

Table of Contents

- Lecture 02 - linear - algebra - I
- Lecture 03 - linear - algebra - II
- Lecture 04 - linear - algebra - III
- Lecture 05 - linear - algebra - IV
- Lecture 06 - pca - I
- Lecture 07 - pca - II
- Lecture 08 - pca - III
- Lecture 09 - laplacian - eigenmaps
- Lecture 10 - linear - models
- Lecture 11 - feature - maps
- Lecture 12 - rbf - networks
- Lecture 13 - neural - networks
- Lecture 14 - training - nns
- Lecture 15 - sgd
- Lecture 16 - cnn
- Lecture 17 - autoencoders

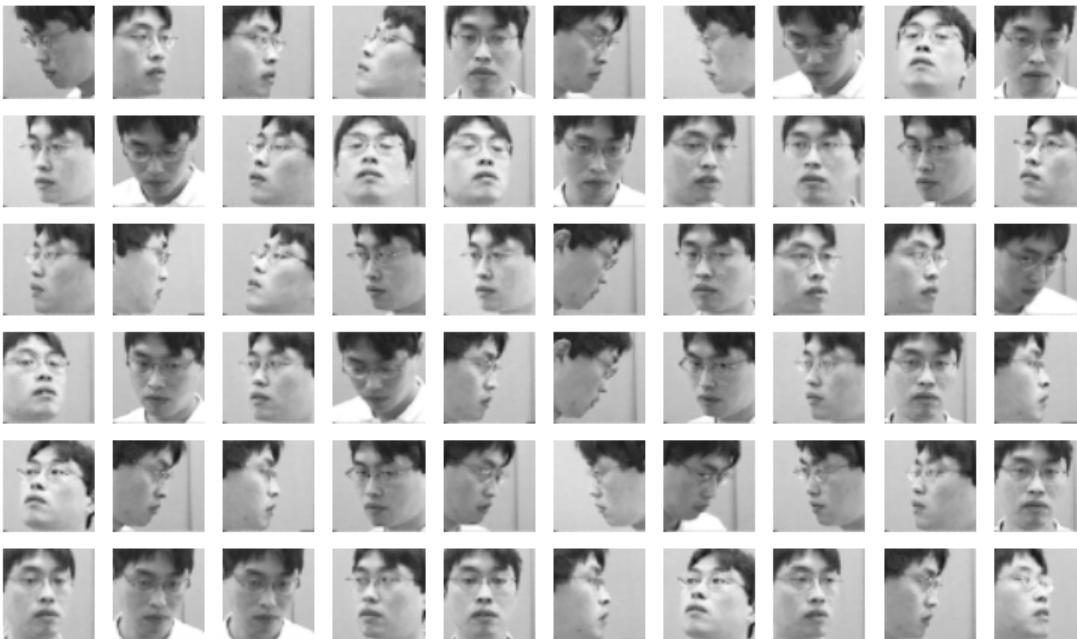
DSC 140B

Representation Learning

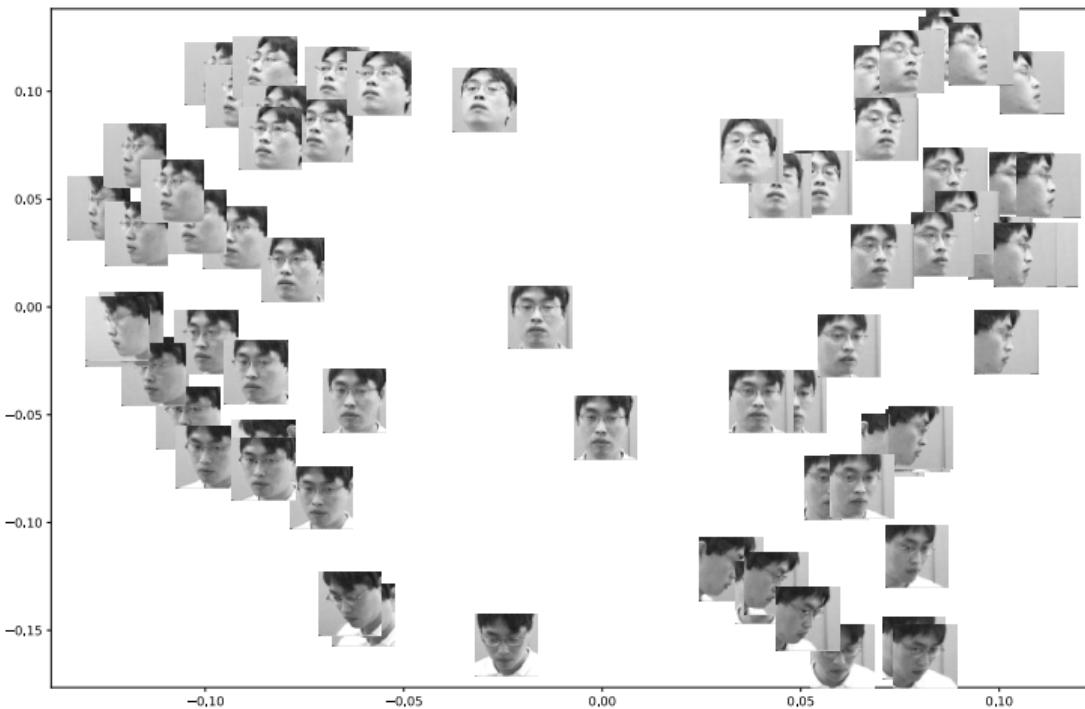
Lecture 02 | Part 1

Why Linear Algebra?

Last Time



Last Time



Dimensionality Reduction

- ▶ This is an example of **dimensionality reduction**:
 - ▶ Input: vectors in $\mathbb{R}^{10,000}$.
 - ▶ Output: vectors in \mathbb{R}^2 .
- ▶ The method which produced this result is called **Laplacian Eigenmaps**.
- ▶ How does it work?

A Preview of Laplacian Eigenmaps

To reduce dimensionality from d to d' :

1. Create an undirected **similarity graph** G
 - ▶ Each vector in \mathbb{R}^d becomes a node in the graph.
 - ▶ Make edge (u, v) if u and v are “close”
2. Form the **graph Laplacian matrix**, L :
 - ▶ Let A be the adjacency matrix, D be the degree matrix.
 - ▶ Define the graph Laplacian matrix, $L = D - A$.
3. Compute d' **eigenvectors** of L .
 - ▶ Each eigenvector gives one new feature.

Why eigenvectors?

- ▶ We will cover Laplacian Eigenmaps in much greater detail.
- ▶ For now: why do eigenvectors appear here?
 - ▶ What are eigenvectors?
 - ▶ How are they useful?
 - ▶ Why is linear algebra important in ML?

DSC 140B

Representation Learning

Lecture 02 | Part 2

Coordinate Vectors

Coordinate Vectors

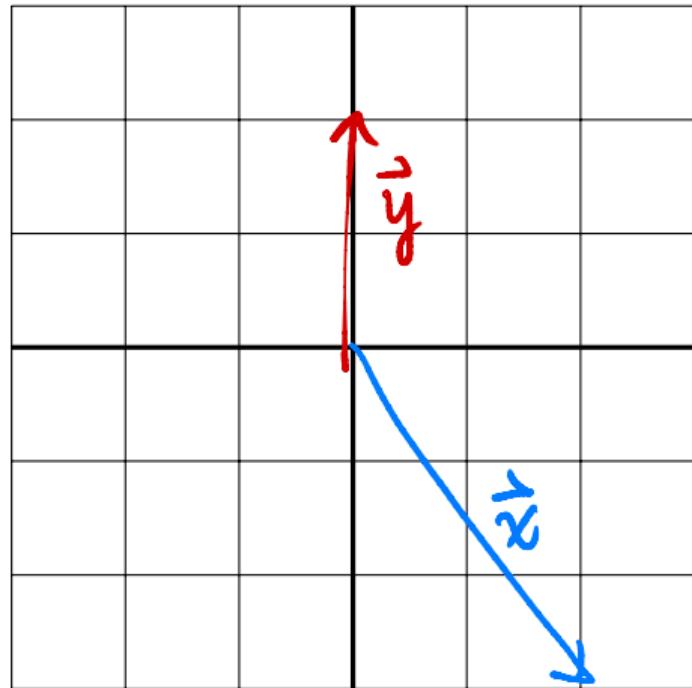
- We can write a vector $\vec{x} \in \mathbb{R}^d$ as a **coordinate vector**:

$$\vec{x} = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_d \end{pmatrix}$$

Example

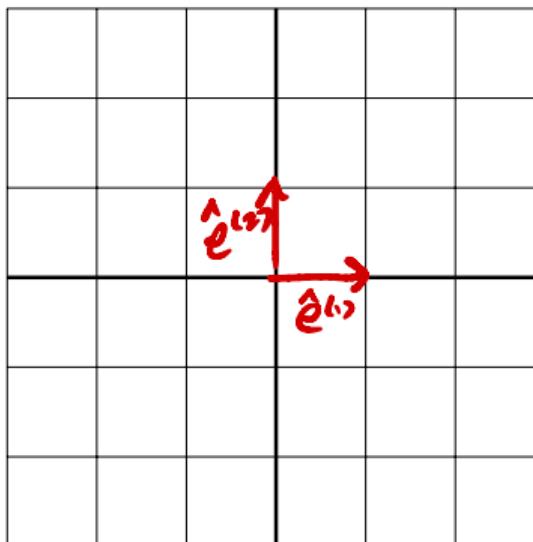
$$\vec{x} = \begin{pmatrix} 2 \\ -3 \end{pmatrix}$$

$$\vec{y} = \begin{pmatrix} 0 \\ 2 \end{pmatrix}$$



Standard Basis

- ▶ Writing a vector in coordinate form requires choosing a **basis**.
- ▶ The “default” is the **standard basis**: $\hat{e}^{(1)}, \dots, \hat{e}^{(d)}$.

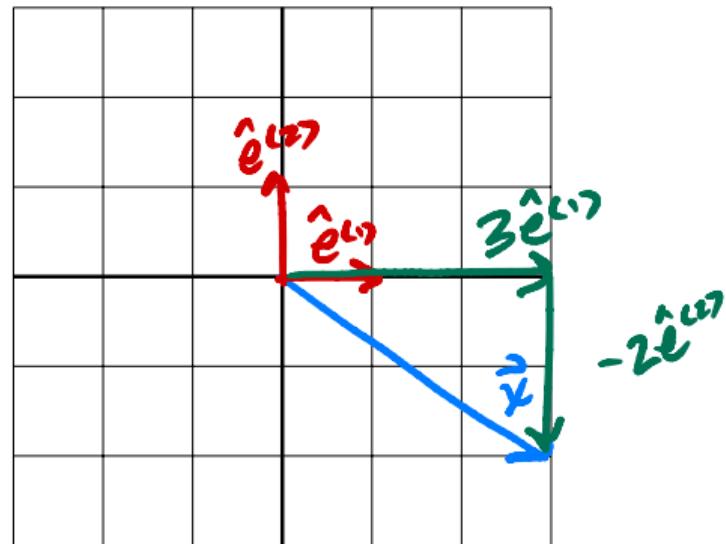


Standard Basis

- When we write $\vec{x} = (x_1, \dots, x_d)^T$, we mean that
 $\vec{x} = x_1 \hat{e}^{(1)} + x_2 \hat{e}^{(2)} + \dots + x_d \hat{e}^{(d)}$.

Example: $\vec{x} = (3, -2)^T$

$$\vec{x} = 3 \hat{e}^{(1)} - 2 \hat{e}^{(2)}$$



Standard Basis Coordinates

- In coordinate form:

$$\hat{e}^{(1)} = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$

$$\hat{e}^{(2)} = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

$$\hat{e}^{(i)} = \begin{pmatrix} 0 \\ 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \\ 0 \end{pmatrix}$$

where the 1 appears in the i th place.

Exercise

Let $\vec{x} = (3, 7, 2, -5)^T$. What is $\vec{x} \cdot \hat{e}^{(4)}$?

-5

Recall: the Dot Product

- ▶ The **dot product** of \vec{u} and \vec{v} is defined as:

$$\vec{u} \cdot \vec{v} = \|\vec{u}\| \|\vec{v}\| \cos \theta$$

where θ is the angle between \vec{u} and \vec{v} .

- ▶ $\vec{u} \cdot \vec{v} = 0$ if and only if \vec{u} and \vec{v} are orthogonal

Dot Product (Coordinate Form)

- ▶ In terms of coordinate vectors:

$$\vec{u} \cdot \vec{v} = \vec{u}^T \vec{v}$$

$$\begin{aligned} &= (u_1 \ u_2 \ \dots \ u_d) \begin{pmatrix} v_1 \\ v_2 \\ \dots \\ v_d \end{pmatrix} \\ &= u_1 v_1 + u_2 v_2 + \dots + u_d v_d \end{aligned}$$

- ▶ This definition assumes the standard basis.

$$\vec{x} = 3\hat{e}^{(1)} + 7\hat{e}^{(2)} + 2\hat{e}^{(3)} - 5\hat{e}^{(4)}$$

Example

$$\begin{pmatrix} 3 \\ 7 \\ 2 \\ -5 \end{pmatrix} \cdot \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix} = -5$$

$$\begin{aligned}\vec{x} \cdot \hat{e}^{(4)} &= (3\hat{e}^{(1)} + 7\hat{e}^{(2)} + 2\hat{e}^{(3)} - 5\hat{e}^{(4)}) \cdot \hat{e}^{(4)} \\ &= 3\cancel{\hat{e}^{(1)} \cdot \hat{e}^{(4)}}^0 + 7\cancel{\hat{e}^{(2)} \cdot \hat{e}^{(4)}}^0 + 2\cancel{\hat{e}^{(3)} \cdot \hat{e}^{(4)}}^0 - 5\hat{e}^{(4)}\end{aligned}$$

What does ChatGPT say?



I will ask you a simple vector math question. Respond with the answer, and nothing else. Do not provide an explanation. Your answer should be a number.

Let $\vec{x} = (3, 7, 2, -5)^T$. What is $\vec{x} \cdot \hat{e}_4$?



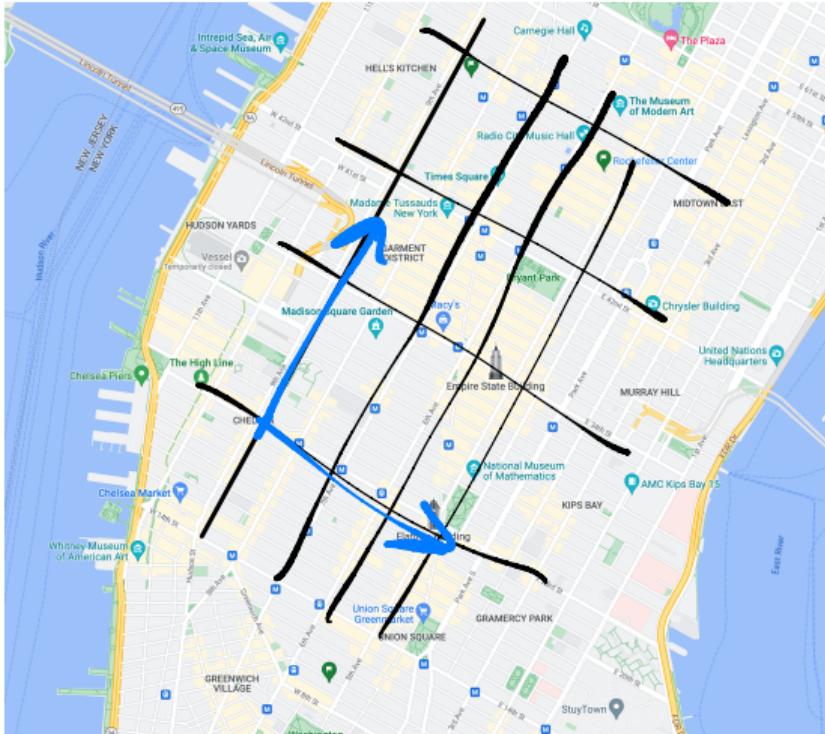
-5.



Other Bases

- ▶ The standard basis is not the **only** basis.
- ▶ Sometimes more convenient to use another.

Example

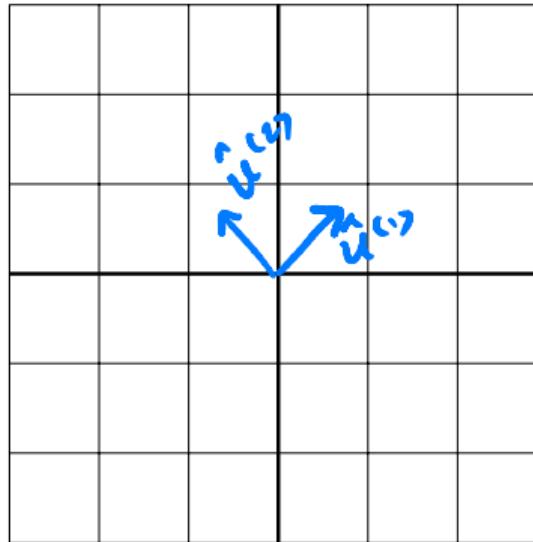


Orthonormal Bases

- ▶ **Orthonormal bases** are particularly nice.
- ▶ A set of vectors $\hat{u}^{(1)}, \dots, \hat{u}^{(d)}$ forms an **orthonormal basis** \mathcal{U} for R^d if:
 - ▶ They are mutually orthogonal: $\hat{u}^{(i)} \cdot \hat{u}^{(j)} = 0$.
 - ▶ They are all unit vectors: $\|\hat{u}^{(i)}\| = 1$.

Example

$$\hat{u}^{(1)} = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 \\ 1 \end{pmatrix} \quad \hat{u}^{(2)} = \frac{1}{\sqrt{2}} \begin{pmatrix} -1 \\ 1 \end{pmatrix}$$



Coordinate Vectors

- ▶ A vector's coordinates depend on the basis used.
- ▶ If we are using the basis $\mathcal{U} = \{\hat{u}^{(1)}, \hat{u}^{(2)}\}$, then $[\vec{x}]_{\mathcal{U}} = (x_1, x_2)^T$ means $\vec{x} = x_1 \hat{u}^{(1)} + x_2 \hat{u}^{(2)}$.
- ▶ We will write $[\vec{x}]_{\mathcal{U}} = (x_1, \dots, x_d)^T$ to denote that the coordinates are with respect to the basis \mathcal{U} .

Exercise

Let $\hat{u}^{(1)} = \frac{1}{\sqrt{2}}(1, 1)^T$ and $\hat{u}^{(2)} = \frac{1}{\sqrt{2}}(-1, 1)^T$. Suppose $[\vec{x}]_{\mathcal{U}} = (3, -4)^T$. What is $\vec{x} \cdot \hat{u}^{(1)}$?

3

$$\vec{x} = 3\hat{u}^{(1)} - 4\hat{u}^{(2)}$$

$$\begin{aligned}\vec{x} \cdot \hat{u}^{(1)} &= (3\hat{u}^{(1)} - 4\hat{u}^{(2)}) \cdot \hat{u}^{(1)} \\ &= 3\hat{u}^{(1)} \cdot \hat{u}^{(1)} - 4\hat{u}^{(2)} \cdot \hat{u}^{(1)}\end{aligned}$$

What did ChatGPT say?



Let $\mathbf{u}_1 = \frac{1}{\sqrt{2}}(1, 1)^T$ and $\mathbf{u}_2 = \frac{1}{\sqrt{2}}(-1, 1)^T$. Suppose $\vec{x} \cdot \mathbf{U} = (3, -4)^T$. What is $\vec{x} \cdot \mathbf{u}_1$?



-0.5. (Note: This answer assumes that $\vec{x} \cdot \mathbf{U}$ represents the coordinate vector of \vec{x} with respect to the basis \mathbf{U} , and that \mathbf{u}_1 and \mathbf{u}_2 are orthonormal basis vectors.) The dot product between \vec{x} and \mathbf{u}_1 represents the projection of \vec{x} onto the direction of \mathbf{u}_1 . In this case, \mathbf{u}_1 is a unit vector, so the dot product gives the signed length of the projection of \vec{x} onto the direction of \mathbf{u}_1 . The result of -0.5 indicates that the projection of \vec{x} onto the direction of \mathbf{u}_1 is in the opposite direction of \mathbf{u}_1 , and the magnitude of the projection is 0.5 times the length of \mathbf{u}_1 . Note that the negative sign indicates that the projection is in the opposite direction of \mathbf{u}_1 . The dot product can be used to determine the alignment or misalignment of vectors, and is a useful tool in various applications, such as in physics, engineering, and computer graphics. It is important to understand how to compute and interpret dot products in the context of vector projections and geometric relationships between vectors. Therefore, the result of -0.5 indicates that \vec{x} is projected onto the opposite direction of \mathbf{u}_1 with a magnitude of 0.5 times the length of \mathbf{u}_1 . Keep practicing vector math problems to further strengthen your understanding of dot products and their applications!

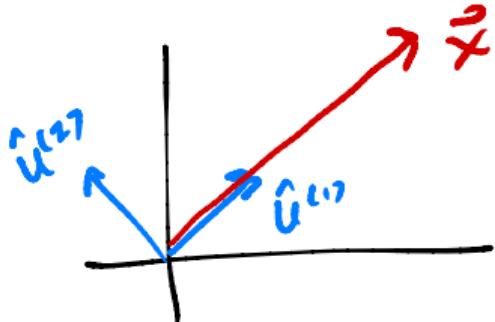
$$2\sqrt{2} \cdot \frac{1}{\sqrt{2}}(1,1) = \frac{2\sqrt{2}}{\sqrt{2}}(1,1)$$

$$= 2(1,1)$$

$$= (2,2)$$

Exercise

Consider $\vec{x} = (2, 2)^T$ and let $\hat{u}^{(1)} = \frac{1}{\sqrt{2}}(1, 1)^T$ and $\hat{u}^{(2)} = \frac{1}{\sqrt{2}}(-1, 1)^T$. What is $[\vec{x}]_{\mathcal{U}}$?



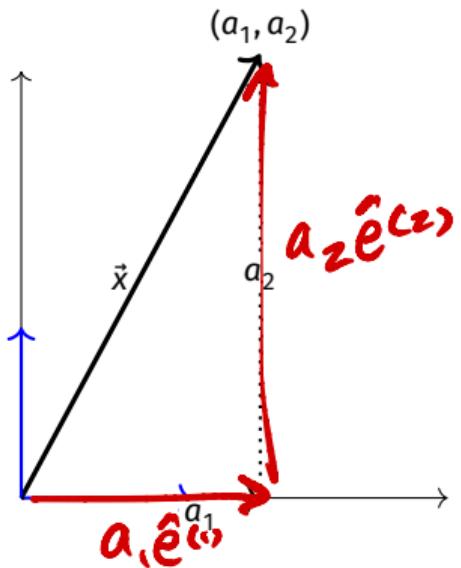
$$[\vec{x}]_{\mathcal{U}} = (2\sqrt{2}, 0)^T$$

$$\vec{x} = 2\sqrt{2}\hat{u}^{(1)}$$

Change of Basis

- ▶ How do we compute the coordinates of a vector in a new basis, \mathcal{U} ?
- ▶ Some trigonometry is involved.
- ▶ **Key Fact:** $\vec{a} \cdot \vec{b} = \|\vec{a}\| \|\vec{b}\| \cos \theta$

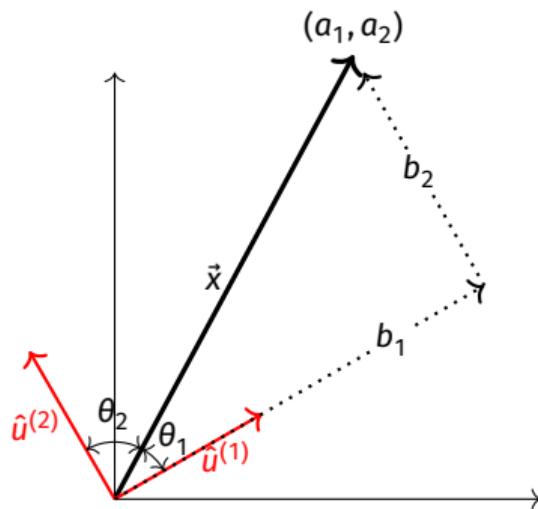
Change of Basis



$\wedge \backslash \text{hat} \rightarrow \backslash \text{vec}$

- ▶ Suppose we know $\vec{x} = (a_1, a_2)^T$ w.r.t. standard basis.
- ▶ Then $\vec{x} = a_1 \hat{e}^{(1)} + a_2 \hat{e}^{(2)}$

Change of Basis

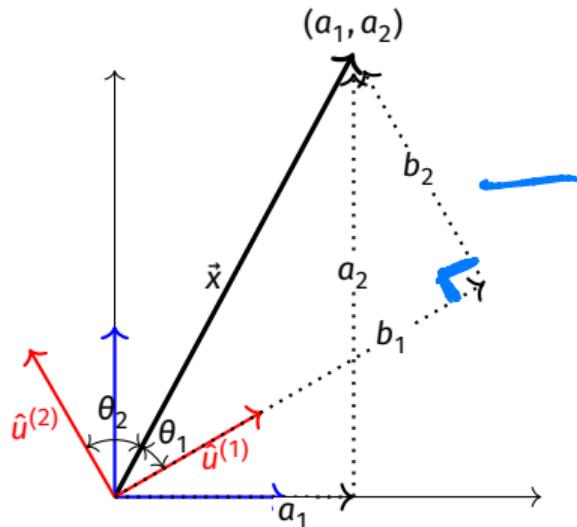
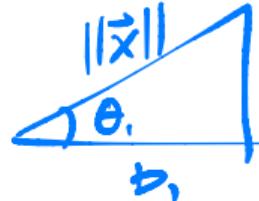


$$[\vec{x}]_u = (b_1, b_2)^T$$

- ▶ Want to write:
 $\vec{x} = b_1 \hat{u}^{(1)} + b_2 \hat{u}^{(2)}$
- ▶ Need to find b_1 and b_2 .

Change of Basis

$$b_1 = \|\vec{x}\| \cos\theta,$$



► **Exercise:** Solve for b_1 , writing the answer as a dot product.

► Hint: $\cos\theta = \text{adjacent/hypotenuse}$

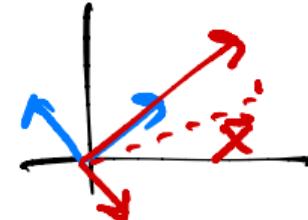
$$\vec{x} \cdot \hat{u}^{(1)} = \|\vec{x}\| \|\hat{u}^{(1)}\| \cos\theta,$$

Change of Basis

- ▶ Let $\mathcal{U} = \{\hat{u}^{(1)}, \dots, \hat{u}^{(d)}\}$ be an orthonormal basis.
- ▶ The coordinates of \vec{x} w.r.t. \mathcal{U} are:

$$[\vec{x}]_{\mathcal{U}} = \begin{pmatrix} \vec{x} \cdot \hat{u}^{(1)} \\ \vec{x} \cdot \hat{u}^{(2)} \\ \vdots \\ \vec{x} \cdot \hat{u}^{(d)} \end{pmatrix}$$

$$[\vec{x}]_U = \begin{pmatrix} \vec{x} \cdot \hat{u}^{(1)} \\ \vec{x} \cdot \hat{u}^{(2)} \end{pmatrix} = \begin{pmatrix} 3/\sqrt{2} \\ -1/\sqrt{2} \end{pmatrix}$$



Exercise

Suppose $\vec{x} = (2, 1)^T$ and let $\hat{u}^{(1)} = \frac{1}{\sqrt{2}}(1, 1)^T$ and $\hat{u}^{(2)} = \frac{1}{\sqrt{2}}(-1, 1)^T$. What is $[\vec{x}]_U$?

$$\begin{aligned}\vec{x} \cdot \hat{u}^{(1)} &= \begin{pmatrix} 2 \\ 1 \end{pmatrix} \cdot \frac{1}{\sqrt{2}} \begin{pmatrix} 1 \\ 1 \end{pmatrix} \\ &= \frac{1}{\sqrt{2}} \begin{pmatrix} 2 \\ 1 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 1 \end{pmatrix} = \frac{1}{\sqrt{2}} \times 3 = \frac{3}{\sqrt{2}}\end{aligned}$$

$$\begin{aligned}\vec{x} \cdot \hat{u}^{(2)} &= \begin{pmatrix} 2 \\ 1 \end{pmatrix} \cdot \frac{1}{\sqrt{2}} \begin{pmatrix} -1 \\ 1 \end{pmatrix} \\ &= \frac{1}{\sqrt{2}} \times -1 = \frac{-1}{\sqrt{2}}\end{aligned}$$

$$\hat{u}^{(1)} = 3\hat{e}^{(1)} - 2\hat{e}^{(2)} = \begin{pmatrix} 3 \\ -2 \end{pmatrix} \quad \hat{u}^{(2)} = \begin{pmatrix} -1 \\ 5 \end{pmatrix}$$

Exercise

Let $\vec{x} = (-1, 4)^T$ and suppose:

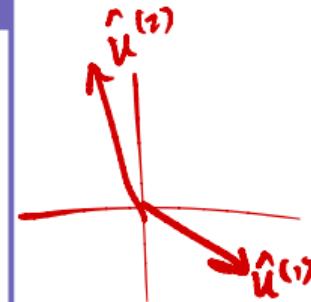
$$\hat{u}^{(1)} \cdot \hat{e}^{(1)} = 3$$

$$\hat{u}^{(1)} \cdot \hat{e}^{(2)} = -2$$

$$\hat{u}^{(2)} \cdot \hat{e}^{(1)} = -1$$

$$\hat{u}^{(2)} \cdot \hat{e}^{(2)} = 5$$

What is $[\vec{x}]_{\mathcal{U}}$?



$\hat{u}^{(1)}$ & $\hat{u}^{(2)}$ are not orthonormal

$$\vec{x} = \alpha \hat{u}^{(1)} + \beta \hat{u}^{(2)} \quad \begin{pmatrix} -1 \\ 4 \end{pmatrix} = \alpha \begin{pmatrix} 3 \\ -2 \end{pmatrix} + \beta \begin{pmatrix} -1 \\ 5 \end{pmatrix} \quad \begin{aligned} -1 &= 3\alpha - \beta \\ 4 &= -2\alpha + 5\beta \end{aligned}$$

DSC 140B

Representation Learning

Lecture 02 | Part 3

Functions of a Vector

Functions of a Vector

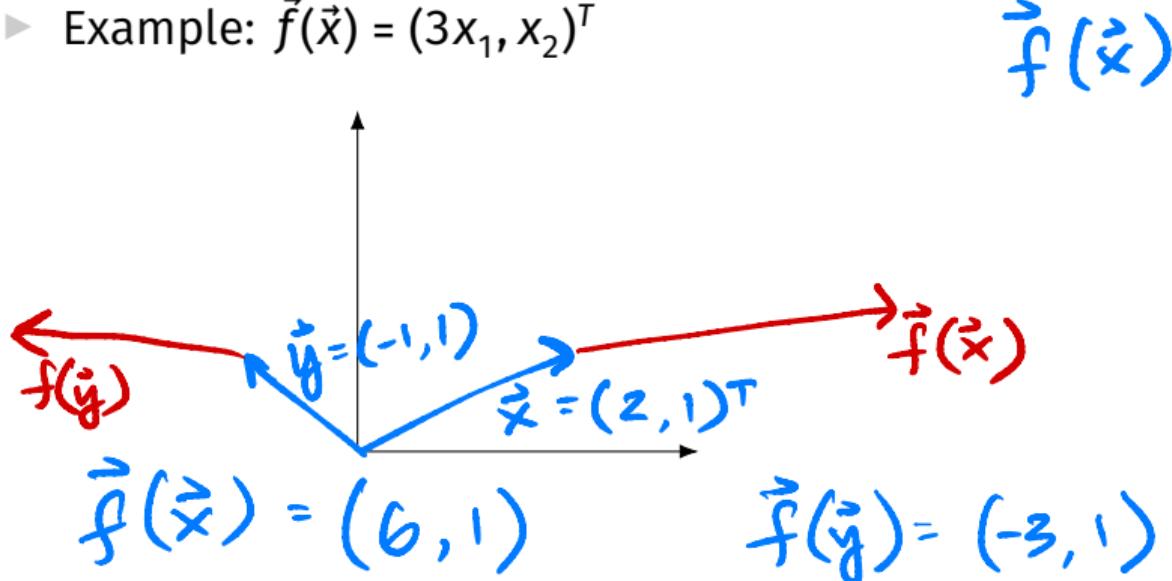
- ▶ In ML, we often work with functions of a vector:
 $\vec{f} : \mathbb{R}^d \rightarrow \mathbb{R}^{d'}$.
- ▶ Example: a prediction function, $H(\vec{x})$.
- ▶ Functions of a vector can return:
 - ▶ a number: $f : \mathbb{R}^d \rightarrow \mathbb{R}^1$
 - ▶ a vector $\vec{f} : \mathbb{R}^d \rightarrow \mathbb{R}^{d'}$
 - ▶ something else?

Transformations

- ▶ A **transformation** \vec{f} is a function that takes in a vector, and returns a vector *of the same dimensionality*.
- ▶ That is, $\vec{f} : \mathbb{R}^d \rightarrow \mathbb{R}^d$.

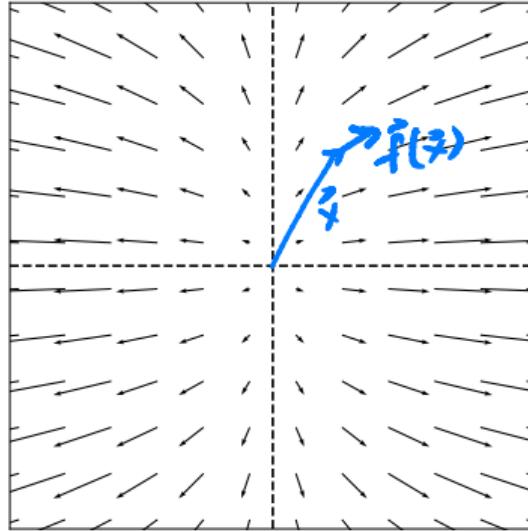
Visualizing Transformations

- ▶ A transformation is a **vector field**.
 - ▶ Assigns a vector to each point in space.
 - ▶ Example: $\vec{f}(\vec{x}) = (3x_1, x_2)^T$



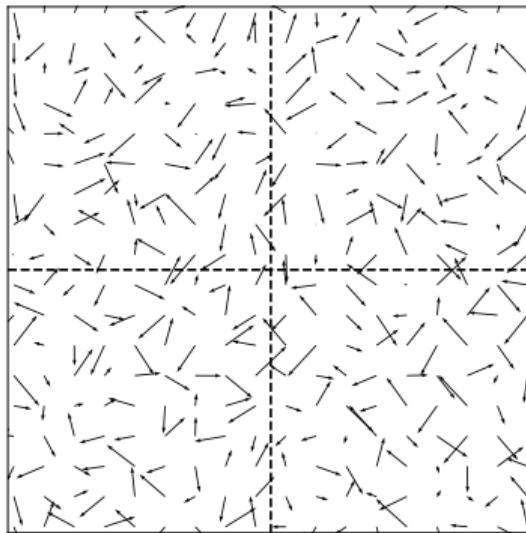
Example

► $\vec{f}(\vec{x}) = (3x_1, x_2)^T$



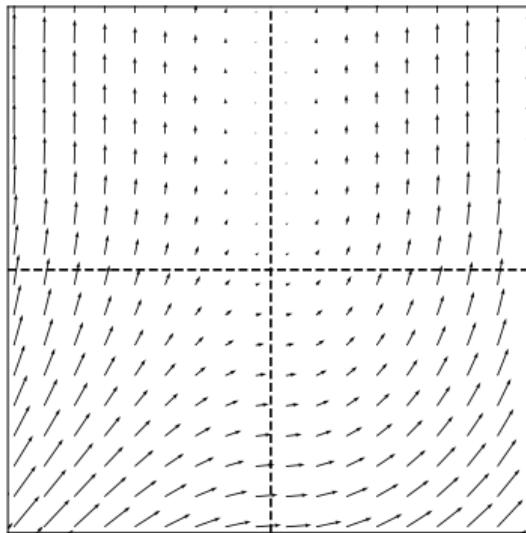
Arbitrary Transformations

- ▶ Arbitrary transformations can be quite complex.



Arbitrary Transformations

- ▶ Arbitrary transformations can be quite complex.



Linear Transformations

- ▶ Luckily, we often¹ work with simpler, **linear transformations**.
- ▶ A transformation f is linear if:

$$\vec{f}(\alpha \vec{x} + \beta \vec{y}) = \alpha \vec{f}(\vec{x}) + \beta \vec{f}(\vec{y})$$

¹Sometimes, just to make the math tractable!

Checking Linearity

- To check if a transformation is linear, use the definition.

$$f(\vec{z}) = (z_2, -z_1)^T \quad \vec{f}(\alpha \vec{x} + \beta \vec{y}) = \alpha f(\vec{x}) + \beta f(\vec{y})$$

- Example: $\vec{f}(\vec{x}) = (x_2, -x_1)^T$

Let $\vec{x} = (x_1, x_2)^T$ $\vec{y} = (y_1, y_2)^T$.

Then:

$$f(\alpha \vec{x} + \beta \vec{y}) = f\left(\alpha \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} + \beta \begin{pmatrix} y_1 \\ y_2 \end{pmatrix}\right)$$

$$= f\left(\begin{pmatrix} \alpha x_1 \\ \alpha x_2 \end{pmatrix} + \begin{pmatrix} \beta y_1 \\ \beta y_2 \end{pmatrix}\right)$$

$$\begin{aligned} &= f\left(\begin{pmatrix} \alpha x_1 + \beta y_1 \\ \alpha x_2 + \beta y_2 \end{pmatrix}\right) \\ &= \begin{pmatrix} \alpha x_2 + \beta y_2 \\ -(\alpha x_1 + \beta y_1) \end{pmatrix} \\ &= \alpha \begin{pmatrix} x_2 \\ -x_1 \end{pmatrix} + \beta \begin{pmatrix} y_2 \\ -y_1 \end{pmatrix} \\ &= \alpha f(\vec{x}) + \beta f(\vec{y}) \end{aligned}$$

$$(x_1^2 + x_2, x_2 - x_1^3)$$

Exercise

Let $\vec{f}(\vec{x}) = (x_1 + 3, x_2)$. Is \vec{f} a linear transformation?

$$f(3(0) + 4(1)) \neq 3f(0) + 4f(1)$$

Implications of Linearity

- ▶ Suppose \vec{f} is a linear transformation. Then:

$$\begin{aligned}\vec{f}(\vec{x}) &= \vec{f}(x_1 \hat{e}^{(1)} + x_2 \hat{e}^{(2)}) \\ &= x_1 \vec{f}(\hat{e}^{(1)}) + x_2 \vec{f}(\hat{e}^{(2)})\end{aligned}$$

- ▶ I.e., \vec{f} is **totally determined** by what it does to the basis vectors.

The **Complexity** of Arbitrary Transformations

- ▶ Suppose f is an **arbitrary** transformation.
- ▶ I tell you $\vec{f}(\hat{e}^{(1)}) = (2, 1)^T$ and $\vec{f}(\hat{e}^{(2)}) = (-3, 0)^T$.
- ▶ I tell you $\vec{x} = (x_1, x_2)^T$.
- ▶ What is $\vec{f}(\vec{x})$?

The **Simplicity** of Linear Transformations

- ▶ Suppose f is a **linear** transformation.
- ▶ I tell you $\vec{f}(\hat{e}^{(1)}) = (2, 1)^T$ and $\vec{f}(\hat{e}^{(2)}) = (-3, 0)^T$.
- ▶ I tell you $\vec{x} = (x_1, x_2)^T$.
- ▶ What is $\vec{f}(\vec{x})$?

Exercise

- ▶ Suppose f is a **linear** transformation.
- ▶ I tell you $\vec{f}(\hat{e}^{(1)}) = (2, 1)^T$ and $\vec{f}(\hat{e}^{(2)}) = (-3, 0)^T$.
- ▶ I tell you $\vec{x} = (3, -4)^T$.
- ▶ What is $\vec{f}(\vec{x})$?

Key Fact

- ▶ Linear functions are determined **entirely** by what they do on the basis vectors.
- ▶ I.e., to tell you what f does, I only need to tell you $\vec{f}(\hat{e}^{(1)})$ and $\vec{f}(\hat{e}^{(2)})$.
- ▶ This makes the math easy!



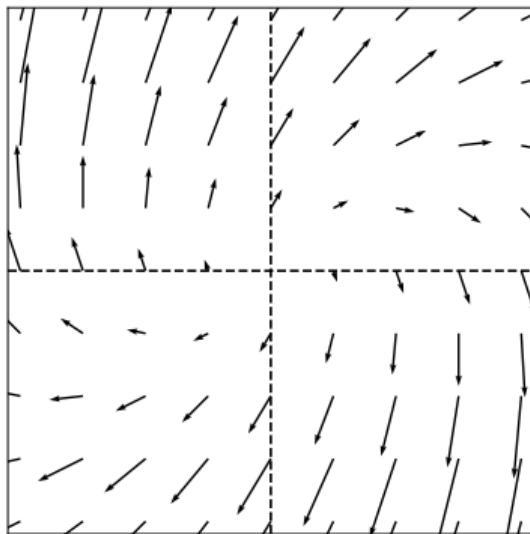
A photograph of a lush, overgrown garden with dense greenery and trees in the background. The garden features a variety of plants, including tall grasses, low shrubs, and colorful flowers, creating a complex and intricate scene.

Arbitrary
Transformations

Linear
Transformations

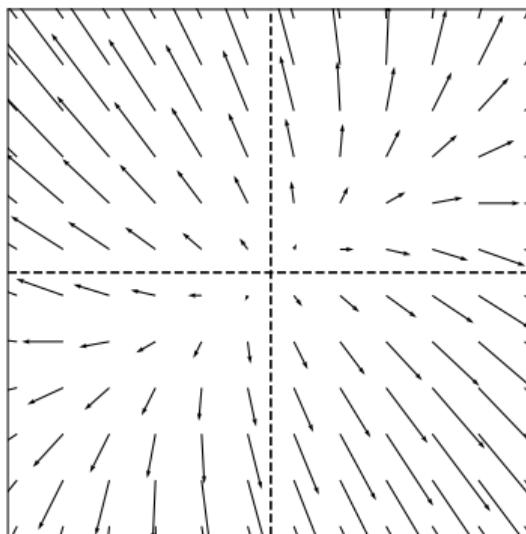
Example Linear Transformation

► $\vec{f}(\vec{x}) = (x_1 + 3x_2, -3x_1 + 5x_2)^T$



Another Example Linear Transformation

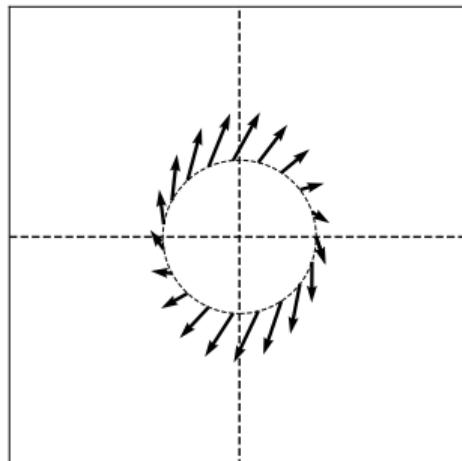
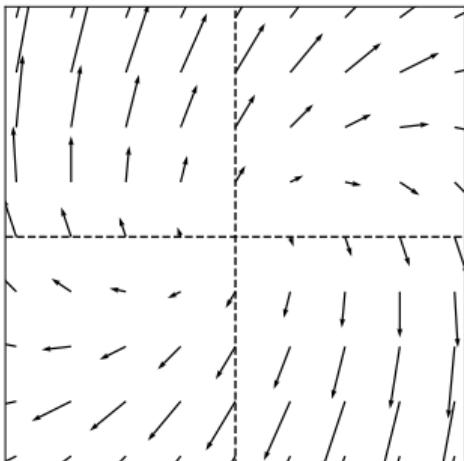
► $\vec{f}(\vec{x}) = (2x_1 - x_2, -x_1 + 3x_2)^T$

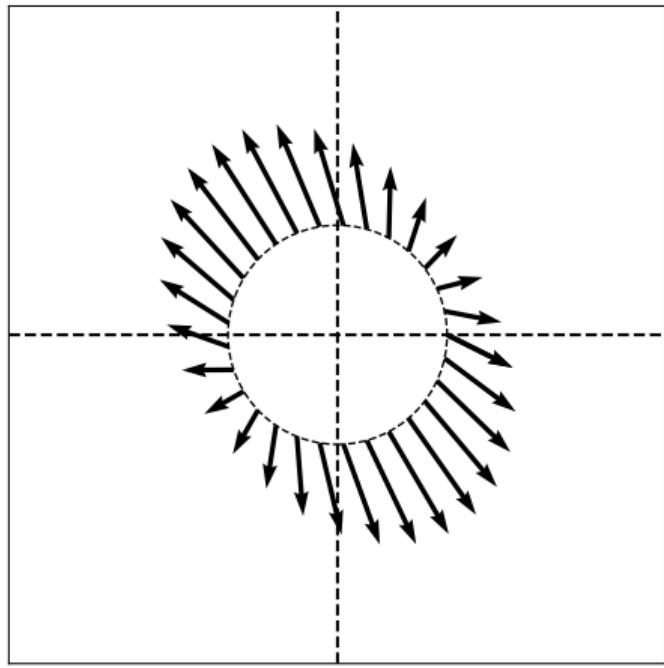
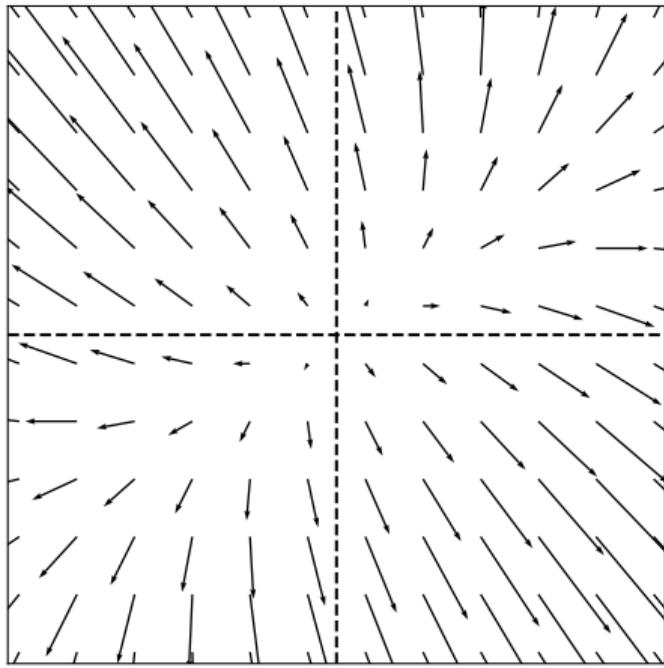


Note

- ▶ Because of linearity, along any given direction \vec{f} changes only in scale.

$$\vec{f}(\lambda \hat{x}) = \lambda \vec{f}(\hat{x})$$





Linear Transformations and Bases

- ▶ We have been writing transformations in coordinate form. For example:

$$\vec{f}(\vec{x}) = (x_1 + x_2, x_1 - x_2)^T$$

- ▶ To do so, we assumed the **standard basis**.
- ▶ If we use a different basis, the formula for \vec{f} changes.

Example

- ▶ Suppose that in the standard basis, $\vec{f}(\vec{x}) = (x_1 + x_2, x_1 - x_2)^T$.
- ▶ Let $\hat{u}^{(1)} = \frac{1}{\sqrt{2}}(1, 1)^T$ and $\hat{u}^{(2)} = \frac{1}{\sqrt{2}}(-1, 1)^T$.
- ▶ Write $[\vec{x}]_{\mathcal{U}} = (z_1, z_2)^T$.
- ▶ What is $[\vec{f}(\vec{x})]_{\mathcal{U}}$ in terms of z_1 and z_2 ?

DSC 140B

Representation Learning

Lecture 02 | Part 4

Matrices

Matrices?

- ▶ I thought this was supposed to be about linear algebra... Where are the matrices?

Matrices?

- ▶ I thought this was supposed to be about linear algebra... Where are the matrices?
- ▶ What is a matrix, anyways?

What is a matrix?

$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$$

Recall: Linear Transformations

- ▶ A **transformation** $\vec{f}(\vec{x})$ is a function which takes a vector as input and returns a vector of the same dimensionality.
- ▶ A transformation f is **linear** if

$$\vec{f}(\alpha \vec{u} + \beta \vec{v}) = \alpha \vec{f}(\vec{u}) + \beta \vec{f}(\vec{v})$$

Recall: Linear Transformations

- ▶ A **key** property: to compute $\vec{f}(\vec{x})$, we only need to know what f does to basis vectors.
- ▶ Example:

$$\vec{x} = 3\hat{e}^{(1)} - 4\hat{e}^{(2)} = \begin{pmatrix} 3 \\ -4 \end{pmatrix}$$

$$\vec{f}(\hat{e}^{(1)}) = -\hat{e}^{(1)} + 3\hat{e}^{(2)}$$

$$\vec{f}(\hat{e}^{(2)}) = 2\hat{e}^{(1)}$$

$$\vec{f}(\vec{x}) =$$

Matrices

- ▶ f defined by what it does to basis vectors
- ▶ Place $\vec{f}(\hat{e}^{(1)})$, $\vec{f}(\hat{e}^{(2)})$, ... into a table as columns
- ▶ This is the **matrix** representing² f

$$\begin{aligned}\vec{f}(\hat{e}^{(1)}) &= -\hat{e}^{(1)} + 3\hat{e}^{(2)} = \begin{pmatrix} -1 \\ 3 \end{pmatrix} & \begin{pmatrix} -1 & 2 \\ 3 & 0 \end{pmatrix} \\ \vec{f}(\hat{e}^{(2)}) &= 2\hat{e}^{(1)} = \begin{pmatrix} 2 \\ 0 \end{pmatrix}\end{aligned}$$

²with respect to the standard basis $\hat{e}^{(1)}, \hat{e}^{(2)}$

Example

$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$$

$$\vec{f}(\hat{e}^{(1)}) = (1, 4, 7)^T$$

$$\vec{f}(\hat{e}^{(2)}) = (2, 5, 7)^T$$

$$\vec{f}(\hat{e}^{(3)}) = (3, 6, 9)^T$$

Main Idea

A square $(n \times n)$ matrix can be interpreted as a compact representation of a linear transformation $f : \mathbb{R}^n \rightarrow \mathbb{R}^n$.

What is matrix multiplication?

$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix} \begin{pmatrix} -2 \\ 1 \\ 3 \end{pmatrix} = \begin{pmatrix} \quad \\ \quad \end{pmatrix}$$

A low-level definition

$$(A\vec{x})_i = \sum_{j=1}^n A_{ij}x_j$$

A low-level interpretation

$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix} \begin{pmatrix} -2 \\ 1 \\ 3 \end{pmatrix} = -2 \begin{pmatrix} 1 \\ 4 \\ 7 \end{pmatrix} + 1 \begin{pmatrix} 2 \\ 5 \\ 8 \end{pmatrix} + 3 \begin{pmatrix} 3 \\ 6 \\ 9 \end{pmatrix}$$

In general...

$$\begin{pmatrix} \uparrow & \uparrow & \uparrow \\ \vec{a}^{(1)} & \vec{a}^{(2)} & \vec{a}^{(3)} \\ \downarrow & \downarrow & \downarrow \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = x_1 \vec{a}^{(1)} + x_2 \vec{a}^{(2)} + x_3 \vec{a}^{(3)}$$

Matrix Multiplication

$$\vec{x} = x_1 \hat{e}^{(1)} + x_2 \hat{e}^{(2)} + x_3 \hat{e}^{(3)} = (x_1, x_2, x_3)^T$$
$$\vec{f}(\vec{x}) = x_1 \vec{f}(\hat{e}^{(1)}) + x_2 \vec{f}(\hat{e}^{(2)}) + x_3 \vec{f}(\hat{e}^{(3)})$$

$$A = \begin{pmatrix} \vec{f}(\hat{e}^{(1)}) & \vec{f}(\hat{e}^{(2)}) & \vec{f}(\hat{e}^{(3)}) \\ \downarrow & \downarrow & \downarrow \end{pmatrix}$$
$$A\vec{x} = \begin{pmatrix} \vec{f}(\hat{e}^{(1)}) & \vec{f}(\hat{e}^{(2)}) & \vec{f}(\hat{e}^{(3)}) \\ \downarrow & \downarrow & \downarrow \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}$$
$$= x_1 \vec{f}(\hat{e}^{(1)}) + x_2 \vec{f}(\hat{e}^{(2)}) + x_3 \vec{f}(\hat{e}^{(3)})$$

Matrix Multiplication

- ▶ Matrix A represents a linear transformation \vec{f}
 - ▶ With respect to the standard basis
 - ▶ If we use a different basis, the matrix changes!
- ▶ Matrix multiplication $A\vec{x}$ **evaluates** $\vec{f}(\vec{x})$

What are they, *really*?

- ▶ Matrices are sometimes just tables of numbers.
- ▶ But they often have a deeper meaning.

Main Idea

A square ($n \times n$) matrix can be interpreted as a compact representation of a linear transformation $\vec{f} : \mathbb{R}^n \rightarrow \mathbb{R}^n$.

What's more, if A represents \vec{f} , then $A\vec{x} = \vec{f}(\vec{x})$; that is, multiplying by A is the same as evaluating \vec{f} .

Example

$$\vec{x} = 3\hat{e}^{(1)} - 4\hat{e}^{(2)} = \begin{pmatrix} 3 \\ -4 \end{pmatrix} \quad A =$$

$$\vec{f}(\hat{e}^{(1)}) = -\hat{e}^{(1)} + 3\hat{e}^{(2)}$$

$$\vec{f}(\hat{e}^{(2)}) = 2\hat{e}^{(1)}$$

$$\vec{f}(\vec{x}) =$$

$$A\vec{x} =$$

Note

- ▶ All of this works because we assumed \vec{f} is **linear**.
- ▶ If it isn't, evaluating \vec{f} isn't so simple.

Note

- ▶ All of this works because we assumed \vec{f} is **linear**.
- ▶ If it isn't, evaluating \vec{f} isn't so simple.
- ▶ Linear algebra = simple!

DSC 140B

Representation Learning

Lecture 03 | Part 1

Functions of a Vector

Functions of a Vector

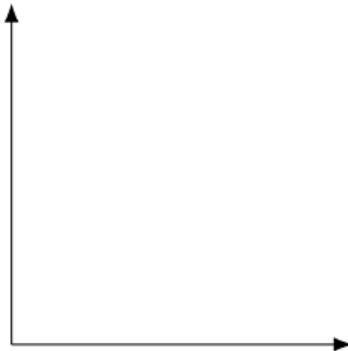
- ▶ In ML, we often work with functions of a vector:
 $f : \mathbb{R}^d \rightarrow \mathbb{R}^{d'}$.
- ▶ Example: a prediction function, $H(\vec{x})$.
- ▶ Functions of a vector can return:
 - ▶ a number: $f : \mathbb{R}^d \rightarrow \mathbb{R}^1$
 - ▶ a vector $\vec{f} : \mathbb{R}^d \rightarrow \mathbb{R}^{d'}$
 - ▶ something else?

Transformations

- ▶ A **transformation** \vec{f} is a function that takes in a vector, and returns a vector *of the same dimensionality*.
- ▶ That is, $\vec{f} : \mathbb{R}^d \rightarrow \mathbb{R}^d$.

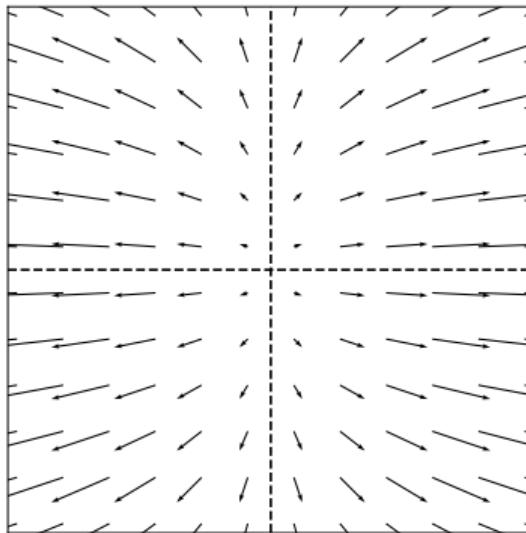
Visualizing Transformations

- ▶ A transformation is a **vector field**.
 - ▶ Assigns a vector to each point in space.
 - ▶ Example: $\vec{f}(\vec{x}) = (3x_1, x_2)^T$



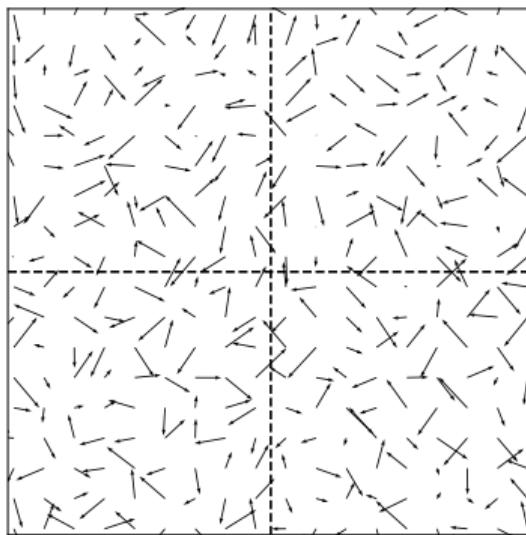
Example

► $\vec{f}(\vec{x}) = (3x_1, x_2)^T$



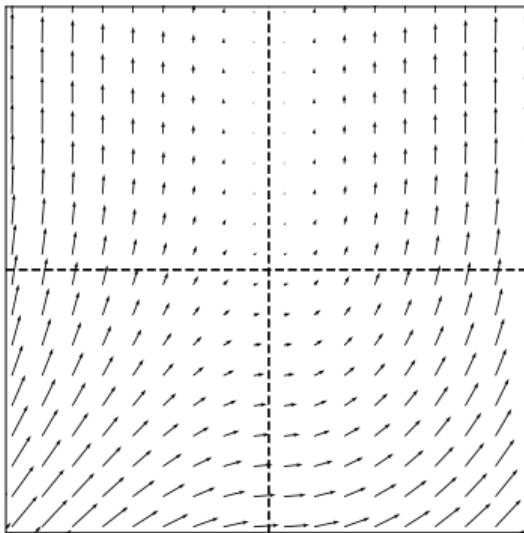
Arbitrary Transformations

- ▶ Arbitrary transformations can be quite complex.



Arbitrary Transformations

- ▶ Arbitrary transformations can be quite complex.



Linear Transformations

- ▶ Luckily, we often¹ work with simpler, **linear transformations**.

- ▶ A transformation f is linear if:

$$\vec{f}(\alpha \vec{x} + \beta \vec{y}) = \alpha \vec{f}(\vec{x}) + \beta \vec{f}(\vec{y})$$

¹Sometimes, just to make the math tractable!

Checking Linearity

- ▶ To check if a transformation is linear, use the definition.
- ▶ Example: $\vec{f}(\vec{x}) = (x_2, -x_1)^T$

$$\vec{f}(\alpha \vec{x} + \beta \vec{y}) = \dots = \alpha f(\vec{x}) + \beta f(\vec{y})$$

$$\vec{x} = \begin{pmatrix} x_1 \\ \vdots \\ x_d \end{pmatrix} \quad \vec{y} = \begin{pmatrix} y_1 \\ \vdots \\ y_d \end{pmatrix}$$

$$\alpha \vec{a} + \beta \vec{b} \quad \vec{f}(\alpha \vec{a} + \beta \vec{b}) \neq \alpha f(\vec{a}) + \beta f(\vec{b})$$

$$\begin{matrix} \alpha = 2 \\ \beta = 3 \end{matrix} \quad \vec{a} = \begin{pmatrix} 1 \\ 1 \end{pmatrix} \quad \vec{b} = \begin{pmatrix} 2 \\ 3 \end{pmatrix}$$

Exercise

Let $\vec{f}(\vec{x}) = (x_1 + 3, x_2)$. Is \vec{f} a linear transformation?

$$\alpha \vec{a} + \beta \vec{b} = 2 \begin{pmatrix} 1 \\ 1 \end{pmatrix} + 3 \begin{pmatrix} 2 \\ 3 \end{pmatrix} = \begin{pmatrix} 2 \\ 2 \end{pmatrix} + \begin{pmatrix} 6 \\ 9 \end{pmatrix} = \begin{pmatrix} 8 \\ 11 \end{pmatrix}$$

$$\vec{f}(\alpha \vec{a} + \beta \vec{b}) = \vec{f}\left(\begin{pmatrix} 8 \\ 11 \end{pmatrix}\right) = \begin{pmatrix} 11 \\ 11 \end{pmatrix}$$

$$\alpha f(\vec{a}) + \beta f(\vec{b}) = 2 f\left(\begin{pmatrix} 1 \\ 1 \end{pmatrix}\right) + 3 f\left(\begin{pmatrix} 2 \\ 3 \end{pmatrix}\right) = 2 \begin{pmatrix} 4 \\ 1 \end{pmatrix} + 3 \begin{pmatrix} 5 \\ 3 \end{pmatrix} = \begin{pmatrix} 23 \\ 11 \end{pmatrix}$$

$$\alpha = 2 \quad \vec{a} = \begin{pmatrix} 1 \\ 1 \end{pmatrix} \quad \vec{b} = \begin{pmatrix} 2 \\ 1 \end{pmatrix}$$

Exercise

Let $\vec{f}(\vec{x}) = (x_1 + 3, x_2)$. Is \vec{f} a linear transformation?

$$\alpha \vec{a} + \beta \vec{b} = 2 \begin{pmatrix} 1 \\ 1 \end{pmatrix} + 3 \begin{pmatrix} 2 \\ 1 \end{pmatrix} = \begin{pmatrix} 2 \\ 2 \end{pmatrix} + \begin{pmatrix} 6 \\ 3 \end{pmatrix} = \begin{pmatrix} 8 \\ 5 \end{pmatrix}$$

$$\vec{f}(\alpha \vec{a} + \beta \vec{b}) = \vec{f}\left(\begin{pmatrix} 8 \\ 5 \end{pmatrix}\right) = \begin{pmatrix} 11 \\ 5 \end{pmatrix}$$

$$\alpha f(\vec{a}) + \beta f(\vec{b}) = 2 f\left(\begin{pmatrix} 1 \\ 1 \end{pmatrix}\right) + 3 f\left(\begin{pmatrix} 2 \\ 1 \end{pmatrix}\right) = 2 \begin{pmatrix} 4 \\ 1 \end{pmatrix} + 3 \begin{pmatrix} 5 \\ 1 \end{pmatrix} = \begin{pmatrix} 23 \\ 5 \end{pmatrix}$$

Implications of Linearity

- ▶ Suppose \vec{f} is a linear transformation. Then:

$$\vec{x} = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$$

$$\begin{aligned}\vec{f}(\vec{x}) &= \vec{f}(x_1 \hat{e}^{(1)} + x_2 \hat{e}^{(2)}) \\ &= x_1 \vec{f}(\hat{e}^{(1)}) + x_2 \vec{f}(\hat{e}^{(2)})\end{aligned}$$

- ▶ I.e., \vec{f} is **totally determined** by what it does to the basis vectors.

The **Complexity** of Arbitrary Transformations

- ▶ Suppose f is an **arbitrary** transformation.
- ▶ I tell you $\vec{f}(\hat{e}^{(1)}) = (2, 1)^T$ and $\vec{f}(\hat{e}^{(2)}) = (-3, 0)^T$.
- ▶ I tell you $\vec{x} = (x_1, x_2)^T$.
- ▶ What is $\vec{f}(\vec{x})$?

The **Simplicity** of Linear Transformations

- ▶ Suppose f is a **linear** transformation.
- ▶ I tell you $\vec{f}(\hat{e}^{(1)}) = (2, 1)^T$ and $\vec{f}(\hat{e}^{(2)}) = (-3, 0)^T$.
- ▶ I tell you $\vec{x} = (x_1, x_2)^T$.
- ▶ What is $\vec{f}(\vec{x})$?

$$\vec{f}(\vec{x}) = \vec{f}(x_1 \hat{e}^{(1)} + x_2 \hat{e}^{(2)}) = x_1 f(\hat{e}^{(1)}) + x_2 f(\hat{e}^{(2)})$$

↑
because of linearity!

Exercise

- ▶ Suppose f is a **linear** transformation.
- ▶ I tell you $\vec{f}(\hat{e}^{(1)}) = (2, 1)^T$ and $\vec{f}(\hat{e}^{(2)}) = (-3, 0)^T$.
- ▶ I tell you $\vec{x} = (3, -4)^T$.
- ▶ What is $\vec{f}(\vec{x})$?

$$\begin{aligned}
 f\left(\begin{pmatrix} 3 \\ -4 \end{pmatrix}\right) &= 3f(\hat{e}^{(1)}) - 4f(\hat{e}^{(2)}) \\
 &= 3\begin{pmatrix} 2 \\ 1 \end{pmatrix} - 4\begin{pmatrix} -3 \\ 0 \end{pmatrix} \\
 &= \begin{pmatrix} 6 \\ 3 \end{pmatrix} + \begin{pmatrix} 12 \\ 0 \end{pmatrix} = \begin{pmatrix} 18 \\ 3 \end{pmatrix}
 \end{aligned}$$

$$\begin{aligned}
 f\left(\begin{pmatrix} 2 \\ -1 \end{pmatrix}\right) &= 2f(\hat{e}^{(1)}) - 1f(\hat{e}^{(2)}) \\
 &= 2\begin{pmatrix} 2 \\ 1 \end{pmatrix} - 1\begin{pmatrix} -3 \\ 0 \end{pmatrix} \\
 &= \begin{pmatrix} 4 \\ 2 \end{pmatrix} + \begin{pmatrix} 3 \\ 0 \end{pmatrix} = \begin{pmatrix} 7 \\ 2 \end{pmatrix}
 \end{aligned}$$

Exercise

- ▶ Suppose f is a **linear** transformation.
- ▶ I tell you $\vec{f}(\hat{e}^{(1)}) = (2, 1)^T$ and $\vec{f}(\hat{e}^{(2)}) = (-3, 0)^T$.
- ▶ I tell you $\vec{x} = (3, -4)^T$.
- ▶ What is $\vec{f}(\vec{x})$?

$$\vec{x} = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$$

$$\begin{aligned}\vec{f}(\vec{x}) &= x_1 \begin{pmatrix} 2 \\ 1 \end{pmatrix} + x_2 \begin{pmatrix} -3 \\ 0 \end{pmatrix} \\ &= \begin{pmatrix} 2x_1 - 3x_2 \\ x_1 \end{pmatrix}\end{aligned}$$

Key Fact

- ▶ Linear functions are determined **entirely** by what they do on the basis vectors.
- ▶ I.e., to tell you what f does, I only need to tell you $\vec{f}(\hat{e}^{(1)})$ and $\vec{f}(\hat{e}^{(2)})$.
- ▶ This makes the math easy!



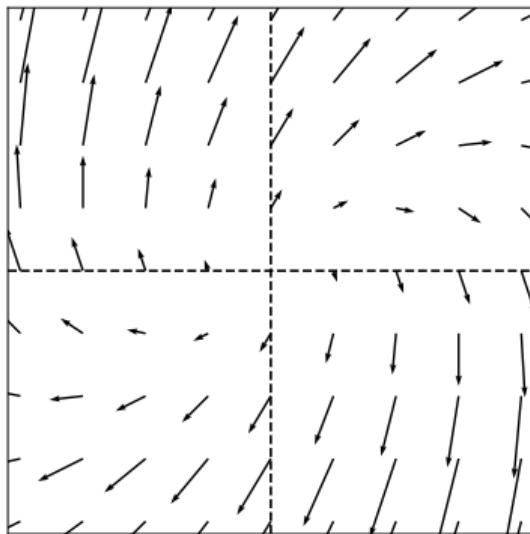
Arbitrary
Transformations

Linear
Transformations



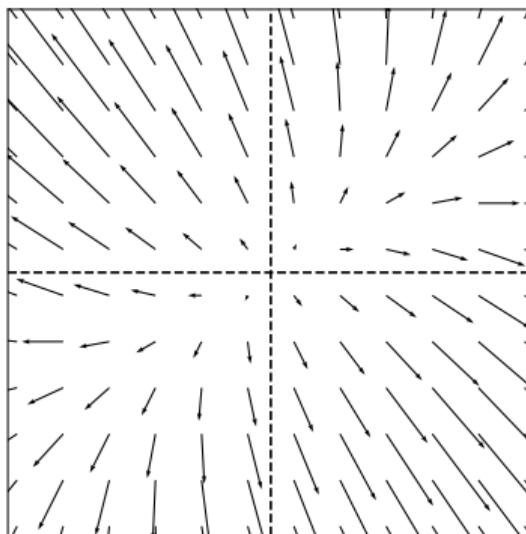
Example Linear Transformation

► $\vec{f}(\vec{x}) = (x_1 + 3x_2, -3x_1 + 5x_2)^T$



Another Example Linear Transformation

► $\vec{f}(\vec{x}) = (2x_1 - x_2, -x_1 + 3x_2)^T$

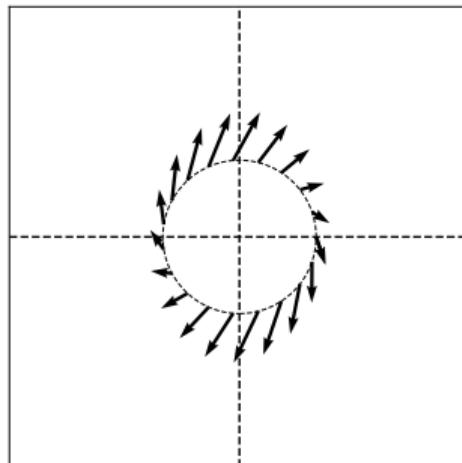
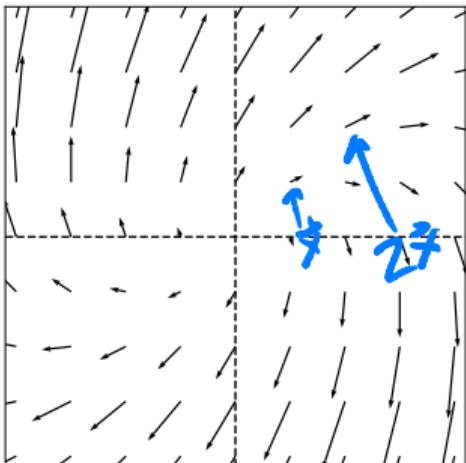


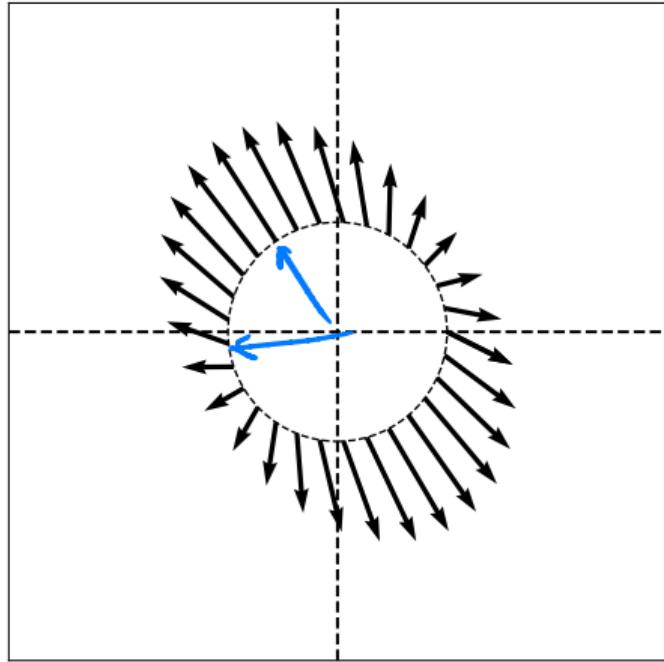
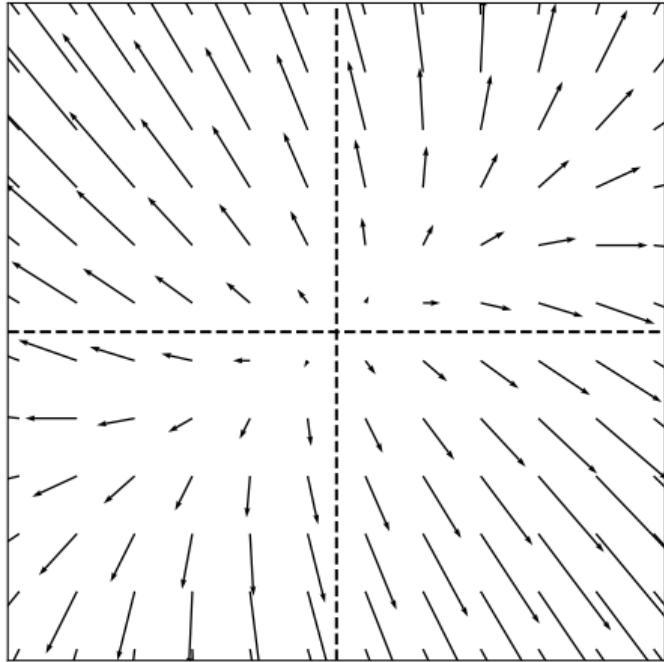
Note

- ▶ Because of linearity, along any given direction \vec{f} changes only in scale.

$$\vec{f}(\lambda \hat{x}) = \lambda \vec{f}(\hat{x})$$

$$f(\vec{x})$$
$$f(2\vec{x}) = 2f(\vec{x})$$



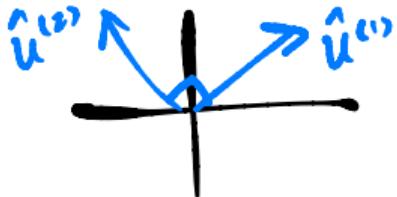


Linear Transformations and Bases

- ▶ We have been writing transformations in coordinate form. For example:

$$\vec{f}(\vec{x}) = (x_1 + x_2, x_1 - x_2)^T$$

- ▶ To do so, we assumed the **standard basis**.
- ▶ If we use a different basis, the formula for \vec{f} changes.



Example

2 6

- ▶ Suppose that in the standard basis, $\vec{f}(\vec{x}) = (x_1 + x_2, x_1 - x_2)^T$.
- ▶ Let $\hat{u}^{(1)} = \frac{1}{\sqrt{2}}(1, 1)^T$ and $\hat{u}^{(2)} = \frac{1}{\sqrt{2}}(-1, 1)^T$.
- ▶ Write $[\vec{x}]_{\mathcal{U}} = (z_1, z_2)^T$. $\rightarrow \vec{x} = z_1 \hat{u}^{(1)} + z_2 \hat{u}^{(2)}$
- ▶ What is $[\vec{f}(\vec{x})]_{\mathcal{U}}$ in terms of z_1 and z_2 ?

$$\vec{f}(\vec{x}) = \alpha \hat{u}^{(1)} + \beta \hat{u}^{(2)}$$

$$\begin{aligned}\vec{f}(\hat{u}^{(1)}) &= \vec{f}\left(\frac{1}{\sqrt{2}}\begin{pmatrix} 1 \\ 1 \end{pmatrix}\right) = \frac{1}{\sqrt{2}}\vec{f}\begin{pmatrix} 1 \\ 1 \end{pmatrix} = \frac{1}{\sqrt{2}}\begin{pmatrix} 2 \\ 0 \end{pmatrix} = \frac{2}{\sqrt{2}}\hat{e}^{(1)} \\ \vec{f}(\hat{u}^{(2)}) &= \vec{f}\left(\frac{1}{\sqrt{2}}\begin{pmatrix} -1 \\ 1 \end{pmatrix}\right) = \frac{1}{\sqrt{2}}\vec{f}\begin{pmatrix} -1 \\ 1 \end{pmatrix} = \frac{1}{\sqrt{2}}\begin{pmatrix} 0 \\ -2 \end{pmatrix} = \frac{-2}{\sqrt{2}}\hat{e}^{(2)}\end{aligned}$$

Example

- ▶ Suppose that in the standard basis, $\vec{f}(\vec{x}) = (x_1 + x_2, x_1 - x_2)^T$.
- ▶ Let $\hat{u}^{(1)} = \frac{1}{\sqrt{2}}(1, 1)^T$ and $\hat{u}^{(2)} = \frac{1}{\sqrt{2}}(-1, 1)^T$.
- ▶ Write $[\vec{x}]_{\mathcal{U}} = (z_1, z_2)^T$.
- ▶ What is $[\vec{f}(\vec{x})]_{\mathcal{U}}$ in terms of z_1 and z_2 ?

$$\vec{f}(\vec{u}^{(1)}) = \frac{1}{\sqrt{2}} \begin{pmatrix} 2 \\ 0 \end{pmatrix} \quad [\vec{f}(\vec{u}^{(1)})]_{\mathcal{U}} = \begin{pmatrix} \vec{f}(\hat{u}^{(1)}) \cdot \hat{u}^{(1)} \\ \vec{f}(\hat{u}^{(1)}) \cdot \hat{u}^{(2)} \end{pmatrix}$$

$$\vec{f}(\hat{u}^{(1)}) \cdot \hat{u}^{(1)} = \frac{1}{\sqrt{2}} \begin{pmatrix} 2 \\ 0 \end{pmatrix} \cdot \frac{1}{\sqrt{2}} \begin{pmatrix} 1 \\ 1 \end{pmatrix} = \frac{1}{2} 2 = 1$$

$$\vec{f}(\hat{u}^{(1)}) \cdot \hat{u}^{(2)} = \frac{1}{\sqrt{2}} \begin{pmatrix} 2 \\ 0 \end{pmatrix} \cdot \frac{1}{\sqrt{2}} \begin{pmatrix} -1 \\ 1 \end{pmatrix} = \frac{1}{2} (-2) = -1$$

$$[\vec{f}(\vec{u}^{(1)})]_{\mathcal{U}} = \begin{pmatrix} 1 \\ -1 \end{pmatrix} \Leftrightarrow \vec{f}(\vec{u}^{(1)}) = \hat{u}^{(1)} - \hat{u}^{(2)}$$

Example

- ▶ Suppose that in the standard basis, $\vec{f}(\vec{x}) = (x_1 + x_2, x_1 - x_2)^T$.
- ▶ Let $\hat{u}^{(1)} = \frac{1}{\sqrt{2}}(1, 1)^T$ and $\hat{u}^{(2)} = \frac{1}{\sqrt{2}}(-1, 1)^T$.
- ▶ Write $[\vec{x}]_{\mathcal{U}} = (z_1, z_2)^T$.
- ▶ What is $[\vec{f}(\vec{x})]_{\mathcal{U}}$ in terms of z_1 and z_2 ?

$$[\vec{f}(\hat{u}^{(1)})]_{\mathcal{U}} \quad \vec{f}(\hat{u}^{(1)}) = \vec{f}\left(\frac{1}{\sqrt{2}}\begin{pmatrix} 1 \\ 1 \end{pmatrix}\right) = \frac{1}{\sqrt{2}}\vec{f}\begin{pmatrix} 1 \\ 1 \end{pmatrix} = \frac{1}{\sqrt{2}}\begin{pmatrix} 0 \\ 2 \end{pmatrix}$$

$$\vec{f}(\hat{u}^{(1)}) \cdot \hat{u}^{(1)} = \frac{1}{\sqrt{2}}\begin{pmatrix} 0 \\ 2 \end{pmatrix} \cdot \frac{1}{\sqrt{2}}\begin{pmatrix} 1 \\ 1 \end{pmatrix} = \frac{1}{2} \cdot 2 = 1$$

$$\vec{f}(\hat{u}^{(2)}) \cdot \hat{u}^{(2)} = \frac{1}{\sqrt{2}}\begin{pmatrix} 0 \\ -2 \end{pmatrix} \cdot \frac{1}{\sqrt{2}}\begin{pmatrix} -1 \\ 1 \end{pmatrix} = \frac{1}{2} \cdot 2 = 1$$

$$[\vec{f}(\hat{u}^{(1)})]_{\mathcal{U}} = \begin{pmatrix} 1 \\ 1 \end{pmatrix} \Leftrightarrow \vec{f}(\hat{u}^{(1)}) = -\hat{u}^{(1)} - \hat{u}^{(2)}$$

$$\vec{f}(\hat{u}^{(1)}) = \hat{u}^{(1)} - \hat{u}^{(2)}$$

$$\vec{f}(\hat{u}^{(2)}) = -\hat{u}^{(1)} - \hat{u}^{(2)}$$

Example

- ▶ Suppose that in the standard basis, $\vec{f}(\vec{x}) = (x_1 + x_2, x_1 - x_2)^T$.
- ▶ Let $\hat{u}^{(1)} = \frac{1}{\sqrt{2}}(1, 1)^T$ and $\hat{u}^{(2)} = \frac{1}{\sqrt{2}}(-1, 1)^T$.
- ▶ Write $[\vec{x}]_{\mathcal{U}} = (z_1, z_2)^T$. $\vec{x} = z_1 \hat{u}^{(1)} + z_2 \hat{u}^{(2)}$
- ▶ What is $[\vec{f}(\vec{x})]_{\mathcal{U}}$ in terms of z_1 and z_2 ?

$$\vec{f}(\vec{x}) = z_1 \vec{f}(\hat{u}^{(1)}) + z_2 \vec{f}(\hat{u}^{(2)})$$

$$= z_1 [\hat{u}^{(1)} - \hat{u}^{(2)}] + z_2 [-\hat{u}^{(1)} - \hat{u}^{(2)}]$$

$$= (z_1 - z_2) \hat{u}^{(1)} + (-z_1 - z_2) \hat{u}^{(2)}$$

$$[\vec{f}(\vec{x})]_{\mathcal{U}} = \begin{pmatrix} z_1 - z_2 \\ -z_1 - z_2 \end{pmatrix}$$

DSC 140B

Representation Learning

Lecture 03 | Part 2

Matrices

Matrices?

- ▶ I thought this week was supposed to be about linear algebra... Where are the matrices?

Matrices?

- ▶ I thought this week was supposed to be about linear algebra... Where are the matrices?
- ▶ What is a matrix, anyways?

What is a matrix?

$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$$

Recall: Linear Transformations

- ▶ A **transformation** $\vec{f}(\vec{x})$ is a function which takes a vector as input and returns a vector of the same dimensionality.
- ▶ A transformation \vec{f} is **linear** if

$$\vec{f}(\alpha \vec{u} + \beta \vec{v}) = \alpha \vec{f}(\vec{u}) + \beta \vec{f}(\vec{v})$$

Recall: Linear Transformations

- ▶ Key consequence of **linearity**: to compute $\vec{f}(\vec{x})$, only need to know what \vec{f} does to basis vectors.
- ▶ Example:

$$\vec{x} = 3\hat{e}^{(1)} - 4\hat{e}^{(2)} = \begin{pmatrix} 3 \\ -4 \end{pmatrix}$$

$$\vec{f}(\hat{e}^{(1)}) = -\hat{e}^{(1)} + 3\hat{e}^{(2)} = (-1, 3)^T$$

$$\vec{f}(\hat{e}^{(2)}) = 2\hat{e}^{(1)} = (2, 0)^T$$

$$\vec{f}(\vec{x}) = \vec{f}\left(\begin{pmatrix} 3 \\ -4 \end{pmatrix}\right) = 3f(\hat{e}^{(1)}) - 4f(\hat{e}^{(2)}) = 3\begin{pmatrix} -1 \\ 3 \end{pmatrix} - 4\begin{pmatrix} 2 \\ 0 \end{pmatrix} = \begin{pmatrix} -3 \\ 9 \end{pmatrix} + \begin{pmatrix} -8 \\ 0 \end{pmatrix} = \begin{pmatrix} -11 \\ 9 \end{pmatrix}$$

Matrices

- ▶ **Idea:** Since \vec{f} is defined by what it does to basis, place $\vec{f}(\hat{e}^{(1)})$, $\vec{f}(\hat{e}^{(2)})$, ... into a table as columns
- ▶ This is the **matrix** representing² \vec{f}

$$\begin{aligned}\vec{f}(\hat{e}^{(1)}) &= -\hat{e}^{(1)} + 3\hat{e}^{(2)} = \begin{pmatrix} -1 \\ 3 \end{pmatrix} & \begin{pmatrix} -1 & 2 \\ 3 & 0 \end{pmatrix} \\ \vec{f}(\hat{e}^{(2)}) &= 2\hat{e}^{(1)} = \begin{pmatrix} 2 \\ 0 \end{pmatrix}\end{aligned}$$

²with respect to the standard basis $\hat{e}^{(1)}, \hat{e}^{(2)}$

Exercise

Write the matrix representing \vec{f} with respect to the standard basis, given:

$$\vec{f}(\hat{e}^{(1)}) = (1, 4, 7)^T$$

$$\vec{f}(\hat{e}^{(2)}) = (2, 5, 7)^T$$

$$\vec{f}(\hat{e}^{(3)}) = (3, 6, 9)^T$$

$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 7 & 9 \end{pmatrix}$$

$$\begin{pmatrix} -2 \\ -8 \\ -14 \end{pmatrix} + \begin{pmatrix} 2 \\ 5 \\ 8 \end{pmatrix} + \begin{pmatrix} 9 \\ 18 \\ 27 \end{pmatrix} = \begin{pmatrix} 9 \\ 15 \\ 21 \end{pmatrix}$$

Exercise

Suppose \vec{f} has the matrix below:

$$\begin{matrix} f(\hat{e}^{(1)}) & f(\hat{e}^{(2)}) \\ \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix} \\ f(\hat{e}^{(3)}) \end{matrix}$$

Let $\vec{x} = (-2, 1, 3)^T$. What is $\vec{f}(\vec{x})$?

$$\begin{aligned} f(\vec{x}) &= f\left(\begin{pmatrix} -2 \\ 1 \\ 3 \end{pmatrix}\right) = -2 f(\hat{e}^{(1)}) + 1 f(\hat{e}^{(2)}) + 3 f(\hat{e}^{(3)}) \\ &= -2 \begin{pmatrix} 1 \\ 4 \\ 7 \end{pmatrix} + 1 \begin{pmatrix} 2 \\ 5 \\ 8 \end{pmatrix} + 3 \begin{pmatrix} 3 \\ 6 \\ 9 \end{pmatrix} = \begin{pmatrix} -2 \\ -8 \\ -14 \end{pmatrix} + \begin{pmatrix} 2 \\ 5 \\ 8 \end{pmatrix} + \begin{pmatrix} 9 \\ 18 \\ 27 \end{pmatrix} \end{aligned}$$

Main Idea

A square $(n \times n)$ matrix can be interpreted as a compact representation of a linear transformation $f : \mathbb{R}^n \rightarrow \mathbb{R}^n$.

$$A \ B$$
$$\begin{matrix} \uparrow & \uparrow \\ f_A & f_B \end{matrix}$$

$$f_A(f_B(\vec{x}))$$

What is matrix multiplication?

$$\vec{f} \quad \leftarrow \vec{x} \quad \leftarrow \vec{f}(\vec{x})$$
$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix} \begin{pmatrix} -2 \\ 1 \\ 3 \end{pmatrix} = \begin{pmatrix} 9 \\ 15 \\ 21 \end{pmatrix}$$

$$(1)(-2) + (2)(1) + (3)(3) = -2 + 2 + 9 = 9$$

$$-14 + 8 + 27$$

A low-level definition

$$(A\vec{x})_i = \sum_{j=1}^n A_{ij}x_j$$

A low-level interpretation

$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix} \begin{pmatrix} -2 \\ 1 \\ 3 \end{pmatrix} = -2 \begin{pmatrix} 1 \\ 4 \\ 7 \end{pmatrix} + 1 \begin{pmatrix} 2 \\ 5 \\ 8 \end{pmatrix} + 3 \begin{pmatrix} 3 \\ 6 \\ 9 \end{pmatrix}$$

\vec{x}

$x_1 \quad x_2 \quad x_3$

$f(\vec{x}^{(1)}) \quad f(\vec{x}^{(2)}) \quad f(\vec{x}^{(3)})$

In general...

$$\begin{pmatrix} \uparrow & \uparrow & \uparrow \\ \vec{a}^{(1)} & \vec{a}^{(2)} & \vec{a}^{(3)} \\ \downarrow & \downarrow & \downarrow \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = x_1 \vec{a}^{(1)} + x_2 \vec{a}^{(2)} + x_3 \vec{a}^{(3)}$$

Matrix Multiplication

$$\vec{x} = x_1 \hat{e}^{(1)} + x_2 \hat{e}^{(2)} + x_3 \hat{e}^{(3)} = (x_1, x_2, x_3)^T$$
$$\vec{f}(\vec{x}) = x_1 \vec{f}(\hat{e}^{(1)}) + x_2 \vec{f}(\hat{e}^{(2)}) + x_3 \vec{f}(\hat{e}^{(3)})$$

$$A = \begin{pmatrix} \vec{f}(\hat{e}^{(1)}) & \vec{f}(\hat{e}^{(2)}) & \vec{f}(\hat{e}^{(3)}) \\ \downarrow & \downarrow & \downarrow \end{pmatrix}$$
$$A\vec{x} = \begin{pmatrix} \vec{f}(\hat{e}^{(1)}) & \vec{f}(\hat{e}^{(2)}) & \vec{f}(\hat{e}^{(3)}) \\ \downarrow & \downarrow & \downarrow \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}$$
$$= x_1 \vec{f}(\hat{e}^{(1)}) + x_2 \vec{f}(\hat{e}^{(2)}) + x_3 \vec{f}(\hat{e}^{(3)})$$

$$(\vec{x}^\top \vec{x})^{-1} \vec{x}^\top \vec{b}$$

Matrix Multiplication

- ▶ Matrix A represents a linear transformation \vec{f}
 - ▶ With respect to the standard basis
 - ▶ If we use a different basis, the matrix changes!
- ▶ Matrix multiplication $A\vec{x}$ **evaluates** $\vec{f}(\vec{x})$

What are they, *really*?

- ▶ Matrices are sometimes just tables of numbers.
- ▶ But they often have a deeper meaning.

Main Idea

A square ($n \times n$) matrix can be interpreted as a compact representation of a linear transformation $\vec{f} : \mathbb{R}^n \rightarrow \mathbb{R}^n$.

What's more, if A represents \vec{f} , then $A\vec{x} = \vec{f}(\vec{x})$; that is, multiplying by A is the same as evaluating \vec{f} .

Example

$$\vec{x} = 3\hat{e}^{(1)} - 4\hat{e}^{(2)} = \begin{pmatrix} 3 \\ -4 \end{pmatrix}$$

$$\vec{f}(\hat{e}^{(1)}) = -\hat{e}^{(1)} + 3\hat{e}^{(2)} = \begin{pmatrix} -1 \\ 3 \end{pmatrix}$$

$$\vec{f}(\hat{e}^{(2)}) = 2\hat{e}^{(1)} = \begin{pmatrix} 2 \\ 0 \end{pmatrix}$$

$$\vec{f}(\vec{x}) = \begin{pmatrix} -11 \\ 9 \end{pmatrix}$$

$$A = \begin{pmatrix} -1 & 2 \\ 3 & 0 \end{pmatrix}$$

$$\begin{aligned} A\vec{x} &= \begin{pmatrix} -1 & 2 \\ 3 & 0 \end{pmatrix} \begin{pmatrix} 3 \\ -4 \end{pmatrix} \\ &= \begin{pmatrix} -3 - 8 \\ 9 + 0 \end{pmatrix} = \begin{pmatrix} -11 \\ 9 \end{pmatrix} \end{aligned}$$

Note

- ▶ All of this works because we assumed \vec{f} is **linear**.
- ▶ If it isn't, evaluating \vec{f} isn't so simple.

Note

- ▶ All of this works because we assumed \vec{f} is **linear**.
- ▶ If it isn't, evaluating \vec{f} isn't so simple.
- ▶ Linear algebra = simple!

Matrices in Other Bases

- ▶ The matrix of a linear transformation wrt the **standard basis**:

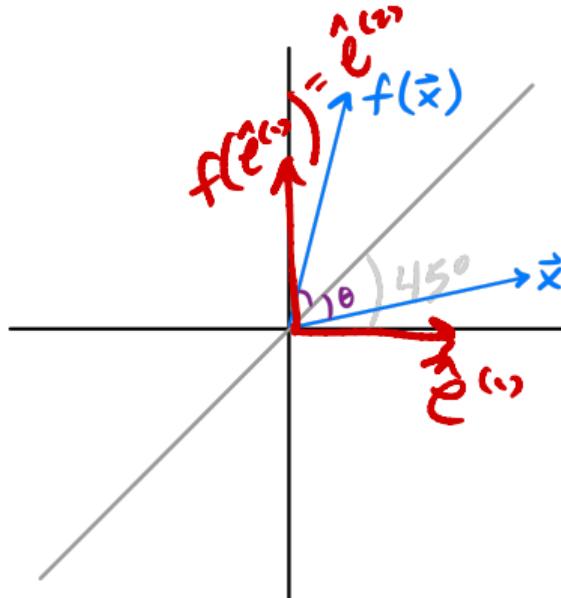
$$\begin{pmatrix} \uparrow & \uparrow & \uparrow & \\ \vec{f}(\hat{e}^{(1)}) & \vec{f}(\hat{e}^{(2)}) & \dots & \vec{f}(\hat{e}^{(d)}) \\ \downarrow & \downarrow & \downarrow & \end{pmatrix}$$

- ▶ With respect to basis \mathcal{U} :

$$\begin{pmatrix} \uparrow & \uparrow & \uparrow & \\ [\vec{f}(\hat{u}^{(1)})]_{\mathcal{U}} & [\vec{f}(\hat{u}^{(2)})]_{\mathcal{U}} & \dots & [\vec{f}(\hat{u}^{(d)})]_{\mathcal{U}} \\ \downarrow & \downarrow & \downarrow & \end{pmatrix}$$

Matrices in Other Bases

- ▶ Consider the transformation \vec{f} which “mirrors” a vector over the line of 45° .



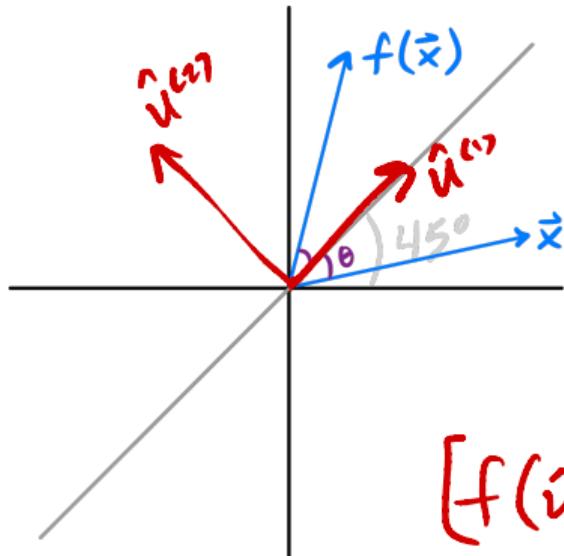
$$f(\hat{e}^{(1)}) = \hat{e}^{(2)}$$

$$f(\hat{e}^{(2)}) = \hat{e}^{(1)}$$

- ▶ What is its matrix in the standard basis?

$$\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$$

Matrices in Other Bases



$$f(\hat{u}^{(1)}) = \hat{u}^{(1)}$$

$$f(\hat{u}^{(2)}) = -\hat{u}^{(2)}$$

► Let $\hat{u}^{(1)} = \frac{1}{\sqrt{2}}(1, 1)^T$

► Let $\hat{u}^{(2)} = \frac{1}{\sqrt{2}}(-1, 1)^T$

► What is $[f(\hat{u}^{(1)})]_U$?

► $[f(\hat{u}^{(2)})]_U$?

► What is the matrix?

$$[f(\hat{u}^{(1)})]_U = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$

$$[f(\hat{u}^{(2)})]_U = \begin{pmatrix} 0 \\ -1 \end{pmatrix}$$

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

DSC 140B

Representation Learning

Lecture 03 | Part 3

The Spectral Theorem

Eigenvectors

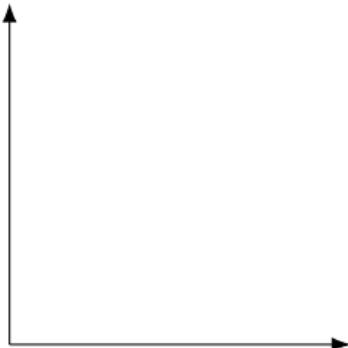
- ▶ Let A be an $n \times n$ matrix. An **eigenvector** of A with **eigenvalue** λ is a nonzero vector \vec{v} such that $A\vec{v} = \lambda\vec{v}$.

Eigenvectors (of Linear Transformations)

- ▶ Let \vec{f} be a linear transformation. An **eigenvector** of \vec{f} with **eigenvalue** λ is a nonzero vector \vec{v} such that $f(\vec{v}) = \lambda\vec{v}$.

Geometric Interpretation

- ▶ When \vec{f} is applied to one of its eigenvectors, \vec{f} simply scales it.
 - ▶ Possibly by a negative amount.



Symmetric Matrices

- Recall: a matrix A is **symmetric** if $A^T = A$.

The Spectral Theorem³

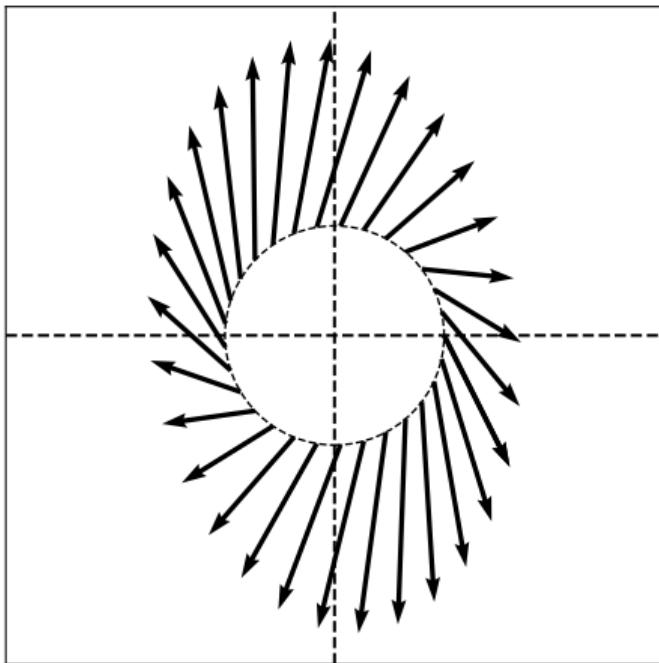
- ▶ **Theorem:** Let A be an $n \times n$ symmetric matrix. Then there exist n eigenvectors of A which are all mutually orthogonal.

³for symmetric matrices

What?

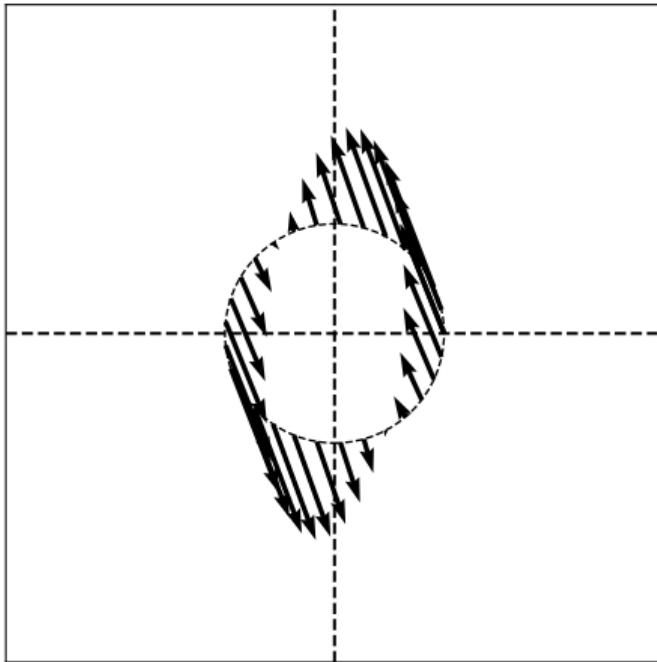
- ▶ What does the spectral theorem mean?
- ▶ What is an eigenvector, really?
- ▶ Why are they useful?

Example Linear Transformation



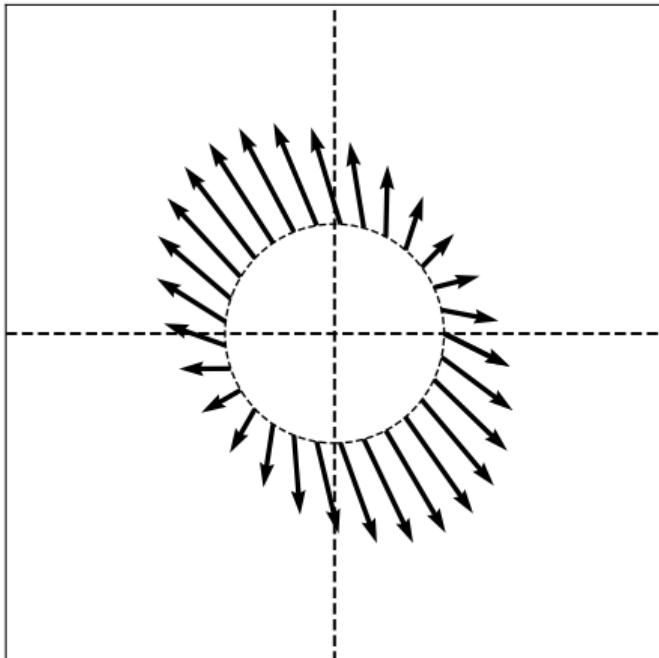
$$A = \begin{pmatrix} 5 & 5 \\ -10 & 12 \end{pmatrix}$$

Example Linear Transformation



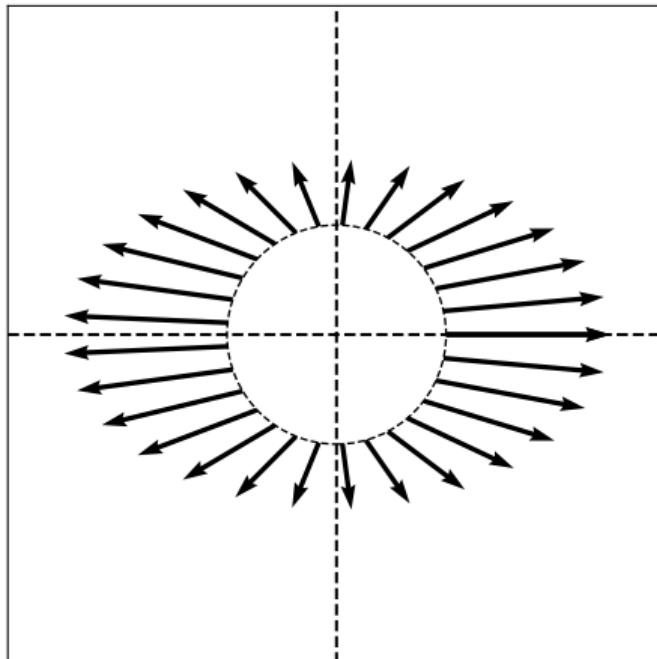
$$A = \begin{pmatrix} -2 & -1 \\ -5 & 3 \end{pmatrix}$$

Example Symmetric Linear Transformation



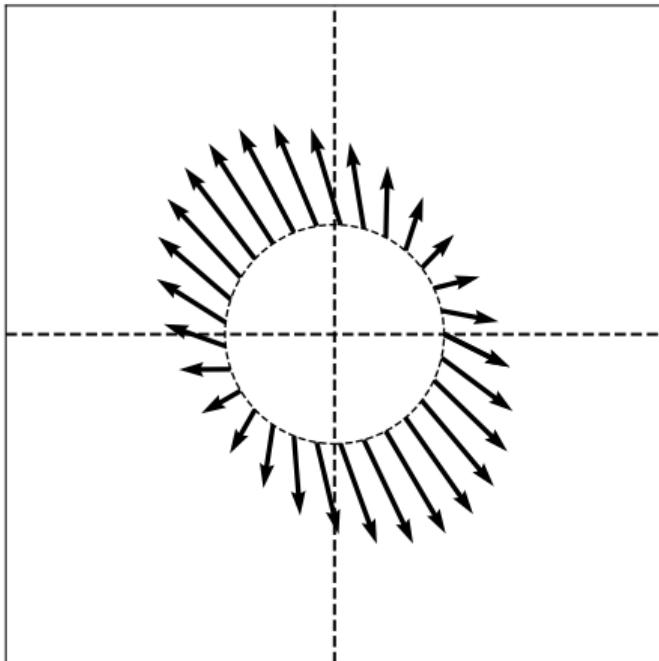
$$A = \begin{pmatrix} 2 & -1 \\ -1 & 3 \end{pmatrix}$$

Example Symmetric Linear Transformation



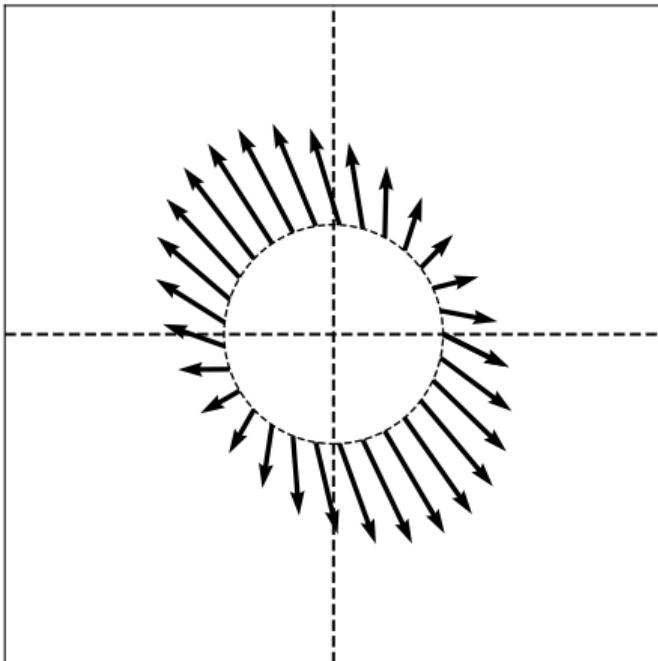
$$A = \begin{pmatrix} 5 & 0 \\ 0 & 2 \end{pmatrix}$$

Observation #1



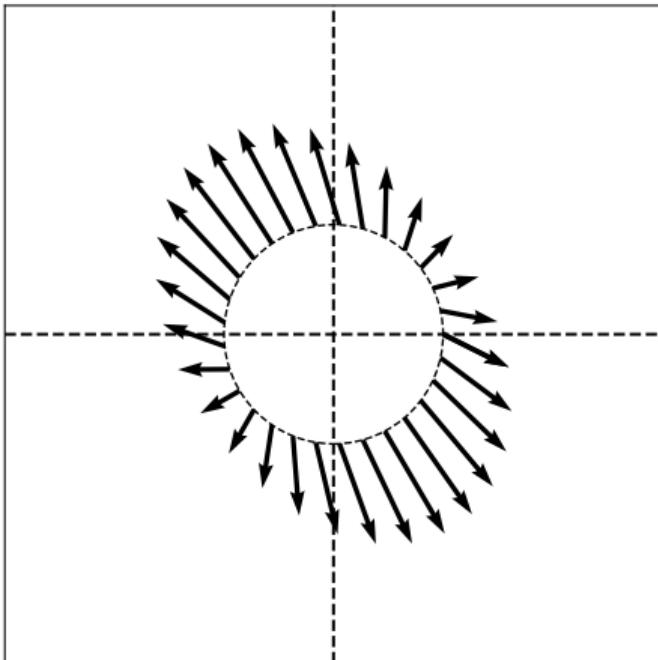
- ▶ Symmetric linear transformations have **axes of symmetry**.

Observation #2



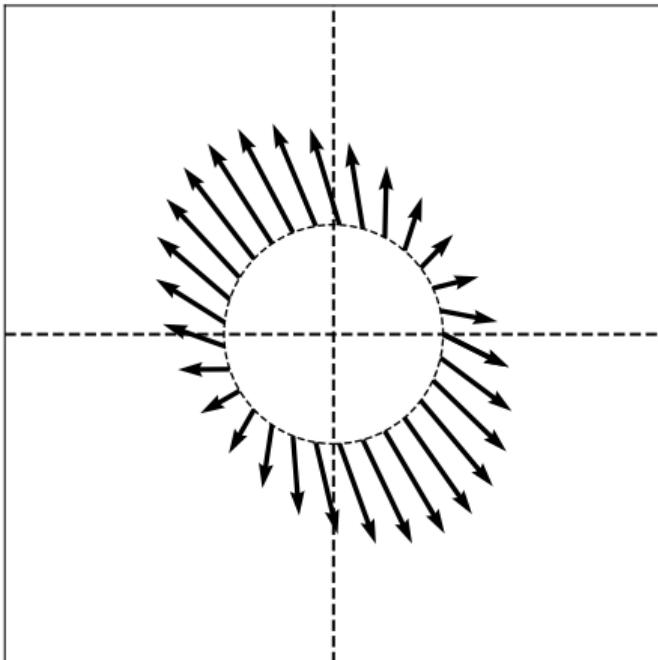
- ▶ The axes of symmetry are **orthogonal** to one another.

Observation #3



- ▶ The action of \vec{f} along an axis of symmetry is simply to scale its input.

Observation #4



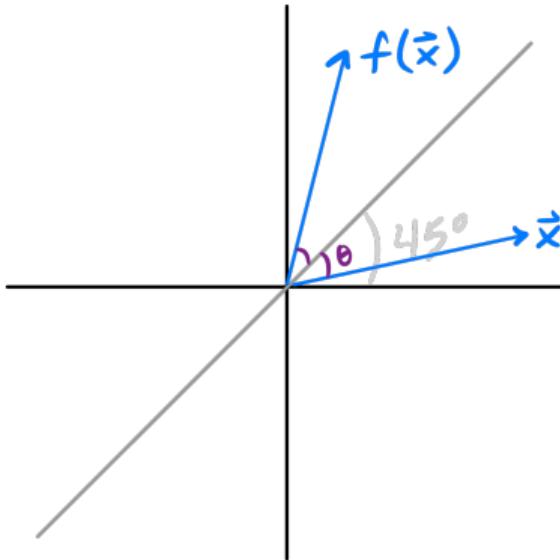
- ▶ The size of this scaling can be different for each axis.

Main Idea

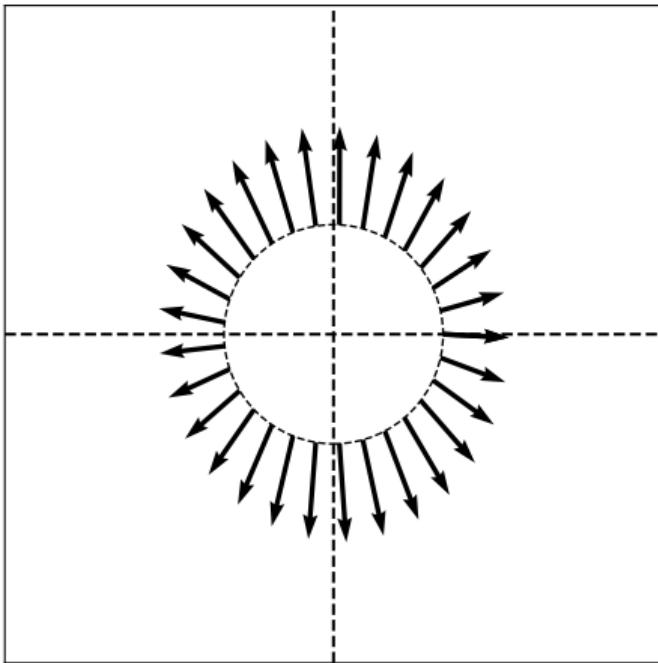
The **eigenvectors** of a symmetric linear transformation (matrix) are its axes of symmetry. The **eigenvalues** describe how much each axis of symmetry is scaled.

Exercise

Consider the linear transformation which mirrors its input over the line of 45° . Give two orthogonal eigenvectors of the transformation.

