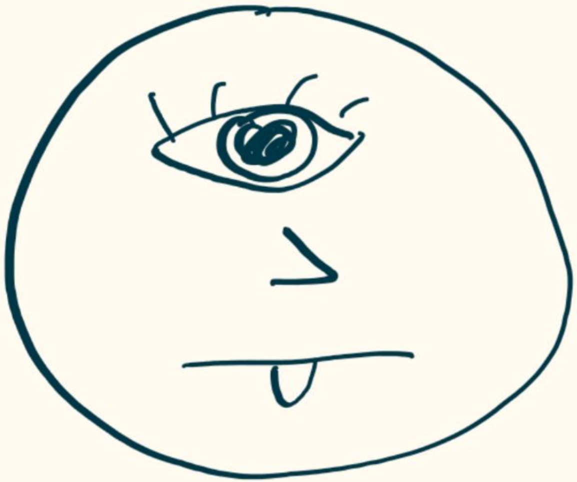
Superposition in neural networks II

—

by N. Elhage et al.

Toy problem

Let us have feature vector $X = [x_1, x_2, x_3, x_4]$, representing a face:

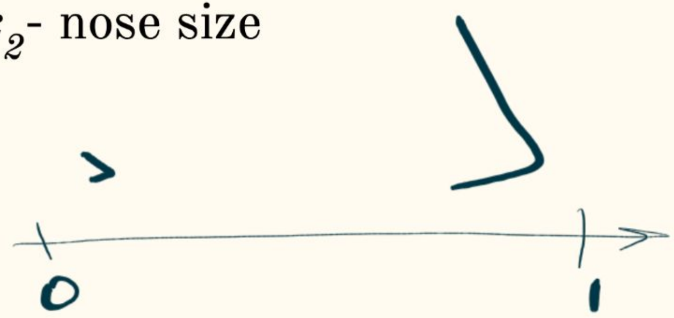


Toy problem

Let us have feature vector $X = [x_1, x_2, x_3, x_4]$, representing a face:

x_1 - eye size

x_2- nose size



x_3- lips size

x_4- head size

Toy problem

We want to project the 4D face $\boldsymbol{X}$ into a 2D representation $\boldsymbol{H} = [\boldsymbol{h_1}, \boldsymbol{h_2}]$, and then reconstruct the original 4D face from the 2D representation.

Let us also assume that reconstructing the eye and nose features correctly is more important than reconstructing other features.

$$L = \sum_x \sum_i I_i (x_i - x'_i)^2$$

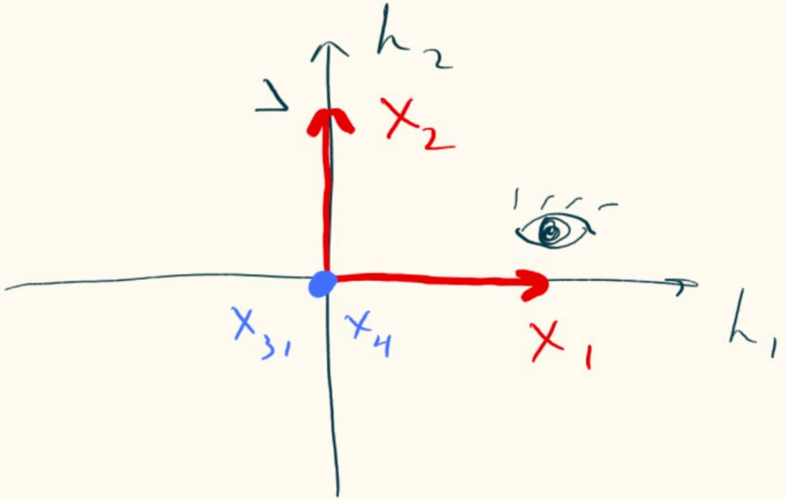
Our **loss function** looks like this, where $\boldsymbol{I_i}$ is the importance of feature $\boldsymbol{i}$. Let $\boldsymbol{I_1}$ and $\boldsymbol{I_2} = 1$, $\boldsymbol{I_3}$ and $\boldsymbol{I_4} = 0.5$.

Toy problem

One of the

What $\boldsymbol{H = [h_1, h_2]}$ might look like in this case?

Since eye and nose features are more important, a straightforward solution that minimizes the loss function is $\boldsymbol{H = [x_1, x_2]}$



Why not encode lips and head features too?

- We don't have enough dimensions for them.
- They are less important.
- Encoding them would interfere the reconstruction, harming the more important features.

Toy problem

But what if the features are sparse?

X_1

X_2

X_3

X_4

X_5

X_6

X_7

X_8

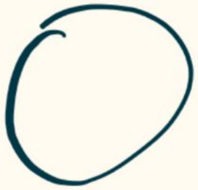
Toy problem

...or even more sparse?

