~~**Data Analysis: Machine Learning Techniques**~~

Toe In Water

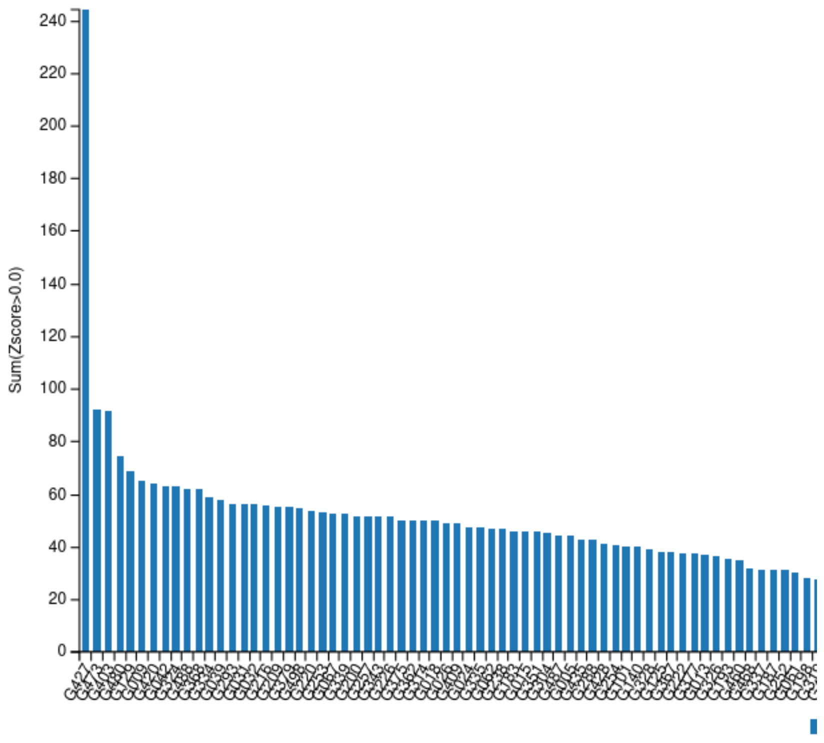
….. should be thought of as "put a toe in the bath water"

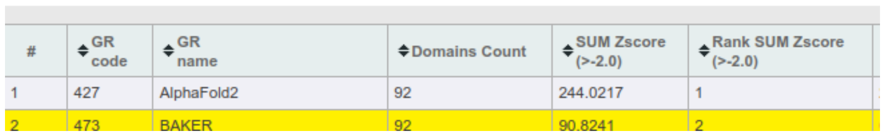
Kurtis F. Johnson, Florida State University & University Bari
17 June 2022

Tirana School for Data Science, Particle & Astroparticle Physics

The activity here is aimed at those with little, or even less, experience of ML; to have them acquire enough tools, in the form of an understood running code example and pointers on how to continue, so that they are comfortable and motivated to do so. *We have 3 hours.*

Why ML? - because ML can help generate new scientific insights by uncovering connections in a mass of data. Just one quick example (from https://predictioncenter.org/casp14/zscores_final.cgi) AlphaFold's correct reconstructions of protein 3-D configurations using ML.





#	GR code	GR name	Domains Count	SUM Zscore (>-2.0)	Rank SUM Zscore (>-2.0)
1	427	AlphaFold2	92	244.0217	1
2	473	BAKER	92	90.8241	2

But undetected errors can occur (from arxiv.org/pdf/1808.03305.pdf ; “The Elephant in the Room” Amir Rosenfeld et al.) where a very large animal is not detected by the state-of-the-art image ML

