An introduction to Neural Networks in HEP and developments for their application to new physics searches

Adam Elwood

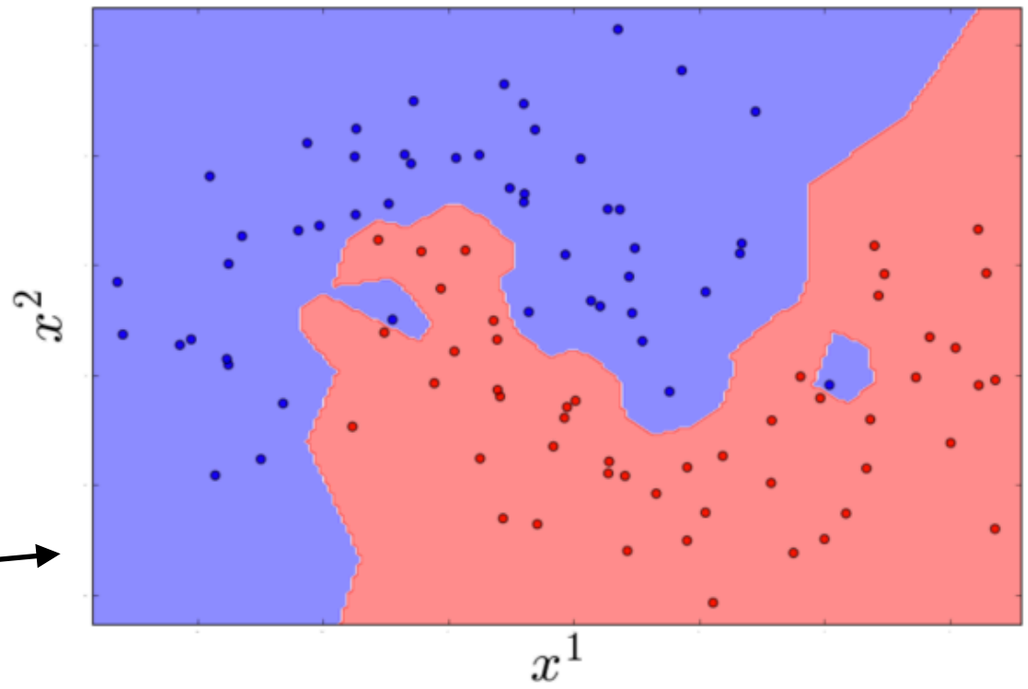
adam.elwood@desy.de

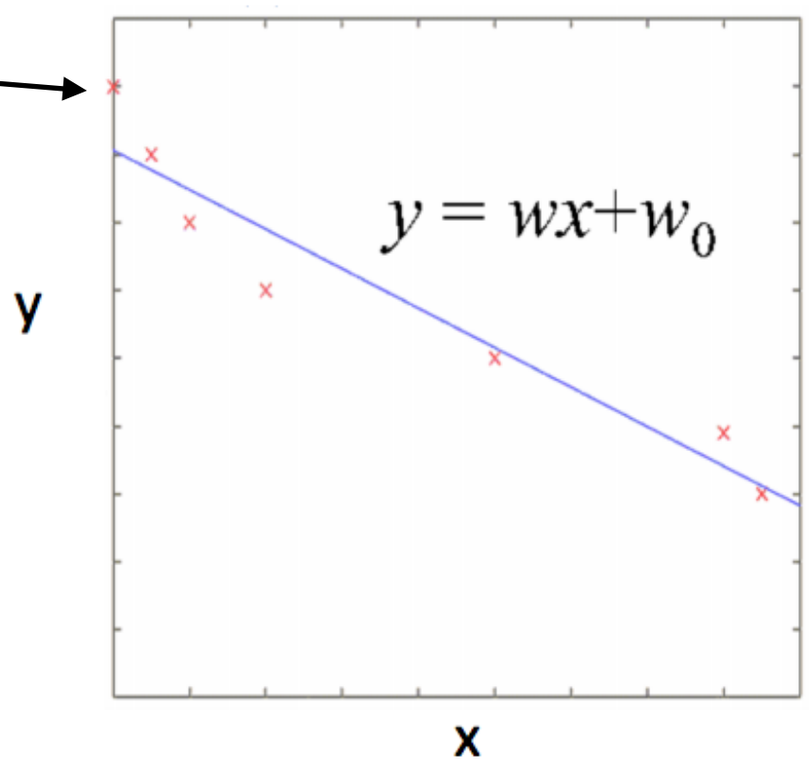
Outline

- Machine learning basics
- Introduction to neural networks
  - What they are
  - How to optimise them
  - Common architectures
- Applications to HEP
  - Object reconstruction
  - Physics analyses
- Direct optimisation of discovery significance
  - New loss functions
  - Results with representative SUSY models

Machine learning basics

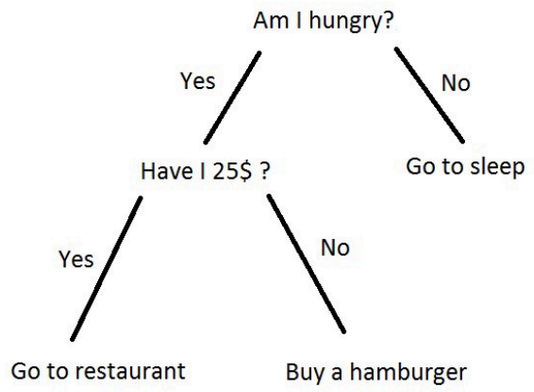
- Build a mathematical model to represent inherent characteristics of a dataset
  - While not explicitly programming it
- Use it to make predictions based on new data, typically:
  - Classification
  - Regression
- I.e. predicting an objective, "y", given a vector of input features (variables), "**x**"
- Can exploit the correlations of inputs to gain a sophisticated prediction that would be very difficult to design otherwise

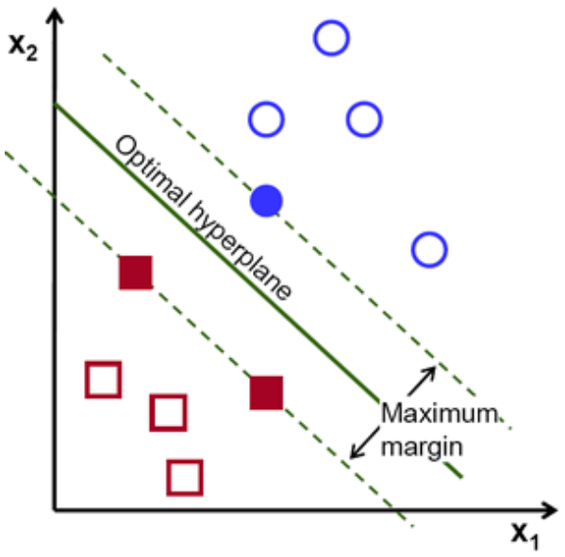


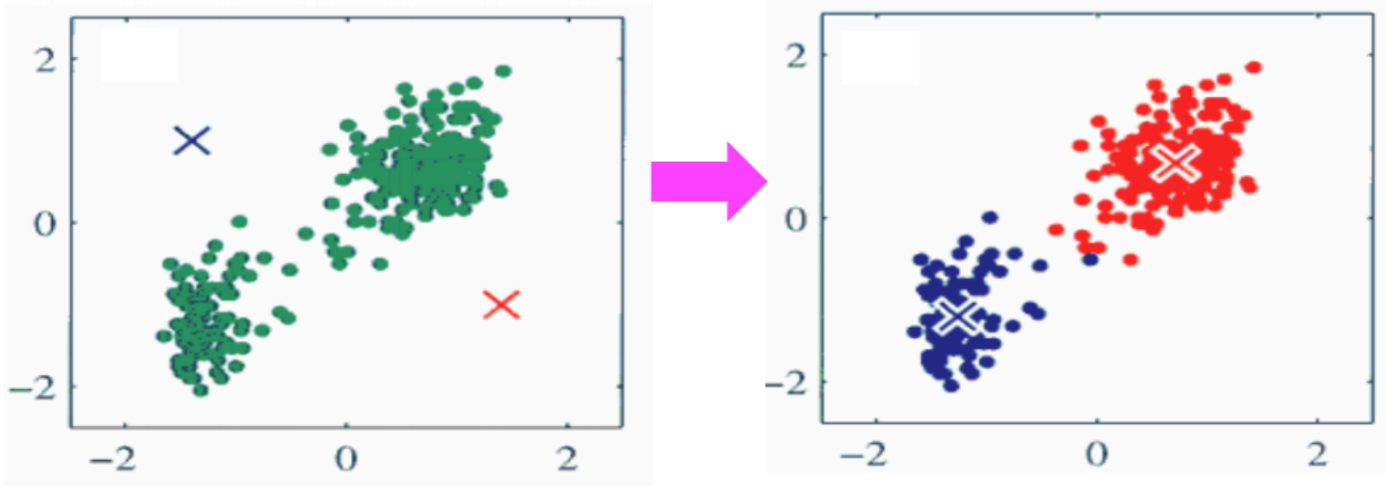


Machine learning types

- **Supervised learning** (training data required), e.g.:
  - Decision trees
    - Including the boosted decision tree (BDT), random forest
  - Support vector machines
  - Neural networks
- **Semi-supervised learning** (some training data required), e.g.:
  - Autoencoders for anomaly detection
- **Unsupervised learning** (no training data required) e.g.:
  - K Nearest Neighbour (KNN) clustering
- **Reinforcement learning** (data delayed)

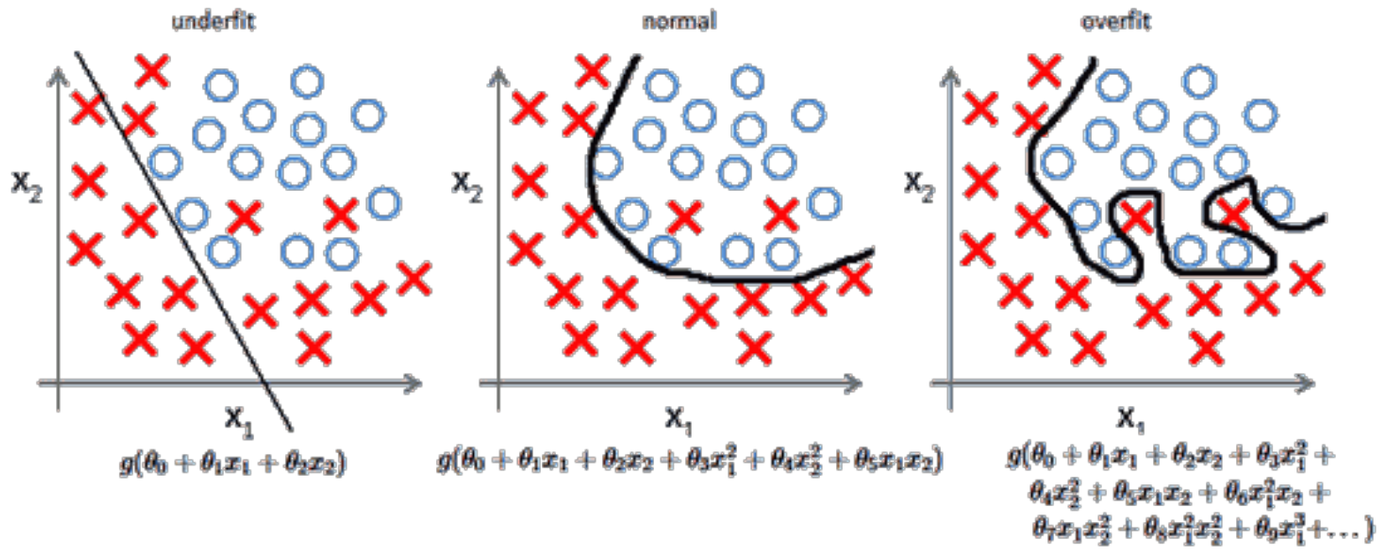




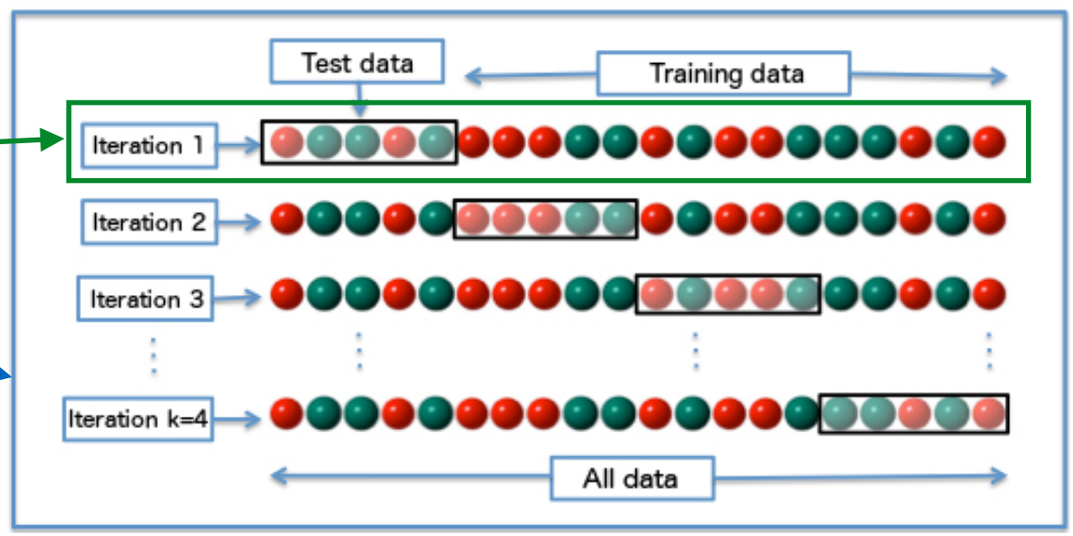


Basics of supervised learning

- Most HEP use cases revolve around supervised learning
  - Access to lots of labelled data (MC simulation)
- Want to build a model with good predictive power without overfitting to the training data,

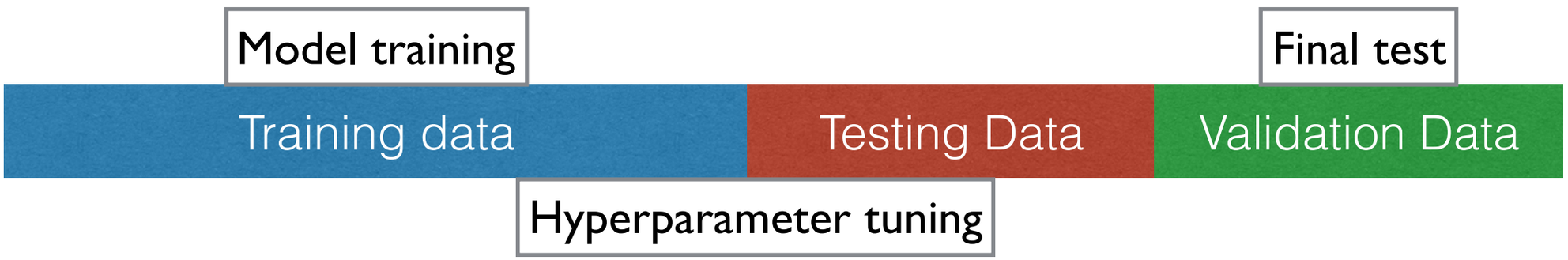


- Use an independent test dataset
  - Holdout
  - K-fold cross-validation



Hyperparameter optimisation

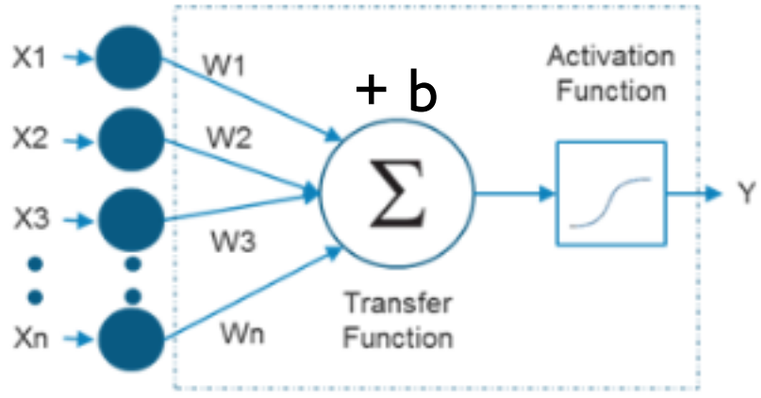
- Usually lots of parameters associated with machine learning models
  - e.g. tree depth, min leaf size for BDT
- Known as **'hyperparameters'** and typically require tuning to a particular problem and dataset
- Can lead to tuning towards a test set, i.e. information leakage
- Good practise to maintain an extra validation set to avoid biasing your test scores after hyperparameter optimisation



- Suite of dedicated algorithms to search through hyperparameter space
  - Grid search, bayesian optimisation, particle swarm

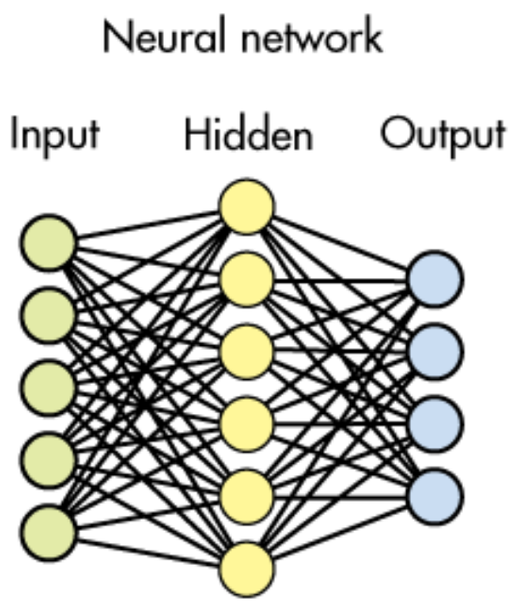
Neural networks basics

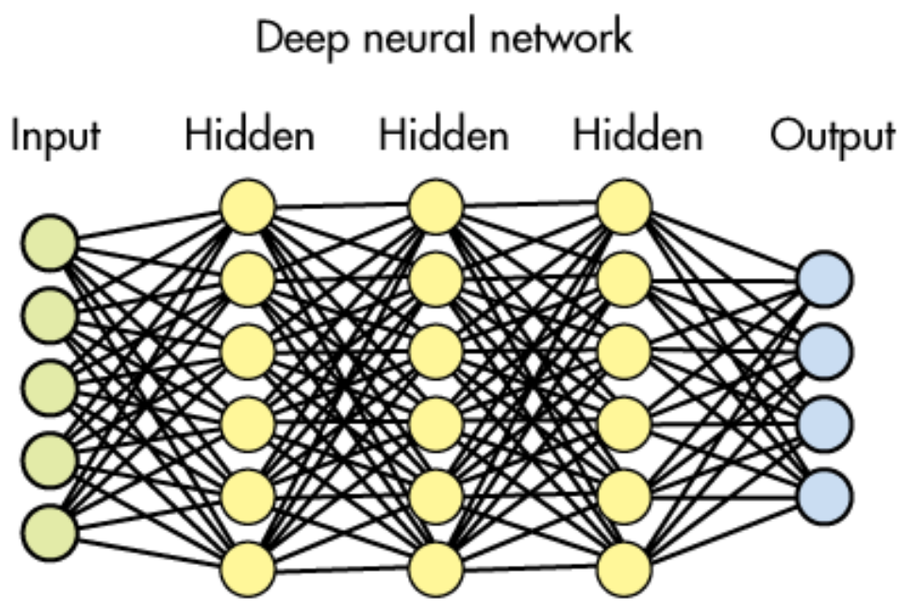
- Build a network of nodes with features of data as input and N nodes for output (where N is task dependent)
- Between them have an array of neurons arranged into layers with a configurable width



