An AIX Kitbag

DISCUSSION DOCUMENT

Draft Version 1, 21/12/2017

Minor revisions 12/5/2018

Alan Dix

Computational Foundry, Swansea University, Wales

http://alandix.com/academic/talks/ sufficient-reason-2018/

This document contains some techniques used by the author, either in academic work or consultancy, some being used elsewhere in some form, or highly likely to work, and some more speculative.

The techniques are roughly classed into white-box, black-box and grey-box techniques, but this is not a hard-and-fast distinction as most techniques have a hybrid nature.

Several are inspired by human expert knowledge elicitation/externalisation techniques, and what counts as an acceptable 'explanation' in human–human conversation. For this, we do not expect to be able to dump the neural assembly of our brains, but still manage to create what we regard as acceptable levels of model or reasoning.

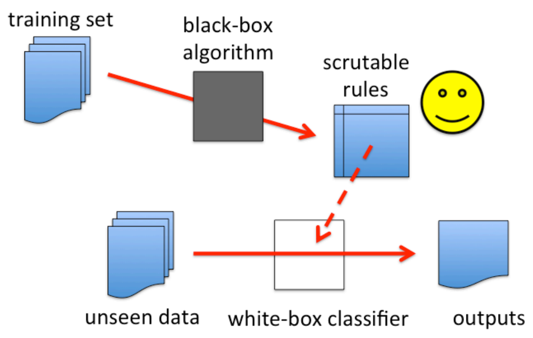
The focus here is on creating more humanly comprehensible ways to represent the behaviour of machine learning, but as these typically involve some level of data reduction or simplification, it is possible that they may also be useful in creating more robust machine representations.

White-Box Techniques

WB0. *KISS*
The simplest technique is to deliberately choose an algorithm that matches the domain and can be rendered in a comprehensible manner – for example the use of ID3 to generate SQL (via a decision tree) in QbB.

This is trivial, but has two slightly more interesting variants:

WB1. *Black-box generation of white-box solution*.
For example, one variant of QbB used genetic programming to generate decision trees, similarly genetic algorithms or similar techniques have been used to generate rule-based classifiers.

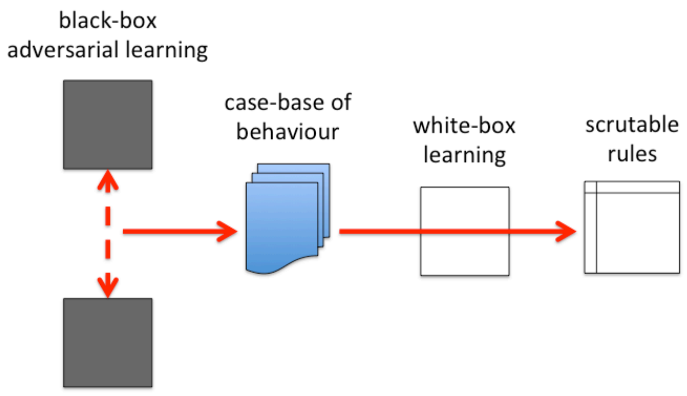


This is rather like the mathematician who can write the proof, but may struggle to explain how they came by it, or the programmer who creates the code, but may not be able to introspect their programming practice.

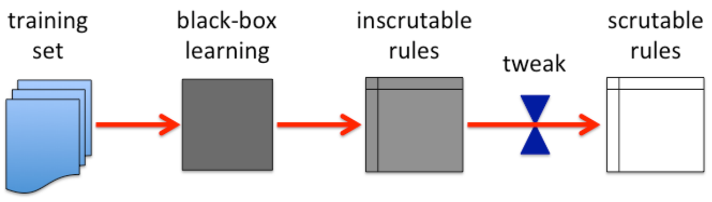
Note that it is may be easier to build in additional 'comprehensibility' goals into a fitness function for a black-box technique than it is to find a more procedural algorithm. For example, a variant of Quinlan's ID3 generated long thin trees as it is easier to understand a set of conjunctive rules than a more balanced decision tree. To achieve this he had to make subtle tweaks to the ID3 algorithm, but in, say GA generation of decision trees you simple add tree shape as well as accuracy as part of the overall fitness function.

