

Basic Neural Network Circuits

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

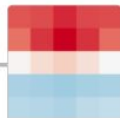
Analogous features and circuits form across models and tasks.

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

They demonstrate these claims in the context of image models

1. Features:

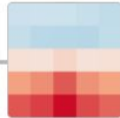
Windows (4b:237)
excite the car detector
at the top and inhibit
at the bottom.



Car Body (4b:491)
excites the car
detector, especially at
the bottom.



Wheels (4b:373) excite
the car detector at the
bottom and inhibit at
the top.



2. Circuits:

● positive (excitation)
● negative (inhibition)



A car detector (4c:447)
is assembled from
earlier units.

3. Universality:

Curve detectors

ALEXNET

Krizhevsky et al. [34]



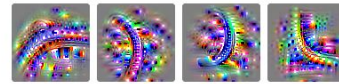
INCEPTIONV1

Szegedy et al. [26]



VGG19

Simonyan et al. [35]

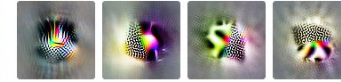
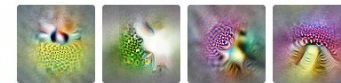
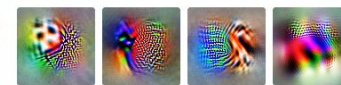


RESNETV2-50

He et al. [36]



High-Low Frequency detectors



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



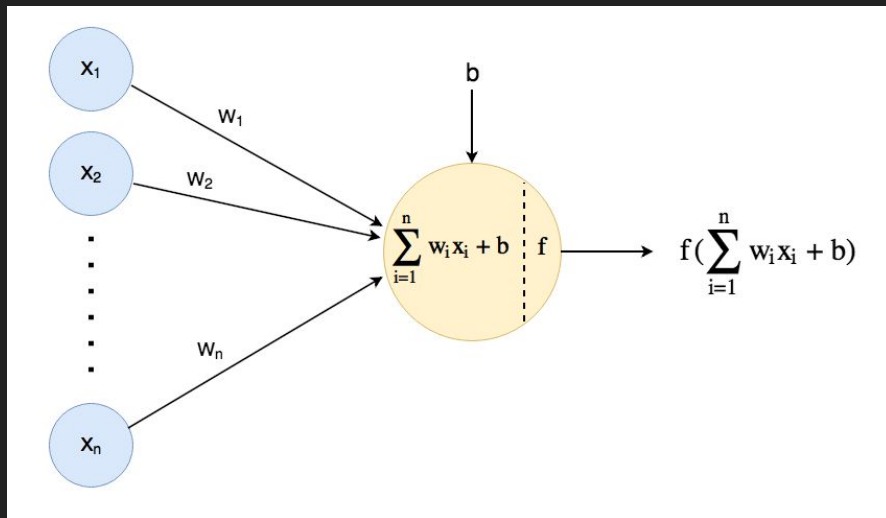
A chalkboard with mathematical definitions written in white chalk. On the left, the absolute value function is defined as $|x| = \begin{cases} x & x \geq 0 \\ -x & x < 0 \end{cases}$. On the right, the two conditional cases are written separately: $x \geq 0$ and $x < 0$. A hand holding a piece of chalk is visible at the bottom right, near the $x < 0$ expression.

$$|x| = \begin{cases} x & x \geq 0 \\ -x & x < 0 \end{cases}$$
$$x \geq 0$$
$$x < 0$$

