# STATISTICS AND MACHINE LEARNING 3

Harrison B. Prosper

Florida State University

## Topics

- Lecture 1
  - Frequentist Analysis (1)
- Lecture 2
  - Frequentist Analysis (2)
  - Bayesian Analysis
- **Lectures 3**
  - Introduction to Machine Learning
    - Foundations
    - Models

## Jupyter Notebooks

I encourage you to try out the jupyter notebooks at

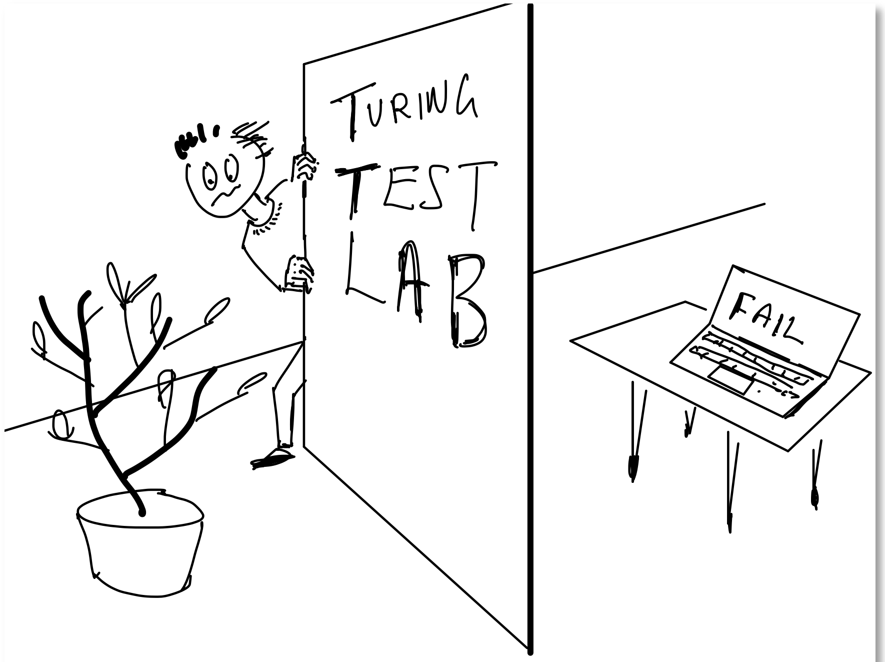
https://github.com/hbprosper/AEPSHEP

Also: https://github.com/hbprosper/GSW

**Recommendation** (for Windows, Linux and OSX)

1. Install miniconda. See instructions at:
   https://docs.conda.io/en/latest/miniconda.html
2. Create a miniconda environment
   ```
   conda create --name aepshep
   ```
3. Activate environment
   ```
   conda activate aepshep
   ```

## FOUNDATIONS



"George rethinks his life after failing the Turing Test"

## What is Machine Learning?

The art and science of creating statistical *models* $f(x, \omega) \in F$ of data by minimizing a quantity called the *average loss,* or *empirical risk*,

**$F$** = Function class

$$R(\omega) = \frac{1}{N}\sum_{i=1}^{N} L(t_i, f_i)$$

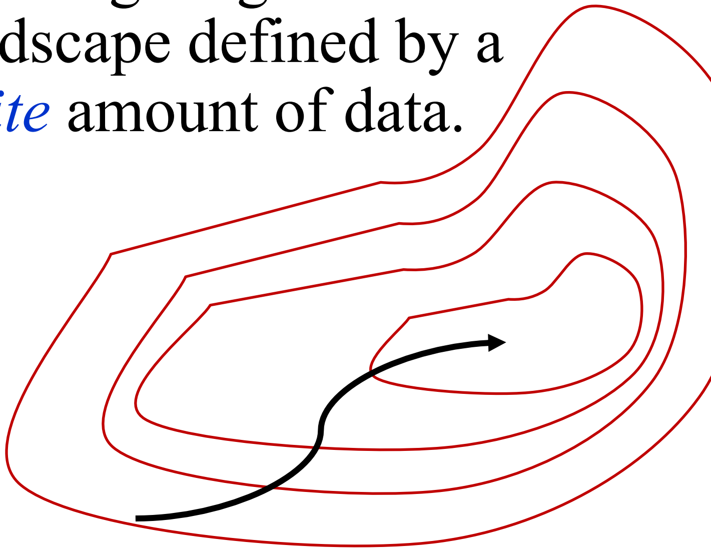
where

| | |
|---|---|
| $T = \{(t_i, x_i)\}$ | are training data (*targets*, *inputs*), |
| $f_i$ | is the model $f(x, \omega)$ evaluated at $x_i$, and |
| $L(t_i, f_i)$, | is the *loss function*, a measure of the loss incurred by choosing a function from **$F$**. |

## Minimizing the Average Loss

The average loss, $R(\omega)$, defines a "landscape" in the *parameter space* of the model $f(x, \omega) \in F$.

The Goal: find the lowest point in the landscape defined by an *infinite* amount of data by navigating the landscape defined by a *finite* amount of data.



## Minimizing the Average Loss

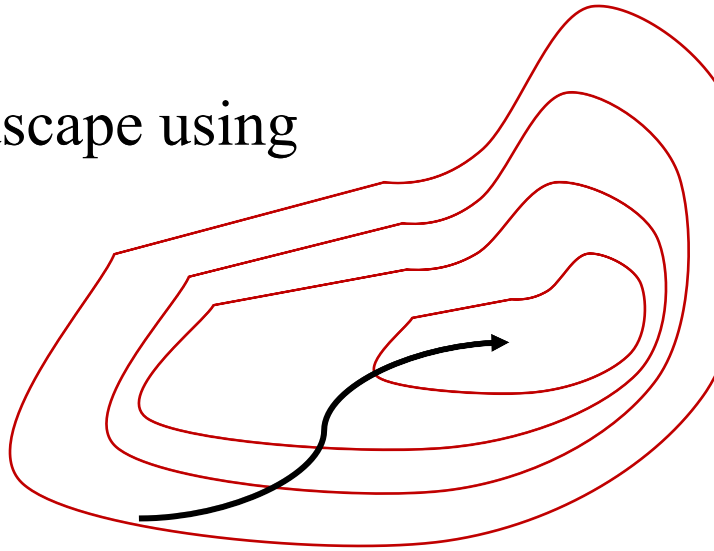
This is typically done by moving in the direction of steepest descent using **Stochastic Gradient Descent**.

At every step:

1. Compute the local gradient of $R(\omega) = \frac{1}{n}\sum_{i=1}^{n} L(t_i, f_i)$ using a *batch* of training data with $\boldsymbol{n} \ll \boldsymbol{N}$.

2. Move to the next position in the landscape using

$$\omega_{j+1} = \omega_j - \eta \nabla R$$



## Minimizing the Average Loss

Why does this algorithm

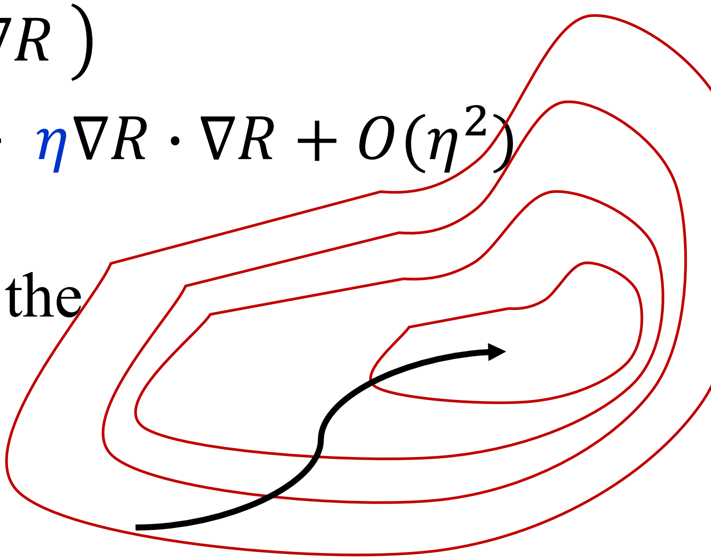
$$\omega_{j+1} = \omega_j - \eta \nabla R$$

work?

Here's why:

$$\begin{aligned} R(\omega_{j+1}) &= R(\omega_j - \eta \nabla R) \\ &= R(\omega_j) - \eta \nabla R \cdot \nabla R + O(\eta^2) \end{aligned}$$

If the $O(\eta^2)$ can be neglected, and since the $O(\eta)$ term is always negative, then $R(\omega_{j+1}) < R(\omega_j)$.



