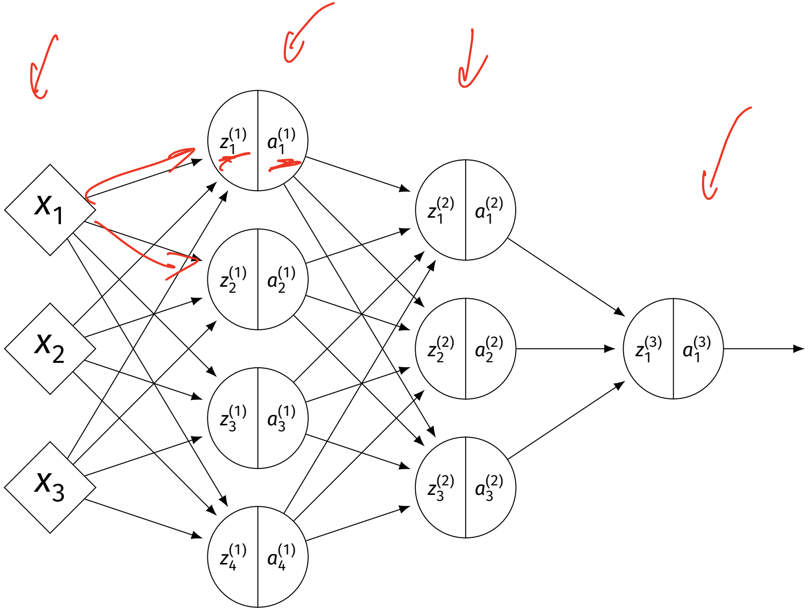
DSC 140B
Representation Learning

Lecture 22 | Part 1

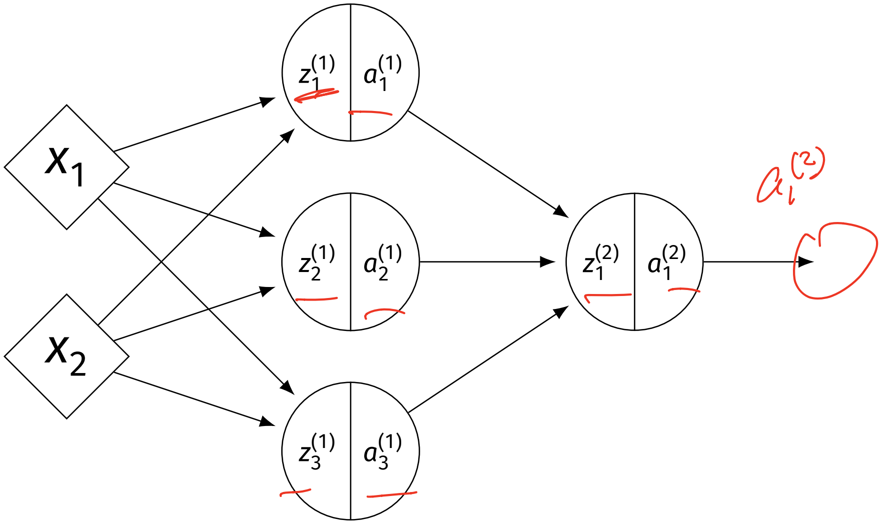
Neural Networks

Notation



- $z_j^{(i)}$ is the linear activation before g is applied.
- $a_j^{(i)} = g(z^{(i)})$ is the actual output of the neuron.

Example

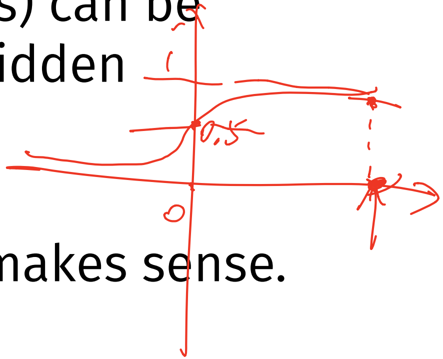


- g = ReLU
- Linear output
- $\vec{x} = (3, -1)^T$
- $z_1^{(1)} =$
- $a_1^{(1)} =$
- $z_2^{(1)} =$
- $a_2^{(1)} =$
- $z_3^{(1)} =$
- $a_3^{(1)} =$
- $z_1^{(2)} =$

$W^{(1)} = \begin{pmatrix} 2 & -1 & 0 \\ 4 & 5 & 2 \end{pmatrix}$ $W^{(2)} = \begin{pmatrix} 3 \\ 2 \\ -4 \end{pmatrix}$ $\vec{b}^{(1)} = (3, -2, -2)^T$ $\vec{b}^{(2)} = (-4)^T$

Output Activations

- The activation of the output neuron(s) can be different than the activation of the hidden neurons.



- In classification, **sigmoid** activation makes sense.
- In regression, **linear** activation makes sense.

Main Idea

A neural network with linear activations is a linear model. If non-linear activations are used, the model is made non-linear.

DSC 140B
Representation Learning

Lecture 22 | Part 2

Demo

Feature Map

- We have seen how to fit non-linear patterns with linear models via **basis functions** (i.e., a feature map).

$$H(\vec{x}) = w_0 + w_1\phi_1(\vec{x}) + \ldots + w_k\phi_k(\vec{x})$$

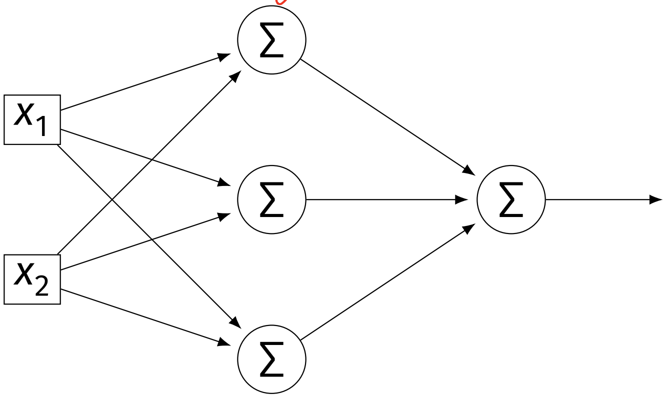
- These basis functions are fixed **before** learning.
- **Downside:** we have to choose $\vec{\phi}$ somehow.

Learning a Feature Map

- **Interpretation:** The hidden layers of a neural network **learn** a feature map.

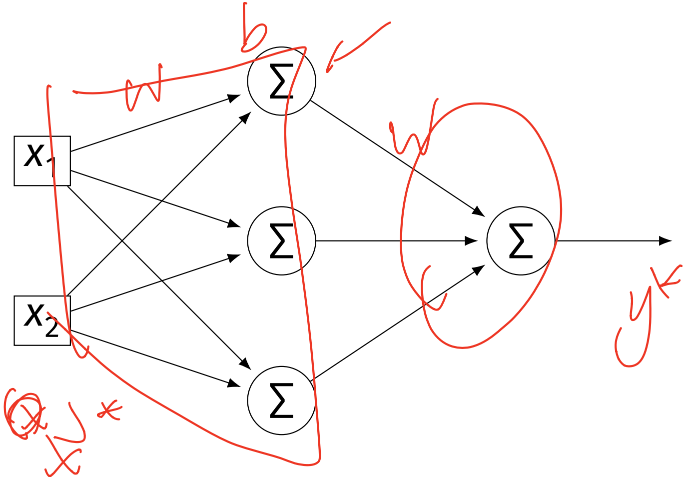
Each Layer is a Function

- We can think of each layer as a function mapping a vector to a vector.
- $H^{(1)}(\vec{z}) = \left[W^{(1)}\right]^T \vec{z} + \vec{b}^{(1)}$
  - $H^{(1)} : \mathbb{R}^2 \to \mathbb{R}^3$
- $H^{(2)}(\vec{z}) = \left[W^{(2)}\right]^T \vec{z} + \vec{b}^{(2)}$
  - $H^{(2)} : \mathbb{R}^3 \to \mathbb{R}^1$



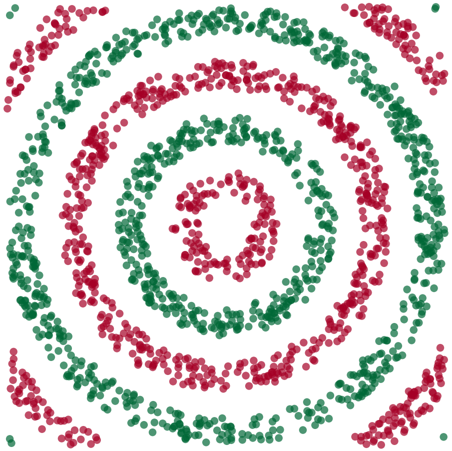
Each Layer is a Function

- The hidden layer performs a feature map from $\mathbb{R}^2$ to $\mathbb{R}^3$.
- The output layer makes a prediction in $\mathbb{R}^3$.
- **Intuition:** The feature map is learned so as to make the output layer's job "easier".



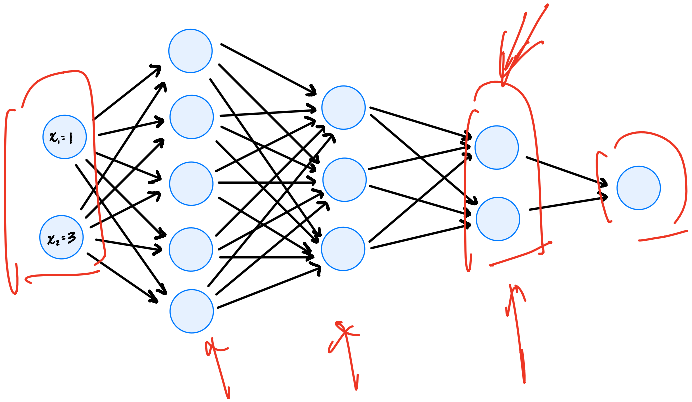
Demo

- Train a deep network to classify the data below.
- Hidden layers will learn a new feature map that makes the data linearly separable.



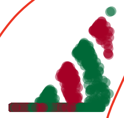
Demo

- We'll use three hidden layers, with last having two neurons.
- We can see this new representation!
- Plug in $\vec{x}$ and see activations of last hidden layer.

