

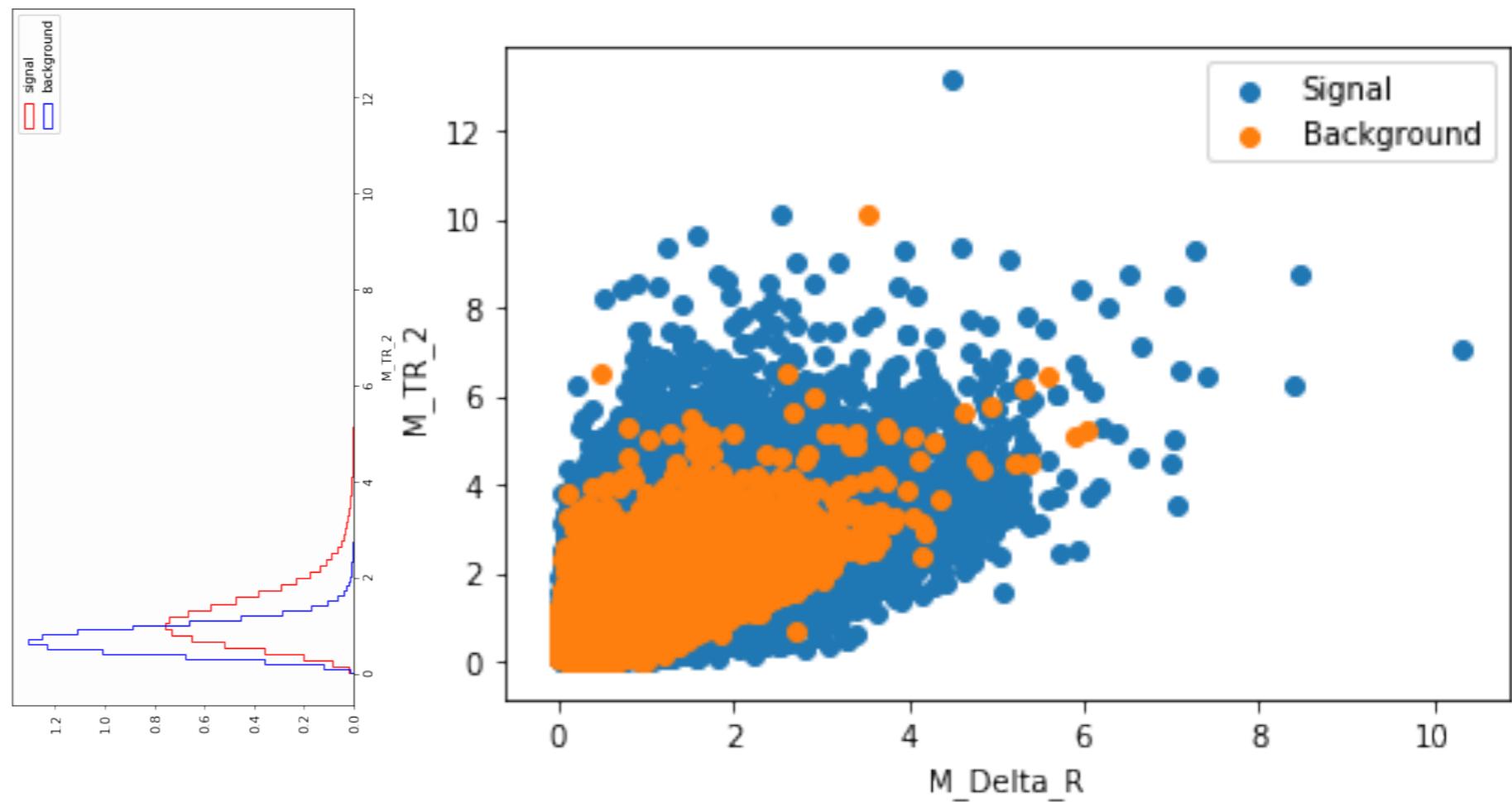
Python for Data Science 2

Lecture 15
- Machine Learning

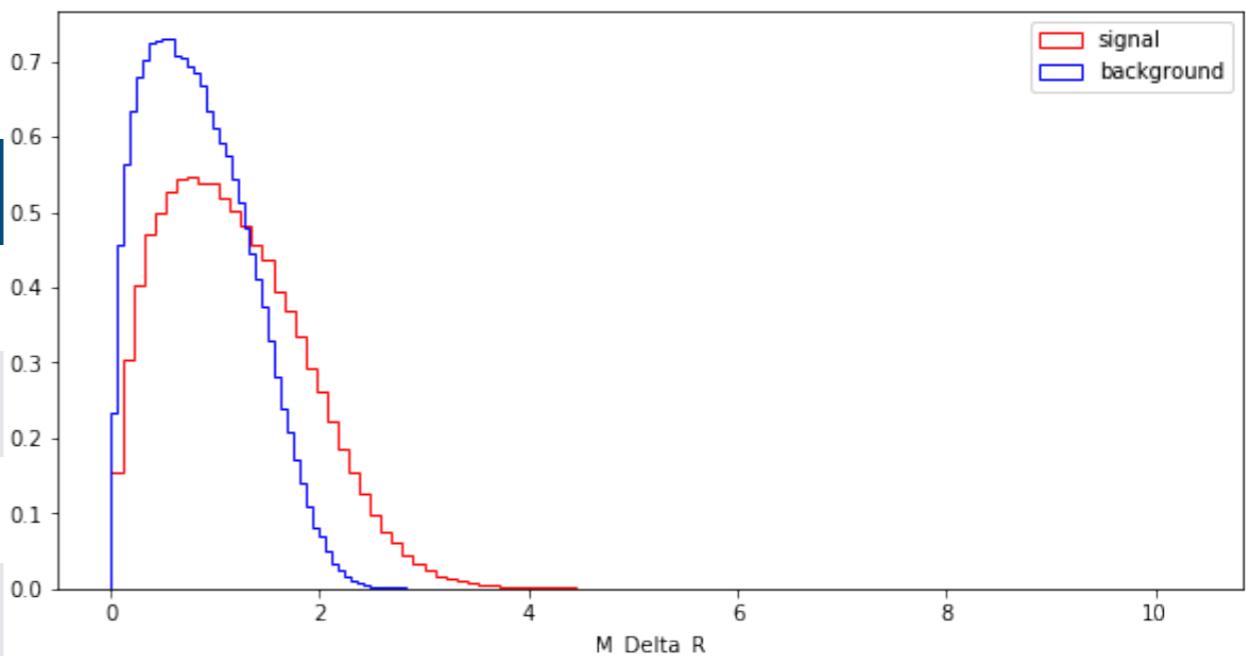
Amir Farbin

Test Statistic

- In your lab you are investigating different observables (aka features) $\{\vec{x}_i\}$
 - Each has separation power between signal and background.
 - Some are better than others
 - We can refer to any of these observables as a **test statistic**
 - A statistical variable that we can use to perform a test (i.e. is it signal?).
 - Many were created after years of careful “feature engineering”.
 - Many are very correlated → Capturing similar information
 - But not 100% correlated → Capture unique information
 - Suggests combining selections on multiple observables might be good idea.
 - → Difficult to do... lose statistical power.
- Can we find a single observable that optimally combines all of the observable?
 - Single test statistic $f(\{\vec{x}_i\}) \equiv t_i$



	signal	background	S/B
$M_{TR_2 > 2}$	0.114	0.006	19
$M_{\Delta R > 2}$	0.133	0.014	9.5
Product	0.015	0.000084	179
Combined	0.051	0.002	25.5



Hypothesis Testing

- Neyman-Pearson lemma: best possible test statistic, given a set of observables is
- But it is rather difficult to obtain P .
 - Unless you know it analytically, you'll have to rely on Data with ground truth or Simulation (ground truth inputed)
 - → Build Probability Density Function (high dimensional histogram)
 - Curse of Dimensionality → very difficult to populate a high dimensional histogram.
 - Alternatives:
 - Feature Engineering - That's what was attempted with the observables you are using.
 - Machine Learning - Algorithm that learn an optimal test statistic for your task.

