

Lecture 1: Introduction

General organization: People

- Lecturer: Mathieu Salzmann

- Teaching Assistants:

- Sena Kiciroglu

- Andrey Davydov

- Krishna Nakka

- Vudit Vudit

- Benoît Guillard

- Baran Ozaydin

- Student Assistants:

- Julien Vignoud

- Antoine Magron

- Andrea Gilot

General organization: Schedule/Info

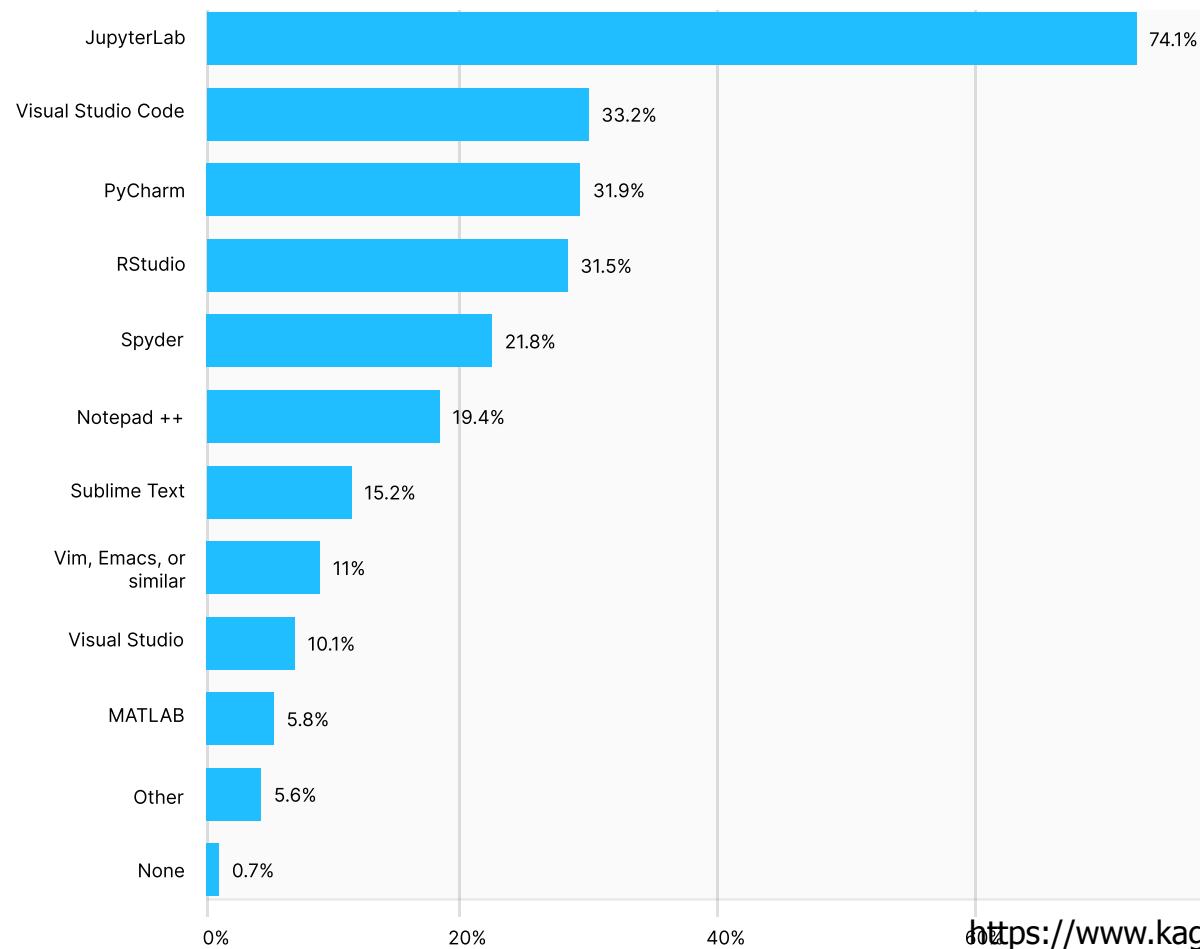
- Lectures: Tuesdays 8:15-10 in CE4
- Exercises: Fridays 8:15-10 in INJ218 and INM202
- Moodle: <https://moodle.epfl.ch/course/view.php?id=16071>
- Main references:
 - C.M. Bishop, [Pattern Recognition and Machine Learning](#), Springer, 2006
 - M. Welling, [A First Encounter with Machine Learning](#), 2011

General organization: Lectures/exercises

- The lectures will be taught in the classroom, streamed live and recorded
- The videos will be posted on SWITCHtube
- The exercise sessions will be managed in presence

General organization: Exercises

- The practical exercises will be done in Python
- According to a 2020 survey among data scientists regarding the most commonly used coding environments:



General organization: Evaluation

- Two graded exercise sessions
 - Tentative dates: 05.11.21 & 17.12.21
 - Each is worth 10% of the final grade
 - They will be done during the regular exercise session time
- Final exam: 80% of the grade
- Past years' exams and graded exercise sessions are available on Moodle

Course content

1. Introduction

- ML Basics

2. Linear Supervised ML

- Linear regression
- Linear models for classification

3. Nonlinear Supervised ML

- K-nearest neighbors
- Feature expansion
- Kernel methods
- Artificial Neural Networks

4. Unsupervised ML

- Linear dimensionality reduction
- K-means clustering

Goals of today's lecture

- Introduce at a high level what Machine Learning is
- Introduce some basic Machine Learning concepts
 - These concepts will keep coming back throughout the semester
- Derive an initial formulation for linear regression

What is Machine Learning?

Learn from experience



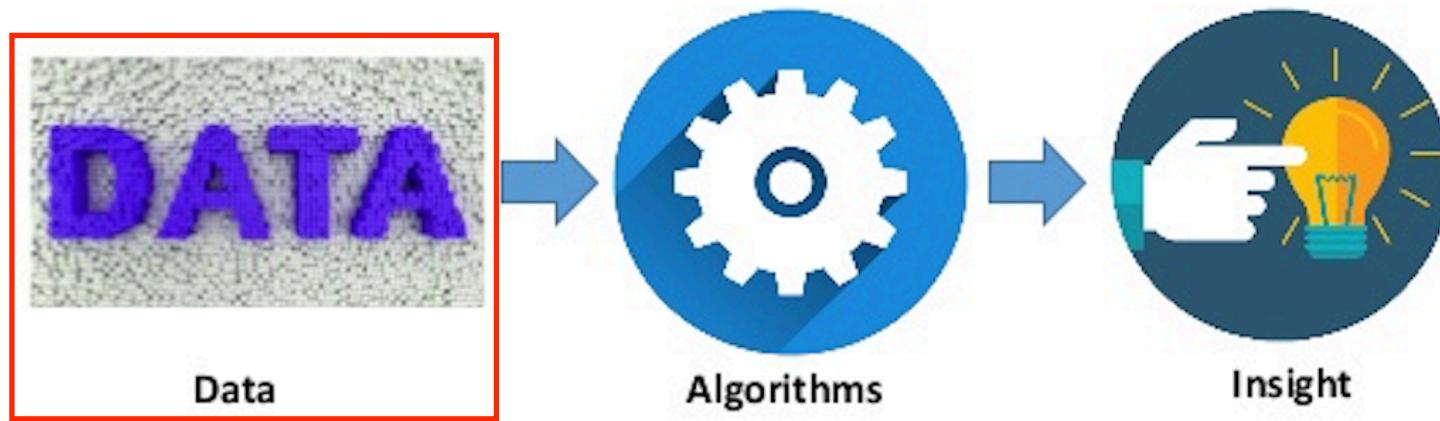
Learn from experience



What is Machine Learning?

- Machine Learning is the science of getting computers to learn and act like humans do, and improve their learning over time in autonomous fashion, by feeding them data and information in the form of observations and real-world interactions.
- Machine learning algorithms seek to provide knowledge to computers through data, observations, and interaction with the world. It is then used to make accurate predictions given new observations.
- Machine Learning is applied statistics.

What is Machine Learning?



What data?

- Attributes: E.g., patient information related to births

	Age at delivery	Weight prior to pregnancy (pounds)	Smoker	Doctor visits during 1 st trimester	Race	Birth Weight (grams)
Patient 1	29	140	Yes	2	Caucasian	2977
Patient 2	32	132	No	4	Caucasian	3080
Patient 3	36	175	No	0	African-Am	3600
*	*	*	*	*	*	*
*	*	*	*	*	*	*
Patient 189	30	95	Yes	2	Asian	3147

Image from Lumen Learning

What data?

- Text: E.g., Movie reviews

5	Column1
6	A very, very, very slow-moving, aimless movie about a distressed, drifting young man.
8	Not sure who was more lost - the flat characters or the audience, nearly half of whom walked out.
10	Attempting artiness with black & white and clever camera angles, the movie disappointed - became e
11	Very little music or anything to speak of.
13	The best scene in the movie was when Gerardo is trying to find a song that keeps running through his
15	The rest of the movie lacks art, charm, meaning... If it's about emptiness, it works I guess because it's
16	Wasted two hours.
18	Saw the movie today and thought it was a good effort, good messages for kids.
20	A bit predictable.
22	Loved the casting of Jimmy Buffet as the science teacher.
23	And those baby owls were adorable.
25	The movie showed a lot of Florida at it's best, made it look very appealing.
26	The Songs Were The Best And The Muppets Were So Hilarious.

Image from Integrated Knowledge Solutions

What data?

- Speech:

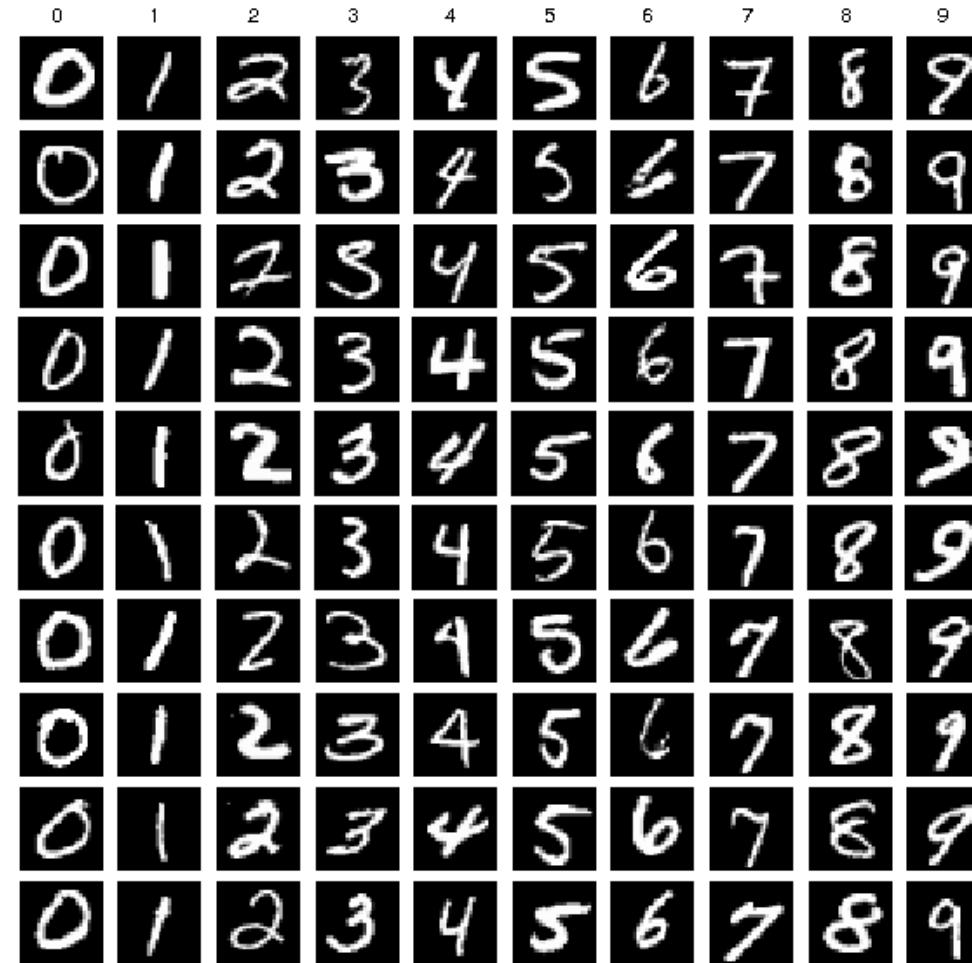
Sample 1: “without the dataset the article is useless”

Sample 2: “the boy looked out at the horizon”

Data from Nicholas Carlini

What data?

- Images: E.g., handwritten digits (MNIST dataset)



What data?

- Mixed: E.g., Collection from the Carnegie Museum of Art in Pittsburgh
 - <https://github.com/cmoa/collection>

Data set vs data sample

- Data sample (or example, or point): An individual observation.
E.g., an image, a list of attributes for one patient

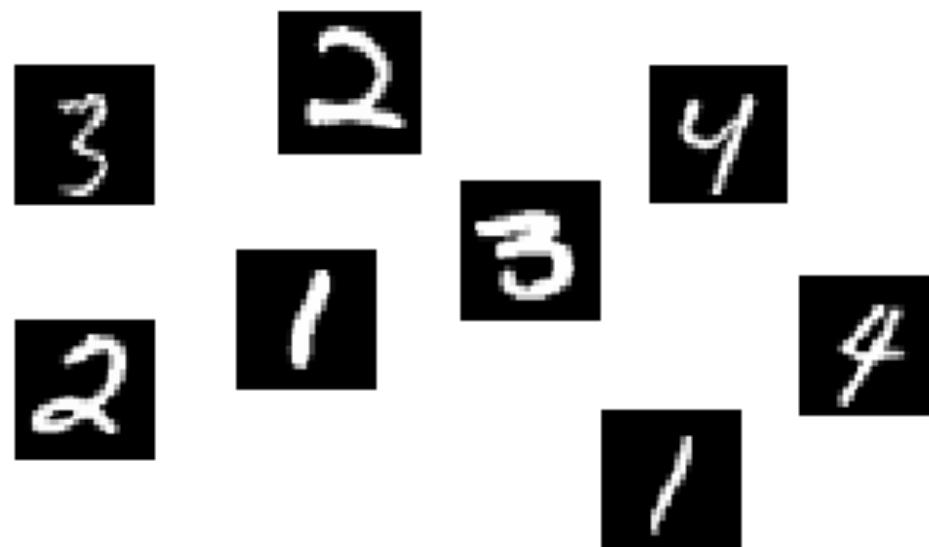


Patient 3	36	175	No	0	African-Am	3600
-----------	----	-----	----	---	------------	------

- Data set: A collection of multiple data samples. E.g., a collection of images, the attribute lists for multiple patients

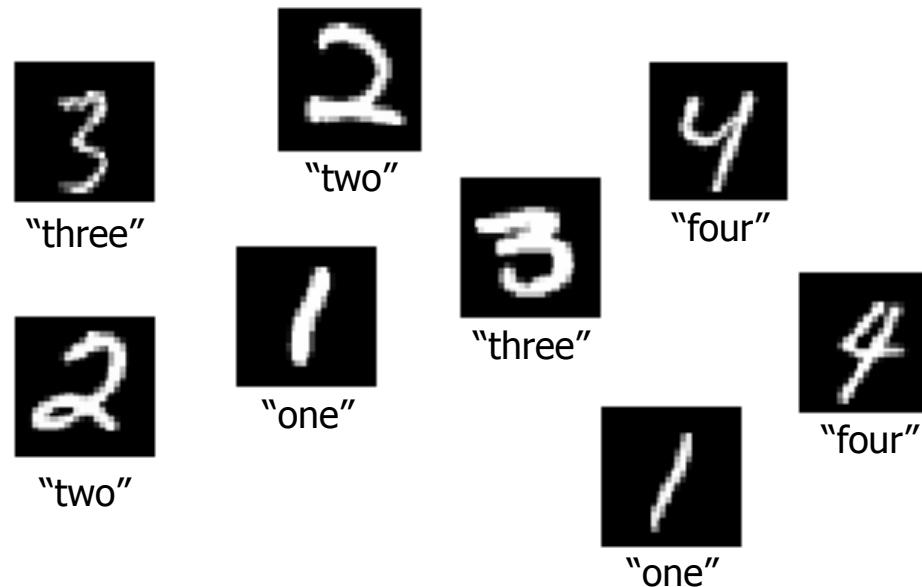
Unsupervised data

- Each sample consists only of an observation, e.g., an image (without information about its content)

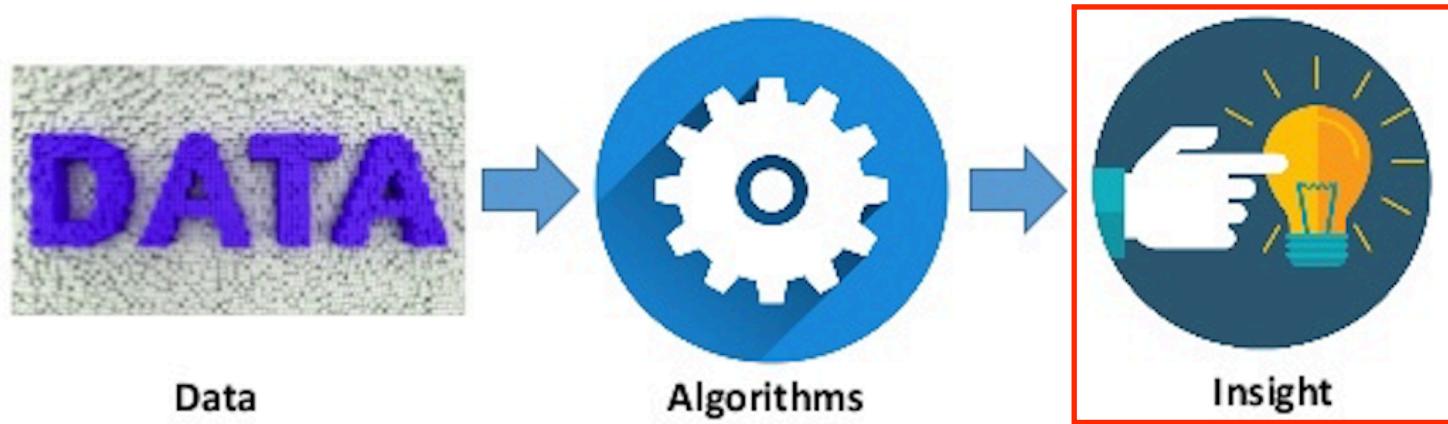


Supervised data

- Each sample comes with additional annotations, e.g., category label

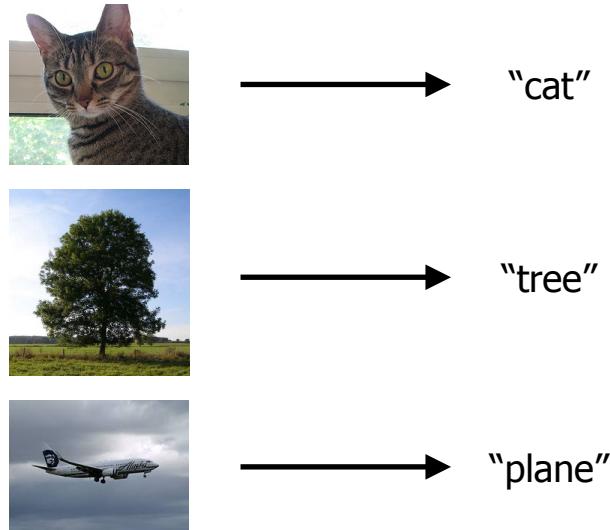


What is Machine Learning?



What insight?

- Precise, concrete prediction. E.g., the category depicted by an image



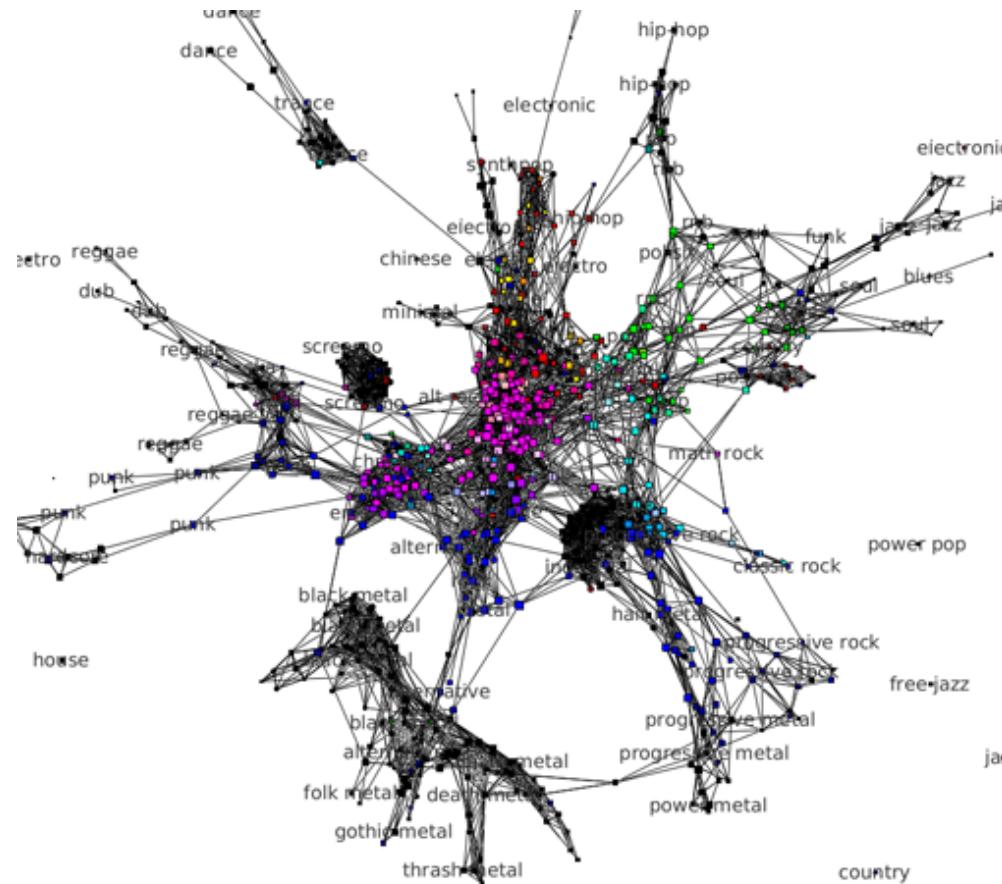
What insight?

- Better understanding of a phenomenon. E.g., identify the mother's characteristics that lead to low birth weight

	Age at delivery	Weight prior to pregnancy (pounds)	Smoker	Doctor visits during 1 st trimester	Race	Birth Weight (grams)
Patient 1	29	140	Yes	2	Caucasian	2977
Patient 2	32	132	No	4	Caucasian	3080
Patient 3	36	175	No	0	African-Am	3600
*	*	*	*	*	*	*
*	*	*	*	*	*	*
Patient 189	30	95	Yes	2	Asian	3147

What insight?

- Better understanding of data. E.g., analyze the influences between different musical genres



What insight?

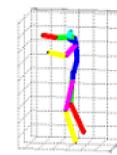
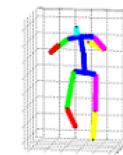
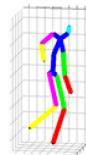
- In this course, we will mostly focus on precise insights. In particular, we will study two classes of ML problems:
 - Regression
 - Classification
- In the unsupervised learning part, we will nonetheless aim to obtain some form of data understanding

Regression vs Classification

- Regression: Predict continuous value(s) for a given sample
 - E.g., predict the birth weight of a baby (single value)

	Age at delivery	Weight prior to pregnancy (pounds)	Smoker	Doctor visits during 1 st trimester	Race	Birth Weight (grams)
Patient 1	29	140	Yes	2	Caucasian	2977
Patient 2	32	132	No	4	Caucasian	3080
Patient 3	36	175	No	0	African-Am	3600
*	*	*	*	*	*	*
*	*	*	*	*	*	*
Patient 189	30	95	Yes	2	Asian	3147

- E.g., human pose estimation: predict the 3D positions of human joints (multiple values)



Regression vs Classification

- Classification: Predict one discrete label for a given sample
 - E.g., binary classification for movies (like vs not like; 1 vs 0)

5	Column1	Column2
6	A very, very, very slow-moving, aimless movie about a distressed, drifting young man.	0
8	Not sure who was more lost - the flat characters or the audience, nearly half of whom walked out.	0
10	Attempting artiness with black & white and clever camera angles, the movie disappointed - became e	0
11	Very little music or anything to speak of.	0
13	The best scene in the movie was when Gerardo is trying to find a song that keeps running through his	1
15	The rest of the movie lacks art, charm, meaning... If it's about emptiness, it works I guess because it's	0
16	Wasted two hours.	0
18	Saw the movie today and thought it was a good effort, good messages for kids.	1

- E.g., multi-class image recognition (cat vs tree vs car; 0 vs 1 vs 2)



Cat



Tree



Car

Regression vs Classification

- In regression, the values typically follow an order
 - E.g., predicting a weight of 3002g instead of 2977g is better than predicting 2500g

	Age at delivery	Weight prior to pregnancy (pounds)	Smoker	Doctor visits during 1 st trimester	Race	Birth Weight (grams)
Patient 1	29	140	Yes	2	Caucasian	2977

- In classification, the categories do not follow an order
 - E.g., predicting “tree” instead of “cat” is just as wrong as predicting “car”
 - Even when categories are represented with numbers (e.g., 1, 2, 3), predicting category 2 instead of 1 is as wrong as predicting category 3



Cat

Example

- Image recognition:
 - <http://scs.ryerson.ca/%7Eaharley/vis/fc/>

Example

- Text analysis:
 - <https://natural-language-understanding-demo.ng.bluemix.net>

Example

- Image captioning:
 - <https://azure.microsoft.com/en-us/services/cognitive-services/computer-vision/>

Example

- Human pose estimation:
 - [https://storage.googleapis.com/tfjs-models/demos/posenet/camera.html?
source=post_page-----](https://storage.googleapis.com/tfjs-models/demos/posenet/camera.html?source=post_page-----)
 - From EPFL: <https://vitademo.epfl.ch>

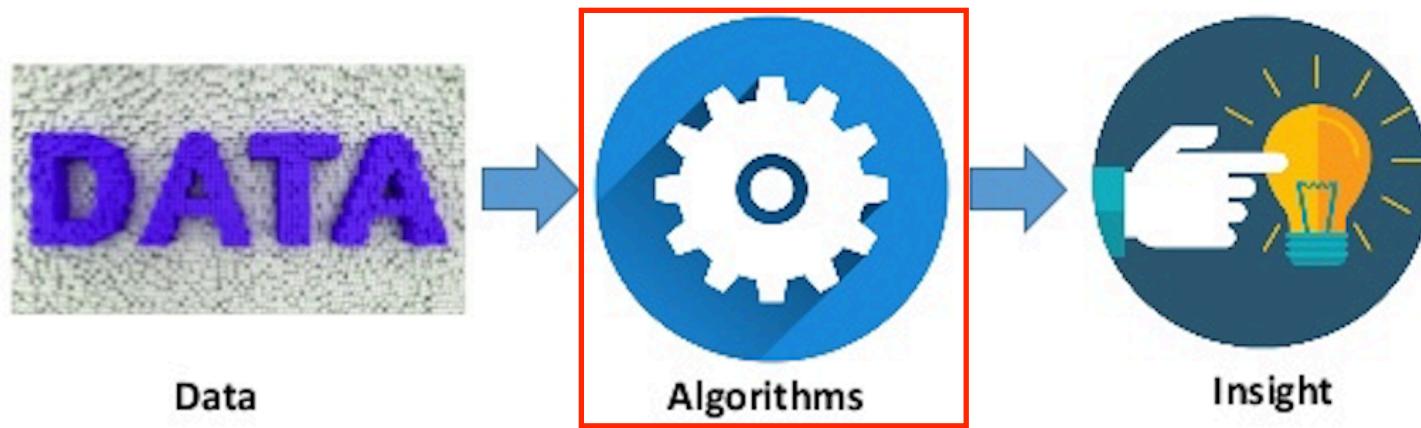
Example

- Image to image translation:
 - <https://affinelayer.com/pixsrv/>

Other fun demos (for you to play)

- AI Pictionary:
 - <https://quickdraw.withgoogle.com>
- Semantris: ML-based word association games
 - <https://research.google.com/semantris/>
- Talk to books:
 - <https://books.google.com/talktobooks/>

What is Machine Learning?