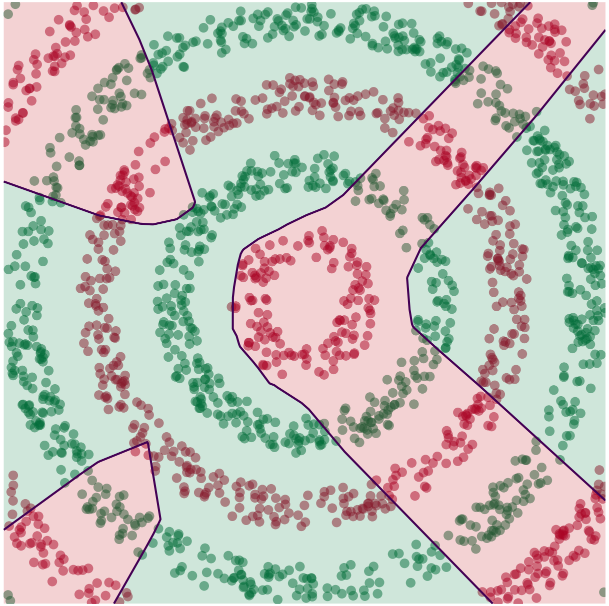


Learning a New Representation

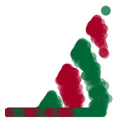
2d feature 3rd hidden layer

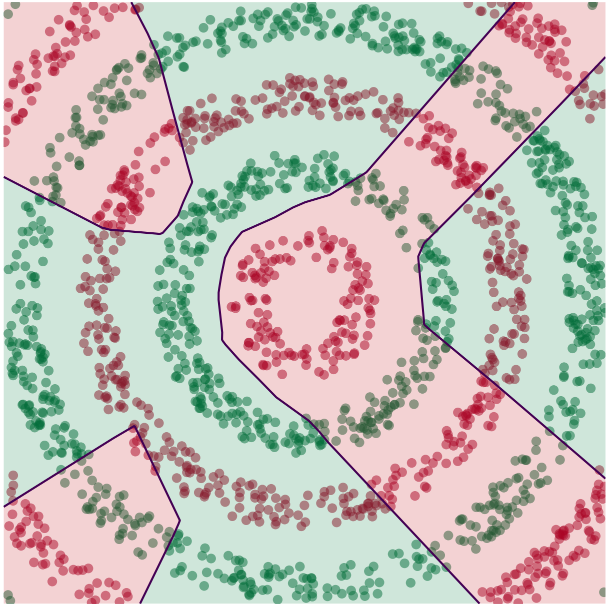


decision boundary

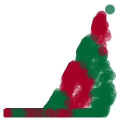


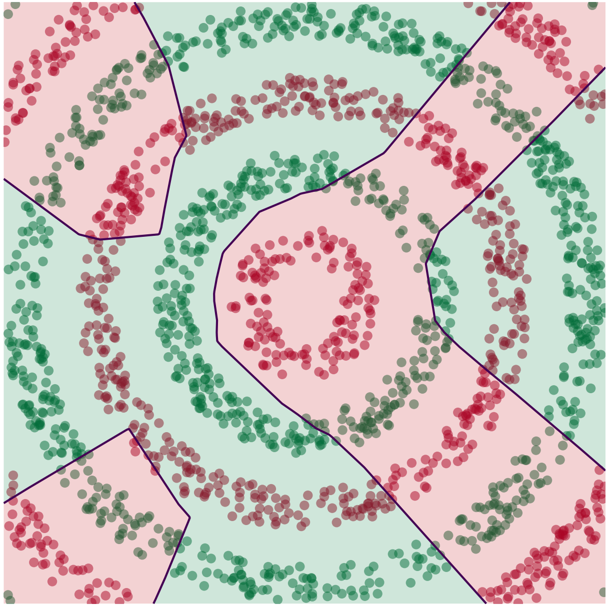
Learning a New Representation



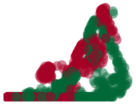


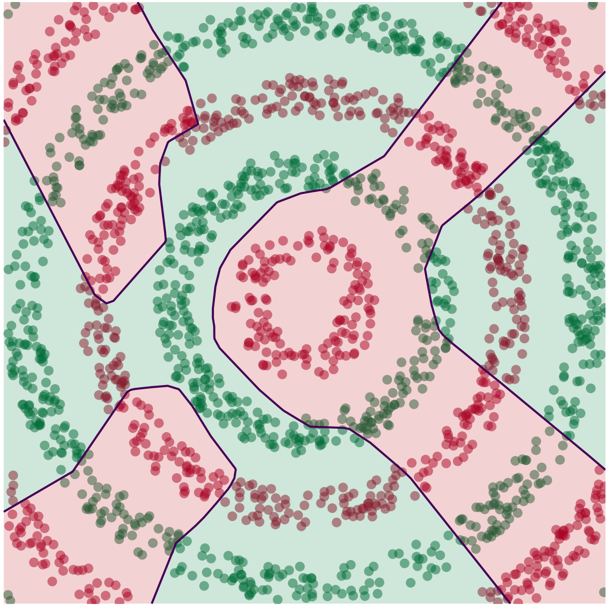
Learning a New Representation



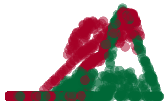


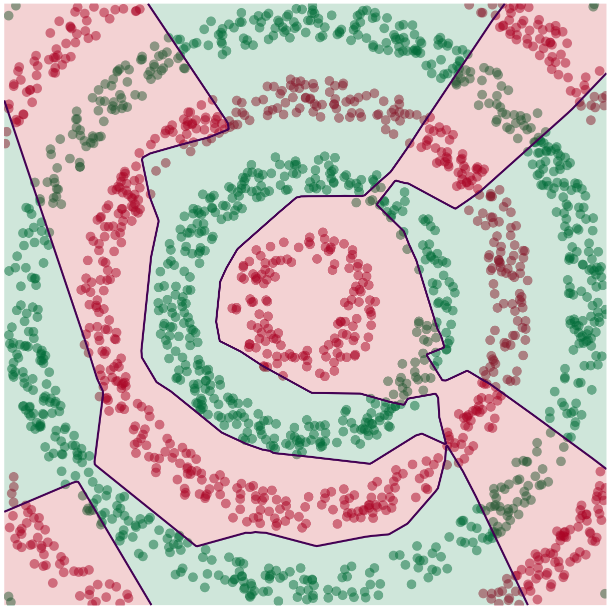
Learning a New Representation



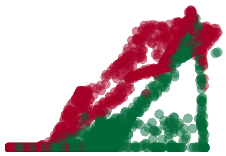


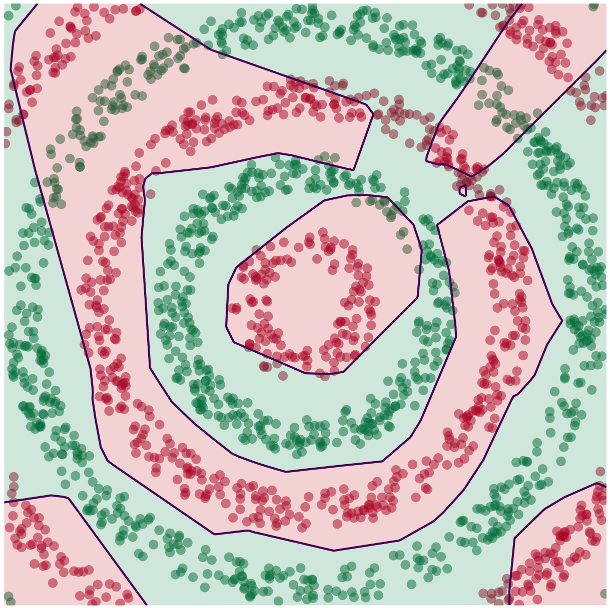
Learning a New Representation



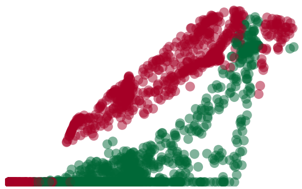


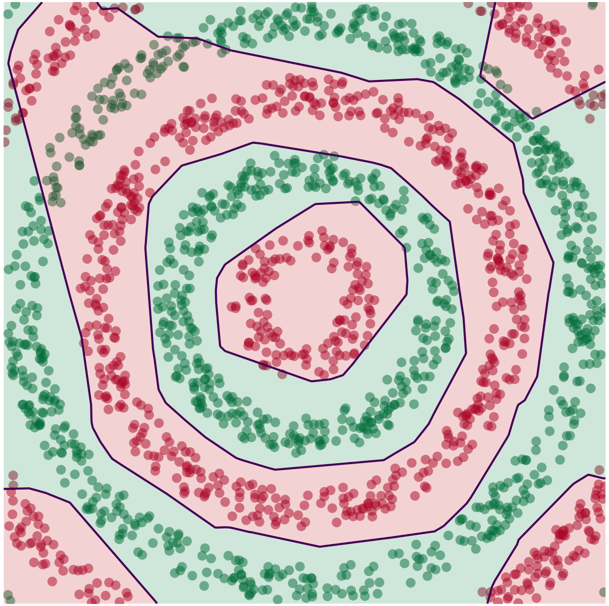
Learning a New Representation



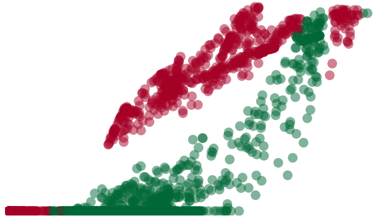


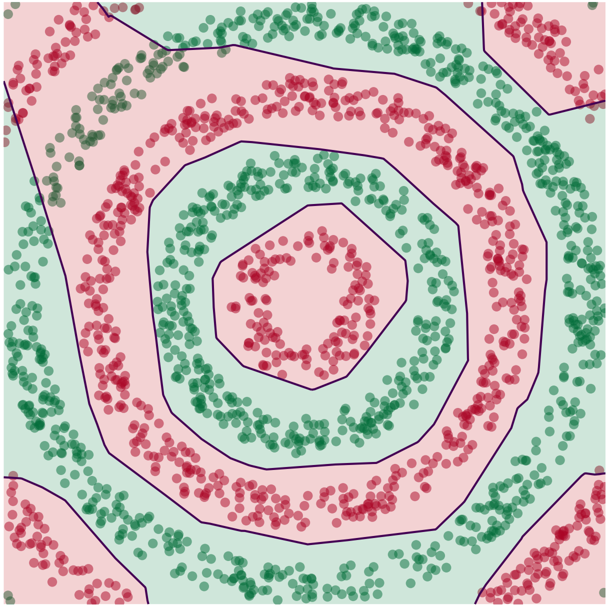
Learning a New Representation



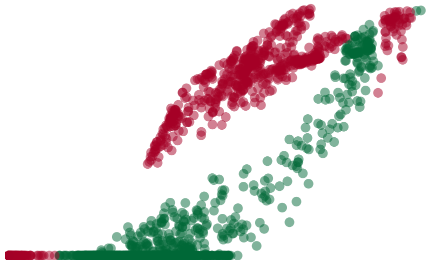


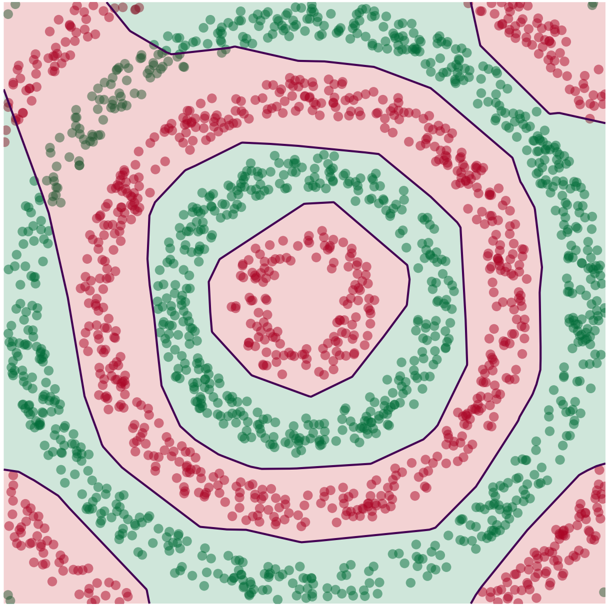
Learning a New Representation



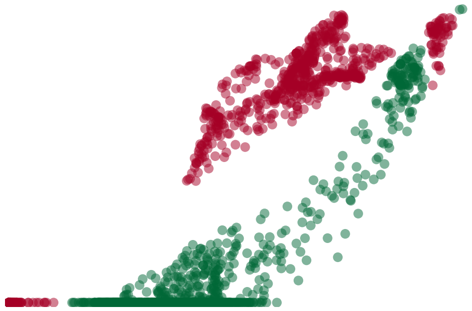


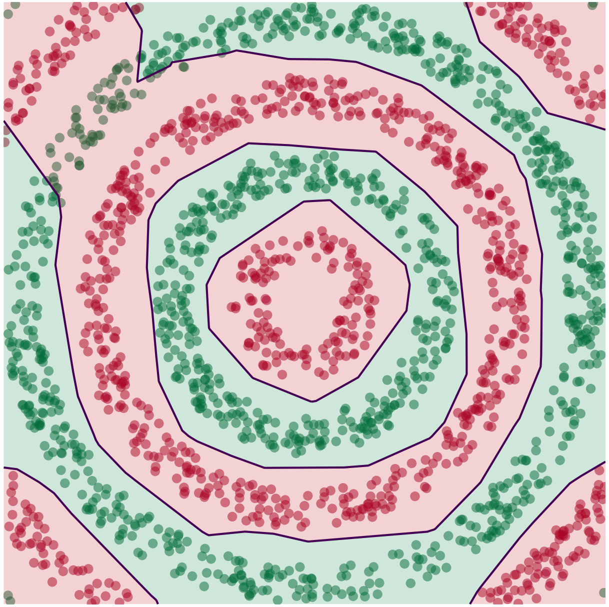
Learning a New Representation



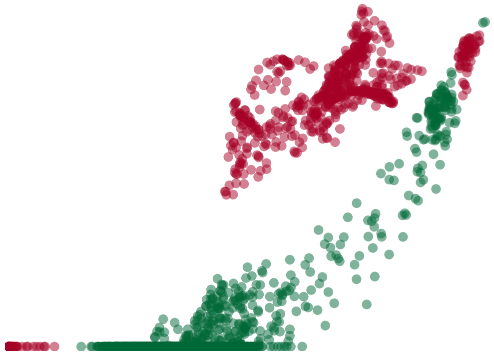


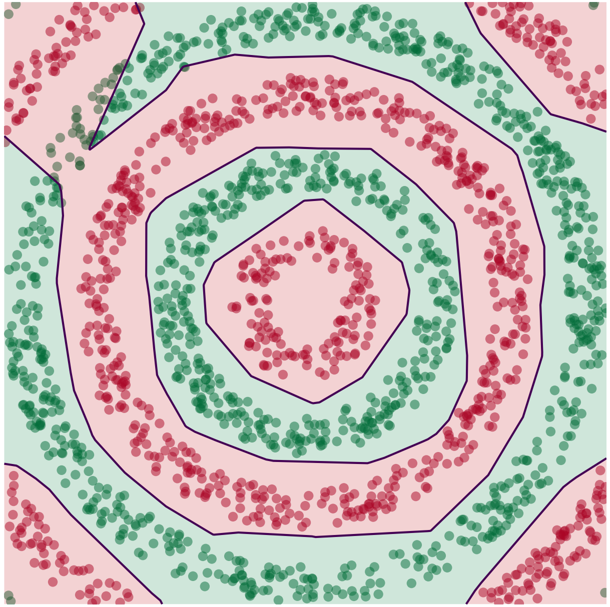
Learning a New Representation



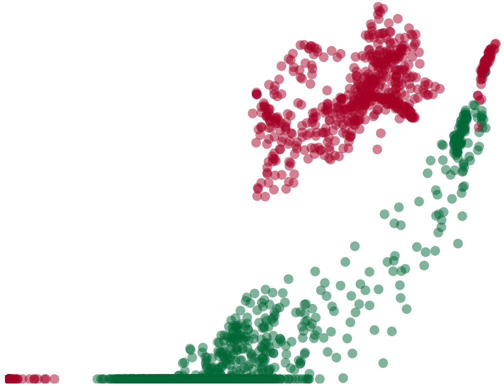


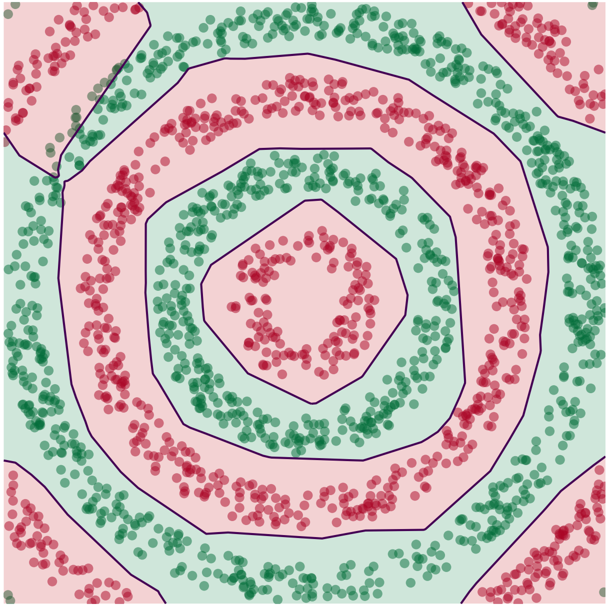