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:** $F(\text{Input} \mid \text{Parameters}) = \text{Output}$
 - **Target: Cost (aka loss) function**
 - *Supervised:* Compare F vs Ground Truth Output
 - *Unsupervised:* e.g. Cluster like inputs
 - *Semi-supervised:* $F(\text{Input}) = \text{Input}$
 - **Training: Optimization**
 - Choose how to find best parameters

Machine Learning

Basic Formulation

- **AI Algorithm:** $\text{Input} \rightarrow \text{AI Algorithm} \rightarrow \text{Output}$
 - $X \rightarrow F(\theta) \rightarrow y: y = F(X|\theta)$
 - 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 *Validation* subsets
 - **Choose:** Pick an ML algorithm F
 - **Train:** Use labeled Dataset $\{x_d, c\}_i$ to obtain θ
 - **Validate:** Estimate $P(F(X|\theta)|c_i)$ using Test sample
 - **Optimize:** Select $F(X|\theta) > \text{cut}$ to optimize sensitivity for a hypothesis test
 - $P(F(X|\theta)| \text{signal}) / P(F(X|\theta)| \text{background})$
 - **Test:** 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 $\frac{1}{N} \sum_i^N (F(\vec{x}_i | \mathbf{W}, \vec{b}) - \vec{y}_i)^2$

- 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

- \Rightarrow Analytic solution

$$W = (X^T X)^{-1} X^T Y$$

$$X = \begin{pmatrix} \vec{x}_1 \\ \vec{x}_2 \\ \vdots \\ \vec{x}_N \end{pmatrix} \quad Y = \begin{pmatrix} \vec{y}_1 \\ \vec{y}_2 \\ \vdots \\ \vec{y}_N \end{pmatrix}$$

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:
$$S = \frac{(\mathbf{W} \cdot \vec{\mu}_0 - \mathbf{W} \cdot \vec{\mu}_1)^2}{\mathbf{W}^T \Sigma_0 \mathbf{W} - \mathbf{W}^T \Sigma_1 \mathbf{W}}$$
 - Maximal separation between means
 - Smallest possible variance
 - Analytic Solution: $\mathbf{W} = (\Sigma_0 + \Sigma_1)^{-1} (\vec{\mu}_0 - \vec{\mu}_1)$

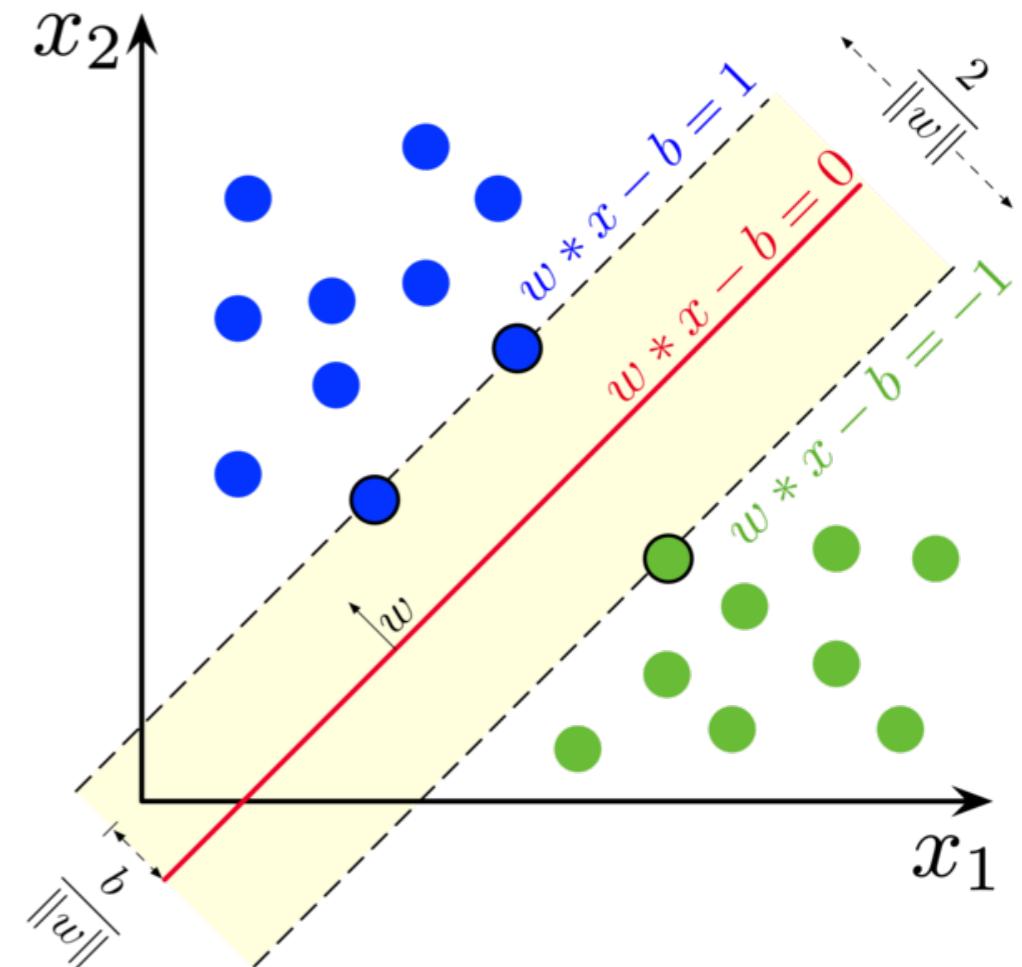
ML Models (Linear SVM)

- 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

$$f(\vec{x}) = \left(\sum_i^N a_i y_i \vec{x}_i \right) \cdot \vec{x} + \vec{b}$$

- Where $y_i \in \{+1, -1\}$ 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

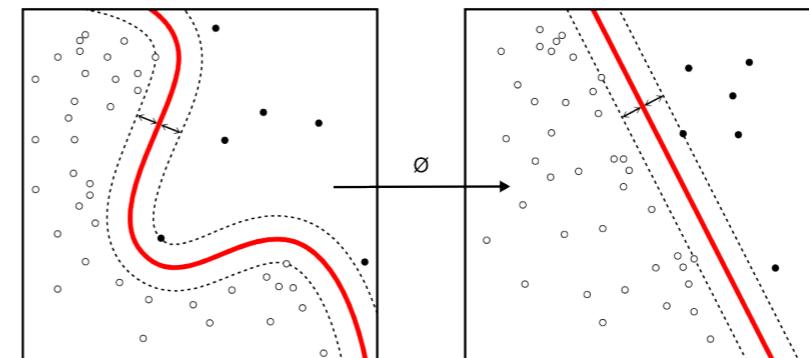


ML Models (Kernel SVM)

- If boundary not linear \Rightarrow Kernel Trick:

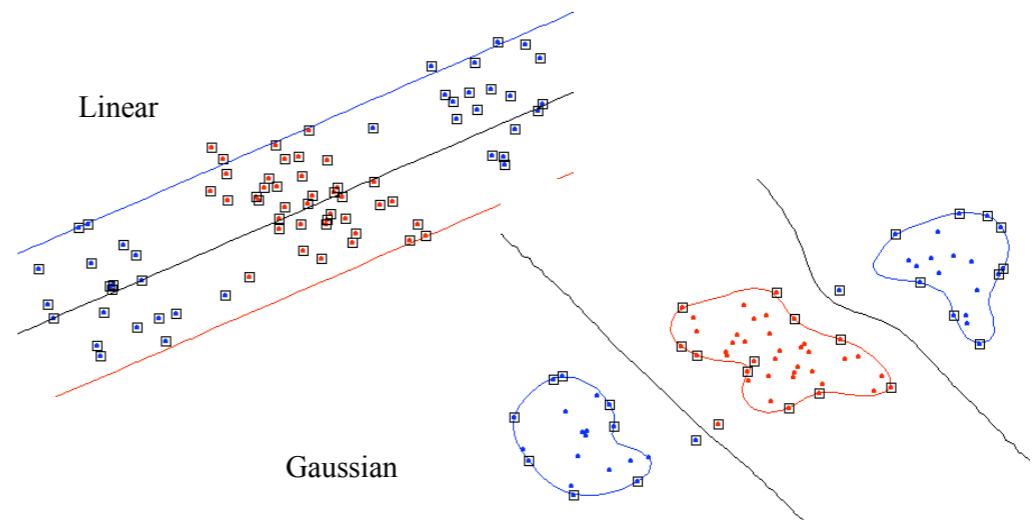
- Replace dot product: $f(\vec{x}) = \sum_i^N a_i y_i k(\vec{x}_i, \vec{x}) + \vec{b}$ $k(\vec{x}_i, \vec{x}) = \phi(\vec{x}_i) \cdot \phi(\vec{x})$

- Example: if x is in radial coordinates, then kernel gets back proper dot product.

