Lecture 1:
Introduction to Machine Learning:
From Linear Regression to Deep Learning

Amir Farbin

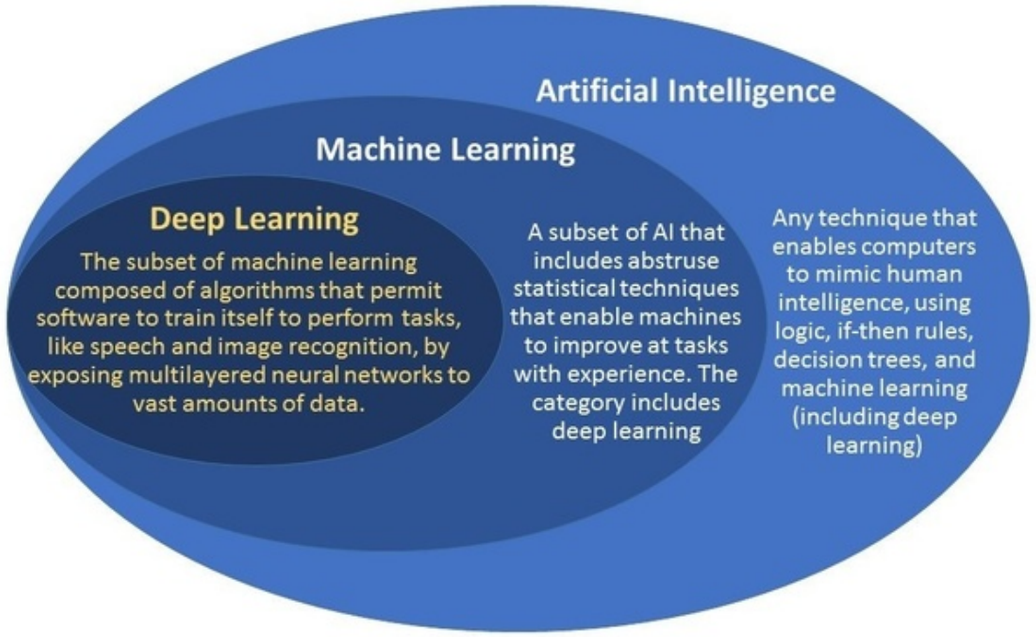
Plans and Disclaimers

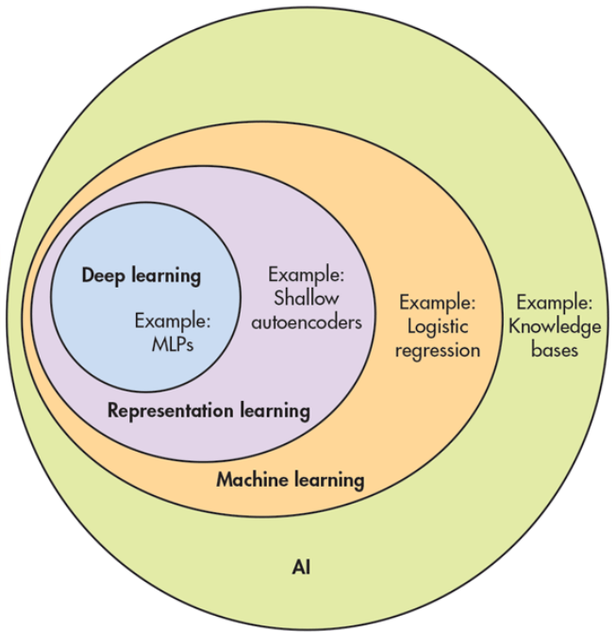
- Plan:
    - *Lecture 1:* Broad Perspective on ML… some formalism
    - *Lecture 2:* Exploration of Data Science / Machine Learning Tools
    - *Lecture 3:* Closer Look at the Techniques I
    - *Lecture 4:* Deep Learning
    - *Lecture 5:* Closer Look at the Techniques II
    - *Lecture 6: Exploration of Deep Learning Tools*
        - *Example Implementations of various architectures and techniques*
        - Deep Learning in HEP
- Disclaimers:
    - I have a very HEP/LHC perspective… some background in B and neutrino physics.
    - Would love to learn about more nuclear physics problems.
    - Lots of stolen graphics… mostly from Wikipedia.

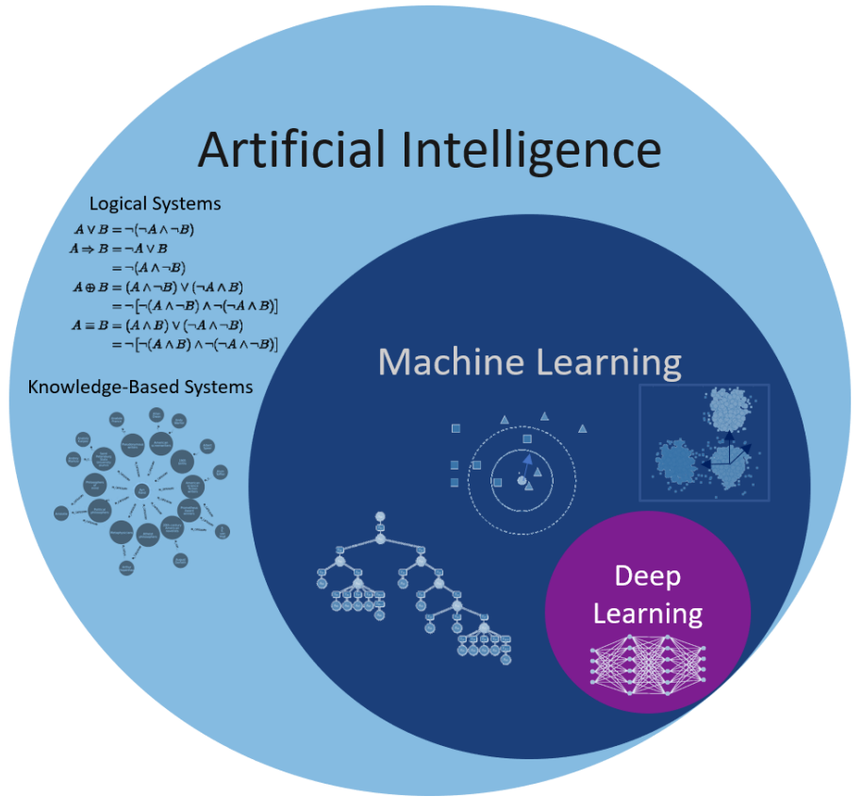
Lecture 1

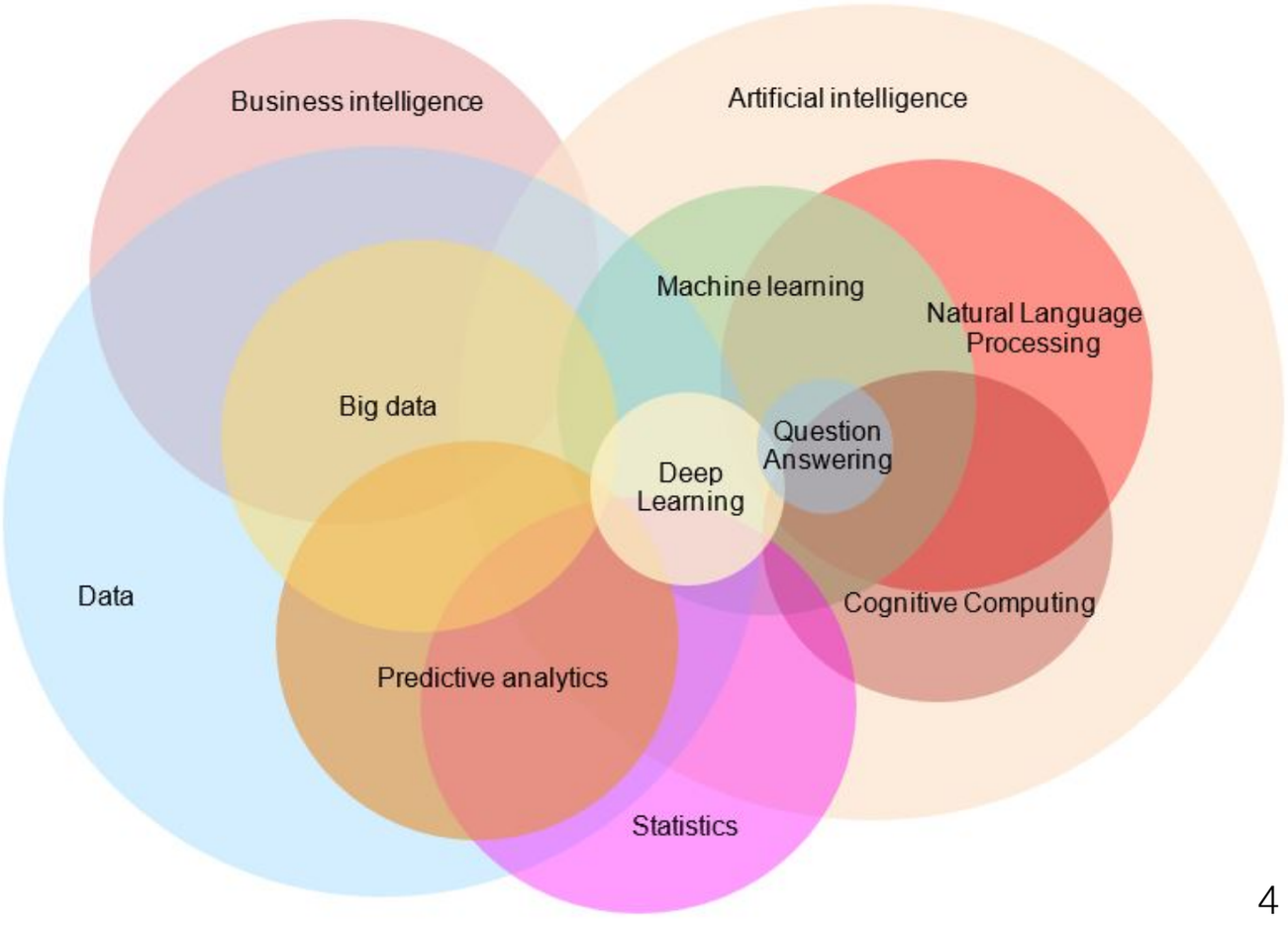
- What is Machine Learning?
- HEP Data
    - What is it?
    - How to think about it.
    - What do we do with it?
- Machine Learning
    - A Formulation
    - Linear Techniques $\longrightarrow$ Kernel Techniques $\longrightarrow$ Decision Trees $\longrightarrow$ Neutral Networks $\longrightarrow$ Deep Learning
    - Bayesian vs Frequentist

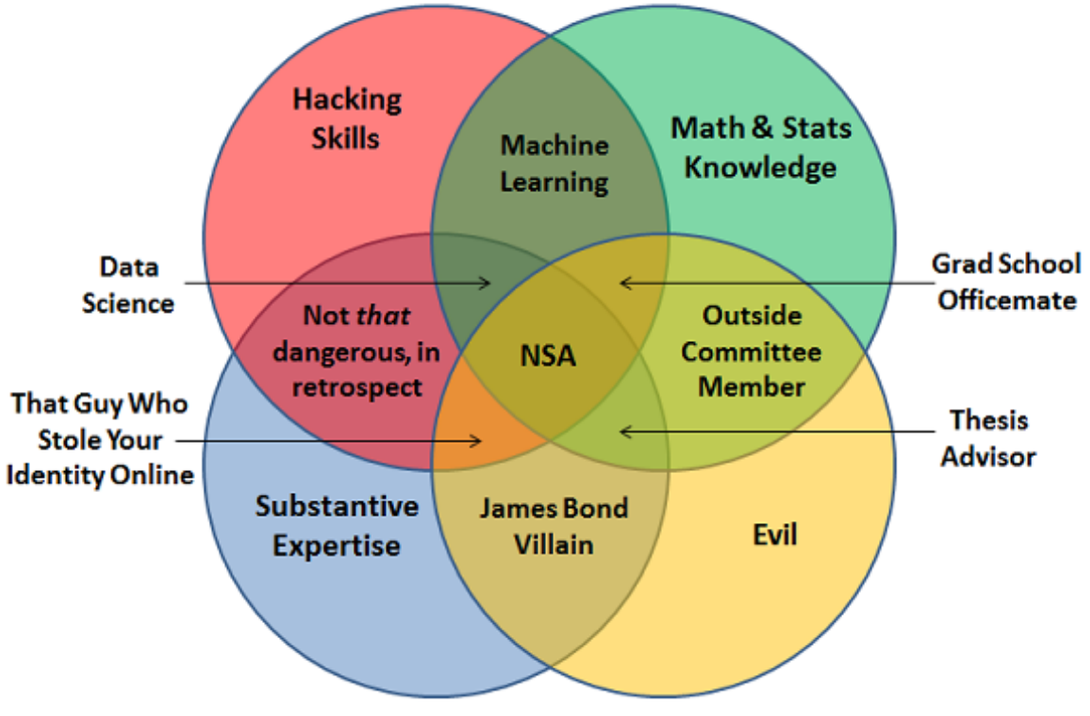
AI vs ML vs DL


Artificial Intelligence
Machine Learning
Deep Learning
The subset of machine learning composed of algorithms that permit software to train itself to perform tasks, like speech and image recognition, by exposing multilayered neural networks to vast amounts of data.
A subset of AI that includes abstruse statistical techniques that enable machines to improve at tasks with experience. The category includes deep learning
Any technique that enables computers to mimic human intelligence, using logic, if-then rules, decision trees, and machine learning (including deep learning)


Deep learning
Example: MLPs
Example: Shallow autoencoders
Representation learning
Example: Logistic regression
Machine learning
Example: Knowledge bases
AI


Artificial Intelligence
Logical Systems
$A \vee B = \neg(\neg A \wedge \neg B)$
$A \Rightarrow B = \neg A \vee B$
$= \neg(A \wedge \neg B)$
$A \oplus B = (A \wedge \neg B) \vee (\neg A \wedge B)$
$= \neg[\neg(A \wedge \neg B) \wedge \neg(\neg A \wedge B)]$
$A \equiv B = (A \wedge B) \vee (\neg A \wedge \neg B)$
$= \neg[\neg(A \wedge B) \wedge \neg(\neg A \wedge \neg B)]$
Knowledge-Based Systems
Machine Learning
Deep Learning


Business intelligence
Artificial intelligence
Machine learning
Natural Language Processing
Big data
Question Answering
Deep Learning
Data
Cognitive Computing
Predictive analytics
Statistics


Hacking Skills
Machine Learning
Math & Stats Knowledge
Data Science
Grad School Officemate
Not *that* dangerous, in retrospect
NSA
Outside Committee Member
That Guy Who Stole Your Identity Online
Thesis Advisor
Substantive Expertise
James Bond Villain
Evil

https://www.wired.com/2017/01/move-coders-physicists-will-soon-rule-silicon-valley/

MOVE OVER, CODERS—PHYSICISTS WILL SOON RULE SILICON VALLEY

Most of the graduate students and postdocs in this room are probably headed in this direction…

it's happening across Silicon Valley. Because structurally and technologically, ***the things that just about every internet company needs to do are more and more suited to the skill set of a physicist***.