Schematic for a neuron in a neural net

- Can approximate any function





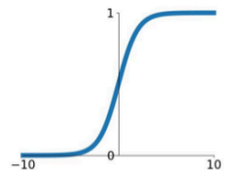
Useful activation functions

- For the internal neurons:
  - Traditionally use sigmoid or tanh
  - Now recommended to use ReLU or Leaky ReLU
- For the output neurons:
  - Regression
    - One output node with, typically, a linear activation function (i.e. return the raw sum)
  - Binary classification
    - One output node with a sigmoid activation function (y<0.5 is first class, y>0.5 is second class)
  - N class classification
    - N nodes with a soft-max activation function (a series of sigmoids that all must add up to one)

Activation Functions

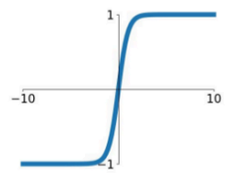
Sigmoid

$\sigma(x) = \frac{1}{1+e^{-x}}$



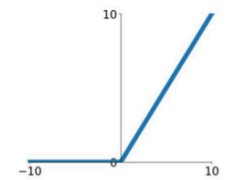
tanh

$\tanh(x)$



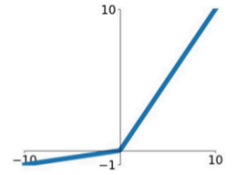
ReLU

$\max(0, x)$



Leaky ReLU

$\max(0.1x, x)$

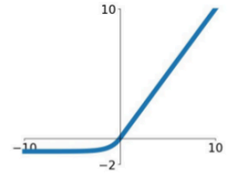


Maxout

$\max(w_1^T x + b_1, w_2^T x + b_2)$

ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



Loss functions

- Define loss (error, cost, objective) functions to quantify how well the network predictions, $\hat{y}$, models the real value from the data, y
- These loss functions typically signify the total error in the representation and training the network revolves around minimising the loss function
- The appropriate function depends on the use case:
  - For regression typically use mean squared error

$$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2.$$

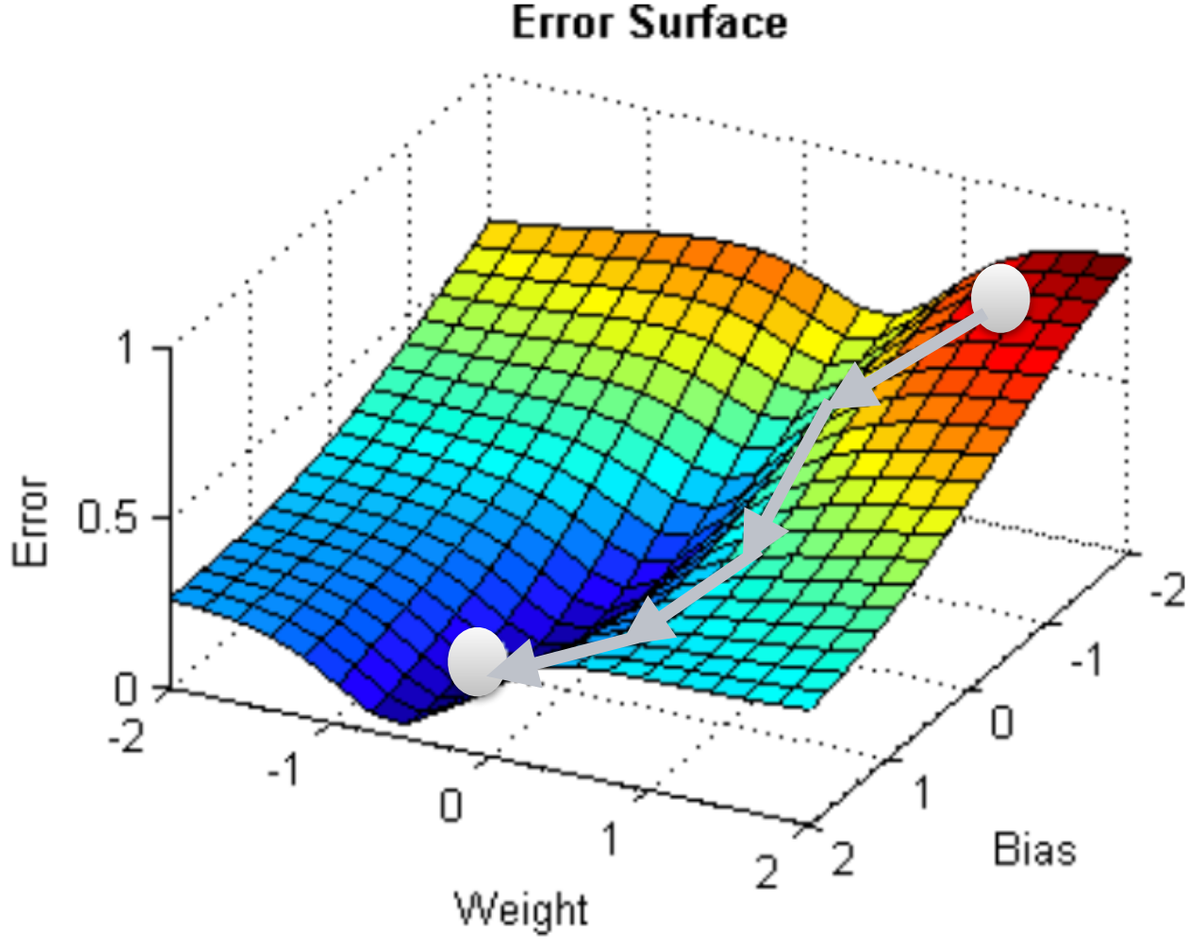
  - For classification typically use cross entropy (equivalent to minimising negative log likelihood)

$$C = -\frac{1}{n}\sum_{x}\left[y \ln \hat{y} + (1-y)\ln(1-\hat{y})\right],$$

    - Allows interpretation sigmoid or softmax output as probability of class
  - Many others available for more specific use cases

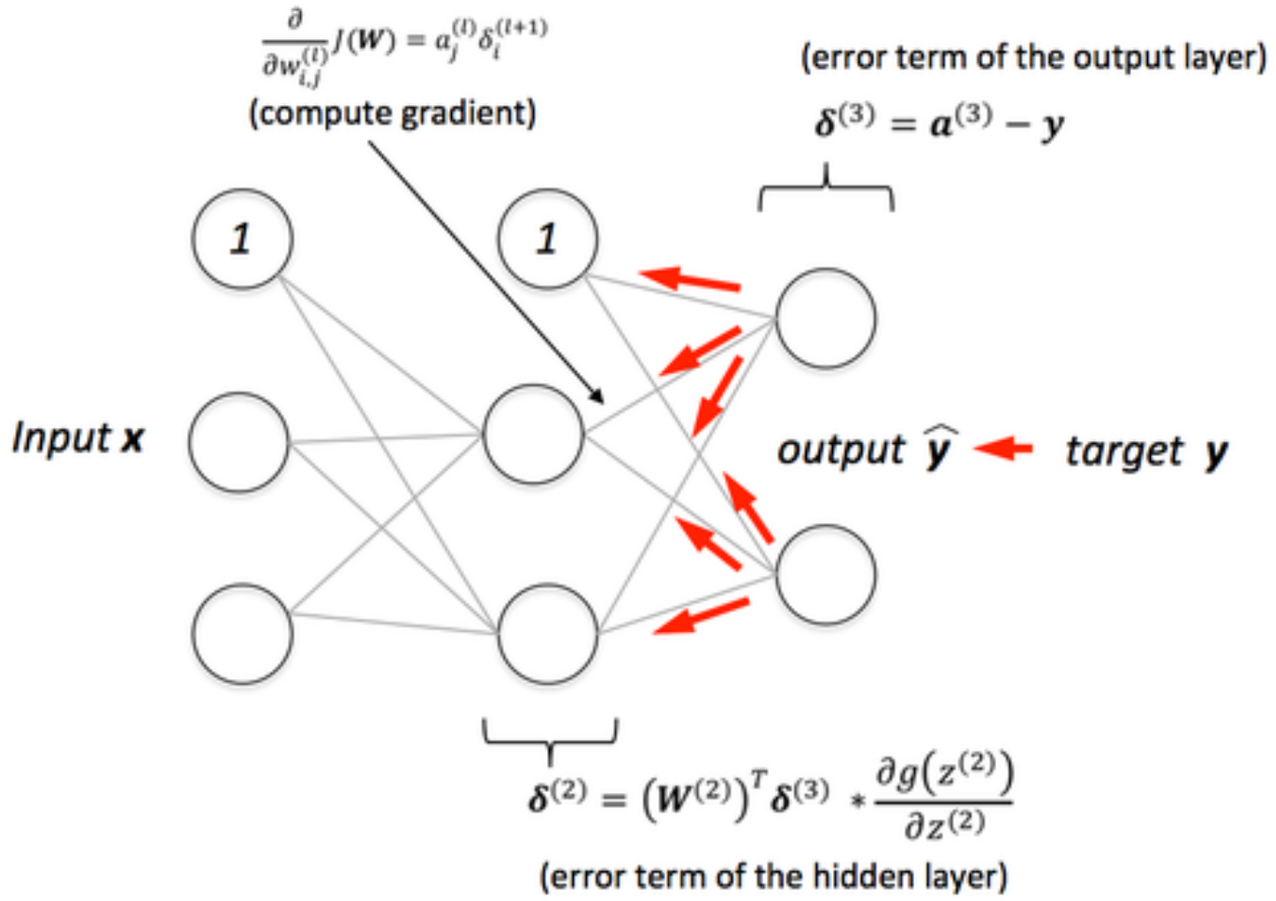
Network optimisation

- To optimise the network want to choose the set of weights and biases that minimise the loss function (or 'error')
- Can carry out a gradient descent in the weight and bias parameter space
- E.g. a 2D optimisation in a one neuron network:



Backpropagation

- To carry out the gradient descent need to know the gradient of the loss function with respect to the weights and biases in the network
- To find this we calculate how a change in the loss propagates backwards into changes in the weights and biases of the network with the chain rule



- Backpropagation is actually a special version of a computational technique known as "automatic differentiation"

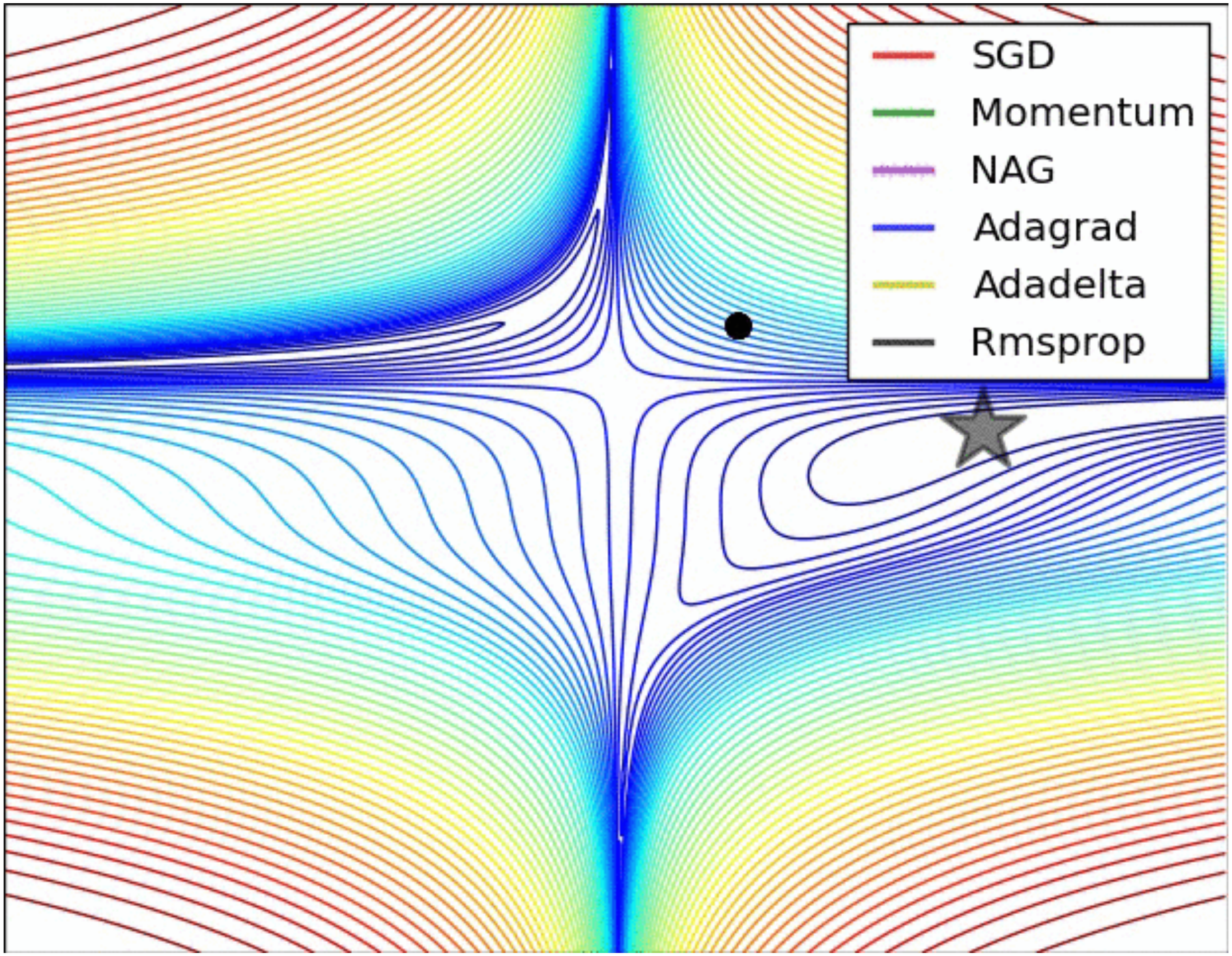
More detail: http://neuralnetworksanddeeplearning.com/chap2.html

Methods of gradient descent

- With the basic tools in place it is desirable to speed up the process of gradient descent and find ways to escape local minima
- Can choose number of events, the **'batch'**, used to calculate each gradient step
  - Using a subset of events gives fluctuations that can help escape local minima
  - Ultimately a tradeoff between accuracy and speed of each step
  - This gives you a basic algorithm **'Stochastic Gradient Descent'**
- Different algorithms to follow the descent
  - Have to choose the size of the steps in the gradient (the **'learning rate'**), trade off between overstepping and going too slowly
  - Can use tricks to speed up the process, such as stepping further when travelling in the same direction for a lot of iterations, **'momentum'**, and correcting for over steps **'Nesterov accelerated gradient (NAG)'**
  - This leads to new algorithms such as **'Adam'** that exploit these techniques to speed up training time

More detail: https://towardsdatascience.com/types-of-optimization-algorithms-used-in-neural-networks-and-ways-to-optimize-gradient-95ae5d39529f

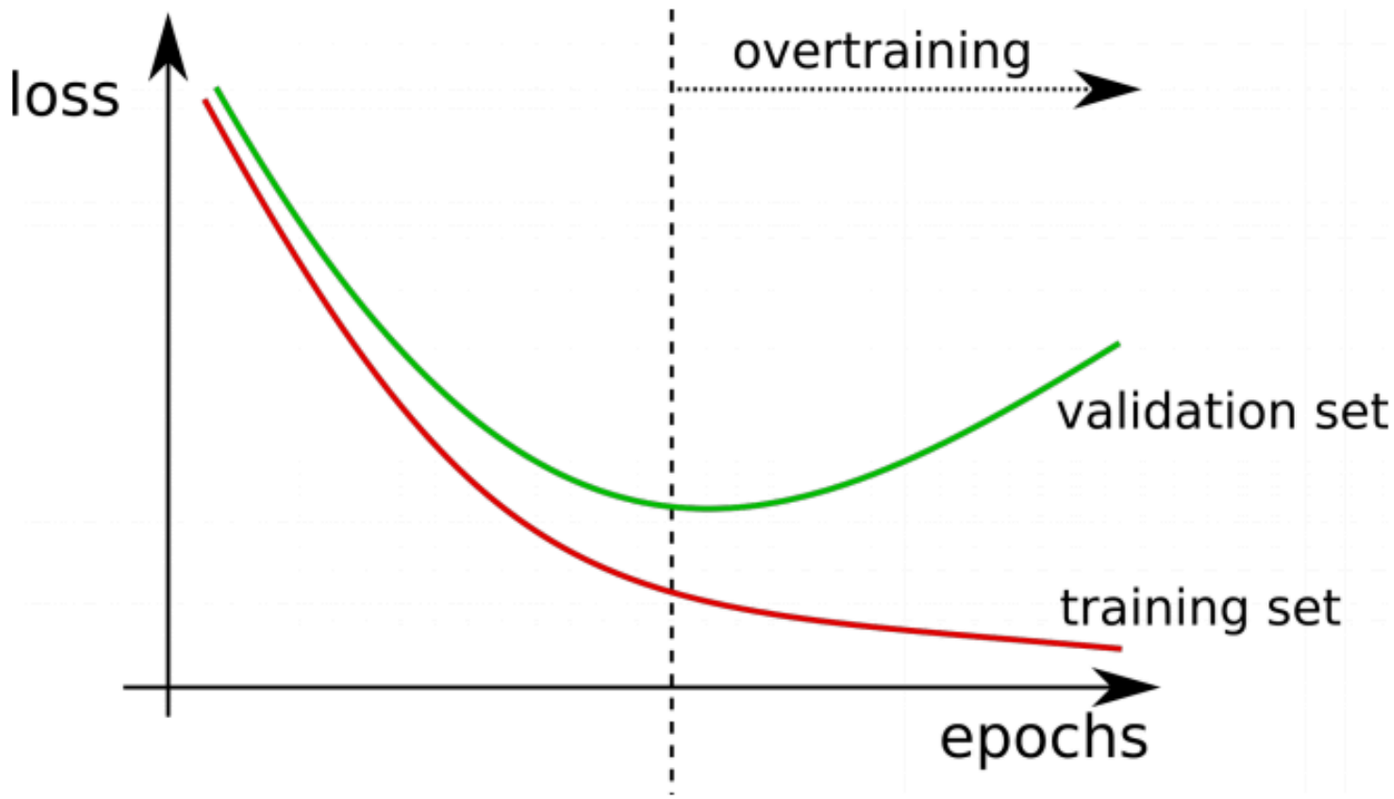
Methods of gradient descent



More detail: https://towardsdatascience.com/types-of-optimization-algorithms-used-in-neural-networks-and-ways-to-optimize-gradient-95ae5d39529f

