DSC 140B
Representation Learning

Lecture 18 | Part 1

Radial Basis Functions

Recap

- Linear prediction functions are limited.
- Idea: transform the data to a new space where prediction is "easier".
- To do so, we used **basis functions**.

Overview: Feature Mapping

1. Start with data in original space, $\mathbb{R}^d$.

2. Choose some basis functions, $\varphi_1, \varphi_2, \ldots, \varphi_{d'}$

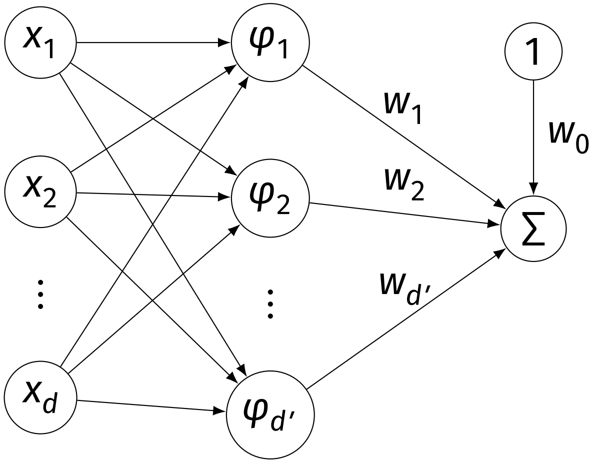
3. Map each data point to **feature space** $\mathbb{R}^{d'}$:

$$\vec{x} \mapsto (\varphi_1(\vec{x}), \varphi_2(\vec{x}), \ldots, \varphi_{d'}(\vec{x}))^t$$

4. Fit linear prediction function in new space:

$$H(\vec{x}) = w_0 + w_1\varphi_1(\vec{x}) + w_2\varphi_2(\vec{x})$$

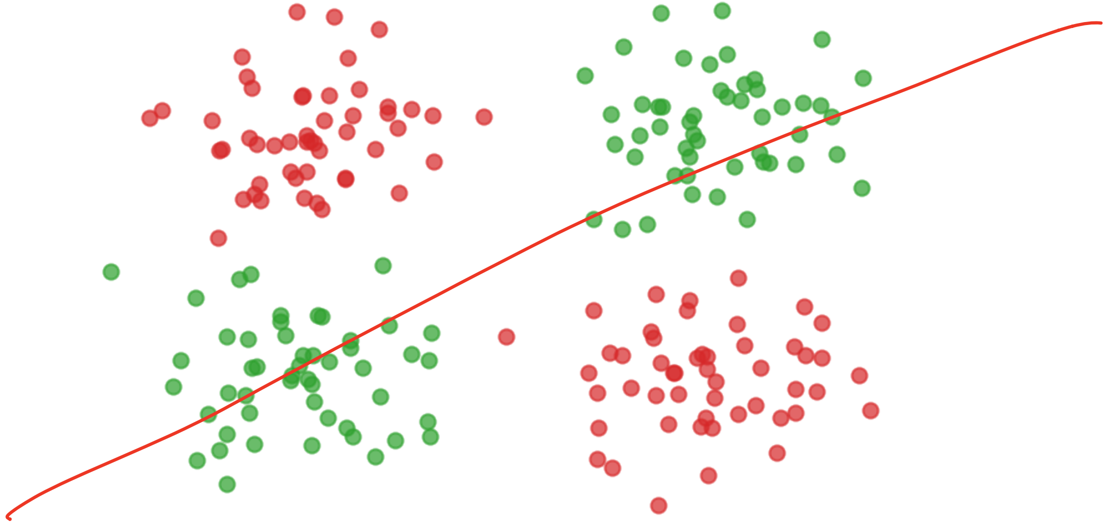
$$H(\vec{x}) = w_0 + w_1\varphi_1(\vec{x}) + w_2\varphi_2(\vec{x})$$



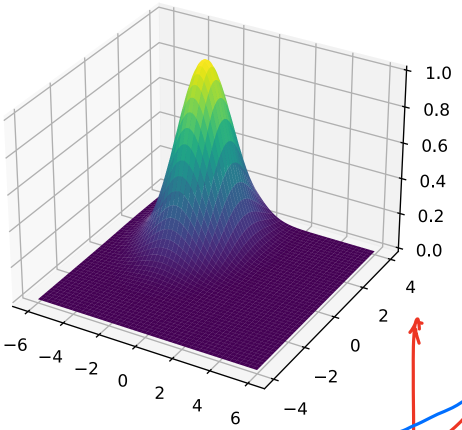
Generic Basis Functions

- The basis functions we used before were engineered using domain knowledge.
- They were specific to the problem at hand.
- **Very manual process!**
- **Now:** features that work for many problems.

Example



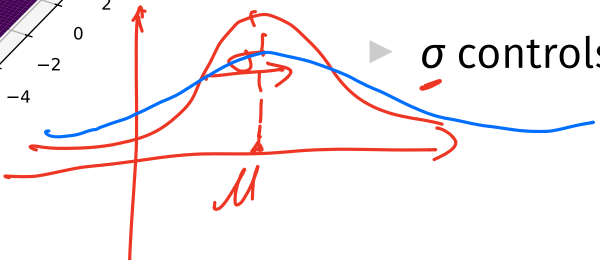
Gaussian Basis Functions



- A common choice: **Gaussian** basis functions:

$$\varphi(\vec{x}; \vec{\mu}, \sigma) = e^{-\|\vec{x}-\vec{\mu}\|^2/\sigma^2}$$

- $\vec{\mu}$ is the center.
- σ controls the "width"



Gaussian Basis Function

- If $\vec{x}$ is close to $\vec{\mu}$, $\varphi(\vec{x}; \vec{\mu}, \sigma)$ is large.
- If $\vec{x}$ is far from $\vec{\mu}$, $\varphi(\vec{x}; \vec{\mu}, \sigma)$ is small.
- Intuition: φ measures how "similar" $\vec{x}$ is to $\vec{\mu}$.
  - Assumes that "similar" objects have close feature vectors.

New Representation

- Pick number of new features, d'.
- Pick centers for Gaussians $\vec{\mu}^{(1)}, \ldots, \vec{\mu}^{(2)}, \ldots, \vec{\mu}^{(d')}$
- Pick widths: $\sigma_1, \sigma_2, \ldots, \sigma_{d'}$ (usually all the same)
- Define ith basis function:

$$\varphi_i(\vec{x}) = e^{-\|\vec{x}-\vec{\mu}^{(i)}\|^2/\sigma_i^2}$$

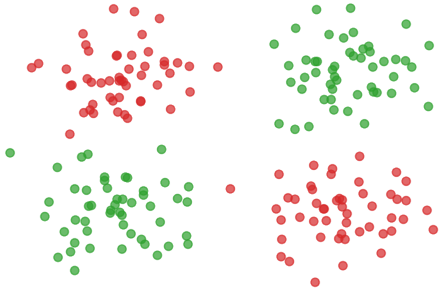
New Representation

- For any feature vector $\vec{x} \in \mathbb{R}^d$, map to vector $\vec{\varphi}(\vec{x}) \in \mathbb{R}^{d'}$.
  - φ_1: "similarity" of $\vec{x}$ to $\vec{\mu}^{(1)}$
  - φ_2: "similarity" of $\vec{x}$ to $\vec{\mu}^{(2)}$
  - ...
  - $\varphi_{d'}$: "similarity" of $\vec{x}$ to $\vec{\mu}^{(d')}$

- Train linear classifier in this new representation.
  - E.g., by minimizing expected square loss.

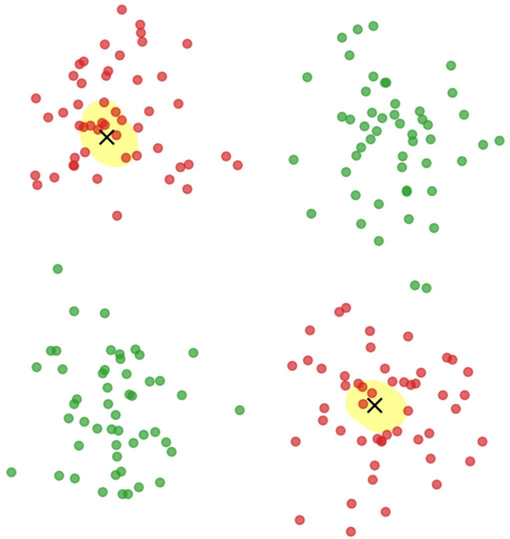
Exercise

How many Gaussian basis functions would you use, and where would you place them to create a new representation for this data?

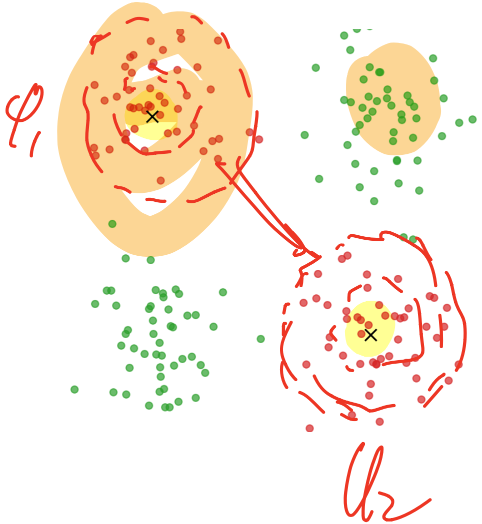


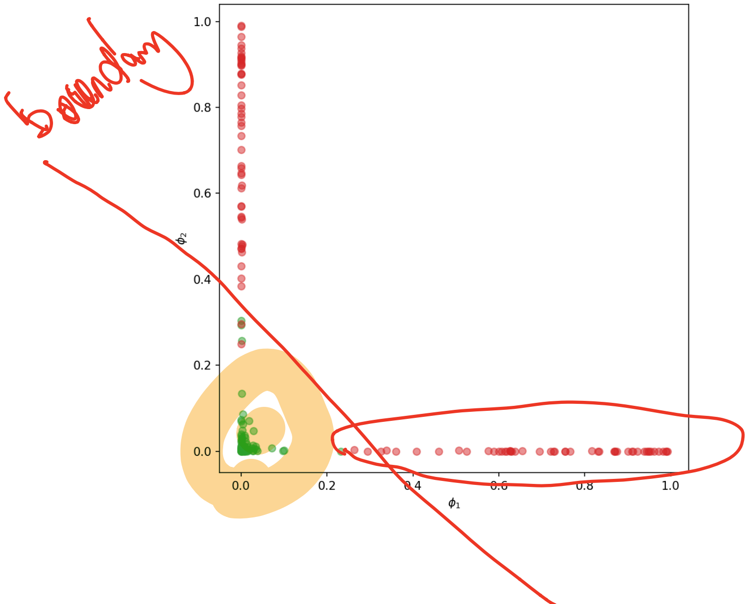
Placement

d = 2



Feature Space

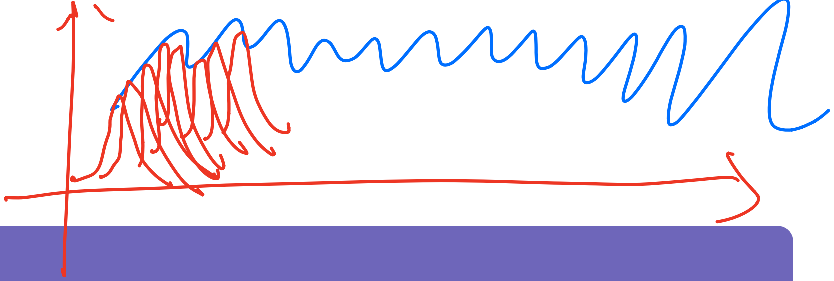




Prediction Function

- $H(\vec{x})$ is a sum of Gaussians:

$$H(\vec{x}) = w_0 + w_1\varphi_1(\vec{x}) + w_2\varphi_2(\vec{x}) + \ldots$$
$$= w_0 + w_1 e^{-\|\vec{x}-\vec{\mu}_1\|^2/\sigma^2} + w_2 e^{-\|\vec{x}-\vec{\mu}_2\|^2/\sigma^2} + \ldots$$



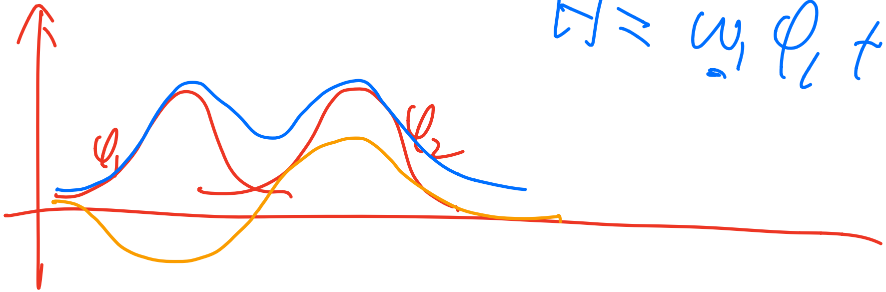
Exercise

What does the surface of the prediction function look like?

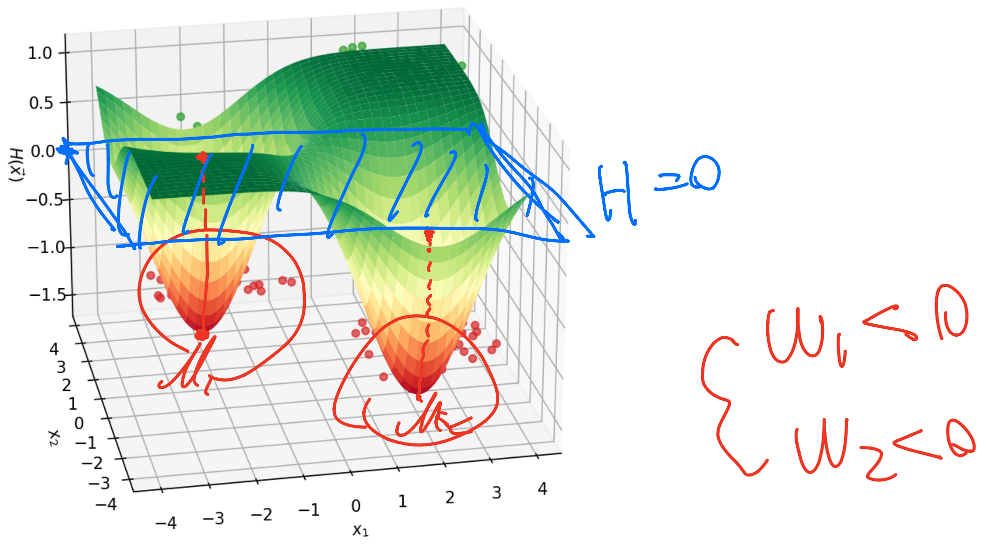
Hint: what does the sum of 1-d Gaussians look like?

