

Machine Learning

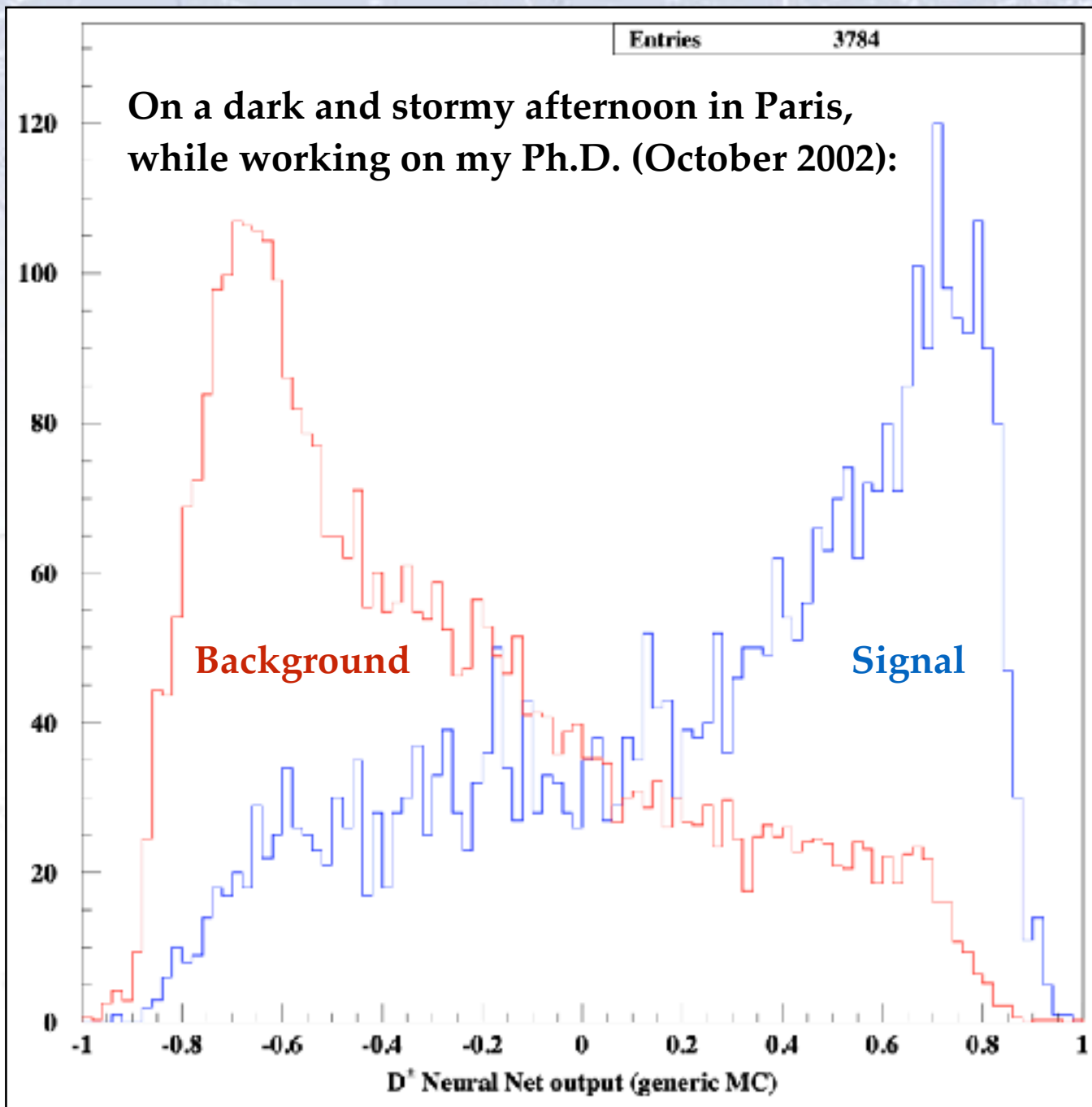
Lecture 2: Unsupervised Learning

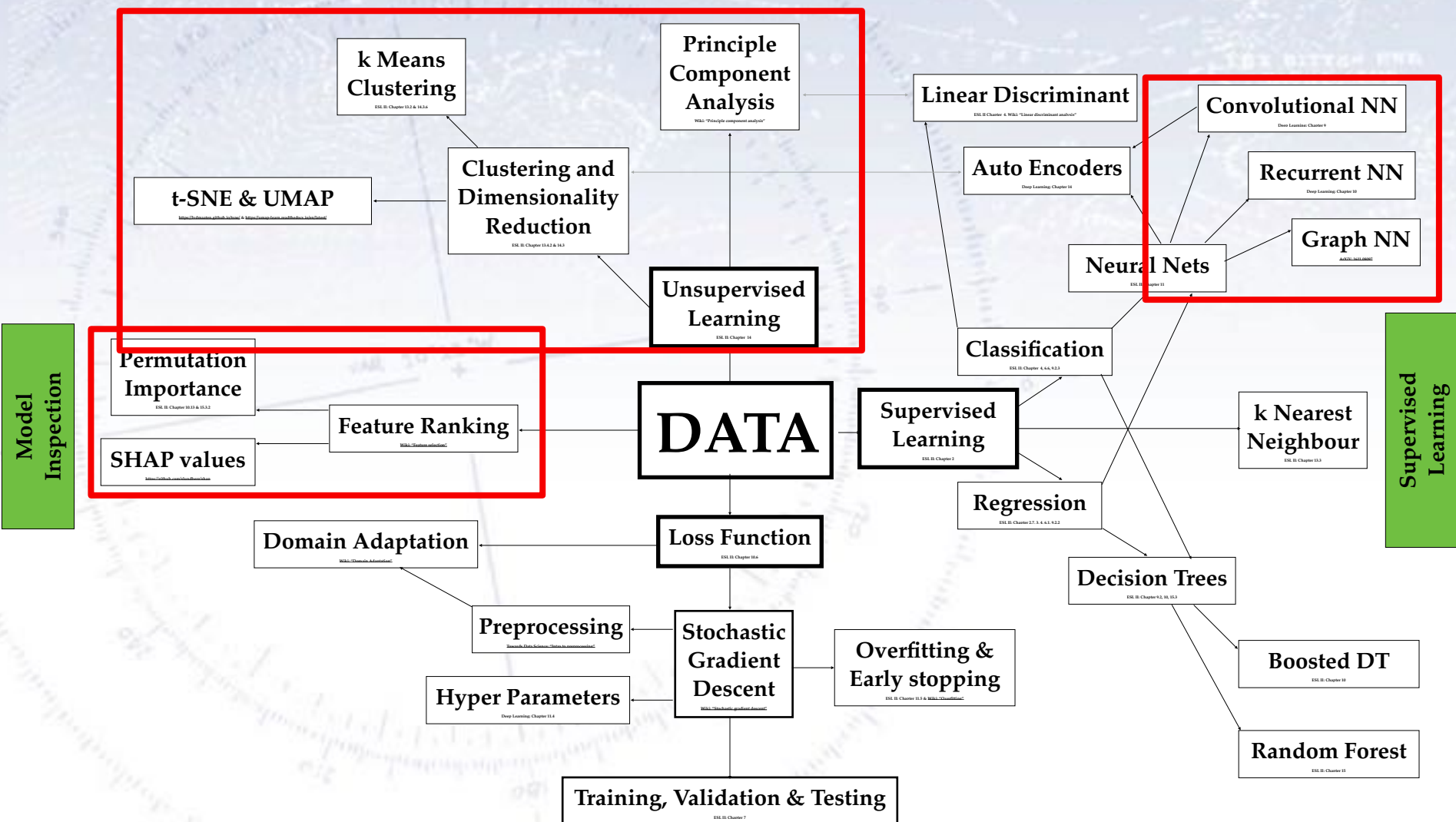


Troels C. Petersen (NBI)



"Statistics is merely a quantisation of common sense - Machine Learning is a sharpening of it!"





Lecture 2: Feature Ranking, Unsupervised Learning, CNNs and GNNs

Outline

Input feature ranking

Unsupervised Learning

- Clustering
- Dimensionality reduction
- AutoEncoders

Why is image data special?

Convolutional Neural Networks (CNN)

- Filter convolutions
- Adding attention to a CNN
- An example analysis with a CNN

What is a graph?

Graph Neural Networks (GNN)

- Motivation for GNNs
- From data to graphs
- Edge convolutions
- Transformers
- An example analysis with GNN

Dreaming...



A faded nautical chart of the Pacific Ocean, showing depth contours and latitude/longitude lines. A red 'X' marks a specific location at 10°15'N, 150°W. The chart includes labels for 'PACIFIC' and '10°15'N' and '150°W'.

Feature Ranking

ML as a science

While Machine Learning is fantastic, it is a black box, and thus unsatisfactory both regarding understanding it, and as a science in itself.

"As a data scientist, I can predict what is likely to happen, but I cannot explain why it is going to happen. I can predict when someone is likely to attrite, or respond to a promotion, or commit fraud, or pick the pink button over the blue button, but I cannot tell you why that's going to happen. And I believe that the inability to explain why something is going to happen is why I struggle to call 'data science' a science."

[Bill Schmarzo, Author of "Big Data: Understanding How Data Powers Big Business"]

However, there are ways of "opening the box", and the most common one is to find out, which input features are important and which are not.

For more info, see:

[Interpretable Machine Learning](#)

A Guide for Making Black Box Models Explainable.

Christoph Molnar

Input Feature Ranking

It is of course useful to know, which of your input features/variables are useful, and which are not. Thus a **ranking of the features** is desired.

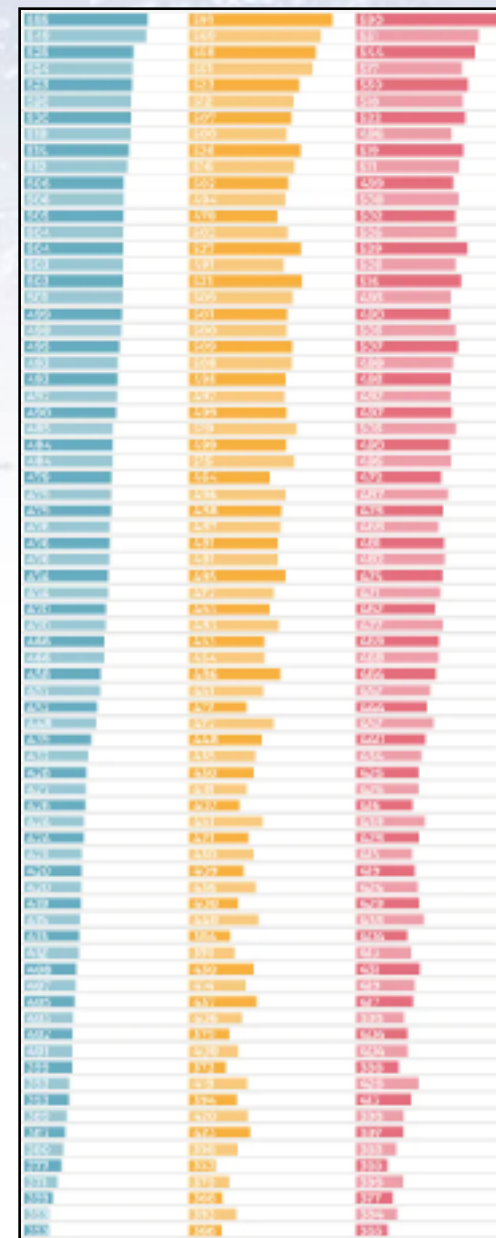
This is not only possible, but actually a general nice feature of ML and feature ranking:

It works as an automation of the detective work behind finding relations.

In principle, one could obtain a variables ranking by testing **all combinations** of variables. But that is not feasible in most situation (N features $> 5-7$)...

Most algorithms have a build-in input feature ranking, which is based on various approximations.

A very simple idea (next slide) that works quite well is **“permutation importance”**.



Input Feature Ranking

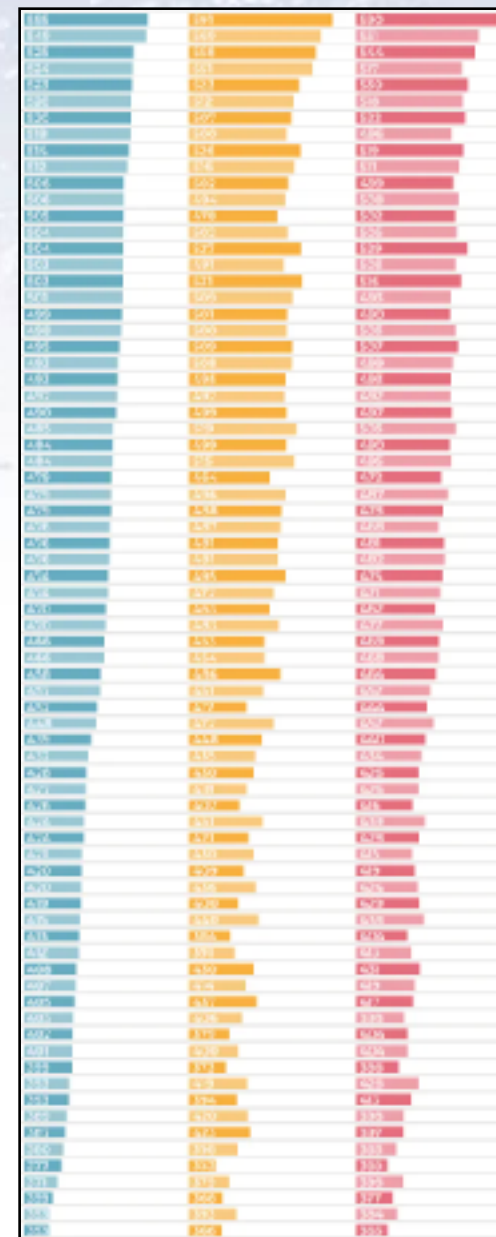
There are many different ways of ranking input features. Three (simple) implementations into XGBoost are:

- **Weight.** The number of times a feature is used to split the data across all trees.
- **Cover.** The number of times a feature is used to split the data across all trees weighted by the number of training data points that go through those splits.
- **Gain.** The average training loss reduction gained when using a feature for splitting.

These have different pro's and con's.

Personally, I much like the idea of...

“permutation invariance”





Permutation Importance

Permutation Importance

One of the most used methods is “permutation importance” (below quoting Christoph M.: ["Interpretable ML" chapter 5.5](#)). The idea is really simple:

We measure the importance of a feature **by calculating the increase in the model's loss function after permuting the feature.**

- A feature is **“important”** if shuffling its values increases the model error, because in this case the model relied on the feature for the prediction.
- A feature is **“unimportant”** if shuffling its values leaves the model error unchanged, because the model thus ignored the feature for the prediction.

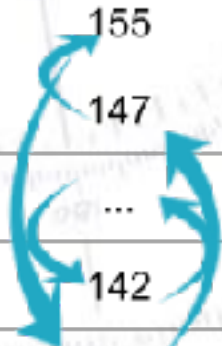
Permutation Importance

One of the most used methods is “permutation importance” (below quoting Christoph M.: ["Interpretable ML" chapter 5.5](#)). The idea is really simple:

We measure the importance of a feature **by calculating the increase in the model’s loss function after permuting the feature.**

- A feature is **“important”** if shuffling its values increases the model error, because in this case the model relied on the feature for the prediction.
- A feature is **“unimportant”** if shuffling its values leaves the model error unchanged, because the model thus ignored the feature for the prediction.

Height at age 20 (cm)	Height at age 10 (cm)	...	Socks owned at age 10
182	155	...	20
175	147	...	10
...
156	142	...	8
153	130	...	24



Permutation Importance

Input: Trained model f , feature matrix X , target vector y , loss function $L(y, f)$.

[Fisher, Rudin, and Dominici (2018)]

- Estimate the original model error $e_{\text{orig}} = L(y, f(X))$
- For each feature $j = 1, \dots, p$ do:
 - Generate feature matrix $X_{\text{perm},j}$ by permuting feature j in the data X .
This breaks the association between feature j and true outcome y .
 - Estimate error $e_{\text{perm},j} = L(Y, f(X_{\text{perm},j}))$ based on the predictions of $X_{\text{perm},j}$.
 - Calculate permutation feature importance $FI_j = e_{\text{perm},j} / e_{\text{orig}}$ (or $e_{\text{perm},j} - e_{\text{orig}}$).
- Sort features by descending FI_j .

X_A	X_B	X_C	Y
x_{a1}	x_{b1}	x_{c1}	y_1
x_{a2}	x_{b2}	x_{c2}	y_2
x_{a3}	x_{b3}	x_{c3}	y_3
x_{a4}	x_{b4}	x_{c4}	y_4
x_{a5}	x_{b5}	x_{c5}	y_5
x_{a6}	x_{b6}	x_{c6}	y_6

Note: Permutation Importance calculations are computationally fast. (why?)

Feature importance with Neural Networks (Towards Data Science)



SHAP Values

The background is a light blue map of the North Atlantic Ocean. A red dot is located in the central part of the map, with a white circle around it. The map shows latitude and longitude lines, with labels like '50°N', '60°N', '70°N', '80°N', '90°N', '10°W', '20°W', '30°W', '40°W', '50°W', '60°W', '70°W', '80°W', '90°W', '100°W', '110°W', '120°W', '130°W', '140°W', '150°W', '160°W', '170°W', '180°W', '190°W', '200°W', '210°W', '220°W', '230°W', '240°W', '250°W', '260°W', '270°W', '280°W', '290°W', '300°W', '310°W', '320°W', '330°W', '340°W', '350°W', '360°W'. There are also some text labels on the map, such as 'WAG-1470', 'WAG 50°15'N', and '18° BITTEN ERN TACHT 17475'.

SHAP Values

SHAP is a technique for deconstructing a machine learning model's predictions into a sum of contributions from each of its input variables.

The result is an evaluation of the input variables for each single case!

Shapley values

Shapley values is a concept from **corporative game theory**, where they are used to provide a possible answer to the question:

“How important is each player to the overall cooperation, and what payoff can each player reasonably expect?”

The Shapley values are considered “fair”, as they are the only distribution with the following properties:

- **Efficiency:** Sum of Shapley values of all agents equals value of grand coalition.
- **Linearity:** If two coalition games described by v and w are combined, then the distributed gains should correspond to the gains derived from the sum of v and w .
- **Null player:** The Shapley value of a null player is zero.
- **Stand alone test:** If v is sub/super additive, then $\phi_i(v) \leq / \geq v(\{i\})$, where ϕ is the Shapley value for agent i , and v is the worth function (of a coalition). Also called “Monotonicity”: A consistently more contributing feature much a get higher v .
- **Anonymity:** Labelling of agents doesn't play a role in assignment of their gains.
- **Marginalism:** Function uses only marginal contributions of player i as arguments.

From such values, one can determine which variables contribute to a final result. And summing the values, one can get an overall idea of which variables are important.

Shapley value calculation

Consider a set \mathbf{N} (of n players) and a (characteristic or worth) function v that maps any subset of players to real numbers:

$$v : 2^{\mathbf{N}} \rightarrow \mathbb{R}, \quad v(\emptyset) = 0$$

If S is a coalition of players, then $v(S)$ yields the total expected sum of payoffs the members of S can obtain by cooperation.

The Shapley values are calculated as:

$$\varphi_i(v) = \sum_{S \subseteq \mathbf{N} \setminus \{i\}} \frac{|S|!(n - |S| - 1)!}{n!} [v(S \cup \{i\}) - v(S)]$$

To formula can be understood, if we imagine a coalition being formed one actor at a time, with each actor demanding their contribution $v(S \cup \{i\}) - v(S)$ as a fair compensation, and then for each actor take the average of this contribution over the possible different permutations in which the coalition can be formed.

Shapley value calculation

Consider a set \mathbf{N} (of n players) and a (characteristic or worth) function v that maps any subset of players to real numbers:

$$v : 2^{\mathbf{N}} \rightarrow \mathbb{R}, \quad v(\emptyset) = 0$$

If S is a coalition of players, then $v(S)$ yields the total expected sum of payoffs the members of S can obtain by cooperation.

The Shapley values can also be calculated as:

$$\varphi_i(v) = \frac{1}{n!} \sum_R [v(P_i^R \cup \{i\}) - v(P_i^R)]$$

where the sum ranges over all $n!$ orders R of the players and P_i^R is the set of players in \mathbf{N} which precede i in the order R . This has the interpretation:

$$\varphi_i(v) = \frac{1}{N_{\text{players}}} \sum_{C \setminus i} \frac{\text{marginal contribution of } i \text{ to coalition } C}{\text{number of coalitions excluding } i \text{ of this size}}$$

Shapley value calculation example

Example 1:

Two friends (F1 and F2) make a business: **Payoff 600\$** (i.e. $v(F1, F2) = 600$).

If F1 or F2 did not participate, payoff would be 0\$ (i.e. $v(F1) = v(F2) = 0$).

Result: F1 and F2 each gets 300\$.

Example 2:

Two friends (F1 and F2) make a business: **Payoff 600\$** (i.e. $v(F1, F2) = 600$).

If F1 did not participate, payoff would be 0\$ (i.e. $v(F1) = 0$).

If F2 did not participate, payoff would be 200\$ (i.e. $v(F2) = 200$).

Cases:

F1 1st gets 0\$. With F2 also they get 600\$. F2's marginal contribution: 600\$.

F2 1st gets 200\$. With F1 also they get 600\$. F1's marginal contribution: 400\$.

Result:

F1 should have: $0.5 \times 0\$ + 0.5 \times 400\$ = 200\$$

F2 should have: $0.5 \times 200\$ + 0.5 \times 600\$ = 400\$$

Note that the number of cases quickly expands!

SHAP Values

A great approximation was developed by Scott Lundberg with **SHAP values**:

SHAP (SHapley Additive exPlanations):

<https://github.com/slundberg/shap>

This algorithm provides - **for each entry** - a ranking of the input variables, i.e. a sort of explanation for the result.

One can also sum of the SHAP values over all entries, and then get the overall ranking of feature variables. **They are based on Shapley values.**

Note: SHAP values are computationally “heavy”.

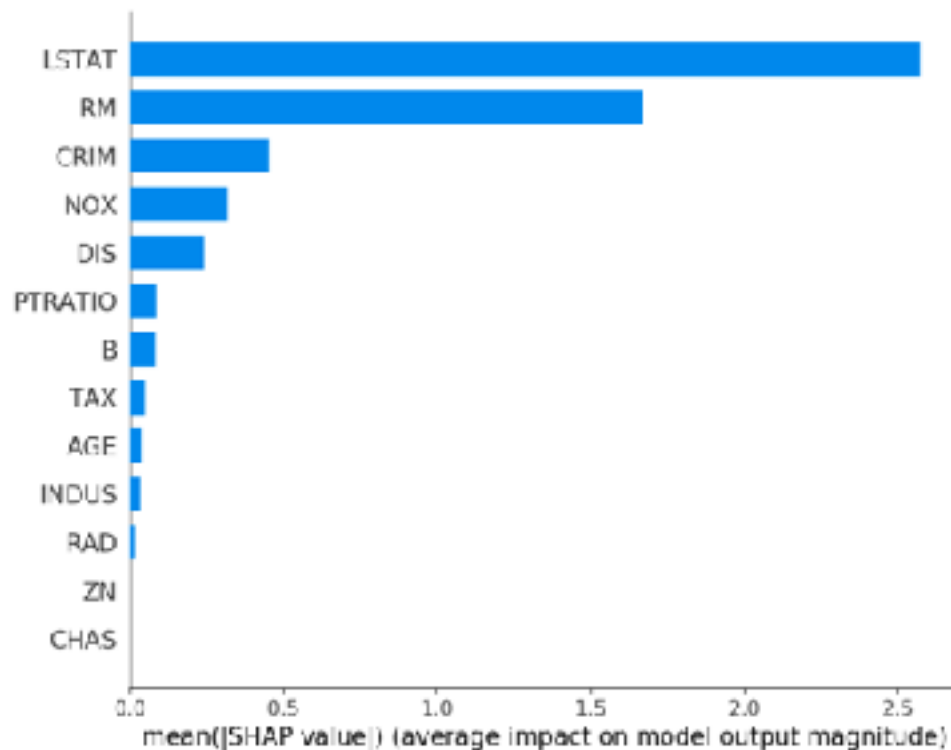
Note2: Lately, this seems to have been improved in 2023!

Input Feature Ranking

Here is an example from SHAP's [github](#) site.

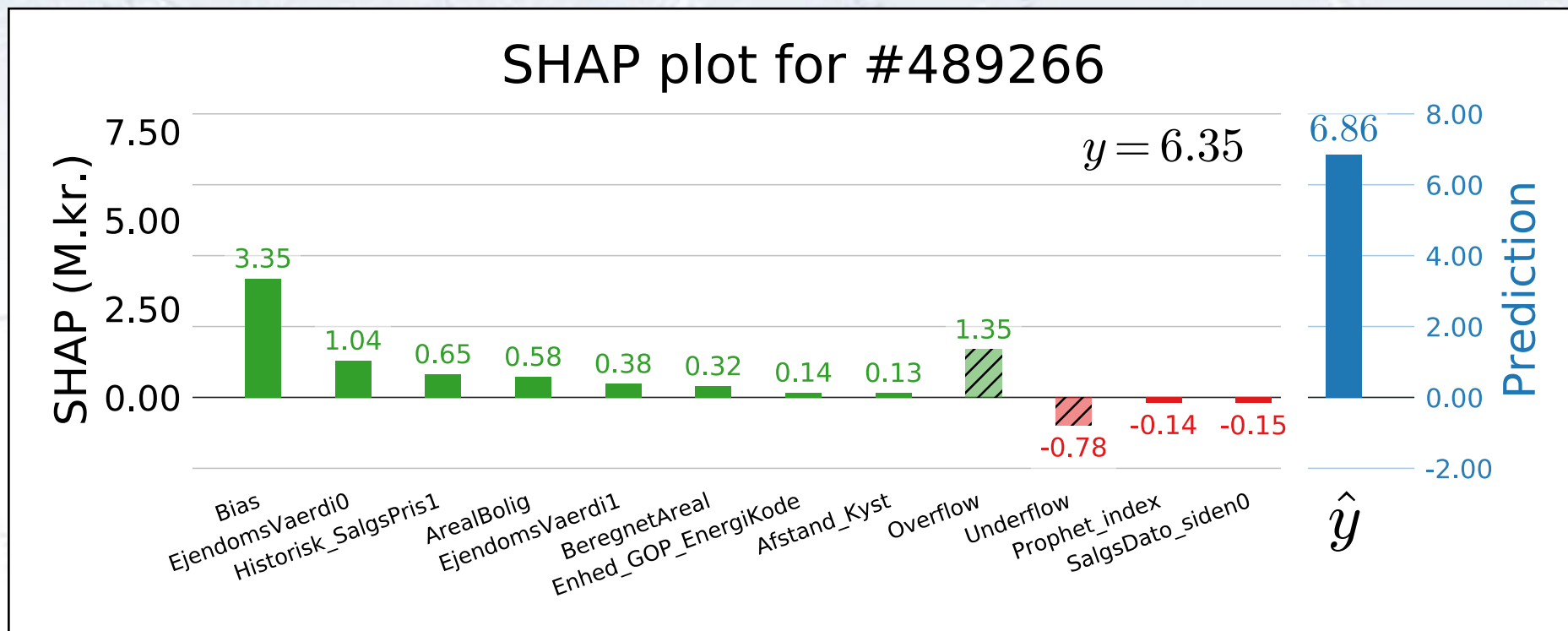
Clearly, LSTAT and RM are the best variables (whatever they are!).

```
shap.summary_plot(shap_values, X, plot_type="bar")
```



Individuel estimates

Shapley-values also gives the possibility to see the reason behind **individuel estimates**. Below is an example, illustrating this point.



Above is shown which factors that influences the final estimate of the sales price (and how much). The estimate is the sum of the contributions (here 6.86 MKr.).

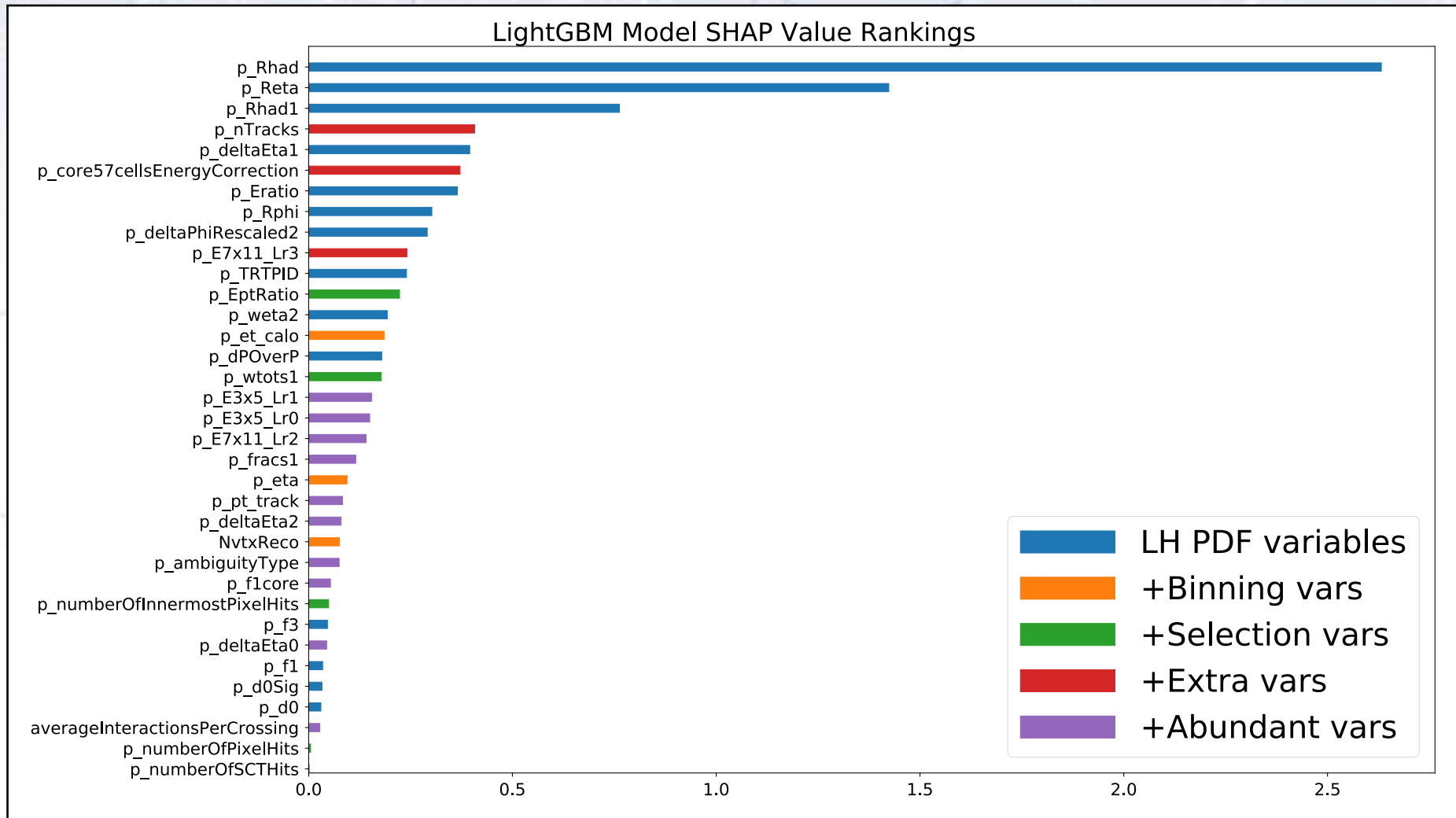
This is a fantastic tool to get insight into the ML workings!!!