Training neural networks

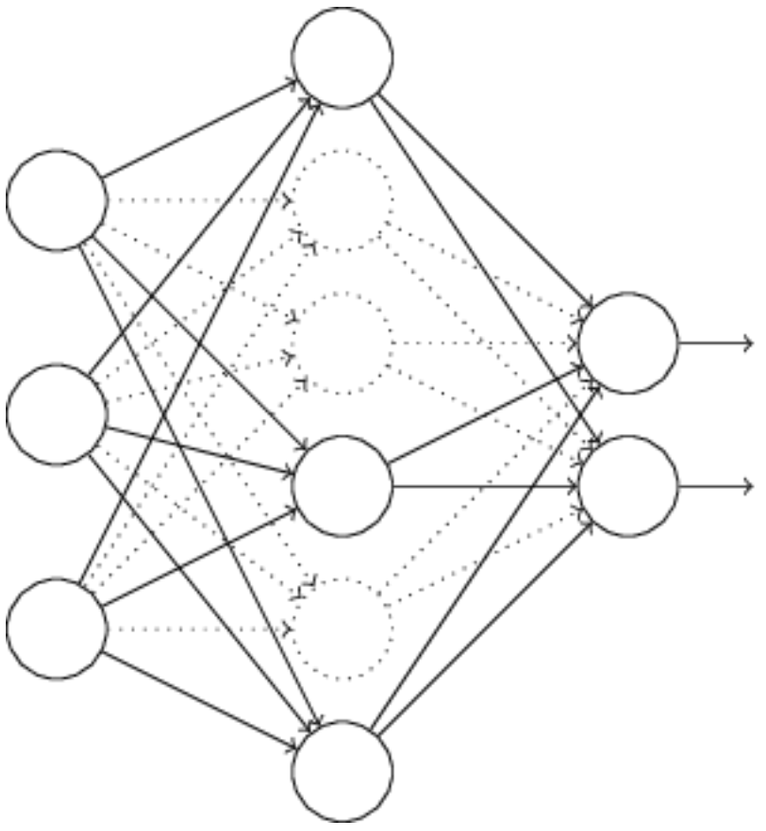
- After choosing a batch size and optimiser you can train the network
  - Default batch size in lots of libraries is 32, often stick to powers of 2 for hardware optimisation
  - Adam, with the default parameters, works well in most cases
- Carry out the gradient descent, updating the gradient until a persistent minimum is reached
  - Batches of events are randomly picked from the test data, taking events out of the pool for subsequent batches
  - When all the events in the test set have been sampled, one **‘epoch’** of training is complete



Other considerations

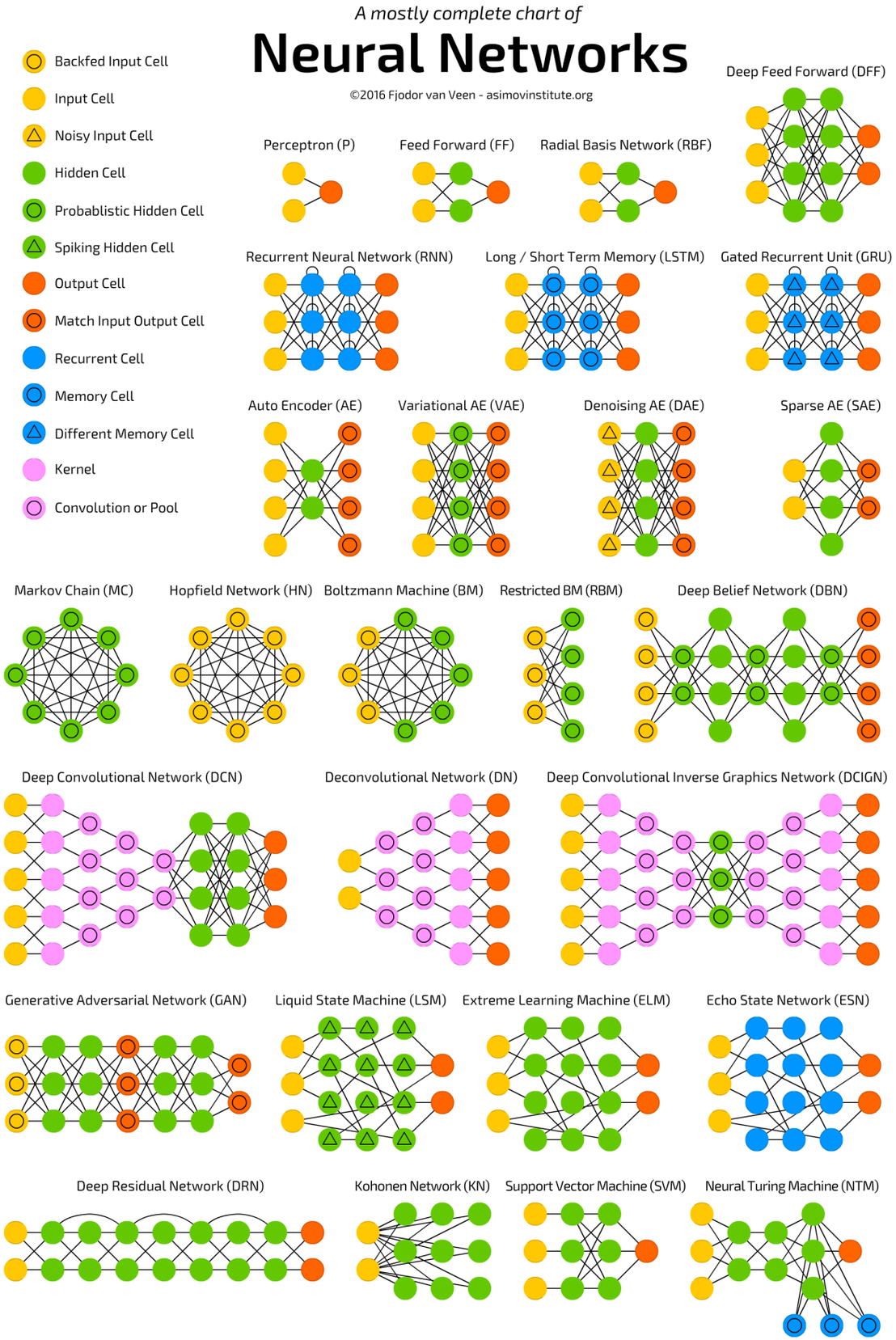
- A suite of other techniques and considerations exist to solve problems such as overtraining and disappearing gradients, e.g.:
  - Standardise the inputs to have mean 0 and variance 1
  - L1 and L2 regularisation
    - Prevent the weights becoming too large by putting a penalty on the loss function for large network weights
  - Dropout regularisation
    - Prevent overtraining by removing a subset of nodes in each training step
  - Tweaking weight and bias initialisation
  - Batch normalisation
- Can ultimately treat all these things as hyperparameters to optimise

Dropout



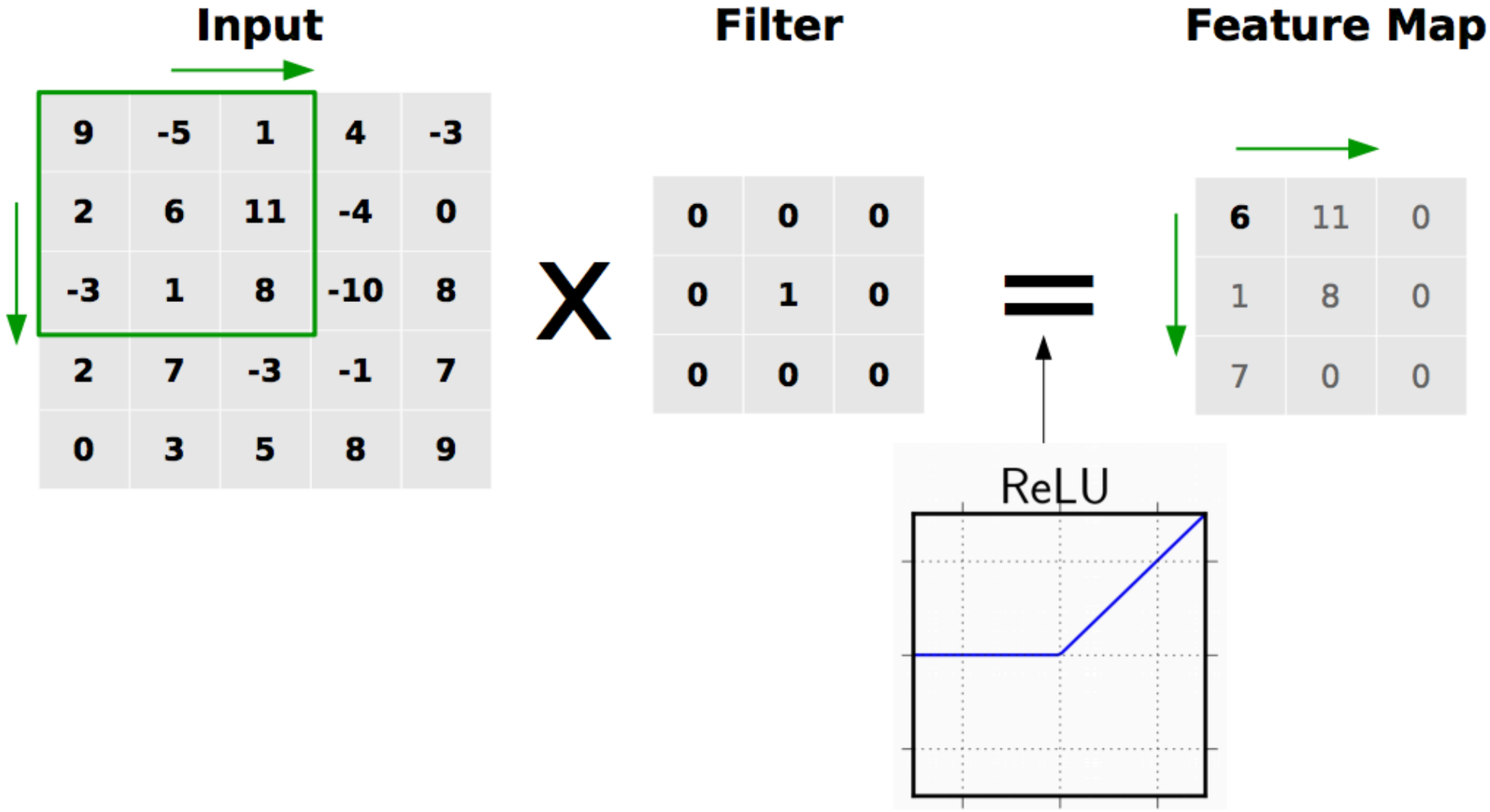
Different architectures

- Many architectures are available that determine the connection of different nodes
- Most basic is the dense (feed forward) neural network where every node is connected to every other node in the next layer
- In practise the big gains from neural networks occur when appropriate architecture is chosen to take account of a structure and symmetry in the data
  - Dense neural networks aren't a magic bullet….



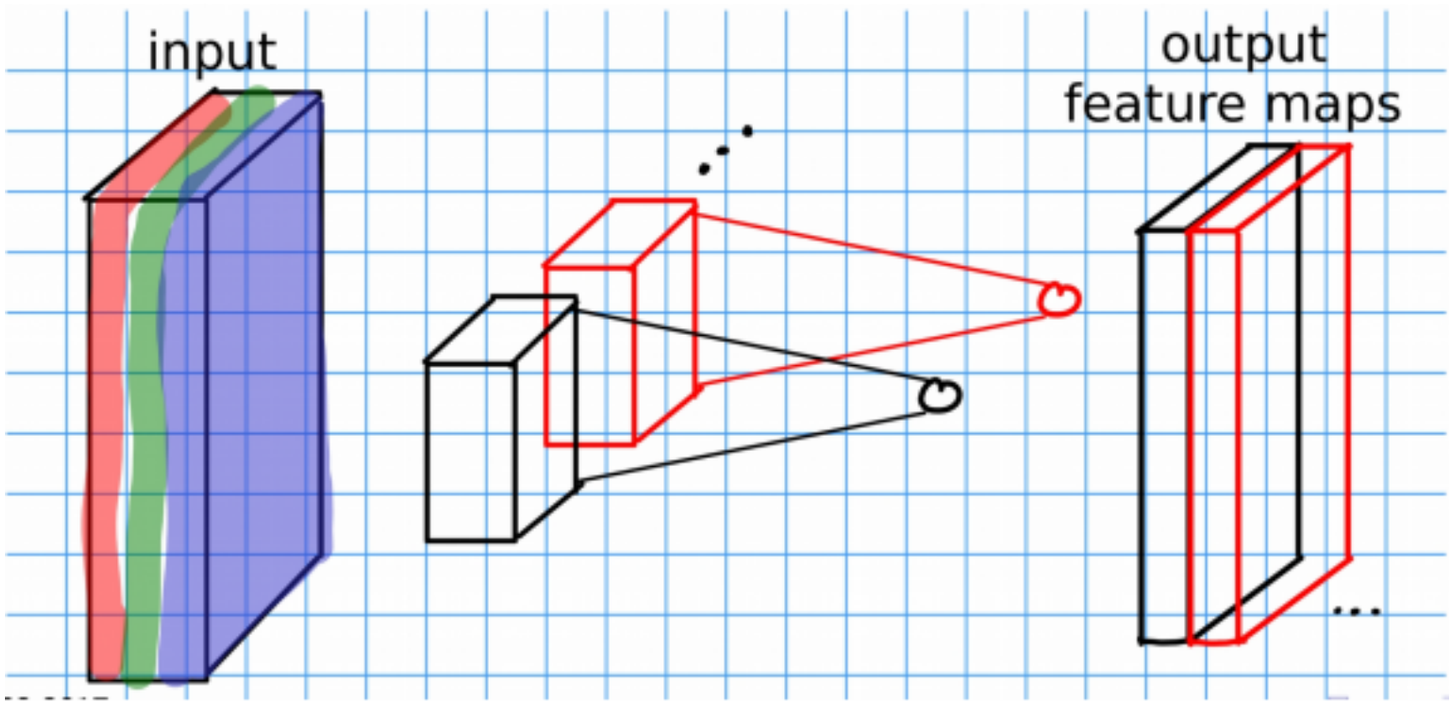
Convolutional neural networks

- Many tasks have a translationally invariant input
  - Image recognition based on pixels as input
- Convolutional networks scan filters with learnable parameters over the input, allowing them to learn translational invariant features

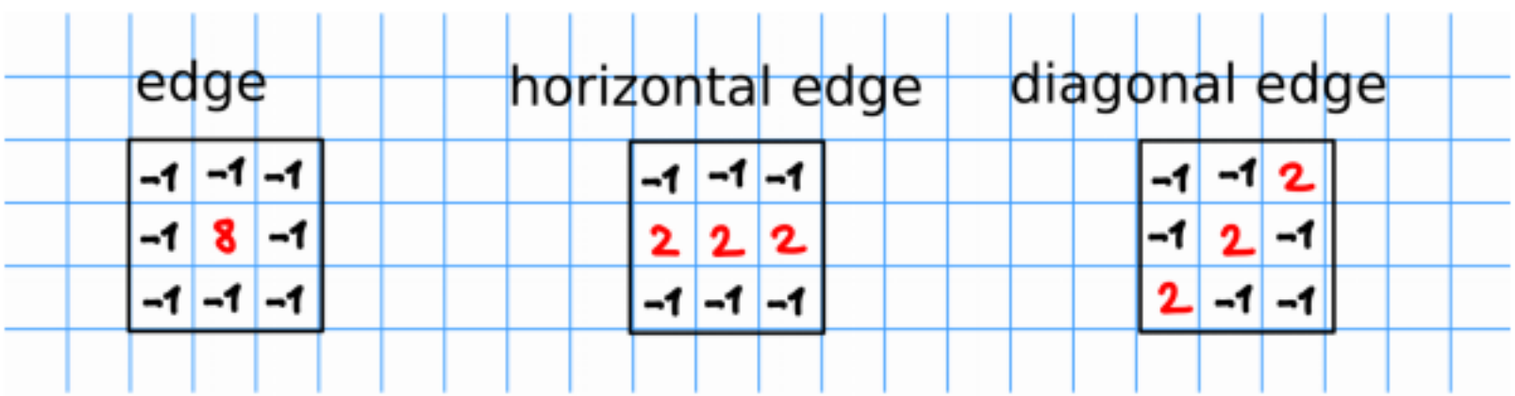


Convolutional neural networks

- Applied to a full 2x2 image with 3 colours:

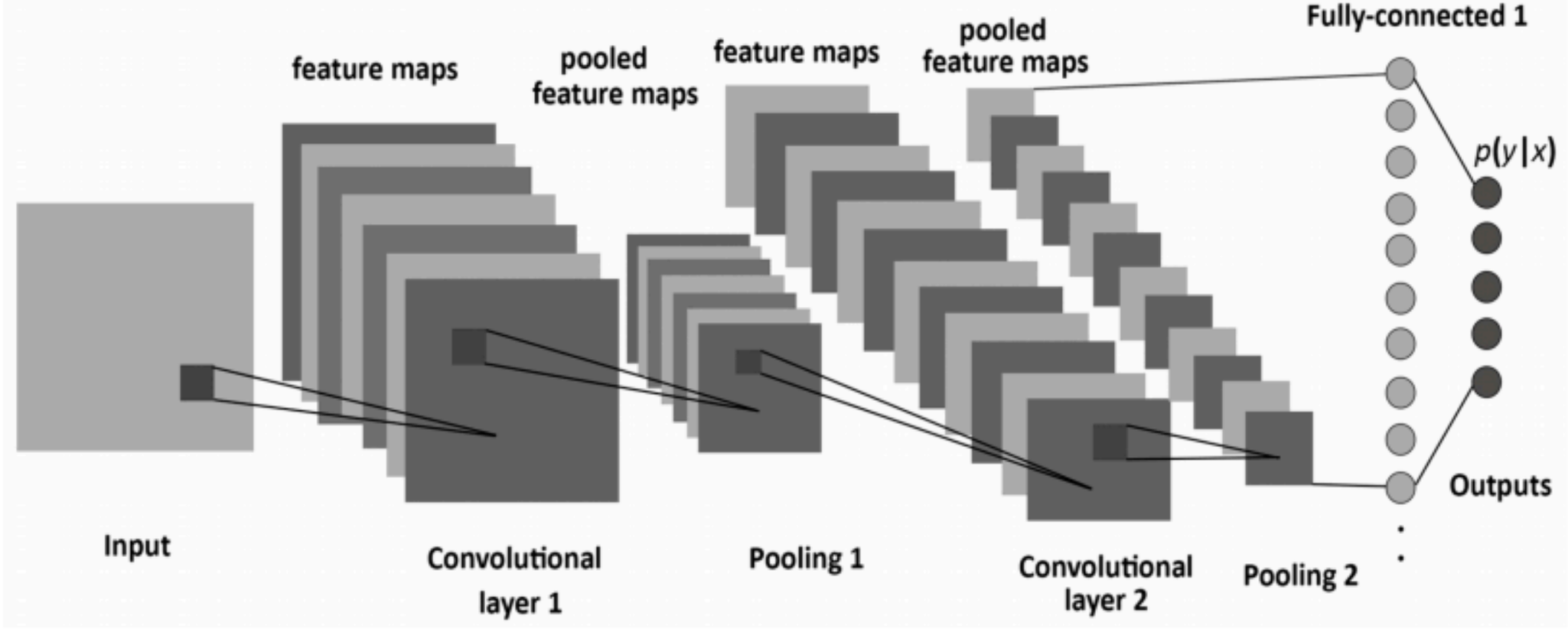


- Example features that could be learned:



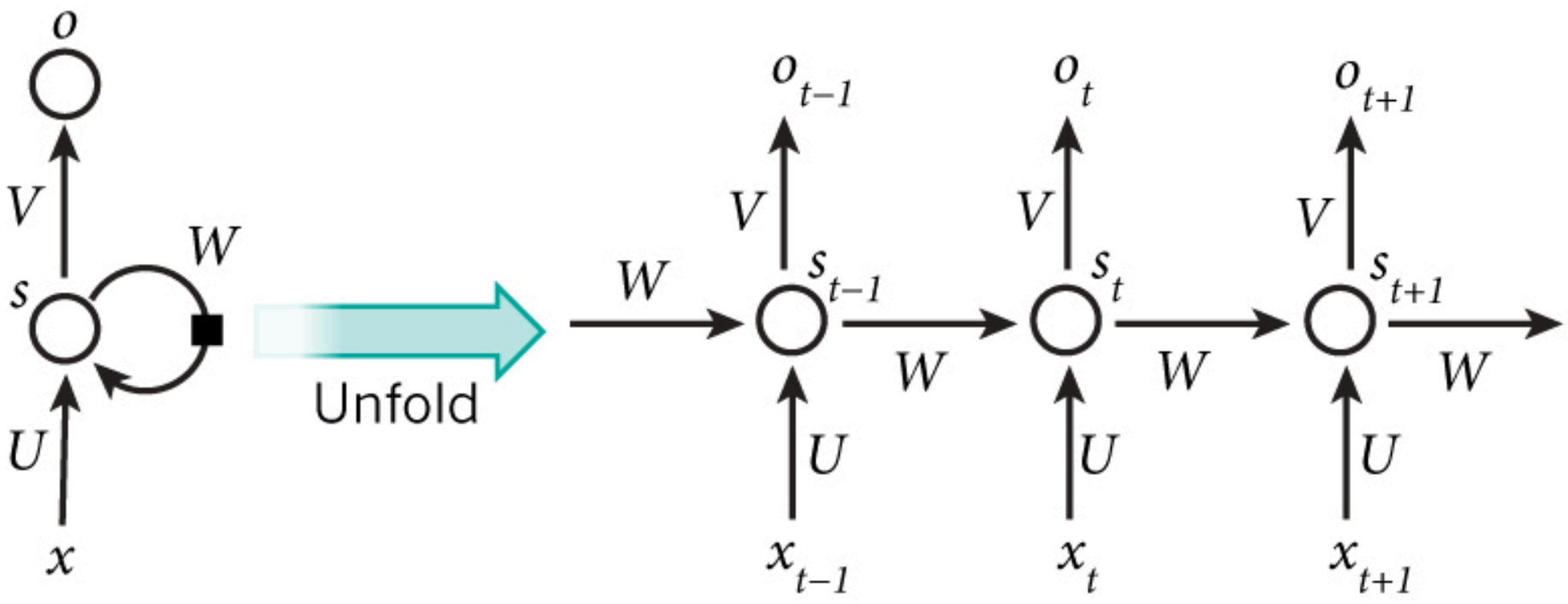
Convolutional neural networks

- After features are learned often perform 'pooling', e.g. just taking the maximum value in an area
- Note the use of 1x1 convolutional filters to expand or compress feature space
- Example of a complete convolutional network for classification:



Recurrent neural networks

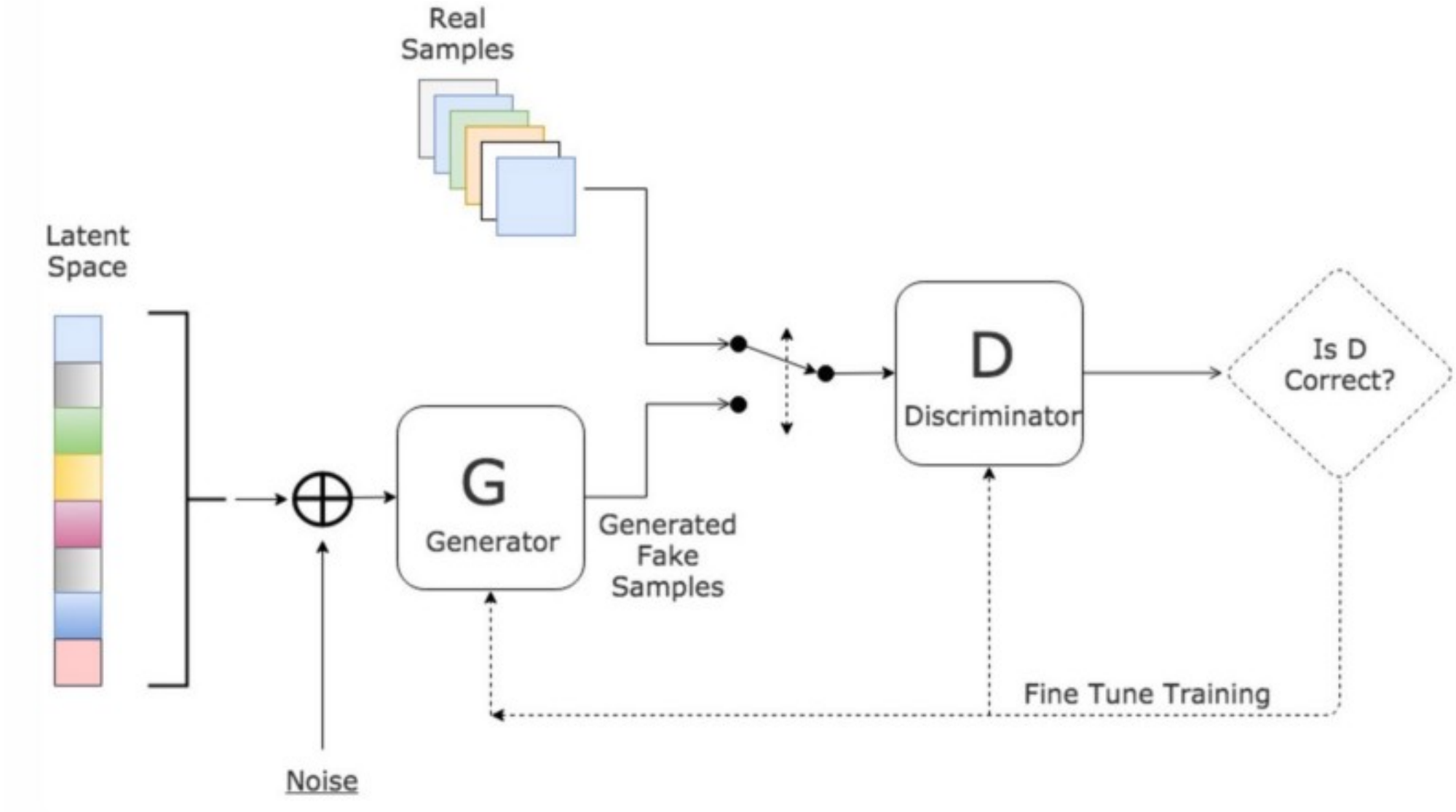
- RNNs are useful when the inputs have a sequential structure
  - Natural language processing
  - Ordered particle list (e.g. by p_T)
- RNN neurons have an internal state that remembers previous data passing through them
- Long Short Term Memory networks (LSTMs) are a popular variety of RNNs that can learn long term dependencies better than vanilla RNNs



http://www.wildml.com/2015/09/recurrent-neural-networks-tutorial-part-1-introduction-to-rnns/
http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Generative adversarial networks (GANs)

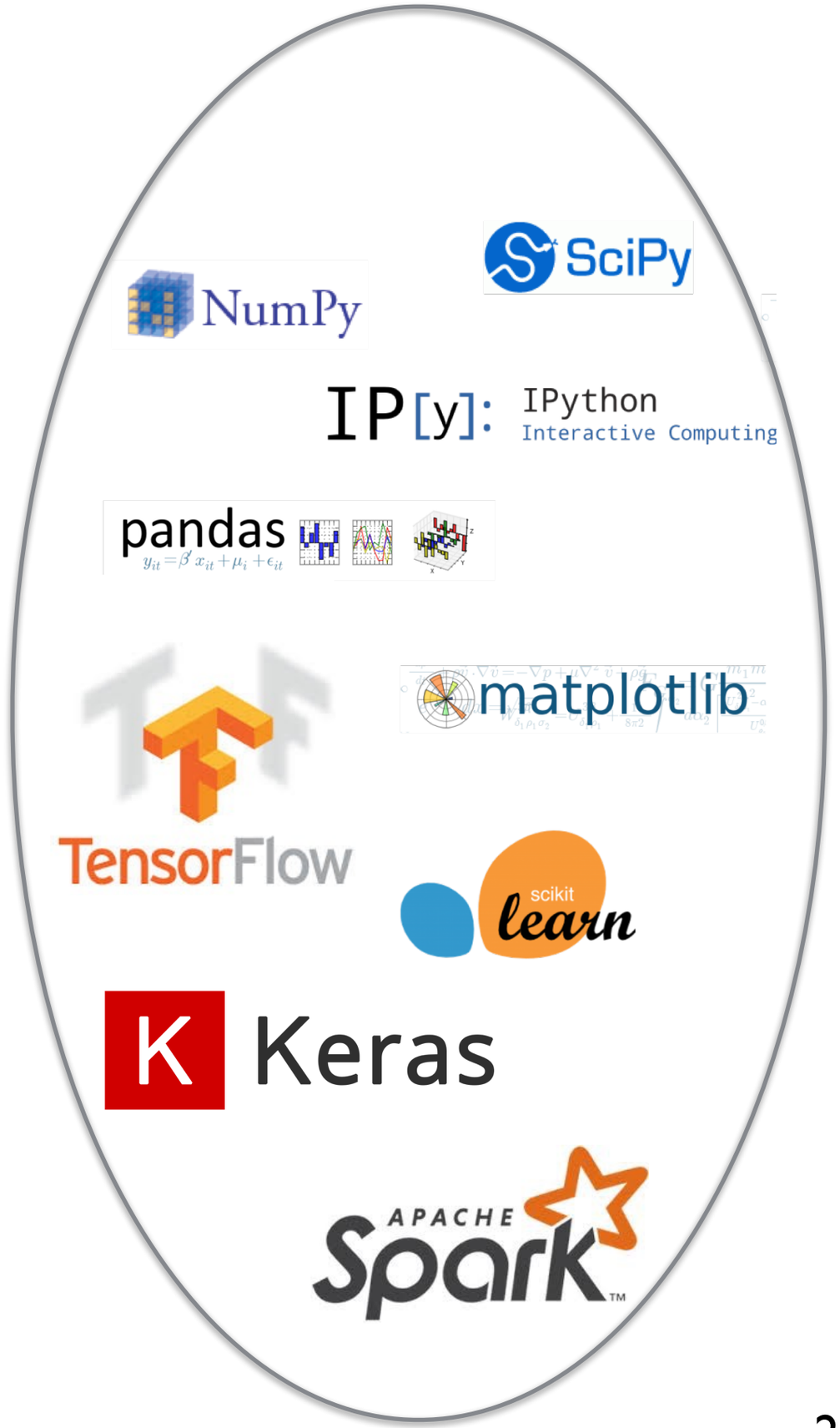
- Relatively new development in neural networks shows great potential for randomly generating realistic looking samples (e.g. images)
- Train two networks concurrently, a generator that tries to make fake samples and a discriminator that tries to tell the difference



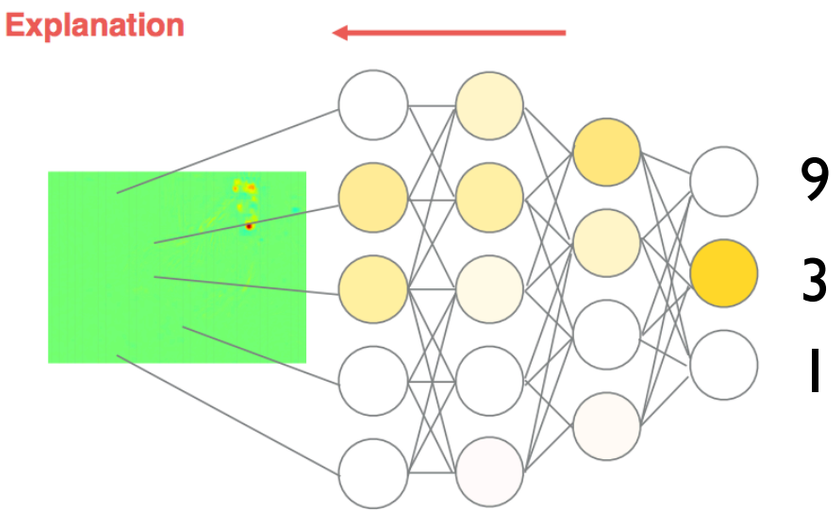
https://www.datasciencecentral.com/profiles/blogs/generative-adversarial-networks-gans-engine-and-applications

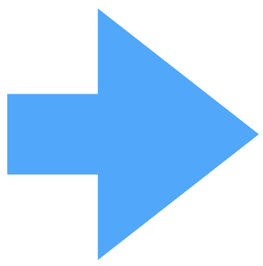
Software

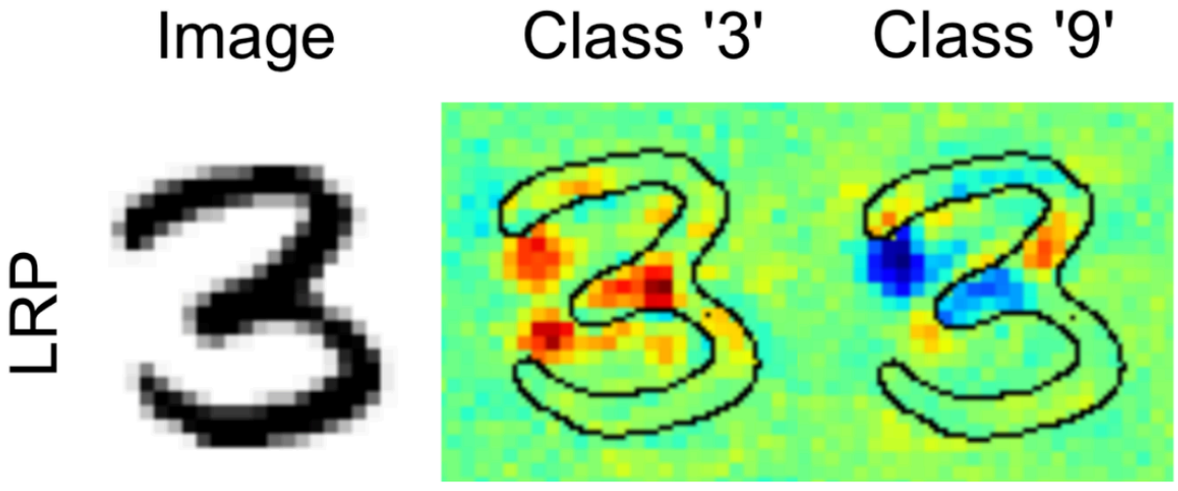
- In recent years industry has driven a big development in data analysis software
- In particular very powerful and flexible tools for data visualisation and machine learning based around python
- Machine learning:
  - Scikit learn
  - TMVA - a bit behind but catching up
- Neural networks specifically:
  - Tensorflow
    - Keras is a very convenient wrapper for this
  - Pytorch



Attempts to look inside the black box

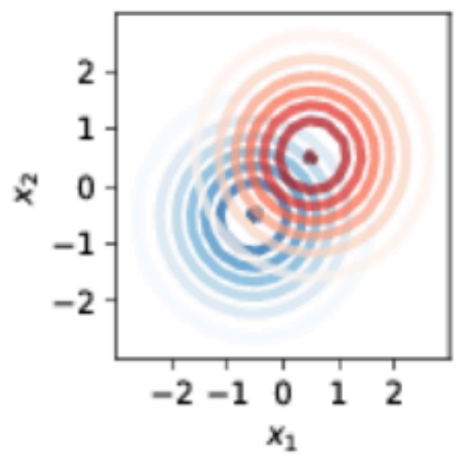


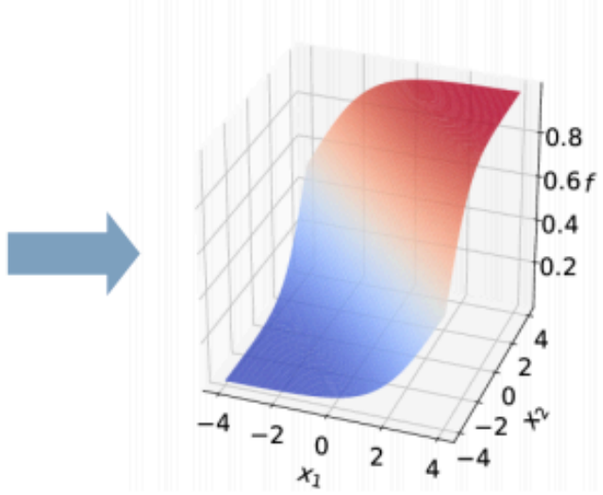


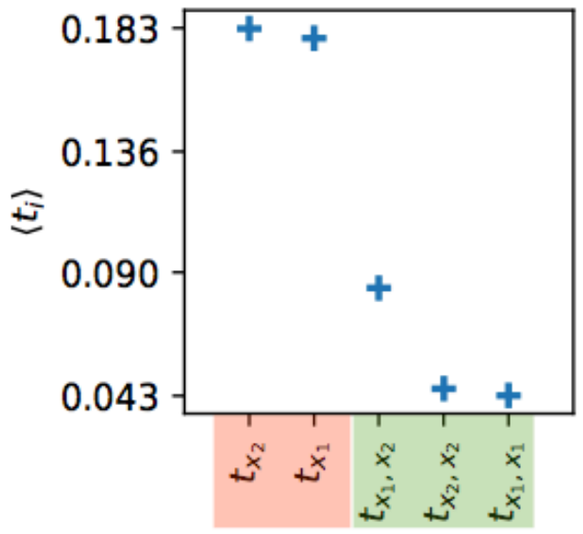


Layer-wise relevance conservation

$$\sum_i R_i = \ldots = \sum_i R_i^{(l)} = \sum_j R_j^{(l+1)} = \ldots = f(x)$$







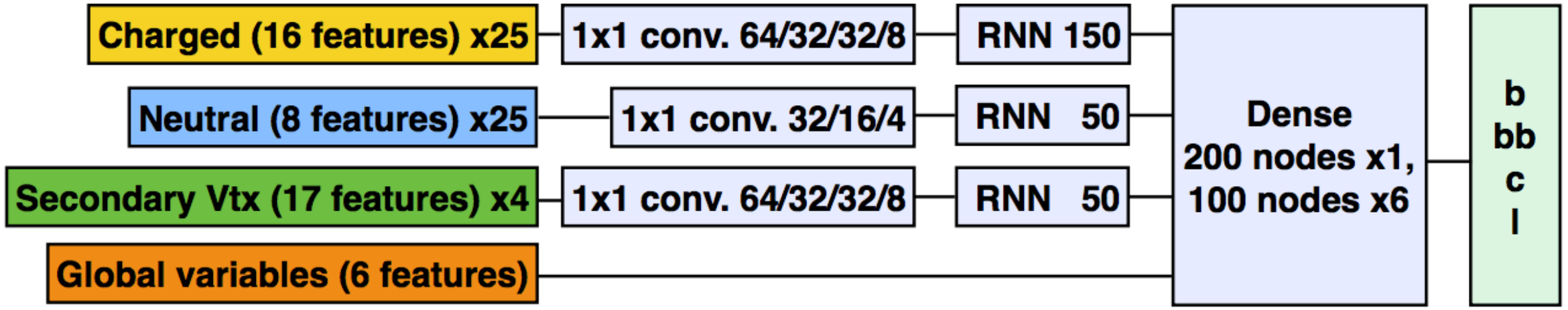
$$T(x,y) = f(a,b) + (x-a)f_x(a,b) + (y-b)f_y(a,b) + \frac{1}{2!}\Big((x-a)^2 f_{xx}(a,b) + 2(x-a)(y-b)f_{xy}(a,b) + (y-b)^2 f_{yy}(a,b)\Big) + \cdots$$