What (classes of) algorithms?

- **Supervised learning**

- Relies on supervised data
- The annotations typically correspond to the desired insight

- **Unsupervised learning**

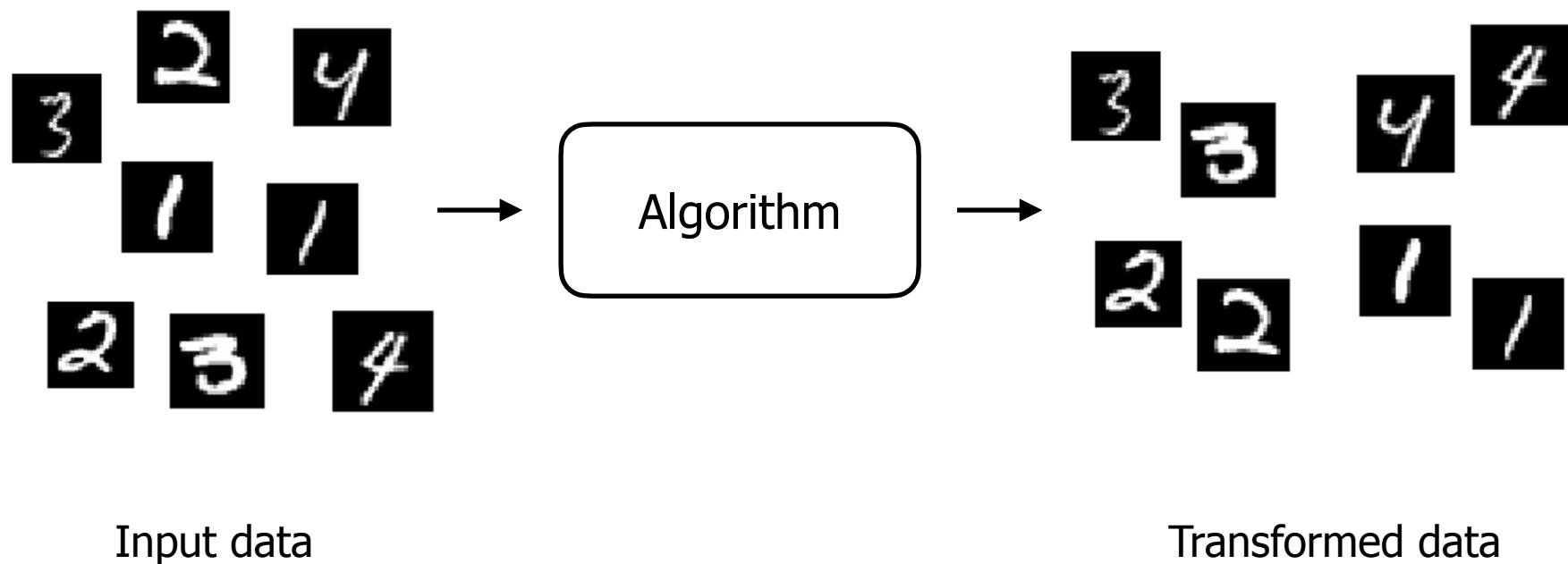
- Relies on unsupervised data
- The goal is rather to analyze the observed data set

- **(Reinforcement learning)**

- Learn to react to the environment
- Not covered in this course

Unsupervised learning

- A single stage: Transform the data for further analysis

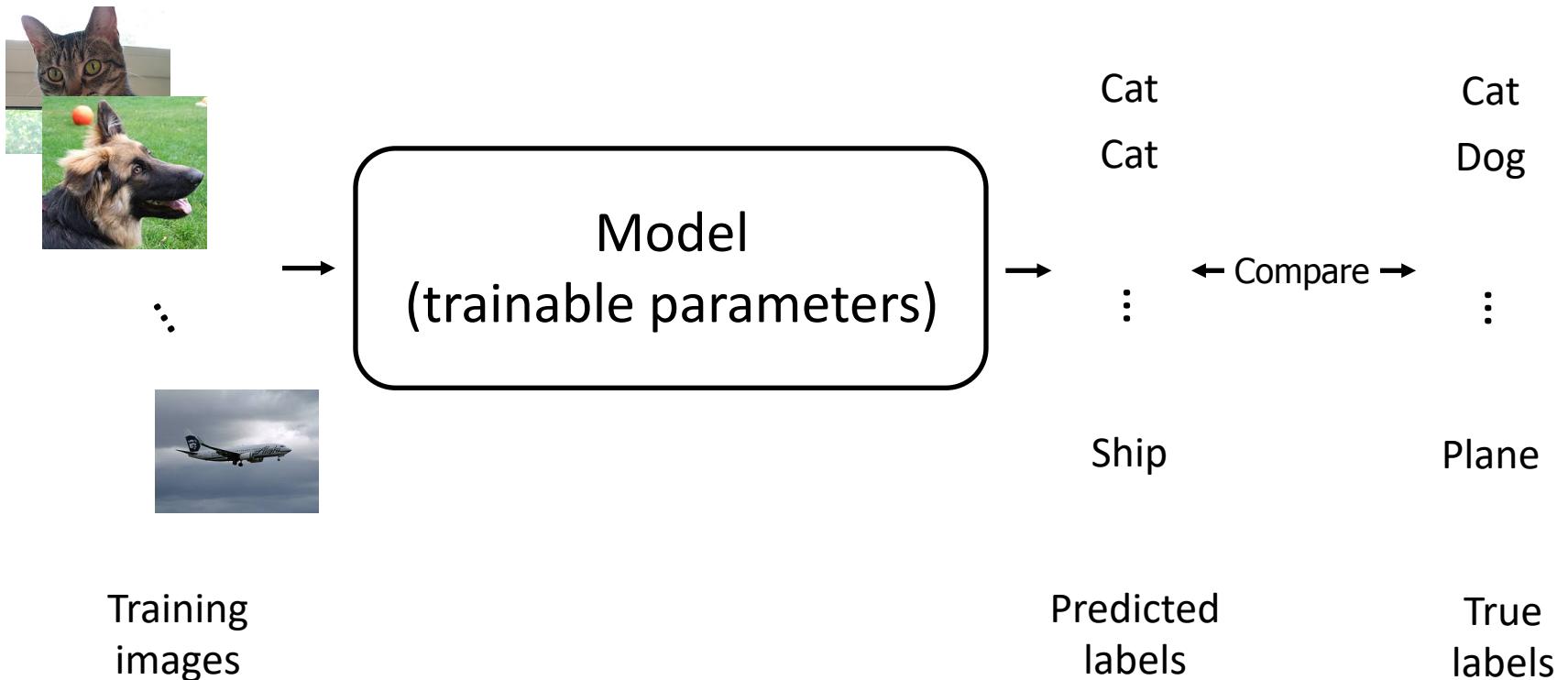


Input data

Transformed data

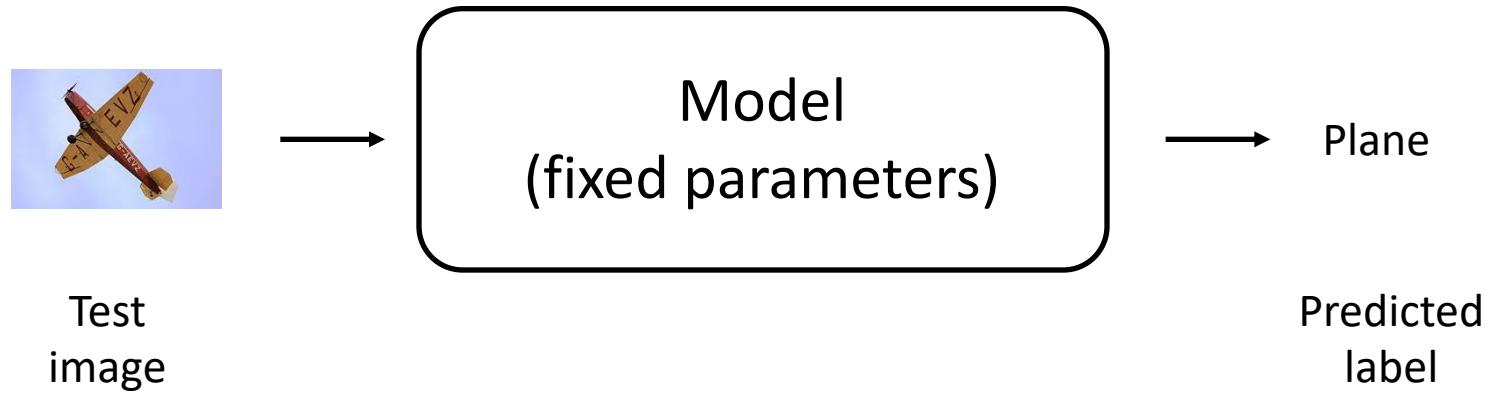
Supervised learning

- Stage 1: Training: Use data with ground-truth labels to optimize model parameters



Supervised learning

- Stage 2: Testing: Predict the output for a new data sample



Supervised learning: Assumption

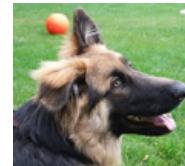
- The annotations (supervisory signal) are the insight that we seek to obtain from the data
 - Example: Category label



Cat



Car



Dog



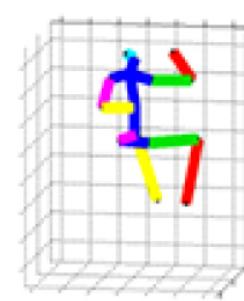
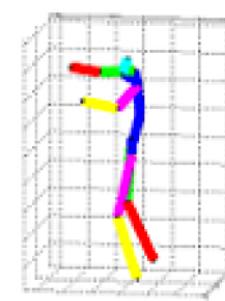
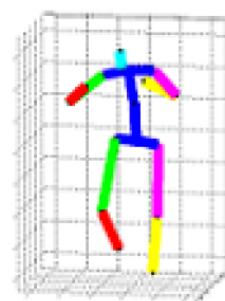
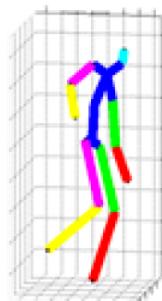
Plane



Chair

Supervised learning: Assumption

- The annotations (supervisory signal) are the insight that we seek to obtain from the data
 - Example: Human pose



Training set vs test set

- **The training set and the test set should always be completely separate!**
- Never use the test annotations during training
 - Using the test observations (inputs) is occasionally possible and referred to as transductive learning (we will not do it in this course)



Test
image



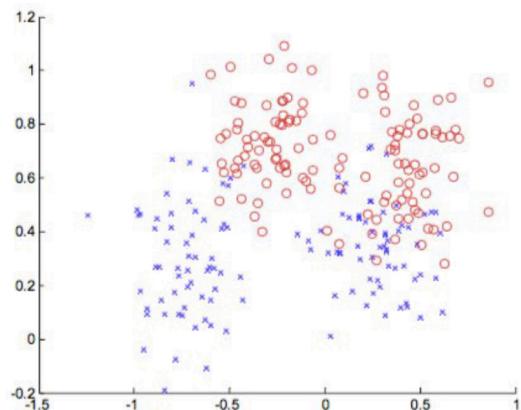
⋮



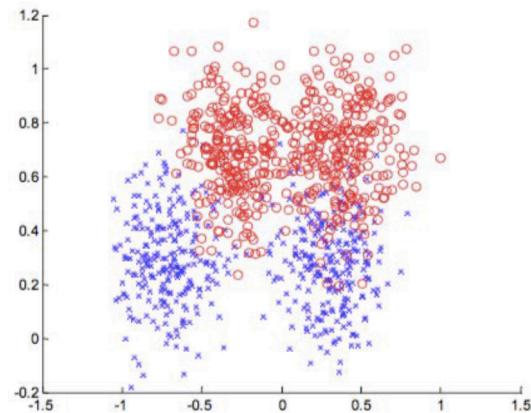
Training
images

Training set vs test set

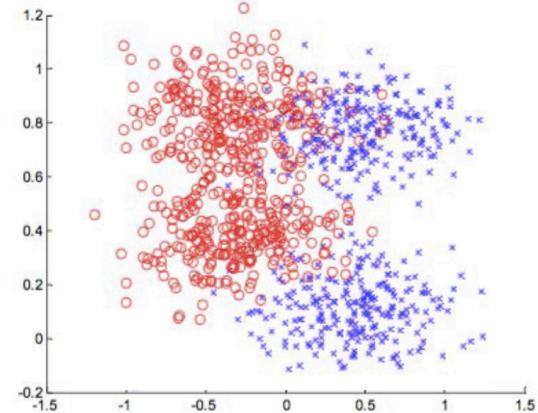
- Assumption: Training and test samples are drawn from the same statistical distribution
 - E.g., synthetic data: 2D inputs with 2 classes (colors)



Training data



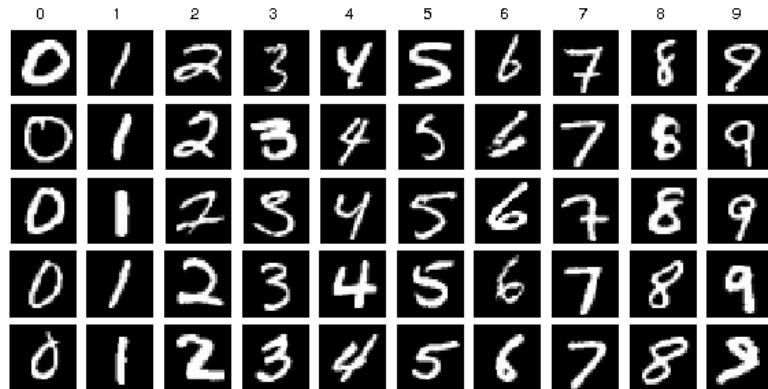
Test data from
same distribution



Test data from
different distribution

Training set vs test set

- Training and test samples are drawn from the same statistical distribution
 - E.g., digit recognition: A model trained on MNIST will work poorly on Street View House Numbers



Training data



Test data

Exercises

- Given a dataset where each sample is represented with a list of meteorological measurements, the goal is to predict the burned area of the corresponding forest fire
 - If you are given the ground-truth burned areas for a set of training samples, what general class of algorithms would you use to solve this problem? *supervised learning*
 - What type of Machine Learning problem is this? *regression*

Your first machine learning model

- Let's start with a simple supervised learning model:

The Linear Model

- We will spend a few weeks on this
 - This will allow us to also cover general ML topics

Notation

We denote the i^{th} data sample (input) in the collection of N samples as

$$\mathbf{x}_i \in \mathbb{R}^D,$$

a vector of dimension D

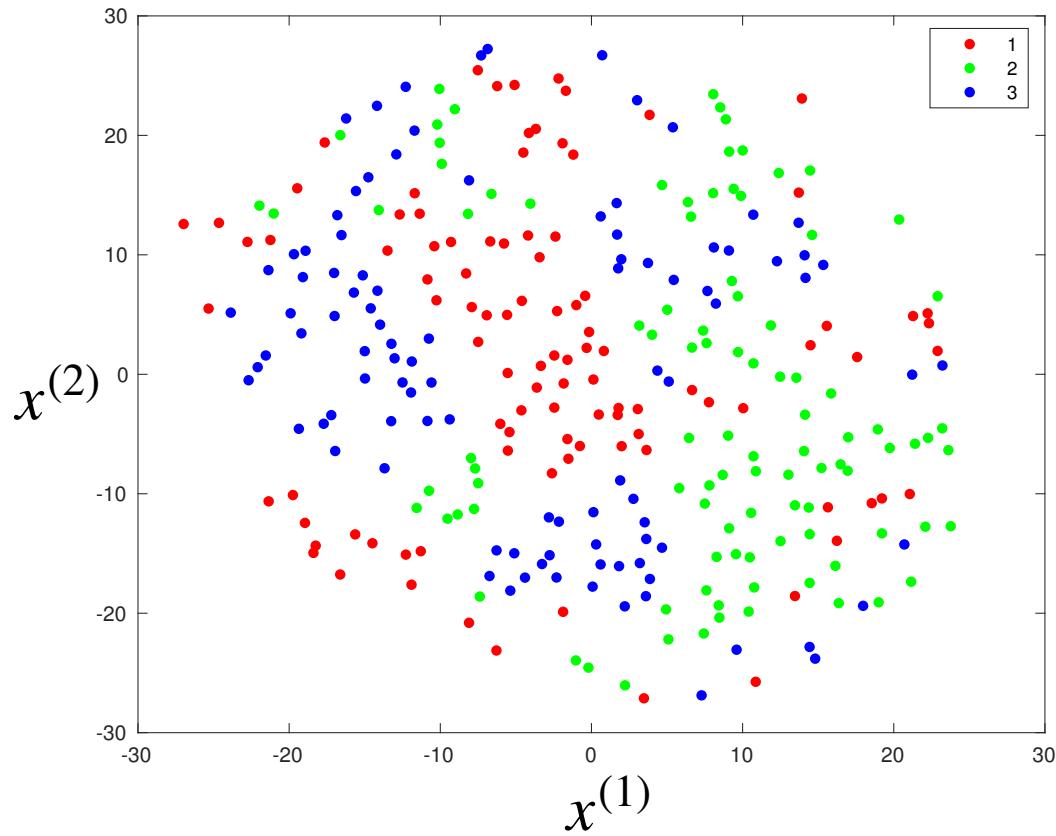
We denote the i^{th} label (output) in the collection of N samples as \mathbf{y}_i

For classification, \mathbf{y}_i is represents a single discrete value

For regression, \mathbf{y}_i can be a single continuous value, or a vector of dimension C

Notation made concrete

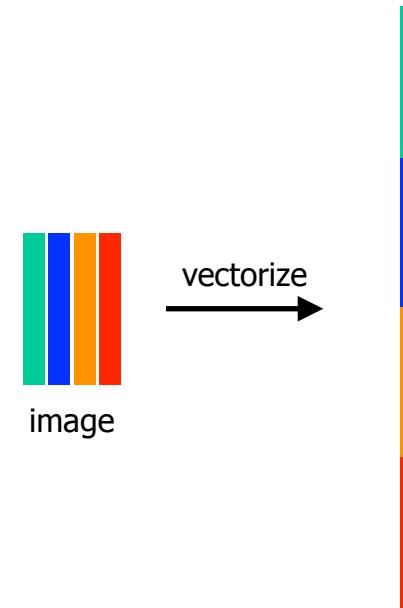
- In the following toy example, \mathbf{x}_i is one 2D point, and thus $D = 2$
- y_i is a discrete value indicating the class (color), i.e., 1, 2, or 3



Notation made concrete

- In the digit recognition example, \mathbf{x}_i is a grayscale image. If it has height H and width W , then $D = H \cdot W$

$\mathbf{x}_i = \text{vectorize}(\mathbf{2})$



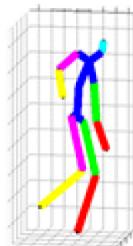
- \mathbf{y}_i is a single discrete value indicating the digit, e.g., $\mathbf{y}_i = 2$

Notation made concrete

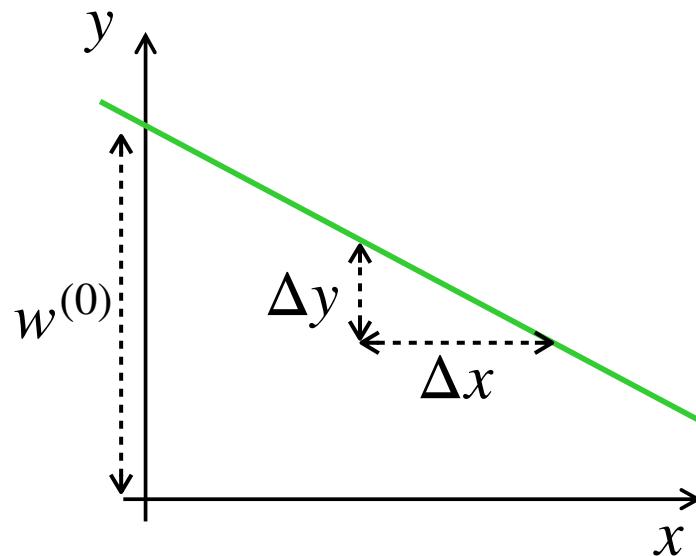
- In the human pose example, \mathbf{x}_i is a color image. If it has height H and width W , then $D = 3 \cdot H \cdot W$

$$\mathbf{x}_i = \text{vectorize}(\quad)$$


- Then, \mathbf{y}_i is a human pose. If a human pose is defined as a skeleton with 12 joints (wrists, elbows,...), and each joint is a 3D point, then $C = 3 \cdot 12 = 36$

$$\mathbf{y}_i =$$


A simple parametric model: The line

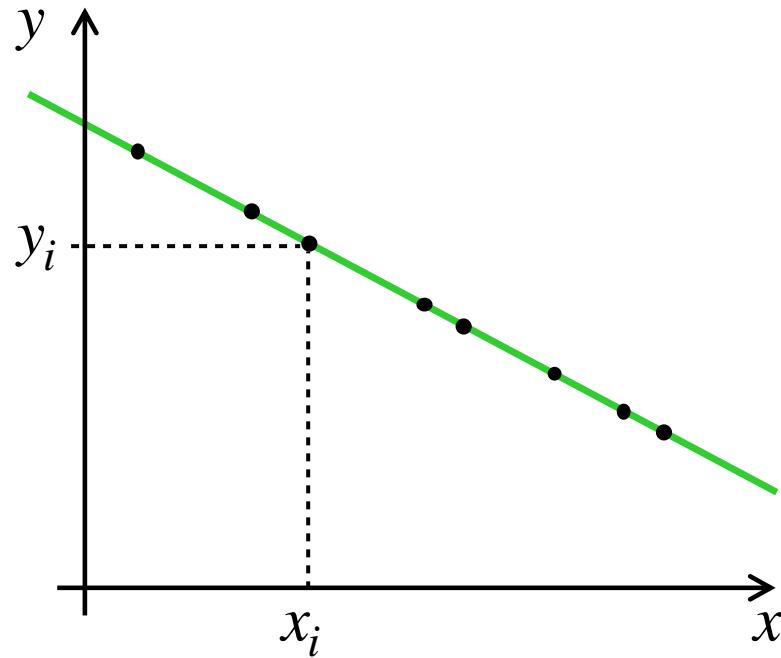


- Defined by 2 parameters
 - The **y**-intercept $w^{(0)}$
 - The **slope** $w^{(1)} = \frac{\Delta y}{\Delta x}$
- Mathematically, **a line is expressed as**

$$y = w^{(1)}x + w^{(0)}$$

Line fitting

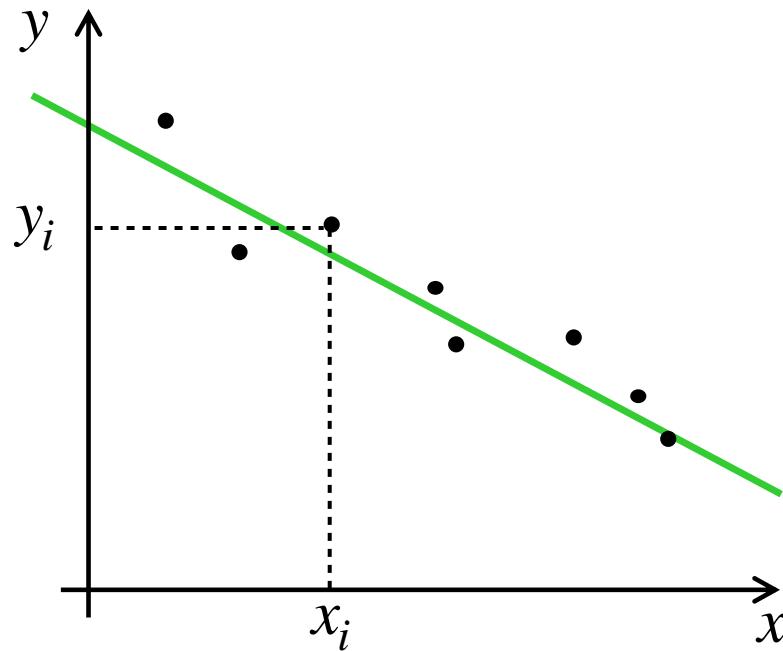
- Given N pairs $\{(x_i, y_i)\}$, find the line that passes through these observations



- This ideal case never occurs in practice

Line fitting with noise

- Given N pairs $\{(x_i, y_i)\}$ of noisy measurements, find the line that best fits these observations

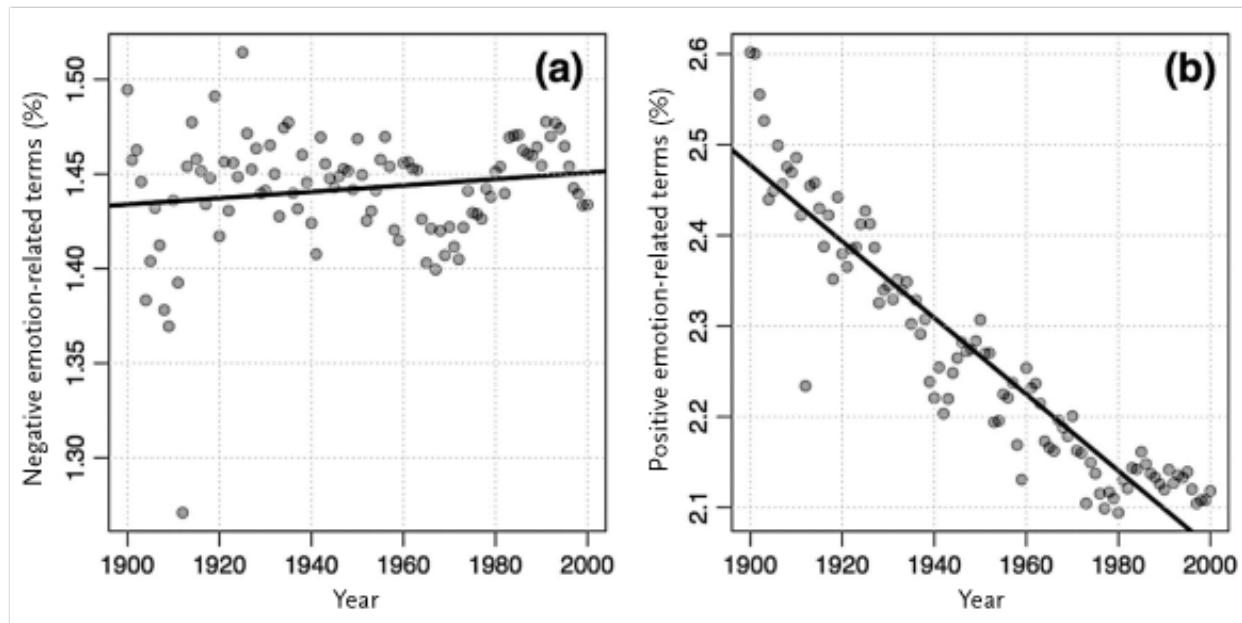


- This process is called *linear regression*

first ML model

1D linear regression: Example

- Discover trends:
 - Example: Proportion of negative and positive emotions in anglophone fiction (Morin & Acerbi, 2016. Figure from Moretti & Sobchuk, 2019)