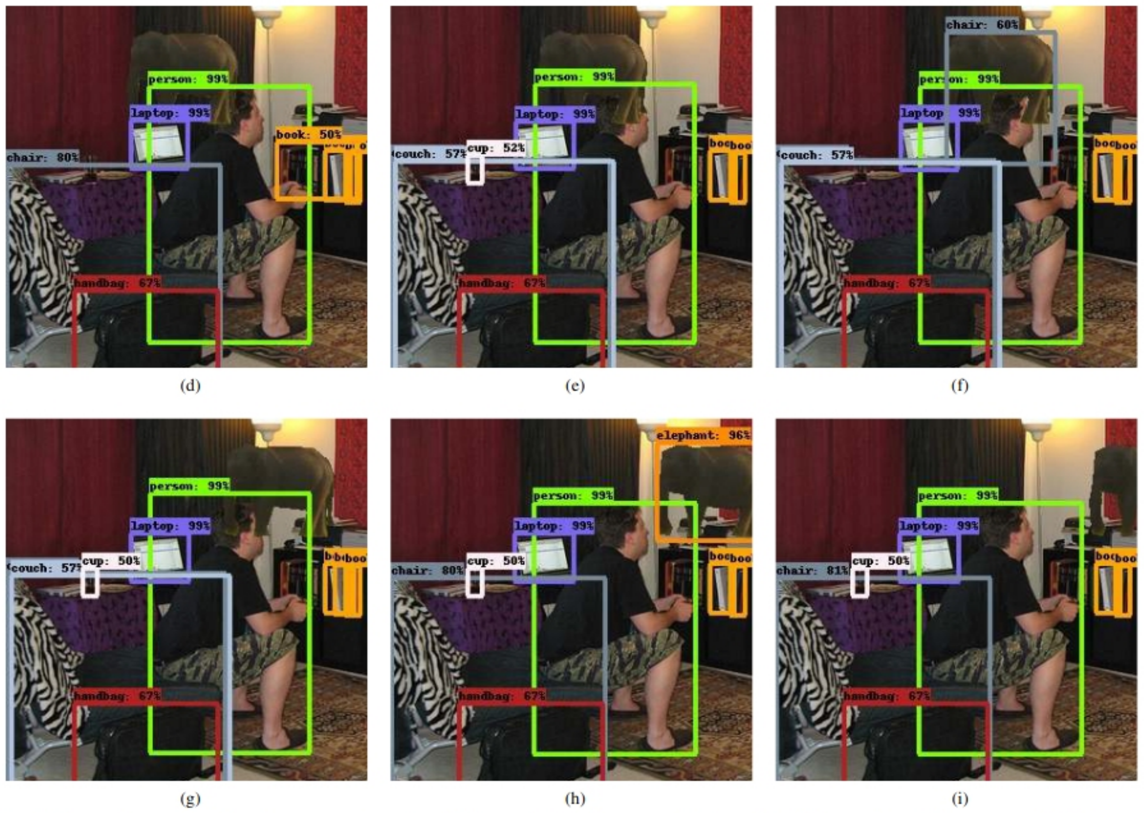


Fig. 1: Detecting an elephant in a room. A state-of-the-art object detector detects multiple images in a living-room (*a*). A transplanted object (elephant) can remain undetected in many situations and arbitrary locations (*b,d,e,g,i*). It can assume incorrect identities such as a chair (*f*). The object has a non-local effect, causing other objects to disappear (cup, *d,f,* book, *e-i*) or switch identity (chair switches to couch in *e*). It is recommended to view this image in color online.

Given that ML’s are fallible, it is very, very, ***very***, ***very*** important to certify your finished ML by (at the beginning) creating a “holdback” or “test” dataset which is distinct from the training dataset, and which is “touched” only once at the end of development. This is the one method which can provide an independent measure of the reliability of your ML.

Why ML now? …. machines which learn have been explored for >50 years. Why now?

The simple answer: CHEAP GPU’s

Many ML algorithms can be **parallelized** or require **lots** of dot products (matrix multiplications). GPU’s are stripped down CPU’s to do just that. As a GPU is much less capable than a CPU it can be made much more simply and can be packed tightly onto a chip and thousands can be run in parallel.