Example of usage

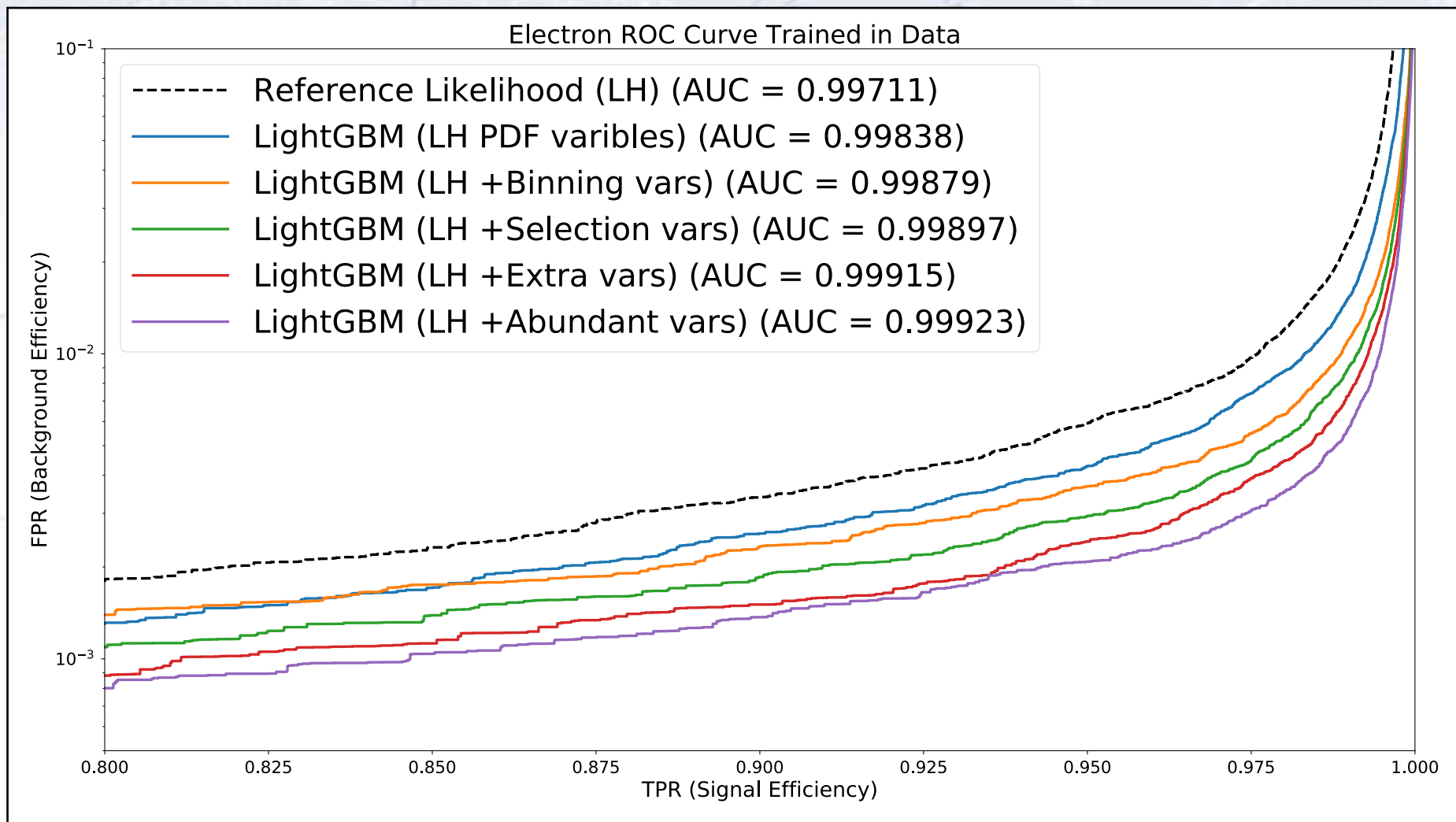
Input Feature Ranking

Here is an example from particle physics. The blue variables were “known”, but with SHAP we discovered three new quite good variables in data.



Input Feature Ranking

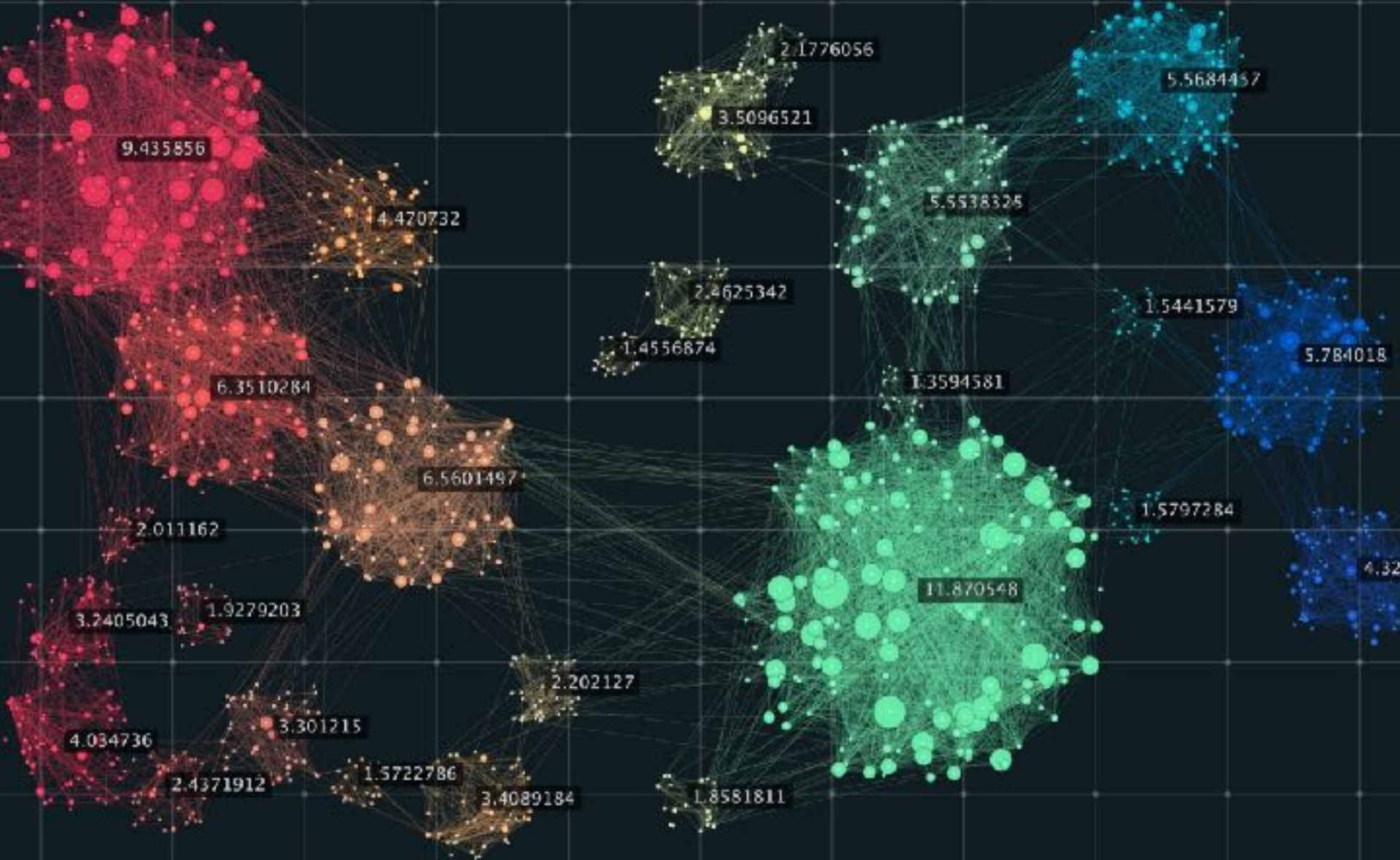
We could of course just add all variables, but want to stay simple, and training the models, we see that the three extra variables gives most of gain.



The background is a detailed nautical chart of the North Atlantic Ocean. It features depth contours in fathoms, with labels such as 100, 150, 200, 250, 300, and 350. The chart also shows latitude and longitude lines, with latitude marked from 30 to 60 North and longitude from 100 to 150 West. Various geographical features and names are visible, including 'WACHSME' and '187 BITTEN ERN TAUCHT 1879'. The overall tone is light blue and white, typical of a faded map.

Unsupervised Learning: Clustering

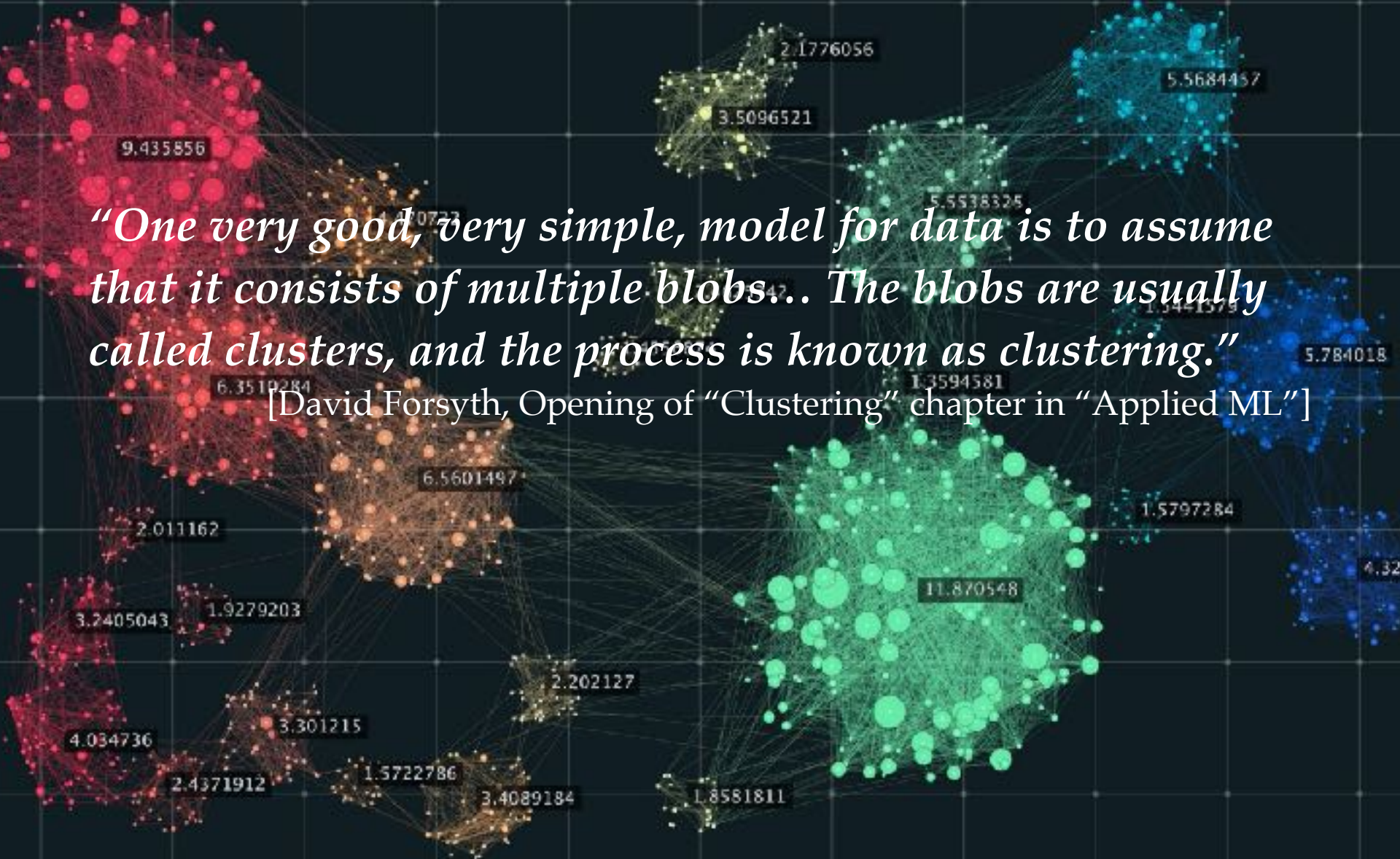
Clustering... is an art!



Clustering... is an art!

"One very good, very simple, model for data is to assume that it consists of multiple blobs... The blobs are usually called clusters, and the process is known as clustering."

[David Forsyth, Opening of "Clustering" chapter in "Applied ML"]



Evaluating clustering

Evaluation of identified clusters is subjective and may require a domain expert, although many clustering-specific quantitative measures do exist.

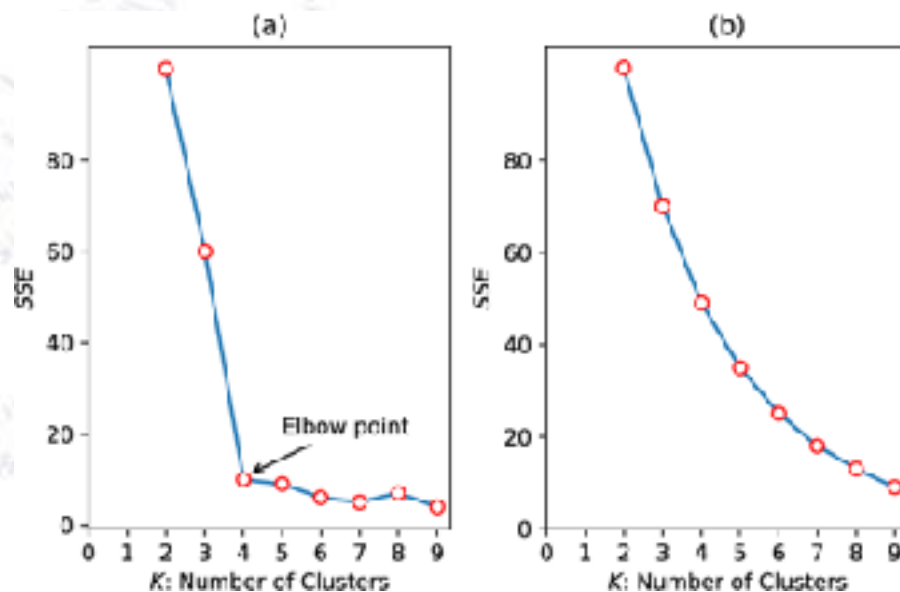
Typically, clustering algorithms are compared on synthetic datasets with pre-defined clusters, which an algorithm is expected to discover.

“Clustering is an unsupervised learning technique, so it is hard to evaluate the quality of the output of any given method.”

[Page 534, Machine Learning: A Probabilistic Perspective, 2012.]

One of the simple principles is that of the “Elbow Method”.

If the loss function shows an “elbow” (sudden stop in rate of improvement), then that probably reflects some structure in the data.



Evaluating clustering

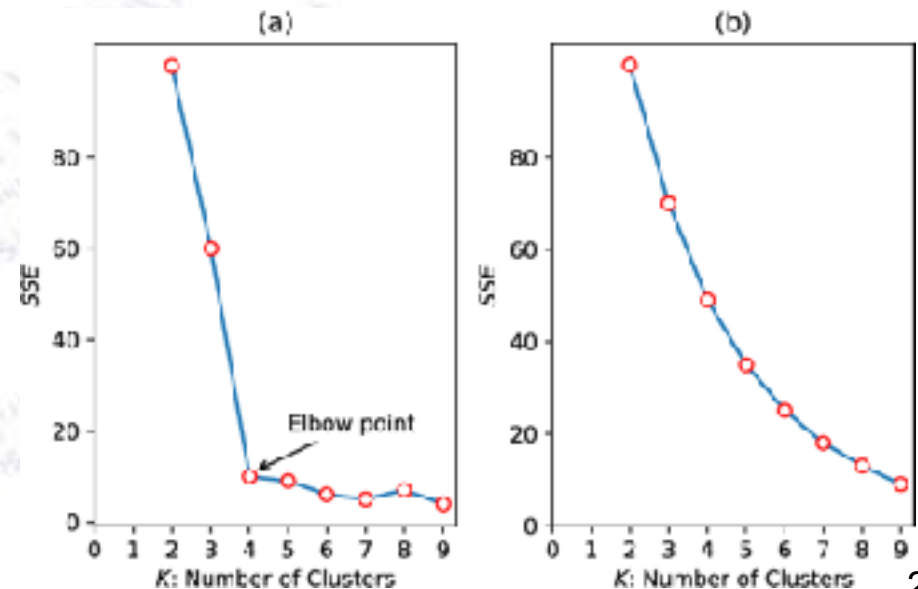
Evaluation of identified clusters is subjective and may require a domain expert, although many clustering-specific quantitative measures do exist.

Typically, clustering algorithms are compared on synthetic datasets with pre-defined clusters, which an algorithm is expected to discover.

One way of visually evaluating a clustering algorithm is to combine it with a dimensionality reduction, though one then observes the combined performance of the two.

One of the simple principles is that of the “Elbow Method”.

If the loss function shows an “elbow” (sudden stop in rate of improvement), then that probably reflects some structure in the data.

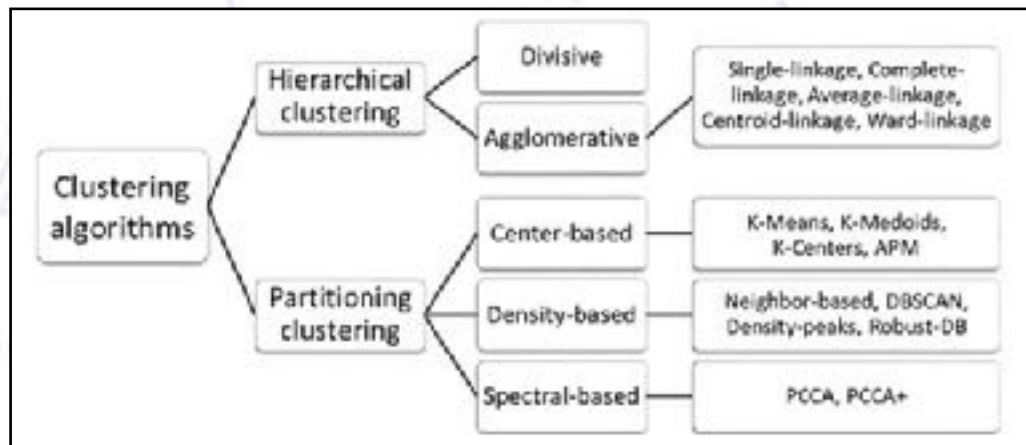


Clustering algorithms

Clustering is an “old field” and many philosophies (and algorithms) have been developed. They can roughly be reduced to two approaches:

- **Hierarchical clustering** algorithms are based on recursively either merging smaller clusters in to larger ones or dividing larger clusters to smaller ones.
- **Partitioning clustering** algorithms generate various partitions and then iteratively place each instance best in one of k mutually exclusive clusters.

Hierarchical clustering does not require any input parameters, while partitioning clustering algorithms require the number of clusters to start running. Hierarchical clustering returns a much more meaningful and subjective division of clusters but partitioning clustering results in exactly k clusters.

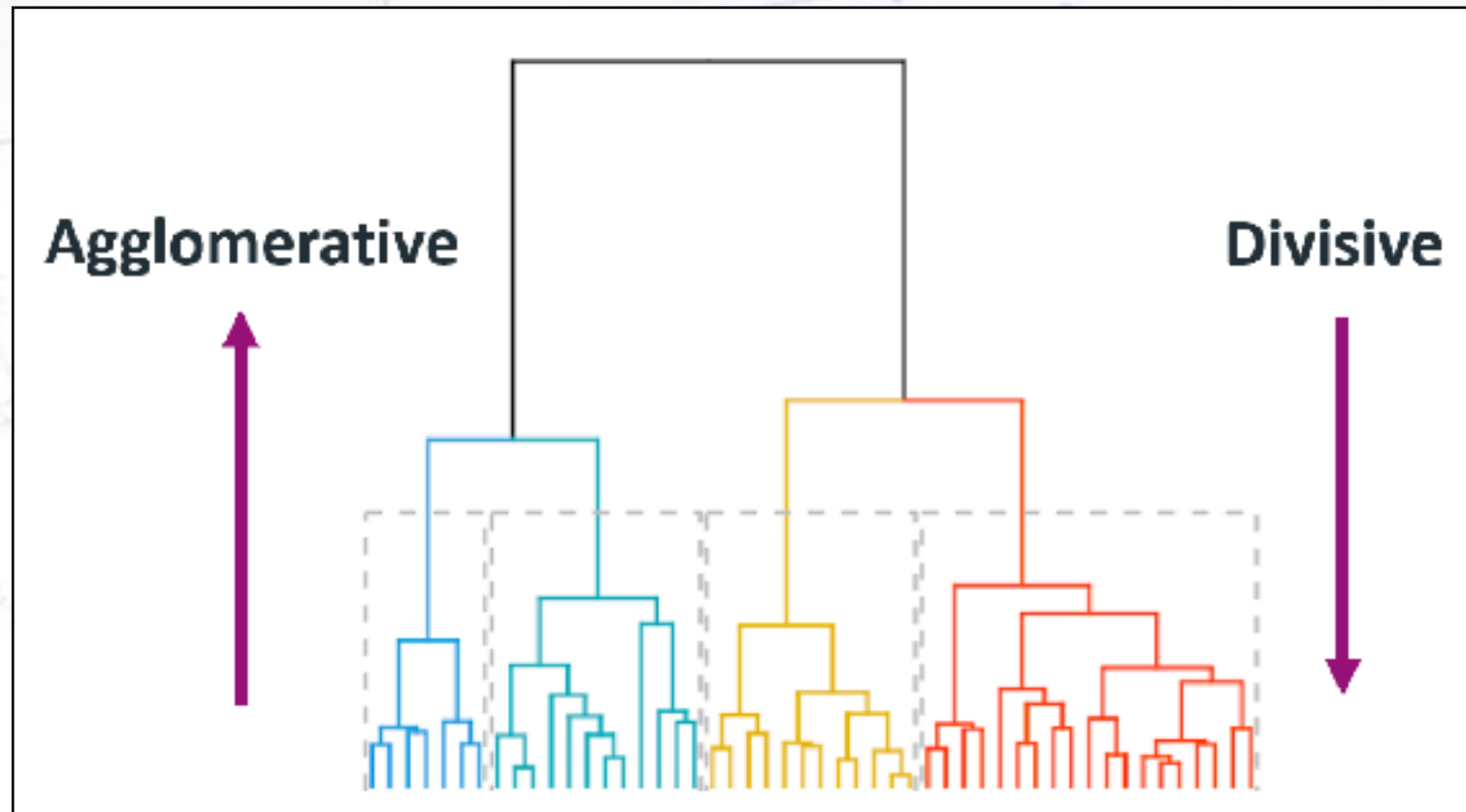


Hierarchical clustering algorithms

Hierarchical clustering algorithms can be further divided:

- **Agglomerative:** Merge smaller clusters into larger ones
- **Divisive:** Divide larger clusters into smaller ones.

The only requirement is a **similarity measure** to decide distance between cases.



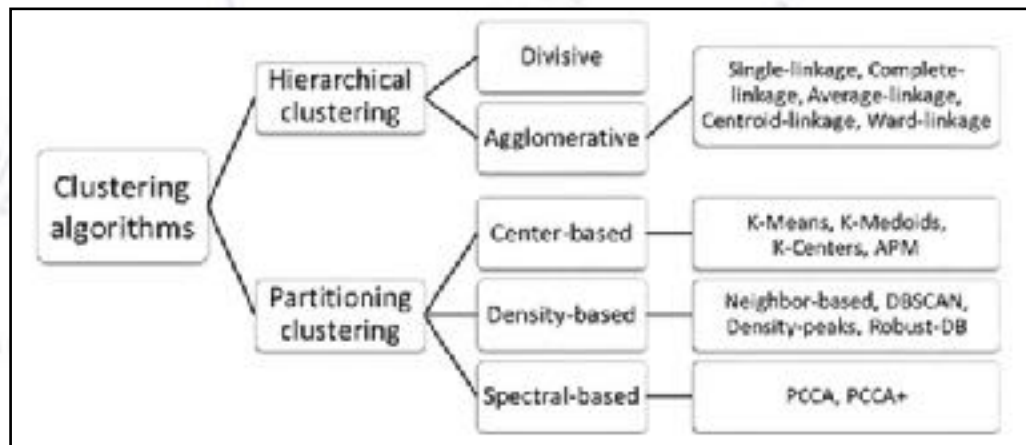
Partitioning clustering algorithms

Partitioning clustering algorithms can (also) be further divided:

- **Center-based:** Build clusters around (random?) centers (**k-Means**).
- **Density-based:** Build clusters around (high) densities (**DBSCAN**).
- **Spectral-based:** Uses eigenvalues of the similarity matrix to perform dimensionality reduction before clustering (**PCCA+**).

“k-Means clustering is the “go-to” clustering algorithm. You should see it as a basic recipe from which many algorithms can be concocted.”

[David Forsyth, “Applied ML” chapter 8.2.6]



k-Means clustering

The recipe is to iterate the below points, until movements are “small”:

- Allocate each data point to the closest cluster center
- Re-estimate cluster centers from their data points.

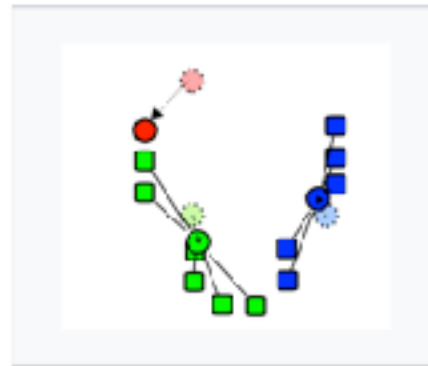
Demonstration of the standard algorithm



1. k initial "means" (in this case $k=3$) are randomly generated within the data domain (shown in color).



2. k clusters are created by associating every observation with the nearest mean. The partitions here represent the **Voronoi diagram** generated by the means.



3. The **centroid** of each of the k clusters becomes the new mean.



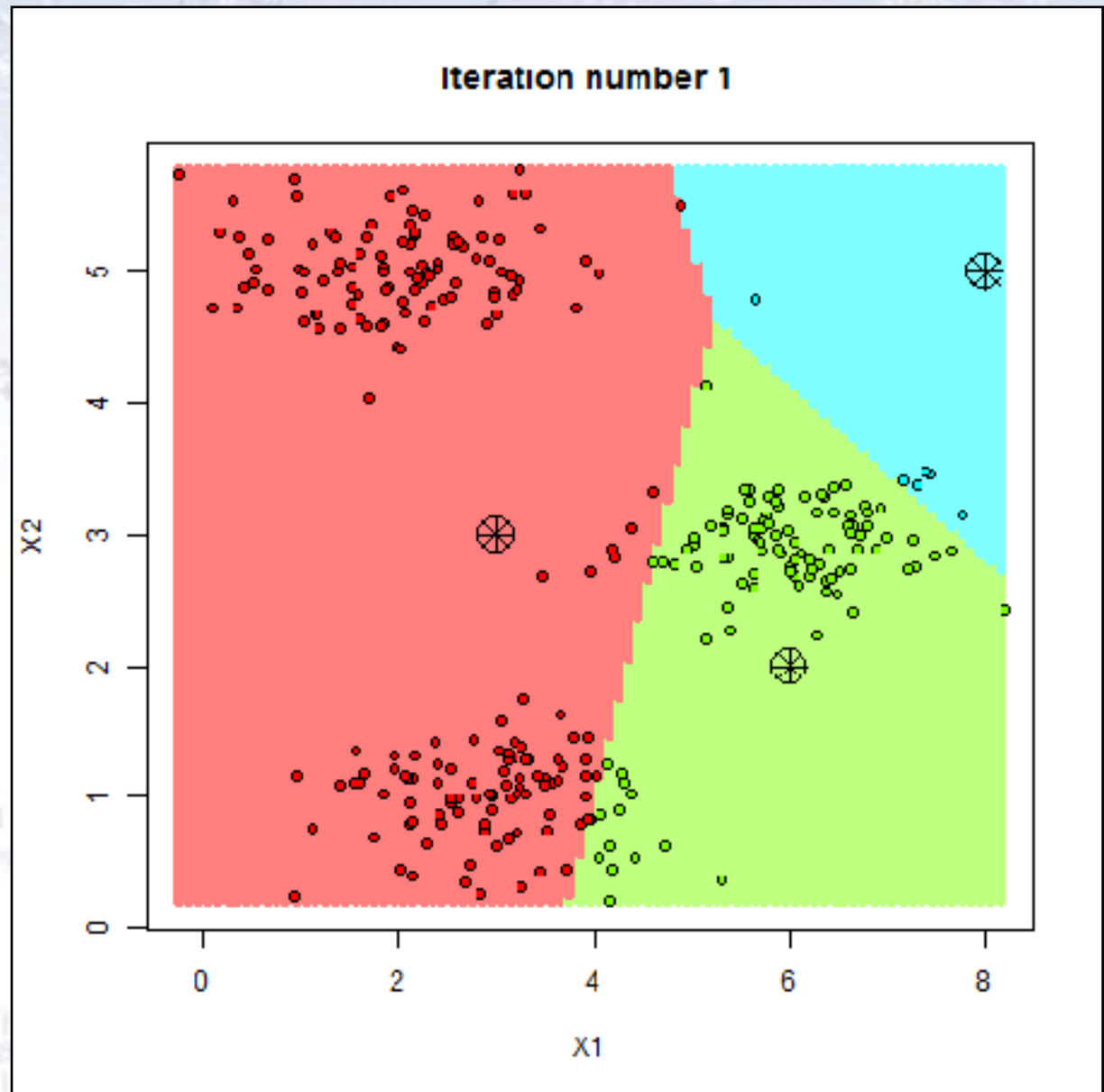
4. Steps 2 and 3 are repeated until convergence has been reached.

There are many variations, improvements, etc. that refine this algorithm. Most notably are the k-means++ (better initial points) and k-medoids methods.

k-Means clustering

The recipe:

- Allocate each data point to the closest cluster center
- Re-estimate cluster centers from their data points.



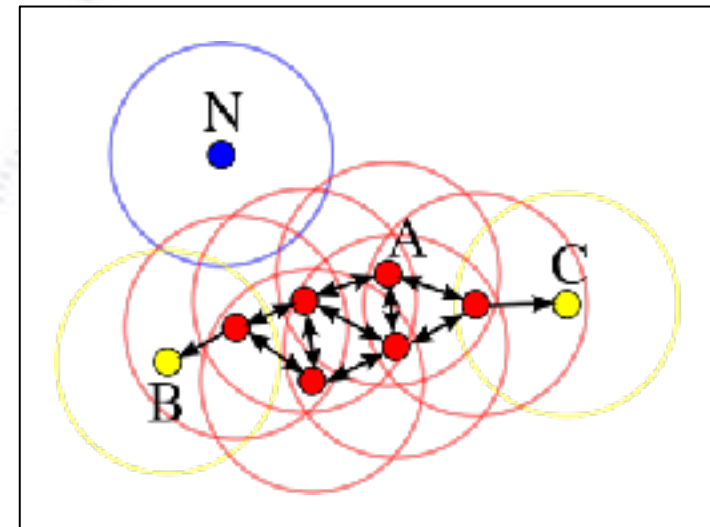
DBSCAN algorithm

DBSCAN classifies points as core points, reachable points and outliers:

- A point p is a **core point** if at least minPts points are within distance ε of it.
- A point q is **directly reachable** from p if point q is within distance ε from core point p . Points are only said to be directly reachable from core points.
- A point q is **reachable** from p if there is a path p_1, \dots, p_n with $p_1 = p$ and $p_n = q$, where each p_{i+1} is directly reachable from p_i . Note that this implies that the initial point and all points on the path must be core points, with the possible exception of q .
- All points not reachable from any other point are outliers or noise points.

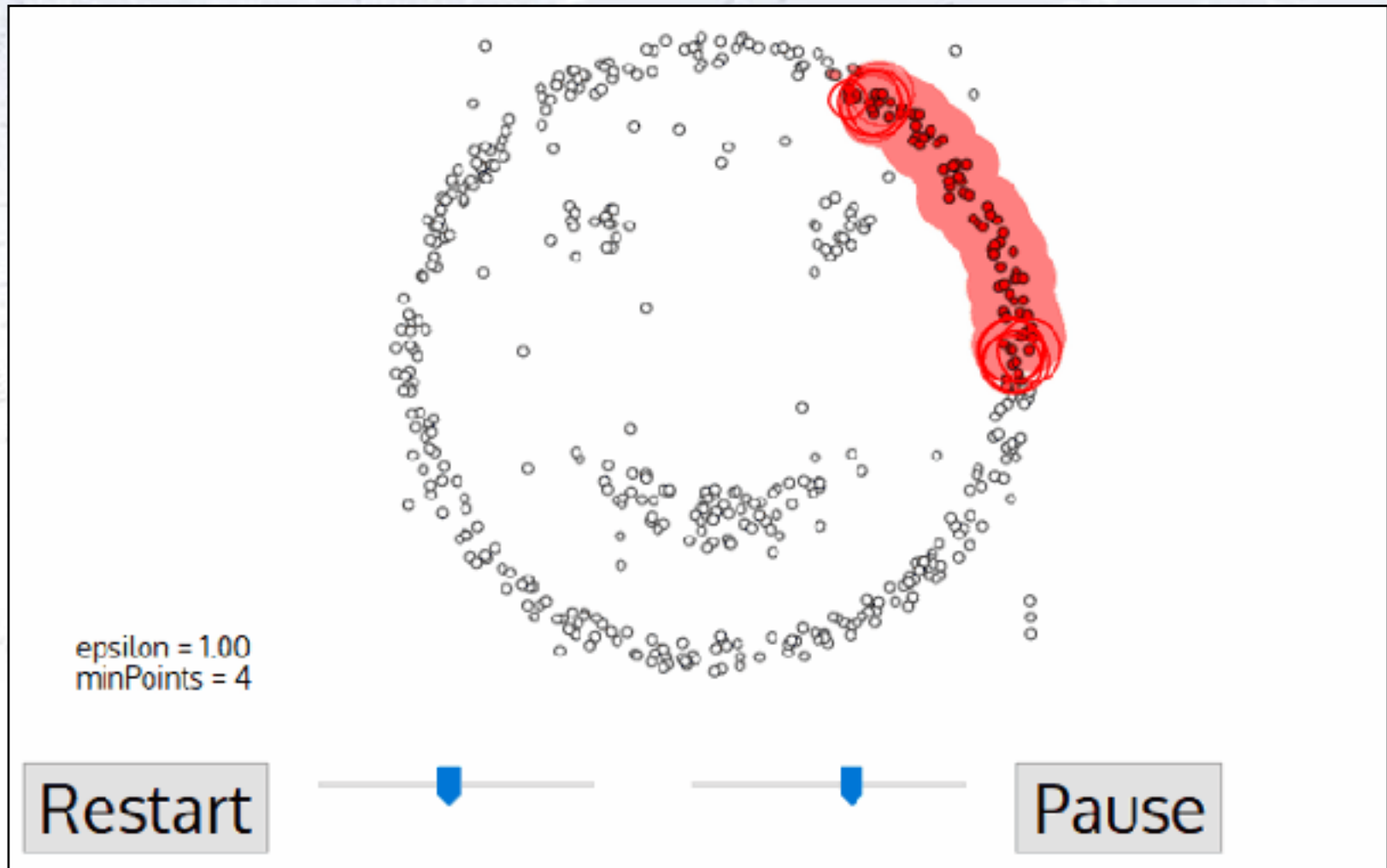
DBSCAN has two parameters: minPts and ε .

