

DSC 140B

Representation Learning

Lecture 13 | Part 1

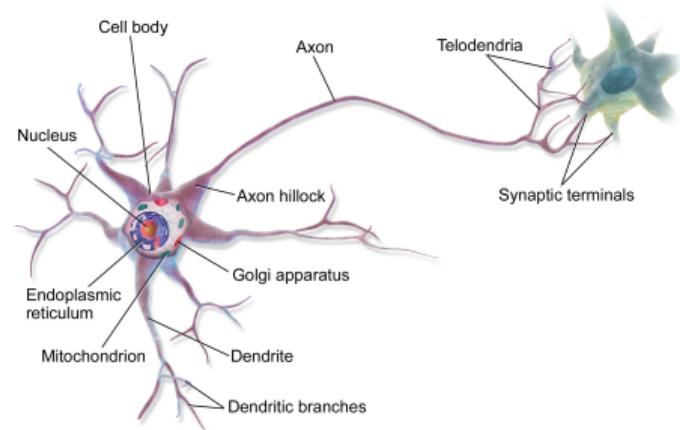
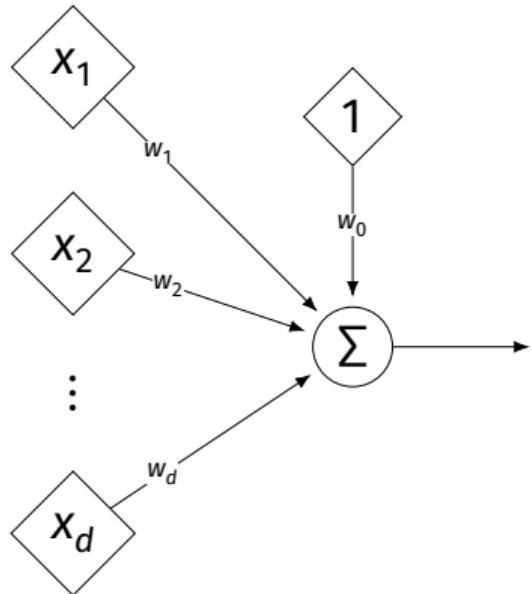
Neural Networks

Beyond RBFs

- ▶ When training RBFs, we fixed the basis functions *before* training the weights.
- ▶ Representation learning was decoupled from learning the prediction function.
- ▶ **Now:** learn representation **and** prediction function together.

Linear Models

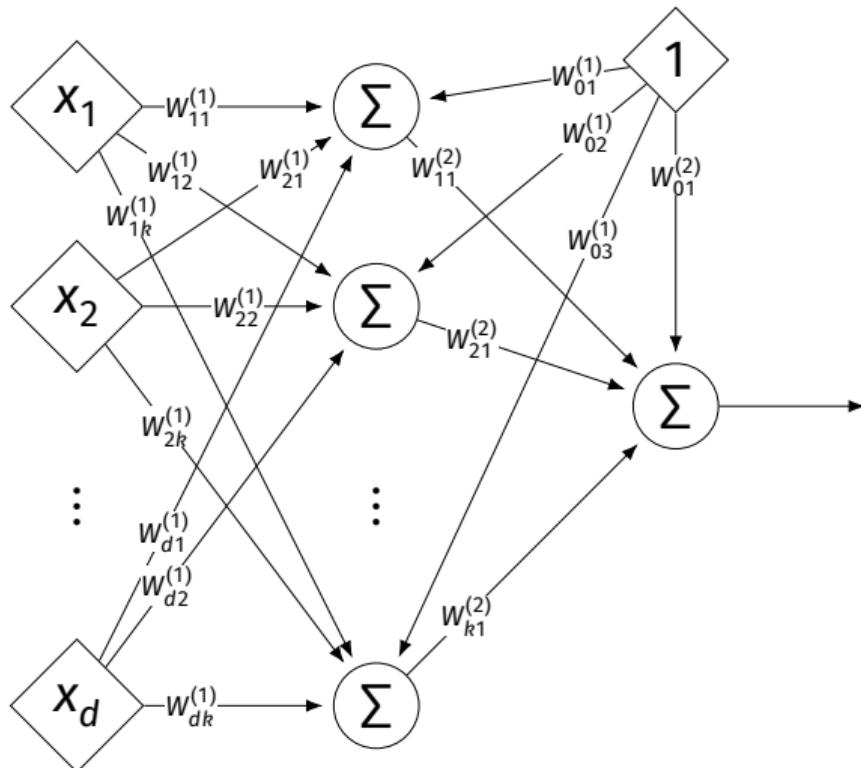
$$H(\vec{x}) = w_0 + w_1x_1 + \dots + w_dx_d$$



Generalizing Linear Models

- ▶ The brain is a **network** of neurons.
- ▶ The output of a neuron is used as an input to another.
- ▶ **Idea:** chain together multiple “neurons” into a **neural network**.

Neural Network¹ (One Hidden Layer)



¹Specifically, a fully-connected, feed-forward neural network

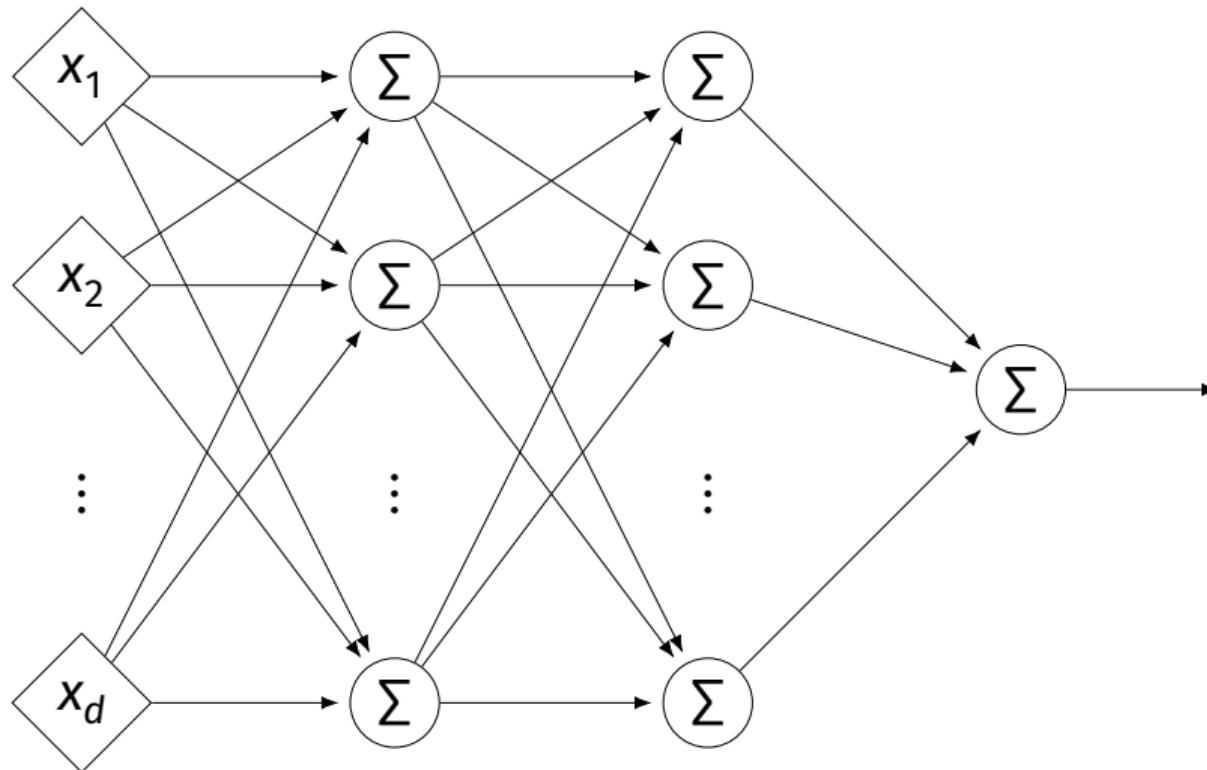
Architecture

- ▶ Neurons are organized into **layers**.
 - ▶ **Input layer**, **output layer**, and **hidden layers**.
- ▶ Number of cells in input layer determined by dimensionality of input feature vectors.
- ▶ Number of cells in hidden layer(s) is determined by you.
- ▶ Output layer can have >1 neuron.

Architecture

- ▶ Can have more than one hidden layer.
 - ▶ A network is “**deep**” if it has >1 hidden layer.
- ▶ Hidden layers can have different number of neurons.

Neural Network (Two Hidden Layers)

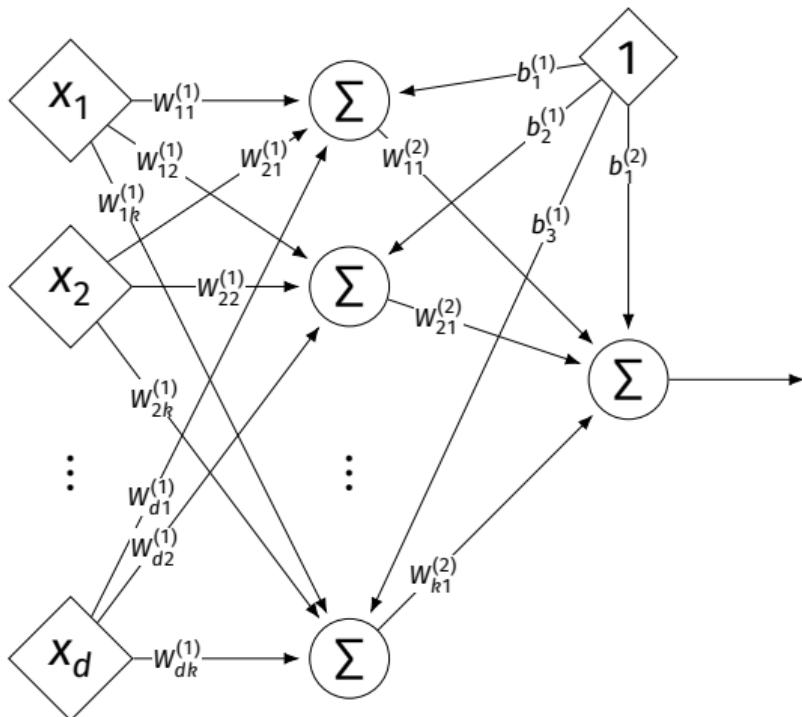


Network Weights

- ▶ A neural network is a type of function.
- ▶ Like a linear model, a NN is **totally determined** by its weights.
- ▶ But there are often many more weights to learn!

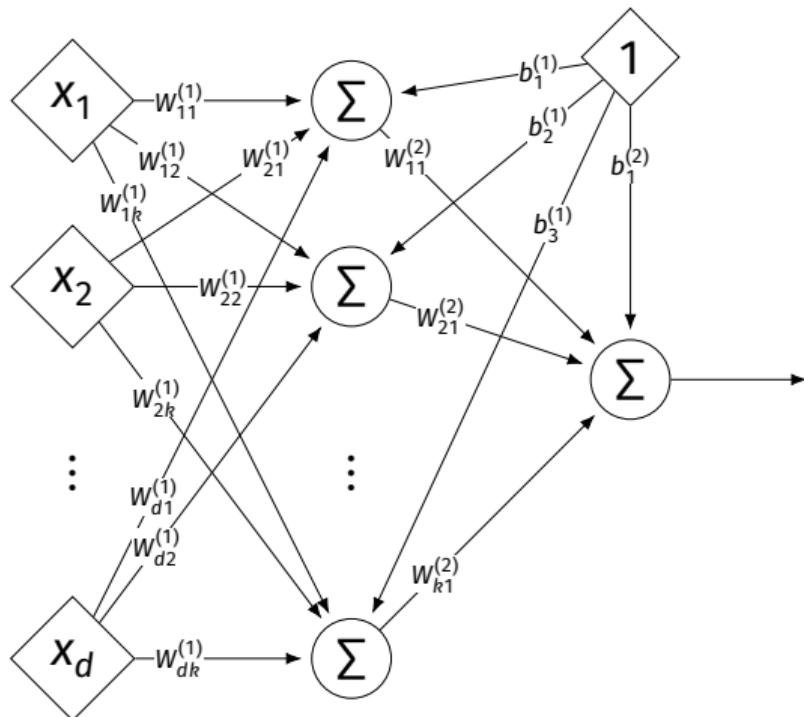
Notation

- ▶ Input is layer #0.
- ▶ $W_{jk}^{(i)}$ denotes weight of connection between neuron j in layer $(i - 1)$ and neuron k in layer i
- ▶ Layer weights are 2-d arrays.



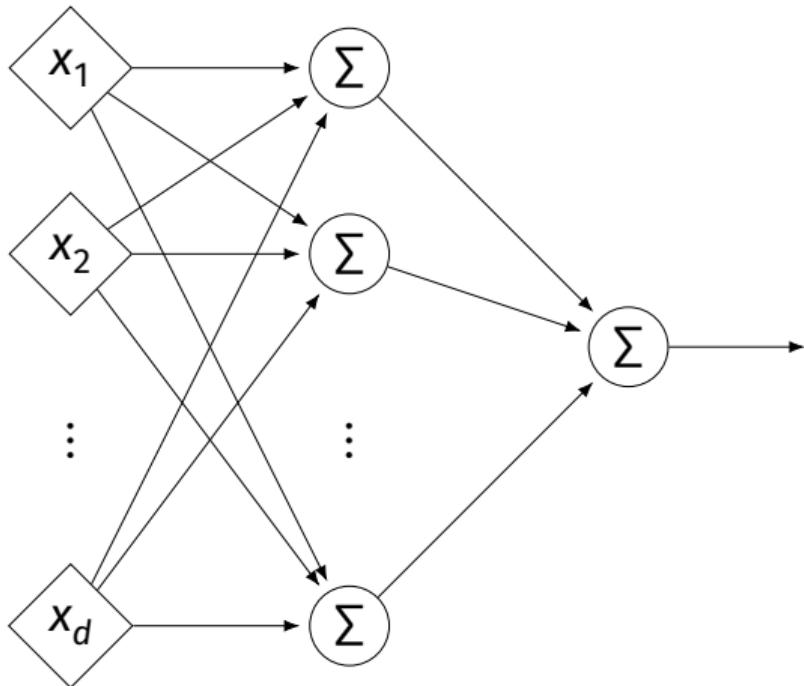
Notation

- ▶ Each hidden/output neuron gets a “dummy” input of 1.
- ▶ j th node in i th layer assigned a bias weight of $b_j^{(i)}$
- ▶ Biases for layer are a vector: $\vec{b}^{(i)}$

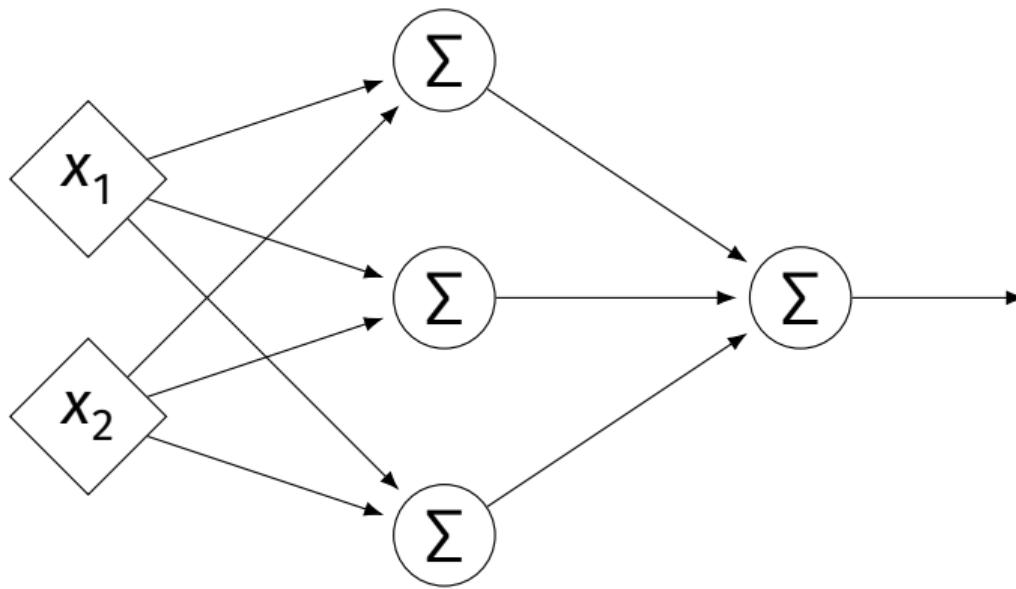


Notation

- ▶ Typically, we will not draw the weights.
- ▶ We will not draw the dummy input, too, but it is there.



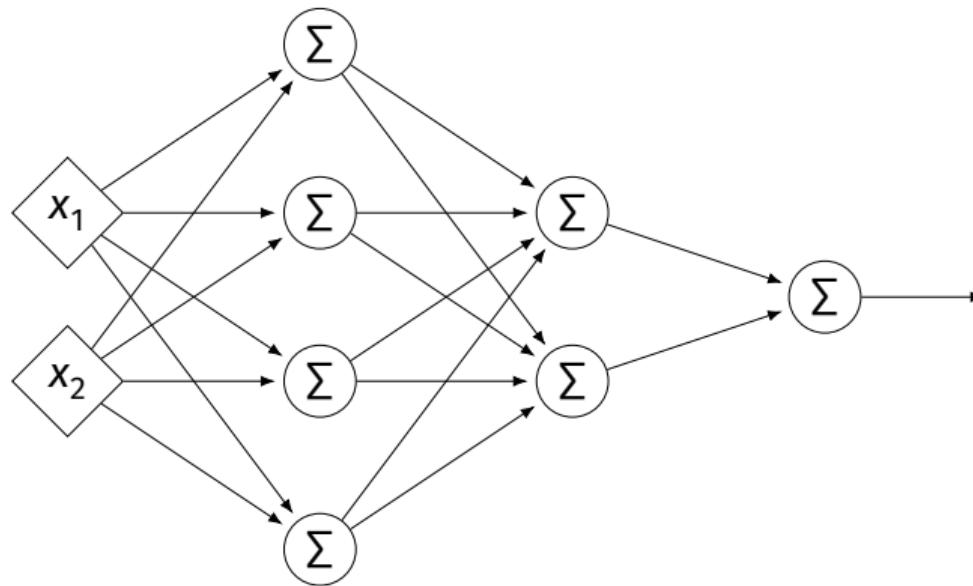
Example



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

$$\vec{b}^{(1)} = (3, -2, -2)^T \quad \vec{b}^{(2)} = (-4)^T$$

Example



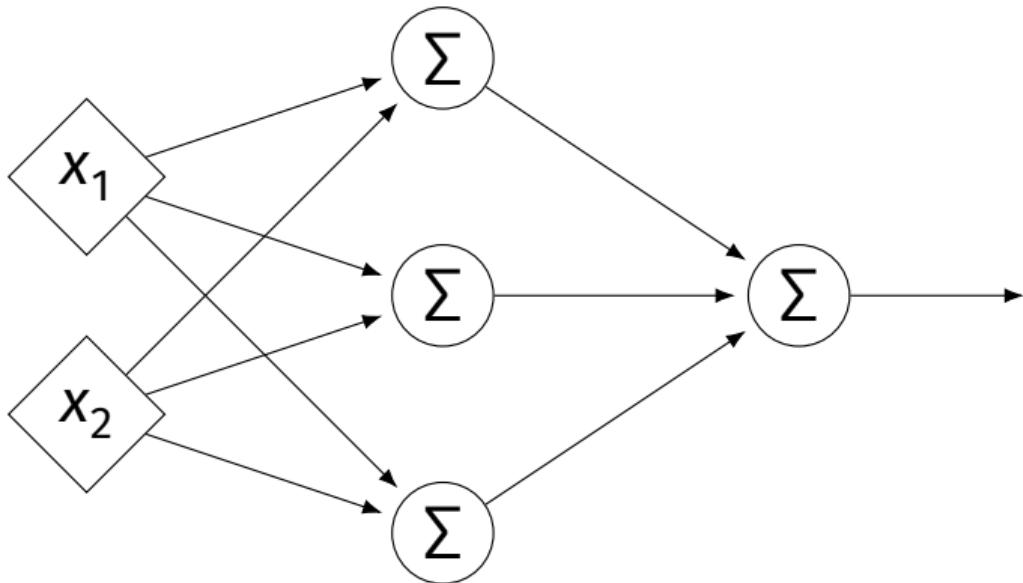
$$W^{(1)} = \begin{pmatrix} 2 & -1 & -3 & 0 \\ 4 & 5 & -7 & 2 \end{pmatrix} \quad W^{(2)} = \begin{pmatrix} 1 & 2 \\ -4 & 3 \\ -6 & -2 \\ 3 & 4 \end{pmatrix} \quad W^{(3)} = \begin{pmatrix} -1 & 5 \end{pmatrix}$$

$$\vec{b}^{(1)} = (3, 6, -2, -2)^T \quad \vec{b}^{(2)} = (-4, 0)^T \quad \vec{b}^{(3)} = (1)^T$$

Evaluation

- ▶ These are “**fully-connected, feed-forward**” networks with one output.
- ▶ They are functions $H(\vec{x}) : \mathbb{R}^d \rightarrow \mathbb{R}^1$
- ▶ To evaluate $H(\vec{x})$, compute result of layer i , use as inputs for layer $i + 1$.

Example



► $\vec{x} = (3, -1)^T$

► $z_1^{(1)} =$

► $z_2^{(1)} =$

► $z_3^{(1)} =$

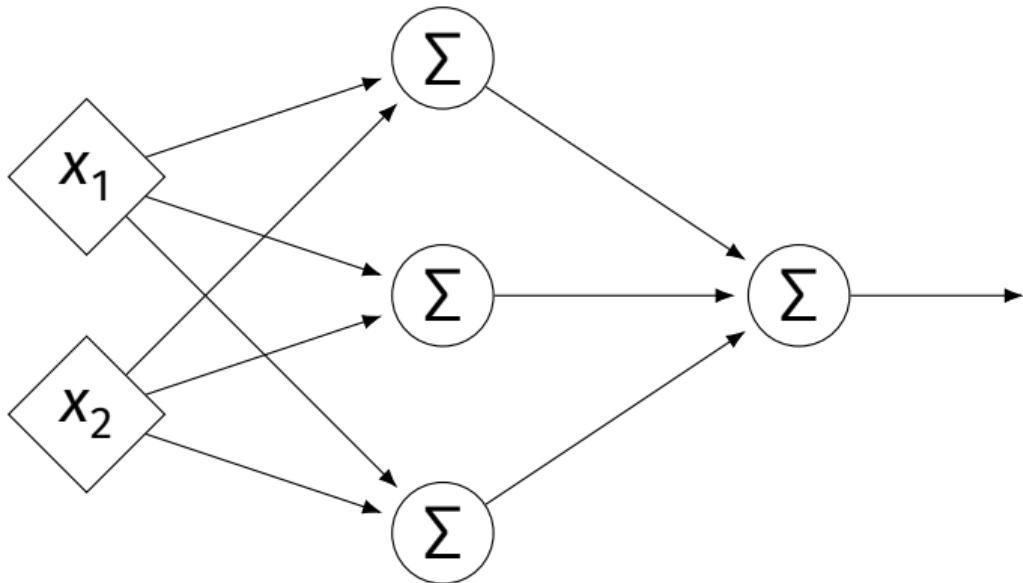
► $z_1^{(2)} =$

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

Evaluation as Matrix Multiplication

- ▶ Let $z_j^{(i)}$ be the output of node j in layer i .
- ▶ Make a vector of these outputs: $\vec{z}^{(i)} = (z_1^{(i)}, z_2^{(i)}, \dots)^T$
- ▶ Observe that $\vec{z}^{(i)} = [W^{(i)}]^T \vec{z}^{(i-1)} + \vec{b}^{(i)}$

Example



► $\vec{x} = (3, -1)^T$

► $z_1^{(1)} =$

► $z_2^{(1)} =$

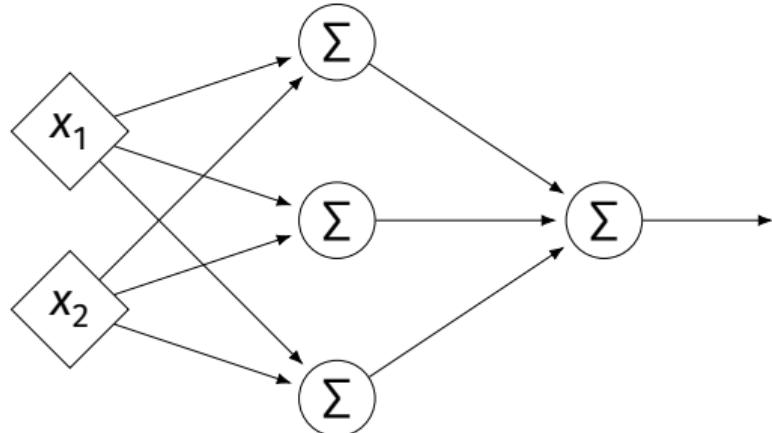
► $z_3^{(1)} =$

► $z_1^{(2)} =$

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

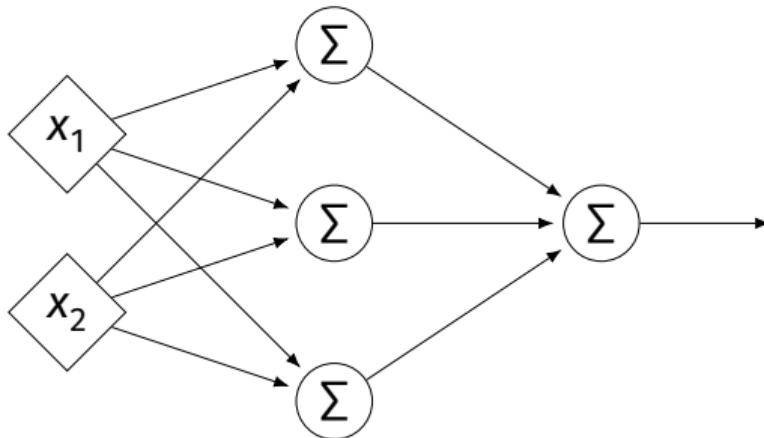
Each Layer is a Function

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



NNs as Function Composition

- The full NN is a composition of layer functions.



$$H(\vec{x}) = H^{(2)}(H^{(1)}(\vec{x})) = \underbrace{\left[W^{(2)} \right]^T \left(\left[W^{(1)} \right]^T \vec{x} + \vec{b}^{(1)} \right) + \vec{b}^{(2)}}_{\vec{z}^{(1)}}$$

NNs as Function Composition

- ▶ In general, if there k hidden layers:

$$H(\vec{x}) = H^{(k+1)} \left(\dots H^{(3)} \left(H^{(2)} \left(H^{(1)}(\vec{x}) \right) \right) \dots \right)$$

Exercise

Show that:

$$H(\vec{x}) = [W^{(2)}]^T \left([W^{(1)}]^T \vec{x} + \vec{b}^{(1)} \right) + \vec{b}^{(2)} = \vec{w} \cdot \text{Aug}(\vec{x})$$

for some appropriately-defined vector \vec{w} .

Result

- ▶ The composition of linear functions is again a linear function.
- ▶ The NNs we have seen so far are all equivalent to linear models!
- ▶ For NNs to be more useful, we will need to add **non-linearity**.

Activations

- ▶ So far, the output of a neuron has been a linear function of its inputs:

$$w_0 + w_1 x_1 + w_2 x_2 + \dots$$

- ▶ Can be arbitrarily large or small.
- ▶ But real neurons are **activated** non-linearly.
 - ▶ E.g., saturation.

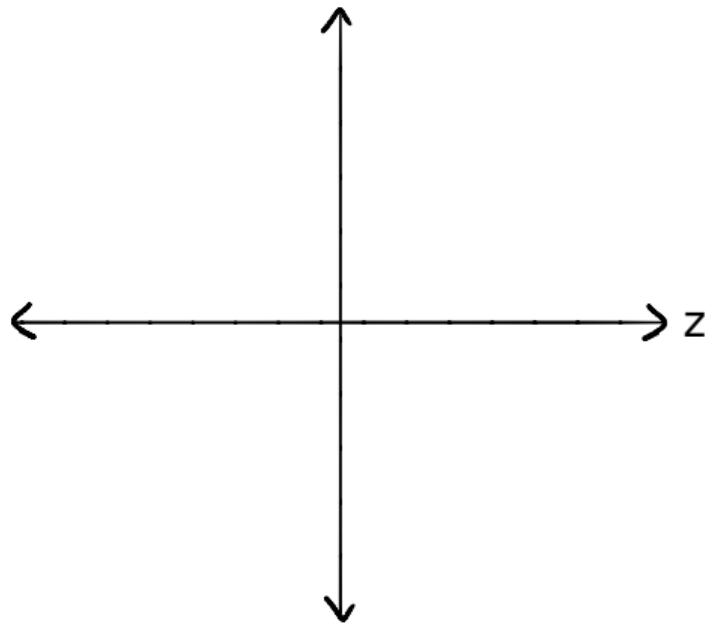
Idea

- ▶ To add nonlinearity, we will apply a non-linear **activation function** g to the output of **each** hidden neuron (and sometimes the output neuron).

Linear Activation

- ▶ The **linear** activation is what we've been using.

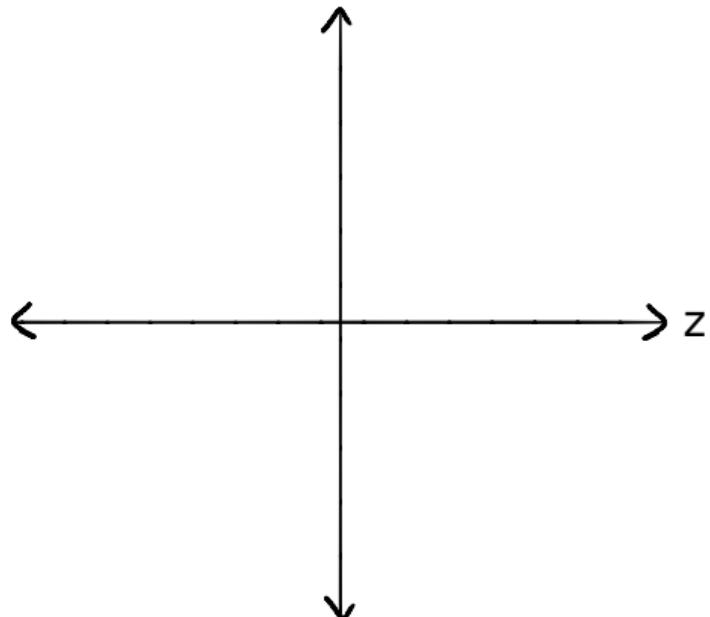
$$\sigma(z) = z$$



Sigmoid Activation

- The **sigmoid** models saturation in many natural processes.

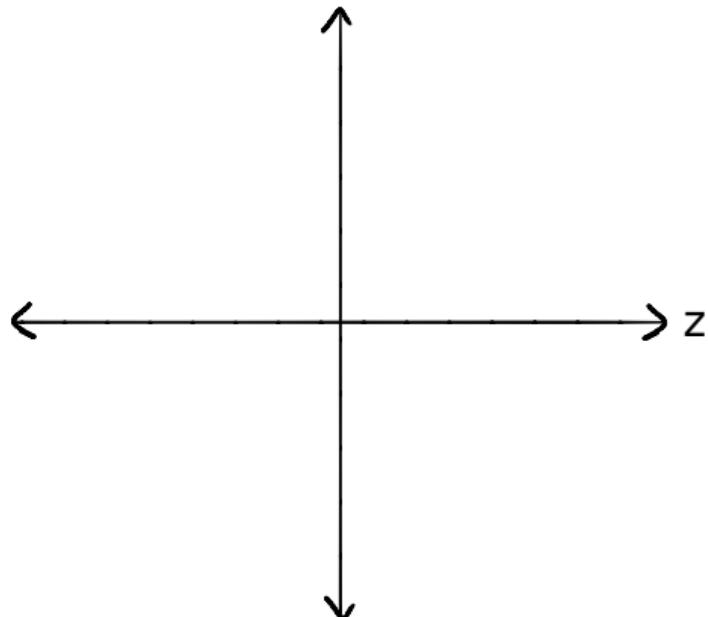
$$\sigma(z) = \frac{1}{1 + e^{-z}}$$



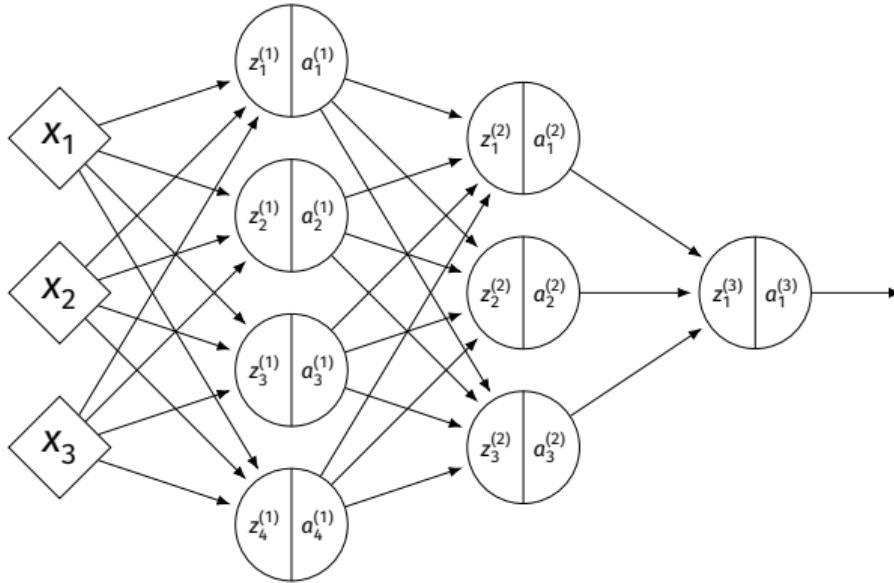
ReLU Activation

- The **Rectified Linear Unit (ReLU)** tends to work better in practice.

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

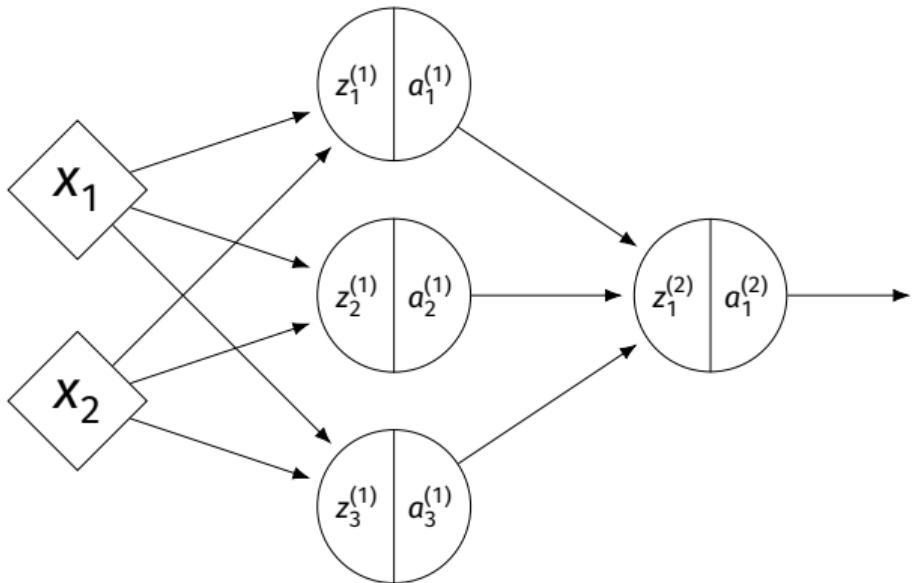


Notation



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

Example



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

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

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 13 | 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).

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

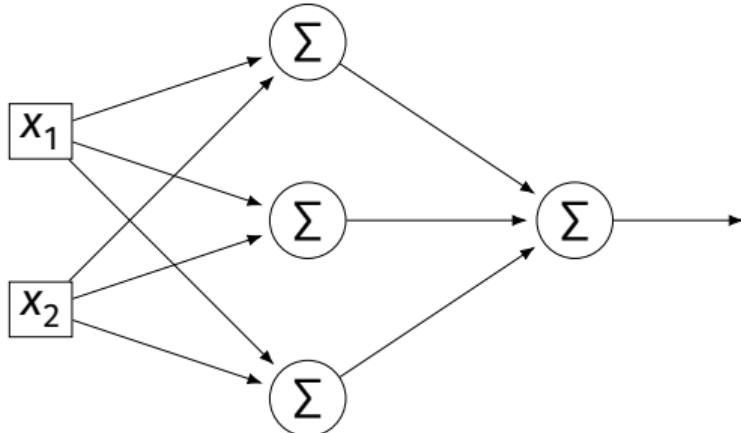
- ▶ These basis functions are fixed **before** learning.
- ▶ **Downside:** we have to choose $\vec{\phi}$ somehow.

Learning a Feature Map

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

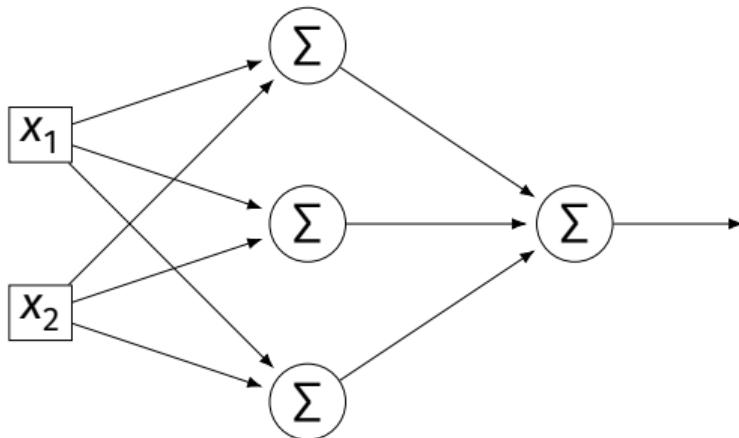
Each Layer is a Function

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



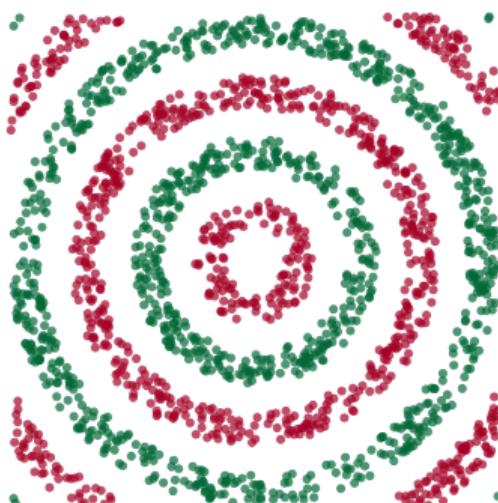
Each Layer is a Function

- ▶ 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”.



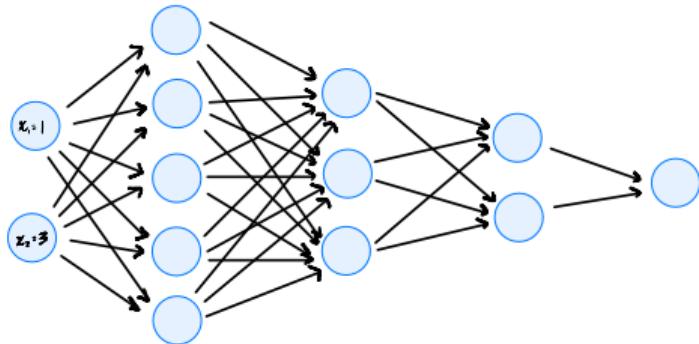
Demo

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

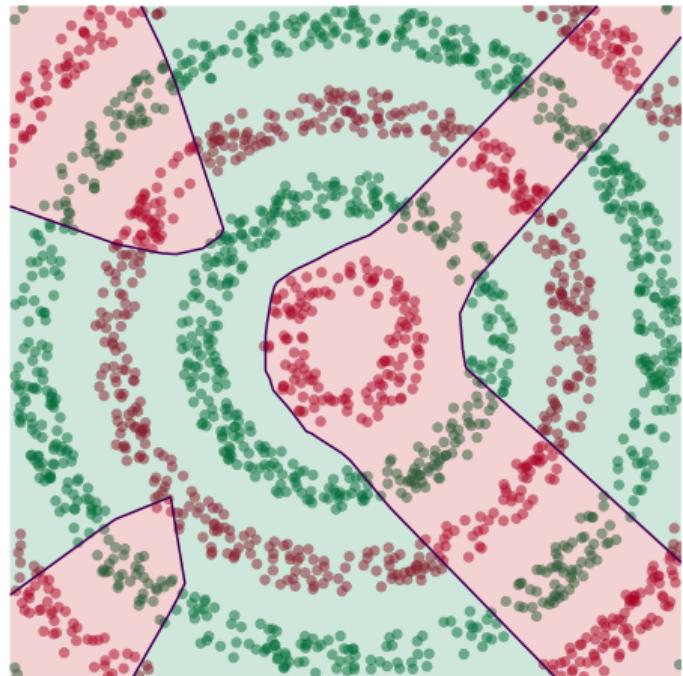


Demo

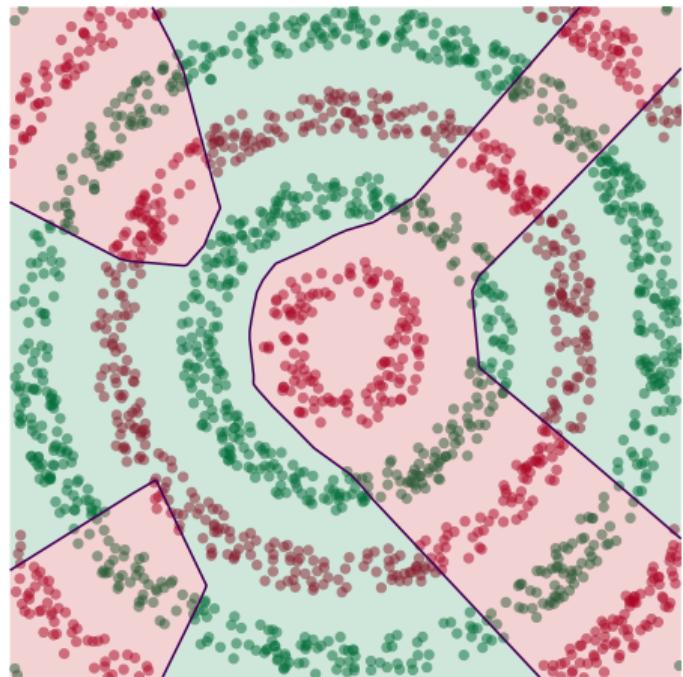
- ▶ We'll use three hidden layers, with last having two neurons.
- ▶ We can see this new representation!
- ▶ Plug in \vec{x} and see activations of last hidden layer.



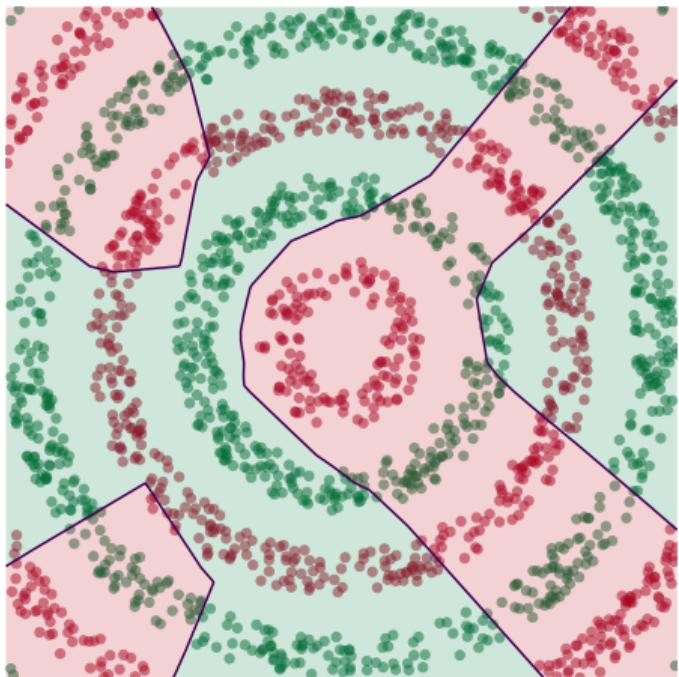
Learning a New Representation



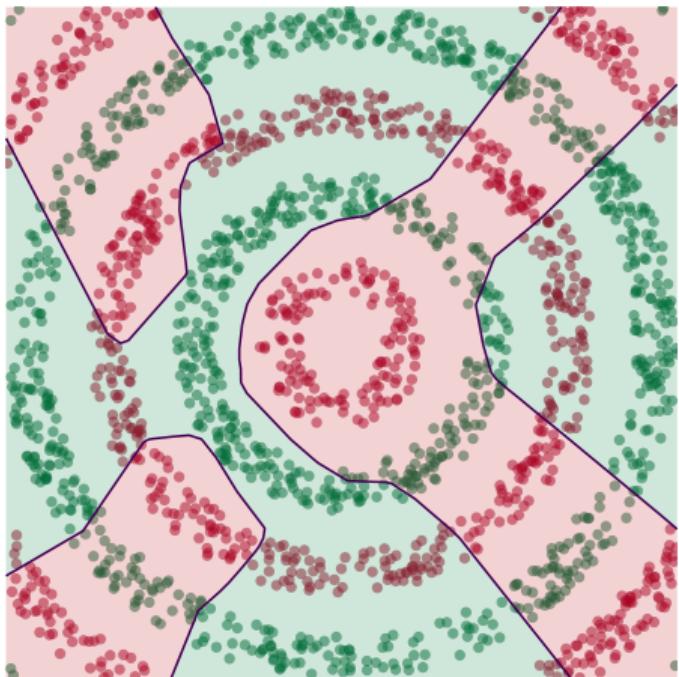
Learning a New Representation



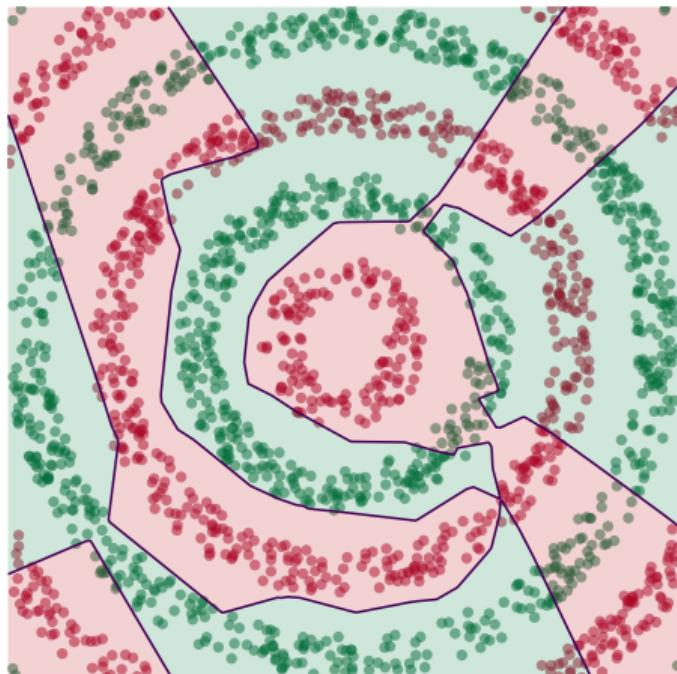
Learning a New Representation



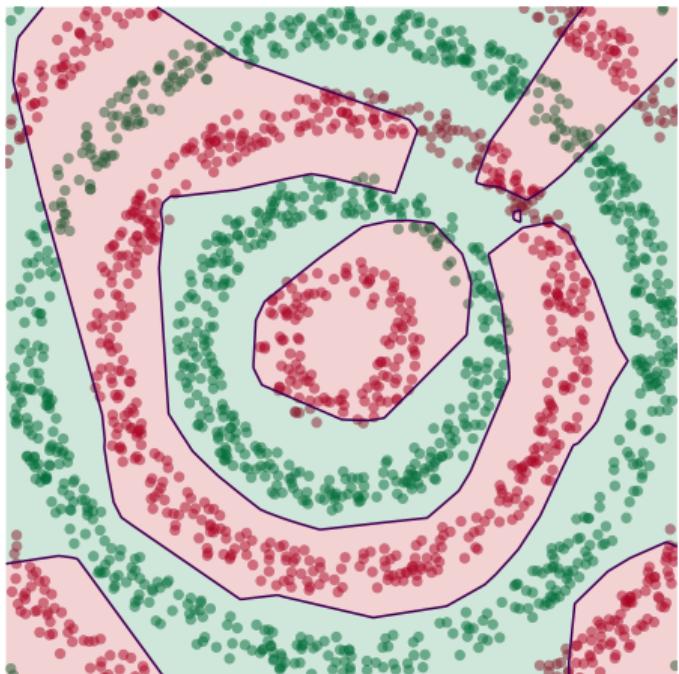
Learning a New Representation



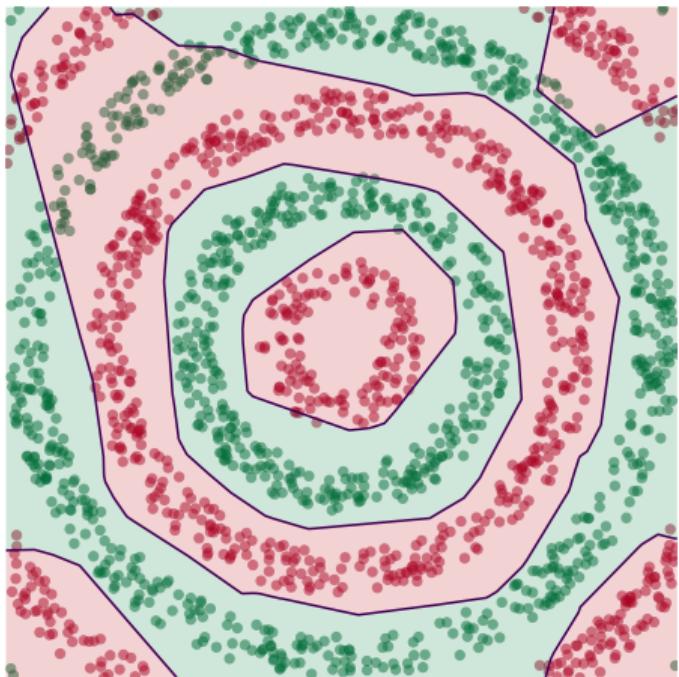
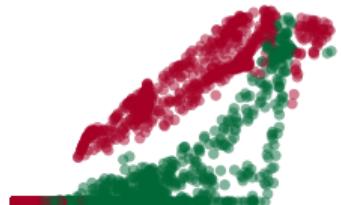
Learning a New Representation



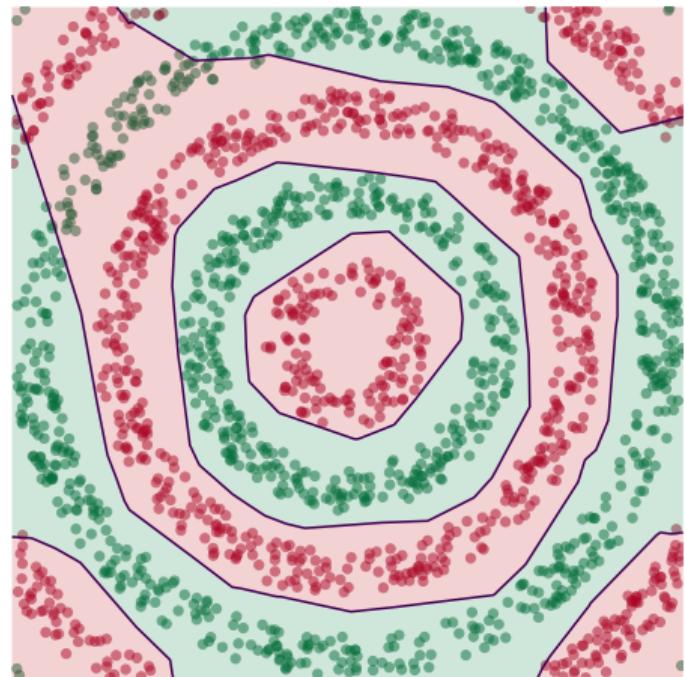
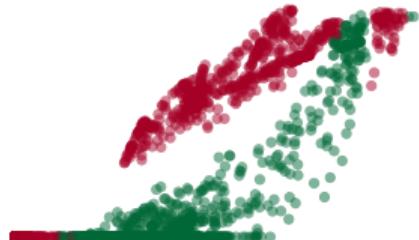
Learning a New Representation



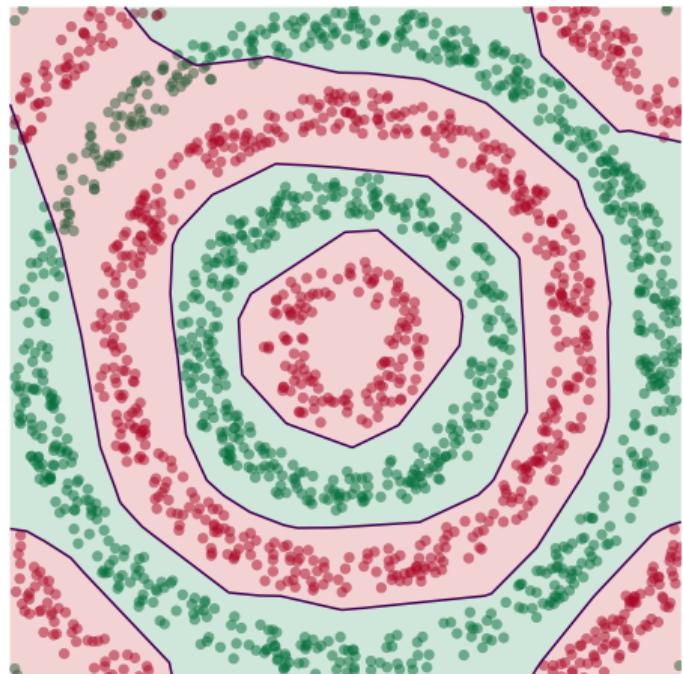
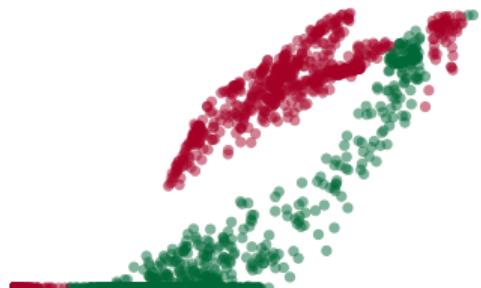
Learning a New Representation



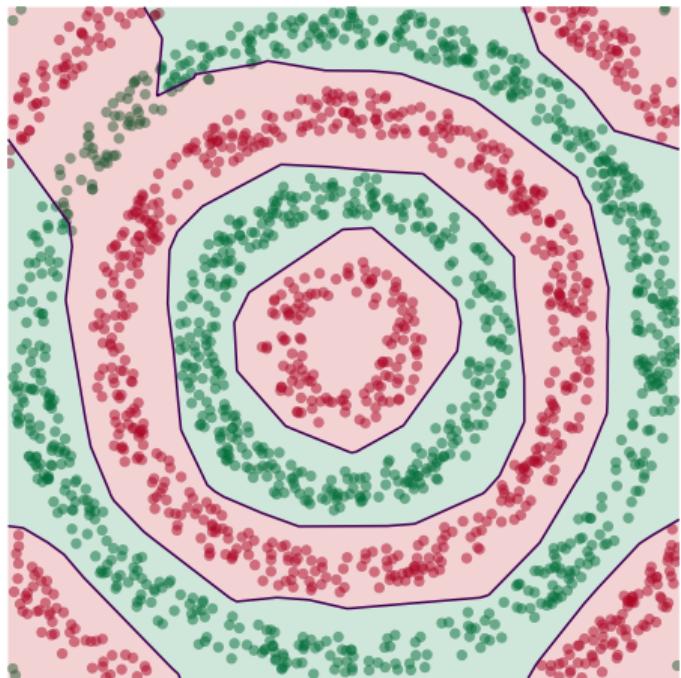
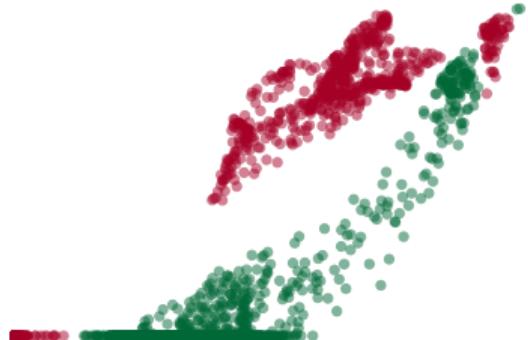
Learning a New Representation



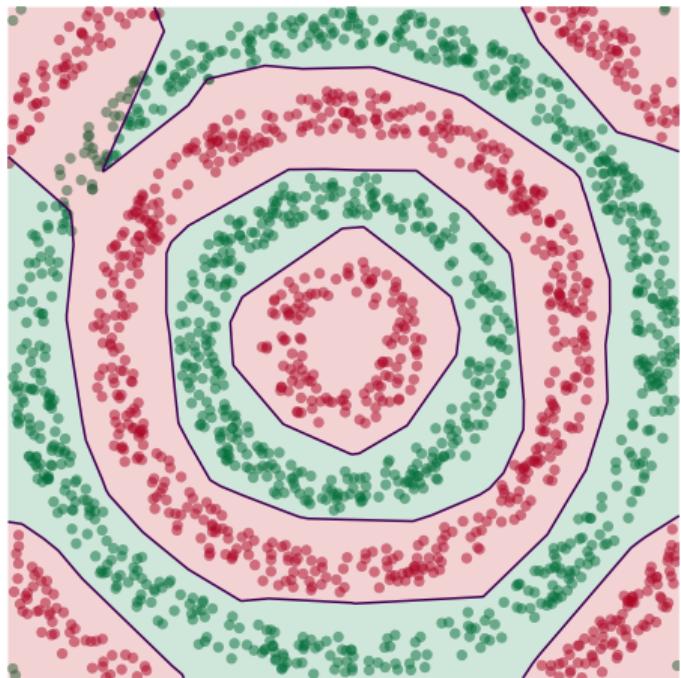
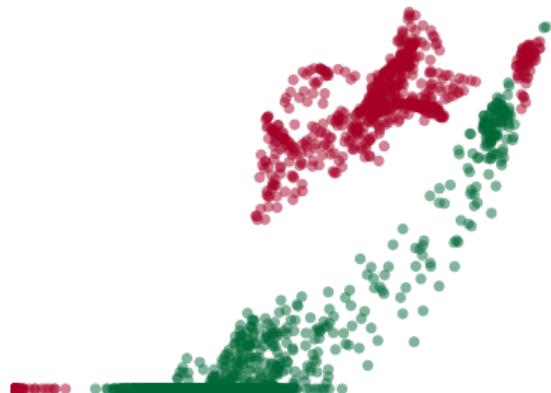
Learning a New Representation



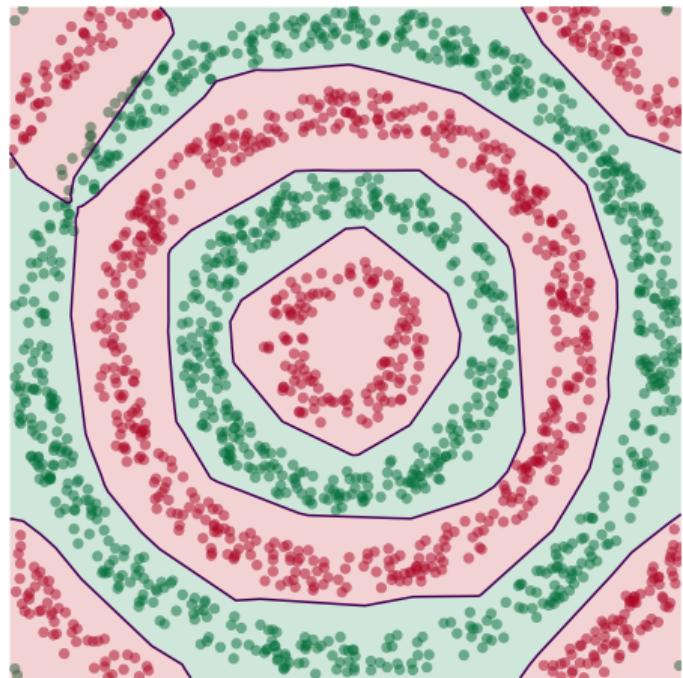
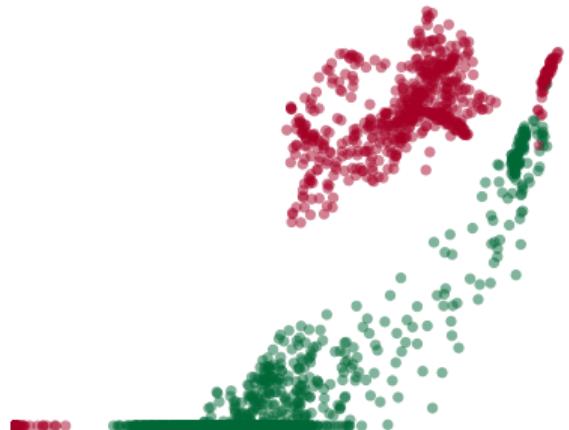
Learning a New Representation



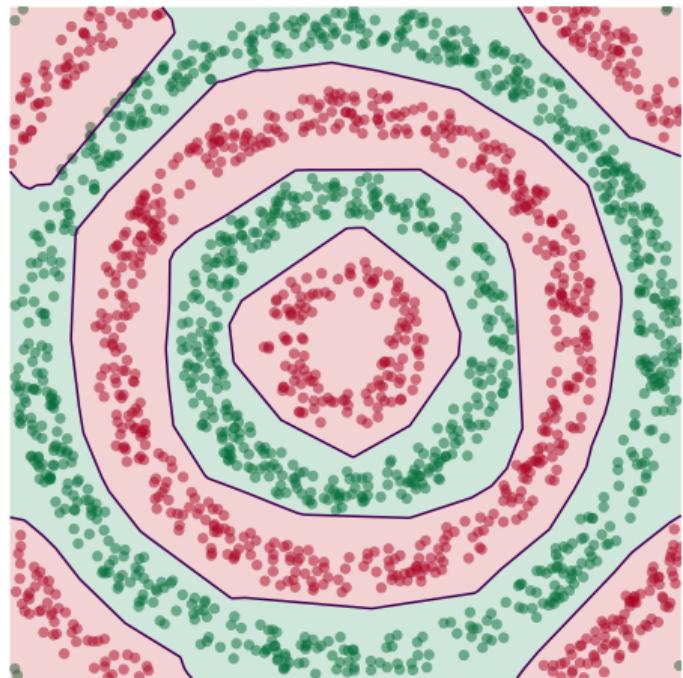
Learning a New Representation



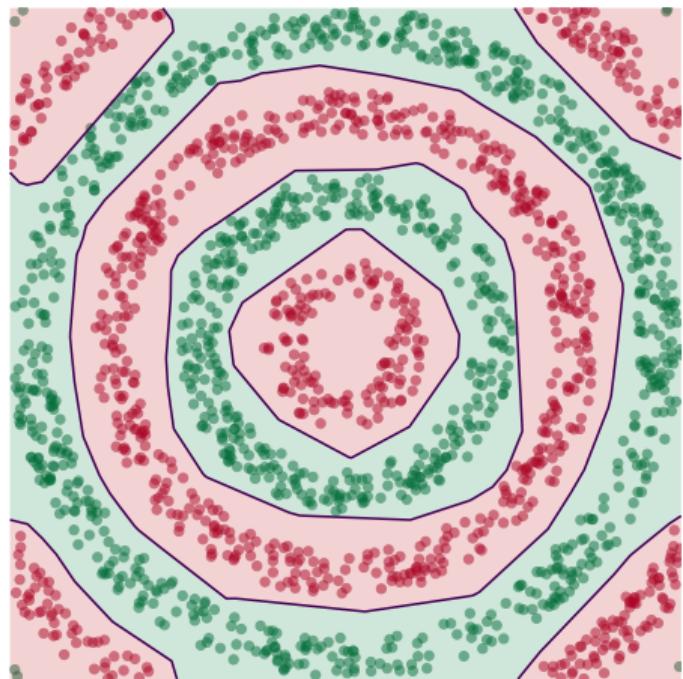
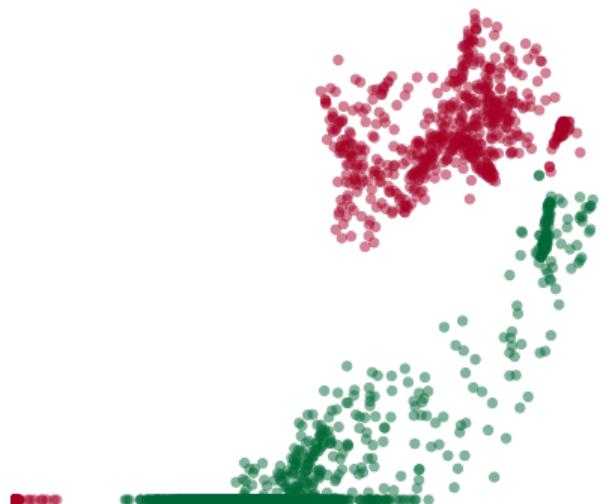
Learning a New Representation



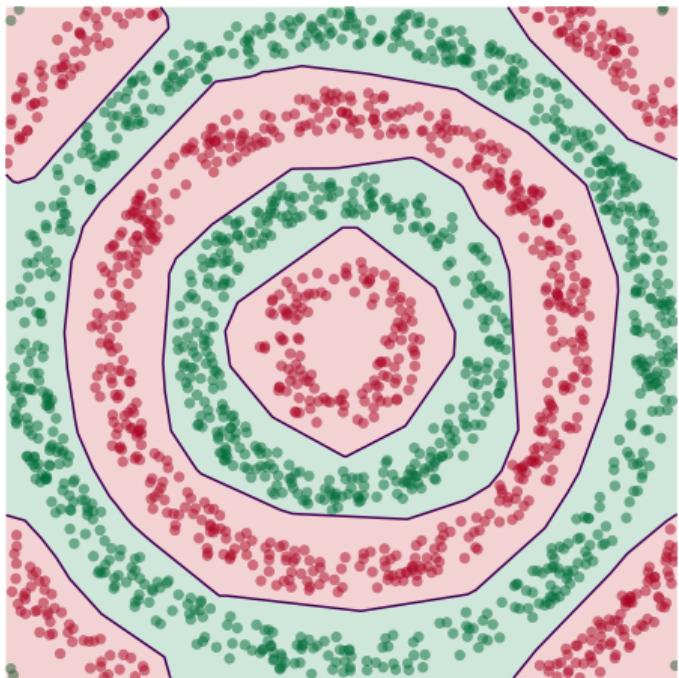
Learning a New Representation



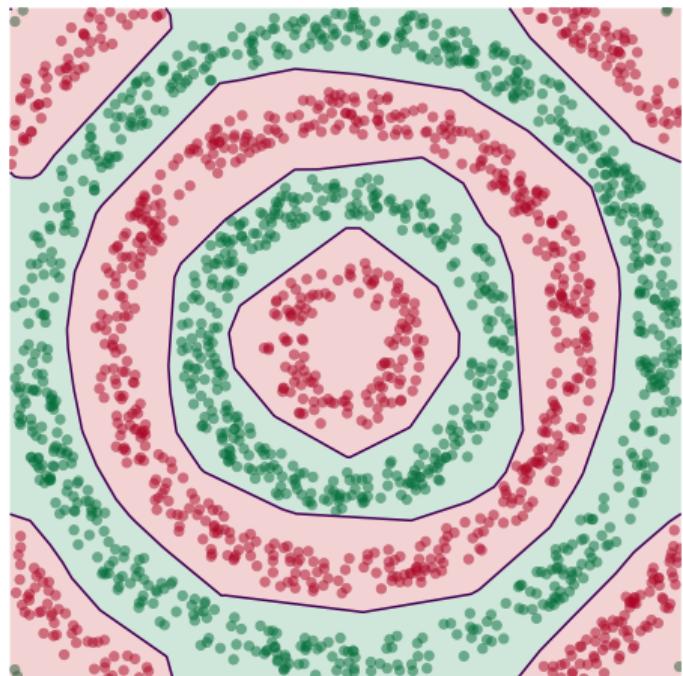
Learning a New Representation



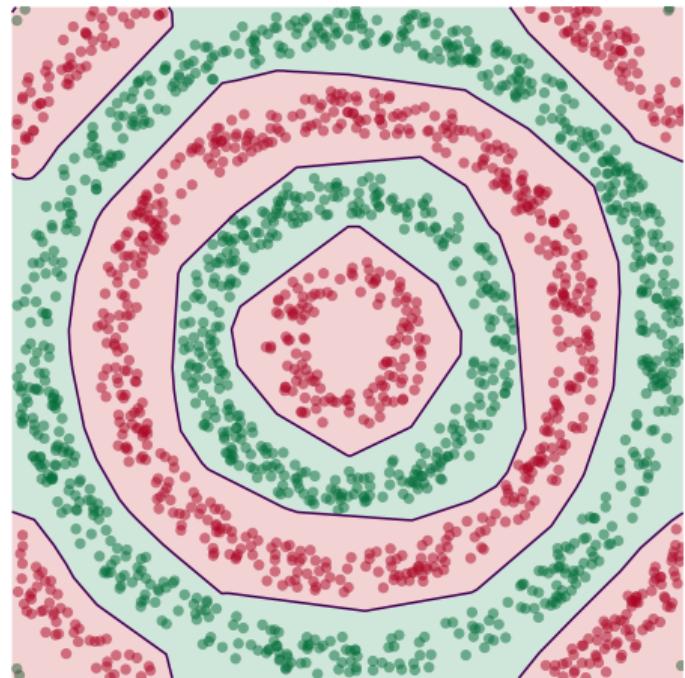
Learning a New Representation



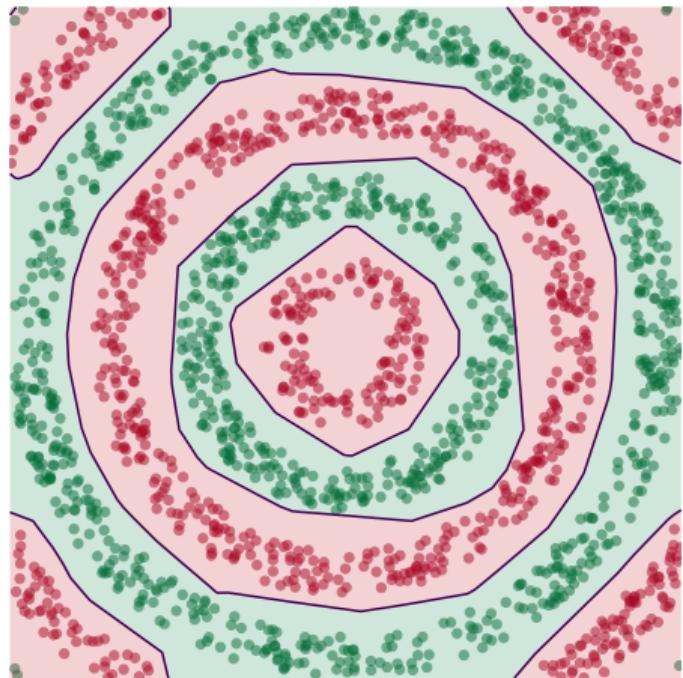
Learning a New Representation



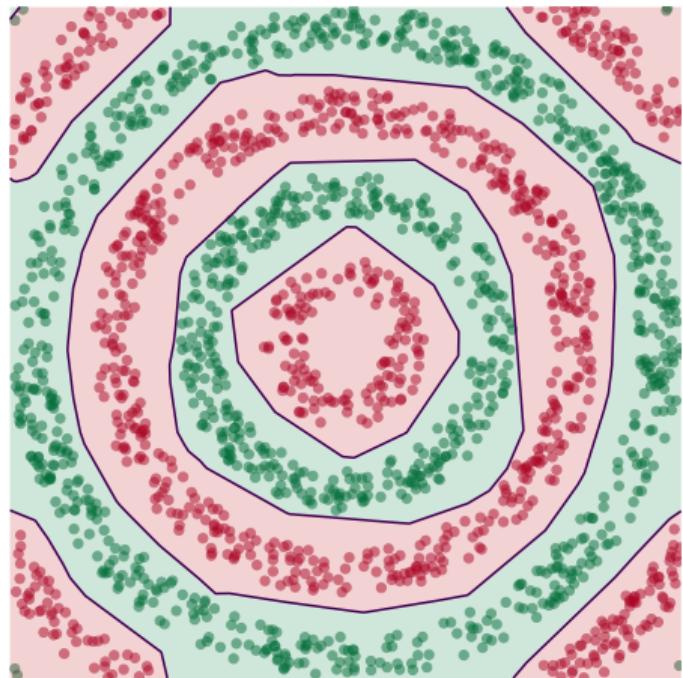
Learning a New Representation



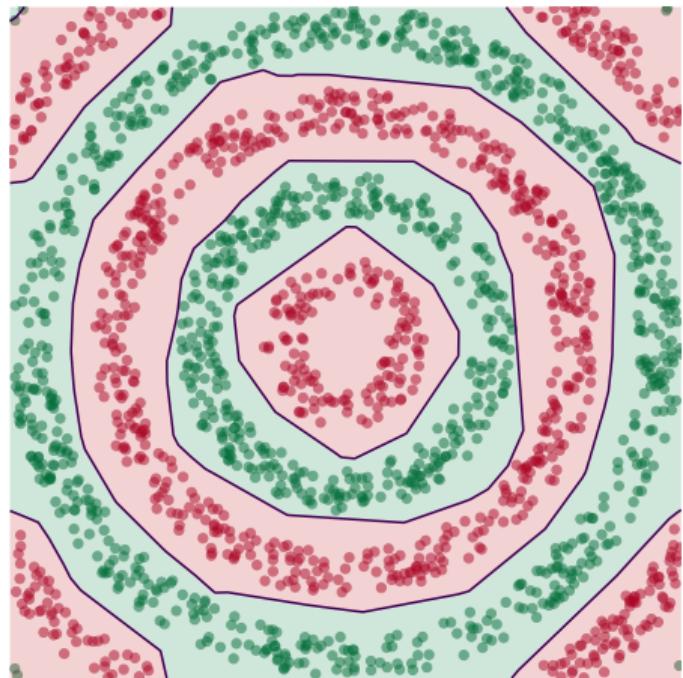
Learning a New Representation



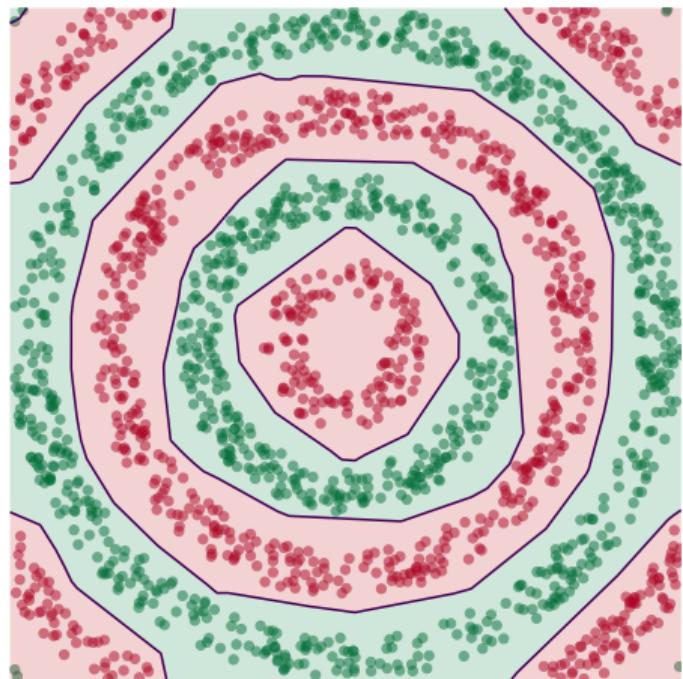
Learning a New Representation



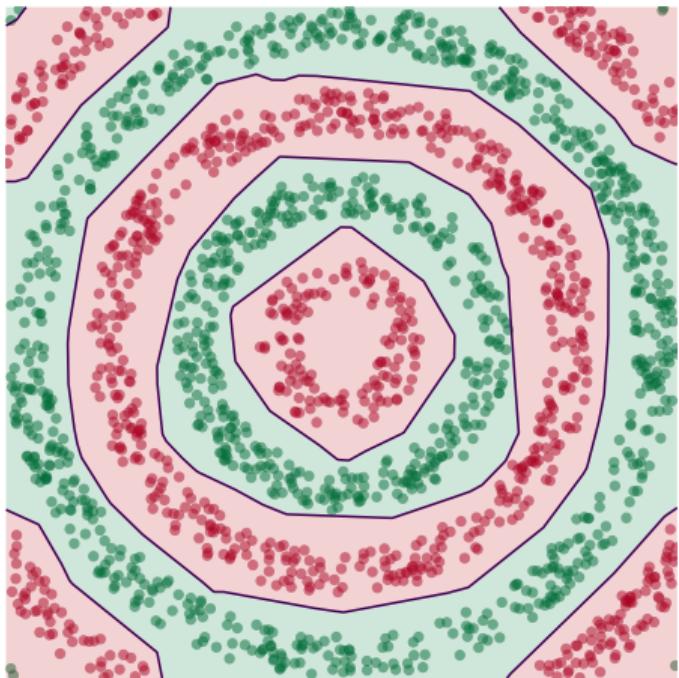
Learning a New Representation



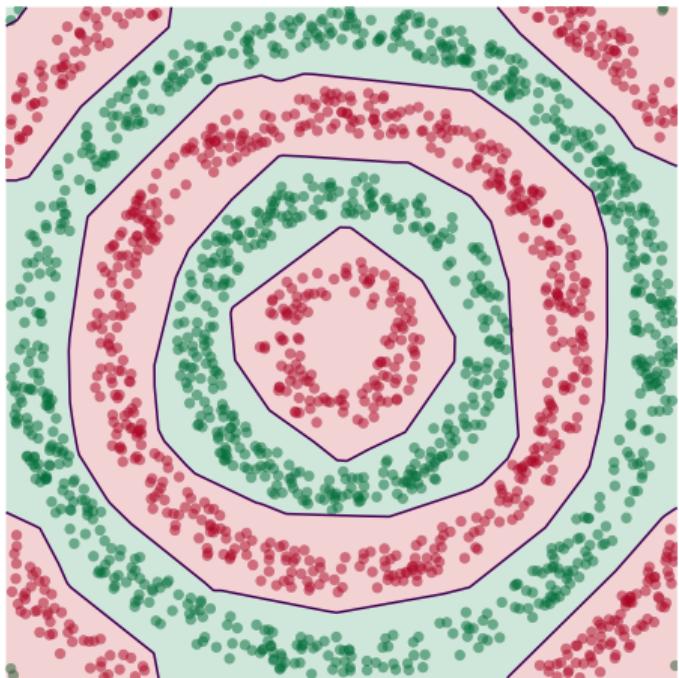
Learning a New Representation



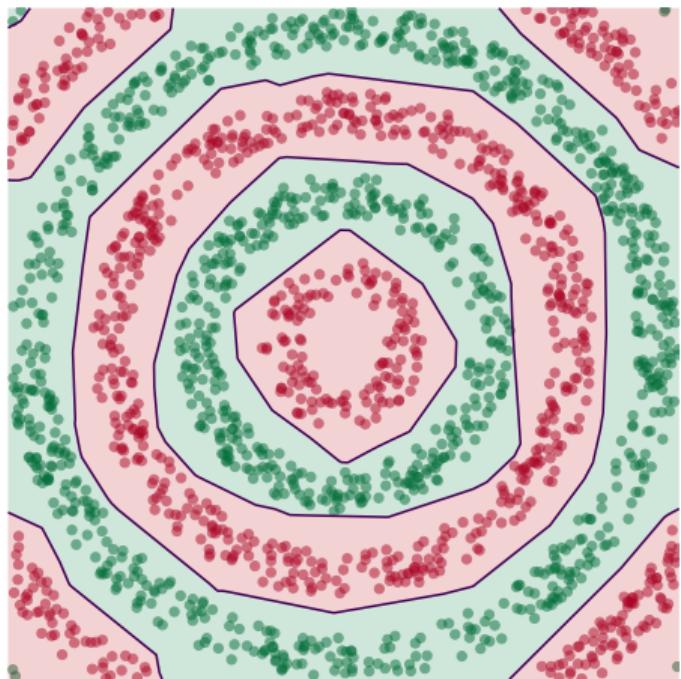
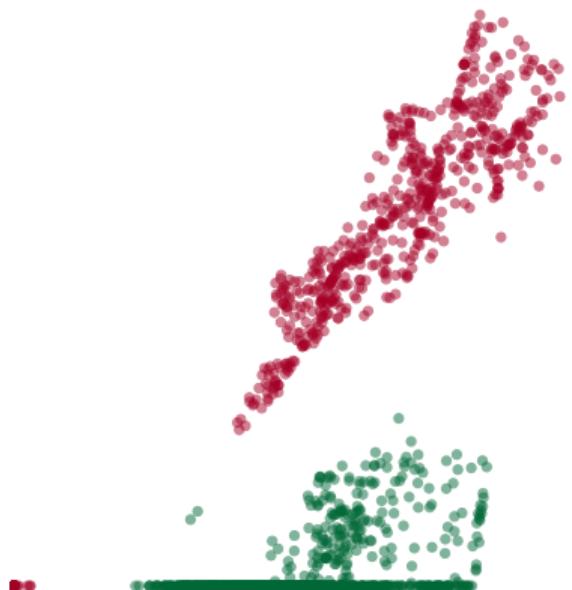
Learning a New Representation



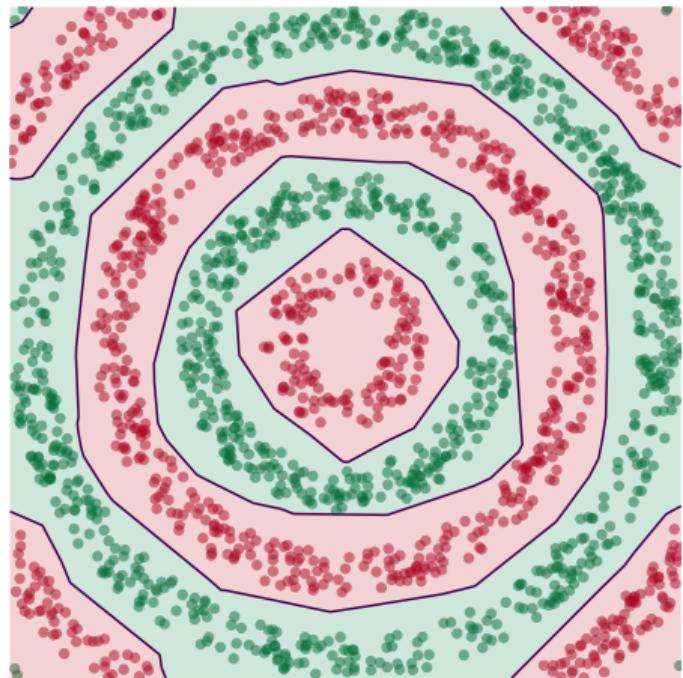
Learning a New Representation



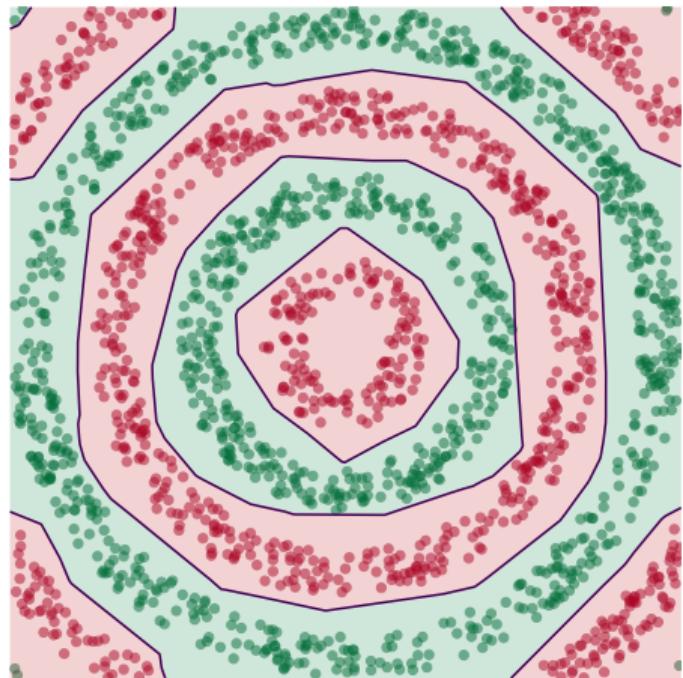
Learning a New Representation



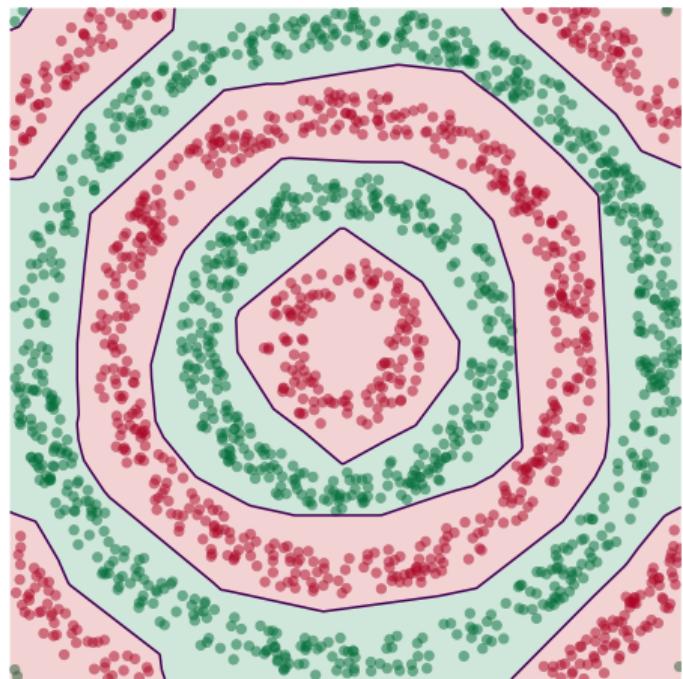
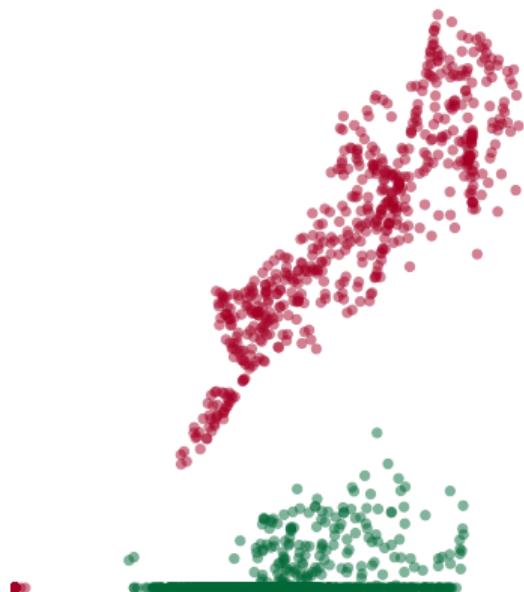
Learning a New Representation



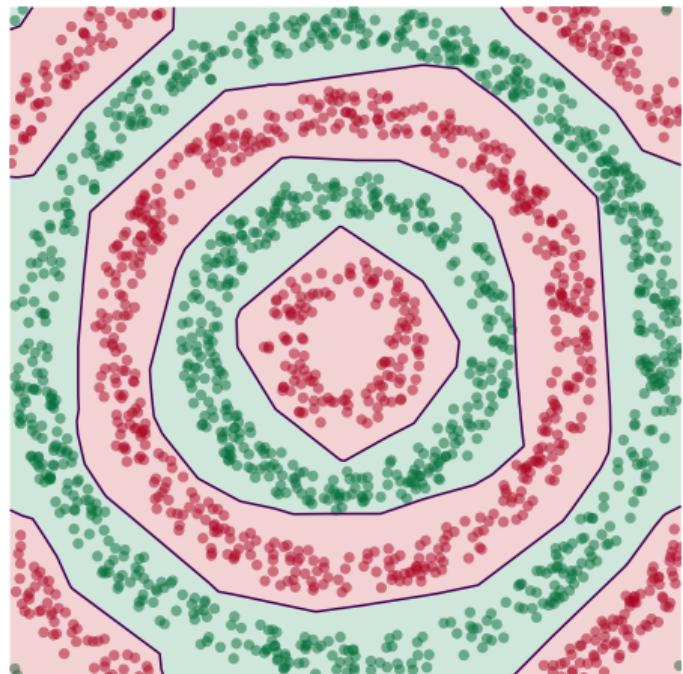
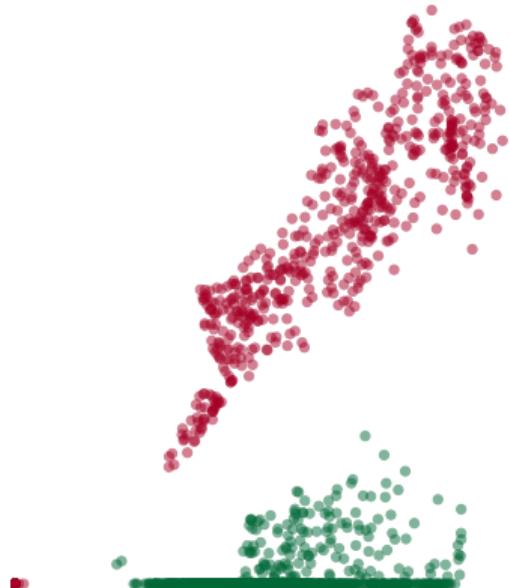
Learning a New Representation



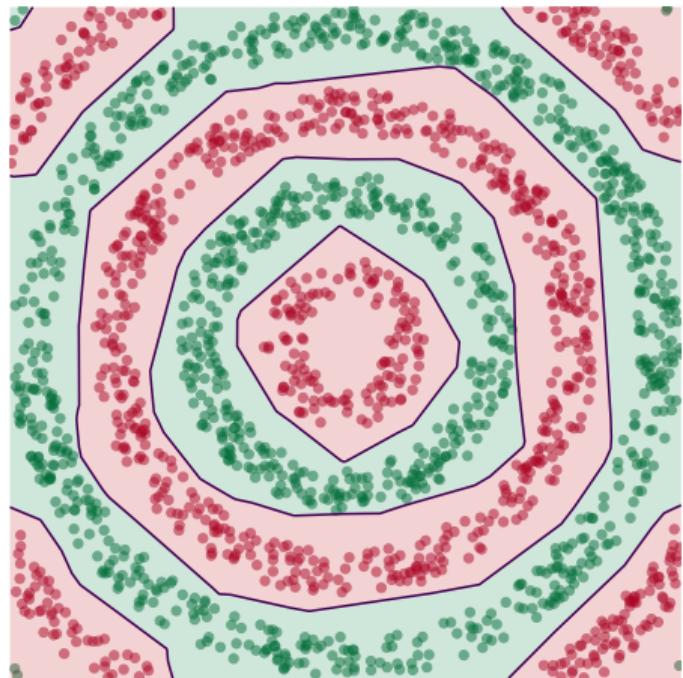
Learning a New Representation



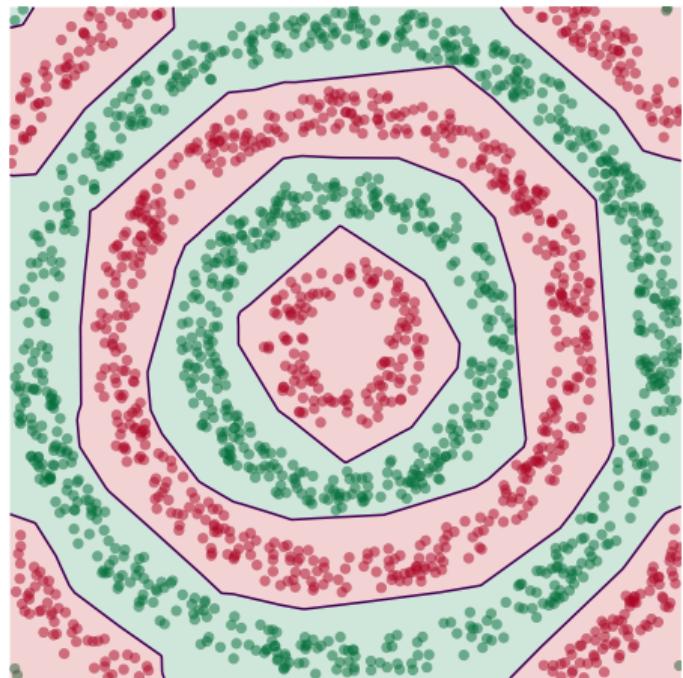
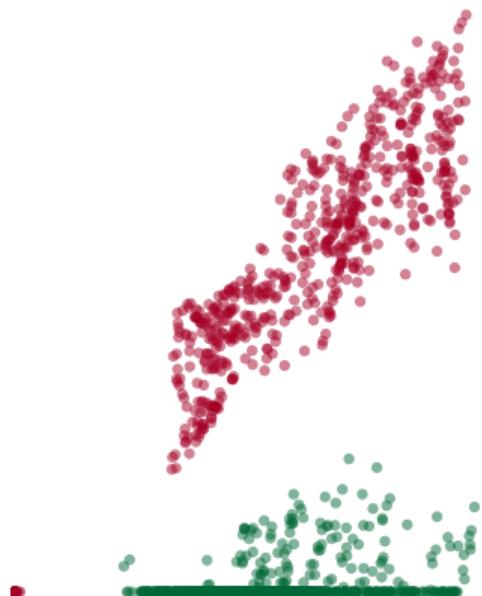
Learning a New Representation



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

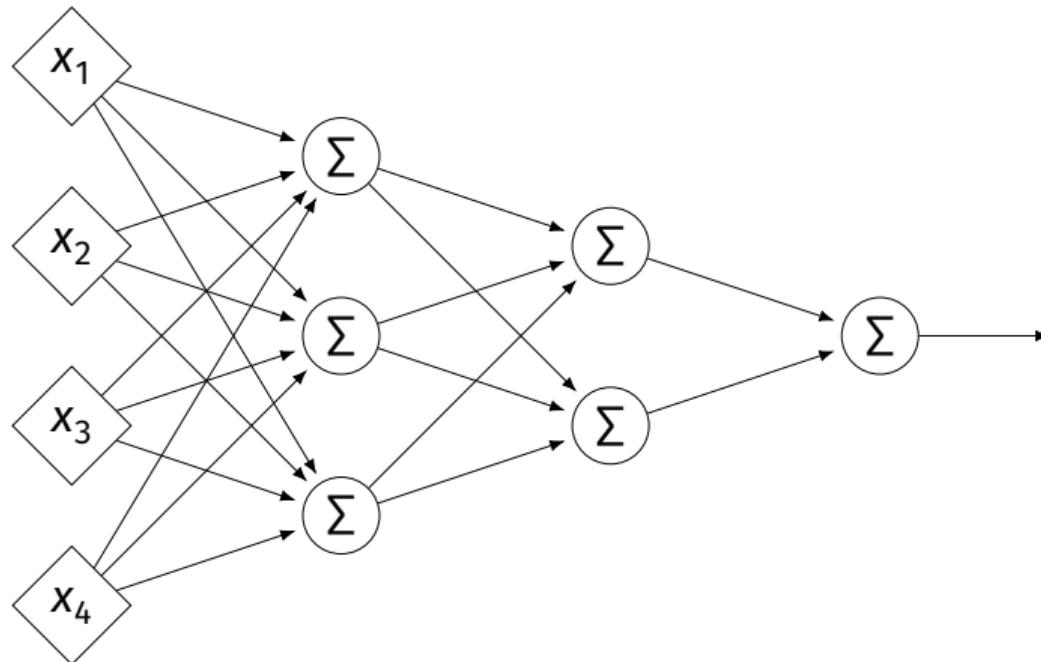
Representation Learning

Lecture 13 | Part 3

Training Neural Networks

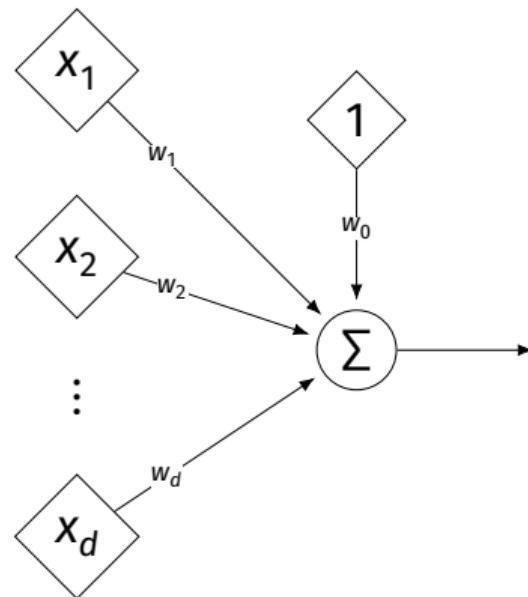
Training

- ▶ 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: $H(\vec{x}; \vec{w}) = w_0 + w_1x_1 + \dots + w_dx_d = \text{Aug}(\vec{x}) \cdot \vec{w}$
1. Pick a loss function.
 - ▶ E.g., the square loss.
2. Minimize the empirical risk w.r.t. that loss:

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

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, $\nabla_{\vec{w}} R(\vec{w}) = (\partial R / \partial w_0, \partial R / \partial w_1, \dots, \partial R / \partial w_d)^T$

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

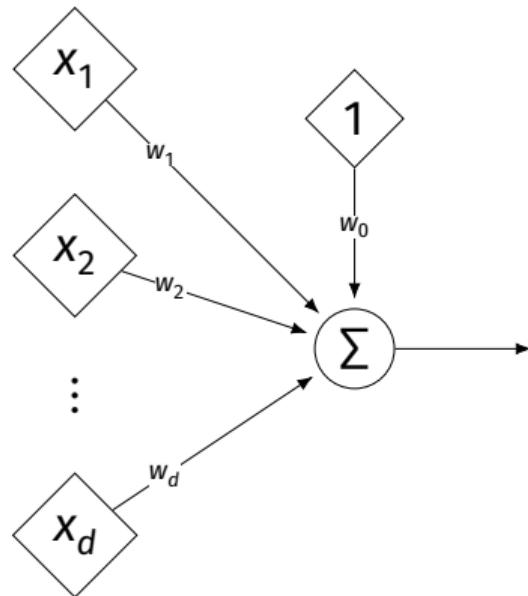
- ▶ Suppose H is a linear prediction function:

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

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

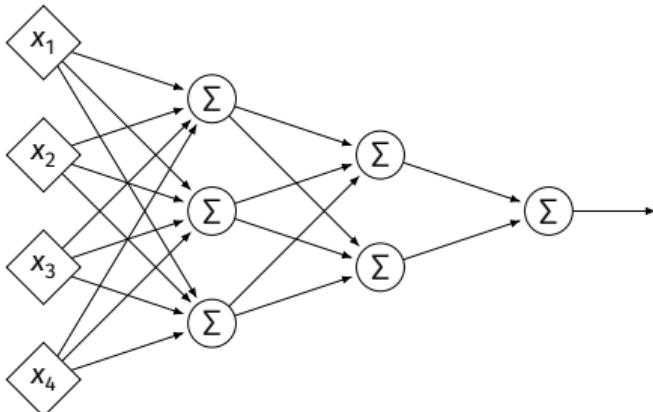
Example: Linear Model

- ▶ Consider $\partial H / \partial w_1$:



Example: Neural Networks

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



Parameter Vectors

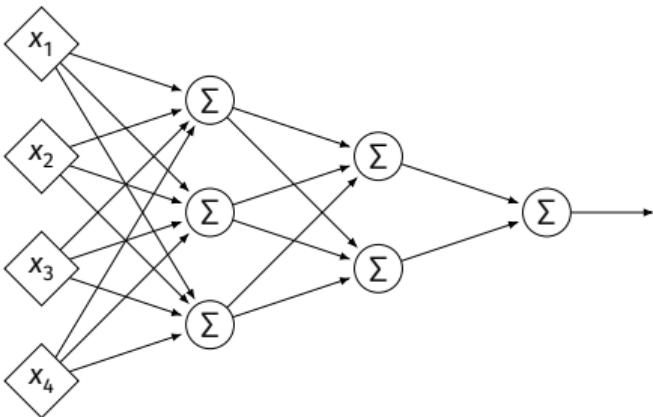
- ▶ It is often useful to pack all of the network's weights into a **parameter vector**, \vec{w} .
- ▶ Order is arbitrary:

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

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

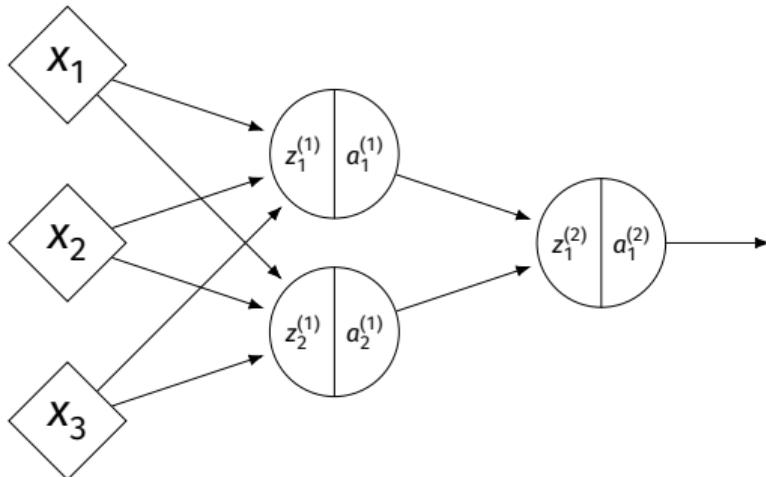
Example: Neural Networks

- ▶ $\nabla_{\vec{w}} H$ is a vector-valued function with one entry per parameter.
- ▶ Plugging a data point, \vec{x} , and a parameter vector, \vec{w} , into $\nabla_{\vec{w}} H$ “evaluates the gradient”, results in a vector.



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 $\partial H / \partial W_{11}^{(1)}(\vec{x}, \vec{w})$?

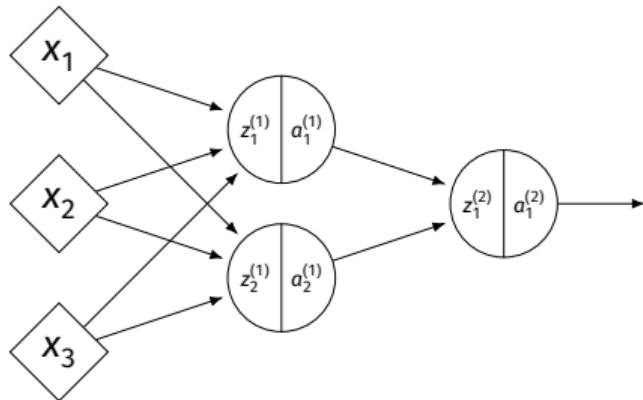


Recall: Chain Rule

$$\begin{aligned}\frac{df}{dx}[f(g(x))] &= \frac{df}{dg}(g(x)) \frac{dg}{dx} \\ &= f'(g(x)) g'(x)\end{aligned}$$

Example: Neural Network

- ▶ Compute $\partial H / \partial W_{11}^{(1)}$ for the network shown below. Assume the hidden layer activation function is σ .

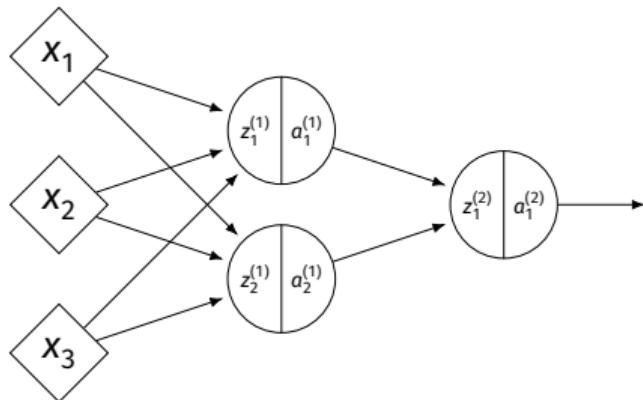


$$W^{(1)} = \begin{pmatrix} 3 & 2 \\ 4 & 1 \\ 1 & 3 \end{pmatrix} \quad W^{(2)} = \begin{pmatrix} 3 \\ 1 \end{pmatrix}$$

$$\vec{x} = (3, 1, 2)^T$$

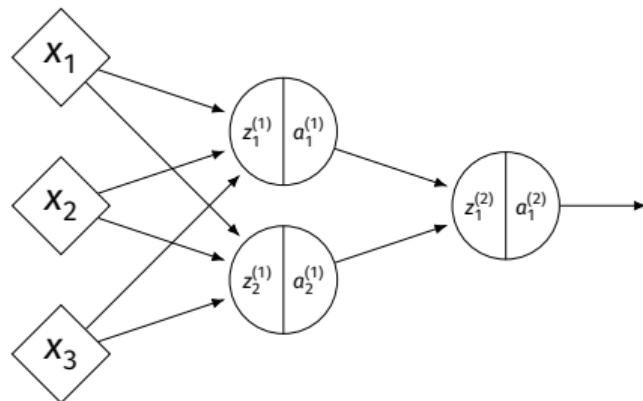
Example: Neural Network

- ▶ Compute $\partial H / \partial W_{11}^{(1)}$ for the network shown below. Assume the hidden layer activation function is σ .



Example: Neural Network

- ▶ Compute $\partial H / \partial W_{21}^{(1)}$ for the network shown below. Assume the hidden layer activation function is σ .

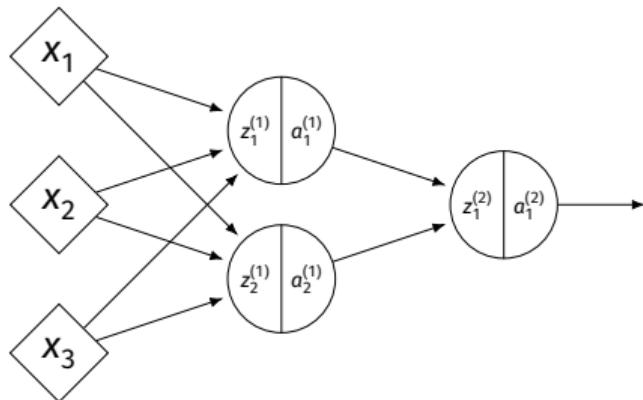


$$W^{(1)} = \begin{pmatrix} 3 & 2 \\ 4 & 1 \\ 1 & 3 \end{pmatrix} \quad W^{(2)} = \begin{pmatrix} 3 \\ 1 \end{pmatrix}$$

$$\vec{x} = (3, 1, 2)^T$$

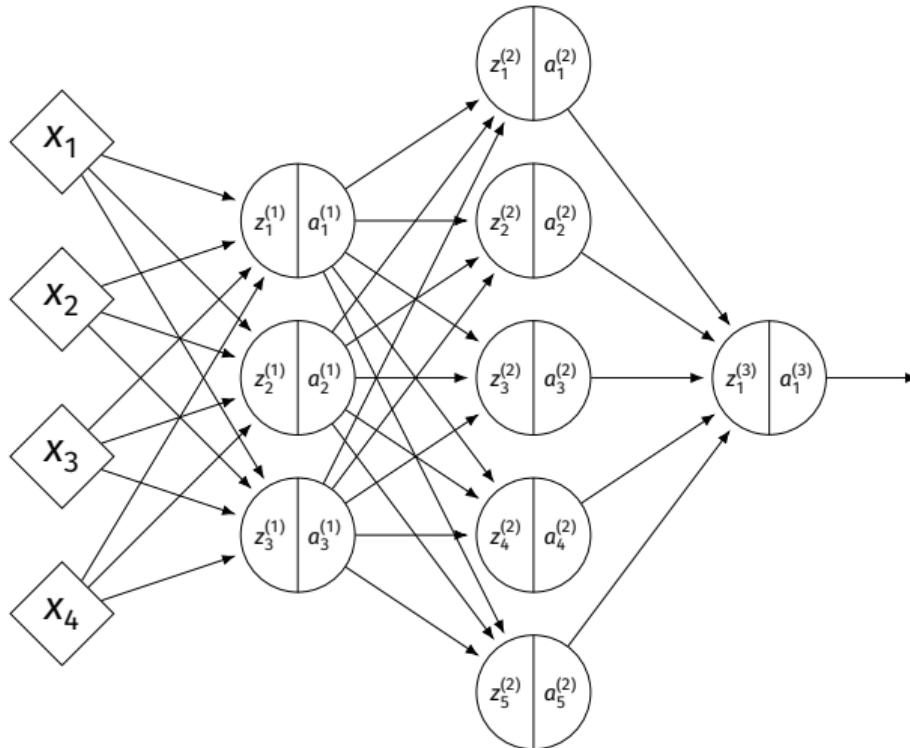
Example: Neural Network

- ▶ Compute $\partial H / \partial W_{21}^{(1)}$ for the network shown below. Assume the hidden layer activation function is σ .



Imagine...

► Compute $W_{11}^{(1)}$:



A Better Way

- ▶ Computing the gradient is straightforward...
- ▶ But can involve a lot of redundant work.
- ▶ **Backpropagation** is a methodical approach to computing the gradient that is more efficient.