1D Linear regression: Training

- In essence, fitting a line consists of finding the best line parameters $w^{(0)*}$ and $w^{(1)*}$ for some given data
- This corresponds to the training stage:
 - Given N training pairs $\{(x_i, y_i)\}$, we aim to find $(w^{(0)*}, w^{(1)*})$, such that the predictions of the model

$$\hat{y}_i = w^{(1)*}x_i + w^{(0)*}$$

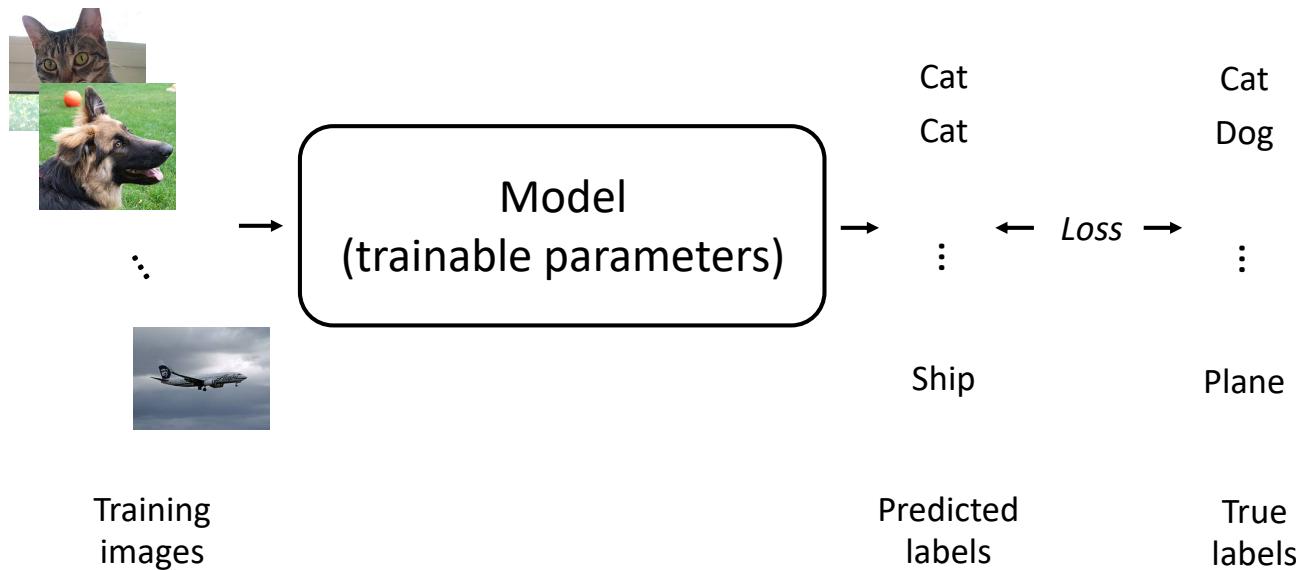
- are close to the true values y_i
- We then need to define a measure of closeness between y_i and \hat{y}_i

Interlude

Loss Function and Empirical Risk

Loss function

- The loss function $\ell(\hat{y}_i, y_i)$ computes an error value between the prediction and the true value
 - This is a general ML concept, not only for linear regression
 - We will see different loss functions in the upcoming lectures



Empirical risk

- Given N training samples $\{(\mathbf{x}_i, y_i)\}$, the empirical risk is defined as

$$R(\{\mathbf{x}_i\}, \{y_i\}, \mathbf{w}) = \frac{1}{N} \sum_{i=1}^N \ell(\hat{y}_i, y_i)$$

- where $\{\mathbf{x}_i\}$ and $\{y_i\}$ are the sets of training inputs and labels, respectively
- During training, our goal is to find the parameters \mathbf{w} (e.g., $w^{(0)}$ and $w^{(1)}$) that minimize the empirical risk
 - Note that the risk depends on \mathbf{w} via \hat{y}_i

Minimizing the risk

- This is expressed as the optimization problem

$$\min_{\mathbf{w}} \frac{1}{N} \sum_{i=1}^N \ell(\hat{y}_i, y_i)$$

- We can also write that the best parameters are the solution

$$\mathbf{w}^* = \operatorname{argmin}_{\mathbf{w}} \frac{1}{N} \sum_{i=1}^N \ell(\hat{y}_i, y_i)$$

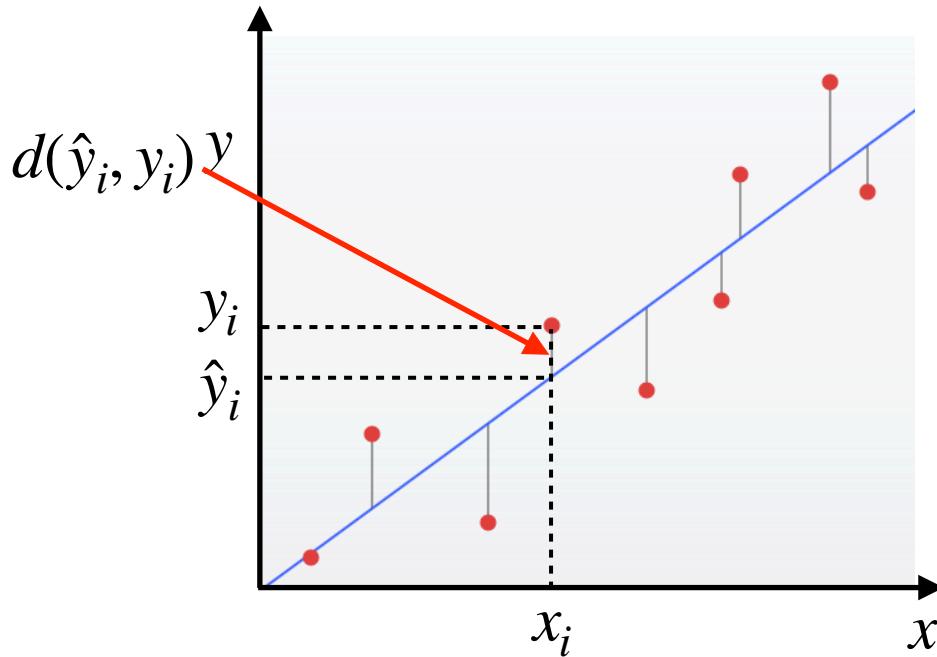
End of the interlude

Back to Linear Regression

1D Linear regression: Training

- A natural measure of closeness is the Euclidean distance

$$d(\hat{y}_i, y_i) = \sqrt{(\hat{y}_i - y_i)^2}$$



- The difference between \hat{y}_i and y_i is often referred to as the *residual*

1D Linear regression: Training

- In practice, one often prefers using the squared Euclidean distance

$$d^2(\hat{y}_i, y_i) = (\hat{y}_i - y_i)^2$$

- Training can then be expressed as the *least-squares* minimization problem

$$\min_{w^{(0)}, w^{(1)}} \frac{1}{N} \sum_{i=1}^N d^2(\hat{y}_i, y_i)$$

where \hat{y}_i depends on $w^{(0)}$ and $w^{(1)}$

- We will see how to solve this problem next week

1D linear regression: Demo

- http://digitalfirst.bfwpub.com/stats_applet/stats_applet_5_correg.html

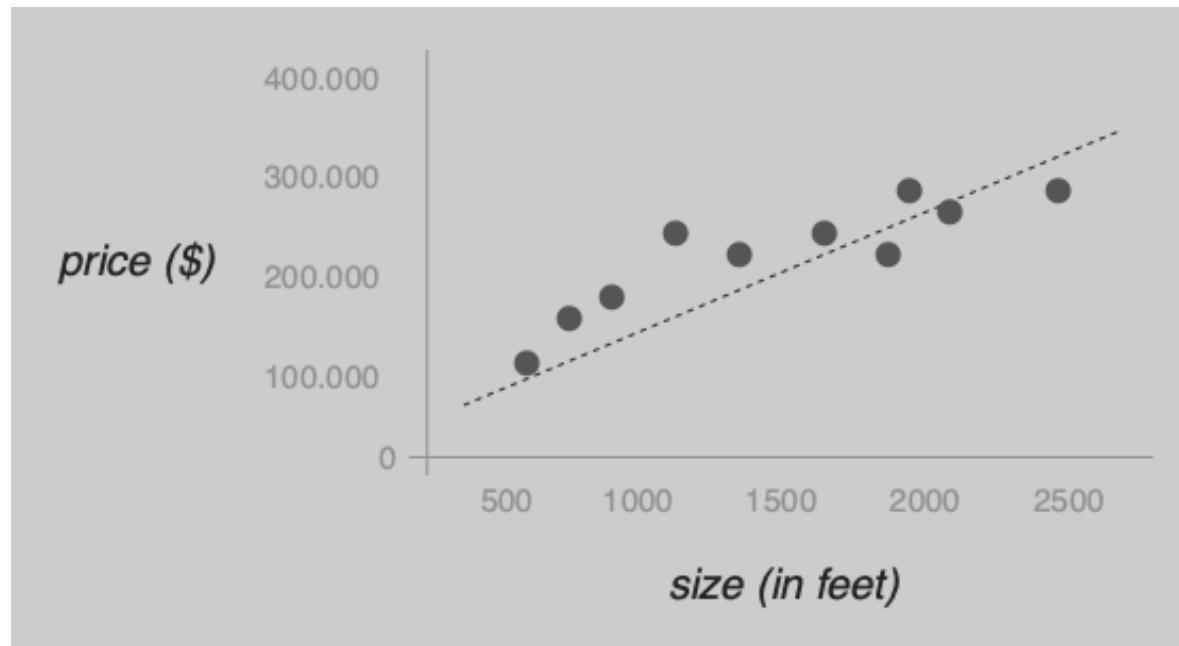
1D Linear regression: Prediction

- Once you found the best line for the given N observations, you can use it to predict a y value for a new x
- Let $w^{(0)*}$ and $w^{(1)*}$ be the best line parameters given the observations
- Then, for any value x , you can predict an estimate of the corresponding y as

$$\hat{y} = w^{(1)*}x + w^{(0)*}$$

1D linear regression: Example

- Predict quantities
 - Predict the price of a house based on its size (example from <https://www.internalpointers.com/post/linear-regression-one-variable>)



- With temporal trends, one can predict what will happen in the future

Model evaluation

- Once an ML model is trained, one would typically understand how well it performs on unseen test data
 - At this stage, the parameters of the model are fixed
 - Recall that the training and testing data must be separated!
- During this evaluation, one compares the predictions of the model with the true annotations of the test data
 - In contrast to the training stage, the model parameters are *not* updated
 - The evaluation metric may directly be the loss function, but may also differ from it

Evaluation metrics for regression

- Mean Squared Error (MSE)
 - Same as the loss function but for N_t test samples

$$MSE = \frac{1}{N_t} \sum_{i=1}^{N_t} (\hat{y}_i - y_i)^2$$

where \hat{y}_i is the prediction for test sample i and y_i the corresponding ground-truth value

- Root Mean Squared Error (RMSE)
 - Square-root of the MSE

Evaluation metrics for regression

- Mean Absolute Error (MAE)

$$MAE = \frac{1}{N_t} \sum_{i=1}^{N_t} |\hat{y}_i - y_i|$$

- Mean Absolute Percentage Error (MAPE)

$$MAPE = \frac{1}{N_t} \sum_{i=1}^{N_t} \left| \frac{\hat{y}_i - y_i}{y_i} \right|$$

Taking a percentage w.r.t. the true value might be easier to interpret

Exercises

- Given the following dataset for birth weight prediction:

	Age at delivery	Weight prior to pregnancy (pounds)	Smoker	Doctor visits during 1 st trimester	Race	Birth Weight (grams)
Patient 1	29	140	Yes	2	Caucasian	2977
Patient 2	32	132	No	4	Caucasian	3080
Patient 3	36	175	No	0	African-Am	3600
*	*	*	*	*	*	*
*	*	*	*	*	*	*
Patient 189	30	95	Yes	2	Asian	3147

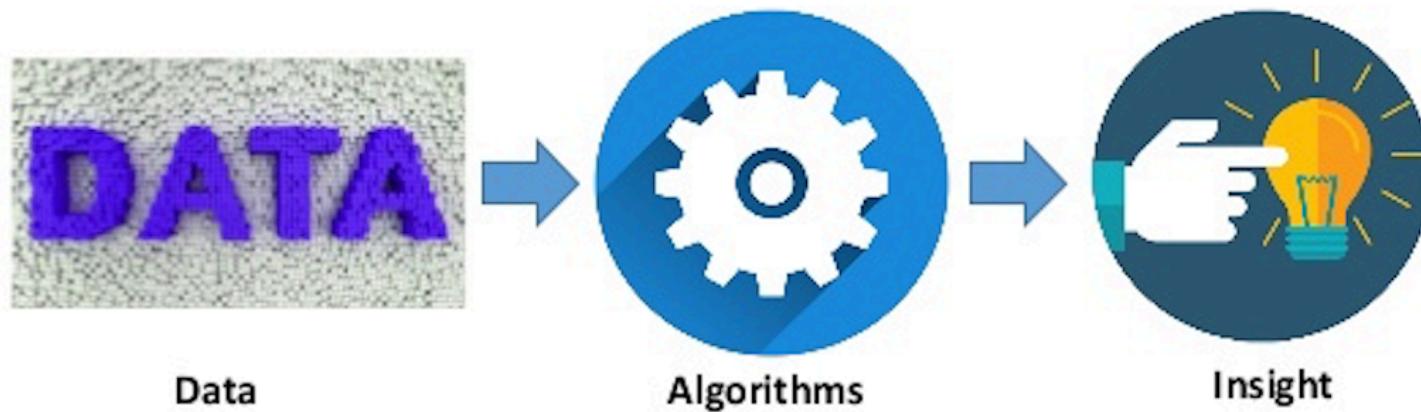
- How many samples (N) can you assume this dataset to contain? 189
- What is the dimensionality (D) of the input to the ML model? 5 
- What is the dimensionality (C) of the output of the ML model? 1 •

Survey

- Please fill in the survey on the Moodle page to comment on the pace of the lecture

Lecture 2: Linear Regression

Recap: What is Machine Learning?



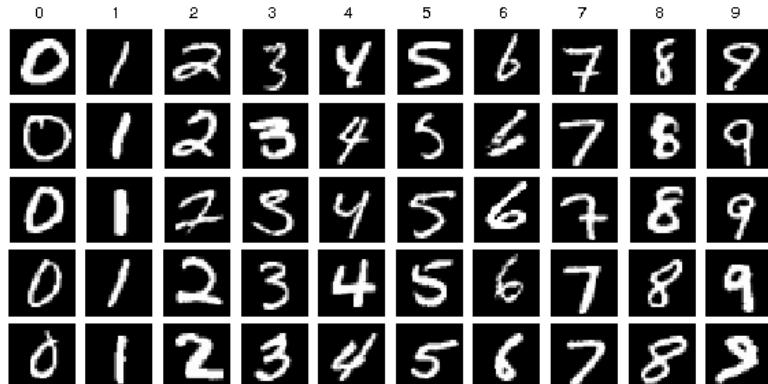
Recap: What data?

- Attributes:

	Age at delivery	Weight prior to pregnancy (pounds)	Smoker	Doctor visits during 1 st trimester	Race	Birth Weight (grams)
Patient 1	29	140	Yes	2	Caucasian	2977
Patient 2	32	132	No	4	Caucasian	3080
Patient 3	36	175	No	0	African-Am	3600
*	*	*	*	*	*	*
*	*	*	*	*	*	*
Patient 189	30	95	Yes	2	Asian	3147

Image from Lumen Learning

- Images



- Text:

5	Column1	Column2
6	A very, very slow-moving, aimless movie about a distressed, drifting young man.	0
8	Not sure who was more lost - the flat characters or the audience, nearly half of whom walked out.	0
10	Attempting artiness with black & white and clever camera angles, the movie disappointed - became e	0
11	Very little music or anything to speak of.	0
13	The best scene in the movie was when Gerardo is trying to find a song that keeps running through his	1
15	The rest of the movie lacks art, charm, meaning... If it's about emptiness, it works I guess because it's	0
16	Wasted two hours.	0
18	Saw the movie today and thought it was a good effort, good messages for kids.	1
20	A bit predictable.	0
22	Loved the casting of Jimmy Buffet as the science teacher.	1
23	And those baby owls were adorable.	1
25	The movie showed a lot of Florida at its best, made it look very appealing.	1
26	The Songs Were The Best And The Muppets Were So Hilarious.	1

Image from Integrated Knowledge Solutions

- Speech/sound:

“without the dataset
the article is useless”

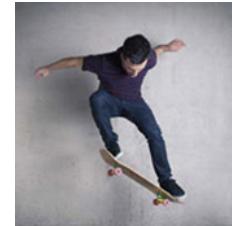
Recap: What insight?

Category



→ “cat”

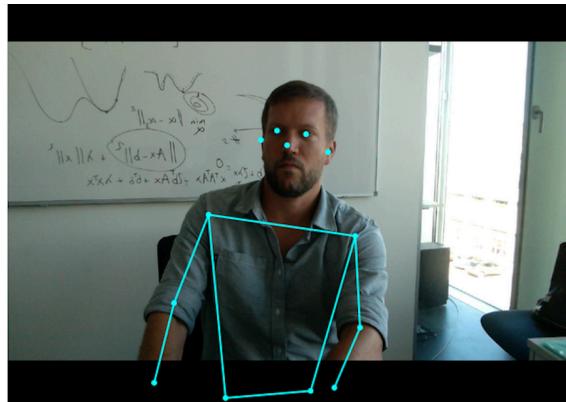
Description



→

“A young man riding
a skateboard”

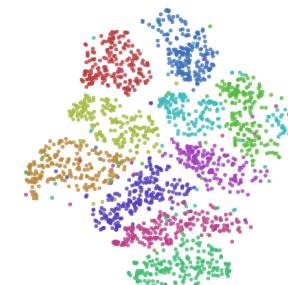
Vector-valued quantity



Different representation

0	1	2	3	4	5	6	7	8	9
0	1	2	3	4	5	6	7	8	9
0	1	2	3	4	5	6	7	8	9
0	1	2	3	4	5	6	7	8	9
0	1	2	3	4	5	6	7	8	9
0	1	2	3	4	5	6	7	8	9
0	1	2	3	4	5	6	7	8	9
0	1	2	3	4	5	6	7	8	9
0	1	2	3	4	5	6	7	8	9
0	1	2	3	4	5	6	7	8	9

→



Recap: What (classes of) algorithms?

- Supervised learning
 - Relies on supervised data
 - The annotations typically correspond to the desired insight
- Unsupervised learning
 - Relies on unsupervised data
 - The goal is rather to analyze the observed data set
- (Reinforcement learning)
 - Learn to react to the environment
 - Not covered in this course

Exercises: Solutions

- Given a dataset where each sample is represented with a list of meteorological measurements, the goal is to predict the burned area of the corresponding forest fire
 - If you are given the ground-truth burned areas for a set of training samples, what general class of algorithms would you use to solve this problem?
 - Solution: Supervised learning
 - What type of Machine Learning problem is this?
 - Solution: A regression problem (we are predicting a continuous quantity, not a category label)

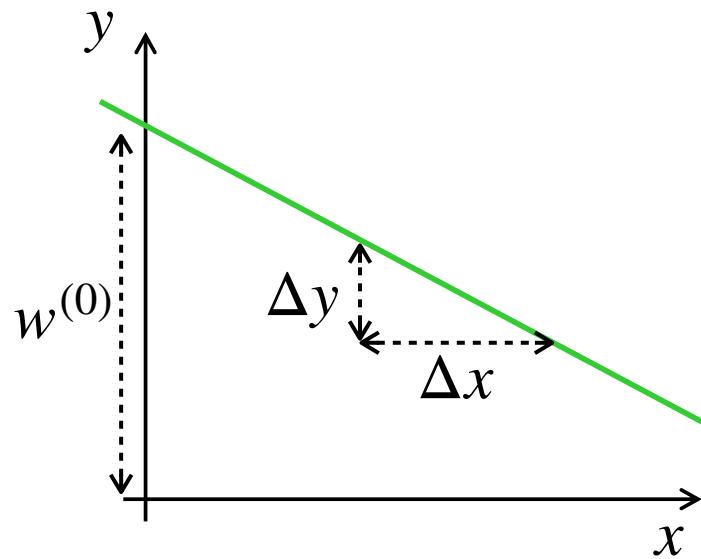
Exercises: Solutions

- Given the following dataset for birth weight prediction:

	Age at delivery	Weight prior to pregnancy (pounds)	Smoker	Doctor visits during 1 st trimester	Race	Birth Weight (grams)
Patient 1	29	140	Yes	2	Caucasian	2977
Patient 2	32	132	No	4	Caucasian	3080
Patient 3	36	175	No	0	African-Am	3600
*	*	*	*	*	*	*
*	*	*	*	*	*	*
Patient 189	30	95	Yes	2	Asian	3147

- How many samples (N) can you assume this dataset to contain?
- What is the dimensionality (D) of the input to the ML model?
- What is the dimensionality (C) of the output of the ML model?
 - Solutions: $N = 189$, $D = 5$, $C = 1$

Recap: A simple parametric model: The line

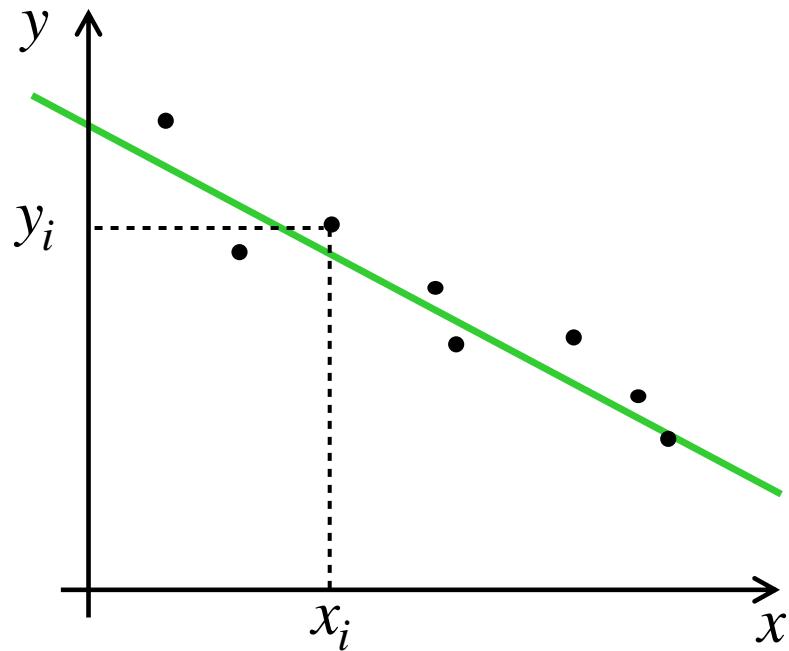


- Defined by 2 parameters
 - The y -intercept $w^{(0)}$
 - The slope $w^{(1)} = \frac{\Delta y}{\Delta x}$
- Mathematically, a line is expressed as

$$y = w^{(1)}x + w^{(0)}$$

Recap: Line fitting with noise

- Given N pairs $\{(x_i, y_i)\}$ of noisy measurements, find the line that best fits these observations



- This process is called *linear regression*

Recap: 1D Linear regression: Training

- In essence, fitting a line consists of finding the best line parameters $w^{(0)*}$ and $w^{(1)*}$ for some given data
- This corresponds to the training stage:
 - Given N training pairs $\{(x_i, y_i)\}$, we aim to find $(w^{(0)*}, w^{(1)*})$, such that the predictions of the model

$$\hat{y}_i = w^{(1)*}x_i + w^{(0)*}$$

are close to the true values y_i

- We then need to define a measure of closeness between y_i and \hat{y}_i

Recap: 1D Linear regression: Training

- In practice, one typically uses the squared Euclidean distance

$$d^2(\hat{y}_i, y_i) = (\hat{y}_i - y_i)^2$$

- Training can then be expressed as the *least-squares* minimization problem

$$\min_{w^{(0)}, w^{(1)}} \frac{1}{N} \sum_{i=1}^N d^2(\hat{y}_i, y_i)$$

where \hat{y}_i depends on $w^{(0)}$ and $w^{(1)}$

Goals of today's lecture

- Review basic derivatives and gradients concepts
- Derive the solution for linear regression, first with 1D inputs, then with multi-dimensional inputs, finally with multi-dimensional outputs
- Interpret a trained linear model
- Introduce the initial steps to go from regression to classification

Minimizing the risk

- The linear regression training problem is a special case of empirical risk minimization
- Minimizing the risk involves computing its derivatives w.r.t. the model parameters
- Let us then review the notion of derivatives/gradient
 - This part is not specific to ML, but it will be useful in the upcoming lectures

Interlude

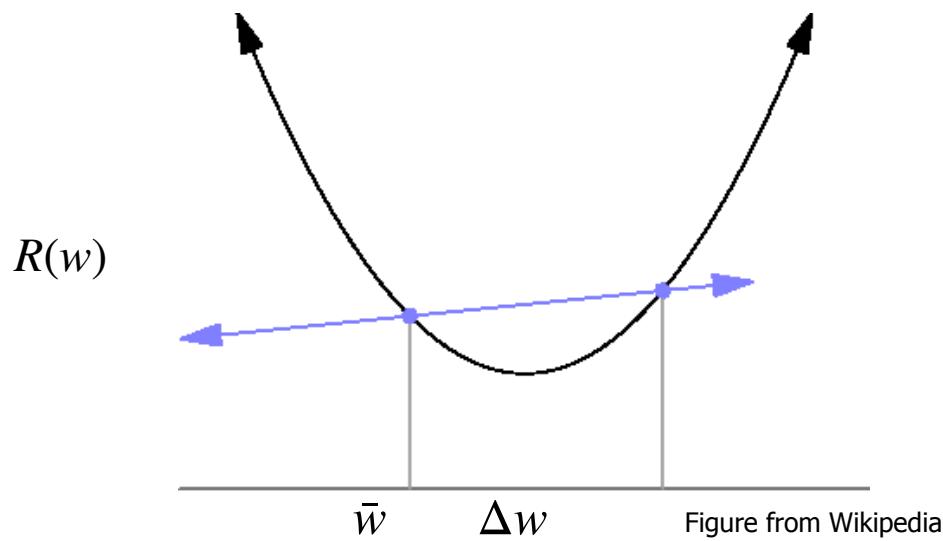
Derivatives and Gradients

Derivative of a 1-variable function

- The derivative of a function $R(w)$ of a single variable w is the rate at which R changes as w changes
- It is measured for an infinitesimal change in w , starting from a point \bar{w} , and written as

$$R'(\bar{w}) = \frac{dR}{dw} = \lim_{\Delta w \rightarrow 0} \frac{R(\bar{w} + \Delta w) - R(\bar{w})}{\Delta w}$$