H

$H = w_1 \varphi_1 + w_2 \varphi_2 + w_0$



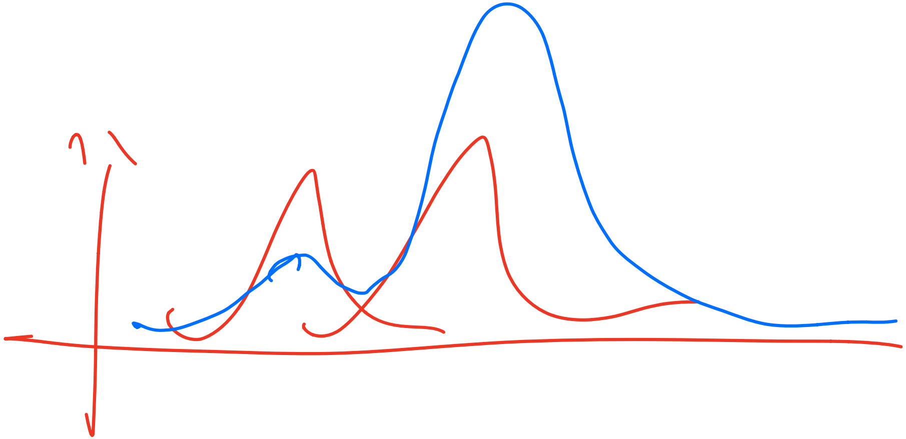
Prediction Function Surface



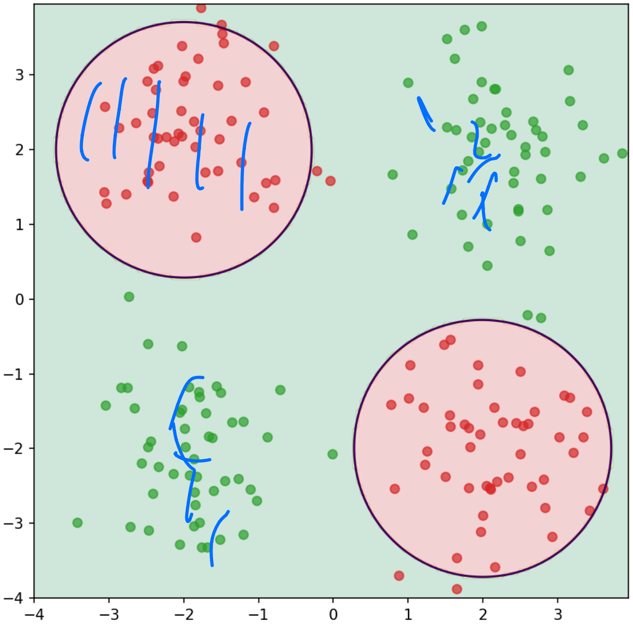
$$H(\vec{x}) = w_0 + w_1 e^{-\|\vec{x}-\vec{\mu}_1\|^2/\sigma^2} + w_2 e^{-\|\vec{x}-\vec{\mu}_2\|^2/\sigma^2}$$

An Interpretation

- Basis function φ_i makes a "bump" in surface of H
- w_i adjusts the "prominance" of this bump

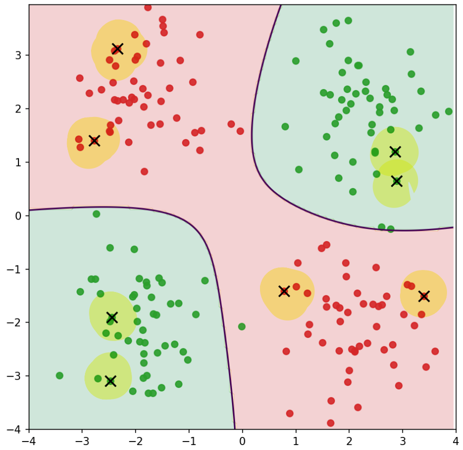


Decision Boundary

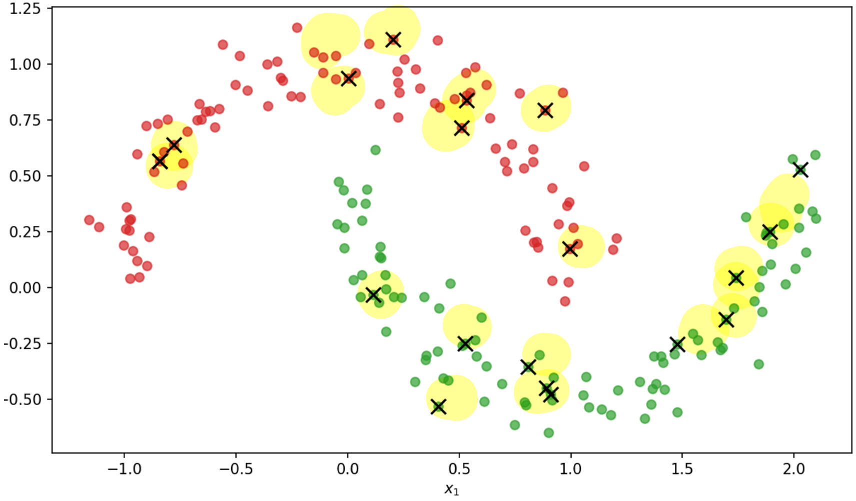


More Features

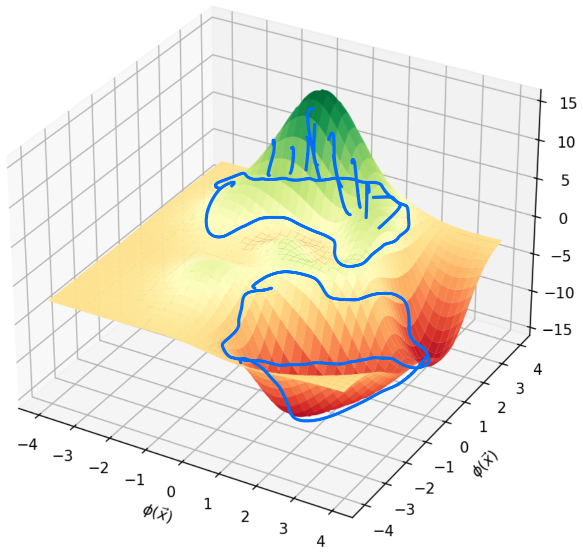
- By increasing number of basis functions, we can make more complex decision surfaces.



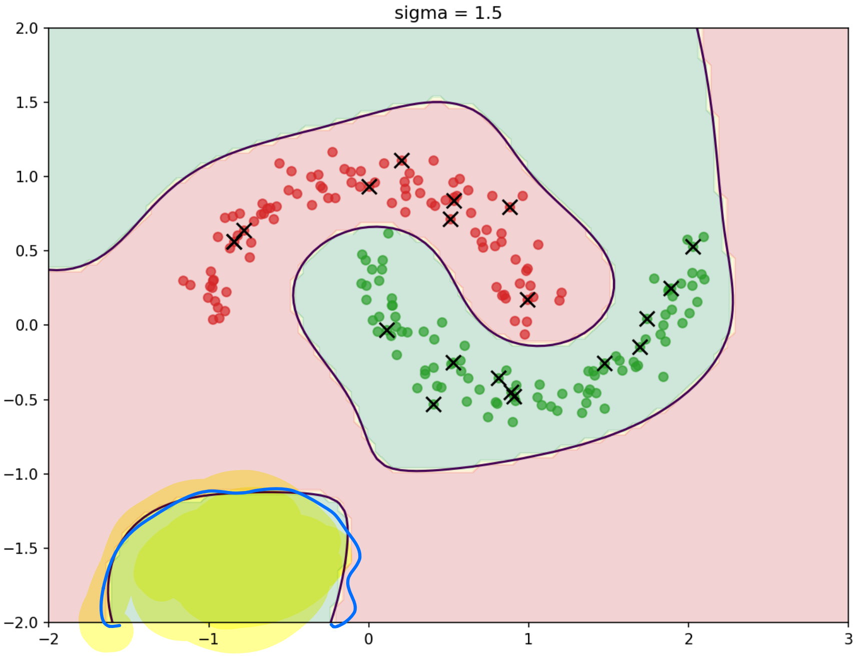
Another Example



Prediction Surface



Decision Boundary



Radial Basis Functions

- Gaussians are examples of **radial basis functions**.
- Each basis function has a **center**, $\vec{c}$.
- Value depends only on distance from center:

$$\varphi(\vec{x}; \vec{c}) = f(\|\vec{x} - \vec{c}\|)$$

Another Radial Basis Function

- **Multiquadric:** $\varphi(\vec{x}; \vec{c}) = \sqrt{\sigma^2 + \|\vec{x} - \vec{c}\|}/\sigma$

DSC 140B

Representation Learning

Lecture 18 | Part 2

Radial Basis Function Networks

Recap

1. Choose basis functions, $\varphi_1, \ldots, \varphi_{d'}$

2. Transform data to new representation:

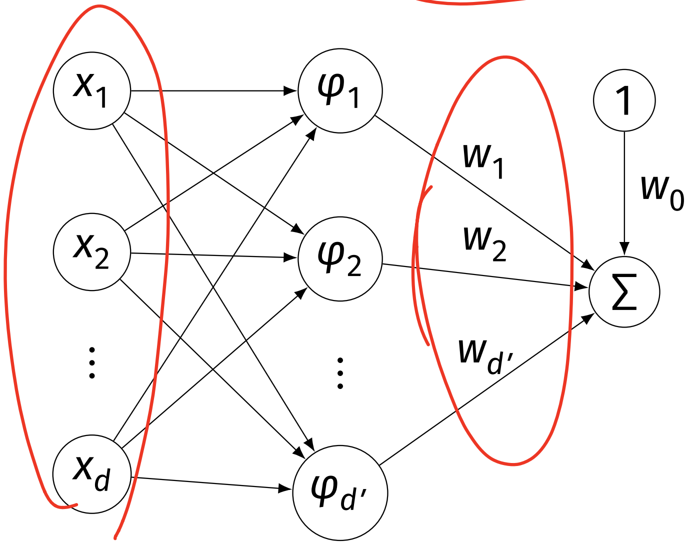
$$\vec{x} \mapsto (\varphi_1(\vec{x}), \varphi_2(\vec{x}), \ldots, \varphi_{d'}(\vec{x}))^T$$

3. Train a linear classifier in this new space:

$$H(\vec{x}) = w_0 + w_1\varphi_1(\vec{x}) + w_2\varphi_2(\vec{x}) + \ldots + w_{d'}\varphi_{d'}(\vec{x})$$

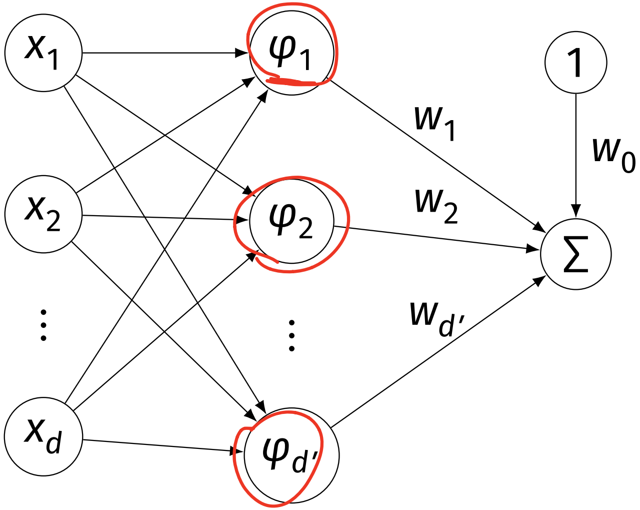
The Model

- The φ are **basis functions.**



$$H(\vec{x}) = w_0 + w_1 \varphi_1(\vec{x}) + w_2 \varphi_2(\vec{x})$$

Radial Basis Function Networks



If the basis functions are **radial basis functions**, we call this a **radial basis function (RBF) network**.

Training

- An RBF network has these parameters:
  - the parameters of each individual basis function:
    - $\vec{\mu}_i$ (the center)
    - possibly others (e.g., σ)
  - w_i: the weights associated to each “new” feature

- How do we choose the parameters?

First Idea

- We can include all parameters in one big cost function, optimize.
- The cost function will generally be **complicated, non-convex** and thus **hard to optimize**.

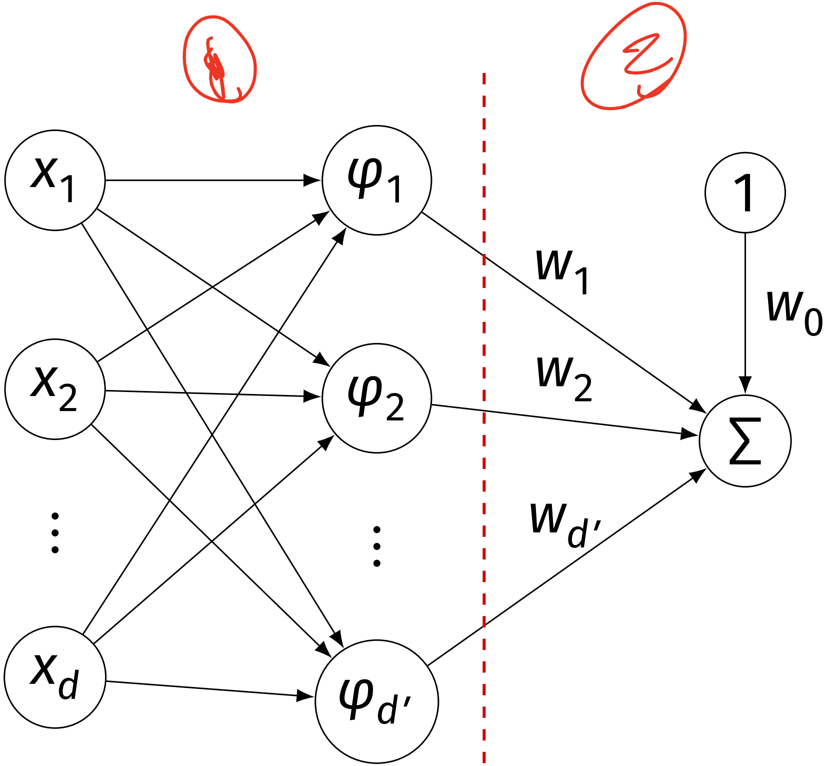
Another Idea

- Break the process into two steps:

1. Find the parameters of the RBFs *somehow*.
    - Some optimization procedure, clustering, randomly, ...
2. Having fixed those parameters, optimize the w's.

- **Linear; easier to optimize.**

Training



Training an RBF Network

1. Choose the form of the RBF, how many.
   - E.g., k Gaussian RBFs, $\varphi_1, \ldots, \varphi_k$.

2. Pick the parameters of the RBFs *somehow*.

3. Create new data set by mapping
   $\vec{x} \mapsto (\varphi_1(\vec{x}), \ldots, \varphi_k(\vec{x}))^T$

4. Train a linear predictor H_f on new data set
   - That is, in feature space.

Making Predictions

1. Given a point $\vec{x}$, map it to feature space:
   $\vec{x} \mapsto (\varphi_1(\vec{x}), \ldots, \varphi_k(\vec{x}))^T$

2. Evaluate the trained linear predictor H_f in feature space

DSC 140B
Representation Learning

Lecture 18 | Part 3

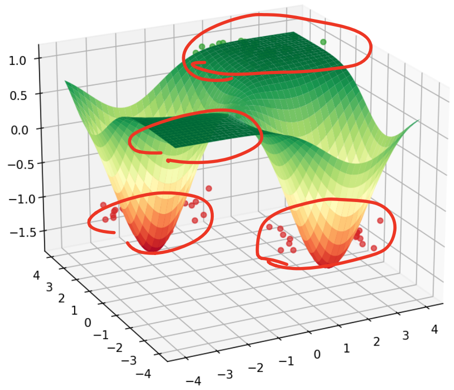
Choosing RBF Locations

Recap

- We map data to a new representation by first choosing **basis functions**.
- Radial Basis Functions (RBFs), such as Gaussians, are a popular choice.
- Requires choosing **center** for each basis function.