Learning a New Representation



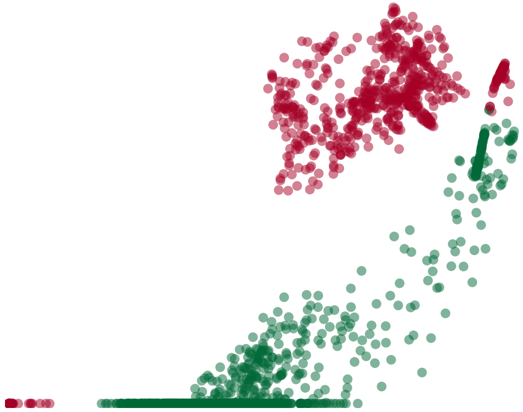


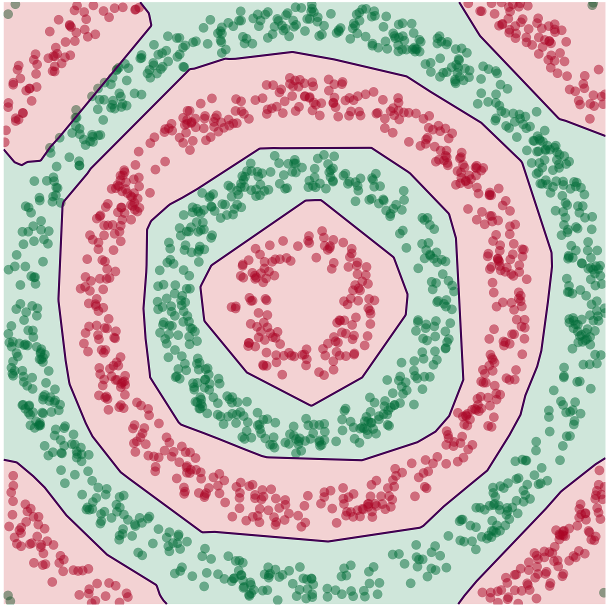
Learning a New Representation



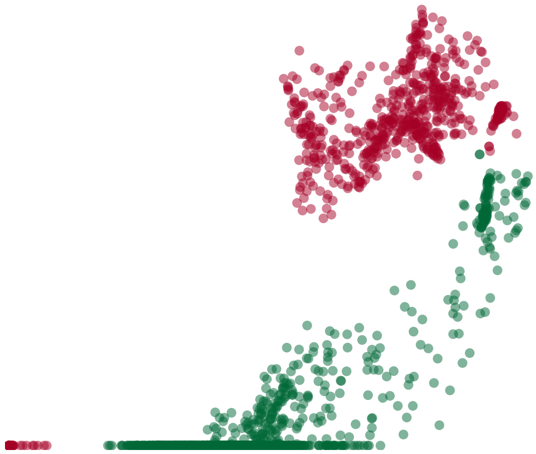


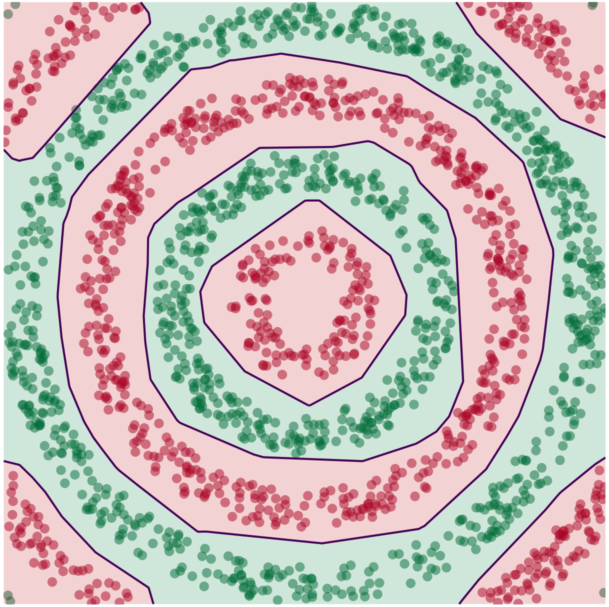
Learning a New Representation



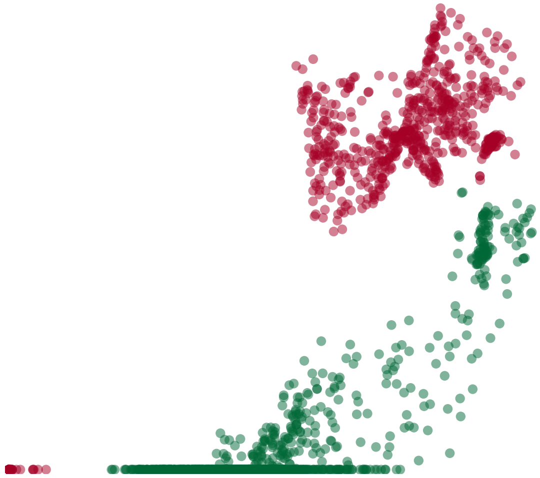


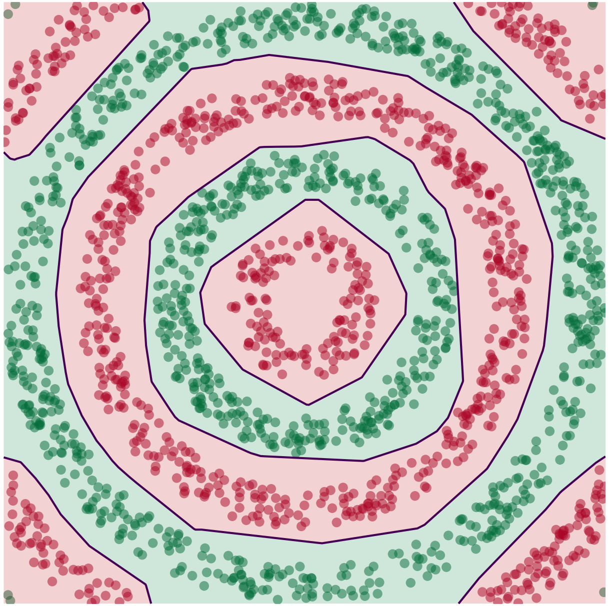
Learning a New Representation



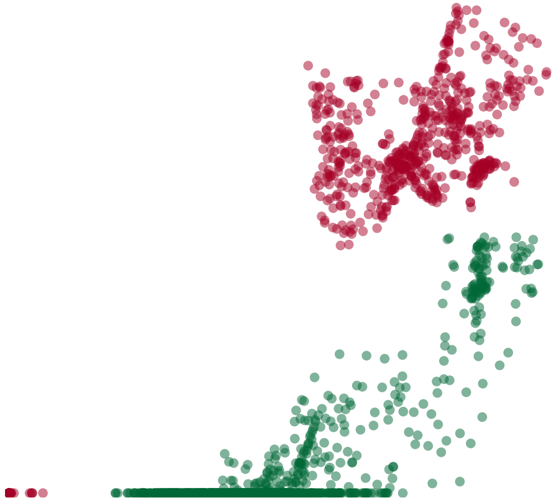


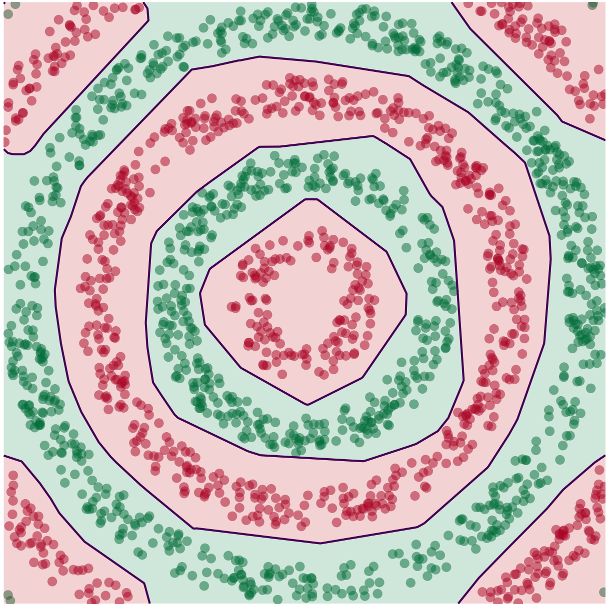
Learning a New Representation



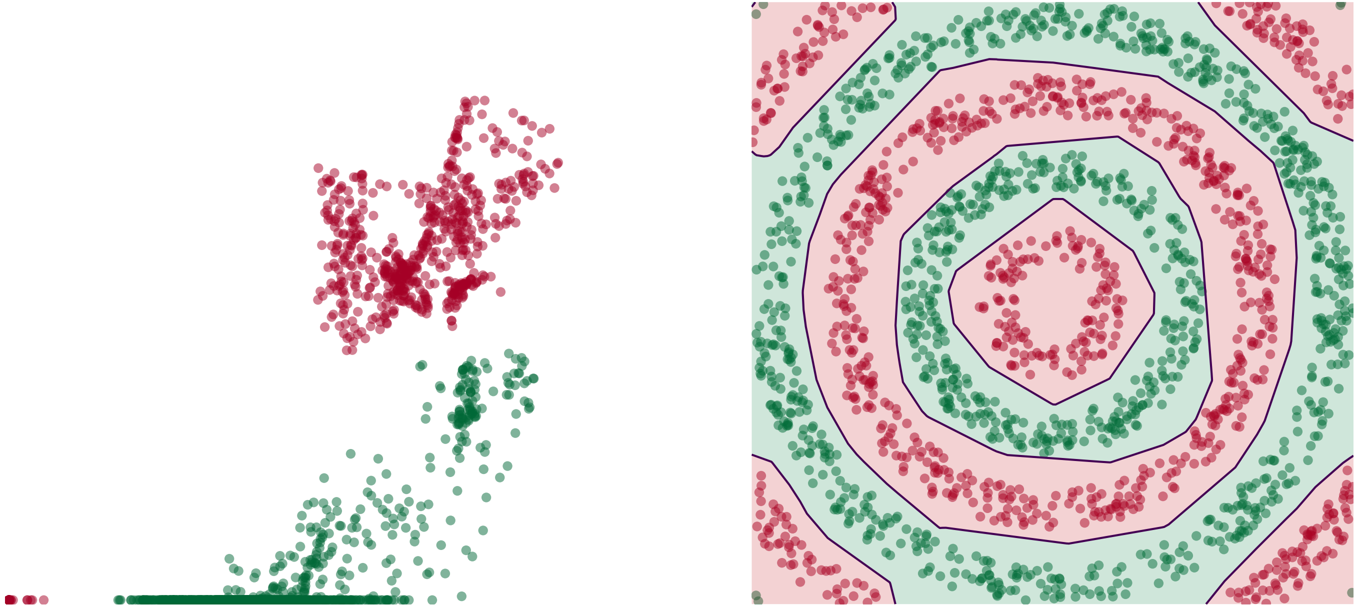


Learning a New Representation

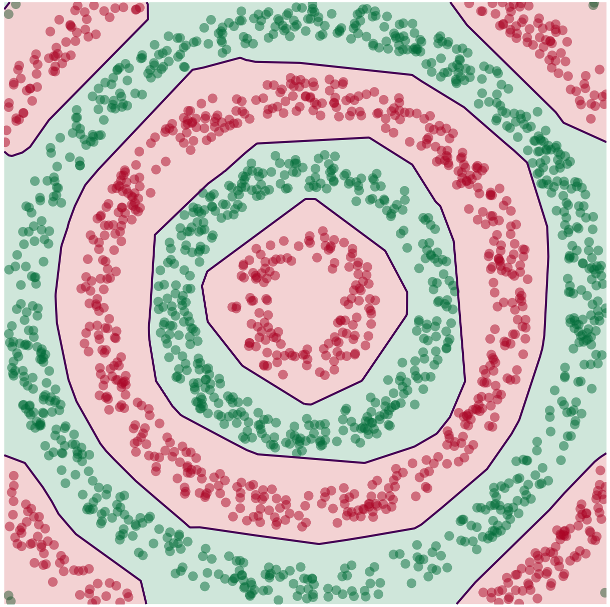




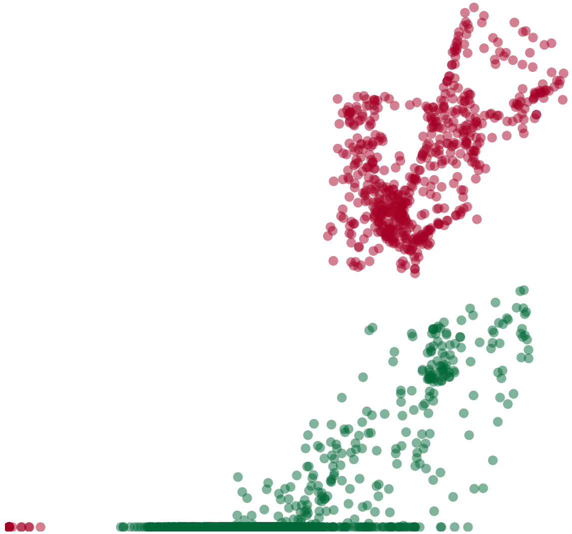
Learning a New Representation

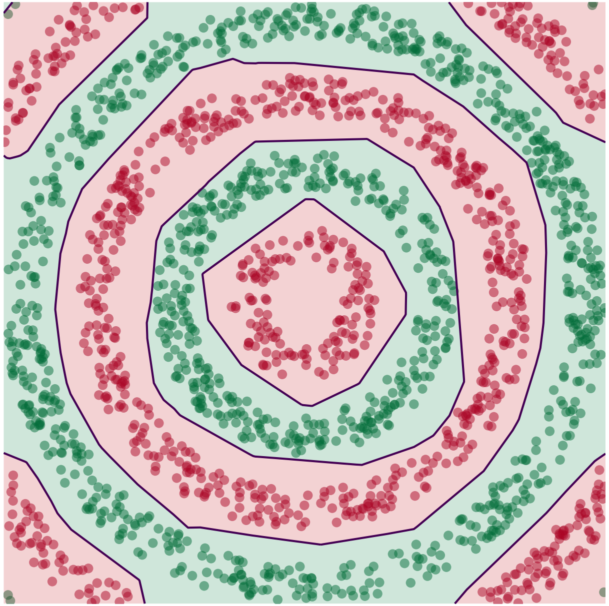


Learning a New Representation

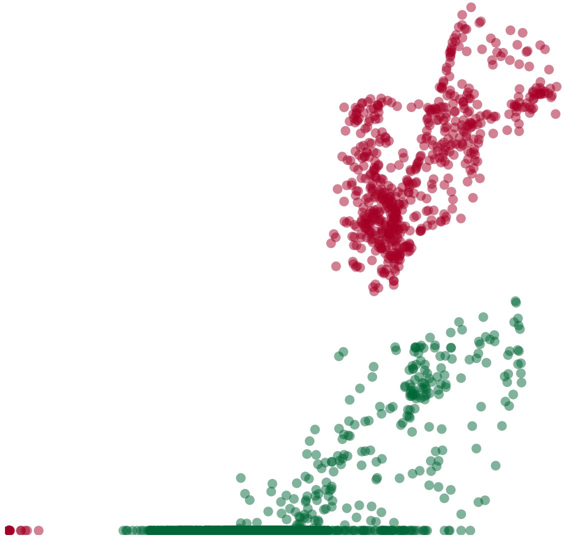


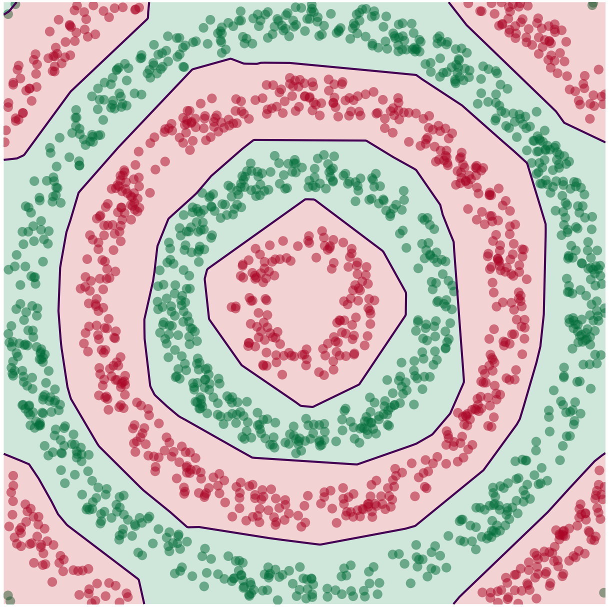
Learning a New Representation



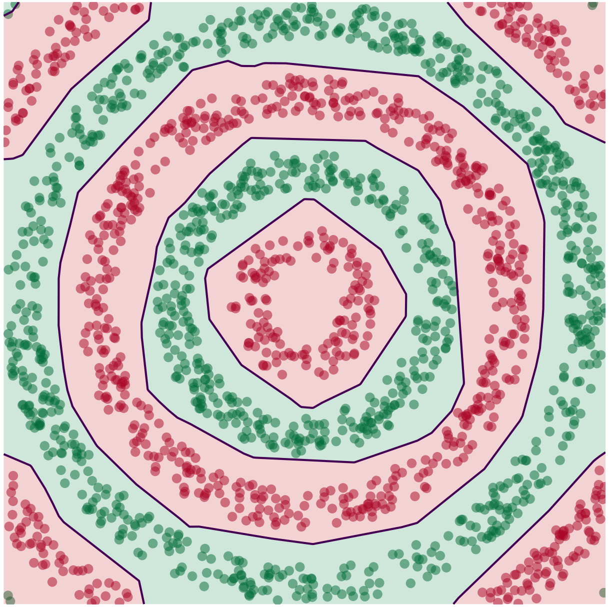


Learning a New Representation

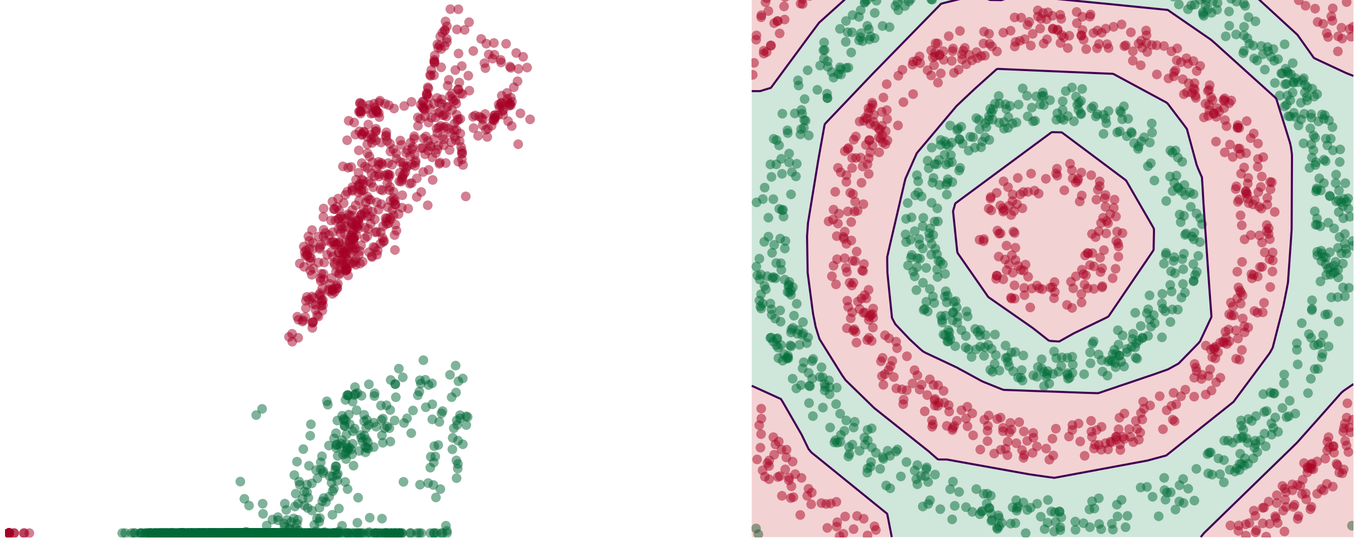




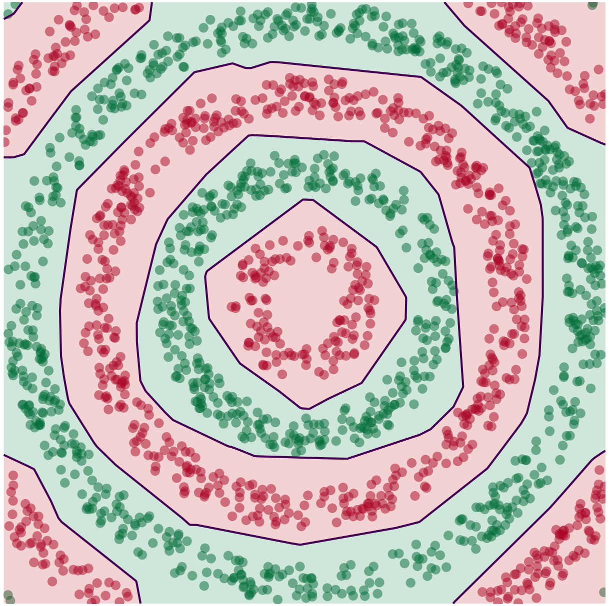
Learning a New Representation



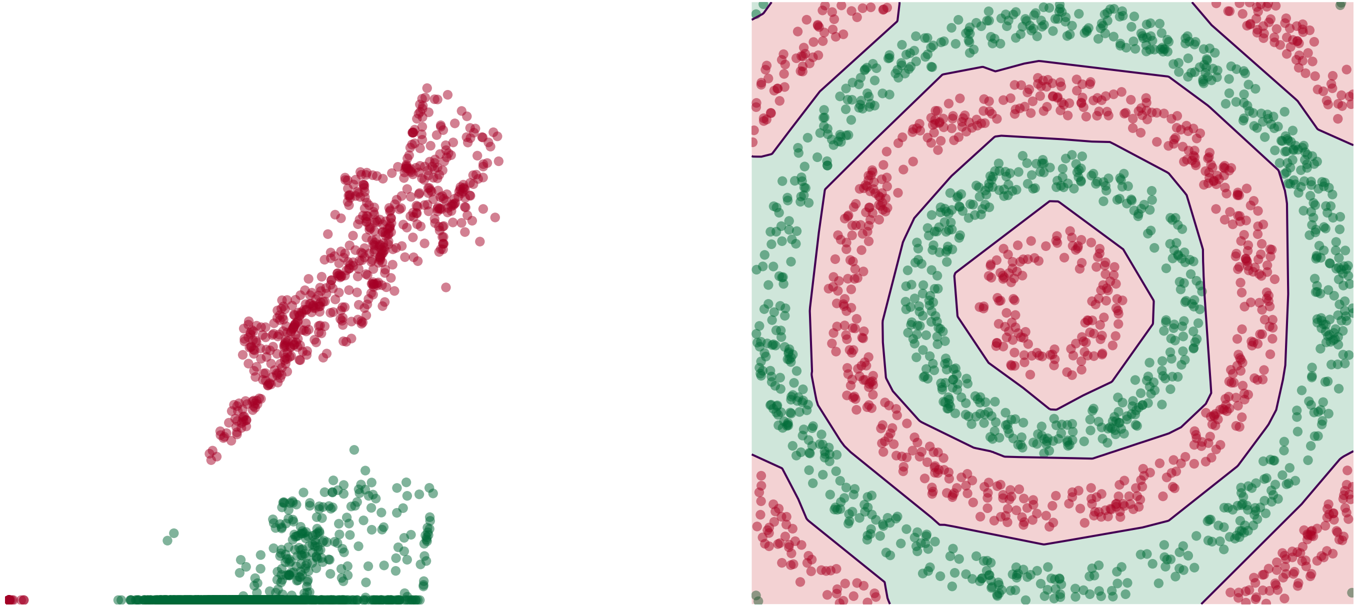
Learning a New Representation



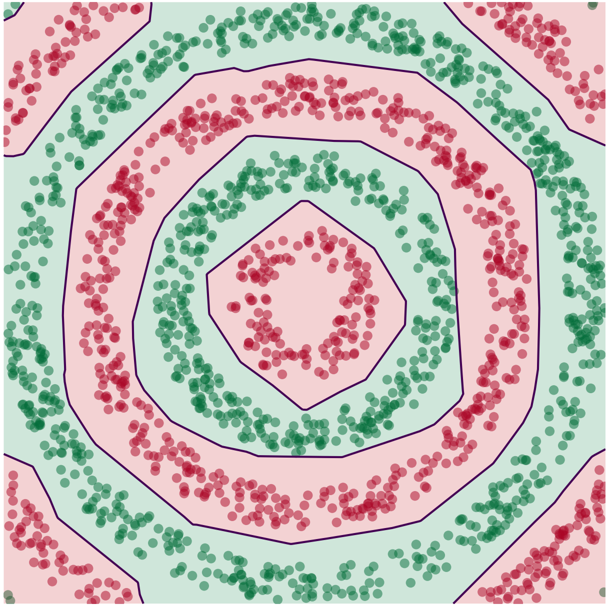
Learning a New Representation



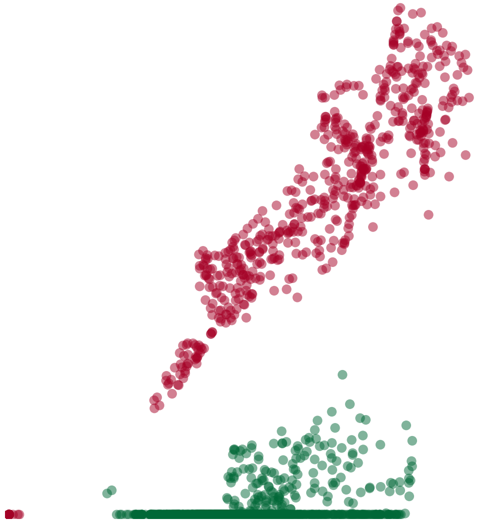
Learning a New Representation