Derivative of a 1-variable function



- As Δw decreases, the line joining \bar{w} and $\bar{w} + \Delta w$ becomes the tangent to the function
- The derivative is the slope of this tangent

Derivative of a 1-variable function

- Example: More complicated function

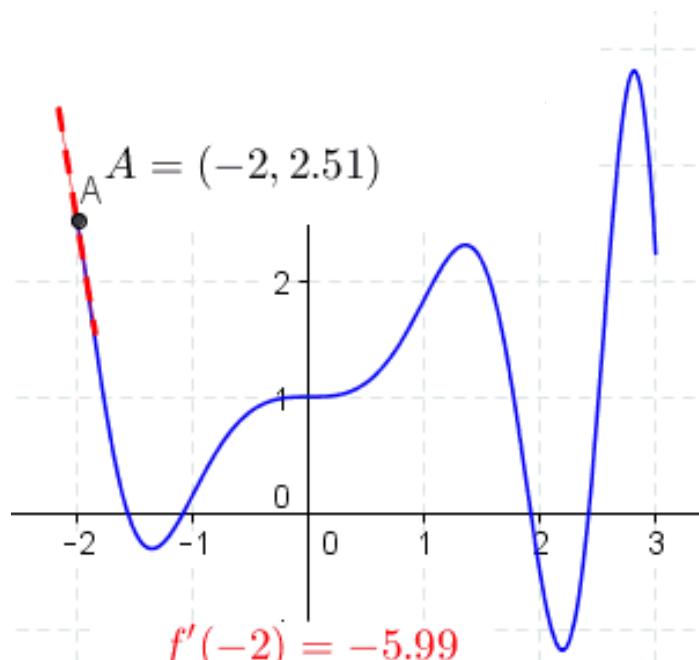
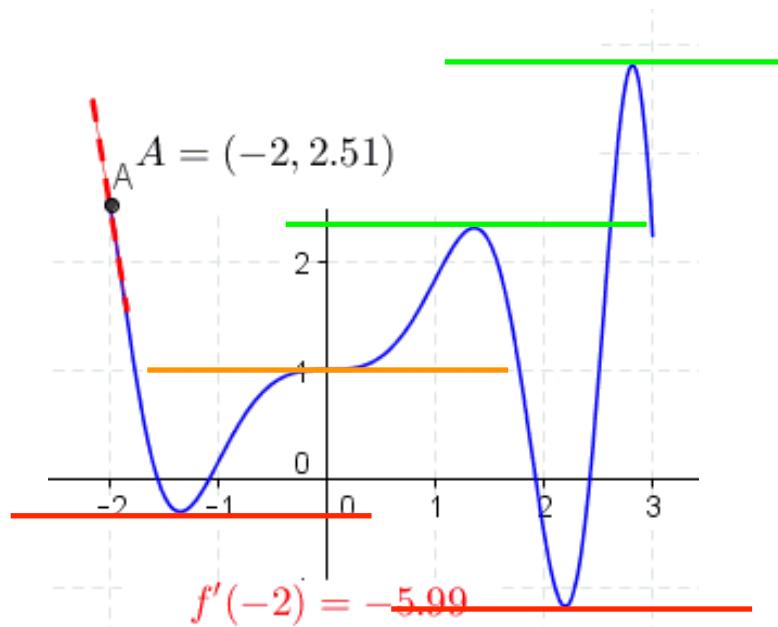


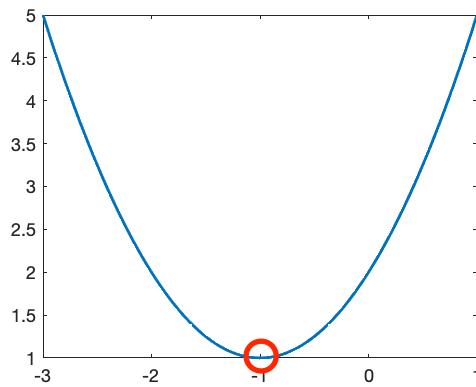
Figure from Wikipedia

Properties of derivatives

- The derivative vanishes at the stationary points:
 - Minima
 - Maxima
 - Saddle points



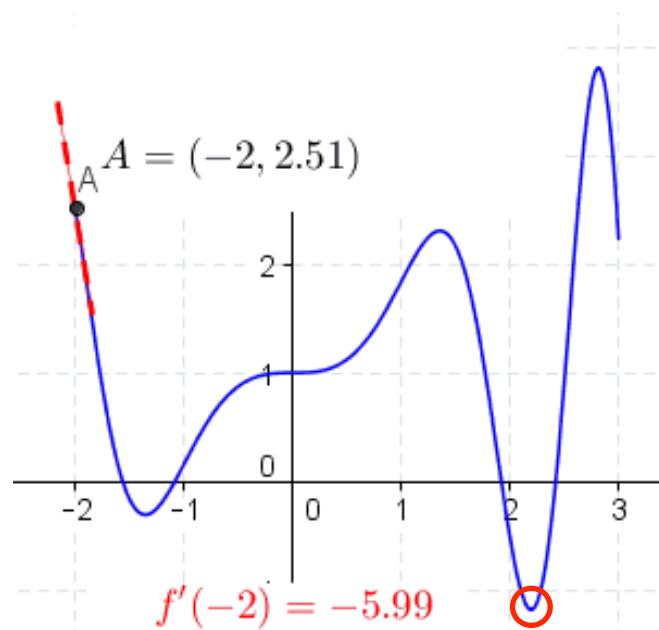
Minimizing a function



- To minimize a function, we seek a point where the derivative vanishes, i.e., we search for w^* such that
$$\frac{dR(w^*)}{dw} = 0$$
- For some simple, convex functions, w^* can be obtained algebraically by solving the corresponding equation
 - This is referred to as closed-form solution

More complicated functions

- For more complicated, non-convex functions, there are typically multiple, local minima



- Note that, in this example, there is still only one global minimum

Properties of derivatives

- The derivative of a sum (average) is the sum (average) of the derivatives
- In our ML context, this means that the derivative of the empirical risk at parameter \bar{w} can be computed as

$$\frac{dR(\bar{w})}{dw} = \frac{1}{N} \sum_{i=1}^N \frac{d\ell(\hat{y}_i(\mathbf{x}_i, \bar{w}), y_i)}{dw}$$

derivative of the loss
for each sample

Dealing with multivariate functions

- In practice, we typically have more than one parameter
 - E.g., even with a single input dimension, linear regression has two parameters ($w^{(0)}$ and $w^{(1)}$)
- To handle this, we rely on the function *gradient*
 - The gradient is a multi-variable generalization of the derivative
 - For a function with D parameters, the gradient is a D -dimensional vector
 - Each element of this vector is the partial derivative of the function w.r.t. a single variable, i.e., the derivative of the function w.r.t. one variable while keeping the other ones constant

Gradient

- Formally, the gradient of a function $R(\mathbf{w})$ with $\mathbf{w} \in \mathbb{R}^D$ is denoted by

$$\nabla R(\mathbf{w}) = \begin{bmatrix} \frac{\partial R}{\partial w^{(1)}} \\ \frac{\partial R}{\partial w^{(2)}} \\ \vdots \\ \frac{\partial R}{\partial w^{(D)}} \end{bmatrix} \in \mathbb{R}^D$$

where $\frac{\partial R}{\partial w^{(j)}}$ denotes the partial derivative w.r.t. variable $w^{(j)}$

Properties of gradient

- The gradient at a point \mathbf{w} has the direction of greatest increase of the function at \mathbf{w}
- Its magnitude is the rate of increase in that direction

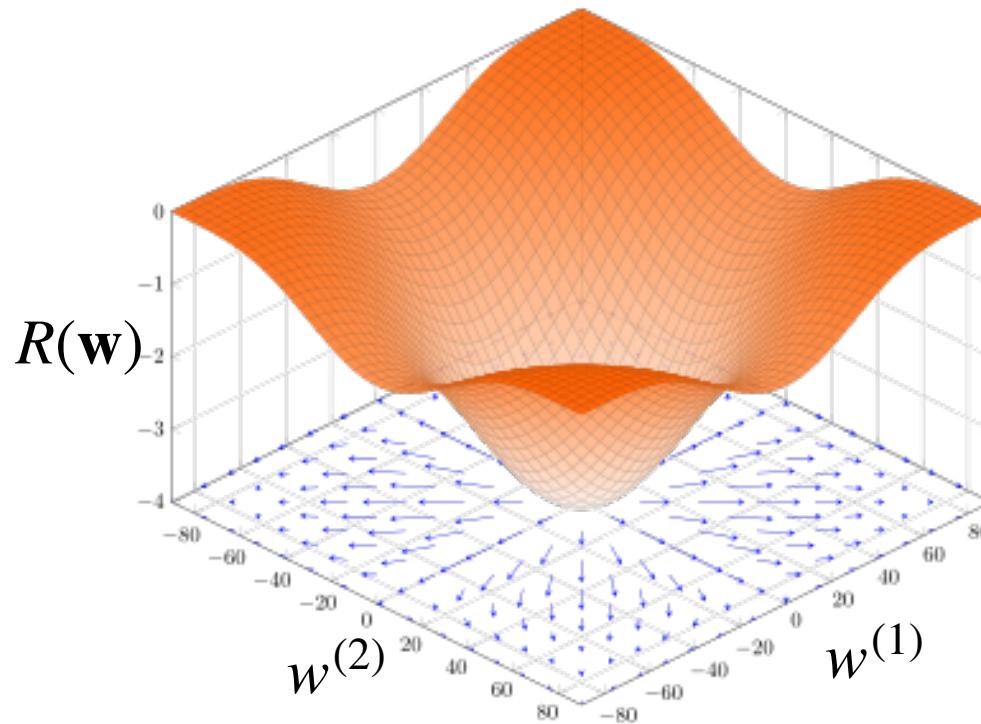
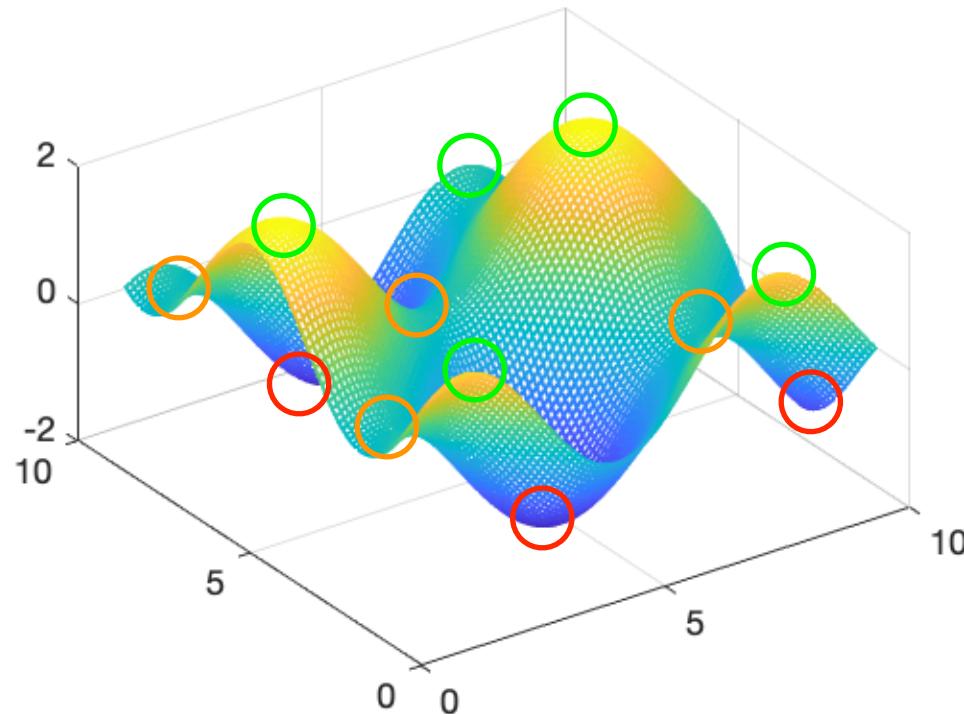


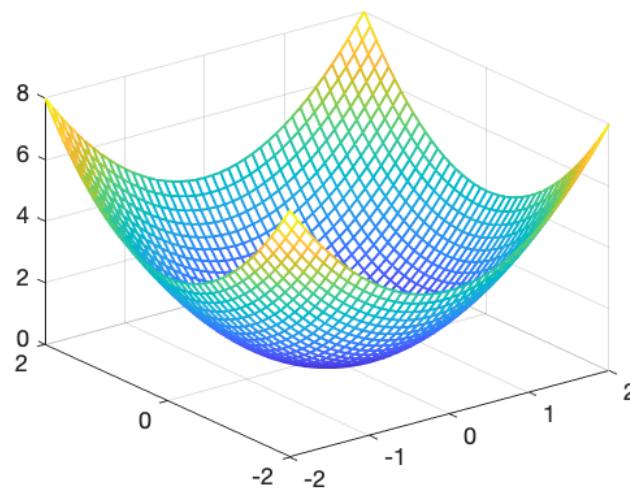
Figure from Wikipedia

Properties of gradient

- The gradient vanishes (becomes a zero vector) at the stationary points of the function
 - Minima
 - Maxima
 - Saddle points



Minimizing a multivariate function



- As in the 1D case, to minimize a function, we seek the point \mathbf{w}^* where the gradient vanishes, i.e.,

$$\nabla R(\mathbf{w}^*) = \mathbf{0}$$

- Recall that the gradient is a vector, so we have a system of equations
- This can still be solved in closed form for some functions

End of the interlude

Back to Linear Regression

1D linear regression: Solution

- With our least-square formulation, our empirical risk is

$$R = \frac{1}{N} \sum_{i=1}^N d^2(\hat{y}_i, y_i) = \frac{1}{N} \sum_{i=1}^N (w^{(1)}x_i + w^{(0)} - y_i)^2$$

- This gives us the partial derivatives

$$\frac{\partial R}{\partial w^{(0)}} = \frac{2}{N} \sum_{i=1}^N (w^{(1)}x_i + w^{(0)} - y_i)$$

$$\frac{\partial R}{\partial w^{(1)}} = \frac{2}{N} \sum_{i=1}^N (w^{(1)}x_i + w^{(0)} - y_i) \cdot x_i$$

- We would like to find the values of $w^{(0)}$ and $w^{(1)}$ such that these partial derivatives become 0

1D Linear regression: Solution

- To do this, it is easier to group the parameters $w^{(0)}$ and $w^{(1)}$ into a single parameter vector $\mathbf{w} \in \mathbb{R}^2$
- This lets us re-write our least-square empirical risk as

$$R = \frac{1}{N} \sum_{i=1}^N \left(\mathbf{w}^T \begin{bmatrix} x_i \\ 1 \end{bmatrix} - y_i \right)^2$$

where the 1 allows us to account for $w^{(0)}$

- We can then define a vector $\mathbf{x}_i \in \mathbb{R}^2$, whose 2nd element is 1, such that

$$R = \frac{1}{N} \sum_{i=1}^N (\mathbf{x}_i^T \mathbf{w} - y_i)^2$$

Side note: Vector-based derivatives

- To compute the gradient of this empirical risk, we can make use of the following general vector derivative rule:
 - Given 2 vectors (\mathbf{a}, \mathbf{b}) in \mathbb{R}^D

$$\frac{\partial \mathbf{a}^T \mathbf{b}}{\partial \mathbf{b}} = \mathbf{a}$$

- Note that useful rules for matrix (or vector) computations can be found in the Matrix Cookbook:
 - <http://www2.imm.dtu.dk/pubdb/doc/imm3274.pdf>

1D Linear regression: Solution

- With our empirical risk $R = \frac{1}{N} \sum_{i=1}^N (\mathbf{x}_i^T \mathbf{w} - y_i)^2$

we can then use the chain rule:

$$\begin{aligned}\nabla_{\mathbf{w}} R &= \frac{1}{N} \sum_{i=1}^N \frac{\partial(\mathbf{x}_i^T \mathbf{w} - y_i)^2}{\partial(\mathbf{x}_i^T \mathbf{w} - y_i)} \cdot \frac{\partial(\mathbf{x}_i^T \mathbf{w} - y_i)}{\partial \mathbf{w}} \\ &= \frac{2}{N} \sum_{i=1}^N (\mathbf{x}_i^T \mathbf{w} - y_i) \cdot \mathbf{x}_i\end{aligned}$$

- One can easily verify that, because the 2nd element of \mathbf{x}_i is 1, this matches the partial derivatives on Slide 28

1D Linear regression: Solution

- To obtain the solution, we search for \mathbf{w}^* such that

$$\nabla R = \frac{2}{N} \sum_{i=1}^N \mathbf{x}_i (\mathbf{x}_i^T \mathbf{w}^* - y_i) = \mathbf{0}$$

(Note that I moved \mathbf{x}_i in front of $(\mathbf{x}_i^T \mathbf{w} - y_i)$, which makes no difference since $(\mathbf{x}_i^T \mathbf{w} - y_i)$ is a scalar)

- This means

$$\mathbf{w}^* = \left(\sum_{i=1}^N \mathbf{x}_i \mathbf{x}_i^T \right)^{-1} \left(\sum_{i=1}^N \mathbf{x}_i y_i \right)$$

Diagram illustrating the components of the equation:

- The first term $\left(\sum_{i=1}^N \mathbf{x}_i \mathbf{x}_i^T \right)$ is circled in red and labeled "2 × 2 matrix".
- The second term $\left(\sum_{i=1}^N \mathbf{x}_i y_i \right)$ is circled in red and labeled "2 × 1 vector".
- The result \mathbf{w}^* is circled in red.
- Red arrows point from the labels to their respective circled terms.

(Note that I removed the factor $2/N$, which has no influence on the solution)

1D Linear regression: Matrix form

- Let us now group all $\{\mathbf{x}_i\}$ and all $\{y_i\}$ in a matrix and a vector

$$\mathbf{X} = \begin{bmatrix} \mathbf{x}_1^T \\ \mathbf{x}_2^T \\ \vdots \\ \mathbf{x}_N^T \end{bmatrix} = \begin{bmatrix} x_1 & 1 \\ x_2 & 1 \\ \vdots & \vdots \\ x_N & 1 \end{bmatrix} \in \mathbb{R}^{N \times 2}$$

$$\mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix} \in \mathbb{R}^N$$

1D Linear regression: Solution

- Then, we can re-write the solution as

$$\mathbf{X}^T \mathbf{X} \mathbf{w}^* = \mathbf{X}^T \mathbf{y}$$

- This finally gives us

$$\mathbf{w}^* = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y} = \mathbf{X}^\dagger \mathbf{y}$$

where \mathbf{X}^\dagger is known as the *Moore-Penrose pseudo-inverse* of \mathbf{X}

- In other words, we can obtain the solution to linear regression in closed form

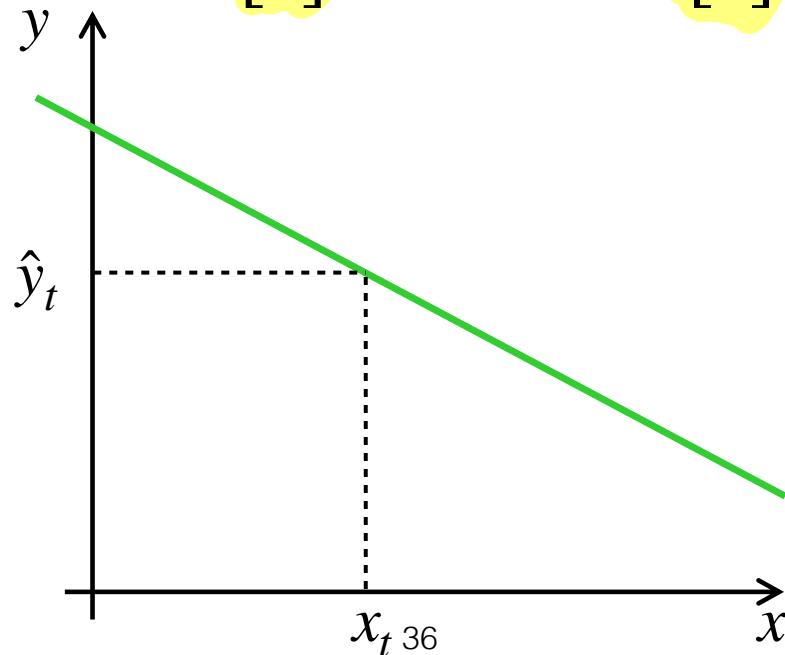
1D linear regression: Demo

- http://digitalfirst.bfwpub.com/stats_applet/stats_applet_5_correg.html

1D Linear regression: Test time

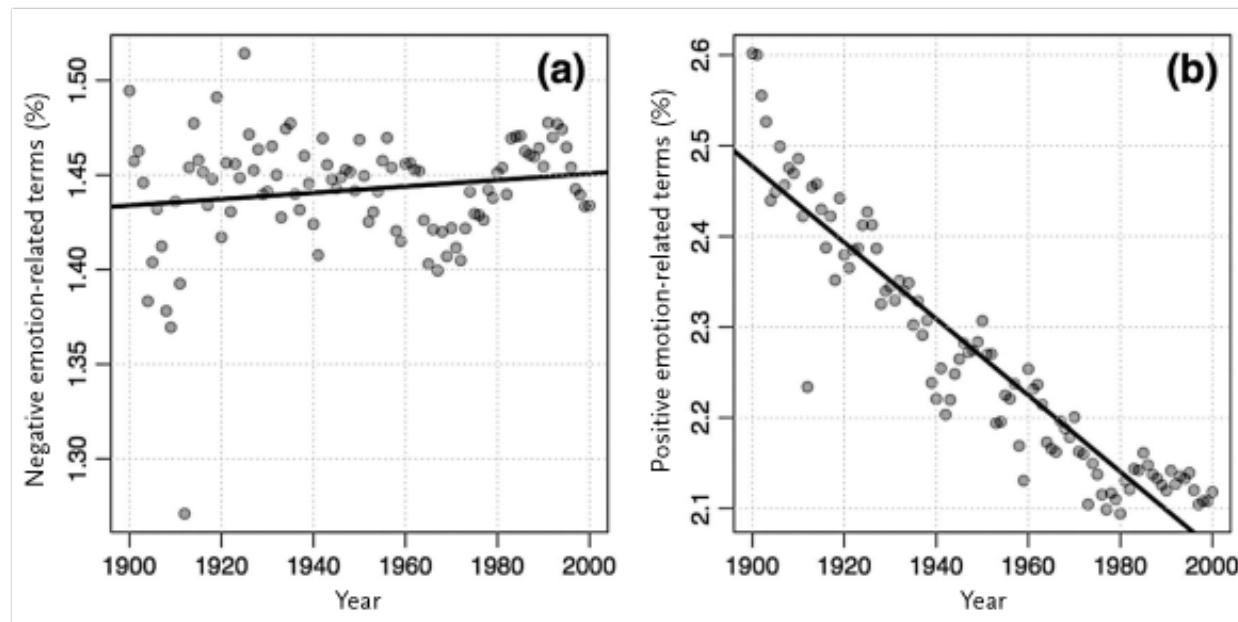
- As mentioned before, once we have the optimal parameters \mathbf{w}^* , we can predict \hat{y}_t for any new x_t
- The predicted value is given by

$$\hat{y}_t = \begin{bmatrix} x_t \\ 1 \end{bmatrix}^T \mathbf{w}^* = (\mathbf{w}^*)^T \begin{bmatrix} x_t \\ 1 \end{bmatrix}$$



1D linear regression: Test time

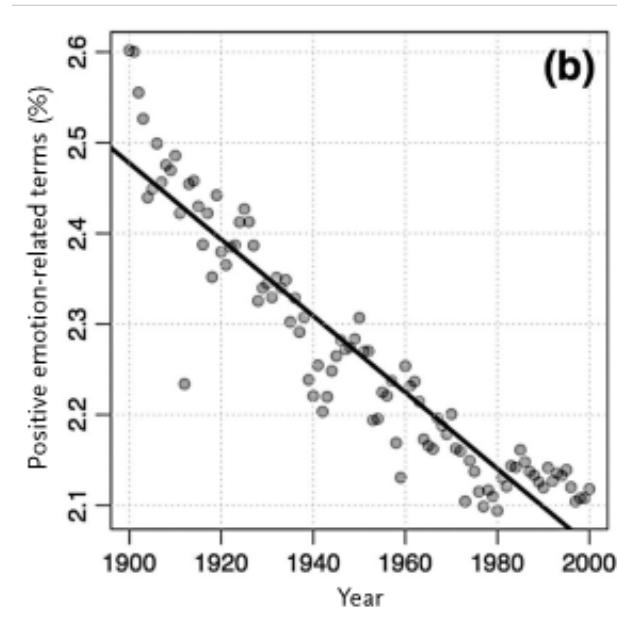
- Discover trends:
 - Example: Proportion of negative and positive emotions in anglophone fiction (Morin & Acerbi, 2016. Figure from Moretti & Sobchuk, 2019)



- With such temporal trends, one can predict what will happen in the future

Exercises

- From the trend data below



- Give an example of what numerical values one \mathbf{x}_i and the corresponding y_i could roughly take
- What would the matrix \mathbf{X} and the vector \mathbf{y} look like (using rough numerical values for a few samples)?