WB2. *Adversarial examples for white-box learning*.
Adversarial reinforcement learning (as used in AlphaGo) generates large case-base of examples. One of the problems with many knowledge rich ML techniques (especially ones that have a stochastic/uncertainty elements) were hamstrung as they often needed to work on small training sets, risking over-learning from repeated exposure to the same

examples, and missing cases where there were none. Some classic techniques may well be able to be used more effectively on these large data sets, or variants created that were previously infeasible due to limited data. Hopefully a lot of the work for this is already in place because of big data; so in a way this is simply using the case base of behaviours generated from adversarial learning as a source of big data!



WB3. *Simplification of rule set.*
There was some early work on neural networks where the neurons were 'hardened' into binary networks after learning. The sigmoid activation function is necessary to 'soften' the network to allow back-propagation learning, but not needed for actual use (a standard heuristic of both human and computer learning is that it is often easier to learn continuous than discrete boundaries, so that you get an idea of 'warmer' rather than just right/wrong).

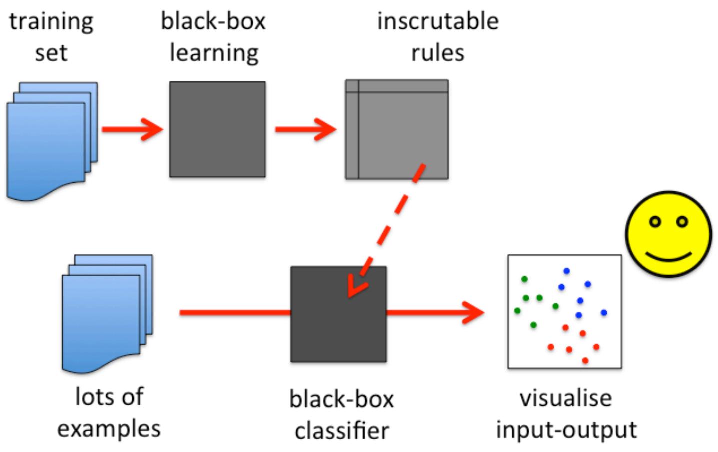


Note that sensitivity analysis, either using mathematical or algorithmic techniques can be used to help this process, for example, transforming sigmoids into either thresholds, linear scaling or more complex functions depending on ranges of (internal) inputs to the node on the training data. (See more later under grey-box techniques.)

Black-Box Techniques

Here we regard the core learning algorithm and its representations as entirely inscrutable. However, this inscrutable representation can be used with the black-box algorithm (e.g. classifier) to act as an oracle on unseen inputs. This enables various forms of exploratory or perturbation analysis that use manipulations of the inputs and explore their effects on the outputs.

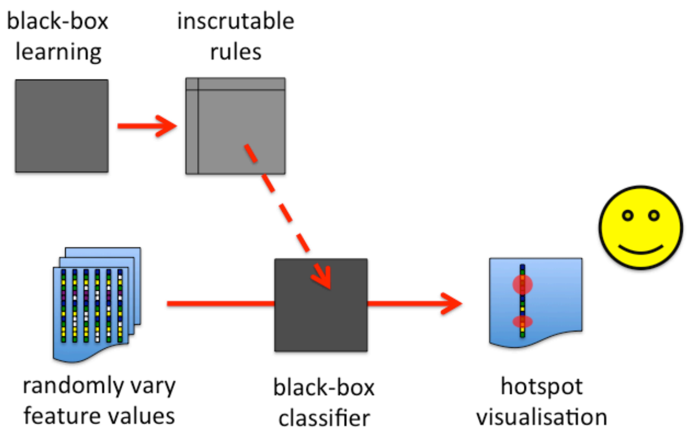
BB1. *Exploration analysis for human visualisation.*
Lots of random or systematically chosen inputs can be used to create input–output maps that can be visualised using standard scientific or information visualisation techniques.. The influence explorer is a good example of this albeit for simulation-based results rather than black-box ML.



