Machine Learning
Steps Towards Training Happiness

C. Tunnell (Rice) G. Watts (University of Washington/Seattle)

Schedule

- Today
  - ML: Bigger than HEP
  - Intro of Machine Learning (this)
  - Quick guided demo of ML using the JAX framework
  - Tutorial: Signal and Background Separation in $H \to WW \to 2\ell 2\nu$
- Tomorrow
  - Survey of more complex ML techniques and network architecture
  - The Data Pipeline
  - Auto Encoders
  - Tutorial: Auto Encoder

Needle In the Haystack

Recall from Rafael's talk:

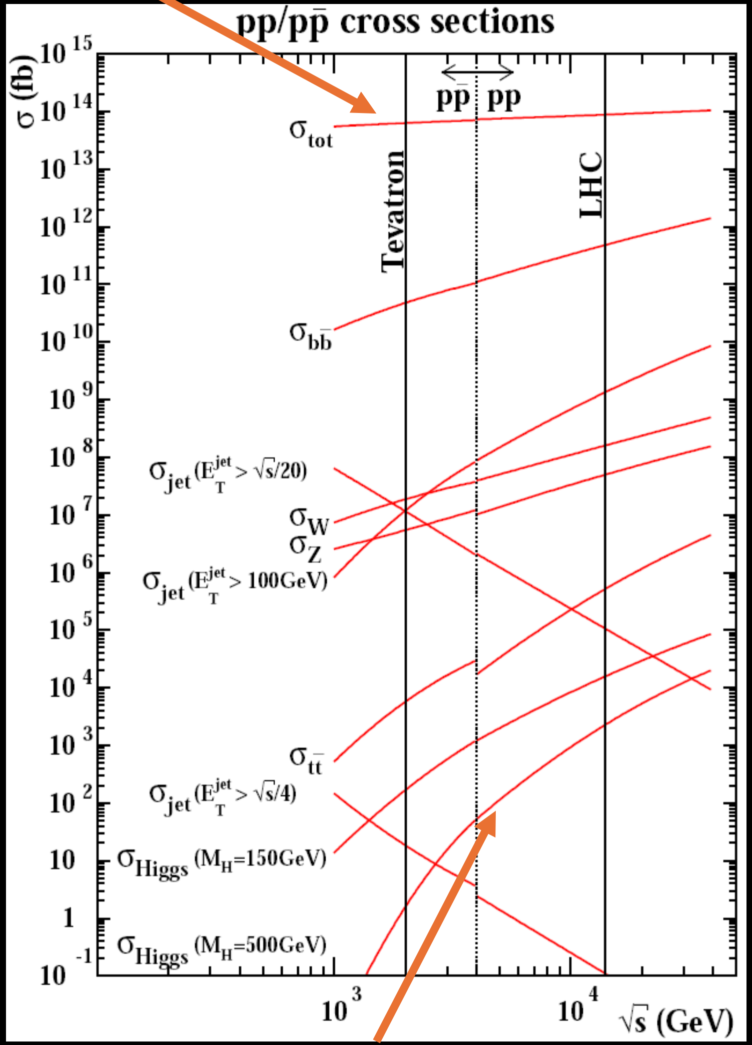
- LHC has collisions at a 40 Mhz Rate
- We save only 10,000 Hz
- A higgs is produced only 1 every second
- Branching ratios for each decay channel put that down to minutes!
- What we want is ~10-14 orders of magnitude below the $\sigma(pp)$!!!

HOW DO WE FIND THE SIGNAL?

- We use selection cuts (filtering that Jim will discuss)
- We use fitting
- We use Machine Learning

ML is now used almost everywhere!

What the LHC Gives us… (mostly)



Your paper's signal

Python won the scientific programming language race!

Why Python?

All the scientific frameworks for Deep Learning Machine Learning are written in Python:

- TensorFlow
  - developed by Google, managed as open source.
  - Not used as much internally but has one of the most active user communities.
  - API is most friendly to new users.
- PyTorch
  - Developed by Facebook, actively used.
  - Faster than TF
  - Is also a framework, but not quite as easy to use for a beginner.
- JAX
  - Developed in Google's DeepMind, used for most (all?) of their research
  - A library, not a framework
  - Great when you want to do something unique or break open the box.
  - Or just learn…

We are going to use JAX because it makes it easier to break things apart…

Personal Opinion

To graduate with an advanced degree in science you need to at least understand ML

What is ML?

1. We had some data
2. We had a function with some parameters
3. Using a procedure, the computer _taught_ itself the parameters

From Chris' talk this morning…

The ***joke*** is that Machine Learning is just a function fit, in the extreme!

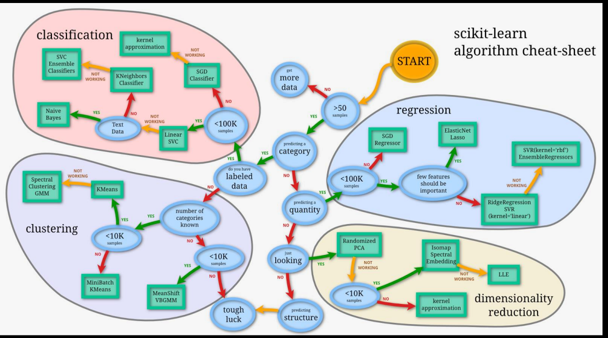
- With 10's of billions of data points.
- With complex functions with millions of parameters
- With a figure of merit that tells you when the function is making a good match to the data.

What Should You Use?

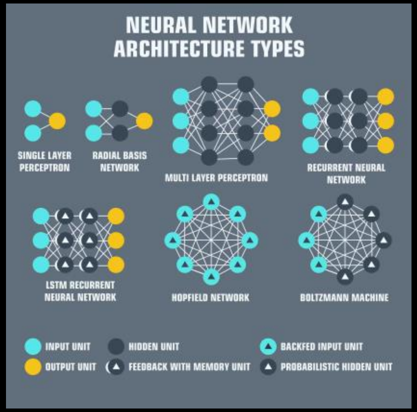
Neural Networks are the last thing you should use!