X_1

X_2

X_3



X_4

X_5

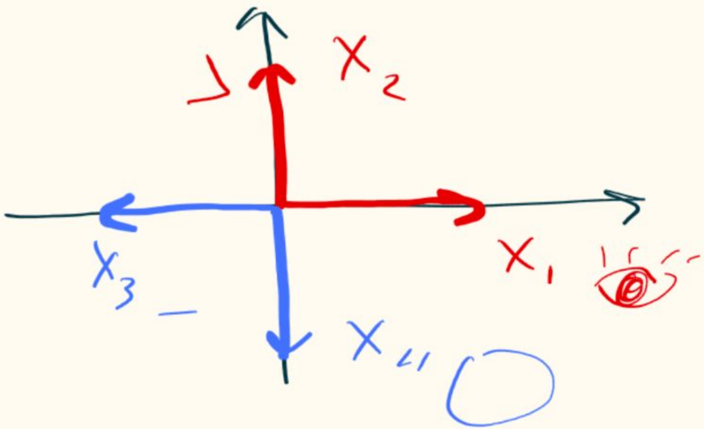
X_6

X_7

X_8

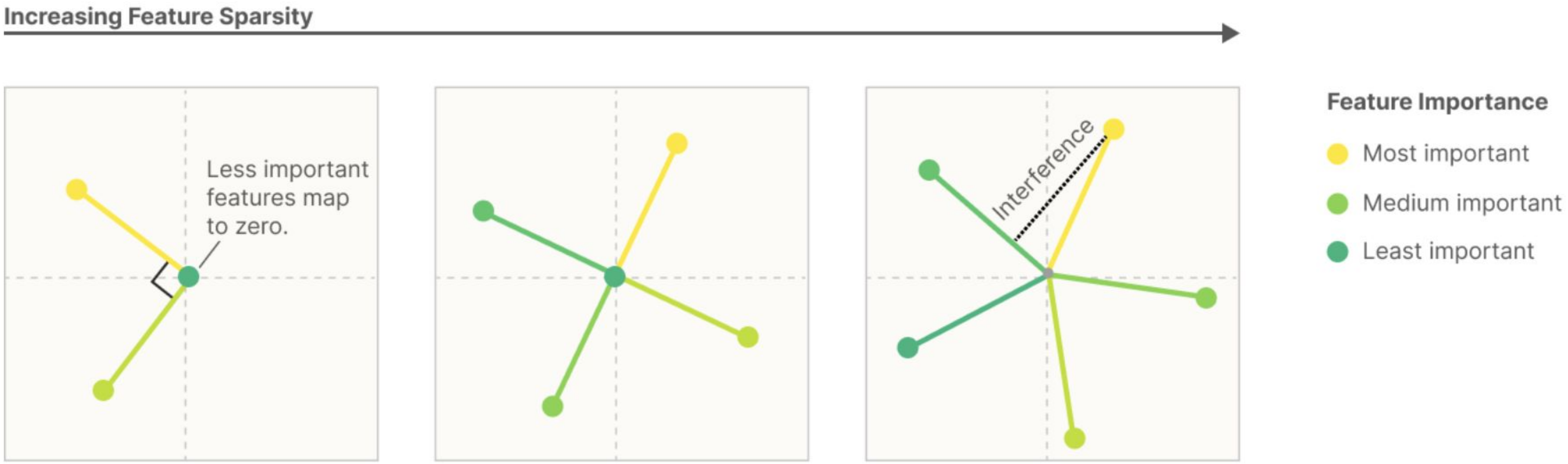
Toy problem

We might assume that a good projection into 2D space will encode not only eye and nose features, but others too. This way, the loss function can be better minimized.
$\boldsymbol{H = [h_1, h_2]}$ in this case can look like this:



But, we need non-linearity for this to work.

As Sparsity Increases, Models Use "Superposition" To Represent More Features Than Dimensions



0% Sparsity

The two most important features are given **dedicated orthogonal dimensions**, while other features are **not embedded**.

80% Sparsity

The four most important features are represented as **antipodal pairs**. The least important features are **not embedded**.

90% Sparsity

All five features are embedded **as a pentagon**, but there is now "positive interference."

Can we quantify superposition?

Let's give it a try. Let us have

- Eye $X_1 = [1, 0, 0, 0]^T$
- Nose $X_2 = [0, 1, 0, 0]^T$
- Lips $X_3 = [0, 0, 1, 0]^T$
- Head $X_4 = [0, 0, 0, 1]^T$

Their 2D representations:

- $H_1 = [1, 0]^T$
- $H_2 = [0, 1]^T$
- $H_3 = [-1, 0]^T$
- $H_4 = [0, -1]^T$

And a matrix that transforms X to H: $W = \begin{pmatrix} 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & -1 \end{pmatrix}$

Can we quantify superposition?