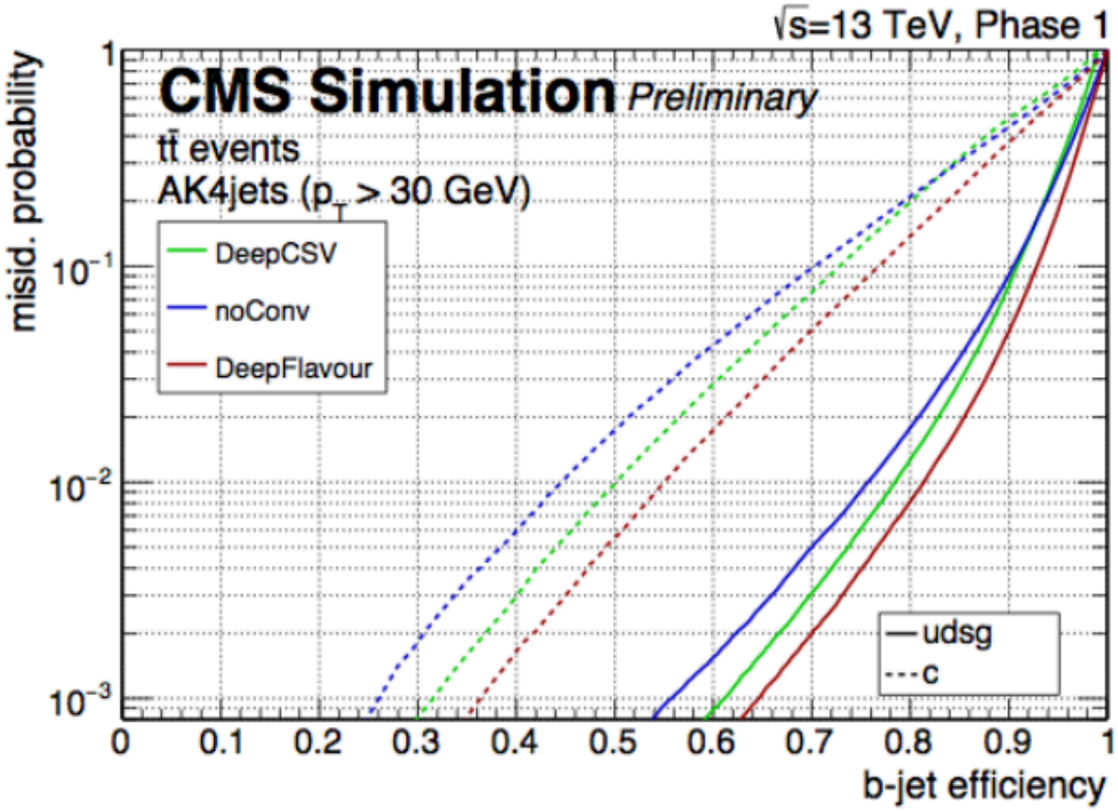
https://indico.cern.ch/event/668017/contributions/2947043/attachments/1630491/2598788/Talk_Samek.pdf https://indico.cern.ch/event/668017/contributions/2947008/attachments/1629022/2595649/slides.pdf https://github.com/marcotcr/lime

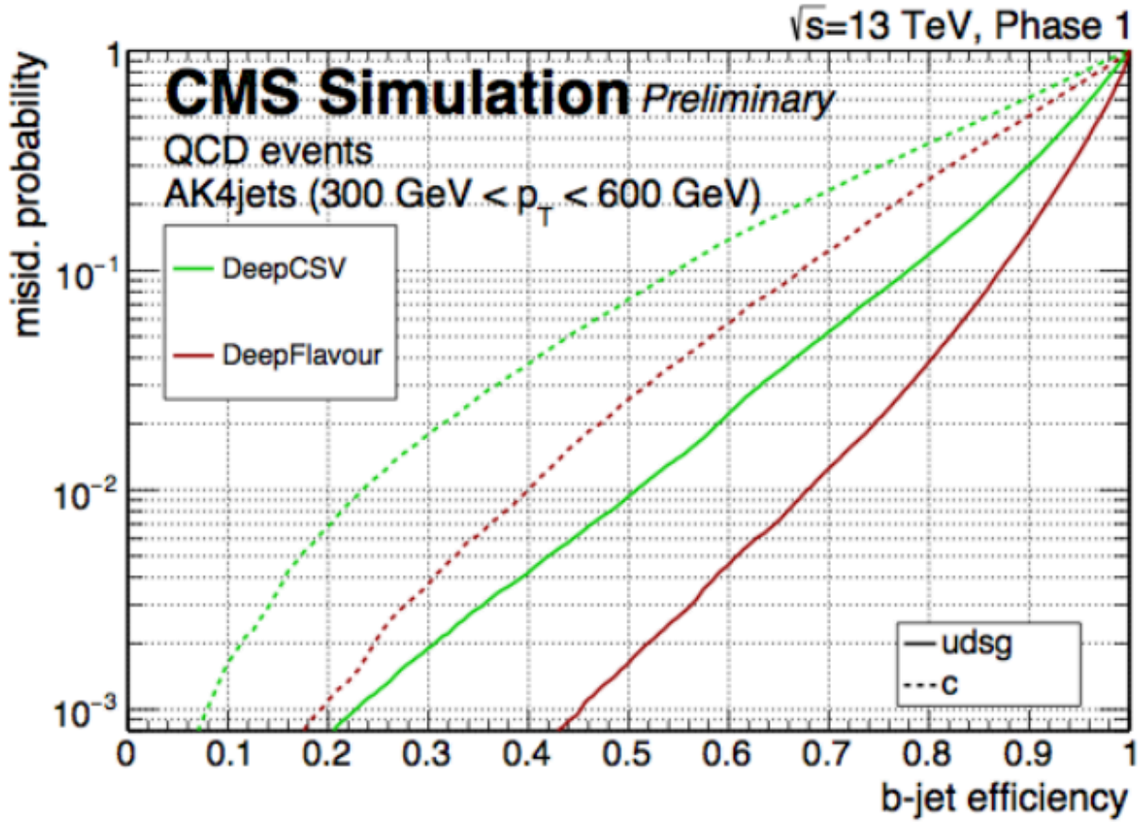
Application of advanced ML techniques in HEP

- Object reconstruction
  - Obvious place to use it, lots of low level correlated variables
  - Significant success in using DNNs for jet tagging (DeepJet)
- Generating simulation quickly
  - GANs are promising although need work to accurately reproduce statistics
  - Can minimise full detector simulation and significantly speed up generation
- Physics analysis
  - Most obvious use is in classification Signal vs Background
  - DNNs are well suited to non-binary classification and accounting for complex correlations
  - Can classify events based on their particular process

DeepJet





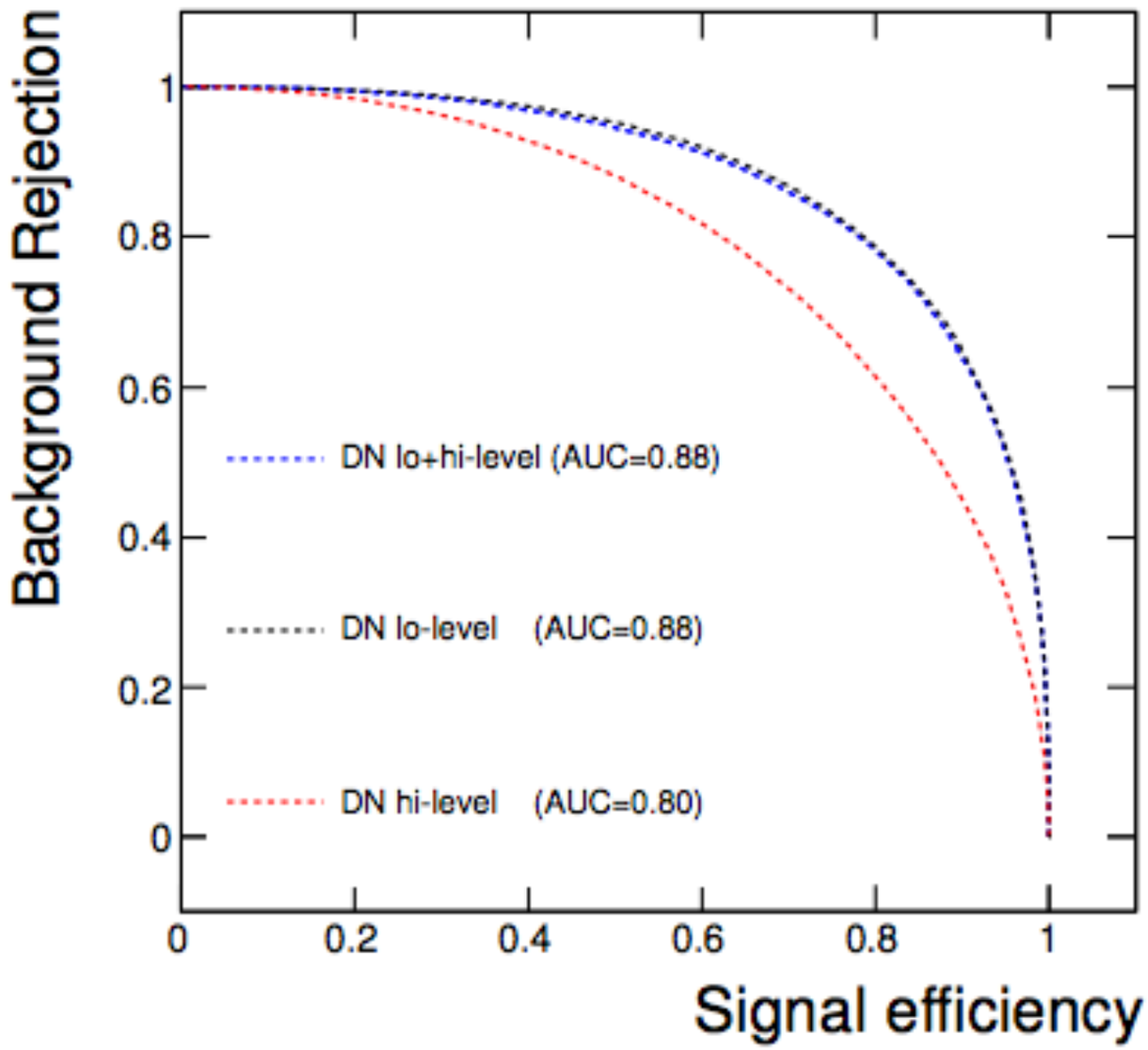


CMS-DP-2017-013

M. Verzetti (CERN and FWO)

Use in searches

- Can gain beyond high level variables with DNNs:

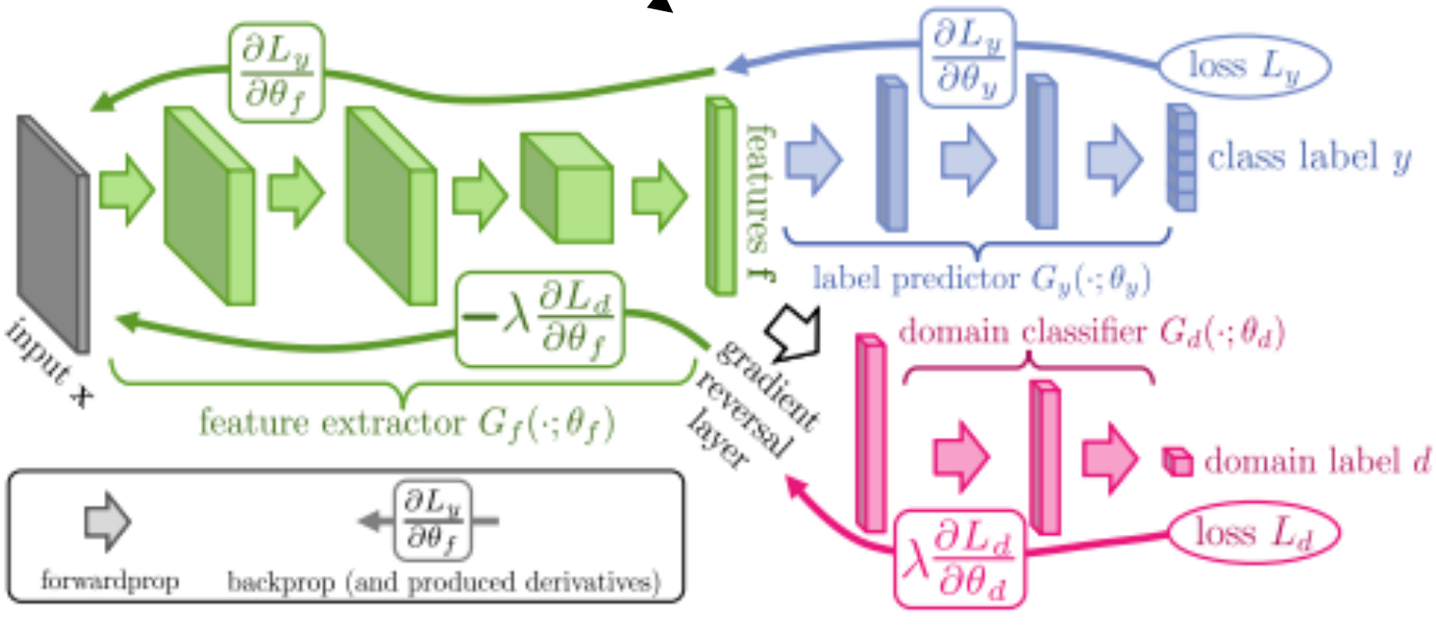


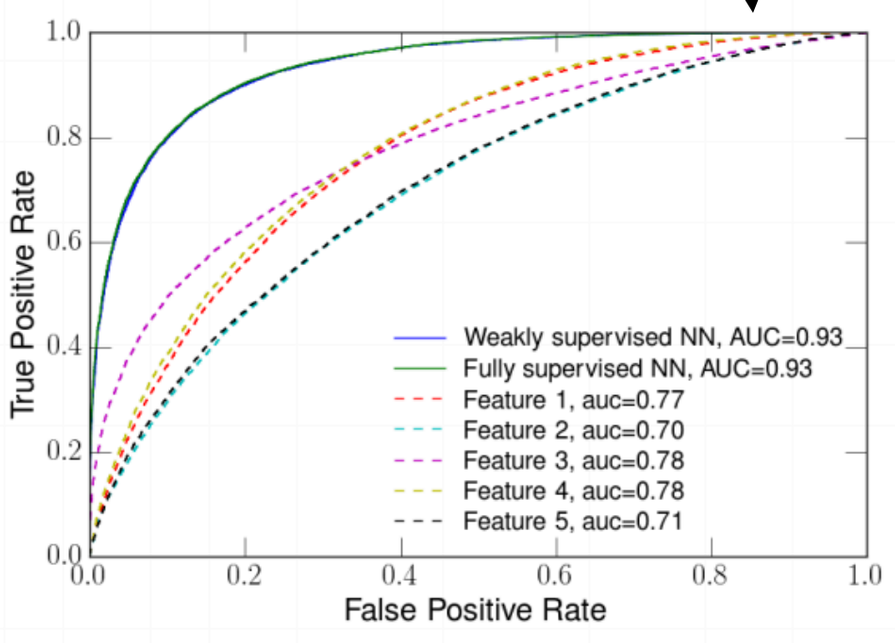
- Latest CMS ttH result has made good use of DNNs

https://arxiv.org/pdf/1402.4735.pdf

Too much reliance on simulation?

