed loops or circularities
- Deep neural networks are FF-NN with more than one hidden layer
- Out of this “classic idea, new architectures emerge, optimised for computing vision, language processing, etc



The role of a network node

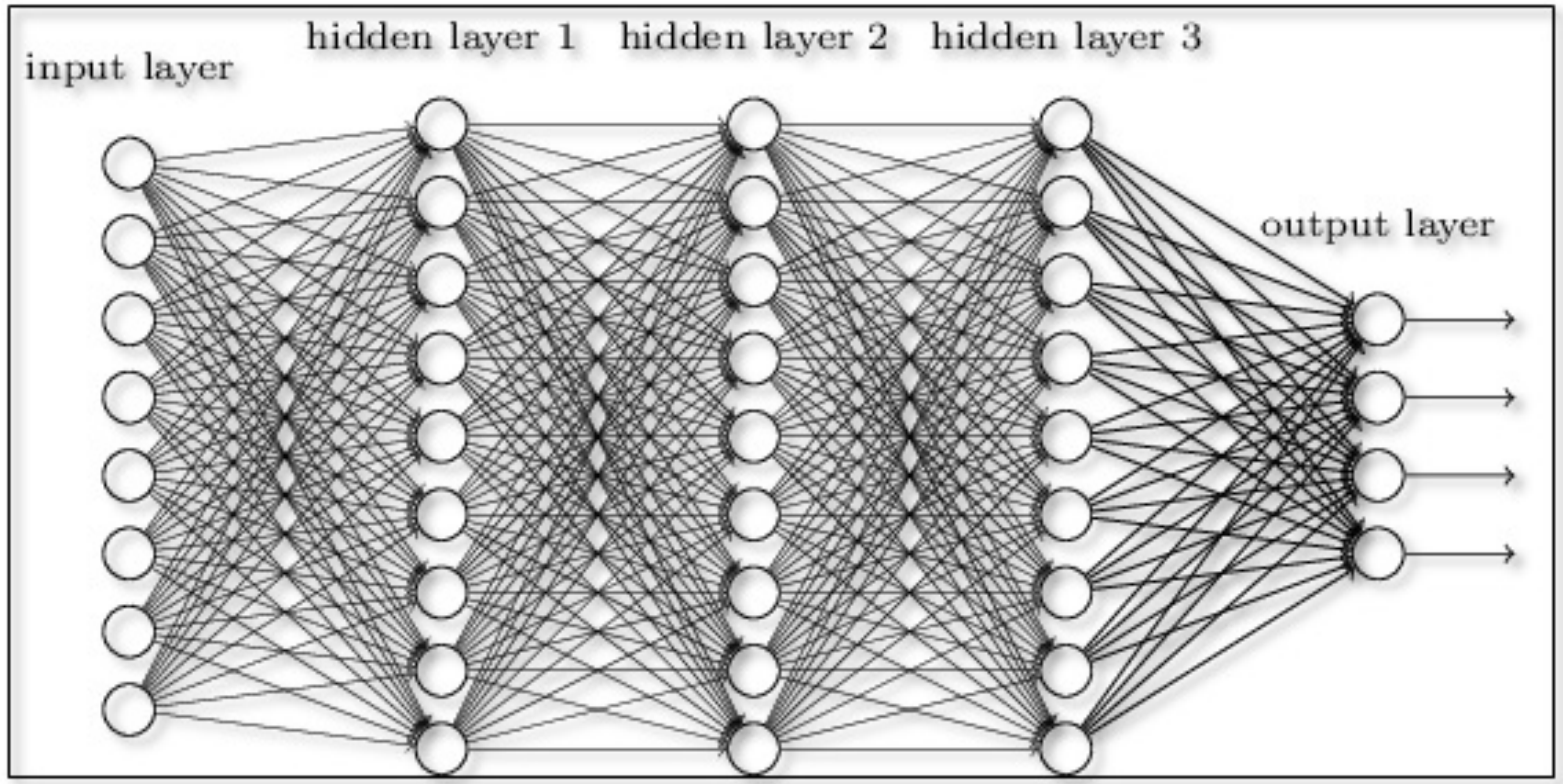
- **Each input is multiplied by a weight**
- The weighted values are summed
- A bias is added
- The result is passed to an activation function



$$w_{ij}x_j$$

The role of a network node

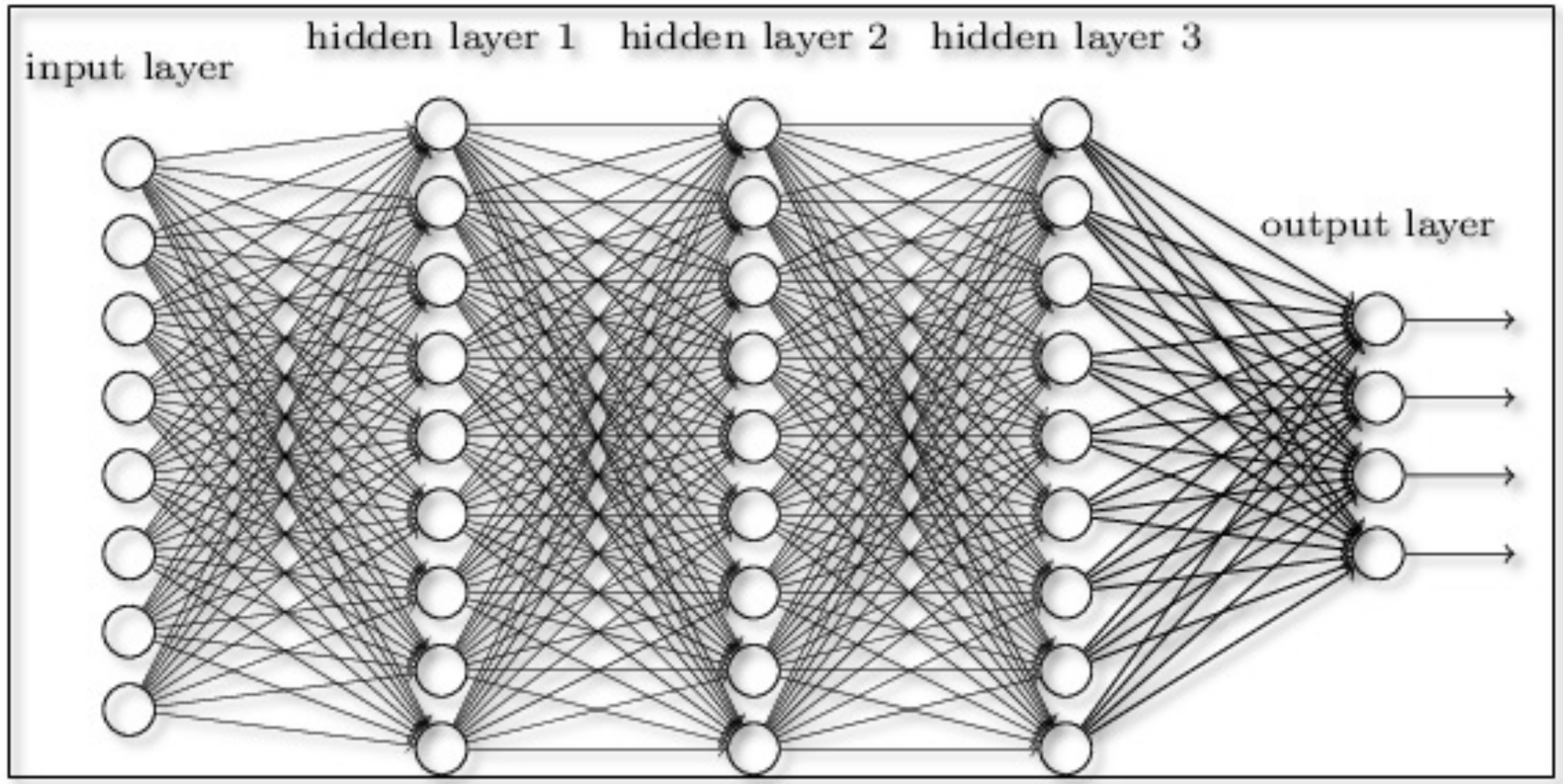
- Each input is multiplied by a weight
- **The weighted values are summed**
- A bias is added
- The result is passed to an activation function



$$\Sigma_j w_{ij} x_j$$

The role of a network node

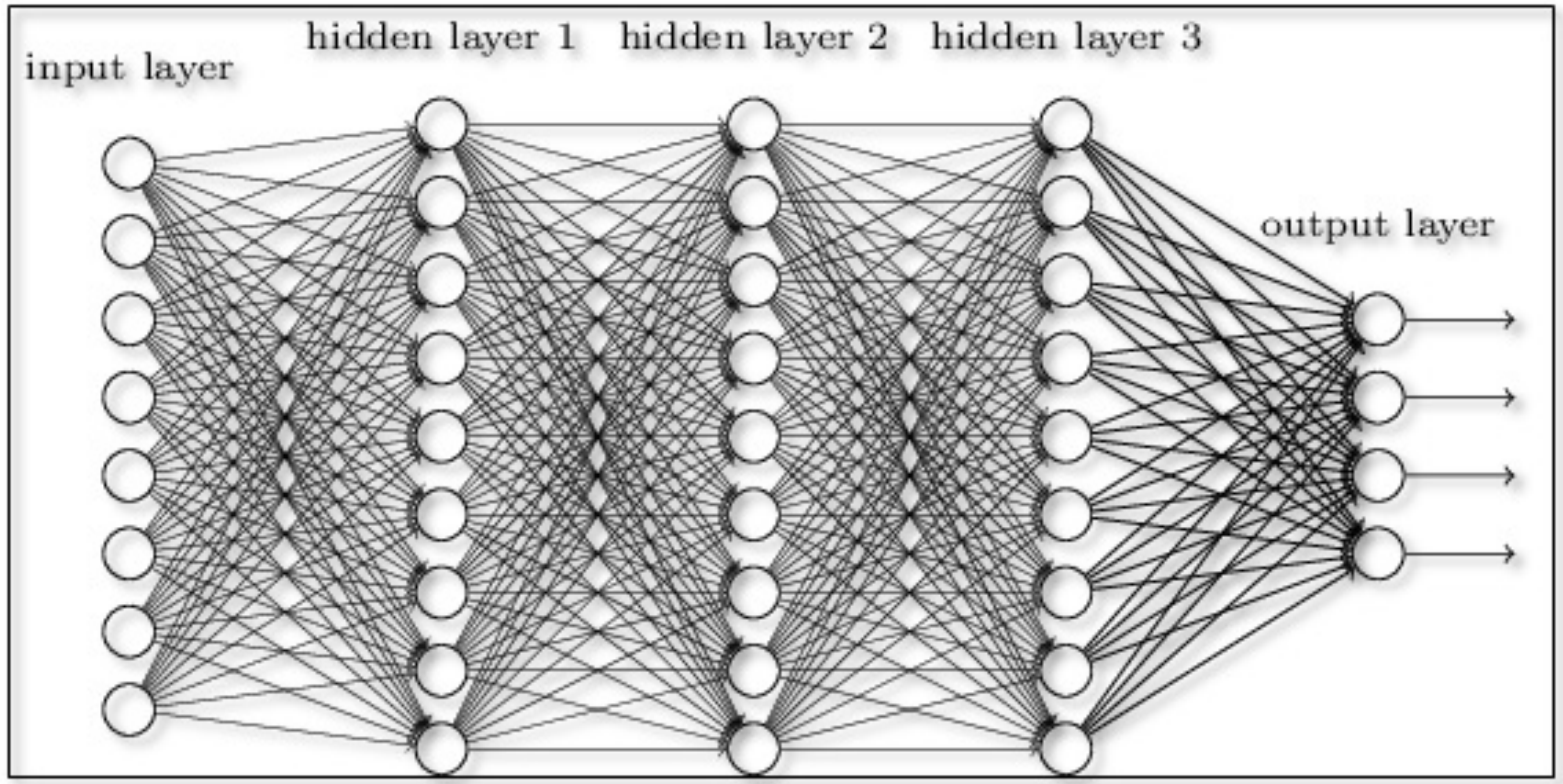
- Each input is multiplied by a weight
- The weighted values are summed
- **A bias is added**
- The result is passed to an activation function



$$\Sigma_j w_{ij} x_j + b_i$$

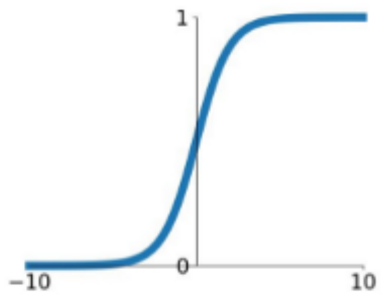
The role of a network node

- Each input is multiplied by a weight
- The weighted values are summed
- A bias is added
- **The result is passed to an activation function**

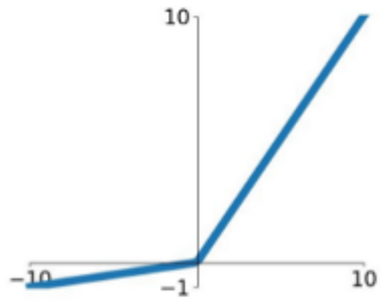


Activation Functions

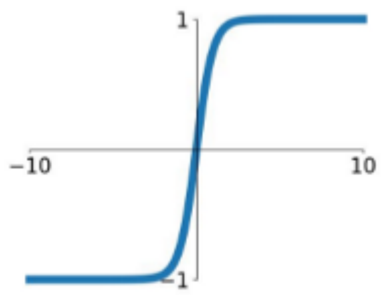
Sigmoid
$\sigma(x) = \frac{1}{1+e^{-x}}$



Leaky ReLU
$\max(0.1x, x)$

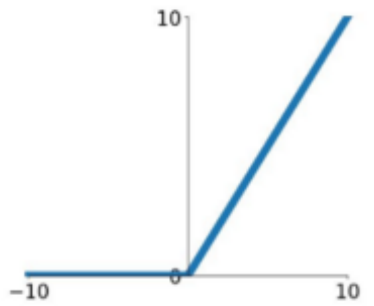


tanh
$\tanh(x)$

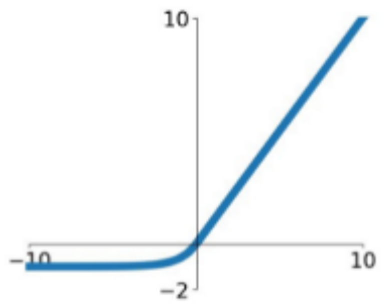


Maxout
$\max(w_1^T x + b_1, w_2^T x + b_2)$

ReLU
$\max(0, x)$



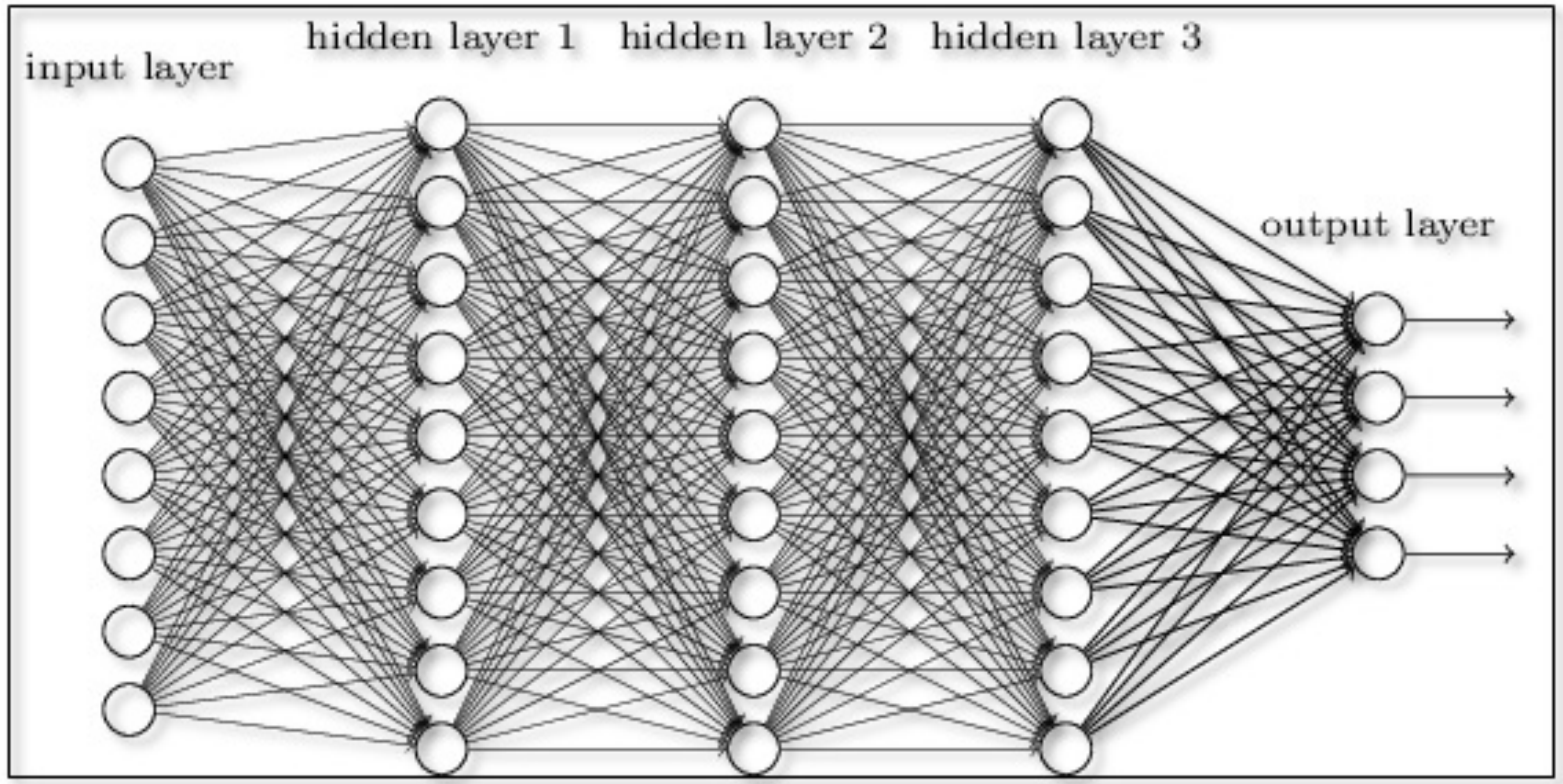
ELU
$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$



$$y_i = f(\Sigma_j w_{ij} x_j + b_i)$$

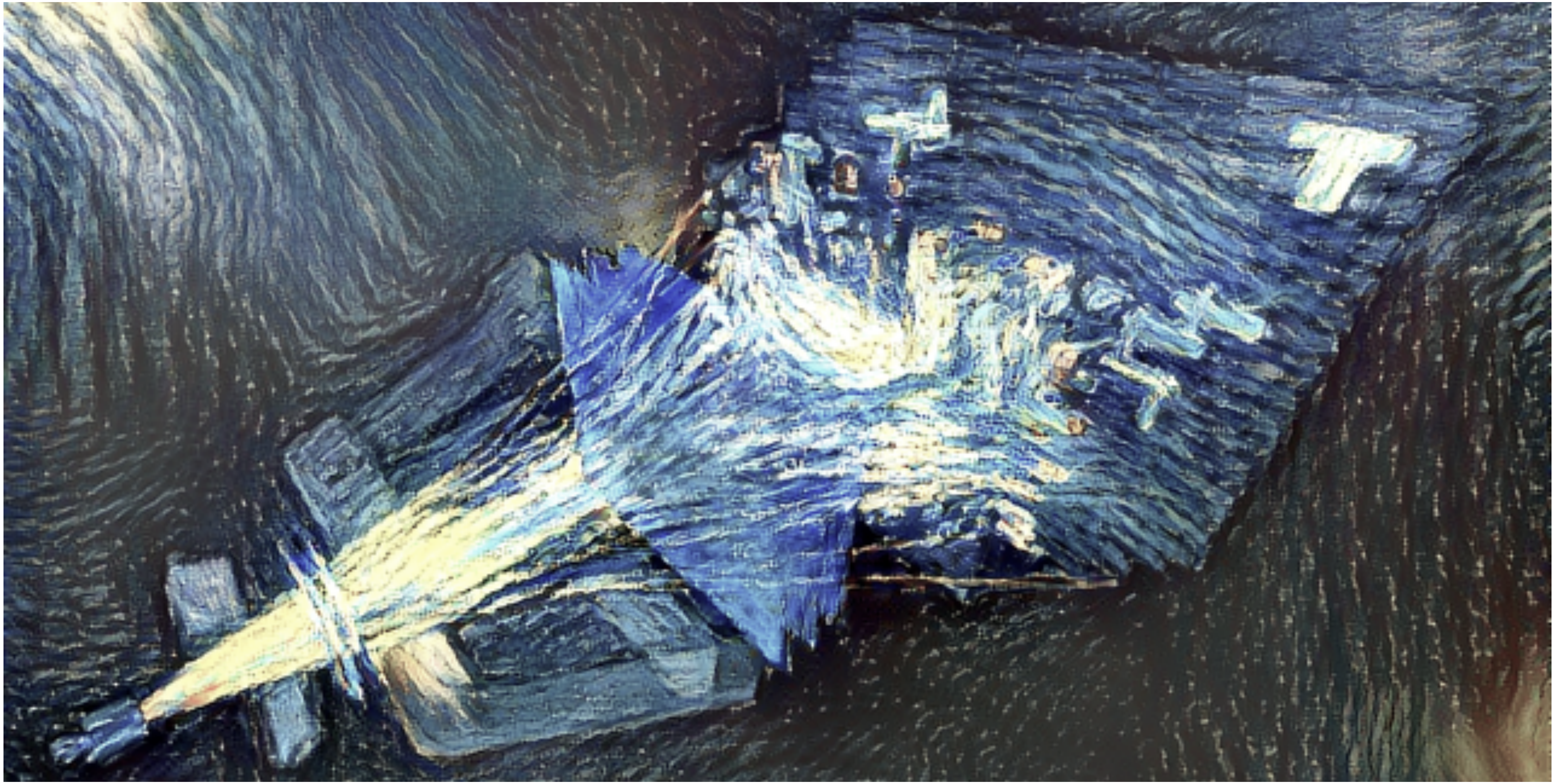
The full picture

- *In a feed-forward chain, each node processes what comes from the previous layer*
- *The final result (depending on the network geometry) is K outputs, given N inputs*



$$y_j = f^{(3)}(\Sigma_l w_{jl}^{(3)} f^{(2)}(\Sigma_k w_{lk}^{(2)} f^{(1)}(\Sigma_i w_{ki}^{(1)} x_i + b_k^{(1)}) + b_l^{(2)}) + b_j^{(3)})$$

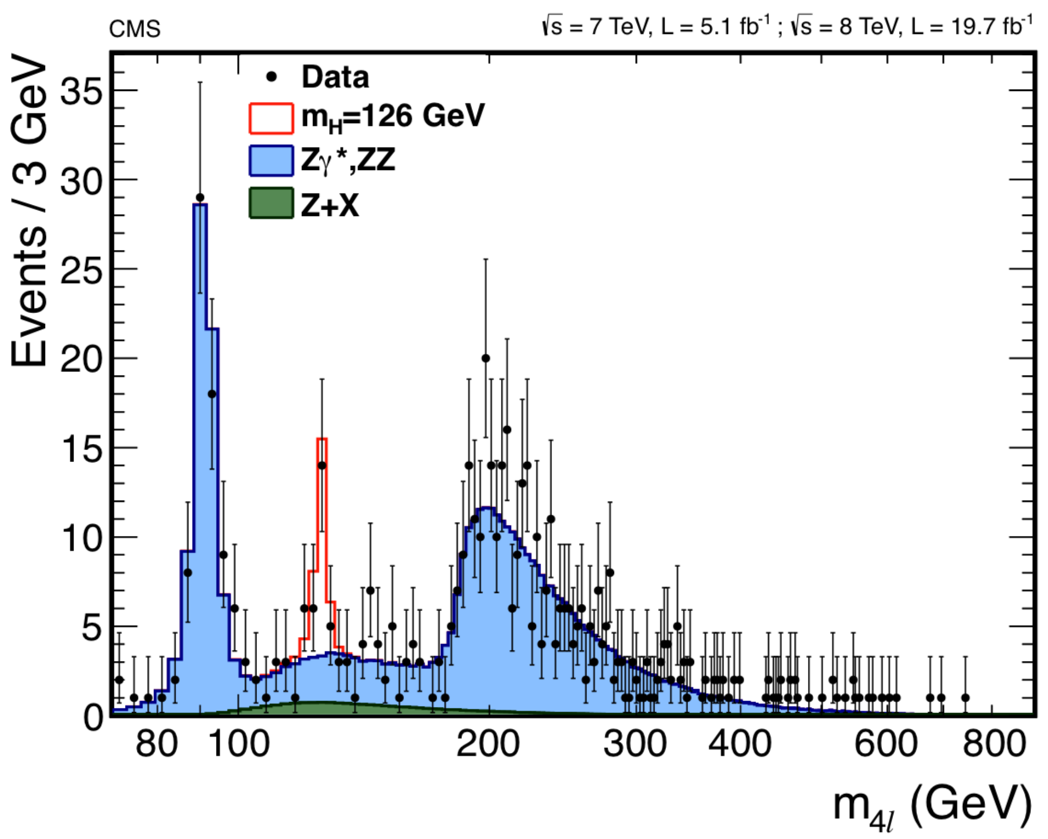
- *One can show that such a mechanism allows to learn generic* $\mathbb{R}^N \rightarrow \mathbb{R}^K$ *functions*

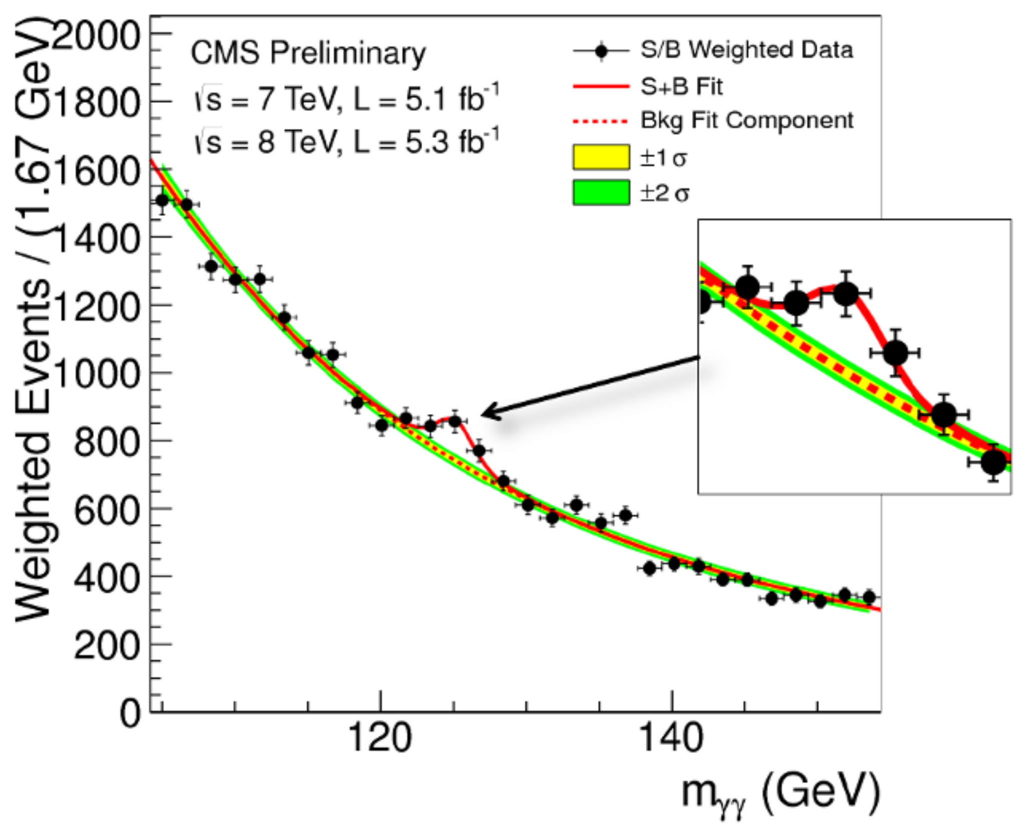


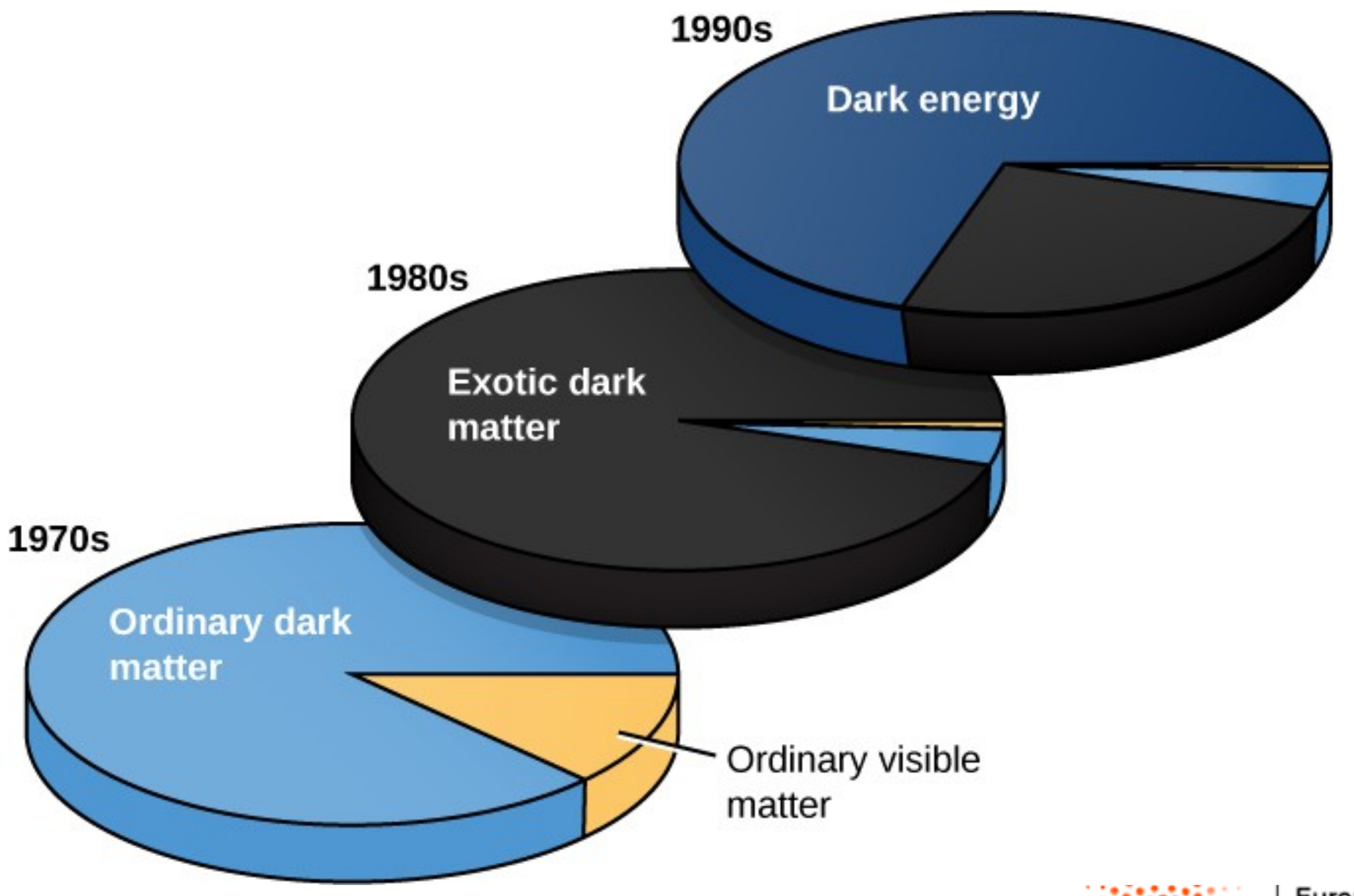
Why DeepLearning for HEP?