But this is a particularly ripe moment for physicists in computer tech, thanks to the rise of machine learning, where machines learn tasks by analyzing vast amounts of data. This ***new wave of data science and AI is something that suits physicists right down to their socks***.

these neural networks are really just math on an enormous scale, mostly linear algebra and probability theory.

Chris Bishop, who heads Microsoft's Cambridge research lab, … ***"There is something very natural about a physicist going into machine learning," he says, "more natural than a computer scientist."***

Physicists know how to handle data—at MIT, Cloudant's founders handled massive datasets from the the Large Hadron Collider—and ***building these enormously complex systems requires its own breed of abstract thought.***

They come because they're suited to the work. And they come because of the money. As Boykin says: "The salaries in tech are arguably absurd." But they also come because there are so many hard problems to solve.

Machine learning will change not only how the world analyzes data but how it *builds software*.

In other words, ***all the physicists pushing into the realm of the Silicon Valley engineer is a sign of a much bigger change to come. Soon, all the Silicon Valley engineers will push into the realm of the physicist.***

AI vs ML vs DL

- ***Artificial Intelligence****:* Any technique that mimics human behavior
  - Code, Logic, Symbolic systems, Knowledge Bases
  - ***Machine Learning***: Any technique that learns from experience (aka Data)
    - Logistic regression (aka fits), Decision Trees, Clustering, Kernel Methods
    - ***Representation Learning*****:** Techniques that learn representations of data amenable to specific or general tasks
      - Shallow Auto-encoders
      - ***Neural Networks***: Biologically inspired ML
        - ***Deep Learning***: Multi-layered Neural Networks
          - MLP, DNN, CNN, RNN, …

Supervised ML

- ***Tasks***: **Classification**, Classification with missing inputs, Regression, Transcription, Machine Translation, Structured Output

- “Traditional” Techniques:

  - Linear/Logistic Regression

  - Support Vector Machines

  - Decision Trees

Un-supervised ML

- ***Tasks***: Clustering, Anomaly Detection, Imputation of Missing Values, Synthesis & Sampling, Denoising, Density Estimation

- "Traditional" Techniques:

  - Principle Component Analysis

  - k-means Clustering

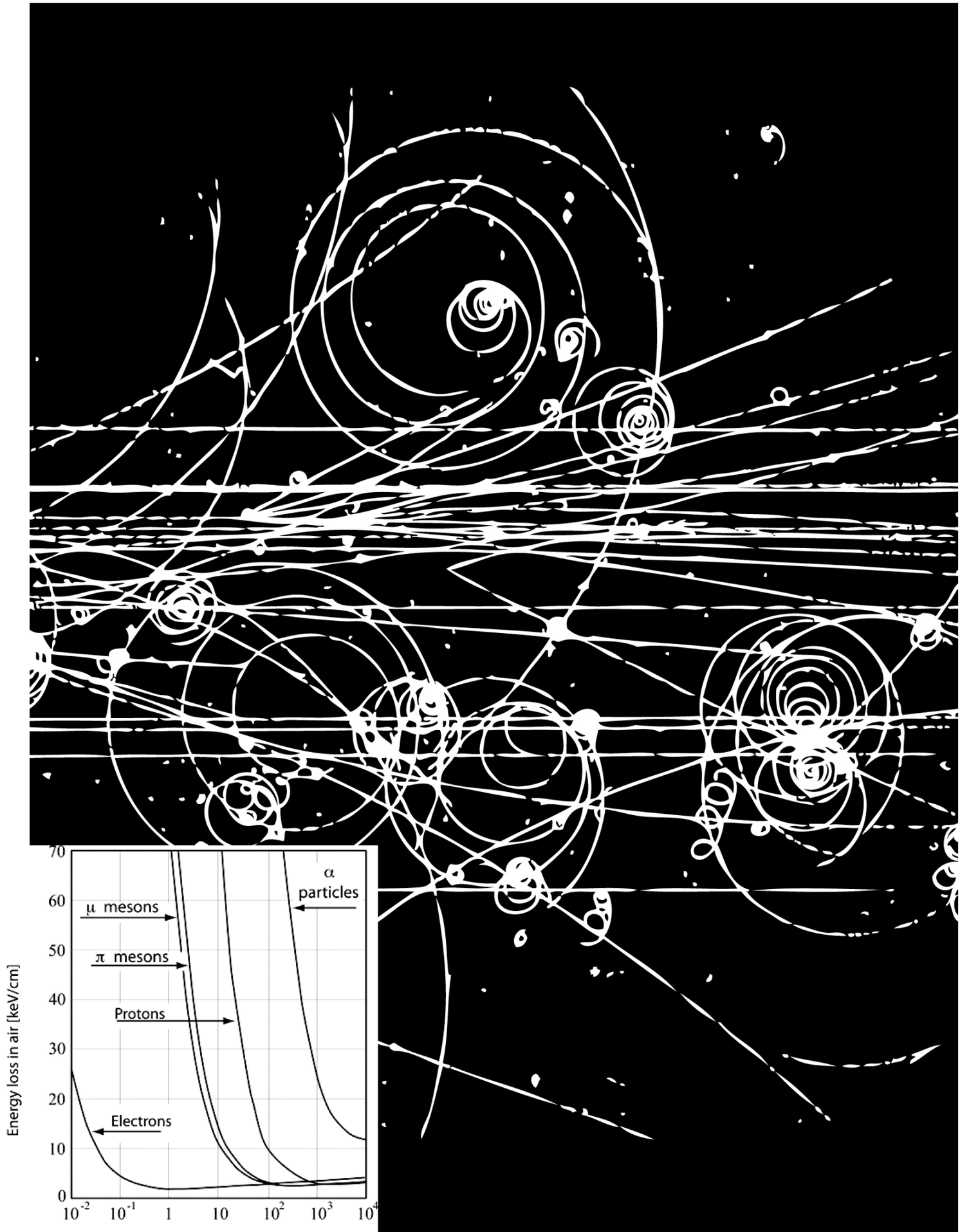
Ingredients of ML

- ***Problem Formulation***. Specify:
  - ***Data Set***: Inputs/Outputs
  - ***ML technique***: F (Input | Parameters) = Output
  - ***Target: Cost (aka loss) function***
    - *Supervised*: Compare F vs Ground Truth Output
    - *Unsupervised*: e.g. Cluster like inputs
    - *Semi-supervised*: F(Input) = Input
- ***Training: Optimization***
  - Choose how to find best parameters

HEP Data

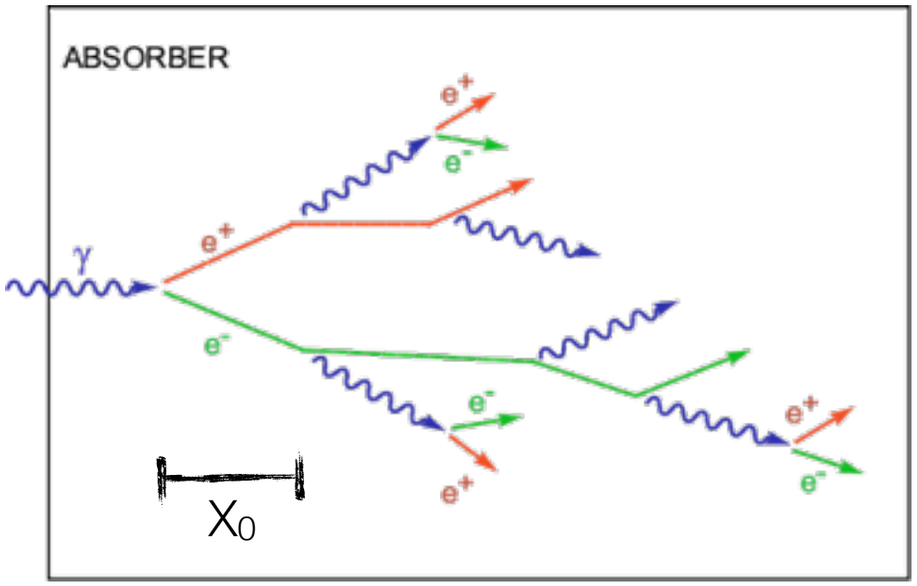
How do we “see” particles?

- ***Charged particles ionize media***
  - Image the ions.
    - In ***Magnetic Field*** the ***curvature*** of trajectory ***measures momentum***.
    - Momentum resolution degrades as less curvature: $\sigma(p) \sim c\ p \oplus d$.
      - d due to multiple scattering.
- Measure ***Energy Loss*** (~ # ions)
  - dE/dx = Energy Loss / Unit Length = f(m, v) = Bethe-Block Function
    - Identify the particle type
  - ***Stochastic process*** (Laudau)
- Loose all energy → range out.
  - Range characteristic of particle type.

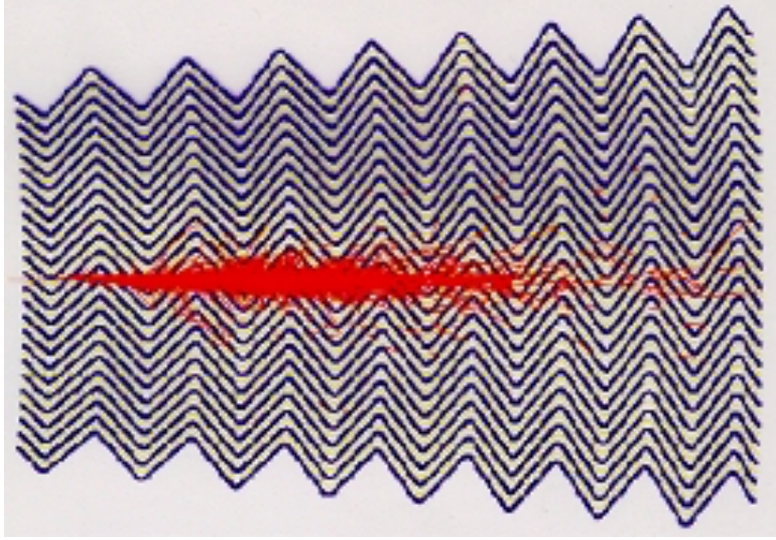


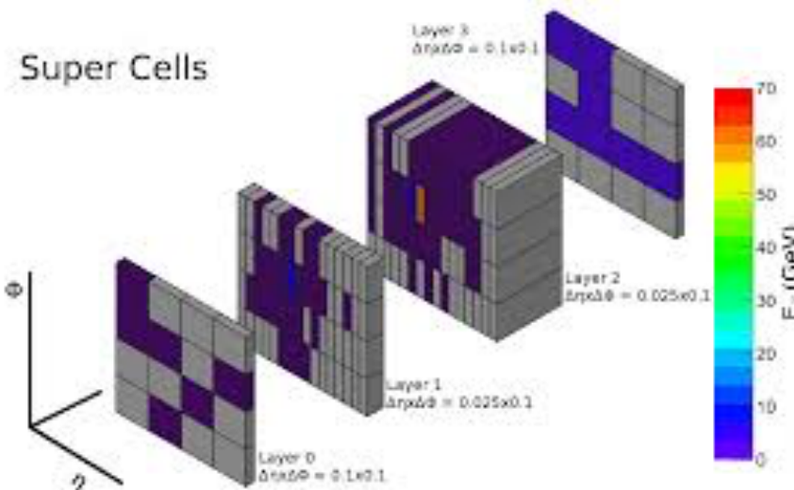
How do we “see” particles?

- Particles deposit their energy in a ***stochastic process*** know as ***“showering”***, secondary particles, that in turn also shower.
  - Number of secondary particles ~ Energy of initial particle.
  - Energy resolution improves with energy: $\sigma(E) / E = a/\sqrt{E} \oplus b/E \oplus c$.
    - a = sampling, b = noise, c = leakage.
  - Density and Shape of shower characteristic of type of particle.



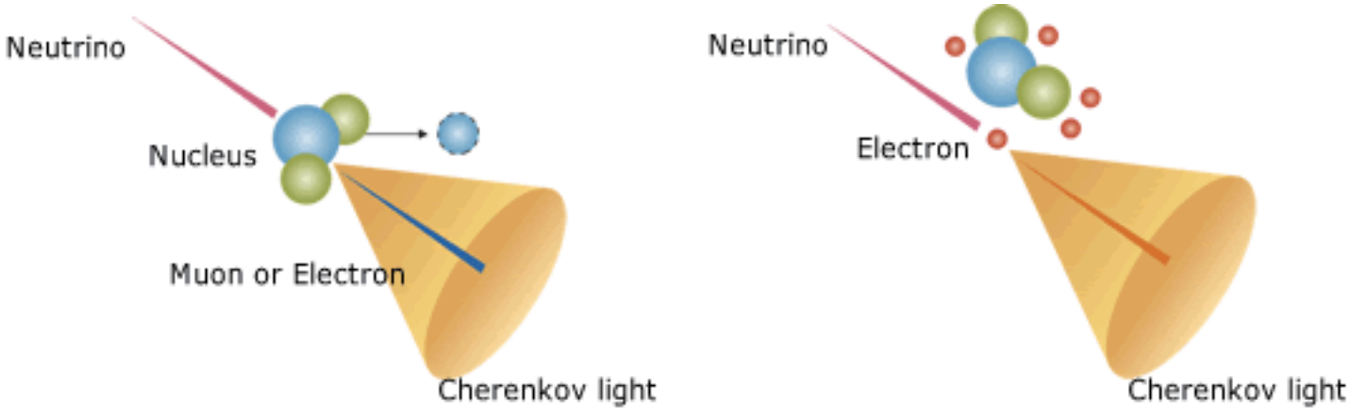
- ***Electromagnetic calorimeter***: Low Z medium
  - ***Light particles***: electrons, photons, $\pi^0 \rightarrow \gamma\gamma$ interact with electrons in medium
- ***Hadronic calorimeters***: High Z medium
  - ***Heavy particles***: Hadrons (particles with quarks, e.g. charged pions/protons, neutrons, or jets of such particles)
    - Punch through low Z.
    - Produce secondaries through strong interactions with the nucleus in medium.
    - Unlike EM interactions, not all energy is observed.