BB2. *Perturbation/exploration analysis for key feature detection.*
This is especially useful where there is a large input feature vector, which is being used in an unknown or hard to comprehend way by an algorithm. Basically one either systematically or randomly changes individual input features or sets them as missing values (if the algorithm can deal with these) and see how this affects results. This can be used to establish which are the most important features for particular aspects of the automated decision-making.

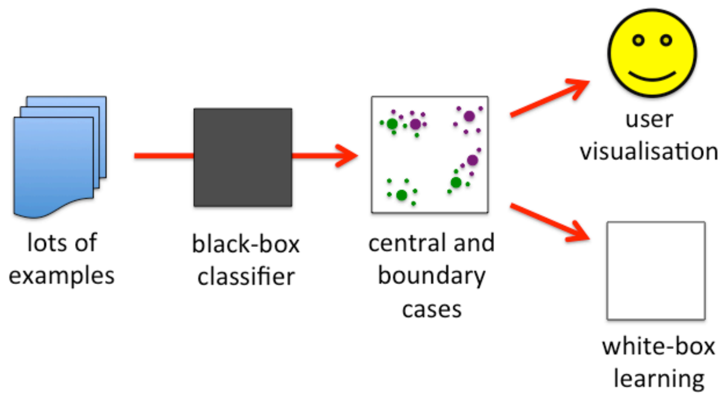
I have seen a recent example of this in vision research where patches of pixels are randomised and this is then used (I think automatically) to create hot-spot images of where the computer is focusing its 'attention' in making particular classifications. This can help determine, for example, whether the learnt rules are using sensible parts of the image or potentially accidental features (like the old example of a NN that appeared to be able differentiate US vs Russian tanks, but turned out effectively be a sunny weather recogniser!).

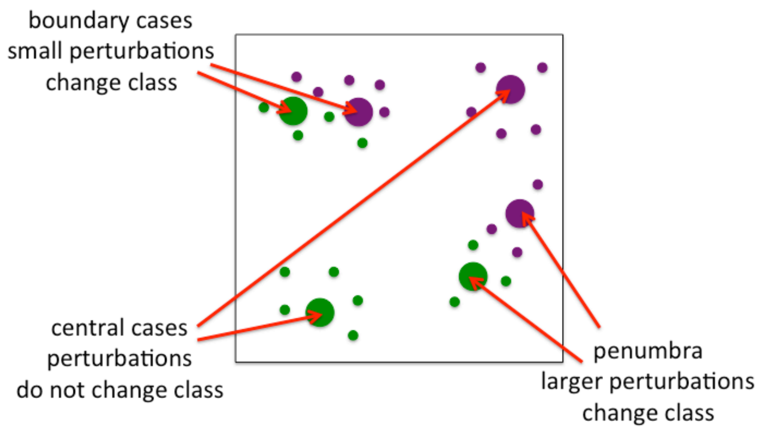
The outputs of this can be used for human comprehension directly, or can be used for feature detection for second round of learning … including white box techniques.