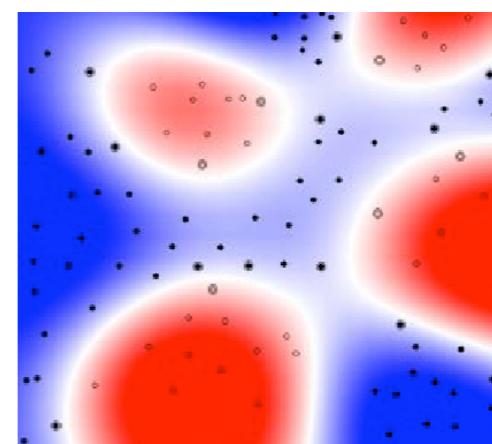


- Example: Gaussian Kernels

- Euclidian Distance in the exponential
- If close to an example in training data \Rightarrow large value
- Equivalent to template matching



Radial Basis Functions

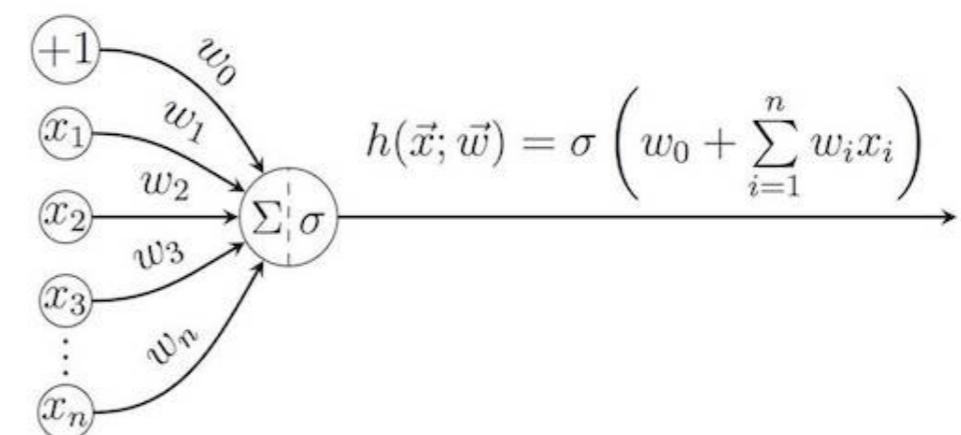
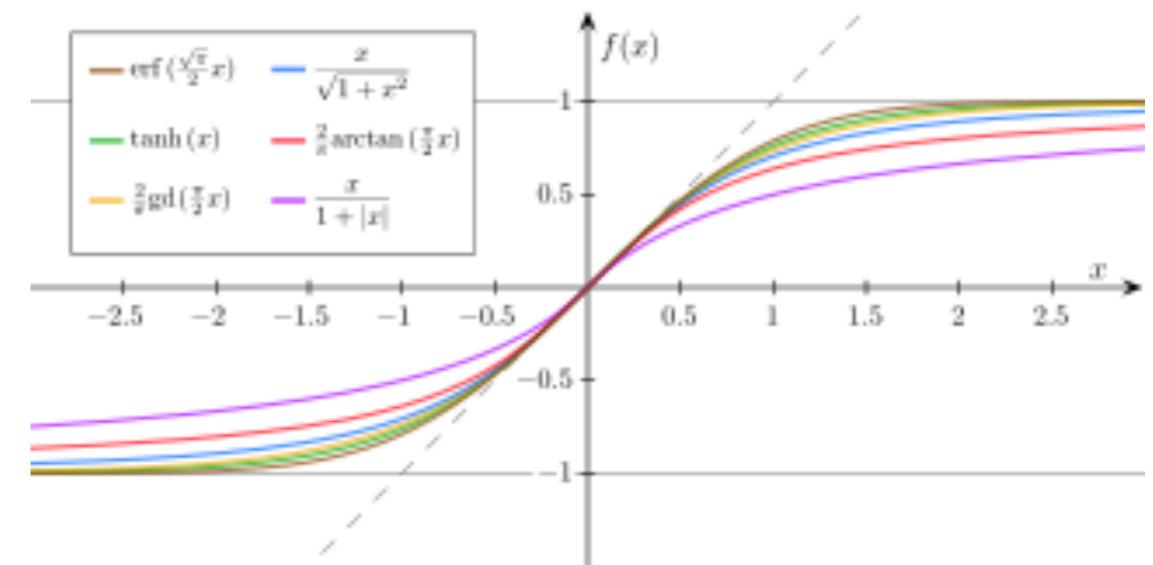


Features

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

Artificial Neural Network

- A simple one layer NN
 - $F(\mathbf{X} | \mathbf{a} = \mathbf{W}, \mathbf{b}) = f(\mathbf{W} \mathbf{X} + \mathbf{b})$
 - \mathbf{W}, \mathbf{b} = “weights”, “biases”
 - $f(x)$ = “activation function”
 - Must be non-linear.
 - Universal Computation Theorem.



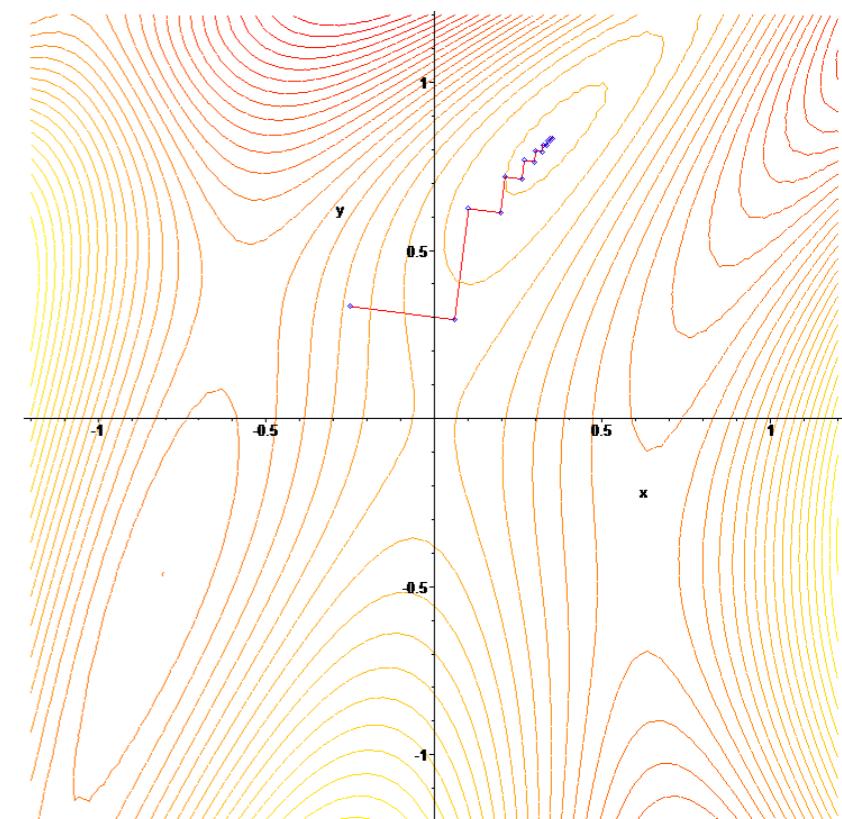
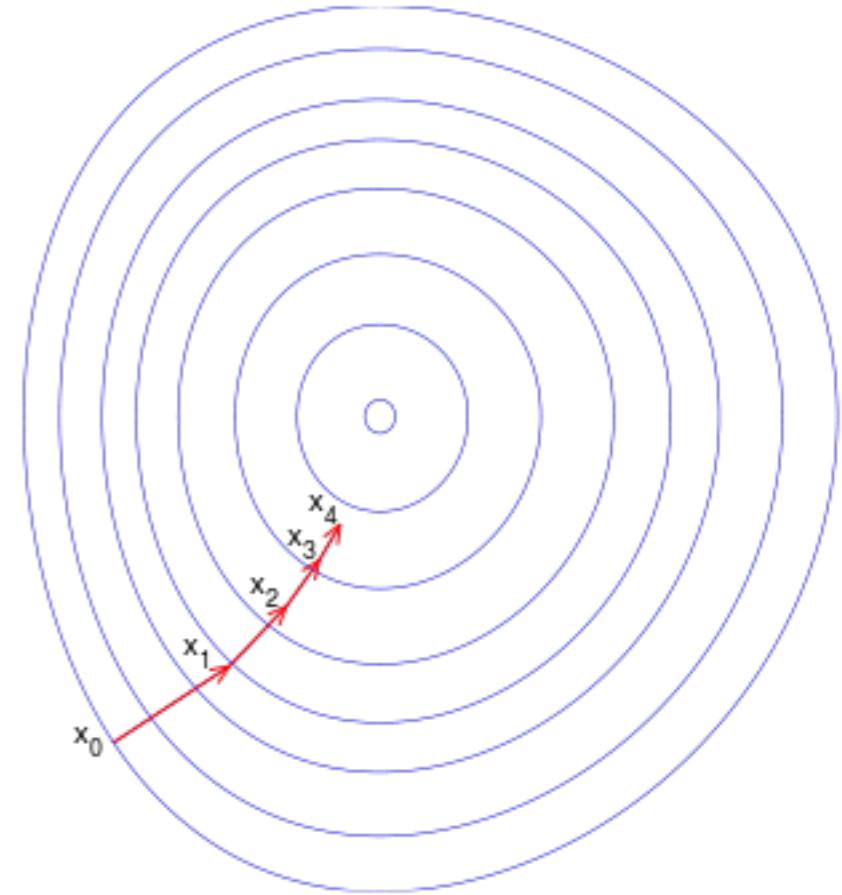
Training == Optimization

