Lectures on Machine Learning

DATA SCIENCE IN (ASTRO)PARTICLE PHYSICS and COSMOLOGY

School 25, 26, 27 MARCH 2019

INFN Istituto Nazionale di Fisica Nucleare

Tommaso Dorigo, INFN-Padova

Braga, March 25-27, 2019

Lecture 3
Neural networks and HEP examples

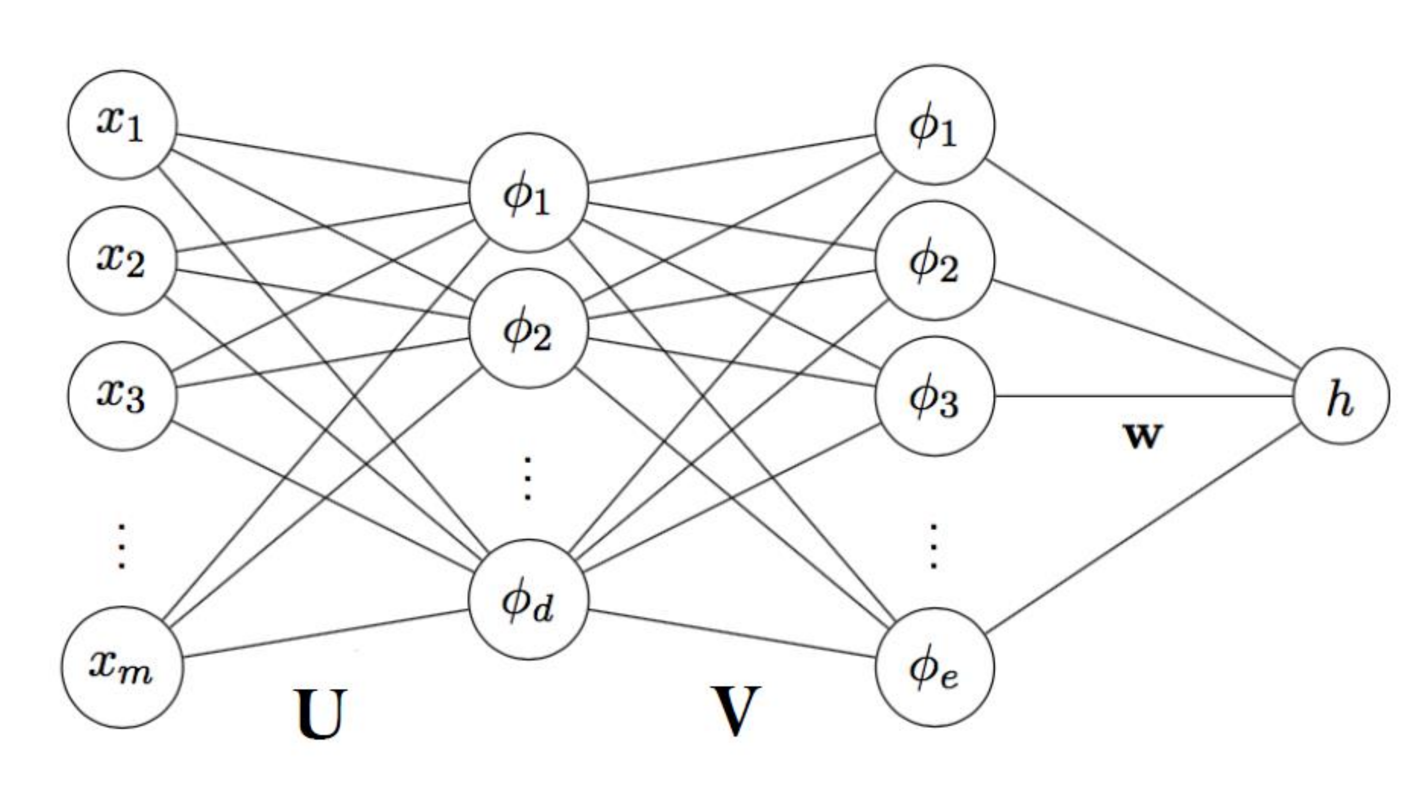
Higgs challenge
the HiggsML challenge
May to September 2014
When High Energy Physics meets Machine Learning
info to participate and compete : https://www.kaggle.com/c/higgs-boson
ATLAS EXPERIMENT
LAL
Inria
kaggle
Google

Contents - 3

Lecture 3: NNs and HEP applications

- Neural networks
- Playing with NNs
- Advanced techniques
  - INFERNO
  - DNNs
  - Convolutional NNs
  - Genetic algorithms
- Practical tips
- Machine Learning in HEP
  - Generalities
  - kNN for H→bb
  - Clustering for HH theory space benchmarking
  - Higgs Kaggle challenge
- Conclusions

NEURAL NETWORKS



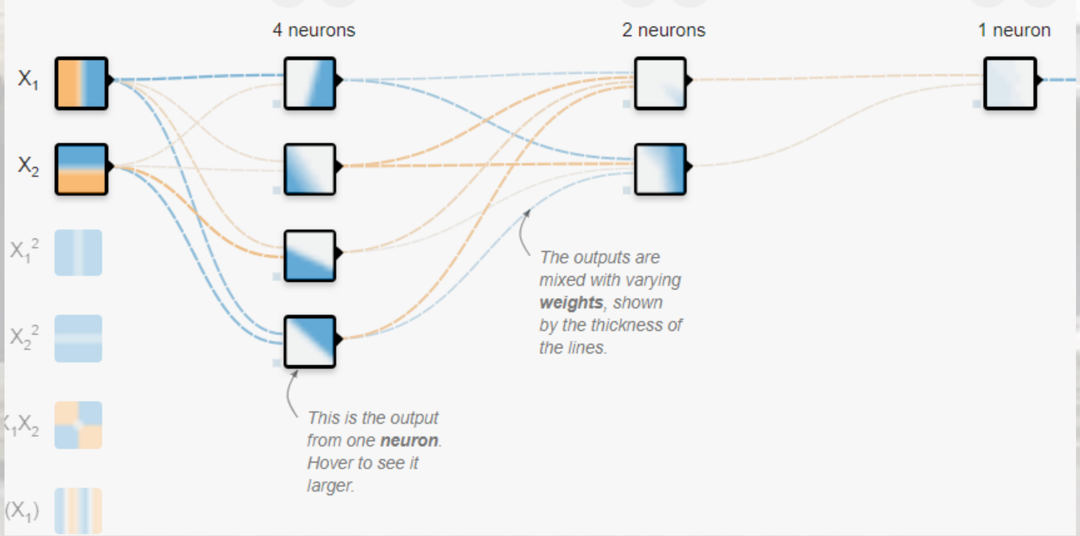
Neural Networks - introduction

An artificial neural network (ANN) is a program that simulates the behaviour of a series of neurons and their connections

ANNs are capable of producing very flexible functions of the feature space variables

At the heart of the ANN there is an architecture of nodes organized in layers. Every "neuron" of a layer receives inputs from some of (or all) the neurons of the previous layer

Neural networks are extremely powerful tools for supervised learning tasks, such as classification and regression.

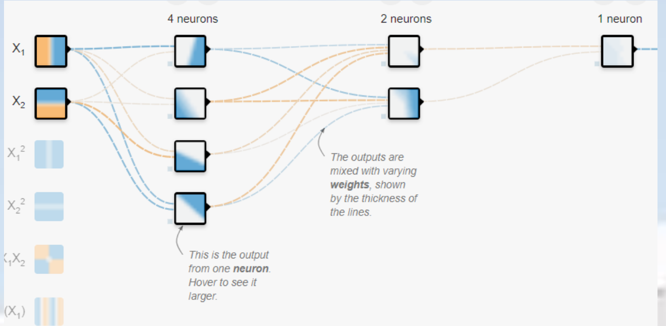


The outputs are mixed with varying **weights**, shown by the thickness of the lines.

This is the output from one **neuron**. Hover to see it larger.

Looking inside

Each neuron may emit a strong or weak signal, in response to the combined stimulus coming from its inputs → **activation functions** parametrize the response signal.



*The outputs are mixed with varying **weights**, shown by the thickness of the lines.*

*This is the output from one **neuron**. Hover to see it larger.*

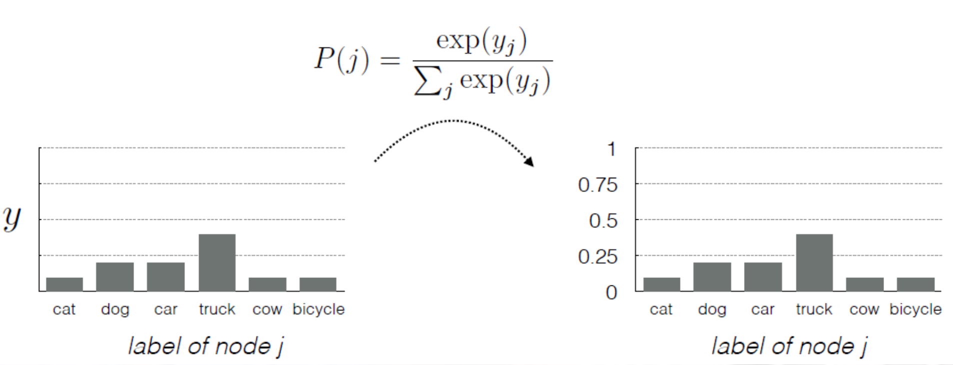
The signal is transmitted to the neurons connected to it in the next layer.
Mathematically, the behaviour of every neuron is described by two parameters (a bias and a weight). The training phase of the ANN (learning) consists in finding parameter values which **minimize a loss function**

For binary classification problems, the loss function may be simply the **fraction of misclassified events.** From that one can construct **a binary cross entropy.**

The crucial step in the minimization phase is **"back-propagation".**
Training events are used to compute the loss function. During back-propagation the contribution of each neuron (with associated weight and bias) to the loss is computed. This way one may estimate how the loss would change if those parameters were changed. The iteration of the procedure allows to obtain optimal values, with different convergence strategies possible (e.g. "**gradient descent**").

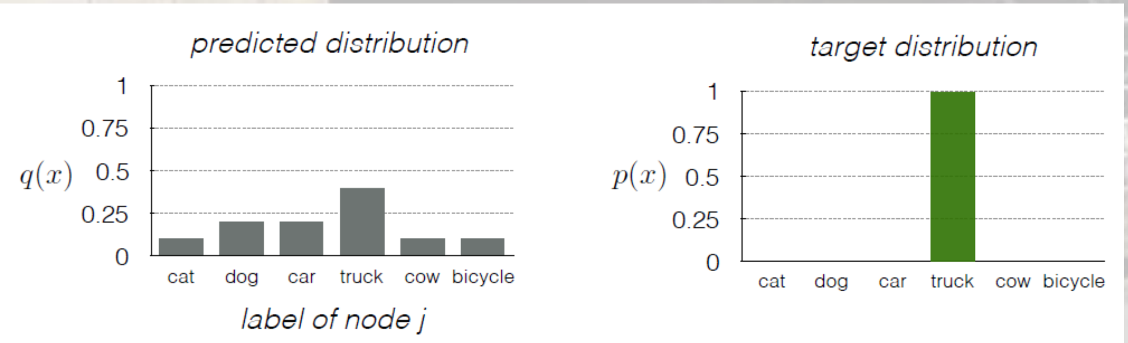
Multi-class NN classification in 2 steps

1) convert the output label into a class probability, using the softmax function

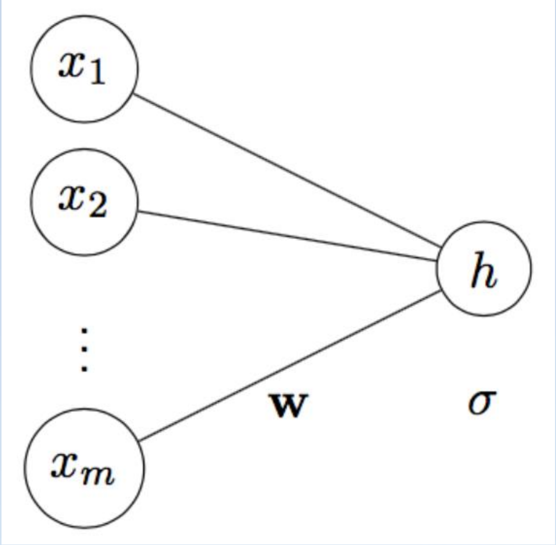


2) minimize the cross-entropy loss between the output probabilities and the target. The cross-entropy loss is computed as the KL divergence,

$$L = \sum p(x) \log \frac{p(x)}{q(x)}$$



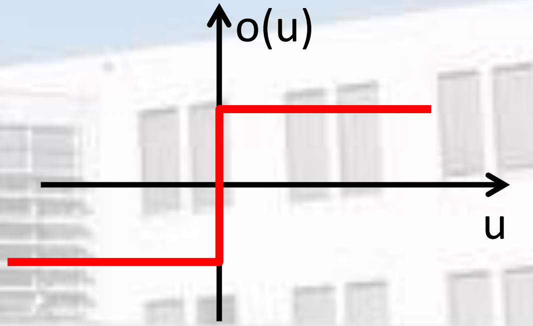
The Perceptron



The perceptron is the simplest NN

The idea is to try to create a mathematical model of a single neuron, as a "node" which receives several inputs, sums them, and gets "activated" if the sum surpasses a fixed threshold

The perceptron task is to select between two classes based in the inputs it receives. The inputs are combined linearly:



$$u = w^T x + b$$

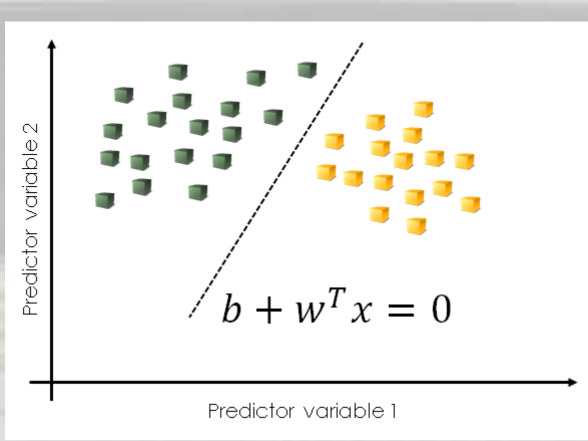
Above, x is a vector of inputs, and w is a "weight vector", b is a constant bias. The output is calculated as

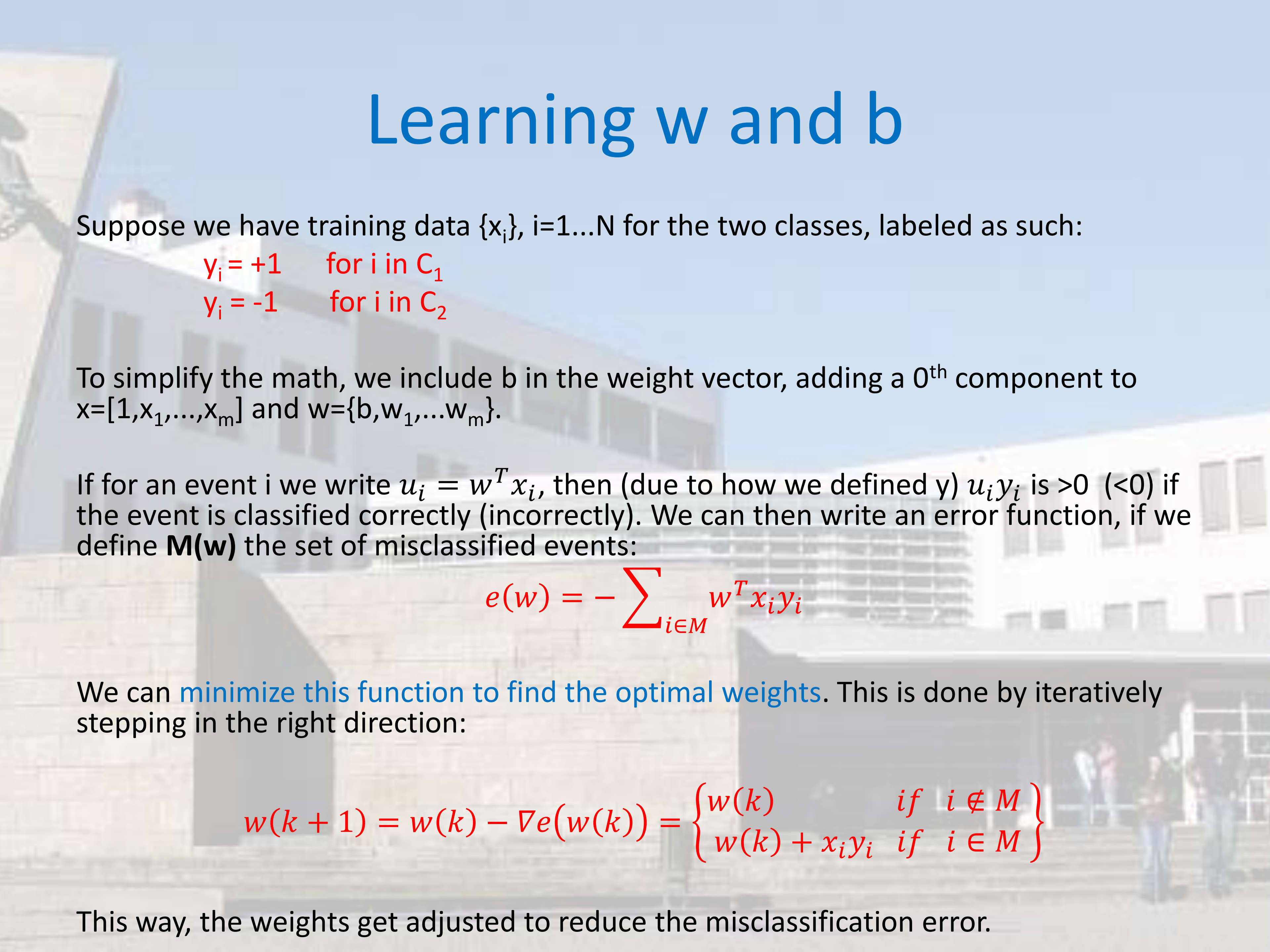
$$o(u) = \begin{Bmatrix} +1 \; if \; u \geq 0 \\ -1 \; if \; u < 0 \end{Bmatrix}$$

The predicted class here depends on the sign of u. As w and b define a hyperplane in the feature space:

$$w^T x + b = 0,$$

by adjusting their values one can achieve ideal classification if the classes are linearly separable





Learning w and b

Suppose we have training data {x_i}, i=1...N for the two classes, labeled as such:

y_i = +1 for i in C_1

y_i = -1 for i in C_2

To simplify the math, we include b in the weight vector, adding a 0th component to x=[1,x_1,...,x_m] and w={b,w_1,...w_m}.

If for an event i we write $u_i = w^T x_i$, then (due to how we defined y) $u_i y_i$ is >0 (<0) if the event is classified correctly (incorrectly). We can then write an error function, if we define **M(w)** the set of misclassified events:

$$e(w) = -\sum_{i \in M} w^T x_i y_i$$

We can minimize this function to find the optimal weights. This is done by iteratively stepping in the right direction:

$$w(k+1) = w(k) - \nabla e\big(w(k)\big) = \begin{Bmatrix} w(k) & if \quad i \notin M \\ w(k) + x_i y_i & if \quad i \in M \end{Bmatrix}$$

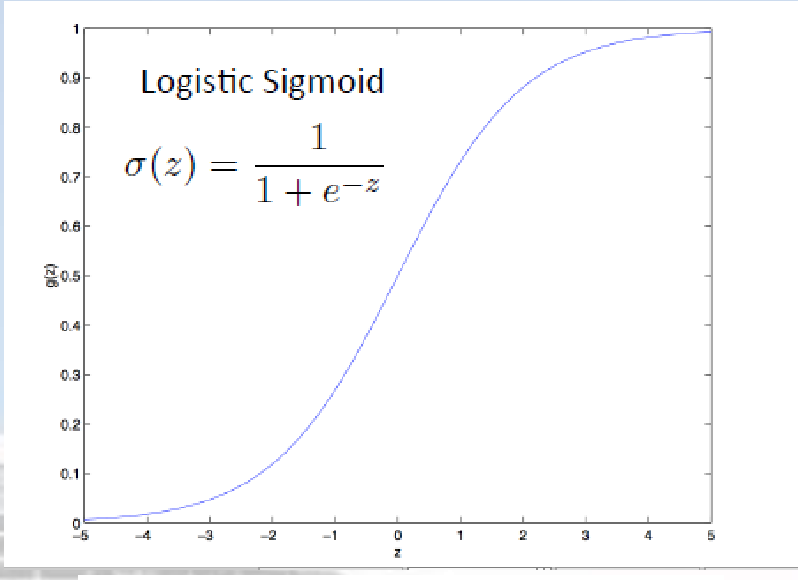
This way, the weights get adjusted to reduce the misclassification error.

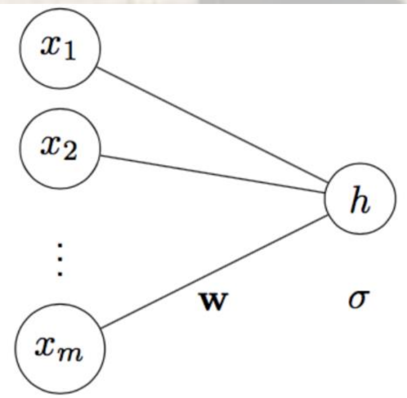
The smooth output: logistic sigmoid

In neural networks, rather than a discontinuous score as in the perceptron, the response of a neuron is modeled by a continuous, differentiable function

The simplest of these is the sigmoid.

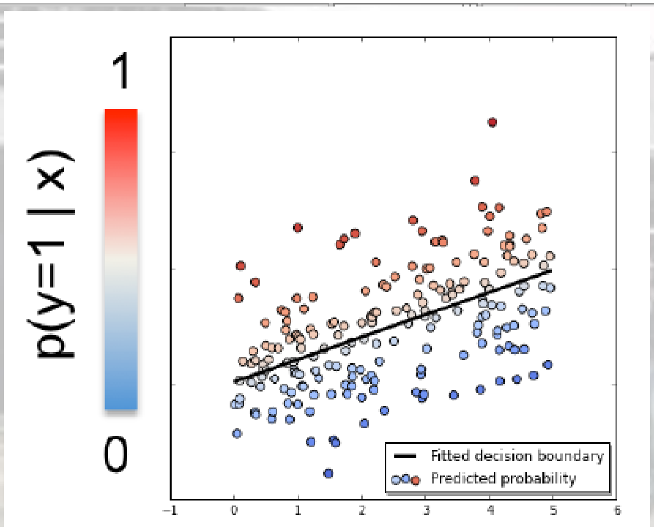
In logistic regression, one assigns a probability to events based on the distance from the boundary, using the logistic sigmoid function:



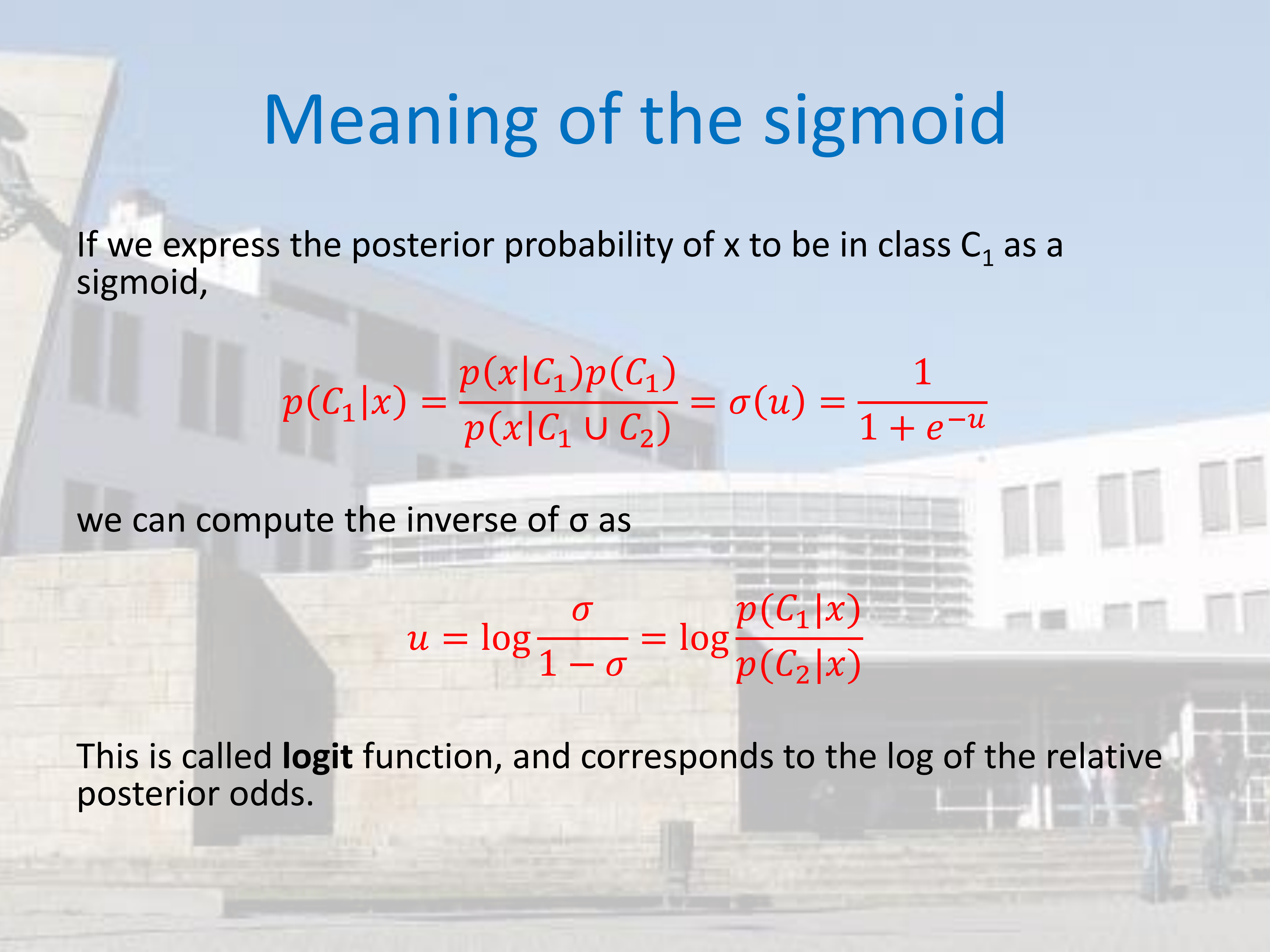


$$h(\mathbf{x};\mathbf{w}) = \mathbf{w}^T\mathbf{x}$$

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



[M.Kagan]



Meaning of the sigmoid

If we express the posterior probability of x to be in class C_1 as a sigmoid,

$$p(C_1|x) = \frac{p(x|C_1)p(C_1)}{p(x|C_1 \cup C_2)} = \sigma(u) = \frac{1}{1+e^{-u}}$$

we can compute the inverse of σ as

$$u = \log\frac{\sigma}{1-\sigma} = \log\frac{p(C_1|x)}{p(C_2|x)}$$

This is called **logit** function, and corresponds to the log of the relative posterior odds.

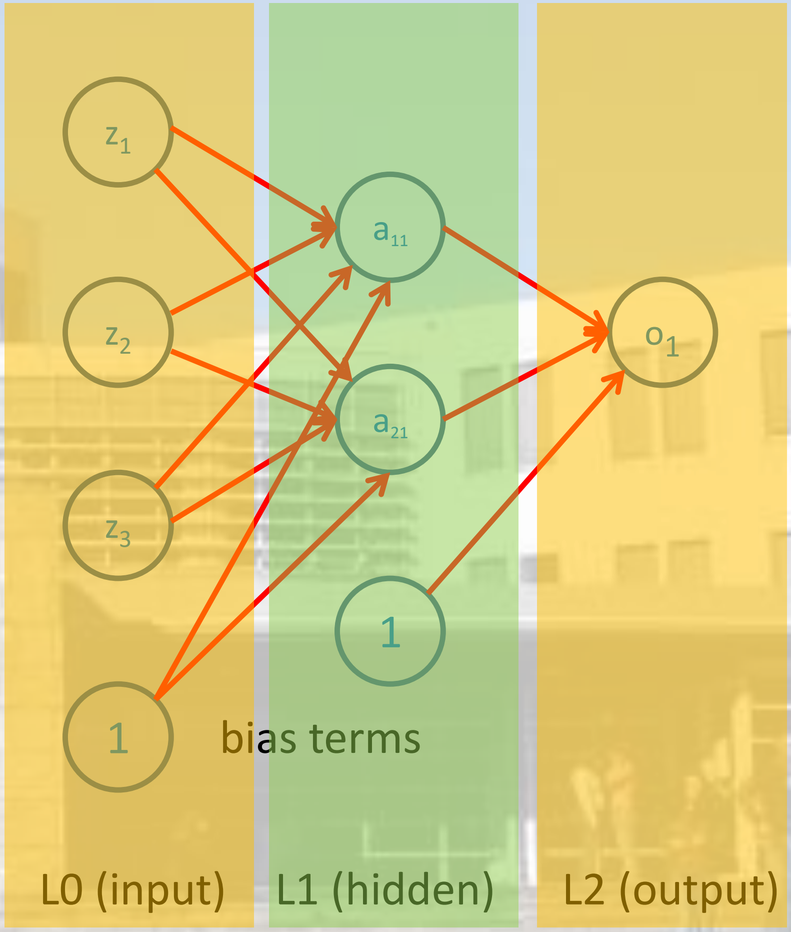
The feed-forward neural network

We can put together these elements to create a **non-linear function of the inputs**, which can learn much more complex separation boundaries than a hyperplane

A feed-forward NN is composed of nodes connected by forward links. There is >=1 hidden layer, and one output layer

- for binary classification, all you need is one node in it

The nodes need not all be connected, but there must be no circular reference – the value of a node must be a deterministic function of what comes before



Calculation of the function

The FFNN is a complex, differentiable function of the inputs, and it offers a simple solution to the problem of optimizing its parameters.

For a formal treatment let us define:

- $\mathbf{z_i}$ be the inputs to node i (i=1,...Z, where Z is the number of inputs for that node; z_0=1). For layer 0 (input layer), z_i=x_i is the vector of inputs.

  [When we need to specify it, we add an index j for the node and an index m for the layer, so $\mathbf{z_{ijm}}$ is the ith input to the jth node of the mth layer]
- $\mathbf{w_{ijm}}$ be the vector of weights for that node (with w_{0jm}=b_{jm} the bias term).

The rule is that at each node the inputs are summed with a linear weighting, to obtain the activation of that node, a ($\mathbf{a_{jm}}$ when needed):

$$a = \sum_{i=0}^{Z} w_i z_i$$

i=1...Z inputs for a node
j=1...J nodes of a layer
m=1...M layers

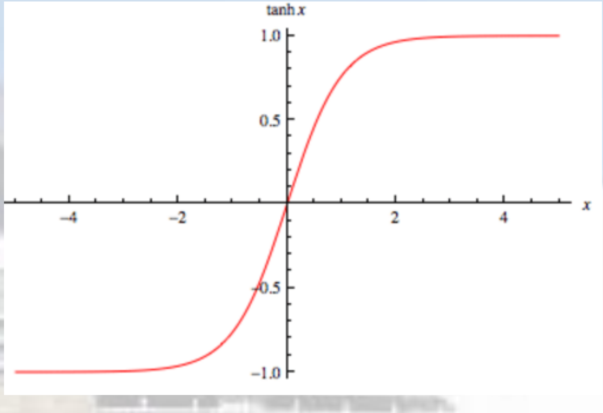
Adding nonlinearity

Nonlinearity is introduced by choosing a suitable continuous differentiable function of a to model the behaviour of the node. A common choice for inner layers is

$$h(a) = \tanh(a)$$

whose derivative is

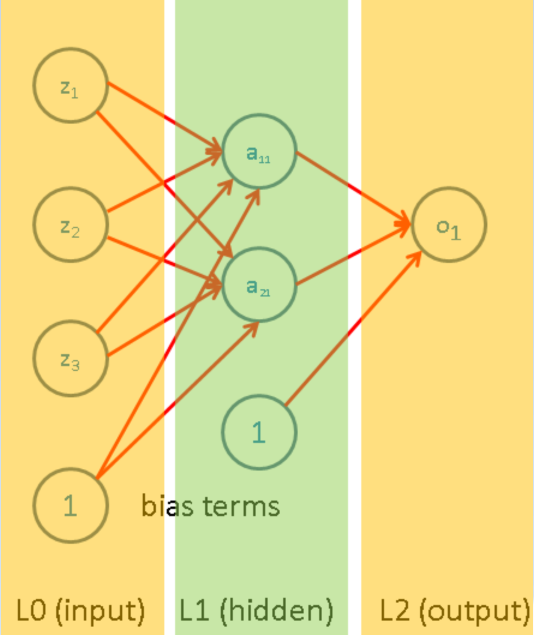
$$h'(a) = 1 - h(a)^2$$



For the output layer, as we saw, the **sigmoid** is the common choice for binary classification (one output). For K classes, one uses the soft-max generalization, which retains the interpretability of outputs as posterior probabilities:

$$o_k = \sigma(a_k) = \frac{e^{ak}}{\sum_{i=1}^{K} e^{ai}}$$

Calculation/2



Let us consider a 3/2/1 architecture, and compute activations in the hidden layer as

(for each node j:) $$a_{j1} = \sum_{i=0}^{3} w_{ij1} x_i$$

The output layer receives as inputs the sum of j=1, j=2 activation functions

$$z_j = h(a_{j1}) = \tanh a_{j1}$$

The activation in the output layer is then

$$a = \sum_{i=0}^{2} w_{i12} z_i$$ (node 1 of layer 2)

and the output is the activation function (a sigmoid) of the above:

$$o = \sigma(a)$$

$$= \left\{1 + \exp\left[-w_{012} - \sum_{j=1}^{2} w_{j12} \tanh\left(w_{0j1} + \sum_{i=1}^{3} w_{ij1} x_i\right)\right]\right\}^{-1}$$

Backpropagation: where the magic happens

Consider the two-class case, with binary output y_{12}=1/0 for signal and background, and the 3/2/1 2-layer network of the previous slides. If a(x,w) is the activation on the output node, the output is

$$o(x, w) = \frac{1}{1 + e^{-a(x,w)}}$$

This means that $p(C_1|x)=o(x,w)$, and $p(C_2|x)=1-o(x,w)$ so we can write syntetically

$$p(y|x) = o(x,w)^y\ [1-o(x,w)]^{1-y}$$

We have a differentiable model of the class probabilities, so we may write the –log(L) for a training dataset $\{x_{(1)}...x_{(N)}\}$ as

$$e(w) = -\sum_{n=1}^{N}[y_n \log o_n(x_n, w) + (1 - y_n) \log(1 - o_n(x_n, w))]$$

This is called (binary) cross entropy; it is the most commonly used loss for classification.

Finding the weights

To find the optimal weights, we need to minimize the BCE loss. We are helped by noting that the loss is decomposable in per-event contributions e_n, which are a function of the weights. We need to only find the gradient of the error function with respect to each weight:

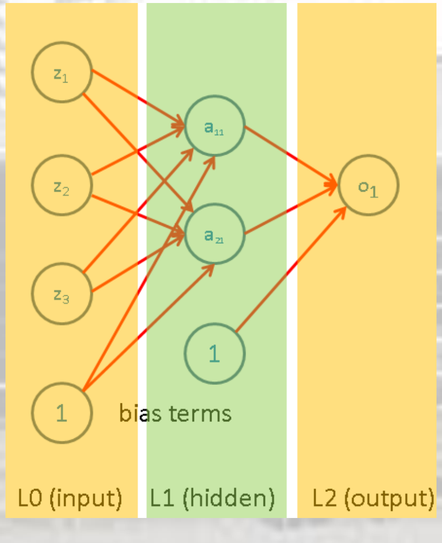
$$e_n(w) = -\{y_n \log o_n\,(x_n, w) + (1 - y_n) log[1 - o_n(x_n, w)]\}$$

We take the derivative of e_n WRT the weight for the i^{th} input to node j in layer m, w_{ijm}:

$$\frac{\partial e_n}{\partial w_{ijm}} = \frac{\partial e_n}{\partial a_{jm}} \frac{\partial a_{jm}}{\partial w_{ijm}} = e_{njm} z_{ijm}$$

where $e_{njm} = \frac{\partial e_n}{\partial a_{jm}}$ and we have used the activation

$$a_{jm} = \sum_{i=0}^{n_{m-1}} w_{ijm} z_{ijm}$$



Finding the weights/2

We work our way from the output layer M to the previous ones. While for the output layer the error caused by event n is $e_{n1M}=o_n-y_n$, for nodes at layer m=M-1 we should write

$$e_{njm} = \frac{\partial e_n}{\partial a_{jm}} = \sum_{q=1}^{n_{m+1}} \frac{\partial e_n}{\partial a_{q,m+1}} \frac{\partial a_{q,m+1}}{\partial a_{jm}} = \sum_{q=1}^{n_{m+1}} e_{nq,m+1} \frac{\partial a_{q,m+1}}{\partial a_{jm}}$$

(At layer m+1=M there is only one node, but this formula works for any layer)

To evaluate derivatives we proceed thus:

Remember: $a_{jm} = \sum_{i=0}^{n_{m-1}} w_{ijm} z_{ijm}$

$$\frac{\partial a_{q,m+1}}{\partial a_{jm}} =$$

$$= \sum_{i=1}^{n_m} w_{iq,m+1} \frac{\partial z_{iq,m+1}}{\partial a_{jm}} = \sum_{i=1}^{n_m} w_{iq,m+1} \frac{\partial h(a_{im})}{\partial a_{jm}} = w_{iq,m+1} h'(a_{jm})$$

Hence we find

$$e_{njm} = h'(a_{jm}) \sum_{q=1}^{n_{m+1}} w_{qj,m+1} e_{nq,m+1}$$

Finding the weights/3

The formulas of the previous slide allow us to work our way back recursively through the NN, using the chain rule.

For tanh() activation in inner layers the error contribution of event n due to weights in node j of layer m is written

$$e_{njm} = \left[1 - h(a_{jm})^2\right] \sum_{q=1}^{n_{m+1}} w_{qj,m+1} e_{nq,m+1}$$

The weight updating process is iterative, and modulated by a learning rate η:

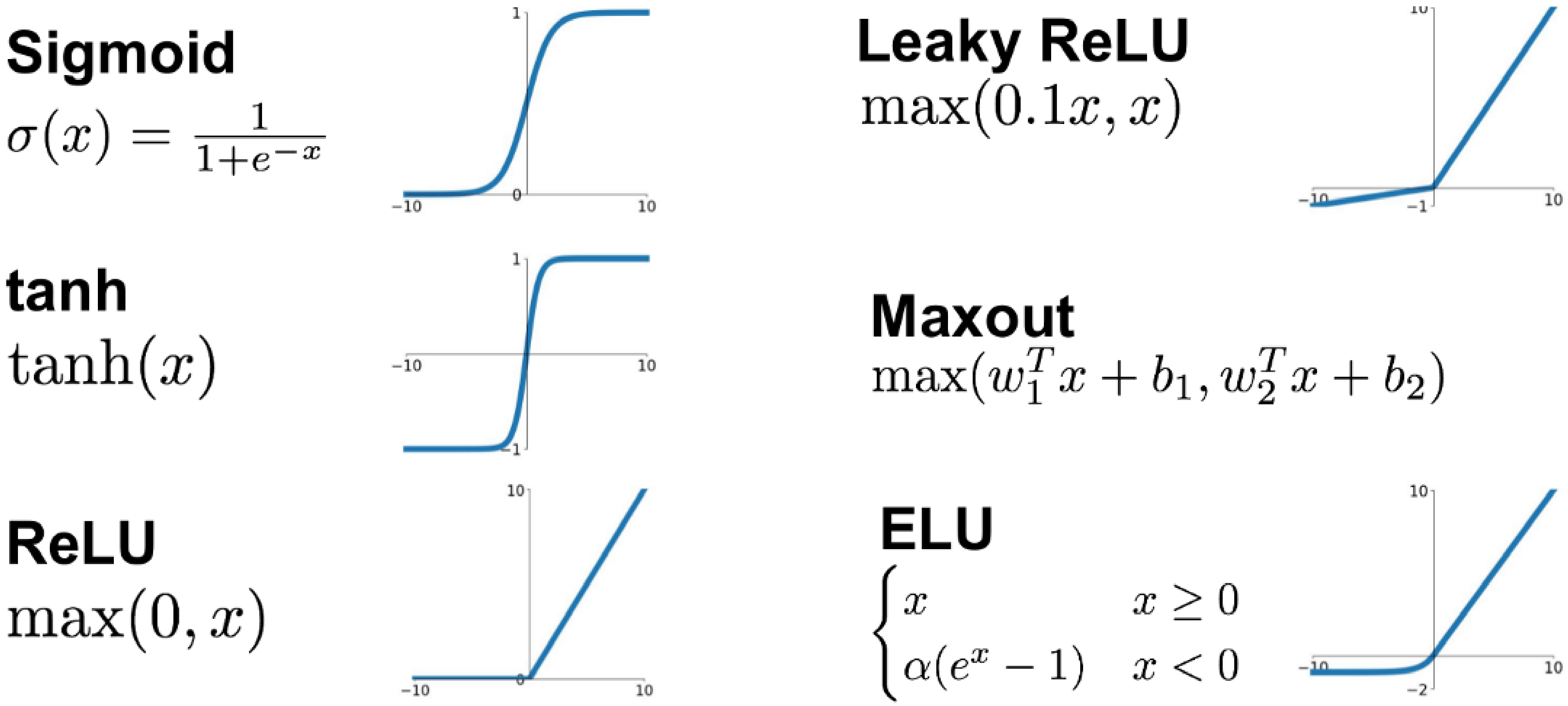
$$w^{(k+1)} = w^k - \eta \nabla e(w^k)$$

One may choose to use the whole training set to evaluate the gradient (batch learning), or to update weights at each new event evaluation (online learning). They have different applications and properties.