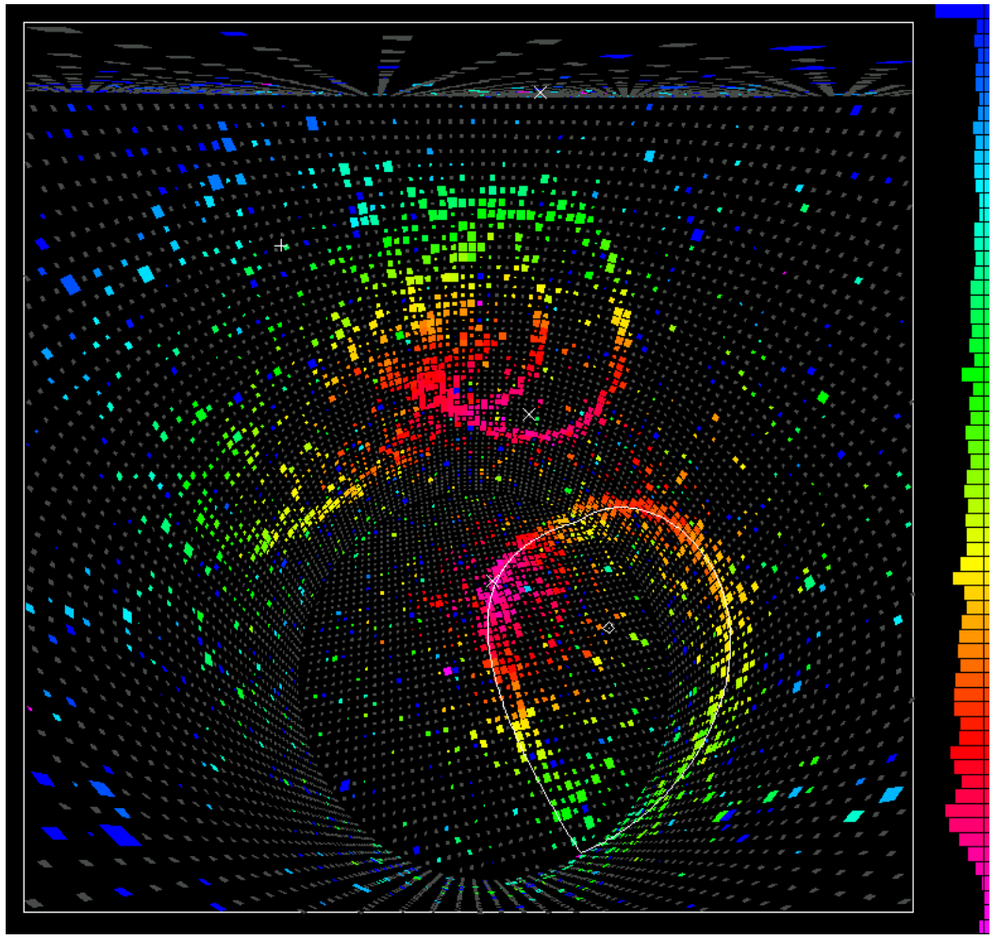


How do we “see” particles?

- Charged Particles traveling faster than speed of light in medium emit ***Cherenkov light*** (analogous to sonic boom).
  - Light emitted in cone, with angle function of speed and mass.
  - Depending on context, allow for particle identification and/or speed measurement.

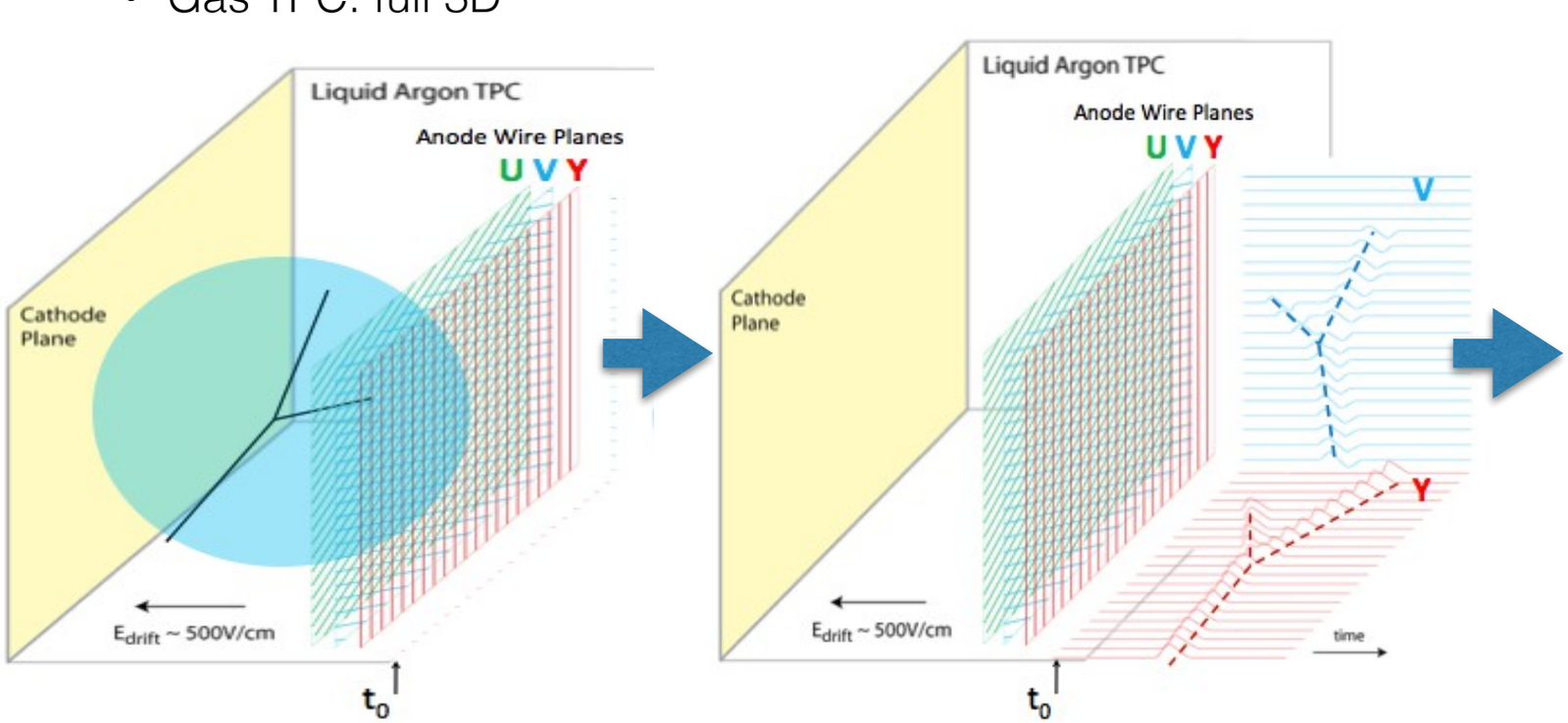


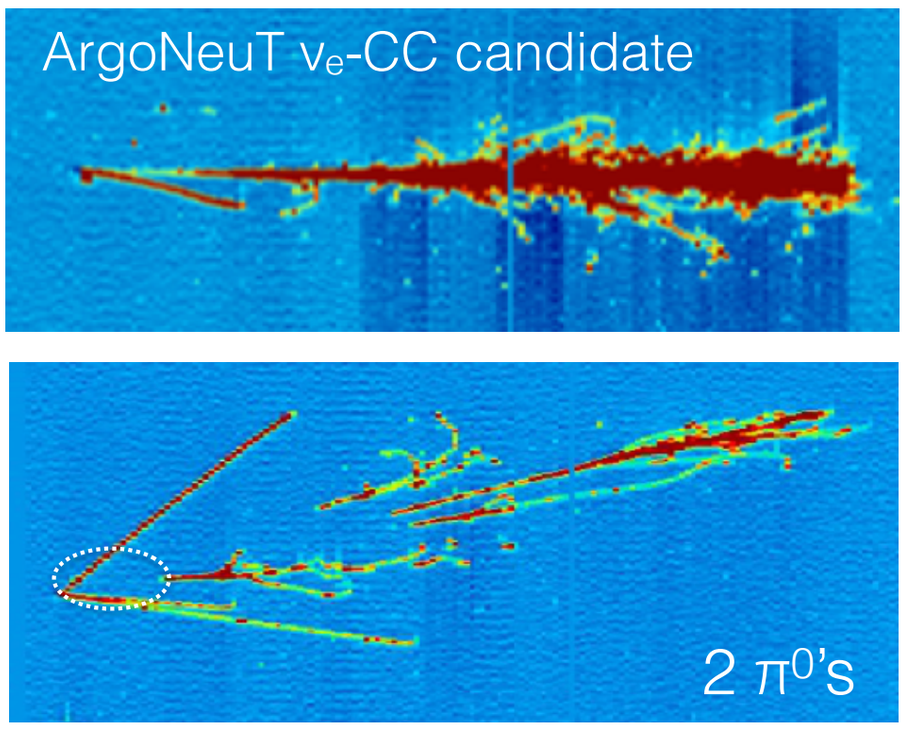
The generated charged particle emits the Cherenkov light.



Neutrino Detectors

- ***Need large mass/volume*** to maximize chance of neutrino interaction.
- Technologies:
  - Water/Oil Cherenkov
  - Segmented Scintillators
  - ***Liquid Argon Time Projection Chamber: promises ~ 2x detection efficiency.***
    - ***Provides tracking, calorimetry, and ID all in same detector***.
    - Chosen technology for US's flagship LBNF/DUNE program.
      - Usually 2D read-out… 3D inferred.
    - Gas TPC: full 3D





HEP Data

- The lowest-level (raw) data we generally have are the digitized outputs of detectors… e.g. voltages.

- Reconstruction is a series of sequential algorithms that construct features from outputs of the previous algorithm.

  - “raw” ⟶ “features”

- Highest level of Reconstruction output is usually particle candidates.

- Analysis, usually

  - choosing candidates ⟶ 4-vectors, separated by PID

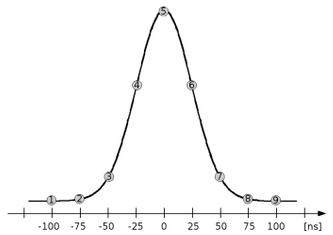
  - 4-vectors ⟶ kinematic features (e.g. masses)

  - kinematic features ⟶ signal/background

  - statistical analysis ⟶ hypothesis test, limits, measurements

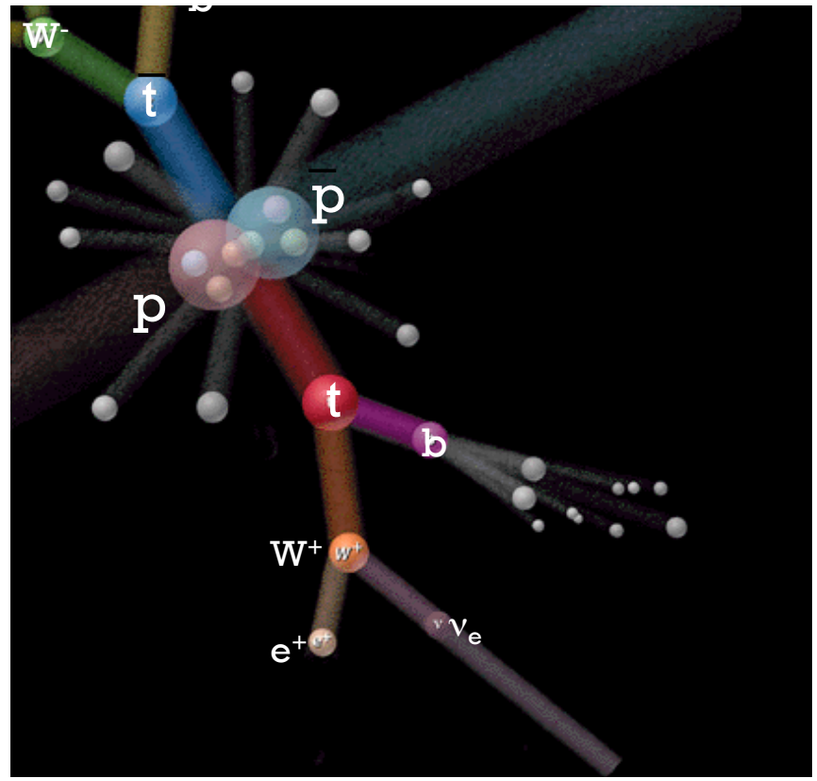
    - Background estimation

    - Lots of systematics

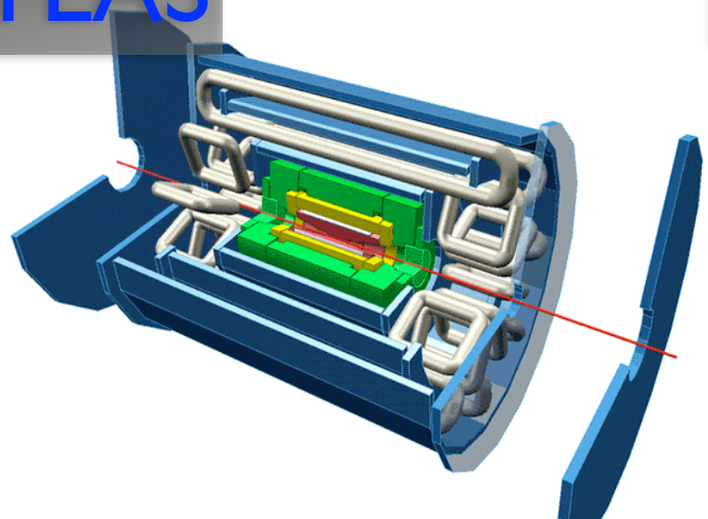


HEP Experiments

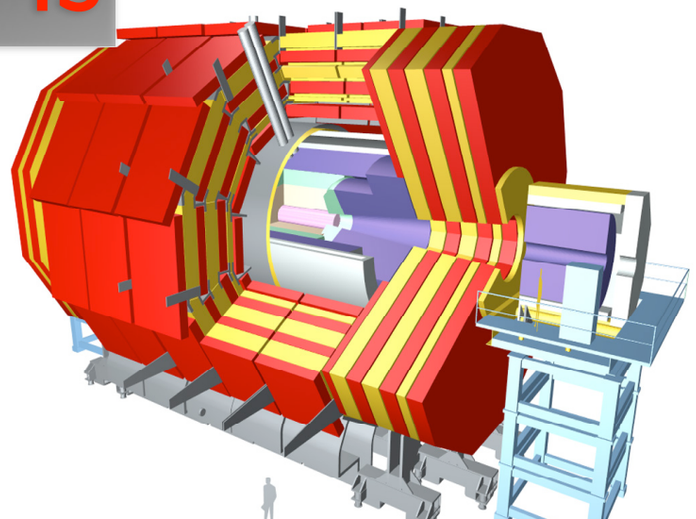
- 5 technical components to HEP experiment:
  - ***Accelerator***: e.g. LHC collisions creating quickly decaying heavy particles. Extremely high rate: 40 * O(50) Million collisions/sec.
  - ***Detector***: a big camera. ~ e.g. LHC 1.5 MB/event (60 TB/s)
    - Pictures of long-lived decay products of short lived heavy/ interesting particles.
    - Sub-detectors parts: Tracking, Calorimeters, Muon system, Particle ID (e.g. Cherenkov, Time of Flight)
  - ***DAQ/Trigger:*** Hardware/software
  - ***Software:*** Reconstruction (Raw data -> particle “features”) / Analysis
  - ***Computing:*** GRID Monarch Model “Cloud” Computing/Data Management (software/hardware)