ML will continue to expand as cost/performance declines (from aiimpacts.org/2019-recent-trends-in-gpu-price-per-flops/ ; Figure 2) by a factor of ten during a decade.

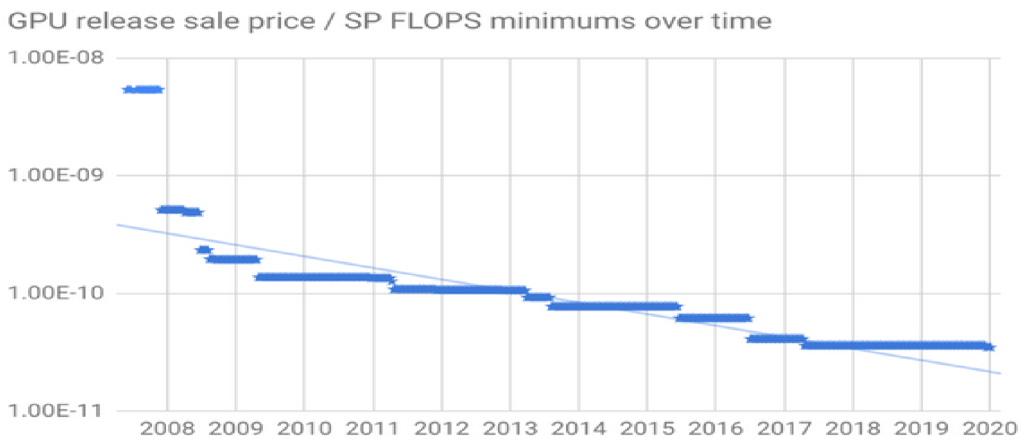
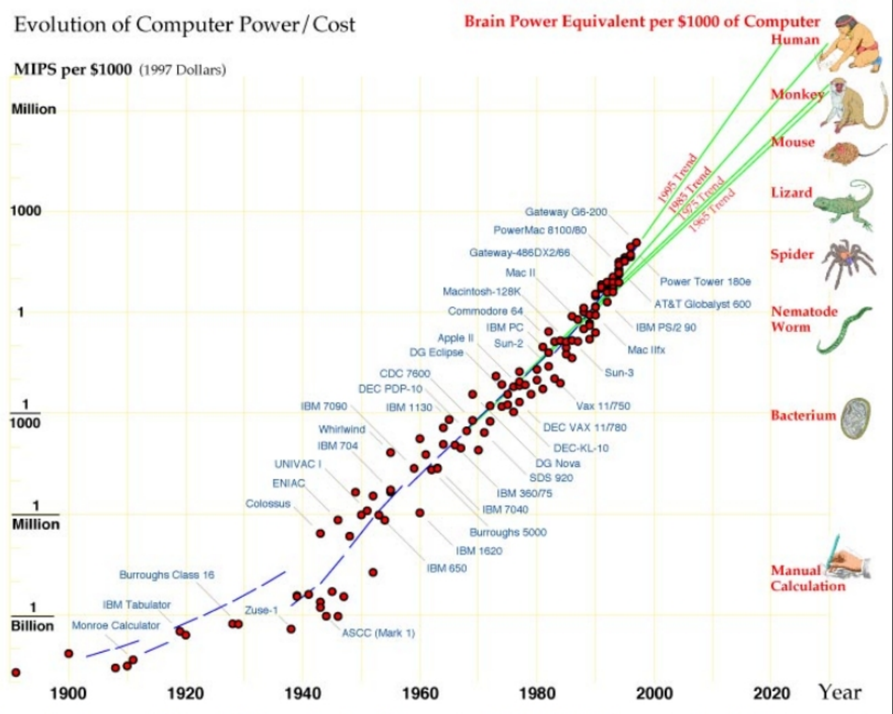


Figure 2: Ten-day minimums in real GPU price / single-precision FLOPS over time. The vertical axis is log-scale. Price is measured in 2019 dollars. The blue line shows the trendline ignoring data before late 2007. (We believe the apparent steep decline prior to late 2007 is an artefact of a lack of data for that time period.)