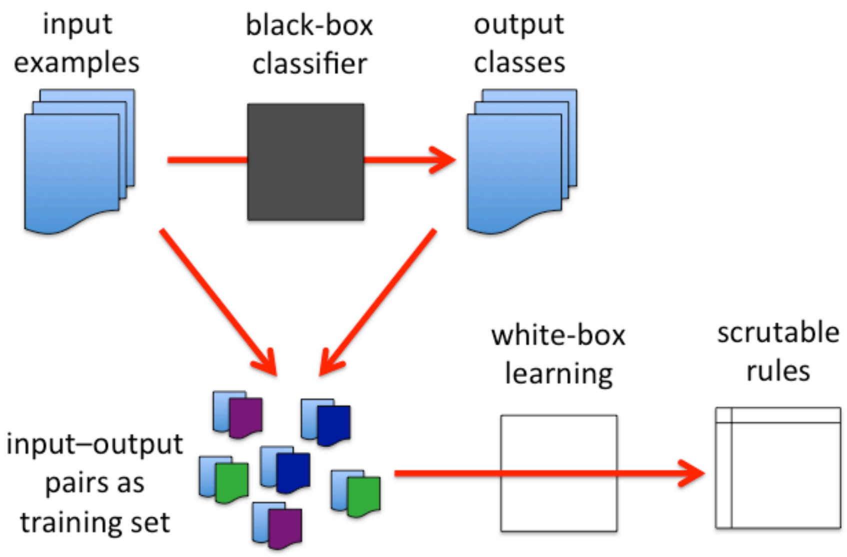
BB3. *Perturbation analysis for central and boundary cases.*
Use this for classification or other forms of discrete output algorithm. As with BB1, start off by generating lots of examples, but then perturb each. If an example's output remains constant (e.g. discrete classification) despite perturbation it is central. If small perturbations change the class it is a boundary example. Those examples where small perturbations do not change it, but larger ones do (large and small as measured by Hamming distance, or other suitable metric) are in the penumbra of the boundary – these may also be useful.

The boundary and central cases can be used to help a human understand the classes. The examples can also be used to train different kinds of classifiers, for example using boundary cases to boost learning for more symbolic ML (BB methods feeding into WB). The actual boundaries or penumbra cases can be chosen depending on whether the secondary ML can deal with uncertainty.





BB4. *Black-box as oracle for white-box learning.*
Rather like BB1, the black-box model can be used to classify large numbers of unseen or generated examples. These generated input–output pairs can then be used as training data for white-box algorithms, as we described doing for adversarial behaviours in WB3.



Grey-Box Techniques

These techniques rely on using the internal representation of the black-box algorithm as an opaque feature vector from which other forms of white and black-box algorithms can be applied in order to obtain humanly meaningful results or create other learned representations.

There are two forms of internal representation at play here:

(i) The parameters, such as NN weights that are learnt during training and are then usually static during use.

(ii) The internal activations of nodes or other forms of intermediate calculations that vary for each input.

The exact form of these will be different for different kinds of algorithm, but as a generic term we'll use the term *parameters* for the former and *activation* for the latter.

For temporal networks (ii) gets a little more complex as the same input might generate a different internal activation. However, the majority of such algorithms have some form of internal representation of state where the rest of the algorithm can be seen as calculating:

state', output = f (parameters, state, input)

So, if we regard the state–input pair as the input, we can effectively, without loss of generality, consider only simple non-temporal input–output algorithms.

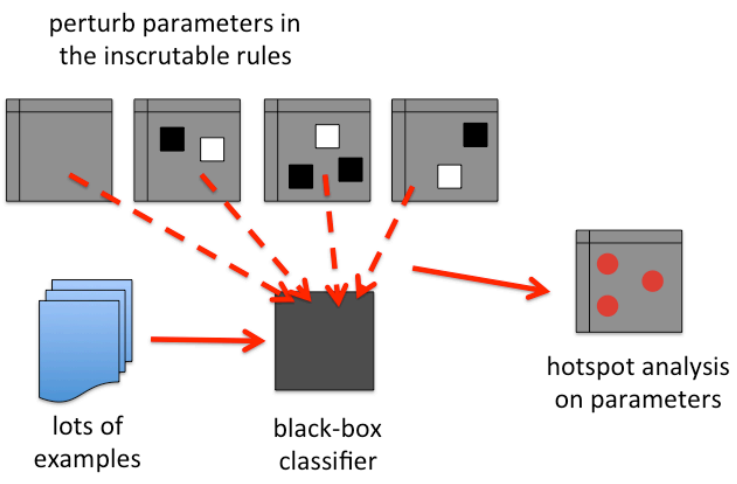
Again for the case of simplicity, we'll refer to the site of atomic calculations within an algorithm as *nodes*, thinking of NNs as central example, but this might also be, for example, branches in a decision tree.