# Minimizing the Average Loss

Since the goal, ideally, is to find the lowest point of the "landscape" for an *infinite* amount of training data, it's instructive to consider the limit $N \to \infty$.

In that limit, the average loss $R(\omega)$ becomes the *functional*

$$R\,[f] = \int dx \int dt\, L(t, f)\, p(t, x)$$

which, given that $p(t, x) = p(t|x)\, p(x)$, can be written as

$$R[f] = \int dx\, p(x) \left[\int dt\, L(t, f)\, p(t|x)\right]$$

## Minimizing the Average Loss

The *calculus of variations* shows that if $p(x) > 0$ for all values of $x$ then the location of the minimum of $R\,[f]$, and hence the optimal function $f\,(x, \omega^*)$, is found by solving the equation

$$\frac{\delta R}{\delta f} = \int \frac{\partial L}{\partial f}\, p(t|x)\, dt = 0$$

The goal of a machine learning training algorithm is to find good approximations to solutions of the above equation using a (necessarily) finite training sample.

## Common Loss Functions

**Quadratic loss:** $L(t, f) = (t - f)^2$

$$\int \frac{\partial L}{\partial f}\, p(t|x)\, dt = 0$$

Solution

$$\boxed{f(x, \omega^*) = \int t\, p(t \mid x)\, dt}$$

Very Important Point (VIP): The solution is independent of the details of the model $f$. The solution depends solely on the form of the loss function and the probability distribution, $p(t, x)$, associated with the training data.

# Common Loss Functions

**Binary cross entropy loss:**

$$L(y, f) = -[t \log f + (1 - t) \log(1 - f)]$$

$$\int \frac{\partial L}{\partial f}\, p(t|x)\, dt = 0$$

Solution

$$\boxed{f(x, \omega^*) = p(t = 1 \mid x) = \frac{p(x|t = 1)\epsilon}{p(x|t = 1)\epsilon + p(x|t = 0)}}$$

where $t \in [0, 1]$ and $\epsilon = \frac{\pi(t=1)}{\pi(t=0)}$ is the ratio of training sample sizes for the two classes of objects labeled by $t \in [0, 1]$.

## Common Loss Functions

**Exponential loss:**

$$L(y, f) = \exp(-wtf/2)$$

$$\int \frac{\partial L}{\partial f} p(t|x)\, dt = 0$$

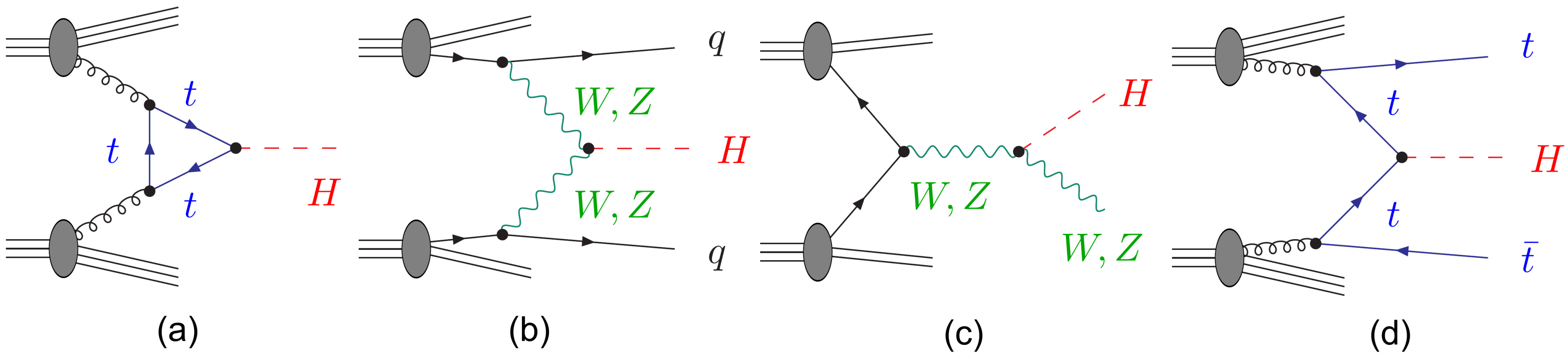
Solution

$$\boxed{f(x, \omega^*) = \frac{1}{w} \log\left(\frac{p(x|t=1)}{p(x|t=-1)}\epsilon\right)}$$

where $t \in [-1, 1]$ and $\epsilon = \frac{\pi(t=1)}{\pi(t=-1)}$ is the ratio of training sample sizes for the two classes labeled by $t \in [-1, 1]$.

## MODELS:
## BOOSTED DECISION TREES (BDT)

## $pp \rightarrow H \rightarrow ZZ \rightarrow 4l$



| **Process** | | $\sigma \times BR$ **(fb)** |
|---|---|---|
| (a) Gluon gluon fusion | (ggF) | |
| (b) Vector boson fusion | (VBF) | |
| (c) Associated production | (VH) | |
| (d) Top anti-top fusion | (ttH) | |

# $pp \to H \to ZZ \to 4l$

We shall use *decision trees* with the variables

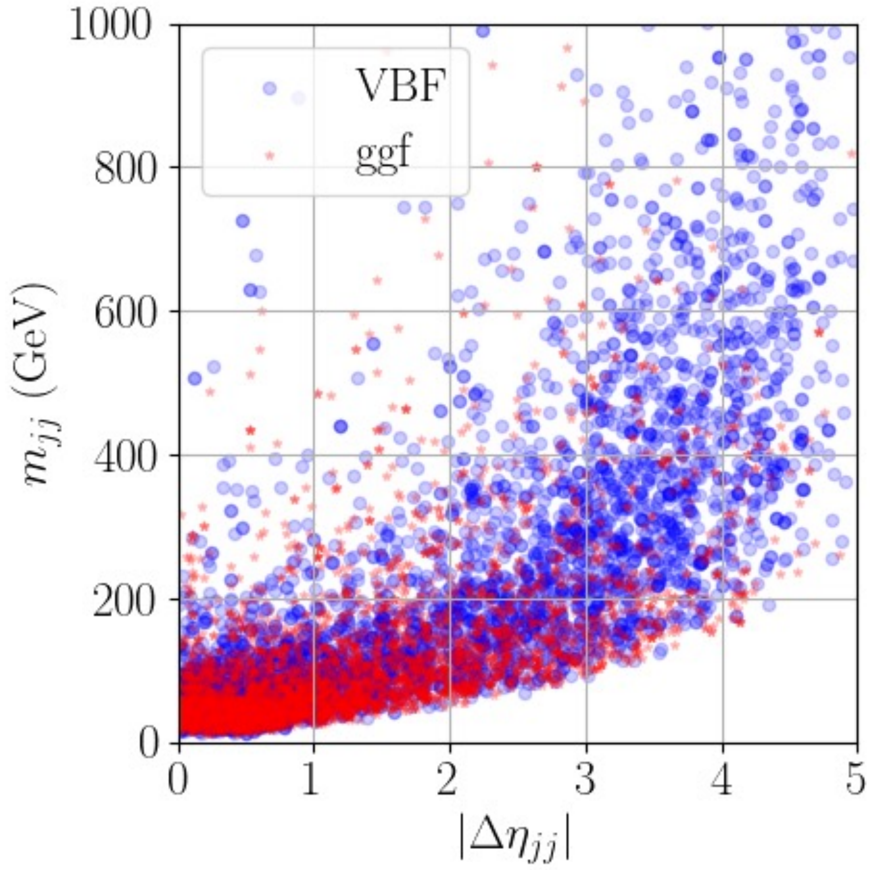
$|\Delta\eta|_{jj}, \qquad m_{jj}$

to try to separate

$VV \to H$

from

$gg \to H.$



ATLAS Open Data

## Decision Trees

A decision tree (DT) is a set of **if then else** statements that form a tree-like structure.

Algorithm: recursively partition the space into regions of diminishing *impurity*.