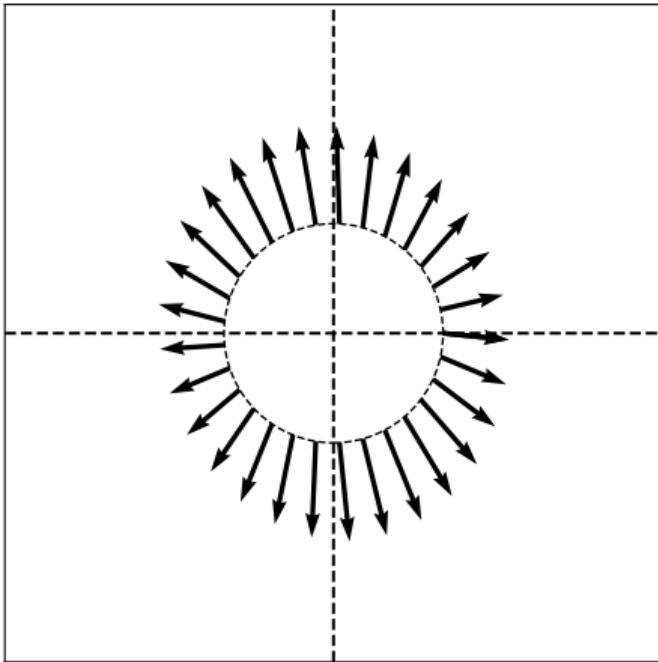


Off-diagonal elements



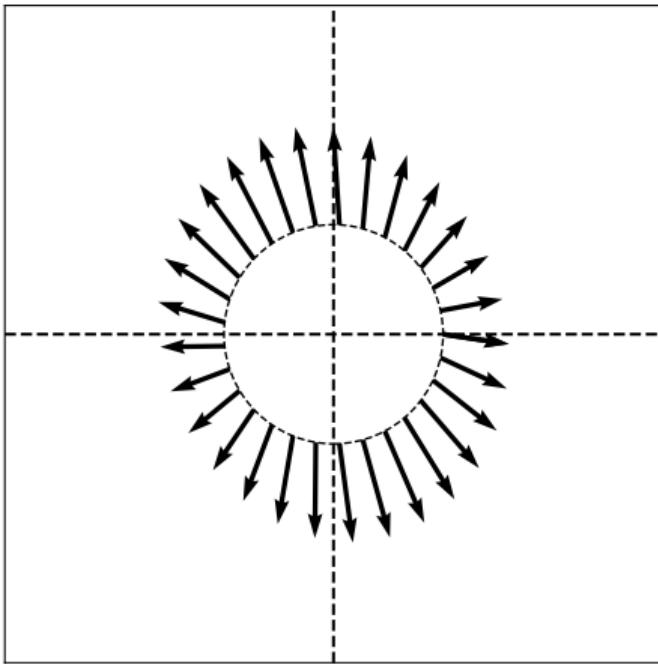
$$A = \begin{pmatrix} 5 & -0.1 \\ -0.1 & 2 \end{pmatrix}$$

Off-diagonal elements



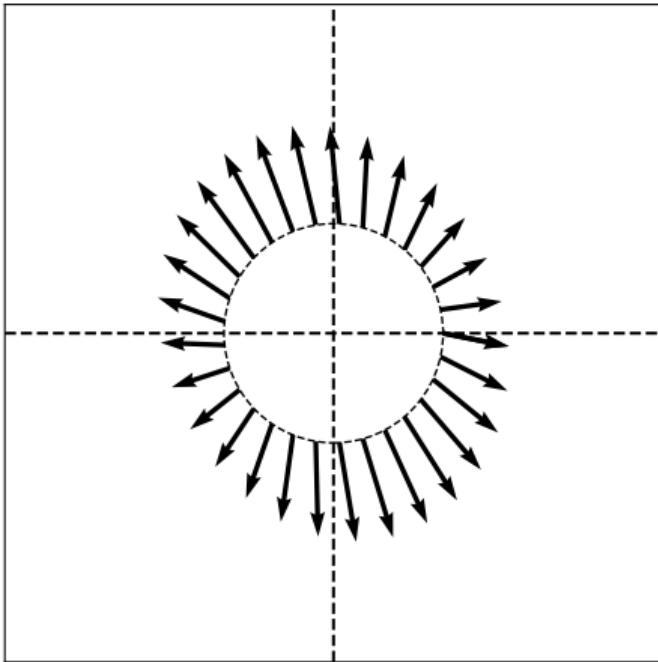
$$A = \begin{pmatrix} 5 & -0.2 \\ -0.2 & 2 \end{pmatrix}$$

Off-diagonal elements



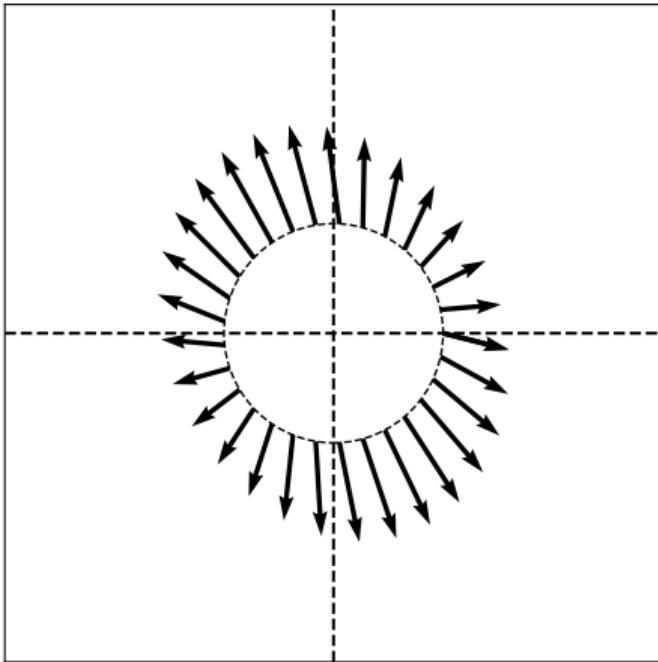
$$A = \begin{pmatrix} 5 & -0.3 \\ -0.3 & 2 \end{pmatrix}$$

Off-diagonal elements



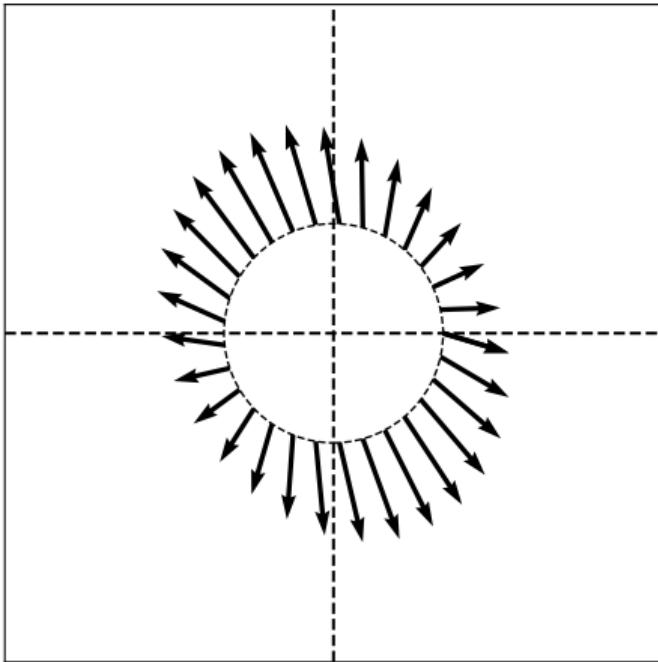
$$A = \begin{pmatrix} 5 & -0.4 \\ -0.4 & 2 \end{pmatrix}$$

Off-diagonal elements



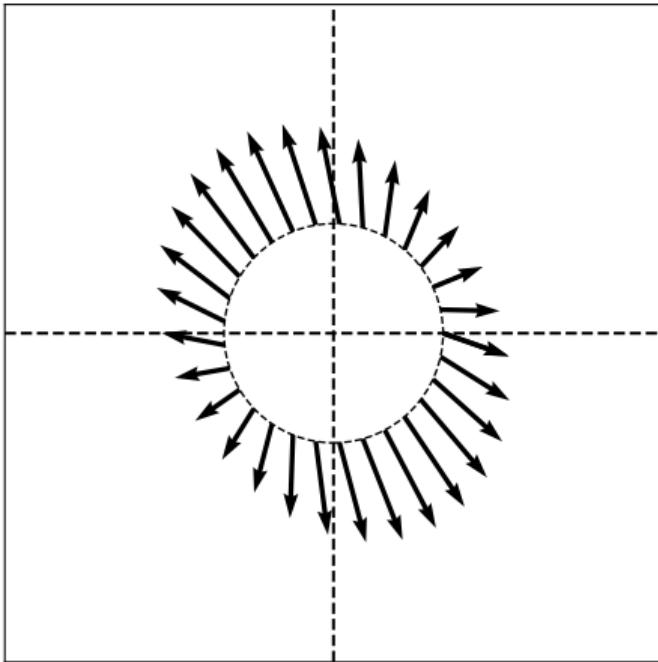
$$A = \begin{pmatrix} 5 & -0.5 \\ -0.5 & 2 \end{pmatrix}$$

Off-diagonal elements



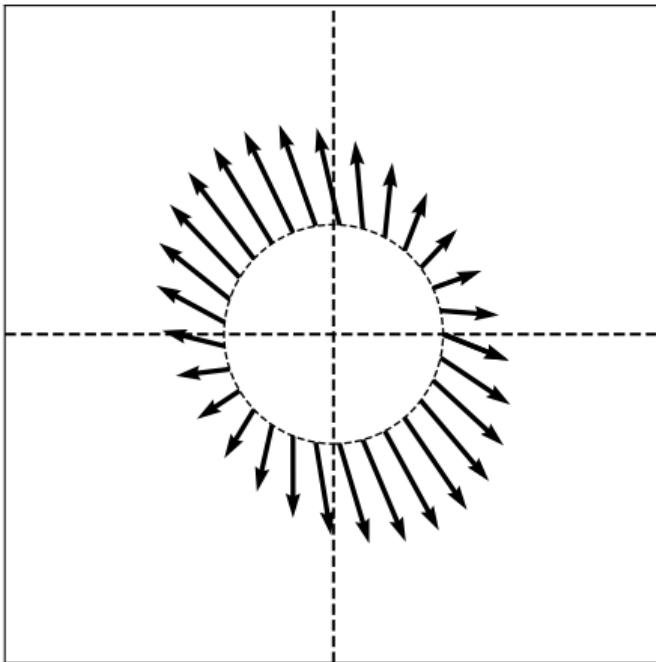
$$A = \begin{pmatrix} 5 & -0.6 \\ -0.6 & 2 \end{pmatrix}$$

Off-diagonal elements



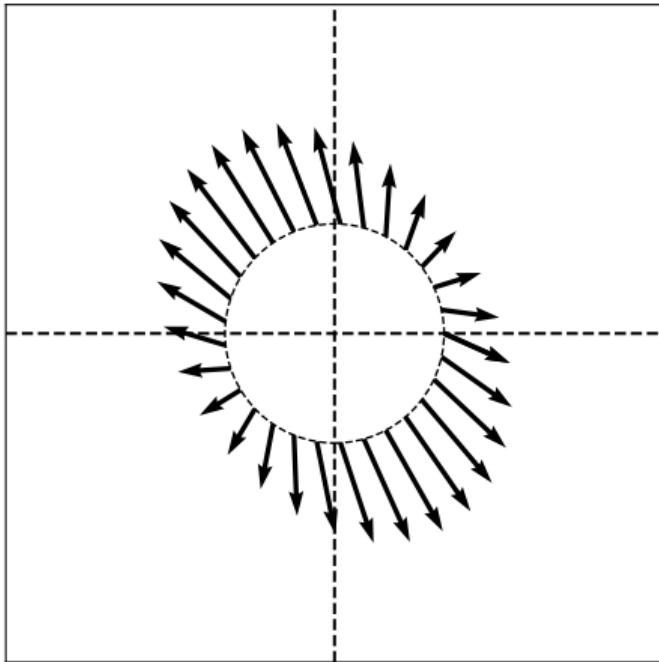
$$A = \begin{pmatrix} 5 & -0.7 \\ -0.7 & 2 \end{pmatrix}$$

Off-diagonal elements



$$A = \begin{pmatrix} 5 & -0.8 \\ -0.8 & 2 \end{pmatrix}$$

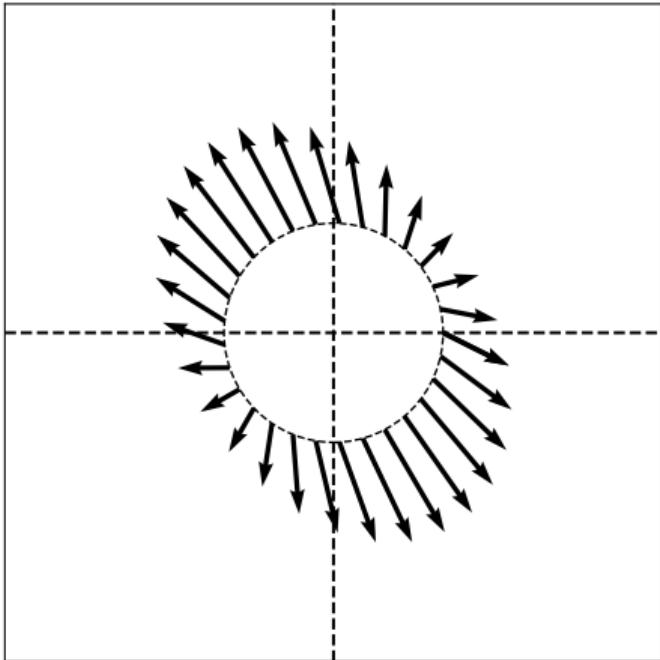
Off-diagonal elements



$$A = \begin{pmatrix} 5 & -0.9 \\ -0.9 & 2 \end{pmatrix}$$

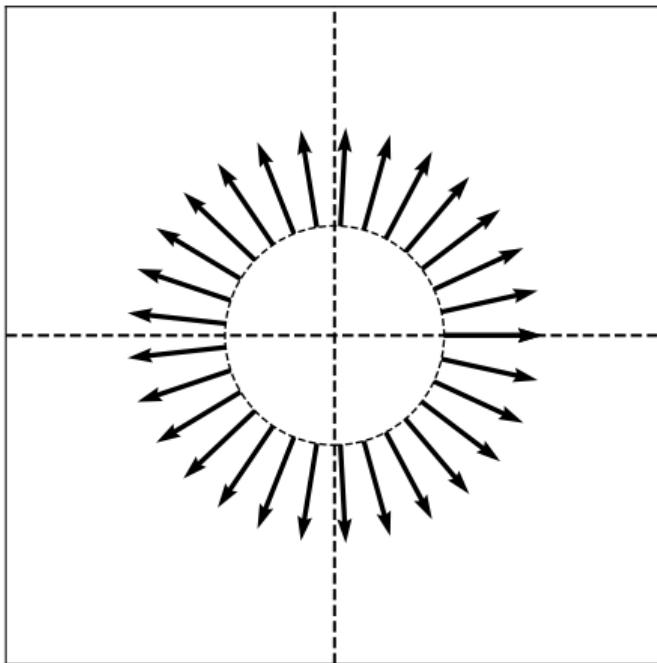
The Spectral Theorem⁴

- ▶ **Theorem:** Let A be an $n \times n$ symmetric matrix. Then there exist n eigenvectors of A which are all mutually orthogonal.



⁴for symmetric matrices

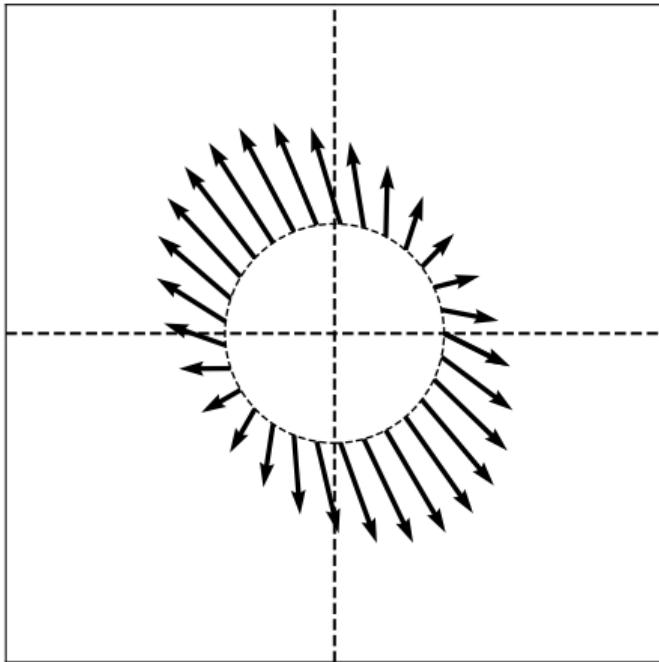
What about total symmetry?



- ▶ Every vector is an eigenvector.

$$A = \begin{pmatrix} 3 & 0 \\ 0 & 3 \end{pmatrix}$$

Computing Eigenvectors



```
>>> A = np.array([[2, -1], [-1, 3]])  
>>> np.linalg.eigh(A)  
(array([1.38196601, 3.61803399]),  
 array([[-0.85065081, -0.52573111],  
       [-0.52573111, 0.85065081]]))
```

DSC 140B

Representation Learning

Lecture 04 | Part 1

The Spectral Theorem

Eigenvectors

- ▶ Let A be an $n \times n$ matrix. An **eigenvector** of A with **eigenvalue** λ is a nonzero vector \vec{v} such that $A\vec{v} = \lambda\vec{v}$.

Eigenvectors (of Linear Transformations)

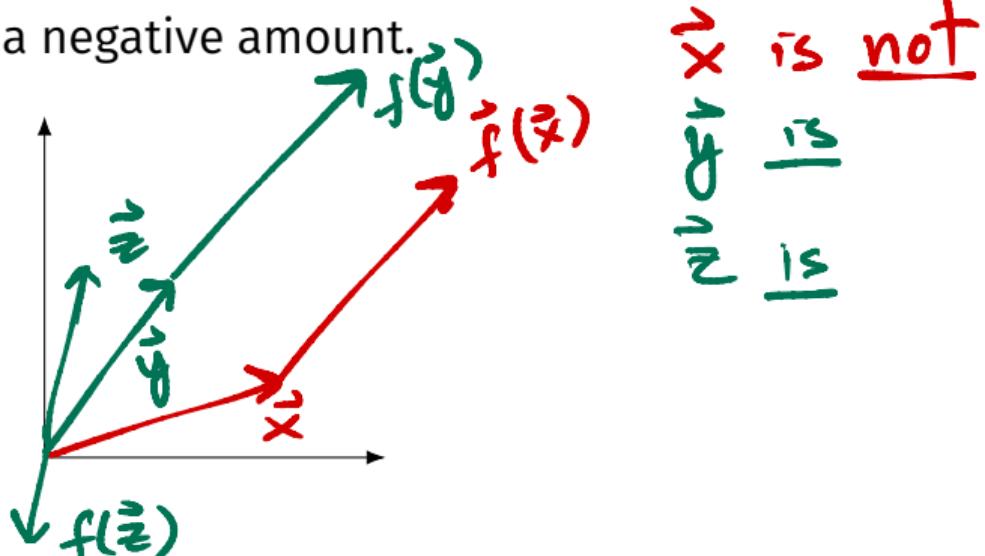
- ▶ Let \vec{f} be a linear transformation. An **eigenvector** of \vec{f} with **eigenvalue** λ is a nonzero vector \vec{v} such that $f(\vec{v}) = \lambda\vec{v}$.

Importance

- ▶ We will see why eigenvectors are important in the next part.
- ▶ For now: what are they?

Geometric Interpretation

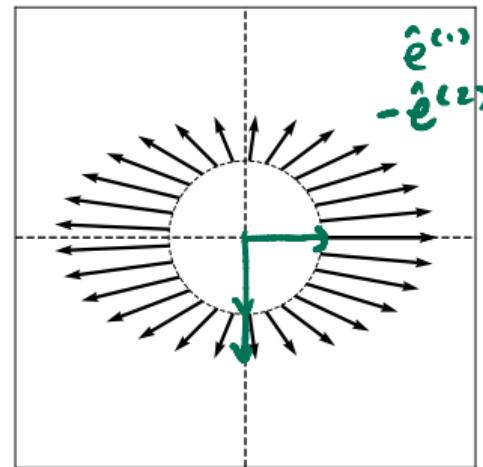
- ▶ When \vec{f} is applied to one of its eigenvectors, \vec{f} simply scales it.
 - ▶ Possibly by a negative amount.



Exercise

Draw as many (linearly independent) eigenvectors as you can:

$$A\hat{e}^{(1)} = 5\hat{e}^{(1)}$$
$$\vec{x} \propto \vec{x}$$
$$A(3\hat{e}^{(2)}) = 3A\hat{e}^{(2)} = 3(5\hat{e}^{(2)})$$
$$A = \begin{pmatrix} 5 & 0 \\ 0 & 2 \end{pmatrix} = 5(3\hat{e}^{(2)})$$
$$A\vec{y} = 5\vec{y}$$



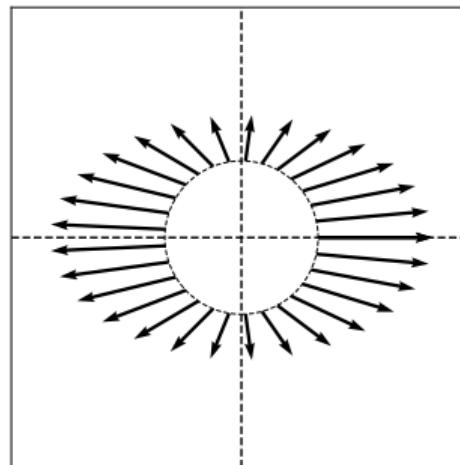
Exercise

Draw as many (linearly independent) eigenvectors as you can:

$\hat{e}^{(1)}$ w/ e-value: 5

$\hat{e}^{(2)}$ w/ e-value: 2

$$A = \begin{pmatrix} 5 & 0 \\ 0 & 2 \end{pmatrix}$$



Finding Eigenvectors

- ▶ We typically compute the eigenvectors of a matrix with a computer.
- ▶ But it can help our understanding to find them “graphically”.

Procedure

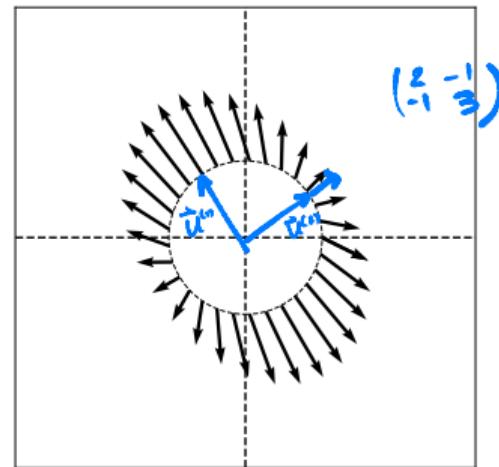
Given a matrix A (or transformation \vec{f}), to find an eigenvector “graphically”.

1. Think about (or draw) the output of \vec{f} for a handful of unit vector inputs.
 - ▶ Linear transformations are continuous so you can “interpolate”.
2. Find place(s) where the input vector and the output vector are parallel.

Exercise

Draw as many (linearly independent) eigenvectors as you can:

$$A = \begin{pmatrix} 2 & -1 \\ -1 & 3 \end{pmatrix}$$

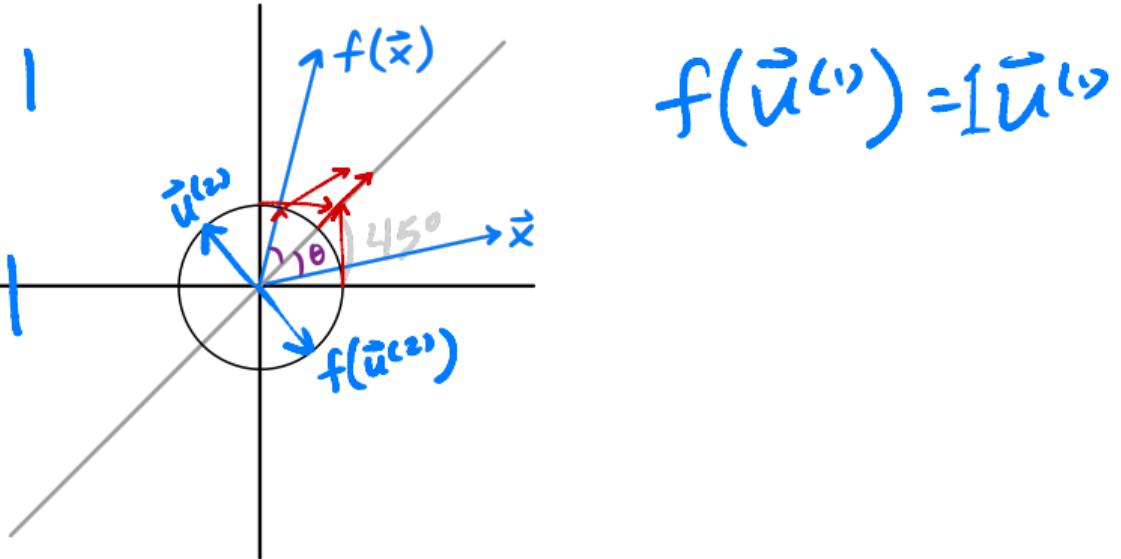


Exercise

Consider the linear transformation which mirrors its input over the line of 45° . Give two orthogonal eigenvectors of the transformation.

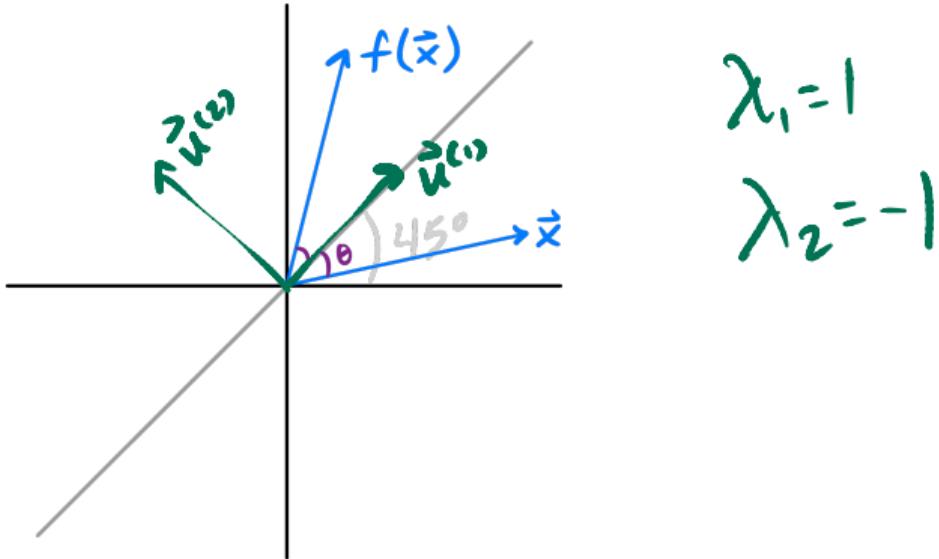
$$\vec{u}^{(1)} = \begin{pmatrix} 1 \\ 1 \end{pmatrix} \quad \lambda_1 = 1$$

$$\vec{u}^{(2)} = \begin{pmatrix} -1 \\ 1 \end{pmatrix} \quad \lambda_2 = -1$$



Exercise

Consider the linear transformation which mirrors its input over the line of 45° . Give two orthogonal eigenvectors of the transformation.



Alternate Procedure: Guess and Check

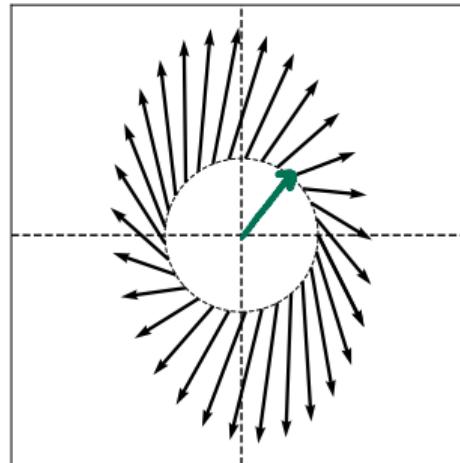
1. Guess a vector \vec{x} .
2. Check that $\vec{f}(\vec{x}) = \lambda\vec{x}$.

Exercise

Draw as many (linearly independent) eigenvectors as you can:

None

$$A = \begin{pmatrix} 5 & 5 \\ -10 & 12 \end{pmatrix}$$



Caution!

- ▶ Not all matrices have even one eigenvector!¹
- ▶ When does a matrix have multiple (linearly independent) eigenvectors?

¹That is, with a *real-valued* eigenvalue.

$$\begin{pmatrix} 5 & -3 \\ 7 & 2 \end{pmatrix}$$

↑
not
is

Symmetric Matrices

- Recall: a matrix A is **symmetric** if $A^T = A$.

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix}^T = \begin{pmatrix} a & c \\ b & d \end{pmatrix}$$

$$\begin{pmatrix} 5 & -3 \\ -3 & 2 \end{pmatrix}$$

The Spectral Theorem²

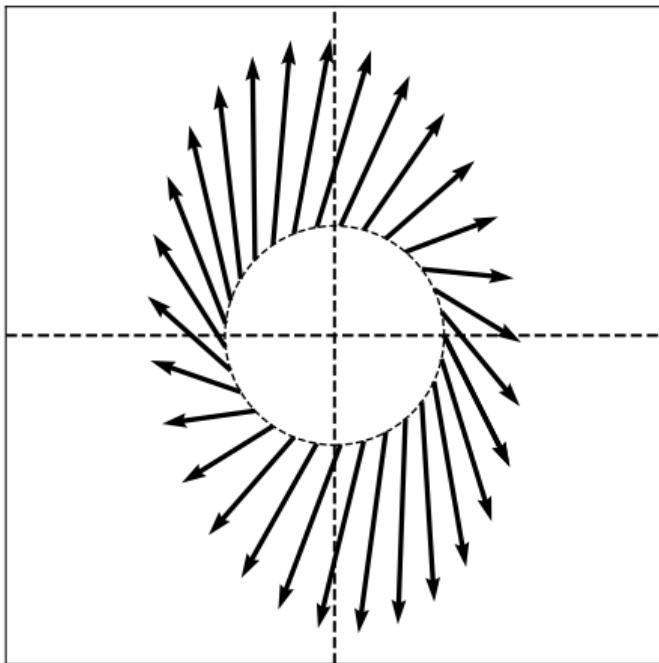
- ▶ **Theorem:** Let A be an $n \times n$ symmetric matrix. Then there exist n eigenvectors of A which are all mutually orthogonal.

²for symmetric matrices

What?

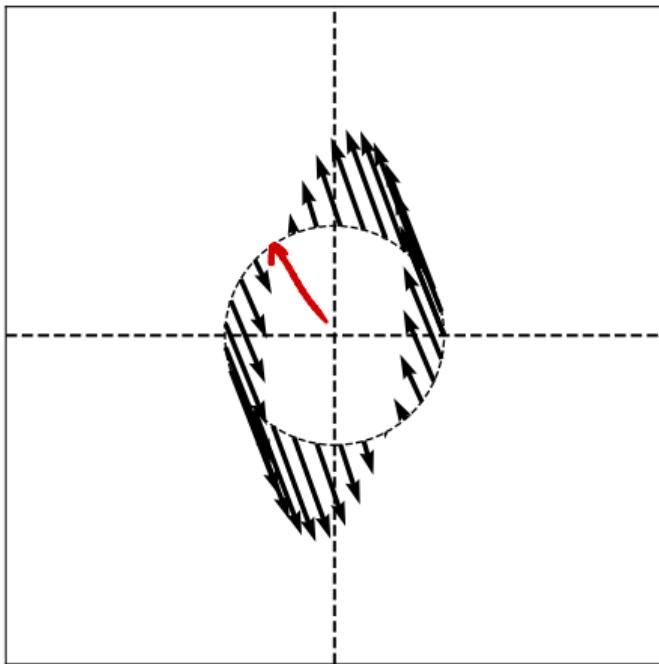
- ▶ What does the spectral theorem mean?
- ▶ What is an eigenvector, really?
- ▶ Why are they useful?

Example Linear Transformation



$$A = \begin{pmatrix} 5 & 5 \\ -10 & 12 \end{pmatrix}$$

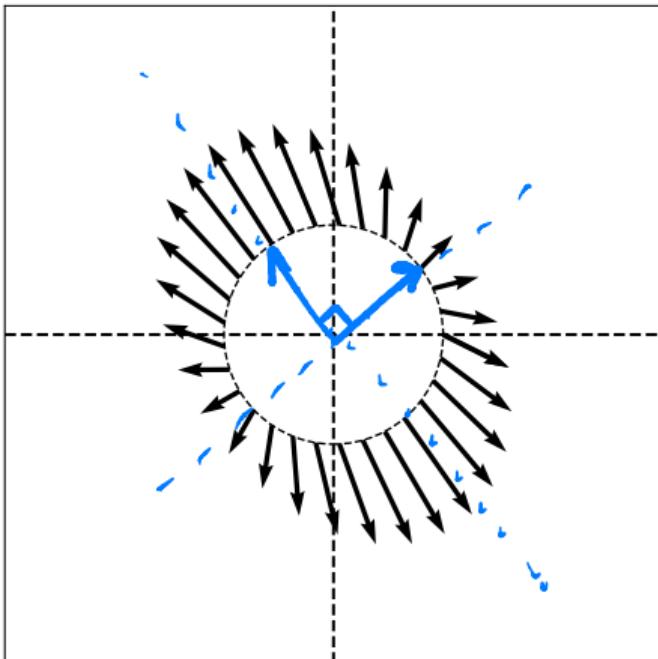
Example Linear Transformation



$$A\vec{x} = \vec{0}$$

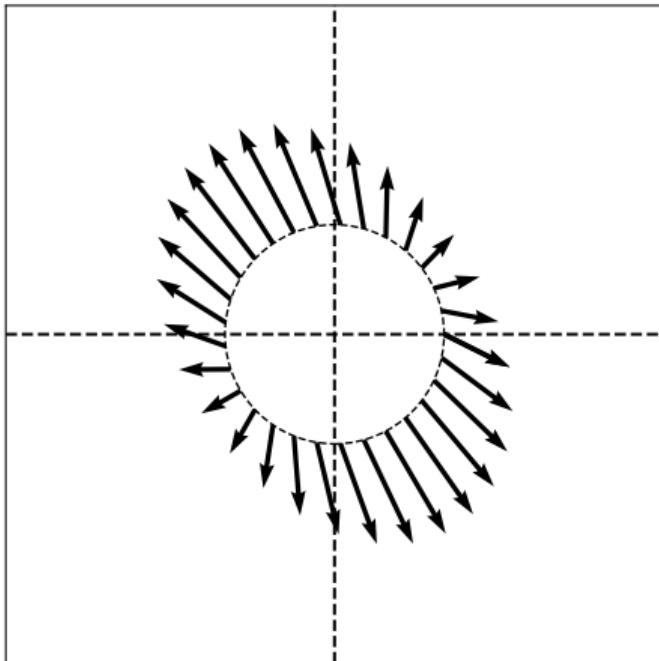
$$A = \begin{pmatrix} -2 & -1 \\ -5 & 3 \end{pmatrix}$$

Example Symmetric Linear Transformation



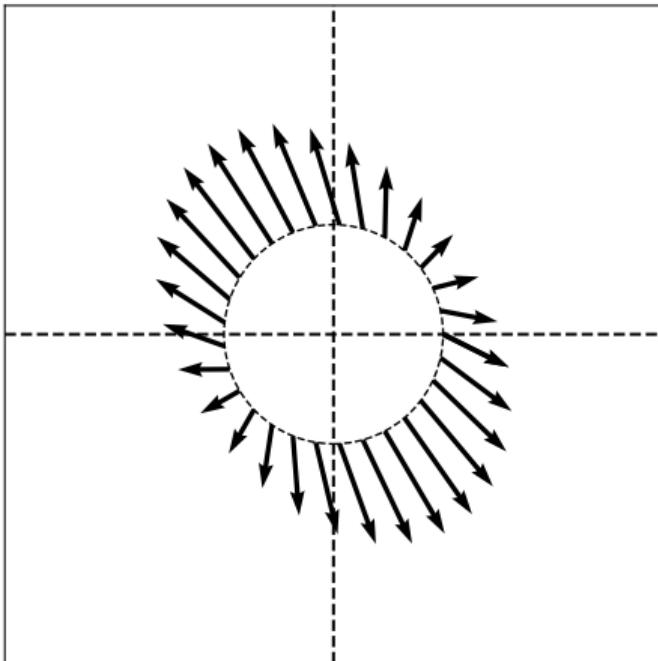
$$A = \begin{pmatrix} 2 & -1 \\ -1 & 3 \end{pmatrix}$$

Observation #1



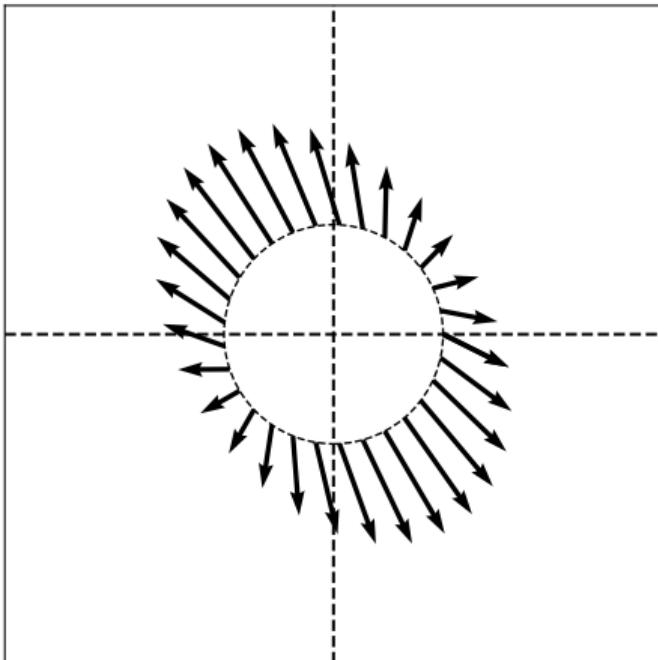
- ▶ Symmetric linear transformations have **axes of symmetry**.

Observation #2



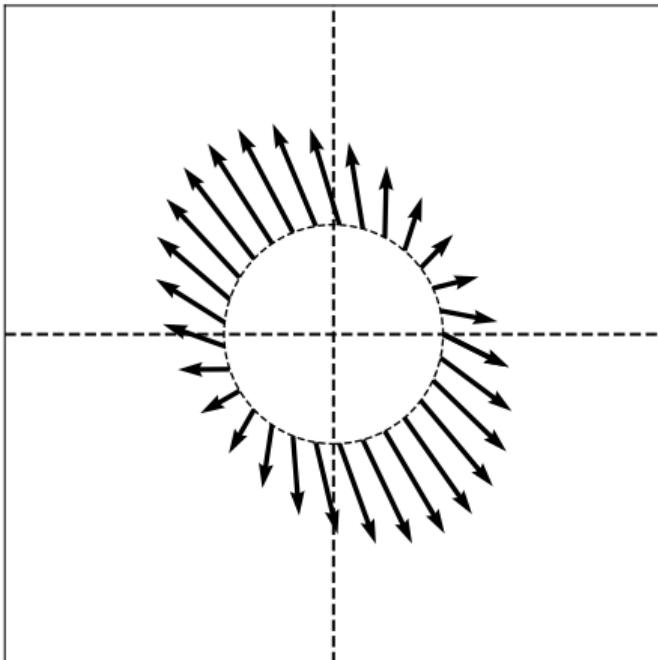
- ▶ The axes of symmetry are **orthogonal** to one another.

Observation #3



- ▶ The action of \vec{f} along an axis of symmetry is simply to scale its input.

Observation #4



- ▶ The size of this scaling can be different for each axis.

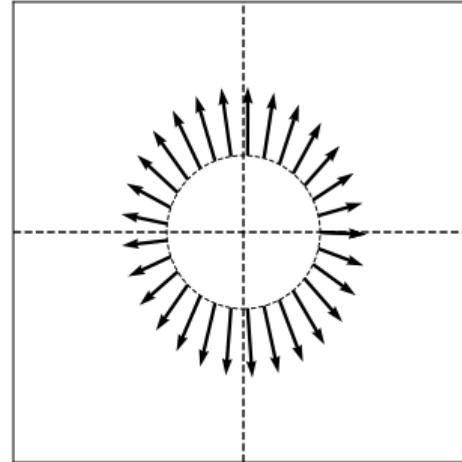
Main Idea

The **eigenvectors** of a symmetric linear transformation (matrix) are its axes of symmetry. The **eigenvalues** describe how much each axis of symmetry is scaled.

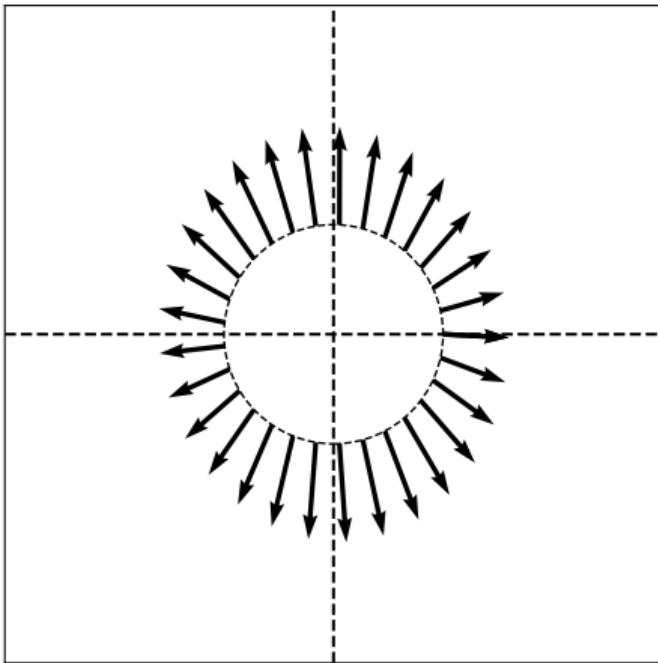
Diagonal Matrices

- ▶ If A is diagonal, its eigenvectors are simply the standard basis vectors.

$$A = \begin{pmatrix} 2 & 0 \\ 0 & 5 \end{pmatrix}$$

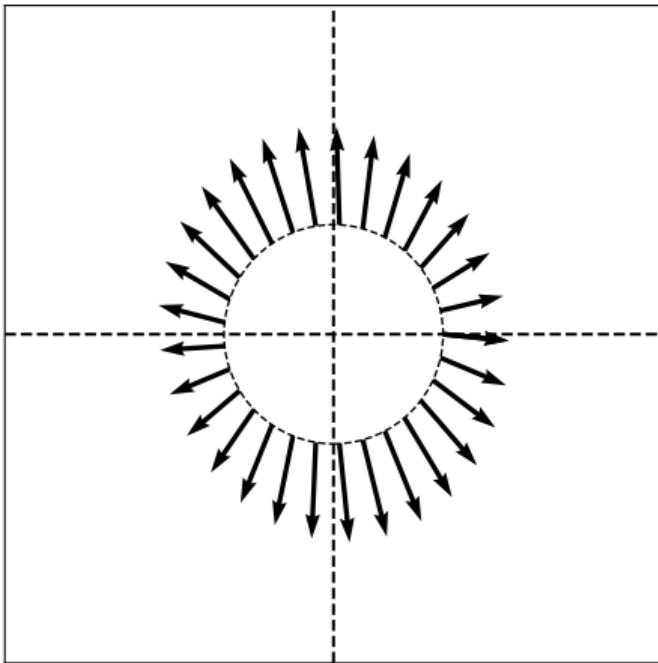


Off-diagonal elements



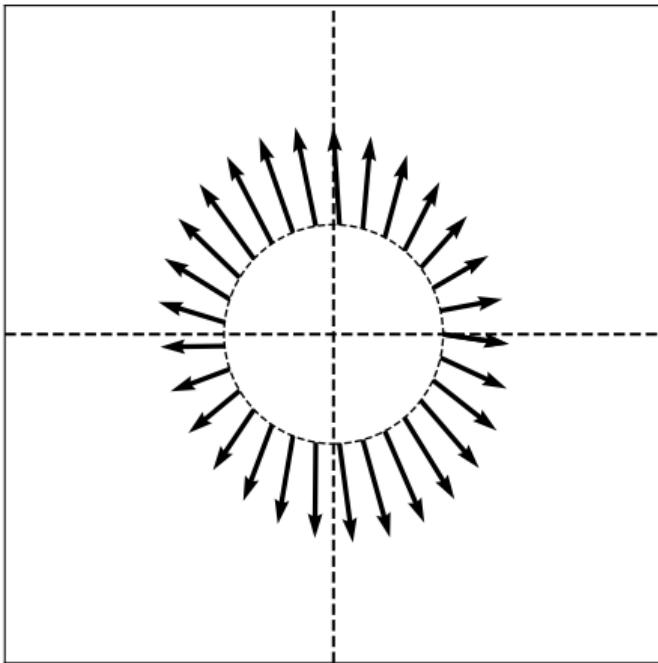
$$A = \begin{pmatrix} 2 & -0.1 \\ -0.1 & 5 \end{pmatrix}$$

Off-diagonal elements



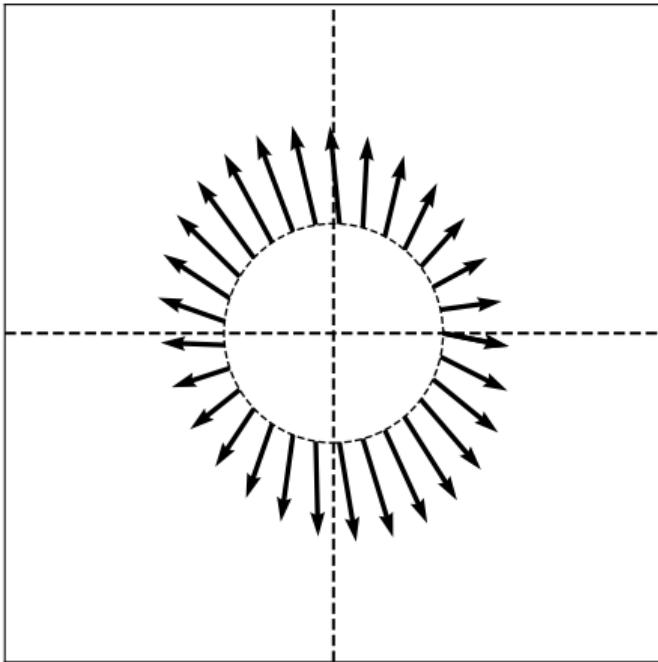
$$A = \begin{pmatrix} 2 & -0.2 \\ -0.2 & 5 \end{pmatrix}$$

Off-diagonal elements



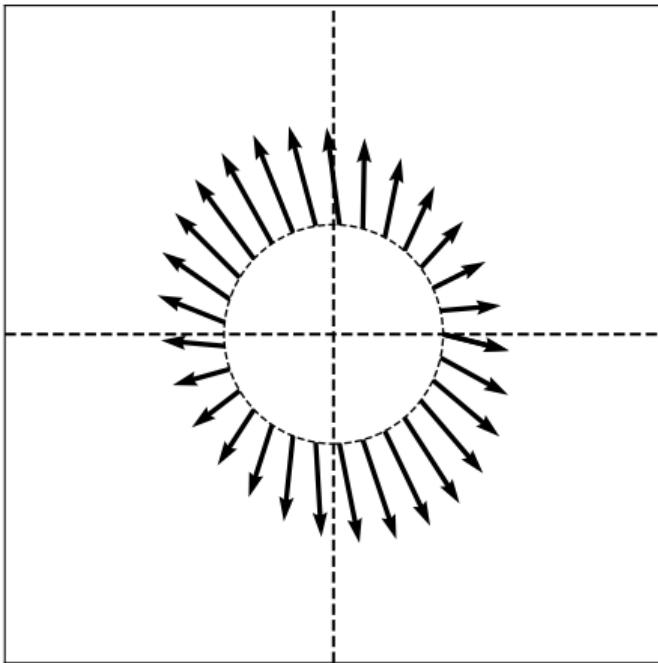
$$A = \begin{pmatrix} 2 & -0.3 \\ -0.3 & 5 \end{pmatrix}$$

Off-diagonal elements



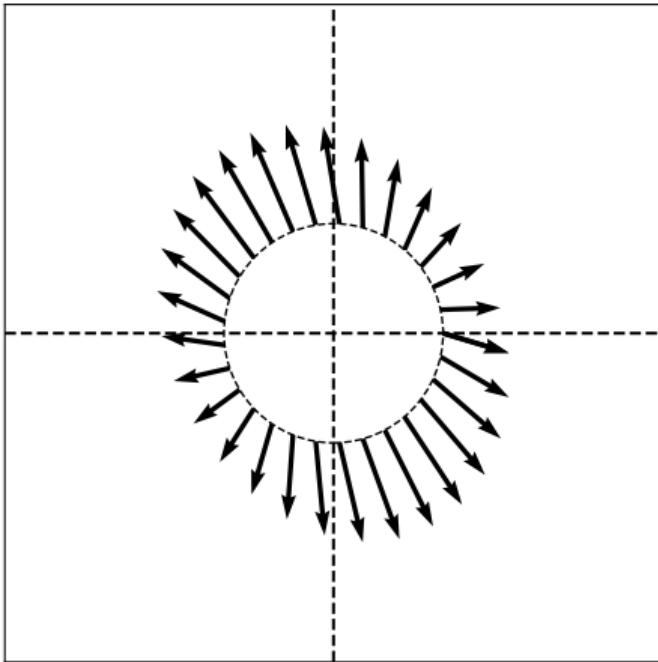
$$A = \begin{pmatrix} 2 & -0.4 \\ -0.4 & 5 \end{pmatrix}$$

Off-diagonal elements



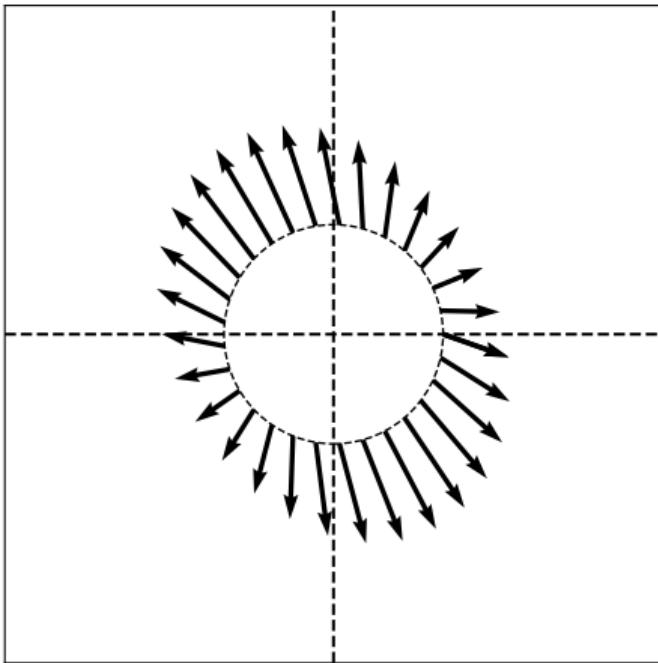
$$A = \begin{pmatrix} 2 & -0.5 \\ -0.5 & 5 \end{pmatrix}$$

Off-diagonal elements



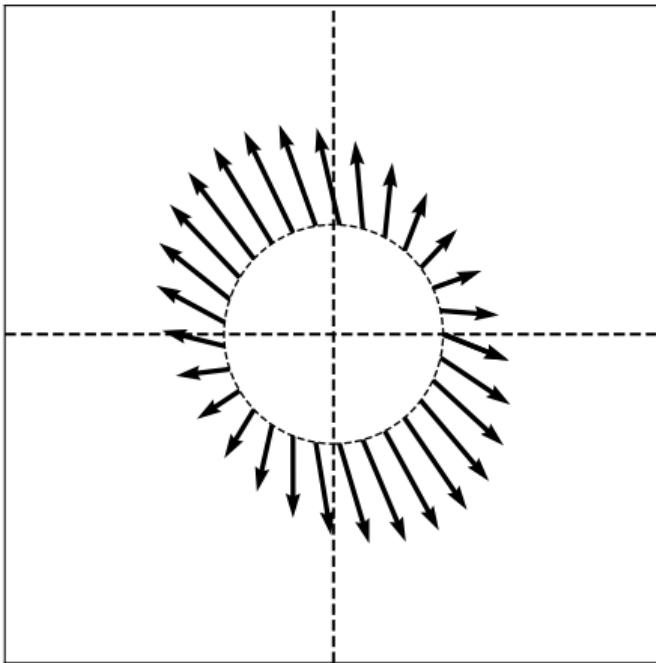
$$A = \begin{pmatrix} 2 & -0.6 \\ -0.6 & 5 \end{pmatrix}$$

Off-diagonal elements



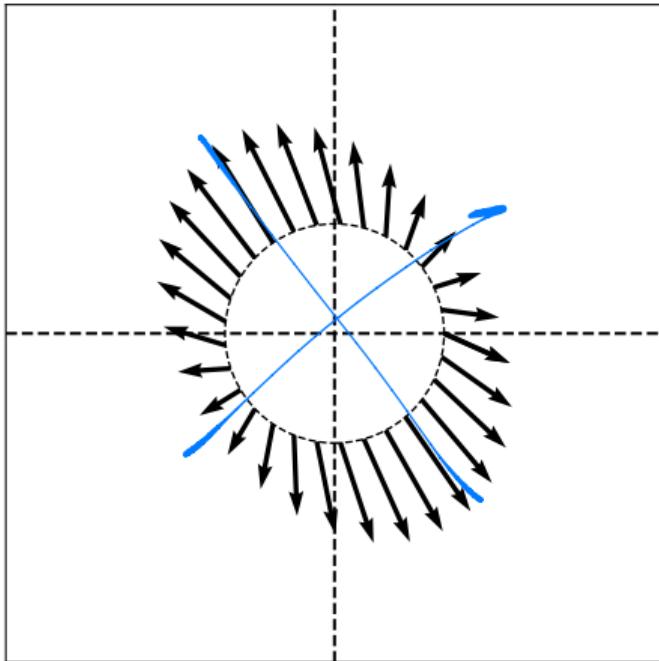
$$A = \begin{pmatrix} 2 & -0.7 \\ -0.7 & 5 \end{pmatrix}$$

Off-diagonal elements



$$A = \begin{pmatrix} 2 & -0.8 \\ -0.8 & 5 \end{pmatrix}$$

Off-diagonal elements



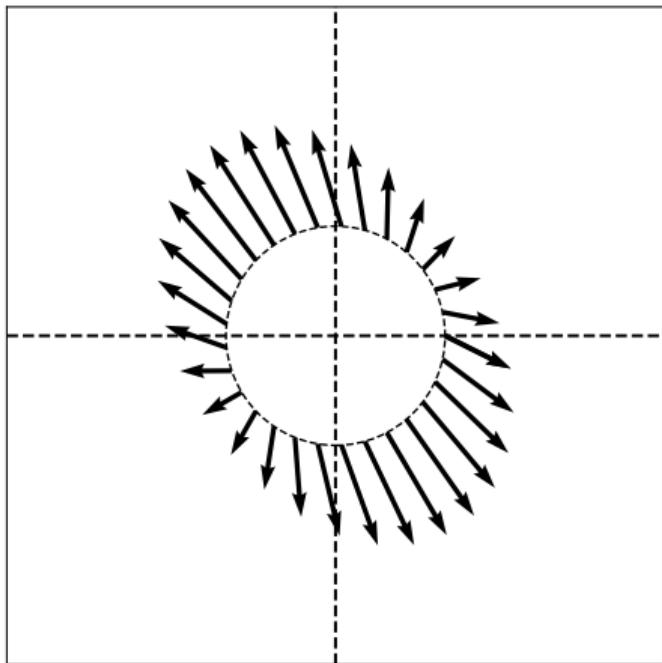
$$A = \begin{pmatrix} 2 & -0.9 \\ -0.9 & 5 \end{pmatrix}$$

Non-Diagonal Symmetric Matrices

- ▶ When a symmetric matrix is not diagonal, its eigenvectors are not the standard basis vectors.
- ▶ But they can be used to form an orthonormal basis!

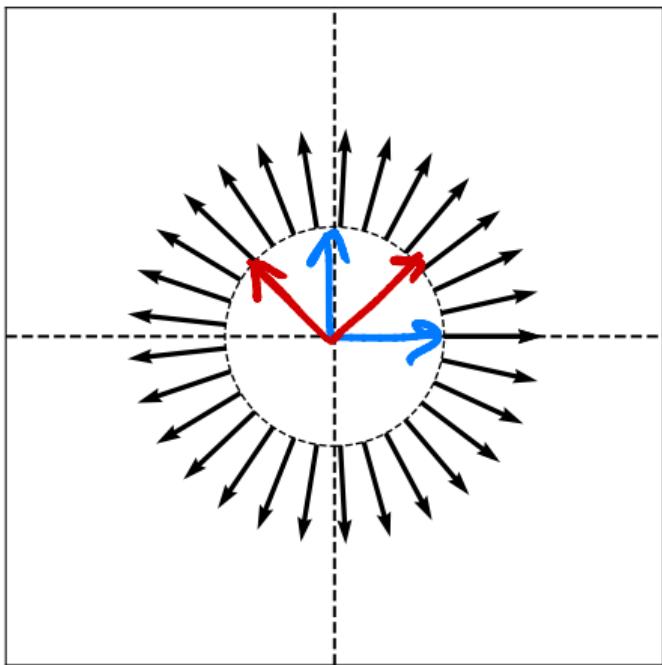
The Spectral Theorem³

- ▶ **Theorem:** Let A be an $n \times n$ symmetric matrix. Then there exist n eigenvectors of A which are all mutually orthogonal.



³for symmetric matrices

What about total symmetry?

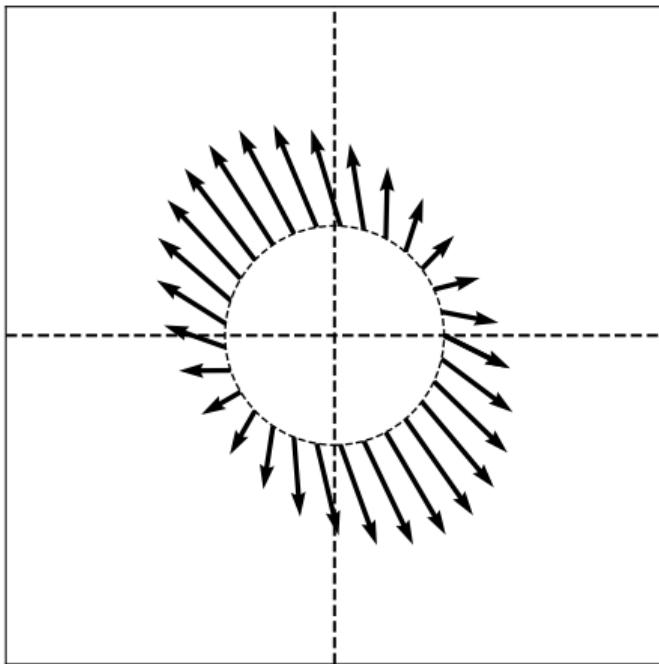


$$\begin{pmatrix} 3 & 0 \\ 0 & 3 \end{pmatrix} \begin{pmatrix} 2 \\ 1 \end{pmatrix} = \begin{pmatrix} 6 \\ 3 \end{pmatrix} = 3 \begin{pmatrix} 2 \\ 1 \end{pmatrix}$$

- ▶ Every vector is an eigenvector.

$$A = \begin{pmatrix} 3 & 0 \\ 0 & 3 \end{pmatrix}$$

Computing Eigenvectors



~~eig()~~

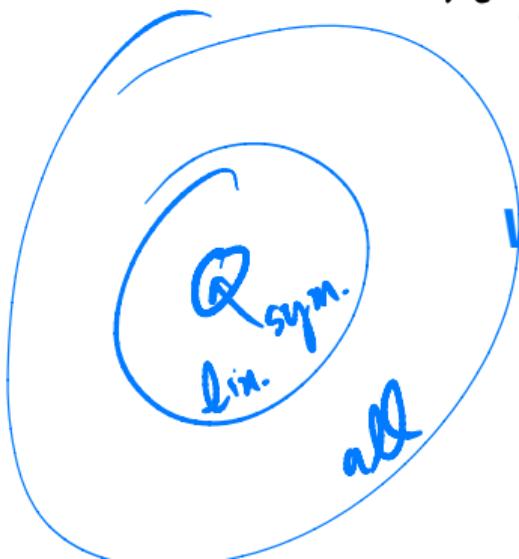
```
»> A = np.array([[2, -1], [-1, 3]])  
»> np.linalg.eigh(A)  
(array([1.38196601, 3.61803399]),  
 array([[[-0.85065081, -0.52573111],  
 [-0.52573111, 0.85065081]])))
```

DSC 140B

Representation Learning

Lecture 04 | Part 2

Why are eigenvectors useful?



OK, but why are eigenvectors⁴ useful?

1. Eigenvectors are nice “building blocks” (basis vectors).
2. Eigenvectors are **maximizers** (or minimizers).
3. Eigenvectors are **equilibria**.

⁴of symmetric matrices

Vector Decomposition

- ▶ We can always “decompose” a vector \vec{x} in terms of the basis vectors.
- ▶ With respect to the standard basis:

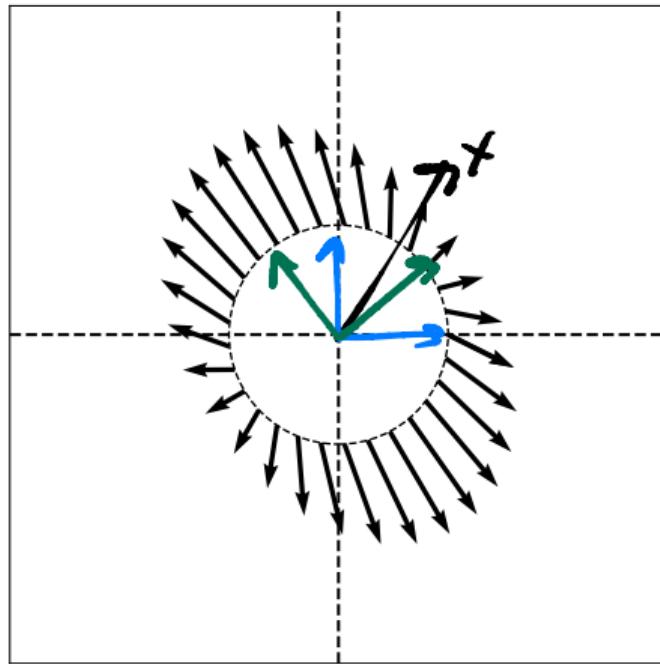
$$\vec{x} = a_1 \hat{e}^{(1)} + a_2 \hat{e}^{(2)} + \dots + a_d \hat{e}^{(d)}$$

Eigendecomposition

- ▶ If A is a symmetric matrix, we can pick d of its eigenvectors $\hat{u}^{(1)}, \dots, \hat{u}^{(d)}$ to form an orthonormal basis.
- ▶ Any vector \vec{x} can be written in terms of this basis.
- ▶ This is called its **eigendecomposition**:

$$\vec{x} = b_1 \hat{u}^{(1)} + b_2 \hat{u}^{(2)} + \dots + b_d \hat{u}^{(d)}$$

Eigendecomposition



Why?

$$A\hat{u}^{(i)} = \lambda_i \hat{u}^{(i)}$$

- ▶ Compare working in the standard basis decomposition:

$$\begin{aligned} A\vec{x} &= A(a_1\hat{e}^{(1)} + a_2\hat{e}^{(2)} + \dots + a_d\hat{e}^{(d)}) \\ &= a_1A\hat{e}^{(1)} + a_2A\hat{e}^{(2)} + \dots + a_dA\hat{e}^{(d)} \end{aligned}$$

- ▶ To work with the eigendecomposition:

$$\begin{aligned} A\vec{x} &= A(b_1\hat{u}^{(1)} + b_2\hat{u}^{(2)} + \dots + b_d\hat{u}^{(d)}) \\ &= b_1A\hat{u}^{(1)} + b_2A\hat{u}^{(2)} + \dots + b_dA\hat{u}^{(d)} \\ &= \lambda_1 b_1\hat{u}^{(1)} + \lambda_2 b_2\hat{u}^{(2)} + \dots + \lambda_d b_d\hat{u}^{(d)} \end{aligned}$$

Main Idea

If A is a symmetric matrix, an eigenbasis formed from its eigenvectors is an especially natural basis.

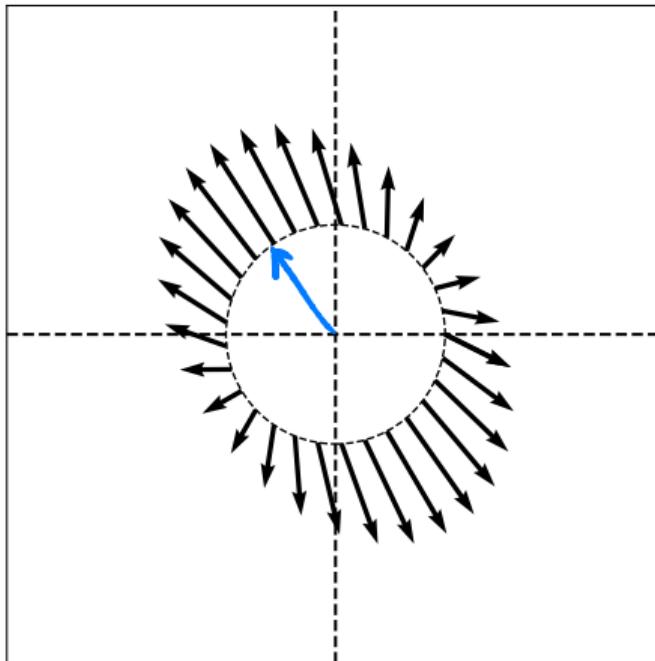
Eigenvectors as Optimizers

- ▶ Eigenvectors are the solutions to certain common optimization problems involving matrices / linear transformations.

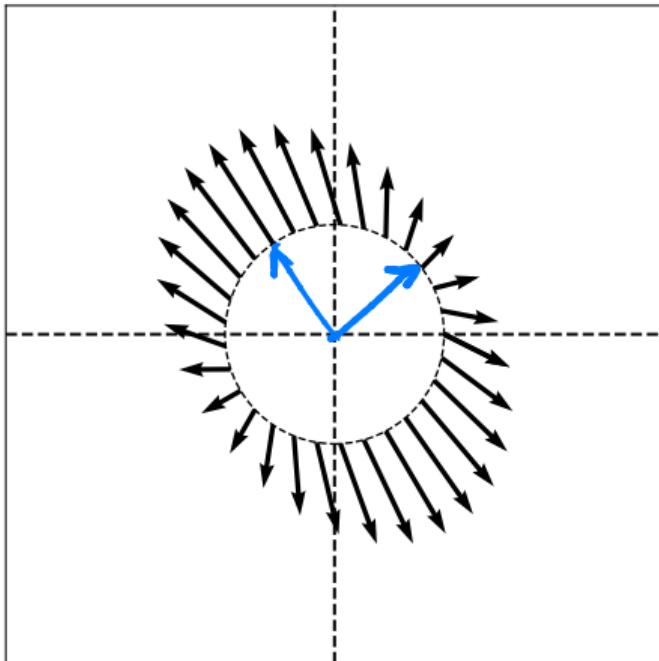
Exercise

Draw a unit vector \vec{x} such that $\|A\vec{x}\|$ is largest.

$$\begin{aligned} & \|A\vec{x}\| \\ & \|\vec{f}(\vec{x})\| \end{aligned}$$



Observation #1



- ▶ $\vec{f}(\vec{x})$ is longest along the “main” axis of symmetry.
 - ▶ In the direction of the eigenvector with largest eigenvalue.

Main Idea

To maximize $\|A\vec{x}\| = \|\vec{f}(\vec{x})\|$ over unit vectors, pick \vec{x} to be an eigenvector of \vec{f} with the largest eigenvalue (in abs. value).

Main Idea

To **minimize** $\|\vec{f}(\vec{x})\|$ over unit vectors, pick \vec{x} to be an eigenvector of \vec{f} with the smallest eigenvalue (in abs. value).

$$\|\vec{z}\|^2 = \vec{z} \cdot \vec{z}$$

Proof

Show that the maximizer of $\|A\vec{x}\|^2$ s.t., $\|\vec{x}\| = 1$ is the top eigenvector of A.

$$\begin{aligned} A\vec{x} &= A(\underbrace{b_1\hat{u}^{(1)} + b_2\hat{u}^{(2)}}_{\vec{x}}) \\ &= b_1A\hat{u}^{(1)} + b_2A\hat{u}^{(2)} \\ &= b_1\lambda_1\hat{u}^{(1)} + b_2\lambda_2\hat{u}^{(2)} \end{aligned}$$

$$\begin{aligned} \vec{x} &= b_1\hat{u}^{(1)} + b_2\hat{u}^{(2)} \\ \|\vec{x}\|=1 \Rightarrow b_1^2+b_2^2 &= 1 \\ \vec{x} &= \hat{u}^{(1)} \end{aligned}$$

$$\begin{aligned} \|A\vec{x}\|^2 &= (b_1\lambda_1\hat{u}^{(1)} + b_2\lambda_2\hat{u}^{(2)}) \cdot (b_1\lambda_1\hat{u}^{(1)} + b_2\lambda_2\hat{u}^{(2)}) \\ &= b_1^2\lambda_1^2 + b_2^2\lambda_2^2 \quad \text{to max, pick } b_1=1 \ b_2=0 \end{aligned}$$

Corollary

To maximize $\vec{x} \cdot A\vec{x}$ over unit vectors, pick \vec{x} to be top eigenvector of A .

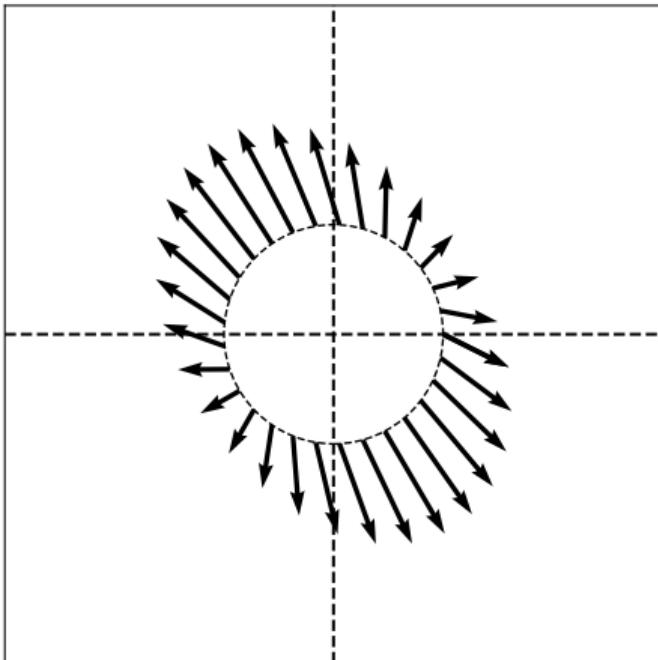
Example

- Maximize $\underbrace{4x_1^2 + 2x_2^2 + 3x_1x_2}$ subject to $x_1^2 + x_2^2 = 1$

$$\begin{pmatrix} x_1 & x_2 \end{pmatrix} \begin{pmatrix} 4 & 1.5 \\ 1.5 & 2 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = 4x_1^2 + 2x_2^2 + 5x_1x_2$$

A

Observation #2



- ▶ $\vec{f}(\vec{x})$ rotates \vec{x} towards the “top” eigenvector \vec{v} .
- ▶ \vec{v} is an equilibrium.

The Power Method

- ▶ Method for computing the top eigenvector/value of A .
- ▶ Initialize $\vec{x}^{(0)}$ randomly
- ▶ Repeat until convergence:
 - ▶ Set $\vec{x}^{(i+1)} = A\vec{x}^{(i)} / \|A\vec{x}^{(i)}\|$

DSC 140B

Representation Learning

Lecture 05 | Part 1

Change of Basis Matrices

Changing Basis

- ▶ Suppose $\vec{x} = \begin{pmatrix} a_1 \\ a_2 \end{pmatrix} = a_1 \hat{e}^{(1)} + a_2 \hat{e}^{(2)}$.
- ▶ $\hat{u}^{(1)}$ and $\hat{u}^{(2)}$ form a new, **orthonormal** basis \mathcal{U} .
- ▶ What is $[\vec{x}]_{\mathcal{U}}$?
- ▶ That is, what are b_1 and b_2 in $\vec{x} = b_1 \hat{u}^{(1)} + b_2 \hat{u}^{(2)}$.

$$[\vec{x}]_u = \begin{pmatrix} \vec{x} \cdot \hat{u}^{(1)} \\ \vec{x} \cdot \hat{u}^{(2)} \end{pmatrix} = \begin{pmatrix} \sqrt{3}/4 + 1/2 \\ -1/4 + \sqrt{3}/2 \end{pmatrix}$$

Exercise

Find the coordinates of \vec{x} in the new basis:

$$\hat{u}^{(1)} = (\sqrt{3}/2, 1/2)^T$$

$$\hat{u}^{(2)} = (-1/2, \sqrt{3}/2)^T$$

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

$$\vec{x} \cdot \hat{u}^{(1)} = \begin{pmatrix} 1/2 \\ 1 \end{pmatrix} \cdot \begin{pmatrix} \sqrt{3}/2 \\ 1/2 \end{pmatrix} = \frac{\sqrt{3}}{4} + \frac{1}{2}$$

Change of Basis

- ▶ Suppose $\hat{u}^{(1)}$ and $\hat{u}^{(2)}$ are our new, **orthonormal** basis vectors.
- ▶ We know $\vec{x} = x_1 \hat{e}^{(1)} + x_2 \hat{e}^{(2)}$
- ▶ We want to write $\vec{x} = b_1 \hat{u}^{(1)} + b_2 \hat{u}^{(2)}$
- ▶ Solution

$$b_1 = \vec{x} \cdot \hat{u}^{(1)} \qquad b_2 = \vec{x} \cdot \hat{u}^{(2)}$$

Change of Basis Matrix

- ▶ Changing basis is a linear transformation

$$\vec{f}(\vec{x}) = (\vec{x} \cdot \hat{u}^{(1)})\hat{u}^{(1)} + (\vec{x} \cdot \hat{u}^{(2)})\hat{u}^{(2)} = \begin{pmatrix} \vec{x} \cdot \hat{u}^{(1)} \\ \vec{x} \cdot \hat{u}^{(2)} \end{pmatrix}_{\mathcal{U}}$$

- ▶ We can represent it with a matrix

$$\begin{pmatrix} \uparrow & \uparrow \\ f(\hat{e}^{(1)}) & f(\hat{e}^{(2)}) \\ \downarrow & \downarrow \end{pmatrix}$$

Example $f(\hat{e}^{(1)}) = \begin{pmatrix} \hat{e}^{(1)} \cdot \hat{u}^{(1)} \\ \hat{e}^{(1)} \cdot \hat{u}^{(2)} \end{pmatrix}$

 $= \begin{pmatrix} \sqrt{3}/2 \\ -1/2 \end{pmatrix}$

$\hat{u}^{(1)} = (\sqrt{3}/2, 1/2)^T$

$\hat{u}^{(2)} = (-1/2, \sqrt{3}/2)^T$

$f(\hat{e}^{(1)}) =$

$f(\hat{e}^{(2)}) =$

$A = \begin{pmatrix} \sqrt{3}/2 & 1/2 \\ -1/2 & \sqrt{3}/2 \\ f(\hat{e}^{(1)}) & f(\hat{e}^{(2)}) \end{pmatrix}$

$f(\hat{e}^{(2)}) = \begin{pmatrix} \hat{e}^{(2)} \cdot \hat{u}^{(1)} \\ \hat{e}^{(2)} \cdot \hat{u}^{(2)} \end{pmatrix} = \begin{pmatrix} 1/2 \\ \sqrt{3}/2 \end{pmatrix}$

Observation

- ▶ The new basis vectors become the **rows** of the matrix.

$$[\vec{x}]_u = \begin{pmatrix} \vec{x} \cdot \hat{u}^{(1)} \\ \vec{x} \cdot \hat{u}^{(2)} \end{pmatrix}$$

Example

$$\begin{aligned}\vec{x} \cdot \hat{u}^{(1)} &= \begin{pmatrix} 1/2 \\ 1 \end{pmatrix} \cdot \begin{pmatrix} \sqrt{3}/2 \\ 1/2 \end{pmatrix} \\ &= \frac{\sqrt{3}}{4} + \frac{1}{2}\end{aligned}$$

- Multiplying by this matrix gives the coordinate vector w.r.t. the new basis.

$$\hat{u}^{(1)} = (\sqrt{3}/2, 1/2)^T$$

$$\hat{u}^{(2)} = (-1/2, \sqrt{3}/2)^T$$

$$A = \begin{pmatrix} \sqrt{3}/2 & 1/2 \\ -1/2 & \sqrt{3}/2 \end{pmatrix}$$

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

$$\begin{pmatrix} \sqrt{3}/2 & 1/2 \\ -1/2 & \sqrt{3}/2 \end{pmatrix} \begin{pmatrix} 1/2 \\ 1 \end{pmatrix} = \begin{pmatrix} \sqrt{3}/4 + 1/2 \\ -\frac{1}{4} + \frac{\sqrt{3}}{2} \end{pmatrix}$$

Change of Basis Matrix

- ▶ Let $\hat{u}^{(1)}, \dots, \hat{u}^{(d)}$ form an orthonormal basis \mathcal{U} .
- ▶ The matrix U whose **rows** are the new basis vectors is the **change of basis** matrix from the standard basis to \mathcal{U} :

$$U = \begin{pmatrix} \leftarrow \hat{u}^{(1)} \rightarrow \\ \leftarrow \hat{u}^{(2)} \rightarrow \\ \vdots \\ \leftarrow \hat{u}^{(d)} \rightarrow \end{pmatrix}$$

Change of Basis Matrix

- ▶ If U is the change of basis matrix, $[\vec{x}]_{\mathcal{U}} = U\vec{x}$
- ▶ To go *back* to the standard basis, use U^T :

$$\vec{x} = U^T[\vec{x}]_{\mathcal{U}}$$

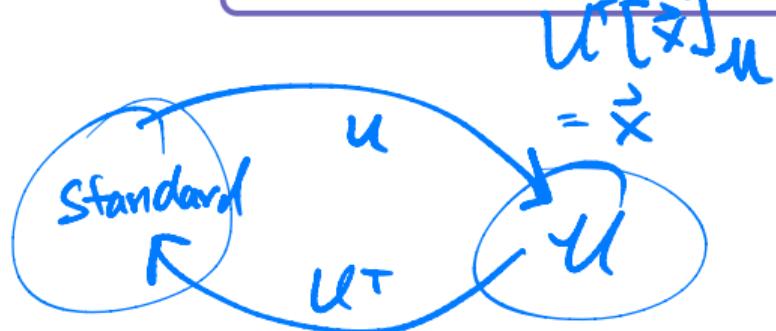
$$U\vec{x} = [\vec{x}]_u$$

$$U^T[\vec{x}]_u = \vec{x}$$

Exercise

Let U be the change of basis matrix for \mathcal{U} .
What is $U^T U$? *it is the identity matrix, I.*

Hint: What is $U^T(U\vec{x})$? $= \vec{x}$



$$I\vec{x} = \vec{x}$$

DSC 140B

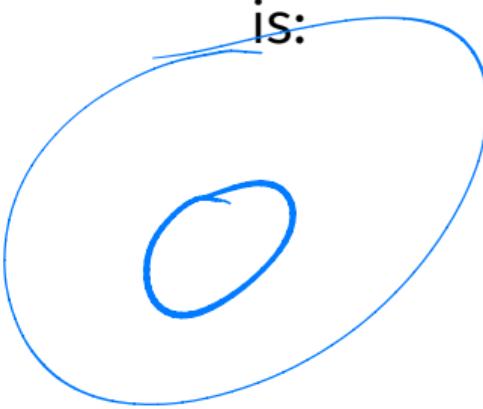
Representation Learning

Lecture 05 | Part 2

Diagonalization

Matrices of a Transformation

- ▶ Let $\vec{f} : \mathbb{R}^d \rightarrow \mathbb{R}^d$ be a linear transformation
- ▶ The matrix representing \vec{f} wrt the **standard basis** is:


$$A = \begin{pmatrix} \uparrow & \uparrow & \uparrow & \uparrow \\ [\vec{f}(\hat{e}^{(1)})] & \vec{f}(\hat{e}^{(2)}) & \dots & \vec{f}(\hat{e}^{(d)}) \\ \downarrow & \downarrow & \downarrow & \downarrow \end{pmatrix}$$

Matrices of a Transformation

- ▶ If we use a different basis $\mathcal{U} = \{\hat{u}^{(1)}, \dots, \hat{u}^{(d)}\}$, the matrix representing \vec{f} is:

$$A_{\mathcal{U}} = \begin{pmatrix} \uparrow & \uparrow & \uparrow & \uparrow \\ [\vec{f}(\hat{u}^{(1)})]_{\mathcal{U}} & [\vec{f}(\hat{u}^{(2)})]_{\mathcal{U}} & \cdots & [\vec{f}(\hat{u}^{(d)})]_{\mathcal{U}} \\ \downarrow & \downarrow & \downarrow & \downarrow \end{pmatrix}$$

- ▶ If $\vec{y} = A\vec{x}$, then $[\vec{y}]_{\mathcal{U}} = A_{\mathcal{U}}[\vec{x}]_{\mathcal{U}}$

Diagonal Matrices

- ▶ Diagonal matrices are very nice / easy to work with.
- ▶ Suppose A is a matrix. Is there a basis \mathcal{U} where $A_{\mathcal{U}}$ is diagonal?
- ▶ Yes! *If A is symmetric.*

The Spectral Theorem¹

- ▶ **Theorem:** Let A be an $n \times n$ symmetric matrix. Then there exist n eigenvectors of A which are all mutually orthogonal.

¹for symmetric matrices

Eigendecomposition

- ▶ If A is a symmetric matrix, we can pick d of its eigenvectors $\hat{u}^{(1)}, \dots, \hat{u}^{(d)}$ to form an orthonormal basis.
- ▶ Any vector \vec{x} can be written in terms of this **eigenbasis**.
- ▶ This is called its **eigendecomposition**:

$$\vec{x} = b_1 \hat{u}^{(1)} + b_2 \hat{u}^{(2)} + \dots + b_d \hat{u}^{(d)}$$

Matrix in the Eigenbasis

- ▶ **Claim:** the matrix of a linear transformation \vec{f} , written in a basis of its eigenvectors, is a **diagonal** matrix.
- ▶ The entries along the diagonal will be the **eigenvalues**.

$$b_1 \hat{u}^{(1)} + b_2 \hat{u}^{(2)} + \dots$$

Why?

$$A_{\mathcal{U}} = \begin{pmatrix} \uparrow & \uparrow & \uparrow & \uparrow \\ [\vec{f}(\hat{u}^{(1)})]_{\mathcal{U}} & [\vec{f}(\hat{u}^{(2)})]_{\mathcal{U}} & \dots & [\vec{f}(\hat{u}^{(d)})]_{\mathcal{U}} \\ \downarrow & \downarrow & \downarrow & \downarrow \end{pmatrix}$$

- ▶ $\vec{f}(\hat{u}^{(1)}) = \lambda_1 \hat{u}^{(1)}$, so $[\vec{f}(\hat{u}^{(1)})]_{\mathcal{U}} = (\lambda_1, 0, \dots, 0)^T$.
- ▶ $\vec{f}(\hat{u}^{(2)}) = \lambda_2 \hat{u}^{(2)}$, so $[\vec{f}(\hat{u}^{(2)})]_{\mathcal{U}} = (0, \lambda_2, \dots, 0)^T$.
- ▶ ...

$$A_{\mathcal{U}} = \begin{pmatrix} \lambda_1 & 0 & 0 & \cdots & \cdots \\ 0 & \lambda_2 & 0 & \ddots & \vdots \\ 0 & 0 & \lambda_3 & \ddots & \vdots \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \ddots & \vdots \end{pmatrix}$$

Matrix Multiplication

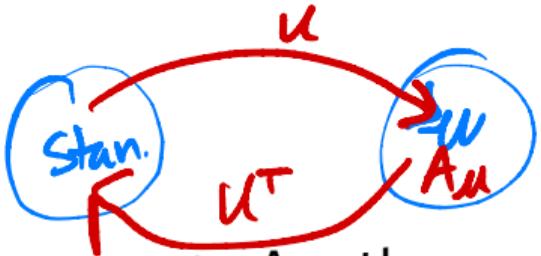
- ▶ We have seen that matrix multiplication evaluates a linear transformation.
- ▶ In the standard basis:

$$\vec{f}(\vec{x}) = A\vec{x}$$

- ▶ In another basis:

$$[\vec{f}(\vec{x})]_{\mathcal{U}} = A_{\mathcal{U}} [\vec{x}]_{\mathcal{U}}$$

Diagonalization



► Another way to compute $\vec{f}(x)$, starting with \vec{x} in the standard basis:

1. Change basis to the eigenbasis with U . $[\vec{x}]_U$

2. Apply \vec{f} in the eigenbasis with the diagonal A_U . $A_U [\vec{x}]_U = [f(\vec{x})]_U$

3. Go back to the standard basis with U^T .

► That is, $A\vec{x} = U^T A_U U \vec{x}$. It follows that $A = U^T A_U U$.

Spectral Theorem (Again)

- ▶ **Theorem:** Let A be an $n \times n$ symmetric matrix.
Then there exists an orthogonal matrix U and a
diagonal matrix Λ such that $A = U^T \Lambda U$.
- ▶ The *rows* of U are the eigenvectors of A , and the
entries of Λ are its eigenvalues.
- ▶ U is said to **diagonalize** A .

DSC 140B

Representation Learning

Lecture 05 | Part 3

Dimensionality Reduction

High Dimensional Data

- ▶ Data is often high dimensional (many features)
- ▶ Example: Netflix user
 - ▶ Number of movies watched
 - ▶ Number of movies saved
 - ▶ Total time watched
 - ▶ Number of logins
 - ▶ Days since signup
 - ▶ Average rating for comedy
 - ▶ Average rating for drama
 - ▶ :

High Dimensional Data

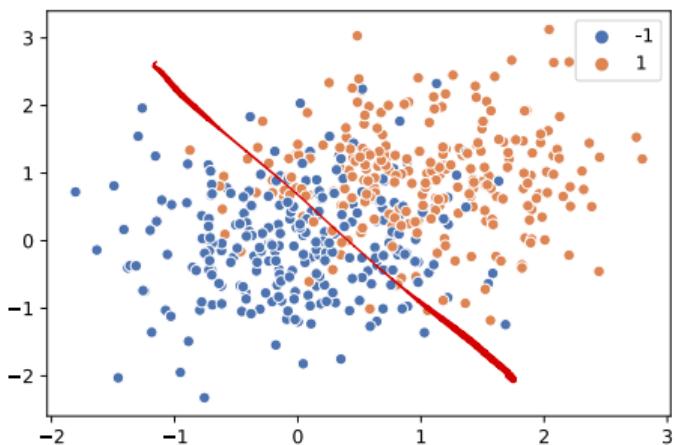
- ▶ More features can give us more information
- ▶ But it can also cause problems
- ▶ **Today:** how do we reduce dimensionality without losing too much information?

More Features, More Problems

- ▶ Difficulties with high dimensional data:
 1. Requires more compute time / space
 2. Hard to visualize / explore
 3. The “curse of dimensionality”: it’s harder to learn

$$\vec{x} = (x_1, x_2, x_3, x_4, \dots, x_{402})$$

Experiment



- ▶ On this data, low 80% train/test accuracy
- ▶ Add 400 features of pure noise, re-train
- ▶ Now: 100% train accuracy, **58%** test accuracy
- ▶ **Overfitting!**

Task: Dimensionality Reduction

- ▶ We'd often like to **reduce** the dimensionality to improve performance, or to visualize.
- ▶ We will typically lose information
- ▶ Want to minimize the loss of useful information

Redundancy

- ▶ Two (or more) features may share the same information.
- ▶ Intuition: we may not need all of them.

Today

- ▶ Today we'll think about reducing dimensionality from \mathbb{R}^d to \mathbb{R}^1
- ▶ Next time we'll go from \mathbb{R}^d to $\mathbb{R}^{d'}$, with $d' \leq d$

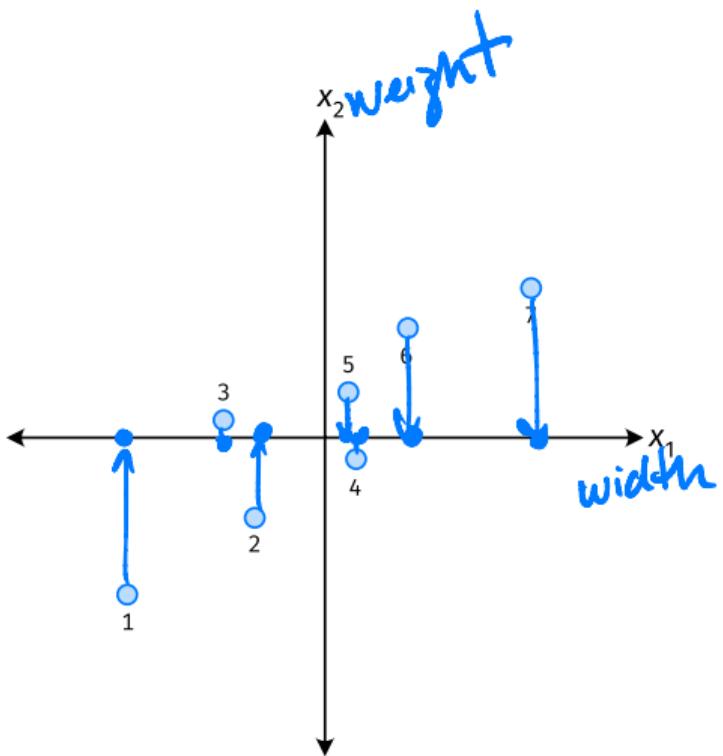
Today's Example

- ▶ Let's say we represent a phone with two features:
 - ▶ x_1 : screen width
 - ▶ x_2 : phone weight
- ▶ Both measure a phone's "size".
- ▶ Instead of representing a phone with both x_1 and x_2 , can we just use a single number, z ?
 - ▶ Reduce dimensionality from 2 to 1.

First Approach: Remove Features

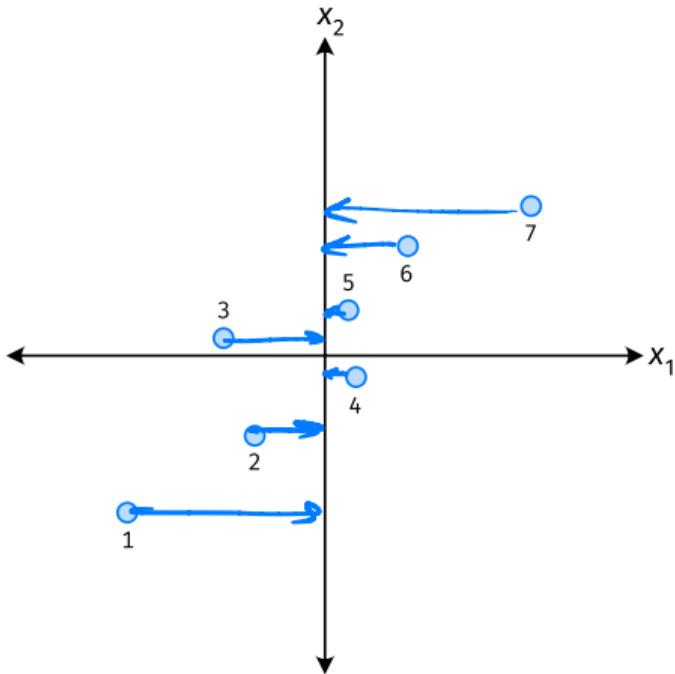
- ▶ Screen width and weight share information.
- ▶ **Idea:** keep one feature, remove the other.
- ▶ That is, set new feature $z = x_1$ (or $z = x_2$).

Removing Features



- ▶ Say we set $z^{(i)} = \vec{x}_1^{(i)}$ for each phone, i .
- ▶ Observe: $z^{(4)} > z^{(5)}$.
- ▶ Is phone 4 really “larger” than phone 5?

Removing Features



- ▶ Say we set $z^{(i)} = \vec{x}_2^{(i)}$ for each phone, i .
- ▶ Observe: $z^{(3)} > z^{(4)}$.
- ▶ Is phone 3 really “larger” than phone 4?

Better Approach: Mixtures of Features

- ▶ **Idea:** z should be a combination of x_1 and x_2 .
- ▶ One approach: linear combination.

$$\begin{aligned} z &= u_1 x_1 + u_2 x_2 & \vec{u} &= (u_1, u_2) \\ &= \vec{u} \cdot \vec{x} & \vec{x} &= (x_1, x_2) \end{aligned}$$

- ▶ u_1, \dots, u_2 are the mixture coefficients; we can choose them.

$1000x_1 + 2000x_2$
 $|x_1 + 2x_2|$

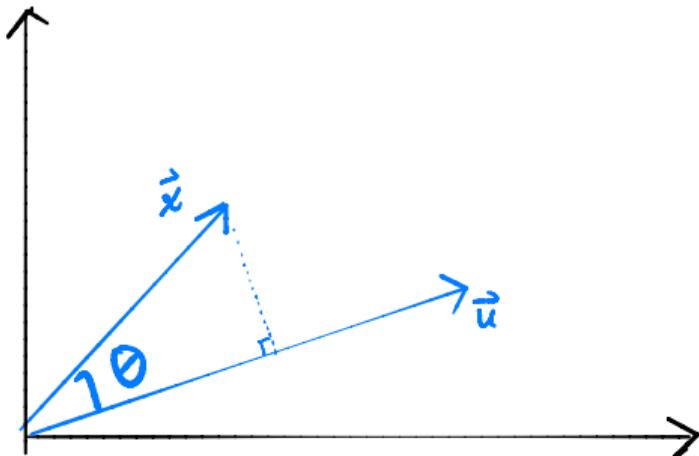
Normalization

- ▶ Mixture coefficients generalize proportions.
- ▶ We could assume, e.g., $|u_1| + |u_2| = 1$.
- ▶ But it makes the math easier if we assume $u_1^2 + u_2^2 = 1$.
- ▶ Equivalently, if $\vec{u} = (u_1, u_2)^T$, assume $\|\vec{u}\| = 1$

$$\|\vec{u}\| = \sqrt{u_1^2 + u_2^2}$$

$$z = \vec{x} \cdot \vec{u} = \|\vec{x}\| \|\vec{u}\| \cos \theta$$

Geometric Interpretation



- ▶ z measures how much of \vec{x} is in the direction of \vec{u}
- ▶ If $\vec{u} = (1, 0)^T$, then $z = x_1$
- ▶ If $\vec{u} = (0, 1)^T$, then $z = x_2$

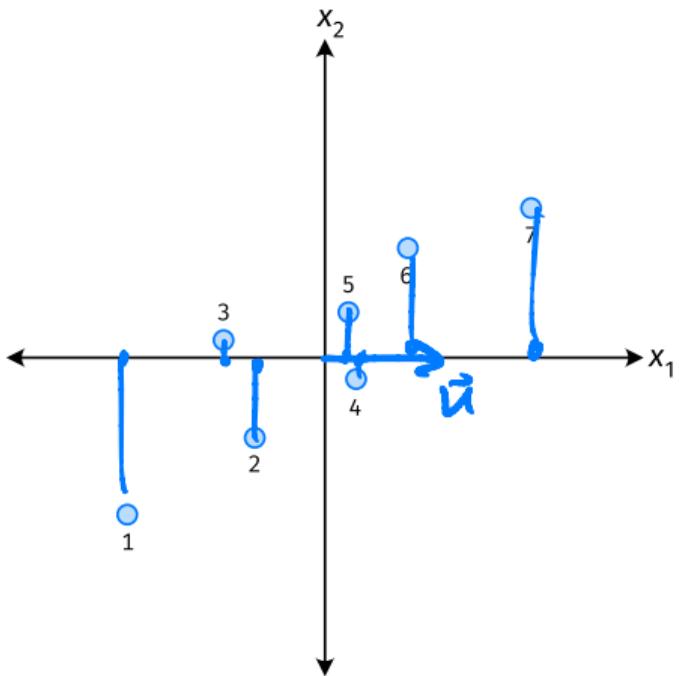
$$\begin{aligned} z &= u_1 x_1 + u_2 x_2 \\ &= x_1 \end{aligned}$$

Choosing \vec{u}

- ▶ Suppose we have only two features:
 - ▶ x_1 : screen size
 - ▶ x_2 : phone thickness
- ▶ We'll create single new feature, z , from x_1 and x_2 .
 - ▶ Assume $z = u_1 x_1 + u_2 x_2 = \vec{x} \cdot \vec{u}$
 - ▶ Interpretation: z is a measure of a phone's size
- ▶ How should we choose $\vec{u} = (u_1, u_2)^T$?

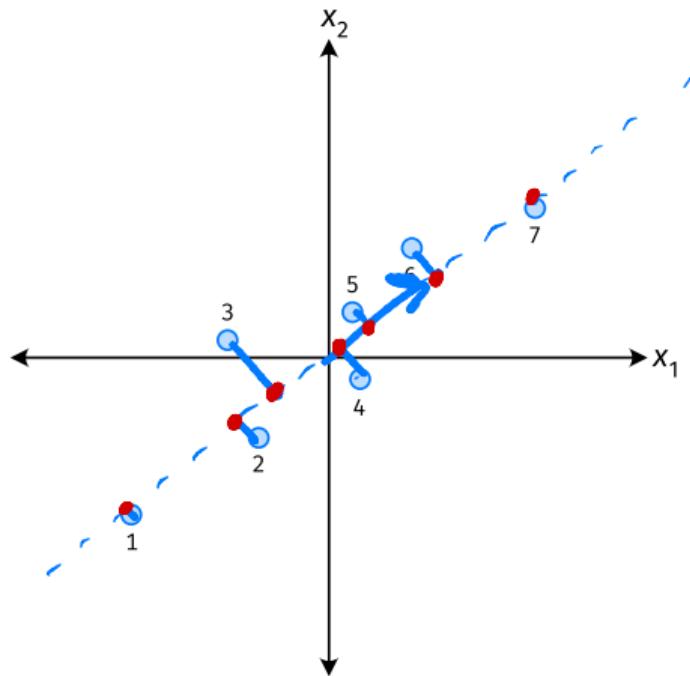
Example

$$\vec{u} = (1, 0)$$



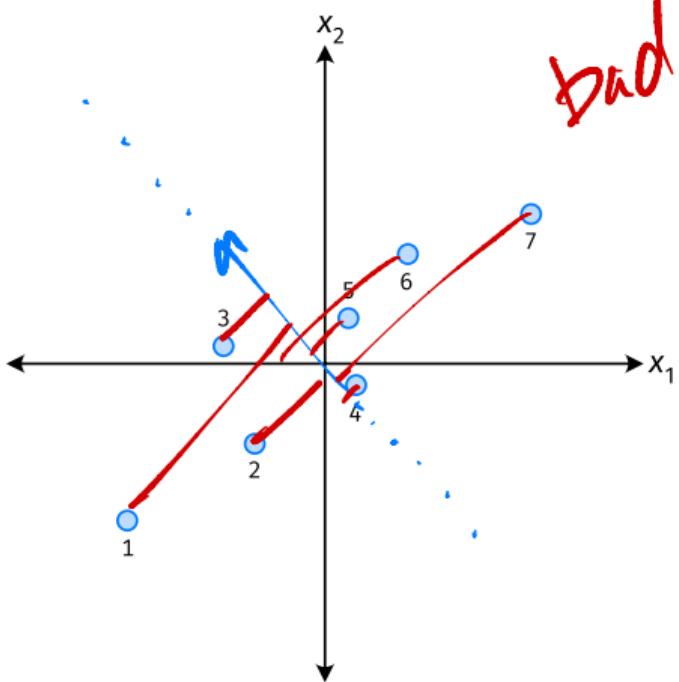
- ▶ \vec{u} defines a direction
- ▶ $\vec{z}^{(i)} = \vec{x}^{(i)} \cdot \vec{u}$ measures position of \vec{x} along this direction

Example



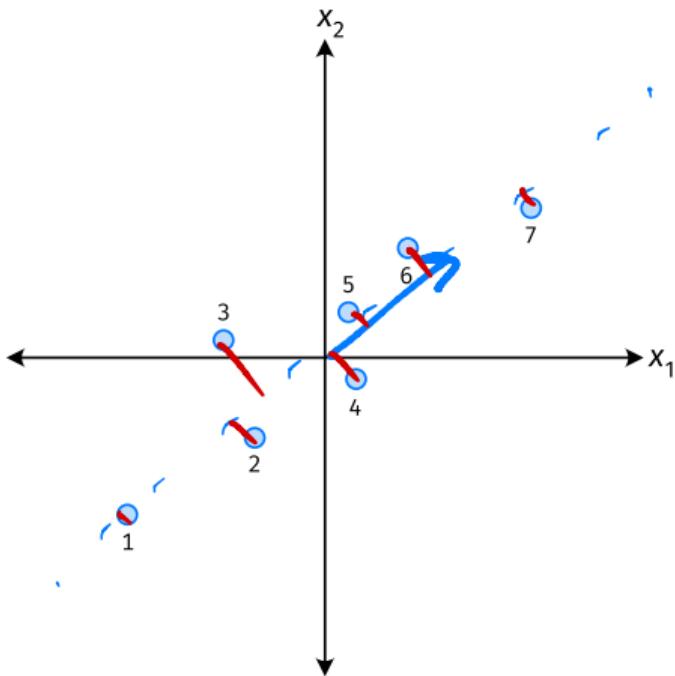
- ▶ \vec{u} defines a direction
- ▶ $\vec{z}^{(i)} = \vec{x}^{(i)} \cdot \vec{u}$ measures position of \vec{x} along this direction

Example



- ▶ Phone “size” varies most along a diagonal direction.
- ▶ Along direction of “max variance”, phones are well-separated.
- ▶ **Idea:** \vec{u} should point in direction of “max variance”.

Example



- ▶ Phone “size” varies most along a diagonal direction.
- ▶ Along direction of “max variance”, phones are well-separated.
- ▶ **Idea:** \vec{u} should point in direction of “max variance”.

Our Algorithm (Informally)

- ▶ **Given:** data points $\vec{x}^{(1)}, \dots, \vec{x}^{(n)} \in \mathbb{R}^d$
- ▶ Pick \vec{u} to be the direction of “max variance”
- ▶ Create a new feature, z , for each point:

$$z^{(i)} = \vec{x}^{(i)} \cdot \vec{u}$$

PCA

- ▶ This algorithm is called **Principal Component Analysis**, or **PCA**.
- ▶ The direction of maximum variance is called the **principal component**.

Exercise

Suppose the direction of maximum variance in a data set is

$$\vec{u} = (1/\sqrt{2}, -1/\sqrt{2})^T$$

Let

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

What are $z^{(1)}$ and $z^{(2)}$?

Problem

- ▶ How do we compute the “direction of maximum variance”?

DSC 140B

Representation Learning

Lecture 05 | Part 4

Covariance Matrices

Variance

- We know how to compute the variance of a set of numbers $X = \{x^{(1)}, \dots, x^{(n)}\}$:

$$\text{Var}(X) = \frac{1}{n} \sum_{i=1}^n (x^{(i)} - \mu)^2$$

- The variance measures the “spread” of the data

Generalizing Variance

- ▶ If we have two features, x_1 and x_2 , we can compute the variance of each as usual:

$$\text{Var}(x_1) = \frac{1}{n} \sum_{i=1}^n (\vec{x}_1^{(i)} - \mu_1)^2$$

$$\text{Var}(x_2) = \frac{1}{n} \sum_{i=1}^n (\vec{x}_2^{(i)} - \mu_2)^2$$

- ▶ Can also measure how x_1 and x_2 vary together.

Measuring Similar Information

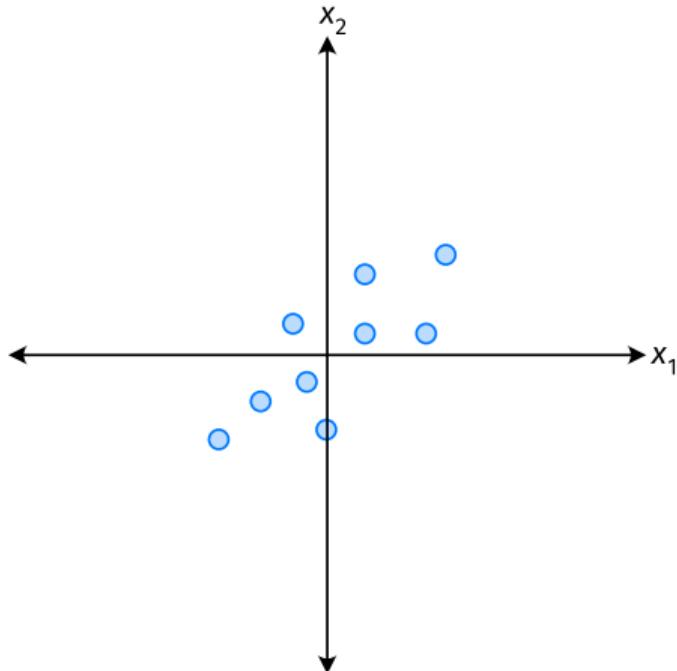
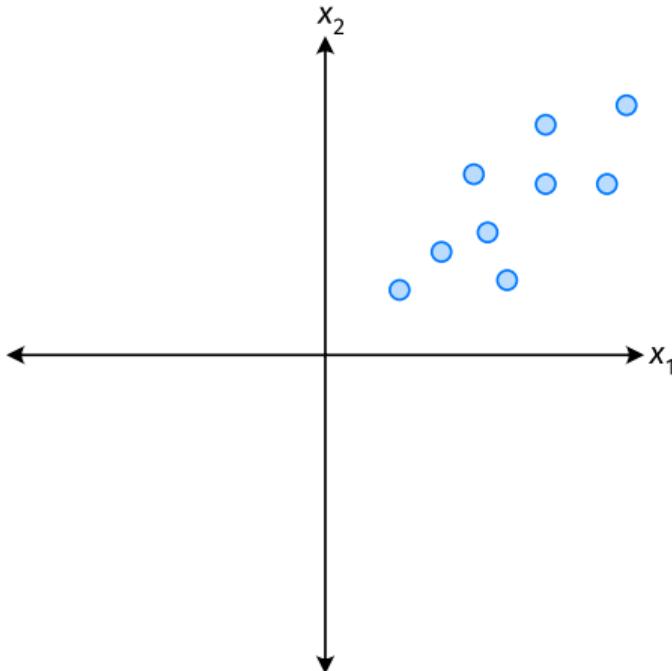
- ▶ Features which share information if they *vary together*.
 - ▶ A.k.a., they “co-vary”
- ▶ Positive association: when one is above average, so is the other
- ▶ Negative association: when one is above average, the other is below average

Examples

- ▶ Positive: temperature and ice cream cones sold.
- ▶ Positive: temperature and shark attacks.
- ▶ Negative: temperature and coats sold.

Centering

- ▶ First, it will be useful to **center** the data.



Centering

- ▶ Compute the mean of each feature:

$$\mu_j = \frac{1}{n} \sum_1^n \vec{x}_j^{(i)}$$

- ▶ Define new centered data:

$$\vec{z}^{(i)} = \begin{pmatrix} \vec{x}_1^{(i)} - \mu_1 \\ \vec{x}_2^{(i)} - \mu_2 \\ \vdots \\ \vec{x}_d^{(i)} - \mu_d \end{pmatrix}$$

Centering (Equivalently)

- ▶ Compute the mean of all data points:

$$\mu = \frac{1}{n} \sum_1^n \vec{x}^{(i)}$$

- ▶ Define new centered data:

$$\vec{z}^{(i)} = \vec{x}^{(i)} - \mu$$

Exercise

Center the data set:

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

$$\vec{x}^{(2)} = (-1, -1, 0)^T$$

$$\vec{x}^{(3)} = (0, 2, 3)^T$$

Quantifying Co-Variance

- ▶ One approach is as follows².

$$\text{Cov}(x_i, x_j) = \frac{1}{n} \sum_{k=1}^n \vec{x}_i^{(k)} \vec{x}_j^{(k)}$$

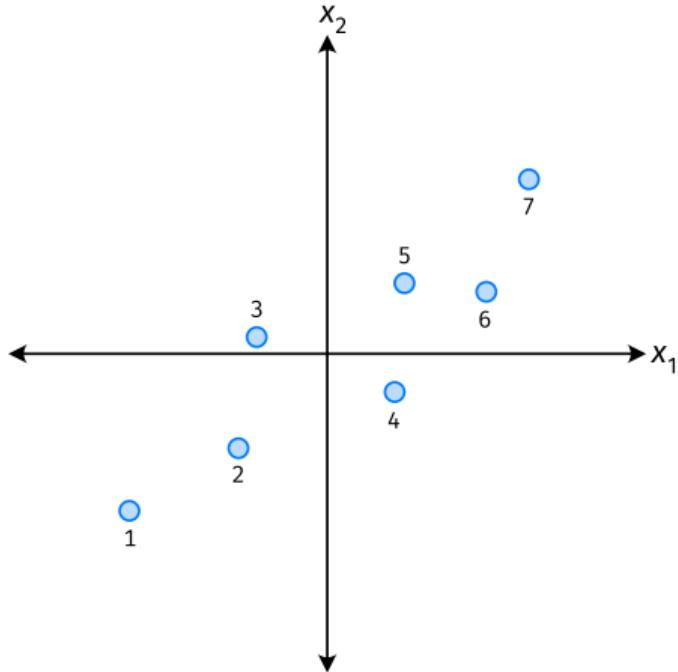
- ▶ For each data point, multiply the value of feature i and feature j , then average these products.
- ▶ This is the **covariance** of features i and j .

²Assuming centered data

Quantifying Covariance

- ▶ Assume the data are **centered**.

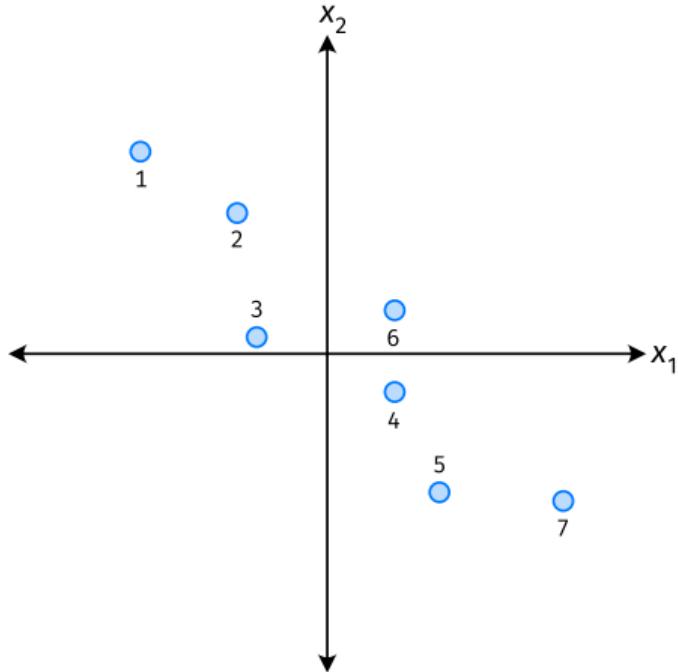
$$\text{Covariance} = \frac{1}{7} \sum_{i=1}^7 \vec{x}_1^{(i)} \times \vec{x}_2^{(i)}$$



Quantifying Covariance

- ▶ Assume the data are **centered**.

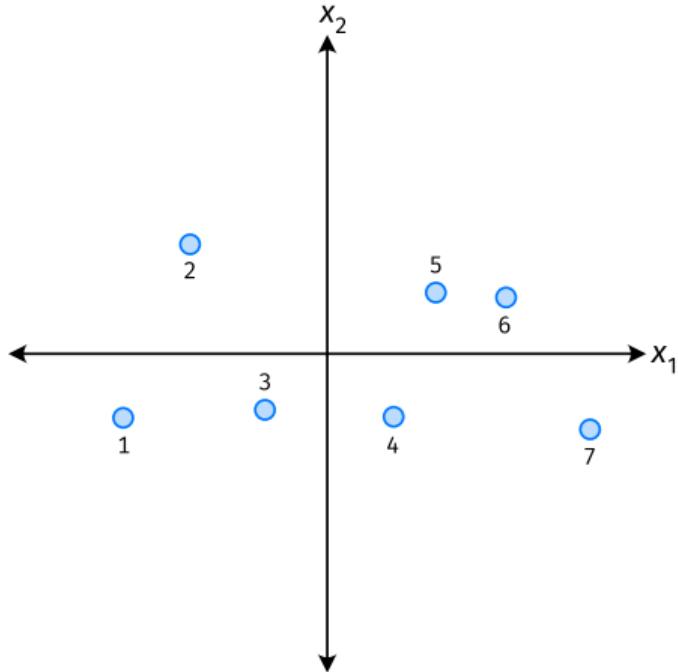
$$\text{Covariance} = \frac{1}{7} \sum_{i=1}^7 \vec{x}_1^{(i)} \times \vec{x}_2^{(i)}$$



Quantifying Covariance

- ▶ Assume the data are **centered**.

$$\text{Covariance} = \frac{1}{7} \sum_{i=1}^7 \vec{x}_1^{(i)} \times \vec{x}_2^{(i)}$$



Quantifying Covariance

- ▶ The **covariance** quantifies extent to which two variables vary together.
- ▶ Assume we have centered the data.
- ▶ The **sample covariance** of feature i and j is:

$$\sigma_{ij} = \frac{1}{n} \sum_{k=1}^n \vec{x}_i^{(k)} \vec{x}_j^{(k)}$$

Exercise

True or False: $\sigma_{ij} = \sigma_{ji}$?

$$\sigma_{ij} = \frac{1}{n} \sum_{k=1}^n \vec{X}_i^{(k)} \vec{X}_j^{(k)}$$

Covariance Matrices

- ▶ Given data $\vec{x}^{(1)}, \dots, \vec{x}^{(n)} \in \mathbb{R}^d$.
- ▶ The **sample covariance matrix** C is the $d \times d$ matrix whose ij entry is defined to be σ_{ij} .

$$\sigma_{ij} = \frac{1}{n} \sum_{k=1}^n \vec{x}_i^{(k)} \vec{x}_j^{(k)}$$

Observations

- ▶ Diagonal entries of C are the variances.
- ▶ The matrix is **symmetric!**

Note

- ▶ Sometimes you'll see the sample covariance defined as:

$$\sigma_{ij} = \frac{1}{n - 1} \sum_{k=1}^n \vec{x}_i^{(k)} \vec{x}_j^{(k)}$$

Note the $1/(n - 1)$

- ▶ This is an **unbiased** estimator of the population covariance.
- ▶ Our definition is the **maximum likelihood** estimator.
- ▶ In practice, it doesn't matter: $1/(n - 1) \approx 1/n$.
- ▶ For consistency, in this class use $1/n$.

Computing Covariance

- ▶ There is a “trick” for computing sample covariance matrices.
- ▶ Step 1: make $n \times d$ data matrix, X
- ▶ Step 2: make Z by centering columns of X
- ▶ Step 3: $C = \frac{1}{n}Z^TZ$

Computing Covariance (in code)³

```
>>> mu = X.mean(axis=0)
>>> Z = X - mu
>>> C = 1 / len(X) * Z.T @ Z
```

³Or use np.cov

DSC 140B

Representation Learning

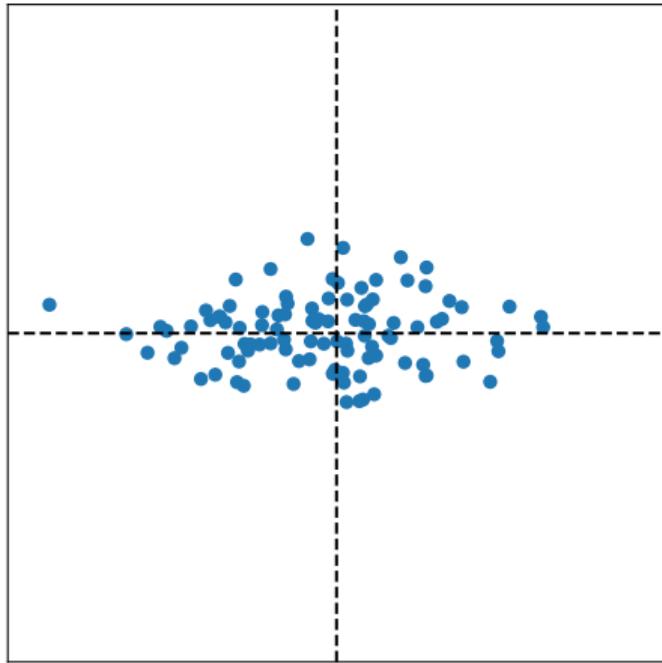
Lecture 05 | Part 5

Visualizing Covariance Matrices

Visualizing Covariance Matrices

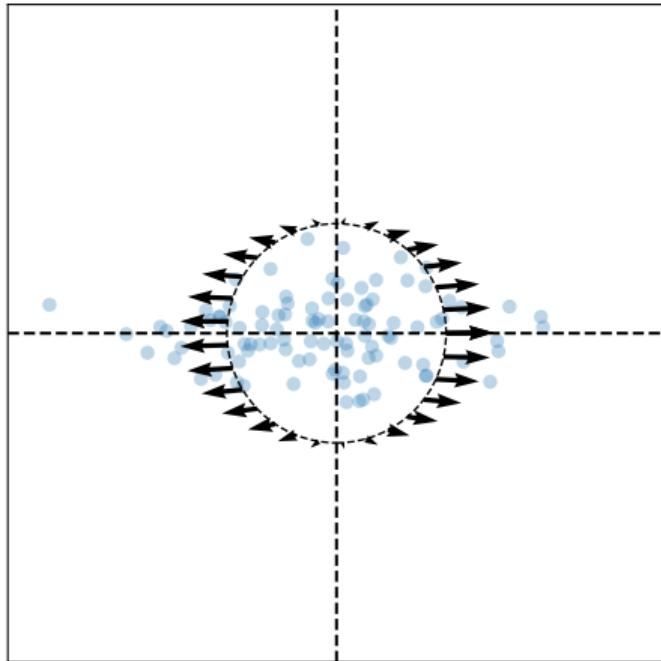
- ▶ Covariance matrices are symmetric.
- ▶ They have axes of symmetry (eigenvectors and eigenvalues).
- ▶ What are they?

Visualizing Covariance Matrices



$$C \approx \begin{pmatrix} & \\ & \end{pmatrix}$$

Visualizing Covariance Matrices

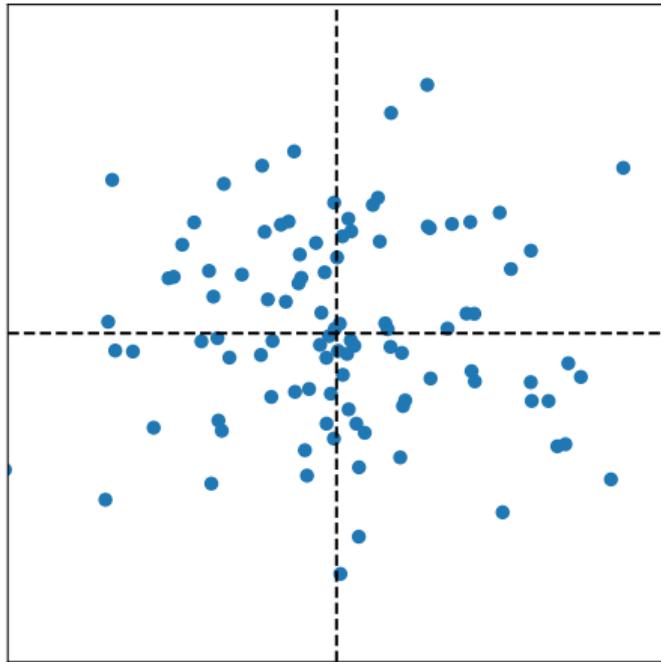


Eigenvectors:

$$\vec{u}^{(1)} \approx$$

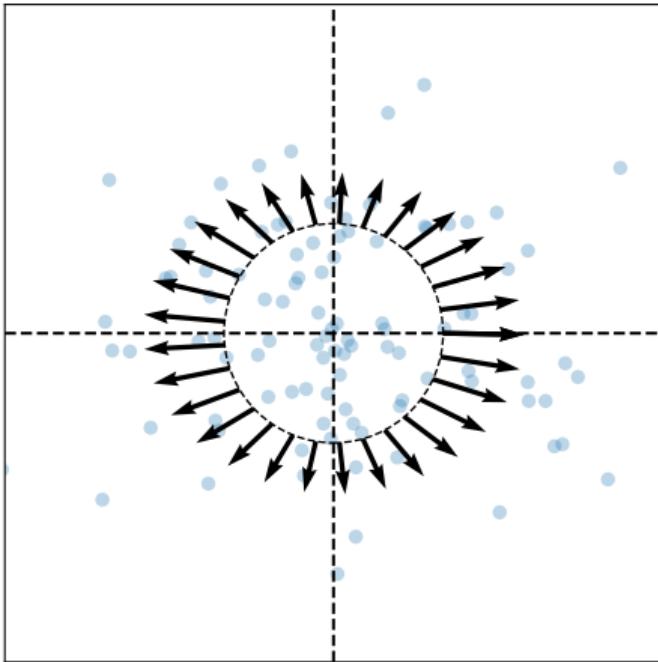
$$\vec{u}^{(2)} \approx$$

Visualizing Covariance Matrices



$$C \approx \begin{pmatrix} & \\ & \end{pmatrix}$$

Visualizing Covariance Matrices

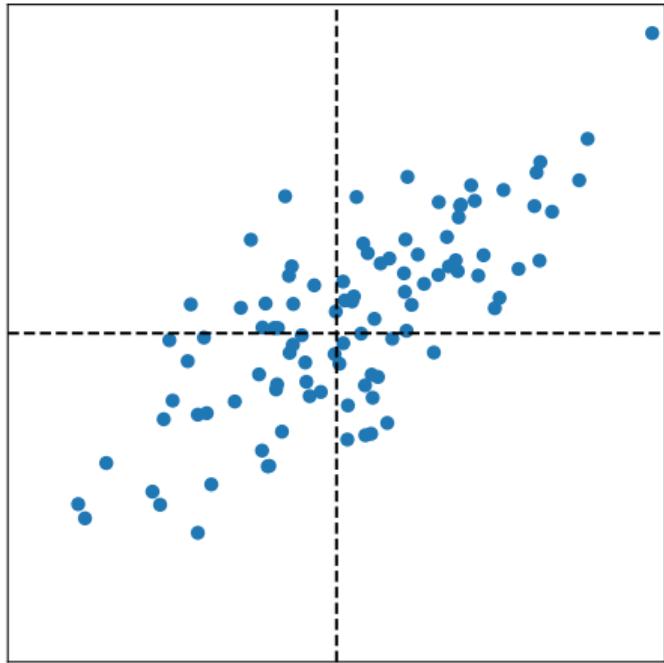


Eigenvectors:

$$\vec{u}^{(1)} \approx$$

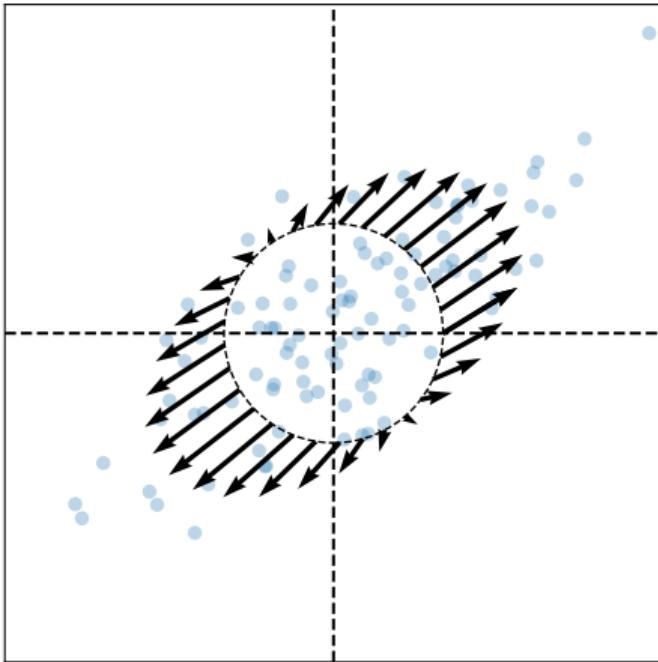
$$\vec{u}^{(2)} \approx$$

Visualizing Covariance Matrices



$$C \approx \begin{pmatrix} & \\ & \end{pmatrix}$$

Visualizing Covariance Matrices



Eigenvectors:

$$\vec{u}^{(1)} \approx$$

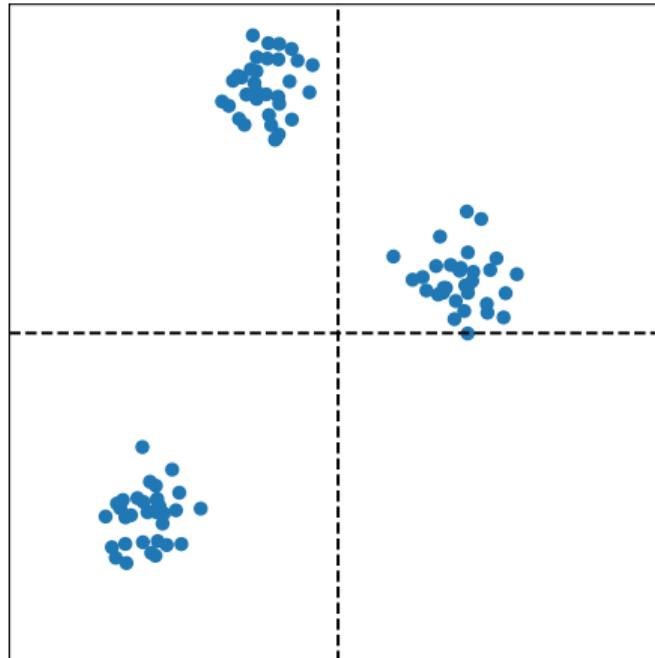
$$\vec{u}^{(2)} \approx$$

Intuitions

- ▶ The **eigenvectors** of the covariance matrix describe the data's “principal directions”
 - ▶ C tells us something about data's shape.
- ▶ The **top eigenvector** points in the direction of “maximum variance”.
- ▶ The **top eigenvalue** is proportional to the variance in this direction.

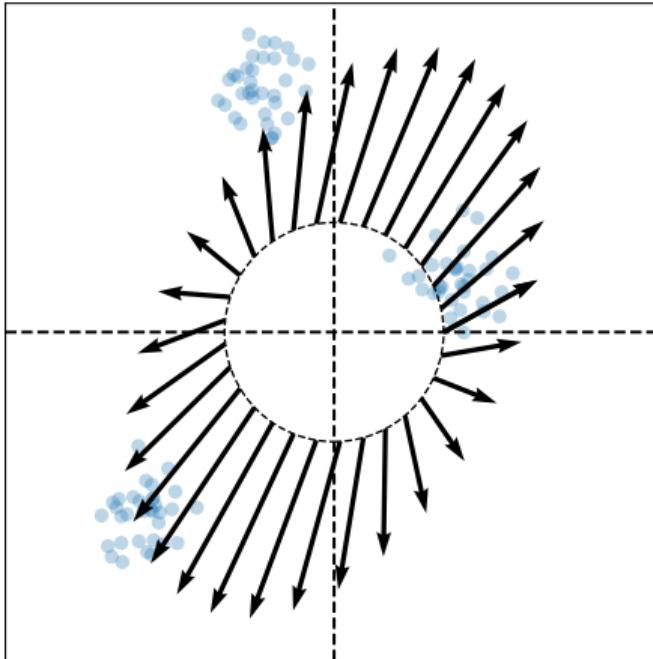
Caution

- ▶ The data doesn't always look like this.
- ▶ We can always compute covariance matrices.
- ▶ They just may not describe the data's shape very well.



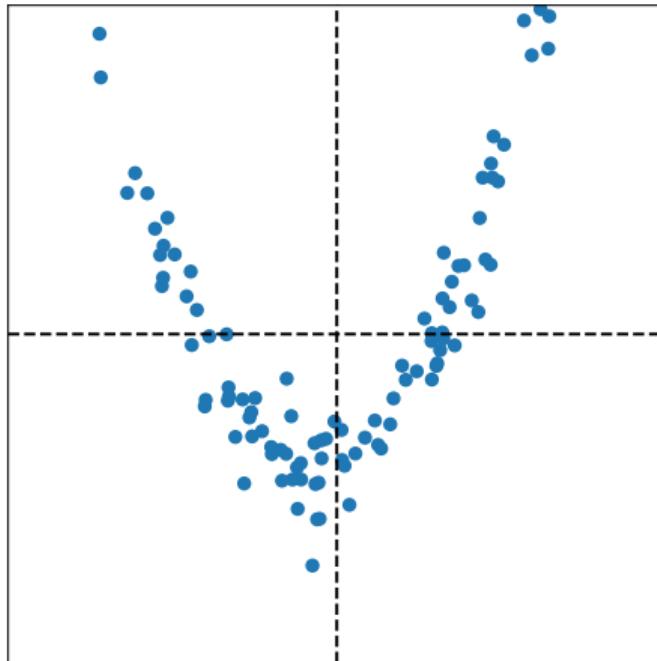
Caution

- ▶ The data doesn't always look like this.
- ▶ We can always compute covariance matrices.
- ▶ They just may not describe the data's shape very well.



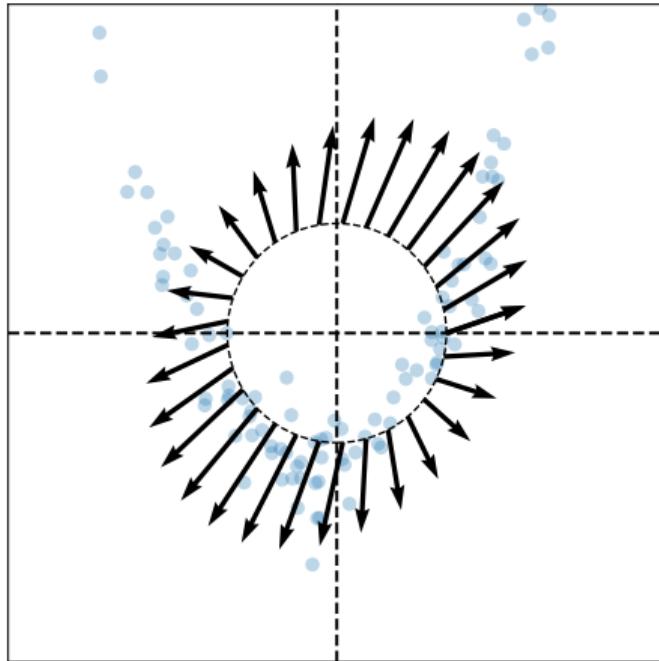
Caution

- ▶ The data doesn't always look like this.
- ▶ We can always compute covariance matrices.
- ▶ They just may not describe the data's shape very well.



Caution

- ▶ The data doesn't always look like this.
- ▶ We can always compute covariance matrices.
- ▶ They just may not describe the data's shape very well.



DSC 140B

Representation Learning

Lecture 06 | Part 1

Dimensionality Reduction

Choosing \vec{u}

$$\|\vec{u}\| = 1$$

$$\vec{u} \cdot \vec{u} = 1$$

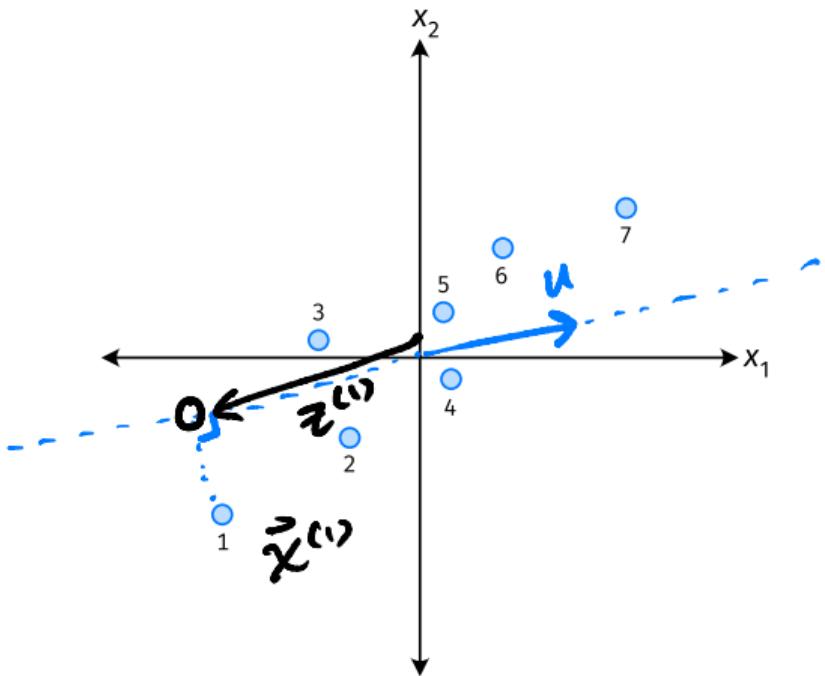
$$u_1^2 + u_2^2 = 1$$

- ▶ Suppose we have only two features:
 - ▶ x_1 : screen size
 - ▶ x_2 : phone thickness
- ▶ We'll create single new feature, z , from x_1 and x_2 .
 - ▶ Assume $z = u_1 x_1 + u_2 x_2 = \vec{x} \cdot \vec{u}$
 - ▶ Interpretation: z is a measure of a phone's size
- ▶ How should we choose $\vec{u} = (u_1, u_2)^T$?

Visualization

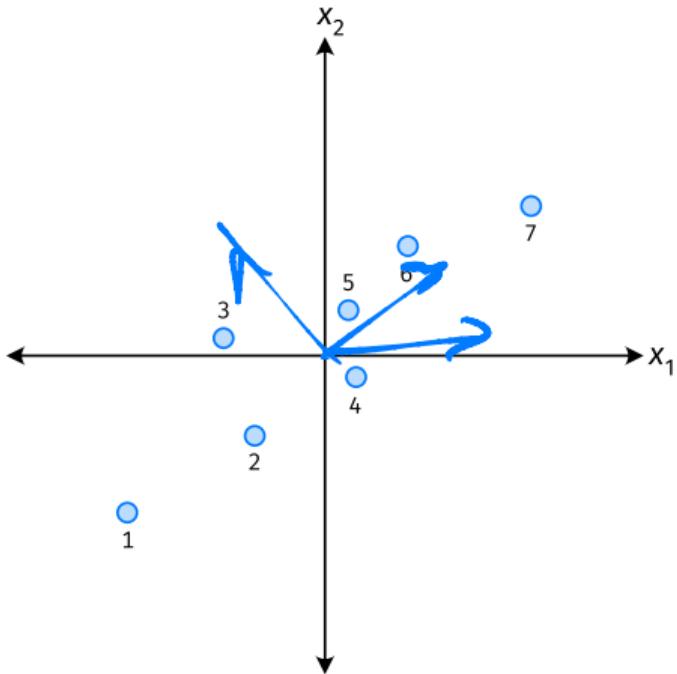
http://dsc140b.com/static/vis/pca-max_variance/

Example



- ▶ \vec{u} defines a direction
- ▶ $\vec{z}^{(i)} = \vec{x}^{(i)} \cdot \vec{u}$ measures position of \vec{x} along this direction

Example



- ▶ Phone “size” varies most along a diagonal direction.
- ▶ Along direction of “max variance”, phones are well-separated.
- ▶ **Idea:** \vec{u} should point in direction of “max variance”.

Our Algorithm (Informally)

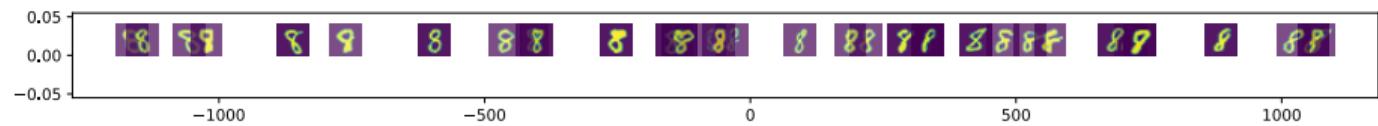
- ▶ **Given:** data points $\vec{x}^{(1)}, \dots, \vec{x}^{(n)} \in \mathbb{R}^d$
- ▶ Pick \vec{u} to be the direction of “max variance”
- ▶ Create a new feature, z , for each point:

$$z^{(i)} = \vec{x}^{(i)} \cdot \vec{u}$$



8 8 8

Example



PCA

- ▶ This algorithm is called **Principal Component Analysis**, or **PCA**.
- ▶ The direction of maximum variance is called the **principal component**.

$$z^{(2)} = \frac{-3}{\sqrt{2}}$$

Exercise

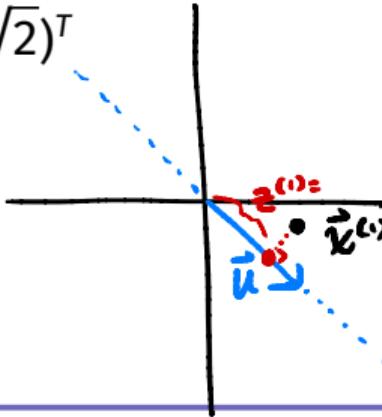
Suppose the direction of maximum variance in a data set is

$$\vec{u} = (1/\sqrt{2}, -1/\sqrt{2})^T$$

Let

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

What are $z^{(1)}$ and $z^{(2)}$?



$$z^{(1)} = \vec{x}^{(1)} \cdot \vec{u} = \begin{pmatrix} 3 \\ -2 \end{pmatrix} \cdot \begin{pmatrix} 1/\sqrt{2} \\ -1/\sqrt{2} \end{pmatrix} = \frac{3}{\sqrt{2}} + \frac{-2}{\sqrt{2}} = \frac{5}{\sqrt{2}}$$

Problem

- ▶ How do we compute the “direction of maximum variance”?

DSC 140B

Representation Learning

Lecture 06 | Part 2

Covariance Matrices

Variance

- We know how to compute the variance of a set of numbers $X = \{x^{(1)}, \dots, x^{(n)}\}$:

$$\text{Var}(X) = \frac{1}{n} \sum_{i=1}^n (x^{(i)} - \mu)^2$$

- The variance measures the “spread” of the data

Generalizing Variance

- ▶ If we have two features, x_1 and x_2 , we can compute the variance of each as usual:

$$\text{Var}(x_1) = \frac{1}{n} \sum_{i=1}^n (\vec{x}_1^{(i)} - \mu_1)^2$$

$$\text{Var}(x_2) = \frac{1}{n} \sum_{i=1}^n (\vec{x}_2^{(i)} - \mu_2)^2$$

- ▶ Can also measure how x_1 and x_2 vary together.

Measuring Similar Information

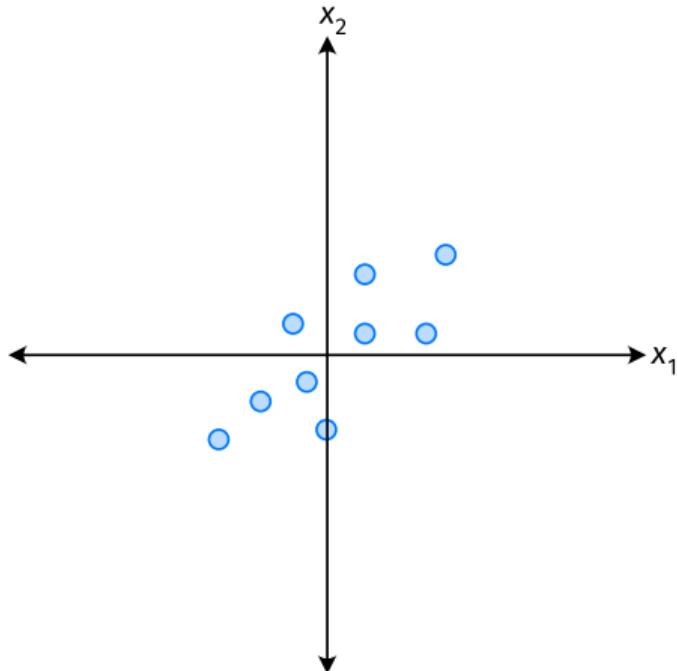
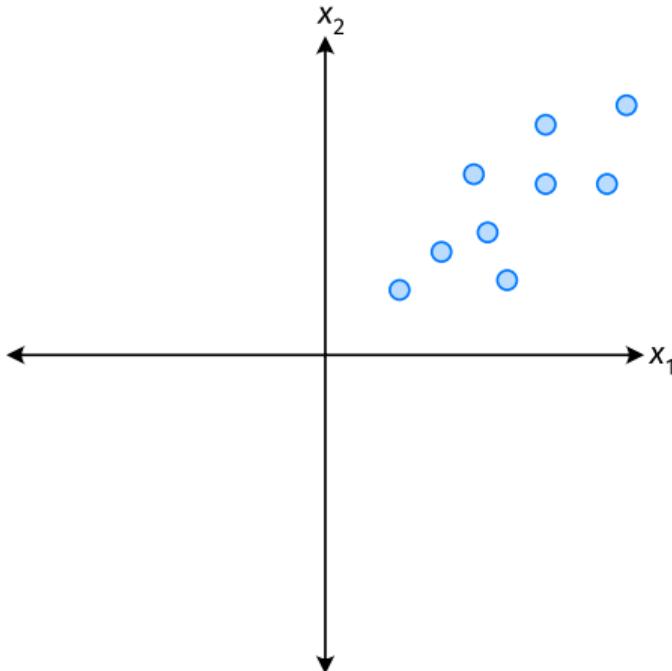
- ▶ Features which share information if they *vary together*.
 - ▶ A.k.a., they “co-vary”
- ▶ Positive association: when one is above average, so is the other
- ▶ Negative association: when one is above average, the other is below average

Examples

- ▶ Positive: temperature and ice cream cones sold.
- ▶ Positive: temperature and shark attacks.
- ▶ Negative: temperature and coats sold.

Centering

- ▶ First, it will be useful to **center** the data.



Centering

- ▶ Compute the mean of each feature:

$$\mu_j = \frac{1}{n} \sum_1^n \vec{x}_j^{(i)}$$

- ▶ Define new centered data:

$$\vec{z}^{(i)} = \begin{pmatrix} \vec{x}_1^{(i)} - \mu_1 \\ \vec{x}_2^{(i)} - \mu_2 \\ \vdots \\ \vec{x}_d^{(i)} - \mu_d \end{pmatrix}$$

Centering (Equivalently)

- ▶ Compute the mean of all data points:

$$\vec{\mu} = \frac{1}{n} \sum_1^n \vec{x}^{(i)}$$

- ▶ Define new centered data:

$$\vec{z}^{(i)} = \vec{x}^{(i)} - \vec{\mu}$$

$$\bar{\mu} = \frac{1}{3} \begin{pmatrix} 0 \\ 3 \\ 6 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \\ 2 \end{pmatrix}$$
$$\bar{z}^{(0)} = \begin{pmatrix} -1 \\ -1 \\ 0 \end{pmatrix} - \begin{pmatrix} 0 \\ 1 \\ 2 \end{pmatrix} = \begin{pmatrix} -1 \\ -2 \\ -2 \end{pmatrix}$$

Exercise

Center the data set:

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

$$\vec{x}^{(2)} = (-1, -1, 0)^T$$

$$\vec{x}^{(3)} = (0, 2, 3)^T$$

$$\bar{z}^{(0)} = \vec{x}^{(0)} - \mu = \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix} - \begin{pmatrix} 0 \\ 1 \\ 2 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}$$

Quantifying Co-Variance

- ▶ One approach is as follows¹.

$$\text{Cov}(x_i, x_j) = \frac{1}{n} \sum_{k=1}^n \vec{x}_i^{(k)} \vec{x}_j^{(k)}$$

- ▶ For each data point, multiply the value of feature i and feature j , then average these products.
- ▶ This is the **covariance** of features i and j .

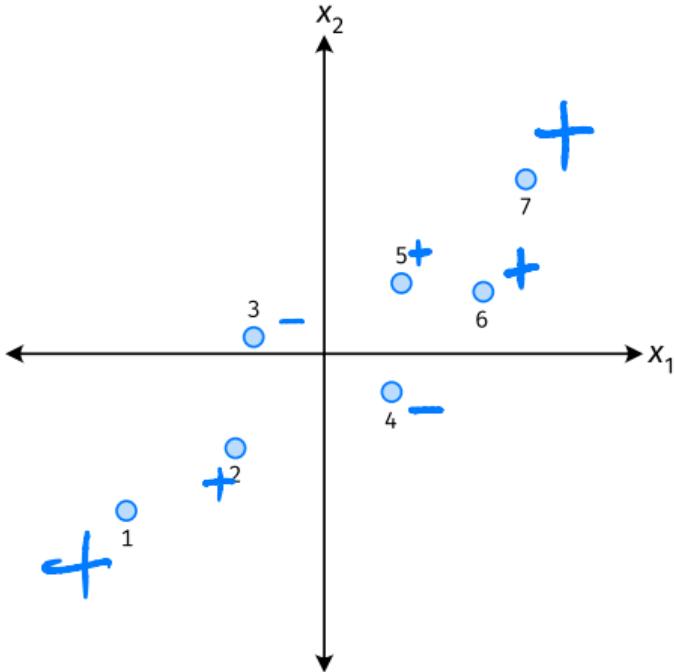
¹Assuming centered data

Quantifying Covariance

- ▶ Assume the data are **centered**.

$$\text{Covariance} = \frac{1}{7} \sum_{i=1}^7 \vec{x}_1^{(i)} \times \vec{x}_2^{(i)}$$

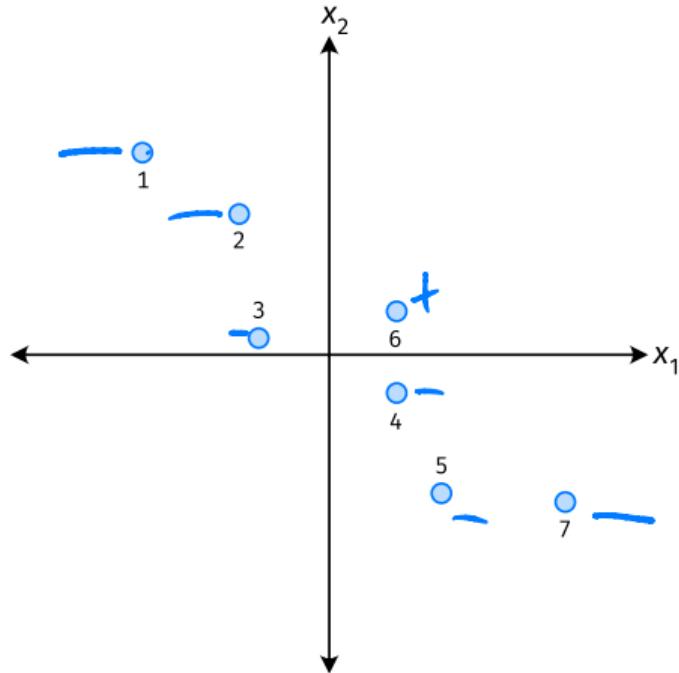
+



Quantifying Covariance

- ▶ Assume the data are **centered**.

$$\text{Covariance} = \frac{1}{7} \sum_{i=1}^7 \vec{x}_1^{(i)} \times \vec{x}_2^{(i)}$$

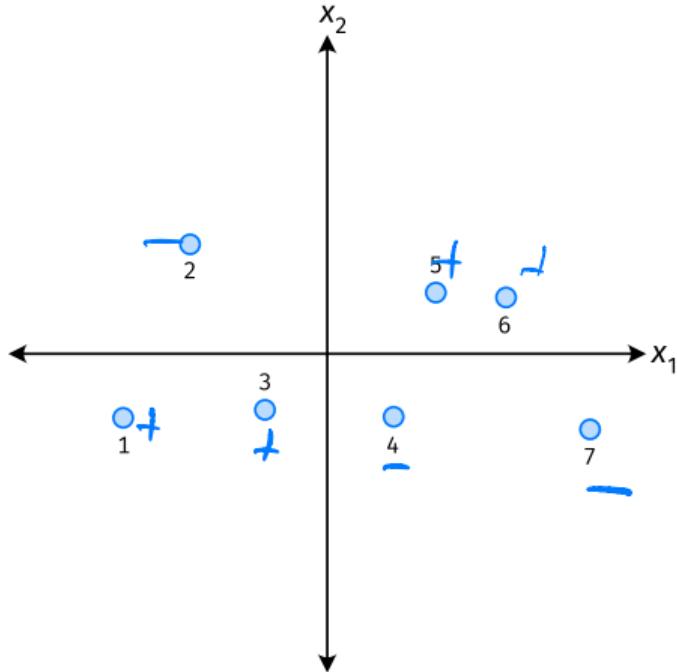


Quantifying Covariance

- ▶ Assume the data are **centered**.

$$\text{Covariance} = \frac{1}{7} \sum_{i=1}^7 \vec{x}_1^{(i)} \times \vec{x}_2^{(i)}$$

≈ 0



Quantifying Covariance

- ▶ The **covariance** quantifies extent to which two variables vary together.
- ▶ Assume we have centered the data.
- ▶ The **sample covariance** of feature i and j is:

$$\sigma_{ij} = \frac{1}{n} \sum_{k=1}^n \vec{x}_i^{(k)} \vec{x}_j^{(k)}$$

$$3 \times 5 = 5 \times 3$$

Exercise

True or False: $\sigma_{ij} = \sigma_{ji}$?

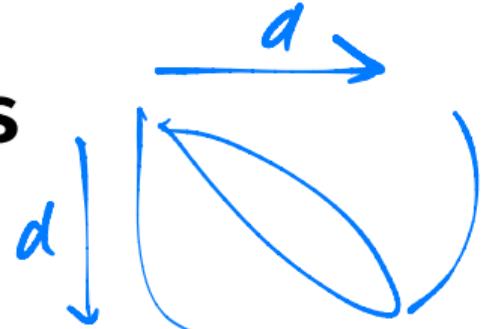
$$\sigma_{ij} = \frac{1}{n} \sum_{k=1}^n \vec{x}_i^{(k)} \vec{x}_j^{(k)}$$

$$\sigma_{ji} = \frac{1}{n} \sum \vec{x}_j^{(k)} \vec{x}_i^{(k)}$$

$$\sigma_{ij} = \sigma_{ji}$$

Covariance Matrices

- Given data $\vec{x}^{(1)}, \dots, \vec{x}^{(n)} \in \mathbb{R}^d$.
- The **sample covariance matrix** C is the $d \times d$ matrix whose ij entry is defined to be σ_{ij} .



$$\sigma_{ij} = \frac{1}{n} \sum_{k=1}^n \vec{x}_i^{(k)} \vec{x}_j^{(k)}$$

$$\sigma_{ii} = \frac{1}{n} \sum (x_i^{(k)} - \underbrace{\mu_i}_{0})^2$$

Observations

- ▶ Diagonal entries of C are the variances.
- ▶ The matrix is **symmetric!**

$$\sigma_{ij} = \sigma_{ji}$$

Note

- ▶ Sometimes you'll see the sample covariance defined as:

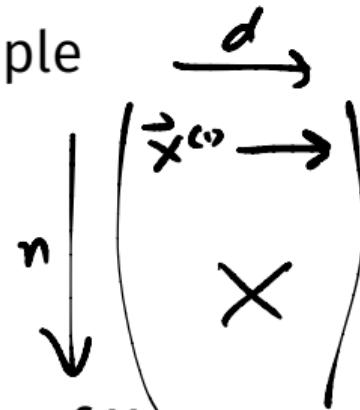
$$\sigma_{ij} = \frac{1}{n - 1} \sum_{k=1}^n \vec{x}_i^{(k)} \vec{x}_j^{(k)}$$

Note the $1/(n - 1)$

- ▶ This is an **unbiased** estimator of the population covariance.
- ▶ Our definition is the **maximum likelihood** estimator.
- ▶ In practice, it doesn't matter: $1/(n - 1) \approx 1/n$.
- ▶ For consistency, in this class use $1/n$.

Computing Covariance

- ▶ There is a “trick” for computing sample covariance matrices.



- ▶ Step 1: make $n \times d$ data matrix, X

- ▶ Step 2: make Z by centering columns of X

- ▶ Step 3: $C = \frac{1}{n}Z^T Z$

Computing Covariance (in code)²

```
>>> mu = X.mean(axis=0)
>>> Z = X - mu
>>> C = 1 / len(X) * Z.T @ Z
```

²Or use np.cov

DSC 140B

Representation Learning

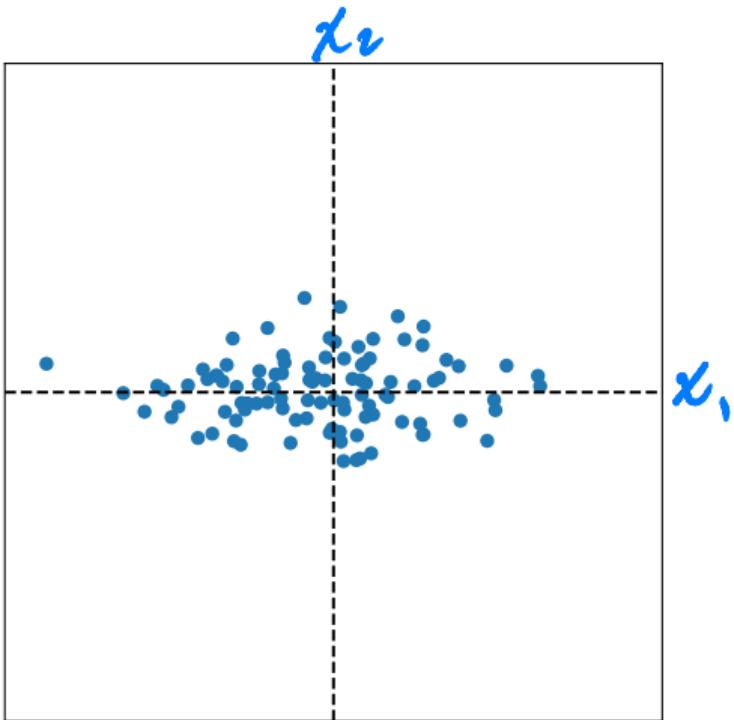
Lecture 06 | Part 3

Visualizing Covariance Matrices

Visualizing Covariance Matrices

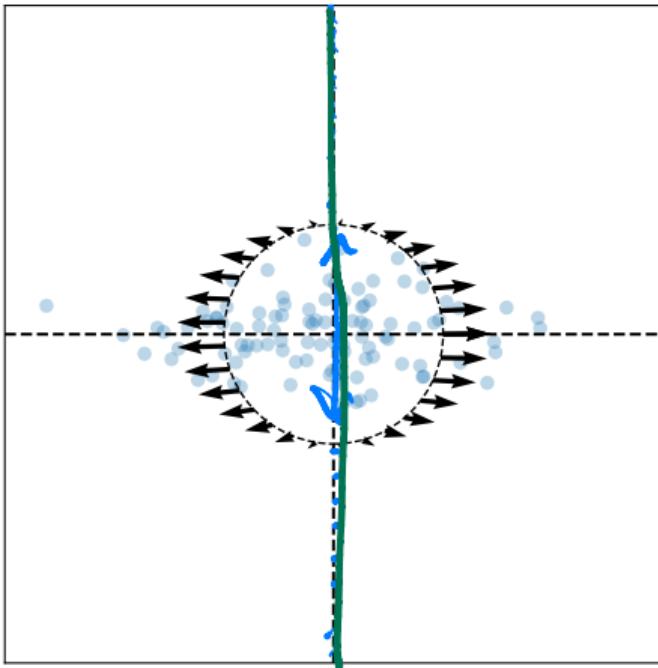
- ▶ Covariance matrices are symmetric.
- ▶ They have axes of symmetry (eigenvectors and eigenvalues).
- ▶ What are they?

Visualizing Covariance Matrices



$$C \approx \begin{pmatrix} 10 & 80 \\ 80 & 5 \end{pmatrix}$$

Visualizing Covariance Matrices



$$\lambda_1 > \lambda_2$$

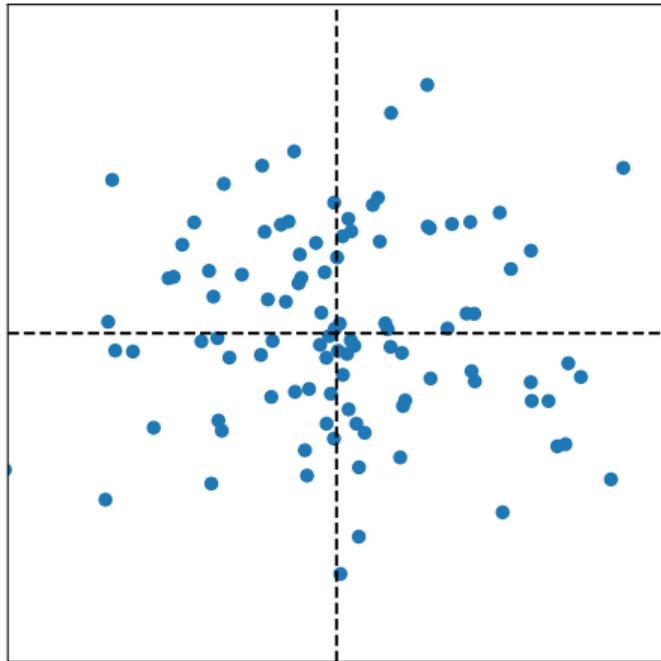
Eigenvectors:

$$\vec{u}^{(1)} \approx (1, 0)$$

$$\vec{u}^{(2)} \approx (0, 1)$$

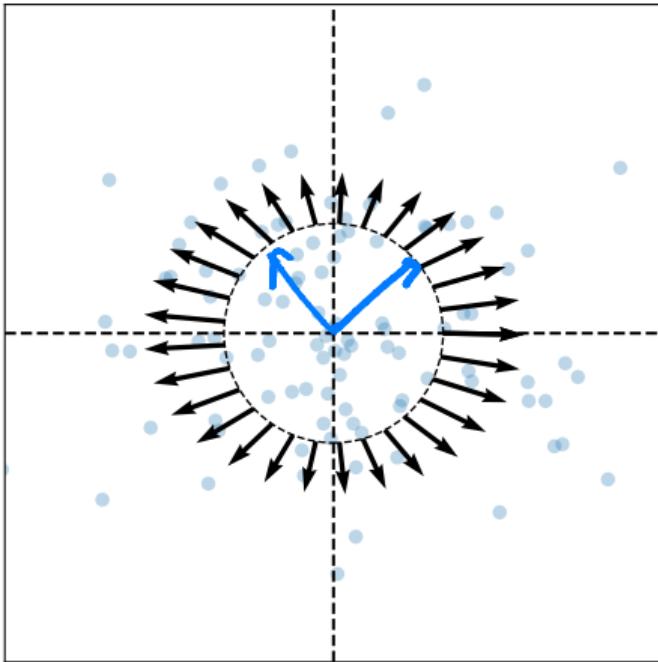
$$(0, -1)$$

Visualizing Covariance Matrices



$$C \approx \begin{pmatrix} 10 & 0 \\ 0 & 10 \end{pmatrix}$$

Visualizing Covariance Matrices

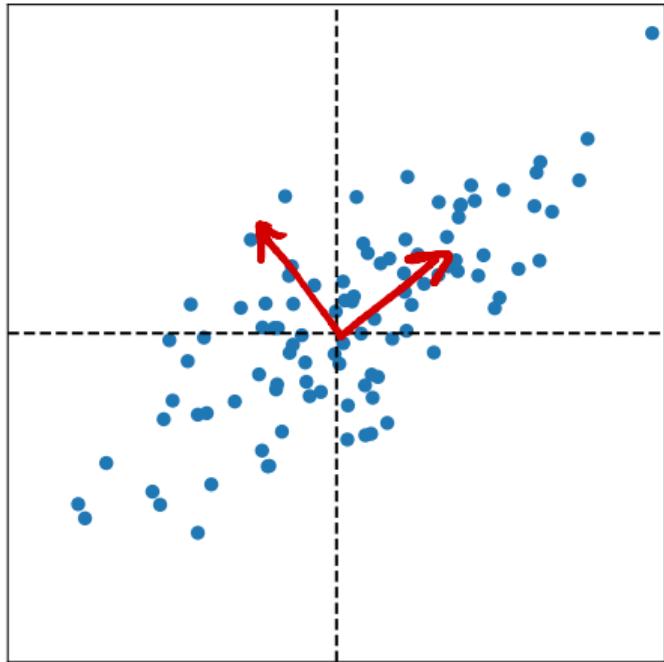


Eigenvectors:

$$\vec{u}^{(1)} \approx$$

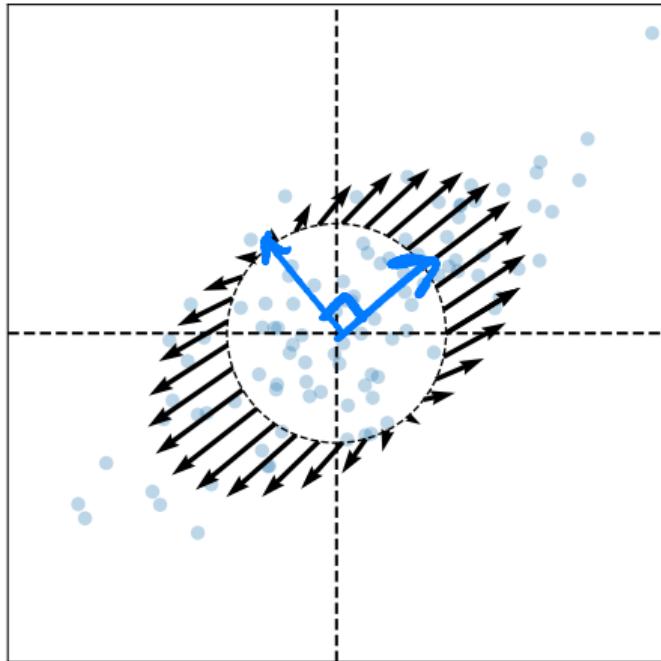
$$\vec{u}^{(2)} \approx$$

Visualizing Covariance Matrices



$$C \approx \begin{pmatrix} 10 & 5 \\ 5 & 7 \end{pmatrix}$$

Visualizing Covariance Matrices



$$\lambda_1 > \lambda_2$$

Eigenvectors:

$$\vec{u}^{(1)} \approx (1, 1)$$

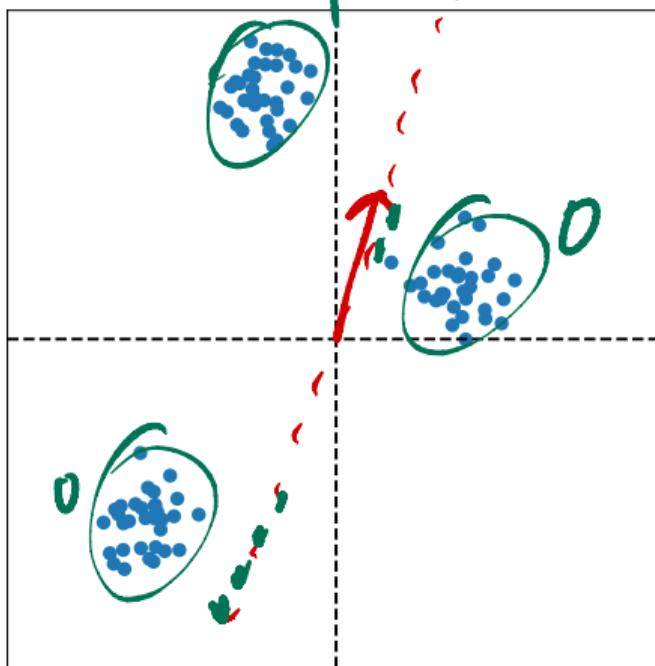
$$\vec{u}^{(2)} \approx (-1, 1)$$

Intuitions

- ▶ The **eigenvectors** of the covariance matrix describe the data's “principal directions”
 - ▶ C tells us something about data's shape.
- ▶ The **top eigenvector** points in the direction of “maximum variance”.
- ▶ The **top eigenvalue** is proportional to the variance in this direction.

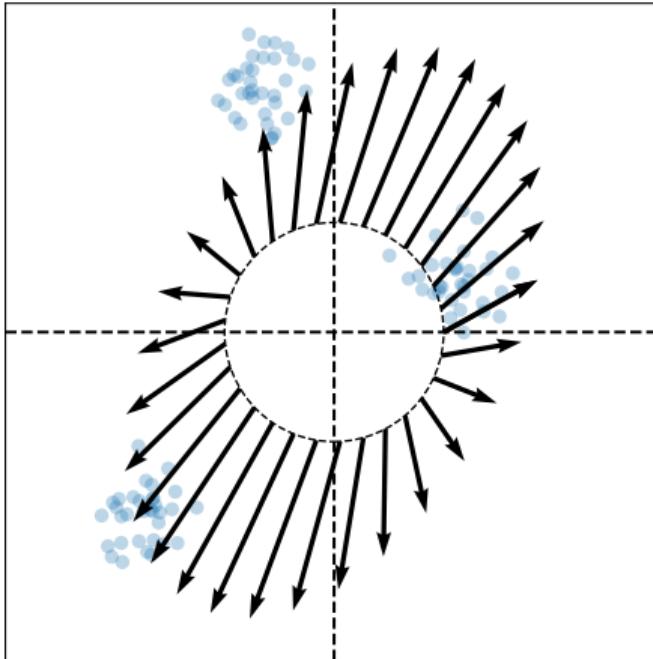
Caution

- ▶ The data doesn't always look like this.
- ▶ We can always compute covariance matrices.
- ▶ They just may not describe the data's shape very well.



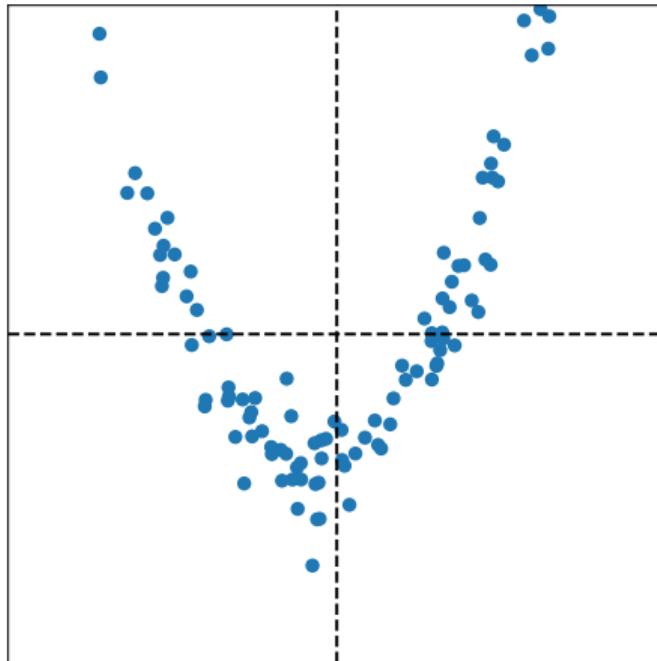
Caution

- ▶ The data doesn't always look like this.
- ▶ We can always compute covariance matrices.
- ▶ They just may not describe the data's shape very well.



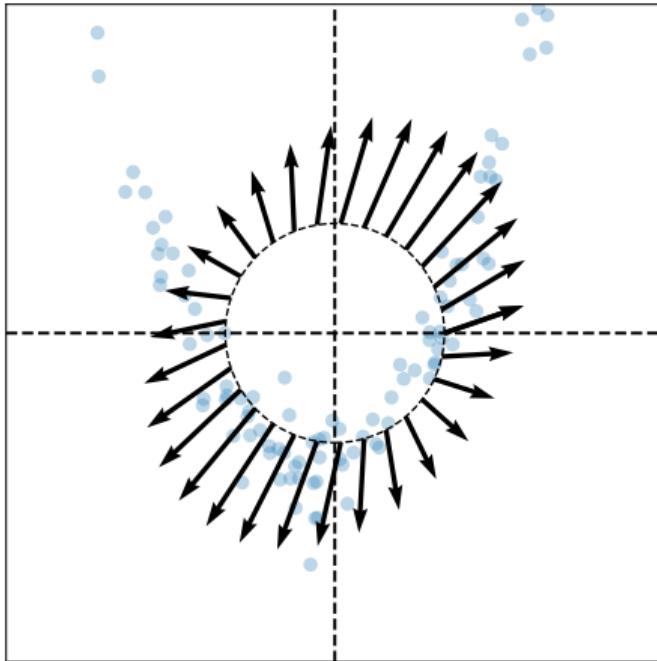
Caution

- ▶ The data doesn't always look like this.
- ▶ We can always compute covariance matrices.
- ▶ They just may not describe the data's shape very well.



Caution

- ▶ The data doesn't always look like this.
- ▶ We can always compute covariance matrices.
- ▶ They just may not describe the data's shape very well.



DSC 140B

Representation Learning

Lecture 06 | Part 4

PCA, More Formally

The Story (So Far)

- ▶ We want to create a single new feature, z .
- ▶ Our idea: $z = \vec{x} \cdot \vec{u}$; choose \vec{u} to point in the “direction of maximum variance”.
- ▶ Intuition: the top eigenvector of the covariance matrix points in direction of maximum variance.

More Formally...

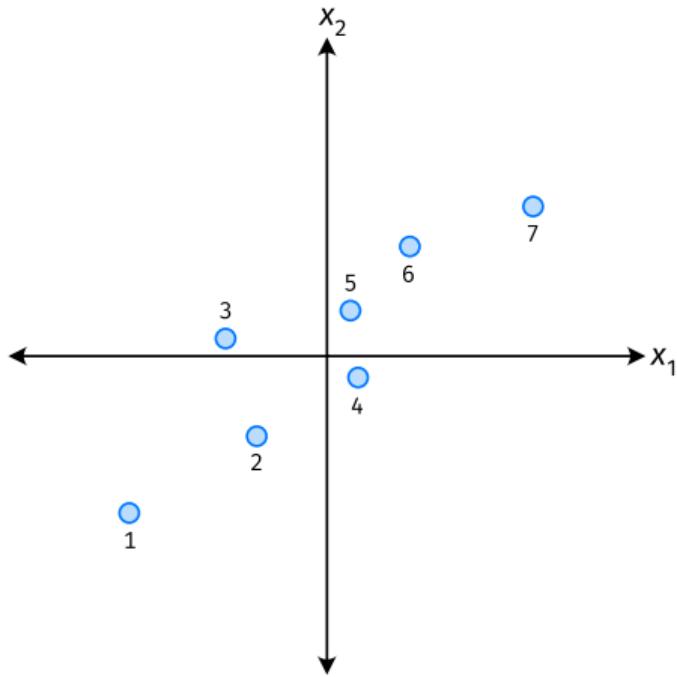
- ▶ We haven't actually defined “direction of maximum variance”
- ▶ Let's derive PCA more formally.

Variance in a Direction

- ▶ Let \vec{u} be a unit vector.
- ▶ $z^{(i)} = \vec{x}^{(i)} \cdot \vec{u}$ is the new feature for $\vec{x}^{(i)}$.
- ▶ The variance of the new features is:

$$\begin{aligned}\text{Var}(z) &= \frac{1}{n} \sum_{i=1}^n (z^{(i)} - \mu_z)^2 \\ &= \frac{1}{n} \sum_{i=1}^n (\vec{x}^{(i)} \cdot \vec{u} - \mu_z)^2\end{aligned}$$

Example



Note

- ▶ If the data are centered, then $\mu_z = 0$ and the variance of the new features is:

$$\begin{aligned}\text{Var}(z) &= \frac{1}{n} \sum_{i=1}^n (z^{(i)})^2 \\ &= \frac{1}{n} \sum_{i=1}^n (\vec{x}^{(i)} \cdot \vec{u})^2\end{aligned}$$

Goal

- ▶ The variance of a data set in the direction of \vec{u} is:

$$g(\vec{u}) = \frac{1}{n} \sum_{i=1}^n (\vec{x}^{(i)} \cdot \vec{u})^2$$

- ▶ Our goal: Find a unit vector \vec{u} which maximizes g .

Claim

$$\frac{1}{n} \sum_{i=1}^n (\vec{x}^{(i)} \cdot \vec{u})^2 = \vec{u}^T C \vec{u}$$

Our Goal (Again)

- ▶ Find a unit vector \vec{u} which maximizes $\vec{u}^T C \vec{u}$.

Claim

- ▶ To maximize $\vec{u}^T C \vec{u}$ over unit vectors, choose \vec{u} to be the top eigenvector of C .
- ▶ Proof:

PCA (for a single new feature)

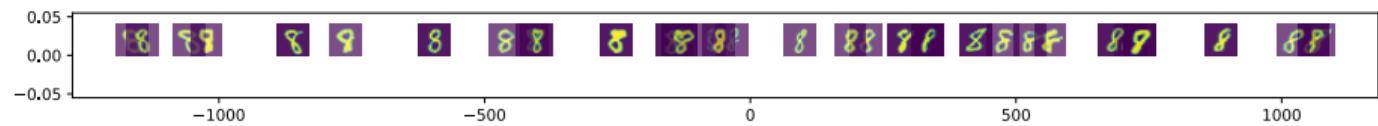
- ▶ **Given:** data points $\vec{x}^{(1)}, \dots, \vec{x}^{(n)} \in \mathbb{R}^d$
- 1. Compute the covariance matrix, C .
- 2. Compute the top eigenvector \vec{u} , of C .
- 3. For $i \in \{1, \dots, n\}$, create new feature:

$$z^{(i)} = \vec{u} \cdot \vec{x}^{(i)}$$

A Parting Example

- ▶ MNIST: 60,000 images in 784 dimensions
- ▶ Principal component: $\vec{u} \in \mathbb{R}^{784}$
- ▶ We can project an image in \mathbb{R}^{784} onto \vec{u} to get a single number representing the image

Example



DSC 140B

Representation Learning

Lecture 07 | Part 1

PCA, More Formally

The Story (So Far)

- ▶ We want to create a single new feature, z .
- ▶ Our idea: $z = \vec{x} \cdot \vec{u}$; choose \vec{u} to point in the “direction of maximum variance”.
- ▶ Intuition: the top eigenvector of the covariance matrix points in direction of maximum variance.

More Formally...

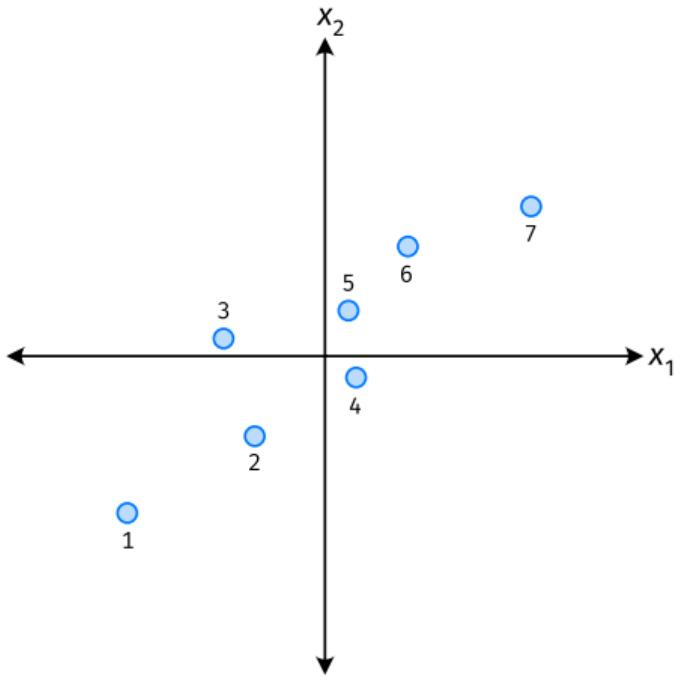
- ▶ We haven't actually defined “direction of maximum variance”
- ▶ Let's derive PCA more formally.

Variance in a Direction

- ▶ Let \vec{u} be a unit vector.
- ▶ $z^{(i)} = \vec{x}^{(i)} \cdot \vec{u}$ is the new feature for $\vec{x}^{(i)}$.
- ▶ The variance of the new features is:

$$\begin{aligned}\text{Var}(z) &= \frac{1}{n} \sum_{i=1}^n (z^{(i)} - \mu_z)^2 \\ &= \frac{1}{n} \sum_{i=1}^n (\vec{x}^{(i)} \cdot \vec{u} - \mu_z)^2\end{aligned}$$

Example



Note

- ▶ If the data are centered, then $\mu_z = 0$ and the variance of the new features is:

$$\begin{aligned}\text{Var}(z) &= \frac{1}{n} \sum_{i=1}^n (z^{(i)})^2 \\ &= \frac{1}{n} \sum_{i=1}^n (\vec{x}^{(i)} \cdot \vec{u})^2\end{aligned}$$

Goal

- ▶ The variance of a data set in the direction of \vec{u} is:

$$g(\vec{u}) = \frac{1}{n} \sum_{i=1}^n (\vec{x}^{(i)} \cdot \vec{u})^2$$
$$= \vec{x}^T C \vec{u}$$

- ▶ Our goal: Find a unit vector \vec{u} which maximizes g .

Claim

$$\frac{1}{n} \sum_{i=1}^n (\vec{x}^{(i)} \cdot \vec{u})^2 = \vec{u}^T C \vec{u}$$

$$u_1^2 + u_2^2 = 1$$

Our Goal (Again)

- ▶ Find a unit vector \vec{u} which maximizes $\vec{u}^T C \vec{u}$.

Claim

$\lambda_1 > \lambda_2 > \dots > \lambda_d$

- ▶ To maximize $\vec{u}^T C \vec{u}$ over unit vectors, choose \vec{u} to be the top eigenvector of C .

- ▶ Proof: Suppose the e-vecs & e-vals of C are $(\vec{v}^{(1)}, \lambda_1), \dots, (\vec{v}^{(d)}, \lambda_d)$

Because C is symmetric, we can assume the eigenvectors are orthonormal.

Let the eigendecomposition of \vec{u} be:

$$\vec{u} = \alpha_1 \vec{v}^{(1)} + \dots + \alpha_d \vec{v}^{(d)}$$

with $\alpha_1^2 + \dots + \alpha_d^2 = 1$.

Claim $C\vec{v}^{(1)} = \lambda_1 \vec{v}^{(1)}$

- To maximize $\vec{u}^T C \vec{u}$ over unit vectors, choose \vec{u} to be the top eigenvector of C .

- Proof: Then

$$\begin{aligned}\vec{u}^T C \vec{u} &= (\alpha_1 \vec{v}^{(1)} + \dots + \alpha_d \vec{v}^{(d)})^T C (\alpha_1 \vec{v}^{(1)} + \dots + \alpha_d \vec{v}^{(d)}) \\ &= (\alpha_1 \vec{v}^{(1)} + \dots + \alpha_d \vec{v}^{(d)})^T (\alpha_1 \lambda_1 \vec{v}^{(1)} + \dots + \alpha_d \lambda_d \vec{v}^{(d)}) \\ &= \alpha_1^2 \lambda_1 + \alpha_2^2 \lambda_2 + \dots + \alpha_d^2 \lambda_d\end{aligned}$$

Maximized by choosing $\alpha_1 = 1 \neq \alpha_2, \dots, \alpha_d = 0$
So $\vec{u} = \vec{v}^{(1)}$

PCA (for a single new feature)

- ▶ **Given:** data points $\vec{x}^{(1)}, \dots, \vec{x}^{(n)} \in \mathbb{R}^d$

$$u^\top C u$$

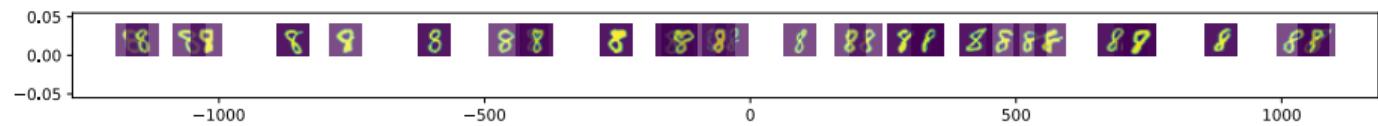
1. Compute the covariance matrix, C .
2. Compute the top eigenvector \vec{u} , of C .
"principal component"
3. For $i \in \{1, \dots, n\}$, create new feature:

$$z^{(i)} = \vec{u} \cdot \vec{x}^{(i)}$$

A Parting Example

- ▶ MNIST: 60,000 images in 784 dimensions
- ▶ Principal component: $\vec{u} \in \mathbb{R}^{784}$
- ▶ We can project an image in \mathbb{R}^{784} onto \vec{u} to get a single number representing the image

Example



DSC 140B

Representation Learning

Lecture 07 | Part 2

Dimensionality Reduction with $d \geq 2$

So far: PCA

- ▶ **Given:** data $\vec{x}^{(1)}, \dots, \vec{x}^{(n)} \in \mathbb{R}^d$
- ▶ **Map:** each data point $\vec{x}^{(i)}$ to a single feature, z_i .
 - ▶ Idea: maximize the variance of the new feature
- ▶ **PCA:** Let $z_i = \vec{x}^{(i)} \cdot \vec{u}$, where \vec{u} is top eigenvector of covariance matrix, C .

Now: More PCA

- ▶ **Given:** data $\vec{x}^{(1)}, \dots, \vec{x}^{(n)} \in \mathbb{R}^d$
- ▶ **Map:** each data point $\vec{x}^{(i)}$ to k new features,
 $\vec{z}^{(i)} = (z_1^{(i)}, \dots, z_k^{(i)})$.

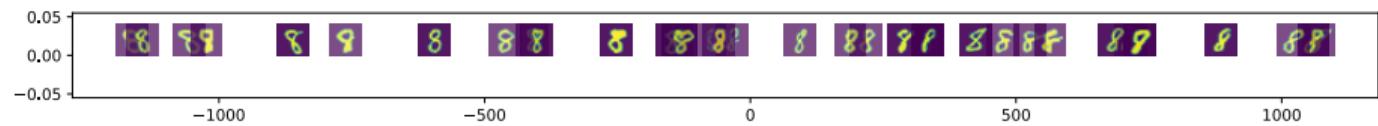
A Single Principal Component

- ▶ Recall: the **principal component** is the top eigenvector \vec{u} of the covariance matrix, C
- ▶ It is a unit vector in \mathbb{R}^d
- ▶ Make a new feature $z \in \mathbb{R}$ for point $\vec{x} \in \mathbb{R}^d$ by computing $z = \vec{x} \cdot \vec{u}$
- ▶ This is dimensionality reduction from $\mathbb{R}^d \rightarrow \mathbb{R}^1$

Example

- ▶ MNIST: 60,000 images in 784 dimensions
- ▶ Principal component: $\vec{u} \in \mathbb{R}^{784}$
- ▶ We can project an image in \mathbb{R}^{784} onto \vec{u} to get a single number representing the image

Example



Another Feature?

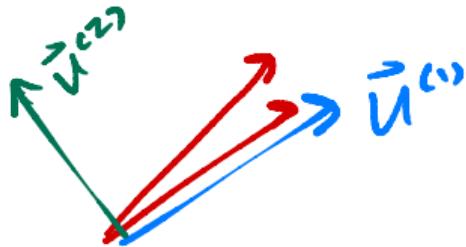
- ▶ Clearly, mapping from $\mathbb{R}^{784} \rightarrow \mathbb{R}^1$ loses a lot of information
- ▶ What about mapping from $\mathbb{R}^{784} \rightarrow \mathbb{R}^2$? \mathbb{R}^k ?

A Second Feature

- ▶ Our first feature is a mixture of features, with weights given by unit vector $\vec{u}^{(1)} = (u_1^{(1)}, u_2^{(1)}, \dots, u_d^{(1)})^T$.

$$z_1 = \vec{u}^{(1)} \cdot \vec{x} = u_1^{(1)}x_1 + \dots + u_d^{(1)}x_d$$

- ▶ To maximize variance, choose $\vec{u}^{(1)}$ to be top eigenvector of C .



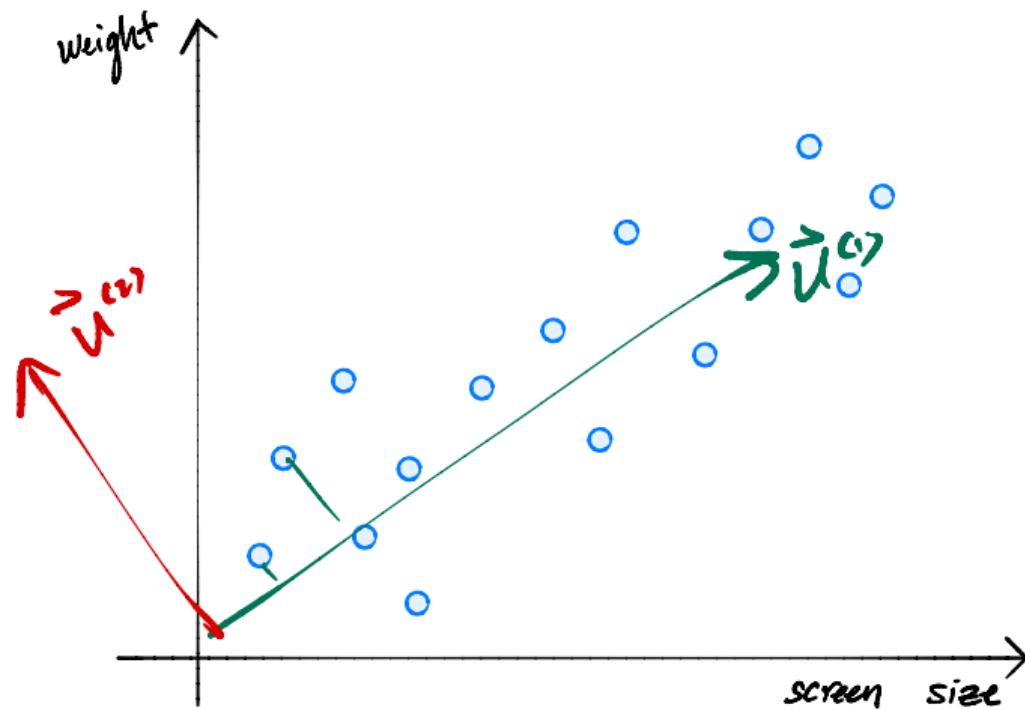
A Second Feature

- ▶ Make same assumption for second feature:

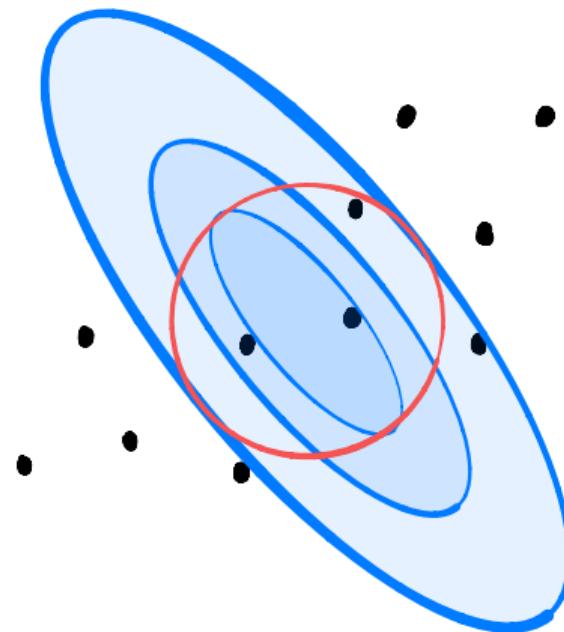
$$z_2 = \vec{u}^{(2)} \cdot \vec{x} = u_1^{(2)}x_1 + \dots + u_d^{(2)}x_d$$

- ▶ How do we choose $\vec{u}^{(2)}$?
- ▶ We should choose $\vec{u}^{(2)}$ to be **orthogonal** to $\vec{u}^{(1)}$.
 - ▶ No “redundancy”.

A Second Feature



A Second Feature

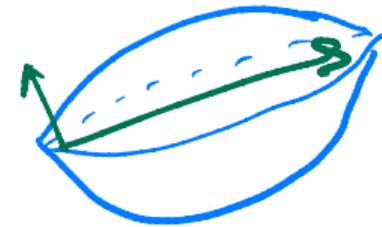


Intuition

- ▶ Claim: if \vec{u} and \vec{v} are eigenvectors of a symmetric matrix with distinct eigenvalues, they are orthogonal.
- ▶ We should choose $\vec{u}^{(2)}$ to be an **eigenvector** of the covariance matrix, C .
- ▶ The second eigenvector of C is called the **second principal component**.

A Second Principal Component

- ▶ Given a covariance matrix C .
- ▶ The principal component $\vec{u}^{(1)}$ is the top eigenvector of C .
 - ▶ Points in the direction of maximum variance.
- ▶ The *second* principal component $\vec{u}^{(2)}$ is the *second* eigenvector of C .
 - ▶ Out of all vectors orthogonal to the principal component, points in the direction of max variance.



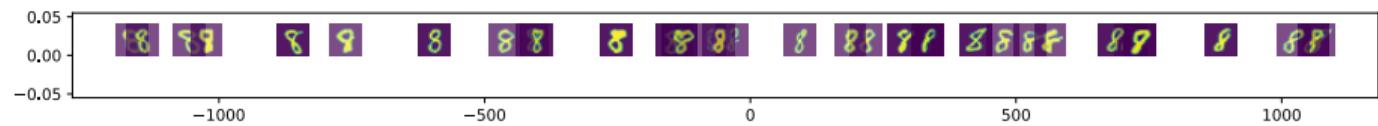
PCA: Two Components

- ▶ Given data $\{\vec{x}^{(1)}, \dots, \vec{x}^{(n)}\} \in \mathbb{R}^d$.
- ▶ Compute covariance matrix C , top two eigenvectors $\vec{u}^{(1)}$ and $\vec{u}^{(2)}$.
- ▶ For any vector $\vec{x} \in \mathbb{R}$, its new representation in \mathbb{R}^2 is $\vec{z} = (z_1, z_2)^T$, where:

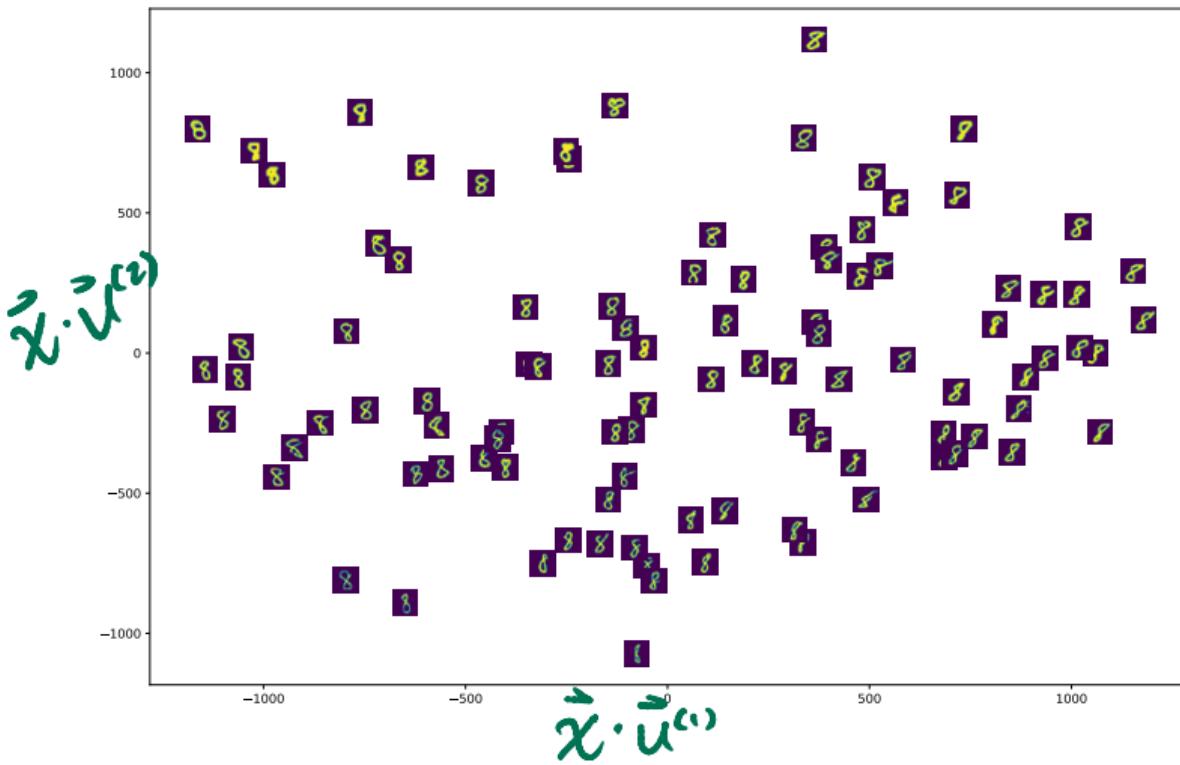
$$z_1 = \vec{x} \cdot \vec{u}^{(1)}$$

$$z_2 = \vec{x} \cdot \vec{u}^{(2)}$$

Example



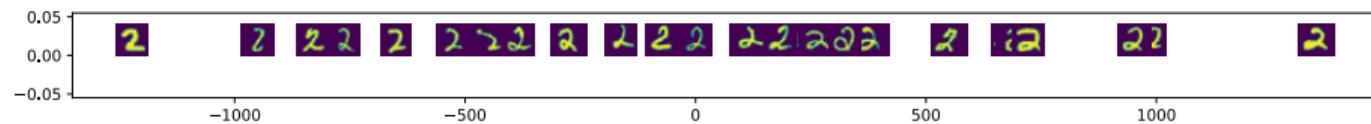
Example



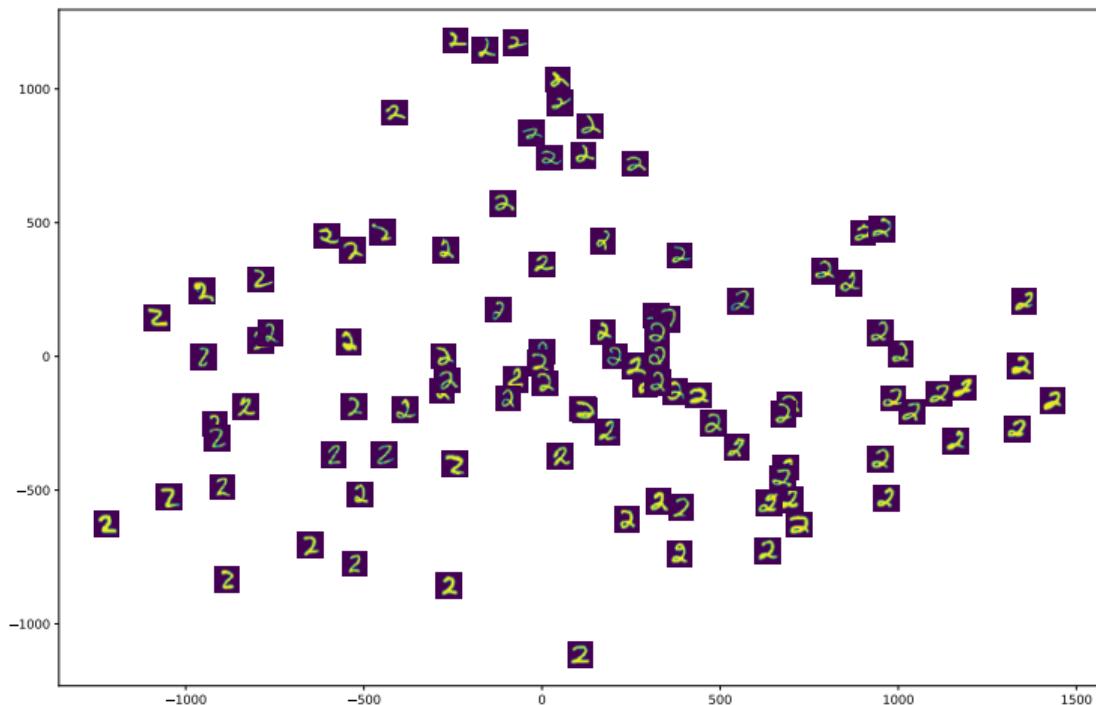
2

2

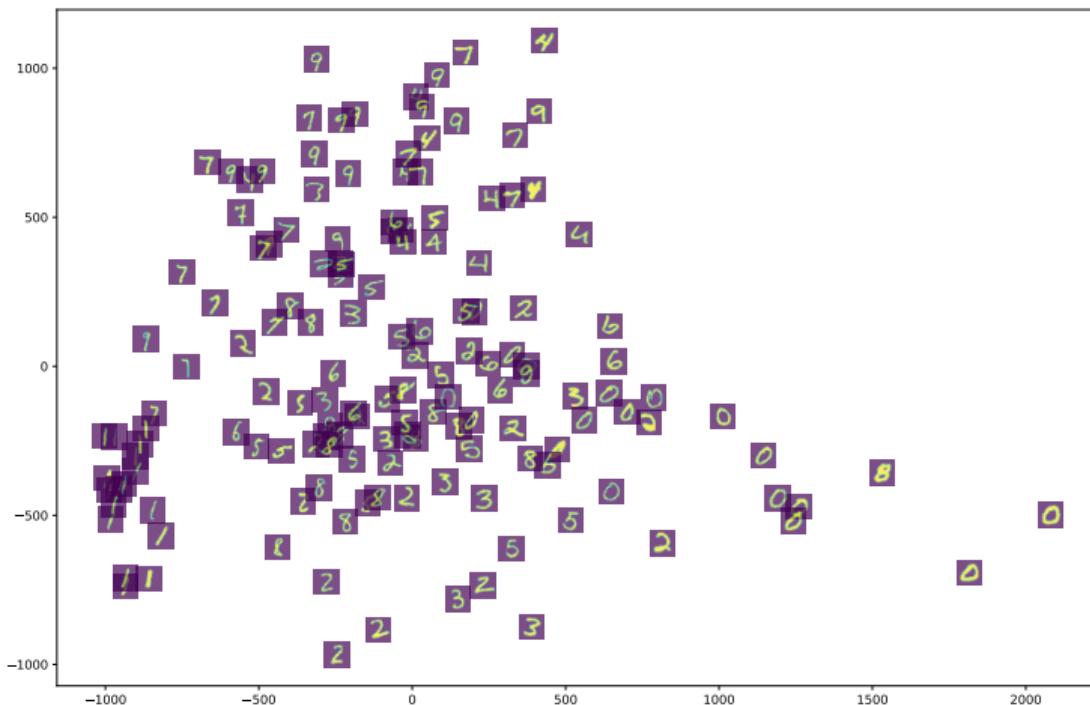
Example



Example



Example



PCA: k Components

- ▶ Given data $\{\vec{x}^{(1)}, \dots, \vec{x}^{(n)}\} \in \mathbb{R}^d$, number of components k .
- ▶ Compute covariance matrix C , top $k \leq d$ eigenvectors $\vec{u}^{(1)}$, $\vec{u}^{(2)}$, ..., $\vec{u}^{(k)}$.
- ▶ For any vector $\vec{x} \in \mathbb{R}$, its new representation in \mathbb{R}^k is $\vec{z} = (z_1, z_2, \dots, z_k)^T$, where:

$$z_1 = \vec{x} \cdot \vec{u}^{(1)}$$

$$z_2 = \vec{x} \cdot \vec{u}^{(2)}$$

$$\vdots$$

$$z_k = \vec{x} \cdot \vec{u}^{(k)}$$

Matrix Formulation

- ▶ Let X be the **data matrix** (n rows, d columns)
- ▶ Let U be matrix of the k eigenvectors as columns
(d rows, k columns)
- ▶ The new representation: $Z = XU$

DSC 140B

Representation Learning

Lecture 07 | Part 3

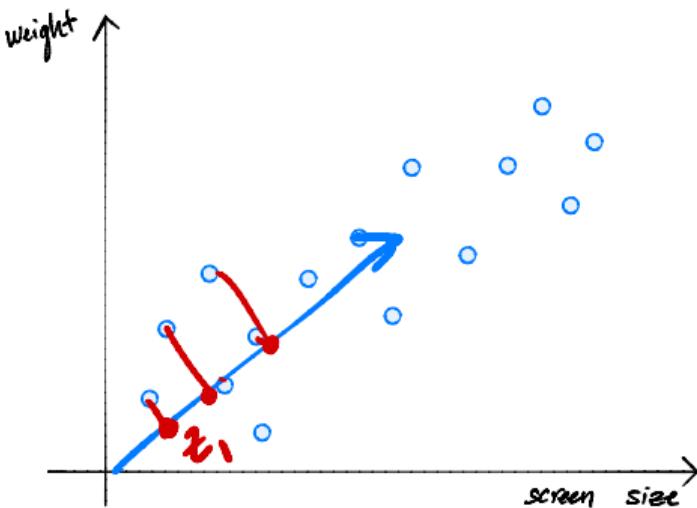
Reconstructions

Reconstructing Points

- ▶ PCA helps us reduce dimensionality from $\mathbb{R}^d \rightarrow \mathbb{R}^k$
- ▶ Suppose we have the “new” representation in \mathbb{R}^k .
- ▶ Can we “go back” to \mathbb{R}^d ?
- ▶ And why would we want to?

Back to \mathbb{R}^d

- ▶ Suppose new representation of \vec{x} is z .
- ▶ $z = \vec{x} \cdot \vec{u}^{(1)}$
- ▶ Idea: $\vec{x} \approx z\vec{u}^{(1)}$



Reconstructions

- ▶ Given a “new” representation of \vec{x} , $\vec{z} = (z_1, \dots, z_k) \in \mathbb{R}^k$
- ▶ And top k eigenvectors, $\vec{u}^{(1)}, \dots, \vec{u}^{(k)}$
- ▶ The **reconstruction** of \vec{x} is

$$z_1 \vec{u}^{(1)} + z_2 \vec{u}^{(2)} + \dots + z_k \vec{u}^{(k)} = U\vec{z}$$

Reconstruction Error

- ▶ The reconstruction *approximates* the original point, \vec{x} .

- ▶ The **reconstruction error** for a single point, \vec{x} :

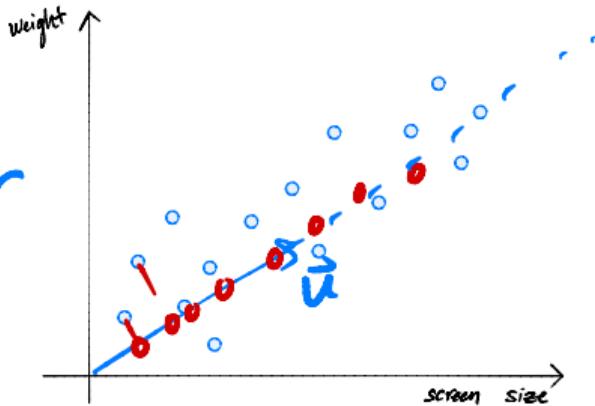
$$\vec{z}\vec{u}$$

$$\|\vec{x} - \vec{U}\vec{z}\|^2$$

reconstructed vector

- ▶ Total reconstruction error:

$$\sum_{i=1}^n \|\vec{x}^{(i)} - \vec{U}\vec{z}^{(i)}\|^2$$



DSC 140B

Representation Learning

Lecture 07 | Part 4

Interpreting PCA

Three Interpretations

- ▶ What is PCA doing?
- ▶ Three interpretations:
 1. Mazimizing variance
 2. Finding the best reconstruction
 3. Decorrelation

Recall: Matrix Formulation

- ▶ Given data matrix X .
- ▶ Compute new data matrix $Z = XU$.
- ▶ PCA: choose U to be matrix of eigenvectors of C .
- ▶ For now: suppose U can be anything – but columns should be orthonormal
 - ▶ Orthonormal = “not redundant”

$$Z = \begin{pmatrix} Z^{(1)} \\ Z^{(2)} \\ Z^{(3)} \\ \vdots \\ Z^{(n)} \end{pmatrix}$$

View #1: Maximizing Variance

► This was the view we used to derive PCA

- Define the **total variance** to be the sum of the variances of each column of Z .
- Claim: Choosing U to be top eigenvectors of C maximizes the total variance among all choices of orthonormal U .

Main Idea

PCA maximizes the total variance of the new data.
I.e., chooses the most “interesting” new features
which are not redundant.

View #2: Minimizing Reconstruction Error

- ▶ Recall: total reconstruction error

$$\sum_{i=1}^n \|\vec{x}^{(i)} - U\vec{z}^{(i)}\|^2$$

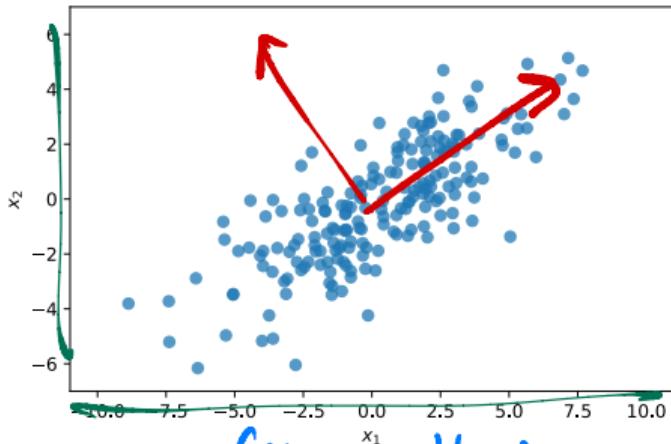
- ▶ Goal: minimize total reconstruction error.
- ▶ Claim: Choosing U to be top eigenvectors of C minimizes reconstruction error among all choices of orthonormal U

Main Idea

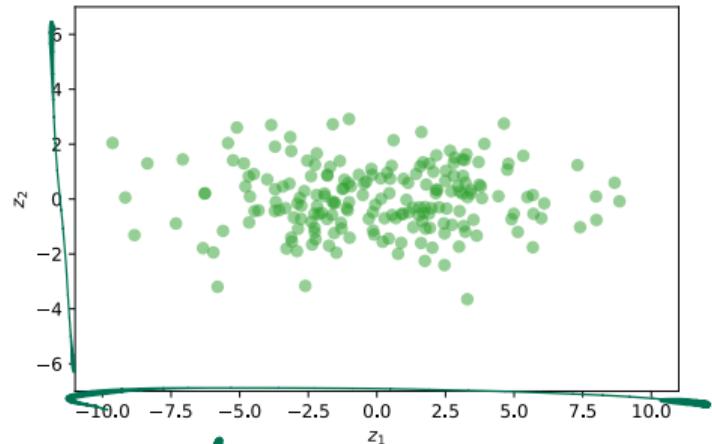
PCA minimizes the reconstruction error. It is the “best” projection of points onto a linear subspace of dimensionality k . When $k = d$, the reconstruction error is zero.

View #3: Decorrelation

- ▶ PCA has the effect of “decorrelating” the features.



$$\begin{pmatrix} 10 & 4 \\ 4 & 7 \end{pmatrix}$$



$$\begin{pmatrix} 20 & 0 \\ 0 & 10 \end{pmatrix}$$

Main Idea

PCA learns a new representation by rotating the data into a basis where the features are uncorrelated (not redundant). That is: the natural basis vectors are the principal directions (eigenvectors of the covariance matrix). PCA changes the basis to this natural basis.

DSC 140B

Representation Learning

Lecture 07 | Part 5

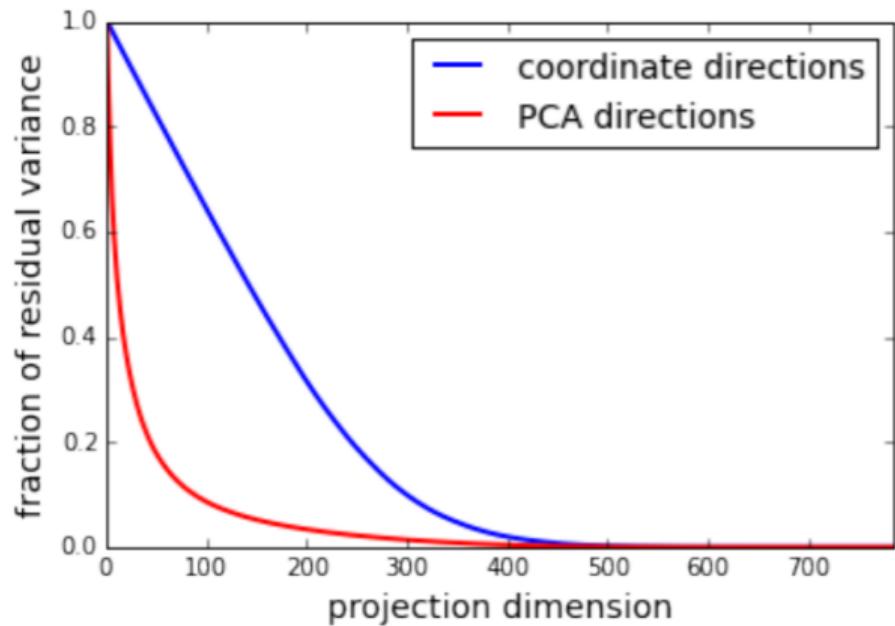
PCA in Practice

PCA in Practice

- ▶ PCA is often used in **preprocessing** before classifier is trained, etc.
- ▶ Must choose number of dimensions, k .
- ▶ One way: cross-validation.
- ▶ Another way: the elbow method.

Total Variance

- ▶ The **total variance** is the sum of the eigenvalues of the covariance matrix.
- ▶ Or, alternatively, sum of variances in each orthogonal basis direction.



Caution

- ▶ PCA's assumption: variance is interesting
- ▶ PCA is totally unsupervised
- ▶ The direction most meaningful for classification may not have large variance!

Demos

DSC 140B

Representation Learning

Lecture 08 | Part 1

Interpreting PCA

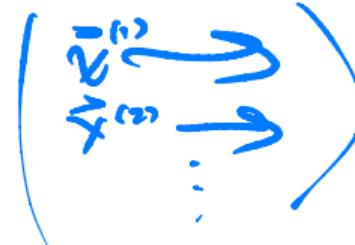
Three Interpretations

- ▶ What is PCA doing?
- ▶ Three interpretations:
 1. Mazimizing variance
 2. Finding the best reconstruction
 3. Decorrelation

$U\vec{x}$

Recall: Matrix Formulation

- ▶ Given data matrix X .
- ▶ Compute new data matrix $Z = XU$.
- ▶ PCA: choose U to be matrix of eigenvectors of C .
- ▶ For now: suppose U can be anything – but columns should be orthonormal
 - ▶ Orthonormal = “not redundant”



change of basis
matrix

View #1: Maximizing Variance

- ▶ This was the view we used to derive PCA
- ▶ Define the **total variance** to be the sum of the variances of each column of Z .
- ▶ Claim: Choosing U to be top eigenvectors of C maximizes the total variance among all choices of orthonormal U .

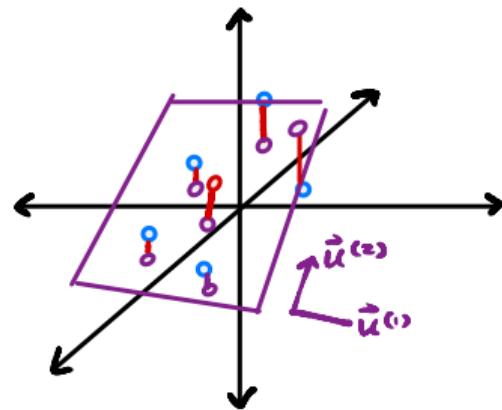
Main Idea

PCA maximizes the total variance of the new data.
I.e., chooses the most “interesting” new features
which are not redundant.

View #2: Minimizing Reconstruction Error

- Recall: total reconstruction error

$$\sum_{i=1}^n \|\vec{x}^{(i)} - U\vec{z}^{(i)}\|^2$$



- Goal: minimize total reconstruction error.
- Claim: Choosing U to be top eigenvectors of C minimizes reconstruction error among all choices of orthonormal U

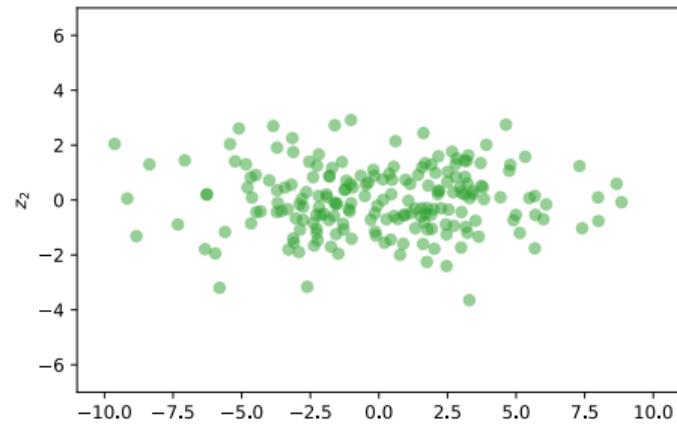
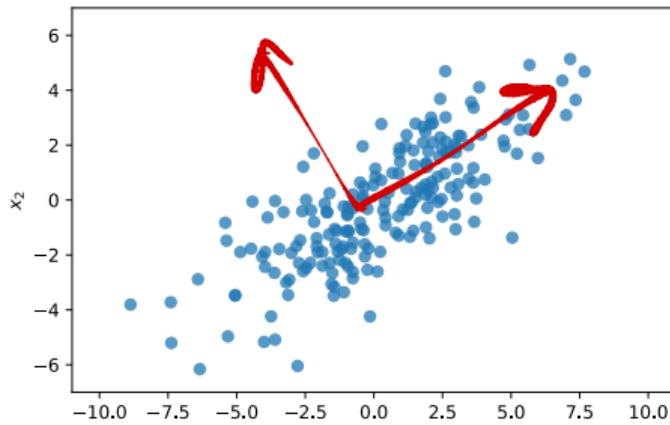
Main Idea

PCA minimizes the reconstruction error. It is the “best” projection of points onto a linear subspace of dimensionality k . When $k = d$, the reconstruction error is zero.

$$(AB)^T = B^T A^T$$

View #3: Decorrelation

- ▶ PCA has the effect of “decorrelating” the features.



$$C_z = \frac{1}{n} Z^T Z = \frac{1}{n} (XU)^T XU = \frac{1}{n} U^T X^T X U = \underbrace{U^T C_U}_{\text{diagonal}}$$
$$Z = XU$$

Main Idea

PCA learns a new representation by rotating the data into a basis where the features are uncorrelated (not redundant). That is: the natural basis vectors are the principal directions (eigenvectors of the covariance matrix). PCA changes the basis to this natural basis.

DSC 140B

Representation Learning

Lecture 08 | Part 2

PCA in Practice

PCA in Practice

- ▶ PCA is often used in **preprocessing** before classifier is trained, etc.
- ▶ Must choose number of dimensions, k .
- ▶ One way: cross-validation.
- ▶ Another way: the elbow method.

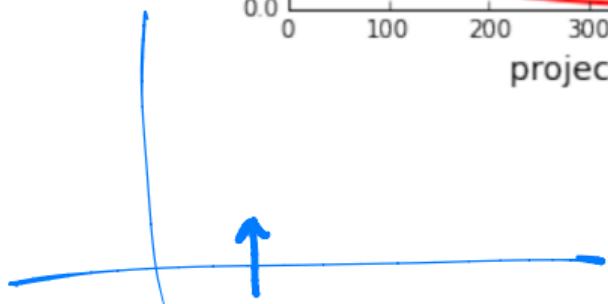
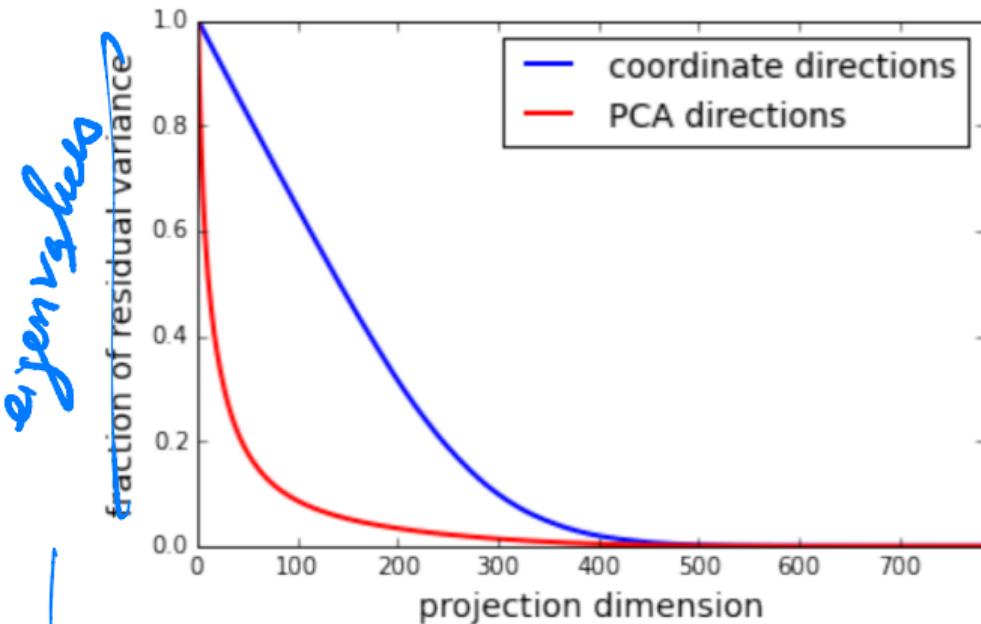
$\hat{u}^T C \hat{u}$ = variance in direction of \hat{u} .

$$\vec{v}^{(1)} \lambda_1$$

Total Variance

$$\begin{aligned}\vec{u}^{(1)^T} C \vec{u}^{(1)} \\ = \vec{u}^{(1)^T} \lambda_1 \vec{u}^{(1)} = \lambda_1\end{aligned}$$

- The **total variance** is the sum of the eigenvalues of the covariance matrix.
- Or, alternatively, sum of variances in each orthogonal basis direction.



Standardize the features?

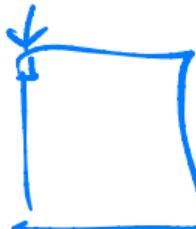
Km	Km
1570	.72
830	1.3

pixel1	pixel2
255	1
130	5
70	4

Caution

- ▶ PCA's assumption: variance is interesting

- ▶ PCA is totally unsupervised
- ▶ The direction most meaningful for classification may not have large variance!



DSC 140B

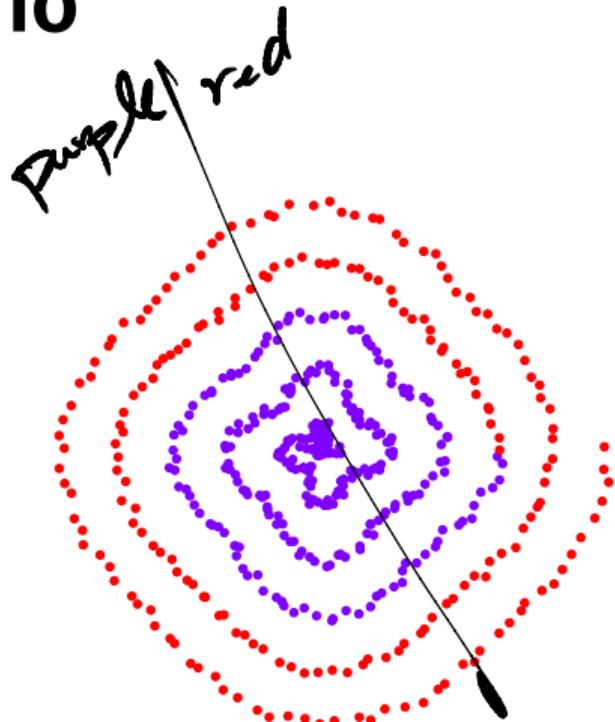
Representation Learning

Lecture 08 | Part 3

Nonlinear Dimensionality Reduction

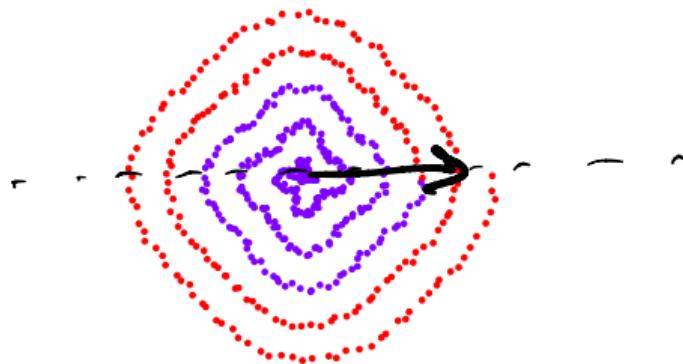
Scenario

- ▶ You want to train a classifier on this data.
- ▶ It would be easier if we could “unroll” the spiral.
- ▶ Data seems to be one-dimensional, even though in two dimensions.
- ▶ Dimensionality reduction?

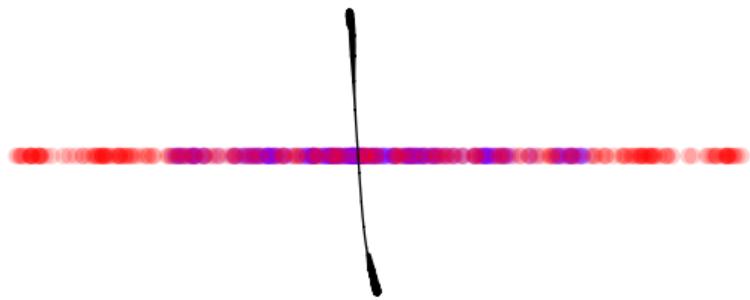


PCA?

- ▶ Does PCA work here?
- ▶ Try projecting onto one principal component.



No



PCA?

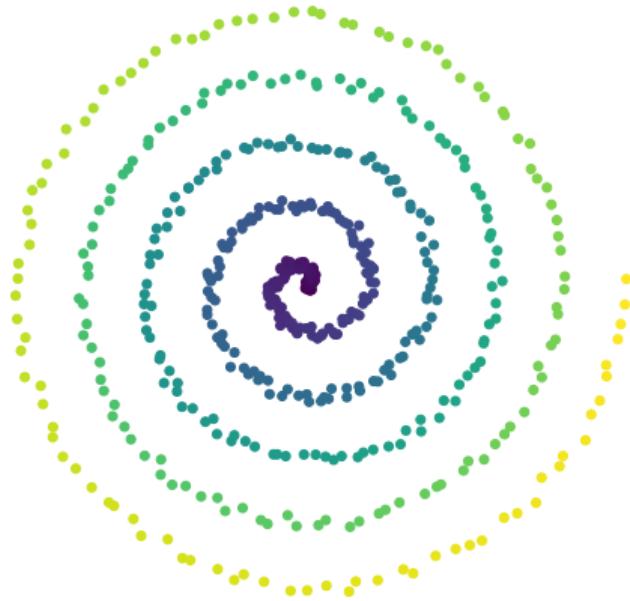
- ▶ PCA simply “rotates” the data.
- ▶ No amount of rotation will “unroll” the spiral.
- ▶ We need a fundamentally different approach that works for non-linear patterns.

Today

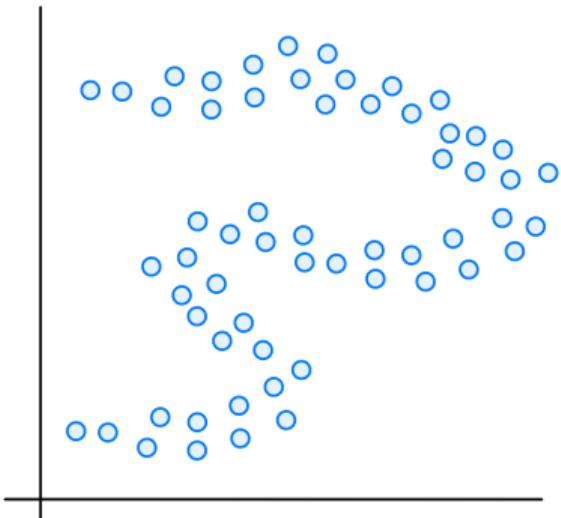
- ▶ Non-linear dimensionality reduction via
spectral embeddings.

Rethinking Dimensionality

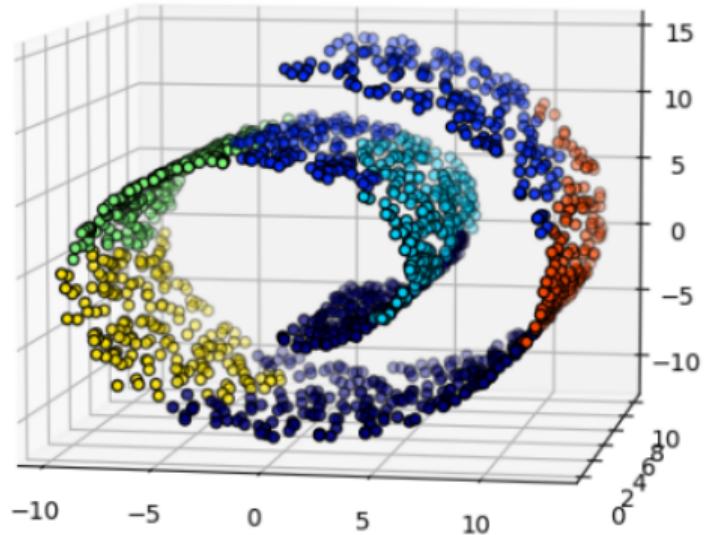
- ▶ Each point is an (x, y) coordinate in two dimensional space
- ▶ But the structure is one-dimensional
- ▶ Could (roughly) locate point using one number: distance from end.



Rethinking Dimensionality



Rethinking Dimensionality



Rethinking Dimensionality

- ▶ Informally: data expressed with d dimensions, but its *really* confined to k -dimensional region
- ▶ This region is called a **manifold**
- ▶ d is the **ambient** dimension
- ▶ k is the **intrinsic** dimension

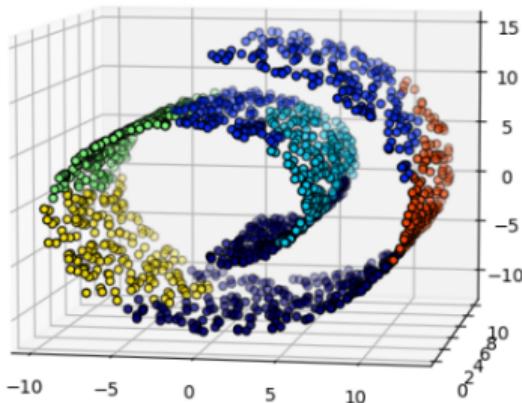
Example

- ▶ Ambient dimension: 2
- ▶ Intrinsic dimension: 1



Example

- ▶ Ambient dimension: 3
- ▶ Intrinsic dimension: 2



Example

- ▶ Ambient dimension: 3
- ▶ Intrinsic dimension: 2



Manifold Learning

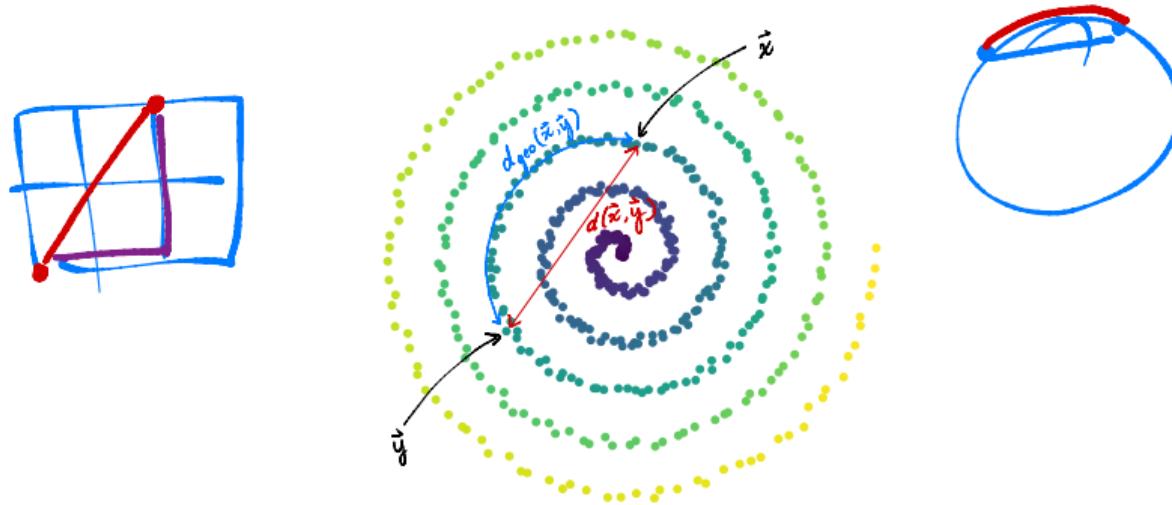
- ▶ **Given:** data in high dimensions
- ▶ **Recover:** the low-dimensional manifold

Types of Manifolds

- ▶ Manifolds can be linear
 - ▶ E.g., linear subspaces – hyperplanes
 - ▶ Learned by PCA
- ▶ Can also be non-linear (locally linear)
 - ▶ Example: the spiral data
 - ▶ Learned by **Laplacian eigenmaps**, among others

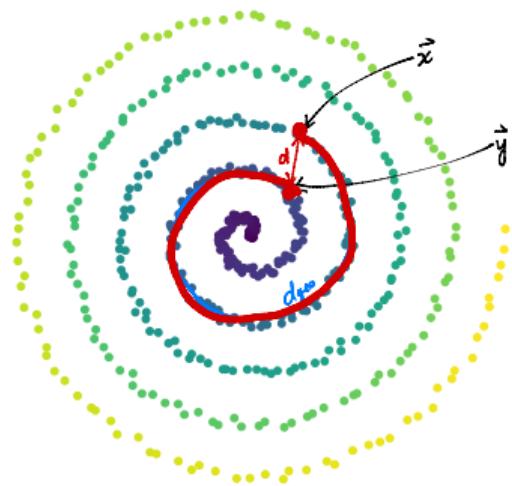
Euclidean vs. Geodesic Distances

- ▶ **Euclidean distance:** the “straight-line” distance
- ▶ **Geodesic distance:** the distance along the manifold



Euclidean vs. Geodesic Distances

- ▶ **Euclidean distance:** the “straight-line” distance
- ▶ **Geodesic distance:** the distance along the manifold



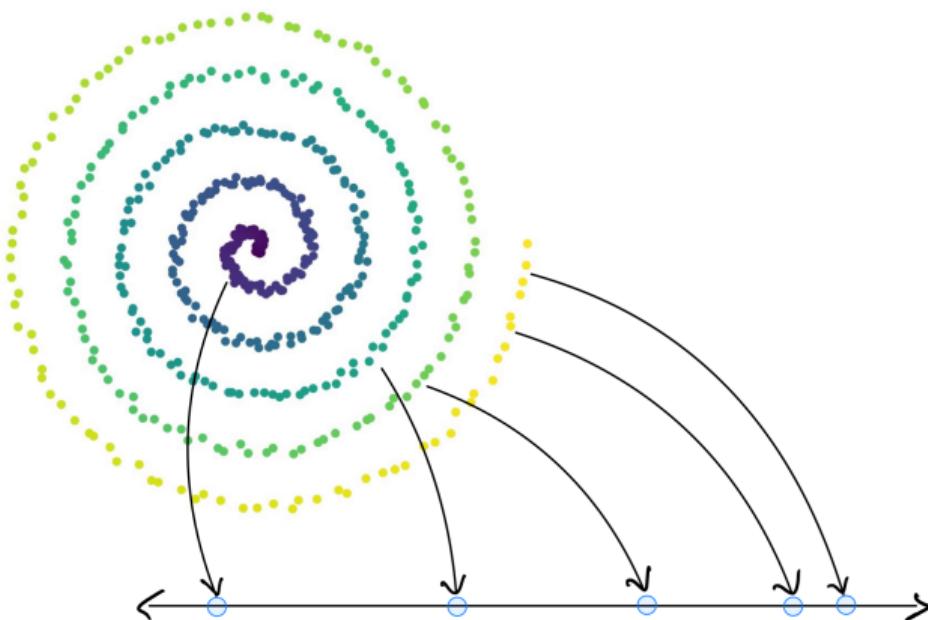
Euclidean vs. Geodesic Distances

- ▶ If data is close to a linear manifold, geodesic \approx Euclidean
- ▶ Otherwise, can be very different

Non-Linear Dimensionality Reduction

- ▶ **Goal:** Map points in \mathbb{R}^d to \mathbb{R}^k
- ▶ **Such that:** if \vec{x} and \vec{y} are close in **geodesic** distance in \mathbb{R}^d , they are close in **Euclidean** distance in \mathbb{R}^k

Embeddings



DSC 140B

Representation Learning

Lecture 08 | Part 4

Embedding Similarities

Similar Netflix Users

- ▶ Suppose you are a data scientist at Netflix
- ▶ You're given an $n \times n$ **similarity matrix** W of users
 - ▶ entry (i, j) tells you how *similar* user i and user j are
 - ▶ 1 means "very similar", 0 means "not at all"
- ▶ Goal: visualize to find patterns

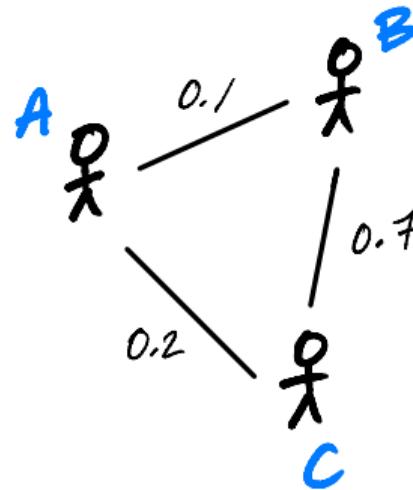
Idea

- ▶ We like scatter plots. Can we make one?
- ▶ Users are **not** vectors / points!
- ▶ They are nodes in a **similarity graph**

Similarity Graphs

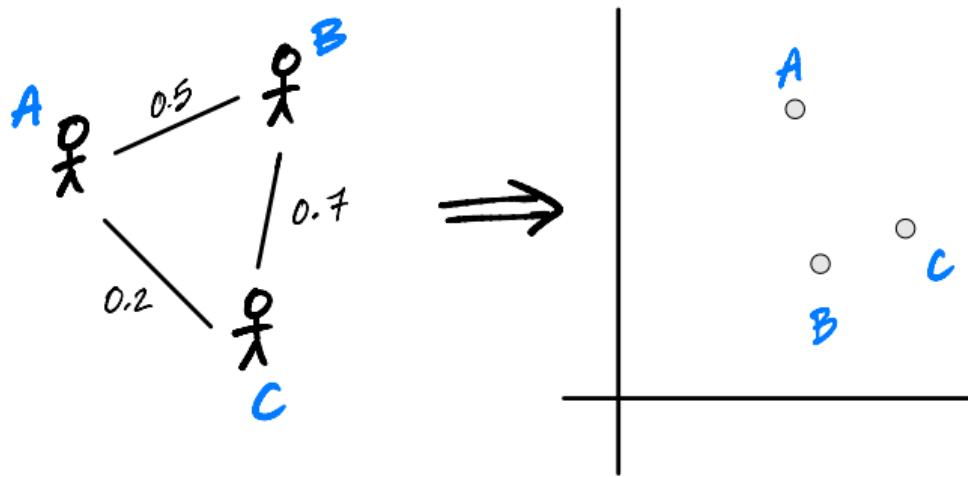
- ▶ Similarity matrices can be thought of as weighted graphs, and *vice versa*.

$$\begin{matrix} & \begin{matrix} A & B & C \end{matrix} \\ \begin{matrix} A \\ B \\ C \end{matrix} & \left(\begin{array}{ccc} 1 & 0.1 & 0.2 \\ 0.1 & 1 & 0.7 \\ 0.2 & 0.7 & 1 \end{array} \right) \end{matrix}$$



Goal

- ▶ **Embed** nodes of a similarity graph as points.
- ▶ Similar nodes should map to nearby points.



Today

- ▶ We will design a graph embedding approach:
 - ▶ **Spectral embeddings** via **Laplacian eigenmaps**

More Formally

- ▶ **Given:**
 - ▶ A **similarity graph** with n nodes
 - ▶ a number of dimensions, k
- ▶ **Compute:** an **embedding** of the n points into \mathbb{R}^k so that similar objects are placed nearby

To Start

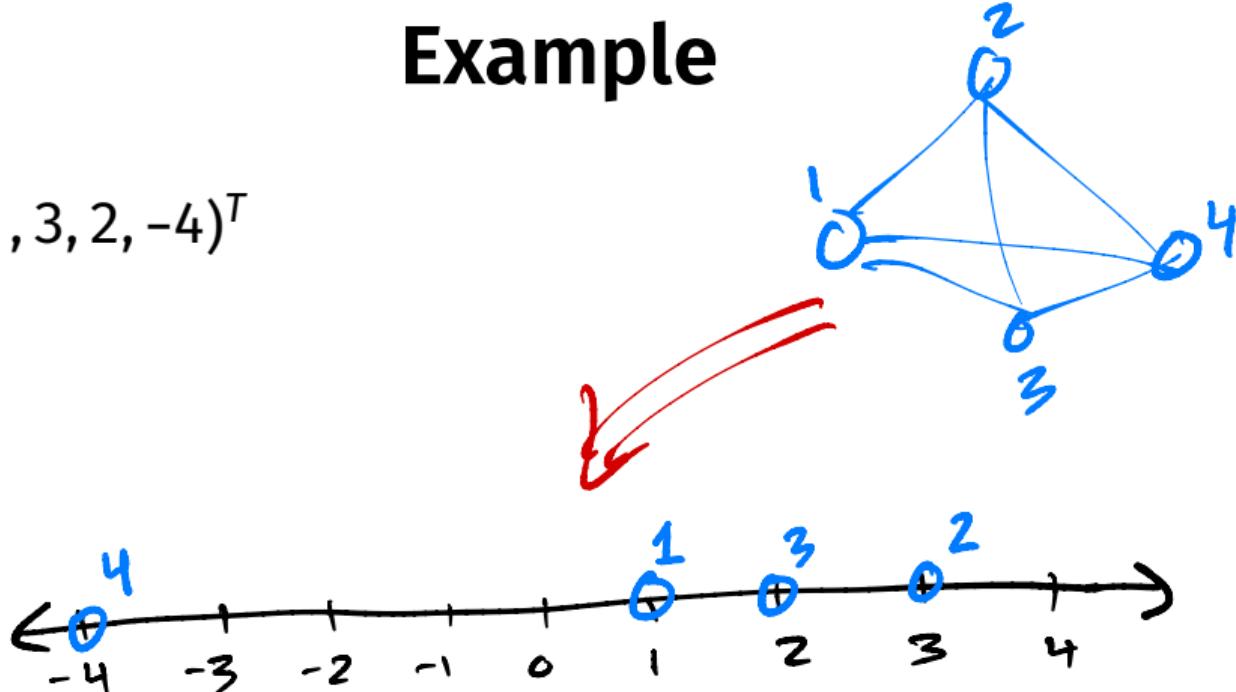
- ▶ **Given:**
 - ▶ A **similarity graph** with n nodes
- ▶ **Compute:** an **embedding** of the n points into \mathbb{R}^1 so that similar objects are placed nearby

Vectors as Embeddings into \mathbb{R}^1

- ▶ Suppose we have n nodes (objects) to embed
- ▶ Assume they are numbered $1, 2, \dots, n$
- ▶ Let $f_1, f_2, \dots, f_n \in \mathbb{R}$ be the embeddings
- ▶ We can pack them all into a vector: \vec{f} .
- ▶ Goal: find a good set of embeddings, \vec{f} .

Example

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

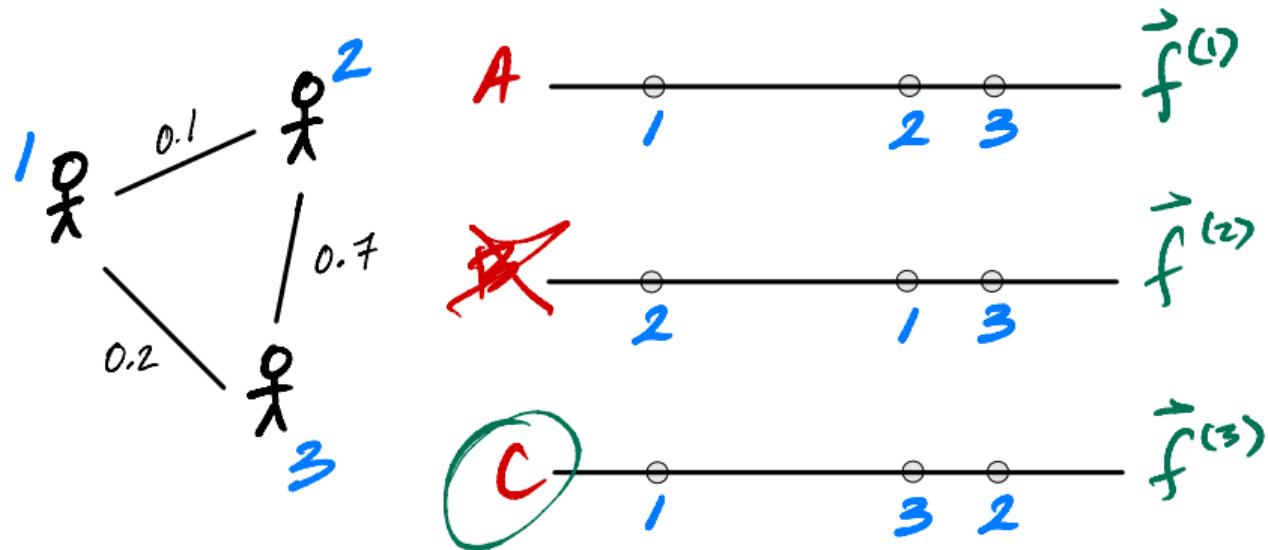


An Optimization Problem

- ▶ We'll turn it into an optimization problem:
- ▶ **Step 1:** Design a cost function quantifying how good a particular embedding \vec{f} is
- ▶ **Step 2:** Minimize the cost

Example

- ▶ Which is the best embedding?



Cost Function for Embeddings

- ▶ Idea: cost is low if similar points are close
- ▶ Here is one approach:

$$\text{Cost}(\vec{f}) = \sum_{i=1}^n \sum_{j=1}^n w_{ij}(f_i - f_j)^2$$

- ▶ where w_{ij} is the weight between i and j .

Interpreting the Cost

$$\text{Cost}(\vec{f}) = \sum_{i=1}^n \sum_{j=1}^n w_{ij}(f_i - f_j)^2$$

- ▶ If $w_{ij} \approx 0$, that pair can be placed very far apart without increasing cost
- ▶ If $w_{ij} \approx 1$, the pair should be placed close together in order to have small cost.

Exercise

Do you see a problem with the cost function?

$$\text{Cost}(\vec{f}) = \sum_{i=1}^n \sum_{j=1}^n w_{ij}(f_i - f_j)^2$$

Hint: what embedding \vec{f} minimizes it?

Problem

- ▶ The cost is **always** minimized by taking $\vec{f} = 0$.
- ▶ This is a “**trivial**” solution. Not useful.
- ▶ **Fix:** require $\|\vec{f}\| = 1$
 - ▶ Really, any number would work. 1 is convenient.

Exercise

Do you see **another** problem with the cost function, even if we require \vec{f} to be a unit vector?

$$\text{Cost}(\vec{f}) = \sum_{i=1}^n \sum_{j=1}^n w_{ij}(f_i - f_j)^2$$

Hint: what other choice of \vec{f} will **always** make this zero?

Problem

- ▶ The cost is **always** minimized by taking
$$\vec{f} = \frac{1}{\sqrt{n}}(1, 1, \dots, 1)^T.$$
- ▶ This is a “**trivial**” solution. Again, not useful.
- ▶ **Fix:** require \vec{f} to be orthogonal to $(1, 1, \dots, 1)^T$.
 - ▶ Written: $\vec{f} \perp (1, 1, \dots, 1)^T$
 - ▶ Ensures that solution is not close to trivial solution
 - ▶ Might seem strange, but it will work!

The New Optimization Problem

- ▶ **Given:** an $n \times n$ similarity matrix W
- ▶ **Compute:** embedding vector \vec{f} minimizing

$$\text{Cost}(\vec{f}) = \sum_{i=1}^n \sum_{j=1}^n w_{ij}(f_i - f_j)^2$$

subject to $\|\vec{f}\| = 1$ and $\vec{f} \perp (1, 1, \dots, 1)^T$

How?

- ▶ This looks difficult.
- ▶ Let's write it in matrix form.
- ▶ We'll see that it is actually (hopefully) familiar.

How?

- ▶ This looks difficult.
- ▶ Let's write it in matrix form.
- ▶ We'll see that it is actually (hopefully) familiar.

DSC 140B

Representation Learning

Lecture 09 | Part 1

Embedding Similarities

Similar Netflix Users

- ▶ Suppose you are a data scientist at Netflix
- ▶ You're given an $n \times n$ **similarity matrix** W of users
 - ▶ entry (i, j) tells you how *similar* user i and user j are
 - ▶ 1 means "very similar", 0 means "not at all"
- ▶ Goal: visualize to find patterns

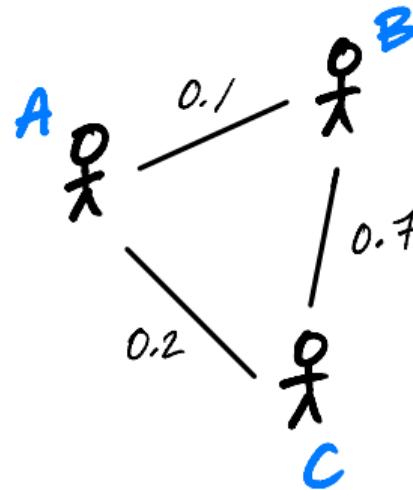
Idea

- ▶ We like scatter plots. Can we make one?
- ▶ Users are **not** vectors / points!
- ▶ They are nodes in a **similarity graph**

Similarity Graphs

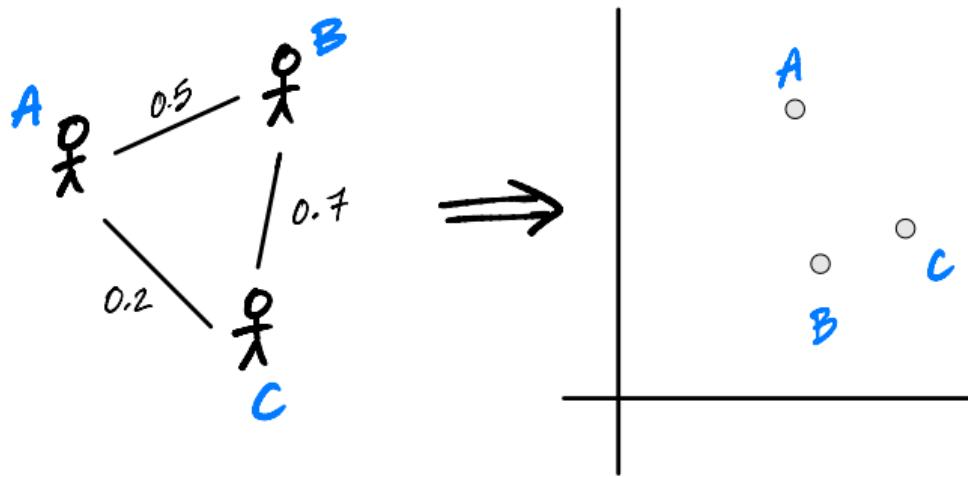
- ▶ Similarity matrices can be thought of as weighted graphs, and *vice versa*.

$$\begin{matrix} & \begin{matrix} A & B & C \end{matrix} \\ \begin{matrix} A \\ B \\ C \end{matrix} & \left(\begin{array}{ccc} 1 & 0.1 & 0.2 \\ 0.1 & 1 & 0.7 \\ 0.2 & 0.7 & 1 \end{array} \right) \end{matrix}$$



Goal

- ▶ **Embed** nodes of a similarity graph as points.
- ▶ Similar nodes should map to nearby points.



Today

- ▶ We will design a graph embedding approach:
 - ▶ **Spectral embeddings** via **Laplacian eigenmaps**

More Formally

- ▶ **Given:**
 - ▶ A **similarity graph** with n nodes
 - ▶ a number of dimensions, k
- ▶ **Compute:** an **embedding** of the n points into \mathbb{R}^k so that similar objects are placed nearby

To Start

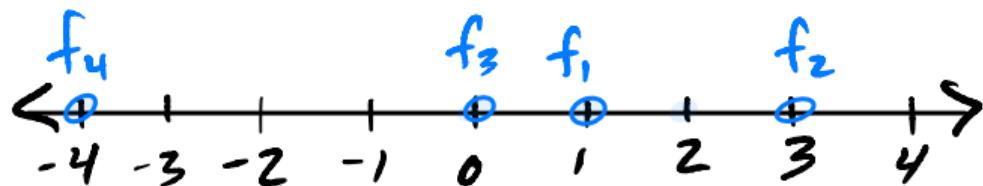
- ▶ **Given:**
 - ▶ A **similarity graph** with n nodes
- ▶ **Compute:** an **embedding** of the n points into \mathbb{R}^1 so that similar objects are placed nearby

Vectors as Embeddings into \mathbb{R}^1

- ▶ Suppose we have n nodes (objects) to embed
- ▶ Assume they are numbered $1, 2, \dots, n$
- ▶ Let $f_1, f_2, \dots, f_n \in \mathbb{R}$ be the embeddings
- ▶ We can pack them all into a vector: \vec{f} .
- ▶ Goal: find a good set of embeddings, \vec{f} .

Example

$$\vec{f} = (1, 3, 0, -4)^T$$

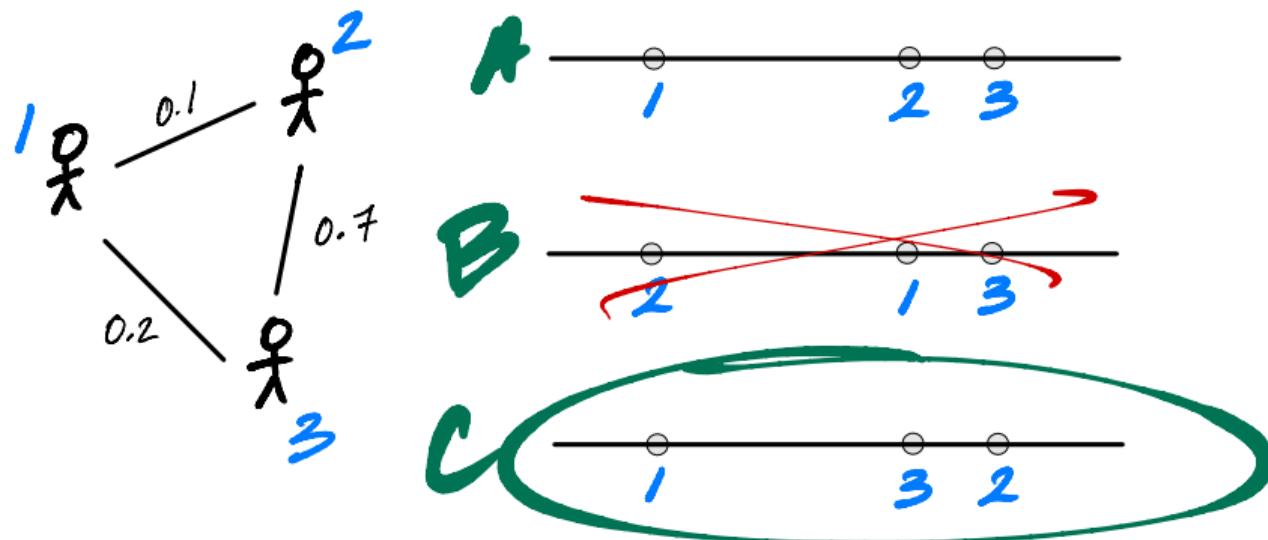


An Optimization Problem

- ▶ We'll turn it into an optimization problem:
- ▶ **Step 1:** Design a cost function quantifying how good a particular embedding \vec{f} is
- ▶ **Step 2:** Minimize the cost

Example

- ▶ Which is the best embedding?



Cost Function for Embeddings

- ▶ Idea: cost is low if similar points are close
- ▶ Here is one approach:

$$\text{Cost}(\vec{f}) = \sum_{i=1}^n \sum_{j=1}^n w_{ij}(f_i - f_j)^2$$

- ▶ where w_{ij} is the weight between i and j .

Interpreting the Cost

$$\text{Cost}(\vec{f}) = \sum_{i=1}^n \sum_{j=1}^n w_{ij}(f_i - f_j)^2$$

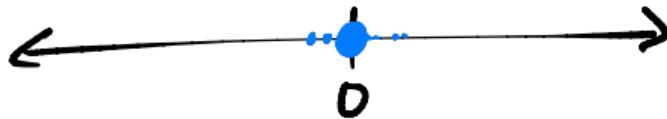
- ▶ If $w_{ij} \approx 0$, that pair can be placed very far apart without increasing cost
- ▶ If $w_{ij} \approx 1$, the pair should be placed close together in order to have small cost.

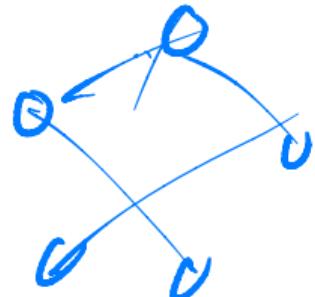
Exercise

Do you see a problem with the cost function?

$$\min \text{Cost}(\vec{f}) = \sum_{i=1}^n \sum_{j=1}^n w_{ij}(f_i - f_j)^2$$

Hint: what embedding \vec{f} minimizes it?





Problem

$0, 0, 0,$

- ▶ The cost is **always** minimized by taking $\vec{f} = 0$.
- ▶ This is a “**trivial**” solution. Not useful.
- ▶ **Fix:** require $\|\vec{f}\| = 1$
 - ▶ Really, any number would work. 1 is convenient.

Exercise

Do you see **another** problem with the cost function, even if we require \vec{f} to be a unit vector?

$$\text{Cost}(\vec{f}) = \sum_{i=1}^n \sum_{j=1}^n w_{ij}(f_i - f_j)^2$$

$$\|\vec{f}\|$$

Hint: what other choice of \vec{f} will **always** make this zero?



Problem

- ▶ The cost is **always** minimized by taking
$$\vec{f} = \frac{1}{\sqrt{n}}(1, 1, \dots, 1)^T.$$
- ▶ This is a “**trivial**” solution. Again, not useful.
- ▶ **Fix:** require \vec{f} to be orthogonal to $(1, 1, \dots, 1)^T$.
 - ▶ Written: $\vec{f} \perp (1, 1, \dots, 1)^T$
 - ▶ Ensures that solution is not close to trivial solution
 - ▶ Might seem strange, but it will work!

The New Optimization Problem

- ▶ **Given:** an $n \times n$ similarity matrix W
- ▶ **Compute:** embedding vector \vec{f} minimizing

$$\text{Cost}(\vec{f}) = \sum_{i=1}^n \sum_{j=1}^n w_{ij}(f_i - f_j)^2$$

subject to $\|\vec{f}\| = 1$ and $\vec{f} \perp (1, 1, \dots, 1)^T$

How?

- ▶ This looks difficult.
- ▶ Let's write it in matrix form.
- ▶ We'll see that it is actually (hopefully) familiar.

DSC 140B

Representation Learning

Lecture 09 | Part 2

The Graph Laplacian

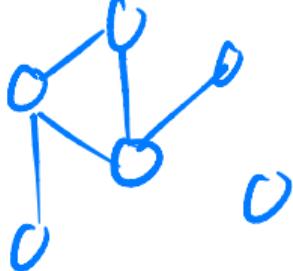
The Problem

- ▶ **Compute:** embedding vector \vec{f} minimizing

$$\text{Cost}(\vec{f}) = \sum_{i=1}^n \sum_{j=1}^n w_{ij}(f_i - f_j)^2$$

subject to $\|\vec{f}\| = 1$ and $\vec{f} \perp (1, 1, \dots, 1)^T$

- ▶ Now: write the cost function as a matrix expression.

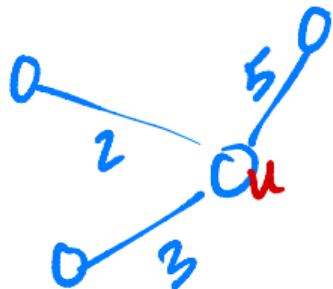


The Degree Matrix

- ▶ Recall: in an unweighted graph, the degree of node i equals number of neighbors.
- ▶ Equivalently (where A is the adjacency matrix):

$$\text{degree}(i) = \sum_{j=1}^n A_{ij}$$

- ▶ Since $A_{ij} = 1$ only if j is a neighbor of i



$$\deg(u) = 3 + 2 + 5 = 10$$

The Degree Matrix

- ▶ In a weighted graph, define **degree** of node i similarly:

$$\text{degree}(i) = \sum_{j=1}^n w_{ij}$$

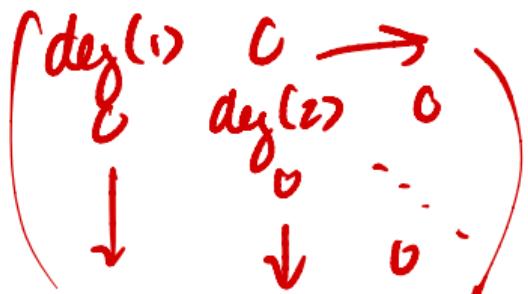
- ▶ That is, it is the total weight of all neighbors.

The Degree Matrix

- The **degree matrix** D of a weighted graph is the diagonal matrix where entry (i, i) is given by:

$$d_{ii} = \text{degree}(i)$$

$$= \sum_{j=1}^n w_{ij}$$



The Graph Laplacian

- ▶ Define $L = D - W$
 - ▶ D is the degree matrix
 - ▶ W is the similarity matrix (weighted adjacency)
- ▶ L is called the **Graph Laplacian** matrix.
- ▶ It is a very useful object

Very Important Fact

- ▶ Claim:

$$\text{Cost}(\vec{f}) = \sum_{i=1}^n \sum_{j=1}^n w_{ij}(f_i - f_j)^2 = \frac{1}{2} \vec{f}^T L \vec{f}$$

$$L = D - W$$

- ▶ Proof: expand both sides

Proof

DSC 140B

Representation Learning

Lecture 09 | Part 3

Solving the Optimization Problem

A New Formulation

- ▶ **Given:** an $n \times n$ similarity matrix W
- ▶ **Compute:** embedding vector \vec{f} **minimizing**

$$\text{Cost}(\vec{f}) = \frac{1}{2} \vec{f}^T L \vec{f}$$

subject to $\|\vec{f}\| = 1$ and $\vec{f} \perp (1, 1, \dots, 1)^T$

- ▶ This might sound familiar...

Recall: PCA

- ▶ **Given:** a $d \times d$ covariance matrix C
- ▶ **Find:** vector \vec{u} **maximizing** the variance in the direction of \vec{u} :

$$\vec{u}^T C \vec{u}$$

subject to $\|\vec{u}\| = 1$.

- ▶ **Solution:** take $\vec{u} = \text{top eigenvector of } C$

A New Formulation

- ▶ Forget about orthogonality constraint for now.
- ▶ **Compute:** embedding vector \vec{f} minimizing

$$\text{Cost}(\vec{f}) = \frac{1}{2} \vec{f}^T L \vec{f}$$

subject to $\|\vec{f}\| = 1$.

*is symmetric
(as long as W
is symm)*

- ▶ **Solution:** the bottom eigenvector of L .
 - ▶ That is, eigenvector with smallest eigenvalue.

Claim

- ▶ The bottom eigenvector is $\vec{f} = \frac{1}{\sqrt{n}}(1, 1, \dots, 1)^T$
- ▶ It has associated eigenvalue of 0.
- ▶ That is, $L\vec{f} = 0\vec{f} = \vec{0}$

Spectral¹ Theorem

Theorem

If A is a symmetric matrix, eigenvectors of A with distinct eigenvalues are orthogonal to one another.

¹“Spectral” not in the sense of specters (ghosts), but because the eigenvalues of a transformation form the “spectrum”

The Fix

- ▶ Remember: we wanted \vec{f} to be orthogonal to $\frac{1}{\sqrt{n}}(1, 1, \dots, 1)^T$.
 - ▶ i.e., should be orthogonal to bottom eigenvector of L .
- ▶ Fix: take \vec{f} to be eigenvector of L with smallest eigenvalue $\neq 0$.
- ▶ Will be $\perp \frac{1}{\sqrt{n}}(1, 1, \dots, 1)^T$ by the **spectral theorem**.

Spectral Embeddings: Problem

- ▶ Given: **similarity graph** with n nodes
- ▶ Compute: an **embedding** of the n points into \mathbb{R}^1 so that similar objects are placed nearby
- ▶ Formally: find embedding vector \vec{f} minimizing

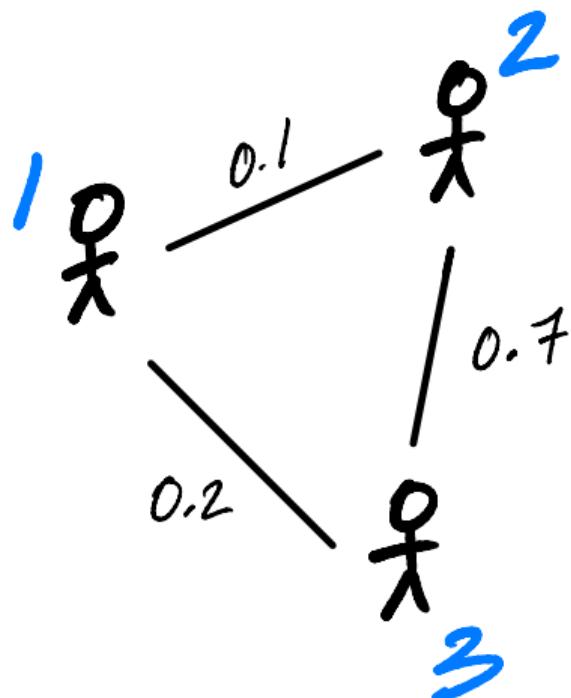
$$\text{Cost}(\vec{f}) = \sum_{i=1}^n \sum_{j=1}^n w_{ij}(f_i - f_j)^2 = \frac{1}{2} \vec{f}^T L \vec{f}$$

subject to $\|\vec{f}\| = 1$ and $\vec{f} \perp (1, 1, \dots, 1)^T$

Spectral Embeddings: Solution

- ▶ Form the **graph Laplacian** matrix, $L = D - W$
- ▶ Choose \vec{f} be an eigenvector of L with smallest eigenvalue > 0
- ▶ This is the embedding!

Example



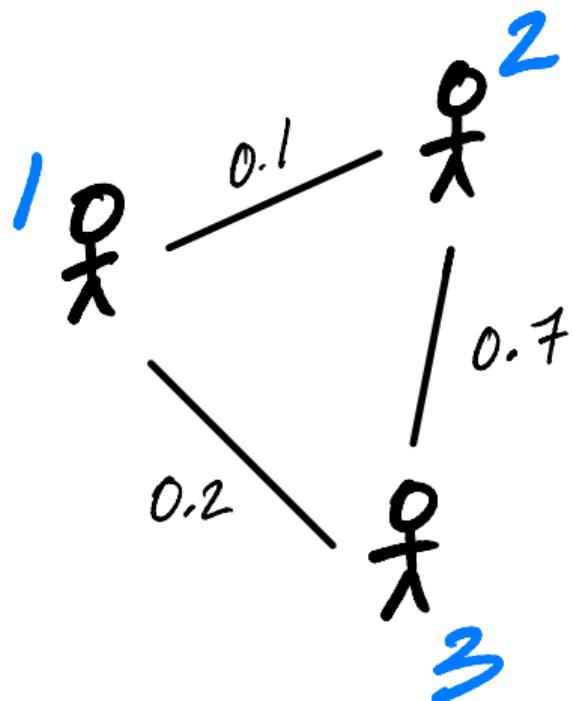
```
W = np.array([
    [1, 0.1, 0.2],
    [0.1, 1, 0.7],
    [0.2, 0.7, 1]
])

D = np.diag(W.sum(axis=1))
L = D - W

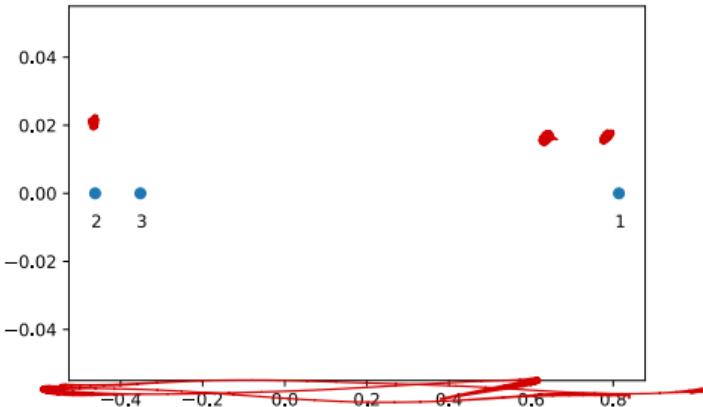
vals, vecs = np.linalg.eigh(L)

f = vecs[:, 1]
```

Example



$$\vec{f} = (\cdot, \cdot, \cdot)^T$$

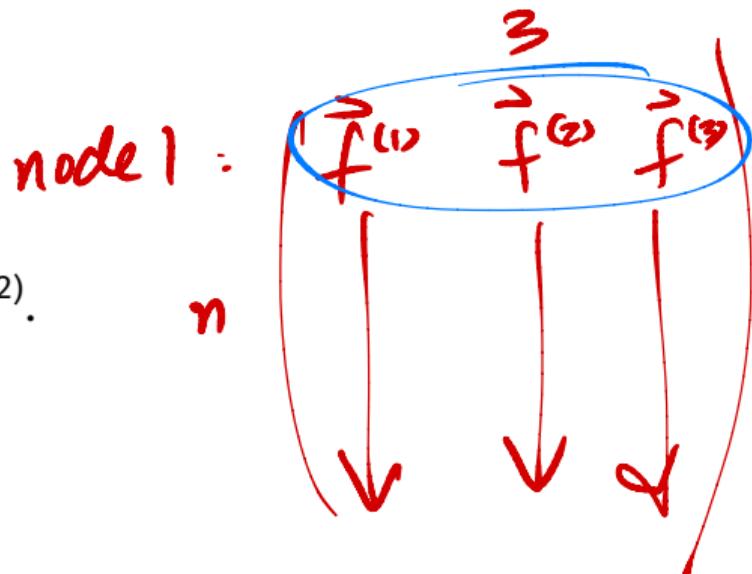


Embedding into \mathbb{R}^k

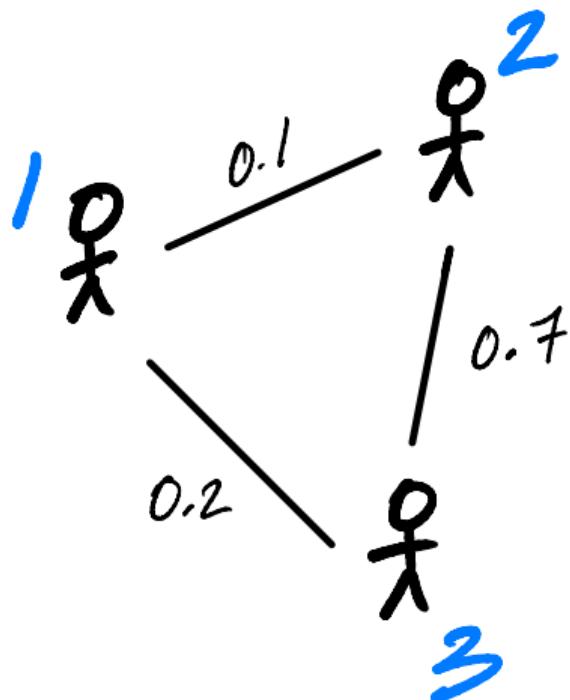
- ▶ This embeds nodes into \mathbb{R}^1 .
- ▶ What about embedding into \mathbb{R}^k ?
- ▶ Natural extension: find bottom k eigenvectors with eigenvalues > 0

New Coordinates

- ▶ With k eigenvectors $\vec{f}^{(1)}, \vec{f}^{(2)}, \dots, \vec{f}^{(k)}$, each node is mapped to a point in \mathbb{R}^k .
- ▶ Consider node i .
 - ▶ First new coordinate is $\vec{f}_i^{(1)}$.
 - ▶ Second new coordinate is $\vec{f}_i^{(2)}$.
 - ▶ Third new coordinate is $\vec{f}_i^{(3)}$.
 - ▶ :



Example



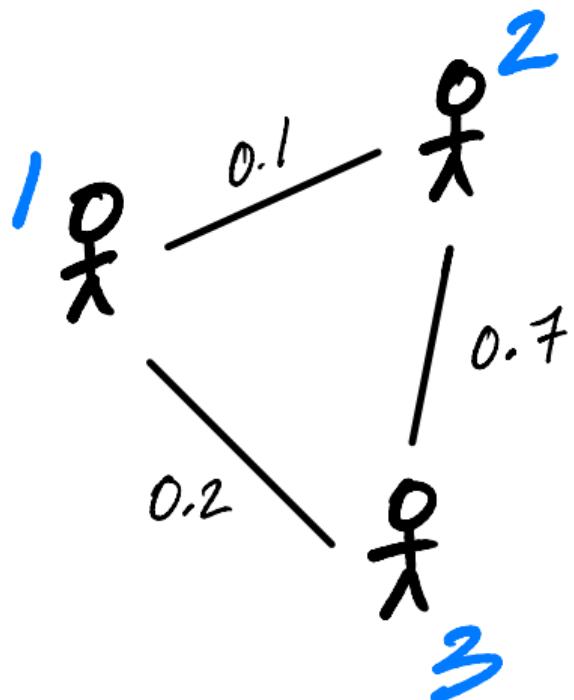
```
W = np.array([
    [1, 0.1, 0.2],
    [0.1, 1, 0.7],
    [0.2, 0.7, 1]
])

D = np.diag(W.sum(axis=1))
L = D - W

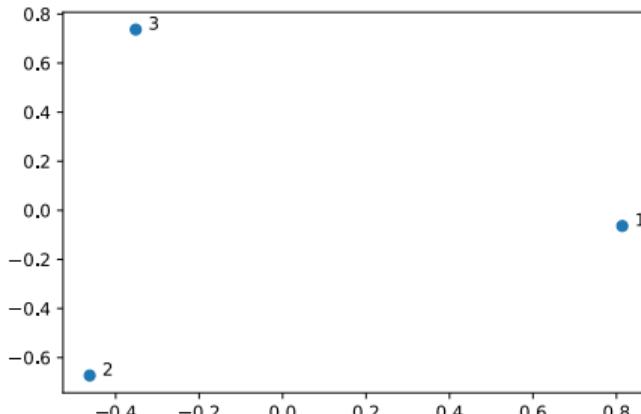
vals, vecs = np.linalg.eigh(L)

# take two eigenvectors
# to map to R^2
f = vecs[:,1:3]
```

Example



$\vec{f}^{(2)}$



$\vec{f}^{(0)}$

Laplacian Eigenmaps

- ▶ This approach is part of the method of “**Laplacian eigenmaps**”
- ▶ Introduced by Mikhail Belkin² and Partha Niyogi
- ▶ It is a type of **spectral embedding**

²Now at HDSI

A Practical Issue

- ▶ The Laplacian is often **normalized**:

$$L_{\text{norm}} = D^{-1/2} L D^{-1/2}$$

where $D^{-1/2}$ is the diagonal matrix whose i th diagonal entry is $1/\sqrt{d_{ii}}$.

- ▶ Proceed by finding the eigenvectors of L_{norm} .

In Summary

- ▶ We can **embed** a similarity graph's nodes into \mathbb{R}^k using the eigenvectors of the graph Laplacian
- ▶ Yet another instance where eigenvectors are solution to optimization problem
- ▶ Next time: using this for dimensionality reduction

DSC 140B

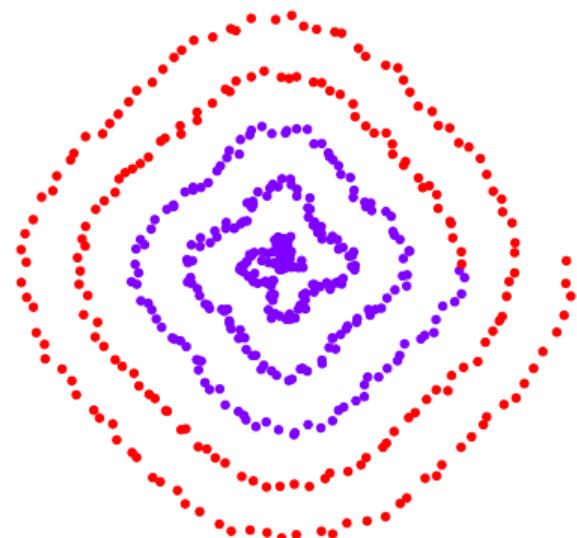
Representation Learning

Lecture 09 | Part 4

Nonlinear Dimensionality Reduction

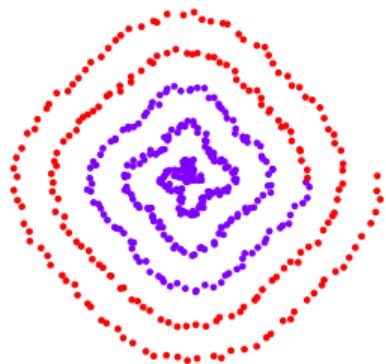
Scenario

- ▶ You want to train a classifier on this data.
- ▶ It would be easier if we could “unroll” the spiral.
- ▶ Data seems to be one-dimensional, even though in two dimensions.
- ▶ Dimensionality reduction?



PCA?

- ▶ Does PCA work here?
- ▶ Try projecting onto one principal component.



No



PCA?

- ▶ PCA simply “rotates” the data.
- ▶ No amount of rotation will “unroll” the spiral.
- ▶ We need a fundamentally different approach that works for non-linear patterns.

Today

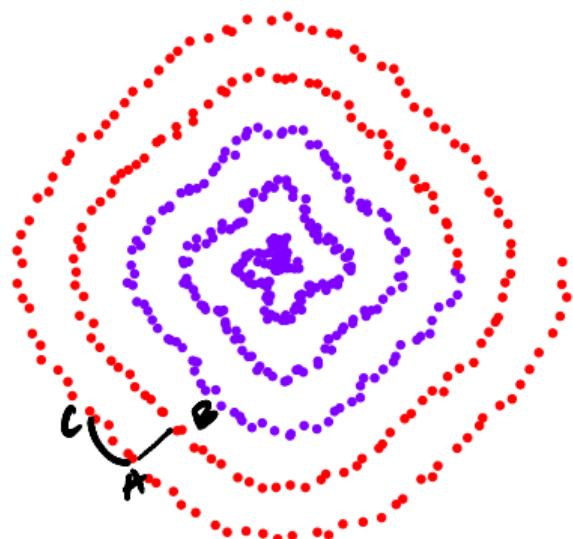
- ▶ Non-linear dimensionality reduction via
spectral embeddings.

Last Time: Spectral Embeddings

- ▶ **Given:** a similarity graph with n nodes, number of dimensions k .
- ▶ **Embed:** each node as a point in \mathbb{R}^k such that similar nodes are mapped to nearby points
- ▶ **Solution:** *bottom k* non-constant eigenvectors of graph Laplacian

Idea

- ▶ Build a similarity graph from points.
- ▶ Points *near the spiral* should be similar.
- ▶ Embed the similarity graph into \mathbb{R}^1



Today

- ▶ 1) How do we build a graph from a set of points?
- ▶ 2) Dimensionality reduction with Laplacian eigenmaps

DSC 140B

Representation Learning

Lecture 09 | Part 5

From Points to Graphs

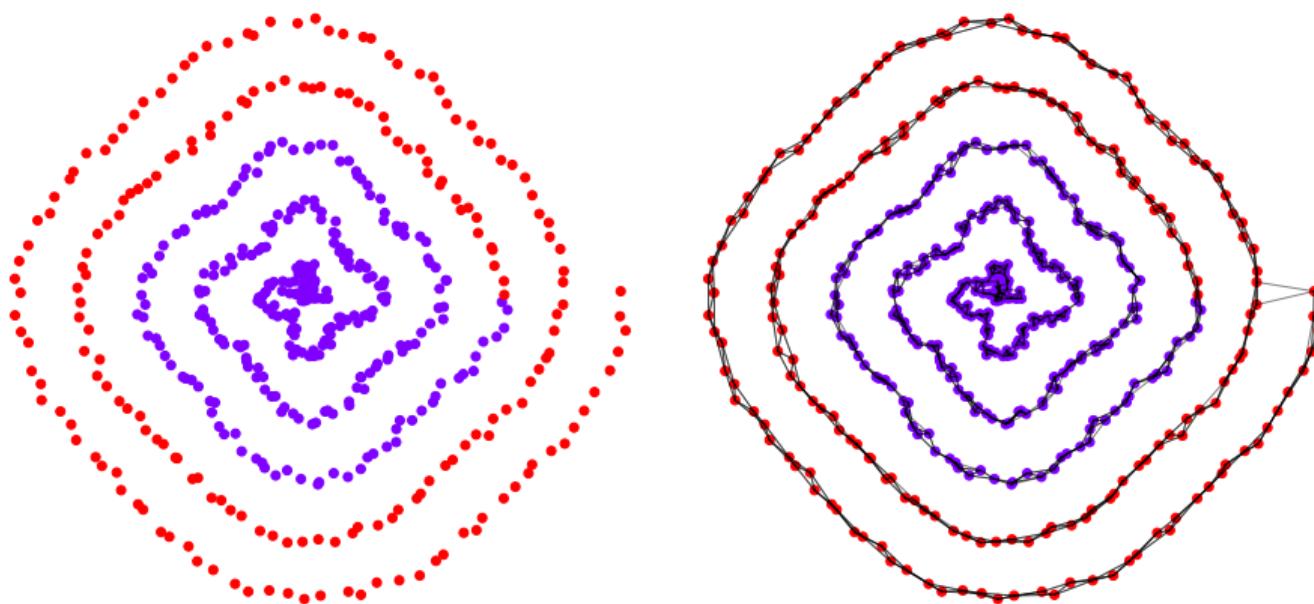
Dimensionality Reduction

- ▶ **Given:** n points in \mathbb{R}^d , number of dimensions $k \leq d$
- ▶ **Map:** each point \vec{x} to new representation $\vec{z} \in \mathbb{R}^k$

Idea

- ▶ Build a similarity graph from points in \mathbb{R}^2
- ▶ Use approach from last lecture to embed into \mathbb{R}^k
- ▶ But how do we represent a set of points as a similarity graph?

Why graphs?

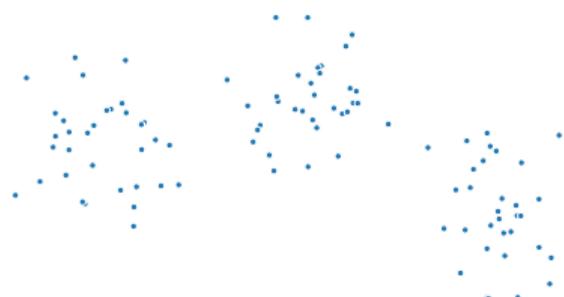


Three Approaches

- ▶ 1) Epsilon neighbors graph
- ▶ 2) k -Nearest neighbor graph
- ▶ 3) fully connected graph with similarity function

Epsilon Neighbors Graph

- ▶ Input: vectors $\vec{x}^{(1)}, \dots, \vec{x}^{(n)}$,
a number ε
- ▶ Create a graph with one
node i per point $\vec{x}^{(i)}$
- ▶ Add edge between nodes i
and j if $\|\vec{x}^{(i)} - \vec{x}^{(j)}\| \leq \varepsilon$
- ▶ Result: **unweighted** graph

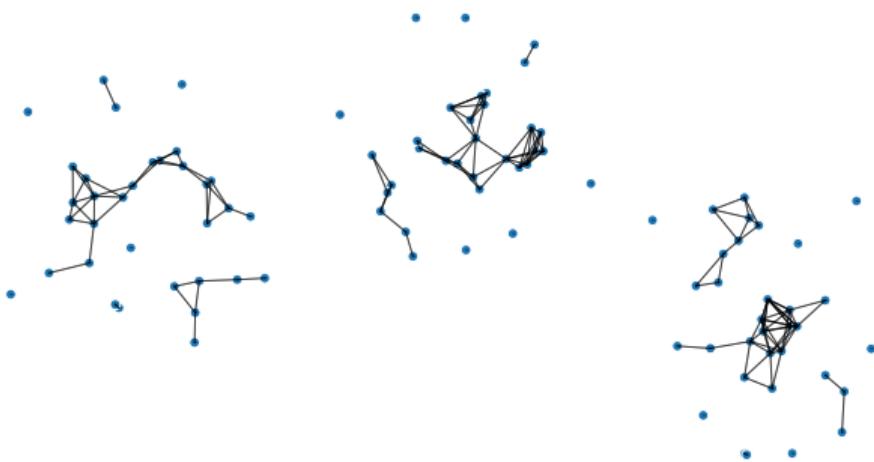


Exercise

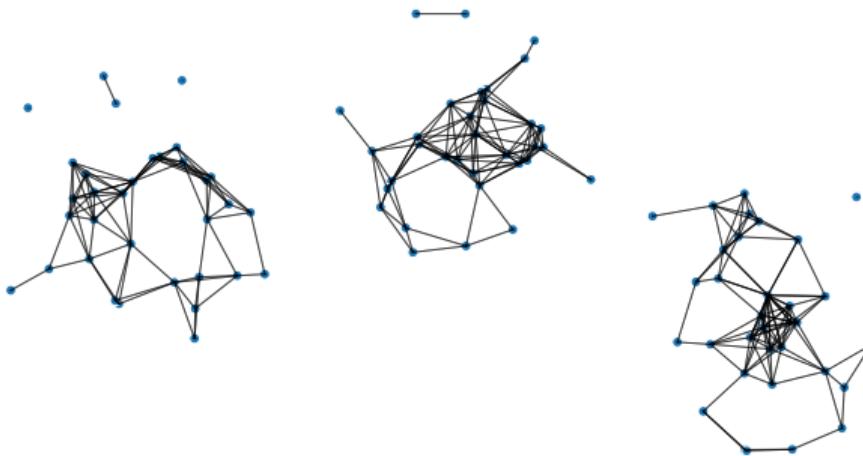
What will the graph look like when ε is small? What about when it is large?



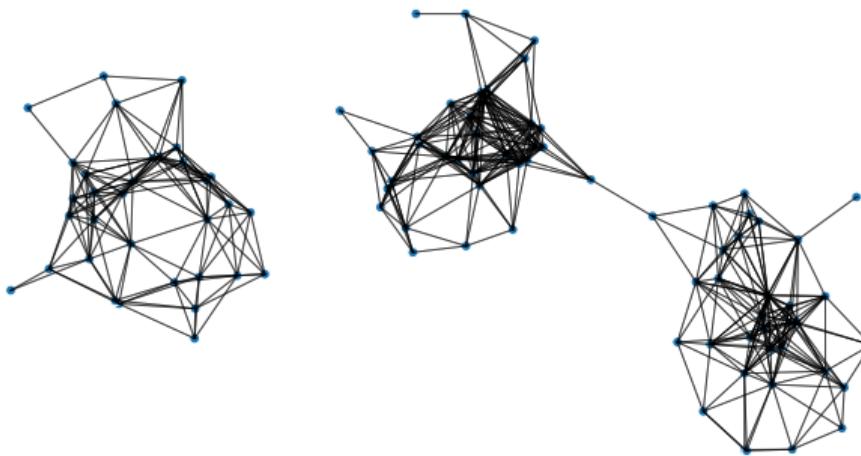
Epsilon Neighbors Graph



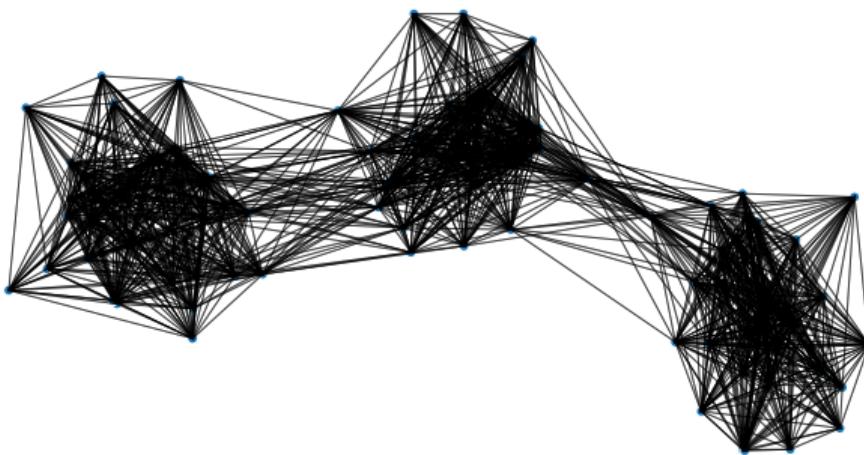
Epsilon Neighbors Graph



Epsilon Neighbors Graph



Epsilon Neighbors Graph



Note

- ▶ We've drawn these graphs by placing nodes at the same position as the point they represent
- ▶ But a graph's nodes can be drawn in any way

Epsilon Neighbors: Pseudocode

```
# assume the data is in X
n = len(X)
adj = np.zeros_like(X) ((n,n))
for i in range(n):
    for j in range(n):
        if distance(X[i], X[j]) <= epsilon:
            adj[i, j] = 1
```

Picking ε

- ▶ If ε is too small, graph is underconnected
- ▶ If ε is too large, graph is overconnected
- ▶ If you cannot visualize, just try and see

With scikit-learn

```
import sklearn.neighbors  
adj = sklearn.neighbors.radius_neighbors_graph(  
    X,  
    radius=...  
)
```

k-Neighbors Graph

- ▶ Input: vectors $\vec{x}^{(1)}, \dots, \vec{x}^{(n)}$,
a number k
- ▶ Create a graph with one
node i per point $\vec{x}^{(i)}$
- ▶ Add edge between each
node i and its k closest
neighbors
- ▶ Result: **unweighted** graph



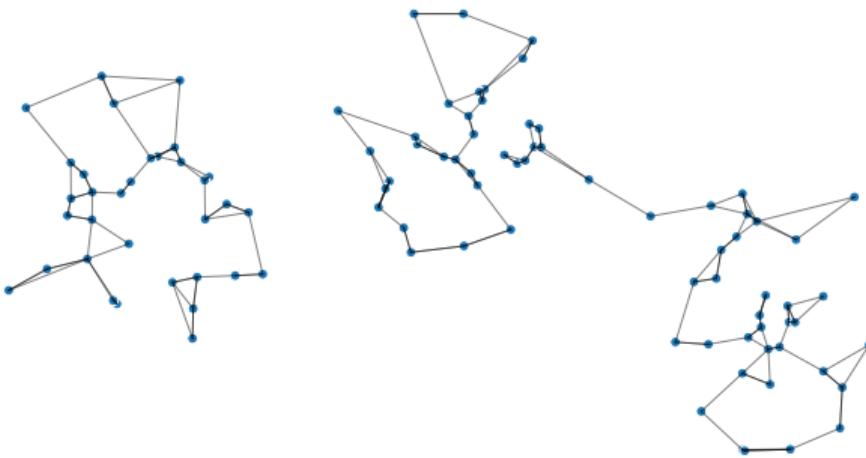
k-Neighbors: Pseudocode

```
# assume the data is in X
n = len(X)
adj = np.zeros_like(X)((n,n))
for i in range(n):
    for j in k_closest_neighbors(X, i):
        adj[i, j] = 1
```

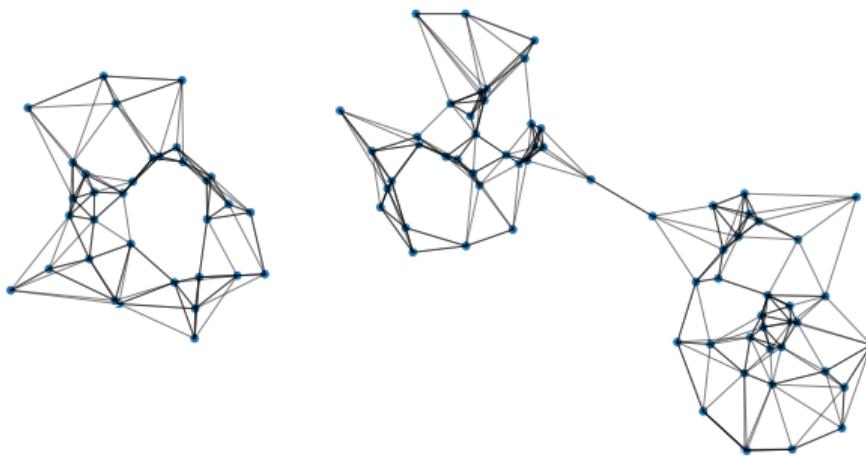
Exercise

Is it possible for a k -neighbors graph to be disconnected?

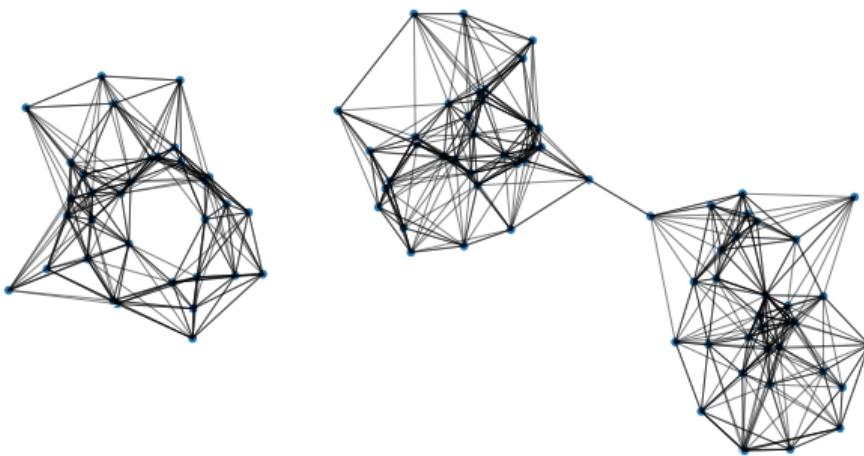
k-Neighbors Graph



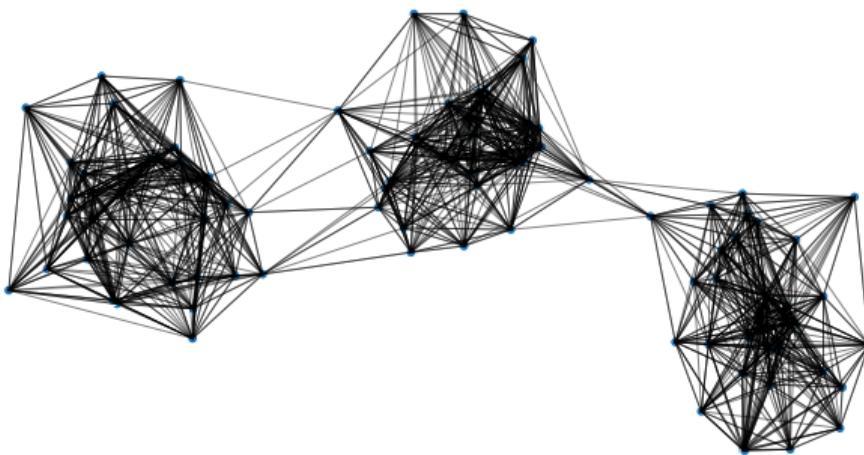
k-Neighbors Graph



k-Neighbors Graph



k-Neighbors Graph

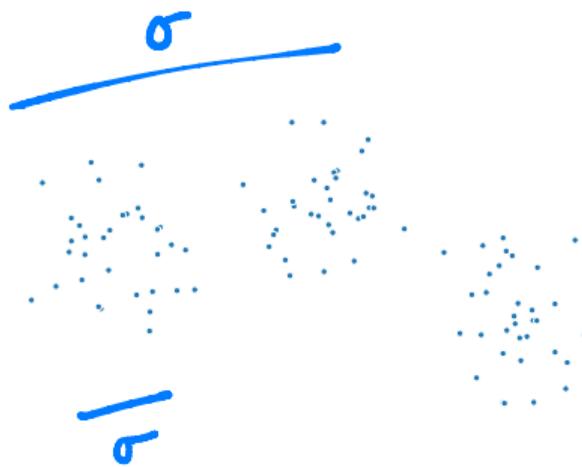


With scikit-learn

```
import sklearn.neighbors  
adj = sklearn.neighbors.kneighbors_graph(  
    X,  
    n_neighbors=...  
)
```

Fully Connected Graph

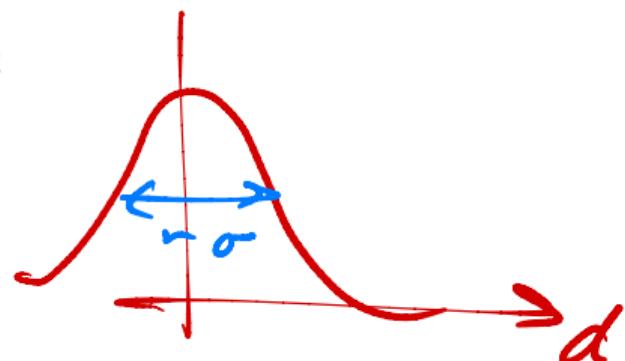
- ▶ Input: vectors $\vec{x}^{(1)}, \dots, \vec{x}^{(n)}$,
a similarity function h
- ▶ Create a graph with one
node i per point $\vec{x}^{(i)}$
- ▶ Add edge between every
pair of nodes. Assign
weight of $h(\vec{x}^{(i)}, \vec{x}^{(j)})$
- ▶ Result: **weighted** graph



Gaussian Similarity

- ▶ A common similarity function: Gaussian
- ▶ Must choose σ appropriately

$$h(\vec{x}, \vec{y}) = e^{-\|\vec{x}-\vec{y}\|^2/\sigma^2}$$



Fully Connected: Pseudocode

x, sigma

```
def h(x, y):
    dist = np.linalg.norm(x - y)
    return np.exp(-dist**2 / sigma**2)

# assume the data is in X
n = len(X)
w = np.ones_like(X) ones((n,n))
for i in range(n):
    for j in range(n):
        w[i, j] = h(X[i], X[j], sigma)
```

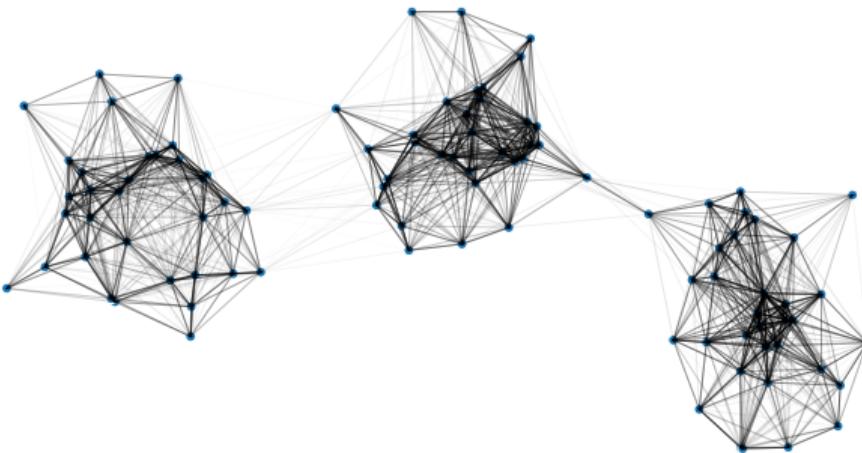
With SciPy

```
distances = scipy.spatial.distance_matrix(X, X)
w = np.exp(-distances**2 / sigma**2)
```

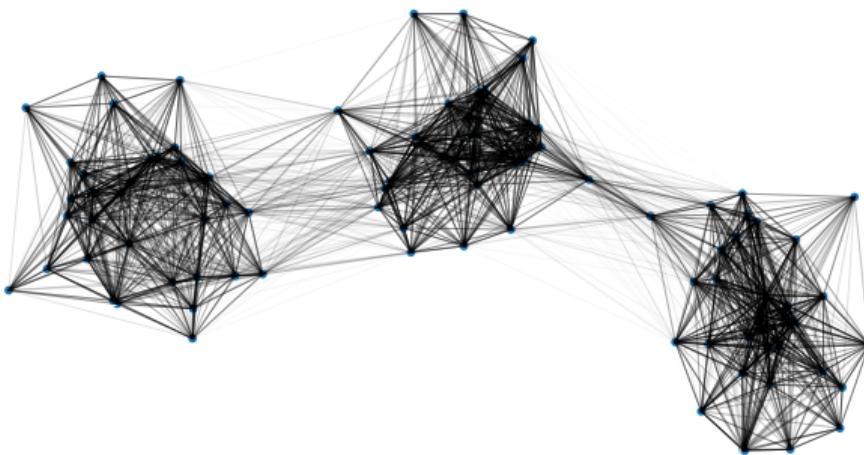
Gaussian Similarity



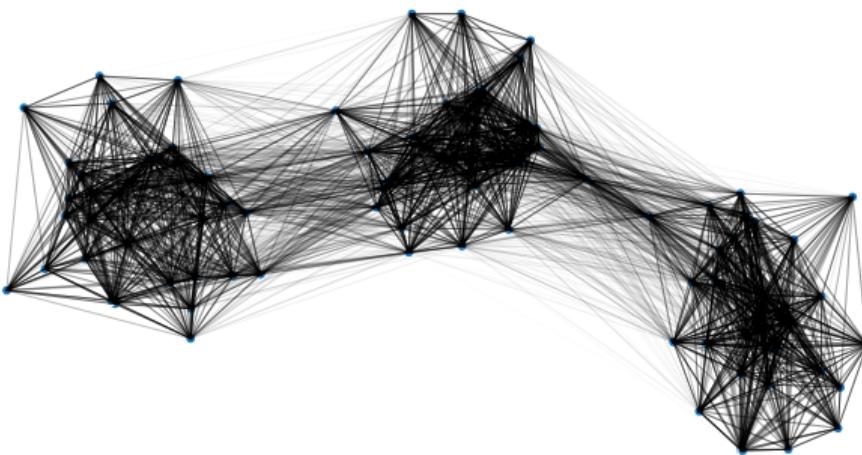
Gaussian Similarity



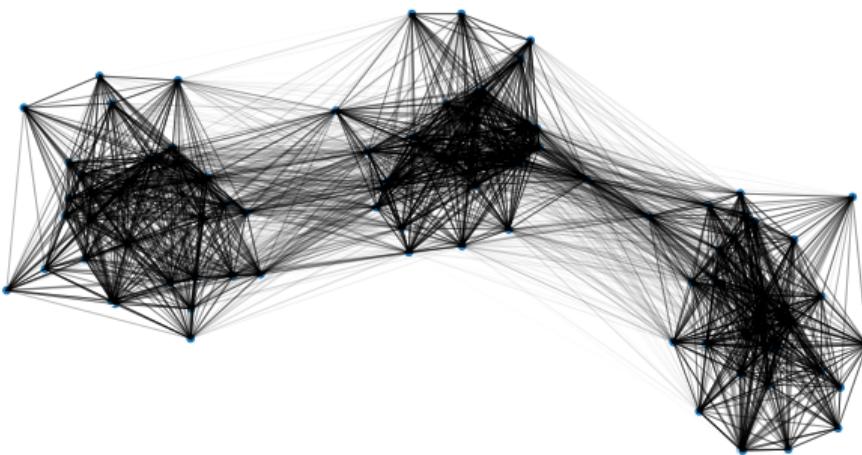
Gaussian Similarity



Gaussian Similarity



Gaussian Similarity



DSC 140B

Representation Learning

Lecture 10 | Part 1

Laplacian Eigenmaps

Problem: Graph Embedding

- ▶ **Given:** a similarity graph, target dimension k
- ▶ **Goal:** **embed** the nodes of the graph as points in \mathbb{R}^k so that similar nodes are nearby
- ▶ **(One) Solution:** Embed using eigenvectors of the graph Laplacian

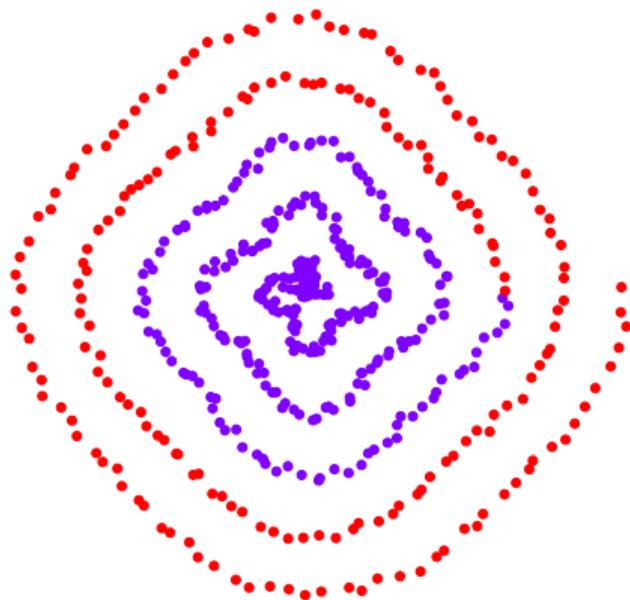
Problem: Non-linear Dimensionality Reduction

- ▶ **Given:** points in \mathbb{R}^d , target dimension k
- ▶ **Goal:** **embed** the points in \mathbb{R}^k so that points that were close in \mathbb{R}^d are close after

Idea

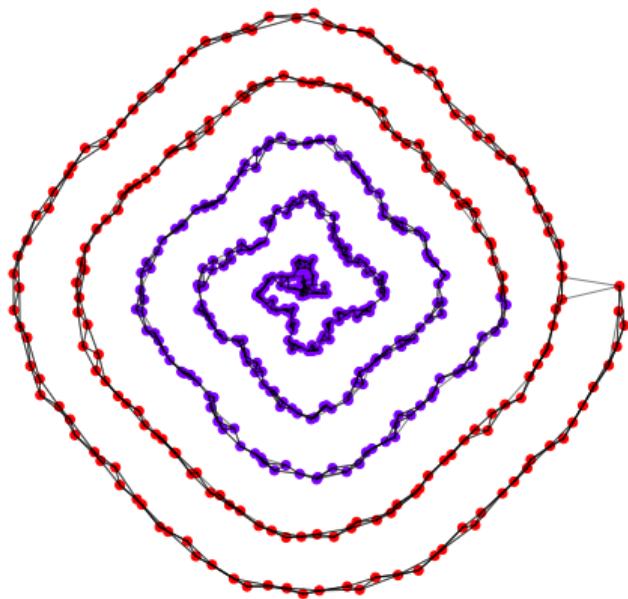
- ▶ Build a similarity graph from points in \mathbb{R}^d
 - ▶ epsilon neighbors, k -neighbors, or fully connected
- ▶ Embed the similarity graph in \mathbb{R}^k using eigenvectors of graph Laplacian

Example 1: Spiral



Example 1: Spiral

- ▶ Build a k -neighbors graph.
- ▶ Note: follows the 1-d shape of the data.

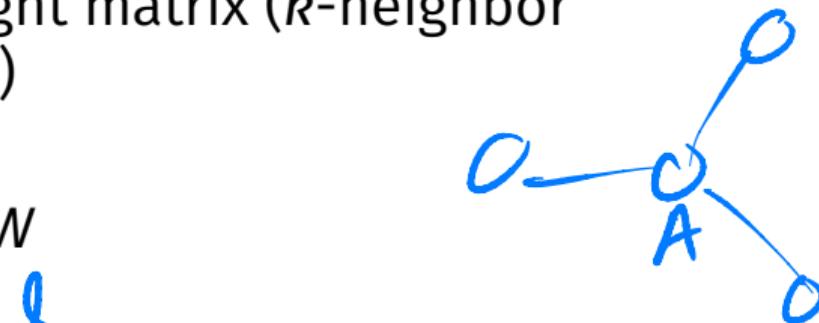


Example 1: Spectral Embedding

- ▶ Let W be the weight matrix (k -neighbor adjacency matrix)

- ▶ Compute $L = D - W$

- ▶ Compute bottom ~~X~~ non-constant eigenvectors of L , use as embedding



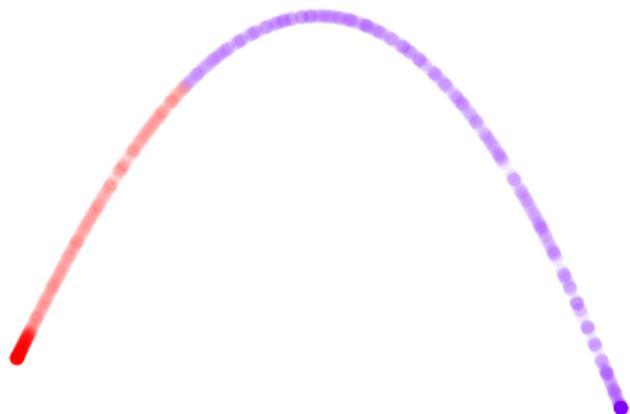
Example 1: Spiral

- ▶ Embedding into \mathbb{R}^1



Example 1: Spiral

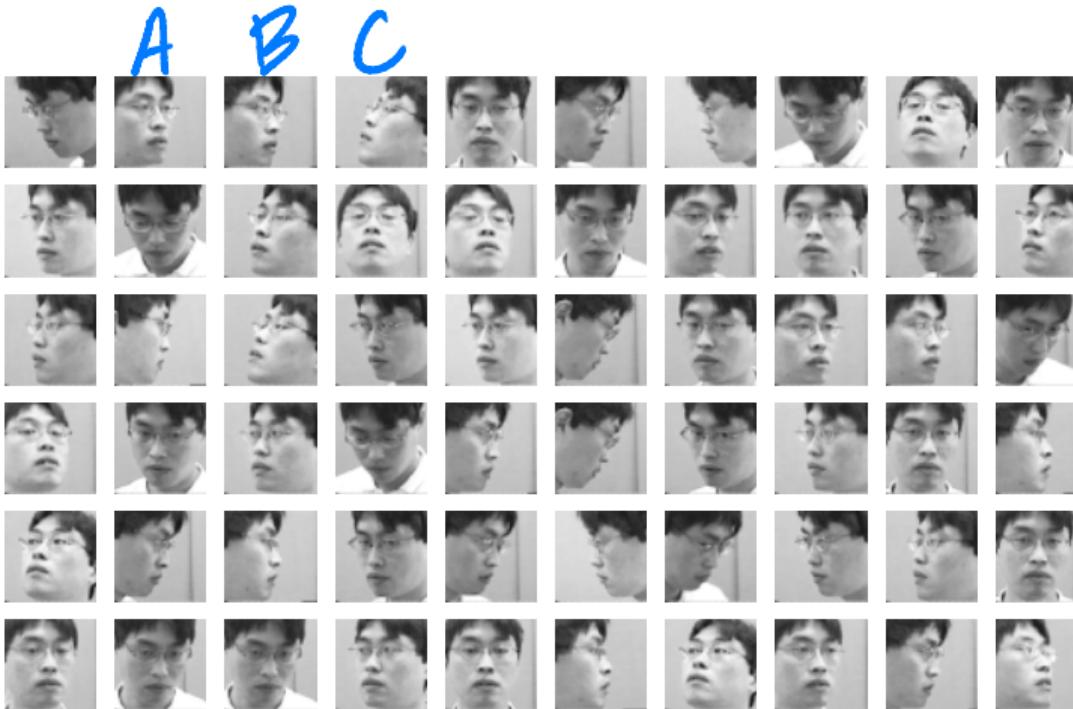
- ▶ Embedding into \mathbb{R}^2



Example 1: Spiral

```
import sklearn.neighbors
import sklearn.manifold
W = sklearn.neighbors.kneighbors_graph(
    X, n_neighbors=4
)
embedding = sklearn.manifold.spectral_embedding(
    W, n_components=2
)
```

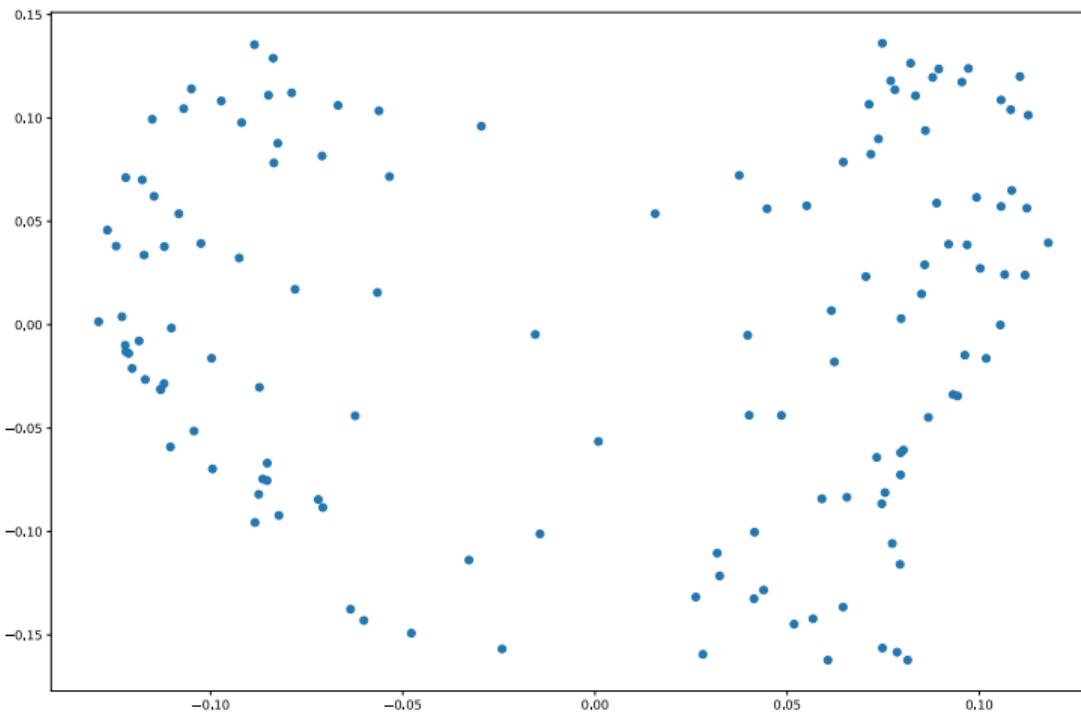
Example 2: Face Pose



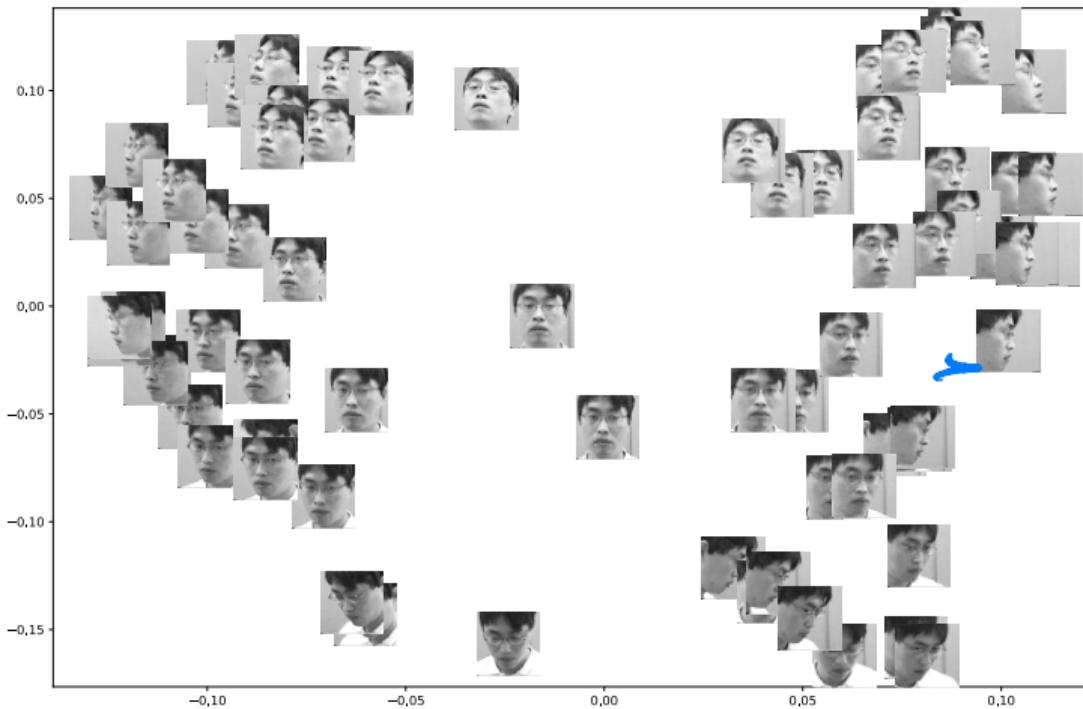
Example 2: Face Pose

- ▶ Construct fully-connected similarity graph with Gaussian similarity
- ▶ Embed with Laplacian eigenmaps

Example 2: Face Pose



Example 2: Face Pose



DSC 140B

Representation Learning

Lecture 10 | Part 2

Prediction

So far...

- ▶ **Unsupervised** representation learning
 - ▶ Reveals the “structure” of the data
 - ▶ Dimensionality reduction and manifold learning
 - ▶ Useful for clustering, visualization, etc.
- ▶ What about **supervised** representation learning?
 - ▶ Suppose we want a representation that makes prediction easier.
 - ▶ PCA, etc., *can* help, but don’t consider labels.

What's ahead...

1. We'll remember a linear predictor from DSC 40A
 - Least squares regression / classification
2. We'll see how using a different representation can make prediction easier
3. We'll introduce **deep learning** as a way of finding representations optimized for prediction

Predicting Opinions

- ▶ We often use the opinions of others to predict our own.
- ▶ But we don't hold all opinions equally...

Movie Ratings

- ▶ Friend A: “This movie was great!”
 - ▶ → I know I’ll like it.
- ▶ Friend B: “This movie was great!”
 - ▶ → I know I won’t like it.
 - ▶ Still useful!
- ▶ Friend C: “This movie was great!”
 - ▶ → I don’t know... they like every movie!
 - ▶ Not useful.

Movie Ratings

- ▶ Five of your friends rate a movie from 0-10:
 - ▶ $x_1: 9$
 - ▶ $x_2: 3$
 - ▶ $x_3: 7$
 - ▶ $x_4: 2$
 - ▶ $x_5: 8$
- ▶ **Task:** What will you rate the movie?

Prediction

- ▶ **Prediction** is a core ML task.
- ▶ **Regression**: output is a number.
 - ▶ Example: movie rating, future salary
- ▶ **Classification**: output is a **class label**.
 - ▶ Example: like the movie? mango is ripe? (yes/no) → **binary**
 - ▶ Example: species (cat, dog, mongoose) → **multiclass**

Prediction Functions

- ▶ Informally: we think our friends' ratings predict our own.
- ▶ Formally: we think there is a function H that takes our friend's ratings $\vec{x} = (x_1, x_2, x_3, x_4, x_5)$ and outputs a good prediction of our rating.

$$H(\vec{x}) \rightarrow \text{prediction}$$

- ▶ H is called a **prediction function**.¹

¹Or, sometimes, a **hypothesis function**

Prediction Functions

- ▶ **Problem:** There are **infinitely many** prediction functions.
 - ▶ $H_1(\vec{x}) = -2x_1 + 3x_5$
 - ▶ $H_2(\vec{x}) = \sin(x_1 \cdot x_2 \cdot x_3 \cdot x_4 \cdot x_5)$
 - ▶ $H_3(\vec{x}) = \sqrt{x_1 + x_3}(x_1 - x_2x_5 + 100)$
 - ▶ ...²
- ▶ How do we pick one?

²Most can't even be expressed algebraically.

The Fundamental Assumption of Learning

- ▶ Informally: The past will repeat itself.
- ▶ Formally: A prediction function that made good predictions in the past will continue to make good predictions in the future³.

³This isn't always true!

Picking a Prediction Function

- ▶ **Idea:** Use **data** to pick a prediction function that worked well in the past.
- ▶ We *hope* it **generalizes** to future predictions.
- ▶ A function that did well in the past but does not generalize is said to have **overfit**.

Training Data

Movie	x_1	x_2	x_3	x_4	x_5	You
#1	8	5	9	2	1	6
#2	3	5	7	8	2	8
#3	1	5	2	3	3	9
#4	0	5	3	8	2	?

A Learning Meta-Algorithm

- ▶ Given data, how do we choose a prediction function?
- ▶ One common strategy is **empirical risk minimization** (ERM).
 - ▶ a.k.a., “minimizing expected loss”

Empirical Risk Minimization (ERM)

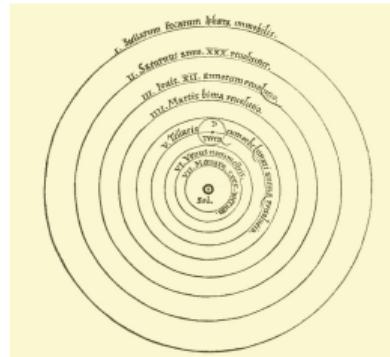
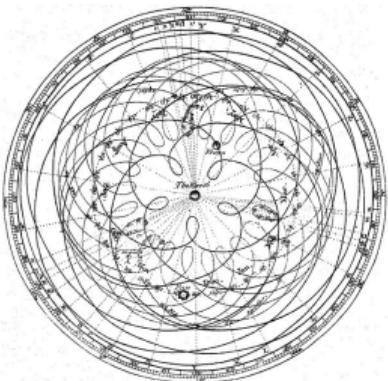
- ▶ Step 1: choose a **hypothesis class**
- ▶ Step 2: choose a **loss function**
- ▶ Step 3: minimize **expected loss (empirical risk)**

Hypothesis Classes

- ▶ A **hypothesis class** \mathcal{H} is a set of possible prediction functions.
- ▶ By choosing a hypothesis class, we are saying something about what the prediction function should look like.
- ▶ Examples:
 - ▶ $\mathcal{H} :=$ linear functions
 - ▶ $\mathcal{H} :=$ functions of the form $\sin(w_1x_1 + \dots + x_5x_5)$
 - ▶ $\mathcal{H} :=$ decision trees of depth 10
 - ▶ $\mathcal{H} :=$ neural networks with one layer

Hypothesis Class Complexity

- ▶ The more complex the hypothesis class, the greater the danger of **overfitting**.
 - ▶ Think: polynomials of degree 10 versus 2.
- ▶ Occam's Razor: assume H is simple.



DSC 140B

Representation Learning

Lecture 10 | Part 3

Least Squares Regression

A Simple Prediction Function

- ▶ We can go a long way by assuming our prediction functions to be **linear**.

A Simple Prediction Function

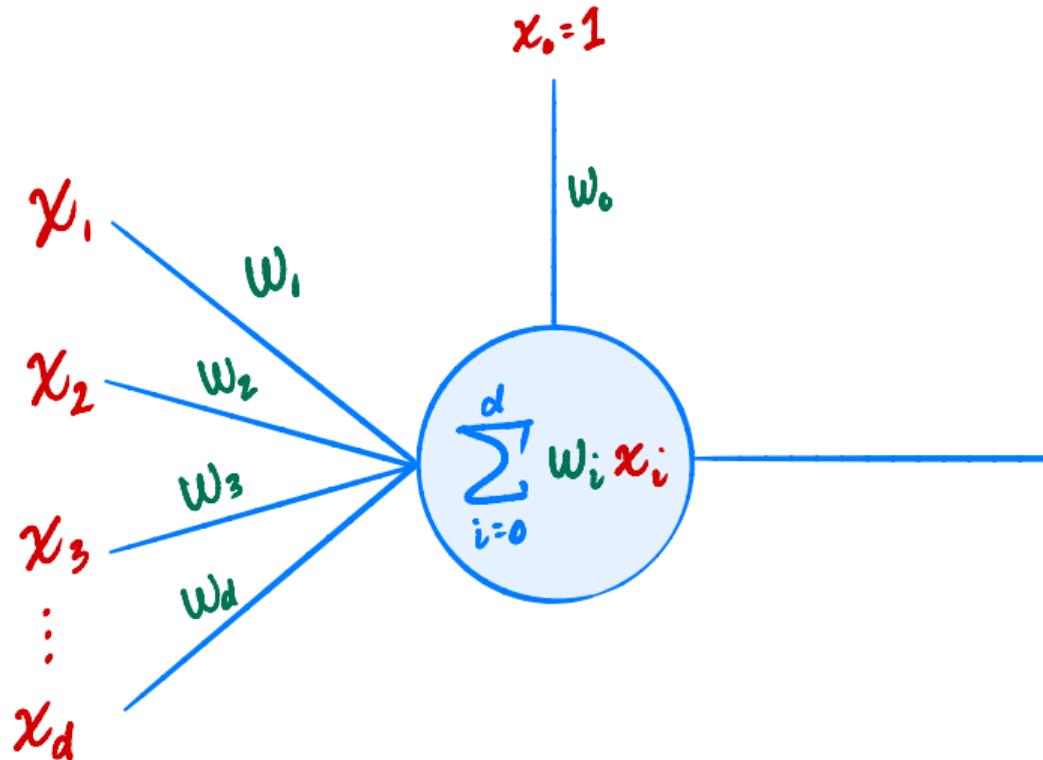
- ▶ Five of your friends rate a movie from 0-10:
 - ▶ $x_1: 9$
 - ▶ $x_2: 3$
 - ▶ $x_3: 7$
 - ▶ $x_4: 2$
 - ▶ $x_5: 8$
- ▶ Idea: predict a **weighted sum**.

Linear Prediction Functions

$$H(\vec{x}) = w_0 + w_1x_1 + w_2x_2 + w_3x_3 + w_4x_4 + w_5x_5$$

- ▶ This is a **linear prediction function**.
- ▶ w_0, w_1, \dots, w_5 are the **parameters** or **weights**.
- ▶ $\vec{w} = (w_0, \dots, w_5)^T$ is a **parameter vector**.

Linear Predictors



Class of Linear Functions

- ▶ There are infinitely many functions of the form

$$H(\vec{x}) = w_0 + w_1x_1 + w_2x_2 + w_3x_3 + w_4x_4 + w_5x_5$$

8 + 3x₁ + 1x₂ + 5x₃ + ...

- ▶ Each one is completely determined by \vec{w} .
 - ▶ Sometimes write $H(\vec{x}; \vec{w})$
- ▶ Example: $\vec{w} = (8, 3, 1, 5, -2, -7)^T$ specifies

$$H(\vec{x}; \vec{w}) = 8 + 3x_1 + 1x_2 + 5x_3 - 2x_4 - 7x_5$$

“Parameterization”

- ▶ A very useful trick.
- ▶ Searching all linear functions \equiv searching over
 $\vec{w} \in \mathbb{R}^6$

In General

- If there are d features, there are $d + 1$ parameters:

$$\begin{aligned}H(\vec{x}) &= w_0 + w_1x_1 + w_2x_2 + \dots + w_dx_d \\&= w_0 + \sum_{i=1}^d w_i x_i\end{aligned}$$

Linear Prediction and the Dot Product

- ▶ The **augmented feature vector** $\text{Aug}(\vec{x})$ is the vector obtained by adding a 1 to the front of \vec{x} :

$$\vec{x} = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_d \end{pmatrix} \quad \text{Aug}(\vec{x}) = \begin{pmatrix} 1 \\ x_1 \\ x_2 \\ \vdots \\ x_d \end{pmatrix}$$

Simplification

- With augmentation, we can write as dot product:

$$\begin{aligned} H(\vec{x}) &= w_0 + w_1x_1 + w_2x_2 + \dots + w_dx_d \\ &= \text{Aug}(\vec{x}) \cdot \vec{w} \end{aligned}$$

$$\vec{w} = \begin{pmatrix} w_0 \\ w_1 \\ \vdots \\ w_d \end{pmatrix} \quad \text{Aug}(\vec{x}) = \begin{pmatrix} 1 \\ x_1 \\ x_2 \\ \vdots \\ x_d \end{pmatrix}$$

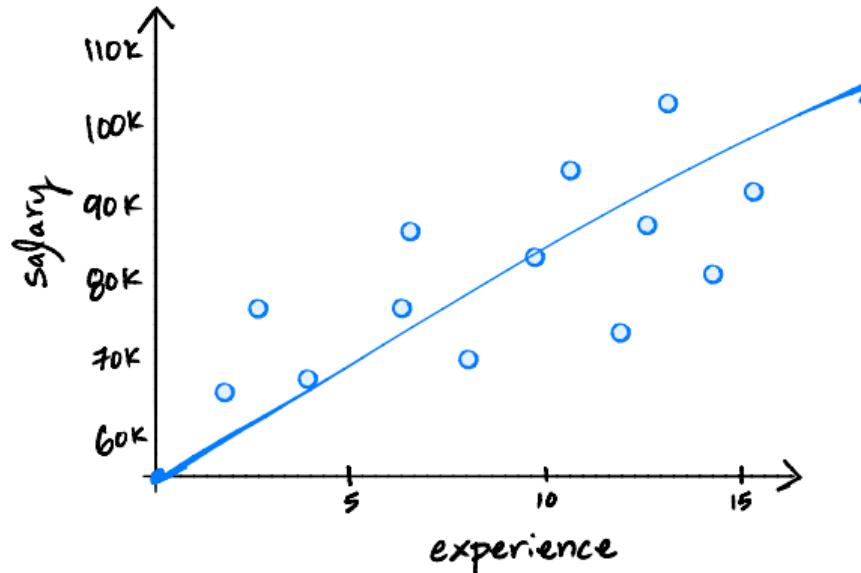
Geometric Meaning

- ▶ It can be very useful to **think geometrically** when reasoning about prediction algorithms.

Example

- ▶ A linear prediction function for salary.

$$H_1(\vec{x}) = \$50,000 + (\text{experience}) \times \$8,000$$



Surface

- ▶ The **surface** of a prediction function H is the surface made by plotting $H(\vec{x})$ for all \vec{x} .
- ▶ If H is a linear prediction function, and⁴
 - ▶ $\vec{x} \in \mathbb{R}^1$, then $H(x)$ is a straight line.
 - ▶ $\vec{x} \in \mathbb{R}^2$, the surface is a plane.
 - ▶ $\vec{x} \in \mathbb{R}^d$, the surface is a d -dimensional **hyperplane**.

⁴when plotted in the original feature coordinate space!

Empirical Risk Minimization (ERM)

- ▶ Step 1: choose a **hypothesis class**
 - ▶ Let's assume we've chosen linear predictors
- ▶ Step 2: choose a **loss function**
- ▶ Step 3: minimize **expected loss (empirical risk)**

Step #2: Choose a loss function

- ▶ Suppose we assume prediction function is linear.
- ▶ There are still infinitely-many possibilities.
- ▶ We'll pick one that works well on training data.
- ▶ What does “works well” mean?

Example: Movie Ratings

Movie	x_1	x_2	x_3	x_4	x_5	You
#1	8	5	9	2	1	6
#2	3	5	7	8	2	8
#3	1	5	2	3	3	9
#4	0	5	3	8	2	?

Quantifying Quality

- ▶ Consider a training example $(\vec{x}^{(i)}, y_i)$
 - ▶ Notation: $\vec{x}^{(i)}$ is the “ith training example”
 - ▶ $\vec{x}_j^{(i)}$ is the “jth entry of the ith training example”
- ▶ The “right answer” is y_i
- ▶ Our prediction function outputs $H(\vec{x}^{(i)})$
- ▶ We measure the difference using a **loss function**.

Loss Function

- ▶ A **loss function** quantifies how wrong a single prediction is.

$$L(H(\vec{x}^{(i)}), y_i)$$

L (prediction for example i , correct answer for example i)

Empirical Risk

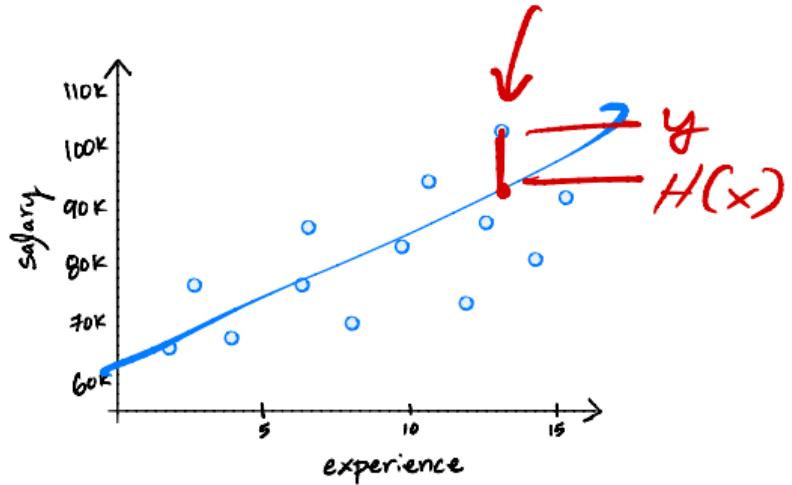
- ▶ A good H is good *on average* over entire data set.
- ▶ The **expected loss** (or **empirical risk**) is one way of measuring this:

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

- ▶ Note: depends on H and the data!

Loss Functions for Regression

- ▶ We want $H(\vec{x}^{(i)}) \approx y_i$.
- ▶ **Absolute loss:** $|H(\vec{x}^{(i)}) - y_i|$
- ▶ **Square loss:** $(H(\vec{x}^{(i)}) - y_i)^2$

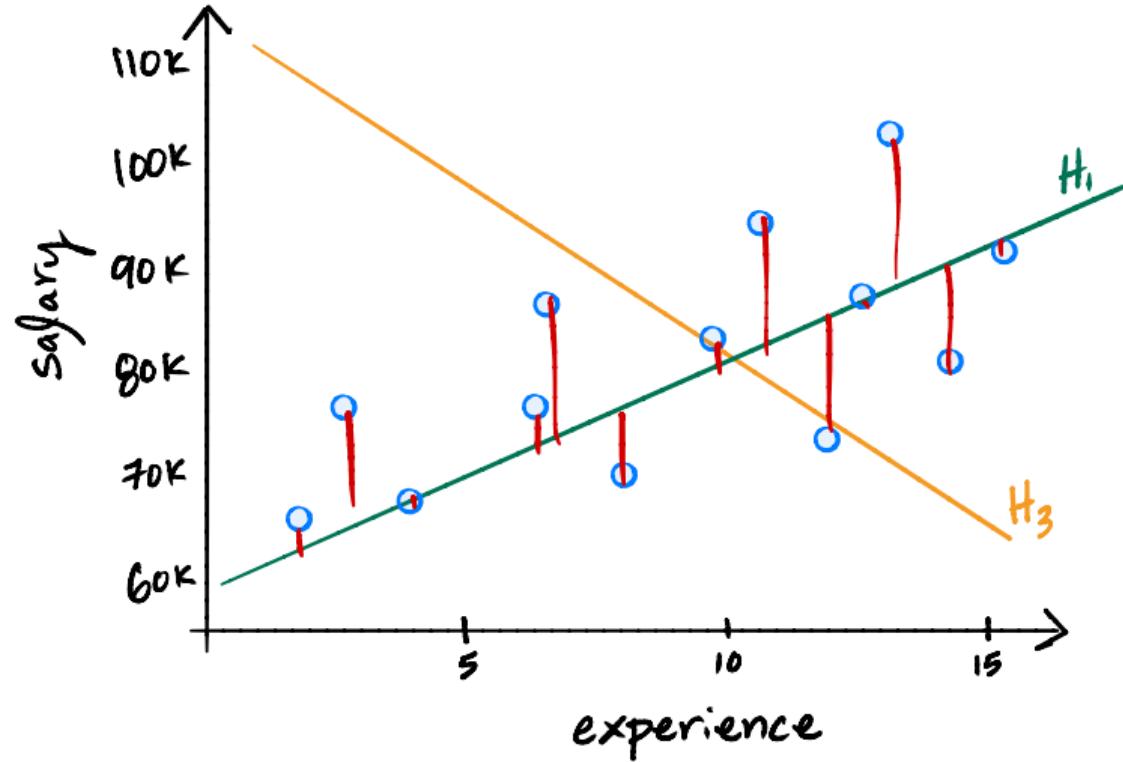


Mean Squared Error

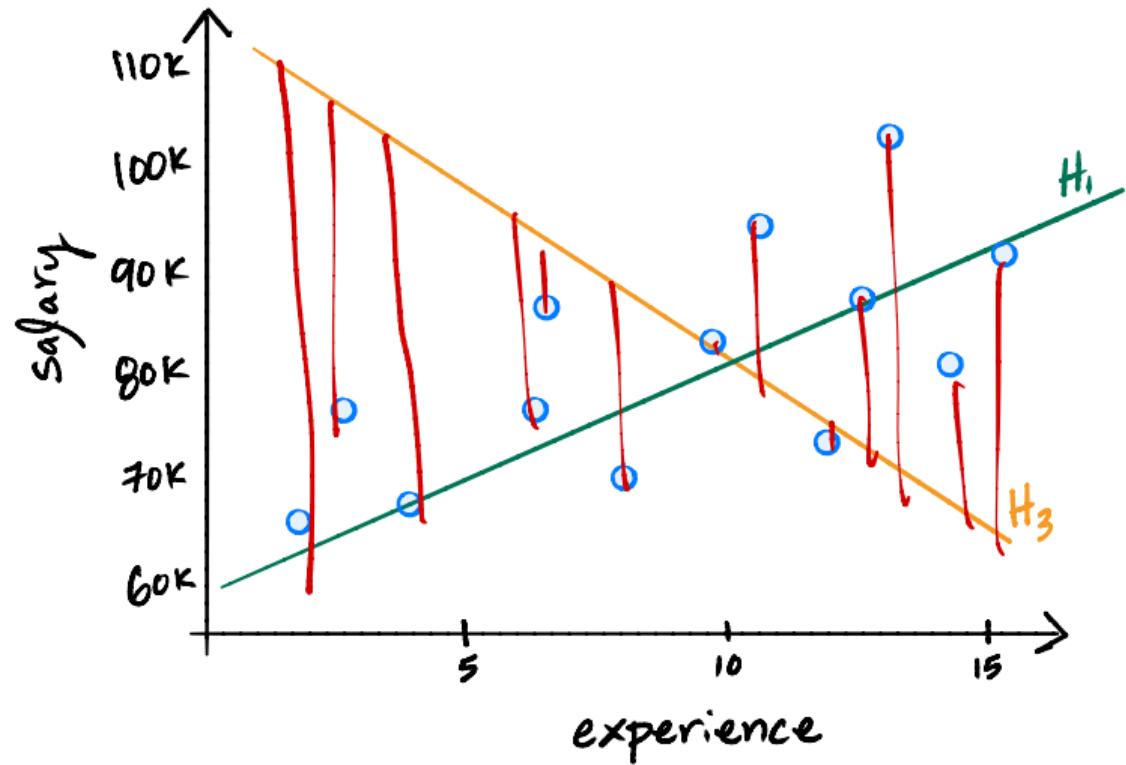
- ▶ **Expected square loss** (mean squared error):

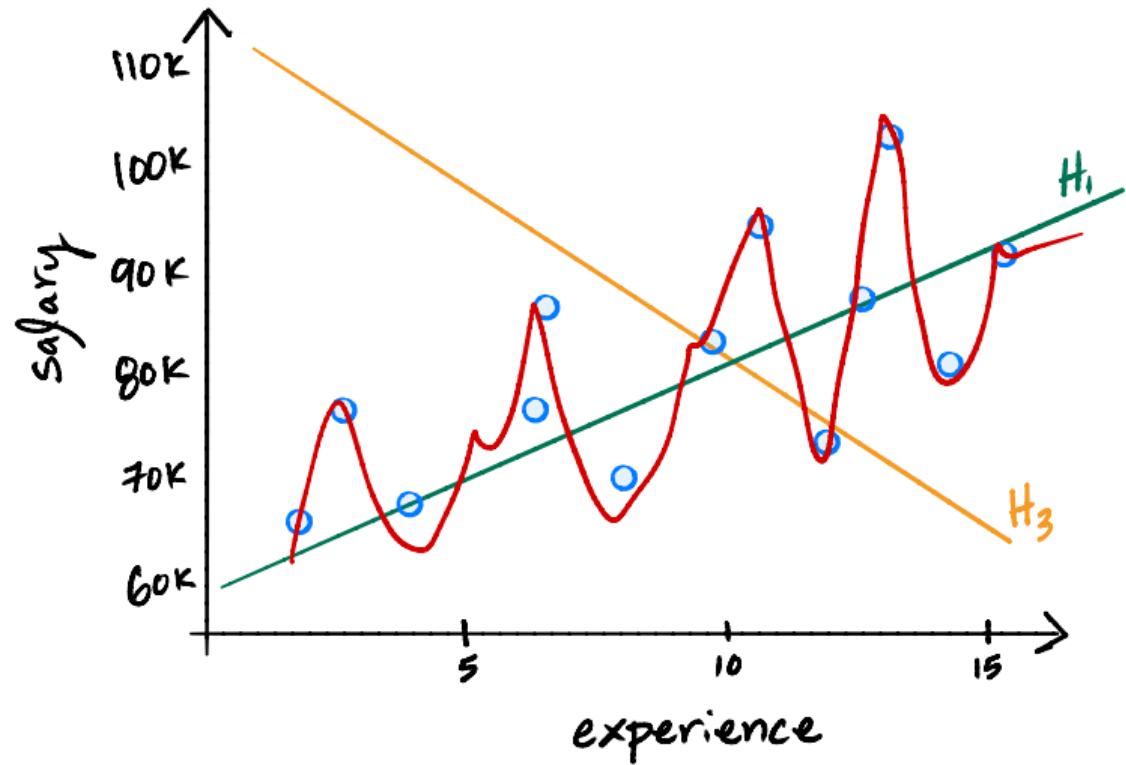
$$R_{\text{sq}}(H) = \frac{1}{n} \sum_{i=1}^n (H(\vec{x}^{(i)}) - y_i)^2$$

- ▶ This is the empirical risk for the square loss.
- ▶ Goal: find H minimizing MSE.



R_1





Step #3: Minimize MSE

- We want to find an H minimizing this:

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

$$H(\vec{x}) = \vec{w} \cdot \text{Aug}(\vec{x})$$

- It helps to use linear assumption:

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

Calculus

$$\nabla R(\vec{w}) = \vec{0}$$

- ▶ We want to find \vec{w} that minimizes the average square loss:

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

$$\vec{y} = (y_1, y_2, \dots, y_n)$$

- ▶ Take the gradient, set to $\vec{0}$, solve.
- ▶ Solution: the **Normal Equations**, $\vec{w} = (X^t X)^{-1} X^t \vec{y}$

Design Matrix

- X is the **design matrix** X :

$$X = \begin{pmatrix} \text{Aug}(\vec{x}^{(1)}) & \longrightarrow \\ \text{Aug}(\vec{x}^{(2)}) & \longrightarrow \\ \vdots & \vdots \\ \text{Aug}(\vec{x}^{(n)}) & \longrightarrow \end{pmatrix} = \begin{pmatrix} 1 & x_1^{(1)} & x_2^{(1)} & \dots & x_d^{(1)} \\ 1 & x_1^{(2)} & x_2^{(2)} & \dots & x_d^{(2)} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & x_1^{(n)} & x_2^{(n)} & \dots & x_d^{(n)} \end{pmatrix}$$

Note

- ▶ There was a closed-form solution!
- ▶ This is a direct consequence of using the **mean squared error**.
- ▶ Not true if we use, e.g., the **mean absolute error**.

Why linear?

- ▶ Easy to work with mathematically.
- ▶ Harder to overfit.
- ▶ But still quite powerful.

DSC 140B

Representation Learning

Lecture 10 | Part 4

Least Squares Classifiers

Movie Ratings

- ▶ Five of your friends rate a movie from 0-10:
 - ▶ $x_1: 9$
 - ▶ $x_2: 3$
 - ▶ $x_3: 7$
 - ▶ $x_4: 2$
 - ▶ $x_5: 8$
- ▶ **Task:** Will you like the movie? (yes / no)

Classification

- ▶ Linear prediction functions can be used in classification, too.

$$H(\vec{x}) = w_0 + w_1x_1 + w_2x_2 + \dots + w_dx_d$$

- ▶ Same ERM paradigm also useful.

A Classifier from a Regressor

- ▶ Binary classification can be thought of as regression where the targets are 1 and -1
 - ▶ (or 0 and 1, or ...)
- ▶ $H(\vec{x})$ outputs a real number. Use the **sign** function to turn it into -1, 1:

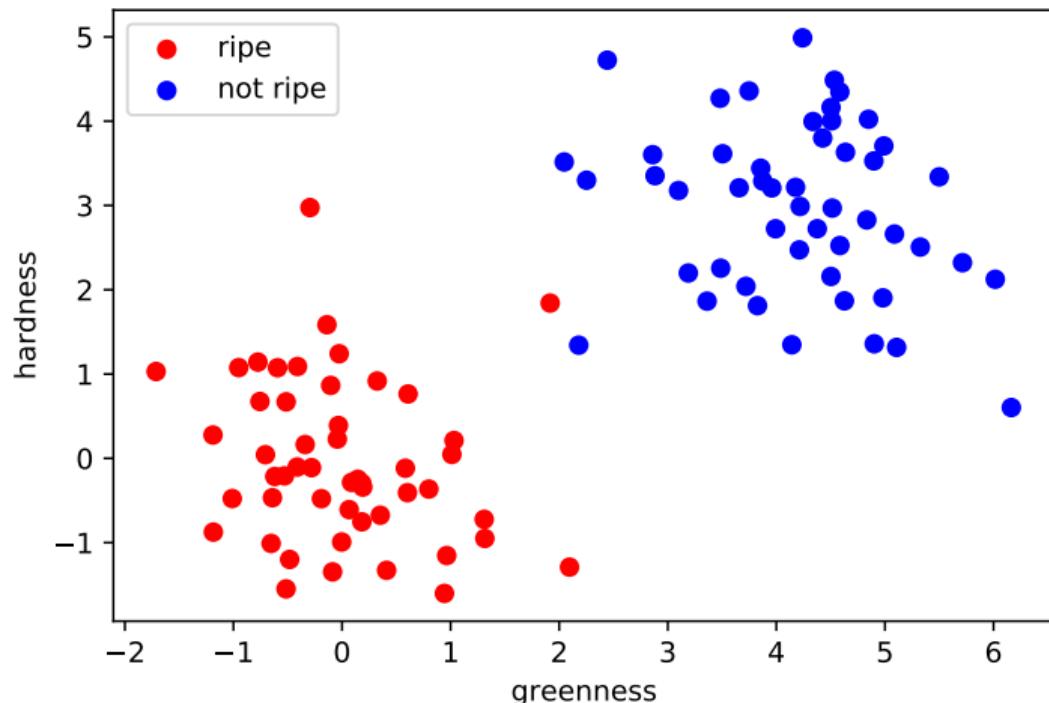
$$\text{sign}(z) = \begin{cases} 1 & z > 0 \\ -1 & z < 0 \\ 0 & \text{otherwise} \end{cases}$$

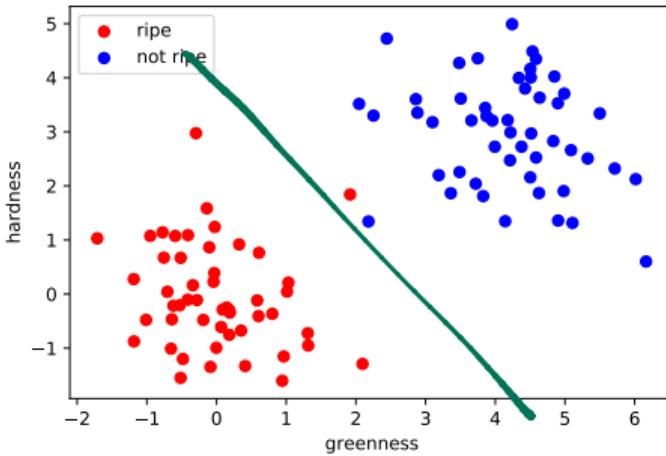
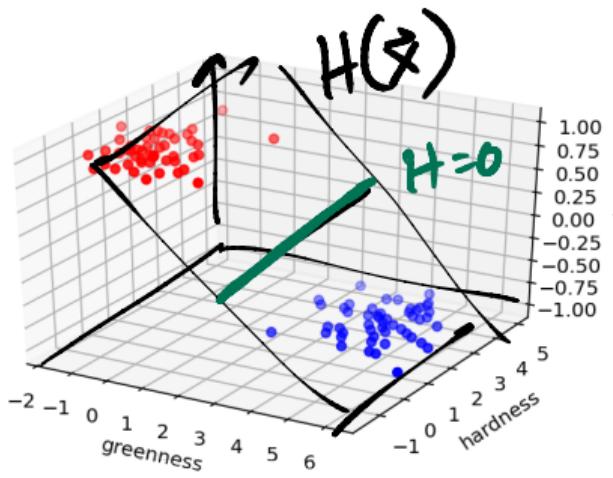
- ▶ Final prediction: $\text{sign}(H(\vec{x}))$

Example: Mango Ripeness

- ▶ Predict whether a mango is ripe given greenness and hardness.
- ▶ Idea: gather a set of labeled **training data**.
 - ▶ Inputs along with correct output (i.e., “the answer”).

Greenness	Hardness		Ripe
0.7	0.9		1
0.2	0.5		-1
0.3	0.1		-1
:	:		:





Decision Boundary

- ▶ The **decision boundary** is the place where the output of $H(x)$ switches from “yes” to “no”.
 - ▶ If $H > 0 \mapsto$ “yes” and $H < 0 \mapsto$ “no”, the decision boundary is where $H = 0$.
- ▶ If H is a linear predictor and⁵
 - ▶ $\vec{x} \in \mathbb{R}^1$, then the decision boundary is just a number.
 - ▶ $\vec{x} \in \mathbb{R}^2$, the boundary is a straight line.
 - ▶ $\vec{x} \in \mathbb{R}^d$, the boundary is a $d - 1$ dimensional (hyper) plane.

⁵when plotted in the original feature coordinate space!

Empirical Risk Minimization

- ▶ Step 1: choose a **hypothesis class**
 - ▶ Let's assume we've chosen linear predictors
- ▶ Step 2: choose a **loss function**
- ▶ Step 3: minimize **expected loss (empirical risk)**

Exercise

Can we use the square loss for classification?

$$(H(\vec{x}^{(i)}) - y_i)^2$$

Least Squares and Outliers

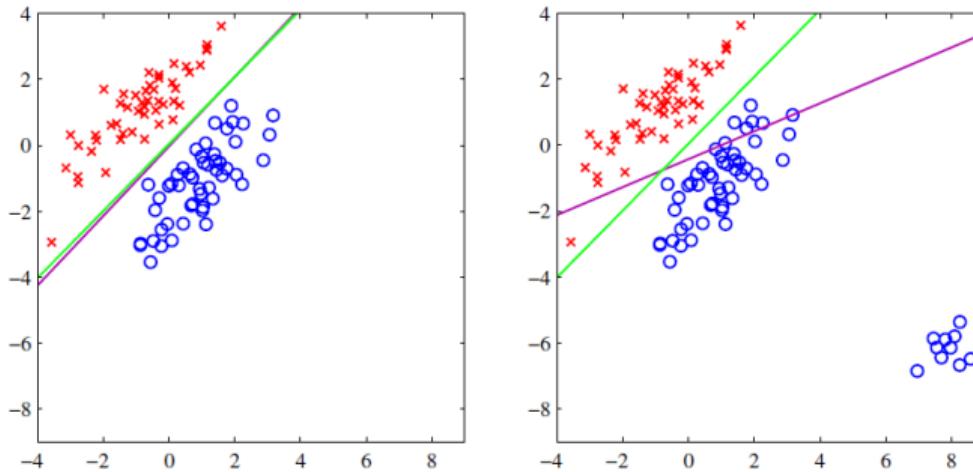


Figure 4.4 The left plot shows data from two classes, denoted by red crosses and blue circles, together with the decision boundary found by least squares (magenta curve) and also by the logistic regression model (green curve), which is discussed later in Section 4.3.2. The right-hand plot shows the corresponding results obtained when extra data points are added at the bottom left of the diagram, showing that least squares is highly sensitive to outliers, unlike logistic regression.

Square Loss for Classification

- ▶ We **can** use the square loss for classification
 - ▶ The “least squares classifier”
- ▶ However, the square loss penalizes being “too correct”
- ▶ **Example:** suppose the correct label is 1. What is the square loss of predicting 10? -9?

Loss Functions

- ▶ There are many different loss functions for classification.
- ▶ Each leads to a different classifier:
 - ▶ Logistic Regression
 - ▶ Support Vector Machine
 - ▶ Perceptron
 - ▶ etc.
- ▶ But that's for another class... (DSC 140A)

DSC 140B

Representation Learning

Lecture 11 | Part 1

Linear Limitations

Linear Predictors

- ▶ Last time, we saw linear prediction functions:

$$\begin{aligned}H(\vec{x}; \vec{w}) &= w_0 + w_1 x_1 + \dots + w_d x_d \\&= \text{Aug}(\vec{x}) \cdot \vec{w}\end{aligned}$$

Linear Decision Functions

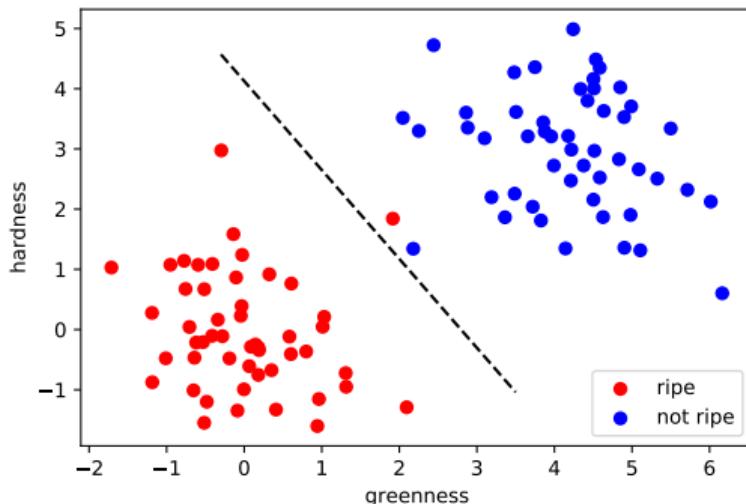
- ▶ A linear prediction function H outputs a number.
- ▶ What if classes are +1 and -1?
- ▶ Can be turned into a **decision function** by taking:

$$\text{sign}(H(\vec{x}))$$

- ▶ **Decision boundary** is where $H = 0$
 - ▶ Where the sign switches from positive to negative.

Decision Boundaries

- ▶ A linear decision function's decision boundary is linear.
 - ▶ A line, plane, hyperplane, etc.



An Example: Parking Predictor

- ▶ **Task:** Predict (yes / no): Is there parking available at UCSD right now?
- ▶ What training data to collect? What features?

Useful Features

- ▶ Time of day?
- ▶ Day's high temperature?
- ▶ ...

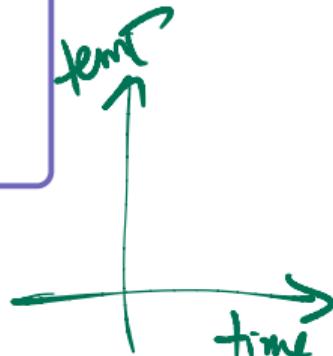
Exercise

Imagine a scatter plot of the training data with the two features:

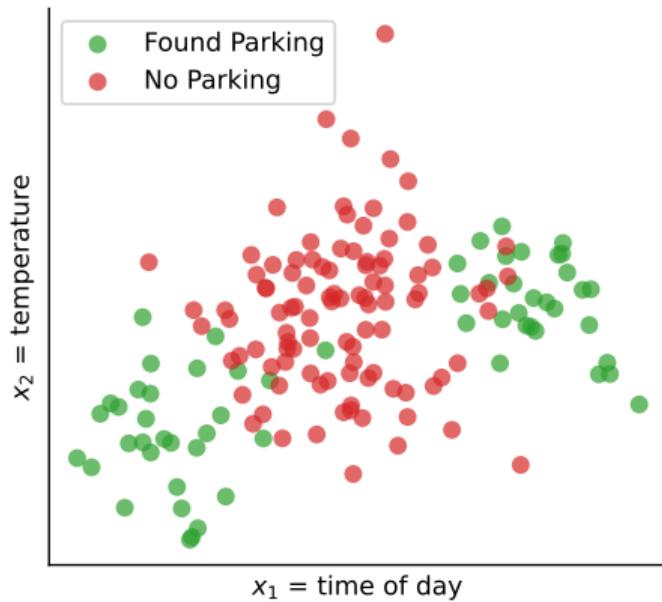
- ▶ x_1 = time of day
- ▶ x_2 = temperature

“yes” examples are green, “no” are red.

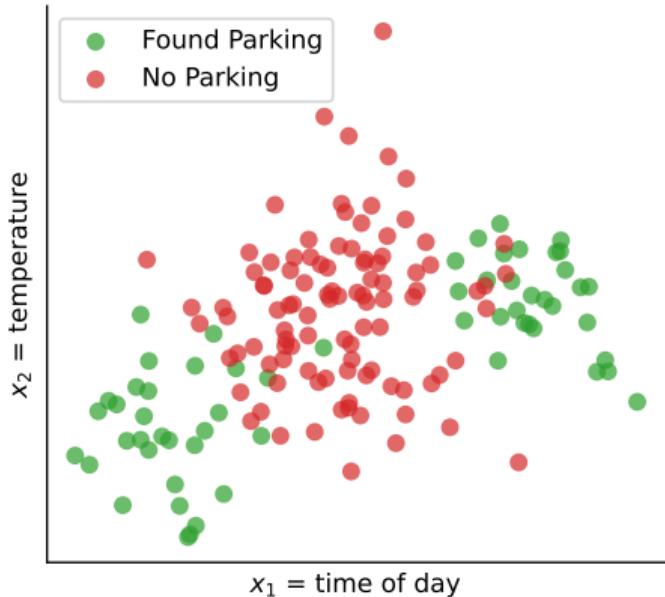
What does it look like?



Parking Data



Uh oh



- ▶ A linear decision function won't work.
- ▶ What do we do?

Today's Question

- ▶ How do we learn non-linear patterns using linear prediction functions?

DSC 140B

Representation Learning

Lecture 11 | Part 2

Feature Maps

Representations

- ▶ We **represented** the data with two features: time and temperature
- ▶ In this **representation**, the trend is **nonlinear**.
 - ▶ There is no good linear decision function
 - ▶ Learning is “difficult”.

Idea

- ▶ **Idea:** We'll make a new **representation** by creating **new features** from the **old features**.
- ▶ The “right” representation makes the problem easy again.
- ▶ What new features should we create?

New Feature Representation

- ▶ Linear prediction functions¹ work well when relationship is linear
 - ▶ When x is small we should predict -1
 - ▶ When x is large we should predict +1
- ▶ But parking's relationship with time is not linear:
 - ▶ When time is small we should predict +1
 - ▶ When time is medium we should predict -1
 - ▶ When time is large we should predict +1

¹Remember: they are weighted votes.

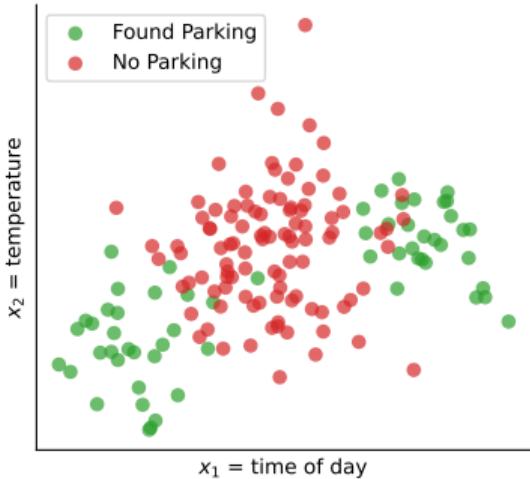
Exercise

How can we “transform” the time of day x_1 to create a new feature x'_1 satisfying:

- ▶ When x'_1 is small, we should predict -1
- ▶ When x'_1 is large, we should predict +1

What about the temperature, x_2 ?

Idea



- ▶ Transform “time” to “absolute time until/since Noon”
- ▶ Transform “temp.” to “absolute difference between temp. and 72°”

Basis Functions

- ▶ We will transform:
 - 1) ▶ the time, x_1 , to $|x_1 - \text{Noon}|$
 - 2) ▶ the temperature, x_2 , to $|x_2 - 72^\circ|$
- ▶ Formally, we've designed non-linear **basis functions**:

$$\varphi_1(x_1, x_2) = |x_1 - \text{Noon}|$$

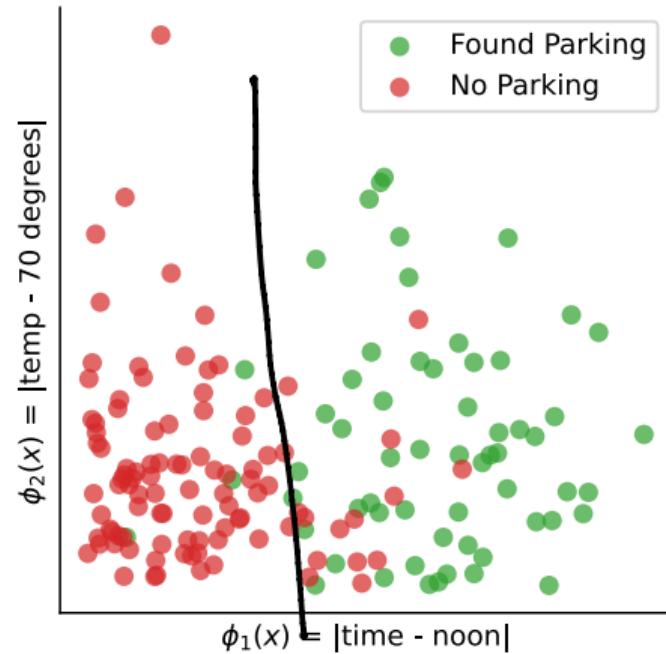
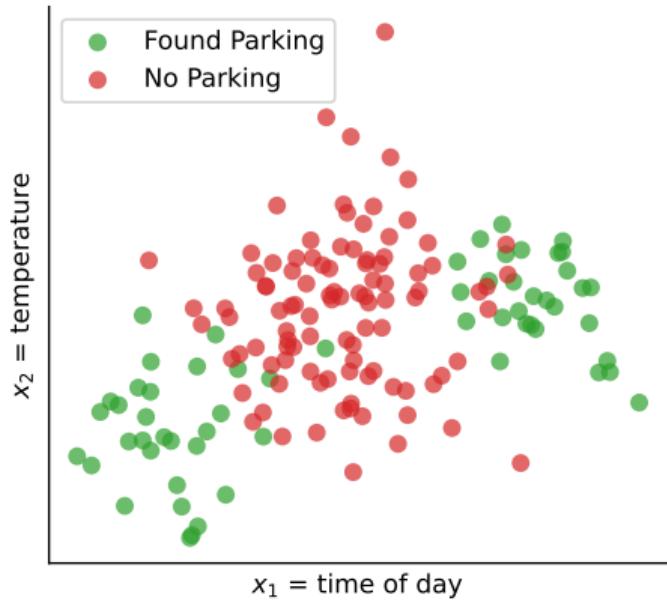
$$\varphi_2(x_1, x_2) = |x_2 - 72^\circ|$$

- ▶ In general a basis function φ maps $\mathbb{R}^d \rightarrow \mathbb{R}$

Feature Mapping

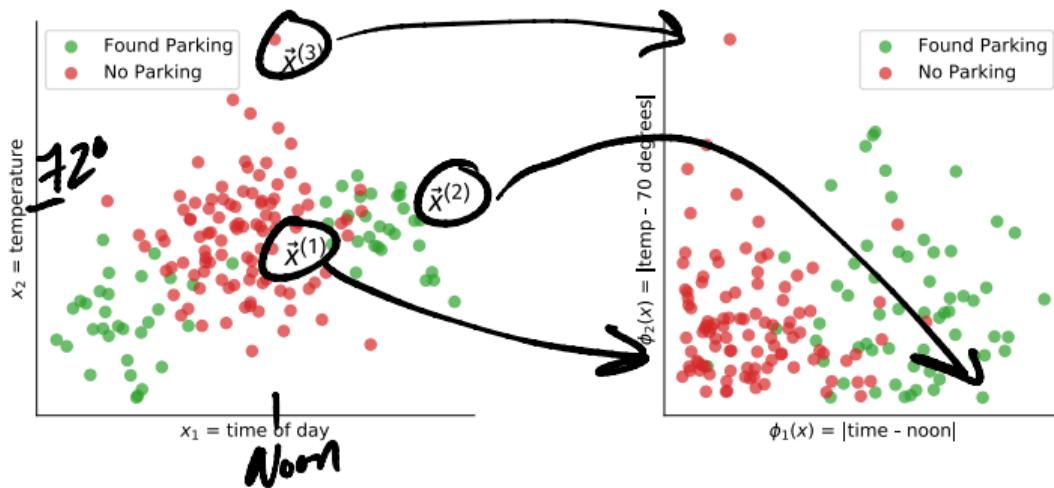
- ▶ Define $\vec{\varphi}(\vec{x}) = (\varphi_1(\vec{x}), \varphi_2(\vec{x}))^T$. $\vec{\varphi}$ is a **feature map**
 - ▶ Input: vector in “old” representation
 - ▶ Output: vector in “new” representation
- ▶ Example: $(2_{pm} \quad 64^\circ)$
 $\vec{\varphi}((10\text{a.m.}, 75^\circ)^T) = (2 \text{ hours}, 3^\circ)^T$
- ▶ $\vec{\varphi}$ maps raw data to a **feature space**.

Feature Space, Visualized

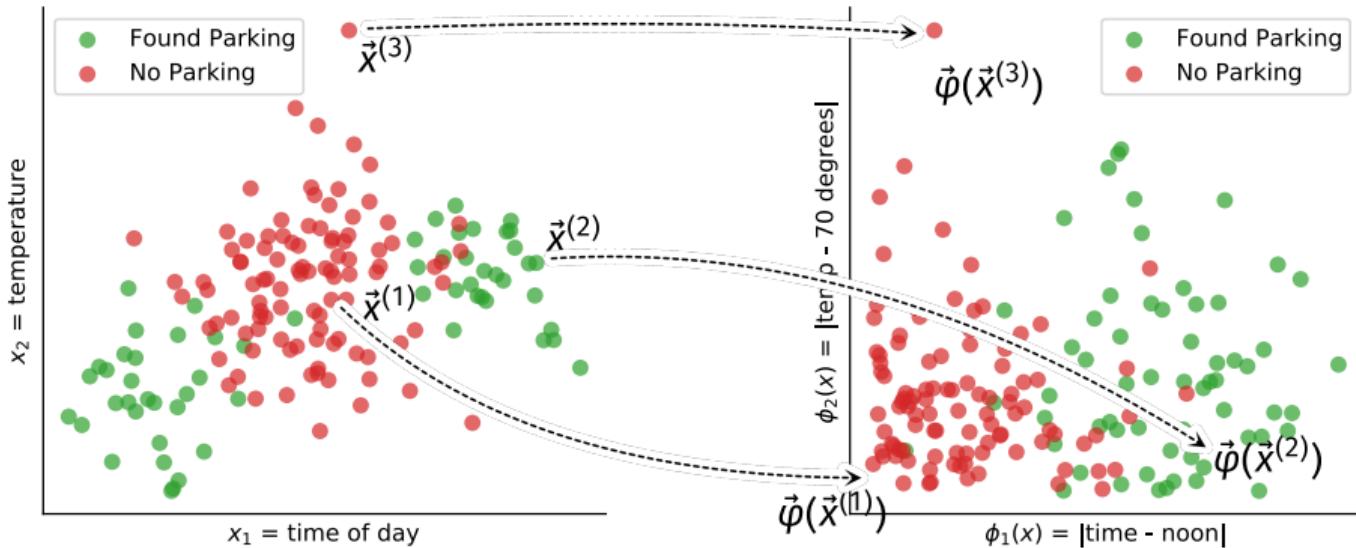


Exercise

Where does $\vec{\phi}$ map $\vec{x}^{(1)}$, $\vec{x}^{(2)}$, and $\vec{x}^{(3)}$?



Solution



After the Mapping

- ▶ The basis functions φ_1, φ_2 give us our “new” features.
- ▶ This gives us a new **representation**.
- ▶ In this representation, learning (classification) is easier.

Training

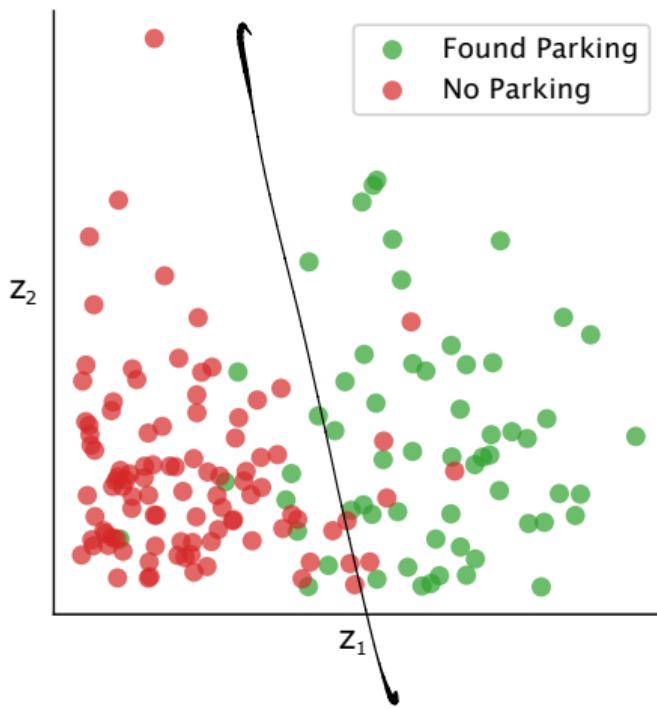
- ▶ Map each training example $\vec{x}^{(i)}$ to feature space, creating new training data:

$$\vec{z}^{(1)} = \vec{\varphi}(\vec{x}^{(1)}), \quad \vec{z}^{(2)} = \vec{\varphi}(\vec{x}^{(2)}), \quad \dots, \quad \vec{z}^{(n)} = \vec{\varphi}(\vec{x}^{(n)})$$

- ▶ Fit linear prediction function H in usual way:

$$H_f(\vec{z}) = w_0 + w_1 z_1 + w_2 z_2 + \dots + w_d z_d$$

Training Data in Feature Space



Prediction

- If we have \vec{z} in feature space, prediction is:

$$H_f(\vec{z}) = w_0 + w_1 z_1 + w_2 z_2 + \dots + w_d z_d$$

Prediction

- But if we have \vec{x} from original space, we must “convert” \vec{x} to feature space first:

$$\begin{aligned}H(\vec{x}) &= H_f(\vec{\varphi}(\vec{x})) \quad \text{z}_1 \quad \text{z}_2 \quad \text{z}_d \\&= H_f((\varphi_1(\vec{x}), \varphi_2(\vec{x}), \dots, \varphi_d(\vec{x}))^T) \\&= w_0 + w_1 \varphi_1(\vec{x}) + w_2 \varphi_2(\vec{x}) + \dots + w_d \varphi_d(\vec{x})\end{aligned}$$

$\text{z}_1 \quad \text{z}_2 \quad \text{z}_d$

Overview: Feature Mapping

- ▶ A basis function can involve any/all of the original features:

$$\varphi_3(\vec{x}) = x_1 \cdot x_2$$

- ▶ We can make more basis functions than original features:

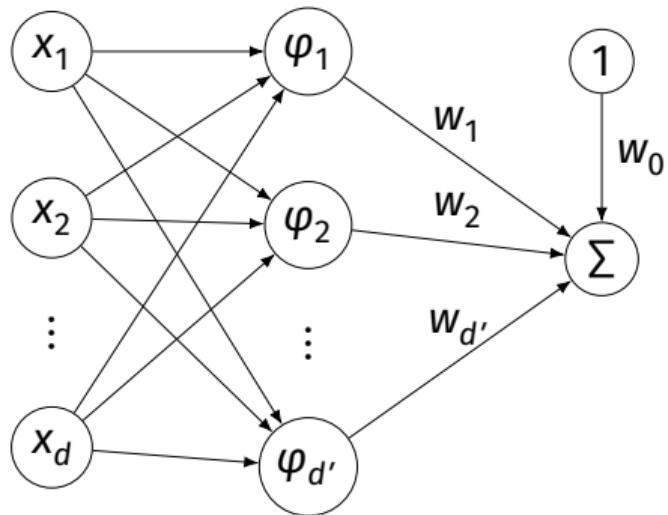
$$\vec{\varphi}(\vec{x}) = (\varphi_1(\vec{x}), \varphi_2(\vec{x}), \varphi_3(\vec{x}))^T$$

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:

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

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



Today's Question

- ▶ Q: How do we learn non-linear patterns using linear prediction functions?
- ▶ A: Use non-linear basis functions to map to a feature space.

DSC 140B

Representation Learning

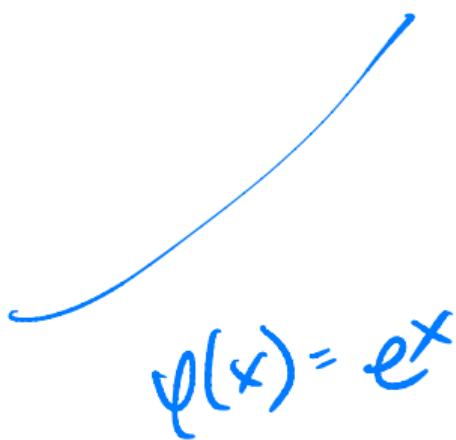
Lecture 11 | Part 3

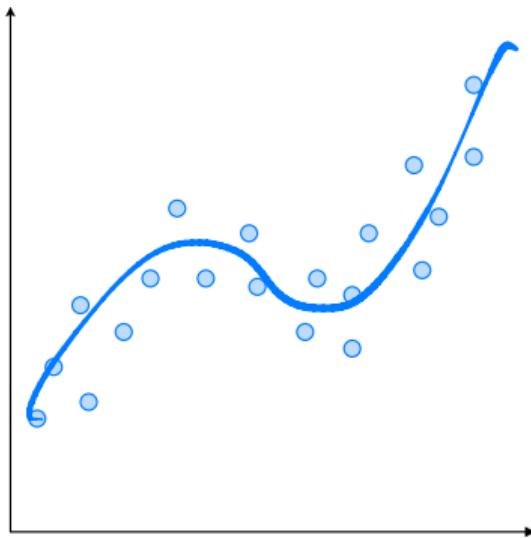
Basis Functions and Regression

By the way...

- ▶ You've (probably) seen basis functions used before.
- ▶ Linear regression for non-linear patterns in DSC 40A.

Example


$$y(x) = e^x$$



Fitting Non-Linear Patterns

- ▶ Fit function of the form

$$H(x) = w_0 + w_1x + w_2x^2 + w_3x^3 + w_4x^4$$

- ▶ Linear function of \vec{w} , non-linear function of x .

The Trick

- ▶ Treat x, x^2, x^3, x^4 as **new** features.
- ▶ Create design matrix:

$$X = \begin{pmatrix} 1 & x_1 & x_1^2 & x_1^3 & x_1^4 \\ 1 & x_2 & x_2^2 & x_2^3 & x_2^4 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & x_n & x_n^2 & x_n^3 & x_n^4 \end{pmatrix}$$

- ▶ Solve $X^T X \vec{w} = X^T \vec{y}$ for \vec{w} , as usual.
- ▶ Works for more than just polynomials.

Another View

- ▶ We have changed the representation of a point:

$$x \mapsto (x, x^2, x^3, x^4)$$

- ▶ Basis functions:

$$\varphi_1(x) = x \quad \varphi_2(x) = x^2 \quad \varphi_3(x) = x^3 \quad \varphi_4(x) = x^4$$

DSC 140B

Representation Learning

Lecture 11 | Part 4

A Tale of Two Spaces

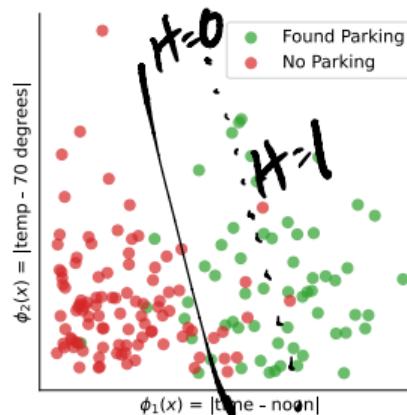
A Tale of Two Spaces

- ▶ The **original space**: where the raw data lies.
- ▶ The **feature space**: where the data lies after feature mapping $\vec{\phi}$
- ▶ Remember: we fit a linear prediction function in the **feature space**.

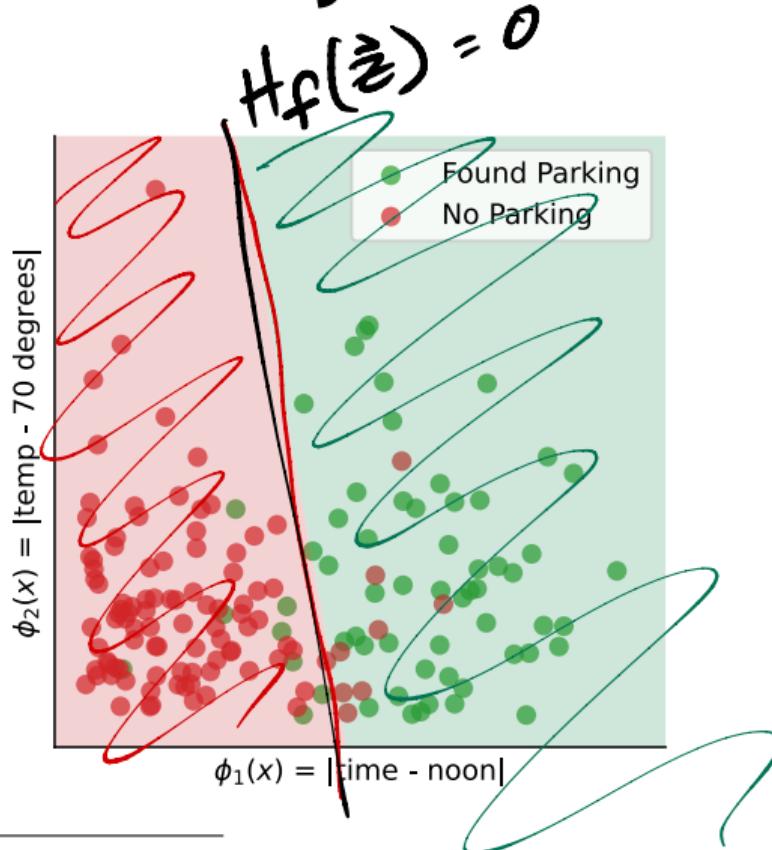
H

Exercise

- ▶ In **feature space**, what does the decision boundary look like?
- ▶ What does the prediction function surface look like?

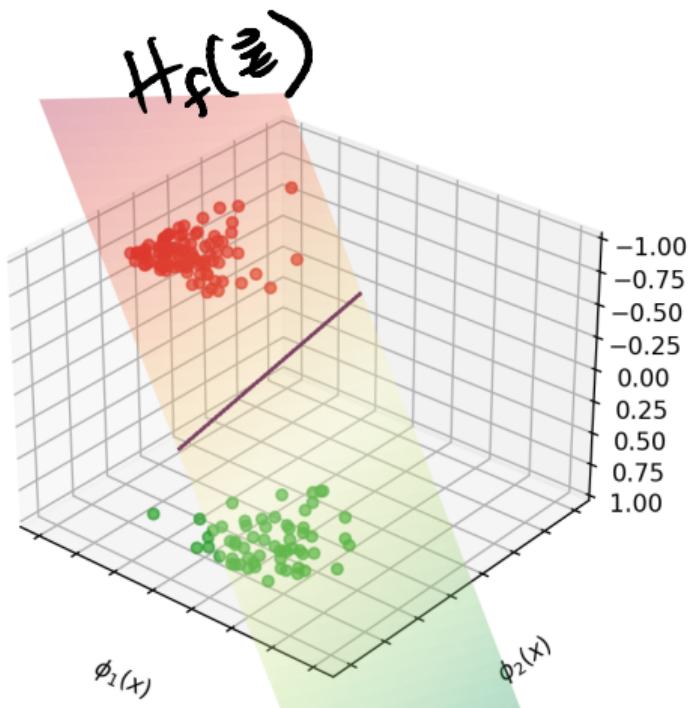


Decision Boundary in Feature Space²



²Fit by minimizing square loss

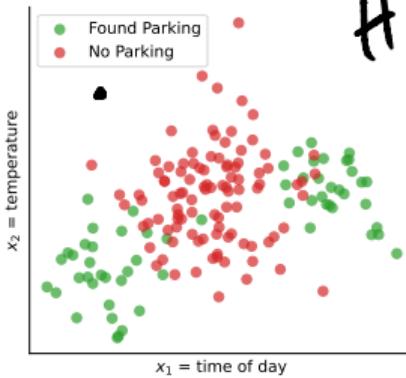
Prediction Surface in Feature Space



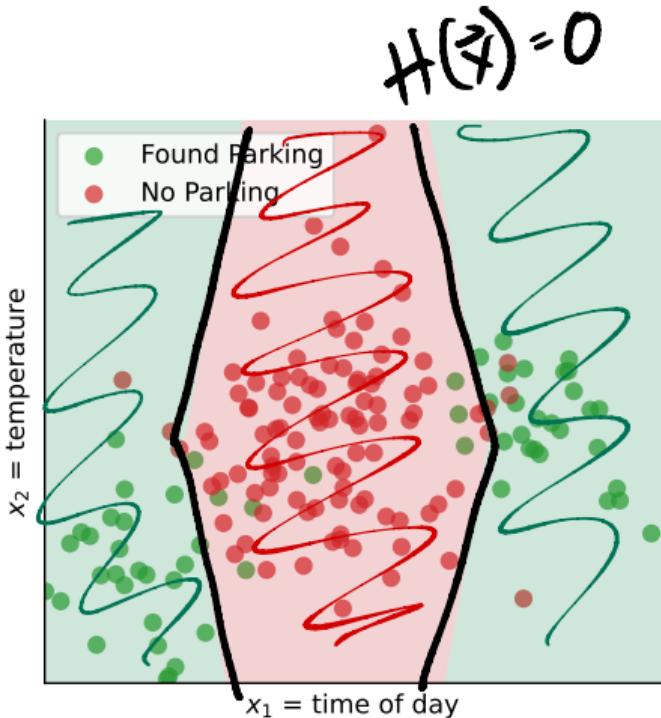
Exercise

- ▶ In the **original space**, what does the decision boundary look like?
- ▶ What does the prediction function surface look like?

$$H(\vec{x}) = H_f(\vec{\varphi}(\vec{x}))$$

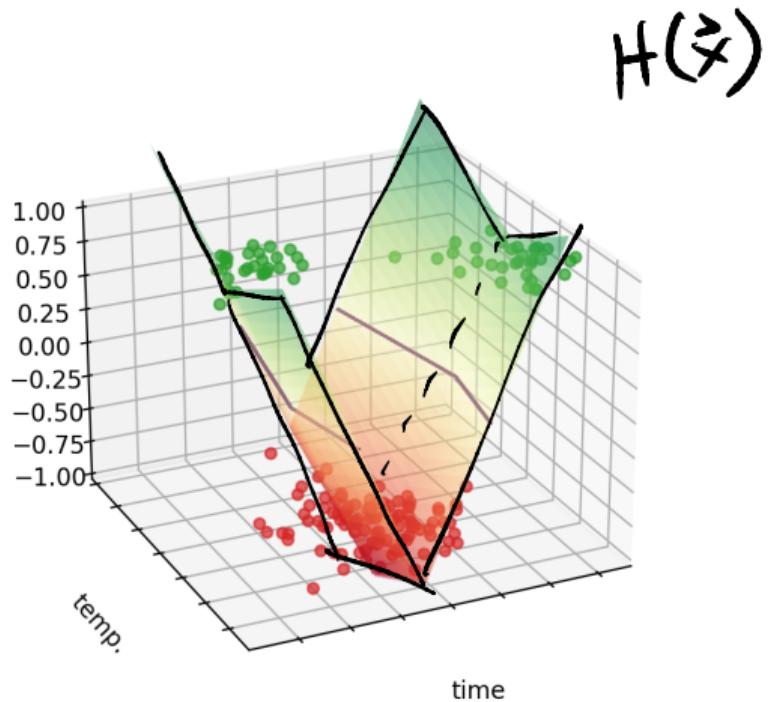


Decision Boundary in Original Space³



³Fit by minimizing square loss

Prediction Surface in Original Space

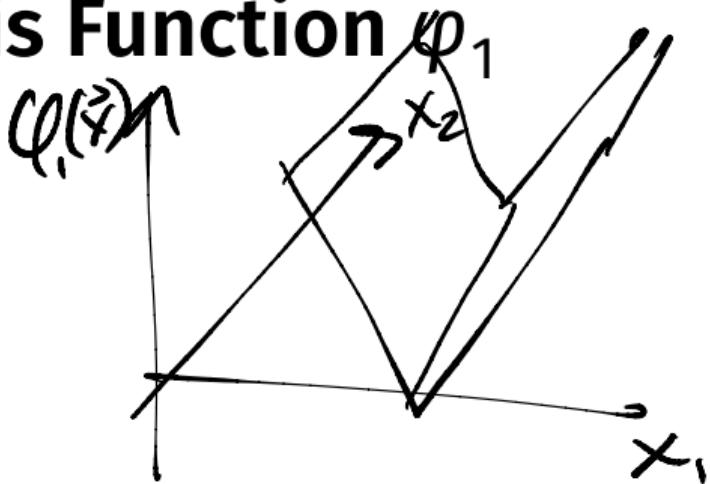
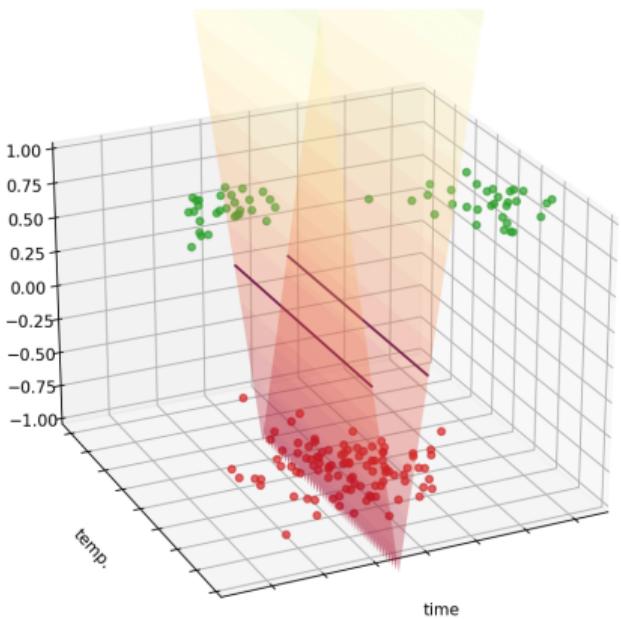


Insight

- ▶ H is a sum of basis functions, φ_1 and φ_2 .
 - ▶ $H(\vec{x}) = w_0 + w_1 \varphi_1(\vec{x}) + w_2 \varphi_2(\vec{x})$

The diagram shows the mathematical expression $H(\vec{x}) = w_0 + w_1 \varphi_1(\vec{x}) + w_2 \varphi_2(\vec{x})$. Above the term $w_1 \varphi_1(\vec{x})$, there is a brace underlining it and the term $w_2 \varphi_2(\vec{x})$, both labeled $f_1(\vec{x})$ and $f_2(\vec{x})$ respectively. This visually represents the function $H(\vec{x})$ as the sum of two separate surfaces, $f_1(\vec{x})$ and $f_2(\vec{x})$.
- ▶ The prediction surface is a sum of other surfaces.
- ▶ Each basis function is a “building block”.

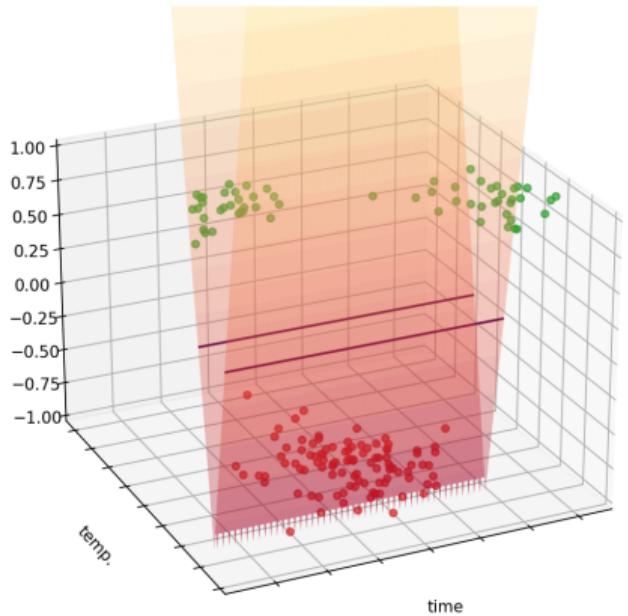
Visualizing the Basis Function



► $w_0 + w_1 |x_1 - \text{noon}|$

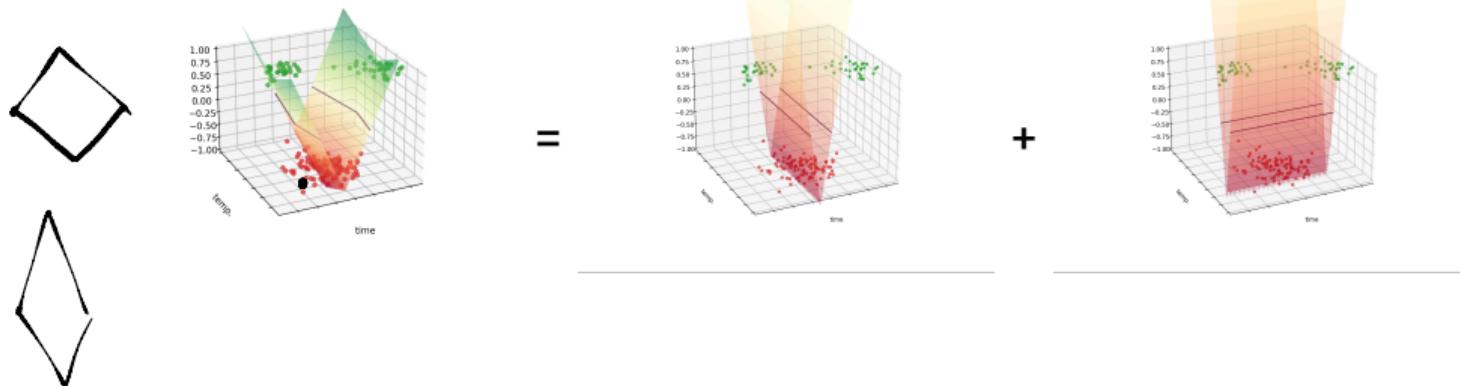
$$\varphi_1(\vec{x})$$

Visualizing the Basis Function φ_2



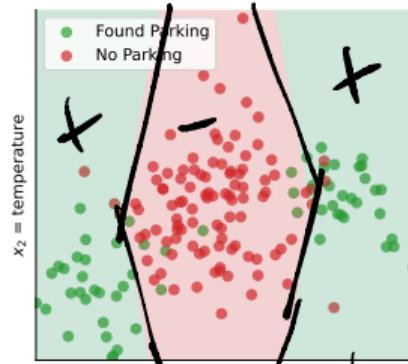
► $w_0 + w_2 |x_2 - 72^\circ|$

Visualizing the Prediction Surface



Exercise

The decision boundary has a single “pocket” where it is negative. Can it have more than one, assuming we use basis functions of the same form? What if we use more than two basis functions?



$$x_i - c$$

$$|x_i - c|$$

Answer: No!

- ▶ Recall: the sum of **convex** functions is **convex**.
- ▶ Each of our basis functions is convex.
- ▶ So the prediction surface will be convex, too.
- ▶ Limited in what patterns they can classify.

View: Function Approximation



- ▶ Find a function that is ≈ 1 near green points and ≈ -1 near red points.

What's Wrong?

- ▶ We've discovered how to learn non-linear patterns using linear prediction functions.
 - ▶ Use non-linear basis functions to map to a feature space.
- ▶ Something should bug you, though...

DSC 140B

Representation Learning

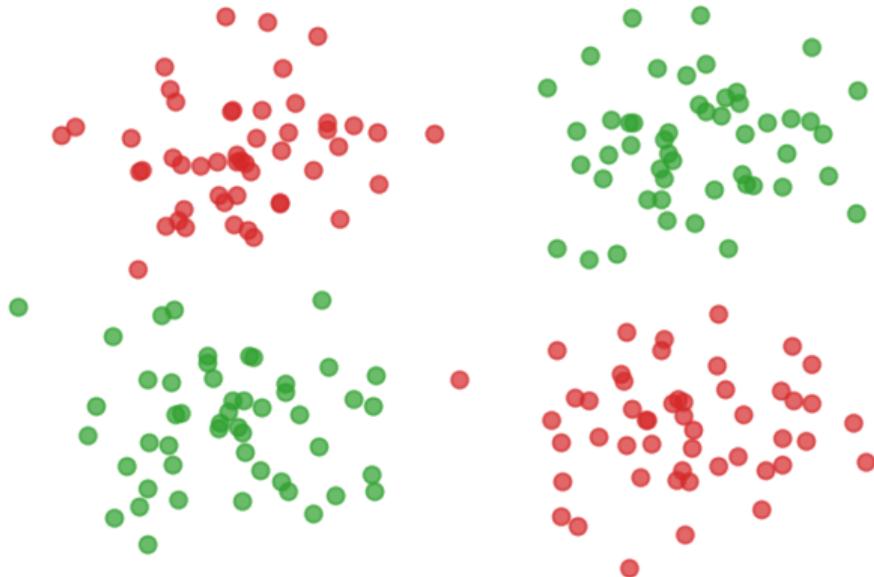
Lecture 11 | Part 5

Radial Basis Functions

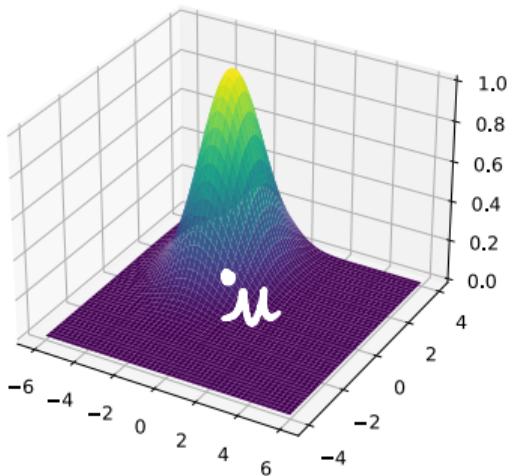
Choosing Basis Functions

- ▶ Our previous basis functions have limitations.
- ▶ They are convex: prediction surface can only have one negative/positive region.
- ▶ They diverge $\rightarrow \infty$ away from their centers.
 - ▶ They get more “confident”?

Example



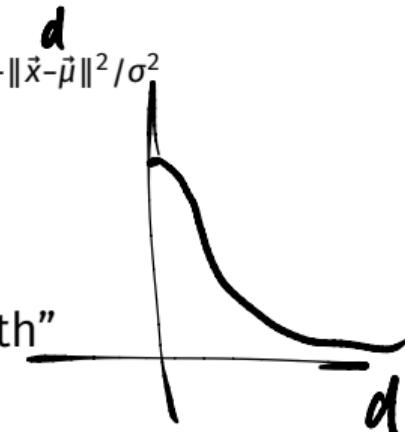
Gaussian Basis Functions



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

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

- ▶ $\vec{\mu}$ 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 $\vec{\mu}^{(1)}, \dots, \vec{\mu}^{(2)}, \dots, \vec{\mu}^{(d')}$
- ▶ 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

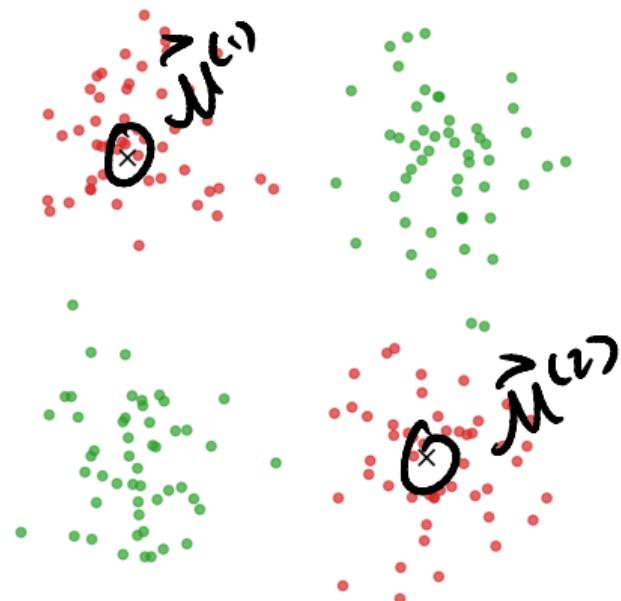
- ▶ For any feature vector $\vec{x} \in \mathbb{R}^d$, map to vector $\vec{\varphi}(\vec{x}) \in \mathbb{R}^{d'}$.
 - ▶ φ_1 : “similarity” of \vec{x} to $\vec{\mu}^{(1)}$
 - ▶ φ_2 : “similarity” of \vec{x} to $\vec{\mu}^{(2)}$
 - ▶ ...
 - ▶ $\varphi_{d'}$: “similarity” of \vec{x} to $\vec{\mu}^{(d')}$
- ▶ Train linear classifier in this new representation.
 - ▶ E.g., by minimizing expected square loss.

Exercise

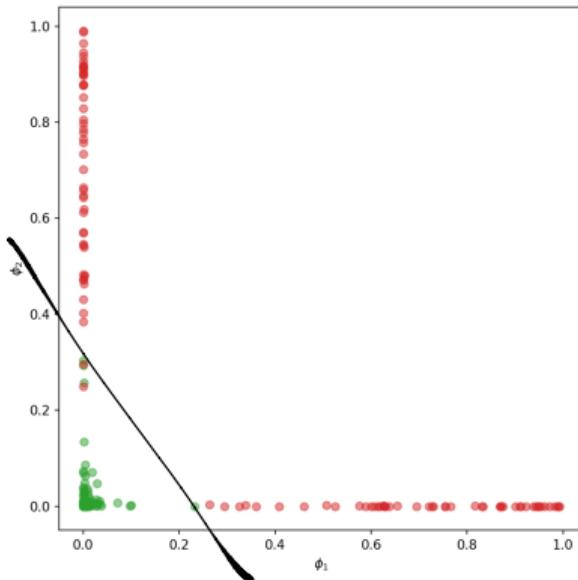
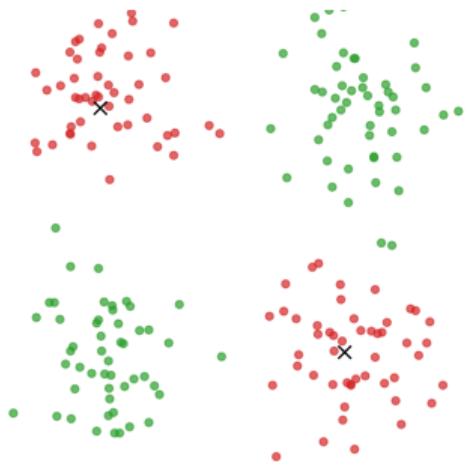
How many Gaussian basis functions would you use, and where would you place them to create a new representation for this data?



Placement



Feature Space



Prediction Function

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

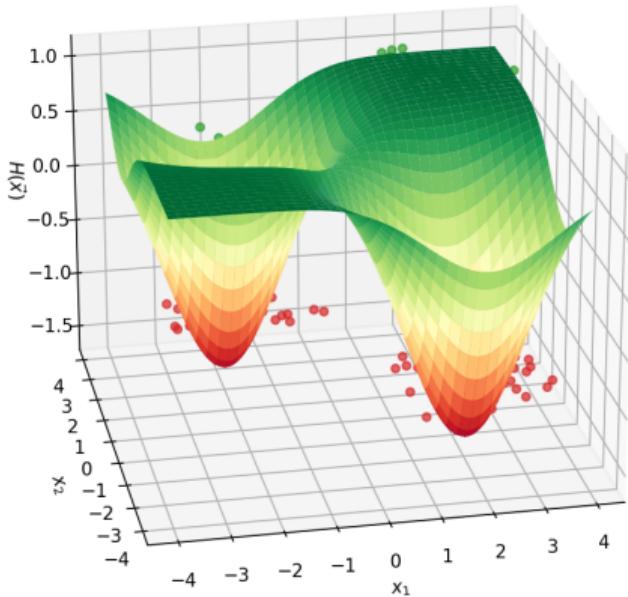
$$\begin{aligned} H(\vec{x}) &= w_0 + w_1 \varphi_1(\vec{x}) + w_2 \varphi_2(\vec{x}) + \dots \\ &= w_0 + w_1 e^{-\|\vec{x}-\vec{\mu}_1\|^2/\sigma^2} + w_2 e^{-\|\vec{x}-\vec{\mu}_2\|^2/\sigma^2} + \dots \end{aligned}$$

Exercise

What does the surface of the prediction function look like?

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

Prediction Function Surface

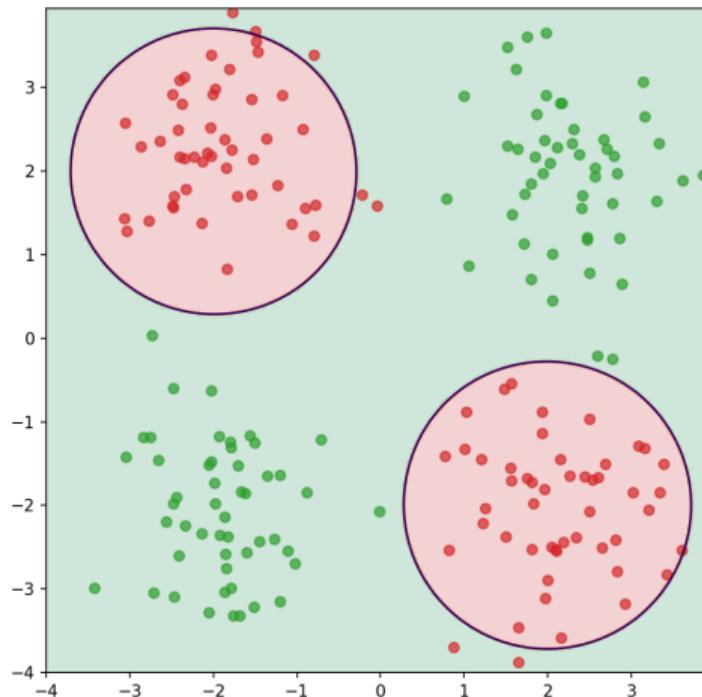


$$H(\vec{x}) = w_0 + w_1 e^{-\|\vec{x} - \vec{\mu}_1\|^2 / \sigma^2} + w_2 e^{-\|\vec{x} - \vec{\mu}_2\|^2 / \sigma^2}$$

An Interpretation

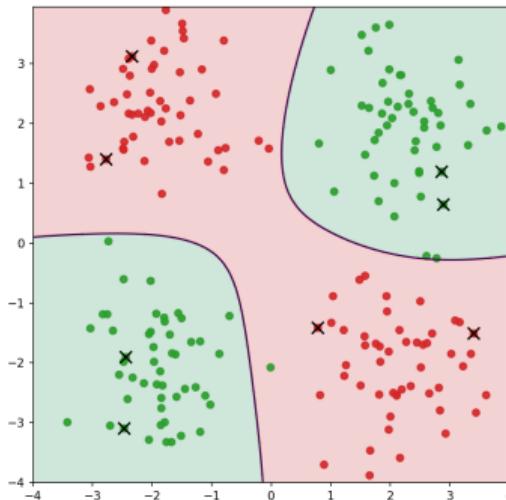
- ▶ Basis function φ_i makes a “bump” in surface of H
- ▶ w_i adjusts the “prominance” of this bump

Decision Boundary

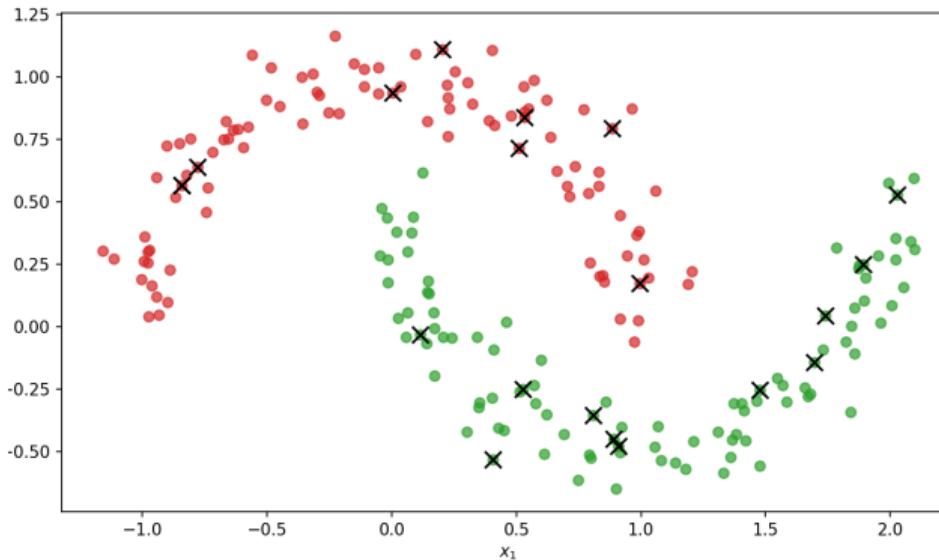


More Features

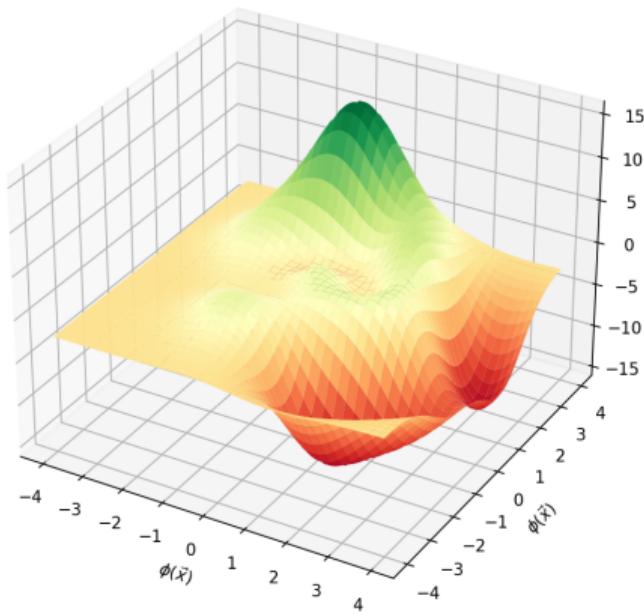
- ▶ By increasing number of basis functions, we can make more complex decision surfaces.



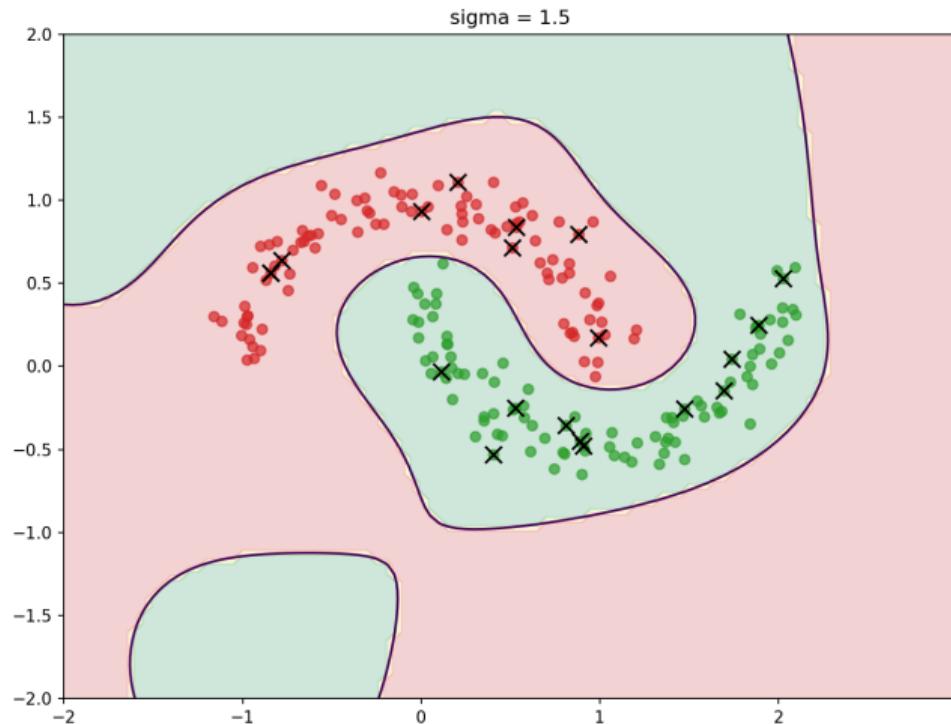
Another Example



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 12 | 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

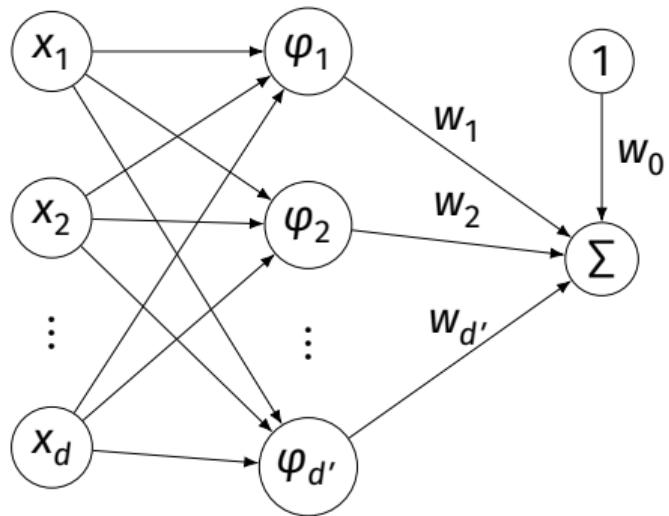
1. Start with data in original space, \mathbb{R}^d .
2. Choose some basis functions, $\varphi_1, \varphi_2, \dots, \varphi_{d'}$
3. Map each data point to **feature space** $\mathbb{R}^{d'}$:

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

4. Fit linear prediction function in new space:

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

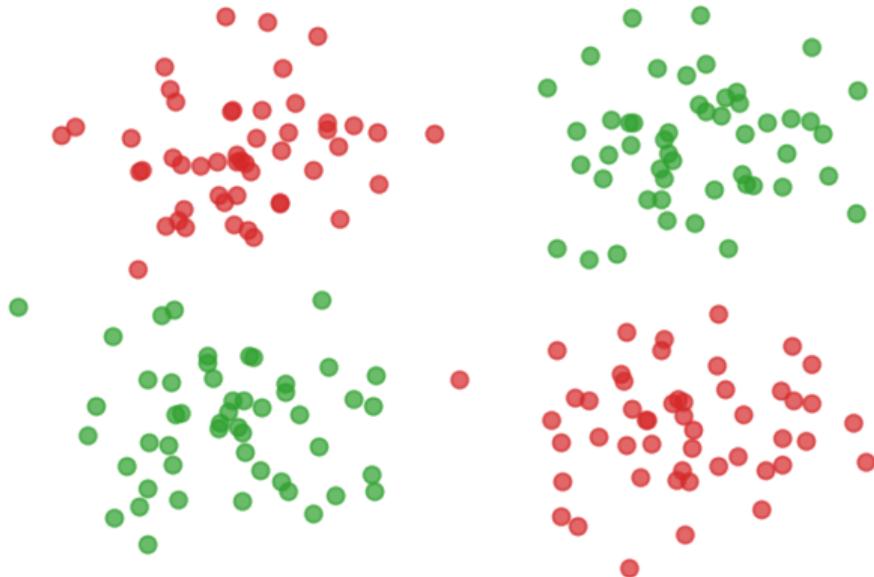
$$H(\vec{x}) = w_0 + w_1 \varphi_1(\vec{x}) + w_2 \varphi_2(\vec{x})$$



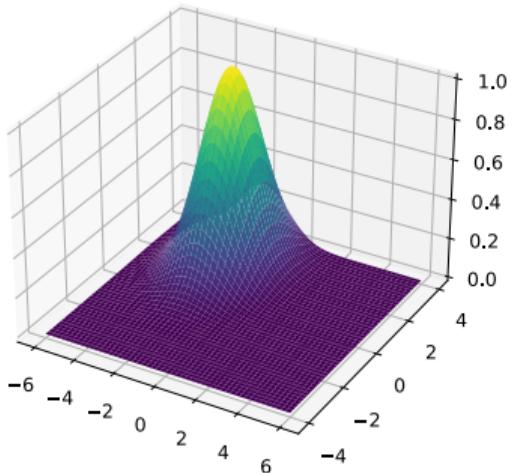
Generic Basis Functions

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

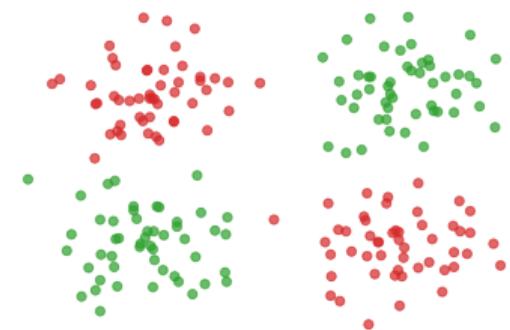
$$\varphi_i(\vec{x}) = e^{-\|\vec{x} - \vec{\mu}^{(i)}\|^2 / \sigma_i^2}$$

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.

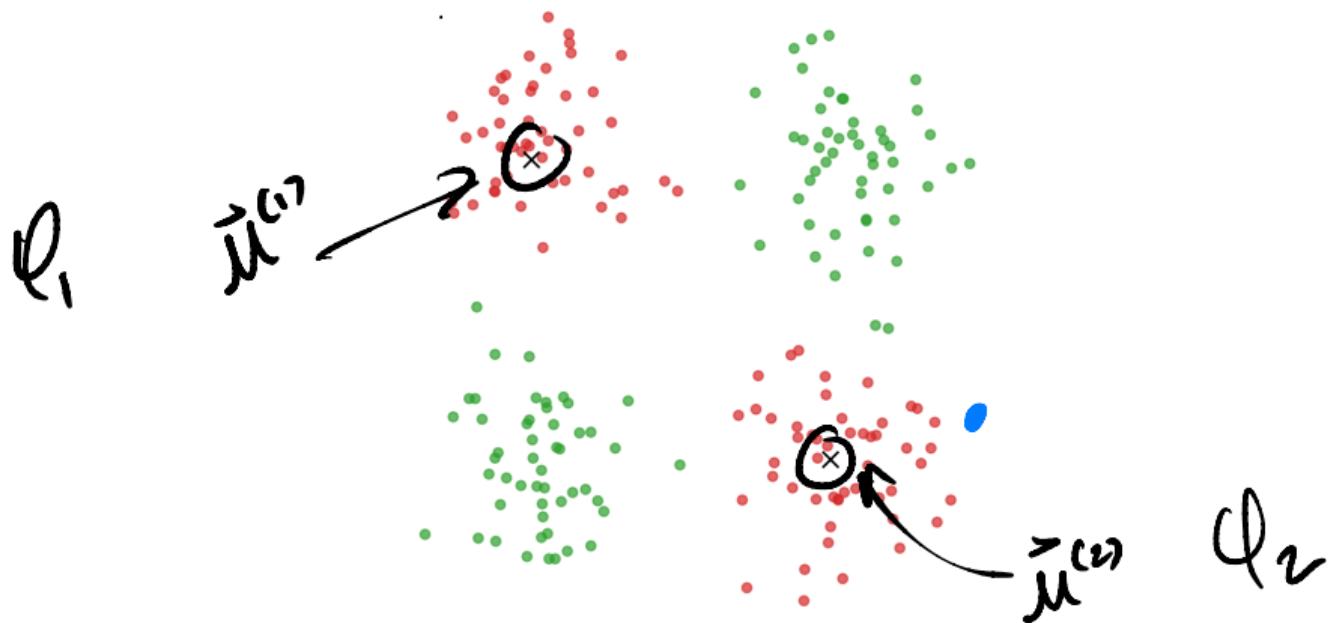
Exercise

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



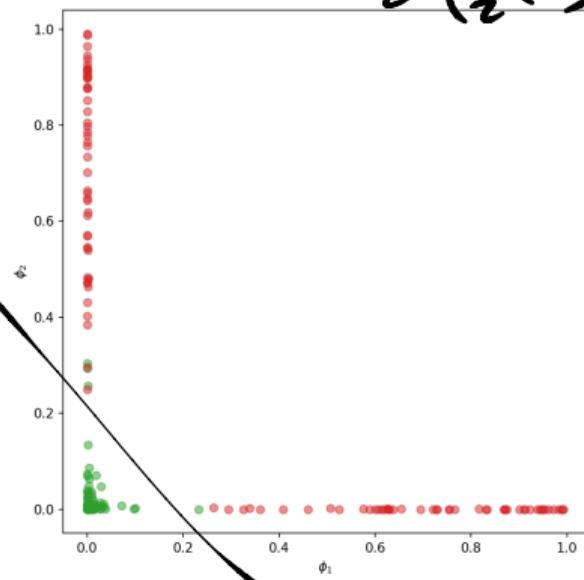
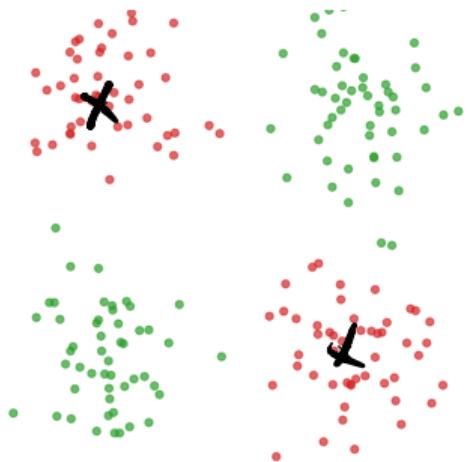
Placement

$$\vec{x} \mapsto (\varrho_1(\vec{x}), \varrho_2(\vec{x}))$$



Feature Space

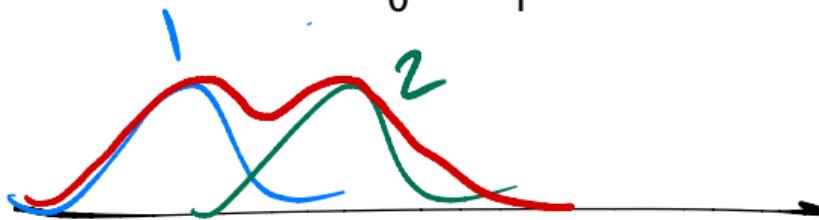
$$H(\vec{x}) = w_0 + w_1 \varphi_1(\vec{x}) + w_2 \varphi_2(\vec{x})$$



Prediction Function

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

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

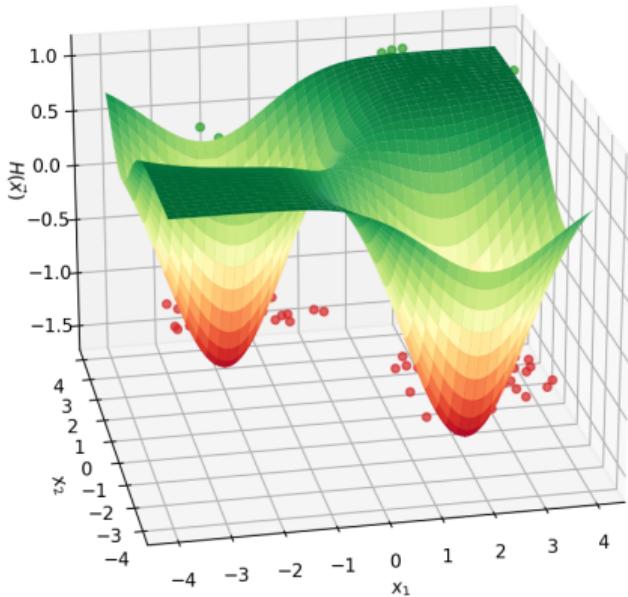


Exercise

What does the surface of the prediction function look like?

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

Prediction Function Surface



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

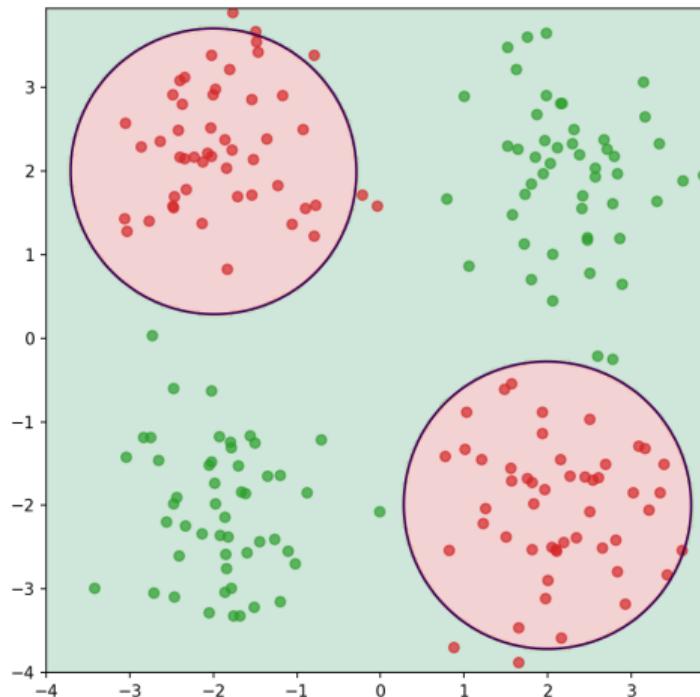
An Interpretation

Gaussian Radial

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

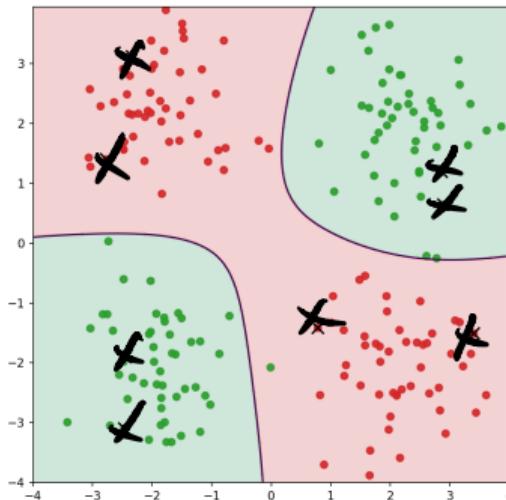
Decision Boundary

$$H(\vec{x}) = 0$$

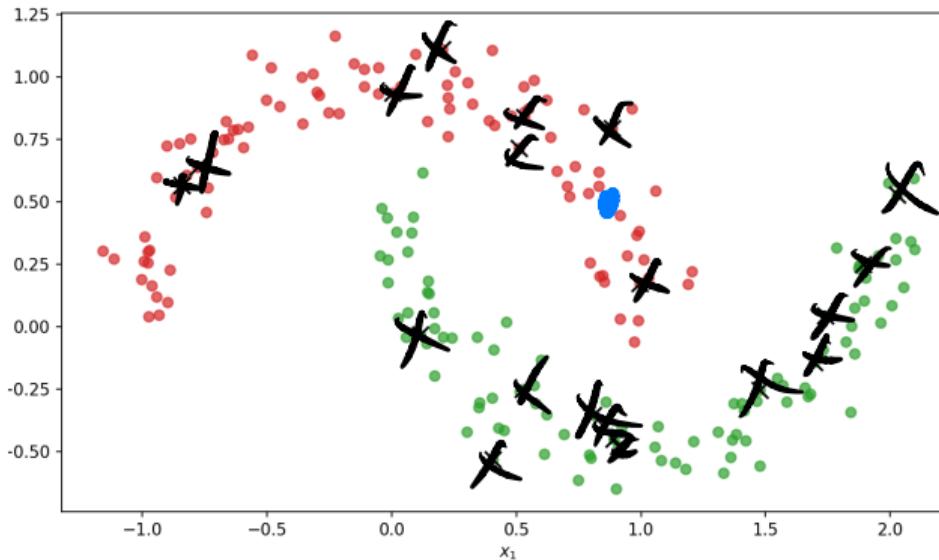


More Features

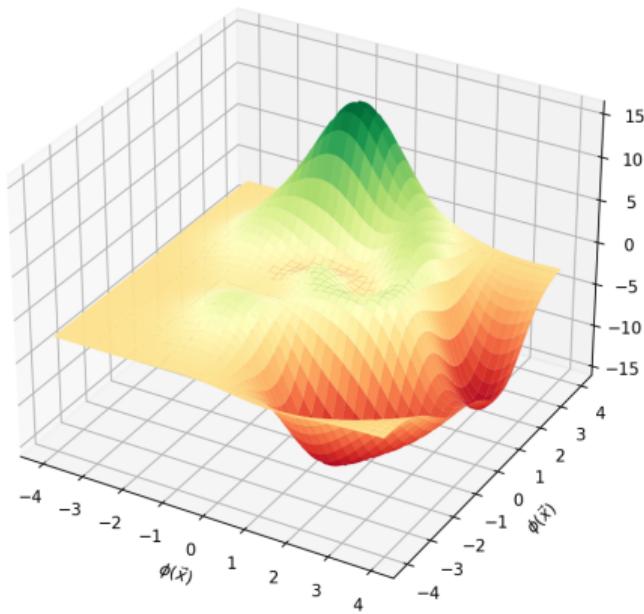
- ▶ By increasing number of basis functions, we can make more complex decision surfaces.



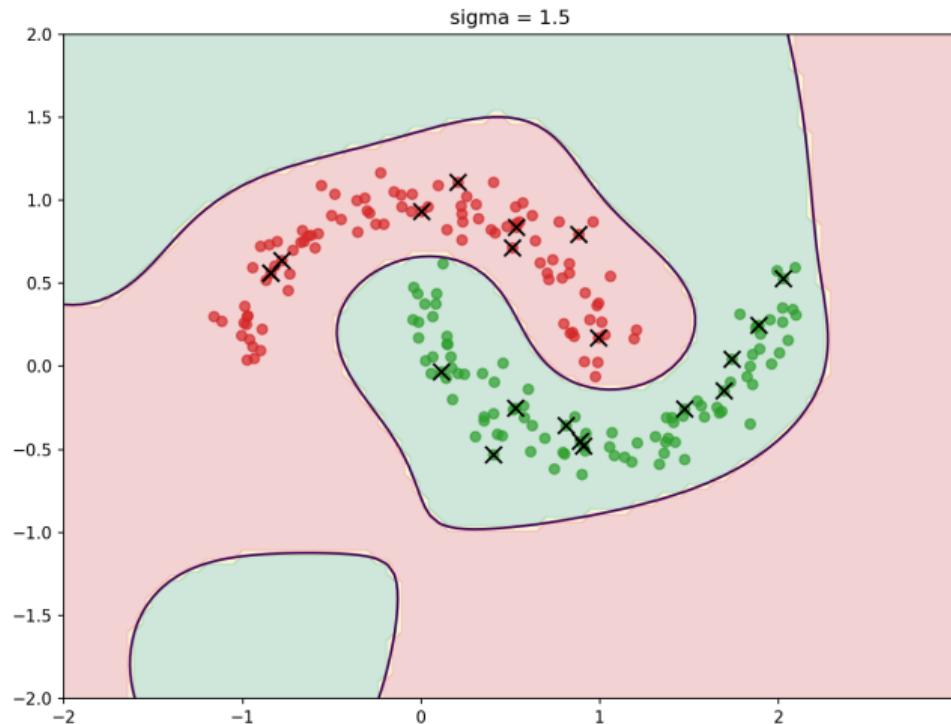
Another Example



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

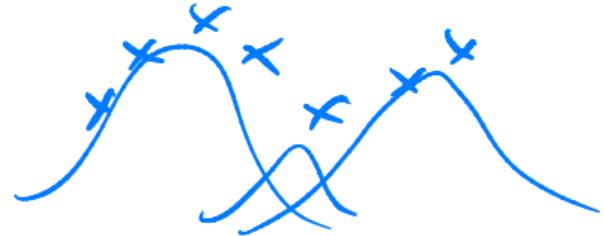
DSC 140B

Representation Learning

Lecture 12 | Part 2

Radial Basis Function Networks

Recap

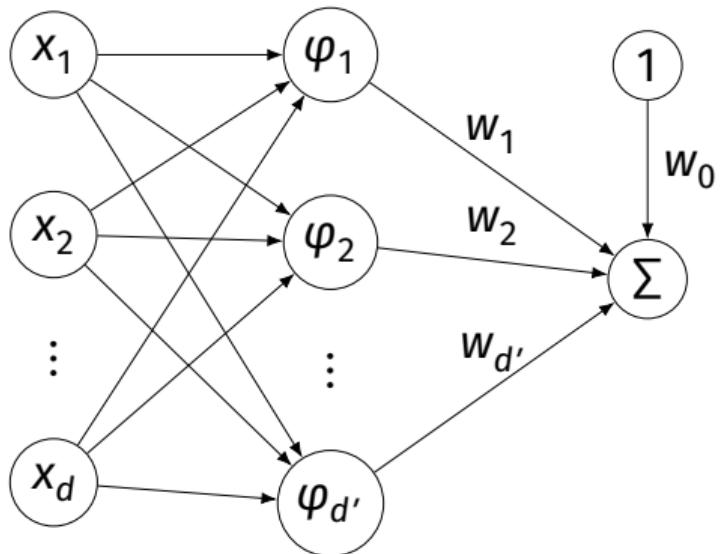


1. Choose basis functions, $\varphi_1, \dots, \varphi_{d'}$
2. Transform data to new representation:
$$\vec{x} \mapsto (\varphi_1(\vec{x}), \varphi_2(\vec{x}), \dots, \varphi_{d'}(\vec{x}))^T$$
3. Train a linear classifier in this new space:

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

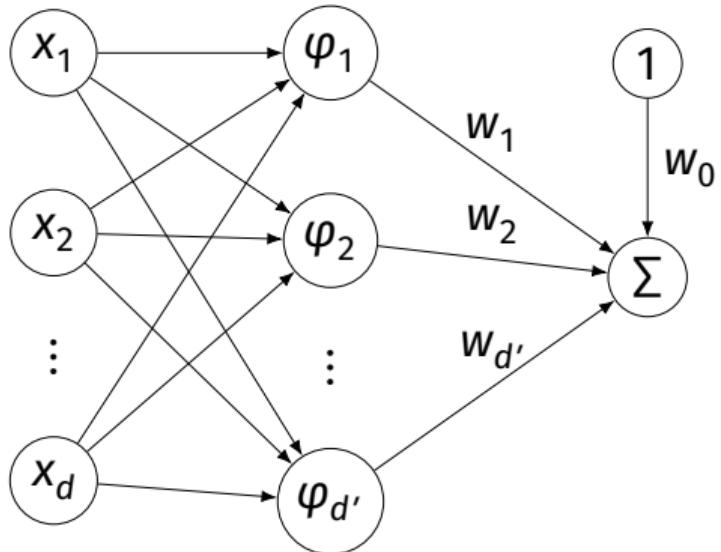
The Model

- 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?

$$C(w_0, w_1, \dots, w_d, \mu^{(1)}, \mu^{(2)}, \dots, \sigma_d)$$

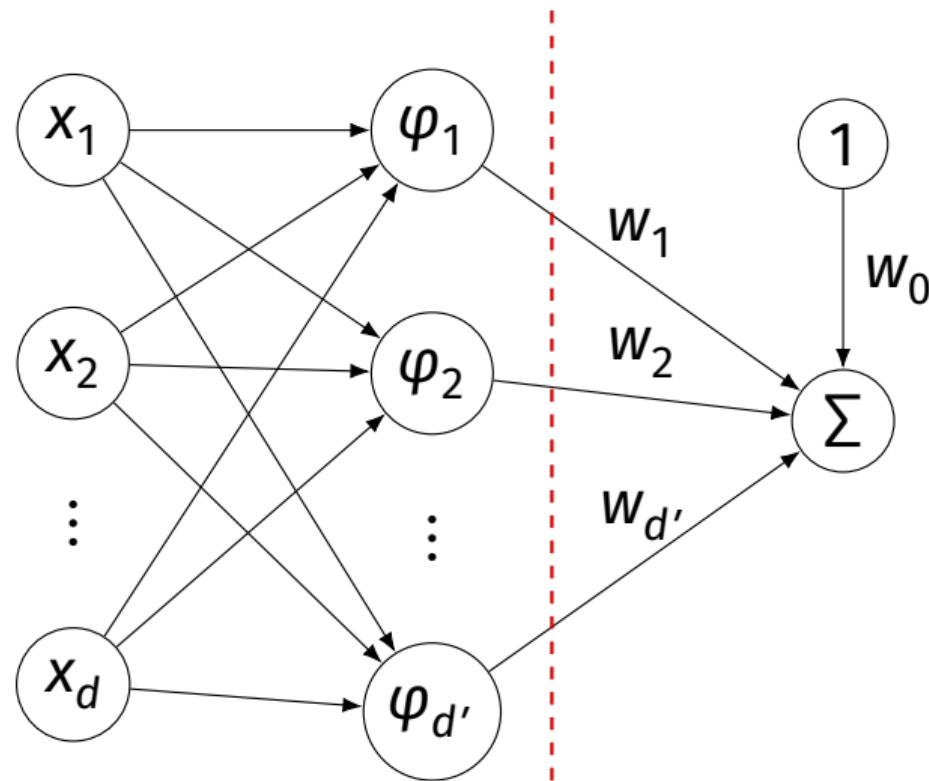
First Idea

- ▶ 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 *somewhat*.
 - ▶ 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 *somewhat*.
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

DSC 140B

Representation Learning

Lecture 12 | Part 3

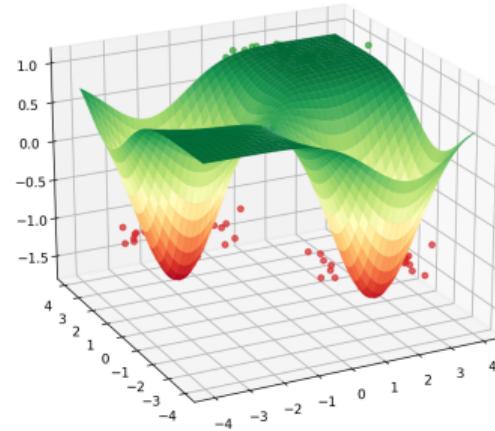
Choosing RBF Locations

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”.

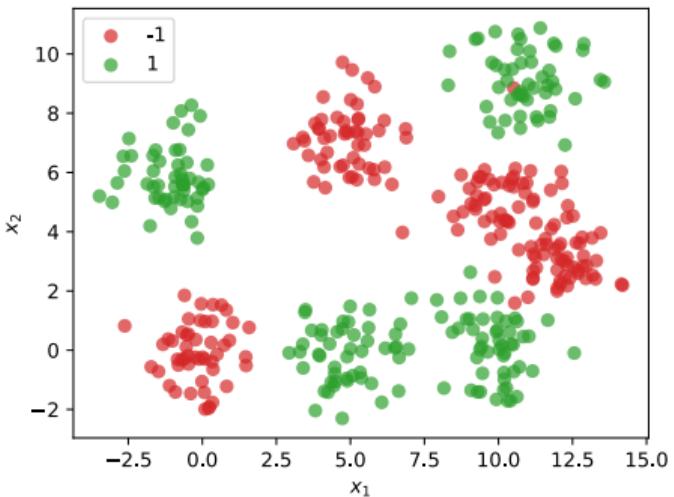


$$H(\vec{x}) = w_0 + w_1 e^{-\|\vec{x} - \vec{\mu}_1\|^2/\sigma^2} + w_2 e^{-\|\vec{x} - \vec{\mu}_2\|^2/\sigma^2}$$

Choosing Centers



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



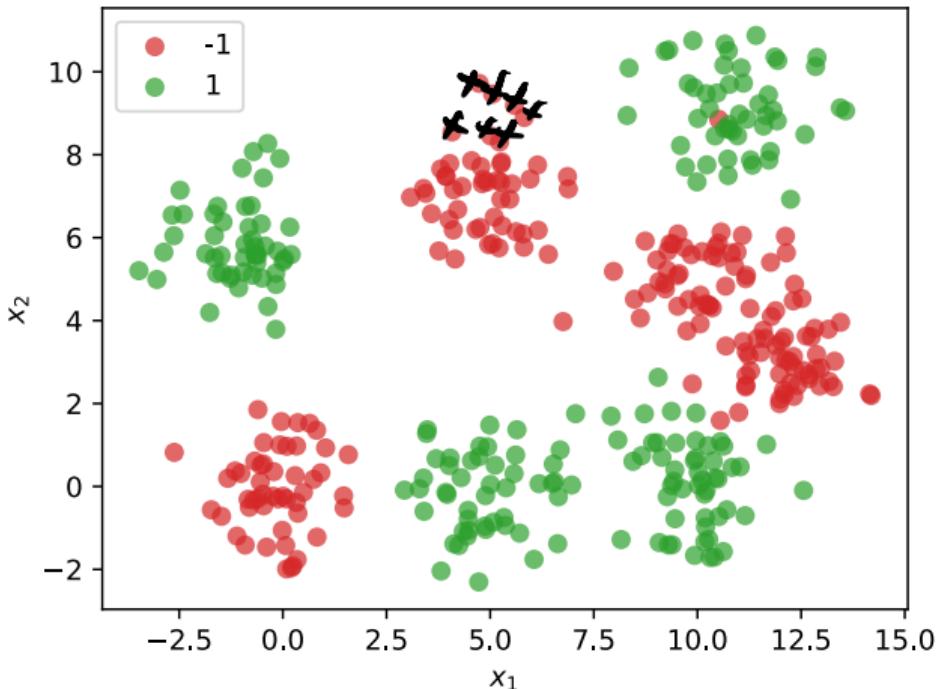
Approaches

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

Approach #1: Every Data Point as a Center

ℓ_i

$$\vec{x} \mapsto (\ell_1(\vec{x}), \dots, \ell_n(\vec{x}))$$



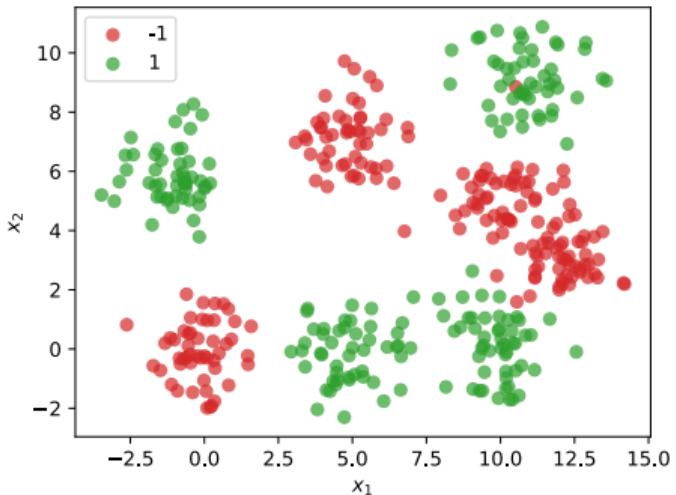
Dimensionality

- ▶ 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



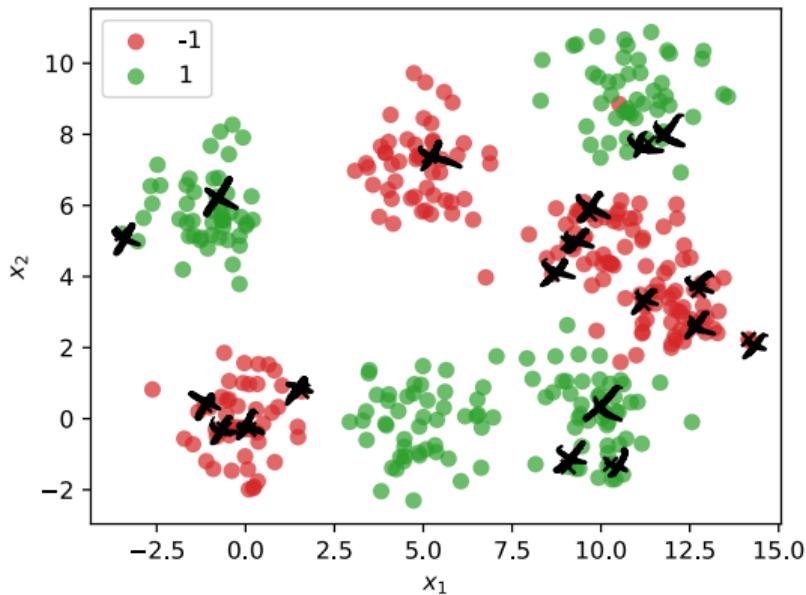
Computational Cost

- ▶ 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.

$$\begin{aligned} H(\vec{x}) &= \sum_{i=1}^n \alpha_i K(\vec{x}, \vec{x}^{(i)}) \\ &= \alpha_1 K(\vec{x}, \vec{x}^{(1)}) + \\ &\quad \alpha_2 K(\vec{x}, \vec{x}^{(2)}) + \\ &\quad \vdots \end{aligned}$$

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 12 | Part 4

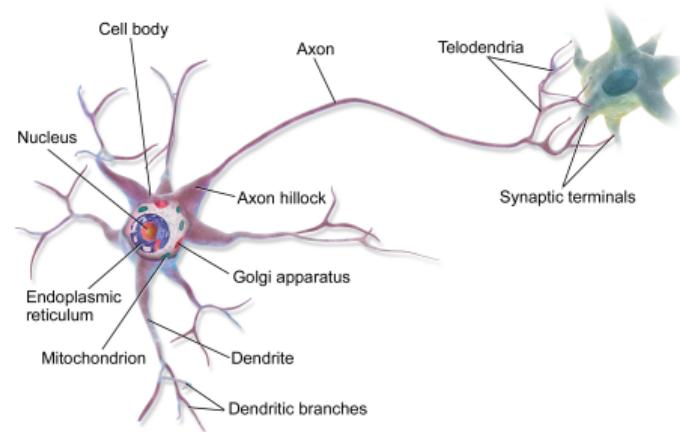
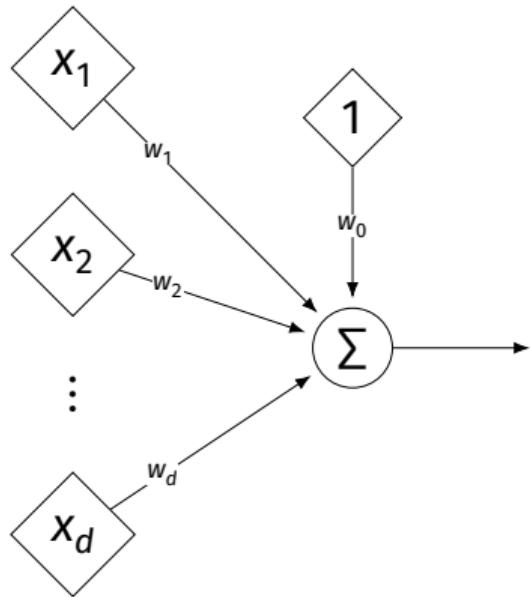
Neural Networks

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.

Linear Models

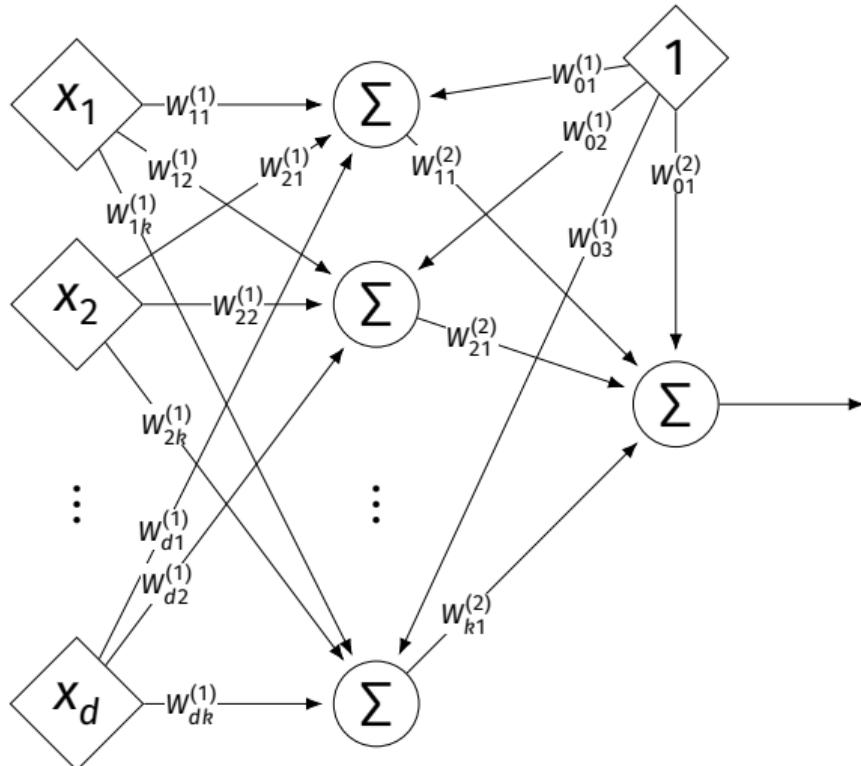
$$H(\vec{x}) = w_0 + w_1x_1 + \dots + w_dx_d$$



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

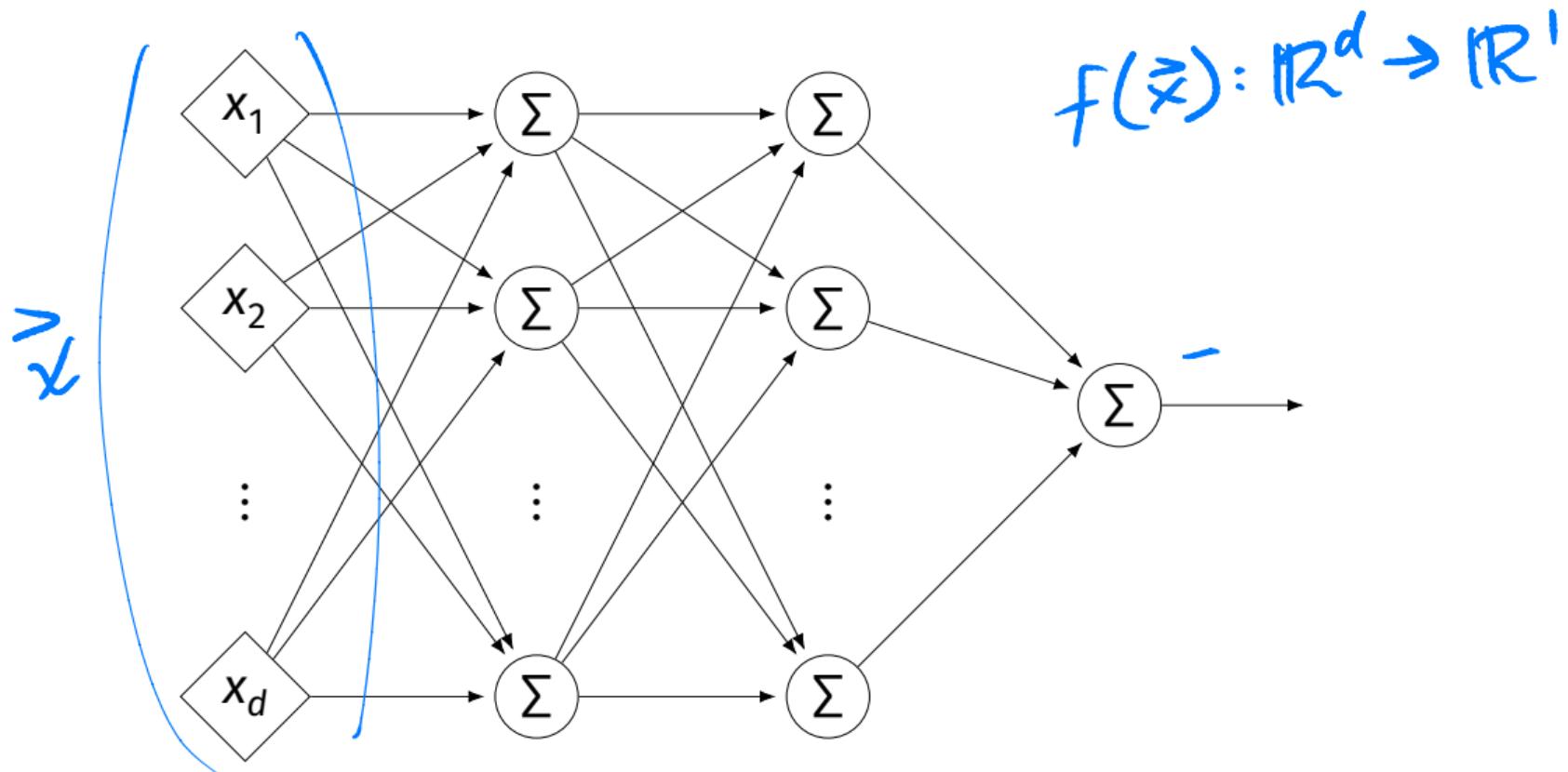
Architecture

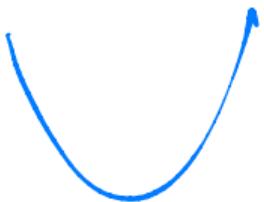
- ▶ Neurons are organized into **layers**.
 - ▶ **Input layer**, **output layer**, and **hidden layers**.
- ▶ Number of cells in input layer determined by dimensionality of input feature vectors.
- ▶ Number of cells in hidden layer(s) is determined by you.
- ▶ Output layer can have >1 neuron.

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)





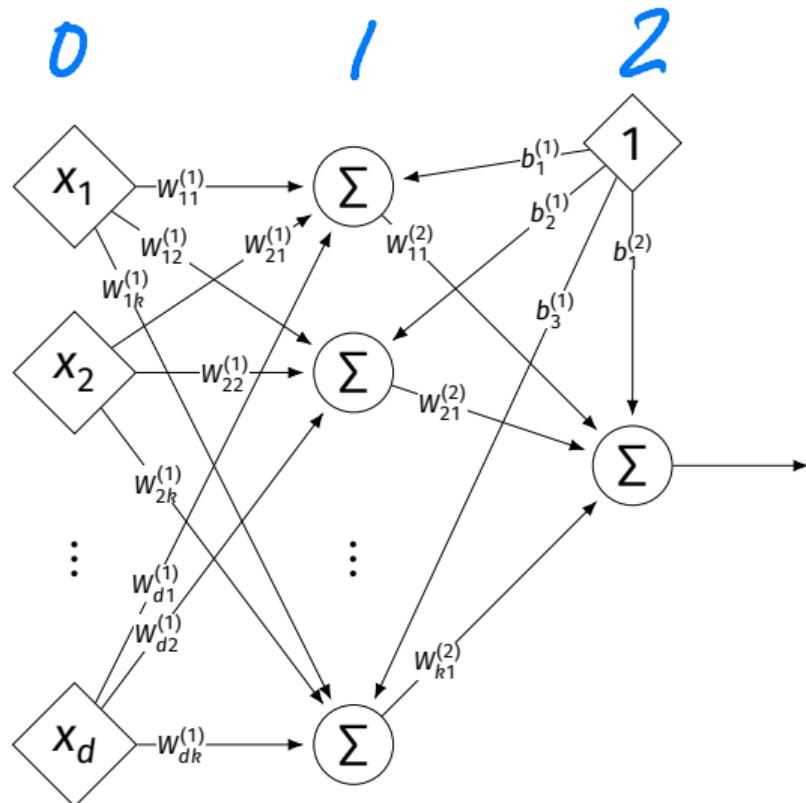
Network Weights

- ▶ 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

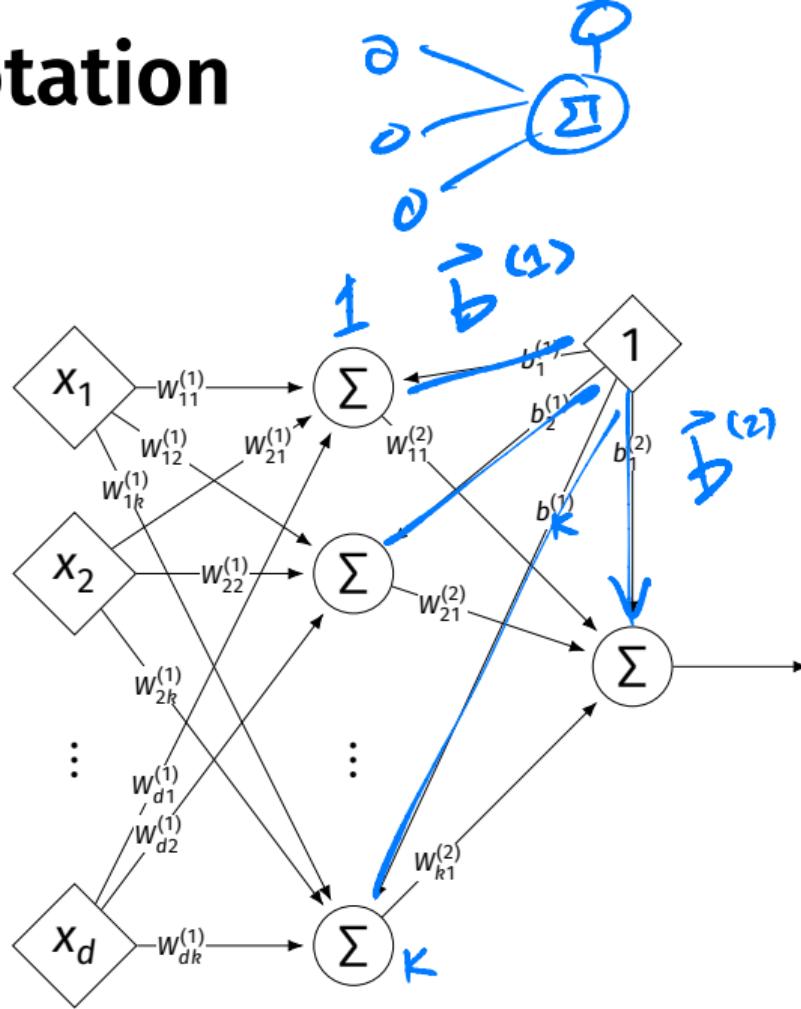
$W_{ij}^{(l)}$

- ▶ 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.



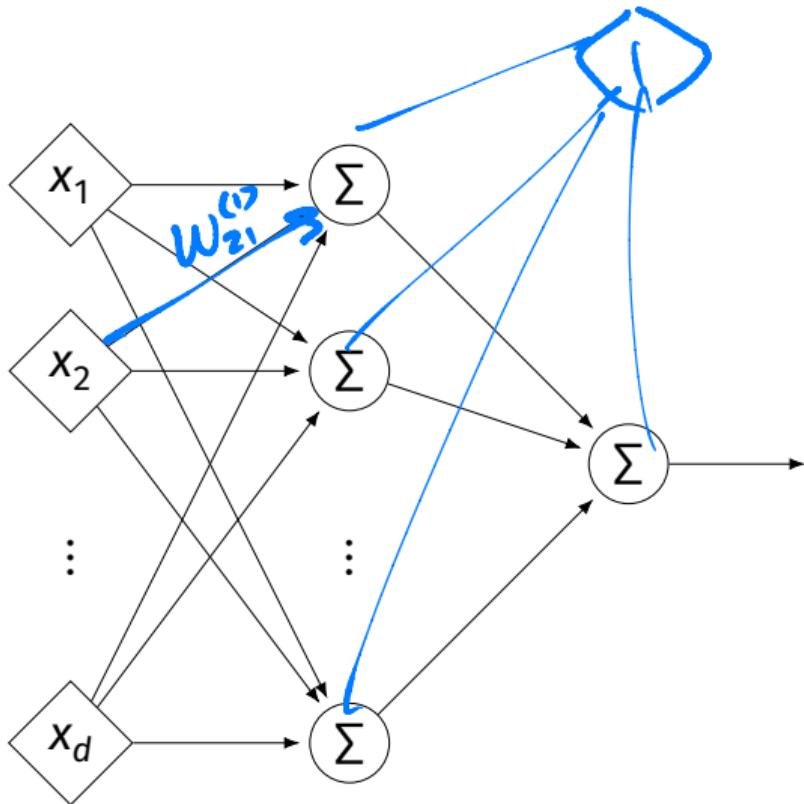
Notation

- ▶ 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: $\vec{b}^{(i)}$

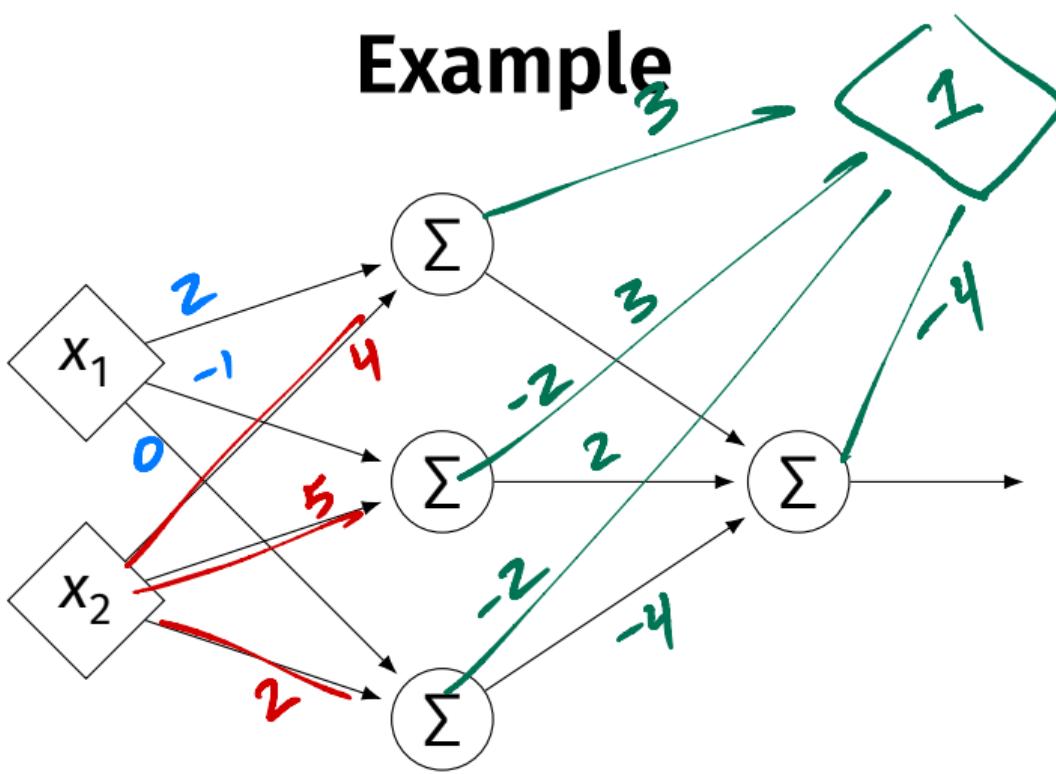


Notation

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



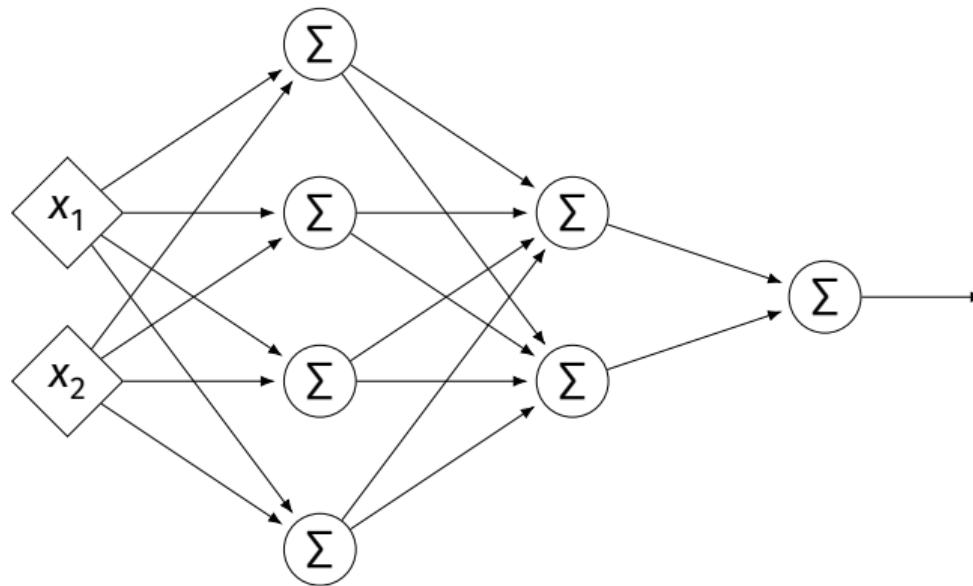
Example 3



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

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

Example



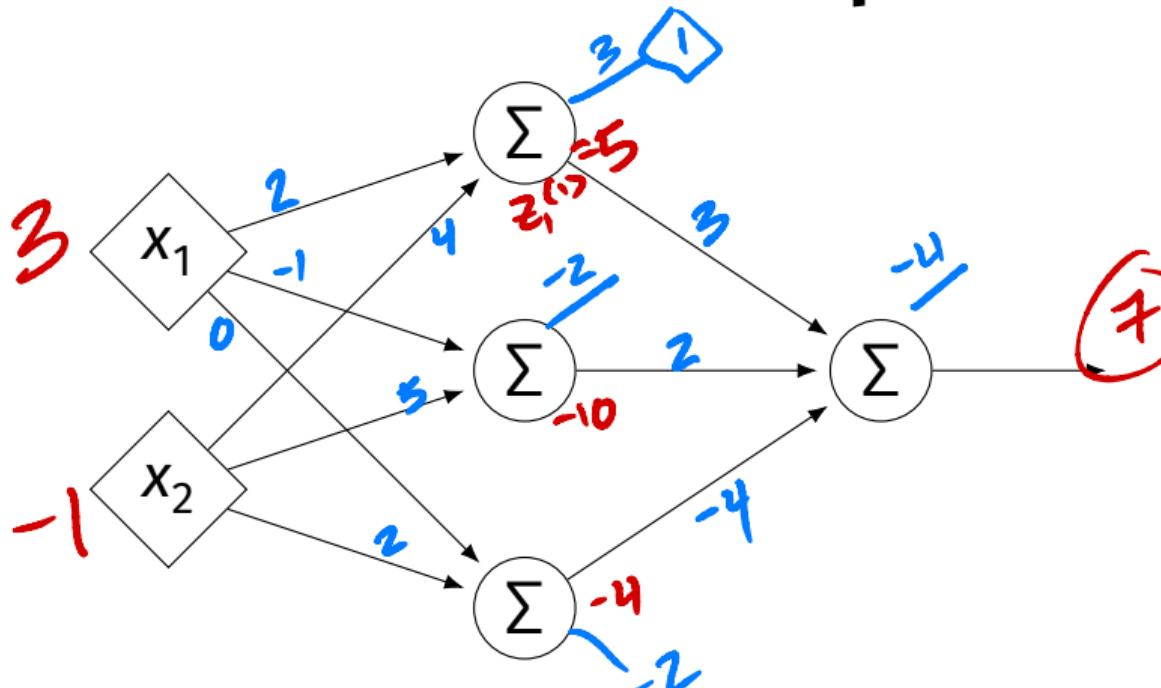
$$W^{(1)} = \begin{pmatrix} 2 & -1 & -3 & 0 \\ 4 & 5 & -7 & 2 \end{pmatrix} \quad W^{(2)} = \begin{pmatrix} 1 & 2 \\ -4 & 3 \\ -6 & -2 \\ 3 & 4 \end{pmatrix} \quad W^{(3)} = \begin{pmatrix} -1 & 5 \end{pmatrix}$$

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

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



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

$$W^{(2)} = \begin{pmatrix} 3 \\ 2 \\ -4 \end{pmatrix}$$

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

$$\vec{b}^{(2)} = (-4)^T$$

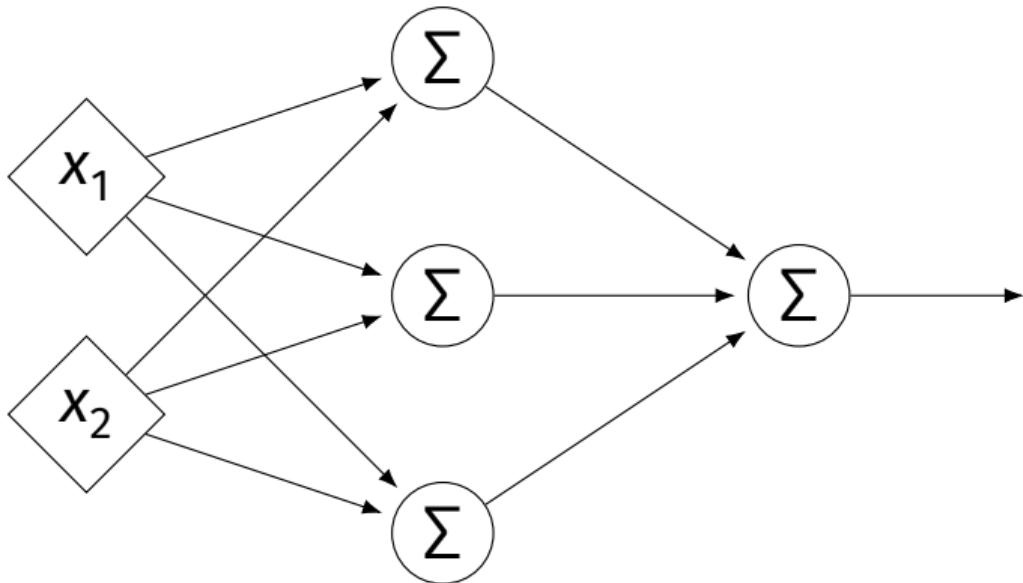
$$= 15 - 20 + 16 \\ - 4 = 7$$

- ▶ $\vec{x} = (3, -1)^T$
- ▶ $(3)(2) + (-1)(4) + 3 = 5$
- ▶ $z_2^{(1)} = (3)(-1) + (-1)(5) + (-2) = -10$
- ▶ $z_3^{(1)} = (-2) + (-2) = -4$
- ▶ $z_1^{(2)} = (5)(3) + (-10)(2) + (-4)(-4)$

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



► $\vec{x} = (3, -1)^T$

► $z_1^{(1)} =$

► $z_2^{(1)} =$

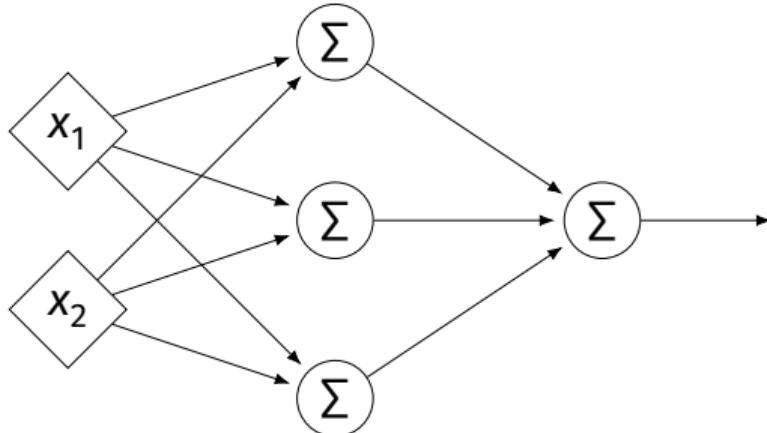
► $z_3^{(1)} =$

► $z_1^{(2)} =$

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

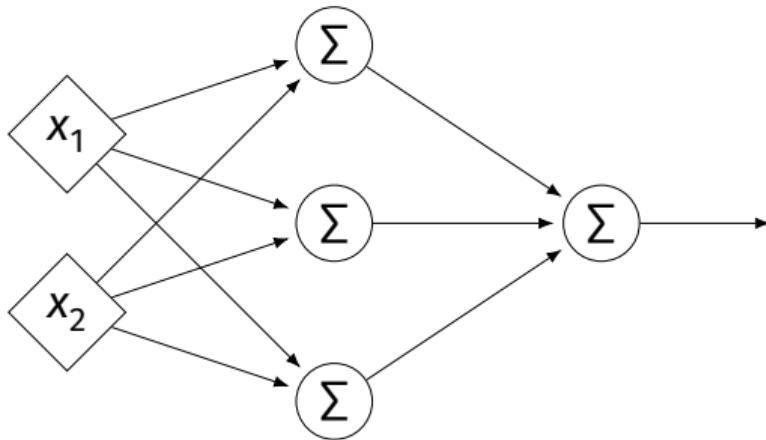
Each Layer is a Function

- ▶ 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.



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

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.

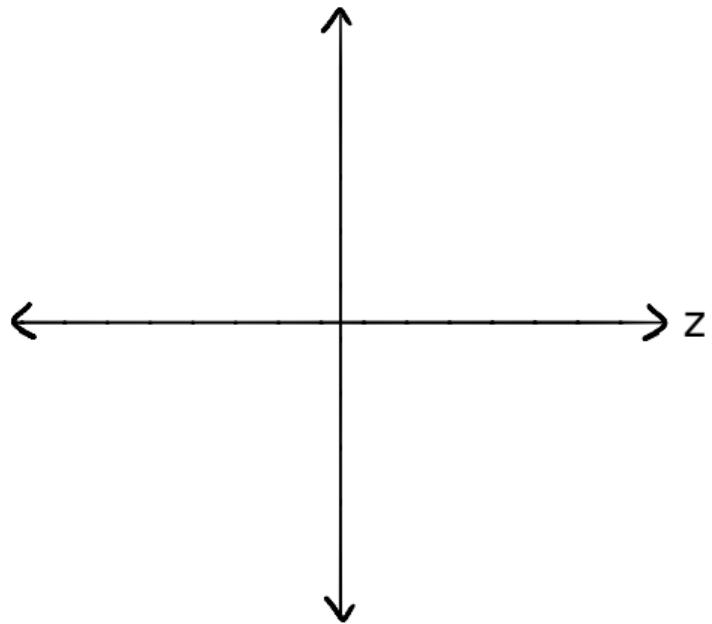
Idea

- ▶ 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.

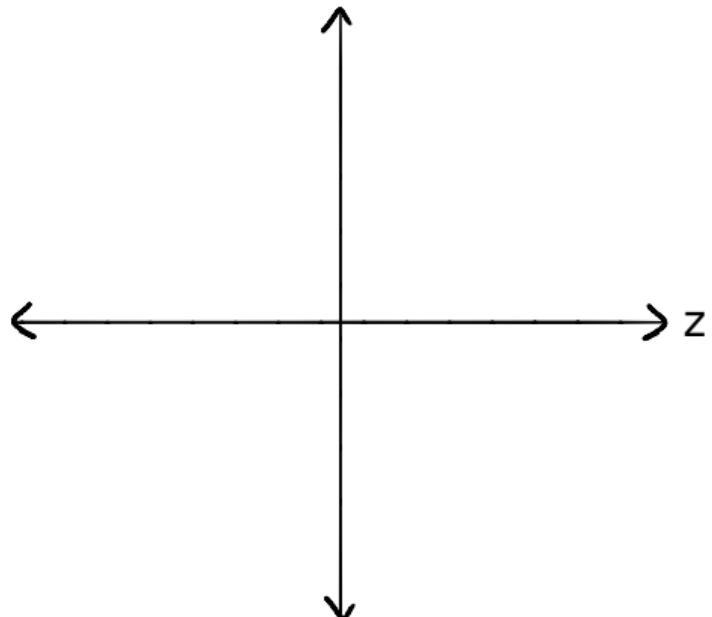
$$\sigma(z) = z$$



Sigmoid Activation

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

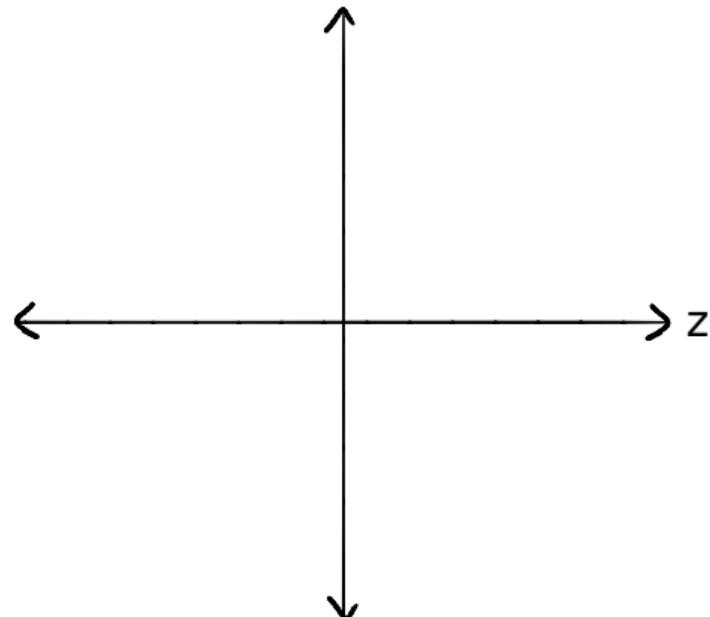
$$\sigma(z) = \frac{1}{1 + e^{-z}}$$



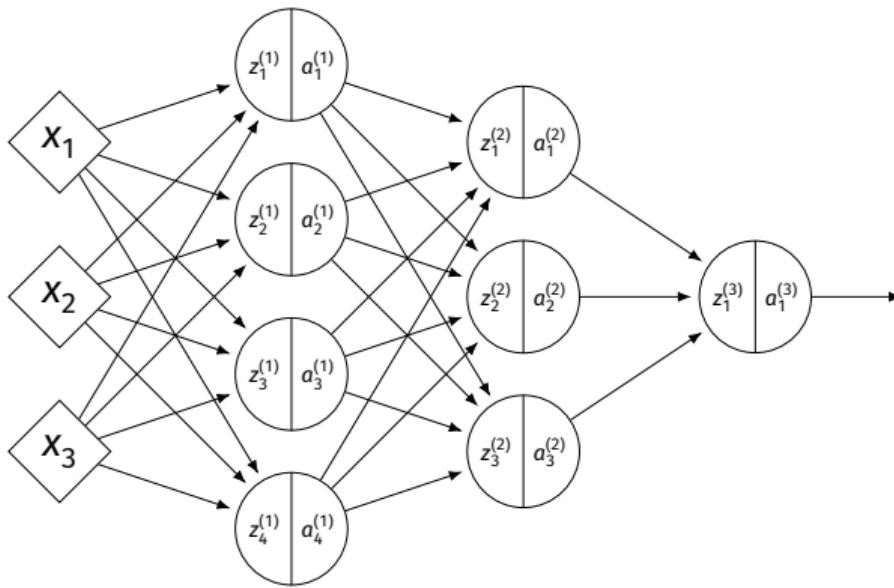
ReLU Activation

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

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

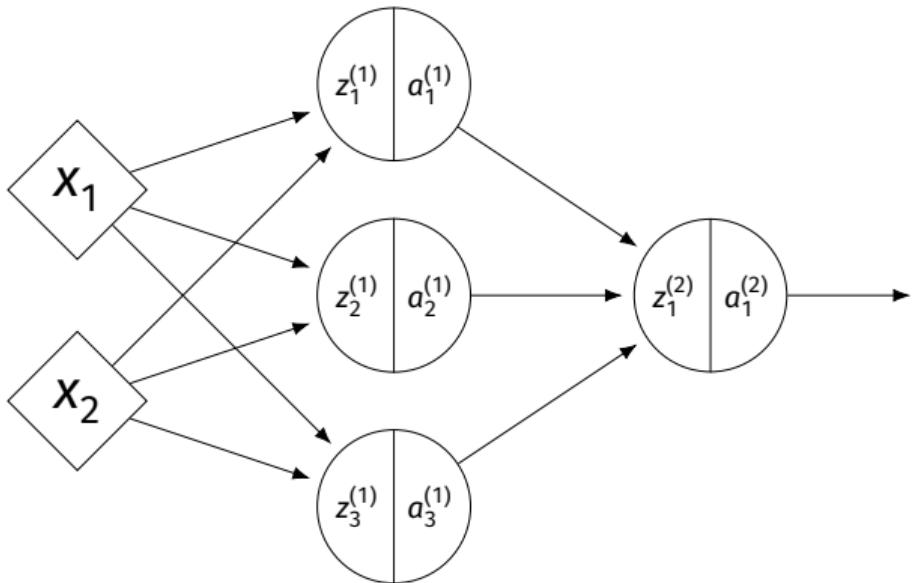


Notation



- ▶ $z_j^{(i)}$ 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.

DSC 140B

Representation Learning

Lecture 12 | 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).

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

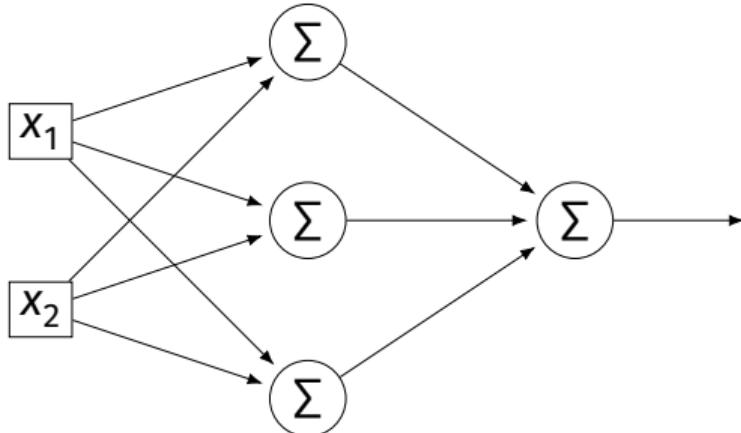
- ▶ These basis functions are fixed **before** learning.
- ▶ **Downside:** we have to choose $\vec{\phi}$ somehow.

Learning a Feature Map

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

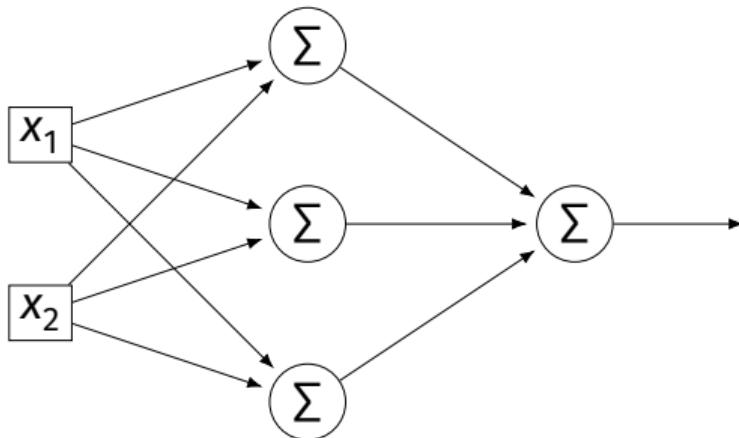
Each Layer is a Function

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



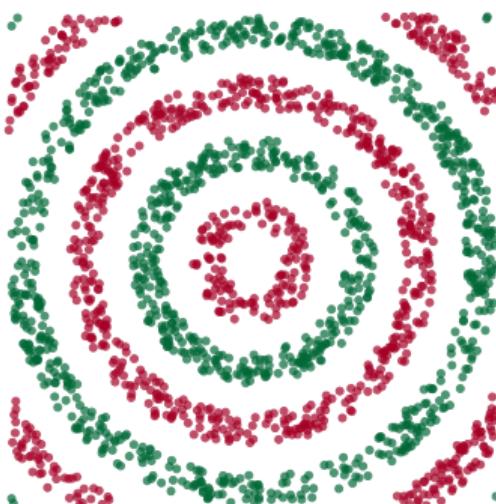
Each Layer is a Function

- ▶ 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”.



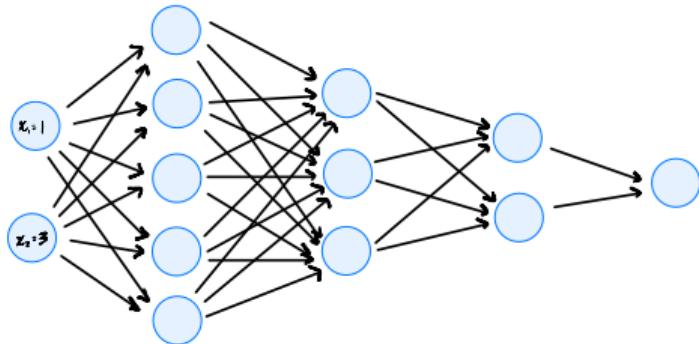
Demo

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

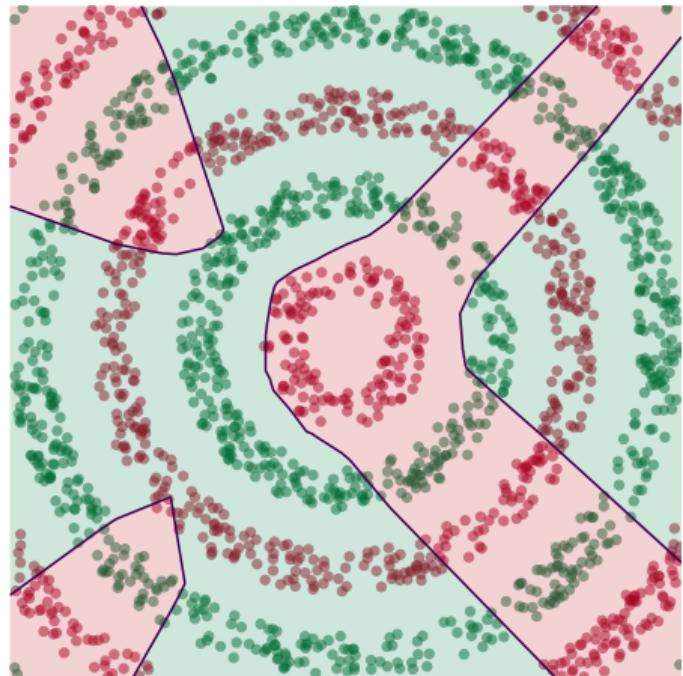


Demo

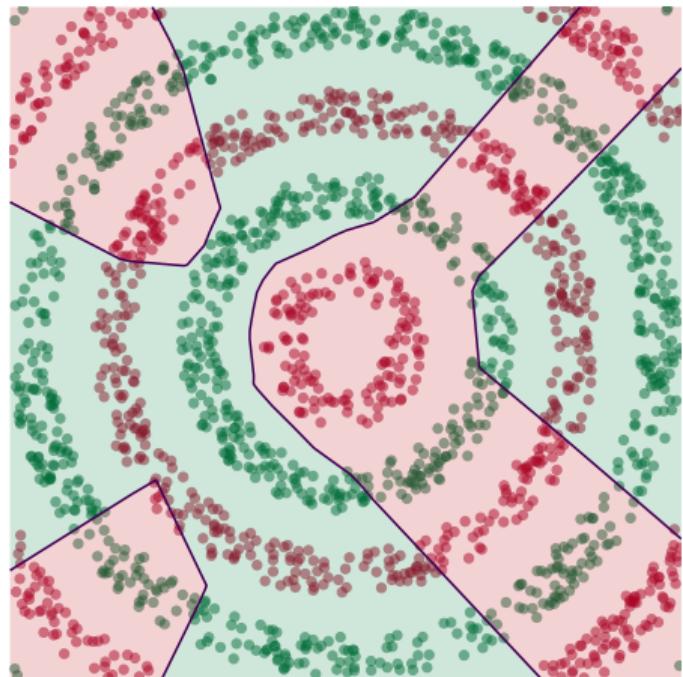
- ▶ We'll use three hidden layers, with last having two neurons.
- ▶ We can see this new representation!
- ▶ Plug in \vec{x} and see activations of last hidden layer.



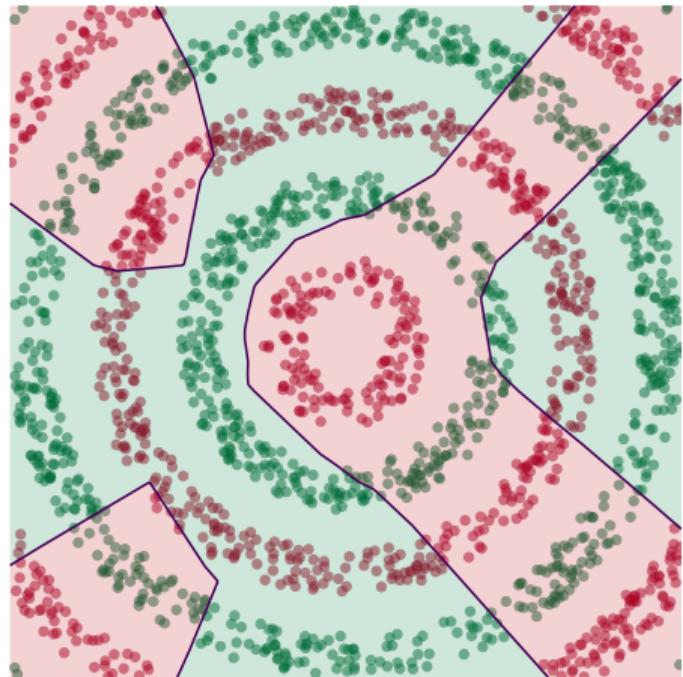
Learning a New Representation



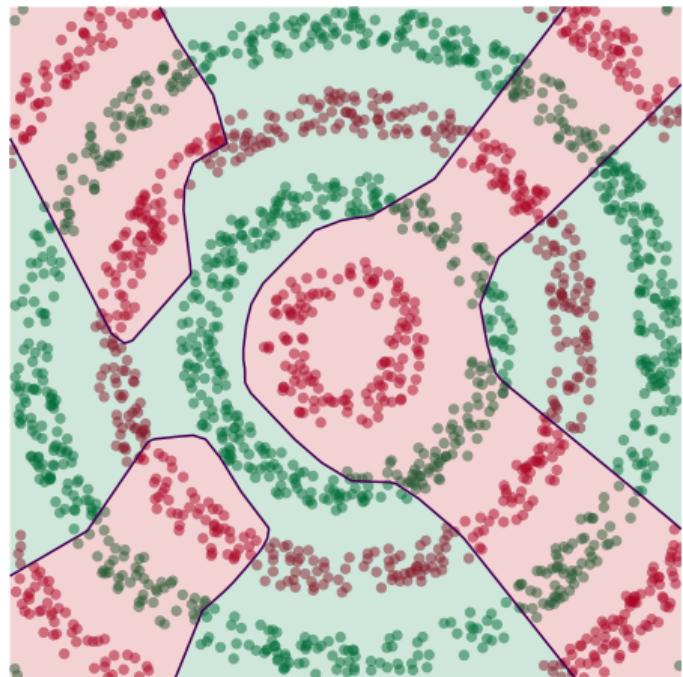
Learning a New Representation



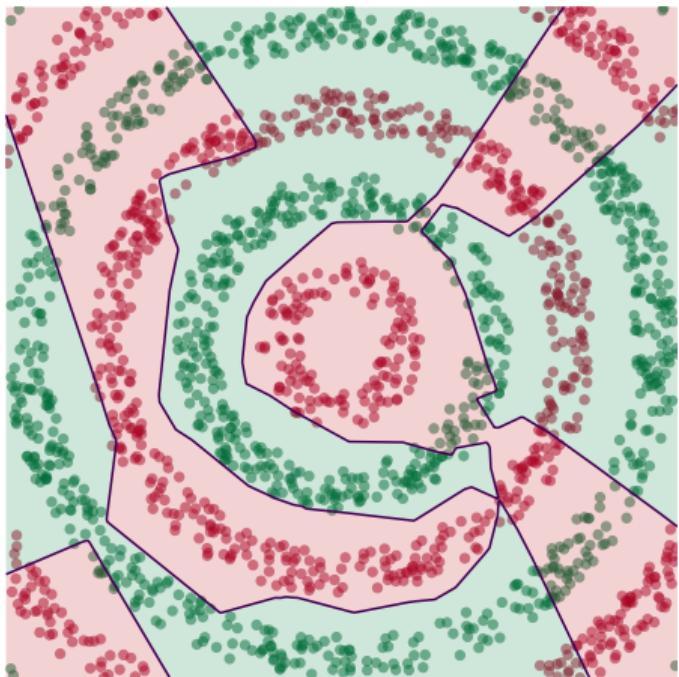
Learning a New Representation



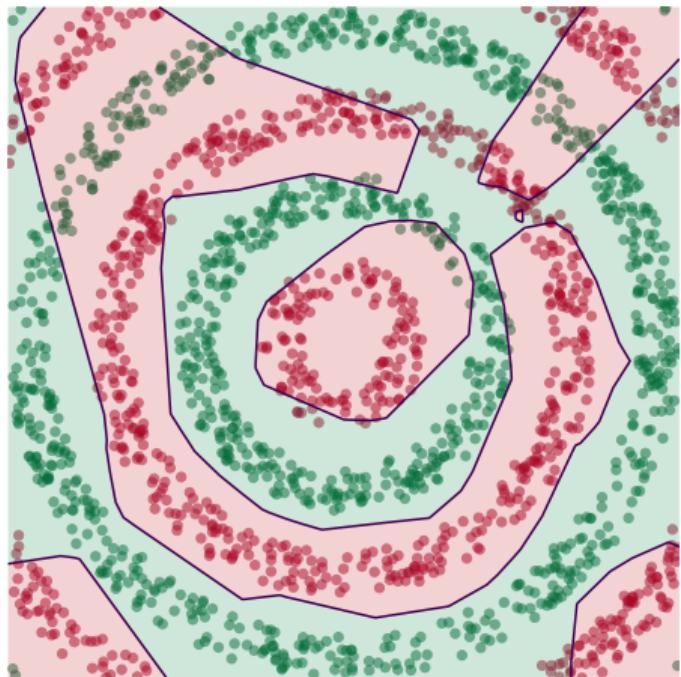
Learning a New Representation



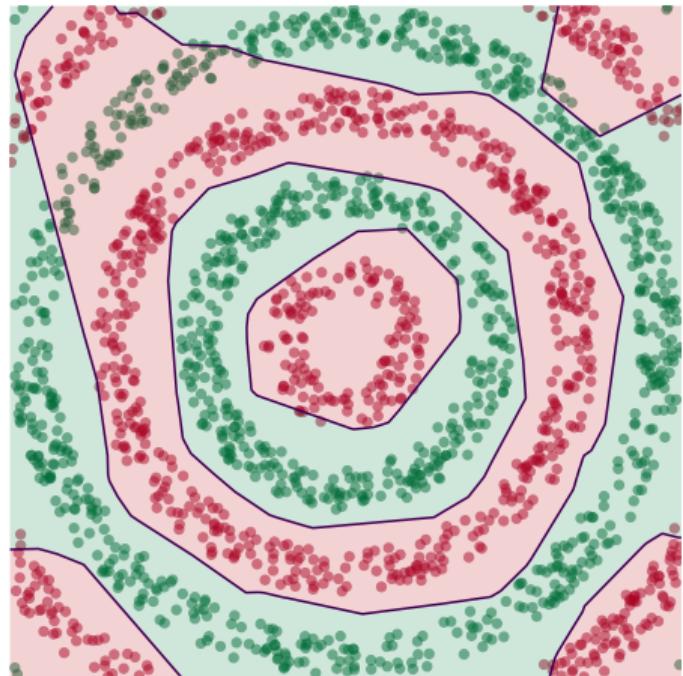
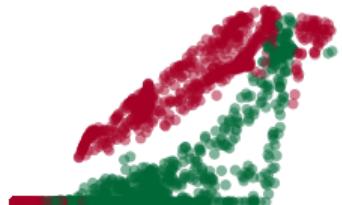
Learning a New Representation



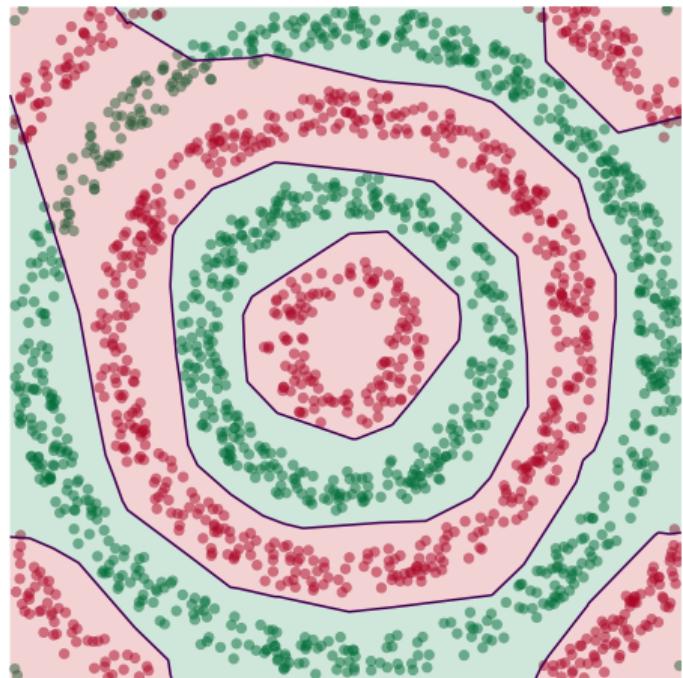
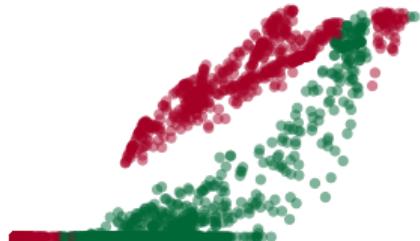
Learning a New Representation



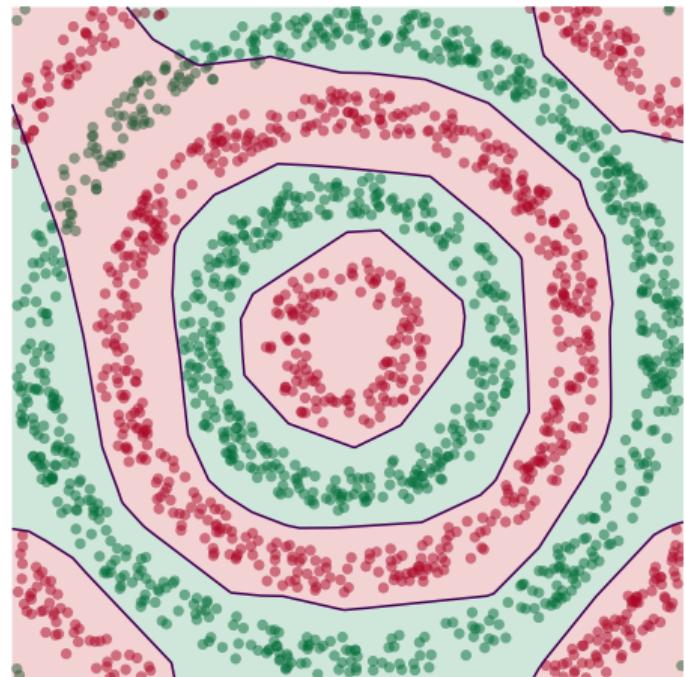
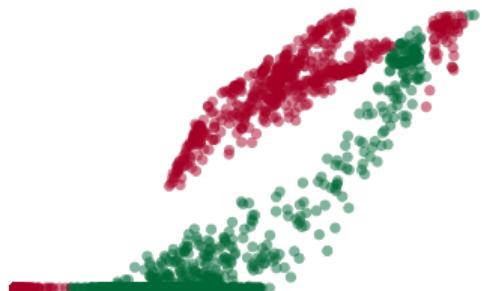
Learning a New Representation



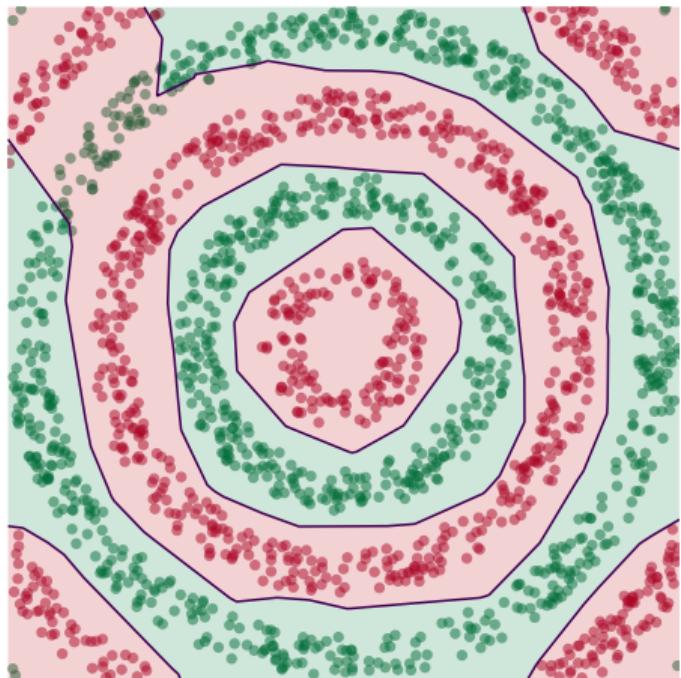
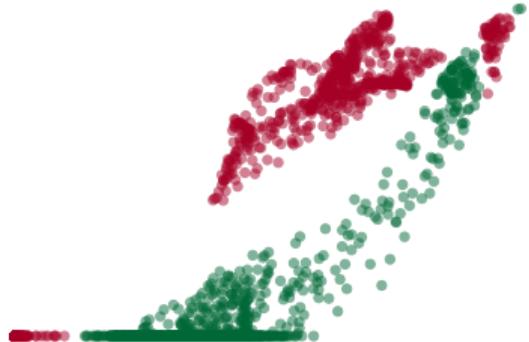
Learning a New Representation



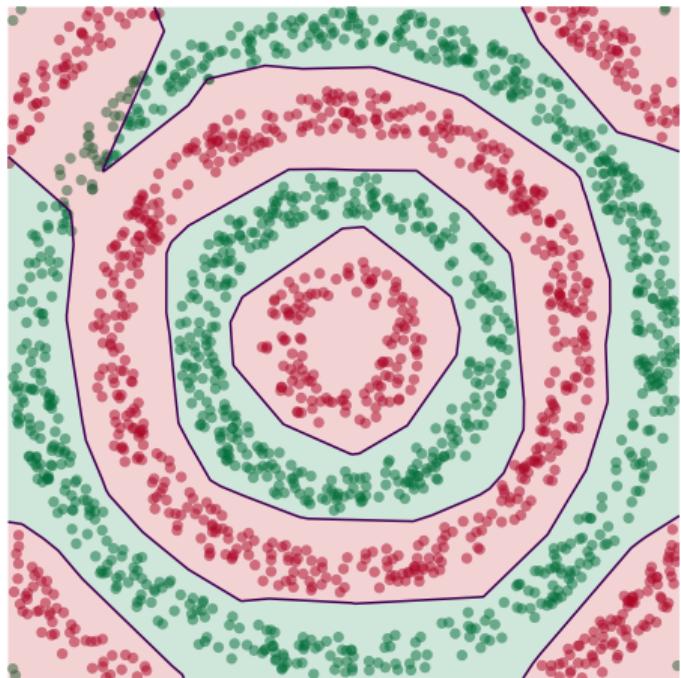
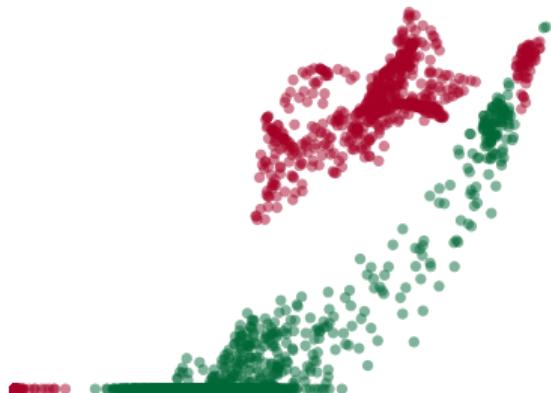
Learning a New Representation



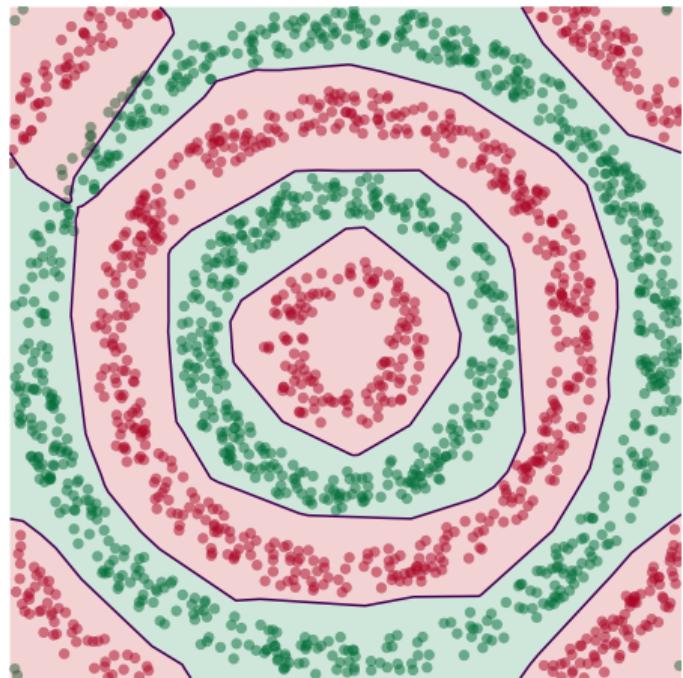
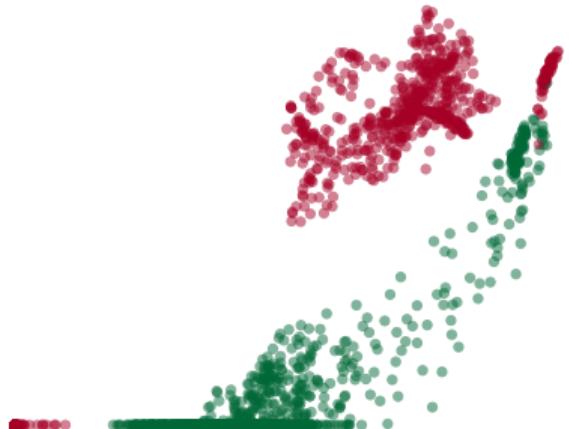
Learning a New Representation



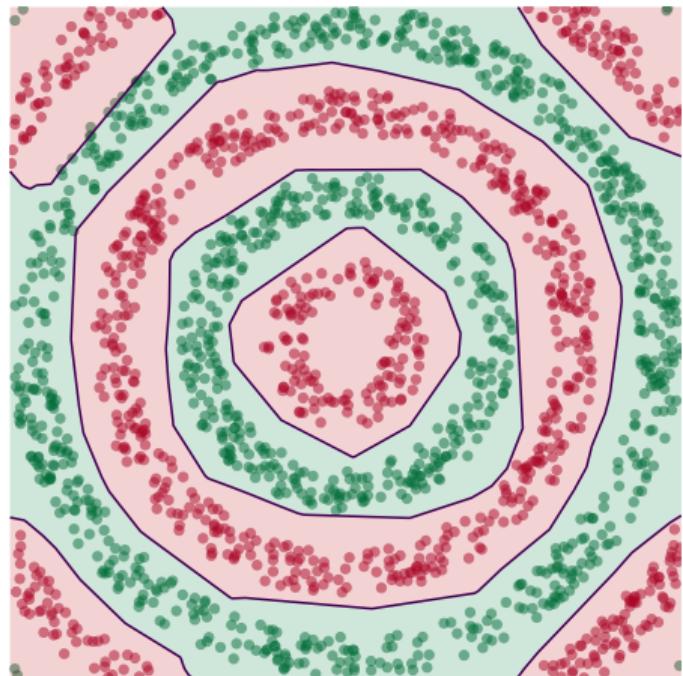
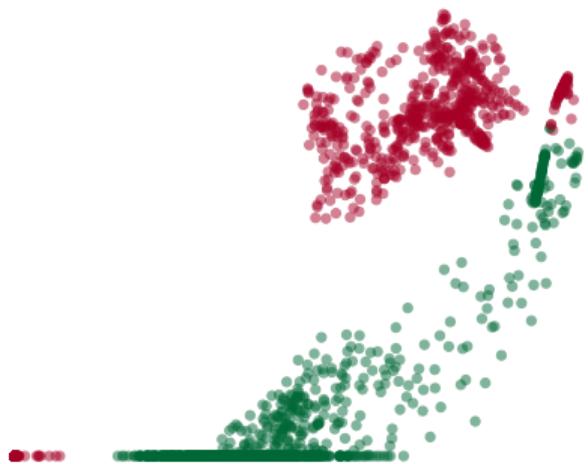
Learning a New Representation



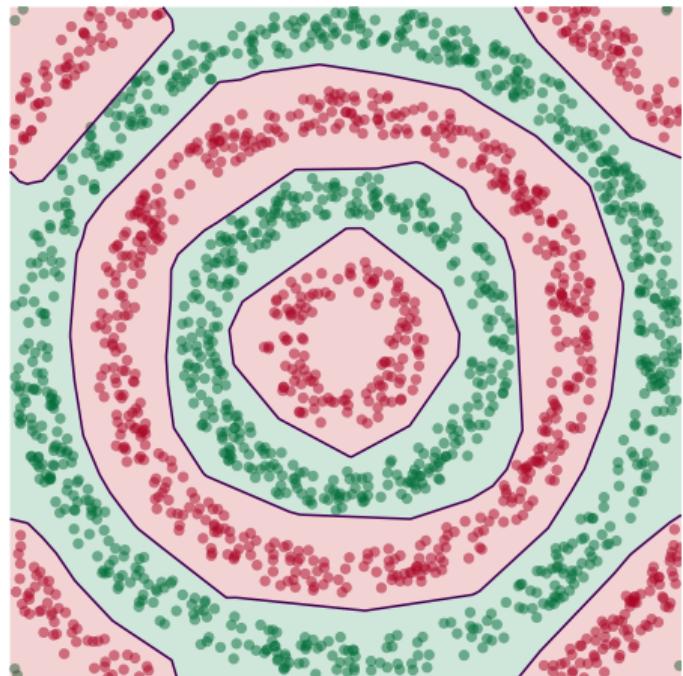
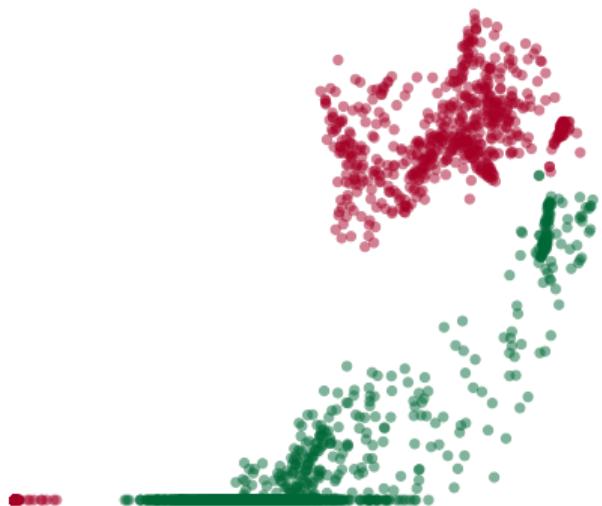
Learning a New Representation



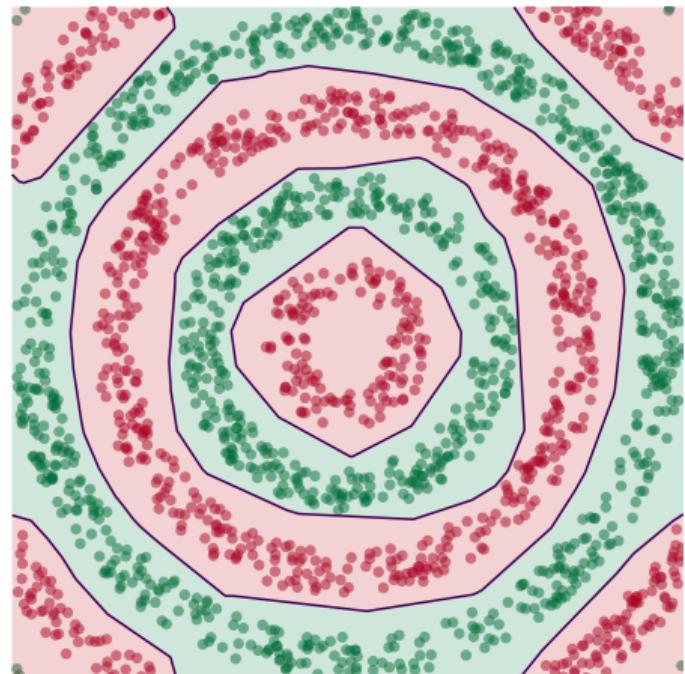
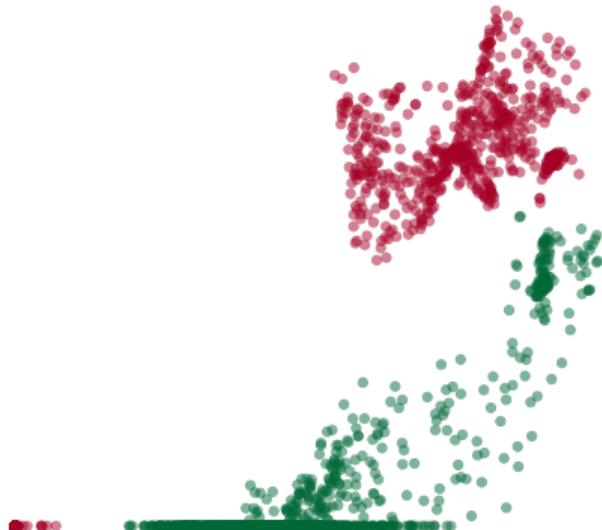
Learning a New Representation



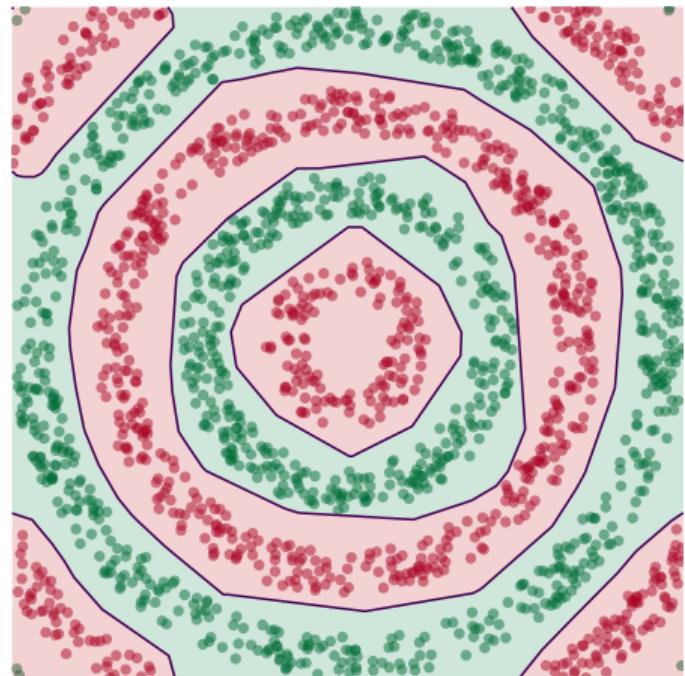
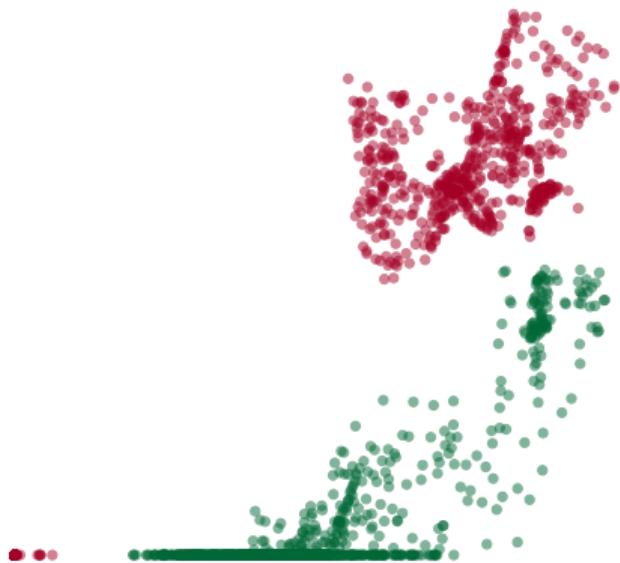
Learning a New Representation



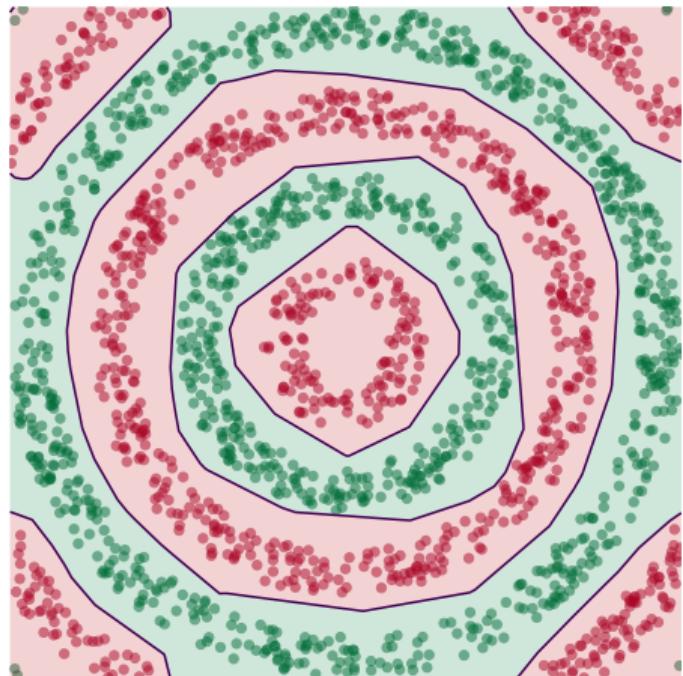
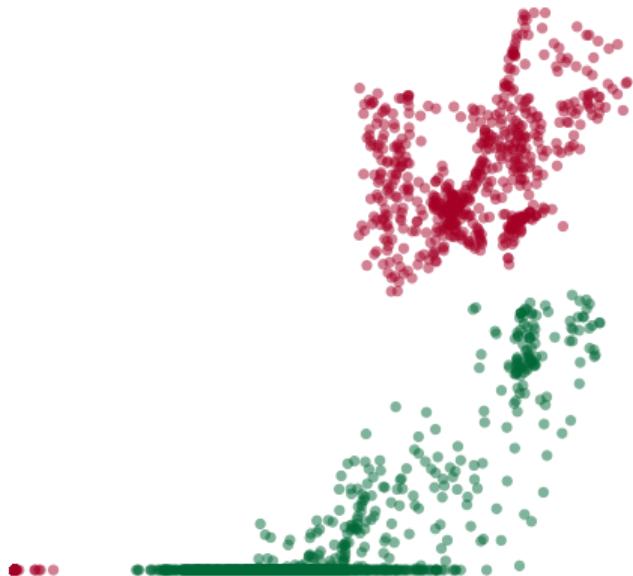
Learning a New Representation



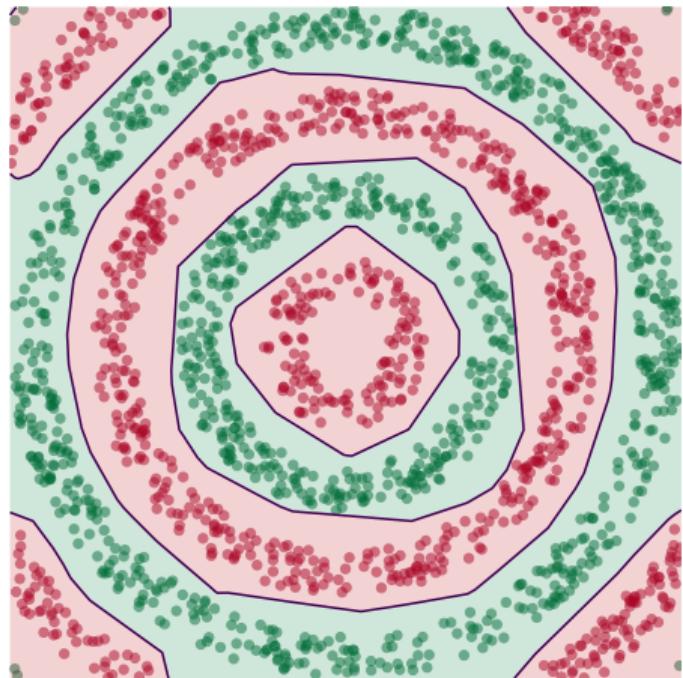
Learning a New Representation



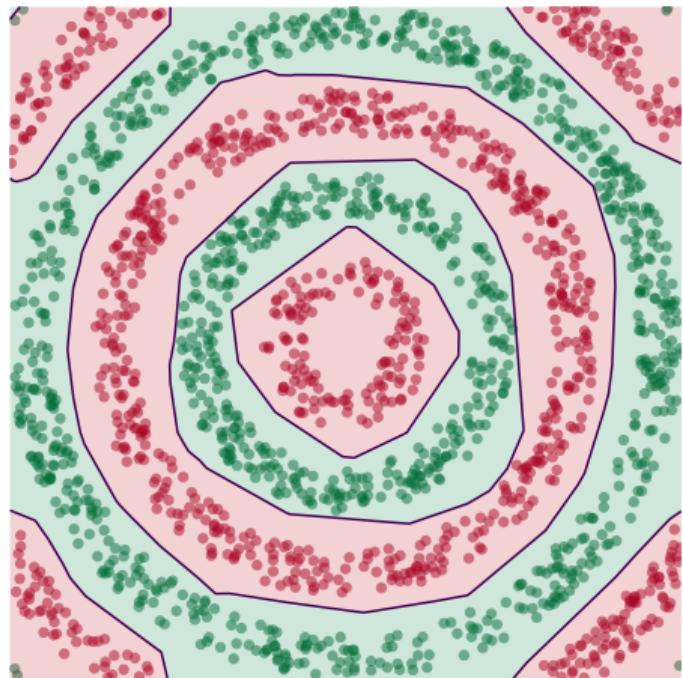
Learning a New Representation



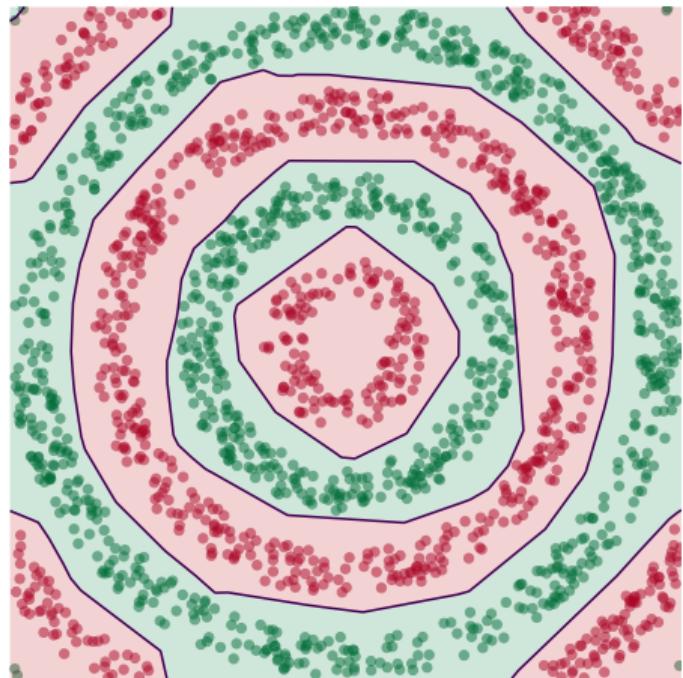
Learning a New Representation



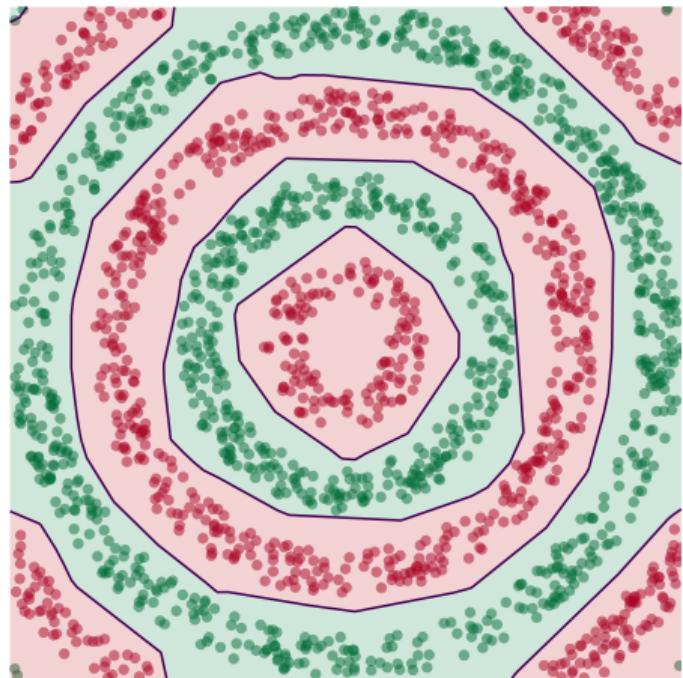
Learning a New Representation



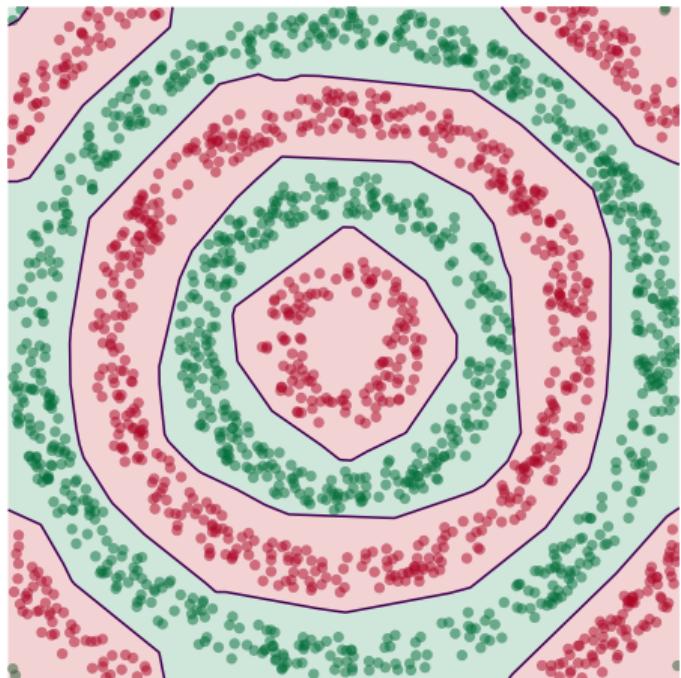
Learning a New Representation



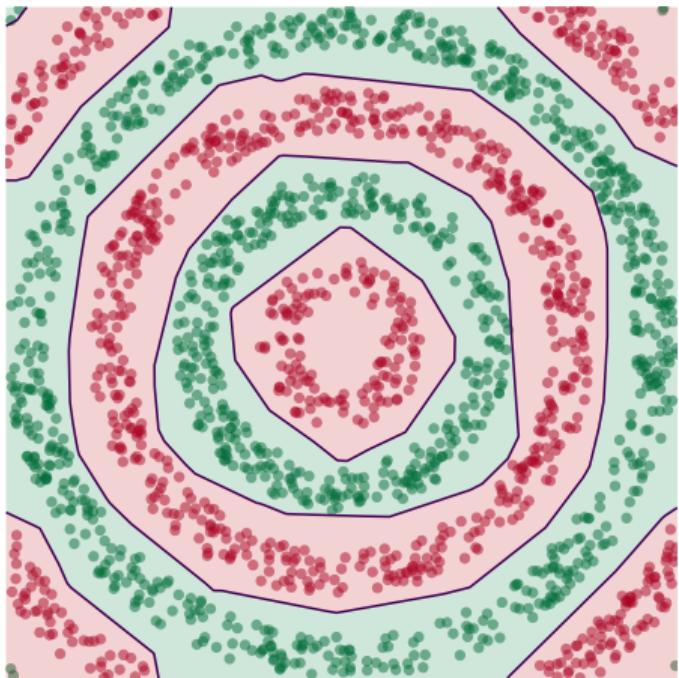
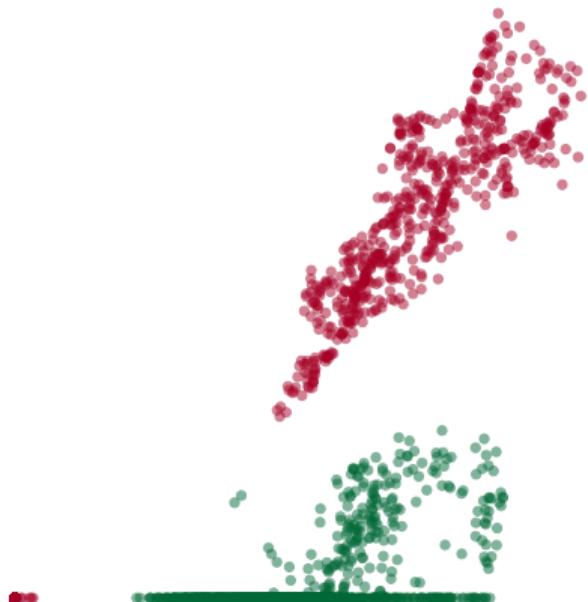
Learning a New Representation



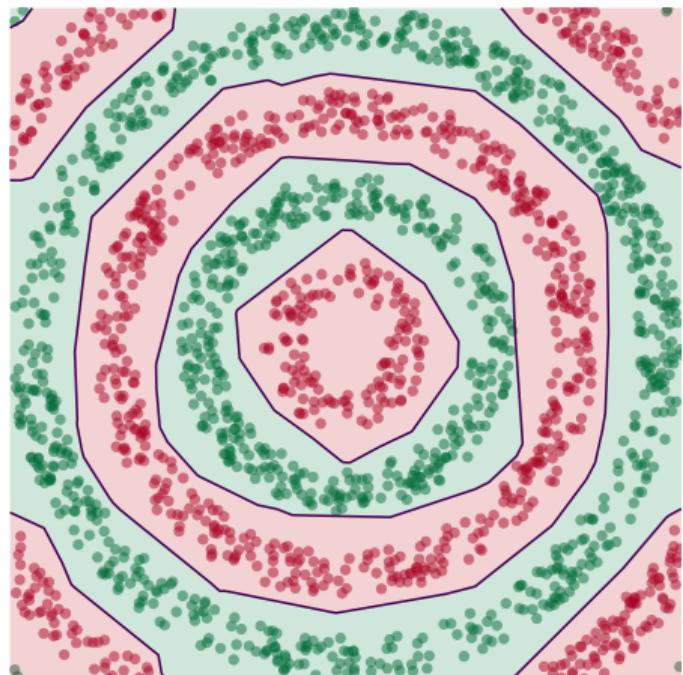
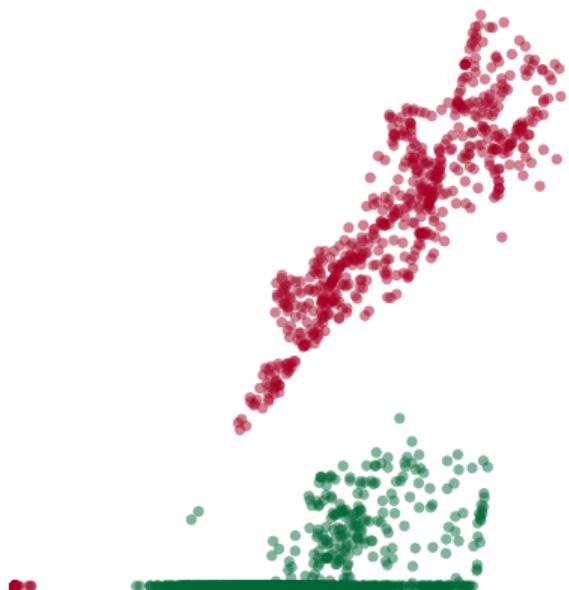
Learning a New Representation



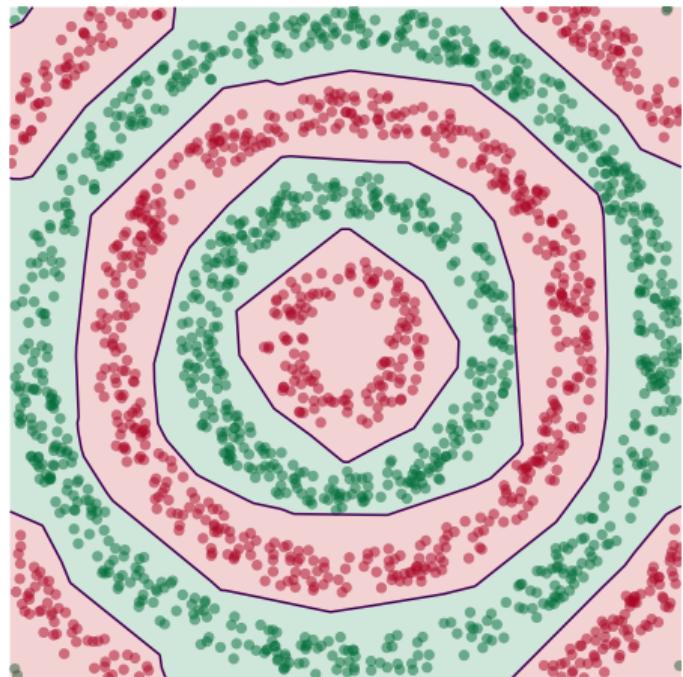
Learning a New Representation



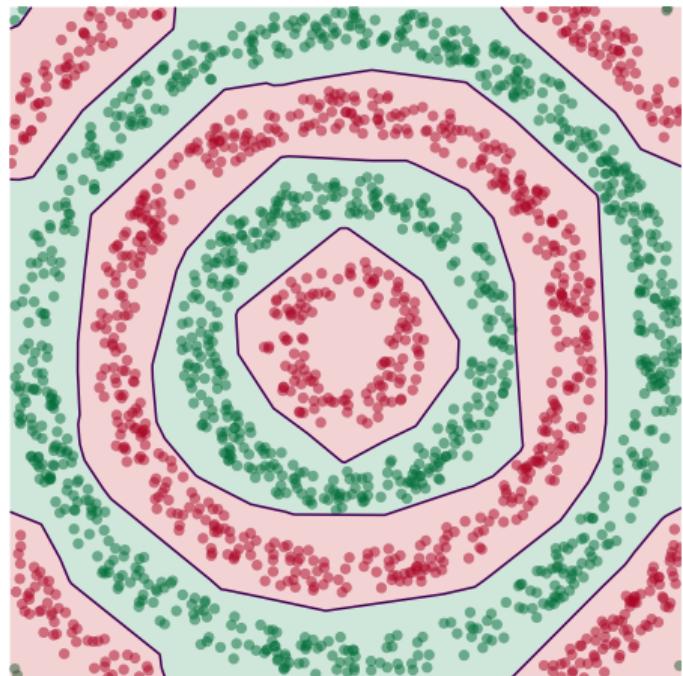
Learning a New Representation



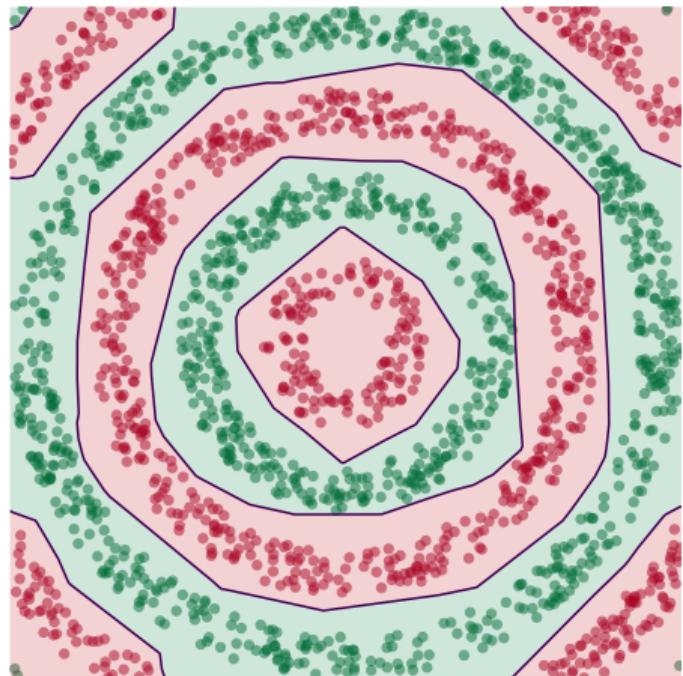
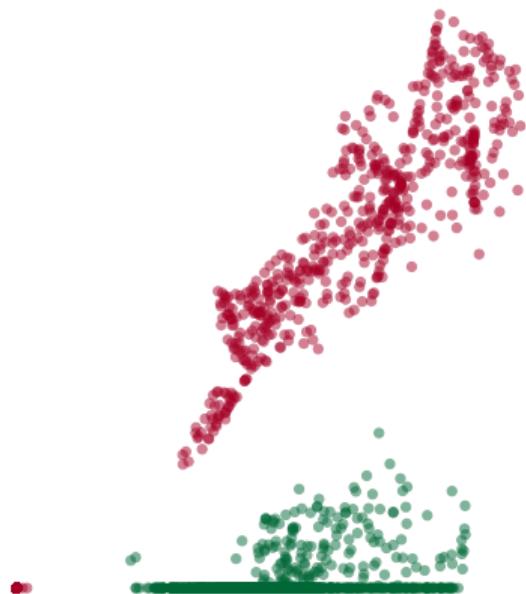
Learning a New Representation



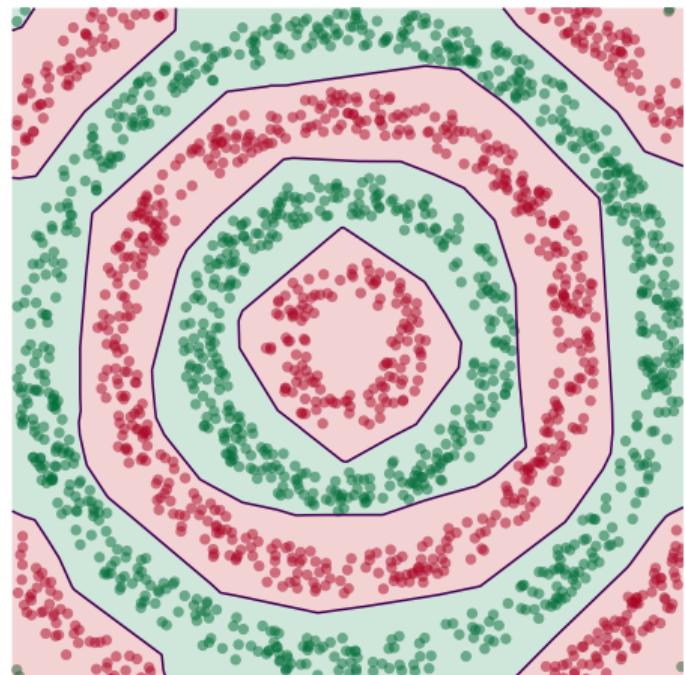
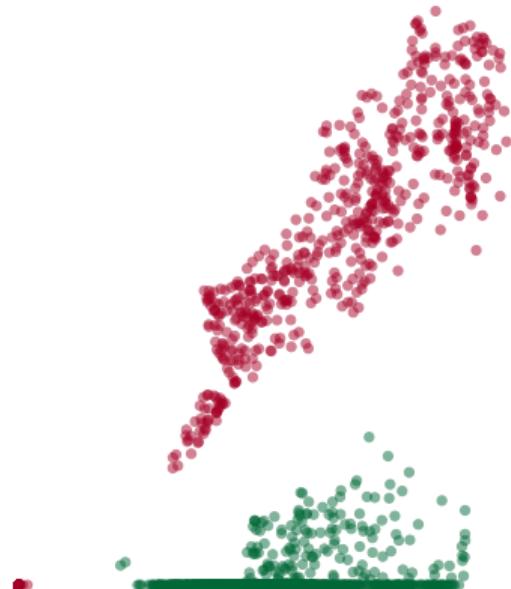
Learning a New Representation



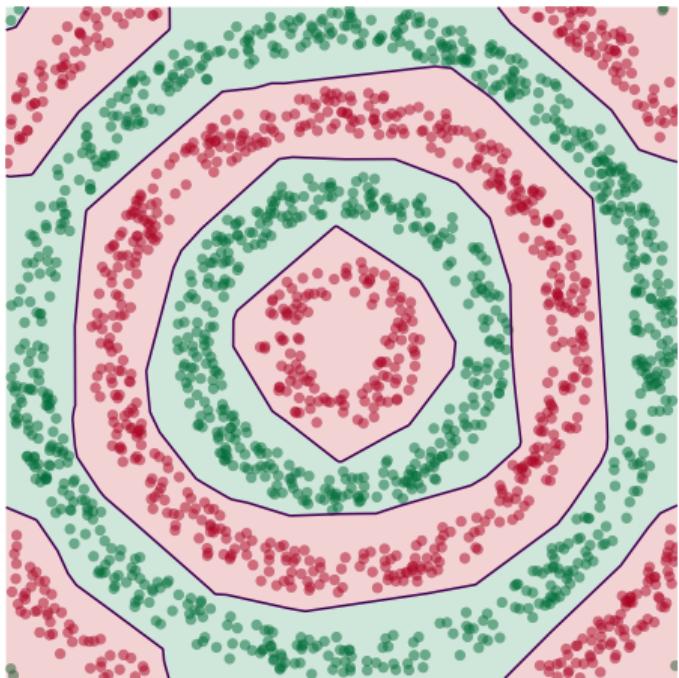
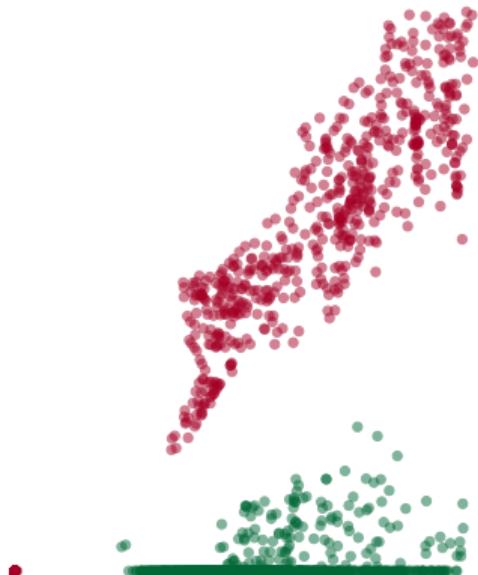
Learning a New Representation



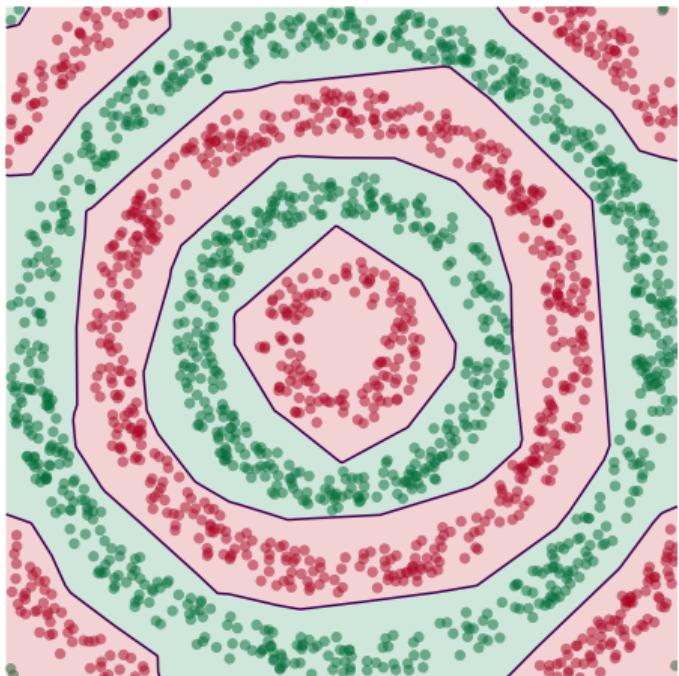
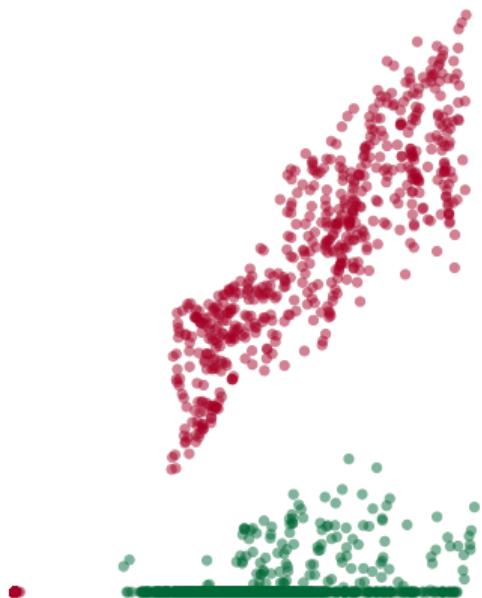
Learning a New Representation



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 1

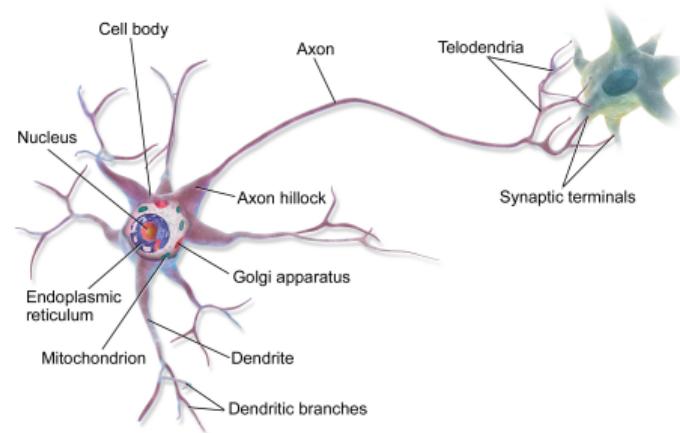
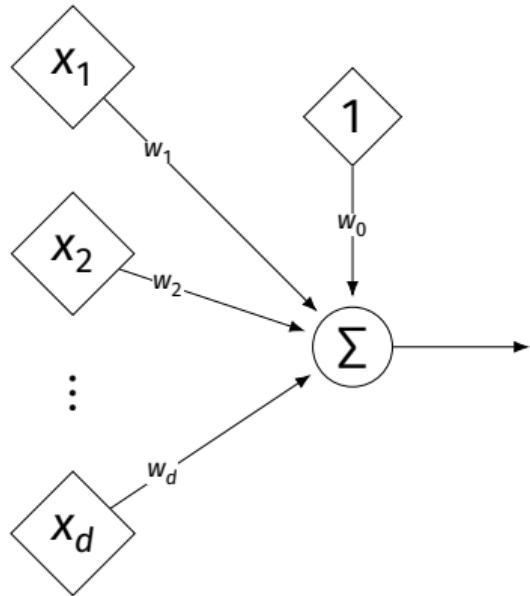
Neural Networks

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

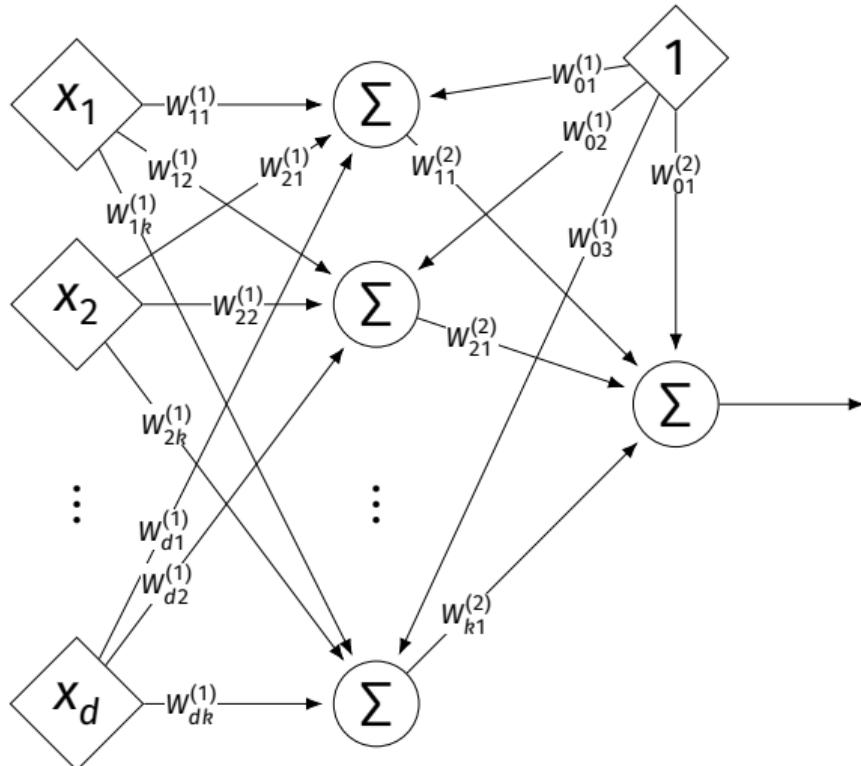
$$H(\vec{x}) = w_0 + w_1x_1 + \dots + w_dx_d$$



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

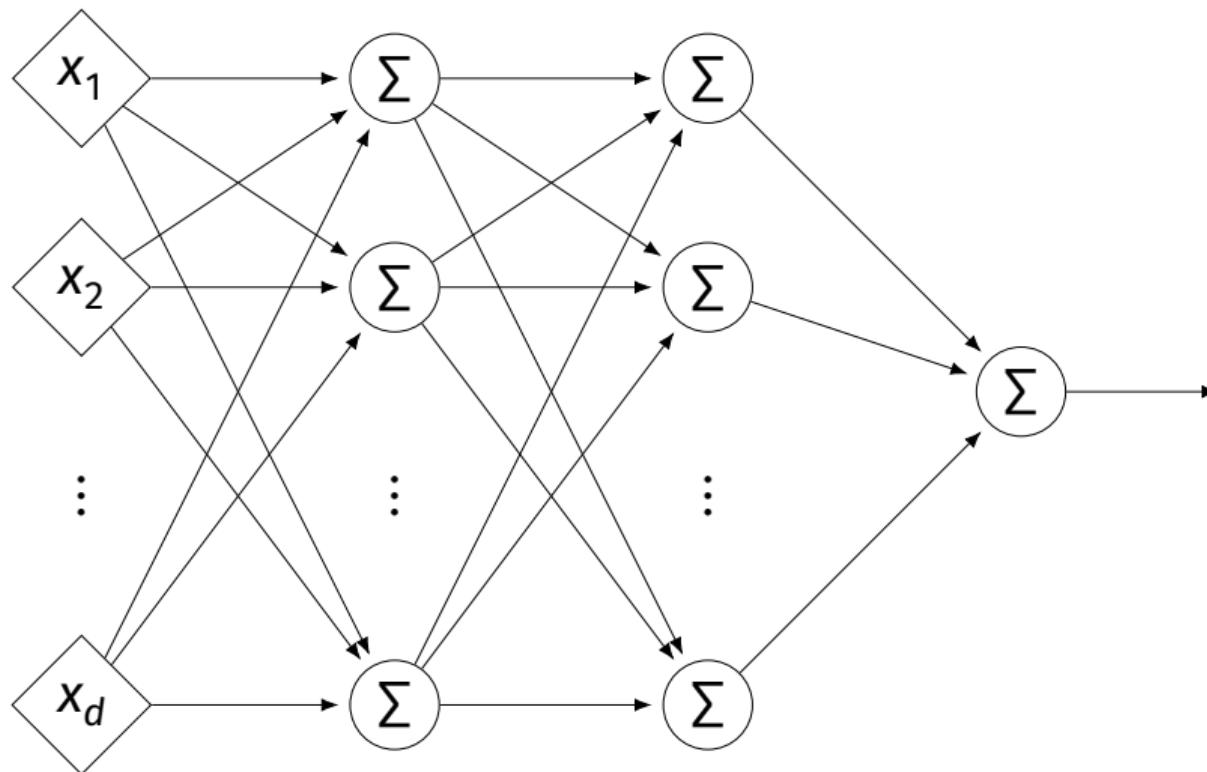
Architecture

- ▶ 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)

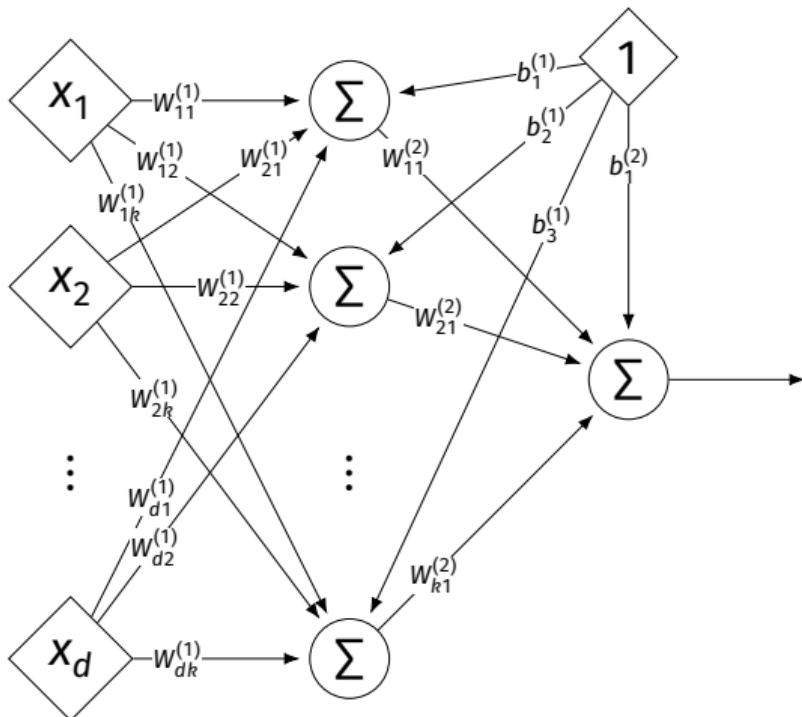


Network Weights

- ▶ 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!

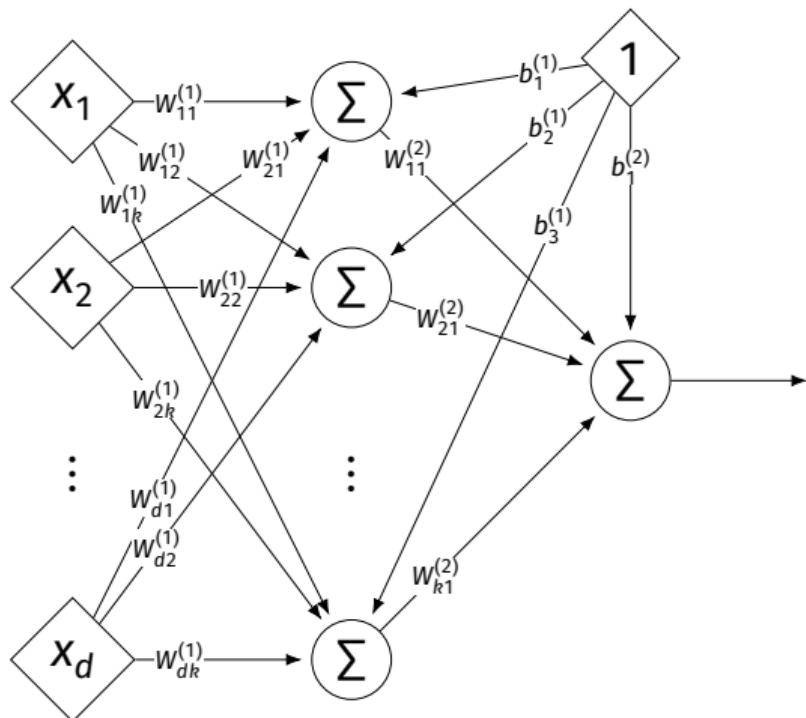
Notation

- ▶ 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.



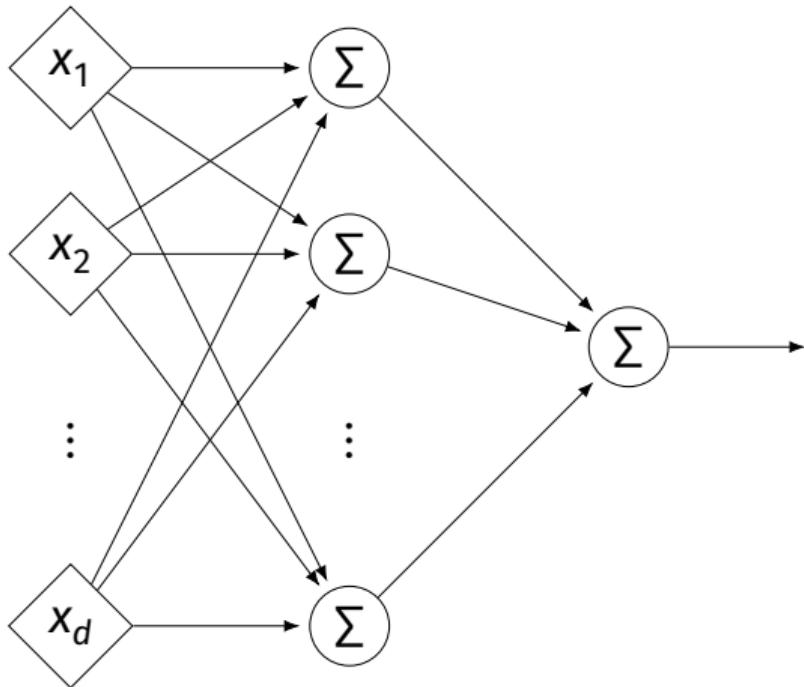
Notation

- ▶ 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: $\vec{b}^{(i)}$

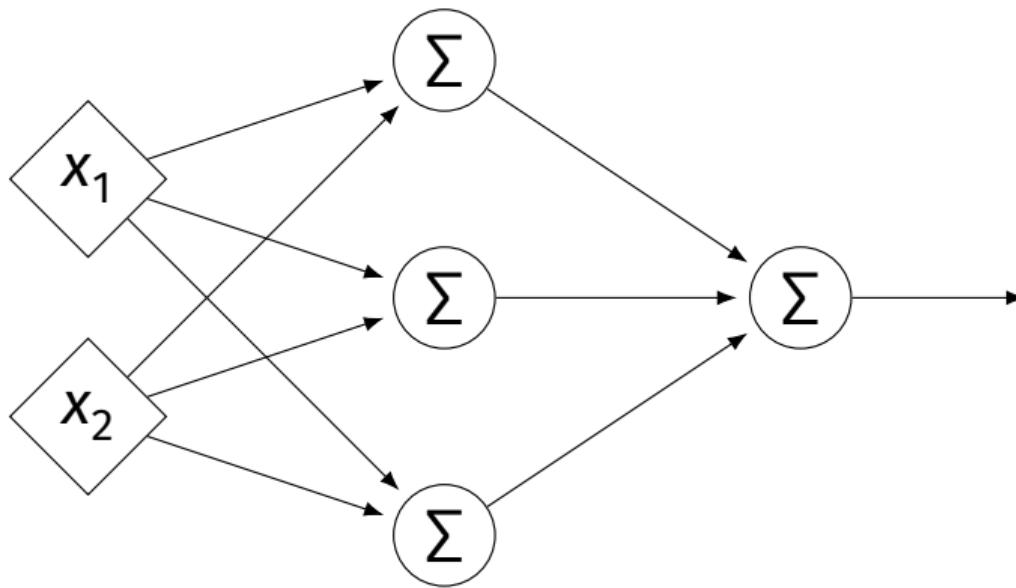


Notation

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



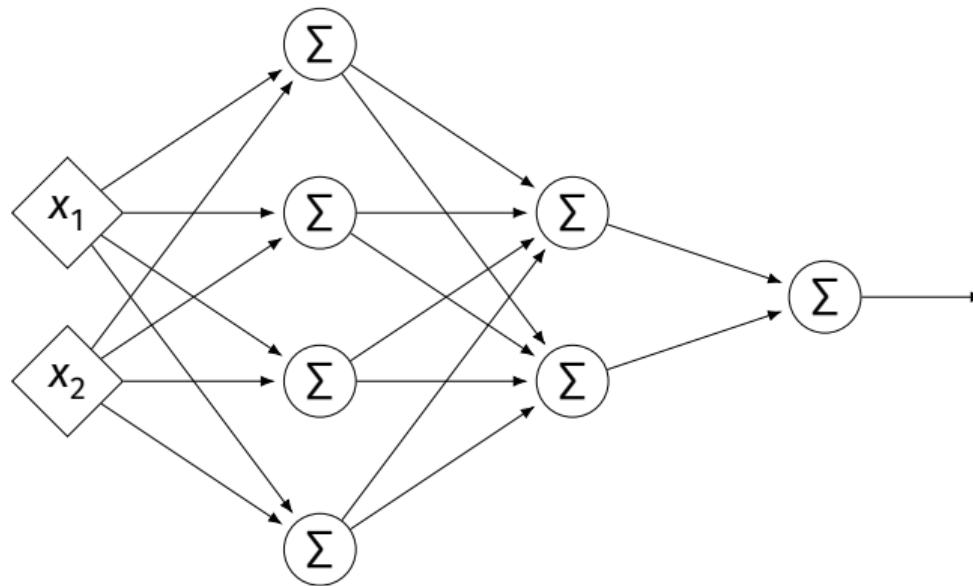
Example



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

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

Example



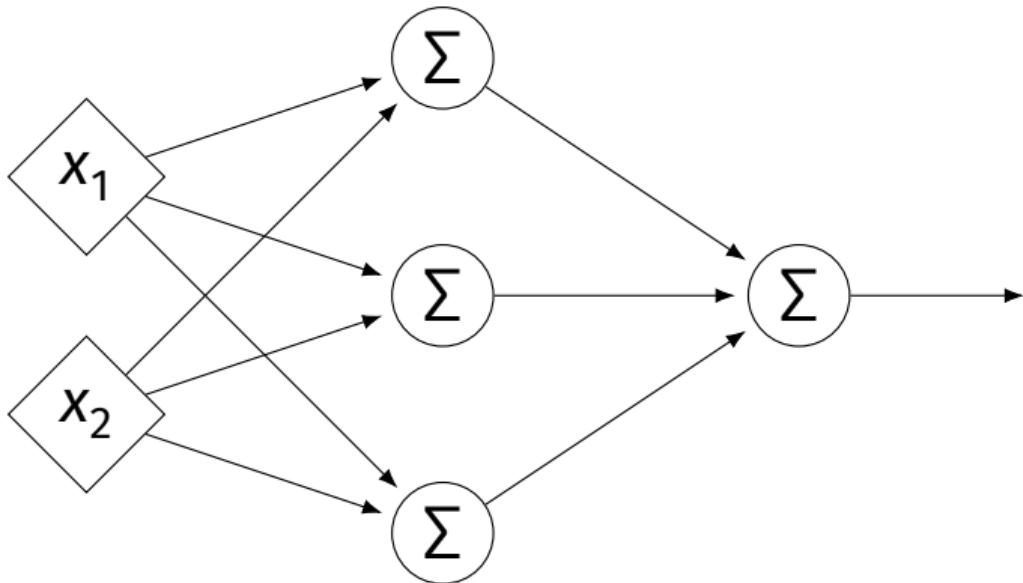
$$W^{(1)} = \begin{pmatrix} 2 & -1 & -3 & 0 \\ 4 & 5 & -7 & 2 \end{pmatrix} \quad W^{(2)} = \begin{pmatrix} 1 & 2 \\ -4 & 3 \\ -6 & -2 \\ 3 & 4 \end{pmatrix} \quad W^{(3)} = \begin{pmatrix} -1 & 5 \end{pmatrix}$$

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

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



► $\vec{x} = (3, -1)^T$

► $z_1^{(1)} =$

► $z_2^{(1)} =$

► $z_3^{(1)} =$

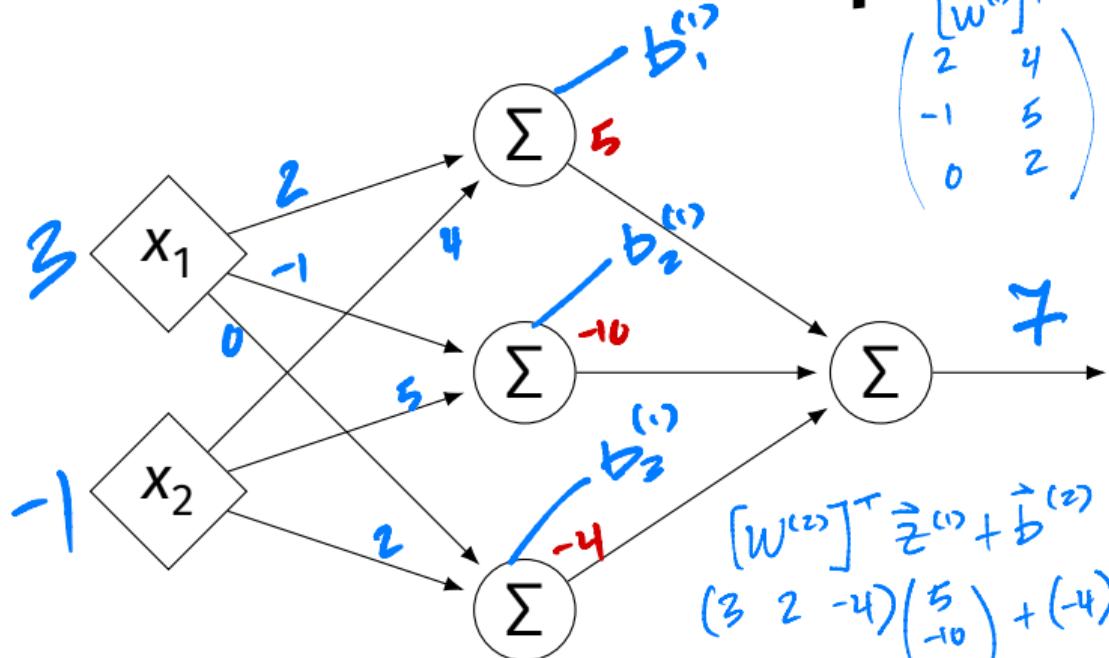
► $z_1^{(2)} =$

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

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



$$W^{(1)} \vec{x} + \vec{b}^{(1)}$$

$$\begin{pmatrix} [W^{(1)}]^T & \vec{x} \\ 2 & 4 \\ -1 & 5 \\ 0 & 2 \end{pmatrix} \begin{pmatrix} 3 \\ -1 \end{pmatrix} + \begin{pmatrix} \vec{b}^{(1)} \\ 5 \\ -10 \\ -4 \end{pmatrix} = \begin{pmatrix} 2 \\ -8 \\ -2 \\ -2 \end{pmatrix} + \begin{pmatrix} 3 \\ -2 \\ -2 \\ -2 \end{pmatrix}$$

$$\vec{x} = (3, -1)^T = \begin{pmatrix} 5 \\ -10 \\ -4 \end{pmatrix}$$

$$z_1^{(1)} = 5$$

$$z_2^{(1)} = -10$$

$$z_3^{(1)} = -4$$

$$[W^{(2)}]^T \vec{z}^{(1)} + \vec{b}^{(2)}$$

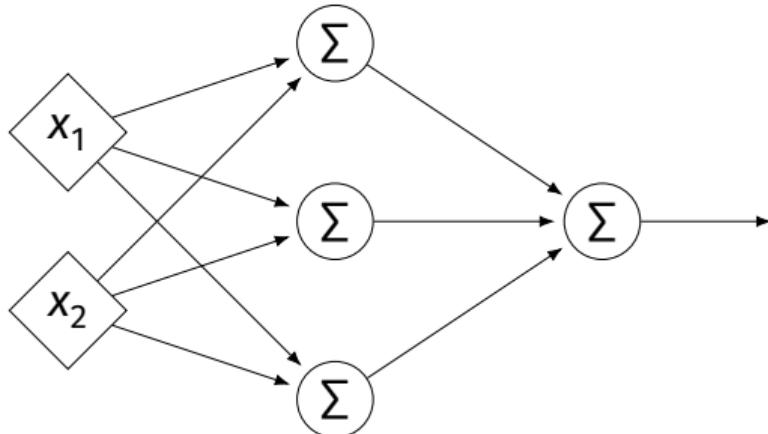
$$(3 \ 2 \ -4) \begin{pmatrix} 5 \\ -10 \\ -4 \end{pmatrix} + (-4) = (15 \ -20 \ +16 \ -4) = 7$$

$$z_1^{(2)} = 7$$

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

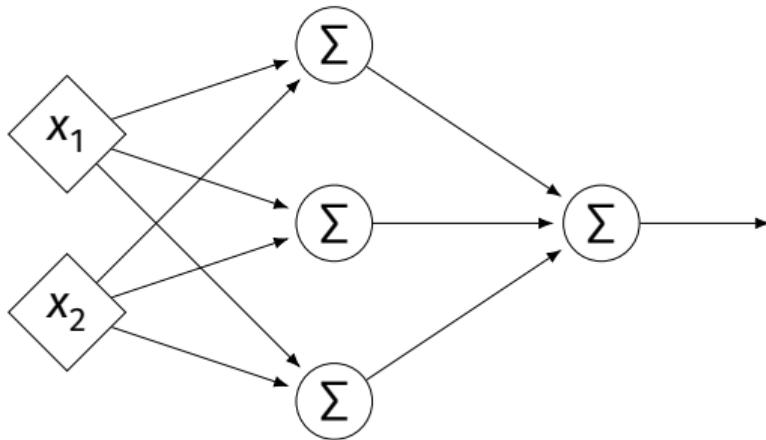
Each Layer is a Function

- ▶ 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:

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

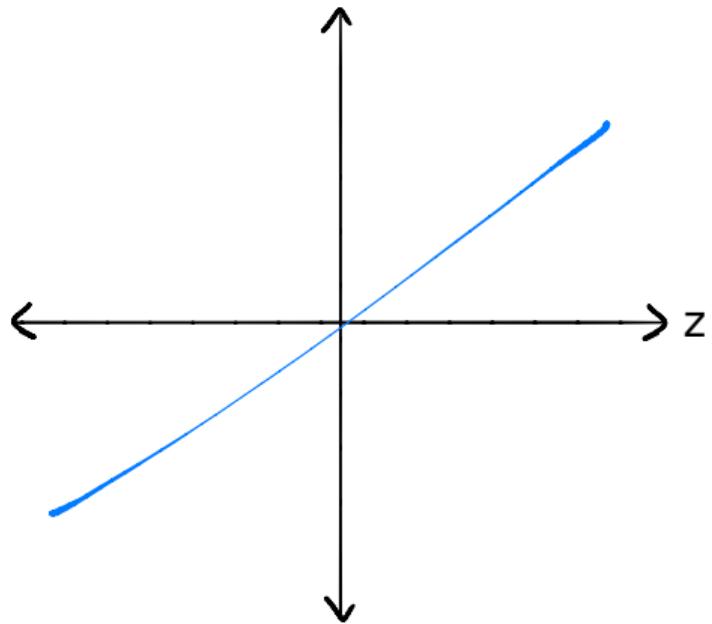
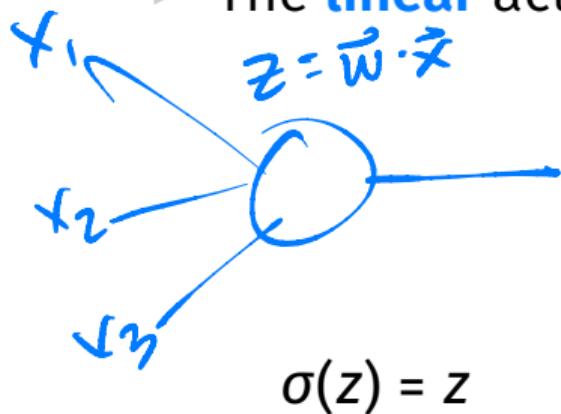
- ▶ Can be arbitrarily large or small.
- ▶ But real neurons are **activated** non-linearly.
 - ▶ E.g., saturation.

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).

