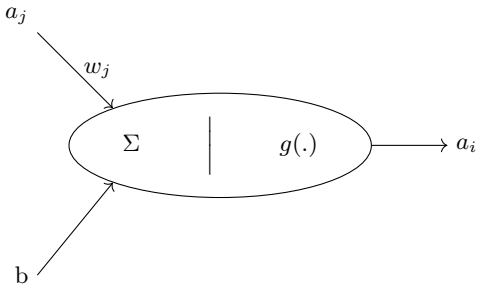
These have been around for approximately 50 years, but have had many ups and downs.
They have had a large impact in image analysis and sequence to sequence translation.
Neural networks are a model-free approach to problem solving based on function fitting.
Traditional approaches are model-based and work via representation and reasoning.

The basic unit of a neural network is the **NEURON**.
These map a set of input activations to an output activation.

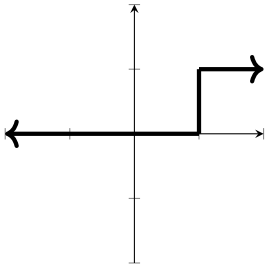


a — activations
w — weights
b — bias
g(.) — activation function
ai = g($[\sum w_{ij} a_j] + b$)

Activation Functions come in the following forms (generally):

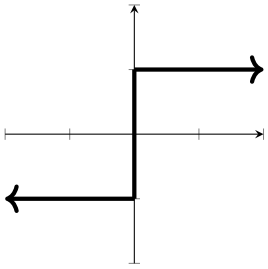
Step Function:

$$g(x) = \left\{ \begin{array}{ll} 1 & x \geq t \\ 0 & x < t \end{array} \right\}$$



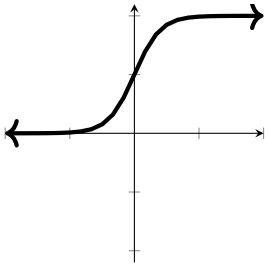
Sign Function:

$$g(x) = \left\{ \begin{array}{ll} 1 & x \geq t \\ -1 & x < t \end{array} \right\}$$



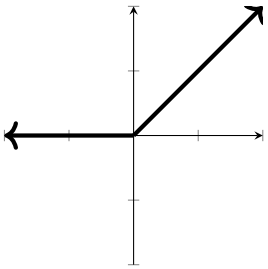
Sigmoid Function:

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



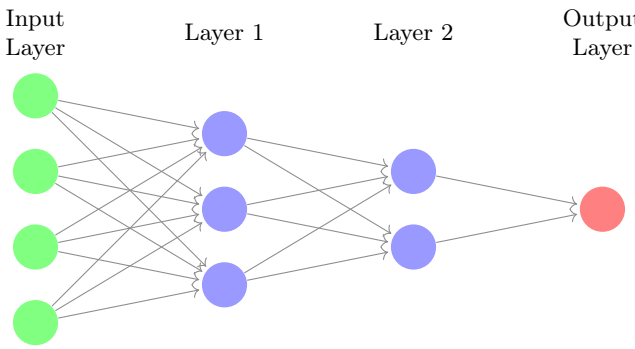
Step Function:

$$g(x) = max(0, x)$$



A connected set of these forms a neural network!
Modern neural networks also use non-neuron elements, but they are application specific.

We could learn the structure, but many situations have heuristically derived templates.
The specific forms of neural networks we consider are **Feed-Forward Neural Networks**.



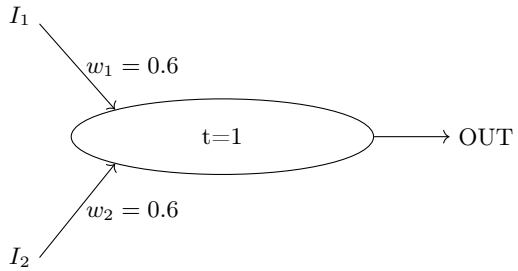
Greater depth requires more training, sp multi layer NN's are computation-heavy.
They used to be nearly impossible to train, but deep-learning has changed this.
Neural networks are universal function approximators, since they map in → out.
Are they expressive enough?

YES; by the above, they can express every function up to a constant error.