If p is a core point, then it forms a cluster together with all points (core or non-core) that are reachable from it. Each cluster contains at least one core point; non-core points can be part of a cluster, but they form its “edge”, since they cannot be used to reach more points.



DBSCAN algorithm

As can be seen, DBSCAN is a rather generic algorithm, capable of handling a large variety of data.

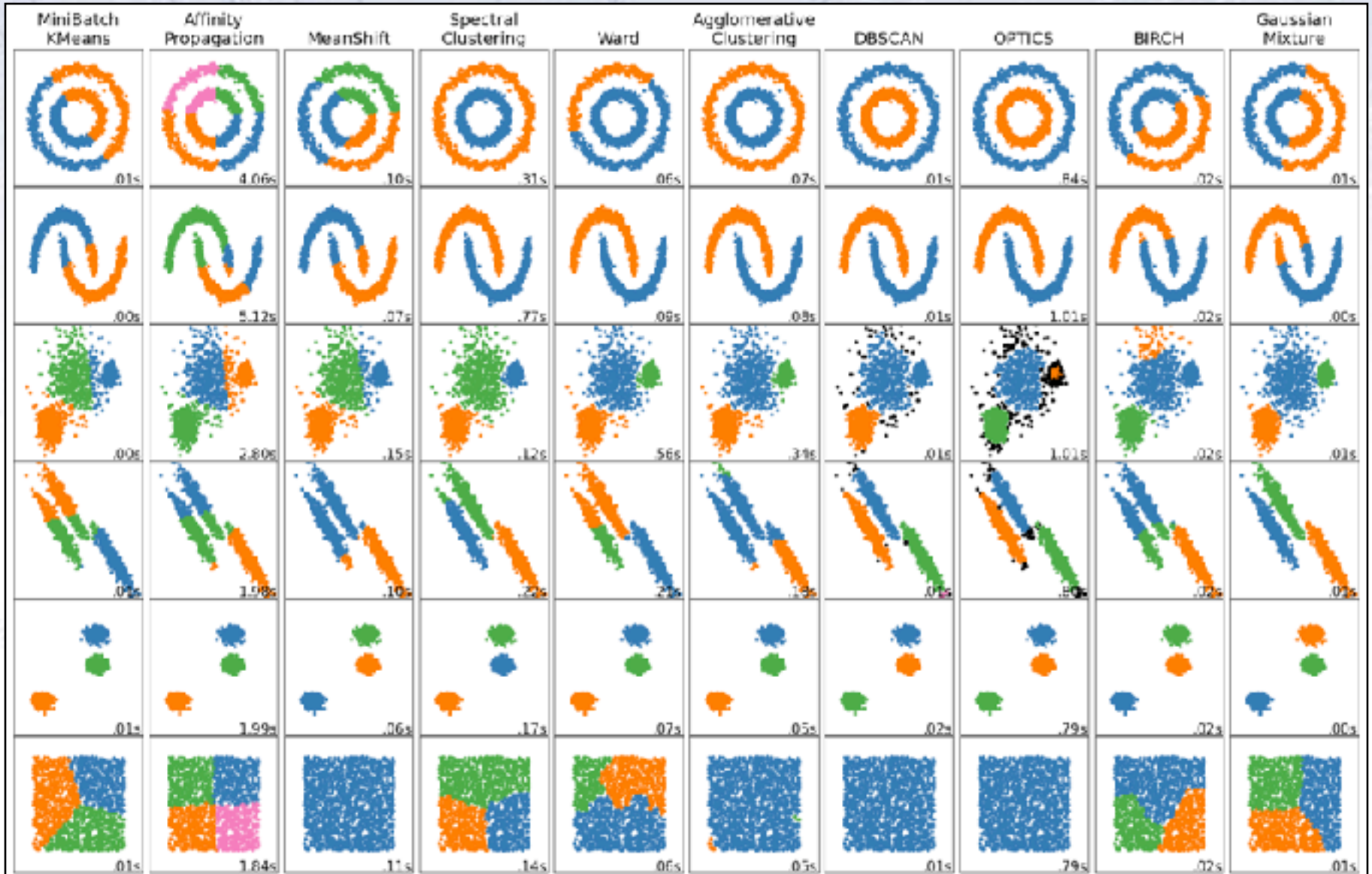


Clustering algorithms in scikit-learn

Scikit-Learn has a rather good selection of clustering algorithms:

Method name	Parameters	Scalability	Usecase	Geometry (metric used)
K-Means	number of clusters	Very large <code>n_samples</code> , medium <code>n_clusters</code> with MiniBatch code	General-purpose, even cluster size, flat geometry, not too many clusters, inductive	Distances between points
Affinity propagation	damping, sample preference	Not scalable with <code>n_samples</code>	Many clusters, uneven cluster size, non-flat geometry, inductive	Graph distance (e.g. nearest-neighbor graph)
Mean-shift	bandwidth	Not scalable with <code>n_samples</code>	Many clusters, uneven cluster size, non-flat geometry, inductive	Distances between points
Spectral clustering	number of clusters	Medium <code>n_samples</code> , small <code>n_clusters</code>	Few clusters, even cluster size, non-flat geometry, transductive	Graph distance (e.g. nearest-neighbor graph)
Ward hierarchical clustering	number of clusters or distance threshold	Large <code>n_samples</code> and <code>n_clusters</code>	Many clusters, possibly connectivity constraints, transductive	Distances between points
Agglomerative clustering	number of clusters or distance threshold, linkage type, distance	Large <code>n_samples</code> and <code>n_clusters</code>	Many clusters, possibly connectivity constraints, non Euclidean distances, transductive	Any pairwise distance
DBSCAN	neighborhood size	Very large <code>n_samples</code> , medium <code>n_clusters</code>	Non-flat geometry, uneven cluster sizes, outlier removal, transductive	Distances between nearest points
OPTICS	minimum cluster membership	Very large <code>n_samples</code> , large <code>n_clusters</code>	Non-flat geometry, uneven cluster sizes, variable cluster density, outlier removal, transductive	Distances between points
Gaussian mixtures	many	Not scalable	Flat geometry, good for density estimation, inductive	Mahalanobis distances to centers
BIRCH	branching factor, threshold, optional global clusterer.	Large <code>n_clusters</code> and <code>n_samples</code>	Large dataset, outlier removal, data reduction, inductive	Euclidean distance between points

Clustering algorithms in scikit-learn



A comparison of the clustering algorithms in scikit-learn

Conclusions

Clustering is an “old” art form, for which there is a vast ocean of methods.

The K-means (and further developments) is the standard algorithm, if there is one such. DBSCAN is also an old (and awarded!) classic.

Note that like in dimensionality reduction, it is important to transform the input variables first, so that mean and variances are of order zero and unity.

It is HARD to evaluate the performance, and visual inspection and testing on similar (typically simulated) cases are some of few methods.

The background is a nautical chart of the North Atlantic Ocean. It features a grid of latitude and longitude lines. A prominent red circle is drawn on the chart, centered around 40°N latitude and 15°W longitude. The chart includes various navigational symbols, soundings, and text labels such as 'WACHS' and '187 BITTEN ERN TACHT 1879'.

Example Use Cases

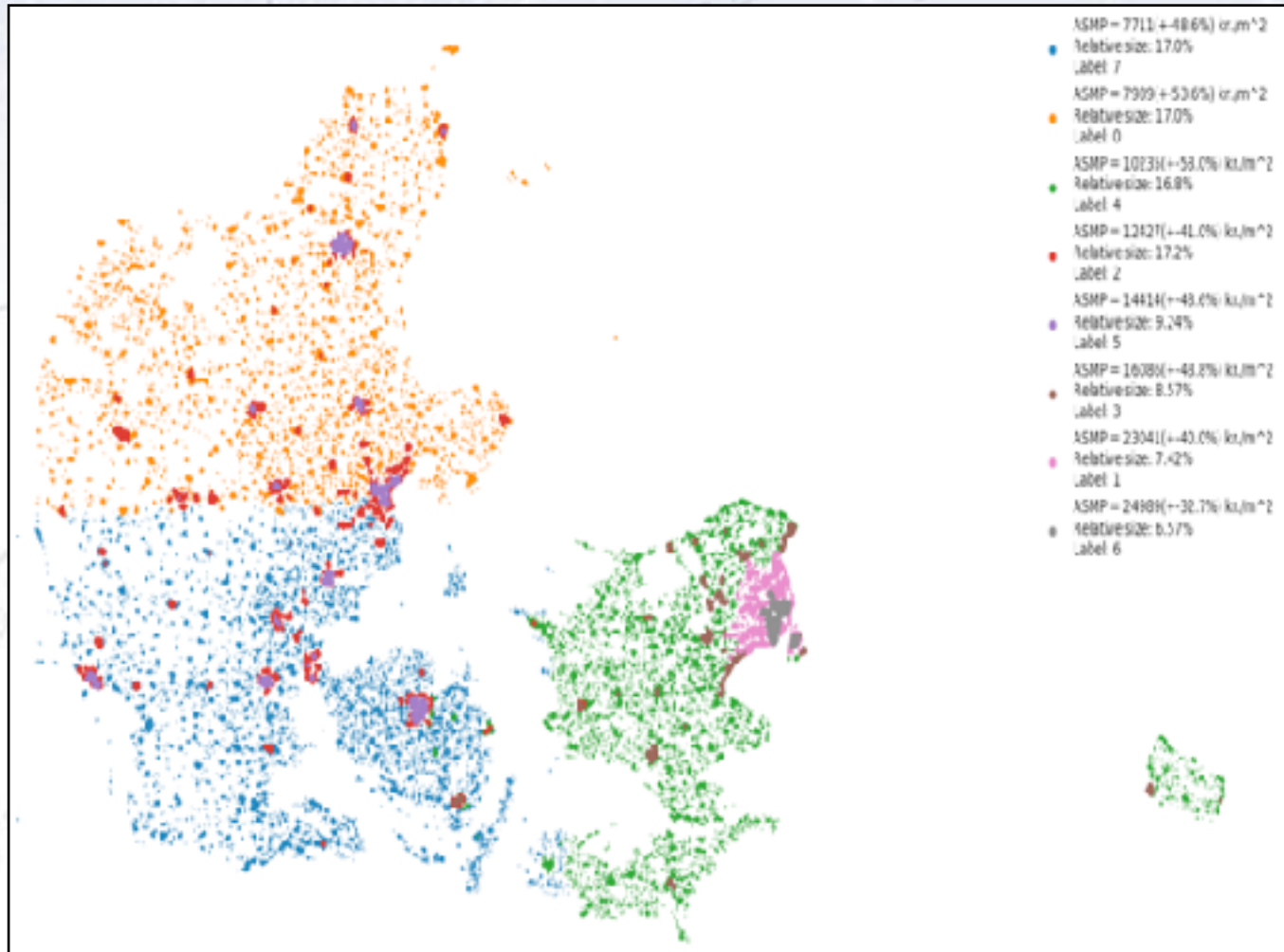
OK - what is it good for?

Clustering is used for several things:

- Market segmentation:
Dividing costumers or products into similar classes is used in advertising.
- Production quality assurance:
Clustering of images is used for detecting faulty productions automatically.
- DNA analysis:
The ability to cluster very high dimensional data (DNA) to groups / families.
- Medical imaging:
Classification of medical images without labels.
- Image segmentation:
Dividing an image into its parts is used in e.g. self-driving and security.
- Anomaly detection:
Quick detection of e.g. credit card fraud saves large amounts of money.

Clustering of Danish housing

Show is a simple clustering of the Danish housing market, based on position (x,y) and price/m². In this way, one can see developments for each market.



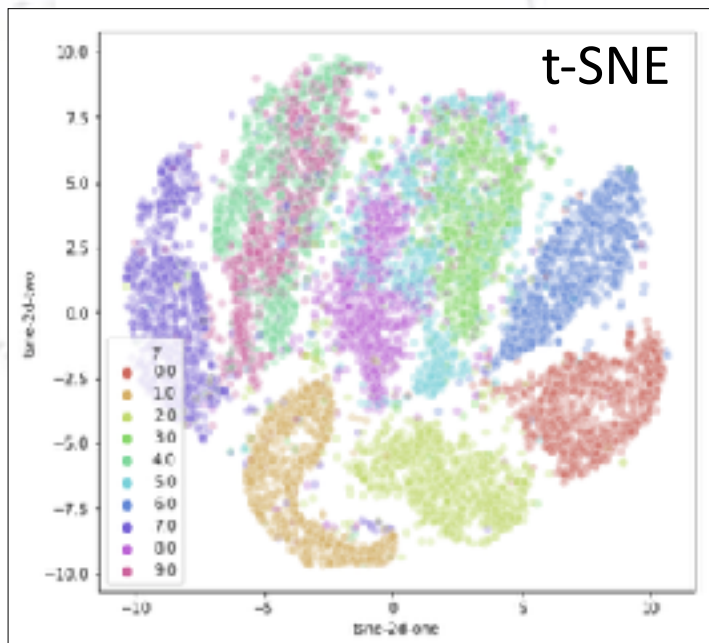
The background of the slide is a faded, light blue map of the North Atlantic Ocean. It features depth contours (isobaths) labeled with values such as 100, 150, 200, 250, 300, 350, and 400. A compass rose is visible in the upper left quadrant, and various navigational markings and text are scattered across the map, including "W 50° 15' N" and "180".

Unsupervised Learning: Dimensionality Reduction

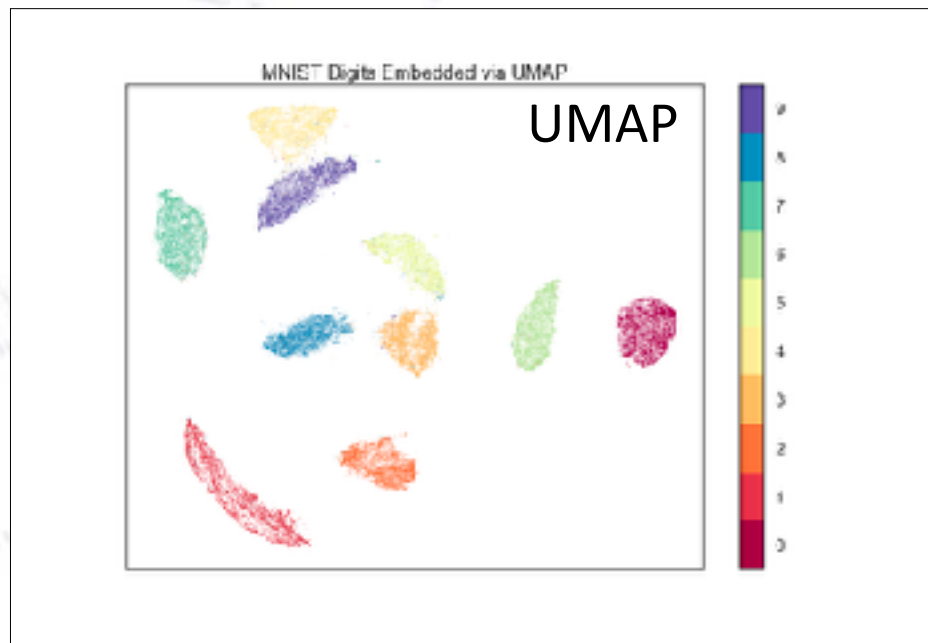
PCA, t-SNE & UMAP

High dimensionality has always been a curse - it is extremely hard to make sense of, and requires a lot of work and domain knowledge to boil down to few dimensions without losing a lot of information.

PCA has long reigned the linear case, and k-means the clustering, but two new(er) non-linear and powerful candidates are around: t-SNE and UMAP. Below are their performance on the MNIST data set.



Source: Towards data science (PCA and t-SNE)



Source: UMAP GitHub page: <https://github.com/lmcinnes/umap>

t-SNE Pro's and Con's

Pro: In the words of the t-Distributed stochastic neighbour embedding (t-SNE) paper, the t-SNE algorithm... *“...minimises the divergence between two distributions: a distribution that measures pairwise similarities of the input objects and a distribution that measures pairwise similarities of the corresponding low-dimensional points in the embedding”*.

The great thing about this is, that there are no assumptions about distributions, relationships, or number of clusters. The algorithm is non-linear, which gives it a clear edge over e.g. PCA.

Con: However, computationally it is a “heavy” (ugly?) algorithm, since t-SNE scales **quadratically** in the number of objects N . This limits its applicability to data sets with only a few thousand input objects; beyond that, learning becomes too slow to be practical (and the memory requirements become too large)”.

In real life, the t-SNE algorithm has especially had its impact in (a)DNA research, where the number of cases is typically not that large.

UMAP

UMAP builds on using **Riemannian manifolds!** Within differential geometry, this allows the definition of angles, hyper-area, and curvature in high dimensionality.

Abstract

UMAP (Uniform Manifold Approximation and Projection) is a novel manifold learning technique for dimension reduction. UMAP is constructed from a theoretical framework based in Riemannian geometry and algebraic topology. The result is a practical scalable algorithm that is applicable to real world data. The UMAP algorithm is competitive with t-SNE for visualization quality, and arguably preserves more of the global structure with superior run time performance. Furthermore, UMAP has no computational restrictions on embedding dimension, making it viable as a general purpose dimension reduction technique for machine learning.

UMAP paper, arXiv 1802.03426, Sep. 2020

The paper is quiet mathematical with (10) definitions, lemmas, and proofs in the appendix. I find it a bit hard to read, but like their discussion of scaling and cons.

UMAP

As in the t-SNE case, UMAP tries to find a metric in both the original (large) space X , and the lower dimension output space Y , which can be (topologically) matched:

At a high level, UMAP uses local manifold approximations and patches together their local fuzzy simplicial set representations to construct a topological representation of the high dimensional data. Given some low dimensional representation of the data, a similar process can be used to construct an equivalent topological representation. UMAP then optimizes the layout of the data representation in the low dimensional space, to minimize the cross-entropy between the two topological representations.

UMAP paper, arXiv 1802.03426, Sep. 2020

However, the metrics in X and Y used by UMAP and t-SNE differ:

For t-SNE these metrics are as follows:

$$v_{j|i} = \exp(-\|x_i - x_j\|_2^2 / 2\sigma_i^2)$$

$$w_{ij} = \left(1 + \|y_i - y_j\|_2^2\right)^{-1}$$

For UMAP they are:

$$v_{j|i} = \exp[(-d(x_i, x_j) - \rho_i) / \sigma_i]$$

$$w_{ij} = \left(1 + a \|y_i - y_j\|_2^{2b}\right)^{-1}$$

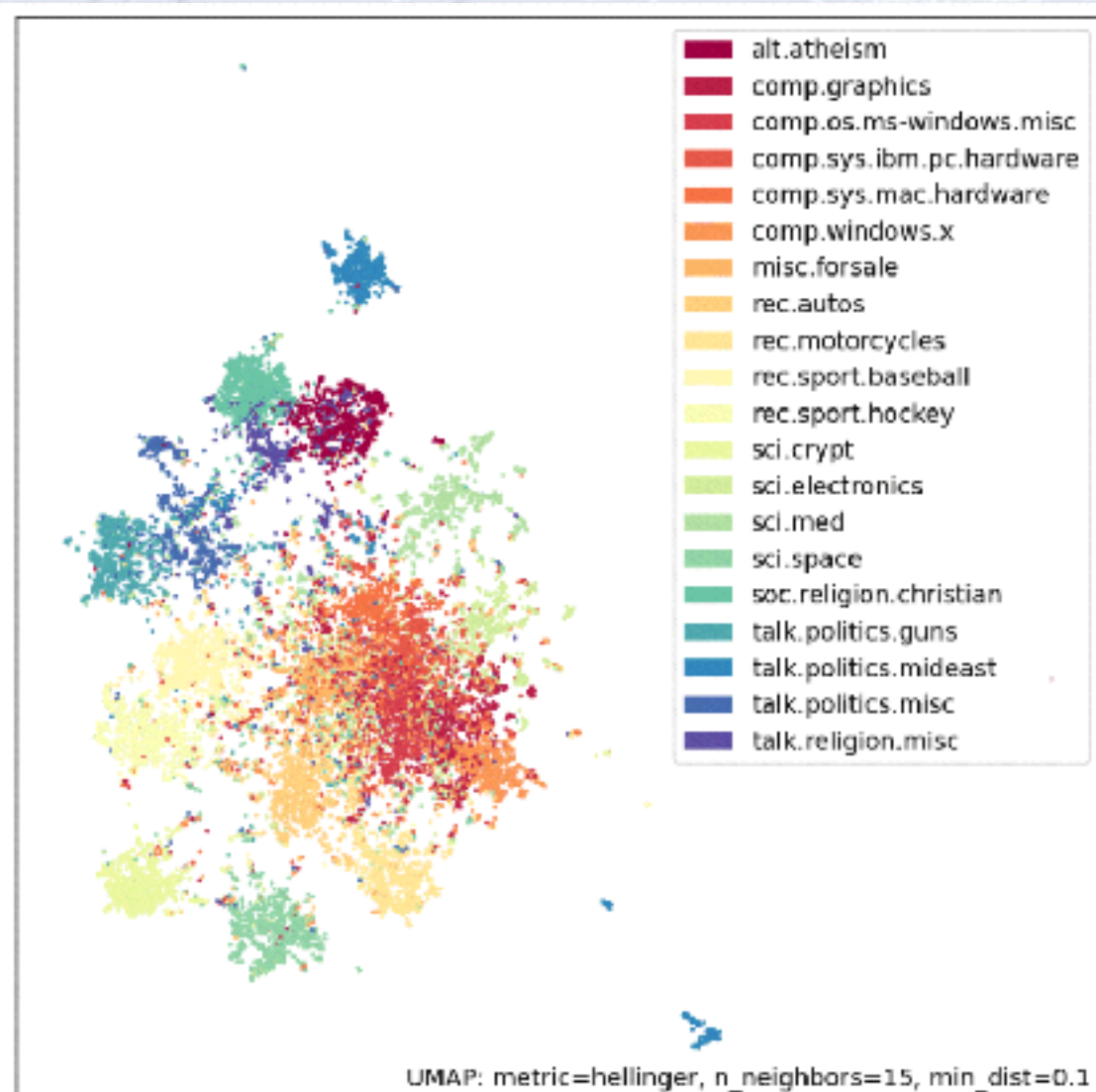
A faded nautical chart is visible in the background. It features depth contours labeled with numbers such as 100, 150, 200, 250, 300, and 350. A red location marker is present, labeled with the coordinates 'VAR 50°15'N' and 'W 115°10'W'. Other text on the chart includes 'WACHETIA' and '181 BITTEN ERW TACHT / 1818'.

Example use cases...

Mapping news group discussions

UMAP showing the differences between different news group discussion fora.

The ability to cluster fairly well would allow editors to direct text to the relevant news group.



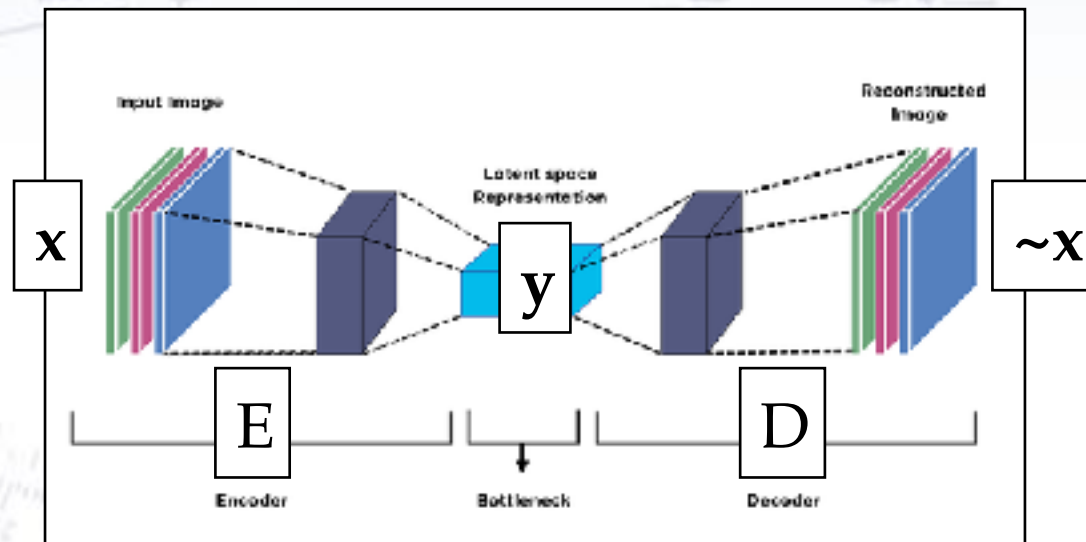


Unsupervised Learning: AutoEncoders

AutoEncoders

An **AutoEncoder** (AE) is a **coupled pair of encoder and decoder**. The encoder maps signals into code, and the decoder reconstructs the original signal from those codes.

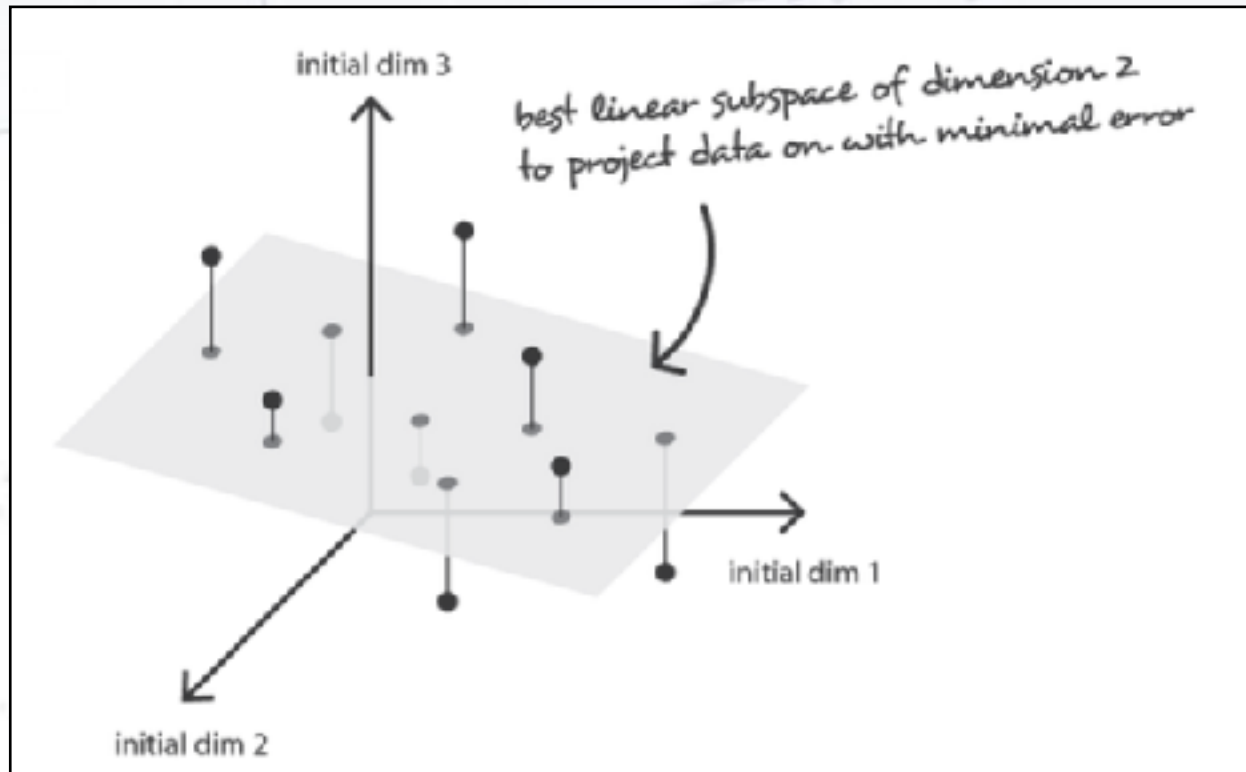
The pair is trained to have the most accurate reconstruction: If you give a signal x to an encoder E to get $y = E(x)$, then the decoder D should ensure that $D(y)$ is close to x .



One application is unsupervised feature learning, where it tries to construct a useful feature set from a set of unlabelled images. We could use the code produced as a source of features.

PCA as an autoencoder

A PCA is a linear AE. One can project a higher dimension down on (fewer) principle components (encoding) and then “reconstruct” the original data from the latent space, by choosing the low dimensional points in the original dimensionality.



Usage of AE

AEs are used for many things, such as:

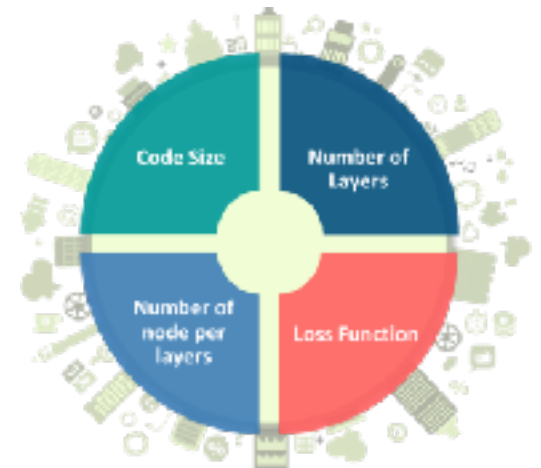
- Unsupervised learning (e.g. clustering) on images, sound, graphs, etc.
- Compression (with loss!) of e.g images
- De-noising and inpainting images
- Anomaly detection
- Training on large dataset with few labels

Most AEs are CNN based and produced for images. However, the AE concept is more general, and applies to anything, that can be passed through an NN.

The most central hyperparameters to consider are:

- Size of the latent space (code)
- Architecture of NN (layers and nodes)
- Loss function

As we shall see, these HPs to some extent determine what type of AE you're making.

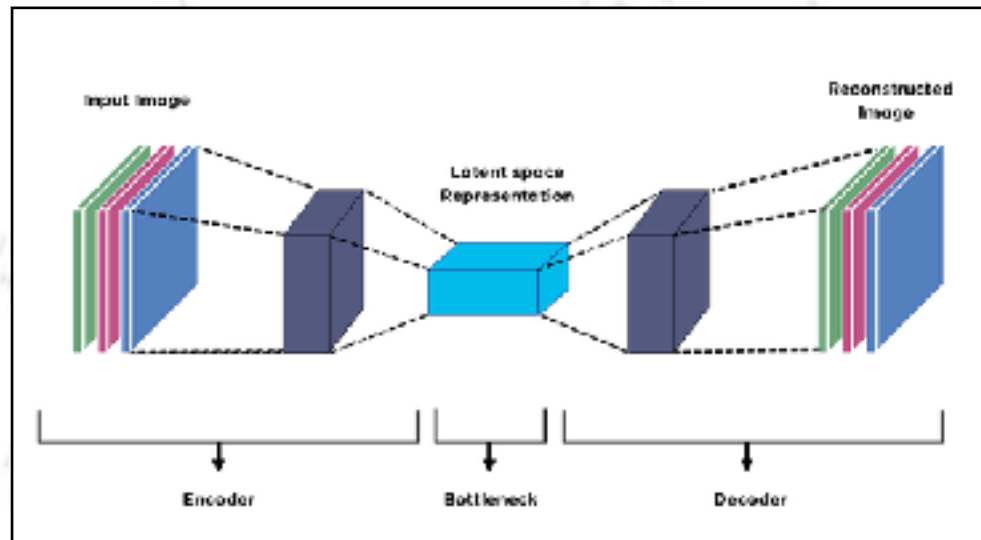


AE Unsupervised

Since one might have no knowledge of the content of images (or sound, etc.), training an AE is inherently unsupervised. The result is simply a latent space that is a good **representation** of the images.

However, this can be used to cluster “less simple data”, as one can apply both dimensionality reduction and/or clustering to the latent space.

This enables one to analyse very complex data in an unsupervised manner.
(e.g. “How many zebra calls exists?”)



Inpainting with AEs

AEs can also be used for “inpainting”, which means replacing damaged/lost parts of an image.





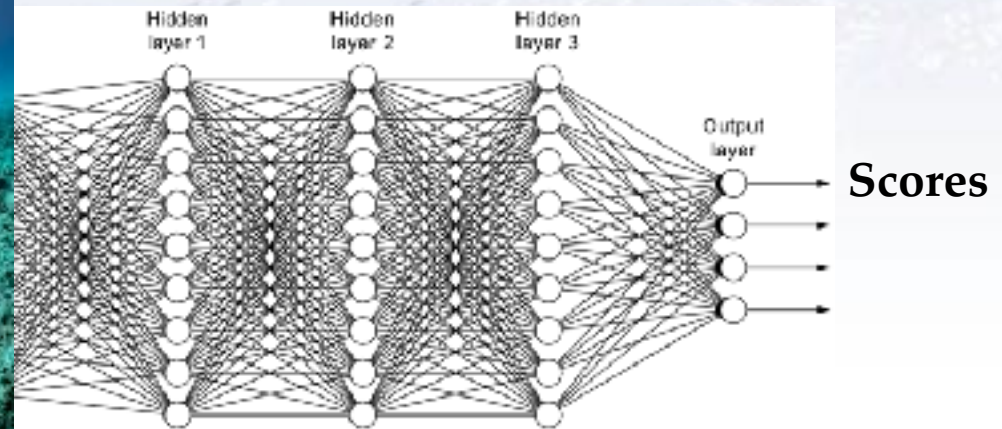
Shifting Gears...

CNN & GNNs

Why is image data special?

Given image data for e.g. classification, could we not just apply an NN?

“Is there a diver in this image?”



