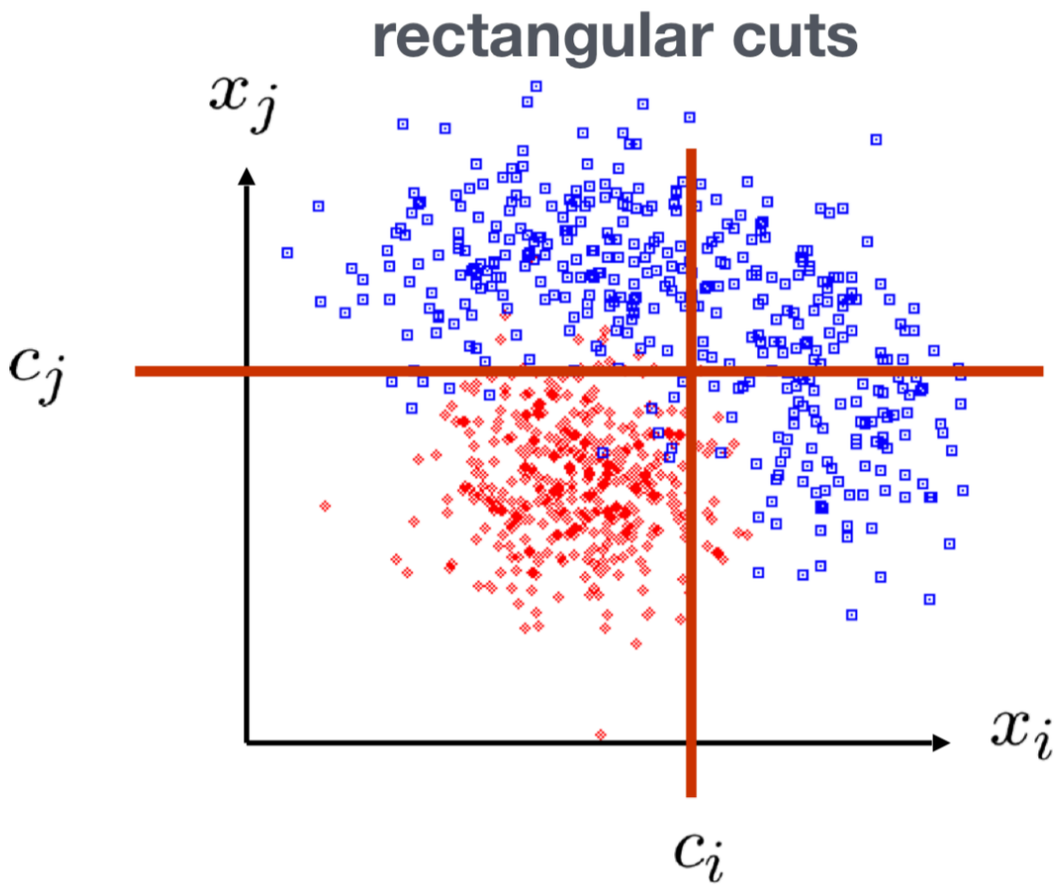
Machine Learning exercise

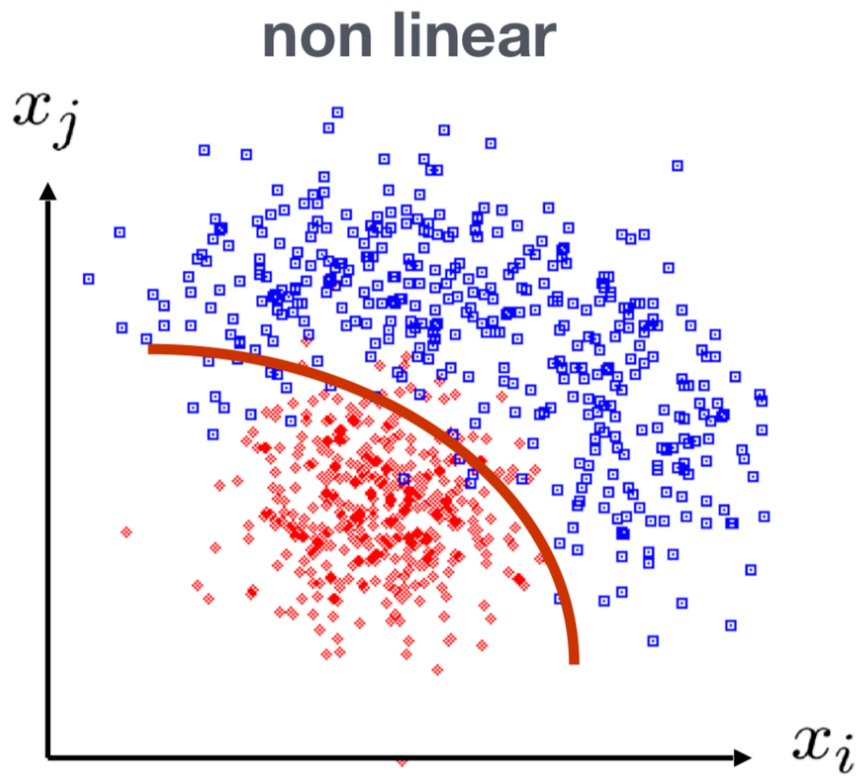
Lukas Layer

Idea of the exercise

- Focus on practical application - as little math as possible
- Use some of the most popular industrial libraries - more powerful (and more fun) than ROOT TMVA
- Learn how to convert ROOT Trees to pandas frames - apply machine learning - convert back to ROOT
- Next time: hyper-parameter optimization, cross-fold validation, multi-class classification

Why ML for S/B classification ?

