Mathematical Circuits in Neural Networks

By: Sean Osier

Motivation

Olah et al. make three claims about the fundamental interpretability of neural networks

THREE SPECULATIVE CLAIMS ABOUT NEURAL NETWORKS

Claim 1: Features

Features are the fundamental unit of neural networks. They correspond to directions. [1] These features can be rigorously studied and understood.

Claim 2: Circuits

Features are connected by weights, forming circuits. [2] These circuits can also be rigorously studied and understood.

Claim 3: Universality

Analogous features and circuits form across models and tasks.

Left: An activation atlas [13] visualizing part of the space neural network features can represent.

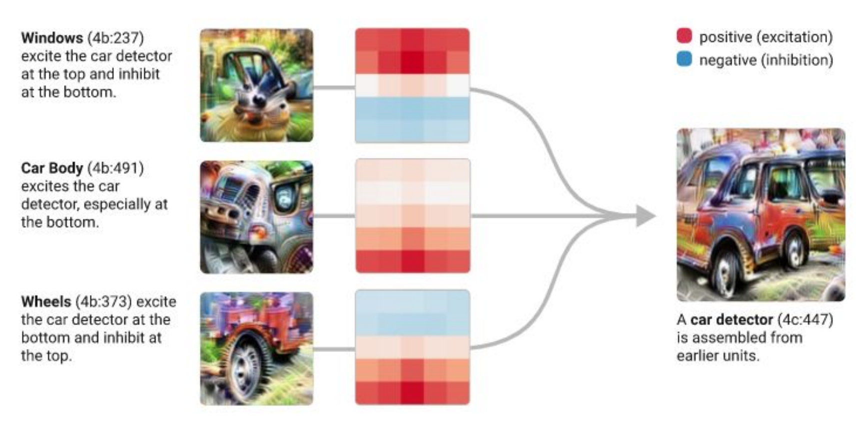
Source: https://distill.pub/2020/circuits/zoom-in/

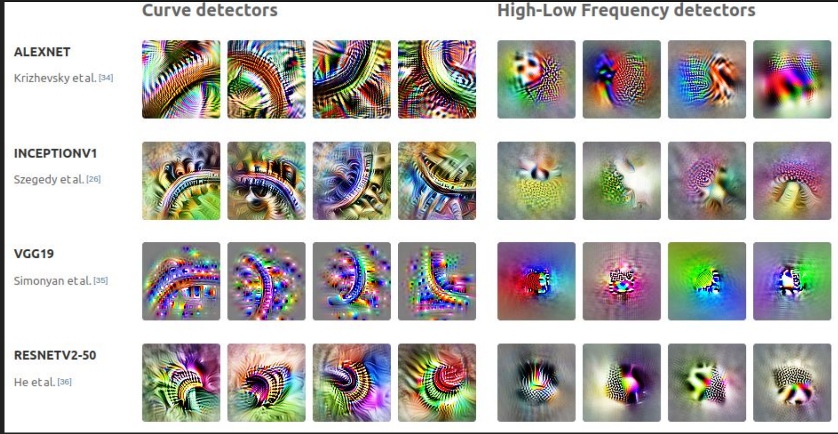
They demonstrates these claims in the context of image models

1. Features:

2. Circuits:

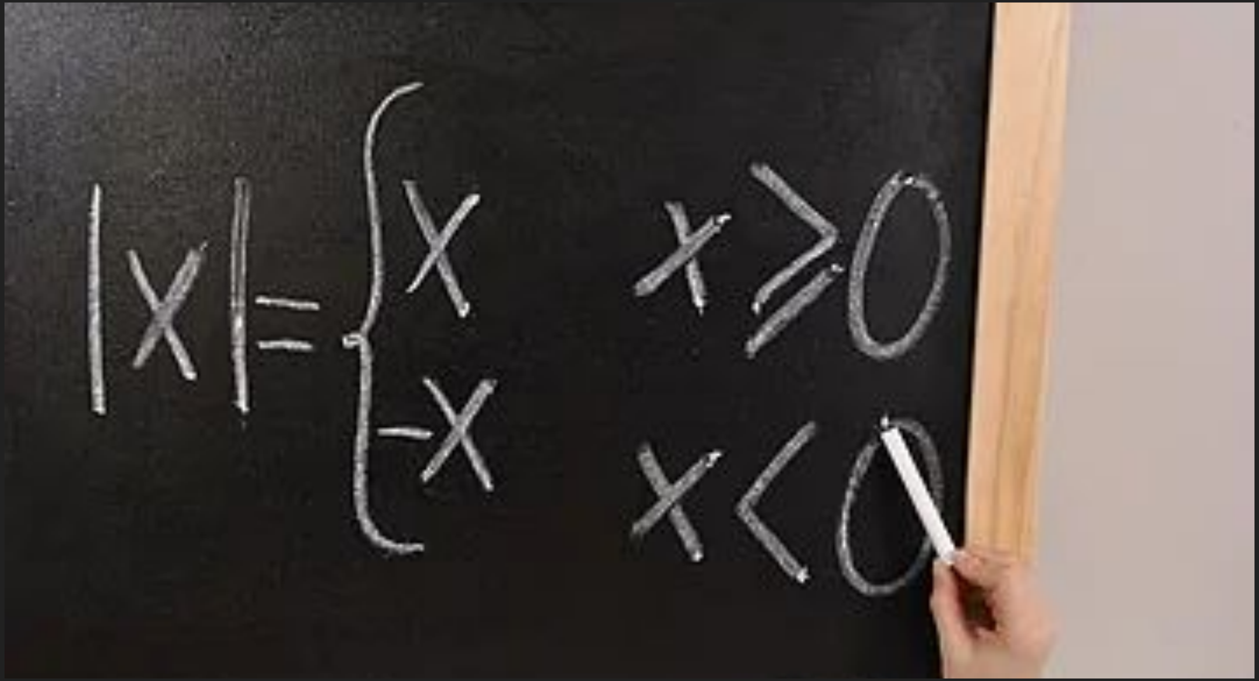
3. Universality:





Source: https://distill.pub/2020/circuits/zoom-in/

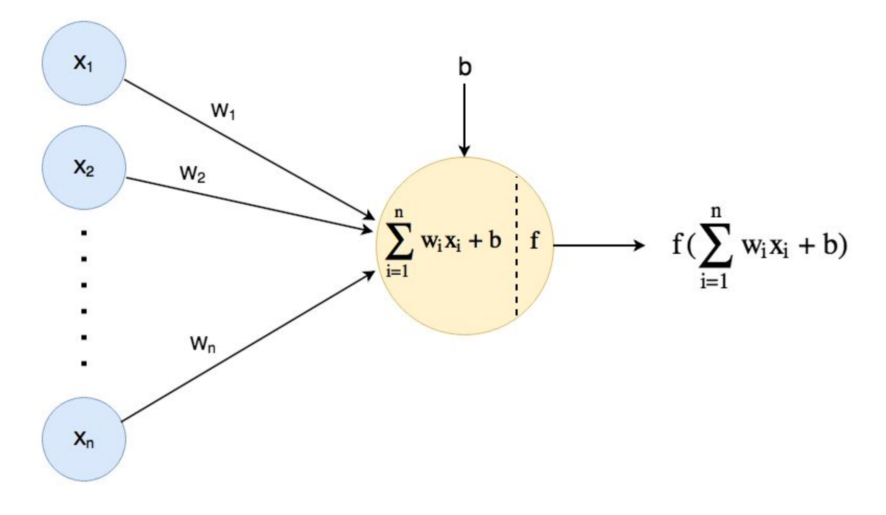
This work demonstrates the same concepts apply in the space of neural networks modeling basic mathematical functions



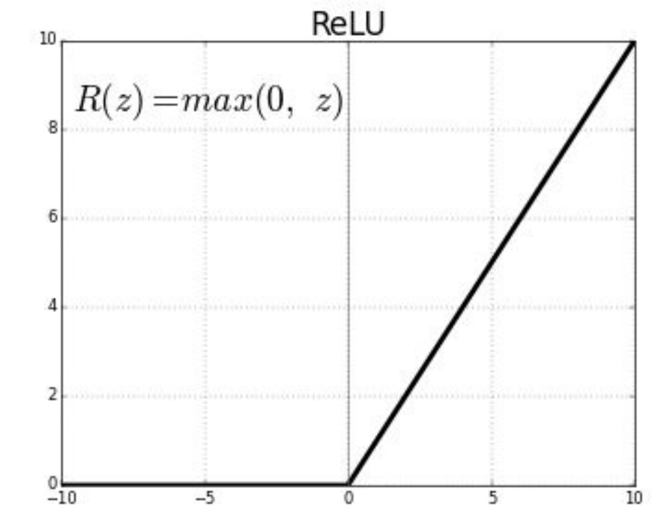
Background

Basic Neural Network Math

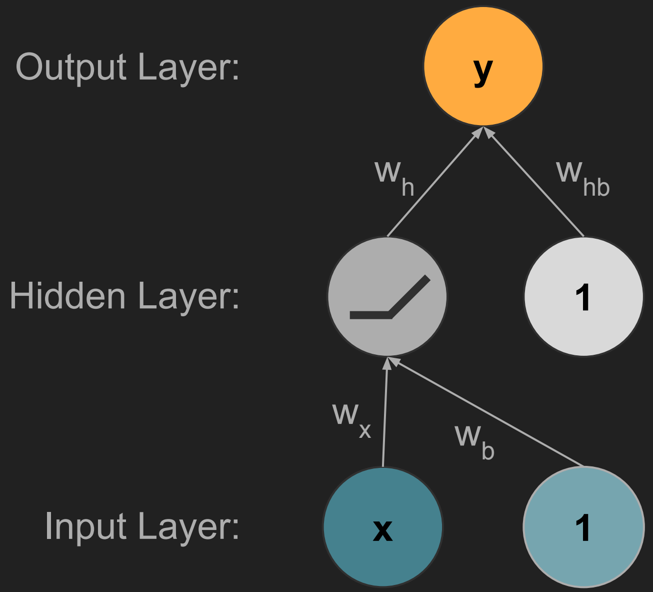
How a Single Neuron Works:

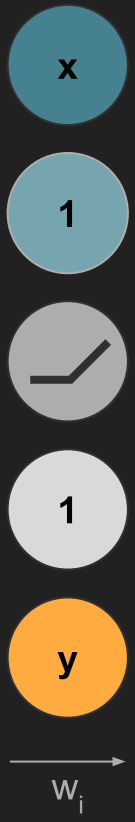


The ReLU Activation Function:



Legend





= The input (a number)

= Bias node for the input layer

= Hidden layer node with relu

= Bias node for the hidden layer

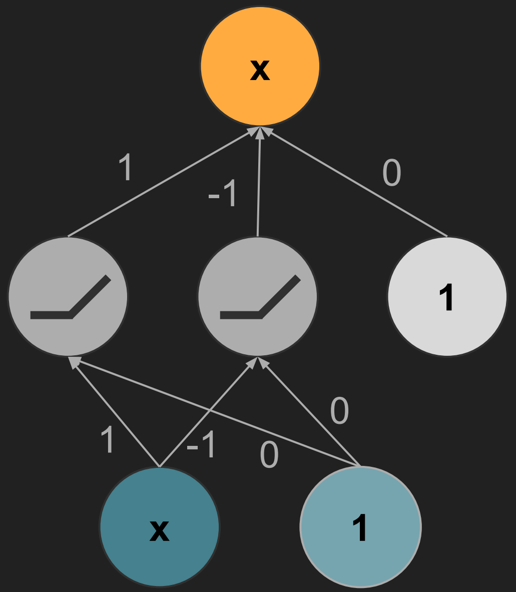
= The output (a number)

