Artificial Neural Networks
in
Artificial Intelligence

Artificial Intelligence involves several different approaches, including

- **Symbolic artificial intelligence** is the collection of all methods in artificial intelligence research that are based on high-level "symbolic" (human-readable) representations of problems, logic and search.

- **Bayesian decision networks** are graphical models that represent a set of variables and their dependencies ... ideal for taking an event that occurred and predicting the likelihood that any one of several possible causes was the contributing factor.

- **Evolutionary algorithms** use mechanisms inspired by biological evolution, such as reproduction, mutation, recombination, and selection. Candidate solutions play the role of individuals in a population, and the fitness function determines the quality of the solutions. Evolution of the candidate population then takes place after the repeated application of the mechanisms.

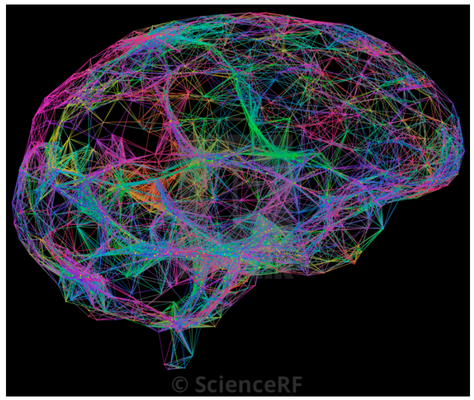
Wikipedia

Here we illustrate a fourth approach

- **Artificial Neural Networks** - with basic introductory examples here

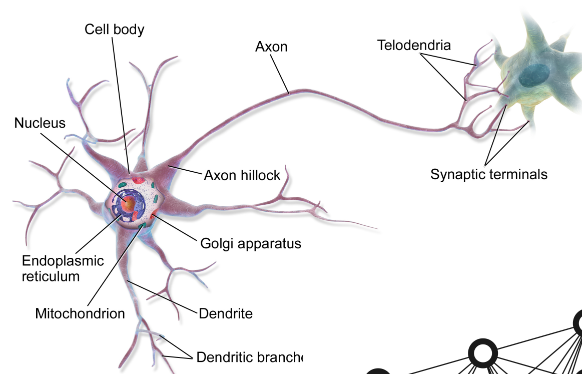
www.braininjuryaustralia.org.au

Our brains sense and think using connected networks of cells called neurons

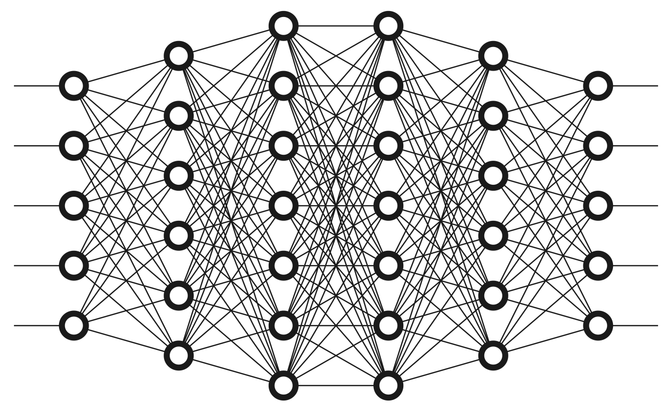


@ScienceRF

Blausen_0657_MultipolarNeuron



These networks are the inspiration for computer simulations of “neural networks,” which can be trained to sense and solve complex problems