Dealing with multiple input dimensions

- In general, an input observation \mathbf{x}_i is not represented by a single value
 - E.g., a grayscale image can be represented by an $W \cdot H$ dimensional vector

$$\mathbf{x}_i = \text{vectorize}(\text{?}) \in \mathbb{R}^{28 \cdot 28} = \mathbb{R}^{784}$$

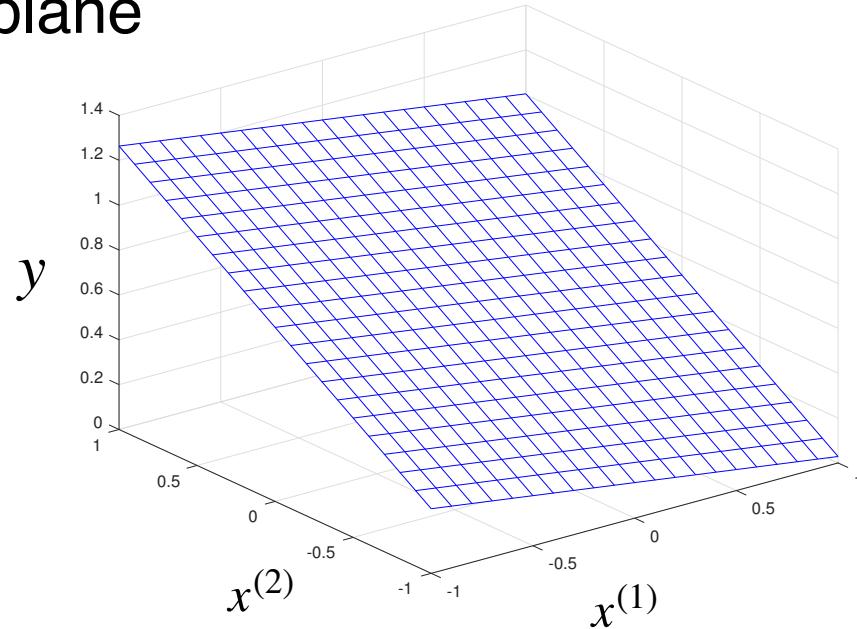
- E.g., with attribute-based representations, multiple attributes are often given

Birth weight prediction

	Age at delivery	Weight prior to pregnancy (pounds)	Smoker	Doctor visits during 1 st trimester	Race	Birth Weight (grams)
Patient 1	29	140	Yes	2	Caucasian	2977
Patient 2	32	132	No	4	Caucasian	3080
Patient 3	36	175	No	0	African-Am	3600
*	*	*	*	*	*	*
*	*	*	*	*	*	*
Patient 189	30	95	Yes	2	Asian	3147

Plane

- When there are two input dimensions, instead of a line, we can define a plane

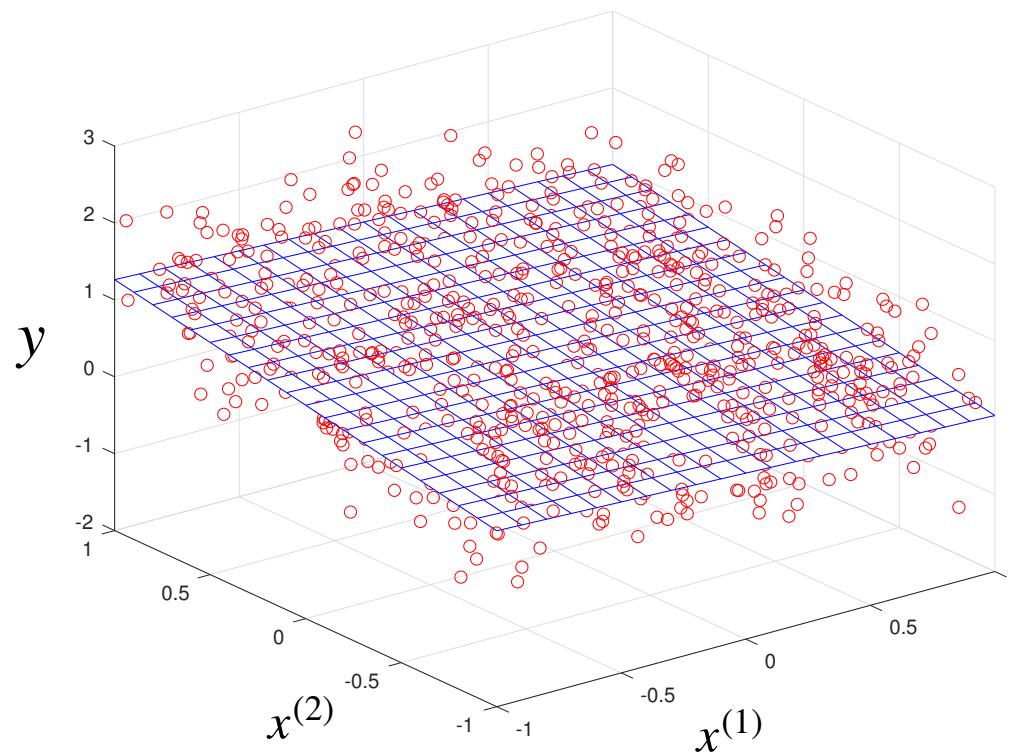


- Mathematically, a plane is expressed as

$$y = w^{(0)} + w^{(1)}x^{(1)} + w^{(2)}x^{(2)} = \mathbf{w}^T \begin{bmatrix} x^{(1)} \\ x^{(2)} \\ 1 \end{bmatrix}$$

Plane fitting

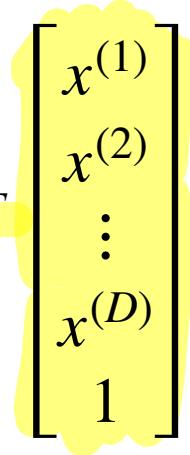
- Given N noisy pairs $\{(\mathbf{x}_i, y_i)\}$, where $\mathbf{x}_i \in \mathbb{R}^2$, find the plane that best fits these observations



Hyperplane

- This can be generalized to higher dimensions
- In dimension D , we can write

$$y = w^{(0)} + w^{(1)}x^{(1)} + w^{(2)}x^{(2)} + \dots + w^{(D)}x^{(D)} = \mathbf{w}^T$$


$$\begin{bmatrix} x^{(1)} \\ x^{(2)} \\ \vdots \\ x^{(D)} \\ 1 \end{bmatrix}$$

- Ultimately, whatever the dimension, we can write

$$y = \mathbf{w}^T \mathbf{x}$$

with $\mathbf{x} \in \mathbb{R}^{D+1}$, where the extra dimension contains a 1 to account for $w^{(0)}$

Multi-input linear regression: Training

- Because the output remains 1D, we can use the same least-square loss function as before, and write training as

$$\min_{\mathbf{w}} \sum_{i=1}^N d^2(\hat{y}_i, y_i) \Leftrightarrow \min_{\mathbf{w}} \sum_{i=1}^N (\mathbf{x}_i^T \mathbf{w} - y_i)^2$$

- Note that this has the same form as in the 1D input case when we grouped $w^{(0)}$ and $w^{(1)}$ in a single vector
 - And I removed the $1/N$ factor in front of the sum, which, as shown before, is unnecessary because it does not affect the solution

Multi-input linear regression: Solution

- Therefore the solution, obtained by zeroing out the gradient of the empirical risk, is exactly the same as before

$$\mathbf{w}^* = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y} = \mathbf{X}^\dagger \mathbf{y}$$

- But now, the matrix $\mathbf{X} \in \mathbb{R}^{N \times (D+1)}$, instead of $\mathbb{R}^{N \times 2}$ before
- Note that this makes perfect sense:
 - Before, we had 1D inputs, and thus $D = 1$ (and so $D + 1 = 2$)

Linear regression: Demo

- <https://playground.tensorflow.org/#activation=linear&batchSize=10&dataset=circle®Dataset=reg-plane&learningRate=0.03®ularizationRate=0&noise=0&networkShape=&seed=0.45772&showTestData=false&discretize=false&percTrainData=50&x=true&y=true&xTimesY=false&xSquared=false&ySquared=false&cosX=false&sinX=false&cosY=false&sinY=false&collectStats=false&problem=regression&initZero=false&hideText=false>

Interlude

Interpreting a Linear Model

Linear regression: Example

- UCI Wine Quality dataset:
 - Predict the quality of wine based on several attributes (5 samples shown below)

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
0	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4	5
1	7.8	0.88	0.00	2.6	0.098	25.0	67.0	0.9968	3.20	0.68	9.8	5
2	7.8	0.76	0.04	2.3	0.092	15.0	54.0	0.9970	3.26	0.65	9.8	5
3	11.2	0.28	0.56	1.9	0.075	17.0	60.0	0.9980	3.16	0.58	9.8	6
4	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4	5

- Final RMSE:
 - 0.65 for training data
 - 0.63 for test data

Example from <https://medium.com/datadriveninvestor/regression-from-scratch-wine-quality-prediction-d61195cb91c8>

Linear regression: Example

- One can then look at the coefficient values (i.e., the $\{w^{(i)}\}$) to see the influence of each attribute

	Coeffecient
fixed acidity	0.017737
volatile acidity	-0.992560
citric acid	-0.139629
chlorides	-1.590943
free sulfur dioxide	0.005597
total sulfur dioxide	-0.003520
density	0.768590
pH	-0.437414
sulphates	0.812888
alcohol	0.301484

Linear regression: Example

- Author age prediction from text
 - N'guyen et al., Workshop on Language Technology for Cultural Heritage, Social Sciences, and Humanities, 2011
 - Influence of features (words)

Younger people

like	-1.295
gender-male	-0.539
LIWC-School	-0.442
just	-0.354
LIWC-Anger	-0.303
LIWC-Cause	-0.290
mom	-0.290
so	-0.271
definitely	-0.263
LIWC-Negemo	-0.256

Older people

years	0.601
POS - dt	0.485
LIWC - Incl	0.483
POS - prp vbp	0.337
granddaughter	0.332
grandchildren	0.293
had	0.277
daughter	0.272
grandson	0.245
ah	0.243

Linear regression: Example

- Example: True age 17, predicted age 16.48

"I can't sleep, but this time I have school tommorow, so I have to try I guess. My parents got all pissed at me today because I forgot how to do the homework [...]. Really mad, I ended it pissing off my mom and [...] NOTHING! Damn, when I'm at my cousin's I have no urge to use the computer like I do here, [...]"

- Example: True age 70, predicted age 71.53

"[...] I was a little bit fearful of having surgery on both sides at once (reduction and lift on the right, tissue expander on the left) [...] On the good side, my son and family live near the plastic surgeon's office and the hospital, [...], at least from my son and my granddaughter [...]"

Interpreting a linear model

- Warning: The magnitude of a coefficient will depend on the magnitude of the corresponding feature/attribute
 - A coefficient might be very small simply to compensate for the fact that the range of the feature is very large
 - E.g., looking at 2 attributes from the UCI Wine Quality example

chlorides	free sulfur dioxide
0.076	11.0
0.098	25.0
0.092	15.0

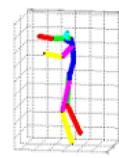
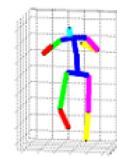
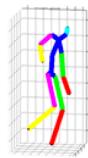
- This can be addressed by normalizing the data
 - We will discuss this in an upcoming lecture

End of the interlude

Let's continue with linear regression but now with
multiple *output* dimensions

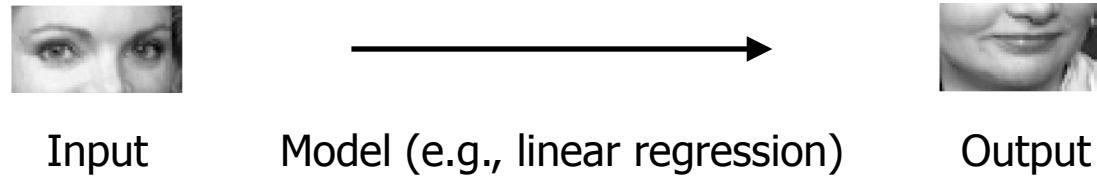
Dealing with multiple output dimensions

- Until now, we have assumed that the output was a single value, i.e., $y_i \in \mathbb{R}$
- In practice, however, one may want to output multiple values for a given input, i.e., $\mathbf{y}_i \in \mathbb{R}^C$, with $C > 1$
- E.g., human pose estimation
 - We aim to predict the 3D position of each joint, i.e., for J joints, $C = 3J$



Multi-output linear regression: Example

- Face completion:
 - Example from [https://scikit-learn.org/stable/auto_examples/misce...
nous-plot-multioutput-face-completion.py](https://scikit-learn.org/stable/auto_examples/miscellaneous/plot_multioutput_face_completion.html#sphx-glr-auto-examples-miscellaneous-plot-multioutput-face-completion-py)
 - Task: Given the top half image of a face, predict the bottom half



- Dataset: Olivetti faces
 - 40 subjects (35 for training, 5 for testing)
 - 10 images per subject

Multi-output linear regression: Example

- Face completion:

- Results:

Linear regression



true faces



Multi-output linear regression: Model

- To output multiple values, the linear model cannot just rely on a vector \mathbf{w} , because the product $\mathbf{w}^T \mathbf{x}_i$ yields a single value
- We therefore use a matrix $\mathbf{W} \in \mathbb{R}^{(D+1) \times C}$, such that

$$\hat{\mathbf{y}}_i = \mathbf{W}^T \mathbf{x}_i = \begin{bmatrix} \mathbf{w}_{(1)}^T \\ \mathbf{w}_{(2)}^T \\ \vdots \\ \mathbf{w}_{(C)}^T \end{bmatrix} \mathbf{x}_i$$

where each $\mathbf{w}_{(j)}$ is a $(D + 1)$ -dimensional vector, used to predict one output dimension

Multi-output linear regression: Training

- Since the outputs are multi-dimensional, we need to slightly modify the loss function
- The multi-dimensional counterpart of the 1D least-square loss is the squared Euclidean distance
- So we can write multi-output linear regression as

$$\min_{\mathbf{W}} \sum_{i=1}^N \|\mathbf{W}^T \mathbf{x}_i - \mathbf{y}_i\|^2$$

where $\|\mathbf{a}\| = \sqrt{\sum_{j=1}^C (a^{(j)})^2}$ indicates the norm of a vector (here $\mathbf{a} \in \mathbb{R}^C$)

Multi-output linear regression: Gradient

- To compute the gradient of this function, we make use of the following two properties:

1. Norm and derivative:

$$\|\mathbf{a}\|^2 = \mathbf{a}^T \mathbf{a}$$

and thus (by the vector derivative we have seen before)

$$\frac{\partial \|\mathbf{a}\|^2}{\partial \mathbf{a}} = 2\mathbf{a}$$

2. Matrix derivative:

$$\frac{\partial \mathbf{a}^T \mathbf{B}}{\partial \mathbf{B}} = \mathbf{a}$$

Multi-output linear regression: Gradient

- Thus, using the chain rule (and replacing $(\mathbf{W}^T \mathbf{x}_i - \mathbf{y}_i)$ with $(\mathbf{x}_i^T \mathbf{W} - \mathbf{y}_i^T)$), we obtain the gradient

$$\nabla_{\mathbf{W}} R = 2 \sum_{i=1}^N \mathbf{x}_i (\mathbf{x}_i^T \mathbf{W} - \mathbf{y}_i^T)$$

- Setting it to zero means that

$$\mathbf{W}^* = \left(\sum_{i=1}^N \mathbf{x}_i \mathbf{x}_i^T \right)^{-1} \left(\sum_{i=1}^N \mathbf{x}_i \mathbf{y}_i^T \right)$$

$\left(\sum_{i=1}^N \mathbf{x}_i \mathbf{x}_i^T \right)$ \mathbf{W}^* $\left(\sum_{i=1}^N \mathbf{x}_i \mathbf{y}_i^T \right)$

$(D + 1) \times (D + 1)$ matrix $(D + 1) \times C$ matrix $(D + 1) \times C$ matrix

Multi-output linear regression: Solution

- We can again group the inputs in a matrix $\mathbf{X} \in \mathbb{R}^{N \times (D+1)}$, but the outputs now need to be grouped in a matrix (not a vector anymore), as

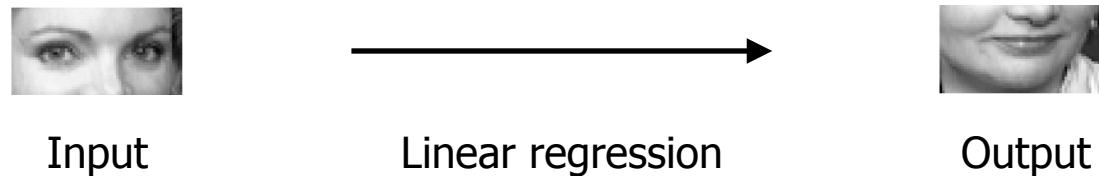
$$\mathbf{Y} = \begin{bmatrix} \mathbf{y}_1^T \\ \mathbf{y}_2^T \\ \vdots \\ \mathbf{y}_N^T \end{bmatrix} = \begin{bmatrix} y_1^{(1)} & y_1^{(2)} & \cdots & y_1^{(C)} \\ y_2^{(1)} & y_2^{(2)} & \cdots & y_2^{(C)} \\ \vdots & \vdots & \vdots & \vdots \\ y_N^{(1)} & y_N^{(2)} & \cdots & y_N^{(C)} \end{bmatrix} \in \mathbb{R}^{N \times C}$$

- Then, following the same strategy as before, we have

$$\mathbf{W}^* = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{Y} = \mathbf{X}^\dagger \mathbf{Y}$$

Exercise

- Given the following face completion task:
 - Task: Given the top half image of a face, predict the bottom half



- Assuming that there are 35 training subjects with 10 images per subject, and that a complete face image is of size 28×28 pixels, what is the shape of the matrices \mathbf{X} and \mathbf{Y} ?
- How many parameters are there to learn?

From regression to classification

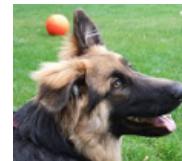
- Predict one discrete label for a given sample
 - E.g., binary classification for movies (like vs not like)

5	Column1	Column2
6	A very, very slow-moving, aimless movie about a distressed, drifting young man.	0
8	Not sure who was more lost - the flat characters or the audience, nearly half of whom walked out.	0
10	Attempting artiness with black & white and clever camera angles, the movie disappointed - became e	0
11	Very little music or anything to speak of.	0
13	The best scene in the movie was when Gerardo is trying to find a song that keeps running through his	1
15	The rest of the movie lacks art, charm, meaning... If it's about emptiness, it works I guess because it's	0
16	Wasted two hours.	0
18	Saw the movie today and thought it was a good effort, good messages for kids.	1
20	A bit predictable.	0
22	Loved the casting of Jimmy Buffet as the science teacher.	1
23	And those baby owls were adorable.	1
25	The movie showed a lot of Florida at it's best, made it look very appealing.	1
26	The Songs Were The Best And The Muppets Were So Hilarious.	1

- E.g., multi-class image recognition



Cat



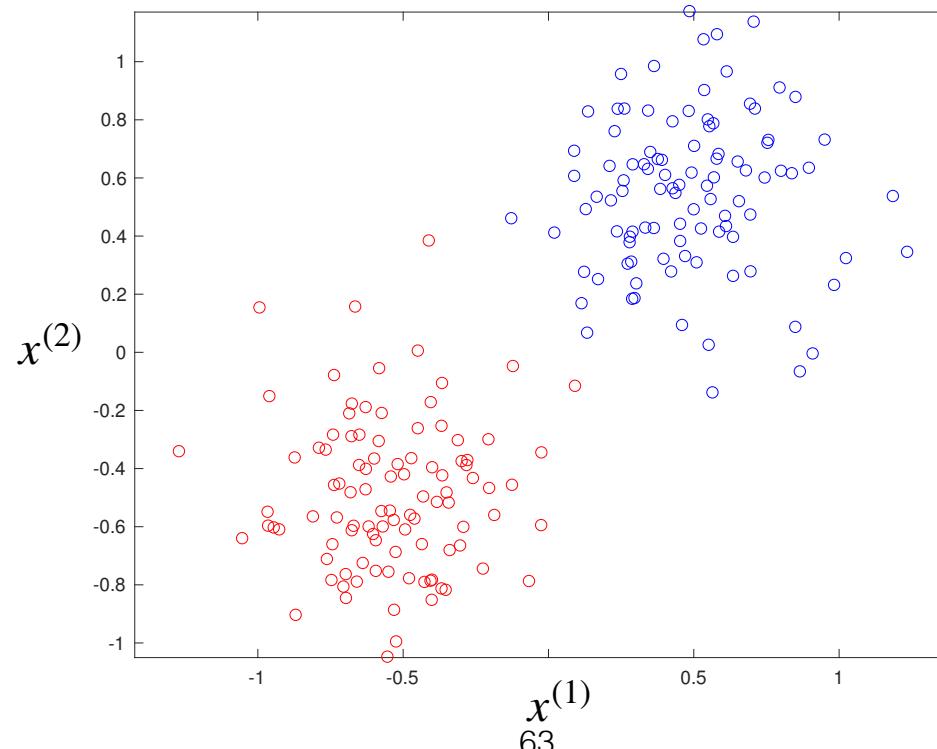
Dog



Car

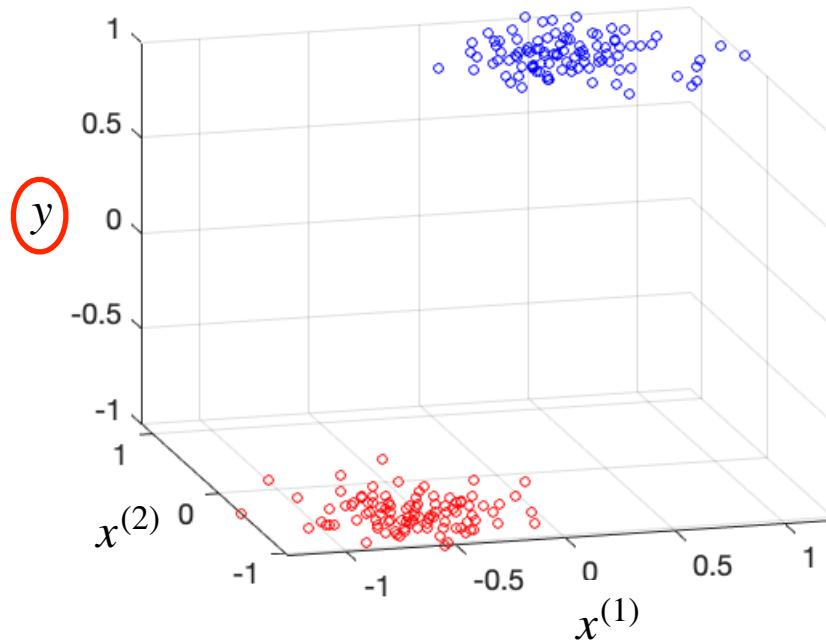
Binary classification

- Two classes (shown as different colors):
 - The label $y \in \{-1, 1\}$, or $y \in \{0, 1\}$
 - The samples with label 1 are called *positive* samples
 - The samples with label -1 (or 0) are called *negative* samples



Binary classification as regression

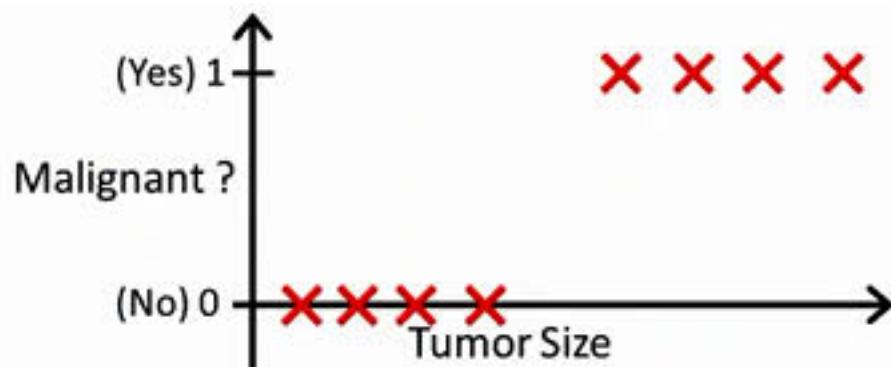
- The same data as before can also be seen in 3D
 - The label acts as the output dimension, just as in a regression problem



- This suggests that one can tackle classification as a regression task

Binary classification as regression

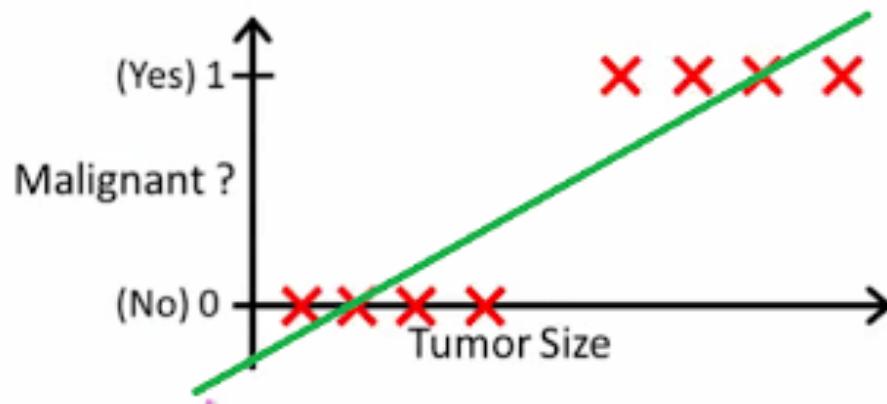
- Example (from Andrew Ng's course): Classify a tumor as benign vs malignant
 - 1D input (tumor size), 1D output (malignant vs benign)



Binary classification as regression

- Example (from Andrew Ng's course): Classify a tumor as benign vs malignant
 - 1D input (tumor size), 1D output (malignant vs benign)
 - Using a linear model, we can predict the label as

$$\hat{y} = w^{(1)}x + w^{(0)}$$



Least-square binary classification

- To fit the linear model to training data, the same strategy as for linear regression can be used:
 - Given N pairs $\{(x_i, y_i)\}$, we can use the least-square loss to write

$$\min_{\mathbf{w}} \sum_{i=1}^N (\mathbf{x}_i^T \mathbf{w} - y_i)^2$$

- Because it has exactly the same formulation as before, the solution is exactly the same, i.e.,

$$\mathbf{w}^* = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y}$$

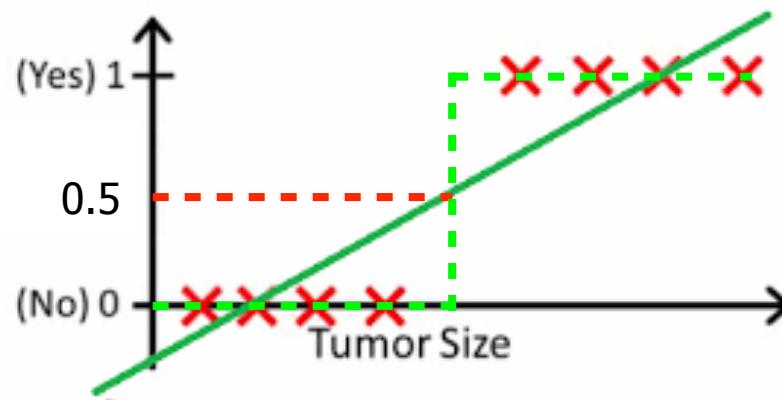
Least-square binary classification

- Linear regression outputs a continuous value
 - We would rather like to have a discrete label, 1 or 0 (-1)
 - This can be achieved by thresholding:

$$\text{label} = \begin{cases} 1 & \text{if } \hat{y} \geq 0.5 \\ 0 & \text{otherwise} \end{cases}$$

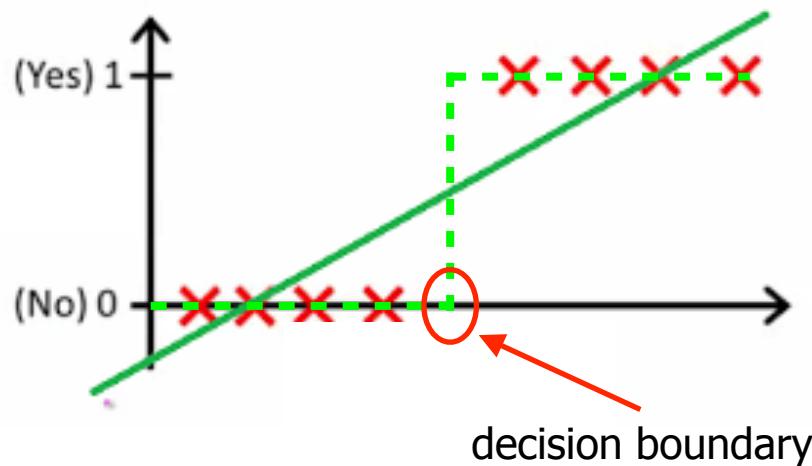
or, if the negative
label is -1:

$$\text{label} = \begin{cases} 1 & \text{if } \hat{y} \geq 0 \\ -1 & \text{otherwise} \end{cases}$$