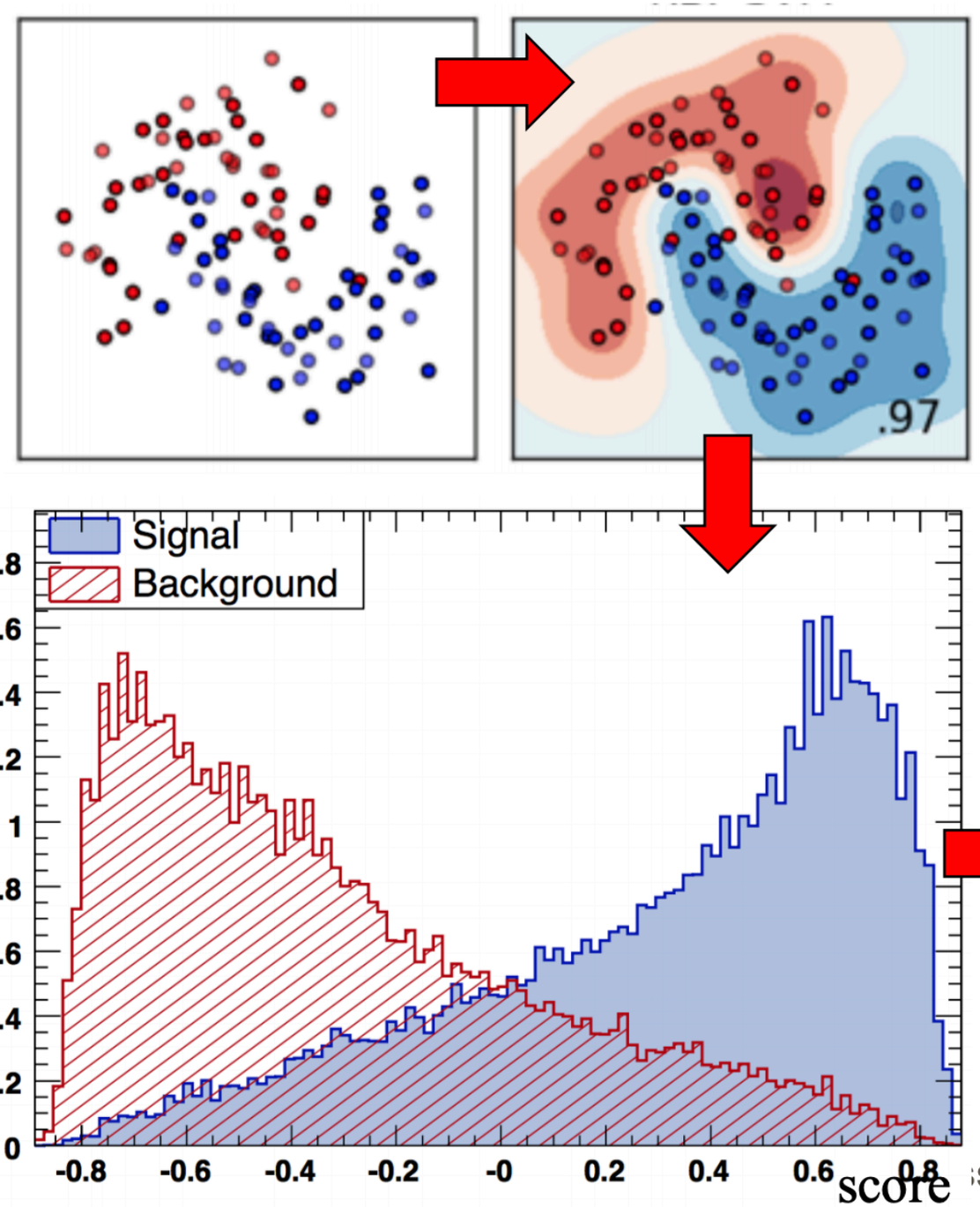




—> Better separation between signal and background

—> Potentially improves sensitivity of your analysis

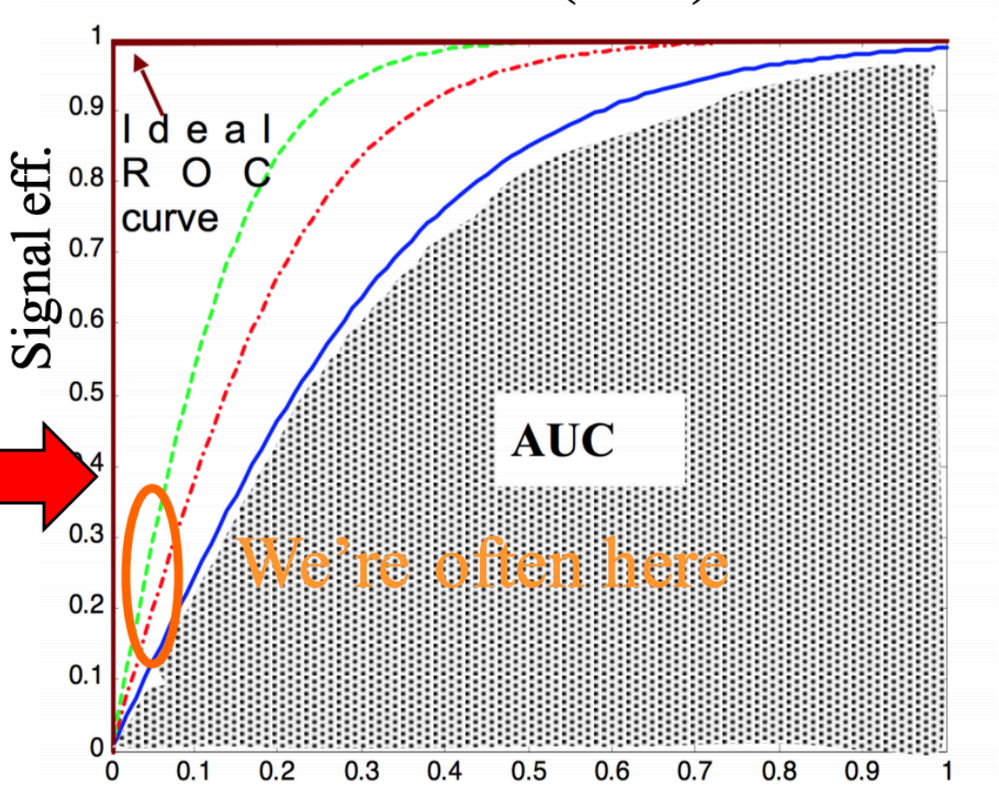
Signal/Background classification



Train on Signal and Background Monte-Carlo
➔learn the separation between S and B distribution
Apply on test sample
Apply on data