A common measure of impurity is the *Gini Index*:

$p\,(1 - p)$, where $p$ is the *purity*

$p = S\,/\,(S + B)$

$p = 0$ or 1: maximum purity

$p = 0.5$: maximum impurity

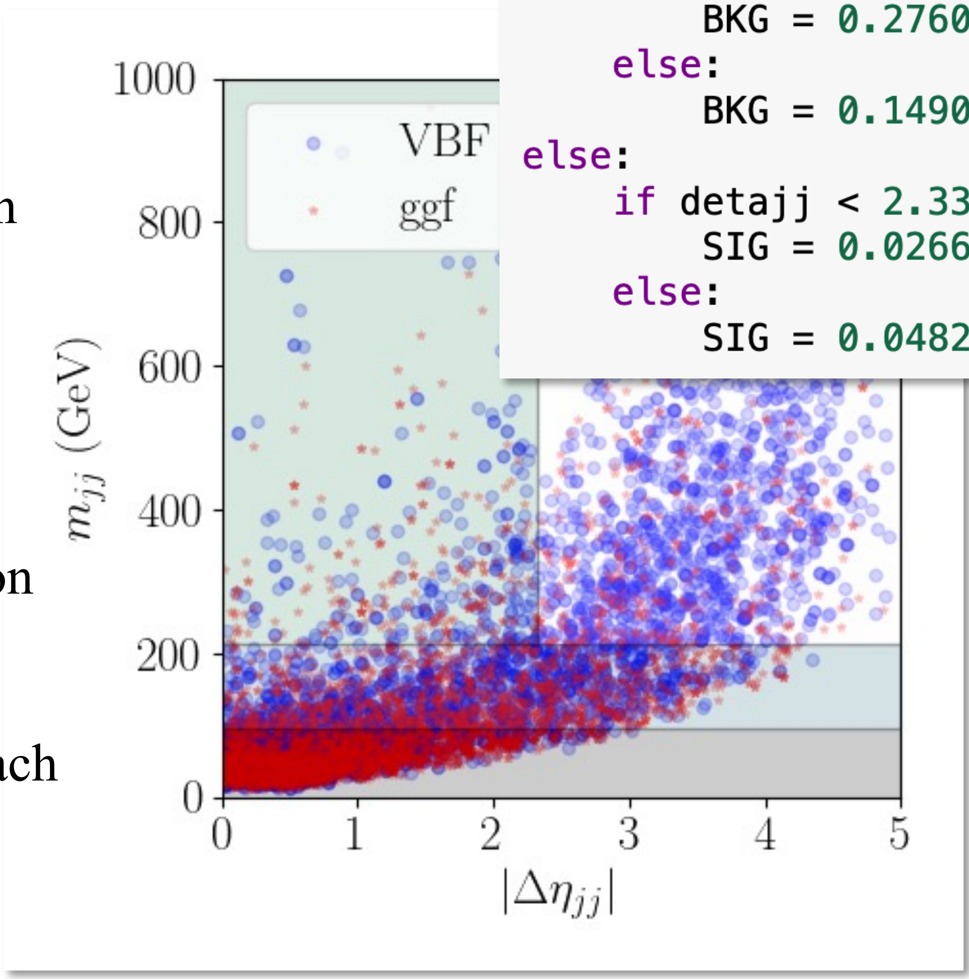
(Corrado Gini, 1884-1965)

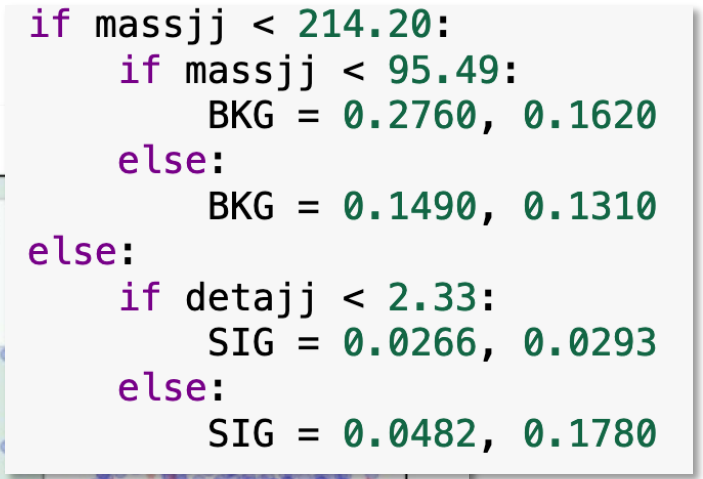
# Decision Trees

1. For each variable, find the partition ("cut") that gives the greatest *decrease* in impurity.

2. Choose the *best partition* among all partitions and split the data along that partition into *two* subsets.

3. Repeat 1. and 2. for each subset of data.





```
if massjj < 214.20:
    if massjj < 95.49:
        BKG = 0.2760, 0.1620
    else:
        BKG = 0.1490, 0.1310
else:
    if detajj < 2.33:
        SIG = 0.0266, 0.0293
    else:
        SIG = 0.0482, 0.1780
```

## Decision Trees (DT)

Unfortunately, decision trees are *unstable*!

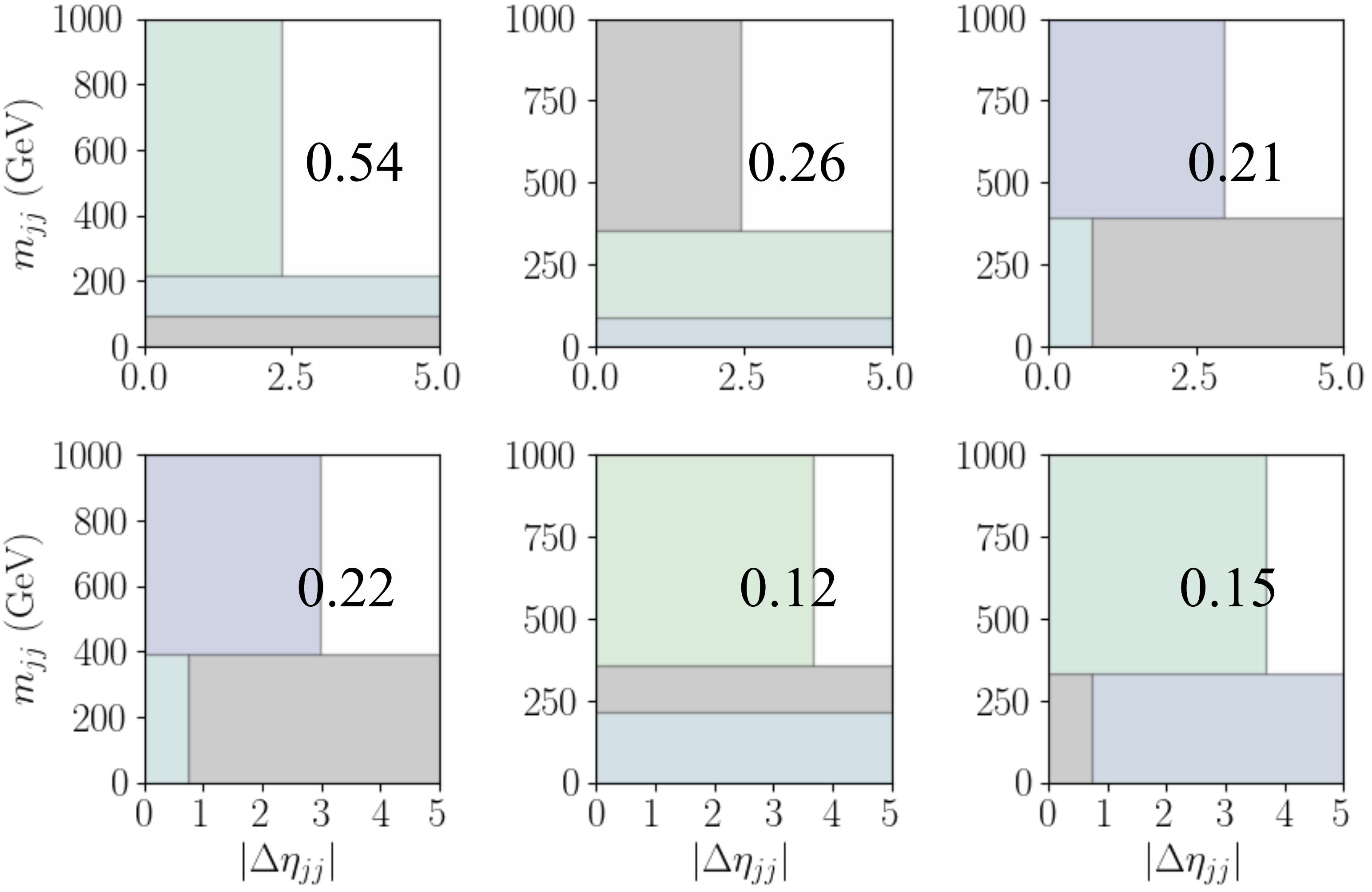
# DT Averaging Methods

The most popular decision tree averaging methods are:

| | |
|---|---|
| • Bagging: | each tree is trained on a bootstrap* sample drawn from the training set |
| • Random Forest: | bagging with randomized trees |
| • Boosting: | each tree trained on a different reweighting of the training set |

*A bootstrap sample is a sample of size $N$ drawn, *with replacement*, from another of the same size. Duplicates can occur and are allowed.