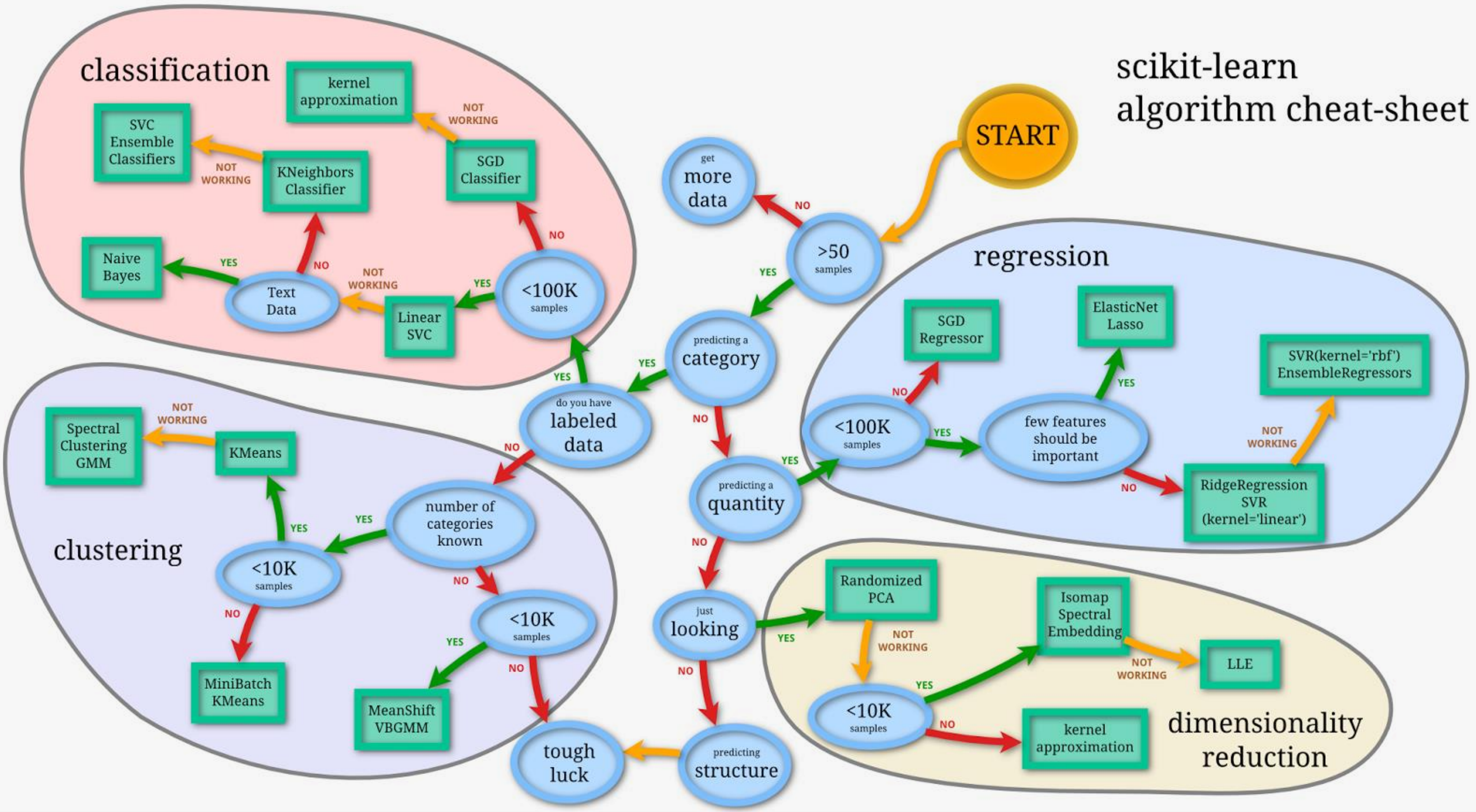
What Should You Use?

Non-Neural Network forms of Machine Learning:
- Great for limited number of inputs (features)
- Great for high level features (angles between jets)
- Great for small number of features (> 100).



Neural Network forms of Machine Learning:
- Great for large numbers of inputs
- Great for patterns in detector data
- Great for low-level data
- Great at Geometrical or Variable length inputs




scikit-learn
algorithm cheat-sheet
START
>50 samples
get more data
NO
YES
predicting a category
do you have labeled data
predicting a quantity
just looking
predicting structure
tough luck
classification
<100K samples
SGD Classifier
kernel approximation
NOT WORKING
Linear SVC
Text Data
Naive Bayes
KNeighbors Classifier
SVC Ensemble Classifiers
regression
SGD Regressor
few features should be important
ElasticNet Lasso
RidgeRegression SVR (kernel='linear')
SVR(kernel='rbf') EnsembleRegressors
clustering
number of categories known
<10K samples
KMeans
Spectral Clustering GMM
MiniBatch KMeans
MeanShift VBGMM
dimensionality reduction
Randomized PCA
Isomap Spectral Embedding
LLE
kernel approximation

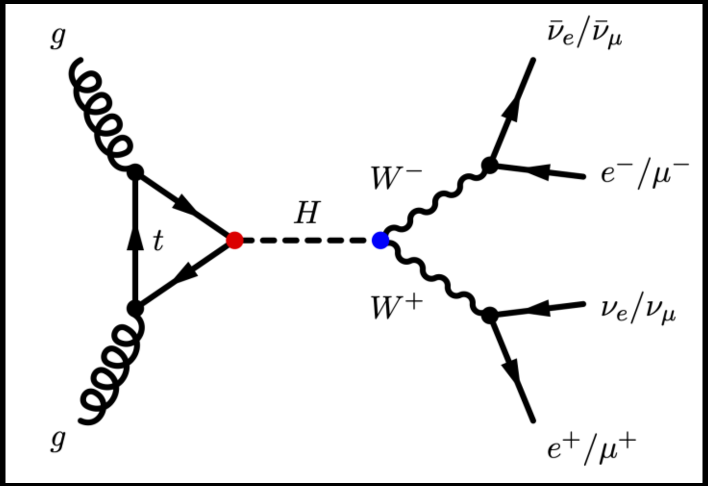
But we aren’t going to talk about that…

We are going to start first looking at Neural Networks

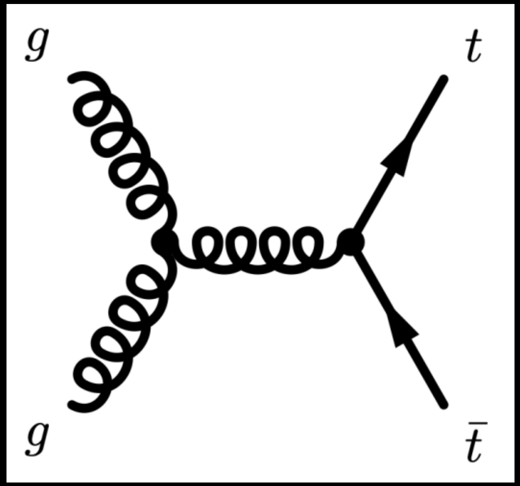
Here is a tutorial on using Boosted Decision Trees
(from HSF)

The Training Data

Signal



Background



Features:

- # of leptons
- p_T of leptons
- # of jets
- Missing Energy
- Etc.