neural_network_shutterstock_all_is_magic.jpg

github.com/RichardHerz

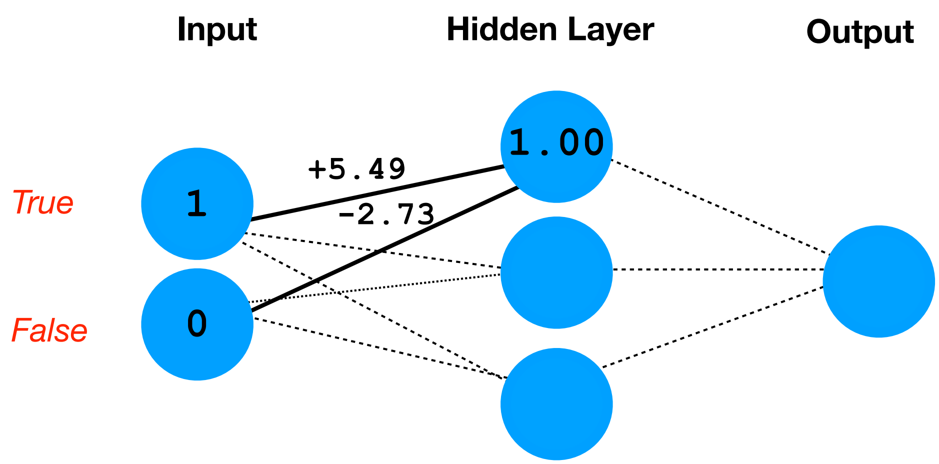
Neural Network

A simple example

Simulates XOR logic - exclusive or

Output is TRUE when one input is TRUE but not both

2 inputs, 1 output, 1 hidden layer with 3 neurons & 9 synapses



These sums over all nodes in a layer are the product of matrix multiplication. Matrix multiplication is well suited to being accelerated in hardware Graphical Processing Units, since graphic transformations also involve matrix multiplication.

EXAMPLE for input of 1
0

node value = sigmaFunc(sum of (connection weight * node activation))

where sigmaFunc(x) = exp(x) / (1 + exp(x)) >> converts all input x values into range 0 to 1

INPUT > HIDDEN LAYER

sigmaFunc(5.4868 * 1 + (-2.7276) * 0) = 0.9959 = hidden node 1 activation

Every node - neuron - has a connection - synapse - to every neuron in nearest-neighbor layers of neurons in this basic type of neural network.

The values are held in memory locations and the CPU executes the math - there are no physical, hardware neurons and synapses.

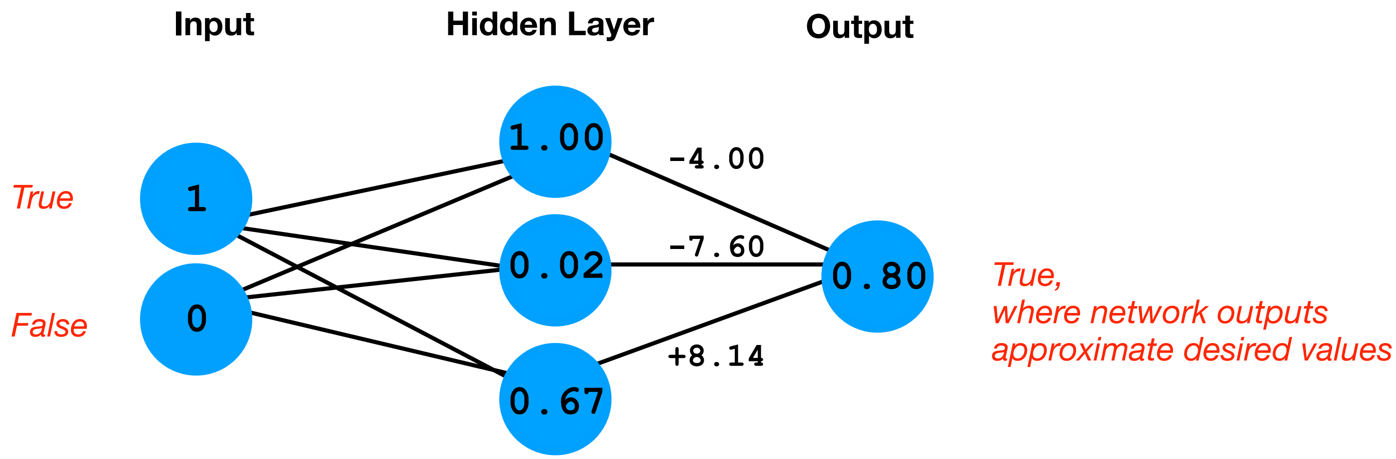
github.com/RichardHerz

Neural Network

Simulates XOR logic - exclusive or

Output is TRUE when one input is TRUE but not both

2 inputs, 1 output,
1 hidden layer
with 3 neurons &
9 synapses



HIDDEN LAYER > OUTPUT

$$\text{sigmaFunc}(\ (-4.0030) * 0.9959\ + (-7.5988) * 0.0153 + 8.1402 * 0.6719\ ) = 0.7969 = \text{output node}$$

The MATLAB code to solve for the output remains the same as that below, regardless of the size of the network:

```
for i = 2 : numHiddenLayers + 2
    a{i} = sigmaFunc( W{i-1} * a{i-1} );
end
```