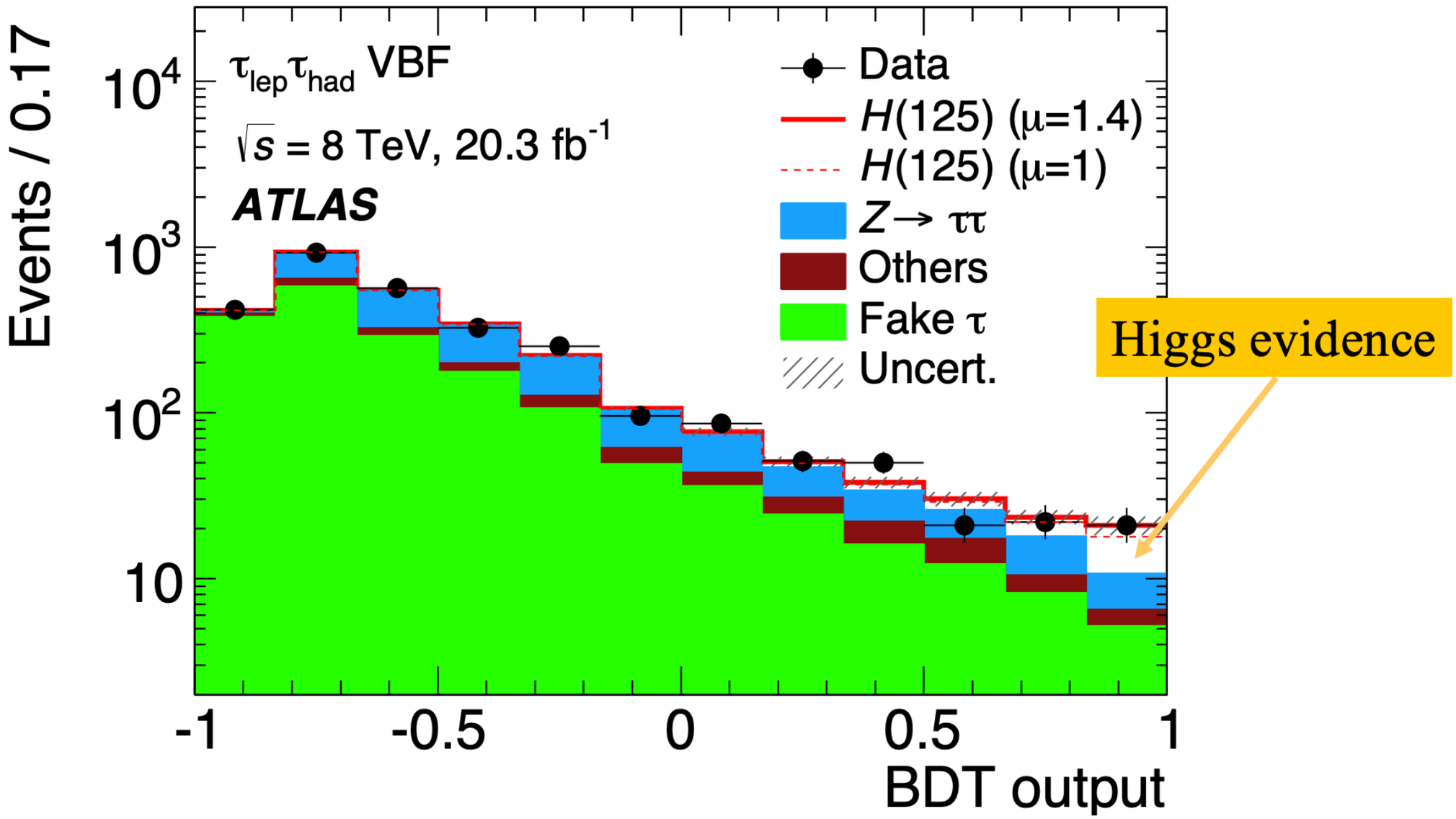
Note: instead of *classifying* 0 or 1, can *regress* !

