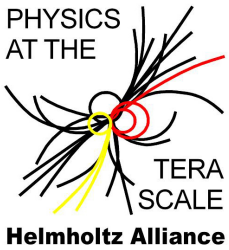


Introduction to Machine Learning

Harrison B. Prosper

Florida State University

Terascale Statistics School 2017

DESY, Hamburg, Germany

6 – 13 March, 2017

Outline

- Lecture 1
  - Introduction
  - Theory
  - Practice
    - Grid Searches
    - Fisher Discriminant
    - Boosted Decision Trees
- Lecture 2
    - Recap
    - Support Vector Machines
    - Neural Networks, Shallow, Deep, Bayesian

Recap

1. Given sufficient training data T, with labels $y = 1$ for objects of class C_1 and $y = 0$ for objects of class C_2, and
2. given a sufficiently flexible function $f(x, w)$, we can approximate

$$p(C_1 \mid x) = \frac{D(x)}{D(x) + [1 - D(x)]/A}, \qquad A = p(C_1) / p(C_2),$$

where

$$D(x) = \frac{p(x|C_1)}{p(x|C_1) + p(x|C_2)}$$

by *minimizing the quadratic risk*.

3. This conclusion is independent of the function class.

Recap

4. If your goal is to *classify* objects with the fewest errors, then the Bayes classifier, $p(C_1 \mid x)$, is the *optimal* solution.

5. Consequently, if you have a classifier known to be *close* to the Bayes limit (the smallest possible error rate) it is clear that any other classifier, however sophisticated it might be, cannot possibly do much better.

6. All classification methods, such as the ones in TMVA, are different numerical approximations of some function of the Bayes classifier.

SUPPORT VECTOR MACHINES

Support Vector Machines

In 1992, Boser, Guyon and Vapnik created an interesting non-linear generalization of the Fisher discriminant (SVM).

Basic Idea

Data that are non-separable in d-dimensions may be better separated if mapped into a space of higher (usually, infinite) dimension

$$h\colon \mathbb{R}^d \to \mathbb{R}^\infty$$

Instead of constructing a plane in the original space $\mathbb{R}^d$ one constructs a plane

$$f(x) = w \cdot h(x) + c$$

in the space $\mathbb{R}^\infty$ that partitions the space into signal and background rich regions.

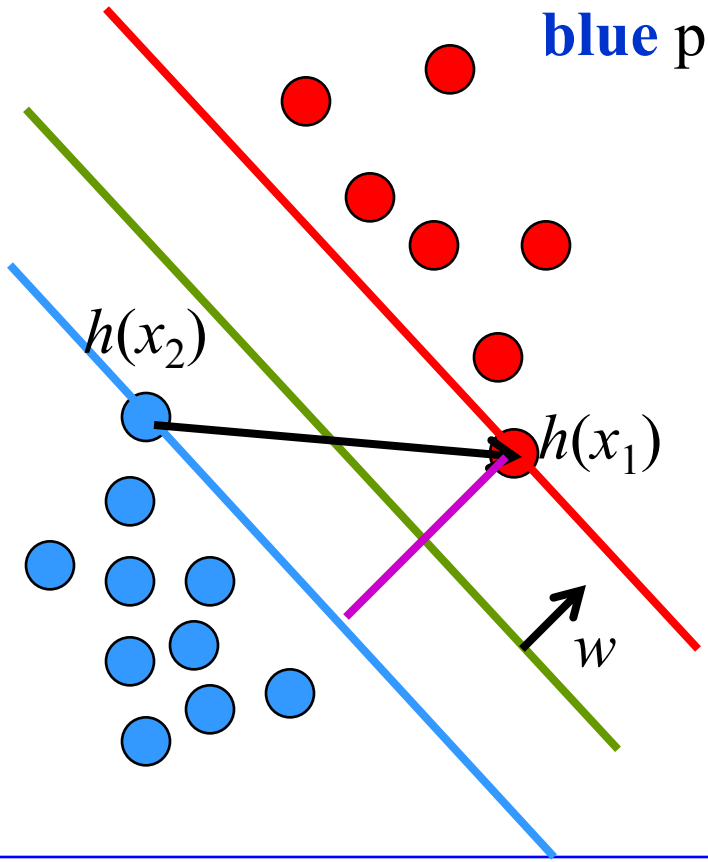
Support Vector Machines

Consider *separable* data in $\mathbb{R}^{\infty}$:

red plane: $w.h(x_1) + c = +1$

green plane: $w.h(x) + c = 0$

blue plane: $w.h(x_2) + c = -1$



subtract **blue** from **red**

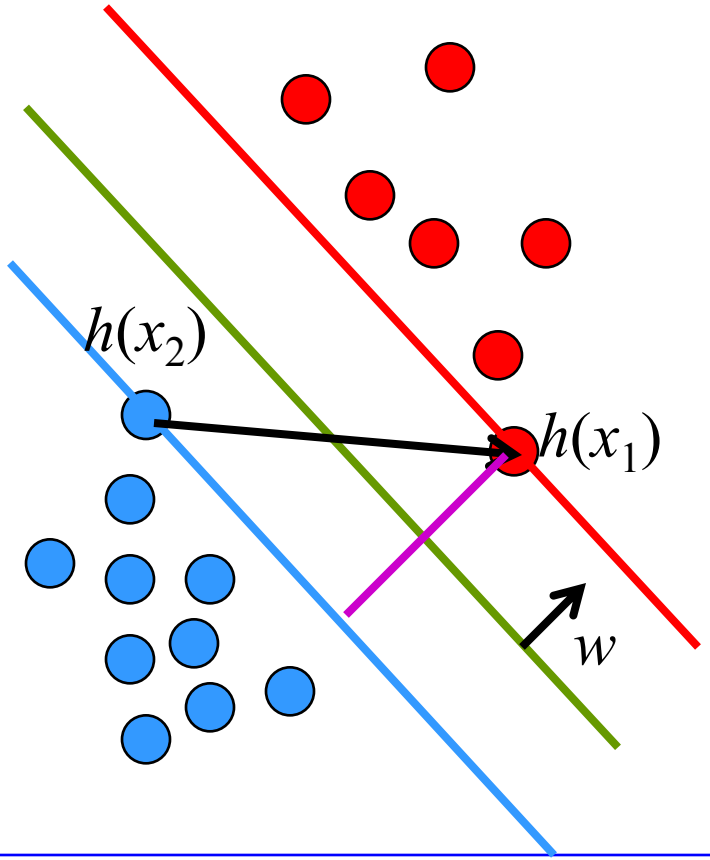
$w.[h(x_1) - h(x_2)] = 2$

and normalize the normal vector w

$\hat{w}.[h(x_1) - h(x_2)] = 2/||w||$

Support Vector Machines

The quantity $m = \hat{w}.[h(x_1) - h(x_2)]$, the distance between the **red** and **blue** planes, is called the margin. The best separation occurs when the margin is as large as possible.



But note: because $m \sim 1/||w||$, maximizing the margin is equivalent to minimizing

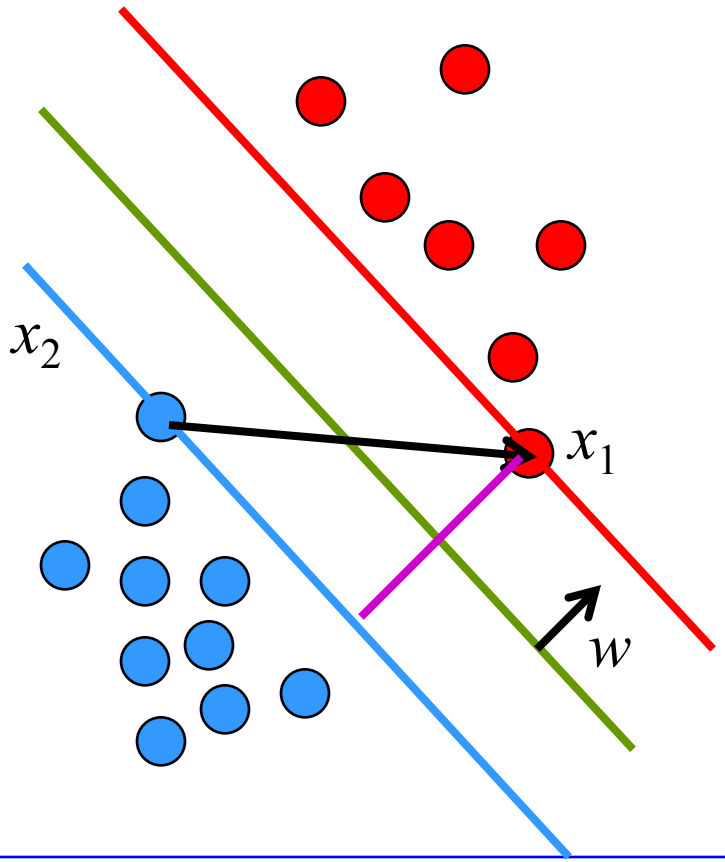
$$||w||^2$$

This is the loss function!

Support Vector Machines

Label the **red** dots $y = +1$ and the **blue** dots $y = -1$. The task is to minimize $||w||^2$ subject to the constraints

$$C = y_i\,(w.h(x_i) + c) \geq 1,\ i = 1 \ldots N$$



that is, the task is to minimize

$$L(w, c, \alpha) = \frac{1}{2\|w\|^2} - \sum_{i=1}^{N} \alpha_i [y_i(w \cdot h(x_i) + c) - 1]$$

where the α are Lagrange multipliers

Support Vector Machines

When $L(\mathrm{w},\mathrm{c},\alpha)$ is minimized with respect to w and c, the function $L(\mathrm{w},\mathrm{c},\alpha)$ can be transformed into the quadratic form:

$$E(\alpha) = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j h(x_i) \cdot h(x_j)$$

At the minimum of $E(\alpha)$, the only non-zero coefficients α are those corresponding to points *on* the **red** and **blue** planes: that is, the so-called support vectors. The key idea is to replace the scalar product $h(x_i).h(x_j)$ between two infinite dimensional vectors by a kernel function $K(x_i, x_j)$. SVMs can work very well, but there is an unsolved problem: it is not known how to choose the correct kernel for a given problem.