Also the internal representation may often be layered (esp. DL), with lower layers closer to the input, one or more intermediate layers and a high layer that generates the output.

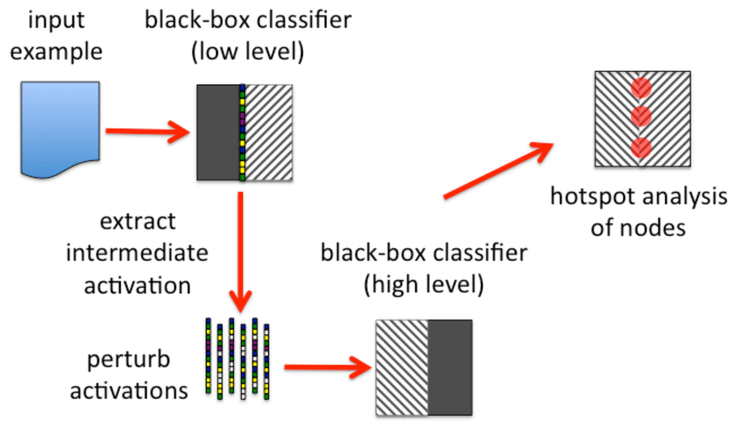
GB0. *Sensitivity analysis.*
Several techniques can benefit from some form of sensitivity analysis to allow pruning/simplification or add weights/uncertainty factors for other algorithms.

There are at least three ways this can be done.

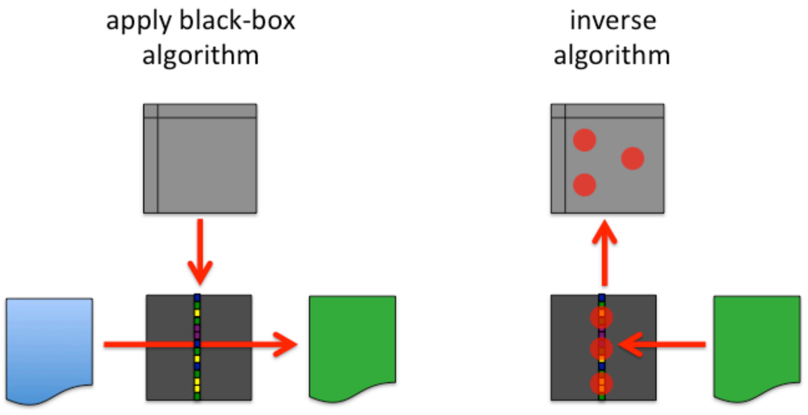
(a) Focus on the parameters (e.g. NN weights) – perturb these and see if this affects the overall goodness of fit of the representation.


perturb parameters in the inscrutable rules
lots of examples
black-box classifier
hotspot analysis on parameters

(b) Focus on the activation – perturb the outputs of individual nodes for specific inputs and see whether this affects the overall output.


input example
black-box classifier (low level)
hotspot analysis of nodes
extract intermediate activation
black-box classifier (high level)
perturb activations

(c) Algorithmically – use some formula to derive this, for example, back-propagation in a NN.