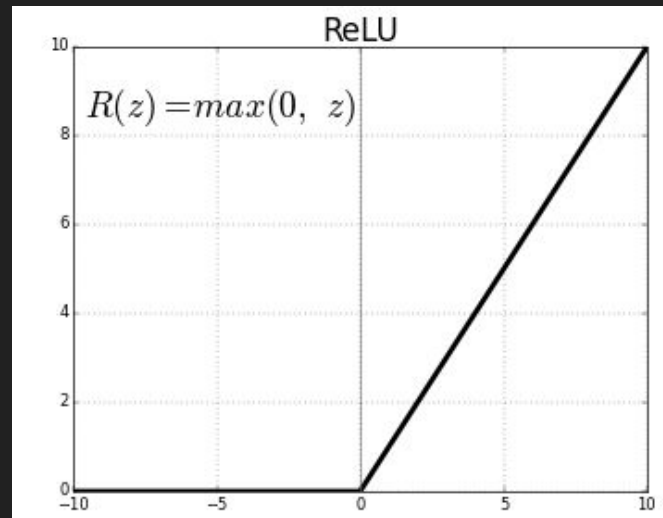
Background

Basic Neural Network Math

How a Single Neuron Works:



The ReLU Activation Function:



Legend

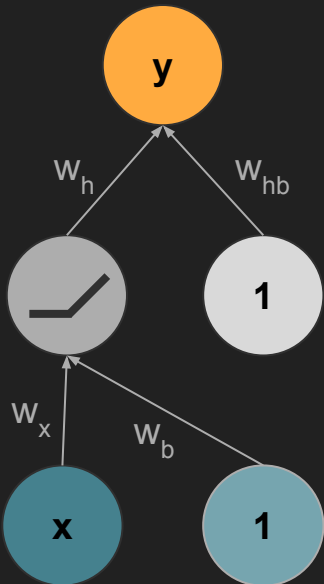
Output Layer:



Hidden Layer:



Input Layer:



= The input (a number)



= Bias node for the input layer



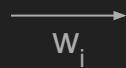
= Hidden layer node with relu



= Bias node for the hidden layer



= The output (a number)

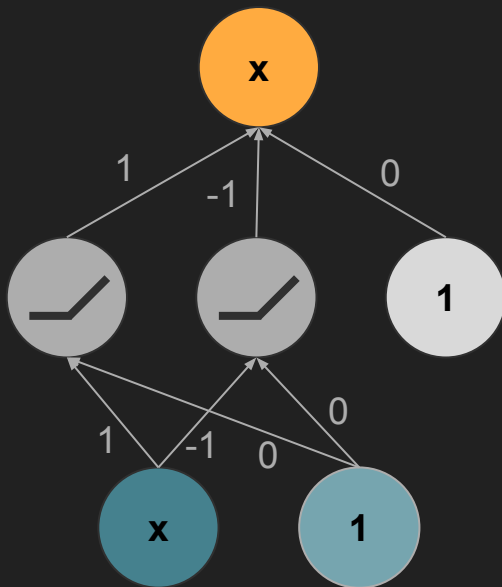


= A connection with its weight

Single Number Inputs

Identity Function: $f(x) = x$

Optimal Solution:

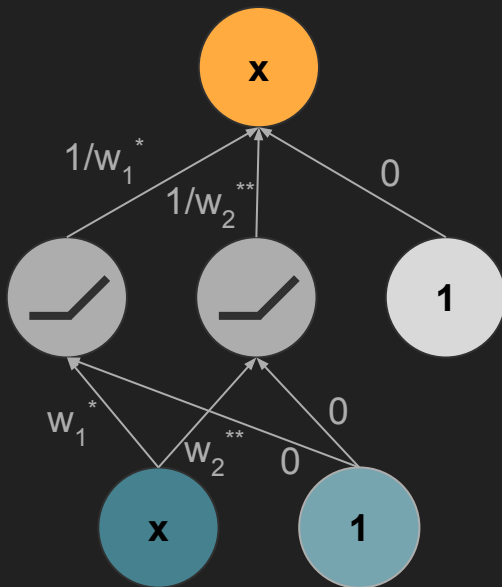


Notes:

- 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 its positiveness and negativeness scores

Identity Function: $f(x) = x$

More Precisely:



* where w_1 is positive

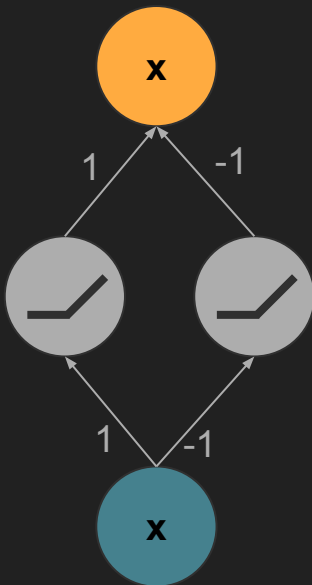
** where w_2 is negative

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 its positiveness and negativeness scores
- w and $1/w$ ensure the final result is neither scaled up or down, while allowing for infinitely many optimal solutions

Identity Function: $f(x) = x$

Simplified:

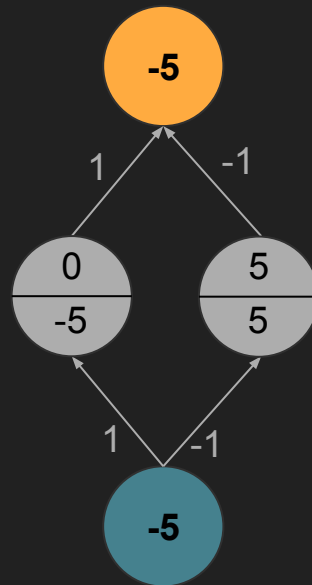
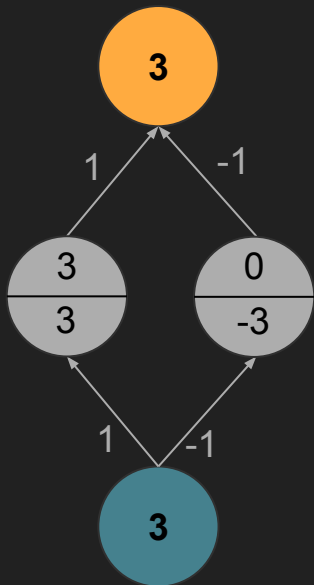


Notes:

- 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 its positiveness and negativeness scores

Identity Function: $f(x) = x$

Examples:



Identity Function: $f(x) = x$

Features:

Positiveness
Detector



x if positive
 0 otherwise

Negativeness
Detector



$|x|$ if negative
 0 otherwise

This is actually just $\text{ReLU}(x)$.
Because the weight = 1, it
effectively does nothing.

Positive
Reconstructor



x if positive
 0 otherwise

Negative
Reconstructor

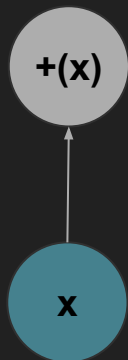


x if negative
 0 otherwise

Identity Function: $f(x) = x$

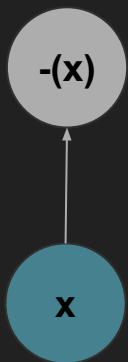
Features:

Positiveness
Detector



x if positive
0 otherwise

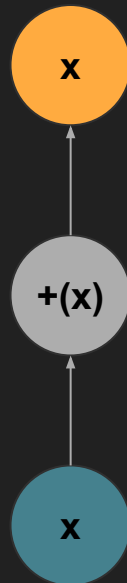
Negativeness
Detector



|x| if negative
0 otherwise

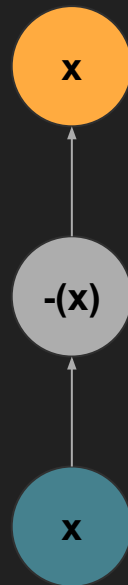
This is actually just $\text{ReLU}(x)$.
Because the weight = 1, it
effectively does nothing.