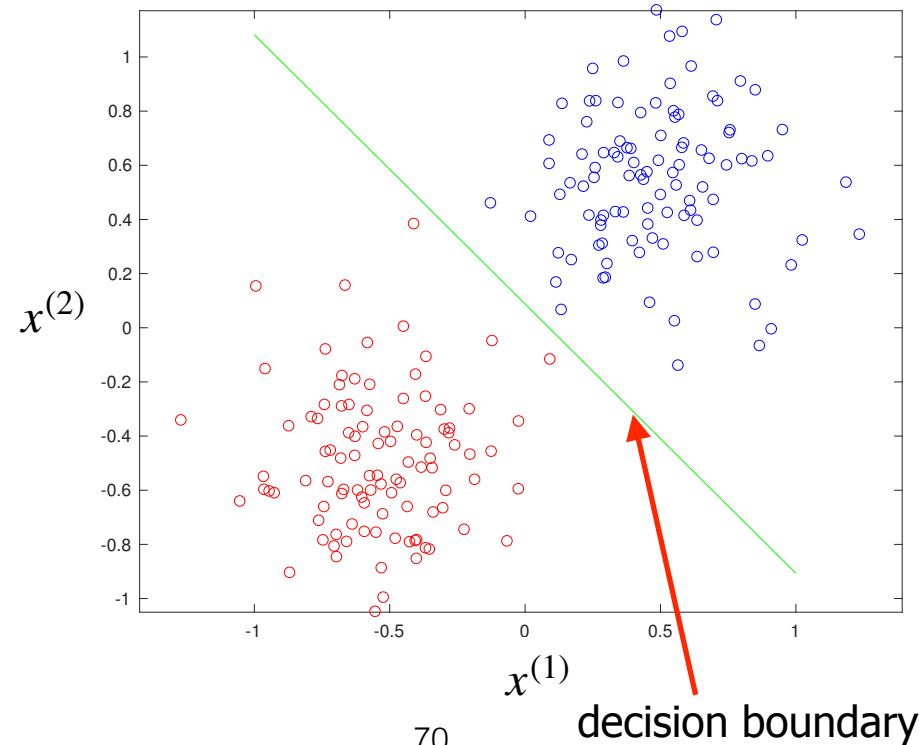
Decision boundary

- The point where the predicted label changes from 0 (or -1) to 1 forms a *decision boundary*
 - With 1D inputs, it is a single point



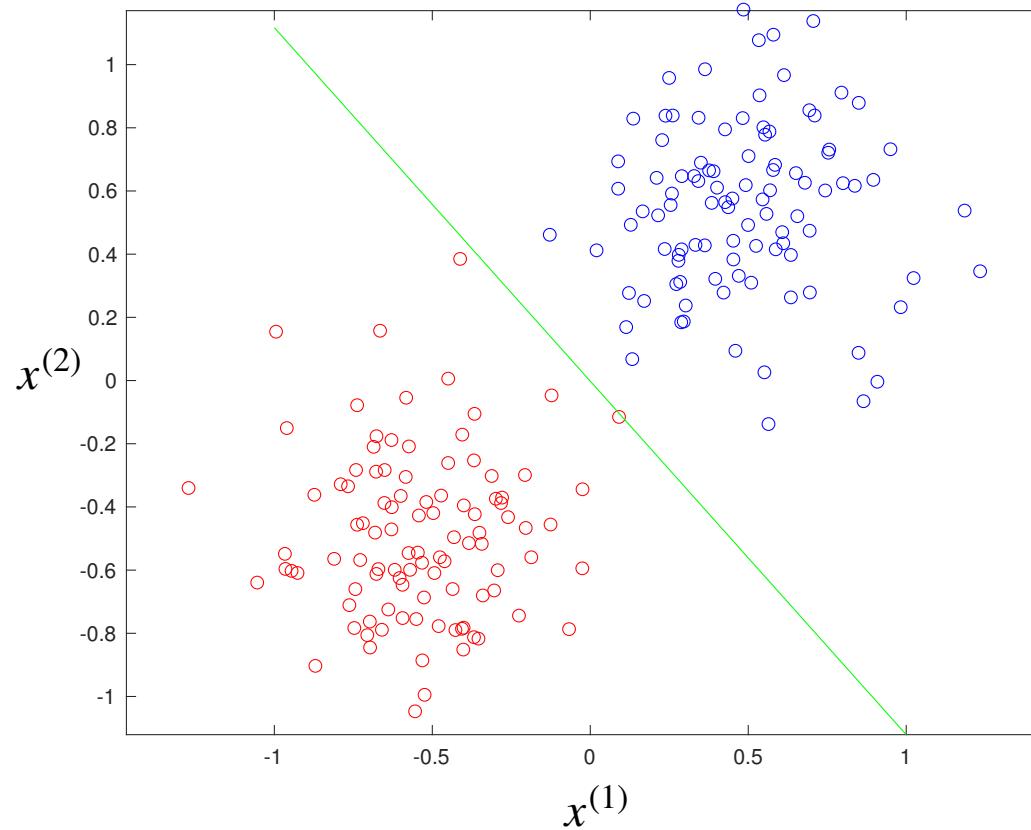
Decision boundary

- The point where the predicted label changes from 0 (or -1) to 1 forms a *decision boundary*
 - With 1D inputs, it is a single point
 - With 2D inputs, it is a line



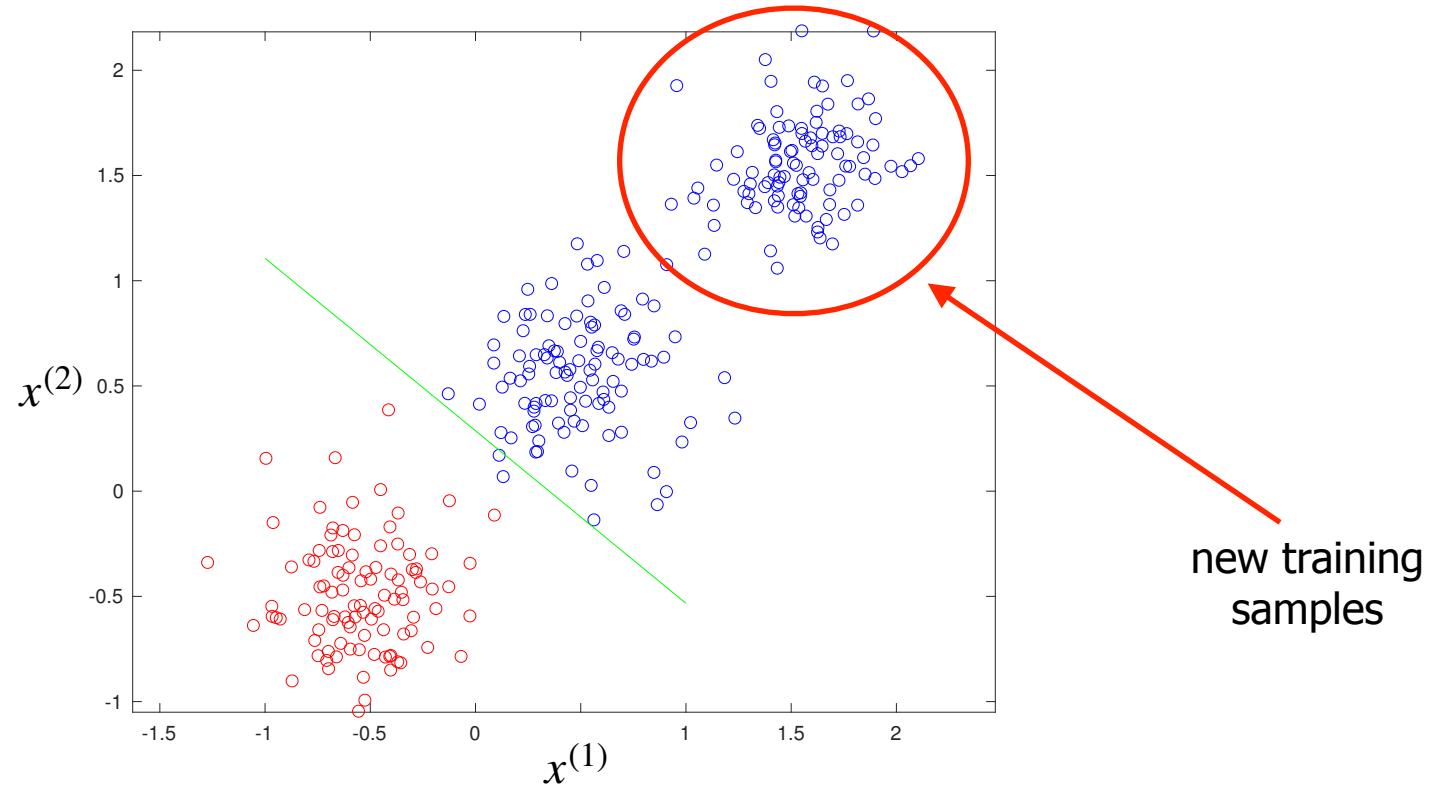
Least-square binary classification: Example

- 2D inputs, 2 classes
 - All training samples are correctly classified



Least-square binary classification: Failure

- However, adding new training samples far from the decision boundary makes some of the training samples be misclassified



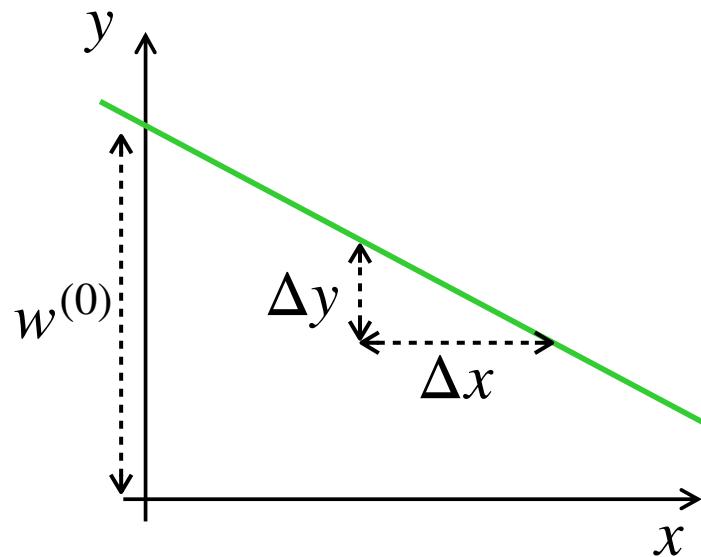
- We will study why and how to address this next week

Survey

- Please fill in the survey on the Moodle page to comment on the pace of the lecture

Lecture 3: Linear Models for Classification (Part 1)

Recap: A simple parametric model: The line

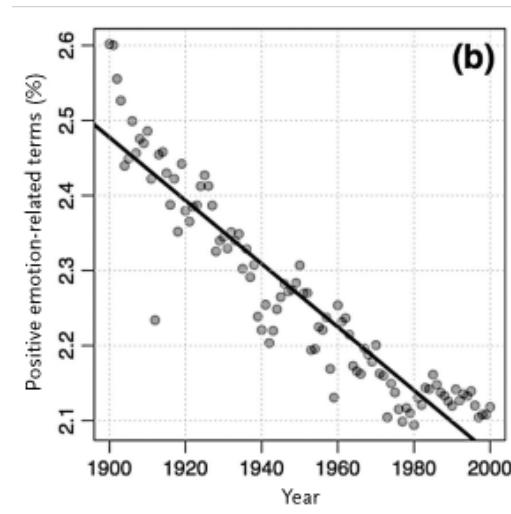


- Defined by 2 parameters
 - The y -intercept $w^{(0)}$
 - The slope $w^{(1)} = \frac{\Delta y}{\Delta x}$
- Mathematically, a line is expressed as

$$y = w^{(1)}x + w^{(0)}$$

Exercises: Solution

- From the trend data below

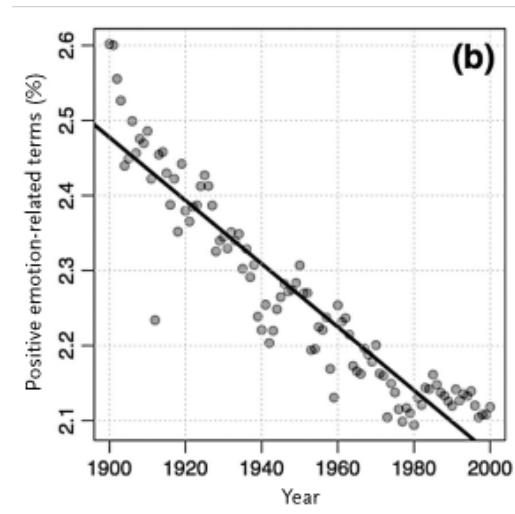


- Given an example of what numerical values one \mathbf{x}_i and the corresponding y_i could roughly take

- . Solution: E.g., $\mathbf{x}_i = \begin{bmatrix} 1940 \\ 1 \end{bmatrix}$, $y_i = 2.22$

Exercises: Solution

- From the trend data below



- What would the matrix \mathbf{X} and the vector \mathbf{y} look like (using rough numerical values for a few samples)?

Solution: $\mathbf{X} = \begin{bmatrix} 1910 & 1 \\ 1915 & 1 \\ 1920 & 1 \\ \vdots & \vdots \\ 2000 & 1 \end{bmatrix}, \quad \mathbf{y} = \begin{bmatrix} 2.45 \\ 2.43 \\ 2.38 \\ \vdots \\ 2.11 \end{bmatrix}$

Recap: Hyperplane

- This can be generalized to higher input dimensions
- In dimension D , we can write

$$y = w^{(0)} + w^{(1)}x^{(1)} + w^{(2)}x^{(2)} + \dots + w^{(D)}x^{(D)} = \mathbf{w}^T \begin{bmatrix} x^{(1)} \\ x^{(2)} \\ \vdots \\ x^{(D)} \\ 1 \end{bmatrix}$$

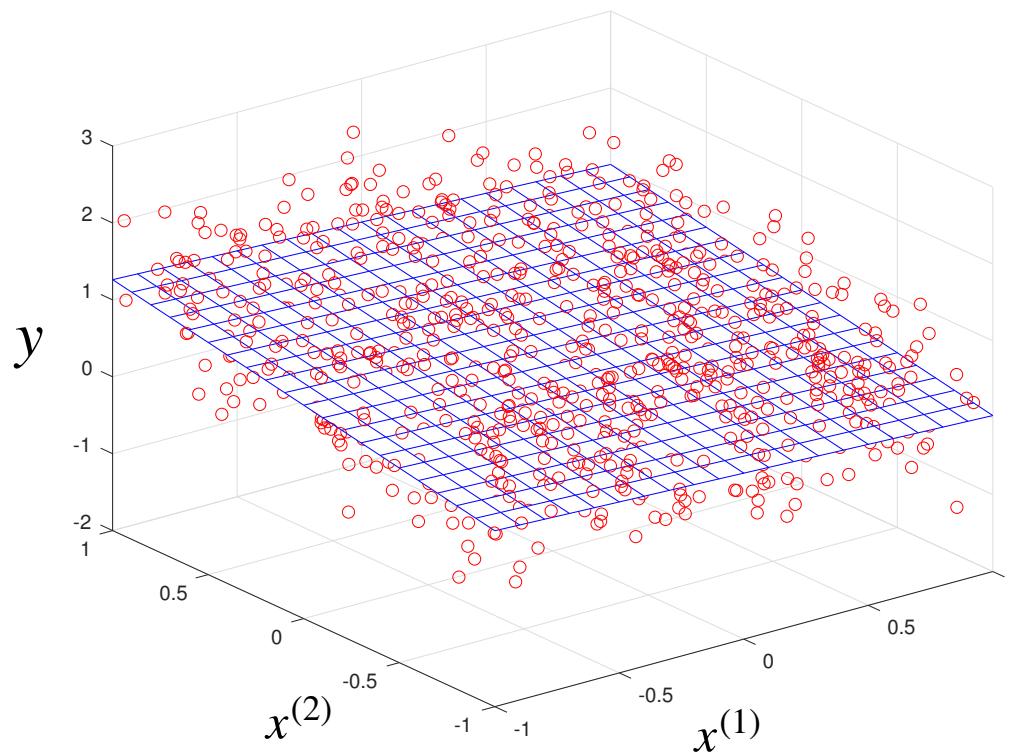
- Ultimately, whatever the input dimension, we can write

$$y = \mathbf{w}^T \mathbf{x}$$

with $\mathbf{x} \in \mathbb{R}^{D+1}$, where the extra dimension contains a 1 to account for $w^{(0)}$

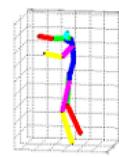
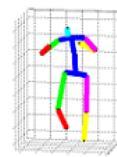
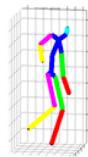
Recap: Linear regression

- Given N noisy pairs $\{(\mathbf{x}_i, y_i)\}$, where $\mathbf{x}_i \in \mathbb{R}^D$ (below, $D = 2$), find the parameters \mathbf{w}^* that best fit these observations



Recap: Dealing with multiple output dimensions

- The previous examples assumed that the output was a single value, i.e., $y_i \in \mathbb{R}$
- In practice, however, one may want to output multiple values for a given input, i.e., $\mathbf{y}_i \in \mathbb{R}^C$, with $C > 1$
- E.g., human pose estimation
 - We aim to predict the 3D position of each joint, i.e., for J joints, $C = 3J$



Recap: Multi-output prediction

- To predict multiple values, we cannot just have a vector \mathbf{w} , because the product $\mathbf{w}^T \mathbf{x}_i$ yields a single value
- We therefore use a matrix $\mathbf{W} \in \mathbb{R}^{(D+1) \times C}$, such that

$$\hat{\mathbf{y}}_i = \mathbf{W}^T \mathbf{x}_i = \begin{bmatrix} \mathbf{w}_{(1)}^T \\ \mathbf{w}_{(2)}^T \\ \vdots \\ \mathbf{w}_{(C)}^T \end{bmatrix} \mathbf{x}_i$$

where each $\mathbf{w}_{(j)}$ is a $(D + 1)$ -dimensional vector, used to predict one output dimension

Recap: Multi-output linear regression

- We write multi-output linear regression as

$$\min_{\mathbf{W}} \sum_{i=1}^N \|\mathbf{W}^T \mathbf{x}_i - \mathbf{y}_i\|^2$$

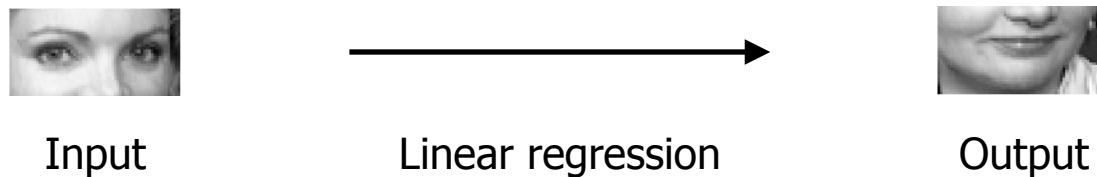
- The closed-form solution then is

$$\mathbf{W}^* = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{Y}$$

where $\mathbf{Y} \in \mathbb{R}^{N \times C}$ is a matrix stacking the N outputs

Exercises: Solution

- Given the following face completion task:
 - Task: Given the top half image of a face, predict the bottom half



- Assuming that there are 35 training subjects with 10 images per subject, and that a complete face image is of size 28×28 pixels, what is the shape of the matrices \mathbf{X} and \mathbf{Y} ?
 - Solution: For \mathbf{X} , the shape is $N \times (D + 1)$. Here $N = 35 \cdot 10 = 350$, $D = 28 \cdot 14 = 392$. For \mathbf{Y} , the shape is $N \times C$, with $C = 28 \cdot 14 = 392$.
 - How many parameters are there to learn?
 - Solution: \mathbf{W} has shape $(D + 1) \times C$. This gives $393 \cdot 392 = 154056$ parameters to learn

Goals of today's lecture

- Move from regression to classification
- Introduce the formulation for logistic regression
- Introduce basic minimization concepts
- Introduce classification evaluation metrics

From regression to classification

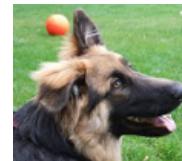
- Predict one discrete label for a given sample
 - E.g., binary classification for movies (like vs not like)

5	Column1	Column2
6	A very, very slow-moving, aimless movie about a distressed, drifting young man.	0
8	Not sure who was more lost - the flat characters or the audience, nearly half of whom walked out.	0
10	Attempting artiness with black & white and clever camera angles, the movie disappointed - became e	0
11	Very little music or anything to speak of.	0
13	The best scene in the movie was when Gerardo is trying to find a song that keeps running through his	1
15	The rest of the movie lacks art, charm, meaning... If it's about emptiness, it works I guess because it's	0
16	Wasted two hours.	0
18	Saw the movie today and thought it was a good effort, good messages for kids.	1
20	A bit predictable.	0
22	Loved the casting of Jimmy Buffet as the science teacher.	1
23	And those baby owls were adorable.	1
25	The movie showed a lot of Florida at it's best, made it look very appealing.	1
26	The Songs Were The Best And The Muppets Were So Hilarious.	1

- E.g., multi-class image recognition



Cat



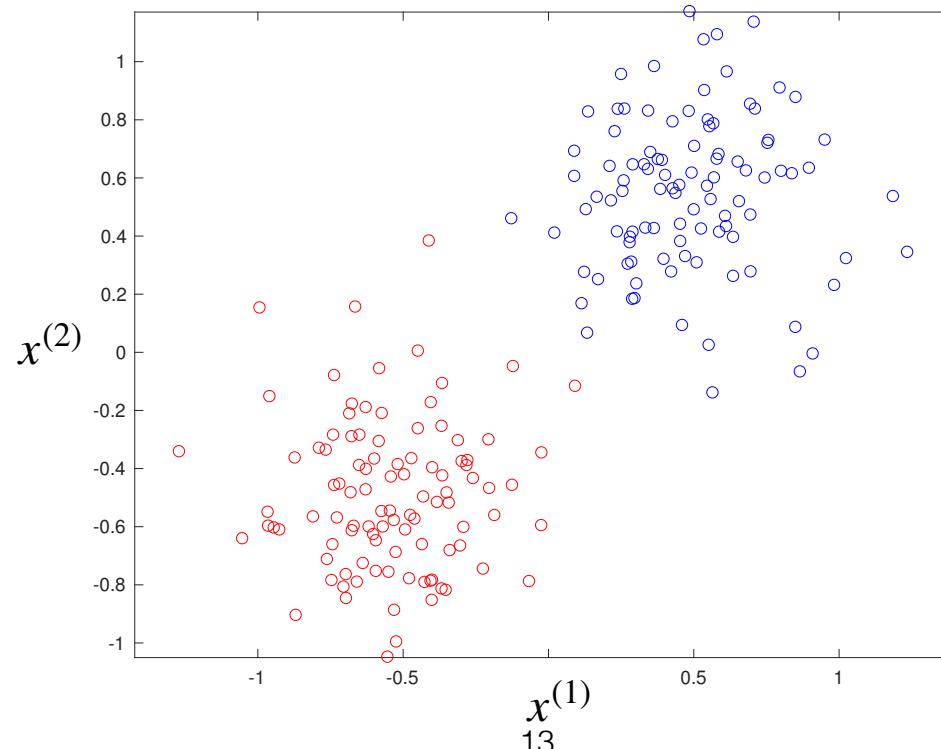
Dog



Car

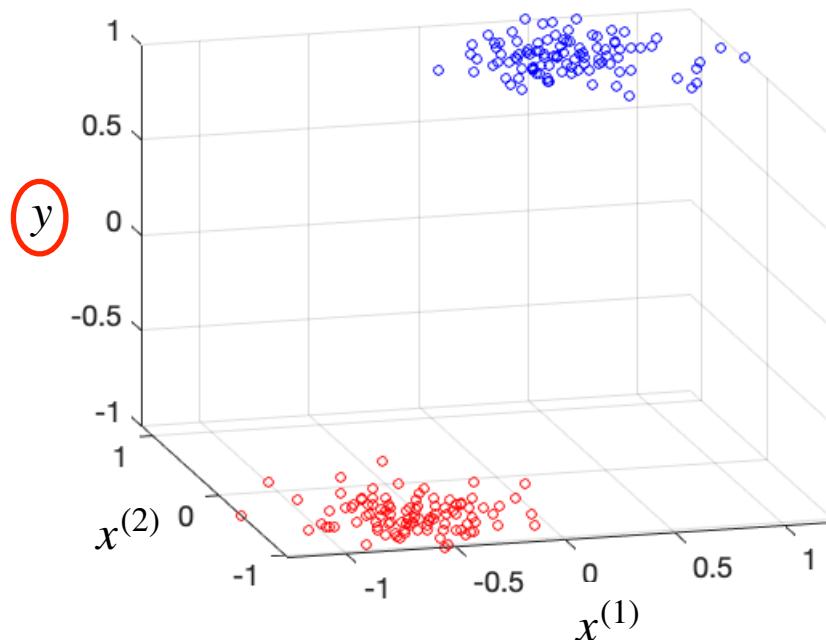
Binary classification

- Two classes (shown as different colors):
 - The label $y \in \{-1, 1\}$, or $y \in \{0, 1\}$
 - The samples with label 1 are called *positive samples*
 - The samples with label -1 (or 0) are called *negative samples*



Binary classification as regression

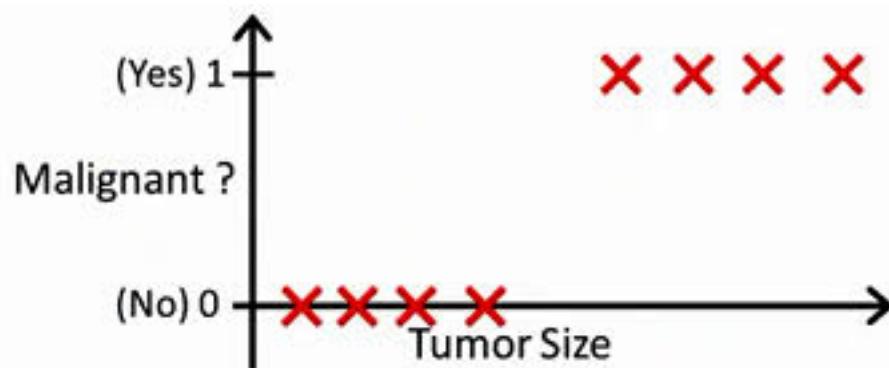
- The same data as before can also be seen in 3D
 - The label acts as the output dimension, just as in a regression problem