An inverse operation that transforms H to X is

$$W^{-1} = \mathrm{ReLU}[W^{T}(\cdot)] = \mathrm{ReLU}\begin{pmatrix} 1 & 0 \\ 0 & 1 \\ -1 & 0 \\ 0 & -1 \end{pmatrix}(\cdot)$$

To quantify superposition, we can

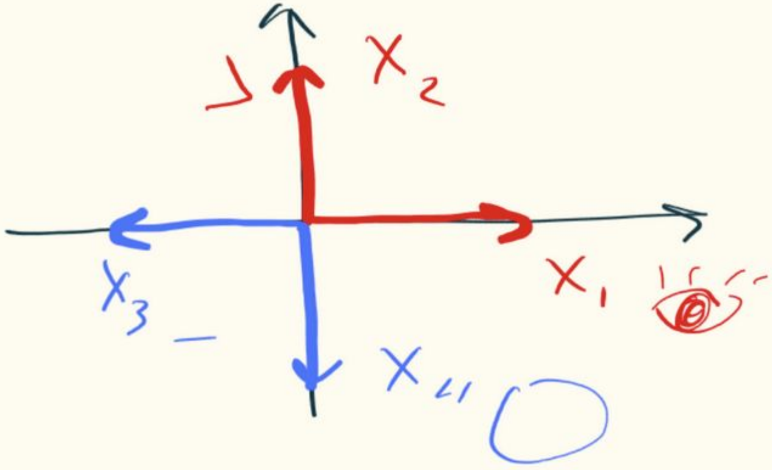
1. Calculate and visualize $W^{T}W$
2. Calculate superposition measure of i^{th} feature representation: $\sum_{j\neq i}(\hat{W}_i \cdot W_j)^2$
3. Calculate dimensions per original feature i: $D_i = \frac{||W_i||^2}{\sum_j(\hat{W}_i \cdot W_j)^2}$
   where W_i is i^{th} column of W

Can we quantify superposition?

$W^T W$

Superposition measure of i th projection

Dimensions per original feature i



Space for scribbles

Outline

Superposition can occur when n features are squeezed into $m < n$ dimensions.

- How superposition can look like
- How superposition is handled during computations
- Is superposition good/bad?

1. How superposition can look like

Another toy problem:

Project a high dimensional vector $x \in R^n$ into a lower dimensional vector $h \in R^m$ and then reconstruct it.

- features x_i are 0 with probability S (sparsity), or uniformly distributed on [0, 1] otherwise.
- features x_i have importance I_i