# First 6 Decision Trees

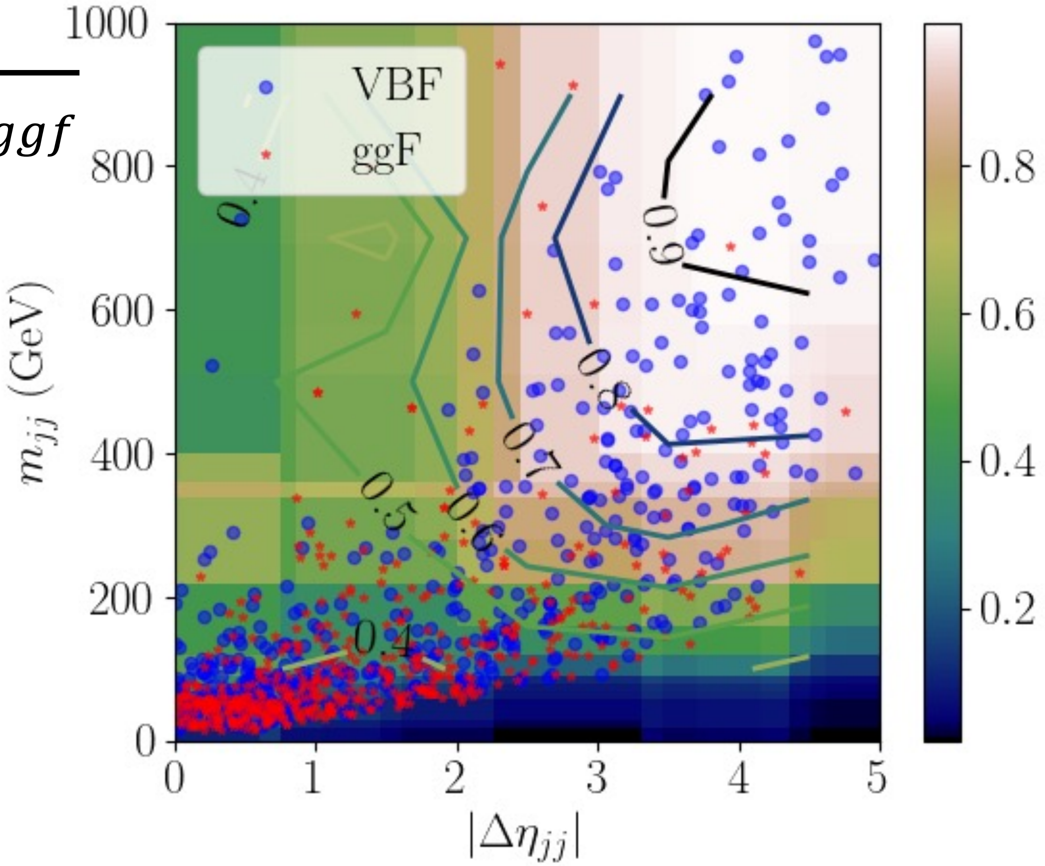


# Boosted Decision Trees (BDT)

The contours are obtained from

$$p(t=1|x) \approx \frac{H_{VBF}}{H_{VBF} + H_{ggf}}$$

where $H_{VBF}$ and $H_{ggf}$ are histograms in the $x = |\Delta\eta_{jj}|, m_{jj}$ space.



A BDT minimizes

$$R[f] = E\left[\exp\left(-\frac{wtf}{2}\right)\right]$$

with $w = 2$, which yields $p(t=1|x) = 1/(1+\exp(-2f))$

## MODELS: CNN, GNN, GD

## Convolutional Neural Networks (CNN)

A few milestones in the development of convolutional neural networks

- 1980 Kunihiko Fukushima invents the neocognitron that is able to perform character recognition.

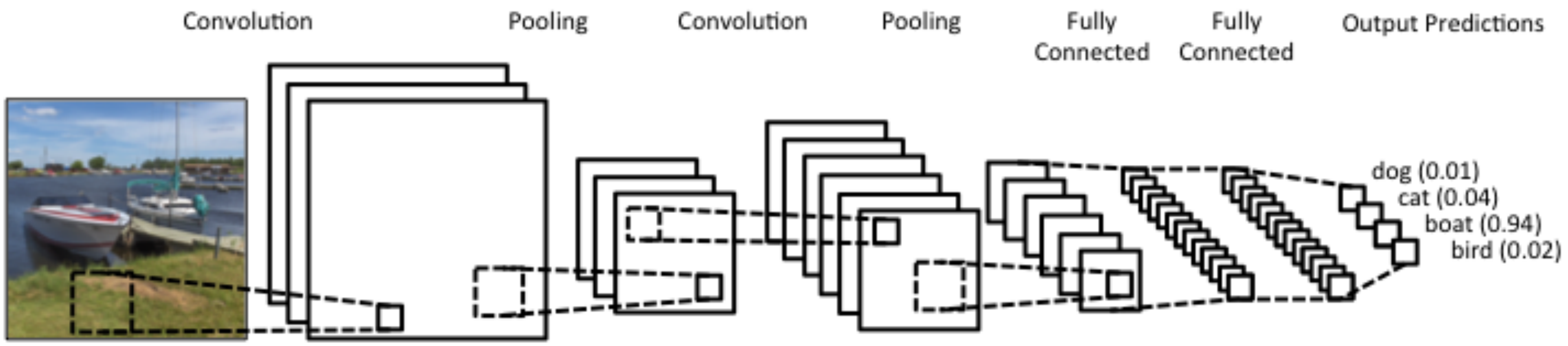
- 1998 Yan LeCun developed LeNet, which is able to recognize handwritten zip code digits.

- 2012 Alex Krizhevsky introduced AlexNet, which produced state of the art results on the image database ImageNet (https://image-net.org/).

## Convolutional Neural Networks

What are CNNs?

CNNs are *ML models* that create a *representation* of data that are naturally structured into 1D, 2D, or 3D arrays. The objects represented by these data are then classified using a fully connected NN.

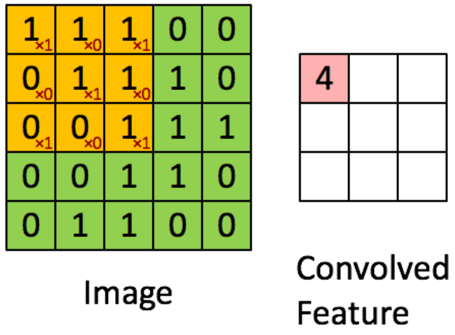


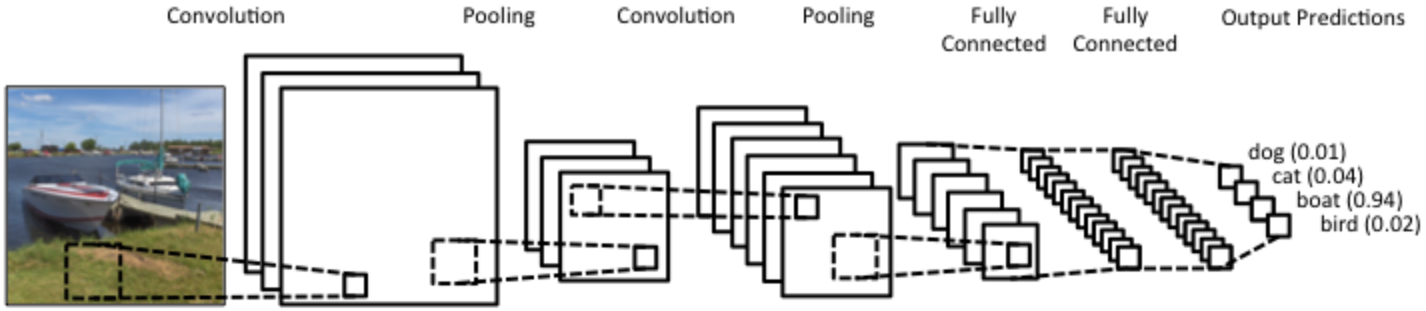
## Convolutional Neural Networks

A CNN comprises three types of processing layers:
1. convolution, 2. pooling, and 3. classification.