Online learning is better fit to jump out of minima, but the learning may take longer.

Choices of Activation function

We want it non-linear, otherwise hidden layers do nothing; it must be monotonic to ensure convergence of the optimization problem; and smooth. Often also preferable to have rapidly changing for input close to zero, slowly changing for large input



Sigmoid

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

tanh

$\tanh(x)$

ReLU

$\max(0, x)$

Leaky ReLU

$\max(0.1x, x)$

Maxout

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

ELU

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

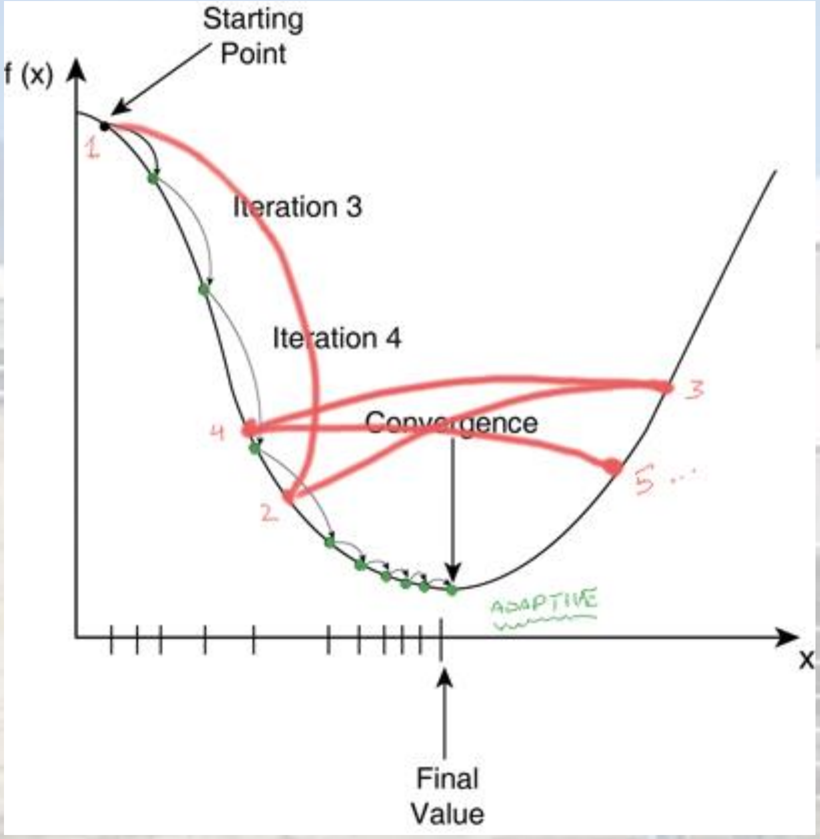
Learning rate

Above we mentioned the learning rate – the parameter η controlling how fast the parameters of the learner are updated

In a NN the weights, on which depend the strength of the response of an activated node, are adjourned by back-propagation

For NNs η is one of the crucial parameters in the search of optimality

Advanced techniques have been devised to overcome the difficulty. These include slowly decreasing η, scheduled modulations in η, momentum, etcetera.



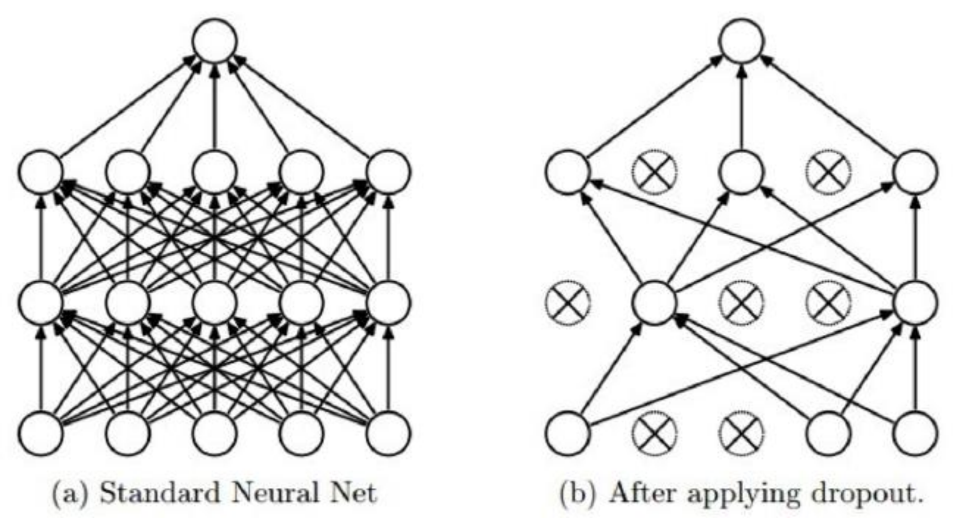
Regularization

We have already encountered the concept in general. In ANNs, regularization is similarly applied by adding to the loss a penalty term

- L1 loss:
- L2 loss: (AKA "weight decay")

A different method is called "dropout": during training, a random set of nodes is removed at each pass.

- prevents collective effects conditioning the training



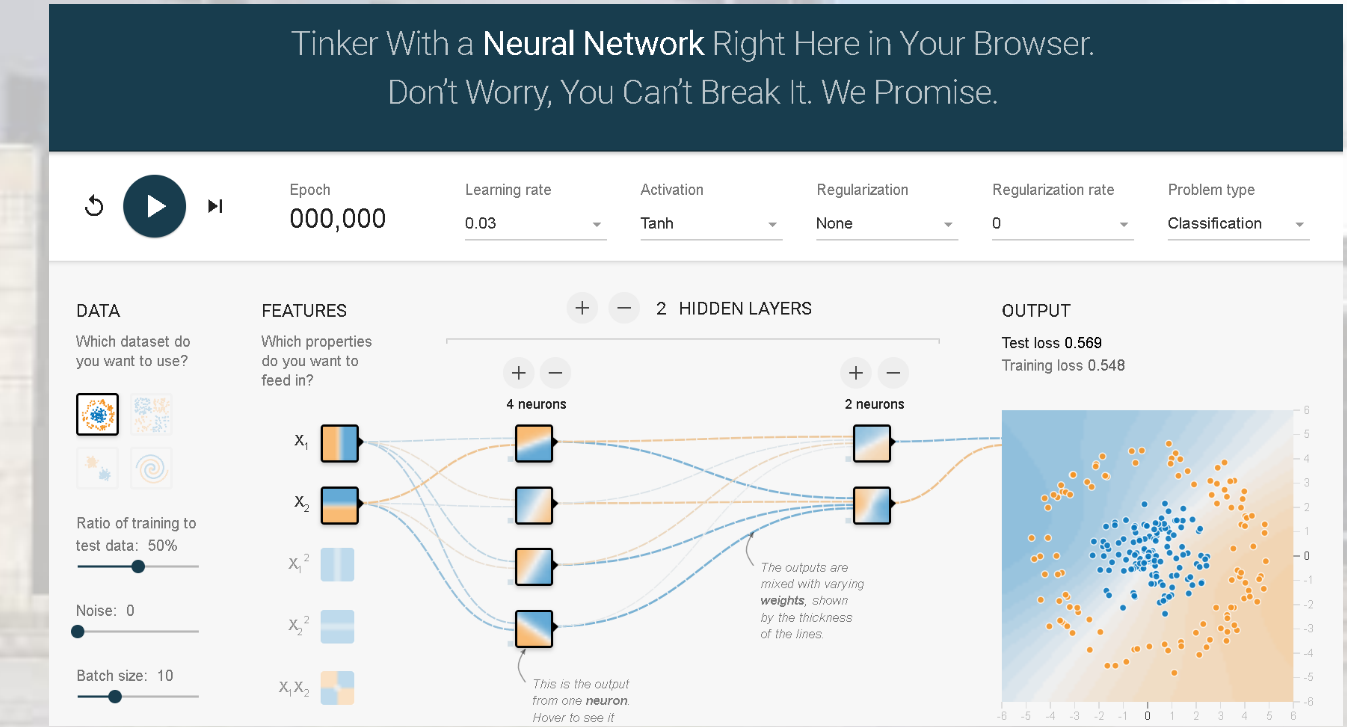
[M.Kagan; see arxiv:1207.0580]

Playing with NNs

Let us play a little

A wonderful web page allows us to fiddle with the architecture and hyperparameters of a NN, training it to solve simple 2D problems.

https://playground.tensorflow.org/



Tinker With a **Neural Network** Right Here in Your Browser.
Don't Worry, You Can't Break It. We Promise.

Epoch 000,000

Learning rate 0.03

Activation Tanh

Regularization None

Regularization rate 0

Problem type Classification

Ratio of training to test data: 50%

Noise: 0

Batch size: 10

Test loss 0.569

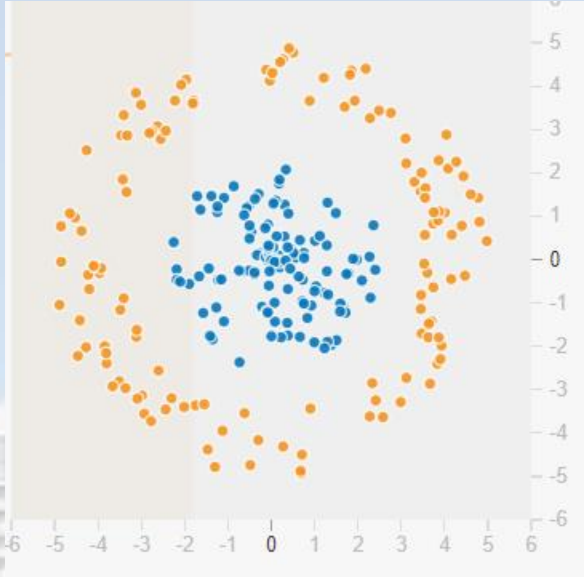
Training loss 0.548

The outputs are mixed with varying **weights**, shown by the thickness of the lines.

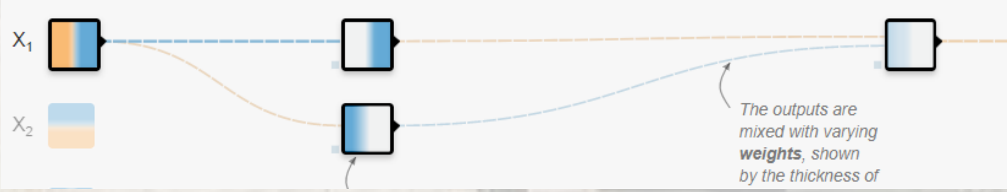
This is the output from one **neuron**. Hover to see it

Exercises with simple NNs

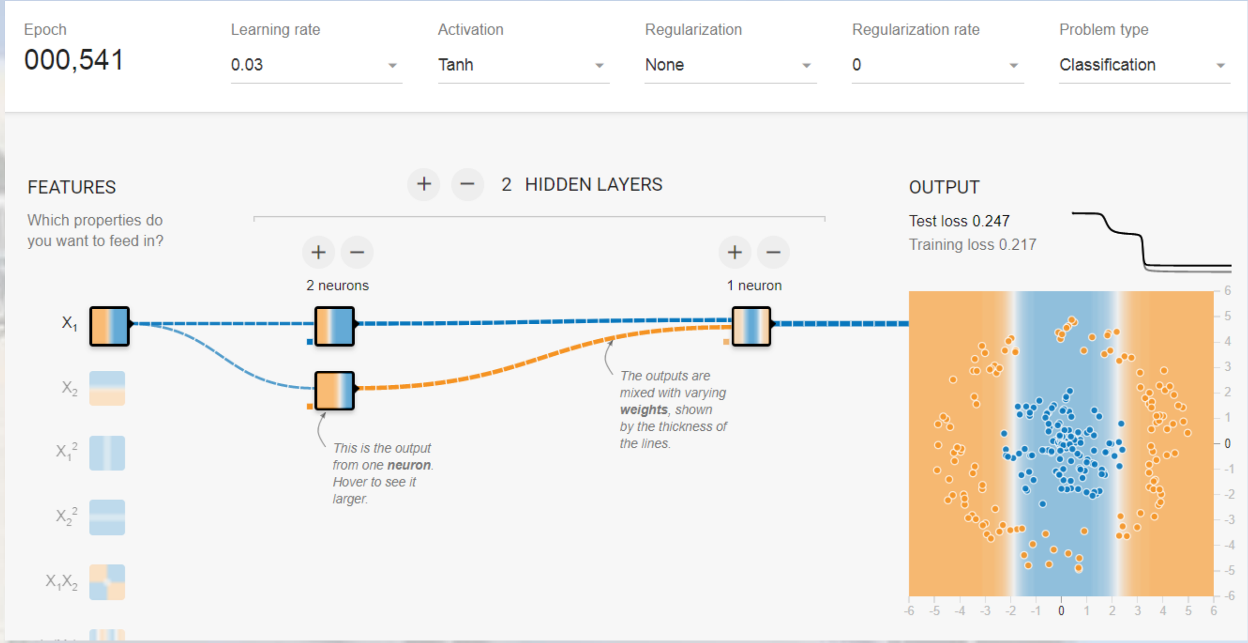
1) We take the first proposed dataset in the web site



What will be able to do if we use only one input (e.g. the first one from top) ?



The outputs are mixed with varying **weights**, shown by the thickness of



Epoch 000,541

Learning rate 0.03

Activation Tanh

Regularization None

Regularization rate 0

Problem type Classification

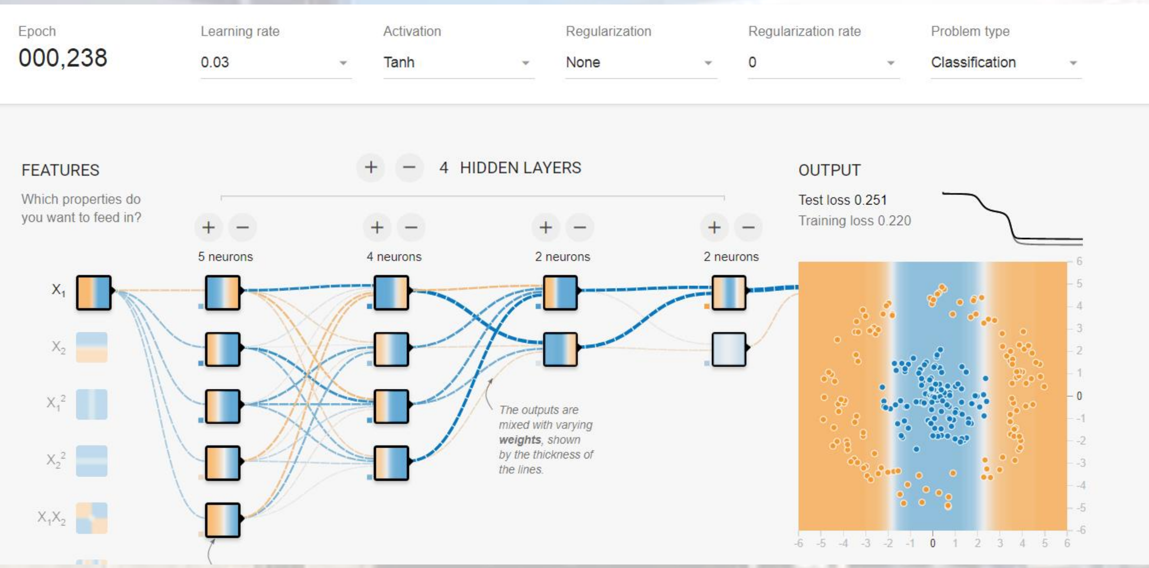
Test loss 0.247

Training loss 0.217

The outputs are mixed with varying **weights**, shown by the thickness of the lines.

This is the output from one **neuron**. Hover to see it larger.

Left: the simplest possible NN quickly converges to the optimal result



Epoch 000,238

Learning rate 0.03

Activation Tanh

Regularization None

Regularization rate 0

Problem type Classification

Test loss 0.251

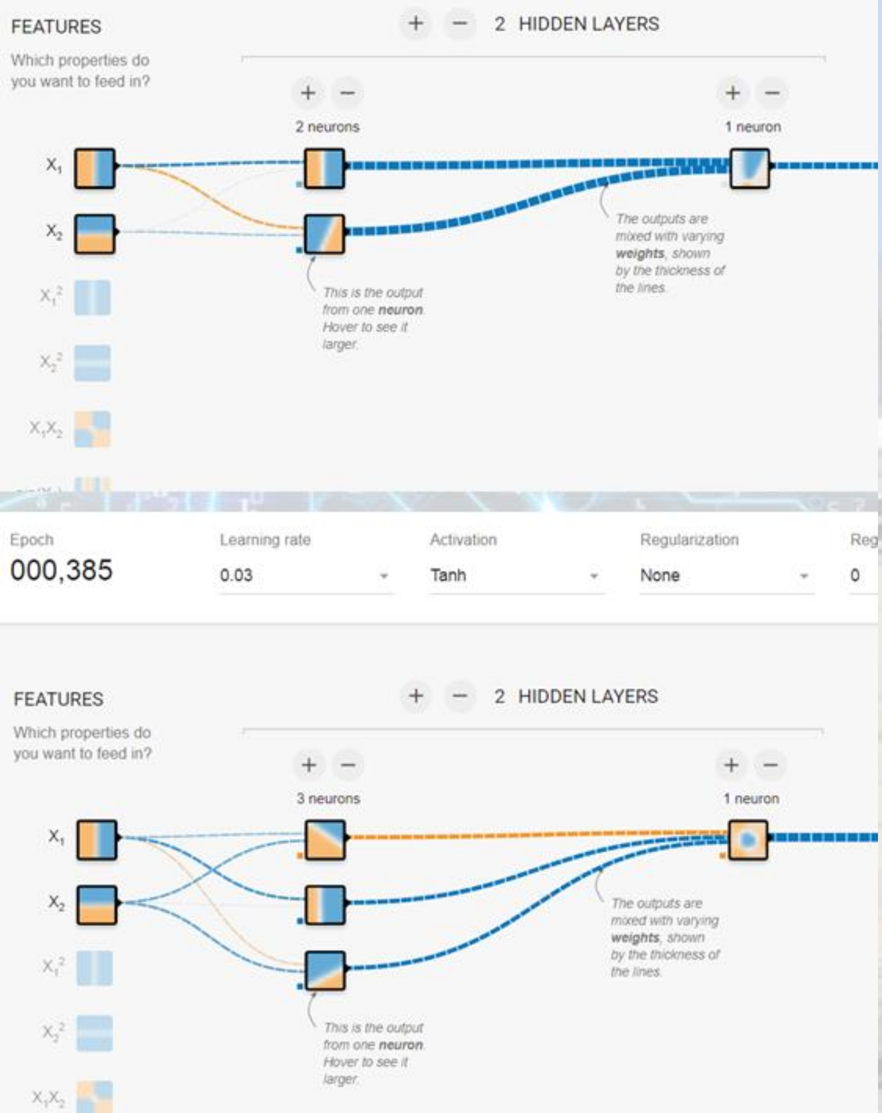
Training loss 0.220

The outputs are mixed with varying **weights**, shown by the thickness of the lines.

Right: a much more complex network does not perform any better, but takes much longer and requires tuning

→ Answer: you can't do better than about 0.25, independently of the chosen architecture / hyperparameters

2) What if we use both the x and y inputs? What changes when we go from a 2-nodes hidden layer to one with more ?



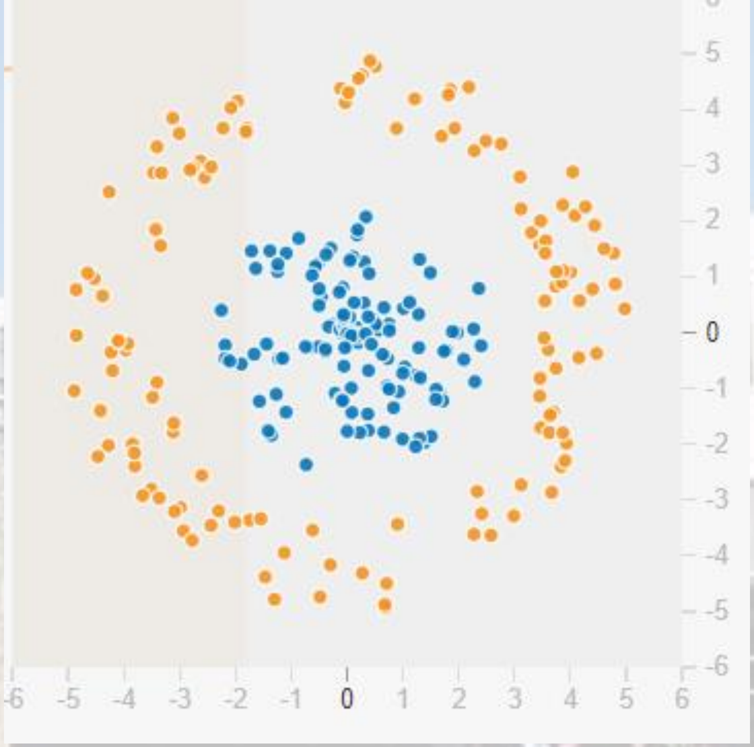
This is the output from one **neuron**. Hover to see it larger.

The outputs are mixed with varying **weights**, shown by the thickness of the lines.

Epoch	Learning rate	Activation	Regularization	Reg
000,385	0.03	Tanh	None	0

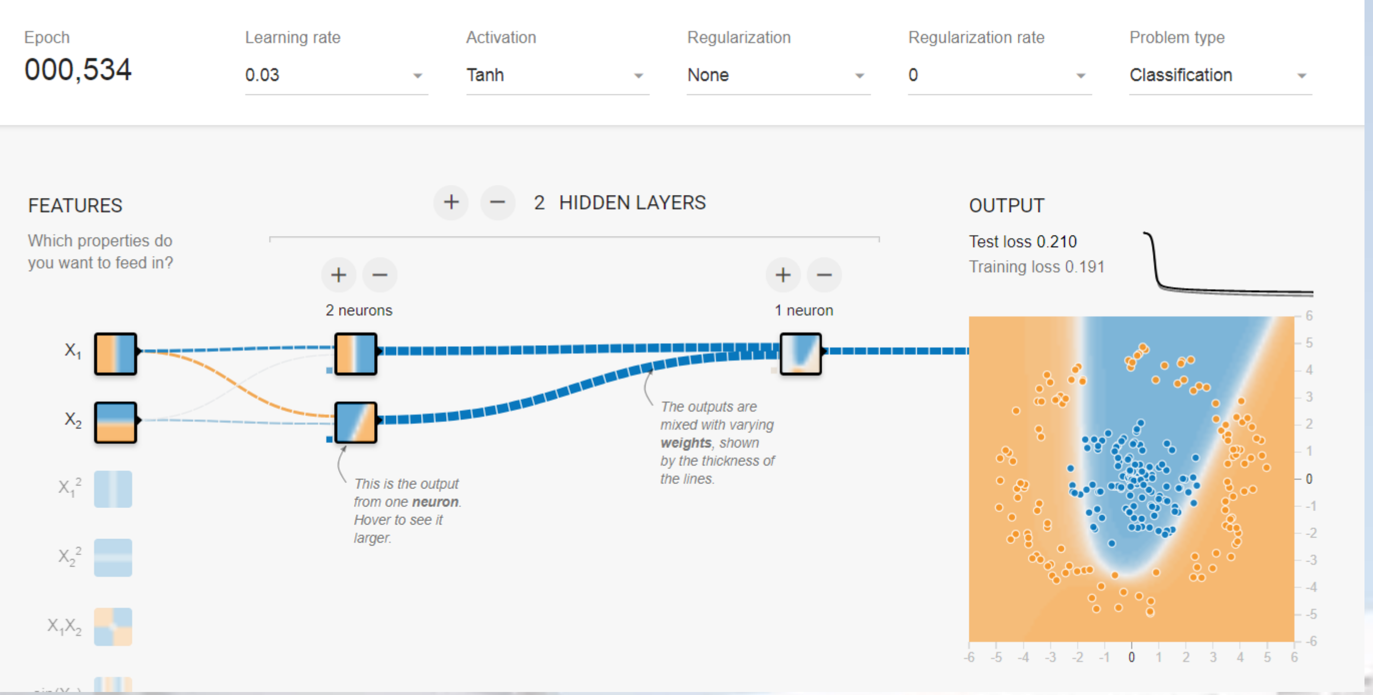
This is the output from one **neuron**. Hover to see it larger.

The outputs are mixed with varying **weights**, shown by the thickness of the lines.



Top: two nodes in the hidden layer. The test loss is 0.21

Bottom: three nodes vastly outperform the 2-node NN; the classification becomes perfect in this case



Epoch 000,534

Learning rate 0.03

Activation Tanh

Regularization None

Regularization rate 0

Problem type Classification

FEATURES

Which properties do you want to feed in?

2 HIDDEN LAYERS

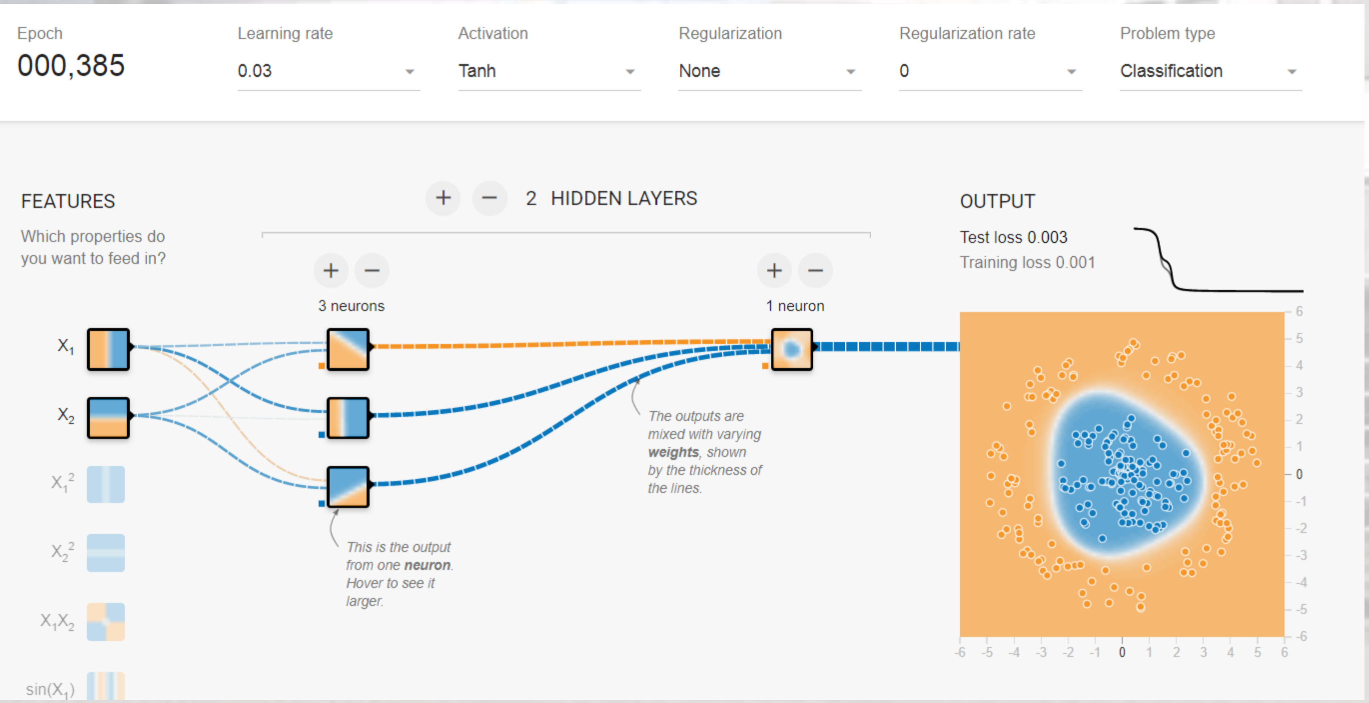
OUTPUT

Test loss 0.210

Training loss 0.191

*This is the output from one **neuron**. Hover to see it larger.*

*The outputs are mixed with varying **weights**, shown by the thickness of the lines.*



Epoch 000,385

Learning rate 0.03

Activation Tanh

Regularization None

Regularization rate 0

Problem type Classification

FEATURES

Which properties do you want to feed in?

2 HIDDEN LAYERS

OUTPUT

Test loss 0.003

Training loss 0.001

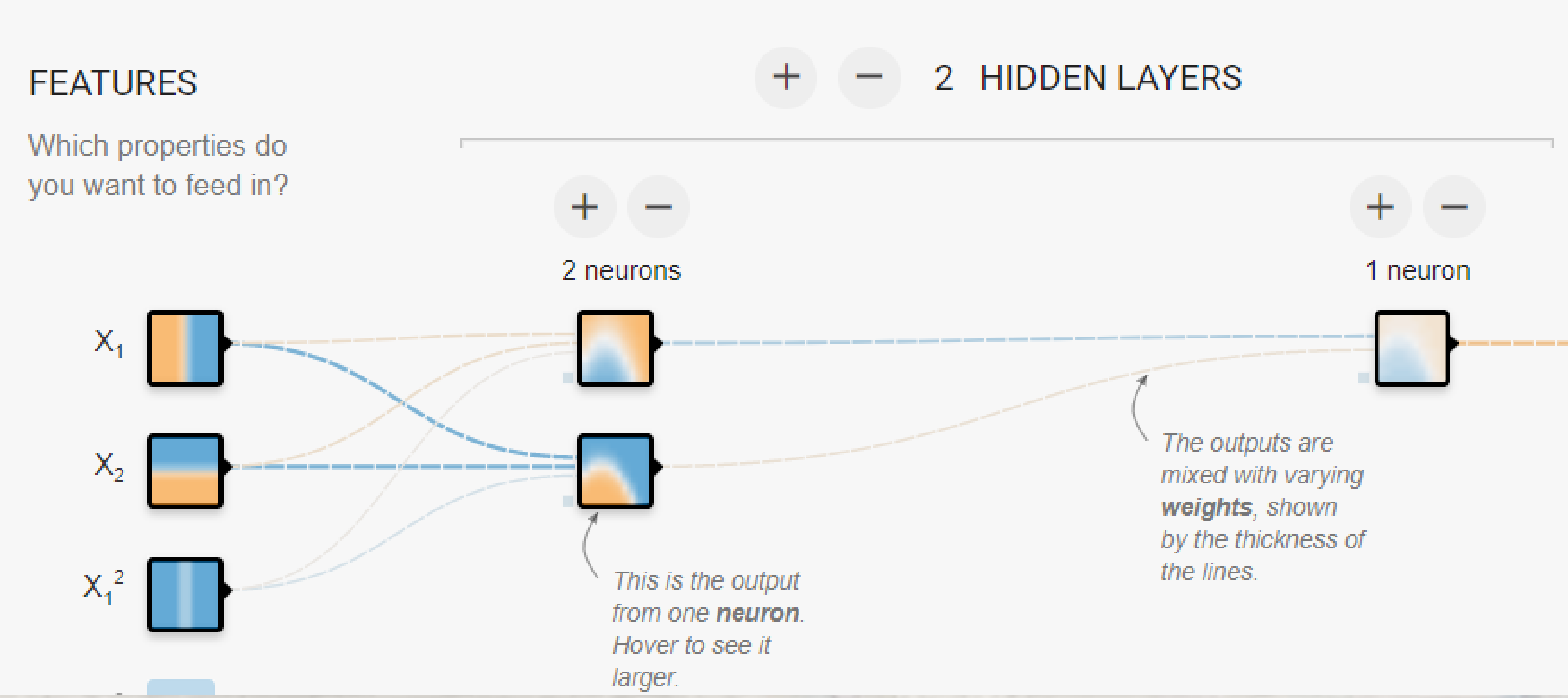
*This is the output from one **neuron**. Hover to see it larger.*

*The outputs are mixed with varying **weights**, shown by the thickness of the lines.*

2b) What happens if we add more inputs, but keep just two nodes in the hidden layer?

→ A 3/2/1 architecture, and let's use a tanh activation, with 0.03 learning rate.

What test loss can you get ? What inputs did you pick?



FEATURES

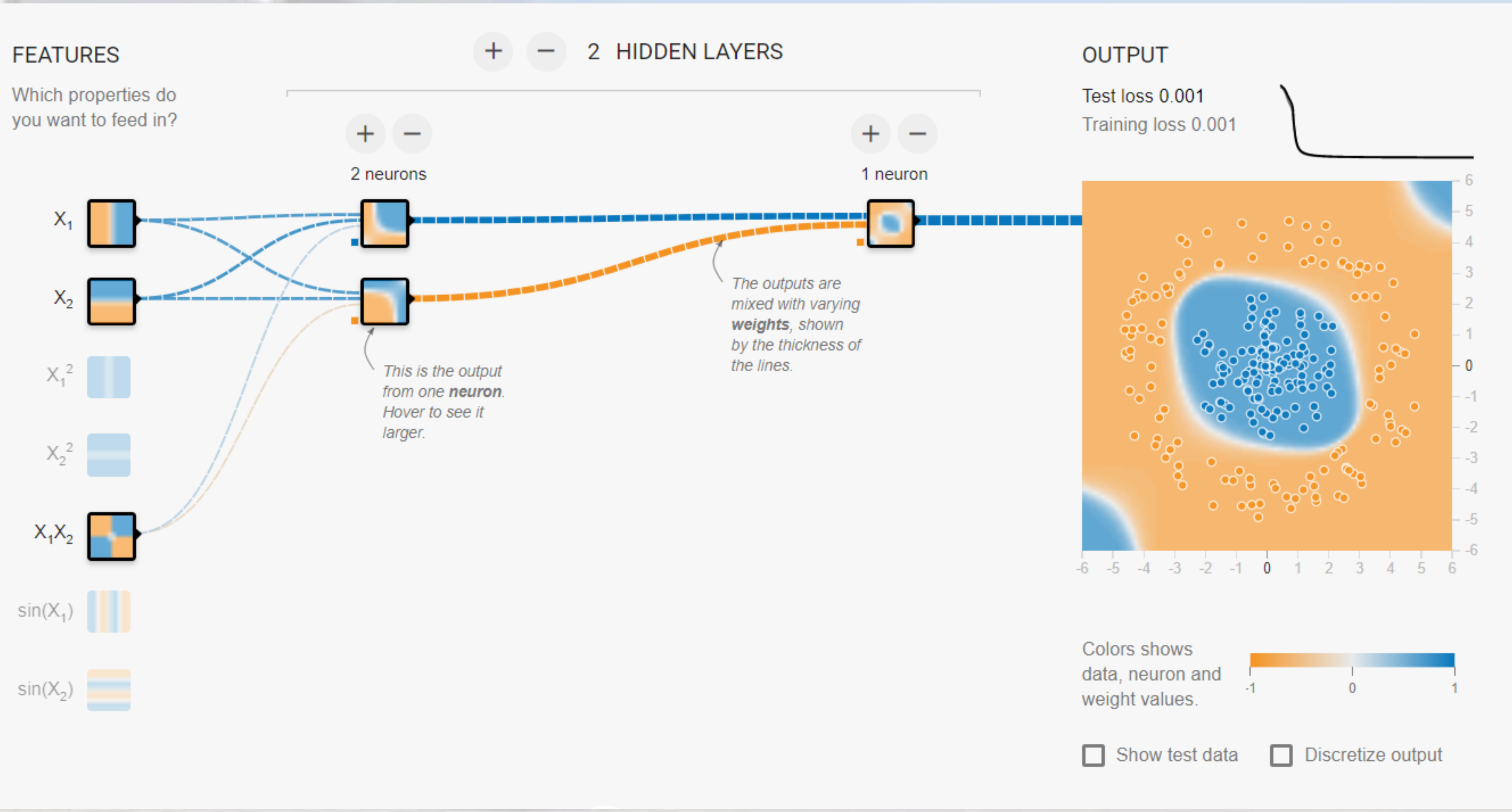
Which properties do you want to feed in?

2 HIDDEN LAYERS

This is the output from one **neuron**. Hover to see it larger.

The outputs are mixed with varying **weights**, shown by the thickness of the lines.

With 3 inputs the classification becomes perfect even with only two nodes in the hidden layer. Why?



FEATURES

Which properties do you want to feed in?

+ − 2 HIDDEN LAYERS

OUTPUT

Test loss 0.001

Training loss 0.001

This is the output from one ***neuron****. Hover to see it larger.*

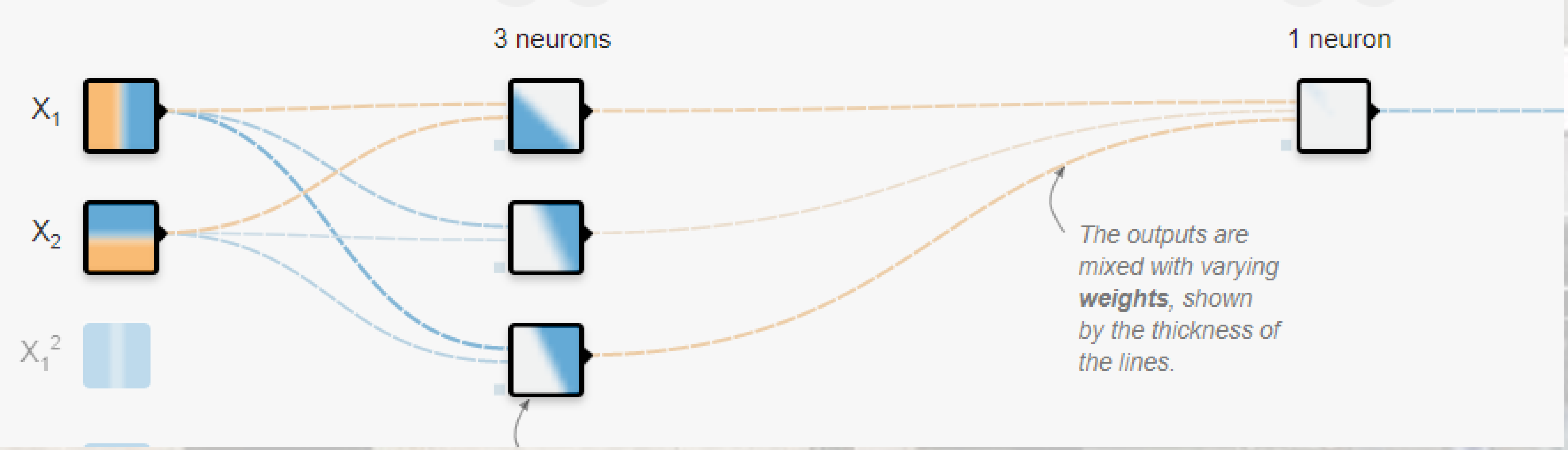
The outputs are mixed with varying ***weights****, shown by the thickness of the lines.*

Colors shows data, neuron and weight values.