W *is a cell array whose elements are the matrices of synapse weights for each layer;* ***a*** *is a cell array whose elements are the vectors of neuron activation values. Each set of* ***W*** *and* ***a*** *are matrix-multiplied to obtain the neuron activation values for the next layer in the series of neuron layers.*

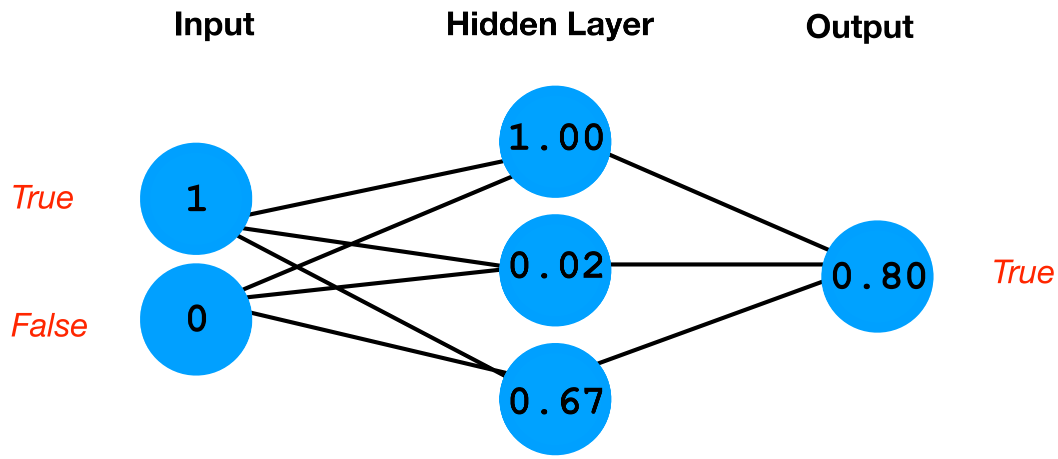
github.com/RichardHerz

Neural Network

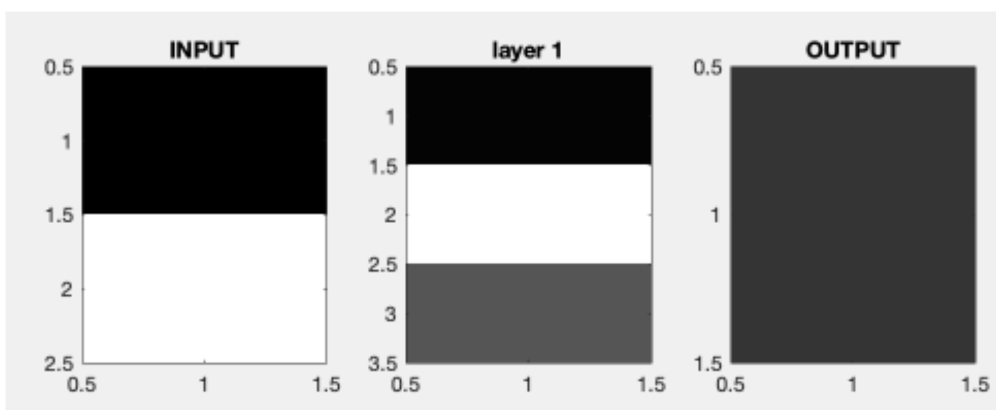
Simulates XOR logic - exclusive or

Output is TRUE when one input is TRUE but not both

2 inputs, 1 output, 1 hidden layer with 3 neurons & 9 synapses



Visualization of neuron values - “activations” - for this input



1 0	0.9959 0.0153 0.6719	0.7969 **True**