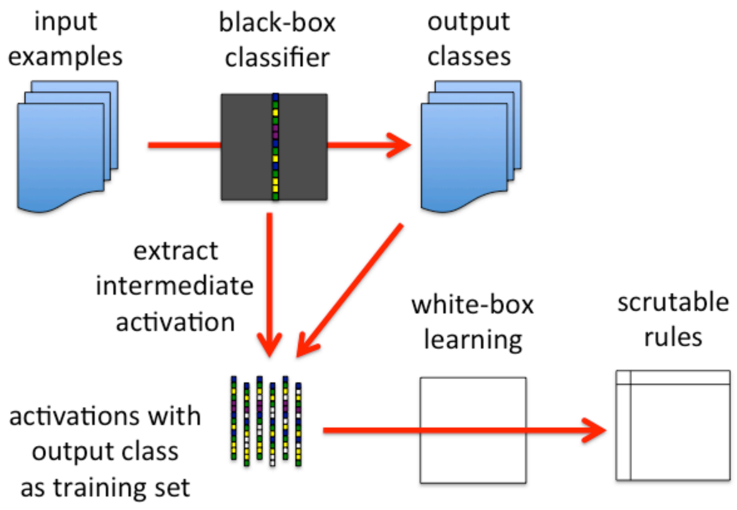
GB1. *High level model generation.*
Intermediate activation, possibly paired with the raw input, is used as the feature vector for some form of white-box learning algorithm that generates the output from these. That is one is seeking to replace the top level(s) of the representation with one that is easier to understand.

Effectively we are seeing the inscrutable algorithm as operating in two parts:

$$\begin{aligned} \text{activation}_{\text{int}} &= f_{\text{low}}\ (\text{ input }) \\ \text{output} &= f_{\text{high}}\ (\text{activation}_{\text{int}}\ ) \end{aligned}$$

We then are seeking to recreate f_{high} using a more comprehensible learning algorithm.

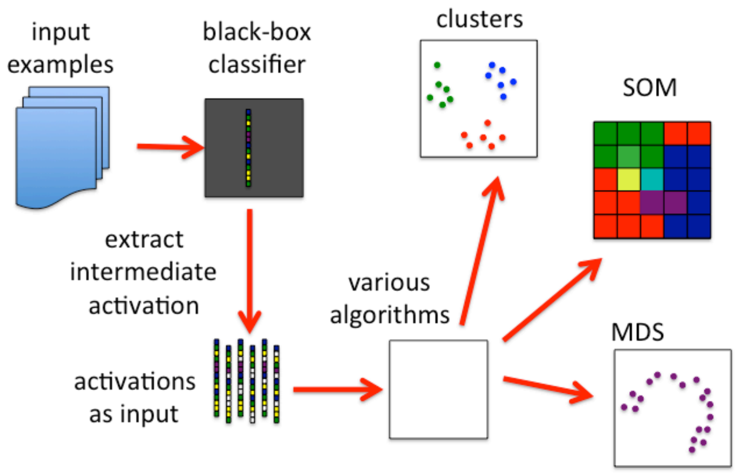
By comparison with a human exert, the aim is to have a level of explanatory model that is like a doctor saying, "this skin patch worries me because it is dark and irregular". The meanings of 'dark and 'irregular' may well be ones that one can only make sense of by looking at examples, but the high level decision making is explicit.



GB2. *Clustering and comprehension of low level.*
Here we focus on the mapping from input to intermediate activation. We effectively treat the intermediate activation (or a subset of it) as a feature vector (rather than the input) and seek to find clusters or other ways to organise the input space (e.g. multi-dimensional scaling, self-organising nets). This may involve initially reducing the example set to a similarity matrix where the cosine or other distance metric is used on the intermediate activations of each pair of examples.

Here we are trying to find ways to understand f_{low}: not primarily intending to replace it (although sometimes this may be possible), but to help create more comprehensible representations of the way it segments the input space.