We have this information for both signal events and background events.
We one “row” per event
From Simulation
And we know which is which!

Inference

$$y_i = f(x_i)$$

A prediction for the ith row
0 – Background
1- Signal

Our NN
(a big hairy function)

The $i$the row of our feature data

How do we determine the function?

$$y_i = f(x_i)$$

$$f(x_i; \Theta)$$

- Some function with parameters
- Straight line: $ax + b$
- Much more complex with millions of parameters!
- All the parameters are usually denoted Θ

How do we determine the functional form?

It is a bit of a dark art…

How do we determine the parameters?

How do we fit a straight line?

1 Define a mathematical criteria for a good fit:

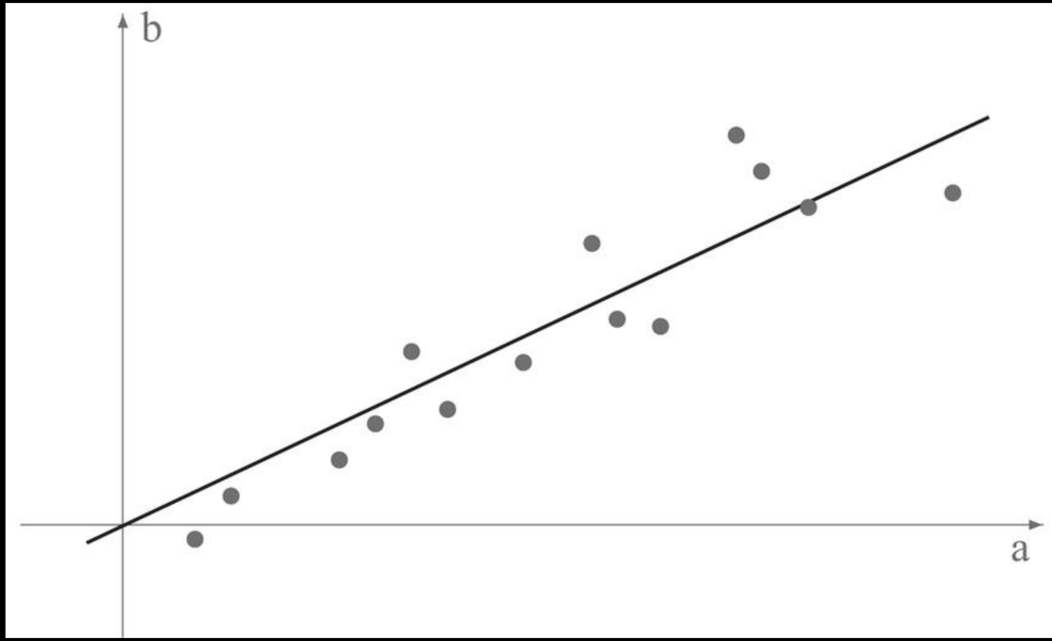
$$r = \sum (x_i - y_i)^2$$

(the ***Loss*** function)

2 Minimize r (find Θ such that):

$$\frac{dr}{d\Theta} = \frac{d}{d\Theta}\left(\sum (x_i - y_i)^2\right) = 0$$

This is why we use TF, PyTorch, JAX, etc.



What Loss Function To Choose?

What do you want to optimize?

- Signal and background separation
- Measured mass
- Decorrelation of two outputs that separate signal and background
- **Etc.!!!**

What must the loss function do?

- Return a value – "figure of merit"
- Some "distance" between perfect fit and the current function
- Differentiable!!

There is a lot of room for creativity!

But there are some standard choices…

What Loss Function To Choose?

Mean Squared Loss

$$r = \sum (x_i - y_i)^2$$

Works very well for **regression** problems!

Cross Entropy Loss

$$r = \frac{1}{N} \sum (y_i \log(p_i) + (1 - y_i) \log(1 - p_i))$$

Works very well for **classification** problems!