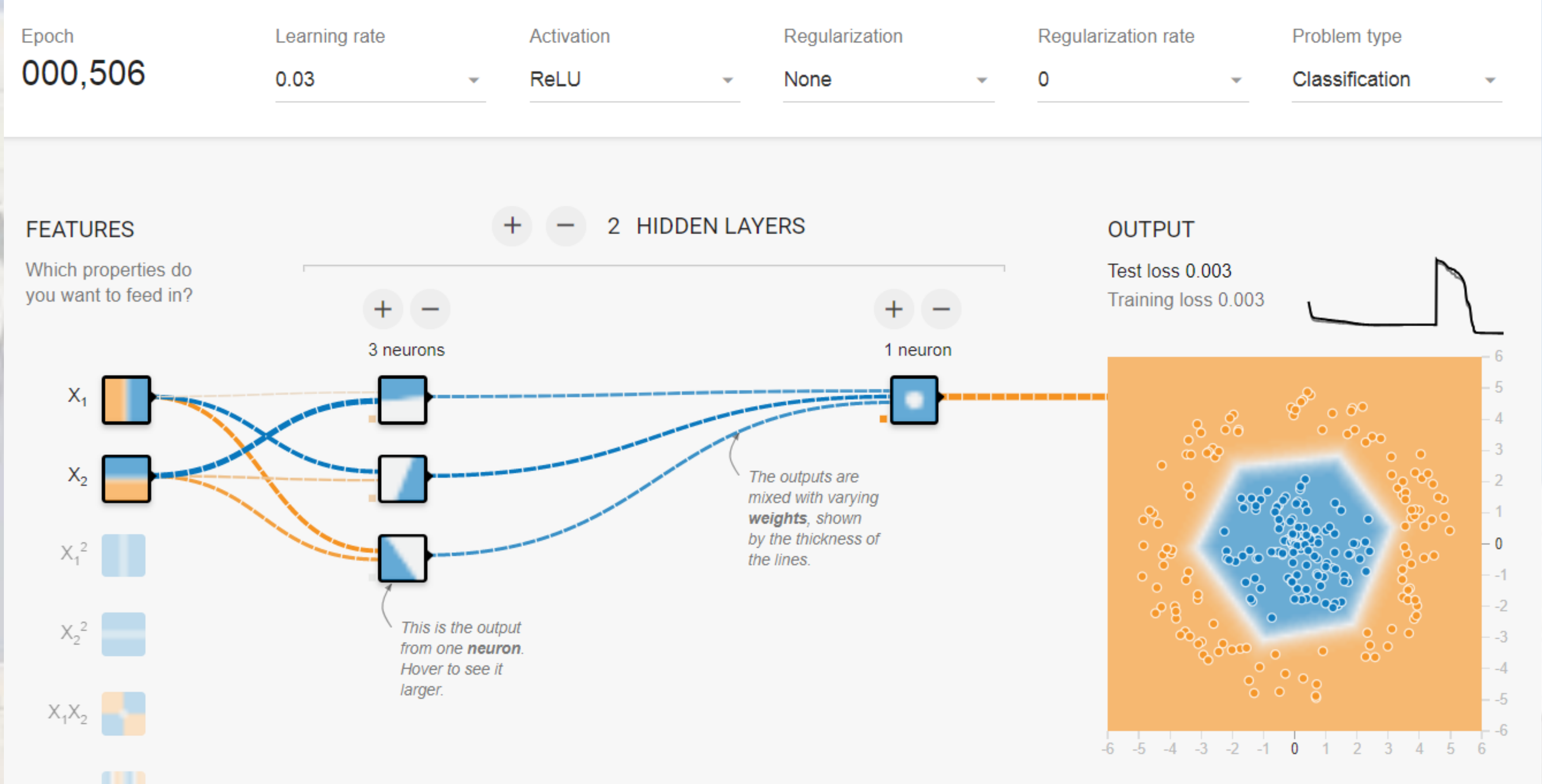
□ Show test data □ Discretize output

3) Now let's try using two inputs and three nodes in the HL, but change from tanh to relu the activation function.

What do we expect will change?



The outputs are mixed with varying **weights**, shown by the thickness of the lines.



Epoch 000,506

Learning rate 0.03

Activation ReLU

Regularization None

Regularization rate 0

Problem type Classification

FEATURES

Which properties do you want to feed in?

2 HIDDEN LAYERS

3 neurons

1 neuron

OUTPUT

Test loss 0.003

Training loss 0.003

*This is the output from one **neuron**. Hover to see it larger.*

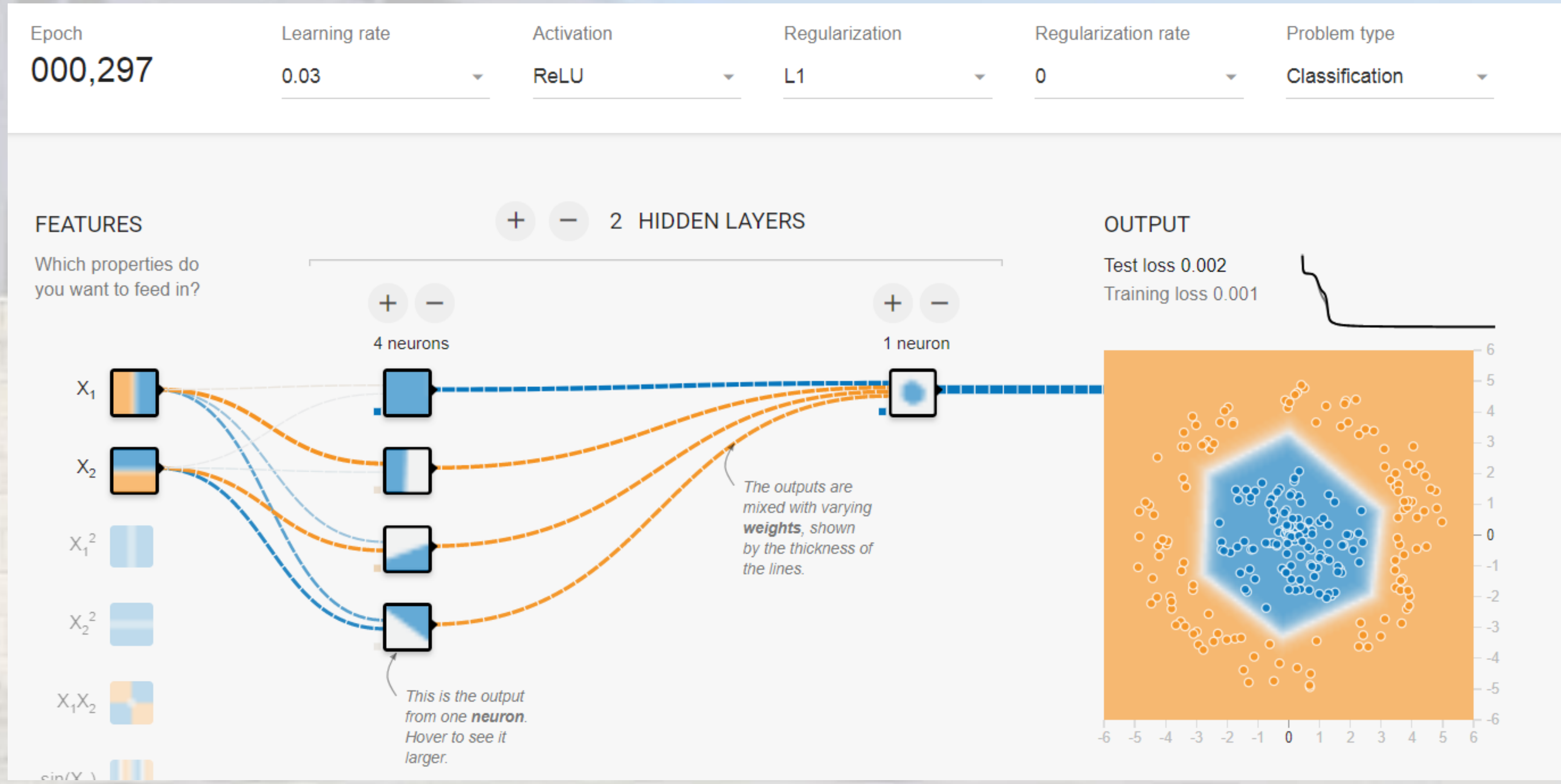
*The outputs are mixed with varying **weights**, shown by the thickness of the lines.*

We get a more spikey decision boundary, an effect of the sharp nature of "relu"

Question: why 6 sides? How many edges do you expect in the decision boundary if you use 4 neurons?

8? Not necessarily!...

Why?



Epoch 000,297

Learning rate 0.03

Activation ReLU

Regularization L1

Regularization rate 0

Problem type Classification

FEATURES

Which properties do you want to feed in?

2 HIDDEN LAYERS

4 neurons

1 neuron

OUTPUT

Test loss 0.002

Training loss 0.001

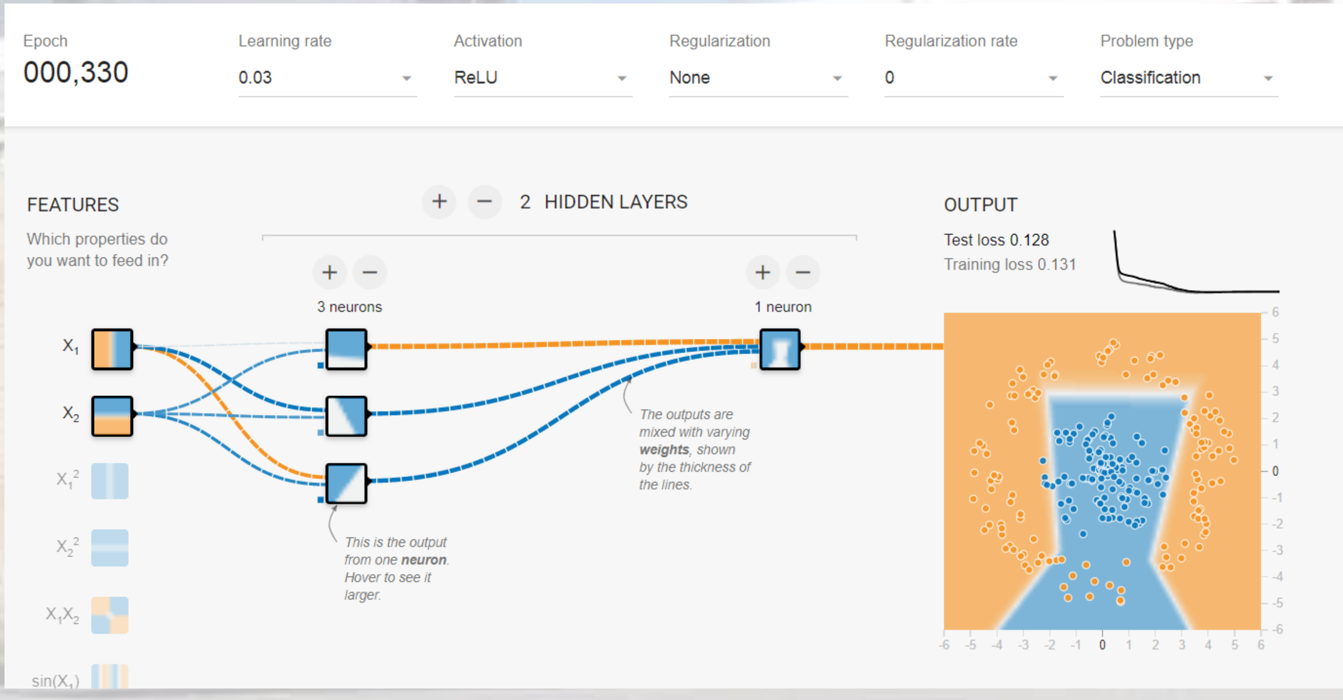
*The outputs are mixed with varying **weights**, shown by the thickness of the lines.*

*This is the output from one **neuron**. Hover to see it larger.*

Here is a pathological case:

we seem to fail to obtain the wanted result from these three neurons!
Maybe we can intervene to drift away from the local minimum?

The application allows you to do this manually, by clicking on the link connecting the first hidden node from the top to the output one, modifying the weight



Epoch 000,330

Learning rate 0.03

Activation ReLU

Regularization None

Regularization rate 0

Problem type Classification

Test loss 0.128

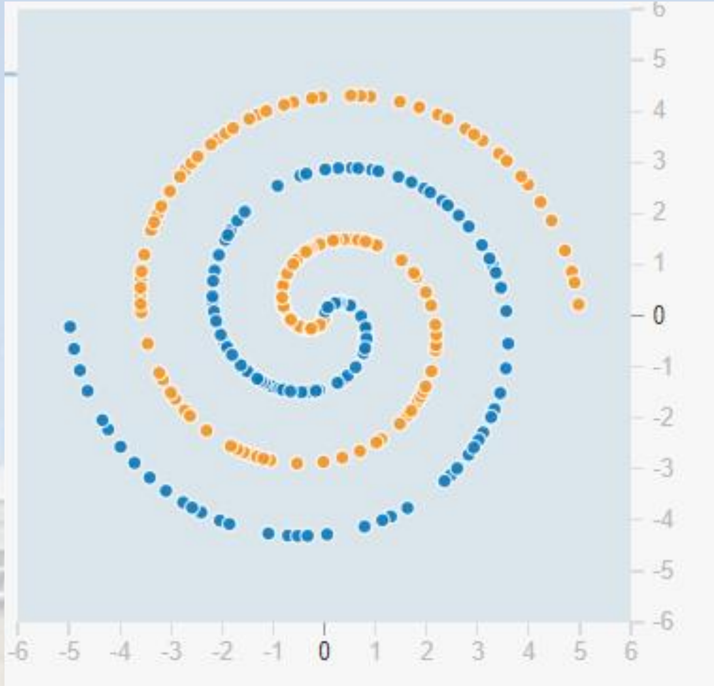
Training loss 0.131

The outputs are mixed with varying **weights**, shown by the thickness of the lines.

This is the output from one **neuron**. Hover to see it larger.

Let us go back to the **tanh** activation

And take the 4th dataset:



This problem is much more complex. We can start with only two inputs x,y

Try this at home. What result do you get ? Do you need many layers? How long does the training requires to converge ?

Using only x,y inputs it is harder to get to a good result...

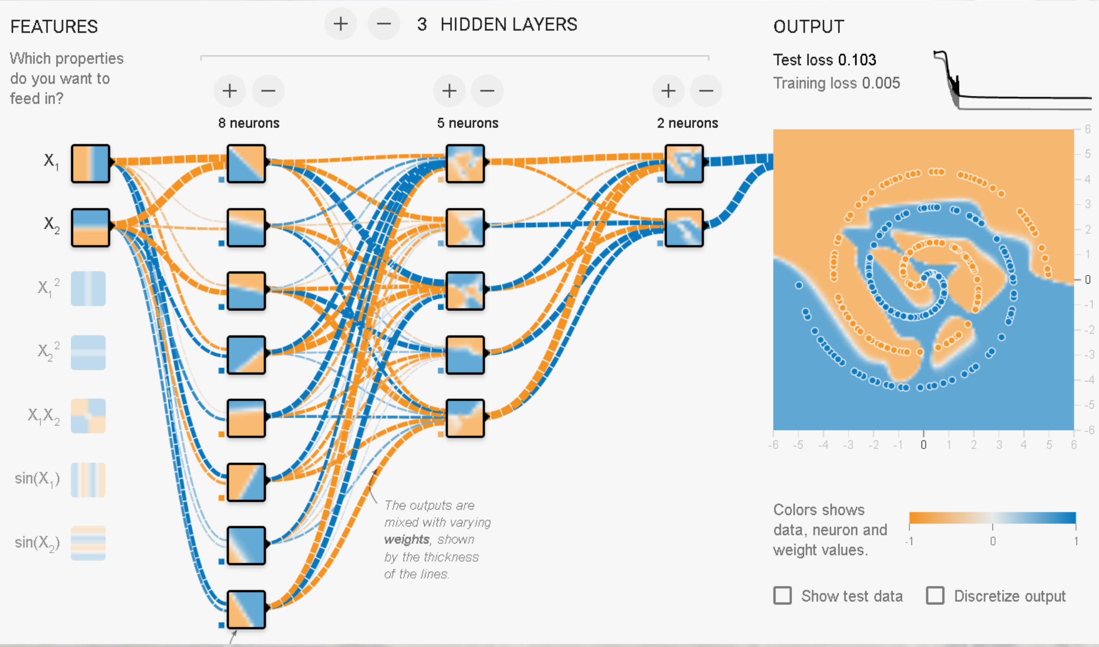
But why? After all, {x,y} are a complete set – a sufficient statistic!

By sweating over this simple problem you will come to learn that in order to give the NN the needed flexibility to learn the correct non-linear combinations of the given features, you need to train for a much longer number of epochs.

Note that a NN can perfectly well emulate, with appropriate training, whatever combination of the inputs, without your need to input it to it.

But this points clearly to a conclusion:

A bit of feature engineering is worth many long hours spent training a very flexible network!



FEATURES

Which properties do you want to feed in?

3 HIDDEN LAYERS

OUTPUT

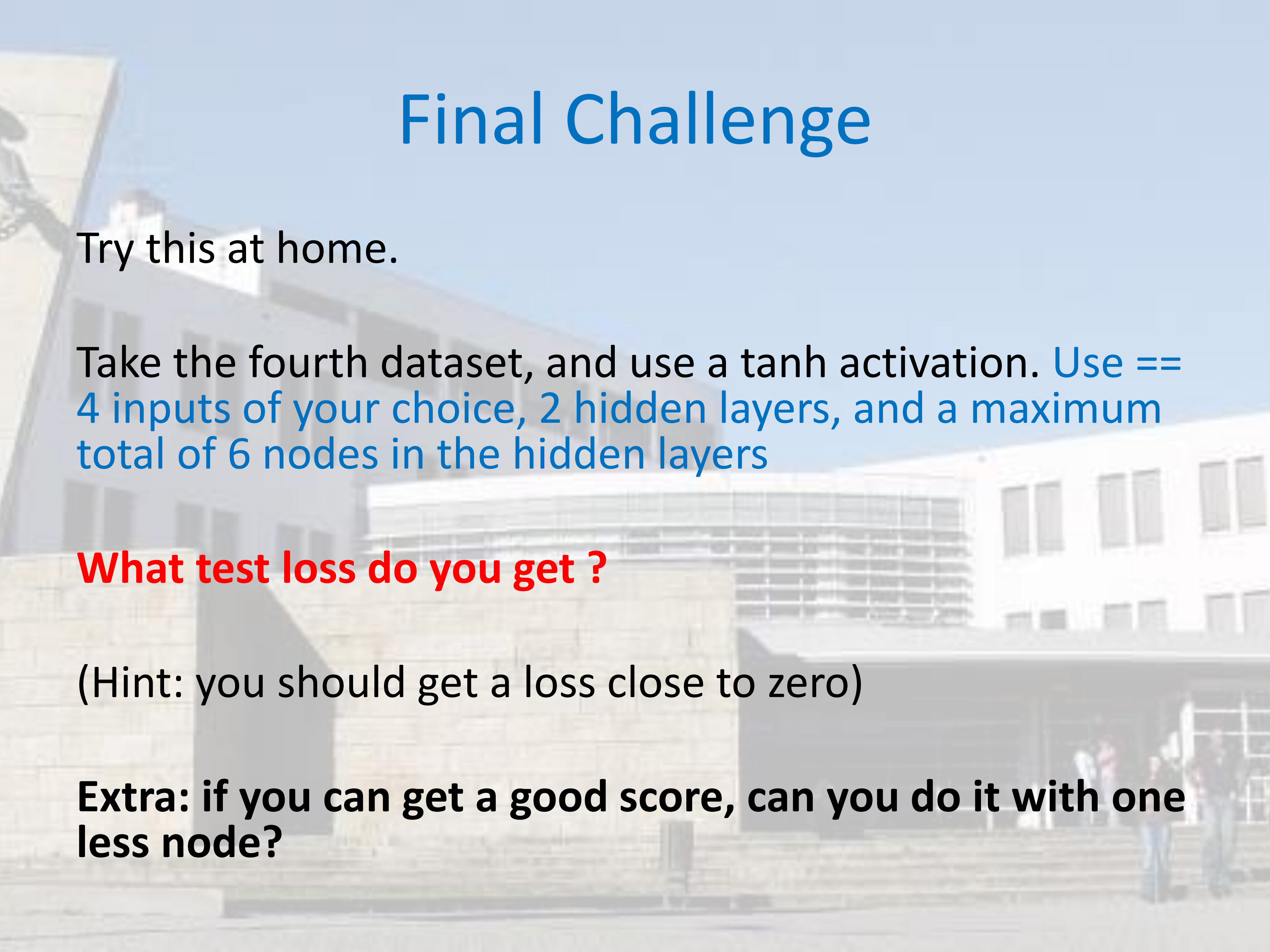
Test loss 0.103

Training loss 0.005

The outputs are mixed with varying **weights**, shown by the thickness of the lines.

Colors shows data, neuron and weight values.

□ Show test data □ Discretize output



Final Challenge

Try this at home.

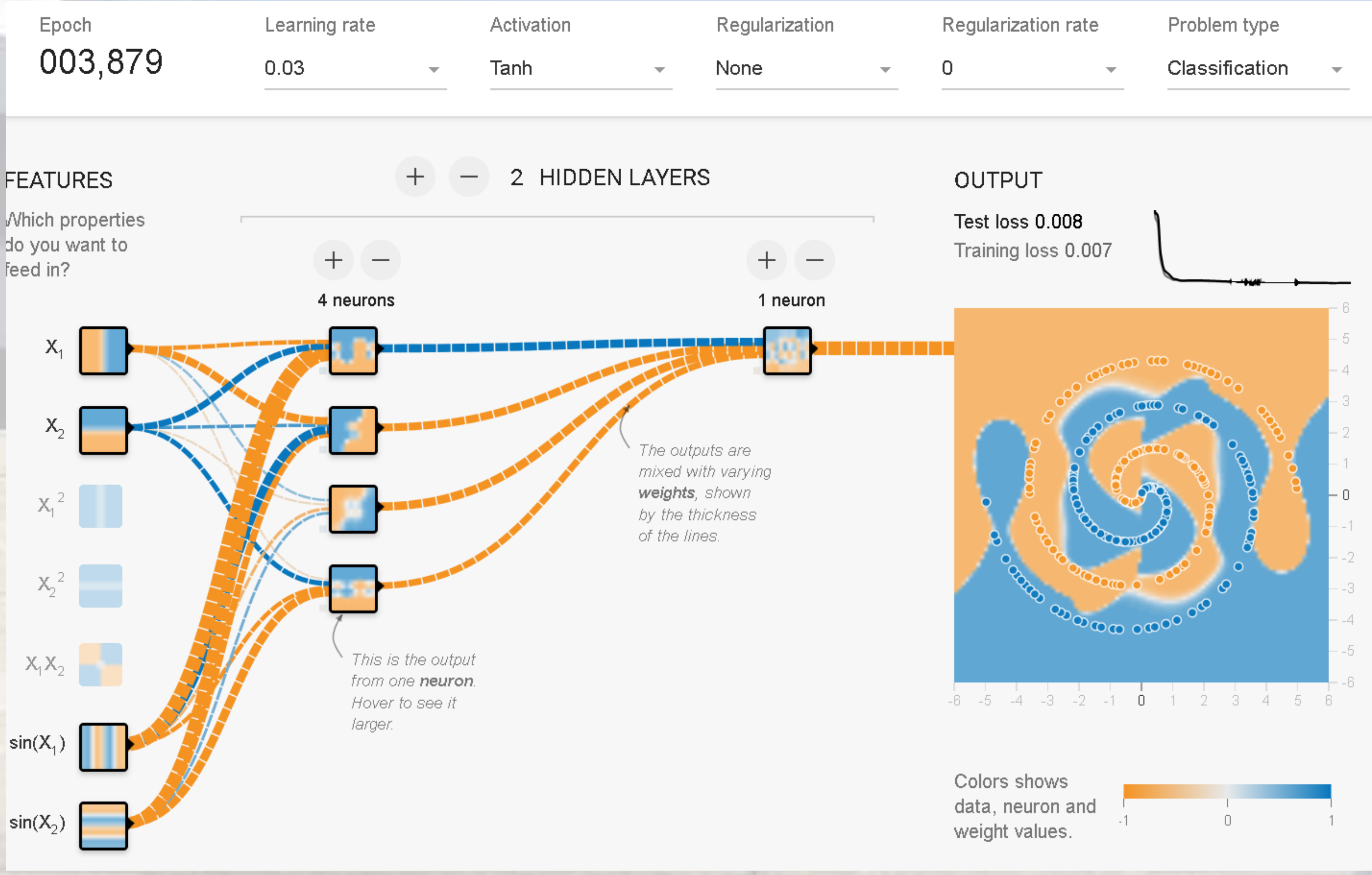
Take the fourth dataset, and use a tanh activation. Use == 4 inputs of your choice, 2 hidden layers, and a maximum total of 6 nodes in the hidden layers

What test loss do you get ?

(Hint: you should get a loss close to zero)

Extra: if you can get a good score, can you do it with one less node?

My solution



Epoch 003,879

Learning rate 0.03

Activation Tanh

Regularization None

Regularization rate 0

Problem type Classification

FEATURES

Which properties do you want to feed in?

2 HIDDEN LAYERS

4 neurons

1 neuron

OUTPUT

Test loss 0.008

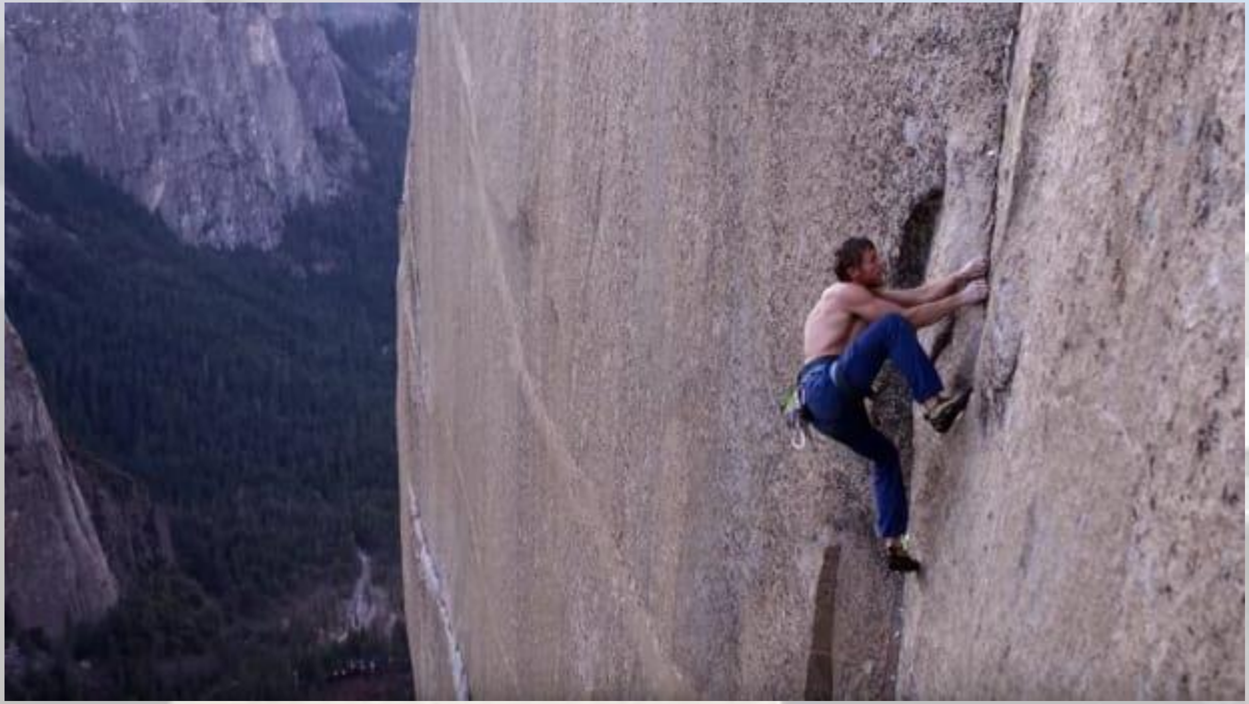
Training loss 0.007

The outputs are mixed with varying **weights**, shown by the thickness of the lines.

This is the output from one **neuron**. Hover to see it larger.

Colors shows data, neuron and weight values.

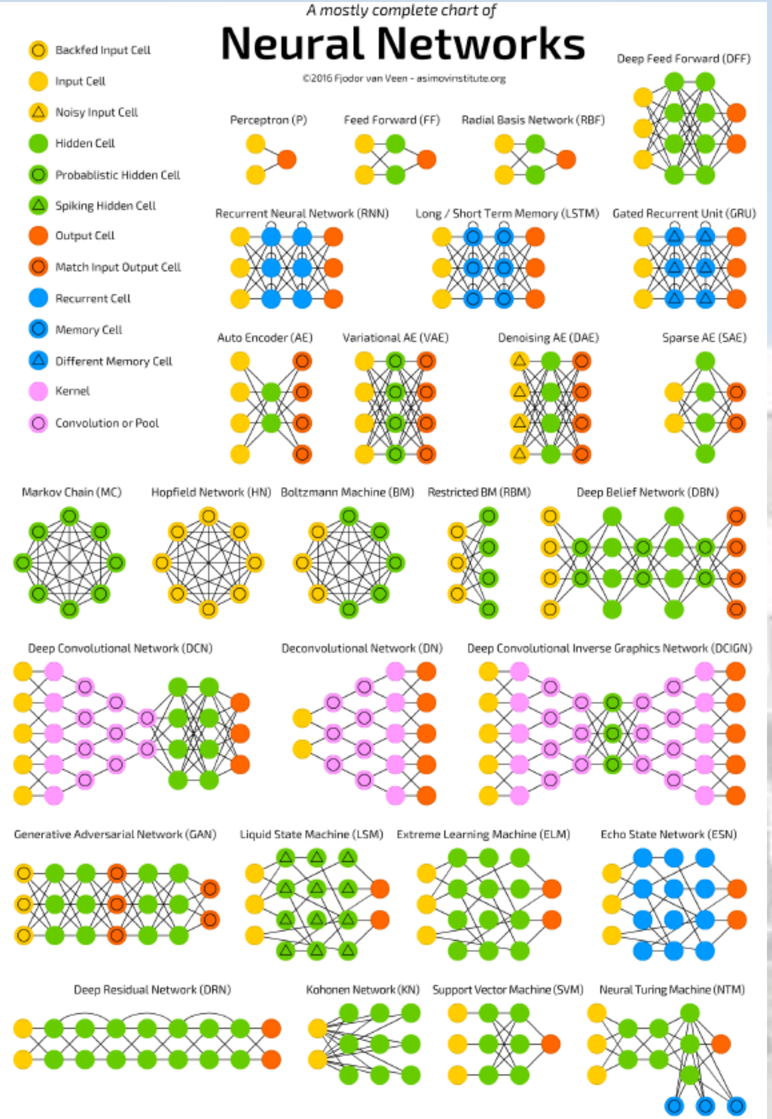
ADVANCED TECHNIQUES



NNs everywhere

The ascent of deep learning in the XXI century has brought to the design and development of many specialized architectures optimized to different tasks

We can give only a peek, as it is more fruitful to make sure you have gotten the gist of the basic concepts


A mostly complete chart of
Neural Networks
©2016 Fjodor van Veen - asimovinstitute.org
Backfed Input Cell
Input Cell
Noisy Input Cell
Hidden Cell
Probablistic Hidden Cell
Spiking Hidden Cell
Output Cell
Match Input Output Cell
Recurrent Cell
Memory Cell
Different Memory Cell
Kernel
Convolution or Pool
Perceptron (P)
Feed Forward (FF)
Radial Basis Network (RBF)
Deep Feed Forward (DFF)
Recurrent Neural Network (RNN)
Long / Short Term Memory (LSTM)
Gated Recurrent Unit (GRU)
Auto Encoder (AE)
Variational AE (VAE)
Denoising AE (DAE)
Sparse AE (SAE)
Markov Chain (MC)
Hopfield Network (HN)
Boltzmann Machine (BM)
Restricted BM (RBM)
Deep Belief Network (DBN)
Deep Convolutional Network (DCN)
Deconvolutional Network (DN)
Deep Convolutional Inverse Graphics Network (DCIGN)
Generative Adversarial Network (GAN)
Liquid State Machine (LSM)
Extreme Learning Machine (ELM)
Echo State Network (ESN)
Deep Residual Network (DRN)
Kohonen Network (KN)
Support Vector Machine (SVM)
Neural Turing Machine (NTM)

http://www.asimovinstitute.org/neural-network-zoo/

The multi-body problem

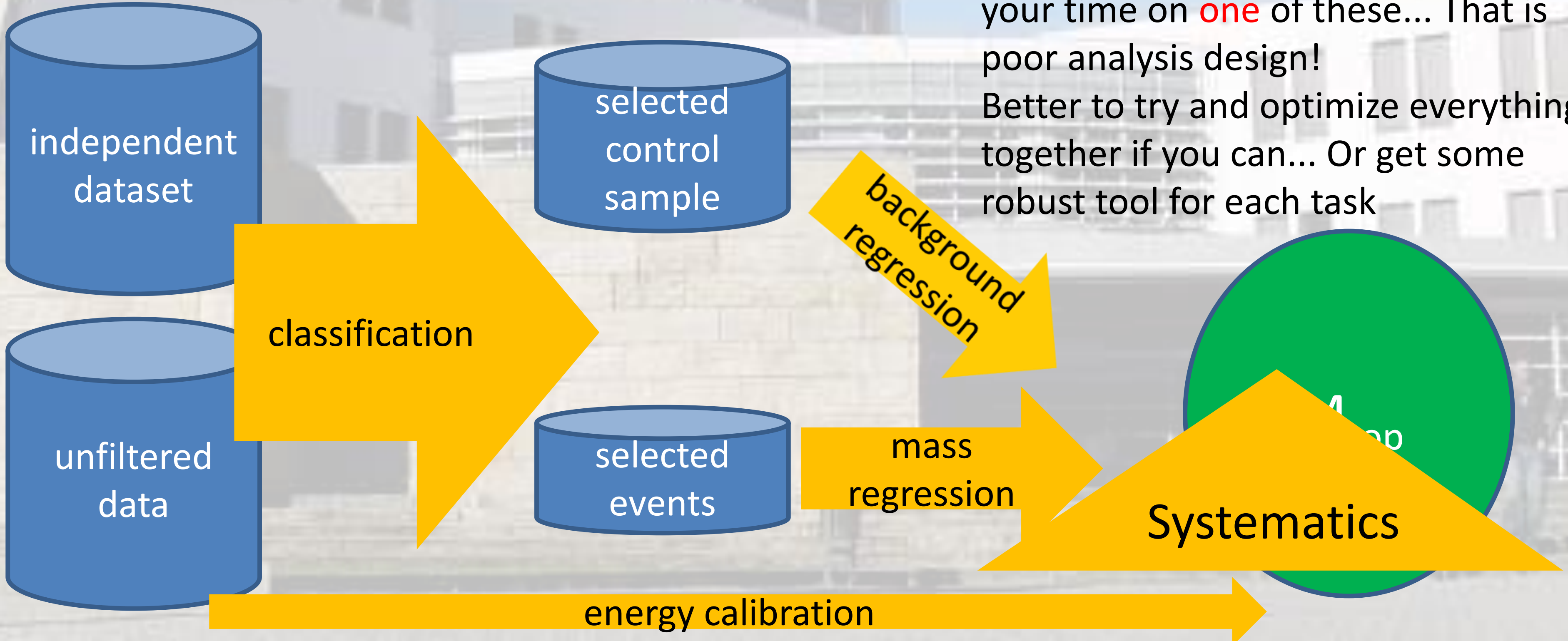
The more one works on a NN, the better results can be achieved (up to a limit given by the NP lemma, which is however not usually achieved)

So is the message "take a problem and work at it very hard"? Not necessarily...

Take e.g. a HEP analysis where e.g. we want to measure the mass of a particle (e.g. top quark). There are multi-variate problems everywhere:

It makes little sense to spend all your time on one of these... That is poor analysis design!

Better to try and optimize everything together if you can... Or get some robust tool for each task



independent dataset

unfiltered data

classification

selected control sample

selected events

background regression

mass regression

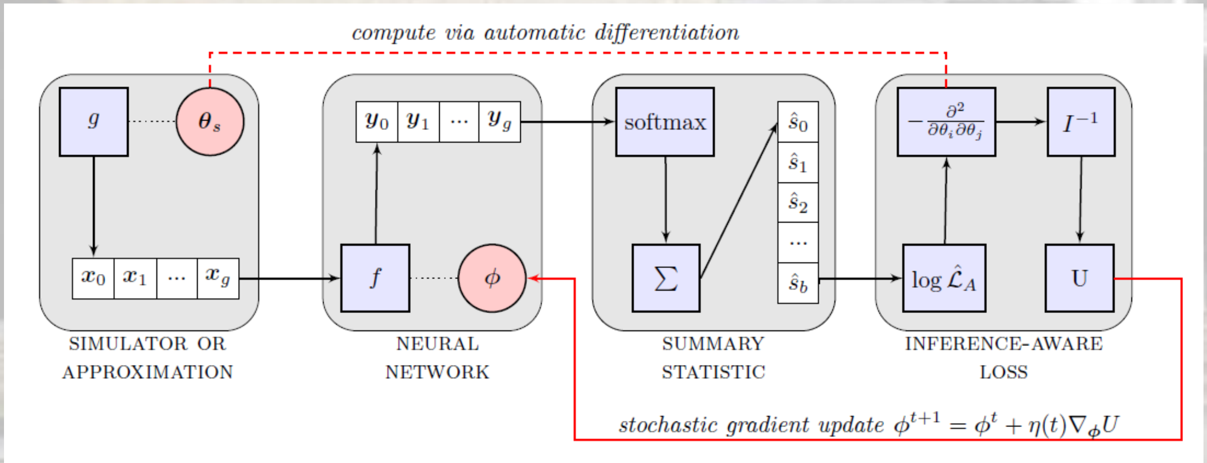
Systematics

energy calibration

INFERNO

One common problem of many analyses searching for a signal: the classification of S vs N is normally "optimized" without accounting for systematic effects, which are assessed later.