LHC: Energy frontier exploration

- *Discover the Higgs boson or exclude its existence*
- *Help answering the big questions left in particle physics*
  - *What stabilises physics at EW scale?*
  - *What's the nature of Dark Matter?*
  - *Origin of cosmological matter/antimatter asymmetry*
  - *Are there unexpected phenomena at the energy frontier?*

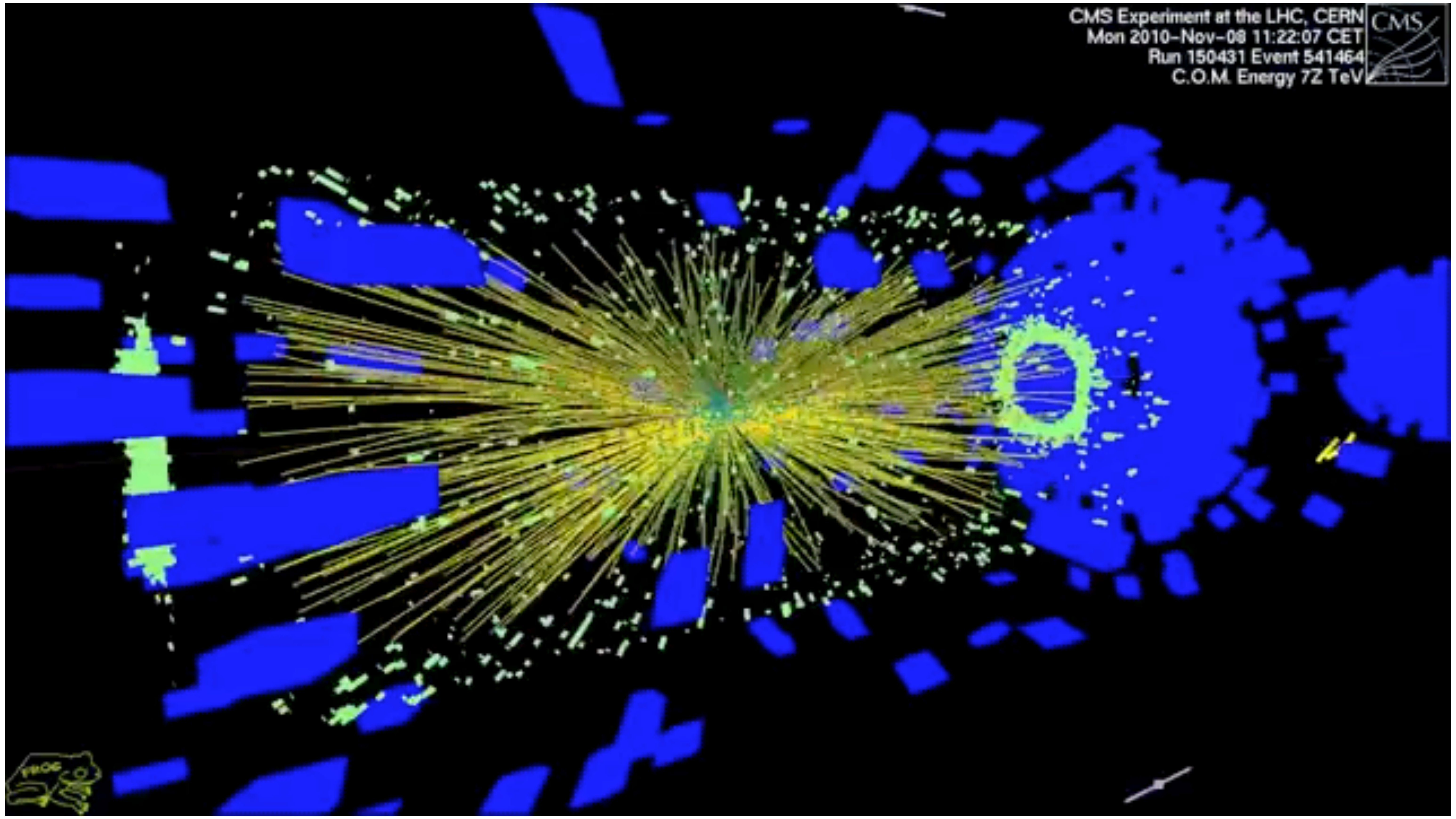




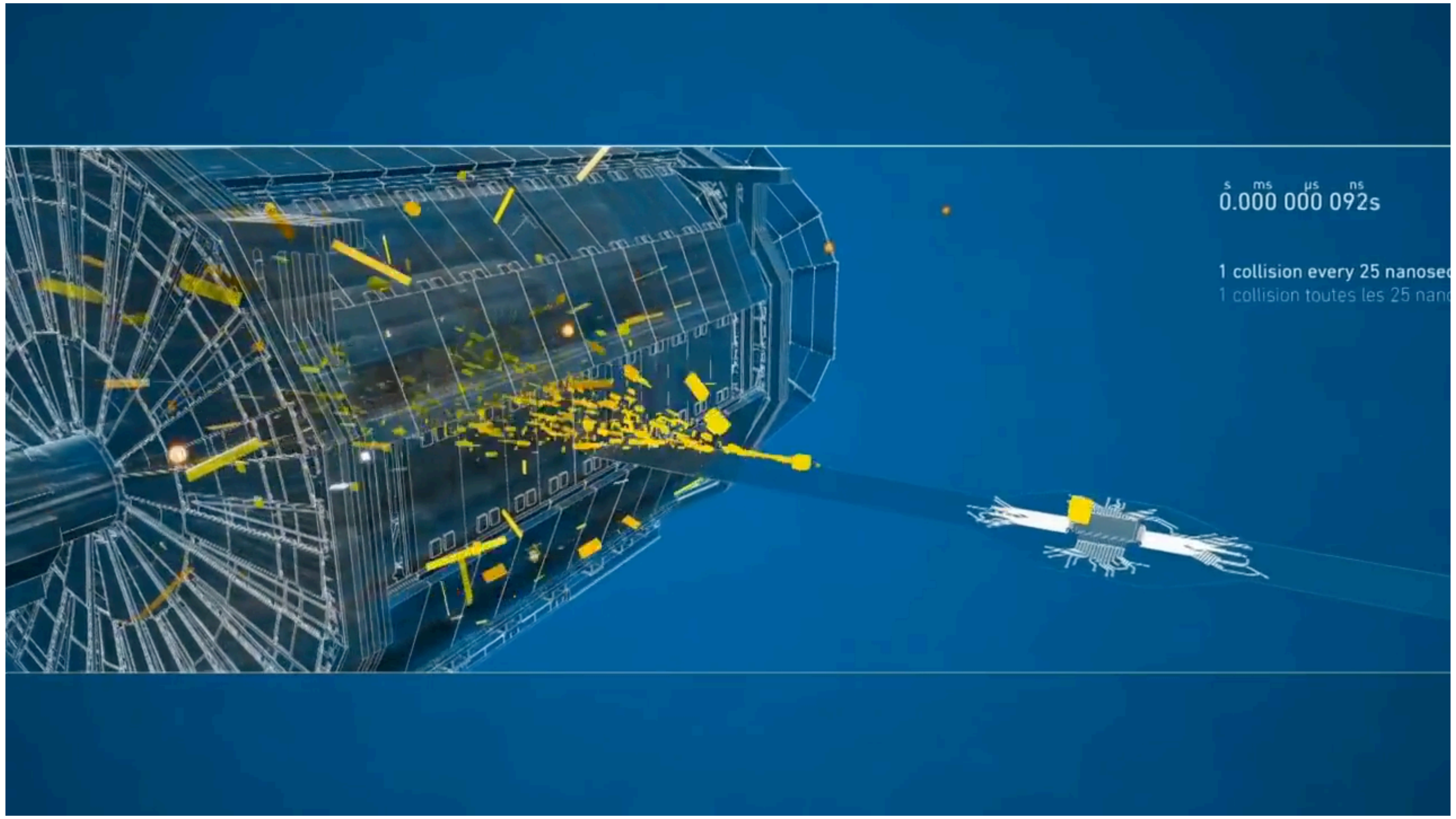


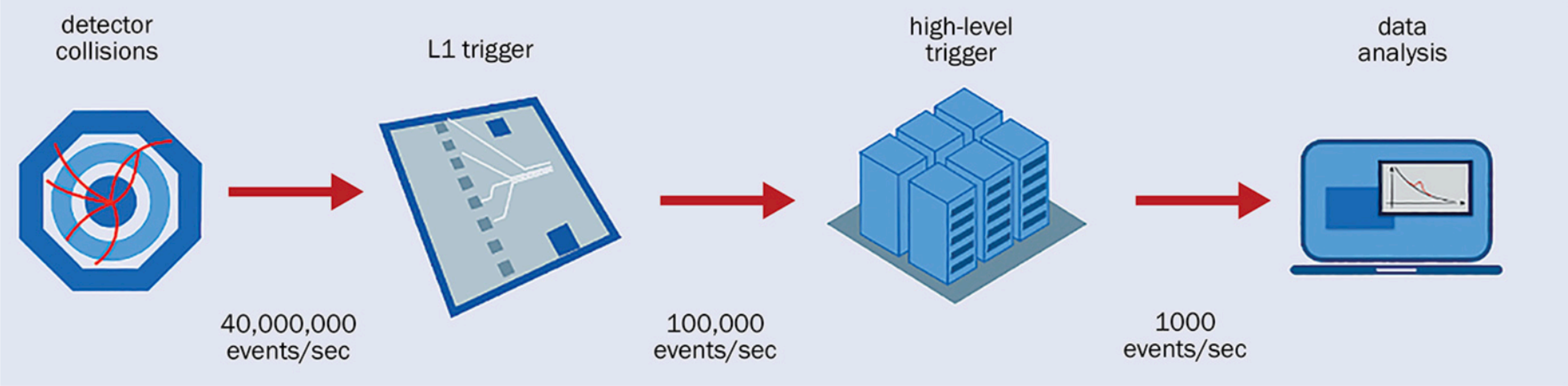
Big Data @LHC

- *The LHC collides protons at unprecedented energy (equivalent to ~13,000 times their mass)*
  - *(nominally) one collision event every 25 ns (= 40 Million collisions/sec)*
  - *Thousands of particles emerging from each time*
  - *1 MB of data recorded at each collision event by big detectors*



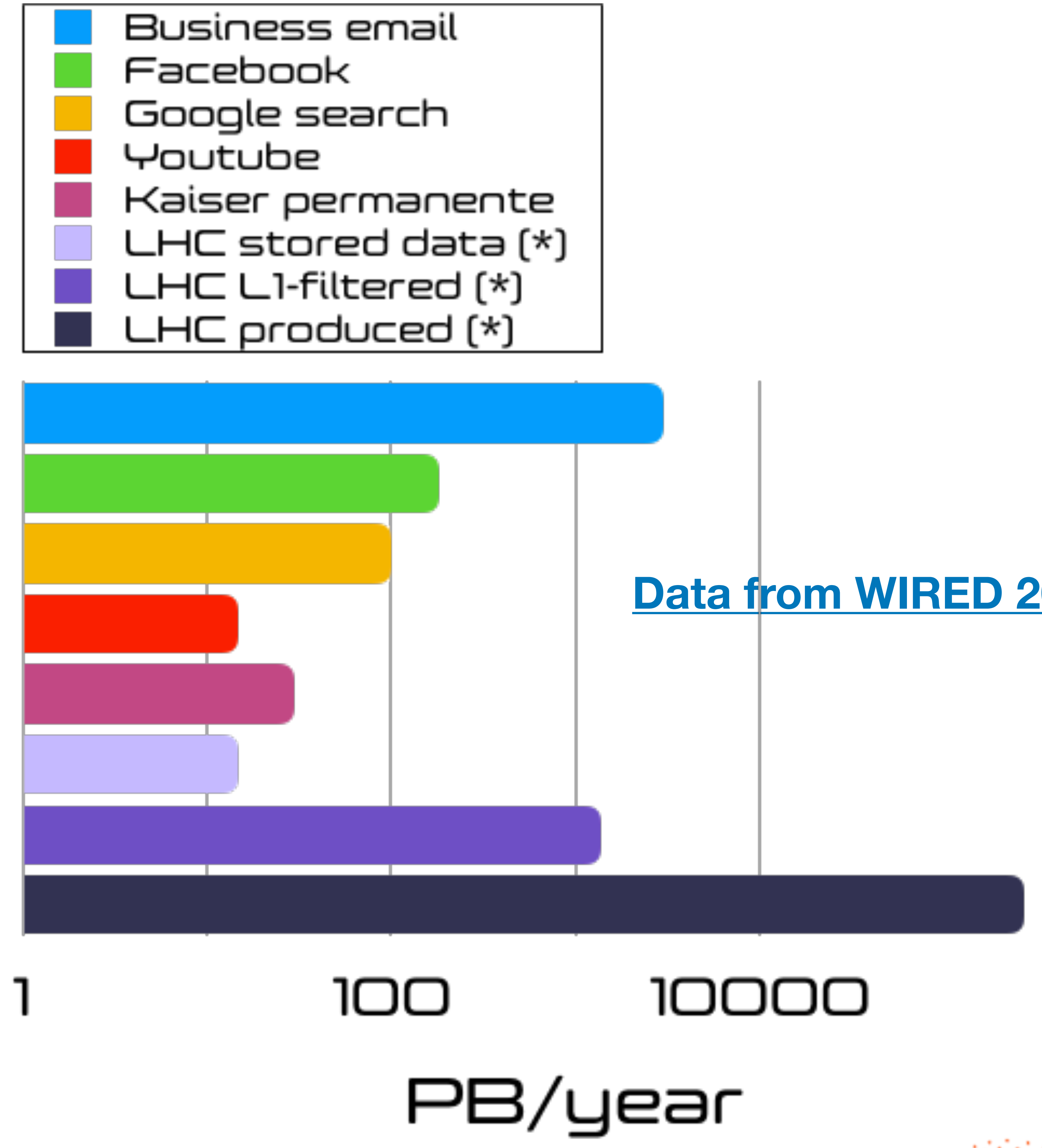
Real-time selection





Big Data @LHC

- The amount of produced data is too much to be stored
  - 1,000 times the data generated by google searches+youtube+facebook back in 2013
  - Reduced to 5x(google searches+youtube+facebook) after first filtering
  - Can only store 5% of those

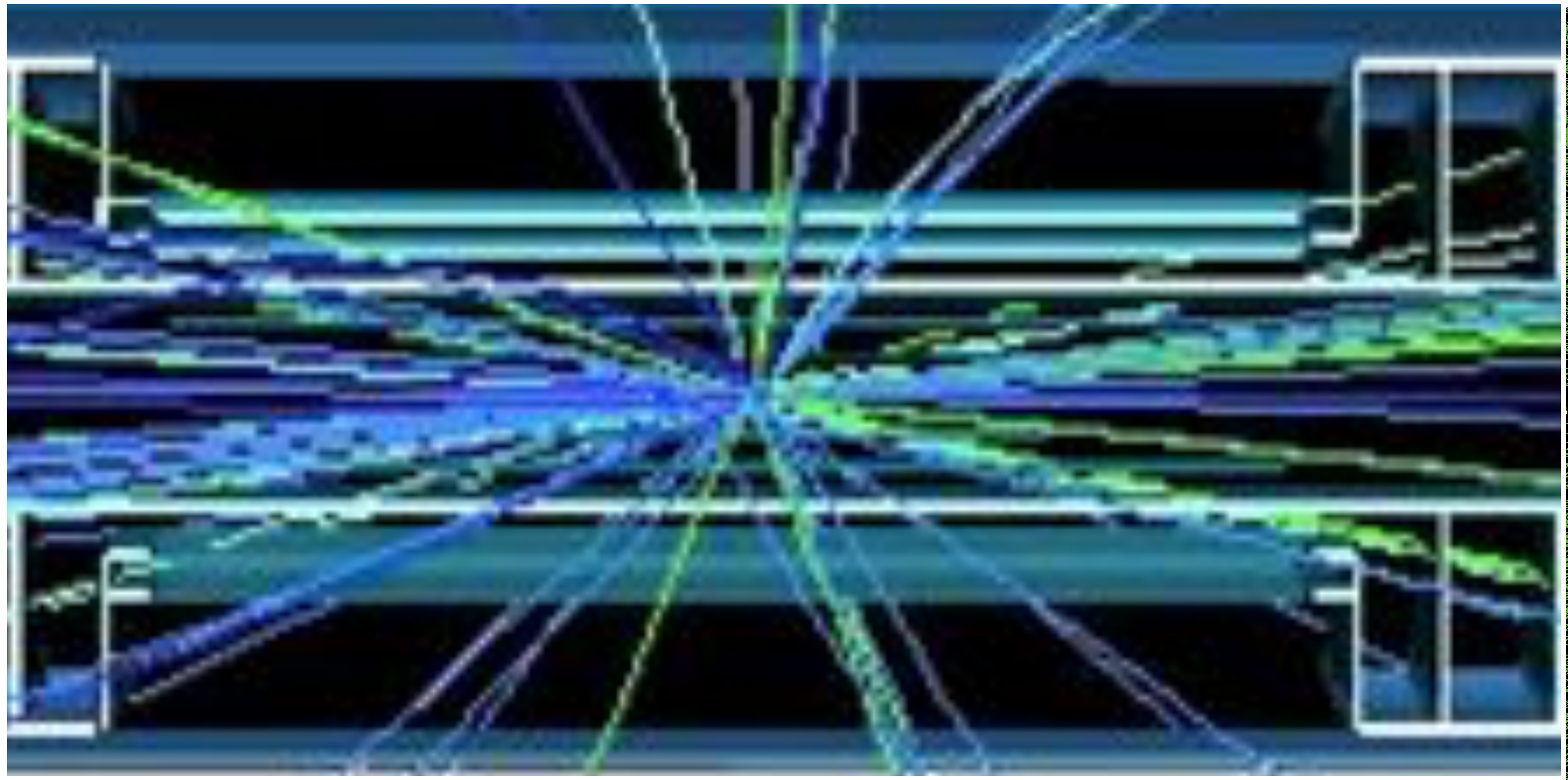


Data from WIRED 2013

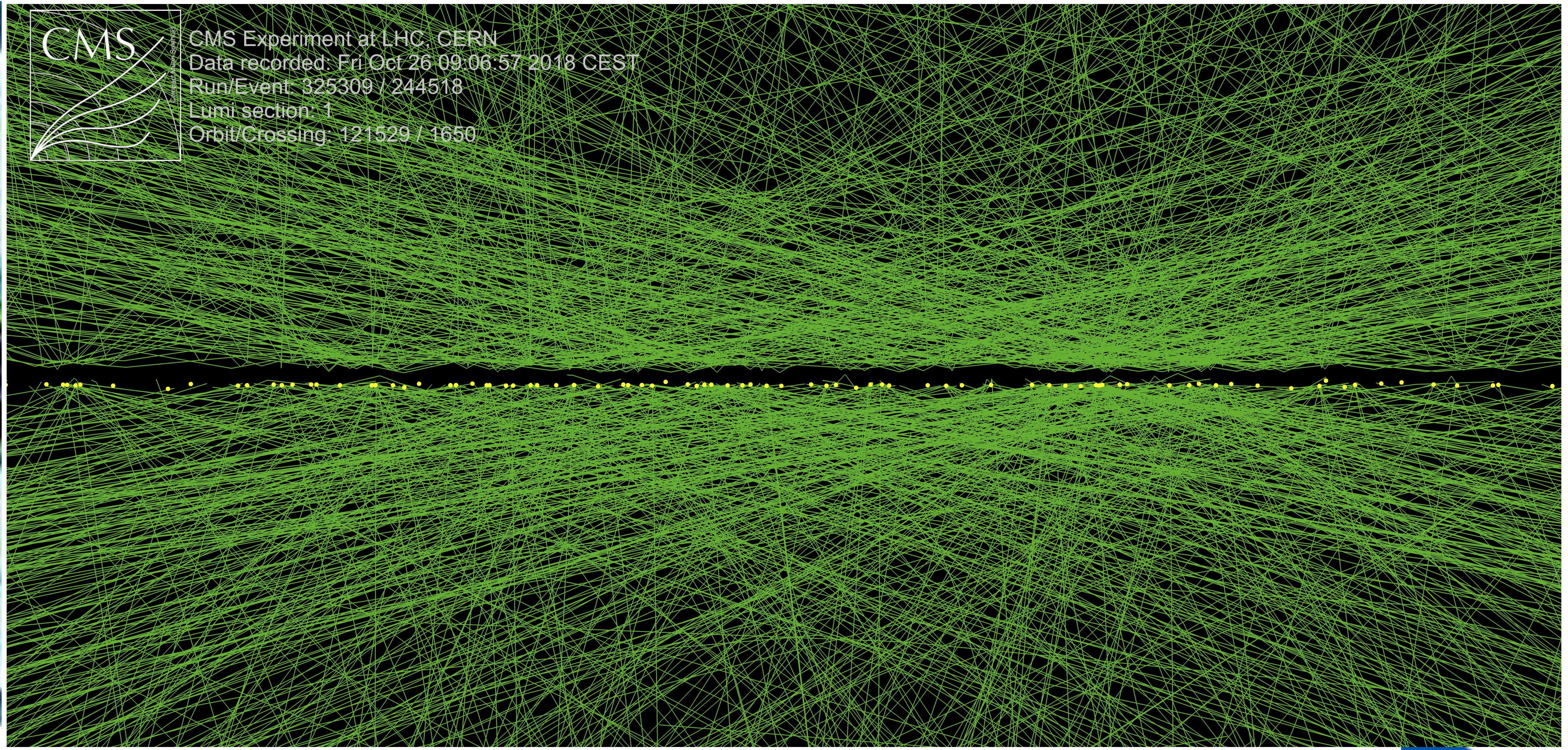
(*) Only two big experiments (ATLAS and CMS), only RAW data

Things will get worse

5 interactions/beam cross



140 interactions/beam cross



- ~40 collisions/event
- ~10 sec/event processing time
- (at best)Same computing resources as today

- ~200 collisions/event
- ~minute/event processing time
- (at best)Same computing resources as today

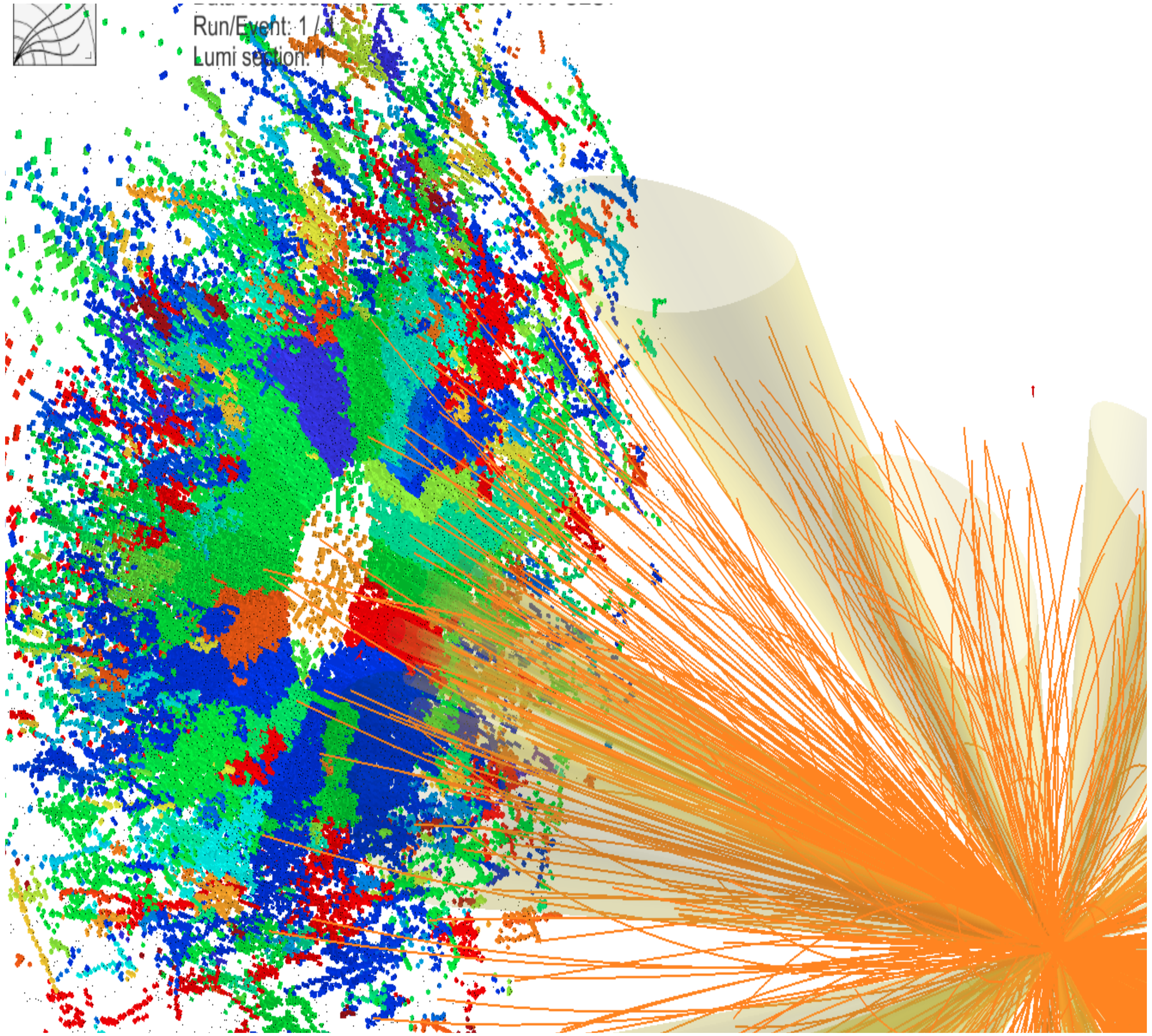
More sensors, more RECO troubles

- *To disentangle 200 collisions happening at once, we will build new detectors with more (smaller) sensors*

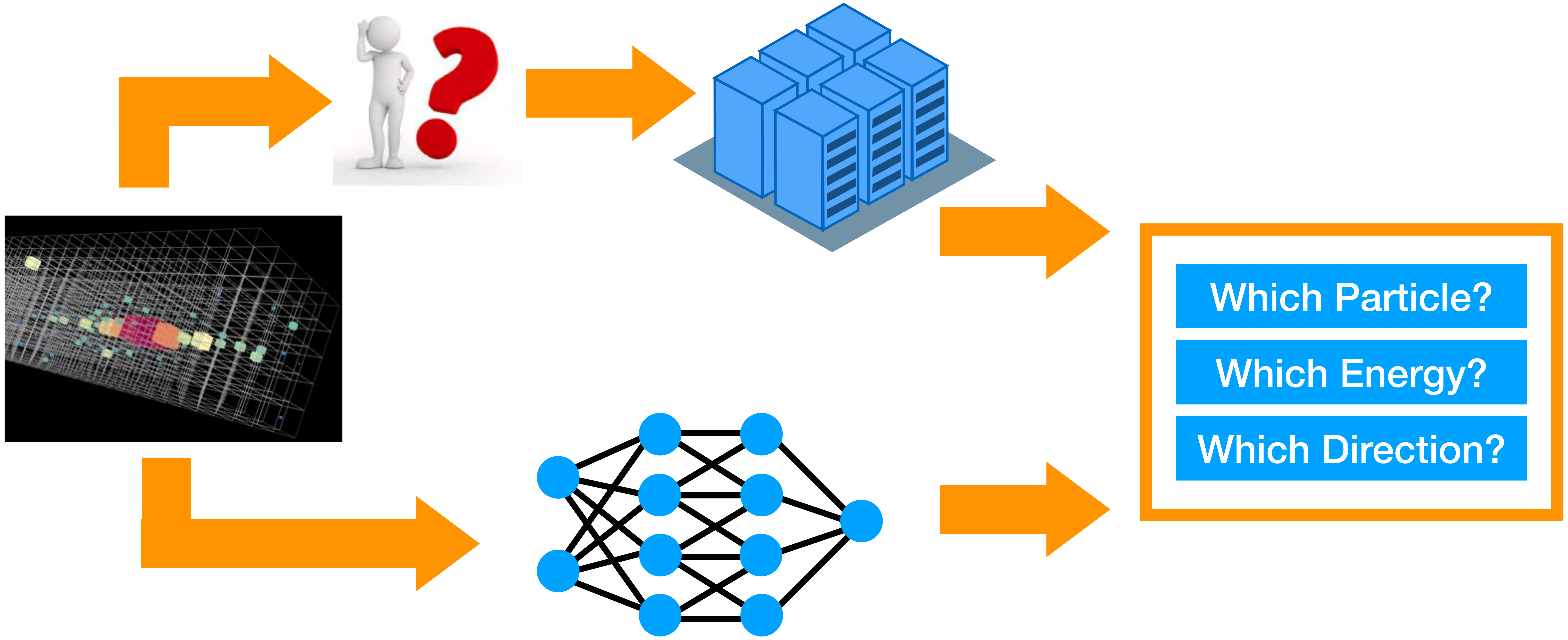
- *Event complexity grows non linearly*

- *To profit of that, computing resources for data processing will have to increase*

- ***We are off by a factor ~10 if we project to 2027***



Deep Learning at Rescue: Reco

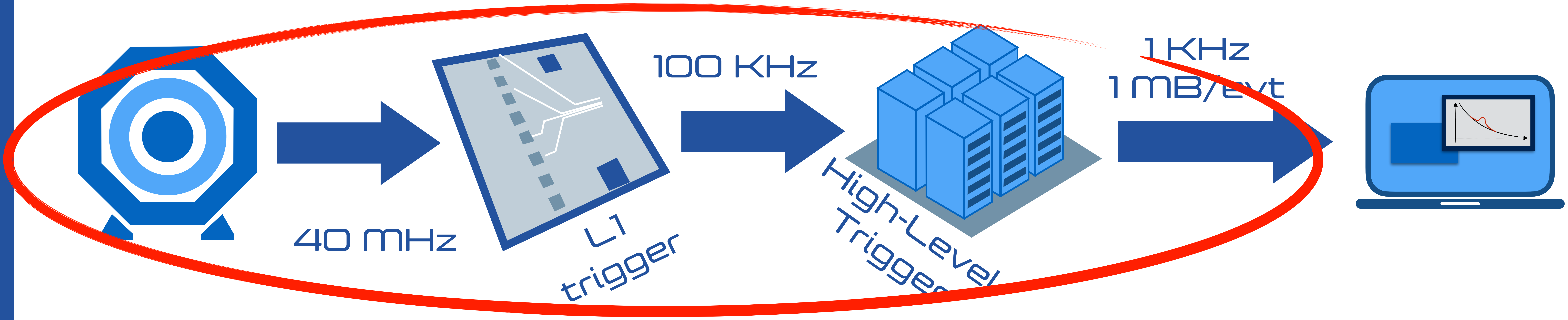


- *We know how to get from the data the answers we want*
  - *physics + intuition + computing*
- *But running these algorithms takes too much time*
  - *We can use DL solutions as a shortcut: we teach neural networks how to get the answer we want directly from the raw data*