AUC : Area Under the (ROC) Curve

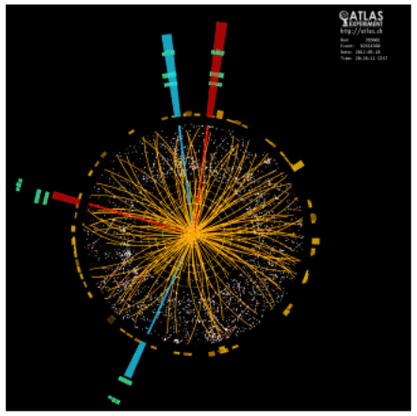


HEP analysis example

Boosted Decision Tree (BDT) using ~dozen of high level variables



SUPERVISED LEARNING



h(**x**; **w**)
Function with adjustable parameters

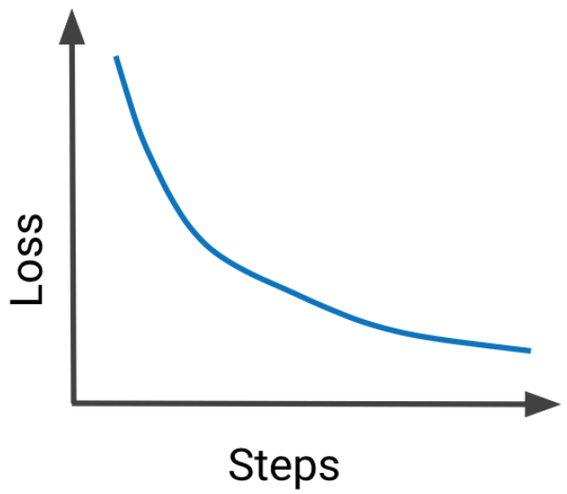
Loss Function

Compare prediction with true label

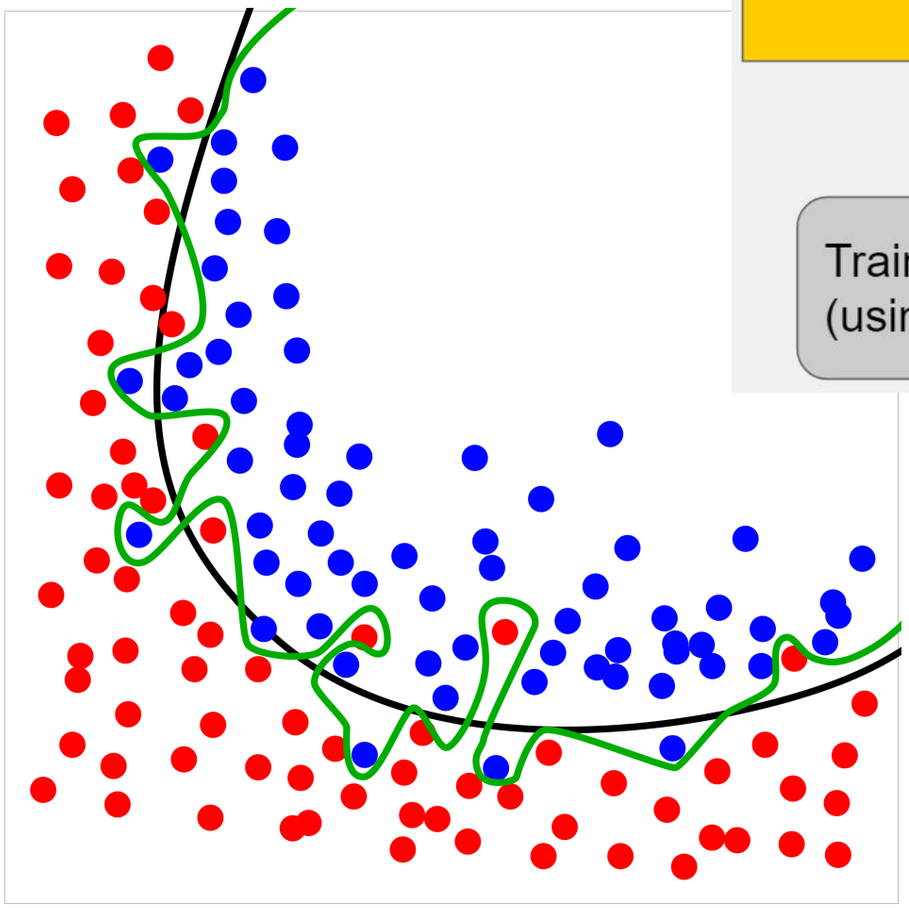
Loss

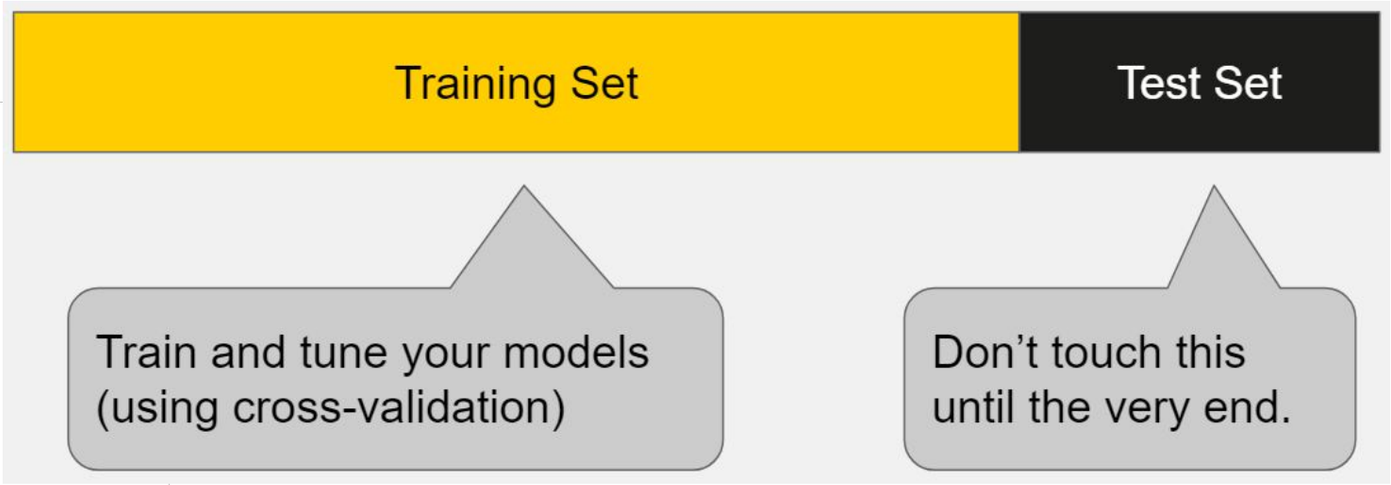
True labels:
Higgs = 1
Bkg = 0

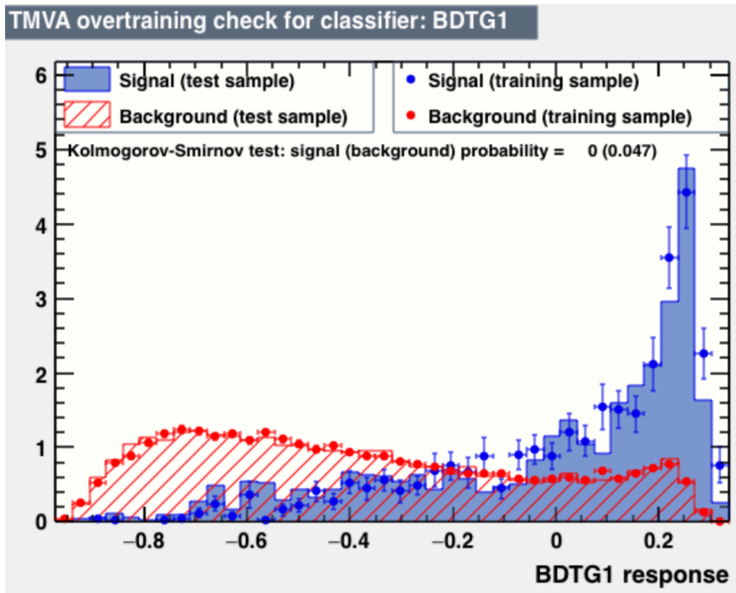
- Loss function: evaluate how well your algorithm models your dataset
- If predictions are bad: loss function will output a higher number.
- If predictions are good: lower number.
- Loss tells you if your model improves during training
- e.g. Mean Squared Error: $L(\theta) = \sum_i (y_i - \hat{y}_i)^2$