- Training = Minimizing cost function w.r.t. parameters $\vec{\alpha}$

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

- Gradient Decent (Newton's Method):
 - Gradient points to direction of maximal change.
 - Iterate (ϵ sets the step size == *Learning Rate*)

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

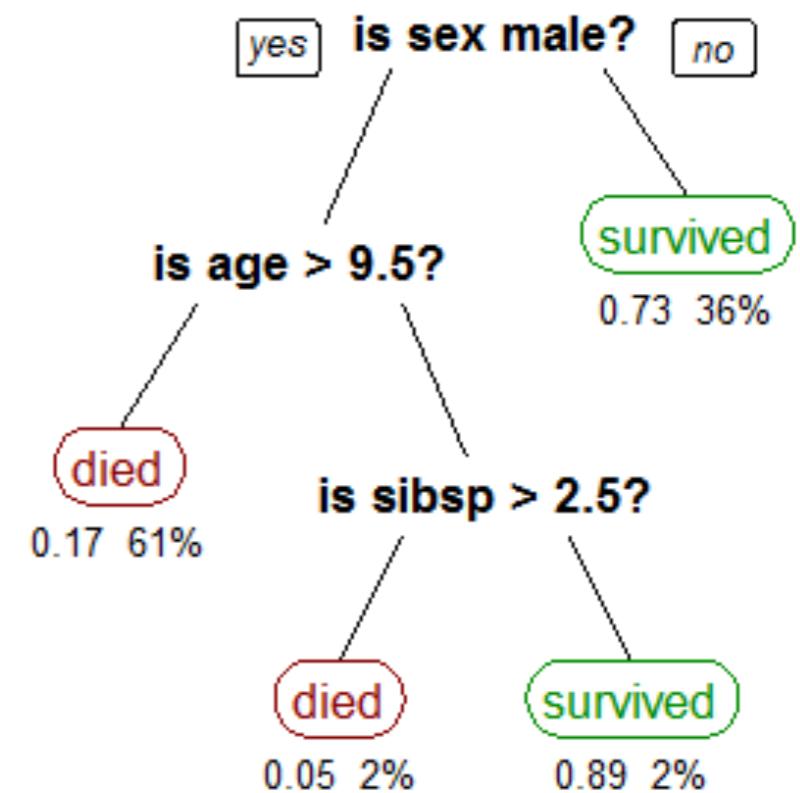


Bayesian vs Frequentist

- ***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 \Rightarrow learn μ, σ
 - Allows you to propagate uncertainties \Rightarrow estimate uncertainty on output Y

Decision Trees

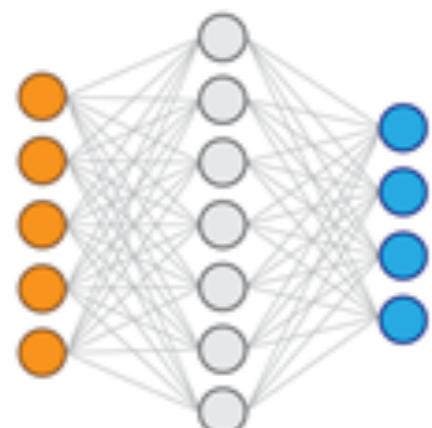
- Doesn't fit the $F(\mathbf{X} | \mathbf{W}, \mathbf{b})$ formulation.
- Powerful technique
 - Random Forest
 - Boosting
- Up to recently, in HEP, Boost Decision Trees (BDTs) on well constructed/chosen features
 - have been the best performing techniques, and
 - become the standard to beat,



Titanic Survivors

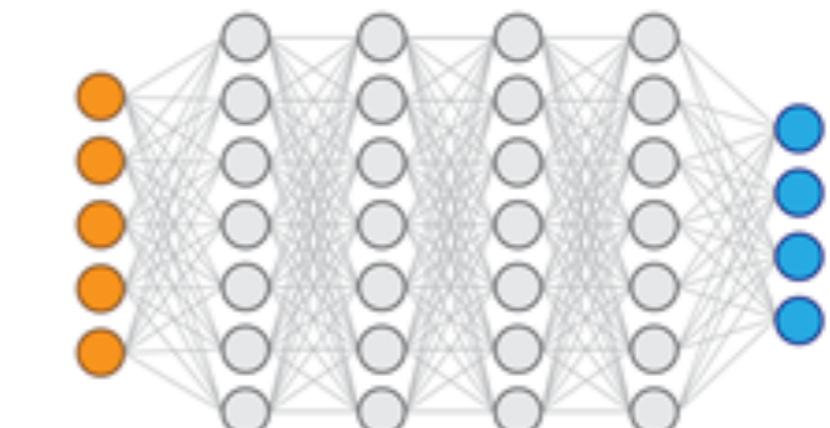
Deep Learning

shallow neural network



input layer hidden layer output layer

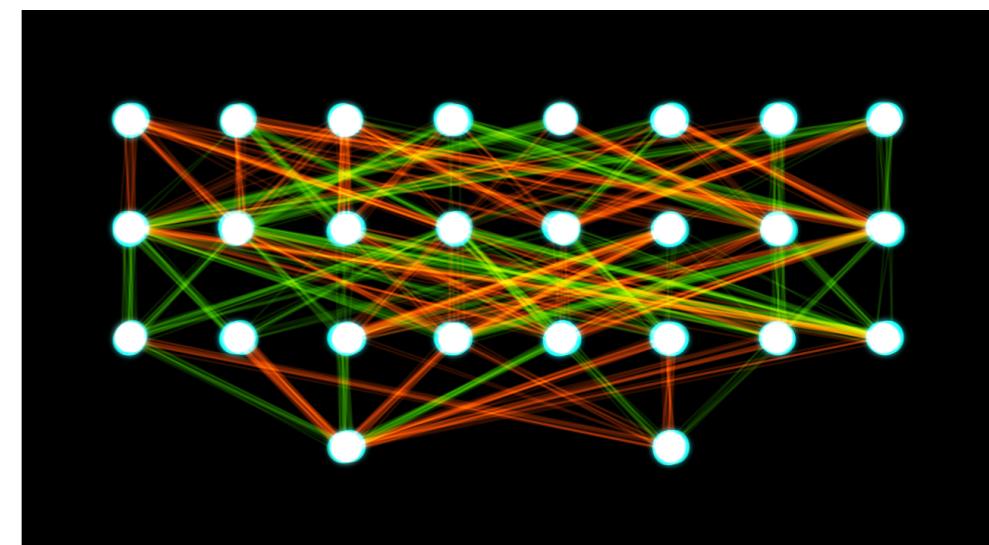
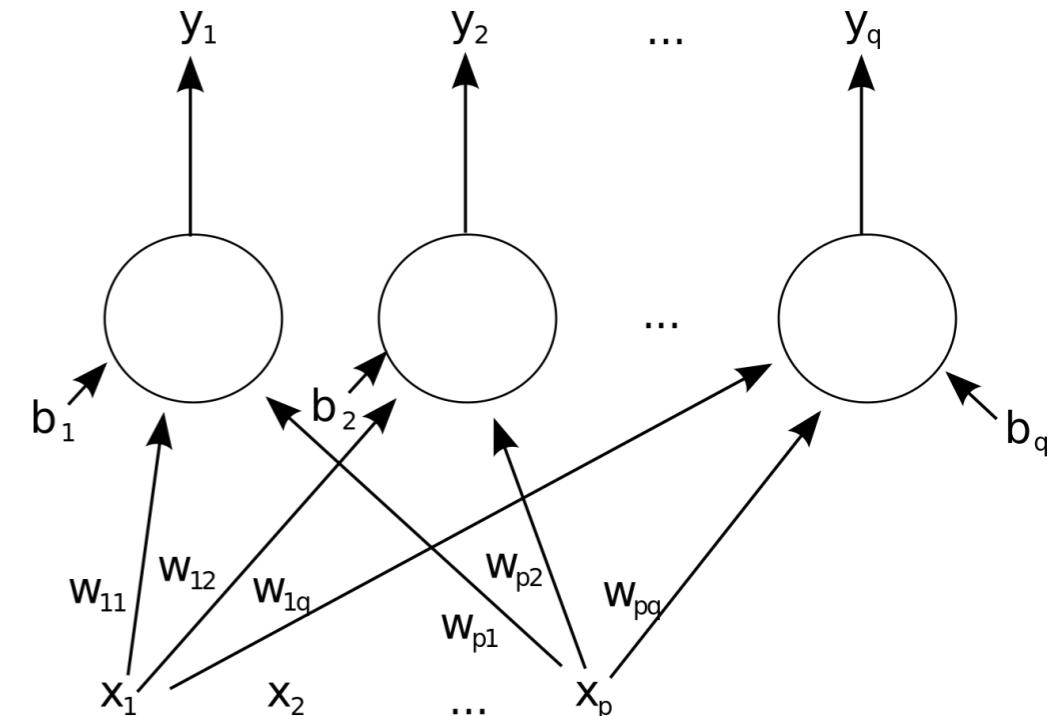
deep neural network