1. **Convolution layers**

   The input layer is "convolved" with one or more matrices using element-wise products that are then summed. In this example, since the sliding matrix fits 9 times, we compress the input from a 5 x 5 to a to a 3 x 3 matrix.

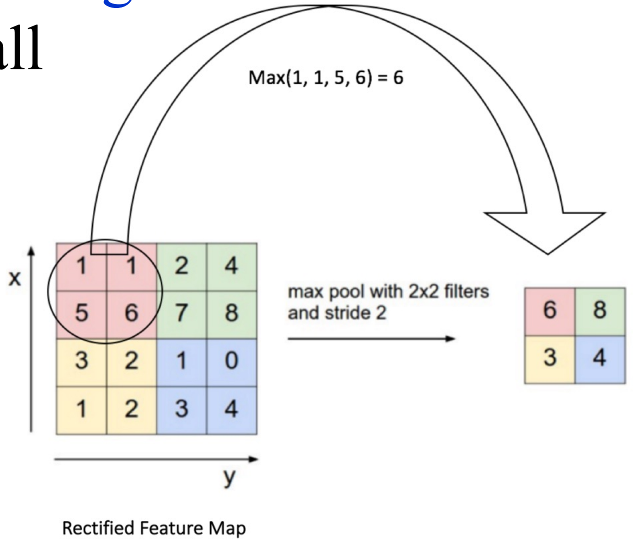


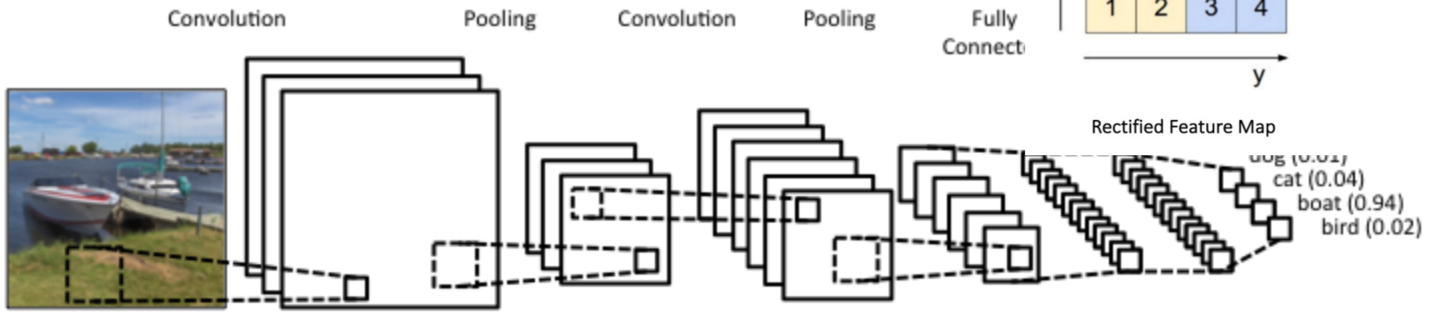


# Convolutional Neural Networks

## 2. Pooling Layers

After convolution, and a pixel-by-pixel non-linear map (using, e.g., the function $y = \mathrm{ReLU}(x)$), a coarse-graining of the layer is performed called max pooling in which the maximum values within a series of small windows are selected and become the output of a pooling layer.



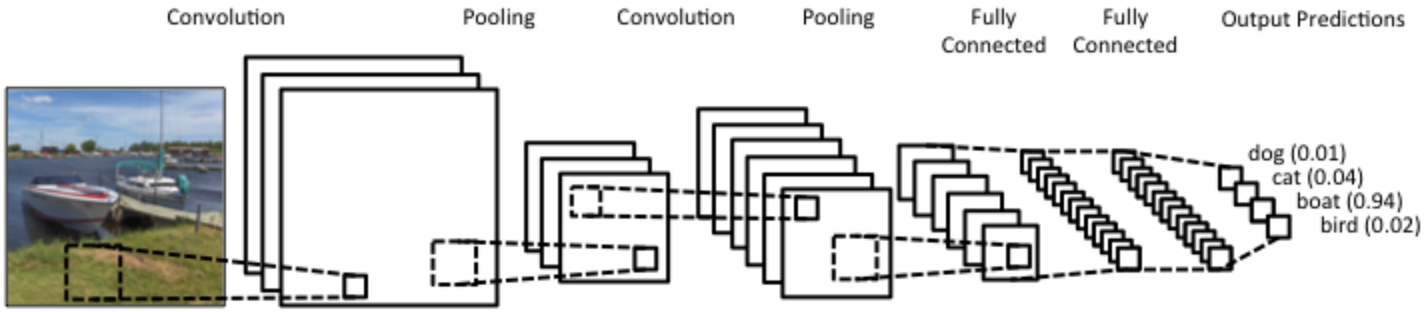


# Convolutional Neural Networks

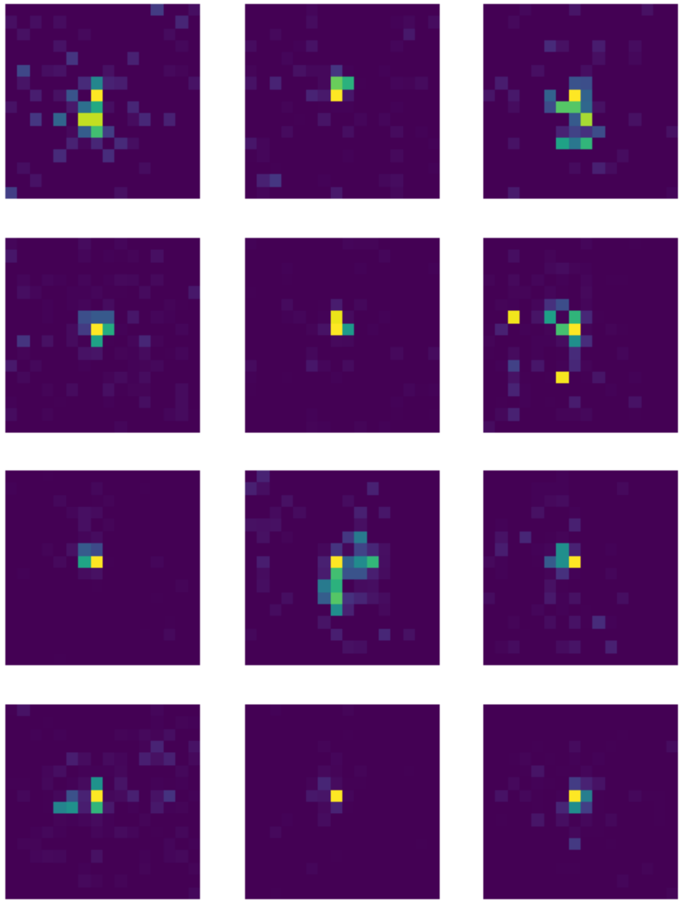
3. **Classification Layers**

   After an alternating sequence of convolution and pooling layers, the outputs go to a standard neural network, either shallow or deep. The final outputs correspond to the different classes, which approximate the probabilities:

$$p(C_k|x) = p(x|C_k)p(C_k)/\sum_{m=1}^{M} p(x|C_m)p(C_m)$$



# Example 1: Quark/Gluon Jets



Consider the task of classifying *single-channel* images of quark- and gluon-initiated jets[1].

A *batch* of input data is of shape $(N, C\ H, W)$, where $N$ is the batch size, $C$, the number of channels/image, and $H \times W$ is the size in pixels of each channel of an image.