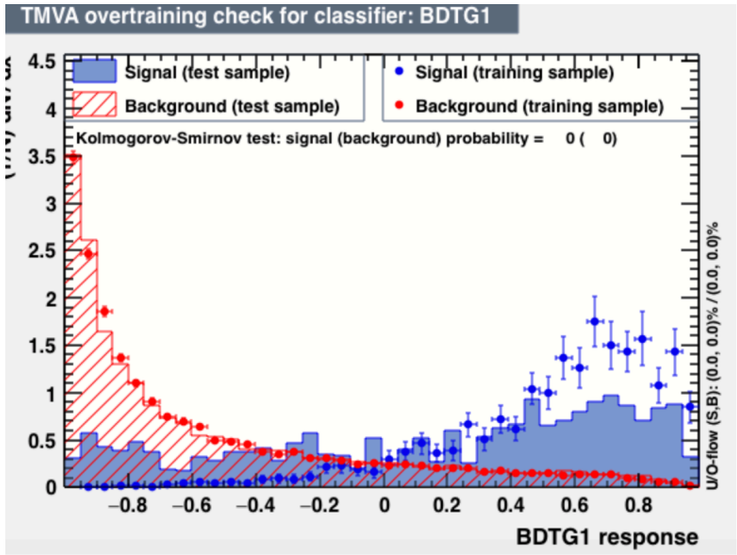


Overfitting and how to prevent it



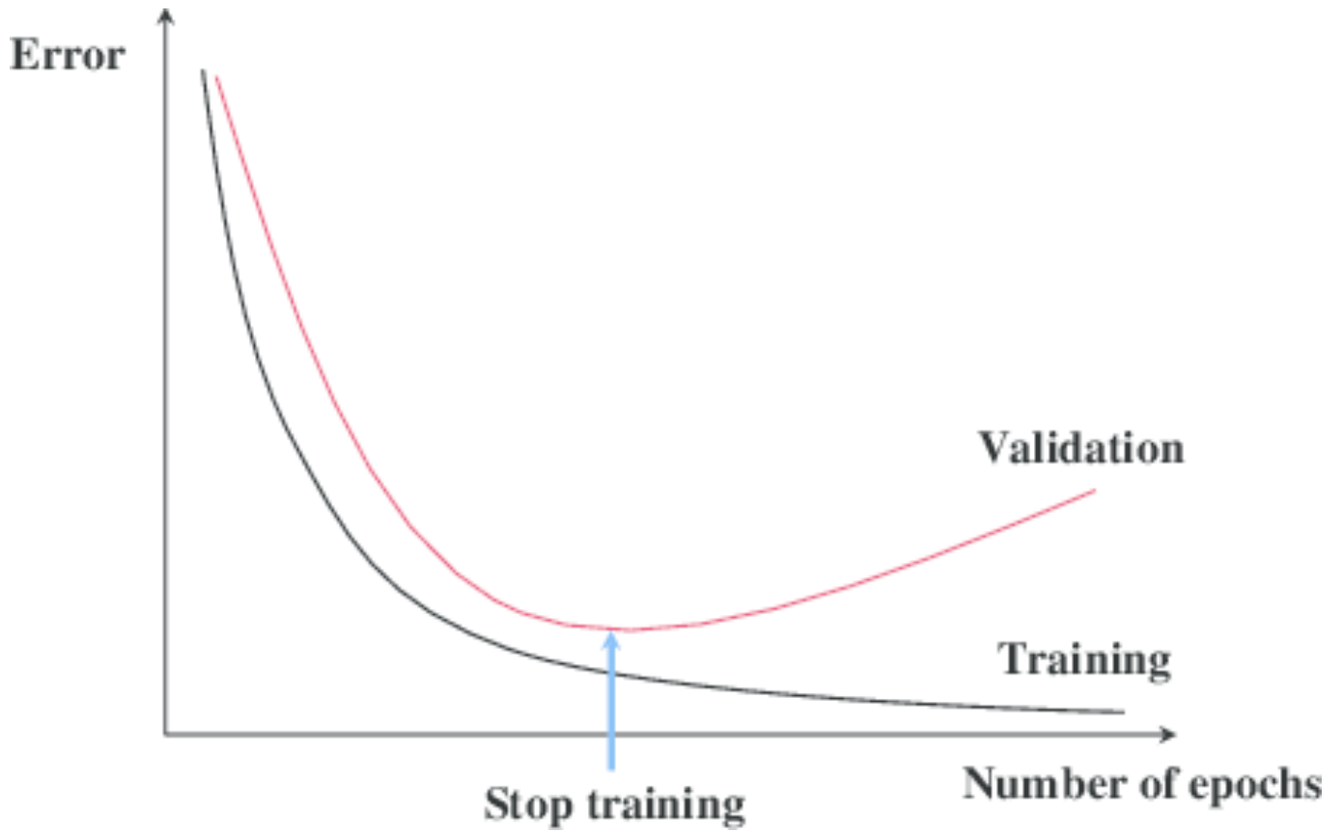






- Split the data into train and test samples
- Check the score on train and test sample

Early stopping



- Easy way to prevent overfitting
- Monitor the loss on a validation set and stop when it gets worse