Deep Learning and LHC Big Data

- *One BIG challenge: DL deployment needs to happen in between collisions and data analysis (trigger, reconstruction, …), where freeing resources will make a difference*

- *Other issue: our data are not mainstream Deep Learning data (images, sequences, etc.). Lot of work going into designing custom solutions with physics knowledge injected*

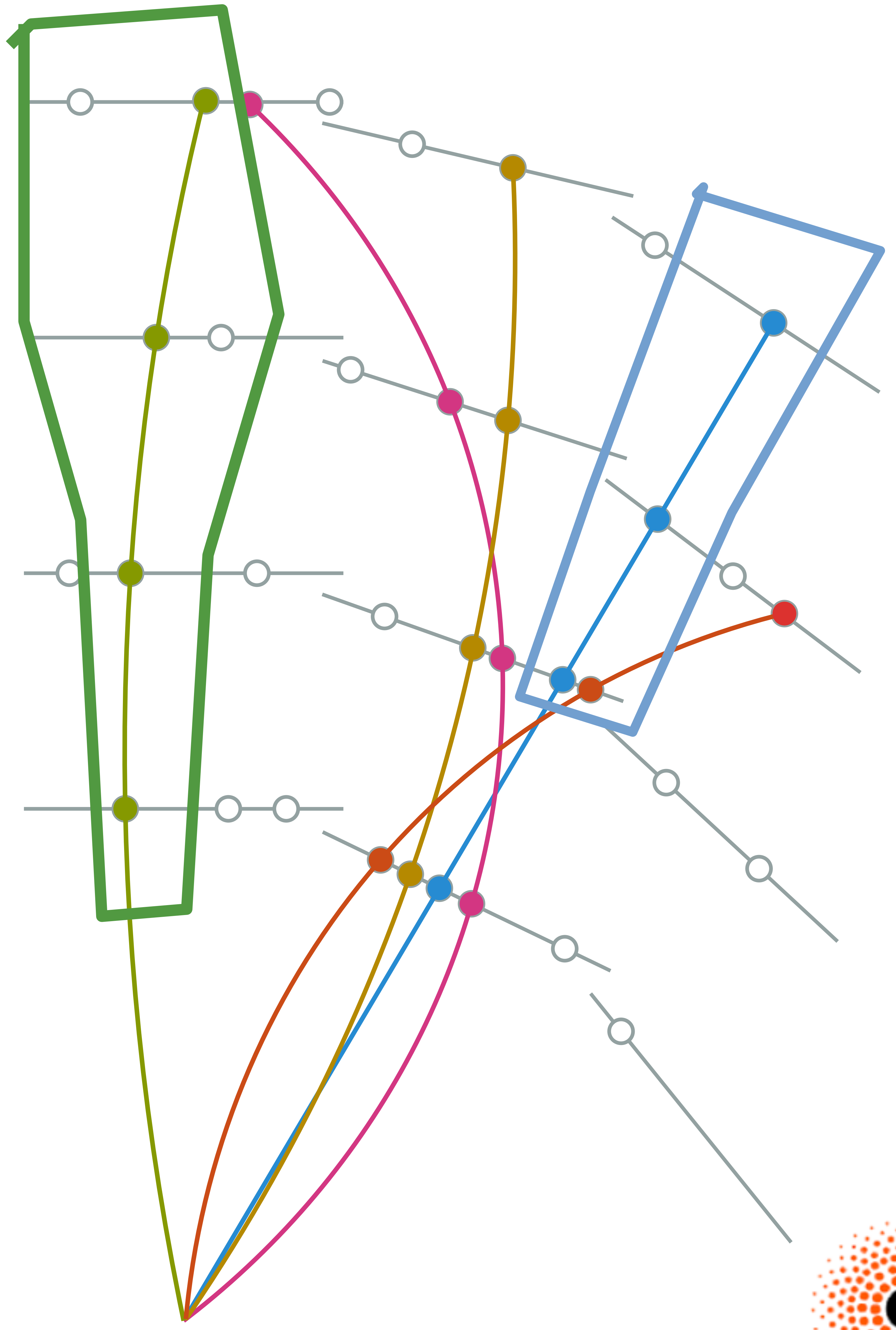


Dealing with HEP data

- ***Sparse data:*** *HEP data are sets (point clouds) of* ***detector hits***

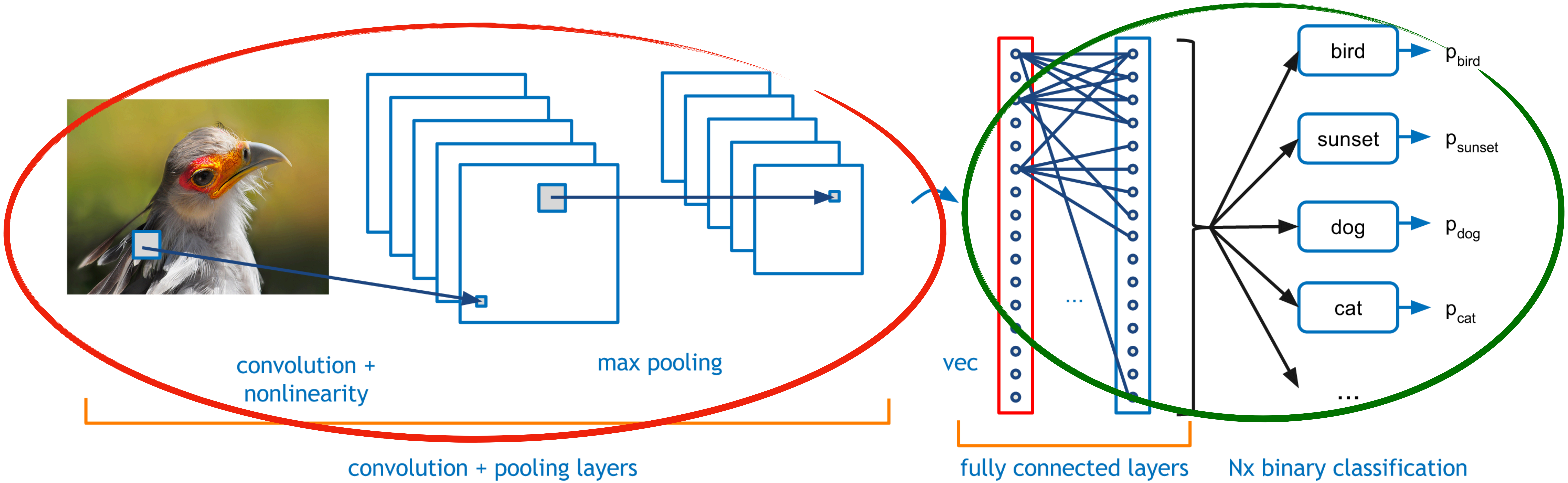
- ***Custom edge computing:*** *algorithms have to run on special resources: custom custom electronic chips, dedicated computer centres, the worldwide GRID*

- ***Real-time:*** *execution has to happen within short time (as fast as ~100 nsec)*



Accessing Raw Data

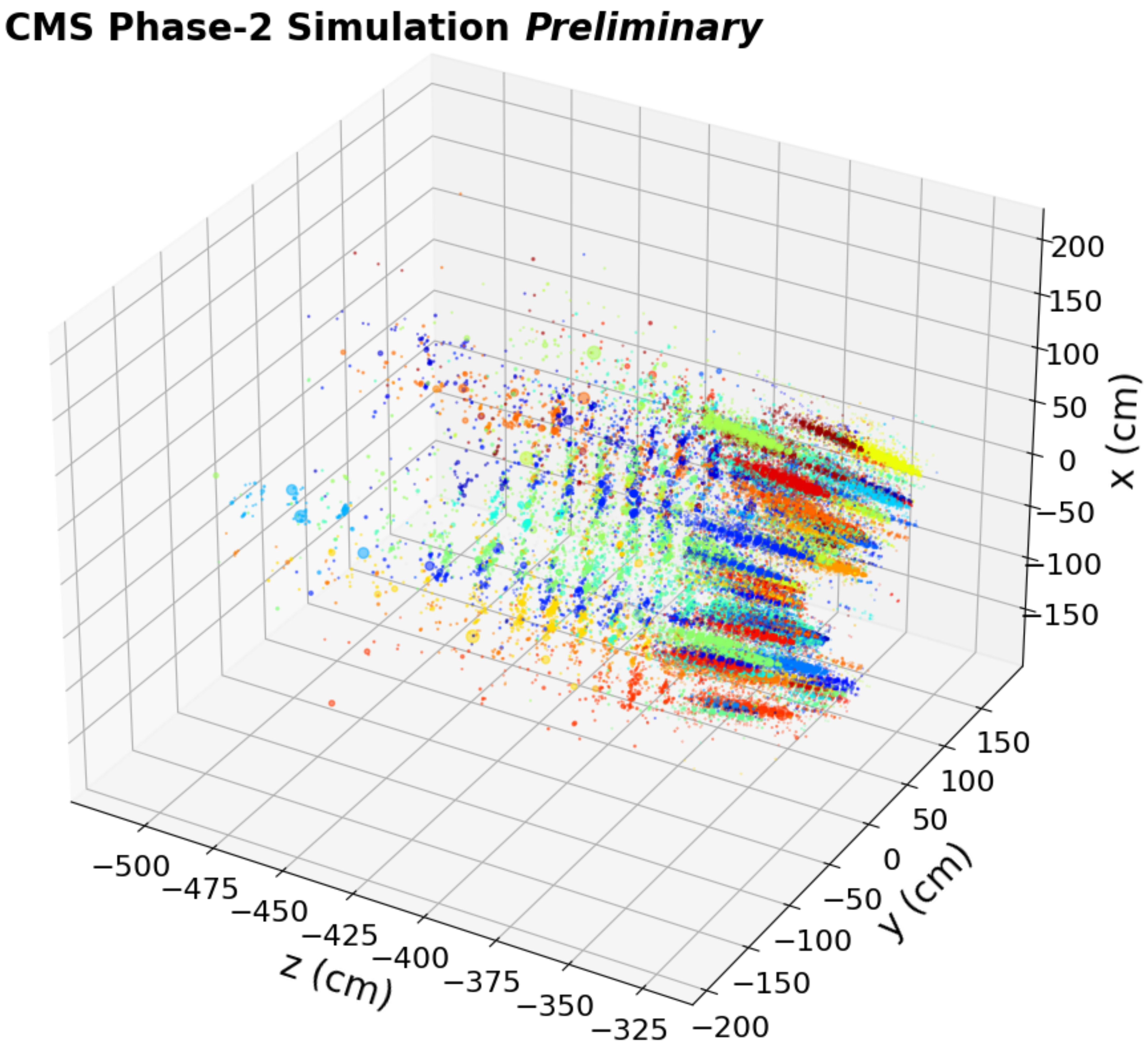
- *DNNs typically rely on two phases:*
  - ***Feature engineering*** *from Raw Data. This is where new & exotic architectures (depending on data type) take the best out of your data*



  - ***Task solving****: start from engineered features and solve the task (classification, regression, etc.)*

New Opportunities

- *Because we can process raw data directly, we can go beyond high-level classification and regression*
  - *We can do classification/regression directly on raw detector hits*
  - *We can generate detector hits (generative models)*
  - *We can look for strange/new kinds of patterns in data (anomaly detection)*
- *To do so, different architectures are used*
  - *Autoencoders*
  - *Generative models*
  - …



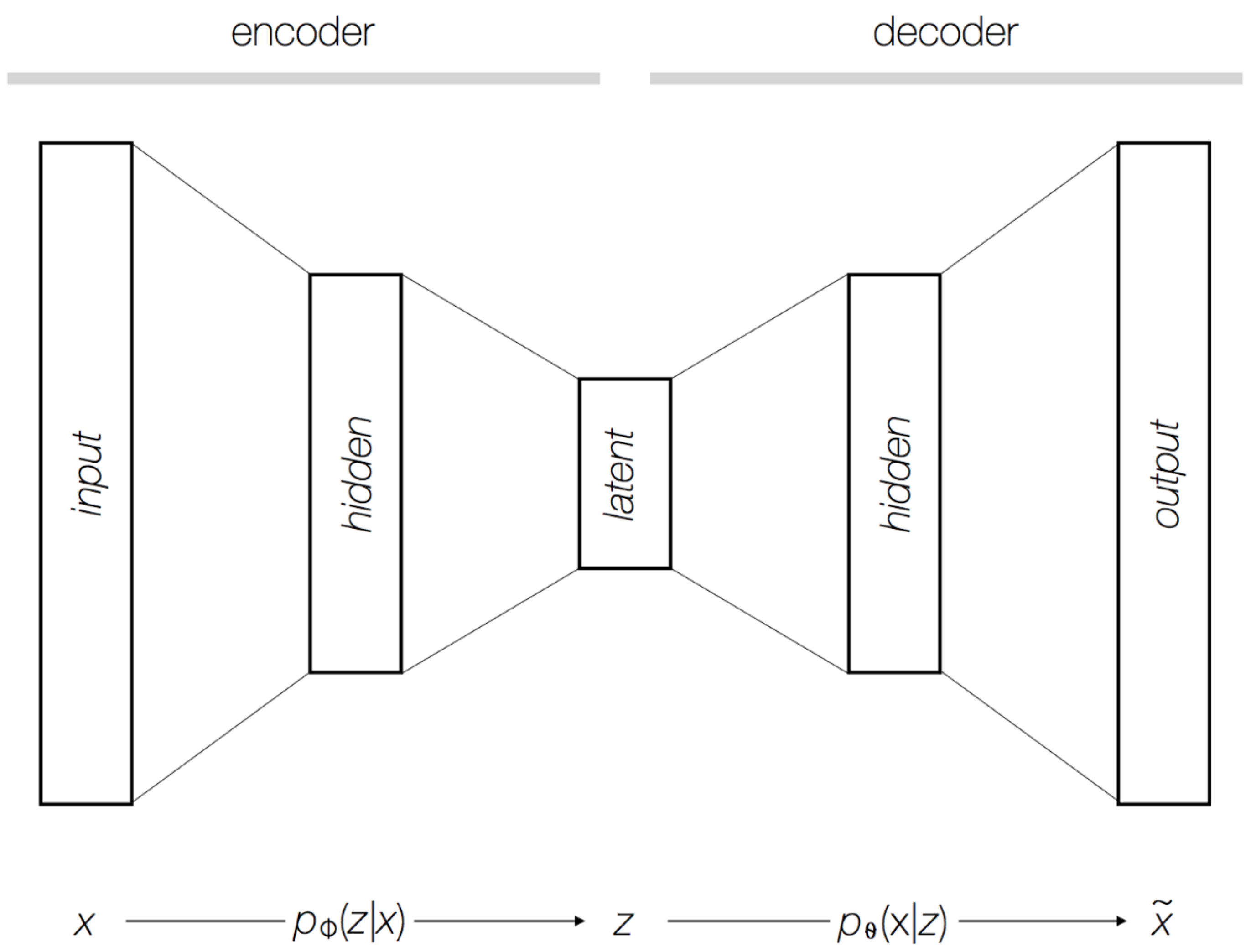
Autoencoders in a nutshell

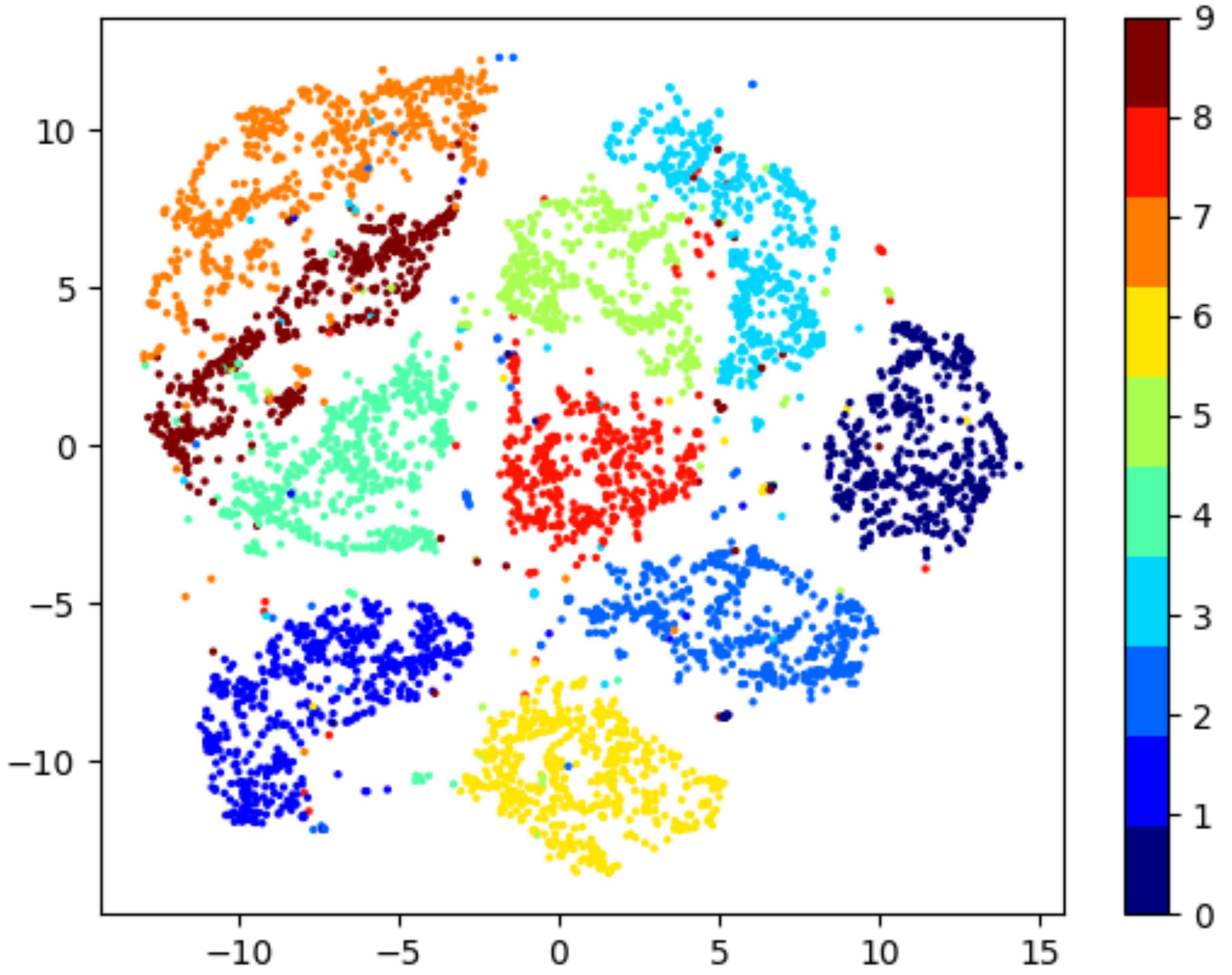
- *Autoencoders are compression-decompression algorithms that learn to describe a given dataset in terms of points in a lower-dimension latent space*

- *UNSUPERVISED algorithm, used for data compression, generation, clustering (replacing PCA), etc.*

- *Used in particular for anomaly detection: when applied on events of different kind, compression-decompression tuned on refer sample might fail*

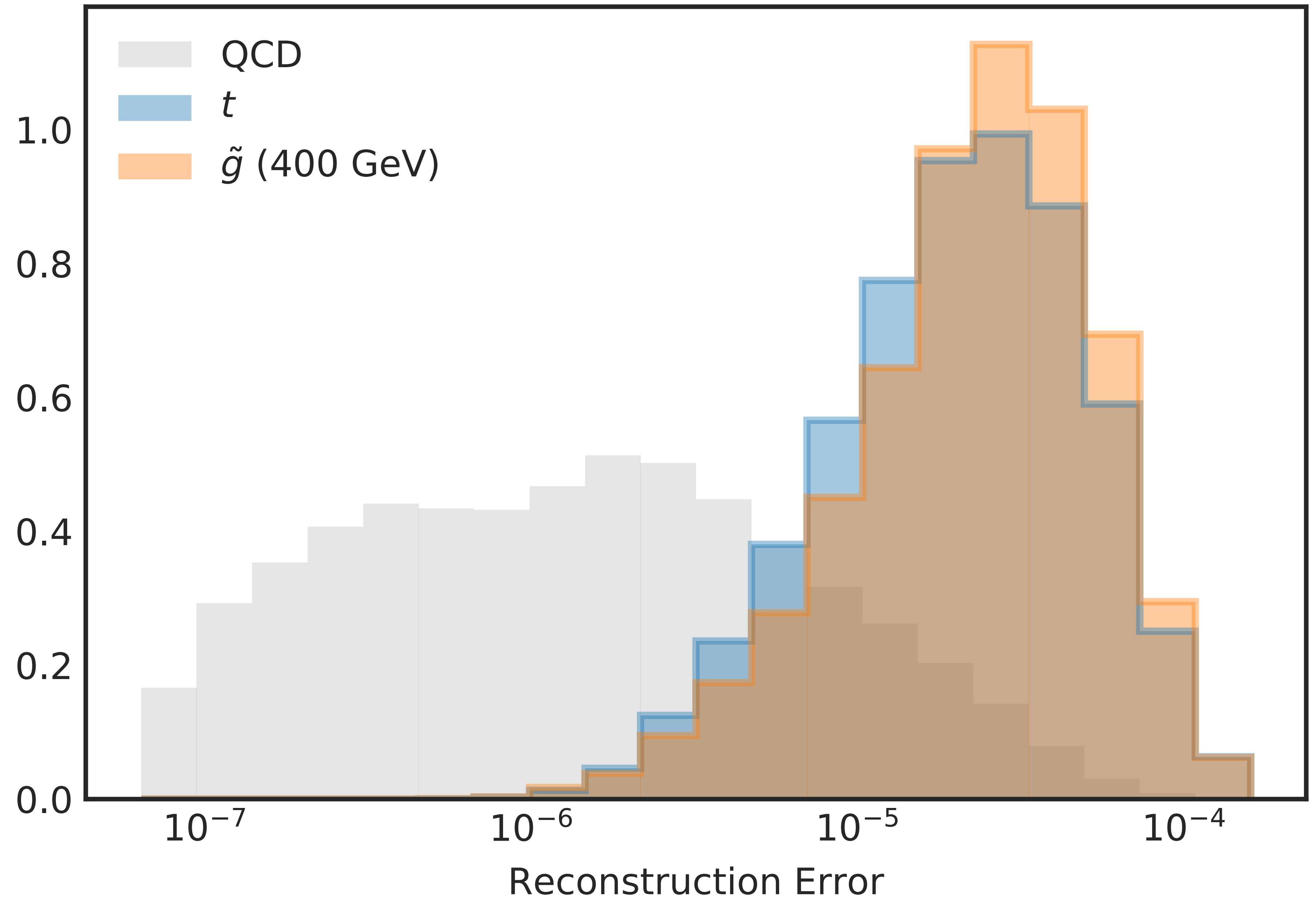
- *One can define anomalous any event whose decompressed output is "far" from the input, in some metric (e.g., the metric of the auto-encoder loss)*

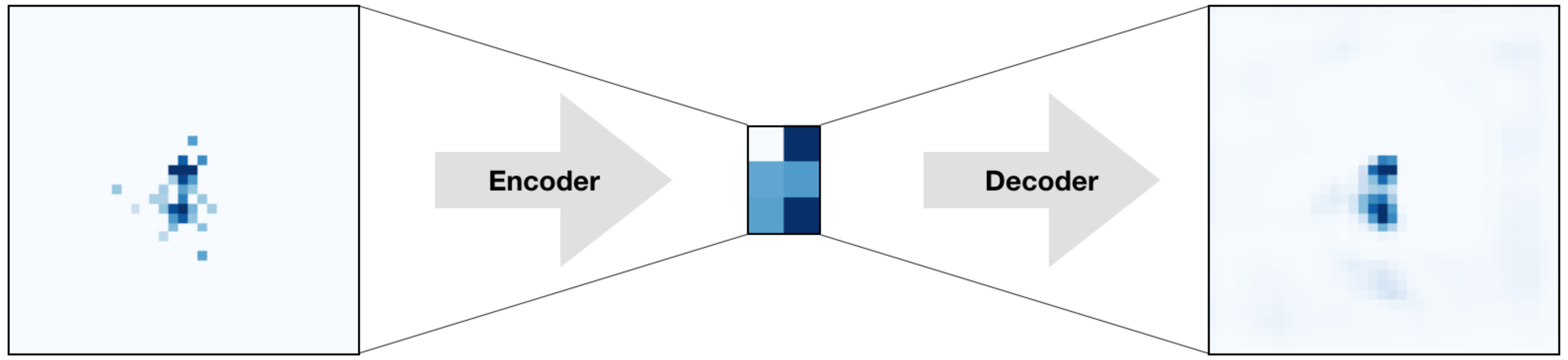


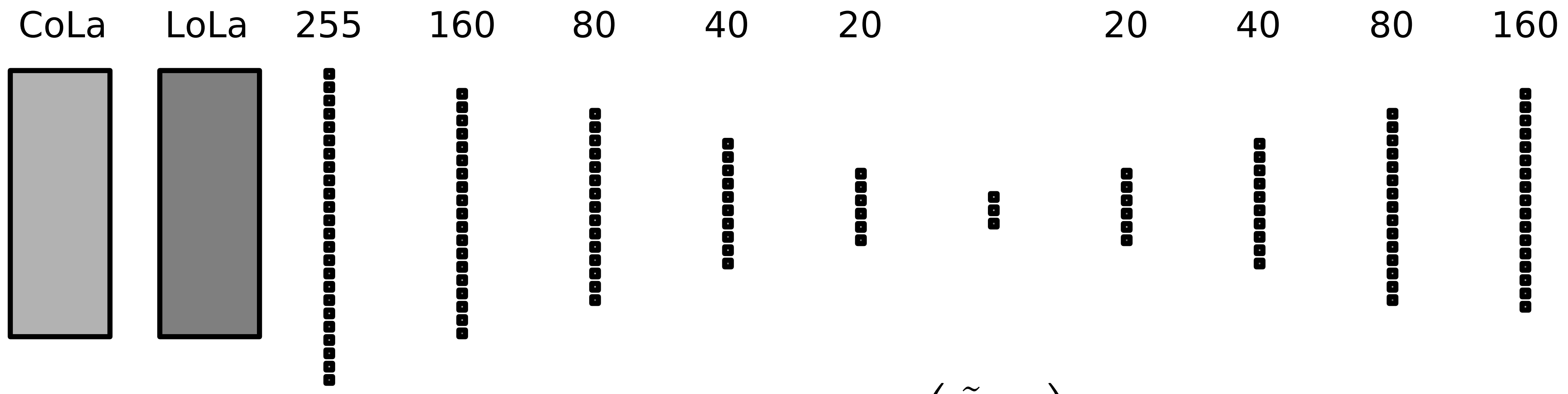


Example: Jet autoencoders

- *Idea applied to tagging jets, in order to define a QCD-jet veto*
- *Applied in a BSM search (e.g., dijet resonance) could highlight new physics signal*
- *Based on image and physics-inspired representations of jets*







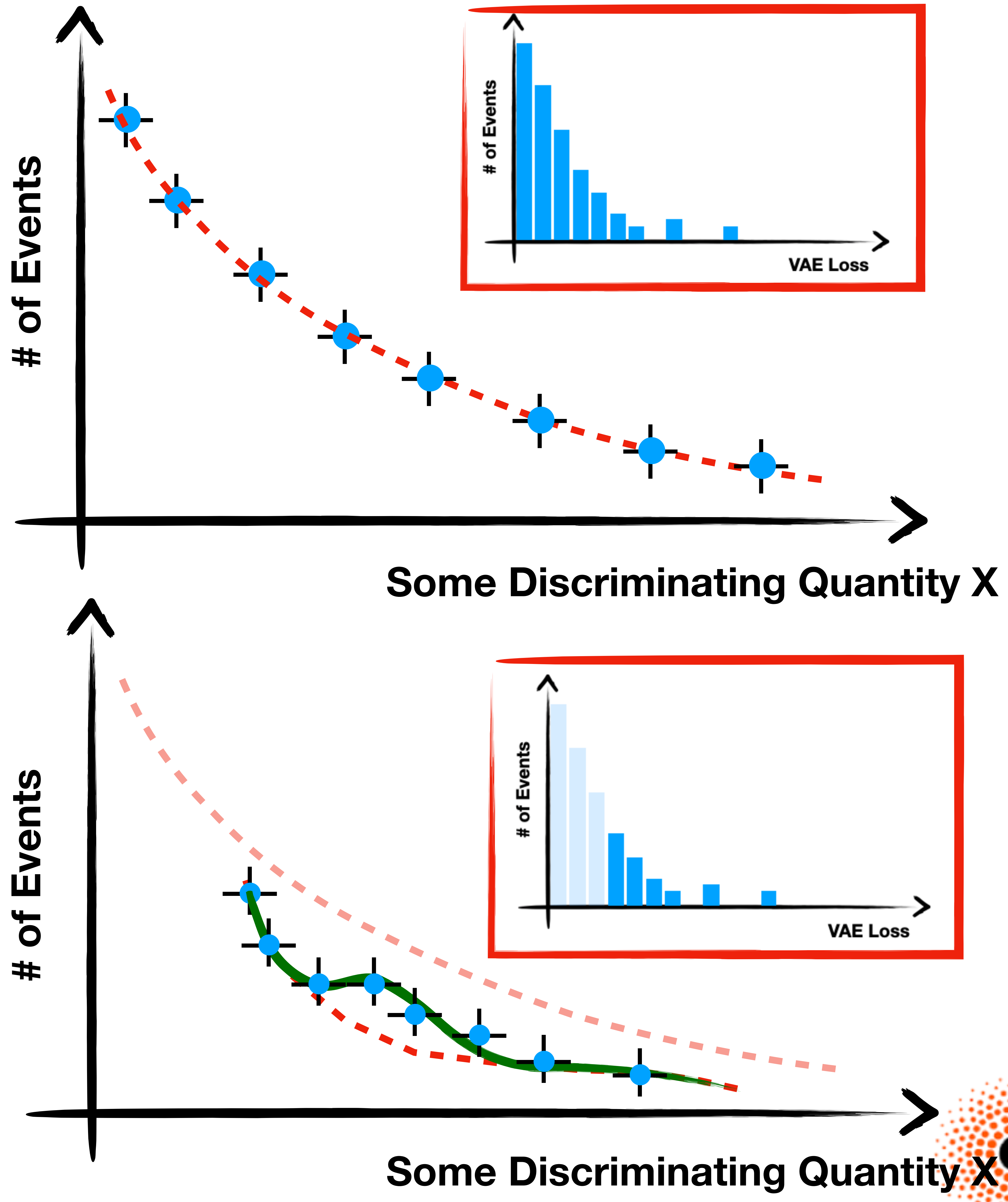
$$\tilde{k}_j = \begin{pmatrix}\tilde{k}_{0,j}\\ \tilde{k}_{1,j}\\ \tilde{k}_{2,j}\\ \tilde{k}_{3,j}\end{pmatrix} \xrightarrow{\text{LoLa}} \begin{pmatrix}\tilde{k}_{0,j}\\ \tilde{k}_{1,j}\\ \tilde{k}_{2,j}\\ \tilde{k}_{3,j}\\ \sqrt{\tilde{k}_j^2}\end{pmatrix}.$$

[Farina et al., arXiv:1808.08992]

[Heimel et al., arXiv:1808.08979]

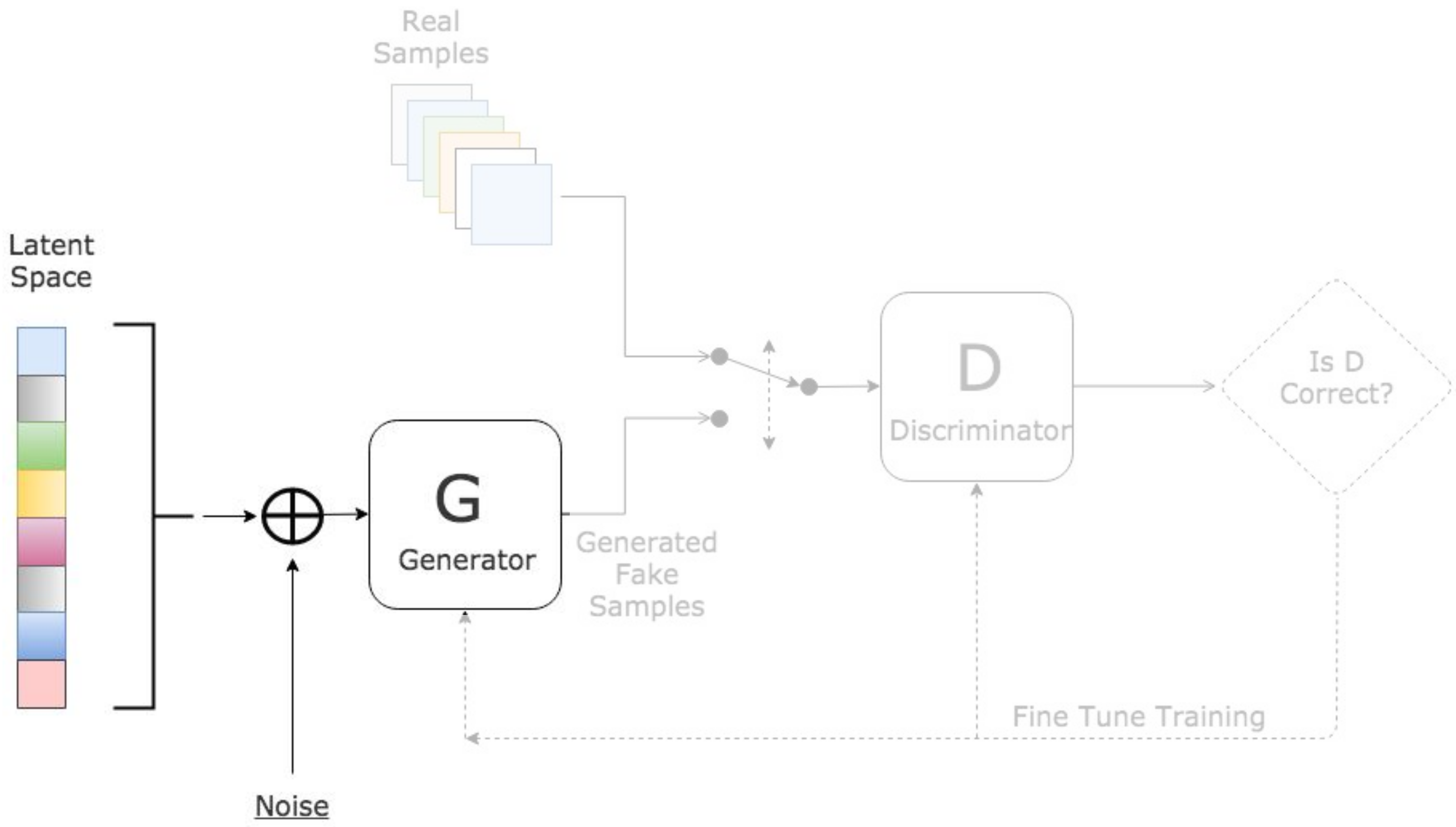
How does one use this in analysis?