NEURAL NETWORKS: SHALLOW, DEEP, BAYESIAN

A Bit of History: Hilbert's 13th Problem

(One version of Problem 13): Prove

that it is *impossible* to do the following:

$$f(x_1,\ldots,x_n) = F(\, g_1(x_1),\ldots,\, g_n(x_n))$$

In 1957, Kolmogorov proved that it was possible with $n = 3$.

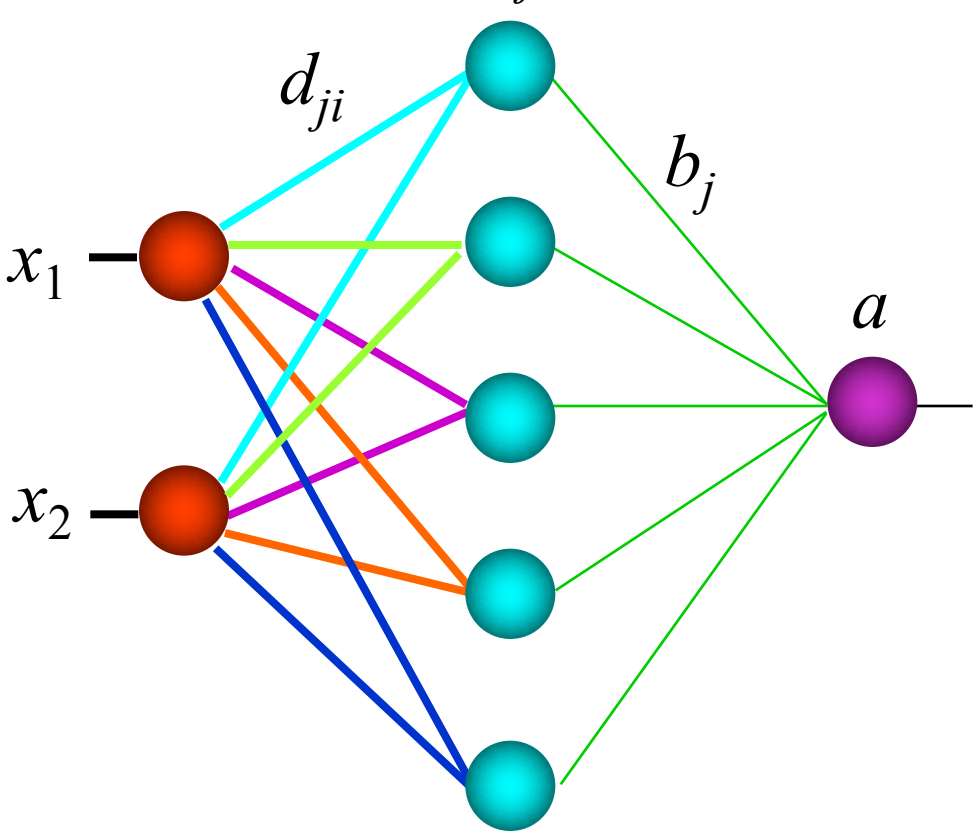
In 1989, it was shown that functions of the form

$$f(x_1,\ldots,x_I) = a + \sum_{j=1}^{H} b_j \tanh\left(c_j + \sum_{i=1}^{I} d_{ji} x_i\right)$$

can provide arbitrarily accurate approximations of real functions of *I* real variables.

(Hornik, Stinchcombe, and White, *Neural Networks* **2**, 359-366 (1989))

Shallow Neural Networks



$$f(x,w) = a + \sum_{j=1}^{H} b_j \tanh\left(c_j + \sum_{i=1}^{I} d_{ji} x_i\right)$$

$$n(x,w) = \frac{1}{1+\exp[-f(x,w)]}$$

f is used for regression
n is used for classification
$w = a, b, c, d$ are free parameters to be fitted.

Example 2: Wine Tasting

Recall that our goal is to classify wines as good or bad using the variables below:

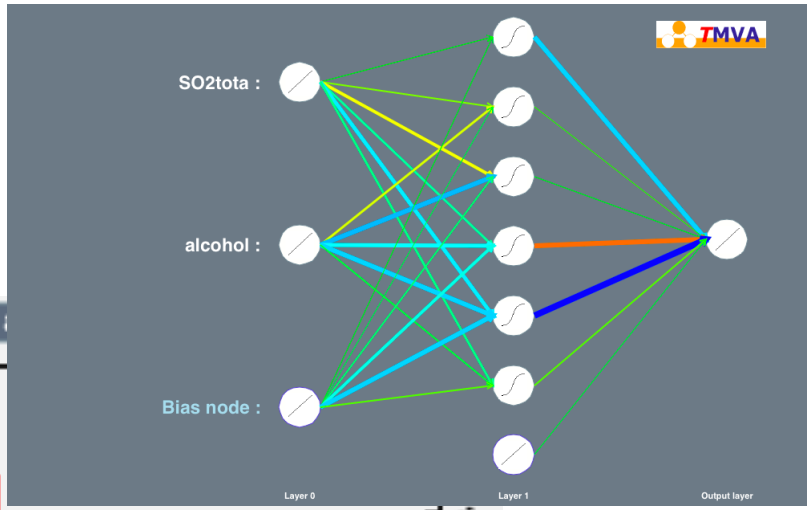
variables	description
acetic	acetic acid
citric	citric acid
sugar	residual sugar
salt	NaCl
SO2free	free sulfur dioxide
SO2tota	total sulfur dioxide
pH	pH
sulfate	potassium sulfate
alcohol	alcohol content
quality	(between 0 and 1)

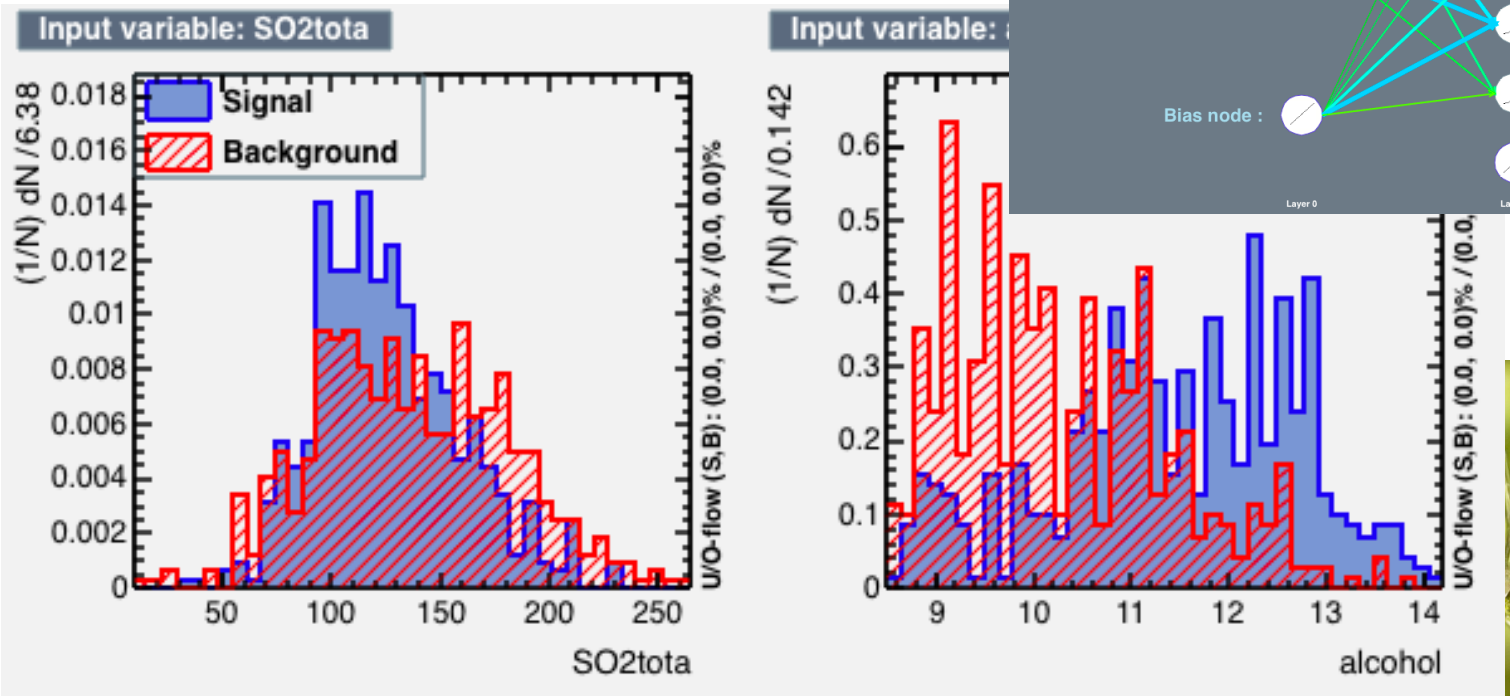
http://www.vinhoverde.pt/en/history-of-vinho-verde

Example 2: Wine Tasting

We consider the same two variables SO2tota and alcohol and the NN function shown here:

The training data comprise 500 good wines and 500 bad wines.