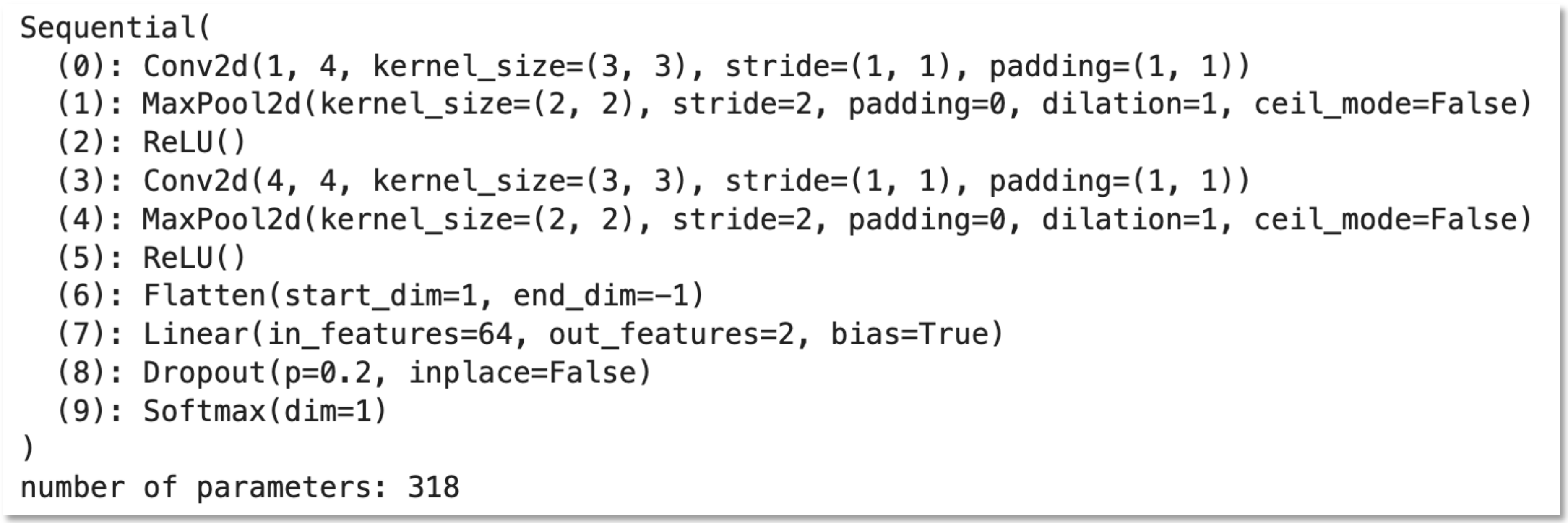
1. https://www.kaggle.com/datasets/anonymous2506/quarkgluon.

## Example 1: Quark/Gluon Jets

Here is a high-level view of a simple CNN model:

$$f(x) = \text{softmax}\left(\text{dropout}(\text{linear}(\text{flatten}\left(\boldsymbol{g}(\boldsymbol{c}\left(\boldsymbol{h}(\boldsymbol{c}(x))\right))\right)))\right)$$

And here is a code-level view:

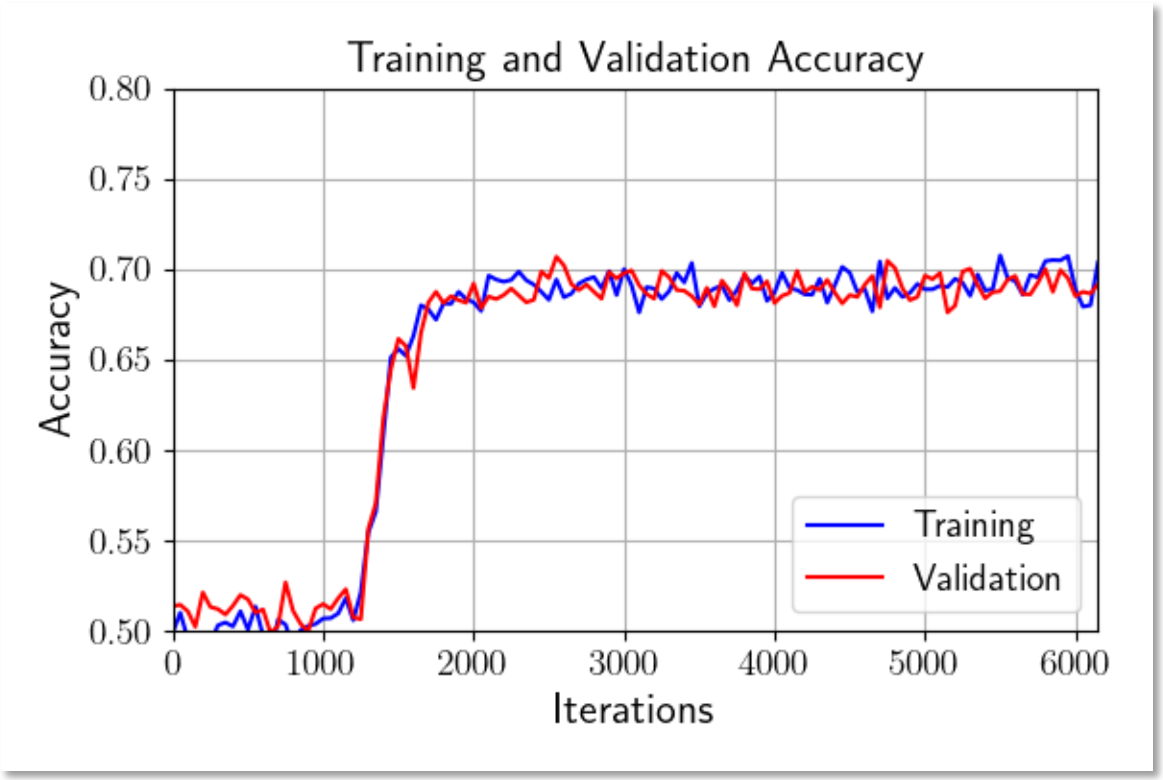


```
Sequential(
  (0): Conv2d(1, 4, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (1): MaxPool2d(kernel_size=(2, 2), stride=2, padding=0, dilation=1, ceil_mode=False)
  (2): ReLU()
  (3): Conv2d(4, 4, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (4): MaxPool2d(kernel_size=(2, 2), stride=2, padding=0, dilation=1, ceil_mode=False)
  (5): ReLU()
  (6): Flatten(start_dim=1, end_dim=-1)
  (7): Linear(in_features=64, out_features=2, bias=True)
  (8): Dropout(p=0.2, inplace=False)
  (9): Softmax(dim=1)
)
number of parameters: 318
```

## Example 1: Quark/Gluon Jets

$$f(x) = \text{softmax}\left(\text{dropout}(\text{linear}(\text{flatten}\left(\boldsymbol{g}(\boldsymbol{c}(\boldsymbol{h}(\boldsymbol{c}(x))))\right)))\right)$$

Accuracy on a balanced dataset: 69%.



## MODELS: CNN, GNN, GD

## Example 3: *ν* Classification

Graph neural networks (GNN) are extremely popular in particle physics. We consider a recent example from IceCube*.