= A connection with its weight

Single Number Inputs

Identity Function: *f(x)* = *x*

Optimal Solution:

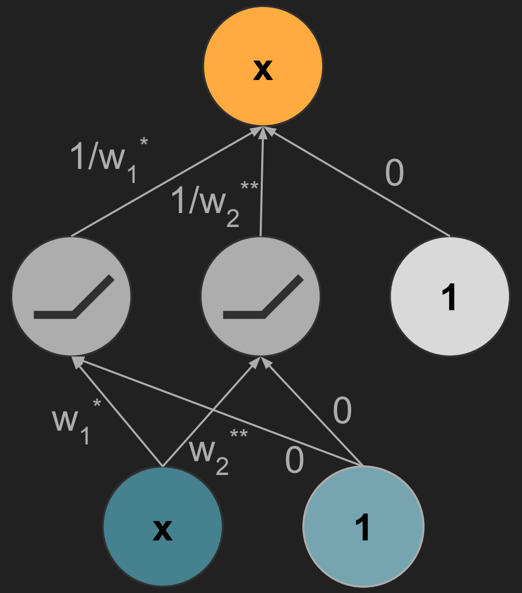


Notes:

- All biases are zeroed out
- 2 hidden nodes are required:
  - One detects x's "positiveness" (on left here)
  - One detects x's "negativeness" (on right here)
- Final layer reconstructs x from it's positiveness and negativeness scores

Identity Function: *f(x)* = *x*

More Precisely:



* where w_1 is positive
** where w_2 is negative

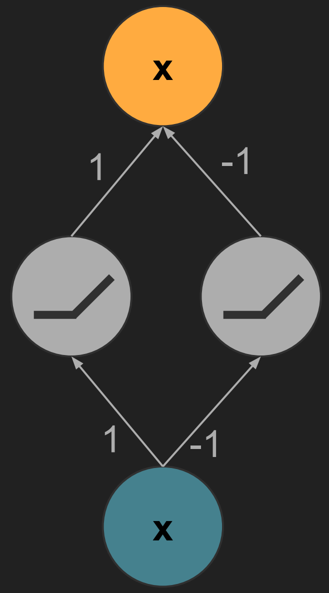
Notes:

- All biases are zeroed out
- 2 hidden nodes are required:
  - One detects x's "positiveness" (on left here)
  - One detects x's "negativeness" (on right here)
- Final layer reconstructs x from it's positiveness and negativeness scores

- *w* and *1/w* ensure the final result is is neither scaled up or down, while allowing for infinitely many optimal solutions

Identity Function: *f(x) = x*

Simplified:



Notes:

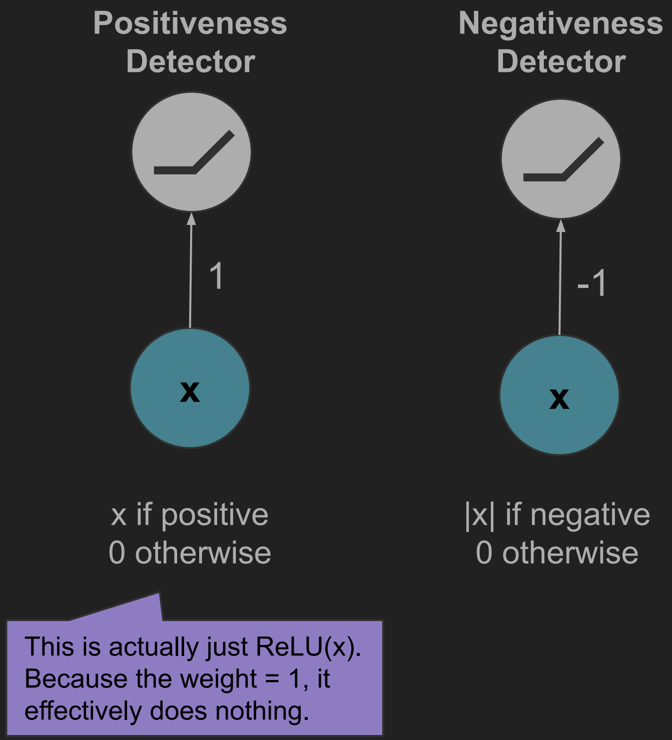
- All biases are zeroed out
- 2 hidden nodes are required:
  - One detects x's "positiveness" (on left here)
  - One detects x's "negativeness" (on right here)
- Final layer reconstructs x from it's positiveness and negativeness scores

