■ OBJECT AND MATERIAL EIGENFUNCTIONS – Cont'd.

• Laplacian eigenfunctions: ... Such a **"decider function"** $\mathcal{F}$ can have a small or large number of available scale-specific probability values p_i as argument input. Depending on the "needs and purpose" of a specific applied classification problem, specific relevant/irrelevant material characteristics at certain scales and "the expert's definition of a material match", **$\mathcal{F}$'s design must or should be a semantics-based design informed by a domain expert.** These are examples:

- One decides that two image segments must have (high) p-values only for a very small number of scales to match.
- Depending on the used imaging technology, the degree of noise at certain scales etc., one decides that only certain scales and their associated p-values should be involved in determining a match.
- An expert knows the exact scales necessary and sufficient to detect a match; thus, one should use these scales.

...

[Referring to the table on previous page]

- **A "decider function" $\mathcal{F}_{b_1 \ldots b_H}$, with bitstring index components $b_i \in \{0,1\}$, considers only probabilities p_i when $b_i = 1$.**
- **The operators used by a "decider function" $\mathcal{F}$ can be any combination of, for example, arithmetic and Boolean operators applied to the operands.**
- **Results of multiple $\mathcal{F}$ "decider functions" can be combined as well for a final decision.**

OBJECT AND MATERIAL EIGENFUNCTIONS – Cont'd.

• Laplacian eigenfunctions: ... Since multi-layer neural networks are of particular interest in the context of our overarching material data classification problem, some crucially important aspects and issues concerning these networks should be reviewed. Relevant topics include:

• Explainability of neural network's data processing and output generation

• Conversion of a neural network to a tractable, veryfiable circuit

• Considering that LOGICAL GATES can be implemented via neural networks: "Logical gate to (multi-layer) neural network"

• Considering that (multi-layer) NEURAL NETWORKS (with binary input) can be converted to tractable Boolean circuits: "Neural network to Boolean circuit"

• Necessity to adapt these model conversion methods to, for example, neural networks with non-binary input and activation functions that are not step activation functions.

...

• The Wall Street Journal (10 August 2017):

"It is now recognized that opacity, or lack of explainability, is 'one of the biggest obstacles to widespread adoption of artificial intelligence'."

• Simple standard operators are arithmetic (+, −, ·, /, ...), relational/comparative (<, =, >) and Boolean operators (¬, ∧, ∨); numbers and Boolean data are the "basic" data types processed by the operators. Understanding a neural network's operations "at the level of equivalent Boolean logical circuits" is crucial for verification.

OBJECT AND MATERIAL EIGENFUNCTIONS – Cont'd.

- Laplacian eigenfunctions – and the relationship to multi-layer neural networks:

- Hierarchy of LAYERS defined by Boolean expression (XNOR):
  Input: x_1, x_2 (numbers)
  Output:
  $$(((x_1>0)\wedge(x_2>0)) \vee ((\neg(x_1>0))\wedge(\neg(x_2>0))))$$
  $$\Leftrightarrow ((A \wedge B) \vee ((\neg A)\wedge(\neg B)))$$
  The "LAYERS of data processing" are defined by the order of operator applications: $>, \neg, \wedge, \vee$. These layers will be the layers of a multi-layer neural network design – when we DESIGN A NETWORK EXPLICITLY AND **NOT** VIA TRAINING. (The network must generate the expected output always correctly.)

- Need to consider noise and uncertainty: Include concepts of FUZZY LOGIC, FUZZY SYSTEMS etc. in the design, conversion, analysis and "verification" of Boolean circuits and neural networks (probability, p-values used with $p \in [0,1]$).
  See, for example, publications by Rudolf Kruse regarding computational intelligence, fuzzy logic and fuzzy systems.

- A trained multi-layer neural network classification architecture computes its output based on "operations that are not known explicitly" applied in multiple stages to the input data – using specific activation functions, bias values and 'optimized' weights. The essential task is:
  IF input satisfies condition for class i
  THEN return class i as output. !

...

■ OBJECT AND MATERIAL EIGENFUNCTIONS – Cont'd.

• Relationship to neural networks: ...

• Basic simple SIGMOID model:

Input: data $x_1, \ldots, x_n$;

weights $w_1, \ldots, w_n$

Output: activation value f

Algorithm:

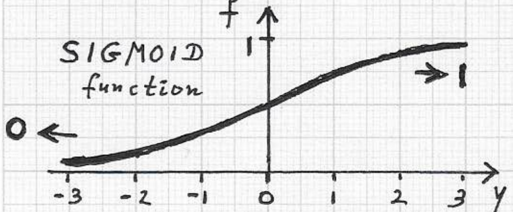
– compute linear function

$$y = b + \sum_{i=1}^{n} w_i x_i \,,$$

with b being "bias"

– compute activation function

$$f = 1/(1+e^{-y})$$



sketch of activation function

Use of the sigmoid function in the context of Boolean algebra:

$$f(y) \overset{y \to -\infty}{=} 0 \quad \text{"false"}$$

$$f(y) \overset{y \to \infty}{=} 1 \quad \text{"true"}$$

(The "step function" $f(y)=0$, $y<0$, and $f(y)=1$, $y>0$, is obtained by

$$f(y) = 1/(1+e^{-\gamma y}),$$

for $\gamma \to \infty$.)