- Anomaly defined as a p-value threshold on a given test statistics
  - Loss function an obvious choice
- Doing so, one wants to avoid deformations in the background distribution that could fake a signal



Generative Adversarial Training

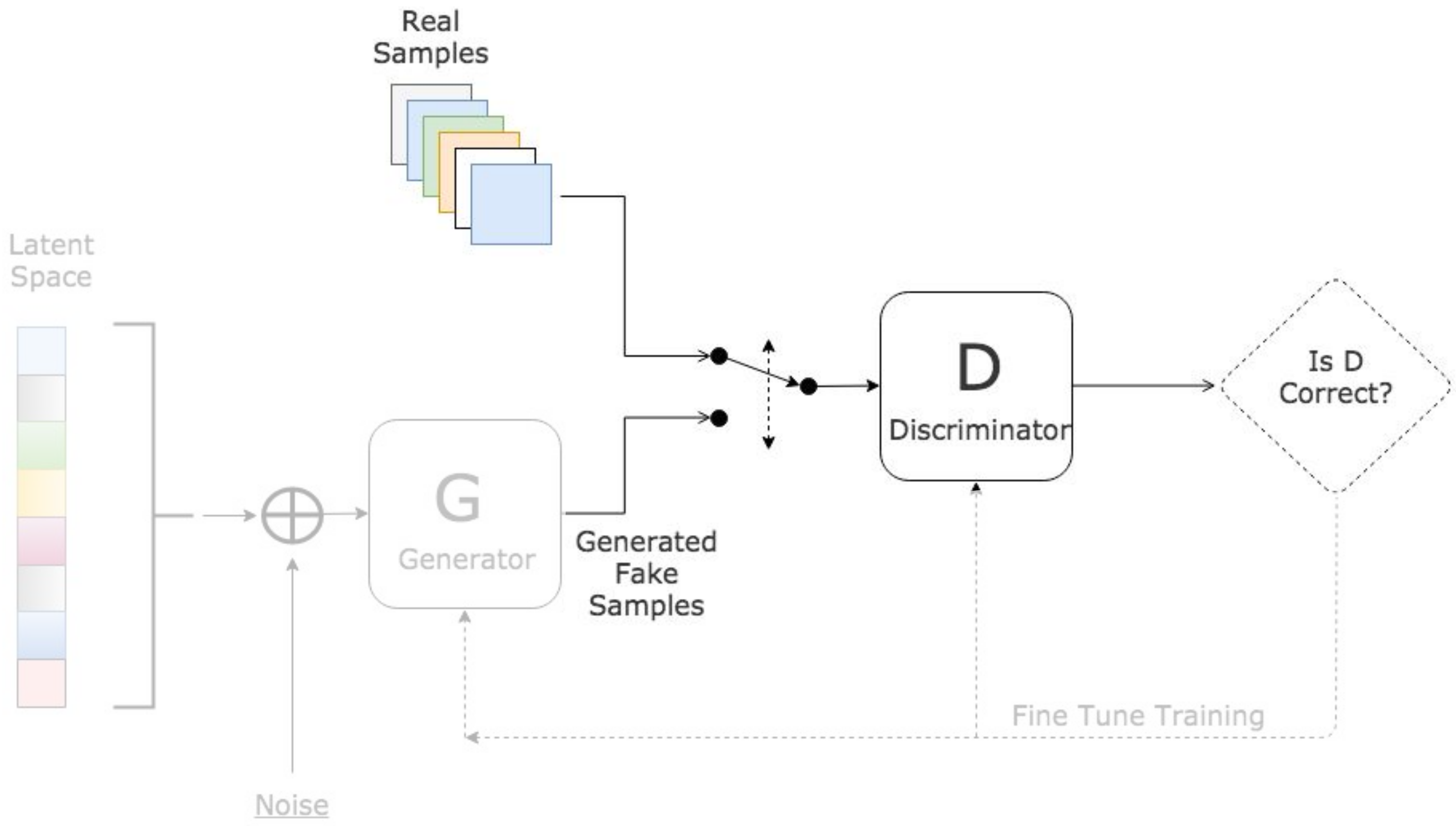
- Two networks trained against each other
  - **Generator: create images (from noise, other images, etc)**
  - Discriminator: tries to spot which image comes from the generator and which is genuine



- Loss function to minimise: Loss(Gen)-Loss(Disc)
  - Better discriminator -> bigger loss
  - Better generator -> smaller loss
- Trying to full the discriminatore, generatore learns how to create more realistic images

Generative Adversarial Training

- *Two networks trained against each other*
  - *Generator: create images (from noise, other images, etc)*
  - ***Discriminator: tries to spot which image comes from the generator and which is genuine***

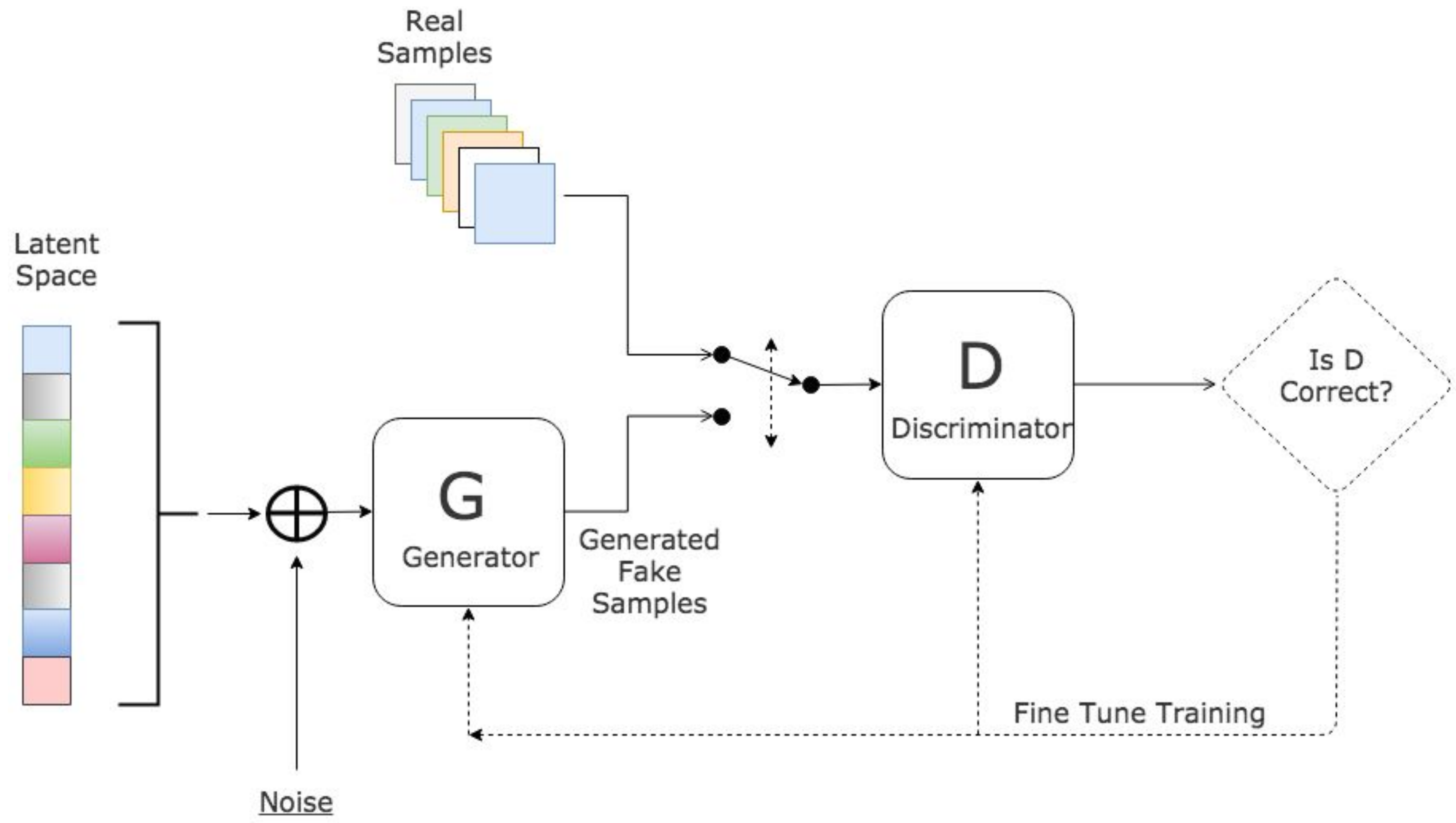


- *Loss function to minimise: Loss(Gen)-Loss(Disc)*
  - *Better discriminator -> bigger loss*
  - *Better generator -> smaller loss*
- *Trying to full the discriminatore, generatore learns how to create more realistic images*

European Research Council

Generative Adversarial Training

- *Two networks trained against each other*
  - *Generator: create images (from noise, other images, etc)*
  - *Discriminator: tries to spot which image comes from the generator and which is genuine*

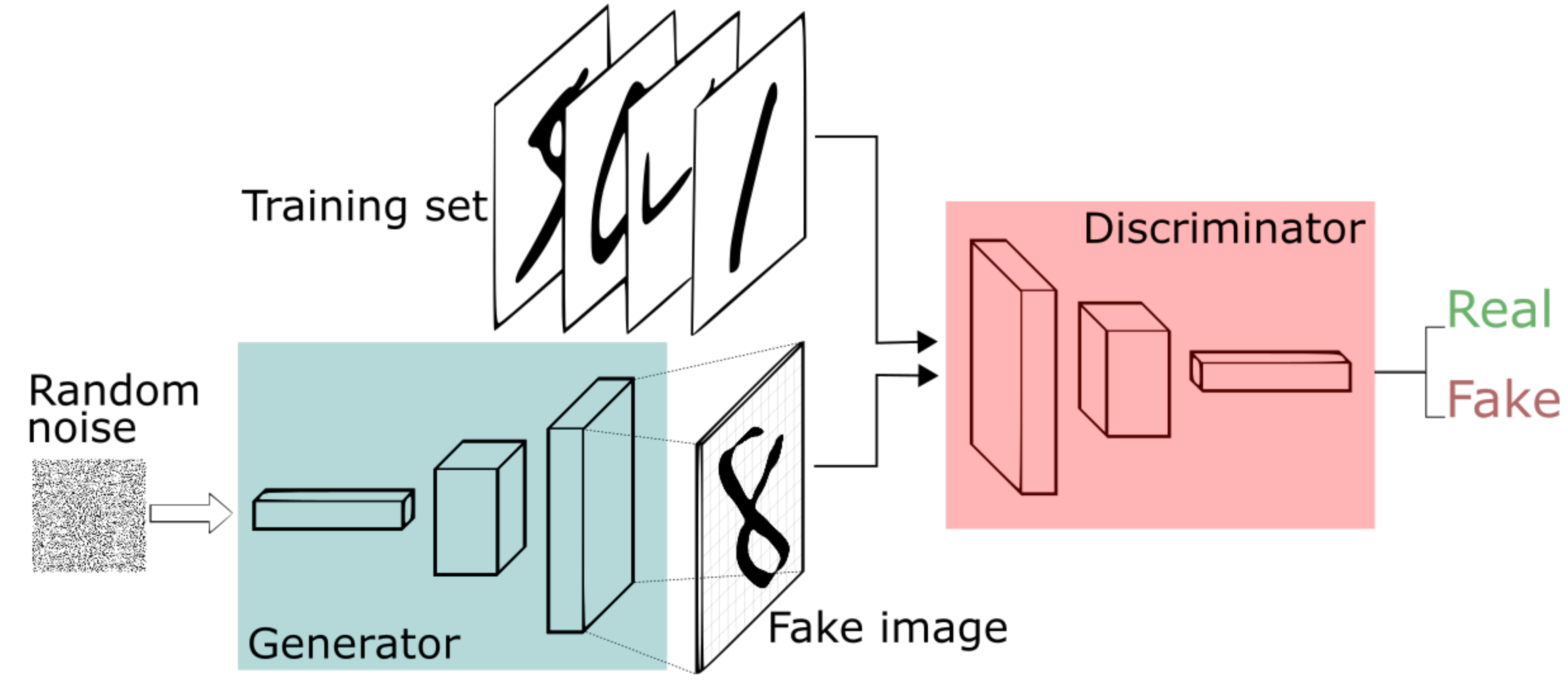


- ***Loss function to minimise: Loss(Gen)-Loss(Disc)***
  - ***Better discriminator -> bigger loss***
  - ***Better generator -> smaller loss***
  - ***Trying to full the discriminatore, generatore learns how to create more realistic images***

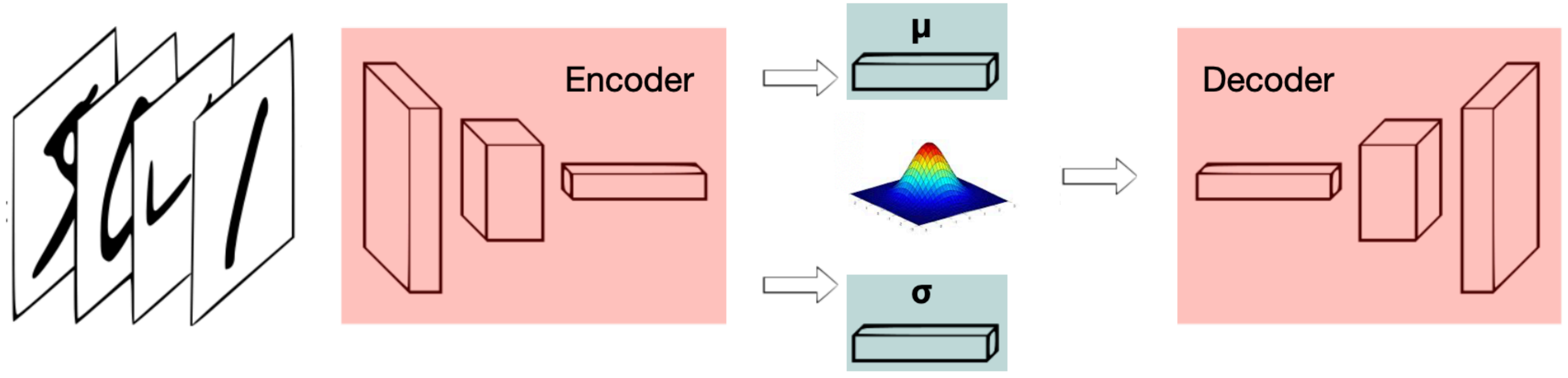
Generative Adversarial Training

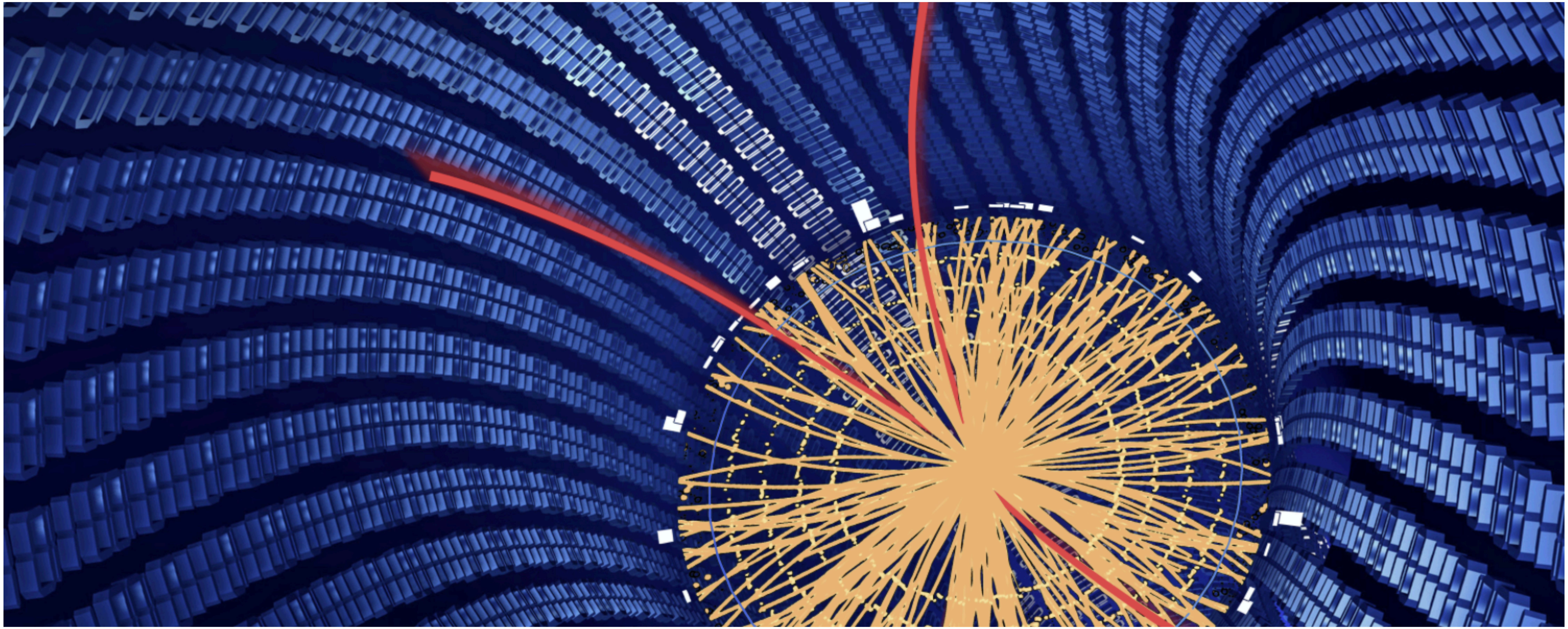
Generative Models

- *GAN:*
  - *create fake data from random noise with Generator*
  - *train it against a Discriminator which tried to identify the fakes*
  - *until the Generator confuses the discriminator*



- *VAE:*
  - *compress the input to a (Gaussian) pdf in some latent space*
  - *sample from the Gaussian*
  - *decompress back to the input space*
  - *use the last two steps above as a generator*

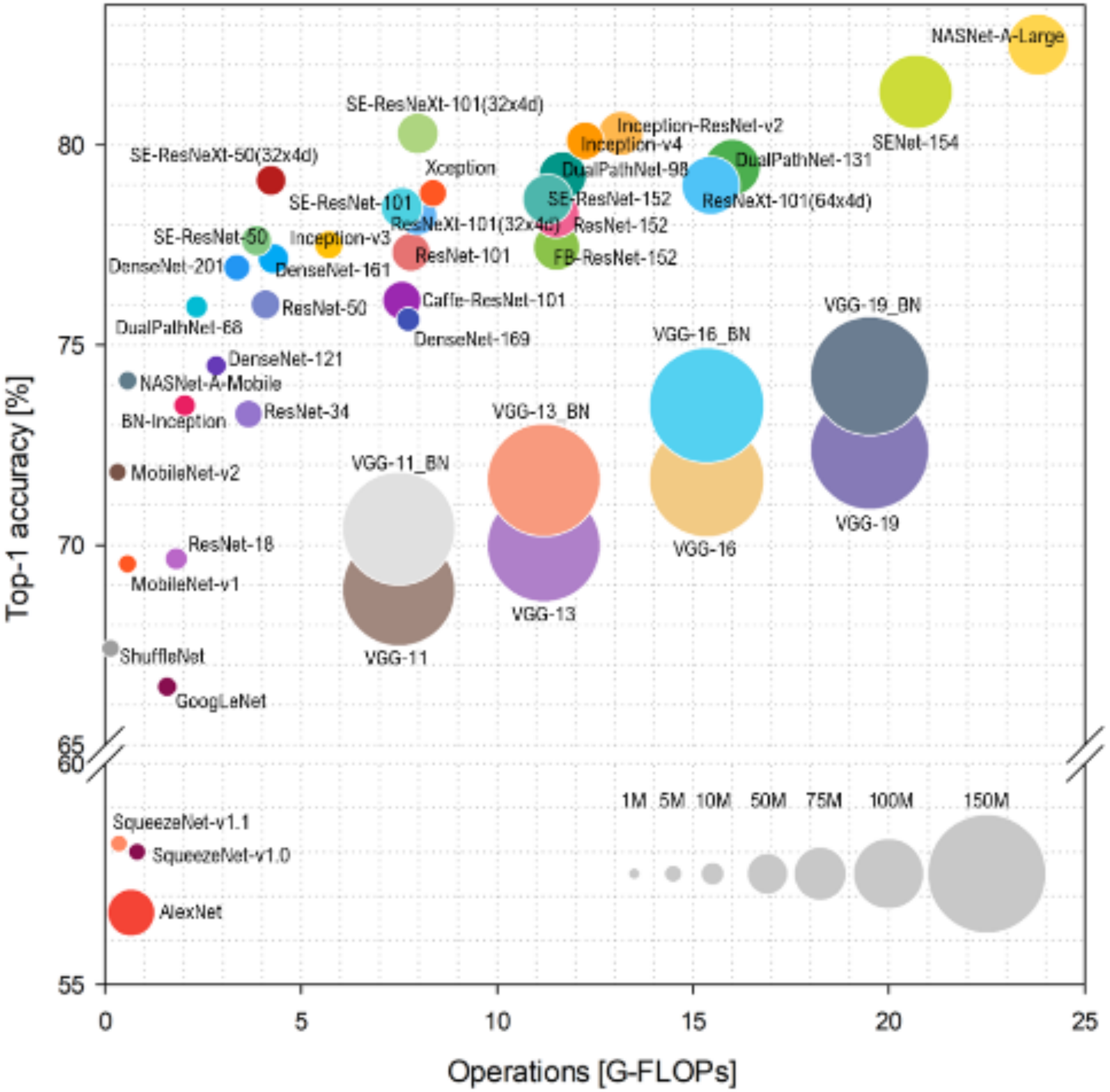




Data Representation

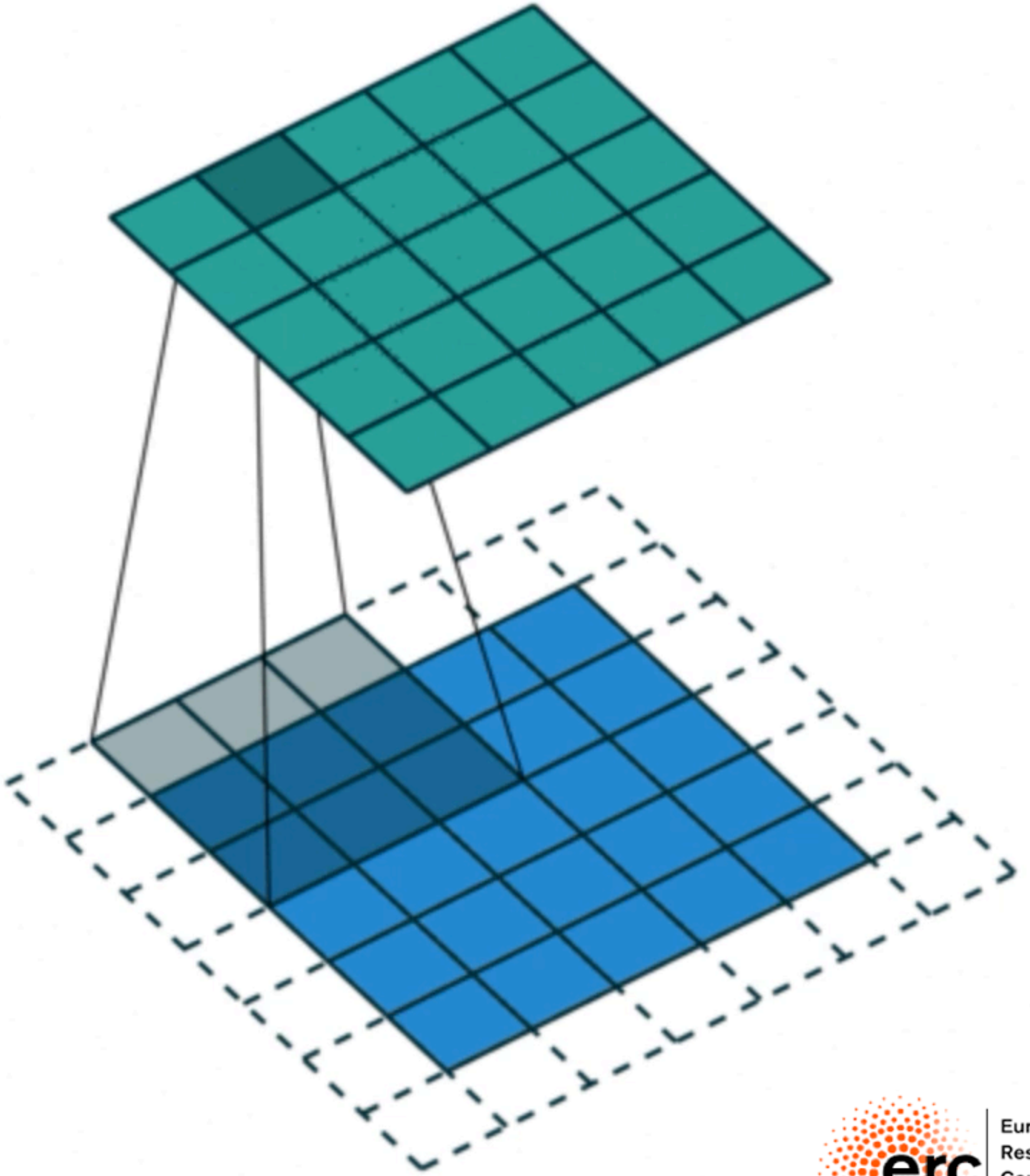
Deep Learning & Computing Vision

- The most evident success of Deep Learning is computing vision with Convolutional NNs
  - A kernel scans an array of pixels
  - The network is translation invariant
  - The network knows which pixels are near each other and learns from there



Deep Learning & Computing Vision

- *The most evident success of Deep Learning is computing vision with Convolutional NNs*
  - *A kernel scans an array of pixels*
  - *The network is translation invariant*
  - *The network knows which pixels are near each other and learns from there*



CNN in Science

- Paradigm applied successfully to many scientific problems
  - Exoplanet detection
  - Frequency-domain analysis of Gravitational Interferometer data
  - Neutrino detection
  - etc…