Prediction Function

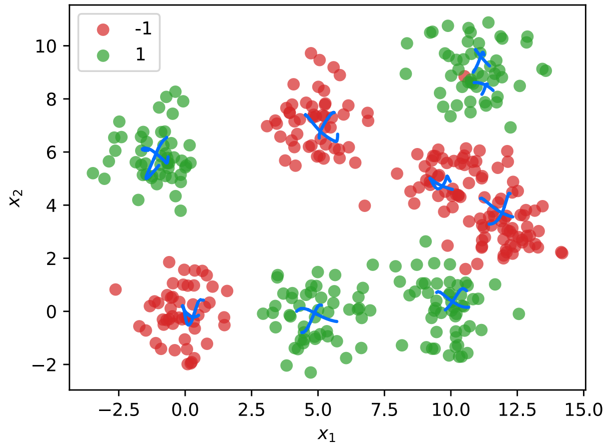
- Our prediction function H is a surface that is made up of Gaussian "bumps".



$$H(\vec{x}) = w_0 + w_1 e^{-\|\vec{x}-\vec{\mu}_1\|^2/\sigma^2} + w_2 e^{-\|\vec{x}-\vec{\mu}_2\|^2/\sigma^2}$$

Choosing Centers

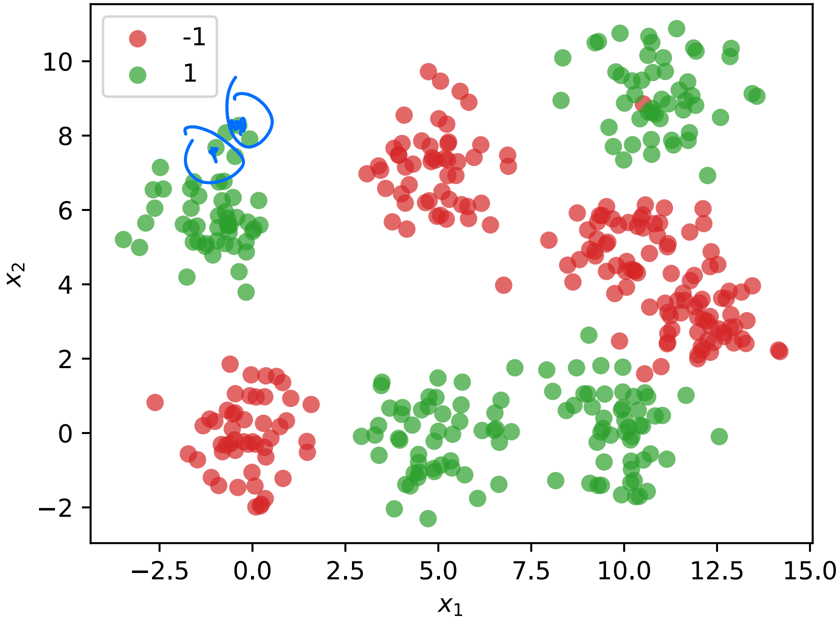
- Place the centers where the value of the prediction function should be controlled.
- Intuitively: place centers where the data is.



Approaches

1. Every data point as a center
2. Randomly choose centers
3. Clustering

Approach #1: Every Data Point as a Center



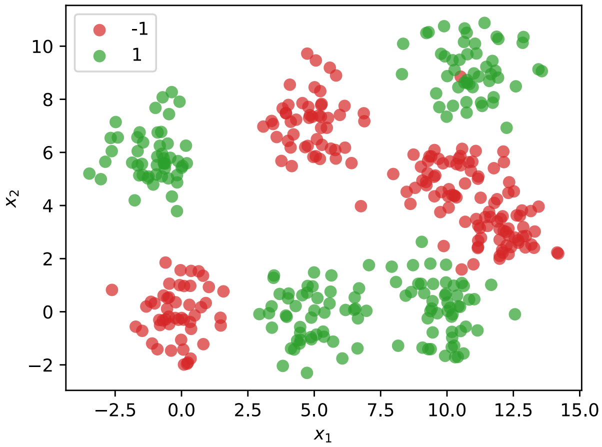
Dimensionality

- We'll have n basis functions – one for each point.
- That means we'll have n features.
- Each feature vector $\vec{\phi}(\vec{x}) \in \mathbb{R}^n$.

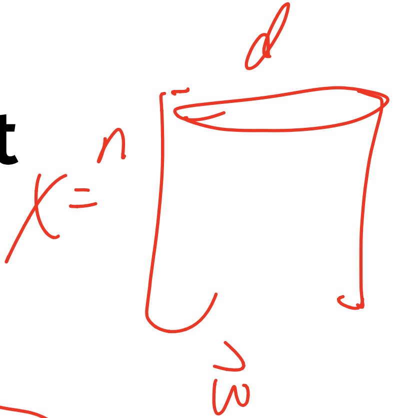
$$\vec{\phi}(\vec{x}) = (\phi_1(\vec{x}), \phi_2(\vec{x}), \ldots, \phi_n(\vec{x}))^T$$

Problems

- This causes problems.
- First: more likely to **overfit**.
- Second: computationally expensive



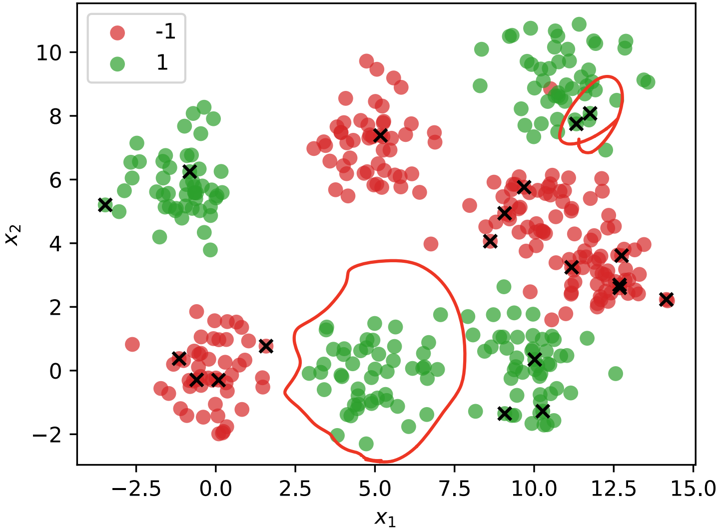
Computational Cost



- Suppose feature matrix X is $n \times d$
  - n points in d dimensions
- Time complexity of solving $X^T X \vec{w} = X^T \vec{y}$ is $\Theta(nd^2)$
- Usually $d \ll n$. But if $d = n$, this is $\Theta(n^3)$.
- Not great! If $n \approx 10,000$, then takes > 10 minutes.

Approach #2: A Random Sample

- Idea: randomly choose k data points as centers.



Problem

- May undersample/oversample a region.
- More advanced sampling approaches exist.

Approach #3: Clustering

- Group data points into **clusters**.
- Cluster centers are good places for RBFs.
- For example, use k-means clustering to pick k centers.

DSC 140B
Representation Learning

Lecture 18 | Part 4

Neural Networks

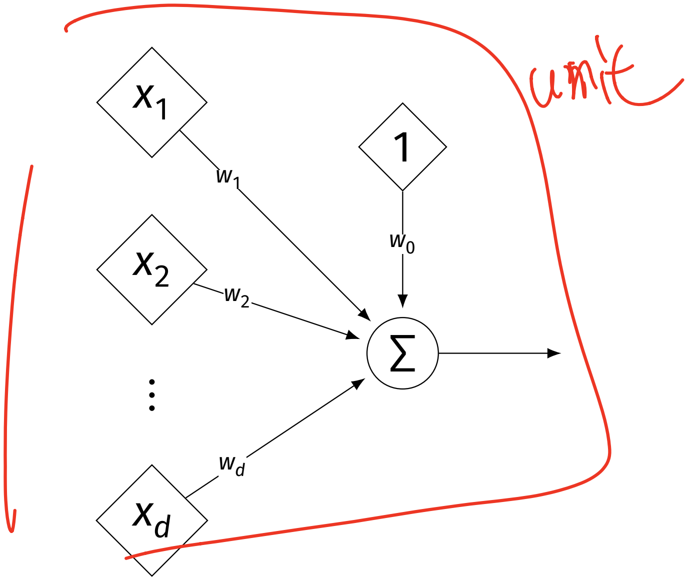
Beyond RBFs

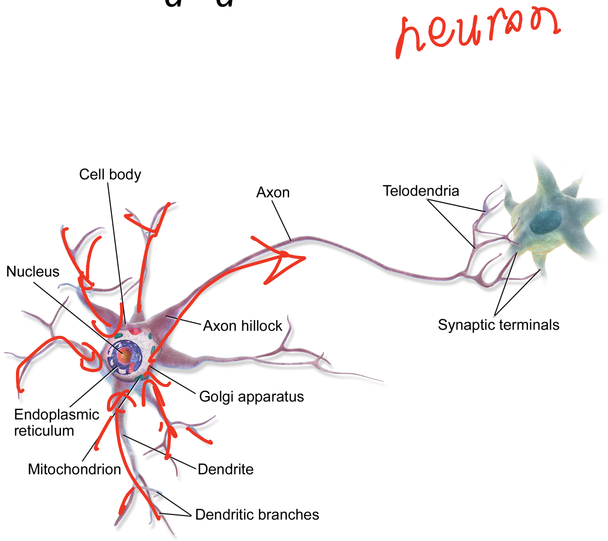
- When training RBFs, we fixed the basis functions *before* training the weights.
- Representation learning was decoupled from learning the prediction function.
- **Now:** learn representation **and** prediction function together.

end-to-end

Linear Models

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

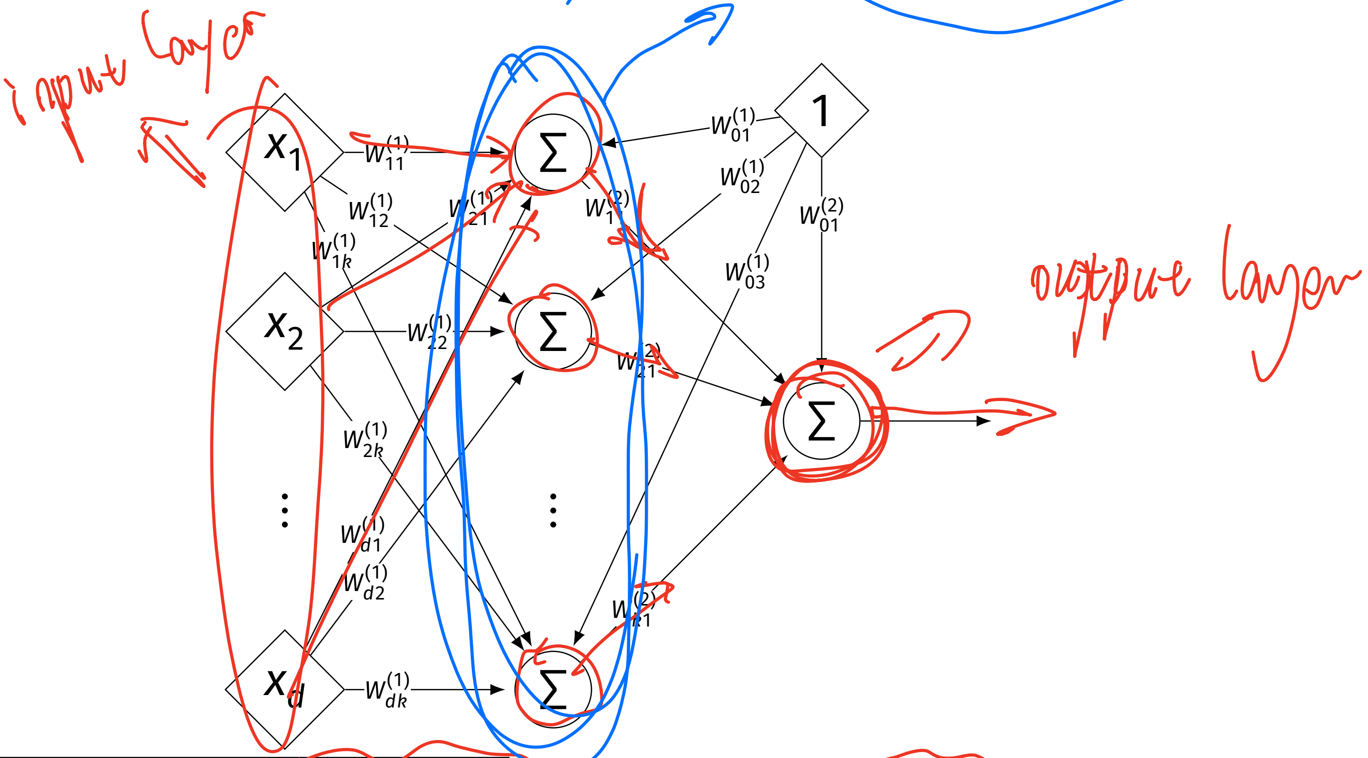




Generalizing Linear Models

- 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 Network[1] (One Hidden Layer)



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

Architecture

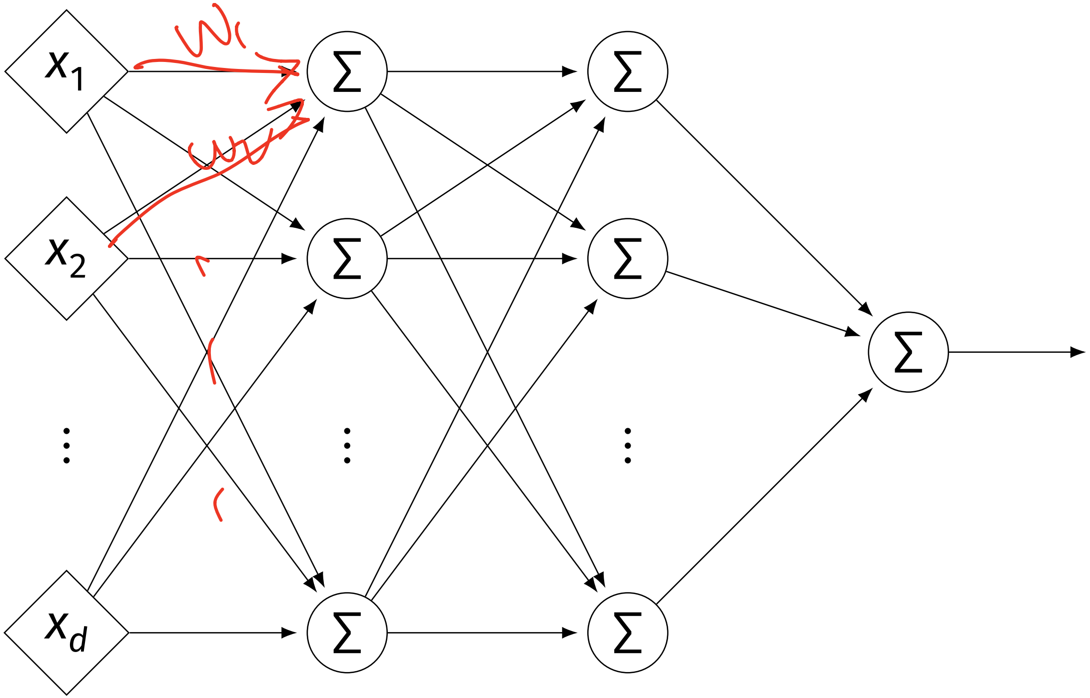
- Neurons are organized into **layers**.
  - **Input layer**, **output layer**, and **hidden layers**.
- Number of cells in input layer determined by dimensionality of input feature vectors.
- Number of cells in hidden layer(s) is determined by you.
- Output layer can have >1 neuron.

k-way

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)

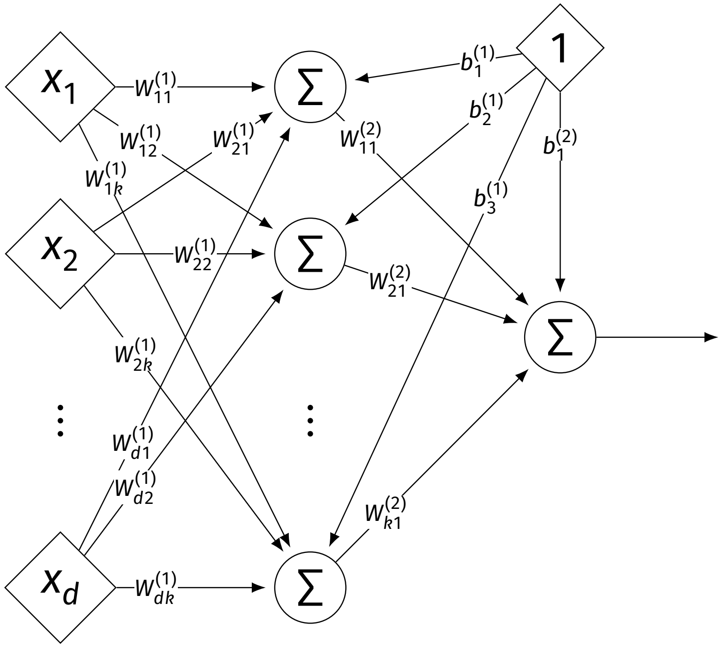


Network Weights

- 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!