Problem Types

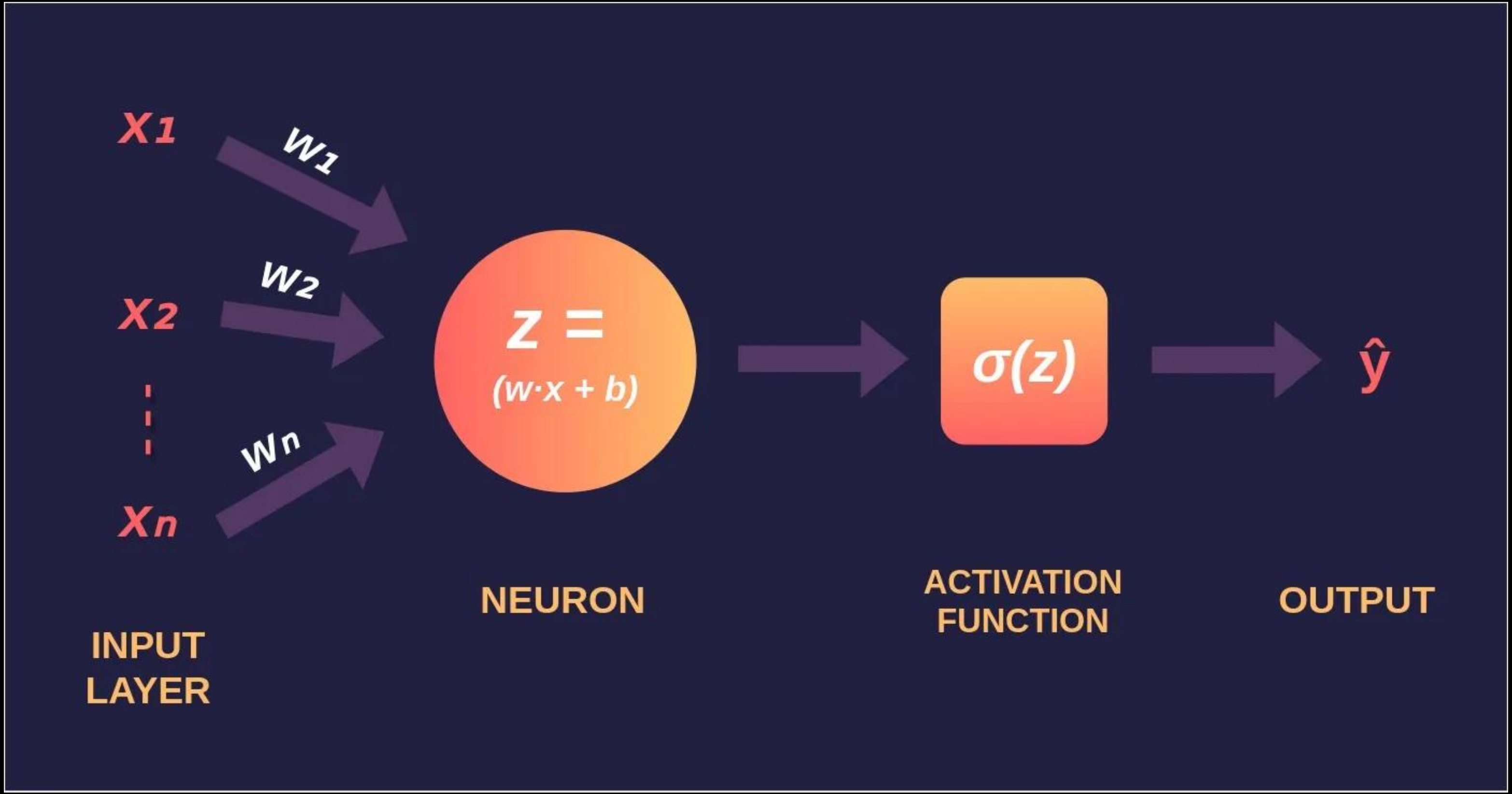
Regression

- Continuous output
- Jet Energy Calibration
- Mass of the Higgs

Classification

- 1 or 0 type output
- Is it signal or background?
- Is it signal, QCD background, or Beam Induced Background

How do we come up with the Function?



What is a NN?

"A single Neuron"

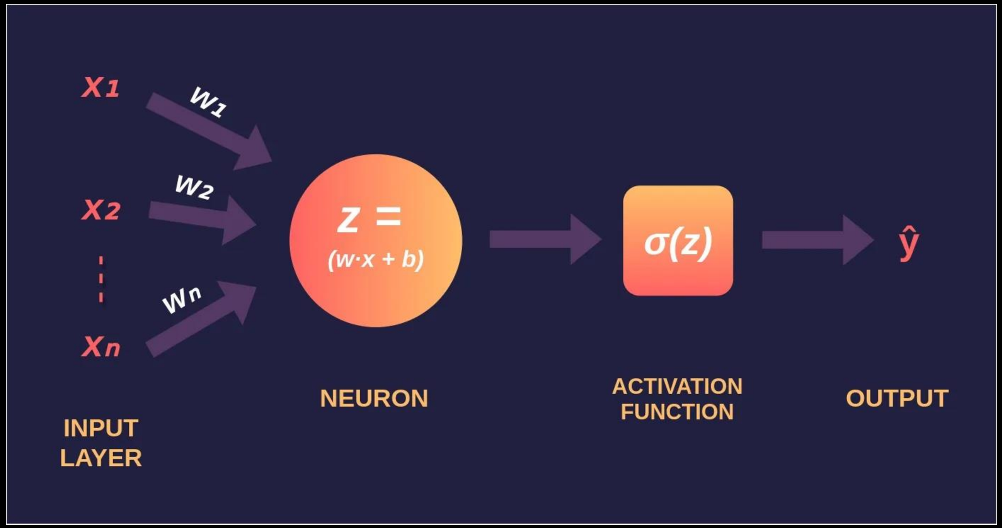
$$\hat{y} = \sigma(W \cdot x + b)$$

Activation Function

Weights

Inputs

Offsets



Everything is a matrix multiplication

Why GPU's are so well suited to this!

Activation Functions

Form	Name	Pros	Cons
$\sigma(z) = \dfrac{1}{1 + e^{-z}}$	Softmax	Differentiable, finite range	Vanishing derivative at $\pm\infty$
$\sigma(z) = \max(0, z)$	Rectified Linear Unit (ReLU)	Computationally Efficient, does not saturate on one side	Dead Neuron, unbounded on positive side
$\sigma(z) = \ln(1 + e^{z})$	Softplus	Vanishing derivative is better handled	Computationally expensive

Most popular: **Softmax**! – especially as last layer
But a combination is frequently used

1. Use ReLU inside network
2. Use Softmax to control outputs for classification

Can we fit any data shape?

The Universal Approximation Theorem

Theorem 3. *Let σ be a continuous sigmoidal function. Let f be the decision function for any finite measurable partition of I_n. For any $\varepsilon > 0$, there is a finite sum of the form*

$$G(x) = \sum_{j=1}^{N} \alpha_j \sigma(y_j^{\mathrm{T}} x + \theta_j)$$

and a set $D \subset I_n$, so that $m(D) \geq 1 - \varepsilon$ and

$$|G(x) - f(x)| < \varepsilon \qquad \text{for} \quad x \in D.$$

Why we need the non-linearity of activation functions
Why we need to build a network out of many neurons!