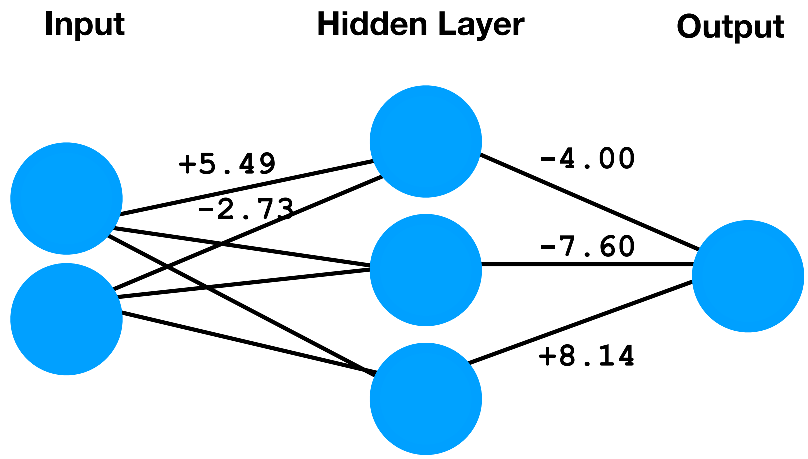
github.com/RichardHerz

Neural Network

Simulates XOR logic - exclusive or

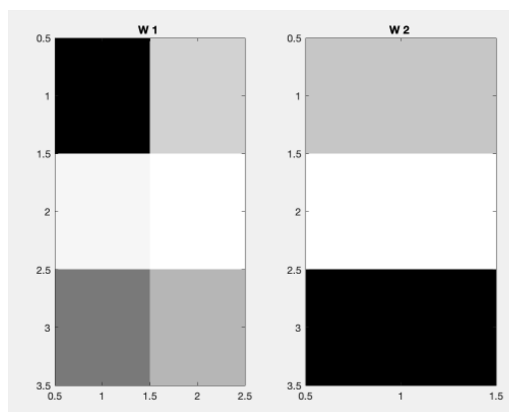
Output is TRUE when one input is TRUE but not both

2 inputs, 1 output,
1 hidden layer
with 3 neurons &
9 synapses



Visualization of synapse connection “weights” to hidden layer and to output
min = -7.60 (white), max = +8.14 (black)

5.4868	-2.7276
-4.1663	-4.4991
0.7169	-1.5748

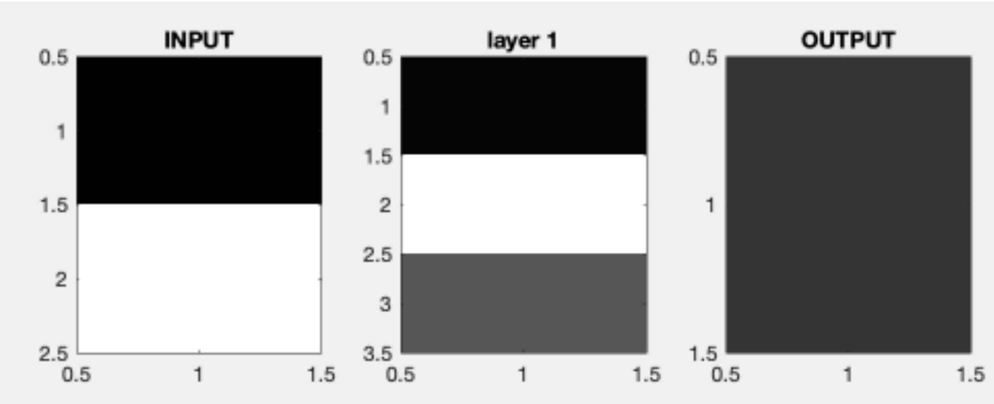


-4.0030
-7.5988
8.1402

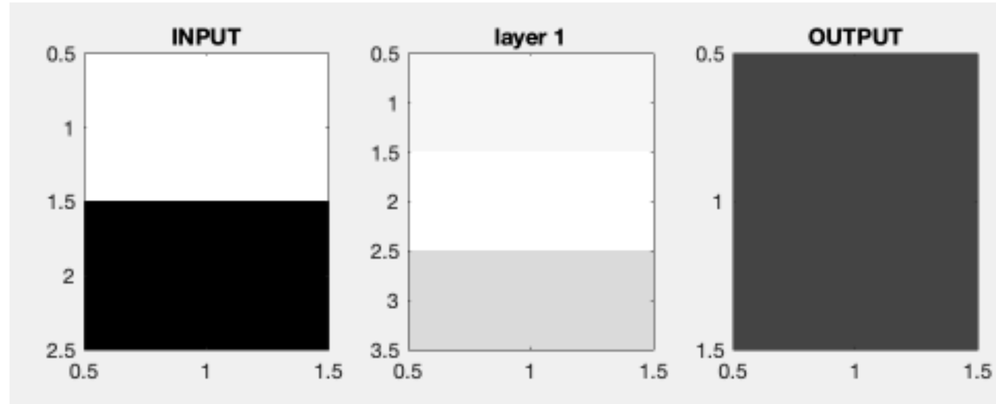
The synapse connection weights were determined when the network was “trained” using combinations of known inputs and outputs.