Positive
Reconstructor



x if positive
0 otherwise

Negative
Reconstructor

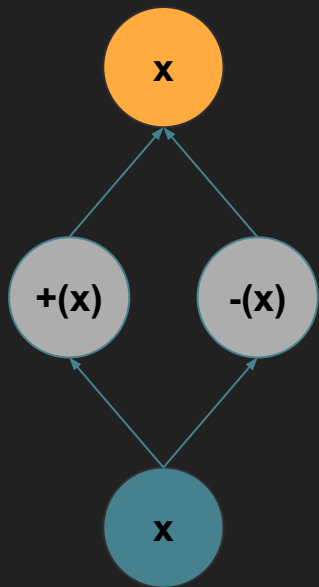


x if negative
0 otherwise

Identity Function: $f(x) = x$

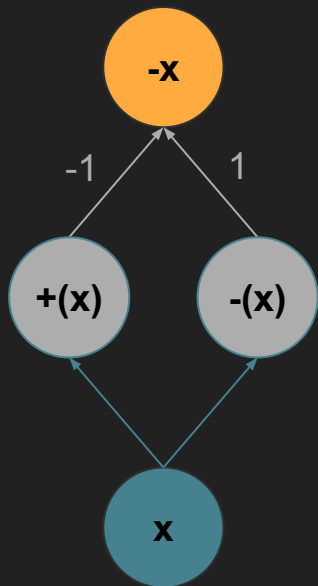
Features:

Final Reconstructor
Circuit



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

Simplified:

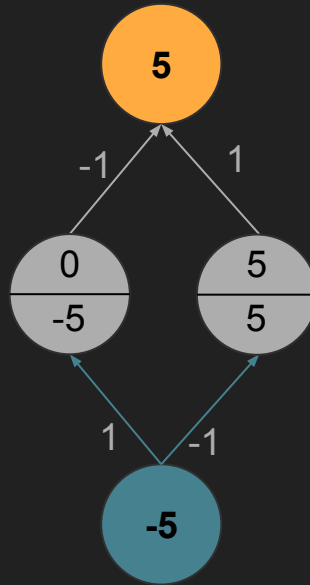
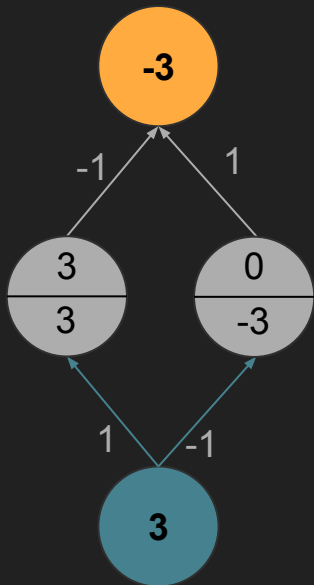


Notes:

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

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

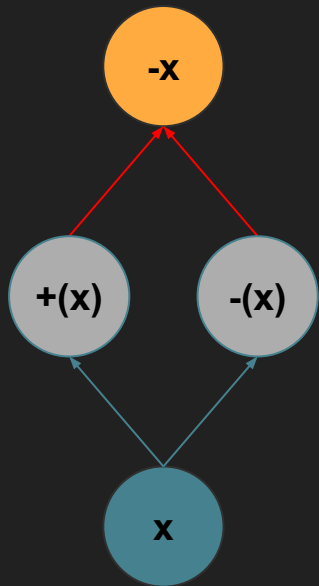
Examples:



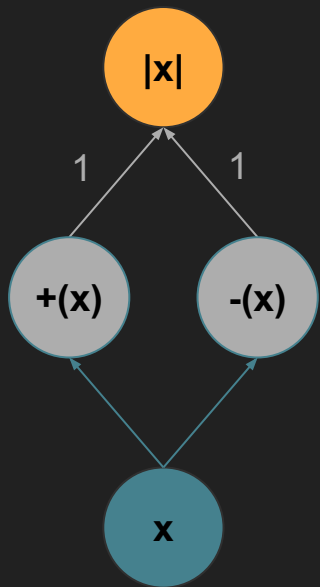
Negative Identity Function: $f(x) = -x$

Features:

Final Negative
Reconstructor Circuit



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

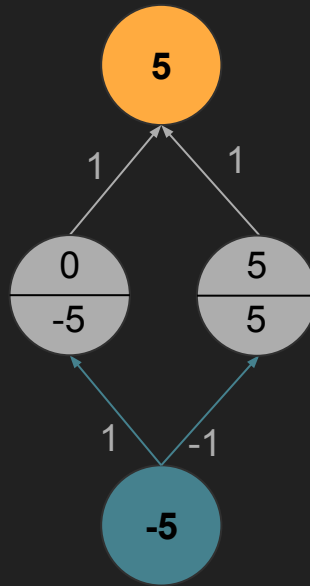
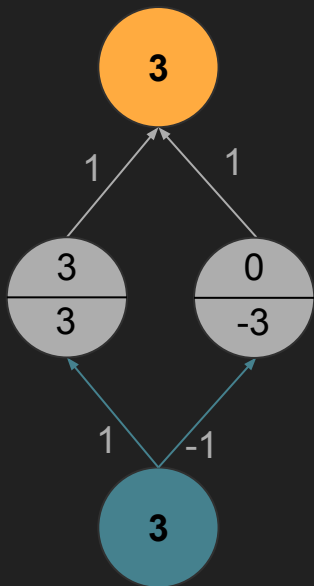


Notes:

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

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

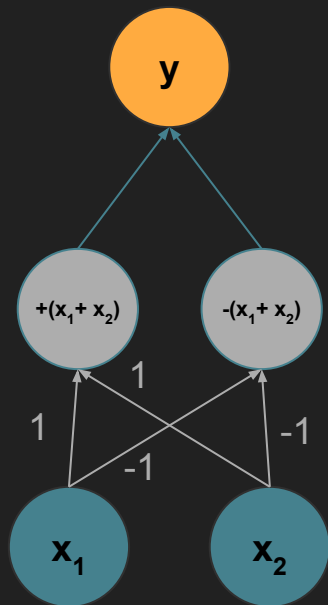
Examples:



Two Number Inputs

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

Simplified:

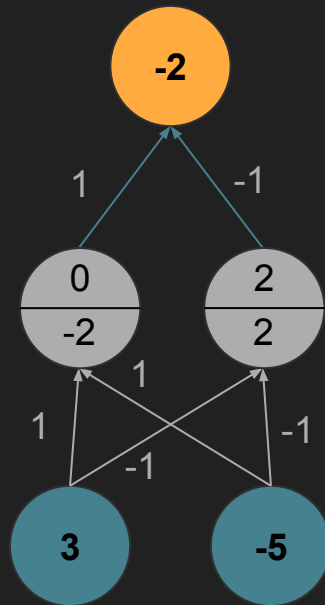
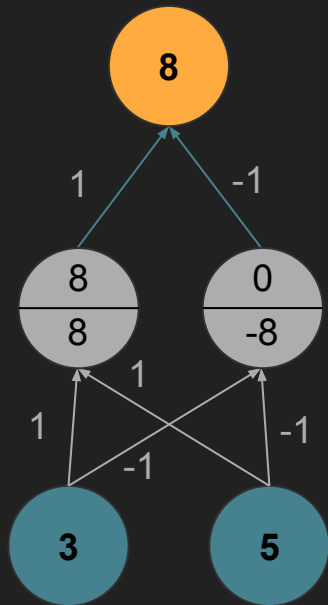


Notes:

- (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

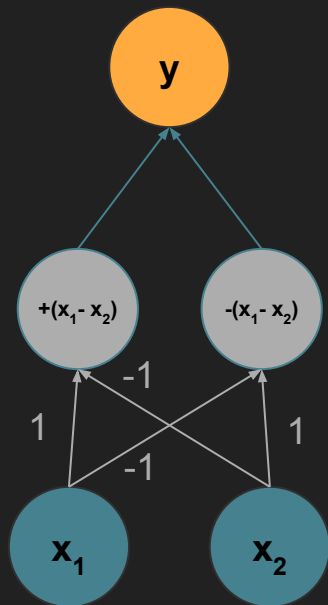
Addition: $f(x_1, x_2) = x_1 + x_2$

Example:



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

Simplified:

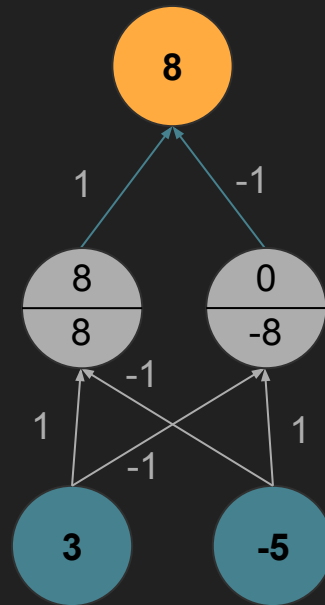
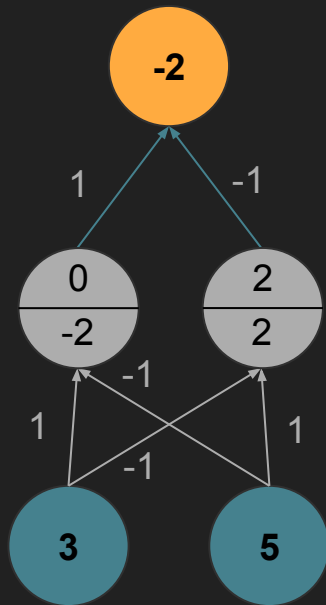


Notes:

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

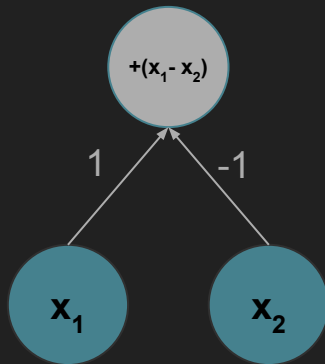
Example:



Subtraction: $f(\mathbf{x}_1, \mathbf{x}_2) = \mathbf{x}_1 - \mathbf{x}_2$

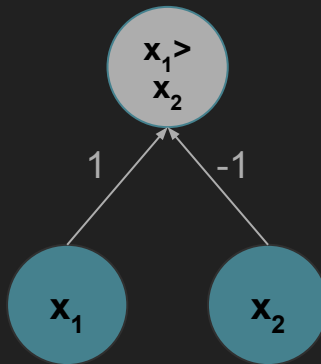
Features:

Positiveness
Detector
for $\mathbf{x}_1 - \mathbf{x}_2$



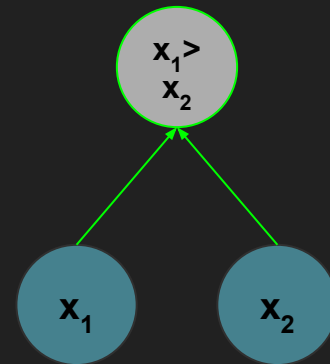
=

Greaterness
Detector



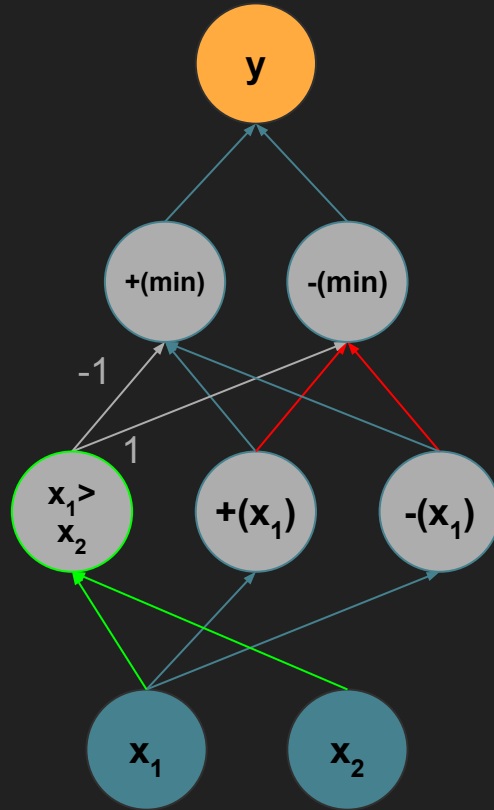
=

Greaterness
Detector



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

Simplified:

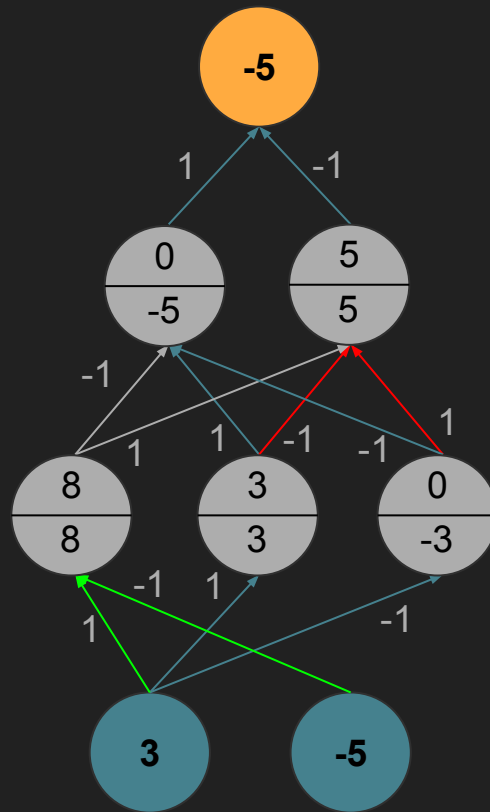
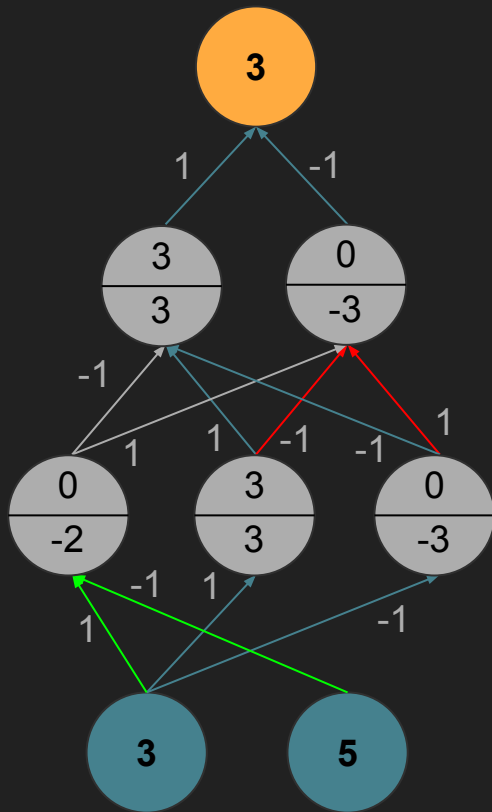


Notes:

- All biases are zeroed out
- First hidden compares x_1 and x_2 and passes along x_1 (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

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

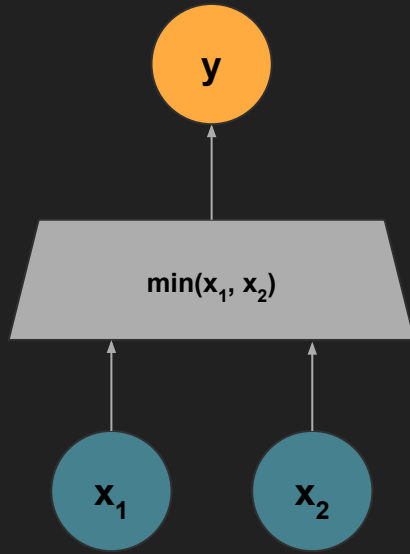
Example:



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

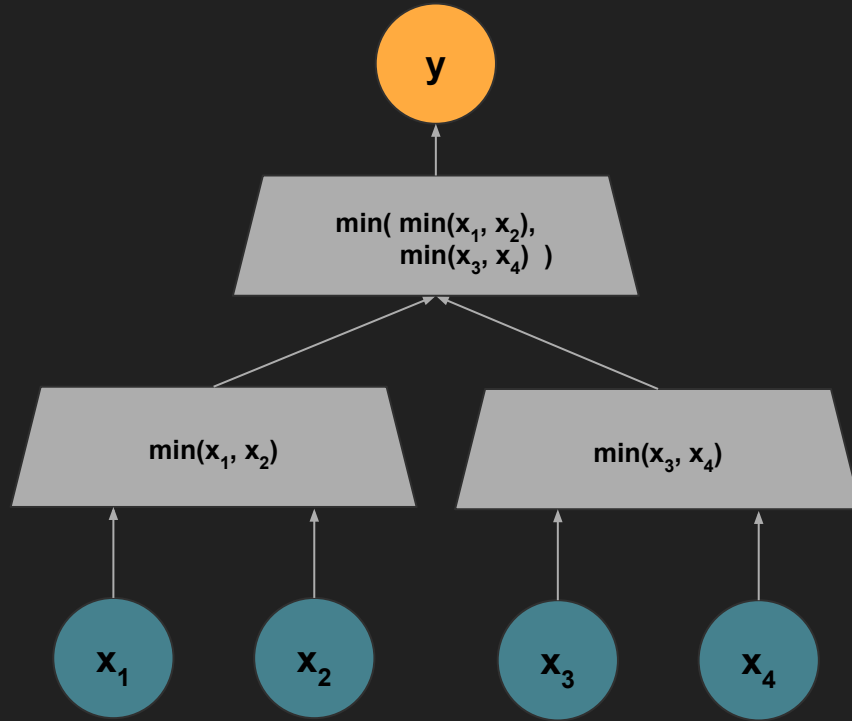
Circuit:

Minimum 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 most of the mathematical functions discussed in this deck

Successfully Validated Solutions:

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

```
loss = 1.9709477783180773e-07

Weights:
-----
hidden_layer_0:
tensor([[ -0.9192,  1.0035]], grad_fn=<PermuteBackward0>)

output_layer:
tensor([[ -1.0872],
        [ 0.9957]], grad_fn=<PermuteBackward0>)
```

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

```
loss = 1.849424222655216e-07

Weights:
-----
hidden_layer_0:
tensor([[ 1.0627, -1.0206]], grad_fn=<PermuteBackward0>)

output_layer:
tensor([[ -0.9403],
        [ 0.9790]], grad_fn=<PermuteBackward0>)
```

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

```
loss = 1.8525956591020076e-07

Weights:
-----
hidden_layer_0:
tensor([[ 1.0627, -1.0206]], grad_fn=<PermuteBackward0>)

output_layer:
tensor([[0.9403],
        [0.9790]], grad_fn=<PermuteBackward0>)
```

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

```
loss = 3.158412766879337e-07

Weights:
-----
hidden_layer_0:
tensor([[ 0.8511, -0.9304],
        [ 0.8522, -0.9291]], grad_fn=<PermuteBackward0>)

output_layer:
tensor([[ 1.1742],
        [-1.0755]], grad_fn=<PermuteBackward0>)
```

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

```
loss = 3.452058763286914e-07

Weights:
-----
hidden_layer_0:
tensor([[ -0.8519,  0.9296],
        [ 0.8508, -0.9309]], grad_fn=<PermuteBackward0>)

output_layer:
tensor([[ -1.1746],
        [ 1.0750]], grad_fn=<PermuteBackward0>)
```

Notably for the Minimum network / circuit, the model seemed to consistently get stuck in a local optimum

Even given the perfect architecture, a simple goal, and unlimited perfect training data...

...the models converge to and get stuck in a local optima that isn't even that optimal:

Conclusion & Next Steps

Some Conclusions 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

Next Steps for this Work:

1. Figure out how to learn the optimal “Minimum Function” network
 - a. Experiment with other optimizers?
 - b. Experiment with alternate network initialization strategies?
 - c. Experiment with ways of getting “unstuck” once it appears a local optima has been found, but loss is still > 0 ?
2. Expand the range of mathematical functions studied
3. Develop methodology for detecting these features / circuits in trained models