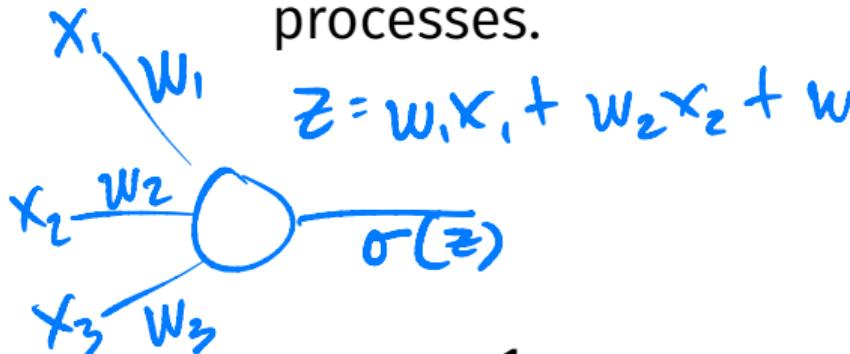
Linear Activation

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



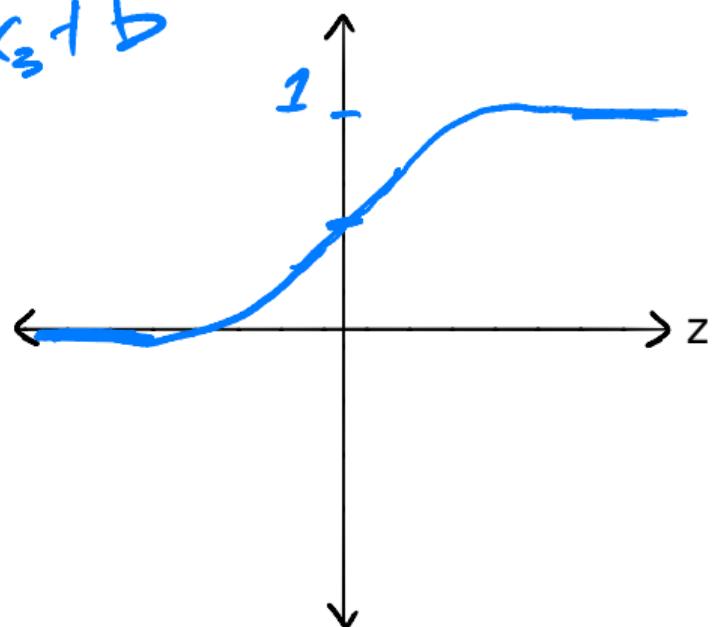
Sigmoid Activation

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



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

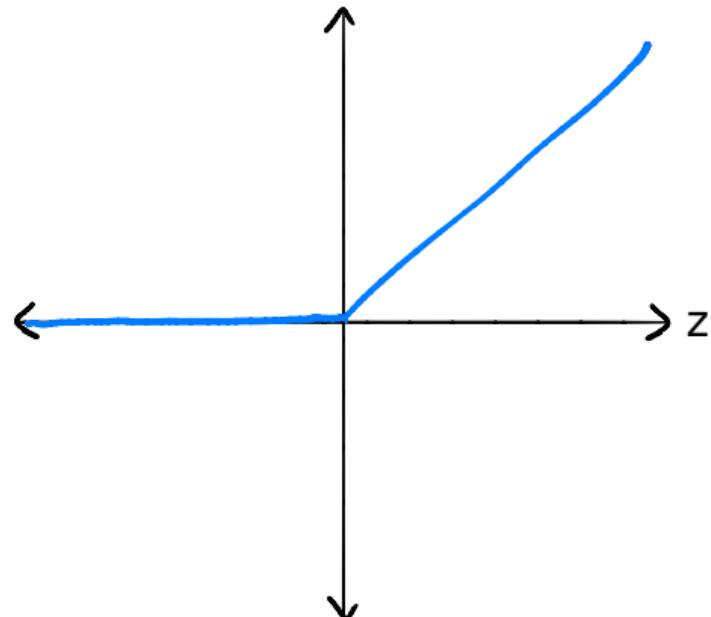
$$\sigma(-\infty) = 0 \quad \sigma(0) = \frac{1}{2}$$
$$\sigma(\infty) = 1$$



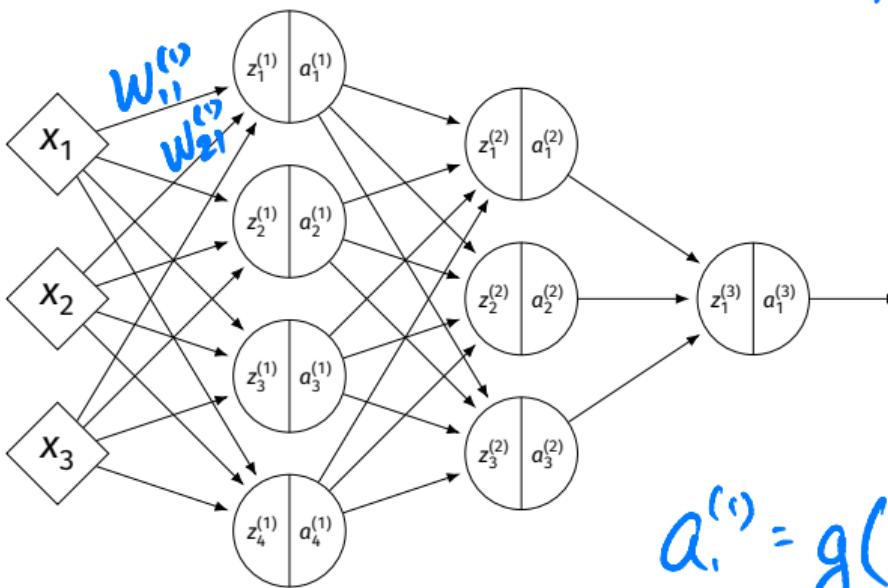
ReLU Activation

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

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



Notation



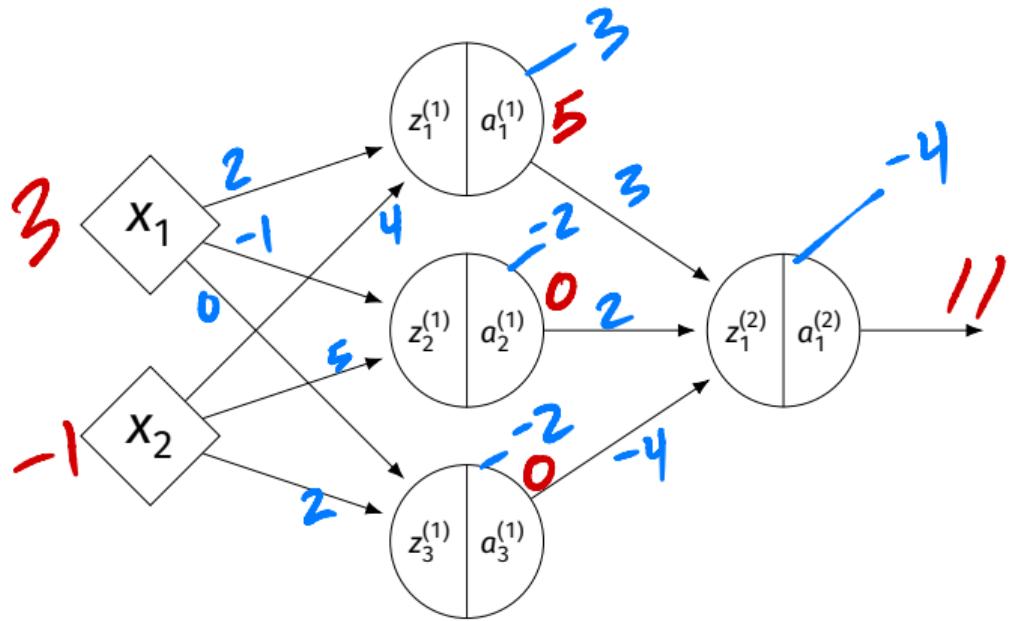
$$z_1^{(1)} = W_{11}^{(1)}x_1 + W_{21}^{(1)}x_2 + W_{31}^{(1)}x_3 + b_1^{(1)}$$

$$a_1^{(1)} = g(z_1^{(1)})$$

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

Example

$$(2)(3) + (4)(-1) + 3 \\ = 6 - 4 + 3 = 5$$



- ▶ $g = \text{ReLU}$
- ▶ Linear output
- ▶ $\vec{x} = (3, -1)^T$
- ▶ $z_1^{(1)} = 5$
- ▶ $a_1^{(1)} = g(5) = 5$
- ▶ $z_2^{(1)} = -10$
- ▶ $a_2^{(1)} = g(-10) = 0$
- ▶ $z_3^{(1)} = -4$
- ▶ $a_3^{(1)} = g(-4) = 0$
- ▶ $z_1^{(2)} = 11$
- ▶ $a_1^{(2)} = 11$

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

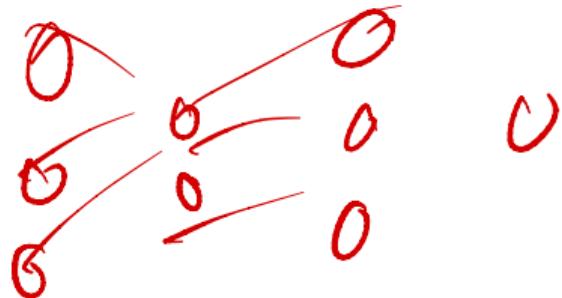
$$\vec{z}^{(q)} = g(W^{(l)} \vec{z}^{(l-1)} + \vec{b}^{(l)})$$

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).

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

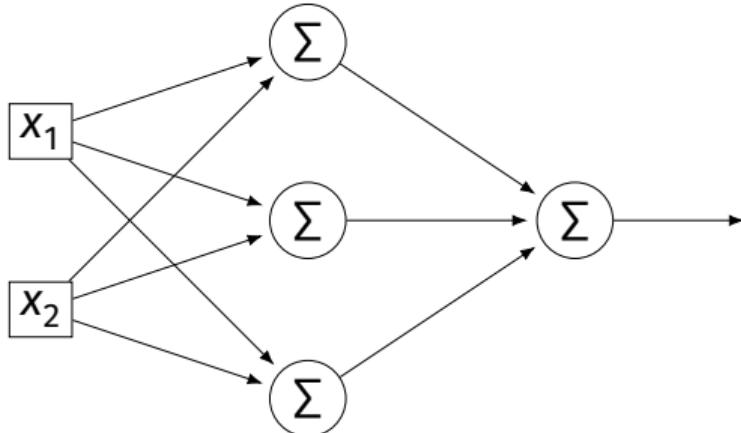
- ▶ These basis functions are fixed **before** learning.
- ▶ **Downside:** we have to choose $\vec{\phi}$ somehow.

Learning a Feature Map

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

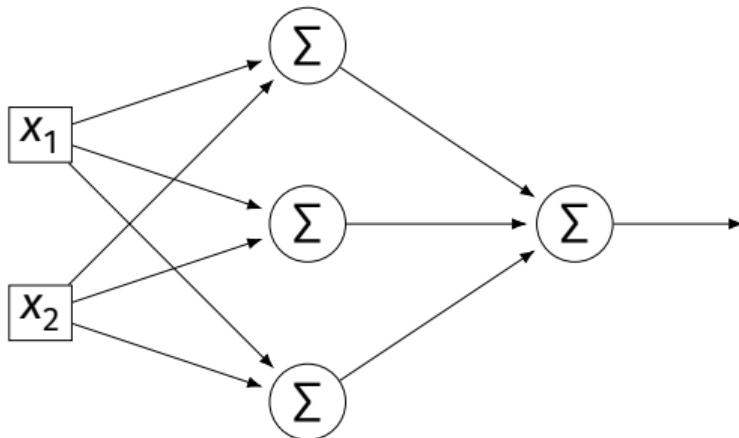
Each Layer is a Function

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



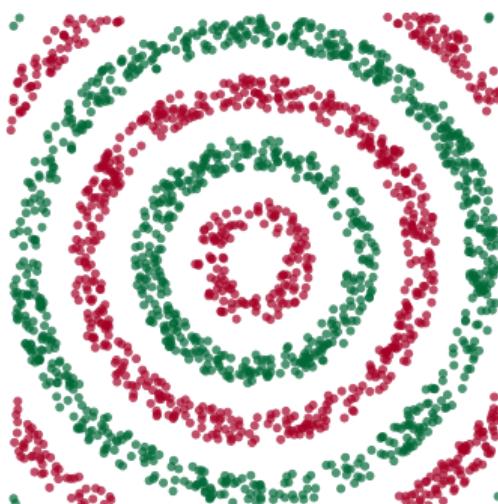
Each Layer is a Function

- ▶ 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

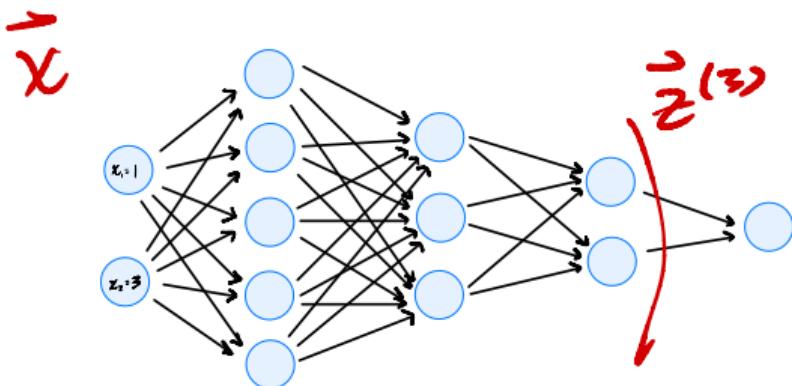
- ▶ Train a deep network to classify the data below.
- ▶ Hidden layers will learn a new feature map that makes the data linearly separable.



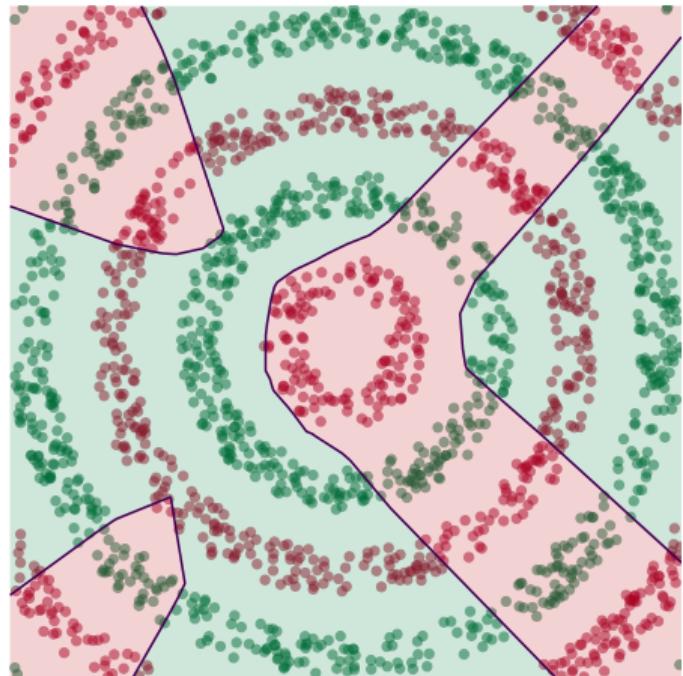
Demo

$$\vec{x} \mapsto \vec{z}^{(3)}$$

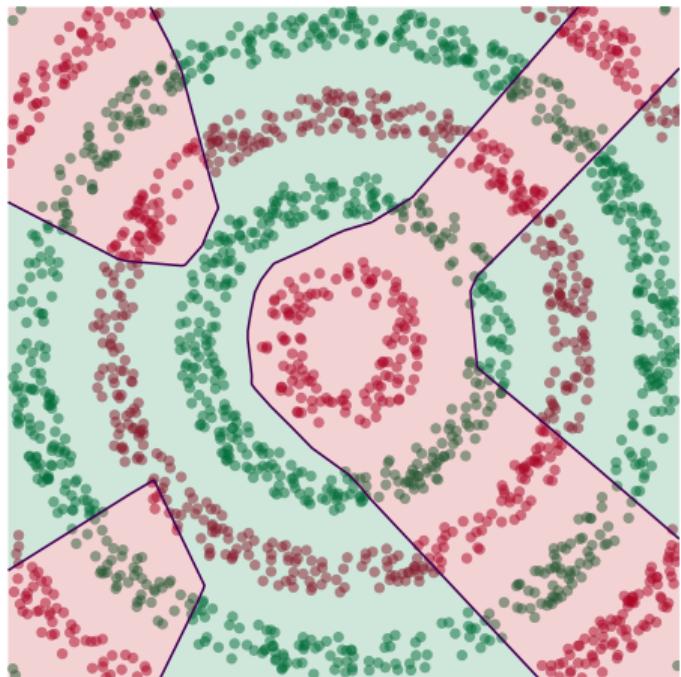
- ▶ We'll use three hidden layers, with last having two neurons.
- ▶ We can see this new representation!
- ▶ Plug in \vec{x} and see activations of last hidden layer.



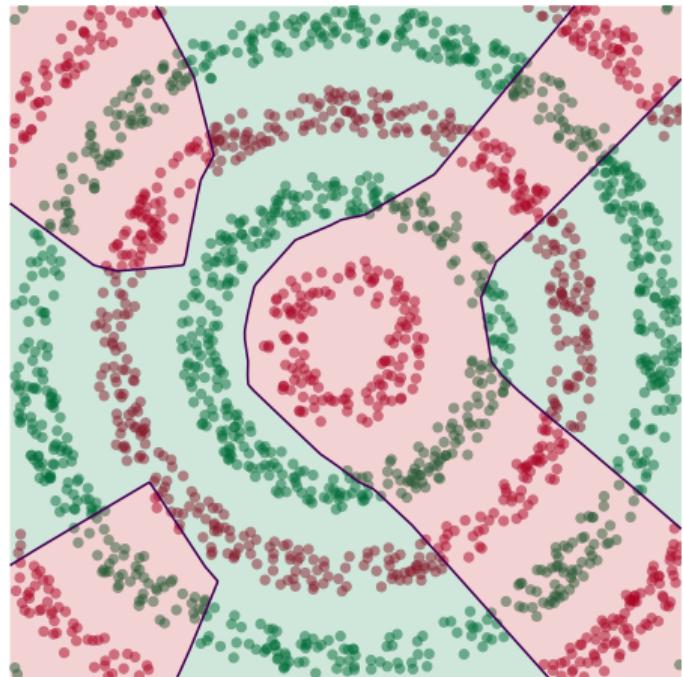
Learning a New Representation



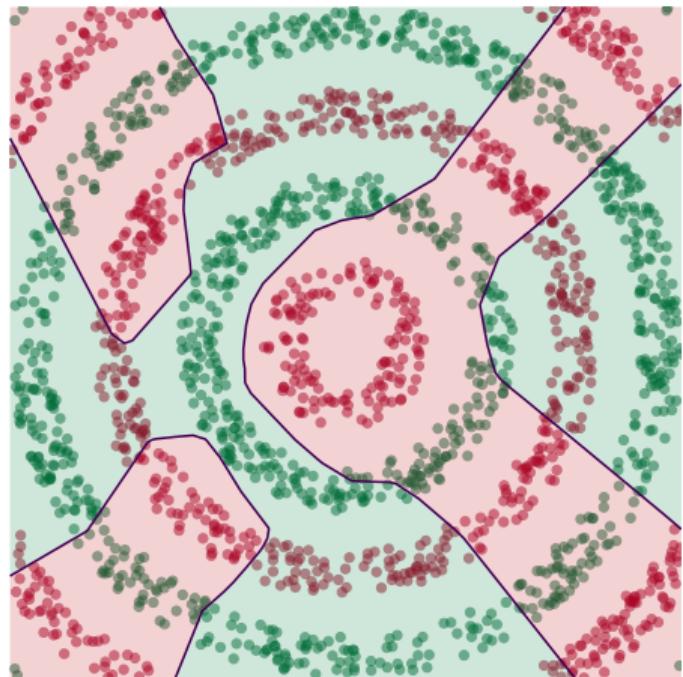
Learning a New Representation



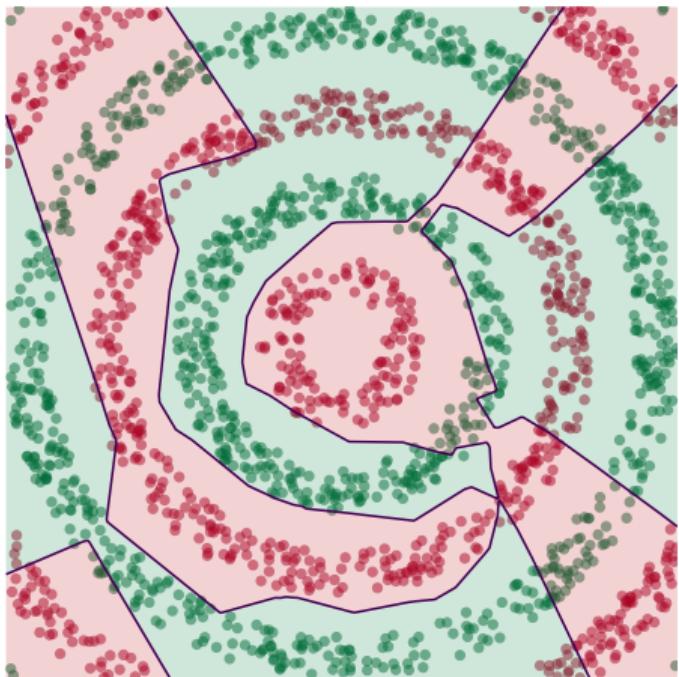
Learning a New Representation



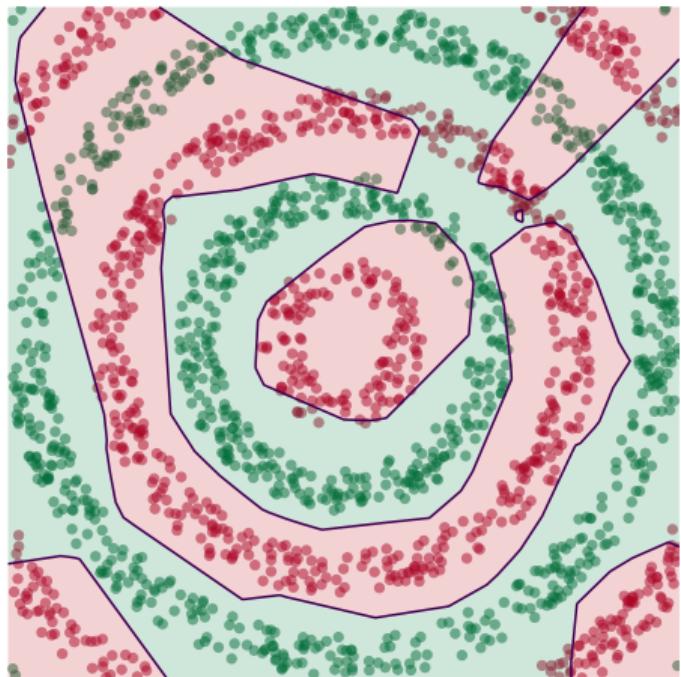
Learning a New Representation



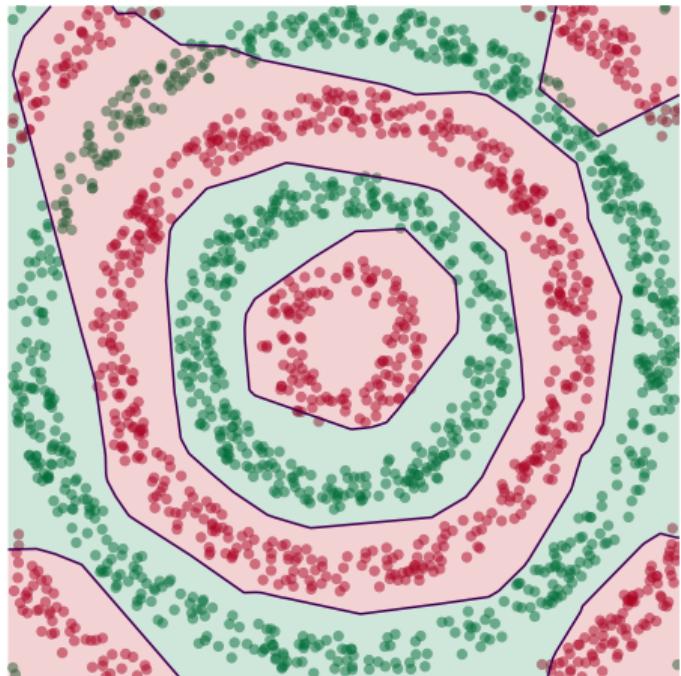
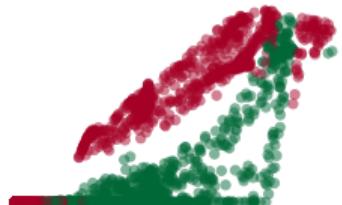
Learning a New Representation



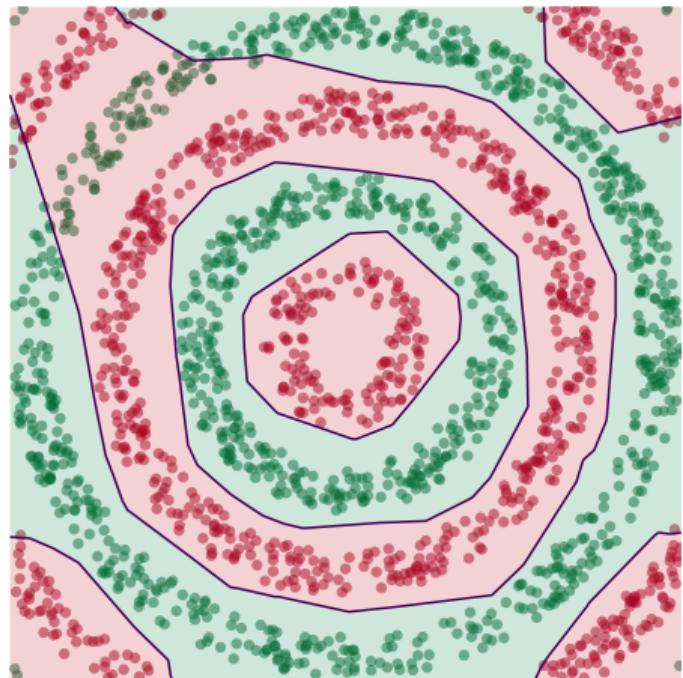
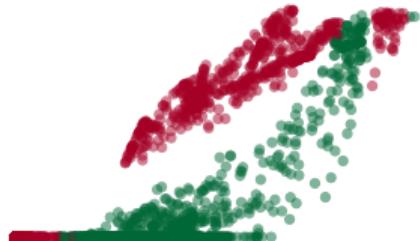
Learning a New Representation



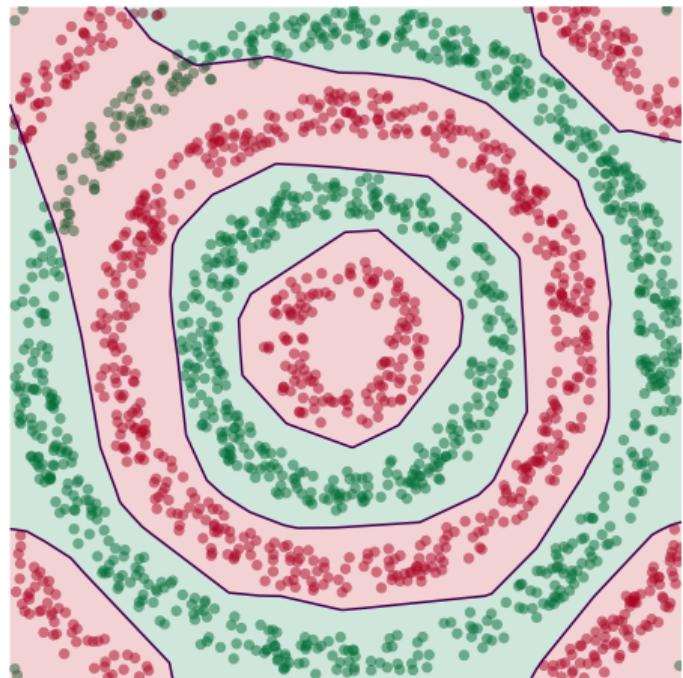
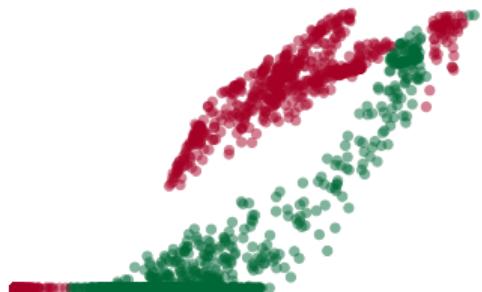
Learning a New Representation



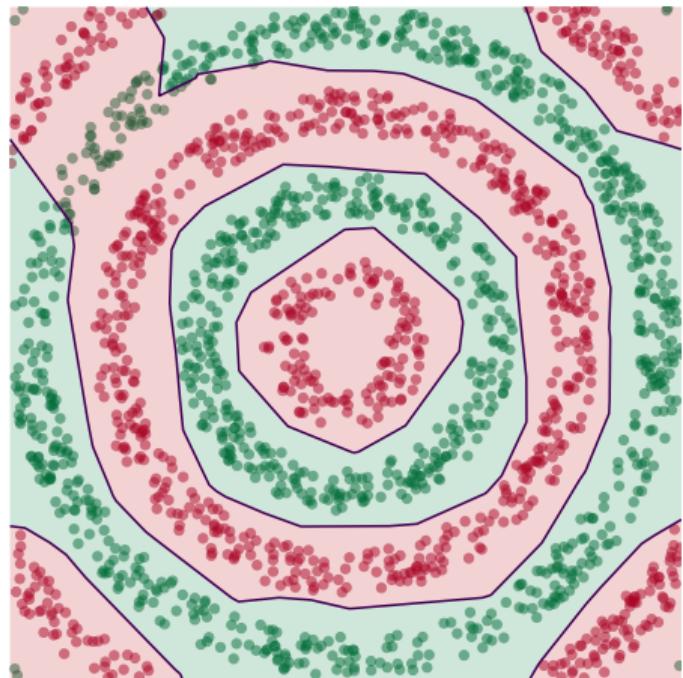
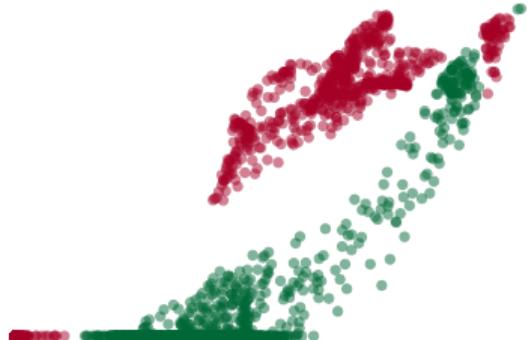
Learning a New Representation



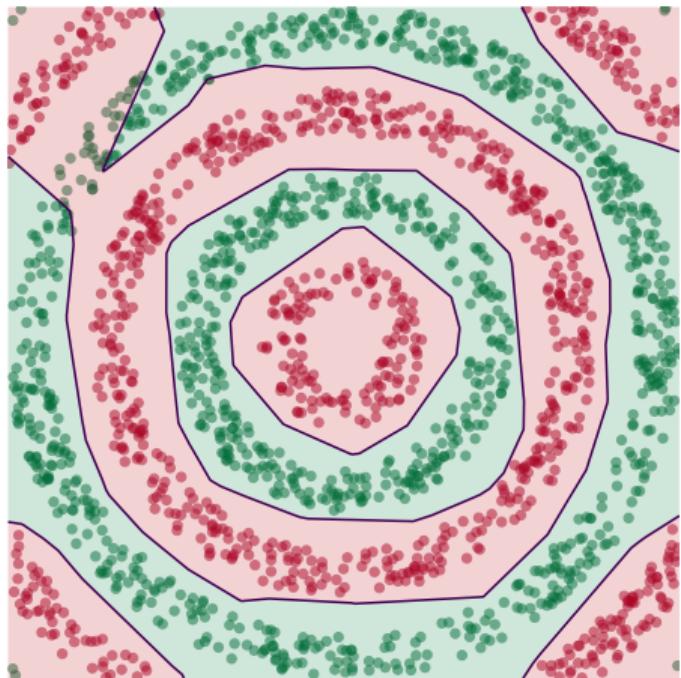
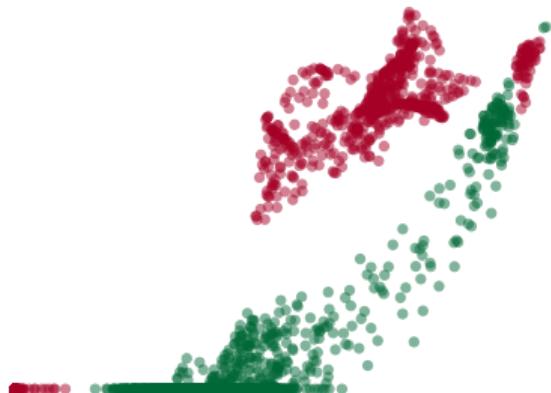
Learning a New Representation



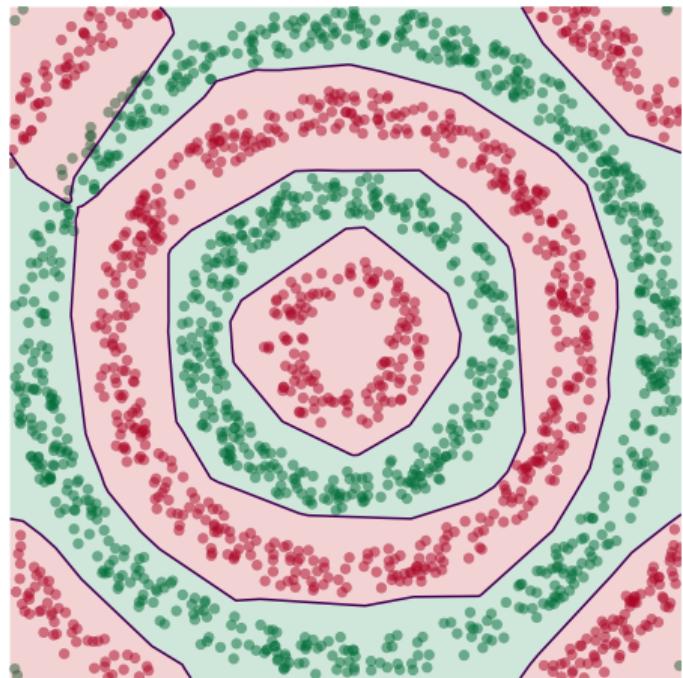
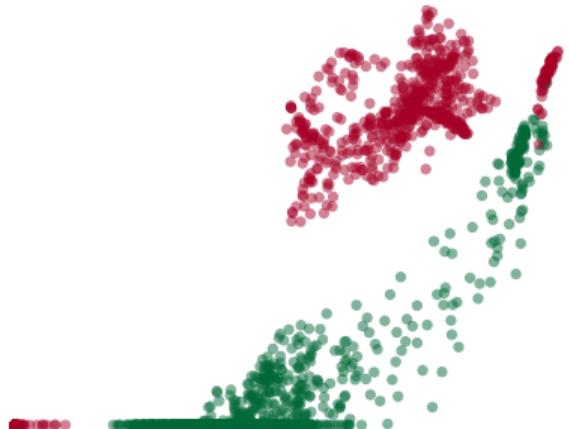
Learning a New Representation



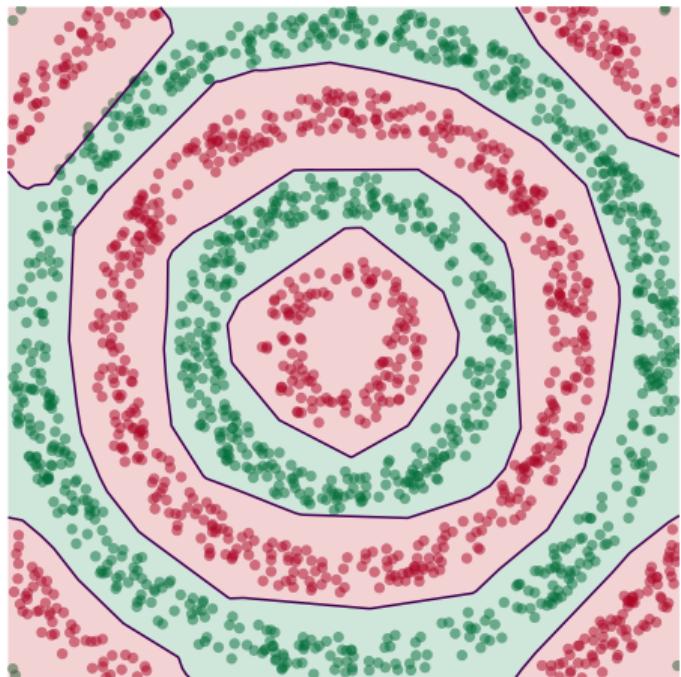
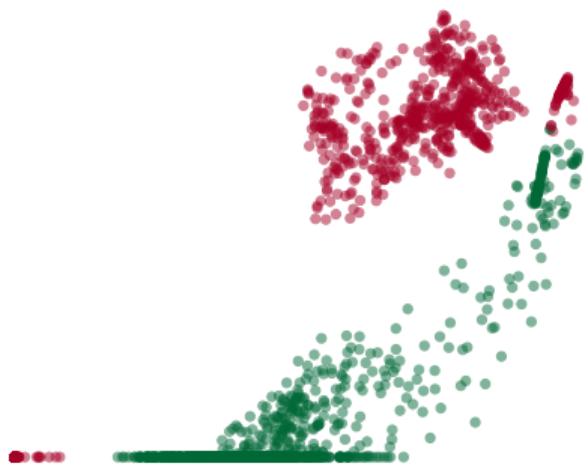
Learning a New Representation



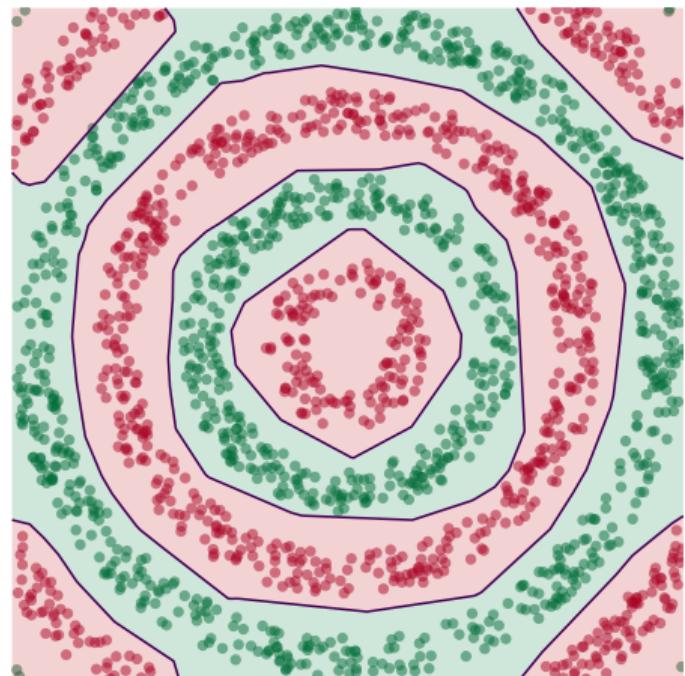
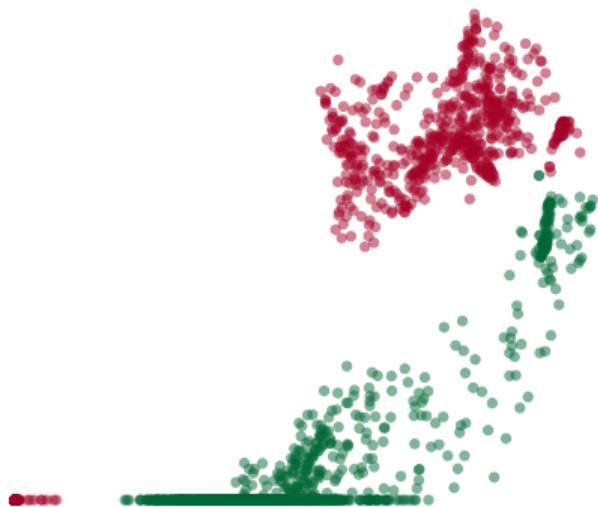
Learning a New Representation



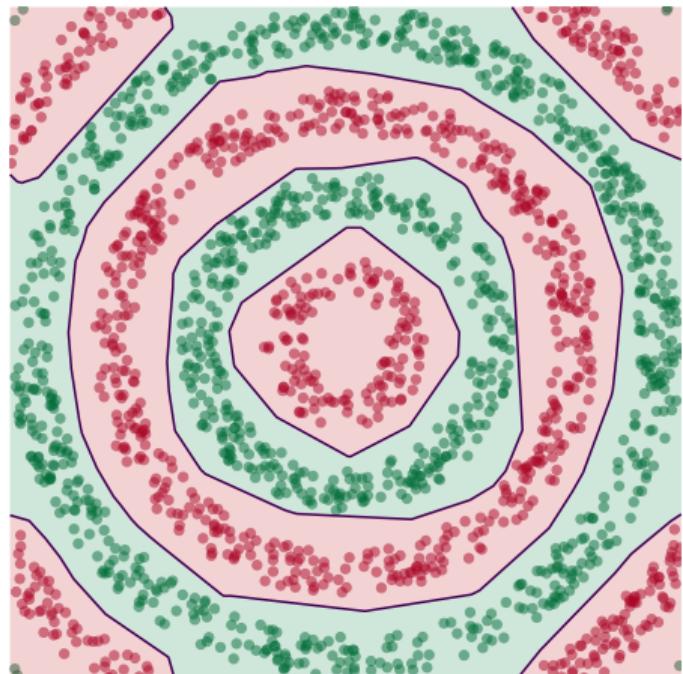
Learning a New Representation



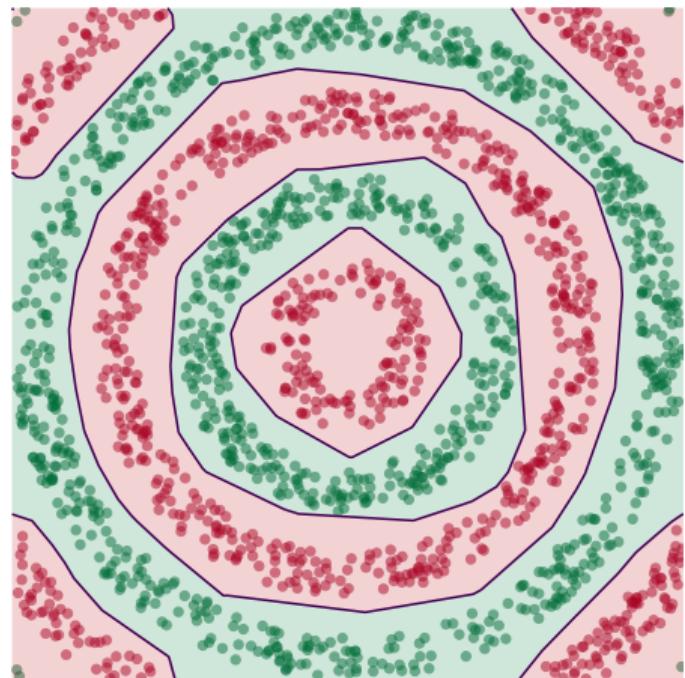
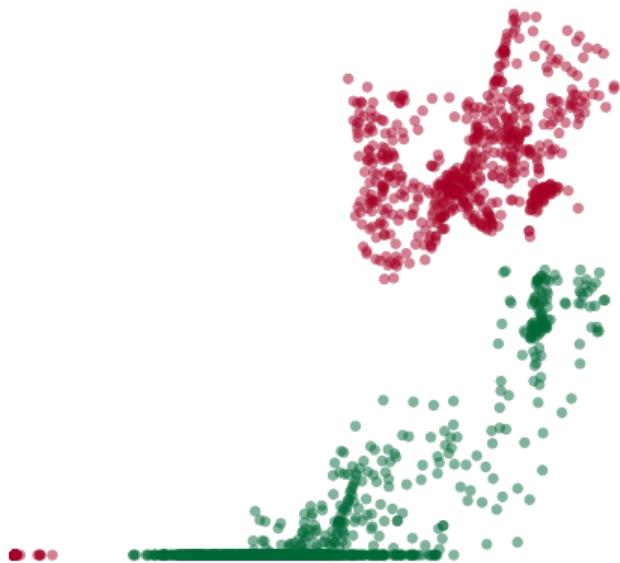
Learning a New Representation



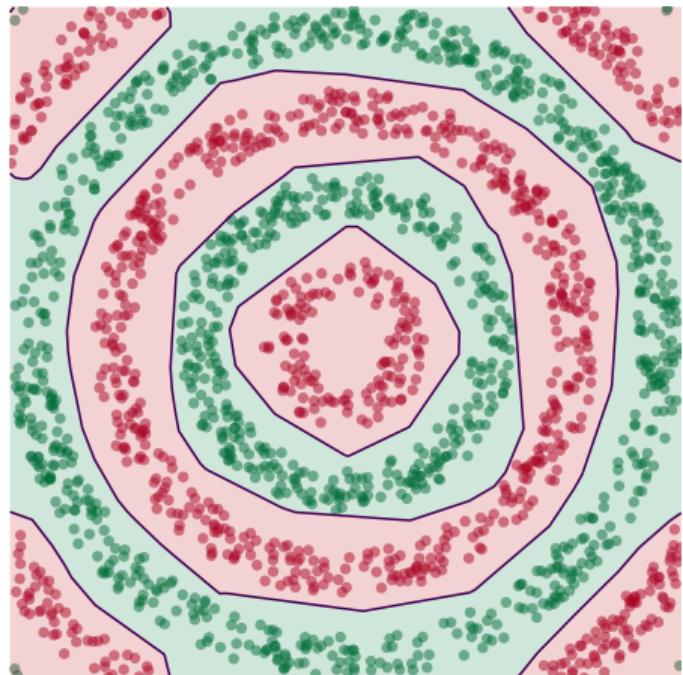
Learning a New Representation



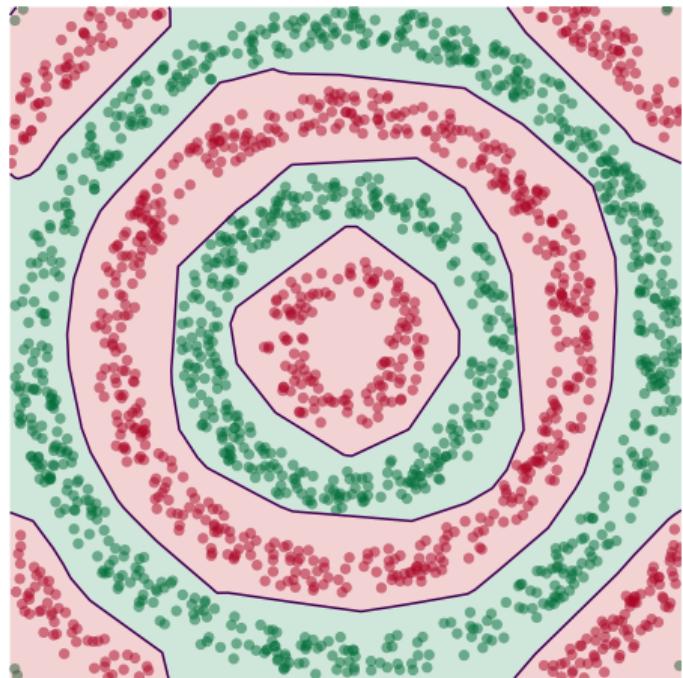
Learning a New Representation



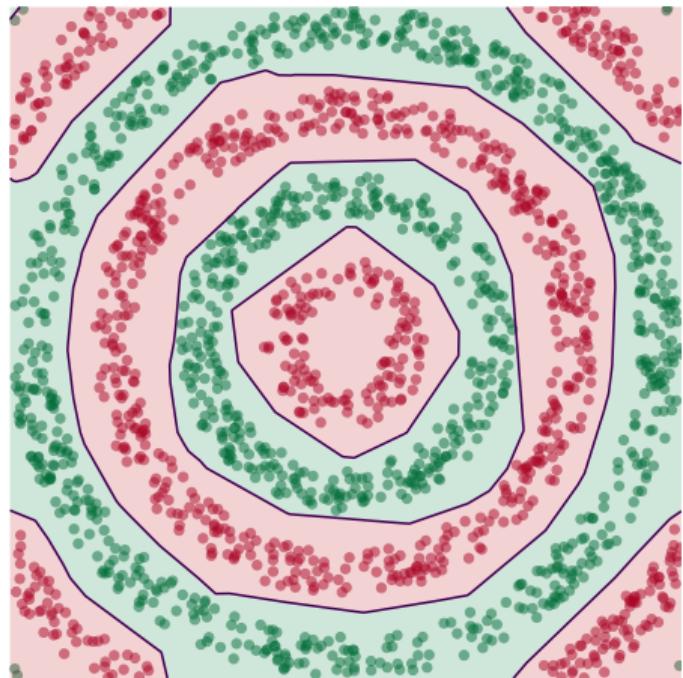
Learning a New Representation



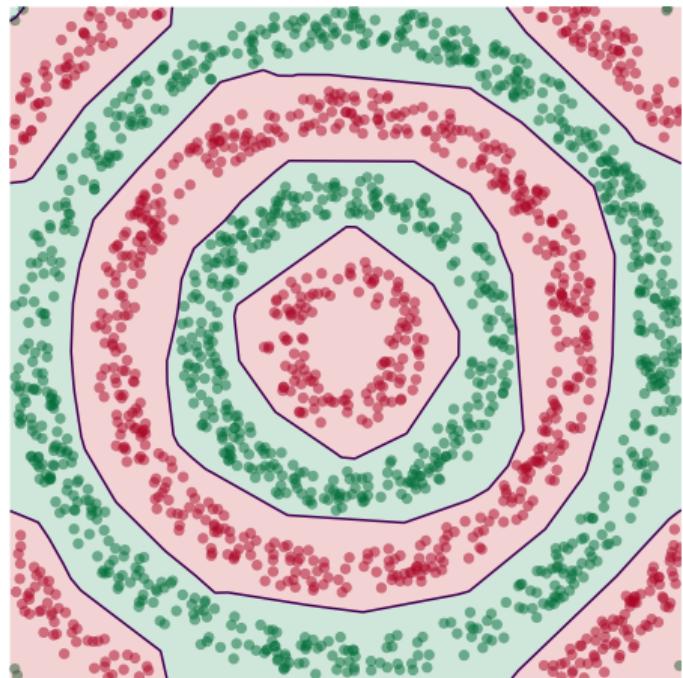
Learning a New Representation



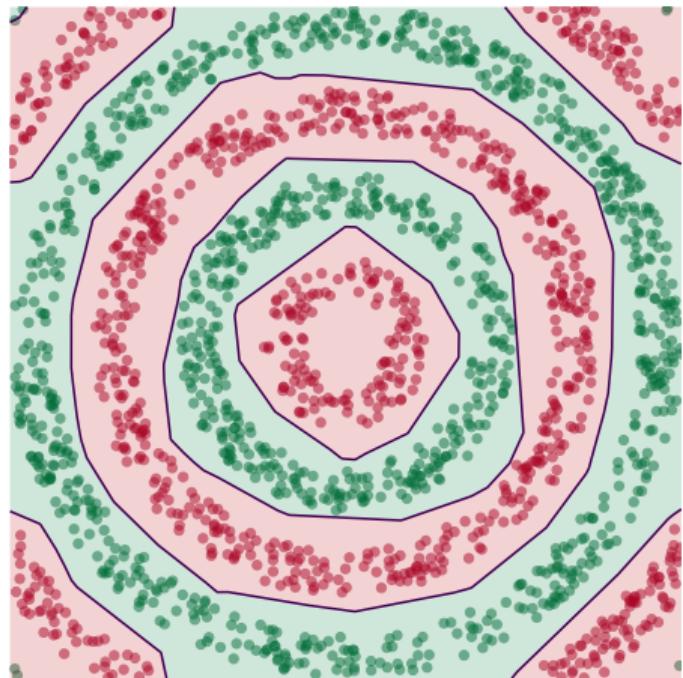
Learning a New Representation



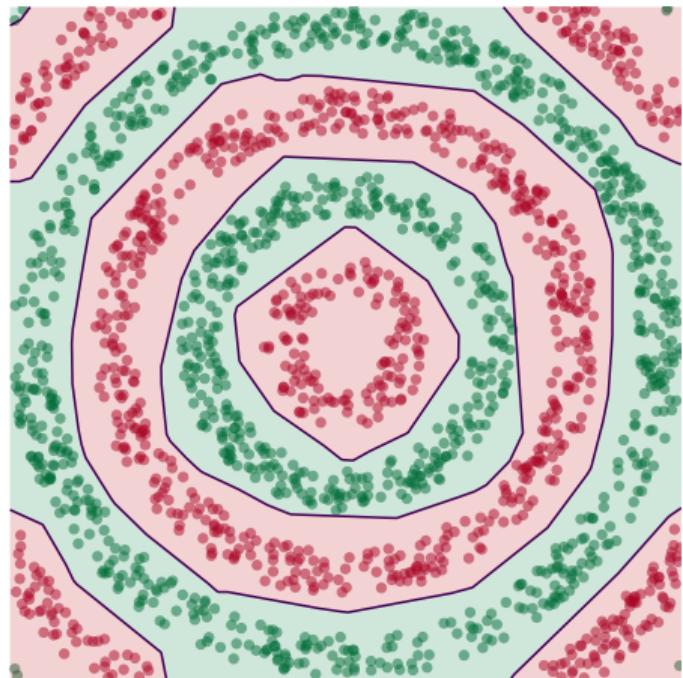
Learning a New Representation



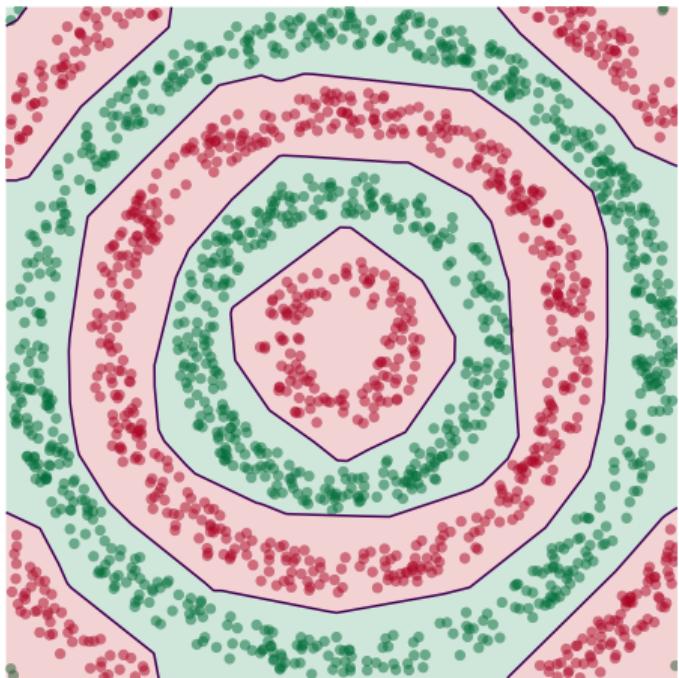
Learning a New Representation



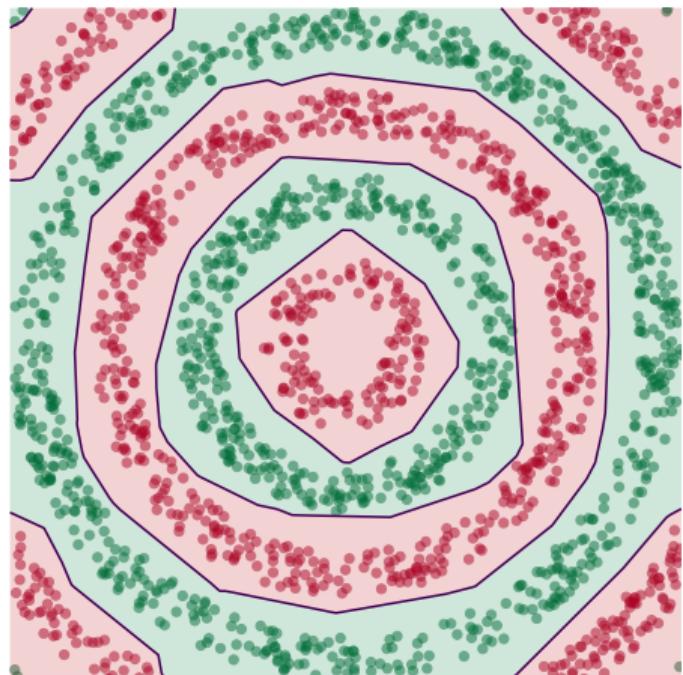
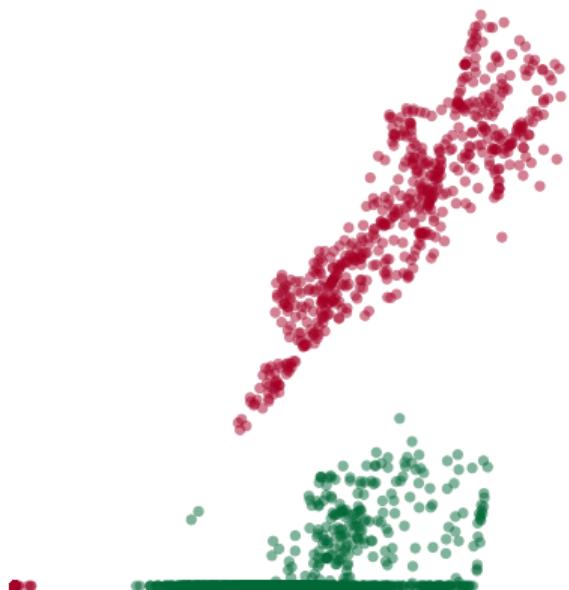
Learning a New Representation



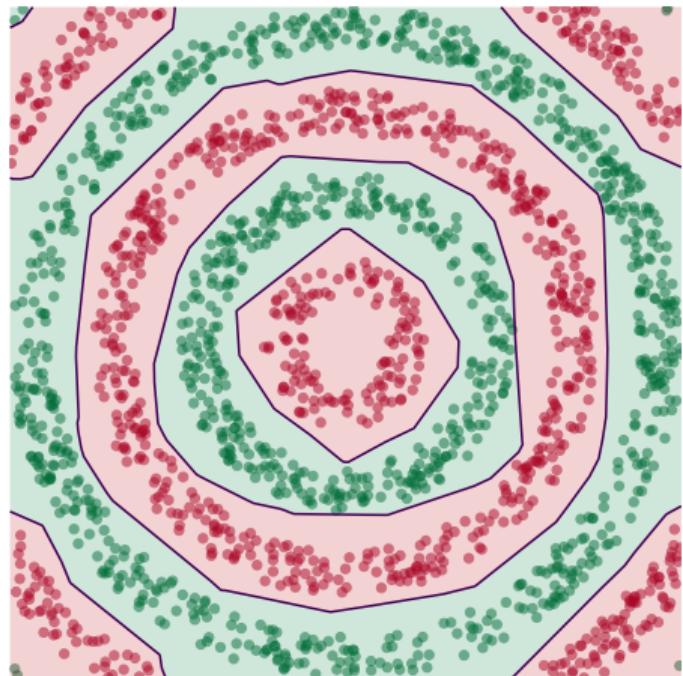
Learning a New Representation



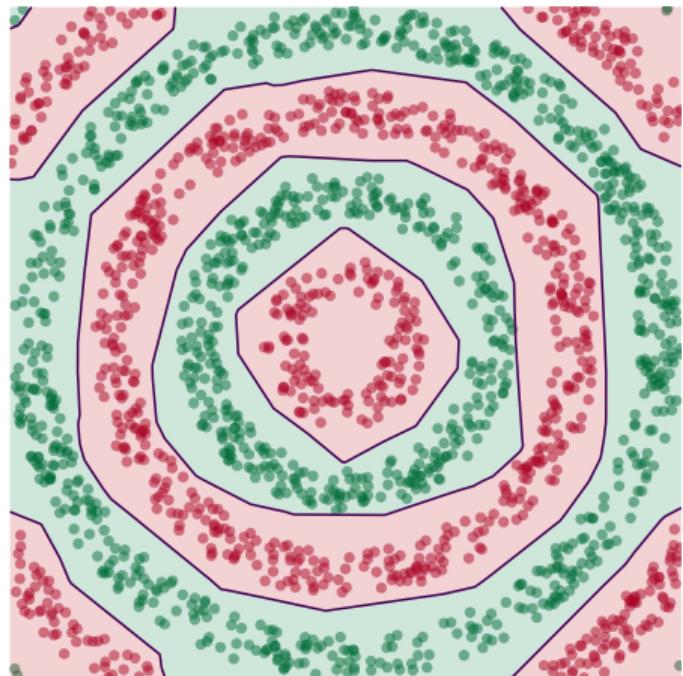
Learning a New Representation



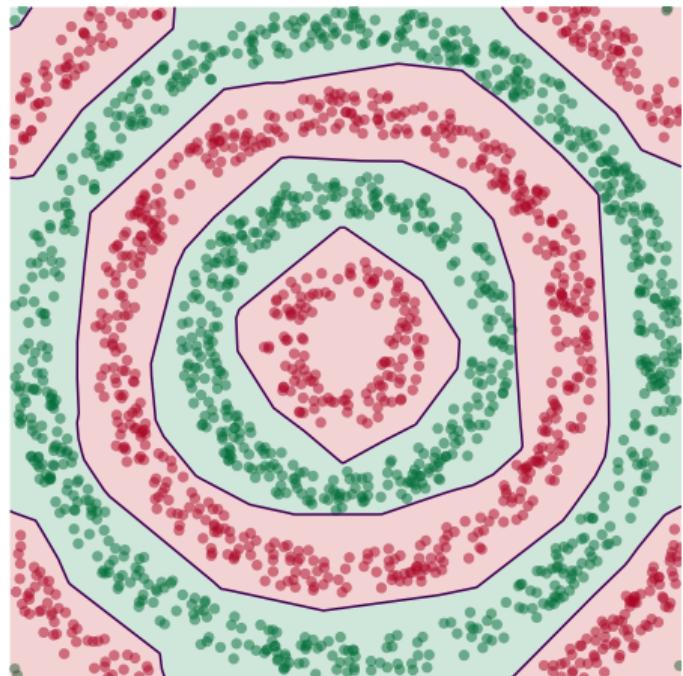
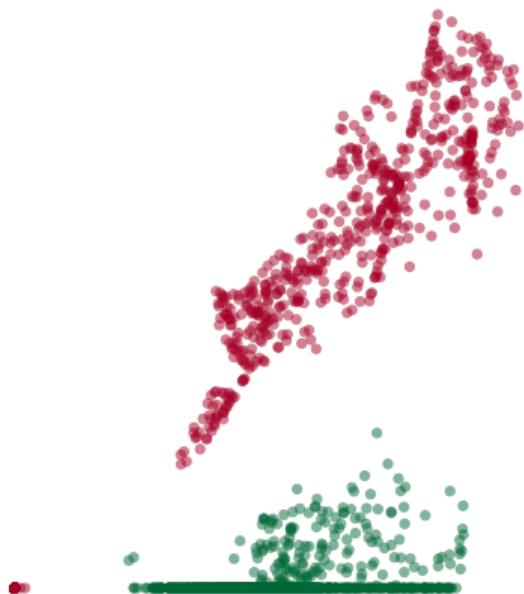
Learning a New Representation



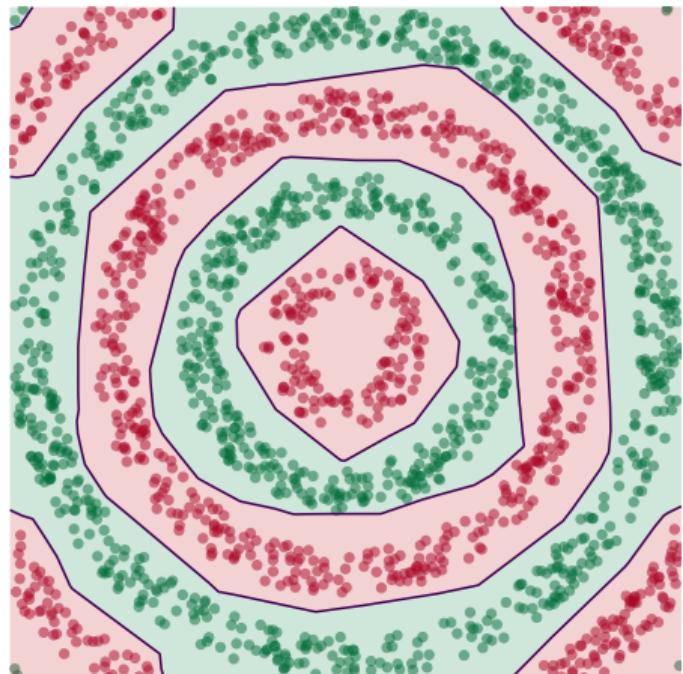
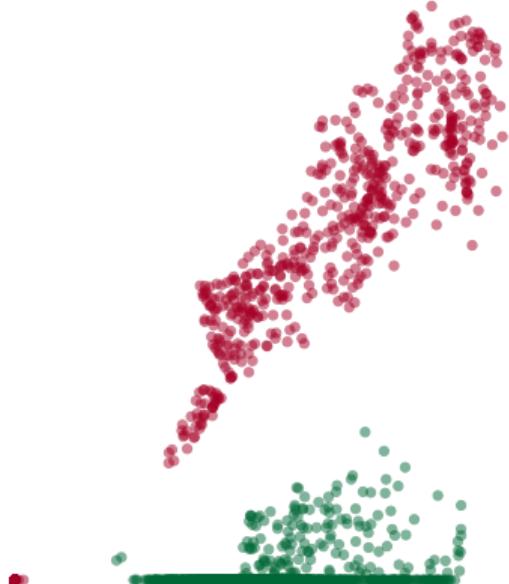
Learning a New Representation



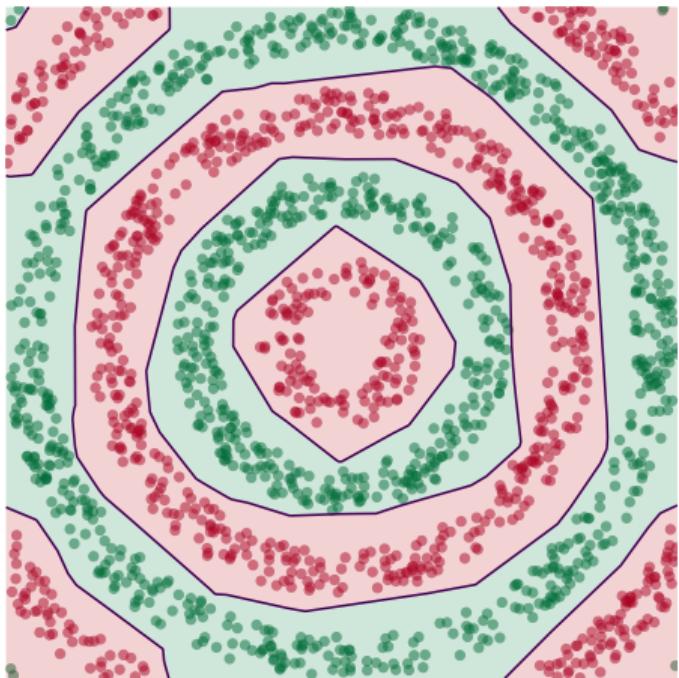
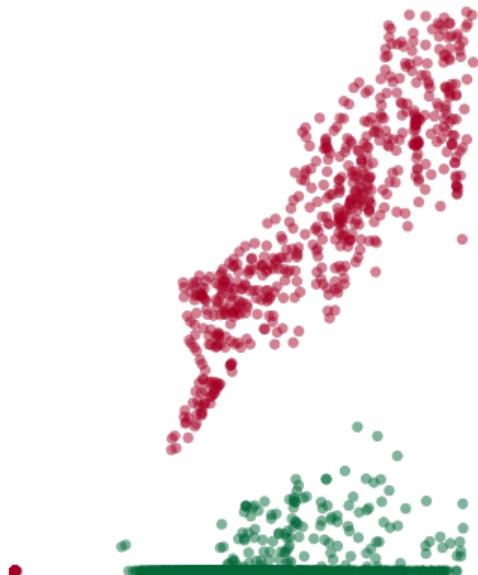
Learning a New Representation



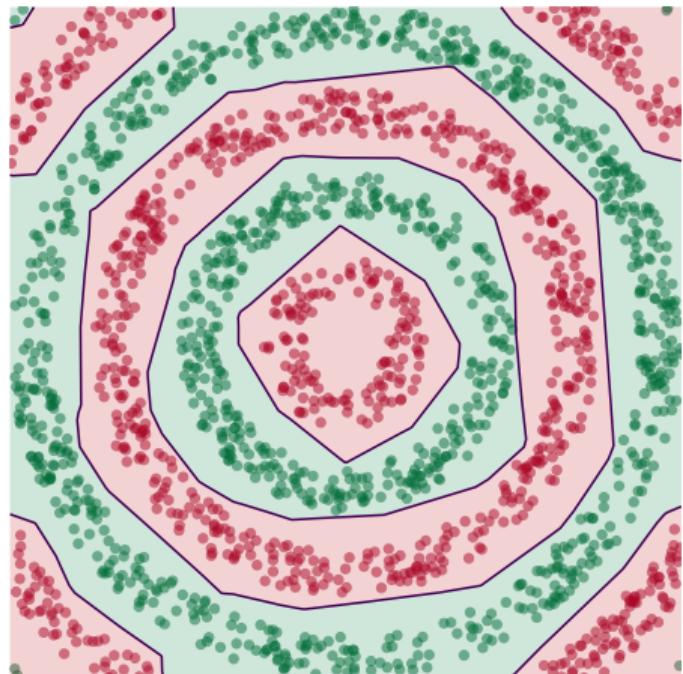
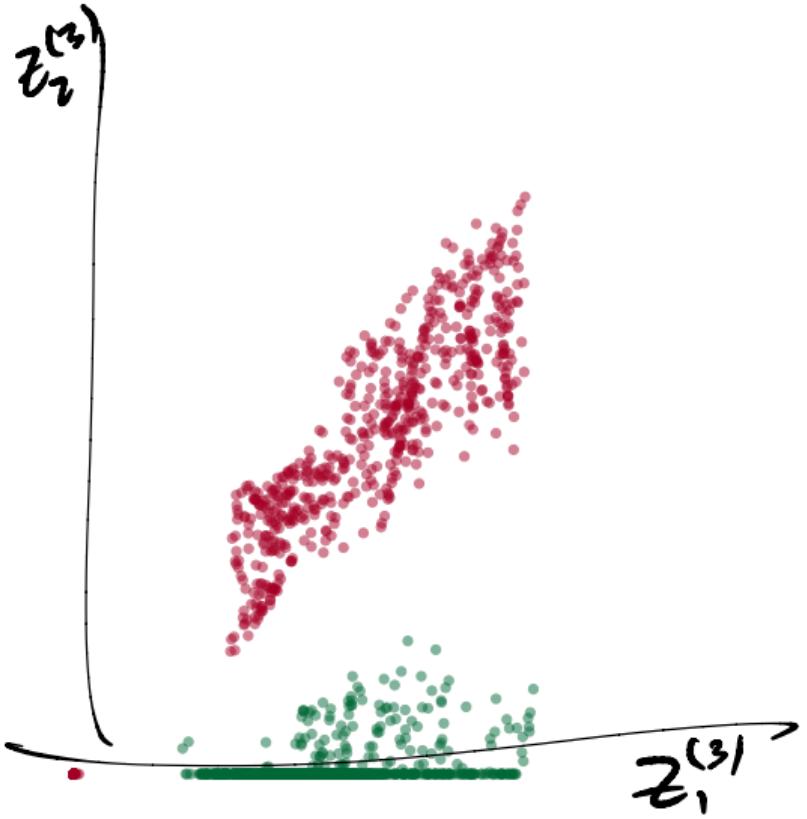
Learning a New Representation



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

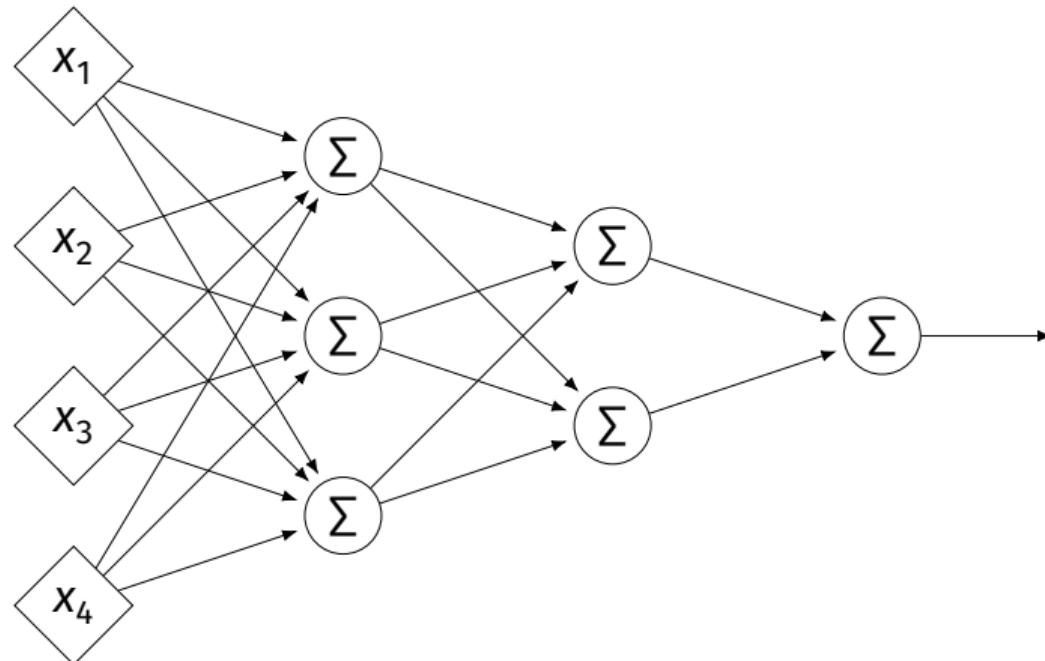
Representation Learning

Lecture 13 | Part 3

Training Neural Networks

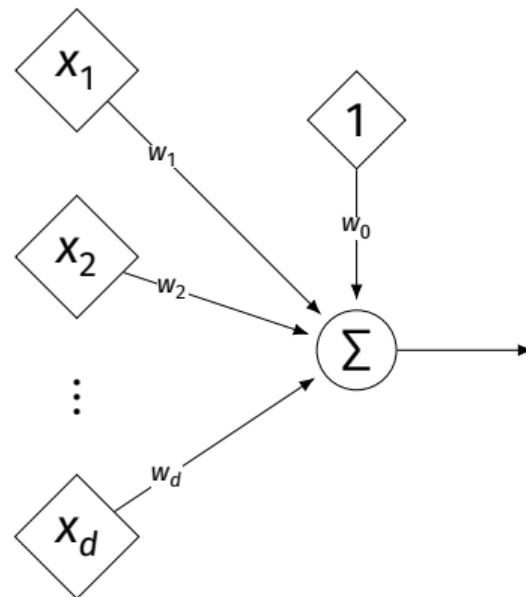
Training

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

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

$$\frac{d}{dx} f(x) = 0$$

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

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

$$\frac{d}{d\vec{w}} R(\vec{w}) = \nabla_{\vec{w}} R(\vec{w})$$

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:

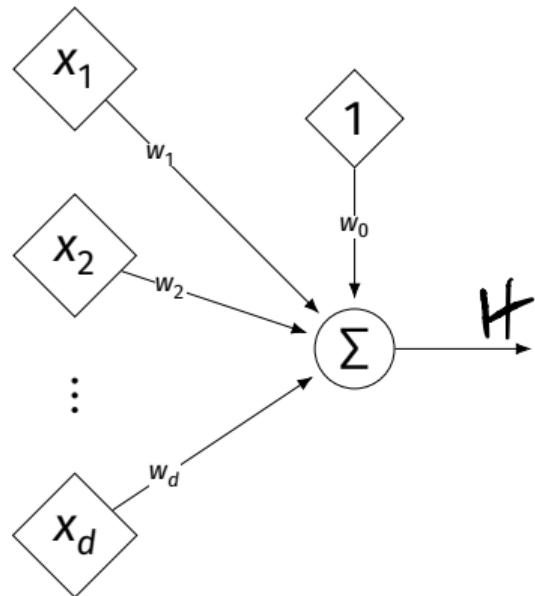
$$H(\vec{x}; \vec{w}) = w_0 + w_1 x_1 + \dots + w_d x_d$$

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

$$\begin{aligned}\nabla_{\vec{w}} H &= \left(\frac{\partial H}{\partial w_0}, \frac{\partial H}{\partial w_1}, \dots, \frac{\partial H}{\partial w_d} \right)^T \\ &= (1, x_1, x_2, \dots, x_d)^T\end{aligned}$$

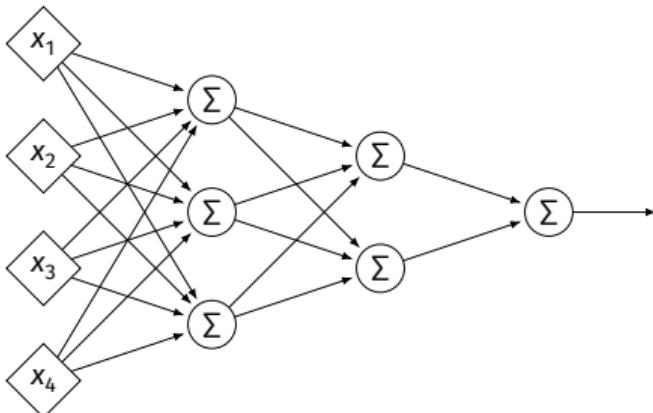
Example: Linear Model

- ▶ Consider $\partial H / \partial w_1$: \cancel{x}_1



Example: Neural Networks

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



Parameter Vectors

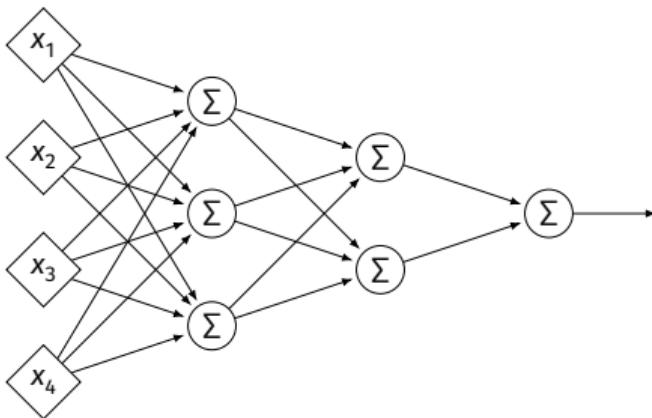
- ▶ It is often useful to pack all of the network's weights into a **parameter vector**, \vec{w} .
- ▶ Order is arbitrary:

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

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

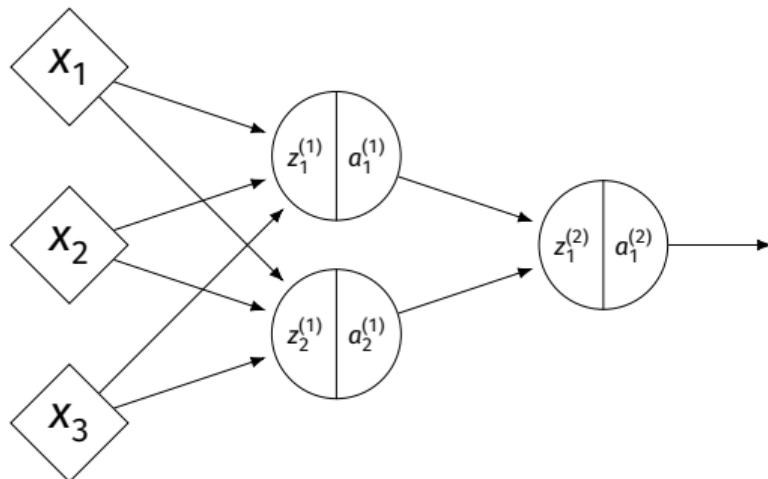
Example: Neural Networks

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

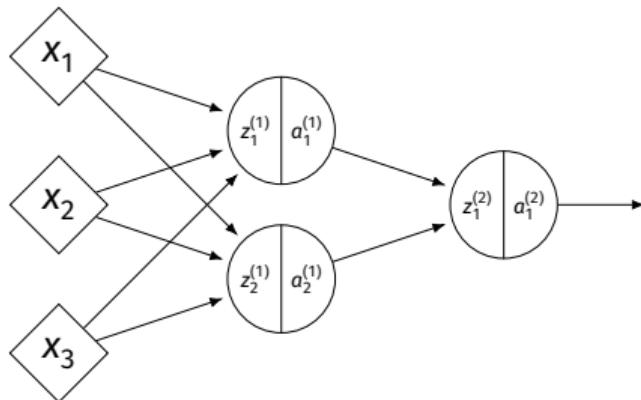


Recall: Chain Rule

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

Example: Neural Network

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

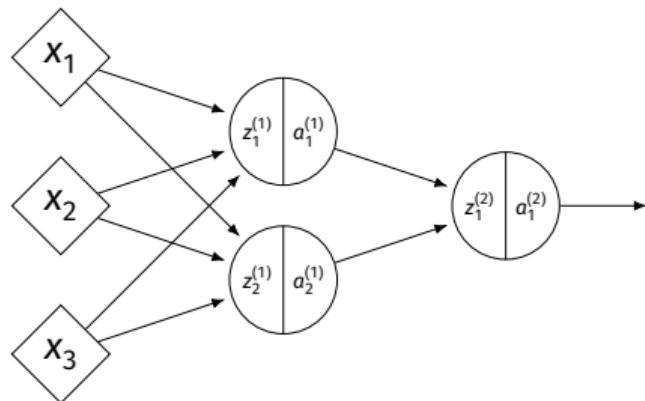


$$W^{(1)} = \begin{pmatrix} 3 & 2 \\ 4 & 1 \\ 1 & 3 \end{pmatrix} \quad W^{(2)} = \begin{pmatrix} 3 \\ 1 \end{pmatrix}$$

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

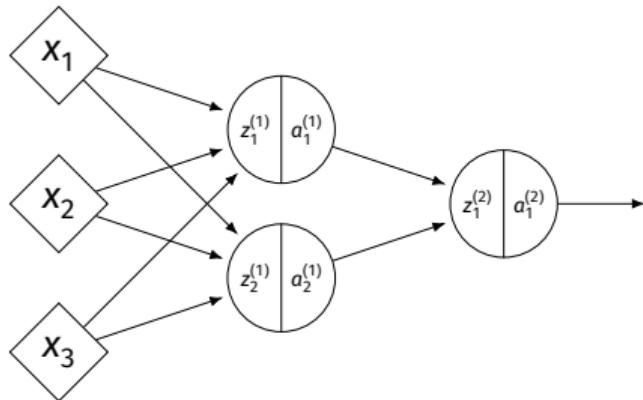
Example: Neural Network

- ▶ 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 .

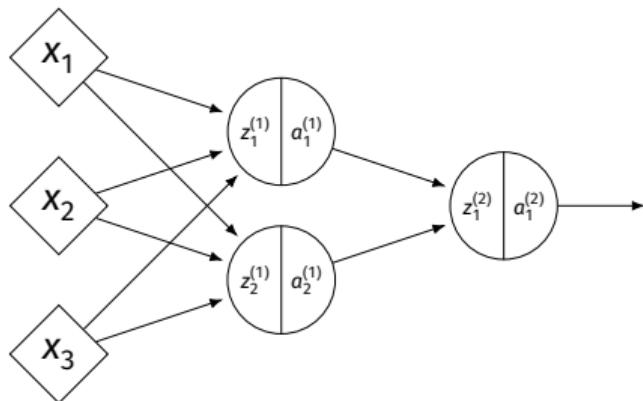


$$W^{(1)} = \begin{pmatrix} 3 & 2 \\ 4 & 1 \\ 1 & 3 \end{pmatrix} \quad W^{(2)} = \begin{pmatrix} 3 \\ 1 \end{pmatrix}$$

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

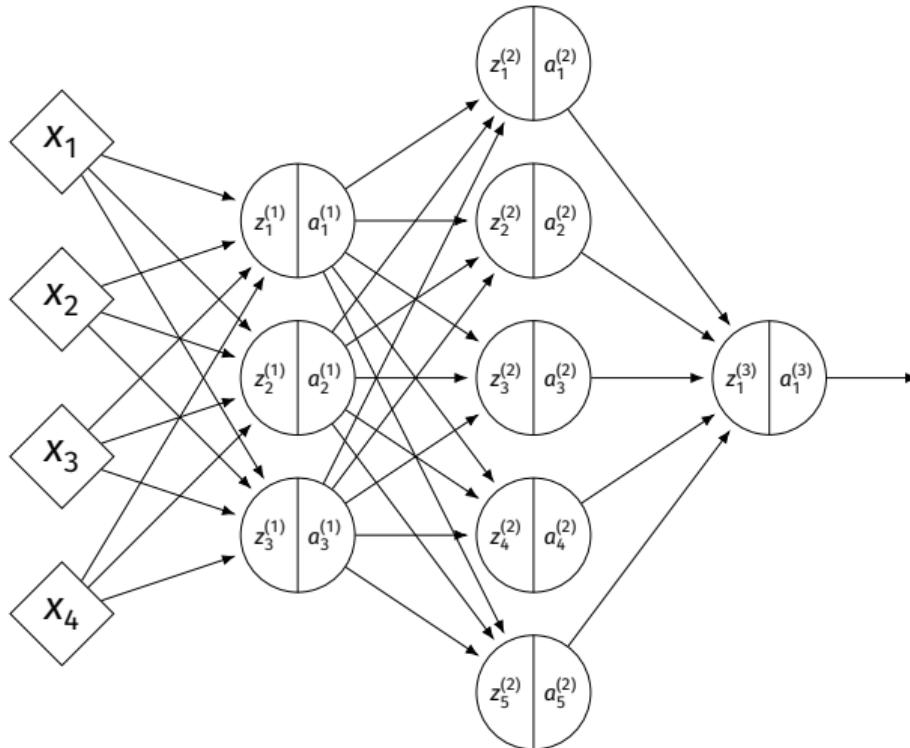
Example: Neural Network

- ▶ 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.

dsc-capstone.org/enrollment

DSC 140B

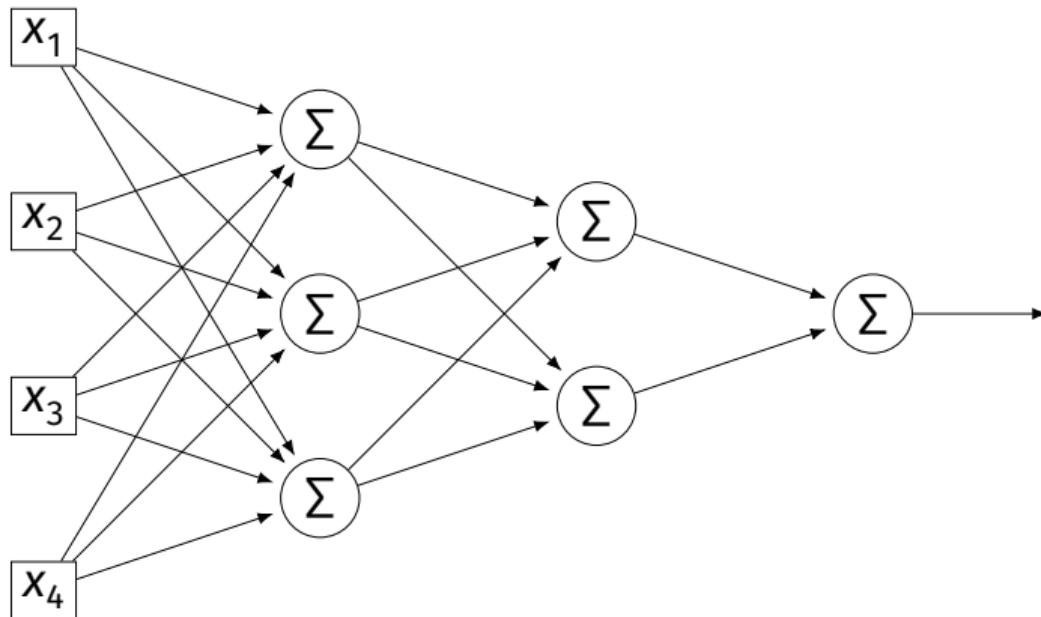
Representation Learning

Lecture 14 | Part 1

Training Neural Networks

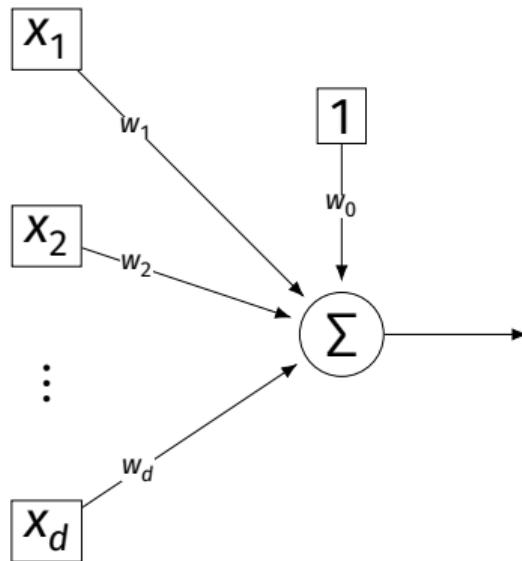
Training

- ▶ 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.

$$\begin{aligned} & (\hat{H} - y)^2 \\ & 2(\hat{H} - y) \end{aligned}$$

- ▶ 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

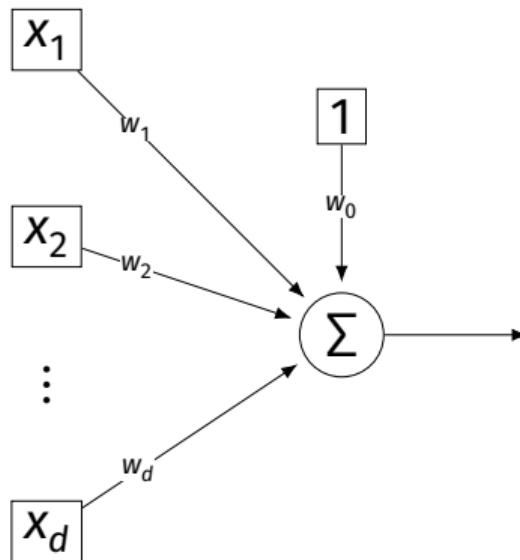
- ▶ Suppose H is a linear prediction function:

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

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

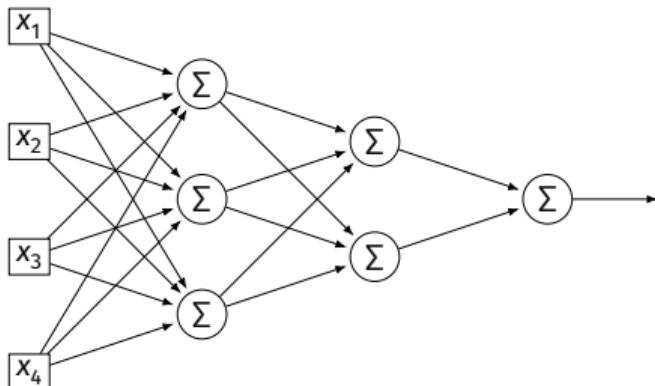
Example: Linear Model

- ▶ 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

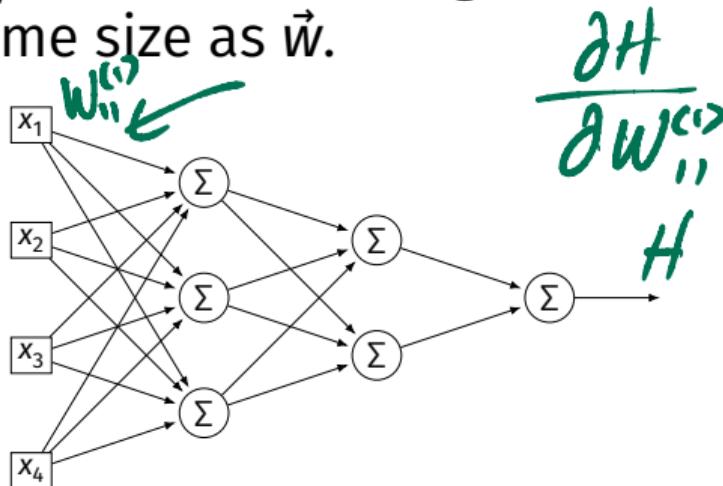
- ▶ It is often useful to pack all of the network's weights into a **parameter vector**, \vec{w} .
- ▶ Order is arbitrary:

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

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

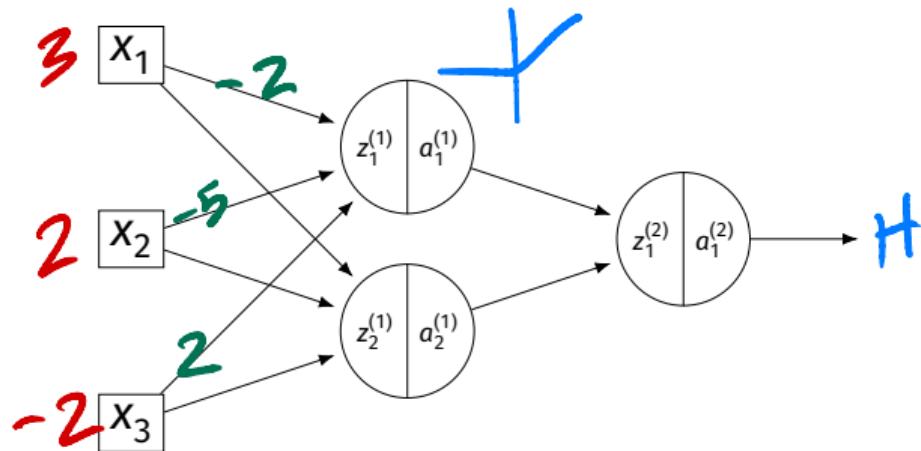
Gradient of Neural Network

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



Exercise

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

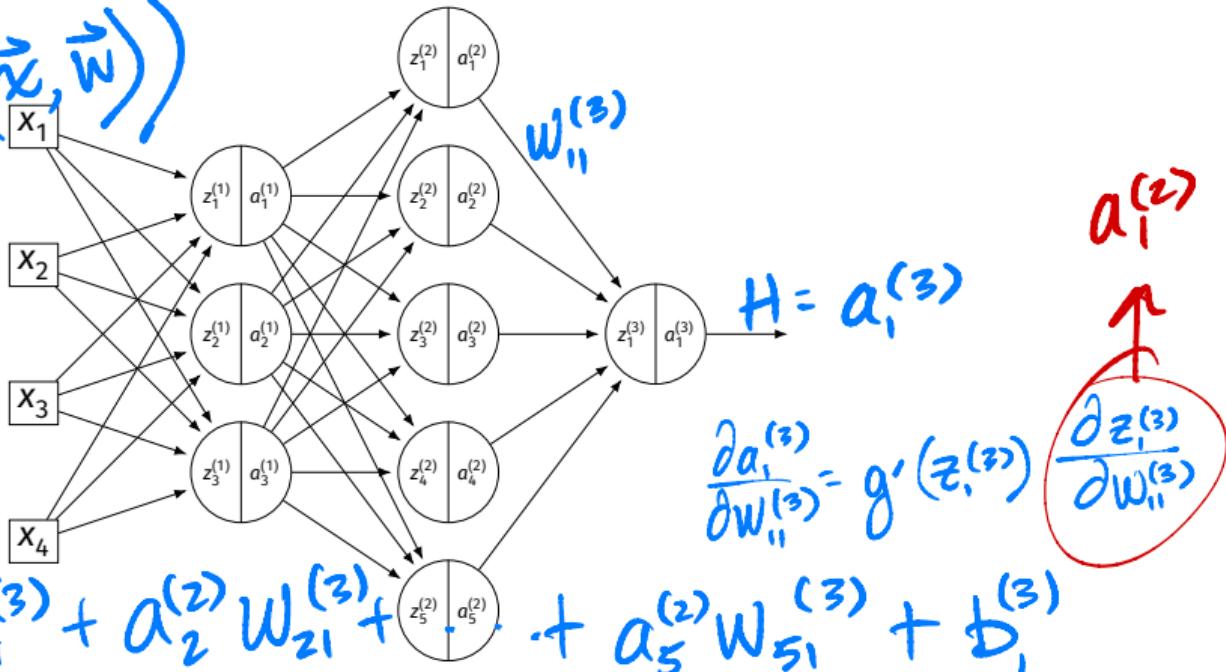


Example

$$\frac{\partial a_i^{(3)}}{\partial w_{ii}^{(3)}} = g'(z_i^{(3)}) a_i^{(2)}$$

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

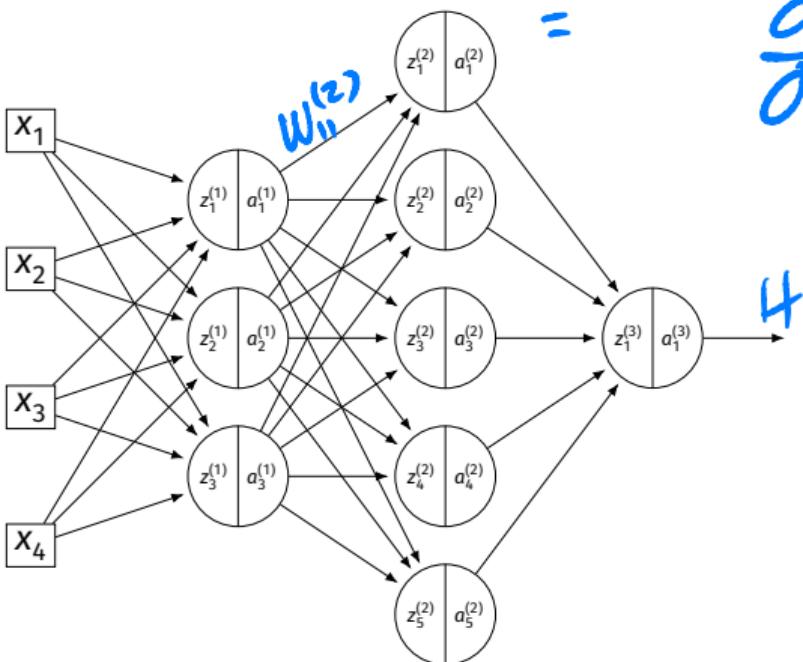
$$a_i^{(3)} = g(z_i^{(3)}(\vec{x}, \vec{w}))$$



$$\frac{1}{2} \cdot \frac{2}{3} = \frac{1}{3}$$

Example

► Consider $\partial H / \partial W_{11}^{(2)}$: $\frac{\partial H}{\partial W_{11}^{(2)}} = \frac{\partial H}{\partial a_1^{(3)}}$

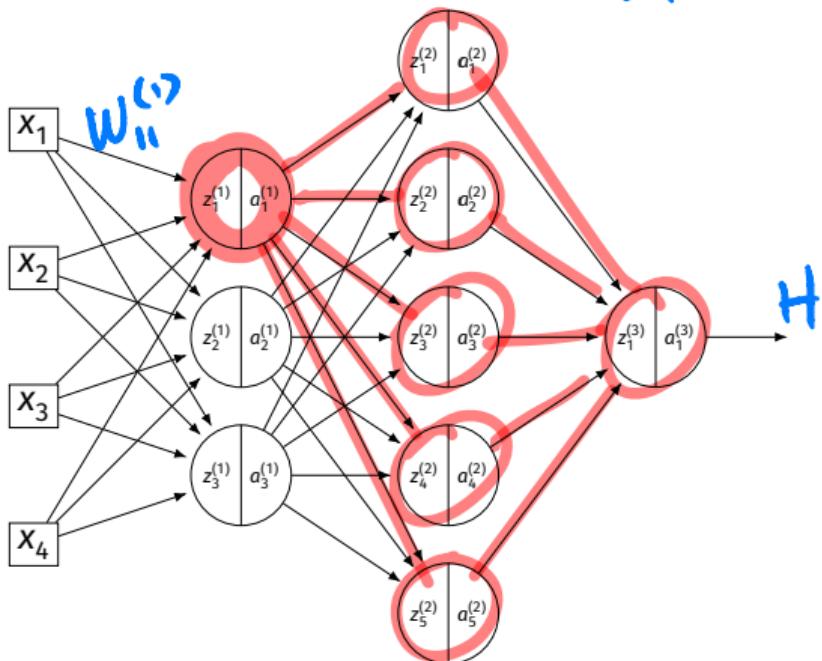


$$\begin{aligned} \frac{\partial a_i^{(3)}}{\partial W_{11}^{(2)}} &= \\ \frac{\partial a_i^{(3)}}{\partial z_i^{(3)}} \frac{\partial z_i^{(3)}}{\partial a_i^{(2)}} \frac{\partial a_i^{(2)}}{\partial z_i^{(2)}} &= \\ \times \frac{\partial z_i^{(2)}}{\partial W_{11}^{(2)}} & \end{aligned}$$

Example

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

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



A Better Way

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

DSC 140B

Representation Learning

Lecture 14 | Part 2

Backpropagation

Gradient of a Network

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

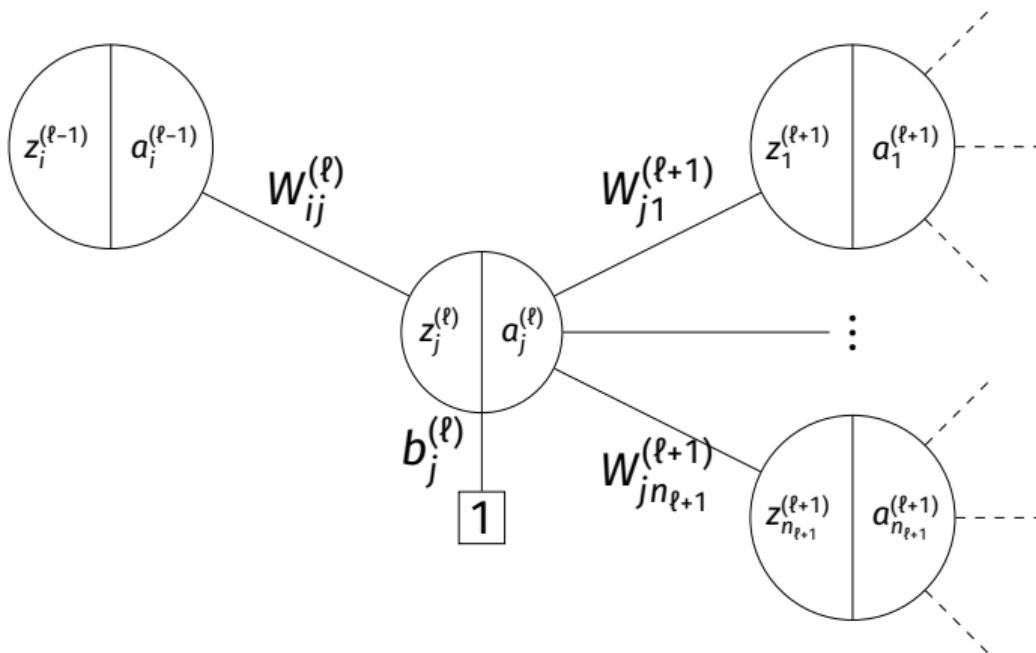
Recall: The Chain Rule

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

Some Notation

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

Arbitrary Node



► $\frac{\partial H}{\partial W_{ij}^{(\ell)}} ?$

► $\frac{\partial H}{\partial b_j^{(\ell)}} ?$

Claim #1

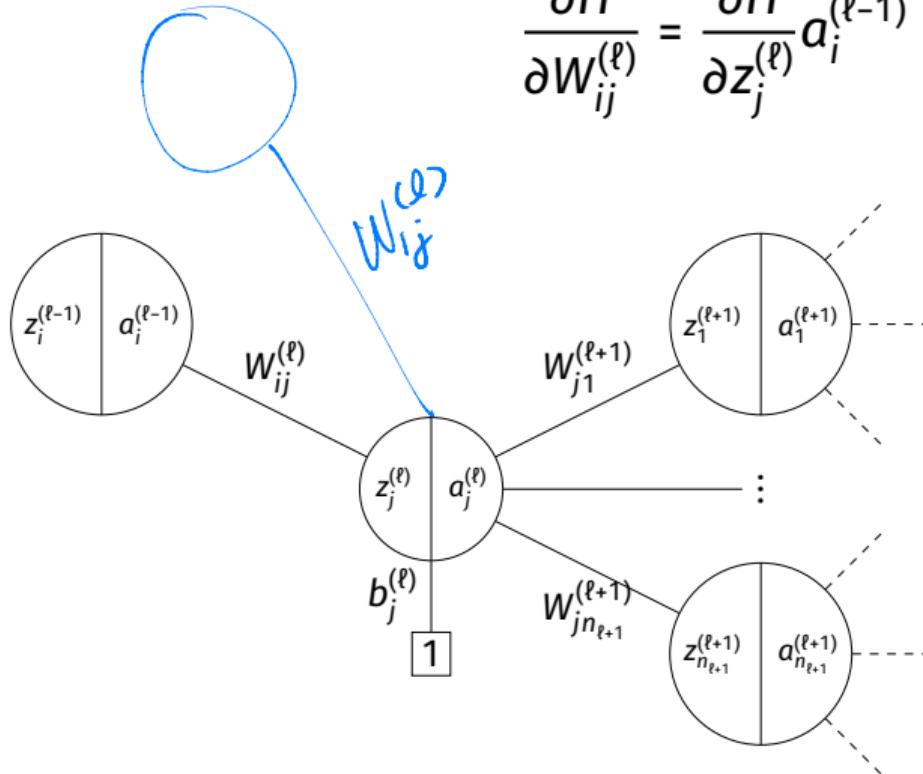
$$z_j^{(\ell)} = w_{ij}^{(\ell)} a_i^{(\ell-1)} + w_{iz}^{(\ell)} a_z^{(\ell-1)} + \dots + w_{n_{\ell-1}j}^{(\ell)} a_{n_{\ell-1}}^{(\ell-1)} + b_j^{(\ell)}$$

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

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

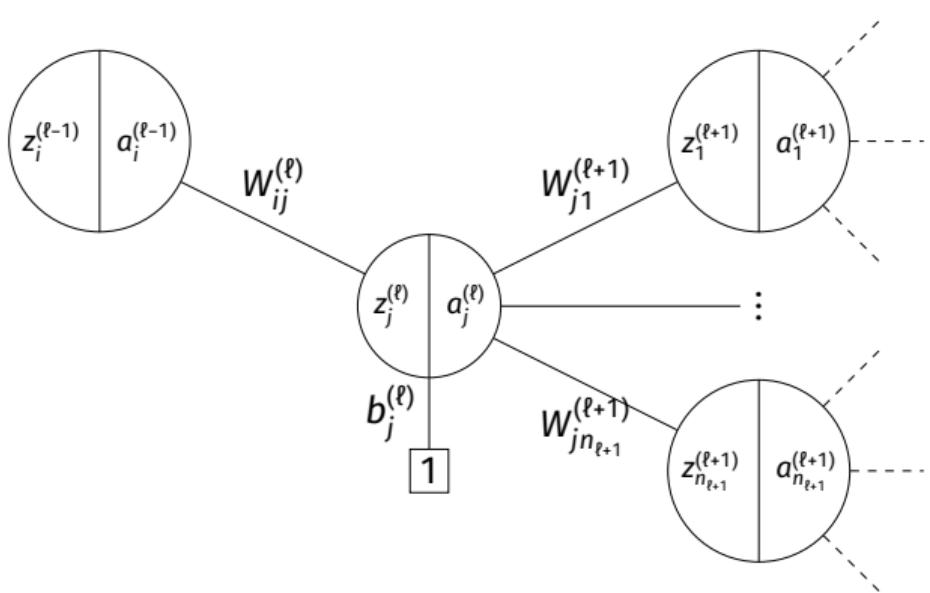


$$\downarrow a_i^{(\ell-1)}$$



Claim #2

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



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

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

Claim #3

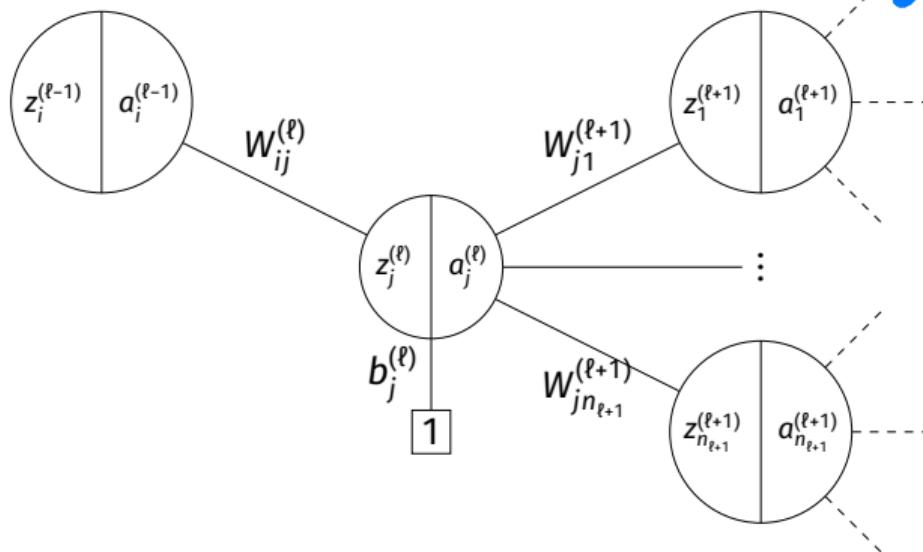
$$\frac{\partial H}{\partial a_j^{(\ell)}} = \sum_{k=1}^{n_{\ell+1}} \frac{\partial H}{\partial z_k^{(\ell+1)}} W_{jk}^{(\ell+1)}$$

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

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

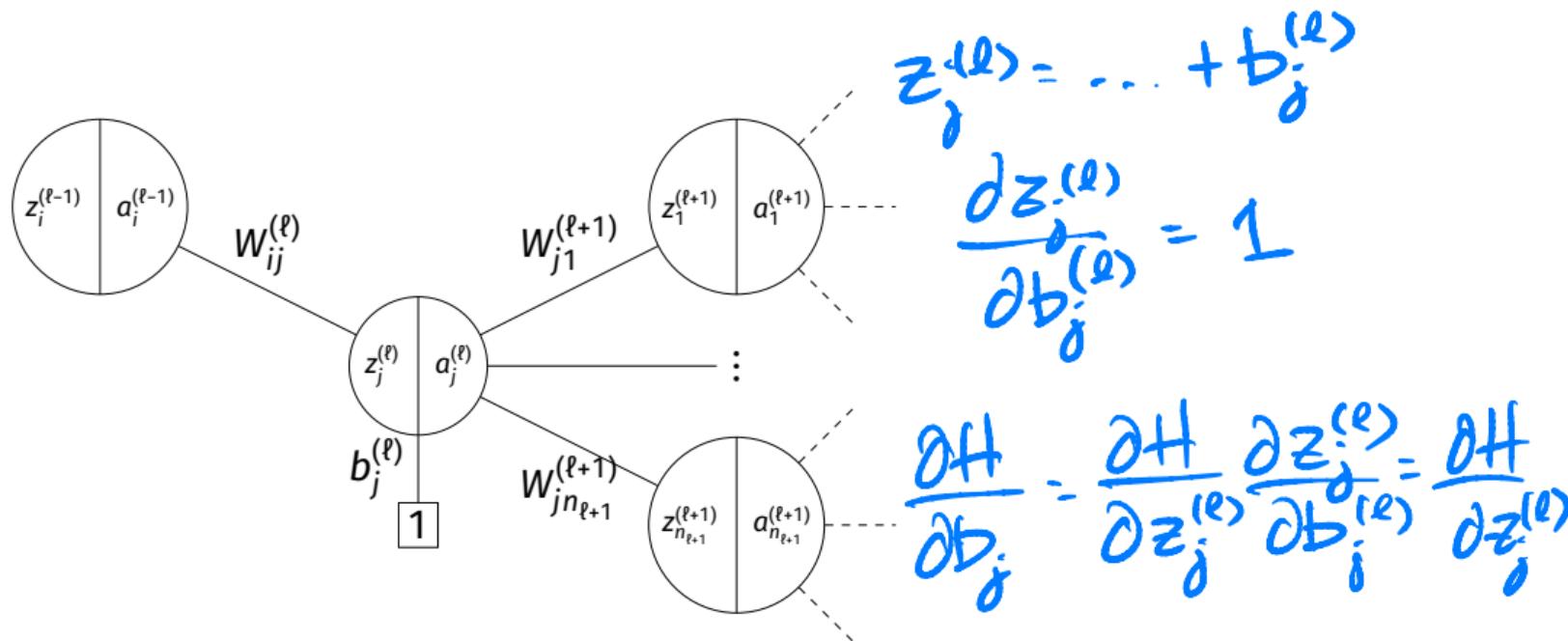
$W_{jk}^{(\ell+1)}$

$\frac{\partial z_k^{(\ell+1)}}{\partial a_j^{(\ell)}}$



Exercise

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



General Formulas

- ▶ For any node in any neural network¹, we have the following recursive formulas:

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

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

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

$$\blacktriangleright \frac{\partial H}{\partial b_j^{(\ell)}} = \frac{\partial H}{\partial z_j^{(\ell)}}$$

¹Fully-connected, feedforward network

Main Idea

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

Backpropagation

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

Backpropagation

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

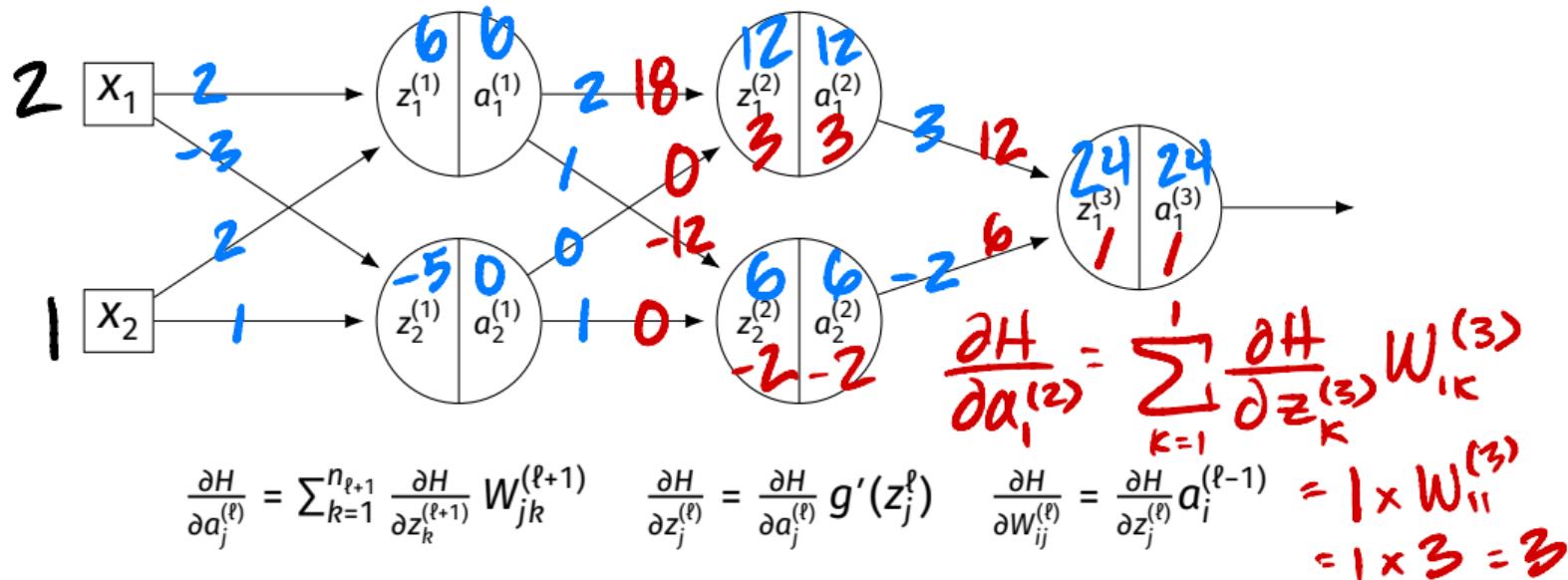
1. Evaluate the network to compute $z_i^{(\ell)}$ and $a_i^{(\ell)}$ for all nodes.
2. For each layer ℓ from last to first:
 - ▶ Compute $\frac{\partial H}{\partial a_j^{(\ell)}} = \sum_{k=1}^{n_{\ell+1}} \frac{\partial H}{\partial z_k^{(\ell+1)}} W_{jk}^{(\ell+1)}$
 - ▶ Compute $\frac{\partial H}{\partial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

Example

$$H = a_1^{(3)}$$

Compute the entries of the gradient given:

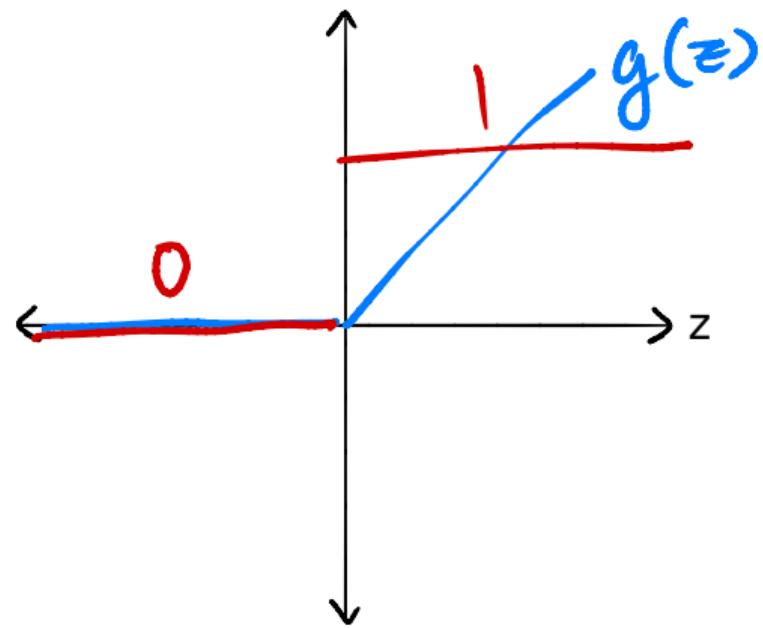
$$W^{(1)} = \begin{pmatrix} 2 & -3 \\ 2 & 1 \end{pmatrix} \quad W^{(2)} = \begin{pmatrix} 2 & 1 \\ 0 & 1 \end{pmatrix} \quad W^{(3)} = \begin{pmatrix} 3 \\ -2 \end{pmatrix} \quad \vec{x} = (2, 1)^T \quad g(z) = \text{ReLU}$$



Aside: Derivative of ReLU

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

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



Summary: Backprop

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

DSC 140B

Representation Learning

Lecture 15 | Part 1

Backpropagation

Gradient of a Network

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

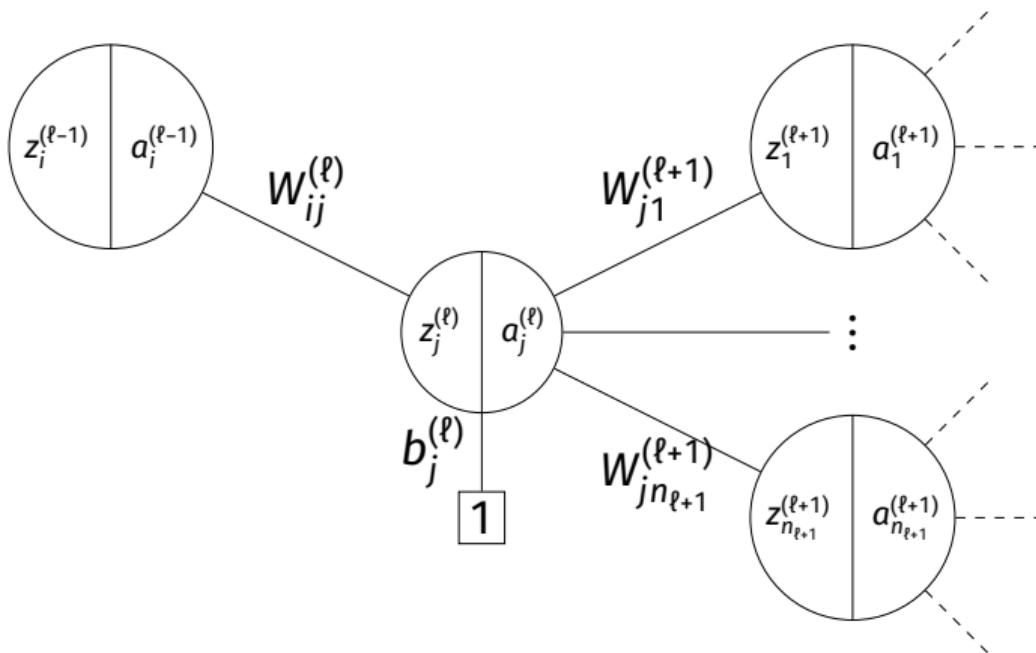
Recall: The Chain Rule

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

Some Notation

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

Arbitrary Node

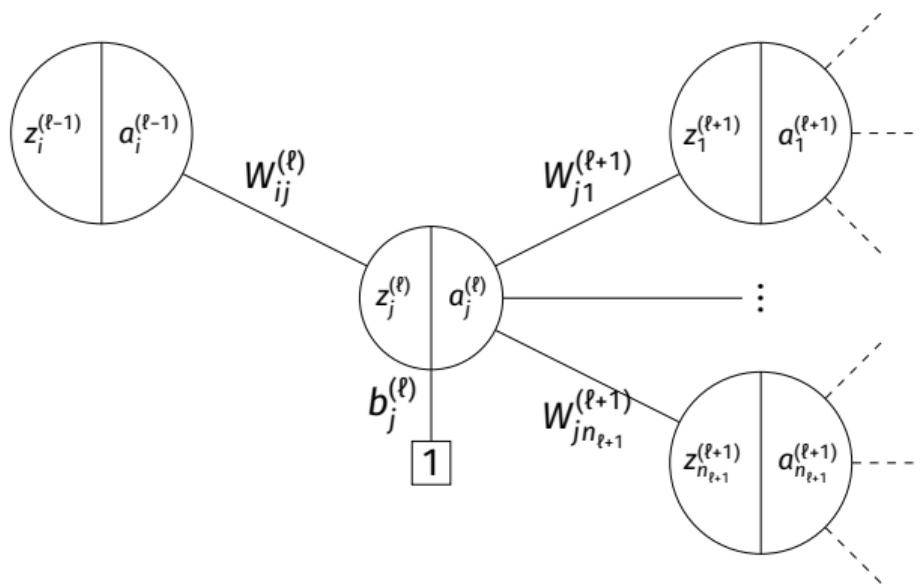


► $\frac{\partial H}{\partial W_{ij}^{(\ell)}} ?$

► $\frac{\partial H}{\partial b_j^{(\ell)}} ?$

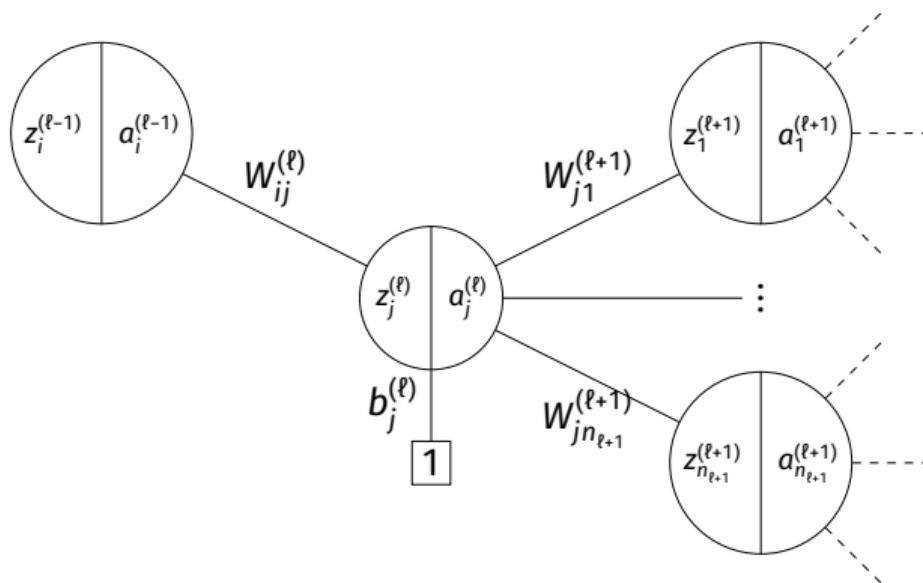
Claim #1

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



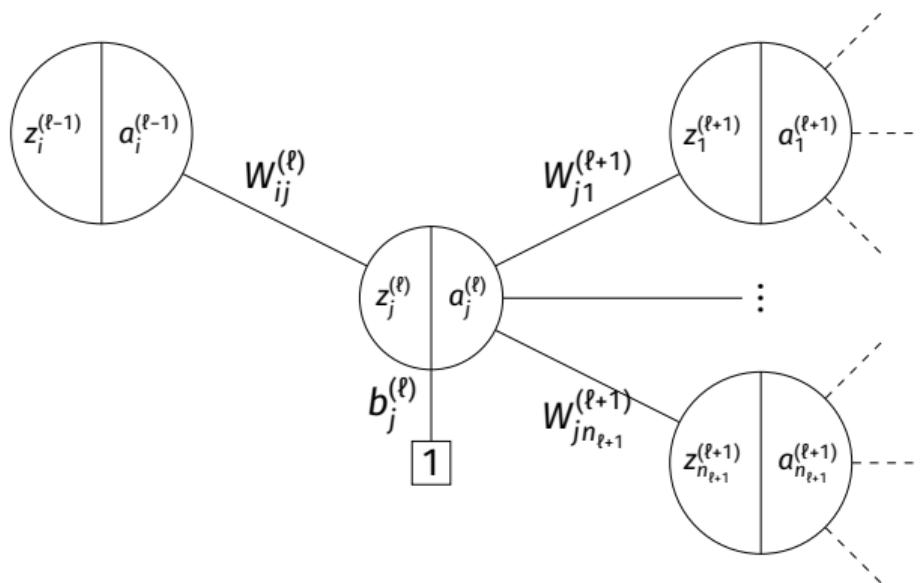
Claim #2

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



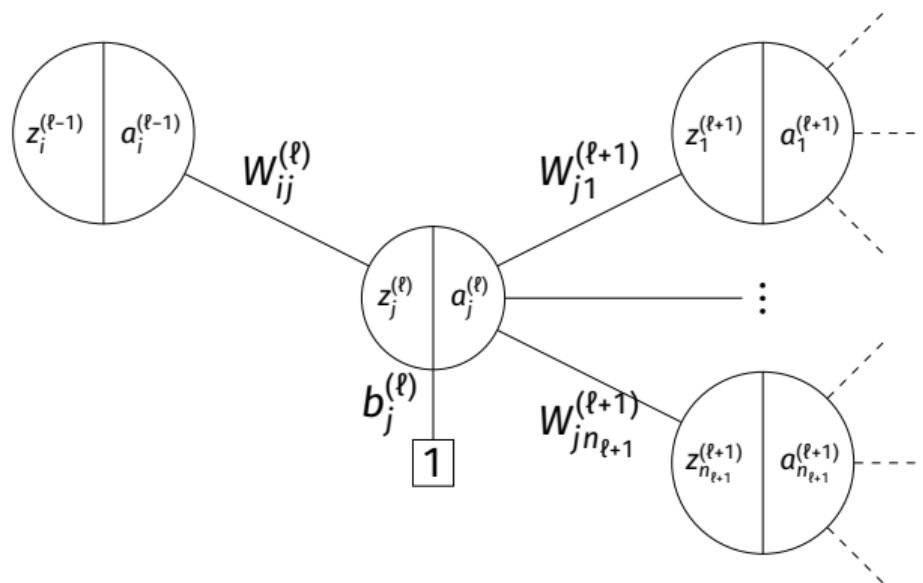
Claim #3

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



Exercise

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



General Formulas

- ▶ For any node in any neural network¹, we have the following recursive formulas:

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

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

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

$$\blacktriangleright \frac{\partial H}{\partial b_j^{(\ell)}} = \frac{\partial H}{\partial z_j^{(\ell)}}$$

¹Fully-connected, feedforward network

Main Idea

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

Backpropagation

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

Backpropagation

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

1. Evaluate the network to compute $z_i^{(\ell)}$ and $a_i^{(\ell)}$ for all nodes.
2. For each layer ℓ from last to first:

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

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

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

d) Compute $\frac{\partial H}{\partial b_j^{(\ell)}} = \frac{\partial H}{\partial z_j^{(\ell)}}$

Example

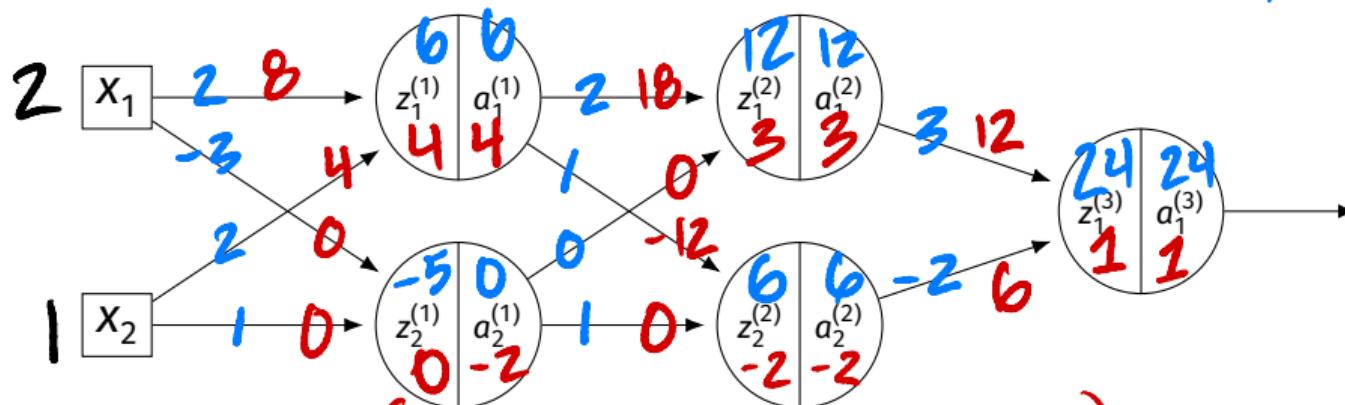
$$H = a_i^{(3)}$$

$g(z) = \text{linear}$

Compute the entries of the gradient given:

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

for hidden



$$\nabla_{\vec{w}} H(\vec{x}; \vec{w}) = (8, 4, 0, 0, 18, 0, -12, \dots)$$

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

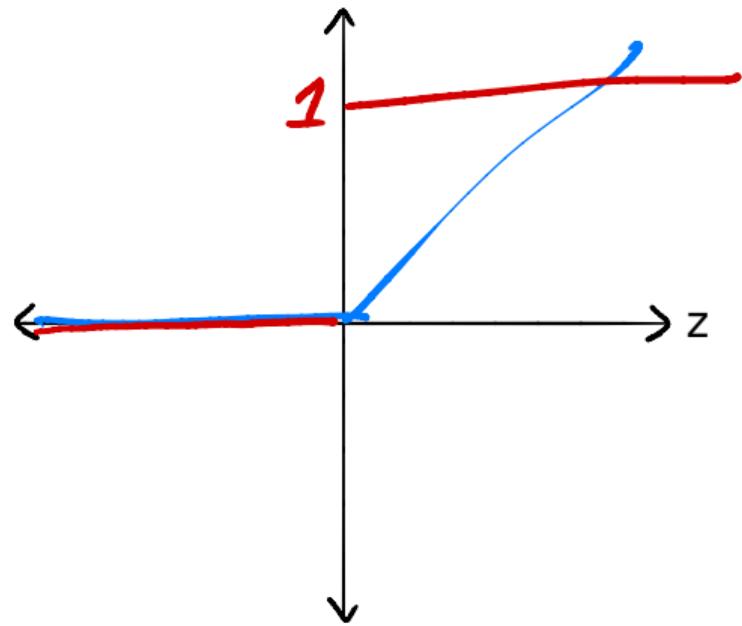
$$\frac{\partial H}{\partial z_j^{(l)}} = \frac{\partial H}{\partial a_j^{(l)}} g'(z_j^{(l)})$$

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

Aside: Derivative of ReLU

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

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



Summary: Backprop

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

DSC 140B

Representation Learning

Lecture 15 | Part 2

Gradient Descent for NN Training

Empirical Risk Minimization

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

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:

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

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

Training Neural Networks

- ▶ For neural networks with nonlinear activations, the risk $R(\vec{w})$ is typically **complicated**.
- ▶ The minimizer cannot be found directly.
- ▶ Instead, we use iterative methods, such as **gradient descent**.

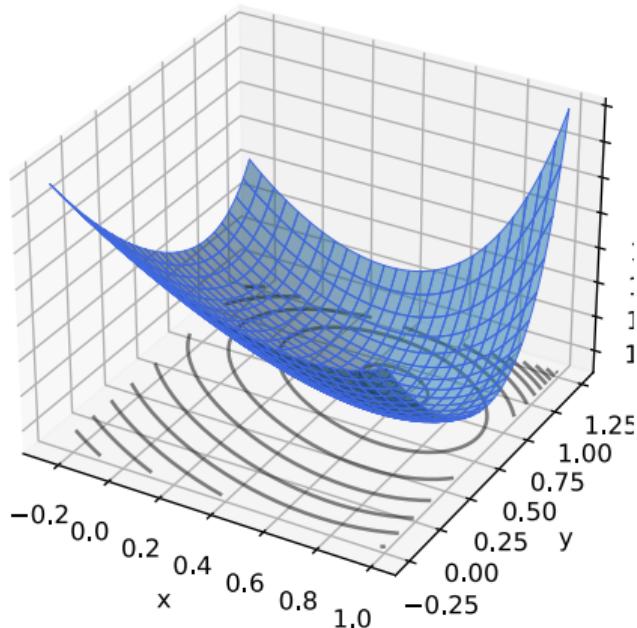
Iterative Optimization

- ▶ To minimize a function $f(\vec{x})$, we may try to compute $\vec{\nabla}f(\vec{x})$; set to 0; solve.
- ▶ Often, there is **no closed-form solution**.
- ▶ How do we minimize f ?

$$\vec{x} = (x, y)^\top$$

Example

- ▶ Consider $f(x, y) = e^{x^2+y^2} + (x - 2)^2 + (y - 3)^2$.



Example

► Try solving $\vec{\nabla}f(x, y) = 0$.

► The gradient is:

$$\vec{\nabla}f(x, y) = \begin{pmatrix} 2xe^{x^2+y^2} + 2(x - 2) \\ 2ye^{x^2+y^2} + 2(y - 3) \end{pmatrix}$$

► Can we solve the system?

$$2xe^{x^2+y^2} + 2(x - 2) = 0$$

$$2ye^{x^2+y^2} + 2(y - 3) = 0$$

Example

- ▶ Try solving $\vec{\nabla}f(x, y) = 0$.

- ▶ The gradient is:

$$\vec{\nabla}f(x, y) = \begin{pmatrix} 2xe^{x^2+y^2} + 2(x - 2) \\ 2ye^{x^2+y^2} + 2(y - 3) \end{pmatrix}$$

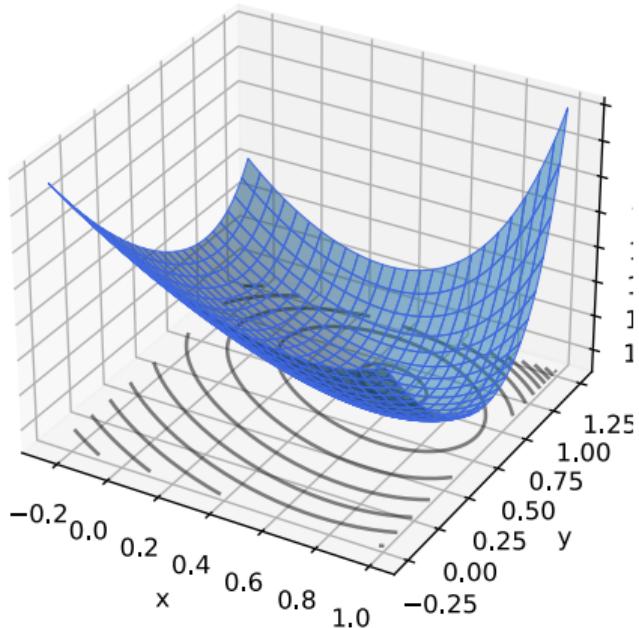
- ▶ Can we solve the system? **Not in closed form.**

$$2xe^{x^2+y^2} + 2(x - 2) = 0$$

$$2ye^{x^2+y^2} + 2(y - 3) = 0$$

Idea

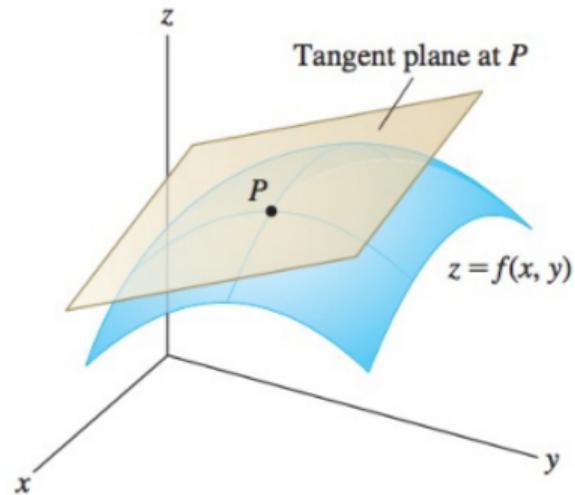
- ▶ Apply an iterative approach.
- ▶ Start at an arbitrary location.
- ▶ “Walk downhill”, towards minimum.



Which way is down?

- ▶ Consider a differentiable function $f(x, y)$.
- ▶ We are standing at $P = (x_0, y_0)$.
- ▶ In a small region around P , f looks like a plane.
- ▶ Slope of plane in x, y directions:

$$\frac{\partial f}{\partial x}(x_0, y_0) \quad \frac{\partial f}{\partial y}(x_0, y_0)$$



The Gradient

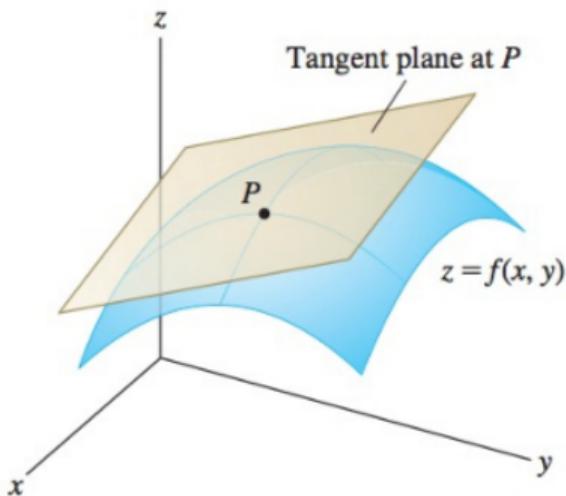
- ▶ Let $f : \mathbb{R}^d \rightarrow \mathbb{R}$ be differentiable. The **gradient** of f at \vec{x} is defined:

$$\vec{\nabla}f(\vec{x}) = \left(\frac{\partial f}{\partial x_1}(\vec{x}), \frac{\partial f}{\partial x_2}(\vec{x}), \dots, \frac{\partial f}{\partial x_d}(\vec{x}) \right)^T$$

- ▶ **Note:** $\vec{\nabla}f(\vec{x})$ is a **function** mapping $\mathbb{R}^d \rightarrow \mathbb{R}^d$.

Which way is down?

- ▶ $\vec{\nabla}f(x_0, y_0)$ points in direction of steepest **ascent** at (x_0, y_0) .
- ▶ $-\vec{\nabla}f(x_0, y_0)$ points in direction of steepest **descent** at (x_0, y_0) .



Gradient Properties

- ▶ The gradient is used in the linear approximation of f :

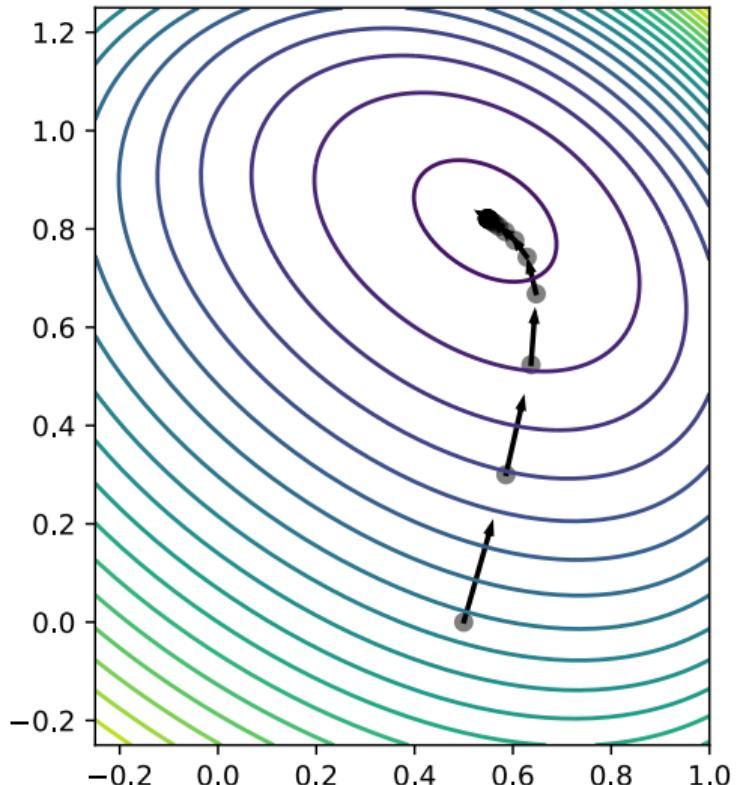
$$f(x_0 + \delta_x, y_0 + \delta_y) \approx f(x_0, y_0) + \vec{\delta} \cdot \vec{\nabla}f(x_0, y_0)$$

- ▶ Important properties:
 - ▶ $\vec{\nabla}f(\vec{x})$ points in direction of **steepest ascent** at \vec{x} .
 - ▶ $-\vec{\nabla}f(\vec{x})$ points in direction of **steepest descent** at \vec{x} .
 - ▶ In directions orthogonal to $\vec{\nabla}f(\vec{x})$, f does not change!
 - ▶ $\|\vec{\nabla}f(\vec{x})\|$ measures steepness of ascent

Gradient Descent

- ▶ Pick arbitrary starting point $\vec{x}^{(0)}$, learning rate parameter $\eta > 0$.
- ▶ Until convergence, repeat:
 - ▶ Compute gradient of f at $\vec{x}^{(i)}$; that is, compute $\vec{\nabla}f(\vec{x}^{(i)})$.
 - ▶ Update $\vec{x}^{(i+1)} = \vec{x}^{(i)} - \eta \vec{\nabla}f(\vec{x}^{(i)})$.
- ▶ When do we stop?
 - ▶ When difference between $\vec{x}^{(i)}$ and $\vec{x}^{(i+1)}$ is negligible.
 - ▶ I.e., when $\|\vec{x}^{(i)} - \vec{x}^{(i+1)}\|$ is small.

```
def gradient_descent(  
    gradient, x, learning_rate=.01,  
    threshold=.1e-4  
):  
    while True:  
        x_new = x - learning_rate * gradient(x)  
        if np.linalg.norm(x - x_new) < threshold:  
            break  
        x = x_new  
    return x
```



Backprop Revisited

- ▶ The weights of a neural network can be trained using **gradient descent**.
- ▶ This requires the gradient to be calculated repeatedly; this is where **backprop** enters.
- ▶ Sometimes people use “backprop” to mean “backprop + SGD”, but this is not strictly correct.

Backprop Revisited

$$\ell = (H - y)^2$$

- ▶ Consider training a NN using the square loss:

$$\begin{aligned}\nabla_{\vec{w}} R(\vec{w}) &= \frac{1}{n} \sum_{i=1}^n \frac{\partial \ell}{\partial H} \nabla_{\vec{w}} H(\vec{x}^{(i)}; \vec{w}) \\ &= \frac{2}{n} \sum_{i=1}^n (H(\vec{x}^{(i)}) - y_i) \nabla_{\vec{w}} H(\vec{x}^{(i)}; \vec{w})\end{aligned}$$

Backprop Revisited

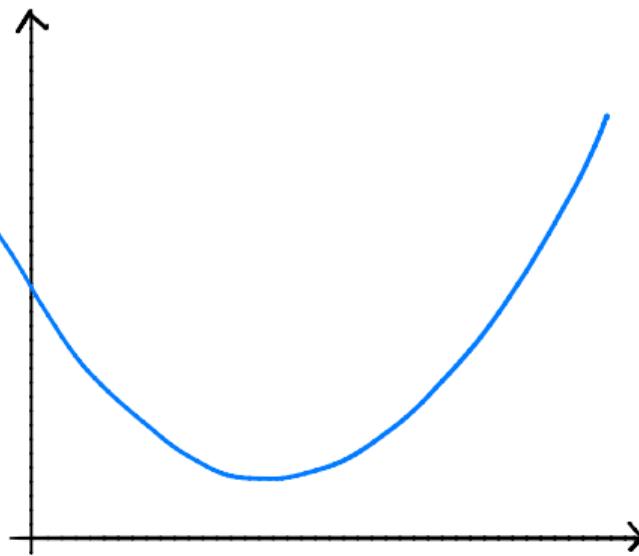
- ▶ Interpretation:

$$\nabla_{\vec{w}} R(\vec{w}) = \frac{2}{n} \sum_{i=1}^n \underbrace{(H(\vec{x}^{(i)}) - y_i)}_{\text{Error}} \underbrace{\nabla_{\vec{w}} H(\vec{x}^{(i)}; \vec{w})}_{\text{Blame}}$$

- ▶ When used in SGD, backprop “propagates error backward” in order to update weights.

Difficulty of Training NNs

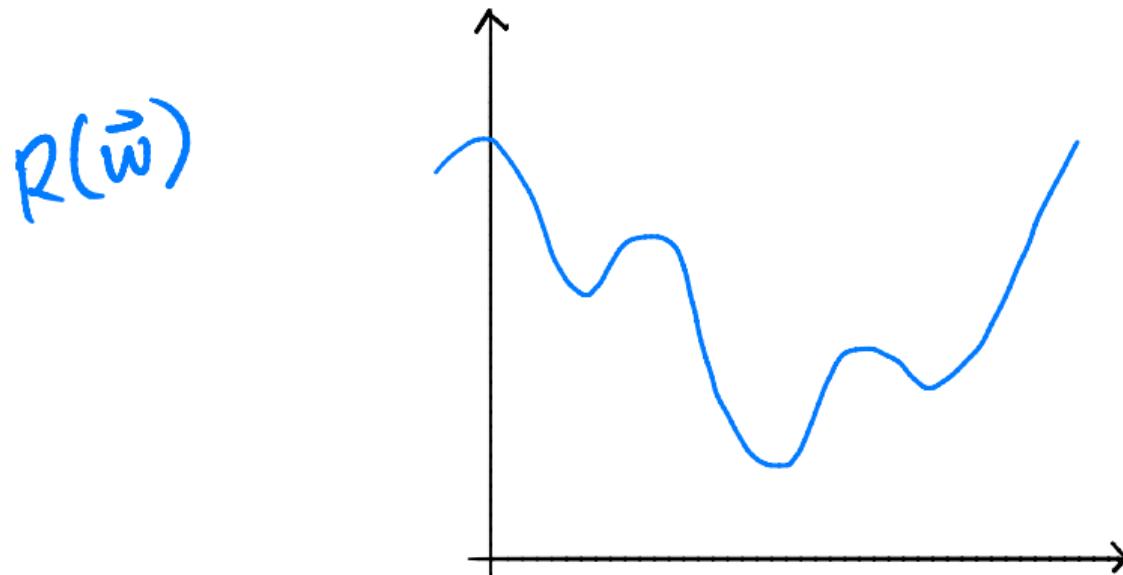
- ▶ Gradient descent is guaranteed to find optimum when objective function is **convex**.²



²Assuming it is properly initialized

Difficulty of Training NNs

- When activations are non-linear, neural network risk is **highly non-convex**:



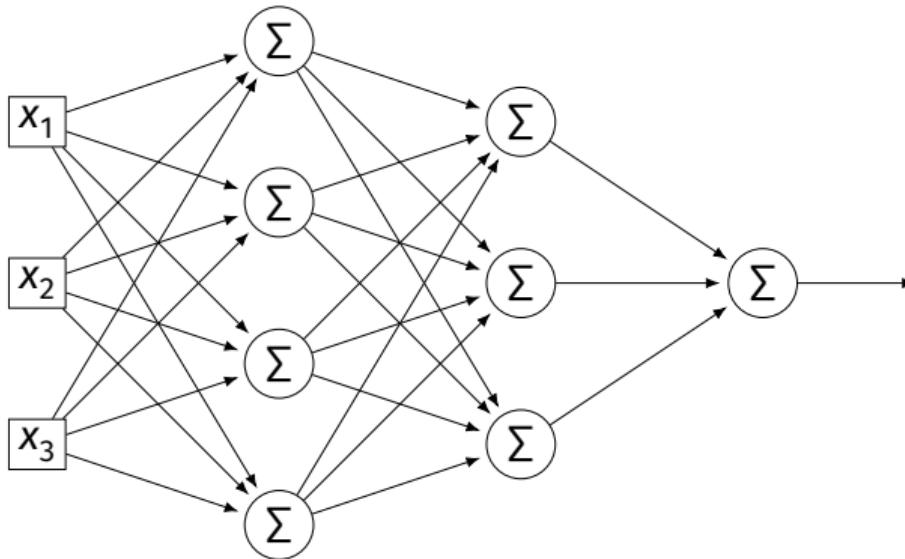
35480
1000

Non-Convexity

- ▶ When R is non-convex, GD can get “stuck” in local minima.
 - ▶ Solution depends on initialization.
- ▶ More sophisticated optimizers, using momentum, adaptation, better initialization, etc.
 - ▶ Adagrad, RMSprop, Adam, etc.

Difficulty of Training (Deep) NNs

- ▶ Deep networks can suffer from the problem of **vanishing gradients**: if w is a weight at the “front” of the network, $\partial H / \partial w$ can be very small

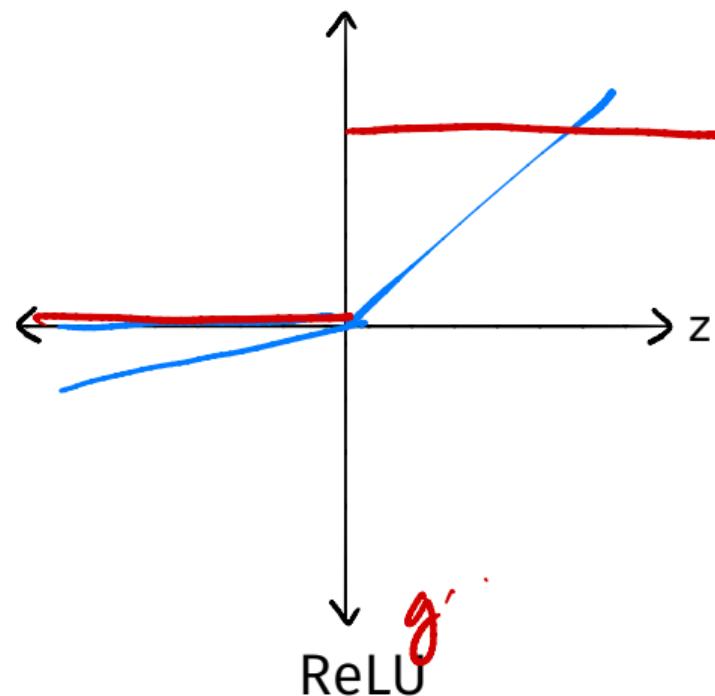
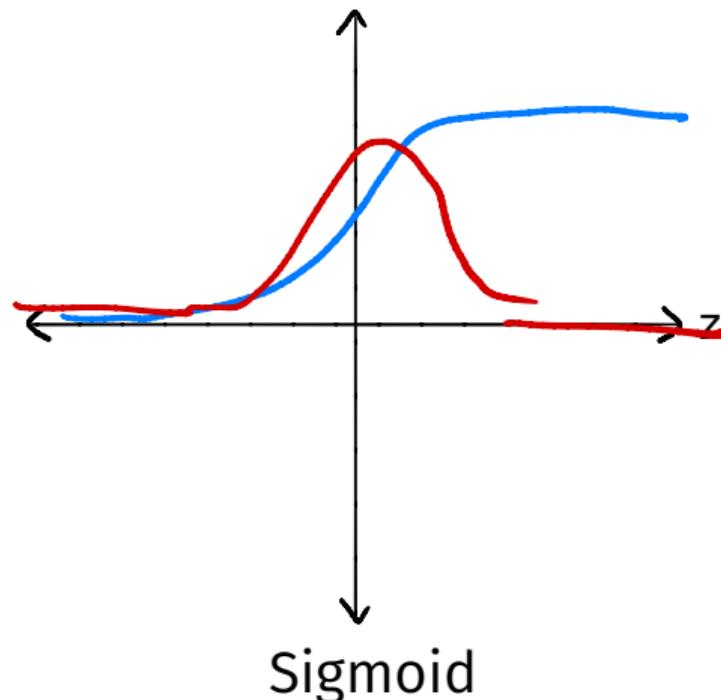


Vanishing Gradients

- ▶ If $\partial H / \partial w$ is always close to zero, w is updated **very slowly** by gradient descent.
- ▶ In short: early layers are slower to train.
- ▶ One mitigation: use ReLU instead of sigmoid.

Vanishing Gradients

$g'(z)$



DSC 140B

Representation Learning

Lecture 15 | Part 3

Stochastic Gradient Descent

Gradient Descent for Minimizing Risk

- ▶ In ML, we often want to minimize a **risk function**:

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

Observation

- ▶ The gradient of the risk function is a sum of gradients:

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

- ▶ One term for each point in training data.

Problem

- ▶ In machine learning, the number of training points n can be **very large**.
- ▶ Computing the gradient can be **expensive** when n is large.
- ▶ Therefore, each step of gradient descent can be **expensive**.

Idea

- ▶ The (full) gradient of the risk uses all of the training data:

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

- ▶ It is an average of n gradients.
- ▶ **Idea:** instead of using all n points, randomly choose $\ll n$.

Stochastic Gradient

- ▶ Choose a random subset (**mini-batch**) B of the training data.
- ▶ Compute a **stochastic gradient**:

$$\nabla R(\vec{w}) \approx \sum_{i \in B} \vec{\nabla} \ell(H(\vec{x}^{(i)}; \vec{w}), y_i)$$

Stochastic Gradient

$$\nabla R(\vec{w}) \approx \sum_{i \in B} \vec{\nabla} \ell(H(\vec{x}^{(i)}; \vec{w}), y_i)$$

- ▶ **Good:** if $|B| \ll n$, this is much faster to compute.
- ▶ **Bad:** it is a (random) approximation of the full gradient, noisy.

Stochastic Gradient Descent (SGD) for ERM

- ▶ Pick arbitrary starting point $\vec{x}^{(0)}$, learning rate parameter $\eta > 0$, batch size $m \ll n$.
- ▶ Until convergence, repeat:
 - ▶ Randomly sample a batch B of m training data points (on each iteration).
 - ▶ Compute stochastic gradient of f at $\vec{x}^{(i)}$:

$$\vec{g} = \sum_{i \in B} \vec{\nabla} \ell(H(\vec{x}^{(i)}; \vec{w}), y_i)$$

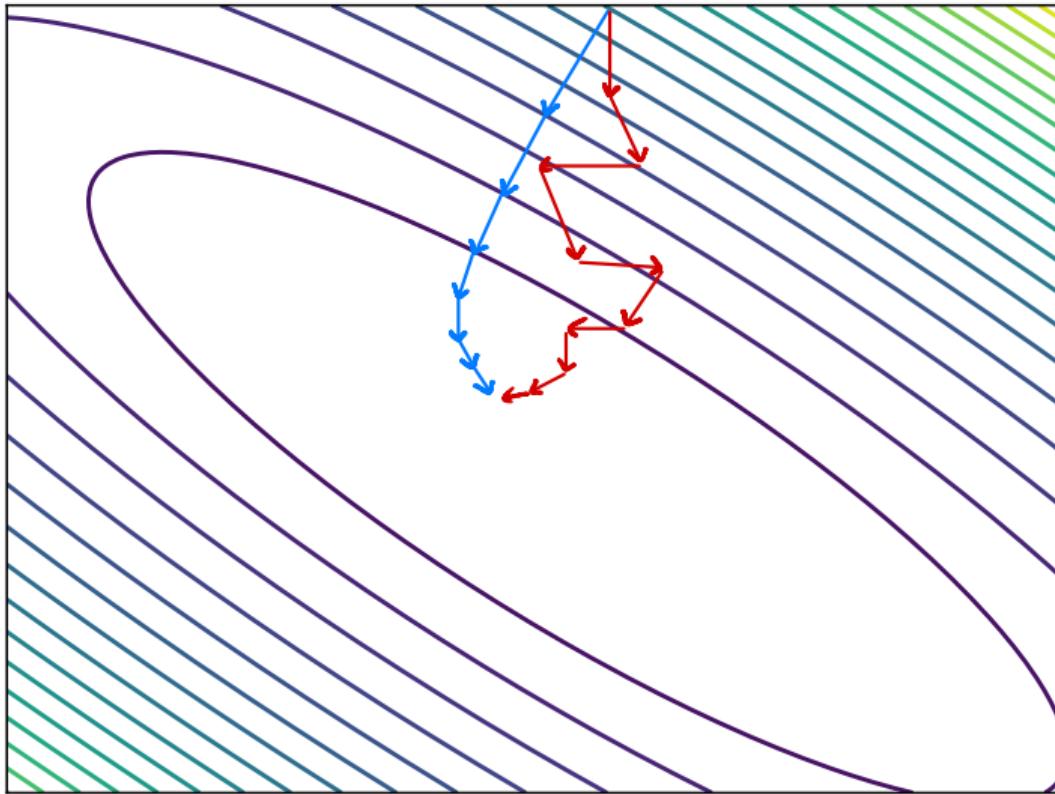
- ▶ Update $\vec{x}^{(i+1)} = \vec{x}^{(i)} - \eta \vec{g}$

Idea

- ▶ In practice, a stochastic gradient often works well enough.
- ▶ It is better to take many noisy steps quickly than few exact steps slowly.

Batch Size

- ▶ Batch size m is a parameter of the algorithm.
- ▶ The larger m , the more reliable the stochastic gradient, but the more time it takes to compute.
- ▶ Extreme case when $m = 1$ will still work.



Usefulness of SGD

- ▶ SGD allows learning on **massive** data sets.
- ▶ Useful even when exact solutions available.
 - ▶ E.g., least squares regression / classification.

Training NNs in Practice

- ▶ There are several Python packages for training NNs:
 - ▶ PyTorch
 - ▶ Tensorflow / Keras
- ▶ This week's discussion was a Tensorflow tutorial.

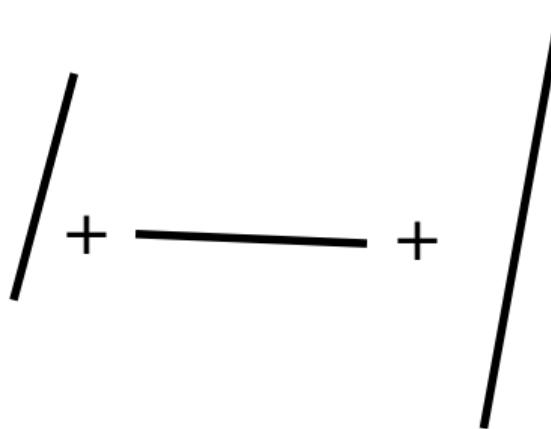
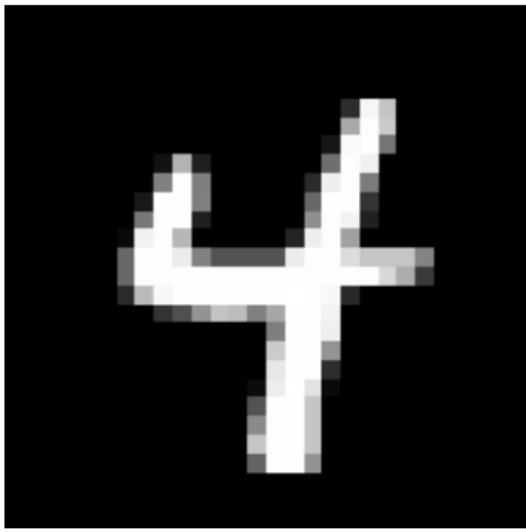
DSC 140B

Representation Learning

Lecture 16 | Part 1

Convolutions

$$H(x) = w_1x_1 + w_2x_2 + \dots + w_dx_d + w_0$$

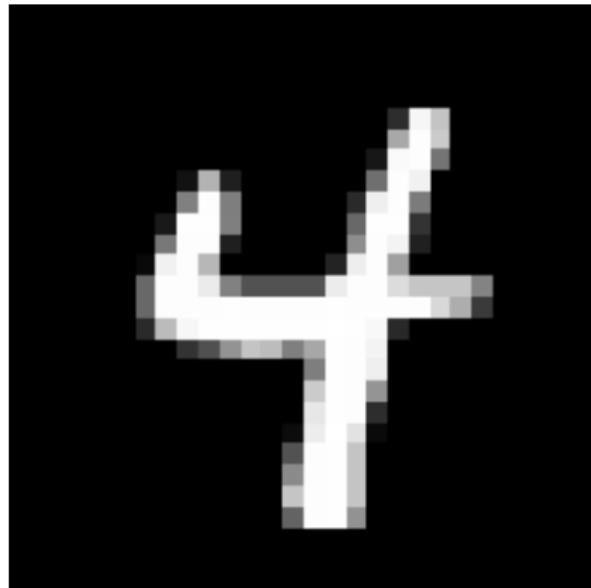


From Simple to Complex

- ▶ Complex shapes are made of simple patterns
- ▶ The human visual system uses this fact
- ▶ Line detector → shape detector → ... → face detector
- ▶ Can we replicate this with a deep NN?

Edge Detector

- ▶ How do we find **vertical edges** in an image?
- ▶ One solution: **convolution** with an **edge filter**.



Vertical Edge Filter

0	-1	1
0	-1	1
0	-1	1

Idea

$$\begin{aligned}0 + (-10) + 60 + \\0 + (-10) + 60 + \\0 + (-10) + 60\end{aligned}$$

$$= 180 - 30 \\= \boxed{150}$$

image patch:

0	10	60
0	10	60
0	10	60

- ▶ Take a patch of the image, same size as filter.
- ▶ Perform “dot product” between patch and filter.
- ▶ If large, this is a (vertical) edge.

filter:

0	-1	/
0	-1	/
0	-1	/

Idea

$$\begin{aligned} & 0 + 0 + 0 + \\ & 0 + (-1) + 10 + = 0 \\ & 0 - 60 + 60 \end{aligned}$$

image patch:

0	0	0
10	10	10
60	60	60

- ▶ Take a patch of the image, same size as filter.
- ▶ Perform “dot product” between patch and filter.
- ▶ If large, this is a (vertical) edge.

filter:

0	-1	1
0	-1	1
0	-1	1

Idea

$$0+0+0+ \\ 0+0+.9+ \\ 0+0+.9 = 1.8$$

- ▶ Move the filter over the entire image, repeat procedure.

Idea

- ▶ Move the filter over the entire image, repeat procedure.

$$\begin{matrix} 0 & 0 & 0 & 0 & 0 & 0 & \dots \\ 0 & 0 & .9 & 0 & 0 & 0 & .7 \\ 0 & 0 & .9 & 0 & 0 & 0 & .8 \\ 0 & 0 & .8 & 0 & 0 & 0 & .9 \\ 0 & 0 & .7 & 0 & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \end{matrix} * \begin{matrix} 0 & -1 & 1 \\ 0 & -1 & 1 \\ 0 & -1 & 1 \end{matrix} = \begin{matrix} 1.8 & -1.8 & \dots \\ \vdots & \vdots & \vdots \end{matrix}$$

Idea

- ▶ Move the filter over the entire image, repeat procedure.

Idea

- ▶ Move the filter over the entire image, repeat procedure.

$$\begin{matrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & .9 & 0 & 0 & .7 \\ 0 & 0 & .9 & 0 & 0 & .8 \\ 0 & 0 & .8 & 0 & 0 & .9 \\ 0 & 0 & .7 & 0 & 0 & 0 \end{matrix} * \begin{matrix} 0 & -1 & 1 \\ 0 & -1 & 1 \\ 0 & -1 & 1 \end{matrix} = \begin{matrix} 1.8 & -1.8 & 0 \\ 2.6 & \end{matrix}$$

Idea

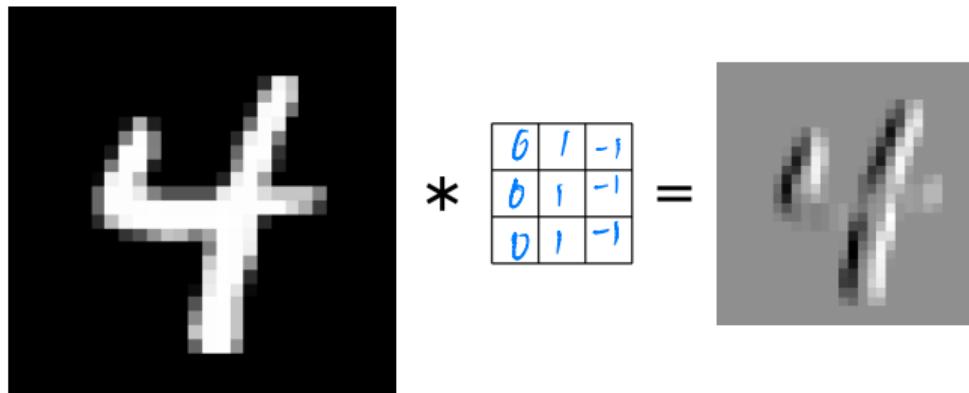
- ▶ Move the filter over the entire image, repeat procedure.

$$\begin{matrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & .9 & 0 & 0 & .7 \\ 0 & 0 & .9 & 0 & 0 & .8 \\ 0 & 0 & .8 & 0 & 0 & .9 \\ 0 & 0 & .7 & 0 & 0 & 0 \end{matrix} * \begin{matrix} \boxed{} & \boxed{} & \boxed{} \\ \boxed{} & \boxed{} & \boxed{} \\ \boxed{} & \boxed{} & \boxed{} \end{matrix} = \begin{matrix} \boxed{} & \boxed{} & \boxed{} & \boxed{} & \boxed{} \\ \boxed{} & \boxed{} & \boxed{} & \boxed{} & \boxed{} \\ \boxed{} & \boxed{} & \boxed{} & \boxed{} & \boxed{} \\ \boxed{} & \boxed{} & \boxed{} & \boxed{} & \boxed{} \\ \boxed{} & \boxed{} & \boxed{} & \boxed{} & \boxed{} \end{matrix}$$

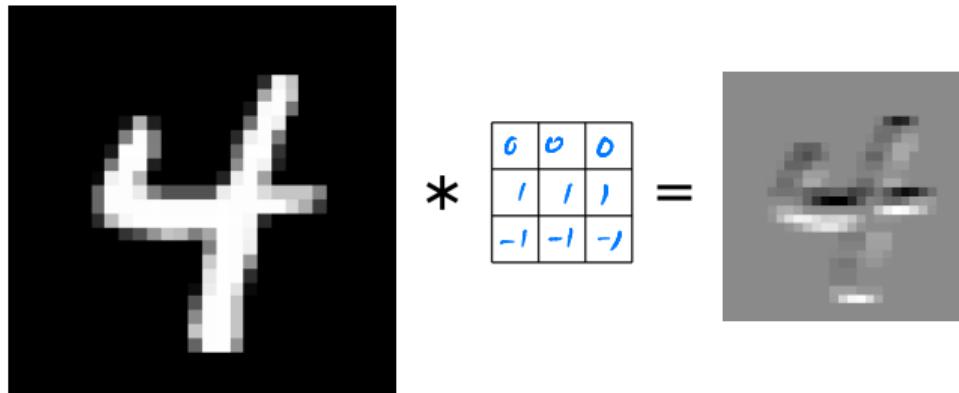
Convolution

- ▶ The result is the (2d) **convolution** of the filter with the image.
- ▶ Output is also 2-dimensional array.
- ▶ Called a **response map**.

Example: Vertical Filter



Example: Horizontal Filter



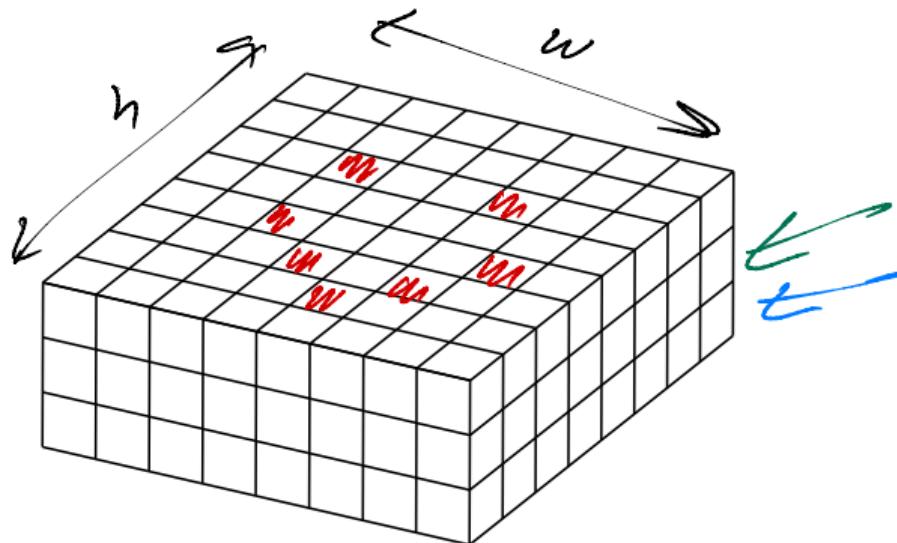
More About Filters

- ▶ Typically 3×3 or 5×5.
- ▶ Variations: different **stride**, image **padding**.

3-d Filters

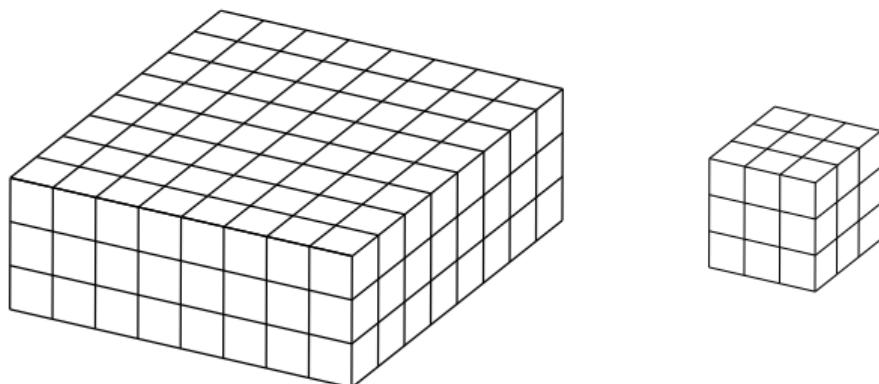
- ▶ Black and white images are 2-d arrays.
- ▶ But color images are 3-d arrays:
 - ▶ a.k.a., **tensors**
 - ▶ Three color **channels**: red, green, blue.
 - ▶ $\text{height} \times \text{width} \times 3$
- ▶ How does convolution work here?

Color Image

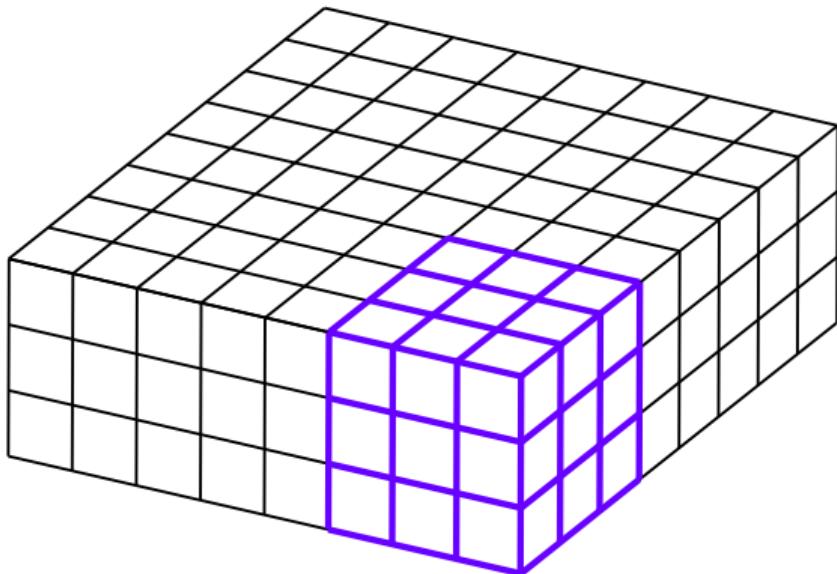


3-d Filter

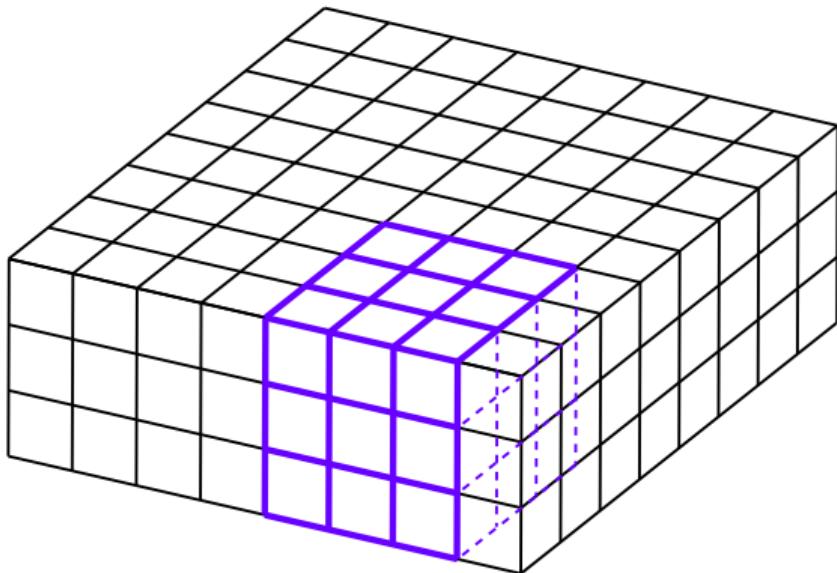
- ▶ The filter must also have three channels:
 - ▶ $3 \times 3 \times 3$, $5 \times 5 \times 3$, etc.



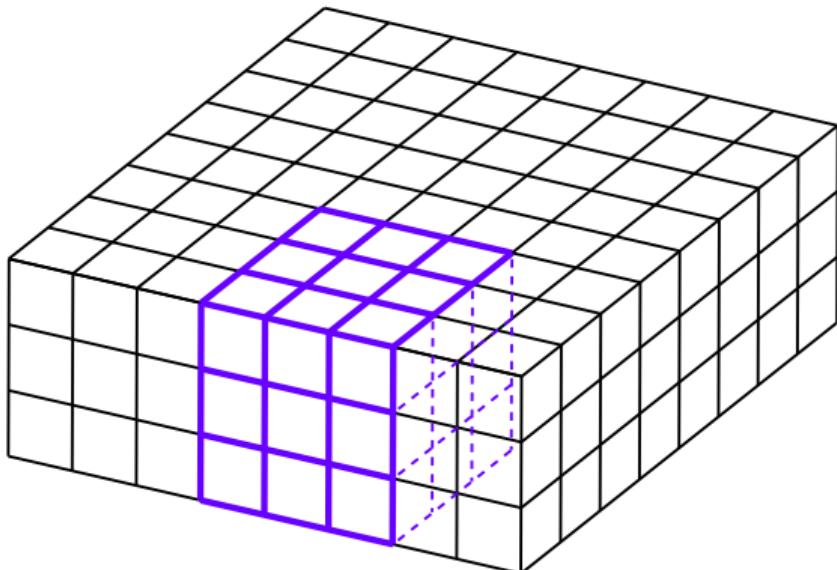
3-d Filter



3-d Filter



3-d Filter

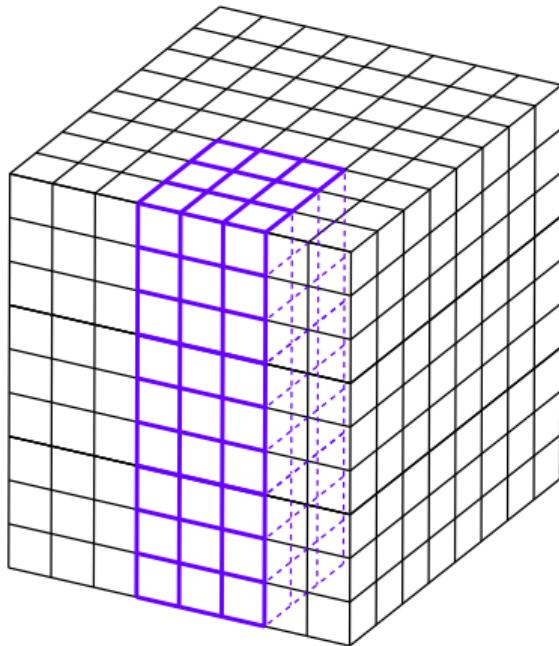


Convolution with 3-d Filter

- ▶ Filter must have same number of channels as image.
 - ▶ 3 channels if image RGB.
- ▶ Result is still a 2-d array.

General Case

- ▶ Input “image” has k channels.
- ▶ Filter must have k channels as well.
 - ▶ e.g., $3 \times 3 \times k$
- ▶ Output is still $2 - d$



DSC 140B

Representation Learning

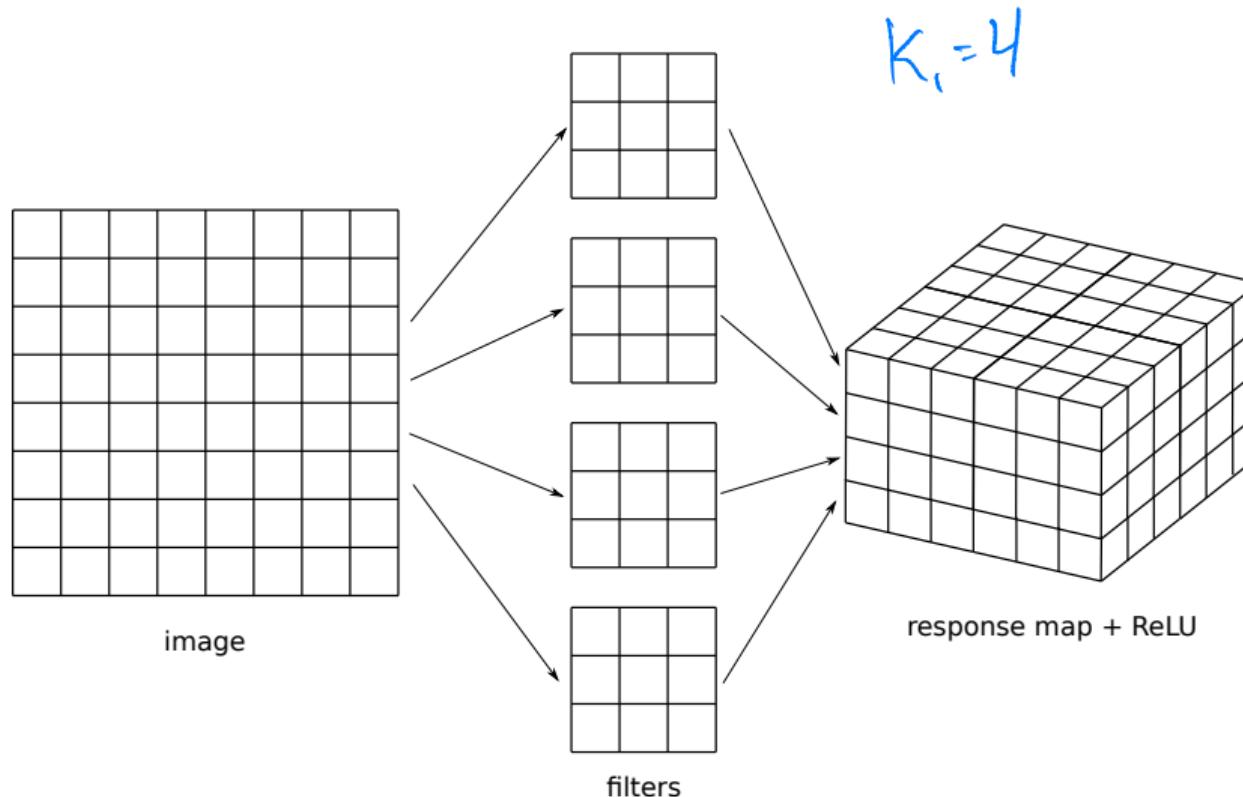
Lecture 16 | Part 2

Convolutional Neural Networks

Convolutional Neural Networks

- ▶ **CNNs** are the state-of-the-art for many computer vision tasks
- ▶ **Idea:** use convolution in early layers to create new feature representation.
- ▶ **But!** Filters are **learned**.

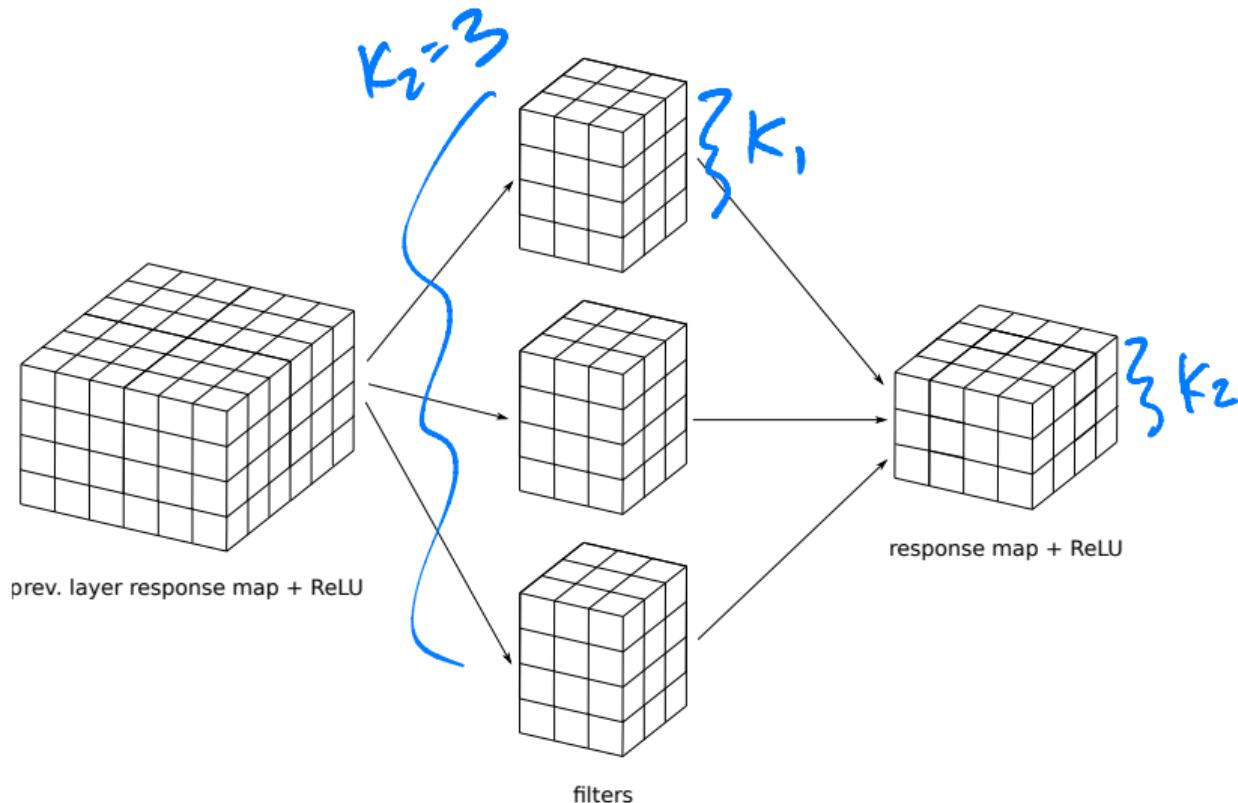
Input Convolutional Layer



Input Convolutional Layer

- ▶ Input image with one channel (grayscale)
- ▶ k_1 filters of size $\ell \times \ell \times 1$
- ▶ Results in k_1 convolutions, stacked to make response map.
- ▶ ReLU (or other nonlinearity) applied entrywise.

Second Convolutional Layer



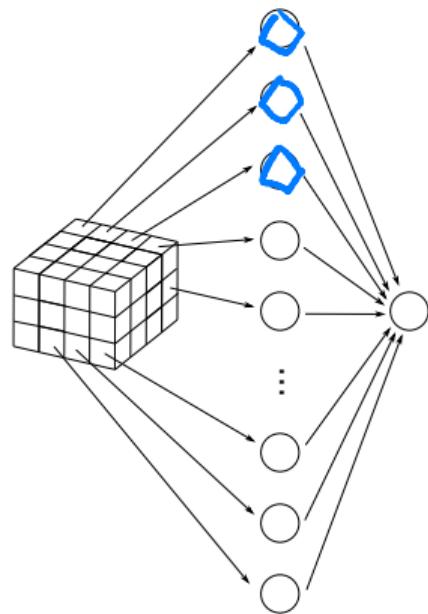
Second Convolutional Layer

- ▶ Input is a 3-d **tensor**.
 - ▶ “Stack” of k_1 response maps.
- ▶ k_2 filters, each a 3-d tensor with k_1 channels.
- ▶ Output is a 3-d tensor with k_2 channels.

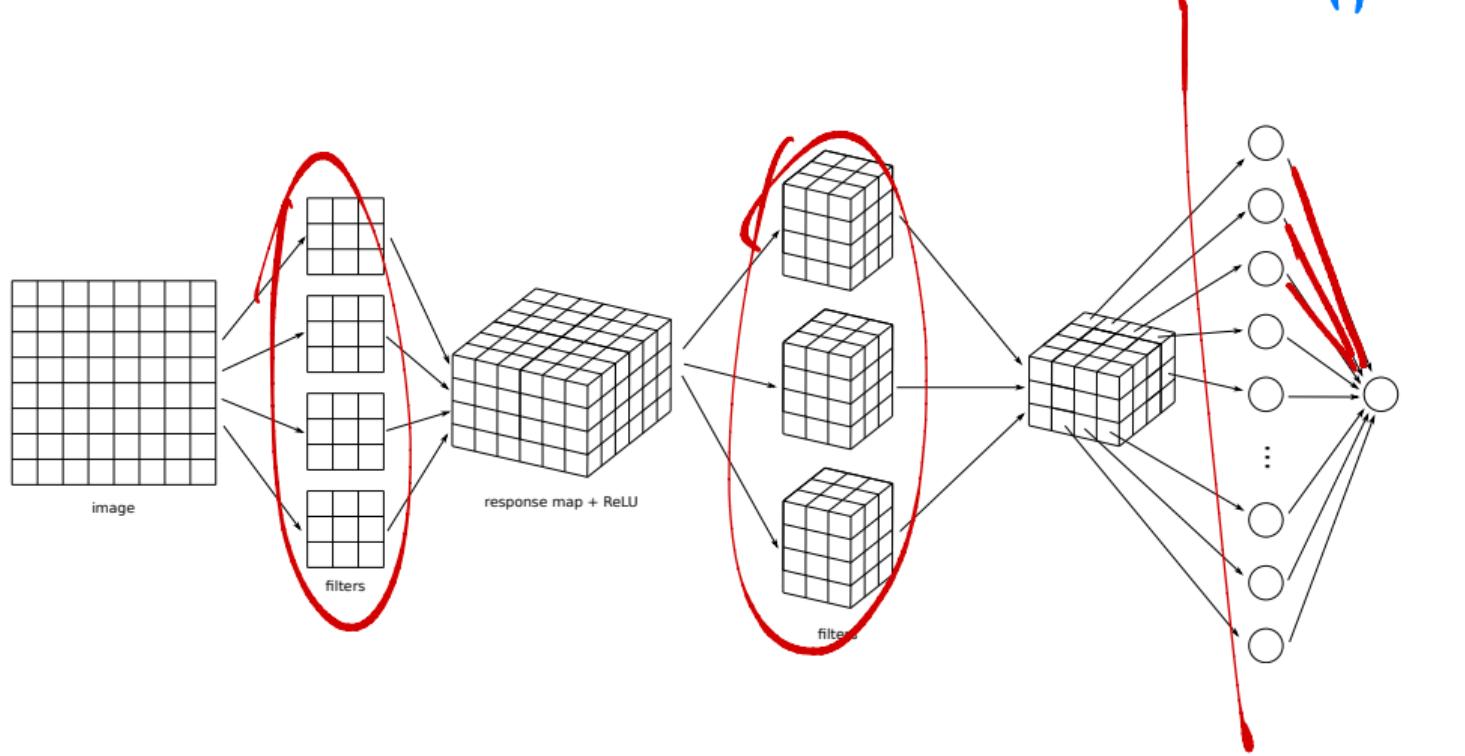
More Convolutional Layers

- ▶ May add more convolutional layers.
- ▶ Last convolutional layer used as input to a feedforward, fully-connected network.
- ▶ Need to “flatten” the output tensor.

Flattening



Full Network



What is learned?

- ▶ The filters themselves.
- ▶ The weights in the feedforward NN used for prediction.

Max Pooling

- Max pooling is an important part of convolutional layers in practice.
 - Reduces size of response map, number of parameters.

20	30	10	5	
10	15	20	30	
5	2	10	4	

DSC 140B

Representation Learning

Lecture 16 | Part 3

Example: Image Classification

Problem

- ▶ Predict whether image is of a **car** or a **truck**.



Problem

- ▶ Predict whether image is of a **car** or a **truck**.



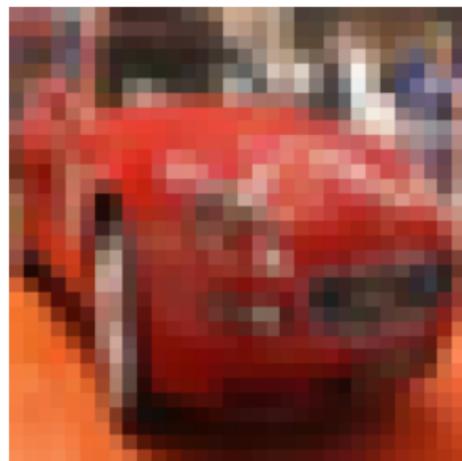
Problem

- ▶ Predict whether image is of a **car** or a **truck**.



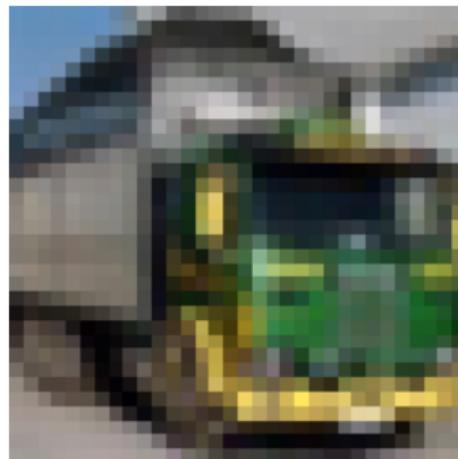
Problem

- ▶ Predict whether image is of a **car** or a **truck**.



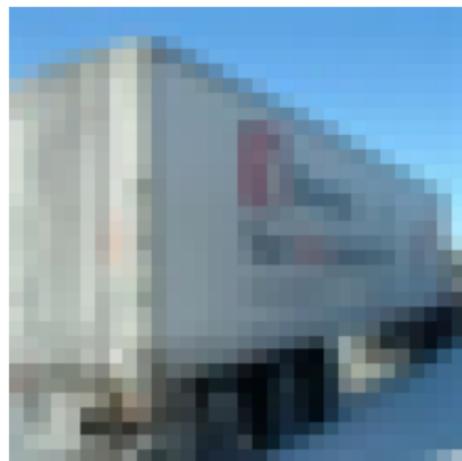
Problem

- ▶ Predict whether image is of a **car** or a **truck**.



Problem

- ▶ Predict whether image is of a **car** or a **truck**.



Problem

- ▶ Predict whether image is of a **car** or a **truck**.



Problem

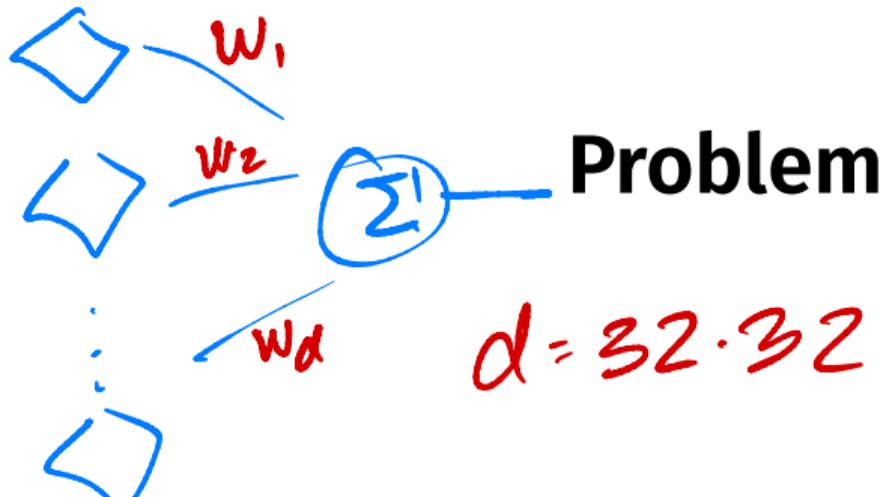
- ▶ Predict whether image is of a **car** or a **truck**.



Problem

- ▶ Predict whether image is of a **car** or a **truck**.





- ▶ Predict whether image is of a **car** or a **truck**.



Details

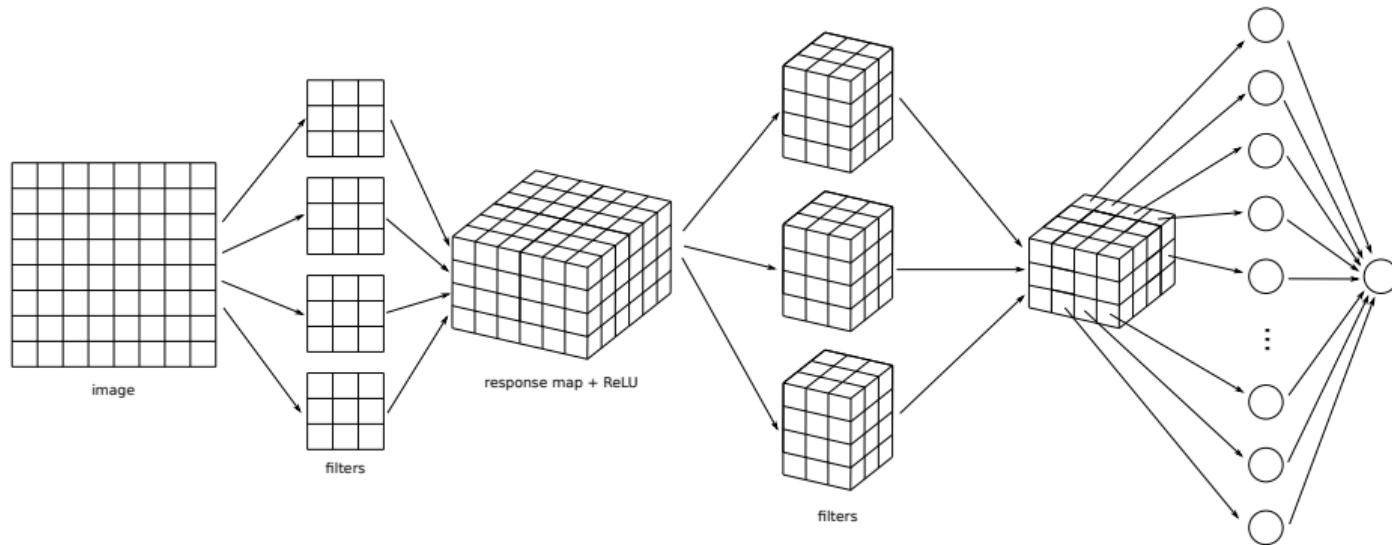
- ▶ 3-channel 32×32 color images
- ▶ 10,000 training images; 2,000 test¹
- ▶ Cars, trucks in different orientations, scales
- ▶ Balanced: 50% cars, 50% trucks

¹CIFAR-10

Approach #1: Least Squares Classifier

- ▶ Train directly on raw features (grayscale)
- ▶ Result: 72% train accuracy, 63% test accuracy
- ▶ Need a better feature representation

Approach #2: Convolutional Neural Network



Architecture

- ▶ 3 convolutional layers with 32, 64, 64 filters
- ▶ ReLU, max pooling after first two
- ▶ Dense layer with 64 hidden neurons, ReLU
- ▶ Output layer with sigmoid activation
- ▶ Minimize cross-entropy loss; use *dropout*

The Code

```
model = keras.models.Sequential()

model.add( keras.layers.Conv2D(32, (7, 7), activation='relu', input_shape=(32, 32, 1)))
model.add(keras.layers.MaxPooling2D((2, 2)))

model.add(keras.layers.Conv2D(64, (5, 5), activation='relu'))
model.add(keras.layers.MaxPooling2D((2, 2)))

model.add(keras.layers.Conv2D(64, (3, 3), activation='relu'))

model.add(keras.layers.Flatten())
model.add(keras.layers.Dropout(0.5))
model.add(keras.layers.Dense(64, activation='relu'))
model.add(keras.layers.Dense(1, activation='sigmoid'))
```

The Code

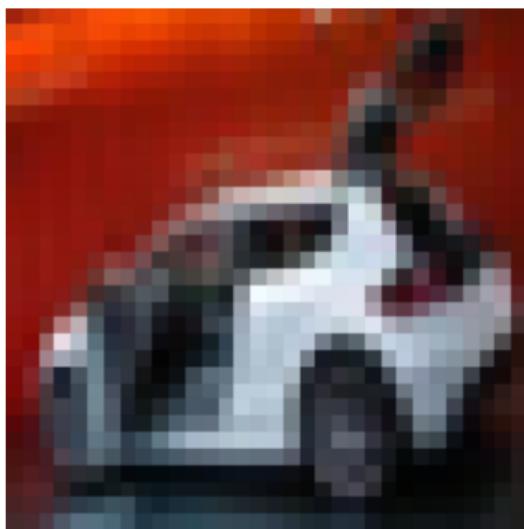
```
model.compile(  
    optimizer=keras.optimizers.RMSprop(),  
    loss=keras.losses.BinaryCrossentropy(),  
    metrics=['accuracy'])  
  
model.fit(  
    X_train,  
    y_train,  
    epochs=30,  
    validation_data=(X_test, y_test))
```

Results

- ▶ 94% train accuracy, 90% test accuracy

Results

car / car



Results

truck / car



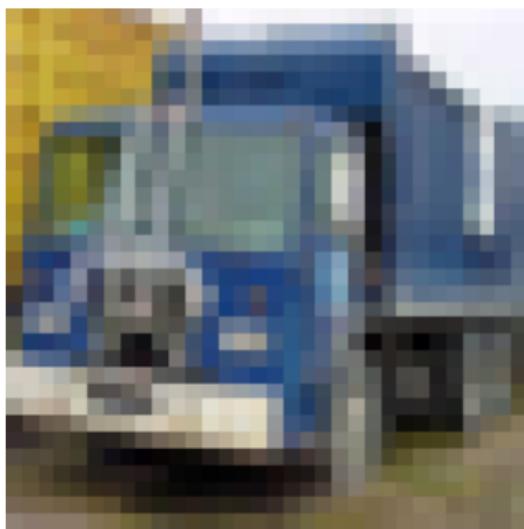
Results

truck / truck



Results

truck / truck



Results

truck / truck



Results

truck / truck



Results

truck / truck



Results

car / car



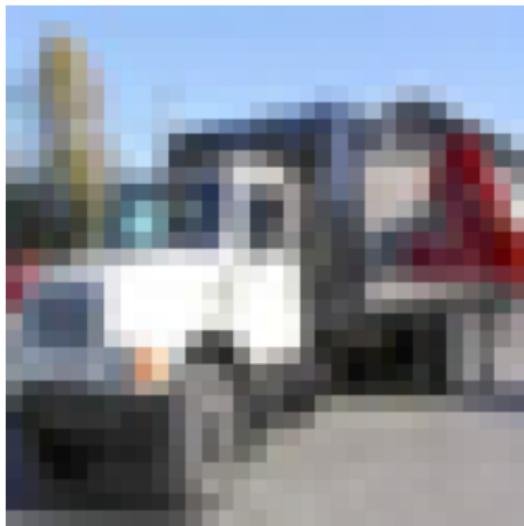
Results

truck / truck



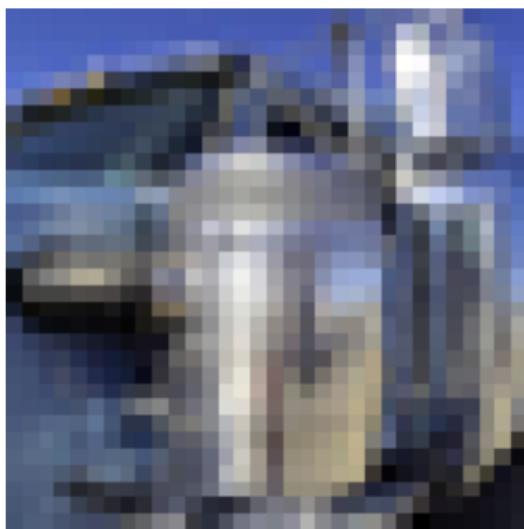
Results

truck / truck



Results

truck / truck



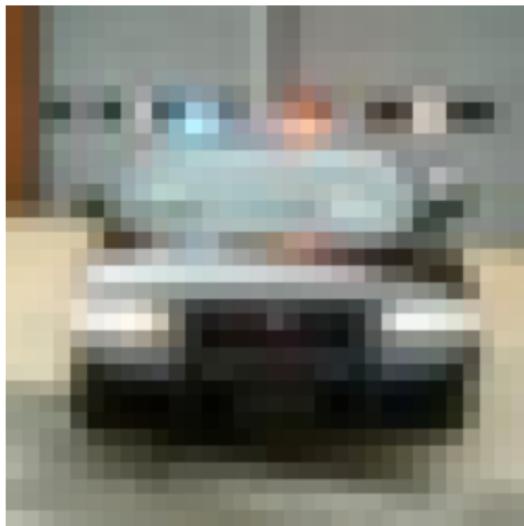
Results

truck / truck



Results

car / car



Results

car / truck



Results

truck / car



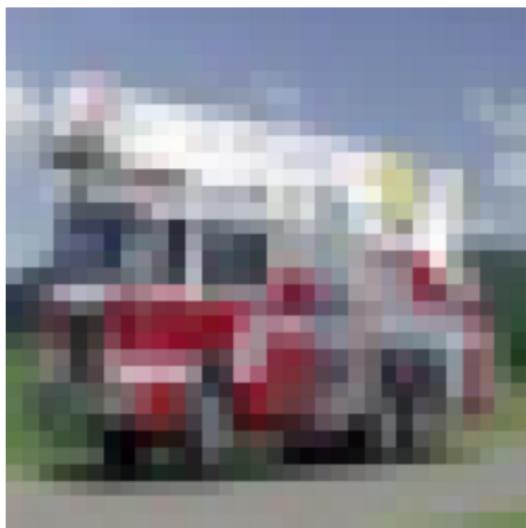
Results

car / car



Results

truck / truck



Results

car / car



Results

car / car



Results

car / car



Filters



Next Steps

- ▶ In practice, you might not train your own CNN
- ▶ Instead, take “pre-trained” convolutional layers from a much bigger network
- ▶ Attach untrained fully-connected layer and train
- ▶ This is **transfer learning**

DSC 140B

Representation Learning

Lecture 17 | Part 1

Autoencoders

Generalizing PCA

- ▶ We started the quarter with PCA.
- ▶ PCA is a **linear** method.
- ▶ We can generalize upon PCA to derive nonlinear representation learners.

Representation Learning

- ▶ At a high level, representation learning finds an **encoding function** $\text{encode}(\vec{x}) : \mathbb{R}^d \rightarrow \mathbb{R}^k$.
- ▶ Ideally, this function captures useful aspects of the data distribution.

Example: PCA

- ▶ In PCA, we **encode** a point \vec{x} by projecting it onto the top k eigenvectors of data covariance matrix:

$$\text{encode}(\vec{x}) = U^T \vec{x}$$

Decoding

- ▶ Encoding can decrease dimensionality.
- ▶ Intuitively, we may want to preserve as much “information” about \vec{x} as possible.
- ▶ We should be able to **decode** the encoding and **reconstruct** the original point, approximately.

$$\vec{x} \approx \text{decode}(\text{encode}(\vec{x}))$$


reconstruction

Example: PCA

- ▶ In PCA, given a point $\vec{z} \in \mathbb{R}^k$ in the new representation, the **reconstruction** is:

$$\text{decode}(\vec{z}) = U\vec{z}$$

Representation Learning

- ▶ **Goal:** find an encoder (and decoder) such that

$$\text{encode}(\overset{\text{de}}{\cancel{\text{en}}} \overset{\text{en}}{\cancel{\text{de}}}(\vec{x})) \approx \vec{x}$$

$$\text{encoder}(\vec{x}) = \vec{x}$$

Reconstruction Error

- ▶ In general, $\text{decode}(\text{encode}(\vec{x}))$ will not be exactly equal to \vec{x} .
- ▶ One way of quantifying the difference w.r.t. data is the (ℓ_2) **reconstruction error**:

$$\sum_{i=1}^n \|\vec{x}^{(i)} - \text{decode}(\text{encode}(\vec{x}^{(i)}))\|^2$$

Note



- ▶ Of course, it is trivial to find an encoder/decoder with zero reconstruction error:

$$\text{encode}(\vec{x}) = \vec{x} = \text{decode}(\vec{x})$$

- ▶ Such an encoder is not useful.
- ▶ Instead, we constrain the form of the encoder so that it cannot simply copy the input.

Example: PCA

- ▶ Assume $\text{encode}(\vec{x}) = U\vec{x}$, for some matrix U whose $k \leq d$ columns are orthonormal.
 - ▶ That is, the encoding is an orthogonal projection.
- ▶ **Goal:** find U to minimize reconstruction error on a dataset $\vec{x}^{(1)}, \dots, \vec{x}^{(d)}$.
- ▶ **Solution:** pick columns of U to be top k eigenvectors of data covariance matrix.

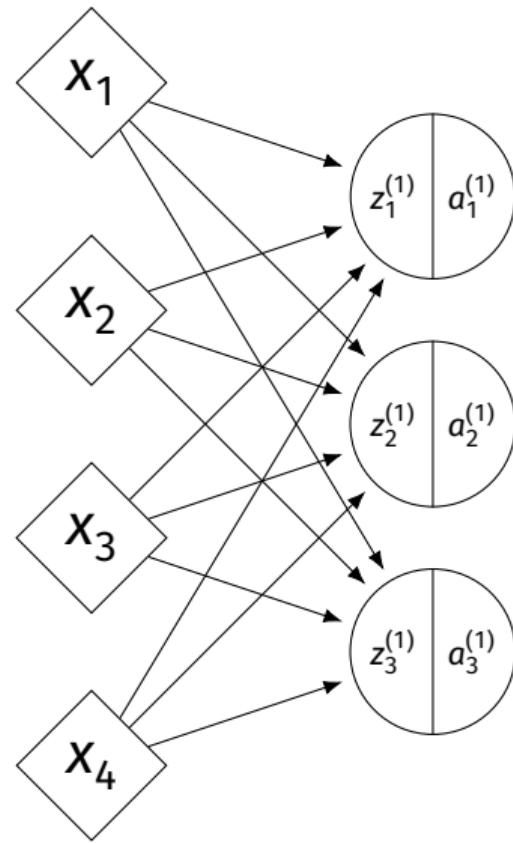
Now

- ▶ $\text{encode}(\vec{x}) = U\vec{x}$ is a linear encoding function.
- ▶ What if we let encode be nonlinear?
- ▶ That is, let's generalize PCA.

Encoder as a Neural Network

- ▶ Assume $\text{encode}(\vec{x})$ is a (deep) **neural network**.
- ▶ Output is not a single number, but k numbers.
 - ▶ I.e., a vector in \mathbb{R}^k
- ▶ Can use nonlinear activations, have more than one layer.

Encoder as a Neural Network



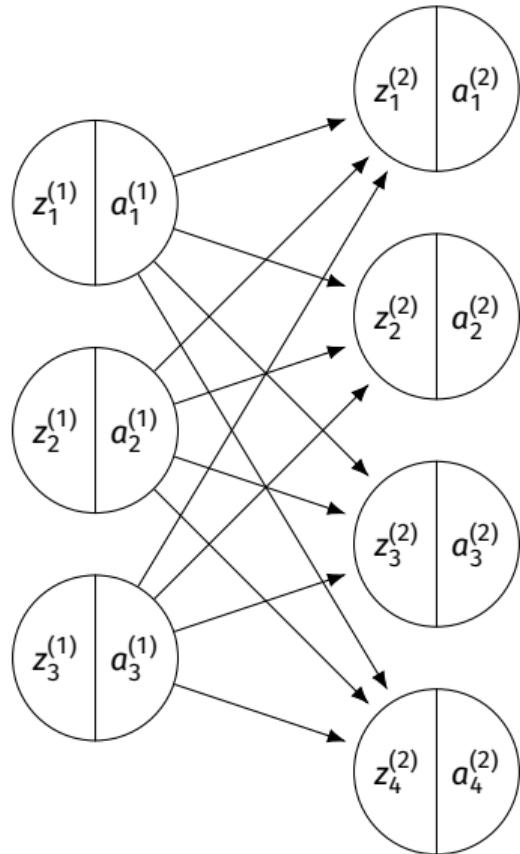
Encoder as a Neural Network

- ▶ The output of the encoder is the new representation.
- ▶ To train the encoder, we'll need a **decoder**.

Decoder as a Neural Network

- ▶ Assume $\text{decode}(\vec{z})$ is a (deep) **neural network**.
- ▶ Output is not a single number, but d numbers.
 - ▶ Same dimensionality as original input, \vec{x} .
 - ▶ I.e., a vector in \mathbb{R}^d
- ▶ Can use nonlinear activations, have more than one layer.

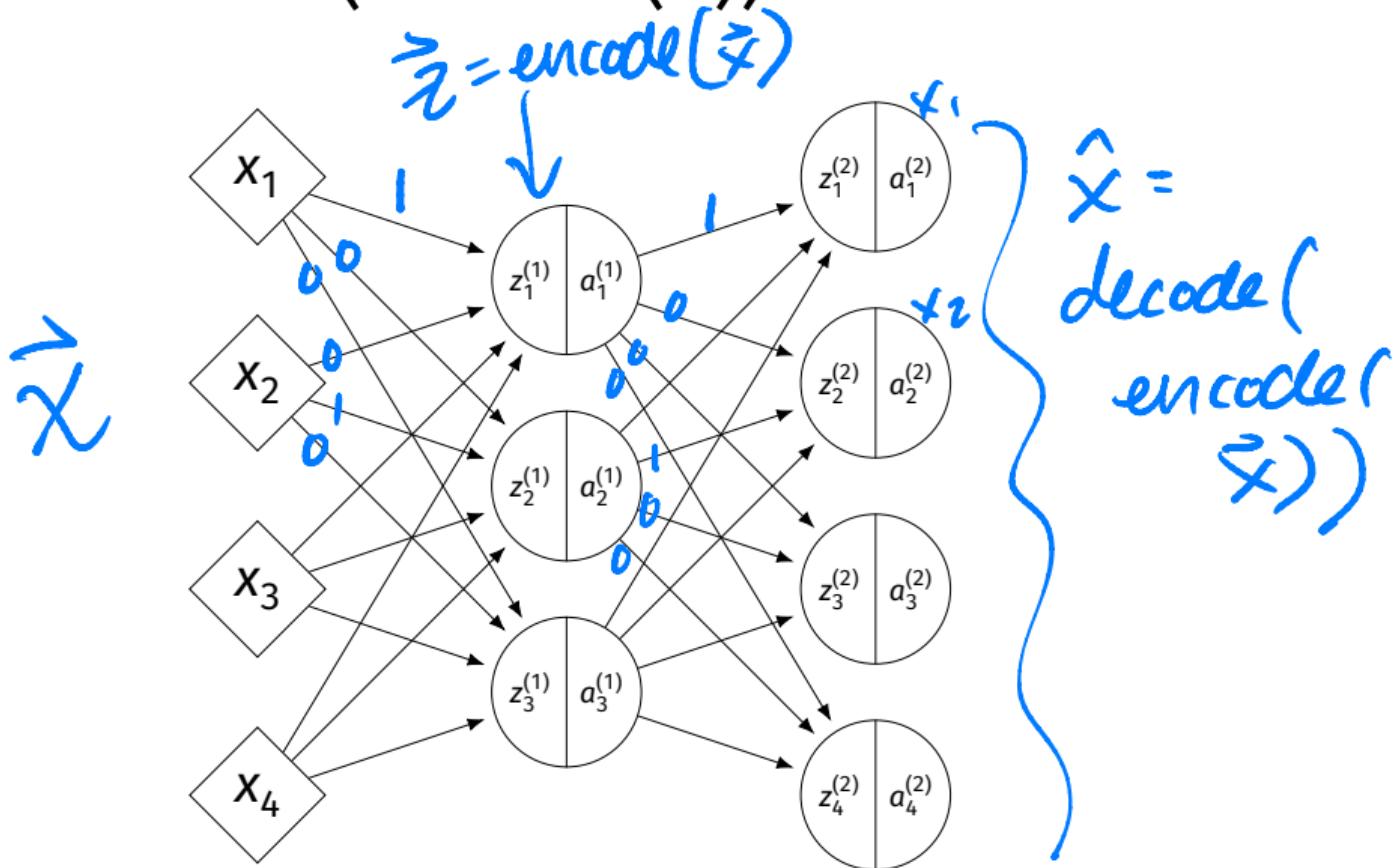
Decoder as a Neural Network

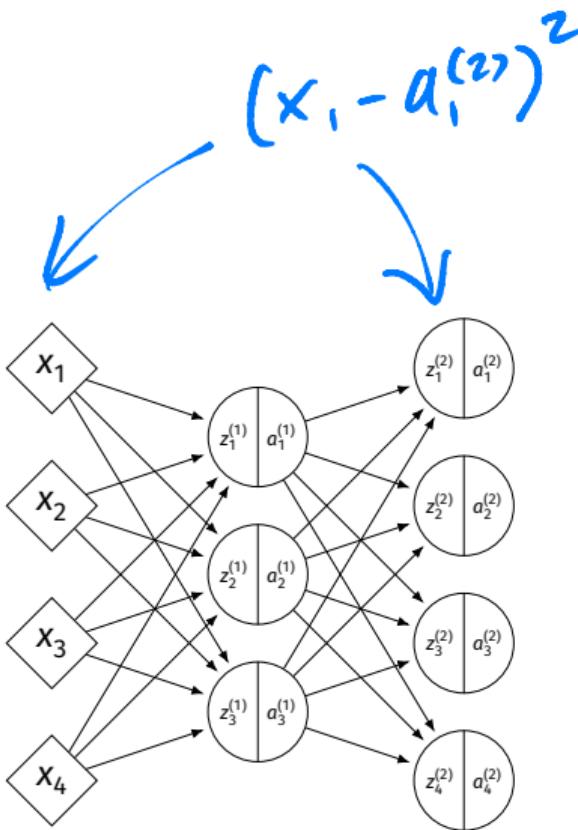


decode(encode(\vec{x})) as a NN

- ▶ Together, $\text{decode}(\text{encode}(\vec{x}))$ is a neural network
 $H(\vec{x}) : \mathbb{R}^d \rightarrow \mathbb{R}^d$.

$\text{decode}(\text{encode}(\vec{x}))$ as a NN





Training

- We want $H(\vec{x}) \approx \vec{x}$
- One approach: train network to minimize reconstruction error.

$$\begin{aligned} \sum_{i=1}^n \|\vec{x}^{(i)} - H(\vec{x}^{(i)})\|^2 &= \sum_{i=1}^n \sum_{j=1}^d (\vec{x}_j^{(i)} - (H(\vec{x}^{(i)}))_j)^2 \\ &= \sum_{i=1}^n \sum_{j=1}^d (\vec{x}_j^{(i)} - a_j^{(2)}(\vec{x}^{(i)}))^2 \end{aligned}$$

Training

- ▶ The network can be trained using gradient-based methods.
 - ▶ E.g., stochastic gradient descent.
- ▶ **Note:** this is an **unsupervised** learning problem.

Autoencoders

- ▶ When the encoder/decoder are NNs,
 $H(\vec{x}) = \text{decode}(\text{encode}(\vec{x}))$ is an **autoencoder**.

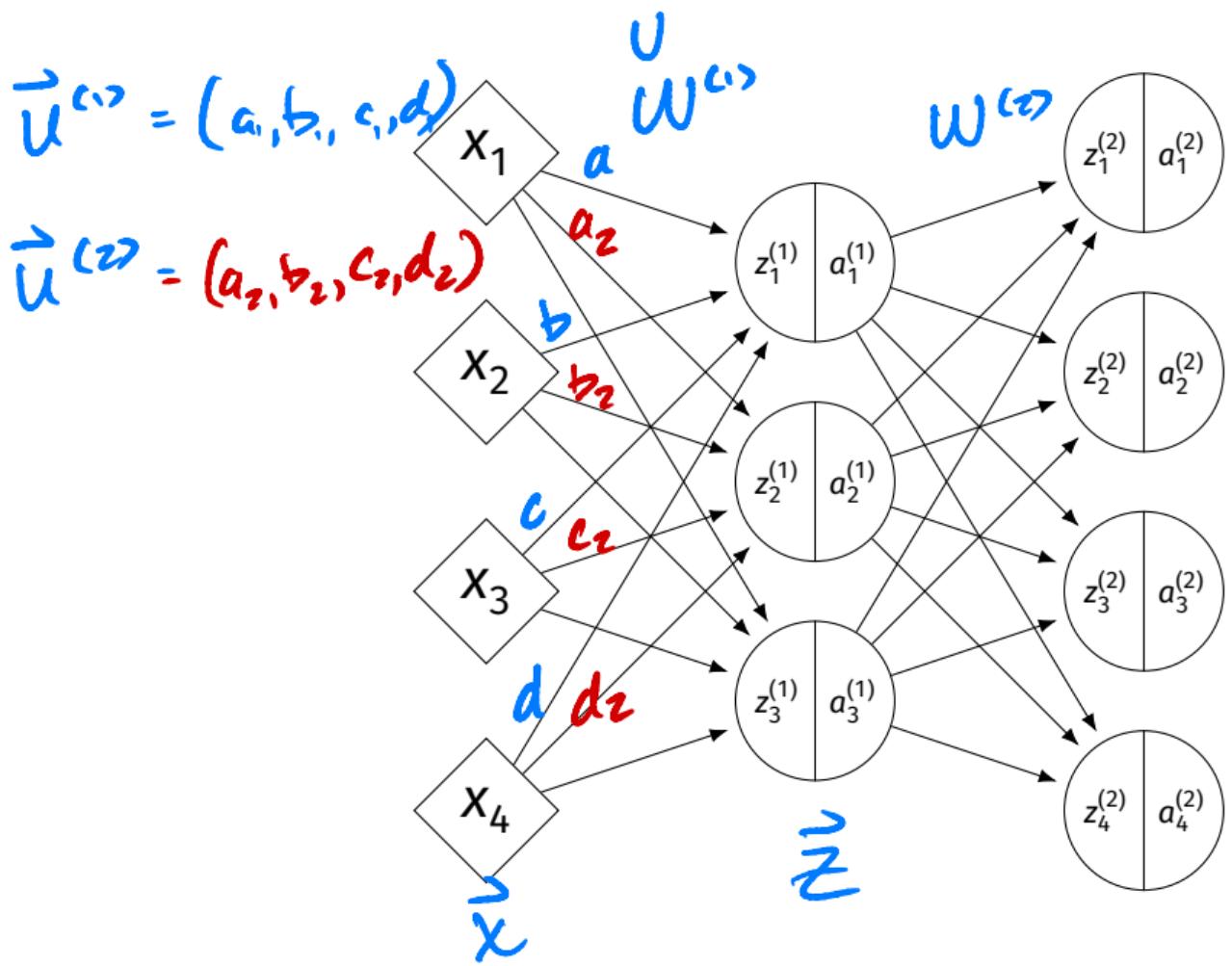
Generalizing PCA

- ▶ We can view autoencoders as generalizations of PCA.
- ▶ Consider again the encoder that performs an orthogonal projection:

$$\text{encode}(\vec{x}) = U^T \vec{x}$$

$$\text{decode}(\vec{z}) = U\vec{z}$$

- ▶ encode/decode are neural networks (with linear activations).



Exercise

True/False: training an autoencoder to minimize reconstruction error will result in the same $\text{encode}(\vec{x})$ function as PCA.

Answer: False

- ▶ PCA minimizes reconstruction error **subject to** the constraint that the columns of U are orthonormal.
- ▶ Without the orthonormality constraint, the autoencoder learns a different encoding.
- ▶ *However*, the autoencoder learns a (non-orthogonal) projection into the same space as PCA.

In other words...

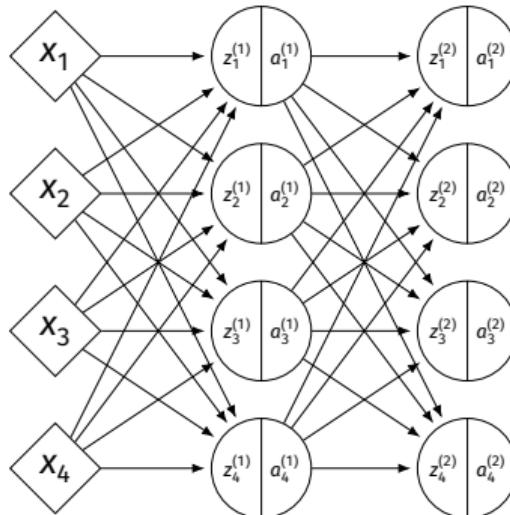
- ▶ PCA is an autoencoder trained with an additional orthonormality constraint.
- ▶ Cannot easily be learned by gradient descent; find eigenvectors instead.

Uses of Autoencoders

- ▶ Like PCA, autoencoders can be used for **dimensionality reduction**.
- ▶ Unlike PCA, autoencoders can learn nonlinear maps.
- ▶ Encoded data can be used as input to predictive model, etc.

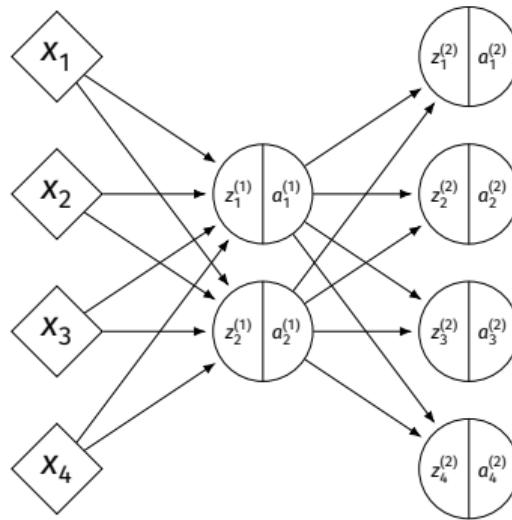
Dimensionality Reduction

- ▶ If the dimensionality of the encoder is the same as the dimensionality of \vec{x} , the autoencoder can learn to simply reproduce the input.



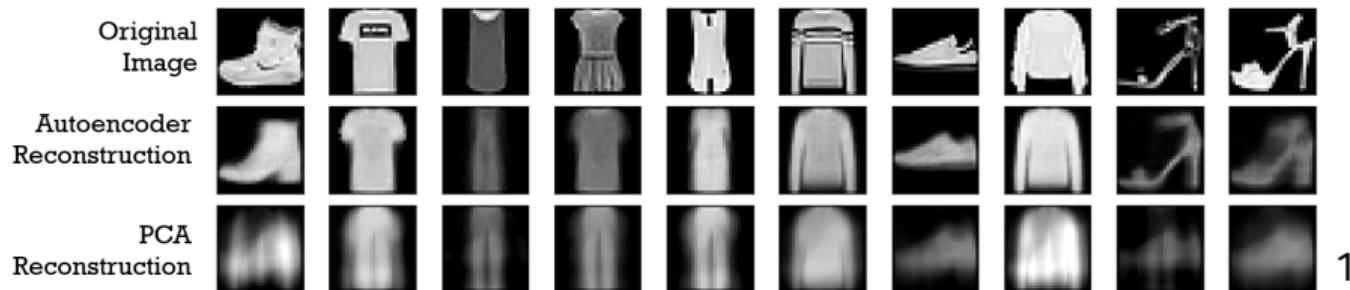
Dimensionality Reduction

- ▶ As such, we choose number of hidden nodes $< d$.



- ▶ Called an **undercomplete autoencoder**.

Example

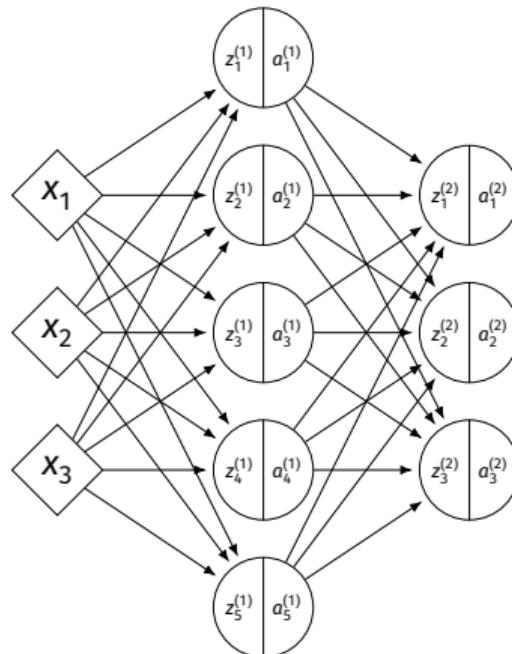


1

¹By Michela Massi - Own work, CC BY-SA 4.0,
<https://commons.wikimedia.org/w/index.php?curid=80176900>

Other Uses

- ▶ However, sometimes it is useful for hidden layer to have **greater** dimensionality.



Denoising Autoencoders

- ▶ One such case is in **denoising autoencoders**.
- ▶ **Idea:** train an autoencoder to remove noise.
- ▶ Add random noise to each $\vec{x}^{(i)}$ to get $\tilde{x}^{(i)}$.
- ▶ Train network so that $H(\tilde{x}^{(i)}) \approx \vec{x}$.

DSC 140B

Representation Learning

Lecture 17 | Part 2

Conclusion of DSC 140B

Recap

- ▶ DSC 140B was about **representation learning**.
- ▶ We saw PCA, Laplacian Eigenmaps, RBF Networks, neural networks and deep learning
- ▶ Learned ML methods, but also theoretical tools for understanding why other ML methods work

More Deep Learning

- ▶ We have only scratched the surface of deep learning.
 - ▶ LSTMs, transformer models, graph neural networks, deep RL, GANs, etc.
- ▶ In this class, we focused on the fundamental principles behind NNs.
- ▶ You might consider taking CSE 151B.

Thanks!