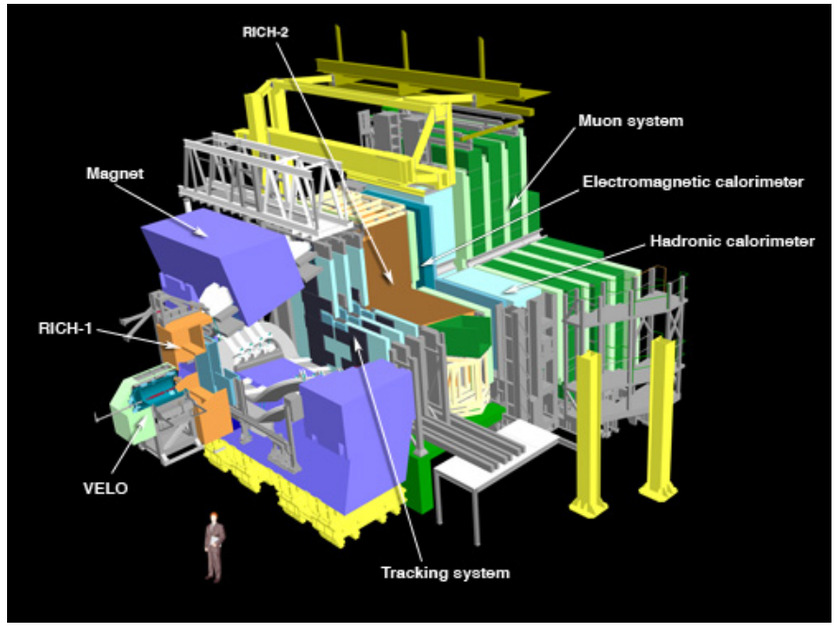


ATLAS



CMS





A bit of Formalism

Data Formulation

- Lets formalize what we mean by a ***dataset*** with a Probabilistic Model:
  - Assumption: Observed **Data is a mixture of *M* different processes**
  - Data set of *N* **data points**, $\{\{x_d\}_i\}$
    - each $\{x_d\}_i$ consisting of
      - *d* **observations** $\{x_d\}$
      - probability f_j of uniquely coming from one of *M* **classes**
        - each class has label c_j is indexed by *j*
        - dependent on parameters $\{\alpha_k\}_j$ (some **parameters of of interest**, some **nuisance parameters**)
    - Dependent on other parameters $\{\beta_l\}$
  - $\implies P(\{x\}|\theta) = P(\{x\}|\{f_j, c_j, \{\{\alpha_k\}_j\}, \{\beta_l\}) = \sum_j f_j P(\{x\}|c_j, \{\alpha_k\}_j, \{\beta_l\})$
- Typical HEP Examples:
  - ***Analysis***: $\{x_d\}$ = {4-vectors}, c_j= {signal, background}, f_j = cross-section * integrated luminosity * efficiency * acceptance, $\{\alpha_j^k\}$= signal/background properties, $\{\beta_l\}$ = detector properties
  - ***Particle ID***: $\{x_d\}$ = {measurements}, c_j= {particle type}, f_j = rate * efficiency, $\{\alpha_j^k\}$= particle properties, $\{\beta_l\}$ = detector properties

What is it good for?

- If we know $P(X|\theta)$, what is it good for?
  - ***Prediction***: Assume $\theta \Longrightarrow$ distribution of $\{x_d\}$.
  - ***Classification***: Observation $\{x_d\} \Longrightarrow$ most likely class c
  - ***Regression***: Dataset $\{\{x_d\}_i\} \Longrightarrow$ parameters of interest $\{\alpha_j{}^k\}$ or $\{\beta_l\}$
  - ***Hypothesis test***: Dataset $\{\{x_d\}_i\} \Longrightarrow$ is H_1 true (or H_0 null hypothesis)

Data Analysis

- Objectives:
  - ***Searches*** (hypothesis testing): Likelihood Ratio Test (Neyman-Pearson lemma)
  - ***Limits*** (confidence intervals): Also based on Likelihood $\frac{P(x|H_1)}{P(x|H_0)} > k_\alpha$
  - ***Measurements***: Maximum Likelihood Estimate

- ***Likelihood***

$$p(\{x\}|\theta) = \mathrm{Pois}(n|\nu(\theta)) \prod_{e=1}^{n} p(x_e|\theta)$$

  - *n* Independent Events (*e*) with Identically Distributed Observables ({*x*})
  - Significant part of Data Analysis is ***approximating the likelihood*** as best as we can.

Obtaining the Likelihood

- How do we obtain $P(X|\theta)$?
  - In HEP, we have precise ***algorithmic simulations*** that generate $\{x_d\}$ given θ.
    - We estimate P by comparing observed x_d with simulated $\{x_d\}$.
- We can build ***analytical first principle models***. *Matrix Element Method* is such a technique.
  - But it's technically difficult, computationally expensive, and only tractable with physics and detector simplifications.
- We can using ML to learn P from simulation or data.

Algorithmic Simulation

- Physics is all about establishing a very precise “model” of the underlying phenomena… so in general ***we can model our data very well.***

- For example for LHC we do ***multi-step ab-initio simulations***:

  1. ***Generation***: Standard Model and New Physics are expressed in language of Quantum Field Theory.

     ➡ Feynman Diagrams simplify perturbative prediction of HEP interactions among the most fundamental particles (leptons, quarks)

  2. ***Hadronization***: Quarks turn to jets of particles via Quantum Chromodynamics (QCD) at energies where theory is too strong to compute perturbatively.

     ➡ Use semi-empirical models tuned to Data.

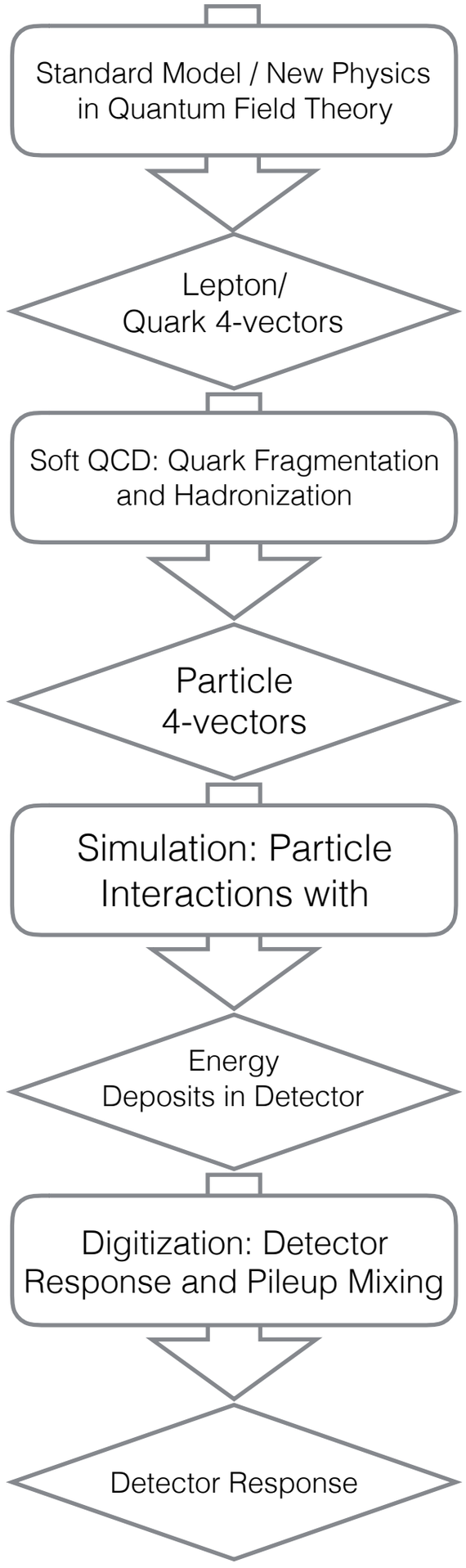
  3. ***Simulation***: Particles interact with the Detector via stochastic processes

     ➡ Use detailed Monte Carlo integration over the “micro-physics”

  4. ***Digitization***: Ultimately the energy deposits lead to electronic signals in the O(100 Million) channels of the detector.

     ➡ Model using test beam data and calibrations.

- Output is fed through ***same reconstruction as real data***.



Likelihood Approximations

- Need $P(\{x_d\}|\theta)$ of an observed event (i). The better we do, the more sensitive our measurements.

- Steps 2 (Hadronization) and 3 (Simulation) can only be done in the ***forward mode***…

  ➡ ***cannot evaluate the likelihood.***

- So we simulate a lot of events and generally use histograms (a crude *Probability Density Estimator (PDE) technique)*

- $\{x_d\}$ = {100M Detector Channels} or even { particle 4-vectors } are too high dimensional.

  - Curse of dimensionality… more on this later

- Instead we derive $\{x_d\}$ = { small set of physics motivated observables } → ***Lose information.***

  - ***Isolate signal*** dominating regions of $\{x_d\}$ → ***Lose efficiency.***

  - Sometimes use ***ML-based classifiers*** to further reduce dimensionality and improve significance

  - ***Profile the likelihood*** in 1 or 2 (ideally uncorrelated) observables.

Machine Learning

Basic Formulation

- ***AI Algorithm***: *Input* ⟶ AI Algorithm ⟶ *Output*
  - $X \longrightarrow F(\theta) \longrightarrow y$: $y = F(X|\theta)$
    - X: observable inputs
    - θ: Model parameters (most likely learned)
    - y: observable outputs
- ***Typical HEP ML task***: Isolate signal in real data $\{ x_d \}_i$
  - ***Formulate problem***: $X = x_d$, y = 0 or 1 if c = background or signal
- ***Obtain training data***: generate simulated labeled Dataset $\{x_d, c\}_i$
  - ***Separate dataset:*** into *Train, Test*, and possibly *Validation* subsets
- ***Choose***: Pick an ML algorithm F
- ***Train***: Use labeled Dataset $\{x_d, c\}_i$ to obtain θ
- ***Test***: Estimate $P(F(X|\theta)|c_j)$ using Test sample
- ***Optimize***: Select F(X|θ) > cut to optimize sensitivity for a hypothesis test
  - $P(F(X|\theta)|$ *signal* $)$ / $P(F(X|\theta)|$ *background* $)$
- ***Validate***: Obtain signal / background efficiency using Validation sample
- ***Apply***: to data $\{x_d\}_i$

ML Models (Linear)

- Linear Models: $F(\vec{x}) = \mathbf{W}\vec{x} + \vec{b}$
  - Solve equation $\vec{y} = \mathbf{W}\vec{x} + \vec{b}$
  - for “weights” w in matrix W and biases b
  - by minimizing $\frac{1}{N}\sum_{i}^{N}\left(F(\vec{x}|\mathbf{W},\vec{b}) - \vec{y}\right)^2$
  - Note
    - W is d by m matrix
    - x is d component vector
    - y is m component vector
- Simple example: b = 0, W is 1 by d
  - $\Longrightarrow$ Analytic solution

$$W = (X^T X)^{-1} X^T Y \qquad X = \begin{pmatrix} \vec{x}_1 \\ \vec{x}_2 \\ \dots \\ \vec{x}_N \end{pmatrix} \quad Y = \begin{pmatrix} \vec{y}_1 \\ \vec{y}_2 \\ \dots \\ \vec{y}_N \end{pmatrix}$$

ML Models (LDA)