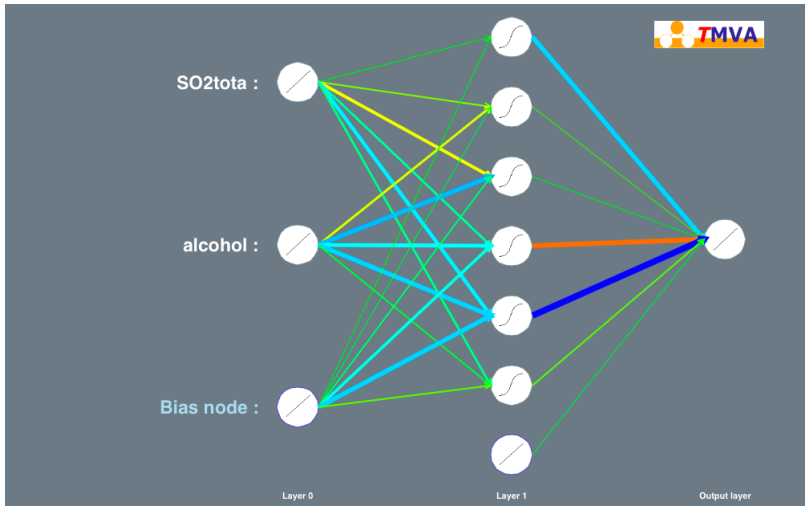




Example 2: Wine Tasting

To construct the NN, we minimize the quadratic risk function

$$R(w) = 1/2 \sum_{i=1}^{1000} [y_i - n(x_i, w)]^2$$



where x = (SO2tota, alcohol),

$y = 1$ for good wines and

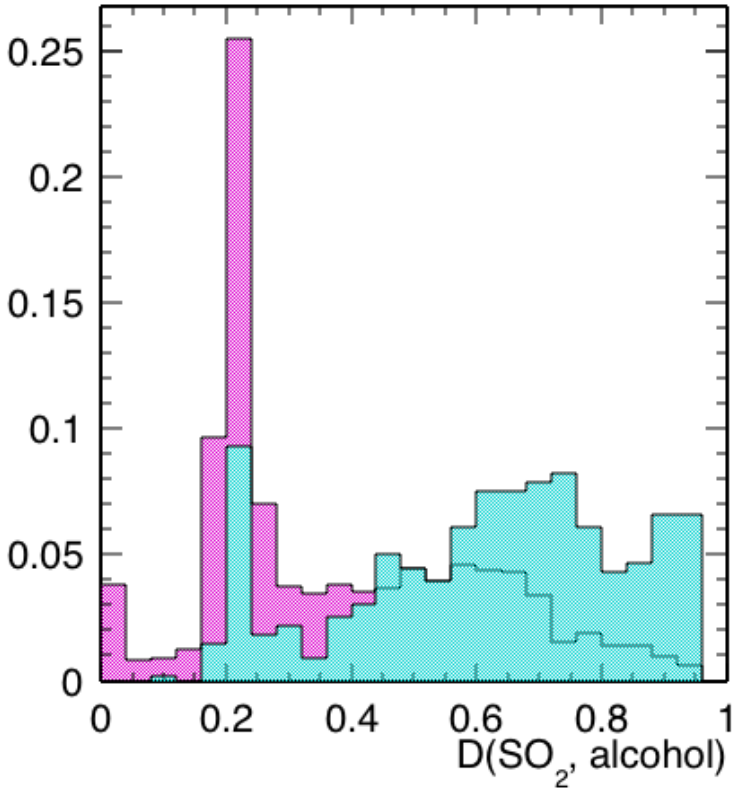
$y = 0$ for bad wines.

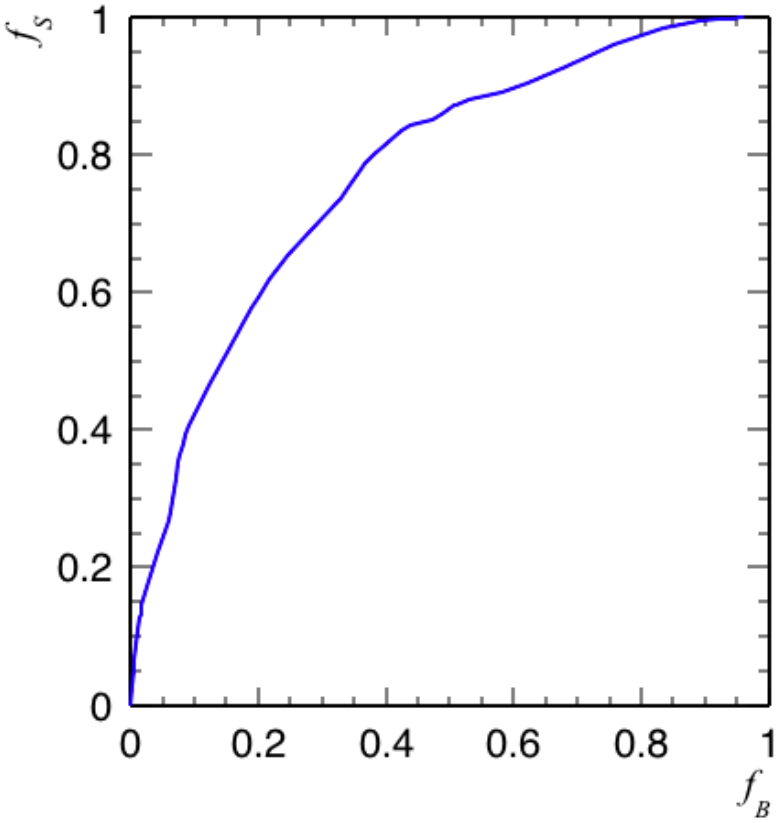
As shown yesterday, this procedure yields an approximation to

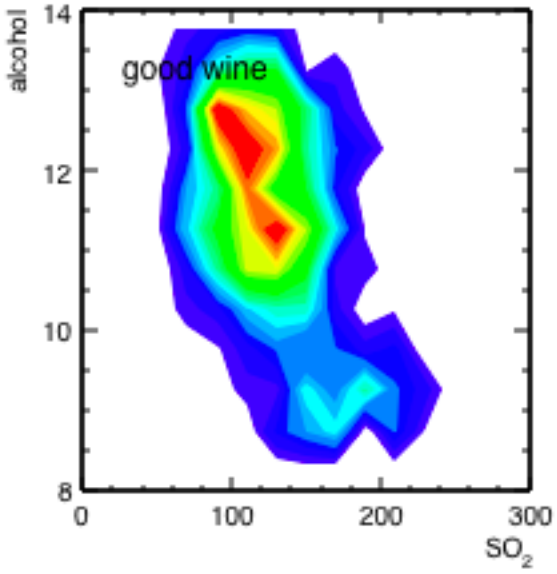
$$D(x) = \frac{p(x|good)}{p(x|good) + p(x|bad)}$$

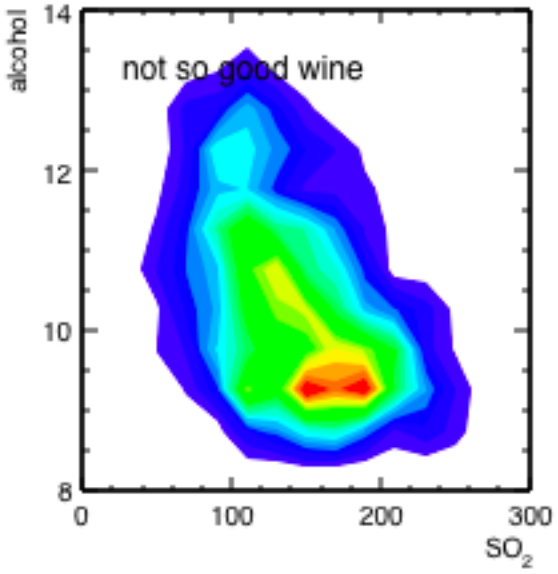
Example 2: Wine Tasting – Results

Shown are the distributions of $D(x)$ and the ROC curve. In this example, the ROC curve shows the fraction f of each class of wine *accepted*.