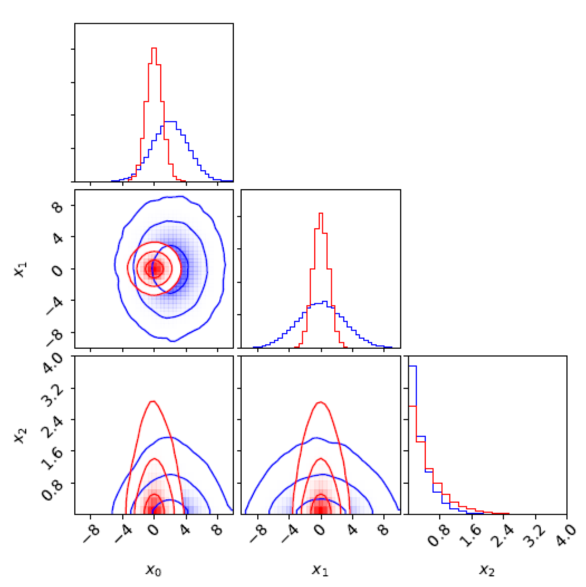
The INFERNO algorithm (from "***Infer**ence-aware **N**eural **O**ptimization*") proposes a NN architecture that incorporates the evaluation of systematic effects on the estimate of the parameter of interest (signal strength) in the definition of the loss.
The loss can be formulated as the expected sqrt(variance) of the likelihood fit that extracts the signal from the classifier output
→ the classification becomes robust WRT systematics and strongly reduces their impact



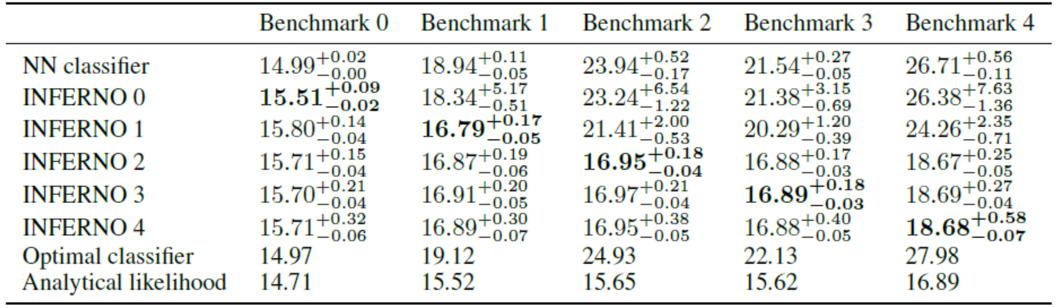
compute via automatic differentiation
g
θ_s
x_0 x_1 ... x_g
SIMULATOR OR APPROXIMATION
y_0 y_1 ... y_g
f
ϕ
NEURAL NETWORK
softmax
Σ
$\hat{s}_0$ $\hat{s}_1$ $\hat{s}_2$ ... $\hat{s}_b$
SUMMARY STATISTIC
$-\frac{\partial^2}{\partial\theta_i\partial\theta_j}$
I^{-1}
$\log \hat{\mathcal{L}}_A$
U
INFERENCE-AWARE LOSS
stochastic gradient update $\phi^{t+1} = \phi^t + \eta(t)\nabla_\phi U$

Results on synthetic example

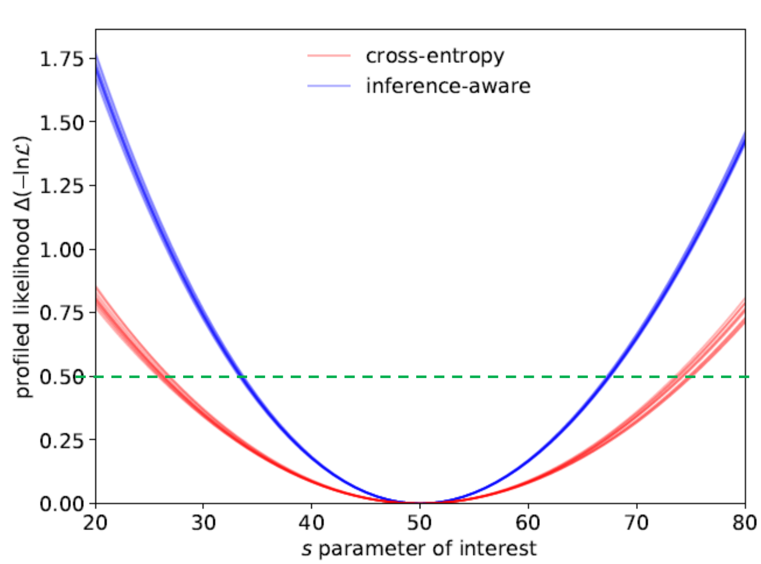
In a 3-D dataset of background plus small (50/1000) signal, where there are three nuisance parameters affecting the shape of the PDFs, the INFERNO classifier vastly outperforms the NN classifier on the quantity of interest



See for details arxiv:1806.04743 (to be published in Computer Physics Communications)



	Benchmark 0	Benchmark 1	Benchmark 2	Benchmark 3	Benchmark 4
NN classifier	$14.99^{+0.02}_{-0.00}$	$18.94^{+0.11}_{-0.05}$	$23.94^{+0.52}_{-0.17}$	$21.54^{+0.27}_{-0.05}$	$26.71^{+0.56}_{-0.11}$
INFERNO 0	$\mathbf{15.51^{+0.09}_{-0.02}}$	$18.34^{+5.17}_{-0.51}$	$23.24^{+6.54}_{-1.22}$	$21.38^{+3.15}_{-0.69}$	$26.38^{+7.63}_{-1.36}$
INFERNO 1	$15.80^{+0.14}_{-0.04}$	$\mathbf{16.79^{+0.17}_{-0.05}}$	$21.41^{+2.00}_{-0.53}$	$20.29^{+1.20}_{-0.39}$	$24.26^{+2.35}_{-0.71}$
INFERNO 2	$15.71^{+0.15}_{-0.04}$	$16.87^{+0.19}_{-0.06}$	$\mathbf{16.95^{+0.18}_{-0.04}}$	$16.88^{+0.17}_{-0.03}$	$18.67^{+0.25}_{-0.05}$
INFERNO 3	$15.70^{+0.21}_{-0.04}$	$16.91^{+0.20}_{-0.05}$	$16.97^{+0.21}_{-0.04}$	$\mathbf{16.89^{+0.18}_{-0.03}}$	$18.69^{+0.27}_{-0.04}$
INFERNO 4	$15.71^{+0.32}_{-0.06}$	$16.89^{+0.30}_{-0.07}$	$16.95^{+0.38}_{-0.05}$	$16.88^{+0.40}_{-0.05}$	$\mathbf{18.68^{+0.58}_{-0.07}}$
Optimal classifier	14.97	19.12	24.93	22.13	27.98
Analytical likelihood	14.71	15.52	15.65	15.62	16.89

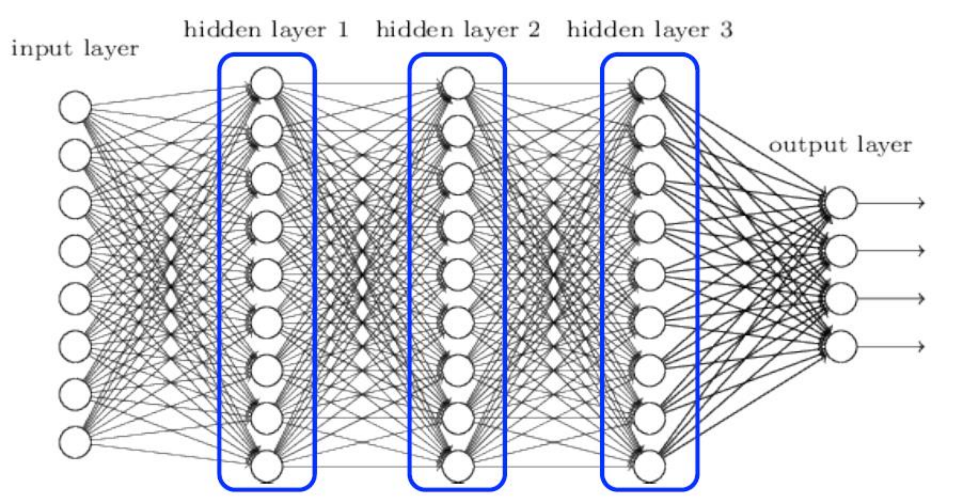


Deep neural networks

What is "deep"? Arguable, but already a network with >2 hidden layers can be enormously complex

DNNs are appropriate for very complex data. While a single hidden layer should suffice to produce arbitrarily complex functions, to do so the number of neurons has to grow exponentially → better to increase the number of layers, essentially factorizing the learning process

DNNs can be powerful but are usually difficult to train



Convolutional neural networks

CNNs are a specialized form of DNNs

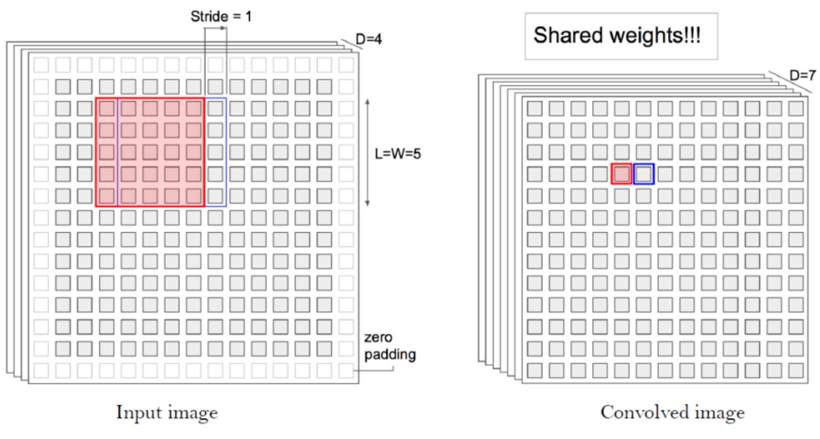
A very important commercial application is image recognition → used to drive cars, recognize faces, interpret content

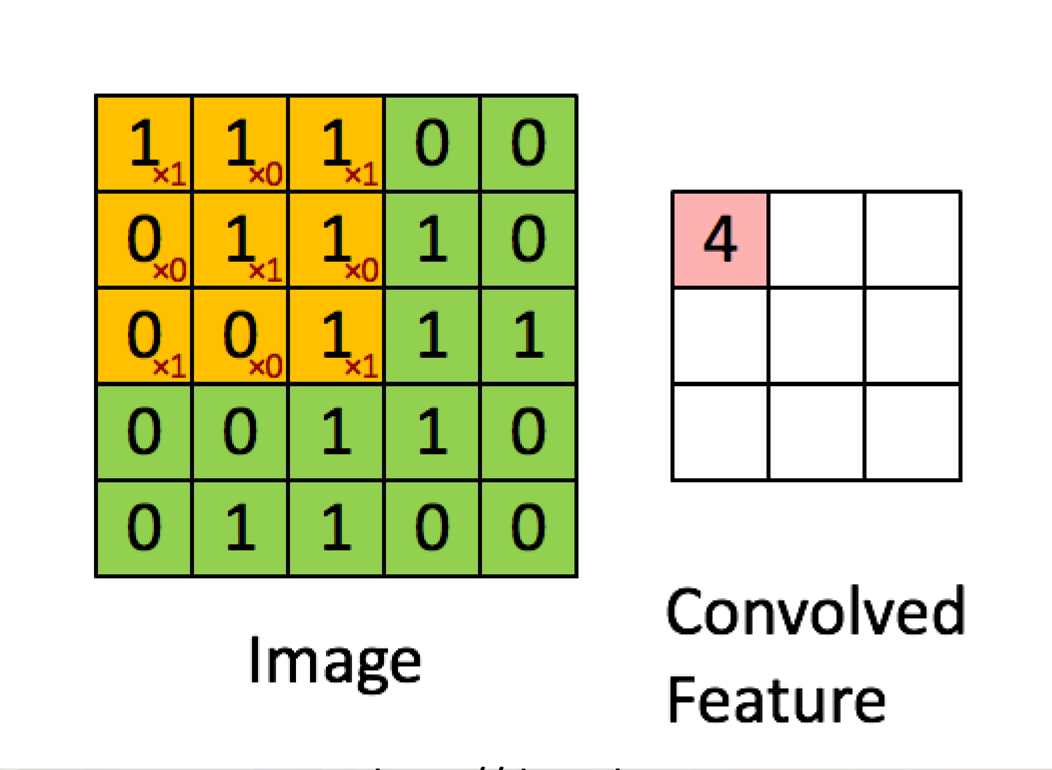
invariance

Convolutional neural networks

A convolution can be applied to reduce the dimensionality of the input (e.g. a high-res image), retaining the important information for later more effective processing

A number of "filters" can be used to reduce the input data





$1_{\times 1}$	$1_{\times 0}$	$1_{\times 1}$	0	0
$0_{\times 0}$	$1_{\times 1}$	$1_{\times 0}$	1	0
$0_{\times 1}$	$0_{\times 0}$	$1_{\times 1}$	1	1
0	0	1	1	0
0	1	1	0	0

4		

http://danielnouri.org

Example of 3x3 filter

original

filter (3 x 3)

0	0	0
0	1	0
0	0	0

identity

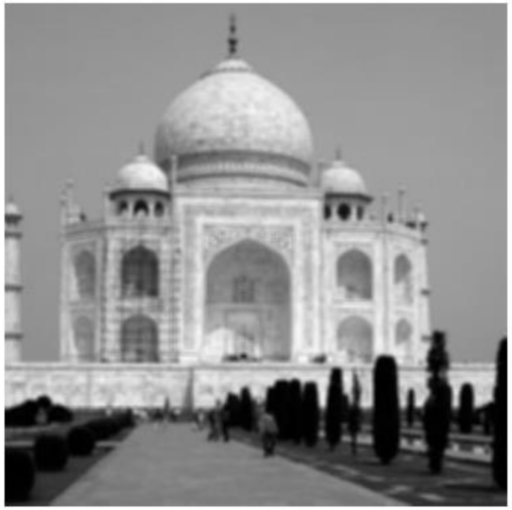
Example of a blurring 5x5 filter

original

filter (5 x 5)

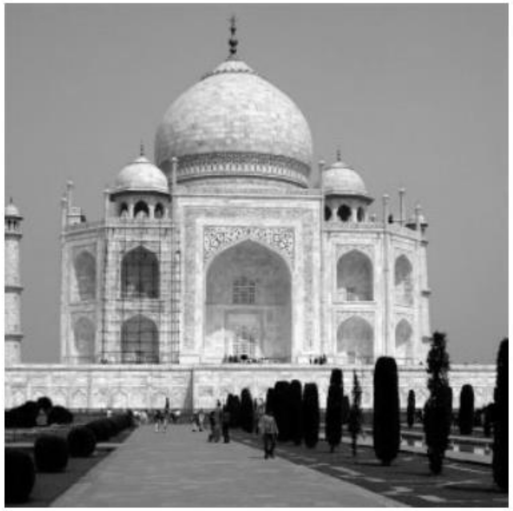
0	0	0	0	0
0	1	1	1	0
0	1	1	1	0
0	1	1	1	0
0	0	0	0	0

blur



A 5x5 sharpening filter

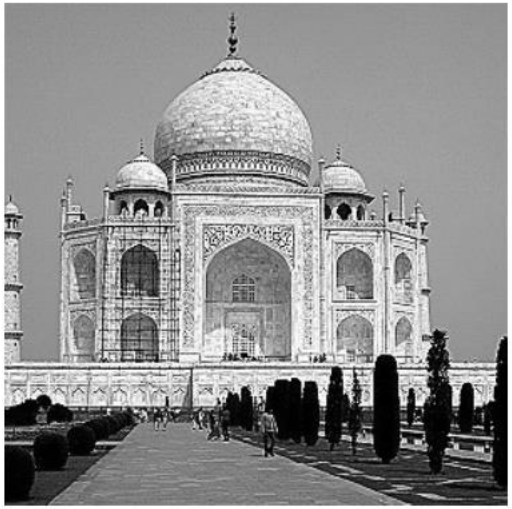
original



filter (5 x 5)

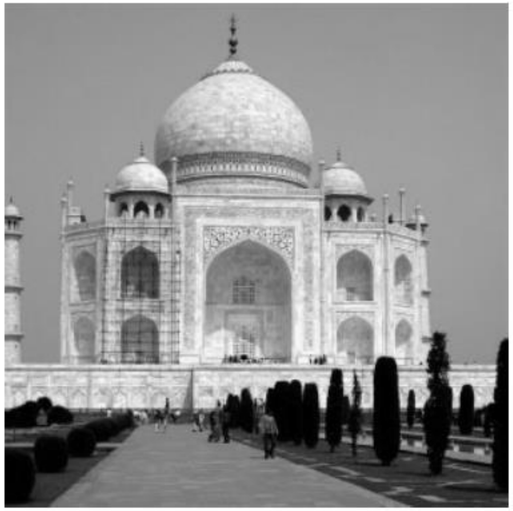
0	0	0	0	0
0	0	-1	0	0
0	-1	5	-1	0
0	0	-1	0	0
0	0	0	0	0

sharpen



Example of a 3x3 edge detector

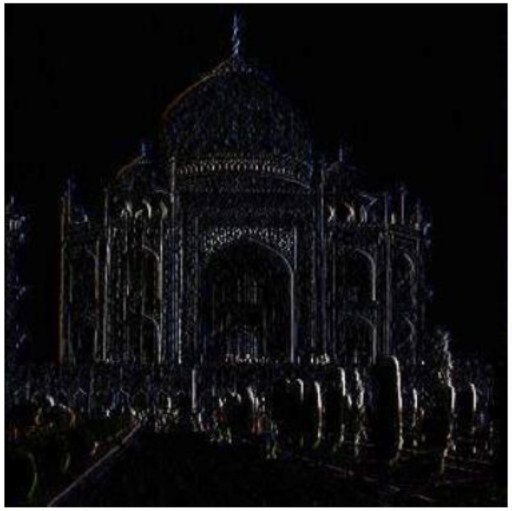
original



filter (3 x 3)

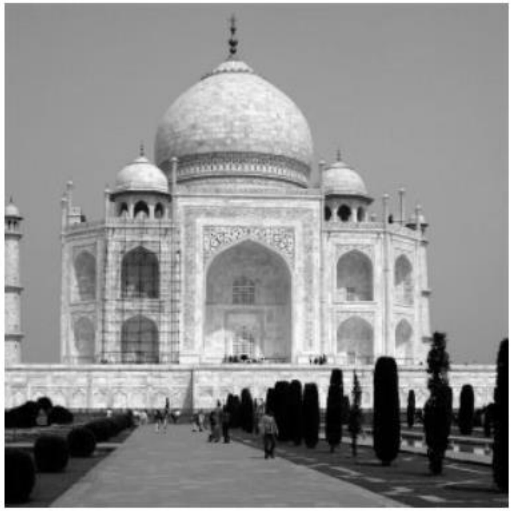
	0	0	0	
	-1	1	0	
	0	0	0	

vertical edge detector



A 3x3 all-edge detector

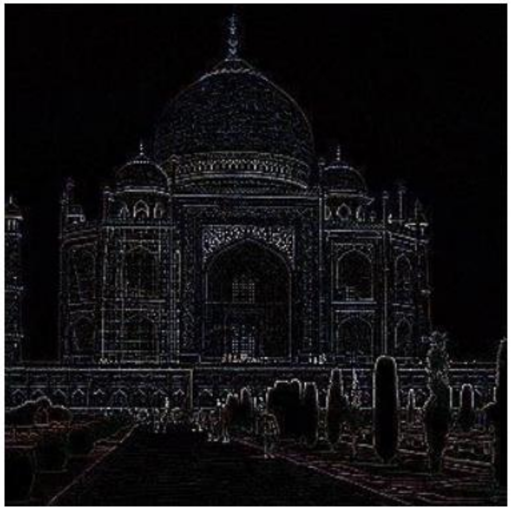
original



filter (3 x 3)

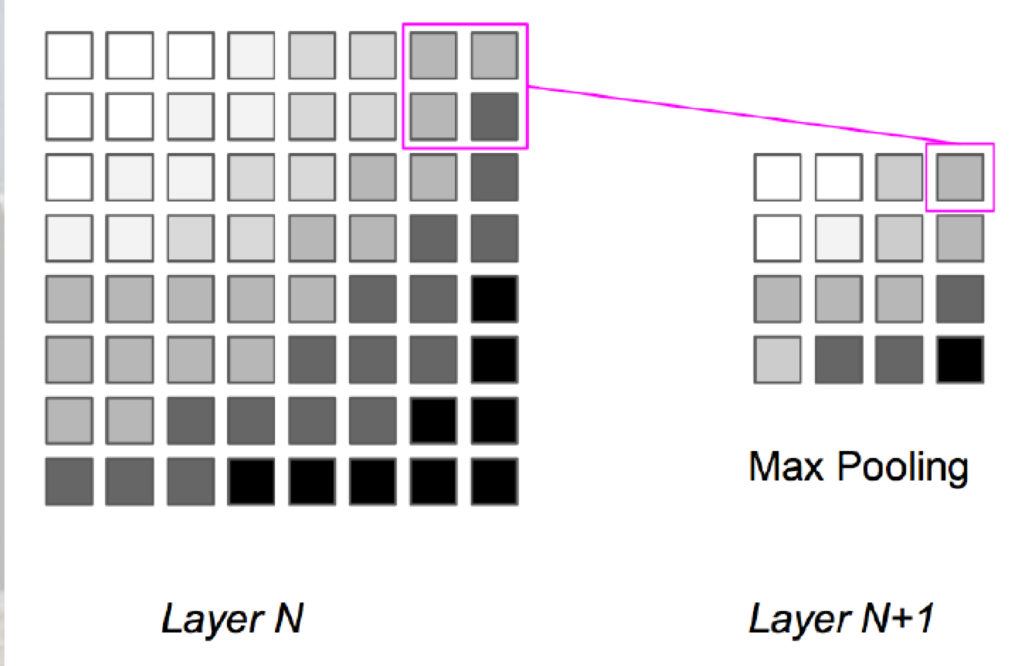
	0	1	0	
	1	-4	1	
	0	1	0	

all edge detector

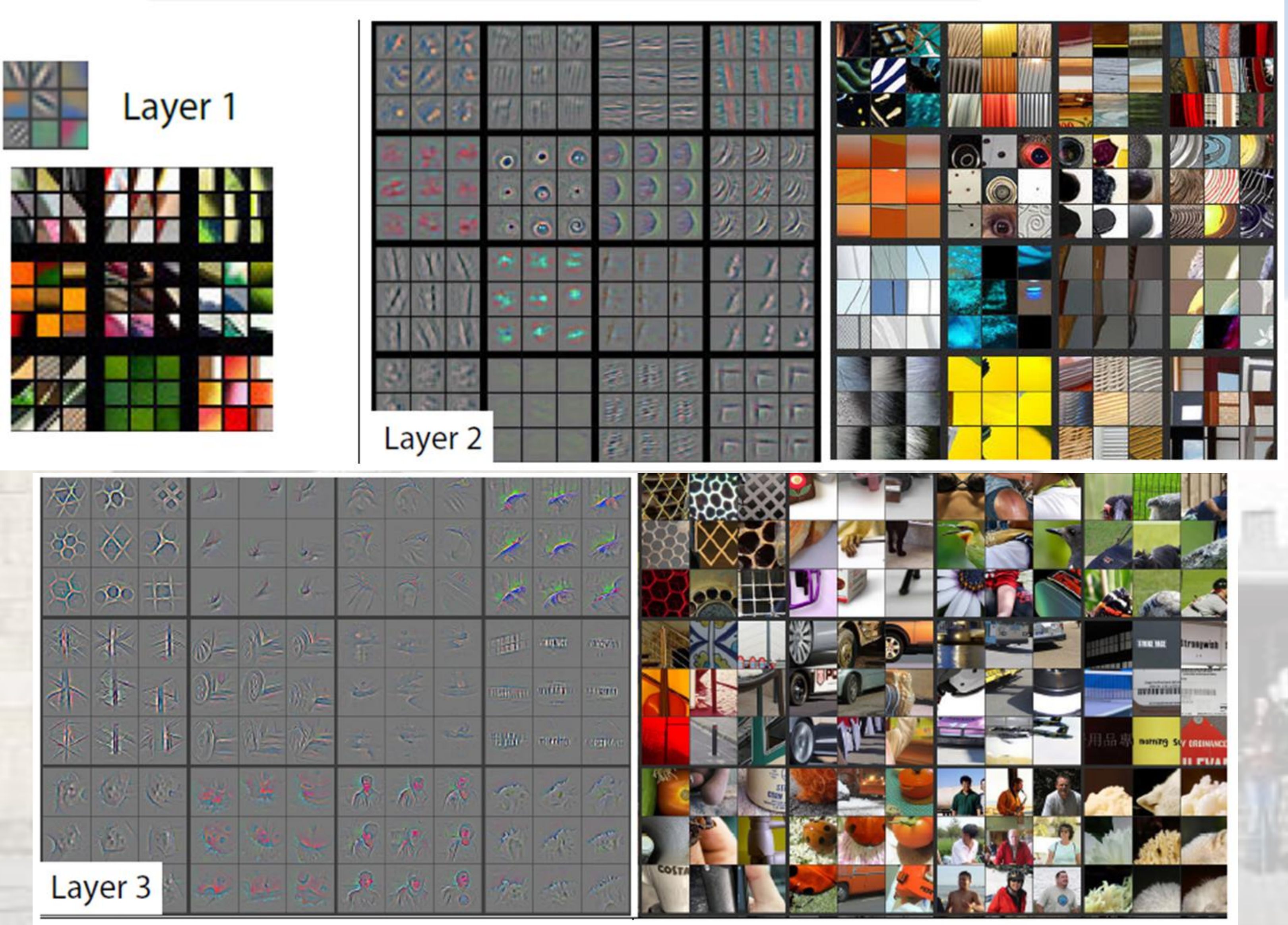


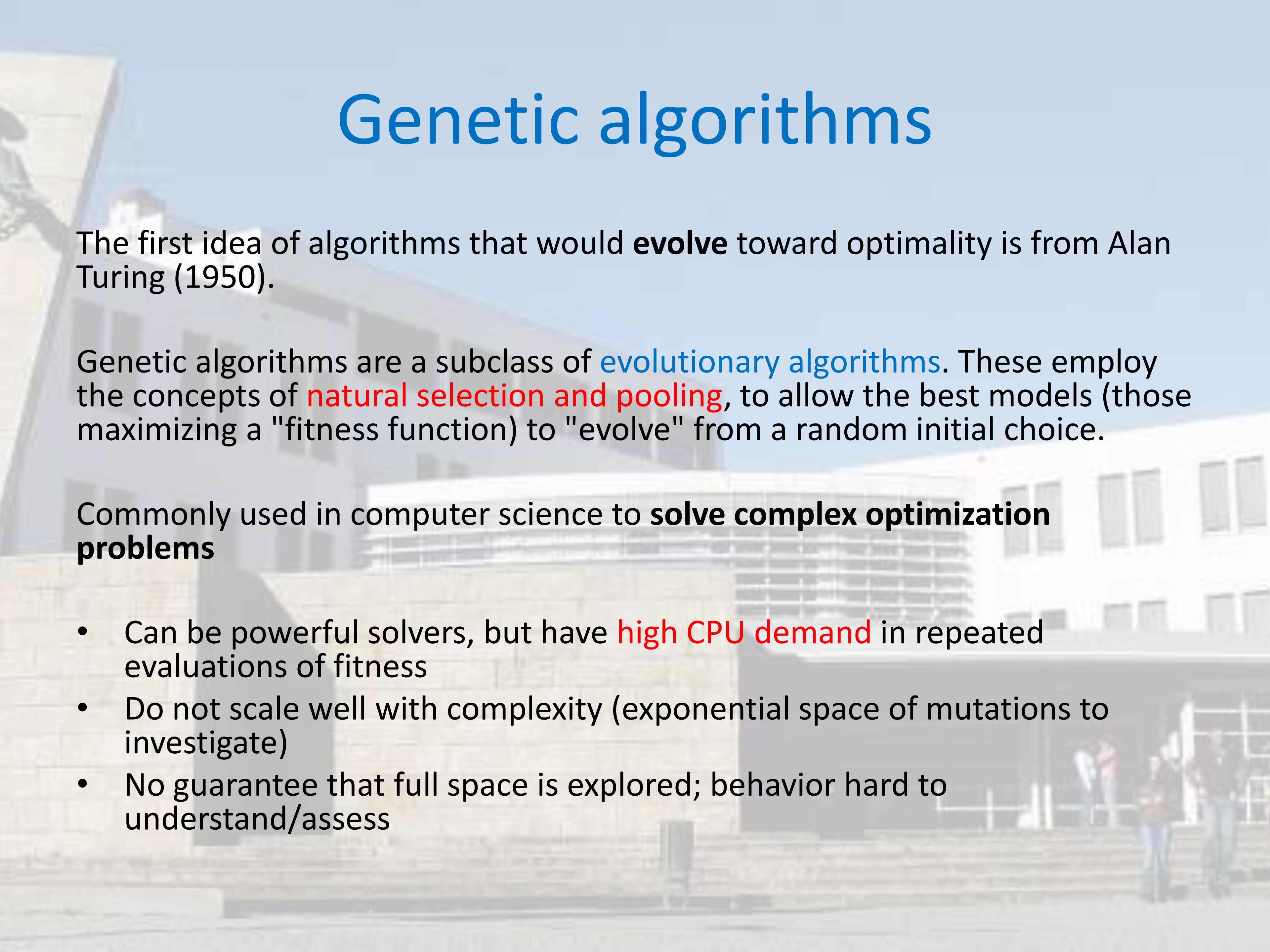
Max pooling

A different reduction of dimensionality is achieved by the method of "max pooling", which retains the most interesting information from the inputs, producing in the output a more compact map of the image



Feature detection





Genetic algorithms

The first idea of algorithms that would **evolve** toward optimality is from Alan Turing (1950).

Genetic algorithms are a subclass of evolutionary algorithms. These employ the concepts of natural selection and pooling, to allow the best models (those maximizing a "fitness function) to "evolve" from a random initial choice.

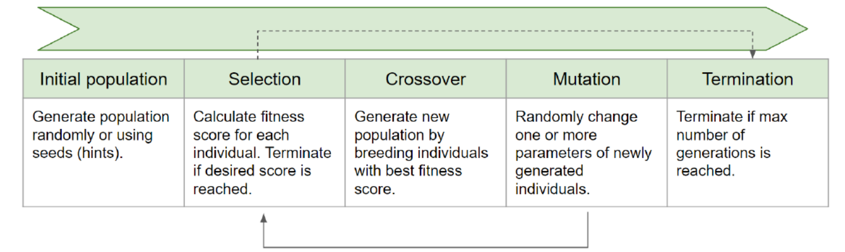
Commonly used in computer science to **solve complex optimization problems**

- Can be powerful solvers, but have high CPU demand in repeated evaluations of fitness
- Do not scale well with complexity (exponential space of mutations to investigate)
- No guarantee that full space is explored; behavior hard to understand/assess

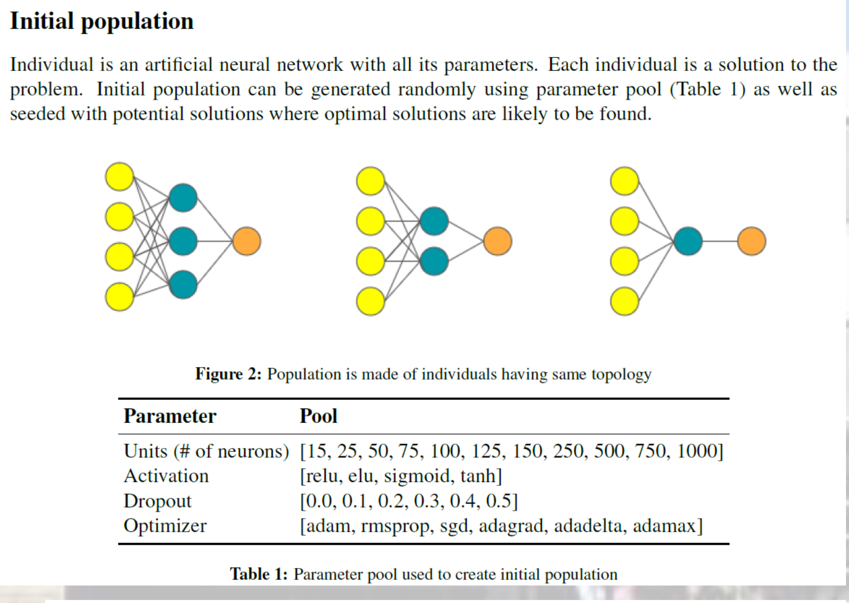
Example: Neuroevolution for data certification in CMS

Optimization of hyperparameters of neural networks trained to classify CMS data events

- study ROC AUC as figure of merit
- Start with random choice of hyperparameter, create pool of classifiers, train them
- let them breed (crossover), insert mutations
- Re-seed periodically picking best-performing 50%, other part fished randomly to avoid getting stuck in local minima, if evolution stops or in case of bad mutations
- Pool evolves to best choice of hyperparameters



Initial population	Selection	Crossover	Mutation	Termination
Generate population randomly or using seeds (hints).	Calculate fitness score for each individual. Terminate if desired score is reached.	Generate new population by breeding individuals with best fitness score.	Randomly change one or more parameters of newly generated individuals.	Terminate if max number of generations is reached.



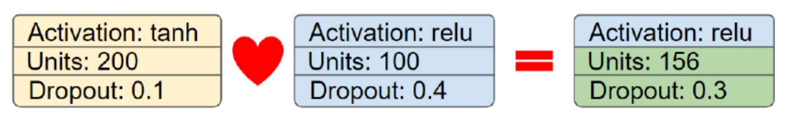
Initial population

Individual is an artificial neural network with all its parameters. Each individual is a solution to the problem. Initial population can be generated randomly using parameter pool (Table 1) as well as seeded with potential solutions where optimal solutions are likely to be found.

Figure 2: Population is made of individuals having same topology

Parameter	Pool
Units (# of neurons)	[15, 25, 50, 75, 100, 125, 150, 250, 500, 750, 1000]
Activation	[relu, elu, sigmoid, tanh]
Dropout	[0.0, 0.1, 0.2, 0.3, 0.4, 0.5]
Optimizer	[adam, rmsprop, sgd, adagrad, adadelta, adamax]

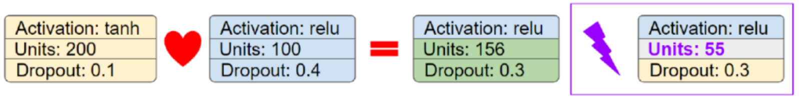
Table 1: Parameter pool used to create initial population



Activation: tanh
Units: 200
Dropout: 0.1

Activation: relu
Units: 100
Dropout: 0.4

Activation: relu
Units: 156
Dropout: 0.3



Activation: tanh
Units: 200
Dropout: 0.1

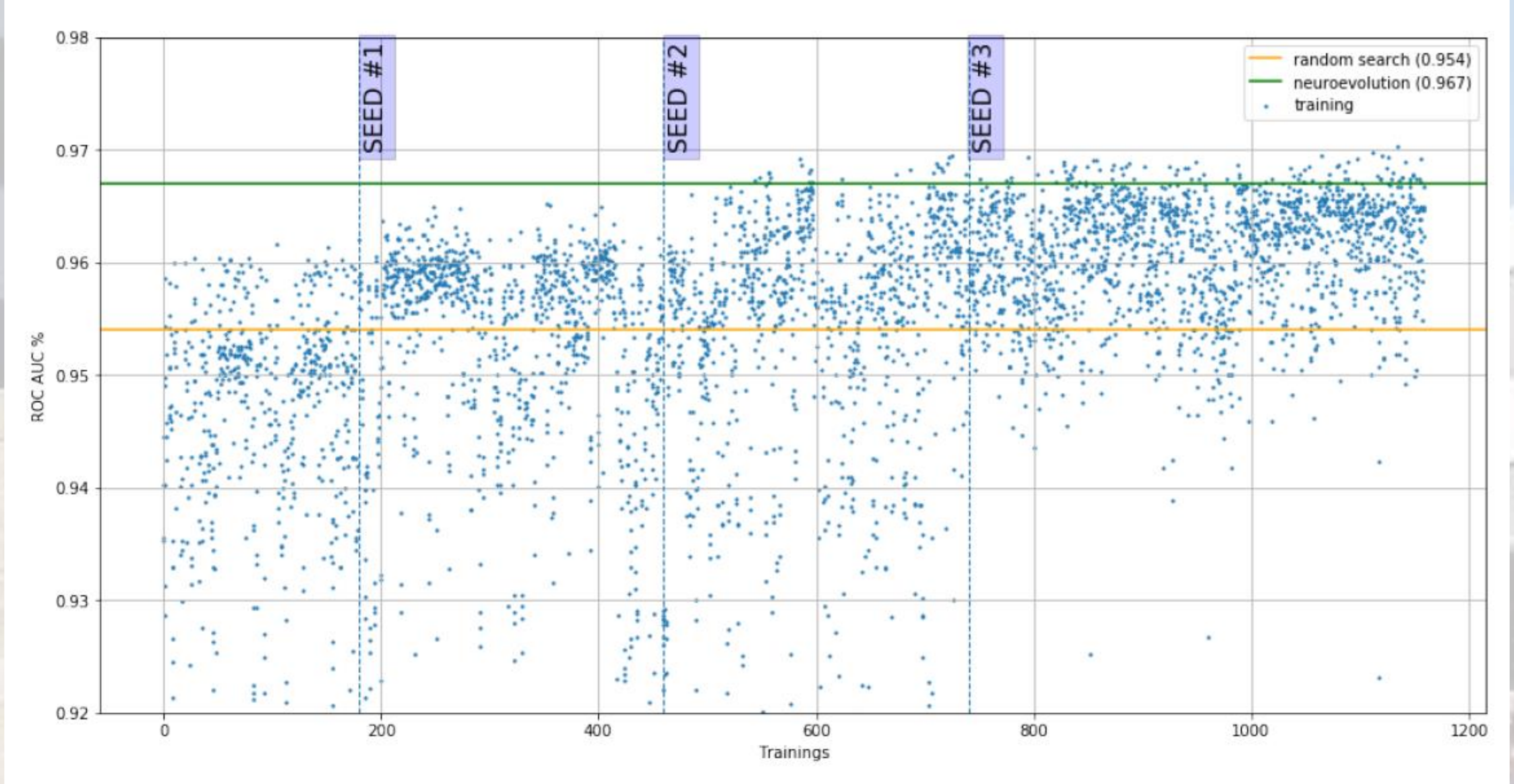
Activation: relu
Units: 100
Dropout: 0.4

Activation: relu
Units: 156
Dropout: 0.3

Activation: relu
Units: 55
Dropout: 0.3

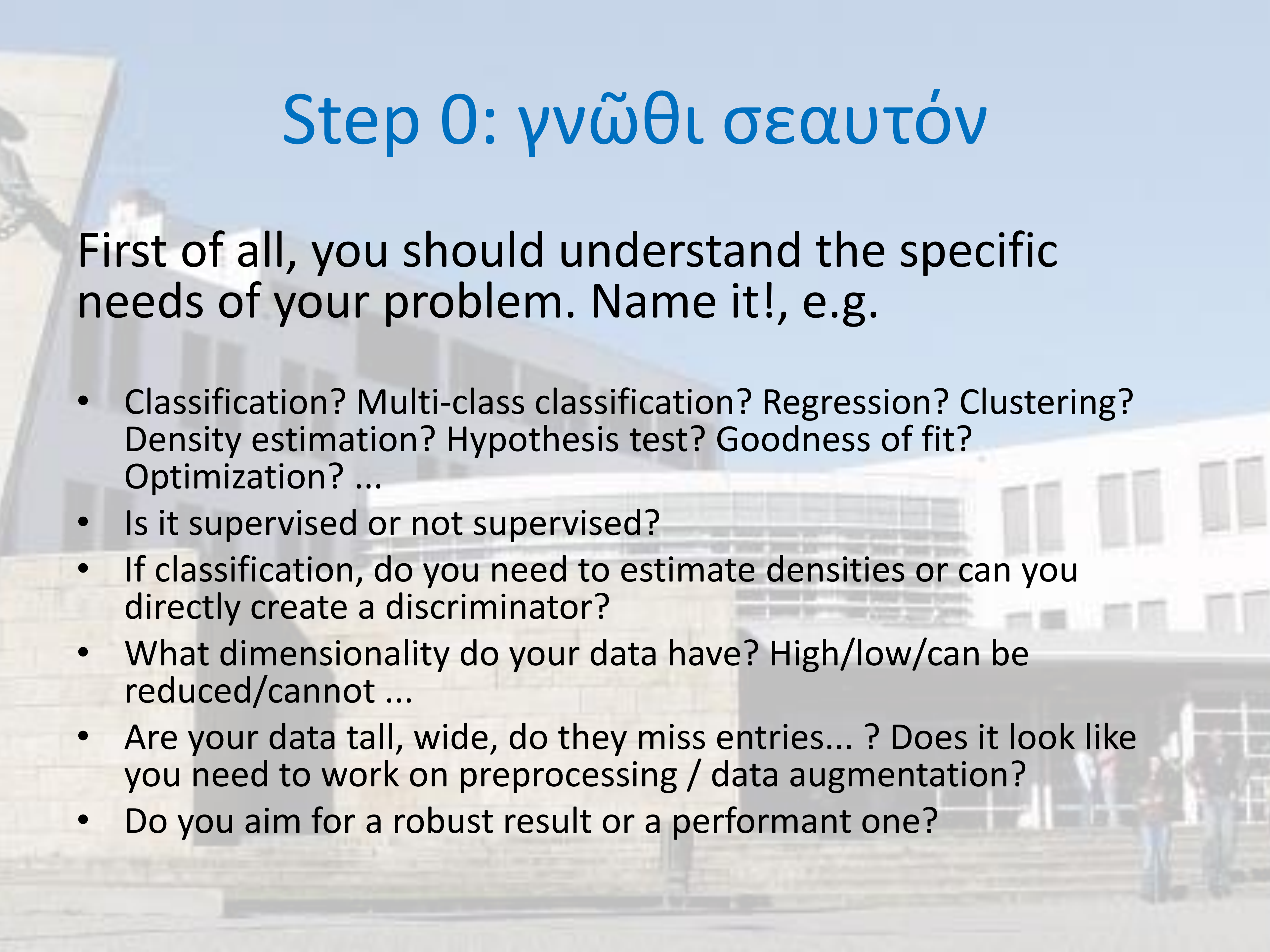
Result of genetic evolution

The networks "evolve" to higher AUC values, as designed.



Mantas Stankevicius, Valdas Rapsevicius, Virginijus Marcinkevicius
Vilnius University, Institute of Data Science and Digital Technologies
mantas.stankevicius@mif.vu.lt

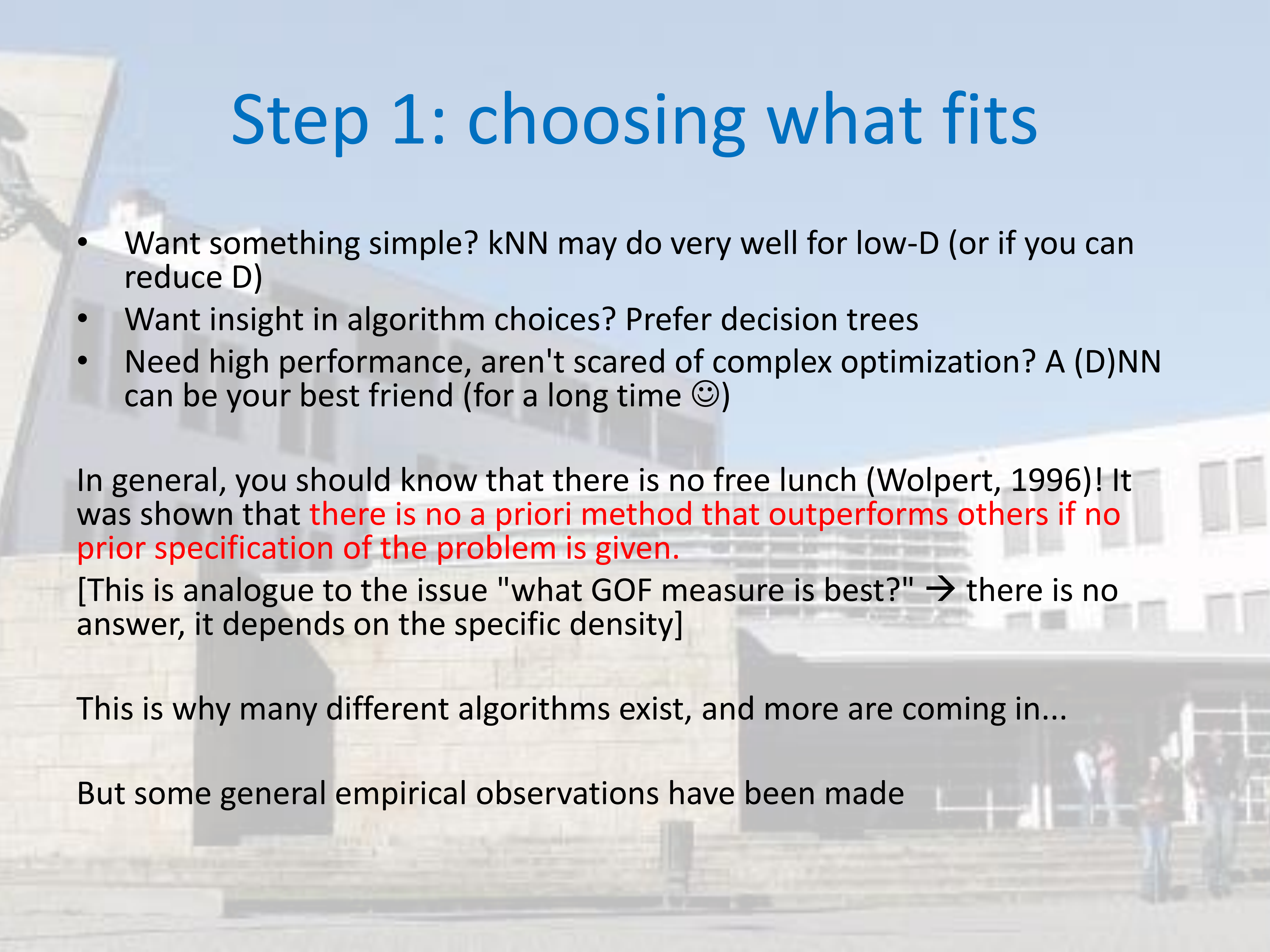
PRACTICAL TIPS



Step 0: γνῶθι σεαυτόν

First of all, you should understand the specific needs of your problem. Name it!, e.g.