• To determine whether a trained neural network performs the "IF... THEN... task" correctly for a specific classification input-output pair, one should (when possible) convert the network to its equivalent Boolean logic circuit. One can then investigate whether this circuit indeed correctly generates the result output expected for a particular input.

We briefly review how one can implement basic relational and Boolean operators via a simple standard neural computing model based on a sigmoid activation function, see left figure.

i) Comparison '>' conditions:

$$f(0) = \tfrac{1}{2} \Rightarrow 1/(1+e^{-(b+w\cdot 0)}) \overset{!}{=} \tfrac{1}{2}$$

$$\Rightarrow 1/(1+e^{-b}) \overset{!}{=} \tfrac{1}{2} \Rightarrow b = 0$$

$$f(1) = \tfrac{999}{1000} \Rightarrow 1/(1+e^{-w}) = \tfrac{999}{1000}$$

$$\Rightarrow 1 + e^{-w} = \tfrac{1000}{999} \Rightarrow w = \ln(999)$$

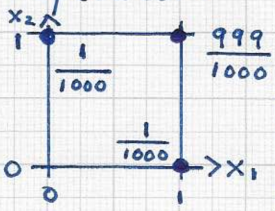
$$\Rightarrow f(-1) = \tfrac{1}{1000} \text{ (due to symmetry)}$$

...

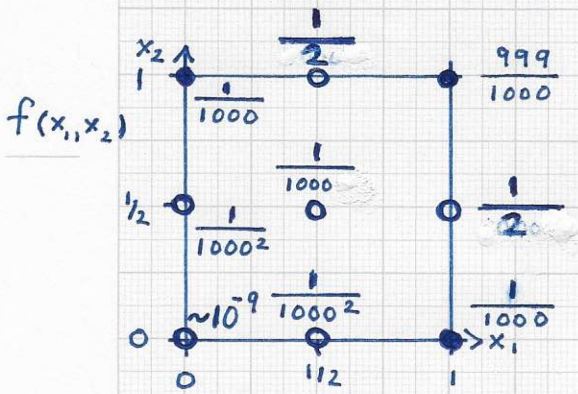
■ OBJECT AND MATERIAL EIGENFUNCTIONS – Cont'd.

• Laplacian eigenfunctions and multi-layer neural networks: ...

• 3 "conditions" for '$\wedge$' operator:



3 f-values specified at specific points '•'



Design of an activation function for the '$\wedge$' operator. Three specified activation function values (top) leads to the final definition of the analytical activation function (bottom). Nine function values are provided in the figure for nine parameter value tuples (x_1, x_2).

! The activation function f return values close to 1 when the values of x_1 and x_2 are close to 1 – by design. BUT: It is only an analytical, smooth approximation of the '$\wedge$' operator.

ii) Boolean operator '$\wedge$':

For the '$\wedge$' operator one must design a linear function $y = b + w_1 x_1 + w_2 x_2$, see figure (left). For example, one can consider the following three conditions for the three parameters b, w_1, and w_2:

$$f(1,1) = \frac{999}{1000} \Rightarrow 1/(1+e^{-(b+w_1+w_2)}) = \frac{999}{1000}$$

$$\Rightarrow b + w_1 + w_2 = \ln(999) \quad ;$$

$$f(1,0) = \frac{1}{1000} \Rightarrow 1/(1+e^{-(b+w_1)}) = \frac{1}{1000}$$

$$\Rightarrow b + w_1 \quad = -\ln(999) \quad ;$$

$$f(0,1) = \frac{1}{1000} \Rightarrow 1/(1+e^{-(b+w_2)}) = \frac{1}{1000}$$

$$\Rightarrow b + \quad w_2 = -\ln(999) .$$

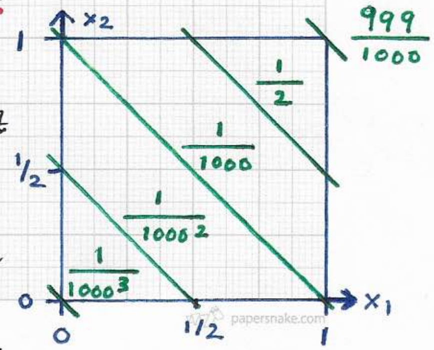
The solution of this 3×3 linear system of equations is:

$$b = -3L \ , \quad w_1 = w_2 = 2L \ , \quad \text{where } L = \ln(999).$$

Thus, the resulting activation function is

$$f(x_1,x_2) = 1/(1+e^{-L(-3+2x_1+2x_2)}) .$$

• **Note.** The contour line behavior of f is shown in the right figure. The contour **LINES** are given by



$$f(x_1,x_2) = c$$

$$\Rightarrow 1 + e^{L(3-2x_1-2x_2)} = \frac{1}{c}$$

$$\Rightarrow \ldots \Rightarrow x_1 + x_2 = \text{'const'}$$

...