$$L = \sum_x \sum_i I_i (x_i - x'_i)^2$$

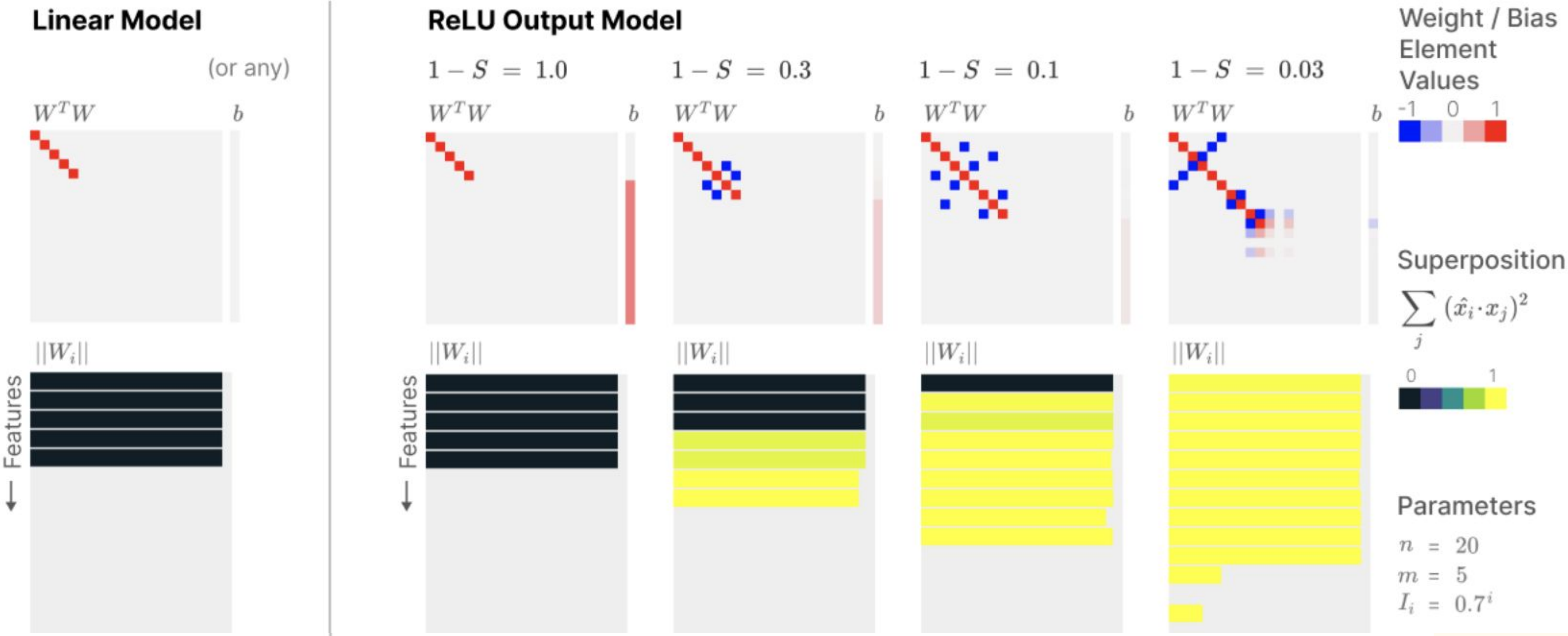
A familiar loss function

1. Models

Linear Model	**ReLU Output Model**
$h = Wx$	$h = Wx$
$x' = W^T h + b$	$x' = \mathrm{ReLU}(W^T h + b)$
$x' = W^T W x + b$	$x' = \mathrm{ReLU}(W^T W x + b)$

Linear model can't have superposition

But slightly nonlinear model can have it



Linear models learn the top m features. $1 - S$ = 0.001 is shown, but others are similar.

In the **dense** regime, ReLU output models also learn the top m features.

As **sparsity increases**, superposition allows models to represent more features. The most important features are initially untouched. This early superposition is organized in antipodal pairs (more on this later).

1. As to why superposition occurs only with nonlinear models

Rewritten linear loss:

$$L \sim \sum_i I_i(1 - ||W_i||^2)^2 \quad + \quad \sum_{i \neq j} I_j(W_j \cdot W_i)^2$$

Feature benefit is the value a model attains from representing a feature. In a real neural network, this would be analagous to the potential of a feature to improve predictions if represented accurately.

Interference betwen x_i and x_j occurs when two features are embedded non-orthogonally and, as a result, affect each other's predictions. This prevents superposition in linear models.

1. As to why superposition occurs only with nonlinear models

Rewritten **non-**linear loss:

$$L_1 = \sum_i \int_{0 \le x_i \le 1} I_i(x_i - \mathrm{ReLU}(||W_i||^2 x_i + b_i))^2 \quad + \sum_{i \ne j} \int_{0 \le x_i \le 1} I_j \mathrm{ReLU}(W_j \cdot W_i x_i + b_j)^2$$

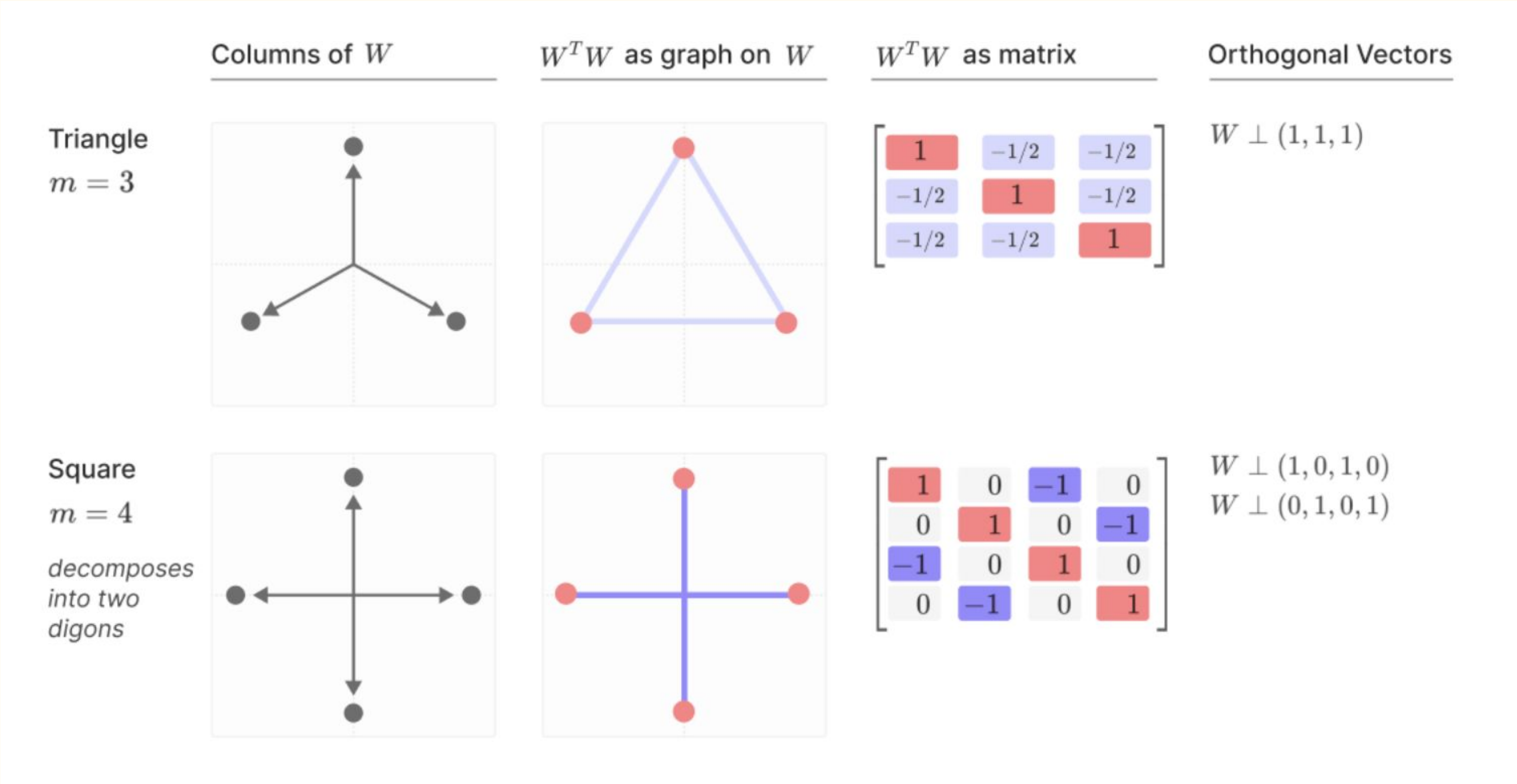
If we focus on the case $x_i = 1$ *, we get something which looks even more analagous to the linear case:*