Decision Tree

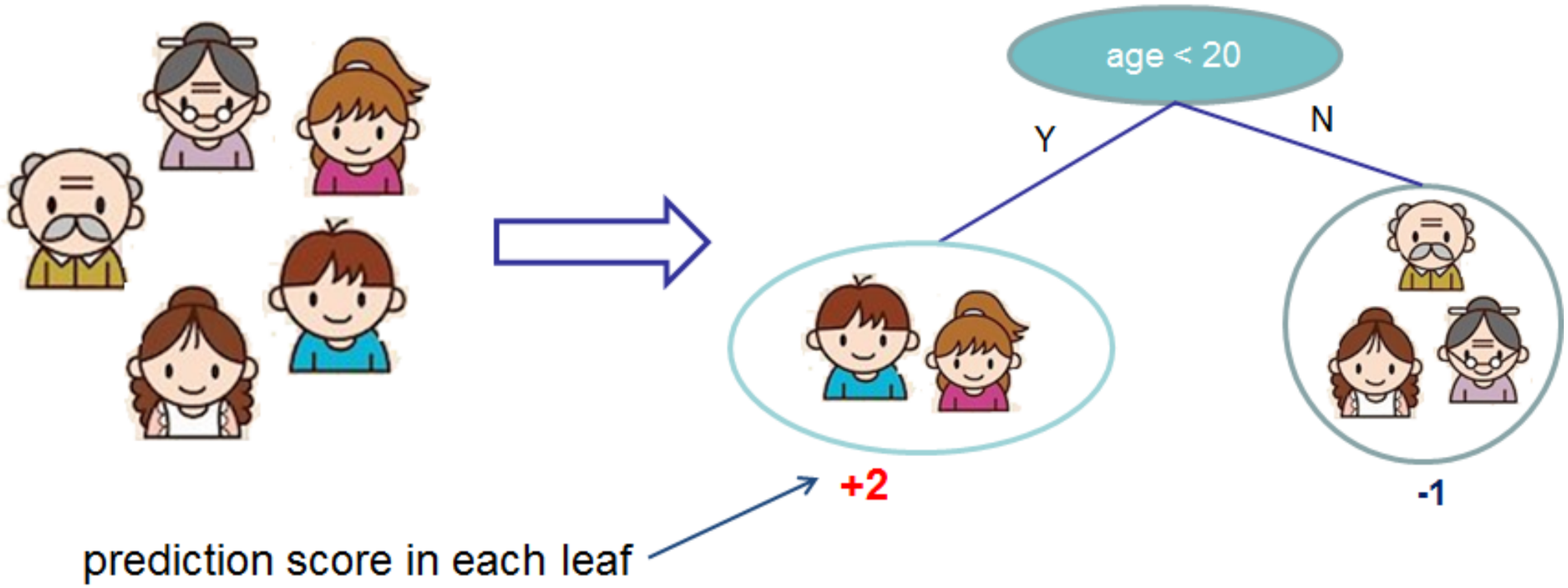
Objective function: training loss (predictive power) and regularization term (complexity):

$$\text{obj}(\theta) = L(\theta) + \Omega(\theta)$$

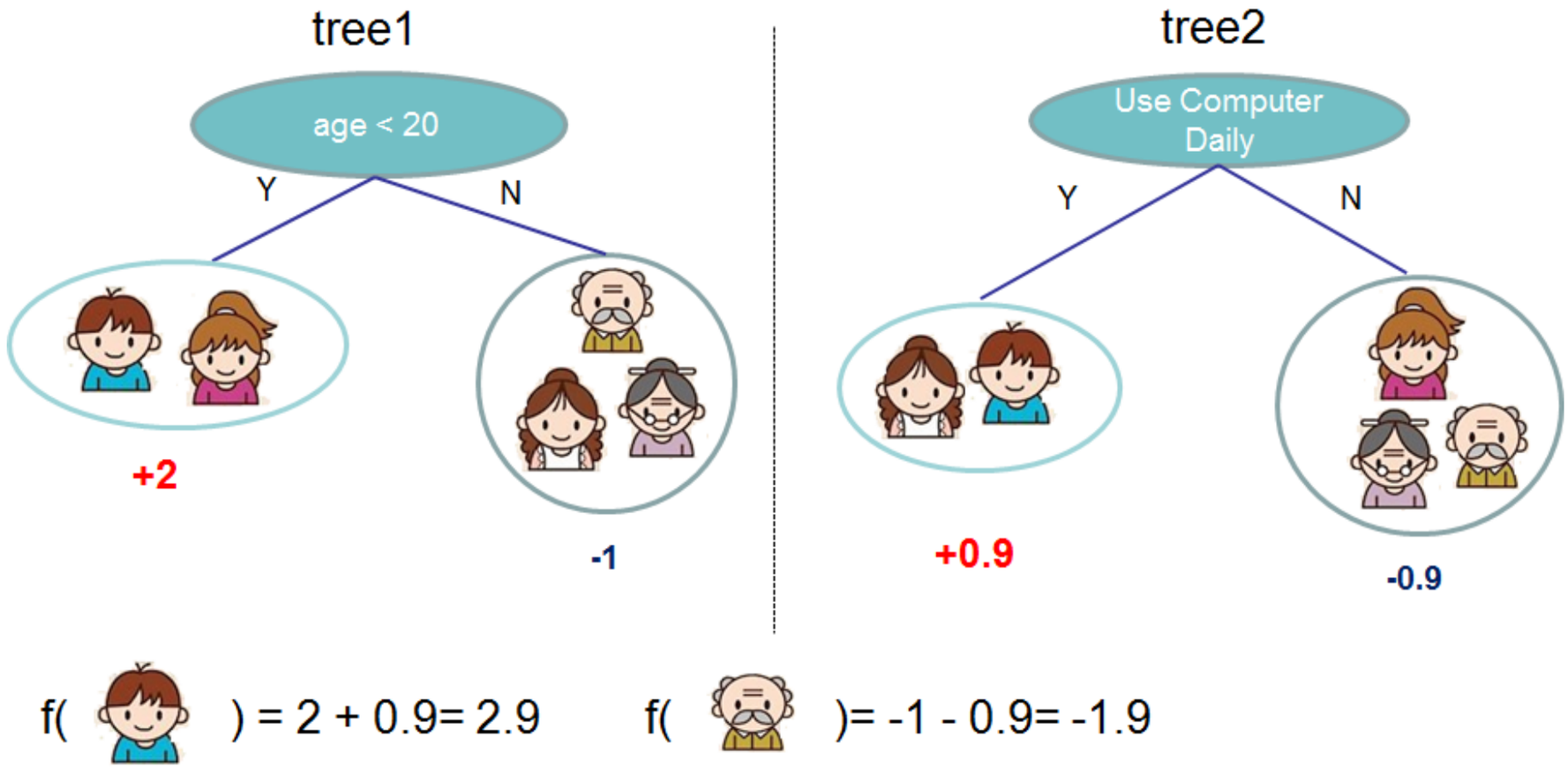
e.g. MSE: $$L(\theta) = \sum_i (y_i - \hat{y}_i)^2$$