input layer hidden layers output layer

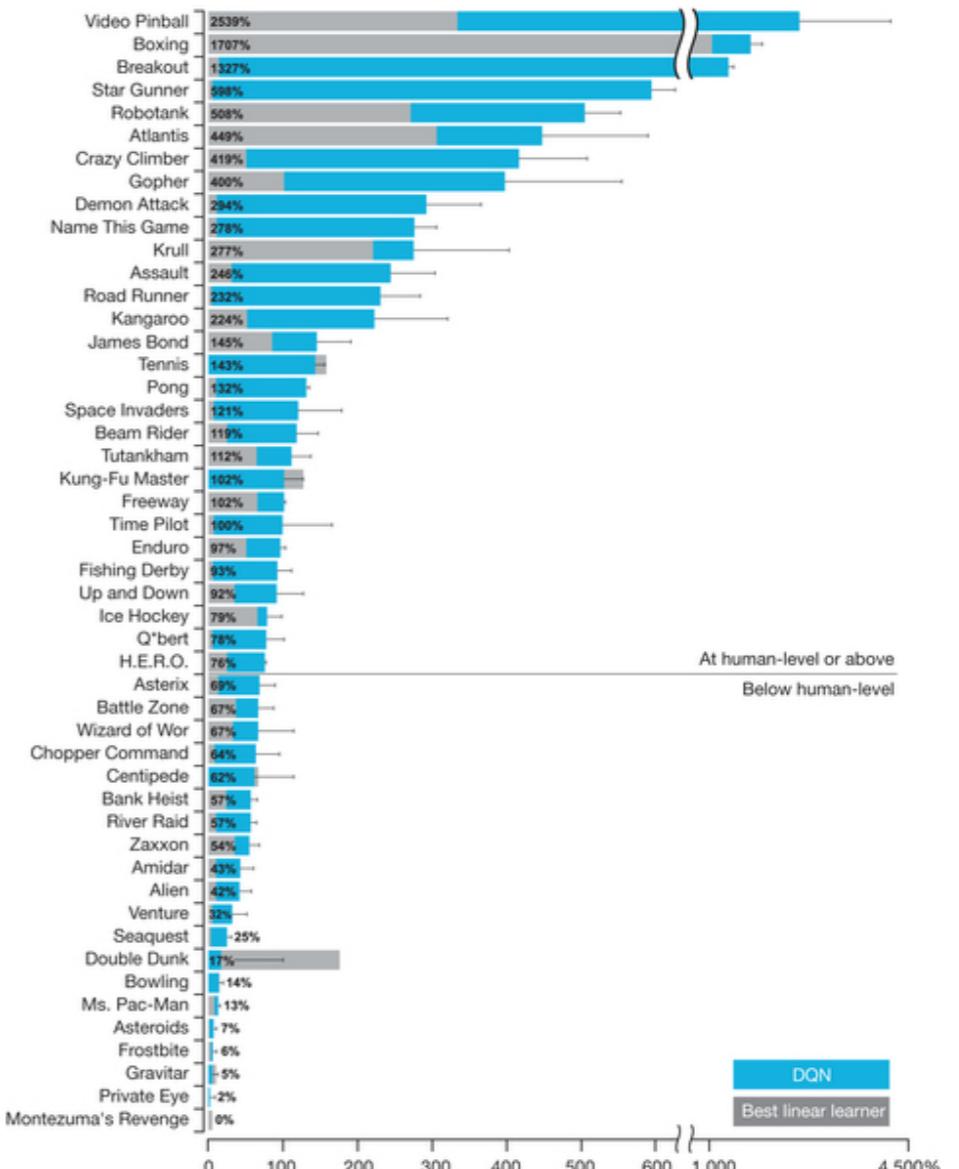
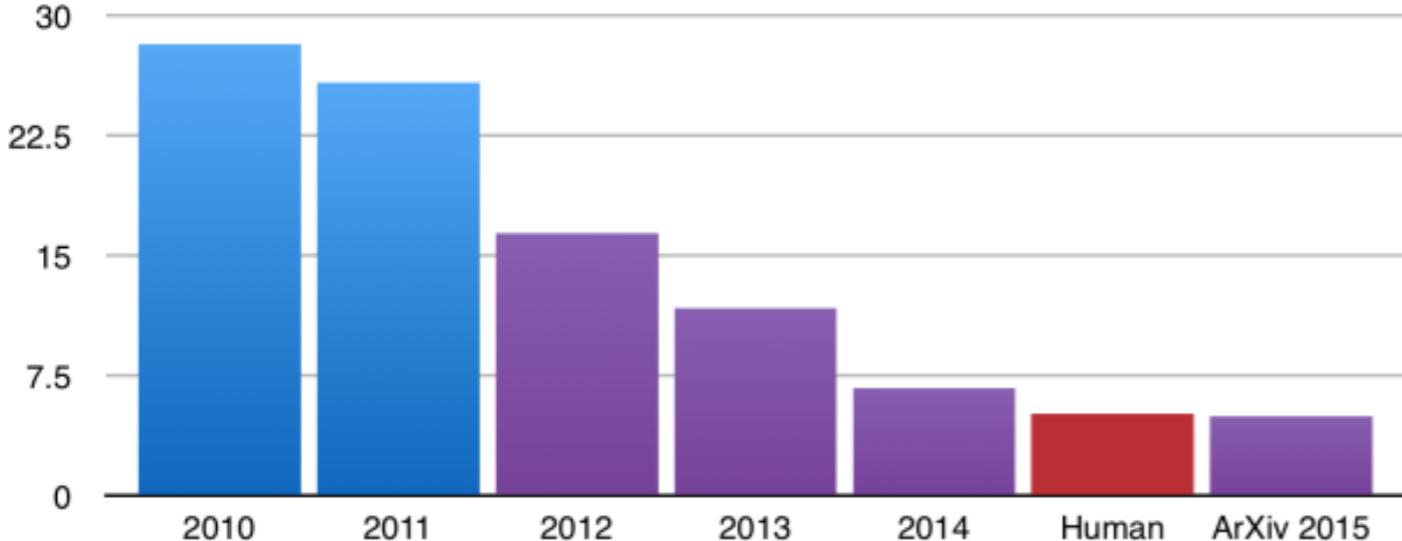
Artificial Neural Networks

- **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.