$$= \sum_i I_i(1 - \mathrm{ReLU}(||W_i||^2 + b_i))^2 \quad + \sum_{i \ne j} I_j \mathrm{ReLU}(W_j \cdot W_i + b_j)^2$$

Feature benefit is similar to before. Note that ReLU never makes things worse, and that the bias can help when the model doesn't represent a feature by taking on the expected value.

Interference is similar to before but ReLU means that negative interference, or interference where a negative bias pushes it below zero, is "free" in the 1-sparse case.

1. How the superposition states can look like?

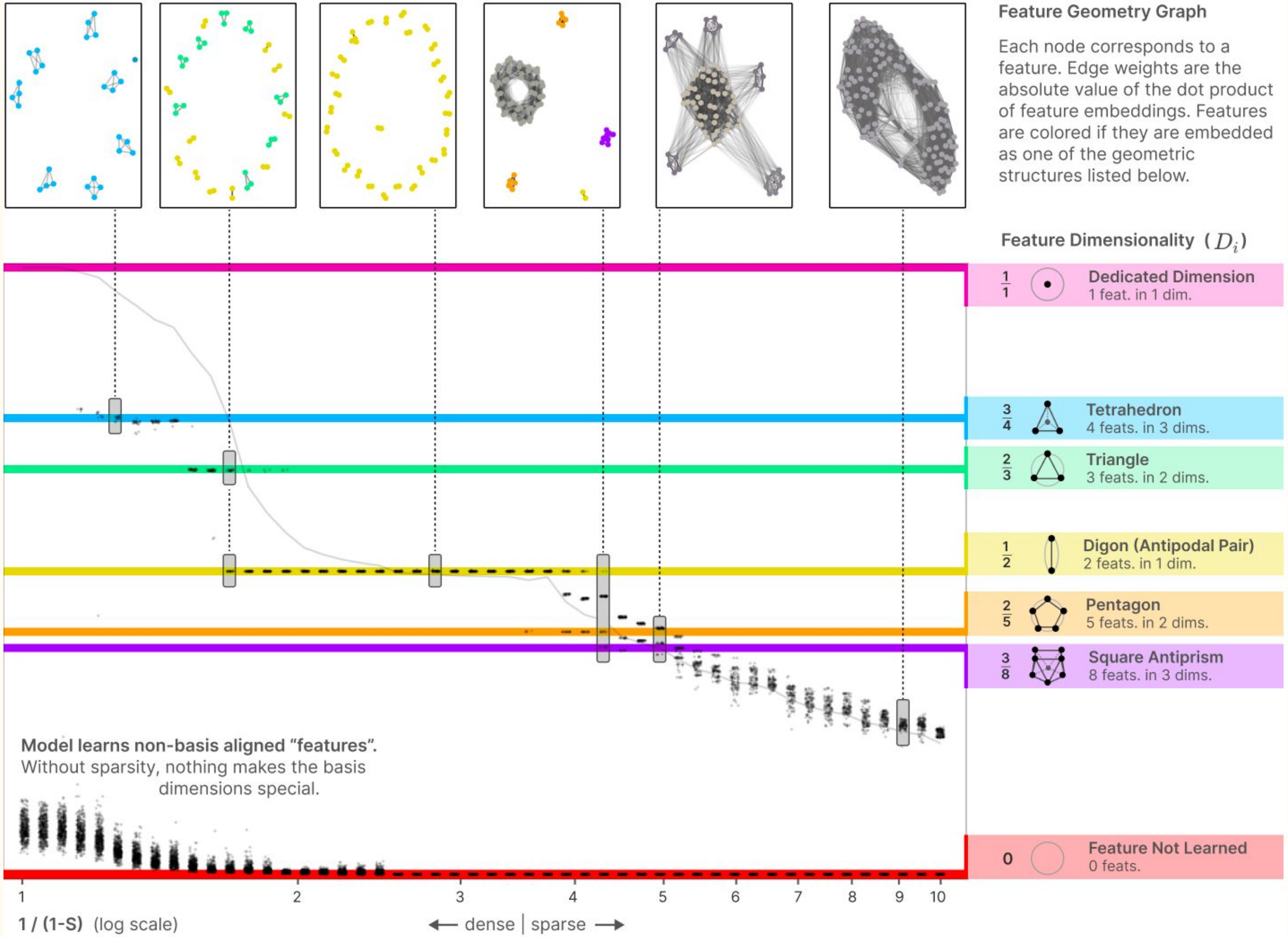


$$\begin{bmatrix} 1 & -1/2 & -1/2 \\ -1/2 & 1 & -1/2 \\ -1/2 & -1/2 & 1 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & -1 \\ -1 & 0 & 1 & 0 \\ 0 & -1 & 0 & 1 \end{bmatrix}$$

For:

- $n = 400$
- $m = 30$
- $I = 1$



Space for scribbles

2. How superposition is handled during computations

So far the shown examples were about autoencoder-like problems, where hidden state utilizes superposition for data storage.

What if a hidden state in superposition is used an MLP-like problem? How will the weights look like, and how can we analyze them?

2. Yet another toy problem

Learning abs() function:

Given vector $x \in R^n$, project it into a hidden state vector $h \in R^m$ and then reconstruct x' = abs(x) from it.

- features x_i are 0 with probability S (sparsity), or uniformly distributed on [-1, 1] otherwise.
- features x_i have importance I_i

$$L = \sum_x \sum_i I_i (x_i - x_i')^2$$

A familiar loss function

2. Model

Now W_1 and W_2 (instead of a single W before) are learnable matrices.

There exists a simple non-superpositional solution with $m = 2n$ hidden neurons:
abs(x)=ReLU(x)+ReLU(-x)

$$h = \mathrm{ReLU}(W_1 x)$$

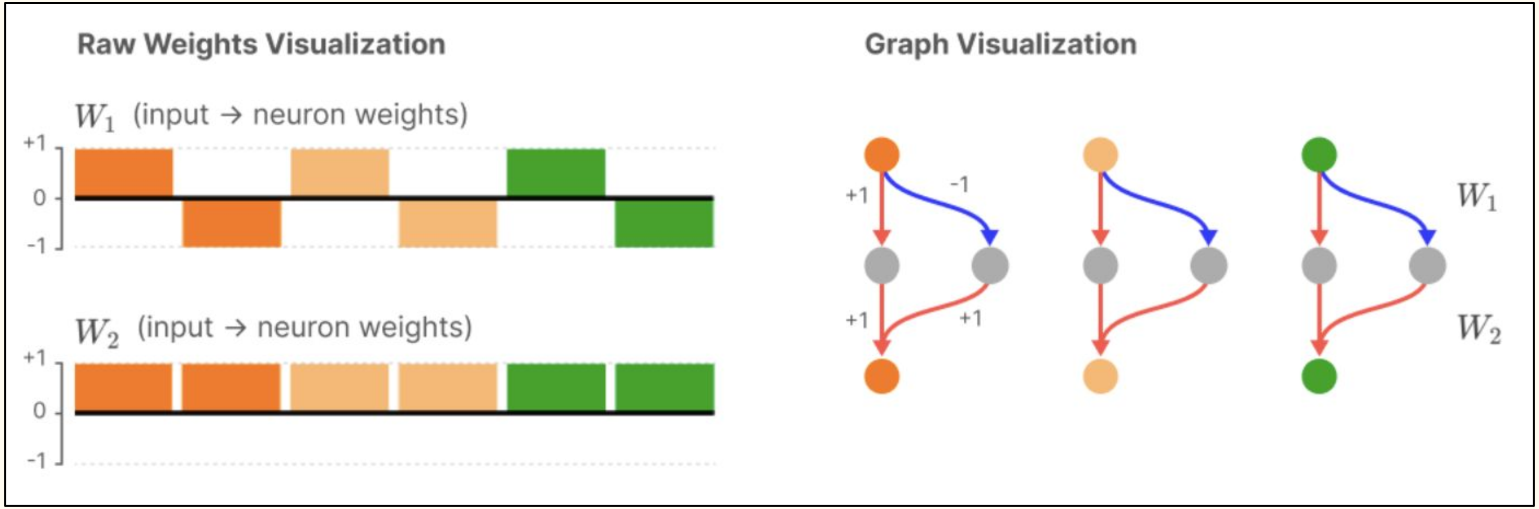
$$y' = \mathrm{ReLU}(W_2 h + b)$$

2. Simple Model

There exists a simple non-superpositional solution with $m = 2n$ hidden neurons: abs(x)=ReLU(x)+ReLU(-x)