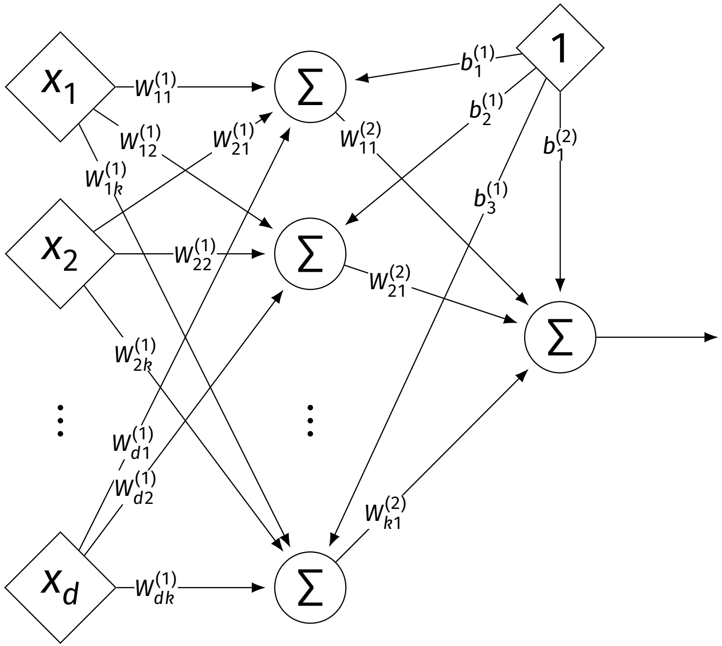
Notation

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



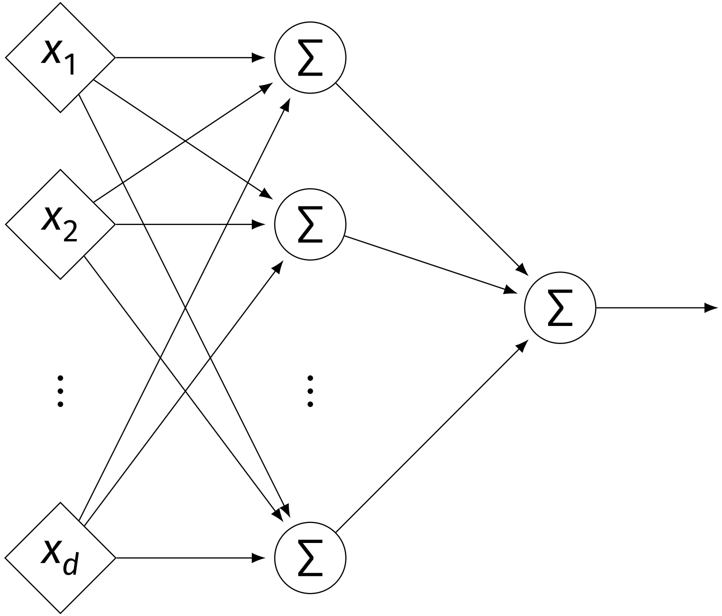
Notation

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

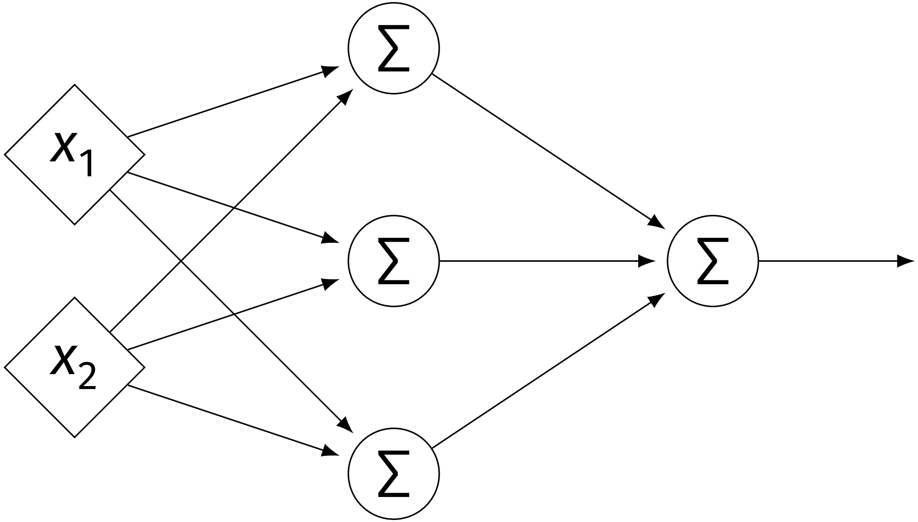


Notation

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



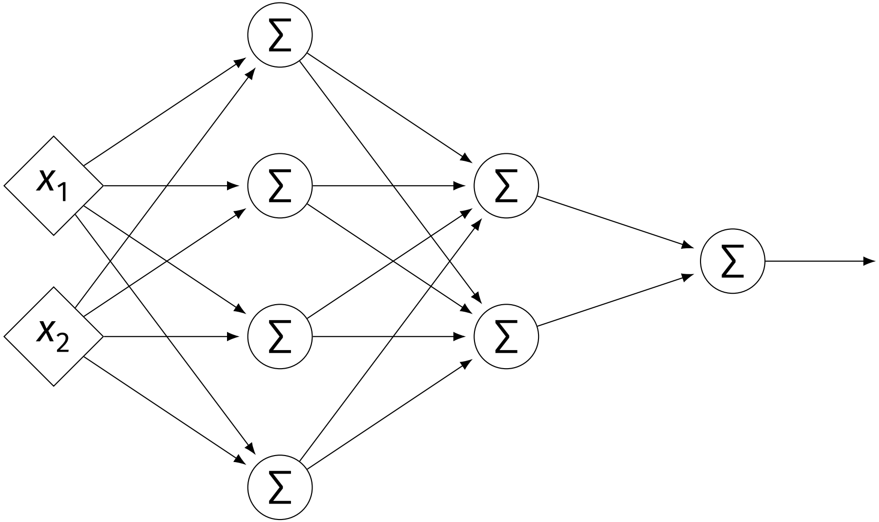
Example



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

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

Example



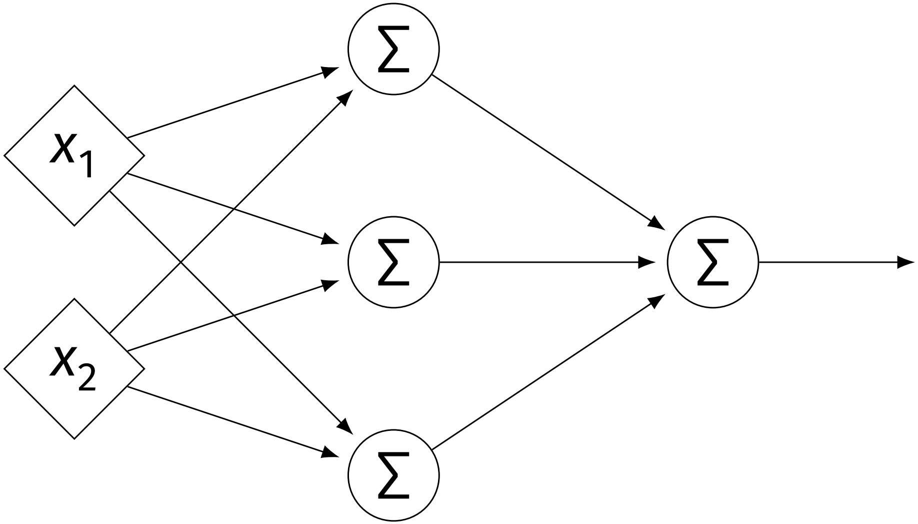
$$W^{(1)} = \begin{pmatrix} 2 & -1 & -3 & 0 \\ 4 & 5 & -7 & 2 \end{pmatrix} \quad W^{(2)} = \begin{pmatrix} 1 & 2 \\ -4 & 3 \\ -6 & -2 \\ 3 & 4 \end{pmatrix} \quad W^{(3)} = \begin{pmatrix} -1 & 5 \end{pmatrix}$$

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

Evaluation

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

Example



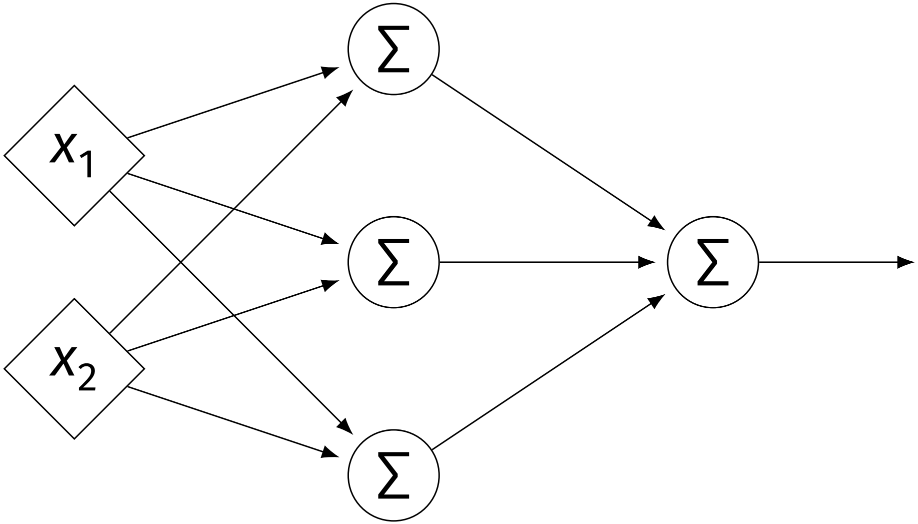
- $\vec{x} = (3, -1)^T$
- $z_1^{(1)} =$
- $z_2^{(1)} =$
- $z_3^{(1)} =$
- $z_1^{(2)} =$

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

Evaluation as Matrix Multiplication

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

Example



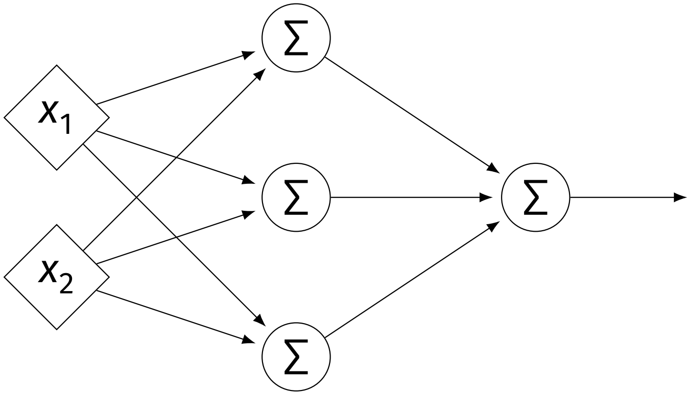
- $\vec{x} = (3, -1)^T$
- $z_1^{(1)} =$
- $z_2^{(1)} =$
- $z_3^{(1)} =$
- $z_1^{(2)} =$

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

Each Layer is a Function

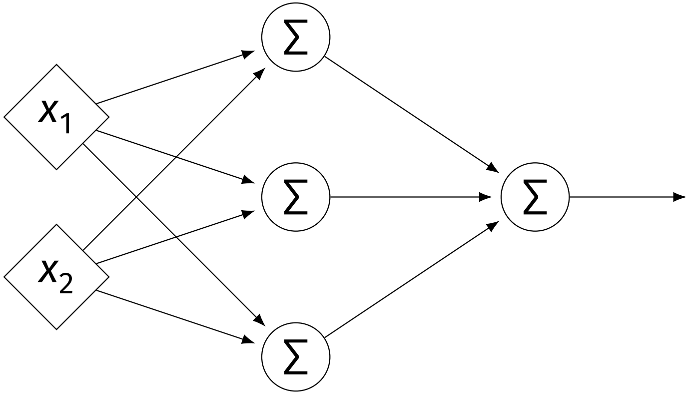
- We can think of each layer as a function mapping a vector to a vector.

- $H^{(1)}(\vec{z}) = \left[W^{(1)}\right]^T \vec{z} + \vec{b}^{(1)}$
  - $H^{(1)} : \mathbb{R}^2 \to \mathbb{R}^3$
- $H^{(2)}(\vec{z}) = \left[W^{(2)}\right]^T \vec{z} + \vec{b}^{(2)}$
  - $H^{(2)} : \mathbb{R}^3 \to \mathbb{R}^1$



NNs as Function Composition

- The full NN is a composition of layer functions.



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

NNs as Function Composition

- In general, if there k hidden layers:

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

Exercise

Show that:

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

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:

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

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

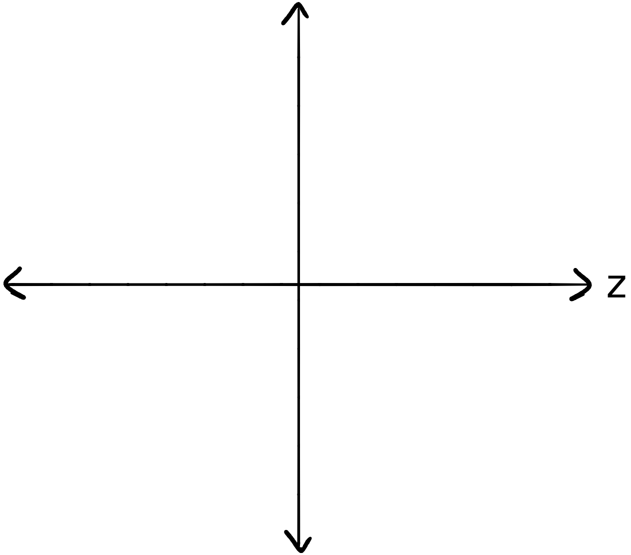
Idea

- 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.