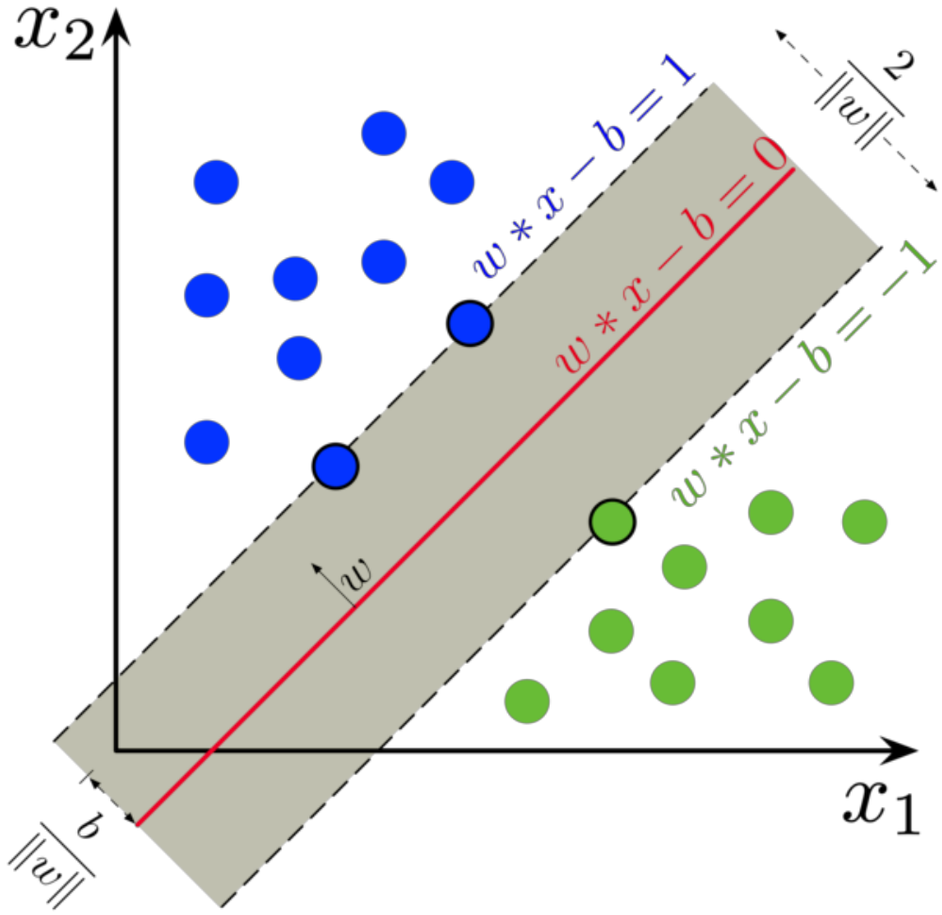
- Linear Discriminant Analysis (Fisher Discriminant)
  - Assume 2 classes: 0 and 1
    - Means μ_0, μ_1 and covariance matrices Σ_0, Σ_1
  - Model: $\vec{y} = \mathbf{W}\vec{x}$
    - Again W is 1 by d, b=0
  - Maximize: $S = \frac{(\mathbf{W} \cdot \vec{\mu}_0 - \mathbf{W} \cdot \vec{\mu}_1)^2}{\mathbf{W}^T \boldsymbol{\Sigma}_0 \mathbf{W} - \mathbf{W}^T \boldsymbol{\Sigma}_1 \mathbf{W}}$
    - Maximal separation between means
    - Smallest possible variance
- Analytic Solution: $\mathbf{W} = (\boldsymbol{\Sigma}_0 + \boldsymbol{\Sigma_1})^{-1} (\vec{\mu_0} - \vec{\mu_1})$

ML Models (Linear SVM)

- Support Vector Machines:
  - Same setup… $\vec{y} = \vec{W} \cdot \vec{x} + \vec{b}$
  - Note that this is an equation for a hyperplane ⟶ formulate problem as finding the hyperplane that optimally separates two classes.
  - Maximize boundary between two classes ⟹ plane depends on points closest to the plane / most difficult to separate… the *Support Vectors*.
- Solving this problems is equivalent to solving a dual problem of solving for a_i

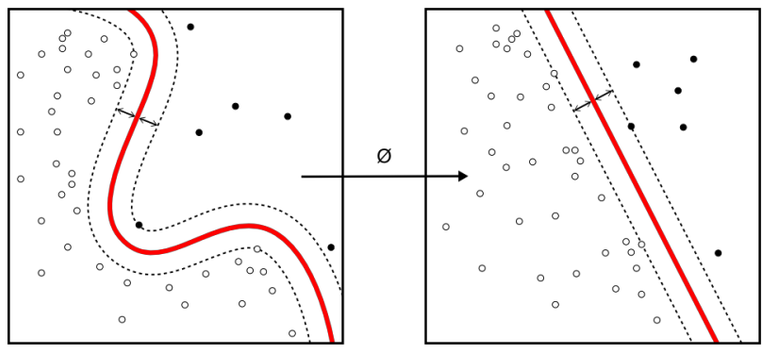
$$f(\vec{x}) = (\sum_{i}^{N} a_i y_i \vec{x}_i) \cdot \vec{x} + \vec{b}$$

  - Where y_i={+1,-1} depending on class.
  - $a_i \geq 0$.
- Dot product measures similarity.
  - ⟶ Sign of output dependent on sign of the most similar example in training set.
  - Support vectors still most important for defining boundary

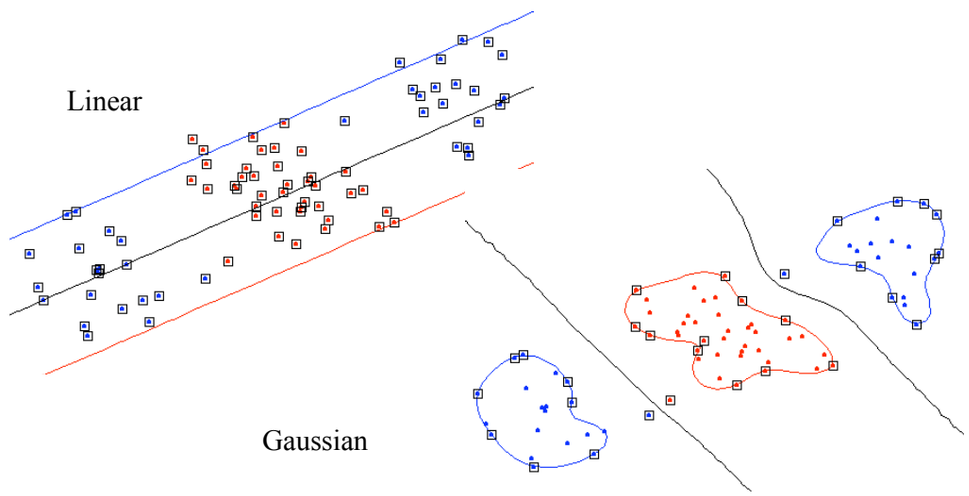


ML Models (Kernel SVM)

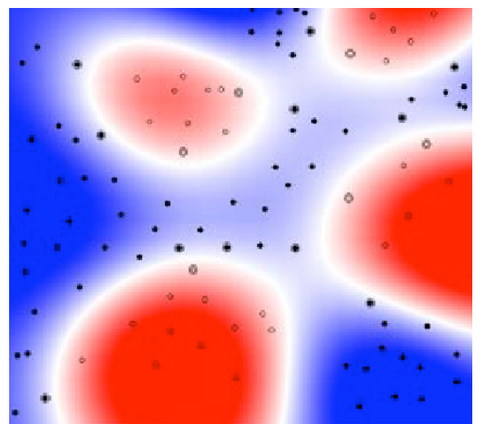
- If boundary not linear $\Longrightarrow$ Kernel Trick:
  - Replace dot product: $f(\vec{x}) = \sum_i^N a_i y_i k(\vec{x}_i, \vec{x}) + \vec{b} \qquad k(\vec{x}_i, \vec{x}) = \phi(\vec{x_i}) \cdot \phi(\vec{x})$
- Example: if *x* is in radial coordinates, then kernel gets back proper dot product.



- Example: Gaussian Kernels
  - Euclidian Distance in the exponential
  - If close to an example in training data $\Longrightarrow$ large value
  - Equivalent to template matching



Radial Basis Functions



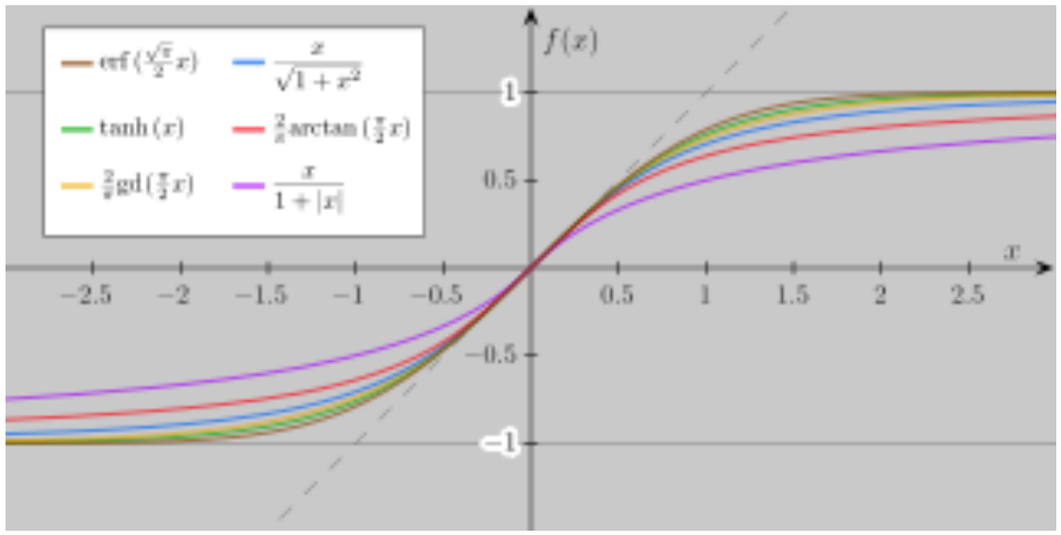
Examples from: http://web.mit.edu/6.034/wwwbob/svm-notes-long-08.pdf

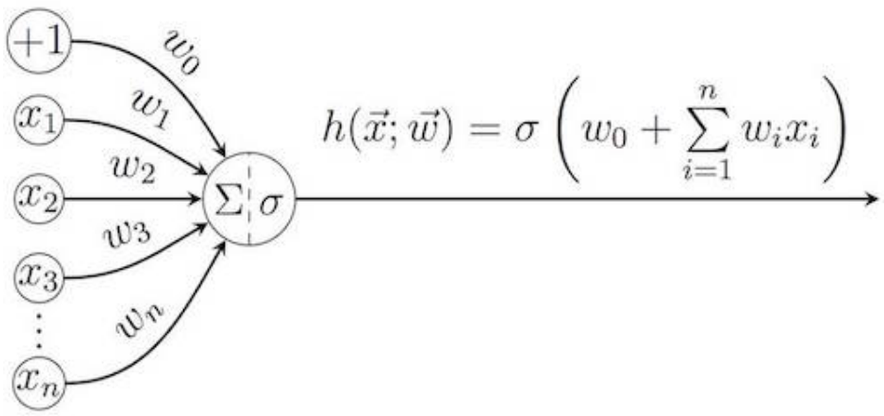
Features

- $X \longrightarrow \phi(X) \longrightarrow$ Linear Model
- ϕ are features or representation derived from X
- ϕ is:
  - Generic, e.g. polynomials $\longrightarrow$ e.g. SVM
  - Manually constructed $\longrightarrow$ Feature Engineering
  - Learned $\longrightarrow$ Feature Learning

Artificial Neural Network

- A simple one layer NN
  - F(**X** | **a**= **W**,**b**) = f (**W X** + **b**)
    - **W, b** = “weights”, “biases”
    - f(x)= “activation function”
      - Must be non-linear.





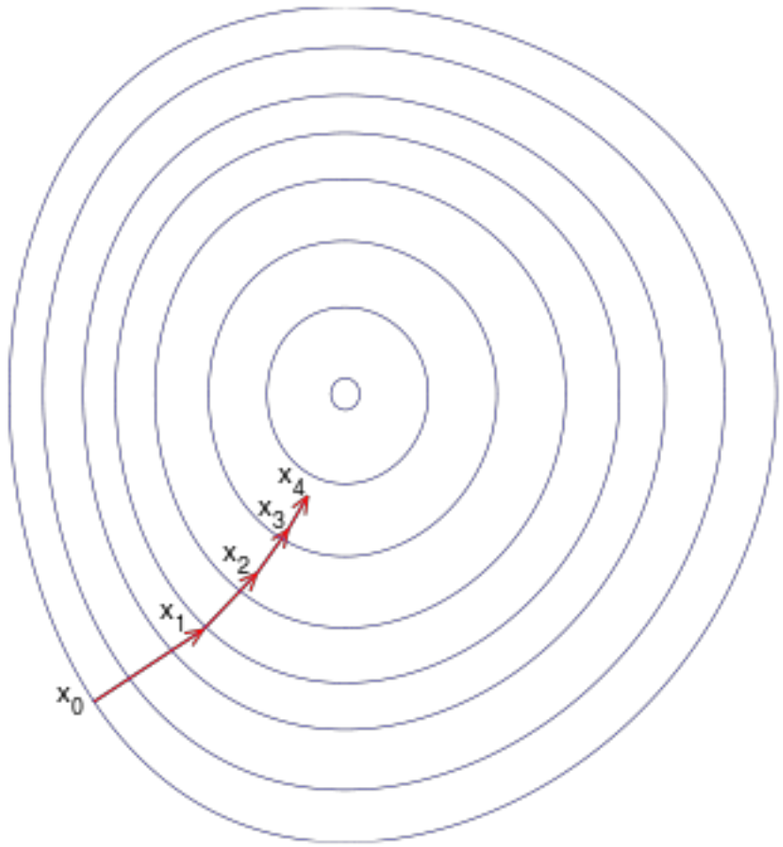
- Universal Computation Theorem.

Training == Optimization

- Training = Minimizing cost function w.r.t. parameters **α**

$$C[F(\vec{X}_{train}|\vec{\alpha}), \vec{Y}_{train}] \equiv C(\vec{\alpha})$$

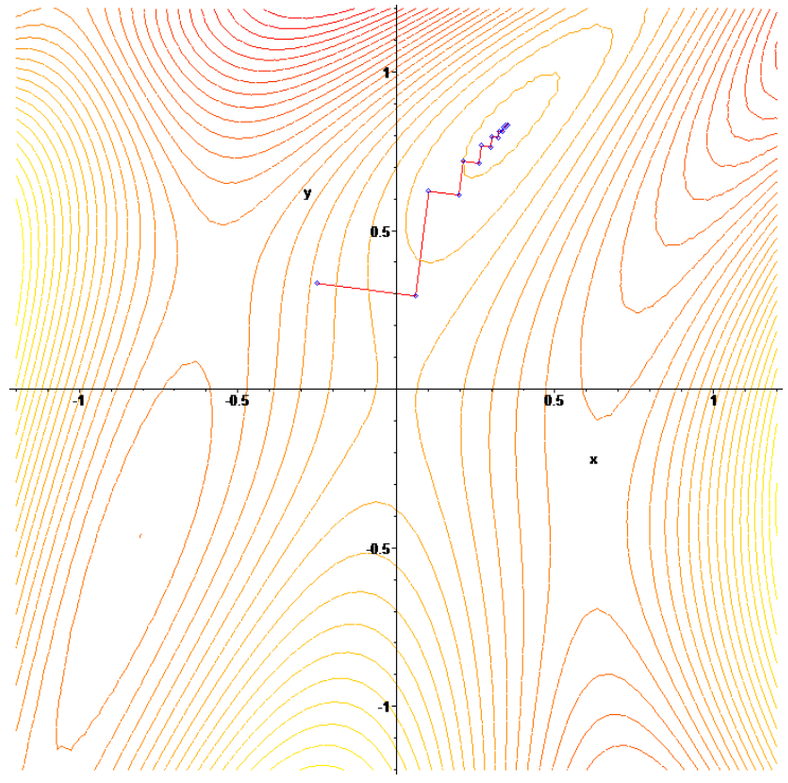
- Gradient Decent (Newton's Method):



- Gradient points to direction of maximal change.