$$h = \mathrm{ReLU}(W_1 x)$$

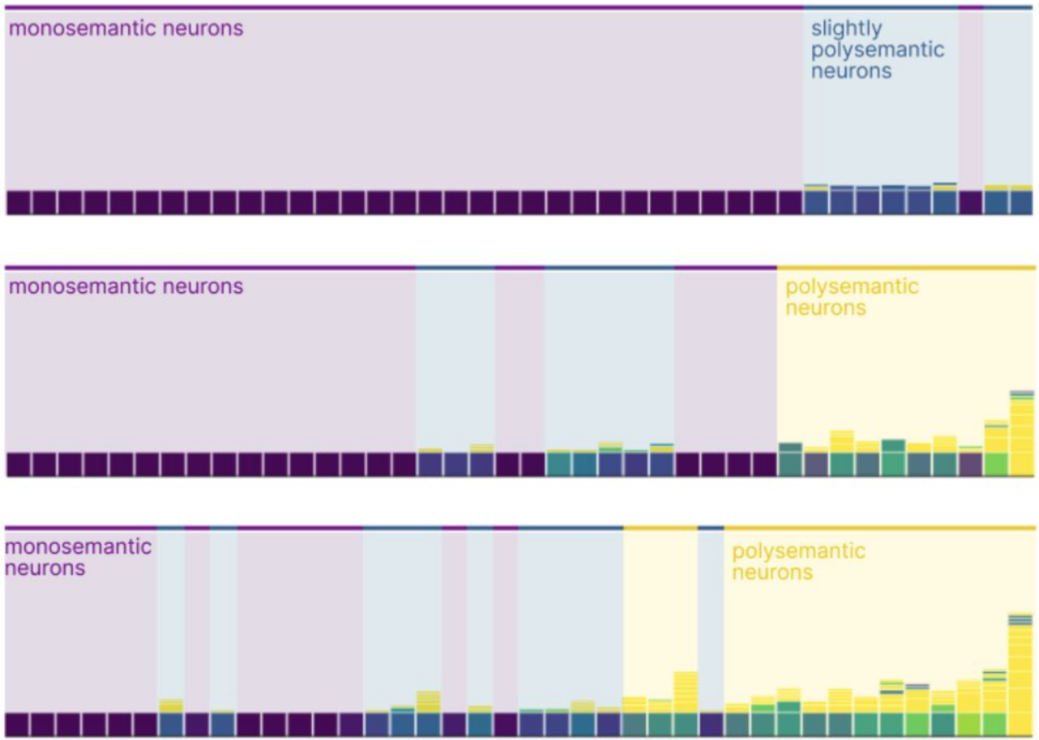
$$y' = \mathrm{ReLU}(W_2 h + b)$$



2. Model with sparsity and superposition

For $n = 100$, $m = 40$ and $I_i = 0.8^i$

2. Model with sparsity and superposition



$1 - S = 0.1$

Eventually, we start to see a few slightly polysemantic neurons.

$1 - S = 0.03$

As sparsity increases further, we see a small number of highly polysemantic neurons representing low importance features.

$1 - S = 0.01$

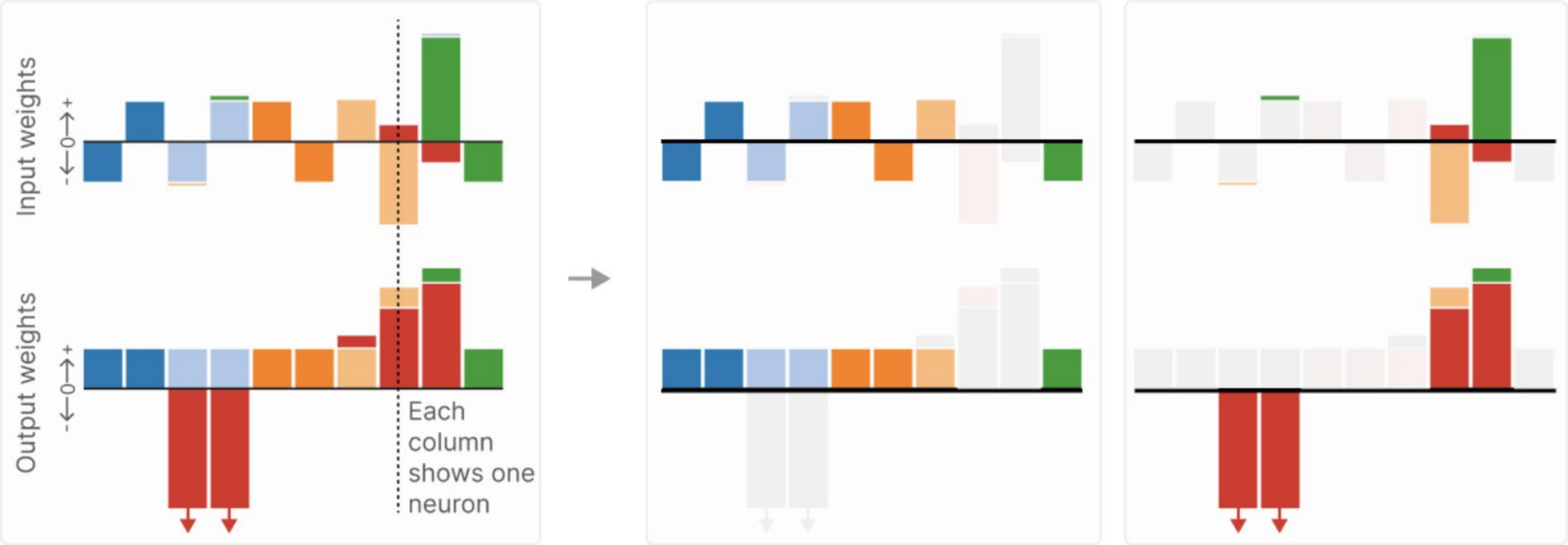
The number of polysemantic neurons grows...