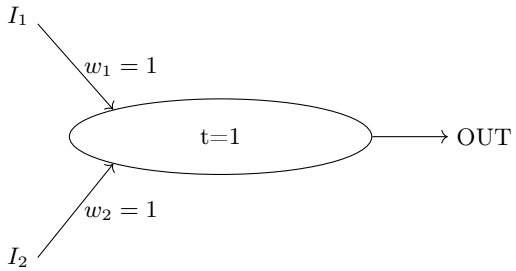
Pure-Step Neural Networks

Basic gates can be formed by networks consisting only of neurons with step functions.

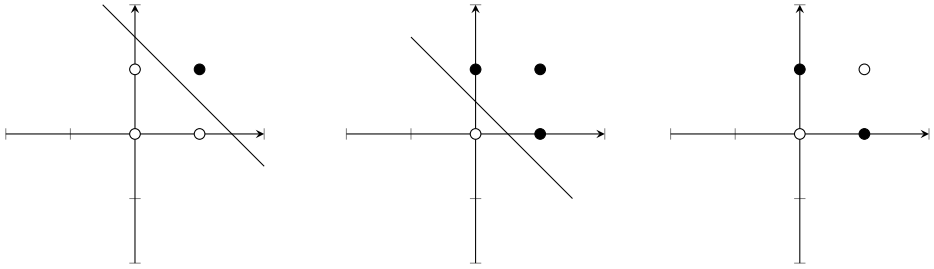
OR gate:



AND gate:

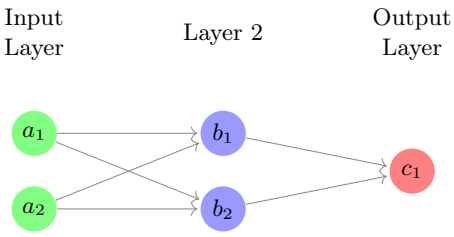


We cannot, however, form an XOR gate from a single neuron.
This is because an XOR gate is not **linearly separable**.



AND Gate OR Gate XOR Gate

Our neuron functions can only represent linearly separable functions.
Networks, however, can represent much more complex representations.



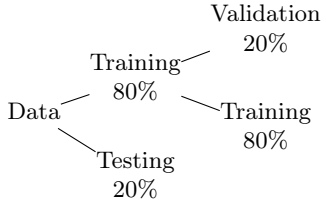
$$
\begin{aligned}
c_1 &= g(w_{b_1c_1}b_1 + w_{b_2c_1}b_2) \\
b_1 &= g(w_{a_1b_1}a_1 + w_{a_2b_1}a_2) \\
b_2 &= g(w_{a_1b_2}a_1 + w_{a_2b_2}a_2) \\
\rightarrow c_1 &= g(w_{b_1c_1}g(w_{a_1b_1}a_1 + w_{a_2b_1}a_2) \\
&\quad + w_{b_2c_1}g(w_{a_1b_2}a_1 + w_{a_2b_2}a_2)) \\
&\sim f(a_1, a_2, b_1, b_2, c_1)
\end{aligned}
$$

This neural network, as can be seen, represents a complex non-linear function.
We train the neural network by setting the weights to match labeled data.
If I fix the values of a_1 and a_2, $a_5 = f_1(w_{13}, ..., w_{45}) = f_2(w_{13}, ..., w_{45})$.
Thus this is just a complex case of function fitting!
Modern neural networks can have billions of parameters, so this can grow very complex.

Training neural networks is an exercise in optimization.
Our optimization criteria is called the **loss function**.
This has a couple common forms:

- cross entropy
- mean-squared error $= \frac{1}{N}\sum(\text{NN}(L_i) - L_i)^2$.
  This is of the form f(w1, . . . , wk), so we can optimize by weight!

We do the actual optimization via gradient descent.
Tensor flow, RUST, and the ADAM optimizer can do this for us.
We can stochastically break the data to increase the accuracy of the fit.



We call a single iteration of the gradient-descent algorithm an **epoch**.
How do we know which epoch to stop on?

1. set a limit on epoch count
2. stop when epoch stops changing much (patience)

We could also divide the data into batches to run optimizations in parallel.
We thus perform an iteration on all batches per epoch.
This has two major advantages:

1. parallelism
2. randomization (stochastic gradient descent)

A batching approach can run extremely fast on GPU's rather than CPU's!