- Classification? Multi-class classification? Regression? Clustering? Density estimation? Hypothesis test? Goodness of fit? Optimization? ...
- Is it supervised or not supervised?
- If classification, do you need to estimate densities or can you directly create a discriminator?
- What dimensionality do your data have? High/low/can be reduced/cannot ...
- Are your data tall, wide, do they miss entries... ? Does it look like you need to work on preprocessing / data augmentation?
- Do you aim for a robust result or a performant one?



Step 1: choosing what fits

- Want something simple? kNN may do very well for low-D (or if you can reduce D)
- Want insight in algorithm choices? Prefer decision trees
- Need high performance, aren't scared of complex optimization? A (D)NN can be your best friend (for a long time ☺)

In general, you should know that there is no free lunch (Wolpert, 1996)! It was shown that there is no a priori method that outperforms others if no prior specification of the problem is given.

[This is analogue to the issue "what GOF measure is best?" → there is no answer, it depends on the specific density]

This is why many different algorithms exist, and more are coming in...

But some general empirical observations have been made

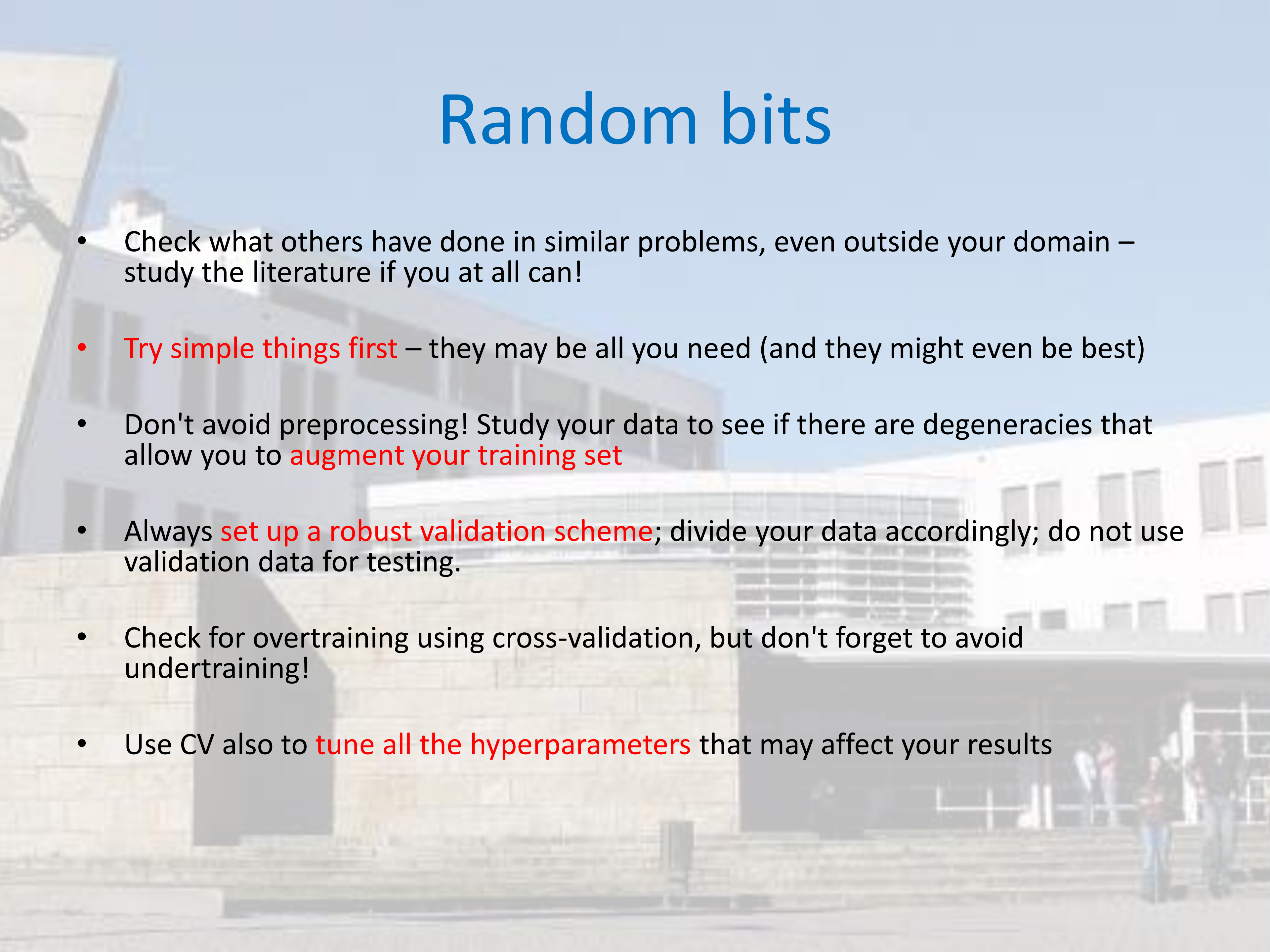
Empirical analysis

A survey of 179 methods (not including DNNs) was made testing them on 121 datasets

→ RF was the best performer in 84% of cases (see http://jmlr.csail.mit.edu/papers/volume15/delgado14a/delgado14a.pdf)

Kaggle competitions also allow to draw some conclusions (M.Kagan):

- When high-level features informative of the system are present, winners are often RF
- When you have lots of unstructured, low-level information per event, DNN outperform all others
- CNNs typically work best in image classification, RNN excel in text/speech recognition



Random bits

- Check what others have done in similar problems, even outside your domain – study the literature if you at all can!

- Try simple things first – they may be all you need (and they might even be best)

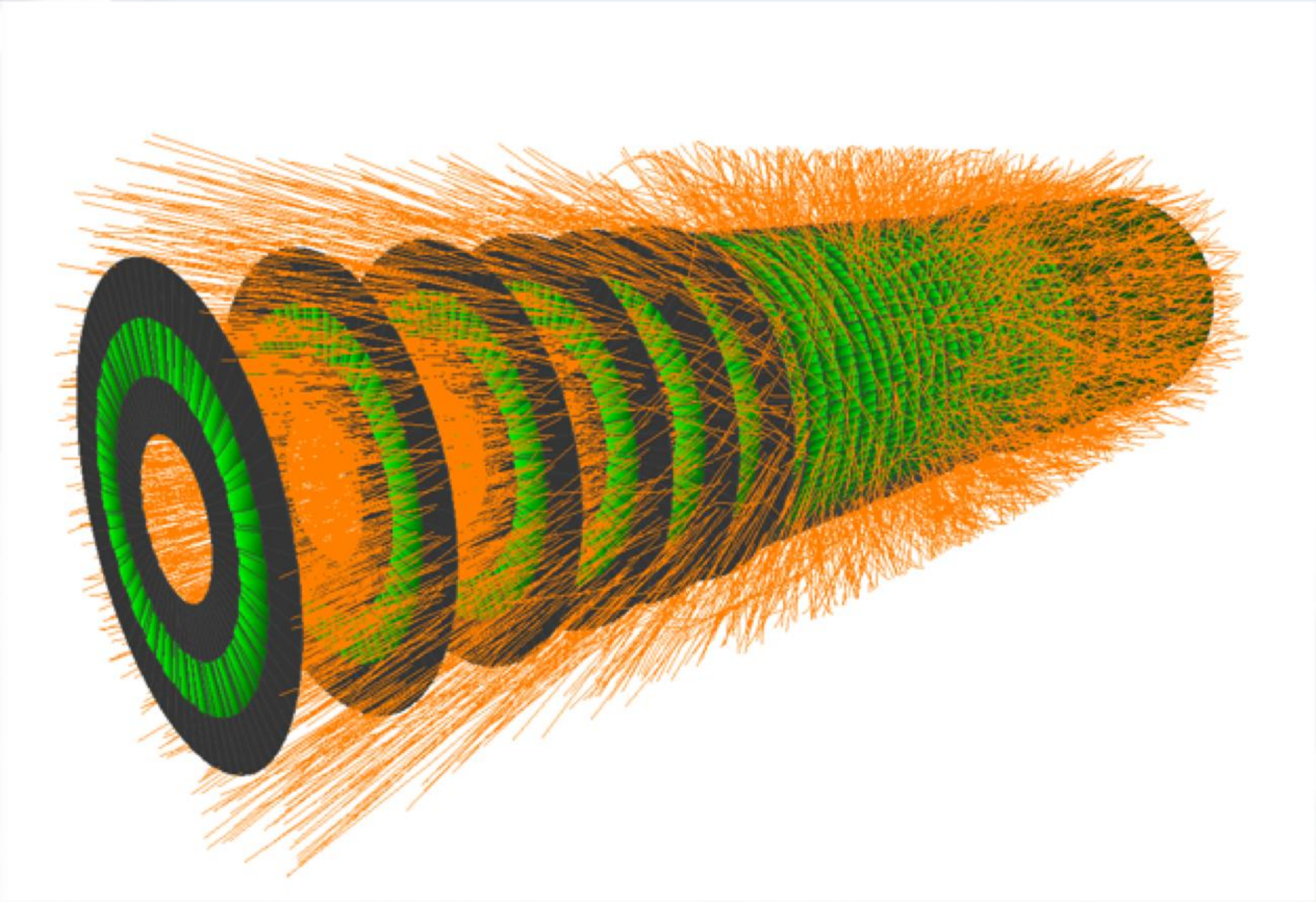
- Don't avoid preprocessing! Study your data to see if there are degeneracies that allow you to augment your training set

- Always set up a robust validation scheme; divide your data accordingly; do not use validation data for testing.

- Check for overtraining using cross-validation, but don't forget to avoid undertraining!

- Use CV also to tune all the hyperparameters that may affect your results

MACHINE LEARNING IN HEP



Statistical learning techniques for HEP: some examples

In HEP the commonest problem is extracting a small signal from large set of background-dominated data → **classification** (sometimes multi-class)

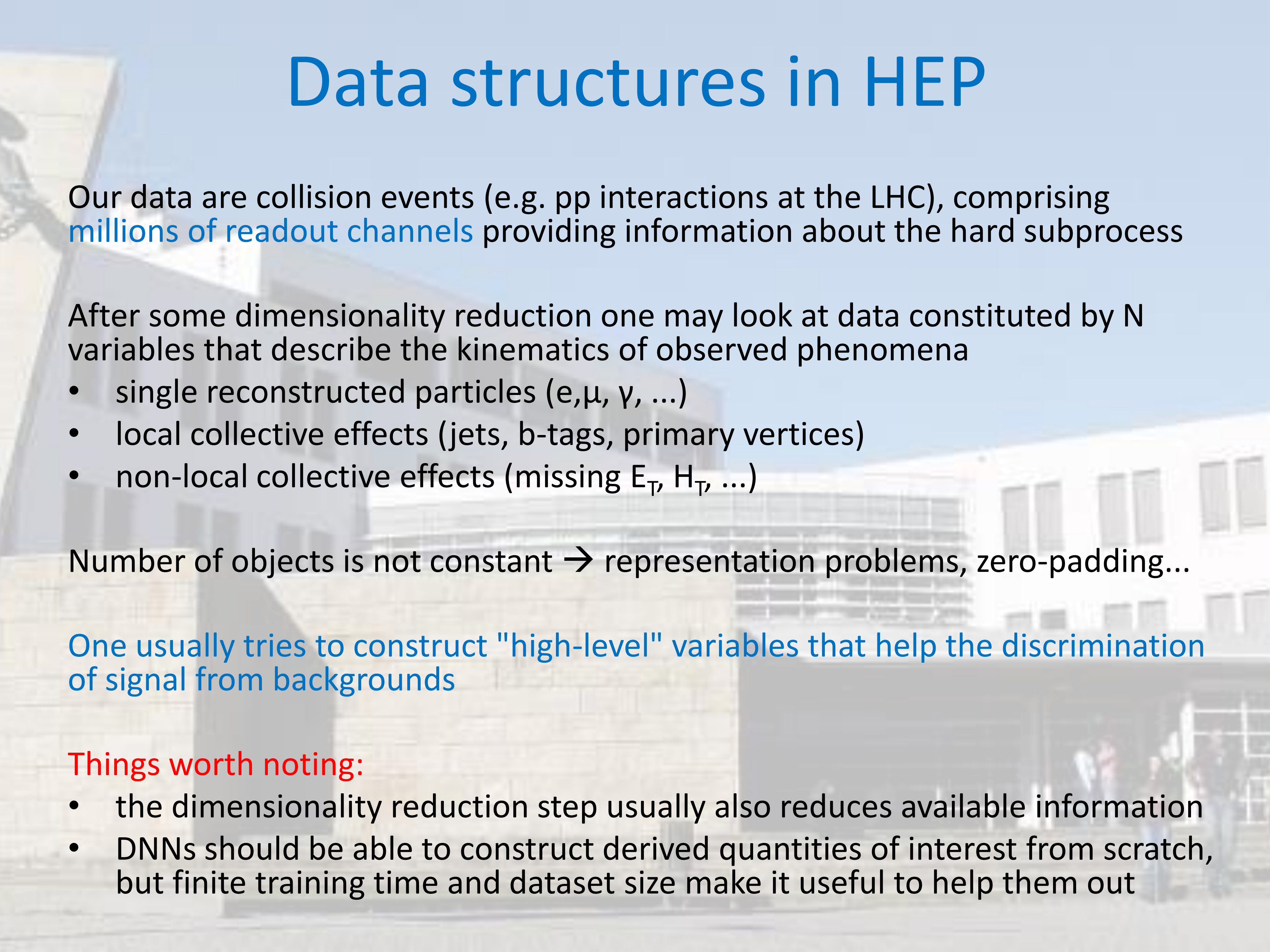
Regression also widely used, e.g. to improve the estimate of an important feature (sometimes as an input to classification)

Almost always we deal with supervised learning tasks, but anomaly detection (unsupervised or semi-supervised) is starting to get attention. Resistance due to insufficient trust in models out of the bulk of the phase space.

Clustering is used as a tool (e.g. jet reconstruction); at high level rarely. Will see an example later (Higgs pair benchmarks).

Recently interest has boomed in more cutting-edge techniques:

- **Generative Adversarial Networks** (e.g. to create better models)
- **Convolutional networks** (image processing for e.g. jet substructure)



Data structures in HEP

Our data are collision events (e.g. pp interactions at the LHC), comprising millions of readout channels providing information about the hard subprocess

After some dimensionality reduction one may look at data constituted by N variables that describe the kinematics of observed phenomena

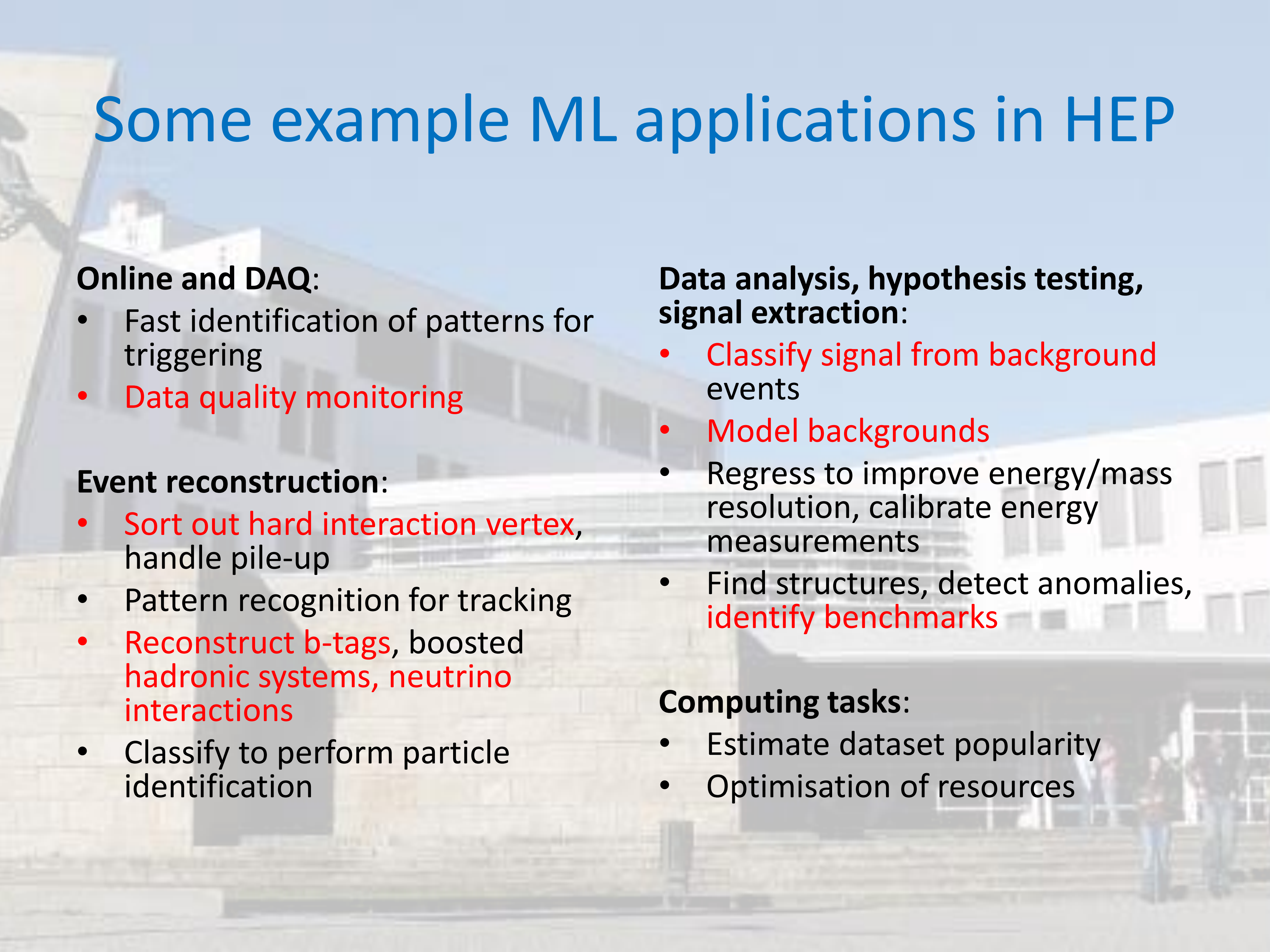
- single reconstructed particles (e,μ, γ, ...)
- local collective effects (jets, b-tags, primary vertices)
- non-local collective effects (missing E_T, H_T, ...)

Number of objects is not constant → representation problems, zero-padding...

One usually tries to construct "high-level" variables that help the discrimination of signal from backgrounds

Things worth noting:

- the dimensionality reduction step usually also reduces available information
- DNNs should be able to construct derived quantities of interest from scratch, but finite training time and dataset size make it useful to help them out



Some example ML applications in HEP

Online and DAQ:

- Fast identification of patterns for triggering
- Data quality monitoring

Event reconstruction:

- Sort out hard interaction vertex, handle pile-up
- Pattern recognition for tracking
- Reconstruct b-tags, boosted hadronic systems, neutrino interactions
- Classify to perform particle identification

Data analysis, hypothesis testing, signal extraction:

- Classify signal from background events
- Model backgrounds
- Regress to improve energy/mass resolution, calibrate energy measurements
- Find structures, detect anomalies, identify benchmarks

Computing tasks:

- Estimate dataset popularity
- Optimisation of resources

Two ubiquitous HEP problems

1) Improve the energy resolution of jet pairs, or photon pairs

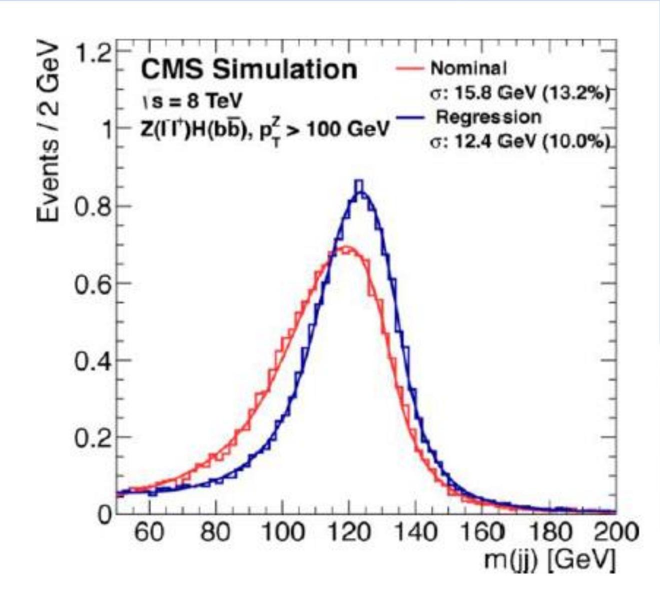
Pre-2012, a lot of emphasis was given to this problem in connection to H searches. Still at the focus for high-mass particles.

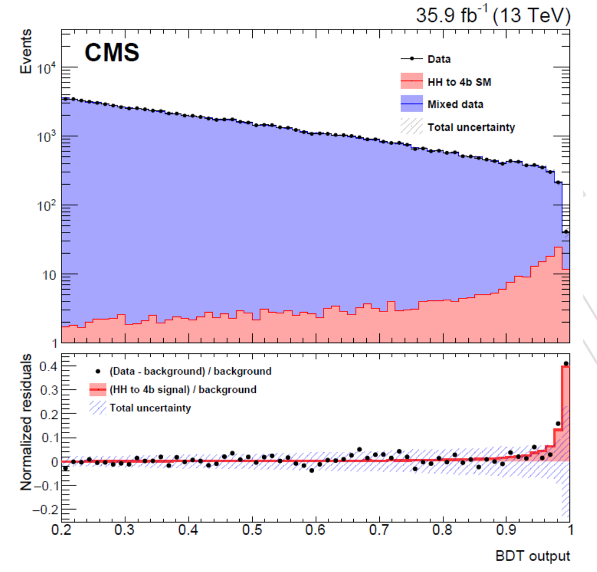
A narrower M_{jj} or $M_{\gamma\gamma}$ peak improves observability proportionally to $1/\sigma$

→ **regression**: achieve the best estimate of m_H given m_{12} and other observables

2) **Classify** to obtain an effective "summary statistic", discriminating signal from backgrounds

Often, rather than selecting data with high value of a discriminant (BDT, NN, ...) one fits the full distribution → tough modeling issues





For instance, in Higgs searches…

In the Nature review article "Machine learning at the energy and intensity frontiers of particle physics" by A.Radovic et al. was offered a summary of the effect of ML techniques employed by ATLAS and CMS on Higgs sensitivity

The improvement over non-ML techniques is quite significant

More "aggressive" methods could certainly achieve still more

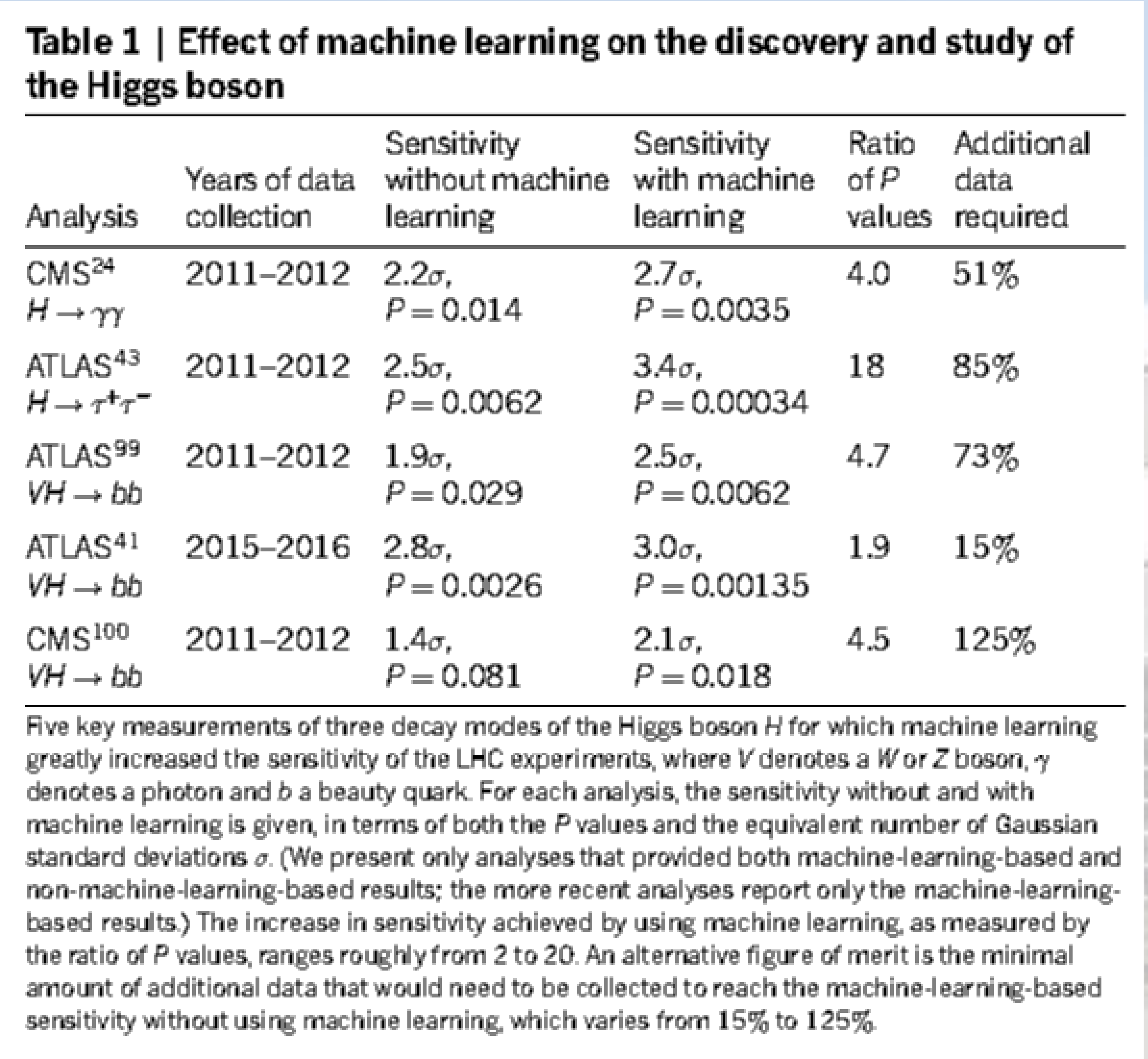


Table 1 | Effect of machine learning on the discovery and study of the Higgs boson

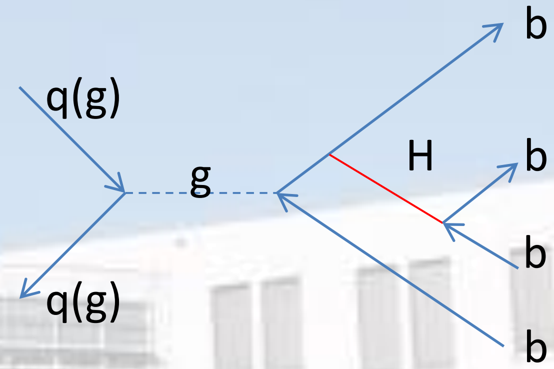
Analysis	Years of data collection	Sensitivity without machine learning	Sensitivity with machine learning	Ratio of *P* values	Additional data required
CMS[24] $H \rightarrow \gamma\gamma$	2011–2012	2.2σ, $P = 0.014$	2.7σ, $P = 0.0035$	4.0	51%
ATLAS[43] $H \rightarrow \tau^+\tau^-$	2011–2012	2.5σ, $P = 0.0062$	3.4σ, $P = 0.00034$	18	85%
ATLAS[99] $VH \rightarrow bb$	2011–2012	1.9σ, $P = 0.029$	2.5σ, $P = 0.0062$	4.7	73%
ATLAS[41] $VH \rightarrow bb$	2015–2016	2.8σ, $P = 0.0026$	3.0σ, $P = 0.00135$	1.9	15%
CMS[100] $VH \rightarrow bb$	2011–2012	1.4σ, $P = 0.081$	2.1σ, $P = 0.018$	4.5	125%

Five key measurements of three decay modes of the Higgs boson *H* for which machine learning greatly increased the sensitivity of the LHC experiments, where *V* denotes a *W* or *Z* boson, γ denotes a photon and *b* a beauty quark. For each analysis, the sensitivity without and with machine learning is given, in terms of both the *P* values and the equivalent number of Gaussian standard deviations σ. (We present only analyses that provided both machine-learning-based and non-machine-learning-based results; the more recent analyses report only the machine-learning-based results.) The increase in sensitivity achieved by using machine learning, as measured by the ratio of *P* values, ranges roughly from 2 to 20. An alternative figure of merit is the minimal amount of additional data that would need to be collected to reach the machine-learning-based sensitivity without using machine learning, which varies from 15% to 125%.

A kNN application: background modeling for SUSY Higgs search with CMS

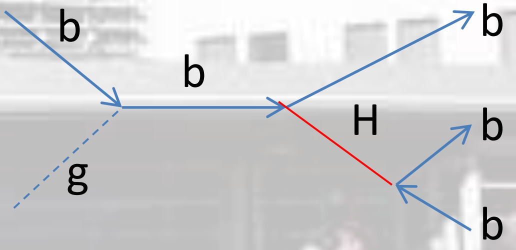
A search performed by the CMS-Padova group in 2011-2012 concerned the associated production of H and b-quarks

The final state includes >=3 b-jets



Supersymmetry foresees that these processes may have a large cross section; also, for much of the par space, two of the neutral Higgs bosons are mass-degenerate and contribute to the process

→ searching for the process contributes to exclude theory space, and probe a region of parameters where DZERO had seen a hint of a signal



The Problem

Multijet events are produced by QCD, and have a largely unknown, little-studied heavy flavour content

- Monte Carlo simulations can somehow model this, but large uncertainties remain from several assumed inputs and free parameters

- The mass spectrum of pairs of b-jets (anyway b-tagged) in a multijet sample is hard to model in shape, very hard to model in normalization

- Errors cannot be constrained to few-% level
- Compare to requirements for MSSM H→bb search: shape errors of 1-2% already significantly diminish the observability of the signal
  - → need a data-driven approach
  - → and it better be a precise one!

Note: a background shape with a normalization constraint is a good thing to have for a likelihood fit that searches for a signal in the spectrum, as the amount of observed data is good information along with the shape of the mass distribution

Data-driven approach

In principle, one can try and determine the double b-tagging probability from the data, in a suitable control sample, and apply it to a signal-enriched sample

- Example: in ==2-jet events (lower signal content) one calculates the fraction of (++) events as a function of critical kinematic variables (e.g. jet E_T's, jet # of tracks, p_T of dijet system...) and derives a $P^{++}(E_T^1, N^1, E_T^2, N^2, p_T^{jj})$ matrix

- Then one can apply it to 3-jet events (higher signal content) extracting a prediction binned in M_{jj} ("++" are double tags in lead jets, "00" untagged jets):

$$N_3^{++}(M_{jj}) = \sum_1^{N_3^{00}(M_{jj},X)} \frac{N_2^{++}(X)}{N_2^{00}(X)}$$

N_2^{00}	N_3^{00}
N_2^{++}	N_3^{++}

- This method can model biases precisely only by binning in many variables X (algorithmic dependence: N_{tracks}, η; physics dependence: P_T^{jj}, E_T, $\Delta\Phi^{jj}$, H_T...)
  → Statistically limited!

The Hyperball algorithm

Originally developed in 2003 to increase resolution of dijet mass shape in low-mass SM Higgs searches at the Tevatron, but in practice, not very different conceptually from standard kNN algorithm

- You have a M_{jj} estimate for two jets coming from the H→bb decay. Jets are mismeasured due to many detector and physics effects; you want to correct these effects.

IDEA: you measure along with jet 4-momenta MANY other kinematical characteristics of the jet and of the event that contains them

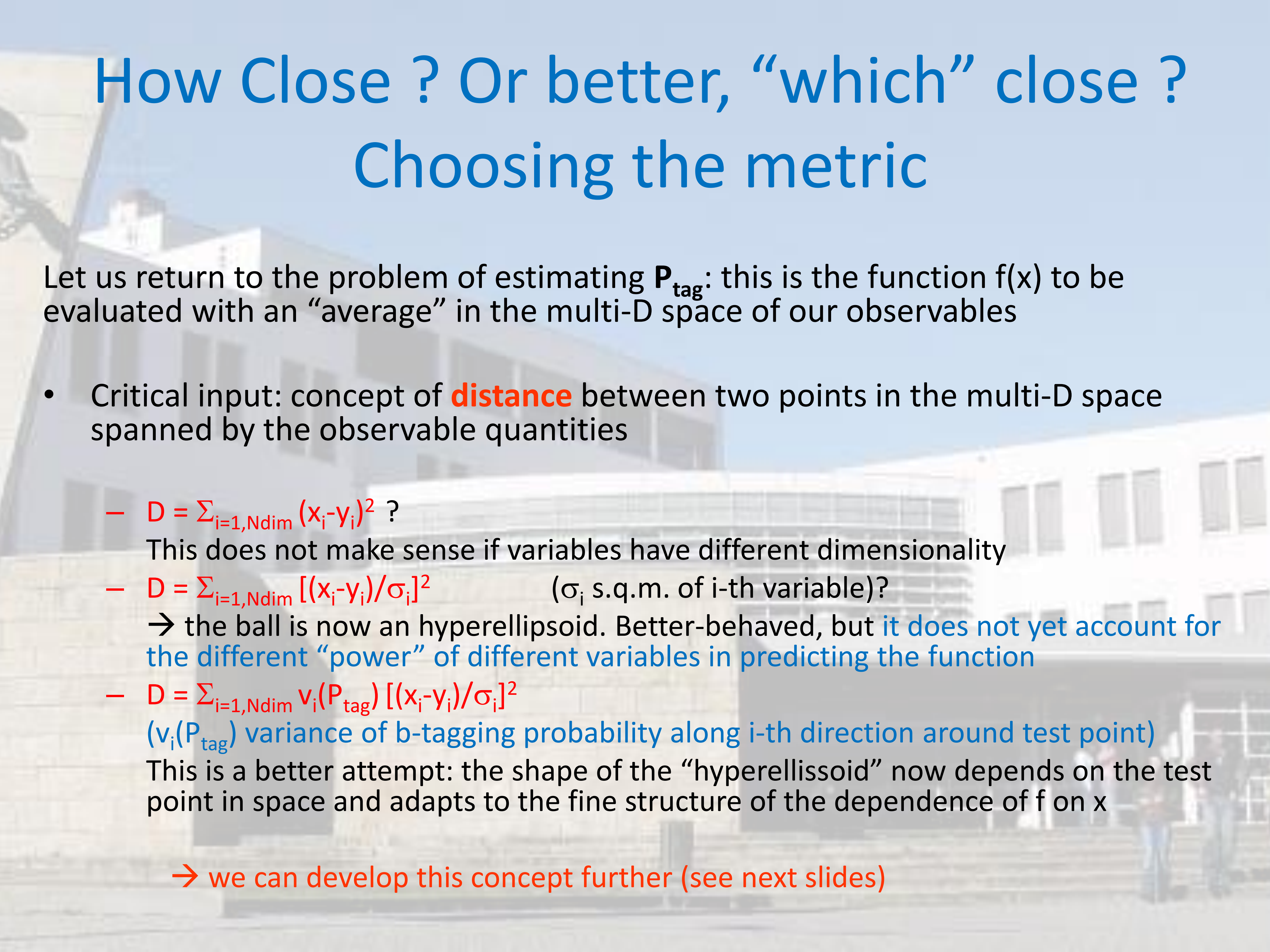
- – Jets are “undermeasured” or “overmeasured” and this reflects in the value of other variables → **Use these variables to guesstimate a correction to M_{jj}**

The approach is multi-dimensional. In general, if a quantity f(x) varies in the space of the observables x, you can determine its value in every point of the space by a nearest-neighbor approach.

- To compute a meaningful average, you need a sizable number of training events. But which ones do you pick ?

→**The CLOSEST ones in the multi-D space**.

Application already described in Lecture 1...



How Close ? Or better, “which” close ? Choosing the metric

Let us return to the problem of estimating $\mathbf{P_{tag}}$: this is the function f(x) to be evaluated with an “average” in the multi-D space of our observables

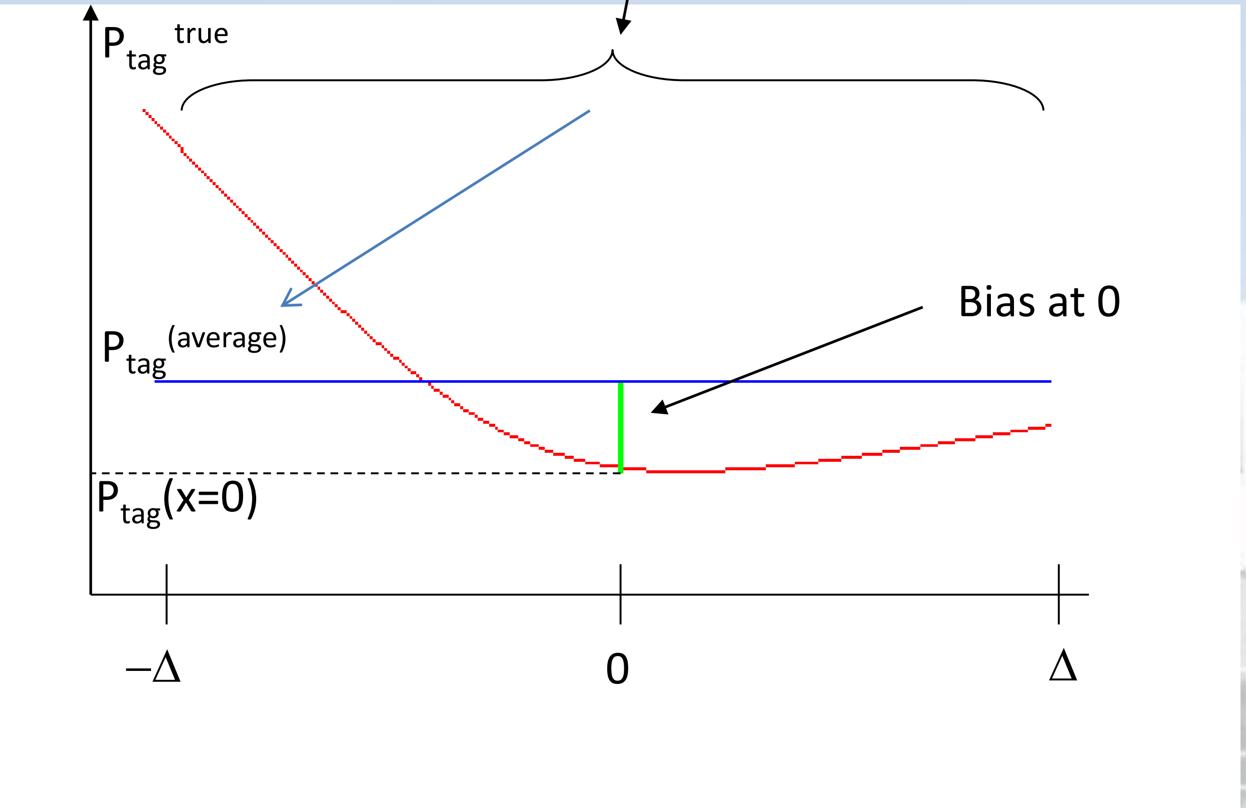
- Critical input: concept of **distance** between two points in the multi-D space spanned by the observable quantities
  - $D = \Sigma_{i=1,Ndim} (x_i-y_i)^2$?
    This does not make sense if variables have different dimensionality
  - $D = \Sigma_{i=1,Ndim} [(x_i-y_i)/\sigma_i]^2$ (σ_i s.q.m. of i-th variable)?
    → the ball is now an hyperellipsoid. Better-behaved, but it does not yet account for the different “power” of different variables in predicting the function
  - $D = \Sigma_{i=1,Ndim} v_i(P_{tag}) [(x_i-y_i)/\sigma_i]^2$
    ($v_i(P_{tag})$ variance of b-tagging probability along i-th direction around test point)
    This is a better attempt: the shape of the “hyperellissoid” now depends on the test point in space and adapts to the fine structure of the dependence of f on x

→ we can develop this concept further (see next slides)

To find the best definition of a distance measure, we can construct an approximate model of P_{tag} in the whereabouts of the test point, to derive bias in P_{tag} as $f(D_i)$ (i labels directions in space)

One needs to determine how quickly P_{tag} varies along each space coordinate, in the surroundings of each test point.
<u>The bigger the variation, the shortest we want the interval to be along that direction, to minimize the bias in the P_{tag} estimate</u>

Averaging the tag rate in the interval [−Δ:Δ]



In other words: we evaluate P_{tag} averaging within [-Δ,Δ] (blue line). However, along the coordinate we are looking at (x_i) the P_{tag} may vary non-linearly (red curve), creating a bias at 0.