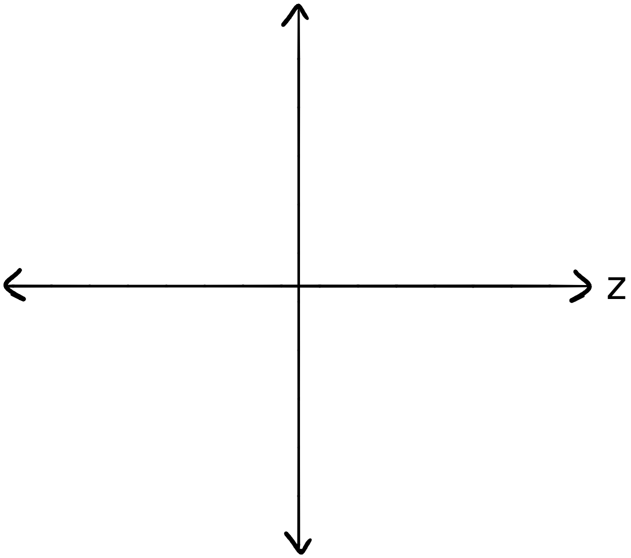
$$\sigma(z) = z$$



Sigmoid Activation

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

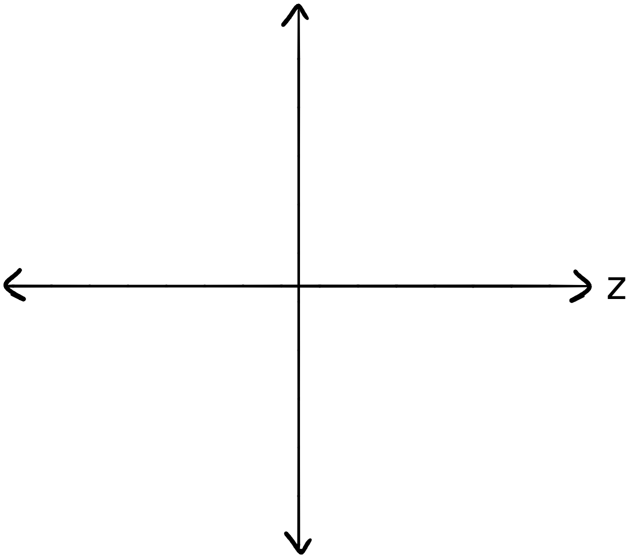
$$\sigma(z) = \frac{1}{1 + e^{-z}}$$



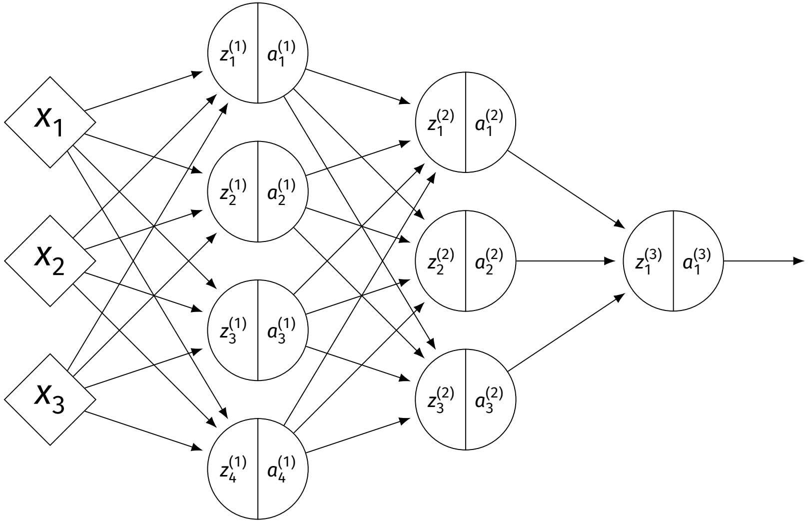
ReLU Activation

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

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

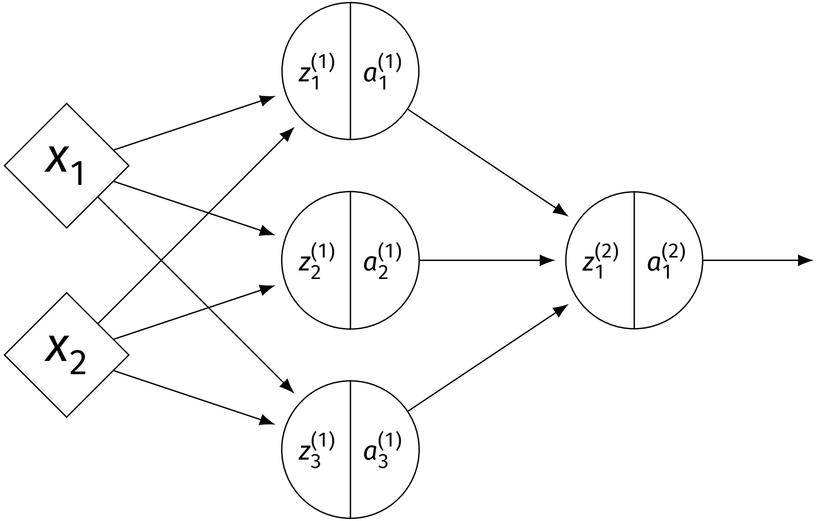


Notation



- $z_j^{(i)}$ is the linear activation before g is applied.
- $a_j^{(i)} = g(z^{(i)})$ 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 18 | Part 5

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

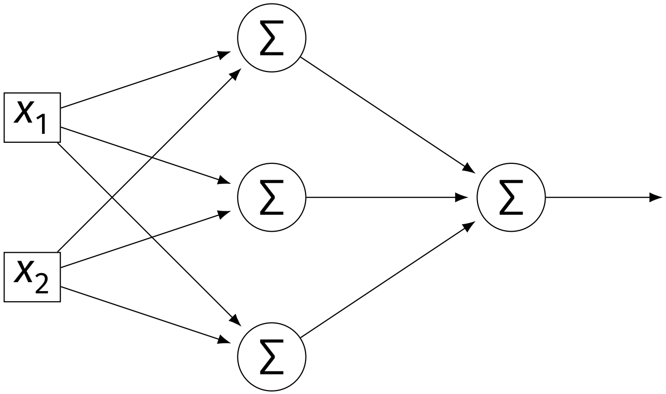
- **Interpretation:** The hidden layers of a neural network **learn** a feature map.

Each Layer is a Function

- We can think of each layer as a function mapping a vector to a vector.

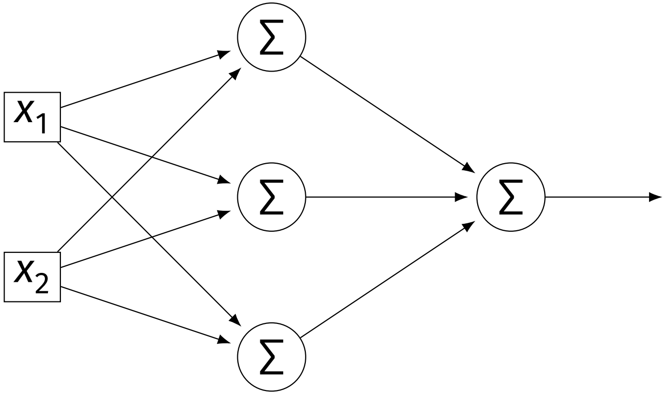
- $H^{(1)}(\vec{z}) = \left[W^{(1)}\right]^T \vec{z} + \vec{b}^{(1)}$
  - $H^{(1)} : \mathbb{R}^2 \to \mathbb{R}^3$

- $H^{(2)}(\vec{z}) = \left[W^{(2)}\right]^T \vec{z} + \vec{b}^{(2)}$
  - $H^{(2)} : \mathbb{R}^3 \to \mathbb{R}^1$



Each Layer is a Function

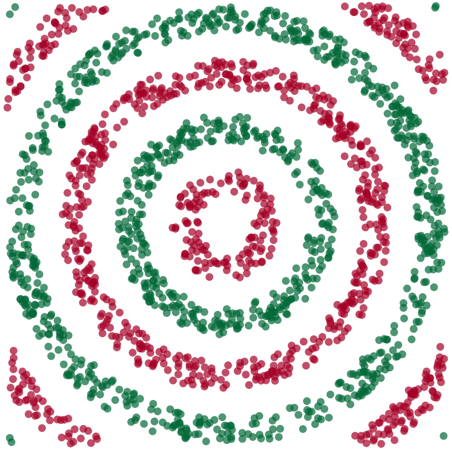
- The hidden layer performs a feature map from $\mathbb{R}^2$ to $\mathbb{R}^3$.
- The output layer makes a prediction in $\mathbb{R}^3$.
- **Intuition:** The feature map is learned so as to make the output layer's job "easier".



Demo

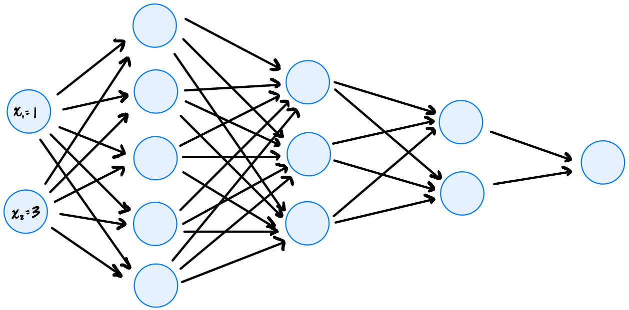
- Train a deep network to classify the data below.

- Hidden layers will learn a new feature map that makes the data linearly separable.

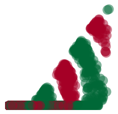


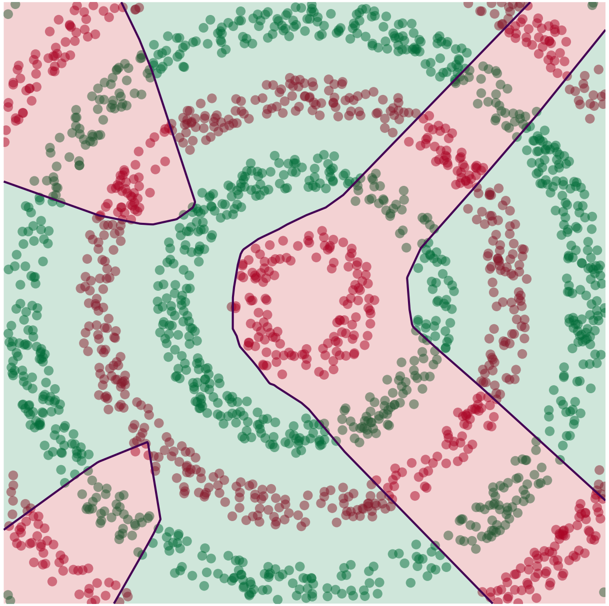
Demo

- 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.