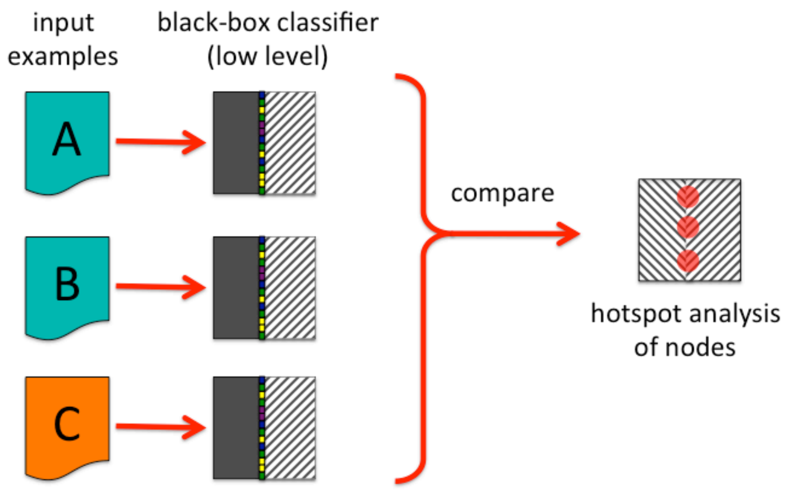
With the medical example, this would be like creating sets of examples of different features such as 'irregular' or 'dark. The centre/boundary exploration of BB3 may be useful here, but focused on the implicit classes revealed in the intermediate activation rather than the final output.



GB3. *Triad distinctions.*
This consists of taking triples of inputs A, B, C, where A and B are members of one class and C of another (or maybe self-organising versions without this constraint). The activations for A, B and C are examined to find main areas where A is similar to B, but different from C.

This is effectively an automated variant of the technique initially used in repertory grid interviews and then been adopted for human expertise elicitation.

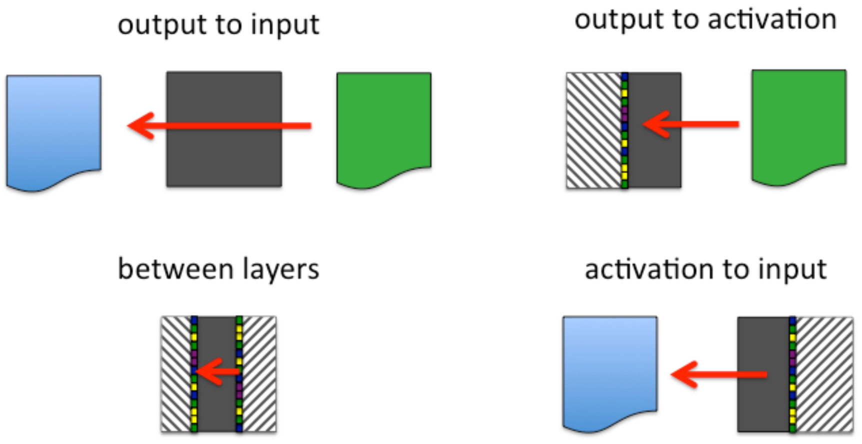
This can be used as a form of weighting/hotspot detection – so can be used like GB0 to help tune GB1 or GB2, or may be used to help in key feature detection, as in BB3. However, unlike BB3, this would be most likely used for actual known-value training set rather than generated examples.



GB4. *Apply generatively.*
Use the workings of the algorithm to generate inputs or intermediate activations that would give rise to a given final output feature or intermediate activation. That is effectively invert f, f_{low} or f_{high}.

You probably cannot fully invert these, but may be able to algorithmically help for the search of central or boundary cases BB3. For example, in a NN, backprop is effectively computing a point partial differential of output wrt node weights (parameters), which is then used to hill-climb the weights (for NN effectively a pseudo-stochastic hill-climb). However, the backprop procedure also gives (as a freebie!) the partial differential wrt inputs and activation; it thus enables hill-climbing (simulated annealing, etc.) on these as well.

The outputs of this may be used to drive other algorithms or as part of visualisation – the precise 'rules' for a effectively fuzzy distinction (e.g. 'irregular') may be hard to grasp, but examples may make it obvious.



In general apply your favourite method to some portion of the internal activation as a feature vector ☺