We estimate that bias with an approximation of the P_{tag} curve as a quadratic function (see next slide for the math)

Some (easy!) math

Problem: We cannot compute f(x)=P_{tag} along the coordinate x, because it is an average! We know it only as a ratio of sparse points (tagged jets divided by all jets)

- What we can do is compute "moments", i.e. integrals: $\lambda_i = \int_{-\Delta}^{\Delta} x^i f(x) dx$

We write f(x) as a power series: $f(x) = \sum_{i=0} \alpha_i x^i$

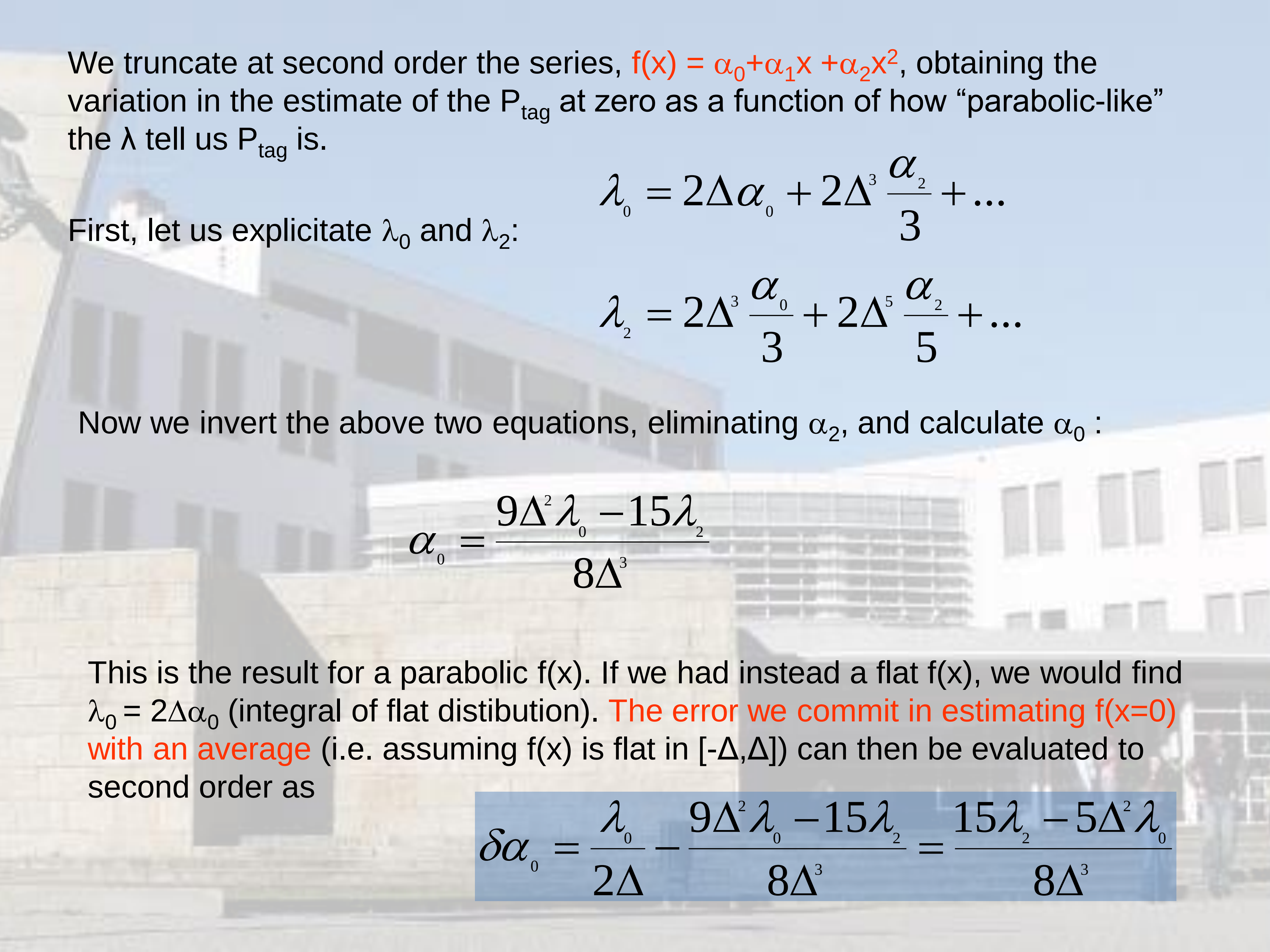
Then we can integrate and write the λ factors as a function of the α :

$$\lambda_N = \sum_i \frac{\alpha_i}{i+N+1}[\Delta^{i+N+1} - (-\Delta)^{i+N+1}]$$

- Remember that we only need to estimate the "bias" α_0 , **i.e. f(x=0)** and how it varies if f(x) is non linear

  → So we evaluate only λ_0 and λ_2 (λ_1 does not depend on α_0 since the expression in parentheses is null for i=0 and odd N) from the number and coordinates of tagged and untagged jets in the interval.

[f(x) = 1 for tagged jets, f(x)=0 for untagged ones, and moments will average these out]



We truncate at second order the series, $f(x) = \alpha_0+\alpha_1 x +\alpha_2 x^2$, obtaining the variation in the estimate of the P_{tag} at zero as a function of how “parabolic-like” the λ tell us P_{tag} is.

First, let us explicitate λ_0 and λ_2:

$$\lambda_0 = 2\Delta\alpha_0 + 2\Delta^3\frac{\alpha_2}{3} + \dots$$

$$\lambda_2 = 2\Delta^3\frac{\alpha_0}{3} + 2\Delta^5\frac{\alpha_2}{5} + \dots$$

Now we invert the above two equations, eliminating α_2, and calculate α_0 :

$$\alpha_0 = \frac{9\Delta^2\lambda_0 - 15\lambda_2}{8\Delta^3}$$

This is the result for a parabolic f(x). If we had instead a flat f(x), we would find $\lambda_0 = 2\Delta\alpha_0$ (integral of flat distibution). The error we commit in estimating f(x=0) with an average (i.e. assuming f(x) is flat in [-Δ,Δ]) can then be evaluated to second order as

$$\delta\alpha_0 = \frac{\lambda_0}{2\Delta} - \frac{9\Delta^2\lambda_0 - 15\lambda_2}{8\Delta^3} = \frac{15\lambda_2 - 5\Delta^2\lambda_0}{8\Delta^3}$$

Recipe summary

The recipe allows to estimate with no bias (to $O(x^2)$) the error $\delta\alpha_0$ we make on the estimate of P_{tag} due to the extension of the integration region, and due to the non-linearity of P_{tag} in it.
The bit of math is needed to deal with discontinuous inputs on f(x), which are either 0 or 1...

In order to minimize the global uncertainty of P_{tag} in a hyperball, weights can now be chosen such that the sums of squares of the $\delta\alpha_0$ factors in each test point are minimized

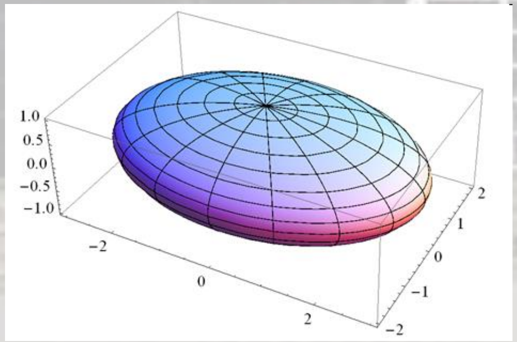
In practice the recipe is the following:

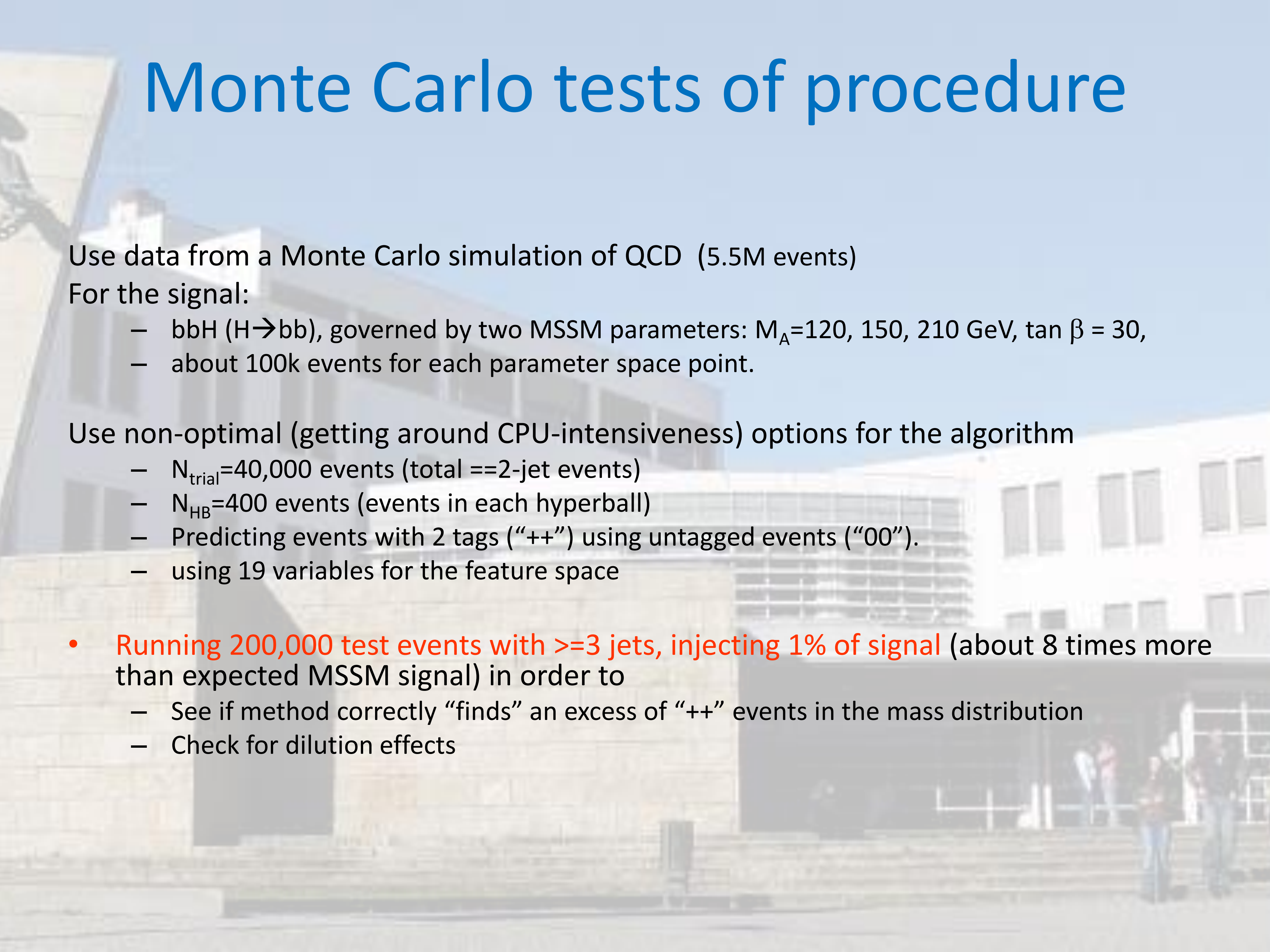
1) For each test point, inflate a ball (with axes determined by all-space variation of tag probability along each direction)

2) use captured training data to compute bias $\delta\alpha_0^{(i)}$ along each space direction

3) reconfigure ball shape such that each axis is inversely proportional to square of bias along that direction

4) compute P_{tag} with events captured in new ball





Monte Carlo tests of procedure

Use data from a Monte Carlo simulation of QCD (5.5M events)

For the signal:

- bbH (H→bb), governed by two MSSM parameters: M_A=120, 150, 210 GeV, tan β = 30,
- about 100k events for each parameter space point.

Use non-optimal (getting around CPU-intensiveness) options for the algorithm

- N_{trial}=40,000 events (total ==2-jet events)
- N_{HB}=400 events (events in each hyperball)
- Predicting events with 2 tags ("++") using untagged events ("00").
- using 19 variables for the feature space

- Running 200,000 test events with >=3 jets, injecting 1% of signal (about 8 times more than expected MSSM signal) in order to
  - See if method correctly "finds" an excess of "++" events in the mass distribution
  - Check for dilution effects

Goal: see whether dijet mass shape can be modeled in QCD MC

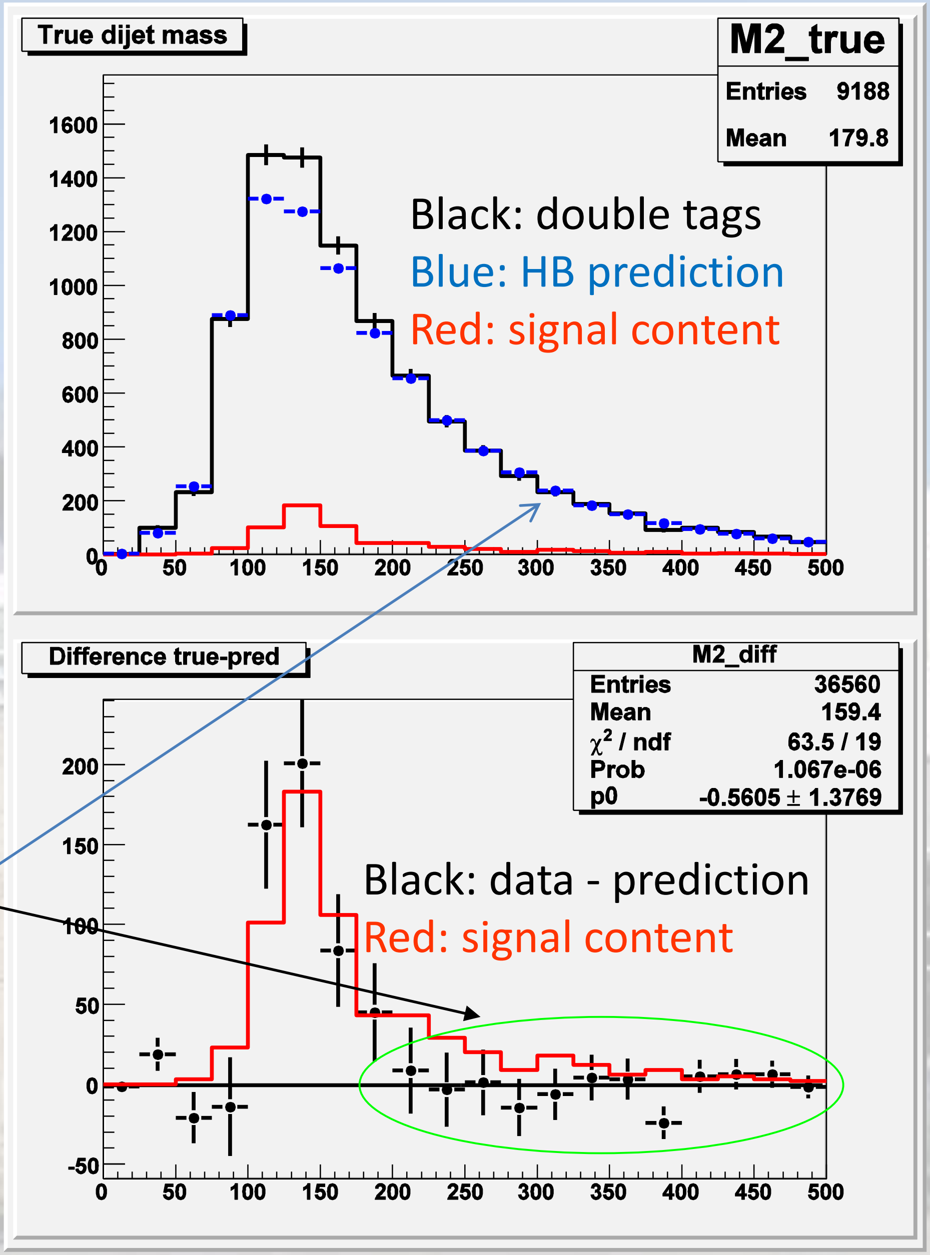
- Test with 200k QCD events, 2000 signal events (**bbH, m$_H$=150 GeV**)
- 19 variables in space; Mass observable: dijet mass of leading jet pair

Modeling events with ==2 b-tags in leading jet pair

- Observe 8977 (++) events
- Expect (HB) 8519.3±28.6

- Excess from HB: 457.7
- Signal in sample: 618 evts

- Excess/signal fraction: 0.74

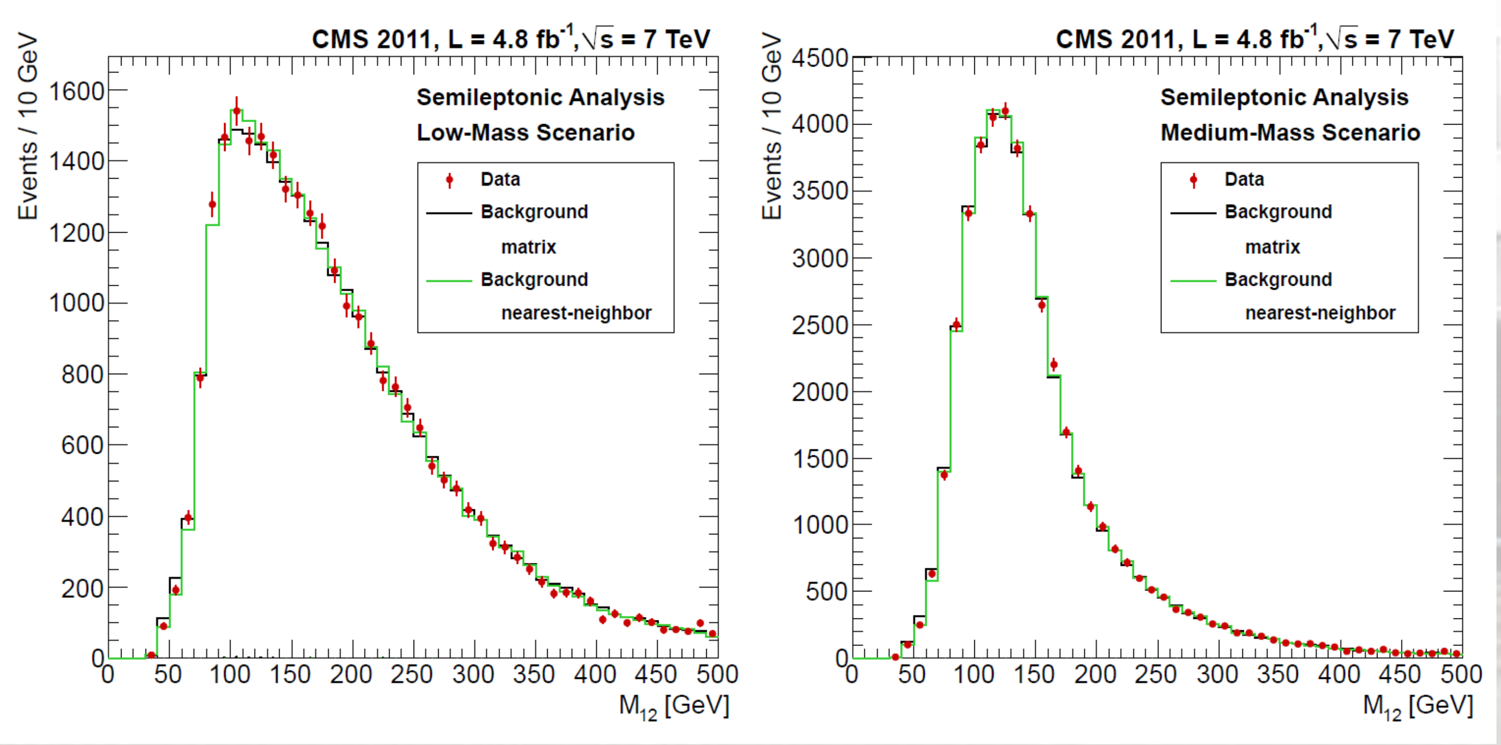


Agreement at the few percent level

Results on data

The technique, applied to control sample of 1-b-tag events, correctly predicts the background shape of 3-b-tag events in two disjunct datasets (low-mass and high-mass search regions)

Result is compatible with one provided by matrix method (working on 2-b-tag events, "closer" to signal region), but more precise

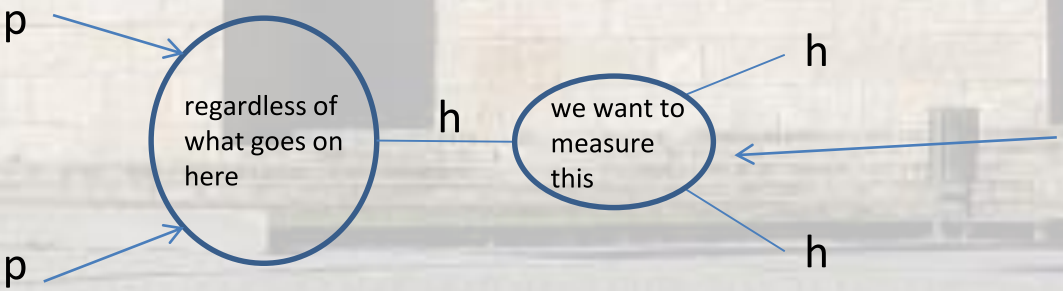


Unsupervised learning example: Cluster analysis of HH production

After the 2012 discovery, one questioned if its characteristics were those predicted by the SM

In particular its couplings to massive bosons and fermions are predicted precisely → production and decay measurement can test this

But H can couple to itself, too → how strongly ? The SM predicts a value, but new physics might change that value. To test it one should study processes sensitive to the self coupling: HH pair production



Self-coupling allows HH production through this diagram

BSM HH production

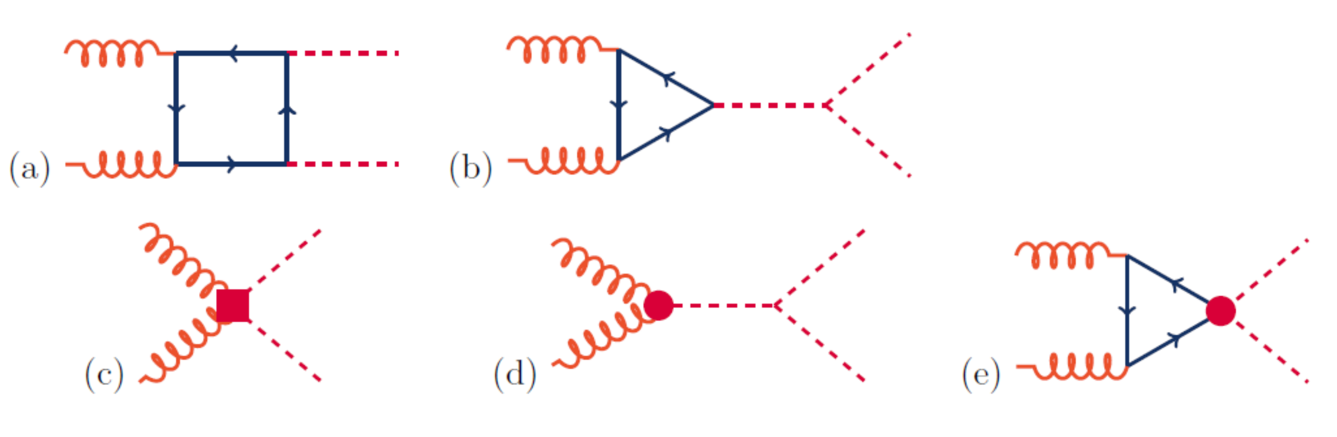
If the Higgs Lagrangian contains differences with SM predictions, these can be explicitated by anomalous terms in an "effective Lagrangian" that considers terms up to dimension-6

In general one may think at 5 classes of diagrams contributing to HH pairs at the LHC. For each one may define a coupling modifier, multiplying the SM value (for (a) and (b) below) or plain generating a non-SM-existing process (in (c), (d), (e)).

By measuring HH production one can thus exclude these theories.

(a) and (b)
exist in SM

(c), (d), (e)
do not exist in SM
→ related couplings
are =0; >0 in BSM



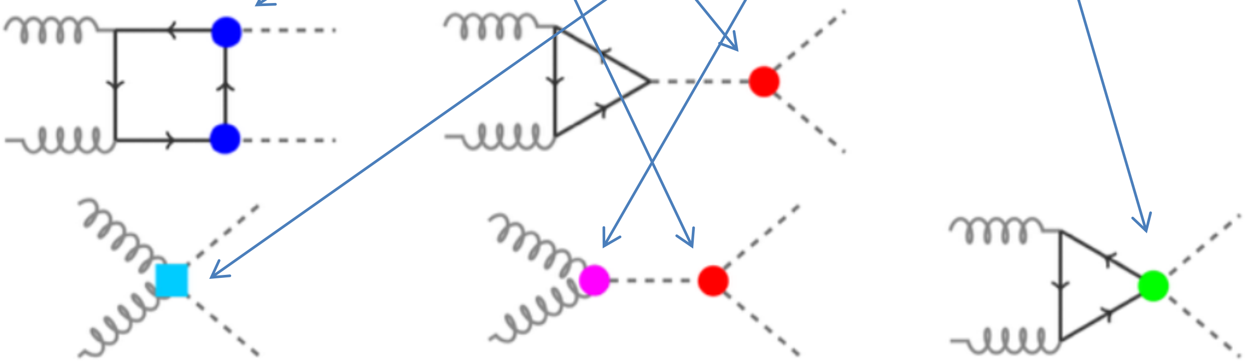
Higgs Effective Lagrangian

Higgs effective Lagrangian after EWSB, neglecting couplings with light fermions $H \to \frac{1}{\sqrt{2}}\begin{pmatrix} 0 \\ v+h \end{pmatrix}$

$$\mathcal{L}_h = \frac{1}{2}\partial_\mu h \partial^\mu h - \frac{1}{2}m_h^2 h^2 - \kappa_\lambda \lambda_{SM} v h^3 - \frac{m_t}{v}(v + \kappa_t h + \frac{c_2}{v} h h)(\bar{t_L} t_R + h.c.)$$
$$+ \frac{1}{4}\frac{\alpha_s}{3\pi v}(c_g h - \frac{c_{2g}}{2v} h h) G^{\mu\nu} G_{\mu\nu}$$

where in EFT linear realisation c_{2g} = -c_g. Constraint relaxed.

ggF Higgs pairs production diagrams



BSM parameter space

We are dealing with a 5-dimensional parameter; each point corresponds to a different phenomenology of HH production

Indeed, the frequency of production varies drastically, and also the kinematics in the final state changes quickly

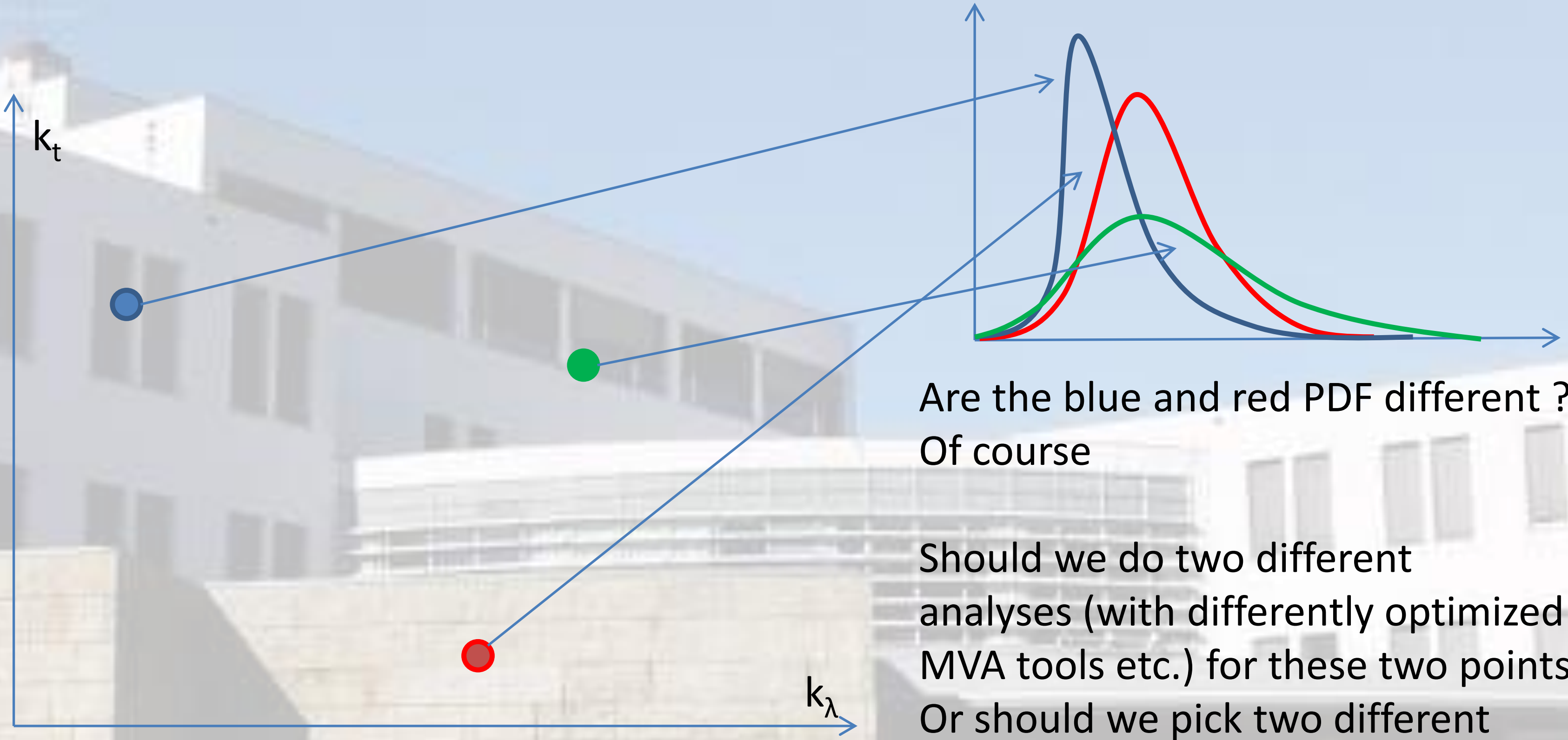
How can experiments test such a vast parameter space? One cannot do hypothesis testing with, e.g., 10^5 theories (imagining a grid of 10 values per parameter)

We are helped by the fact that varying the five couplings, the final state (HH) is the same) → an experimental search looking for HH pairs can be sensitive to many theories at once; but not optimally!

→ **it is advantageous to define "benchmarks", theory points that are "representative"** as they describe situations that are maximally different and separately searchable

The idea is then to find points of parameter space on which to optimize the experimental searches, such that the optimized search remains sensitive to the largest possible range of parameter space points

Parameter space scan



Are the blue and red PDF different ?
Of course

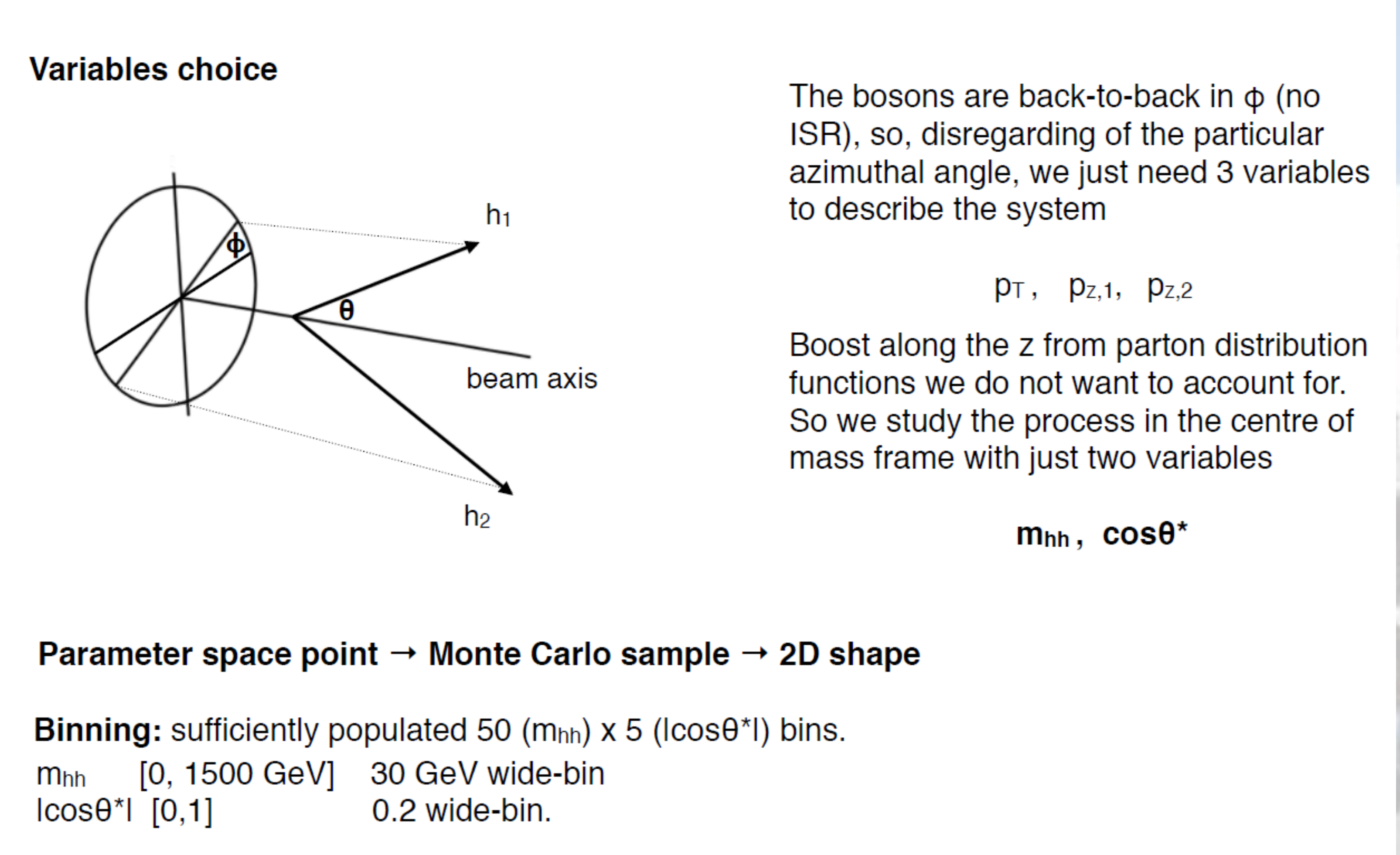
Should we do two different analyses (with differently optimized MVA tools etc.) for these two points? Or should we pick two different points?

In other words**, if we can only train two MVA**, the question is whether we should teach them to distinguish background from the blue and red signals, or from the blue and green, e.g.

→ our choice impacts the sensitivity we have to a BSM hh signal in the full parameter space... so we have to take it wisely!

Dimensionality Reduction!

We are interested in distinguishing different production mechanisms, so we can focus on what really tells apart different BSM models that produce HH pairs before any decay / hadronization / detector / reconstruction effect


h₁
ϕ
θ
beam axis
h₂

Variables choice

The bosons are back-to-back in ϕ (no ISR), so, disregarding of the particular azimuthal angle, we just need 3 variables to describe the system

p_T , $p_{z,1}$, $p_{z,2}$

Boost along the z from parton distribution functions we do not want to account for. So we study the process in the centre of mass frame with just two variables

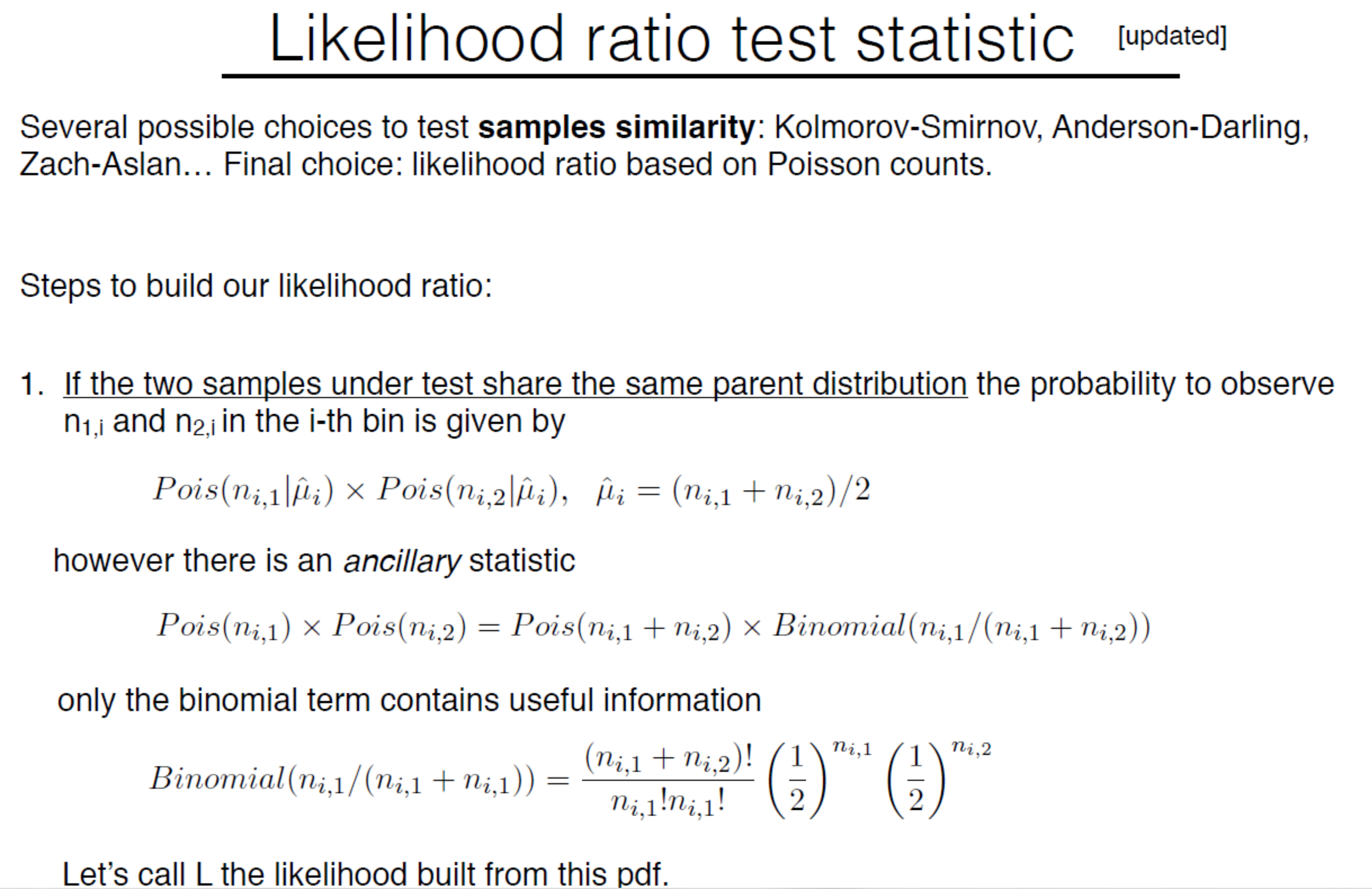
$\mathbf{m_{hh}}$ **,** $\mathbf{\cos\theta^*}$

Parameter space point → Monte Carlo sample → 2D shape

Binning: sufficiently populated 50 (m_{hh}) x 5 ($|\cos\theta^*|$) bins.

m_{hh} [0, 1500 GeV] 30 GeV wide-bin
$|\cos\theta^*|$ [0,1] 0.2 wide-bin.

Construction of a Test Statistic



Likelihood ratio test statistic [updated]

Several possible choices to test **samples similarity**: Kolmorov-Smirnov, Anderson-Darling, Zach-Aslan… Final choice: likelihood ratio based on Poisson counts.

Steps to build our likelihood ratio:

1. If the two samples under test share the same parent distribution the probability to observe $n_{1,i}$ and $n_{2,i}$ in the i-th bin is given by

$$Pois(n_{i,1}|\hat{\mu}_i) \times Pois(n_{i,2}|\hat{\mu}_i), \quad \hat{\mu}_i = (n_{i,1} + n_{i,2})/2$$

however there is an *ancillary* statistic

$$Pois(n_{i,1}) \times Pois(n_{i,2}) = Pois(n_{i,1} + n_{i,2}) \times Binomial(n_{i,1}/(n_{i,1} + n_{i,2}))$$

only the binomial term contains useful information

$$Binomial(n_{i,1}/(n_{i,1} + n_{i,1})) = \frac{(n_{i,1} + n_{i,2})!}{n_{i,1}!n_{i,1}!} \left(\frac{1}{2}\right)^{n_{i,1}} \left(\frac{1}{2}\right)^{n_{i,2}}$$

Let's call L the likelihood built from this pdf.

Likelihood ratio test statistic

2. If the two samples are equal (*saturated hypothesis*) the pdf is just

$$Binomial(n_{i,1} = n_{i,2} = \hat{\mu_i}) = \frac{(2\hat{\mu_i})!}{(\hat{\mu_i}!)^2}\left(\frac{1}{2}\right)^{2\hat{\mu_i}}$$

Let's call L_S the likelihood associated to this pdf.

3. The (log-) likelihood ratio is defined as

$$TS = 2\ log\left(\frac{L}{L_S}\right) = 2\sum_{i=1}^{N_{bins}} log(n_{i,1}!) + log(n_{i,2}!) - 2log\left(\frac{n_{i,1}+n_{i,2}}{2}!\right)$$

thanks to Wilks theorem TS is χ^2 distributed and can be used directly as an ordering parameter.