- Ideas to remove systematics introduced by training on simulation:
  - Only use input variables with good MC/Data agreement * (but take care of correlations)?
  - Train purely on data, one sample signal enriched and one background enriched? **
  - Train an adversarial network that penalises differentiation between data and MC? ***





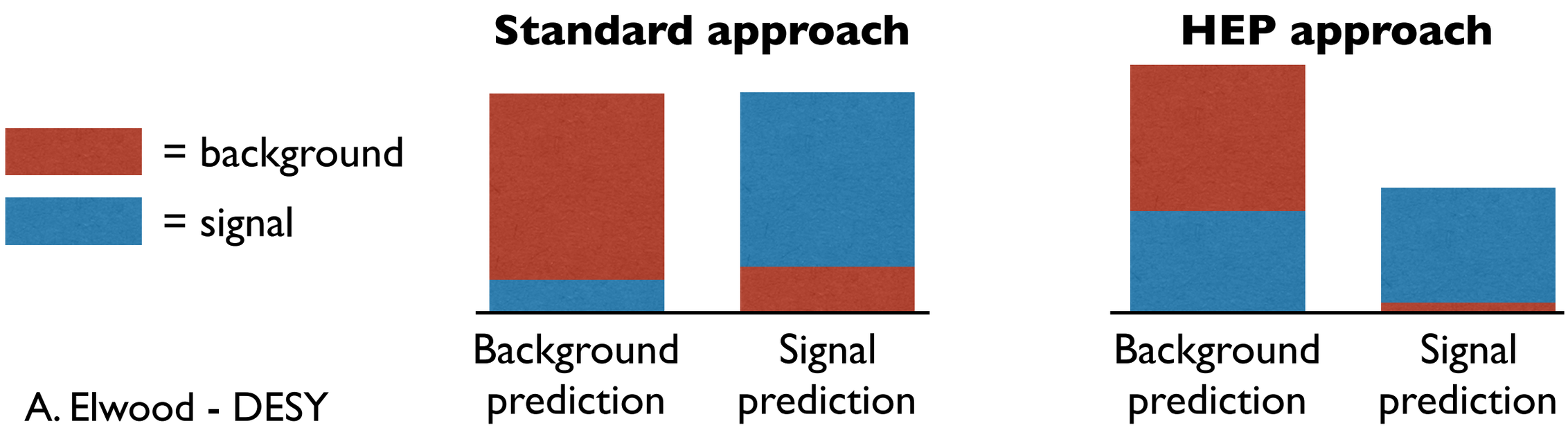
* https://indico.cern.ch/event/646801/contributions/2651014/attachments/1487701/2311252/6Jul2017_CMSML_systematics.pdf
** http://arxiv.org/abs/1702.00414 *** http://jmlr.org/papers/volume17/15-239/15-239.pdf , https://arxiv.org/abs/1702.05464

Directly optimising the discovery significance

Based on work by Adam Elwood and Dirk Krücker
Paper coming soon

A HEP search approach

- Typical machine learning approach is to optimise classification accuracy or area under ROC curve
  - All classes are effectively given the same weight
- When searching for new physics actually care about significance of signal counts over background counts
  - Don't care about the purity of the background classification
  - Can we design a new loss function to directly optimise this?



New optimisation scheme for neural networks

- We can design a loss function based around the direct optimisation of the significance
  - For a batch of training events:
    - s=(N correctly classified signal events)
    - b=(N incorrectly classified background events)
  - Maximise standard estimate of exclusion significance based on gaussian statistical uncertainties:

$$s/\sqrt{s+b}$$

  - Maximise Asimov estimate of discovery significance (including systematic):

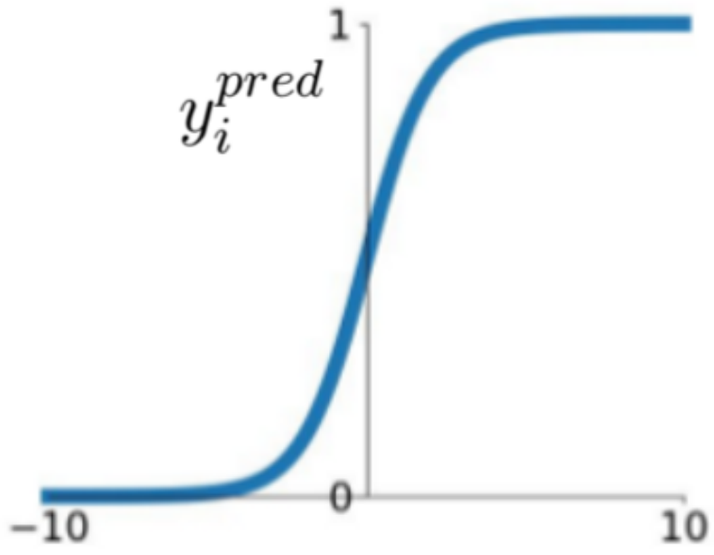
$$Z_A = \left[2\left((s+b)\ln\left[\frac{(s+b)(b+\sigma_b^2)}{b^2+(s+b)\sigma_b^2}\right] - \frac{b^2}{\sigma_b^2}\ln\left[1+\frac{\sigma_b^2 s}{b(b+\sigma_b^2)}\right]\right)\right]^{1/2}. \ *$$

* arXiv:1007.1727v3

Differentiability of loss functions

- To be able to carry out a gradient descent with backpropagation the loss function must be differentiable
- With the naive definition of s and b a reclassification of an event produces a discrete jump in the significance, it isn't differentiable!
- Must define a continuous transition from b to s
- For truth level, y_i^{true}, define as 0 for background event and 1 for signal event

Output neuron activation function



$$s = W_s \sum_i^{N_{batch}} y_i^{pred} \times y_i^{true},$$

$$b = W_b \sum_i^{N_{batch}} y_i^{pred} \times (1 - y_i^{true}),$$

$$W_s = L\sigma_{signal}\epsilon/N_{signal}^{true}, \qquad W_b = L\sigma_{bkgd}\epsilon/N_{bkgd}^{true},$$

Definition of loss functions

- Standard loss function to maximise accuracy or ROC score
  - Binary cross entropy:

$$C = -\frac{1}{n} \sum_x [y \ln a + (1-y) \ln(1-a)] ,$$

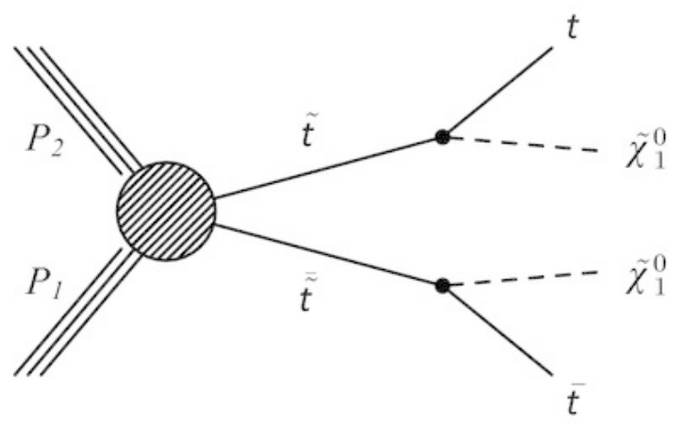
- New loss functions with definition of s and b from above:
  - They must be inverted to frame as a minimisation problem
  - Neglecting the square root to reduce expensive computations

$$\ell_{s/\sqrt{s+b}} = (s+b)/s^2$$

$$\ell_{Asimov} = 1/Z_A$$

Testing with a toy SUSY scenario

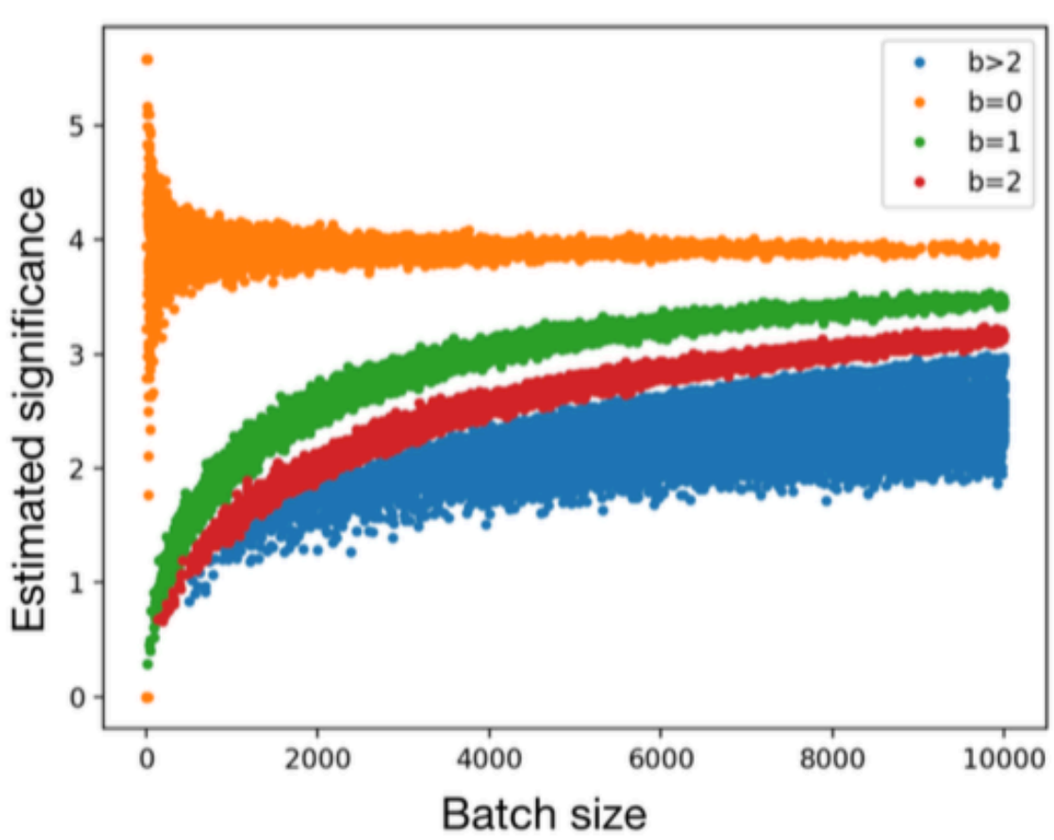
- To test this approach look at two mass points for the T2tt SUSY model that are on the edge of exclusion at 30/fb of 13 TeV LHC data
  - Uncompressed: m_{stop}= 900 GeV, m_{LSP} = 100 GeV
  - Compressed: m_{stop}= 600 GeV, m_{LSP} = 400 GeV

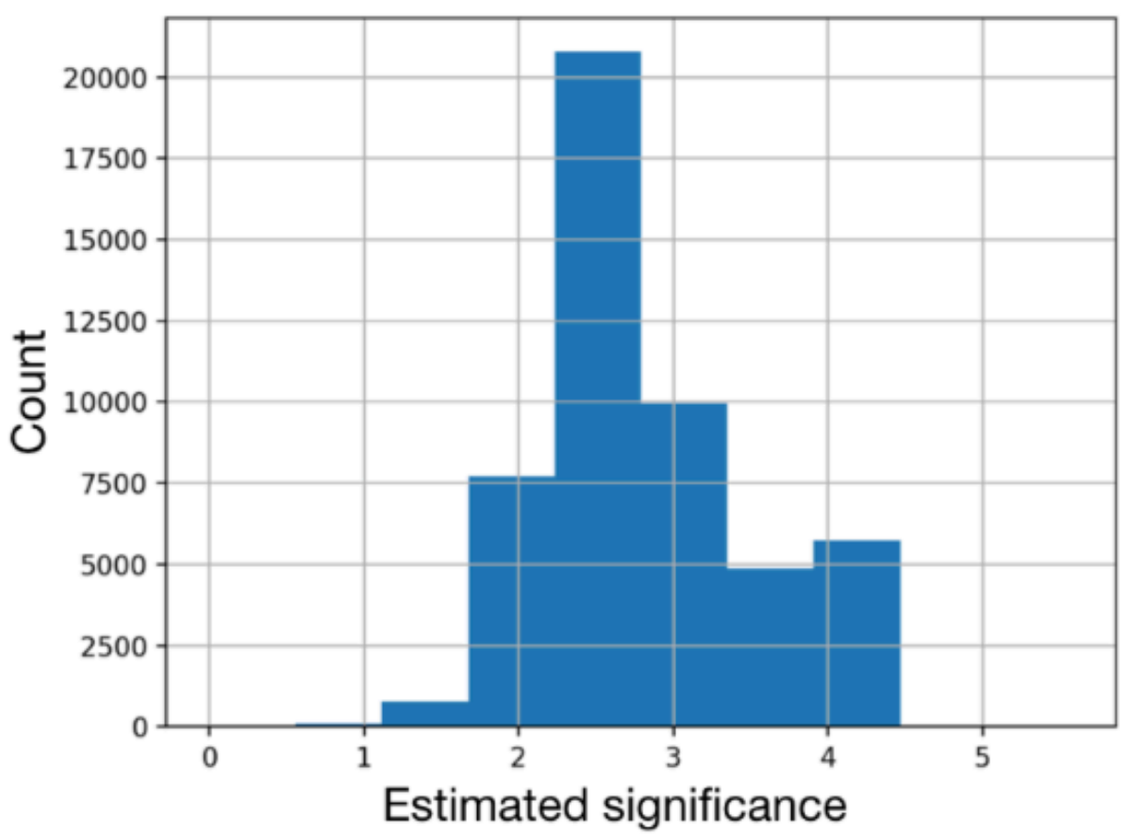


- Consider ttbar as the background
- Generate 1M events of signal and background with Pythia and Delphes after passing a basic selection criteria
  - 1 lepton pT>40 GeV, 4 jets pT>30 GeV, 1 b-tagged jet
- Consider the scenario with different systematics on the background from 0-50%

Batch size dependence

- With standard loss functions the loss is defined per event
  - The loss for a batch is just an average of the losses for each individual event
  - For cross entropy find a batch of 128 works well
- For the new loss functions the loss is defined per batch
  - When b is low, statistical fluctuations can effect the accuracy of the significance estimation
  - Find a batch of 4096 reaches compromise

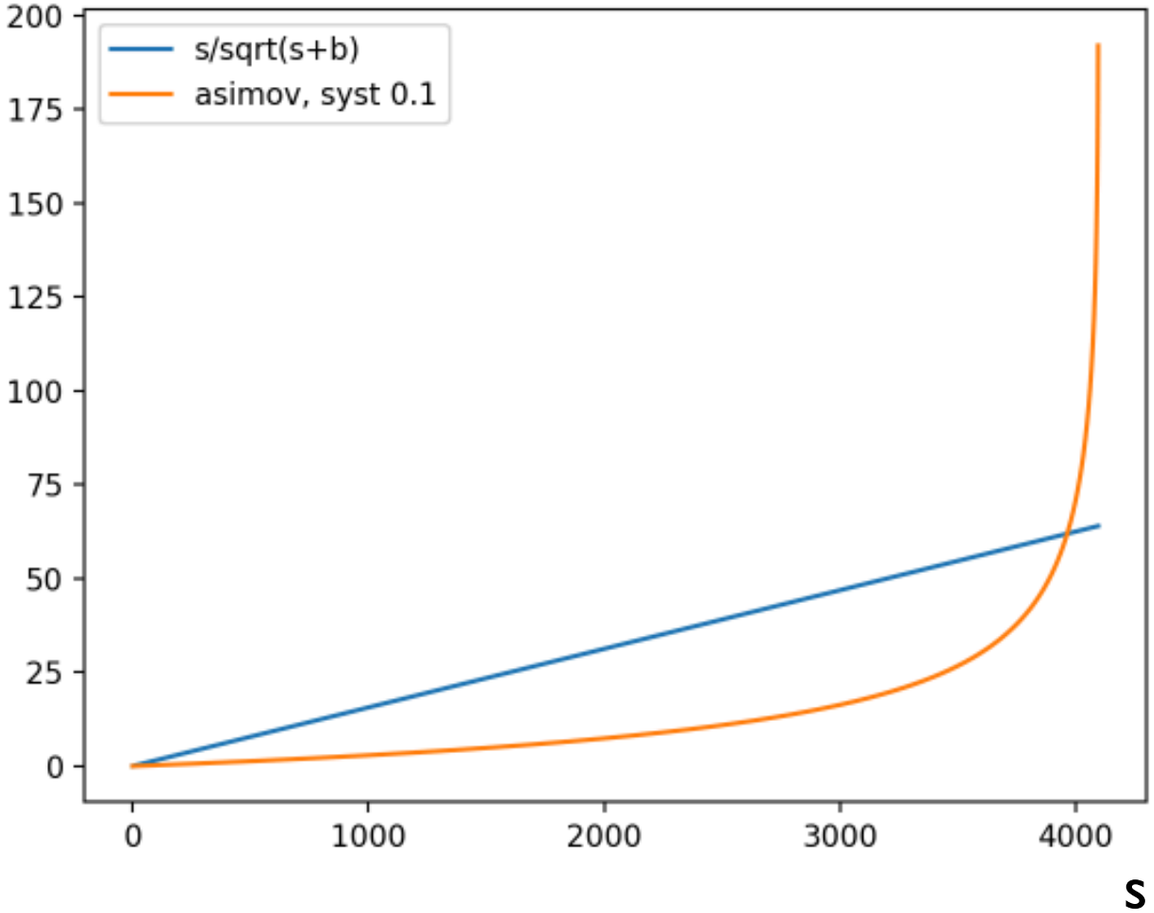




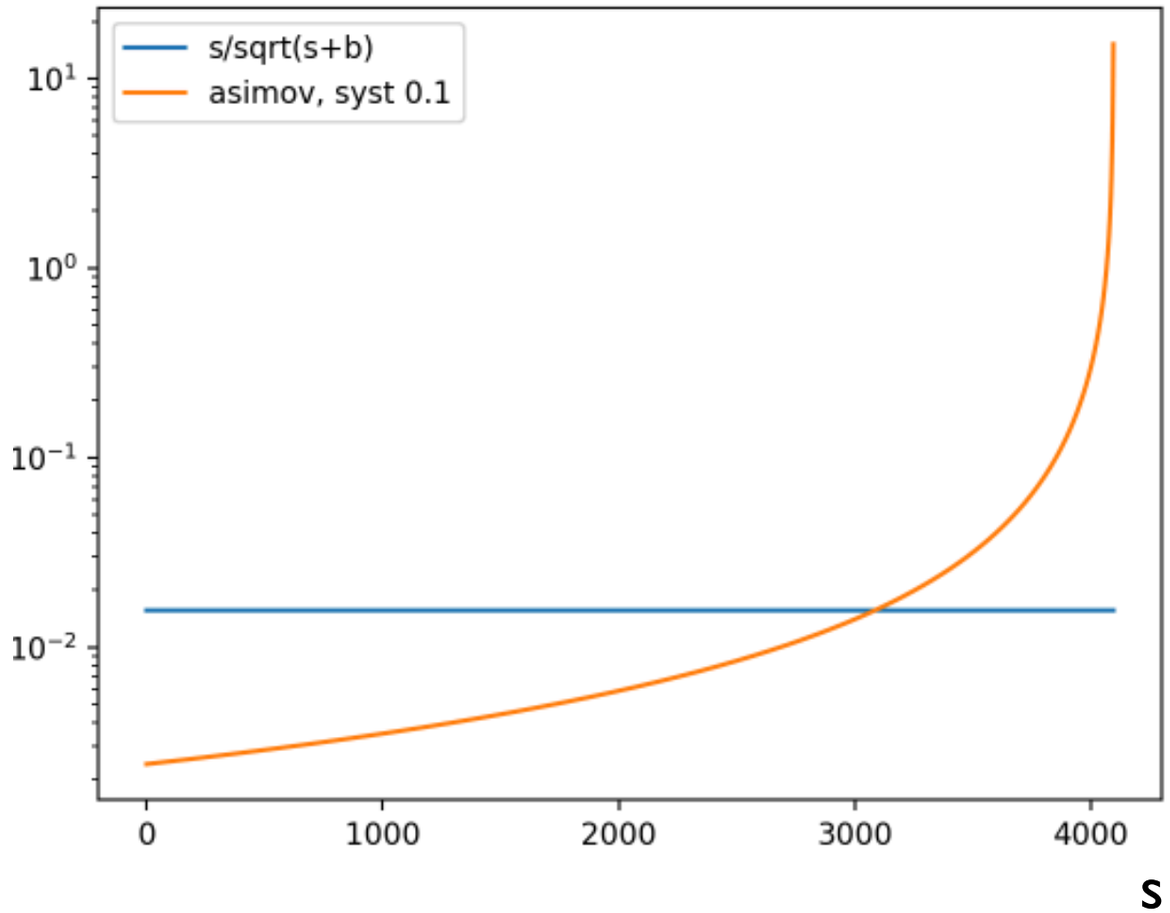
Issues with small gradients

- When s is small relative to the systematic uncertainty on b the gradient of the Asimov significance with respect to s is small
- Gradient descent fails when initialising the network and training with the Asimov loss
- Pretraining with $\ell_{s/\sqrt{s+b}}$ solves the problem, 5 epochs is typically enough

