

Table of Contents

Lecture 01 - intro

Lecture 02 - linear - models

Lecture 03 - basis - functions

Lecture 04 - radial - basis - functions

Lecture 05 - kmeans

Lecture 06 - linear - algebra

Lecture 07 - linear - algebra - II

Lecture 08 - pca

Lecture 09 - pca - II

Lecture 10 - laplacian - eigenmaps

Lecture 11 - laplacian - eigenmaps - II

Lecture 12 - multilayer - perceptrons

Lecture 13 - backpropagation

Lecture 14 - activations

Lecture 15 - architecture

Lecture 16 - convolution

DSC 190

Machine Learning: Representations

Lecture 1 | Part 1

Introduction

Welcome to DSC 190

Introduction to Machine Learning: Representations

History

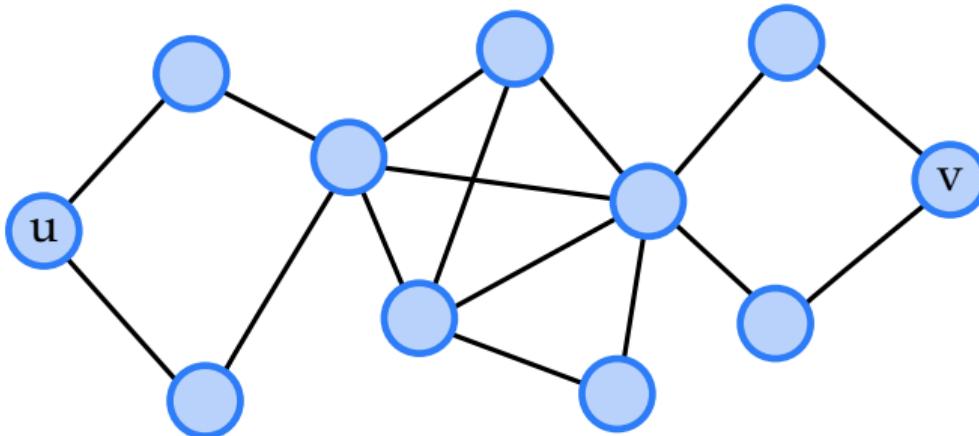
**You've had two classes in ML
already...**

- ▶ DSC 40A (theory)
- ▶ DSC 80 (practice)

What is Machine Learning?

- ▶ Computers can do things very quickly.
- ▶ But must be given really specific instructions.
- ▶ **Problem:** Not all tasks are easy to dictate.

Example (Easy)



Problem: Find a shortest path between u and v .

Example (Not so easy)



Problem: On a scale from 1-10, how happy is this person?

The Trick: Use Data



8



3



5



4



7



6



10



?

What is Machine Learning?

- ▶ Before: Computer is **told** how to do a task.
- ▶ Instead: **learn** how to do a task using data.

What is Machine Learning?

- ▶ Before: Computer is **told** how to do a task.
- ▶ Instead: **learn** how to do a task using data.
- ▶ We still have to **tell** the computer how to learn.

An **ML algorithm** is a set of precise instructions telling the computer **how to learn** from data.

An **ML algorithm** is a set of precise instructions telling the computer **how to learn** from data.

Spoiler: the algorithms are usually pretty simple. It's the **data** that does the real work.

An **ML algorithm** is a set of precise instructions telling the computer **how to learn** from data.

Spoiler: the algorithms are usually pretty simple. It's the **data** that does the real work.

This is because real world data has “**structure**”.



Problem: On a scale from 1-10, how happy is this person?

Exercise

What kind of learning task is this (e.g., classification)? What learning algorithm(s) have you heard of for this kind of task?

Recall: Least Squares Regression

- ▶ Example: predict the price of a laptop.

- ▶ Choose some **features**:

- ▶ CPU speed, amount of RAM, weight (kg).

- ▶ Prediction function (weighted “vote”):

$$(\text{price}) = w_0 + w_1 \times (\text{cpu}) + w_2 \times (\text{ram}) + w_3 \times (\text{weight})$$

- ▶ Learn w_i by minimizing **squared error**.

Representations

- ▶ Computers don't understand the concept of a laptop.
- ▶ We had to **represent** a laptop as a set of features.
 - ▶ CPU speed, amount of RAM, weight (kg).
- ▶ Clearly, choosing right **feature representation** is important.

Now: Predict Happiness



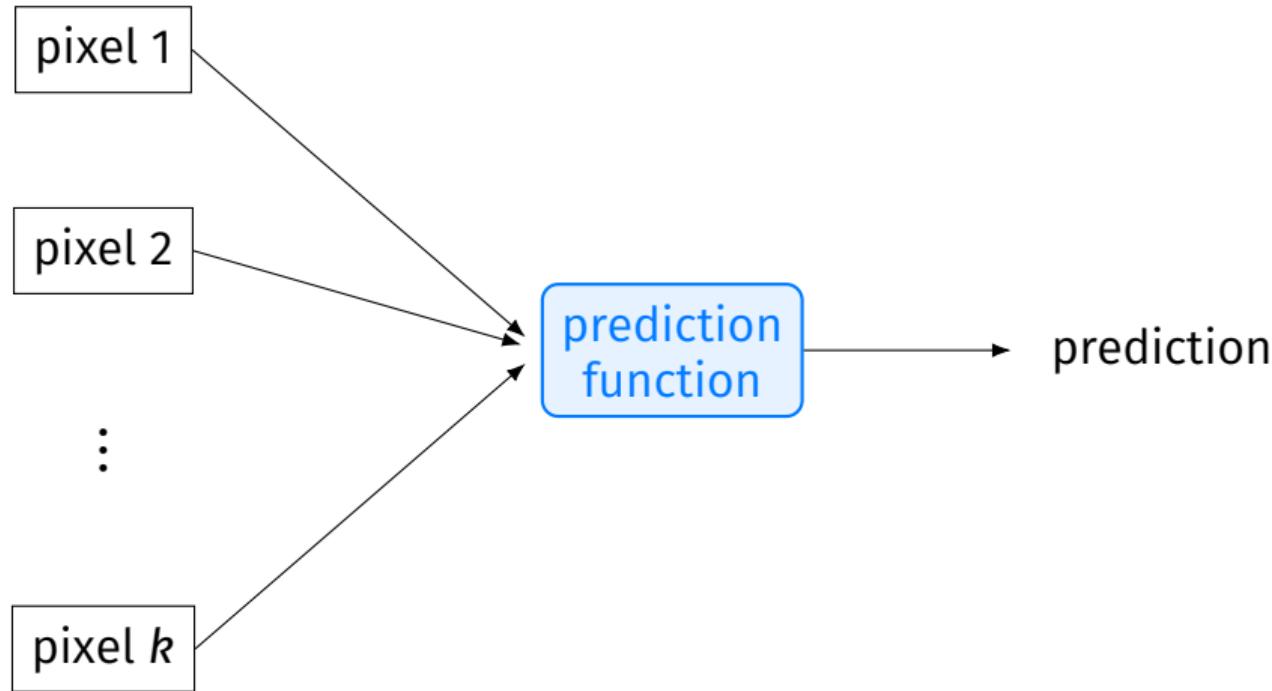
- ▶ Given an image, predict happiness on a 1-10 scale.
- ▶ This is a **regression** problem.
- ▶ Can we use least squares regression?

Problem

- ▶ Computers don't understand images.
- ▶ How do we **represent** them?
- ▶ Easy approach: a bag of pixels.
 - ▶ **Each** pixel has an numerical **intensity**.
 - ▶ Each pixel is a feature.
 - ▶ In this way, an image is represented as a **vector** in some **high dimensional space**.

Least Squares for Happiness

$$\begin{aligned} \text{(happiness)} = & w_0 \\ & + w_1 \times (\text{pixel 1}) \\ & + w_2 \times (\text{pixel 2}) \\ & + \dots \\ & + w_k \times (\text{pixel k}) \end{aligned}$$



Exercise

Say we train a least squares regression model on a set of images to predict happiness. We achieve a mean squared error of M_1 .

Now we scramble every image's pixels *in exactly the same way* (same transformation of each image). We retrain, and achieve MSE of M_2 .

Which is true:

- ▶ $M_1 < M_2$
- ▶ $M_1 = M_2$
- ▶ $M_1 > M_2$

Answer

- ▶ The regression model will work just as well if the images are all scrambled in exactly the same way.
- ▶ This is because the model doesn't use the **proximity** of pixels.
- ▶ The **representation** (each pixel is a feature) does not capture this.

Exercise

Say we train a least squares regression model on a set of images to predict happiness. We achieve a mean squared error of M_1 .

Now we scramble every image's pixels *independently*. We retrain, and achieve MSE of M_2 .

Which is true:

- ▶ $M_1 < M_2$
- ▶ $M_1 = M_2$
- ▶ $M_1 > M_2$

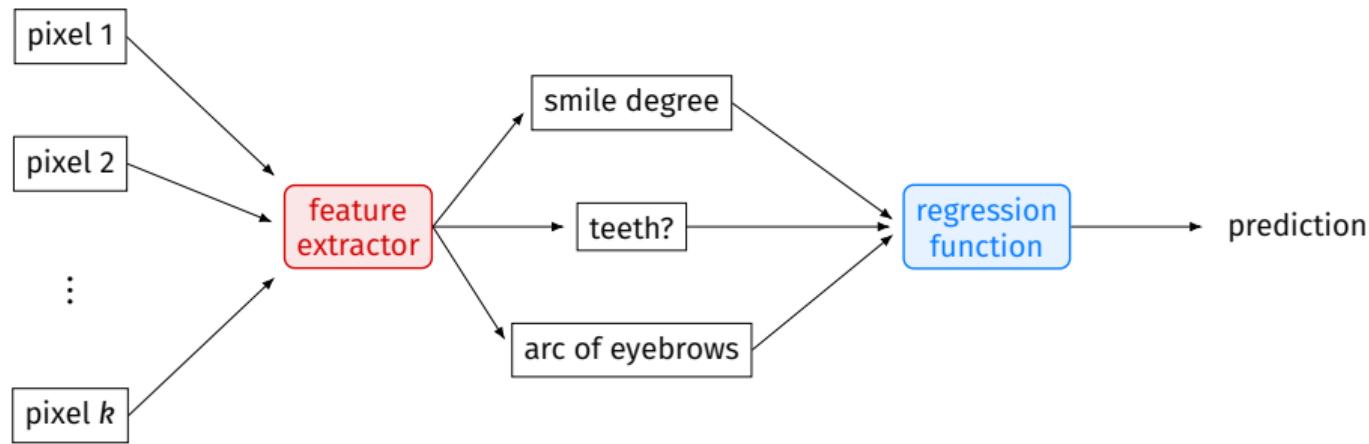
Happiness: it's in the Pixels

- ▶ The information is contained in the image... but not in individual pixels.
- ▶ In **patterns** of pixels:
 - ▶ The shape of the eyebrows.
 - ▶ Angle of the corners of the mouth.
 - ▶ Are teeth visible?
- ▶ The representation is **too simple** – probably won't work well¹.

¹On this example! Works OK on, e.g., MNIST.

Handcrafted Representations

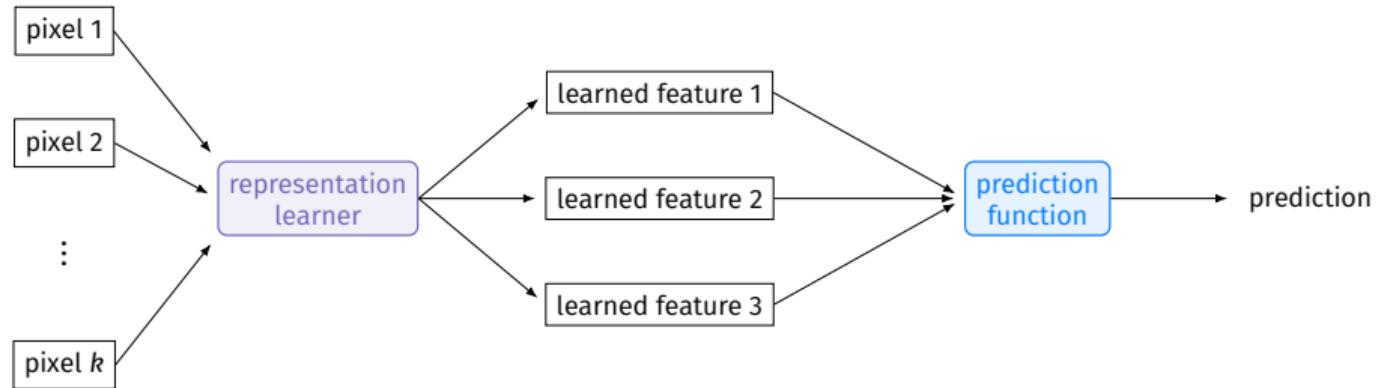
- ▶ Idea: build a **feature extractor** to detect:
 - ▶ The shape of the eyebrows.
 - ▶ Angle of the corners of the mouth.
 - ▶ Are teeth visible?
- ▶ Use these as high-level features instead.



Problem

- ▶ Extractors (may) make good **representations**.
- ▶ But building a feature extractor is **hard**².
- ▶ Can we **learn** a good representation?

²It took evolution a while to come up with the visual hierarchy.



DSC 190

- ▶ We'll see how to **learn good representations**.
- ▶ Good representations help us when:
 1. making predictions;
 2. doing EDA (better visualizations).

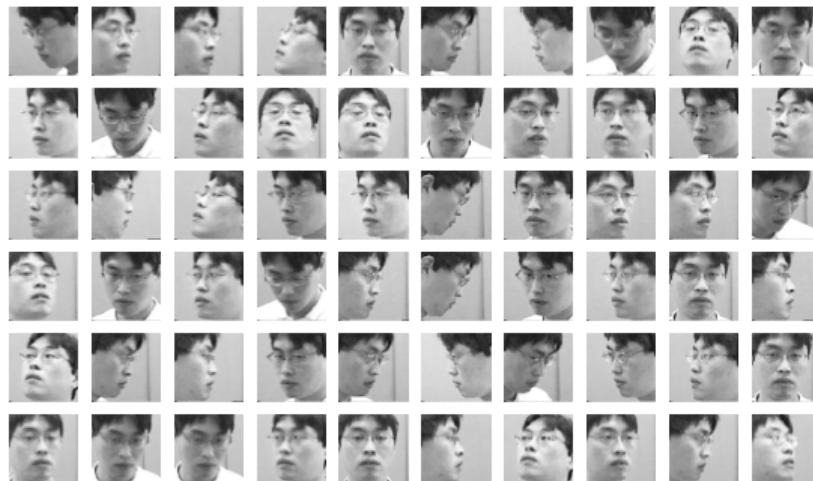
Claim

- ▶ Many of the famous recent advancements in AI/ML are due to **representation learning**.

Representations and Structure

- ▶ Real world data has structure.
- ▶ But “seeing” the structure requires the right representation.

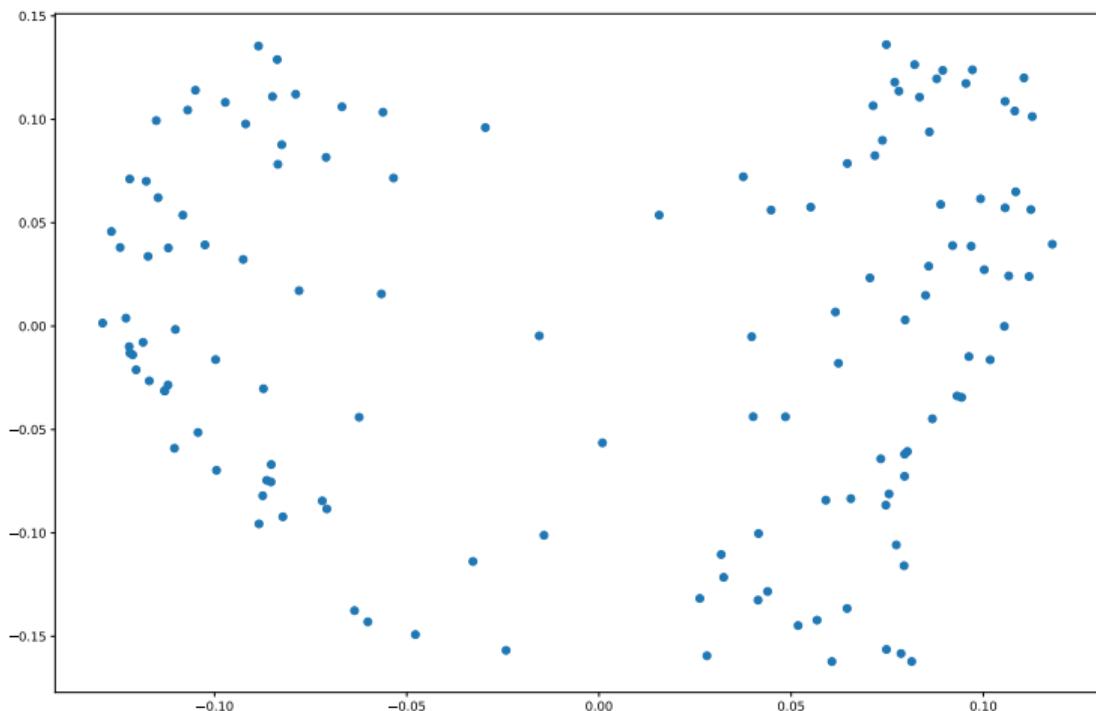
Example: Pose Estimation

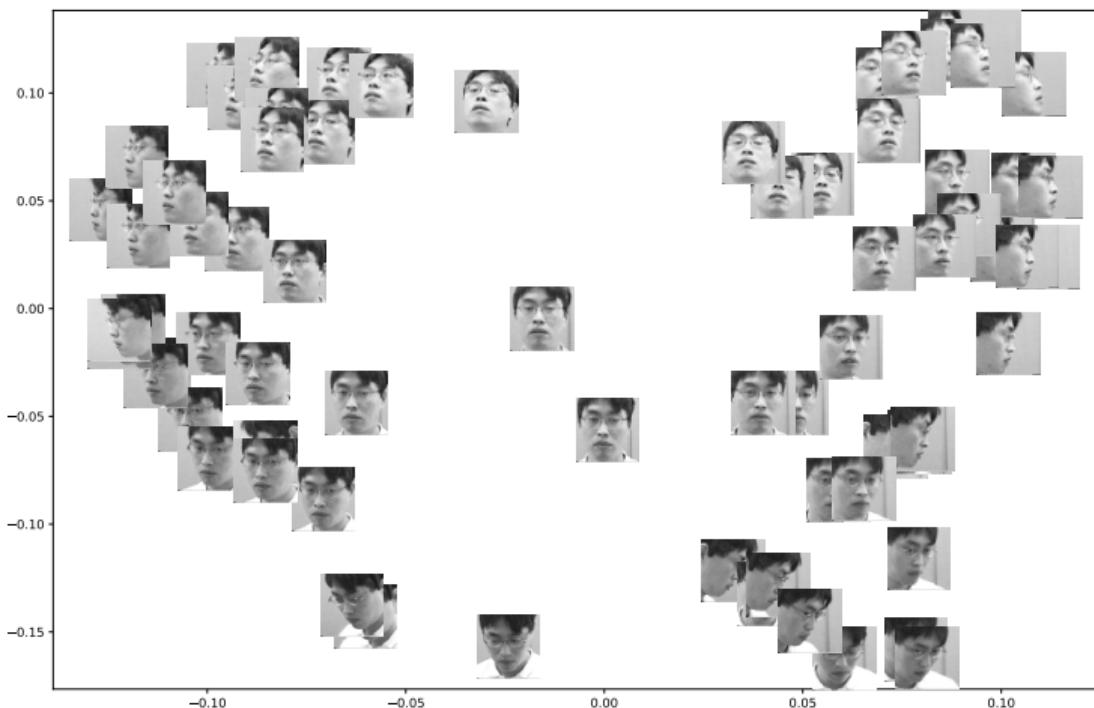


Problem: Classify, is person looking left, right, up, down, neutral?

Example: Pose Estimation

- ▶ As a “bag of pixels” each image is a vector in $\mathbb{R}^{10,000}$.
- ▶ Later: we’ll see how to reduce dimensionality while preserving “closeness”.





Main Idea

By learning a better representation, the original classification problem becomes easy, almost trivial.

Example: word2vec

- ▶ How do we represent a word?
- ▶ Google's word2vec learned a representation of words as points in 300 dimensional space.
- ▶ Two points close \iff words have similar meanings.

Example: word2vec

- ▶ Fun fact: we can now add and subtract words.
 - ▶ They're represented as vectors.
- ▶ Surprising results:

$$\vec{v}_{\text{Paris}} - \vec{v}_{\text{France}} + \vec{v}_{\text{China}} \approx \vec{v}_{\text{Beijing}}$$

Example: word2vec³

Table 8: Examples of the word pair relationships, using the best word vectors from Table 4 (Skip-gram model trained on 783M words with 300 dimensionality).

| Relationship | Example 1 | Example 2 | Example 3 |
|----------------------|---------------------|-------------------|----------------------|
| France - Paris | Italy: Rome | Japan: Tokyo | Florida: Tallahassee |
| big - bigger | small: larger | cold: colder | quick: quicker |
| Miami - Florida | Baltimore: Maryland | Dallas: Texas | Kona: Hawaii |
| Einstein - scientist | Messi: midfielder | Mozart: violinist | Picasso: painter |
| Sarkozy - France | Berlusconi: Italy | Merkel: Germany | Koizumi: Japan |
| copper - Cu | zinc: Zn | gold: Au | uranium: plutonium |
| Berlusconi - Silvio | Sarkozy: Nicolas | Putin: Medvedev | Obama: Barack |
| Microsoft - Windows | Google: Android | IBM: Linux | Apple: iPhone |
| Microsoft - Ballmer | Google: Yahoo | IBM: McNealy | Apple: Jobs |
| Japan - sushi | Germany: bratwurst | France: tapas | USA: pizza |

³“Efficient Estimation of Word Representations in Vector Space” by Mikolov, et al.

Example: Neural Networks

- ▶ word2vec is an example of a neural network model.
- ▶ Deep neural networks have been very successful on certain tasks.
- ▶ They **learn** a good representation.



Main Idea

Building a good model requires picking a good **feature representation**.

We can pick features by hand.

Or we can **learn** a good feature representation from data.

That is what this class is about.

Roadmap

- ▶ Review of DSC 40A:
 - ▶ Minimizing loss
 - ▶ Linear models for regression and classification
- ▶ Clustering as feature learning
 - ▶ k-means clustering
 - ▶ RBF networks
- ▶ Dimensionality reduction
 - ▶ Review of linear algebra
 - ▶ Eigenvalues/Eigenvectors
 - ▶ PCA

Roadmap

- ▶ Manifold learning
- ▶ Neural Networks
- ▶ Autoencoders
- ▶ Deep Learning

Practice vs. Theory

- ▶ Goal of this class: understand the fundamentals of representation learning.
- ▶ Both practical and theoretical.
- ▶ Think: more DSC 40A than DSC 80, but a bit of both.

Tools of the Trade

- ▶ We'll see some of the popular Python tools for feature learning.
 - ▶ numpy
 - ▶ keras
 - ▶ sklearn
 - ▶ ...

DSC 190

Machine Learning: Representations

Lecture 1 | Part 2

Syllabus

Miscellaneous

- ▶ Campuswire > Email
- ▶ No discussion tomorrow.

DSC 190

Machine Learning: Representations

Lecture 1 | Part 3

Is DSC 190 for You?

Is DSC 190 for you?

- ▶ DSC 190 will eventually become DSC 140B.
- ▶ DSC 140A/140B are **targeted** to DSC majors.
 - ▶ Compared to other ML classes, Assume some ML background (40A, 80).

Is DSC 190 for you?

- ▶ Unfortunately, it's a little confusing.
- ▶ DSC 190 and CSE 151A are equivalent in credit.
- ▶ Not equivalent in topics.
- ▶ Consequence of creating our own ML in DSC.

Bottom Line

- ▶ If you are a DSC major, haven't taken an ML class:
 - ▶ Take this class and DSC 140A (in either order).
- ▶ If you are a DSC major, have taken an ML class:
 - ▶ Talk to an advisor.

"This course substitutes the CORE CSE 151A Requirement. Students cannot receive major or minor credit for both CSE 151A and DSC 190 A00- SP22, as only one course can fulfill this major core requirement."

Bottom Line

- ▶ If you're not a DSC major, looking for an ML elective:
 - ▶ This course *might* be a good option if you already have some ML background.
 - ▶ But it is targeted to data scientists.
 - ▶ CSE 151A, DSC 80, DSC 148, CSE 158, etc. may be better options.

Next Time

- ▶ Review of DSC 40A topics.
- ▶ Learning as optimizing loss.
- ▶ Linear models for regression and classification.

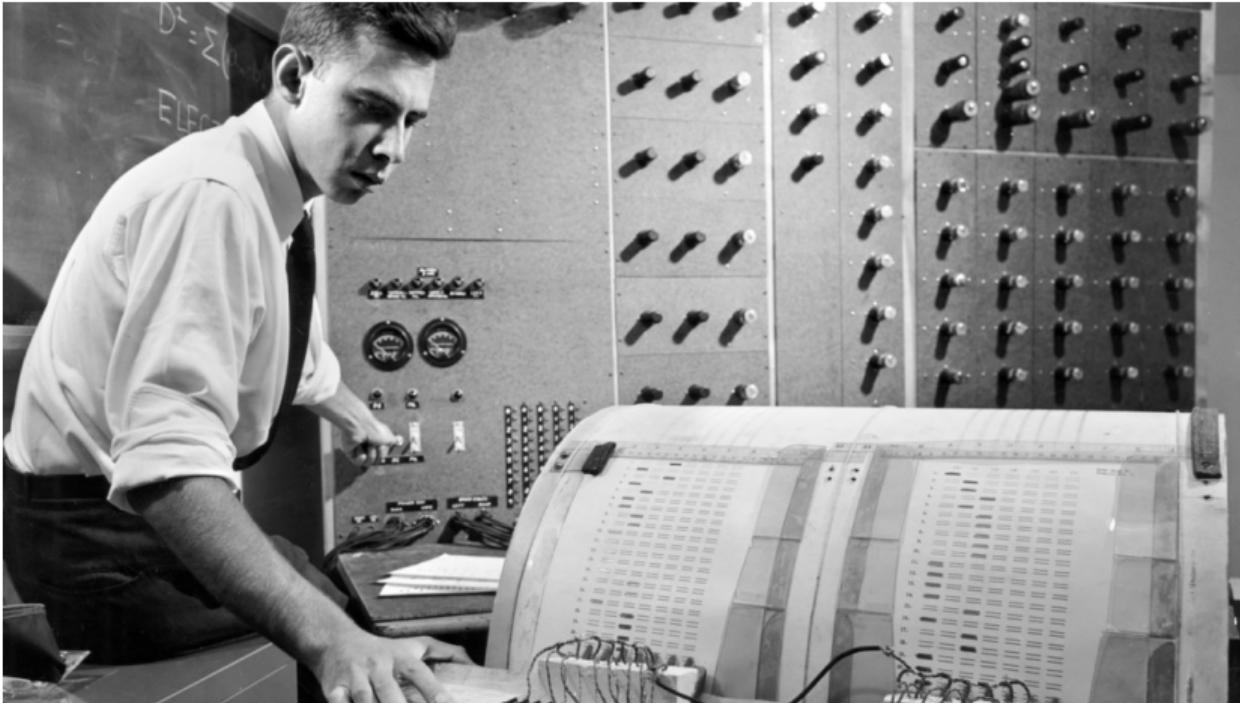
DSC 190

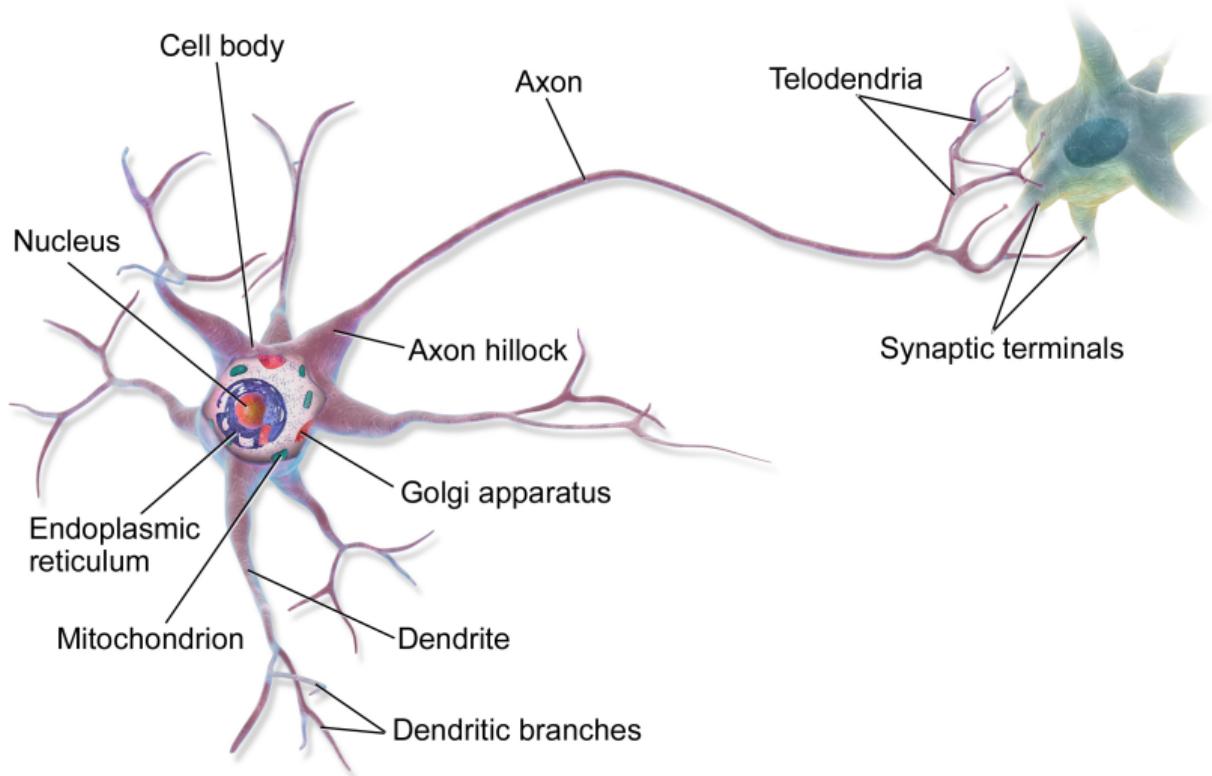
Machine Learning: Representations

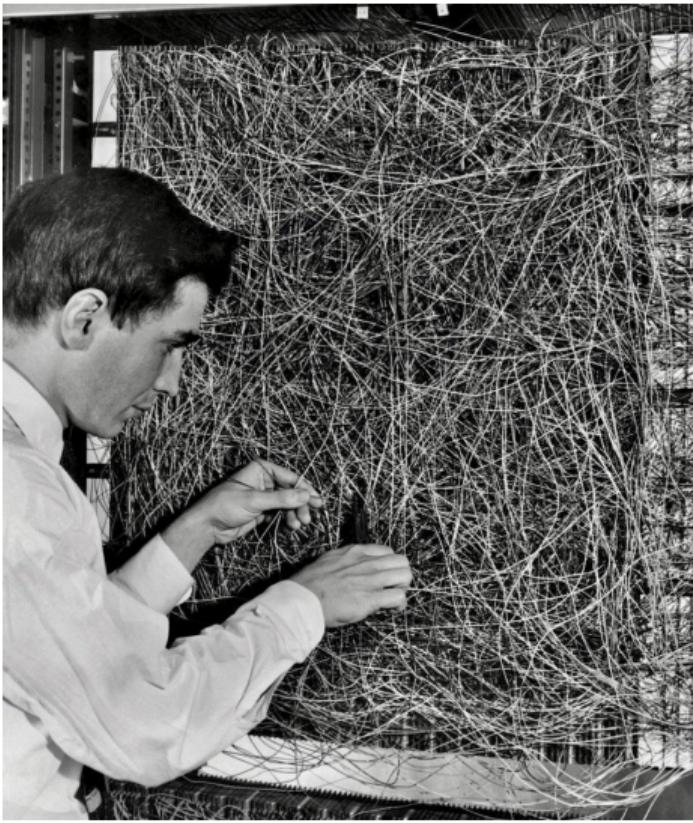
Lecture 2 | Part 1

Learning in the 1950s

Rosenblatt's Perceptron







The Task

NEW NAVY DEVICE LEARNS BY DOING

Psychologist Shows Embryo
of Computer Designed to
Read and Grow Wiser

WASHINGTON, July 7 (UPI) — The Navy revealed the embryo of an electronic computer today that it expects will be able to walk, talk, see, write, reproduce itself and be conscious of its existence.

The embryo—the Weather Bureau's \$2,000,000 "704" computer—learned to differentiate between right and left after fifty attempts in the Navy's demonstration for newsmen.

The service said it would use this principle to build the first of its Perceptron thinking machines that will be able to read and write. It is expected to be finished in about a year at a cost of \$100,000.

Dr. Frank Rosenblatt, designer of the Perceptron, conducted the demonstration. He said the machine would be the first device to think as the human brain. As do human be-

ings, Perceptron will make mistakes at first, but will grow wiser as it gains experience, he said.

Dr. Rosenblatt, a research psychologist at the Cornell Aeronautical Laboratory, Buffalo, said Perceptrons might be fired to the planets as mechanical space explorers.

Without Human Controls

The Navy said the perceptron would be the first non-living mechanism "capable of receiving, recognizing and identifying its surroundings without any human training or control."

The "brain" is designed to remember images and information it has perceived itself. Ordinary computers remember only what is fed into them on punch cards or magnetic tape.

Later Perceptrons will be able to recognize people and call out their names and instantly translate speech in one language to speech or writing in another language, it was predicted.

Mr. Rosenblatt said in principle it would be possible to build brains that could reproduce themselves on an assembly line and which would be conscious of their existence.

1958 New York Times...

In today's demonstration, the "704" was fed two cards, one with squares marked on the left side and the other with squares on the right side.

Learns by Doing

In the first fifty trials, the machine made no distinction between them. It then started registering a "Q" for the left squares and "O" for the right squares.

Dr. Rosenblatt said he could explain why the machine learned only in highly technical terms. But he said the computer had undergone a "self-induced change in the wiring diagram."

The first Perceptron will have about 1,000 electronic "association cells" receiving electrical impulses from an eye-like scanning device with 400 photo-cells. The human brain has 10,000,000,000 responsive cells, including 100,000,000 connections with the eyes.

By the end of today...

- ▶ We'll see how this machine worked.
- ▶ We'll build one to recognize images.
- ▶ We'll see what its limitations are.

DSC 190

Machine Learning: Representations

Lecture 2 | Part 2

Learning to Predict

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

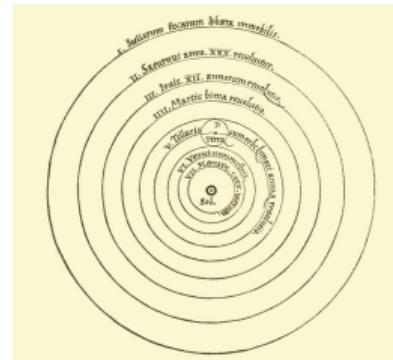
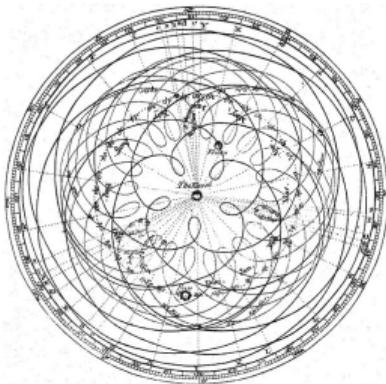
- ▶ A **hypothesis class** \mathcal{H} is a set of possible prediction functions.
- ▶ By choosing a hypothesis class, we are saying something about what the prediction function should look like.
- ▶ Examples:
 - ▶ $\mathcal{H} :=$ linear functions
 - ▶ $\mathcal{H} :=$ functions of the form $\sin(w_1x_1 + \dots + x_5x_5)$
 - ▶ $\mathcal{H} :=$ decision trees of depth 10
 - ▶ $\mathcal{H} :=$ neural networks with one layer

Exercise

Why not just choose \mathcal{H} to be the set of all possible functions?

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 190

Machine Learning: Representations

Lecture 2 | Part 3

Linear 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:
 - ▶ $x_1: 9$
 - ▶ $x_2: 3$
 - ▶ $x_3: 7$
 - ▶ $x_4: 2$
 - ▶ $x_5: 8$
- ▶ Predict the average: ⁴
$$H(\vec{x}) = (x_1 + x_2 + x_3 + x_4 + x_5)/5 = (9 + 3 + 7 + 2 + 8)/5$$

⁴There is only one function in this hypothesis class: $(x_1 + x_2 + \dots + x_5)/5$

Exercise

Why is this a bad prediction function?

A Better Hypothesis Class

- ▶ A weighted “vote”:

$$H(\vec{x}) = w_1x_1 + w_2x_2 + w_3x_3 + w_4x_4 + w_5x_5$$

Exercise

Suppose you are a cynic (you dislike everything). How can the prediction function be changed to take into account the fact that your ratings are likely lower than average across the board?

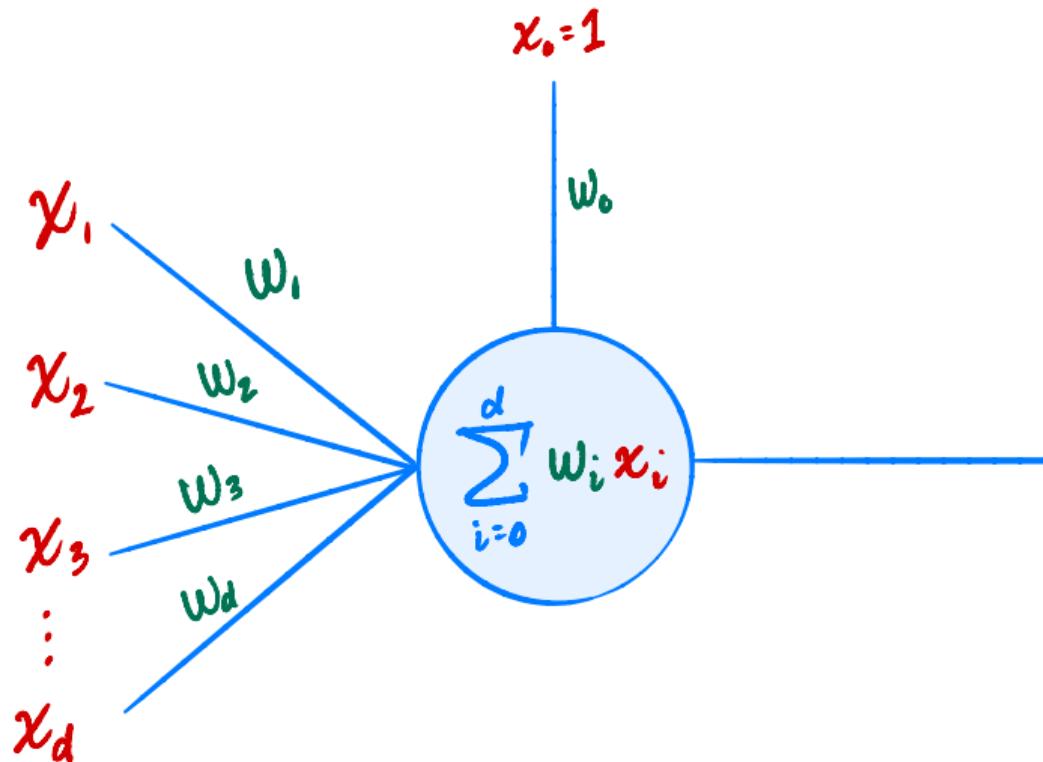
$$H(\vec{x}) = w_1x_1 + w_2x_2 + w_3x_3 + w_4x_4 + w_5x_5$$

An Even Better Hypothesis Class: 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

$$H(\vec{x}) = w_0 + w_1x_1 + w_2x_2 + w_3x_3 + w_4x_4 + w_5x_5$$

- ▶ 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:

$$\begin{aligned} H(\vec{x}) &= w_0 + w_1 x_1 + w_2 x_2 + \dots + w_d x_d \\ &= \text{Aug}(\vec{x}) \cdot \vec{w} \end{aligned}$$

$$\vec{w} = \begin{pmatrix} w_0 \\ w_1 \\ \vdots \\ w_d \end{pmatrix} \quad \text{Aug}(\vec{x}) = \begin{pmatrix} 1 \\ x_1 \\ x_2 \\ \vdots \\ x_d \end{pmatrix}$$

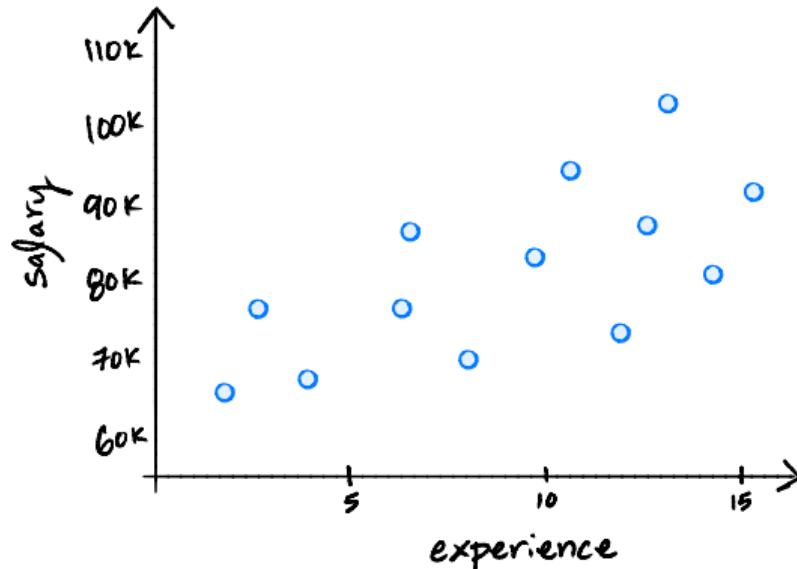
Geometric Meaning

- ▶ It can be very useful to **think geometrically** when reasoning about prediction algorithms.

Example

- ▶ A linear prediction function for salary.

$$H_1(\vec{x}) = \$50,000 + (\text{experience}) \times \$8,000$$



Regression

- ▶ 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, and⁵
 - ▶ $\vec{x} \in \mathbb{R}^1$, 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**.

⁵when 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 “ i th training example”
 - ▶ $\vec{x}_j^{(i)}$ is the “ j th entry of the i th 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

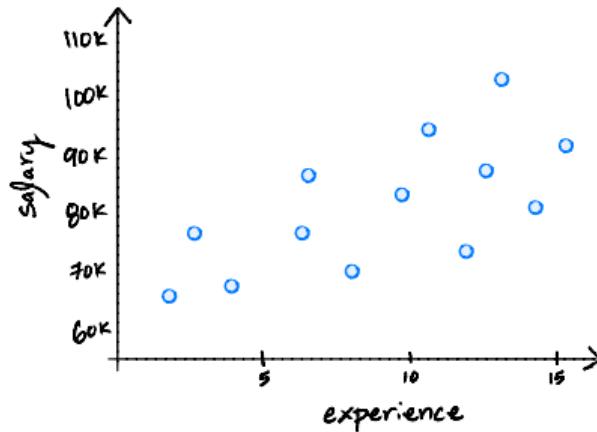
- ▶ A good H is good *on average* over entire data set.
- ▶ The **expected loss** (or **empirical risk**) is one way of measuring this:

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

- ▶ 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:** $(H(\vec{x}^{(i)}) - y_i)^2$

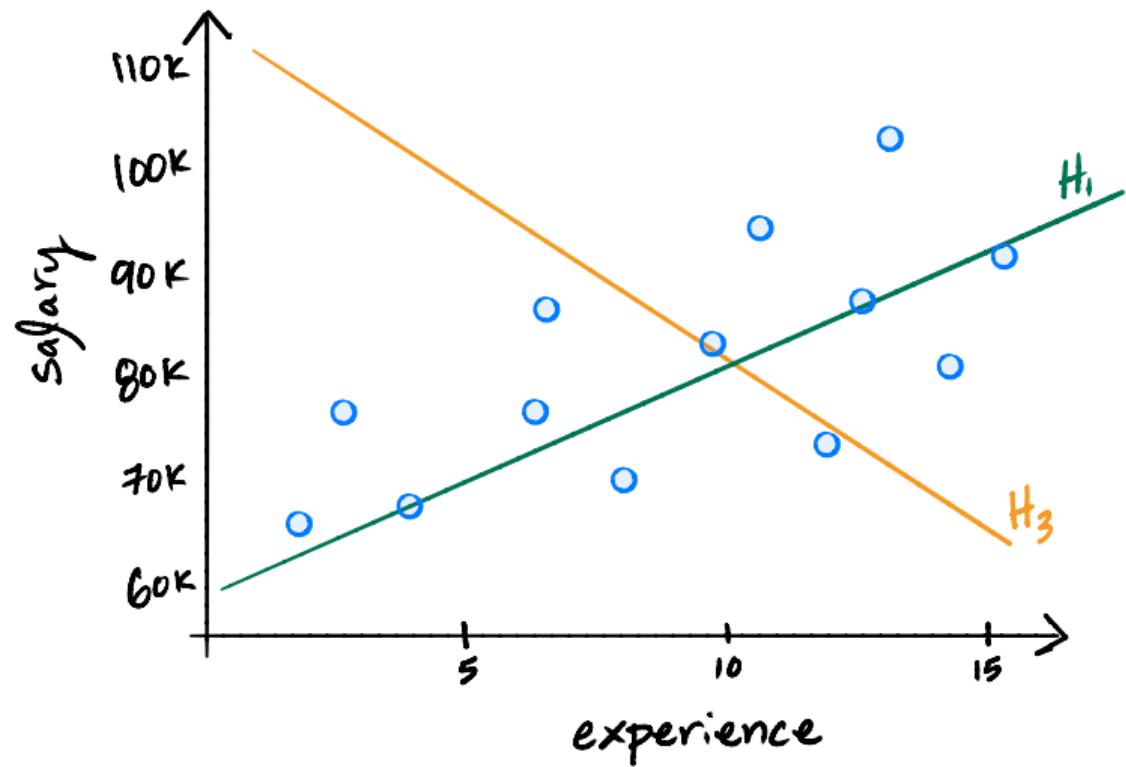


Mean Squared Error

- ▶ **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.



Step #3: Minimize MSE

- We want to find an H minimizing this:

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

- 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

- We want to find \vec{w} that minimizes:

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

- 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 190

Machine Learning: Representations

Lecture 2 | Part 4

Linear Classification

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.

$$H(\vec{x}) = w_0 + w_1 x_1 + w_2 x_2 + \dots + w_d x_d$$

- ▶ Same ERM paradigm also useful.

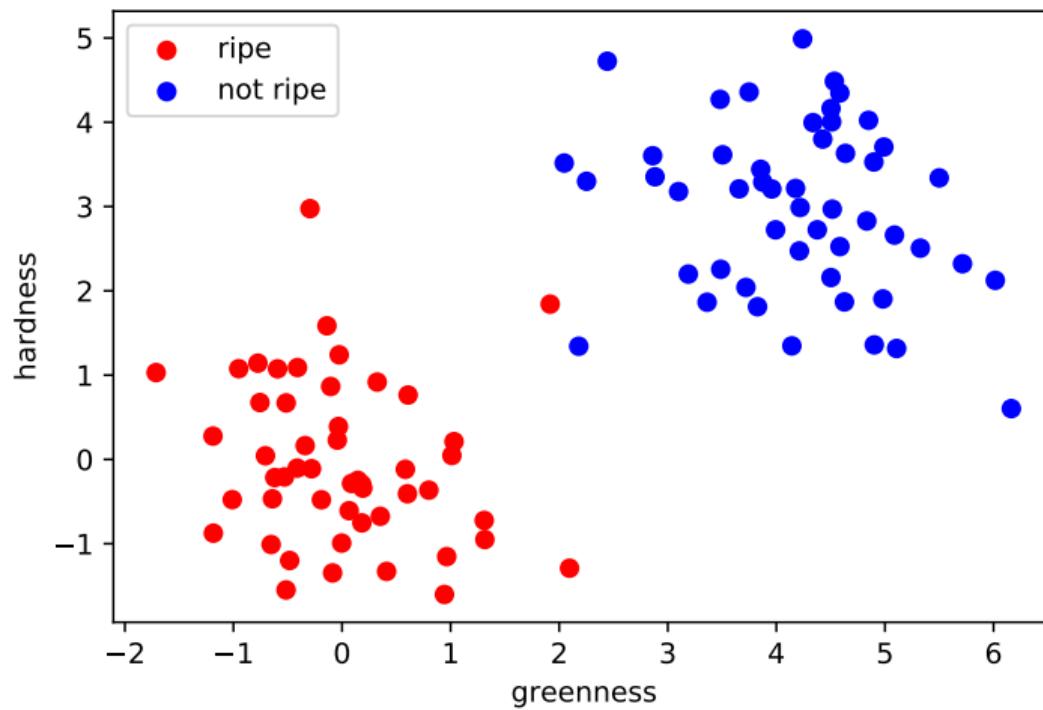
Classification

- ▶ In classification, the output of the prediction function is a discrete **label**.
- ▶ Our linear prediction functions output real numbers.
- ▶ But we can use a linear prediction function as a classifier by, e.g., thresholding.

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 |
| : | : | : |



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:

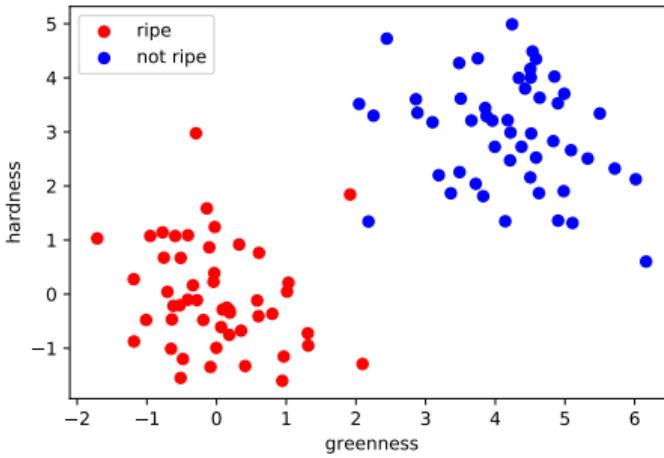
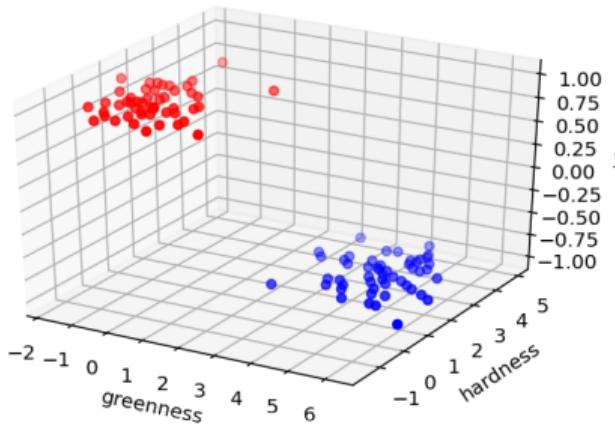
$$\text{sign}(z) = \begin{cases} 1 & z > 0 \\ -1 & z < 0 \\ 0 & \text{otherwise} \end{cases}$$

- ▶ Final prediction: $\text{sign}(H(\vec{x}))$

Decision Boundary

- ▶ The **decision boundary** is the place where the output of $H(x)$ switches from “yes” to “no”.
 - ▶ If $H > 0 \mapsto \text{“yes”}$ and $H < 0 \mapsto \text{“no”}$, the decision boundary is where $H = 0$.
- ▶ If H is a linear predictor and⁶
 - ▶ $\vec{x} \in \mathbb{R}^1$, 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.

⁶when 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

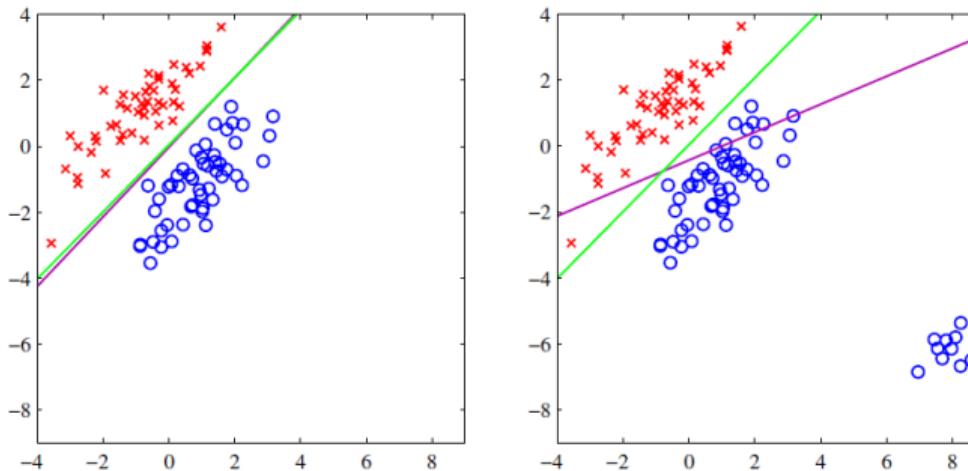


Figure 4.4 The left plot shows data from two classes, denoted by red crosses and blue circles, together with the decision boundary found by least squares (magenta curve) and also by the logistic regression model (green curve), which is discussed later in Section 4.3.2. The right-hand plot shows the corresponding 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.

7

Loss Functions for (Binary) Classification

- ▶ $H(\vec{x})$ is prediction, $y_i \in \{-1, 1\}$ is correct answer
- ▶ Another loss function: **0-1 loss.**

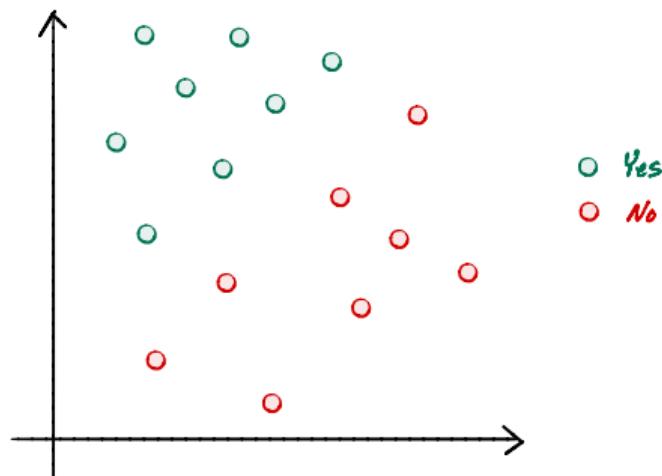
$$L(H(\vec{x}^{(i)}, y_i)) = \begin{cases} 0 & \text{if } \text{sign}(H(\vec{x}^{(i)})) = y_i \\ 1 & \text{if } \text{sign}(H(\vec{x}^{(i)})) \neq y_i \end{cases}$$

- ▶ Expected 0-1 loss:

$$R_{0-1}(H) = \frac{1}{n} \sum_{i=1}^n \begin{cases} 0 & \text{if } \text{sign}(H(\vec{x}^{(i)})) = y_i \\ 1 & \text{if } \text{sign}(H(\vec{x}^{(i)})) \neq y_i \end{cases}$$

0-1 Loss

- The expected 0-1 loss is simply proportion of misclassified points.



Problem

- ▶ The 0-1 loss is not differentiable.
- ▶ Can't even use gradient descent...
- ▶ In fact, NP-Hard to optimize expected 0-1 loss in general.

Perceptron Loss

- ▶ The **perceptron loss** is designed for binary classification.

$$L(H(\vec{x}), y) = \begin{cases} 0, & \text{sign}(H(\vec{x})) = \text{sign}(y) \\ |H(\vec{x})|, & \text{sign}(H(\vec{x})) \neq \text{sign}(y) \end{cases}$$

- ▶ Empirical expected perceptron loss (risk):

$$H_{\text{tron}}(H(\vec{x}), y) = \begin{cases} 0, & \text{sign}(H(\vec{x})) = \text{sign}(y) \\ |H(\vec{x})|, & \text{sign}(H(\vec{x})) \neq \text{sign}(y) \end{cases}$$

Perceptron Training

- ▶ There's no closed form solution for the \vec{w} that minimizes expected perceptron loss.
 - ▶ Unlike expected square loss.
- ▶ Train iteratively using gradient descent.

Perceptrons

- ▶ A **perceptron** is a linear prediction function trained using the perceptron loss.

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 190

Machine Learning: Representations

Lecture 2 | Part 5

Classification Demo: MNIST

Demo: MNIST

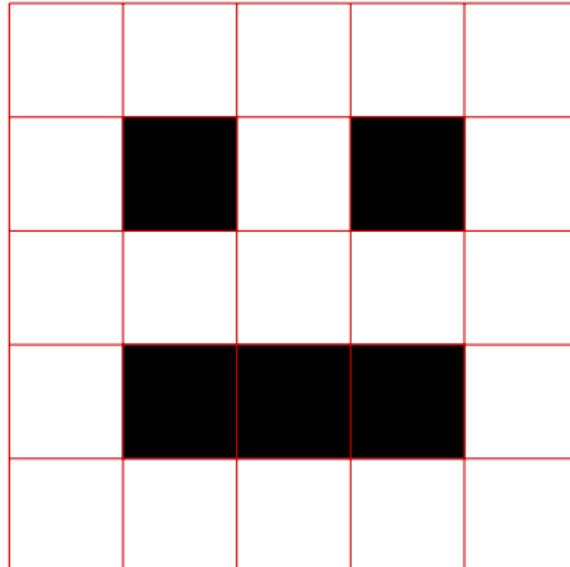
- ▶ MNIST is a classic machine learning data set.
- ▶ Many images of handwritten digits, 0-9.
- ▶ Multiclass classification problem.
- ▶ But we can make it binary: 3 vs. 7.

Example MNIST Digit



- ▶ Grayscale
- ▶ 28 x 28 pixels

Images as Vectors

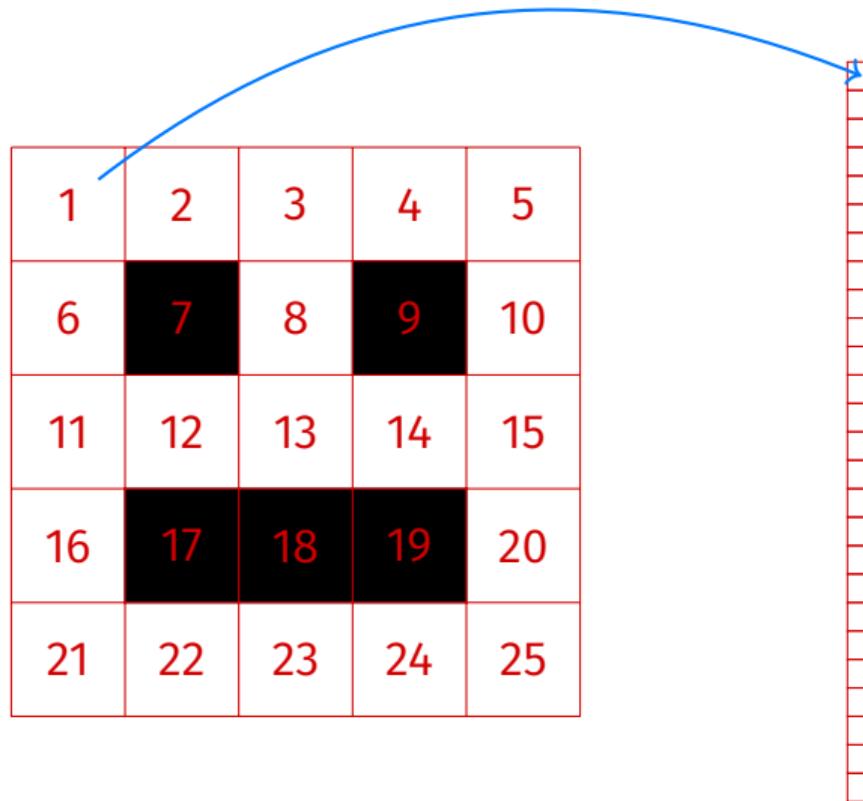


Images as Vectors

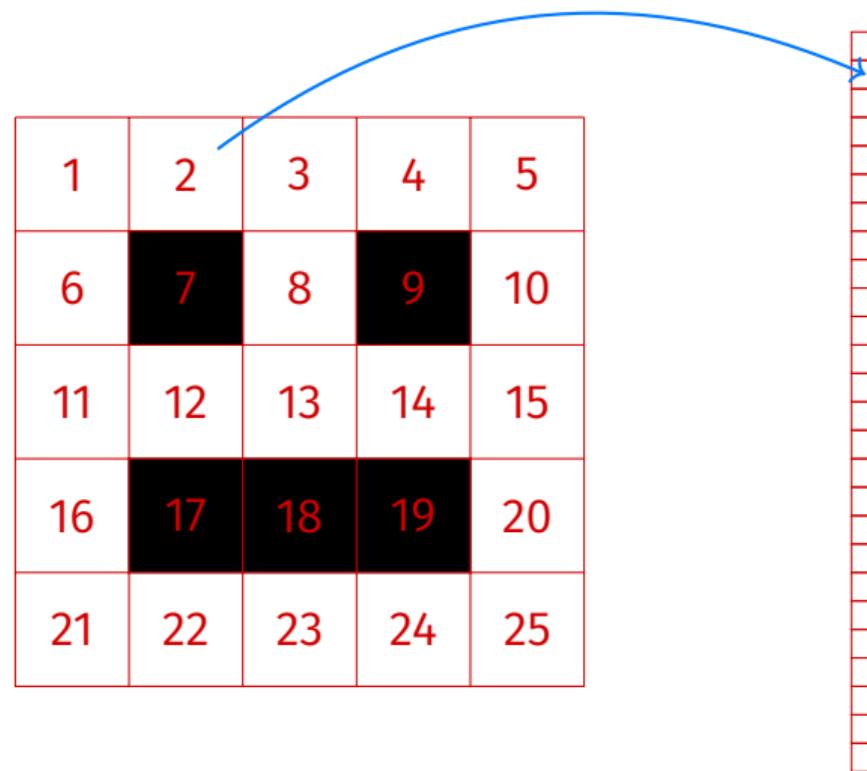
| | | | | |
|----|----|----|----|----|
| 1 | 2 | 3 | 4 | 5 |
| 6 | 7 | 8 | 9 | 10 |
| 11 | 12 | 13 | 14 | 15 |
| 16 | 17 | 18 | 19 | 20 |
| 21 | 22 | 23 | 24 | 25 |



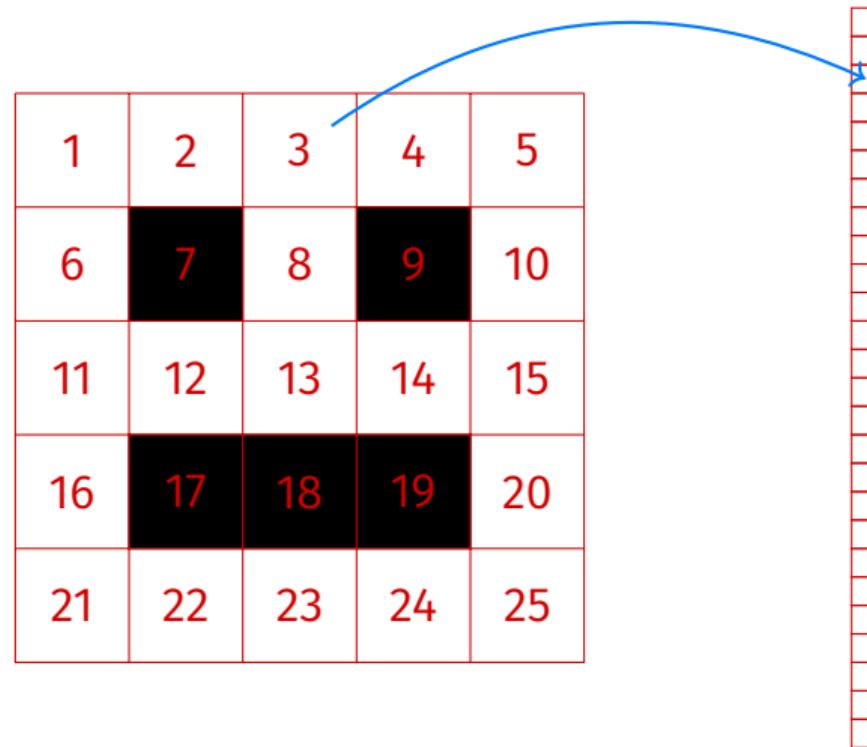
Images as Vectors



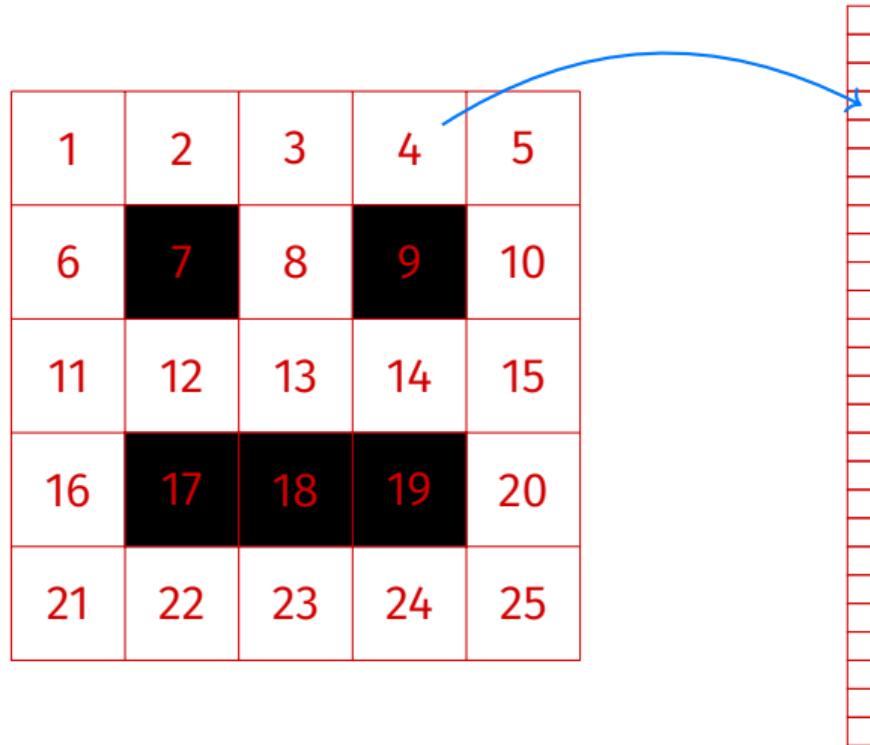
Images as Vectors



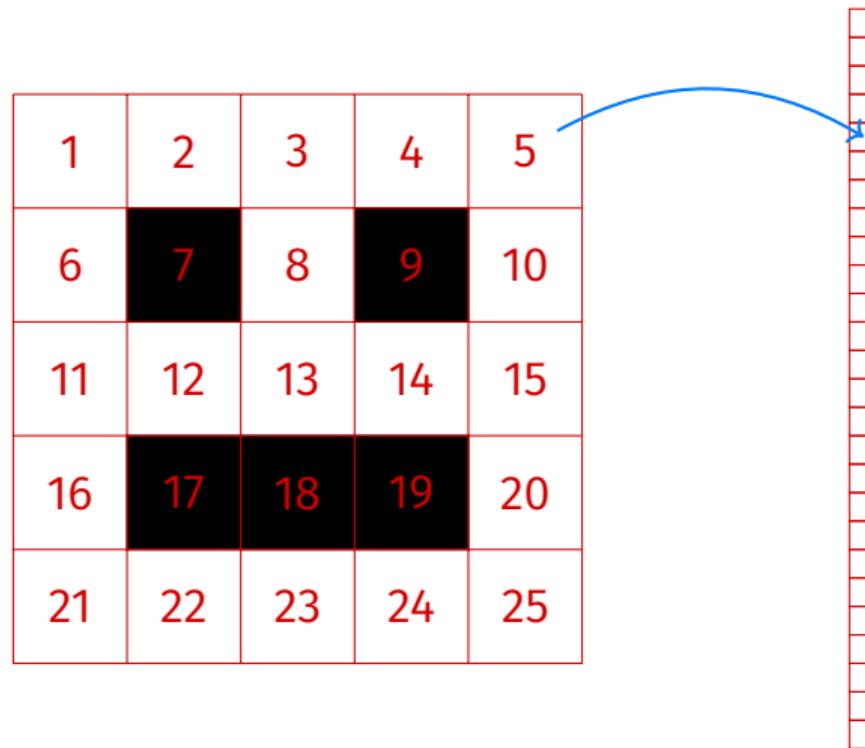
Images as Vectors



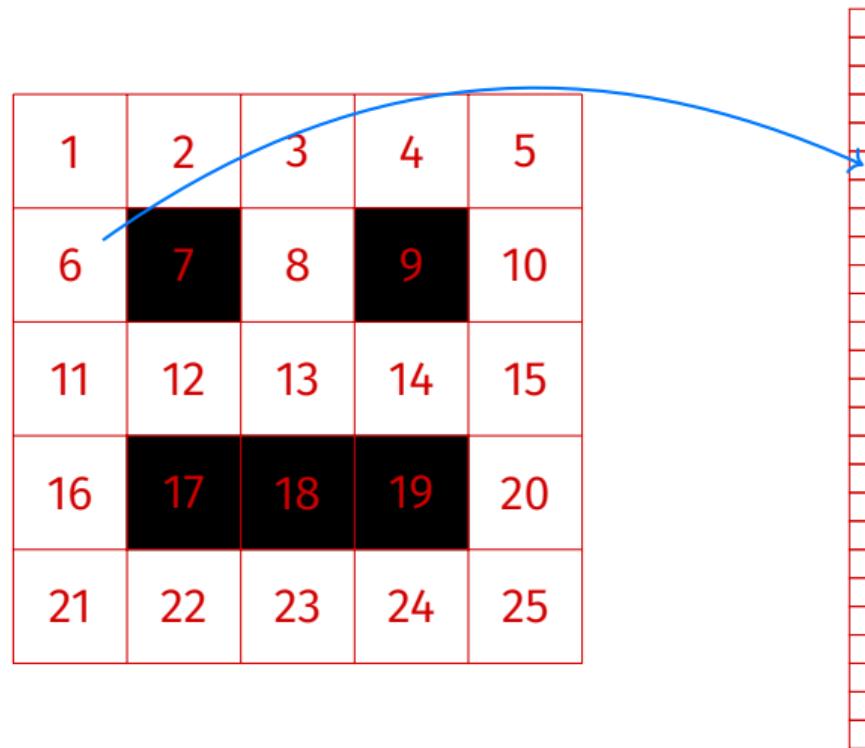
Images as Vectors



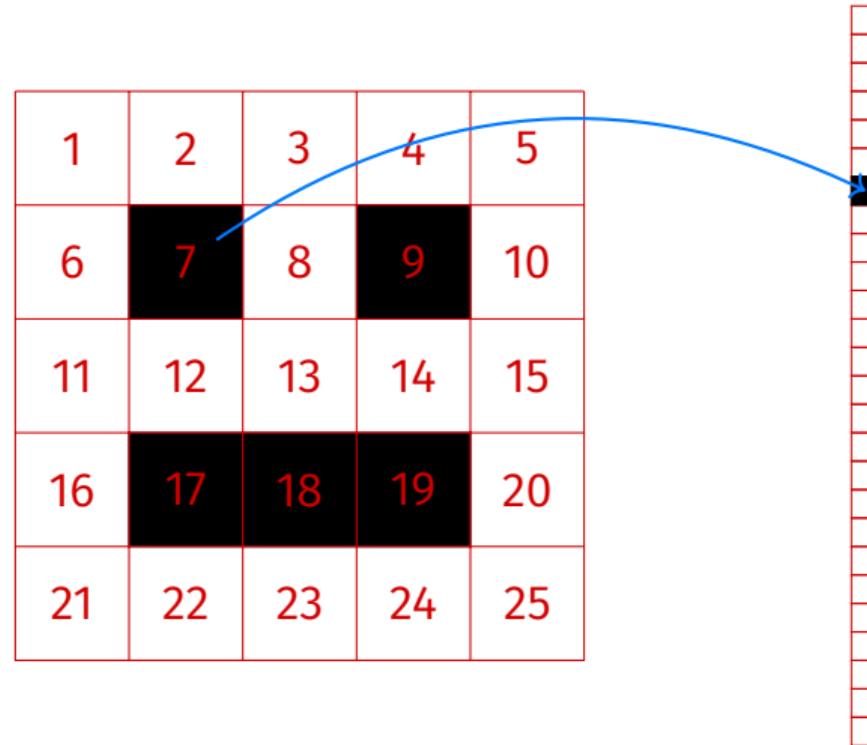
Images as Vectors



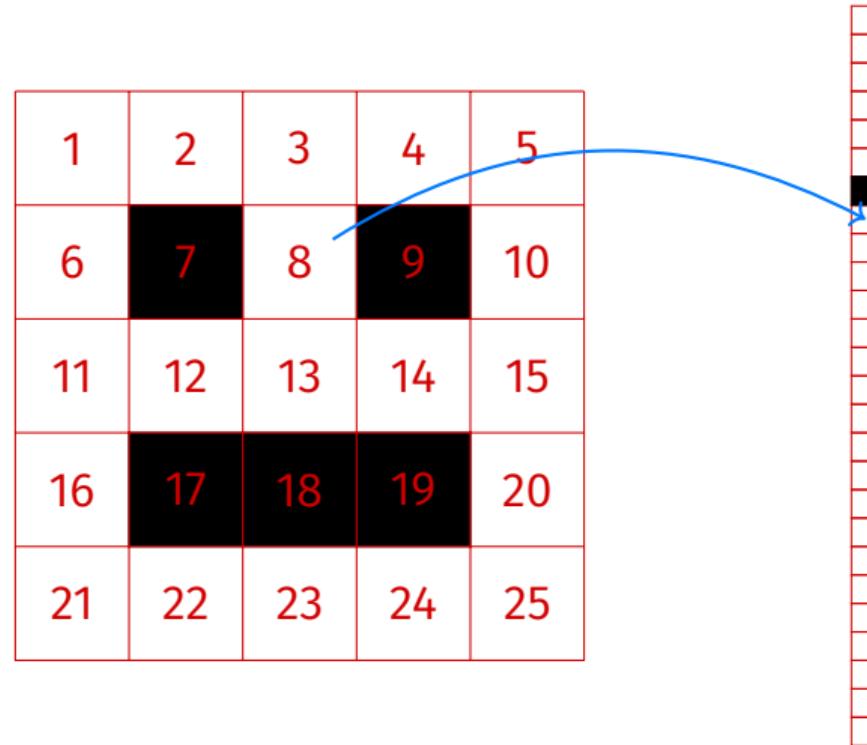
Images as Vectors



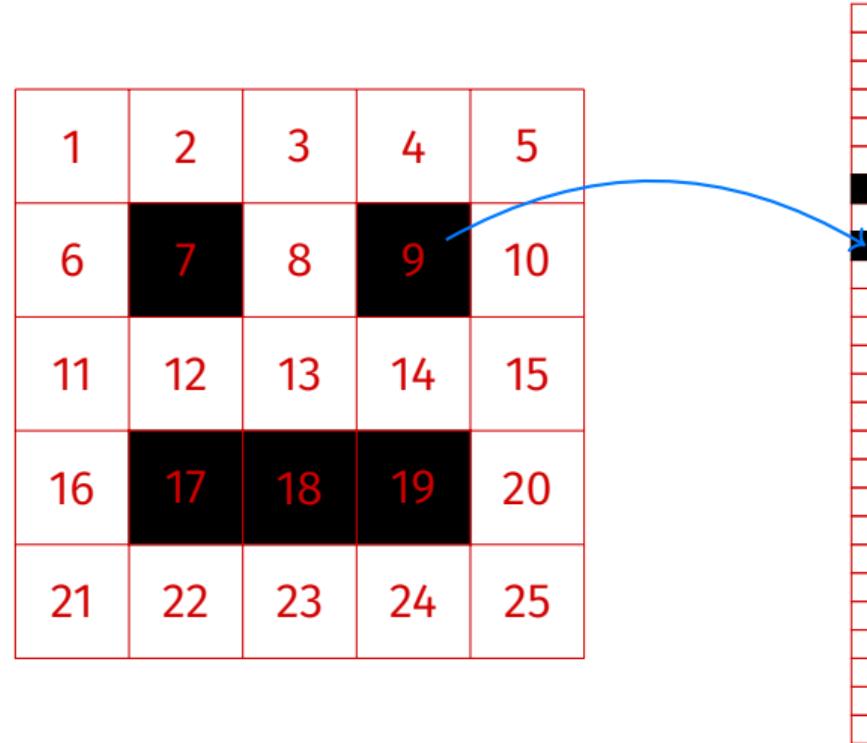
Images as Vectors



Images as Vectors



Images as Vectors



Images as Vectors

| | | | | |
|----|----|----|----|----|
| 1 | 2 | 3 | 4 | 5 |
| 6 | 7 | 8 | 9 | 10 |
| 11 | 12 | 13 | 14 | 15 |
| 16 | 17 | 18 | 19 | 20 |
| 21 | 22 | 23 | 24 | 25 |

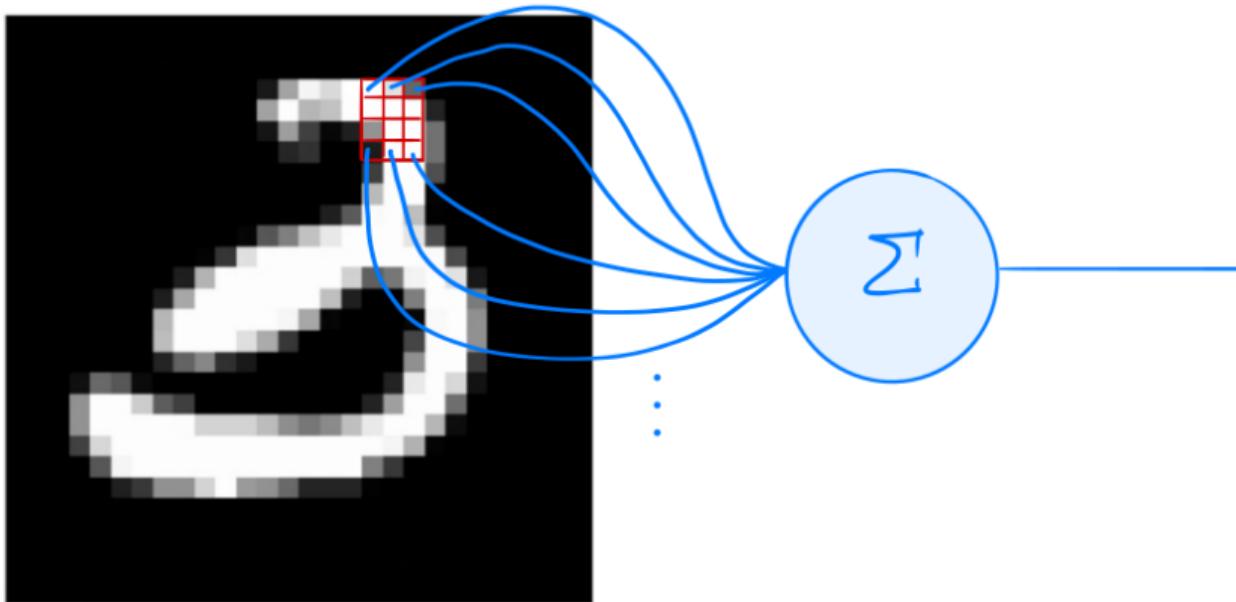


MNIST Feature Vectors

- ▶ $28 \times 28 = 784$ pixels
- ▶ Each image is a vector in \mathbb{R}^{784}
- ▶ Each feature is intensity of single pixel
 - ▶ black → 0, white → 255
- ▶ A **very** simple representation.