Well, since a typical image has 10M pixels, the first layer (size 1000?) would have about **10 billion parameters!**

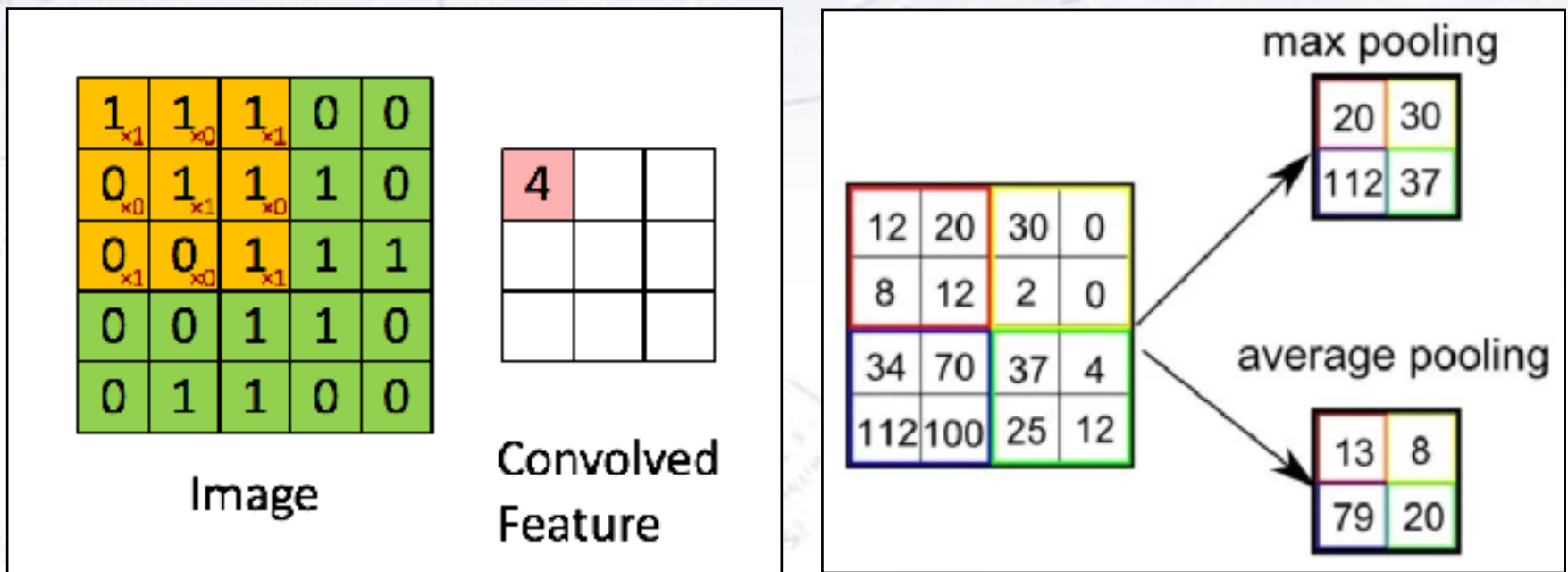
And even if we could train this, it wouldn't perform well:

If the diver was in another position or size, then the network wouldn't work.

Images represent data in a very special structure, that needs consideration.

Image convolution

The trick lies in using a convolution, which consists of filters/kernels (convolved feature), which is applied to the image as a matrix convolution, followed by an aggregation (not shown) and a pooling of pixels:



The filters contain learnable parameters, which adjust so that they match the task (“Is there a diver?”). The aggregation consists of adding all the “images” resulting from the convolutions, and values are then typically max pooled.

In colors...?

For color images, the process is the same, just with three convolutions:

0	0	0	0	0	0	...
0	156	155	156	158	158	...
0	133	154	157	159	159	...
0	149	151	155	158	159	...
0	146	146	149	153	158	...
0	143	143	143	148	158	...
...

Input Channel #1 (Red)

0	0	0	0	0	0	...
0	167	166	167	169	169	...
0	164	165	168	170	170	...
0	160	162	166	169	170	...
0	156	156	159	163	168	...
0	155	153	153	158	168	...
...

Input Channel #2 (Green)

0	0	0	0	0	0	...
0	163	162	163	165	165	...
0	160	161	164	166	166	...
0	156	158	162	165	166	...
0	155	155	158	162	167	...
0	154	152	152	157	167	...
...

Input Channel #3 (Blue)

-1	-1	1
0	1	-1
0	1	1

Kernel Channel #1



308

1	0	0
1	-1	-1
1	0	-1

Kernel Channel #2



-498

0	1	1
0	1	0
1	-1	1

Kernel Channel #3



164

+

+

Bias = 1

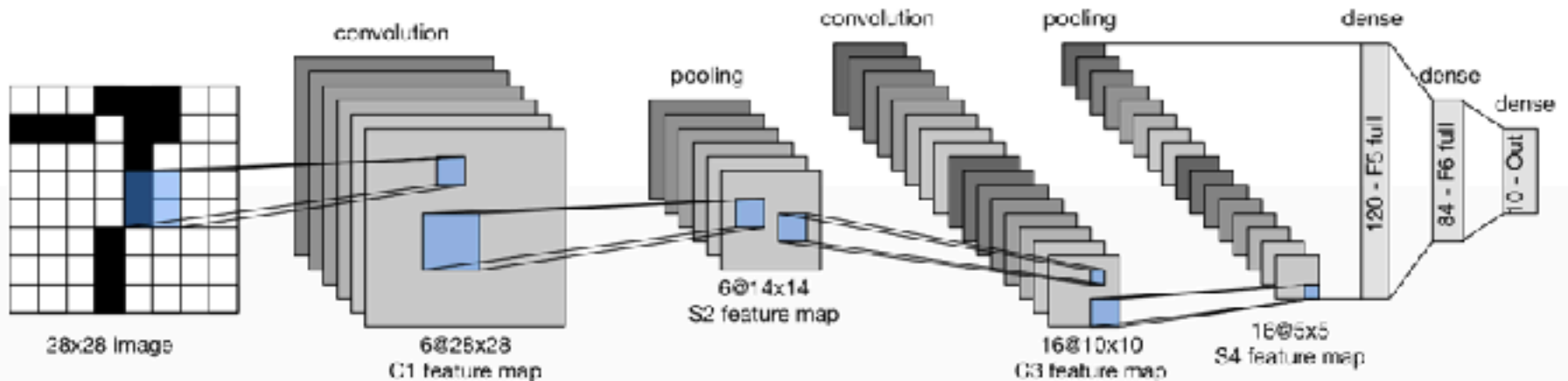
+ 1 - 25

Output

-25			...
			...
			...
			...
...

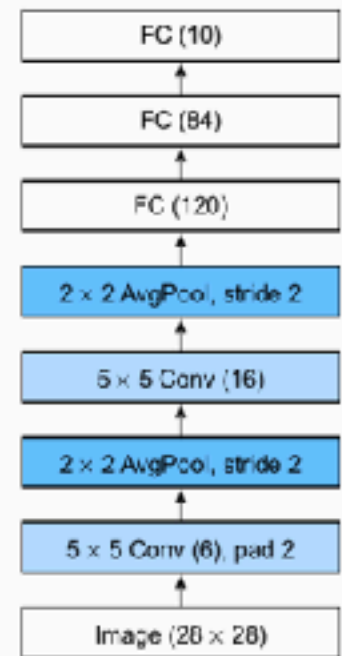
LeNet (1998)

LeCun et al, Proceedings of the IEEE, 1998



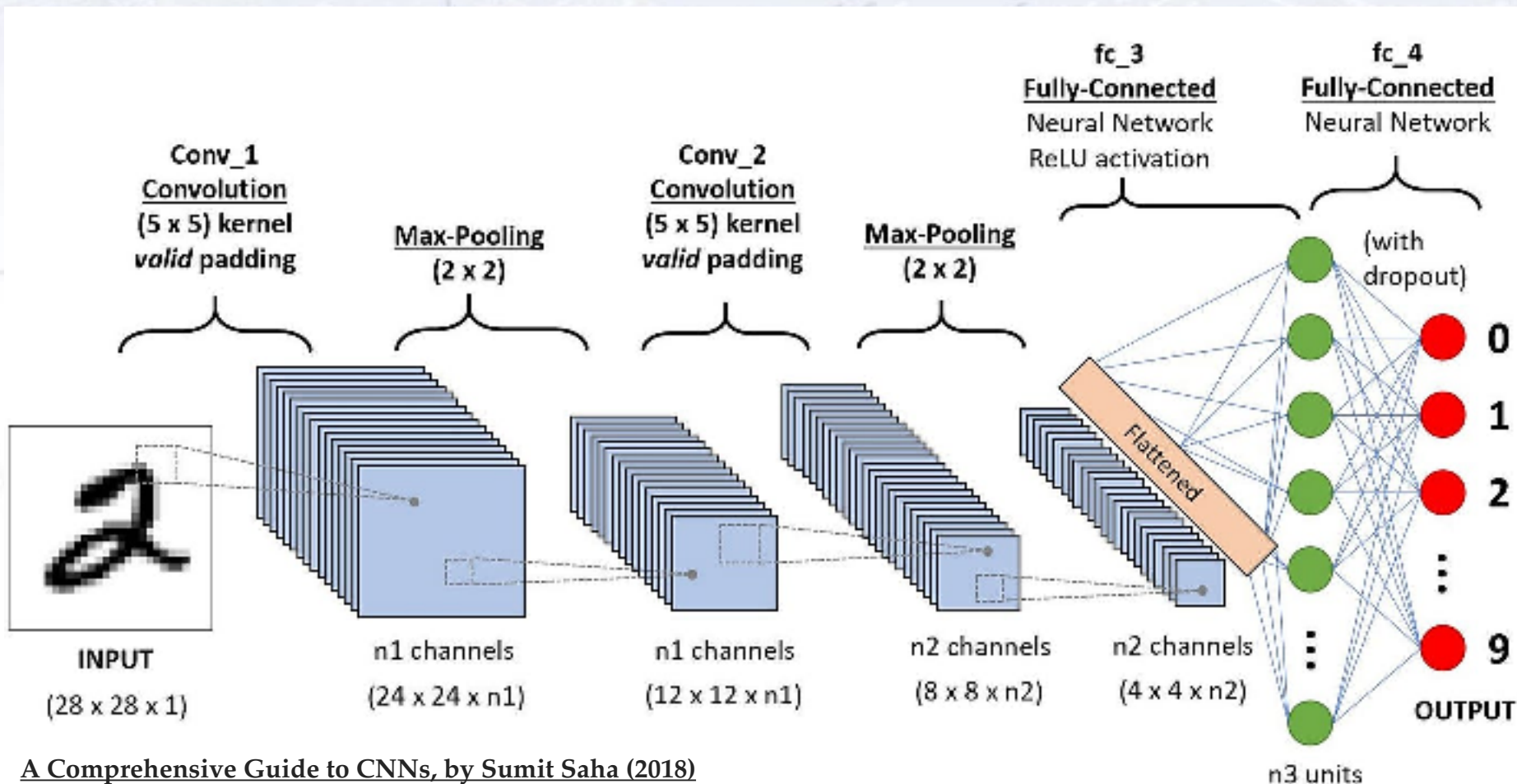
MNIST dataset

- Very early CNN ("the" CNN)
- Shows typical features of also modern classification CNNs: (pooling, pixel dims \rightarrow feature dims, ...)



LeNet (1998)

Typical (simple) CNN structure with dimensions and actions annotated.
Try to work out the number of parameters in this model...



[A Comprehensive Guide to CNNs, by Sumit Saha \(2018\)](#)

Interactive CNN

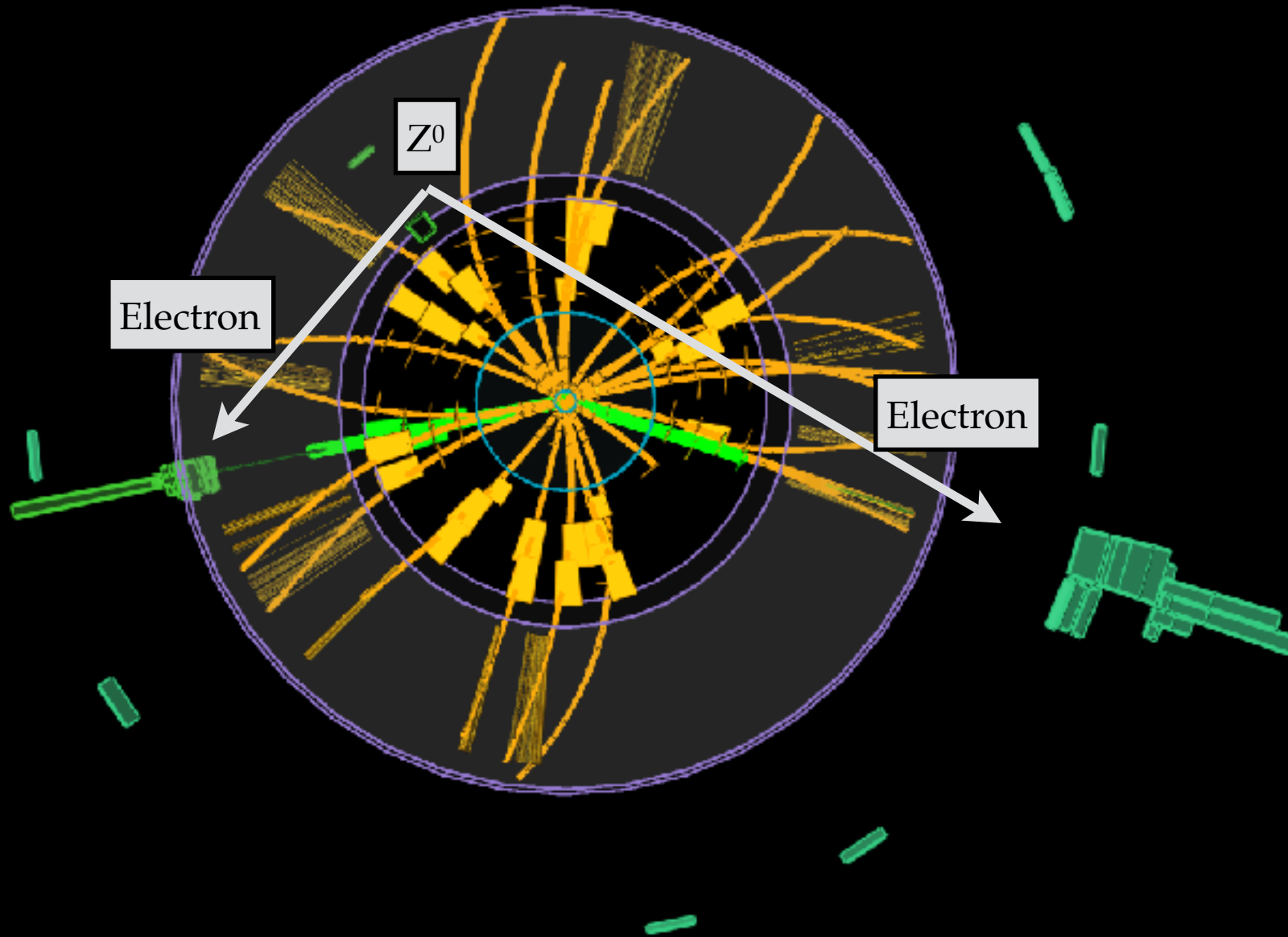
This network has 1024 nodes on the bottom layer (corresponding to pixels), six 5x5 (stride 1) convolutional filters in the first hidden layer, followed by sixteen 5x5 (stride 1) convolutional filters in the second hidden layer, then three fully-connected layers, with 120 nodes in the first, 100 nodes in the second, and 10 nodes in the third. The convolutional layers are each followed by downsampling layer that does 2x2 max pooling (with stride 2).

The screenshot shows an interactive CNN interface for digit recognition. On the left, there is a drawing area with a large white digit '2' on a black background. Below the drawing area are controls for erasing (X), drawing (pencil), and smoothing (brush). The main display area shows the input drawing and its processed versions through various layers. At the top, there are two rows of small, colorful feature maps. Below these are several rows of larger, more refined digit '2' images, showing the network's internal representations and final output. On the right side, there is a 'Layer visibility' panel with a list of layers and their visibility status:

Layer	Visibility
Input layer	Hide
Convolution layer 1	Hide
Downsampling layer 1	Hide
Convolution layer 2	Hide
Downsampling layer 2	Hide
Fully-connected layer 1	Hide
Fully-connected layer 2	Hide
Output layer	Hide

At the bottom left, the text 'Adam Harley: Interactive number CNN' is visible. At the bottom center, there is a large, clear digit '2' on a black background.

Electron energy with CNN

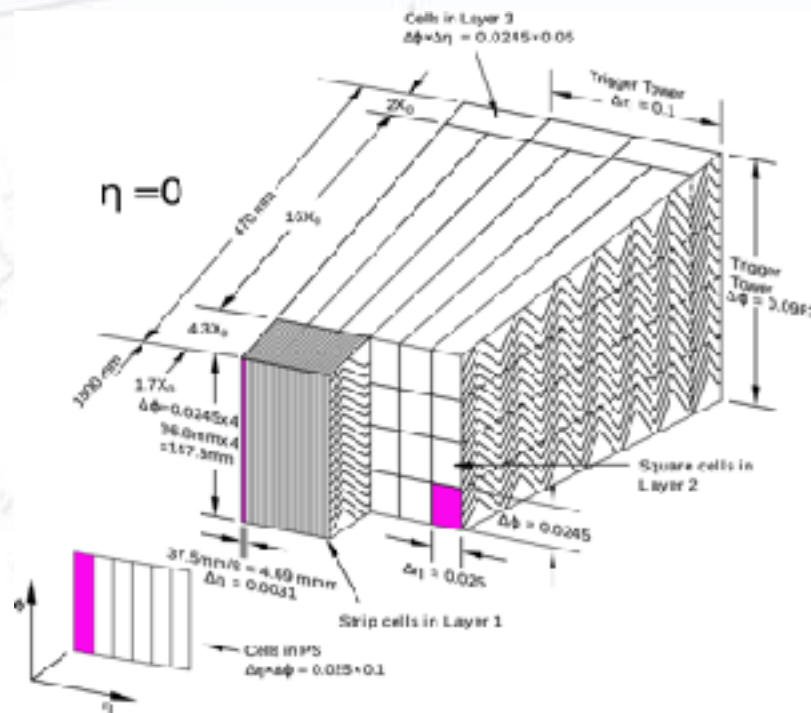


The input variables

The variables are both scalar and cell based.
The scalars can be seen in table on the right.

We consider the cell energies in the LAr calorimeter as pixels in four images. The cells contain two (used) types of information:

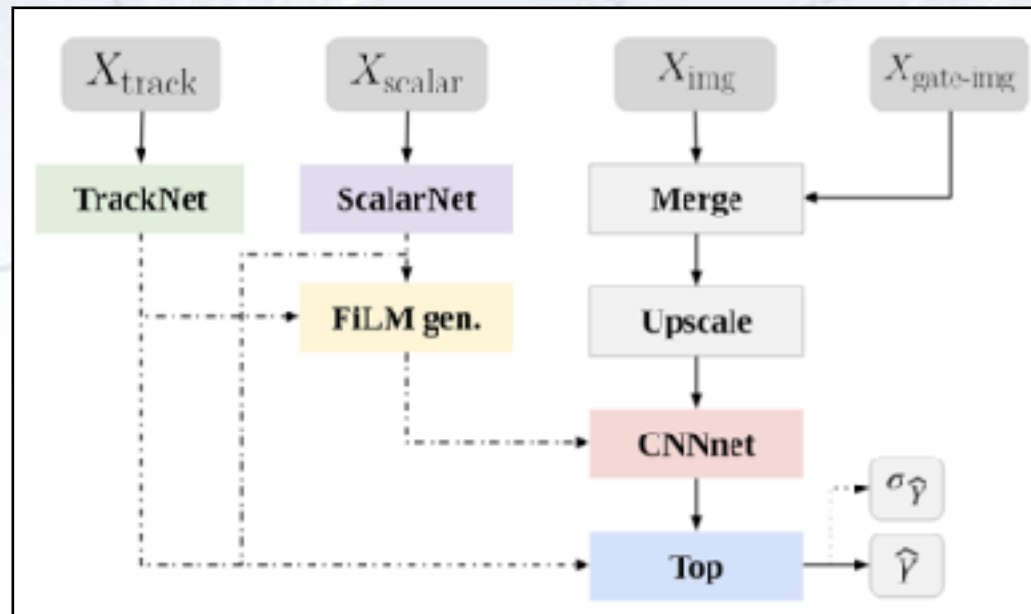
- Energy (primary variable)
- Time of cell energy



Type	Name	Description
Energy	E_{cell}	Energy deposit in layer 1-3 of ECAL.
	η_{cluster}	η cell index of cluster of layer 2.
	$f_{\text{D}_{\text{layer}}}$	Ratio of energy between layer 0 and E_{cell} in $ \eta < 1.8$ (end of layer 0).
	R12	Ratio of energy between layer 1 and 2 in the ECAL.
	p_T^{track}	p_T estimated from tracking for the particle (only e).
	E_{EGS}	Ratio between the energy in the crack scintillators and E_{cell} within $1.1 < \eta < 1.6$.
Geometric	$E_{\text{tile-gap}}$	Sum of the energy deposited in the tile-gap.
	η	Pseudorapidity of the particle.
	$\Delta\phi_2^{\text{recoiled}}$	Difference between ϕ , as extrapolated by tracking, use for ECAL momentum estimation and ϕ of the ECAL cluster.
	η_{recoiled}	Relative η position w.r.t. the cell edge of layer 2 in the ECAL.
	$\Delta\eta_2$	Difference between η , as extrapolated by tracking, use for ECAL momentum estimation and η of the ECAL cluster (only e).
	$\text{pos}_{\text{cell}}^{\eta}$	Relative position of η within cell in layer 2 in ECAL. $2(\eta_{\text{cluster}} - \eta_{\text{recoiled}})/0.025 - 1$, η_{cluster} is η of the barycenter of the cluster and η_{recoiled} is η of the most energetic cell of the cluster.
Misc	$\Delta\phi_{\text{max}}$	Relative position in ϕ in a cell $\text{mod}(2\pi + \phi, \pi/32) - \pi/32$.
	$\langle\mu\rangle$	Average proton-proton interaction per bunch crossing.
	n_{tracks}	# of tracks assigned (only e).
	$n_{\text{verticesReco}}$	Number of reconstructed vertices.

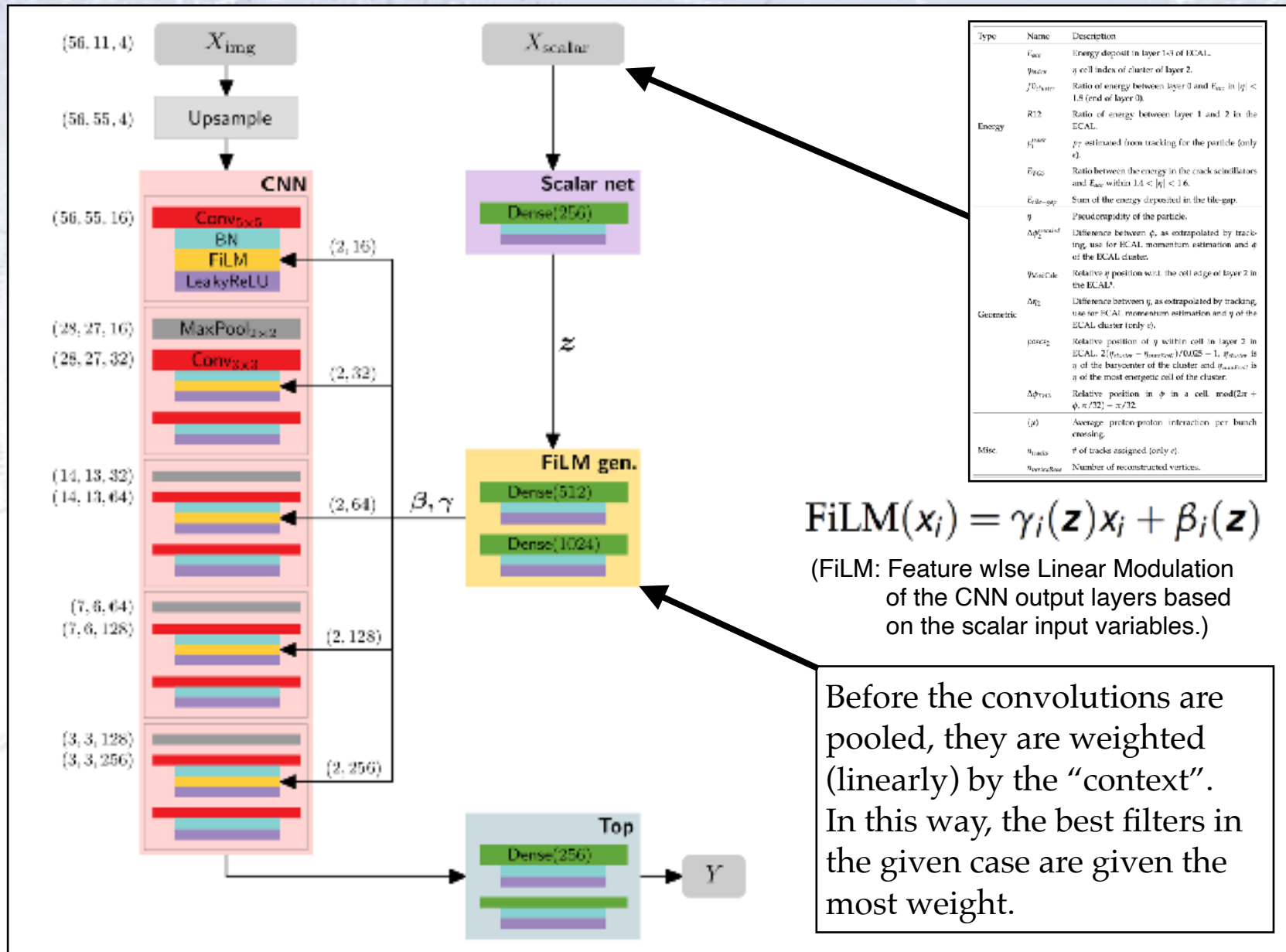
The network architecture

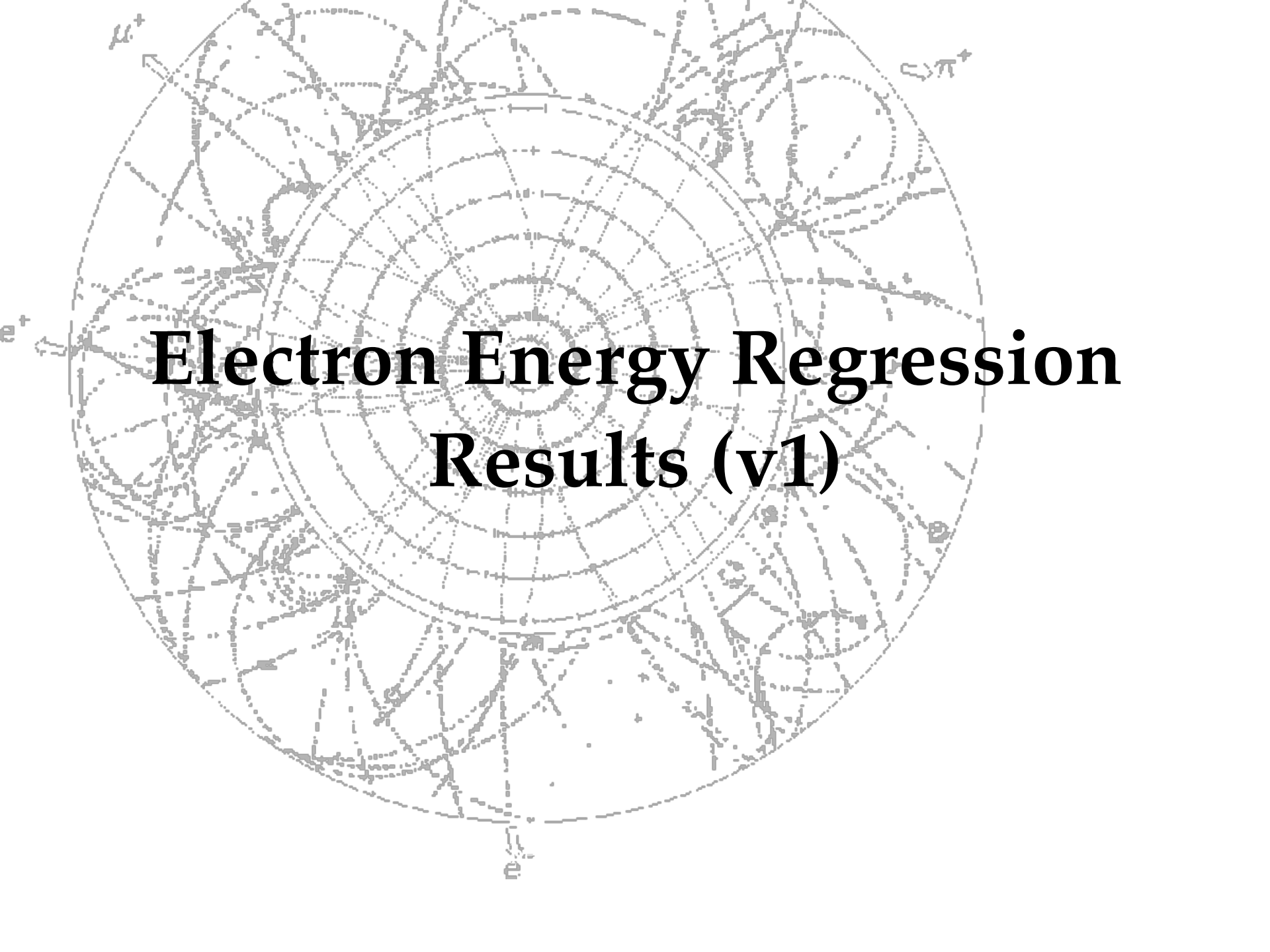
There are many ways to combine the input variables, and we have considered the following architectures, where the dashed lines are the considerations.



First, let us consider each part...

Feature wise Linear Modulation

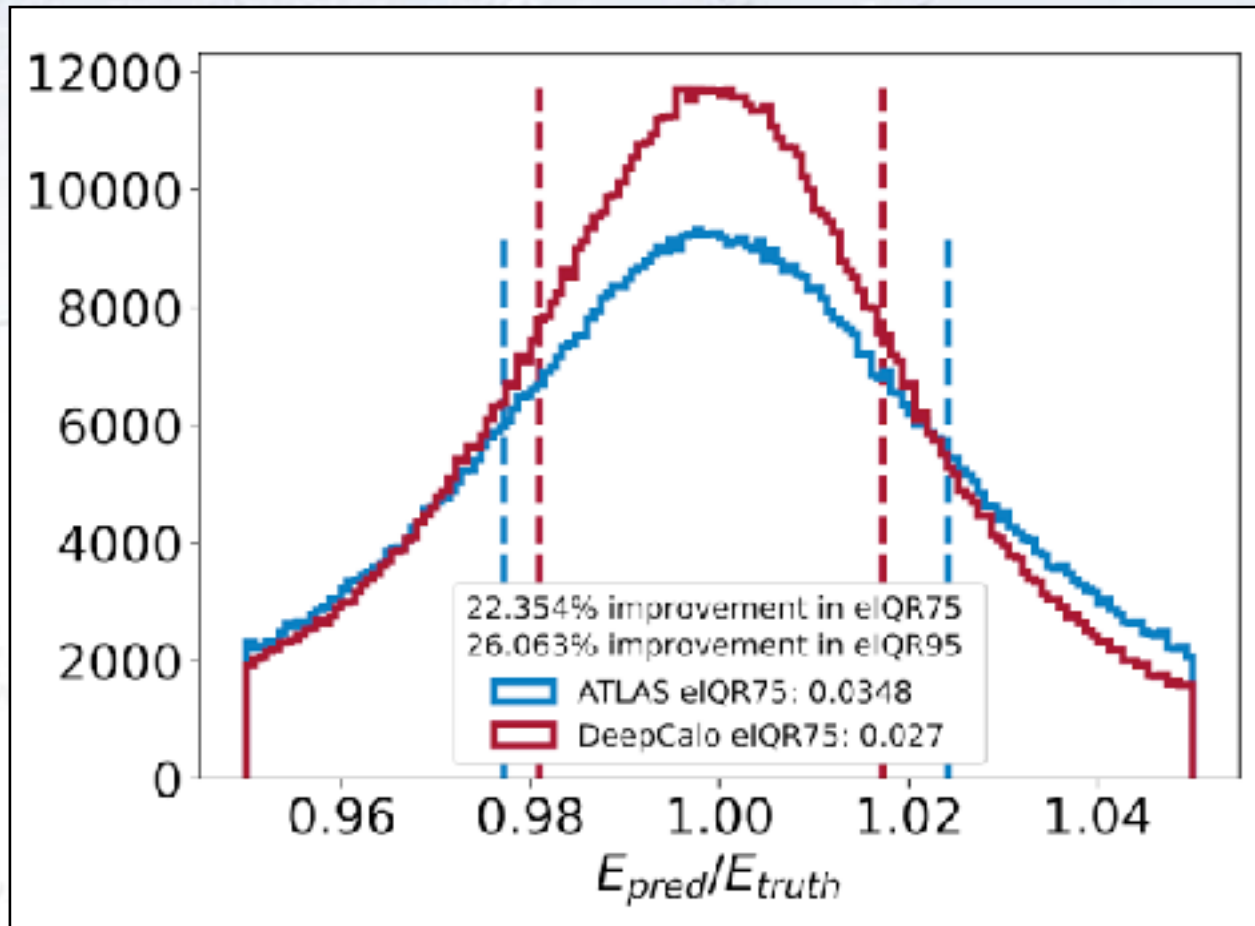




**Electron Energy Regression
Results (v1)**

The results in 1D - MC

Integrating the previous plot into 1D considering the RE distribution, we see a general sharpening. The improvement in relative eIQR (reIQR) is about 22%.