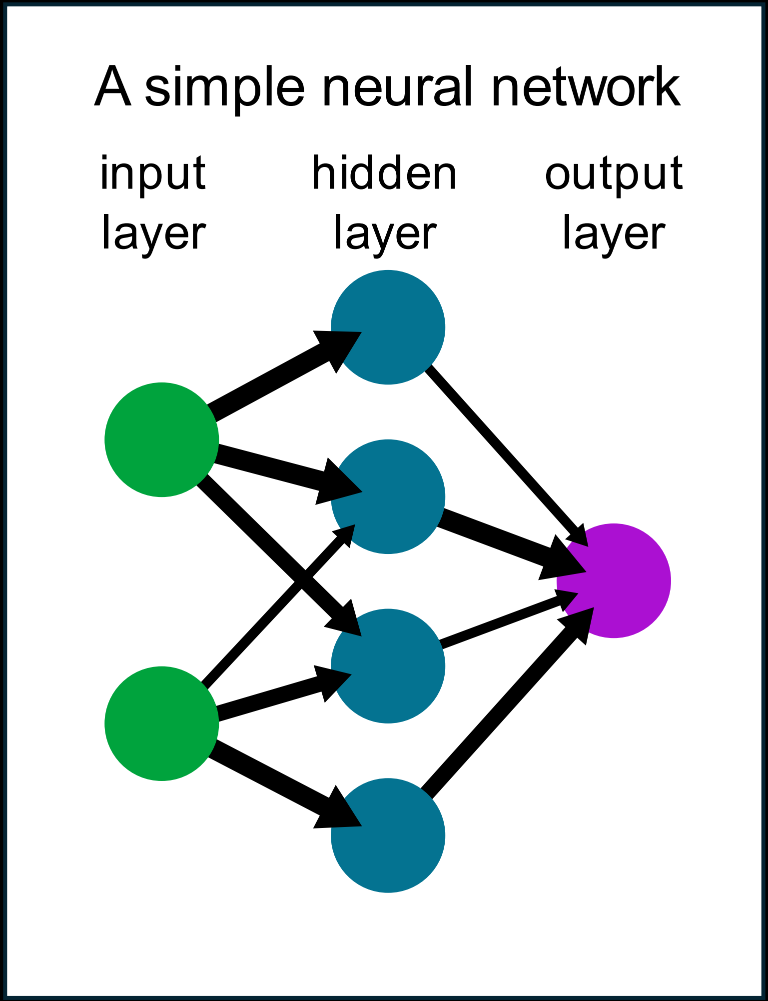
A Real NN

1. Put multiple neurons in a “layer”
2. Chain together multiple layers

Fully Connected Network

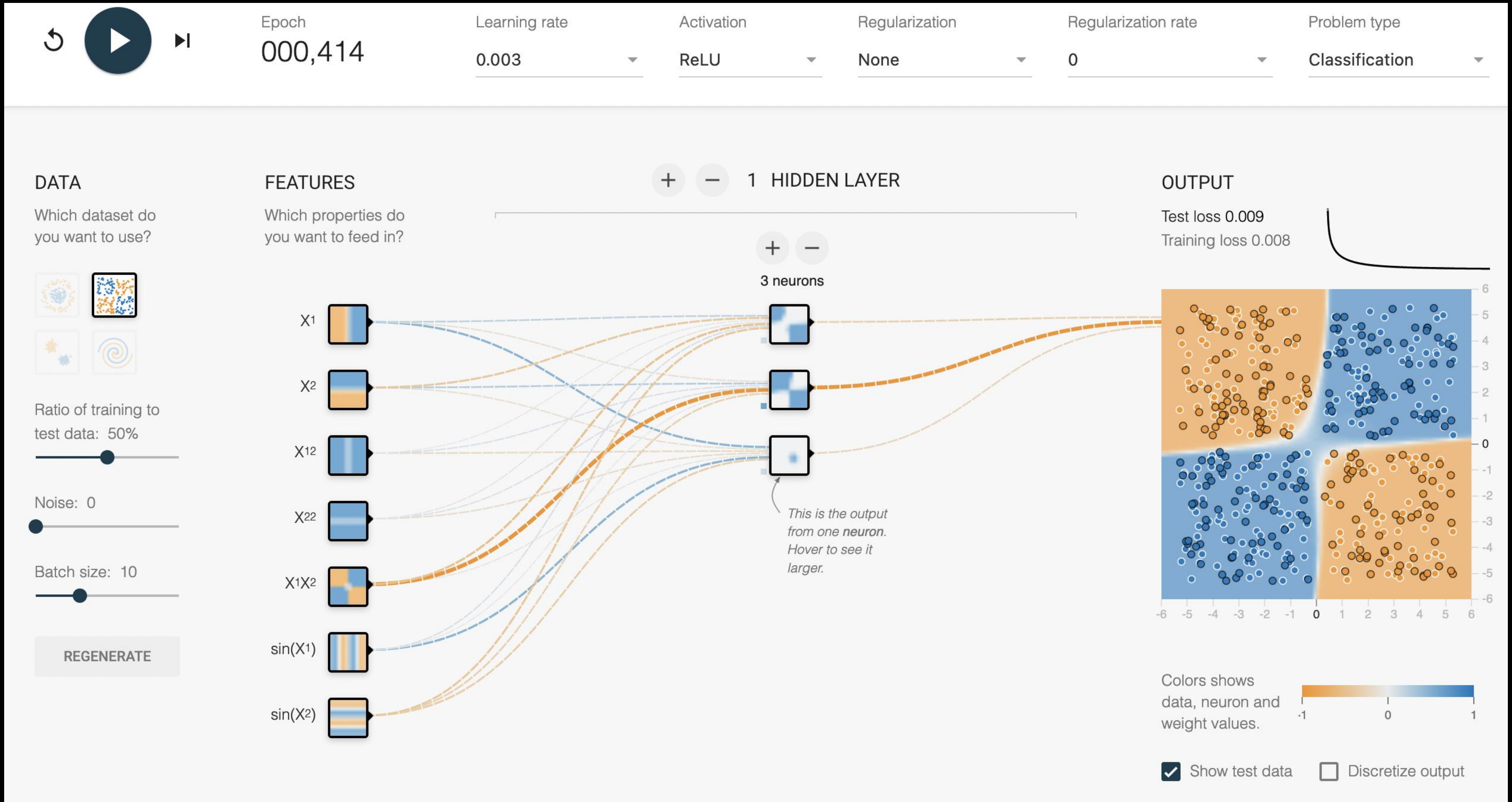
This is the simplest place to start – and perhaps best place to start

- More sophisticated architectures are out there
- Frequently help with the training speed
- Architect data flow
- Internal loops inside the functions, etc.