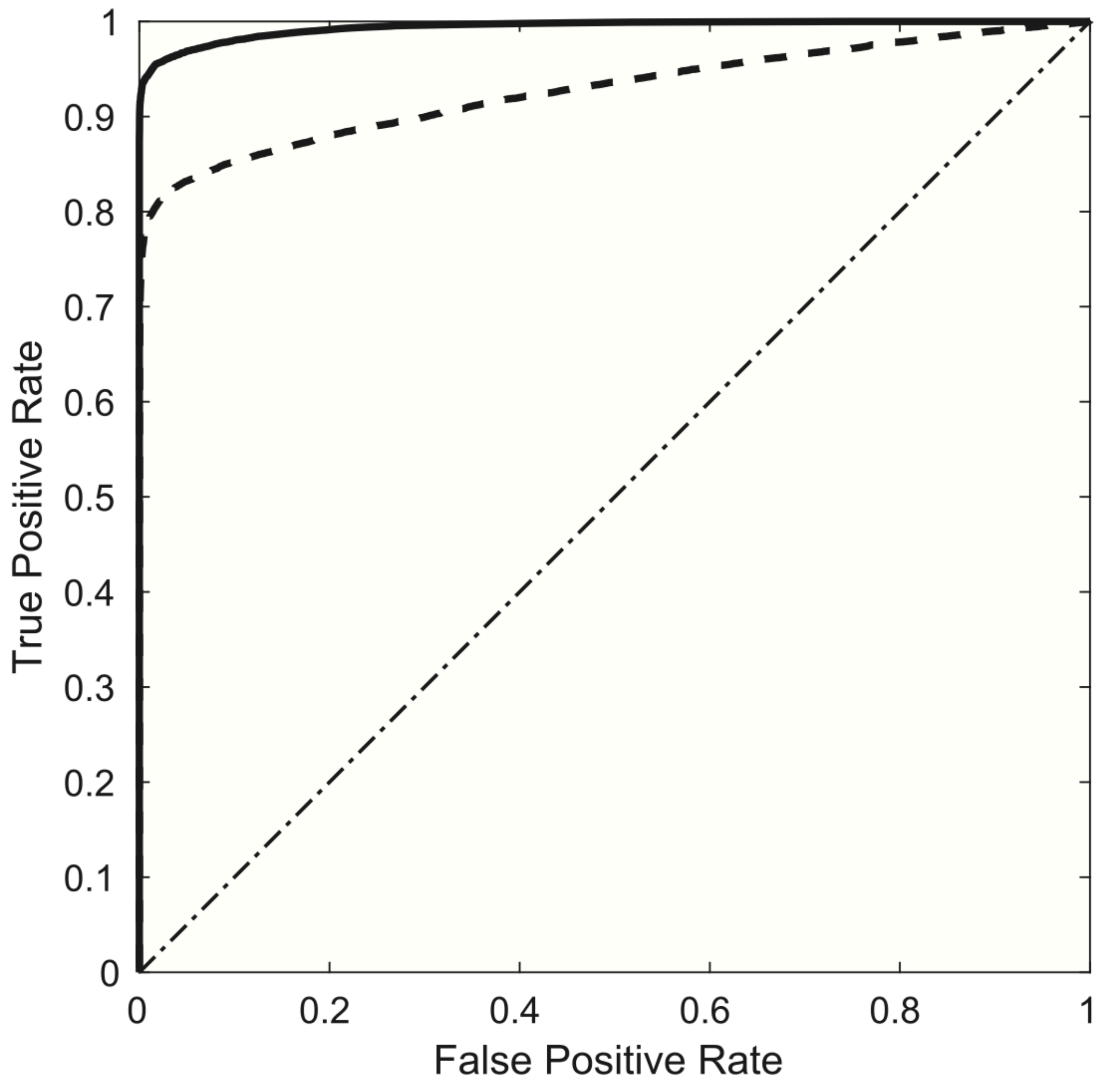
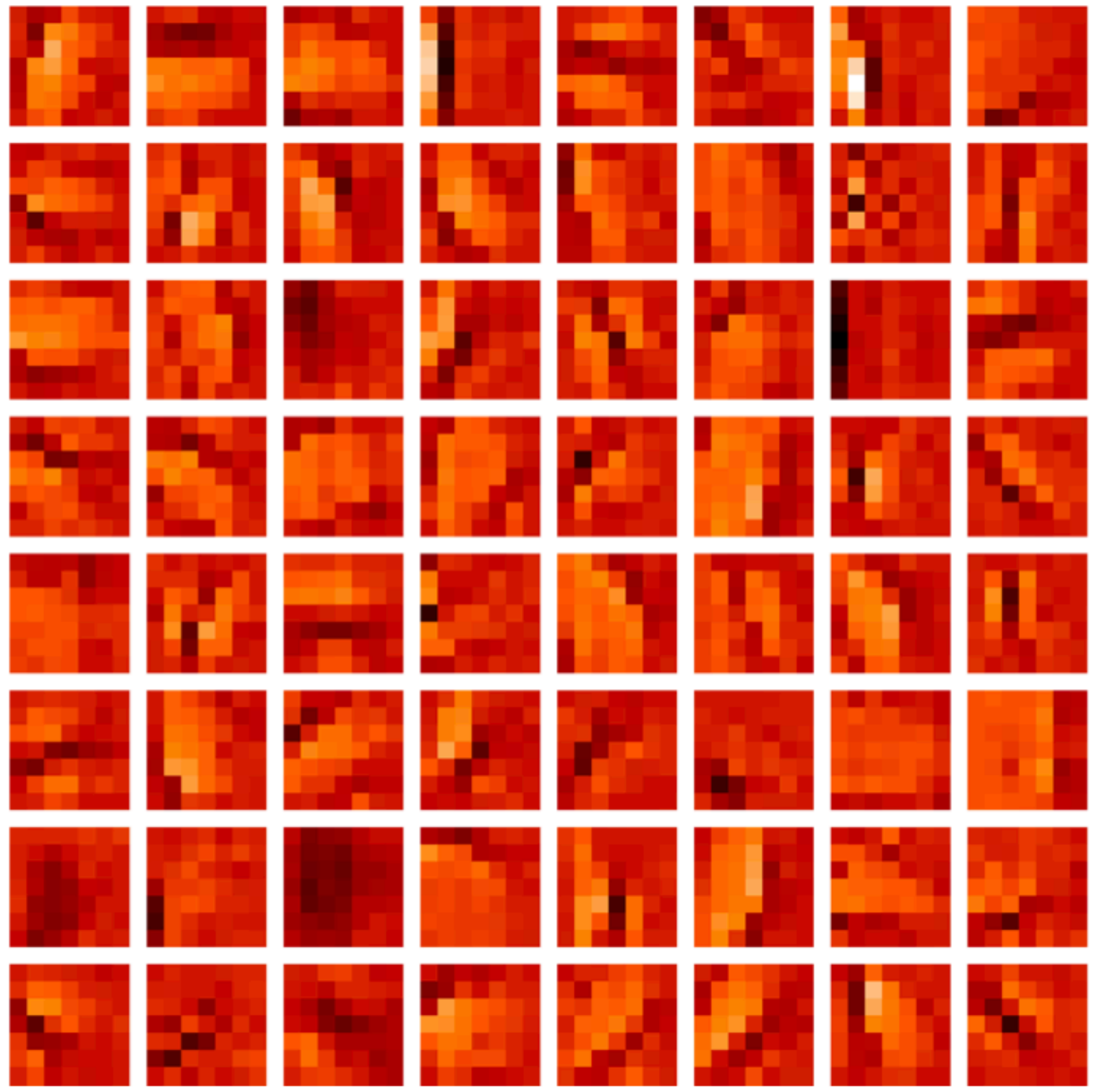
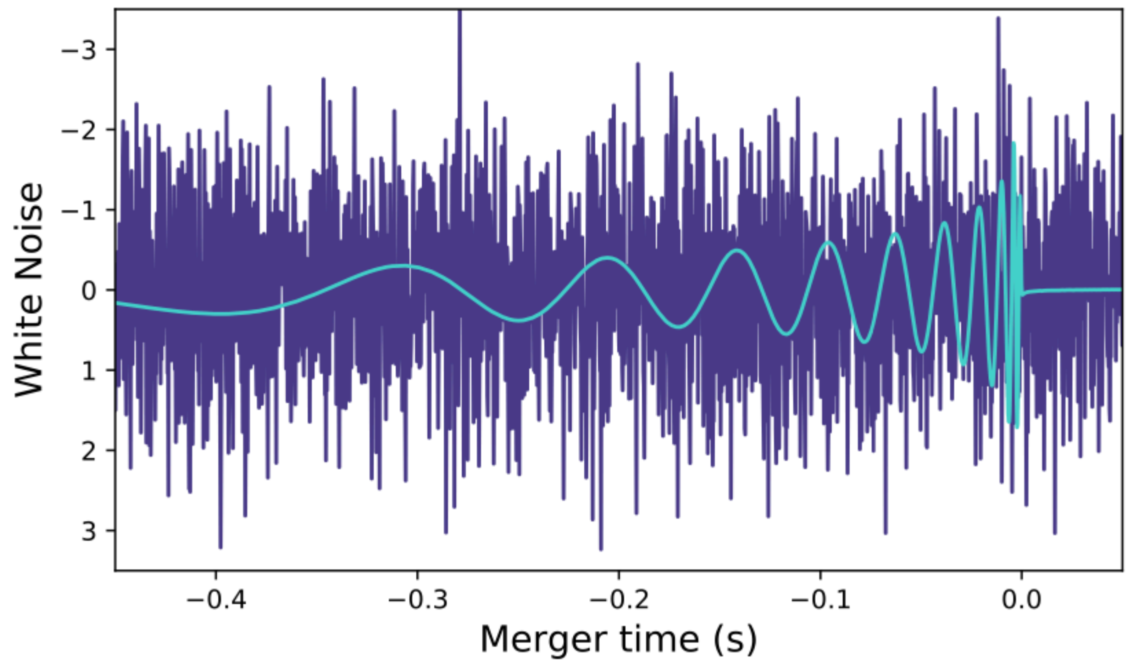
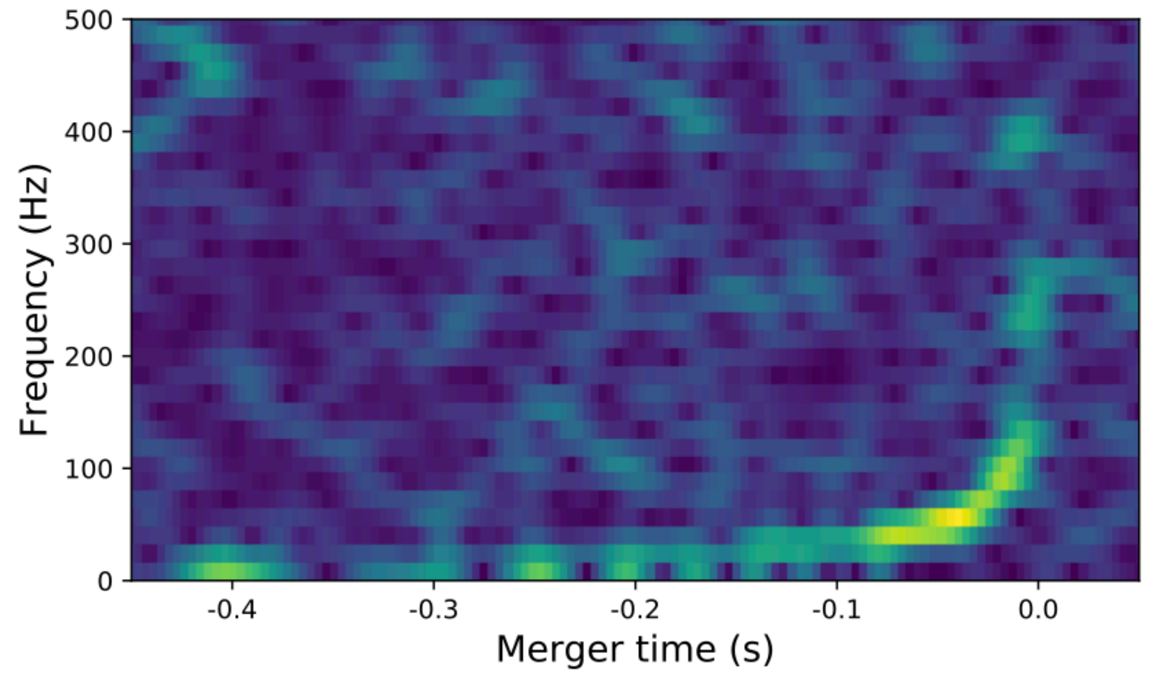


Figure 2. Receiver Operating Characteristic (ROC) curve for the neural network and the data set presented in this work. The dashed line represents the performance of the BLS preceded by a high-pass filter. The dotted–dashed line is the so-called "no-discrimination" line, corresponding to random guess.





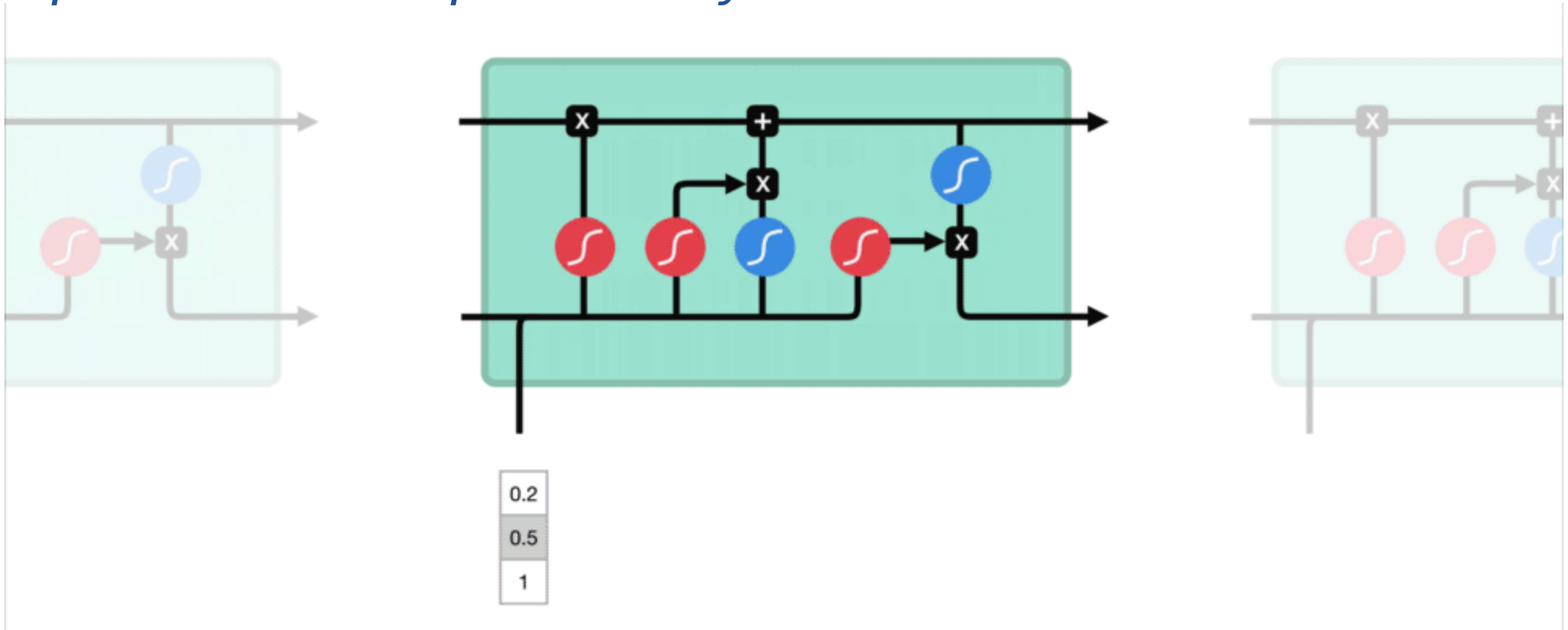


Deep Learning & Natural Language

- *Natural language processing is another big success of Deep Learning*
- *Based on recurrent neural networks*
  - *data ordered (time sequence, words in sentence, etc.)*
  - *data processed sequentially*

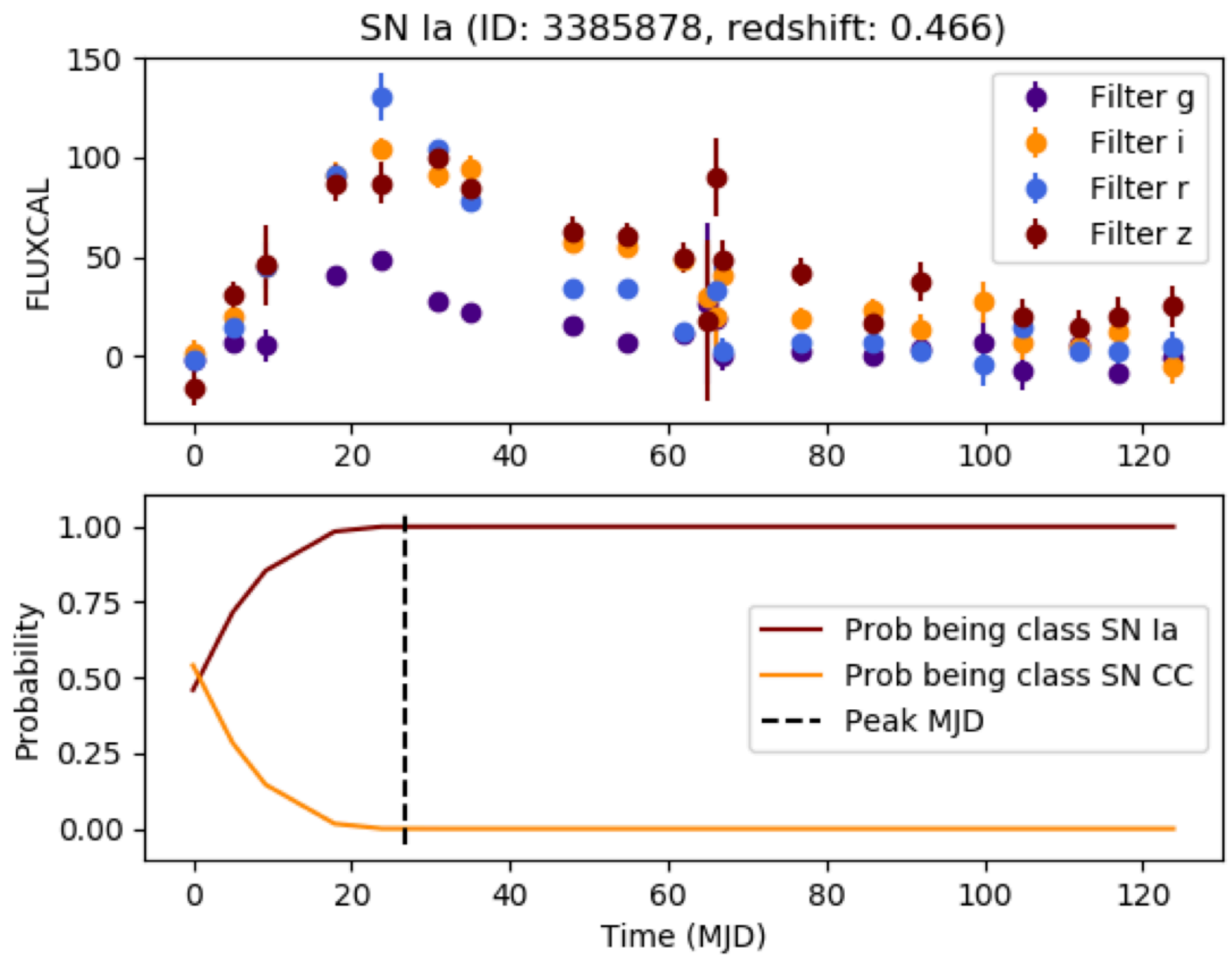
Deep Learning & Natural Language

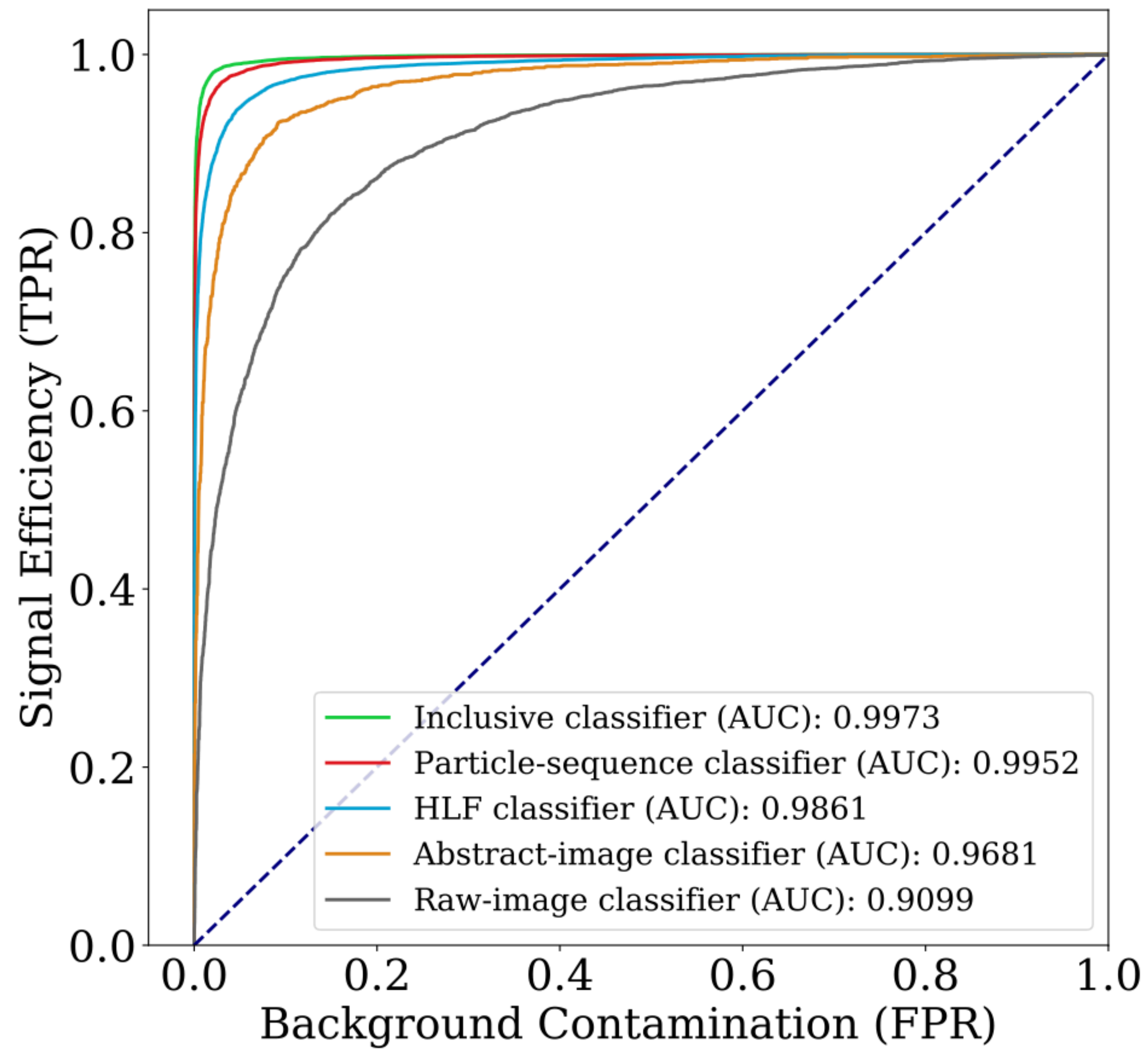
- *Natural language processing is another big success of Deep Learning*
- *Based on recurrent neural networks*
  - *data ordered (time sequence, words in sentence, etc.)*
  - *data processed sequentially*

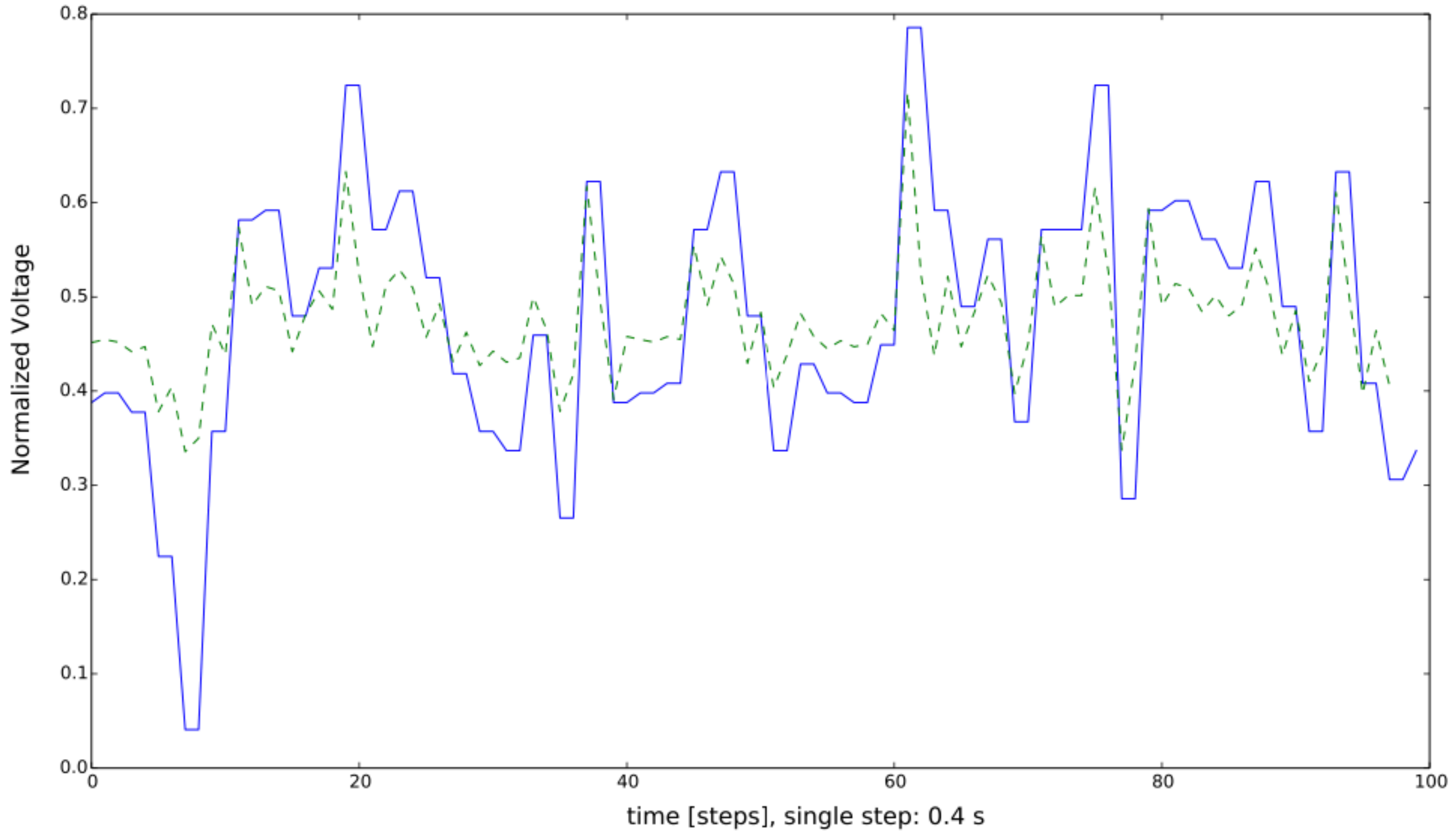


RNN in Science

- RNNs can be used to monitor time sequences and look for transient events
  - Supernovae detection
  - Monitoring LHC magnets
  - Event classification at LHC
  - ...



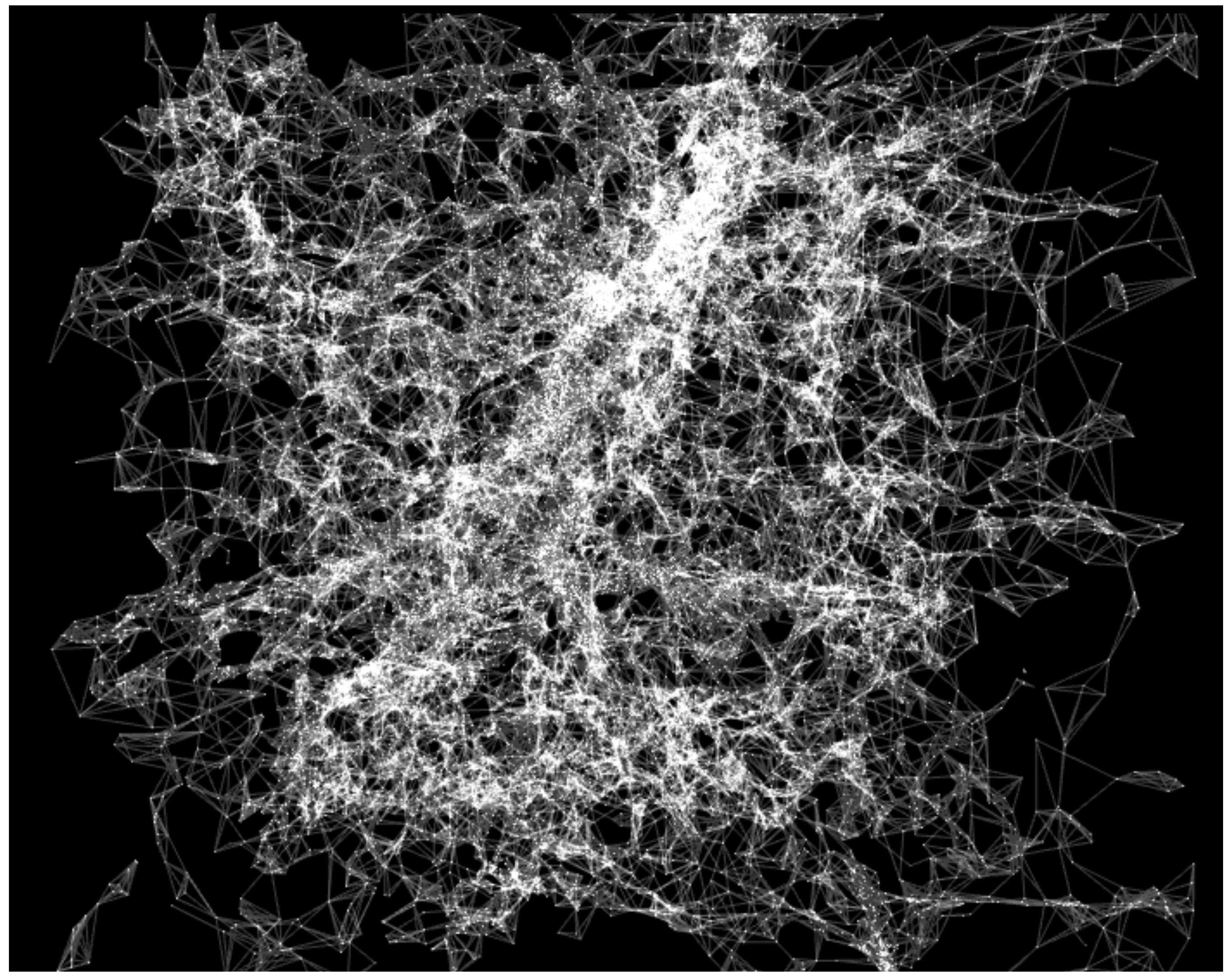


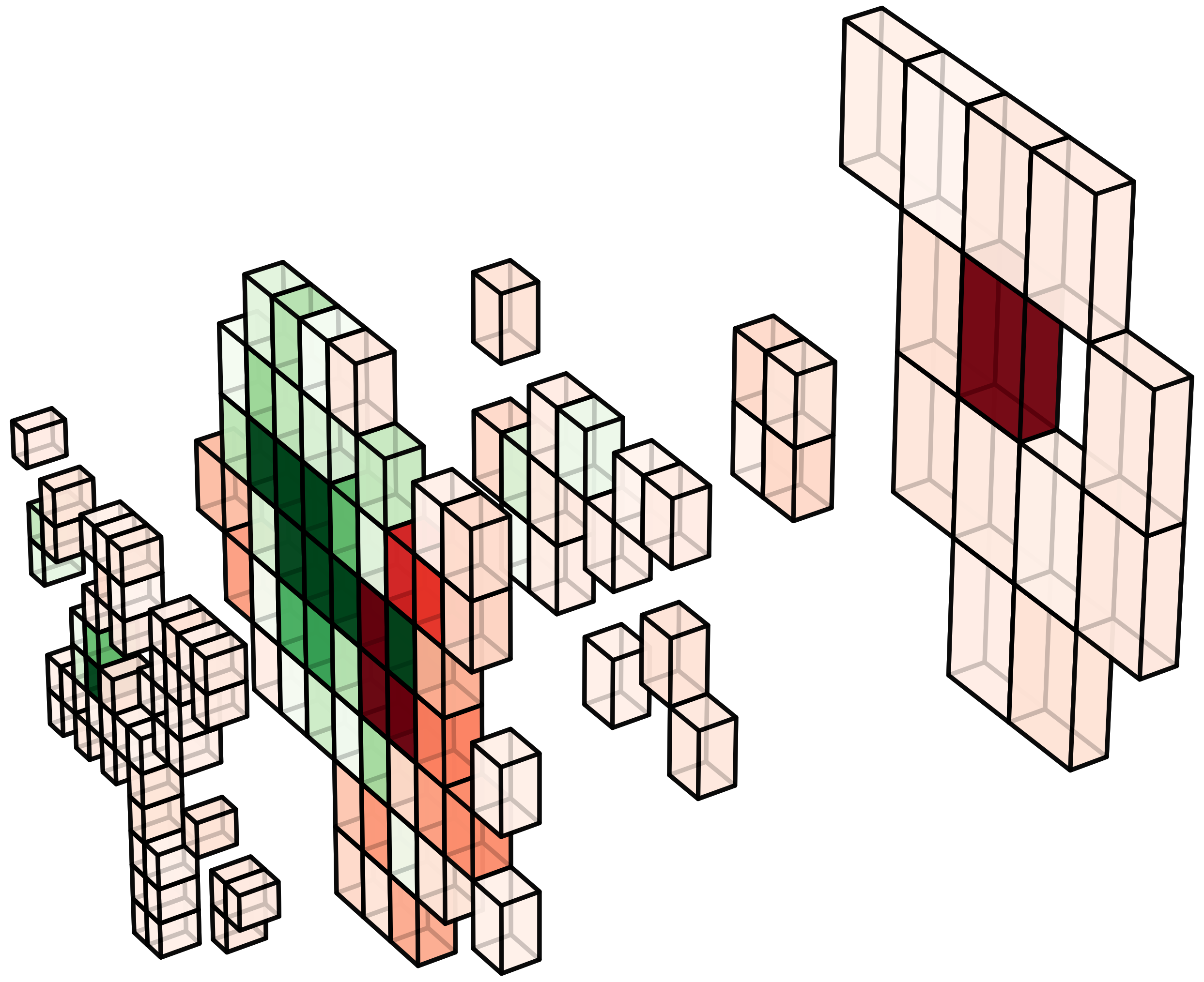


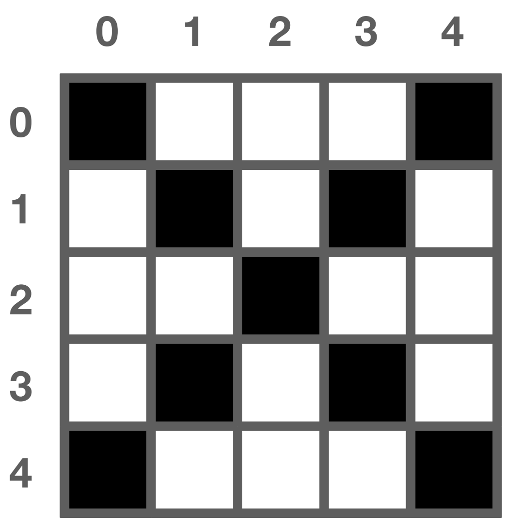
European Research Council

What about irregular data?