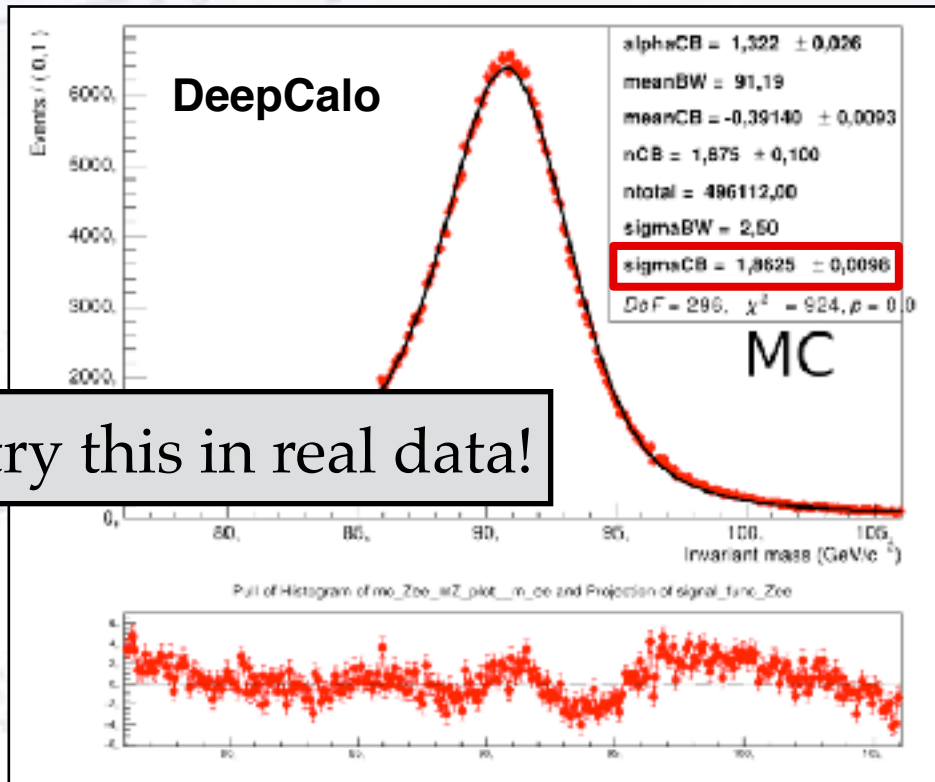
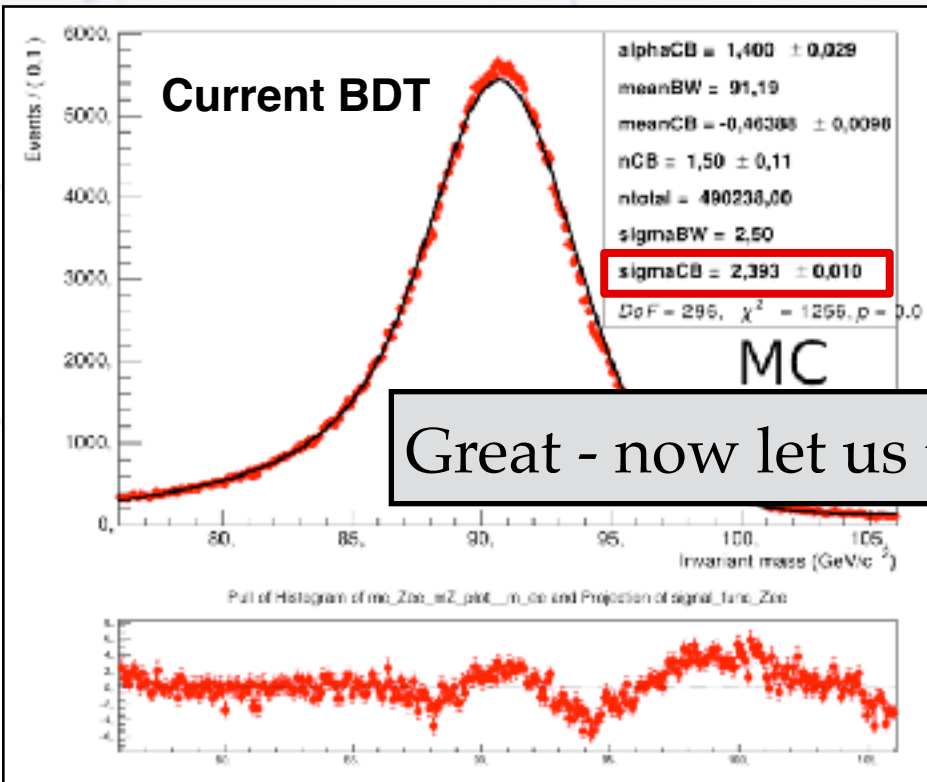


Naively, we would of course love to see a similar number in data!

Result in Zee - MC

On the Zee peak, we evaluate the improvement by fitting with a BW \otimes CB fit, considering the CB width (sigmaCB) as the performance parameter. We get:

$$\left\langle 1 - \frac{\sigma_{CB}^{DeepCalo}}{\sigma_{CB}^{ATLAS}} \right\rangle = 1 - \frac{1.8310 \pm 0.006}{2.393 \pm 0.01} = 23.5 \pm 0.4\%$$



Great - now let us try this in real data!

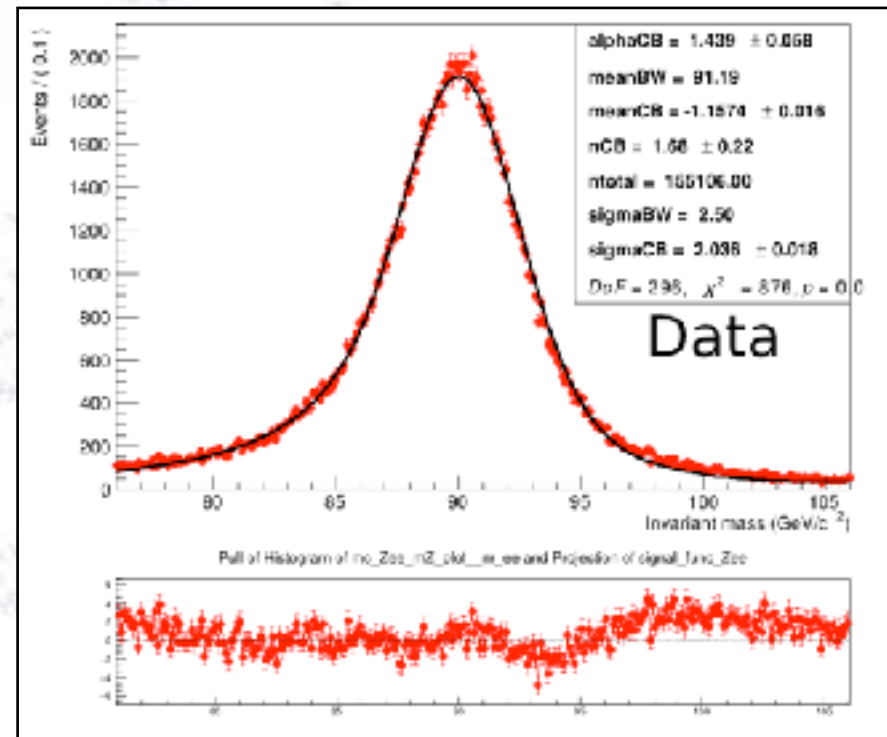
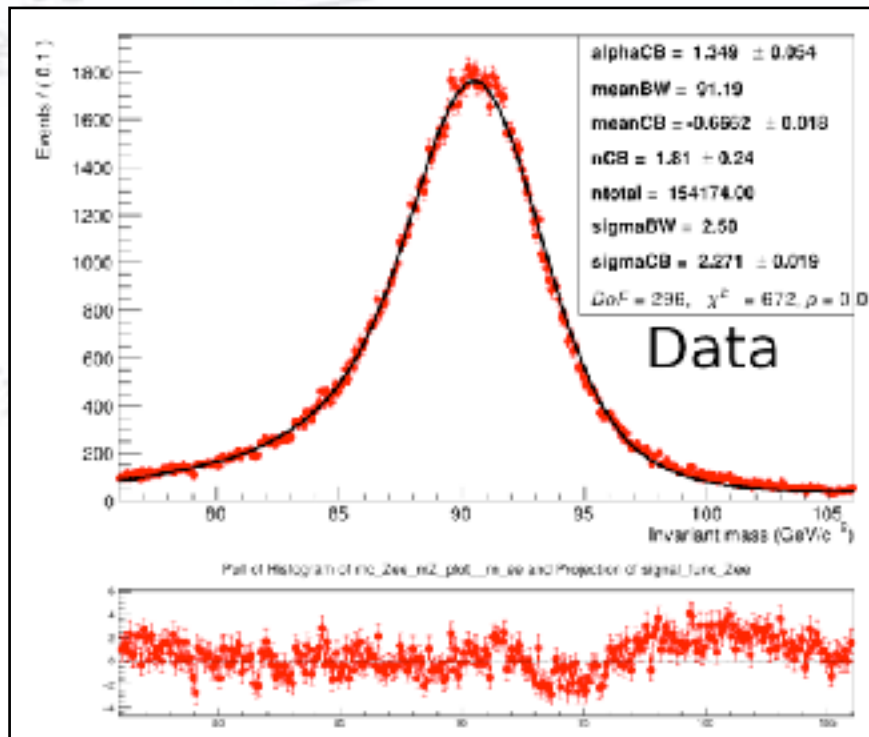
Results on Zee - data (v1)

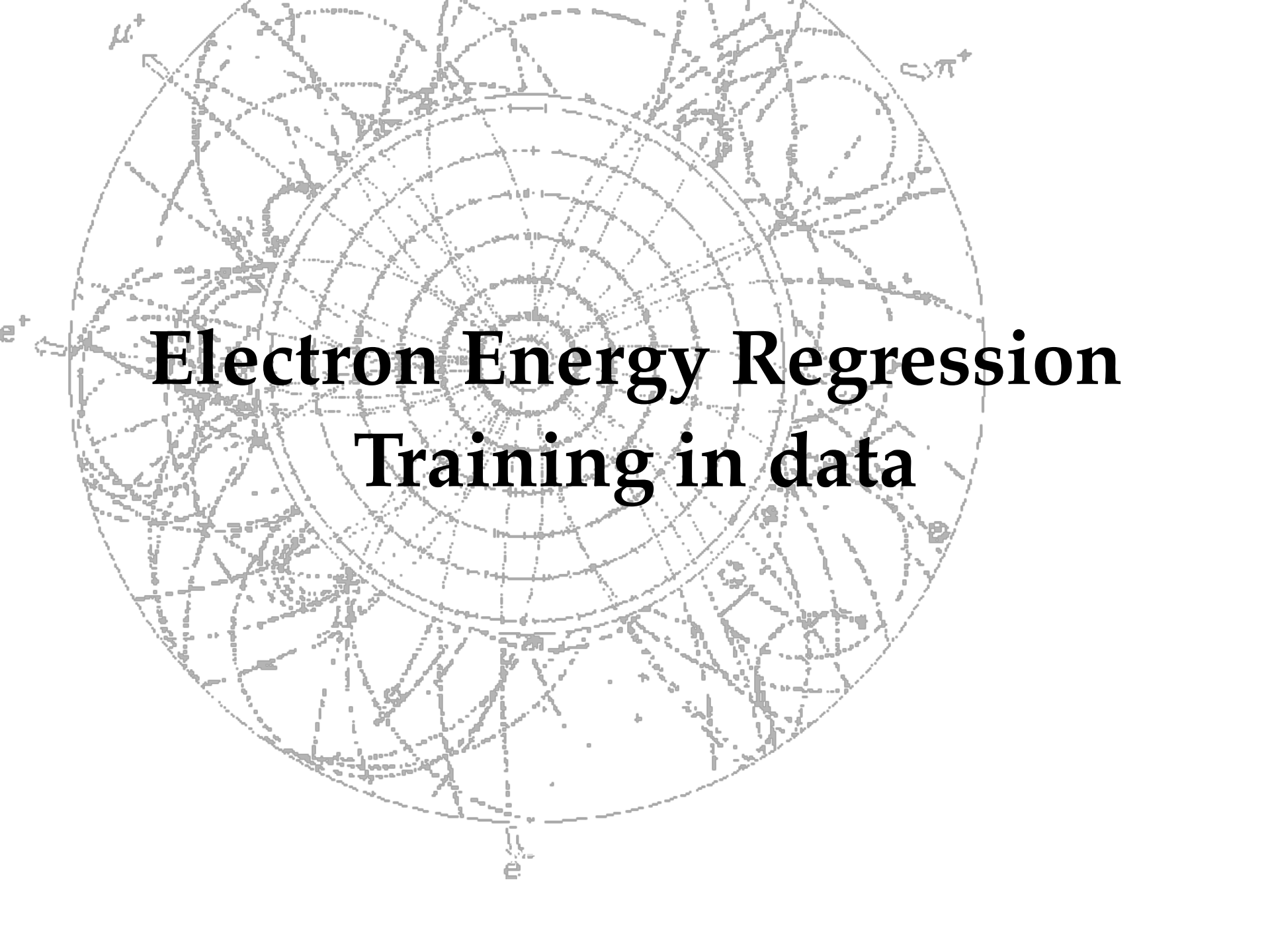
The result we get is a much more modest improvement:

$$\left\langle 1 - \frac{\sigma_{CB}^{DeepCalo}}{\sigma_{CB}^{ATLAS}} \right\rangle = 1 - \frac{2.058 \pm 0.010}{2.271 \pm 0.019} = 9.4 \pm 0.9\%.$$

Though perhaps a little disappointing, this is not surprising, as we can not expect the MC to mimic data perfectly in the very large space considered.

Also, models trained on Zee do not generalise well to all energies (EG, 6.8%).





Electron Energy Regression Training in data

Training in data

Using Zee events with invariant masses 86-97 GeV, one can get “approximate labels” in data, by assuming the true Z mass:

Using such labels, we train in data and get...

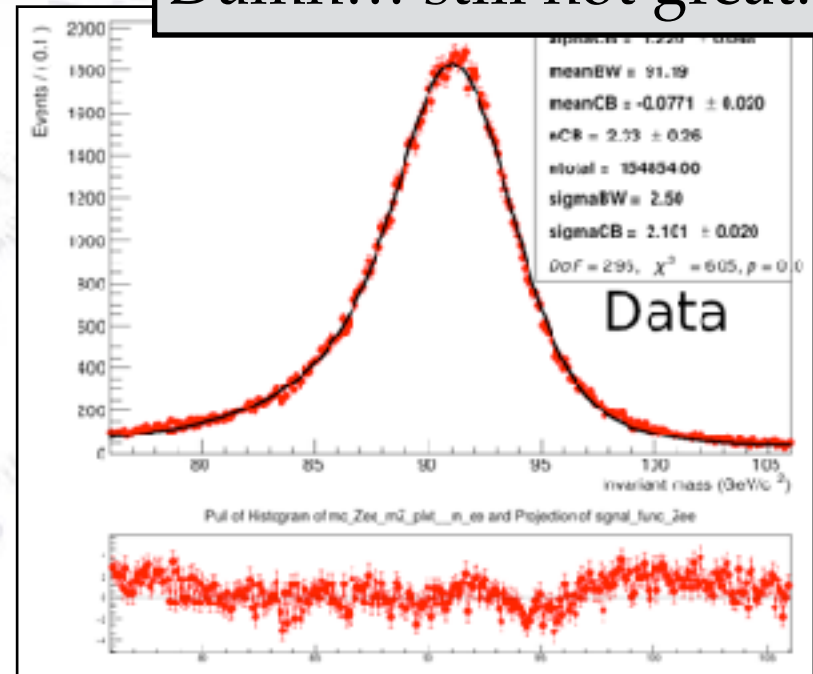
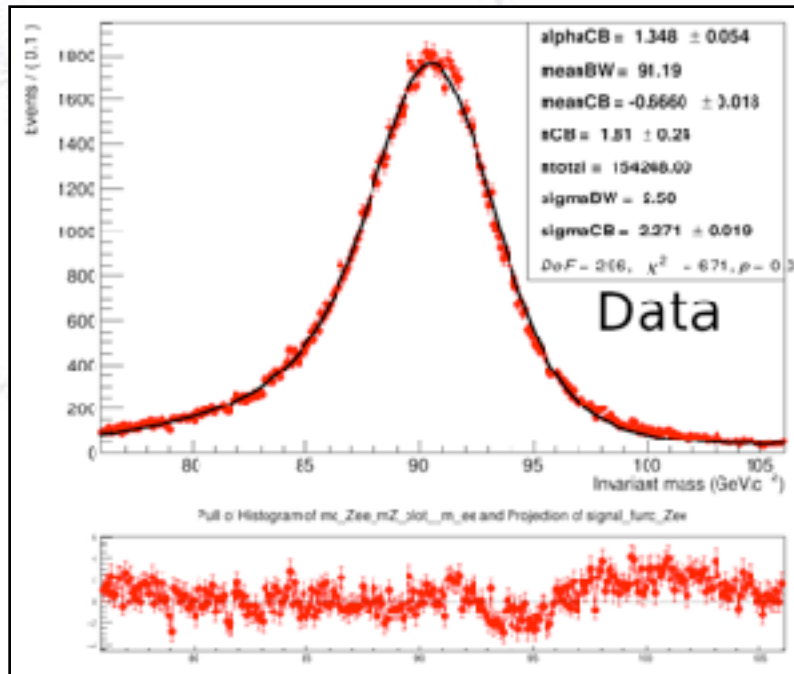
$$M^2 = 2p_{T,1}p_{T,2}(\cosh(\eta_1 - \eta_2) - \cos(\phi_1 - \phi_2)), \quad p_T = E_T \downarrow$$

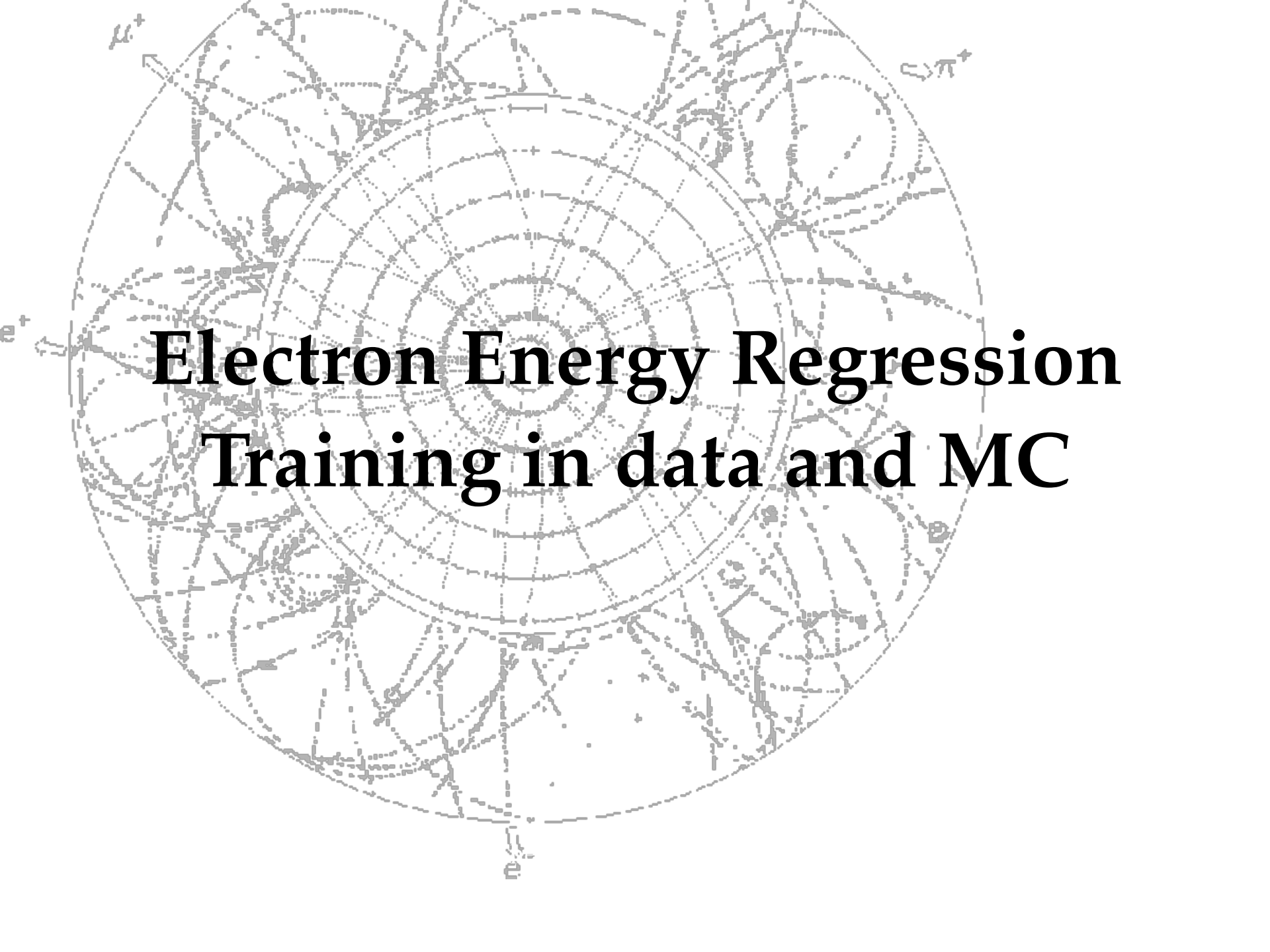
$$E_{\text{label,data}} = \frac{M^2}{2E_{T,2}(\cosh(\eta_1 - \eta_2) - \cos(\phi_1 - \phi_2))'}$$

with $E_{T,2} = E_{\text{calib}}^{(BDT)}$ and $M^2 = 91.19^2$

$$\left\langle 1 - \frac{\sigma_{CB}^{\text{DeepCalo}}}{\sigma_{CB}^{\text{ATLAS}}} \right\rangle = 5.9 \pm 0.9\%$$

Damn... still not great!





Electron Energy Regression
Training in data and MC

Training in data and MC

Once we have labels in data, there is nothing keeping us from combining the loss functions of MC and data (they even have the same form), and thus training **simultaneously** in data and MC:

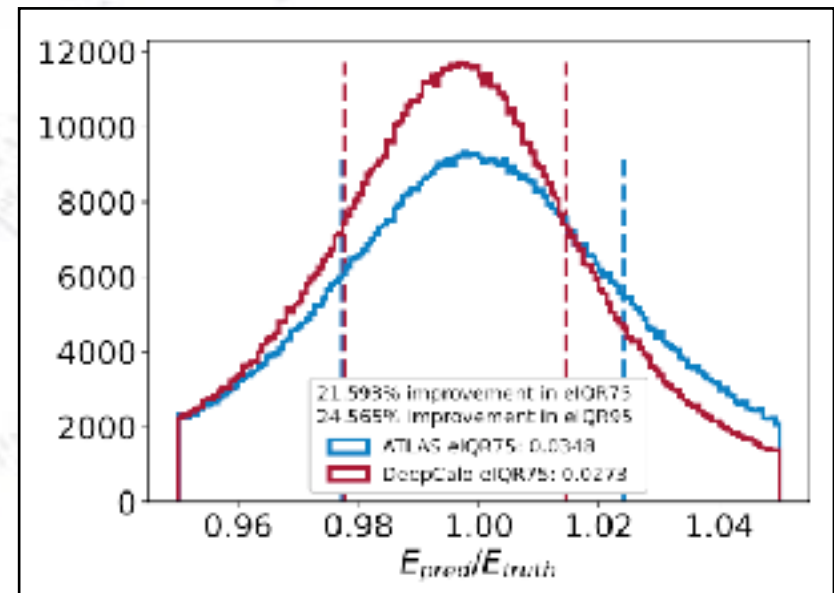
$$\mathcal{L}(y, \hat{y}) = \mathcal{L}(y_{(\text{Zee, MC})}, \hat{y}_{(\text{Zee, MC})}) + \mathcal{L}(y_{(\text{Zee, Data})}, \hat{y}_{(\text{Zee, Data})})$$

This allows the model to both use the “strength” of MC, but also learn the differences between MC and real data.

Doing this and trying out the result in MC first yields:

$$\langle \text{relIQR}_{75}^{\text{DeepCalo}} \rangle = 22.1 \pm 0.3\%$$

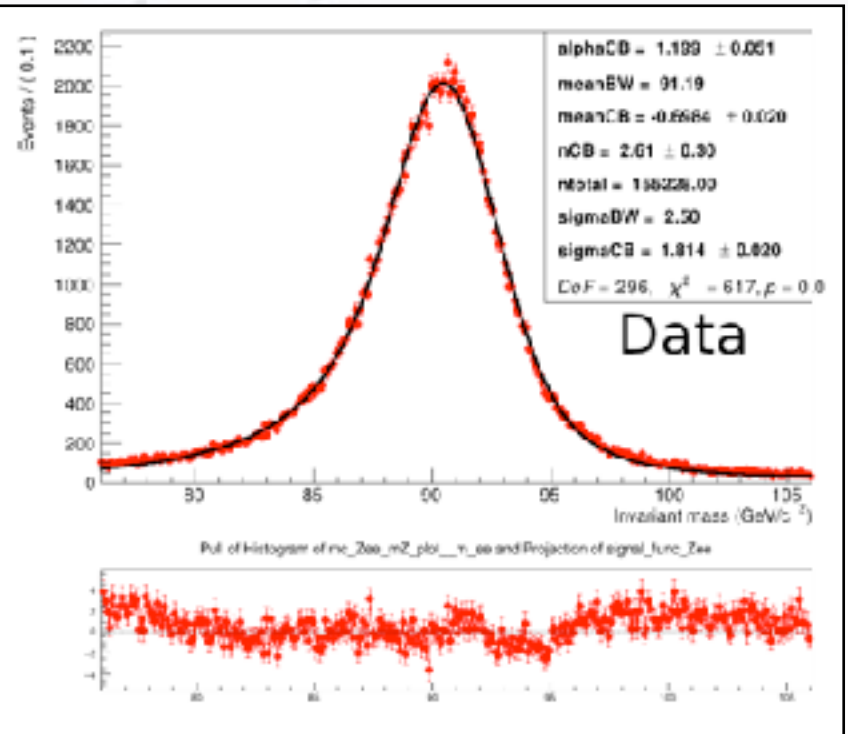
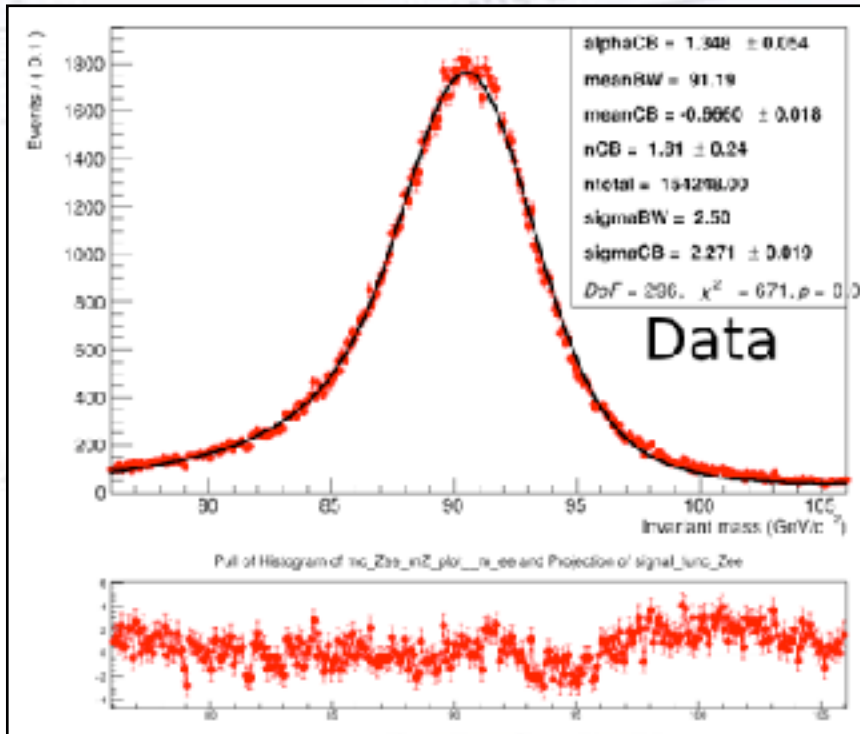
OK, so at least it doesn't ruin the model for MC. Now let us try data...



Result in data (v2)

The result in data is rather encouraging, and **greater than the sum of the improvements** from training separately in MC (9.4%) and data (5.9%).

$$\left\langle 1 - \frac{\sigma_{CB}^{DeepCalo}}{\sigma_{CB}^{ATLAS}} \right\rangle = 1 - \frac{1.86 \pm 0.010}{2.271 \pm 0.019} = 18.3 \pm 0.8\%$$



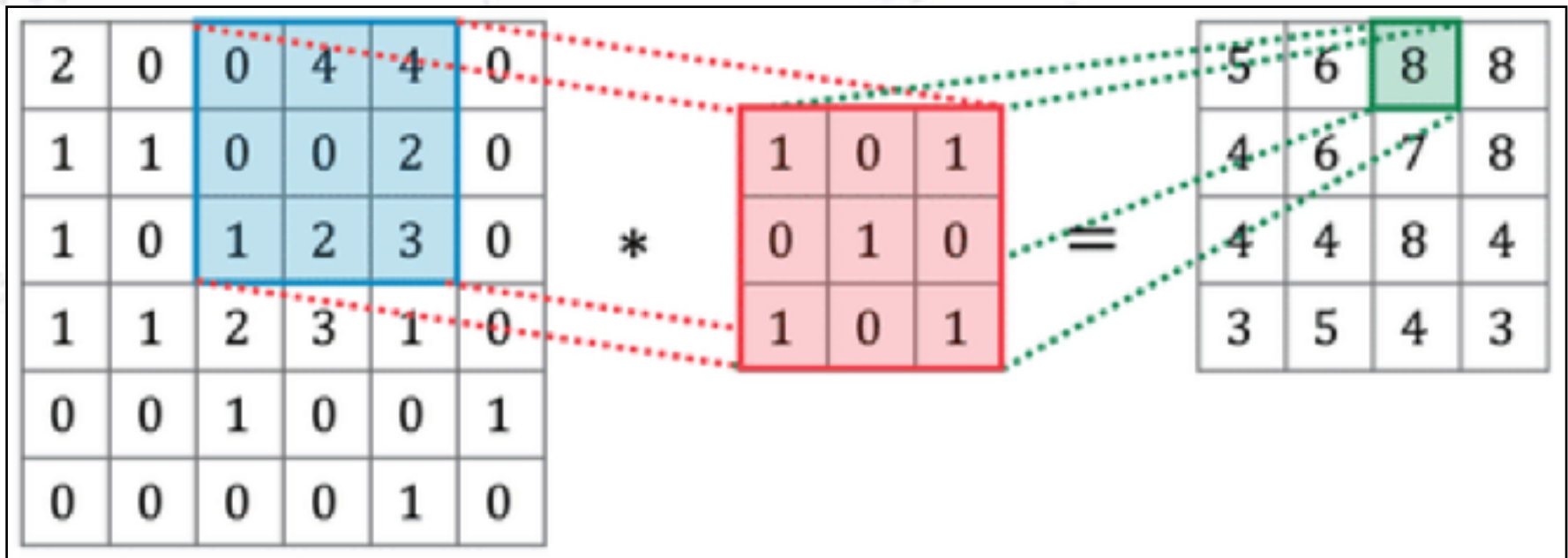
A background map of the Pacific Ocean, showing a coordinate grid with latitude and longitude lines. A red dot is marked on the map at approximately 30°N, 150°W. The text "VAR 30°15'N" is visible near the dot. Other text on the map includes "PACIFIC OCEAN" and "18° BITTER END TACHT 1879".

Graph NNs

Motivation for GNNs

Let us consider images in a more abstract sense:

- They consist of (typically 3-4, RGB or CMYK) numbers in a **matrix structure**.
- The distance between neighbouring cells is **constant**.