From www.jetpress.org/volume1/moravec.htm, “When will computer hardware match the human brain?”, Hans Moravec, Carnegie Mellon University.



Faster than Exponential Growth in Computing Power. The number of MIPS in $1000 of computer from 1900 to the present. Steady improvements in mechanical and electromechanical calculators before World War II had increased the speed of calculation a thousandfold over manual methods from 1900 to 1940. The pace quickened with the appearance of electronic computers during the war, and 1940 to 1980 saw a millionfold increase. The pace has been even quicker since then, a pace which would make humanlike robots possible before the middle of the next century. The vertical scale is logarithmic, the major divisions represent thousandfold increases in computer performance. Exponential growth would show as a straight line, the upward curve indicates faster than exponential growth, or, equivalently, an accelerating rate of innovation. The reduced spread of the data in the 1990s is probably the result of intensified competition: underperforming machines are more rapidly squeezed out. The numerical data for this power curve are presented in the appendix.

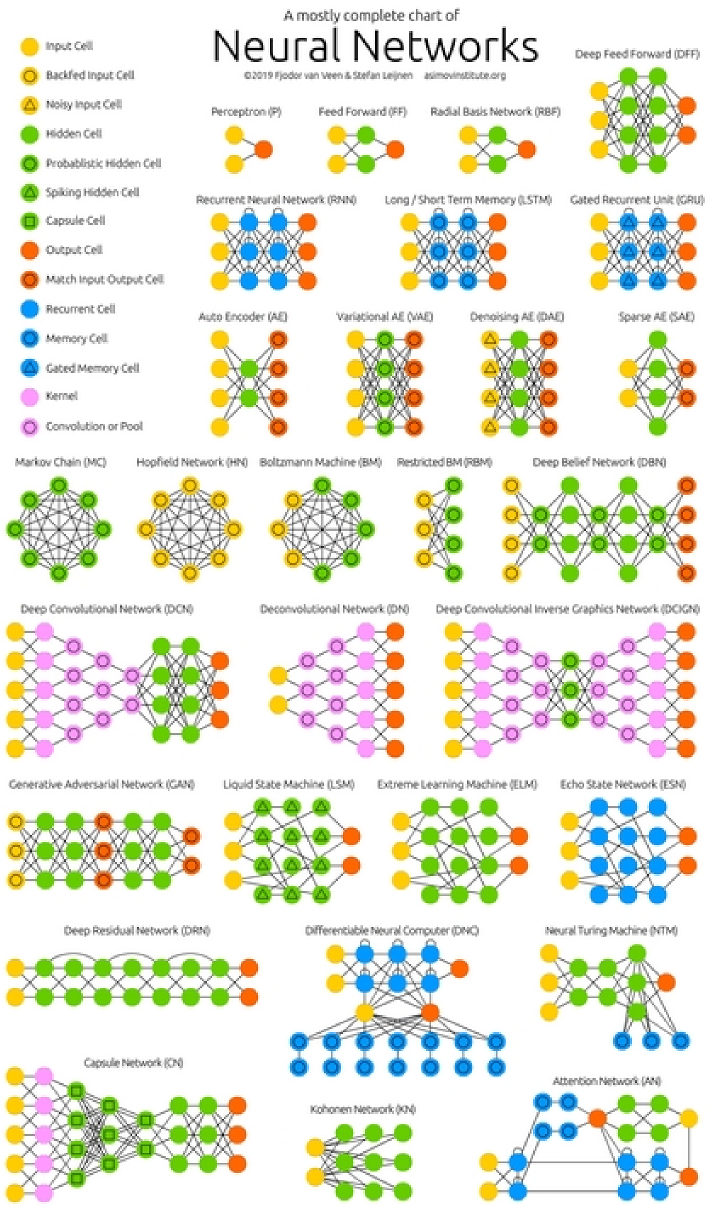
Today's Procedure

Start with neural nets, then XGB if enough time.