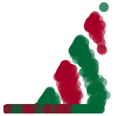


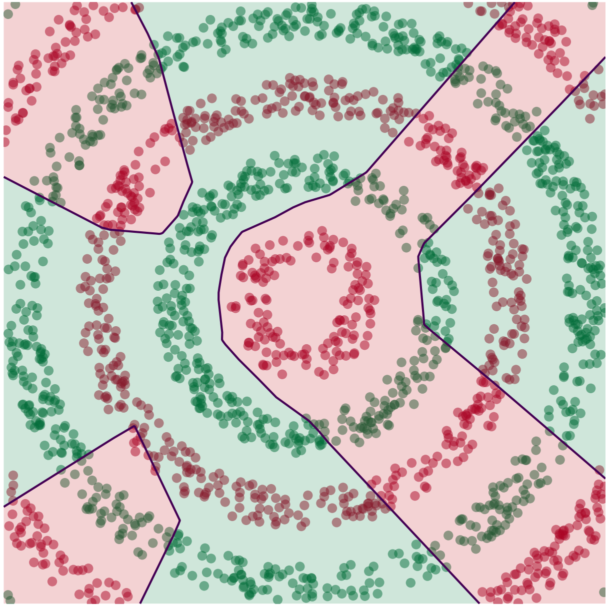
Learning a New Representation



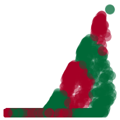


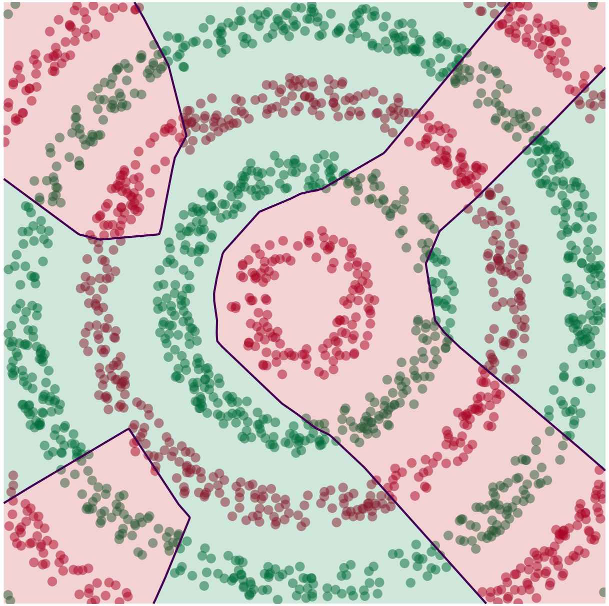
Learning a New Representation



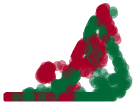


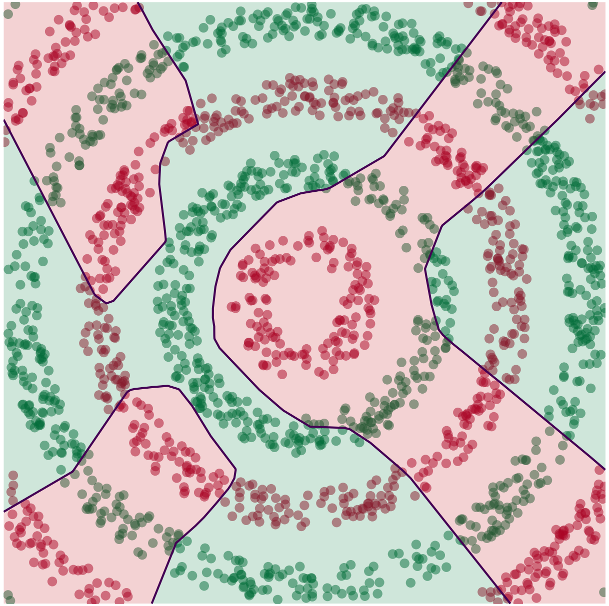
Learning a New Representation



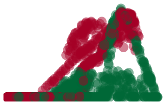


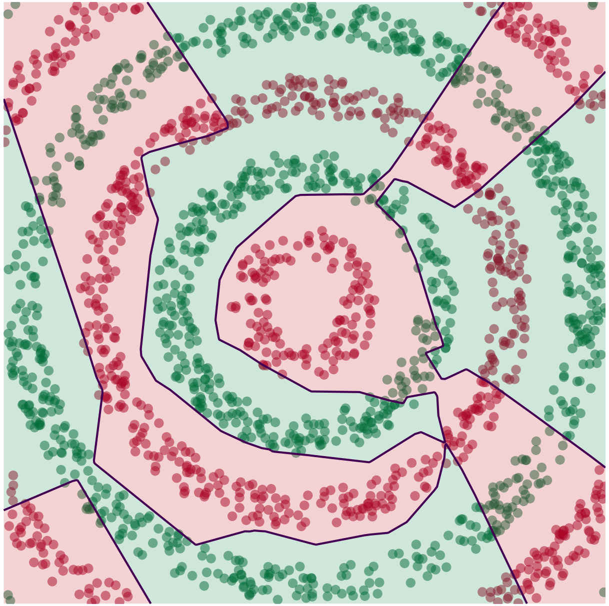
Learning a New Representation



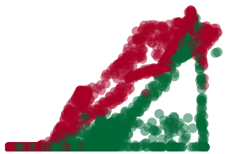


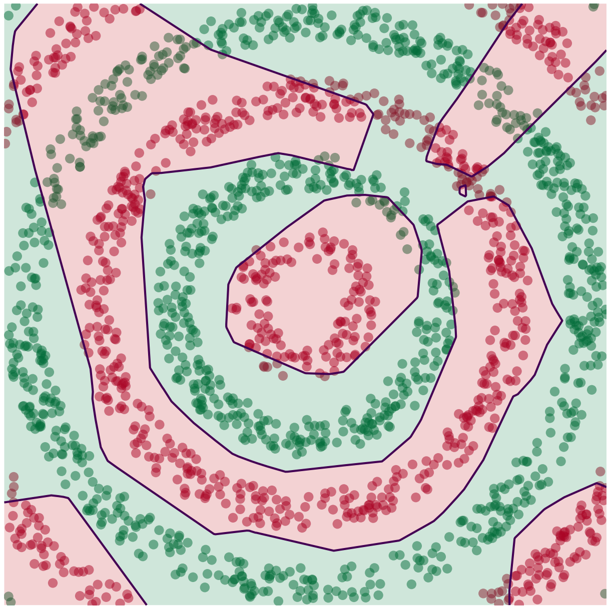
Learning a New Representation



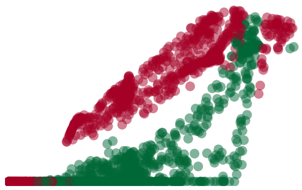


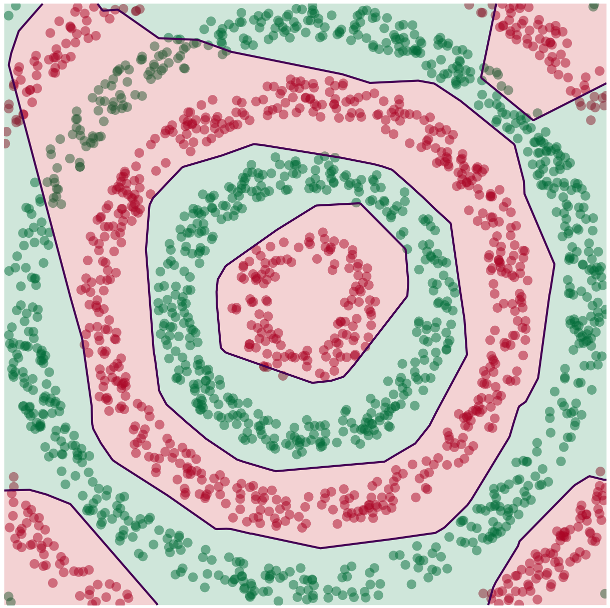
Learning a New Representation



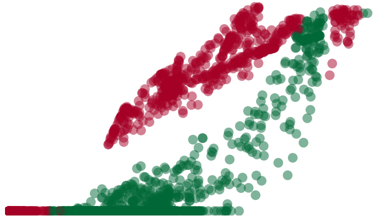


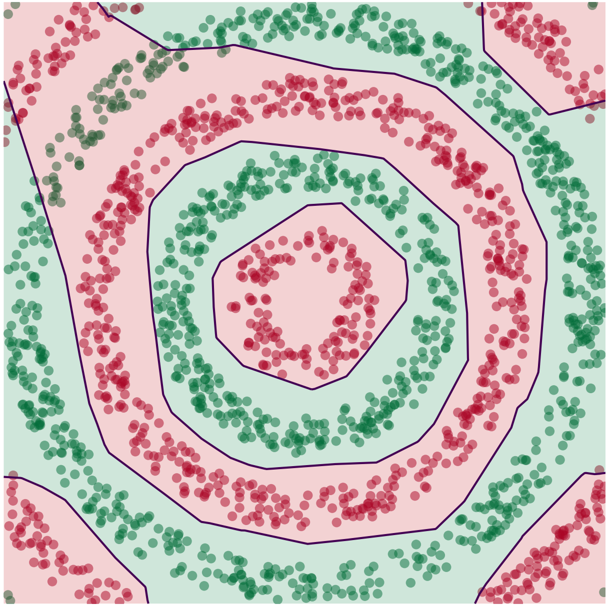
Learning a New Representation



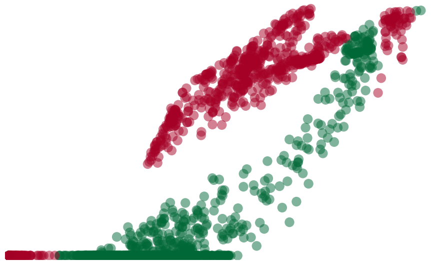


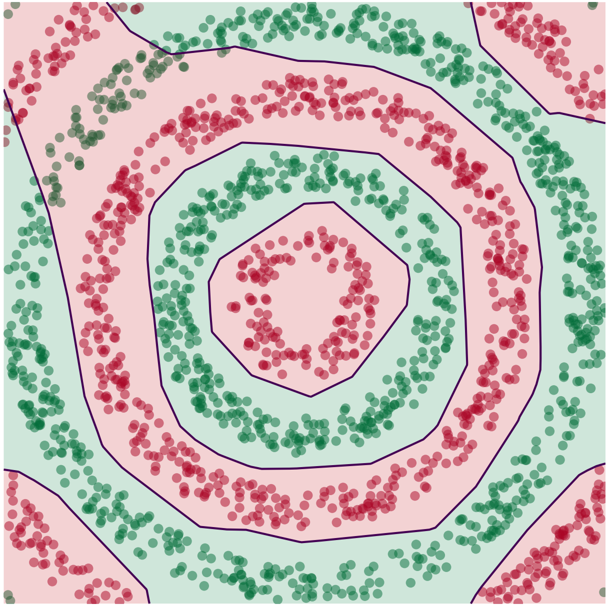
Learning a New Representation



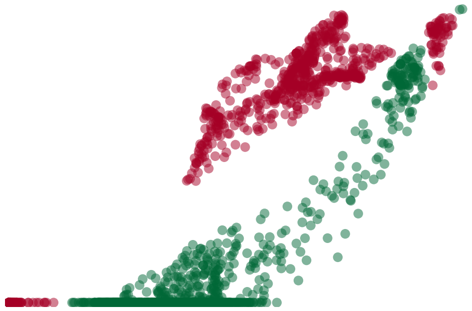


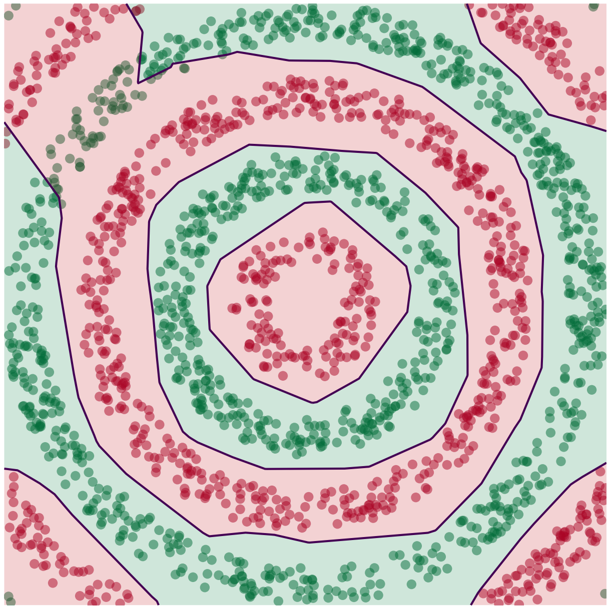
Learning a New Representation



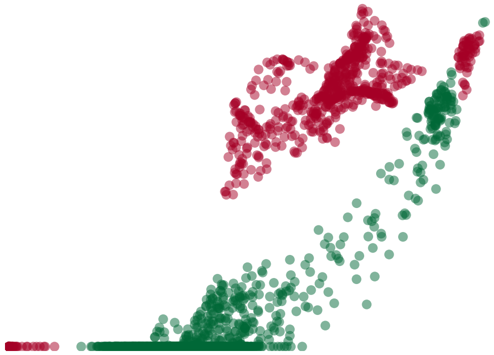