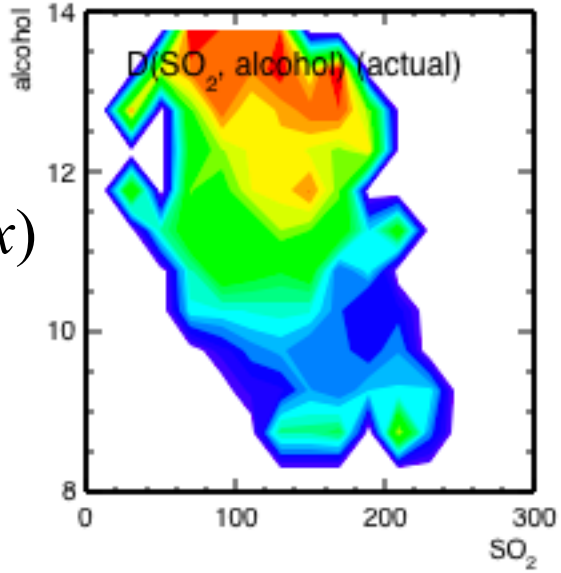


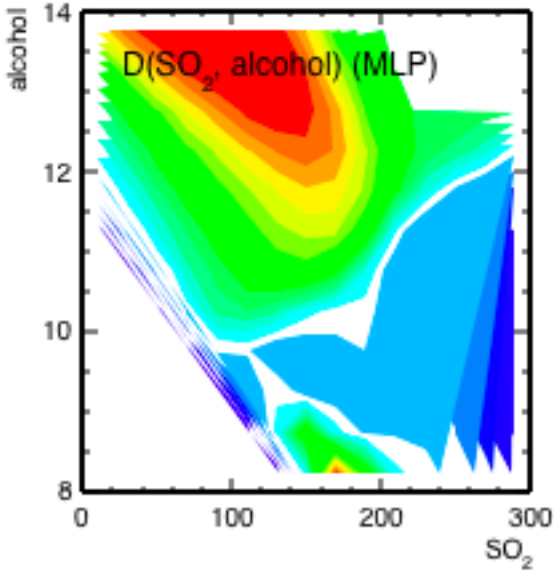




Actual

$p(good \mid x)$



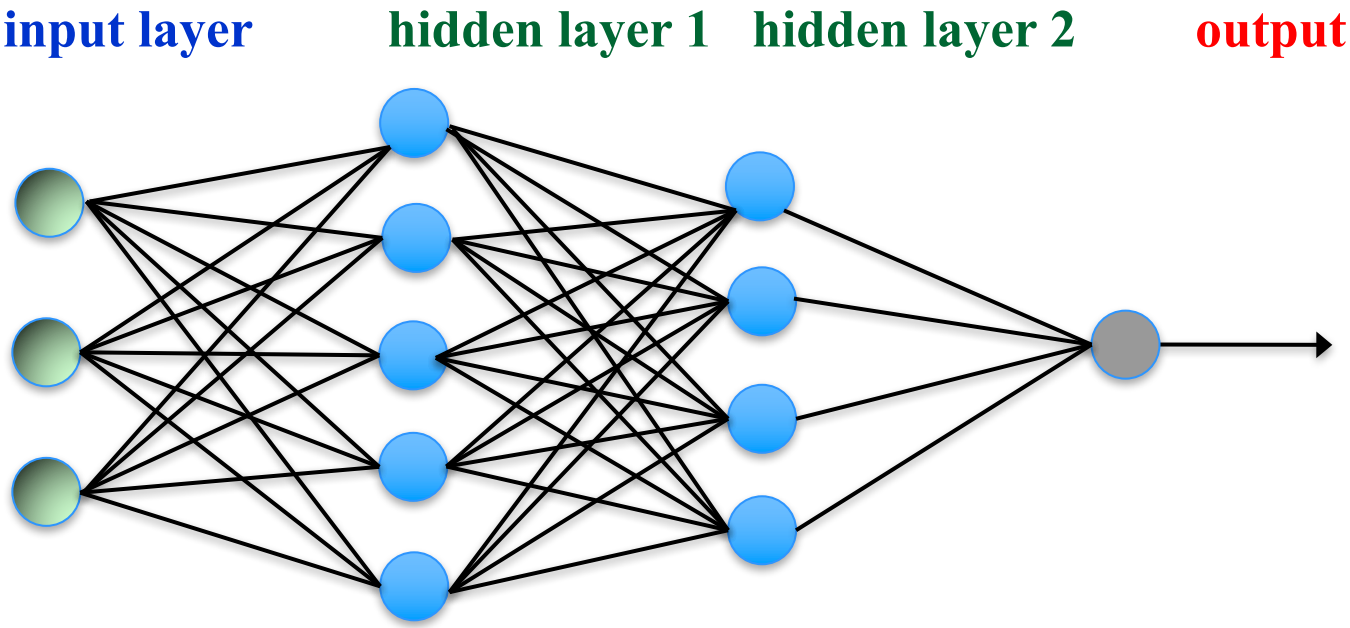


Best fit NN

DEEP NEURAL NETWORKS

Deep Neural Networks

A deep neural network (DNN) is a multi-layer neural network!



$$\boldsymbol{x}$$

$$\mathbf{h}_1(\mathbf{b}_1 + \mathbf{W}_1\boldsymbol{x})$$

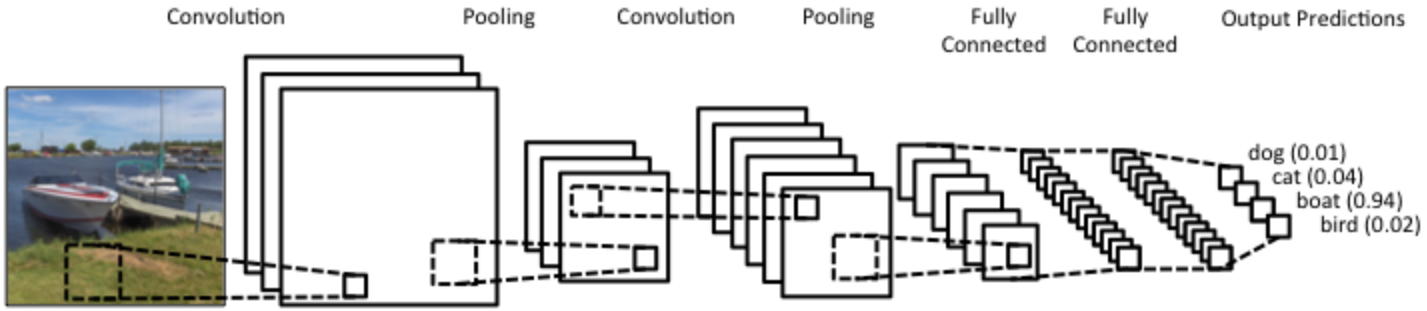
$$\mathbf{h}_2(\mathbf{b}_2 + \mathbf{W}_2\mathbf{h}_1)$$

$$\boldsymbol{o} = \mathbf{h}_3(\mathbf{b}_3 + \mathbf{W}_3\mathbf{h}_2)$$

Deep Neural Networks

Many of the remarkable breakthroughs in tasks such as face recognition use a type of DNN called a convolutional neural network (CNN).

CNNs are *functions* that compress data in a clever way and classify objects using their compressed representations via a standard fully connected NN. The compression reduces the dimensionality of the space to be searched.



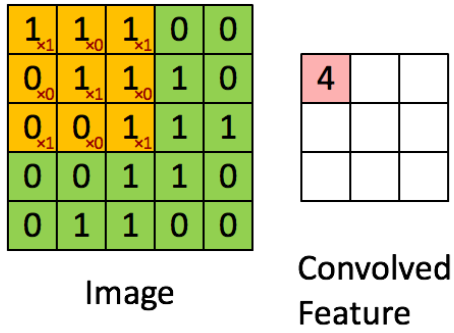
Source: https://www.clarifai.com/technology

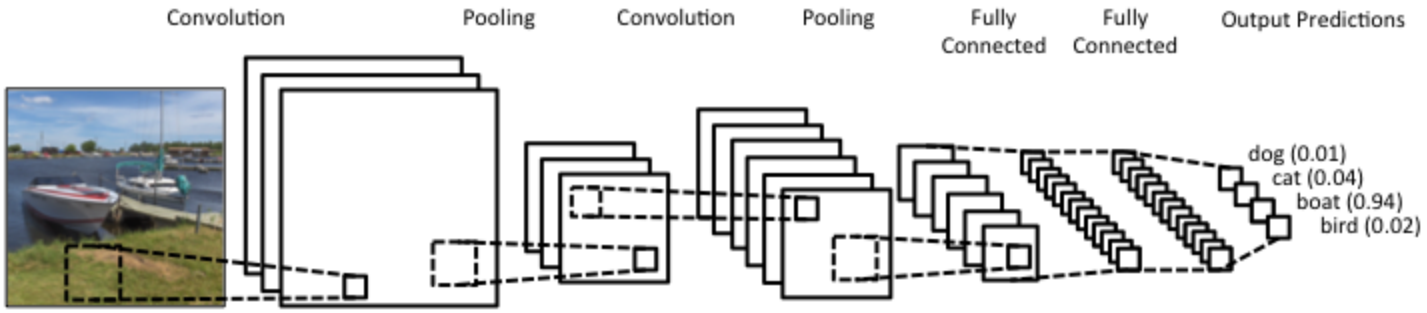
Deep Neural Networks

A CNN comprises three types of processing layers: 1. convolution, 2. pooling, and 3. classification.

1. Convolution layers

The input layer is "convolved" with one or more matrices using element-wise products that are then summed. In this example, since the sliding matrix fits 9 times, we compress the input from 5 x 5 to a to 3 x 3 matrix.

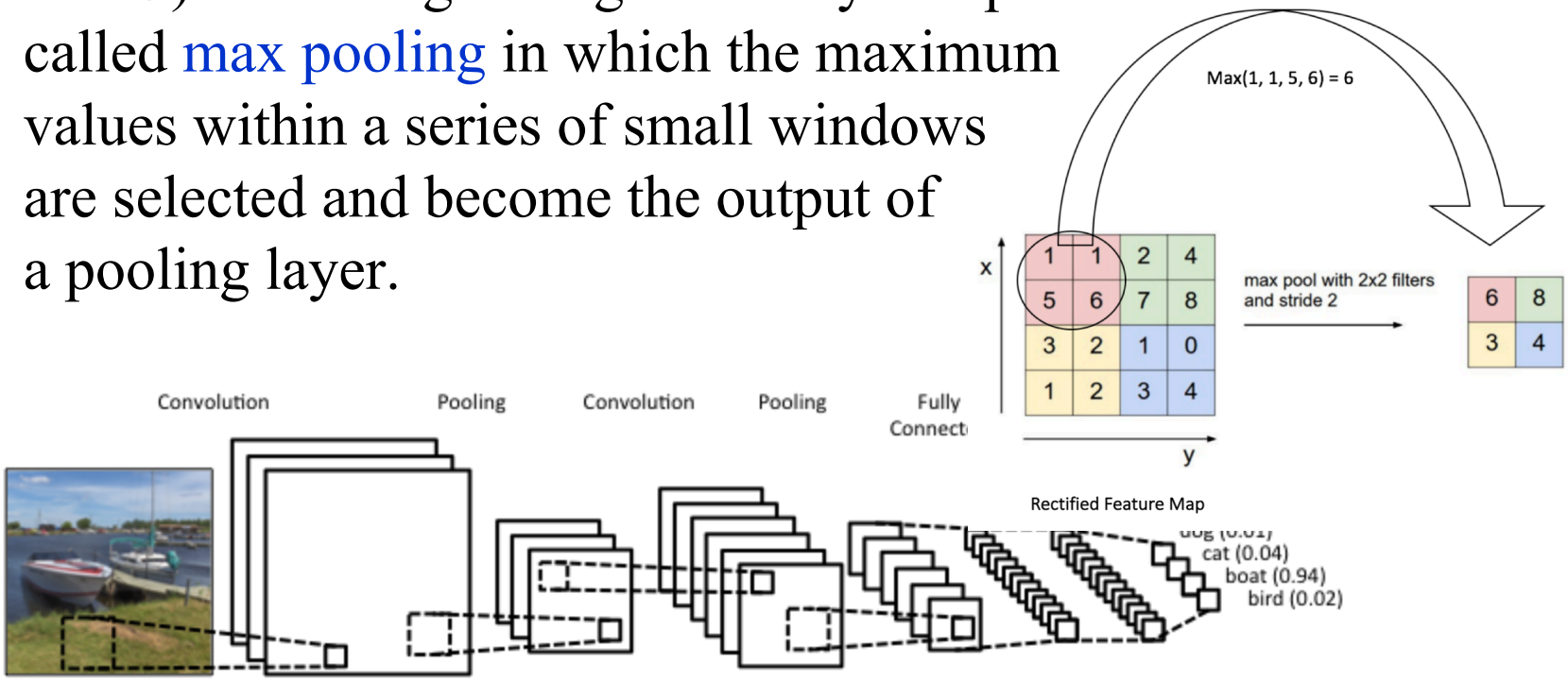




Deep Neural Networks

2. Pooling Layers

After convolution, and a pixel by pixel non-linear transformation (using, e.g., the function $y = \max(0, x)$ – ReLU) a coarse-graining of the layer is performed called max pooling in which the maximum values within a series of small windows are selected and become the output of a pooling layer.