- Iterate (ε sets the step size == *Learning Rate*)

$$\vec{\alpha}_{i+1} = \vec{\alpha}_i - \epsilon \nabla C(\vec{\alpha})$$



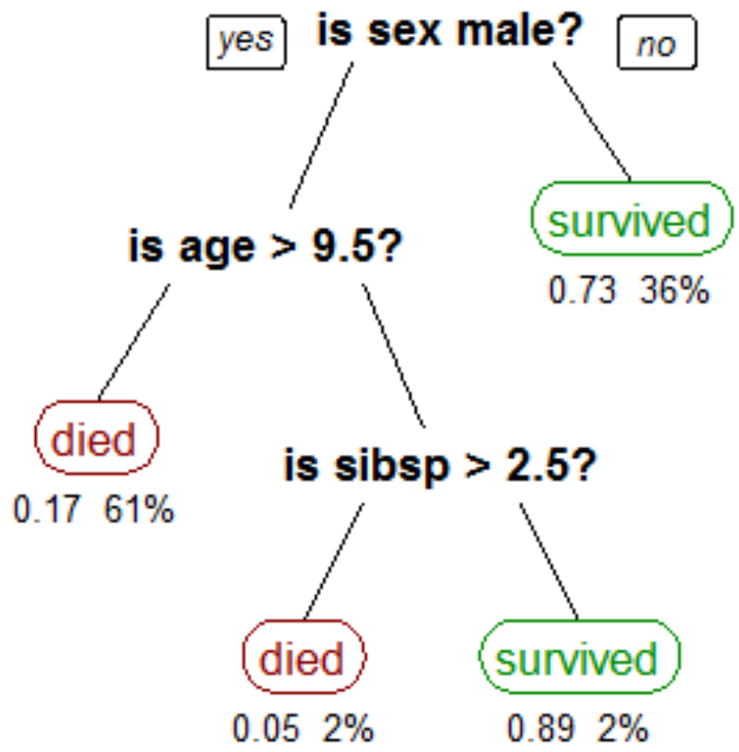
Bayesian vs Frequentist

- ***Supervised Learning:***
  - Data: (X , Y)
  - True: $f^*(X) = Y$
  - Learn: $f(X|\theta) \sim f^*$
- ***Frequentist***:
  - (X, Y) random.
  - Ideal θ exists.
  - Estimate θ.
- ***Bayesian***:
  - (X, Y) fixed.
  - θ is random.
  - Learn $p(\theta)$

- ***Bayesian Learning:***
  - For example: θ is Gaussian distributed $\Longrightarrow$ learn μ, σ
  - Allows you to propagate uncertainties $\Longrightarrow$ estimate uncertainty on output Y

Decision Trees

- Doesn’t fit the F(**X** | **W**,**b**) formulation.
- Powerful technique
  - Random Forest
  - Boosting
- Covered by Daniele
- Up to recently, in HEP, Boost Decision Trees (BDTs) on well constructed/chosen features
  - have been the best performing techniques, and
  - become the standard to beat,

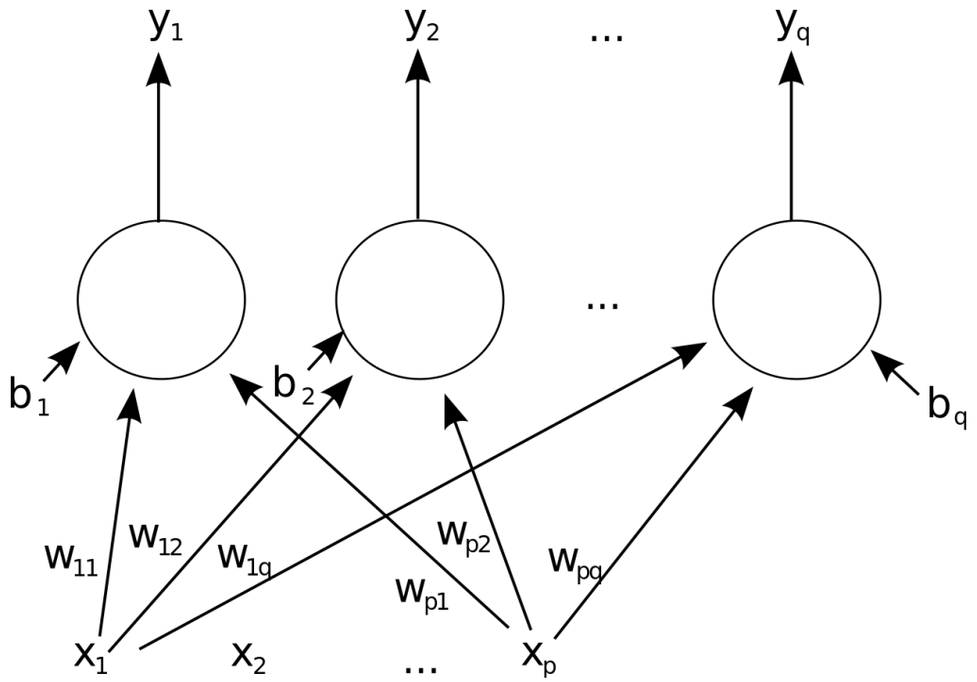


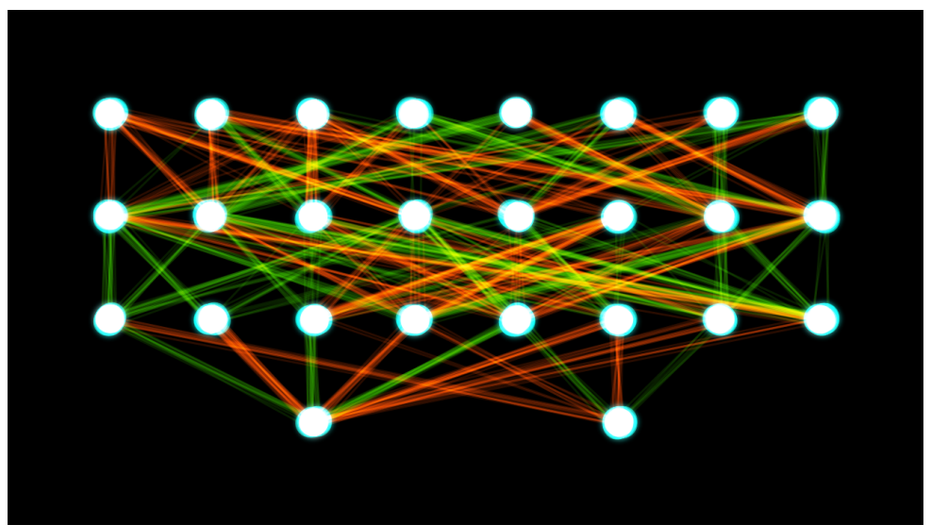
Titanic Survivors

Deep Learning

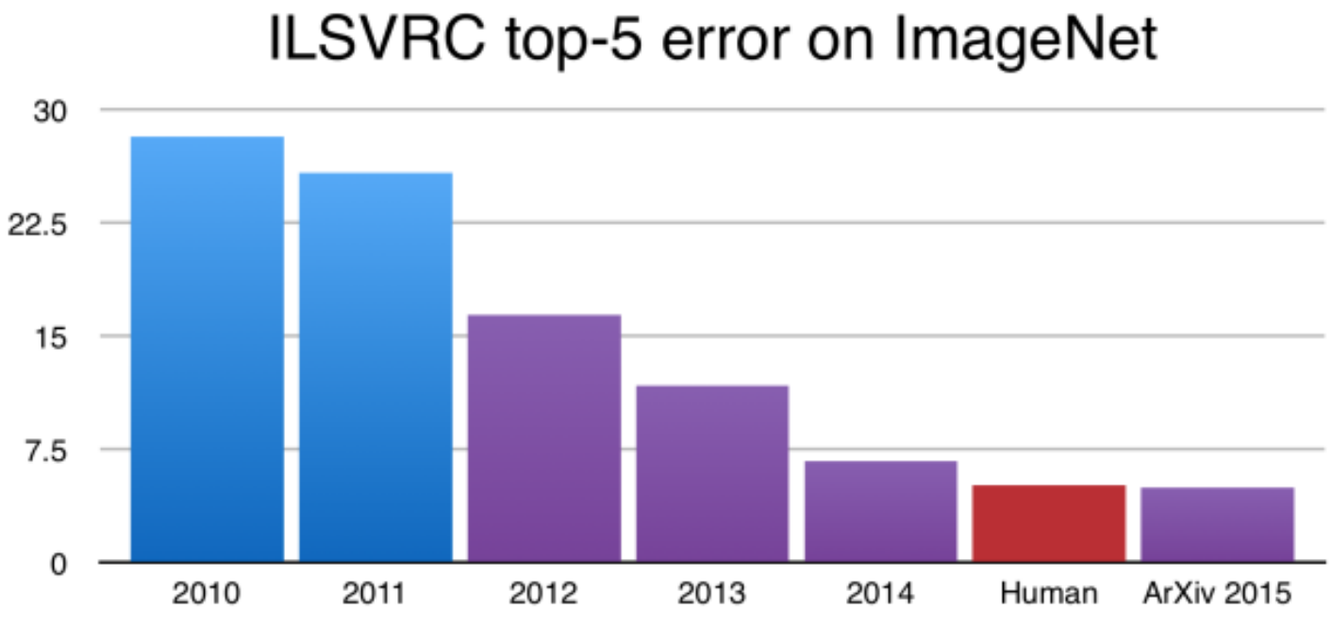
Artificial Neural Networks

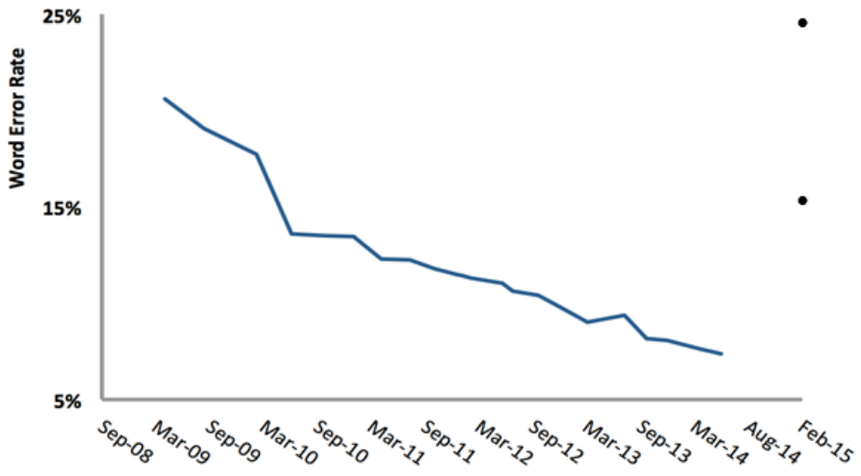
- ***Biologically inspired computation***, (first attempts in 1943)
  - ***Probabilistic Inference***: e.g. signal vs background
  - ***Universal Computation Theorem*** (1989)
- Multi-layer (***Deep***) Neutral Networks:
  - Not a new idea (1965), just impractical to train. ***Vanishing Gradient problem*** (1991)
  - Solutions:
    - New techniques: e.g. better activation or layer-wise training
    - ***More training***: big training datasets and lots of computation … ***big data and GPUs***
- ***Deep Learning Renaissance***. First DNN in HEP (2014).
- ***Amazing Feats***: Audio/Image/Video recognition, captioning, and generation. Text (sentiment) analysis. Language Translation. Video game playing agents.
- ***Rich field***: Variety of architectures, techniques, and applications.