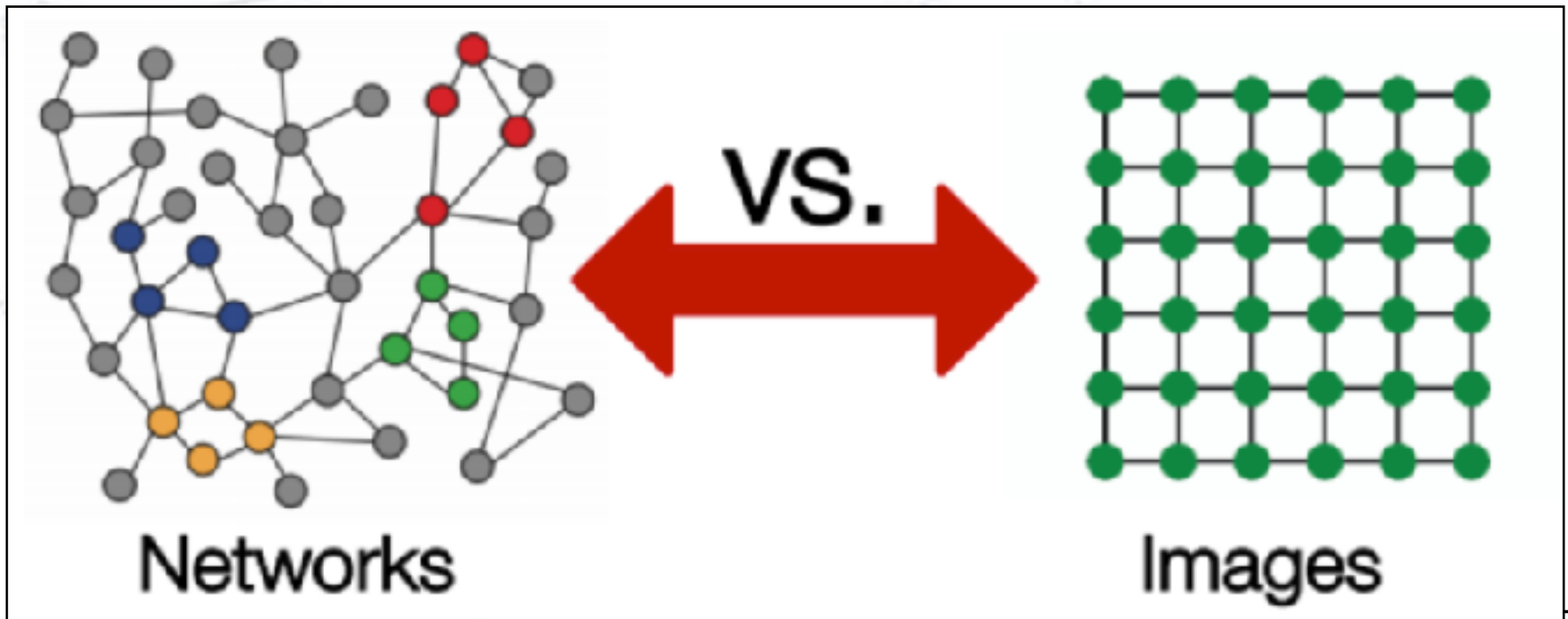
Motivation for GNNs

Let us consider images in a more abstract sense:

- They consist of (typically 3-4, RGB or CMYK) numbers in a **matrix structure**.
- The distance between neighbouring cells is **constant**.

What if the data was not an image, but we wanted to use a CNN anyway?

This problem is not uncommon... (<https://arxiv.org/pdf/2101.11589.pdf>)

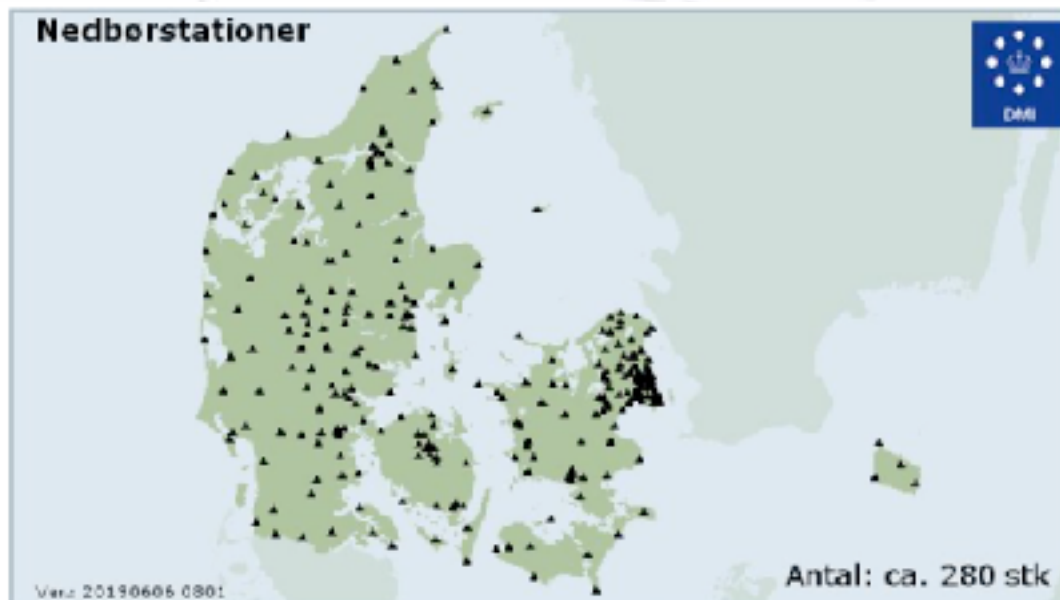


Example of non-CNN data

Take the example of weather stations:

- There are about 280 weather stations distributed **non-uniformly** throughout Denmark.
- Each station provides say:
[temperature, wind speed, wind direction, humidity, latitude, longitude]

What would an “image” of this data look like? And would a CNN work on it?



Example of non-CNN data

Take the example of weather stations:

- There are about 280 weather stations distributed **non-uniformly** throughout Denmark.
- Each station provides say:
[temperature, wind speed, wind direction, humidity, latitude, longitude]

What would an “image” of this data look like? And would a CNN work on it?



Nedborstationer

The background of the slide is a map of Denmark with small black dots representing weather stations. The map is titled 'Nedborstationer' (Precipitation stations) and includes a small European Union flag in the top right corner. At the bottom right of the map, it says 'Total: ca. 280 stk'.

By forcing problems with irregular geometry into images, we're shaping the problem to the tool, and not the tool to the problem!

Is there a different Machine Learning paradigm that has no underlying assumption on the geometry of the data?

Yes!

Graph Neural Networks

A faded nautical chart serves as the background. It features depth contours labeled with numbers like 100, 150, 200, 250, 300, and 350. A prominent contour is labeled '100'. The chart also shows latitude and longitude lines, with '10° 15' N' and '150° 15' W' visible. Other text on the chart includes 'PACIFIC' and '187 BITTEN ERN TAUCHT 1875'.

What is a Graph?

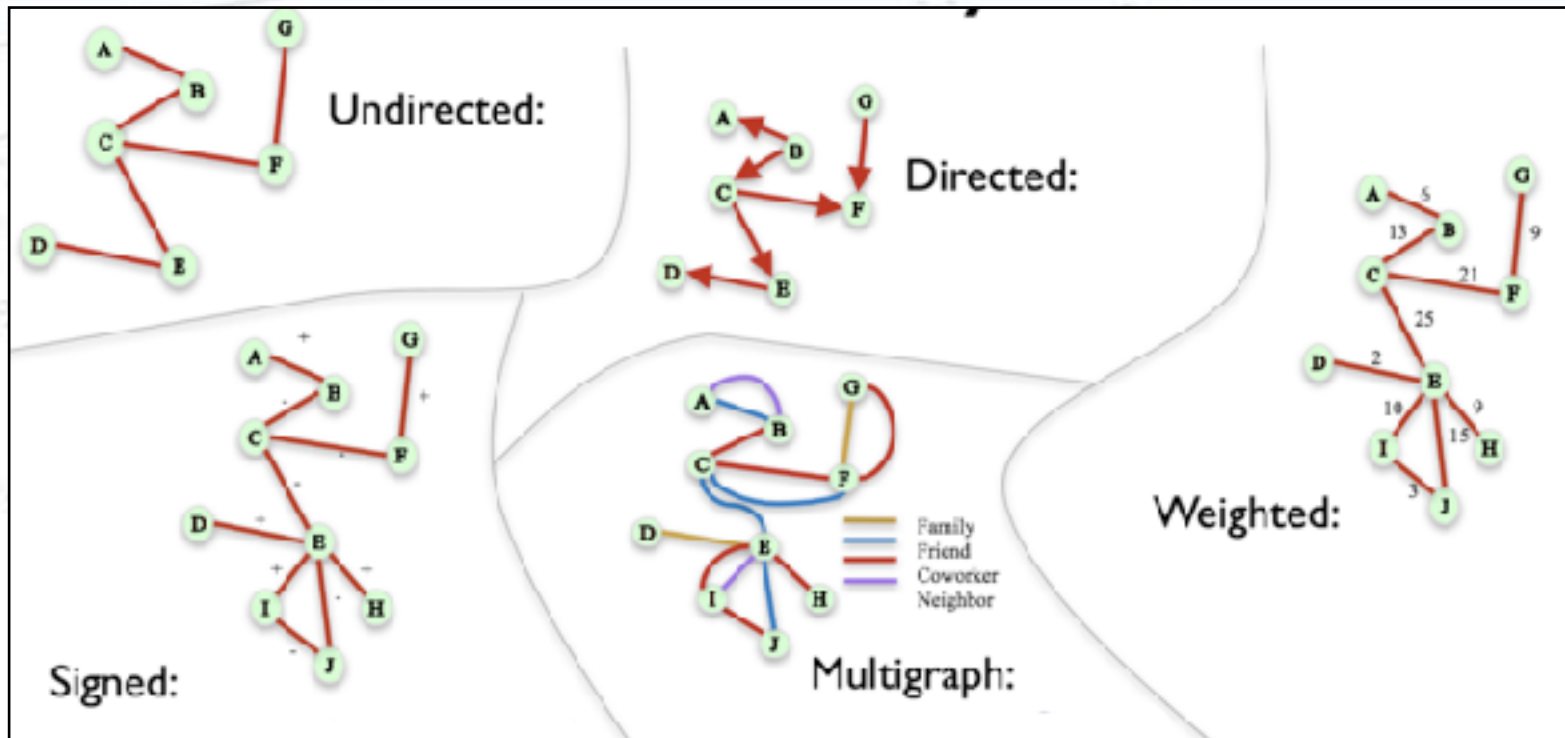
Graph Definition

A **graph** G is defined as a combined pair $G = \{V, E\}$ consisting of:

- **Nodes:** A set V , that typically contain input features (also called vertices)
- **Edges:** A set E of pair nodes, thus connecting the nodes (also called links)

In plain words:

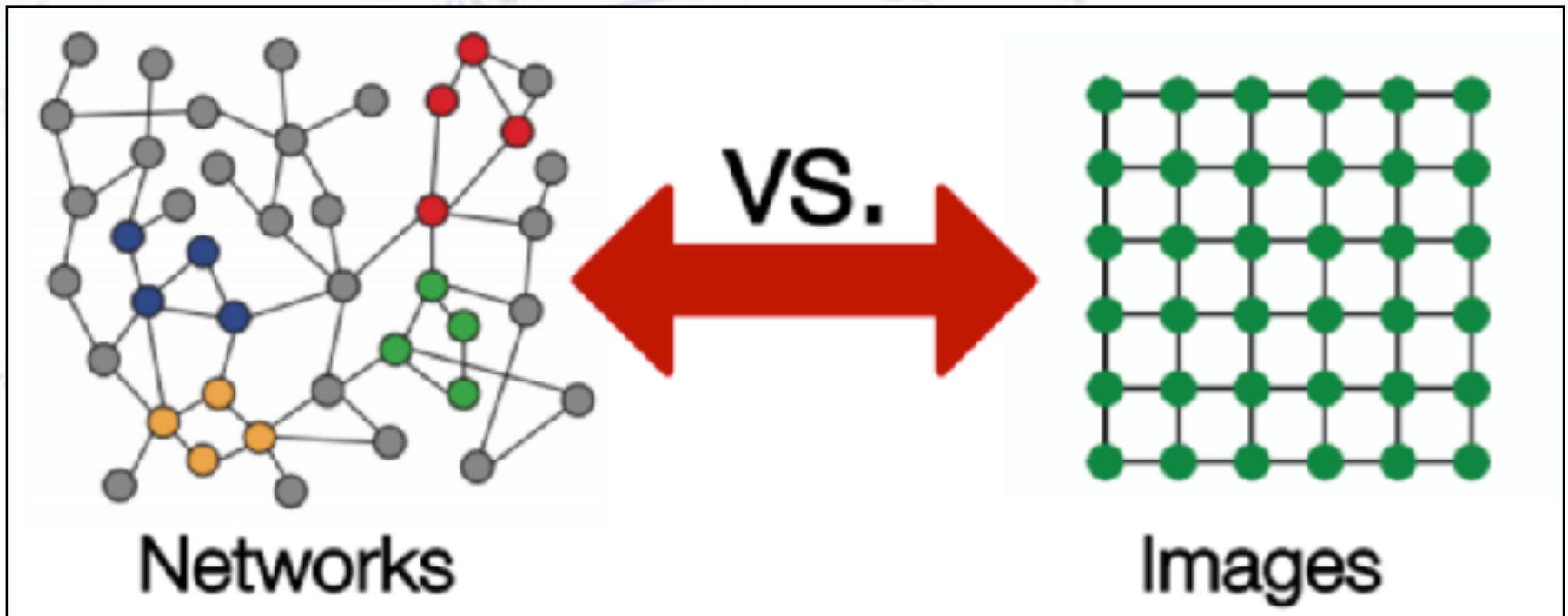
You got a list of points (nodes) that are somehow connected (with edges).



Geometrical data

Unlike e.g. images, **graphs have no underlying assumption on the geometry of the data.** This structure has to be specified by the user using the edges.

Many of the techniques in Machine Learning that you have been introduced to are also available for graphs (convolution, LSTM, Attention, Auto-Encoder, etc.)



Graph Convolution

The **graph convolution** proceeds much like for CNNs/images, as the **output is another graph**, possibly of different dimensionality. There are many different types of convolutions, **edgeconv** (<https://arxiv.org/abs/1801.07829>) considered below:

$$\tilde{x}_j = \sum_{i=1}^n f(x_j, x_j - x_i)$$

The “tilde” denotes the updated node

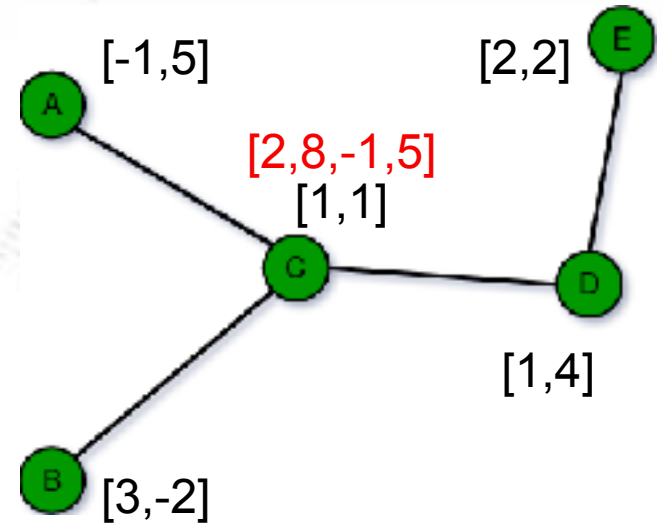
Graph Convolution

The **graph convolution** proceeds much like for CNNs/images, as the **output is another graph**, possibly of different dimensionality. There are many different types of convolutions, **edgeconv** (<https://arxiv.org/abs/1801.07829>) considered below:

$$\tilde{x}_j = \sum_{i=1}^n f(x_j, x_j - x_i)$$

The “tilde” denotes the updated node, and so if we applied the edgeconv operator with $f(\mathbf{x}) = \mathbf{1} * \mathbf{x} + \mathbf{0}$ to node D, we would obtain:

$$\begin{aligned}\tilde{x}_D &= f(x_D, x_D - x_C) + f(x_D, x_D - x_E) \\ &= f([1, 4], [1, 4] - [1, 1]) + f([1, 4], [1, 4] - [2, 2]) \\ &= f([1, 4], [0, 3]) + f([1, 4], [-1, 2]) \\ &= f([1, 4, 0, 3]) + f([1, 4, -1, 2]) \quad (\text{by concatenation}) \\ &= 1 \cdot [1, 4, 0, 3] + 1 \cdot [1, 4, -1, 2] \\ &= [2, 8, -1, 5]\end{aligned}$$



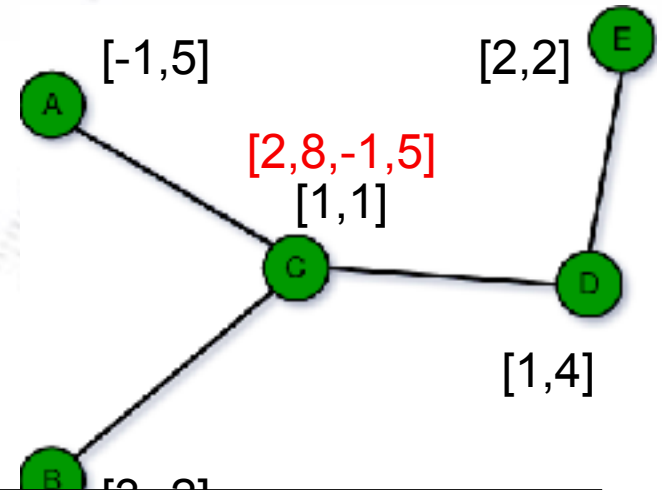
Graph Convolution

The **graph convolution** proceeds much like for CNNs/images, as the **output is another graph**, possibly of different dimensionality. There are many different types of convolutions, **edgeconv** (<https://arxiv.org/abs/1801.07829>) considered below:

$$\tilde{x}_j = \sum_{i=1}^n f(x_j, x_j - x_i)$$

The “tilde” denotes the updated node, and so if we applied the edgeconv operator with $f(\mathbf{x}) = \mathbf{1} * \mathbf{x} + \mathbf{0}$ to node D, we would obtain:

$$\begin{aligned}\tilde{x}_D &= f(x_D, x_D - x_C) + f(x_D, x_D - x_E) \\ &= f([1, 4], [1, 4] - [1, 1]) + f([1, 4], [1, 4] - [2, 2]) \\ &= f([1, 4], [0, 3]) + f([1, 4], [-1, 2]) \\ &= f([1, 4, 0, 3]) + f([1, 4, -1, 2]) \quad (\text{by concatenation})\end{aligned}$$

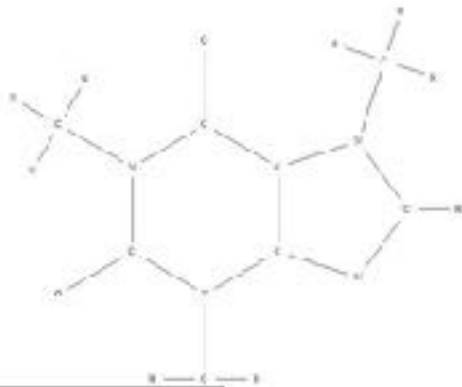


However...

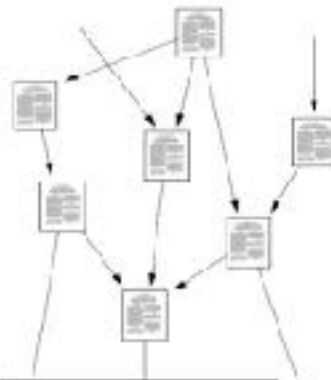
This only shows the start of a Graph Neural Network, not how to continue!

Example of Application

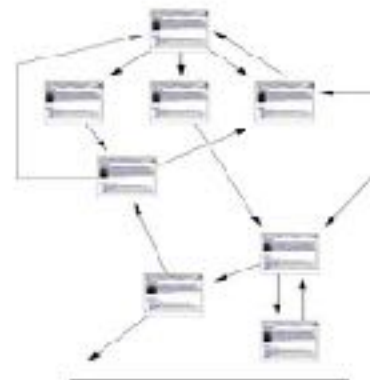
The graph examples/solutions are starting to enter the scene in many places:



Molecules



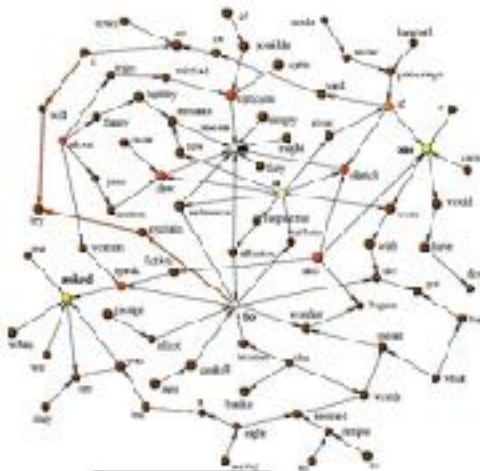
Knowledge



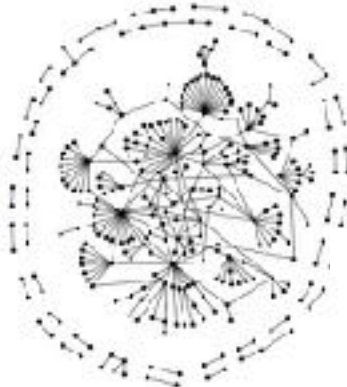
Information



Brain/neurons



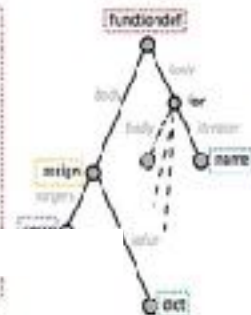
Genes



Communication

```
def encodeable():  
    """Returns a list of all the  
    dictionary objects in the value  
    list a tree structure is created  
    using this"""  
    l = []  
    for k, v in obj.items():  
        if isinstance(v, dict):  
            l.append({k: encodeable(v)})  
        elif isinstance(v, list):  
            l.append({k: [encodeable(i) for i in v]})  
    return l  
  
import sys  
tree = encodeable(sys.argv[1])
```

Software



Social

A background map of the Pacific Ocean, showing latitude and longitude lines. A red dot is placed at the coordinates 50°15'N, 150°15'W. The map includes labels for 'PACIFIC OCEAN' and '18° BITTER END TAOHT 1979'.

Notes on Transformers

Transformers

Transformers were developed to make Large Language Models work better.

The problem was the ability to “remember” what was important in earlier text...

...where to put the attention!

However, a powerful attention mechanism is useful everywhere.

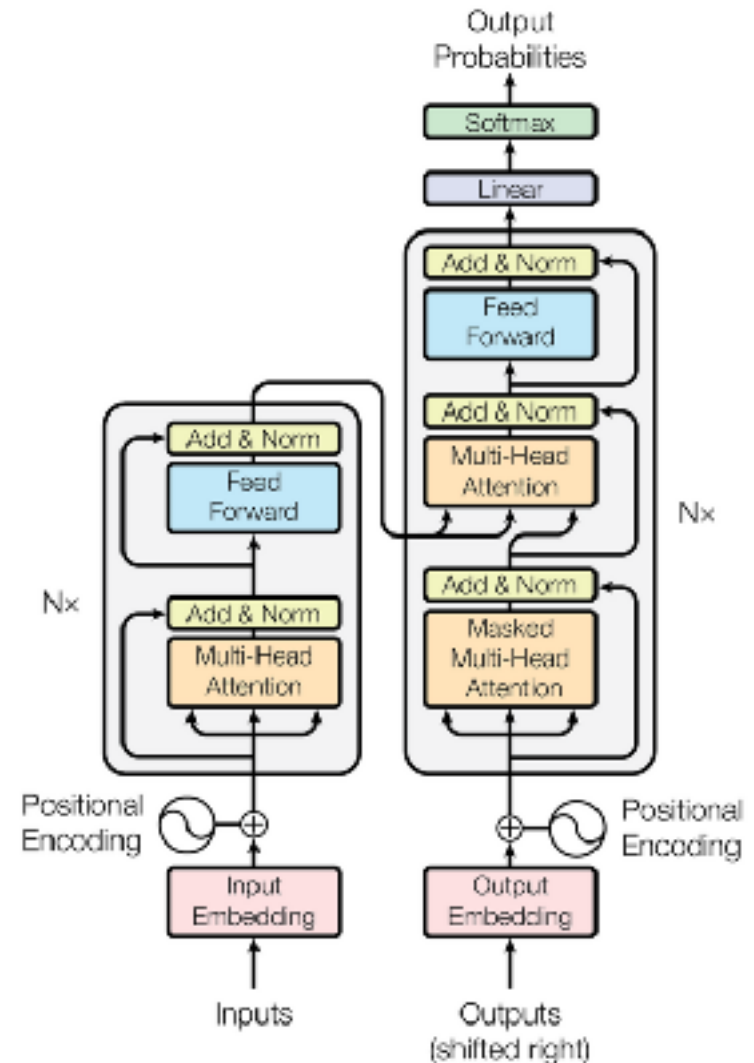
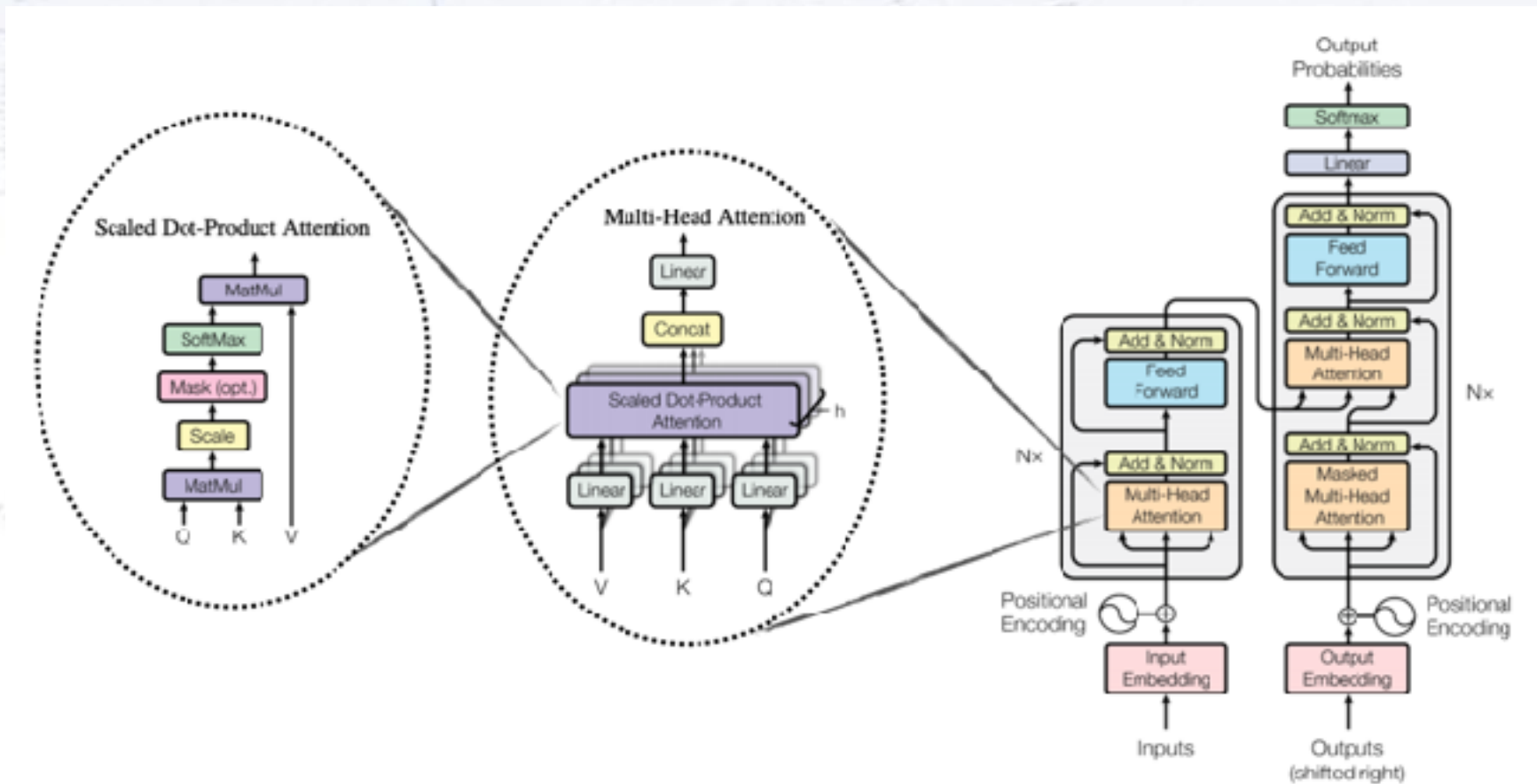


Figure 1: The Transformer - model architecture.

Transformers

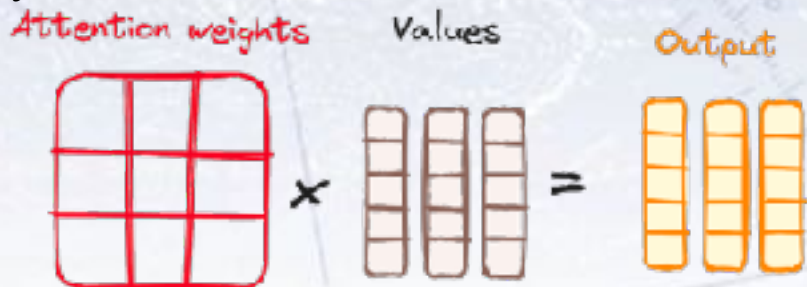
Transformers consist of a “Multi-Head Attention”, which again consists of (repeated) “Scaled Dot-Product Attention” between a Query (Q), Key (K), and Value (V).

Q, K, and V are simply three vectors from linear transformations of the input.

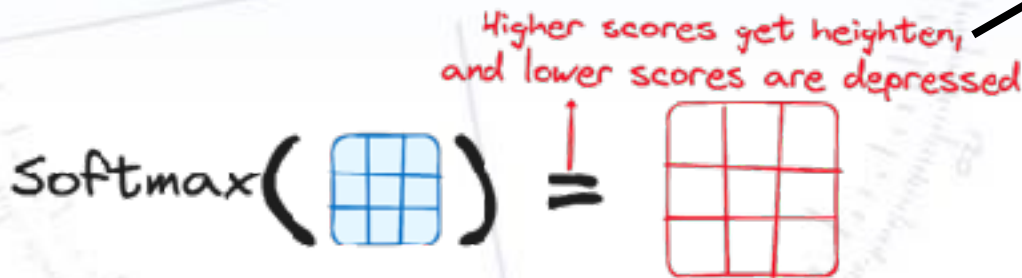


Scaled Dot-Product Attention

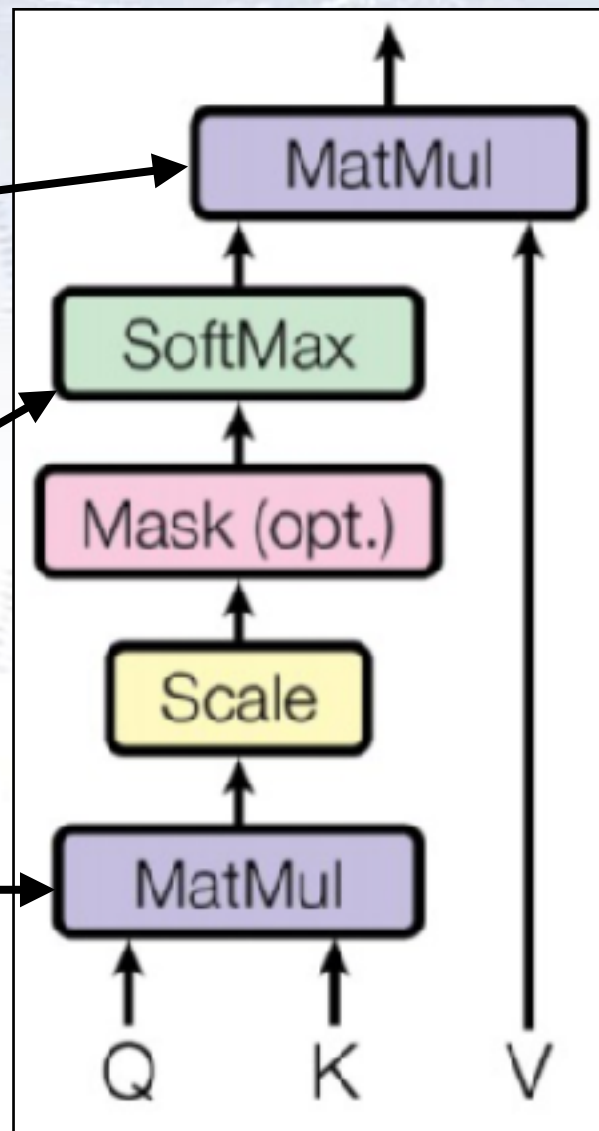
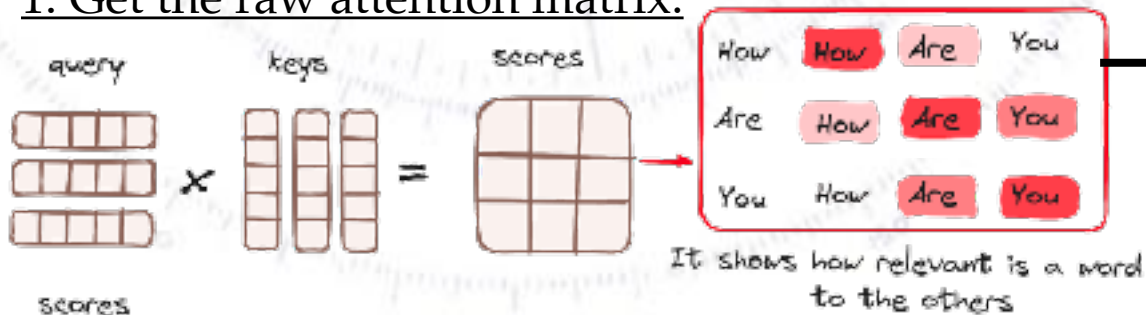
3. Apply the attention to the values:



2. Enhance the important relevances:

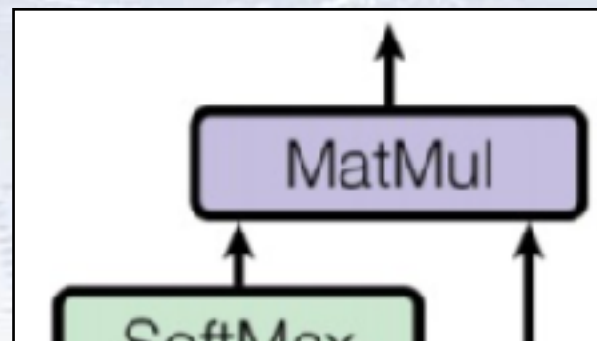
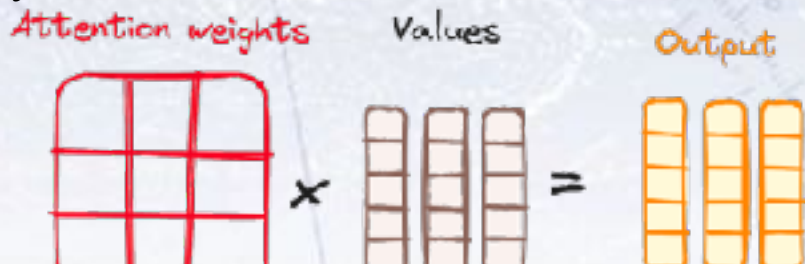


1. Get the raw attention matrix:



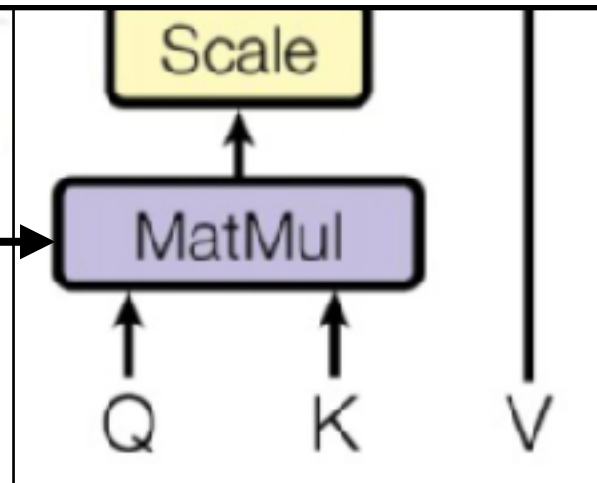
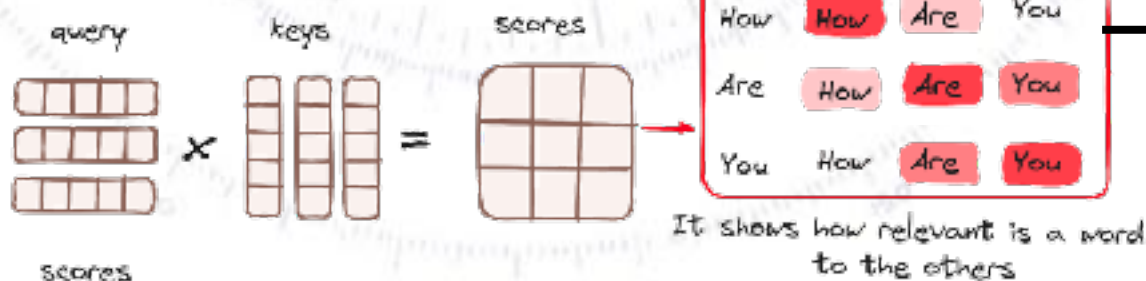
Scaled Dot-Product Attention

3. Apply the attention to the values:



$$\text{Attention}(Q, K, V) = \text{SoftMax} \left(\frac{QK^T}{\sqrt{d_K}} \right) V$$

1. Get the raw attention matrix:



GNN as a Transformer

As it happens, one can obtain a transformer with a very specific GNN:

- For each node, define three features: K , Q , and V .