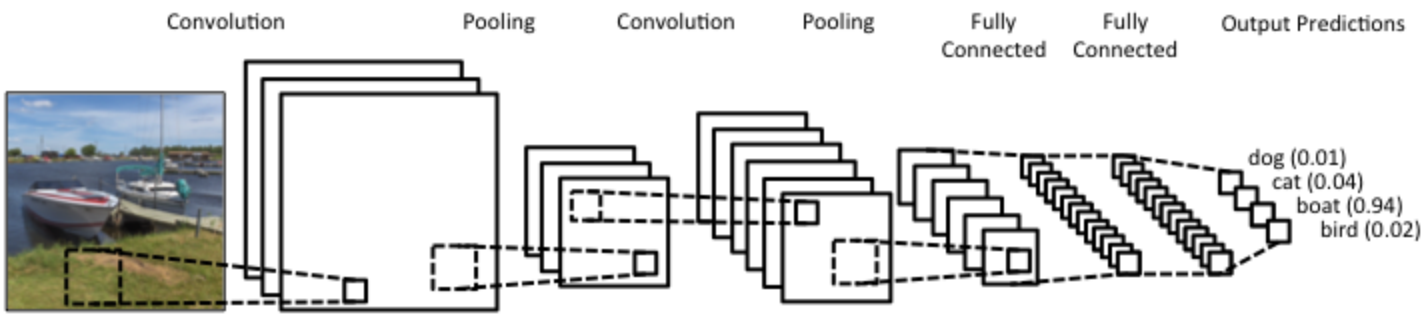


Deep Neural Networks

3. Classification Layers

After an alternating sequence of convolution and pooling layers, the outputs go to a standard neural network, either shallow or deep. The final outputs correspond to the different classes. In spite of the complexity, a CNN approximates the same thing as any other classifier, namely,

$$p(C_k|x) = p(x|C_k)p(C_k) / \sum_{m=1}^{M} p(x|C_m)p(C_m)$$



Deep Neural Networks

- In 2014, a machine learning competition (HiggsML) was conducted (https://higgsml.lal.in2p3.fr) in which the goal was to separate ATLAS simulated $H \to \tau^+\tau^-$ events from simulated background.

- The competition was won by Gabor Melis (Franz Inc., Fixnum Services, Hungary) who created a classifier using the average of 70 DNNs, each with architecture (35,600,600,600,2), i.e., 35 inputs, 3 hidden layers of 600 nodes each, and 2 outputs. This is a classifier with more than 70 million fitted parameters!

Deep Neural Networks

- In 2006, University of Toronto researchers Hinton, Osindero, and Teh (HOT*) finally succeeded in training a deep neural network. Each layer was trained to produce a representation of its inputs that served as the training data for the next layer. Then the entire network was tweaked using gradient descent.

- This breakthrough seemed to provide compelling evidence that the training of deep neural networks requires careful initialization of parameters and sophisticated machine learning algorithms.

* Hinton, G. E., Osindero, S. and Teh, Y. (HOT), A fast learning algorithm for deep belief nets, Neural Computation 18, 1527-1554.

Deep Neural Networks

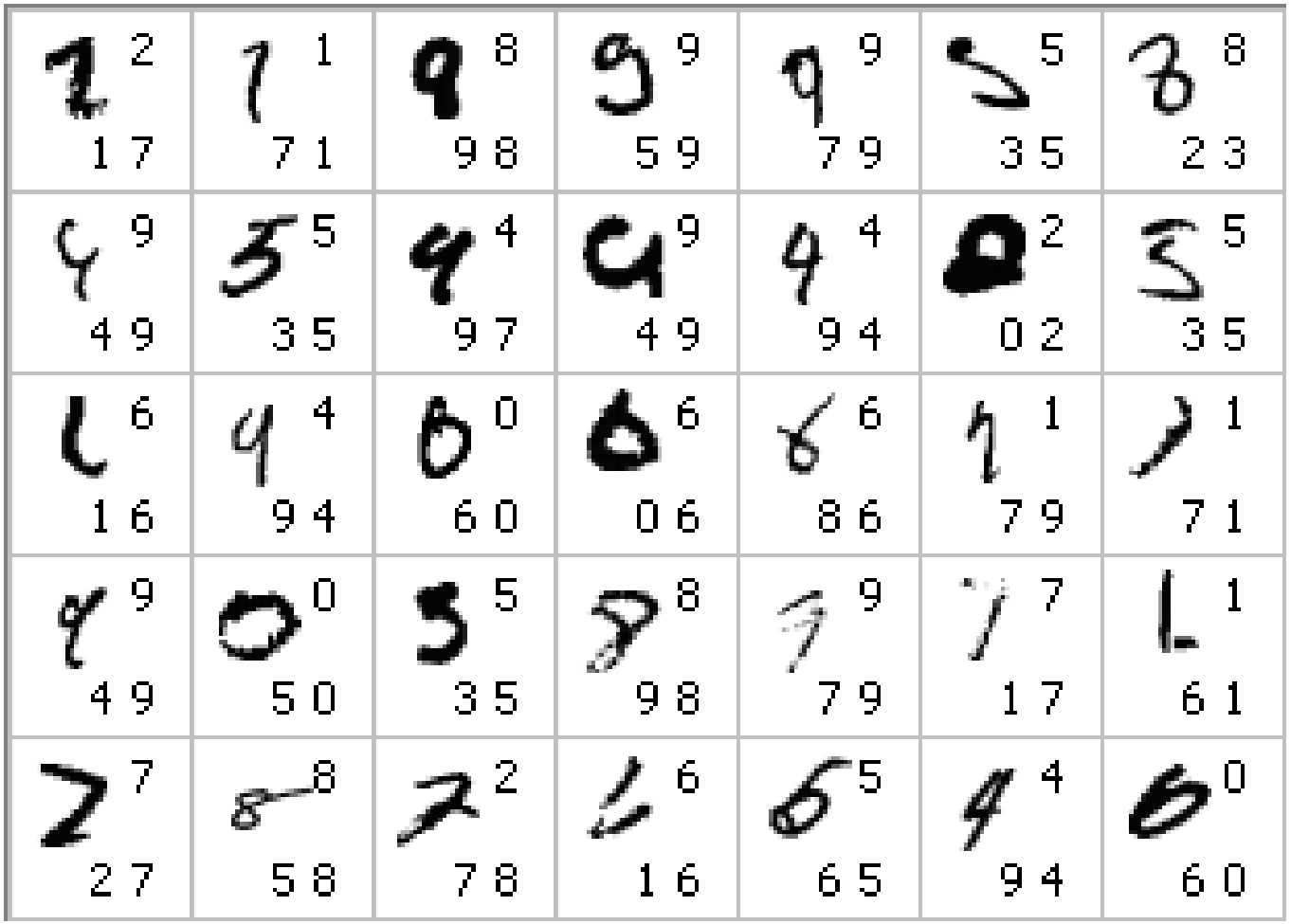
- But, in 2010, Ciresan *et al.** showed that such cleverness was not needed! The authors succeeded in training a DNN with architecture (784, 2500, 2000, 1500, 1000, 500, 10) that classified the hand-written digits in the MNIST database.

- The database comprises 60,000 $28 \times 28 = 784$ pixel images for training and validation, and 10,000 for testing.

- The error rate of their ~12-million parameter DNN was 35 images out of 10,000. The misclassified images are shown on the next slide.

* Cireşan DC, Meier U, Gambardella LM, Schmidhuber J. , Deep, big, simple neural nets for handwritten digit recognition. Neural Comput. 2010 Dec; 22 (12): 3207-20. http://yann.lecun.com/exdb/mnist/

Deep Neural Networks



Deep Neural Networks

Question:
how is it possible to fit a 12 million-parameter function, in a matter of *hours*, with a mere 60,000 images, avoid overfitting, and beat the most sophisticated methods (as of 2010)?

Answer:
1. brute force: use lots of computing power, e.g., GPUs, and
2. a clever trick: generate a limitless amount of training data by *randomly deforming* the images every training cycle.

By deforming the images, the original 60,000 images could be used as the validation set!

BAYESIAN NEURAL NETWORKS

Recap: Bayesian Learning

Choose

Function space $\boldsymbol{F} = \{ f(x, \boldsymbol{w}) \}$

Likelihood $p(\boldsymbol{T} \mid \boldsymbol{w})$, training set $\boldsymbol{T} = \{(\boldsymbol{y}, \boldsymbol{x})\}$

Loss function $\boldsymbol{L}$

Prior $p(\boldsymbol{w})$

Method

Use Bayes' theorem to assign a probability (density)

$$p(\boldsymbol{w} \mid \boldsymbol{T}) = p(\boldsymbol{T} \mid \boldsymbol{w})\, p(\boldsymbol{w}) / p(\boldsymbol{T})$$
$$= p(\boldsymbol{y} \mid \boldsymbol{x}, \boldsymbol{w})\, p(\boldsymbol{x} \mid \boldsymbol{w})\, p(\boldsymbol{w}) / p(\boldsymbol{y} \mid \boldsymbol{x})\, p(\boldsymbol{x})$$
$$\sim p(\boldsymbol{y} \mid \boldsymbol{x}, \boldsymbol{w})\, p(\boldsymbol{w}) \qquad \text{(assuming } p(\boldsymbol{x} \mid \boldsymbol{w}) = p(\boldsymbol{x})\text{)}$$

to *every* function in the function space, where $p(\boldsymbol{y} \mid \boldsymbol{x}, \boldsymbol{w})$ is the product of the likelihoods $p(y_i \mid x_i, \boldsymbol{w})$ for each (y_i, x_i).

Recap: Bayesian Learning

Given, the posterior density $p(\boldsymbol{w} \mid T)$, and some *new* data x one can compute the predictive distribution of y

$$p(y \mid x, T) = \int p(y \mid x, w)\, p(w \mid T)\, dw$$

In general, a different L, will yield a different estimate of $f(x)$.

Bayesian Neural Networks

For regression, the likelihood $p(y \mid x, w)$ for a given (y, x) is typically assumed to be Gaussian

$$p(y|x,w) = A\exp\left[-\frac{1}{2}\big(y - f(x,w)\big)^2/\sigma^2\right]$$

For classification one uses

$$p(y|x,w) = [n(x,w)]^y[1-n(x,w)]^{1-y}, \qquad \text{y} = 0 \text{ or } 1$$

Setting $y = 1$, in $p(y|x,T) = \int p(y|x,w)\, p(w|T)dw$

gives $p(y=1|x,T) = \int n(x,w)\, p(w|T)dw$

which is an estimate of the probability that the object characterized by x belongs to the class for which $y = 1$.

Bayesian Neural Networks

In practice, the high dimensional integral

$$p(y = 1|x, T) = \int n(x, w)\, p(w|T) dw$$
$$\approx \frac{1}{M} \sum_{m=1}^{M} n(x, w_m)$$

is approximated using Markov Chain Monte Carlo (MCMC) integration in which the parameters w are sampled from the posterior density $p(w|T)$.

Pros: Automatically obtain an uncertainty estimate and the averaging makes the function estimates more robust.

Cons: Training times can be very long.

Example 1: Higgs to ZZ to 4 Leptons

Dots

$p(s \mid x) \sim H_s / (H_s + H_b)$

H_s, H_b, are 1-D histograms of the transverse momentum of the Higgs boson.

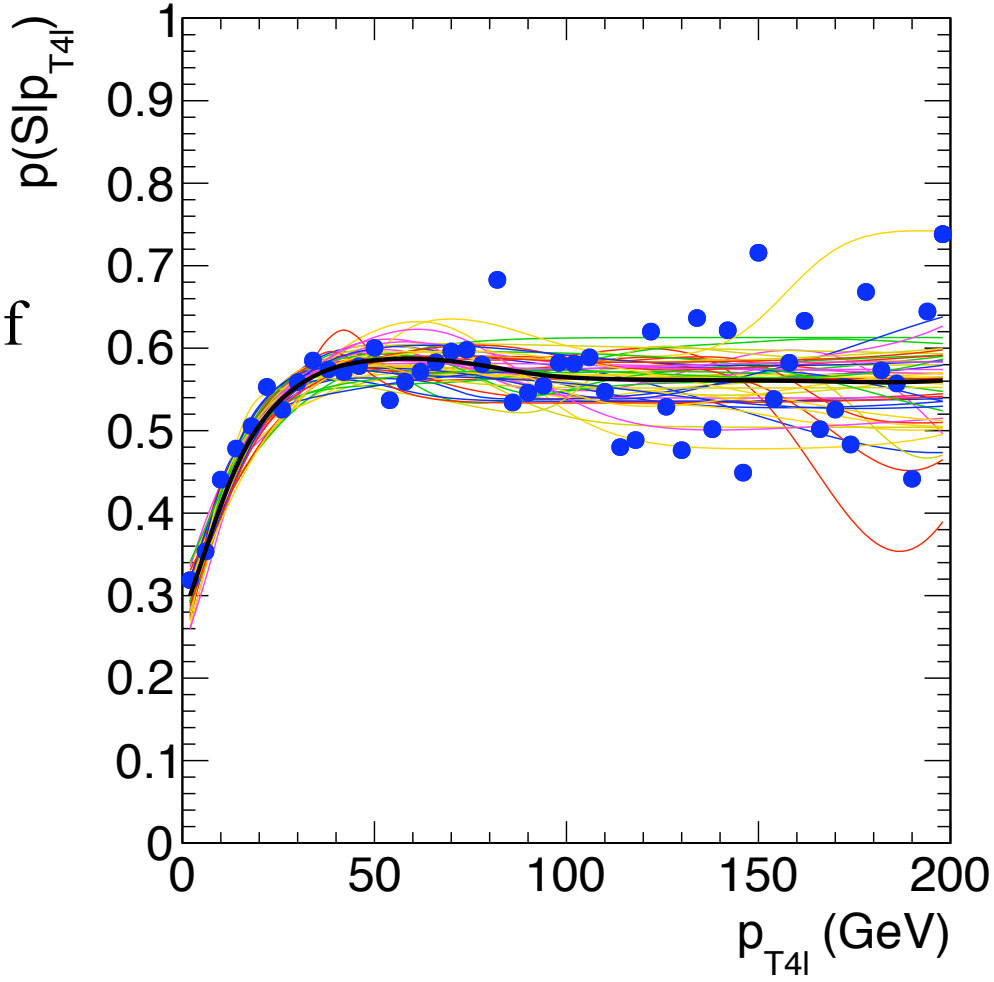
Curves

Individual NNs

$n(x, w_k)$

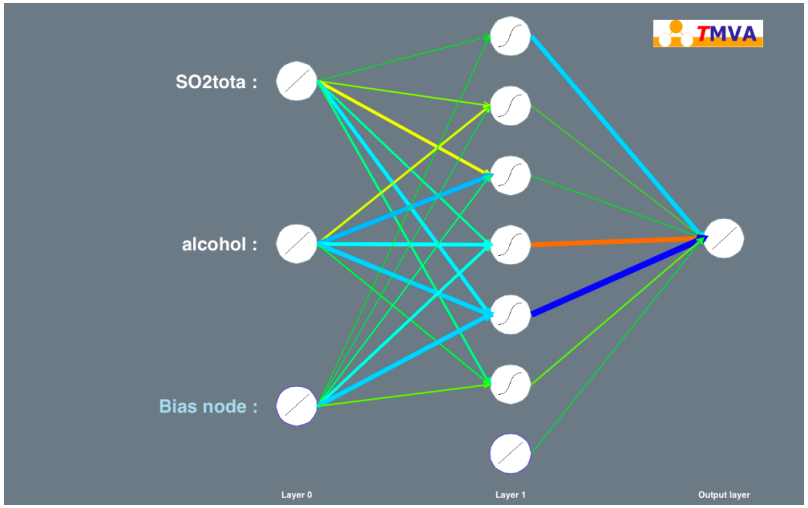
Black curve

$n(x) = \langle n(x, w) \rangle$



Example 2: Wine Tasting

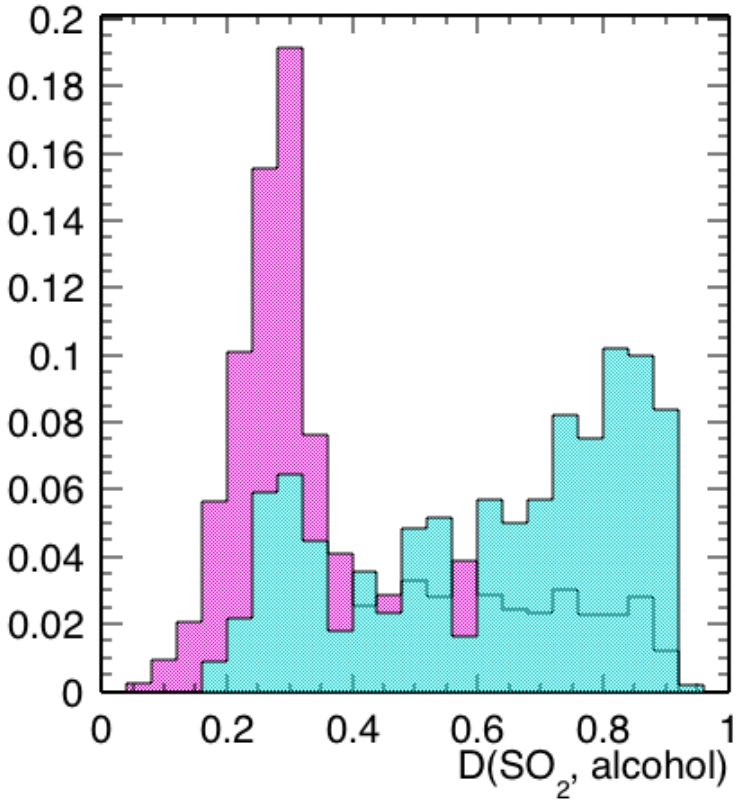
To construct the BNN, we *sample* points w from the posterior density $p(w|T)$ using the Hybrid Monte Carlo method implemented in the **F**lexible **B**ayesian **M**odeling package by statistician Radford Neal.

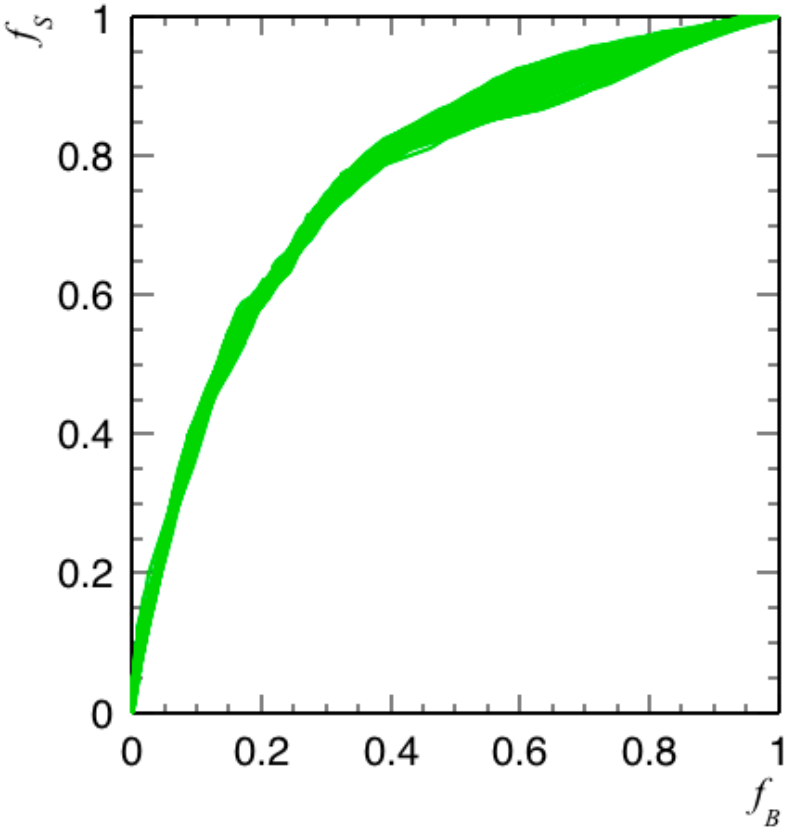


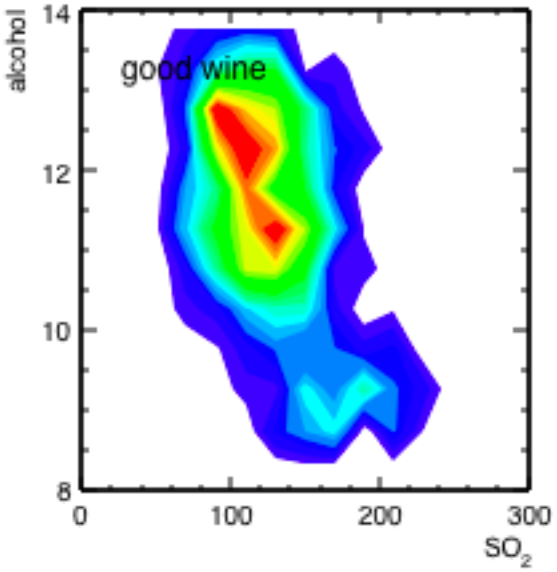
This sampling generates an *ensemble* of neural networks, just as in the 1-D example on the previous slide.

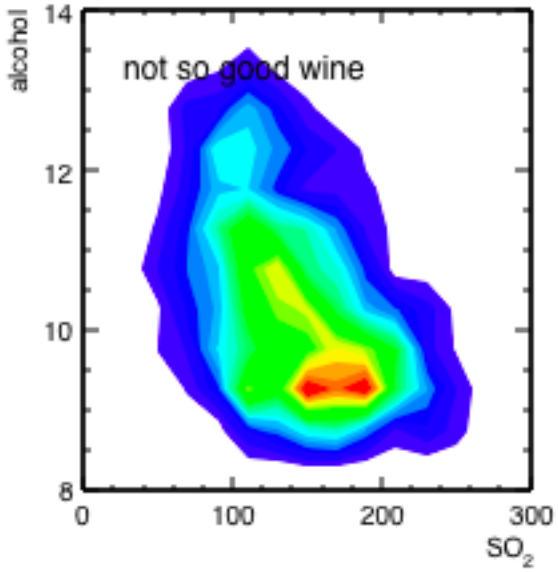
Example 2: Wine Tasting – Results

Shown ares the distributions of $D(x)$ (using the *average* of 100 NNs) and the *ensemble* of ROC curves, one for each NN.

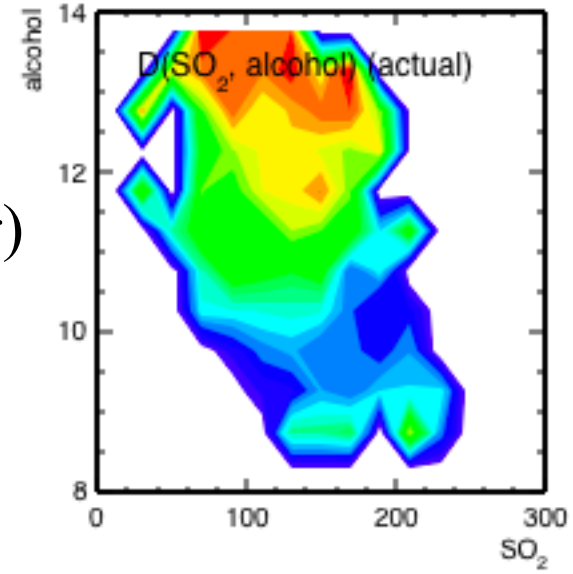


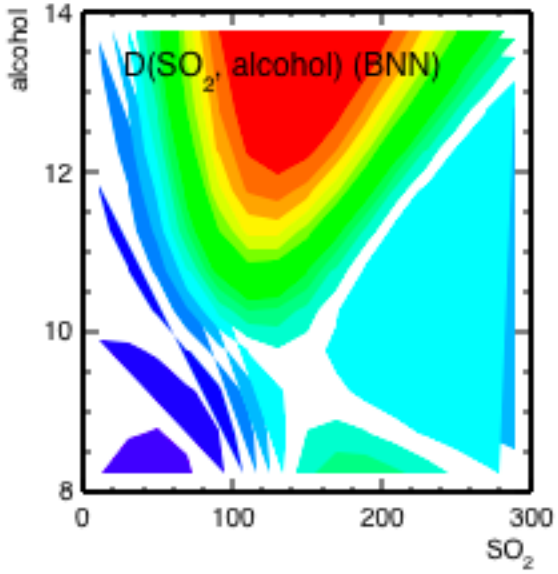






Actual
$p(good \mid x)$



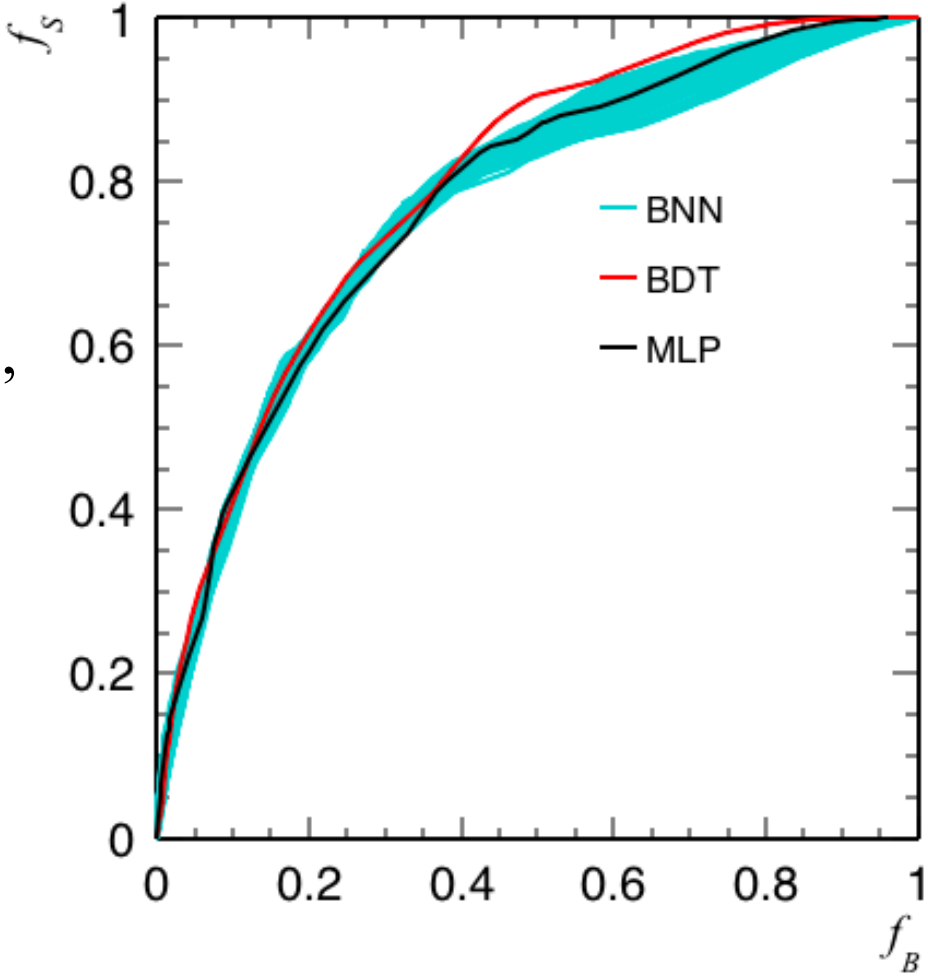


<NN>

Comparing ROCs!

We see that if good and bad wines are equally likely and we are prepared to accept 40% of bad wines, while accepting 80% of good ones, then the BDT does slightly better.

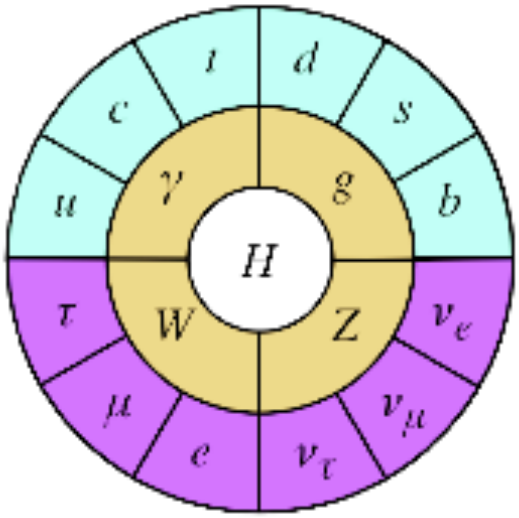
But, if we lower the acceptance rate of bad wines, all three methods work equally well.



REGRESSION

Function Approximation

Now that the Higgs boson has been found, our primary goal at the LHC is to look for, and if we're lucky, discover new physics.



There are two basic strategies:

1. Look for any deviation from the predictions of our current theory of particles (the Standard Model).

2. Look for the specific deviations predicted by theories of new physics, such as those based on supersymmetry.

Function Approximation

While finishing a CMS paper related to his PhD dissertation, my former student Sam Bein had to approximate a likelihood function $L(\theta) = p(x \mid \theta)$, which was a function of the 19 parameters θ of the pMSSM.

Targeting the Minimal Supersymmetric Standard Model with the Compact Muon Solenoid Experiment

by

Samuel Louis Bein

Submitted to the Department of Physics
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy in Physics

at the

FLORIDA STATE UNIVERSITY

June 2016

Function Approximation

This is what I suggested he try.

1. Write the likelihood function to be approximated as

$$L(\theta) = C(\theta) \prod_{i=1}^{19} g_i(\theta_i)$$

2. Approximate each 1-D density $g_i(\theta_i)$ using, e.g., a NN.

3. Use a neural network to approximate

$$D(\theta) = \frac{p(\theta)}{p(\theta) + \prod_{i=1}^{19} g_i(\theta_i)}$$

where the "background" is sampled from the 1-D functions and the "signal" is the original distribution of points sampled from the pMSSM parameter space.