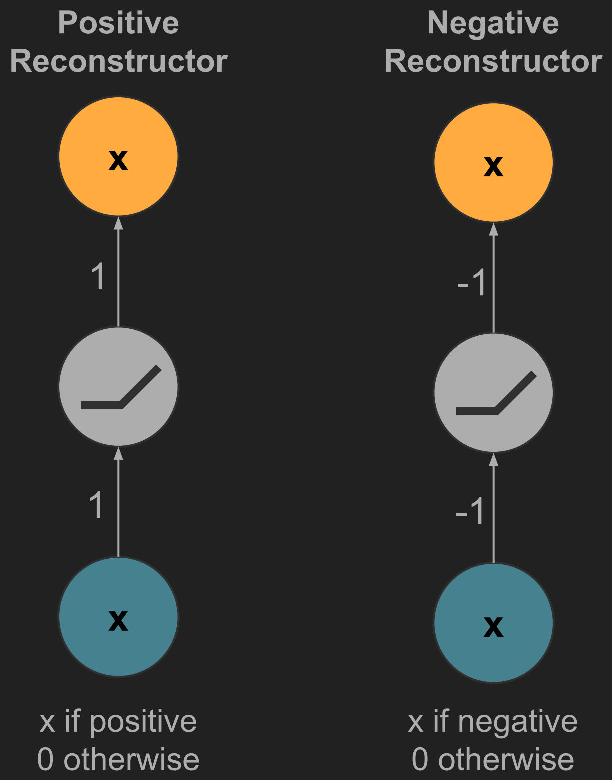
Identity Function: *f(x)* = *x*

Examples:

Identity Function: *f(x)* = *x*

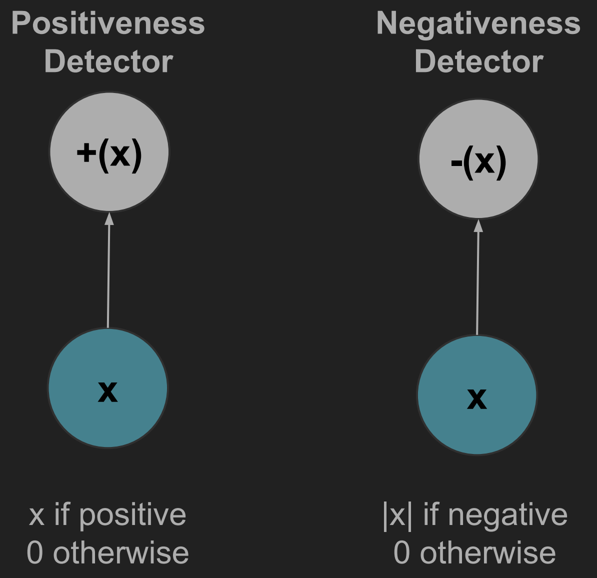
Features:



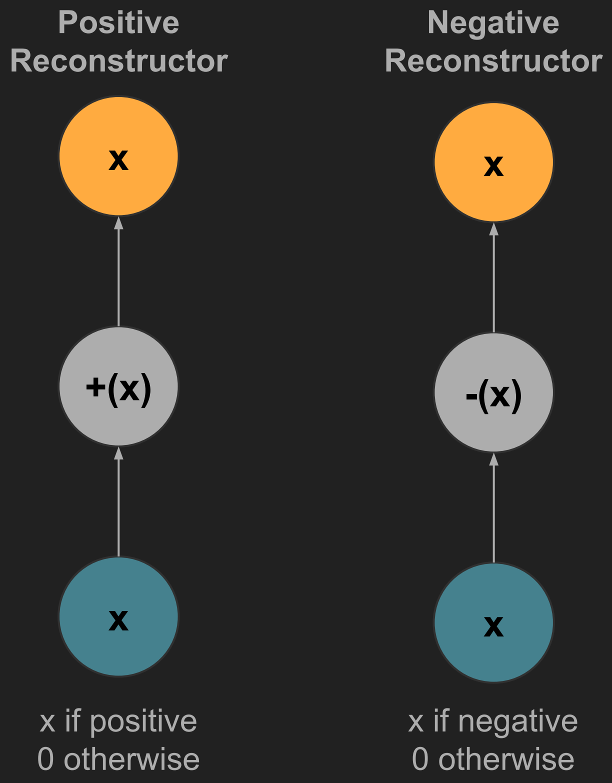


Identity Function: *f(x)* = *x*

Features:

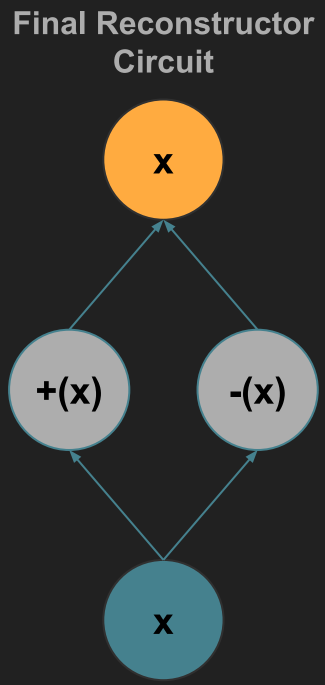


This is actually just ReLU(x). Because the weight = 1, it effectively does nothing.



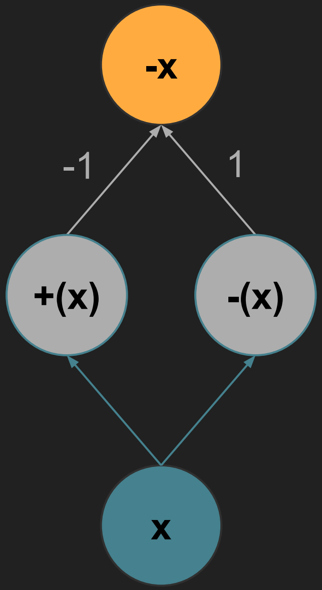
Identity Function: *f(x)* = *x*

Features:



Negative Identity Function: *f(x) = -x*

Simplified:

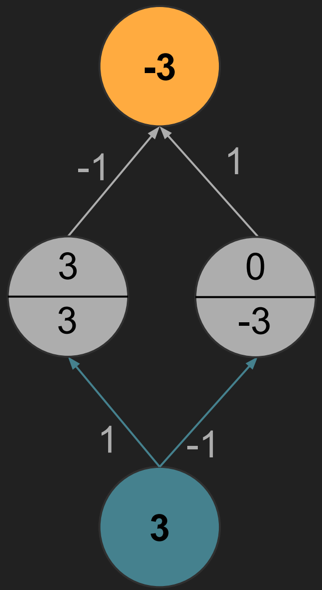


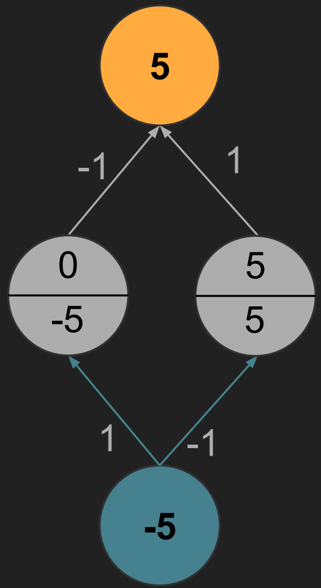
Notes:

- Same as Identity Function / Reconstructor Circuit, just with weights for the output layer reversed

Negative Identity Function: *f(x) = -x*

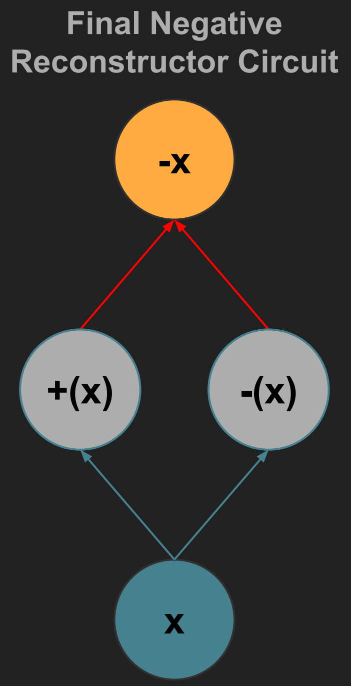
Examples:



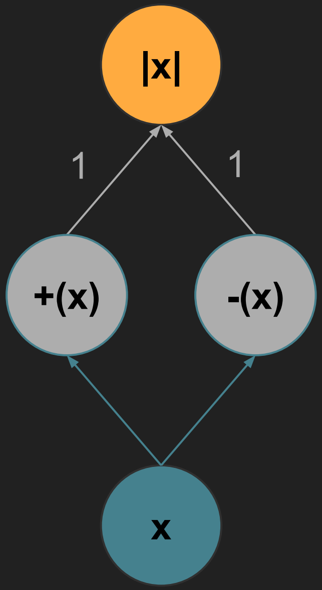


Negative Identity Function: *f(x) = -x*

Features:



Absolute Value: *f(x)* = |*x*|



Notes:

- Same as Identity Function / Reconstructor Circuit, just with 1 for both the output weights

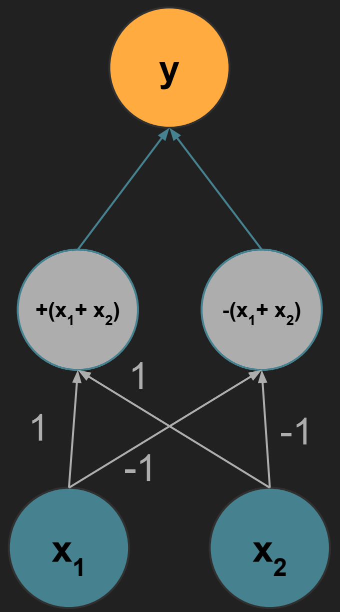
Absolute Value: $f(x) = |x|$

Examples:

Two Number Inputs

Addition: $f(x_1, x_2) = x_1 + x_2$

Simplified:

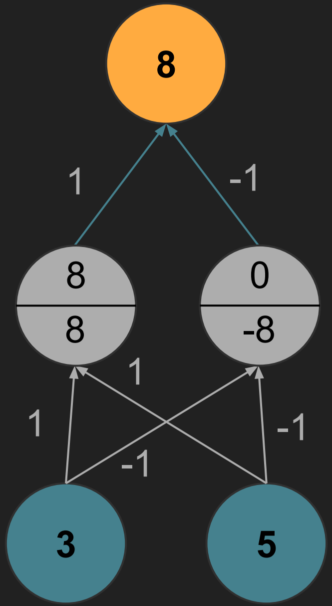


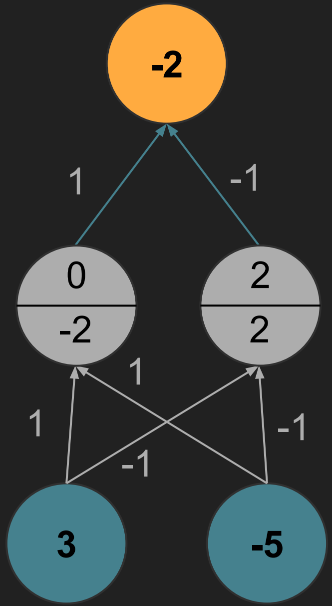
Notes:

- (Assuming we have to use at least one hidden layer)
- All biases are zeroed out
- Hidden layer just creates positiveness and negativeness constructors so that final layer can reconstruct the sum

Addition: $f(x_1, x_2) = x_1 + x_2$

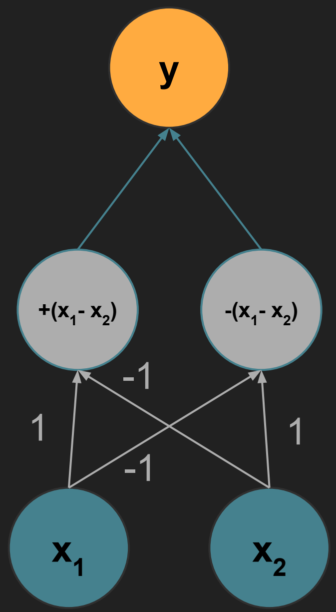
Example:





Subtraction: $f(x_1, x_2) = x_1 - x_2$

Simplified:

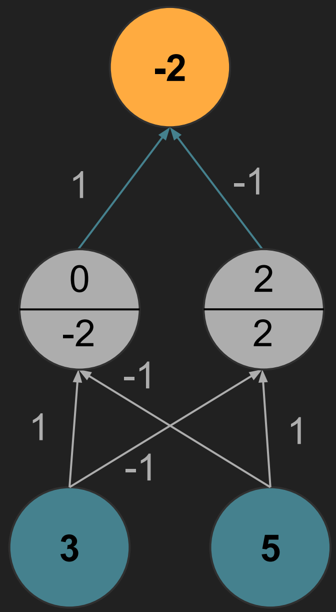


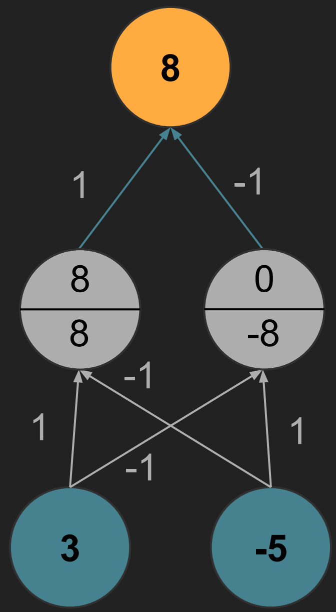
Notes:

- (Assuming we have to use at least one hidden layer)
- All biases are zeroed out
- Hidden layer just creates positiveness and negativeness constructors so that final layer can reconstruct the difference

Subtraction: $f(x_1, x_2) = x_1 - x_2$

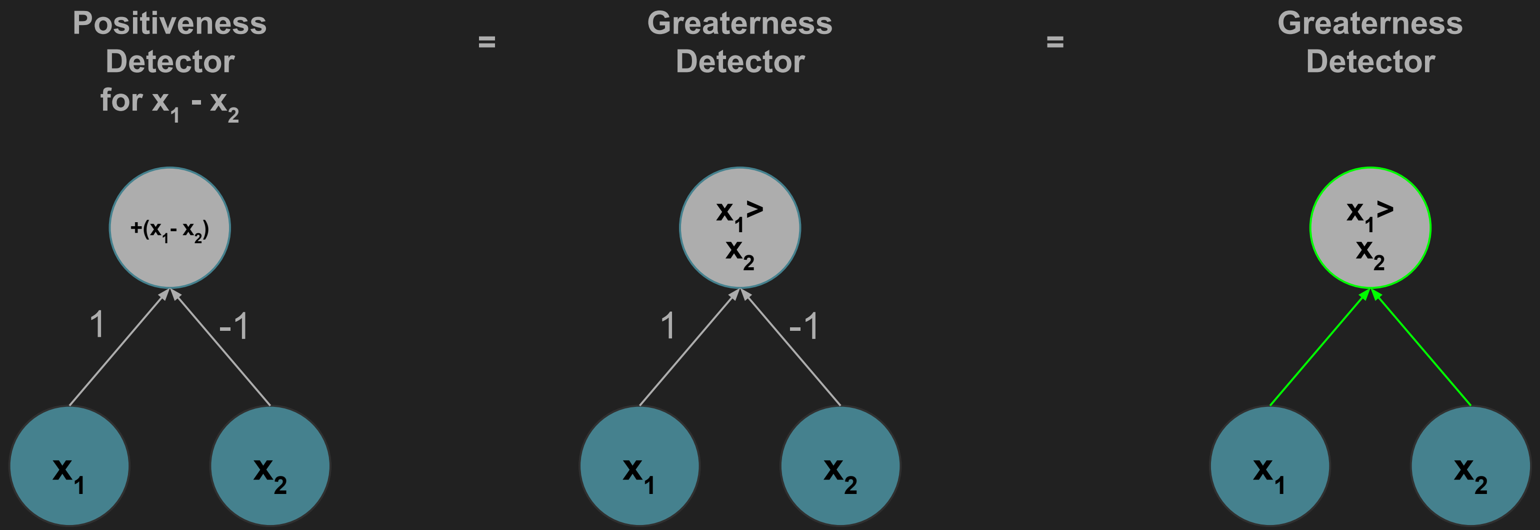
Example:





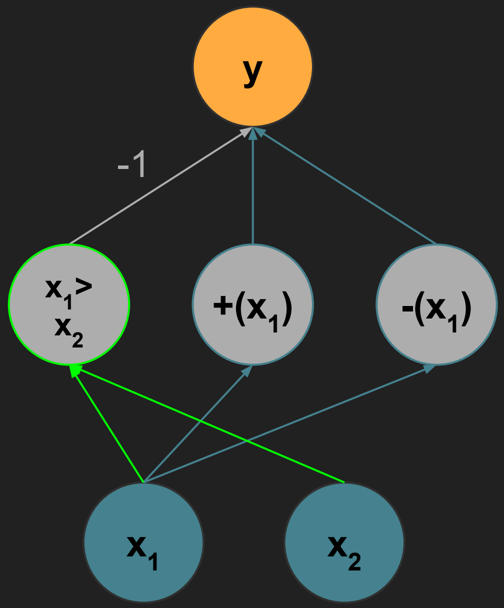
Subtraction: $f(x_1, x_2) = x_1 - x_2$

Features:



Minimum Function: $f(x_1, x_2) = min(x_1, x_2)$

Simplified:



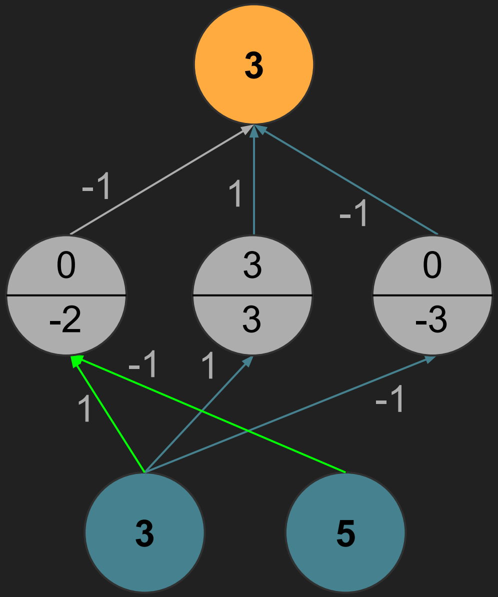
Notes:

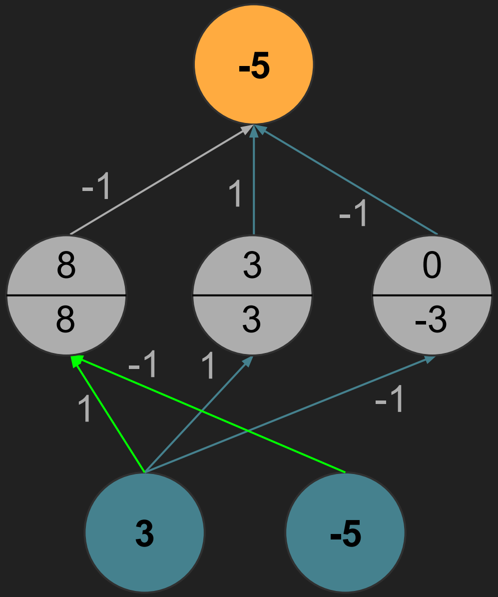
- All biases are zeroed out
- Hidden layers compares x_1 and x2 and passes along x1 (in the form of its positiveness and negativeness scores) for later use
- Output layer subtracts the “greaterness” of x_1 from the reconstructed x_1 to return the minimum

Historical Note: *This was NOT the original “minimum function” network configuration that I derived by hand. While still optimal, it required a slightly larger network to calculate. You can find that original version in the appendix of this presentation.*

Minimum Function: $f(x_1, x_2) = min(x_1, x_2)$

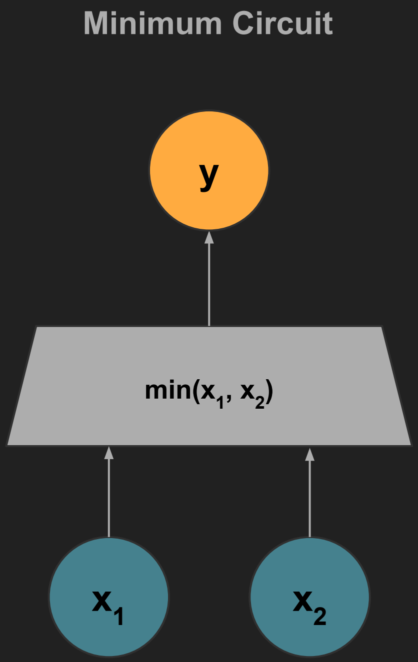
Example:





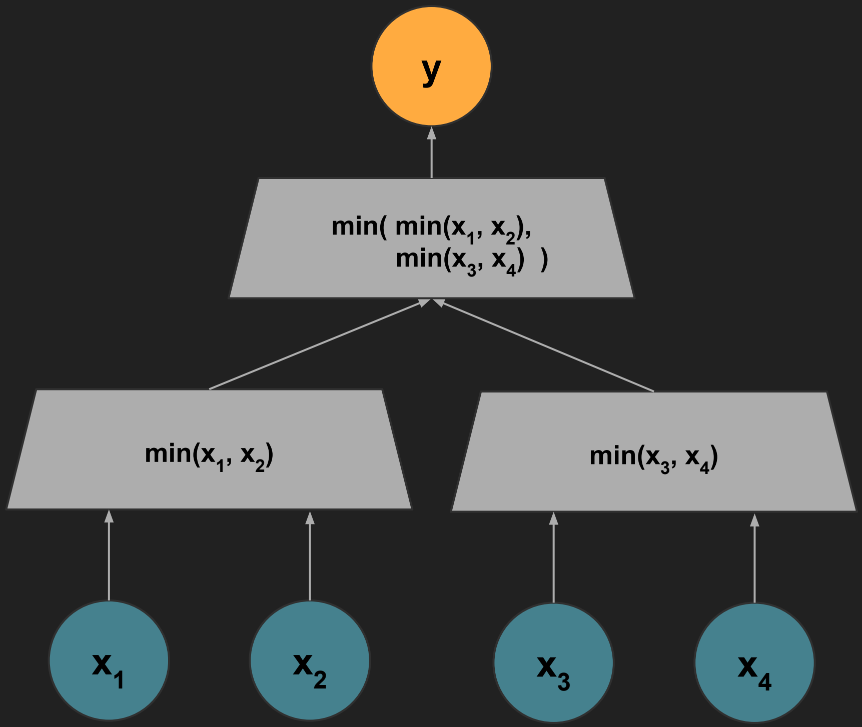
Minimum Function: $f(x_1, x_2) = min(x_1, x_2)$

Circuit:



Minimum: $f(x_1, x_2, x_3, x_4) = min(x_1, x_2, x_3, x_4)$

Circuit:



Experimental Results

I was able to train the optimal networks for *all* of the mathematical functions discussed in this deck

Successfully Validated Solutions:

1. Identity Function: $f(x) = x$

```
loss = 1.6763340390281434e-13

Weights:
--------
hidden_layer_0:
tensor([[-0.9212,  1.0036]], grad_fn=<PermuteBackward0>)

output_layer:
tensor([[-1.0855],
        [ 0.9964]], grad_fn=<PermuteBackward0>)
```

2. Negative Identity Function: $f(x) = -x$

```
loss = 1.8276527481101562e-13

Weights:
--------
hidden_layer_0:
tensor([[ 1.0619, -1.0210]], grad_fn=<PermuteBackward0>)

output_layer:
tensor([[-0.9417],
        [ 0.9794]], grad_fn=<PermuteBackward0>)
```

3. Absolute Value: $f(x) = |x|$

```
loss = 1.828202574136878e-13

Weights:
--------
hidden_layer_0:
tensor([[ 1.0608, -1.0210]], grad_fn=<PermuteBackward0>)

output_layer:
tensor([[0.9427],
        [0.9794]], grad_fn=<PermuteBackward0>)
```

4. Addition: $f(x_1, x_2) = x_1 + x_2$

```
loss = 8.1084677817575e-13

Weights:
--------
hidden_layer_0:
tensor([[ 0.8516, -0.9257],
        [ 0.8516, -0.9257]], grad_fn=<PermuteBackward0>)

output_layer:
tensor([[ 1.1742],
        [-1.0802]], grad_fn=<PermuteBackward0>)
```

5. Subtraction: $f(x_1, x_2) = x_1 - x_2$

```
loss = 6.022274259137594e-12

Weights:
--------
hidden_layer_0:
tensor([[-0.8518,  0.9320],
        [ 0.8518, -0.9320]], grad_fn=<PermuteBackward0>)

output_layer:
tensor([[-1.1740],
        [ 1.0730]], grad_fn=<PermuteBackward0>)
```

6. Minimum: $f(x_1, x_2) = min(x_1, x_2)$

```
loss = 4.303819453599367e-12

Weights:
--------
hidden_layer_0:
tensor([[ 9.5740e-06, -4.7060e-08, -8.5670e-01],
        [ 1.0363e+00, -1.1006e+00,  8.5669e-01]], grad_fn=<PermuteBackward0>)

output_layer:
tensor([[ 0.9650],
        [-0.9086],
        [-1.1673]], grad_fn=<PermuteBackward0>)
```

Code available on GitHub: https://github.com/sosier/Mathematical_Circuits_in_Neural_Nets

Reflections & Next Steps

Some Reflections for AI Safety:

1. Mathematical features / circuits seem quite real

2. However a big obstacle to realizing this level of interpretability may be that models simply never reach sufficient quality for these features to appear

3. To me this suggests, Vanilla Neural Nets alone won't cut it. At a minimum better training methods are needed.

4. Just because a known optimal solution (configuration) exists doesn't mean it's easy (or possible) to find via any specific training regimen.

5. As such, even if we 1) have the perfect, safe goal for an AGI and 2) have theoretical guarantees it could in theory perfectly learn that goal, there is no guarantee it will actually learn the goal in practice

6. To ensure alignment, we also need theoretical guarantees that a given training procedure will produce the desired (optimal) solution

Possible Next Steps:

1. Expand the range of mathematical functions studied

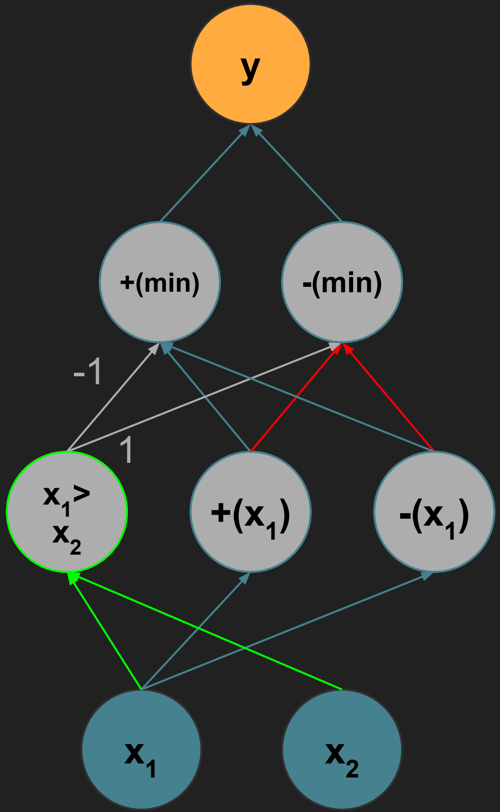
2. Develop methodology for detecting these features / circuits in trained models

3. Identify training regimens that more reliably produce optimal, interpretable models (e.g. what optimizers, intializations, regularization, etc.)

Appendix

Minimum Function: $f(x_1, x_2) = min(x_1, x_2)$

Simplified:



Notes:

- All biases are zeroed out
- First hidden compares x_1 and x2 and passes along x1 (in the form of its positiveness and negativeness scores) for later use
- The second hidden layer calculates the minimum's positiveness and negativeness
- Output layer constructs the minimum from it's positiveness and negativeness scores

***Historical Note:** This was the original "minimum function" network configuration that I derived by hand. While optimal, I discovered during experimental testing that this network is NOT the smallest optimal network configuration. Specifically, the last hidden layer and output layer can be combined producing a smaller network.*

Minimum Function: $f(x_1, x_2) = min(x_1, x_2)$

Example: