

DSC 140B

Representation Learning

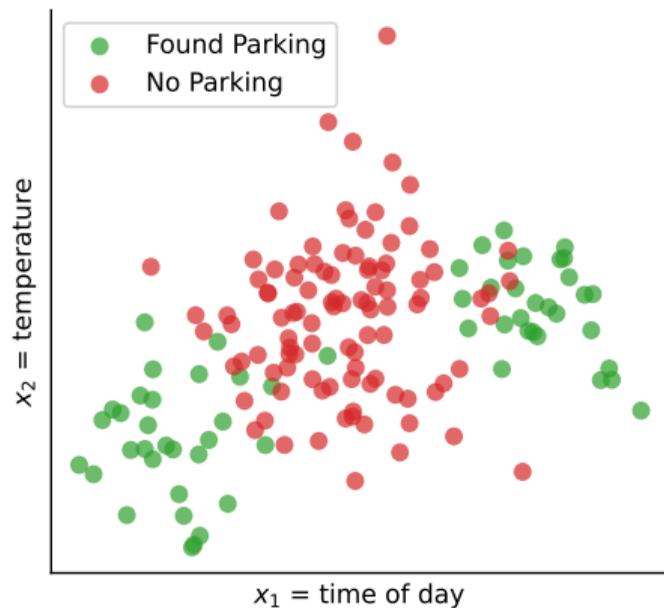
Lecture 10 | Part 1

Recap: Feature Maps

Problem

- ▶ Linear predictors only learn **linear** patterns.
 - ▶ Straight lines / planes.
- ▶ Patterns in real world data are often **non-linear**.

Example: Classification



Idea

- ▶ The pattern is **non-linear** in its original representation.
- ▶ **Idea:**
 1. Change to a new representation where the pattern is **linear**.
 2. Train a linear predictor in the new representation.

Idea

- ▶ To change representation, we use non-linear **feature maps**.

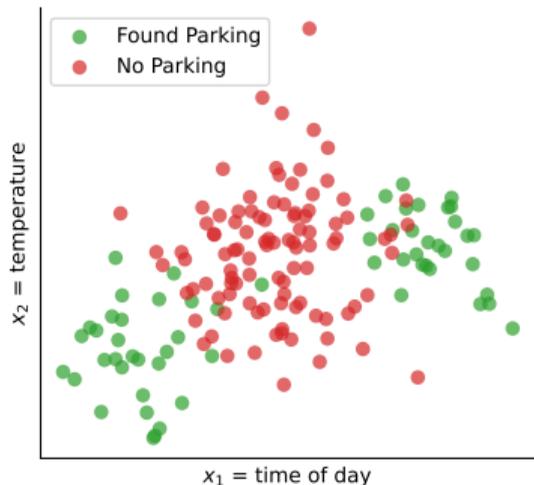
$$\vec{\phi}(\vec{x}) = (\phi_1(\vec{x}), \phi_2(\vec{x}), \dots, \phi_k(\vec{x}))^T$$

- ▶ A (good) feature map can turn **non-linear** patterns into **linear** patterns.

Procedure: Learning with Feature Maps

- ▶ First, pick a feature map $\vec{\phi} : \mathbb{R}^d \rightarrow \mathbb{R}^k$.
- ▶ **To train:**
 - ▶ Given training set $(\vec{x}^{(1)}, y_1), \dots, (\vec{x}^{(n)}, y_n)$.
 - 1. Map each $\vec{x}^{(i)}$ to feature space, creating a new data set $(\vec{\phi}(\vec{x}^{(1)}), y_1), \dots, (\vec{\phi}(\vec{x}^{(n)}), y_n)$.
 - 2. Train linear model on the new data in feature space to get \vec{w}^* .
- ▶ **To predict:**
 - ▶ Given new input \vec{x} .
 - 1. Map \vec{x} to feature space: $\vec{\phi}(\vec{x})$.
 - 2. Predict $H(\vec{x}; \vec{w}^*) = \vec{w}^* \cdot \text{Aug}(\vec{\phi}(\vec{x}))$.

Example: Parking Prediction



- ▶ Original features:

$$\vec{x} = (\text{time}, \text{temp.})^T$$

- ▶ Feature map:

$$\vec{\phi}(\vec{x}) = (|\text{time} - \text{Noon}|, |\text{temp.} - 70|)^T$$

- ▶ We'll train a **least squares classifier** in feature space.

Step 1: Pick a Feature Map

- ▶ In the input space, we have features (x_1, x_2) .

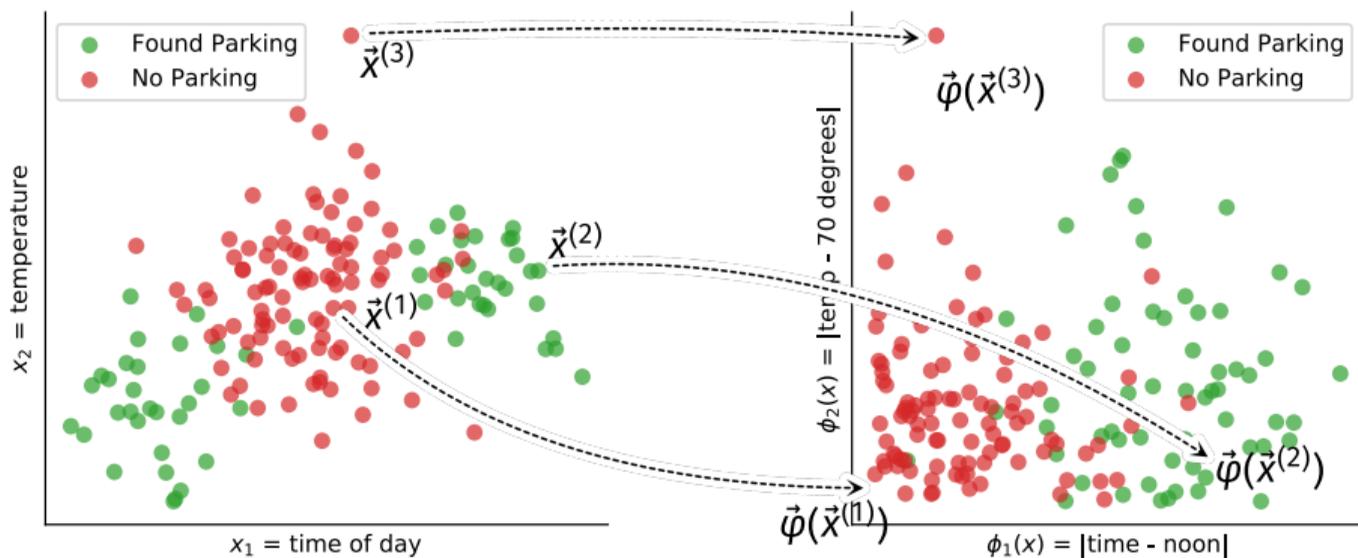
$$x_1 = \text{time}, \quad x_2 = \text{temperature}.$$

- ▶ We'll use the same feature map as before:

$$\vec{\phi}(x_1, x_2) = (|x_1 - 12|, |x_2 - 70|)^T$$

Step 2(a): Map to Feature Space

- ▶ Map every data point to feature space.



Step 2(b): Train in Feature Space

- ▶ In feature space, our feature vectors are $\vec{\phi}(\vec{x}^{(1)}), \dots, \vec{\phi}(\vec{x}^{(n)})$.
- ▶ So the design matrix becomes the $(n \times k)$ matrix:

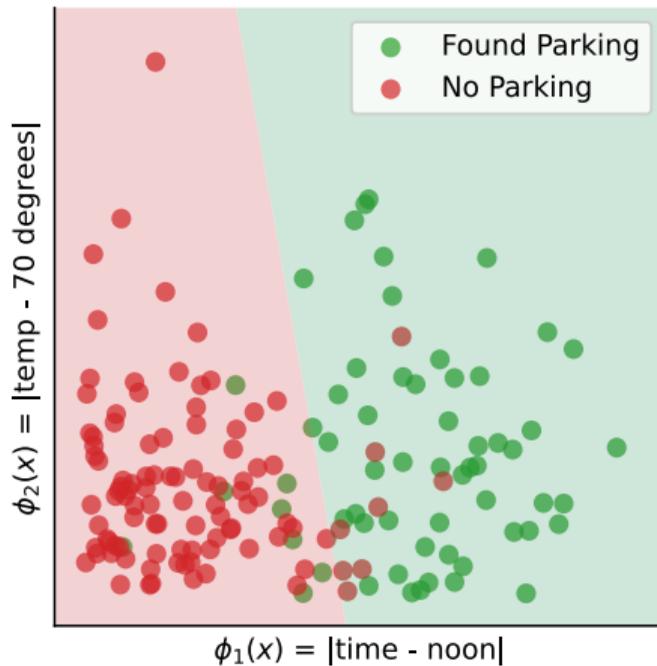
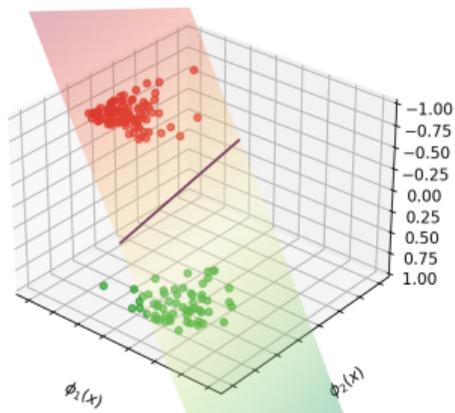
$$\Phi = \begin{pmatrix} \text{Aug}(\vec{\phi}(\vec{x}^{(1)})) \longrightarrow \\ \text{Aug}(\vec{\phi}(\vec{x}^{(2)})) \longrightarrow \\ \vdots \\ \text{Aug}(\vec{\phi}(\vec{x}^{(n)})) \longrightarrow \end{pmatrix} = \begin{pmatrix} 1 & |x_1^{(1)} - 12| & |x_2^{(1)} - 70| \\ 1 & |x_1^{(2)} - 12| & |x_2^{(2)} - 70| \\ \vdots & \vdots & \vdots \\ 1 & |x_1^{(n)} - 12| & |x_2^{(n)} - 70| \end{pmatrix}$$

Step 2(b): Train in Feature Space

- ▶ The least squares solution in feature space is:

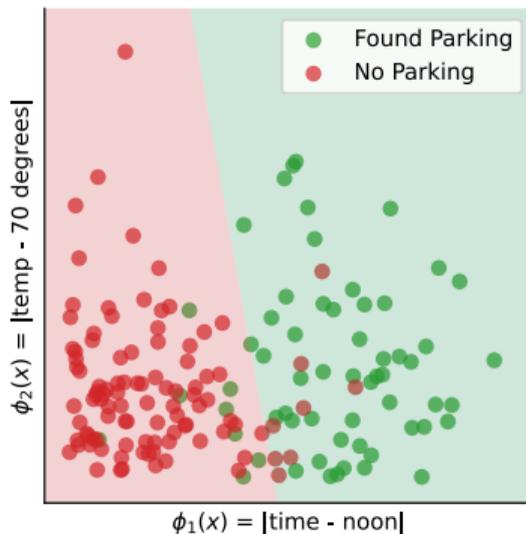
$$\vec{w}^* = (\Phi^T \Phi)^{-1} \Phi^T \vec{y}$$

Solution in Feature Space



Step 3: Predict

- ▶ Given a new example \vec{x} in input space:
 1. Map \vec{x} to feature space: $\vec{\phi}(\vec{x})$.
 2. Predict $\text{sign}(\vec{w}^* \cdot \text{Aug}(\vec{\phi}(\vec{x})))$.



$$\vec{x} \rightarrow \vec{\phi}(\vec{x}) = (|10-12|, |65-70|)^T \\ = (2, 5)^T$$

Exercise

Let $\vec{\phi}(x_1, x_2) = (|x_1 - 12|, |x_2 - 70|)^T$. Suppose we train a least squares classifier in feature space and find $\vec{w}^* = (3, -1, 2)^T$.

Given a new point $\vec{x} = (10, 65)^T$ in input space, what is the prediction, $H(\vec{x})$?

$$\begin{aligned} (3, -1, 2)^T \cdot (1, 2, 5)^T &= 3 - 2 + 10 \\ \vec{w} \cdot A_y(\vec{\phi}(\vec{x})) &= 11 \rightarrow 1 \end{aligned}$$

The Prediction Function(s)

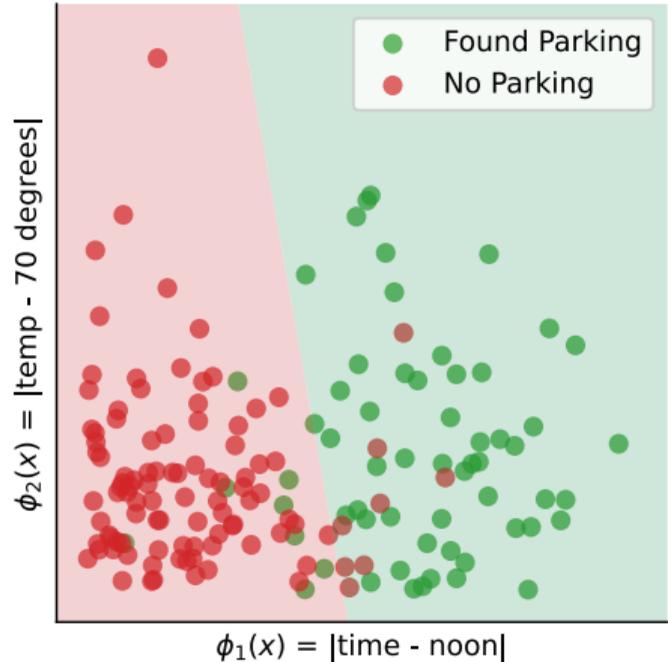
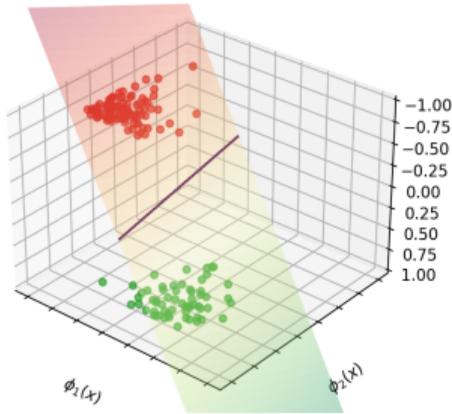
- ▶ There are, in a sense, *two* prediction functions:
 - ▶ H , which accepts inputs in **input** space.
 - ▶ H_ϕ , which accepts inputs in **feature** space.

- ▶ First, the prediction function in **feature space**:

$$\begin{aligned}H_\phi(\vec{Z}) &= \vec{W} \cdot \text{Aug}(\vec{Z}) \\ &= W_0 + W_1 Z_1 + W_2 Z_2 + \dots + W_k Z_k\end{aligned}$$

H_ϕ in Feature Space

$$H_\phi(\vec{z}) = w_0 + w_1 z_1 + w_2 z_2$$



The Prediction Function

- ▶ There is also the prediction function $H(\vec{x})$ that takes in vectors in **input space**.

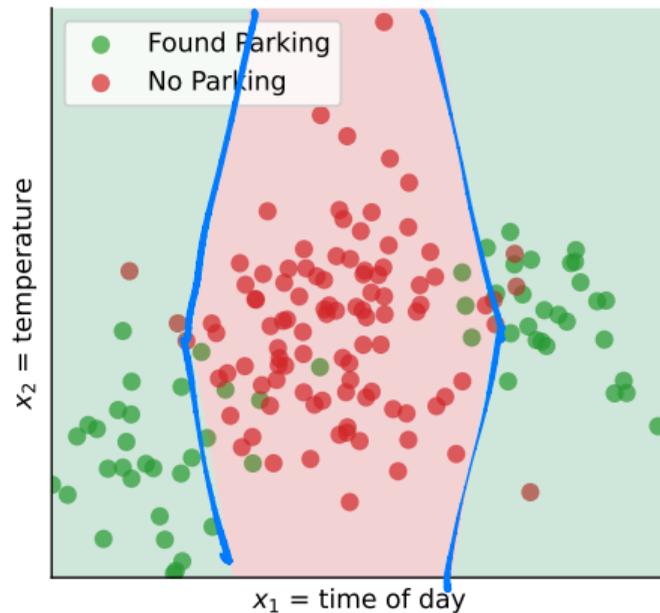
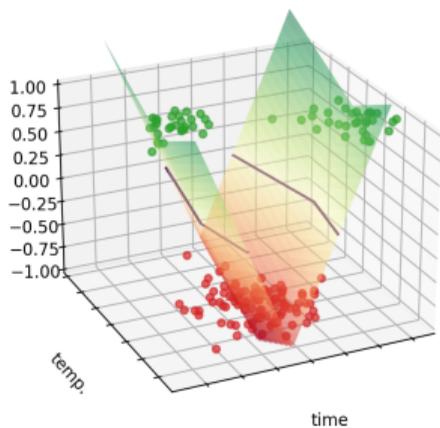
$$\begin{aligned} H(\vec{x}) &= H_{\phi}(\vec{\phi}(\vec{x})) \\ &= \vec{w} \cdot \text{Aug}(\vec{\phi}(\vec{x})) \\ &= w_0 + w_1 \phi_1(\vec{x}) + w_2 \phi_2(\vec{x}) + \dots + w_k \phi_k(\vec{x}) \end{aligned}$$

← temp - 701

- ▶ When plotted, this function will look **non-linear**.

H in Input Space

$$H(\vec{x}) = w_0 + w_1|x_1 - 12| + w_2|x_2 - 70|$$



$$\begin{aligned} H(\vec{x}) &= w_0 + w_1 |x_1 - 12| + w_2 |x_2 - 70| \\ &= 3 + (-1) |10 - 12| + (2) |65 - 70| \end{aligned}$$

Exercise

Let $\vec{\phi}(x_1, x_2) = (|x_1 - 12|, |x_2 - 70|)^T$. Suppose we train a least squares classifier in feature space and find $\vec{w}^* = (3, -1, 2)^T$.

Given a new point $\vec{x} = (10, 65)^T$ in input space, what is the prediction, $H(\vec{x})$? This time, compute the answer *without* explicitly computing $\vec{\phi}(\vec{x})$.

$$= 3 - 2 + 10$$

$$= 11$$

Problem

- ▶ Feature maps can learn non-linear patterns.
- ▶ But so far, we've found them **manually**.
- ▶ How do we find a good one **automatically**?

DSC 140B

Representation Learning

Lecture 10 | Part 2

Radial Basis Functions

Overview: Feature Mapping

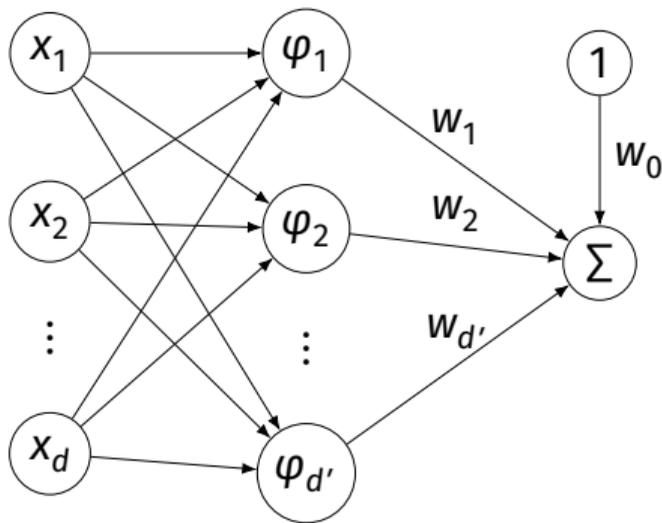
1. Start with data in original space, \mathbb{R}^d .
2. Choose some basis functions, $\varphi_1, \varphi_2, \dots, \varphi_{d'}$
3. Map each data point to **feature space** $\mathbb{R}^{d'}$:

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

4. Fit linear prediction function in new space:

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

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

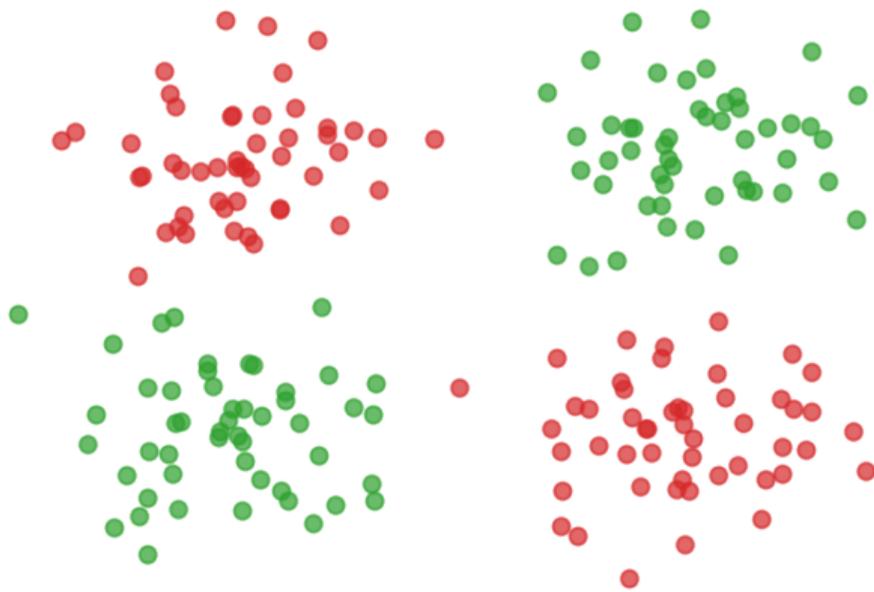


$$\vec{w} \cdot \vec{\varphi}(\vec{x}) = w_1 \varphi_1(\vec{x}) + w_2 \varphi_2(\vec{x}) + \dots$$

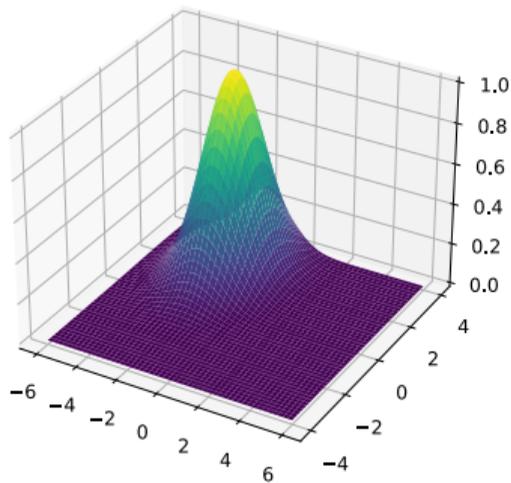
Generic Basis Functions

- ▶ 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. $e^{-\|\vec{x}-\vec{\mu}\|^2/\sigma^2}$
- ▶ 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 $\vec{\mu}^{(1)}, \dots, \vec{\mu}^{(2)}, \dots, \vec{\mu}^{(d')}$
- ▶ Pick widths: $\sigma_1, \sigma_2, \dots, \sigma_{d'}$ (usually all the same)
- ▶ Define i th basis function:

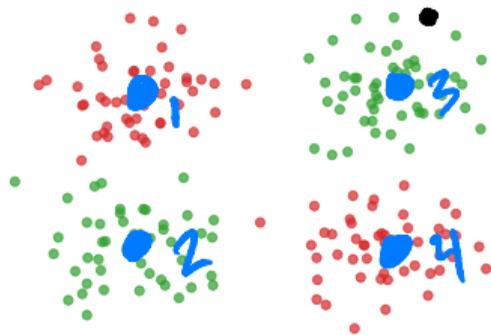
$$\varphi_i(\vec{x}) = e^{-\|\vec{x} - \vec{\mu}^{(i)}\|^2 / \sigma_i^2}$$

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.

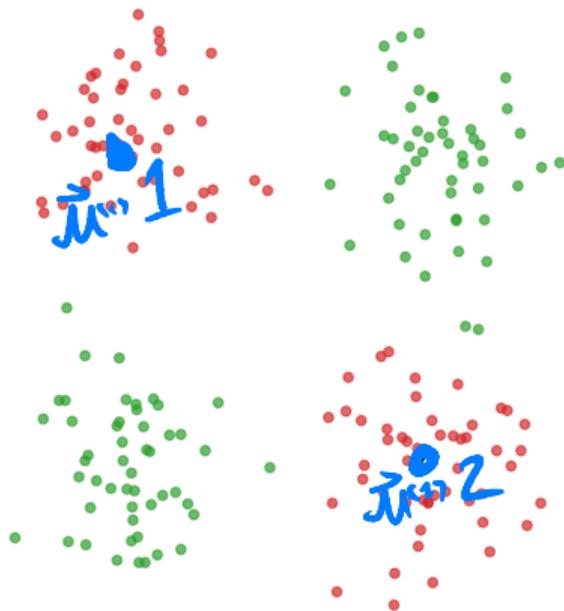
Exercise

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

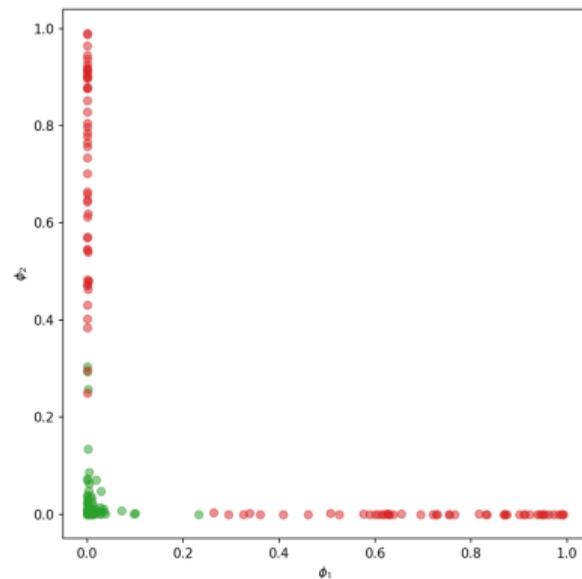
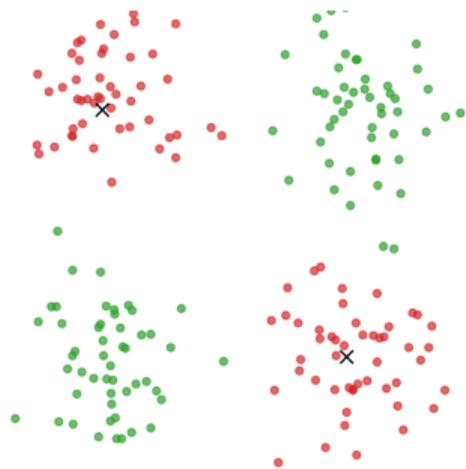


Placement

$$\vec{x} \mapsto (0.1, .1)$$



Feature Space



Prediction Function

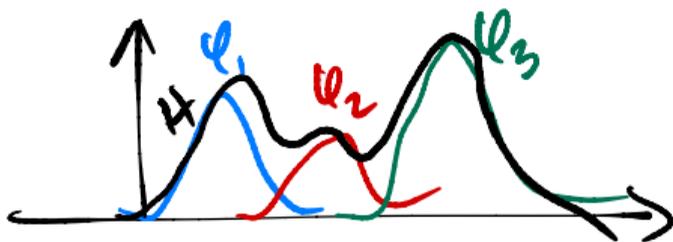
- ▶ $H(\vec{x})$ is a sum of Gaussians:

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

Exercise

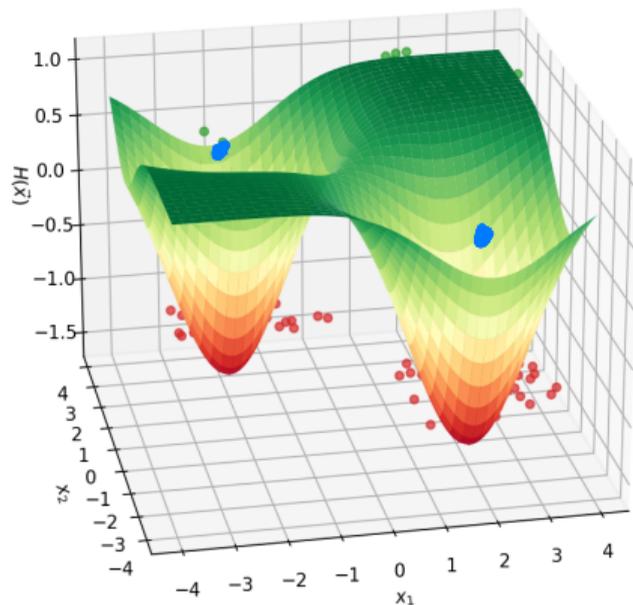
What does the surface of the prediction function look like?

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



$$H = \varphi_1 + \varphi_2 + \varphi_3$$

Prediction Function Surface

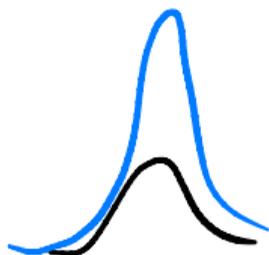


$$H(\vec{x}) = w_0 + w_1 e^{-\|\vec{x} - \vec{\mu}_1\|^2 / \sigma^2} + w_2 e^{-\|\vec{x} - \vec{\mu}_2\|^2 / \sigma^2}$$

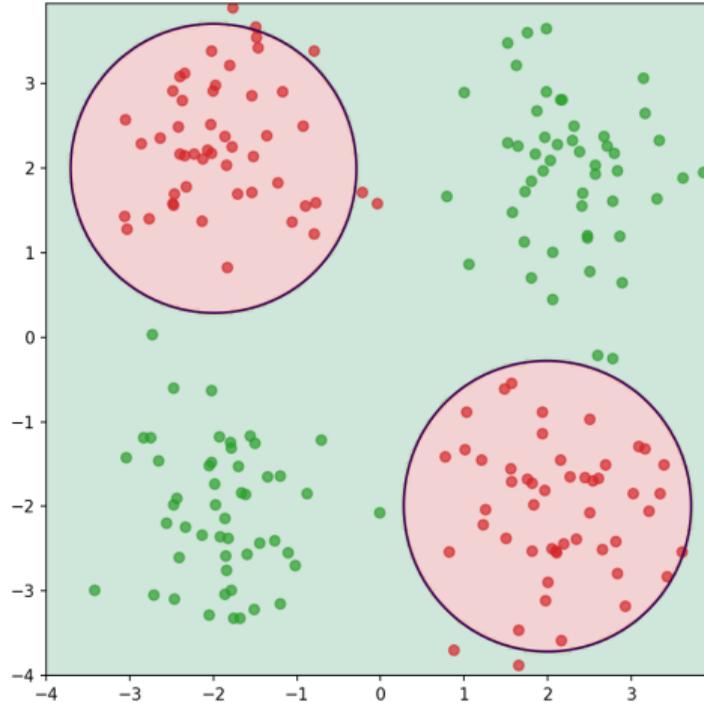
An Interpretation

- ▶ Basis function φ_i makes a “bump” in surface of H
- ▶ w_i adjusts the “prominence” of this bump

$$H = w_0 + w_1 e^{-\|\vec{x} - \vec{\mu}^{(1)}\|^2 / \sigma^2} + w_2 \dots$$

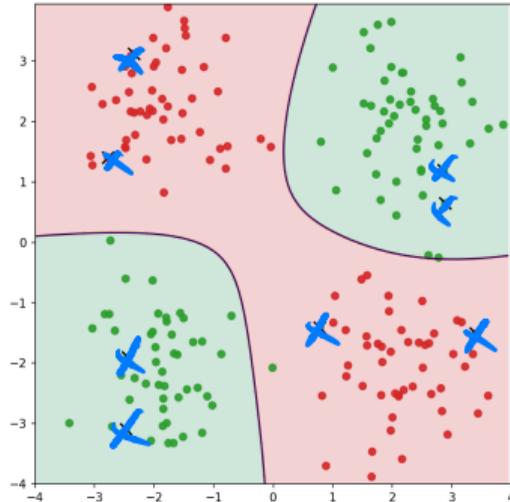


Decision Boundary

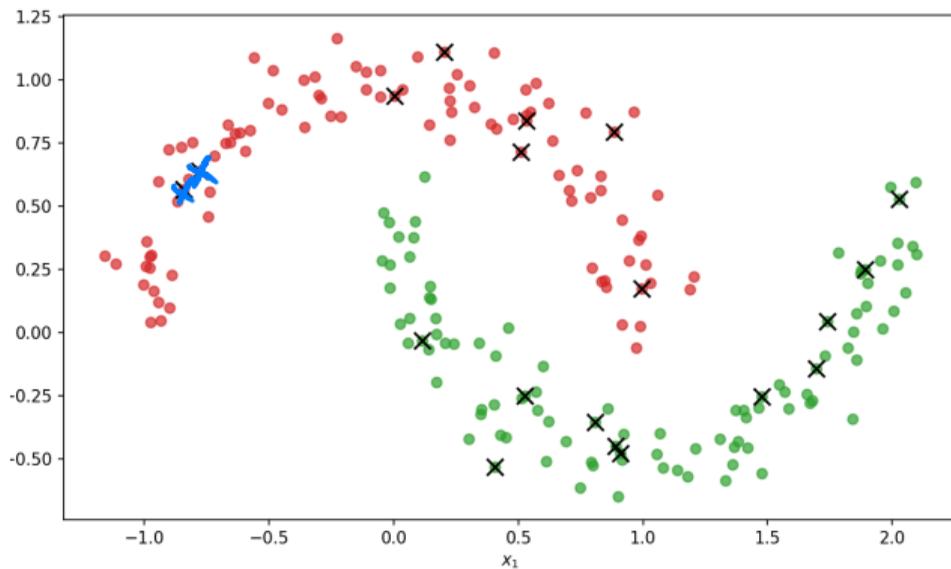


More Features

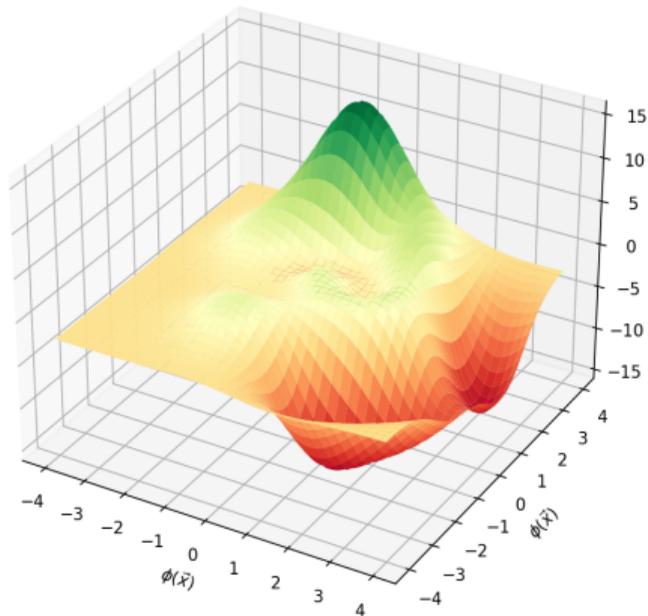
- ▶ By increasing number of basis functions, we can make more complex decision surfaces.



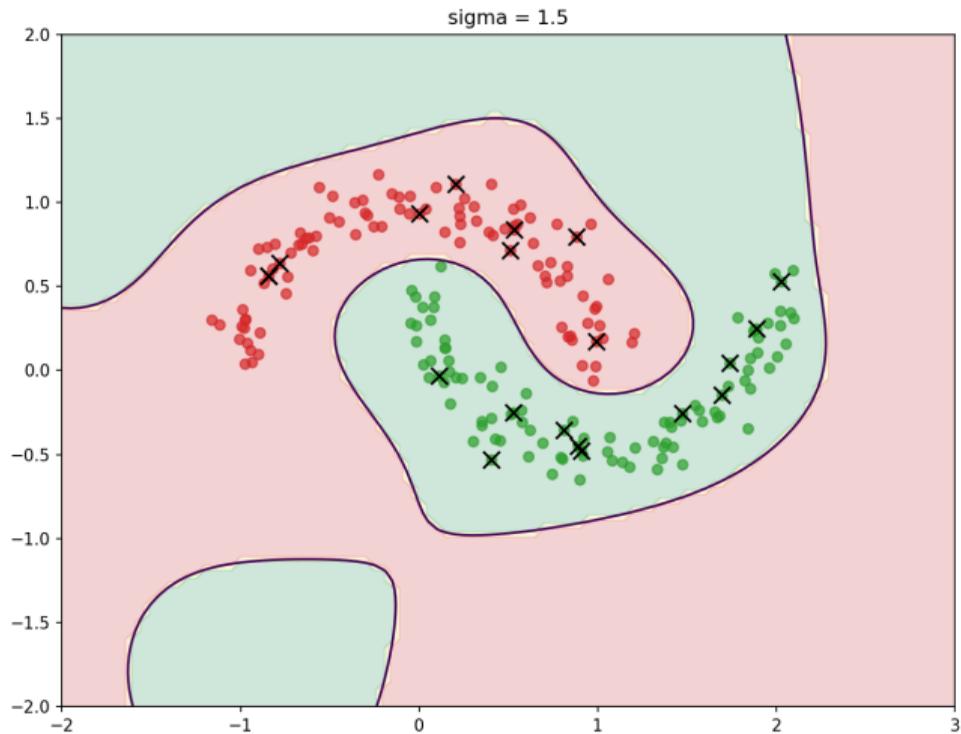
Another Example



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

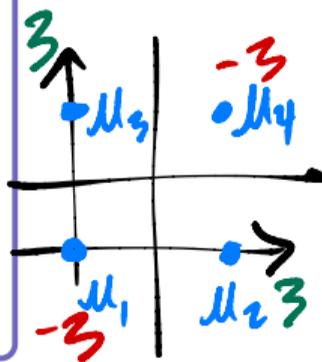
- ▶ **Multiquadric:** $\varphi(\vec{x}; \vec{c}) = \sqrt{\sigma^2 + \|\vec{x} - \vec{c}\|} / \sigma$

Live Q&A

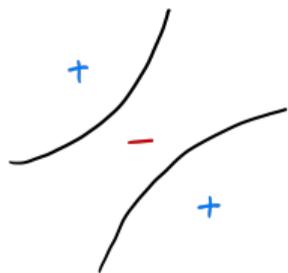
Exercise

Suppose we use four Gaussian RBFs (same σ) at $\vec{\mu}_1 = (0, 0)$, $\vec{\mu}_2 = (1, 0)$, $\vec{\mu}_3 = (0, 1)$, $\vec{\mu}_4 = (1, 1)$ and train a linear classifier in feature space, finding weights $w_1 = -3$, $w_2 = 3$, $w_3 = 3$, $w_4 = -3$.

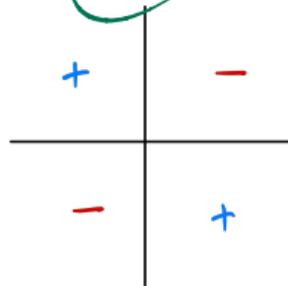
Which of the below could show the decision boundary of H in **input** space?



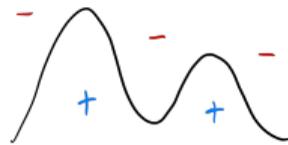
(a)



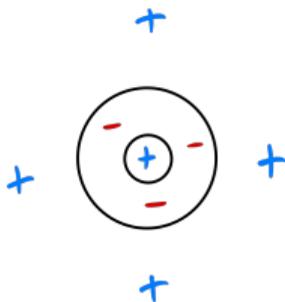
(b)



(c)



(d)



DSC 140B

Representation Learning

Lecture 10 | Part 3

Radial Basis Function Networks

Recap

1. Choose basis functions, $\varphi_1, \dots, \varphi_{d'}$
2. Transform data to new representation:

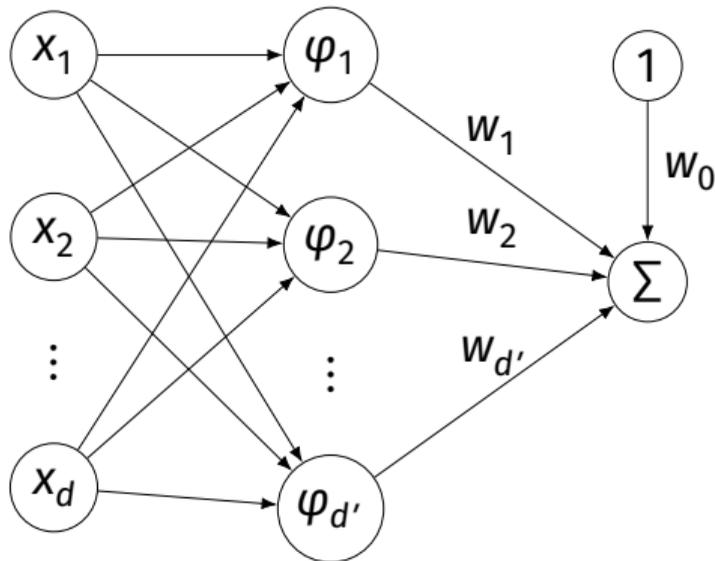
$$\vec{x} \mapsto (\varphi_1(\vec{x}), \varphi_2(\vec{x}), \dots, \varphi_{d'}(\vec{x}))^T$$

3. Train a linear classifier in this new space:

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

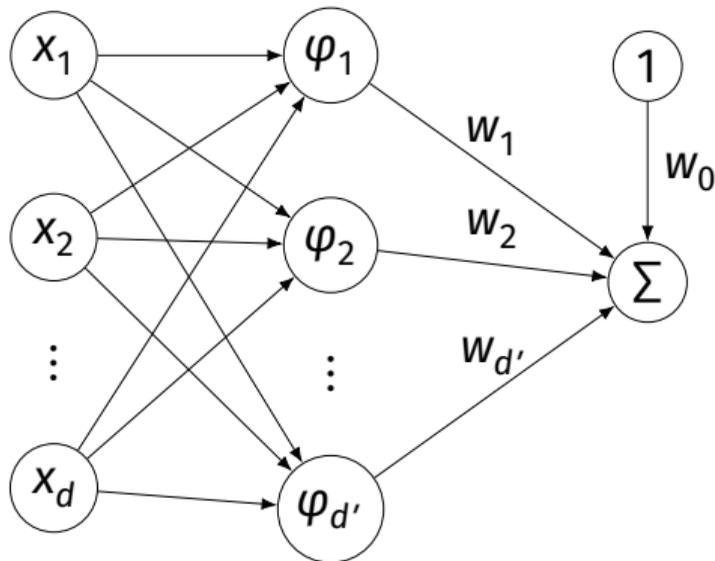
The Model

- ▶ 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?

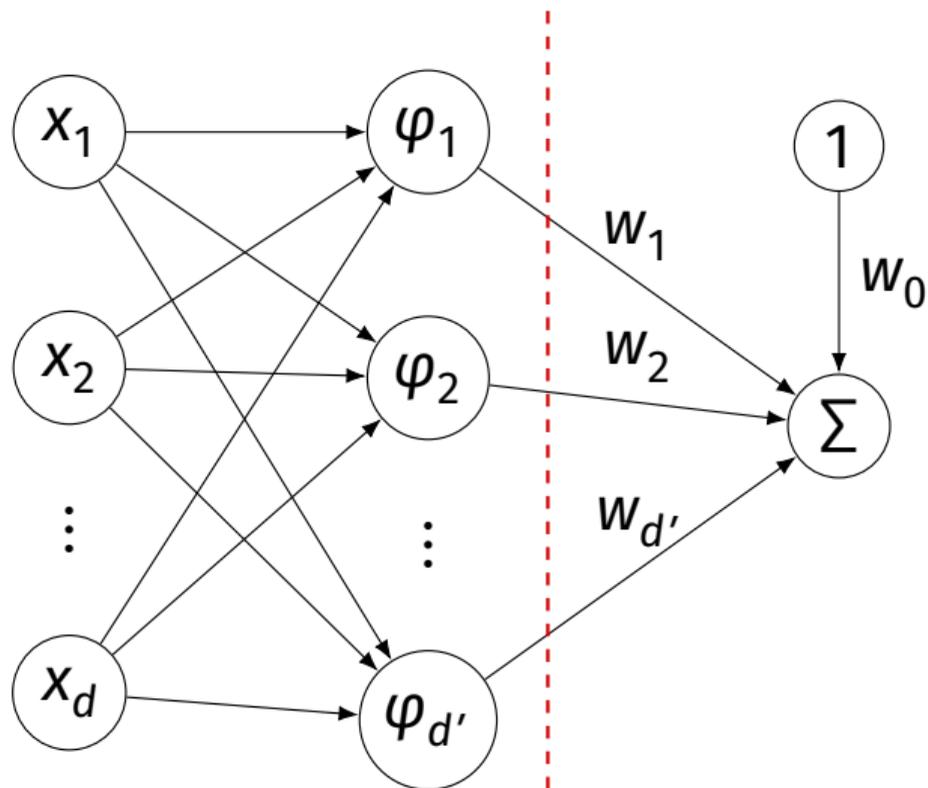
First Idea

- ▶ 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, $\varphi_1, \dots, \varphi_k$.
2. Pick the parameters of the RBFs *somehow*.
3. Create new data set by mapping
$$\vec{x} \mapsto (\varphi_1(\vec{x}), \dots, \varphi_k(\vec{x}))^T$$
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:
$$\vec{x} \mapsto (\varphi_1(\vec{x}), \dots, \varphi_k(\vec{x}))^T$$
2. Evaluate the trained linear predictor H_f in feature space

Exercise

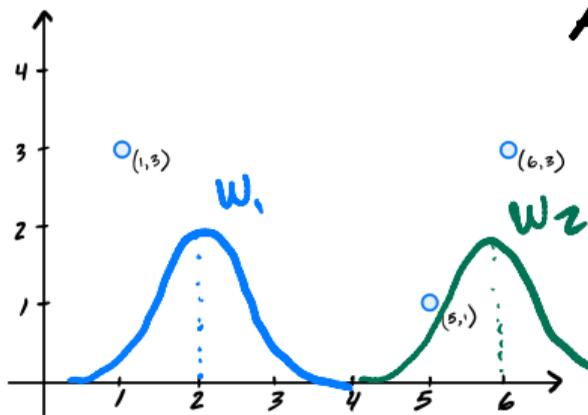
A Gaussian RBF network $H(x) = w_1\varphi_1(x) + w_2\varphi_2(x)$ (no bias) is trained on the data below using two Gaussian basis functions with centers $\mu_1 = 2$, $\mu_2 = 6$ and width $\sigma = 1$.

What is w_1 , approximately?

Want:

$$H(1) \approx 3$$

$$w_1 \approx 3e$$



At $x=1$, $H(x) \approx w_1\varphi_1(x)$

Want: $w_1\varphi_1(1) \approx 3$

$$\rightarrow w_1 e^{-(x-\mu_1)^2/\sigma^2} \approx 3$$

$$\rightarrow w_1 e^{-(1-2)^2/1^2} \approx 3$$

$$\rightarrow w_1 e^{-1} \approx 3$$

DSC 140B

Representation Learning

Lecture 10 | Part 4

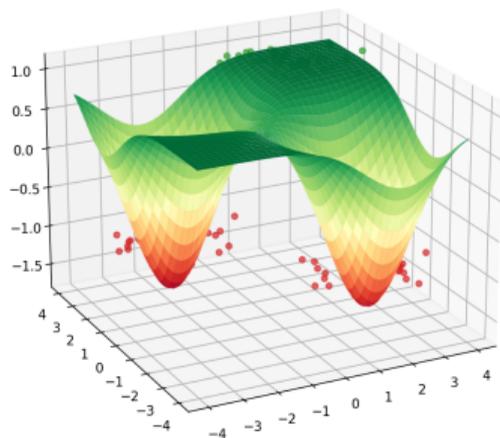
Choosing RBF Locations

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

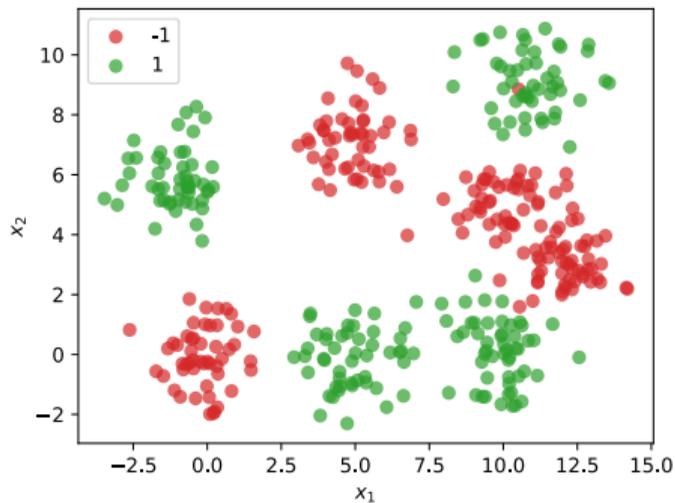
- ▶ Our prediction function H is a surface that is made up of Gaussian “bumps”.



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

Choosing Centers

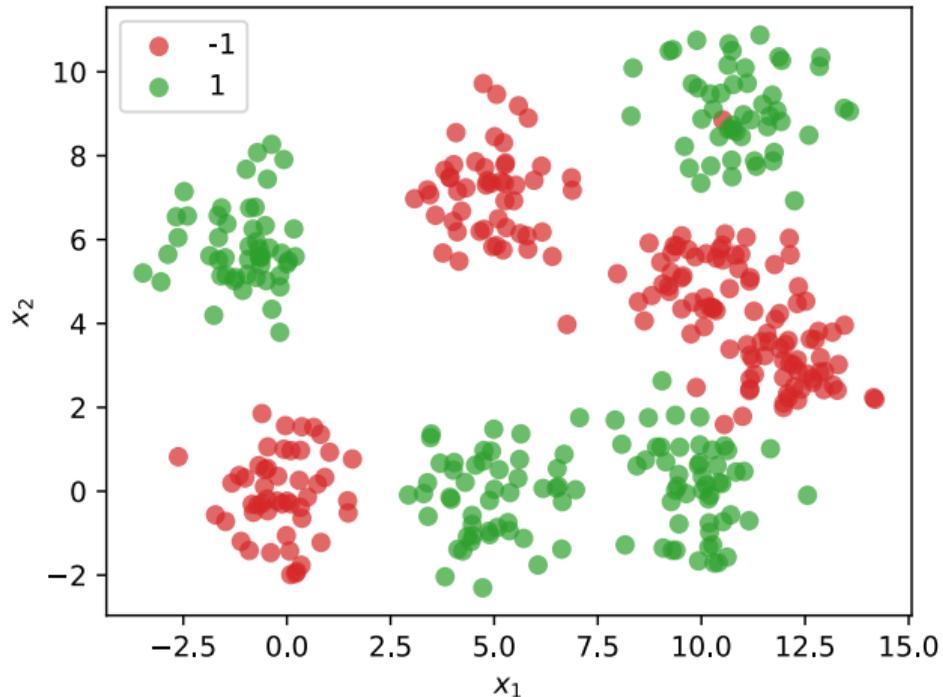
- ▶ Place the centers where the value of the prediction function should be controlled.
- ▶ Intuitively: place centers where the data is.



Approaches

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

Approach #1: Every Data Point as a Center



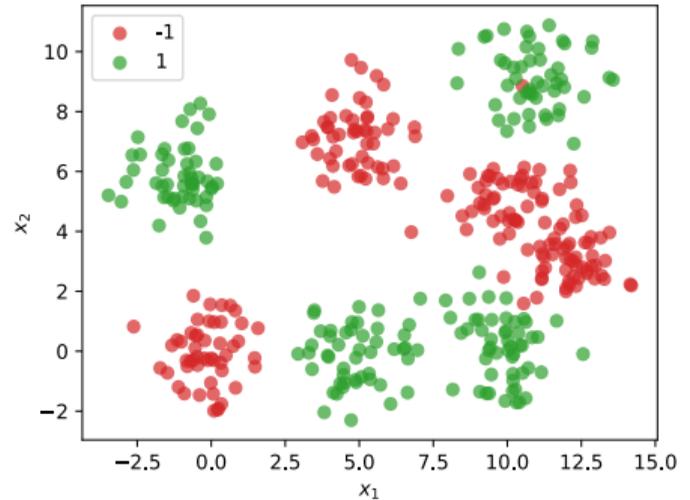
Dimensionality

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

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

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.

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 10 | Part 5

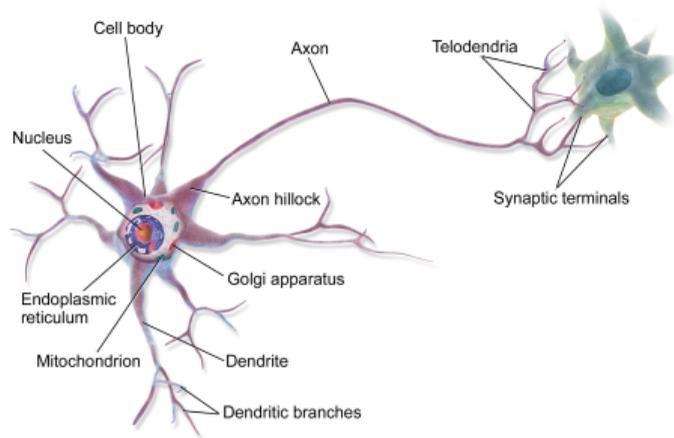
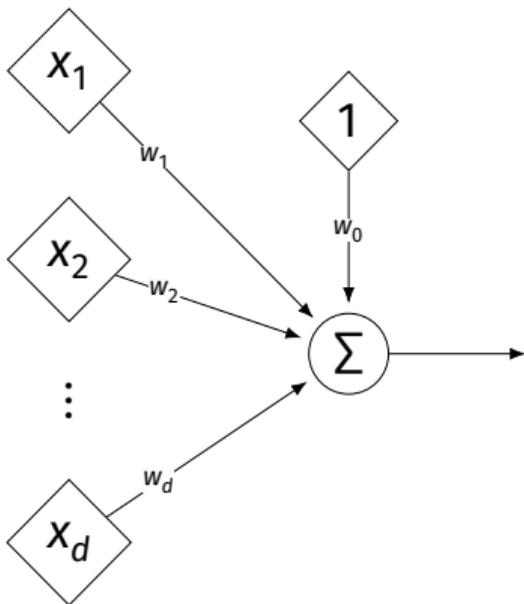
Neural Networks

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

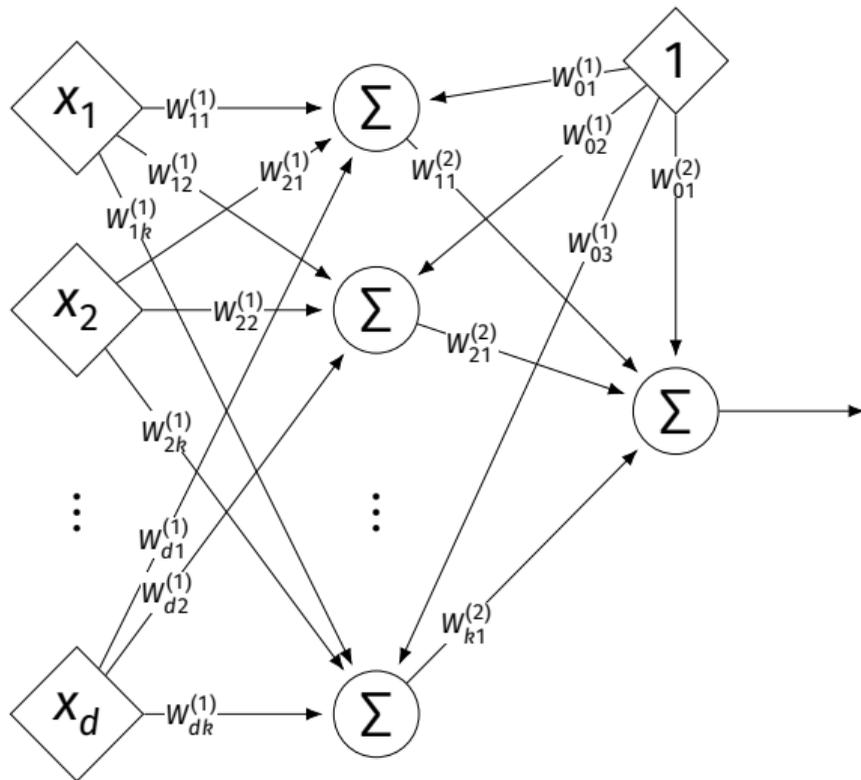
$$H(\vec{X}) = w_0 + w_1 X_1 + \dots + w_d X_d$$



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 Network¹ (One Hidden Layer)



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

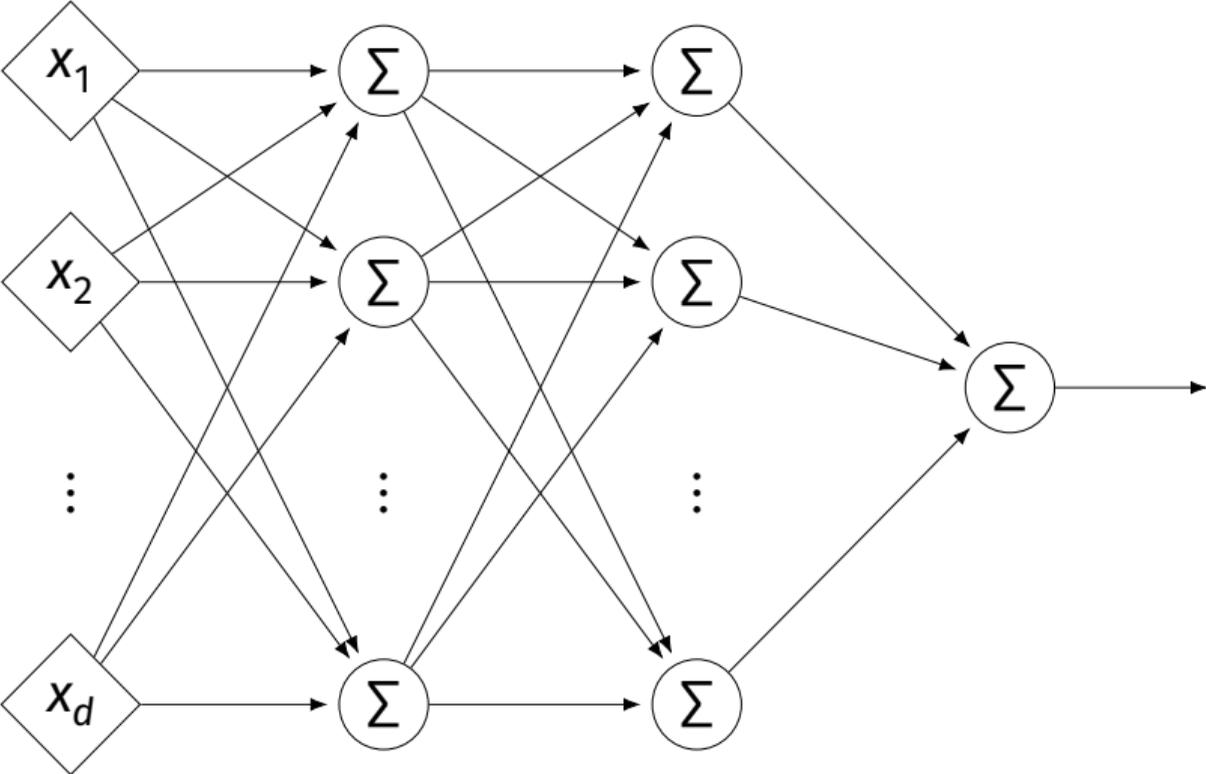
Architecture

- ▶ 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)

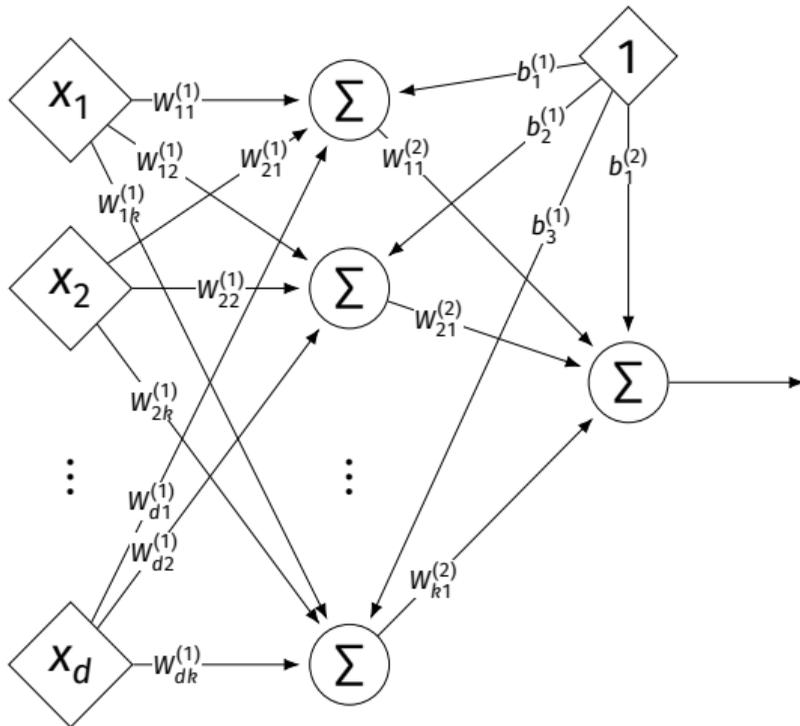


Network Weights

- ▶ 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!

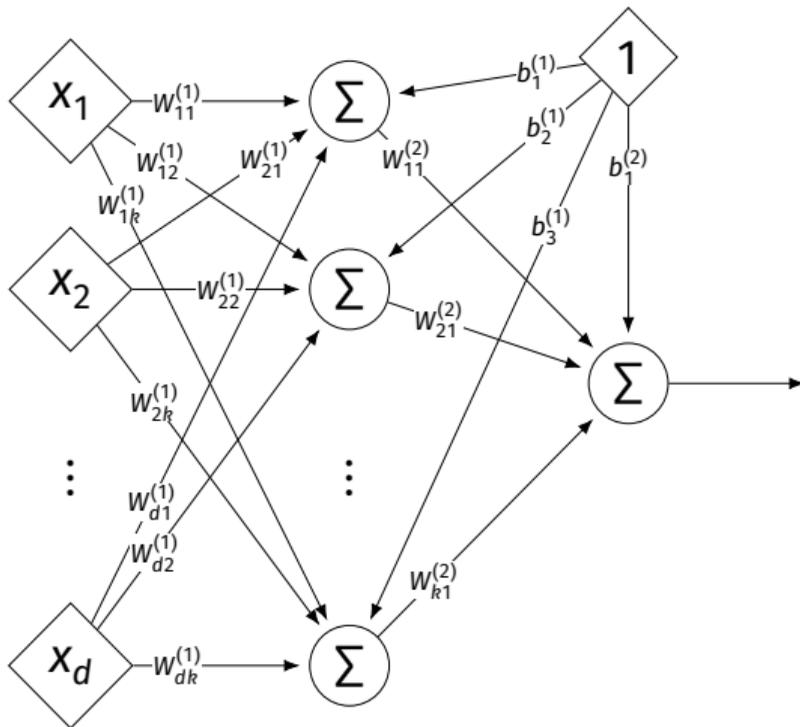
Notation

- ▶ 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.



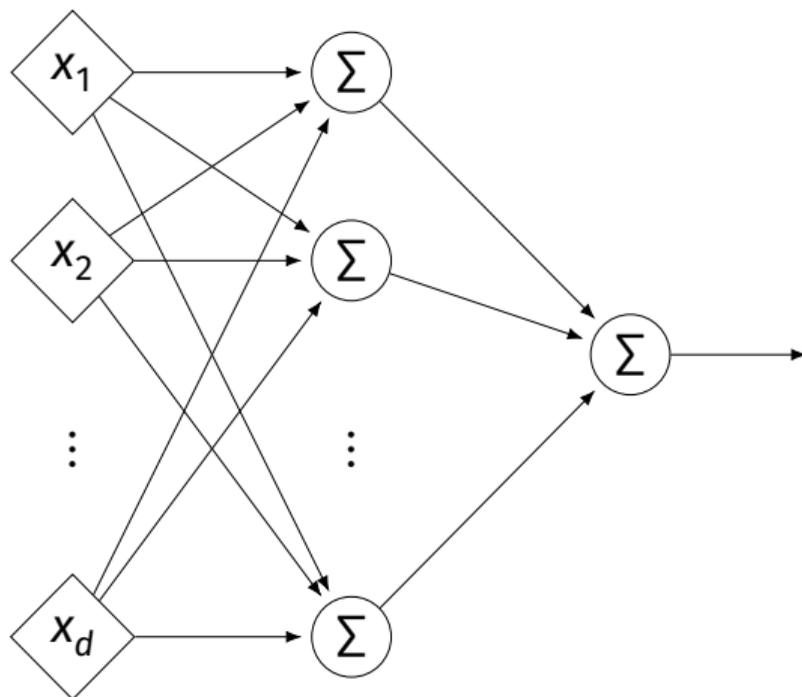
Notation

- ▶ 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: $\vec{b}^{(i)}$

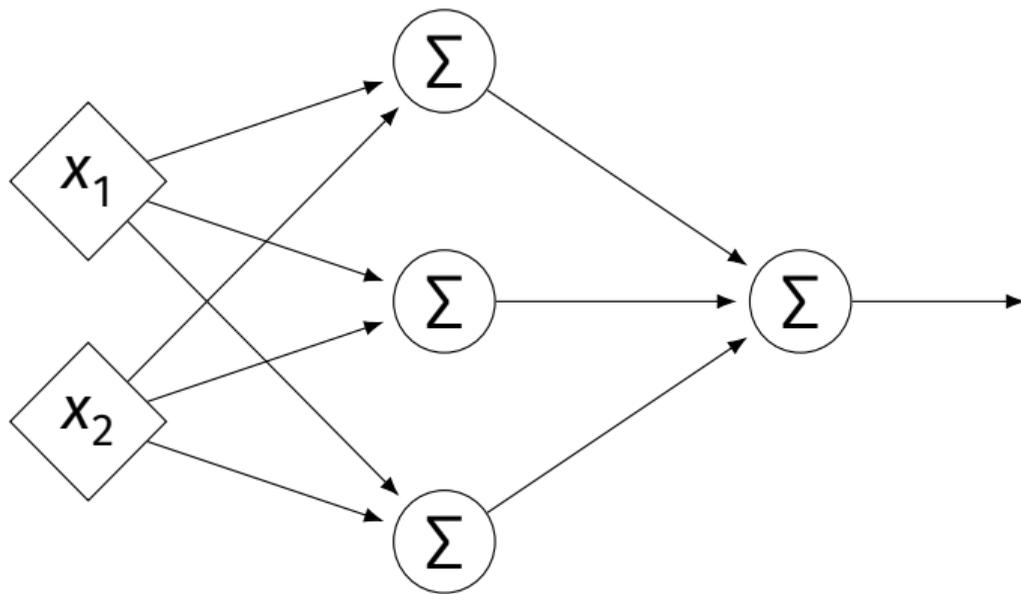


Notation

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



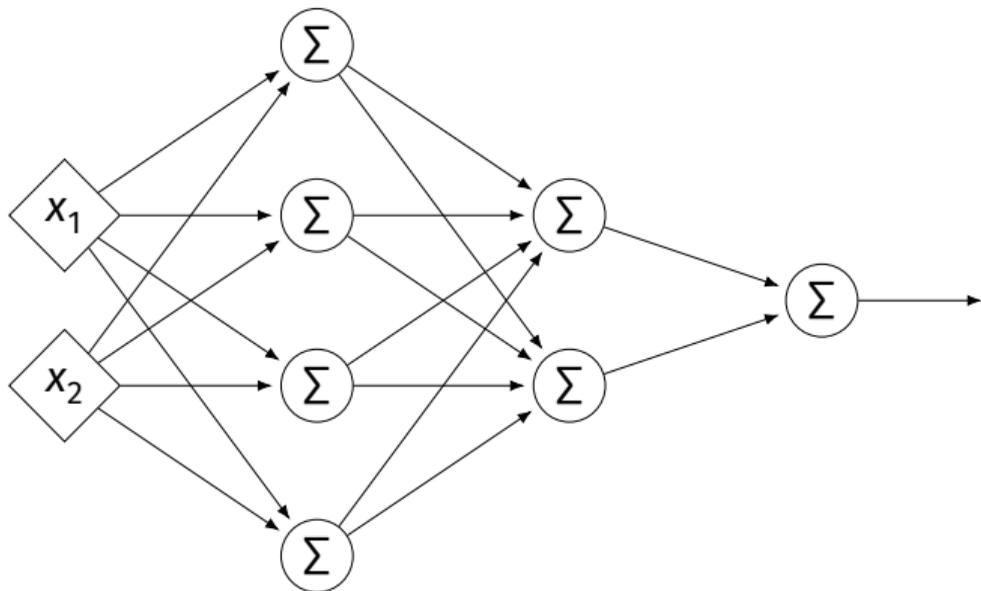
Example



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

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

Example



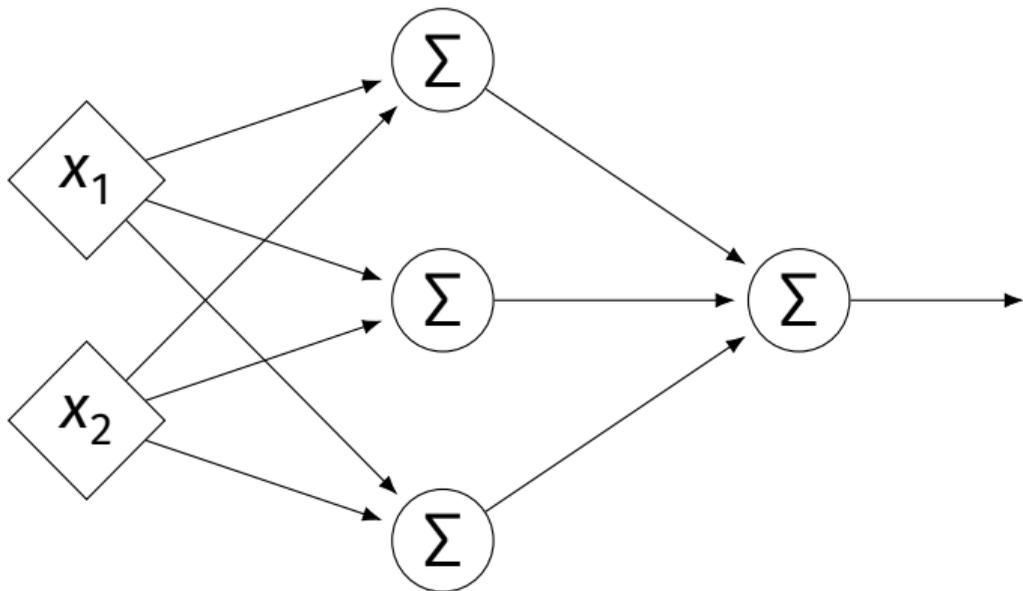
$$W^{(1)} = \begin{pmatrix} 2 & -1 & -3 & 0 \\ 4 & 5 & -7 & 2 \end{pmatrix} \quad W^{(2)} = \begin{pmatrix} 1 & 2 \\ -4 & 3 \\ -6 & -2 \\ 3 & 4 \end{pmatrix} \quad W^{(3)} = (-1 \quad 5)$$

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

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



▶ $\vec{x} = (3, -1)^T$

▶ $z_1^{(1)} =$

▶ $z_2^{(1)} =$

▶ $z_3^{(1)} =$

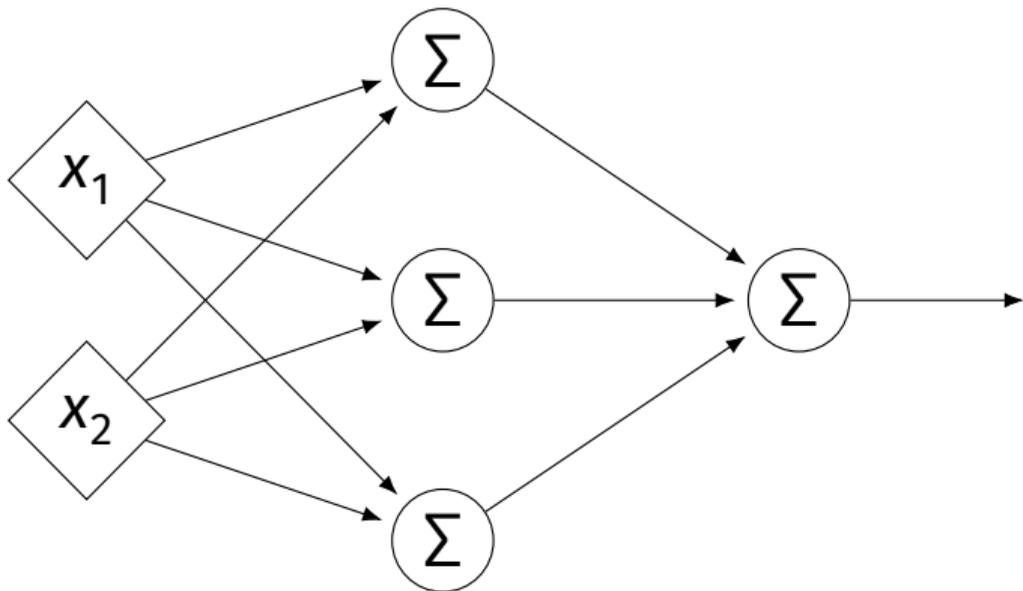
▶ $z_1^{(2)} =$

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

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

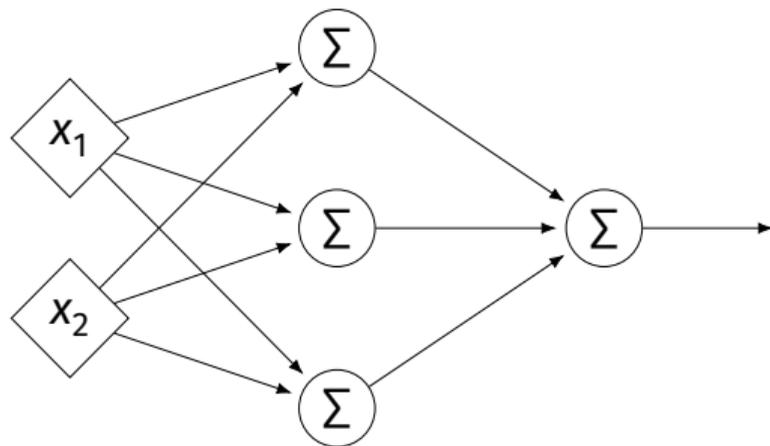
- ▶ We can think of each layer as a function mapping a vector to a vector.

- ▶ $H^{(1)}(\vec{z}) = [W^{(1)}]^T \vec{z} + \vec{b}^{(1)}$

- ▶ $H^{(1)} : \mathbb{R}^2 \rightarrow \mathbb{R}^3$

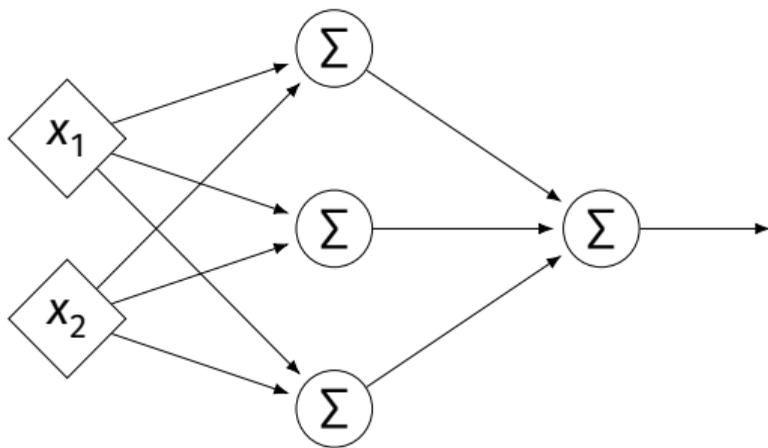
- ▶ $H^{(2)}(\vec{z}) = [W^{(2)}]^T \vec{z} + \vec{b}^{(2)}$

- ▶ $H^{(2)} : \mathbb{R}^3 \rightarrow \mathbb{R}^1$



NNs as Function Composition

- ▶ The full NN is a composition of layer functions.



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

NNs as Function Composition

- ▶ In general, if there k hidden layers:

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

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.

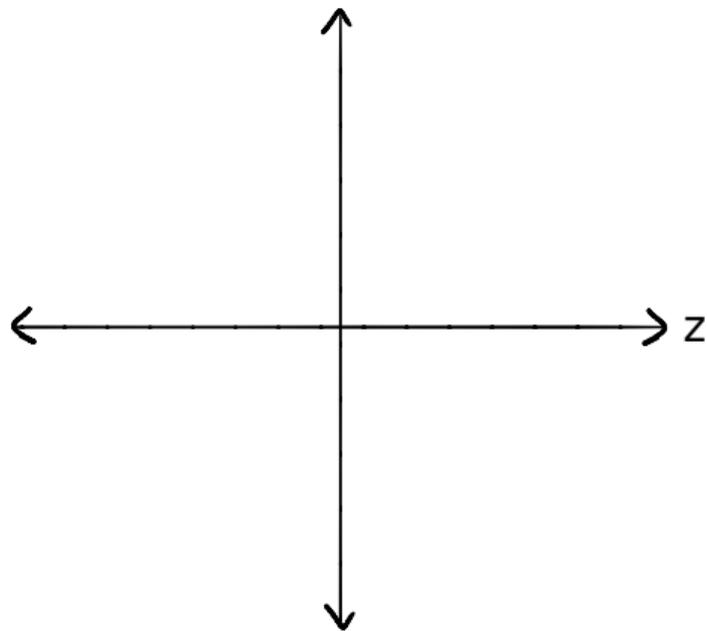
Idea

- ▶ 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.

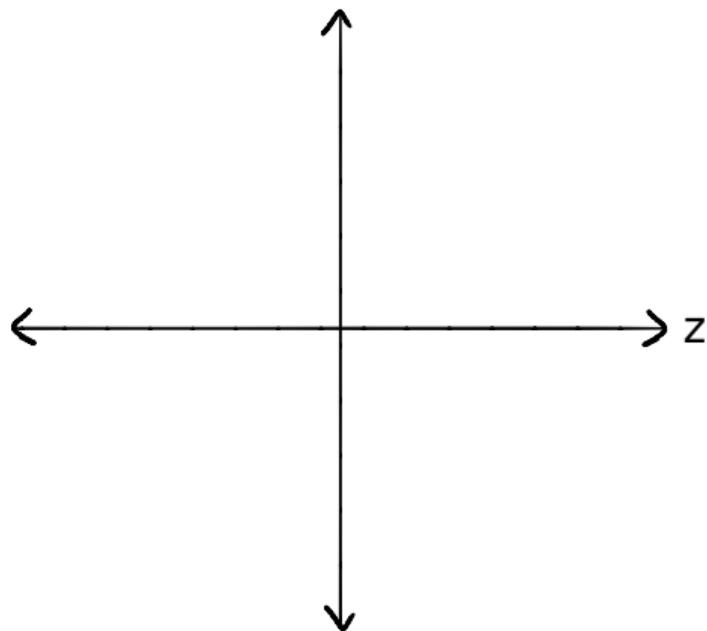
$$\sigma(z) = z$$



Sigmoid Activation

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

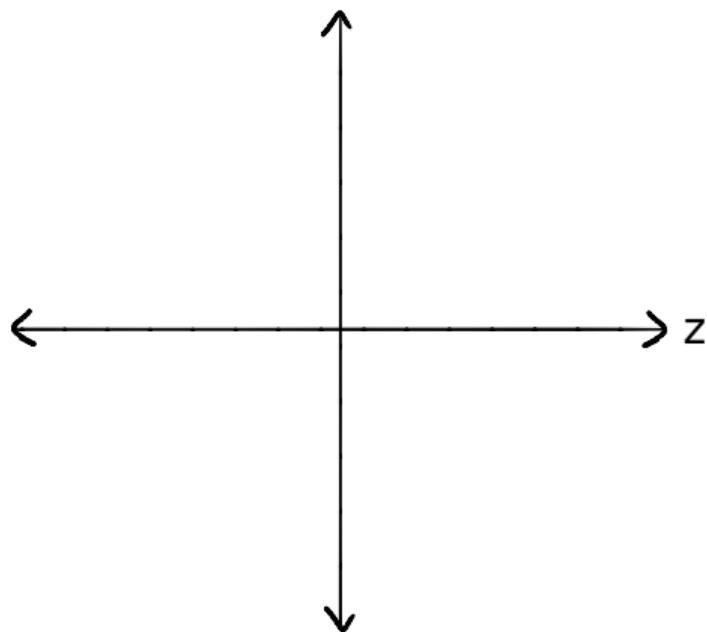
$$\sigma(z) = \frac{1}{1 + e^{-z}}$$



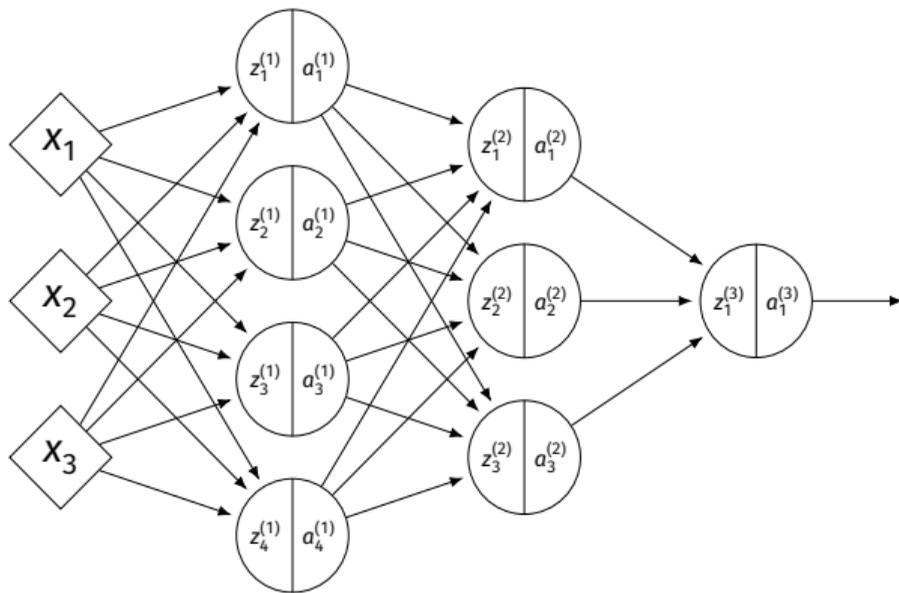
ReLU Activation

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

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

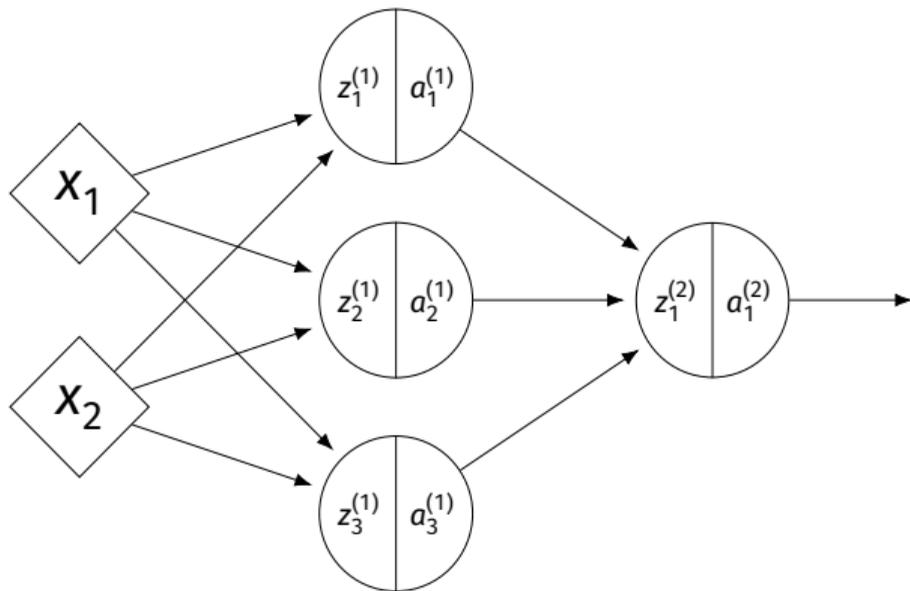


Notation



- ▶ $z_j^{(i)}$ 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.