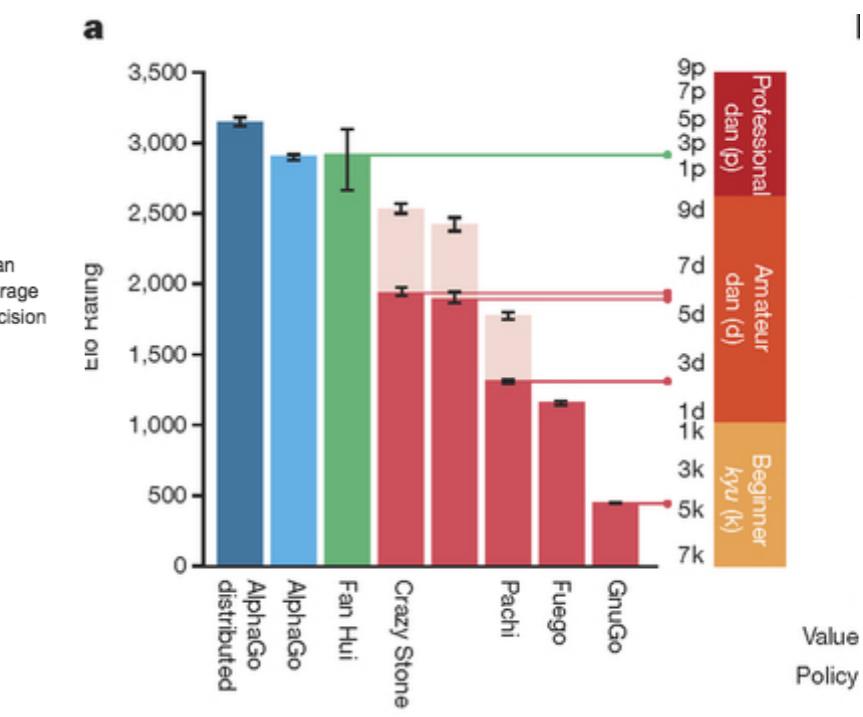
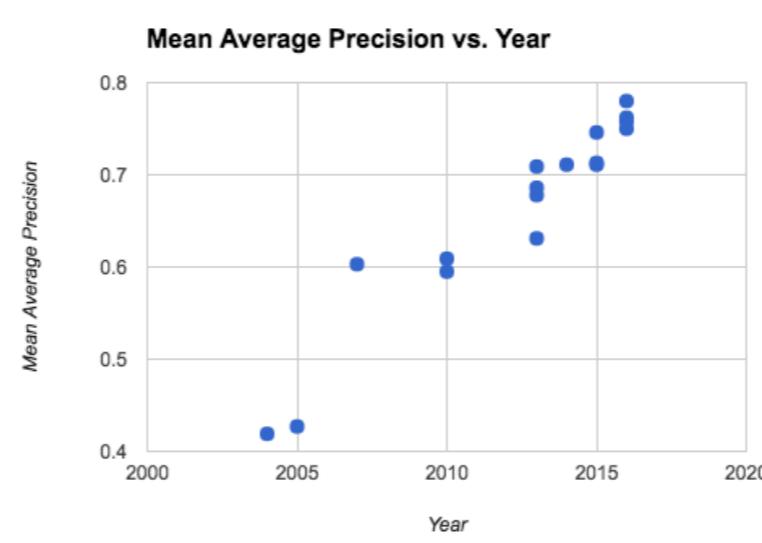
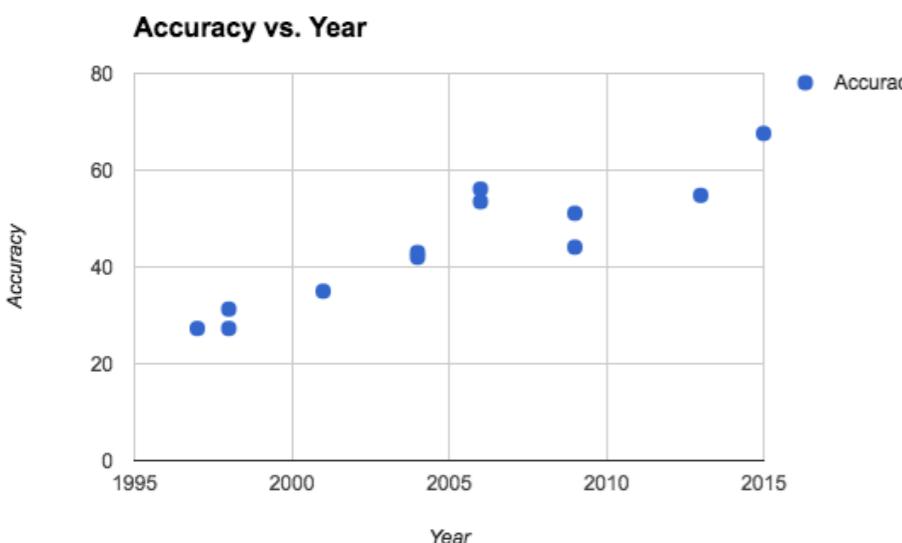
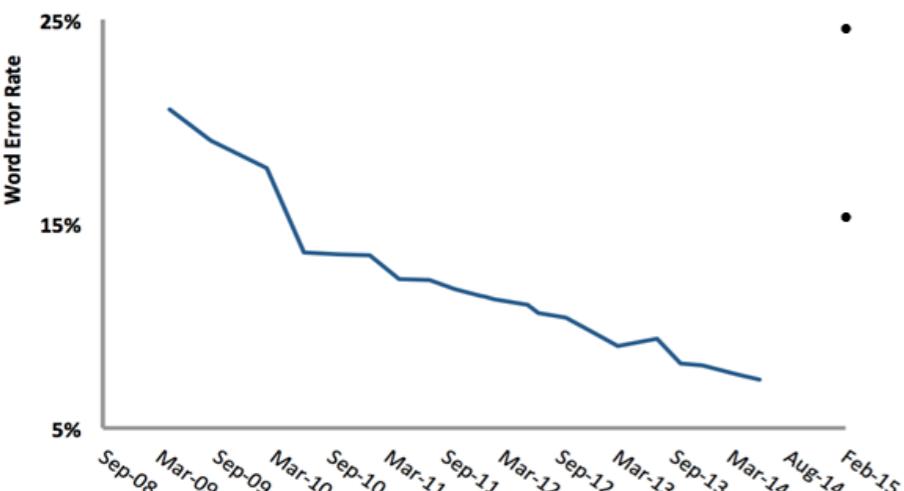


Images from Wikipedia

ILSVRC top-5 error on ImageNet

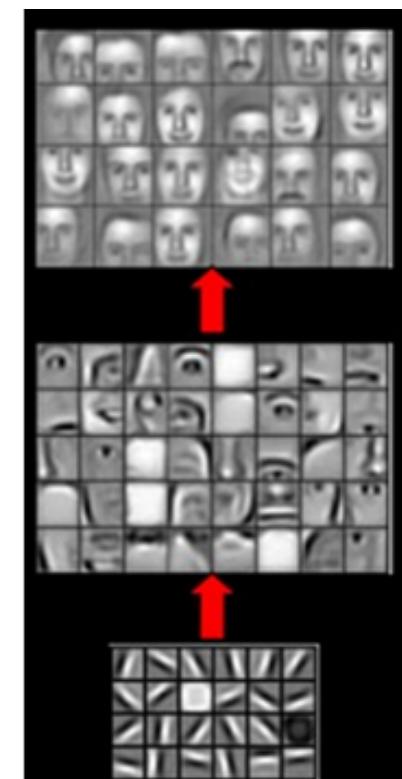
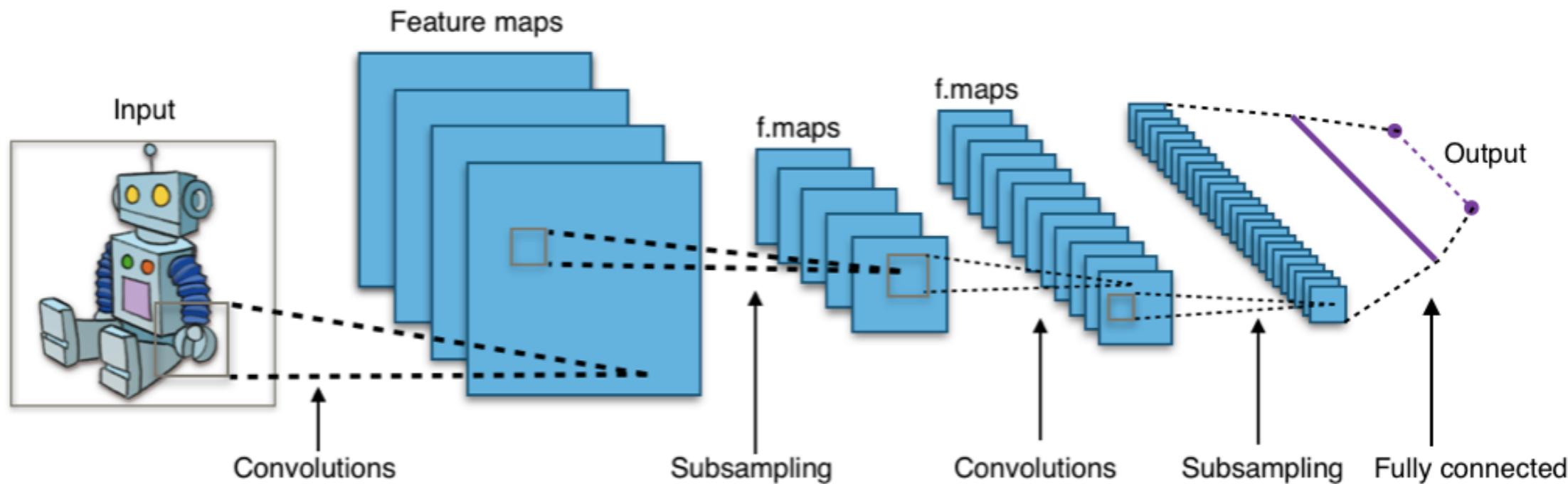


- Continuous server ASR word error rate (WER) reduction ~18% / year: combination of algorithms, data, and computing
- Deep learning (DNNs) is driving recent performance improvements in ASR and meaning extraction