But first, there are two major categories of ML: supervised & unsupervised. In the first, we give the machine examples of data to classify and the correct answers which the machine uses to train. In the second, no answers.   Also, ML's are used to classify examples and to regress (fit, as in non-linear regression). We will streamline by referring to "classifiers".

Today we deal only with supervised classification.

There are many types of NN's:


A mostly complete chart of
Neural Networks
©2019 Fjodor van Veen & Stefan Leijnen   asimovinstitute.org
Input Cell
Backfed Input Cell
Noisy Input Cell
Hidden Cell
Probablistic Hidden Cell
Spiking Hidden Cell
Capsule Cell
Output Cell
Match Input Output Cell
Recurrent Cell
Memory Cell
Gated Memory Cell
Kernel
Convolution or Pool
Perceptron (P)
Feed Forward (FF)
Radial Basis Network (RBF)
Deep Feed Forward (DFF)
Recurrent Neural Network (RNN)
Long / Short Term Memory (LSTM)
Gated Recurrent Unit (GRU)
Auto Encoder (AE)
Variational AE (VAE)
Denoising AE (DAE)
Sparse AE (SAE)
Markov Chain (MC)
Hopfield Network (HN)
Boltzmann Machine (BM)
Restricted BM (RBM)
Deep Belief Network (DBN)
Deep Convolutional Network (DCN)
Deconvolutional Network (DN)
Deep Convolutional Inverse Graphics Network (DCIGN)
Generative Adversarial Network (GAN)
Liquid State Machine (LSM)
Extreme Learning Machine (ELM)
Echo State Network (ESN)
Deep Residual Network (DRN)
Differentiable Neural Computer (DNC)
Neural Turing Machine (NTM)
Capsule Network (CN)
Kohonen Network (KN)
Attention Network (AN)

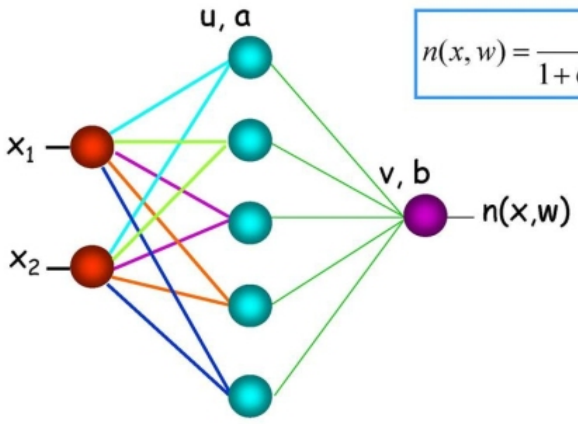
The Neural Network Zoo (download or get the poster).

from www.asimovinstitute.org/neural-network-zoo/

We will use saturated, feed-forward networks like this:

$$\boxed{f(x,w) = b + \sum_{j=1}^{H} v_j \tanh\left[a_j + \sum_{i=1}^{P} u_{ij} x_i\right]}$$

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



In this cartoon of a NN the flow of information starts at the inputs x1 & x2 from which the information is sent to the "hidden layer" (so called because not exposed to outside) where the sums over indices i and j are done, followed by the sum over the non-linear "activation function" tanh(.) and offset b. But for the tanh this would be a linear function. Note that most of the computation are dot products. The f(x,w) is a regression result which we convert to a "probability" using the logistic function n(.).

The cartoon shows a single hidden layer. A NN with more than one hidden layer is called "deep".

Given a NN, defined by its weights u_{ij}, a_j, v_j and b the calculation of its opinion wrt the features presented at x_i , is very fast.   This is called a "forward pass".

Most of ML is concerned with how to find the weights such that your NN produces accurate opinions.