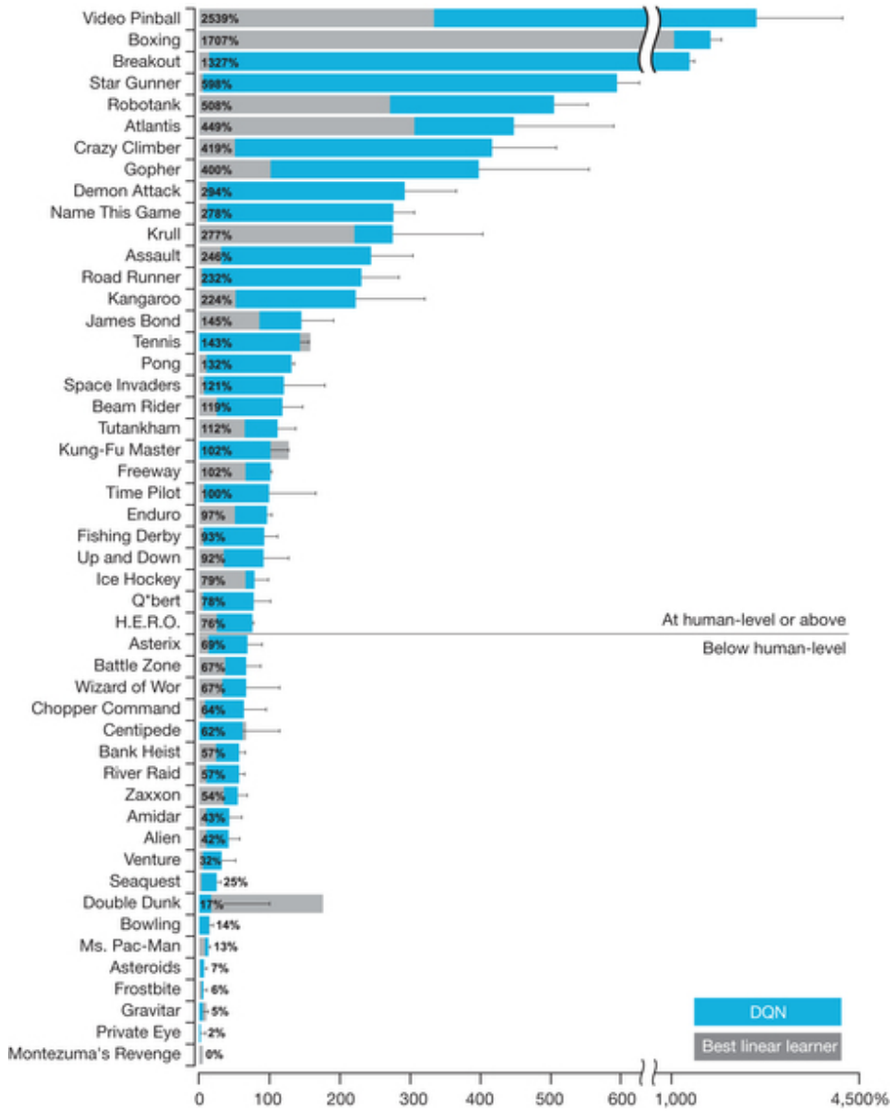


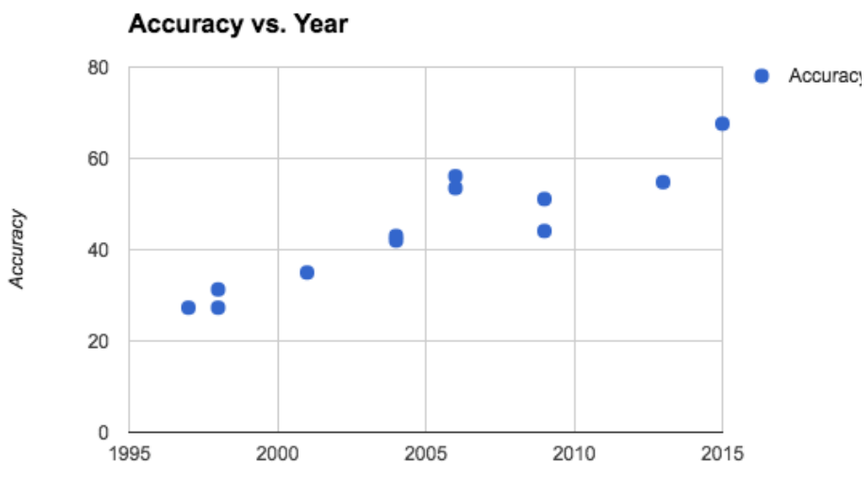
Images from Wikipedia

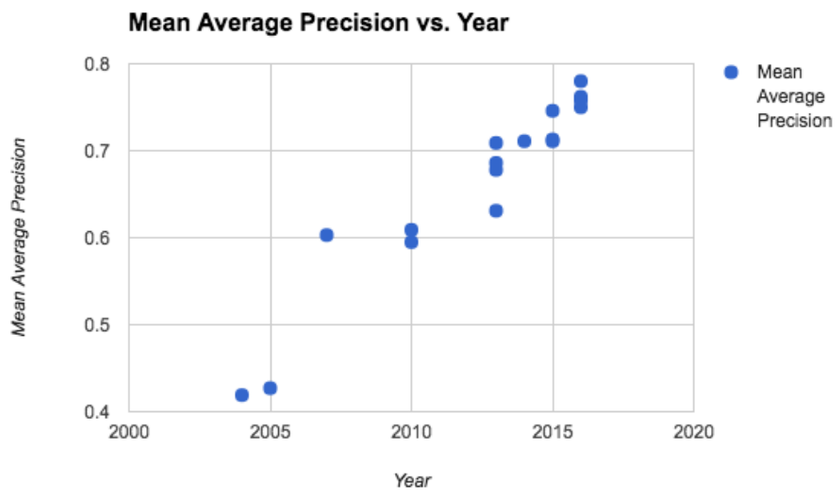


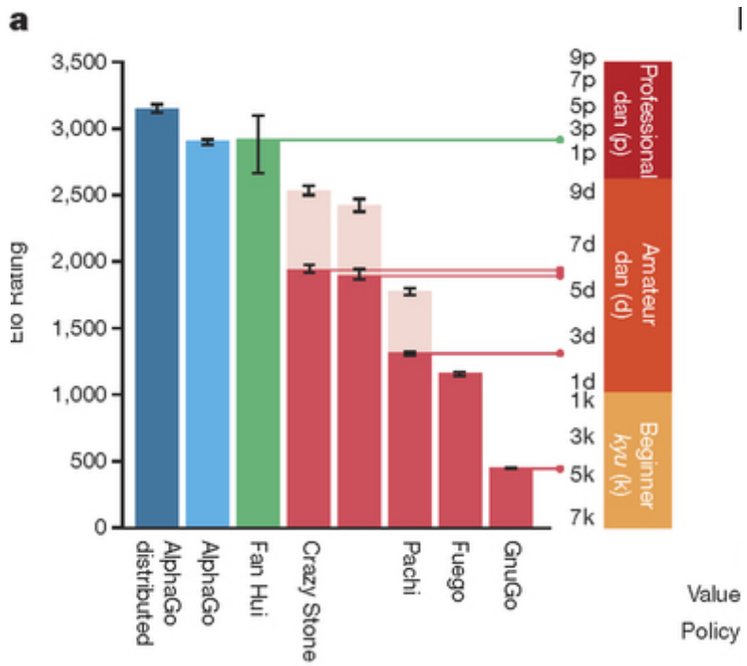


- Continuous server ASR word error rate (WER) reduction ~18% / year: combination of algorithms, data, and computing
- Deep learning (DNNs) is driving recent performance improvements in ASR and meaning extraction



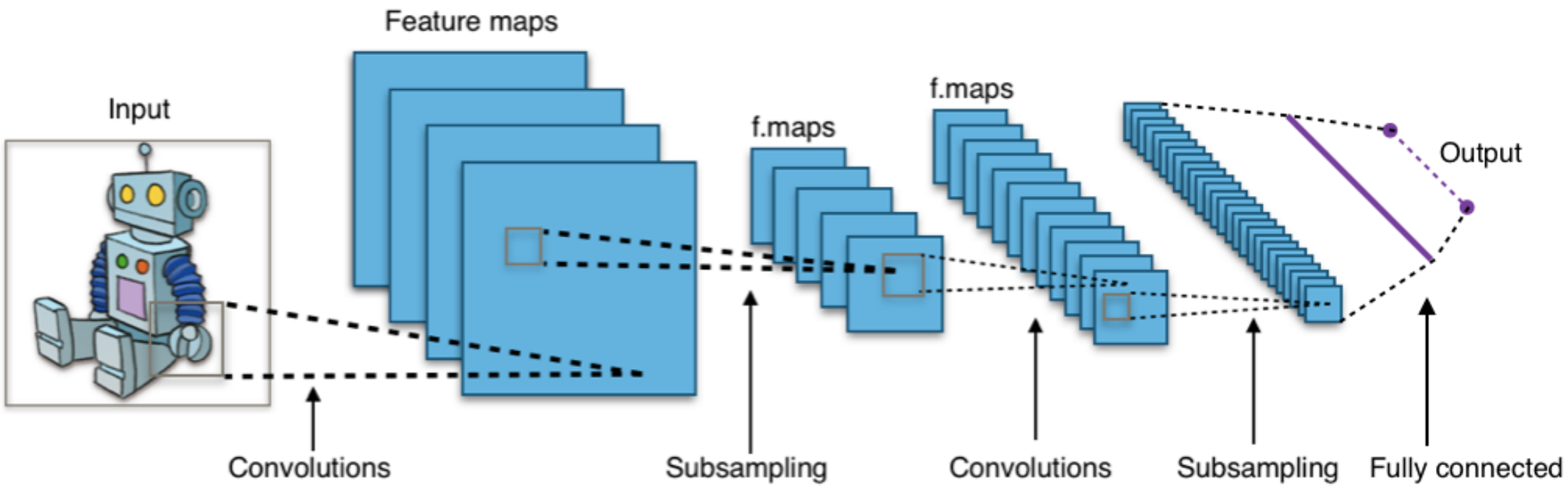


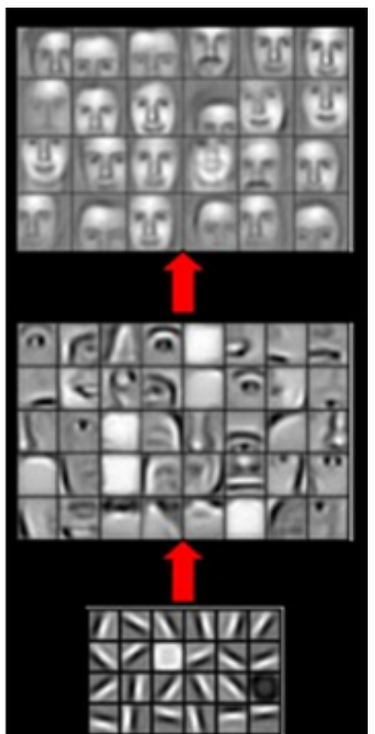




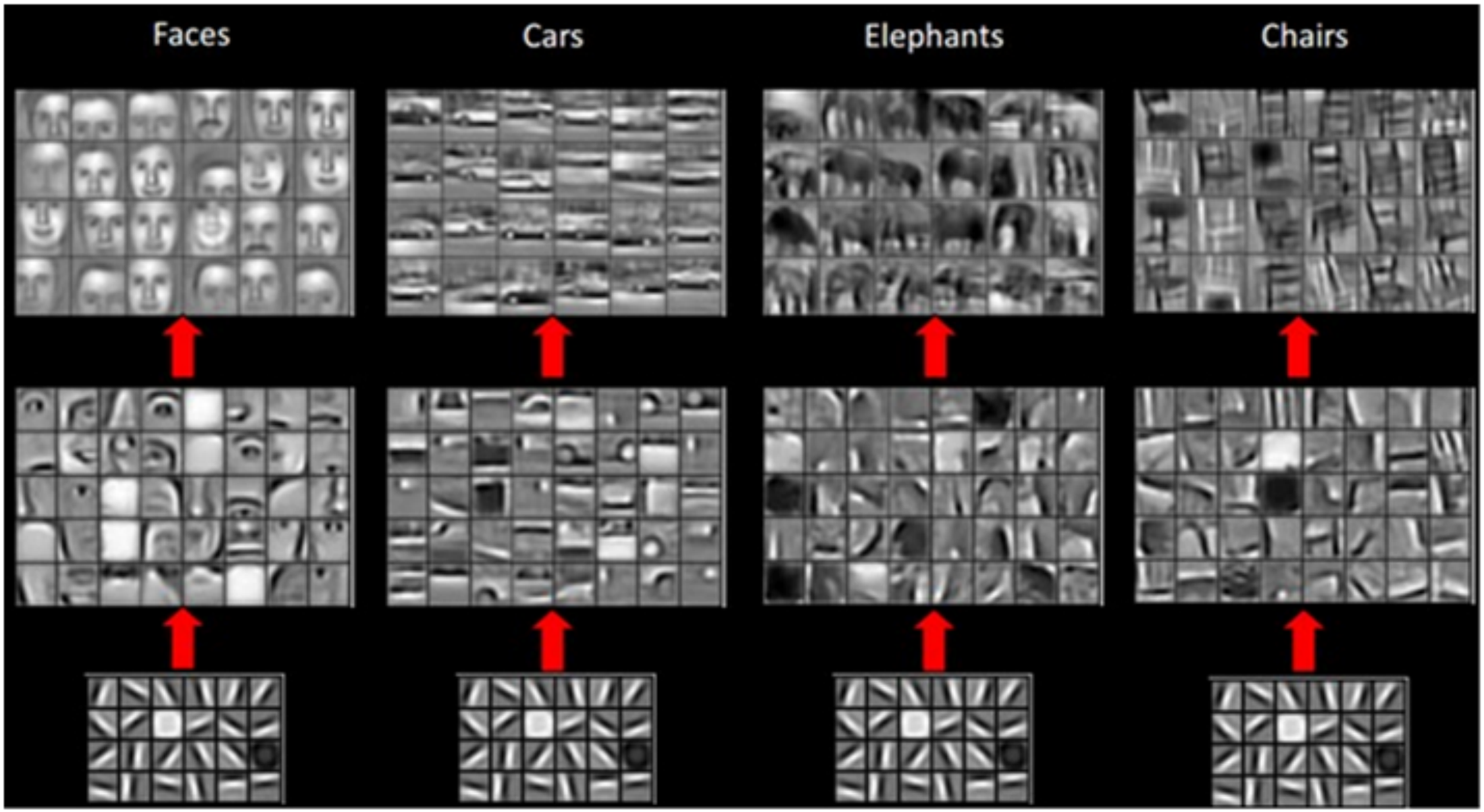
Feature Learning

- ***Feature Engineering***: e.g. Event Reconstruction ~ Feature Extraction, Pattern Recognition, Fitting, …
- Deep Neutral Networks can ***Learn Features*** from ***raw data***.
- Example: ***Convolutional Neural Networks*** - Inspired by visual cortex
  - ***Input***: Raw data… for example 1D = Audio, 2D = Images, 3D = Video
  - ***Convolutions*** ~ learned feature detectors
  - ***Feature Maps***
  - ***Pooling*** - dimension reduction / invariance
  - ***Stack***: Deeper layers recognize higher level concepts.