In fact: a two-sample test produces in general a number (KS distance, AD distance, chi-square, etc) whose PDF is not distribution-independent; being distribution independent is a big advantage for a TS

Clustering algorithm

$TS_{ij} > TS_{kl}$ → i and j are more similar to each other then kl

Steps:

1) Identify each sample as one element cluster

2) define cluster-to-cluster similarity as $TS^{min} = min(TS_{ij})$ where *i* runs on first cluster elements and *j* on the second one

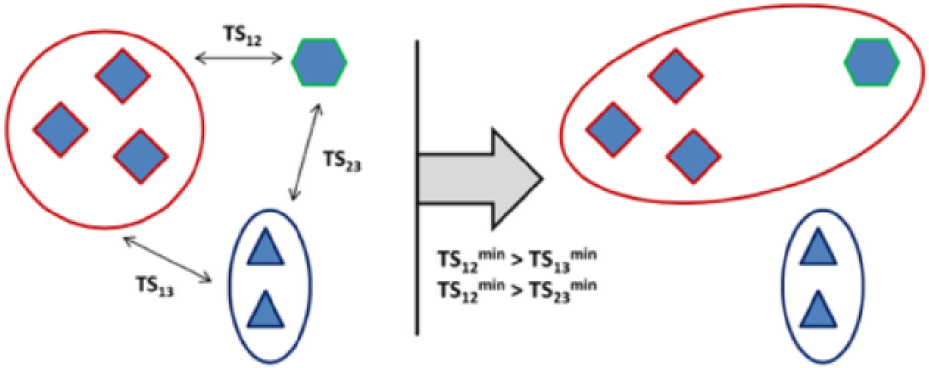
3) merge the pair of clusters with highest TS^{min}

4) repeat until the desired number of clusters $\mathbf{N_{clus}}$ is reached

5) identify the benchmark *k* of each cluster as the one with the highest

$$TS_k^{min} = min_i(TS_{ki})$$

where *i* runs on the cluster elements.

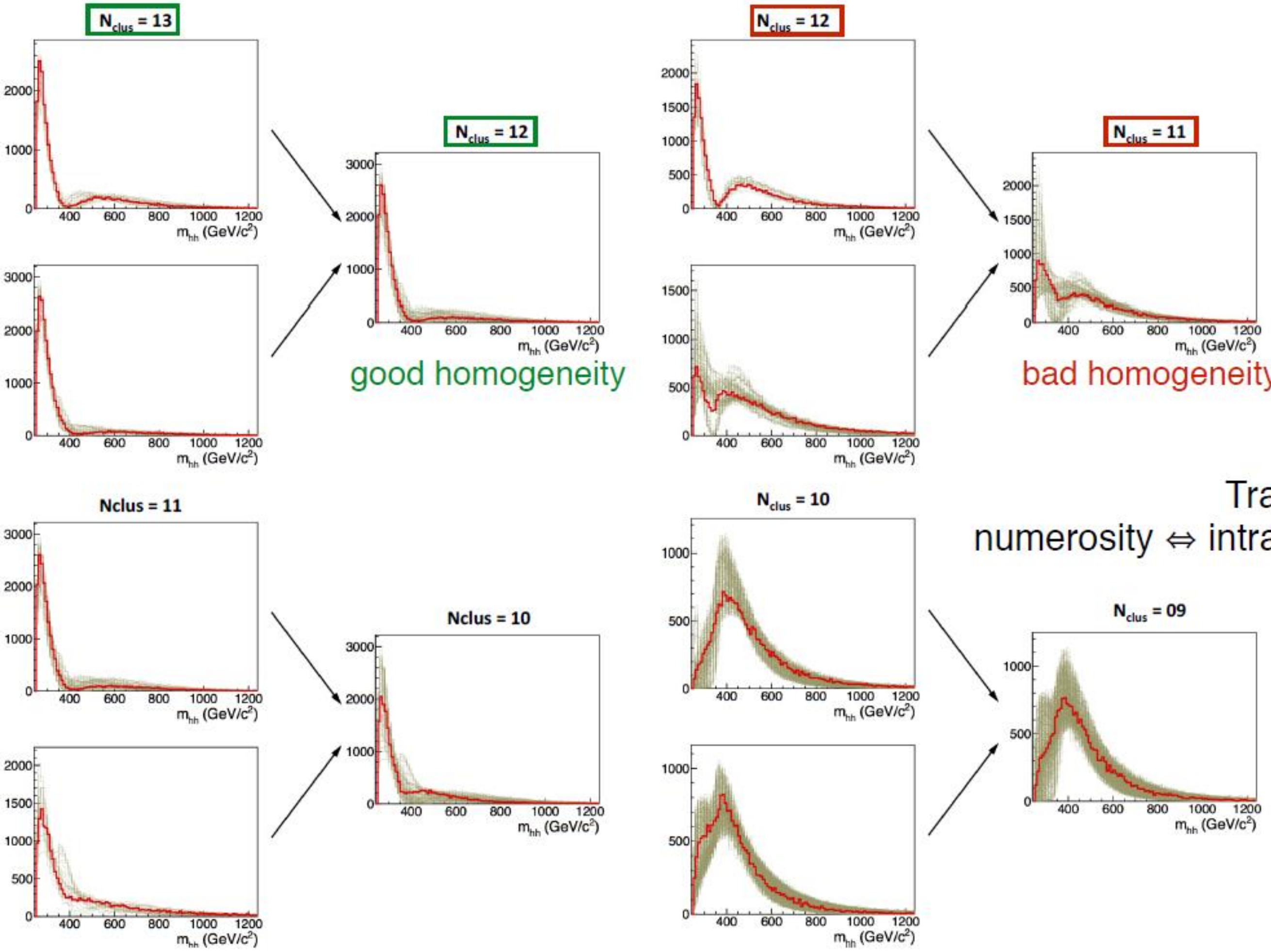

TS_{12}
TS_{23}
TS_{13}
$TS_{12}^{min} > TS_{13}^{min}$
$TS_{12}^{min} > TS_{23}^{min}$

$\mathbf{N_{clus}}$ is the only free parameter, fixed a posteriori.

N_{clus} = 12 choice

Looking for $N_{clus} \sim O(10)$…

Clusters pair merged stepping from final cluster number N_{clus} to N_{clus} -1



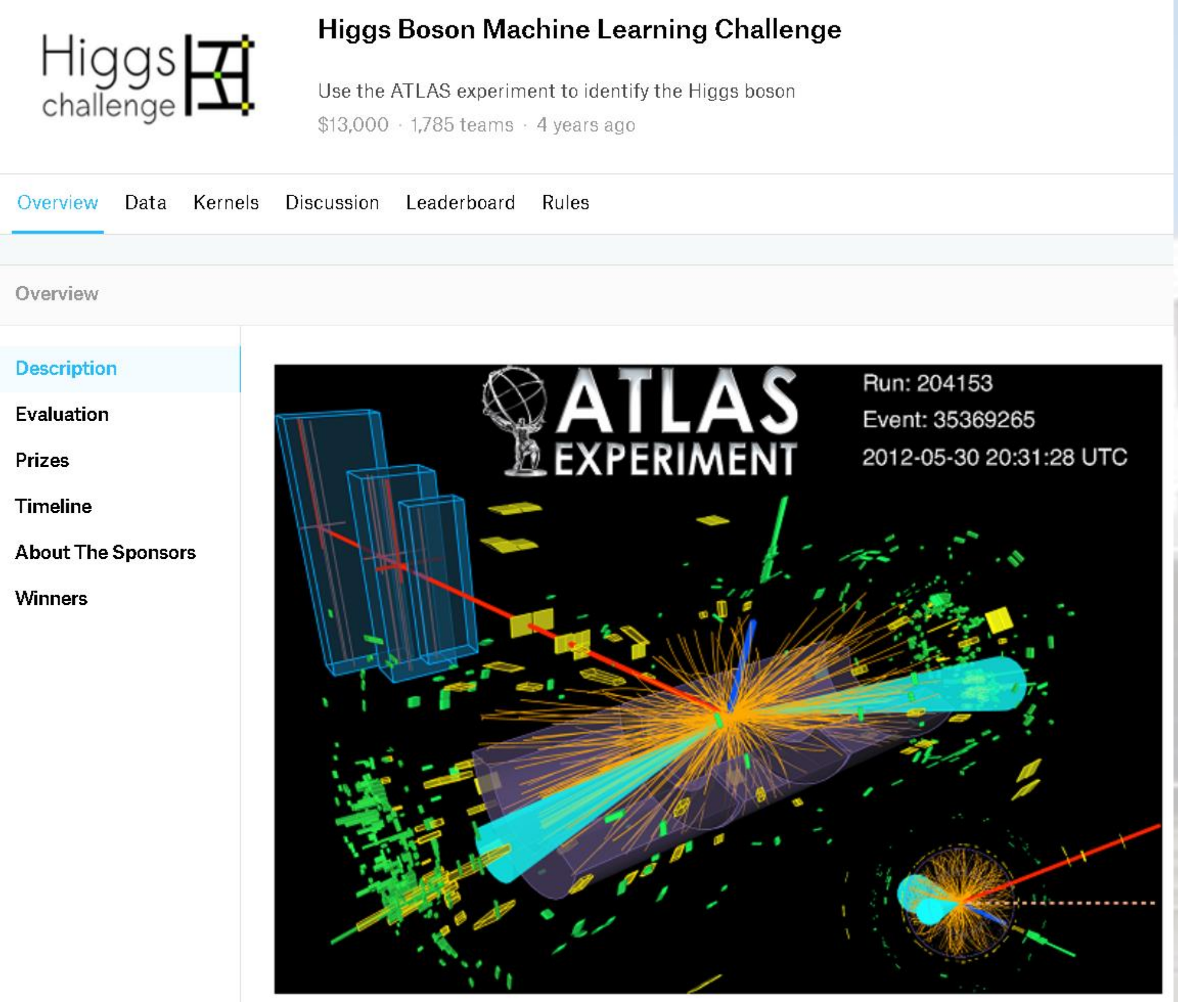
good homogeneity

bad homogeneity

Tradeoff:
numerosity ⇔ intra-cluster homogeneity

The Higgs Kaggle Challenge

https://www.kaggle.com/c/higgs-boson#description



Higgs Boson Machine Learning Challenge

Use the ATLAS experiment to identify the Higgs boson

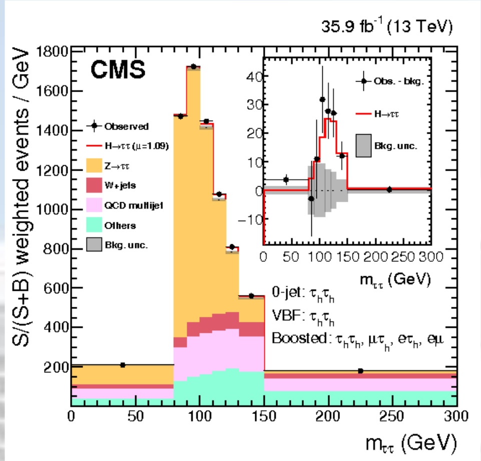
$13,000 · 1,785 teams · 4 years ago

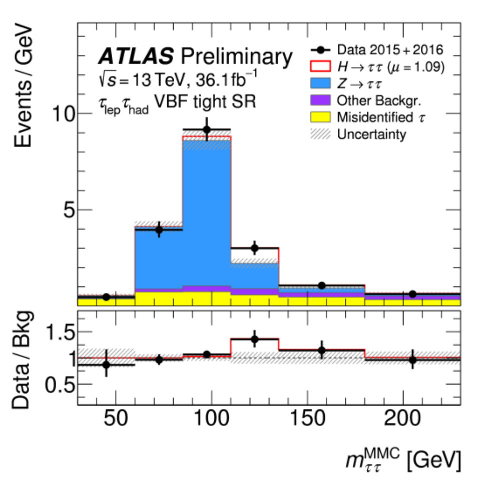
The problem

The gluon-fusion production process pp→H→ττ is not easy to distinguish from large backgrounds in LHC data. CMS and ATLAS competed to "observe" the decay long after the 2012 discovery in the easier H→ZZ, H→γγ channels.

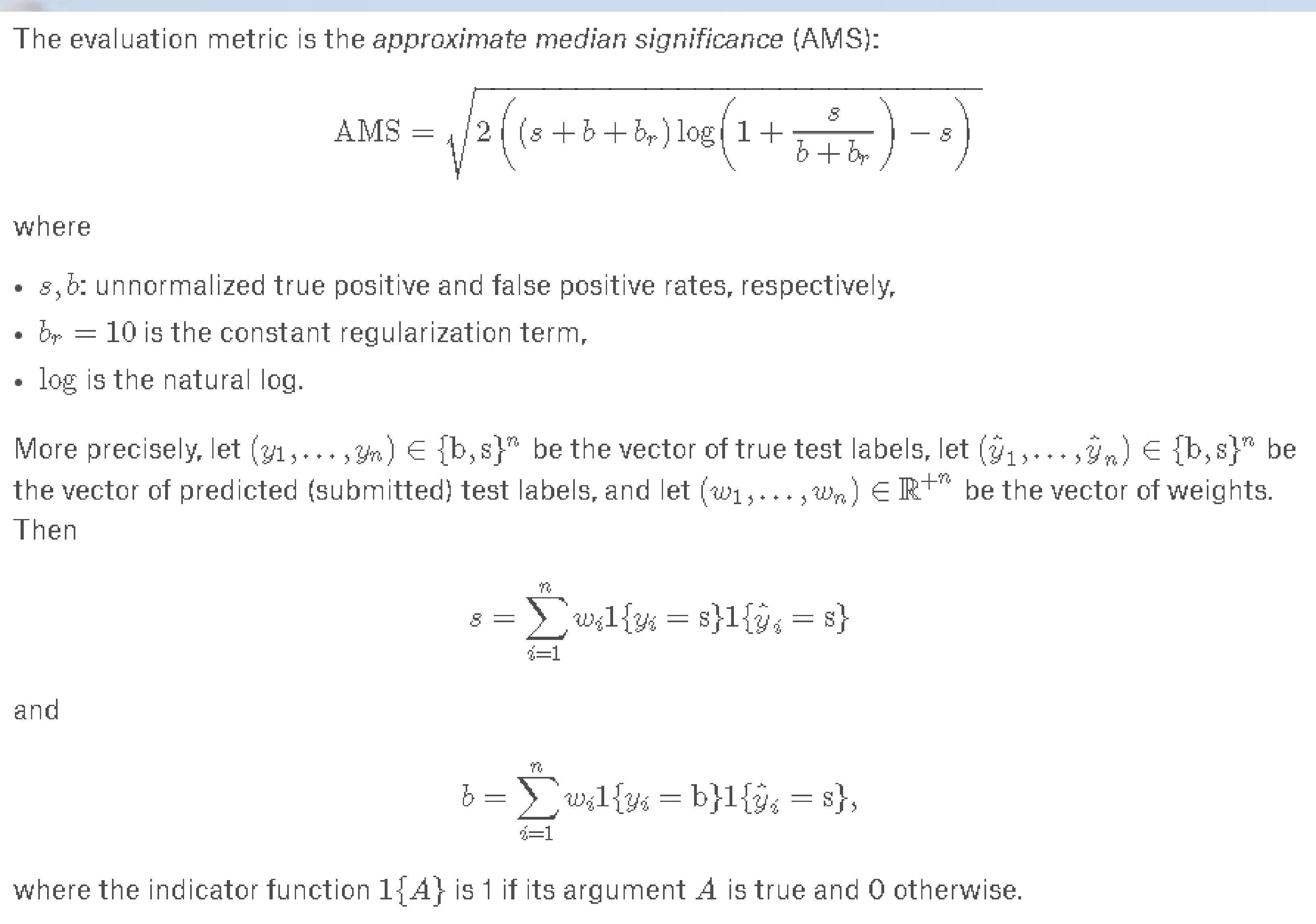
In 2014 ATLAS loaded on Kaggle high-level features of signal and background processes (30 per event), challenging users to tell signal and background apart

In practice the request was to optimize an "approximate median significance" measure on data surviving a selection decided by the user with a discriminant of their craft and tuning





AMS



The evaluation metric is the *approximate median significance* (AMS):

$$\text{AMS} = \sqrt{2\left((s+b+b_r)\log\left(1+\frac{s}{b+b_r}\right)-s\right)}$$

where

- s, b: unnormalized true positive and false positive rates, respectively,
- $b_r = 10$ is the constant regularization term,
- $\log$ is the natural log.

More precisely, let $(y_1,\ldots,y_n) \in \{\text{b},\text{s}\}^n$ be the vector of true test labels, let $(\hat{y}_1,\ldots,\hat{y}_n) \in \{\text{b},\text{s}\}^n$ be the vector of predicted (submitted) test labels, and let $(w_1,\ldots,w_n) \in \mathbb{R}^{+n}$ be the vector of weights. Then

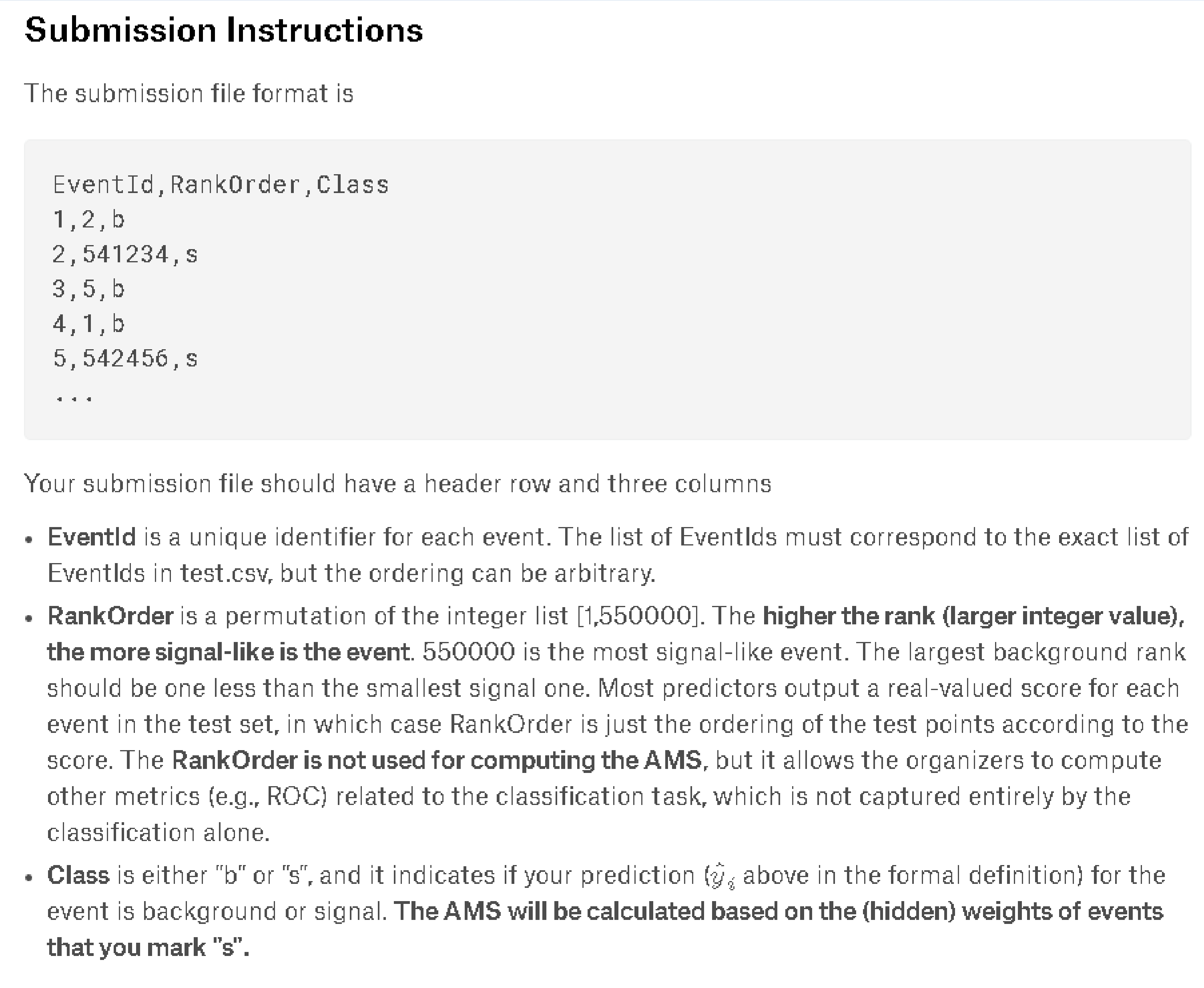
$$s = \sum_{i=1}^{n} w_i 1\{y_i = \text{s}\} 1\{\hat{y}_i = \text{s}\}$$

and

$$b = \sum_{i=1}^{n} w_i 1\{y_i = \text{b}\} 1\{\hat{y}_i = \text{s}\},$$

where the indicator function $1\{A\}$ is 1 if its argument A is true and 0 otherwise.

Details



Submission Instructions

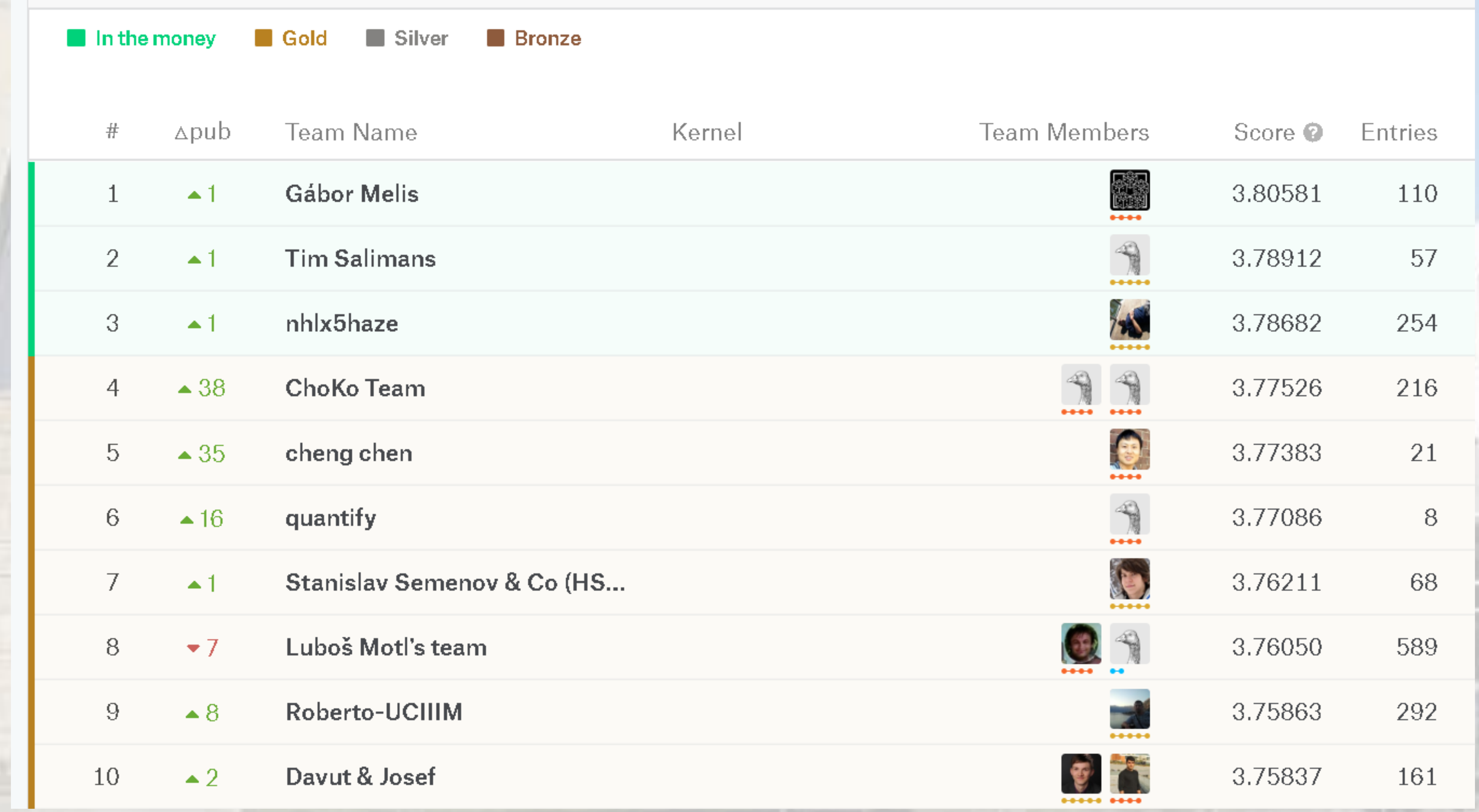
The submission file format is

```
EventId,RankOrder,Class
1,2,b
2,541234,s
3,5,b
4,1,b
5,542456,s
...
```

Your submission file should have a header row and three columns

- **EventId** is a unique identifier for each event. The list of EventIds must correspond to the exact list of EventIds in test.csv, but the ordering can be arbitrary.
- **RankOrder** is a permutation of the integer list [1,550000]. The **higher the rank (larger integer value), the more signal-like is the event**. 550000 is the most signal-like event. The largest background rank should be one less than the smallest signal one. Most predictors output a real-valued score for each event in the test set, in which case RankOrder is just the ordering of the test points according to the score. The **RankOrder is not used for computing the AMS**, but it allows the organizers to compute other metrics (e.g., ROC) related to the classification task, which is not captured entirely by the classification alone.
- **Class** is either "b" or "s", and it indicates if your prediction ($\hat{y}_i$ above in the formal definition) for the event is background or signal. **The AMS will be calculated based on the (hidden) weights of events that you mark "s".**

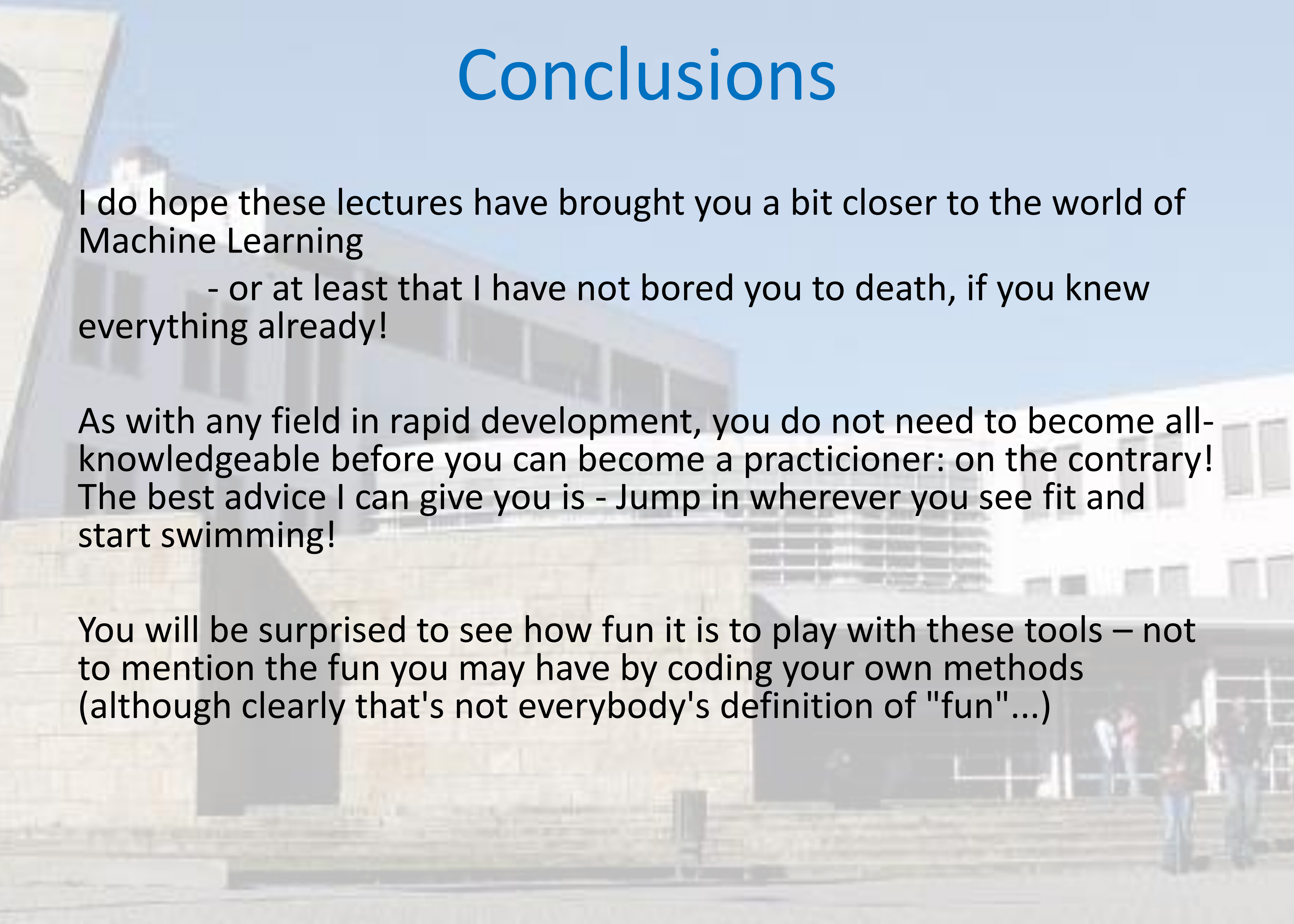
Leaderboard



🟩 In the money 🟨 Gold ⬜ Silver 🟫 Bronze

#	▵pub	Team Name	Kernel	Team Members	Score	Entries
1	▲1	Gábor Melis			3.80581	110
2	▲1	Tim Salimans			3.78912	57
3	▲1	nhlx5haze			3.78682	254
4	▲38	ChoKo Team			3.77526	216
5	▲35	cheng chen			3.77383	21
6	▲16	quantify			3.77086	8
7	▲1	Stanislav Semenov & Co (HS...			3.76211	68
8	▼7	Luboš Motl's team			3.76050	589
9	▲8	Roberto-UCIIIM			3.75863	292
10	▲2	Davut & Josef			3.75837	161

CONCLUSIONS



Conclusions

I do hope these lectures have brought you a bit closer to the world of Machine Learning

- or at least that I have not bored you to death, if you knew everything already!

As with any field in rapid development, you do not need to become all-knowledgeable before you can become a practicioner: on the contrary! The best advice I can give you is - Jump in wherever you see fit and start swimming!

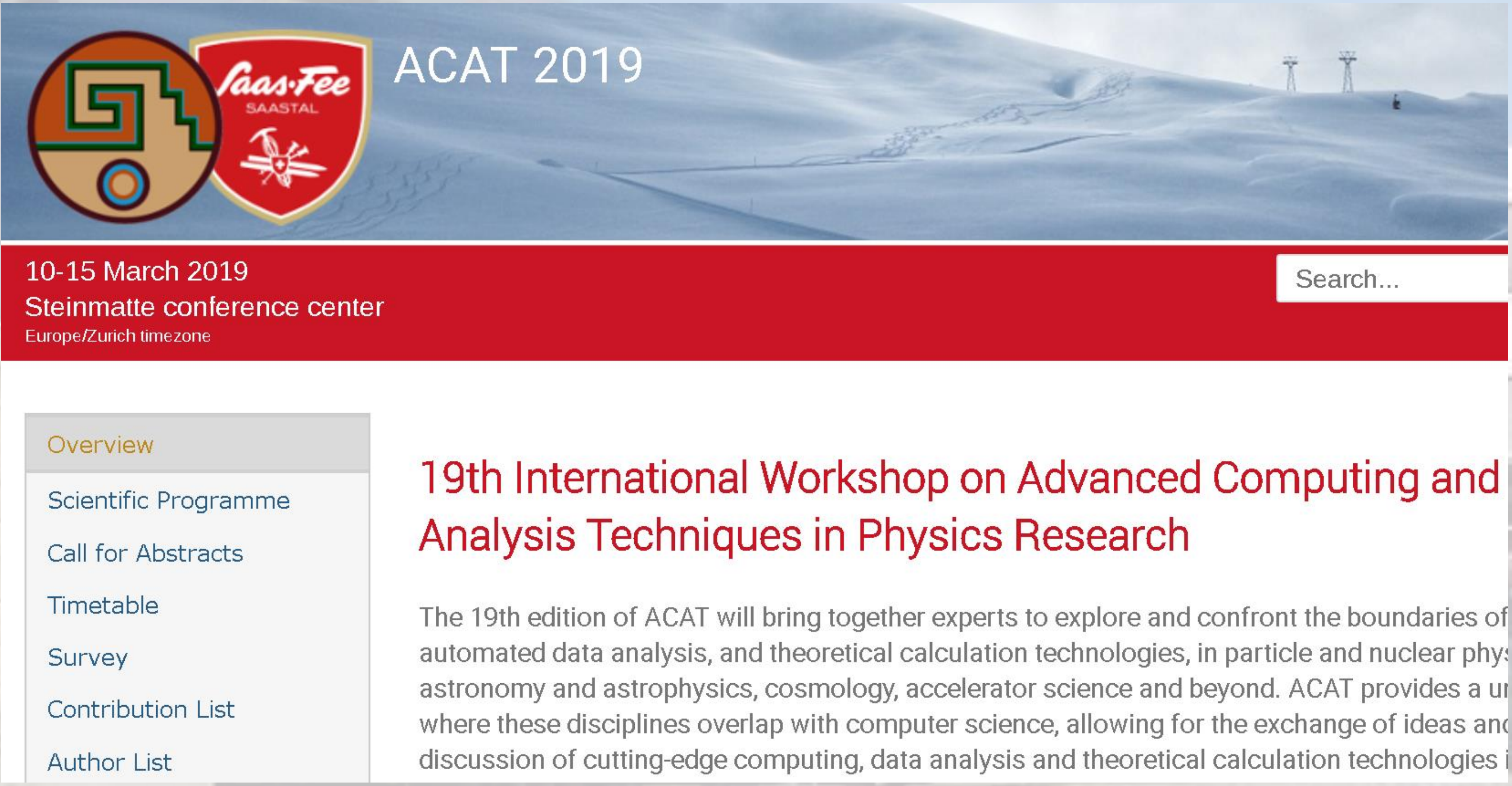
You will be surprised to see how fun it is to play with these tools – not to mention the fun you may have by coding your own methods (although clearly that's not everybody's definition of "fun"...)

Some take-away bits

- Don't look for complex solutions when simple ones work well
  - Hastie: often kNN performs best !
  - Useful to understand easy tools before you can exploit hard ones
- As powerful as individual tools are, they aren't the answer to the question "what is best"
  - the mastery of the data analyzer is to optimally combine the proper ingredients to achieve their task, and then add the killing bit that is only useful in the particular application at hand
- In NN design, the loss function is where the money is
  - improvements in the inputs have also large impact in results (see tutorials)
  - smart scanning for absolute minimum is important
  - attempts to improve on already reasonably flexible designs likely to not give as big gains

Thank you!

A few interesting applications from the ACAT conference



ACAT 2019

10-15 March 2019
Steinmatte conference center
Europe/Zurich timezone

19th International Workshop on Advanced Computing and Analysis Techniques in Physics Research

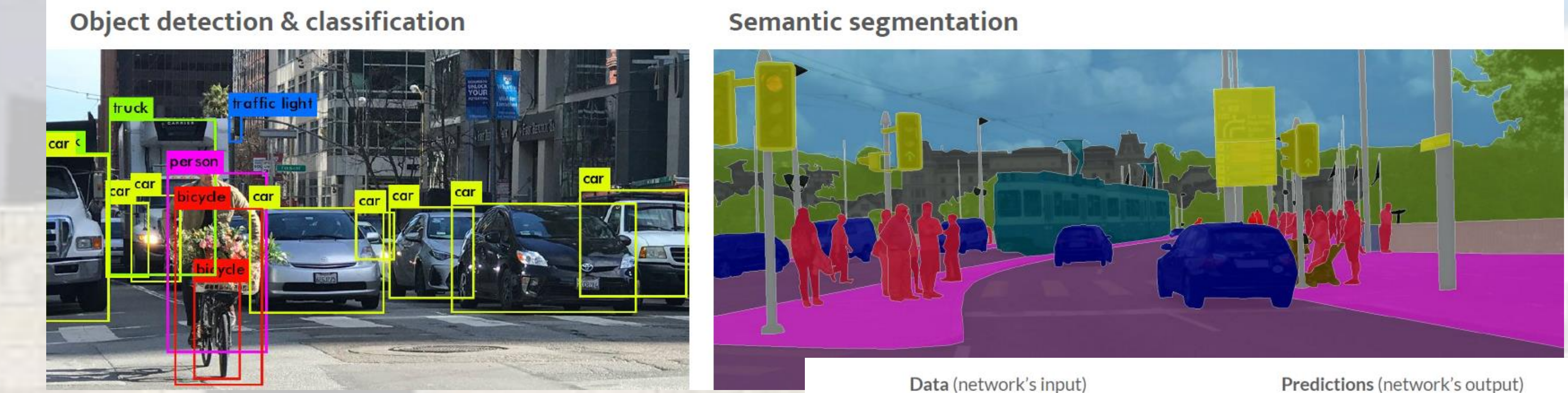
The 19th edition of ACAT will bring together experts to explore and confront the boundaries of automated data analysis, and theoretical calculation technologies, in particle and nuclear phy astronomy and astrophysics, cosmology, accelerator science and beyond. ACAT provides a u where these disciplines overlap with computer science, allowing for the exchange of ideas an discussion of cutting-edge computing, data analysis and theoretical calculation technologies i

Neutrino interactions: reconstruction with CNN

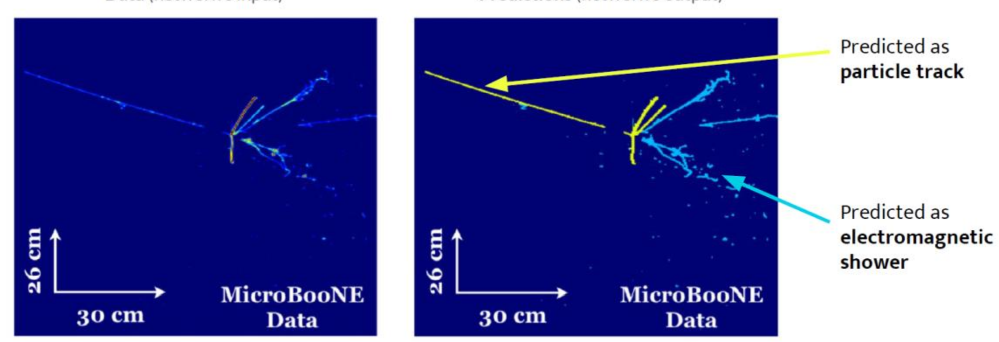
The use of CNN for reconstruction of neutrino interactions in LAr TPC detectors has been investigated in arXiv:1903.05663.

In a talk at ACAT2019, Laura Domine explained the problem, the architecture used to solve it, and the performance achieved

The idea is to use the semantic segmentation capabilities of CNNs:

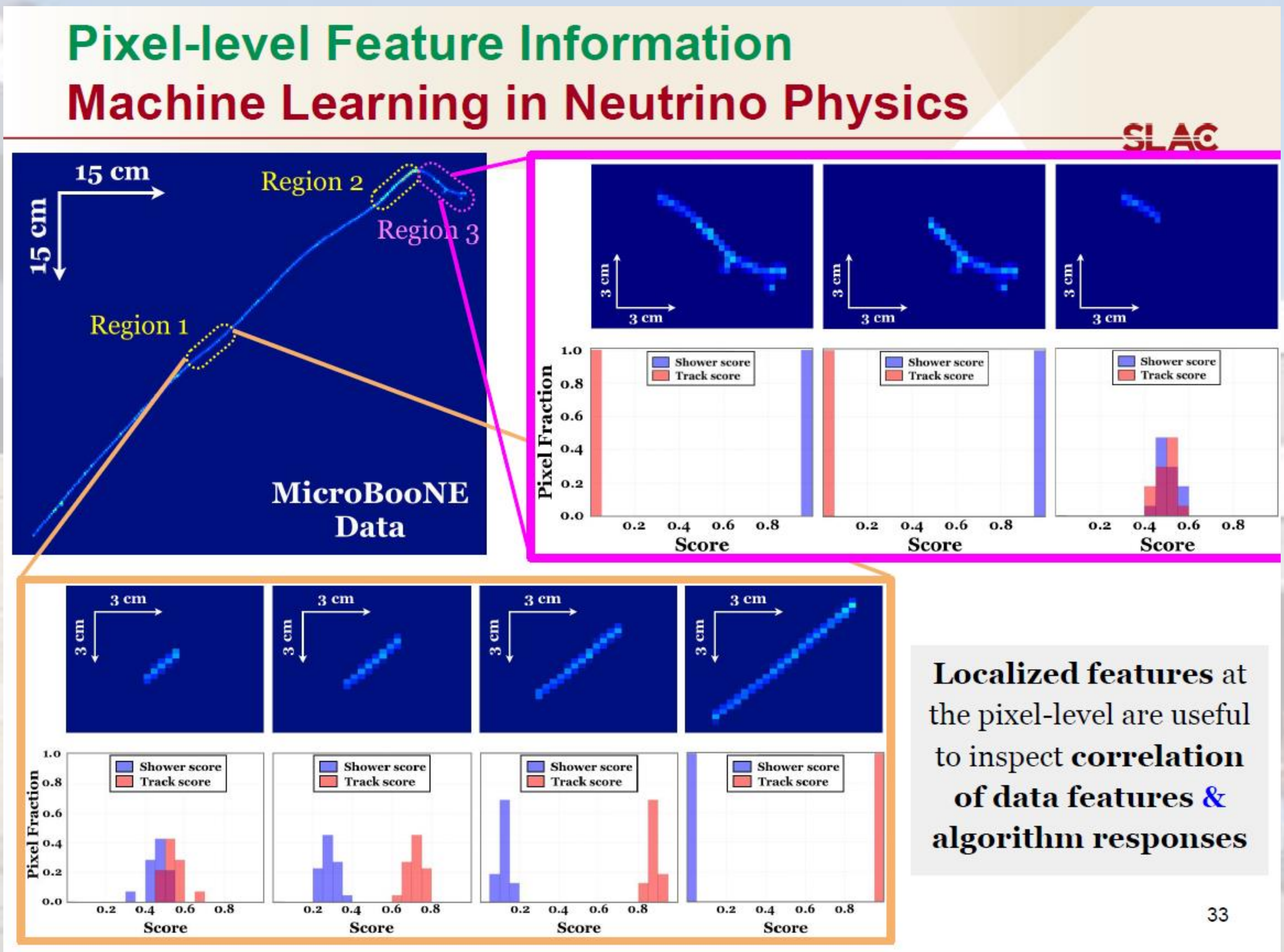


In neutrino interactions, what one cares to measure is energy and direction only. But for this one needs to identify long tracks, EM showers, cosmics, nuclear recoils...