Demo: MNIST

- ▶ Use only images of 3s and 7s.
- ▶ 4132 training images.
- ▶ 680 testing images.
- ▶ Some minor tuning.
 - ▶ Added random noise for robustness.
 - ▶ Picked classification threshold automatically.



Perceptron Learning

- ▶ Linear prediction function parameterized by \vec{w} .
- ▶ In this case, we can “reshape” \vec{w} to be same size as input image.

Weight Vector

- ▶ Recall that the prediction is a **weighted vote**:

$$H(\vec{x}) = \text{sign}(w_0 + w_1 x_1 + w_2 x_2 + \dots + w_{784} x_{784})$$

- ▶ Positive → 7, Negative → 3
- ▶ w_i is the weight of pixel i
 - ▶ positive: if this pixel is bright, I think this is a 7
 - ▶ negative: if this pixel is bright, I think this is a 3
 - ▶ magnitude: confidence in prediction

Perceptron Training



Perceptron Training



Perceptron Training



Perceptron Training



Perceptron Training

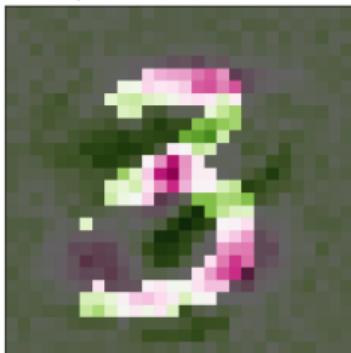


Perceptron Training

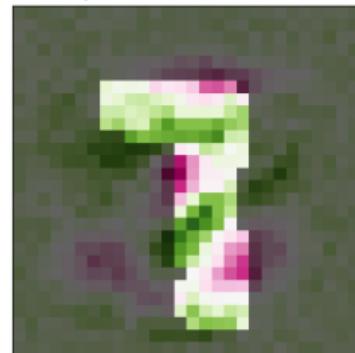


Perceptron Weight Vector

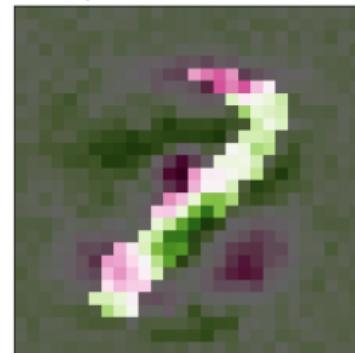
I predict that this is a 3!



I predict that this is a 7!



I predict that this is a 3!



Perceptron Results

- ▶ Test accuracy: 97.3%

Square Loss for Classification

- ▶ What if we use square loss for classification?
- ▶ We *can*, but will it work well?

Results: Least Squares

- ▶ Test Accuracy: 96.7% (marginally worse)

I think that this is a 3.



I think that this is a 7.



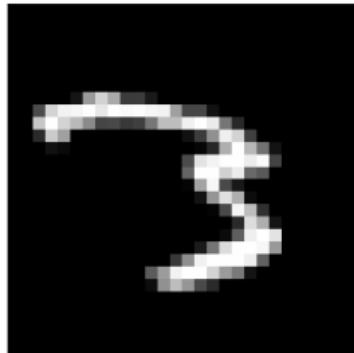
I think that this is a 7.



Results: Least Squares

- ▶ Misclassifications are telling.

I think that this is a 7.



I think that this is a 7.

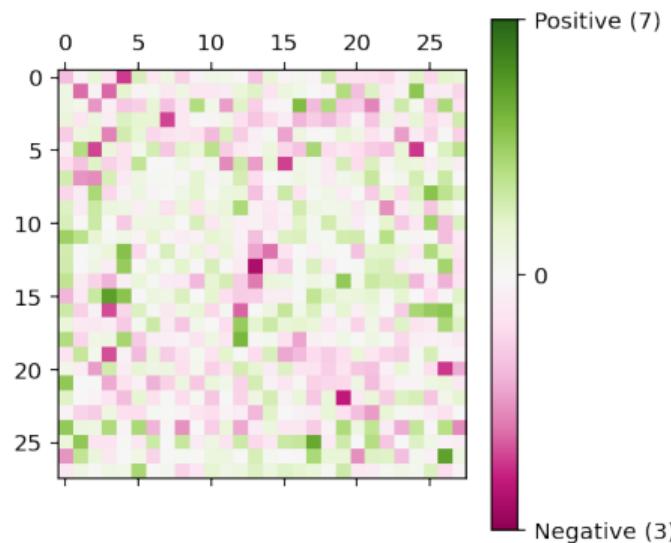


I think that this is a 7.



Least Squares Weight Vector

- ▶ Can visualize weight of each pixel as an image.

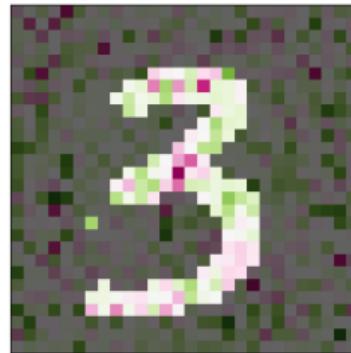


Least Squares Weight Vector

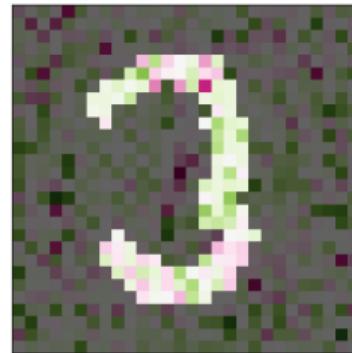
I predict that this is a 7!



I predict that this is a 3!



I predict that this is a 7!

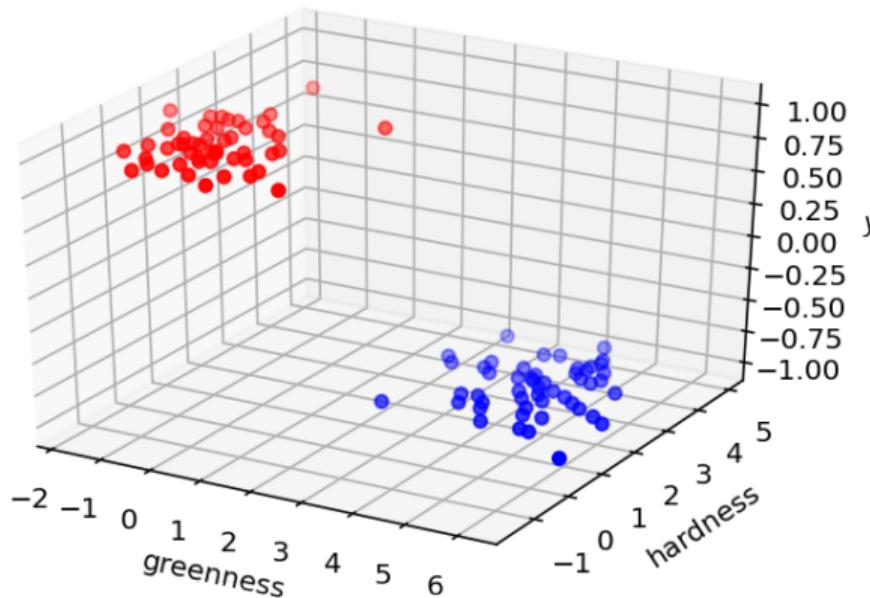


Hack

- ▶ Idea: binary classification is just a special case of regression, where output is either -1 or 1.
- ▶ We can still minimize mean squared error.
- ▶ Somewhat strange: penalizes “very correct” predictions.

$$y_i = 1 \quad H(\vec{x}^{(i)}) = 3 \quad H(\vec{x}^{(i)}) = -1$$

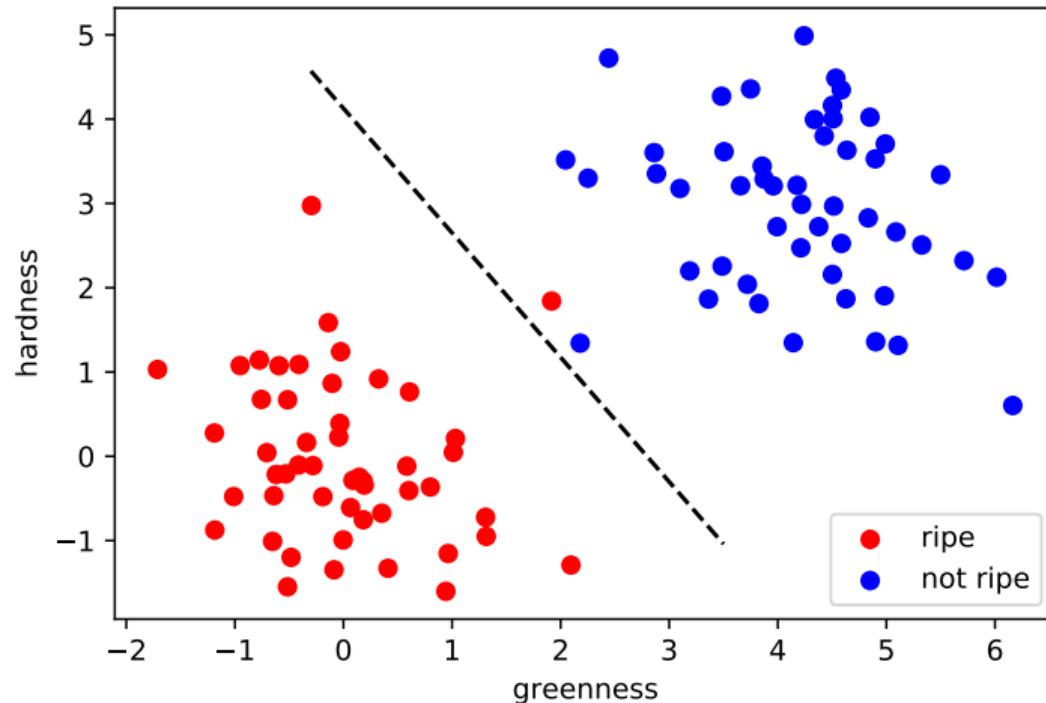
Classification as Regression



```
>>> # assume X is augmented design matrix
>>> # y is array of labels (1 = ripe, -1 = not ripe)
>>> w = np.linalg.lstsq(X, y)
>>> w
array([ 0.78442482, -0.28064357, -0.19005338])
```

That is,

$$H(\vec{x}) = 0.78 - 0.28 \times \text{greenness} - 0.19 \times \text{hardness}$$



Comments

- ▶ Binary classification in MNIST is **easy**.
 - ▶ Multiclass is a lot harder.
- ▶ If the problem is easy, linear prediction functions work well.
- ▶ And it doesn't matter so much which loss we use.

Next

- ▶ What if the problem isn't so easy?

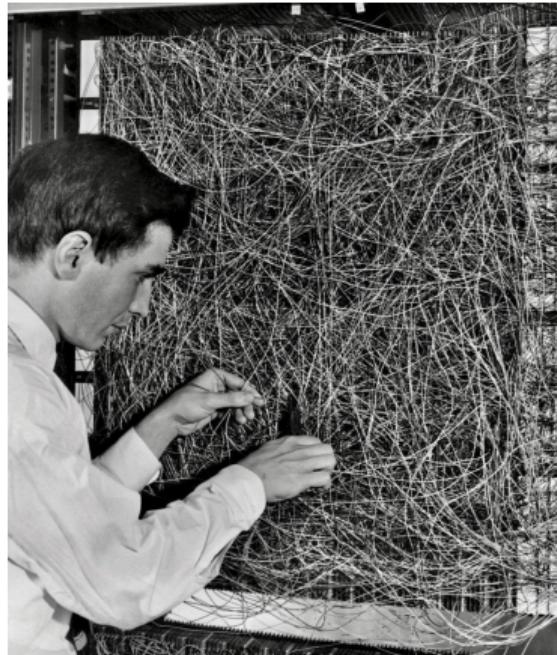
DSC 190

Machine Learning: Representations

Lecture 3 | Part 1

An Embarrassment for the Perceptron

The Perceptron



The Perceptron

- ▶ The perceptron uses a **linear prediction function**:

$$\begin{aligned} H(\vec{x}) &= w_0 + w_1 x_1 + w_2 x_2 + \dots + w_d x_d \\ &= w_0 + \sum_{i=1}^d w_i x_i \\ &= \vec{w} \cdot \text{Aug}(\vec{x}) \end{aligned}$$

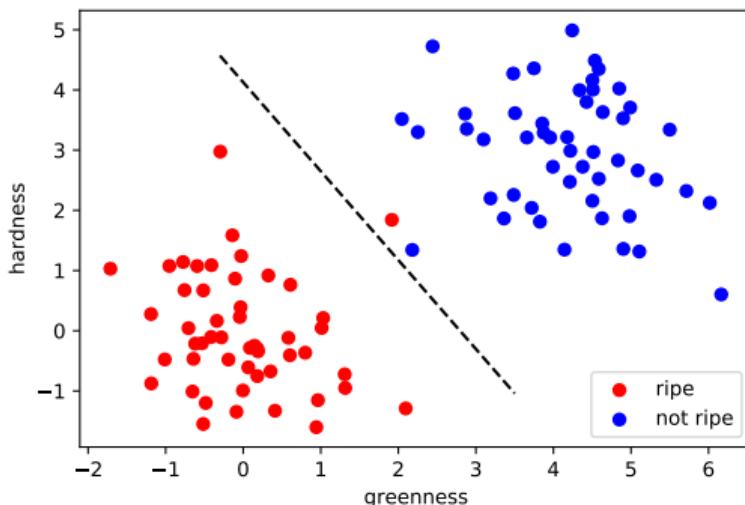
- ▶ Trained using the **perceptron loss**.

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.



NEW NAVY DEVICE LEARNS BY DOING

Psychologist Shows Embryo
of Computer Designed to
Read and Grow Wiser

WASHINGTON, July 7 (UPI) — The Navy revealed the embryo of an electronic computer today that it expects will be able to walk, talk, see, write, reproduce itself and be conscious of its existence.

The embryo—the Weather Bureau's \$2,000,000 "704" computer—learned to differentiate between right and left after fifty attempts in the Navy's demonstration for newsmen.

The service said it would use this principle to build the first of its Perceptron thinking machines that will be able to read and write. It is expected to be finished in about a year at a cost of \$100,000.

Dr. Frank Rosenblatt, designer of the Perceptron, conducted the demonstration. He said the machine would be the first device to think as the human brain. As do human be-

ings, Perceptron will make mistakes at first, but will grow wiser as it gains experience, he said.

Dr. Rosenblatt, a research psychologist at the Cornell Aeronautical Laboratory, Buffalo, said Perceptrons might be fired to the planets as mechanical space explorers.

Without Human Controls

The Navy said the perceptron would be the first non-living mechanism "capable of receiving, recognizing and identifying its surroundings without any human training or control."

The "brain" is designed to remember images and information it has perceived itself. Ordinary computers remember only what is fed into them on punch cards or magnetic tape.

Later Perceptrons will be able to recognize people and call out their names and instantly translate speech in one language to speech or writing in another language, it was predicted.

Mr. Rosenblatt said in principle it would be possible to build brains that could reproduce themselves on an assembly line and which would be conscious of their existence.

1958 New York Times...

In today's demonstration, the "704" was fed two cards, one with squares marked on the left side and the other with squares on the right side.

Learns by Doing

In the first fifty trials, the machine made no distinction between them. It then started registering a "Q" for the left squares and "O" for the right squares.

Dr. Rosenblatt said he could explain why the machine learned only in highly technical terms. But he said the computer had undergone a "self-induced change in the wiring diagram."

The first Perceptron will have about 1,000 electronic "association cells" receiving electrical impulses from an eye-like scanning device with 400 photo-cells. The human brain has 10,000,000,000 responsive cells, including 100,000,000 connections with the eyes.

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

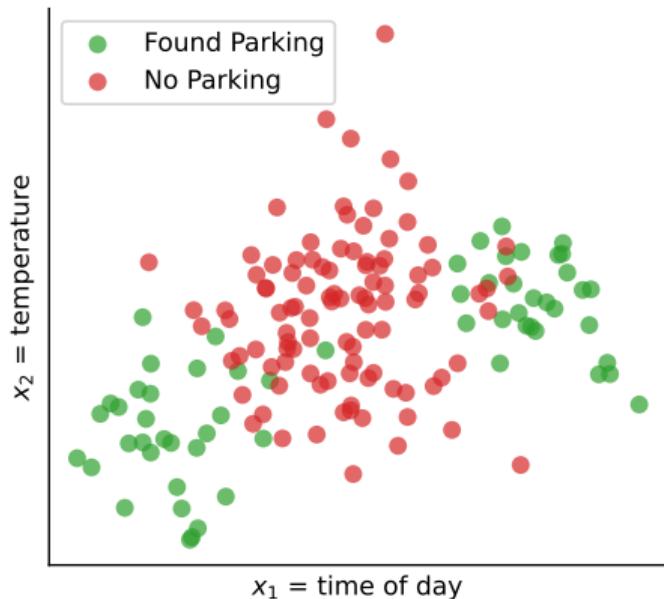
Imagine a scatter plot of the training data with the two features:

- ▶ x_1 = time of day
- ▶ x_2 = temperature

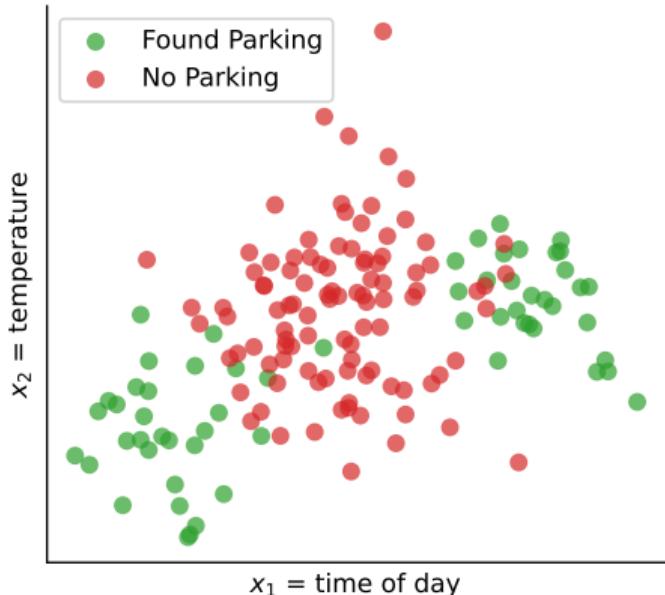
“yes” examples are green, “no” are red.

What does it look like?

Parking Data



Uh oh



- ▶ A linear decision function won't work.
- ▶ A perceptron (or linear SVM, logistic regression model, etc.) won't capture the trend
- ▶ What do we do?

Today's Question

- ▶ How do we learn non-linear patterns using linear prediction functions?

DSC 190

Machine Learning: Representations

Lecture 3 | Part 2

Basis Functions

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.

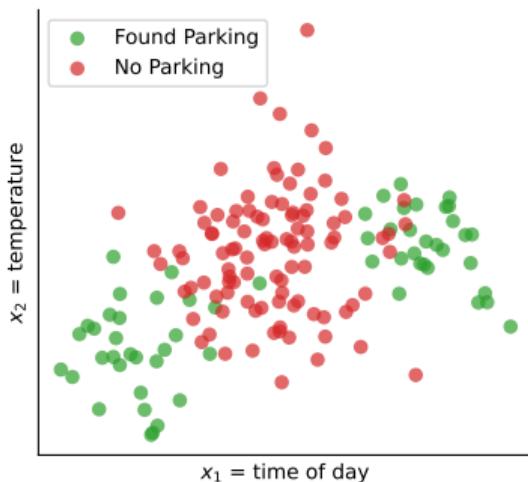
Exercise

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:
 - ▶ the time, x_1 , to $|x_1 - \text{Noon}|$
 - ▶ 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 $\mathbb{R}^d \rightarrow \mathbb{R}$

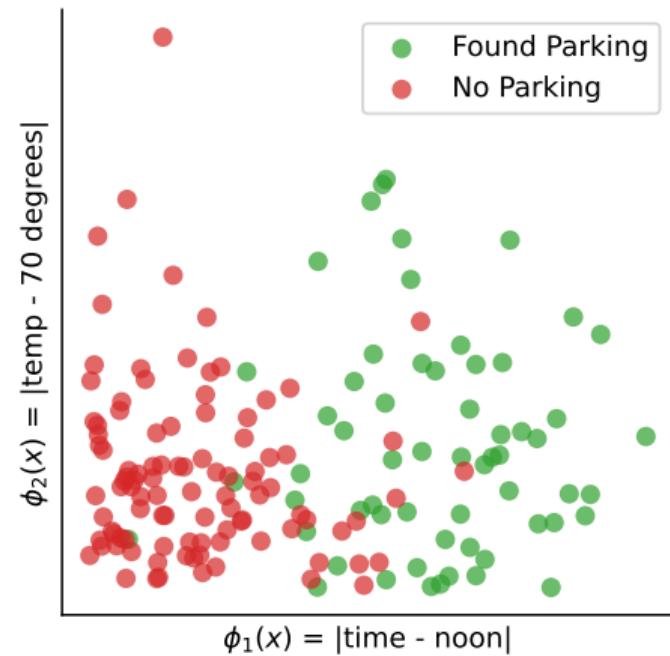
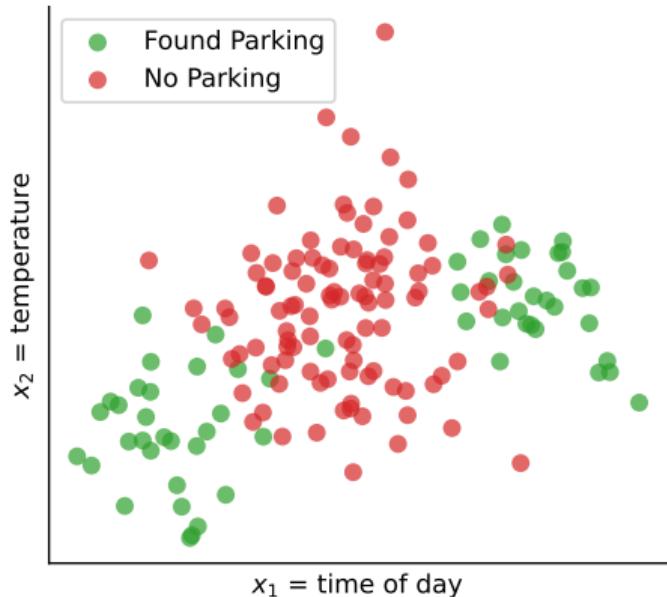
Feature Mapping

- ▶ 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:

$$\vec{\varphi}((10\text{a.m.}, 75^\circ)^T) = (2 \text{ hours}, 3^\circ)^T$$

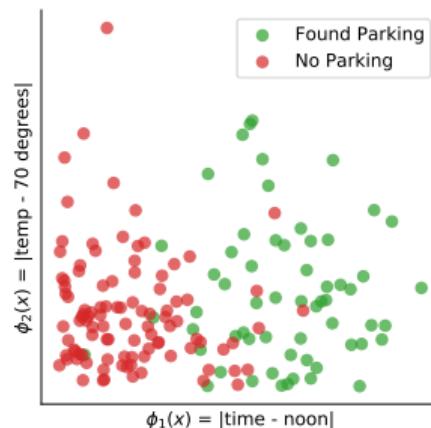
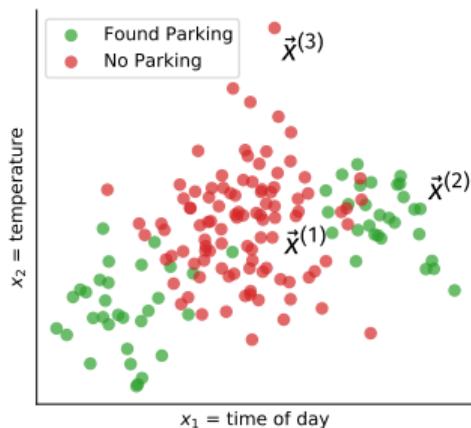
- ▶ $\vec{\varphi}$ maps raw data to a **feature space**.

Feature Space, Visualized

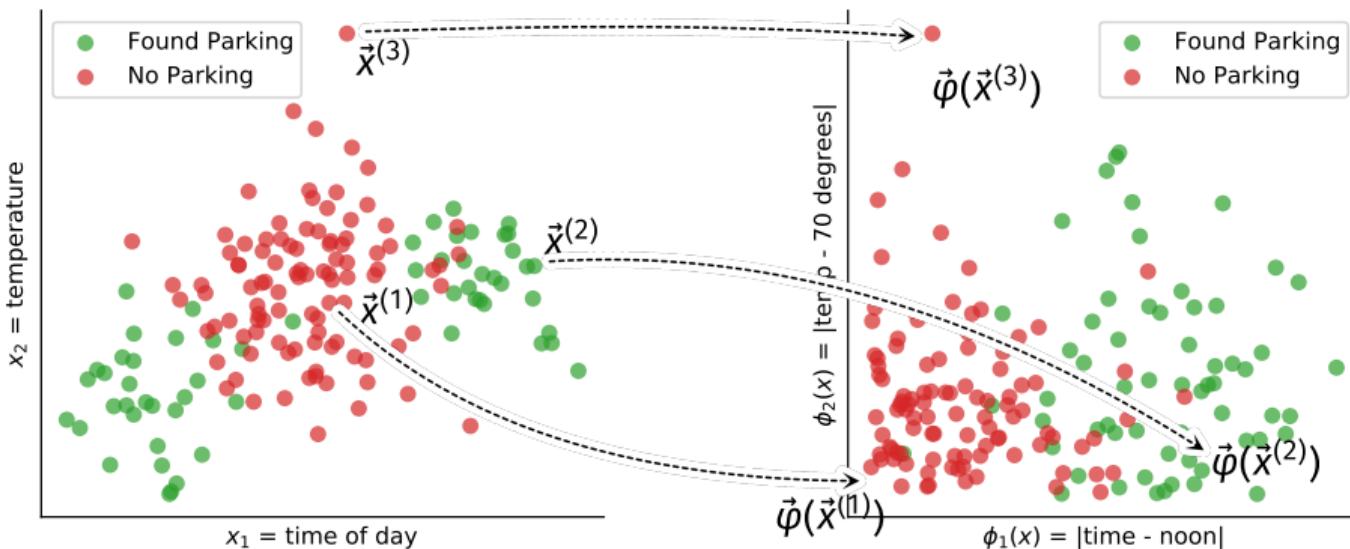


Exercise

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

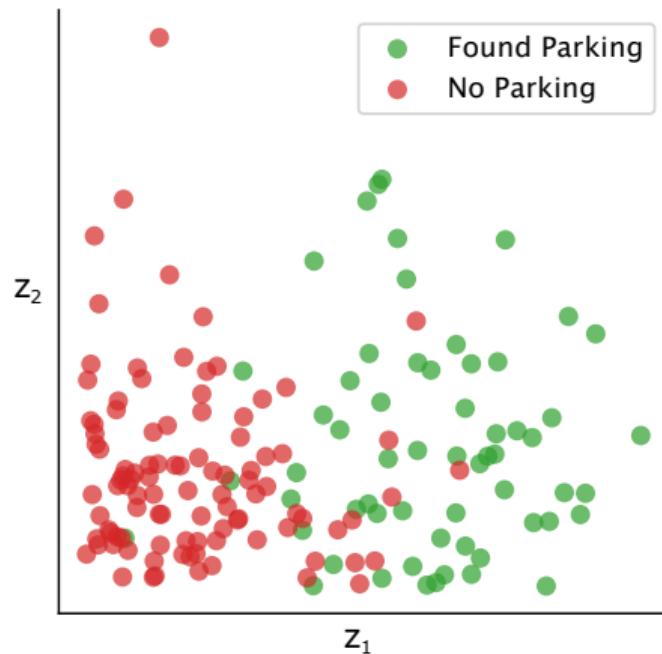
- ▶ Map each training example $\vec{x}^{(i)}$ to feature space, creating new training data:

$$\vec{z}^{(1)} = \vec{\varphi}(\vec{x}^{(1)}), \quad \vec{z}^{(2)} = \vec{\varphi}(\vec{x}^{(2)}), \quad \dots, \quad \vec{z}^{(n)} = \vec{\varphi}(\vec{x}^{(n)})$$

- ▶ 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:

$$\begin{aligned}H(\vec{x}) &= H_f(\vec{\varphi}(\vec{x})) \\&= H_f((\varphi_1(\vec{x}), \varphi_2(\vec{x}), \dots, \varphi_d(\vec{x}))^T) \\&= w_0 + w_1 \varphi_1(\vec{x}) + w_2 \varphi_2(\vec{x}) + \dots + w_d \varphi_d(\vec{x})\end{aligned}$$

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:

$$\vec{\varphi}(\vec{x}) = (\varphi_1(\vec{x}), \varphi_2(\vec{x}), \varphi_3(\vec{x}))^T$$

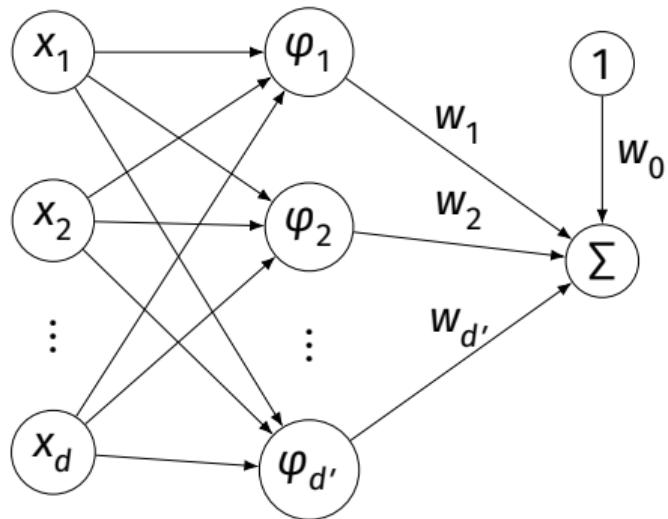
Overview: Feature Mapping

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 190

Machine Learning: Representations

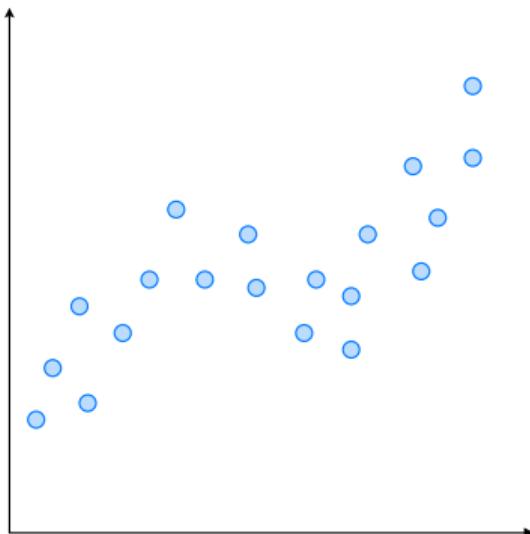
Lecture 3 | Part 3

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



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 $X^T X \vec{w} = X^T \vec{w}$ 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 190

Machine Learning: Representations

Lecture 3 | Part 4

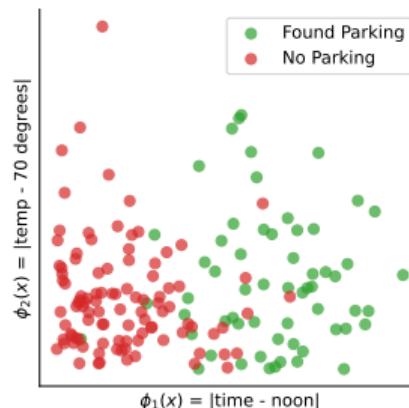
A Tale of Two Spaces

A Tale of Two Spaces

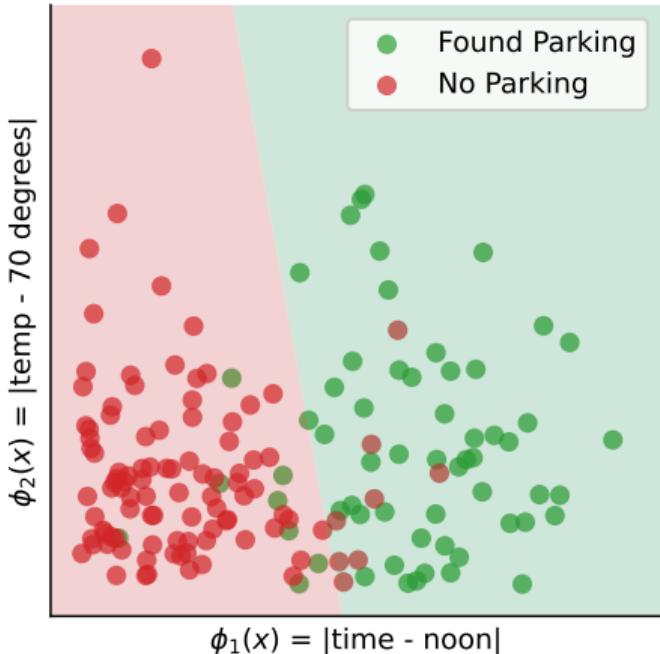
- ▶ The **original space**: where the raw data lies.
- ▶ The **feature space**: where the data lies after feature mapping $\vec{\phi}$
- ▶ Remember: we fit a linear prediction function in the **feature space**.

Exercise

- ▶ In **feature space**, what does the decision boundary look like?
- ▶ What does the prediction function surface look like?

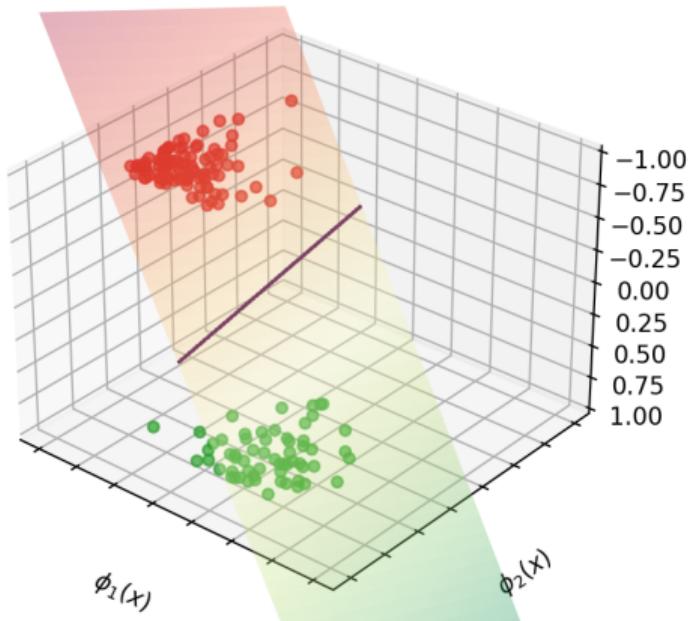


Decision Boundary in Feature Space²



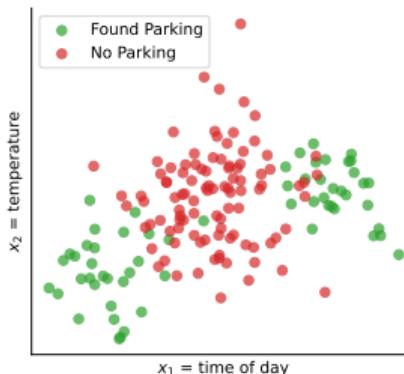
²Fit by minimizing square loss

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?

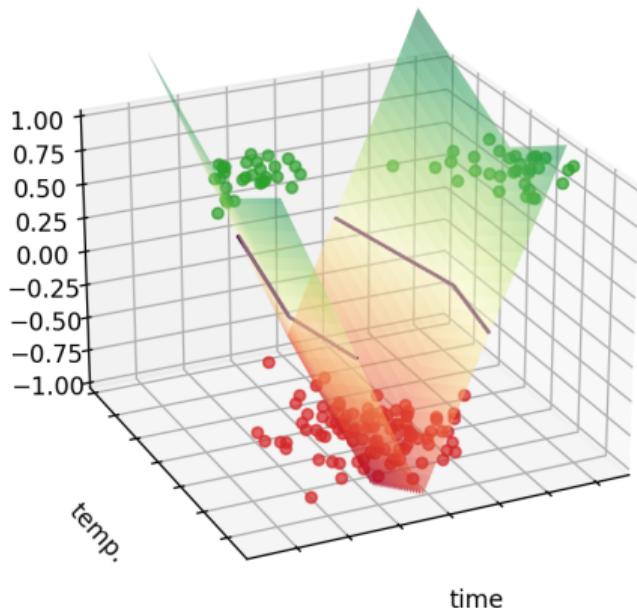


Decision Boundary in Original Space³



³Fit by minimizing square loss

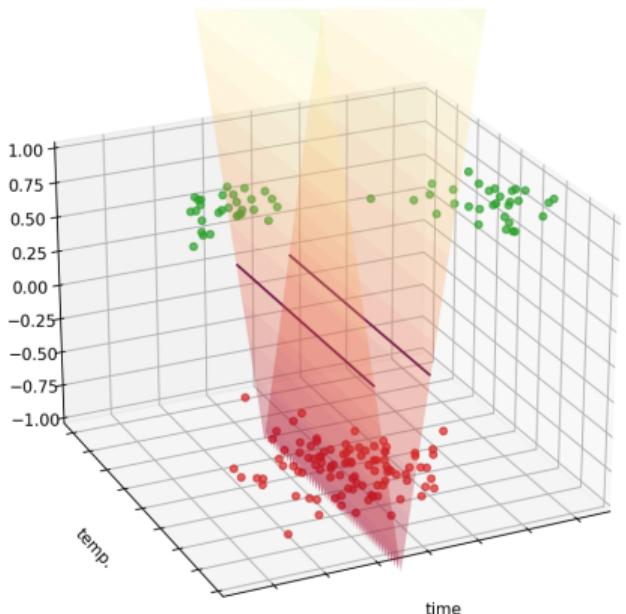
Prediction Surface in Original Space



Insight

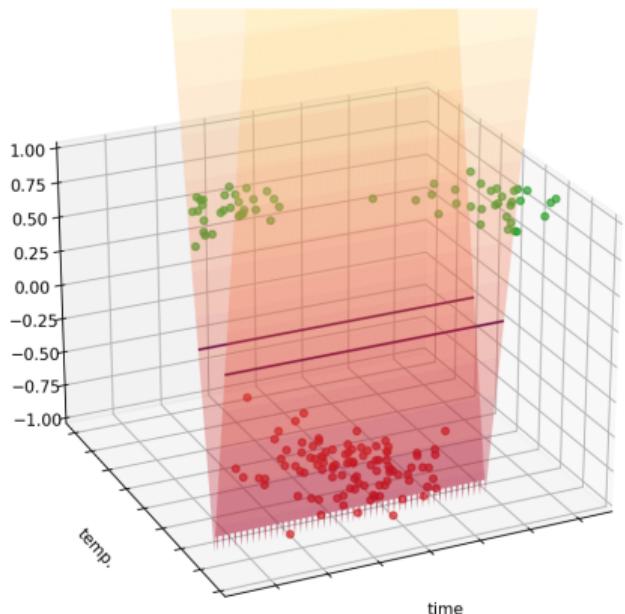
- ▶ H is a sum of basis functions, φ_1 and φ_2 .
 - ▶ $H(\vec{x}) = w_0 + w_1\varphi_1(\vec{x}) + w_2\varphi_2(\vec{x})$
- ▶ The prediction surface is a sum of other surfaces.
- ▶ Each basis function is a “building block”.

Visualizing the Basis Function φ_1



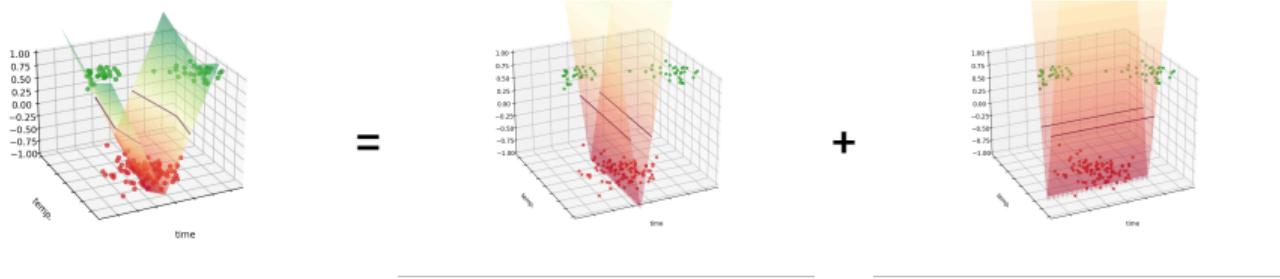
► $w_0 + w_1 |x_1 - \text{noon}|$

Visualizing the Basis Function φ_2



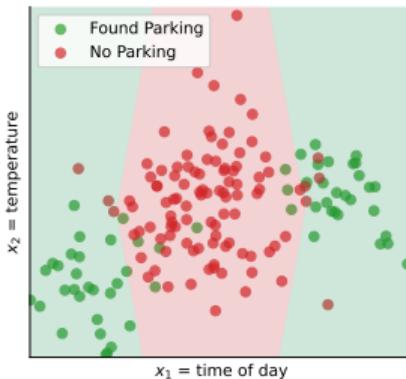
► $w_0 + w_2 |x_2 - 72^\circ|$

Visualizing the Prediction Surface



Exercise

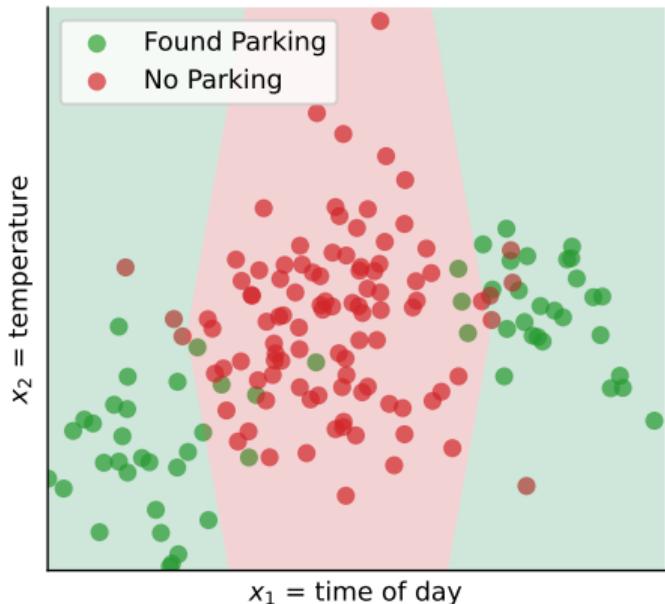
The decision boundary has a single “pocket” where it is negative. Can it have more than one, assuming we use basis functions of the same form? What if we use more than two basis functions?



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...

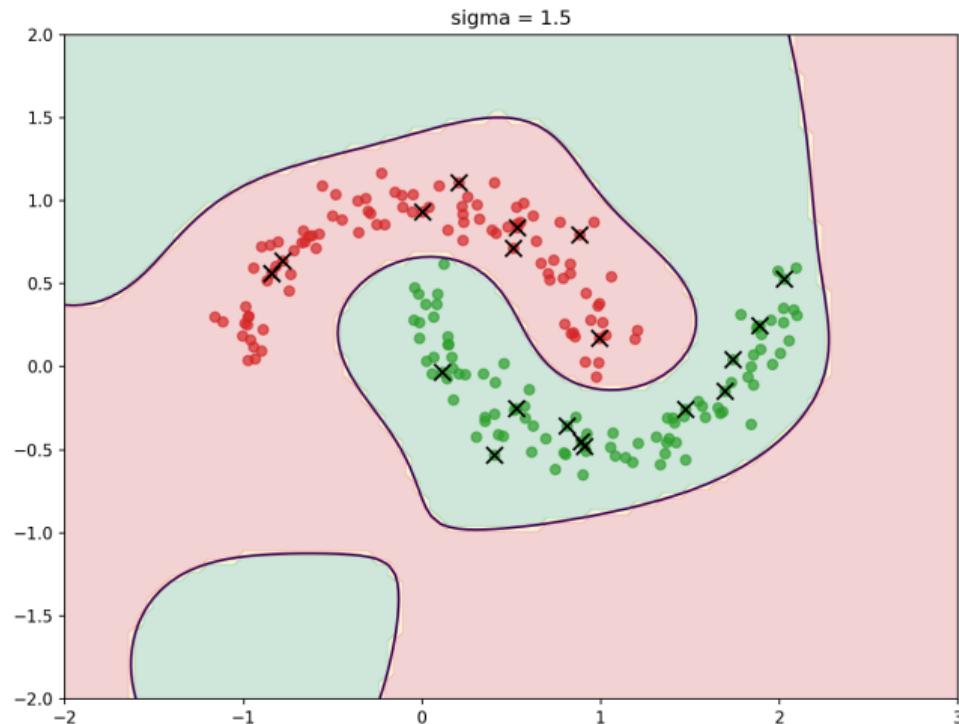
DSC 190

Machine Learning: Representations

Lecture 4 | Part 1

Radial Basis Functions

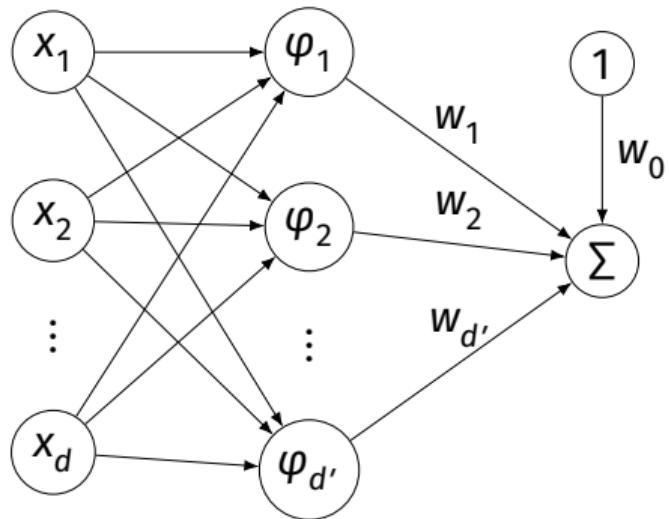
Today



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**.

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



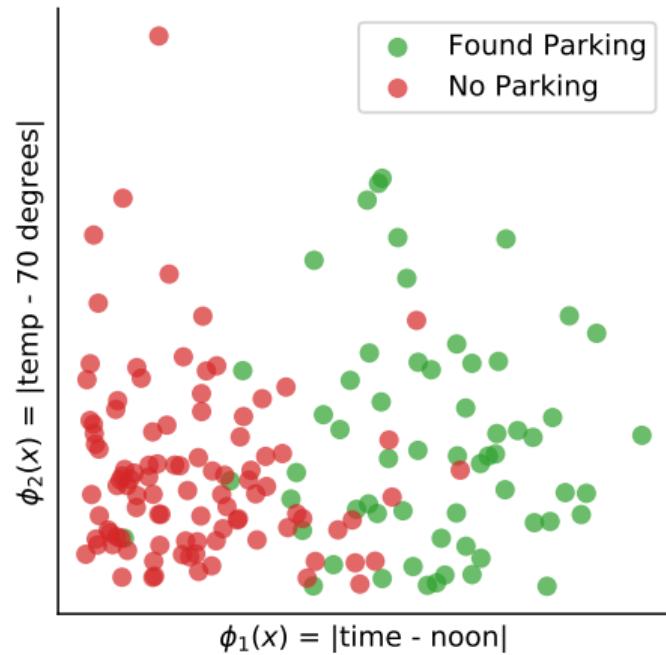
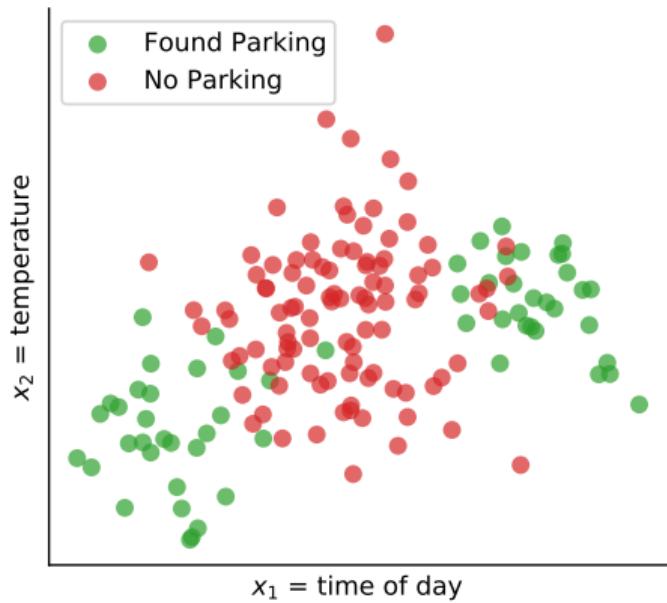
Overview: Feature Mapping

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

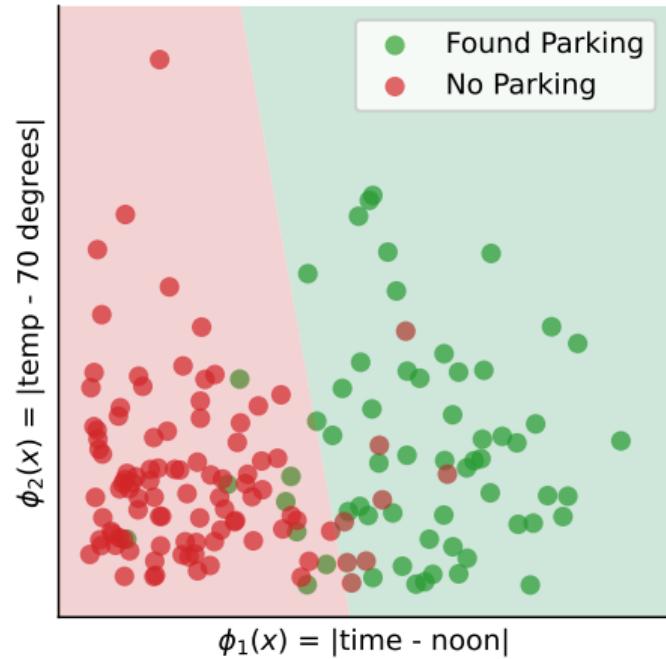
4. Fit linear prediction function in new space:

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

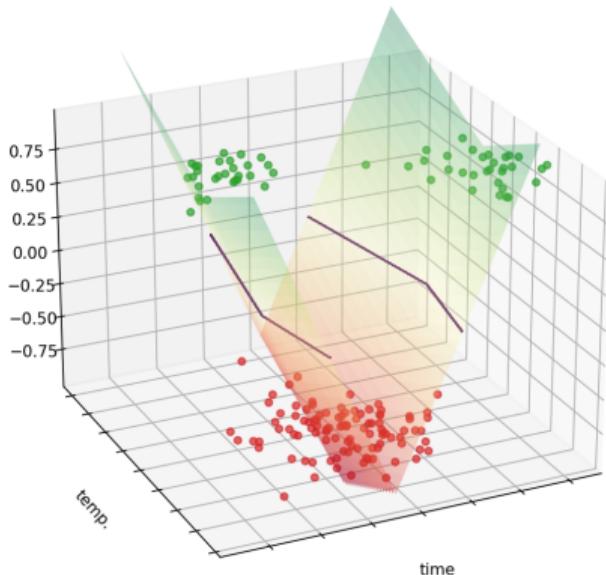
Last Time



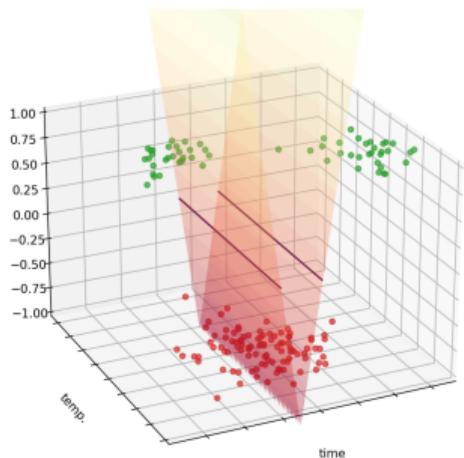
Last Time



Visualizing the “Prediction Surface”

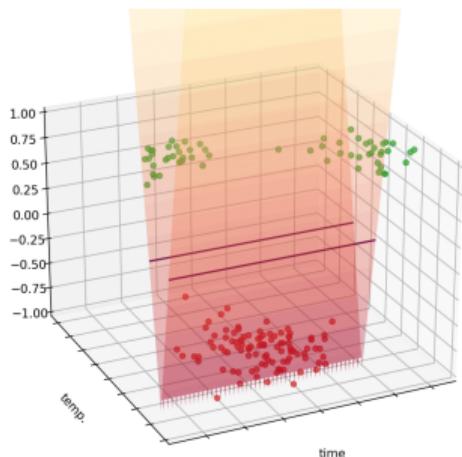


Visualizing the Basis Function φ_1



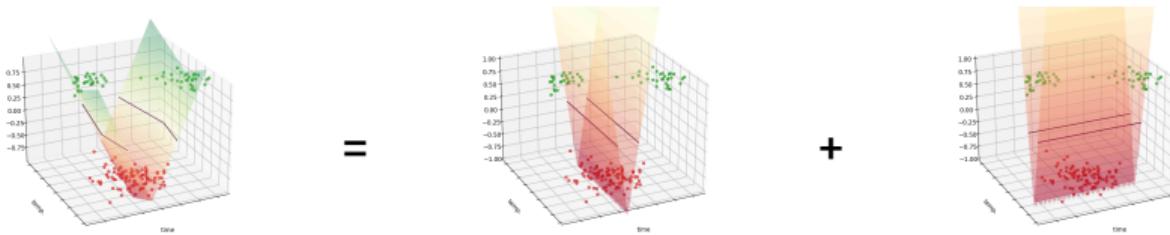
► $w_0 + w_1 |x_1 - \text{noon}|$

Visualizing the Basis Function φ_2



► $w_0 + w_2 |x_2 - 72^\circ|$

Visualizing the “Prediction Surface”



The Decision Boundary

- ▶ The prediction surface is a sum of other surfaces.
- ▶ Each basis function is a “building block”.
- ▶ The **decision boundary** is where surface = zero.

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?



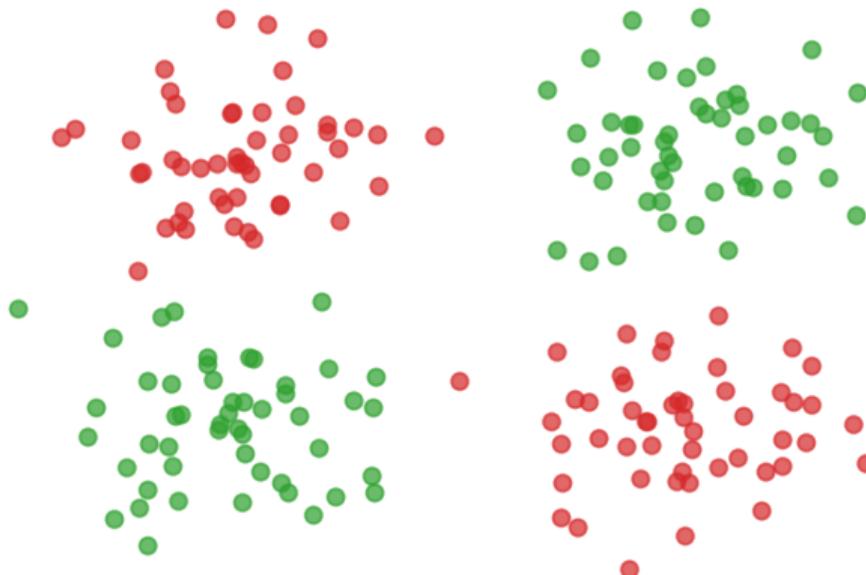
Answer: No!

- ▶ 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.

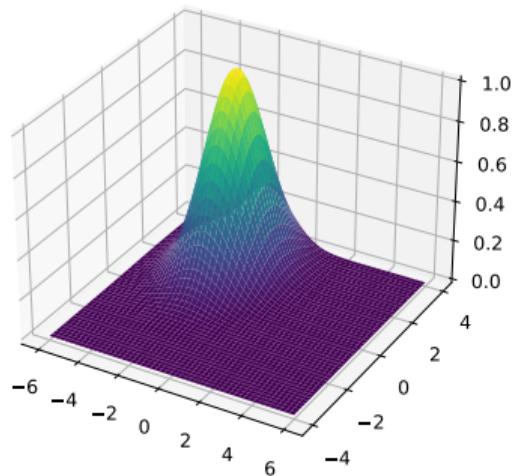
Choosing Basis Functions

- ▶ 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:

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

- ▶ $\vec{\mu}$ 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

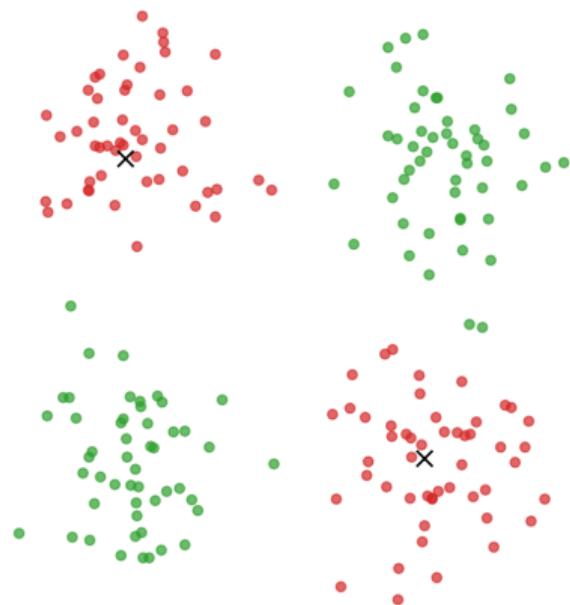
- ▶ For any feature vector $\vec{x} \in \mathbb{R}^d$, map to vector $\vec{\varphi}(\vec{x}) \in \mathbb{R}^{d'}$.
 - ▶ φ_1 : “similarity” of \vec{x} to $\vec{\mu}^{(1)}$
 - ▶ φ_2 : “similarity” of \vec{x} to $\vec{\mu}^{(2)}$
 - ▶ ...
 - ▶ $\varphi_{d'}$: “similarity” of \vec{x} to $\vec{\mu}^{(d')}$
- ▶ Train linear classifier in this new representation.
 - ▶ E.g., by minimizing expected square loss.

Exercise

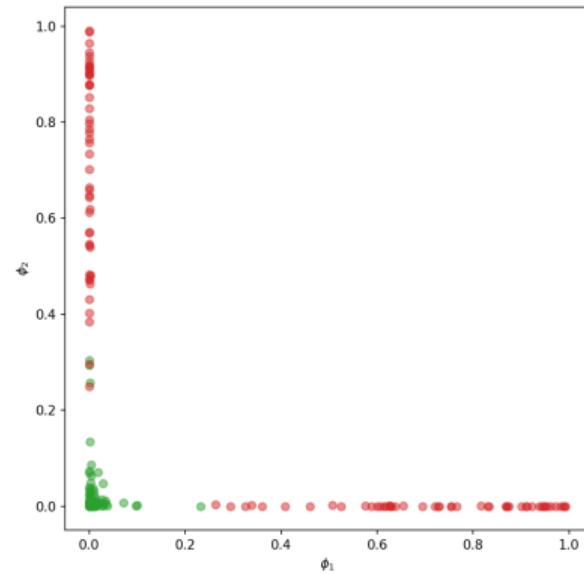
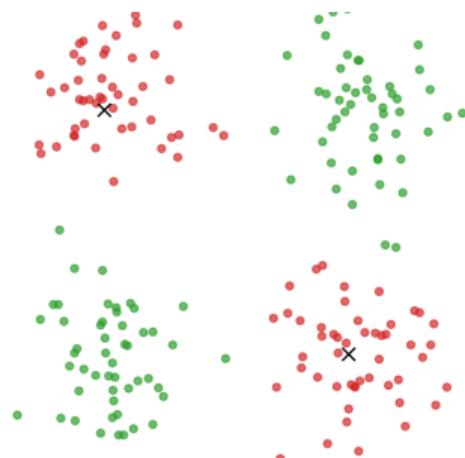
How many Gaussian basis functions would you use, and where would you place them to create a new representation for this data?



Placement



Feature Space



Prediction Function

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

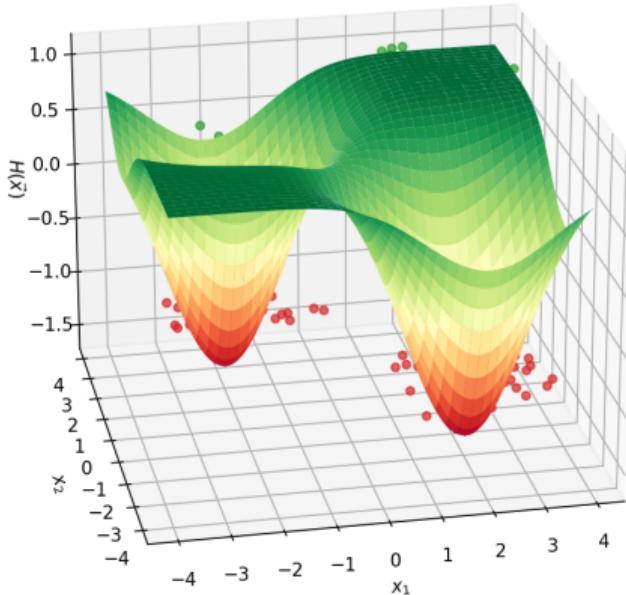
$$\begin{aligned} H(\vec{x}) &= w_0 + w_1 \varphi_1(\vec{x}) + w_2 \varphi_2(\vec{x}) + \dots \\ &= w_0 + w_1 e^{-\|\vec{x}-\vec{\mu}_1\|^2/\sigma^2} + w_2 e^{-\|\vec{x}-\vec{\mu}_2\|^2/\sigma^2} + \dots \end{aligned}$$

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

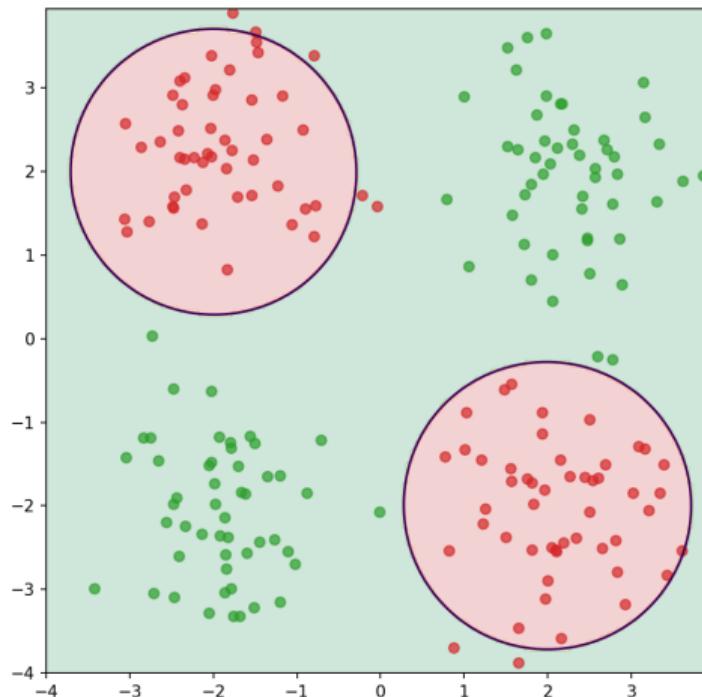


$$H(\vec{x}) = w_0 + w_1 e^{-\|\vec{x} - \vec{\mu}_1\|^2 / \sigma^2} + w_2 e^{-\|\vec{x} - \vec{\mu}_2\|^2 / \sigma^2}$$

An Interpretation

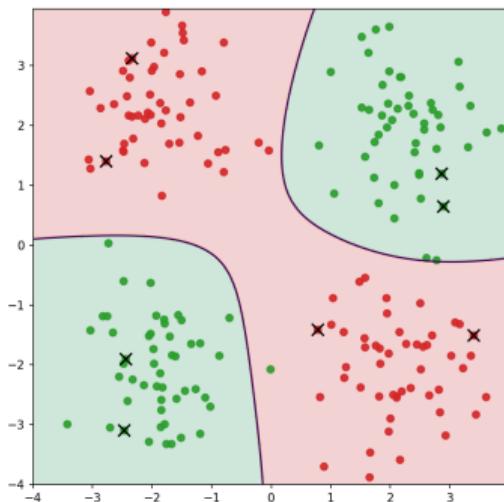
- ▶ Basis function φ_i makes a “bump” in surface of H
- ▶ w_i adjusts the “prominance” of this bump

Decision Boundary

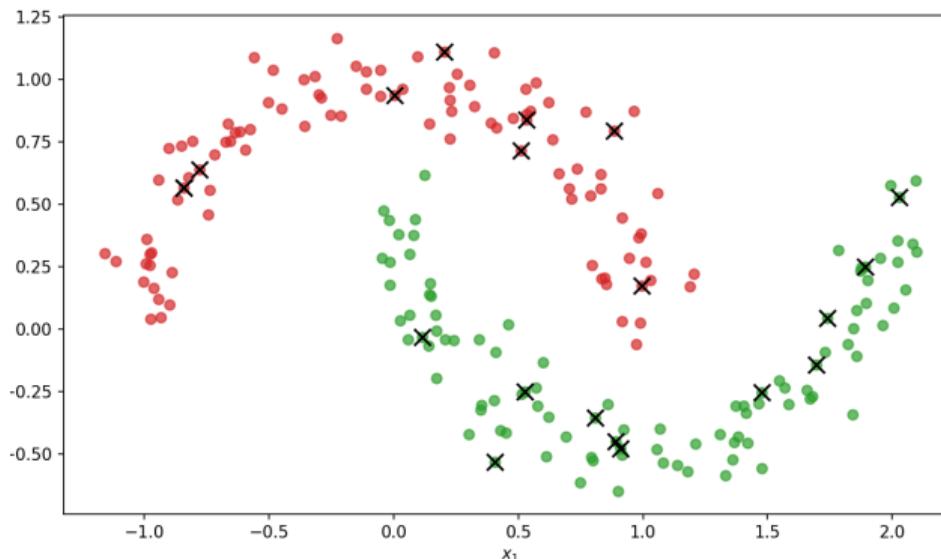


More Features

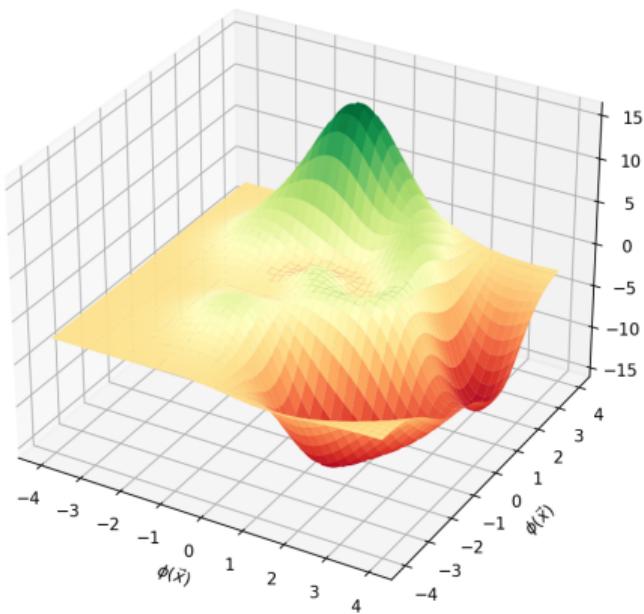
- ▶ By increasing number of basis functions, we can make more complex decision surfaces.



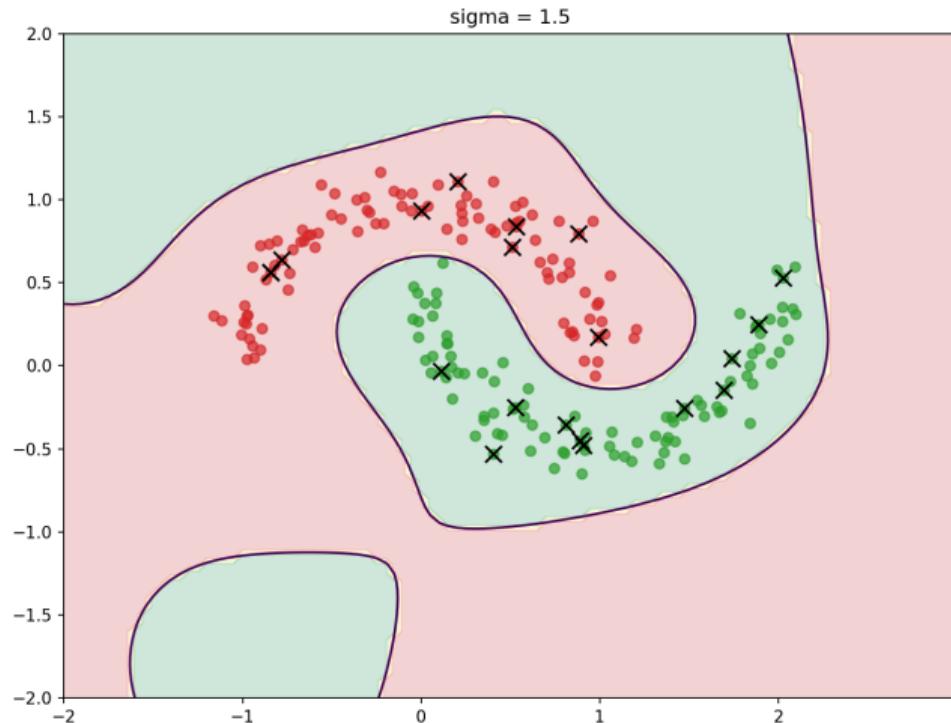
Another Example



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 190

Machine Learning: Representations

Lecture 4 | Part 2

Radial Basis Function Networks

Recap

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

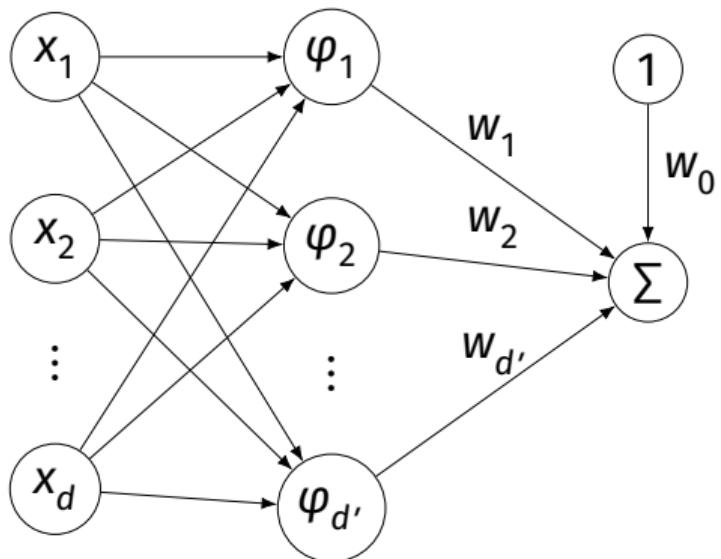
$$\vec{x} \mapsto (\varphi_1(\vec{x}), \varphi_2(\vec{x}), \dots, \varphi_{d'}(\vec{x}))^T$$

3. Train a linear classifier in this new space:

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

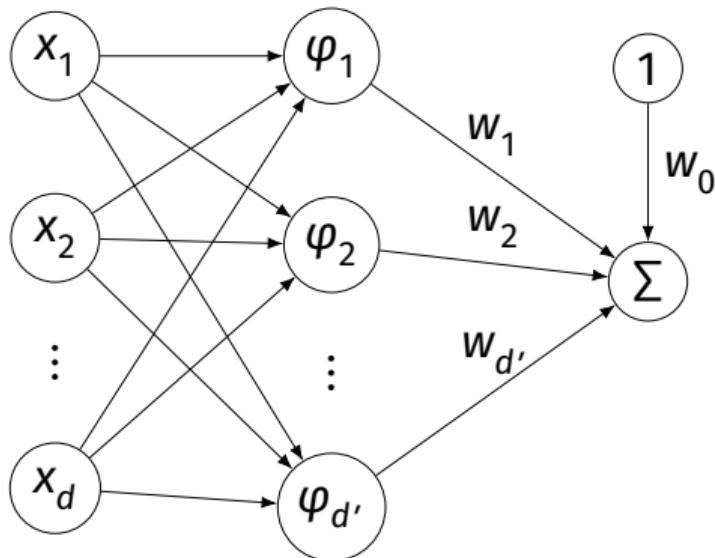
The Model

- 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**.
- ▶ It is a simple type of neural network.

Training

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

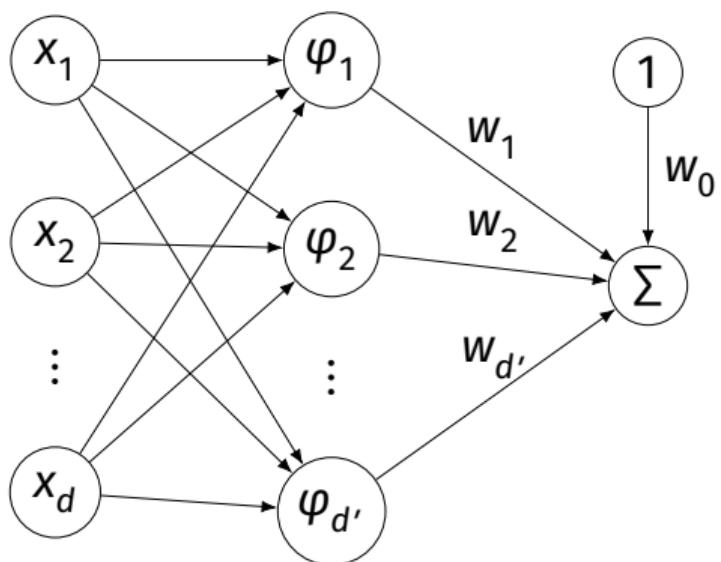
Minimizing Expected Loss

- ▶ As with most any model, we can try to find parameters by minimizing expected loss.
- ▶ However, now the risk is a complex, non-linear function of many things:

$$R(\vec{w}, \vec{\mu}_1, \dots, \vec{\mu}_{d'}, \sigma, \dots).$$

- ▶ As opposed to a simple linear model: $R(\vec{w})$.

Training



- ▶ Optimization is now much harder.
- ▶ Instead, we **decouple**:
 1. Find basis function parameters in some way, consider them fixed.
 2. Now train \vec{w} by minimizing risk

Theory

- ▶ Given suitably-many basis functions, a Gaussian RBF is capable of approximating any continuous function arbitrarily well.

DSC 190

Machine Learning: Representations

Lecture 4 | Part 3

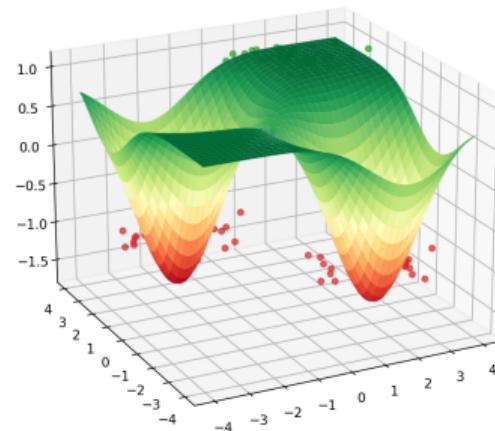
Choosing RBF Locations

Recap

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

Prediction Function

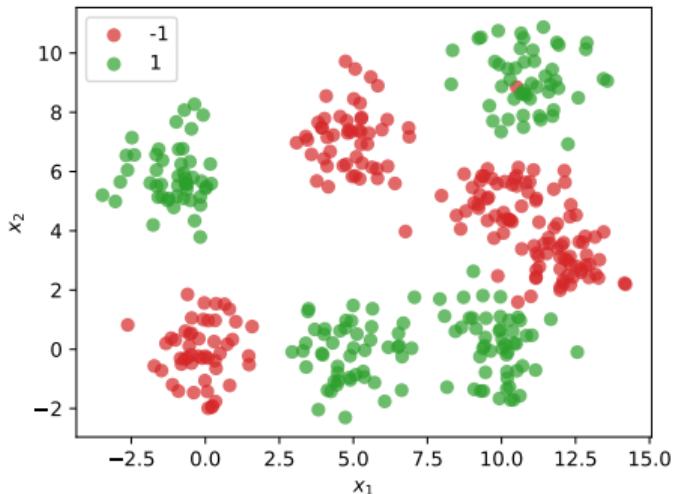
- ▶ Our prediction function H is a surface that is made up of Gaussian “bumps”.



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

Choosing Centers

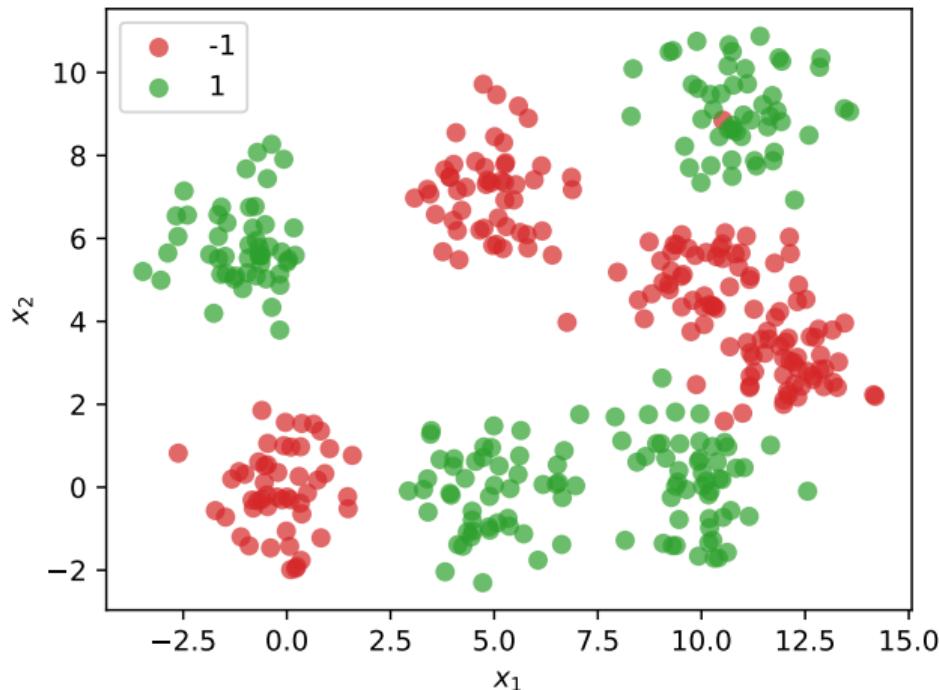
- ▶ Place the centers where the value of the prediction function should be controlled.
- ▶ Intuitively: place centers where the data is.



Approaches

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

Approach #1: Every Data Point as a Center



Dimensionality

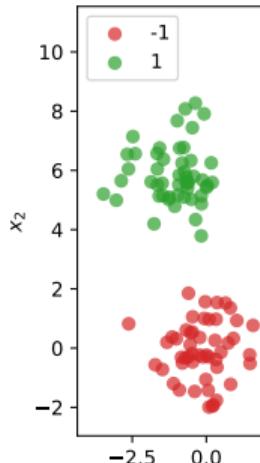
- ▶ We'll have n basis functions – one for each point.
- ▶ That means we'll have n features.
- ▶ Each feature vector $\vec{\phi}(\vec{x}) \in \mathbb{R}^n$.

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

Problems

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

^aHowever, this is very doable with SVMs

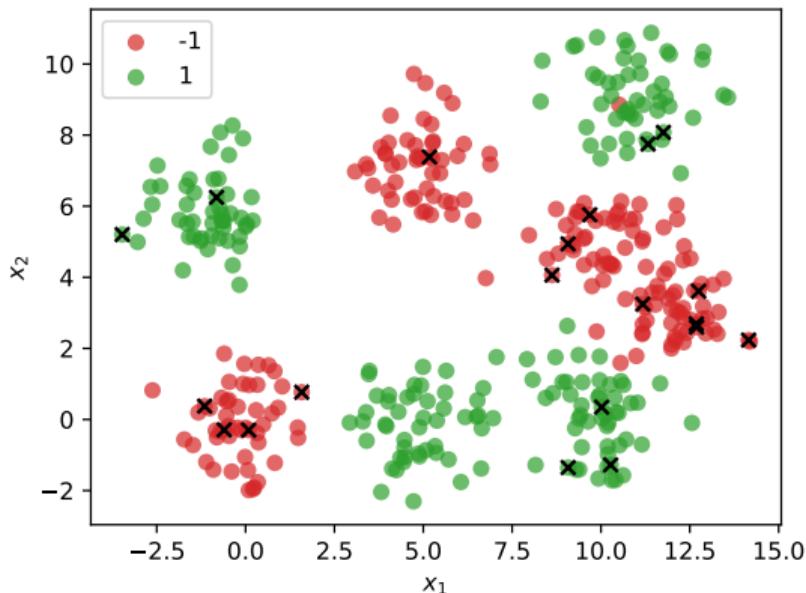


Computational Cost

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

Approach #2: A Random Sample

- Idea: randomly choose k data points as centers.



Problem

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

Approach #3: Clustering

- ▶ Group data points into **clusters**.
- ▶ Cluster centers are good places for RBFs.
- ▶ We'll use k -means clustering to pick k centers.

DSC 190

Machine Learning: Representations

Lecture 5 | Part 1

Choosing RBF Locations

Recap

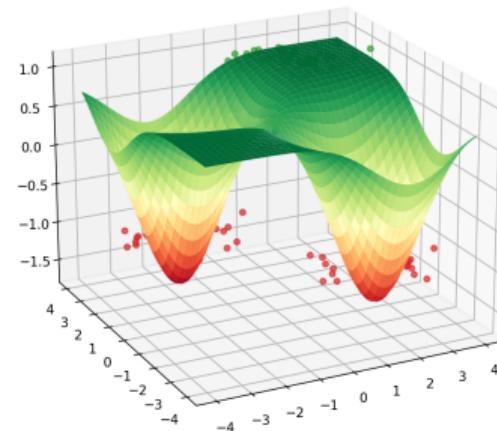
- ▶ We map data to a new representation by first choosing **basis functions**.
- ▶ Radial Basis Functions (RBFs), such as Gaussians, are a popular choice.
- ▶ Requires choosing **center** for each basis function.

Today's Lecture

- ▶ How do we choose basis function centers automatically?

Prediction Function

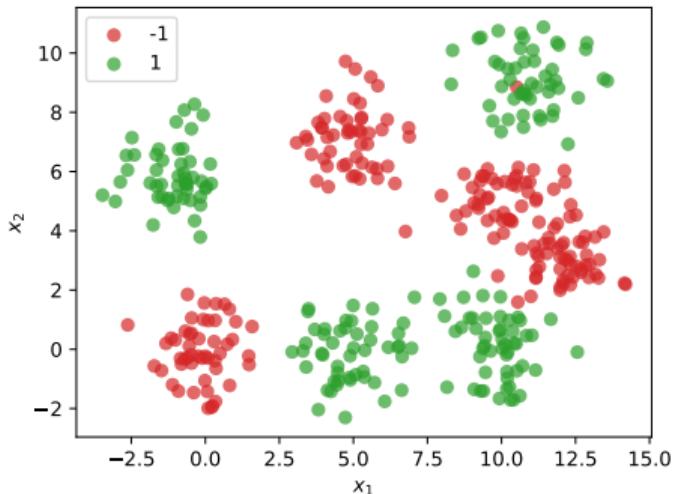
- ▶ Our prediction function H is a surface that is made up of Gaussian “bumps”.



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

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

Clustering

- ▶ Group data points into **clusters**.
- ▶ Cluster centers are good places for RBFs.
- ▶ We'll use k -means clustering to pick k centers.

DSC 190

Machine Learning: Representations

Lecture 5 | Part 2

k-means Clustering

Digression...

- ▶ Let's forget about RBFs for a minute...
- ▶ What is **clustering**?

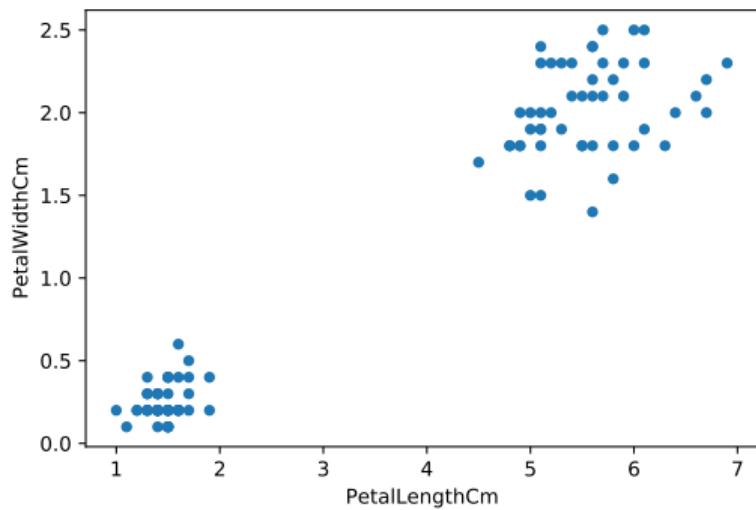
Clustering

- ▶ **Clustering** is a machine learning task whose goal is to find **group structure** in data.
- ▶ Why?
 - ▶ Exploratory data analysis.
 - ▶ Representation learning (today).

Example

- ▶ We gather measurements $\vec{x}^{(i)}$ of a bunch of flowers.
 - ▶ Petal length and petal width
- ▶ **Question:** how many species are there?
- ▶ **Goal:** **cluster** the similar flowers into groups.

Example



Supervised v. Unsupervised

- ▶ Clustering is an example of an **unsupervised** learning task.

Supervised Learning

- ▶ We tell the machine the “right answer”.
 - ▶ There is a **ground truth**.
- ▶ Data set: $\{(\vec{x}^{(i)}, y_i)\}$.
- ▶ **Goal:** learn relationship between features $\vec{x}^{(i)}$ and labels y_i .
- ▶ **Examples:** classification, regression.

Unsupervised Learning

- ▶ We don't tell the machine the “right answer”.
 - ▶ In fact, there might not be one!
- ▶ Data set: $\vec{x}^{(i)}$ (usually no test set)
- ▶ **Goal:** learn the **structure** of the data itself.
 - ▶ To discover something, for compression, to use as a feature later.
- ▶ **Example:** **clustering**

Ground Truth

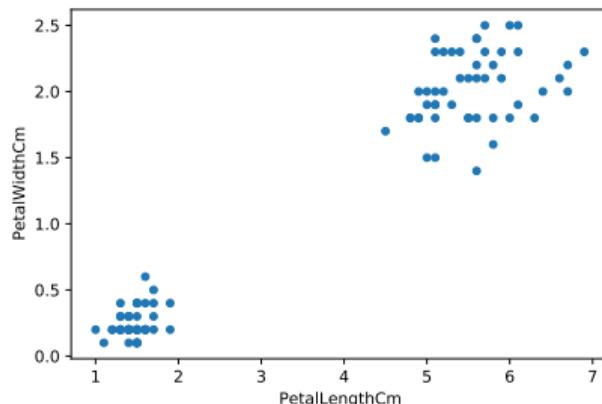
- ▶ If we don't have labels, we can't measure accuracy.
- ▶ Sometimes, labels don't exist.
- ▶ Example: cluster customers into types by previous purchases.

Clustering Approaches

- ▶ There are many approaches to clustering.
- ▶ One of the most popular is ***k-means clustering***.
- ▶ It turns clustering into an optimization problem.

Clustering as Optimization

- ▶ **Goal:** compress each clustering into a single point while minimizing information loss.

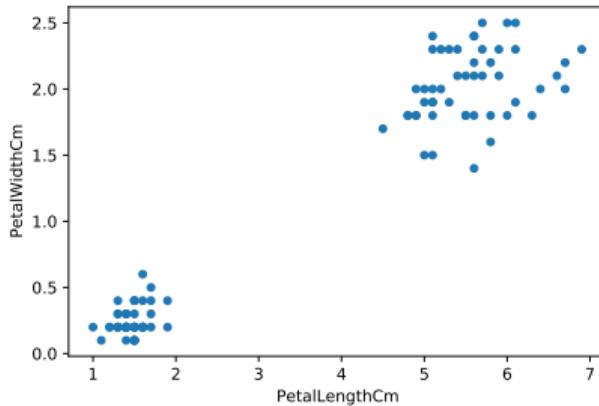


K-Means Objective

- ▶ **Given:** data, $\{\vec{x}^{(i)}\} \in \mathbb{R}^d$ and a parameter k .
- ▶ **Find:** k cluster centers $\vec{\mu}^{(1)}, \dots, \vec{\mu}^{(k)}$ so that the average squared distance from a data point to nearest cluster center is small.
- ▶ The **k-means objective function**:

$$\text{Cost}(\vec{\mu}^{(1)}, \dots, \vec{\mu}^{(k)}) = \frac{1}{n} \sum_{i=1}^n \min_{j \in \{1, \dots, k\}} \|\vec{x}^{(i)} - \vec{\mu}^{(j)}\|^2$$

Clustering as Optimization



$$\text{Cost}(\vec{\mu}^{(1)}, \dots, \vec{\mu}^{(k)}) = \frac{1}{n} \sum_{i=1}^n \min_{j \in \{1, \dots, k\}} \|\vec{x}^{(i)} - \vec{\mu}^{(j)}\|^2$$

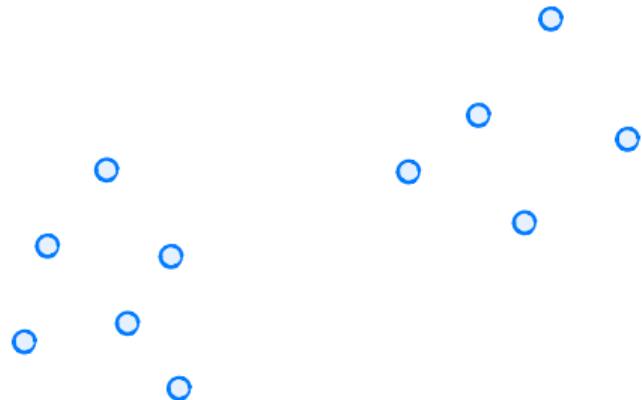
Optimization

- ▶ **Goal:** find $\vec{\mu}^{(1)}, \dots, \vec{\mu}^{(k)}$ minimizing k -means objective function.
- ▶ **Problem:** this is NP-Hard.
- ▶ We use a heuristic instead of solving exactly.

Lloyd's Algorithm for K-Means

- ▶ Initialize centers, $\vec{\mu}^{(1)}, \dots, \vec{\mu}^{(k)}$ somehow.
- ▶ Repeat until convergence:
 - ▶ Assign each point $\vec{x}^{(i)}$ to closest center
 - ▶ Update each $\vec{\mu}^{(i)}$ to be mean of points assigned to it

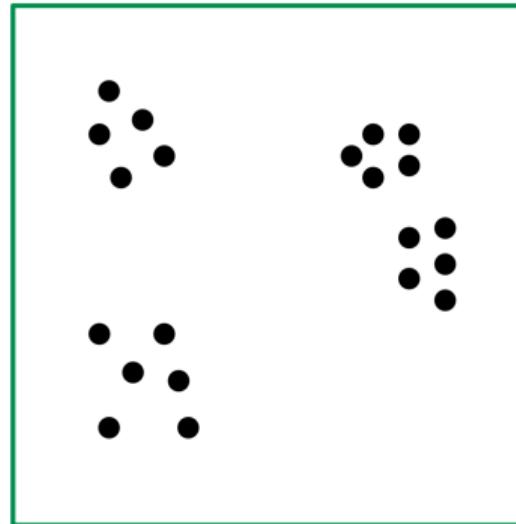
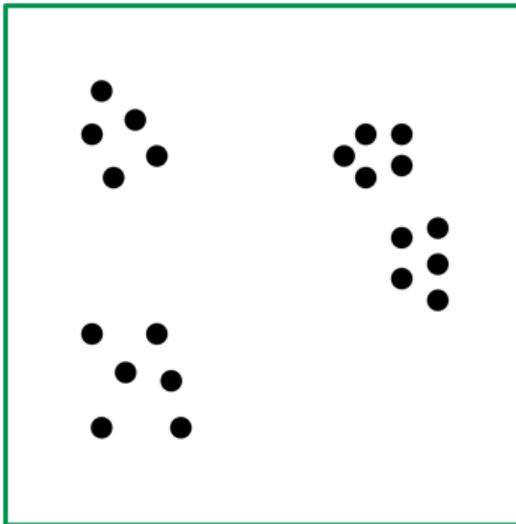
Example



Theory

- ▶ Each iteration reduces cost.
- ▶ This guarantees convergence to a **local** min.
- ▶ Initialization is very important.

Example



Initialization Strategies

- ▶ Basic Approach: Pick k data points at random.
- ▶ Better Approach: **k-means++:**
 - ▶ Pick first center at random from data.
 - ▶ Let $C = \{\vec{\mu}^{(1)}\}$ (centers chosen so far)
 - ▶ Repeat $k - 1$ more times:
 - ▶ Pick random data point \vec{x} according to distribution

$$P(\vec{x}) \propto \min_{\vec{\mu} \in C} \|\vec{x} - \mu\|^2$$

- ▶ Add \vec{x} to C

Picking k

- ▶ How do we know how many clusters the data contains?

Exercise

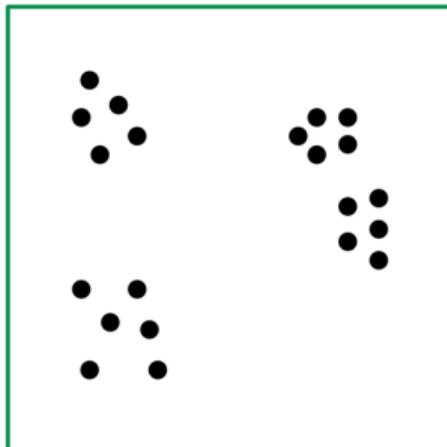
How does the **minimum** of the k -means objective function change as k is increased (that is, as we allow more clusters)?

Hint: What is the minimum when $k = n$ (we have one cluster per data point).

Plot of K-Means Objective

Observation

- ▶ Increasing k **always** decreases objective function
- ▶ But increasing k beyond “true” number of clusters has diminishing returns.



Picking k

- ▶ The **elbow method**:
 - ▶ Run k -means repeatedly with increasing values of k
 - ▶ Plot the value of the objective as a function of k
 - ▶ Find an **elbow** in the plot

Applications of K-Means

- ▶ Discovery
- ▶ Vector Quantization
 - ▶ Find a finite set of representatives of a large (possibly infinite) set.

Example

- ▶ Cluster animal descriptions.
- ▶ 50 animals: grizzly bear, dalmatian, rabbit, pig, ...
- ▶ 85 attributes: long neck, tail, walks, swims, ...
- ▶ 50 data points in \mathbb{R}^{85} . Run k -means with $k = 10$

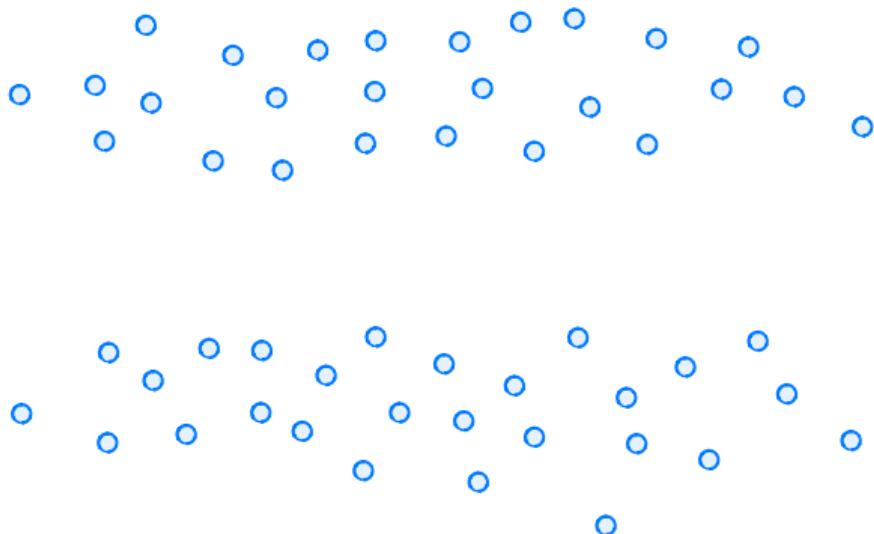
Results

- | | |
|--|--|
| <ul style="list-style-type: none">① zebra② spider monkey, gorilla, chimpanzee③ tiger, leopard, wolf, bobcat, lion④ hippopotamus, elephant, rhinoceros⑤ killer whale, blue whale, humpback whale, seal, walrus, dolphin⑥ giant panda⑦ skunk, mole, hamster, squirrel, rabbit, bat, rat, weasel, mouse, raccoon⑧ antelope, horse, moose, ox, sheep, giraffe, buffalo, deer, pig, cow⑨ beaver, otter⑩ grizzly bear, dalmatian, persian cat, german shepherd, siamese cat, fox, chihuahua, polar bear, collie | <ul style="list-style-type: none">① zebra② spider monkey, gorilla, chimpanzee③ tiger, leopard, fox, wolf, bobcat, lion④ hippopotamus, elephant, rhinoceros, buffalo, pig⑤ killer whale, blue whale, humpback whale, seal, otter, walrus, dolphin⑥ dalmatian, persian cat, german shepherd, siamese cat, chihuahua, giant panda, collie⑦ beaver, skunk, mole, squirrel, bat, rat, weasel, mouse, raccoon⑧ antelope, horse, moose, ox, sheep, giraffe, deer, cow⑨ hamster, rabbit⑩ grizzly bear, polar bear |
|--|--|

K-Means

- ▶ Perhaps the most popular clustering algorithm.
- ▶ **Fast, easy to understand.**
- ▶ **Assumes spherical clusters.**

Example



DSC 190

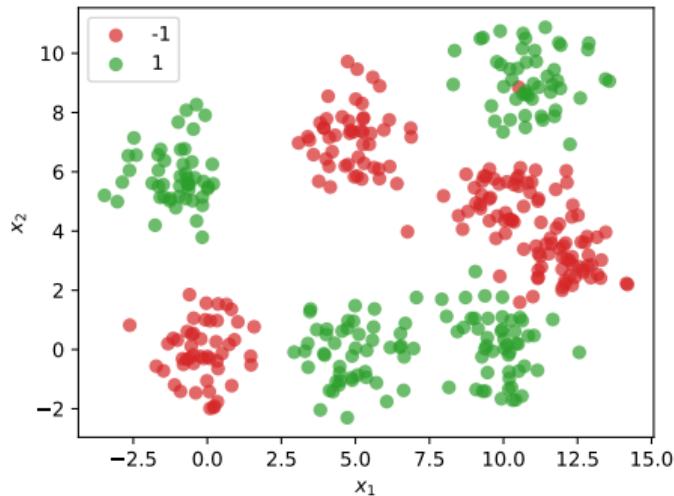
Machine Learning: Representations

Lecture 5 | Part 3

K-Means for Finding RBF Centers

Idea

- ▶ Use k -means centers as RBF centers.
- ▶ Typically “over-cluster” by setting k to be large.



Workflow

- ▶ “Over-cluster” with k -means to find centers
- ▶ Create new features using k RBFs
- ▶ Fit a least squares classifier

The Data

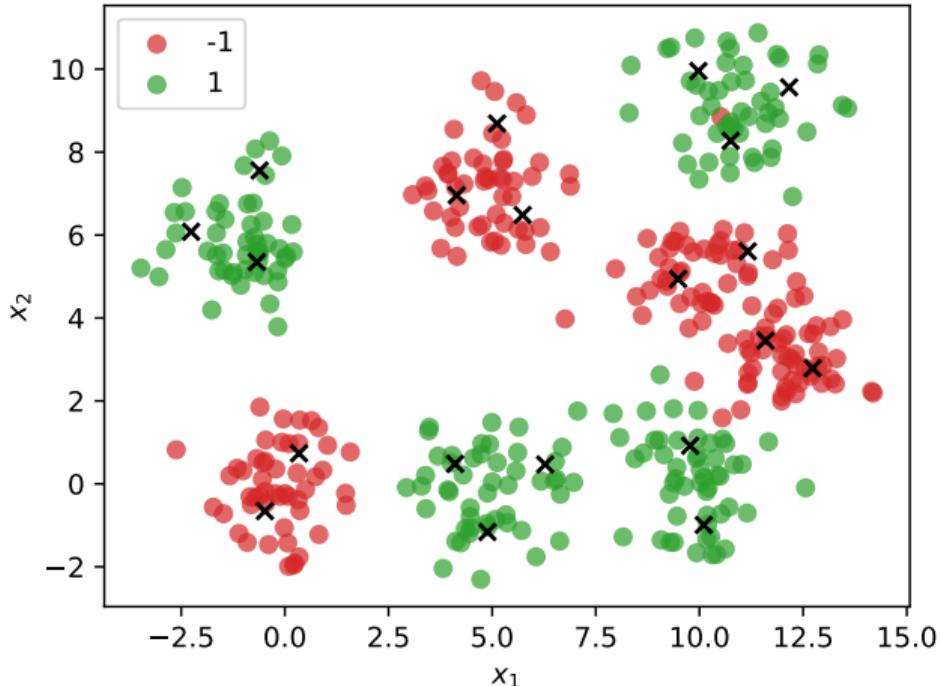
| | x1 | x2 |
|-----|-----------|-----------|
| 0 | 0.496714 | -0.138264 |
| 1 | 0.647689 | 1.523030 |
| 2 | -0.234153 | -0.234137 |
| 3 | 1.579213 | 0.767435 |
| 4 | -0.469474 | 0.542560 |
| ... | ... | ... |
| 395 | 10.429618 | 0.207688 |
| 396 | 10.271579 | -1.276749 |
| 397 | 8.918943 | 1.053153 |
| 398 | 9.960445 | 0.681501 |
| 399 | 10.028318 | 0.029756 |

400 rows × 2 columns

Step 1) k-means

```
>>> import sklearn.cluster  
>>> # let's start with 20 clusters  
>>> kmeans = sklearn.cluster.KMeans(n_clusters=20)  
>>> kmeans.fit(data)  
>>> cluster_centers = kmeans.cluster_centers_  
>>> cluster_centers.shape  
(20, 2)  
>>> cluster_centers  
array([[ 4.10556507,  0.48176175],  
       [ 9.48493465,  4.93921129],  
       [-0.67384089,  5.34791854],  
       [12.73048608,  2.78757872],  
       [12.16178039,  9.56285306],  
       [ 4.14585321,  6.95969919],  
       ...]
```

Cluster Centers



Step 2) Create New Features

- ▶ We've found $k = 20$ cluster centers, $\vec{\mu}^{(1)}, \dots, \vec{\mu}^{(20)}$.
- ▶ Center a Gaussian RBF at each.
- ▶ Take $\sigma = 3$ for now.
- ▶ We have $k = 20$ basis functions \rightarrow 20 newfeatures for every data point \vec{x} .

Creating the Features

```
def make_phi(center, sigma):
    def phi(x):
        return np.exp(
            -np.linalg.norm(x - center, axis=1)**2 / sigma**2
        )
    return phi

phis = [make_phi(center, 3) for center in cluster_centers]
```

Example

```
>>> phi_θ = phis[0]
>>> phi_θ(np.array([[1, 2], [4, 5], [6, 7]]))
array([0.26507796, 0.10336246, 0.00597847])
```

Applying the Basis Functions

```
>>> new_features = np.column_stack([phi(data) for phi in phis])
>>> new_features.shape
(400, 20)
```

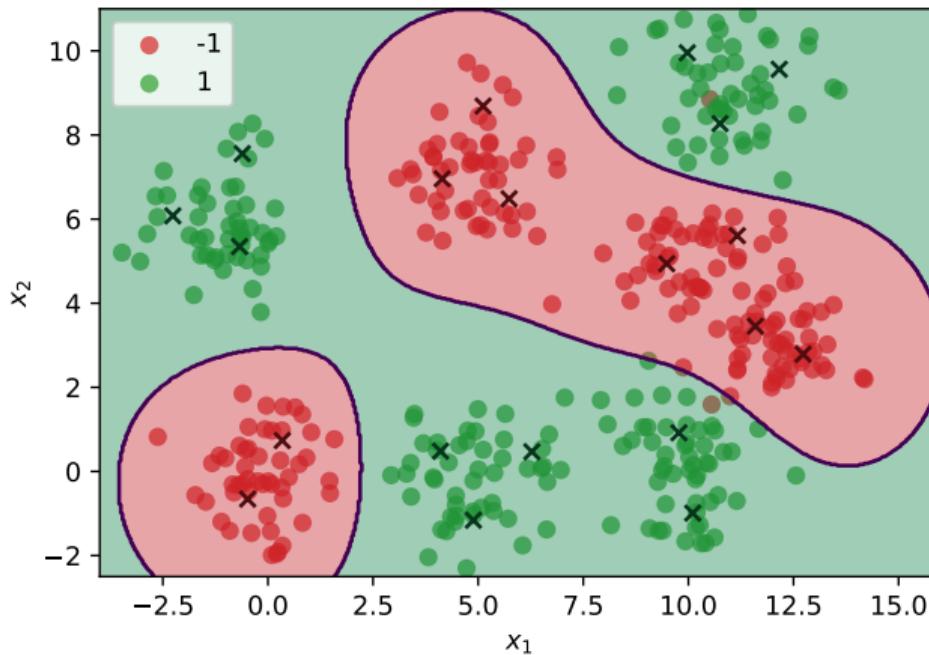
Step 3) Fitting the classifier

```
def augment(X):
    return np.column_stack((
        np.ones(len(X)),
        X
    ))
```

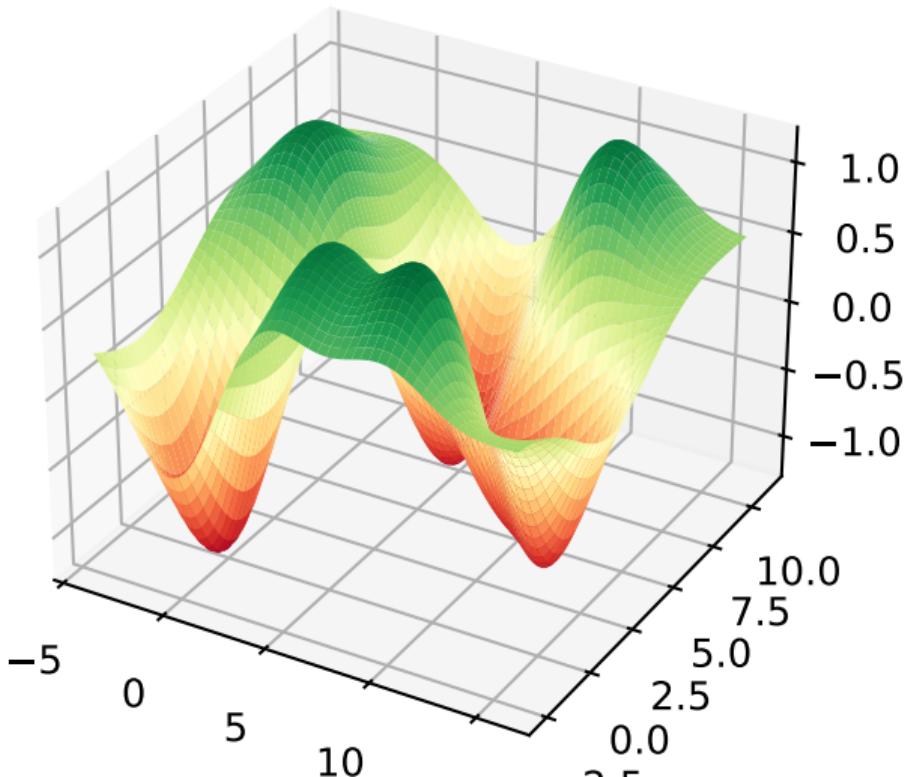
Fitting the Classifier

```
>>> X = augment(new_features)
>>> w = np.linalg.lstsq(X, y)[0]
>>> predictions = np.sign(X @ w)
>>> # training accuracy
>>> (predictions == y).mean()
0.995
```

Decision Boundary



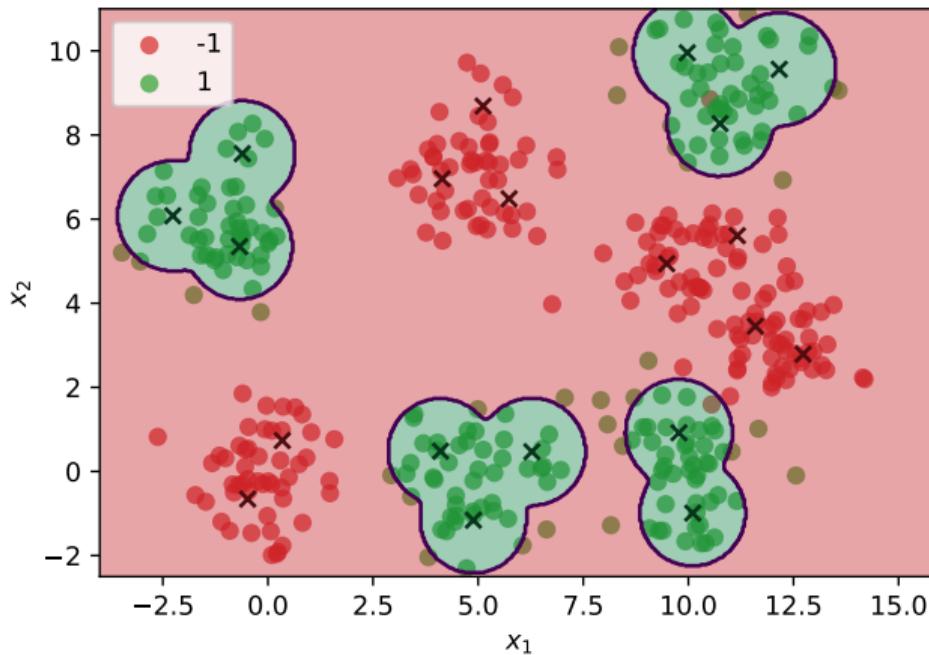
Prediction Surface



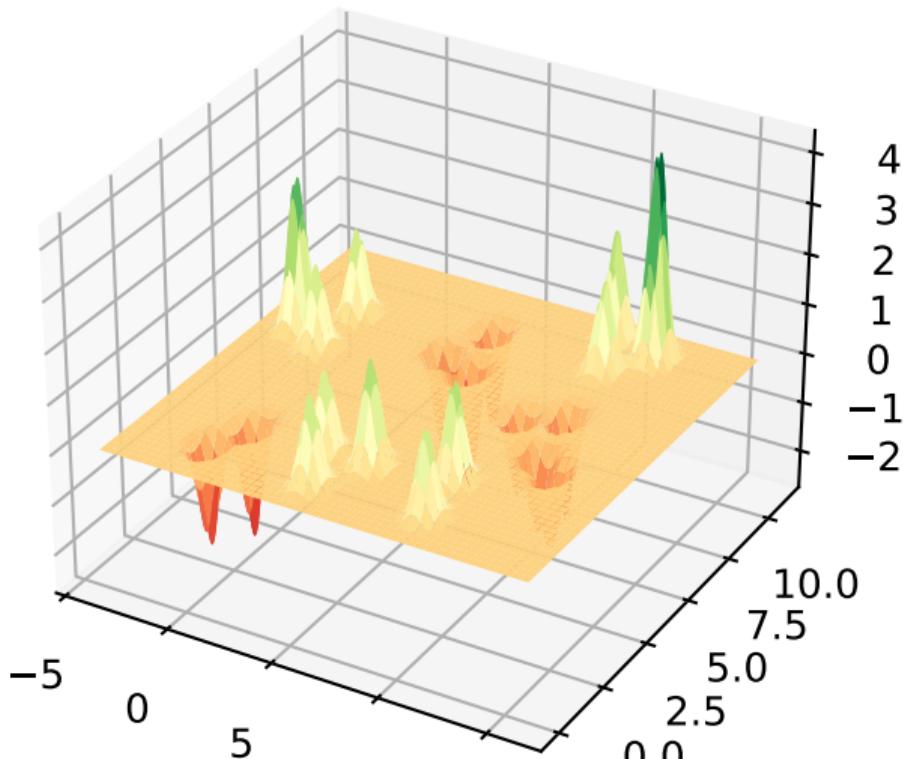
What if...

- ▶ What if we make σ smaller?
- ▶ Set $\sigma = \frac{1}{2}$

Small Sigma



Small Sigma



Model Complexity

- ▶ RBF network complexity is determined by:
 - ▶ Number of basis functions (more = more complex)
 - ▶ RBF width parameter (smaller = more complex)
- ▶ Choose via cross validation
- ▶ More complex = greater danger of overfitting

Representation Learning

- ▶ This class is about “representation learning”
- ▶ This is the first time we’ve actually **learned** a representation.
- ▶ Previously: chose basis functions by hand.
- ▶ Today: we used k-means to **learn** good basis functions using the data.

DSC 190

Machine Learning: Representations

Lecture 6 | Part 1

Vectors

And now for something completely different...

- ▶ This and the next lecture will be linear algebra refreshers.
- ▶ Today: what is a matrix?
- ▶ Next lecture: what are eigenvectors/values?

Vectors

- ▶ A vector \vec{x} is an arrow from the origin to a point.
- ▶ We can make new arrows by:
 - ▶ scaling: $\alpha\vec{x}$
 - ▶ addition: $\vec{x} + \vec{y}$
 - ▶ both: $\alpha\vec{x} + \beta\vec{y}$
- ▶ $\|\vec{x}\|$ is the **norm** (or length) of \vec{x}

Linear Combinations

- ▶ We can add together a bunch of arrows:

$$\vec{y} = \alpha_1 \vec{x}^{(1)} + \alpha_2 \vec{x}^{(2)} + \dots + \alpha_n \vec{x}^{(n)}$$

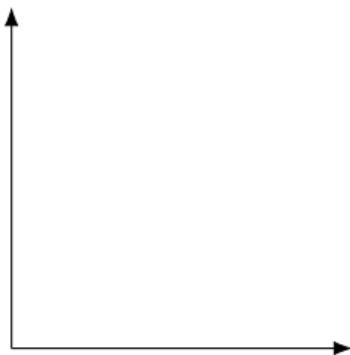
- ▶ This is a **linear combination** of $\vec{x}^{(1)}, \dots, \vec{x}^{(n)}$

Parallel Vectors

- ▶ Two vectors \vec{x} and \vec{y} are **parallel** if (and only if) there is a scalar λ such that $\vec{x} = \lambda\vec{y}$.

Standard Basis Vectors

- $\hat{e}^{(1)}$ and $\hat{e}^{(2)}$ are the **standard basis vectors** in \mathbb{R}^2 .
 - $\|\hat{e}^{(1)}\| = \|\hat{e}^{(2)}\| = 1$

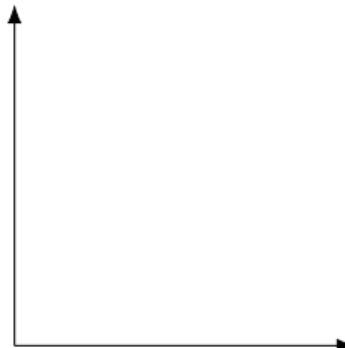


Standard Basis Vectors

- $\hat{e}^{(1)}, \dots, \hat{e}^{(d)}$ are the **standard basis vectors** in \mathbb{R}^d .

Decompositions

- ▶ We can **decompose** any vector $\vec{x} \in \mathbb{R}^2$ in terms of $\hat{e}^{(1)}$ and $\hat{e}^{(2)}$
 - ▶ Write: $\vec{x} = x_1 \hat{e}^{(1)} + x_2 \hat{e}^{(2)}$



Decompositions

- ▶ We can **decompose** any vector $\vec{x} \in \mathbb{R}^d$ in terms of $\hat{e}^{(1)}, \hat{e}^{(2)}, \dots, \hat{e}^{(d)}$
 - ▶ Write: $\vec{x} = x_1 \hat{e}^{(1)} + x_2 \hat{e}^{(2)} + \dots + x_d \hat{e}^{(d)}$

Coordinate Vectors

- We often write a vector \vec{x} as a **coordinate vector**:

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

- Meaning: $\vec{x} = x_1 \hat{e}^{(1)} + x_2 \hat{e}^{(2)} + \dots + x_d \hat{e}^{(d)}$

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:

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

$$= (u_1 \ u_2 \ \cdots \ u_d) \begin{pmatrix} v_1 \\ v_2 \\ \vdots \\ v_d \end{pmatrix}$$

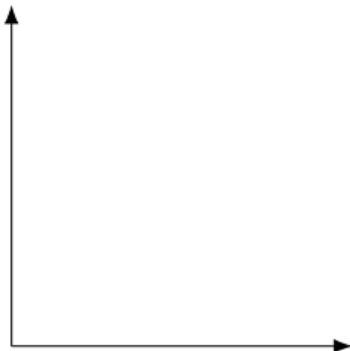
=

Exercise

Show that $\vec{v} \cdot \vec{v} = \|\vec{v}\|^2$.

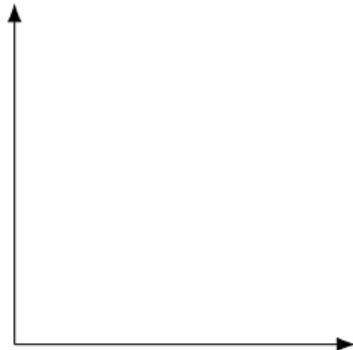
Projections

- ▶ If \hat{u} is a unit vector, $\vec{v} \cdot \hat{u}$ is the “part of \vec{v} that lies in the direction of \hat{u} ”.
 - ▶ $\vec{v} \cdot \hat{u} = \|\vec{v}\| \|\hat{u}\| \cos \theta$



Projections

- ▶ Namely, if $\vec{x} = (x_1, \dots, x_d)^T$, then $\vec{x} \cdot \hat{e}^{(k)} = x_k$.



DSC 190

Machine Learning: Representations

Lecture 6 | Part 2

Functions of a Vector

Functions of a Vector

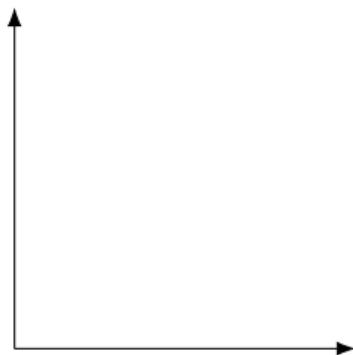
- ▶ In ML, we often work with functions of a vector:
 $f : \mathbb{R}^d \rightarrow \mathbb{R}^{d'}$.
- ▶ Example: a prediction function, $H(\vec{x})$.
- ▶ Functions of a vector can return:
 - ▶ a number: $f : \mathbb{R}^d \rightarrow \mathbb{R}^1$
 - ▶ a vector $\vec{f} : \mathbb{R}^d \rightarrow \mathbb{R}^{d'}$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

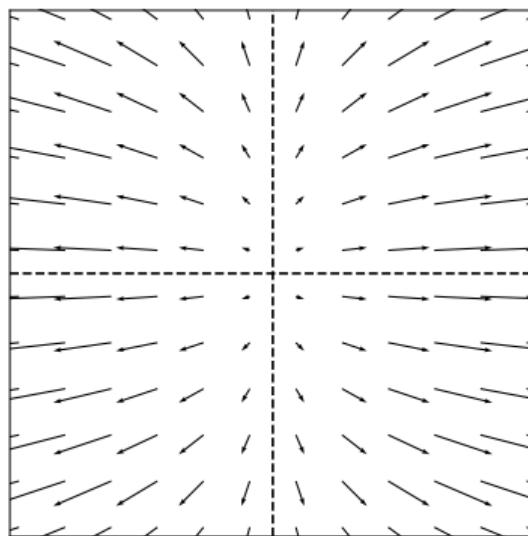
Visualizing Transformations

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



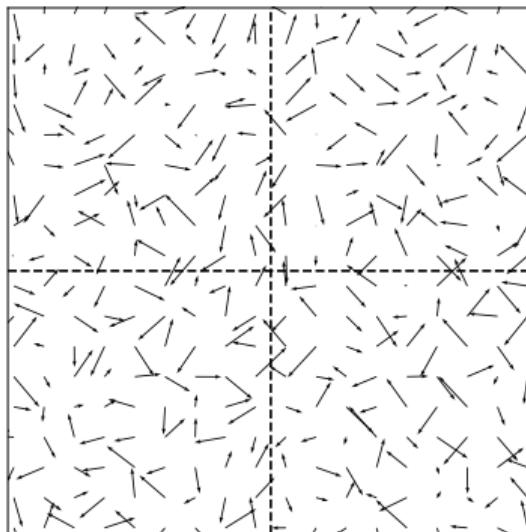
Example

► $\vec{f}(\vec{x}) = (3x_1, x_2)^T$



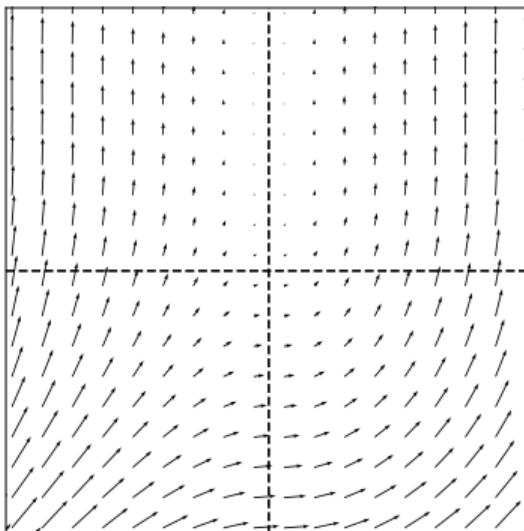
Arbitrary Transformations

- ▶ 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!

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

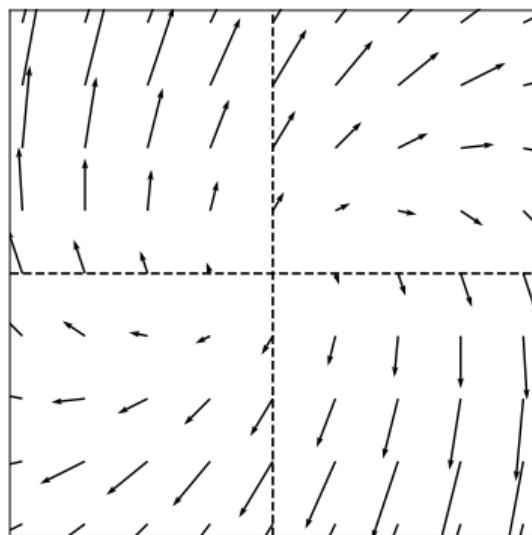
- ▶ Suppose f is a **linear** transformation.
- ▶ I tell you $\vec{f}(\hat{e}^{(1)}) = (2, 1)^T$ and $\vec{f}(\hat{e}^{(2)}) = (-3, 0)^T$.
- ▶ I tell you $\vec{x} = (3, -4)^T$.
- ▶ What is $\vec{f}(\vec{x})$?

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!

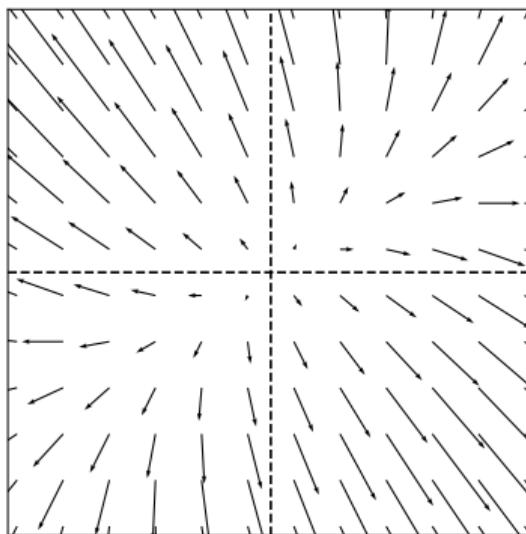
Example Linear Transformation

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



Another Example Linear Transformation

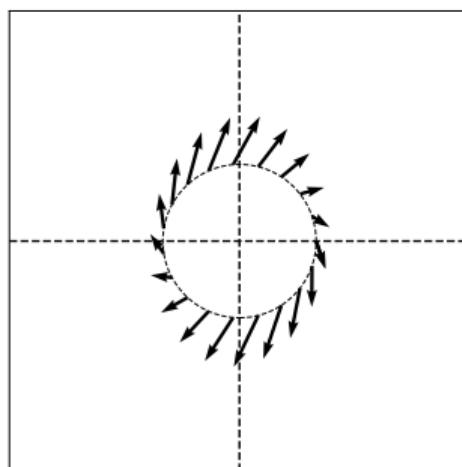
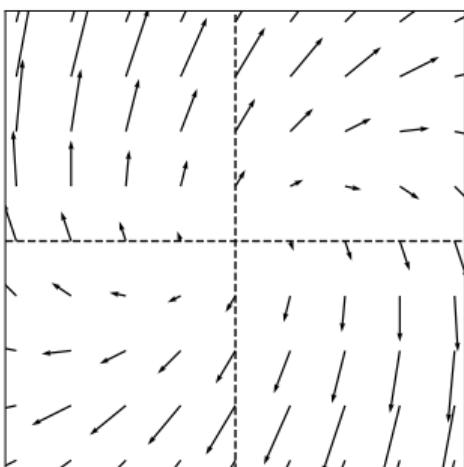
► $\vec{f}(\vec{x}) = (2x_1 - x_2, -x_1 + 3x_2)^T$

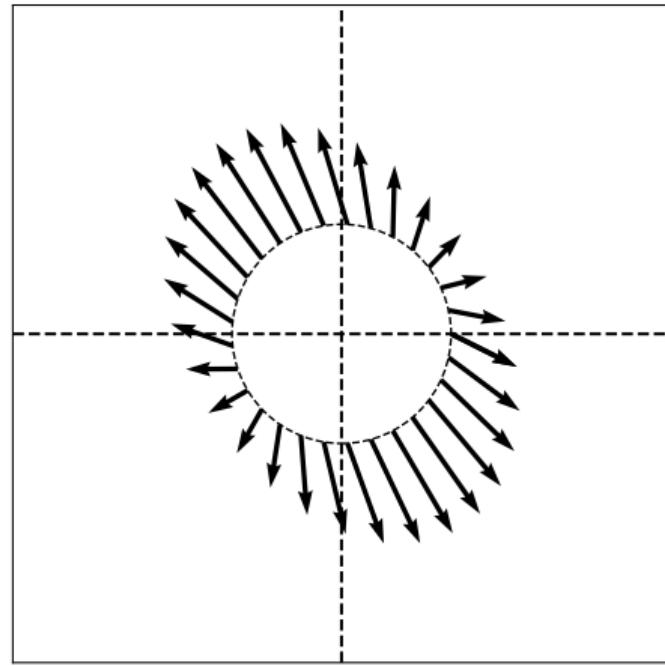
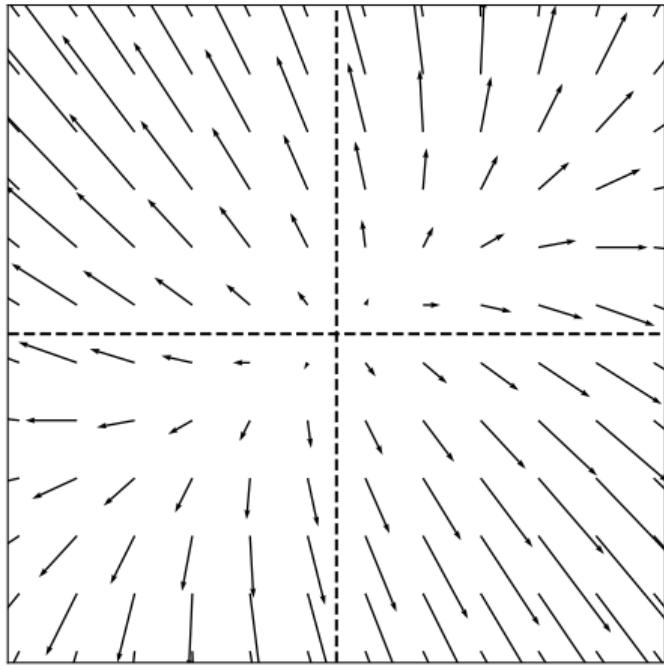


Note

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

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





DSC 190

Machine Learning: Representations

Lecture 6 | Part 3

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?

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

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...

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

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} .

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 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 $f : \mathbb{R}^n \rightarrow \mathbb{R}^n$.

Matrix Multiplication

- ▶ Matrix A represents a function f
- ▶ Matrix multiplication $A\vec{x}$ **evaluates** $\vec{f}(\vec{x})$

Matrix Multiplication

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

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

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

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

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

Example

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

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

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

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

$$A\vec{x} =$$

Main Idea

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

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 190

Machine Learning: Representations

Lecture 7 | Part 1

The Spectral Theorem

Eigenvectors

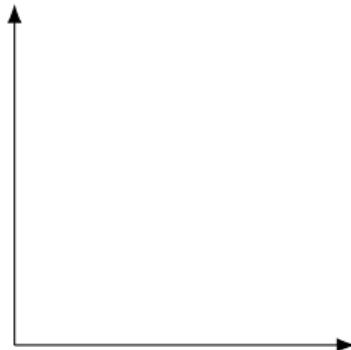
- ▶ Let A be an $n \times n$ matrix. An **eigenvector** of A with **eigenvalue** λ is a nonzero vector \vec{v} such that $A\vec{v} = \lambda\vec{v}$.

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 $f(\vec{v}) = \lambda\vec{v}$.

Geometric Interpretation

- ▶ When \vec{f} is applied to one of its eigenvectors, \vec{f} simply scales it.
- ▶ That is, it doesn't **rotate** it.



Symmetric Matrices

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

The Spectral Theorem¹

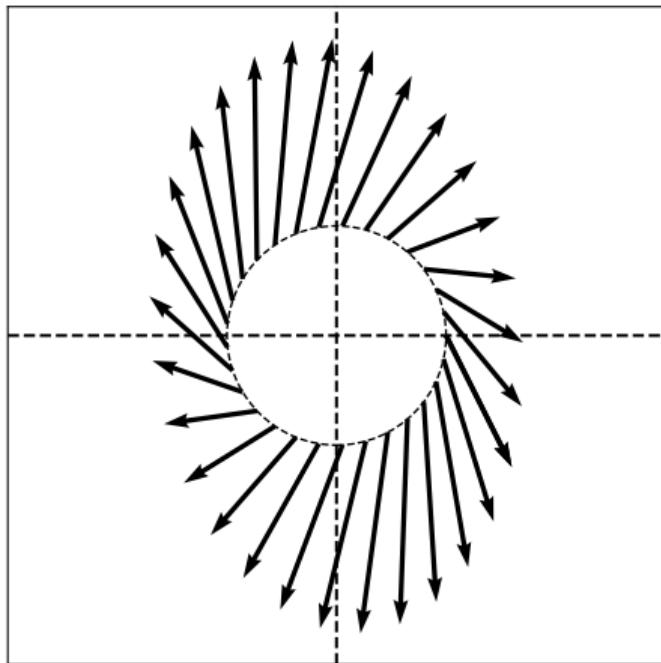
- ▶ **Theorem:** Let A be an $n \times n$ symmetric matrix. Then there exist n eigenvectors of A which are all mutually orthogonal.

¹for symmetric matrices

What?

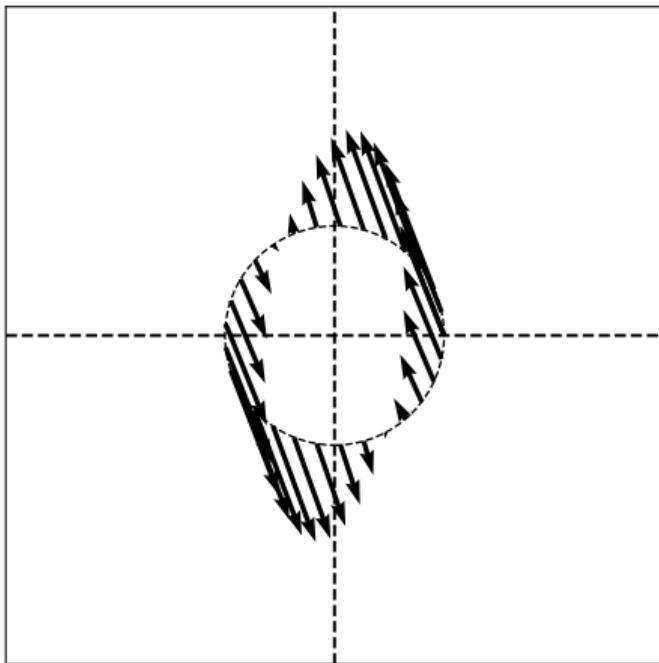
- ▶ What does the spectral theorem mean?
- ▶ What is an eigenvector, really?
- ▶ Why are they useful?

Example Linear Transformation



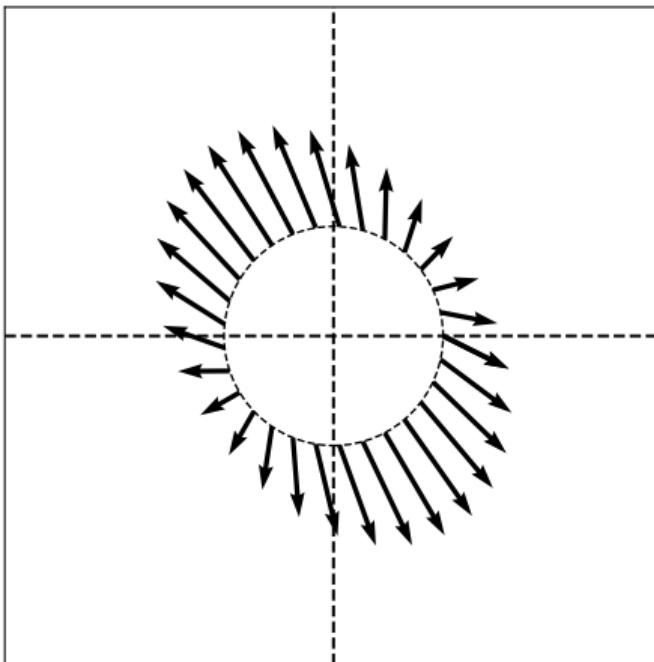
$$A = \begin{pmatrix} 5 & 5 \\ -10 & 12 \end{pmatrix}$$

Example Linear Transformation



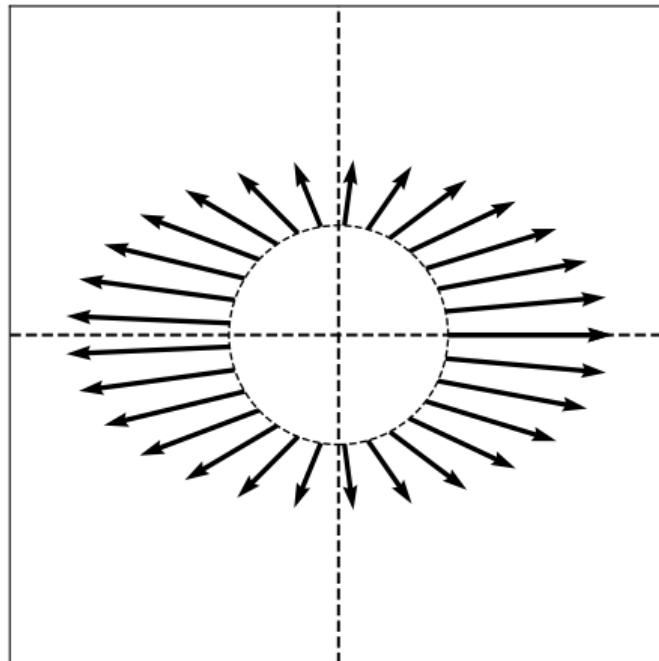
$$A = \begin{pmatrix} -2 & -1 \\ -5 & 3 \end{pmatrix}$$

Example Symmetric Linear Transformation



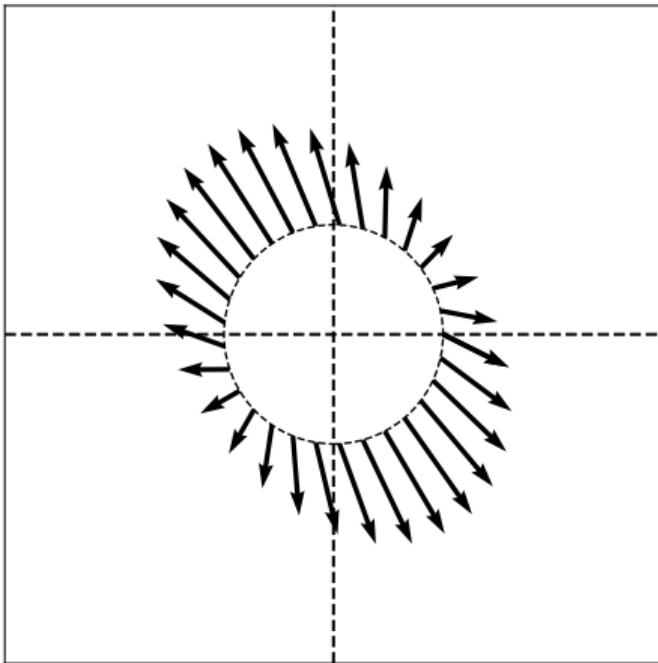
$$A = \begin{pmatrix} 2 & -1 \\ -1 & 3 \end{pmatrix}$$

Example Symmetric Linear Transformation



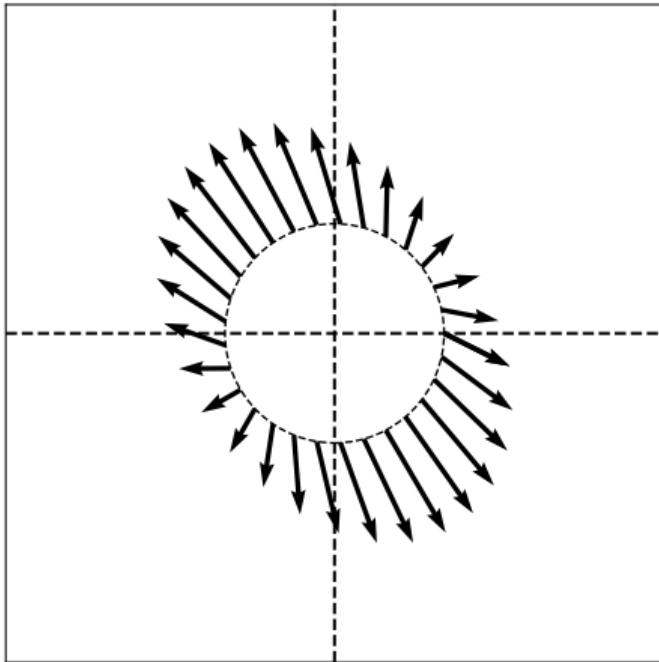
$$A = \begin{pmatrix} 5 & 0 \\ 0 & 2 \end{pmatrix}$$

Observation #1



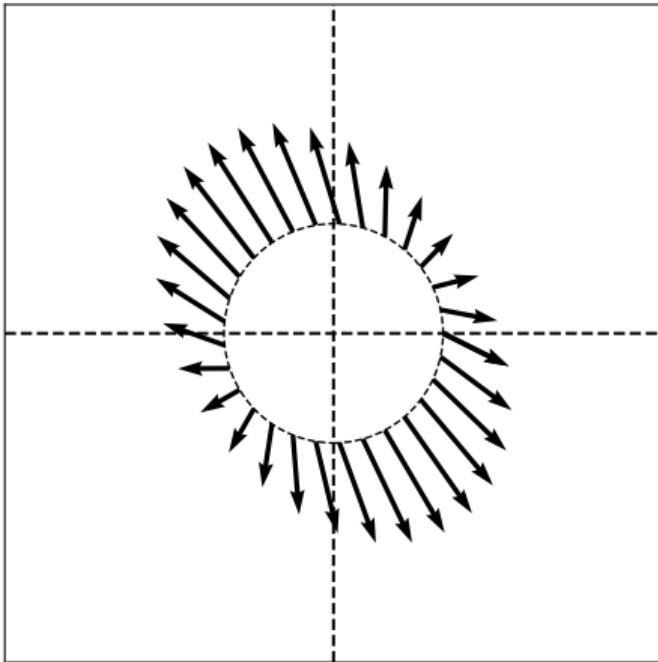
- ▶ Symmetric linear transformations have **axes of symmetry**.

Observation #2



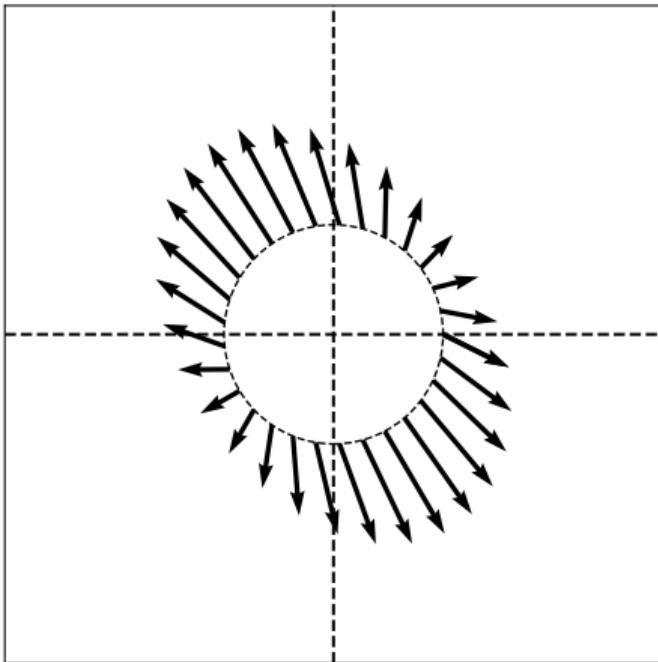
- ▶ The axes of symmetry are **orthogonal** to one another.

Observation #3



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

Observation #4

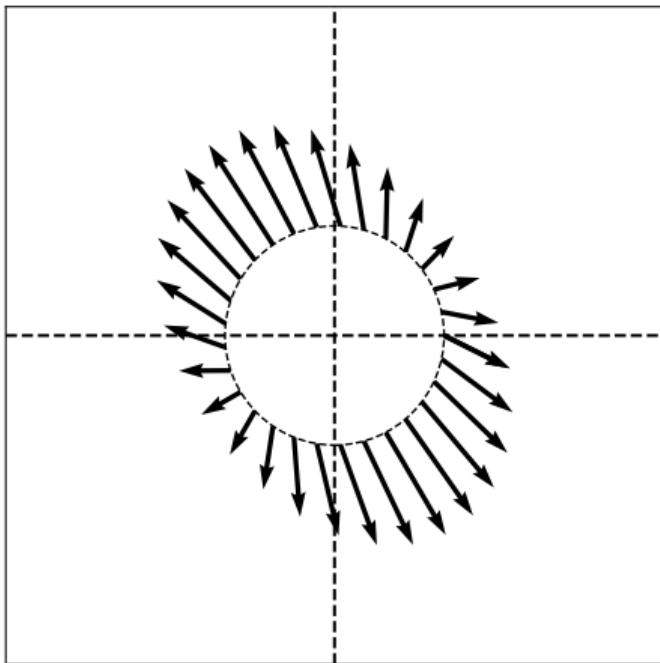


- ▶ The size of this scaling can be different for each axis.

Main Idea

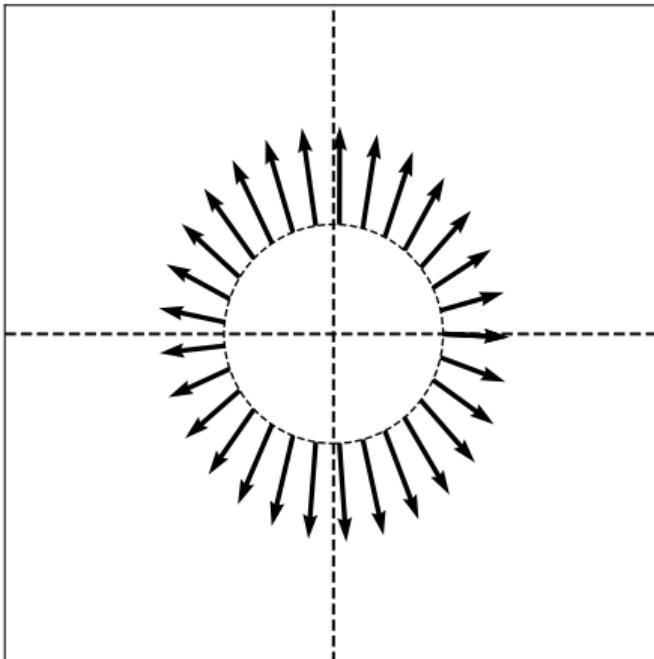
The **eigenvectors** of a symmetric linear transformation (matrix) are its axes of symmetry. The **eigenvalues** describe how much each axis of symmetry is scaled.

Example



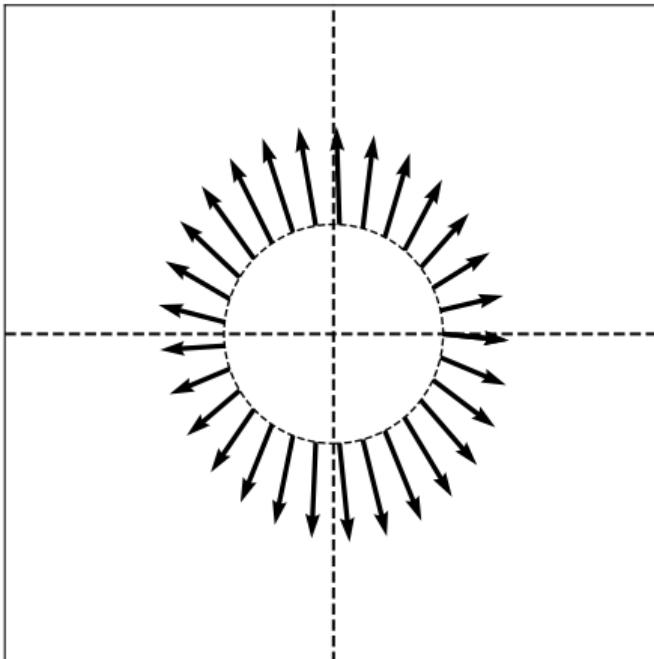
```
>>> A = np.array([[2, -1], [-1, 3]])
>>> np.linalg.eigh(A)
(array([1.38196601, 3.61803399]),
 array([[-0.85065081, -0.52573111],
       [-0.52573111, 0.85065081]]))
```

Off-diagonal elements



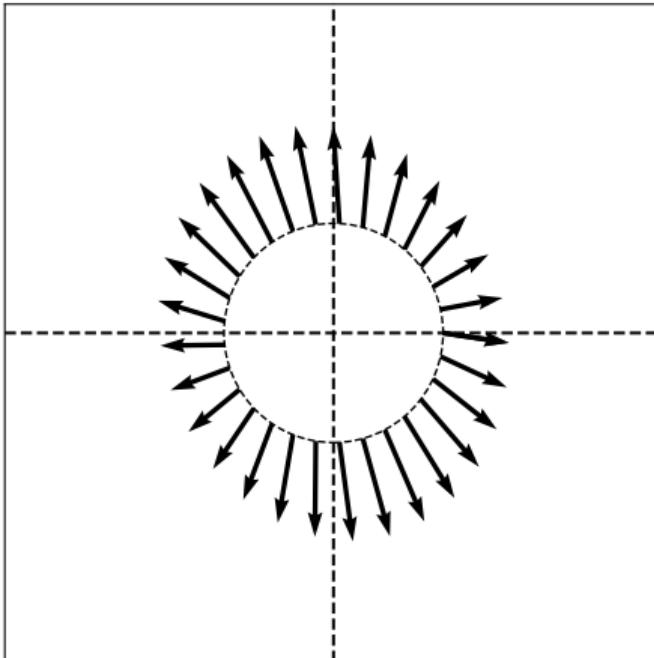
$$A = \begin{pmatrix} 5 & -0.1 \\ -0.1 & 2 \end{pmatrix}$$

Off-diagonal elements



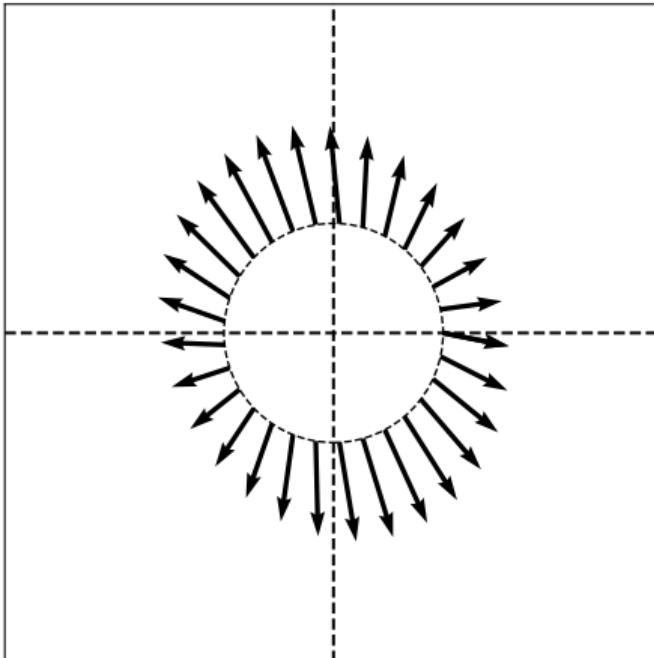
$$A = \begin{pmatrix} 5 & -0.2 \\ -0.2 & 2 \end{pmatrix}$$

Off-diagonal elements



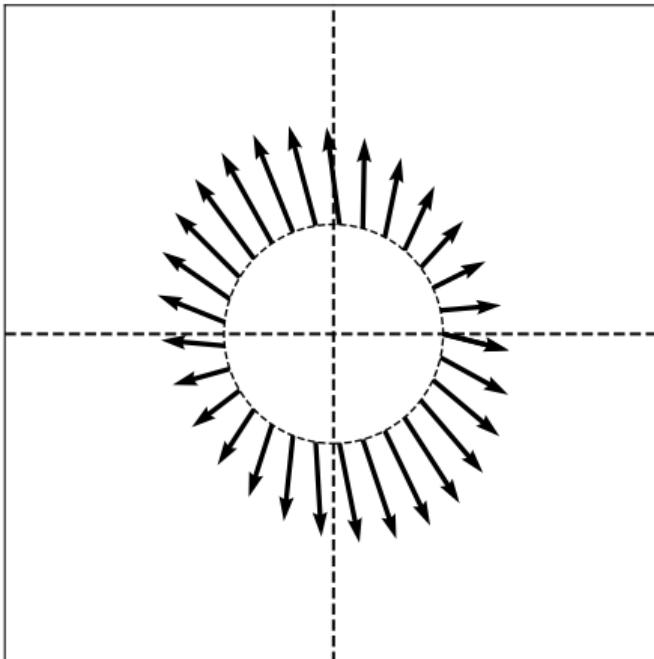
$$A = \begin{pmatrix} 5 & -0.3 \\ -0.3 & 2 \end{pmatrix}$$

Off-diagonal elements



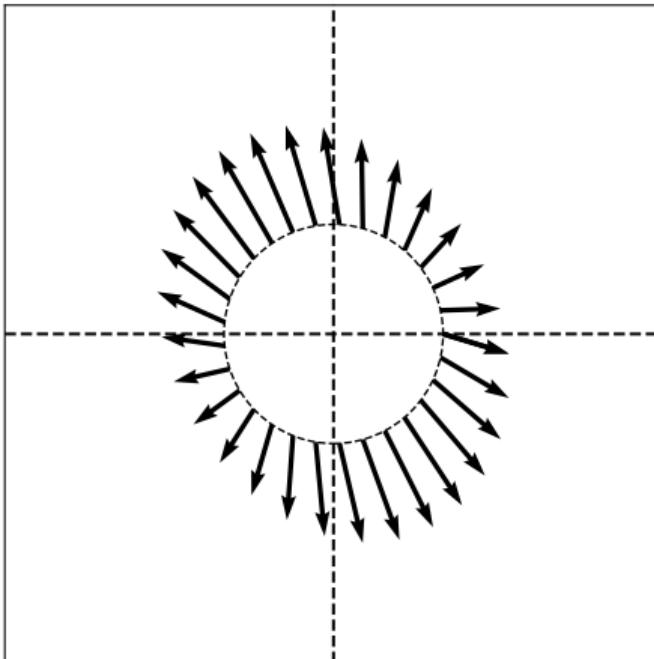
$$A = \begin{pmatrix} 5 & -0.4 \\ -0.4 & 2 \end{pmatrix}$$

Off-diagonal elements



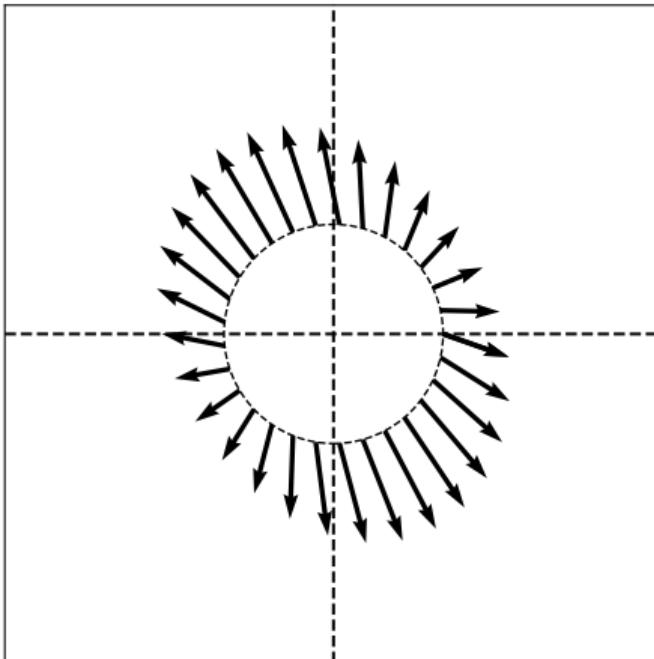
$$A = \begin{pmatrix} 5 & -0.5 \\ -0.5 & 2 \end{pmatrix}$$

Off-diagonal elements



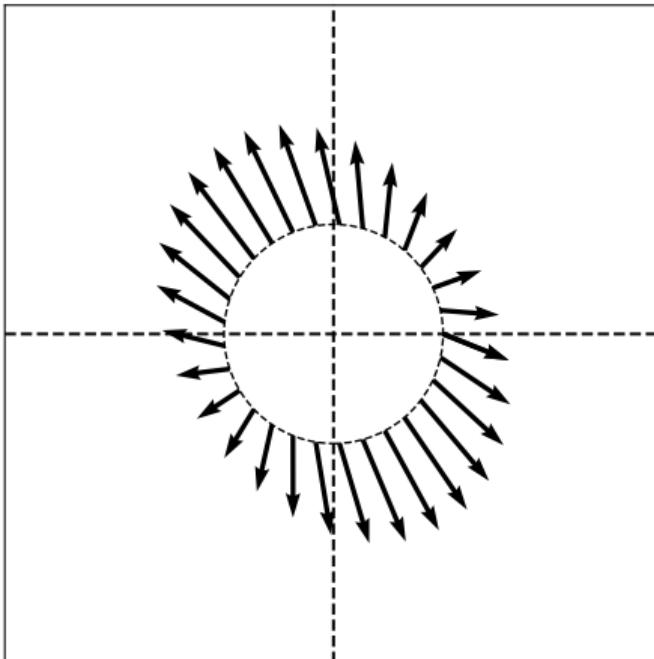
$$A = \begin{pmatrix} 5 & -0.6 \\ -0.6 & 2 \end{pmatrix}$$

Off-diagonal elements



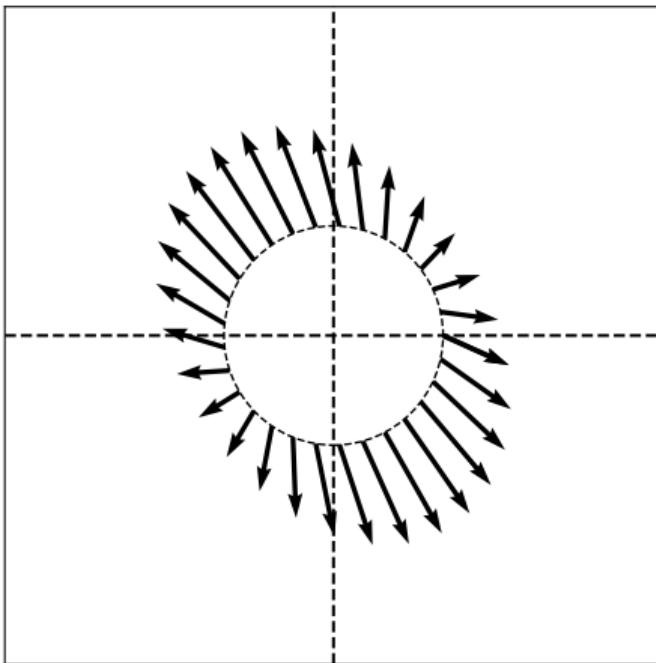
$$A = \begin{pmatrix} 5 & -0.7 \\ -0.7 & 2 \end{pmatrix}$$

Off-diagonal elements



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

Off-diagonal elements



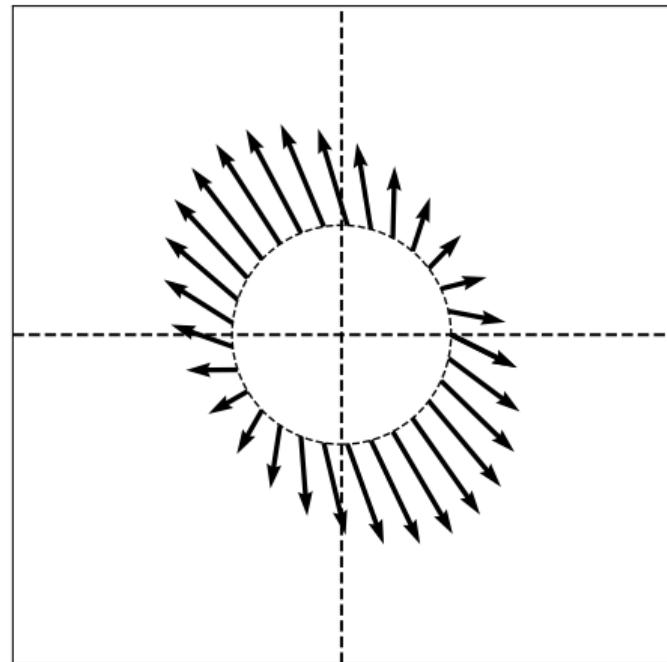
$$A = \begin{pmatrix} 5 & -0.9 \\ -0.9 & 2 \end{pmatrix}$$

Why does $A^T = A$ result in symmetry?

► $A^T = A \implies \vec{f}(\hat{e}^{(1)}) \cdot \hat{e}^{(2)} = \vec{f}(\hat{e}^{(2)}) \cdot \hat{e}^{(1)}$

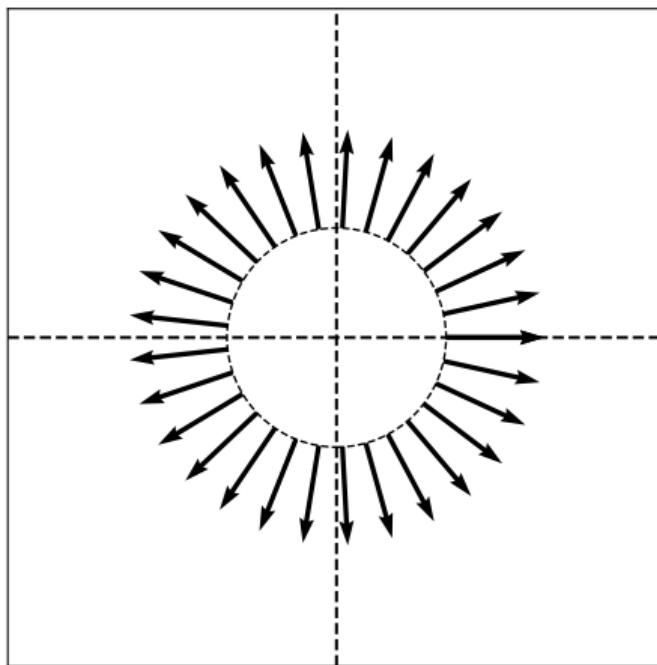
The Spectral Theorem²

- ▶ **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 about total symmetry?



- ▶ Every vector is an eigenvector.

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

DSC 190

Machine Learning: Representations

Lecture 7 | Part 2

Why are eigenvectors useful?

OK, but why are eigenvectors³ useful?

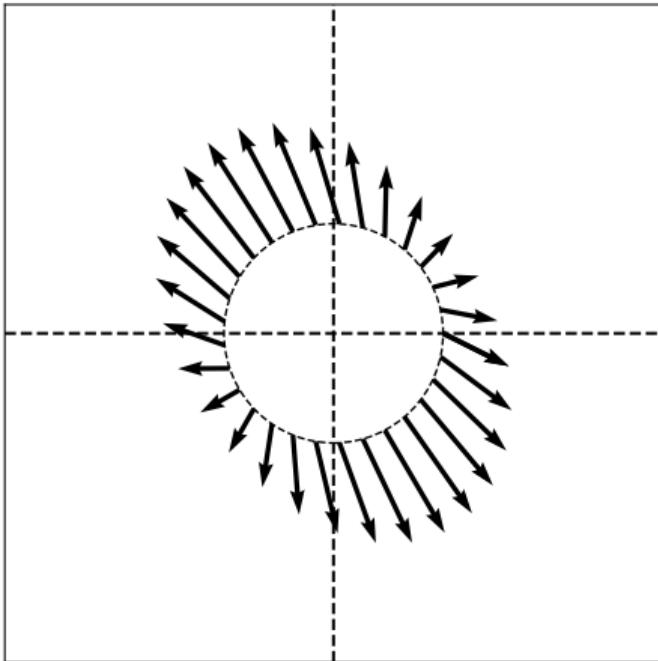
- ▶ Eigenvectors are nice “building blocks” (basis vectors).
- ▶ Eigenvectors are **maximizers** (or minimizers).
- ▶ Eigenvectors are **equilibria**.

³of symmetric matrices

Eigendecomposition

- ▶ Any vector \vec{x} can be written in terms of the eigenvectors of a symmetric matrix.
- ▶ This is called its **eigendecomposition**.

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 $\|\vec{f}(\vec{x})\|$ 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).

Proof

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

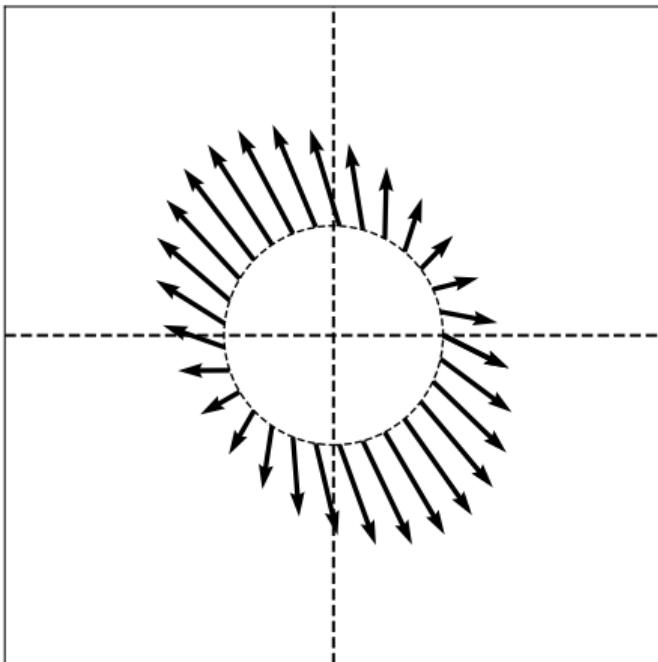
Corollary

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

Example

- ▶ Maximize $4x_1^2 + 2x_2 + 3x_1x_2$ subject to $x_1^2 + x_2^2 = 1$

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 190

Machine Learning: Representations

Lecture 7 | Part 3

Diagonalization

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 .

Note about Bases

- ▶ To write the matrix representation of f , you must first choose a basis.
- ▶ If it isn't stated, we'll assume the standard basis.
- ▶ But we can also write a matrix representing f in some other basis.

$$f(\hat{u}^{(1)}) = 2\hat{u}^{(1)} + 3\hat{u}^{(2)} = (2, 3)_{\mathcal{U}}^T$$

$$f(\hat{u}^{(2)}) = -5\hat{u}^{(1)} - \hat{u}^{(2)} = (-5, -1)_{\mathcal{U}}^T$$

$$A_{\mathcal{U}} =$$

Eigenbasis

- ▶ A basis of eigenvectors is particularly natural.
- ▶ Example: $\vec{f}(\vec{v}^{(1)}) = \lambda_1 \vec{v}^{(1)}$, $\vec{f}(\vec{v}^{(2)}) = \lambda_2 \vec{v}^{(2)}$
- ▶ Matrix representing \vec{f} in the eigenbasis:

Two Approaches

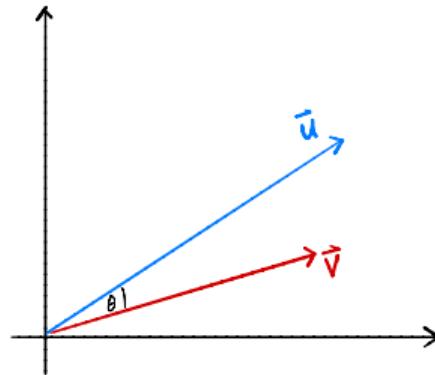
- ▶ Approach 1:
 - ▶ Write matrix for A w.r.t. standard basis
 - ▶ $\vec{f}(\vec{x}) = A\vec{x}$
- ▶ Approach 2:
 - ▶ Change basis to **eigenbasis**
 - ▶ Apply matrix representing \vec{f} in the eigenbasis (simple)
 - ▶ Change basis back to original basis

Spectral Theorem (Again)

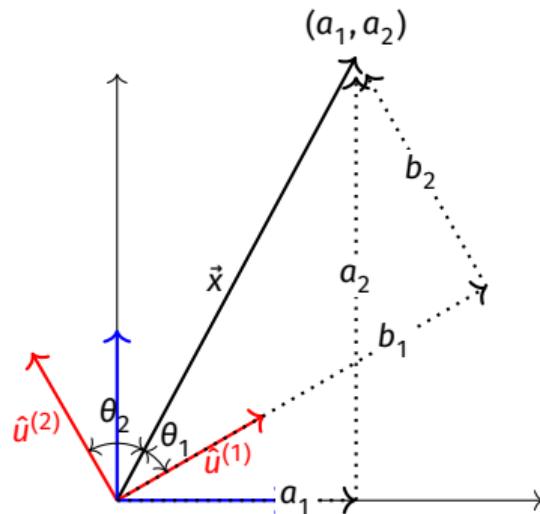
- ▶ **Theorem:** Let A be an $n \times n$ symmetric matrix. Then there exists an orthogonal matrix U and a diagonal matrix Λ such that $A = U^T \Lambda U$.
- ▶ Interpretation:
 - ▶ Change basis by multiplying by U
 - ▶ Λ is the representation of \vec{f} in the eigenbasis
 - ▶ Change basis back by multiplying by U^T

Geometric Interpretation of $\vec{u} \cdot \vec{v}$

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



Change of Basis



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

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

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

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

Example

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

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

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

Change of Basis Matrix

- ▶ Changing basis is a linear transformation

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

- ▶ We can represent it with a matrix

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

Example

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

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

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

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

$$A =$$

Change of Basis Matrix

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

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

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

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

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

Change to Eigenbasis

- ▶ It can be shown that the matrix which changes basis to the eigenbasis of A is the orthogonal matrix U , whose rows are the eigenvectors of A .

DSC 190

Machine Learning: Representations

Lecture 8 | Part 1

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
 - ▶ :

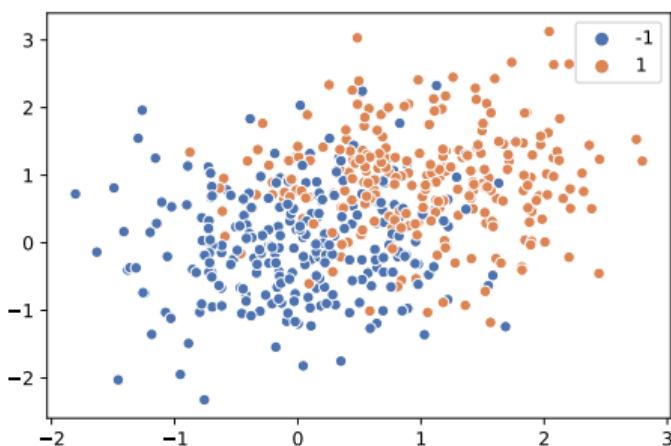
High Dimensional Data

- ▶ 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

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 $d' \leq d$

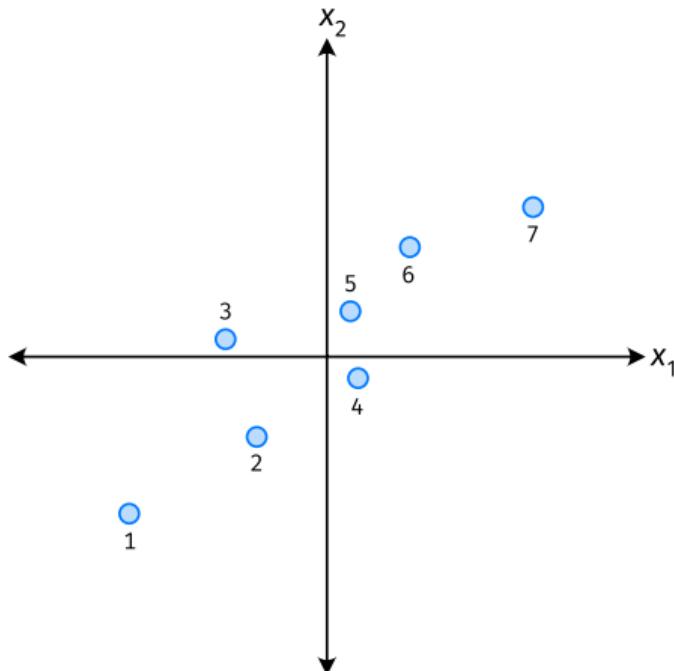
Today's Example

- ▶ 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

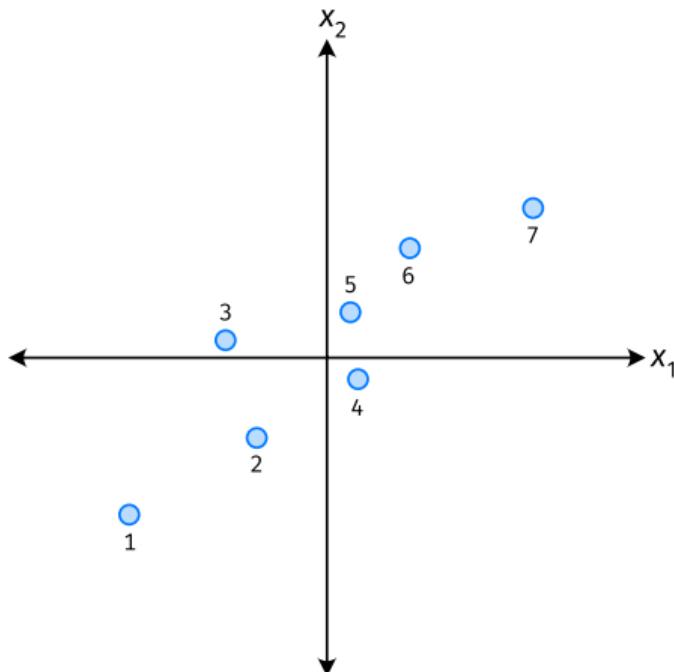
- ▶ Screen width and weight share information.
- ▶ **Idea:** keep one feature, remove the other.
- ▶ That is, set new feature $z = x_1$ (or $z = x_2$).

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.

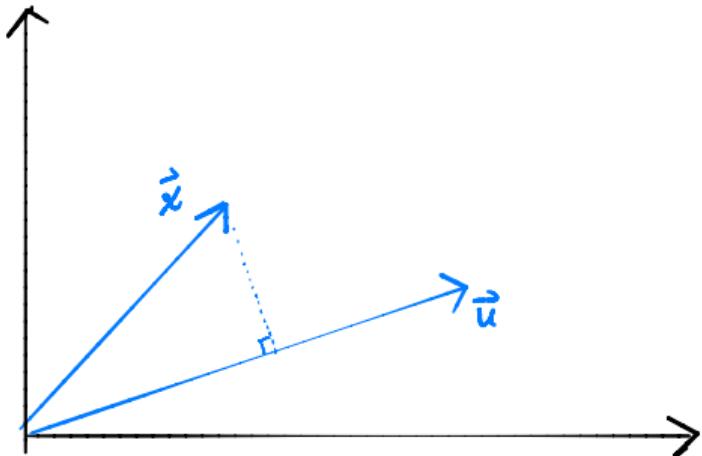
$$\begin{aligned} z &= u_1 x_1 + u_2 x_2 \\ &= \vec{u} \cdot \vec{x} \end{aligned}$$

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

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

Geometric Interpretation

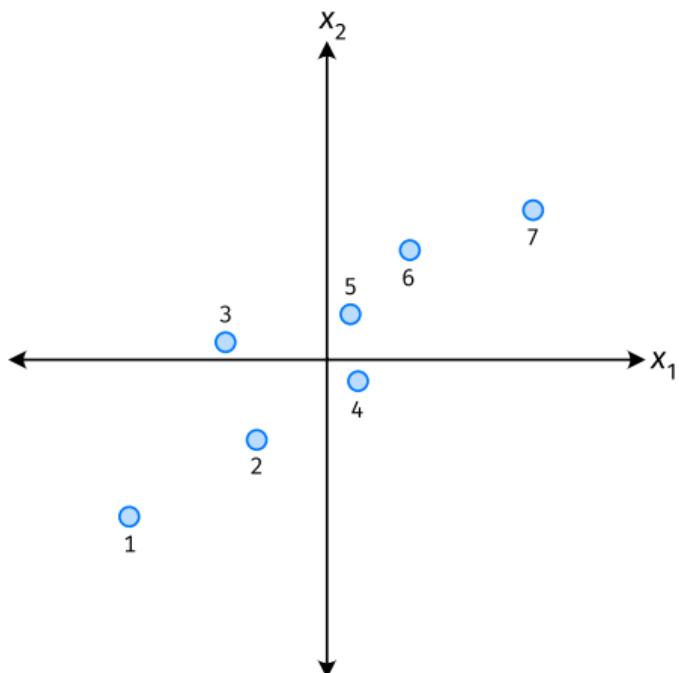


- ▶ z measures how much of \vec{x} is in the direction of \vec{u}
- ▶ If $\vec{u} = (1, 0)^T$, then $z = x_1$
- ▶ If $\vec{u} = (0, 1)^T$, then $z = x_2$

Choosing \vec{u}

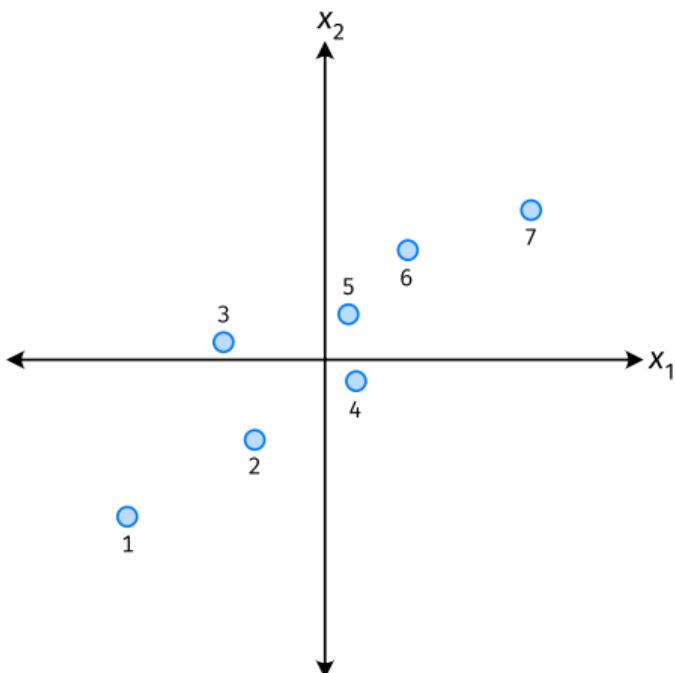
- ▶ Suppose we have only two features:
 - ▶ x_1 : screen size
 - ▶ x_2 : phone thickness
- ▶ We'll create single new feature, z , from x_1 and x_2 .
 - ▶ Assume $z = u_1 x_1 + u_2 x_2 = \vec{x} \cdot \vec{u}$
 - ▶ Interpretation: z is a measure of a phone's size
- ▶ How should we choose $\vec{u} = (u_1,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

Example



- ▶ \vec{u} 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:

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

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 190

Machine Learning: Representations

Lecture 8 | 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

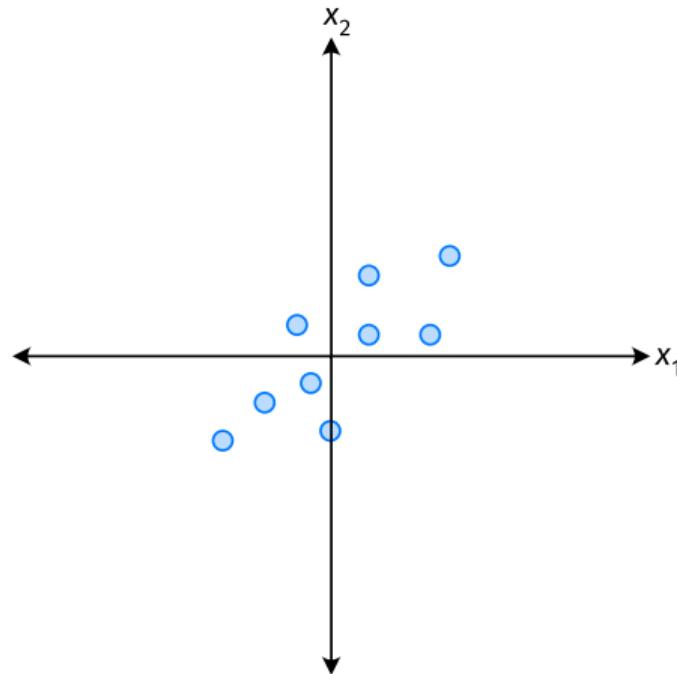
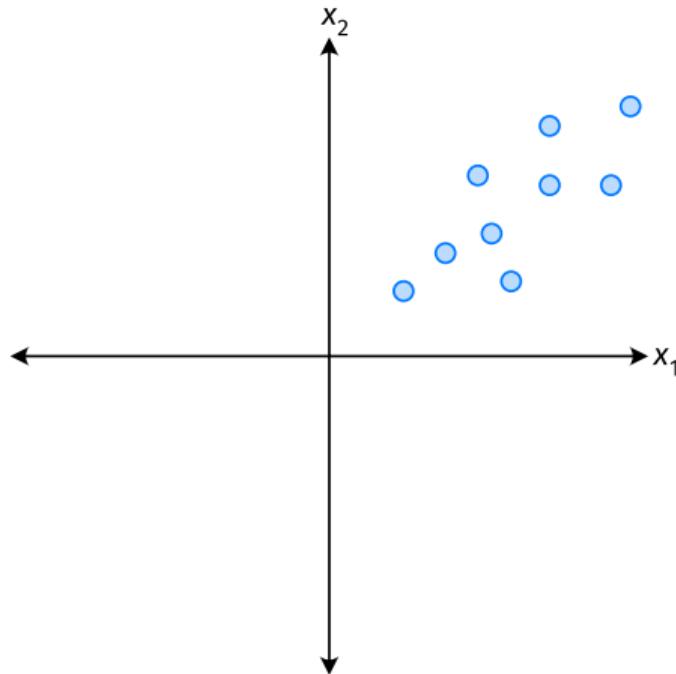
- ▶ Features which share information if they *vary together*.
 - ▶ A.k.a., they “co-vary”
- ▶ Positive association: when one is above average, so is the other
- ▶ Negative association: when one is above average, the other is below average

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:

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

- ▶ 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 follows¹.

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

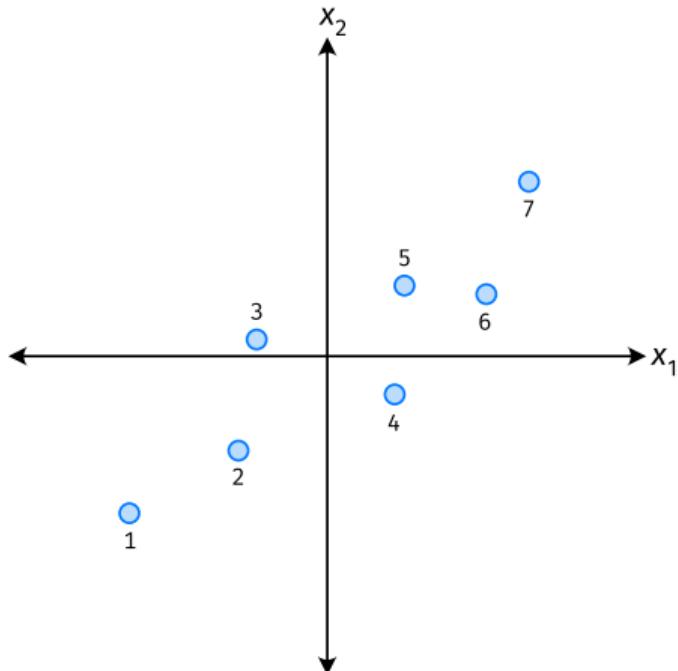
- ▶ For each data point, multiply the value of feature i and feature j , then average these products.
- ▶ This is the **covariance** of features i and j .

¹Assuming centered data

Quantifying Covariance

- ▶ Assume the data are **centered**.

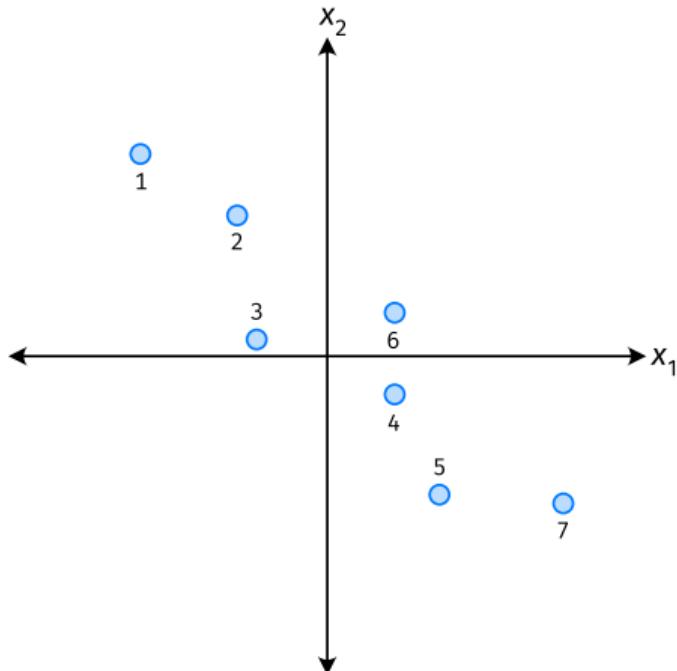
$$\text{Covariance} = \frac{1}{7} \sum_{i=1}^7 \vec{x}_1^{(i)} \times \vec{x}_2^{(i)}$$



Quantifying Covariance

- ▶ Assume the data are **centered**.

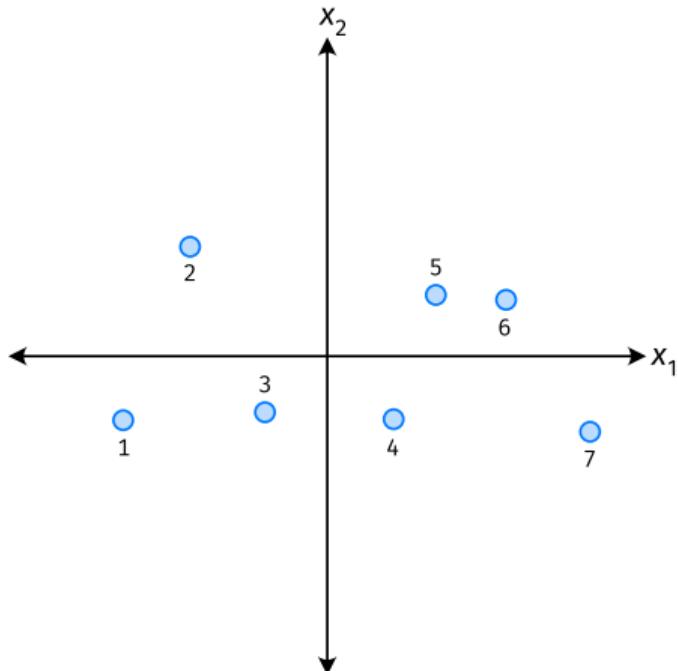
$$\text{Covariance} = \frac{1}{7} \sum_{i=1}^7 \vec{x}_1^{(i)} \times \vec{x}_2^{(i)}$$



Quantifying Covariance

- ▶ Assume the data are **centered**.

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



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)²

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

²Or use np.cov

DSC 190

Machine Learning: Representations

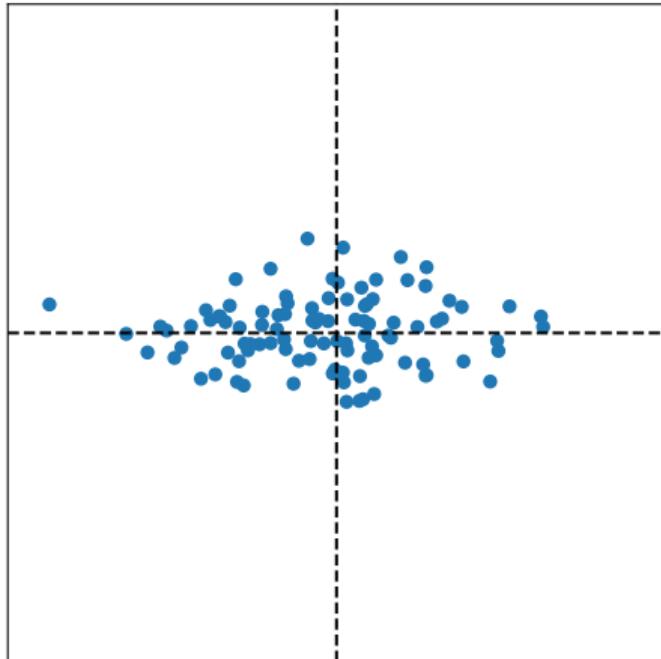
Lecture 8 | Part 3

Visualizing Covariance Matrices

Visualizing Covariance Matrices

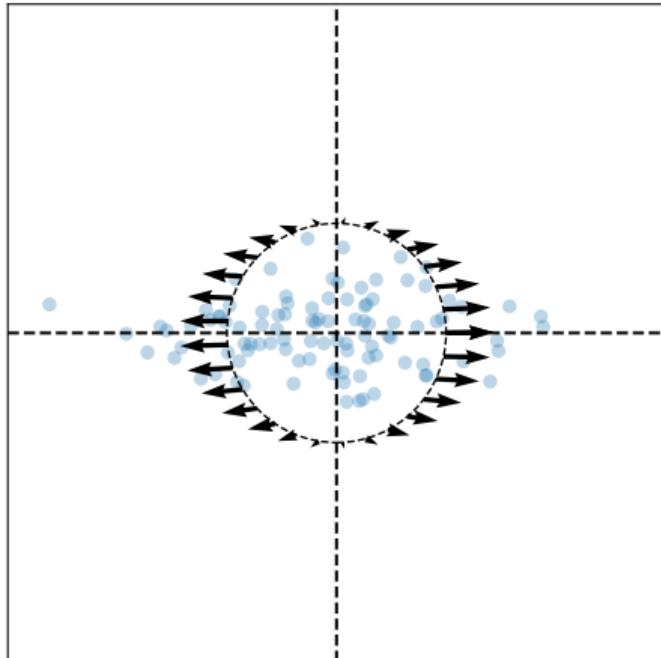
- ▶ Covariance matrices are symmetric.
- ▶ They have axes of symmetry (eigenvectors and eigenvalues).
- ▶ What are they?

Visualizing Covariance Matrices



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

Visualizing Covariance Matrices

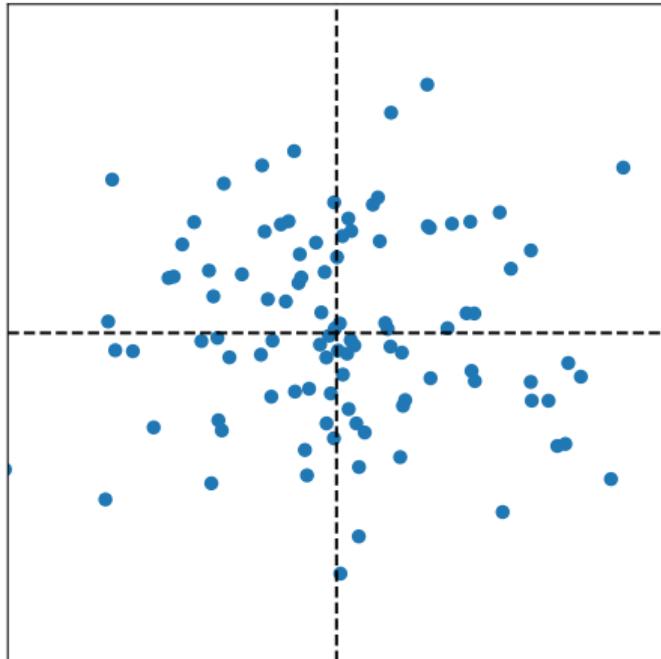


Eigenvectors:

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

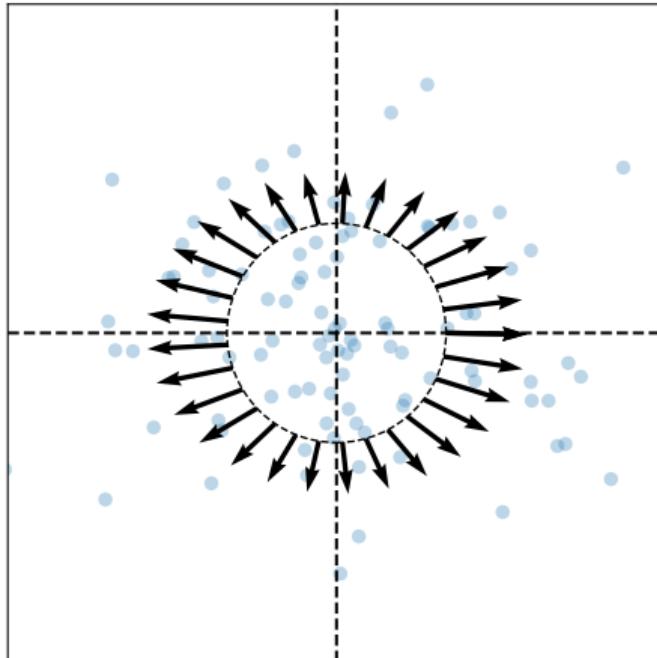
$$\vec{u}^{(2)} \approx$$

Visualizing Covariance Matrices



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

Visualizing Covariance Matrices

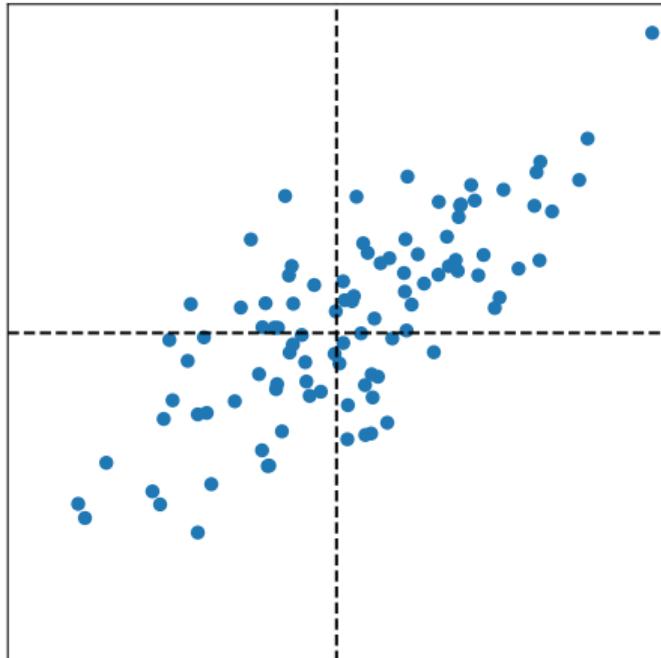


Eigenvectors:

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

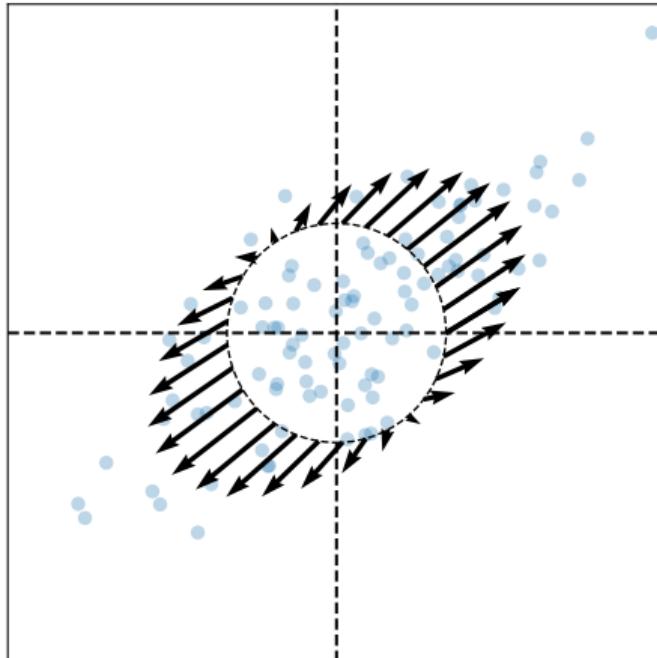
$$\vec{u}^{(2)} \approx$$

Visualizing Covariance Matrices



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

Visualizing Covariance Matrices



Eigenvectors:

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

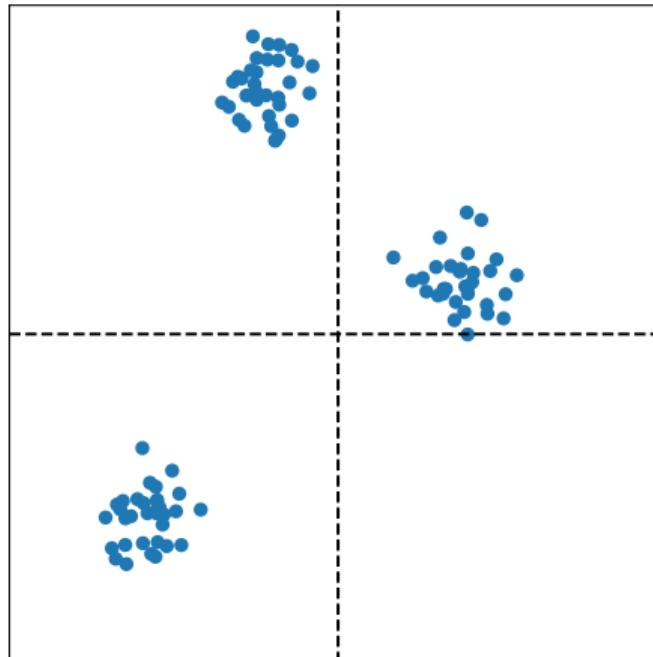
$$\vec{u}^{(2)} \approx$$

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.

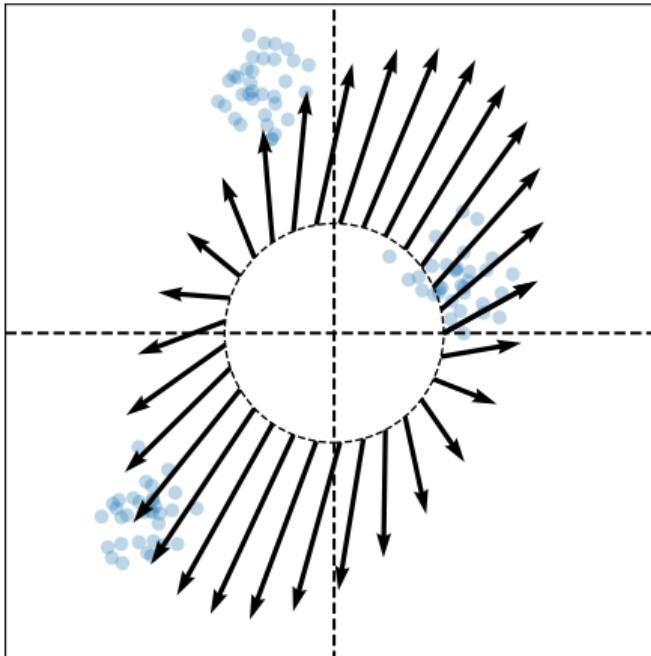
Caution

- ▶ 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.



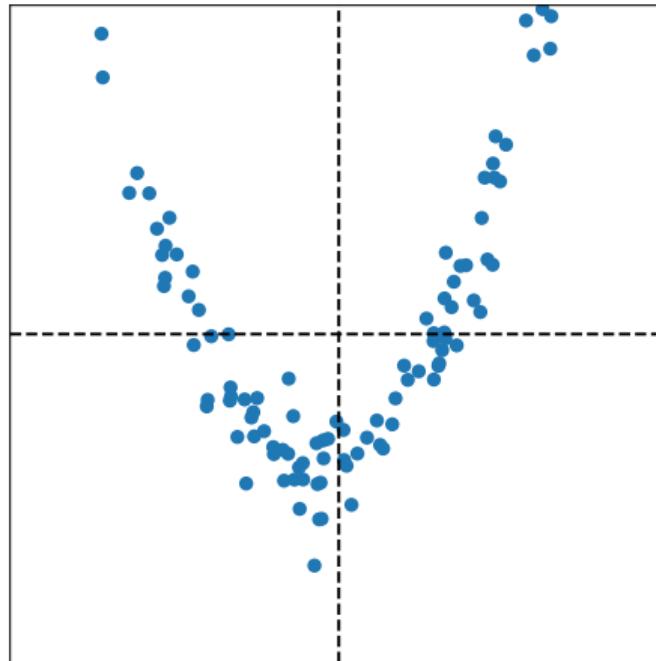
Caution

- ▶ 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.



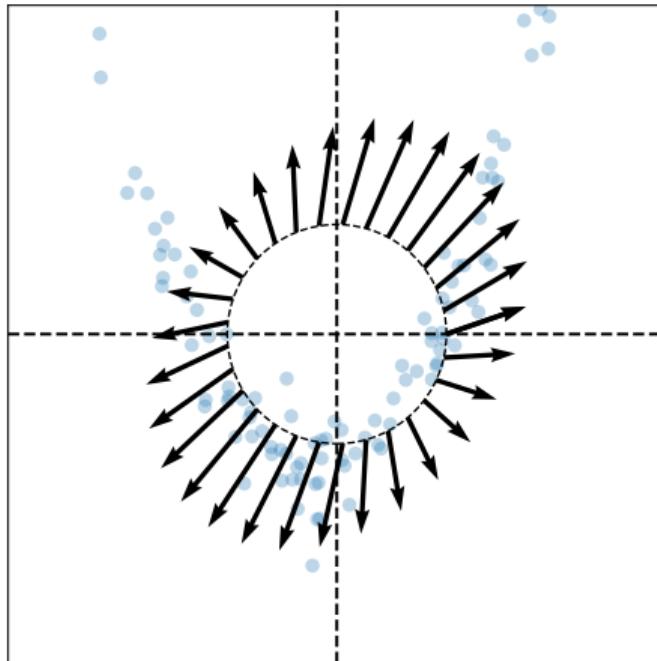
Caution

- ▶ 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 190

Machine Learning: Representations

Lecture 8 | 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...

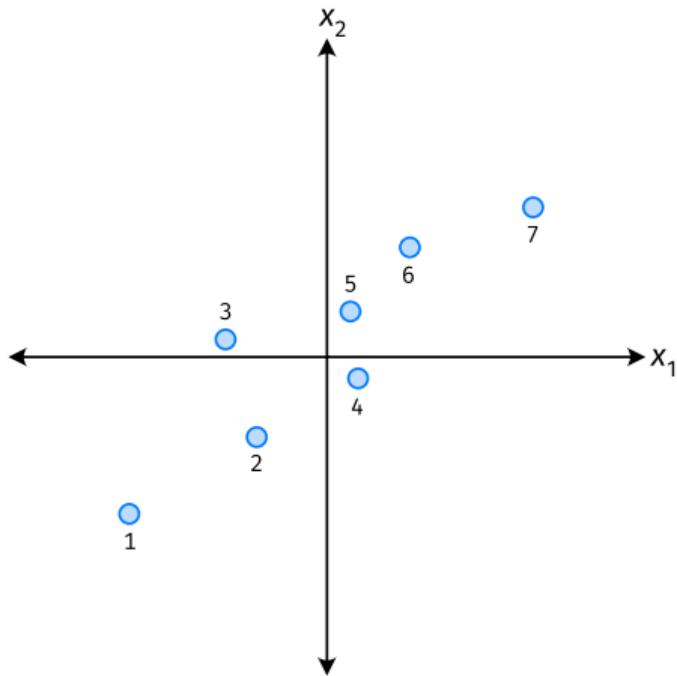
- ▶ We haven't actually defined “direction of maximum variance”
- ▶ Let's derive PCA more formally.

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 $\vec{x}^{(1)}, \dots, \vec{x}^{(n)} \in \mathbb{R}^d$

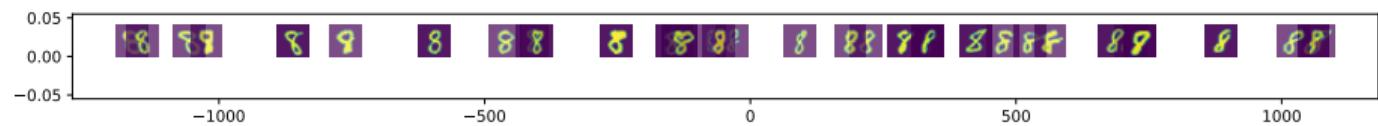
1. Compute the covariance matrix, C .
2. Compute the top eigenvector \vec{u} , of C .
3. For $i \in \{1, \dots, n\}$, create new feature:

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

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 190

Machine Learning: Representations

Lecture 9 | 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...

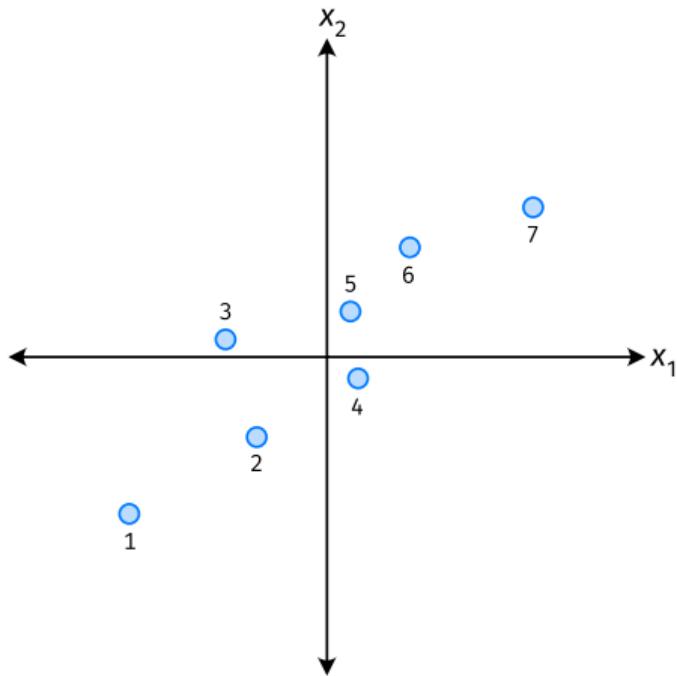
- ▶ We haven't actually defined “direction of maximum variance”
- ▶ Let's derive PCA more formally.

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:

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

- 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:

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

DSC 190

Machine Learning: Representations

Lecture 9 | 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 .

Today: 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 .

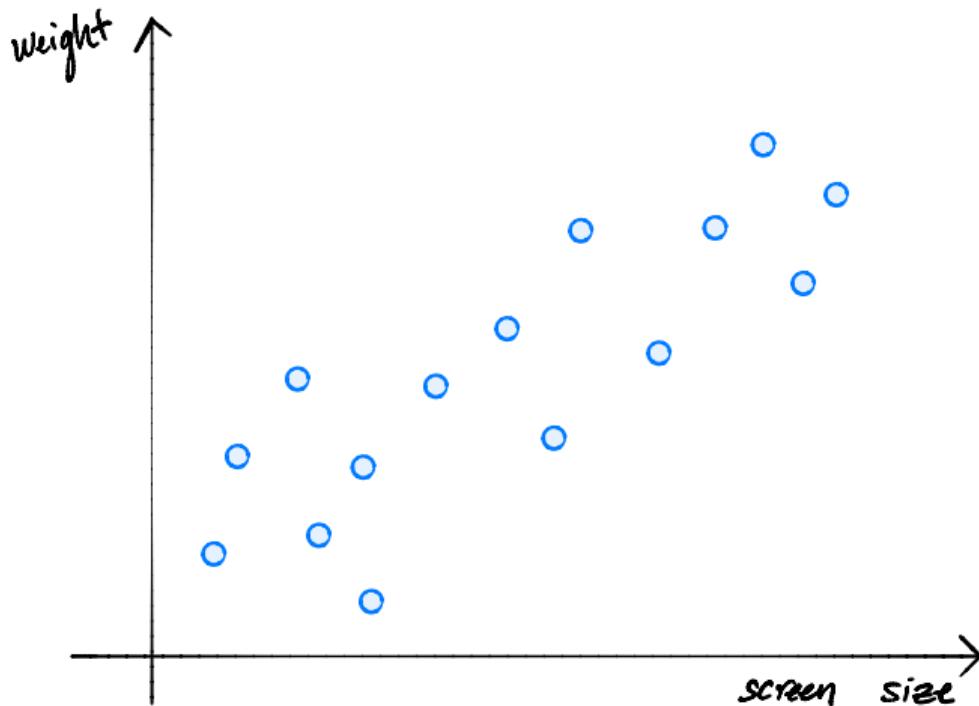
A Second Feature

- ▶ Make same assumption for second feature:

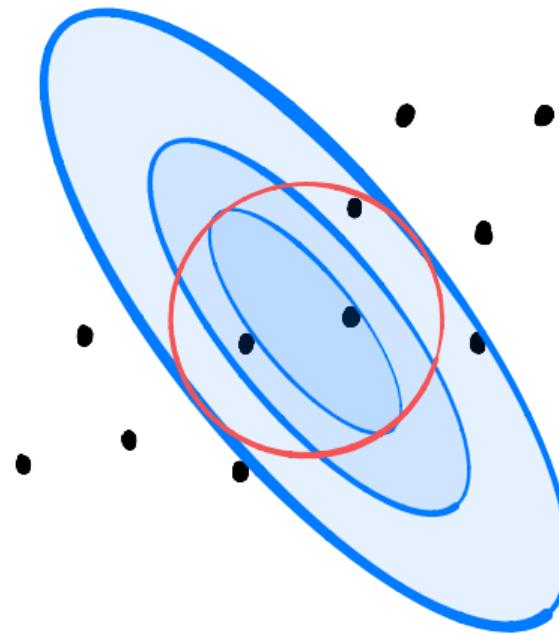
$$z_2 = \vec{u}^{(2)} \cdot \vec{x} = u_1^{(2)} x_1 + \dots + u_d^{(2)} x_d$$

- ▶ 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.

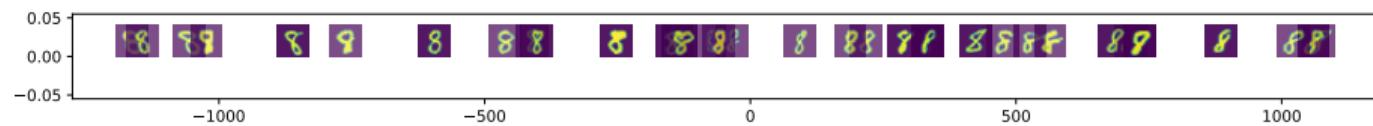
PCA: Two Components

- ▶ 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:

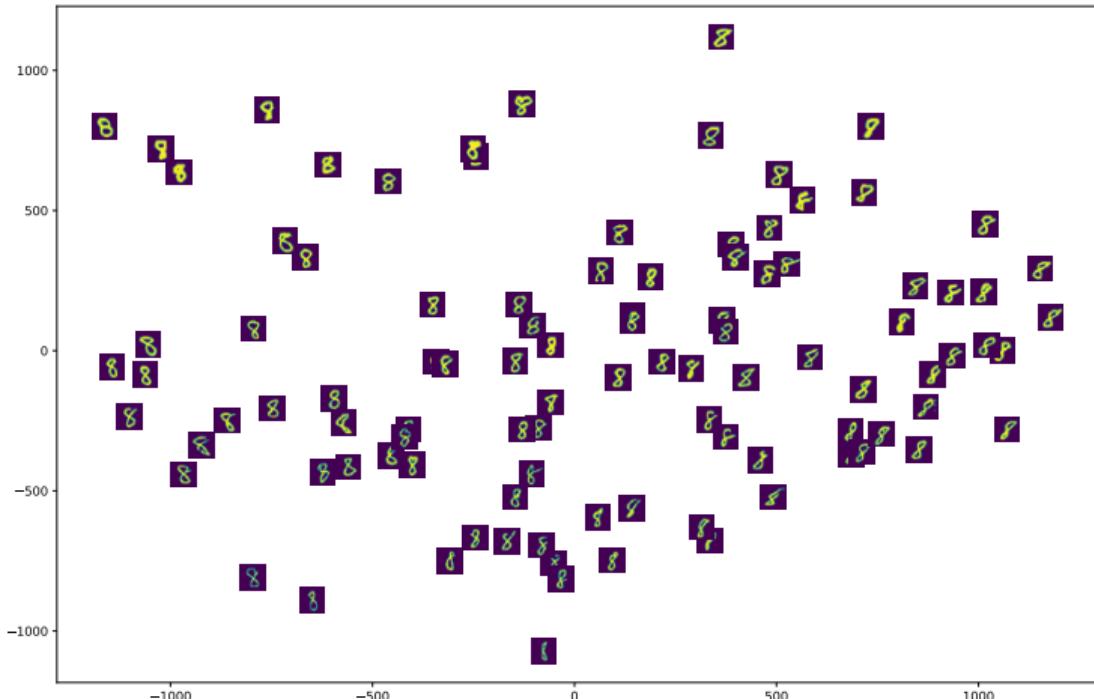
$$z_1 = \vec{x} \cdot \vec{u}^{(1)}$$

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

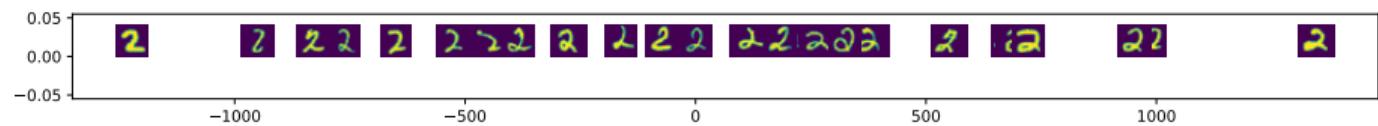
Example



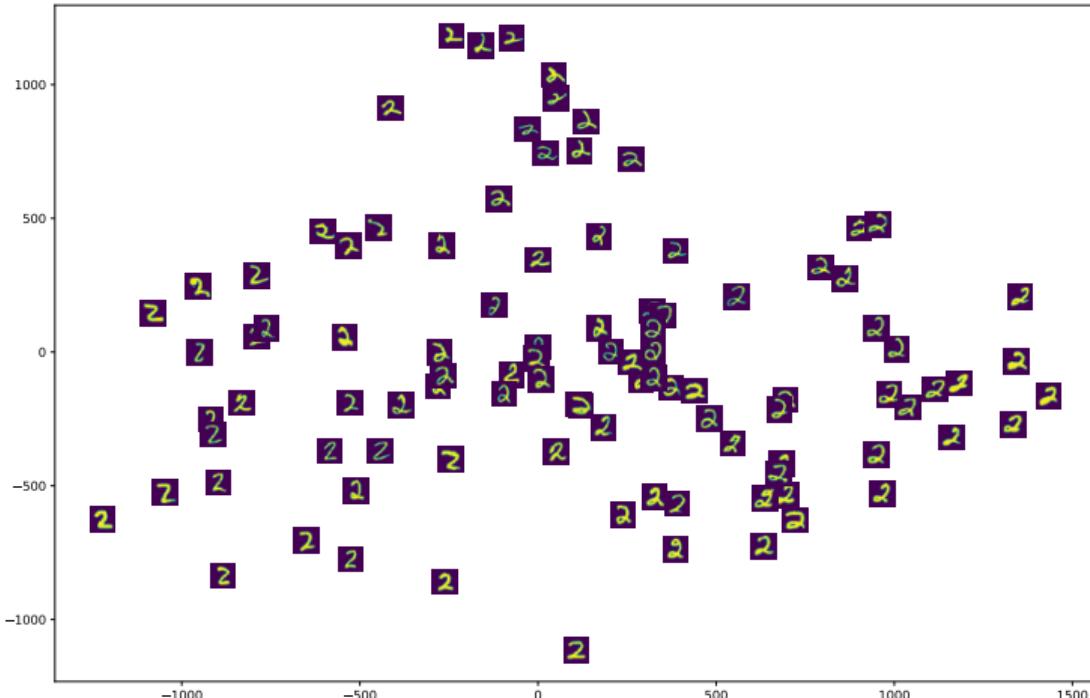
Example



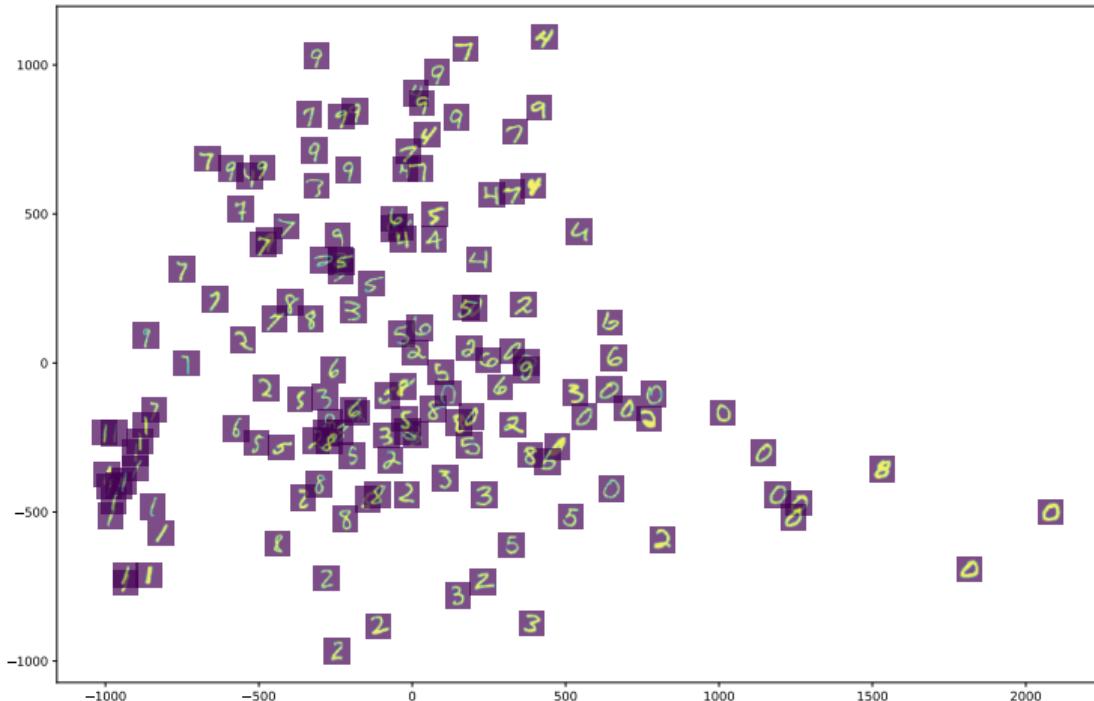
Example



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:

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

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

⋮

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

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 190

Machine Learning: Representations

Lecture 9 | Part 3

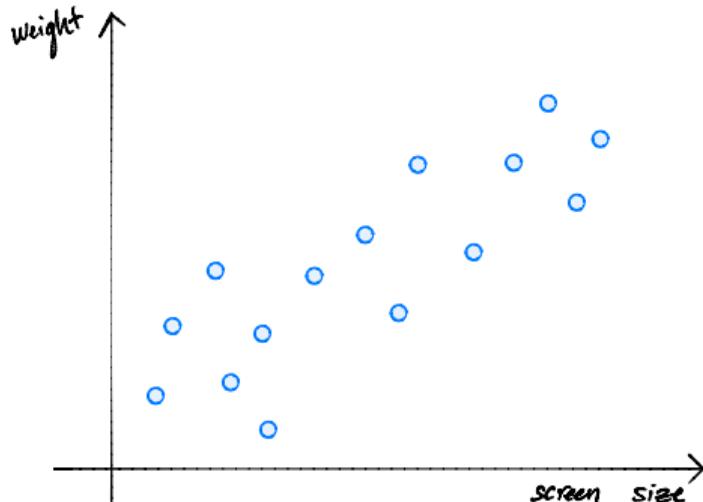
Reconstructions

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

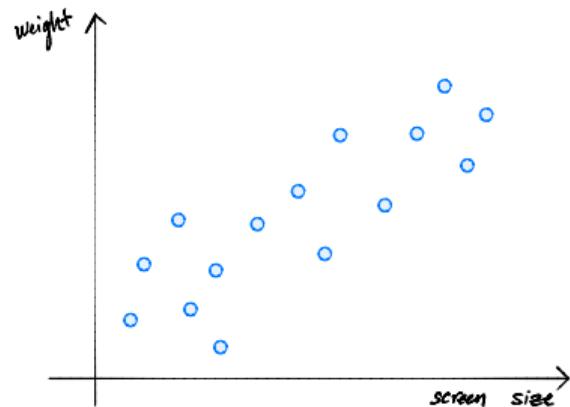
- ▶ The reconstruction *approximates* the original point, \vec{x} .

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

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

- ▶ Total reconstruction error:

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



DSC 190

Machine Learning: Representations

Lecture 9 | 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”

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

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

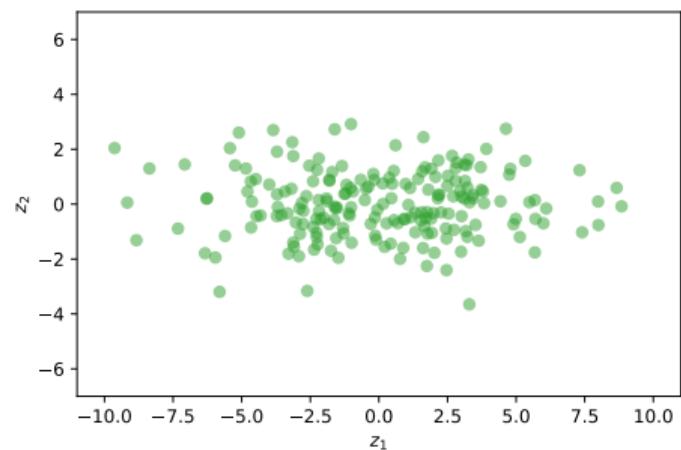
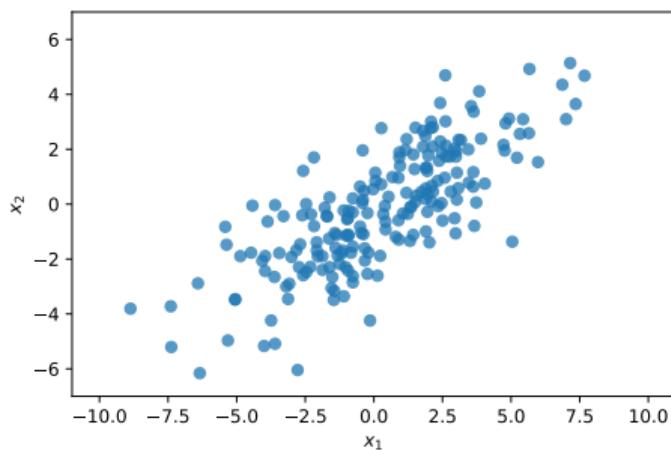
- ▶ Goal: minimize total reconstruction error.
- ▶ Claim: Choosing U to be top eigenvectors of C minimizes reconstruction error among all choices of orthonormal U

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.



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 190

Machine Learning: Representations

Lecture 9 | Part 5

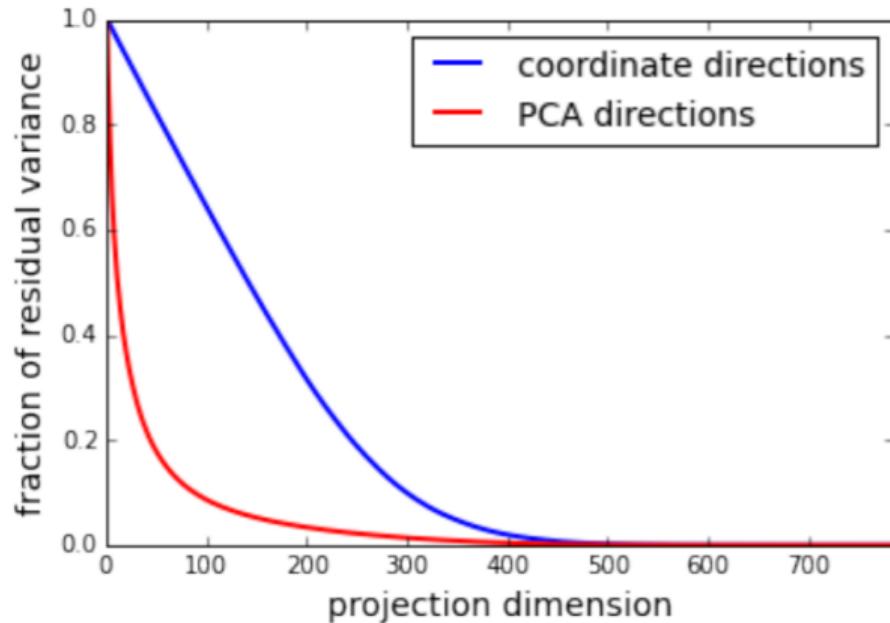
PCA in Practice

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 190

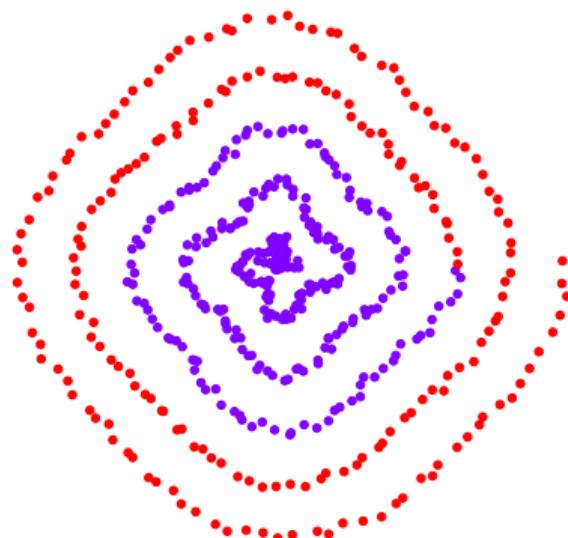
Machine Learning: Representations

Lecture 10 | Part 1

Nonlinear Dimensionality Reduction

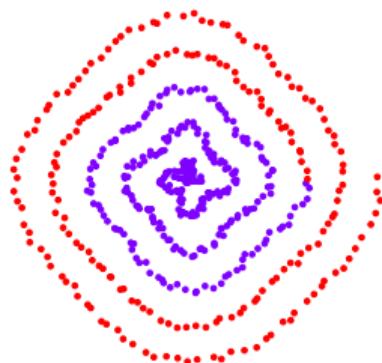
Scenario

- ▶ 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?

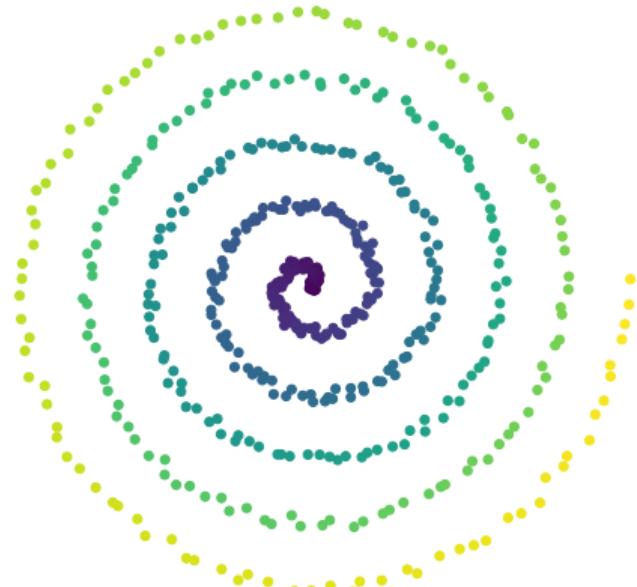
- ▶ PCA simply “rotates” the data.
- ▶ No amount of rotation will “unroll” the spiral.
- ▶ We need a fundamentally different approach that works for non-linear patterns.

Today

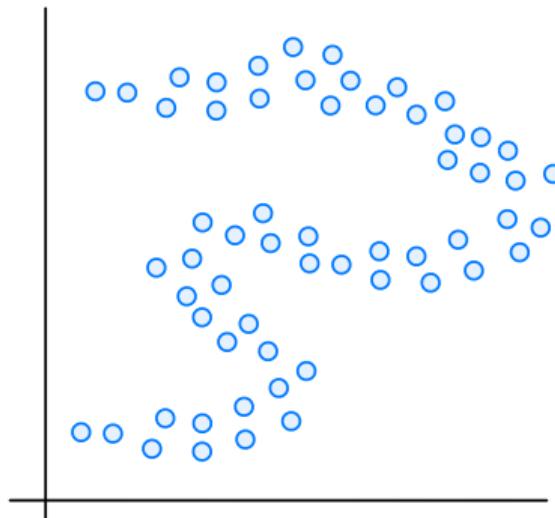
- ▶ Non-linear dimensionality reduction via
spectral embeddings.

Rethinking Dimensionality

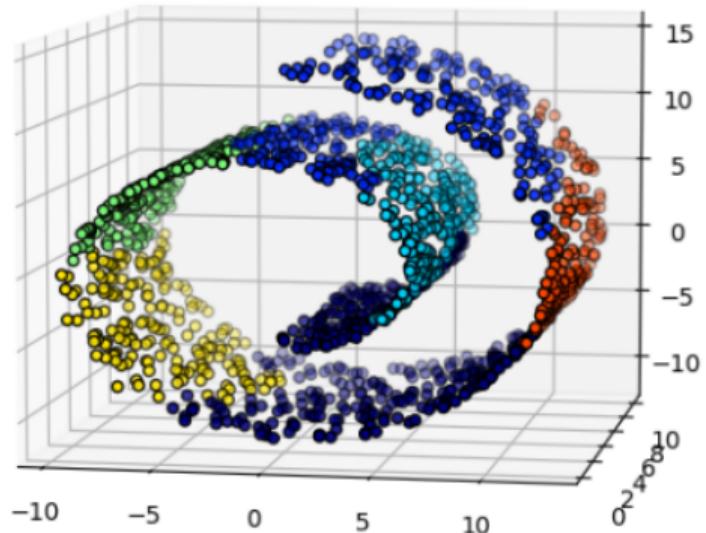
- ▶ Each point is an (x, y) coordinate in two dimensional space
- ▶ But the structure is one-dimensional
- ▶ Could (roughly) locate point using one number: distance from end.



Rethinking Dimensionality



Rethinking Dimensionality

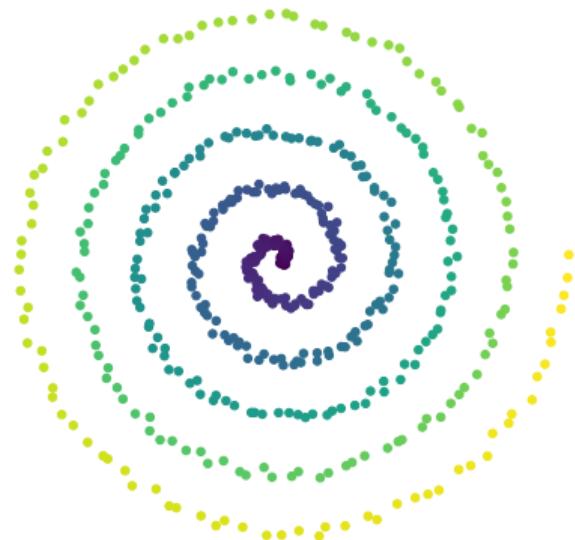


Rethinking Dimensionality

- ▶ 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

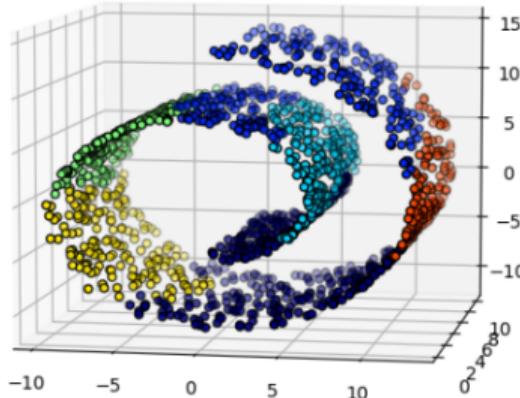
Example

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



Example

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



Example

- ▶ Ambient dimension:
- ▶ Intrinsic dimension:



Manifold Learning

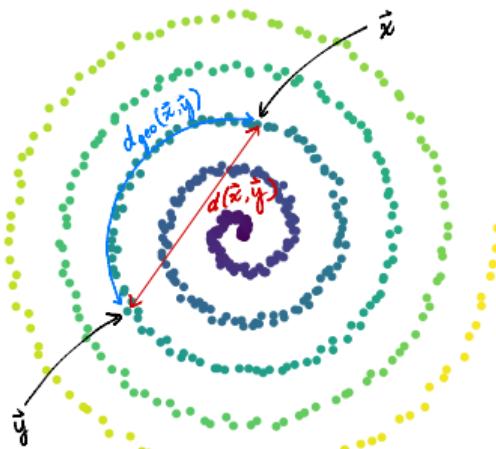
- ▶ **Given:** data in high dimensions
- ▶ **Recover:** the low-dimensional manifold

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

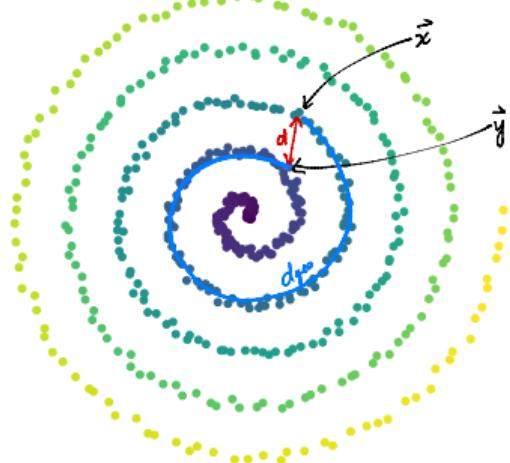
Euclidean vs. Geodesic Distances

- ▶ **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



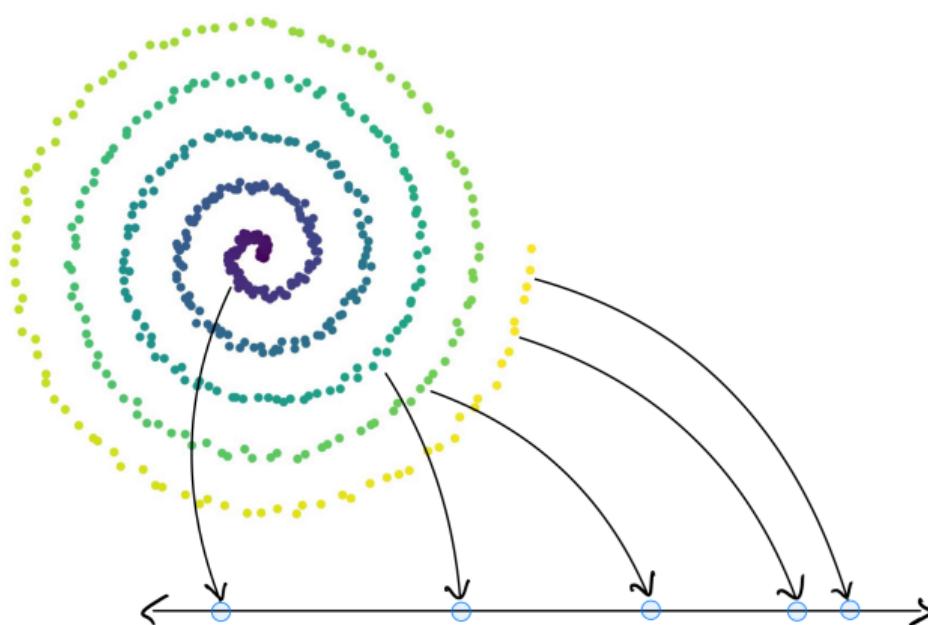
Euclidean vs. Geodesic Distances

- ▶ 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 190

Machine Learning: Representations

Lecture 10 | Part 2

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

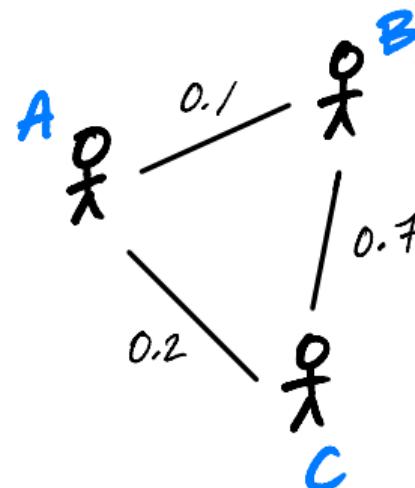
Idea

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

Similarity Graphs

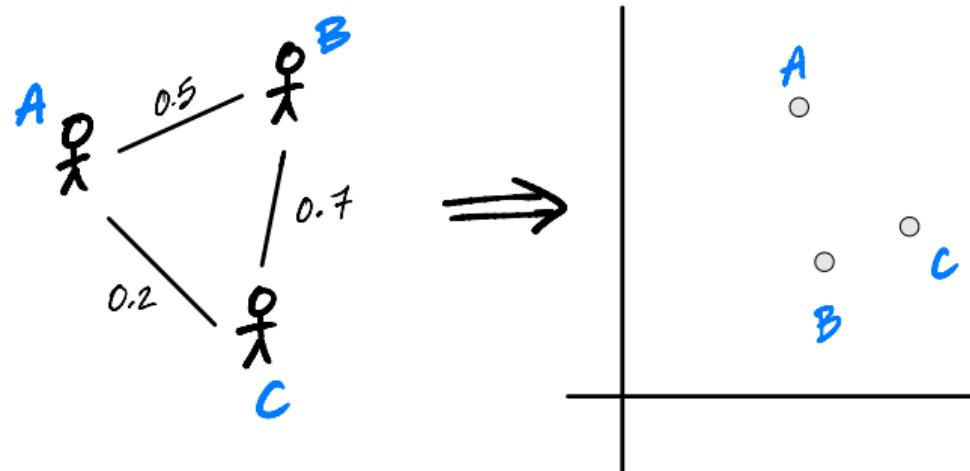
- ▶ Similarity matrices can be thought of as weighted graphs, and vice versa.

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



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

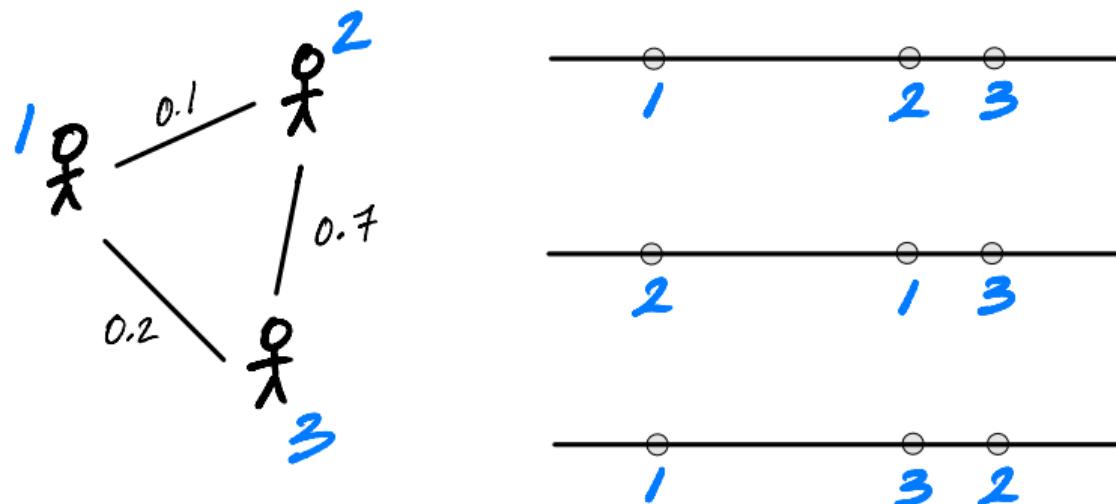
$$\vec{f} = (1, 3, 2, -4)^T$$

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.

DSC 190

Machine Learning: Representations

Lecture 10 | Part 3

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

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:

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

- ▶ Proof: expand both sides

Proof

DSC 190

Machine Learning: Representations

Lecture 10 | Part 4

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

- ▶ **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

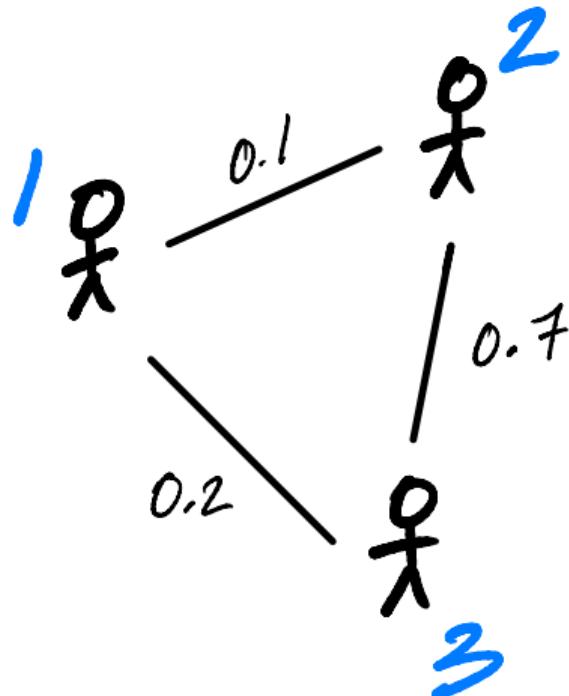
$$\text{Cost}(\vec{f}) = \sum_{i=1}^n \sum_{j=1}^n w_{ij}(f_i - f_j)^2 = \frac{1}{2} \vec{f}^T L \vec{f}$$

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



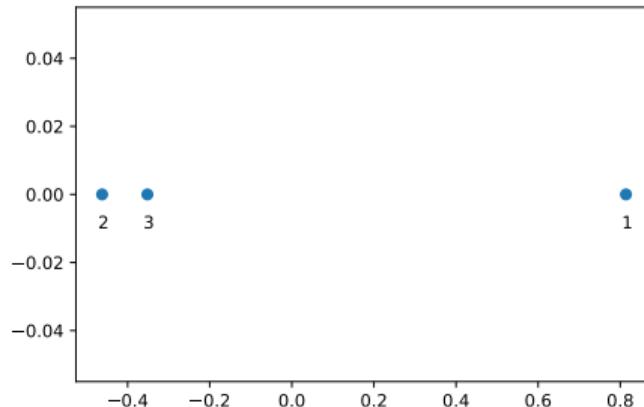
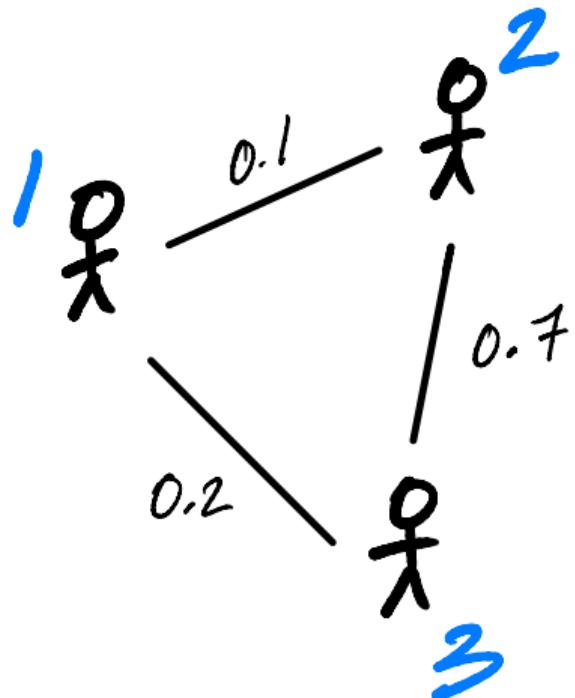
```
W = np.array([
    [1, 0.1, 0.2],
    [0.1, 1, 0.7],
    [0.2, 0.7, 1]
])

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

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

f = vecs[:, 1]
```

Example



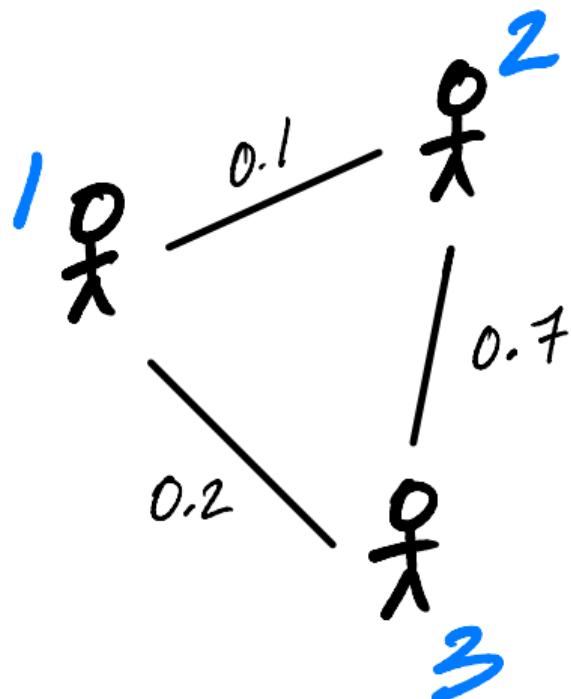
Embedding into \mathbb{R}^k

- ▶ 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



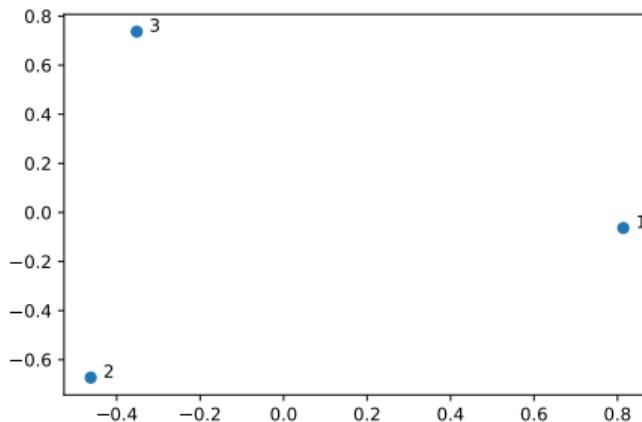
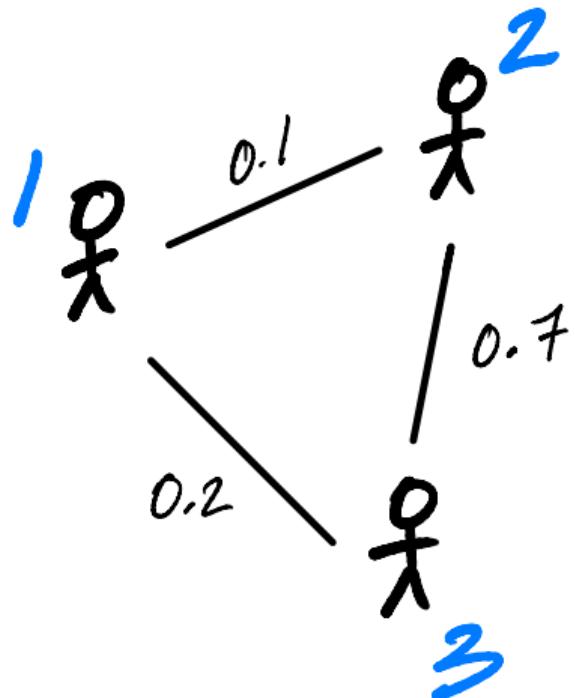
```
W = np.array([
    [1, 0.1, 0.2],
    [0.1, 1, 0.7],
    [0.2, 0.7, 1]
])

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



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 190

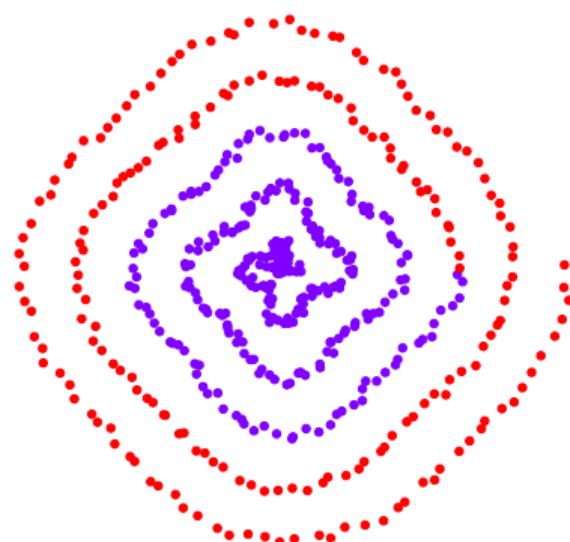
Machine Learning: Representations

Lecture 11 | Part 1

Nonlinear Dimensionality Reduction

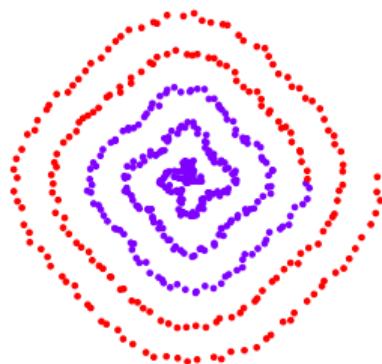
Scenario

- ▶ 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

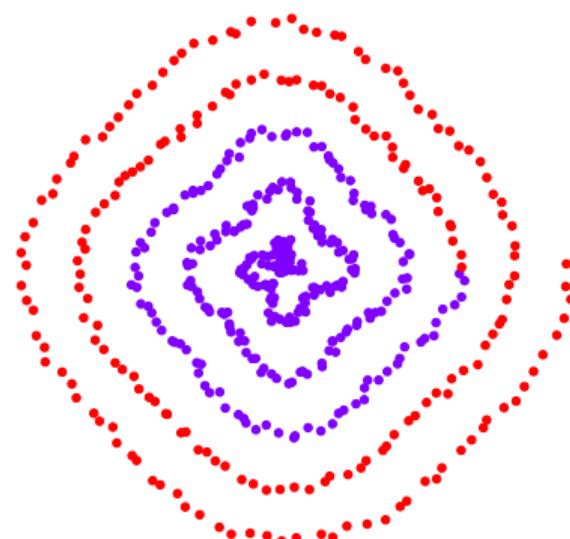
- ▶ Non-linear dimensionality reduction via
spectral embeddings.

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 190

Machine Learning: Representations

Lecture 11 | Part 2

From Points to Graphs

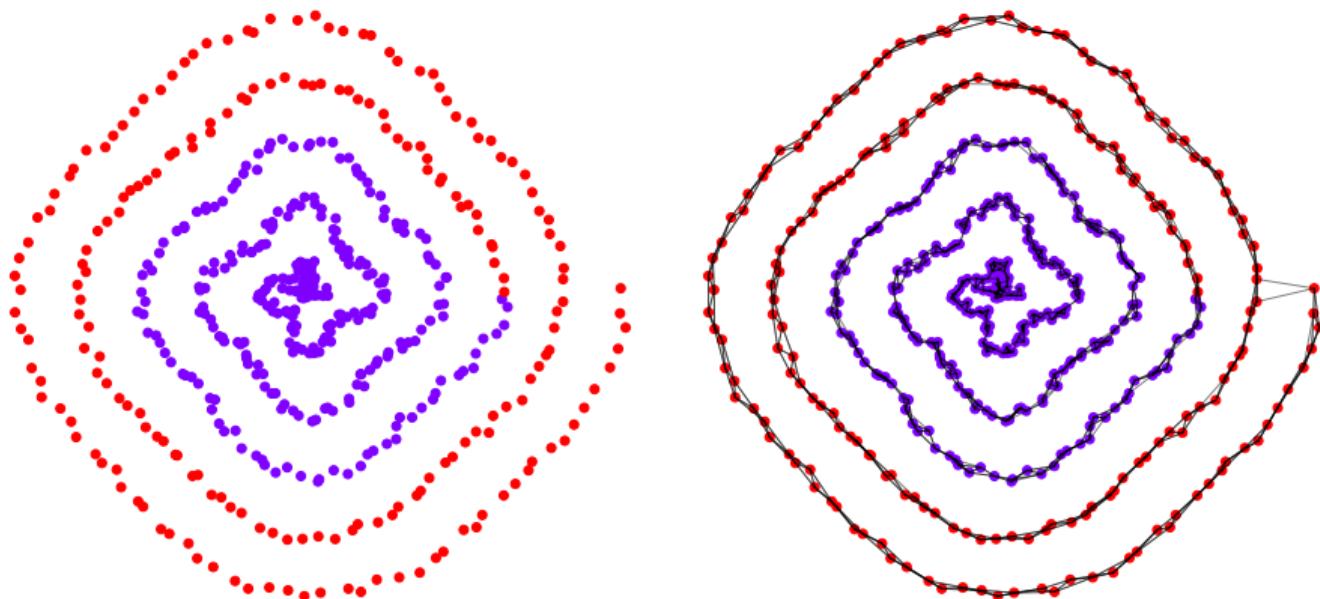
Dimensionality Reduction

- ▶ **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?

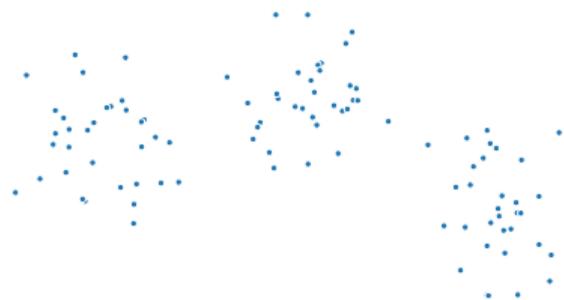


Three Approaches

- ▶ 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

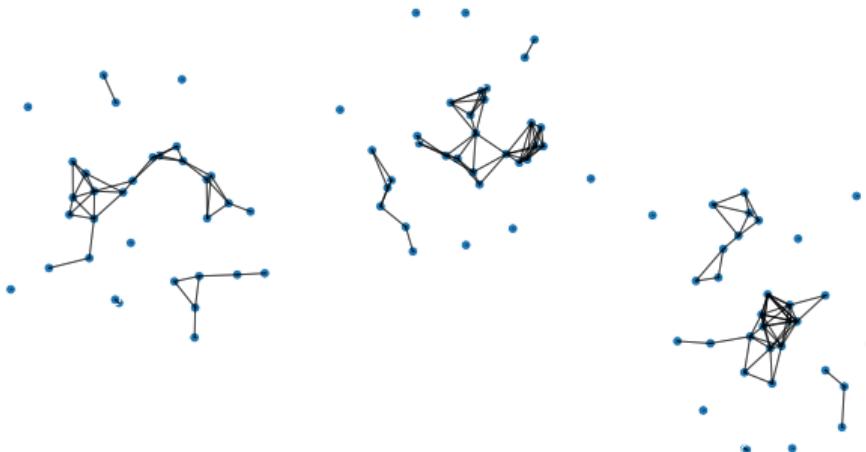


Exercise

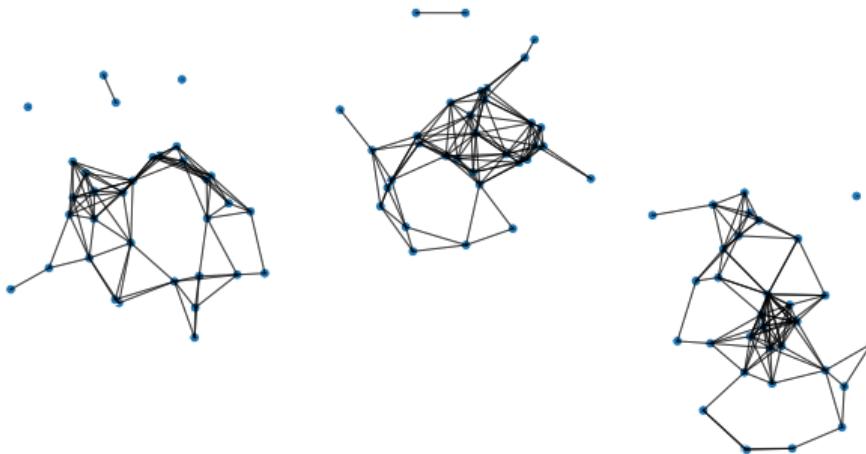
What will the graph look like when ε is small? What about when it is large?



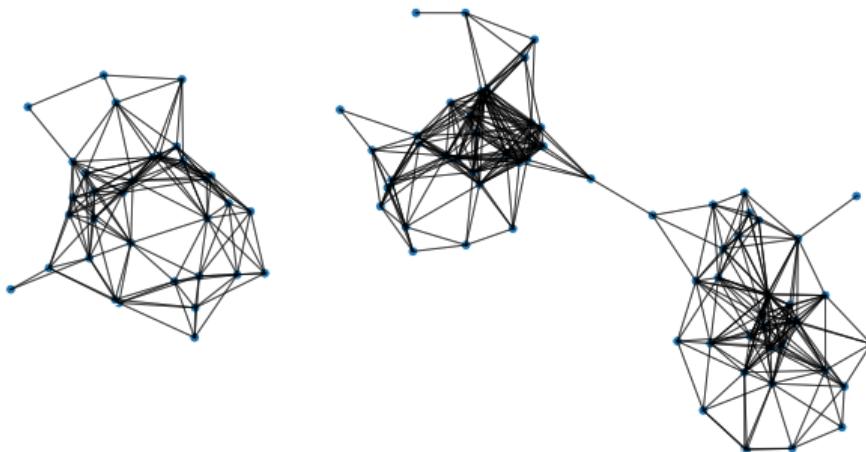
Epsilon Neighbors Graph



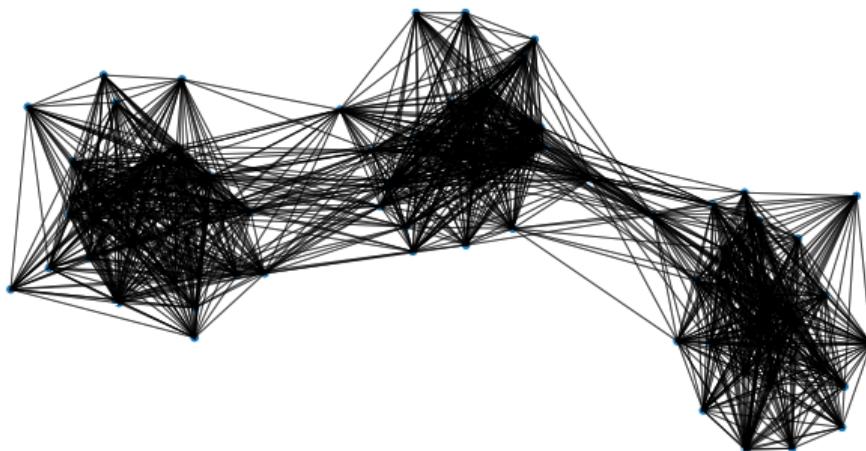
Epsilon Neighbors Graph



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)
for i in range(n):
    for j in range(n):
        if distance(X[i], X[j]) <= epsilon:
            adj[i, j] = 1
```

Picking ε

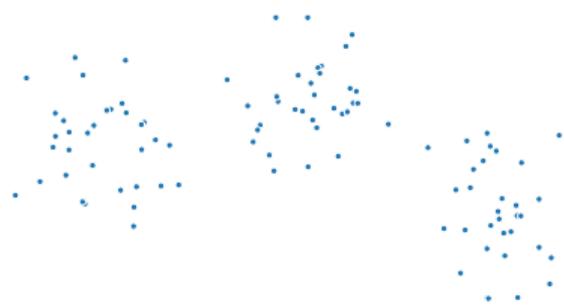
- ▶ If ε is too small, graph is underconnected
- ▶ If ε is too large, graph is overconnected
- ▶ If you cannot visualize, just try and see

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



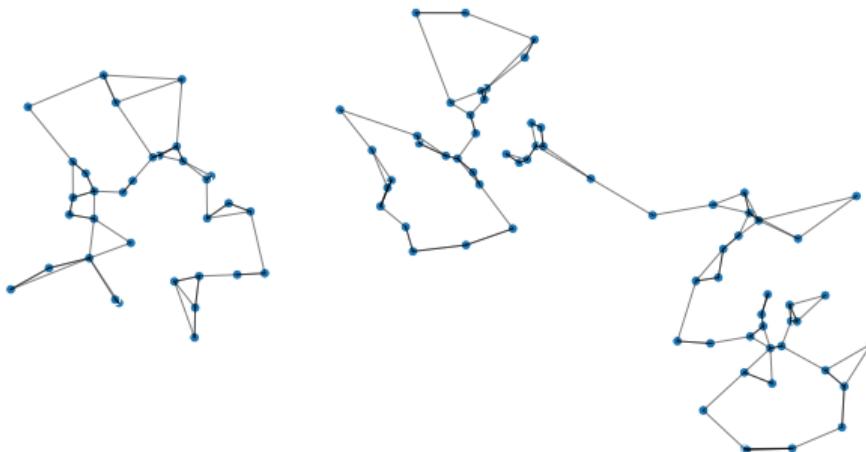
k-Neighbors: Pseudocode

```
# assume the data is in X
n = len(X)
adj = np.zeros_like(X)
for i in range(n):
    for j in k_nearest_neighbors(X, i):
        adj[i, j] = 1
```

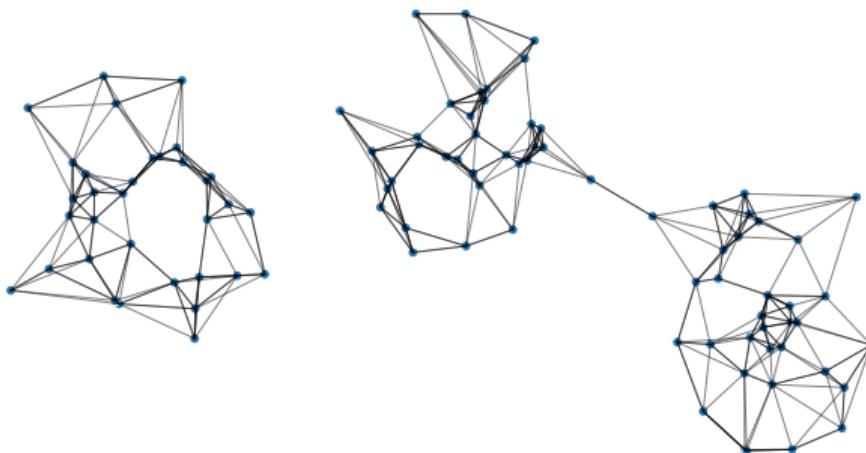
Exercise

Is it possible for a k -neighbors graph to be disconnected?

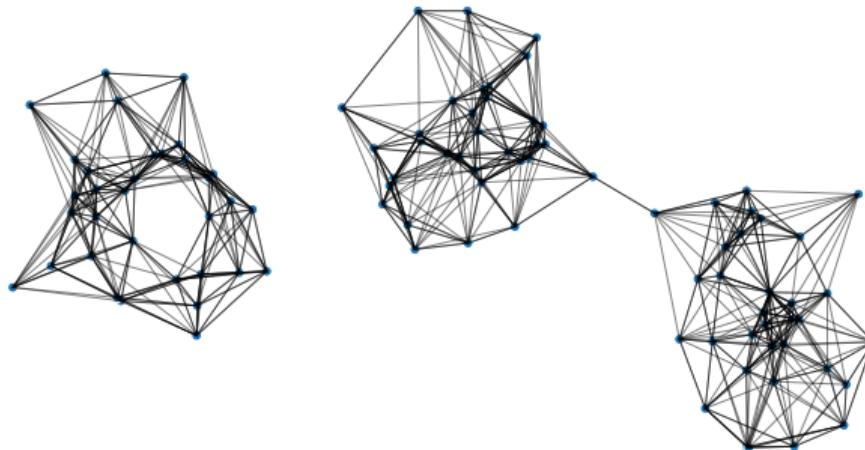
k-Neighbors Graph



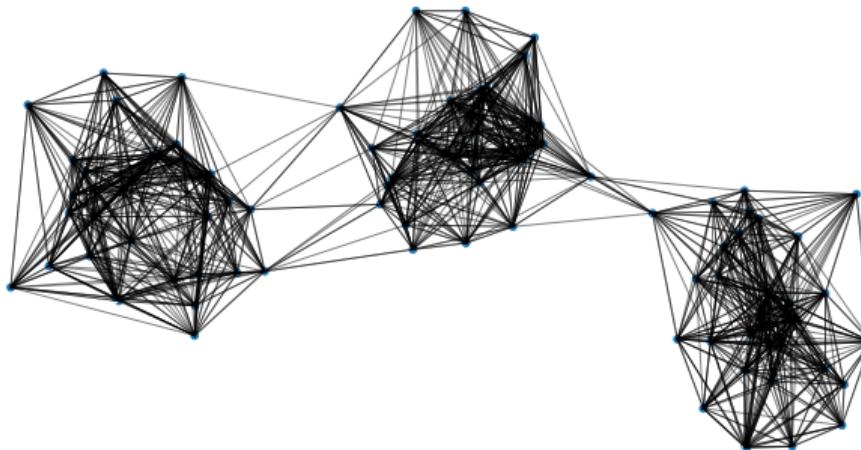
k-Neighbors Graph



k-Neighbors Graph



k-Neighbors Graph

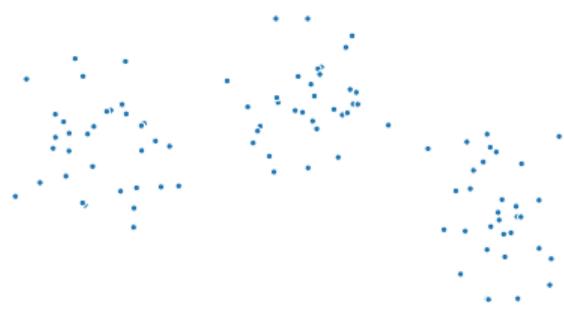


With scikit-learn

```
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

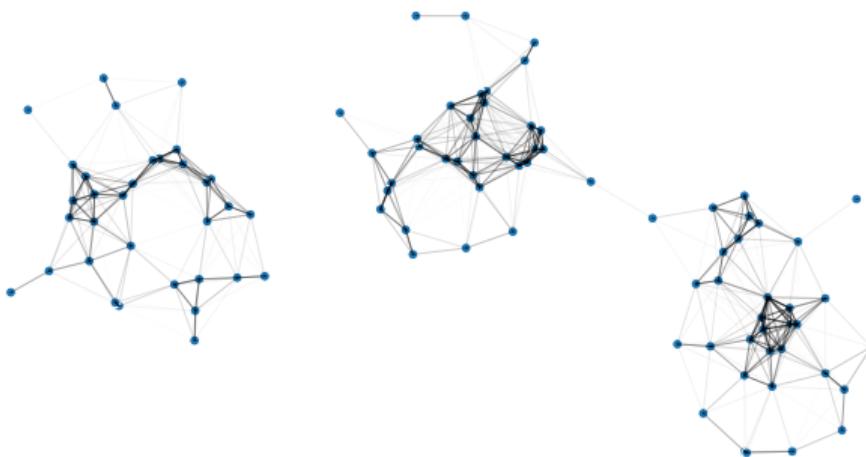
```
def h(x, y):
    dist = np.linalg.norm(x, y)
    return np.exp(-dist**2 / sigma**2)

# assume the data is in X
n = len(X)
w = np.ones_like(X)
for i in range(n):
    for j in range(n):
        w[i, j] = h(X[i], X[j])
```

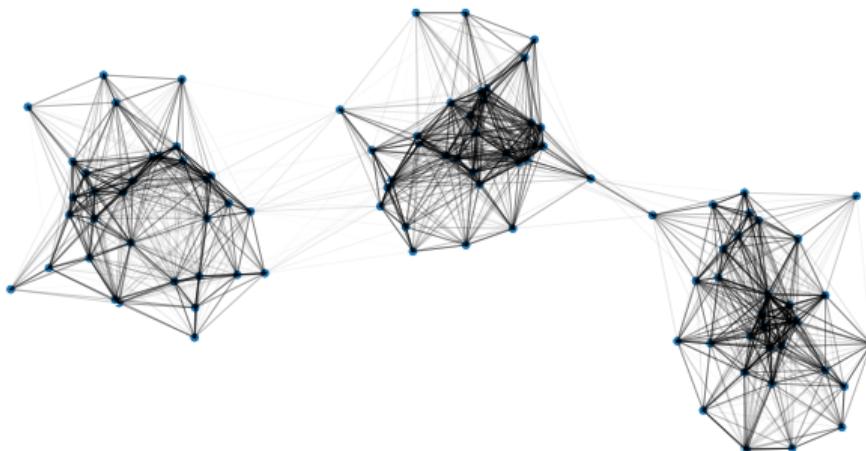
With SciPy

```
distances = scipy.spatial.distance_matrix(X, X)
w = np.exp(-distances**2 / sigma**2)
```

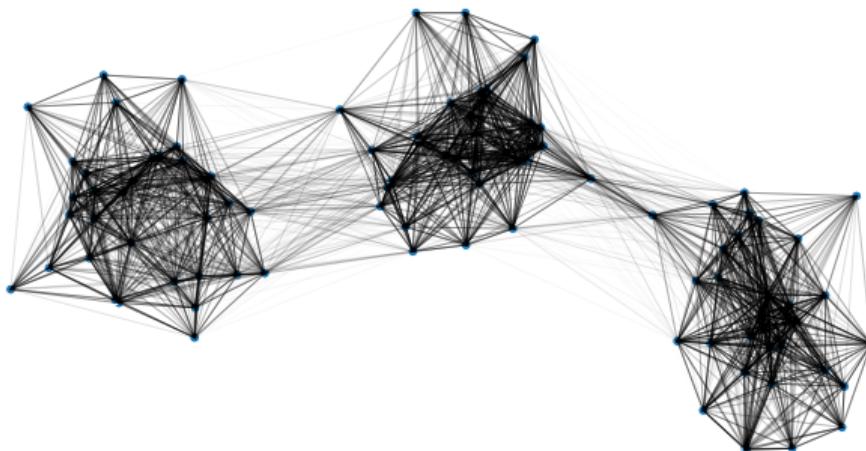
Gaussian Similarity



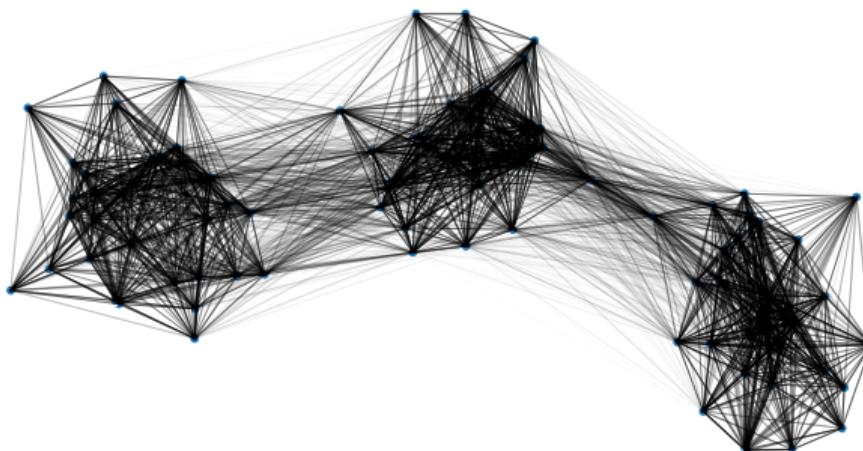
Gaussian Similarity



Gaussian Similarity



Gaussian Similarity



DSC 190

Machine Learning: Representations

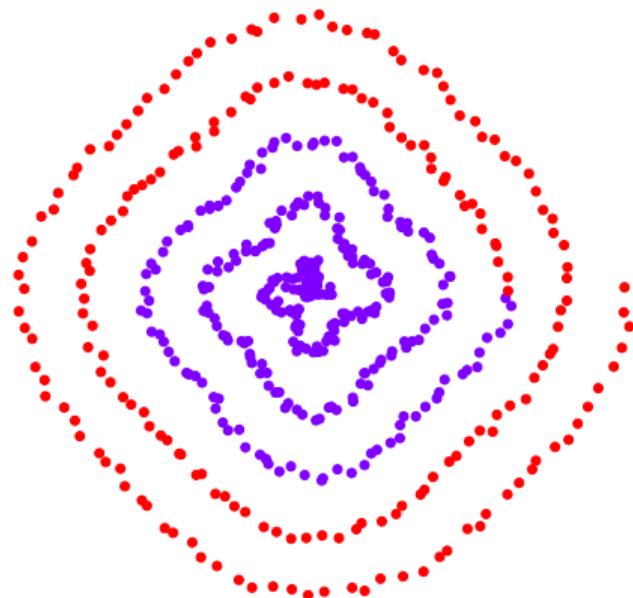
Lecture 11 | Part 3

Laplacian Eigenmaps

Idea

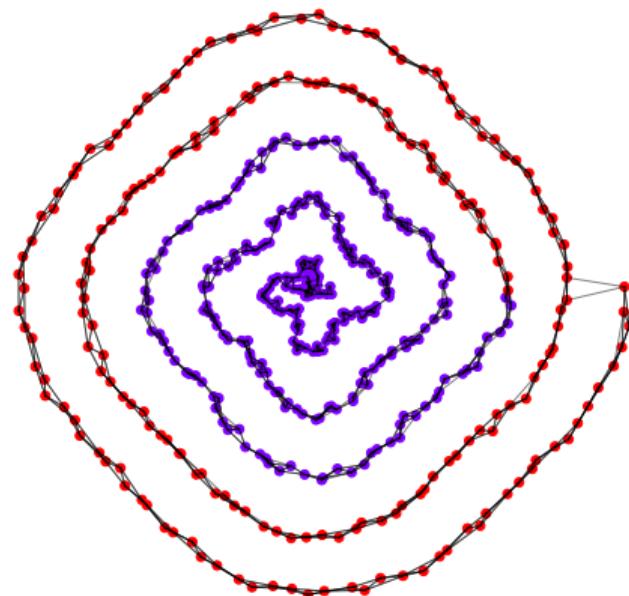
- ▶ Build a similarity graph from points in \mathbb{R}^2
 - ▶ epsilon neighbors, k -neighbors, or fully connected
- ▶ Now: use approach from last lecture to embed into \mathbb{R}^k

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 k non-constant eigenvectors of L , use as embedding

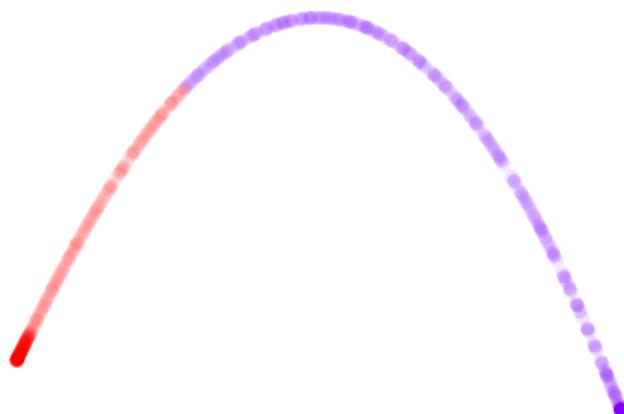
Example 1: Spiral

- ▶ Embedding into \mathbb{R}^1



Example 1: Spiral

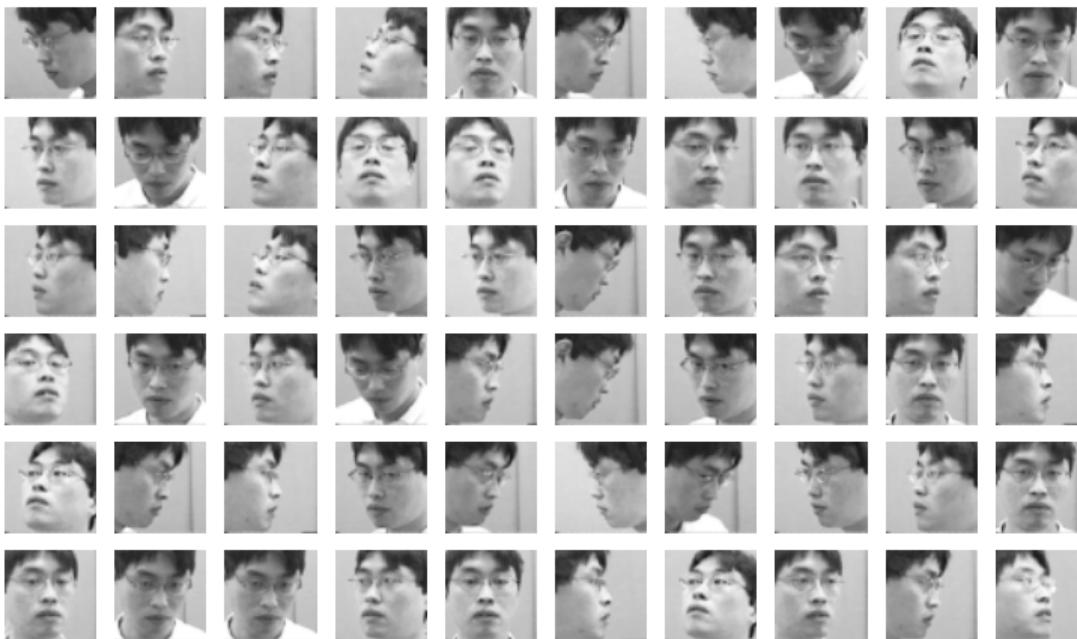
- ▶ Embedding into \mathbb{R}^2



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
)
```

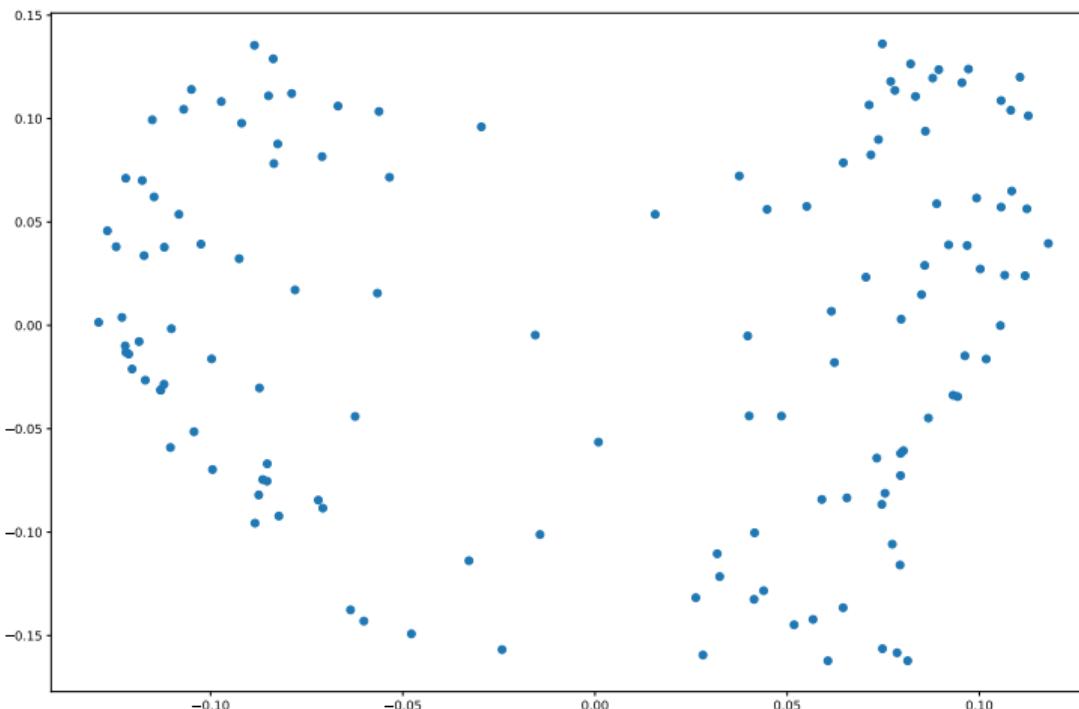
Example 2: Face Pose



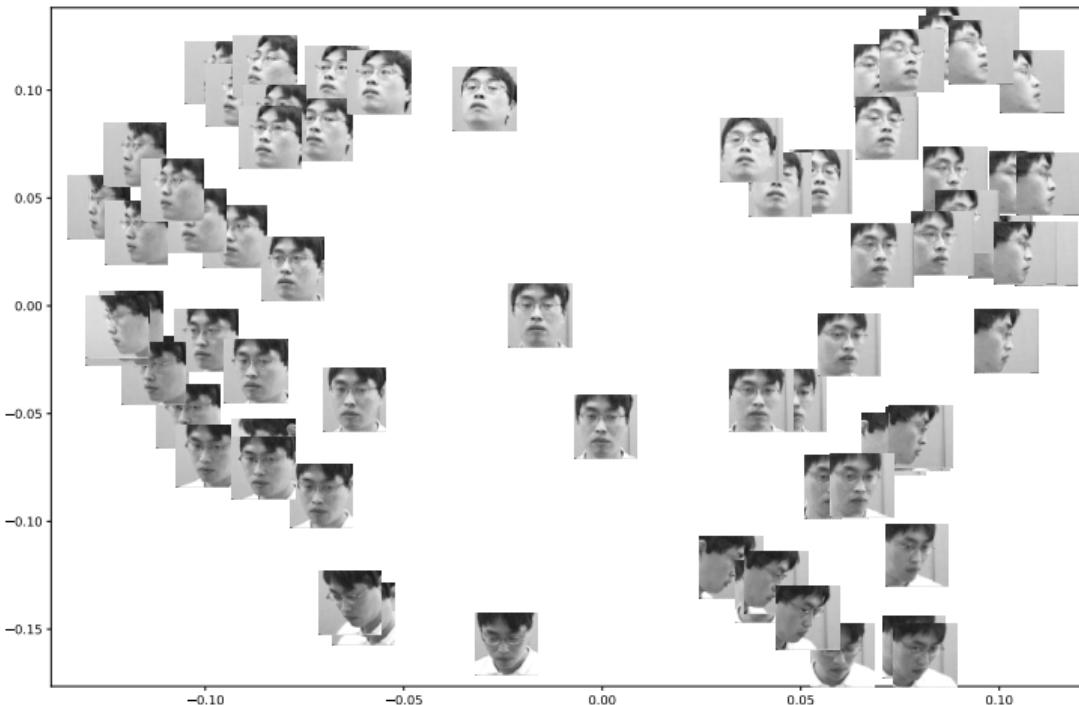
Example 2: Face Pose

- ▶ Construct fully-connected similarity graph with Gaussian similarity
- ▶ Embed with Laplacian eigenmaps

Example 2: Face Pose



Example 2: Face Pose



DSC 190

Machine Learning: Representations

Lecture 11 | Part 4

Spectral Clustering

Spectral Embeddings

- ▶ Useful in multiple tasks:
 - ▶ Feature learning before classification
 - ▶ Visualizing high dimensional data
 - ▶ Clustering

Spectral Clustering

- ▶ Problem: k-means assumptions:
 - ▶ Data are vectors (what about graphs?)
 - ▶ Clusters are spherical (what about more complex patterns?)
- ▶ One idea:
 1. Embed using, e.g., Laplacian eigenmaps
 2. Run k-means on the embedded points

Demo

DSC 190

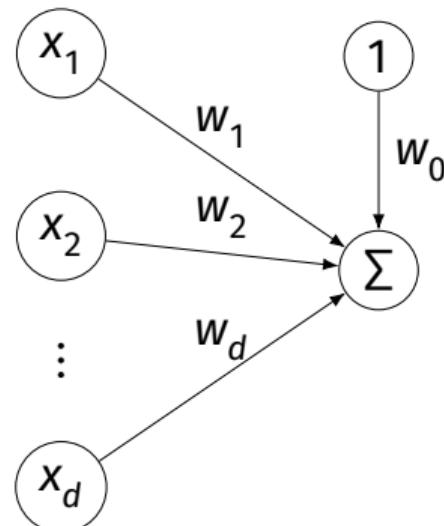
Machine Learning: Representations

Lecture 12 | Part 1

Neural Networks

Recall: Linear Predictor

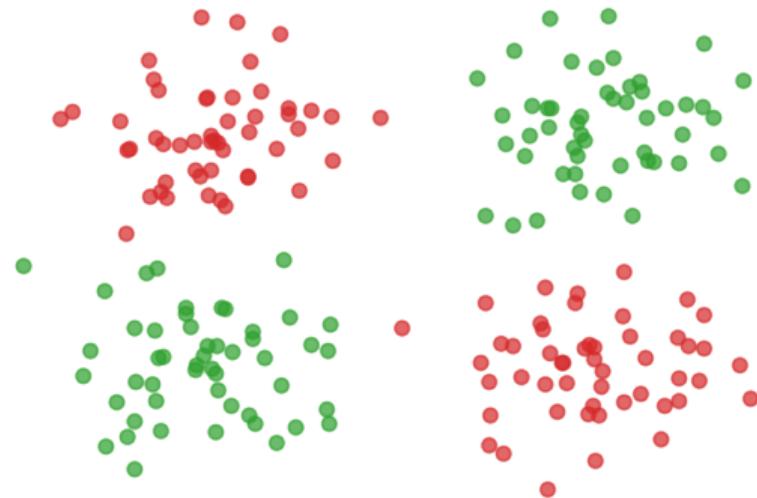
- ▶ **Input:** features $\vec{x} = (x_1, \dots, x_d)^T$
- ▶ **Parameters:**
 $\vec{w} = (w_0, w_1, \dots, w_d)^T$
- ▶ **Output:** $w_0 + w_1 x_1 + \dots + w_d x_d$



Linear Predictors

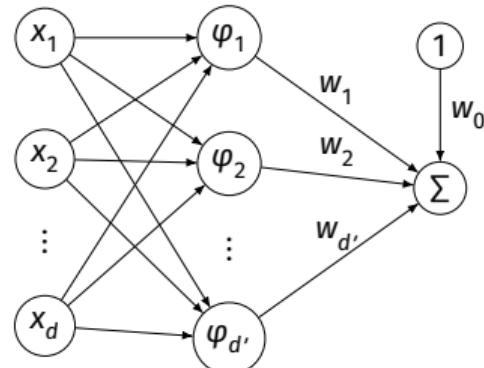
- ▶ **Pro:** simple, usually easy to optimize \vec{w}
 - ▶ With square loss, solution given by normal equations
- ▶ **Con:** Decision boundary is linear

Example



Recall: Basis Functions

- ▶ **Input:** features \vec{x} , basis functions $\varphi_1, \dots, \varphi_d : \mathbb{R}^d \rightarrow \mathbb{R}$
- ▶ **Parameters:**
 $\vec{w} = (w_0, w_1, \dots, w_d)^T$
- ▶ **Output:**
 $w_0 + w_1\varphi_1(\vec{x}) + \dots + w_d\varphi_d(\vec{x})$



Basis Functions

- ▶ **Note:** the basis functions and the weights \vec{w} are **not** chosen at the same time
- ▶ Two step process
- ▶ First, basis functions are chosen and fixed
 - ▶ By hand, by k -means clustering, etc.
- ▶ Then the weights \vec{w} are learned

Exercise

Why do this in two steps as opposed to one?

Answer

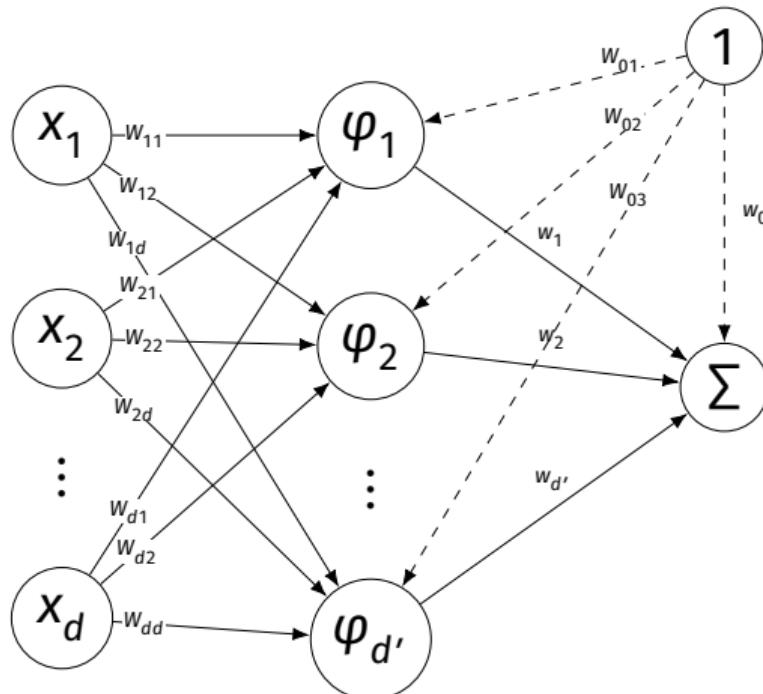
- ▶ By fixing basis functions *then* finding best \vec{w} , optimization is easy again
- ▶ Using square loss, normal equations still work

Idea

- ▶ Try to learn basis functions at same time as weights, \vec{w}
- ▶ Attempt #1: linear basis functions?

$$\varphi_i(\vec{x}) = W_{1i}x_1 + \dots + W_{di}x_d$$

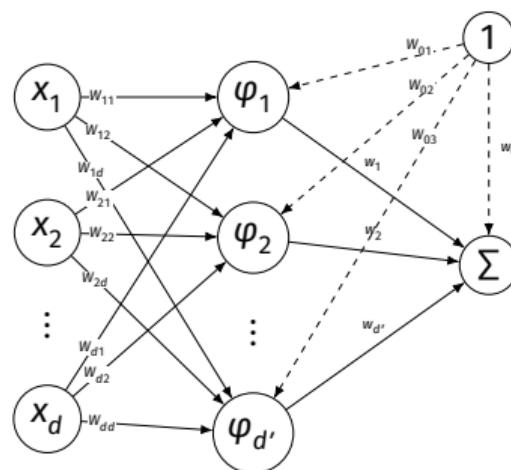
The Model



$$\varphi_i(\vec{x}) = W_{1i}x_1 + \dots + W_{di}x_d$$

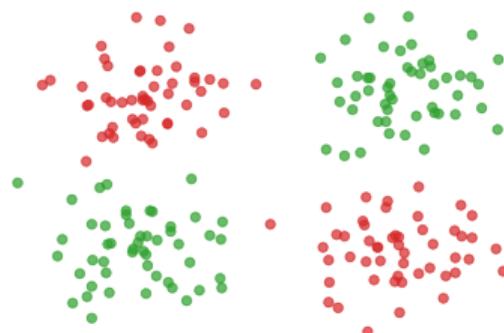
Neural Network

- ▶ **Input:** features \vec{x} ,
- ▶ **Parameters:**
 $\vec{w} = (w_0, w_1, \dots, w_d)^T$,
 $(d + 1) \times d'$ matrix W
- ▶ **Output:**
 $w_0 + w_1 \varphi_1(\vec{x}) + \dots + w_d \varphi_d(\vec{x})$
- ▶ This is a **neural network**



Problem

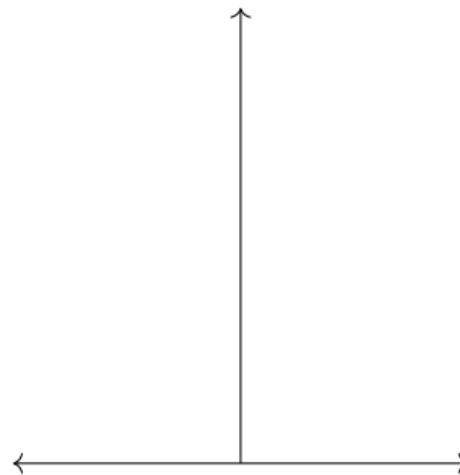
- ▶ If φ_i is linear, so is the decision boundary!



Activation Function

- ▶ To make φ_i nonlinear, we often apply a **activation function**.
- ▶ Very commonly: **rectified linear unit** (ReLU)

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



$$\begin{aligned}\varphi_i(\vec{x}) &= g(W_{0i} + W_{1i}x_1 + W_{2i}x_2 + \dots + W_{di}x_d A) \\ &= \max\{0, W_{0i} + W_{1i}x_1 + W_{2i}x_2 + \dots + W_{di}x_d A\}\end{aligned}$$

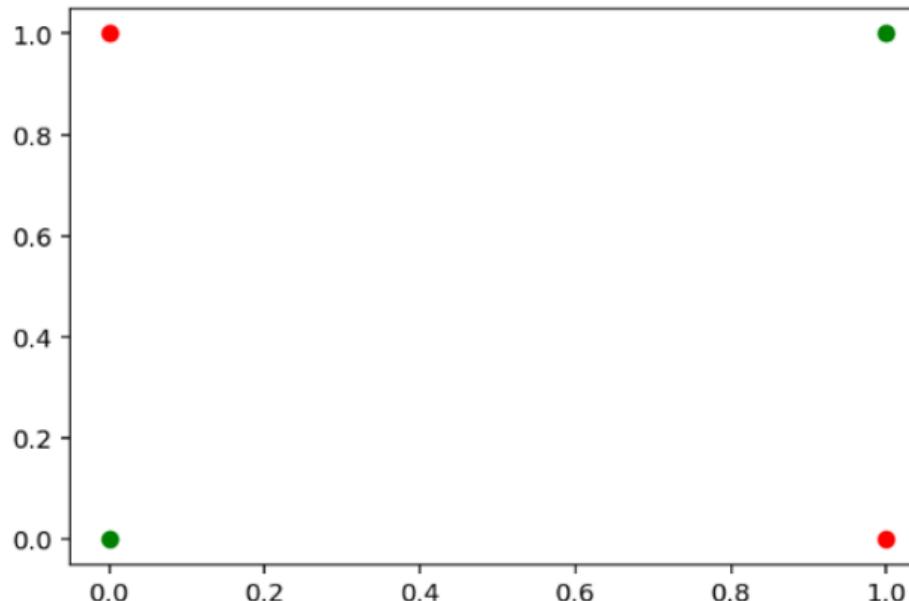
Neural Networks as Functions

- ▶ A neural network is simply a special kind of **function**.
- ▶ $f(\vec{x}; \vec{w}, W)$

Example

$$W = \begin{pmatrix} 2 & -1 \\ 3 & -2 \\ -2 & 1 \end{pmatrix} \quad \vec{w} = \begin{pmatrix} 4 \\ 0 \\ 2 \end{pmatrix} \quad \vec{x} = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

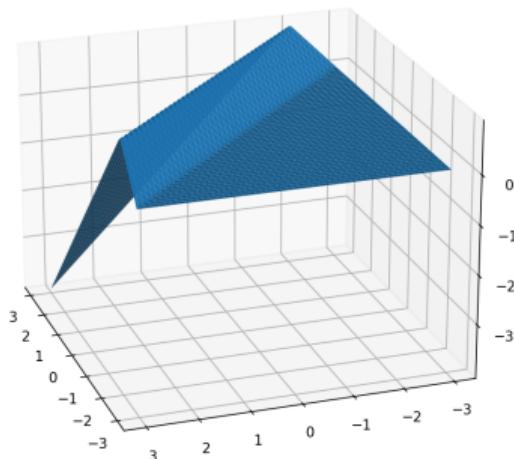
The Xor Problem



A Solution

$$W = \begin{pmatrix} 0 & -1 \\ 1 & 1 \\ 1 & 1 \end{pmatrix} \quad \vec{w} = \begin{pmatrix} 0 \\ 1 \\ -2 \end{pmatrix}$$

Prediction Surface



Learning with NNs

- ▶ We can **learn** weights by gathering data, picking a loss function and minimizing loss.
- ▶ The square loss works:

$$R(\vec{w}, W) = \frac{1}{n} \sum_{i=1}^n (f(\vec{x}^{(i)}; \vec{w}, W) - y_i)^2$$

Problem

- ▶ Now that the basis function weights are learnable, too, there is no simple solution for the best weights.
- ▶ We must instead use **gradient descent**.

DSC 190

Machine Learning: Representations

Lecture 12 | Part 2

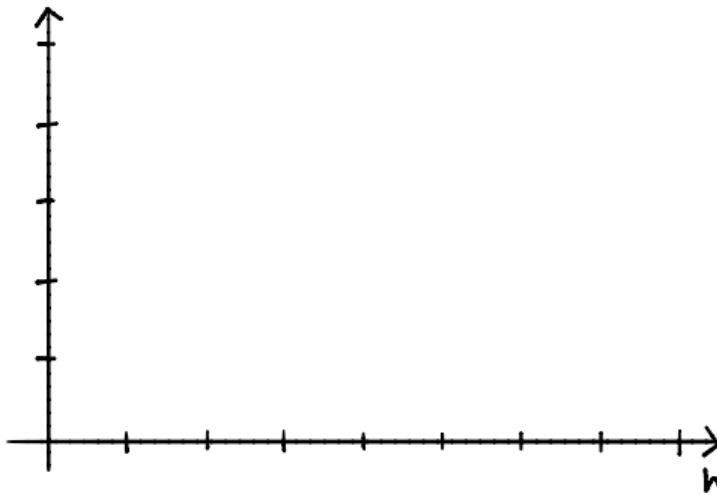
Gradient Descent

Gradient Descent

- ▶ We have a function $f : \mathbb{R} \rightarrow \mathbb{R}$
- ▶ We can't solve for the x that minimizes (or maximizes) $f(x)$
- ▶ Instead, we use the derivative to "walk" towards the optimizer

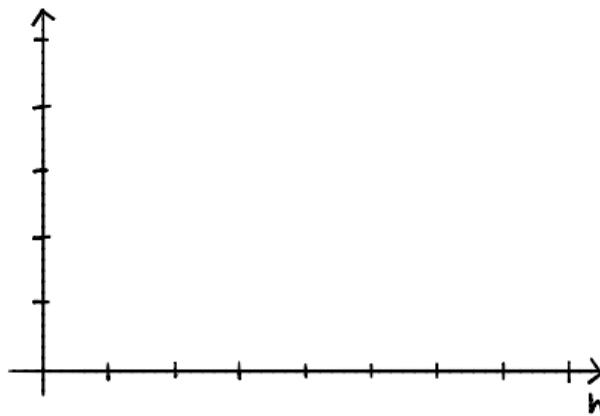
Meaning of the Derivative

- ▶ We have the derivative; can we use it?
- ▶ $\frac{df}{dx}(x)$ is a function; it gives the **slope** at x .



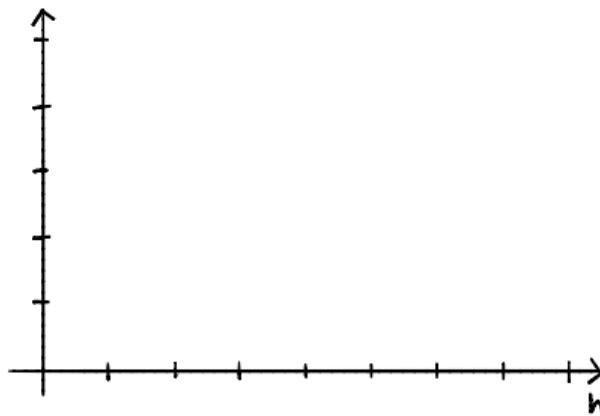
Key Idea Behind Gradient Descent

- ▶ If the slope of f at x is **positive** then moving to the **left** decreases the value of f .
- ▶ i.e., we should **decrease** x



Key Idea Behind Gradient Descent

- ▶ If the slope of f at x is **negative** then moving to the **right** decreases the value of f .
- ▶ i.e., we should **increase** x



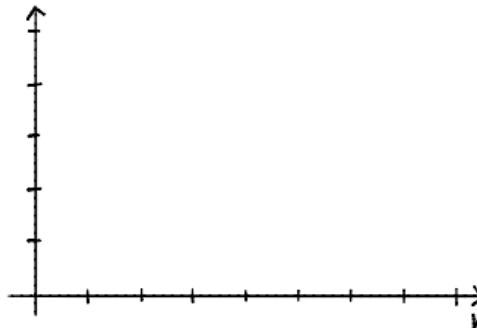
Key Idea Behind Gradient Descent

- ▶ Pick a starting place, x_0 . Where do we go next?
- ▶ Slope at x_0 negative? Then increase x_0 .
- ▶ Slope at x_0 positive? Then decrease x_0 .
- ▶ This will work:

$$x_1 = x_0 - \frac{df}{dx}(x_0)$$

Gradient Descent

- ▶ Pick α to be a positive number. It is the **learning rate**.
- ▶ Pick a starting prediction, x_0 .
- ▶ On step i , perform update $x_i = x_{i-1} - \alpha \cdot \frac{df}{dx}(x_{i-1})$
- ▶ Repeat until convergence (when x doesn't change much).



```
def gradient_descent(derivative, x, alpha, tol=1e-12):
    """Minimize using gradient descent."""
    while True:
        x_next = x - alpha * derivative(x)
        if abs(x_next - x) < tol:
            break
        x = x_next
    return h
```

Example: Minimizing Mean Squared Error

- Recall the mean squared error and its derivative:

$$R_{\text{sq}}(x) = \frac{1}{n} \sum_{i=1}^n (x - y_i)^2 \quad \frac{dR_{\text{sq}}}{dx}(x) = \frac{2}{n} \sum_{i=1}^n (x - y_i)$$

Exercise

Let $y_1 = -4, y_2 = -2, y_3 = 2, y_4 = 4.$

Pick $x_0 = 4$ and $\alpha = 1/4$. What is x_1 ?

- a) -1
- b) 0
- c) 1
- d) 2

Example

Gradient Descent in > 1 dimensions

- ▶ The derivative of f becomes the gradient:

$$\frac{df}{dx} \rightarrow \nabla f(\vec{x})$$

- ▶ Meaning of **differentiable**: locally, f looks linear.
- ▶ **Key:** $\nabla f(\vec{w})$ is a function; it returns a vector pointing in direction of steepest ascent.

Gradient Descent in > 1 dimensions

- ▶ Pick α to be a positive number.
 - ▶ It is the **learning rate**.
- ▶ Pick a starting guess, $\vec{w}^{(0)}$.
- ▶ On step i , update $\vec{w}^{(i)} = \vec{w}^{(i-1)} - \alpha \cdot \nabla f(\vec{w}^{(i-1)})$
- ▶ Repeat until convergence
 - ▶ when \vec{w} doesn't change much
 - ▶ equivalently, when $\|\nabla f(\vec{w}^{(i)})\|$ is small

```
def gradient_descent(gradient, w, alpha, tol=1e-12):
    """Minimize using gradient descent."""
    while True:
        w_next = w - alpha * gradient(x)
        if np.linalg.norm(w_next - w) < tol:
            break
        w = w_next
    return w
```

DSC 190

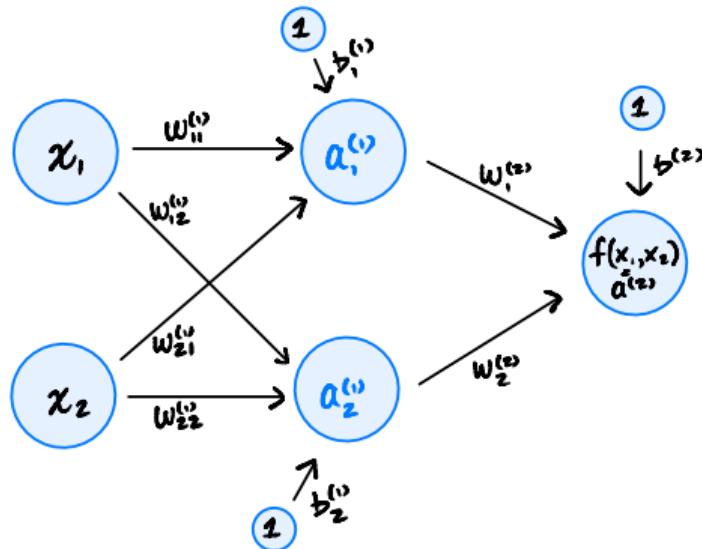
Machine Learning: Representations

Lecture 13 | Part 1

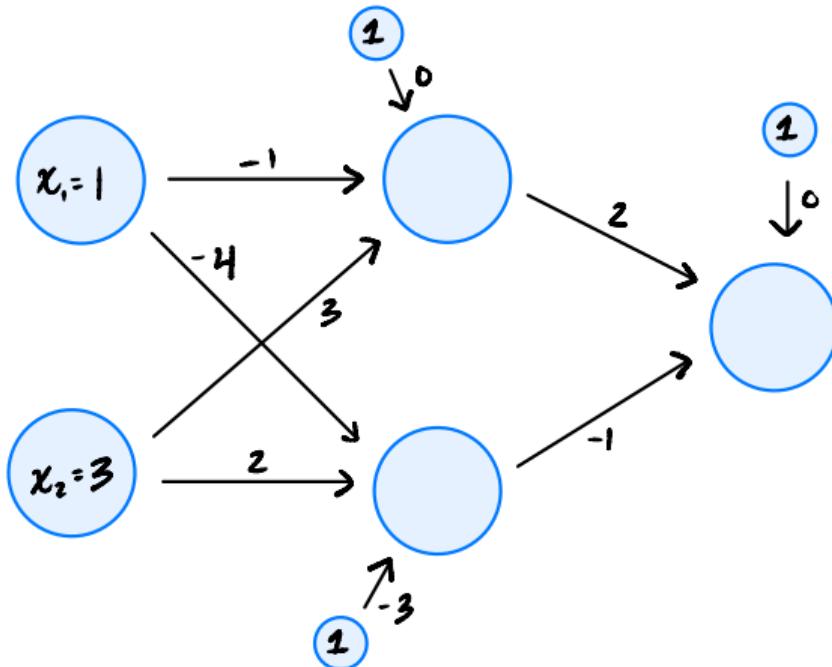
Convexity in 1-d

Neural Networks

A NN is just a function: $f(\vec{x}; \vec{w})$



Example



Learning

- ▶ **Given:** a data set $(\vec{x}^{(i)}, y_i)$
- ▶ **Find:** weights \vec{w} minimizing some cost function (e.g., expected square loss):

$$C(\vec{w}) = \frac{1}{n} \sum_{i=1}^n (f(\vec{x}^{(i)}; \vec{w}) - y_i)^2$$

- ▶ **Problem:** there is no closed-form solution

Gradient Descent

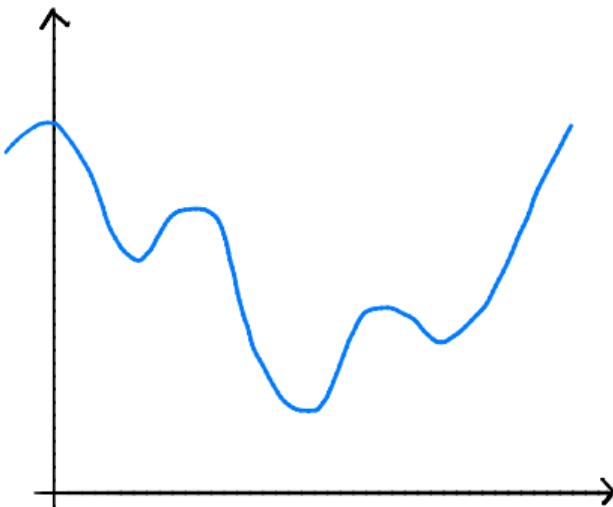
- ▶ **Idea:** start at arbitrary $\vec{w}^{(0)}$, walk in direction of gradient:

$$\nabla C = \begin{pmatrix} \frac{\partial C}{\partial w_0} \\ \frac{\partial C}{\partial w_1} \\ \vdots \\ \frac{\partial C}{\partial w_k} \end{pmatrix}$$

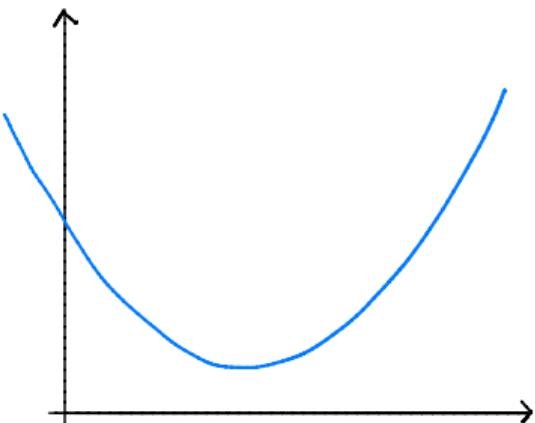
Question

When is gradient descent guaranteed to work?

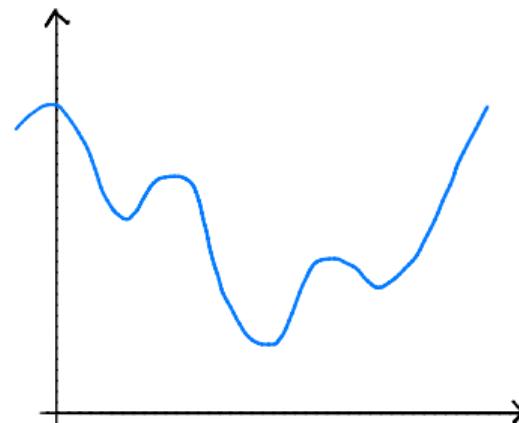
Not here...



Convex Functions



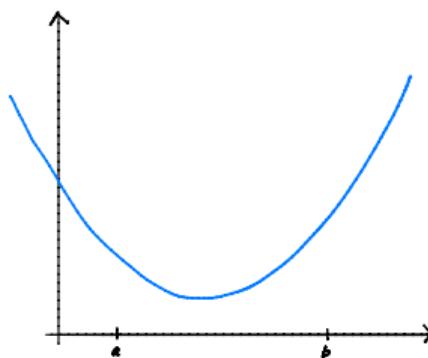
Convex



Non-convex

Convexity: Definition

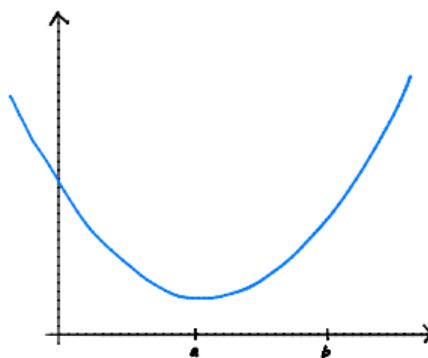
- ▶ f is **convex** if for **every** a, b the line segment between
$$(a, f(a)) \quad \text{and} \quad (b, f(b))$$
does not go below the plot of f .



Convexity: Definition

- ▶ f is **convex** if for **every** a, b the line segment between
$$(a, f(a)) \quad \text{and} \quad (b, f(b))$$

does not go below the plot of f .

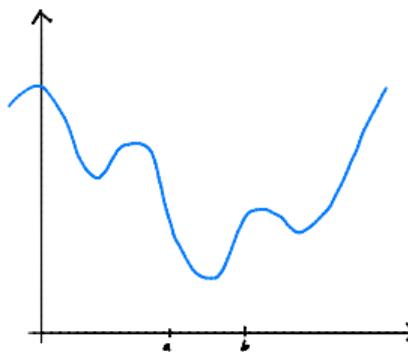


Convexity: Definition

- ▶ f is **convex** if for **every** a, b the line segment between

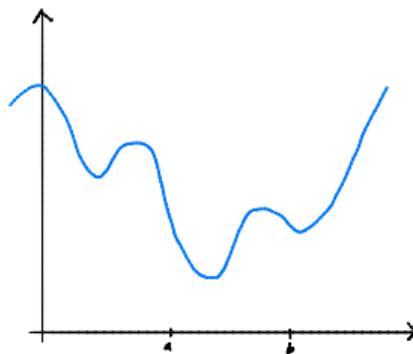
$$(a, f(a)) \quad \text{and} \quad (b, f(b))$$

does not go below the plot of f .



Convexity: Definition

- ▶ f is **convex** if for **every** a, b the line segment between
$$(a, f(a)) \quad \text{and} \quad (b, f(b))$$
does not go below the plot of f .



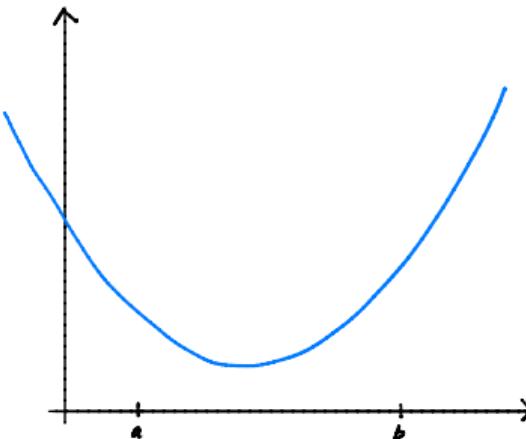
Other Terms

- ▶ If a function is not convex, it is **non-convex**.
- ▶ **Strictly convex**: the line lies strictly above curve.
- ▶ **Concave**: the line lies on or below curve.

Convexity: Formal Definition

- ▶ A function $f : \mathbb{R} \rightarrow \mathbb{R}$ is **convex** if for every choice of $a, b \in \mathbb{R}$ and $t \in [0, 1]$:

$$(1 - t)f(a) + tf(b) \geq f((1 - t)a + tb).$$

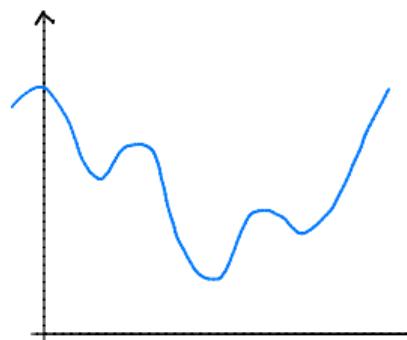
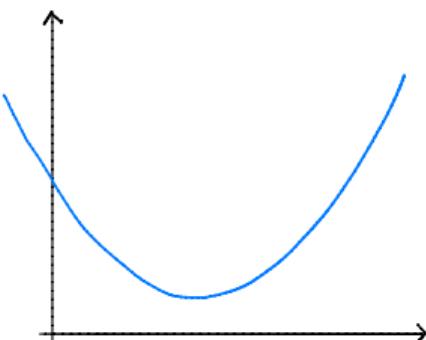


Example

Is $f(x) = |x|$ convex?

Another View: Second Derivatives

- ▶ If $\frac{d^2f}{dx^2}(x) \geq 0$ for all x , then f is convex.
- ▶ Example: $f(x) = x^4$ is convex.
- ▶ **Warning!** Only works if f is twice differentiable!

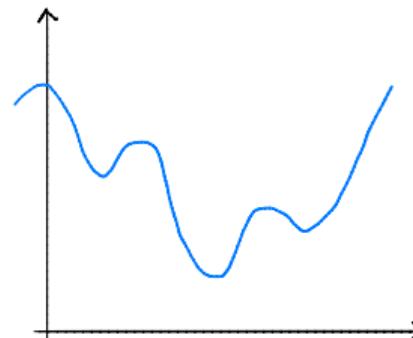
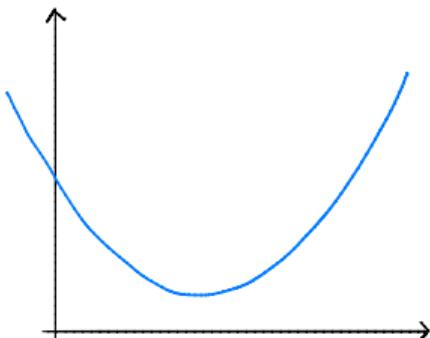


Another View: Second Derivatives

- ▶ “Best” straight line at x_0 :
 - ▶ $h_1(z) = f'(x_0) \cdot z + b$
- ▶ “Best” parabola at x_0 :
 - ▶ At x_0 , f looks like $h_2(z) = \frac{1}{2}f''(x_0) \cdot z^2 + f'(x_0)z + c$
 - ▶ Possibilities: upward-facing, downward-facing.

Convexity and Parabolas

- ▶ Convex if for **every** x_0 , parabola is upward-facing.
 - ▶ That is, $f''(x_0) \geq 0$.



Convexity and Gradient Descent

- ▶ Convex functions are (relatively) easy to optimize.
- ▶ **Theorem:** if $R(x)$ is convex and differentiable¹² then gradient descent converges to a **global optimum** of R provided that the step size is small enough³.

¹and its derivative is not too wild

²actually, a modified GD works on non-differentiable functions

³step size related to steepness.

Nonconvexity and Gradient Descent

- ▶ Nonconvex functions are (relatively) hard to optimize.
- ▶ Gradient descent can still be useful.
- ▶ But not guaranteed to converge to a global minimum.

DSC 190

Machine Learning: Representations

Lecture 13 | Part 2

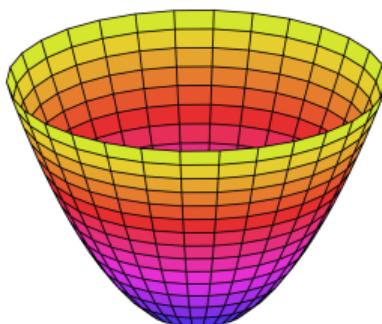
Convexity in Many Dimensions

Convexity: Definition

- ▶ $f(\vec{x})$ is **convex** if for **every** \vec{a}, \vec{b} the line segment between

$$(\vec{a}, f(\vec{a})) \quad \text{and} \quad (\vec{b}, f(\vec{b}))$$

does not go below the plot of f .



Convexity: Formal Definition

- ▶ A function $f : \mathbb{R}^d \rightarrow \mathbb{R}$ is **convex** if for every choice of $\vec{a}, \vec{b} \in \mathbb{R}^d$ and $t \in [0, 1]$:

$$(1 - t)f(\vec{a}) + tf(\vec{b}) \geq f((1 - t)\vec{a} + t\vec{b}).$$

The Second Derivative Test

- ▶ For 1-d functions, convex if second derivative ≥ 0 .
- ▶ For 2-d functions, convex if ???

The Hessian Matrix

- ▶ Create the **Hessian** matrix of second derivatives:

$$H(\vec{x}) = \begin{pmatrix} \frac{\partial f^2}{\partial x_1^2}(\vec{x}) & \frac{\partial f^2}{\partial x_1 x_2}(\vec{x}) \\ \frac{\partial f^2}{\partial x_2 x_1}(\vec{x}) & \frac{\partial f^2}{\partial x_2^2}(\vec{x}) \end{pmatrix}$$

In General

- If $f : \mathbb{R}^d \rightarrow \mathbb{R}$, the **Hessian** at \vec{x} is:

$$H(\vec{x}) = \begin{pmatrix} \frac{\partial f^2}{\partial x_1^2}(\vec{x}) & \frac{\partial f^2}{\partial x_1 x_2}(\vec{x}) & \dots & \frac{\partial f^2}{\partial x_1 x_d}(\vec{x}) \\ \frac{\partial f^2}{\partial x_2 x_1}(\vec{x}) & \frac{\partial f^2}{\partial x_2^2}(\vec{x}) & \dots & \frac{\partial f^2}{\partial x_2 x_d}(\vec{x}) \\ \dots & \dots & \dots & \dots \\ \frac{\partial f^2}{\partial x_d x_1}(\vec{x}) & \frac{\partial f^2}{\partial x_d x_2}(\vec{x}) & \dots & \frac{\partial f^2}{\partial x_d^2}(\vec{x}) \end{pmatrix}$$

The Second Derivative Test

- ▶ A function $f : \mathbb{R}^d \rightarrow \mathbb{R}$ is **convex** if for any $\vec{x} \in \mathbb{R}^d$, the Hessian matrix $H(\vec{x})$ is **positive semi-definite**.
- ▶ That is, all eigenvalues are ≥ 0

DSC 190

Machine Learning: Representations

Lecture 14 | Part 1

Basic Backpropagation

Computing the Gradient

- ▶ To train a neural network, we can use gradient descent.
- ▶ Involves computing the gradient of the cost function.
- ▶ **Backpropagation** is one method for efficiently computing the gradient.

The Gradient

$$\begin{aligned}\nabla_{\vec{w}} C(\vec{w}) &= \nabla_{\vec{w}} \frac{1}{n} \sum_{i=1}^n (f(\vec{x}^{(i)}; \vec{w}) - y_i)^2 \\ &= \frac{1}{n} \sum_{i=1}^n \nabla_{\vec{w}} (f(\vec{x}^{(i)}; \vec{w}) - y_i)^2 \\ &= \frac{1}{n} \sum_{i=1}^n 2(f(\vec{x}^{(i)}; \vec{w}) - y_i) \nabla_{\vec{w}} f(\vec{x}^{(i)}; \vec{w})\end{aligned}$$

Interpreting the Gradient

$$\nabla_{\vec{w}} C(\vec{w}) = \frac{1}{n} \sum_{i=1}^n 2 \left(f(\vec{x}^{(i)}; \vec{w}) - y_i \right) \nabla_{\vec{w}} f(\vec{x}^{(i)}; \vec{w})$$

- ▶ The gradient has one term for each training example, $(\vec{x}^{(i)}, y_i)$
- ▶ If prediction for $\vec{x}^{(i)}$ is good, contribution to gradient is small.
- ▶ $\nabla_{\vec{w}} f(\vec{x}^{(i)}; \vec{w})$ captures how sensitive $f(\vec{x}^{(i)})$ is to value of each parameter.

The Chain Rule

- ▶ Recall the **chain rule** from calculus.

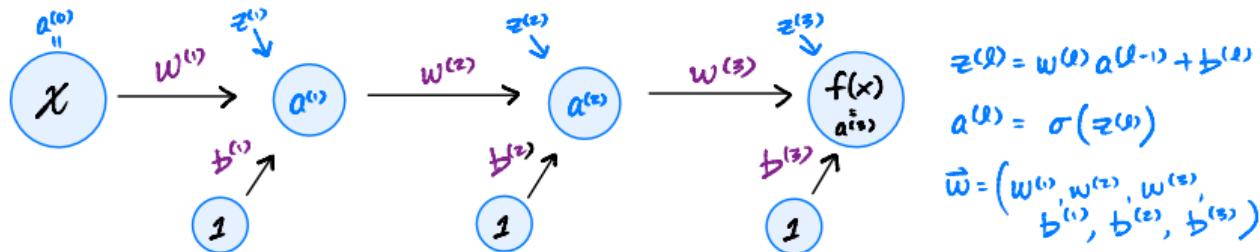
- ▶ Let $f, g : \mathbb{R} \rightarrow \mathbb{R}$

- ▶ Then:

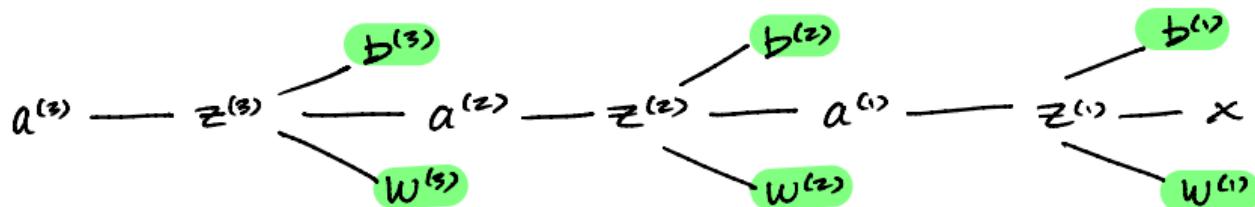
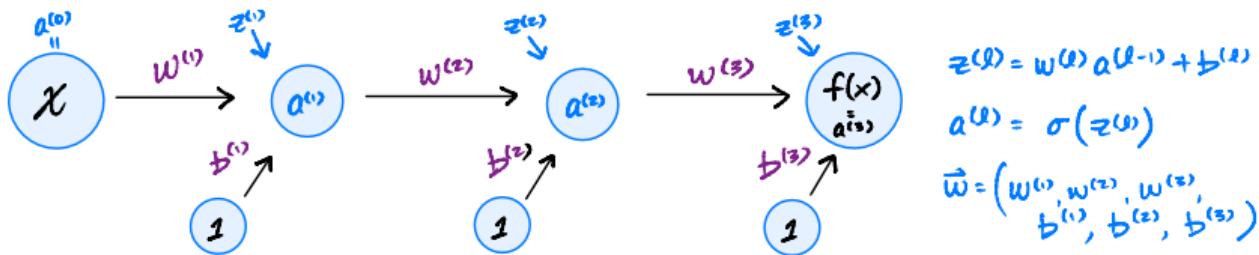
$$\frac{d}{dx} f(g(x)) = f'(g(x)) \cdot g(x)$$

- ▶ Alternative notation: $\frac{d}{dx} f(g(x)) = \frac{df}{dg} \frac{dg}{dx}(x)$

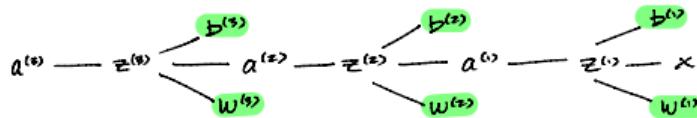
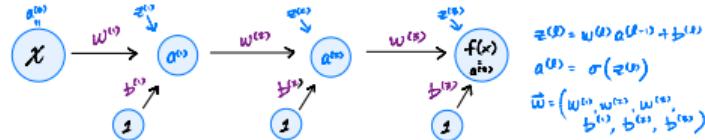
The Chain Rule for NNs



Computation Graphs



Example

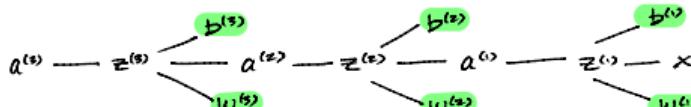


General Formulas

- ▶ Derivatives are defined recursively
- ▶ Easy to compute derivatives for early layers if we have derivatives for later layers.
- ▶ This is **backpropagation**.

$$\frac{\partial f}{\partial w^{(l)}} = \frac{\partial f}{\partial a^{(l)}} \cdot \frac{\partial a^{(l)}}{\partial z^{(l)}} \cdot \frac{\partial z^{(l)}}{\partial w^{(l)}}$$

$$\frac{\partial f}{\partial a^{(l)}} = \frac{\partial f}{\partial a^{(l+1)}} \cdot \frac{\partial a^{(l+1)}}{\partial z^{(l+1)}} \cdot \frac{\partial z^{(l+1)}}{\partial a^{(l)}}$$

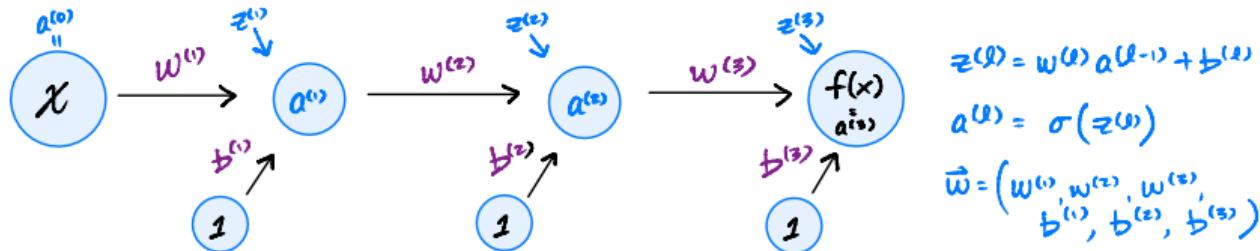


Warning

- ▶ The derivatives depend on the network **architecture**
 - ▶ Number of hidden nodes / layers
- ▶ Backprop is done automatically by your NN library

Backpropagation

Compute the derivatives for the last layers first; use them to compute derivatives for earlier layers.



DSC 190

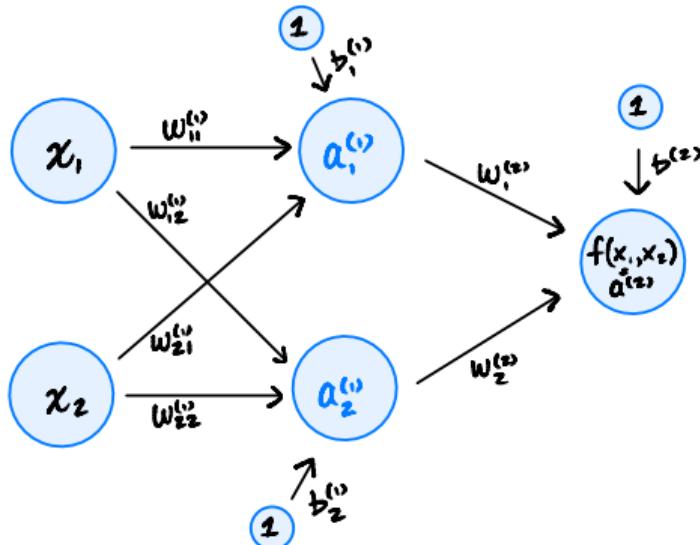
Machine Learning: Representations

Lecture 14 | Part 2

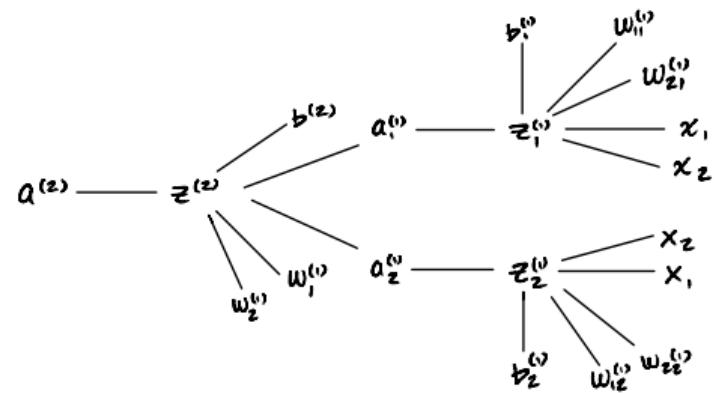
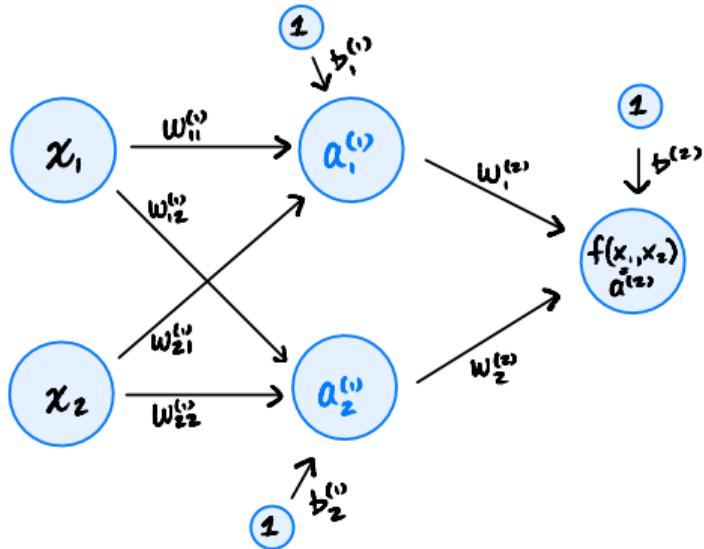
A More Complex Example

Complexity

- The strategy doesn't change much when each layer has more nodes.

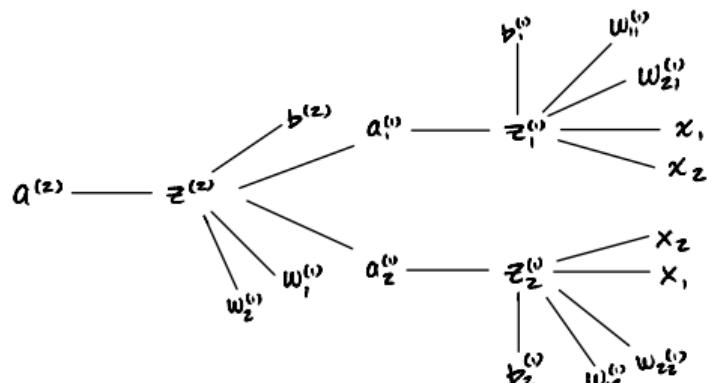


Computational Graph



Example

General Formulas



$$\frac{\partial f}{\partial w_{ij}^{(\ell)}} = \frac{\partial f}{\partial a^{(\ell)}} \cdot \frac{\partial a^{(\ell)}}{\partial z^{(\ell)}} \cdot \frac{\partial z^{(\ell)}}{\partial w_{ij}^{(\ell)}}$$

$$\frac{\partial f}{\partial a^{(\ell)}} = \frac{\partial f}{\partial a^{(\ell+1)}} \cdot \frac{\partial a^{(\ell+1)}}{\partial z^{(\ell+1)}} \cdot \frac{\partial z^{(\ell+1)}}{\partial a^{(\ell)}}$$

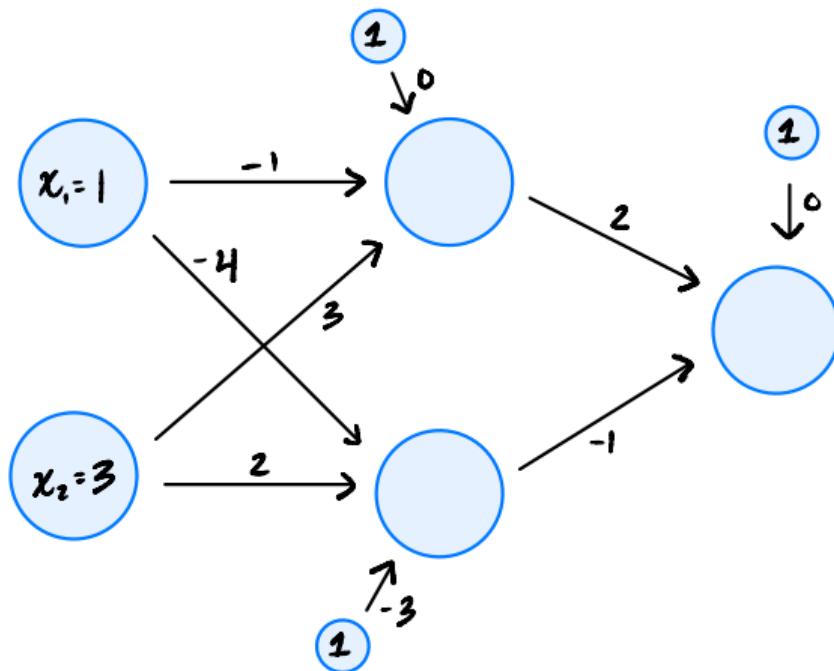
DSC 190

Machine Learning: Representations

Lecture 14 | Part 3

Intuition Behind Backprop

Intuition



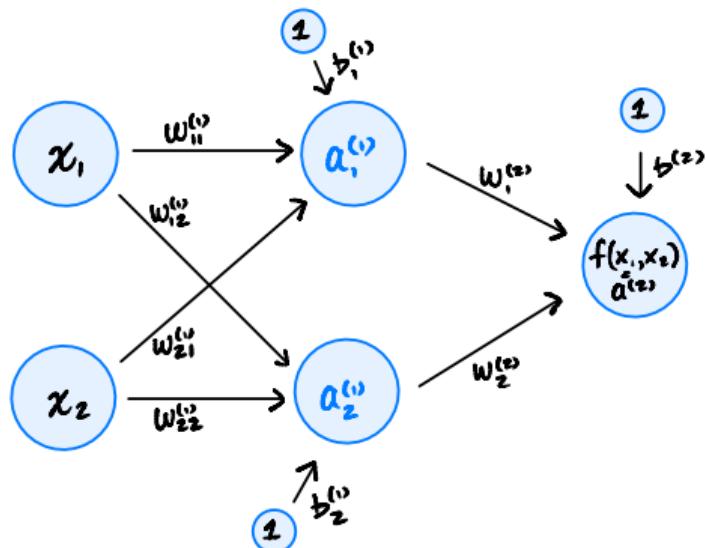
DSC 190

Machine Learning: Representations

Lecture 14 | Part 4

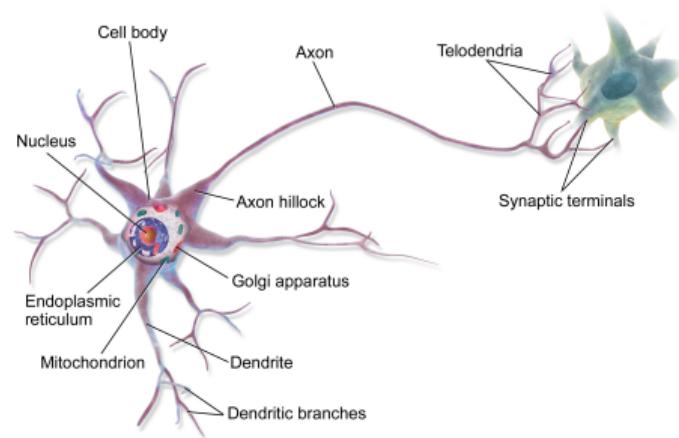
Hidden Units

Hidden Units



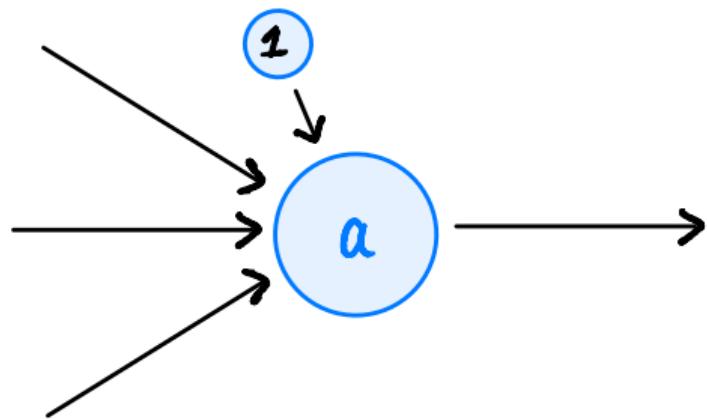
Neuron

- ▶ Neuron accepts signals along **synapses**.
- ▶ Synapses have weights.
- ▶ If weighted sum is “large enough”, the neuron fires, or **activates**.



Neuron

- ▶ Neuron accepts weighted inputs.
- ▶ If weighted sum is “large enough”, the neuron fires, or **activates**.

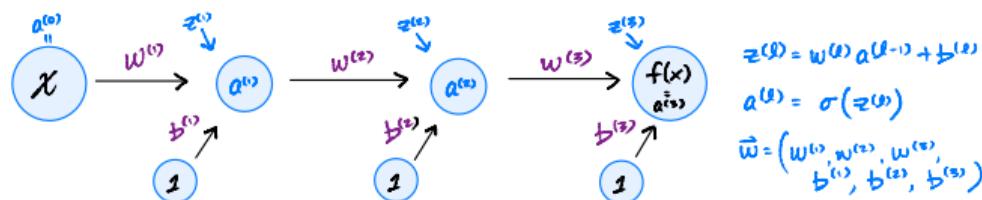


Activation Functions

- ▶ A function g determining whether – and how strong – a neuron fires.
- ▶ We have seen two: ReLU and linear.
- ▶ Many different choices.
- ▶ Guided by intuition and only a little theory.

Backpropagation

- ▶ The choice of activation function affects performance of backpropagation.
- ▶ Example:



$$\frac{\partial f}{\partial w^{(\ell)}} = \frac{\partial f}{\partial a^{(\ell)}} \cdot g'(z^{(\ell)}) \cdot \frac{\partial z^{(\ell)}}{\partial w^{(\ell)}}$$

Vanishing Gradients

- ▶ A major challenge in training deep neural networks with backpropagation is that of **vanishing gradients**.
- ▶ The gradient for layers far from the output becomes very small.
- ▶ Weights can't be changed.

$$\frac{\partial f}{\partial w^{(\ell)}} = \frac{\partial f}{\partial a^{(\ell)}} \cdot g'(z^{(\ell)}) \cdot \frac{\partial z^{(\ell)}}{\partial w^{(\ell)}}$$

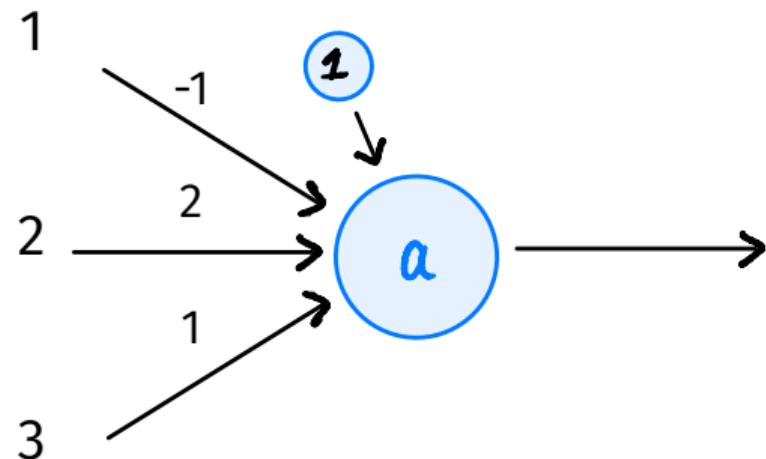
Main Idea

Some activation functions promote “healthier” gradients.

Linear Activations

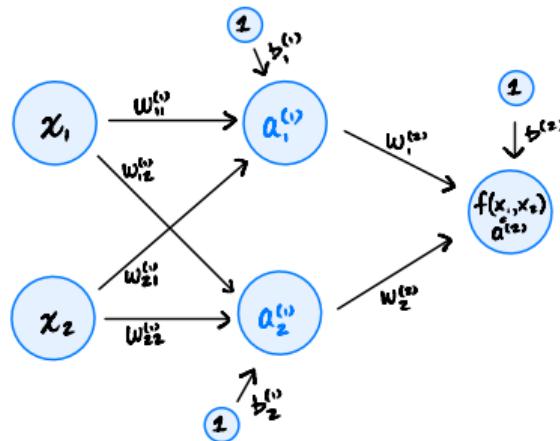
- ▶ A **linear** unit's activation function is:

$$g(z) = z$$

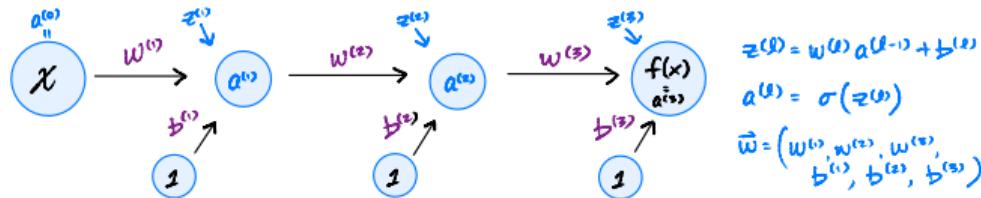


Problem

- ▶ Linear activations result in a linear prediction function.



Backprop. with Linear Activations



$$z^{(l)} = w^{(l)}a^{(l-1)} + b^{(l)}$$

$$a^{(l)} = \sigma(z^{(l)})$$

$$\vec{w} = (w^{(1)}, w^{(2)}, w^{(3)}, b^{(1)}, b^{(2)}, b^{(3)})$$

$$\frac{\partial f}{\partial w^{(\ell)}} = \frac{\partial f}{\partial a^{(\ell)}} \cdot g'(z^{(\ell)}) \cdot \frac{\partial z^{(\ell)}}{\partial w^{(\ell)}}$$

Summary: Linear Activations

- ▶ **Good:** healthy gradients, fast to compute
- ▶ **Bad:** still results in linear prediction function when layers are combined

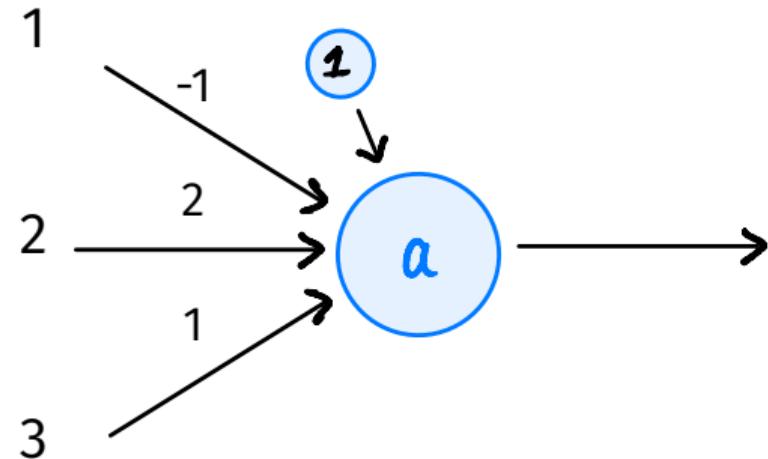
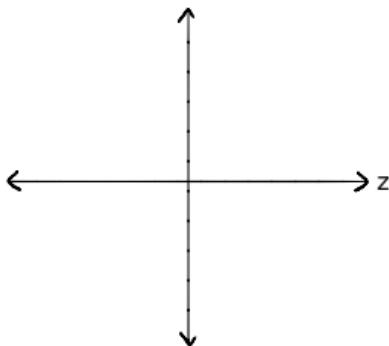
Sigmoidal Activations

- ▶ A basic nonlinearity.
- ▶ Neuron is either “on” (1), “off” (0), or somewhere in between.
- ▶ Very popular before introduction of the ReLU.

Sigmoidal Activations

- ▶ A **sigmoidal** unit's activation function is:

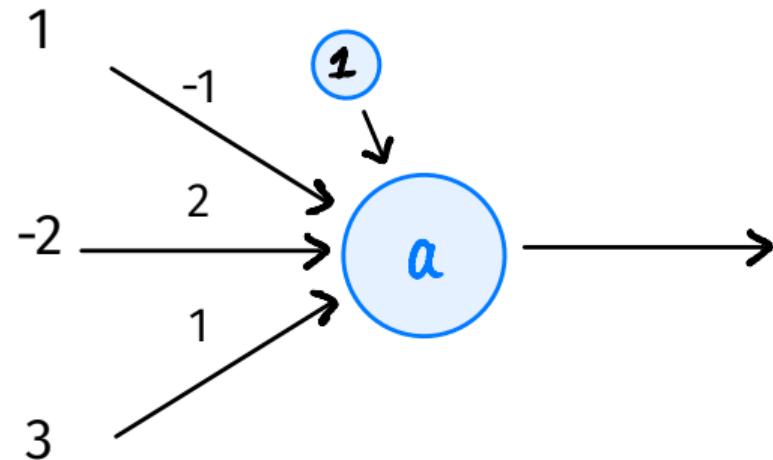
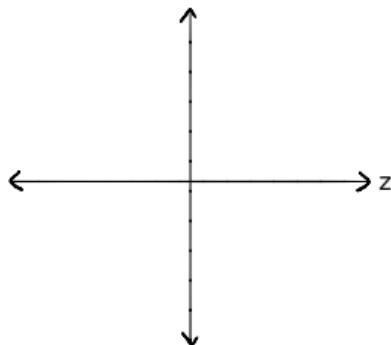
$$g(z) = \frac{1}{1 + e^{-z}}$$



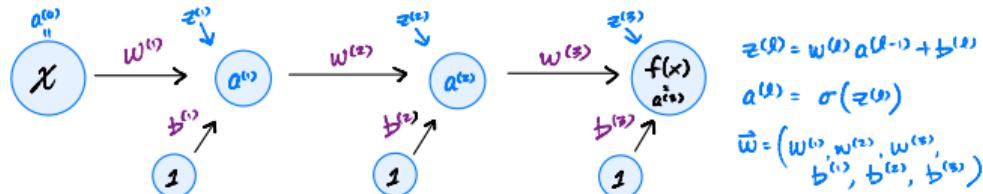
Sigmoidal Activations

- ▶ A **sigmoidal** unit's activation function is:

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



Backprop. with Sigmoids



$$g'(z) = g(z)(1 - g(z)) \quad \frac{\partial f}{\partial w^{(l)}} = \frac{\partial f}{\partial a^{(l)}} \cdot g'(z^{(l)}) \cdot \frac{\partial z^{(l)}}{\partial w^{(l)}}$$

Problem: Saturation

- ▶ Large/small inputs lead $g(z)$ to be very close to 1 or -1.
- ▶ Here, the derivative $\sigma'(z) \approx 0$.
- ▶ Vanishing gradients!
- ▶ Makes learning deep networks with gradient-based algorithms very difficult.

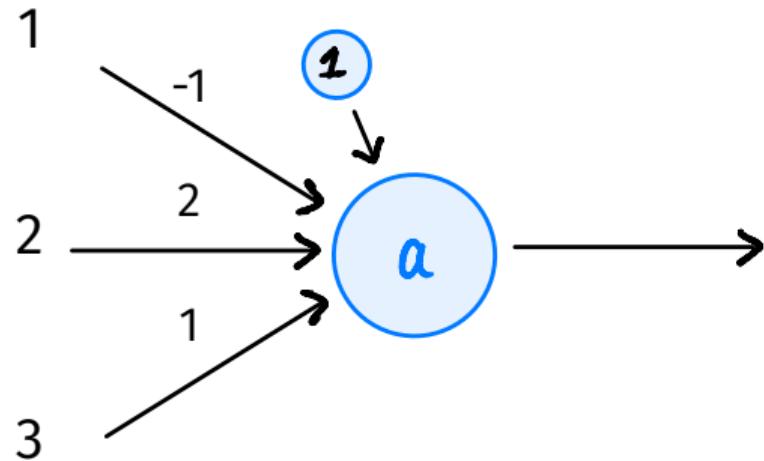
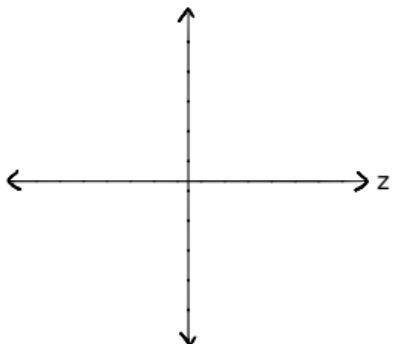
ReLU

- ▶ Linear activations have strong gradients, but combined are still linear.
- ▶ Sigmoidal activations are non-linear, but when saturated lead to weak gradients.
- ▶ Can we have the best of both?

ReLU

- A **rectified linear** unit's (ReLU) activation function is:

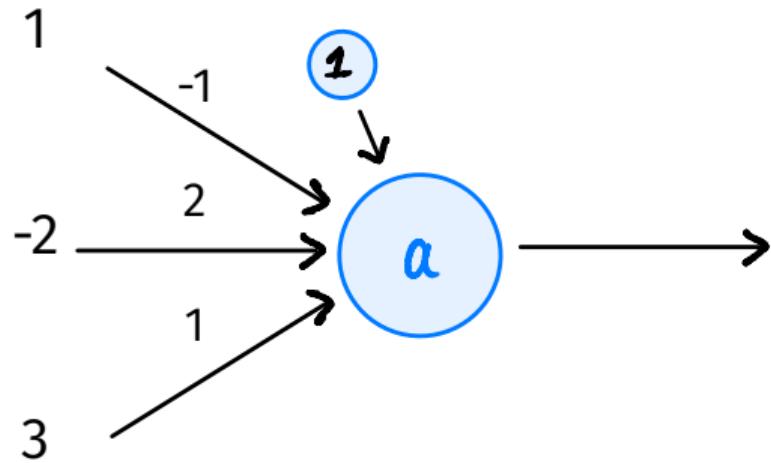
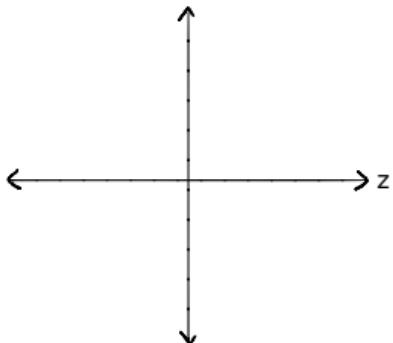
$$g(z) = \max\{0, z\}$$



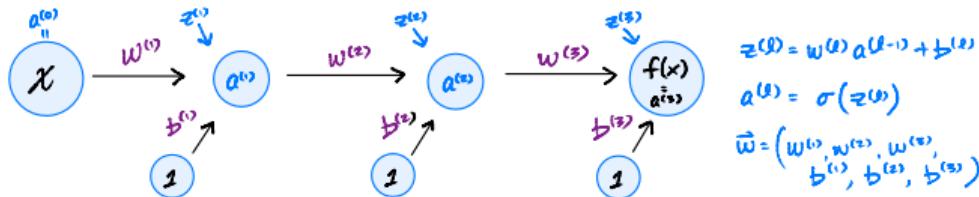
ReLU

- A **rectified linear** unit's (ReLU) activation function is:

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



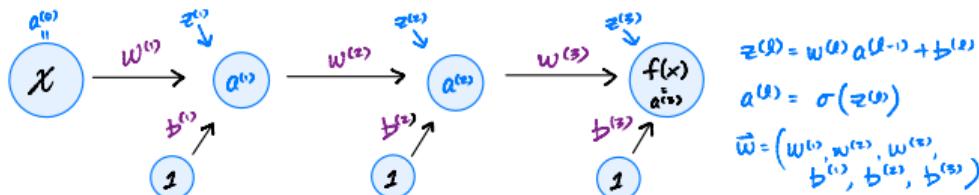
Backprop. with ReLU



$$\frac{\partial f}{\partial w^{(\ell)}} = \frac{\partial f}{\partial a^{(\ell)}} \cdot g'(z^{(\ell)}) \cdot \frac{\partial z^{(\ell)}}{\partial w^{(\ell)}}$$

Backprop. with ReLU

- ▶ **Problem:** If inputs < 0, ReLU “deactivates” and gradients are not passed back.



Fixing Deactivated ReLUs

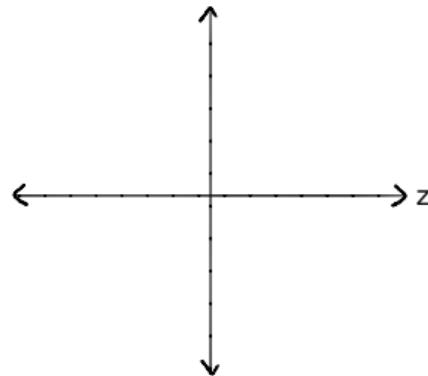
- ▶ One fix: initialize all biases to be small, positive numbers.
- ▶ Ensures that most units are active to begin with.
- ▶ Another fix: modify the ReLU.

Leaky ReLU

- ▶ A **leaky ReLU** activation function is:

$$g(z) = \max\{\alpha z, z\} \quad 0 \leq \alpha < 1$$

- ▶ Usually, $\alpha \approx 0.01$. Nonzero derivative.



Summary: ReLU

- ▶ The popular, “default” choice of activation function.
- ▶ **Good:** Strong gradient when active, fast to compute.
- ▶ **Bad:** No gradient when inactive.

DSC 190

Machine Learning: Representations

Lecture 14 | Part 5

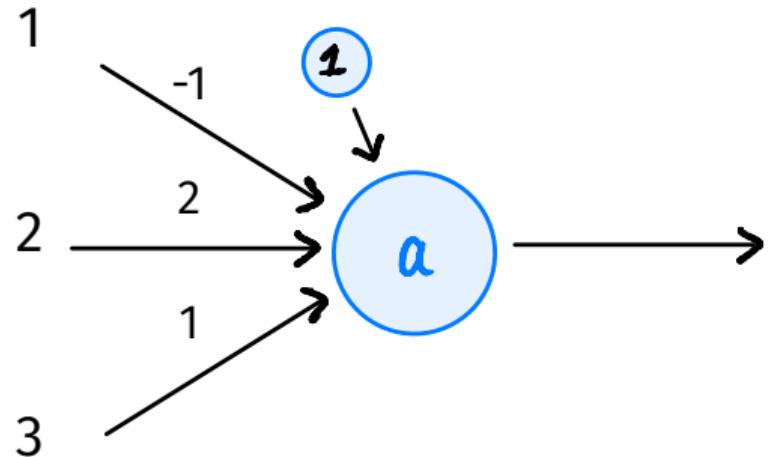
Output Units

Output Units

- ▶ As with units in hidden layers, we can customize output units.
 - ▶ What activation function?
 - ▶ How many units?
- ▶ Good choice depends on task:
 - ▶ Regression, binary classification, multiclass, etc.
- ▶ Which loss?

Setting 1: Regression

- ▶ Output can be any real number.
- ▶ Single output neuron.
- ▶ It makes sense to use a **linear activation**.

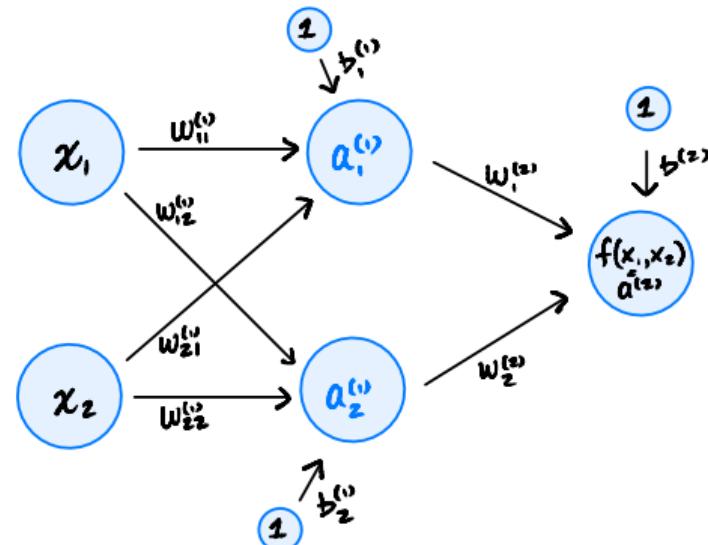


Setting 1: Regression

- ▶ Prediction should not be too high/low.
- ▶ It makes sense to use the **mean squared error**.

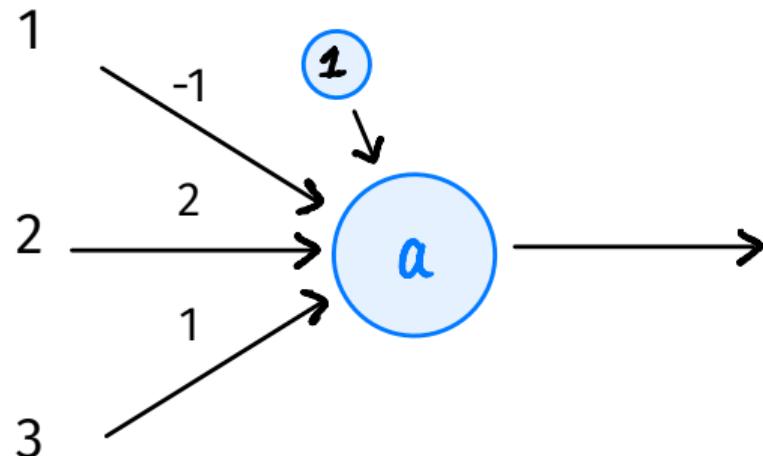
Setting 1: Regression

- ▶ Suppose we use linear activation for output neuron + mean squared error.
- ▶ This is very similar to least squares regression...
- ▶ But! Features in earlier layers are **learned**, non-linear.



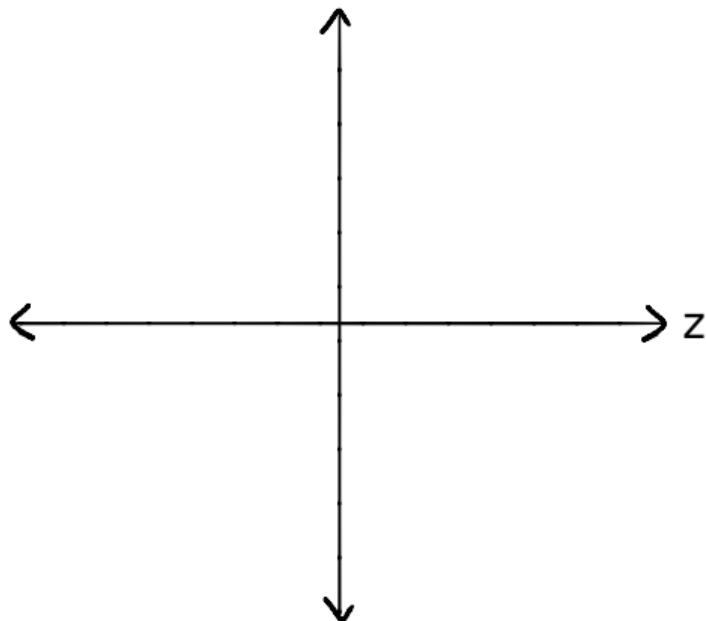
Setting 2: Binary Classification

- ▶ Output can be in $[0, 1]$.
- ▶ Single output neuron.
- ▶ We *could* use a **linear activation**, threshold.
- ▶ But there is a better way.



Sigmoids for Classification

- ▶ Natural choice for activation in output layer for binary classification: the **sigmoid**.



Binary Classification Loss

- ▶ We could use square loss for binary classification. There are several reasons not to:
- ▶ 1) Square loss penalizes predictions which are “too correct”.
- ▶ 2) It doesn’t work well with the sigmoid due to saturation.

The Cross-Entropy

- ▶ Instead, we often train deep classifiers using the **cross-entropy** as loss.
- ▶ Let $y^{(i)} \in \{0, 1\}$ be true label of i th example.
- ▶ The average cross-entropy loss:

$$-\frac{1}{n} \sum_{i=1}^n \begin{cases} \log f(\vec{x}^{(i)}), & \text{if } y^{(i)} = 1 \\ \log [1 - f(\vec{x}^{(i)})], & \text{if } y^{(i)} = 0 \end{cases}$$

The Cross-Entropy and the Sigmoid

- ▶ Cross-entropy “undoes” the exponential in the sigmoid, resulting in less saturation.

Summary: Binary Classification

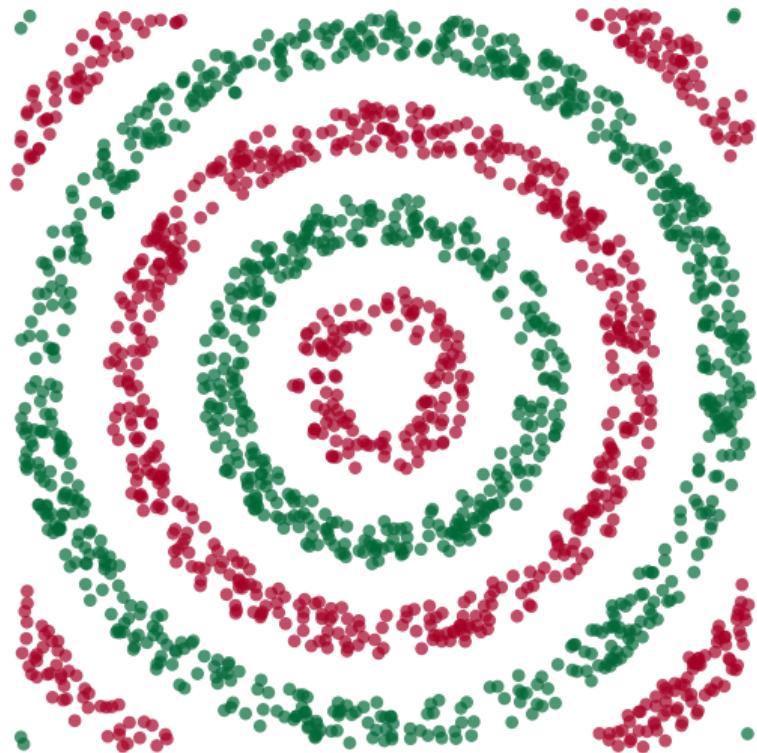
- ▶ Use sigmoidal activation the output layer + cross-entropy loss.
- ▶ This will promote a strong gradient.
- ▶ Use whatever activation for the hidden layers (e.g., ReLU).

DSC 190

Machine Learning: Representations

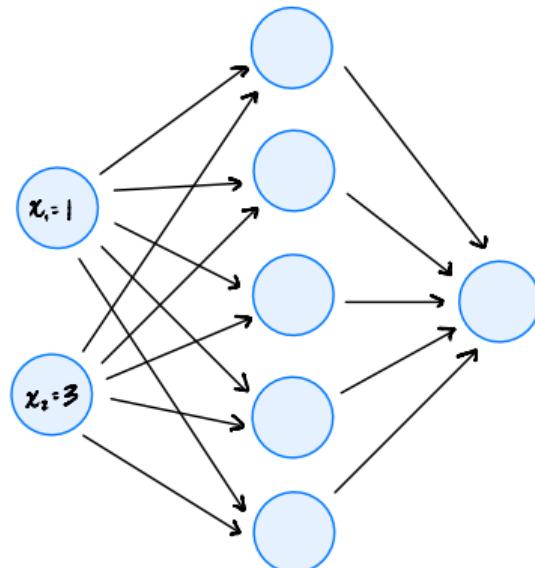
Lecture 15 | Part 1

NNs and Representations



NNs and Representations

- ▶ Hidden layer transforms to new **representation**.
 - ▶ Maps $\mathbb{R}^2 \rightarrow \mathbb{R}^5$
- ▶ Output layer makes prediction.
 - ▶ Maps $\mathbb{R}^5 \rightarrow \mathbb{R}^1$
- ▶ Representation optimized for classification!



NN Design

- ▶ Design a network for classification.
- ▶ Hidden layer activations: ReLU
- ▶ Output layer activation: sigmoid
- ▶ Loss function: cross-entropy

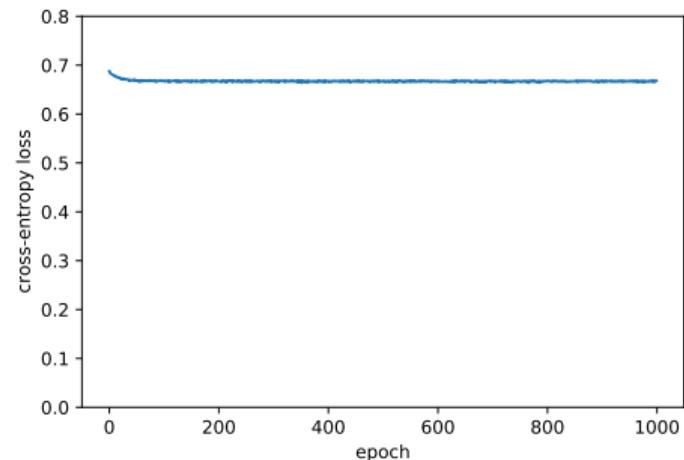
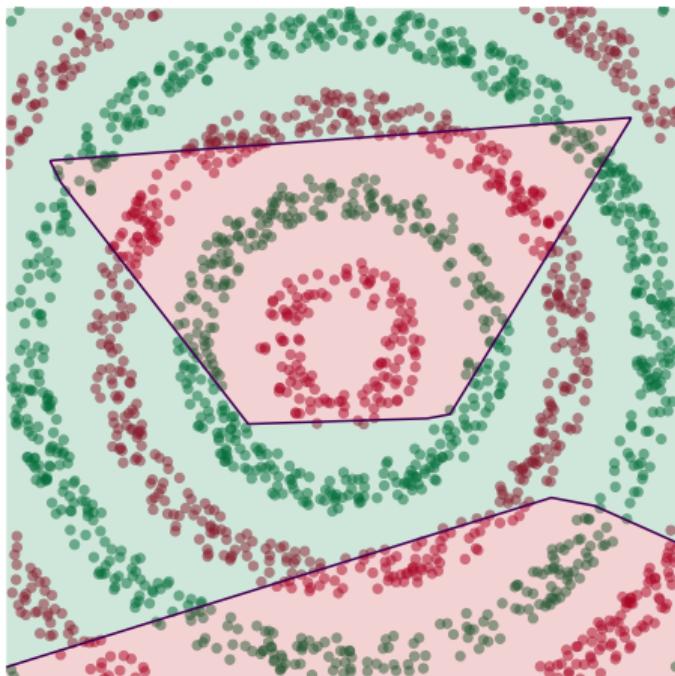
```
from tensorflow import keras

inputs = keras.Input(shape=2)
hidden_1 = keras.layers.Dense(5, activation='relu')(inputs)
outputs = keras.layers.Dense(1, activation='sigmoid')(hidden_1)

model = keras.Model(inputs=inputs, outputs=outputs)

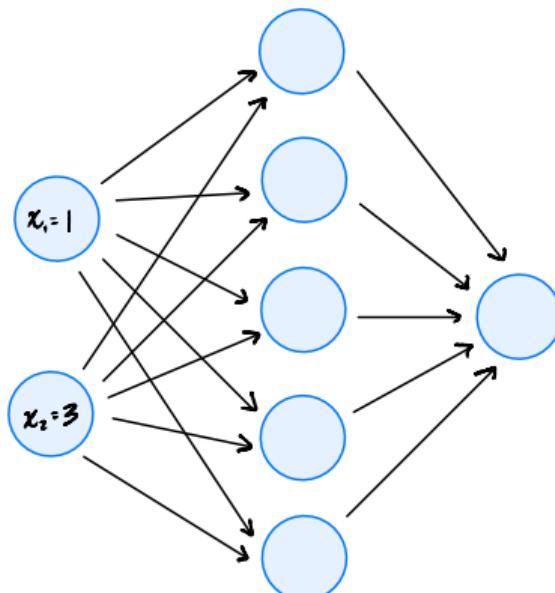
model.compile(
    optimizer=keras.optimizers.RMSprop(learning_rate=.01),
    loss=keras.losses.BinaryCrossentropy()
)
history = model.fit(X, y, epochs=1000, verbose=1)
```

Results



NNs and Representations

- ▶ Data has complex structure.
- ▶ Only 5 hidden neurons not enough to learn a good representation.



DSC 190

Machine Learning: Representations

Lecture 15 | Part 2

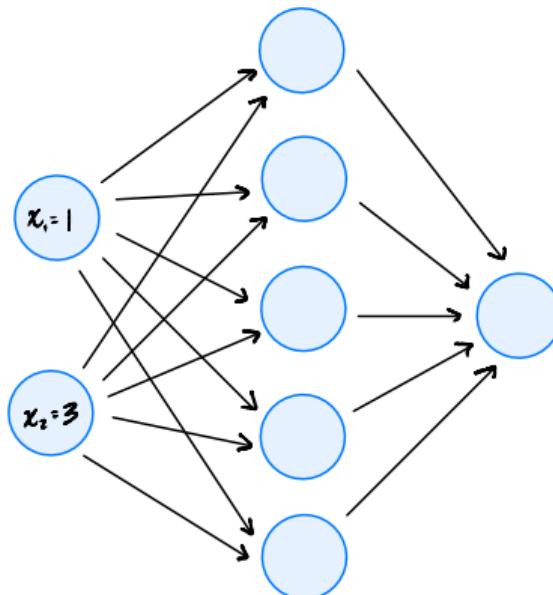
Architecture

Architecture

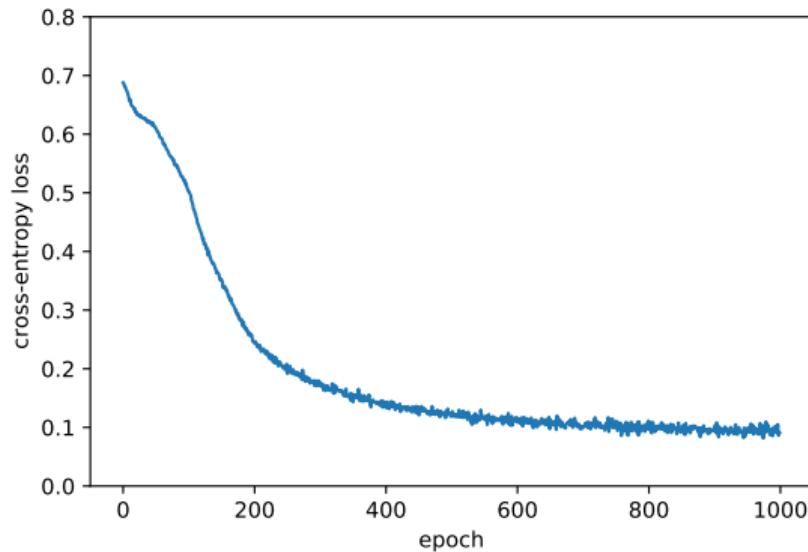
- ▶ We can increase complexity in two ways:
- ▶ Increasing **width**.
- ▶ Increasing **depth**.

Increasing Width

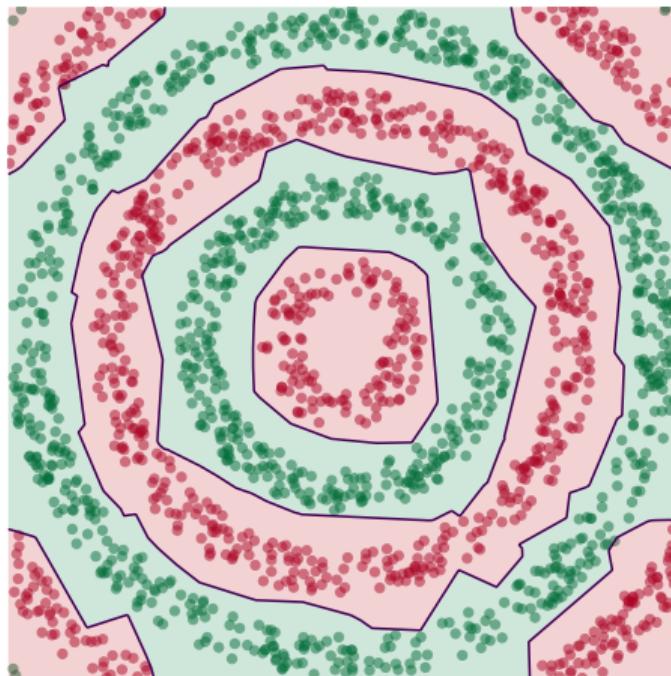
- ▶ Use a single hidden layer.
- ▶ But with 50 hidden neurons instead of 5.
- ▶ I.e., map to \mathbb{R}^{50} , then predict.



LOSS



Result



Universal Approximation Theorem

- ▶ A neural network f is a special type of function.
- ▶ Given another function g , can we make a neural network f so that $f(\vec{x}) \approx g(\vec{x})$?
- ▶ **Yes!** Assuming:
 - ▶ f has a hidden layer with a suitable activation function (ReLU, sigmoid, etc.)
 - ▶ the hidden layer has **enough** neurons
 - ▶ g is not too wild.

Main Idea

A network with a single hidden layer is able to approximate any (not-too-wild) function arbitrarily well as long as it has enough neurons in the hidden layer.

So what?

- ▶ Nature uses some function g to assign class labels to data.
- ▶ We don't see this function. But we see $g(\vec{x})$ for a bunch of points.
- ▶ Our goal is to learn a function f approximating g using this data.

The Challenge

- ▶ NNs are universal approximators (so are RBF networks, etc.)
- ▶ But just because it *can* approximate any function, doesn't mean we can *learn* the approximation.

Number of Neurons

- ▶ UAT says one hidden layer works well with “enough neurons”
- ▶ What is enough?
- ▶ Unfortunately, it can be a lot!

DSC 190

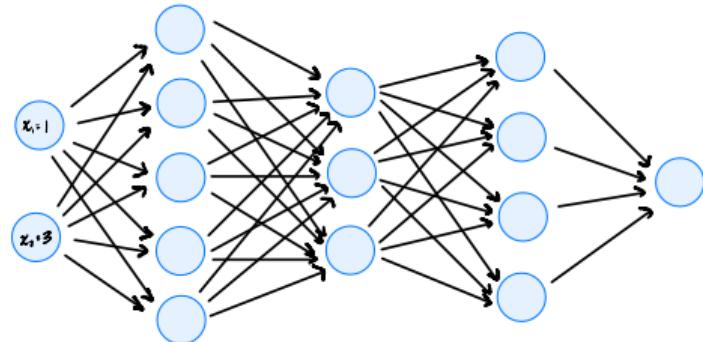
Machine Learning: Representations

Lecture 15 | Part 3

Deep Networks

Deep Networks

- ▶ Use a **multiple** hidden layers.
- ▶ Hidden layers transform to a new representation.
- ▶ Composition of simple transformations.
- ▶ Output layer performs prediction.



Main Idea

In machine learning, “deep” means “more than one hidden layer”. Deep models are useful for **learning** simpler representations.

Designing a Deep NN

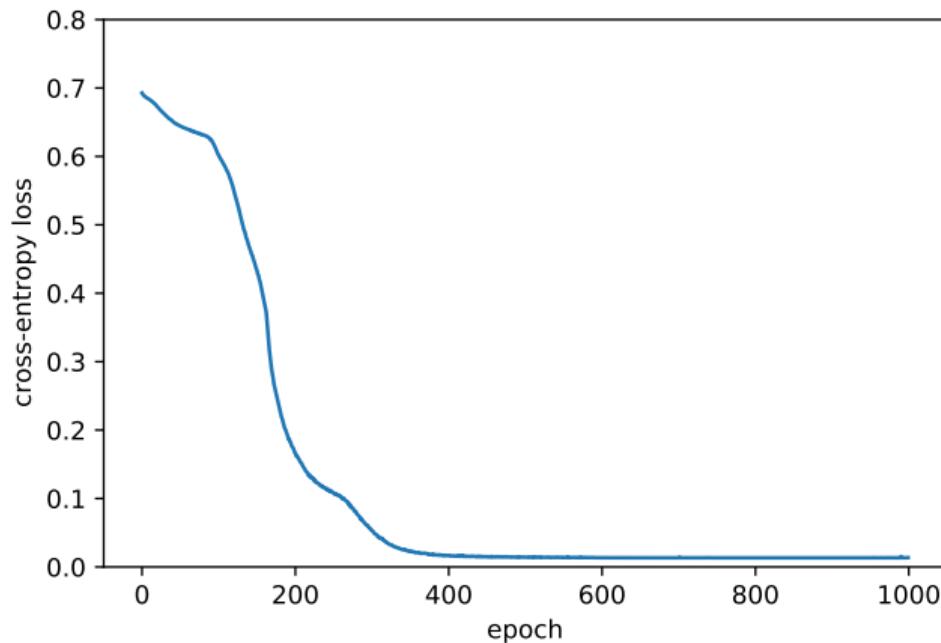
- ▶ Pick a number of hidden layers.
- ▶ Pick width of each hidden layer.
- ▶ There's not much theory to help us here.
- ▶ Experiment with different choices.

```
inputs = keras.Input(shape=2)
hidden_1 = keras.layers.Dense(15, activation='relu')(inputs)
hidden_2 = keras.layers.Dense(20, activation='relu')(hidden_1)
hidden_3 = keras.layers.Dense(2, activation='relu')(hidden_2)
outputs = keras.layers.Dense(1, activation='sigmoid')(hidden_3)

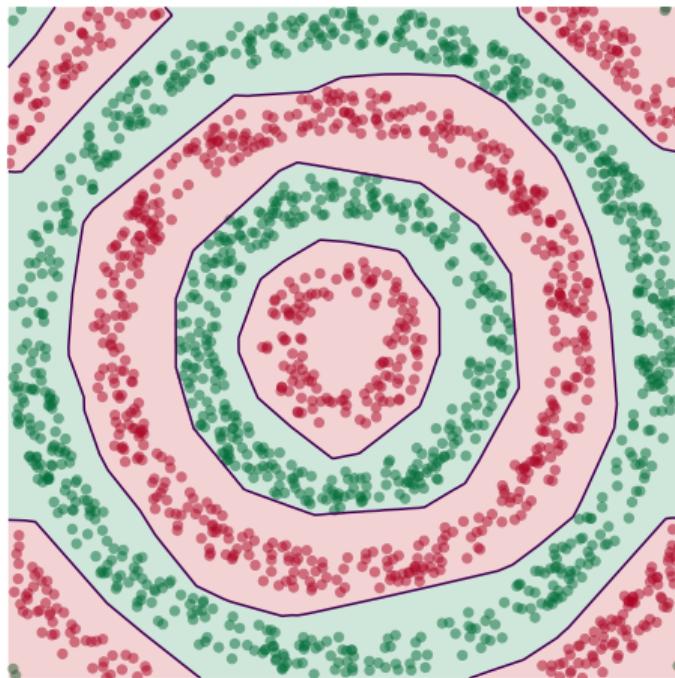
model = keras.Model(inputs=inputs, outputs=outputs)

model.compile(
    optimizer=keras.optimizers.RMSprop(learning_rate=.001),
    loss=keras.losses.BinaryCrossentropy()
)
history = model.fit(X, y, epochs=1000, verbose=1)
```

LOSS

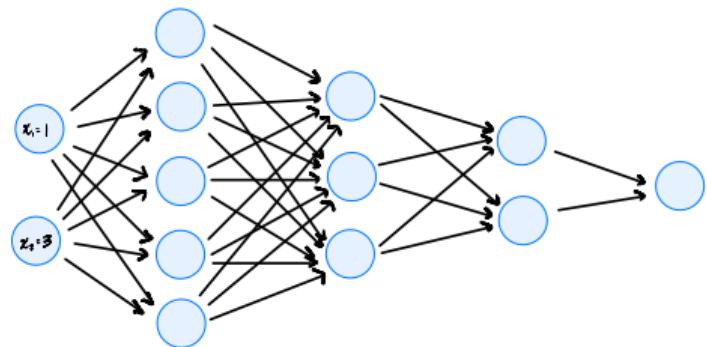


Result

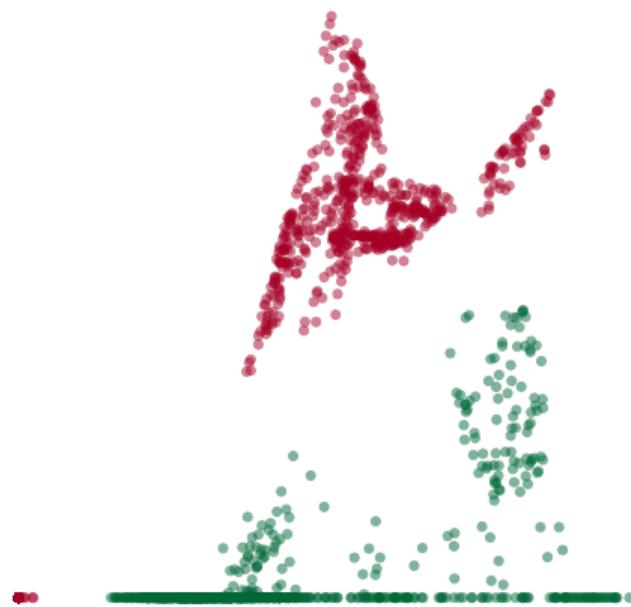


Deep Networks

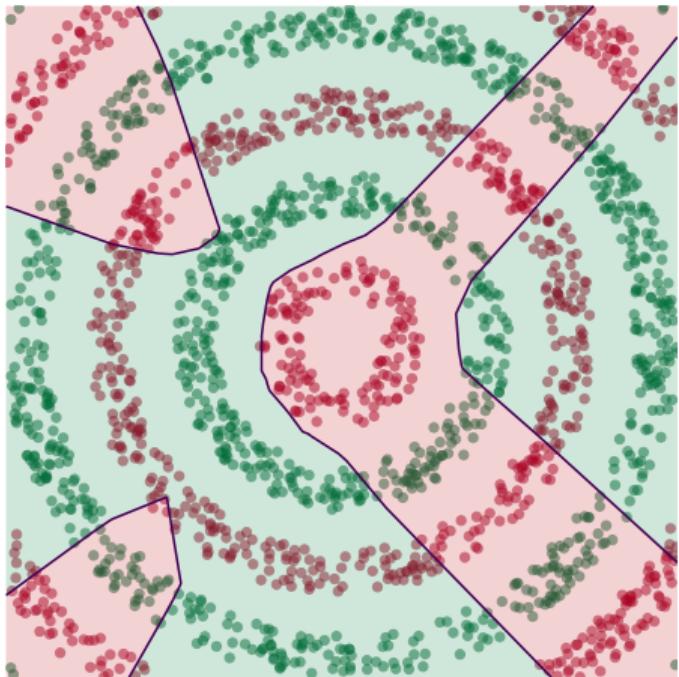
- ▶ Hidden layers map input to new representation.
- ▶ We can see this new representation!
- ▶ Plug in \vec{x} and see activations of last hidden layer.



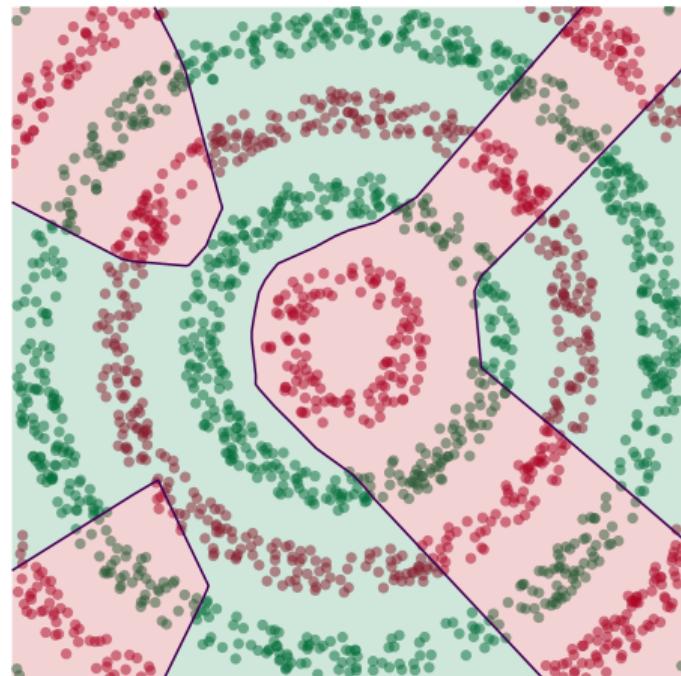
The New Representation



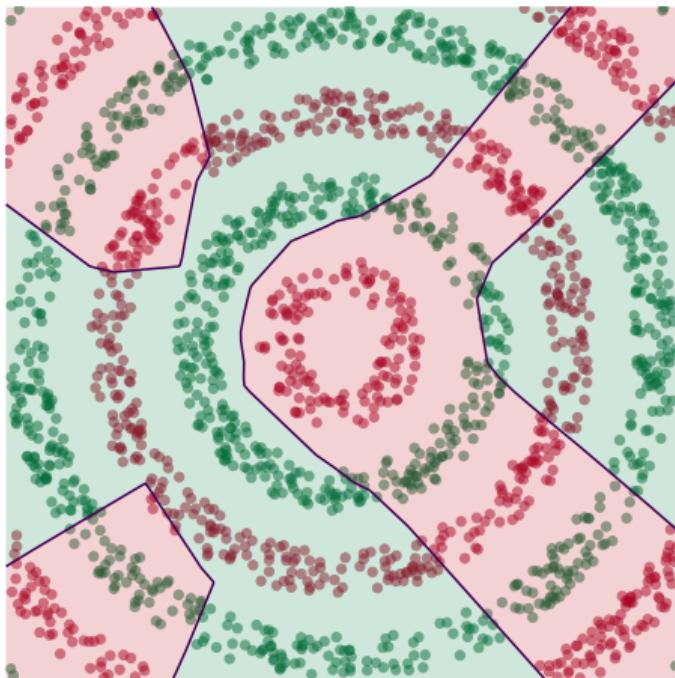
Learning a New Representation



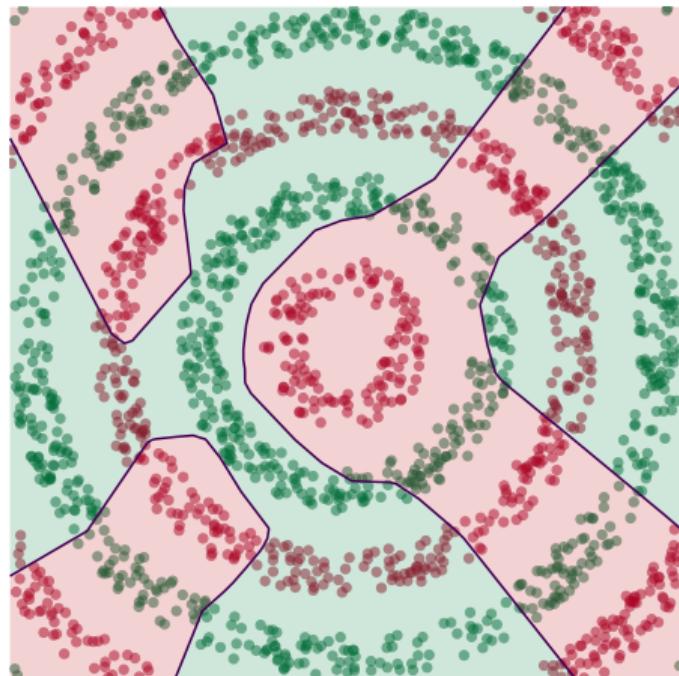
Learning a New Representation



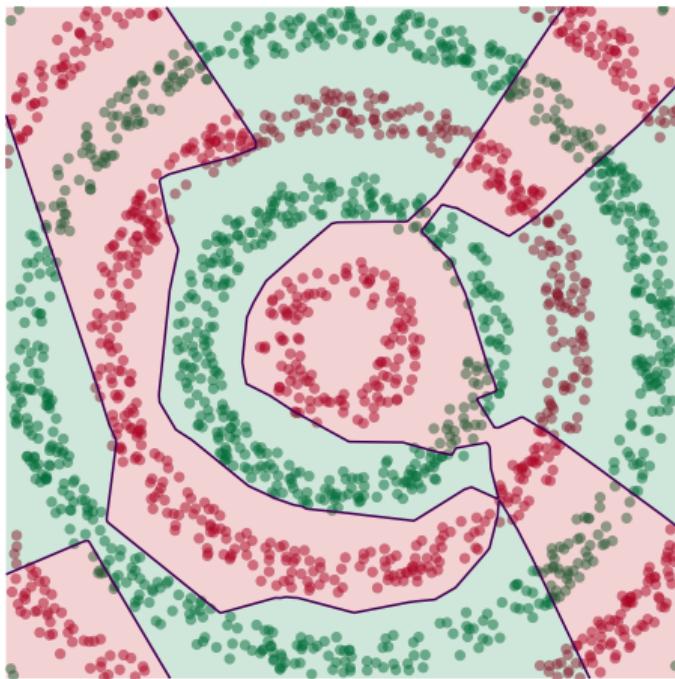
Learning a New Representation



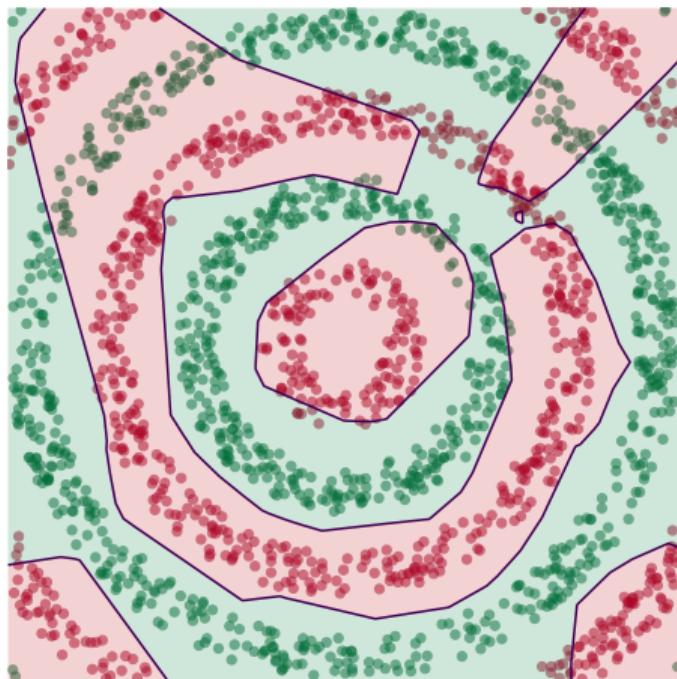
Learning a New Representation



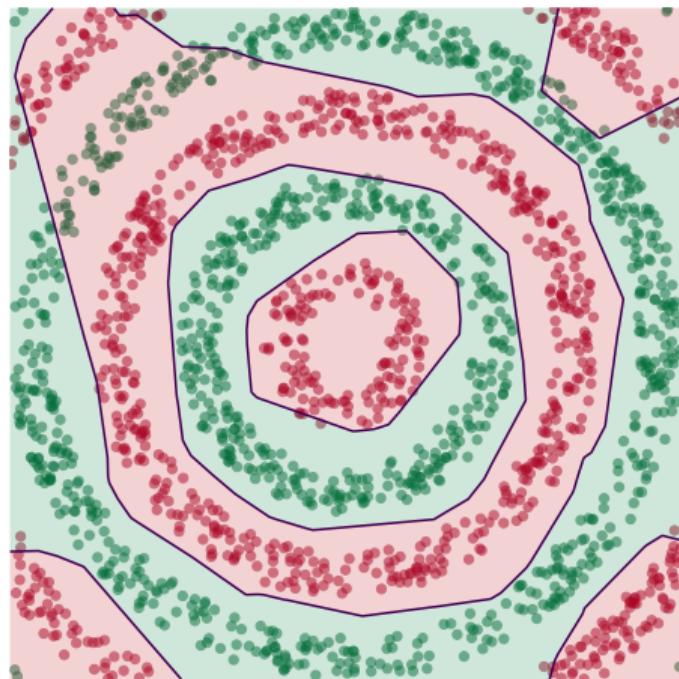
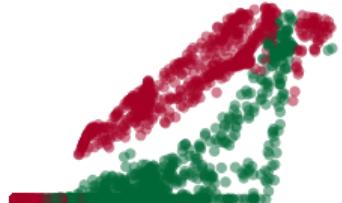
Learning a New Representation



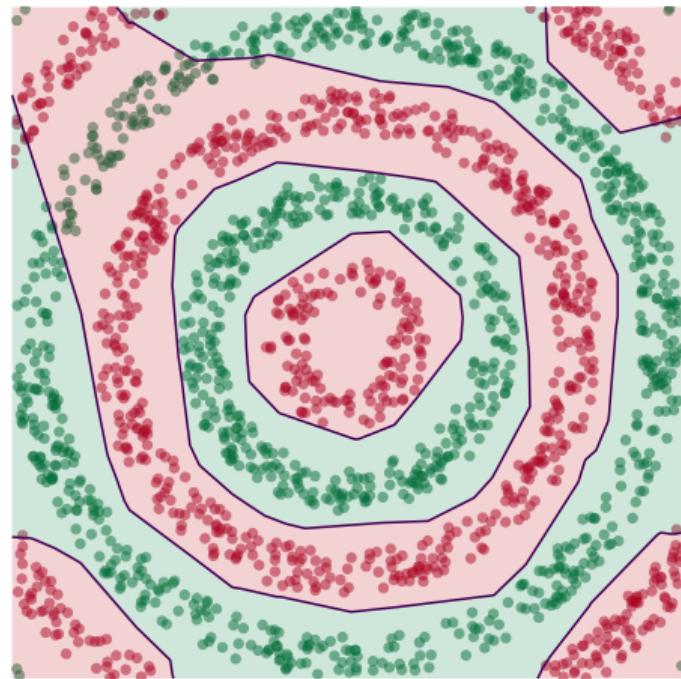
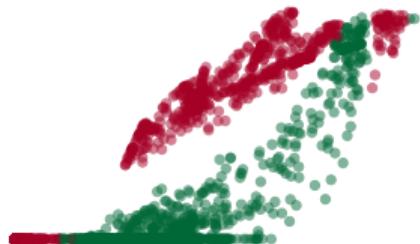
Learning a New Representation



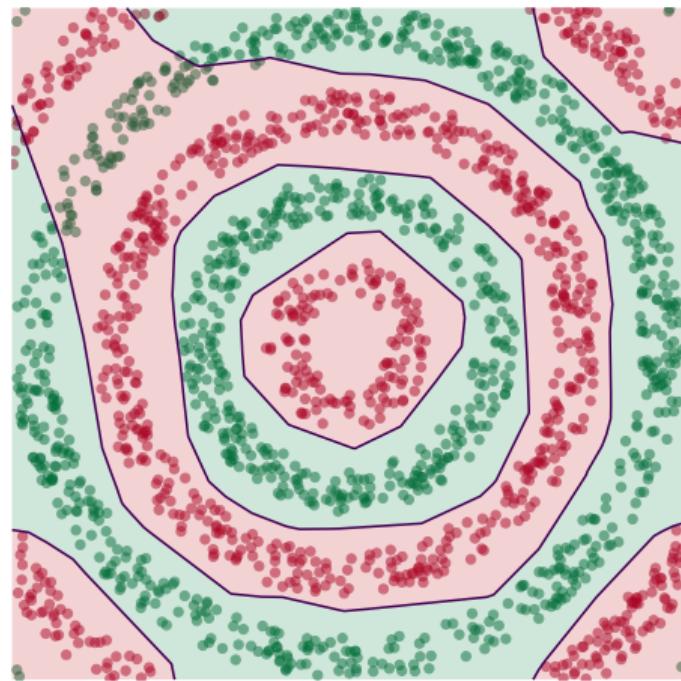
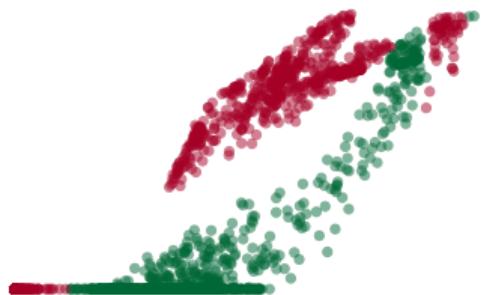
Learning a New Representation



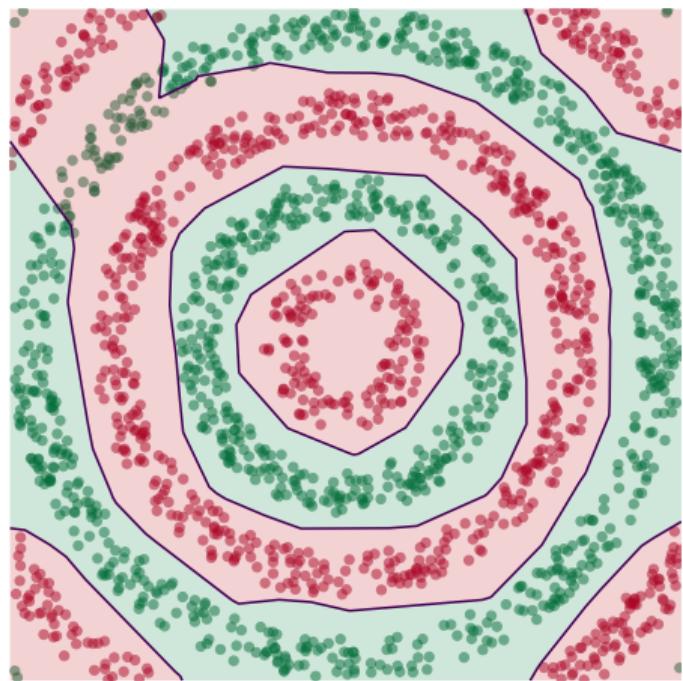
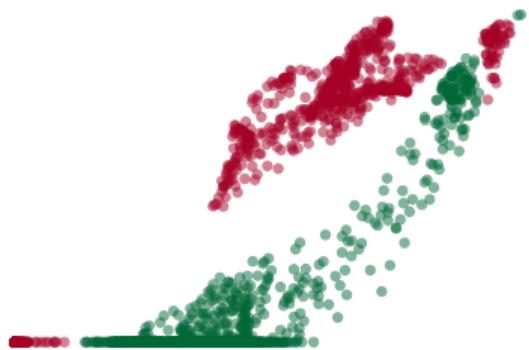
Learning a New Representation



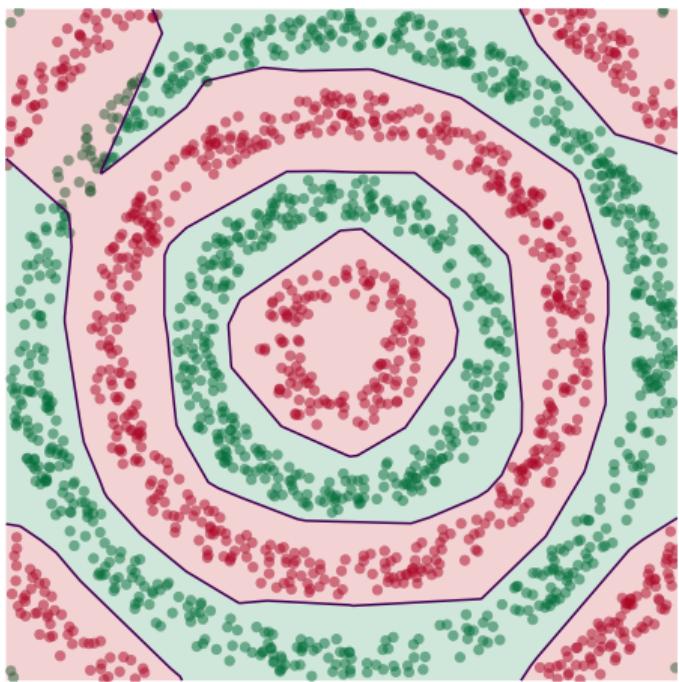
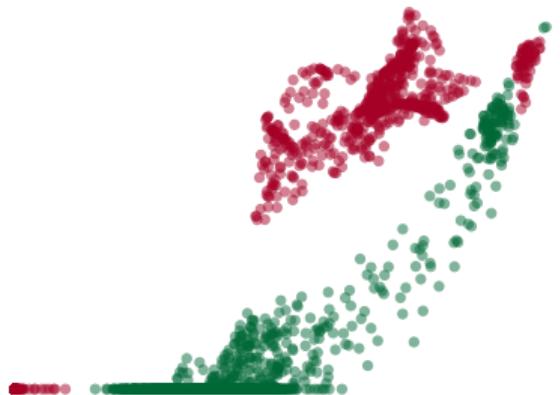
Learning a New Representation



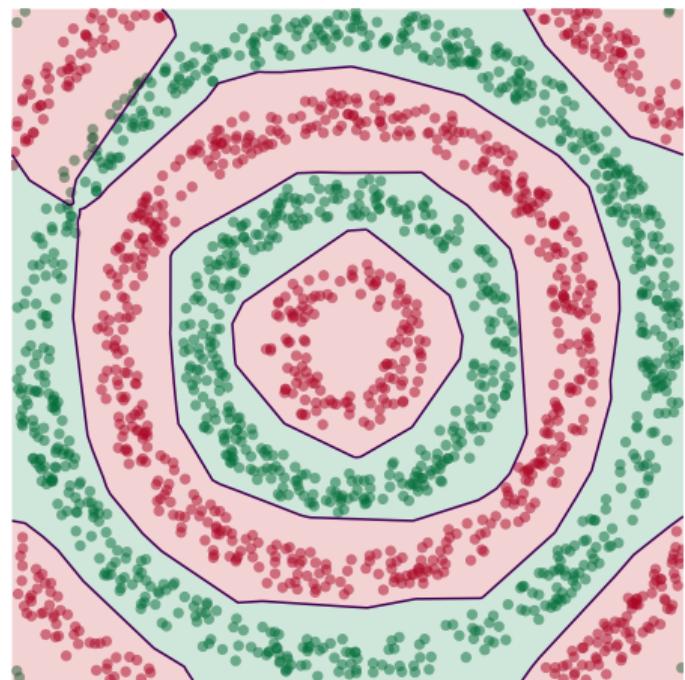
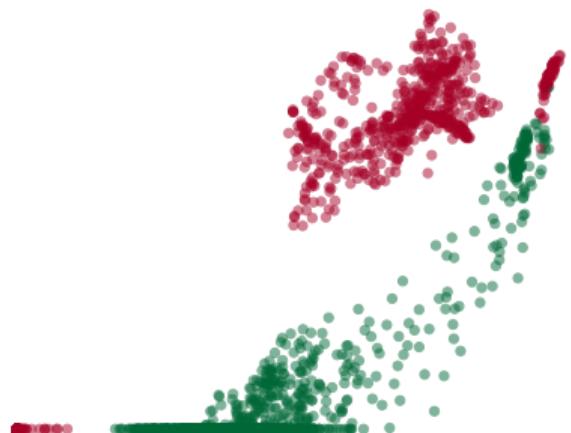
Learning a New Representation



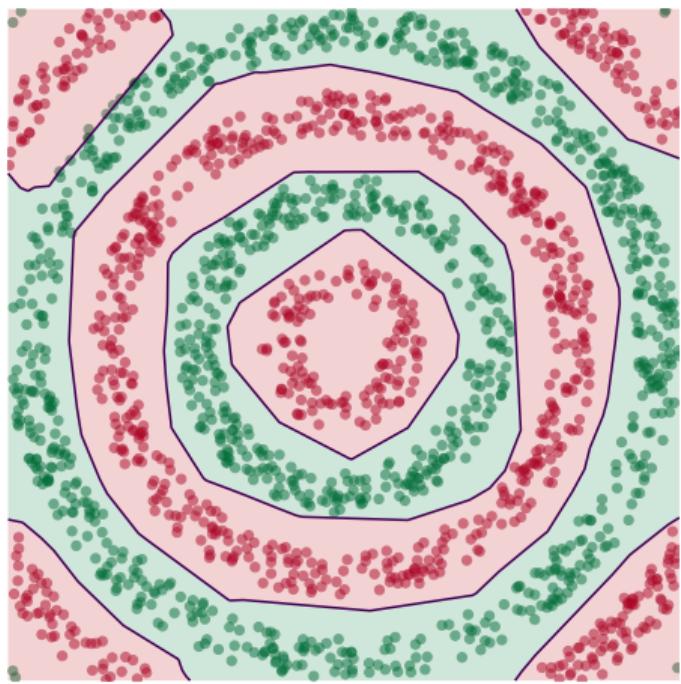
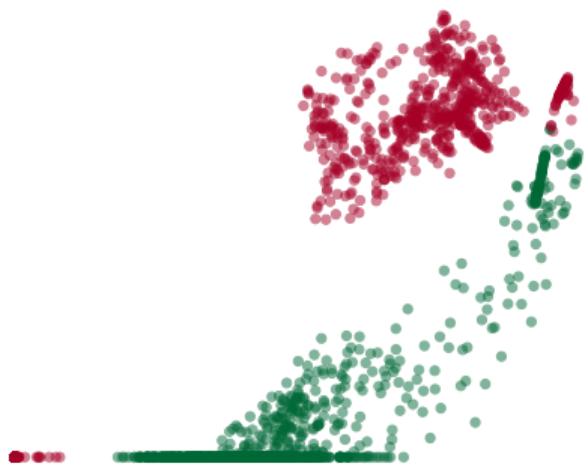
Learning a New Representation



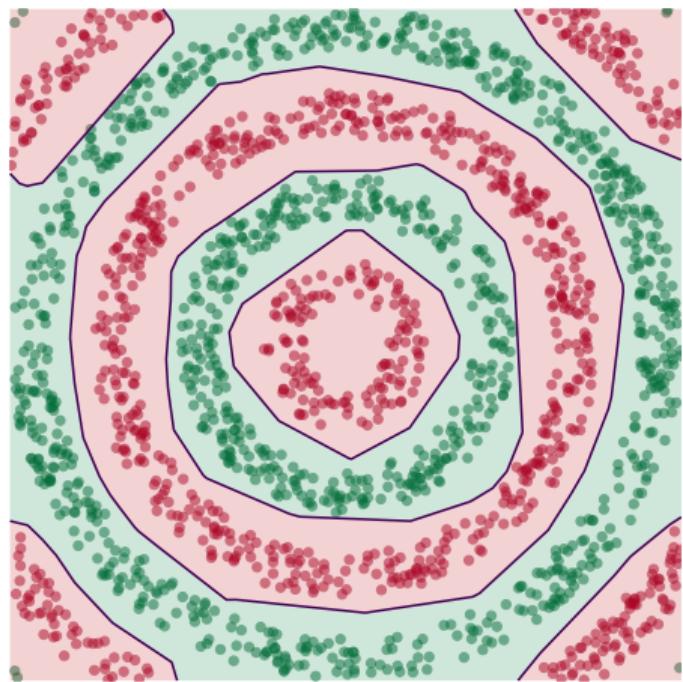
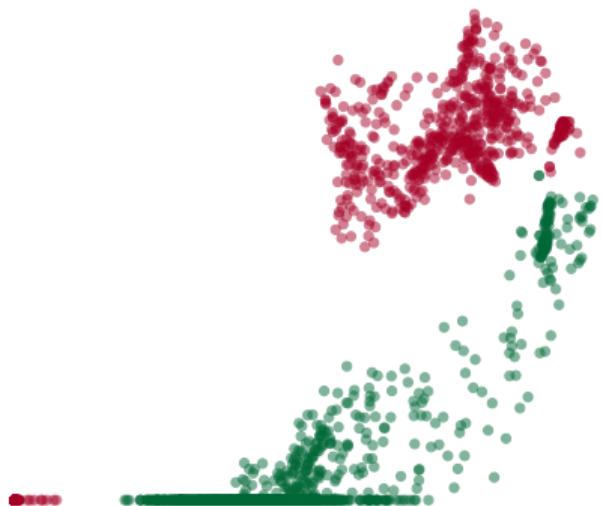
Learning a New Representation



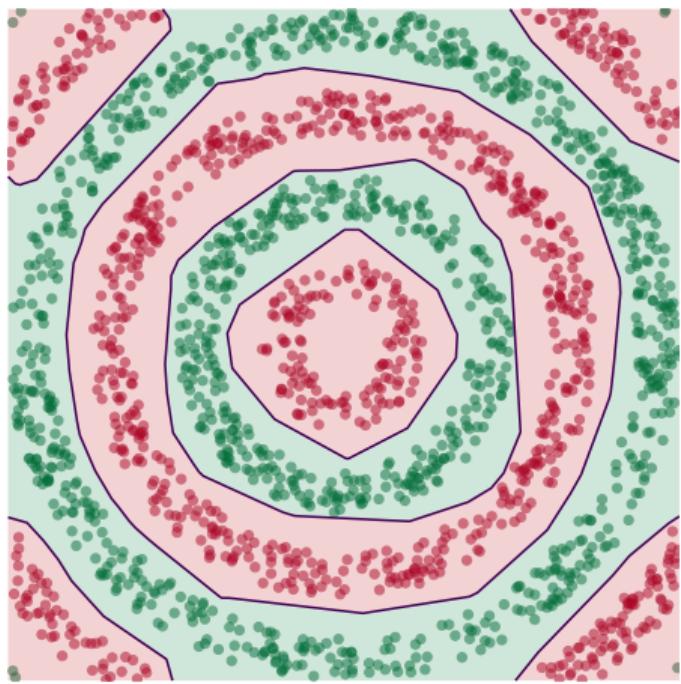
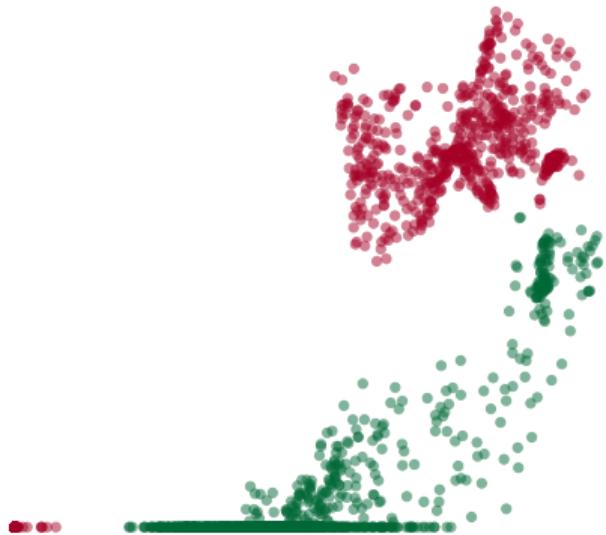
Learning a New Representation



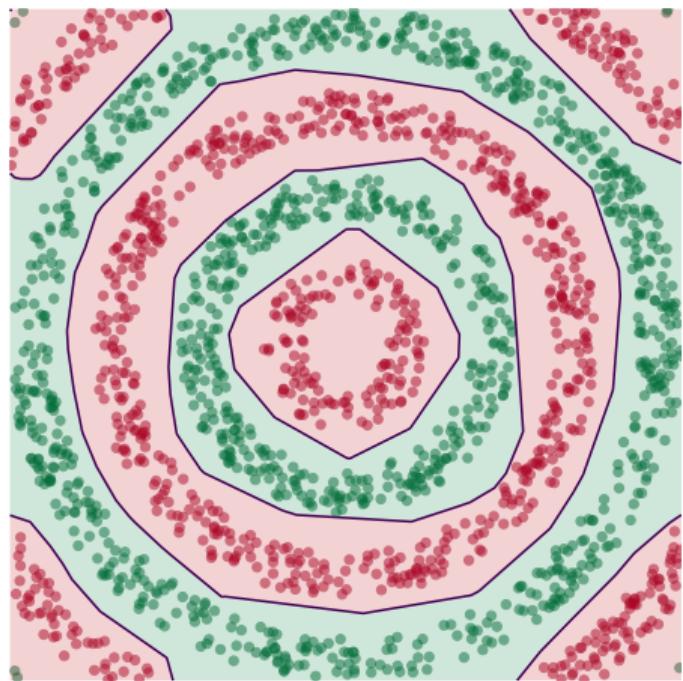
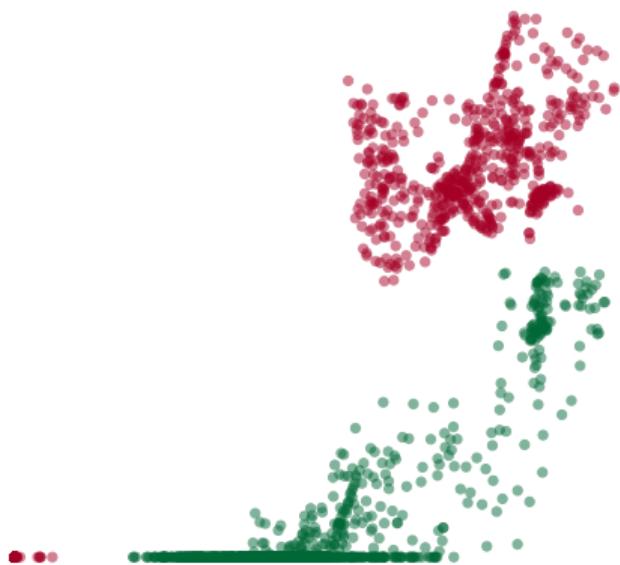
Learning a New Representation



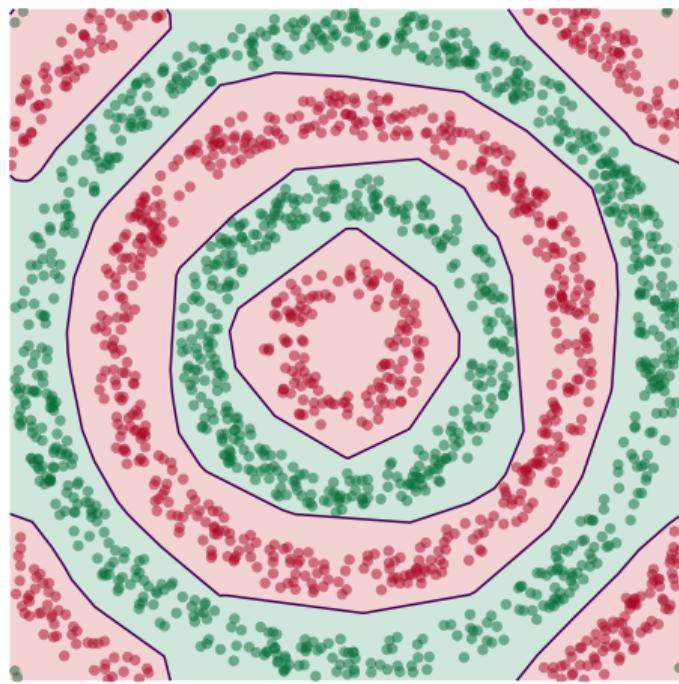
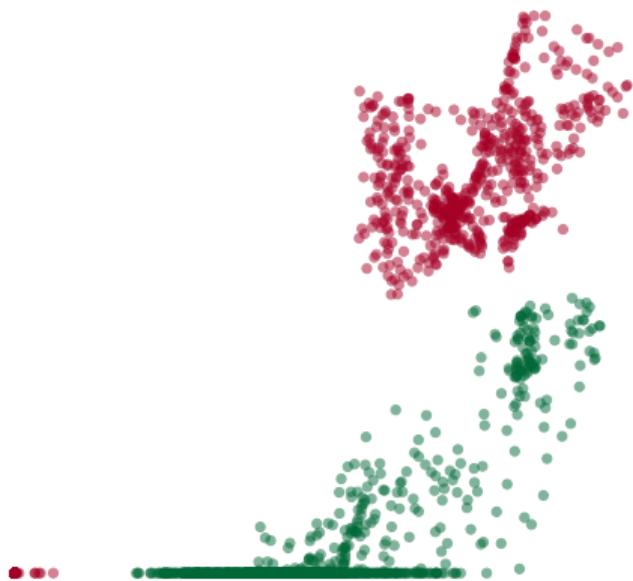
Learning a New Representation



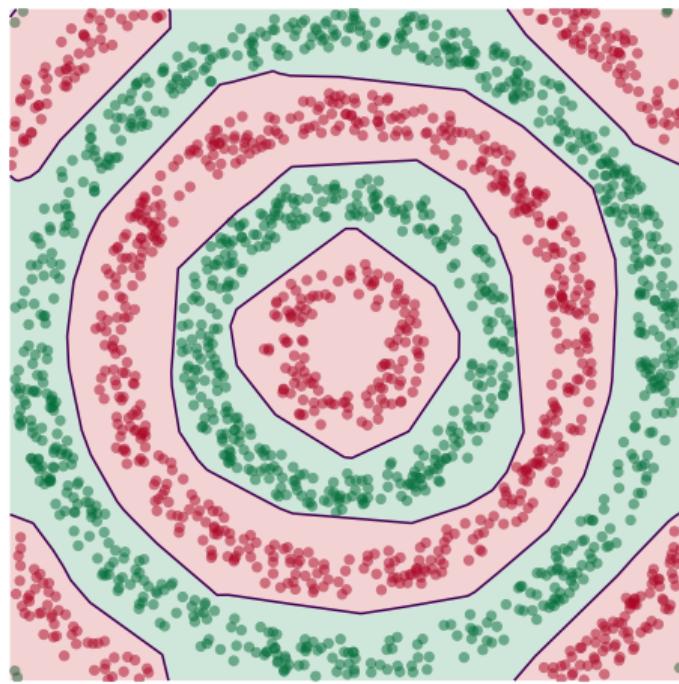
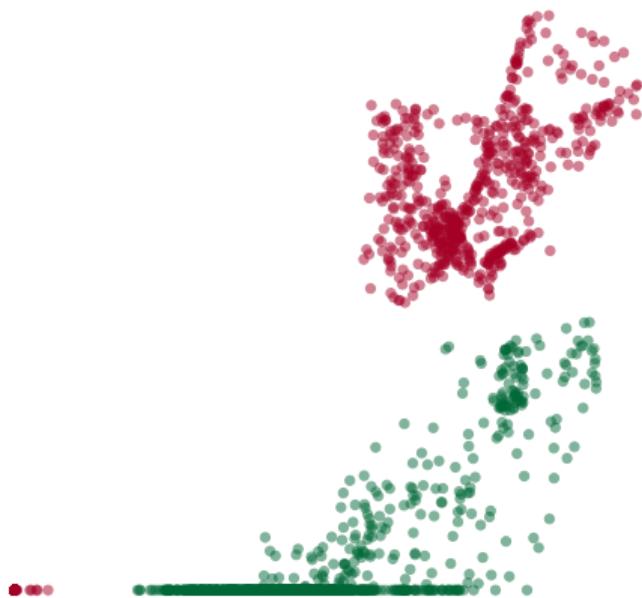
Learning a New Representation



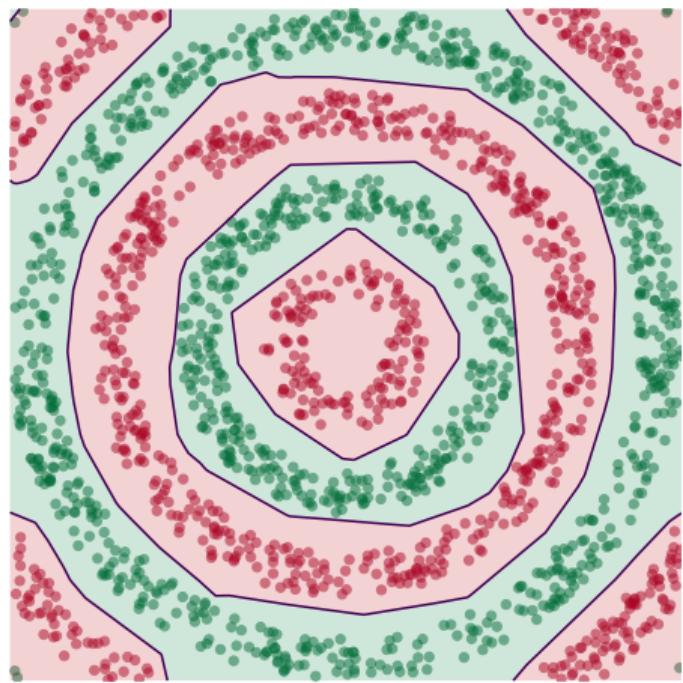
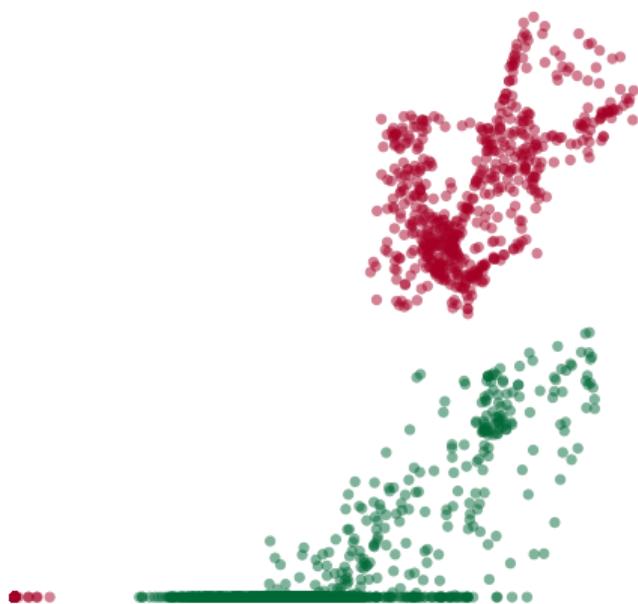
Learning a New Representation



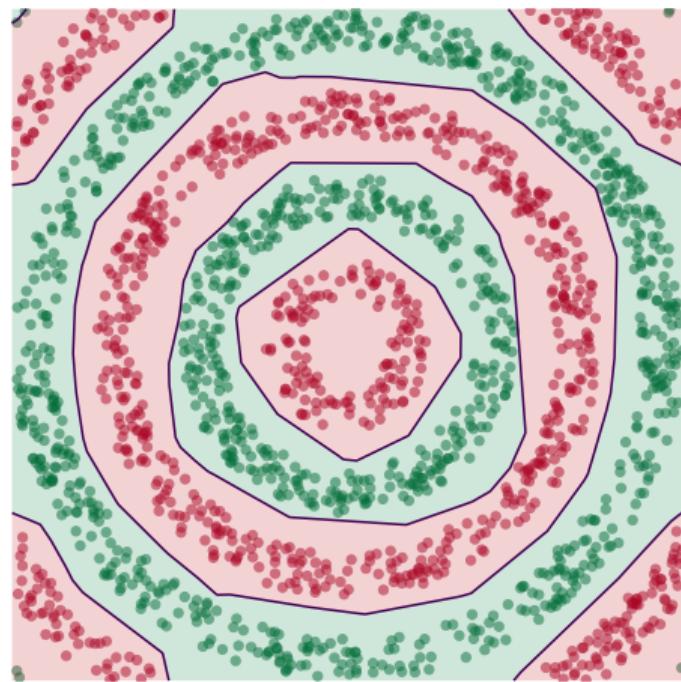
Learning a New Representation



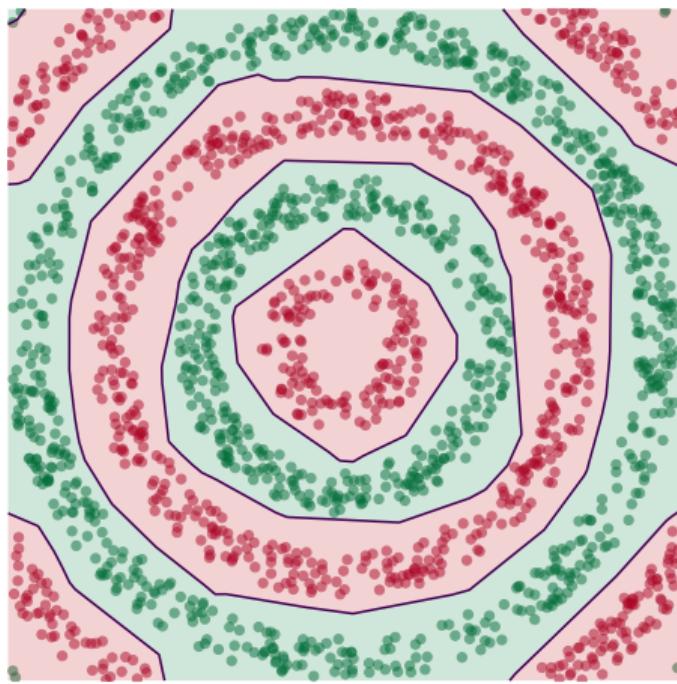
Learning a New Representation



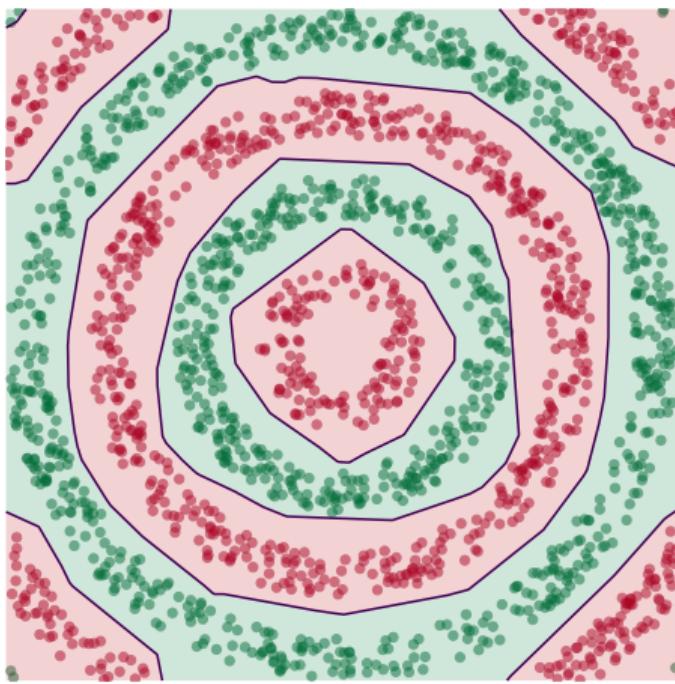
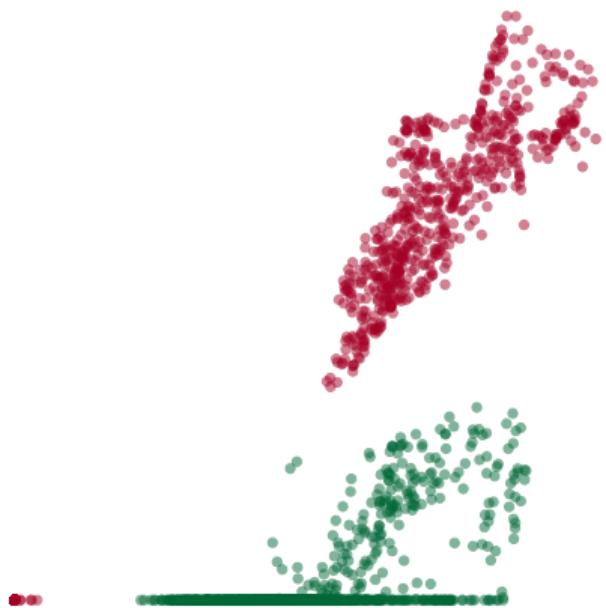
Learning a New Representation



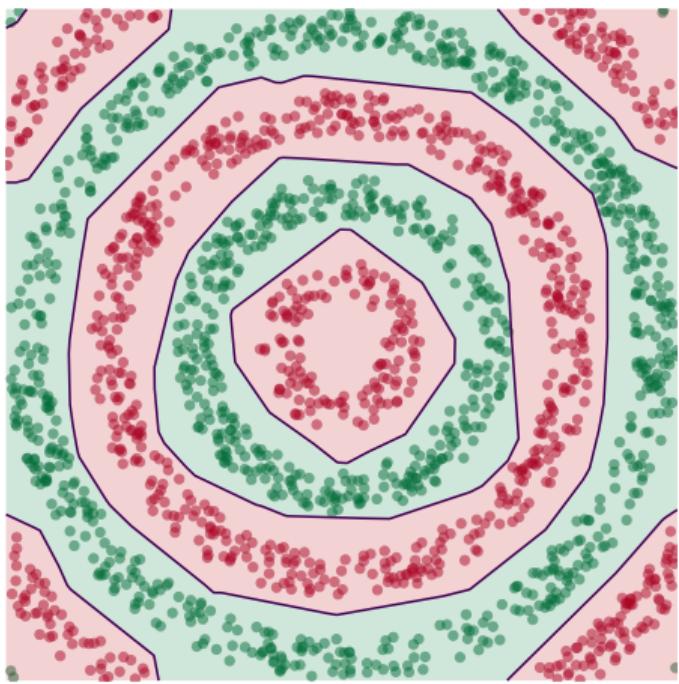
Learning a New Representation



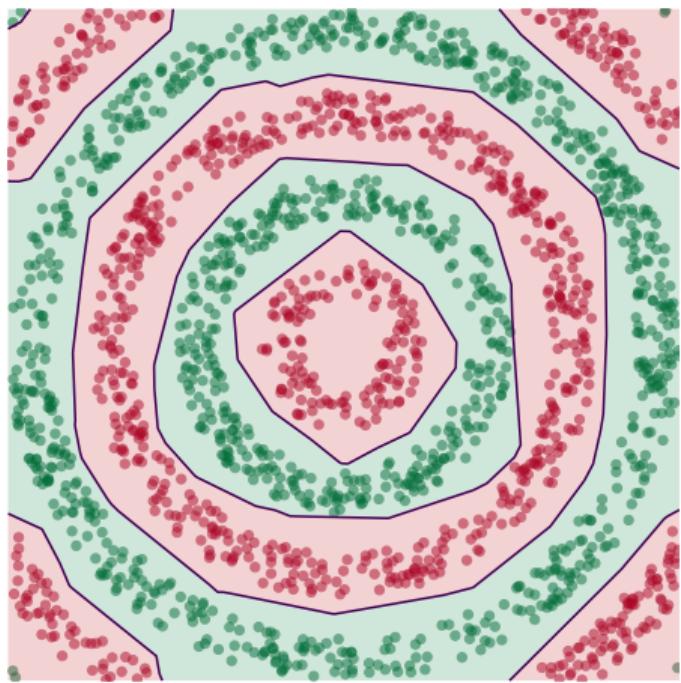
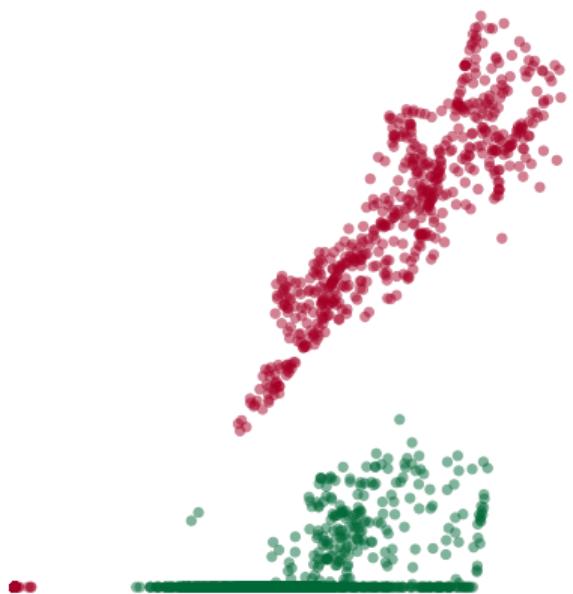
Learning a New Representation



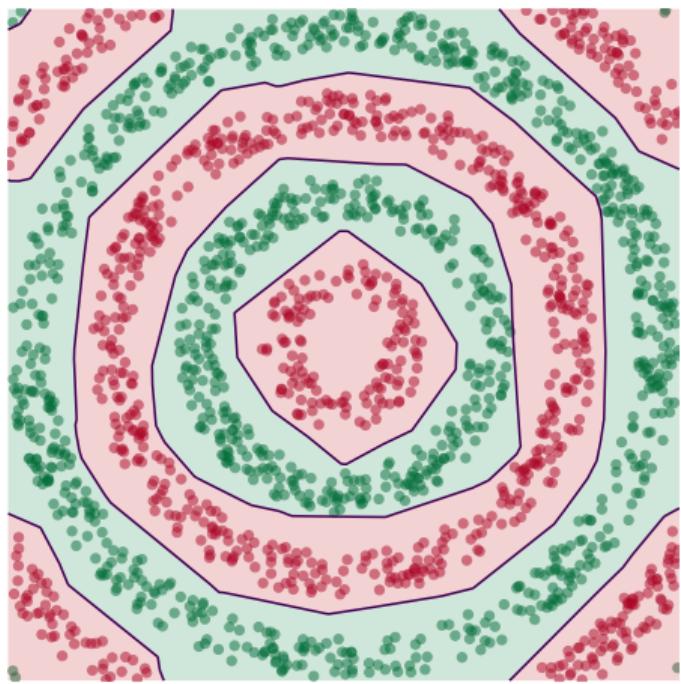
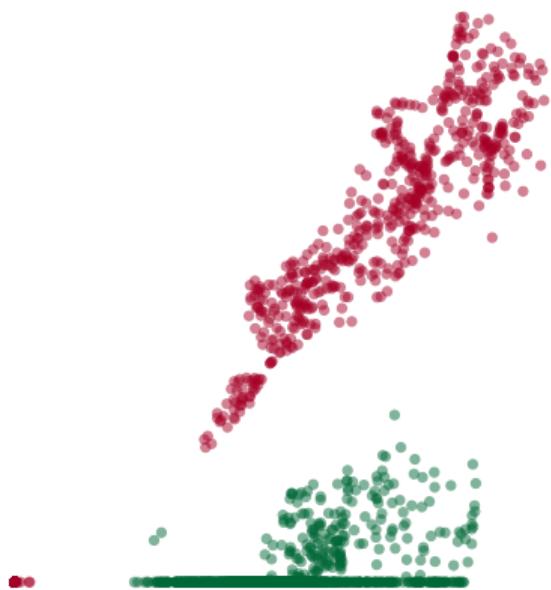
Learning a New Representation



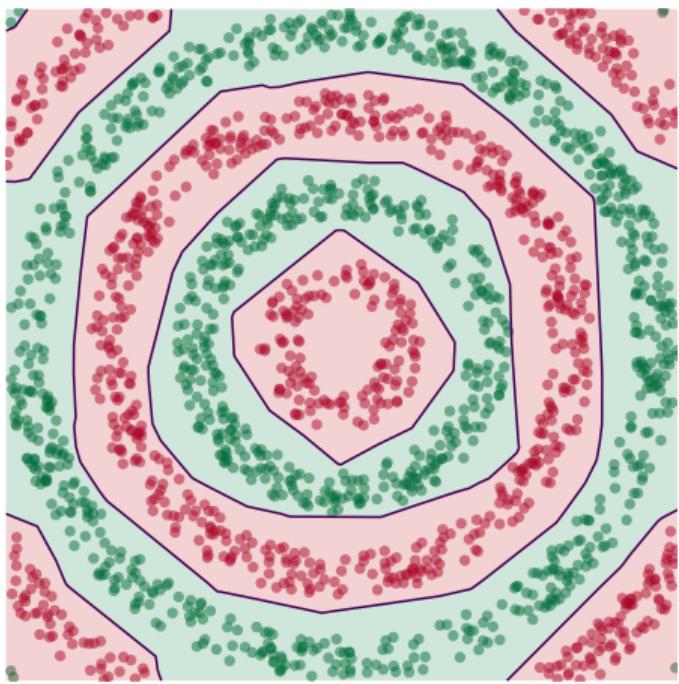
Learning a New Representation



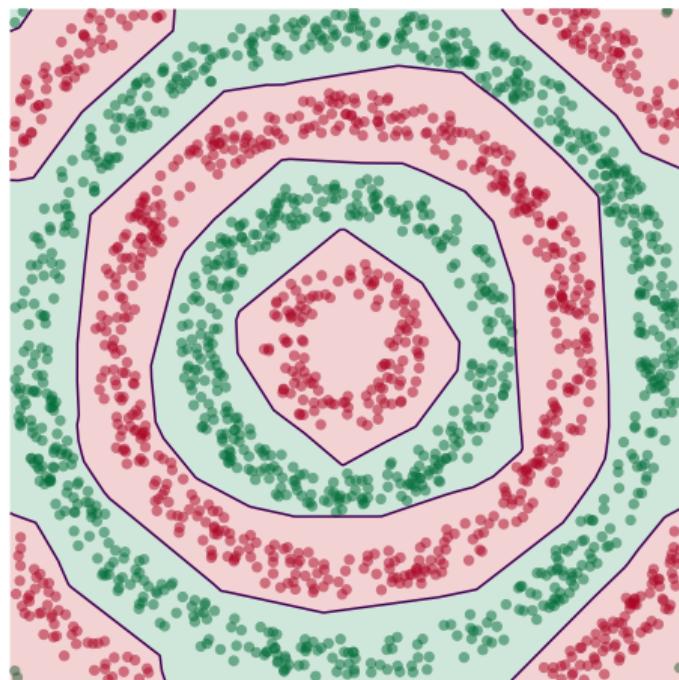
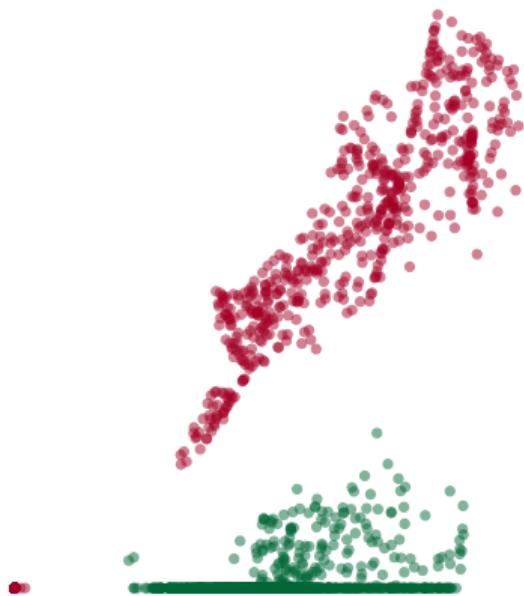
Learning a New Representation



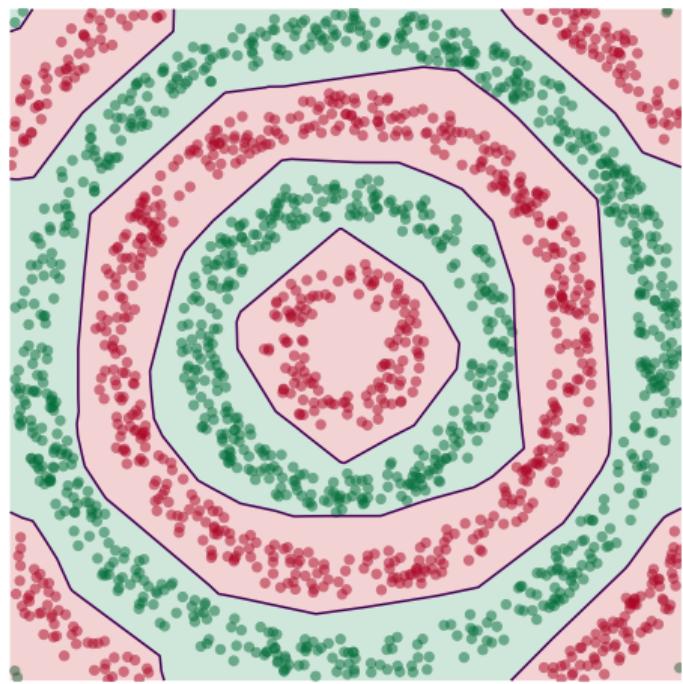
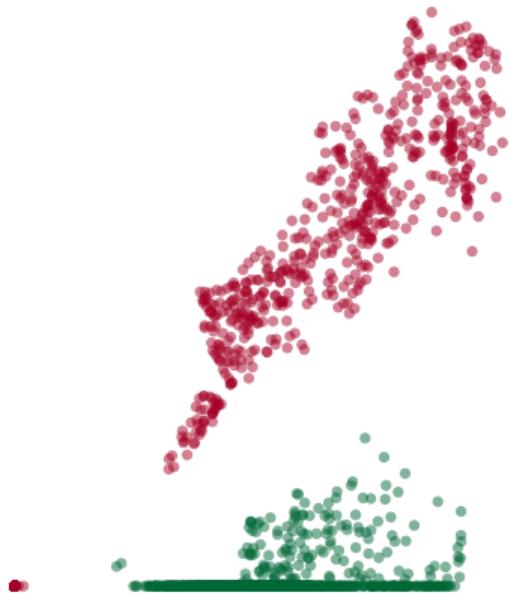
Learning a New Representation



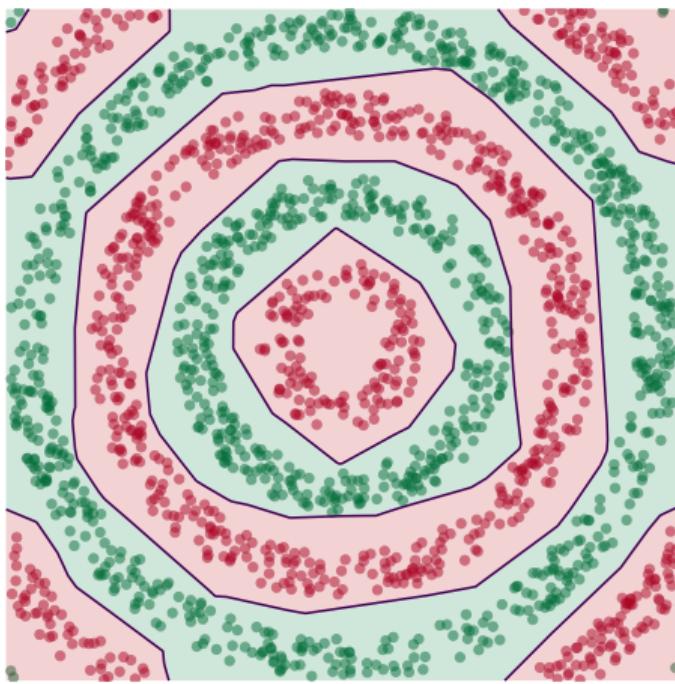
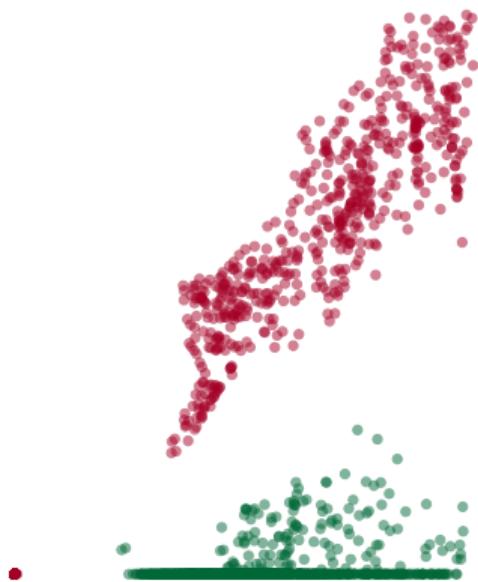
Learning a New Representation



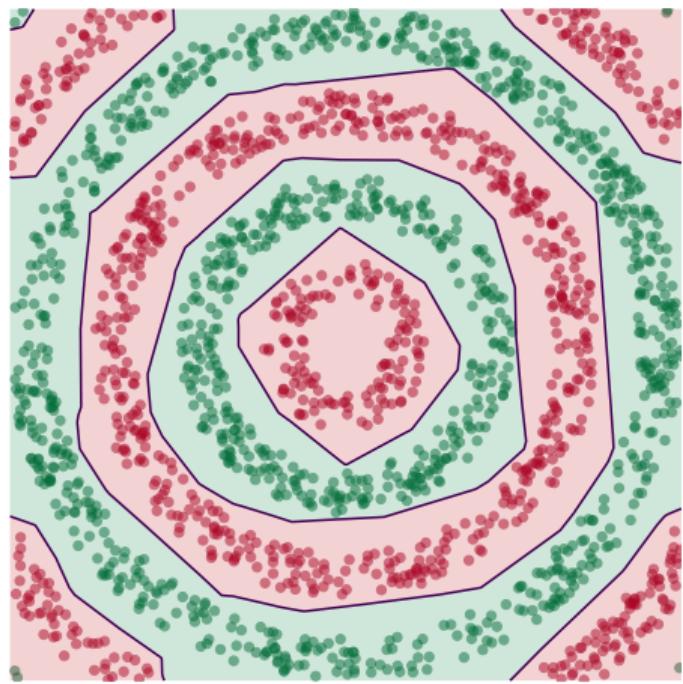
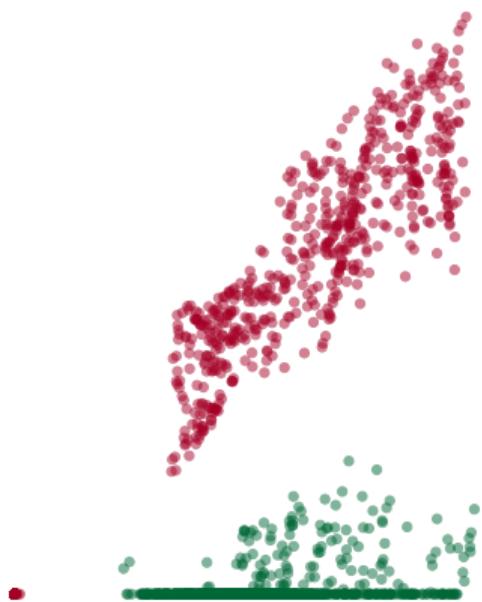
Learning a New Representation



Learning a New Representation



Learning a New Representation



Deep Networks and Approximation

- ▶ Deep networks are also universal approximators.
- ▶ May require fewer nodes and/or parameters than single hidden layer.
- ▶ I.e., there exist functions which require an exponential number of nodes to approximate with a single hidden layer, but not with several layers.

Challenges

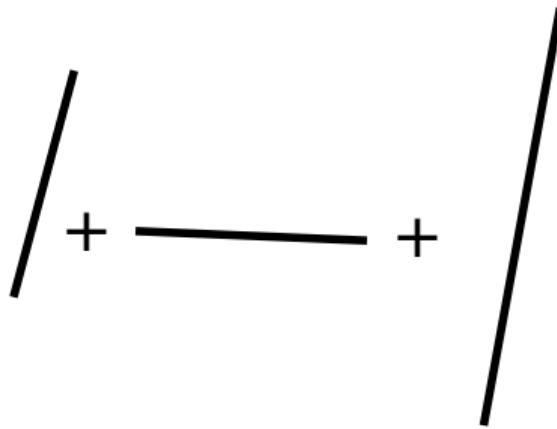
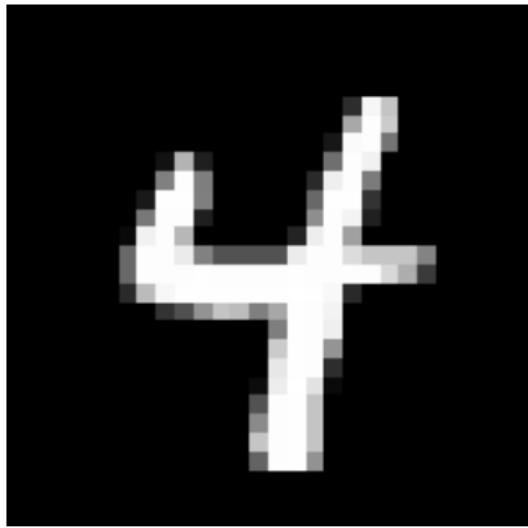
- ▶ The deeper the network, the weaker the gradient gets.
- ▶ Very non-convex!
- ▶ Deeper networks are harder to learn.

DSC 190

Machine Learning: Representations

Lecture 15 | Part 4

Convolutions

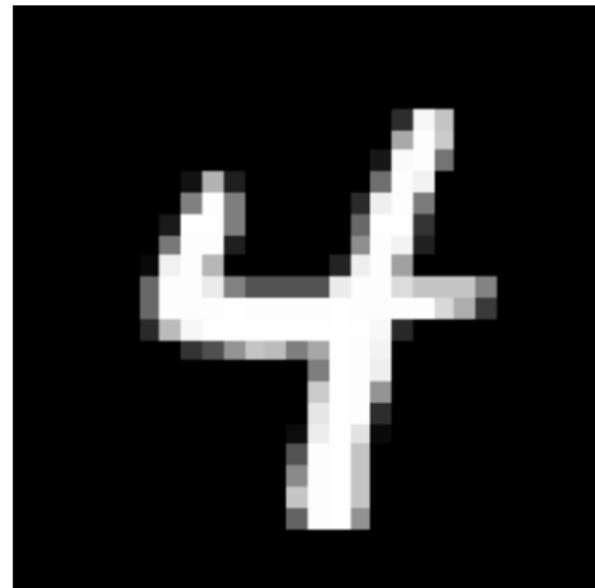


From Simple to Complex

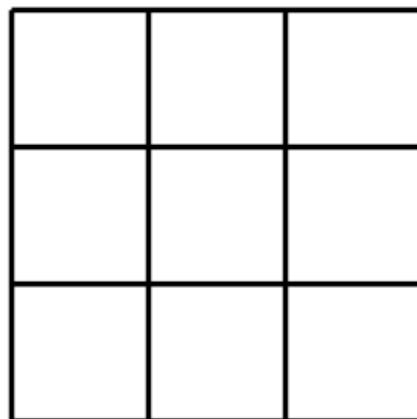
- ▶ Complex shapes are made of simple patterns
- ▶ The human visual system uses this fact
- ▶ Line detector → shape detector → ... → face detector
- ▶ Can we replicate this with a deep NN?

Edge Detector

- ▶ How do we find **vertical edges** in an image?
- ▶ One solution: **convolution** with an **edge filter**.



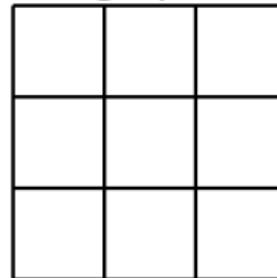
Vertical Edge Filter



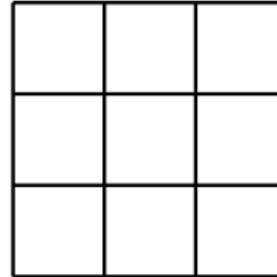
Idea

- ▶ Take a patch of the image, same size as filter.
- ▶ Perform “dot product” between patch and filter.
- ▶ If large, this is a (vertical) edge.

image patch:



filter:



Idea

- ▶ Move the filter over the entire image, repeat procedure.

$$\begin{array}{|c|c|c|} \hline 0 & 0 & 0 \\ \hline 0 & 0 & .9 \\ \hline 0 & 0 & .9 \\ \hline 0 & 0 & .8 \\ \hline 0 & 0 & .7 \\ \hline \end{array} \times \begin{array}{|c|c|c|} \hline \end{array} = \begin{array}{|c|c|c|} \hline \end{array}$$

The diagram illustrates the convolution process. On the left is a 5x3 input matrix with values 0, 0, 0 in the first row; 0, 0, 0.9 in the second; 0, 0, 0.9 in the third; 0, 0, 0.8 in the fourth; and 0, 0, 0.7 in the fifth. The second column of the input matrix is highlighted with a purple border. To its right is a multiplication sign (*). Further to the right is an equals sign (=) followed by a 5x5 output matrix, which is currently empty.

Idea

- ▶ Move the filter over the entire image, repeat procedure.

$$\begin{matrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & .9 & 0 & 0 & 0 & .7 \\ 0 & 0 & .9 & 0 & 0 & 0 & .8 \\ 0 & 0 & .8 & 0 & 0 & 0 & .9 \\ 0 & 0 & .7 & 0 & 0 & 0 & 0 \end{matrix} * \begin{matrix} \quad & \quad & \quad \\ \quad & \quad & \quad \\ \quad & \quad & \quad \end{matrix} = \begin{matrix} \quad & \quad & \quad & \quad & \quad & \quad \\ \quad & \quad & \quad & \quad & \quad & \quad \\ \quad & \quad & \quad & \quad & \quad & \quad \\ \quad & \quad & \quad & \quad & \quad & \quad \\ \quad & \quad & \quad & \quad & \quad & \quad \\ \quad & \quad & \quad & \quad & \quad & \quad \end{matrix}$$

Idea

- ▶ Move the filter over the entire image, repeat procedure.

$$\begin{array}{|c|c|c|c|c|c|} \hline 0 & 0 & 0 & 0 & 0 & 0 \\ \hline 0 & 0 & .9 & 0 & 0 & .7 \\ \hline 0 & 0 & .9 & 0 & 0 & .8 \\ \hline 0 & 0 & .8 & 0 & 0 & .9 \\ \hline 0 & 0 & .7 & 0 & 0 & 0 \\ \hline \end{array} * \begin{array}{|c|c|c|} \hline \end{array} = \begin{array}{|c|c|c|c|c|c|} \hline \end{array}$$

Idea

- ▶ Move the filter over the entire image, repeat procedure.

$$\begin{matrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & .9 & 0 & 0 & .7 \\ 0 & 0 & .9 & 0 & 0 & .8 \\ 0 & 0 & .8 & 0 & 0 & .9 \\ 0 & 0 & .7 & 0 & 0 & 0 \end{matrix} * \begin{matrix} \quad & \quad & \quad \\ \quad & \quad & \quad \\ \quad & \quad & \quad \end{matrix} = \begin{matrix} \quad & \quad & \quad & \quad & \quad & \quad \\ \quad & \quad & \quad & \quad & \quad & \quad \\ \quad & \quad & \quad & \quad & \quad & \quad \\ \quad & \quad & \quad & \quad & \quad & \quad \\ \quad & \quad & \quad & \quad & \quad & \quad \\ \quad & \quad & \quad & \quad & \quad & \quad \end{matrix}$$

Idea

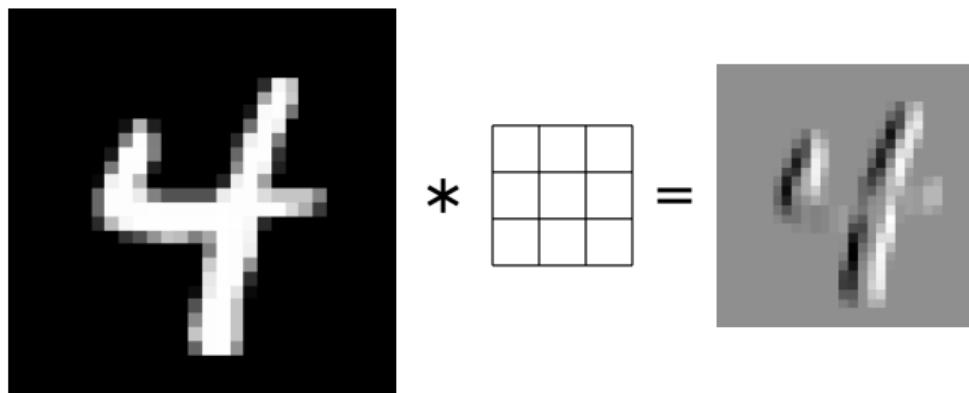
- ▶ Move the filter over the entire image, repeat procedure.

$$\begin{matrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & .9 & 0 & 0 & .7 \\ 0 & 0 & .9 & 0 & 0 & .8 \\ 0 & 0 & .8 & 0 & 0 & .9 \\ 0 & 0 & .7 & 0 & 0 & 0 \end{matrix} * \begin{matrix} \quad & \quad & \quad \\ \quad & \quad & \quad \\ \quad & \quad & \quad \end{matrix} = \begin{matrix} \quad & \quad & \quad & \quad & \quad & \quad \\ \quad & \quad & \quad & \quad & \quad & \quad \\ \quad & \quad & \quad & \quad & \quad & \quad \\ \quad & \quad & \quad & \quad & \quad & \quad \\ \quad & \quad & \quad & \quad & \quad & \quad \\ \quad & \quad & \quad & \quad & \quad & \quad \end{matrix}$$

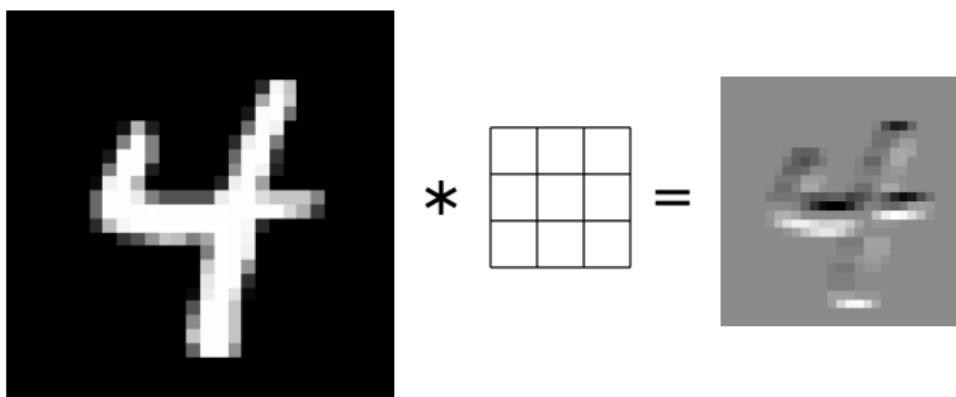
Convolution

- ▶ 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



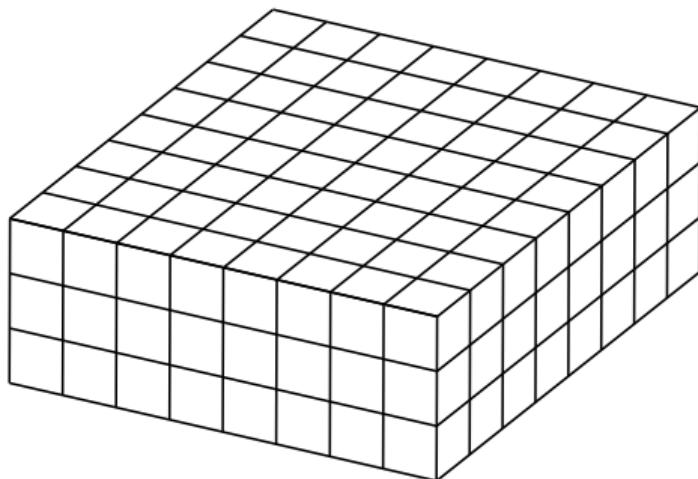
More About Filters

- ▶ Typically 3×3 or 5×5 .
- ▶ Variations: different **stride**, image **padding**.

3-d Filters

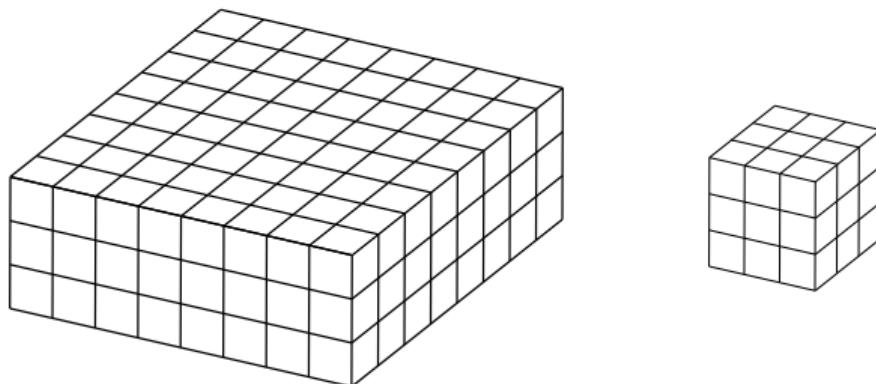
- ▶ Black and white images are 2-d arrays.
- ▶ But color images are 3-d arrays:
 - ▶ a.k.a., **tensors**
 - ▶ Three color **channels**: red, green, blue.
 - ▶ $\text{height} \times \text{width} \times 3$
- ▶ How does convolution work here?

Color Image

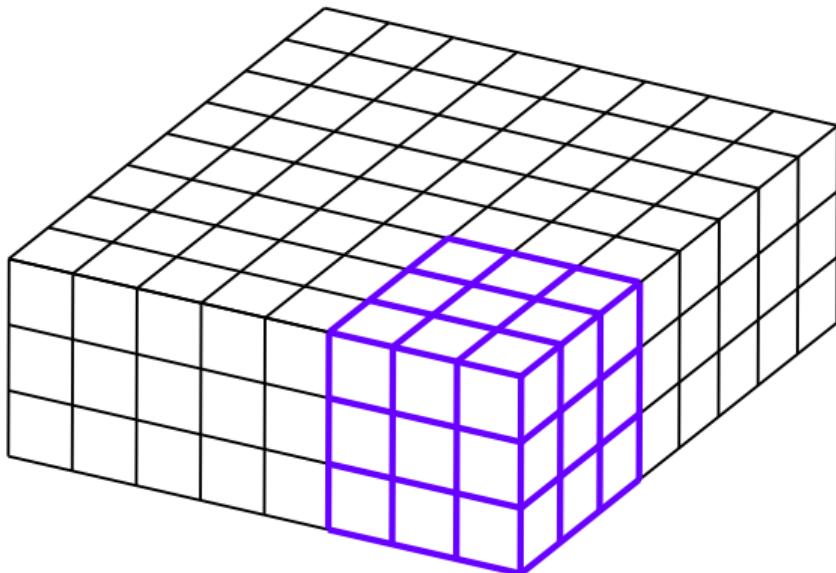


3-d Filter

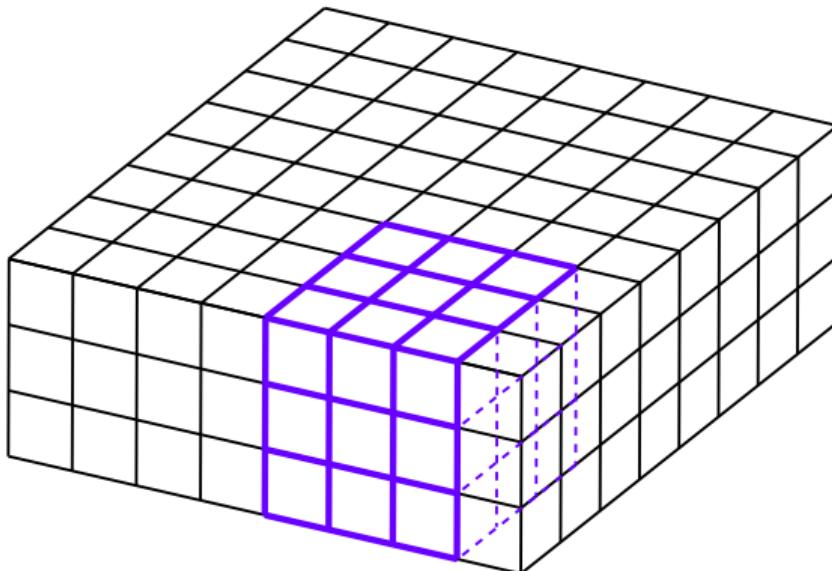
- ▶ The filter must also have three channels:
 - ▶ $3 \times 3 \times 3$, $5 \times 5 \times 3$, etc.



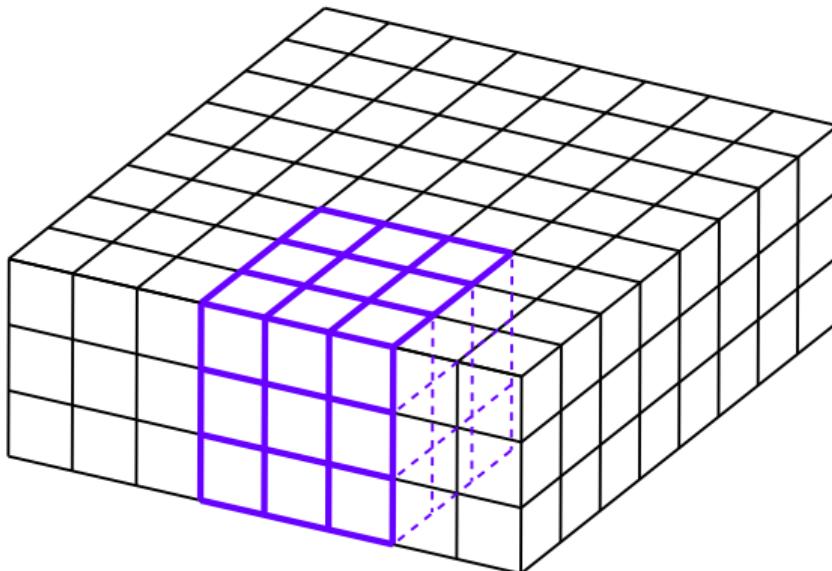
3-d Filter



3-d Filter



3-d Filter

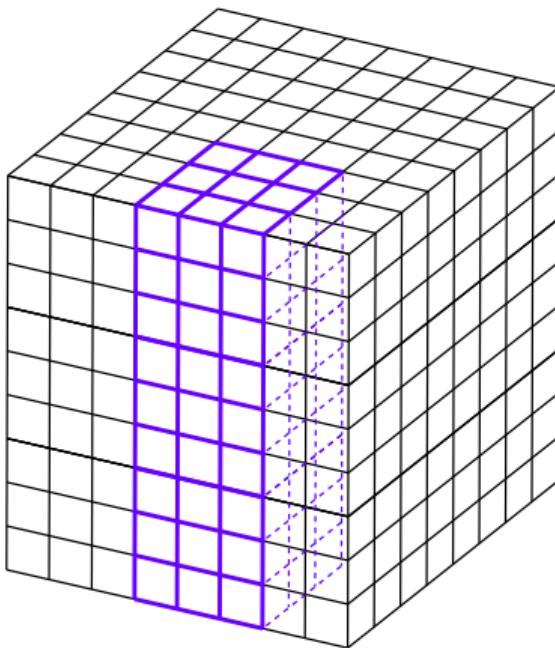


Convolution with 3-d Filter

- ▶ 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 $2 - d$

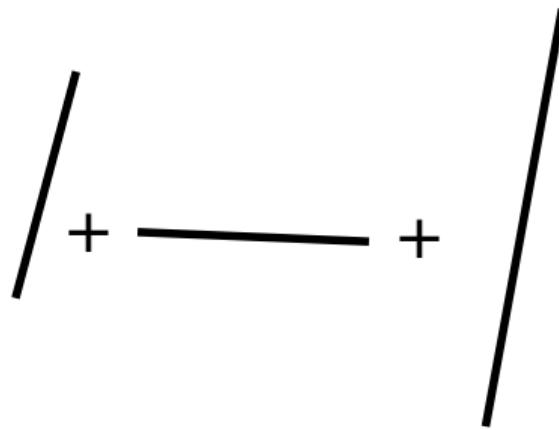
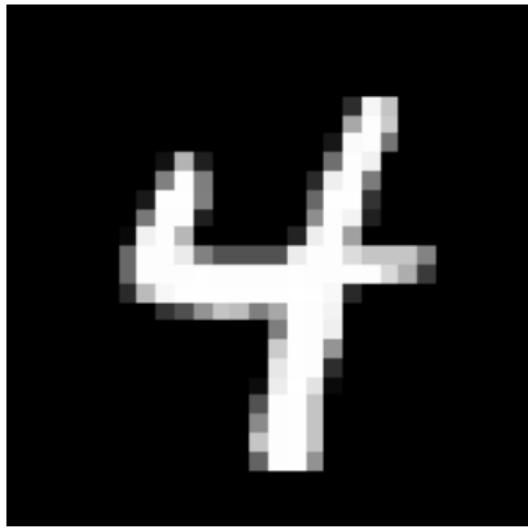


DSC 190

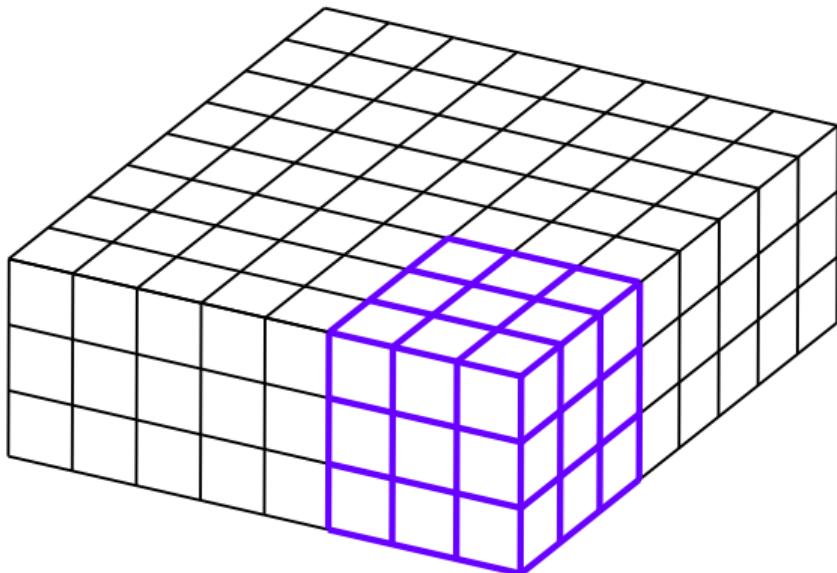
Machine Learning: Representations

Lecture 16 | Part 1

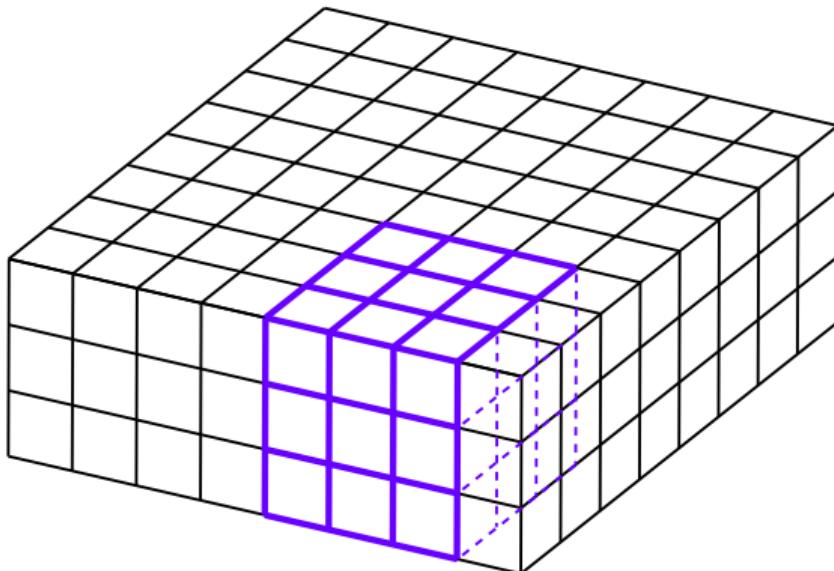
Convolutions



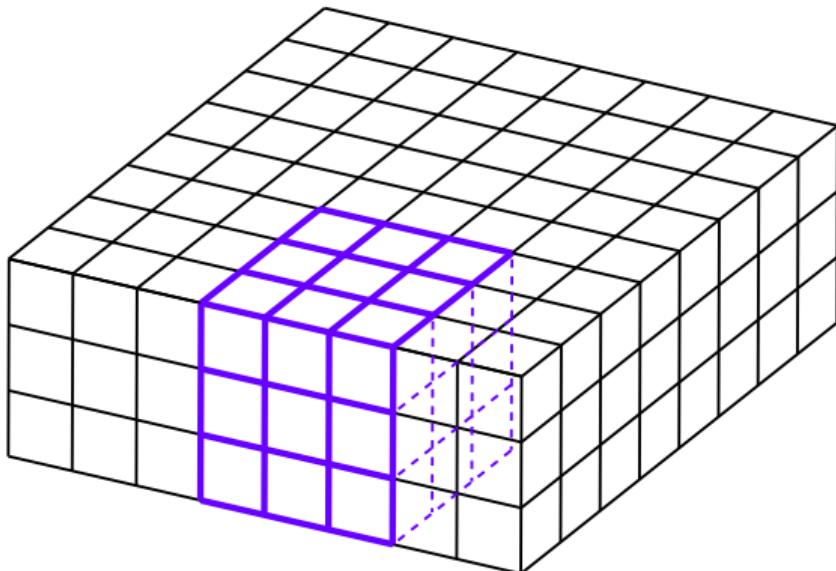
3-d Filter



3-d Filter

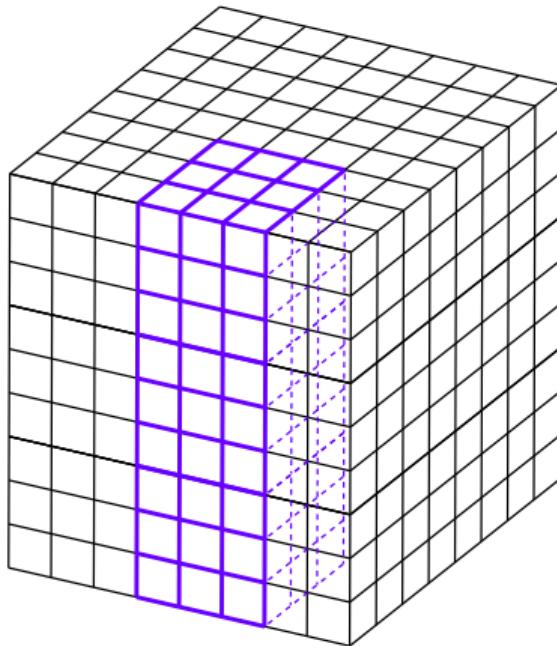


3-d Filter



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 190

Machine Learning: Representations

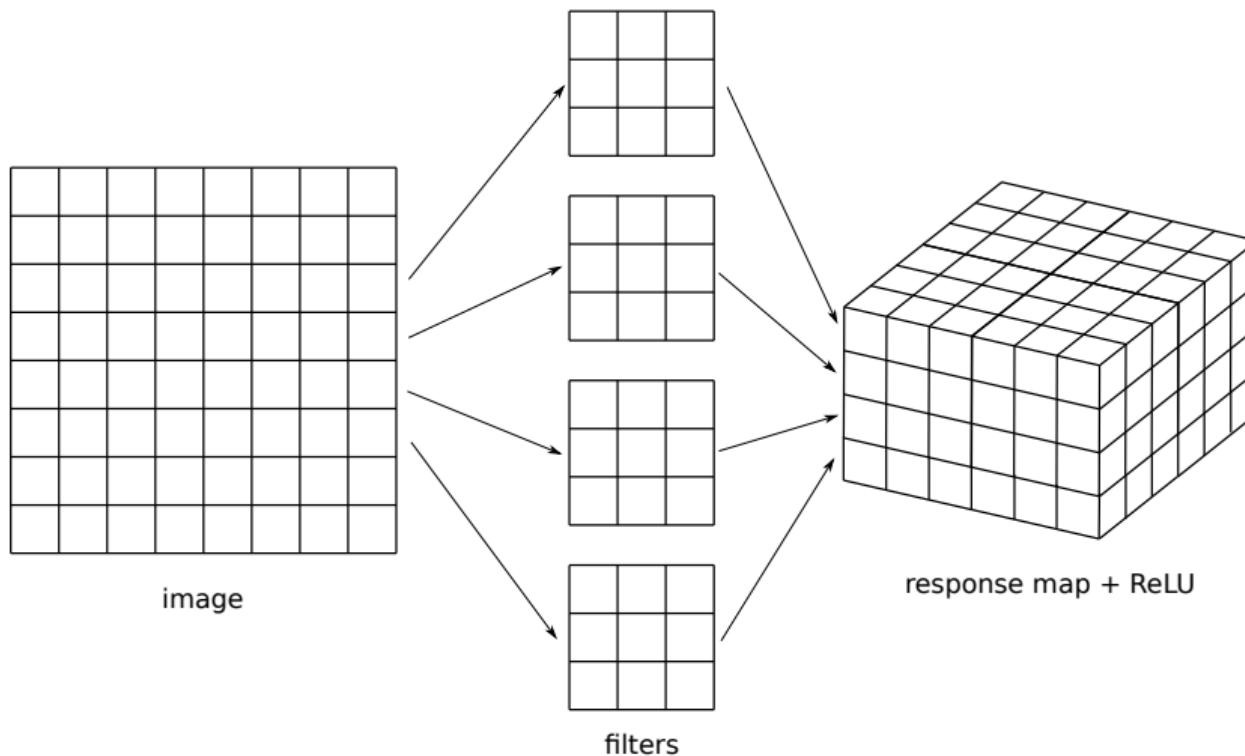
Lecture 16 | Part 2

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**.

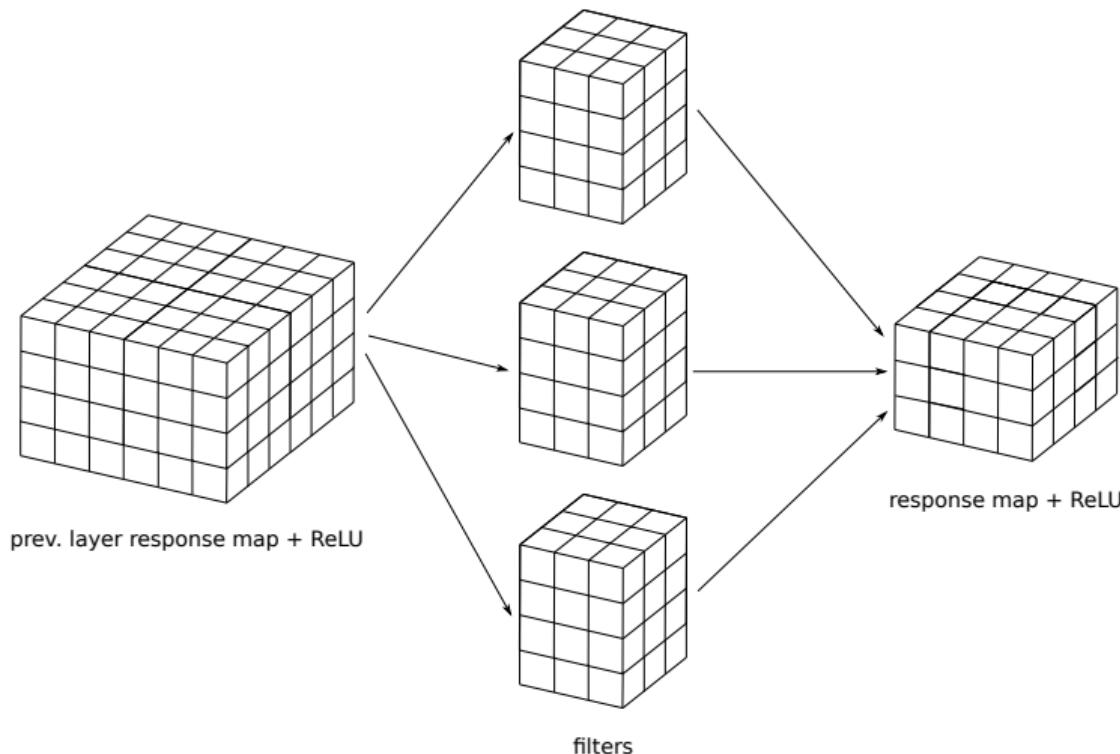
Input Convolutional Layer



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



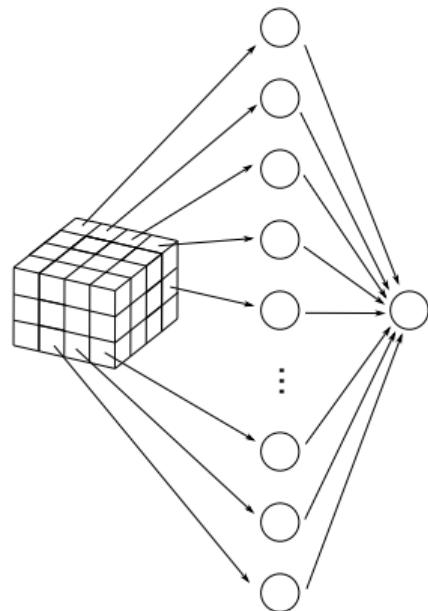
Second Convolutional Layer

- ▶ 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.

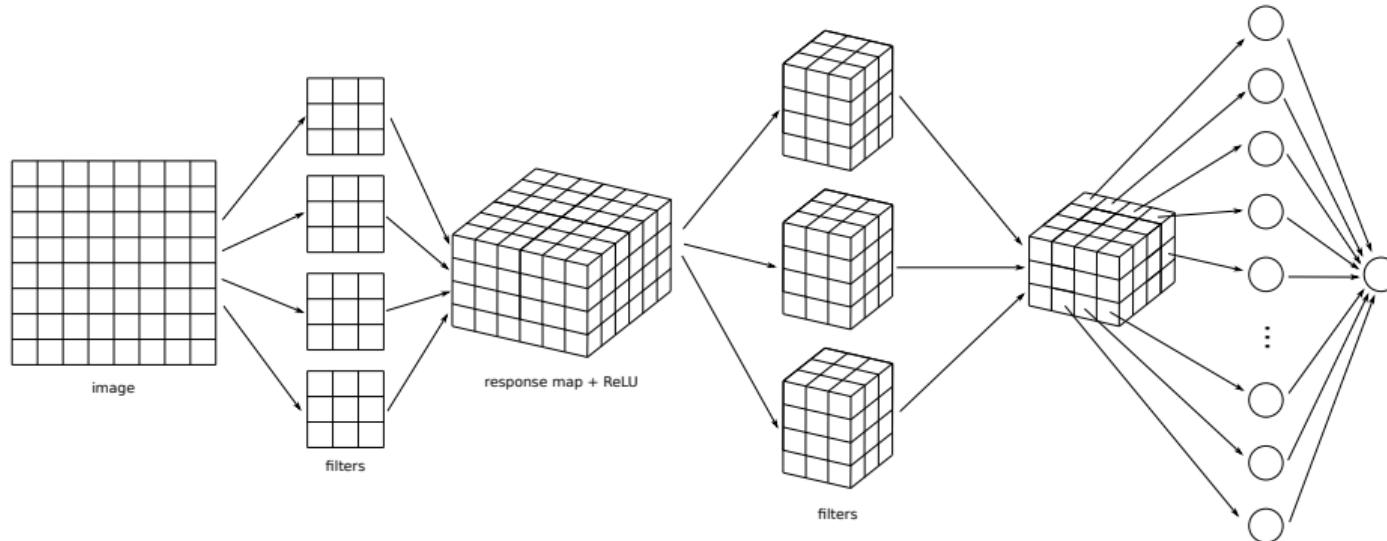
More Convolutional Layers

- ▶ 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

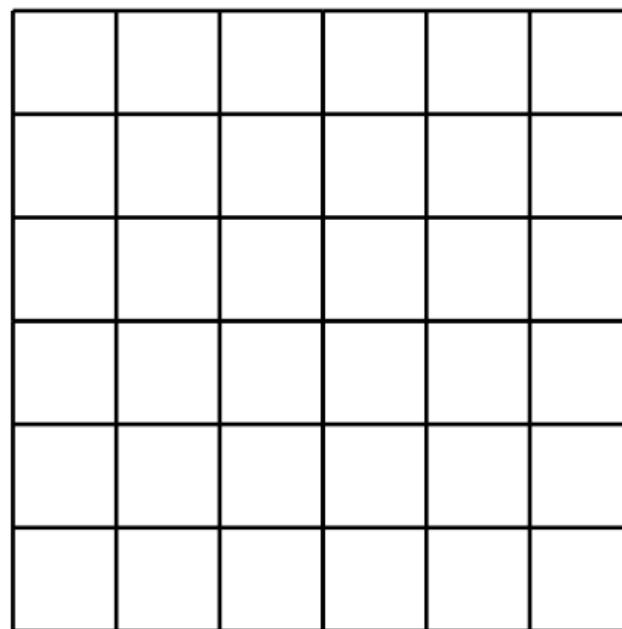


What is learned?

- ▶ 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.



DSC 190

Machine Learning: Representations

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**.



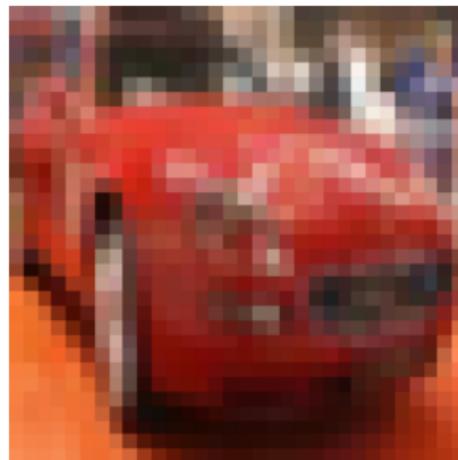
Problem

- ▶ Predict whether image is of a **car** or a **truck**.



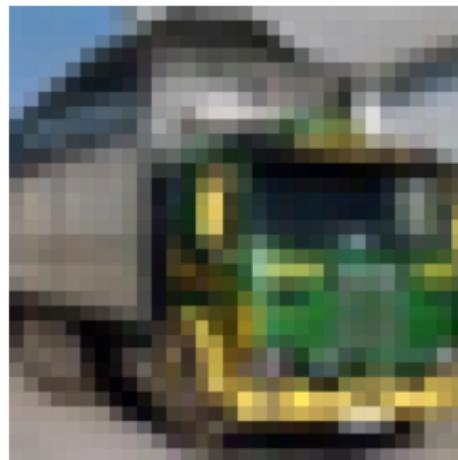
Problem

- ▶ Predict whether image is of a **car** or a **truck**.



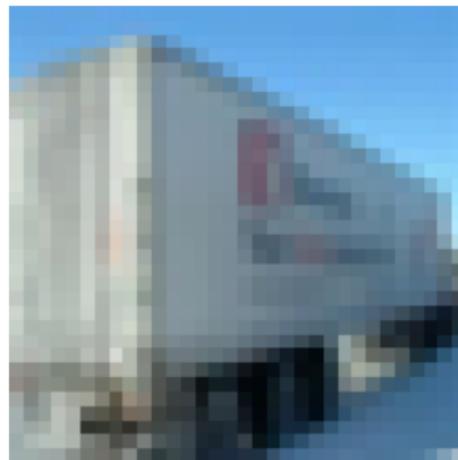
Problem

- ▶ Predict whether image is of a **car** or a **truck**.



Problem

- ▶ Predict whether image is of a **car** or a **truck**.



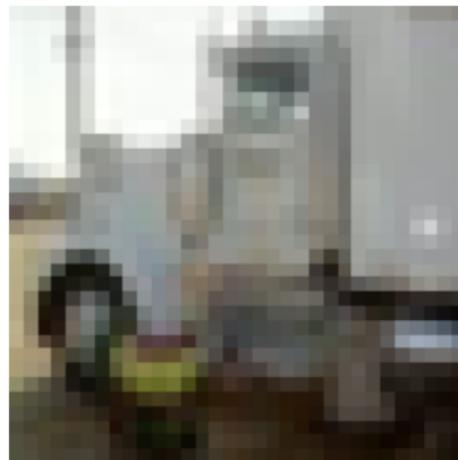
Problem

- ▶ 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**.



Problem

- ▶ Predict whether image is of a **car** or a **truck**.



Details

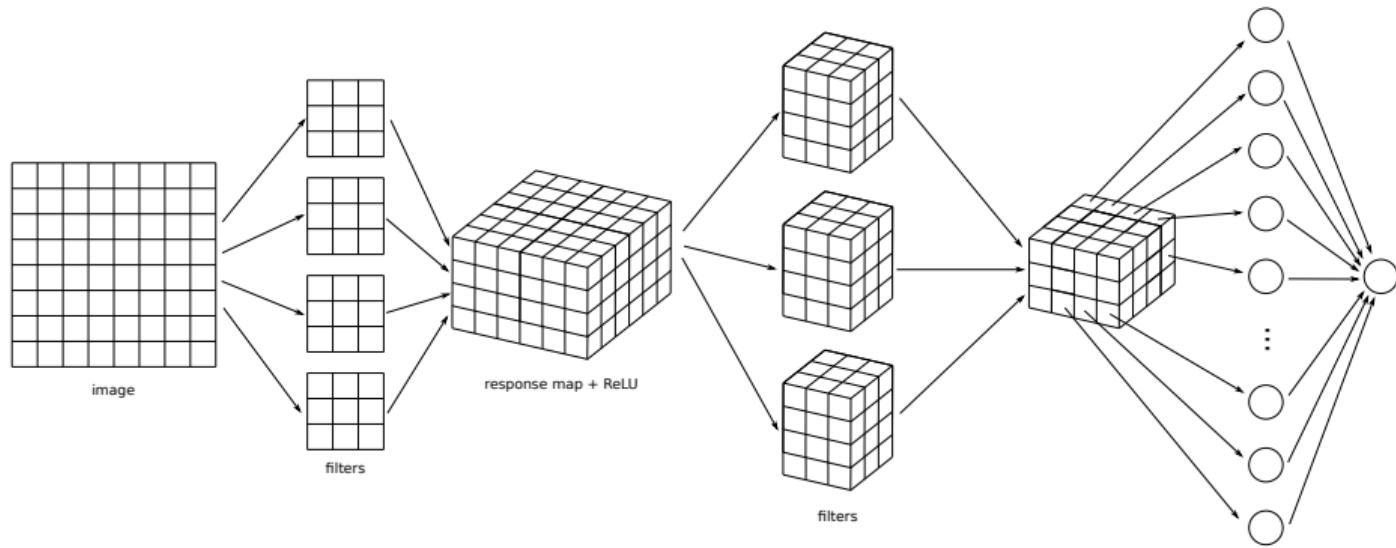
- ▶ 3-channel 32×32 color images
- ▶ 10,000 training images; 2,000 test¹
- ▶ Cars, trucks in different orientations, scales
- ▶ Balanced: 50% cars, 50% trucks

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

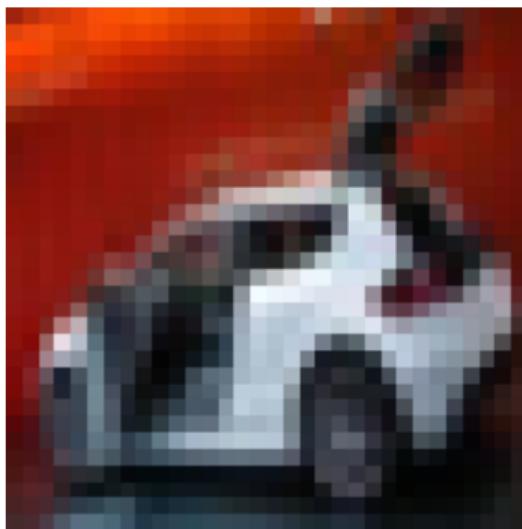
```
model.compile(  
    optimizer=keras.optimizers.RMSprop(),  
    loss=keras.losses.BinaryCrossentropy(),  
    metrics=['accuracy'])  
  
model.fit(  
    X_train,  
    y_train,  
    epochs=30,  
    validation_data=(X_test, y_test))
```

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



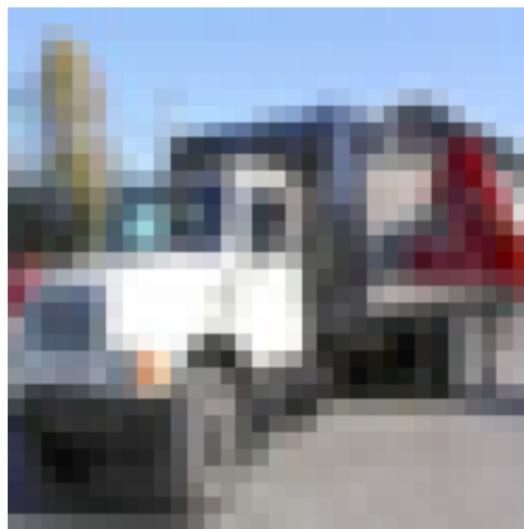
Results

truck / truck



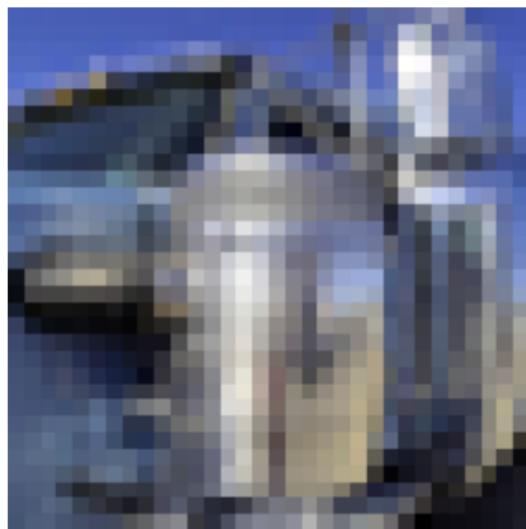
Results

truck / truck



Results

truck / truck



Results

truck / truck



Results

car / car



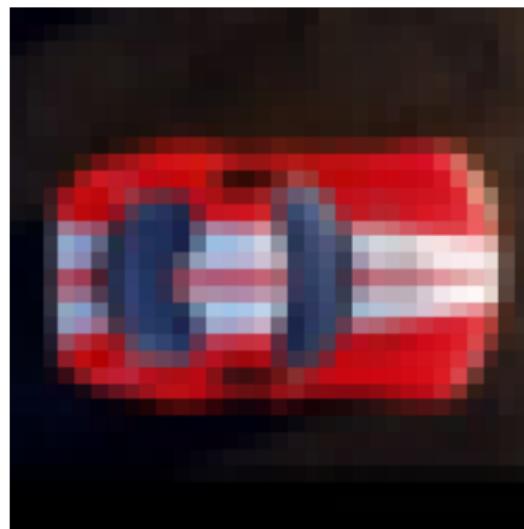
Results

car / truck



Results

truck / car



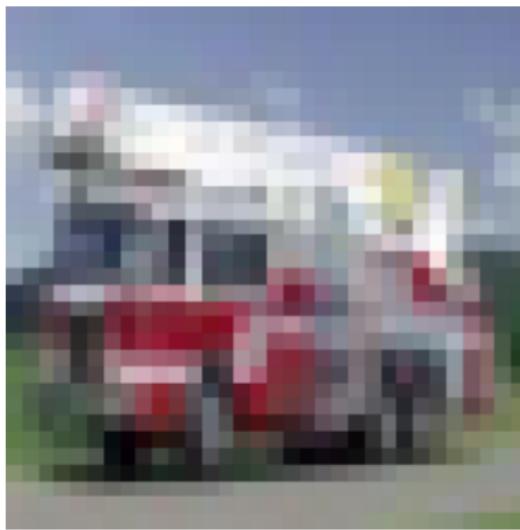
Results

car / car



Results

truck / truck



Results

car / car



Results

car / car

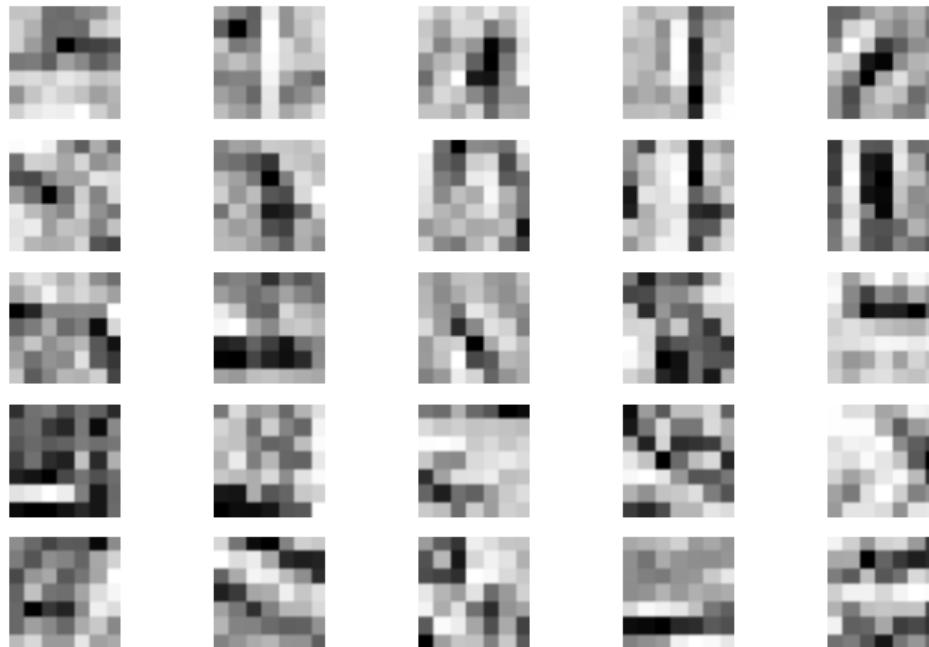


Results

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**