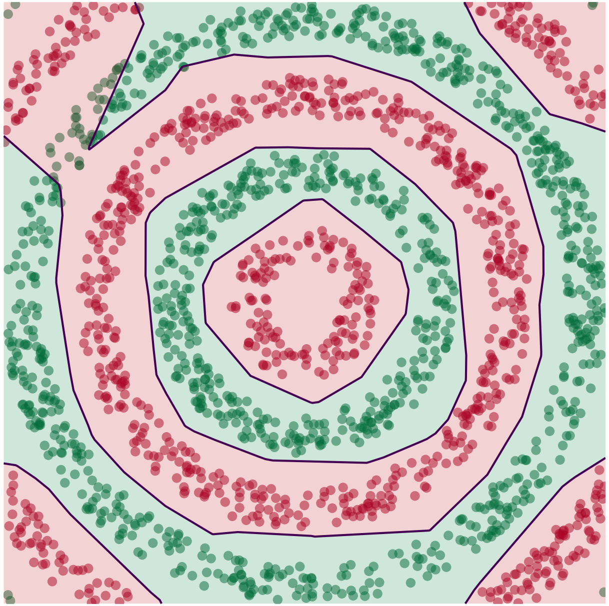
Learning a New Representation



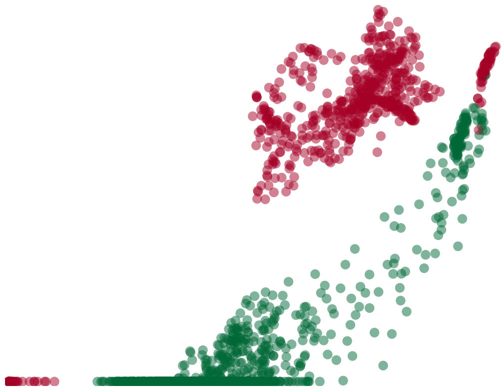


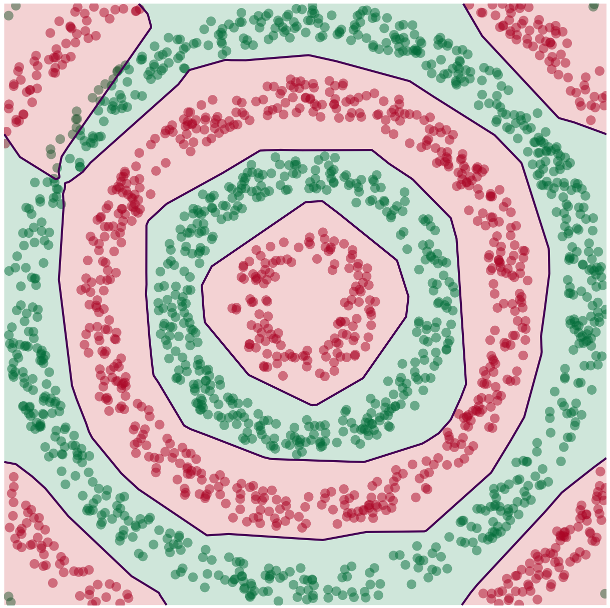
Learning a New Representation



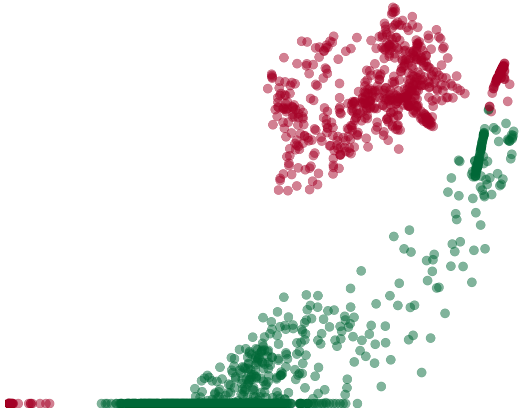


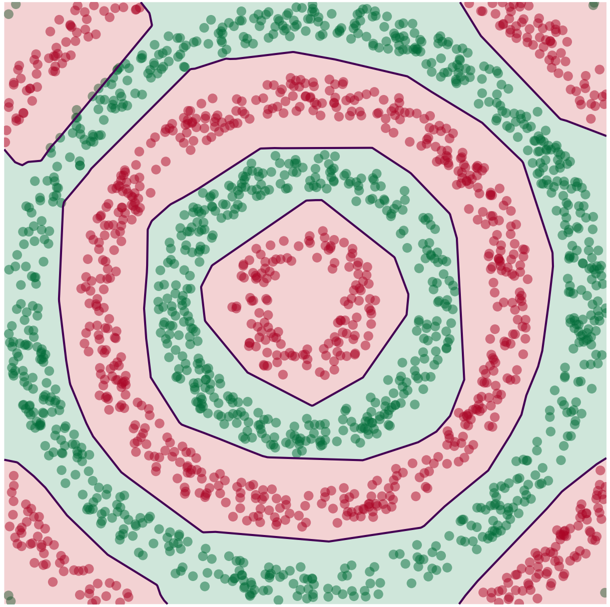
Learning a New Representation



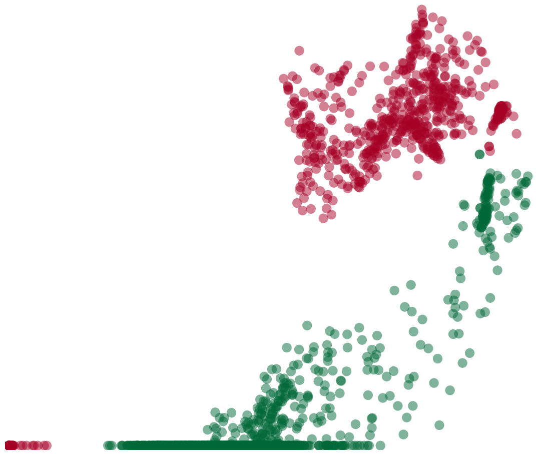


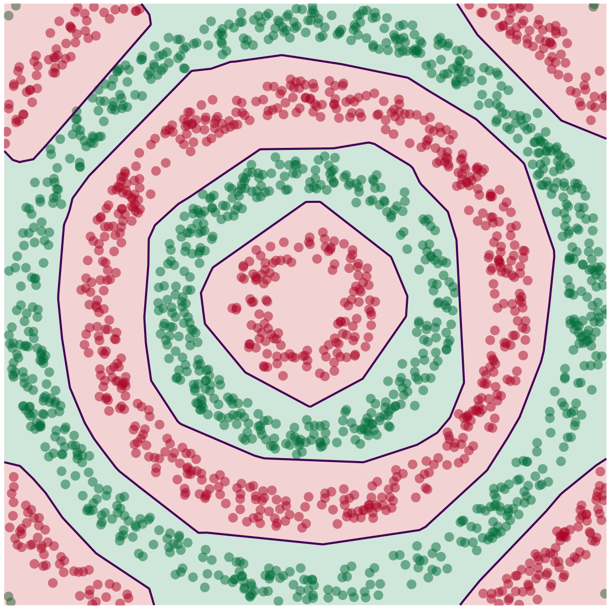
Learning a New Representation



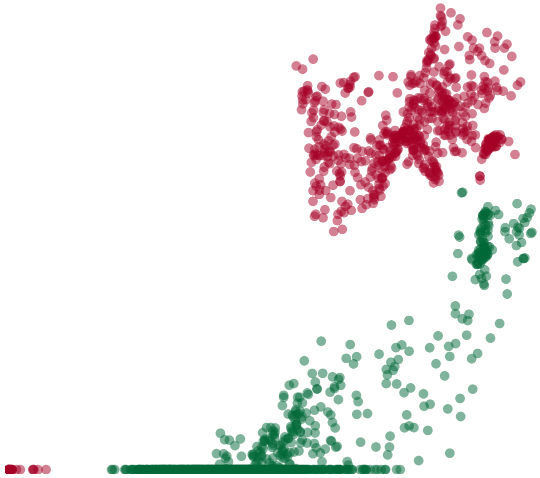


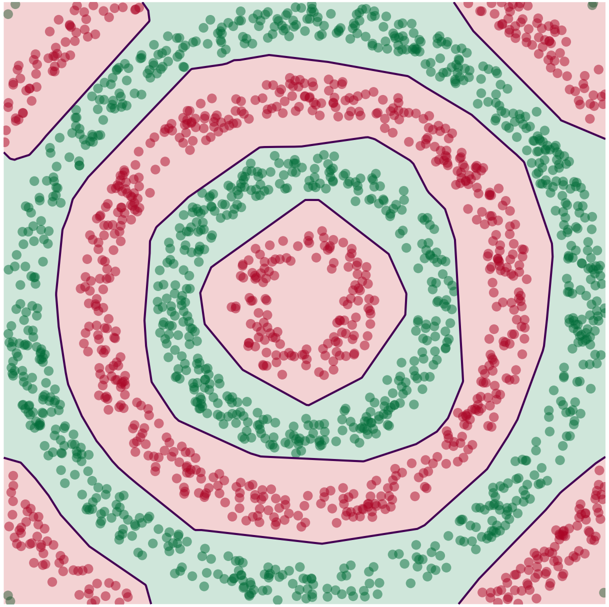
Learning a New Representation



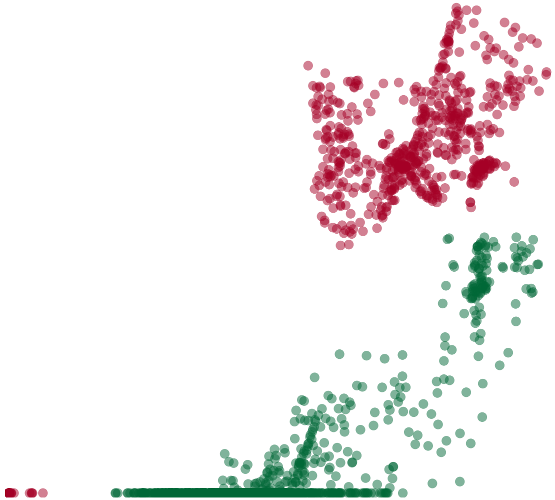


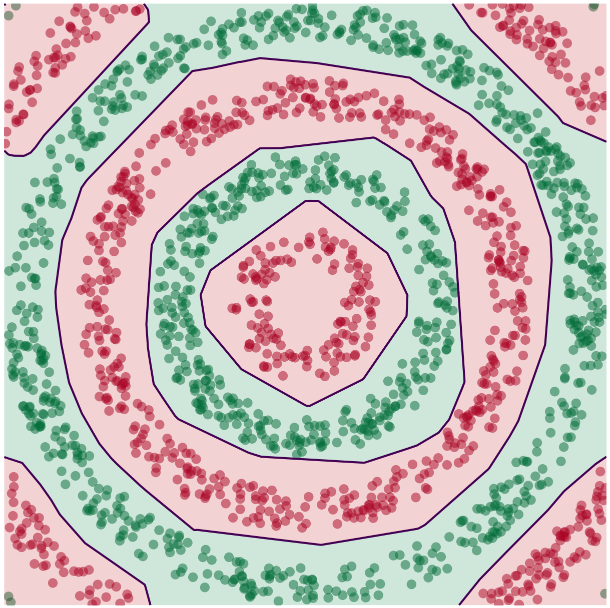
Learning a New Representation



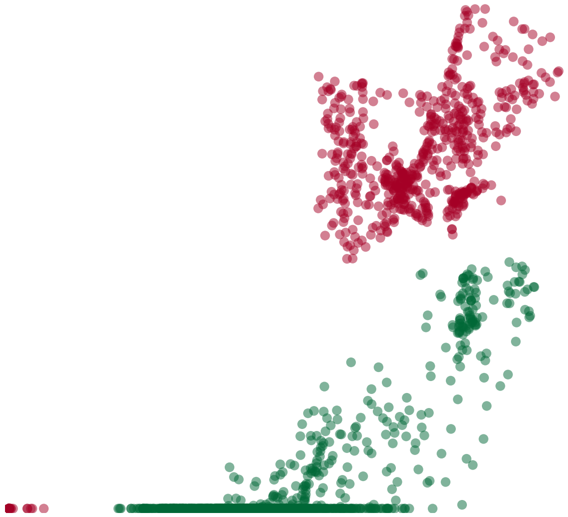


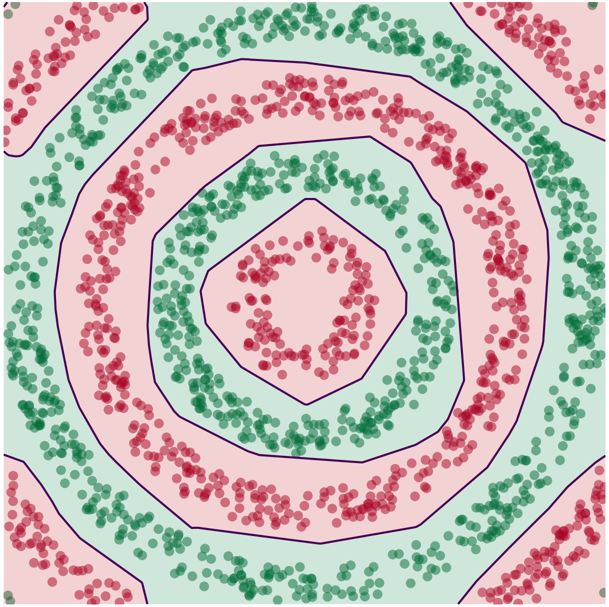
Learning a New Representation



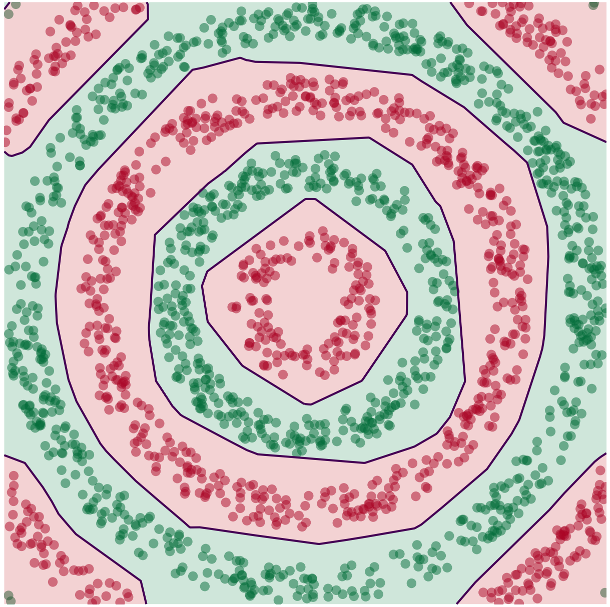


Learning a New Representation

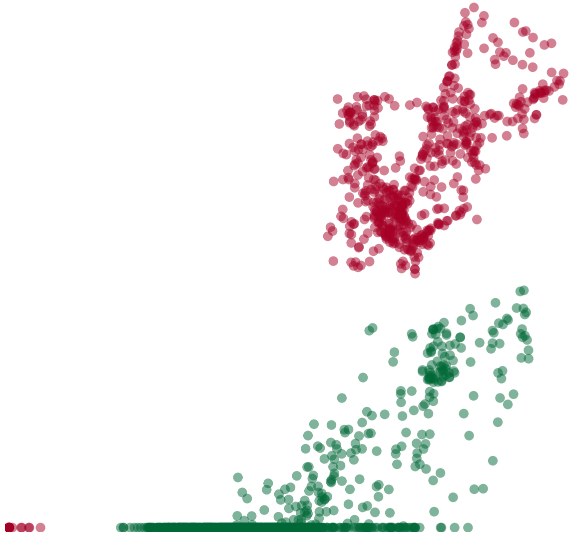


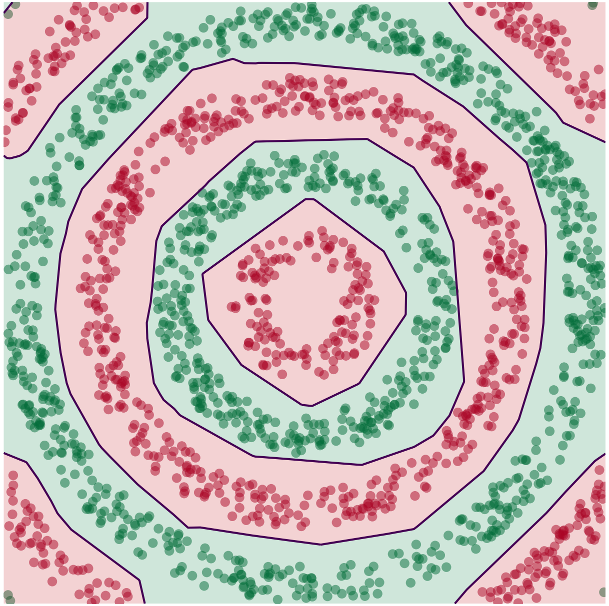


Learning a New Representation

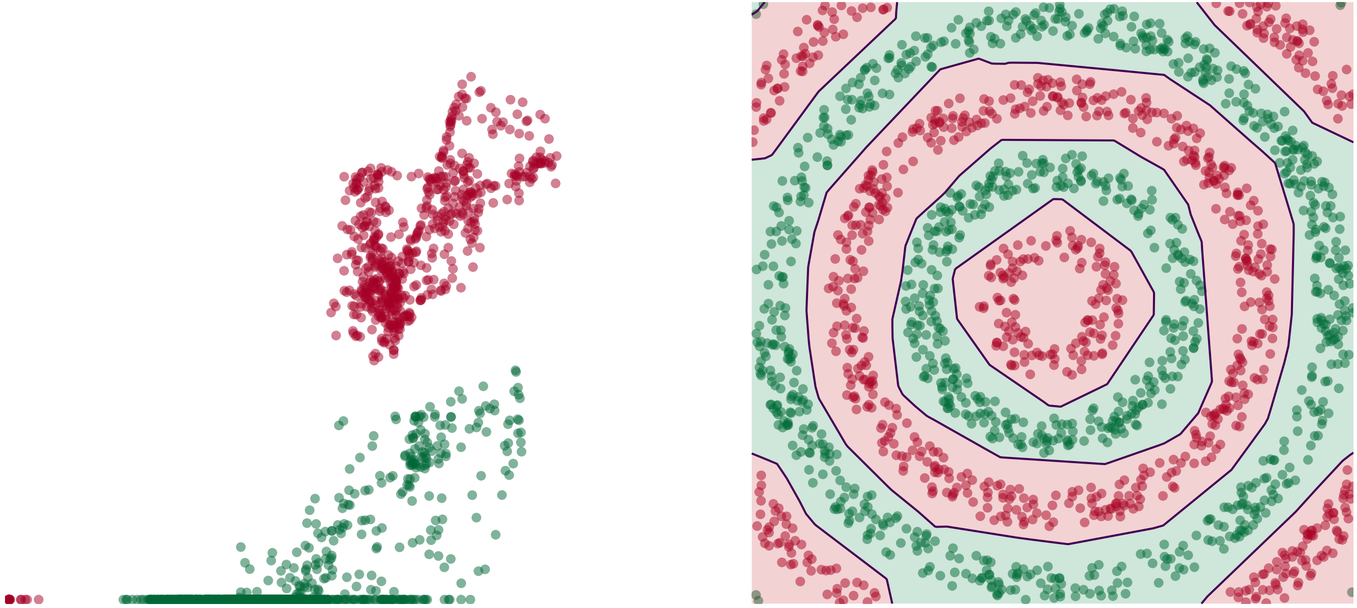


Learning a New Representation

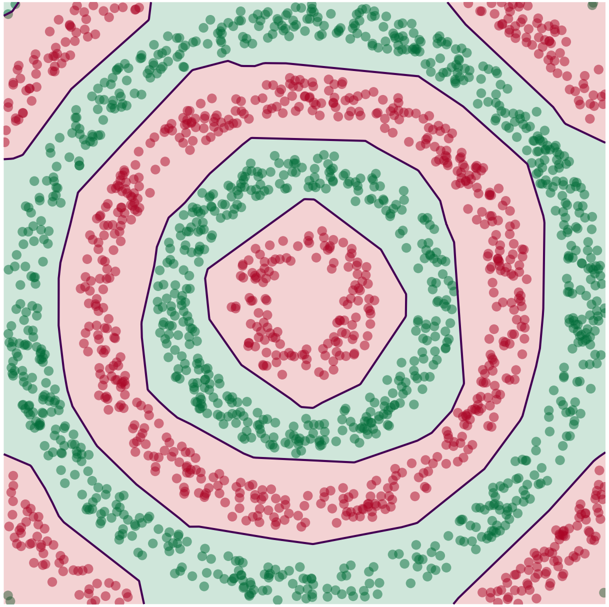