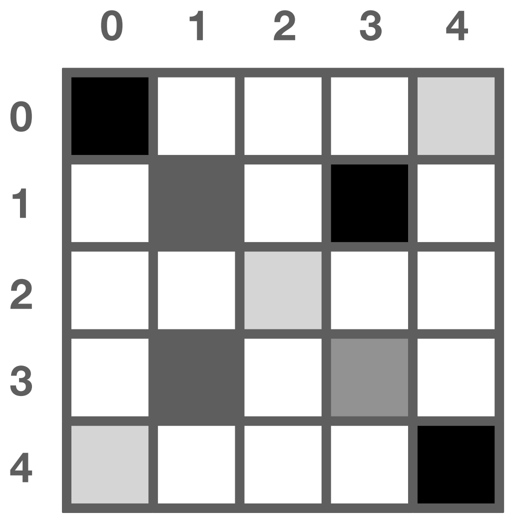
- Unfortunately, many scientific domains deal with data which are not regular arrays (neither images nor sequences)
  - Galaxies or star populations in sky
  - Sensors from HEP detector
  - Molecules in chemistry
- These data can all be seen as sparse sets in some abstract space
  - each element of the set being specified by some array of features
  - geometrical coordinates could be some of these features







0	0	1	1	2	3	3	4	4
0	4	1	3	2	1	3	0	4

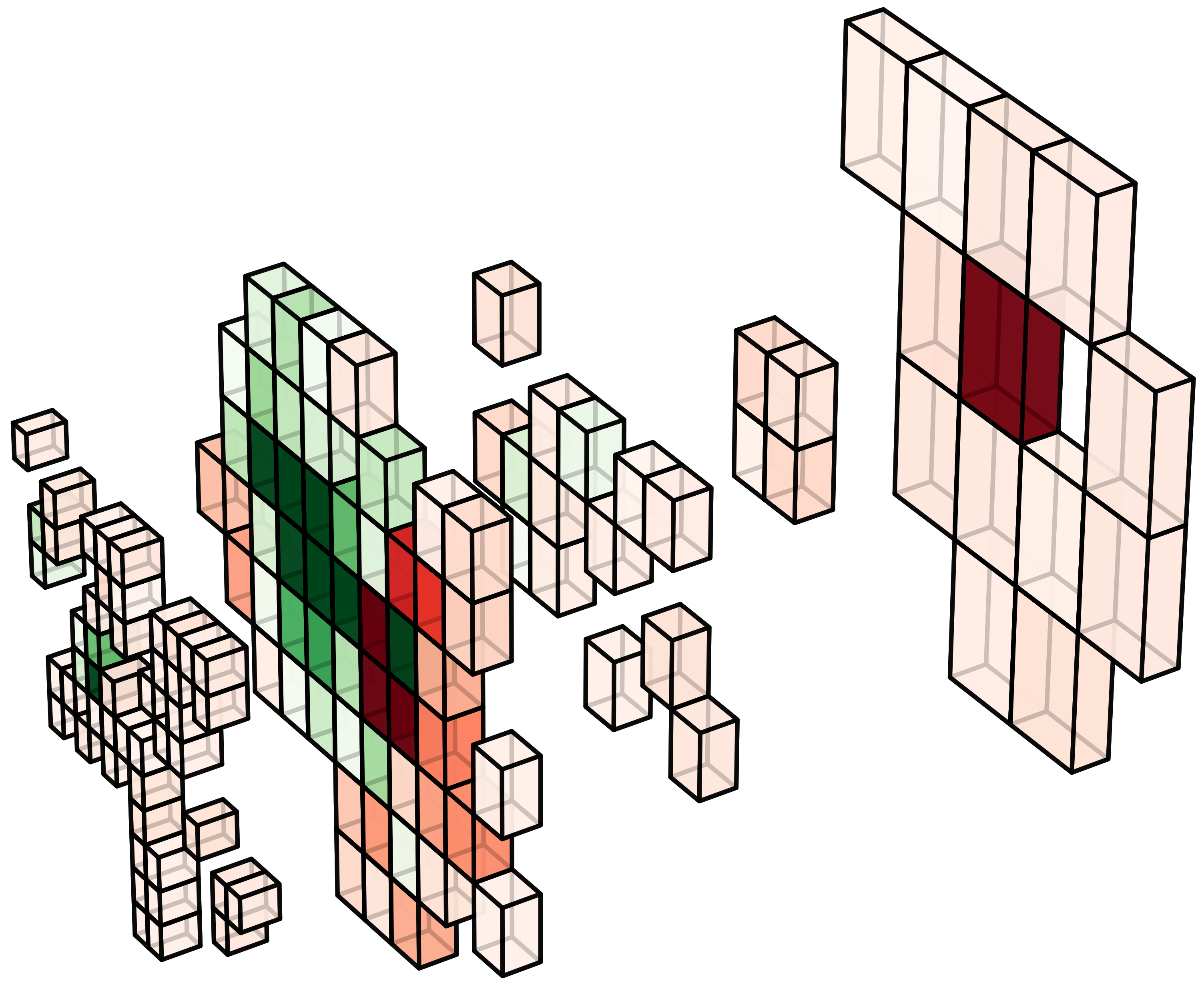


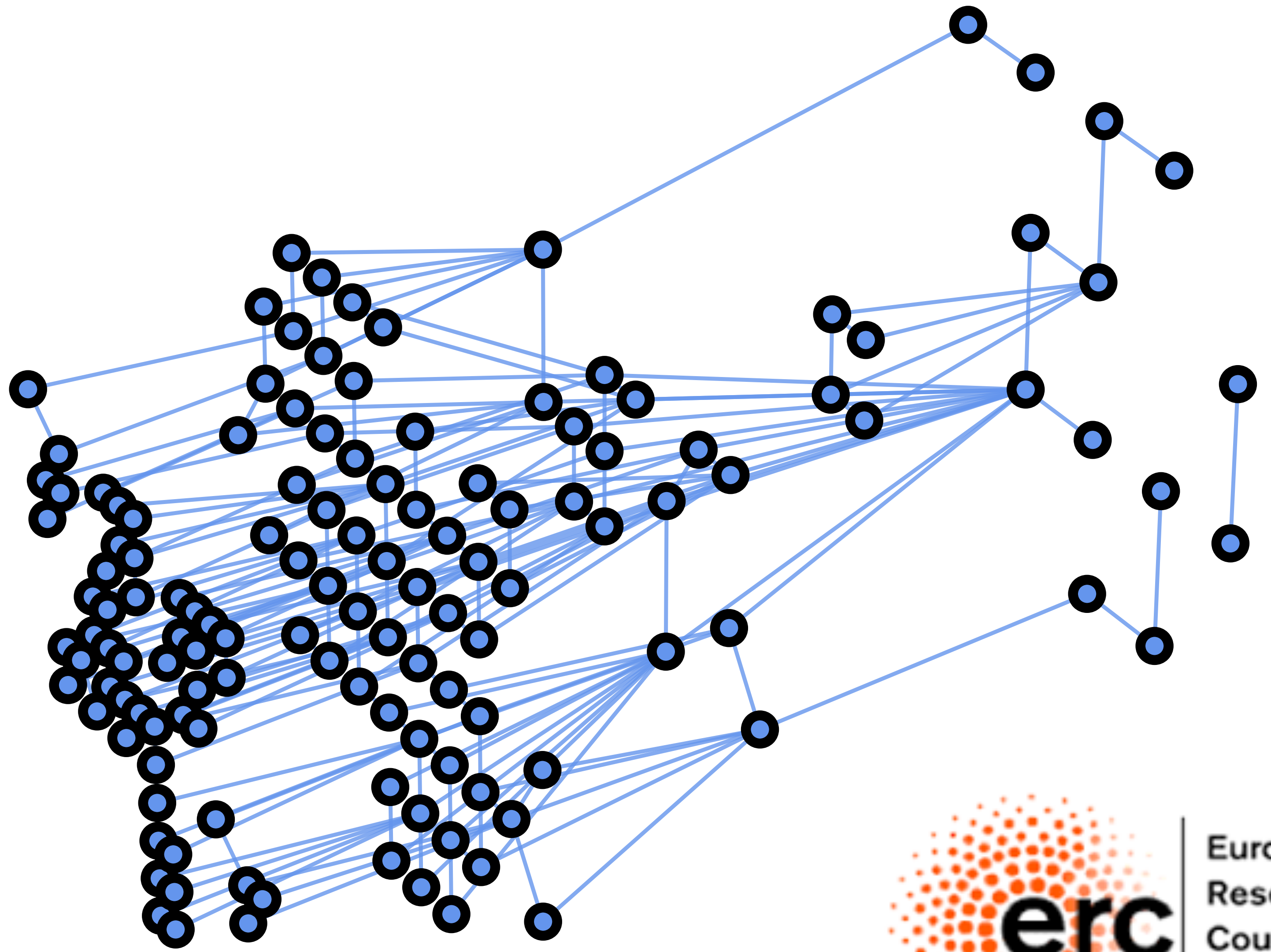
0	0	1	1	2	3	3	4	4
0	4	1	3	2	1	3	0	4
1.00	0.25	0.75	1.00	0.25	0.75	0.50	0.25	1.00

European Research Council

From Sets to Graphs

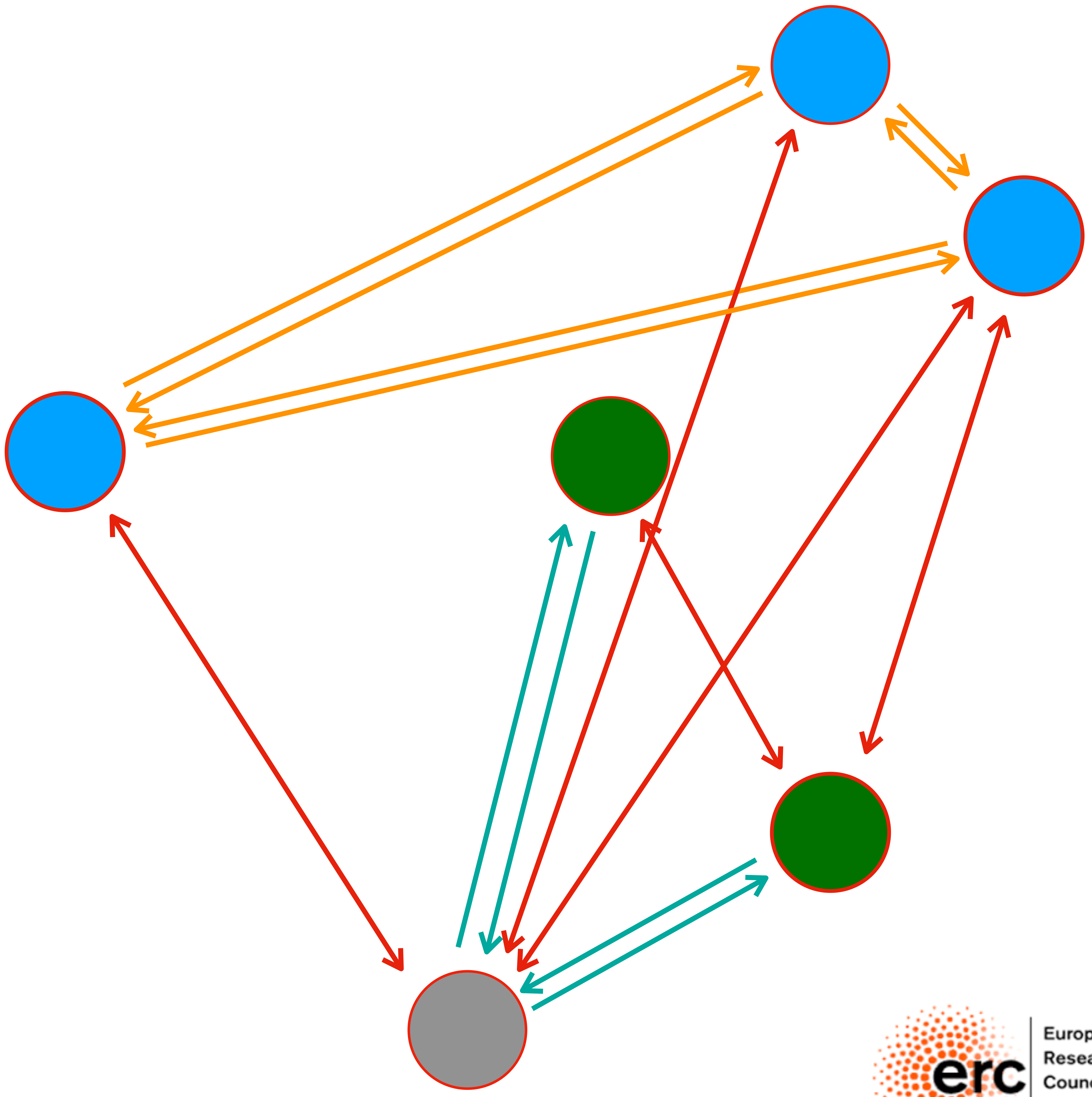
- Given such a set, we want to generalise the image representation as regular array that is fed to a CNN
  - Once that is done, we can generalise CNN itself
- For images, a lot of information is carried by pixels being next to each other. A metric is intrinsic in the data representation as image
- With a set, we need to specify a metric that tell us who is close to who in the abstract space of features that we have at hand
  - SOLUTION: connect elements of sets and learn (e.g., with a neural network) from data which connections are relevant





Building the Graph

- Each element of your set is a vertex V
- Edges E connect them
  - Edges can be made directional
  - Graphs can be fully connected (N^2)
  - Or you could use some criterion (e.g., nearest k neighbours in some space) to reduce number of connections
  - if more than one kind of vertex, you could connect only Vs of same kind, of different kind, etc
- The (V,E) construction is your graph. Building it, you could enforce some structure in your data
  - If you have no prior, then go for a directional fully connected graph



Summary

- *ML models are adaptable algorithms that are trained (and not programmed) to accomplish a task*
- *The training happens minimizing a loss function on a given sample*
- *The loss function has a direct connection to the statistical properties of the problem*
- *Deep Learning is the most powerful class of ML algorithms nowadays*
- *New architectures bring new opportunities for new applications*
- *It could be relevant to the future of HEP, e.g., to face the big-data challenge of the High-Luminosity LHC*

European Research Council

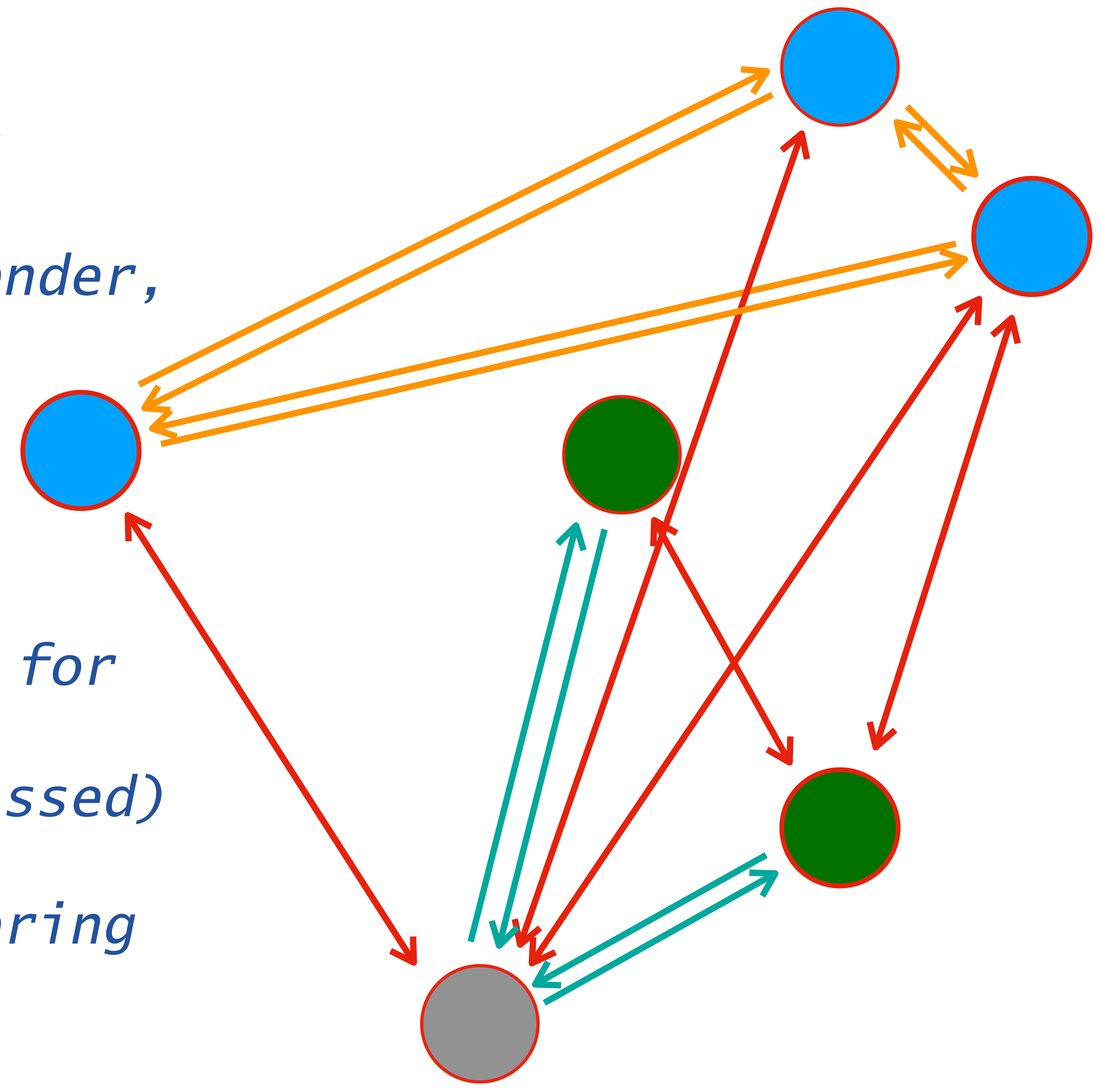
Backup

Message Passing

European
Research
Council

Learning from Graph: an example

- *Imagine a concrete example: given a social-media user, who will she vote for at the next elections?*
- *The graph here comes from social-media connections*
- *The features are what we know for a given user (gender, age, education, etc.)*
- *We want to gather information on someone from the social network of that person*
  - *we might know who some of her connections voted for*
- *We will use NNs to model the influence (message passed) of each user on her connection and learn from data which are the relevant connections. We are engineering features*
- *A final classifier will give us the answer we want*
- *You might become president with this + target pressure (ads, fake news, etc.)*

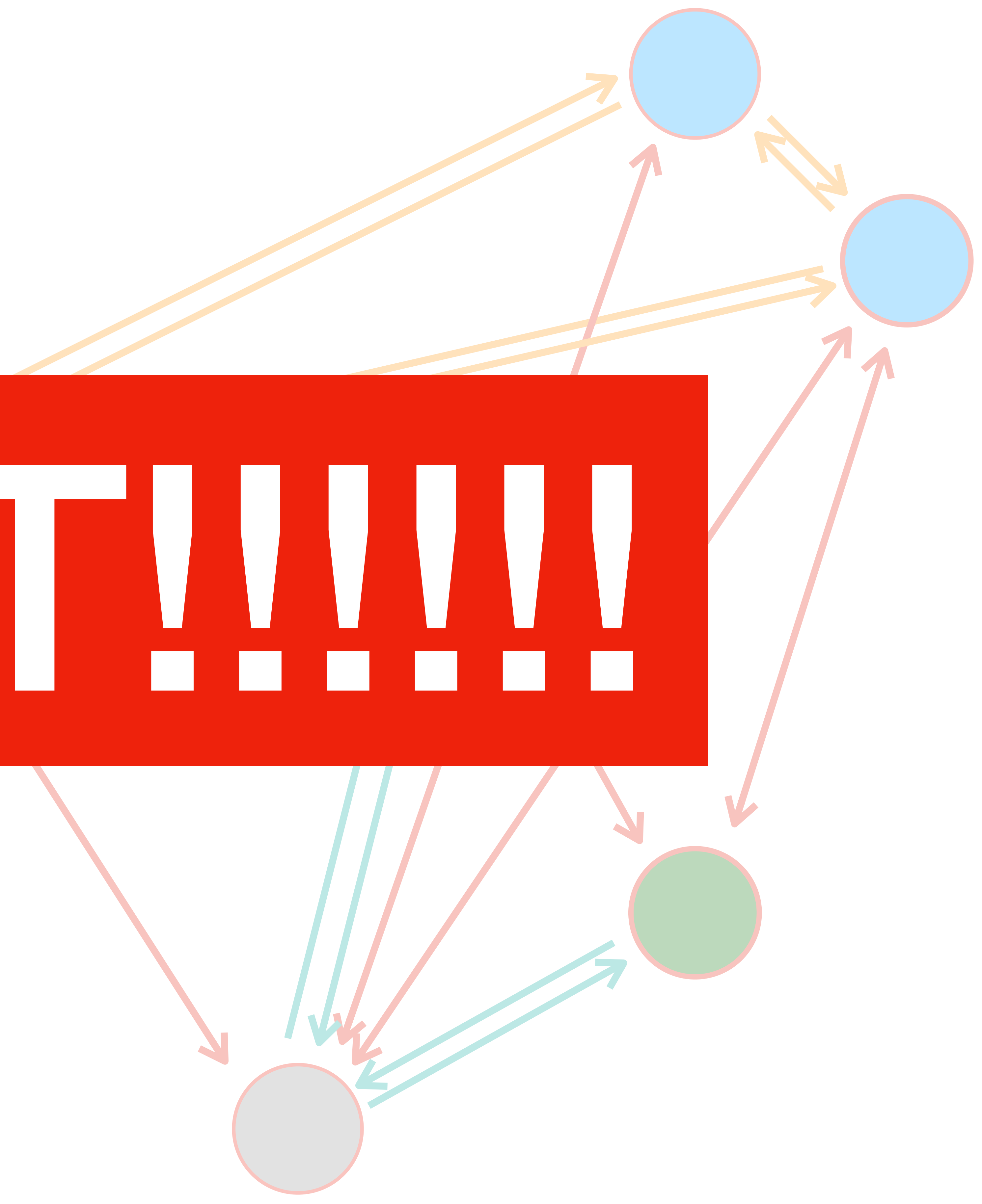


Learning from Graph: an example

- Imagine a concrete example: given a social-media user, who will she vote for at the next elections?
- The graph here comes from social-media connections
- The features are what we know for a given user (gender, age, education, etc.)
- We w…
  soci…
  - we …

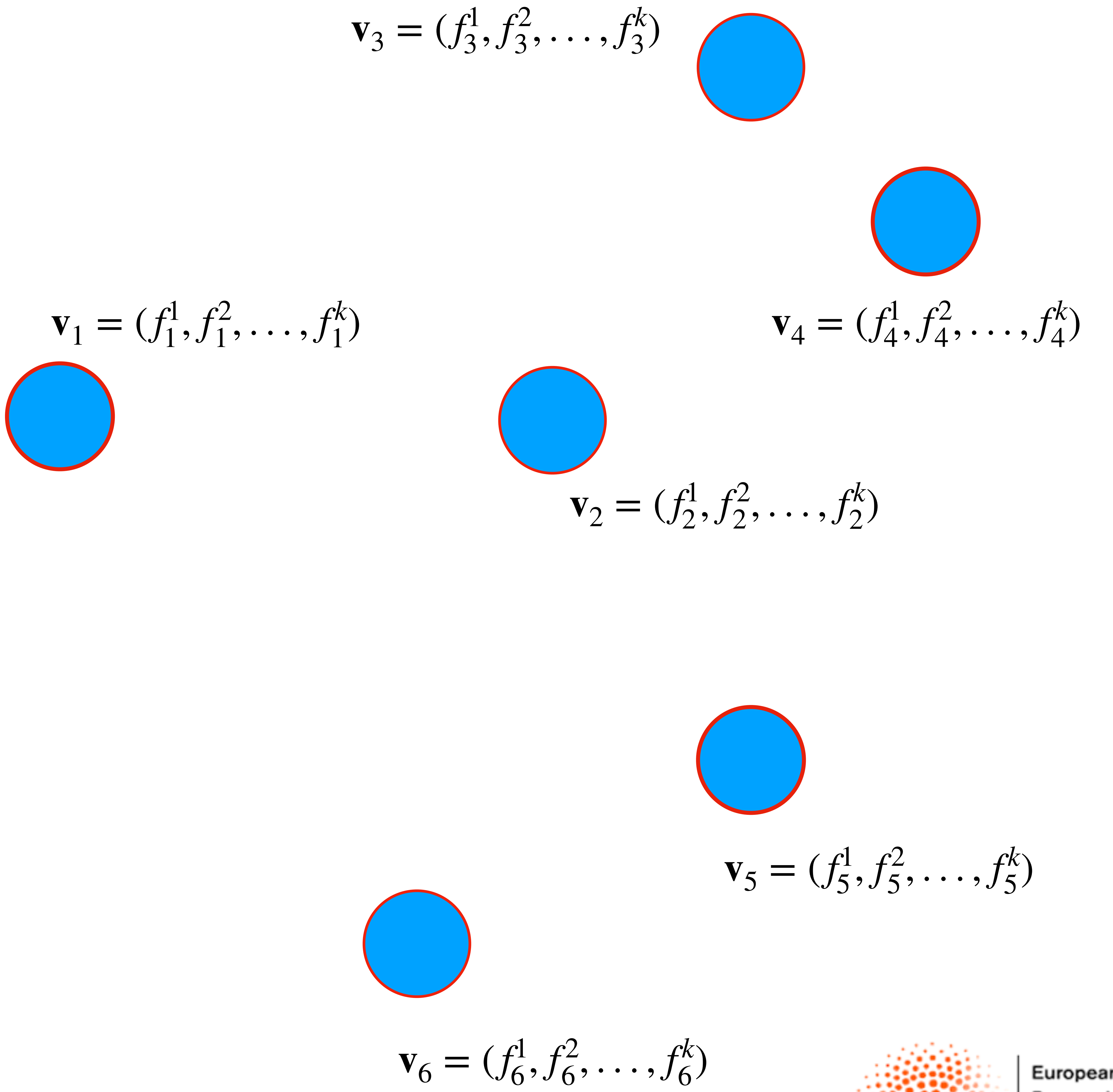
DON'T DO IT!!!!!!

- We will use NNs to model the influence (message passed) of each user on her connection and learn from data which are the relevant connections. We are engineering features
- A final classifier will give us the answer we want
- You might become president with this + target pressure (ads, fake news, etc.)