- This suggests that one can tackle classification as a regression task

Binary classification as regression

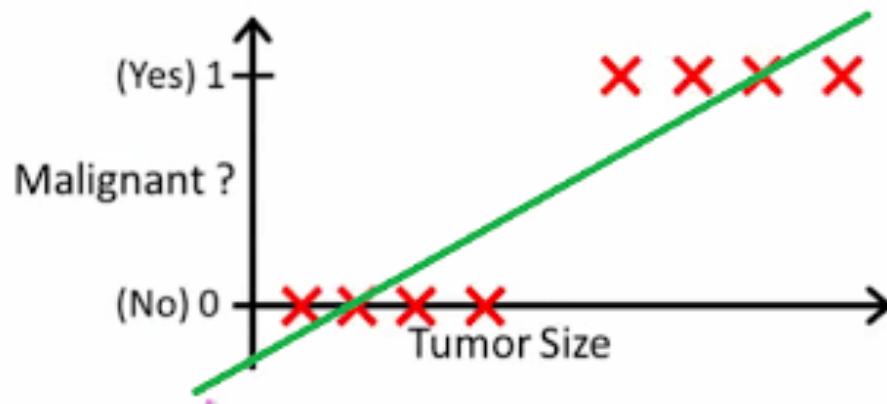
- Example (from Andrew Ng's course): Classify a tumor as benign vs malignant
 - 1D input (tumor size), 1D output (malignant vs benign)



Binary classification as regression

- Example (from Andrew Ng's course): Classify a tumor as benign vs malignant
 - 1D input (tumor size), 1D output (malignant vs benign)
 - Using a linear model, we can predict the label as

$$\hat{y} = w^{(1)}x + w^{(0)}$$



Least-square binary classification

- To fit the linear model to training data, the same strategy as for linear regression can be used:
 - Given N pairs $\{(x_i, y_i)\}$, we can use the least-square loss to write

$$\min_{\mathbf{w}} \sum_{i=1}^N (\mathbf{x}_i^T \mathbf{w} - y_i)^2$$

- Because it has exactly the same formulation as before, the solution is exactly the same, i.e.,

$$\mathbf{w}^* = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y}$$

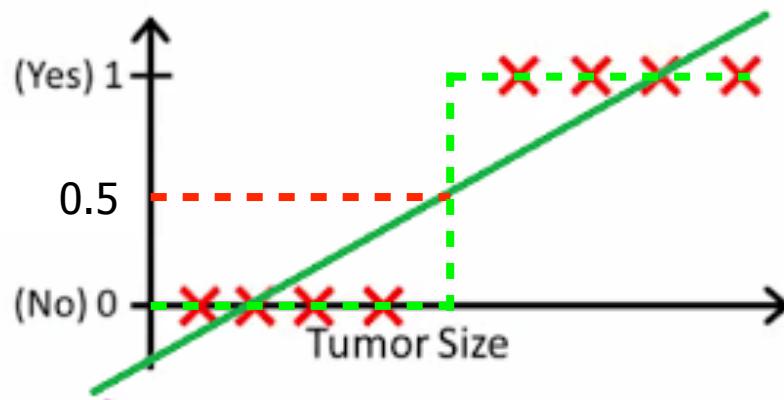
Least-square binary classification

- Linear regression outputs a continuous value
 - We would rather like to have a discrete label, 1 or 0 (-1)
 - This can be achieved by thresholding:

$$\text{label} = \begin{cases} 1 & \text{if } \hat{y} \geq 0.5 \\ 0 & \text{otherwise} \end{cases}$$

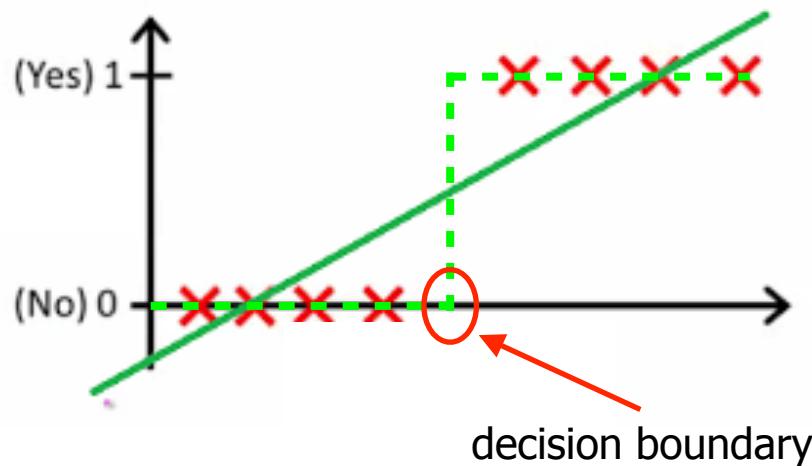
or, if the negative
label is -1:

$$\text{label} = \begin{cases} 1 & \text{if } \hat{y} \geq 0 \\ -1 & \text{otherwise} \end{cases}$$



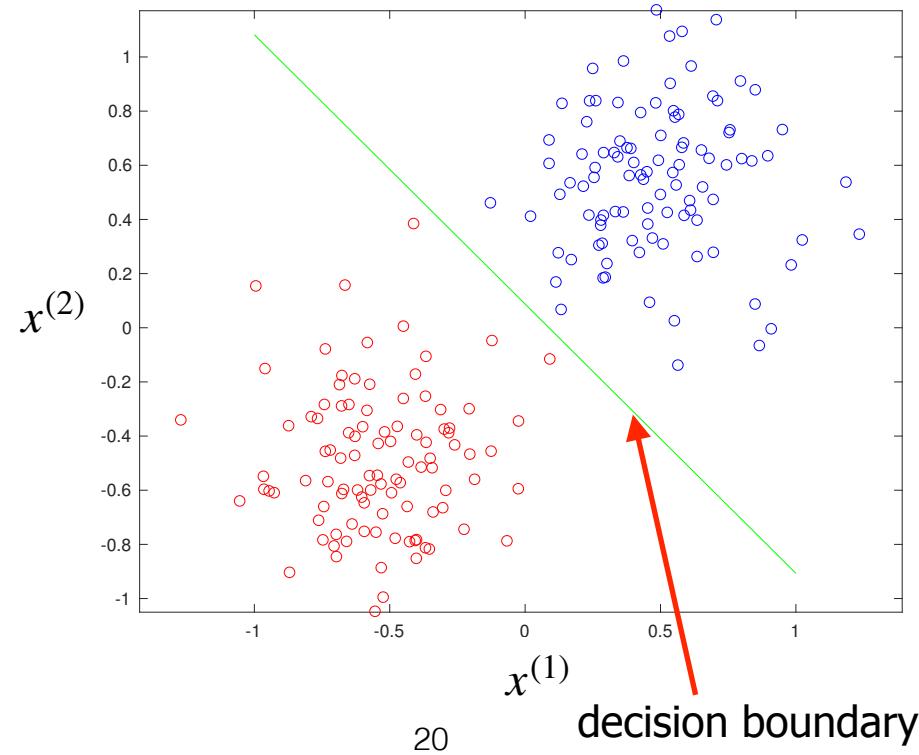
Decision boundary

- The point where the predicted label changes from 0 (or -1) to 1 forms a *decision boundary*
 - With 1D inputs, it is a single point



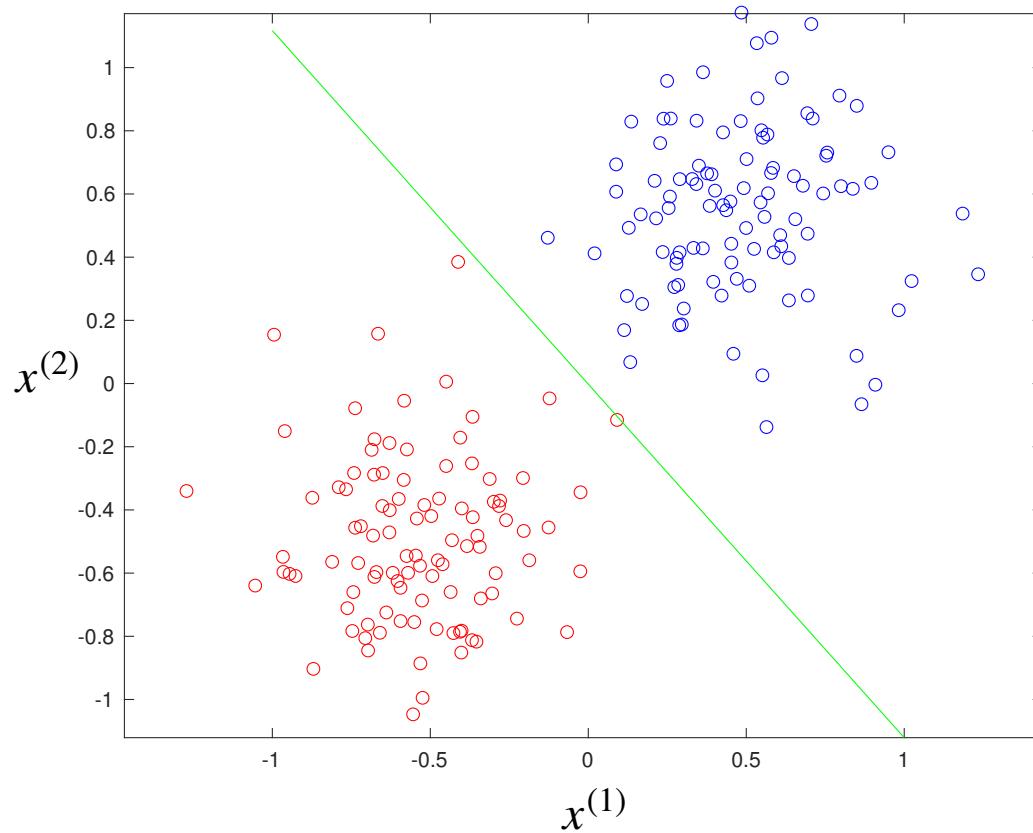
Decision boundary

- The point where the predicted label changes from 0 (or -1) to 1 forms a *decision boundary*
 - With 1D inputs, it is a single point
 - With 2D inputs, it is a line



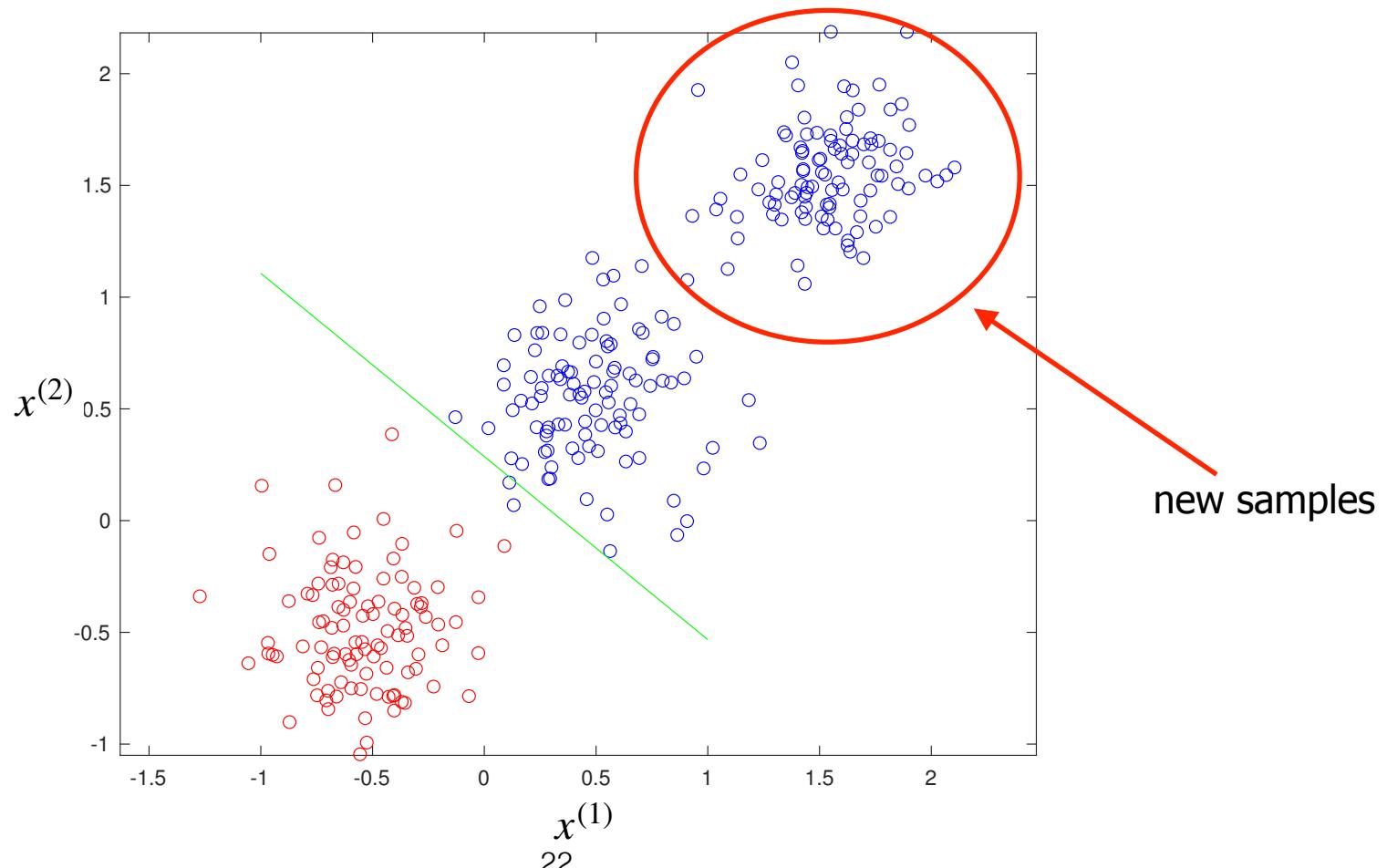
Least-square binary classification: Example

- 2D inputs, 2 classes
 - All training samples are correctly classified



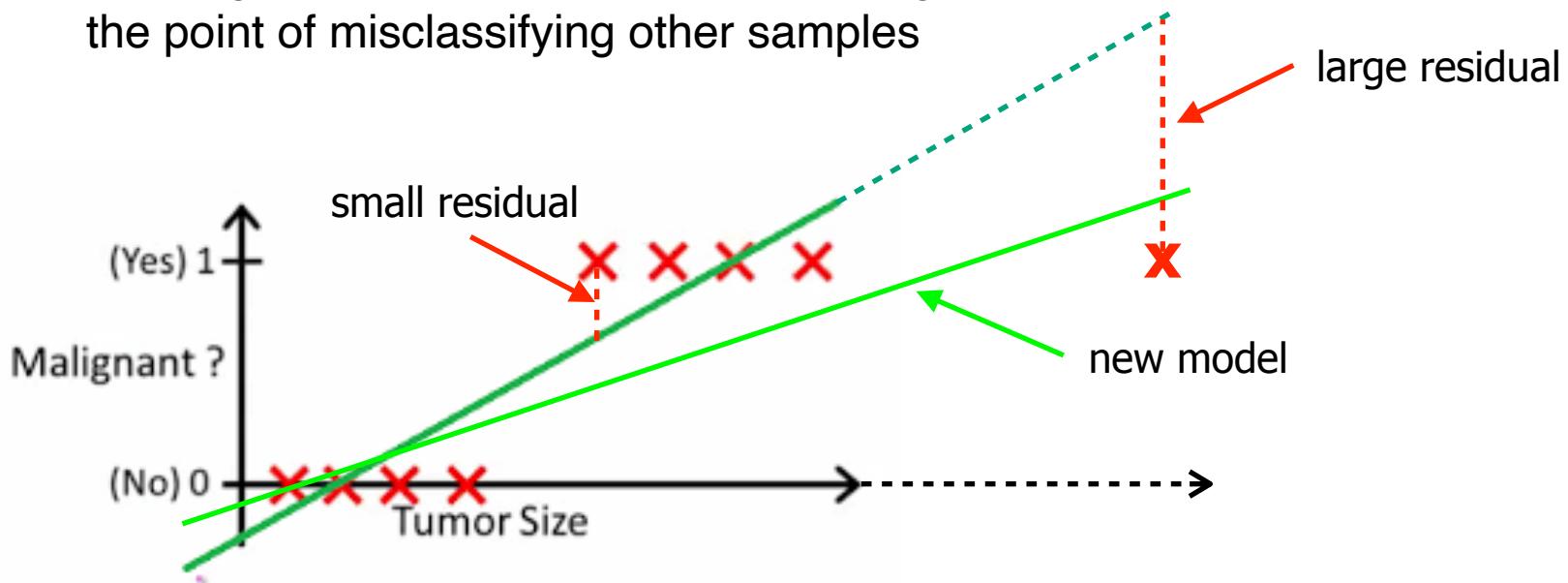
Least-square binary classification: Failure

- However, adding new samples far from the decision boundary makes some of the training samples be misclassified



Failure: Geometric interpretation

- Back to the tumor prediction example
 - The residuals for “normal” samples are quite small
 - However, a sample with a much larger tumor size (with true label 1) would have a much larger residual
 - Training with such a sample would strongly affect the model parameters, to the point of misclassifying other samples



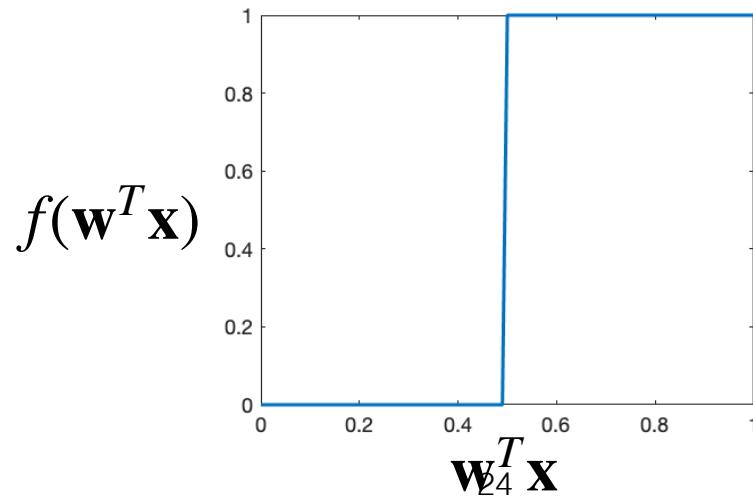
- This is because least-square classification fits a line to non-linear data

Adding non-linearity

- The output of the linear model $\mathbf{w}^T \mathbf{x}$ is a continuous value
- What we would really like is a discrete output, e.g.,

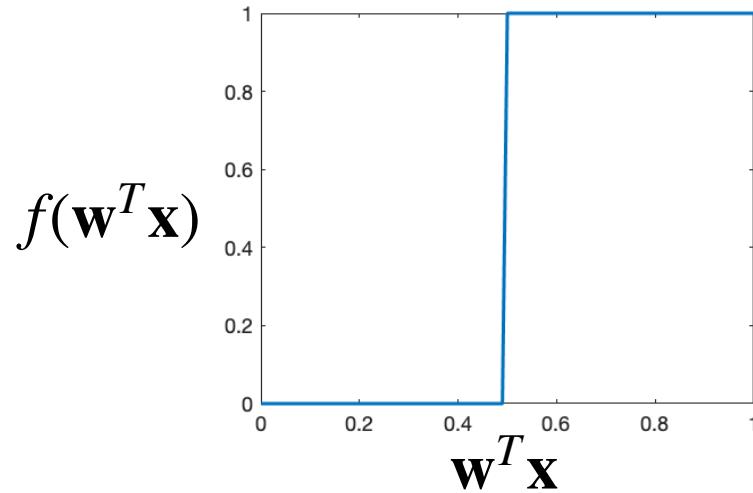
$$y = \begin{cases} 1 & \text{if } \mathbf{w}^T \mathbf{x} \geq 0.5 \\ 0 & \text{otherwise} \end{cases}$$

- This can be achieved by passing the output of the linear model through a step function



Step function: Drawback

- The gradient of the step function is not continuous
 - It is zero almost everywhere, except at $\mathbf{w}^T \mathbf{x} = 0.5$



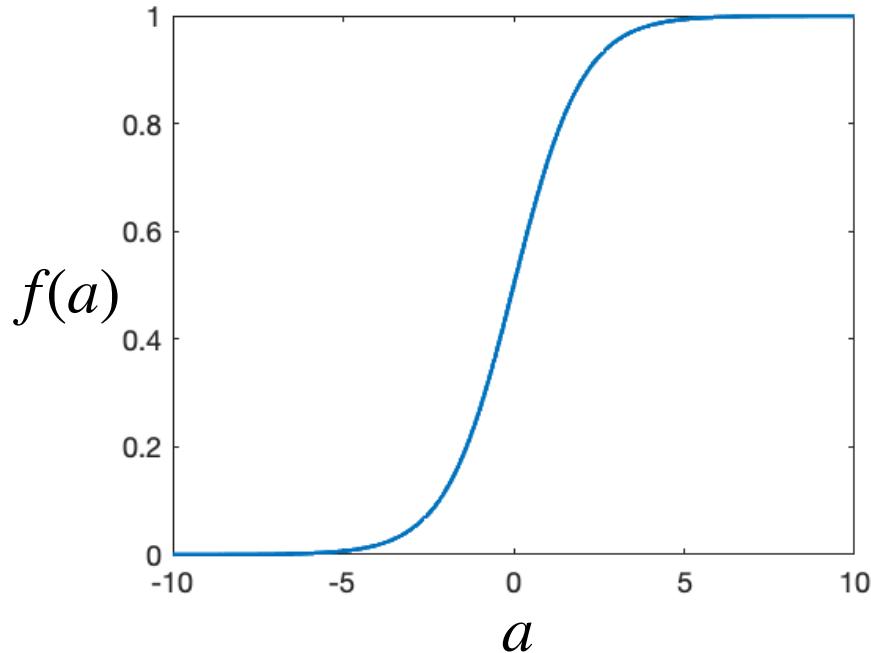
- This behavior typically makes optimization difficult
 - The Perceptron algorithm can nonetheless handle the 2-class case, but we will not cover it in this course

Adding non-linearity

- Instead, one can use a smooth approximation of the step function, such as the *logistic sigmoid function*

$$f(a) = \frac{1}{1 + \exp(-a)}$$

↑

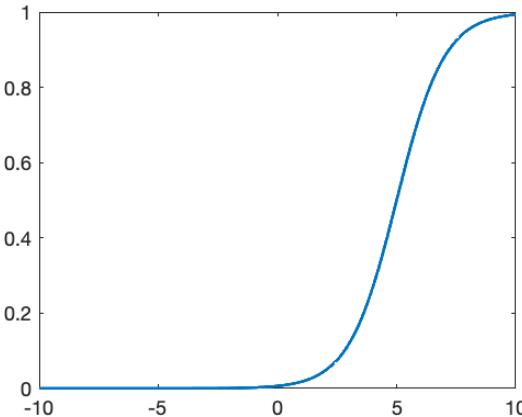


Logistic function: Effect of w

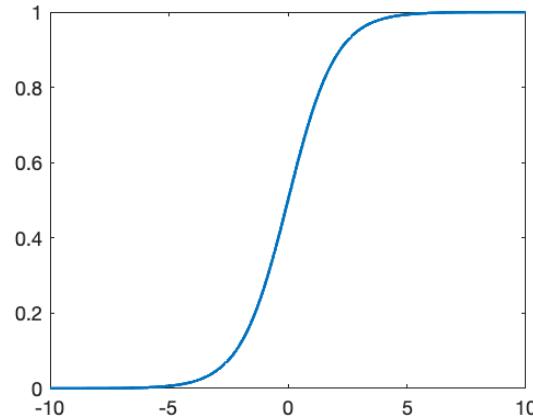
- With a 1D input x , applying the logistic function to the output of a linear model gives a prediction

$$\hat{y} = \frac{1}{1 + \exp(-w^{(1)}x - w^{(0)})}$$

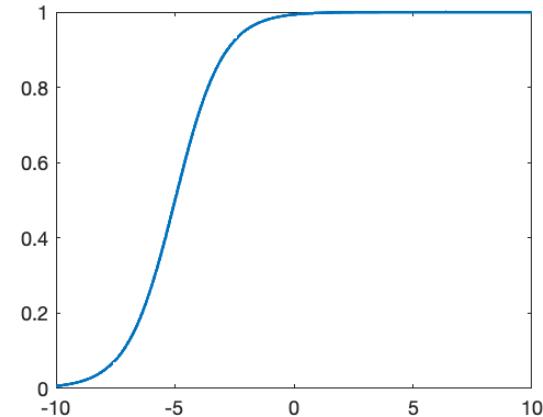
- Effect of $w^{(0)}$



$$w^{(0)} = -5, w^{(1)} = 1$$



$$w^{(0)} = 0, w^{(1)} = 1$$



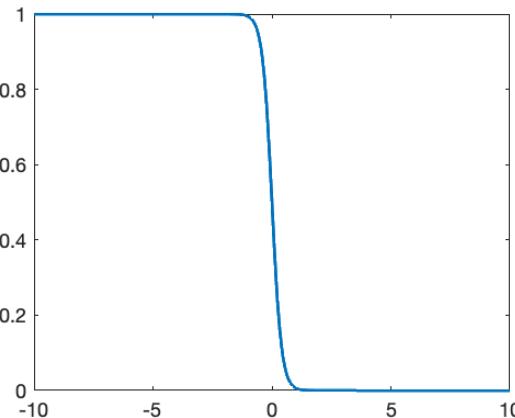
$$w^{(0)} = 5, w^{(1)} = 1$$

Logistic function: Effect of w

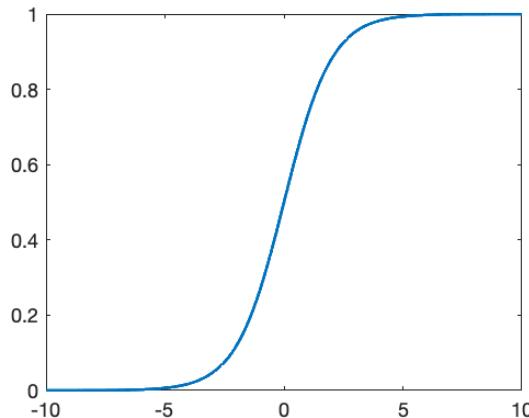
- With a 1D input x , applying the logistic function to the output of a linear model gives a prediction

$$\hat{y} = \frac{1}{1 + \exp(-w^{(1)}x - w^{(0)})}$$

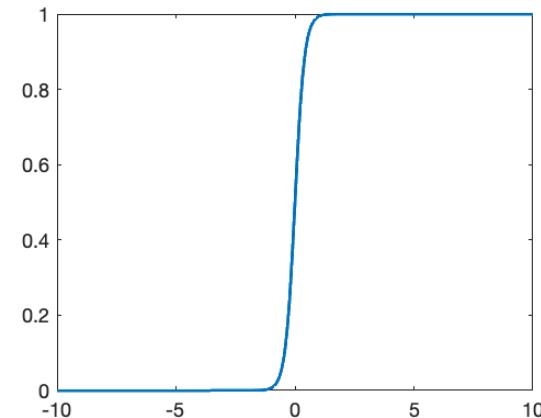
- Effect of $w^{(1)}$



$$w^{(0)} = 0, w^{(1)} = -5$$



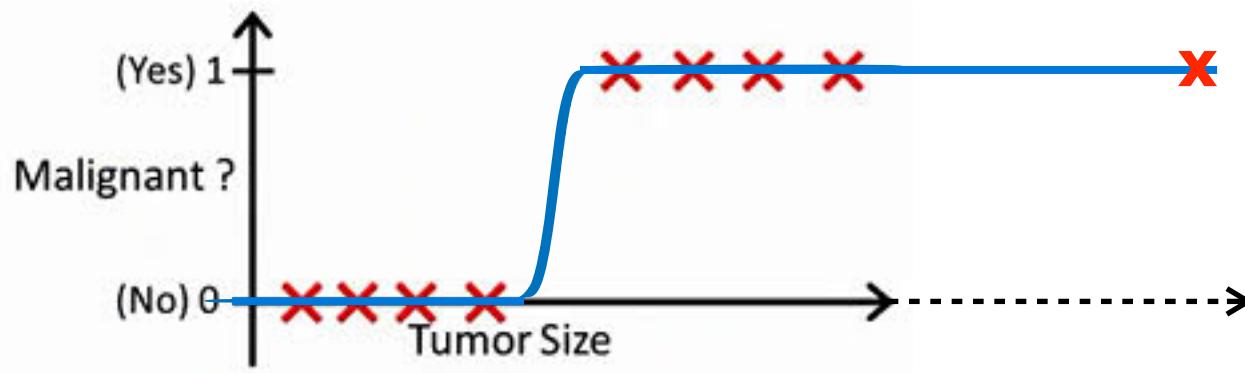
$$w^{(0)} = 0, w^{(1)} = 1$$



$$w^{(0)} = 0, w^{(1)} = 5$$

Fitting the tumor data

- Applying a logistic function to the output of the linear model can fit the data much better
- Even in the presence of outliers