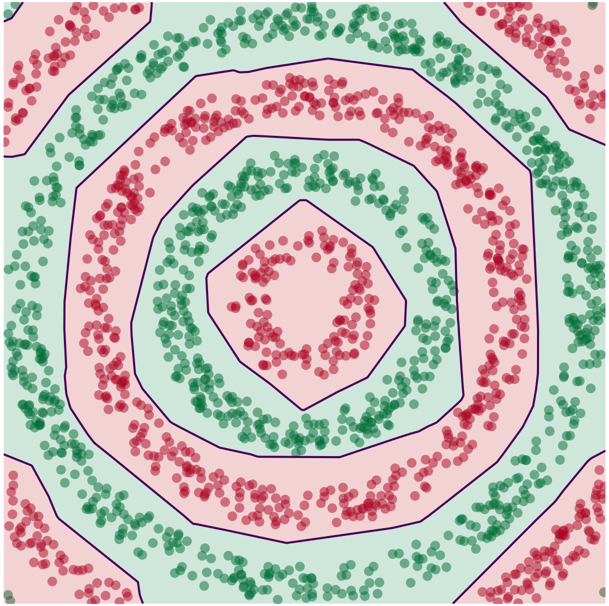


Learning a New Representation

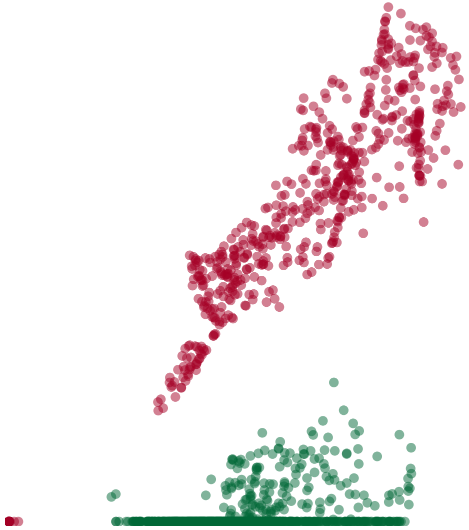


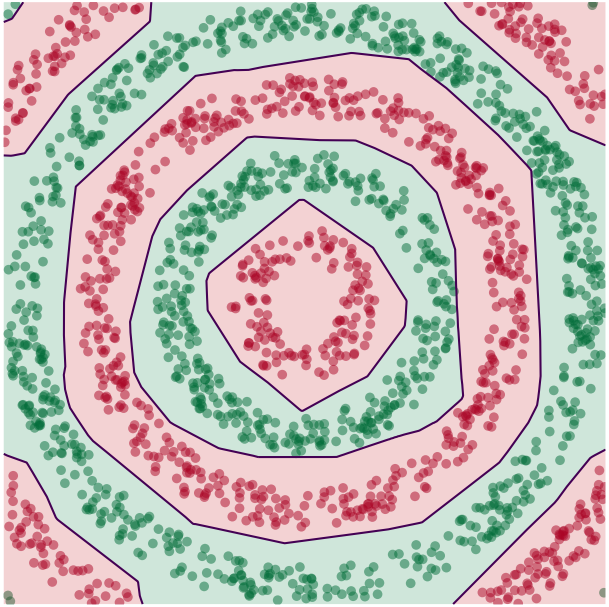
Learning a New Representation



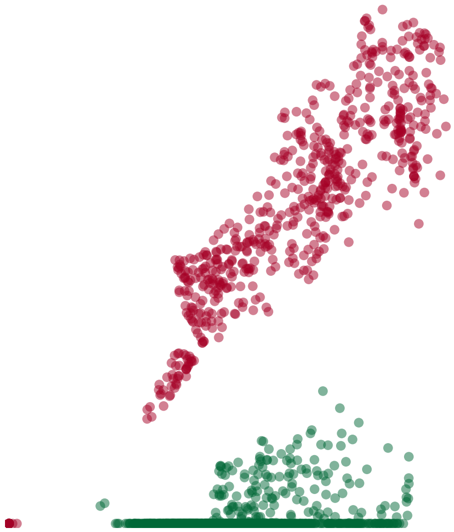


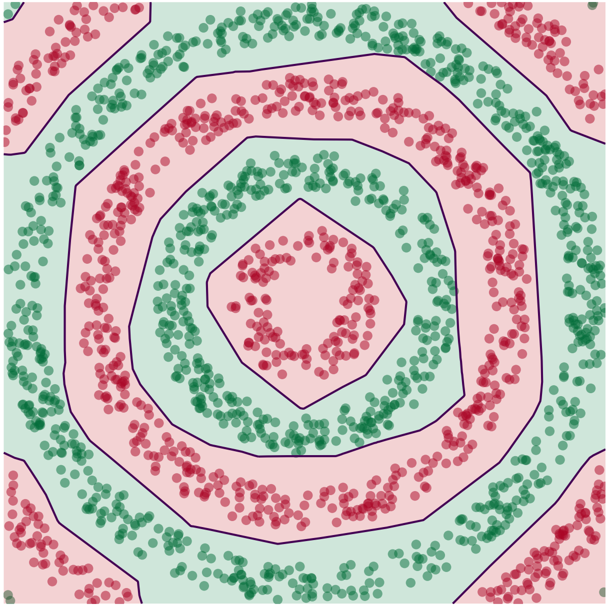
Learning a New Representation



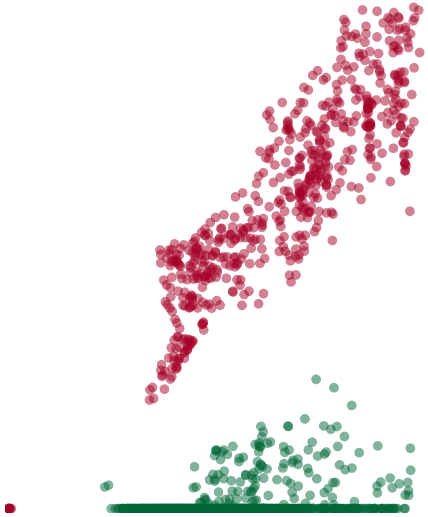


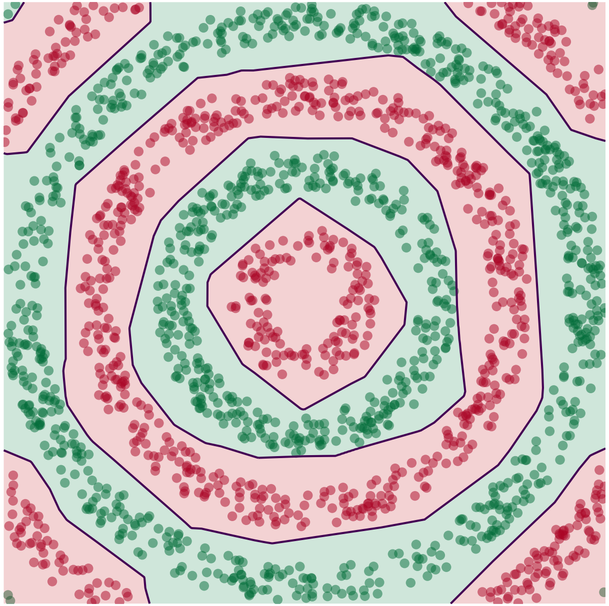
Learning a New Representation



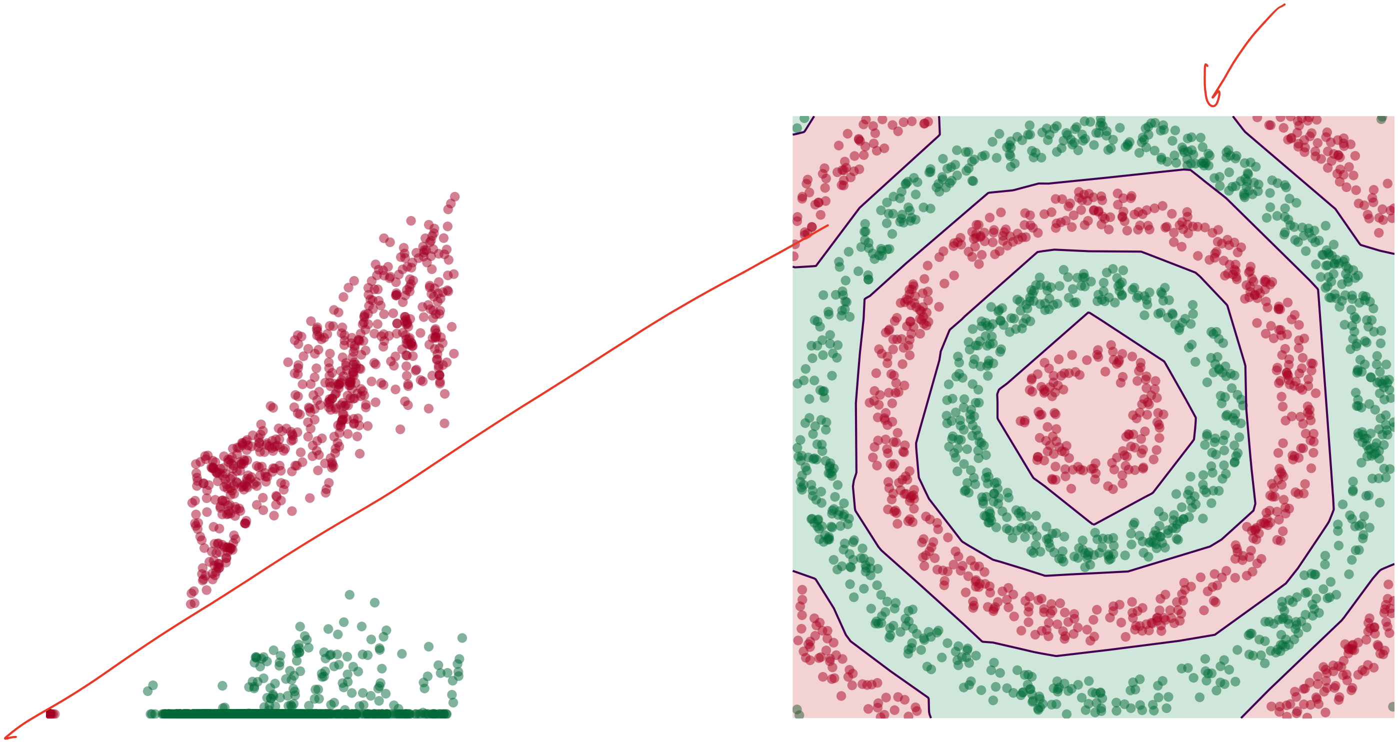


Learning a New Representation





Learning a New Representation



Deep Learning

- The NN has learned a new **representation** in which the data is easily classified.

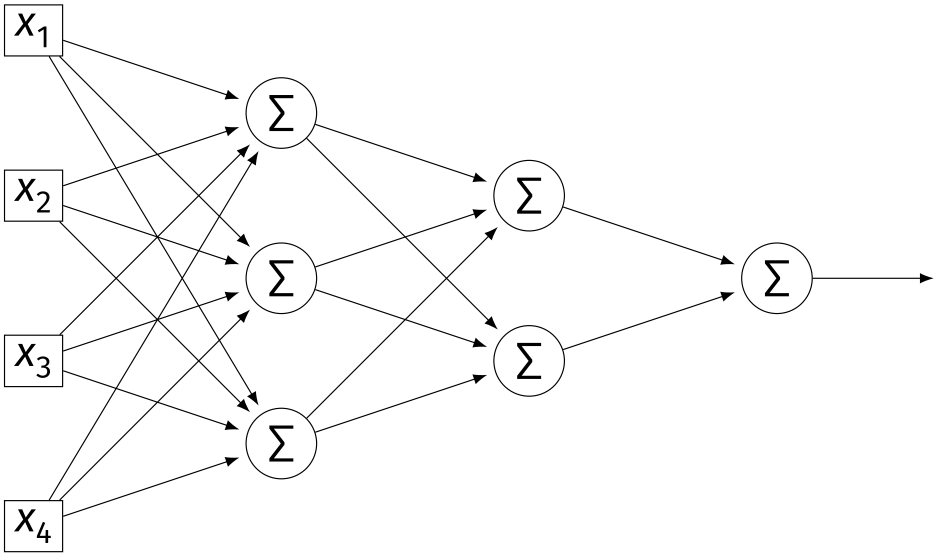
DSC 140B
Representation Learning

Lecture 22 | Part 3

Training Neural Networks

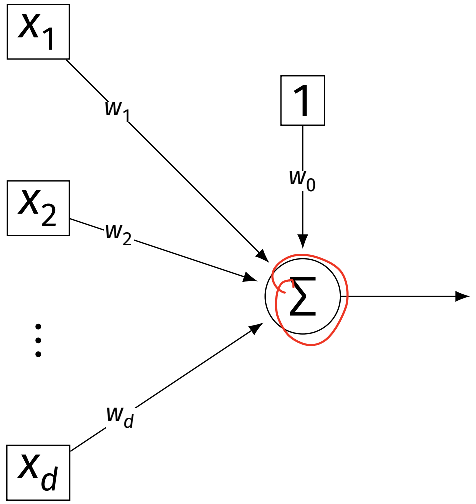
Training

- How do we learn the weights of a (deep) neural network?



Remember...

- How did we learn the weights in linear least squares regression?



Empirical Risk Minimization

0. Collect a training set, $\{(\vec{x}^{(i)}, y_i)\}$
1. Pick the form of the prediction function, H.
2. Pick a loss function.
3. Minimize the empirical risk w.r.t. that loss.

Remember: Linear Least Squares

0. Pick the form of the prediction function, H.
   - E.g., linear: $H(\vec{x}; \vec{w}) = w_0 + w_1 x_1 + \ldots + w_d x_d = \text{Aug}(\vec{x}) \cdot \vec{w}$
1. Pick a loss function.
   - E.g., the square loss.

   $\| H(x^{(i)}; \vec{w}) - y \|^2$
2. Minimize the empirical risk w.r.t. that loss:

$$\min_{\vec{w}} \; R_{\text{sq}}(\vec{w}) = \frac{1}{n} \sum_{i=1}^{n} \left(H(\vec{x}^{(i)}) - y_i \right)^2 = \frac{1}{n} \sum_{i=1}^{n} \left(\text{Aug}(\vec{x}^{(i)}) \cdot \vec{w} - y_i \right)^2$$

Minimizing Risk

- To minimize risk, we often use **vector calculus.**
  - Either set $\nabla_{\vec{w}} R(\vec{w}) = 0$ and solve...
  - Or use gradient descent: walk in opposite direction of $\nabla_{\vec{w}} R(\vec{w})$.
- Recall, $\nabla_{\vec{w}} R(\vec{w}) = (\partial R/\partial w_0, \partial R/\partial w_1, \ldots, \partial R/\partial w_d)^T$

In General

- Let ℓ be the loss function, let $H(\vec{x}; \vec{w})$ be the prediction function.

- The empirical risk:

$$R(\vec{w}) = \frac{1}{n} \sum_{i=1}^{n} \ell(H(\vec{x}^{(i)}; \vec{w}), y_i)$$

- Using the chain rule:

$$\nabla_{\vec{w}} R(\vec{w}) = \frac{1}{n} \sum_{i=1}^{n} \frac{\partial \ell}{\partial H} \nabla_{\vec{w}} H(\vec{x}^{(i)}; \vec{w})$$

Gradient of H

- To minimize risk, we want to compute $\nabla_{\vec{w}} R$.
- To compute $\nabla_{\vec{w}} R$, we want to compute $\nabla_{\vec{w}} H$.
- This will depend on the form of H.

Example: Linear Model

- Suppose H is a linear prediction function:

$$H(\vec{x}; \vec{w}) = w_0 + w_1 x_1 + \ldots + w_d x_d$$

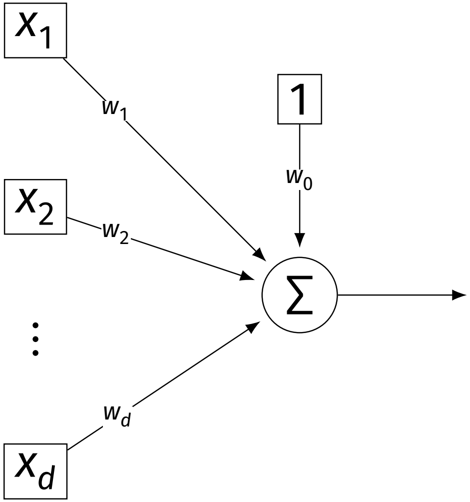
- What is $\nabla_{\vec{w}} H$ with respect to $\vec{w}$?

$$\nabla_{\vec{w}} H(\vec{w}) = \left(\frac{\partial H}{\partial w_0}, \frac{\partial H}{\partial w_1}, \ldots, \frac{\partial H}{\partial w_d} \right)^T$$

$$= \left(1, x_1, \ldots, x_d \right)^T$$

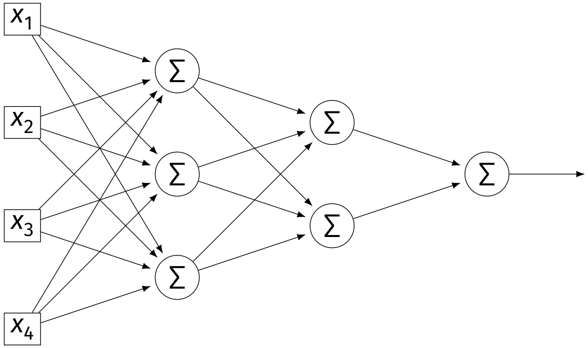
Example: Linear Model

- Consider $\partial H/\partial w_1$: $= x_1$



Example: Neural Networks

- Suppose H is a neural network (with nonlinear activations).
- What is ∇H?
  - It's more complicated...



Parameter Vectors

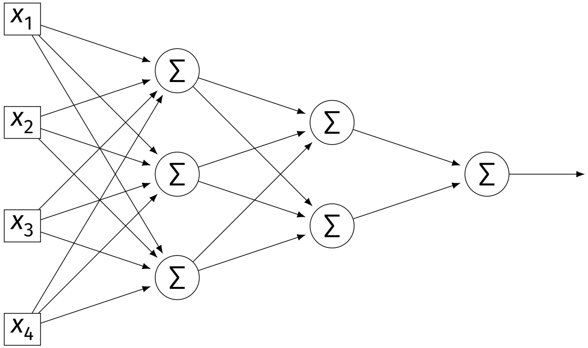
- It is often useful to pack all of the network's weights into a **parameter vector**, $\vec{w}$.
- Order is arbitrary:

$$\vec{w} = (W_{11}^{(1)}, W_{12}^{(1)}, \ldots, b_1^{(1)}, b_2^{(1)}, W_{11}^{(2)}, W_{12}^{(2)}, \ldots, b_1^{(2)}, b_2^{(2)}, \ldots)^T$$

- The network is a function $H(\vec{x}; \vec{w})$.
- Goal of learning: find the "best" $\vec{w}$.

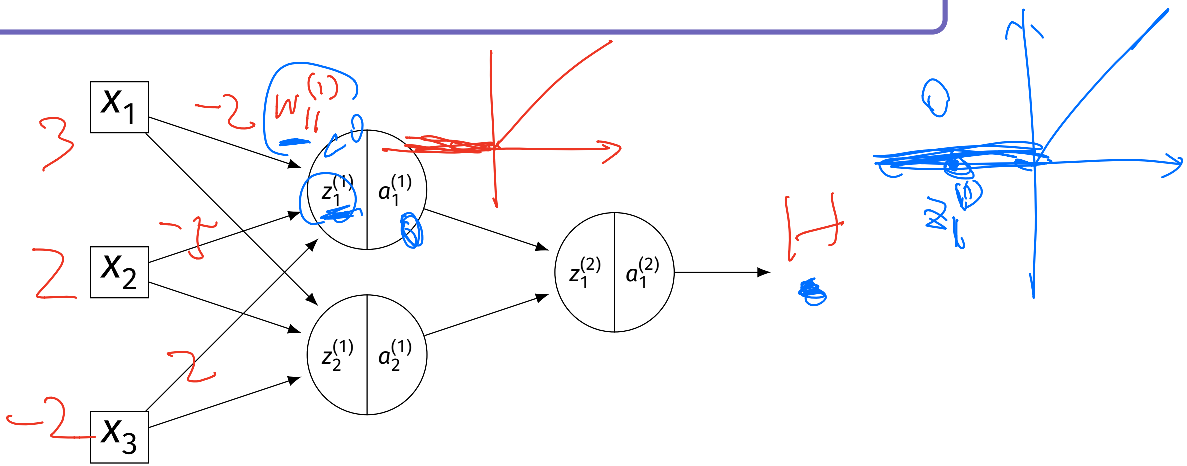
Gradient of Neural Network

- $\nabla_{\vec{w}} H$ is a vector-valued function.
- Plugging a data point, $\vec{x}$, and a parameter vector, $\vec{w}$, into $\nabla_{\vec{w}} H$ "evaluates the gradient", results in a vector, same size as $\vec{w}$.



Exercise

Suppose $W_{11}^{(1)} = -2, W_{21}^{(1)} = -5, W_{31}^{(1)} = 2$ and $\vec{x} = (3, 2, -2)^T$ and all biases are 0. ReLU activations are used. What is $\partial H / \partial W_{11}^{(1)}(\vec{x}, \vec{w})$?



Example

chain rule

$z_1^{(3)}(\vec{x}; \vec{w})$

$z_1^{(3)} = a_1^{(2)} w_{11}^{(3)} + a_2^{(2)} w_{21}^{(3)} + \cdots + a_5^{(2)} w_{51}^{(3)} + b_1^{(3)}$

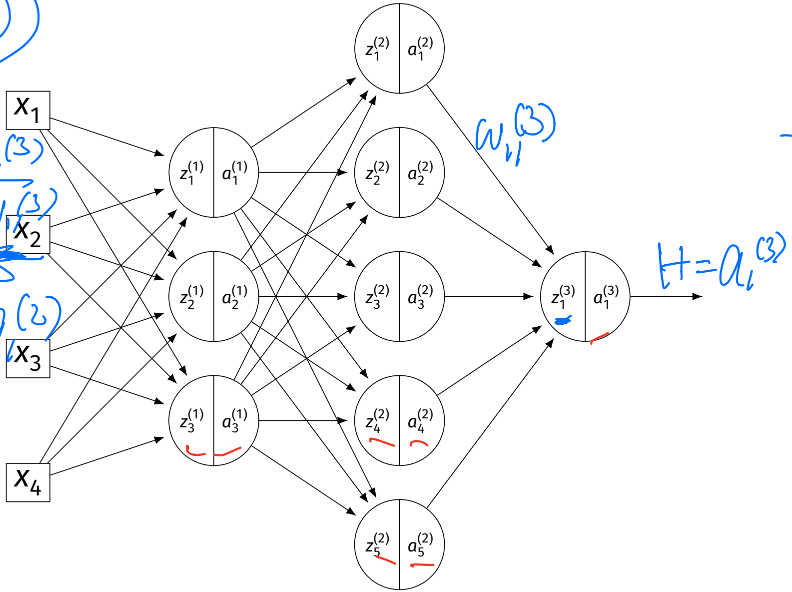
- Consider $\partial H / \partial W_{11}^{(3)}$:

$a_1^{(3)} = g(z_1^{(3)}(\vec{x}; \vec{w}))$

$\frac{\partial a_1^{(3)}}{\partial w_{11}^{(3)}} = \frac{\partial a_1^{(3)}}{\partial z_1^{(3)}} \cdot \frac{\partial z_1^{(3)}}{\partial w_{11}^{(3)}}$

$= g'(z_1^{(3)}) \cdot a_1^{(2)}$

$\frac{\partial z_1^{(3)}}{\partial w_{11}^{(3)}} = a_1^{(2)}$

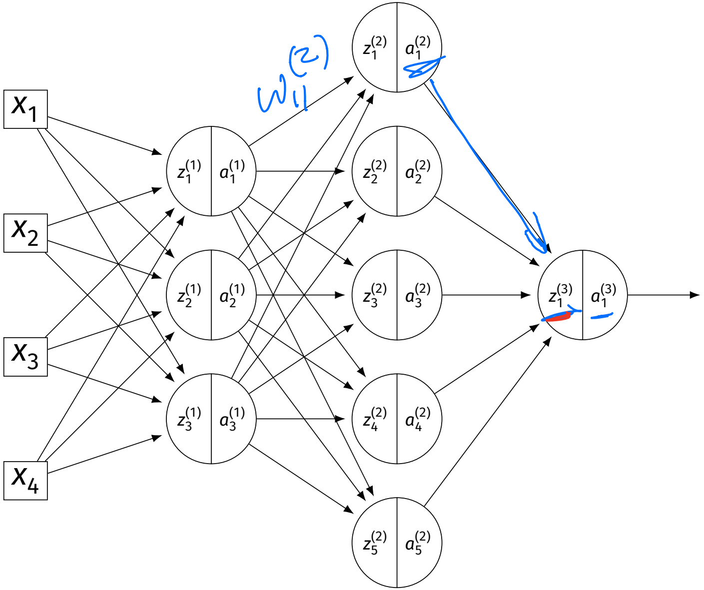


Example

- Consider $\partial H / \partial W_{11}^{(2)}$:

$$\frac{\partial H}{\partial W_{11}^{(2)}} = \frac{\partial a_1^{(3)}}{\partial W_{11}^{(2)}} = \frac{\partial a_1^{(3)}}{\partial z_1^{(3)}} \frac{\partial z_1^{(3)}}{\partial a_1^{(2)}} \frac{\partial a_1^{(2)}}{\partial z_1^{(2)}} \times \frac{\partial z_1^{(2)}}{\partial W_{11}^{(2)}}$$

$$\frac{\partial z_1^{(2)}}{\partial W_{11}^{(2)}} \rightarrow a_1^{(1)}$$



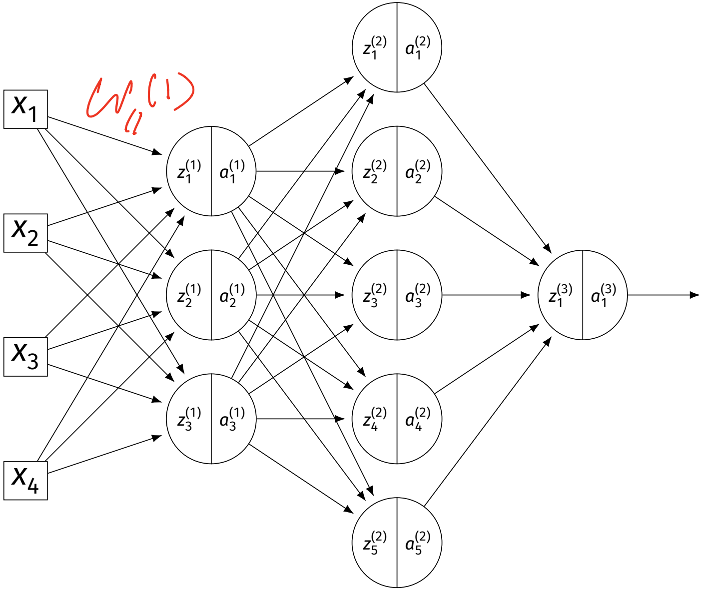
$$\frac{\partial z_1^{(3)}}{\partial a_1^{(2)}} = w_1^{(3)}$$

$$\frac{\partial a_1^{(2)}}{\partial z_1^{(2)}} = g'(z_1^{(2)})$$

Example

- Consider $\partial H / \partial W_{11}^{(1)}$:

$$\frac{\partial H}{\partial W_{11}^{(1)}} = \dots$$



A Better Way

- Computing the gradient is straightforward...
- But can involve a lot of repeated work.
- **Backpropagation** is an algorithm for efficiently computing the gradient of a neural network.

DSC 140B
Representation Learning

Lecture 22 | Part 4

Backpropagation

Gradient of a Network

- We want to compute the gradient $\nabla_{\vec{w}} H$.
  - That is, $\partial H / \partial W_{ij}^{(\ell)}$ and $\partial H / \partial b_i^{(\ell)}$ for all valid i, j, ℓ.
- A network is a composition of functions.
- We'll make good use of the **chain rule**.

Recall: The Chain Rule

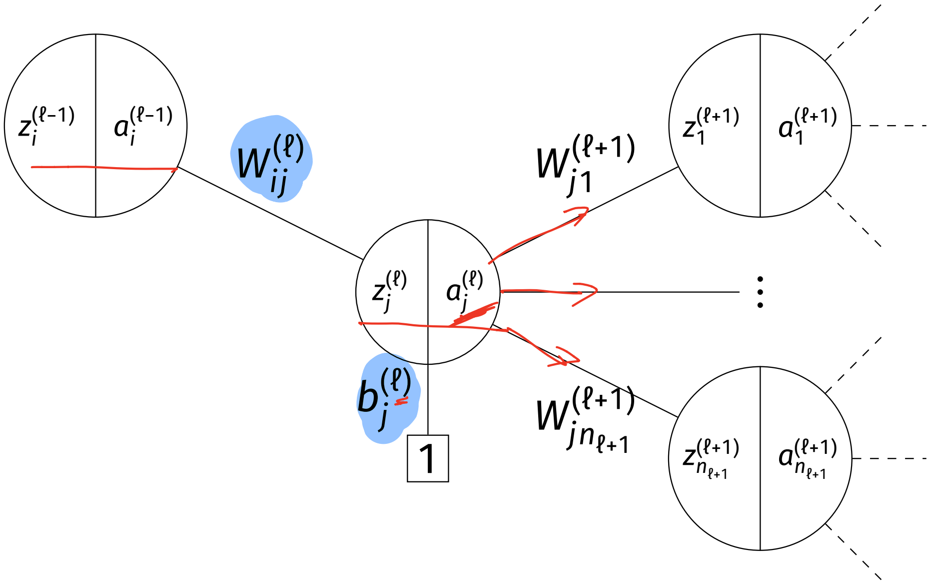
$$\frac{d}{dx} f(g(x)) = \frac{df}{dg}\frac{dg}{dx}$$

$$= f'(g(x))\, g'(x)$$

Some Notation

- We'll consider an arbitrary node in layer ℓ of a neural network.
- Let g be the activation function.
- n_ℓ denotes the number of nodes in layer ℓ.

Arbitrary Node



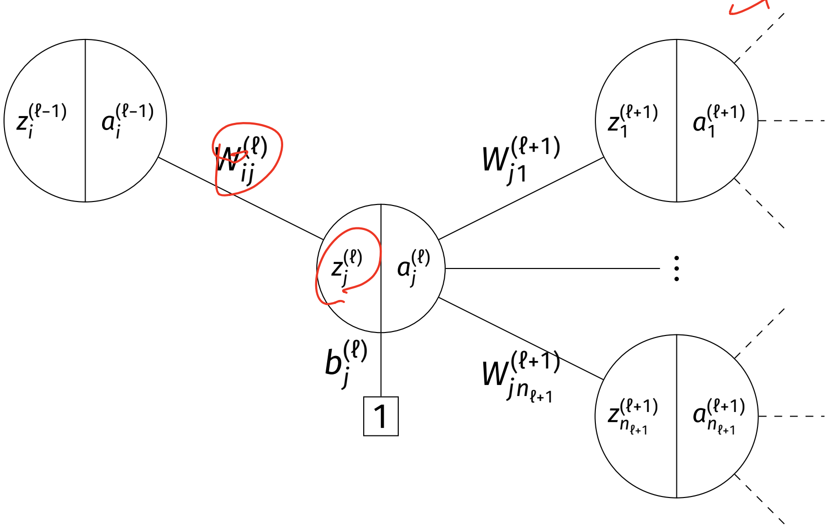
- $\dfrac{\partial H}{\partial W_{ij}^{(\ell)}}$?
- $\dfrac{\partial H}{\partial b_j^{(\ell)}}$?