Input: age, gender, occupation, …

Like the computer game X



Boosted Decision Trees (BDT)



- Boosted trees - less prone to overfitting
- BDTs are widely used in HEP
- Simpler than NNs + good out of the box results + feature importance

- Powerful, fast industrial library

- Good out of the box results

Important parameters to control complexity/overfitting and some of my experiences:

- Number of trees [10 - 1000] higher values -> more complex

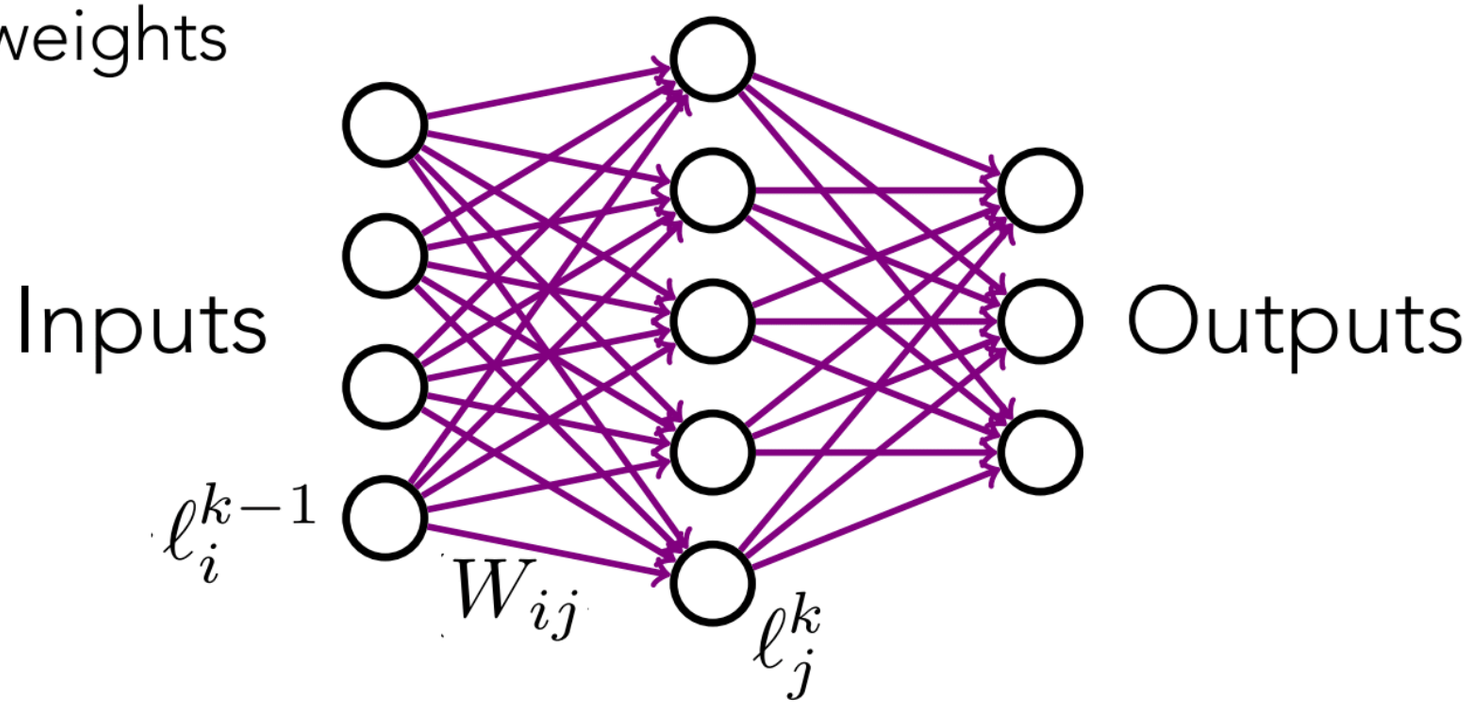
- Learning rate [10e-6 - 0.3] higher values -> faster but can overfit