Graph Networks

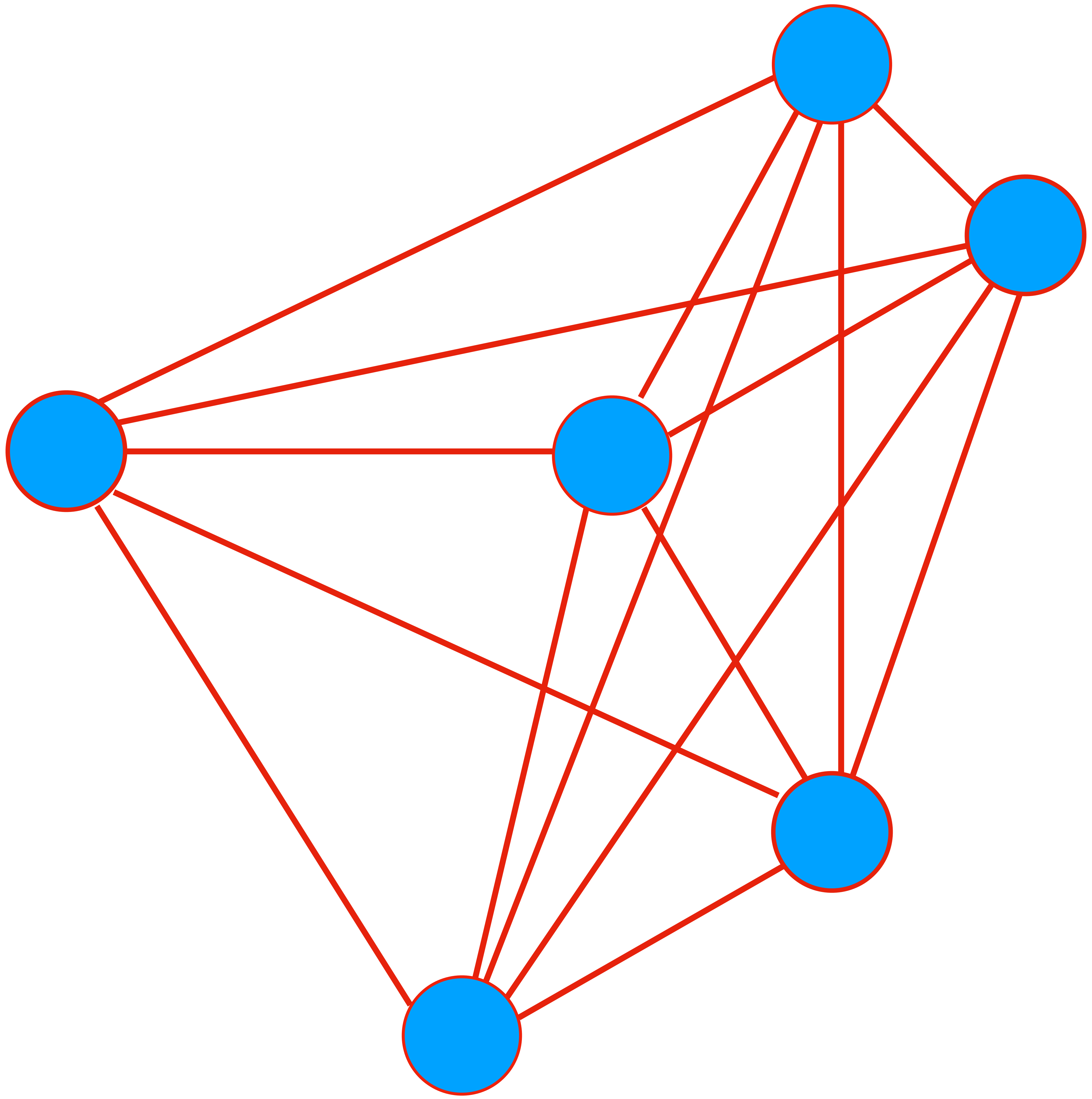
- *Graphs Nets are architectures based on an abstract representation of a given dataset*
  - ***Each example in a dataset is represented as a set of vertices***
  - ***Each vertex is embedded in the graph as a vector of features***



https://arxiv.org/pdf/1704.01212.pdf

Graph Networks

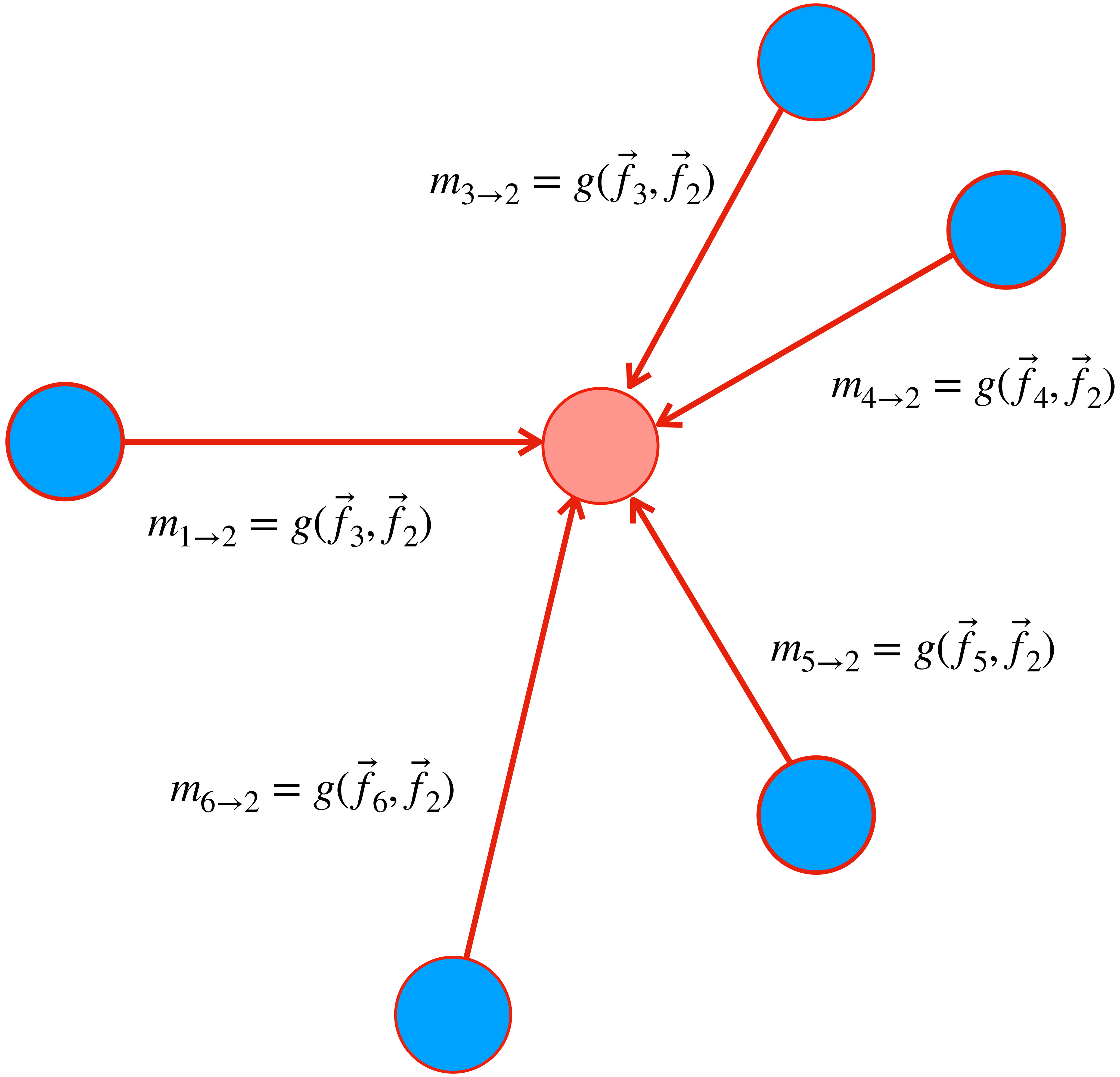
- *Graphs Nets are architectures based on an abstract representation of a given dataset*
  - *Each example in a dataset is represented as a set of vertices*
  - *Each vertex is embedded in the graph as a vector of features*
  - ***Vertices are connected through links (edges)***



https://arxiv.org/pdf/1704.01212.pdf

Graph Networks

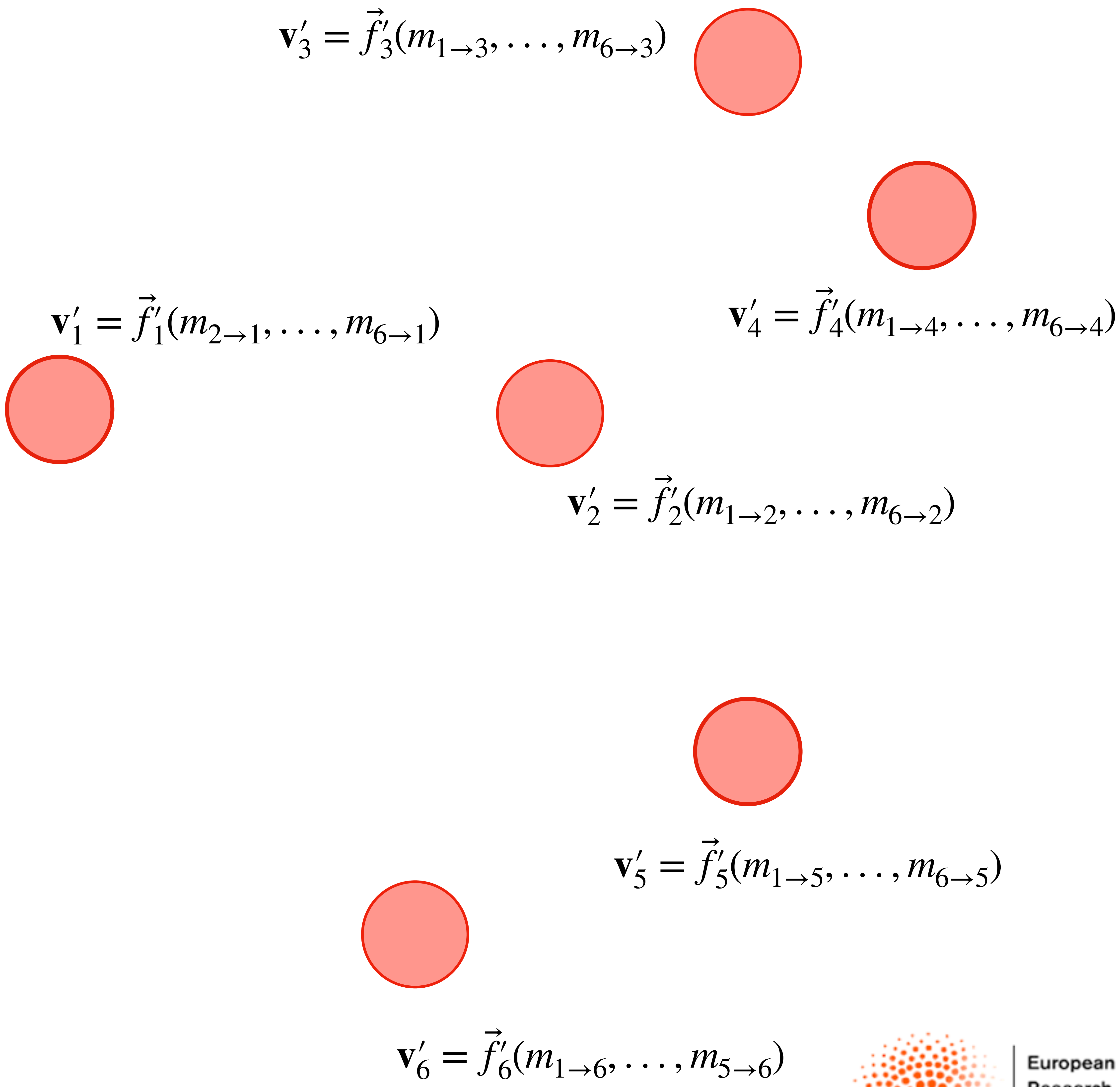
- *Graphs Nets are architectures based on an abstract representation of a given dataset*
  - *Each example in a dataset is represented as a set of vertices*
  - *Each vertex is embedded in the graph as a vector of features*
  - *Vertices are connected through links (edges)*
  - ***Messages are passed through links and aggregated on the vertices***



https://arxiv.org/pdf/1704.01212.pdf

Graph Networks

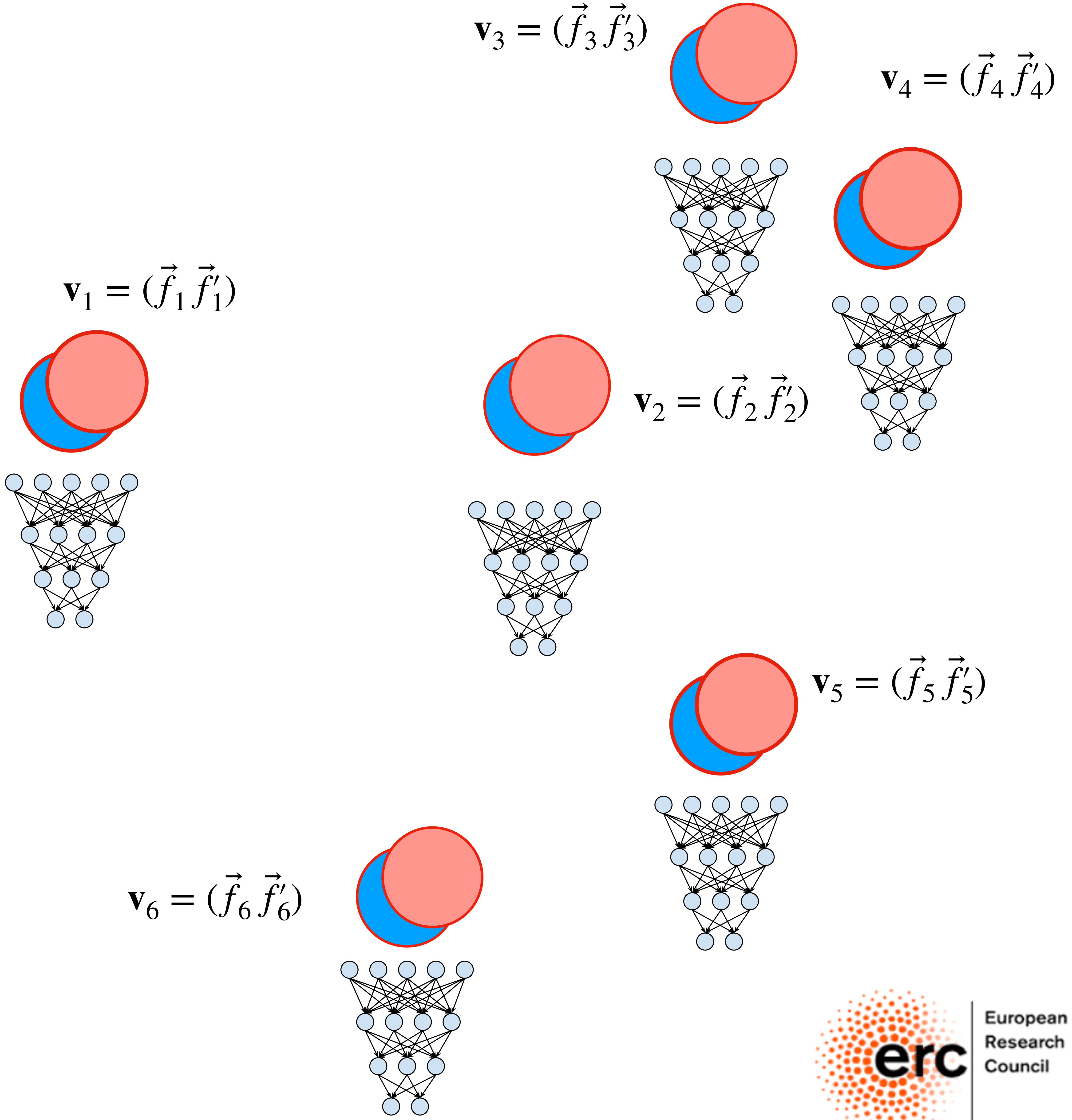
- *Graphs Nets are architectures based on an abstract representation of a given dataset*
  - *Each example in a dataset is represented as a set of vertices*
  - *Each vertex is embedded in the graph as a vector of features*
  - *Vertices are connected through links (edges)*
  - *Messages are passed through links and aggregated on the vertices*
  - ***A new representation of each node is created, based on the information gathered across the graph***



https://arxiv.org/pdf/1704.01212.pdf

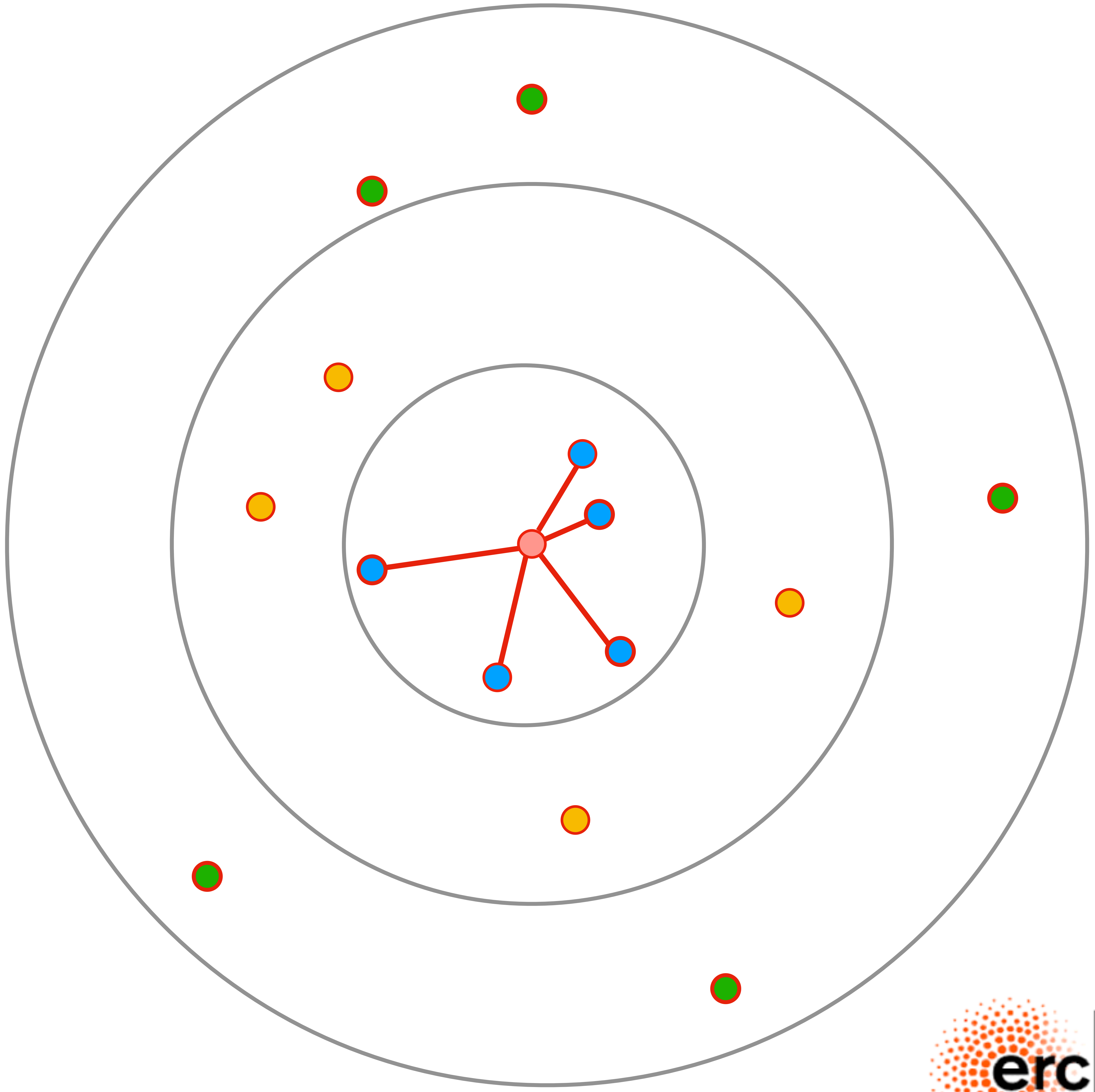
The inference step

- The inference step usually happens on each vertex
- But, depending on the problem, it might happen across the graph
- Usually, this is done with a DNN taking
    - the initial features f_i
    - the learned representation f_i'
    - [optional] some ground-truth label (for classifiers)



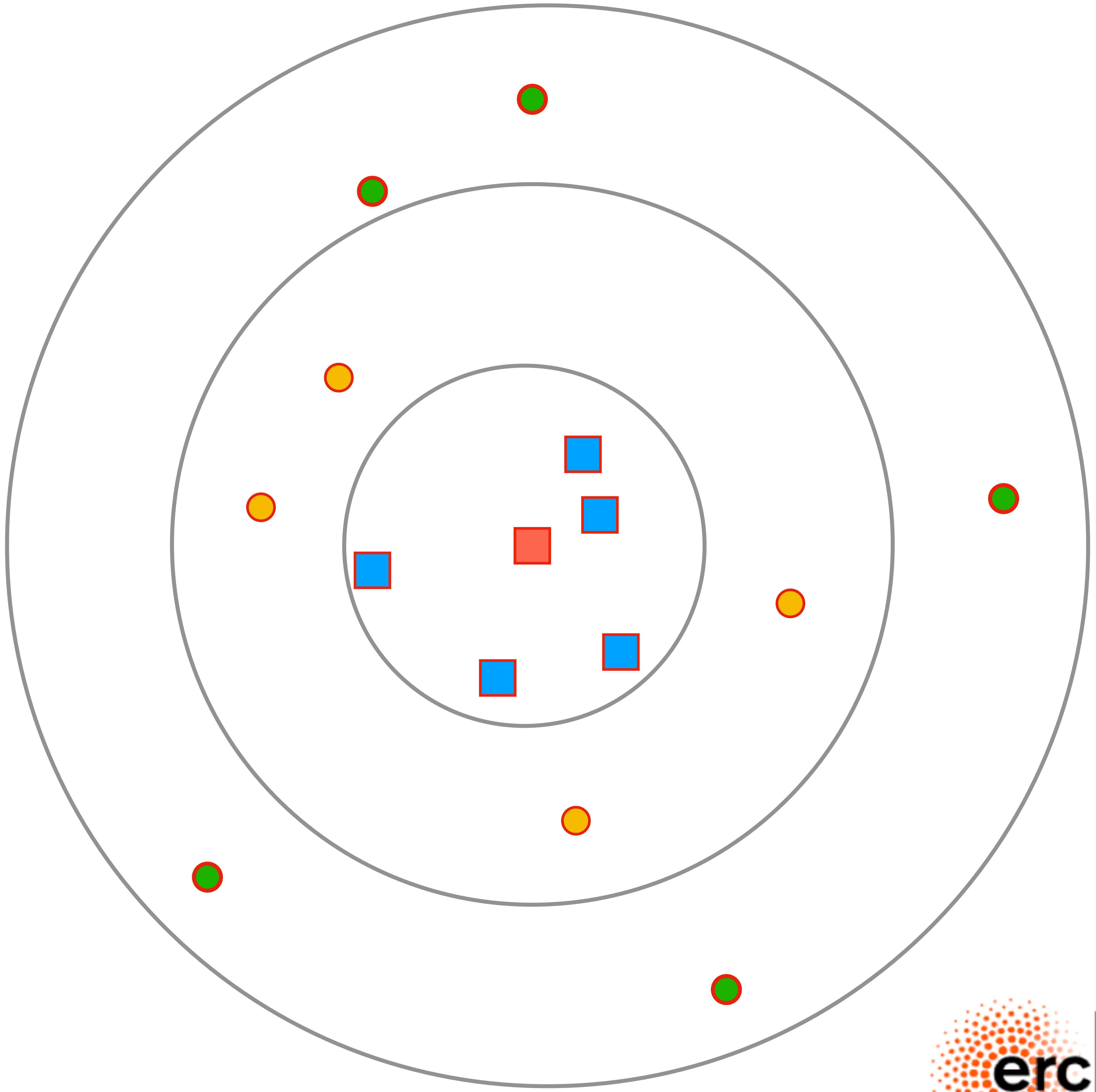
...and repeat

- *Once message is passed, aggregated at each vertex V and processed, it creates a new representation of each vertex*
  - *You could start from coordinates in real space + some feature*
  - *Build function of them*
  - *Build functions of functions of them*
  - *At each step, you improve knowledge on your vertex V*



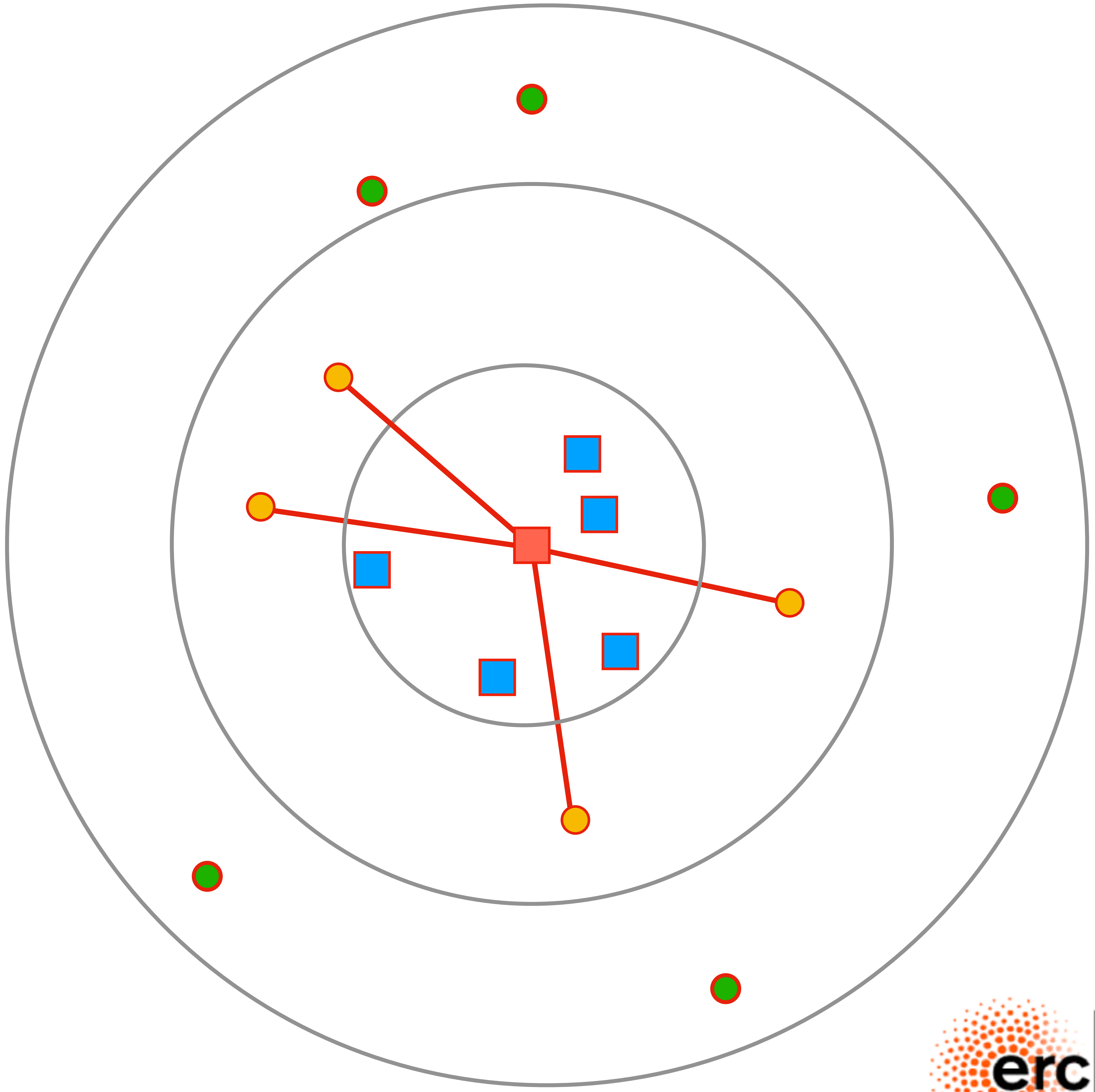
...and repeat

- *Once message is passed, aggregated at each vertex V and processed, it creates a new representation of each vertex*
    - *You could start from coordinates in real space + some feature*
    - *Build function of them*
    - *Build functions of functions of them*
    - *At each step, you improve knowledge on your vertex V*



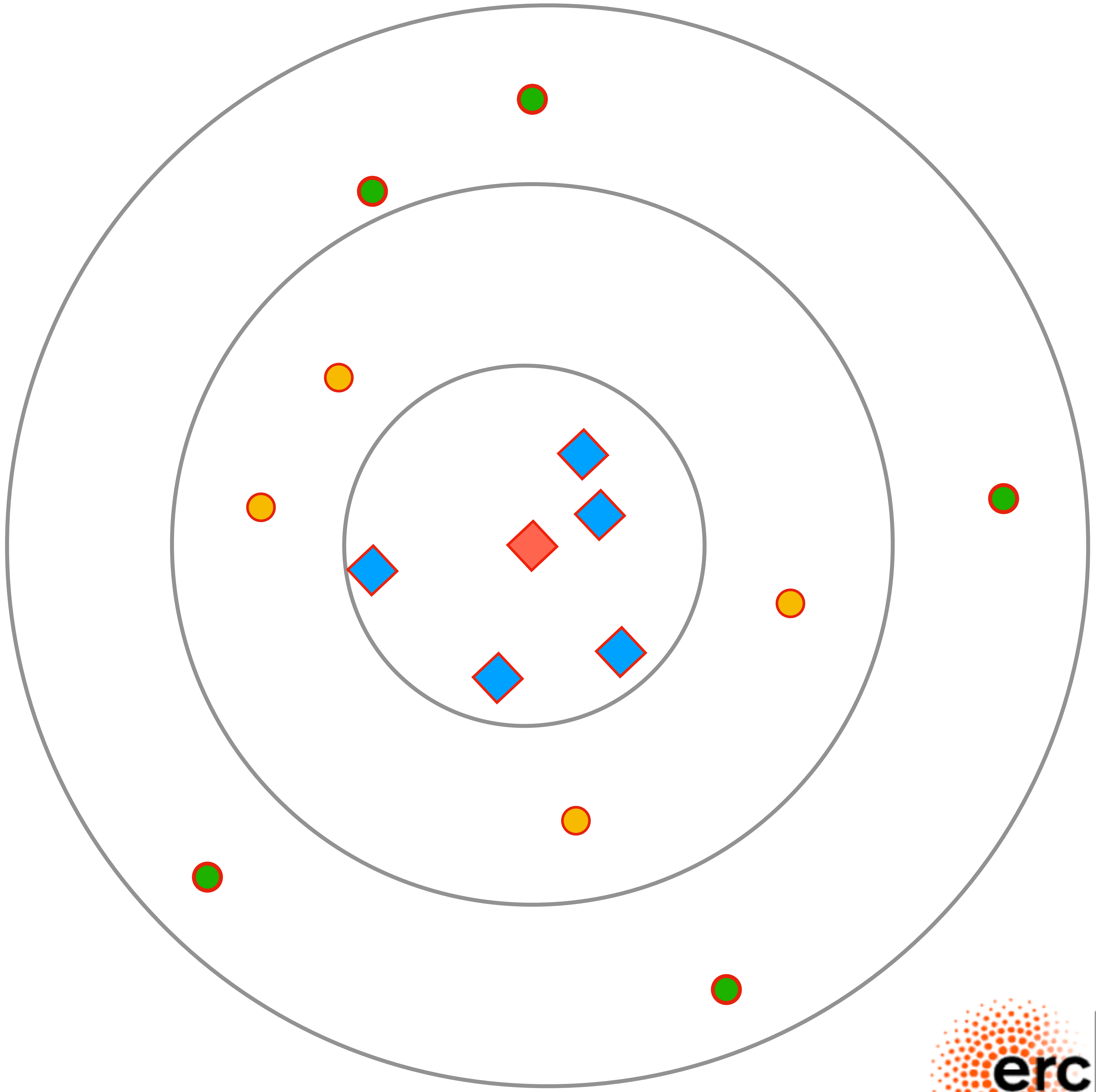
...and repeat

- Once message is passed, aggregated at each vertex V and processed, it creates a new representation of each vertex
  - You could start from coordinates in real space + some feature
  - Build function of them
  - Build functions of functions of them
  - At each step, you improve knowledge on your vertex V