There is a training procedure which finds useful weights; it essentially computes weights such that the Loss is minimized. Given a net with some (perhaps random) weights do:

1) Input a training example ("event" in HEP) into the x layer, one feature per input node.
2) Compute the NN's opinion (e.g. do a forward pass) f(x, w).
3) Compute the error (Loss) $L = (f(x,w) - \text{label})^2$
4) Compute the derivatives (matrix of partials) of dL/dw
5) Change each weight $w_i^{\text{NEXT}} \Leftarrow\!= w_i - \eta\, \partial L/\partial w_i$; where η (the "learning rate") is small ~0.01 ... 0.001
6) Go to 1), or stop if happy with size of error (Loss).

This procedure can be made to be computationally efficient for even multi-layer NN (google "backpropagation").

Overtraining, generalization, OOB error, …

It is the case that many (most) classifiers if trained long enough can learn the training set perfectly, but when presented with a new (not from training set, aka Out Of Bag) instance will do poorly. This situation is called "overtraining".

Most of your effort will be intended to increase the generalization ability of your classifier.

Some common terms

Forward pass: one example is passed through the network the error of the output layer is computed.

Backward pass: update the weights (credit assignment) based on the error hidden nodes are linked to the error not directly but by means of the nodes of the next layer. Starting at the output layer, the error is propagated backwards through the network, layer by layer. This is done by recursively computing the local gradient of each neuron.

Epoch: application of the update rule to all examples of the training set. Execution of the learning algorithm terminates when the weights do not change after one epoch.

Weight seeding: weights should be initialized seeded with small random weights.

Stochastic gradient descent: one or few randomly picked instances for weights update.

Minibatch: update weights after processing small, random batches of instances.

Loss function: some convenient function of the error between label and NN's opinion (output).

Hyperparameters: Learning rate, momentum, batch size, number of epochs, number of hidden layers and units, dropout fraction. *Use what works!* Optimise with grid or random search.

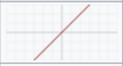
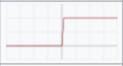
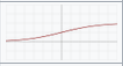
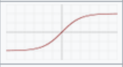
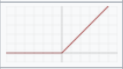
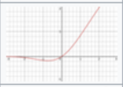
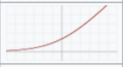
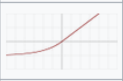
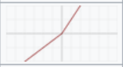
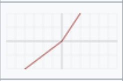
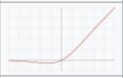
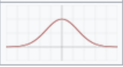
Overfitting (overtraining): Avoid overfitting & check w. validation set. Leave "test"; "holdback" set unsullied for final measurement of generalization level.

Crossvalidation: Set aside a fraction of the training data to use avalidation; repeat with randomly selected instances. (to measure NN quality)

Dropout: Randomly drop connections between nodes of NN. (improve generalization)

Data prep: It has been found that ANN's do better if the input data is "normalized", e.g. the training set is scaled such that the mean is 0 and 1 standard deviation width. Each input variable must be idependently normalized. *For test data the inputs must use the same scaling as the training dataset.* How does this impact your data prep?

Activation function: ANN's usually converge faster with ReLU than tanh; especially multi-layer ANN's.