- Define a message function f_{edge} on each edge:

$$m_{ij} = f_{\text{edge}}(K_i^k, Q^k, j) = \text{SoftMax} \left(\frac{QK^T}{\sqrt{d_K}} \right) V$$

- Define an aggregation function (weighted sum) around each node:

$$a_i^k = \sum_j m_{ij} V_j^k$$

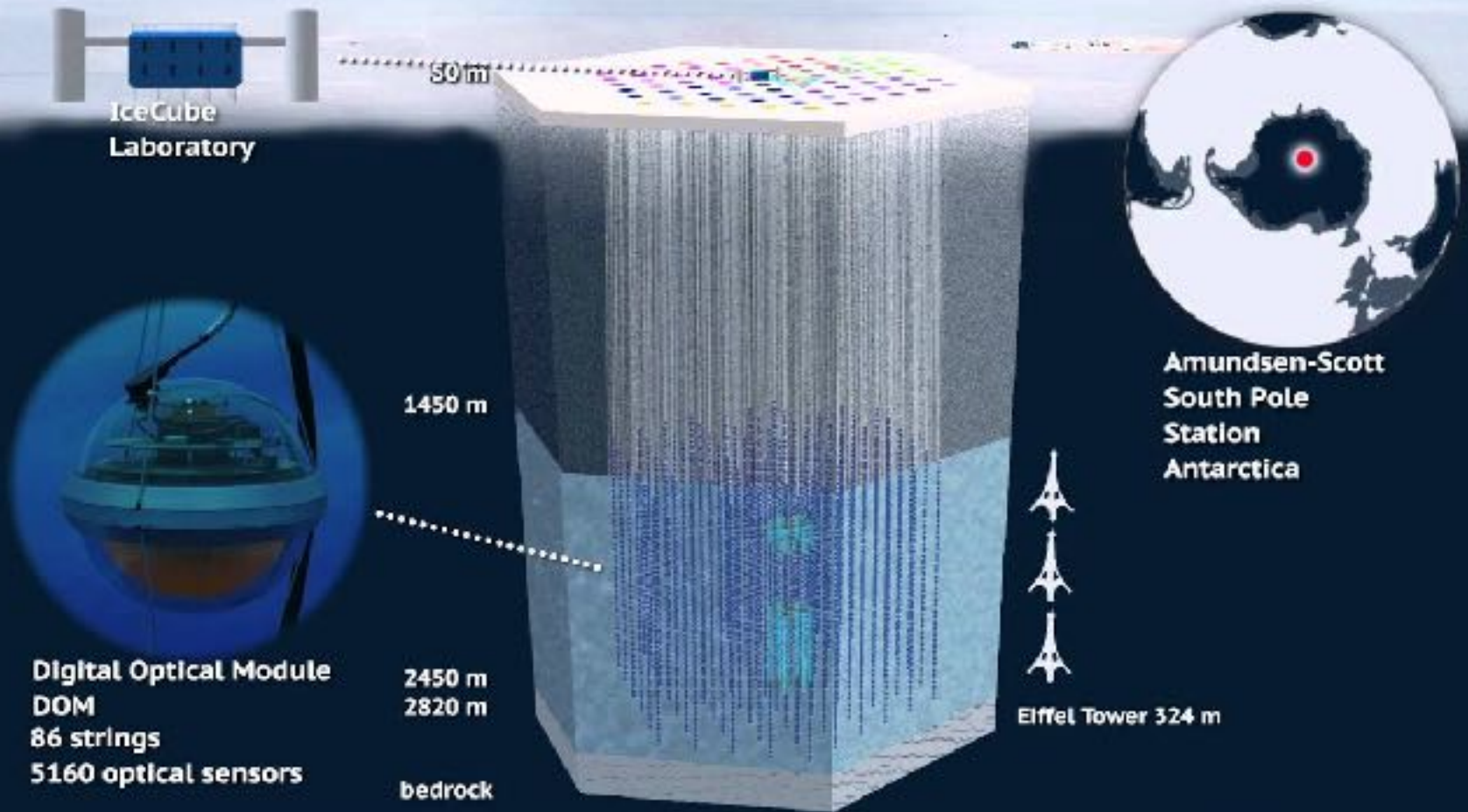
- The node update is then just an NN to get the next K' , Q' , and V' .

This is a Transformer!

A photograph of the IceCube counting house in Antarctica. The building is a multi-level structure with a complex network of metal stairs and walkways. It is situated on a flat, snow-covered landscape. The sky is a pale, hazy blue, suggesting a sunset or sunrise. In the foreground, there is a small, rounded mound of snow. The overall scene is quiet and desolate.

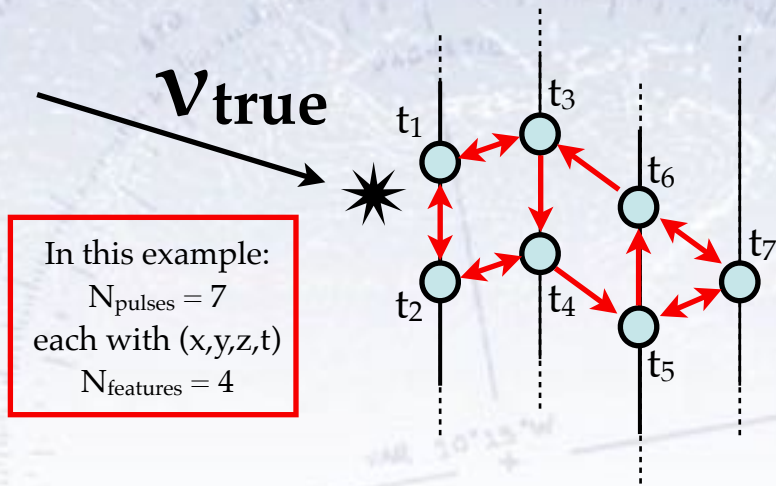
IceCube

IceCube experiment (South Pole)

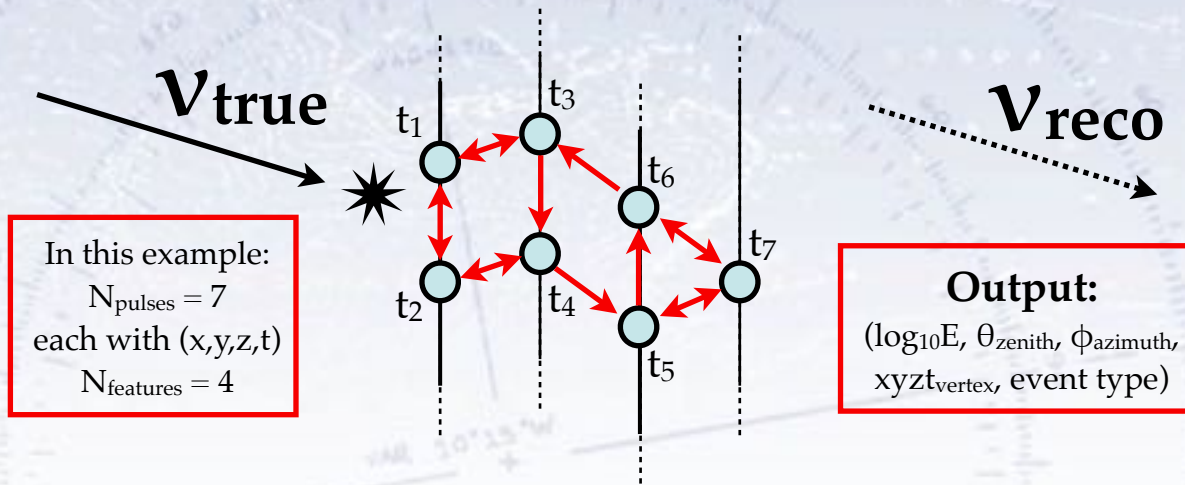


IceCube GNN model

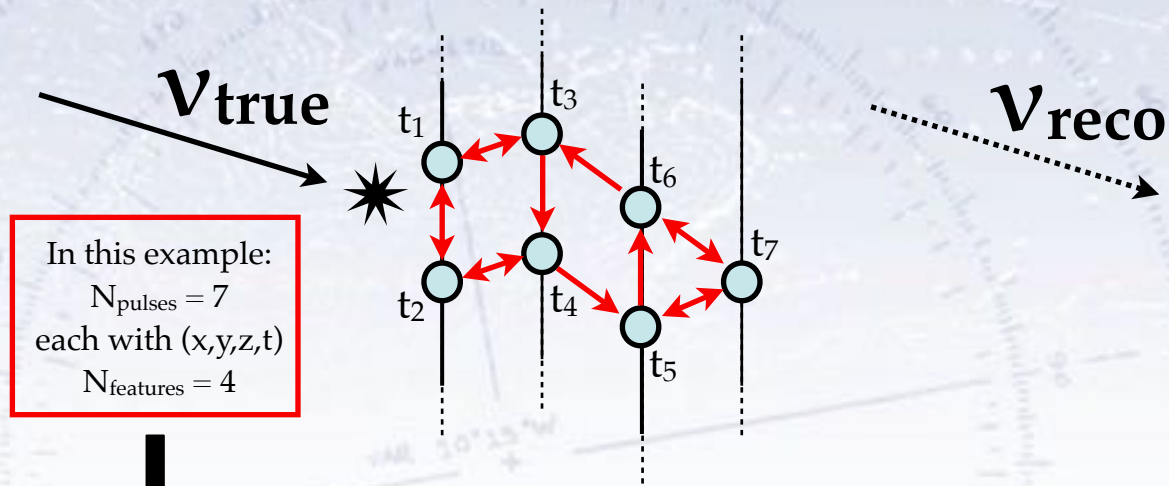
Details of GNN reconstruction



Details of GNN reconstruction



Details of GNN reconstruction



$$\vec{v}_1 = [x_1 \ y_1 \ z_1 \ t_1]$$

$$\vec{v}_2 = [x_2 \ y_2 \ z_2 \ t_2]$$

⋮

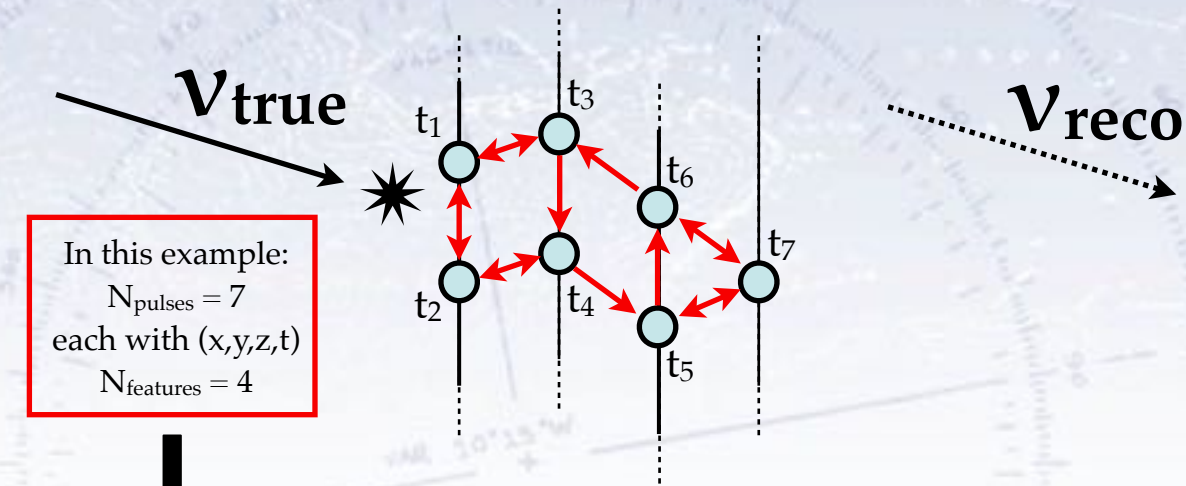
$$\vec{v}_7 = [x_7 \ y_7 \ z_7 \ t_7]$$

Input:

$$N = N_{\text{pulses}} \times N_{\text{features}}$$

The input features of a node are combined with that of $N (=2)$ nearby nodes

Details of GNN reconstruction



$$\begin{array}{l} \vec{v}_1 = [x_1 \ y_1 \ z_1 \ t_1] \\ \vec{v}_2 = [x_2 \ y_2 \ z_2 \ t_2] \\ \vdots \\ \vec{v}_7 = [x_7 \ y_7 \ z_7 \ t_7] \end{array} \xrightarrow{EC(\vec{v}_1, \vec{v}_2, \vec{v}_3)} \begin{array}{l} [g_{11} \ \dots \ g_{1N_1}] \\ [g_{21} \ \dots \ g_{2N_1}] \\ \vdots \\ [g_{71} \ \dots \ g_{7N_1}] \end{array}$$

Input:

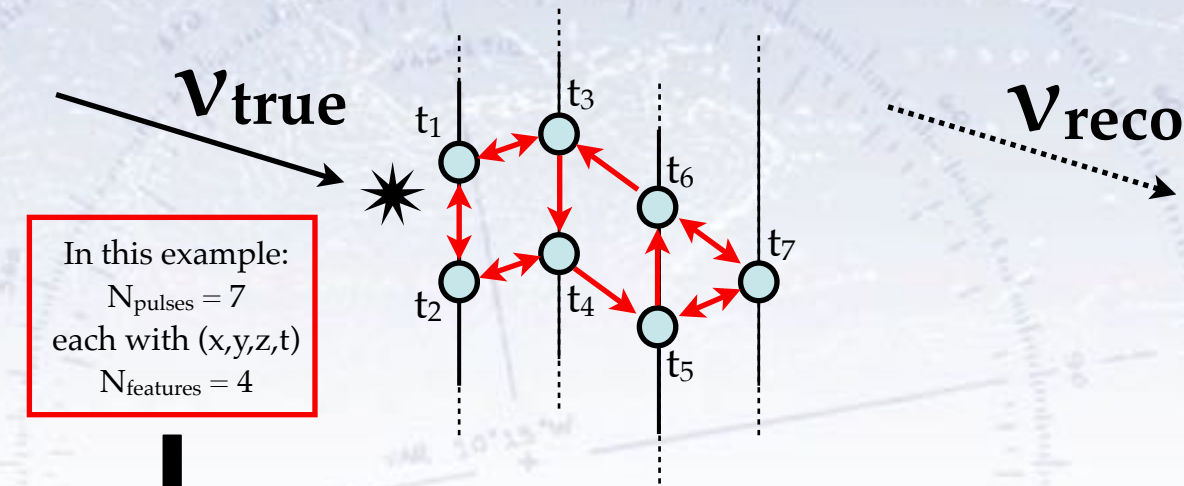
$$N = N_{\text{pulses}} \times N_{\text{features}}$$

Convolution(s):

$$N = N_{\text{pulses}} \times N_1$$

The input features of a node are combined with that of N ($=2$) nearby nodes through an NN (MLP0) function, yielding an (abstract) vector for each node. This can be repeated (not shown).

Details of GNN reconstruction



$$\begin{array}{ccc}
 \vec{v}_1 = [x_1 \ y_1 \ z_1 \ t_1] & \xrightarrow{EC(\vec{v}_1, \vec{v}_2, \vec{v}_3)} & [g_{11} \ \dots \ g_{1N_1}] & [x_1 \ y_1 \ z_1 \ t_1 \ g_{11} \ \dots \ g_{1N_1}] \\
 \vec{v}_2 = [x_2 \ y_2 \ z_2 \ t_2] & & [g_{21} \ \dots \ g_{2N_1}] & [x_2 \ y_2 \ z_2 \ t_2 \ g_{21} \ \dots \ g_{2N_1}] \\
 & \vdots & & \vdots \\
 & \xrightarrow{EC(\vec{v}_4, \vec{v}_5, \vec{v}_6)} & & \\
 & & & \vdots \\
 \vec{v}_7 = [x_7 \ y_7 \ z_7 \ t_7] & & [g_{71} \ \dots \ g_{7N_1}] & [x_7 \ y_7 \ z_7 \ t_7 \ g_{71} \ \dots \ g_{7N_1}]
 \end{array}$$

$$N_{\text{all}} = N_{\text{features}} + N_1$$

Input:

$$N = N_{\text{pulses}} \times N_{\text{features}}$$

Convolution(s):

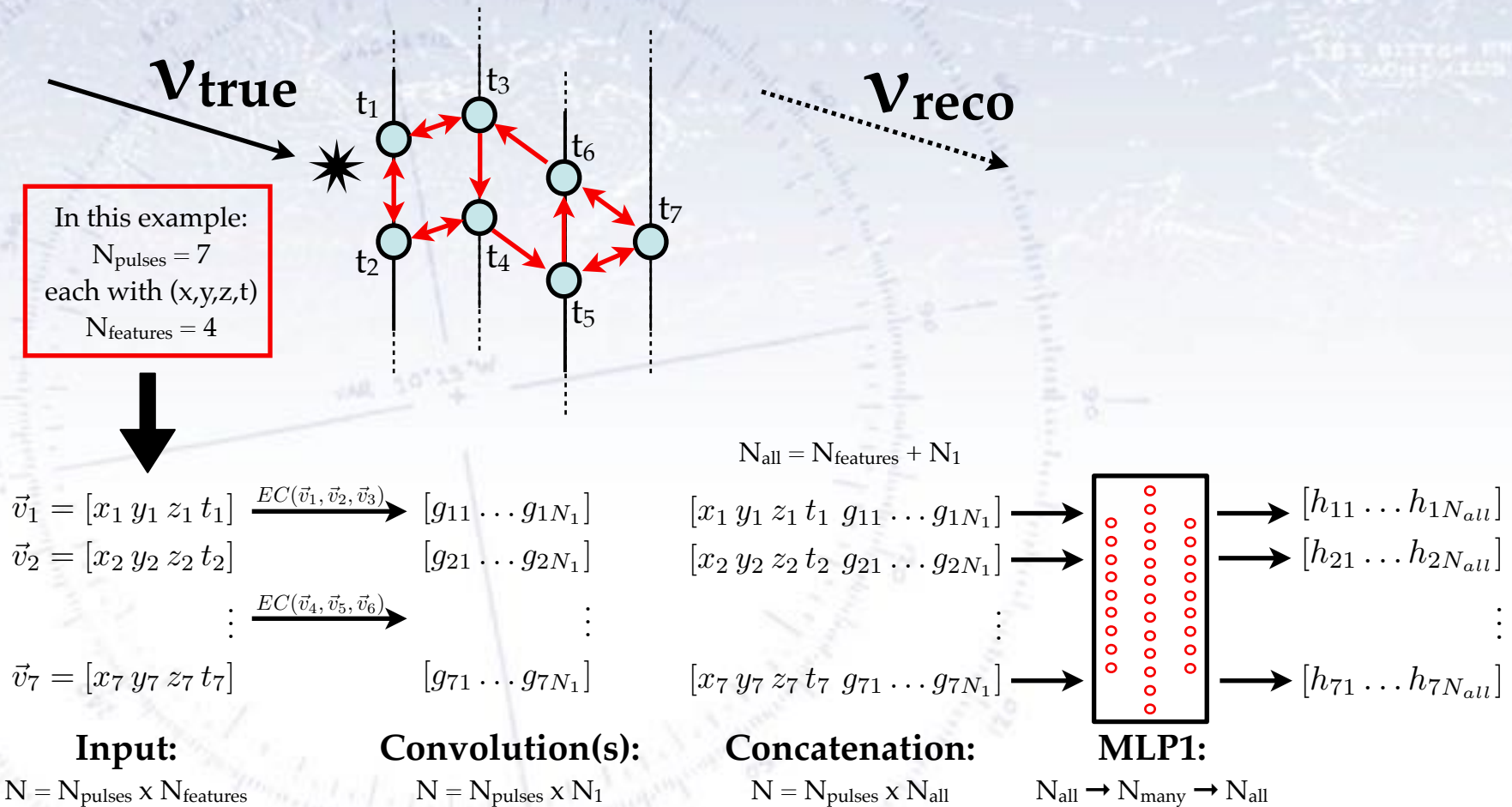
$$N = N_{\text{pulses}} \times N_1$$

Concatenation:

$$N = N_{\text{pulses}} \times N_{\text{all}}$$

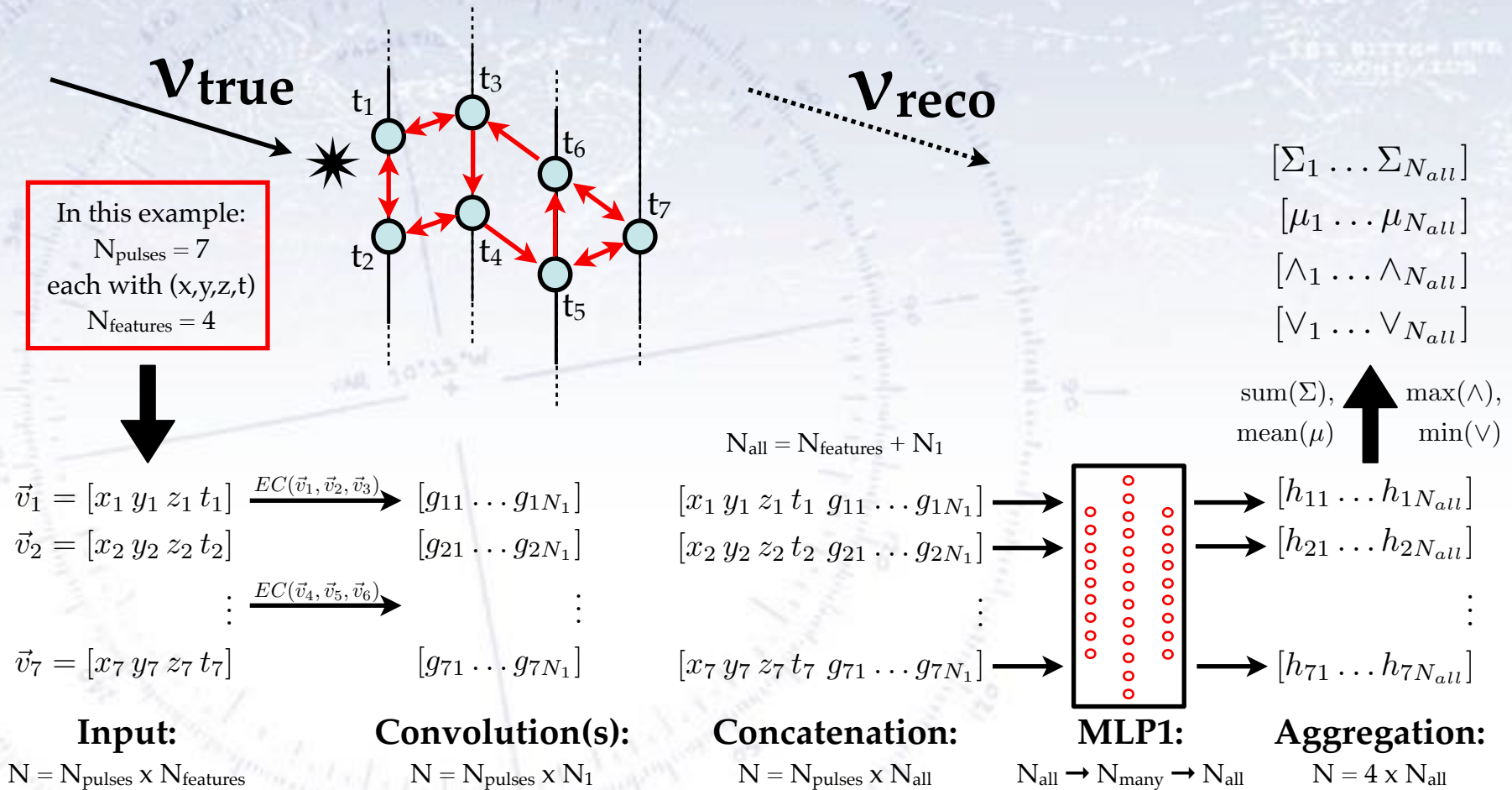
The input features of a node are combined with that of N ($=2$) nearby nodes through an NN (MLP0) function, yielding an (abstract) vector for each node. This can be repeated (not shown). All the features are then combined (concatenated) into long vectors,

Details of GNN reconstruction



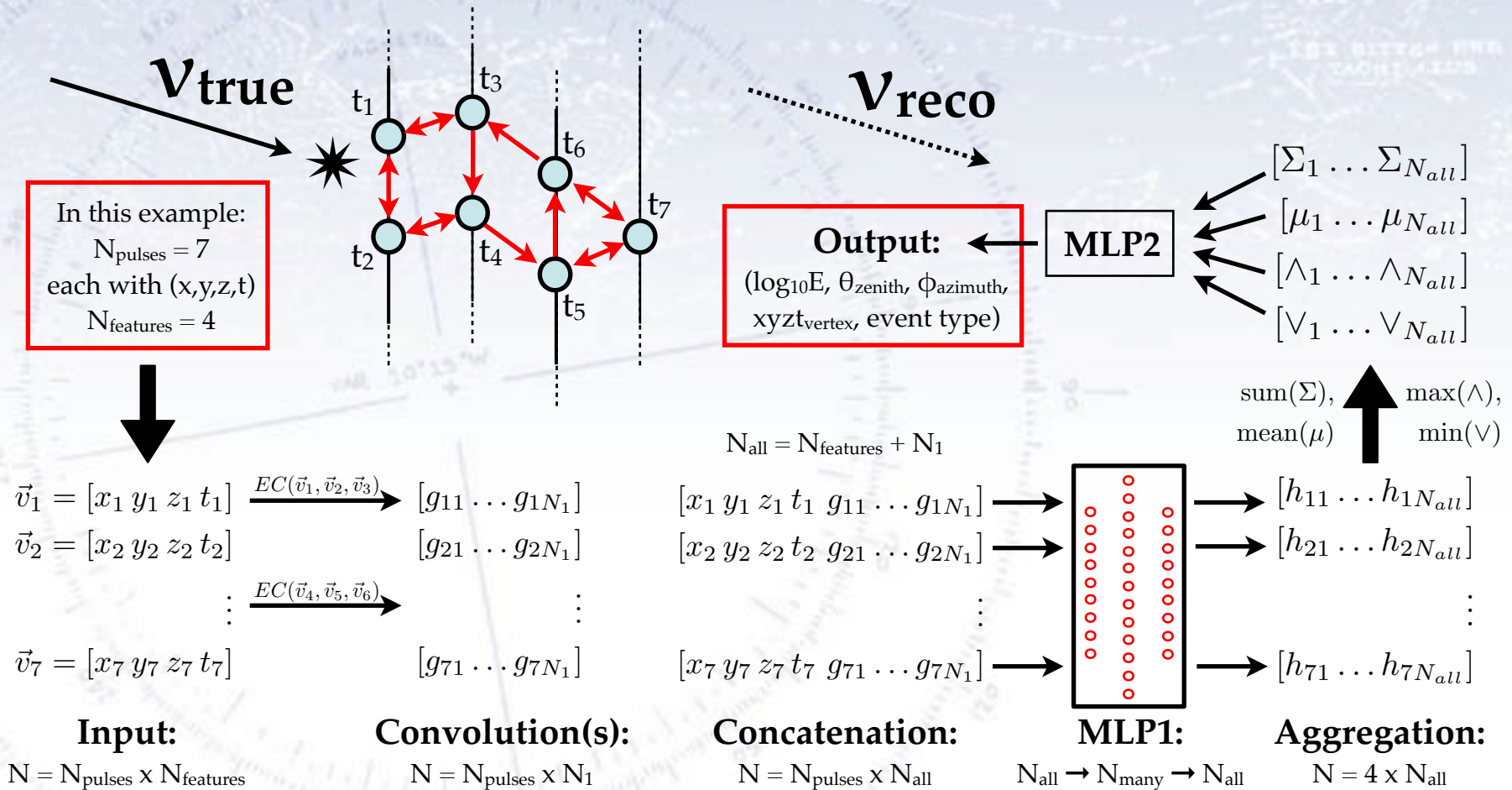
The input features of a node are combined with that of N ($=2$) nearby nodes through an NN (MLP0) function, yielding an (abstract) vector for each node. This can be repeated (not shown). All the features are then combined (concatenated) into long vectors, which are again put through an NN (MLP1) function with a large hidden layer.

Details of GNN reconstruction



The input features of a node are combined with that of N ($=2$) nearby nodes through an NN (MLP0) function, yielding an (abstract) vector for each node. This can be repeated (not shown). All the features are then combined (concatenated) into long vectors, which are again put through an NN (MLP1) function with a large hidden layer. The outputs are aggregated in four ways: Min, Max, Sum & Mean, breaking the variation with number of nodes.

Details of GNN reconstruction



The input features of a node are combined with that of N ($=2$) nearby nodes through an NN (MLP0) function, yielding an (abstract) vector for each node. This can be repeated (not shown). All the features from all the convolutions are then combined (concatenated) into long vectors, which are again put through an NN (MLP1) function with a large hidden layer. The outputs are aggregated in four ways: Min, Max, Sum & Mean, breaking the variation with number of nodes. These are then fed into a final NN (MLP2), which outputs the estimated type(s) and parameters of the event.

Further specifics of DynEdge

In DynEdge, there are several “enlargements” compared to the previous illustration of the GNN architecture. These are essentially:

- We use 6 input features: x , y , z , t , charge, and Quantum Efficiency.
- We convolute each node with the nearest 8 nodes (not two).
- We do 4 (not 1) convolutions, each with 192 inputs and outputs.
- The concatenation is of all convolution layers and the original input.
- In the results to be shown, we have trained separate GNNs for each output.

The repeated convolutions allows all signal parts to be connected.