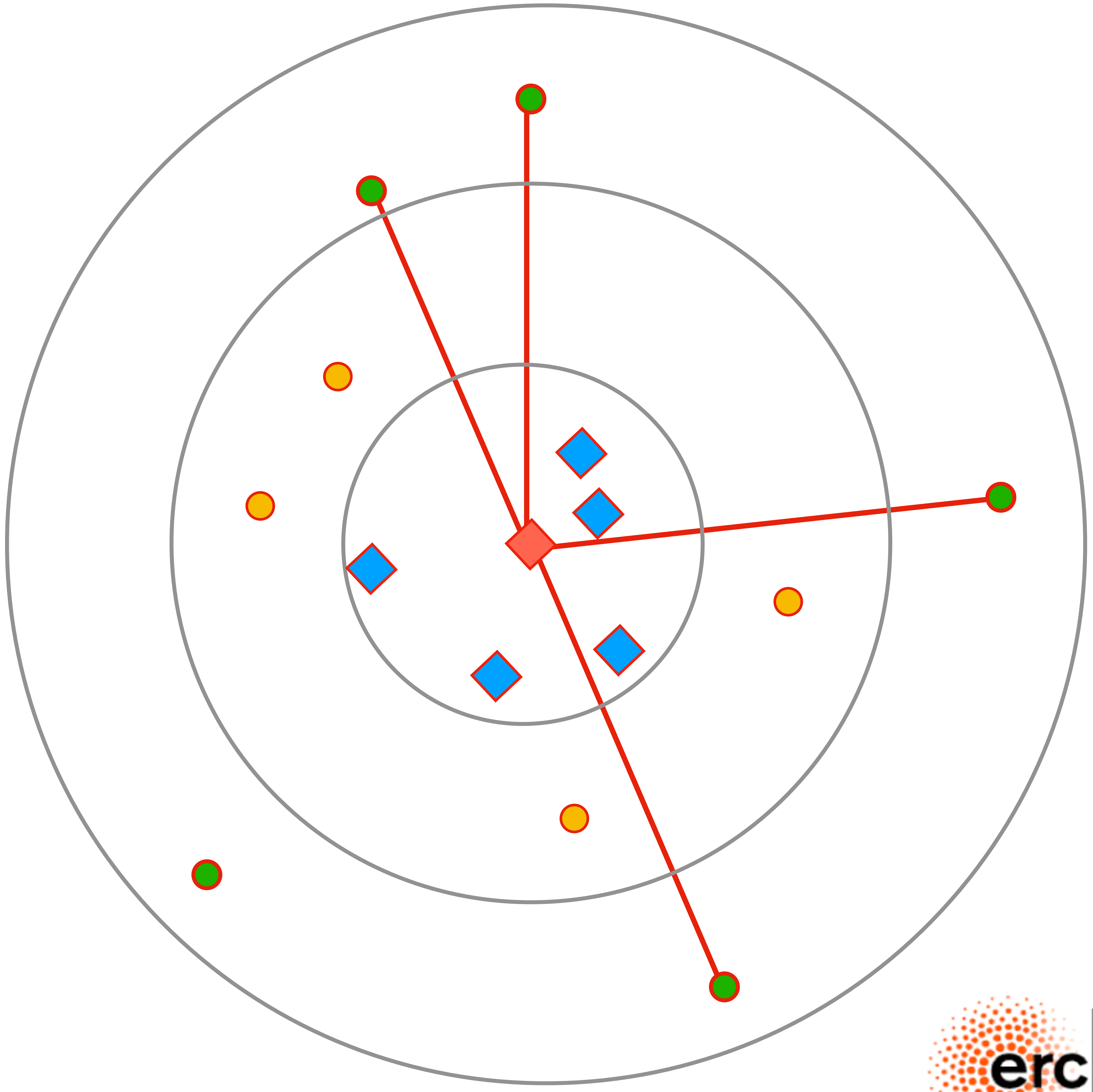
...and repeat

- Once message is passed, aggregated at each vertex V and processed, it creates a new representation of each vertex
  - You could start from coordinates in real space + some feature
  - Build function of them
  - Build functions of functions of them
  - At each step, you improve knowledge on your vertex V



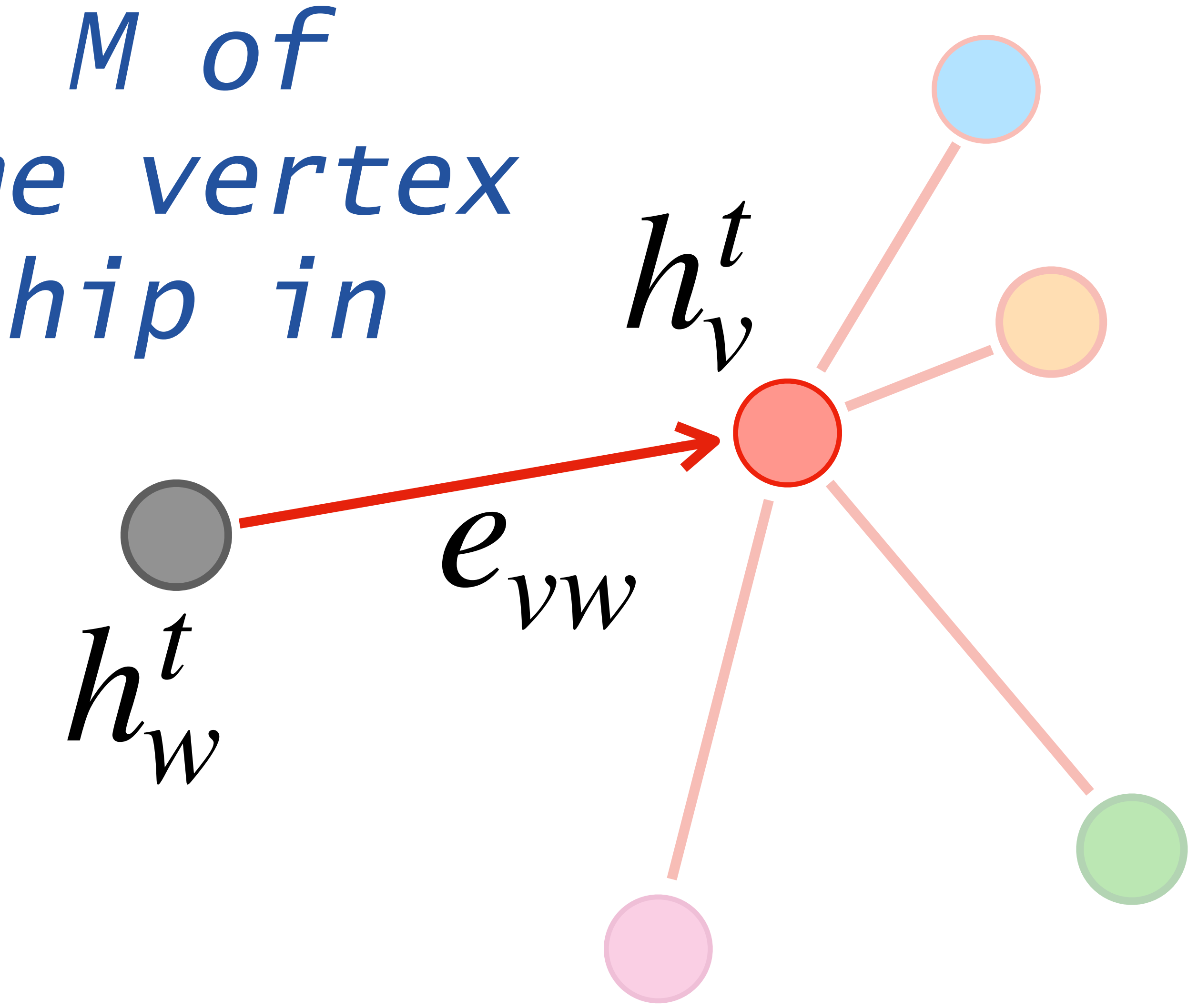
...and repeat

- *Once message is passed, aggregated at each vertex V and processed, it creates a new representation of each vertex*
  - *You could start from coordinates in real space + some feature*
  - *Build function of them*
  - *Build functions of functions of them*
  - *At each step, you improve knowledge on your vertex V*



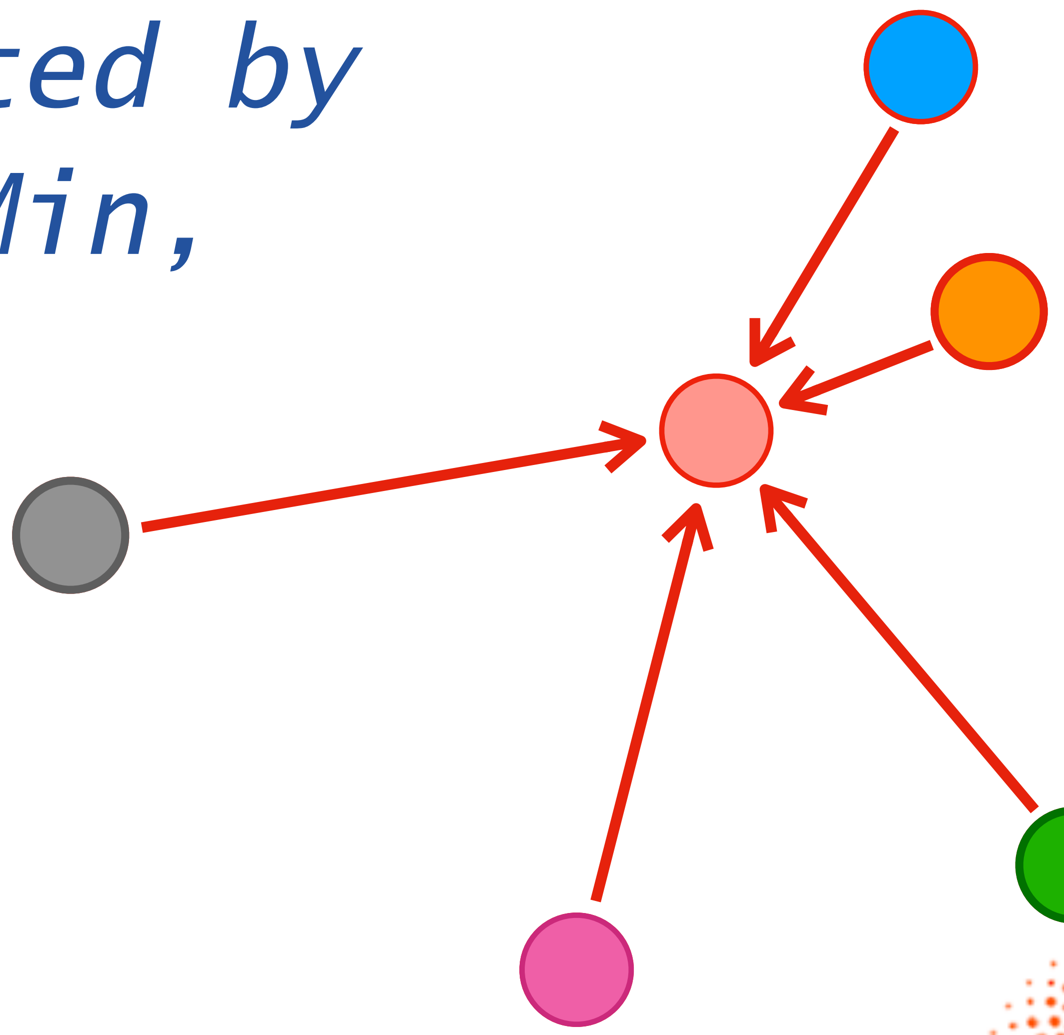
With equations…

- *Your message at iteration t is some function M of the sending and receiving features, plus some vertex features (e.g., business relation vs friendship in social media)*



$$M_t(h_v^t, h_w^t, e_{vw})$$

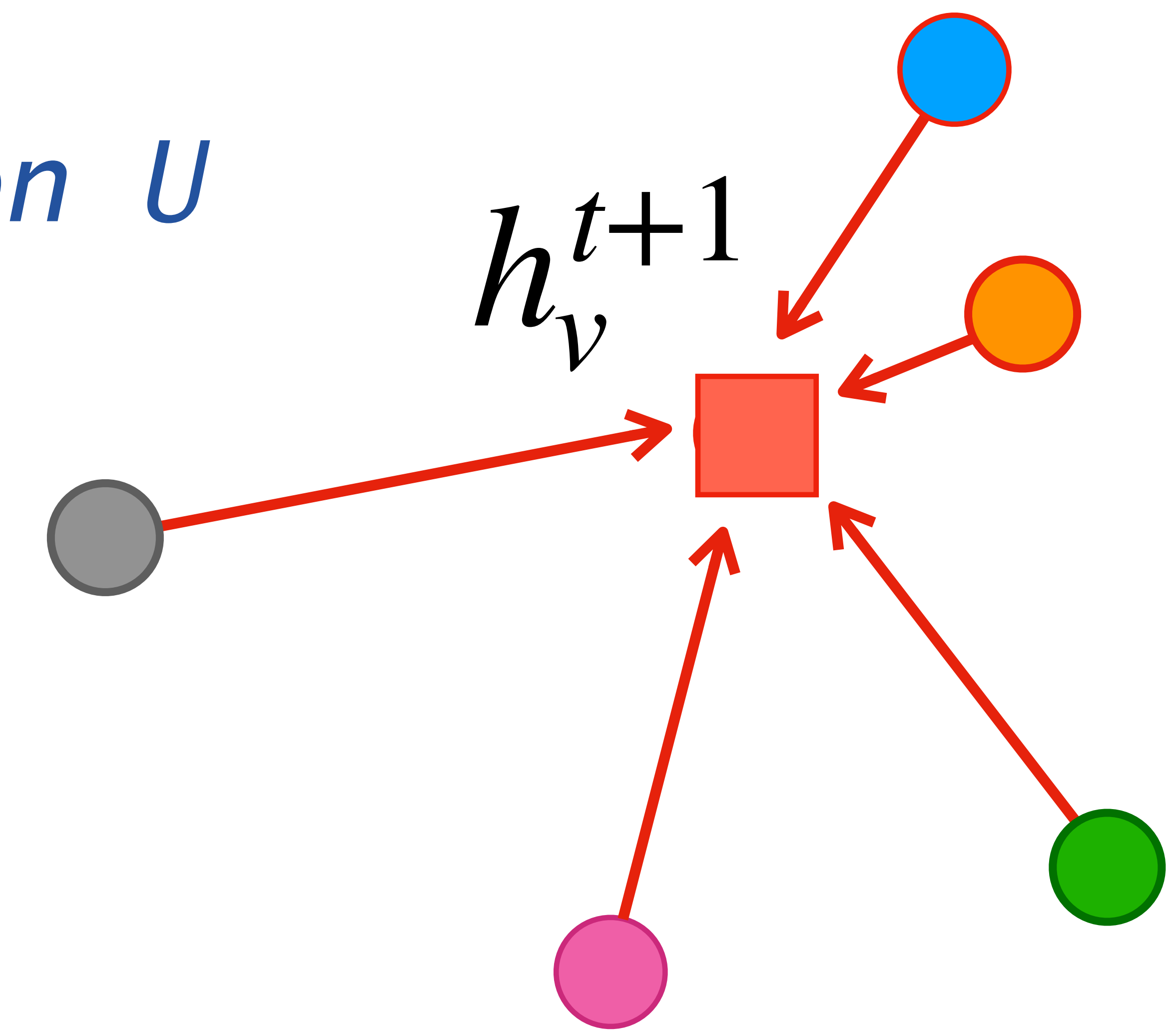
- *The message carried to a vertex v is aggregated by some function (typically sum, but also Max, Min, etc.)*



$$m_v^{t+1} = \sum_{w \in G(v)} M_t(h_v^t, h_w^t, e_{vw})$$

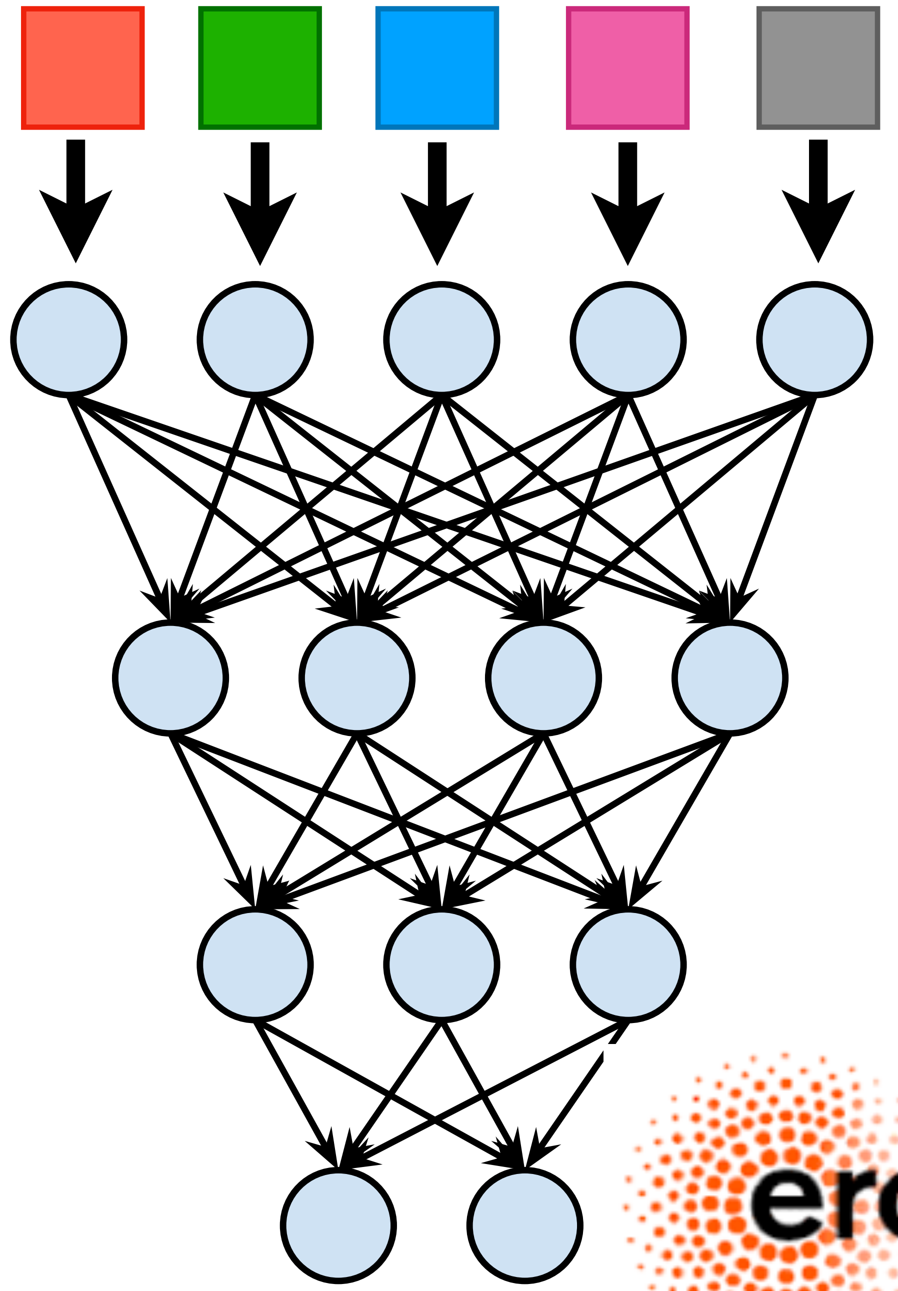
With equations…

- The state of vertex v is updated by some function U of the current state and the gathered message



$$h_v^{t+1} = U_t(h_v^t, m_v^{t+1})$$

- After T iterations, the last representations of the graph vertices are used to derive the final output answering the question asked (classification, regression, etc.), typically through a NN



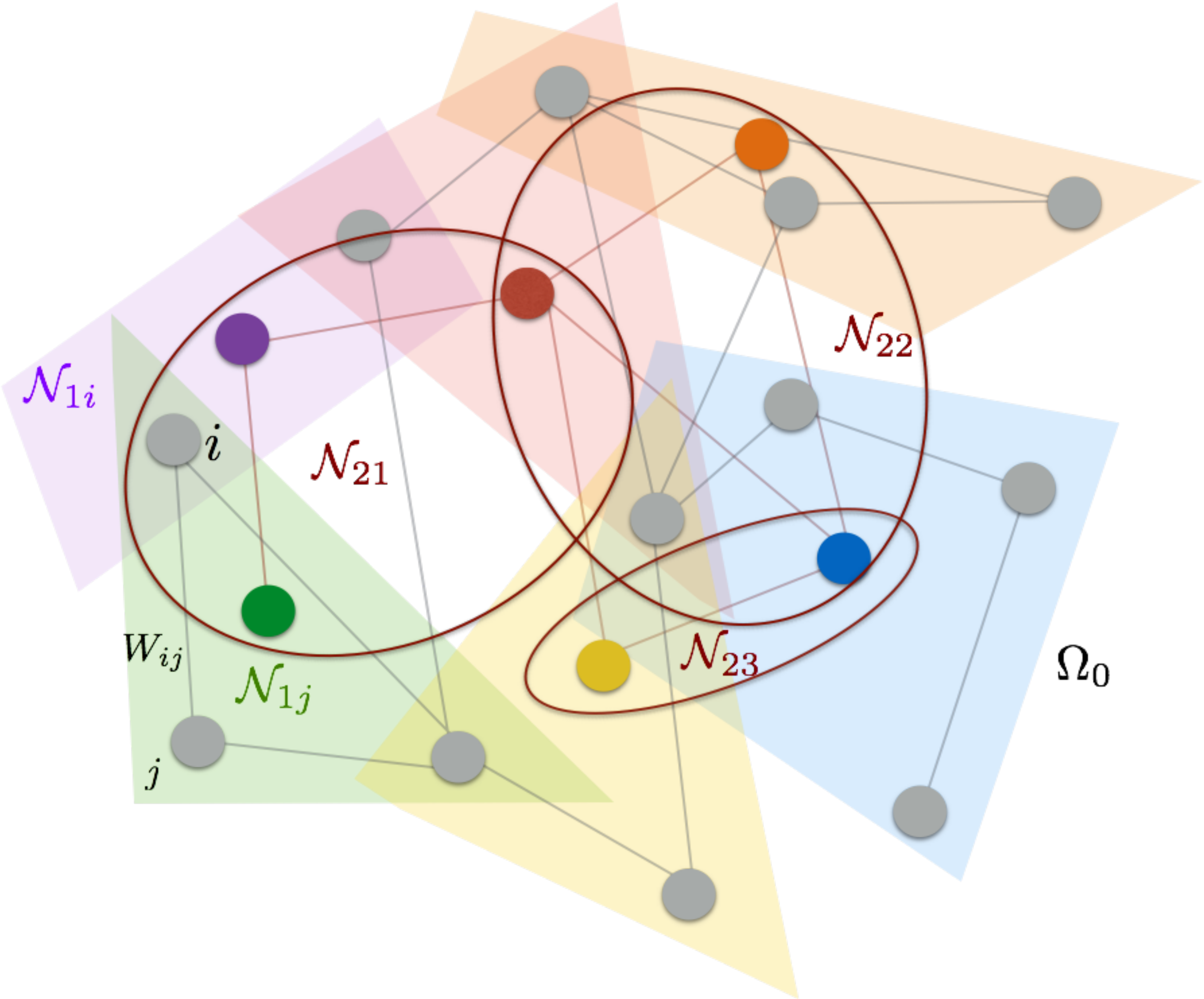
$$\hat{y} = R(h_v^T \mid v \in G)$$

Learning Message

- Typically, the M, U, and R functions are learned from data
    - Expressed as neural networks (fully connected NNs, recurrent NNs, etc.)
    - Which networks to use depends on the specific problem, as much as the graph-building rules
- But you could inject domain knowledge in the game
    - You might know that SOME message is carried by some specific functions (e.,g., Netwon's low for N-body system simulation)
    - You could then use analytic functions for some message
    - You could still use a learned function for other messages
- The trick is dealing with differentiable functions not to spoil your back propagation
    - Graph networks become a tool for probabilistic programming

A little bit of History

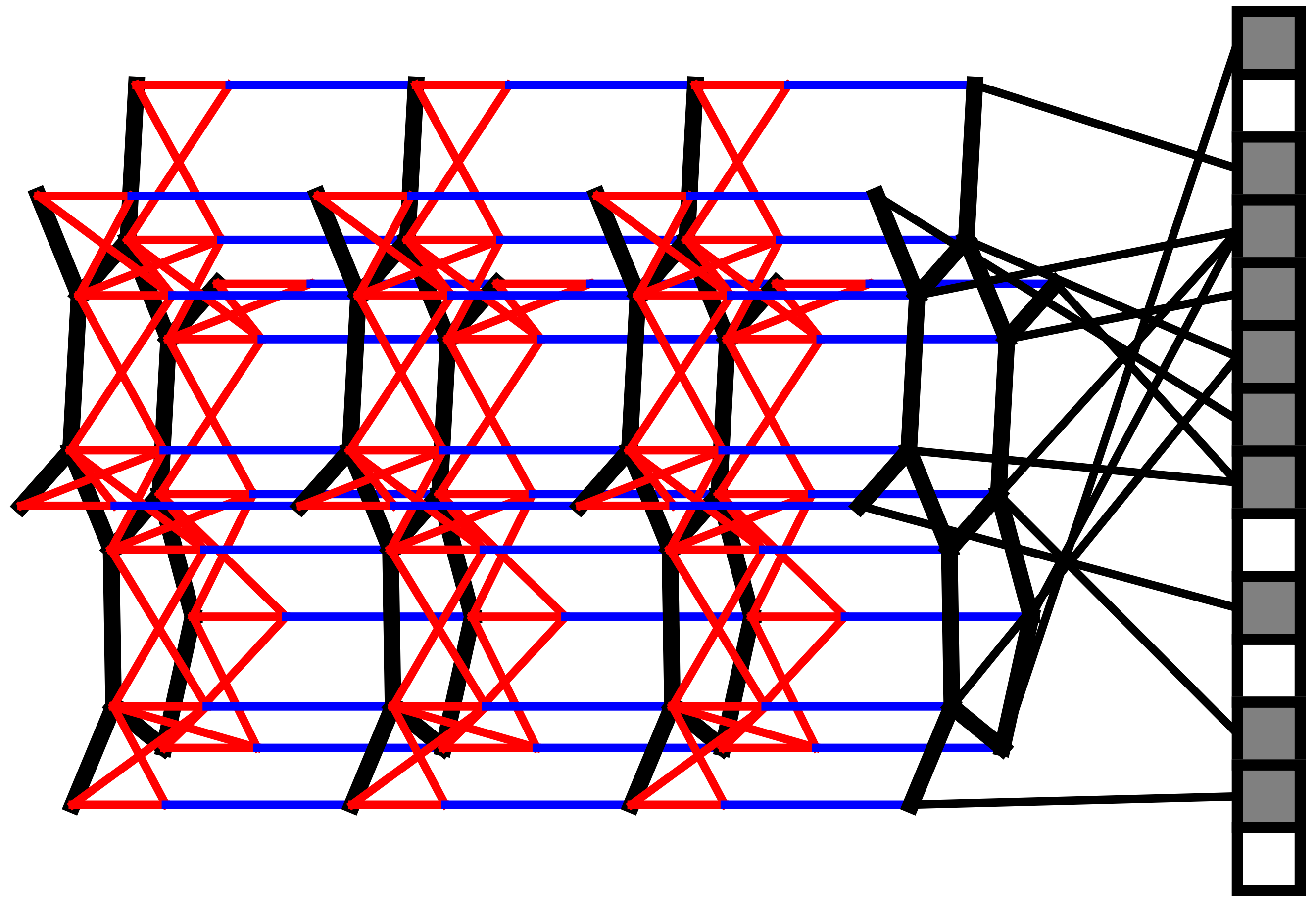
- *(in this millenium) Graph networks started (as often it is the case) with a Yann LeCun et al. paper*
- *They tried to generalise CNNs beyond the regular-array dataset paradigm*
- *They replaced the translation-invariant kernel structure of CNNs with hierarchical clustering*



https://arxiv.org/abs/1312.6203

A little bit of History

- *The idea of message passing can be tracked to a '15 paper by Duvenaud et al.*
- *The paper introduces "a convolutional neural network that operates directly on graphs"*
- *Language is different, but if you look at the algorithm it is pretty much what we discussed (for specific network architecture choices)*



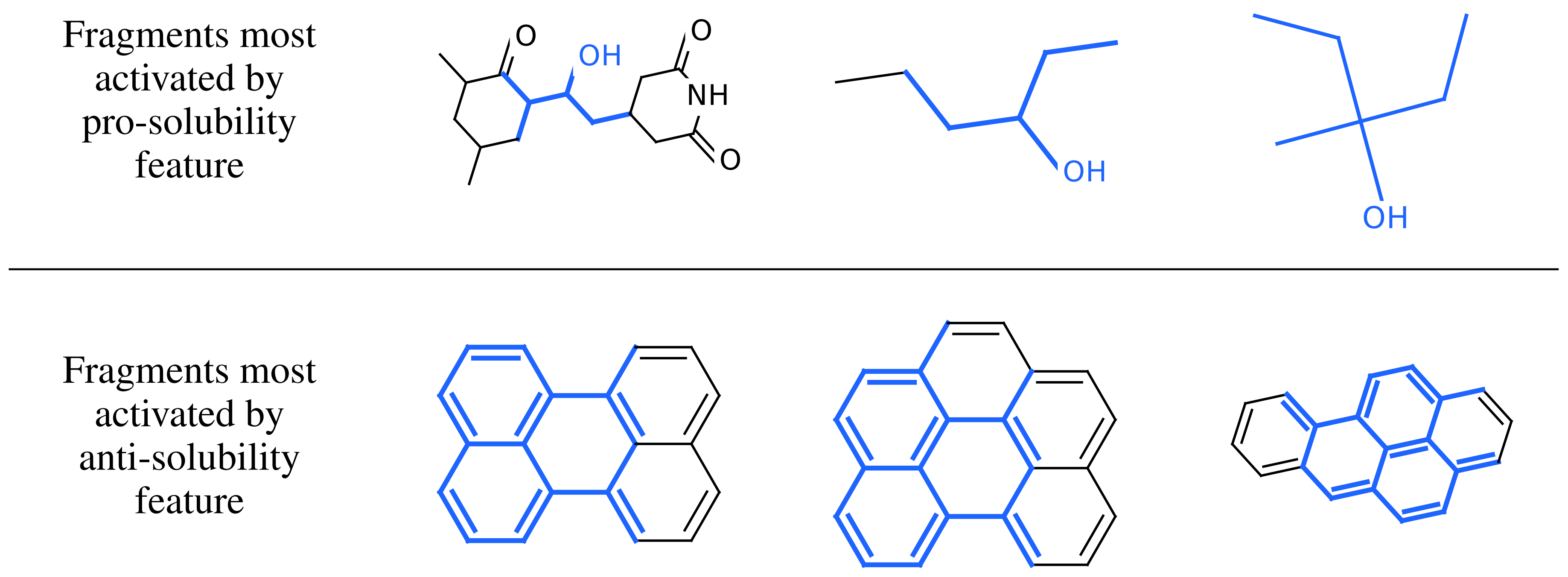


Figure 4: Examining fingerprints optimized for predicting solubility. Shown here are representative examples of molecular fragments (highlighted in blue) which most activate different features of the fingerprint. *Top row:* The feature most predictive of solubility. *Bottom row:* The feature most predictive of insolubility.

https://arxiv.org/pdf/1509.09292.pdf

Algorithm 2 Neural graph fingerprints

Input: molecule, radius R, hidden weights $H_1^1 \dots H_R^5$, output weights $W_1 \dots W_R$
Initialize: fingerprint vector $\mathbf{f} \leftarrow \mathbf{0}_S$
for each atom a in molecule
  $\mathbf{r}_a \leftarrow g(a)$ ▷ lookup atom features
for $L = 1$ to R ▷ for each layer
  for each atom a in molecule
    $\mathbf{r}_1 \dots \mathbf{r}_N = \text{neighbors}(a)$
    $\mathbf{v} \leftarrow \mathbf{r}_a + \sum_{i=1}^{N} \mathbf{r}_i$ ▷ sum
    $\mathbf{r}_a \leftarrow \sigma(\mathbf{v} H_L^N)$ ▷ smooth function
    $\mathbf{i} \leftarrow \text{softmax}(\mathbf{r}_a W_L)$ ▷ sparsify
    $\mathbf{f} \leftarrow \mathbf{f} + \mathbf{i}$ ▷ add to fingerprint
Return: real-valued vector $\mathbf{f}$

Further Reading & Coding

- A few recent reviews that could guide you through the many applications and networks
    - A nice BLOG article on GNNs
    - Another nice BLOG article on GNNs
    - A generic review
    - A particle-physics specific one
- A few GitHub entries
    - JEDI-net Interaction Networks for jet tagging on these data
    - PUPPIML: GGNN for pileup subtraction
    - A small GarNet example that fits an FPGA on these data