Data (network's input)
Predictions (network's output)
Predicted as particle track
Predicted as electromagnetic shower
26 cm
30 cm
MicroBooNE Data
26 cm
30 cm
MicroBooNE Data

A Deep Neural Network for Pixel-Level Electromagnetic Particle Identification in the MicroBooNE Liquid Argon Time Projection Chamber. (arxiv:1808.07269)

Neutrino event reconstruction

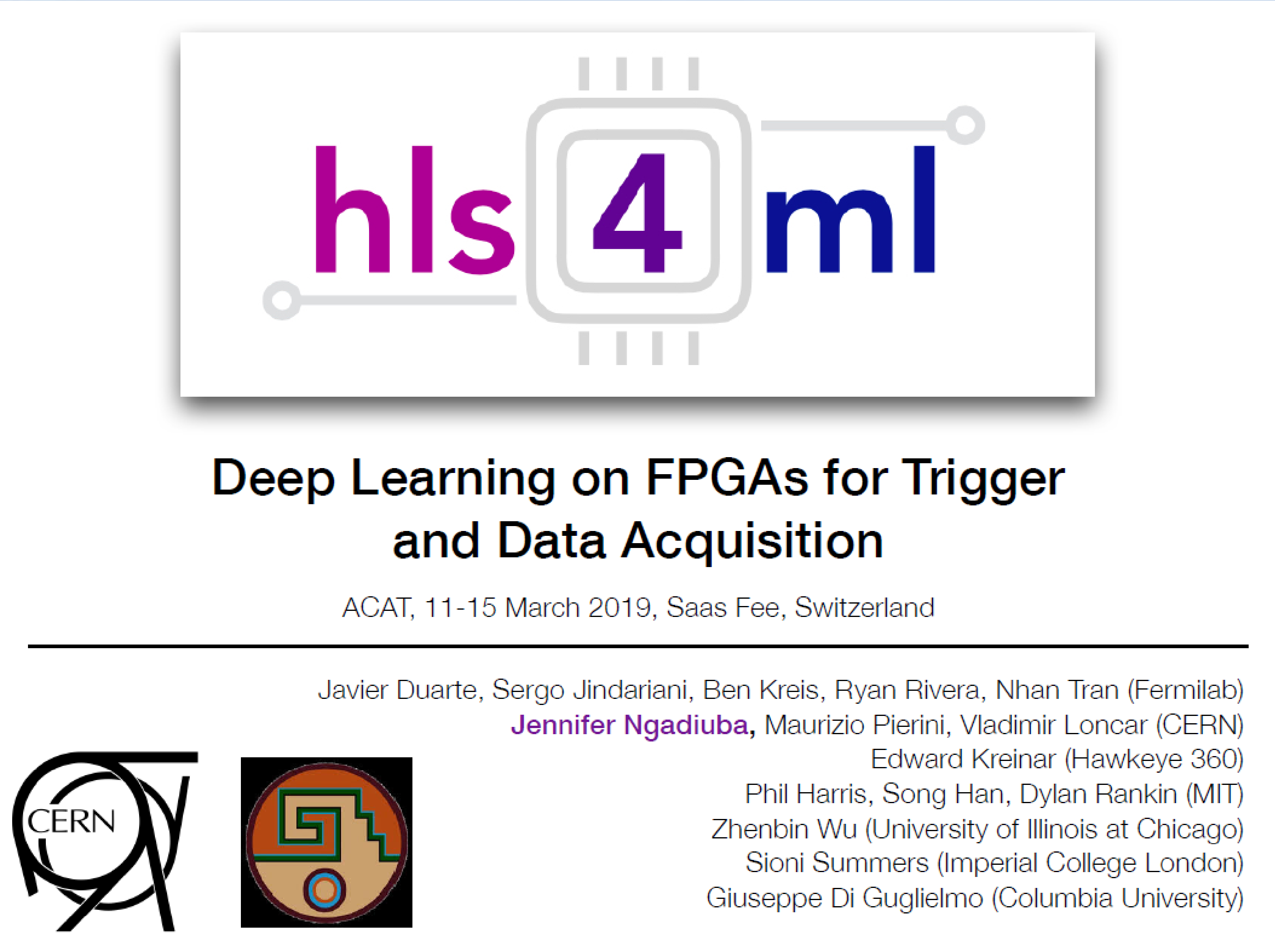


Pixel-level Feature Information

Machine Learning in Neutrino Physics

Localized features at the pixel-level are useful to inspect **correlation of data features & algorithm responses**

One example of pruning w/ L1



hls4ml

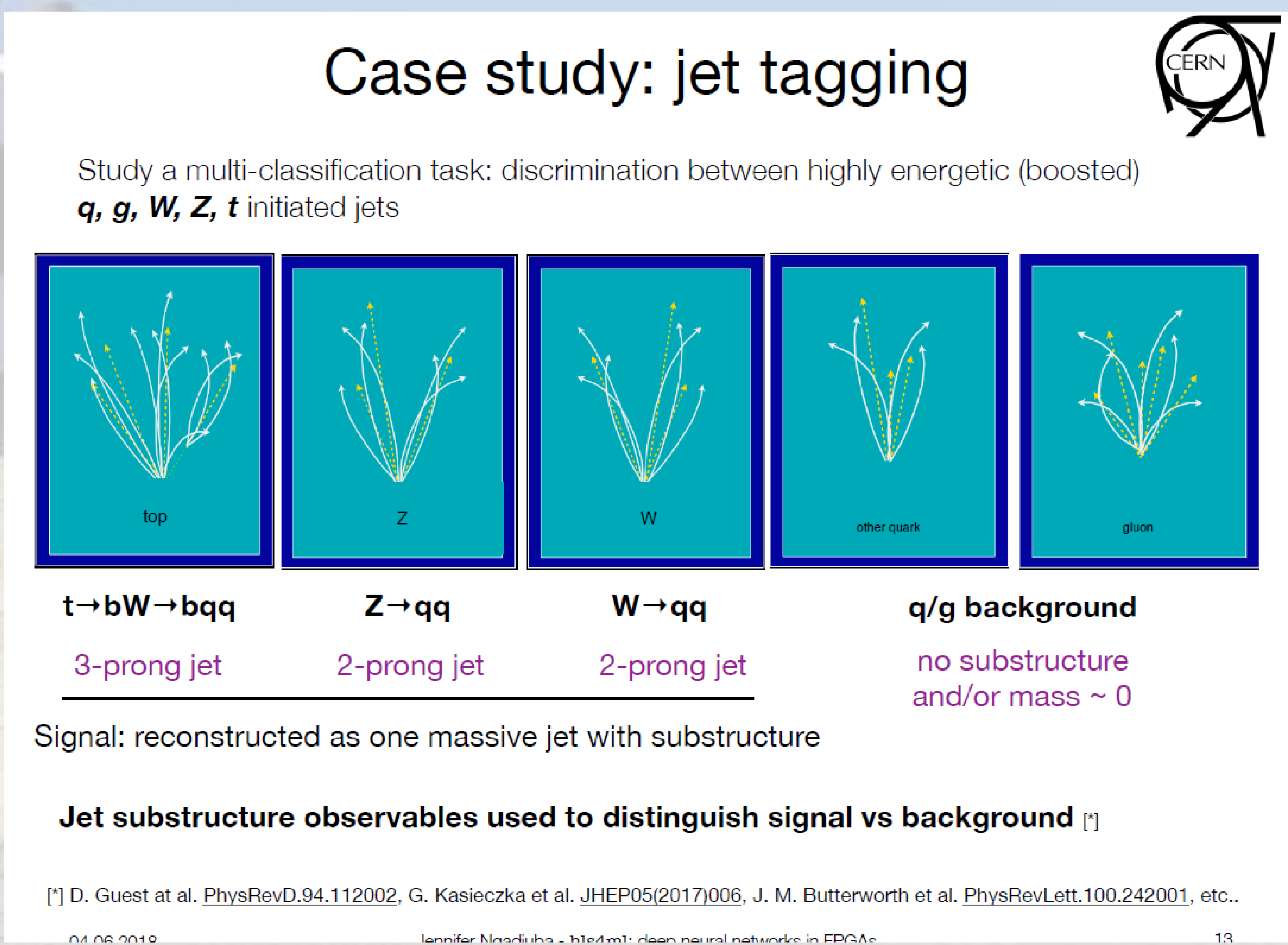
Deep Learning on FPGAs for Trigger and Data Acquisition

ACAT, 11-15 March 2019, Saas Fee, Switzerland

Javier Duarte, Sergo Jindariani, Ben Kreis, Ryan Rivera, Nhan Tran (Fermilab)
Jennifer Ngadiuba, Maurizio Pierini, Vladimir Loncar (CERN)
Edward Kreinar (Hawkeye 360)
Phil Harris, Song Han, Dylan Rankin (MIT)
Zhenbin Wu (University of Illinois at Chicago)
Sioni Summers (Imperial College London)
Giuseppe Di Guglielmo (Columbia University)

CERN

Here the focus is to see if one can implement discriminators for boosted jets in FPGAs



Case study: jet tagging

Study a multi-classification task: discrimination between highly energetic (boosted) ***q, g, W, Z, t*** initiated jets

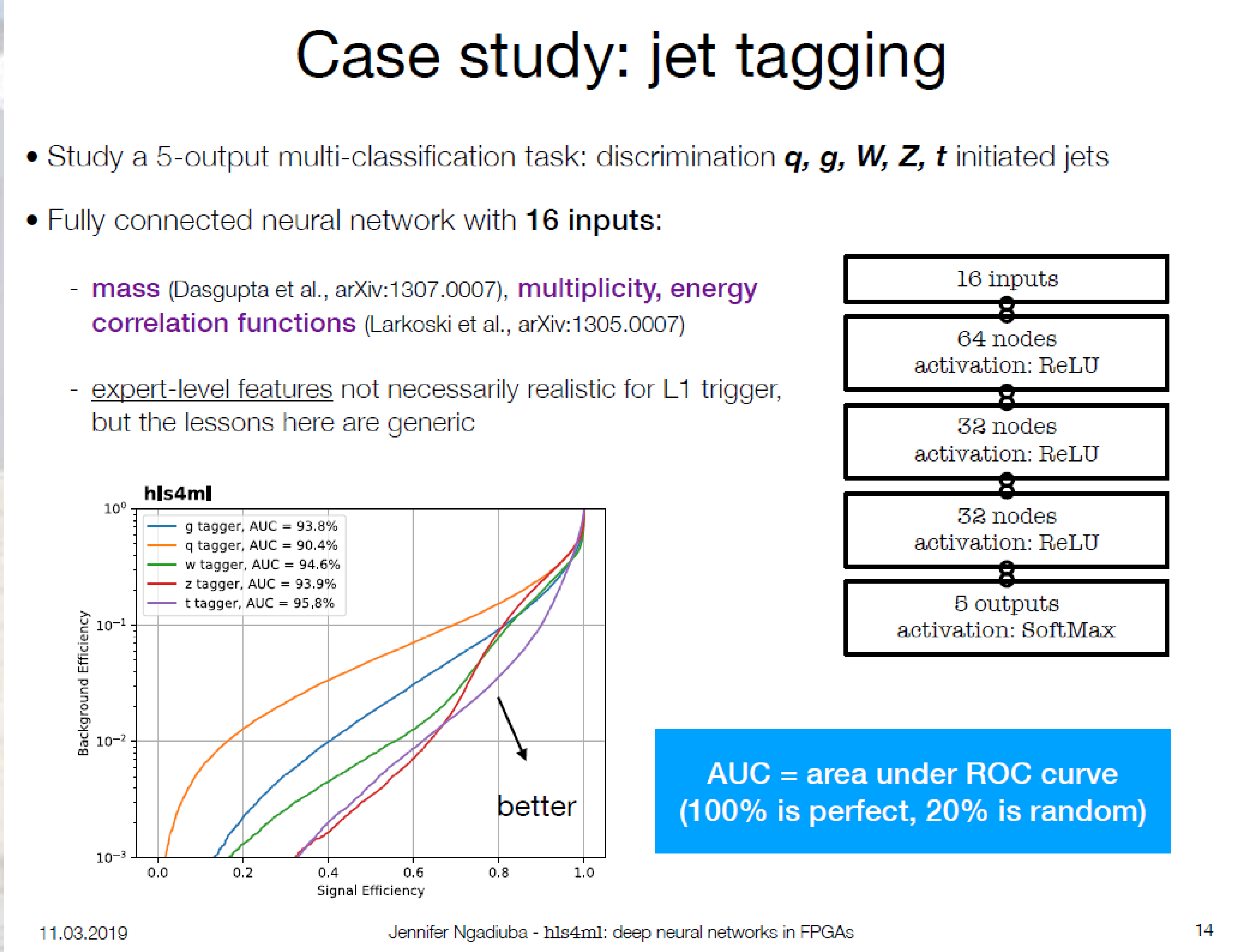
t→bW→bqq	**Z→qq**	**W→qq**	**q/g background**
3-prong jet	2-prong jet	2-prong jet	no substructure and/or mass ~ 0

Signal: reconstructed as one massive jet with substructure

Jet substructure observables used to distinguish signal vs background [*]

[*] D. Guest at al. PhysRevD.94.112002, G. Kasieczka et al. JHEP05(2017)006, J. M. Butterworth et al. PhysRevLett.100.242001, etc..

To manage the identification of heavy particle decays inside hadronic jets, you need to resort to high-level variables, and learn complex image structures



Case study: jet tagging

- Study a 5-output multi-classification task: discrimination ***q, g, W, Z, t*** initiated jets
- Fully connected neural network with **16 inputs**:
  - mass (Dasgupta et al., arXiv:1307.0007), multiplicity, energy correlation functions (Larkoski et al., arXiv:1305.0007)
  - expert-level features not necessarily realistic for L1 trigger, but the lessons here are generic

16 inputs

64 nodes
activation: ReLU

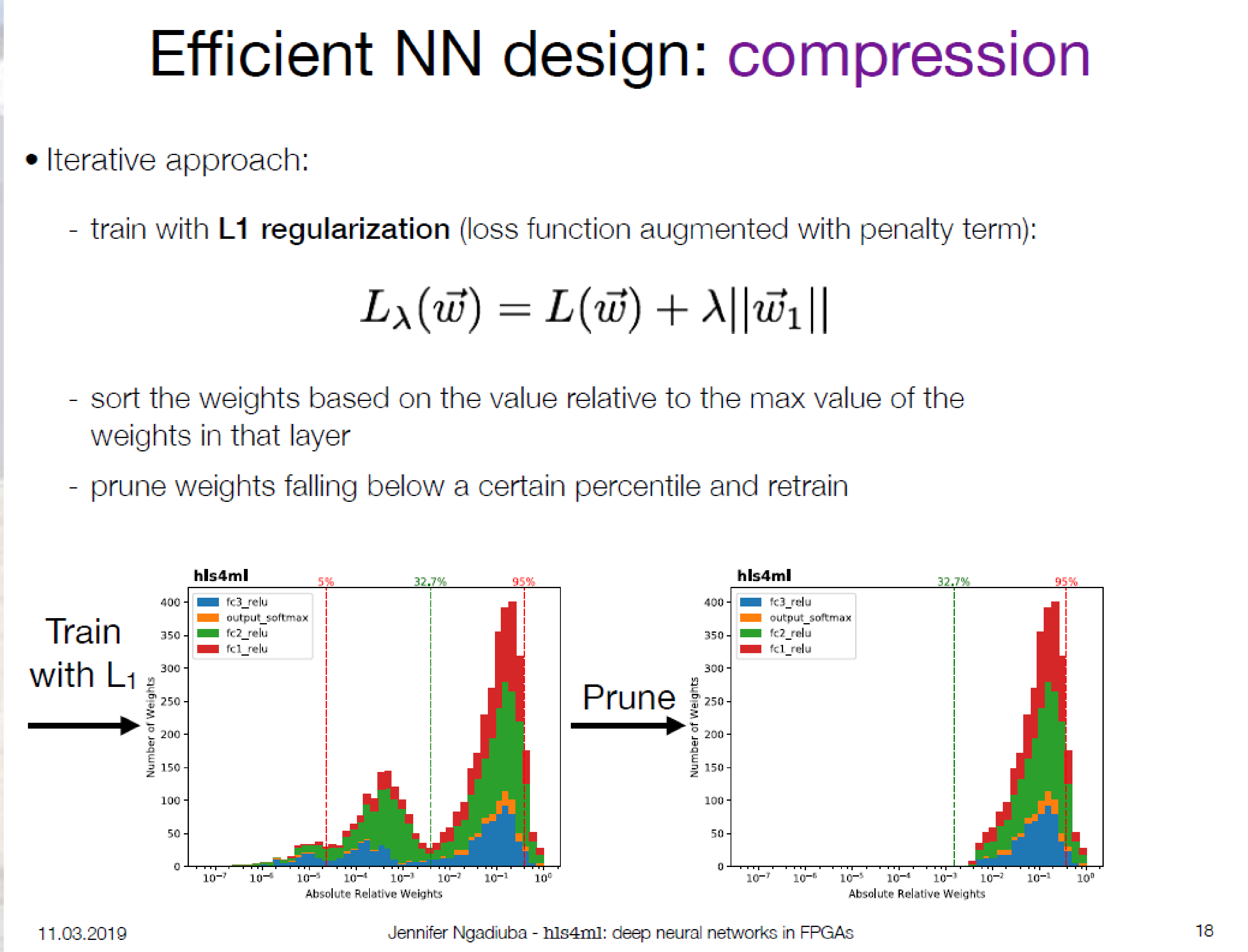
32 nodes
activation: ReLU

32 nodes
activation: ReLU

5 outputs
activation: SoftMax

AUC = area under ROC curve
(100% is perfect, 20% is random)

Starting with a set of features, a reduction of the important features is performed recursively during training, using a Lasso-like norm for the weights



Efficient NN design: compression

- Iterative approach:
  - train with **L1 regularization** (loss function augmented with penalty term):

$$L_\lambda(\vec{w}) = L(\vec{w}) + \lambda||\vec{w}_1||$$

  - sort the weights based on the value relative to the max value of the weights in that layer
  - prune weights falling below a certain percentile and retrain

Train with L_1 → Prune →

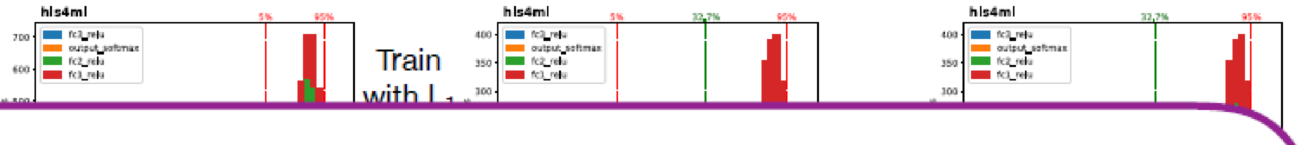
11.03.2019

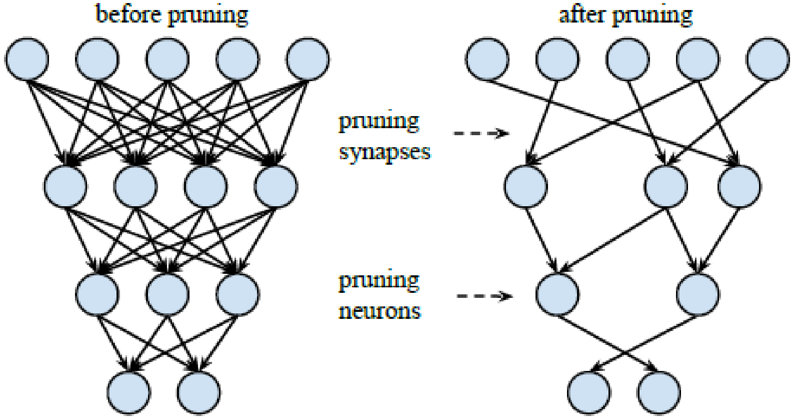
Jennifer Ngadiuba - hls4ml: deep neural networks in FPGAs

18

Efficient NN design: compression

Prune and repeat the training for 7 iterations

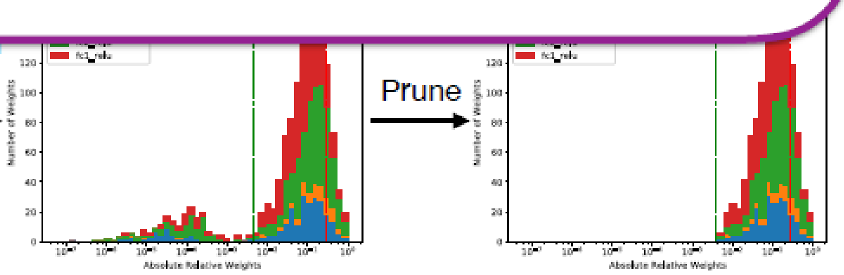




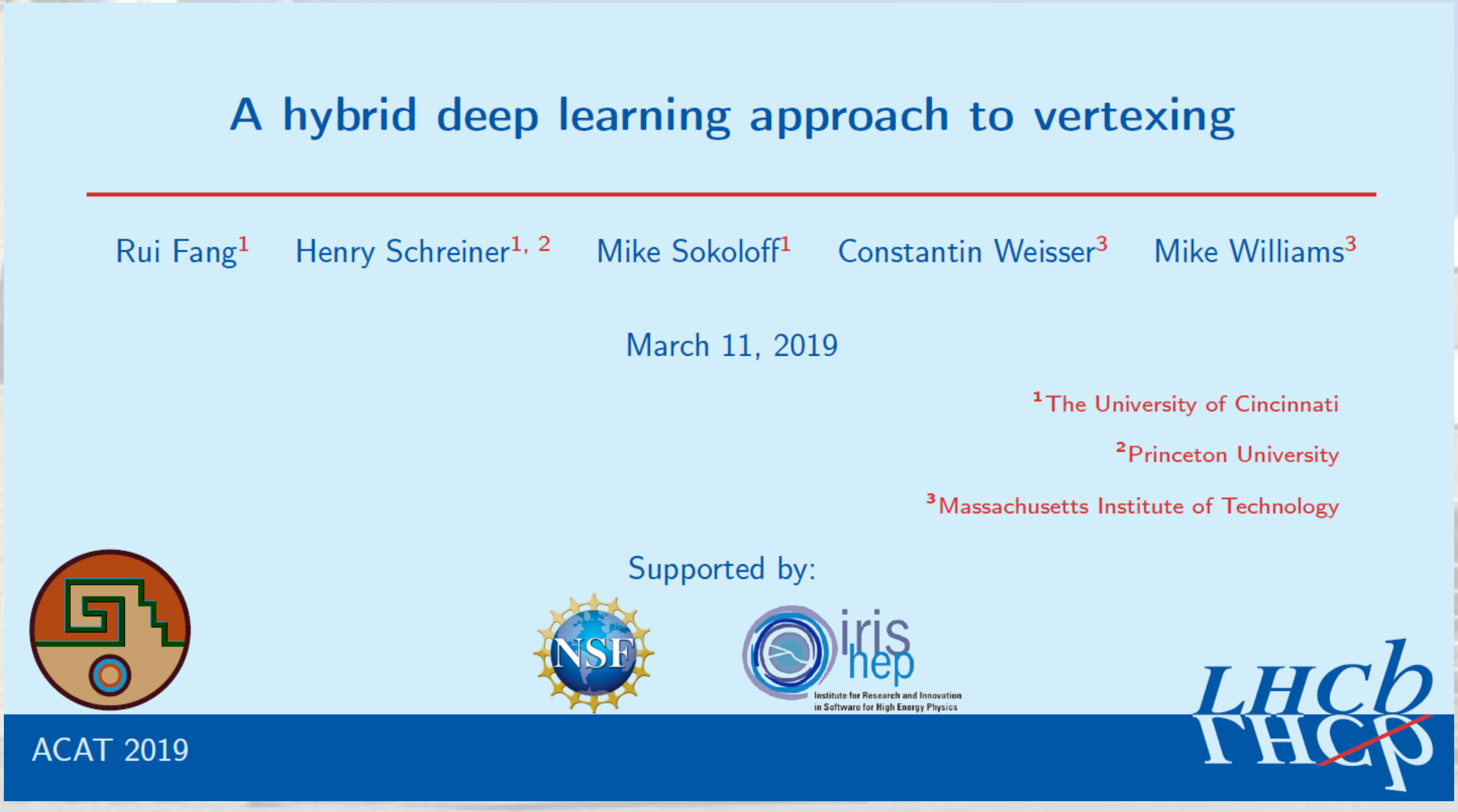
→ 70% reduction of weights and multiplications w/o performance loss

Retrain with L1

7th iteration



Vertex finding in LHCb



A hybrid deep learning approach to vertexing

Rui Fang[1] Henry Schreiner[1, 2] Mike Sokoloff[1] Constantin Weisser[3] Mike Williams[3]

March 11, 2019

[1]The University of Cincinnati

[2]Princeton University

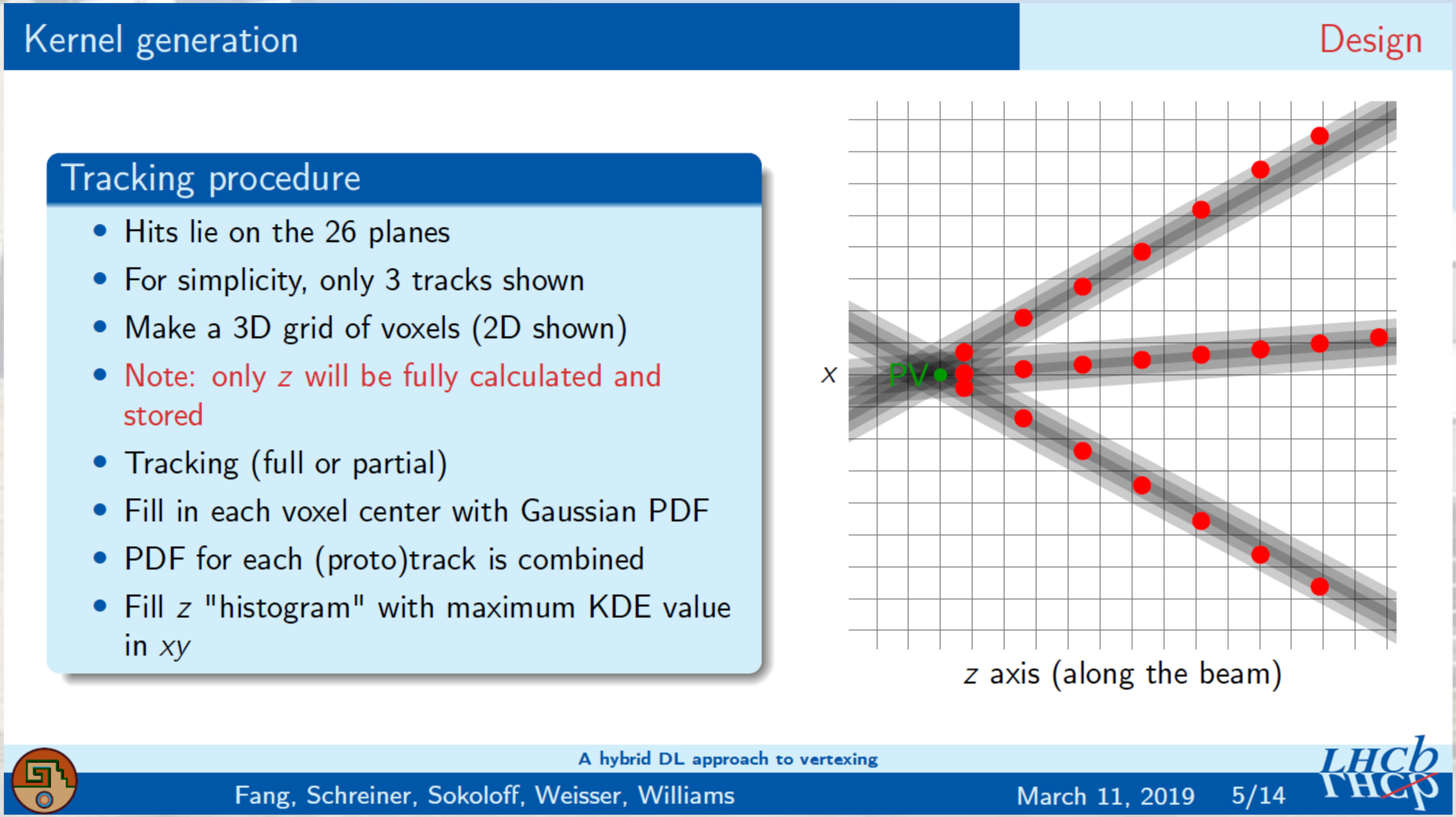
[3]Massachusetts Institute of Technology

Supported by:

NSF
iris hep
Institute for Research and Innovation in Software for High Energy Physics
LHCb

ACAT 2019

Idea: reduce problem to 1D



Kernel generation

Design

Tracking procedure

- Hits lie on the 26 planes
- For simplicity, only 3 tracks shown
- Make a 3D grid of voxels (2D shown)
- Note: only z will be fully calculated and stored
- Tracking (full or partial)
- Fill in each voxel center with Gaussian PDF
- PDF for each (proto)track is combined
- Fill z "histogram" with maximum KDE value in xy

A hybrid DL approach to vertexing

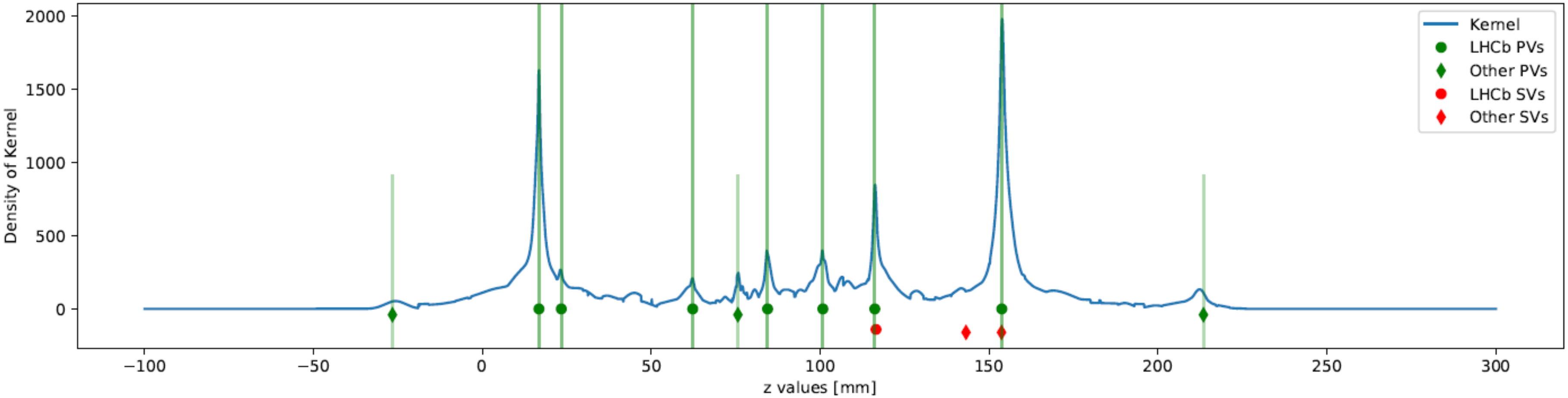
Fang, Schreiner, Sokoloff, Weisser, Williams

March 11, 2019 5/14

Z-view of generated PDF

Example of z KDE histogram

Design



Note: All events from toy detector simulation

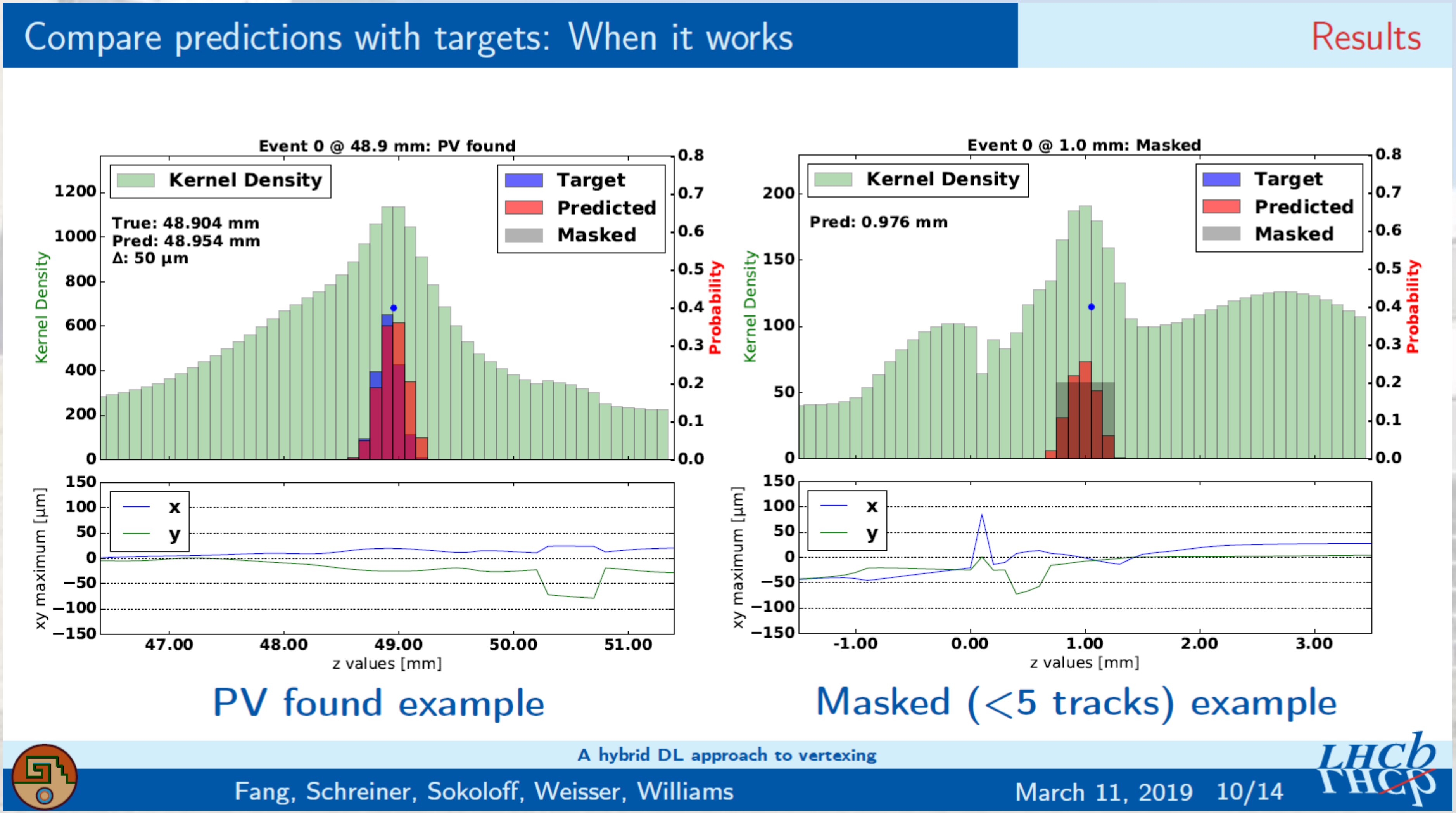
Human learning

- Peaks generally correspond to PVs and SVs

Challenges

- Vertex may be offset from peak
- Vertices interact

The network learns the vertices z



Compare predictions with targets: When it works

Results

PV found example

Masked (<5 tracks) example

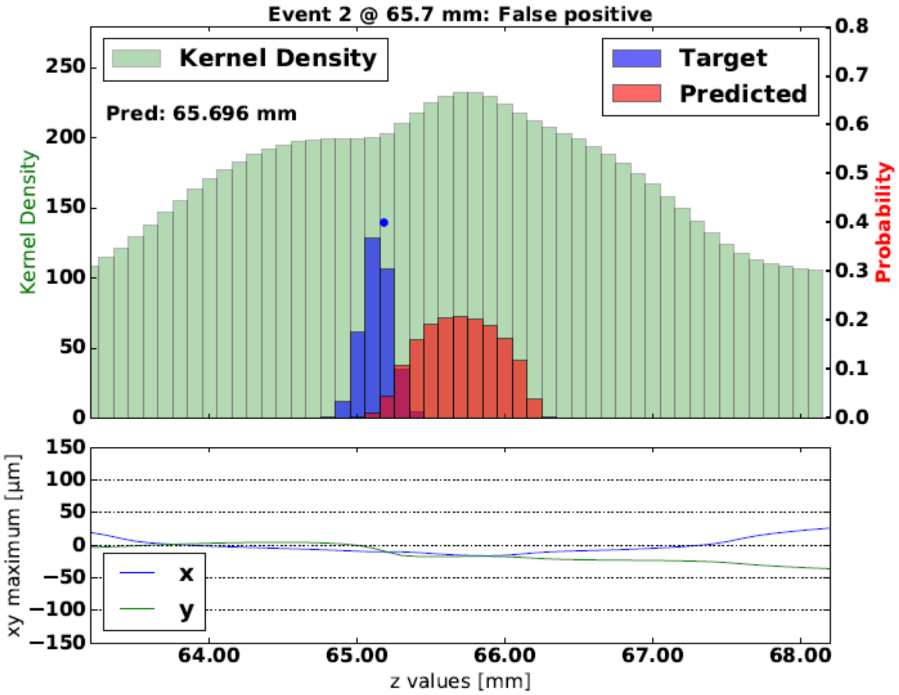
A hybrid DL approach to vertexing

Fang, Schreiner, Sokoloff, Weisser, Williams

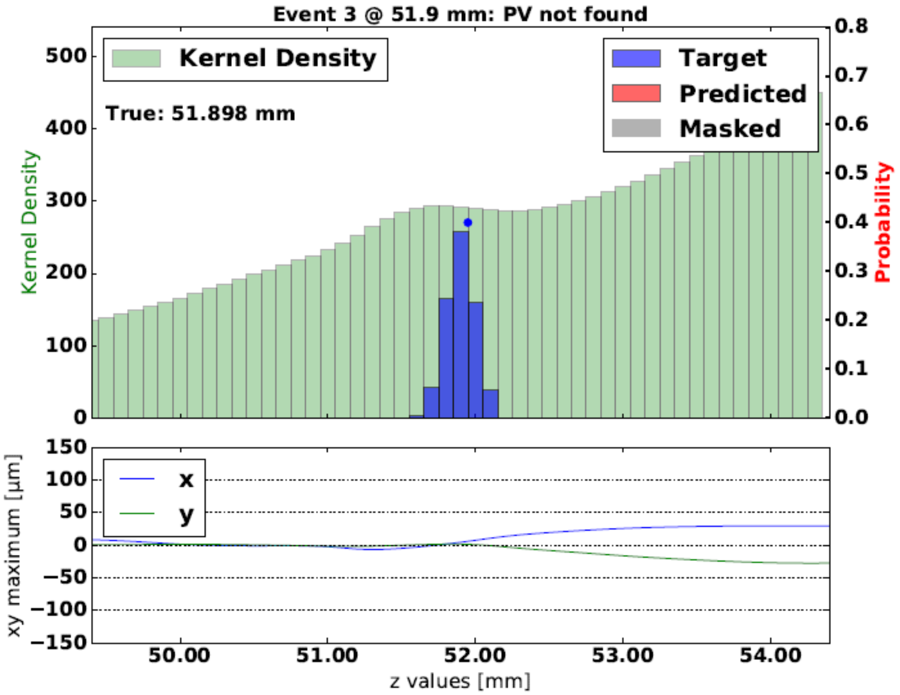
March 11, 2019 10/14

… Almost always

Compare predictions with targets: When it fails



False Positive example



PV not found example