Claim #1

$$\frac{\partial H}{\partial W_{ij}^{(\ell)}} = \frac{\partial H}{\partial z_j^{(\ell)}} a_i^{(\ell-1)}$$

$$\frac{\partial H}{\partial W_{ij}^{(\ell)}} = \frac{\partial H}{\partial z_j^{(\ell)}} \cdot \frac{\partial z_j^{(\ell)}}{\partial W_{ij}^{(\ell)}}$$

$a_i^{(\ell-1)}$

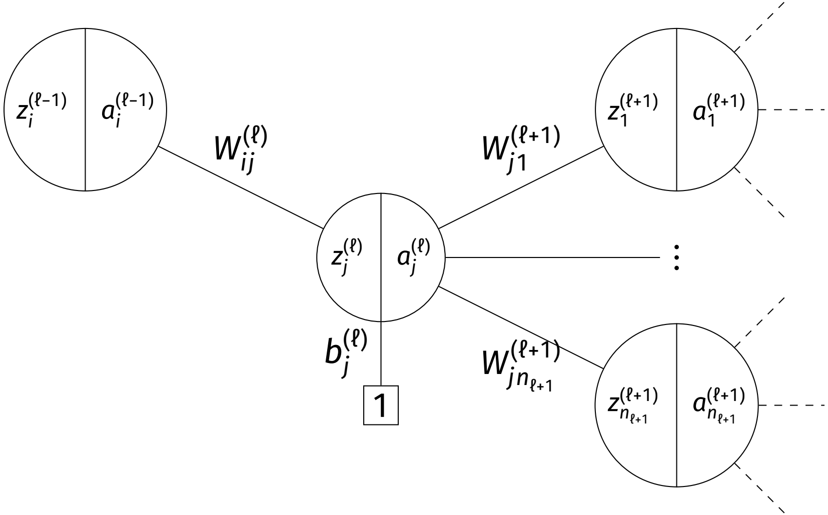


Claim #2

$$\frac{\partial H}{\partial z_j^{(\ell)}} = \frac{\partial H}{\partial a_j^{(\ell)}} g'(z_j^{\ell})$$

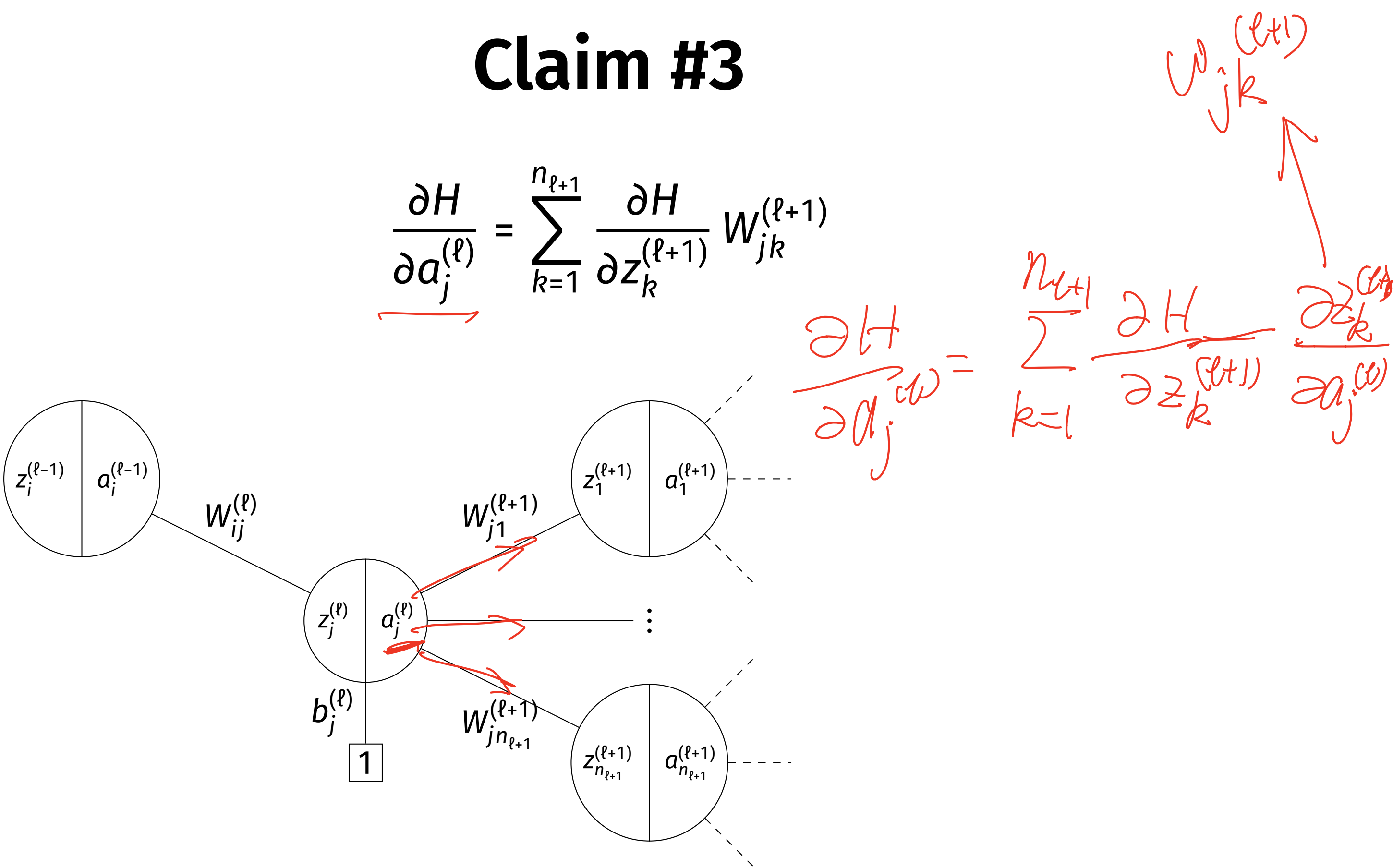
$$\frac{\partial H}{\partial z_j^{(\ell)}} = \frac{\partial H}{\partial a_j^{(\ell)}} \cdot \frac{\partial a_j^{(\ell)}}{\partial z_j^{(\ell)}}$$

$g'(z_j^{(\ell)})$



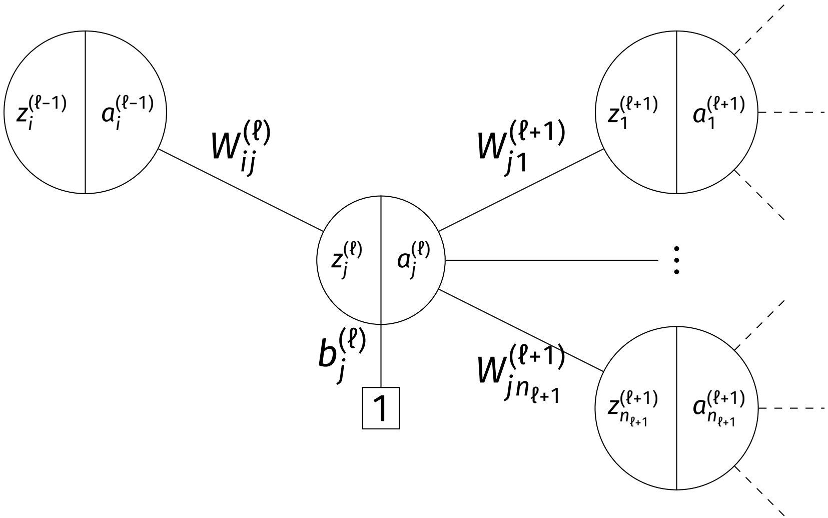
Claim #3

$$\frac{\partial H}{\partial a_j^{(\ell)}} = \sum_{k=1}^{n_{\ell+1}} \frac{\partial H}{\partial z_k^{(\ell+1)}} W_{jk}^{(\ell+1)}$$



Exercise

What is $\partial H / \partial b_j^{(\ell)}$?



General Formulas

- For any node in any neural network[1], we have the following recursive formulas:
  - $\frac{\partial H}{\partial a_j^{(\ell)}} = \sum_{k=1}^{n_{\ell+1}} \frac{\partial H}{\partial z_k^{(\ell+1)}} W_{jk}^{(\ell+1)}$
  - $\frac{\partial H}{\partial z_j^{(\ell)}} = \frac{\partial H}{\partial a_j^{(\ell)}} g'(z_j^{\ell})$
  - $\frac{\partial H}{\partial W_{ij}^{(\ell)}} = \frac{\partial H}{\partial z_j^{(\ell)}} a_i^{(\ell-1)}$
  - $\frac{\partial H}{\partial b_j^{(\ell)}} = \frac{\partial H}{\partial z_j^{(\ell)}}$

[1] Fully-connected, feedforward network

Main Idea

The derivatives in layer ℓ depend on derivatives in layer $\ell + 1$.

Backpropagation

- **Idea:** compute the derivatives in last layers, first.
- That is:
  - Compute derivatives in last layer, ℓ; store them.
  - Use to compute derivatives in layer $\ell - 1$.
  - Use to compute derivatives in layer $\ell - 2$.
  - ...

Backpropagation

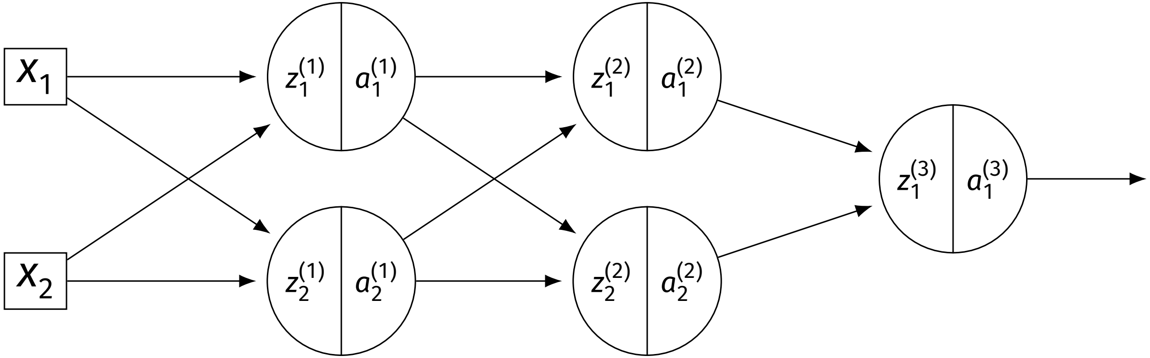
Given an input $\vec{x}$ and a current parameter vector $\vec{w}$:

1. Evaluate the network to compute $z_i^{(\ell)}$ and $a_i^{(\ell)}$ for all nodes.
2. For each layer ℓ from last to first:
   - Compute $\frac{\partial H}{\partial a_j^{(\ell)}} = \sum_{k=1}^{n_{\ell+1}} \frac{\partial H}{\partial z_k^{(\ell+1)}} W_{jk}^{(\ell+1)}$
   - Compute $\frac{\partial H}{\partial z_j^{(\ell)}} = \frac{\partial H}{\partial a_j^{(\ell)}} g'(z_j^{\ell})$
   - Compute $\frac{\partial H}{\partial W_{ij}^{(\ell)}} = \frac{\partial H}{\partial z_j^{(\ell)}} a_i^{(\ell-1)}$
   - Compute $\frac{\partial H}{\partial b_j^{(\ell)}} = \frac{\partial H}{\partial z_j^{(\ell)}}$

Example

Compute the entries of the gradient given:

$$W^{(1)} = \begin{pmatrix} 2 & -3 \\ 2 & 1 \end{pmatrix} \quad W^{(2)} = \begin{pmatrix} 2 & 1 \\ 0 & 1 \end{pmatrix} \quad W^{(3)} = \begin{pmatrix} 3 \\ -2 \end{pmatrix} \quad \vec{x} = (2, 1)^T \quad g(z) = \text{ReLU}$$

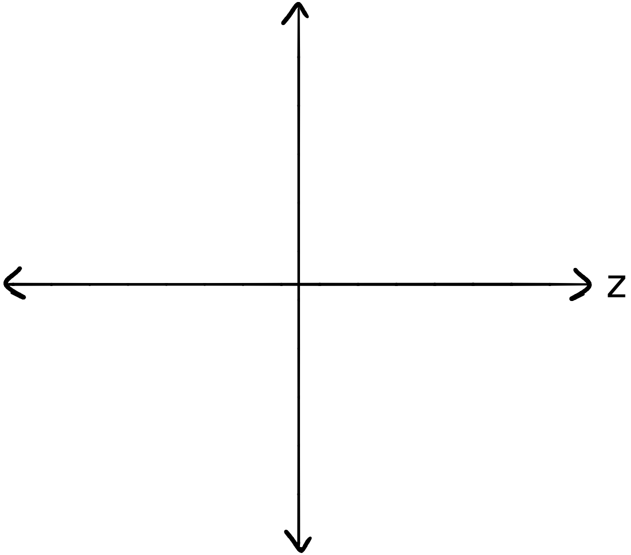


$$\frac{\partial H}{\partial a_j^{(\ell)}} = \sum_{k=1}^{n_{\ell+1}} \frac{\partial H}{\partial z_k^{(\ell+1)}} W_{jk}^{(\ell+1)} \qquad \frac{\partial H}{\partial z_j^{(\ell)}} = \frac{\partial H}{\partial a_j^{(\ell)}} g'(z_j^{\ell}) \qquad \frac{\partial H}{\partial W_{ij}^{(\ell)}} = \frac{\partial H}{\partial z_j^{(\ell)}} a_i^{(\ell-1)}$$

Aside: Derivative of ReLU

$$g(z) = \max\{0, z\}$$

$$g'(z) = \begin{cases} 0, & z < 0 \\ 1, & z > 0 \end{cases}$$



Summary: Backprop

- **Backprop** is an algorithm for efficiently computing the gradient of a neural network
- It is not an algorithm **you** need to carry out by hand: your NN library can do it for you.