2. What about weights?

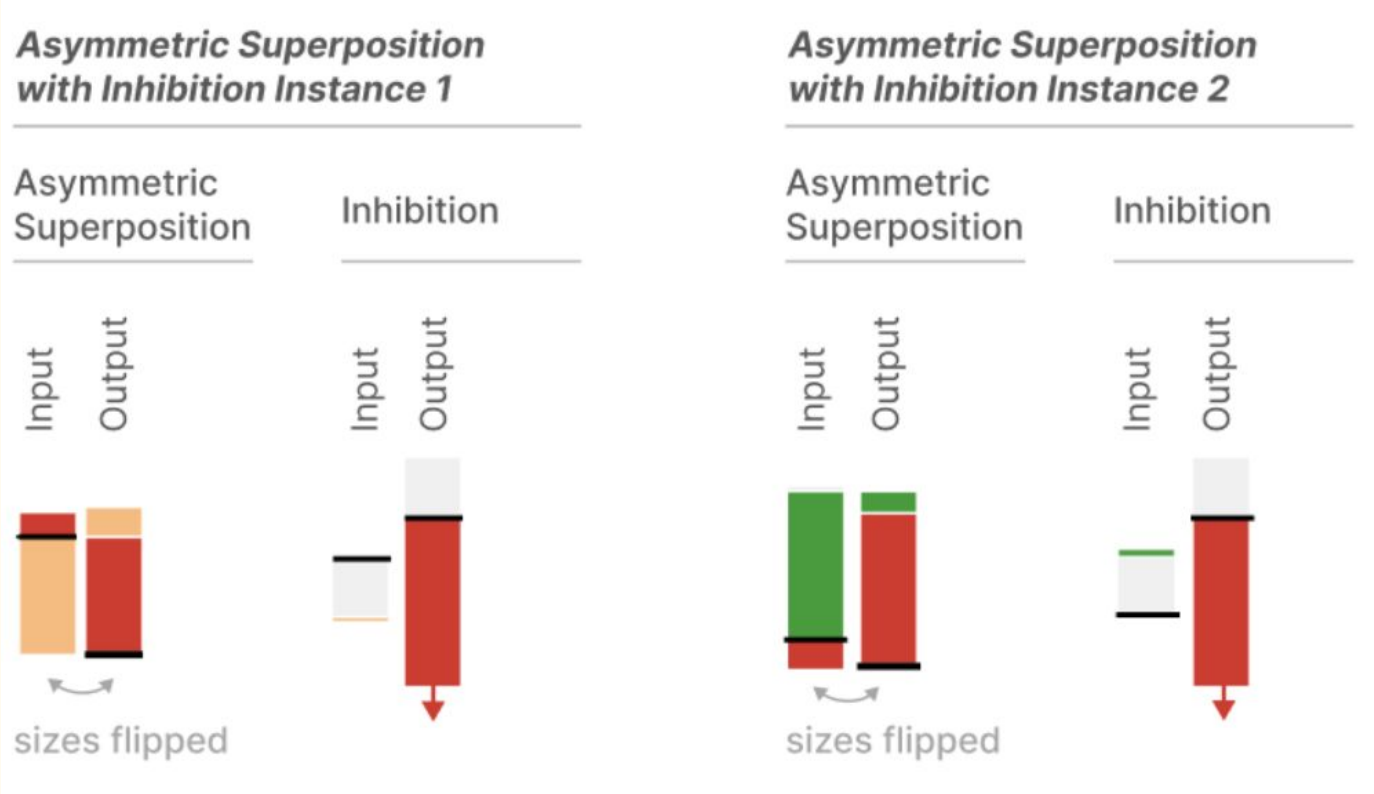
At first glance, this model is quite complicated and tricky to understand. However, we can (mostly) decompose it into two pieces...

Many weights are simply implementing absolute value, or a single side of absolute value, in the expected way.

The main other thing is ***asymmetric superposition with inhibition***. The model has two instances of this motif.



2. Some kind of inhibition happens



3. Is superposition good/bad?

Pros:

- it allows to store information in less memory, and do computations with less operations
- might be useful for training process observation

Cons:

- makes interpretability complicated