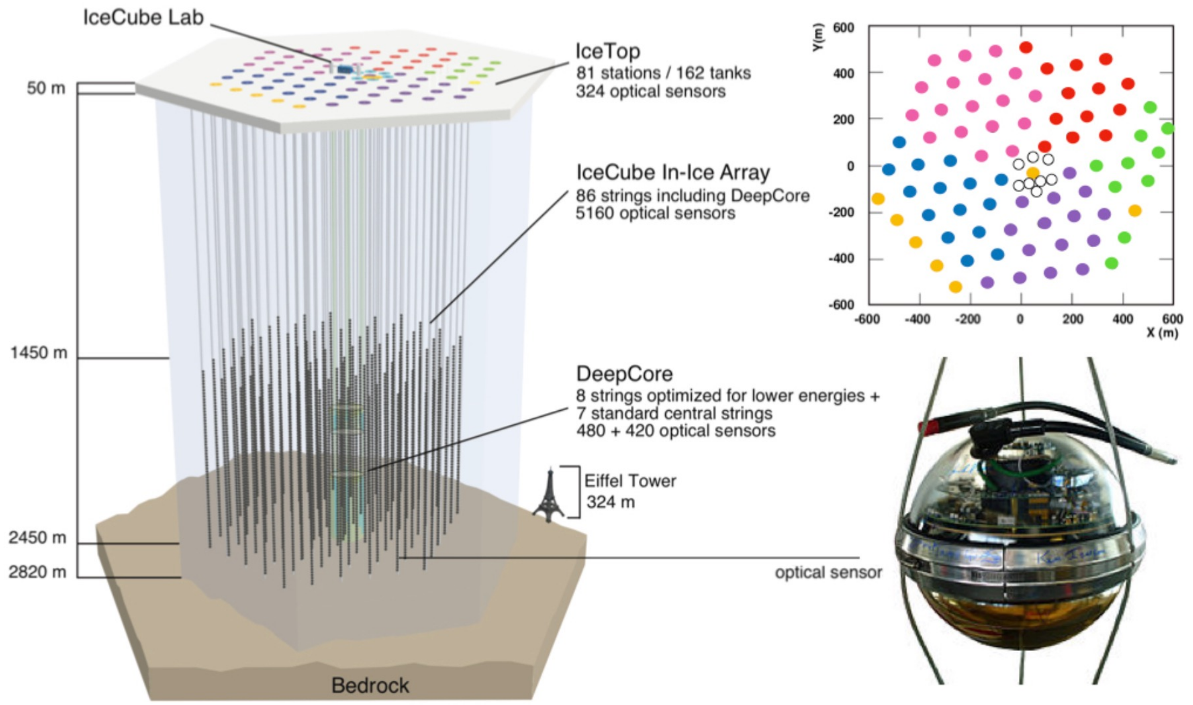
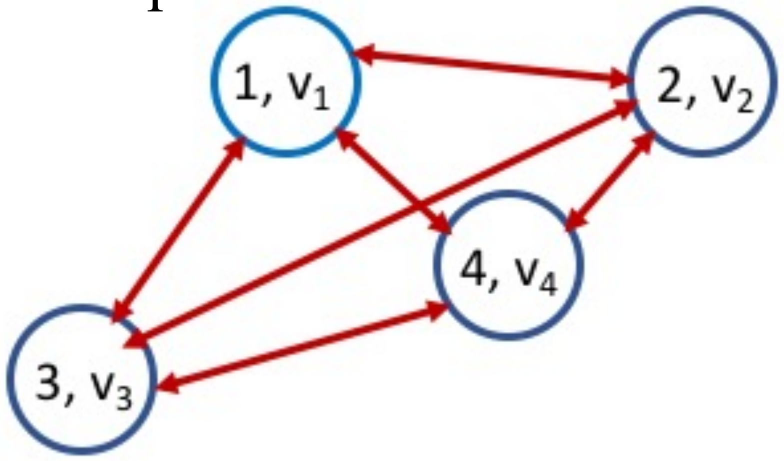


Fig. 1. The IceCube Neutrino Observatory with the in-ice array, its sub-

*N. Choma et al. IceCube collaboration, Graph Neural Networks for IceCube Signal Classification, arXiv:1809.06166v1

# Example 3: $\nu$ Classification

IceCube models the signals from $n$ Digital Optical Modules (DOMs) as the vertices of a graph. Each vertex is associated with a $d$-dimensional (row-wise) vector of attributes $v_i = (x_1, \cdots, x_d)_i$, three of which are the spatial coordinates $(x, y, z)$ of the DOM.



An $n \times n$ *adjacency matrix*,

$\boldsymbol{A}(\sigma)_{ij} = \text{softmax}(d_{ij})$, with

$d_{ij} = \exp\left(-\|x_i - x_j\|^2 / 2\sigma^2\right)$

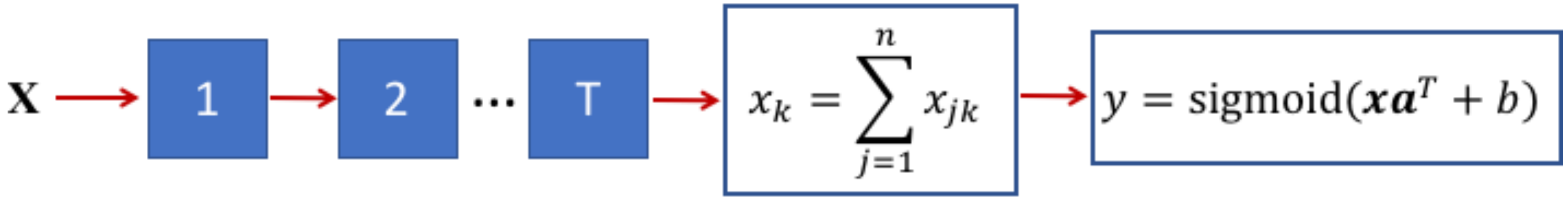
models the edges.

The vectors $v_i$ are concatenated *vertically* into an $n \times d$ matrix: $\boldsymbol{X} = [v_1; \cdots; v_n]$.

(Horizontal concatenation is denoted by $\boldsymbol{X} = [v_1, \cdots, v_n]$.)

# Example 3: $\nu$ Classification

- The $n \times d$ matrix, $\boldsymbol{X}$, passes through a sequence of identical graph processors $\boldsymbol{X}$.
- At the end, $\boldsymbol{X}$ is mapped to a $d$-dimensional vector $\boldsymbol{x}$ using a map that is *permutation invariant* with respect to the vertices and *invariant* with respect to the number of vertices.
- Finally, the vector $\boldsymbol{x}$ is mapped to the scalar output $0 \leq y \leq 1$ using a sigmoid.

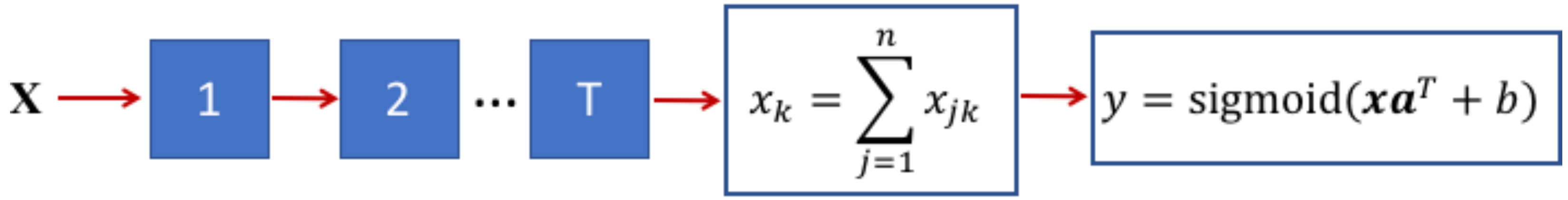

X
1
2
T
$x_k = \sum_{j=1}^{n} x_{jk}$
$y = \text{sigmoid}(\boldsymbol{x}\boldsymbol{a}^T + b)$

# Example 3: $\nu$ Classification

Each graph processor is parameterized by a $2d \times d/2$ matrix, $w$, and a scaler $b$ and computes:

$$\boldsymbol{Y} = [\boldsymbol{AX}, \boldsymbol{X}]w + bu, \qquad \boldsymbol{X} \leftarrow [\mathrm{ReLU}(\boldsymbol{Y}), \boldsymbol{Y}]$$

The ReLU is applied *element-wise* and $u$ is an $n \times d/2$ matrix of ones.


X
1
2
T
$x_k = \sum_{j=1}^{n} x_{jk}$
$y = \text{sigmoid}(\boldsymbol{x}\boldsymbol{a}^T + b)$

# Example 3: $\nu$ Classification

The GNN is **6.3** times more efficient than the IceCube physics baseline analysis at a signal to noise ratio that is **3** times better.

It also outperforms a 3D CNN.

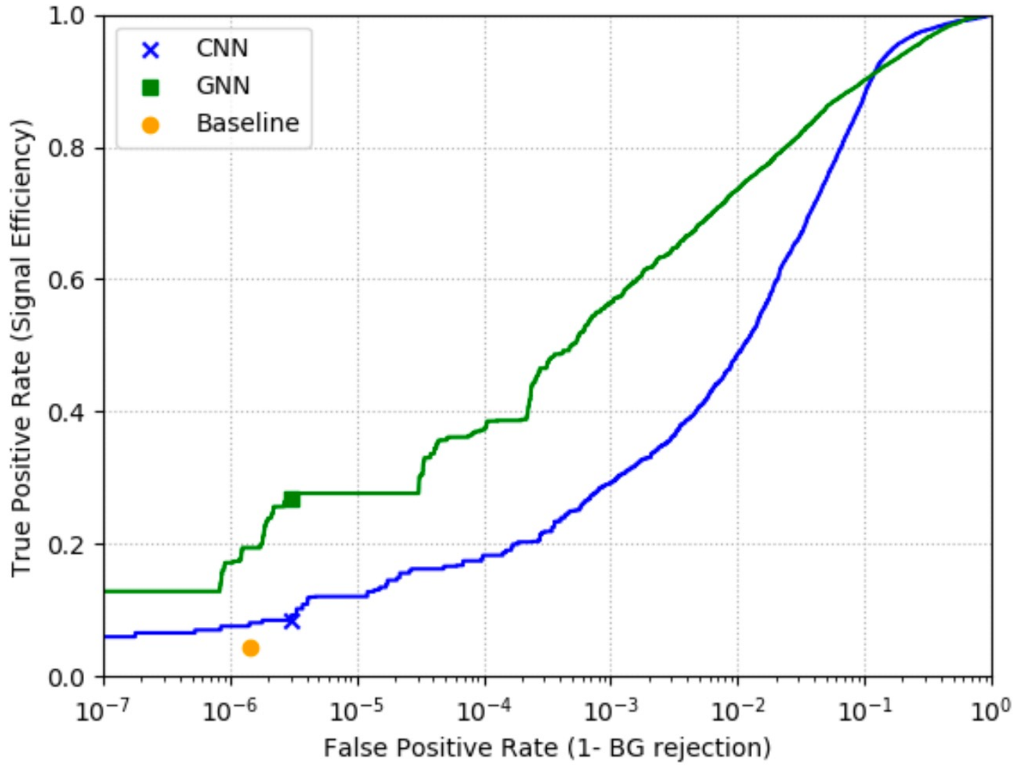


Fig. 3. Receiver operating characteristic curve for various methods considered in this paper. The green square and blue X indicate the evaluation point for the GNN and CNN, respectively.