github.com/RichardHerz

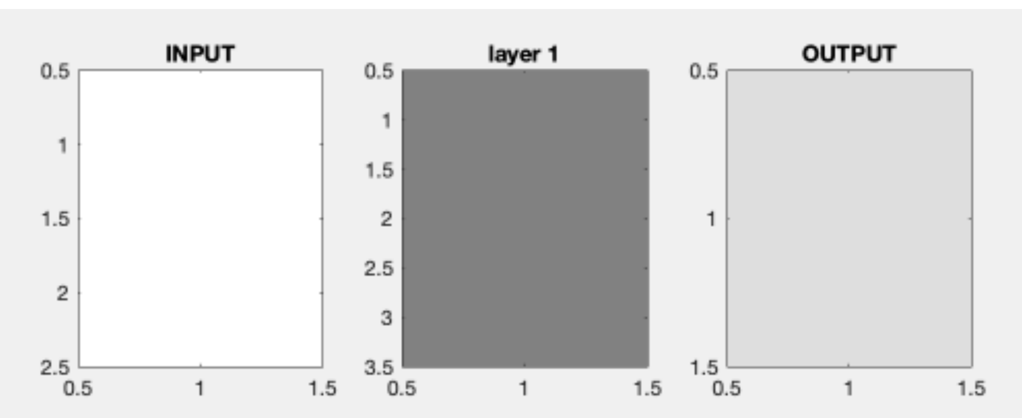
Visualizations of node activations: input > hidden layer > output



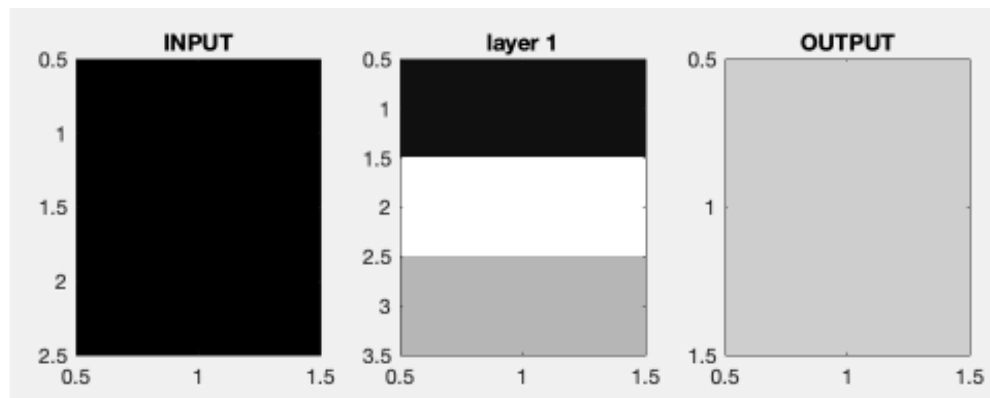
1 0	0.9959 0.0153 0.6719	0.7969 **True**



0 1	0.0614 0.0110 0.1715	0.7440 **True**



0 0	0.5000 0.5000 0.5000	0.1505 **False**



1 1	0.9404 0.0002 0.2978	0.2072 **False**

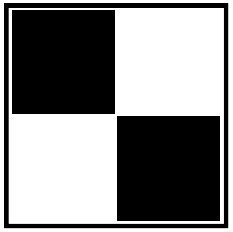
Output is TRUE when one input is TRUE but not both