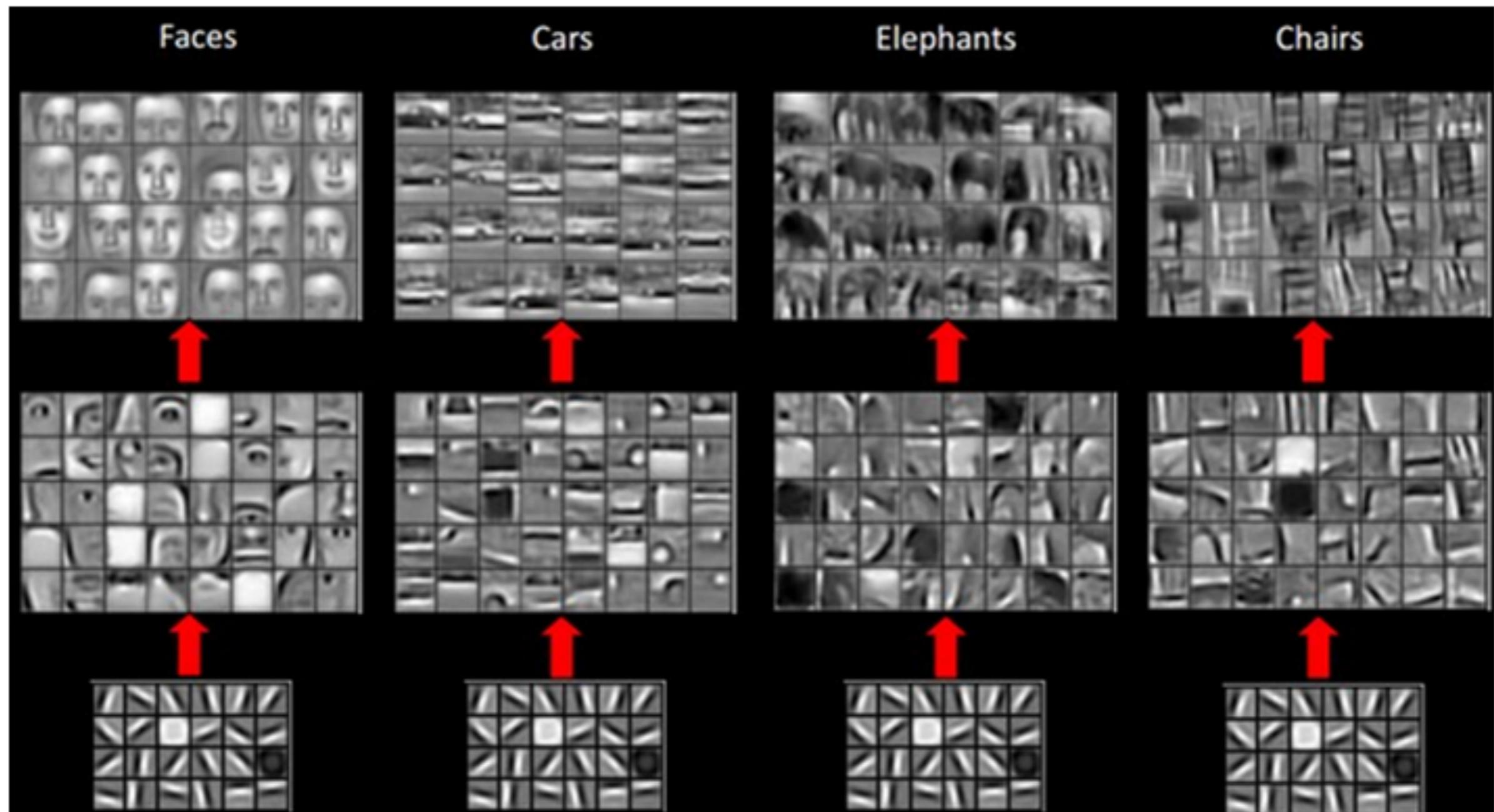


Feature Learning

- **Feature Engineering:** e.g. Event Reconstruction ~ Feature Extraction, Pattern Recognition, Fitting, ...
- Deep Neural 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.

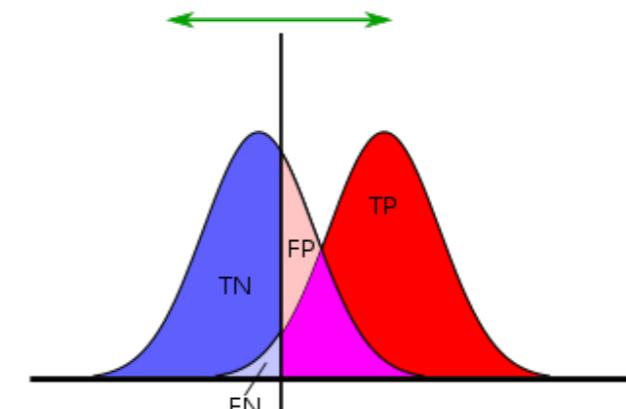
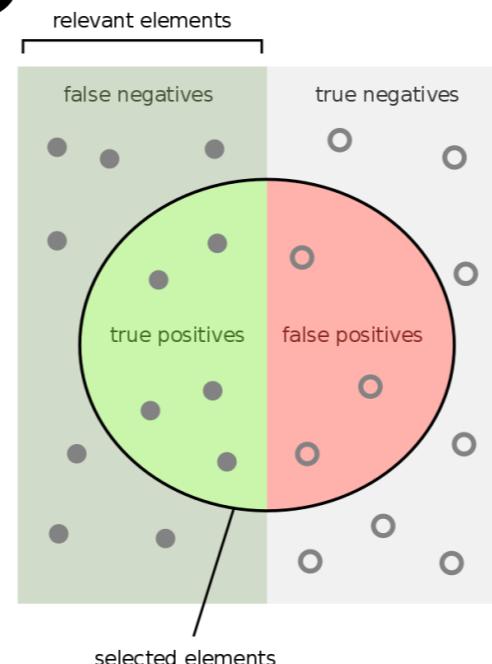