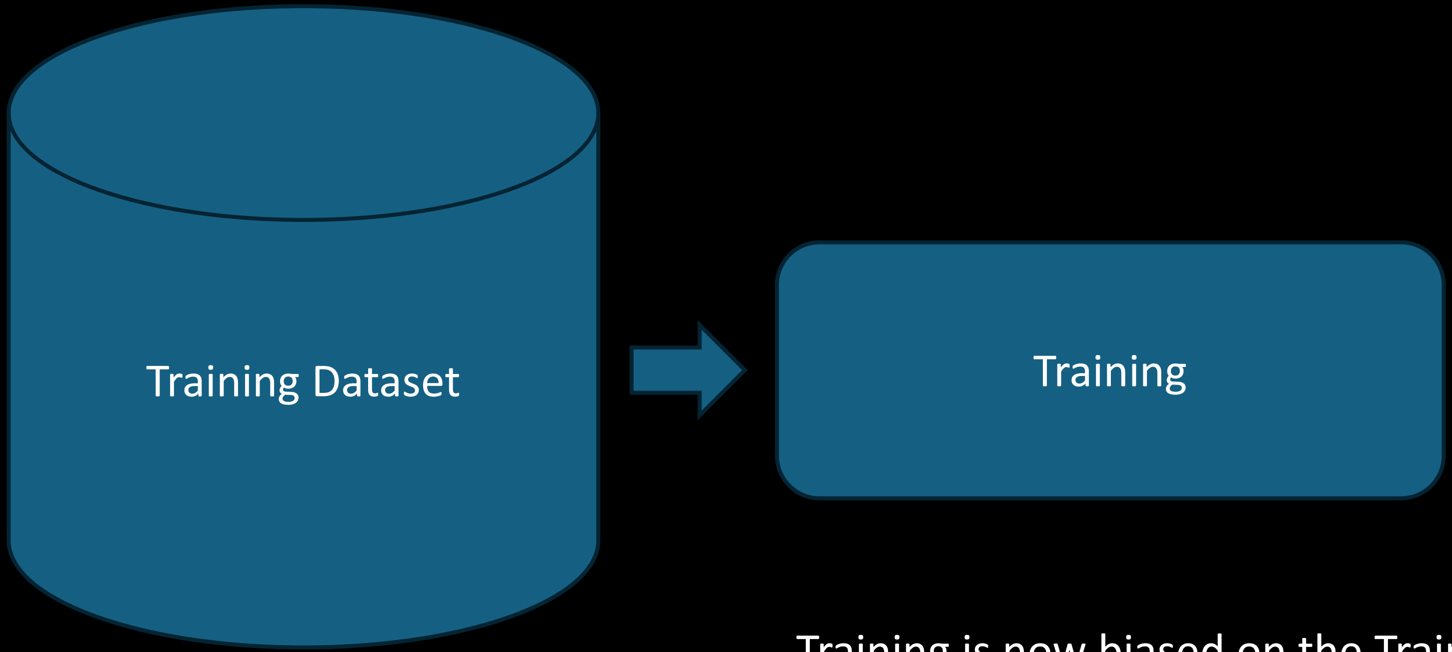


Example Of Learning

Link to TF Playground



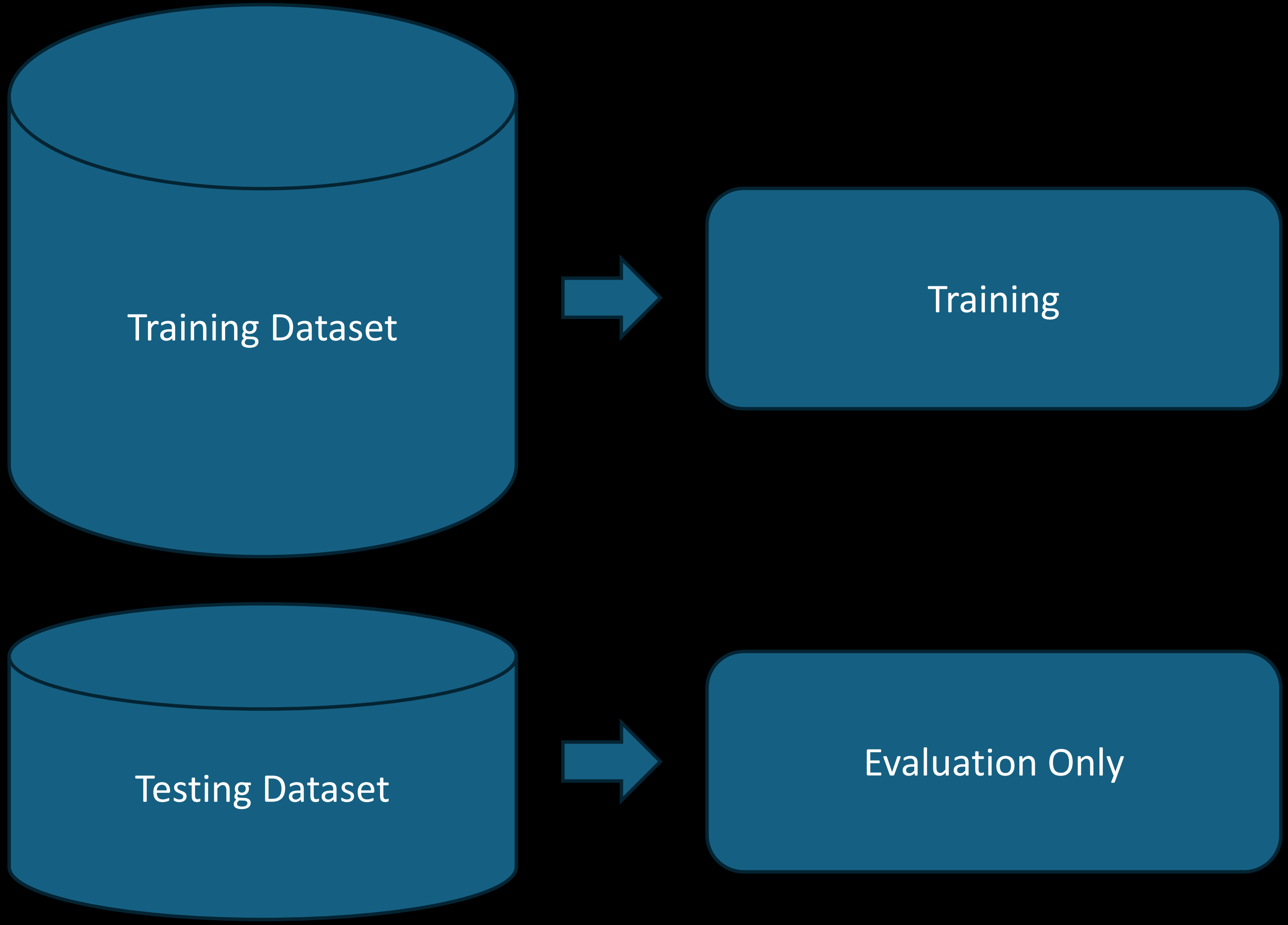
Test & Training & Validation & Overfitting