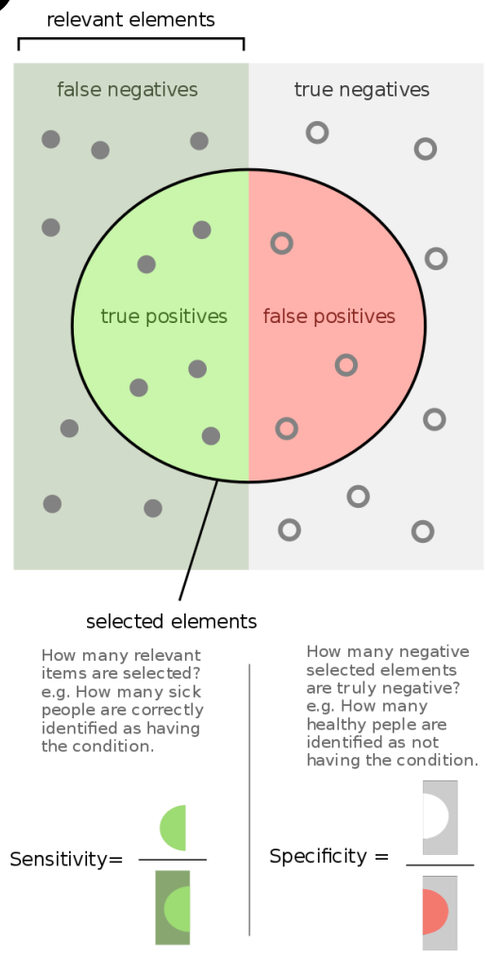


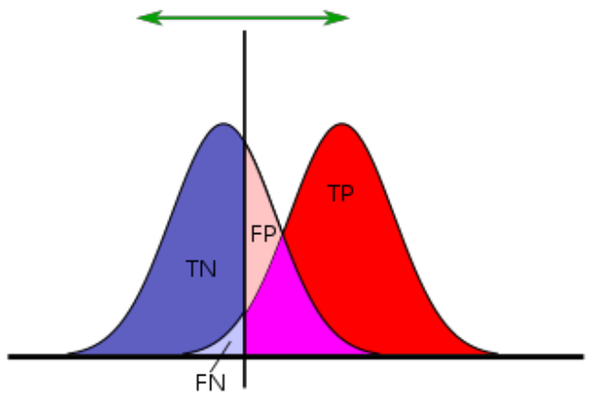
Deep Neutral Networks



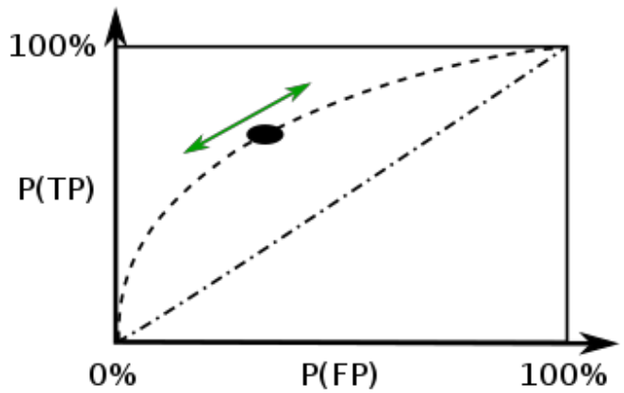
Assessing Performance

- ***True Positive Rate (TPR)***: Efficiency
- ***False Positive Rate (FPR)***: Background efficiency, 1/Background Rejection
- ***Receiver Operator Curve (ROC)***: TPR vs FPR
- ***Area Under Curve (AUC)***: Area under ROC





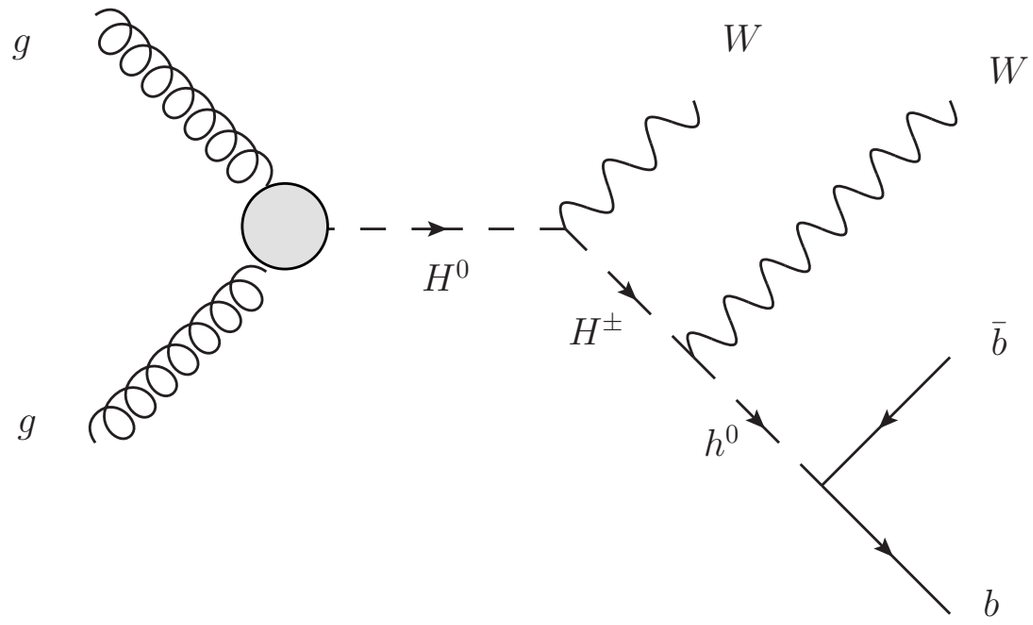
TP	FP
FN	TN

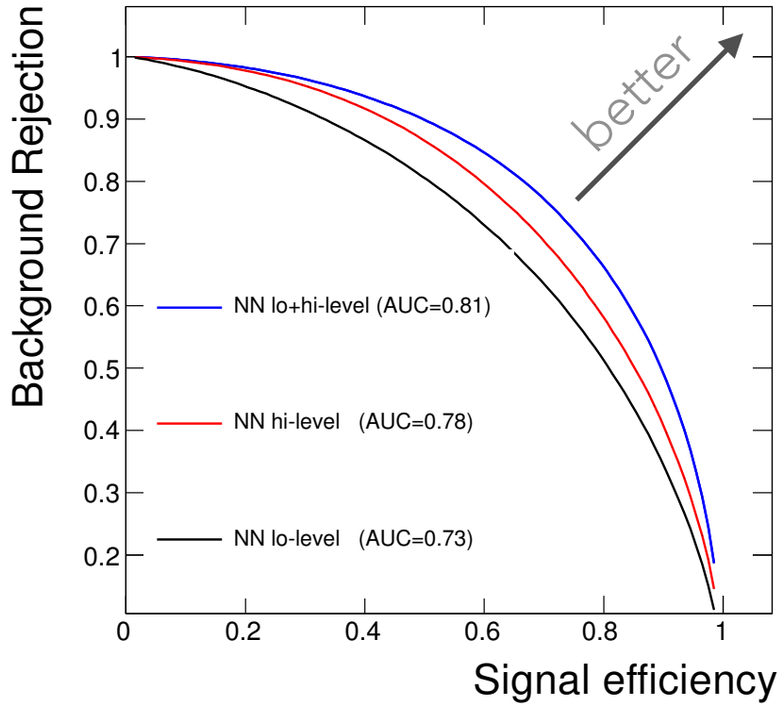


		True condition			
	Total population	Condition positive	Condition negative	Prevalence = $\frac{\Sigma \text{ Condition positive}}{\Sigma \text{ Total population}}$	Accuracy (ACC) = $\frac{\Sigma \text{ True positive} + \Sigma \text{ True negative}}{\Sigma \text{ Total population}}$
Predicted condition	Predicted condition positive	**True positive**	**False positive**, Type I error	Positive predictive value (PPV), Precision = $\frac{\Sigma \text{ True positive}}{\Sigma \text{ Predicted condition positive}}$	False discovery rate (FDR) = $\frac{\Sigma \text{ False positive}}{\Sigma \text{ Predicted condition positive}}$
	Predicted condition negative	**False negative**, Type II error	**True negative**	False omission rate (FOR) = $\frac{\Sigma \text{ False negative}}{\Sigma \text{ Predicted condition negative}}$	Negative predictive value (NPV) = $\frac{\Sigma \text{ True negative}}{\Sigma \text{ Predicted condition negative}}$
		True positive rate (TPR), Recall, Sensitivity, probability of detection, Power = $\frac{\Sigma \text{ True positive}}{\Sigma \text{ Condition positive}}$	False positive rate (FPR), Fall-out, probability of false alarm = $\frac{\Sigma \text{ False positive}}{\Sigma \text{ Condition negative}}$	Positive likelihood ratio (LR+) = $\frac{\text{TPR}}{\text{FPR}}$	Diagnostic odds ratio (DOR) = $\frac{\text{LR+}}{\text{LR−}}$
		False negative rate (FNR), Miss rate = $\frac{\Sigma \text{ False negative}}{\Sigma \text{ Condition positive}}$	Specificity (SPC), Selectivity, True negative rate (TNR) = $\frac{\Sigma \text{ True negative}}{\Sigma \text{ Condition negative}}$	Negative likelihood ratio (LR−) = $\frac{\text{FNR}}{\text{TNR}}$	F_1 score = $2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$

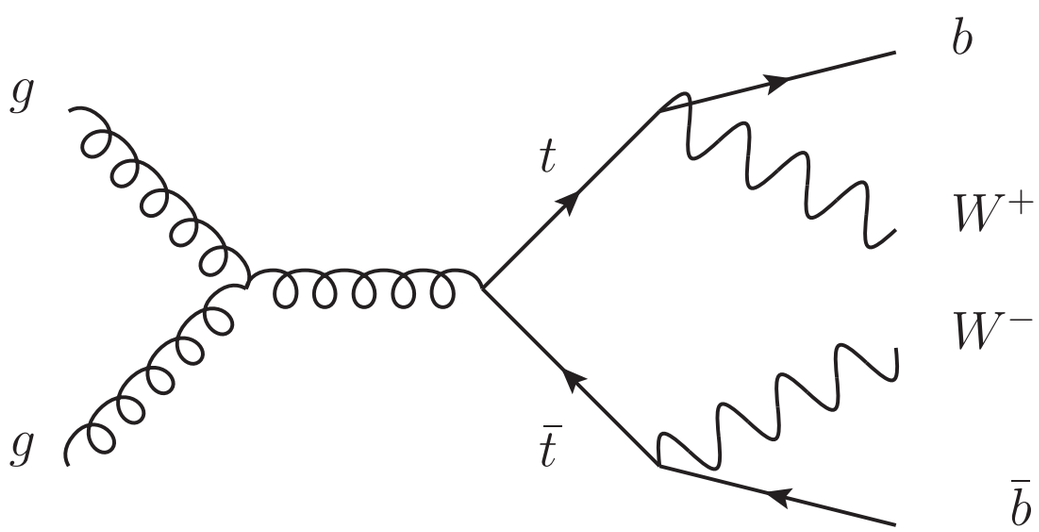
DEEP LEARNING IN HEP

Baldi, Sadowski,Whiteson
arxiv:1402.4735

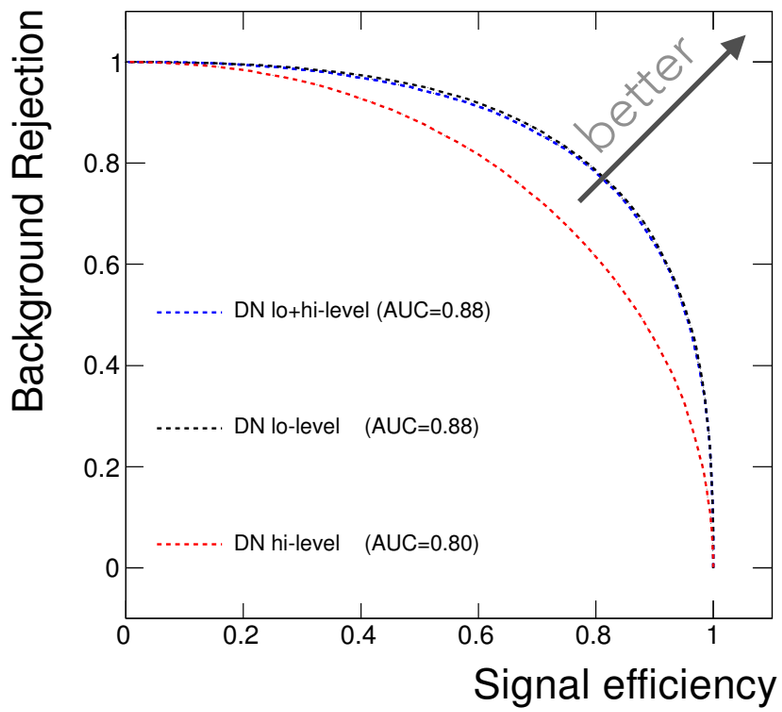




(a)



(b)



Slide from Kyle Cranmer

Data Representation

- Data are stored in “tensors”.
  - Basically an N- Dimensional Array with a “shape”
    - shape = (): Scalar
    - shape = (N,): Vector
    - shape = (N,M): Matrix
    - shape = (N_1, N_2, N_3, …, N_R): Rank R Tensor
  - Inputs: **X**
    - Can be arbitrary shape. Typically first dimension is the example index (usually an “event” or collision in HEP)
    - Example: Let’s say your examples are students, and your data is their age, sex, years at University, undergrad/grad, and department
      - X = [[20, 0, 2, 0, 4] , # 20 year old, 0=male, 2=junior, 0=undergrad, 4=computer science

            [25, 1, 2, 1, 3] , # 25 year old, 1=female, 2=3nd year, 0=grad, 4=physics

            [23, 0, 0, 1, 3]] # 25 year old, 1=make, 2=1st year, 0=grad, 4=physics
      - X[0] = [20, 0, 2, 0, 4]: the first students data.
      - X[0][3] = 1. This is a graduate student
- Outputs : **Y**
  - Can be arbitrary shape. Typically first dimension is the example index (usually an “event” or collision in HEP)
  - Example: Y = 0/1, student does not / does know python

Machine Learning Problem Formulation

- Split *Datasets*:
  - ($\mathbf{X}_{train}$, $\mathbf{Y}_{train}$) = training dataset
  - ($\mathbf{X}_{test}$, $\mathbf{Y}_{test}$) = test dataset
  - ($\mathbf{X}$) = unlabeled data
- Set *Goal*:
  - *Inference* algorithm/function F($\mathbf{X} \mid \boldsymbol{\alpha}$) = $\mathbf{Y}_{predict}$.
    - F can be a heuristic. e.g. if (computer science student) then (student knows python).
    - F can be anything
  - $\boldsymbol{\alpha}$ are parameters of the function, for Neural Networks, these are weights.
    - Note that in a simple classification problem, $\mathbf{Y}_{train}$ can be 0 or 1 for any example. But $\mathbf{Y}_{predict}$ will usually be between 0 and 1.
- *Training*: (for Neural Networks)
  - Optimize (usually a minimization) a *cost function* F($\mathbf{X} \mid \boldsymbol{\alpha}$) = C(F($\mathbf{X}_{train} \mid \boldsymbol{\alpha}$), $\mathbf{Y}_{train}$) w.r.t. $\boldsymbol{\alpha}$
  - For example, C = [F($\mathbf{X} \mid \boldsymbol{\alpha}$) - $\mathbf{Y}_{train}$]^2
  - $\boldsymbol{\alpha}_{trained}$= result of training
- Validation:
  - Compute cost function on test data C(F($\mathbf{X}_{test} \mid \boldsymbol{\alpha}_{trained}$), $\mathbf{Y}_{test}$)
  - Other metrics. For example:
    - Select $\mathbf{Y}_{test}$=1 and see how often F($\mathbf{X}_{test} \mid \boldsymbol{\alpha}_{trained}$) > 0.5
- Inference:
  - $\mathbf{Y}_{predict}$ = F($\mathbf{X} \mid \boldsymbol{\alpha}_{trained}$)