github.com/RichardHerz

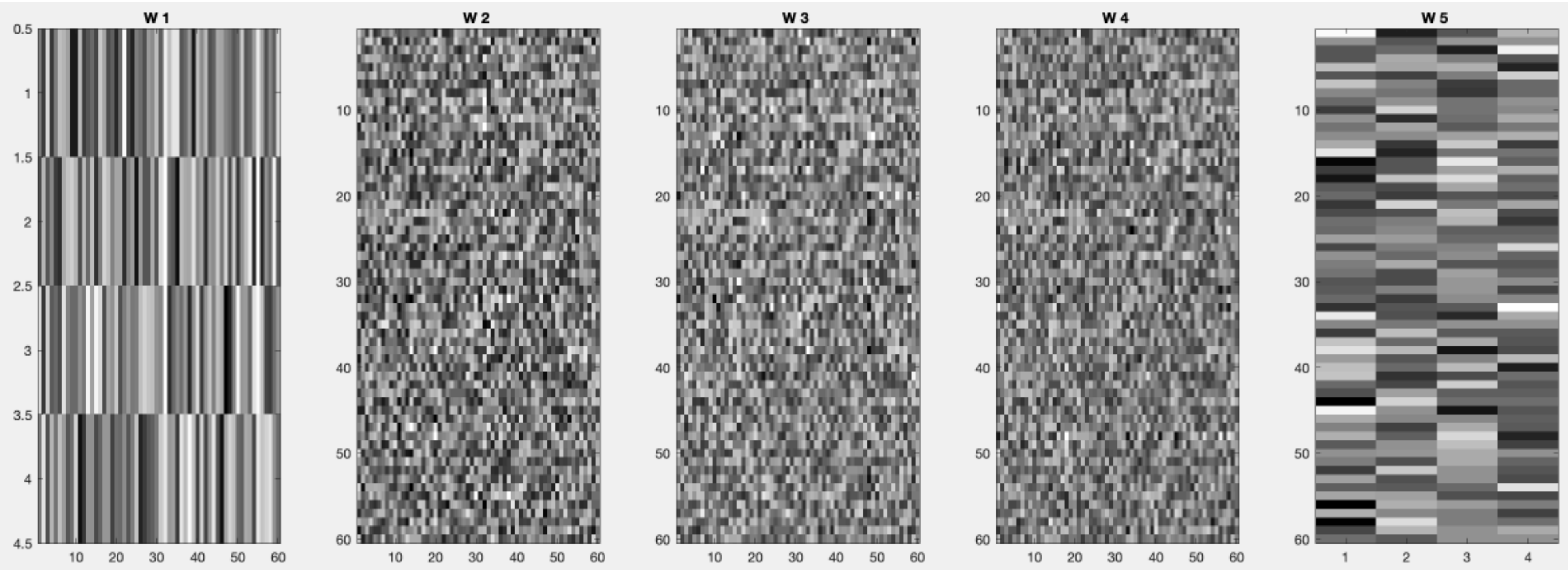
<u>*Neural Network*</u>

4 inputs, 4 outputs
4 hidden layers, each with 60 neurons = 240 neurons & 11,280 synapses

A more complex network which detects