Name	Plot	Function, $f(x)$	Derivative of f, $f'(x)$	Range	Order of continuity
Identity		x	1	$(-\infty,\infty)$	C^∞
Binary step		$\begin{cases}0 & \text{if } x<0\\ 1 & \text{if } x\geq 0\end{cases}$	$\begin{cases}0 & \text{if } x\neq 0\\ \text{undefined} & \text{if } x=0\end{cases}$	$\{0,1\}$	C^{-1}
Logistic, sigmoid, or soft step		$\sigma(x)=\frac{1}{1+e^{-x}}$[1]	$f(x)(1-f(x))$	$(0,1)$	C^∞
Hyperbolic tangent (tanh)		$\tanh(x)=\frac{e^x-e^{-x}}{e^x+e^{-x}}$	$1-f(x)^2$	$(-1,1)$	C^∞
Rectified linear unit (ReLU)[9]		$\begin{cases}0 & \text{if } x\leq 0\\ x & \text{if } x>0\end{cases}$ $=\max\{0,x\}=x\mathbf{1}_{x>0}$	$\begin{cases}0 & \text{if } x<0\\ 1 & \text{if } x>0\\ \text{undefined} & \text{if } x=0\end{cases}$	$[0,\infty)$	C^0
Gaussian Error Linear Unit (GELU)[4]		$\frac{1}{2}x\left(1+\text{erf}\left(\frac{x}{\sqrt{2}}\right)\right)$ $=x\Phi(x)$	$\Phi(x)+x\phi(x)$	$(-0.17\ldots,\infty)$	C^∞
Softplus[10]		$\ln(1+e^x)$	$\frac{1}{1+e^{-x}}$	$(0,\infty)$	C^∞
Exponential linear unit (ELU)[11]		$\begin{cases}\alpha(e^x-1) & \text{if } x\leq 0\\ x & \text{if } x>0\end{cases}$ with parameter α	$\begin{cases}\alpha e^x & \text{if } x<0\\ 1 & \text{if } x>0\\ 1 & \text{if } x=0 \text{ and } \alpha=1\end{cases}$	$(-\alpha,\infty)$	$\begin{cases}C^1 & \text{if } \alpha=1\\ C^0 & \text{otherwise}\end{cases}$
Scaled exponential linear unit (SELU)[12]		$\lambda\begin{cases}\alpha(e^x-1) & \text{if } x<0\\ x & \text{if } x\geq 0\end{cases}$ with parameters $\lambda=1.0507$ and $\alpha=1.67326$	$\lambda\begin{cases}\alpha e^x & \text{if } x<0\\ 1 & \text{if } x\geq 0\end{cases}$	$(-\lambda\alpha,\infty)$	C^0
Leaky rectified linear unit (Leaky ReLU)[13]		$\begin{cases}0.01x & \text{if } x<0\\ x & \text{if } x\geq 0\end{cases}$	$\begin{cases}0.01 & \text{if } x<0\\ 1 & \text{if } x\geq 0\end{cases}$	$(-\infty,\infty)$	C^0
Parameteric rectified linear unit (PReLU)[14]		$\begin{cases}\alpha x & \text{if } x<0\\ x & \text{if } x\geq 0\end{cases}$ with parameter α	$\begin{cases}\alpha & \text{if } x<0\\ 1 & \text{if } x\geq 0\end{cases}$	$(-\infty,\infty)$[2]	C^0
Sigmoid linear unit (SiLU,[4] Sigmoid shrinkage,[15] SiL,[16] or Swish-1[17])		$\frac{x}{1+e^{-x}}$	$\frac{1+e^{-x}+xe^{-x}}{(1+e^{-x})^2}$	$[-0.278\ldots,\infty)$	C^∞
Mish [18]		$x\tanh(\ln(1+e^x))$	$\frac{(e^x(4e^{2x}+e^{3x}+4(1+x)+e^x(6+4x)))}{(2+2e^x+e^{2x})^2}$	$[-0.308\ldots,\infty)$	C^∞
Gaussian		e^{-x^2}	$-2xe^{-x^2}$	$(0,1]$	C^∞

We'll use Scikit-learn because it has decent defaults. For more flexibility one can later use Keras, pyTorch, TMVA, R,….

Here are the first five columns of the first 2 lines of the training dataset of ~24,000 *instances*:

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 <<== column #

1.0000000 -0.2753248 0.0579663 0.4022882 -0.2108261 0.7982173 …. <<==first feature line

0.0000000 0.3018720 -0.7348565 -1.4450299 0.6842730 -1.2888023 …. <<== second feature line

```
  ^^^^^            ^^^ features (23 cols of features)
class  label    (just two labels 0 or 1)
```

These data have already been “standardized” by transforming to mean and standard deviation.