Significance



Gradient of significance

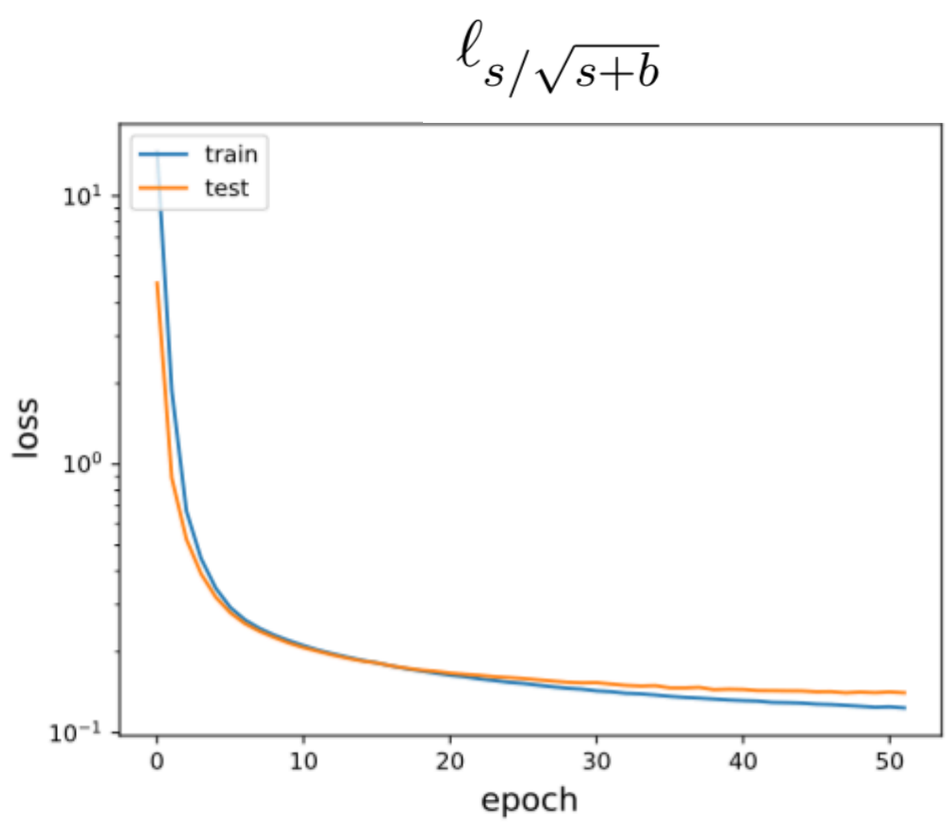


Training

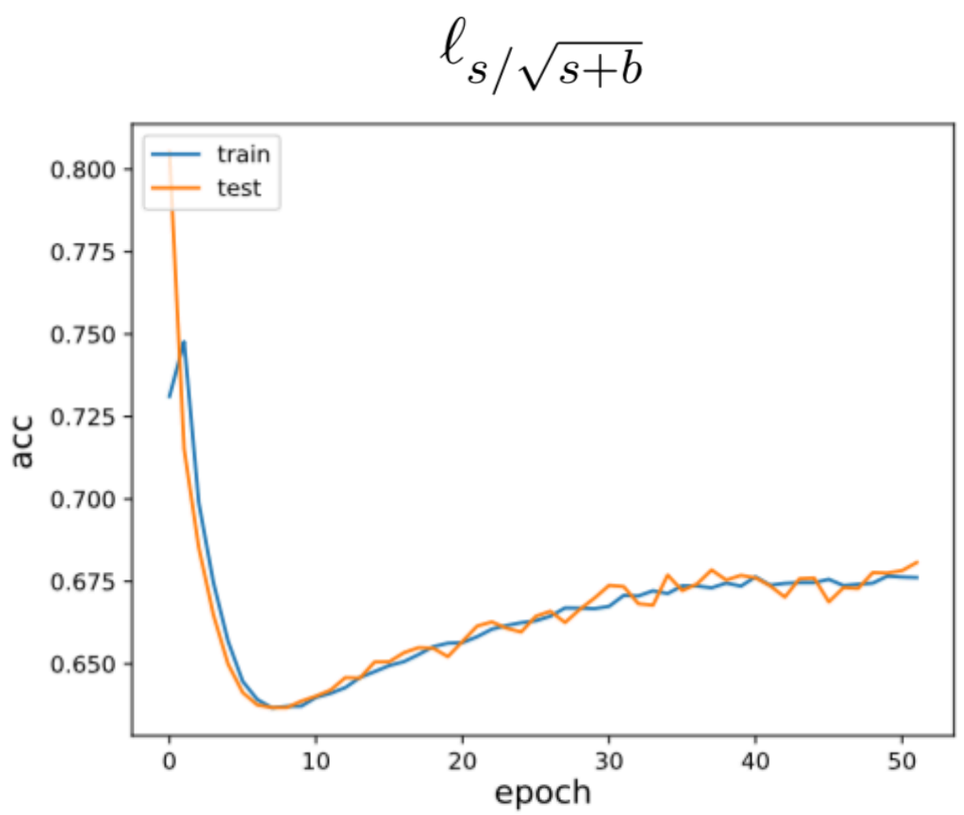
- Tested with various dense neural networks: 1, 3 and 5 hidden layers
- Results shown with a 1 layer, 23 neuron network
- 1.4 M training events, 0.6 M testing events (equally split between signal and background)
- Normalised to 30/fb of 14 TeV LHC data

Input features

low level	high level
$\vec{p}_l$	m_{T}
$\vec{p}_{jet(1,2,3)}$	m_{T2}^{W}
n_{jet}	$\not{E}_{\mathrm{T}}$
$n_{b\,jet}$	H_{T}



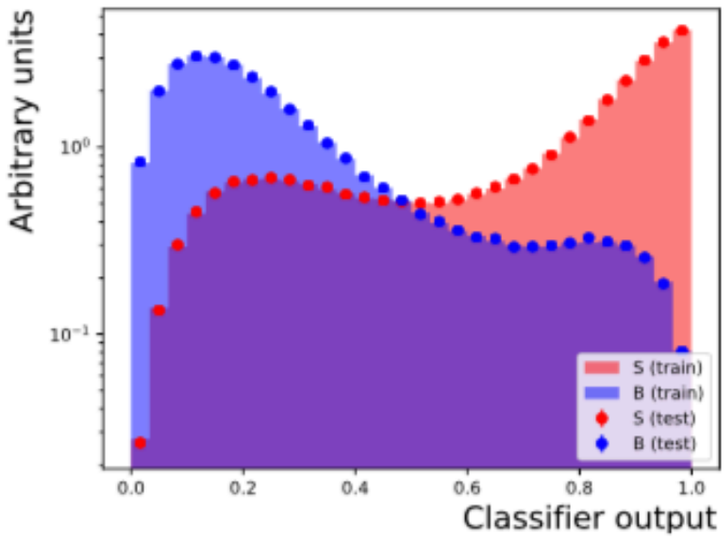
(a) Loss evolution



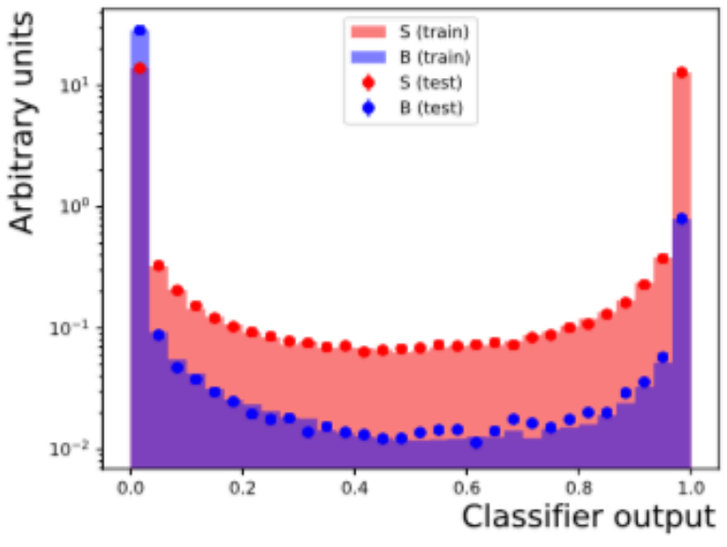
(b) Accuracy evolution

Results: classifier outputs

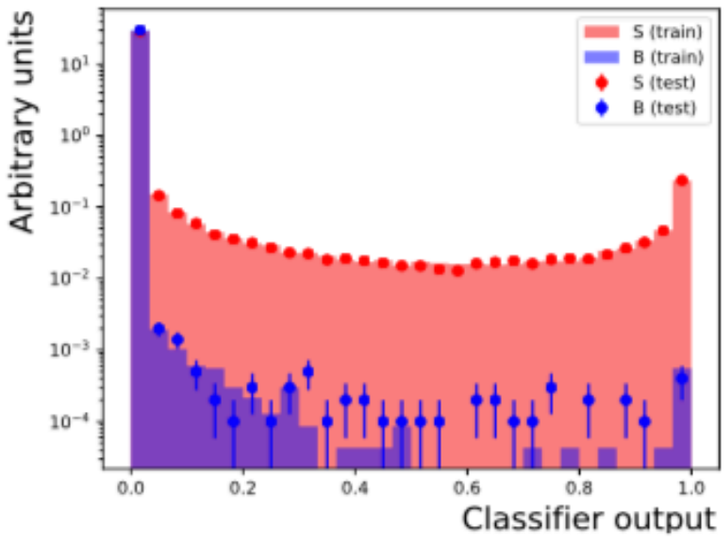
- Carry out training on the compressed model with a 50% background systematic for the Asimov training
- Plot the value of the output neuron
- Can see clear difference between the different approaches
  - New loss functions throw away signal events
  - Asimov loss retains very high purity due to the high systematic



(a) Binary cross entropy



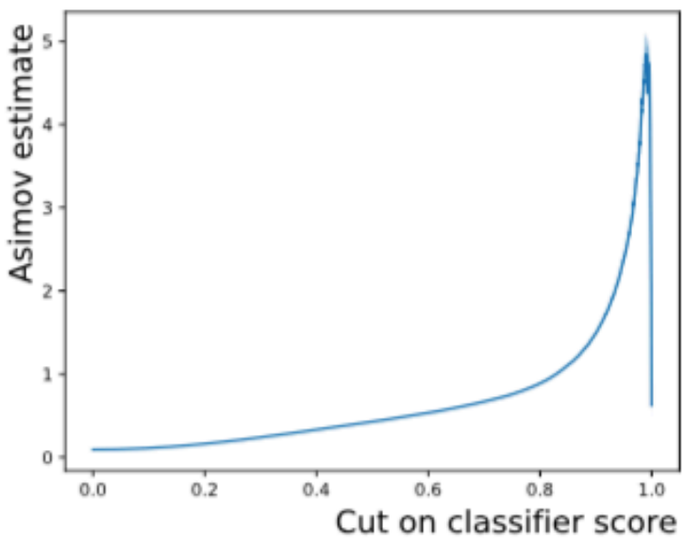
(b) $\ell_{s/\sqrt{s+b}}$



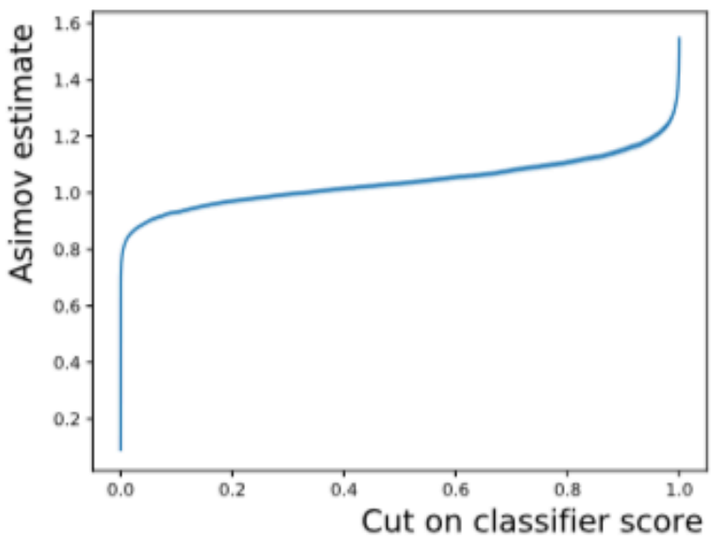
(c) ℓ_{Asimov}

Results: predicted significances

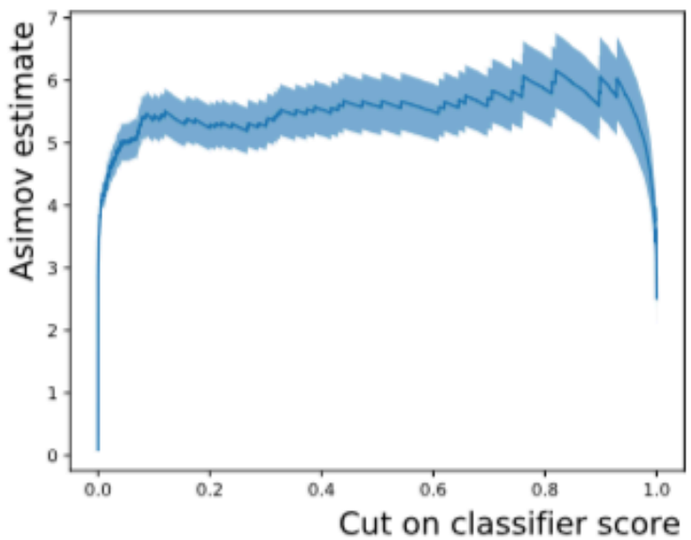
- To have a fair comparison of performance find the optimal cut on the classifier score to obtain the highest significance
  - Again compressed model with a 50% uncertainty for the Asimov estimate
- Going deep into the tail of the BCE does surprisingly well
  - Interpretable as a probability, so can go to very high probability of a signal event
- The awareness of the systematic in the Asimov training allows it to perform best



(a) Binary cross entropy



(b) $\ell_{s/\sqrt{s+b}}$



(c) ℓ_{Asimov}

Statistically dominated case

- The above procedure is carried out for a series of systematic uncertainties for all three loss functions
- The uncompressed model is in a regime dominated by statistical uncertainties
  - Low signal and background event counts
  - All loss functions perform similarly as there is no gain in knowing about systematic uncertainty

Loss	$Z_A(\sigma_b/b = 0.1)$			$Z_A(\sigma_b/b = 0.3)$			$Z_A(\sigma_b/b = 0.5)$		
	s	b	σ	s	b	σ	s	b	σ
Uncompressed model, $m_{stop} = 900\,\mathrm{GeV}$, $m_{LSP} = 100\,\mathrm{GeV}$									
Cross entropy	8.7	1.7	4.5 ± 0.3	7.7	1.2	4.0 ± 0.3	7.7	1.2	3.5 ± 0.3
$\ell_{s/\sqrt{s+b}}$	7.7	1.3	4.3 ± 0.3	7.7	1.3	3.9 ± 0.3	7.7	1.3	3.4 ± 0.3
ℓ_{Asimov}	6.6	1.6	3.6 ± 0.2	3.5	0.1	4.0 ± 0.5	3.3	0.1	4.2 ± 0.7

Systematically dominated case

- The compressed model is in a regime dominated by systematic uncertainties
  - Low signal and background event counts
- The training with $\ell_{s/\sqrt{s+b}}$ performs poorly as there is no flexibility to adapt based on the systematic
- The Asimov training starts to outperform the cross entropy at high systematics

Loss	$Z_A(\sigma_b/b = 0.1)$			$Z_A(\sigma_b/b = 0.3)$			$Z_A(\sigma_b/b = 0.5)$		
	s	b	σ	s	b	σ	s	b	σ
Compressed model, $m_{stop} = 600\,\mathrm{GeV}$, $m_{LSP} = 400\,\mathrm{GeV}$									
Cross entropy	74.4	18.2	10.7 ± 0.3	44.0	7.7	6.8 ± 0.3	40.5	6.8	4.8 ± 0.3
$\ell_{s/\sqrt{s+b}}$	323	326	7.0 ± 0.1	323	326	2.56 ± 0.03	324	327	1.55 ± 0.02
ℓ_{Asimov}	78.4	19.4	10.8 ± 0.3	25.9	3.2	6.8 ± 0.4	11.9	0.5	6.2 ± 0.6