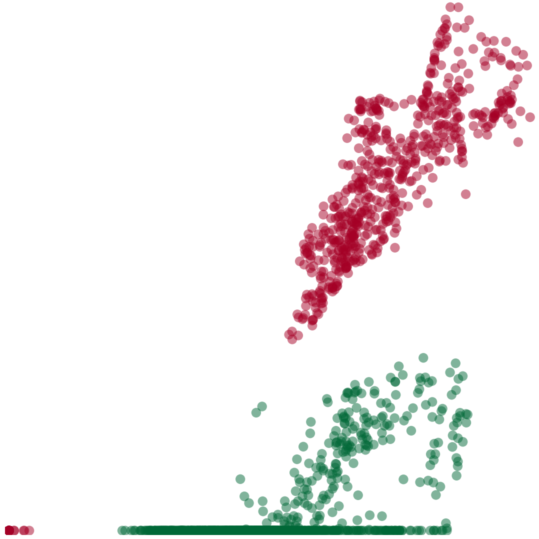
Learning a New Representation

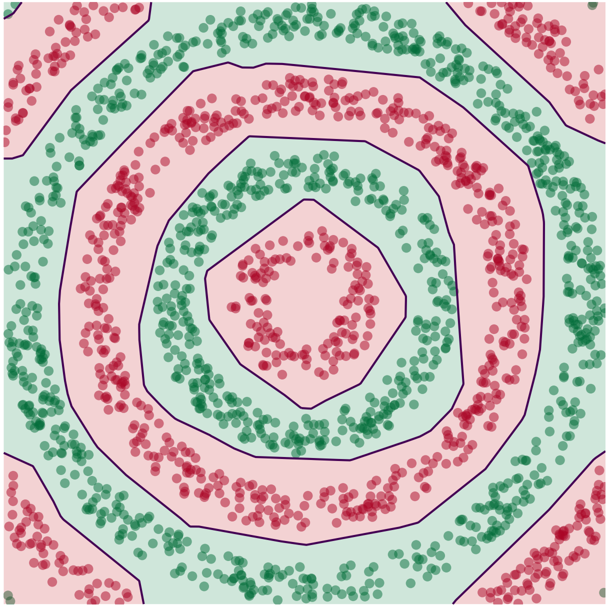


Learning a New Representation

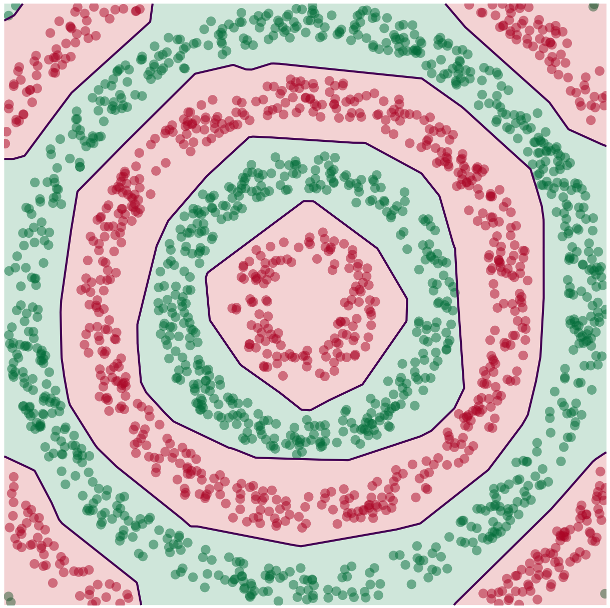


Learning a New Representation

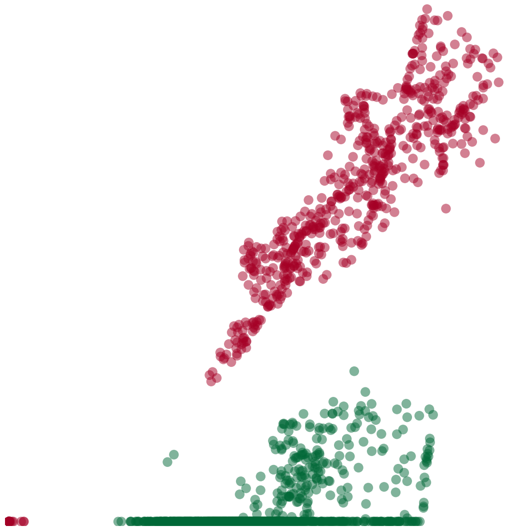


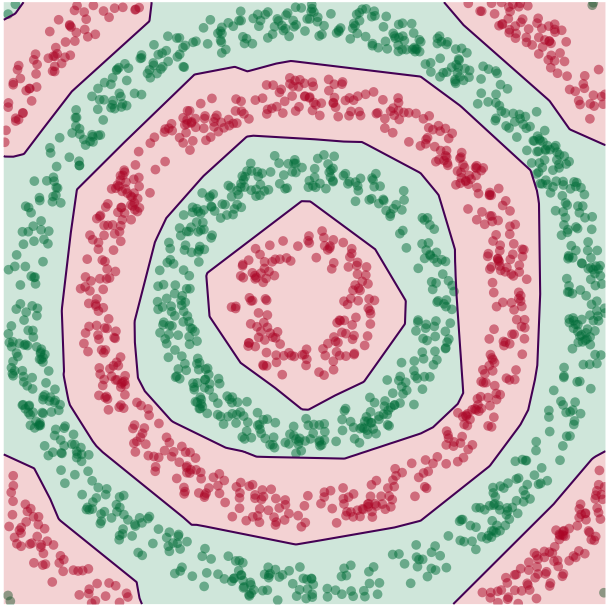


Learning a New Representation

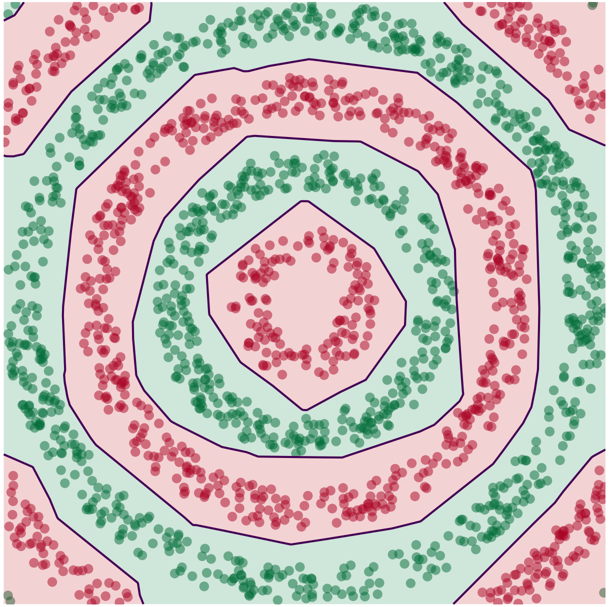


Learning a New Representation

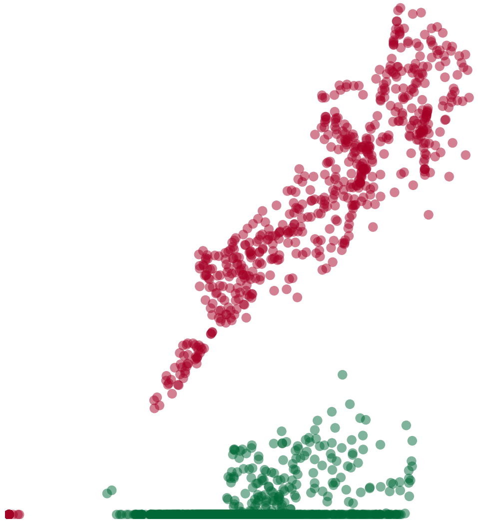


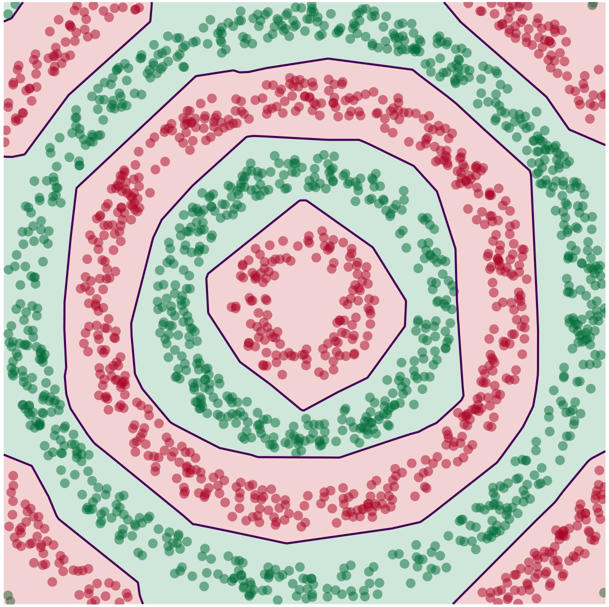


Learning a New Representation

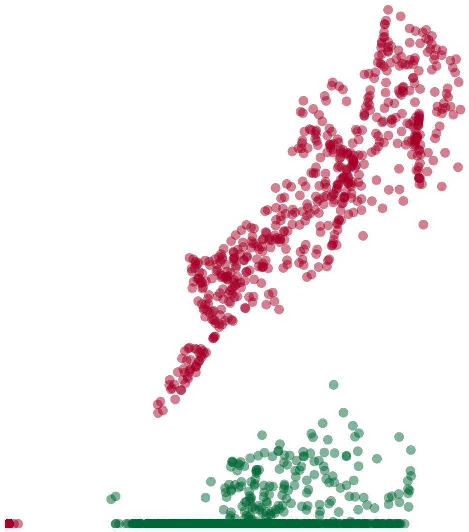


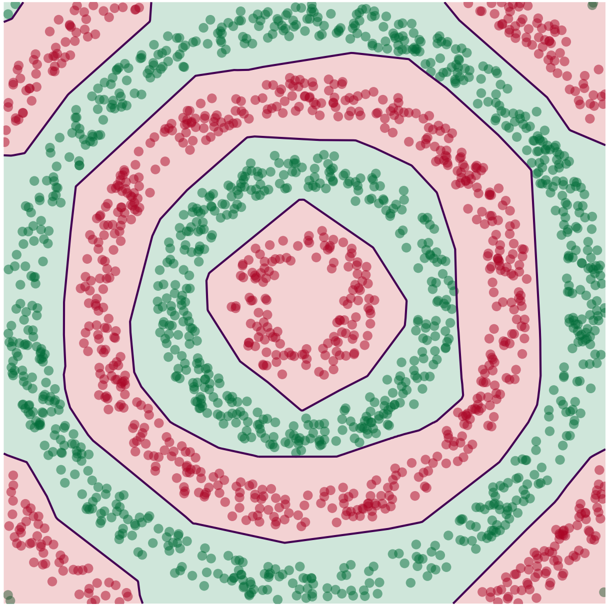
Learning a New Representation



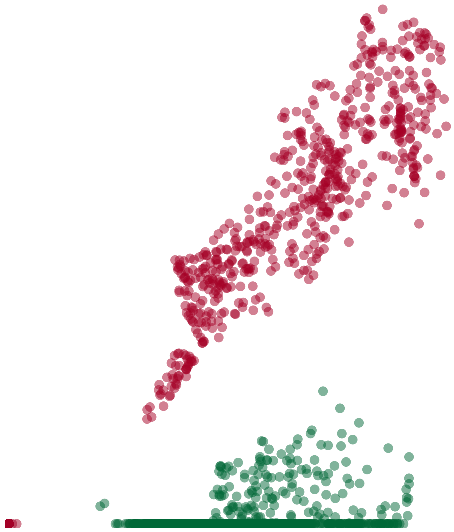


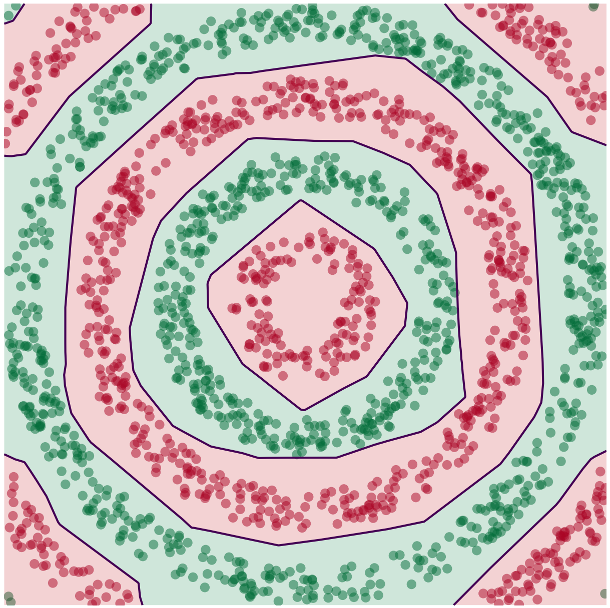
Learning a New Representation



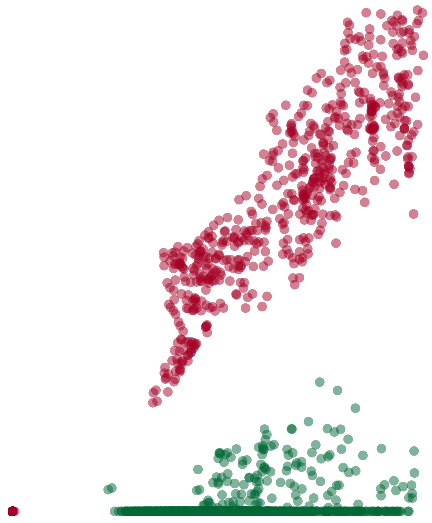


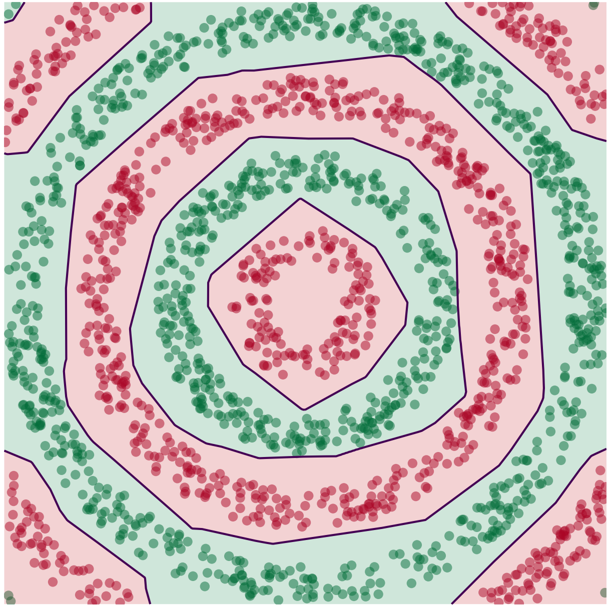
Learning a New Representation



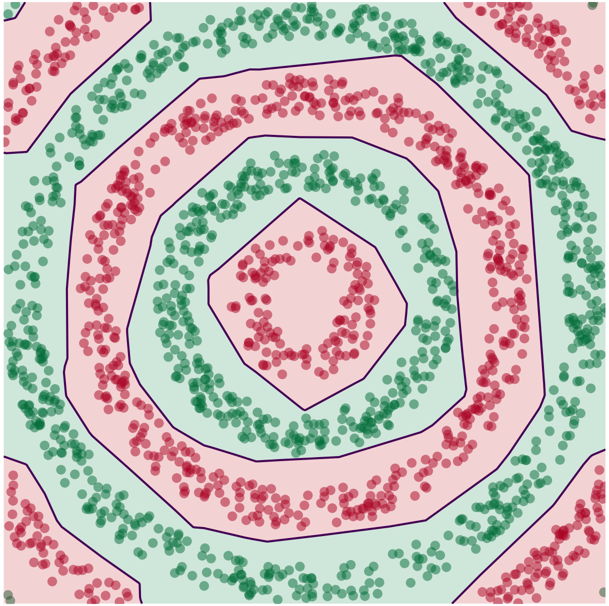


Learning a New Representation





Learning a New Representation



Deep Learning

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