Deep Neutral Networks

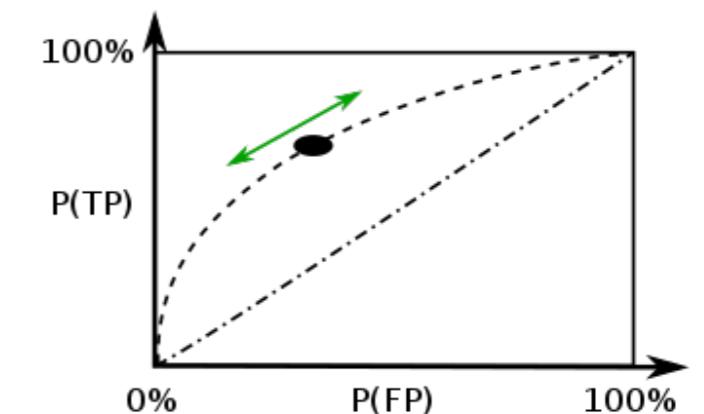


Assessing Performance

- **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



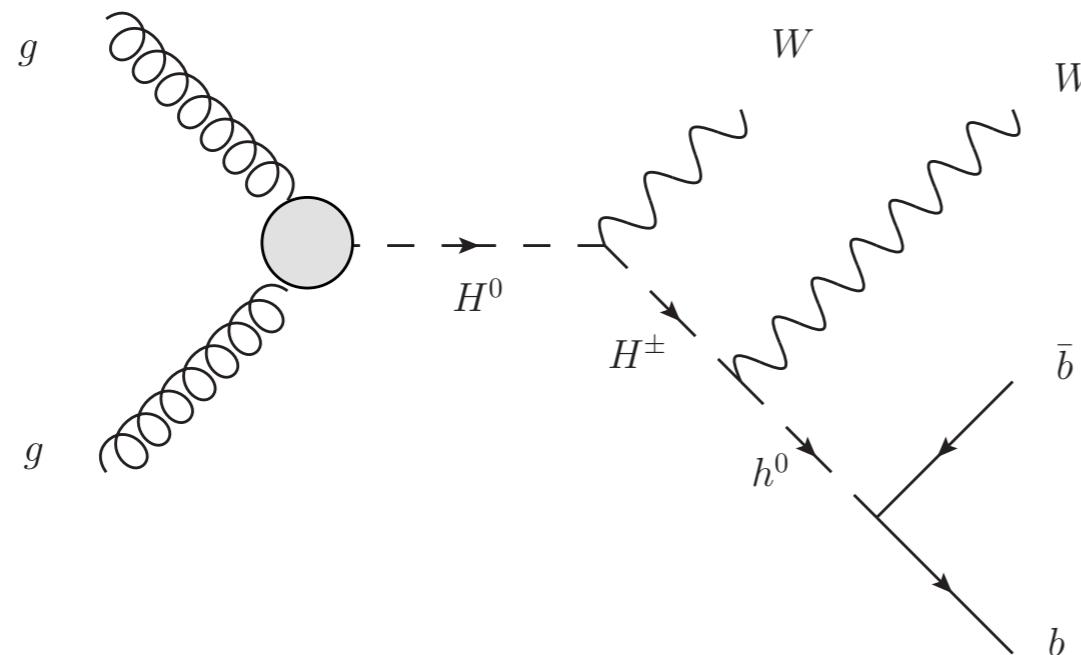
TP	FP
FN	TN



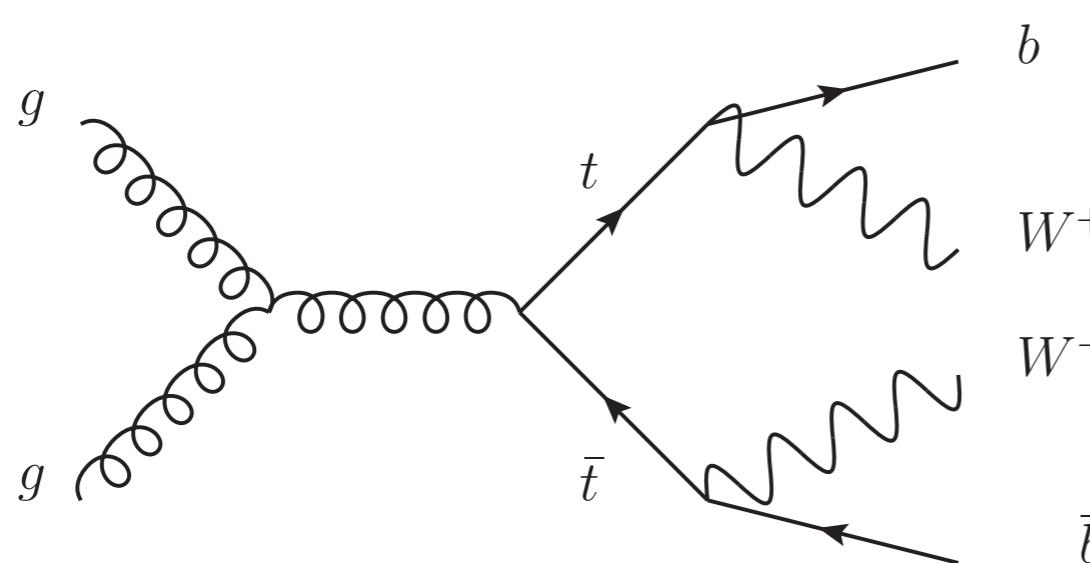
		True condition		Prevalence = $\frac{\sum \text{Condition positive}}{\sum \text{Total population}}$	Accuracy (ACC) = $\frac{\sum \text{True positive} + \sum \text{True negative}}{\sum \text{Total population}}$
		Condition positive	Condition negative		
Predicted condition	Predicted condition positive	True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

DEEP LEARNING IN HEP

Baldi, Sadowski, Whiteson
arxiv:1402.4735

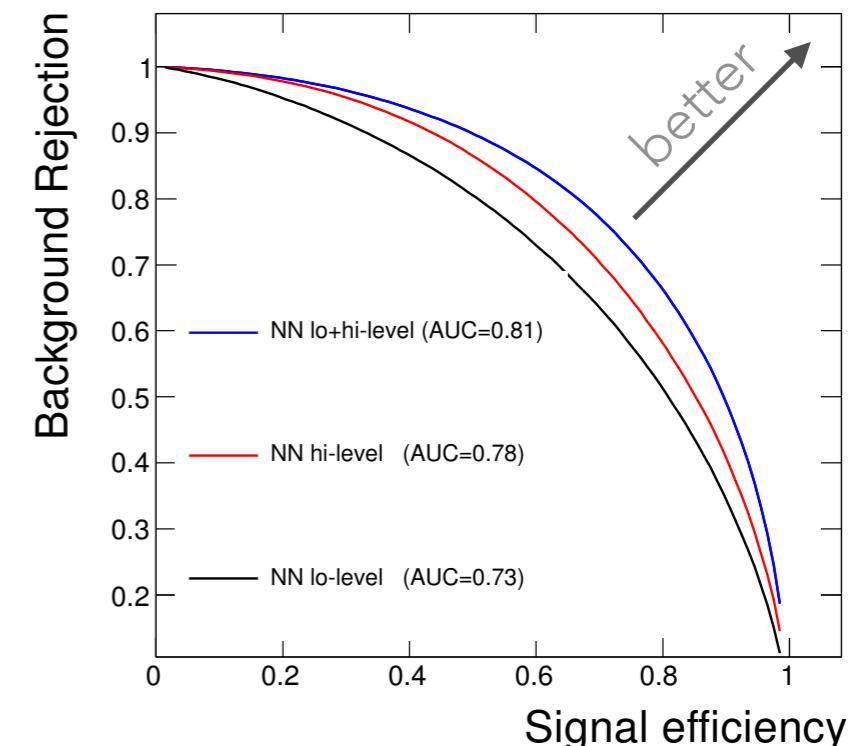


(a)

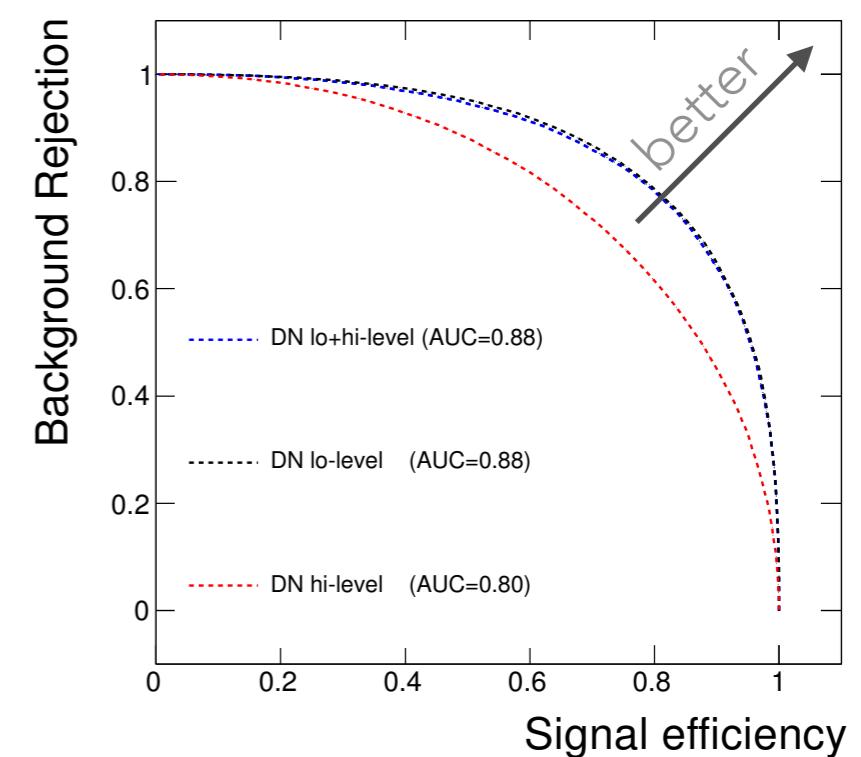


(b)

Slide from Kyle Cranmer



(a)



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 = (N₁, N₂, N₃, ..., 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=3rd 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 student’s 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*:
 - $(\mathbf{X}_{\text{train}}, \mathbf{Y}_{\text{train}})$ = training dataset
 - $(\mathbf{X}_{\text{val}}, \mathbf{Y}_{\text{val}})$ = validation dataset
 - $(\mathbf{X}_{\text{test}}, \mathbf{Y}_{\text{test}})$ = test dataset
 - (\mathbf{X}) = unlabeled data
- Set *Goal*:
 - *Inference* algorithm/function $F(\mathbf{X} | \mathbf{a}) = \mathbf{Y}_{\text{predict}}$.
 - F can be a heuristic. e.g. if (computer science student) then (student knows python).
 - F can be anything
 - \mathbf{a} are parameters of the function, for Neural Networks, these are weights.
 - Note that in a simple classification problem, $\mathbf{Y}_{\text{train}}$ can be 0 or 1 for any example. But $\mathbf{Y}_{\text{predict}}$ will usually be between 0 and 1.
- *Training*: (for Neural Networks)
 - Optimize (usually a minimization) a *cost function* $F(\mathbf{X} | \mathbf{a}) = C(F(\mathbf{X}_{\text{train}} | \mathbf{a}), \mathbf{Y}_{\text{train}})$ w.r.t. \mathbf{a}
 - For example, $C = [F(\mathbf{X} | \mathbf{a}) - \mathbf{Y}_{\text{train}}]^2$
 - $\mathbf{a}_{\text{trained}}$ = result of training
- *Validation*:
 - Compute cost function on test data $C(F(\mathbf{X}_{\text{val}} | \mathbf{a}_{\text{trained}}), \mathbf{Y}_{\text{test}})$
 - Other metrics. For example:
 - Select $\mathbf{Y}_{\text{val}}=1$ and see how often $F(\mathbf{X}_{\text{val}} | \mathbf{a}_{\text{trained}}) > \text{threshold}$
 - Compute AUC, F1, ...
 - *Hyper-parameter Optimization*: Tune ML algorithm and retrain to obtain better performance
 - Select Threshold
- *Test*: Obtain values of metrics on $(\mathbf{X}_{\text{test}}, \mathbf{Y}_{\text{test}}) \Rightarrow$ Expected performance on data
 - Do not retune or change any parameter, including threshold(s)
- *Inference*: $\mathbf{Y}_{\text{predict}} = F(\mathbf{X} | \mathbf{a}_{\text{trained}})$