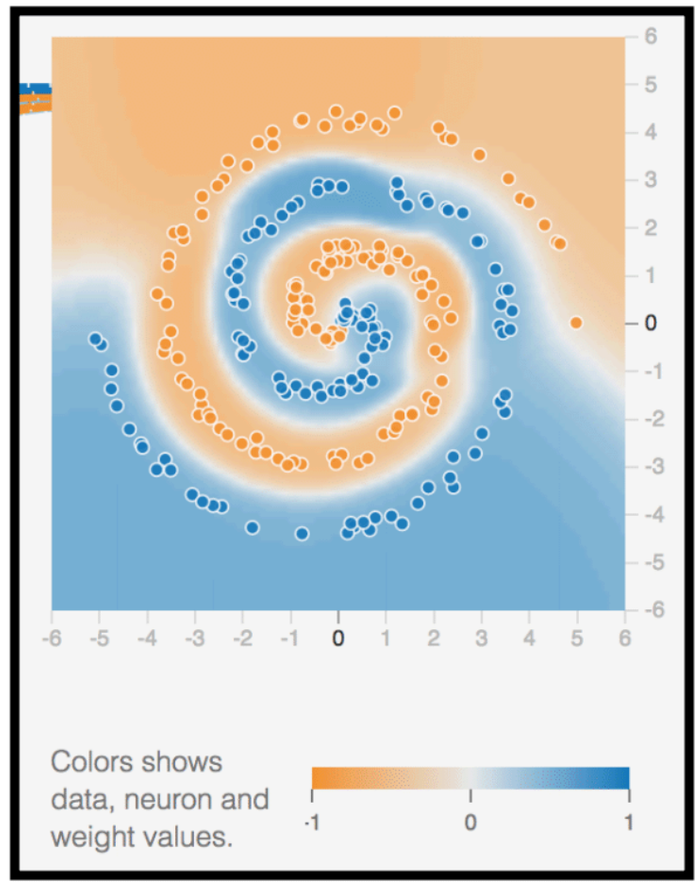
- Tree depth [2 - 15] higher values -> more complex

- min_child_weight [1 - 10] higher values -> reduces complexity

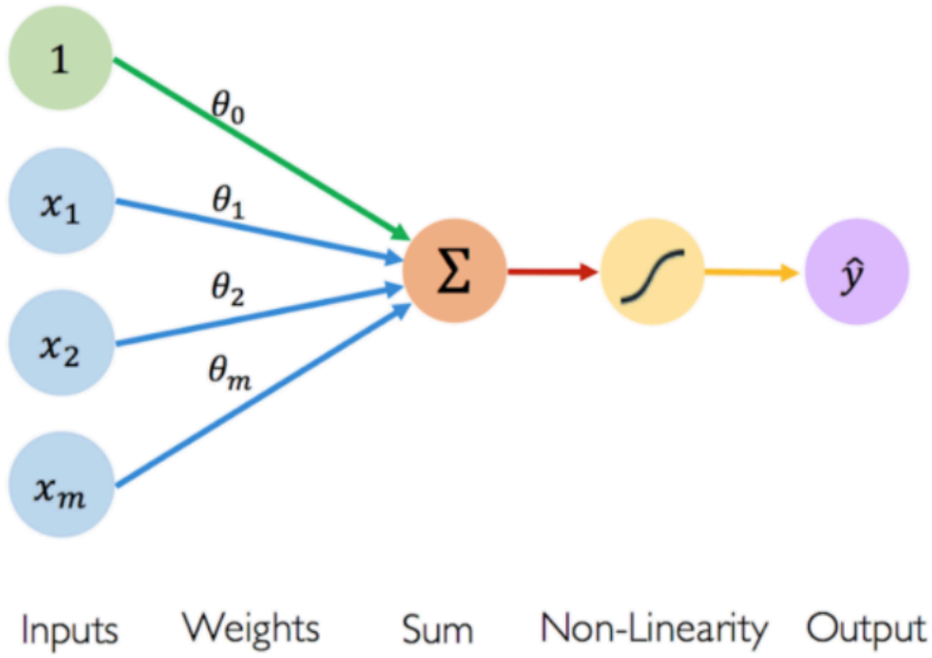
Neural Networks

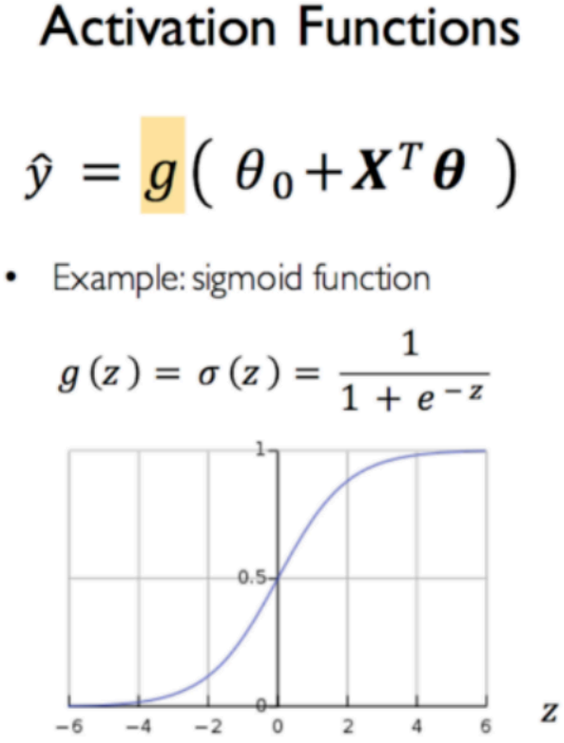
- **Multiple layers**: output of previous layer is ***fed forward*** to next layer after applying ***non-linear*** activation function $\ell_j^k = \phi(W_{ij}\ell_i^{k-1} + b_j)$
- **Fully connected**: many independent weights
- **Learning**: Use analytic derivatives and stochastic gradient descent to find optimal weights





More important terms…





- Activation function: sigmoid, tanh, relu
- Epochs: number of times the data is exposed to the NN
- Batch size: number of samples until weights are updated [10 - 1000]
- Optimizer: algorithm to optimize the loss: SGD, Adam…

- Simple but very powerful deep-learning library

- Tensorflow based - most popular choice for many DL applications

Important parameters for a simple feed-forward NN:

- Learning rate (important!) [10e-6 - 0.1]

- Hidden layers [2-8] higher values -> more complex

- Units layers [10-100] higher values -> more complex

- Dropout [0 - 0.9] higher values -> reduces overfitting

ML in HEP

Difference to data science: Data/MC agreement is crucial

- Simulations imperfect —> although ML performance on MC is great doesn’t mean it is true for Data

- In the end you care about the sensitivity of your analysis

—> Understand your input!

Exercise: discriminate singletop events from tt events

Run on SWAN: https://swan.web.cern.ch

Repo: https://github.com/llayer/ml_exercise