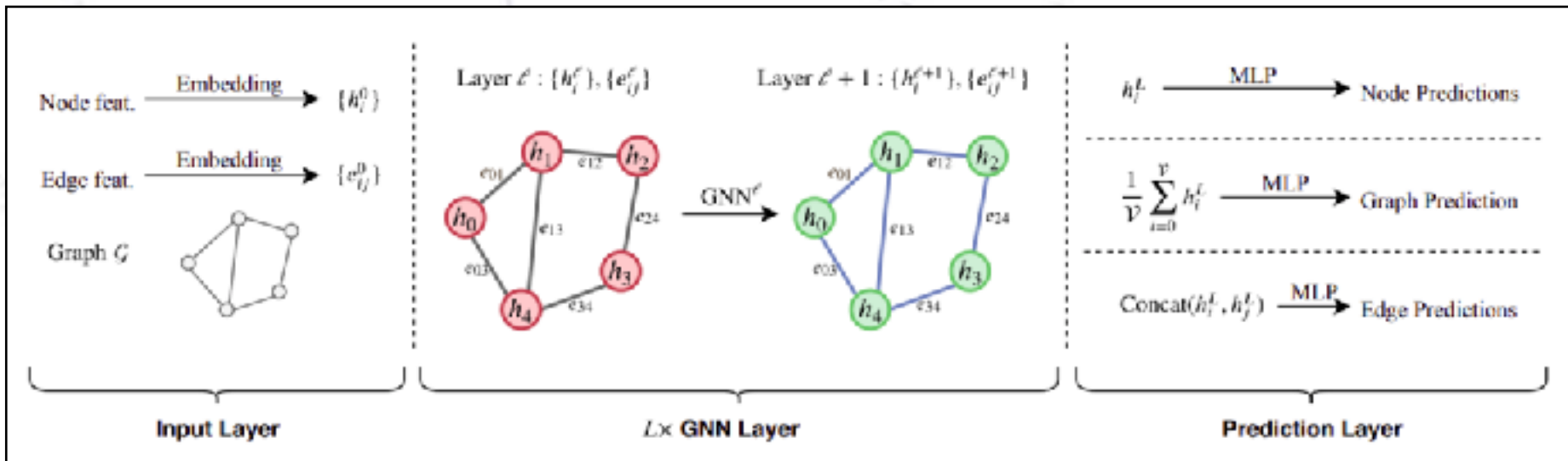
The EdgeConv convolution operator ensures permutation invariance.

The number of model parameters is about 750.000 for the angular regressions, while the energy only requires 150.000. In principle one can go down to 70.000 parameters, but there is no reason for this - it is already a “small” ML model.

What can GNN predict?

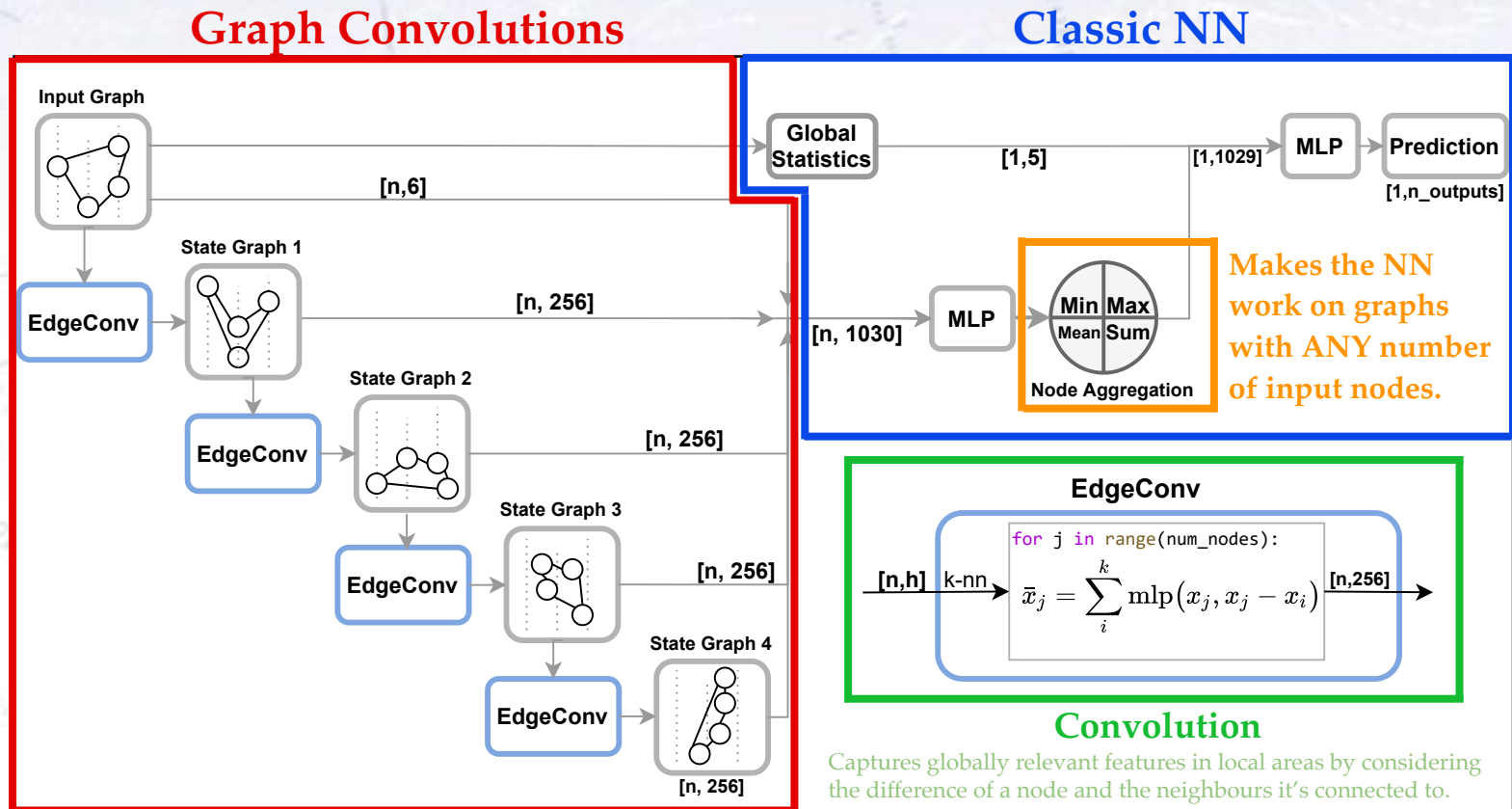
GNNs are capable to make three sorts of predictions:

- Node prediction (is this node signal or noise?)
Obtained simply from an MLP on the convoluted node features.
- Edge prediction (is this edge important or not?)
Obtained from an MLP on (concatenated) pairs of node features.
- Graph prediction (is this graph an X or not? What is the Y of this graph?)
Obtained through an MLP on a summary of the graph nodes.
Here, there are several options of dimensionality and aggregation.



GraphNet

The GNN model is outlined below, which is also the figure for our IceCube GNN paper (<https://arxiv.org/abs/2209.03042>).





Dreaming

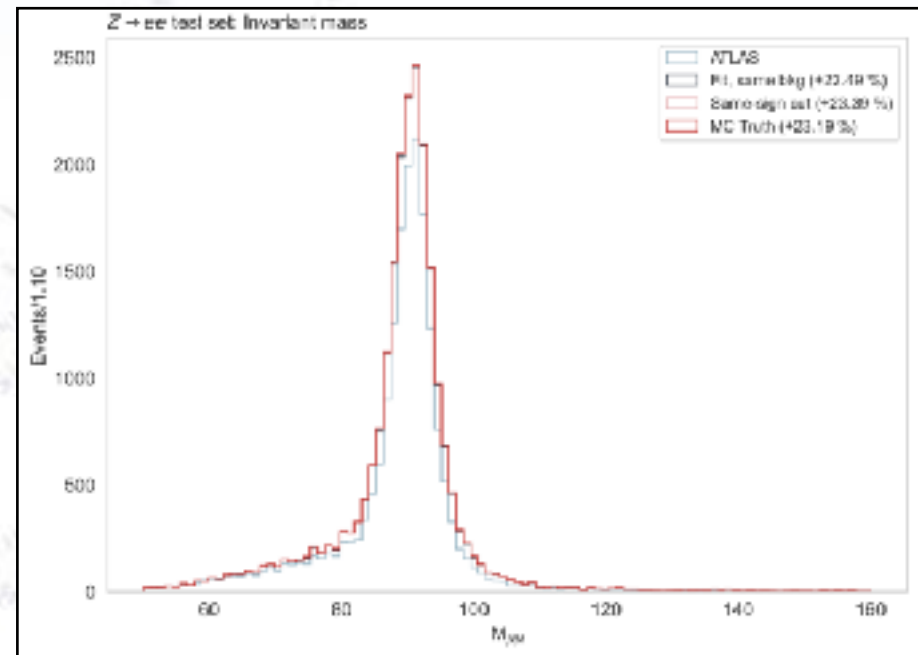
Imagine 1B Z-decays in data+MC

ATLAS and CMS each has about 1B Zee and $Z\mu\mu$ decays in data+MC in total. Each Z has 100+ features, and the resulting performance can be measured accurately in both data and MC (the Z mass peak).

Such a dataset could become a standard reference in ML for testing new architectures, training schemes, parallelisation methods, etc.

It could also be the training ground for **solving the problem of domain shifts**, thereby helping 1000s of particle physicists... and many more ML users in other science fields in the future.

Applying what is learned to the $Z \rightarrow \tau\tau$ decay, and then in turn to $H \rightarrow \tau\tau$ decay to improve the di-Higgs searches.





GraphNeT

Graph Neural Networks for
Neutrino Telescope Event Reconstruction

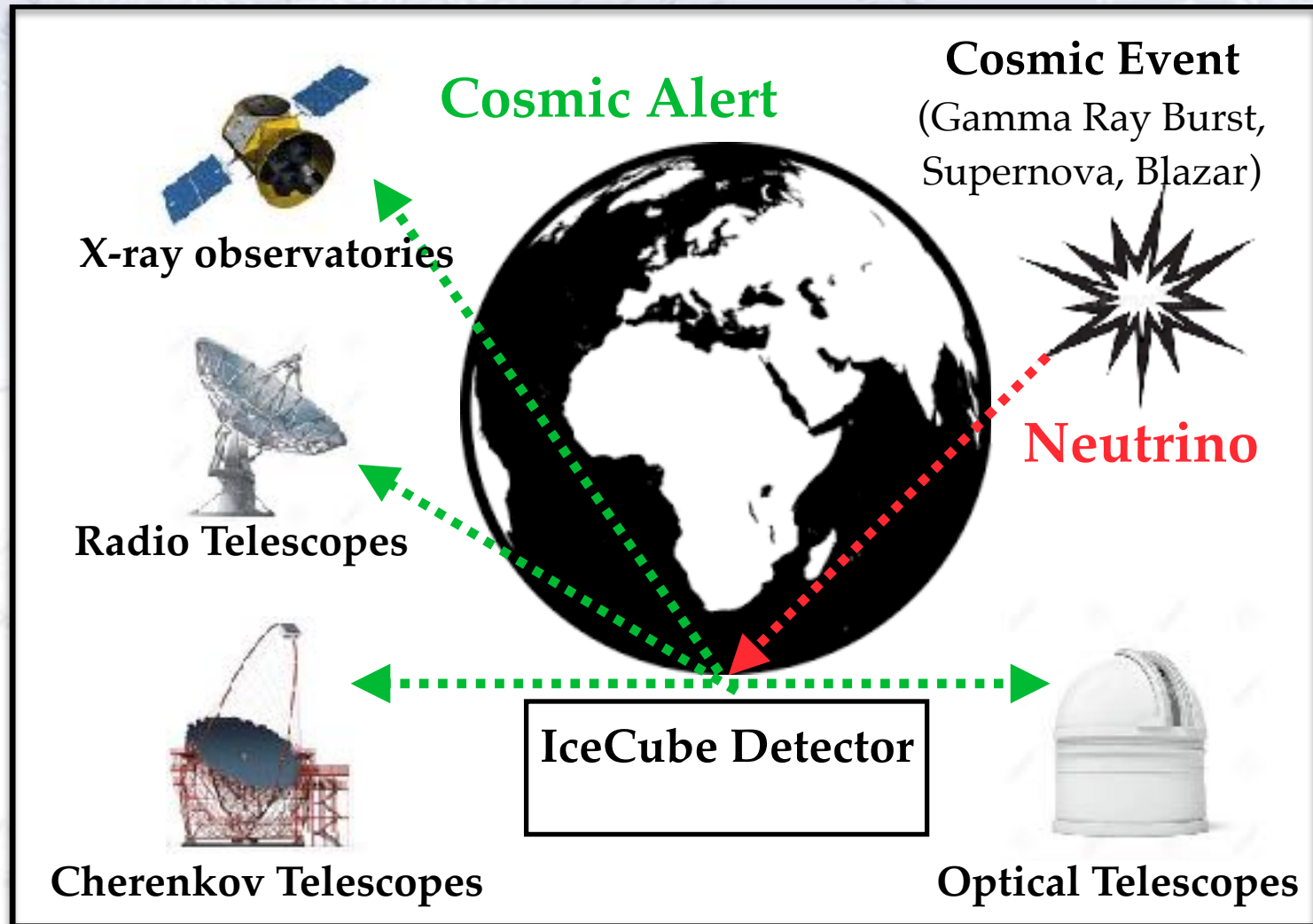
GraphNet is our attempt at putting GNN models for IceCube (and others) using the “DynEdge” architecture build in PyTorch Geometric into an easily available software package.

<https://github.com/graphnet-team/graphnet>

We are writing our results up in an IceCube paper (responded to several rounds of feedback and comments).

The IceCube challenge was also made into a Kaggle competition - .

Seeing the Universe in ν light

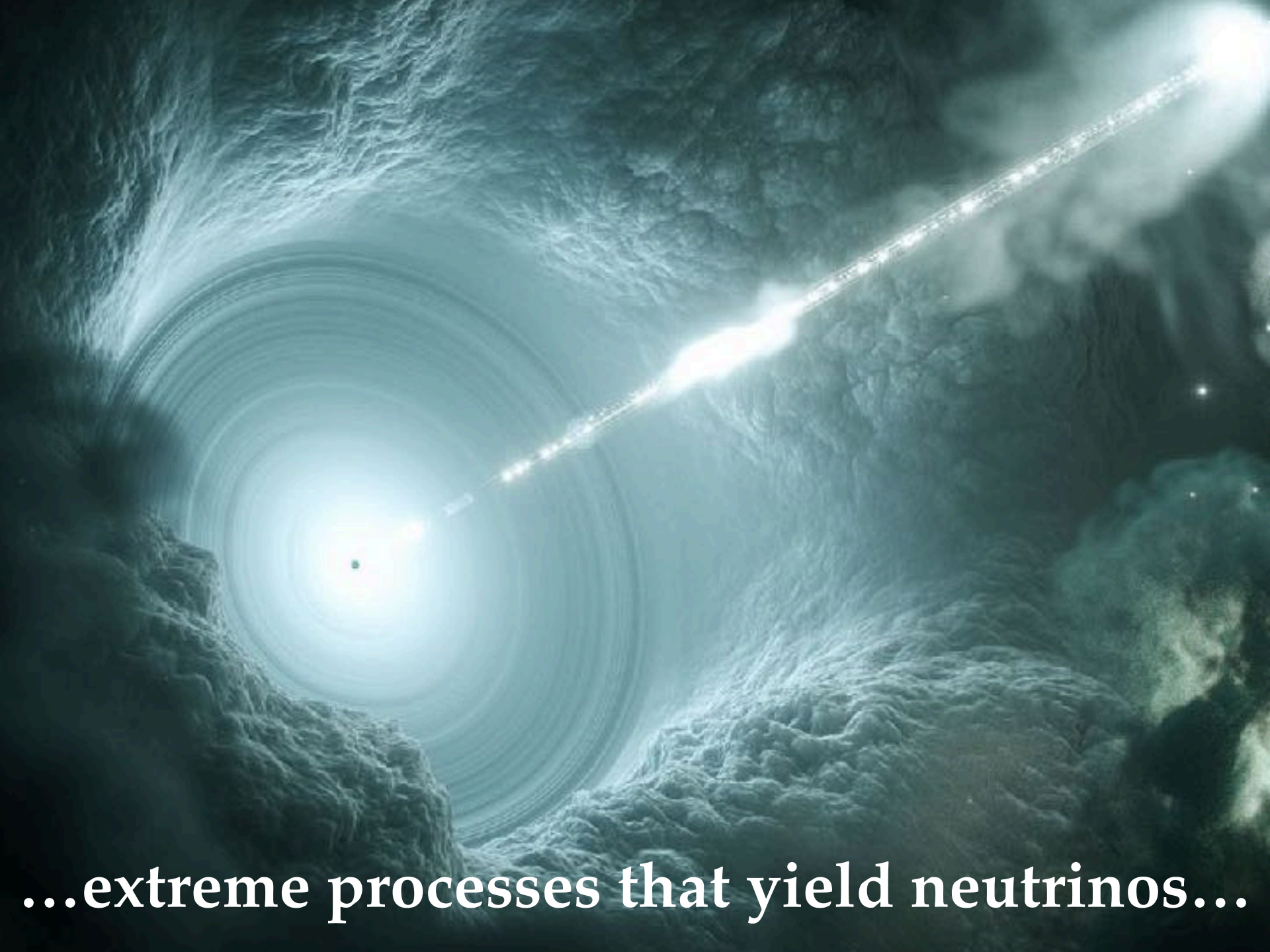


Seeing the Universe with neutrinos



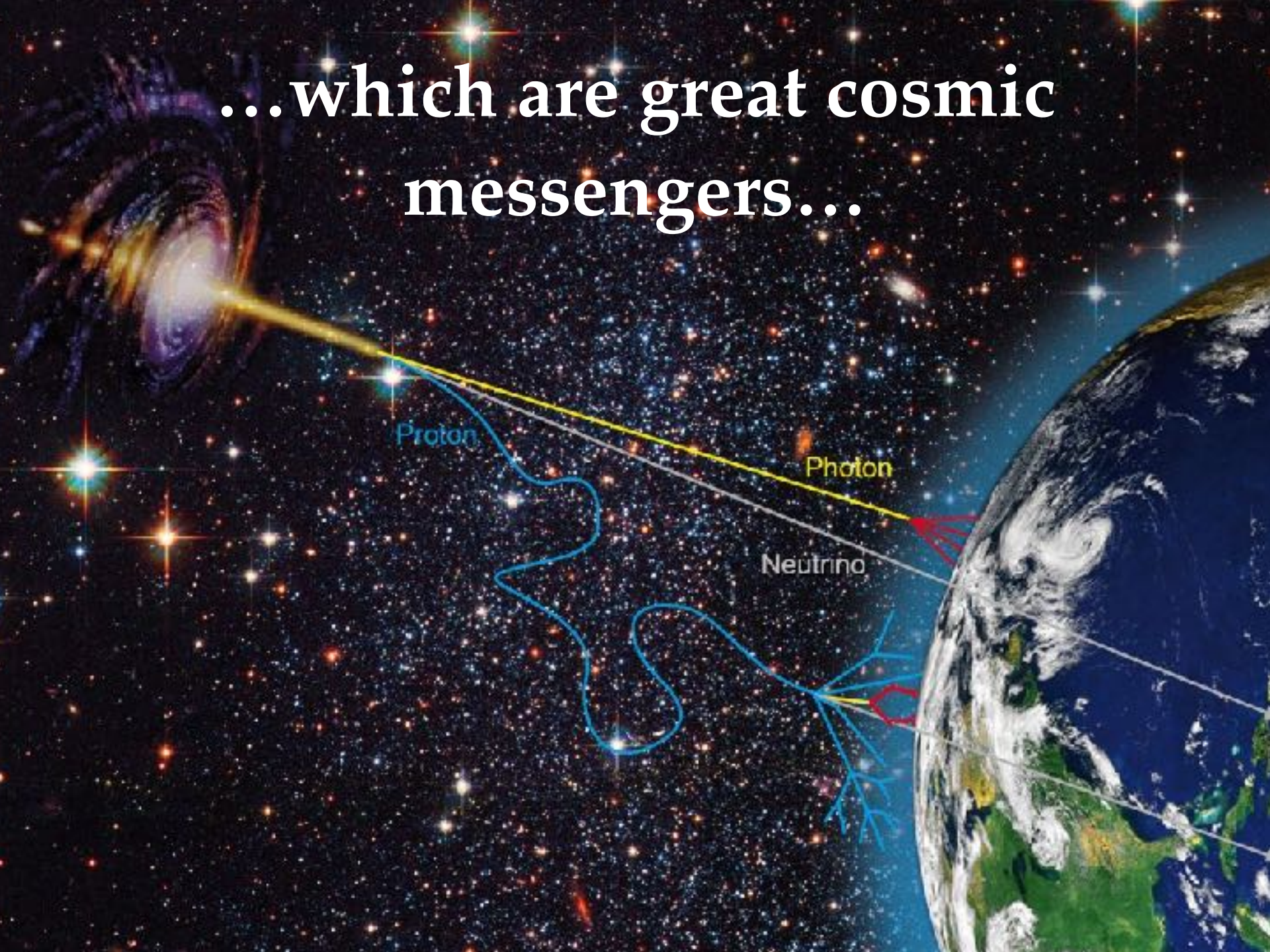
The image shows the IceCube Neutrino Telescope facility on a large ice floe. In the background, a central control building is supported by two tall, cylindrical pillars. The foreground is dominated by numerous vertical strings of colorful detector modules, which are part of the IceCube array. The sky is a pale, hazy blue, and the ice floe is a mix of white and light blue. The text "IceCube Neutrino Telescope can see..." is overlaid in the center of the image.

IceCube Neutrino Telescope can see...

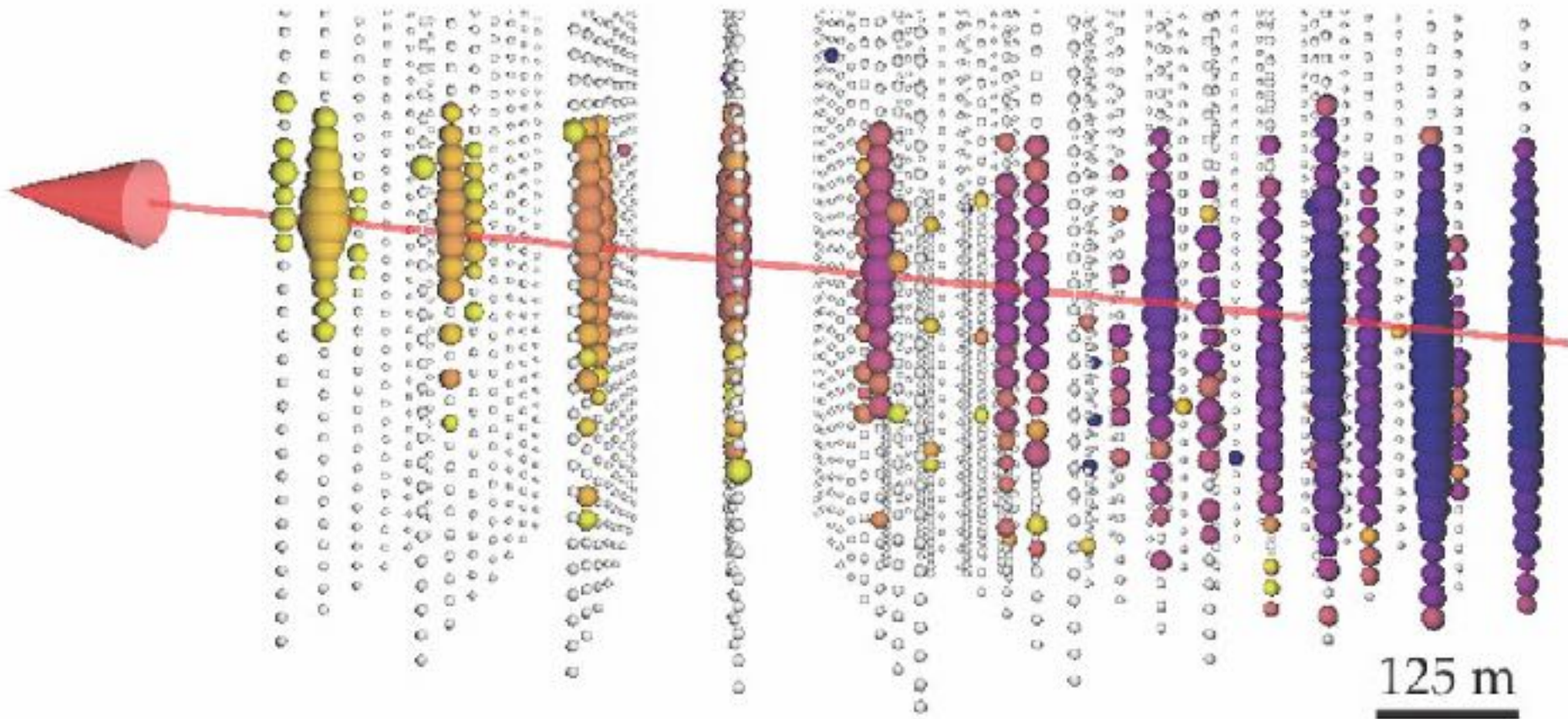


...extreme processes that yield neutrinos...

...which are great cosmic messengers...



...that can be observed by
IceCube ν -Telescope...





GraphNeT

Graph Neural Networks for
Neutrino Telescope Event Reconstruction

...in real time using
Graph Neural Networks



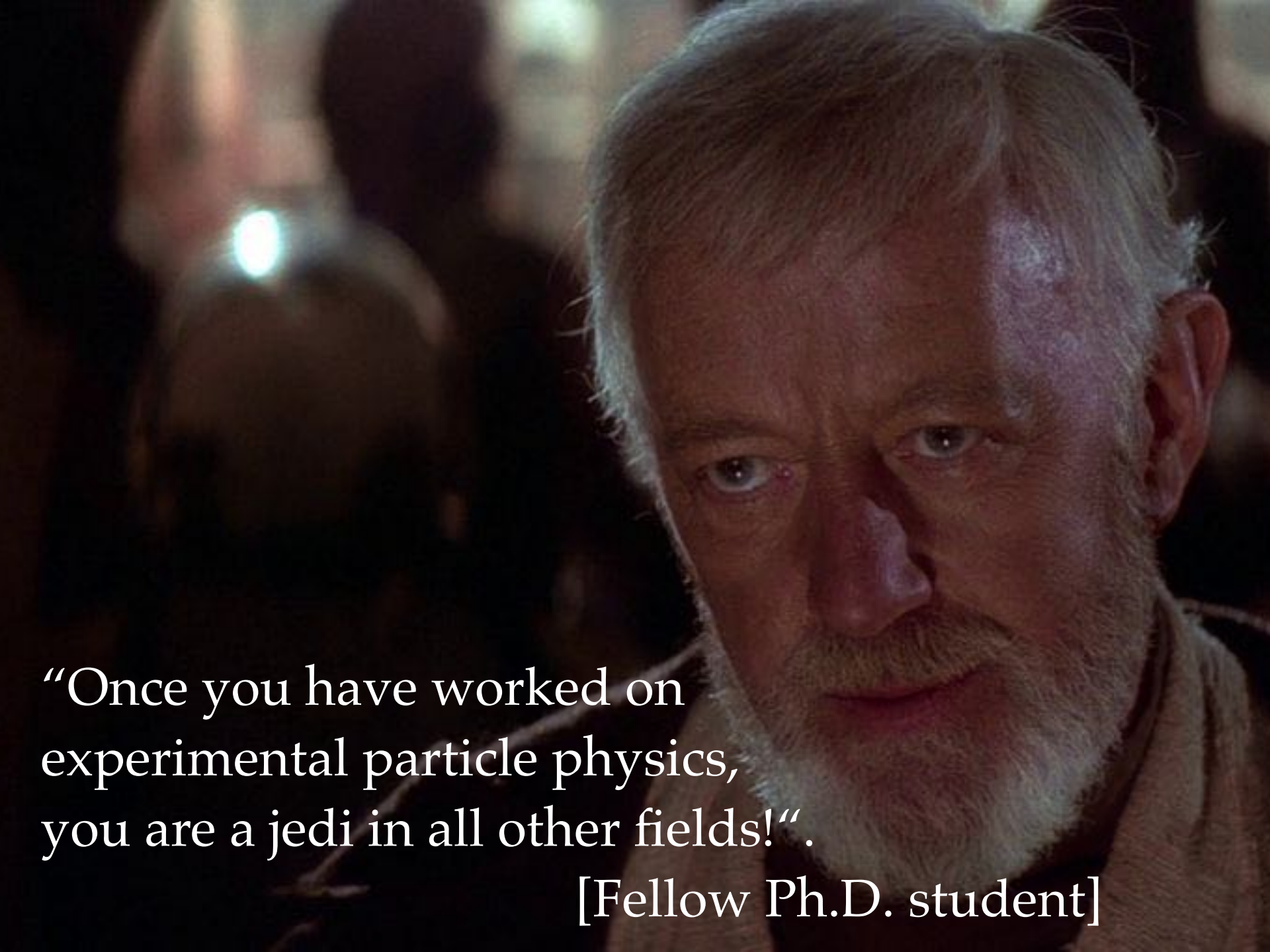
GraphNeT

Graph Neural Networks for
Neutrino Telescope Event Reconstruction

Despite having a very complicated geometry and a great variation in number of signal inputs, new ML methods - Graph Neural Networks - are capable of handling this... very well!

$$x^{\text{update}} = \sum_i^{\text{Neighbours}} NN(x, x_i)$$

...in real time using
Graph Neural Networks



“Once you have worked on experimental particle physics, you are a jedi in all other fields!”.

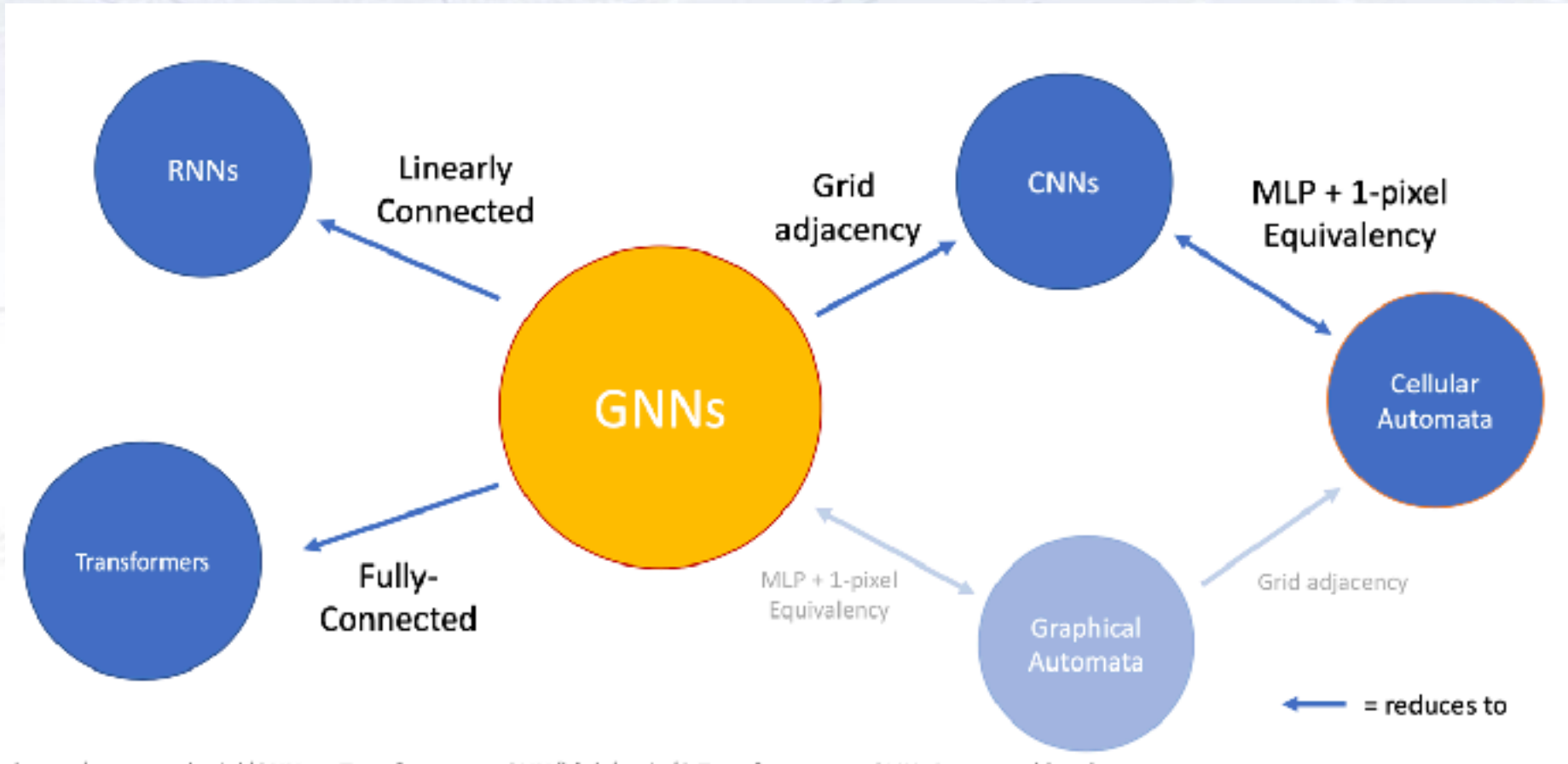
[Fellow Ph.D. student]



Bonus slides

Connection between NNs

The different NN architectures are related, as some paradigms reduces to others. This is illustrated below.



Scaled Dot-Product Attention

The “interesting” attention mechanism from transformers that is relevant for typical particle physics analysis is summarised in this single term:

$$\text{SoftMax} \left(\frac{QK^T}{\sqrt{d_K}} \right) V$$