Training is now biased on the Training Dataset

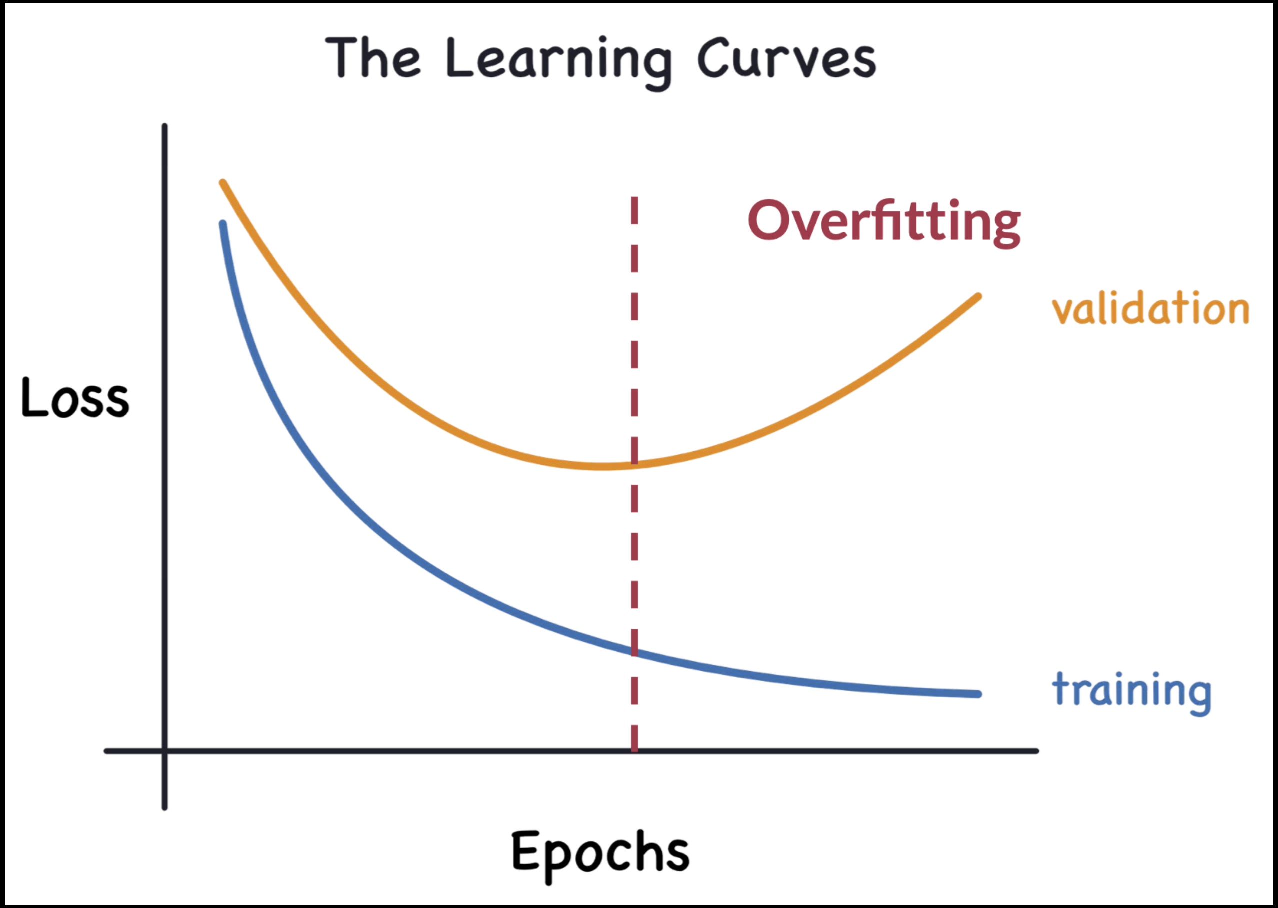
- What if the Dataset isn’t fully representative?
- It learns specific features of the training dataset…