## MODELS: CNN, GNN, GD

## Example 4: Diffusion

Generative models are mathematical functions that generate data according to a well-defined plan.

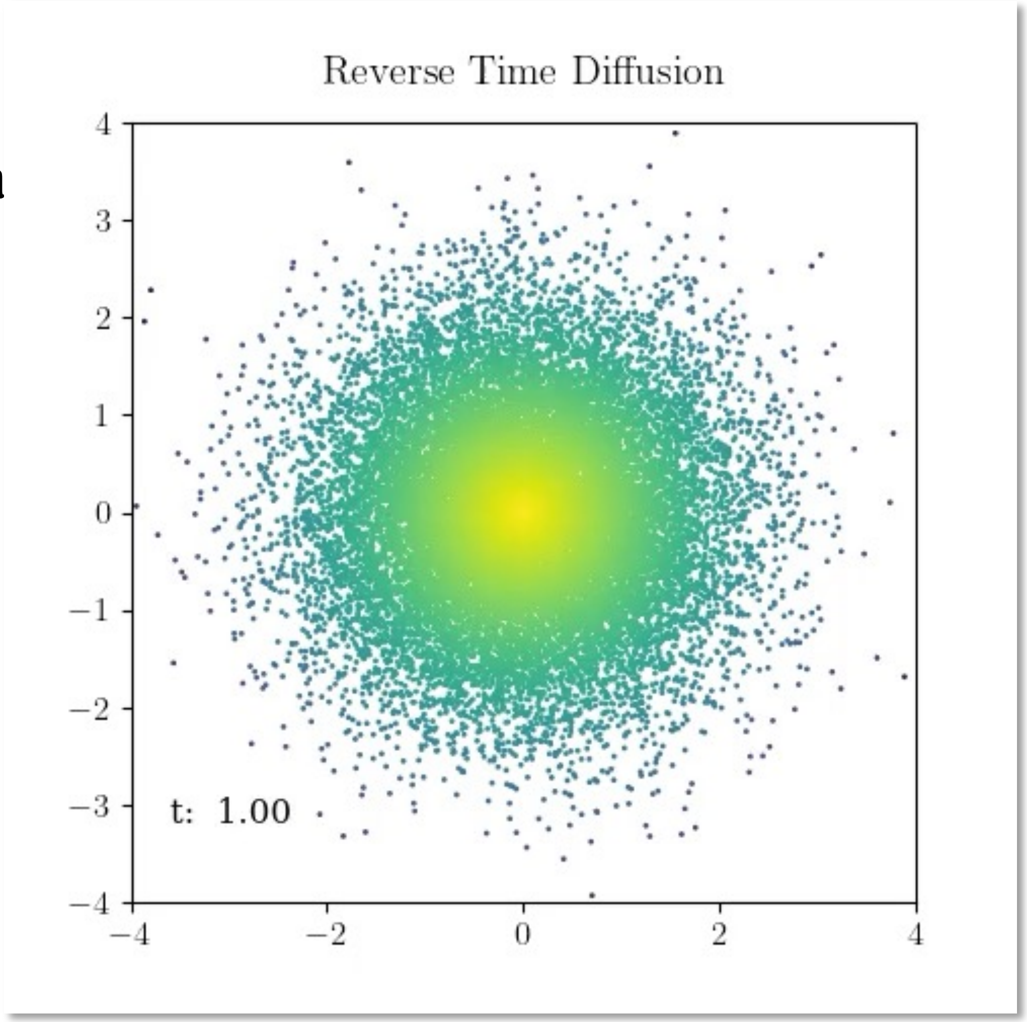
A typical application is generating samples from a complex multi-dimensional distribution from a simpler, known, distribution such as a diagonal multi-dimensional Gaussian.

See, for example,

Yanfang Lui, Minglei Yang, Zezhong Zhang, Feng Bao, Yanzhao Cao, and Guannan Zhang, *Diffusion-Model-Assisted Supervised Learning of Generative Models for Density Estimation*, arXiv:2310.14458v1, 22 Oct 2023

## Example 4: Diffusion

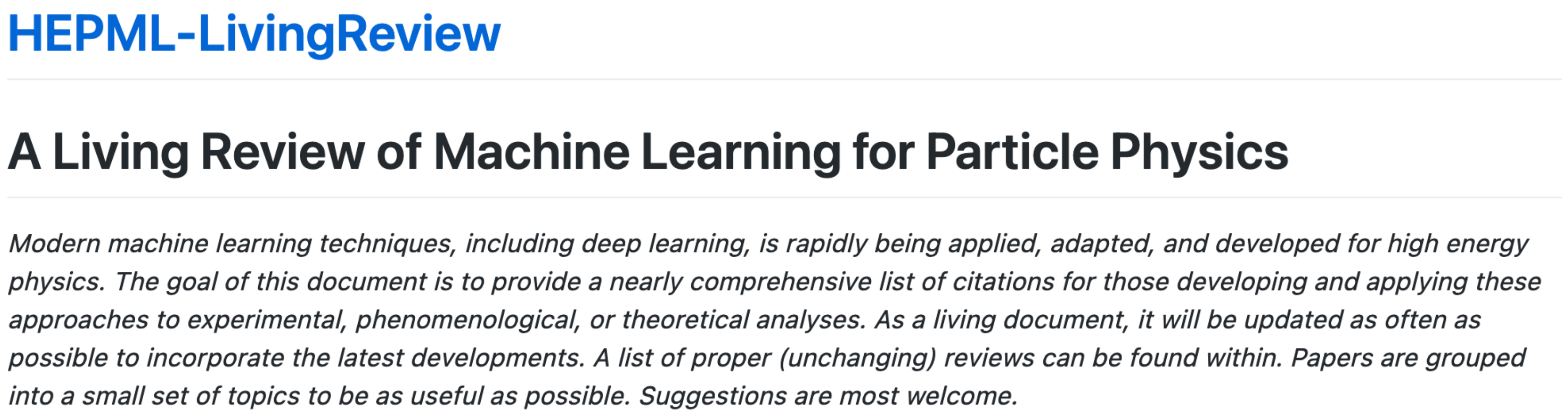
Here is an example of a model that samples from a diagonal bi-variate Gaussian and *deterministically* maps each generated point to a point on a spiral.



## Summary (1)

For a reasonably comprehensive archive of machine learning development and applications, I recommend this website:

https://iml-wg.github.io/HEPML-LivingReview/


HEPML-LivingReview

A Living Review of Machine Learning for Particle Physics

*Modern machine learning techniques, including deep learning, is rapidly being applied, adapted, and developed for high energy physics. The goal of this document is to provide a nearly comprehensive list of citations for those developing and applying these approaches to experimental, phenomenological, or theoretical analyses. As a living document, it will be updated as often as possible to incorporate the latest developments. A list of proper (unchanging) reviews can be found within. Papers are grouped into a small set of topics to be as useful as possible. Suggestions are most welcome.*

## Summary (2)

- Machine learning (ML) models are *statistical models* trained, that is, fitted, to data by minimizing a given *average loss*.

- The mathematical quantity approximated by an ML model depends solely on the *form of the loss function and the probability distribution of the training data*. In particular, it does not depend on the details of the model!

- The quality of the approximation, however, does depend on the model, as well as the amount of data used, and the quality of the training.