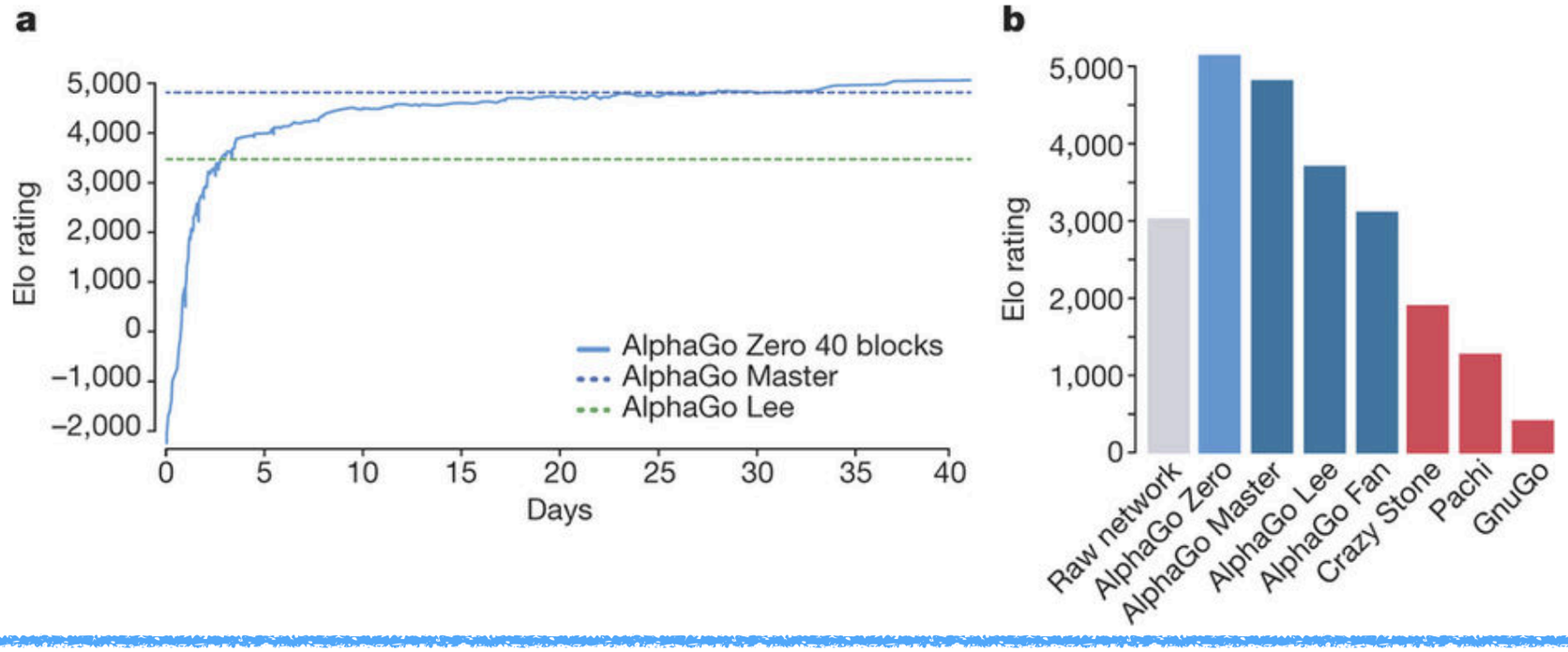
Conclusions

- Two new loss functions that aim to directly maximise the discovery significance instead of the correct classification of signal and background are presented
- They are tested in the context of a search for SUSY
- In the case that there are large systematic uncertainties on the background ℓ_{Asimov} can outperform the binary cross entropy
- Paper detailing the results to be put on the arXiv soon
- Code for the work available on GitHub
  - github.com/aelwood/hepML
  - Includes implementation of networks and loss functions with Keras

Summary

- With more and more development of sophisticated machine learning algorithms from computer science and industry HEP can gain a lot
- Neural networks are very promising
  - Lots of areas being investigated for application to HEP
    - Although don't neglect the BDT… most Kaggle competitions are won with XGBoost somewhere in the mix
- Still a very open area of research
- New optimisation techniques could help us get on our way to maybe finally discovering SUSY…

Potential and dangers…

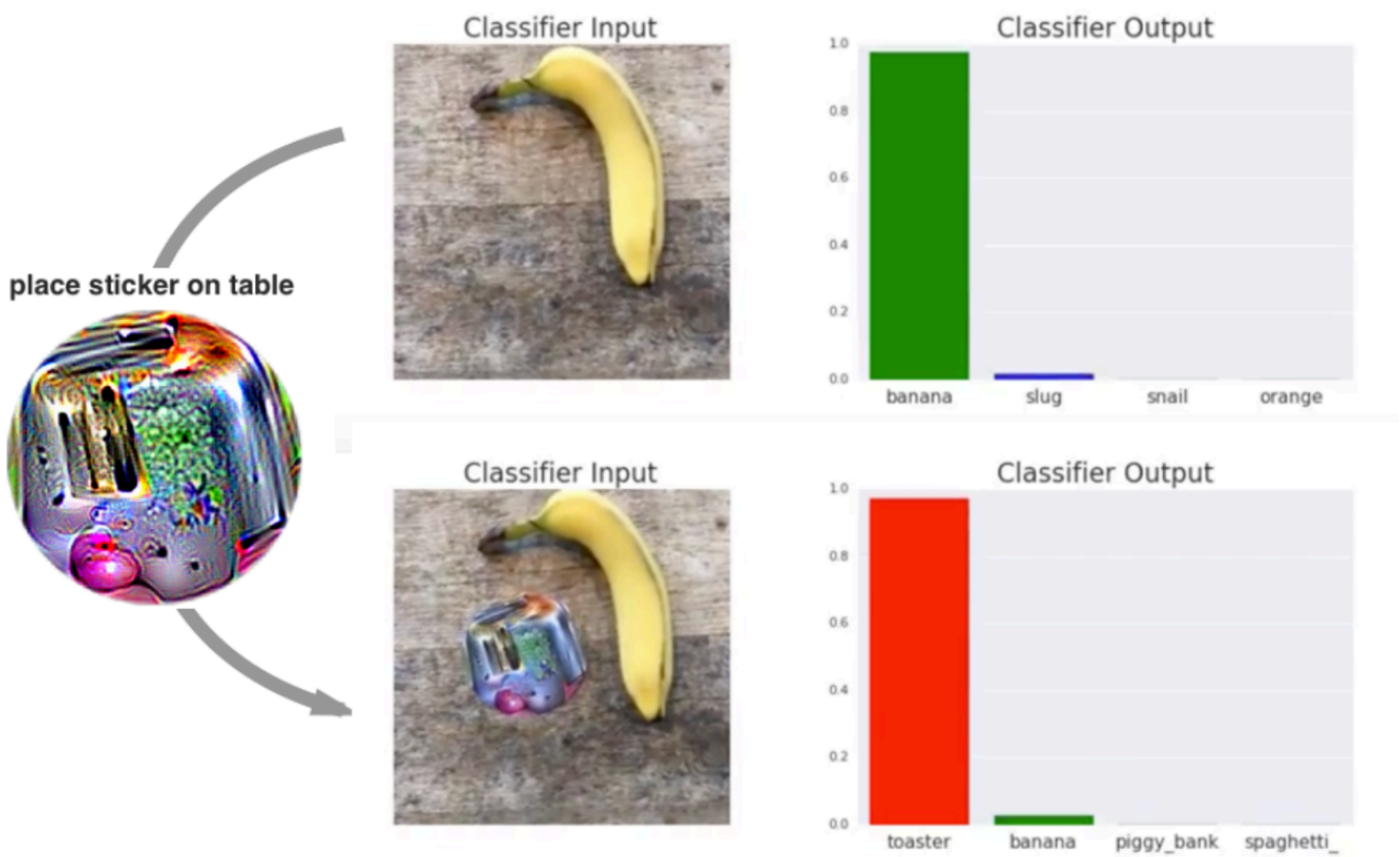


Deep Neural Networks with the right architecture can do this…

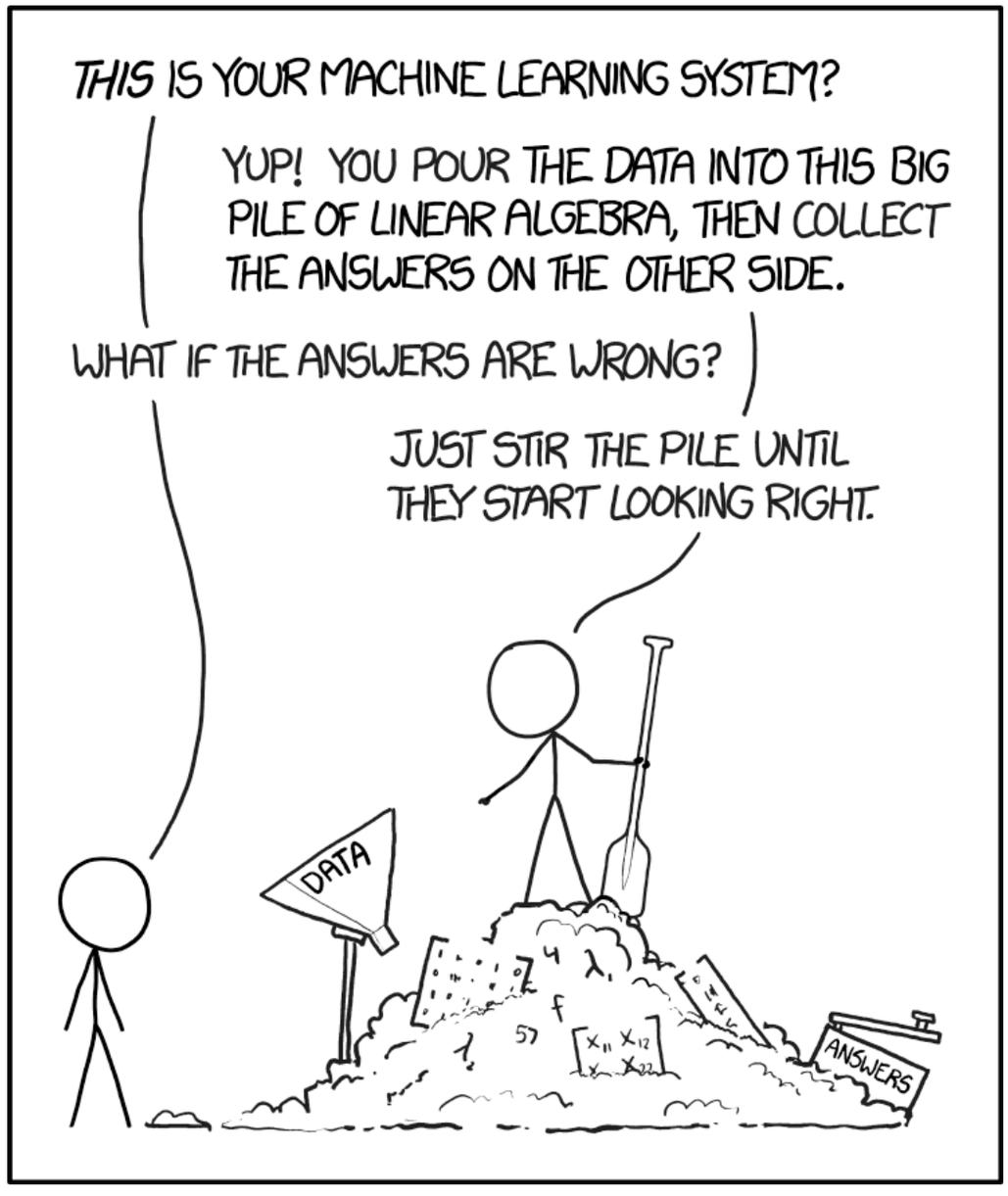
https://www.nature.com/articles/nature24270

But we need to make sure they aren't doing things like this…

https://arxiv.org/pdf/1712.09665.pdf



Backup

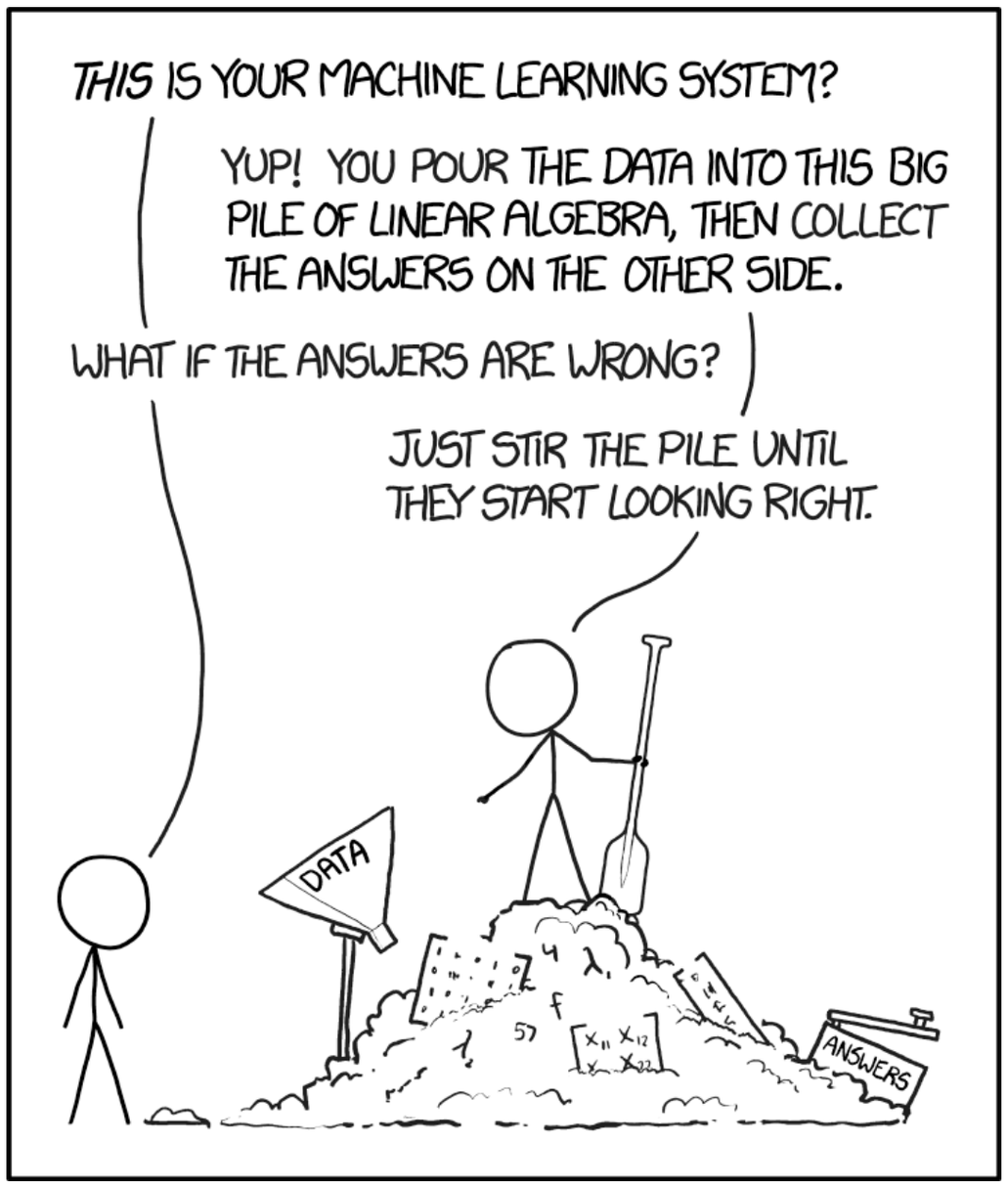

THIS IS YOUR MACHINE LEARNING SYSTEM?
YUP! YOU POUR THE DATA INTO THIS BIG PILE OF LINEAR ALGEBRA, THEN COLLECT THE ANSWERS ON THE OTHER SIDE.
WHAT IF THE ANSWERS ARE WRONG?
JUST STIR THE PILE UNTIL THEY START LOOKING RIGHT.
DATA
ANSWERS

References and useful links

- https://github.com/iml-wg/HEP-ML-Resources#people
- https://indico.cern.ch/event/619370/attachments/1449641/2235734/Kagan_Lecture1.pdf
- https://sebastianraschka.com/faq/docs/visual-backpropagation.html

A word on ML scepticism

- Think of it as a high level variable - not just a black box
- Something that takes account of correlations better than a hypercube in phase space *
- The ‘early data regime’ has ended, conservative cut based analyses aren’t as interesting anymore
- Lots of work is going into looking inside the ML black boxes and understanding why they work
- In the future these techniques will be used throughout HEP, now is the right time to start developing and understanding where they are useful

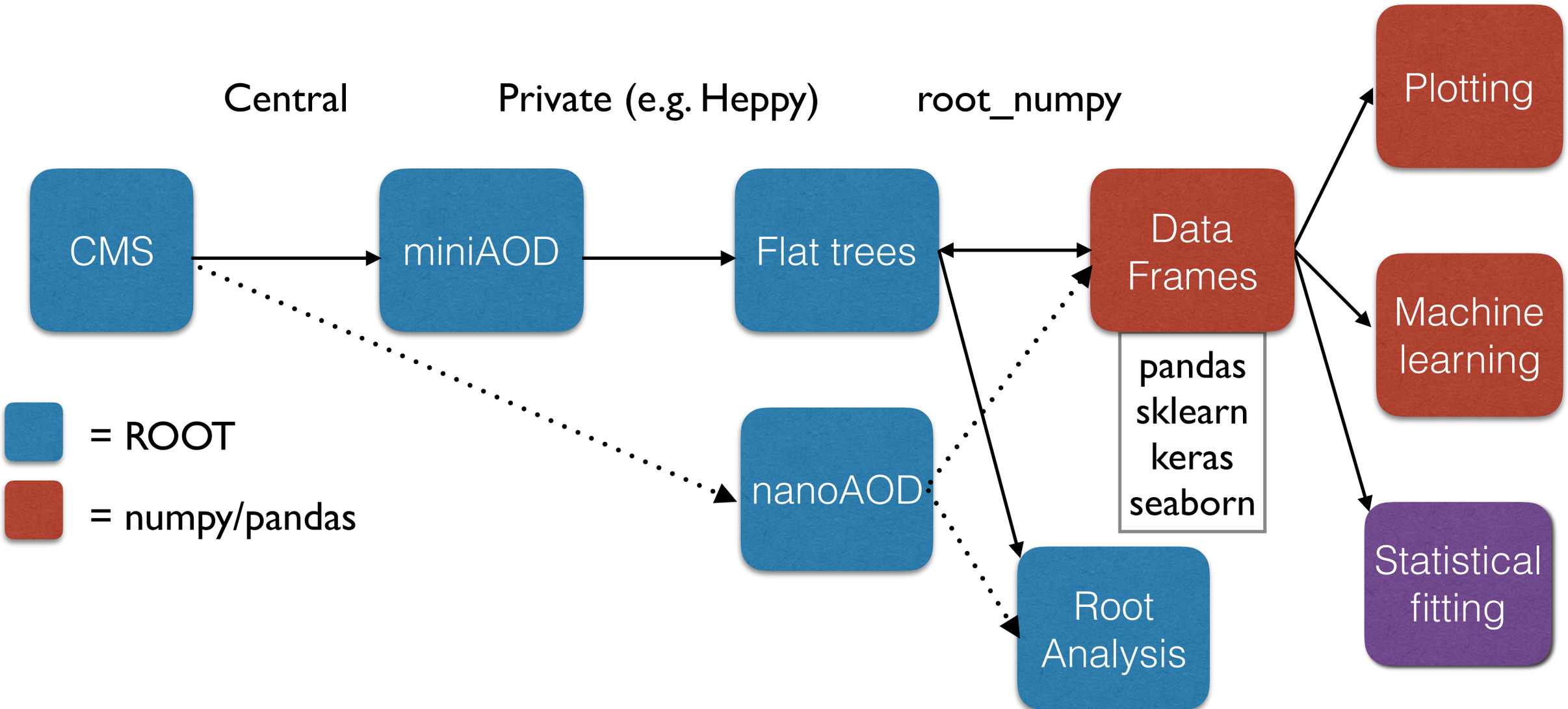


https://xkcd.com/1838/

* https://arxiv.org/pdf/1402.4735.pdf

How we can use the software in a generic analysis

- ROOT is still very good at some things
  - Complex event level calculations
  - Fast and sophisticated event loops
- Not always ideal for the final stage of the analysis



Software developed for ML with new tools

- Developing software exploiting the modern data analysis python libraries (based around numpy)
- Installation setup independent of CMSSW and working on NAF but portable to most computer systems
  - Easy recipe with anaconda available that includes root (standalone installation)
- Check out and contribute to from https://github.com/aelwood/hepML