Function Approximation

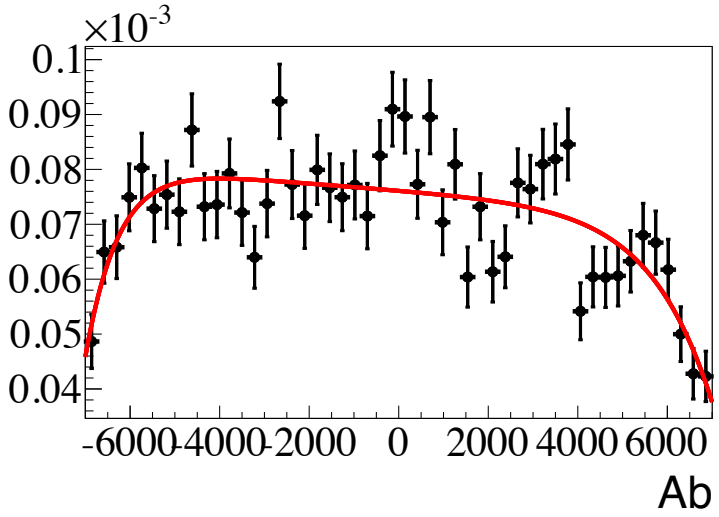
4. Finally, approximate the function L using

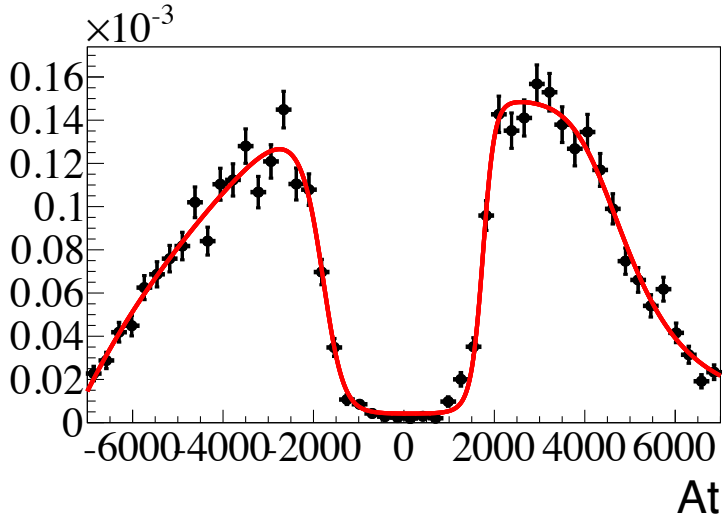
$$L(\theta) = \frac{D(\theta)}{1 - D(\theta)} \prod_{i=1}^{19} g_i(\theta_i)$$

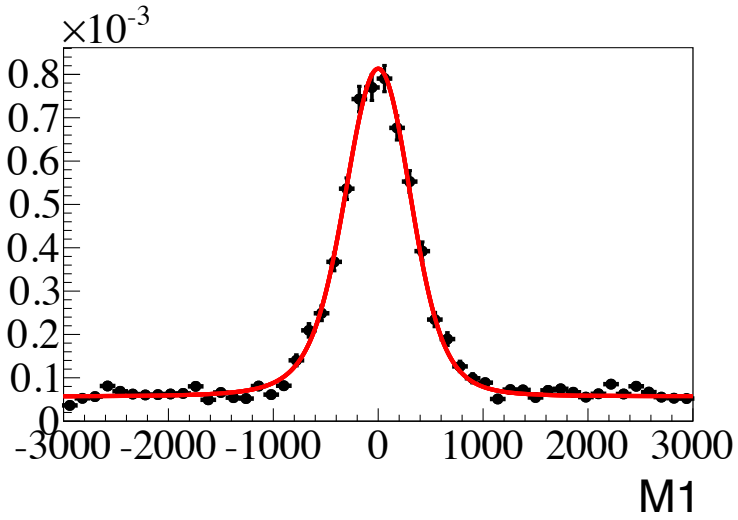
The idea here is that much of the dependence of L on θ is captured by the product function, leaving (we hoped!) a gentler 19-dimensional function to be approximated.

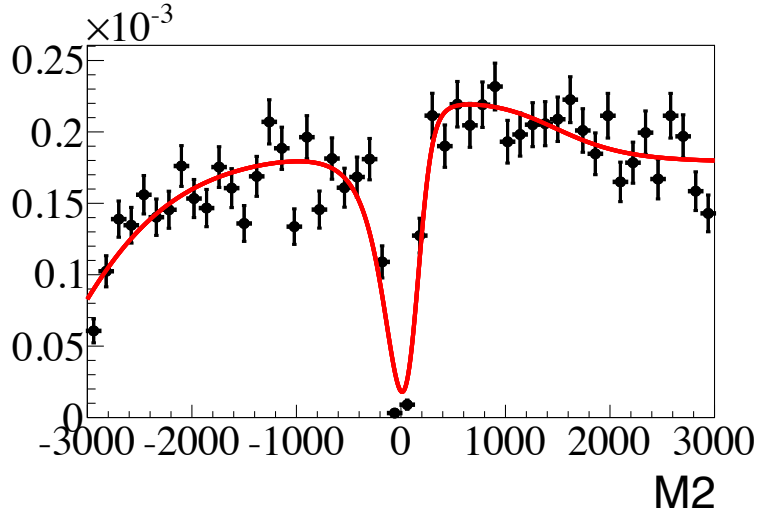
The next slide shows the neural network approximations of a few of the g functions. Of course, 1-D functions can be approximated using a wide variety of methods. The point here is to show the flexibility of a *single* class of NNs.

$g_i(\theta_i)$

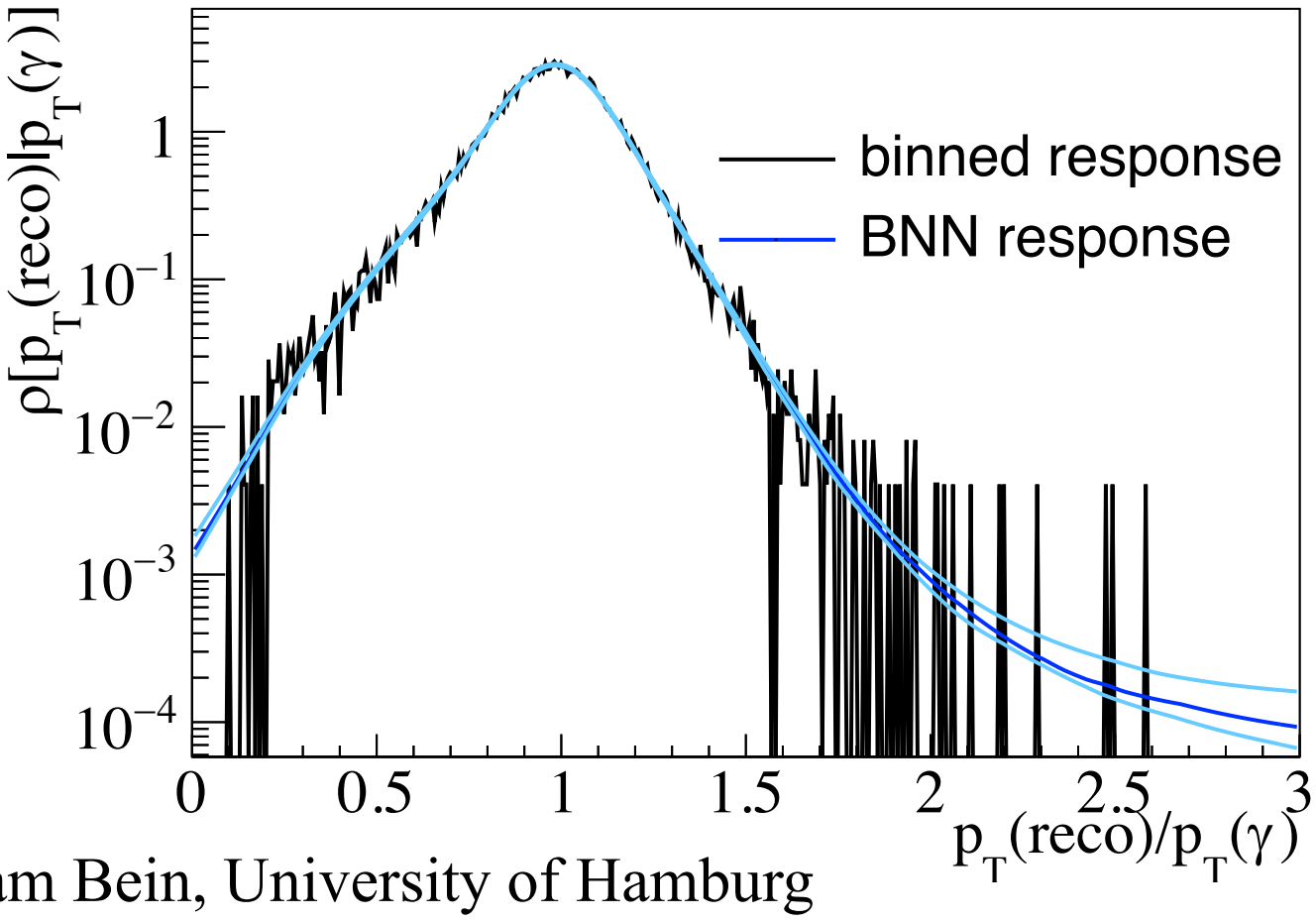

×10^-3
0.1
0.09
0.08
0.07
0.06
0.05
0.04
-6000 -4000 -2000 0 2000 4000 6000
Ab


×10^-3
0.16
0.14
0.12
0.1
0.08
0.06
0.04
0.02
0
-6000 -4000 -2000 0 2000 4000 6000
At


×10^-3
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
-3000 -2000 -1000 0 1000 2000 3000
M1


×10^-3
0.25
0.2
0.15
0.1
0.05
0
-3000 -2000 -1000 0 1000 2000 3000
M2

Hot off the Press: CMS Jet Response



Sam Bein, University of Hamburg

Summary

- Machine learning can be applied to many aspects of data analysis, including classification and regression. There are hundreds of methods, but almost all are *numerical approximations* of the *same* mathematical quantity.

- We considered random grid searches, decision trees, boosted decision trees, neural networks, shallow, deep, and Bayesian.

- *No* method is best for *all* problems. Therefore, it is good practice to try a few of them, say BDTs and NNs, and check that they give approximately the same results.