diagonal, horizontal and vertical inputs to a 2 x 2 "touch screen"



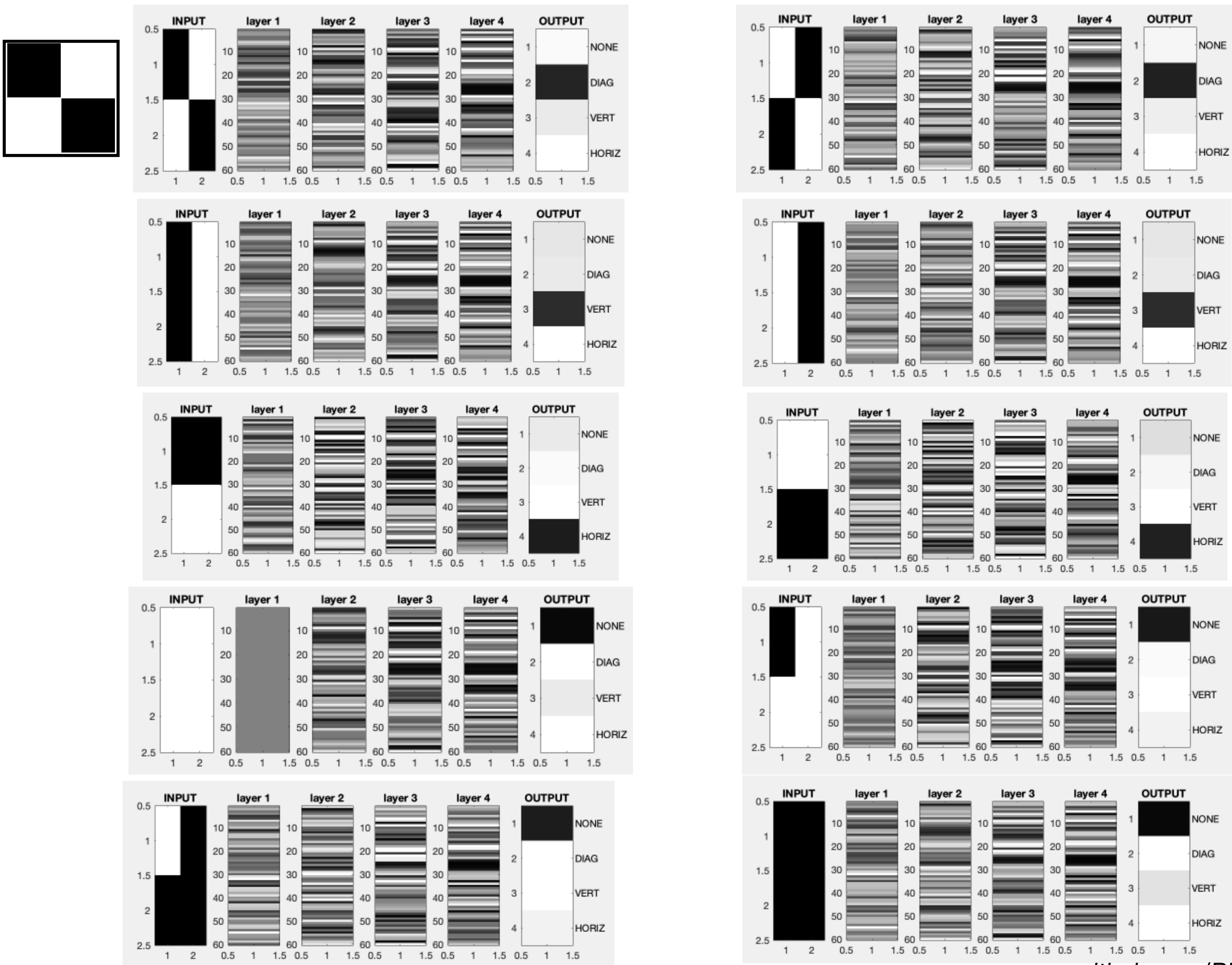
Visualization of synapse weights to hidden layers 1-4 and to output, min = -1.23 (white), max = +1.25 (black)