Test & Training & Validation & Overfitting



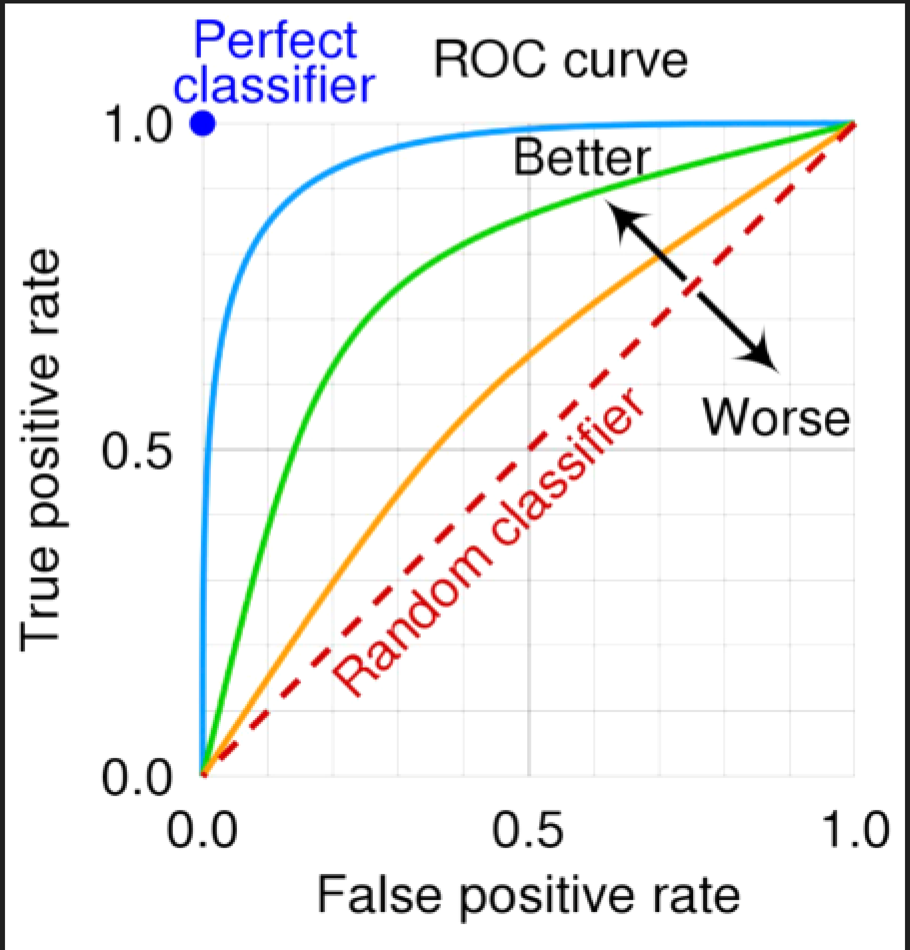
Testing dataset provides that independent testing…

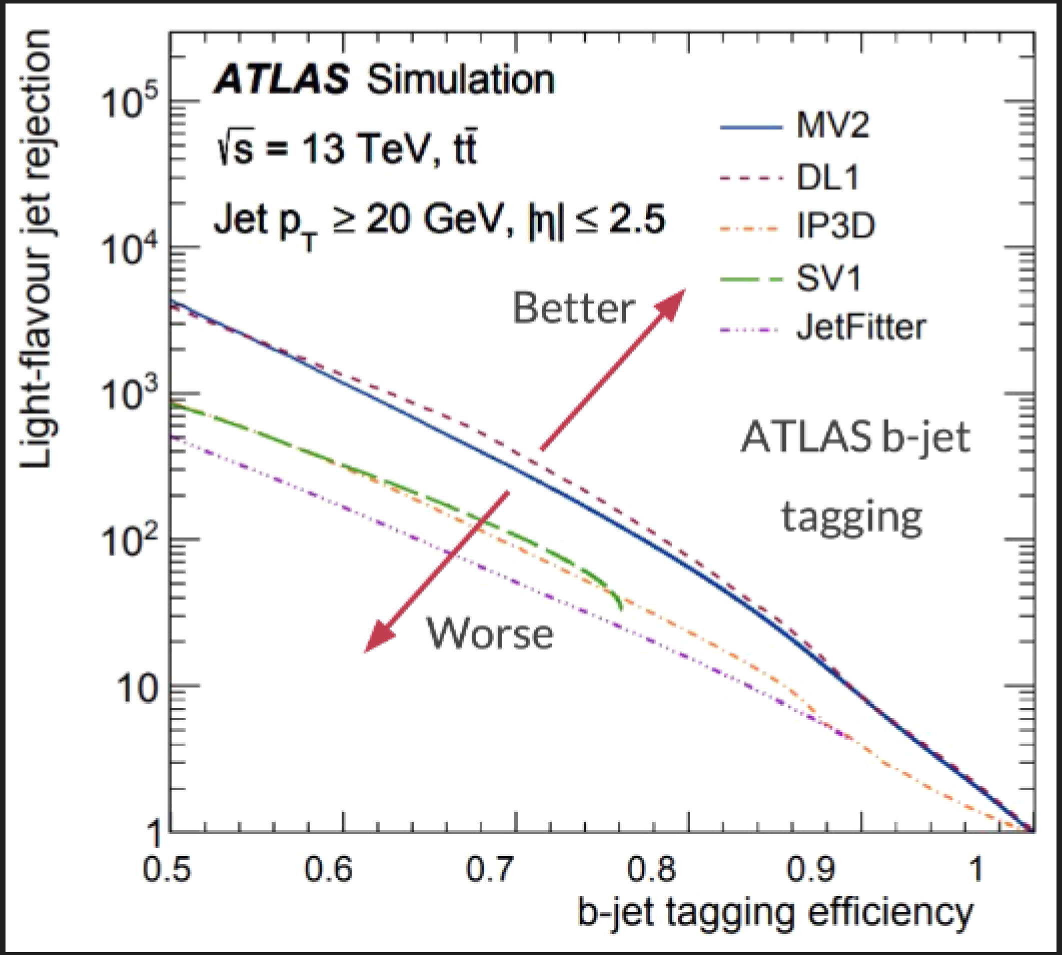
Test & Training & Validation & Overfitting



ROC Curve (Receiver Operating Characteristic)

- Plot of true positive rate/signal efficiency against false positive rate/background efficiency
- In HEP, often use $1/\epsilon_B$ against $\epsilon_S \rightarrow$ for 70% signal efficiency, 1000 bkg. rej





Gordon's 5-Stages of ML-Grief

1. Understand your data
    - ML will find all possible discrepancies. Even ones you can't see.
    - Plot everything you are feeding the network! Really ask - does it make sense?
2. Understand what your ML algorithm is doing
    - Don't trust it further than you can throw a … well… anything heavy.
    - Over training isn't usually what will get you
    - Picking up on non-physical differences in the training samples… will.
    - If the results is too good to be true…
3. Automate Everything
    - The process is very iterative and you'll constantly be going back to previous steps
    - Automation slows you down initially, but is a speed multiplier later on!
4. Use a search-engine/AI/GPT to help you
    - Almost everything basic has been done before. Copy!
    - Even advanced things have been done.
    - Use google, stackoverflow, ChatGPT - whatever you have access to.
    - While it is useful to learn the basics, in the end, you want to do science, not ML research
        - unless you do…
5. Also, Understand your data

Go Forth and ML!