Prior to training the network with known input and output cases, the connection weights were assigned random values in the range -1 to +1. Then the weights were adjusted during training in order to match input cases with their corresponding outputs. The resulting weights are not random. Different sets of weights will be obtained with different random initializations

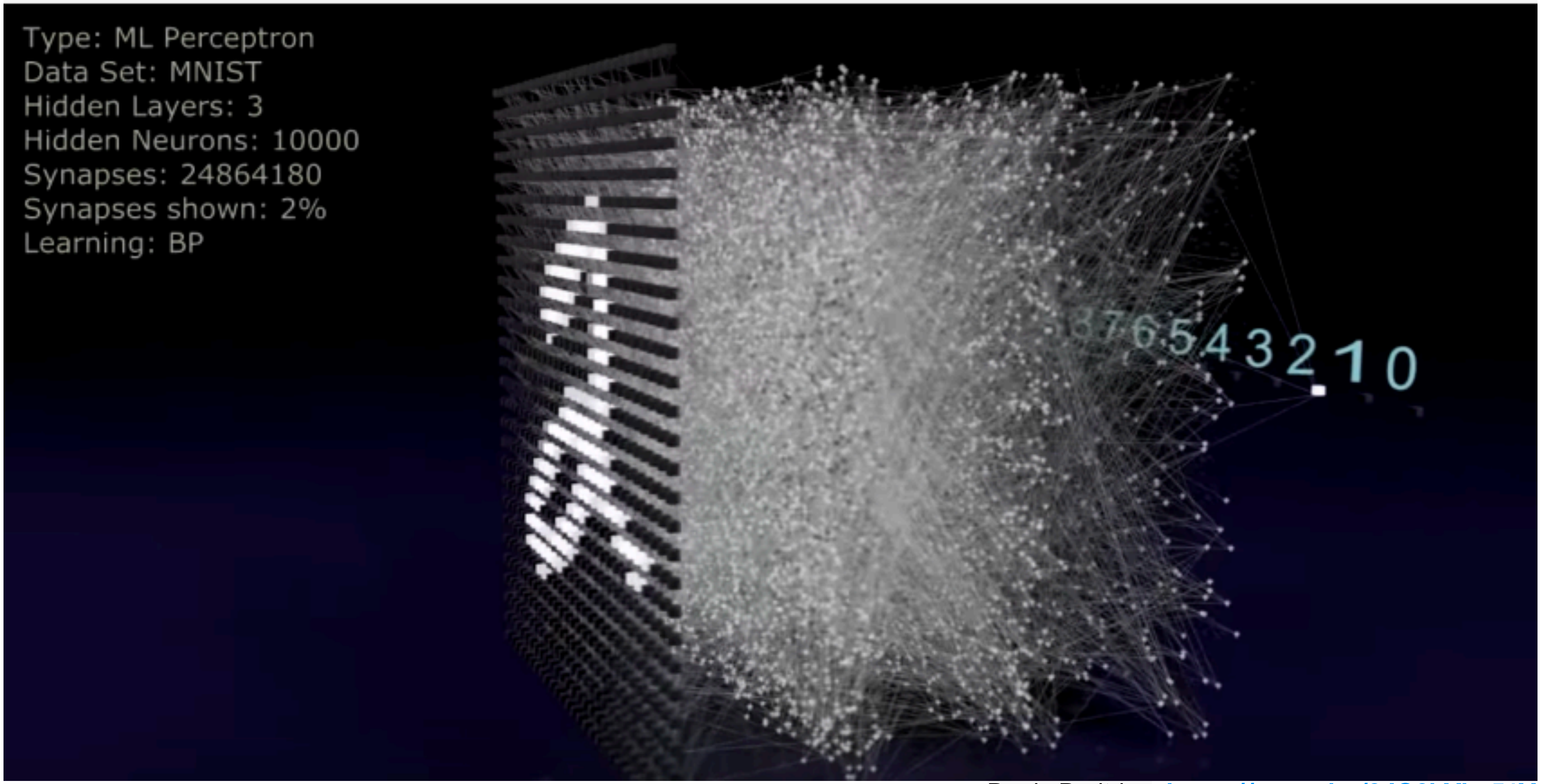
github.com/RichardHerz

Visualization of node activations: input > 4 hidden layers > output



github.com/RichardHerz

A neural network for a 28 x 28 “touch screen”
note that only 2% of the 24+ million “synapses” are shown



Denis Dmitriev https://youtu.be/3JQ3hYko51Y

A neural network represents a large number of coupled equations which, when given a set of input values, can produce a set of desired output values.

The more neurons and synapses - the more equations - and the greater complexity of inputs and outputs which can be "fit" by the system of equations. Note the significant increase in complexity going from the XOR example to the 2 x 2 “touch screen” example to the 28 x 28 touch screen in the figure above.

"Deep learning" refers to solving complex problems using many hidden layers of neurons - many equations - and more complex network structures.

A neural network might be thought of as a general function which can fit anything given enough terms…

In a sense, neural networks are math functions which can “fit” any desired input and output data given enough adjustable parameters, which are the “synapse” connection weights and, thus, enough neurons.

Using a neural network is somewhat similar to using a polynomial to fit a series of data points (empirical fit) vs. using a functional form that represents the underlying physics (theoretical fit).

In a neural network, the functional form is fixed by the network structure. The values of the constants in the function are the connection weights, whose values are determined during training.

For the XOR network above, this is the Matlab code which computes the output a{3} given the input a{1}

```
for i = 2:3
    a{i} = sigmaFunc( W{i-1}*a{i-1} );
end
```

Matrix W{i-1} and vector a{i-1} are elements of the cell arrays W and a. They are matrix multiplied. The Matlab code is very compact. We can see the form of this network’s function by looking at the expanded equation. The output a{3} is a function of the inputs a{1}

$$a^{\{3\}} = f(a^{\{1\}}) = \sigma\left(W_1^{\{2\}}a_1^{\{2\}} + W_2^{\{2\}}a_2^{\{2\}} + W_3^{\{2\}}a_3^{\{2\}}\right)$$

$$= \sigma\left(W_1^{\{2\}}\sigma\left(W_{1,1}^{\{1\}}a_1^{\{1\}} + W_{1,2}^{\{1\}}a_2^{\{1\}}\right) + W_2^{\{2\}}\sigma\left(W_{2,1}^{\{1\}}a_1^{\{1\}} + W_{2,2}^{\{1\}}a_2^{\{1\}}\right) + W_3^{\{2\}}\sigma\left(W_{3,1}^{\{1\}}a_1^{\{1\}} + W_{3,2}^{\{1\}}a_2^{\{1\}}\right)\right)$$

where, for more compact notation, the superscript {n} of cell arrays a and W denotes a matrix in cell array element n, and the subscripts are the indices within that matrix. The hidden layer activations are a{2}. The nonlinear activation function for this network, which constrains activation values between 0 and 1, is

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

For a larger neural network of this type, there are more terms but the functional form remains unchanged. Modern computers allow large networks to be computed rapidly.