Probabilistic interpretation

- In the binary case, with D dimensional inputs, we can write the prediction of the model as

$$\hat{y} = \sigma(\mathbf{w}^T \mathbf{x}) = \frac{1}{1 + \exp(-\mathbf{w}^T \mathbf{x})}$$

- Such a prediction can be interpreted as the probability that \mathbf{x} belongs to the positive class (or as a score for the positive class)
- The probability to belong to the negative class (or score for the negative class) is then computed as $1 - \hat{y}$

Logistic regression

- The model whose prediction is defined as

$$\hat{y} = \sigma(\mathbf{w}^T \mathbf{x}) = \frac{1}{1 + \exp(-\mathbf{w}^T \mathbf{x})}$$

is called *logistic regression*

- Note that this name is slightly misleading, as this is truly a **classification model**, not a regression one
- At its heart, logistic regression still makes use of the linear model, but followed by a nonlinearity to better encode the underlying binary classification problem

Logistic regression: Training

- Given N training samples, to learn the parameters \mathbf{w} of the logistic regression model, we need to define a loss function
- Instead of using the L^2 loss as in the least-square case, this is achieved by making use of the probabilistic interpretation
 - Because knowledge of probabilities is not required for this course, the derivation of the loss function that follows will be a bit hand-wavy.
 - Nevertheless, this loss remains intuitive: The loss is such that it aims to find the parameters that maximize the probability (or score) of the correct class for each training sample

Logistic regression: Training

- Specifically, the logistic regression loss is derived by first writing the following likelihood (think of it as a value representing how well the true labels are predicted given \mathbf{w})

$$p(\mathbf{y} | \mathbf{w}) = \prod_{i=1}^N \hat{y}_i^{y_i} \cdot (1 - \hat{y}_i)^{1-y_i}$$

- Because a product of terms between 0 and 1 can be unstable to optimize, we take the logarithm of this likelihood, and take its negative to define an empirical risk to minimize, written as

$$R(\mathbf{w}) = - \sum_{i=1}^N (y_i \ln(\hat{y}_i) + (1 - y_i) \ln(1 - \hat{y}_i))$$

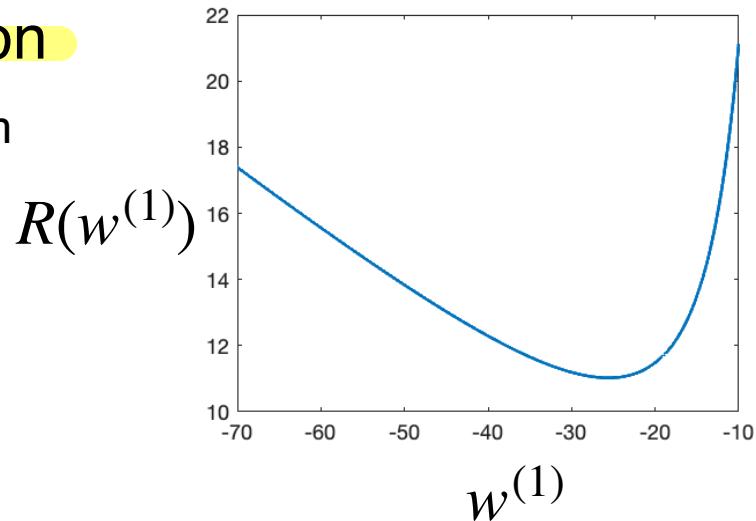
- This loss function is known as the *cross-entropy*

Logistic regression: Training

- Since the true y_i is either 0 or 1, the cross-entropy can be re-written as

$$R(\mathbf{w}) = - \sum_{i \in \text{positive samples}} \ln(\hat{y}_i) - \sum_{i \in \text{negative samples}} \ln(1 - \hat{y}_i)$$

- The cross-entropy is a convex function
 - However, its minimization has no closed-form solution
- We optimize it via gradient descent
 - Let us now review this algorithm



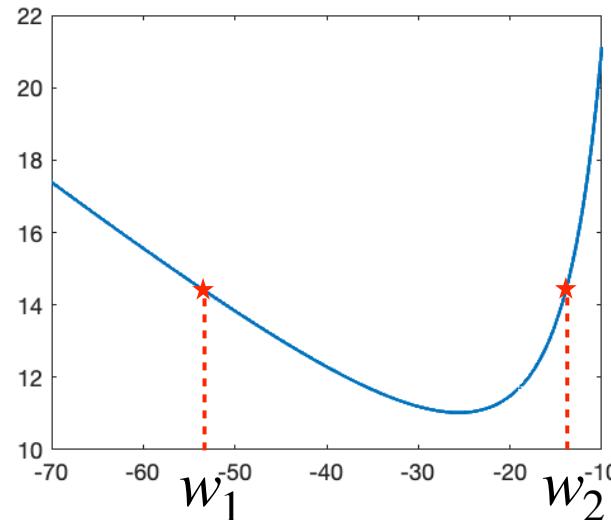
Interlude

Iterative, Gradient-based Optimization

Minimizing a function

- With 1D functions for which the solution cannot be computed in closed form, one can nonetheless use the derivative
 - Recall that it represents the slope at a given point, and thus indicates in which direction to move to reach a lower function value

At w_1 , the slope is negative. However, one should move in the positive direction ($\Delta w > 0$) to go towards the minimum



At w_2 , the slope is positive. However, one should move in the negative direction ($\Delta w < 0$) to go towards the minimum

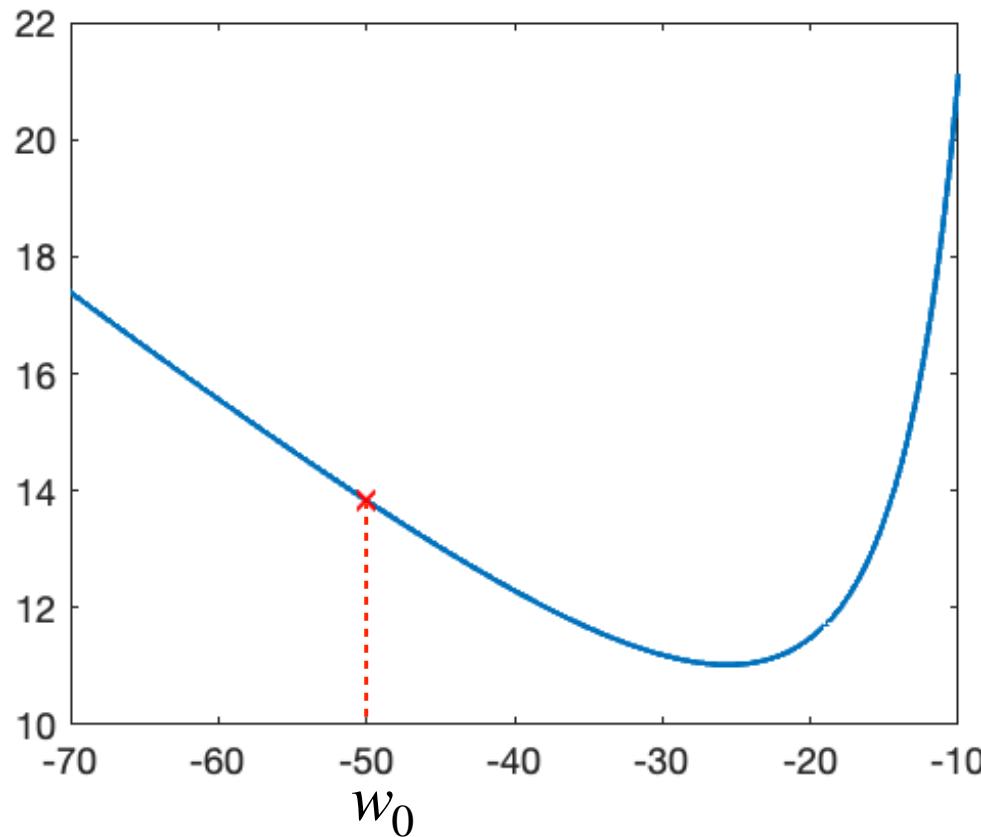
- This indicates that one should move in the direction opposite to the derivative for minimization

Minimizing a function

- Algorithm:
 1. Initialize w_0 (e.g., randomly)
 2. While not converged
 - 2.1. Update $w_k \leftarrow w_{k-1} - \eta \frac{dR(w_{k-1})}{dw}$
- η defines the step size of each iteration
 - In ML, it is often referred to as *learning rate*

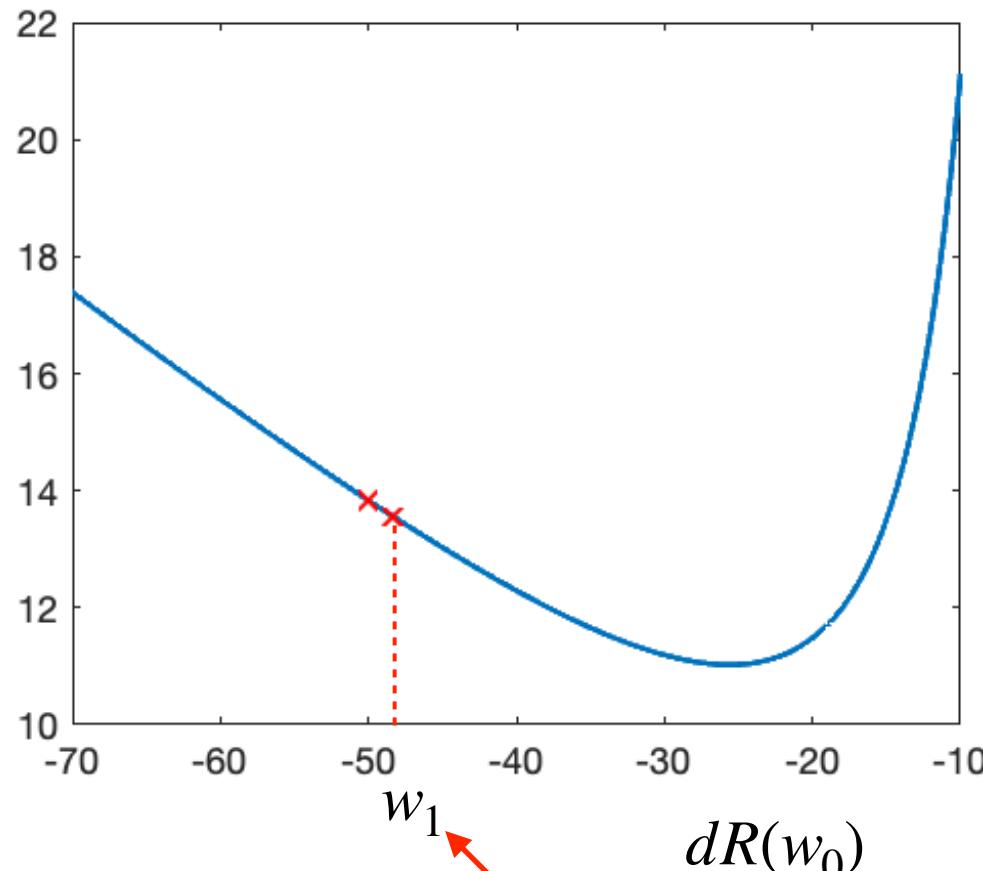
Minimizing a convex function

- Example:



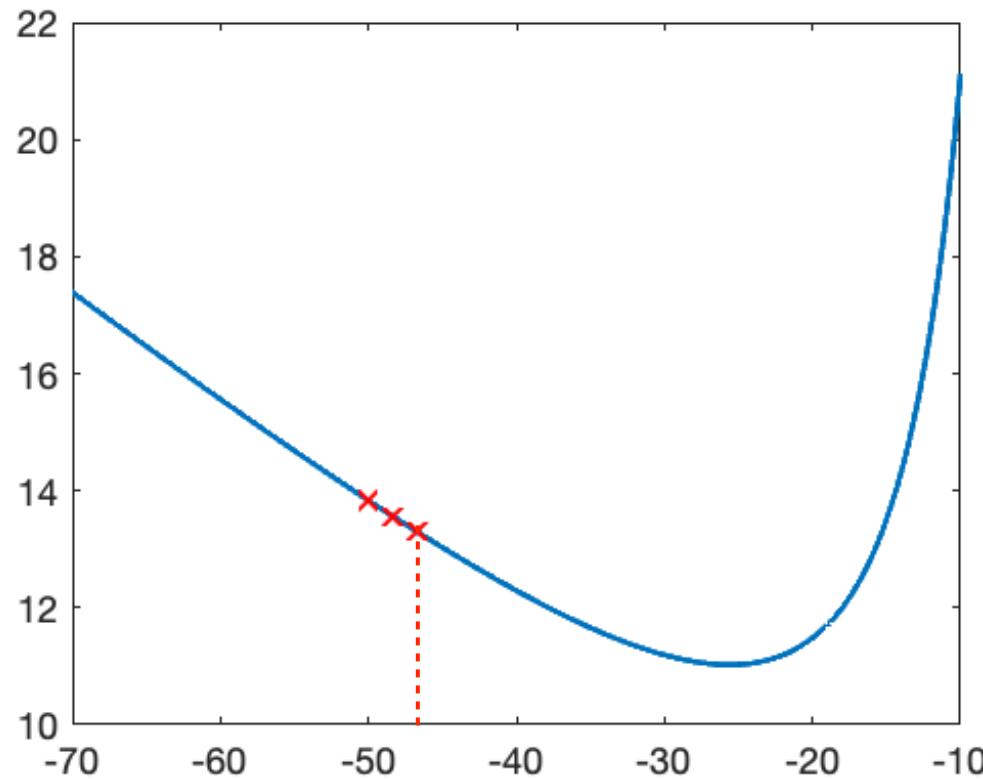
Minimizing a convex function

- Example:



Minimizing a convex function

- Example:

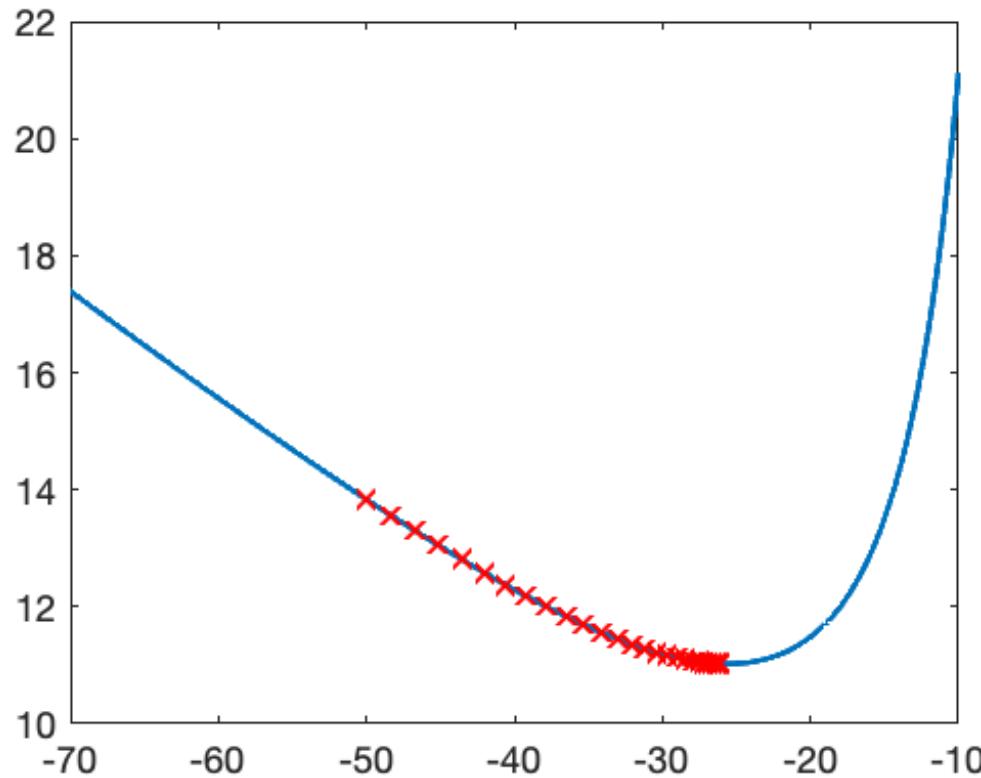


$$w_1 - \eta \frac{dR(w_1)}{dw}$$

40

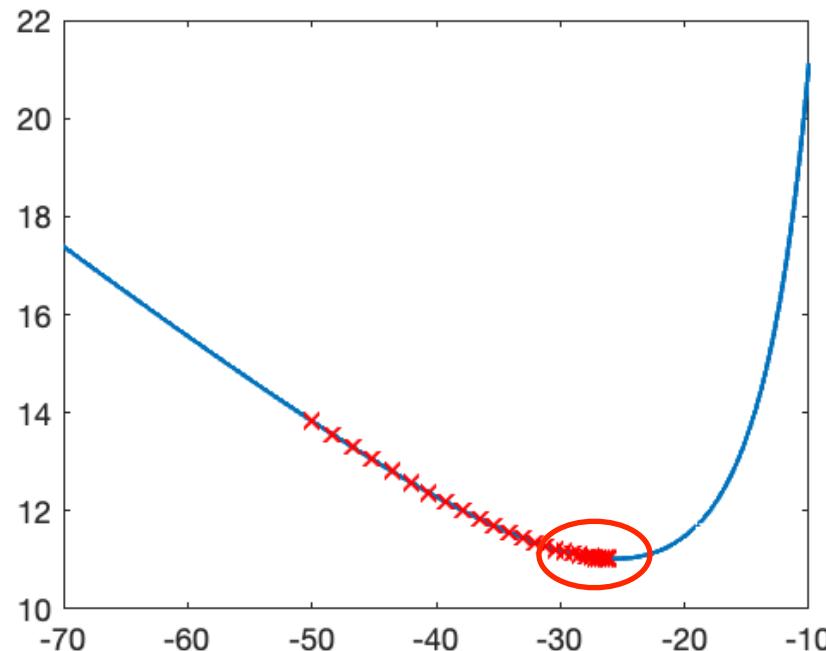
Minimizing a convex function

- Example:



Minimizing a convex function

- Convergence:
 - Because the derivative becomes smaller $\left(\frac{dR(w)}{dw} \rightarrow 0 \right)$, the update Δw becomes smaller, and the algorithm converges



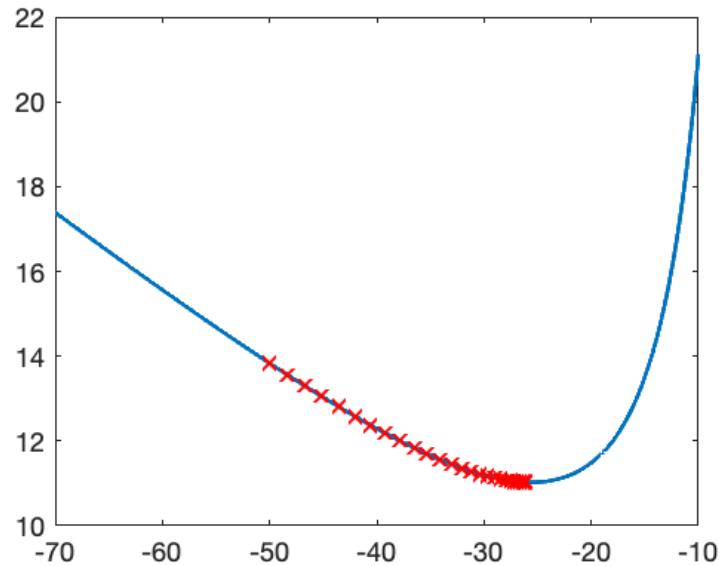
Minimizing a function

- Potential convergence criteria:
 - Change in function values less than threshold: $|R(w_{k-1}) - R(w_k)| < \delta_R$
 - Change in parameter value less than threshold: $|w_{k-1} - w_k| < \delta_w$
 - Maximum number of iterations reached
 - No guarantee to have reached the minimum

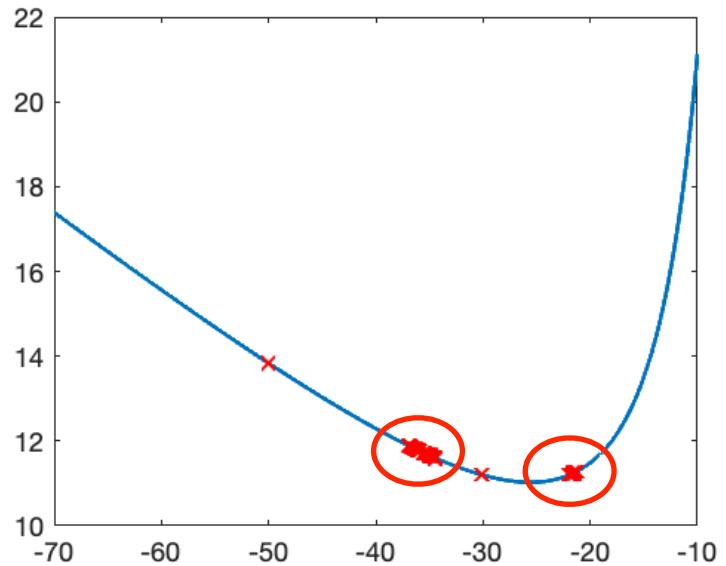
Minimizing a convex function

- Influence of η :

$$\eta = 10$$



$$\eta = 120$$

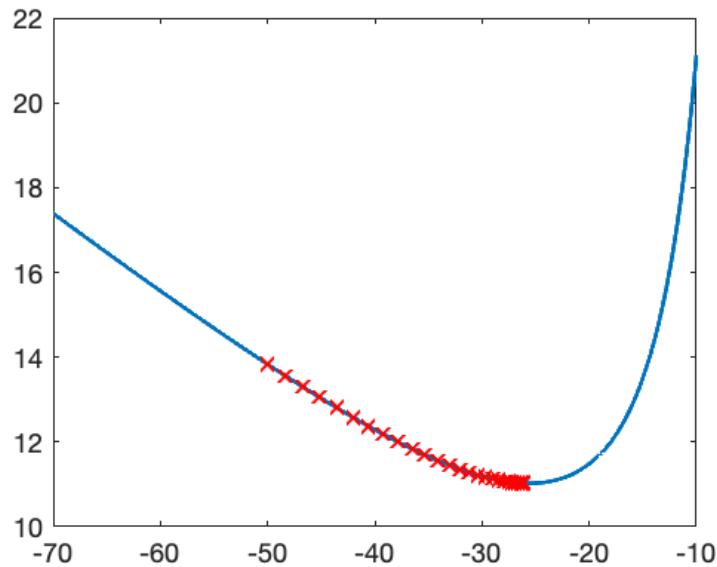


The steps are too large and the algorithm starts jumping between these two points

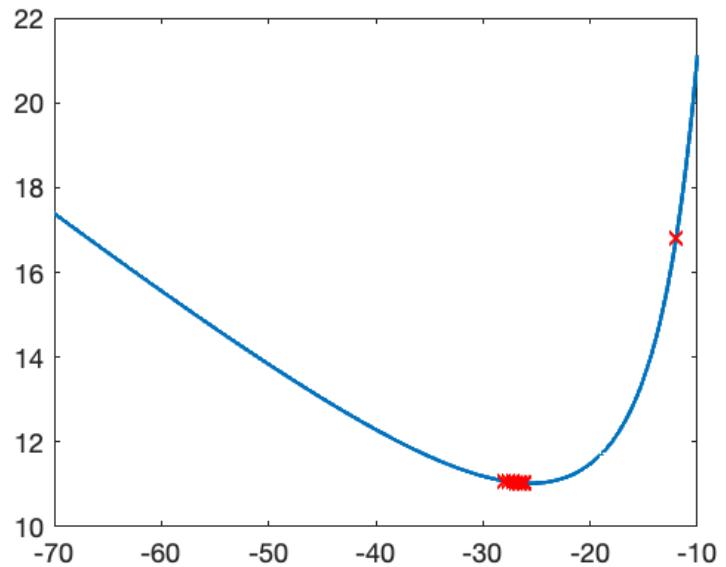
Minimizing a convex function

- Influence of the initial w_0 :

$$w_0 = -50$$



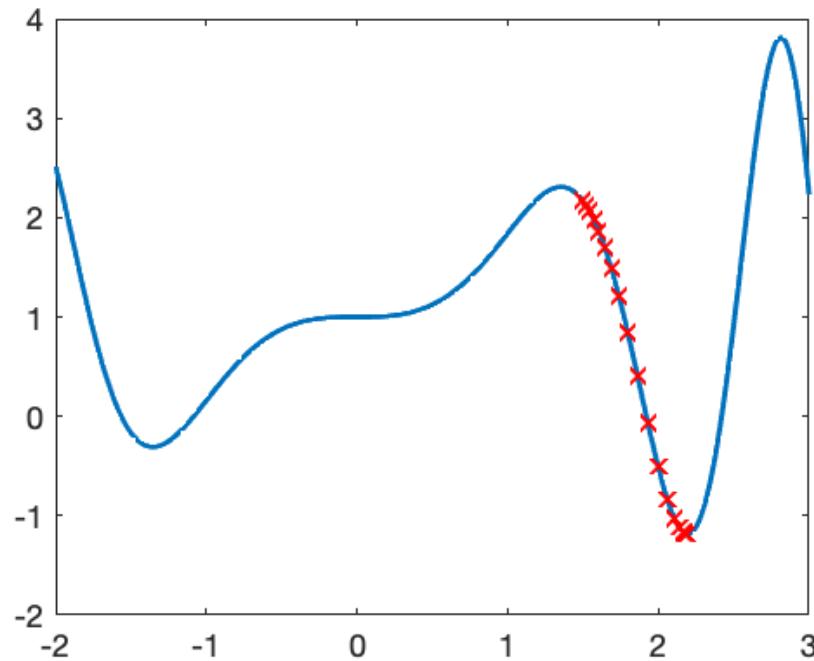
$$w_0 = -12$$



Still converges to the same point, but faster

Minimizing a non-convex function

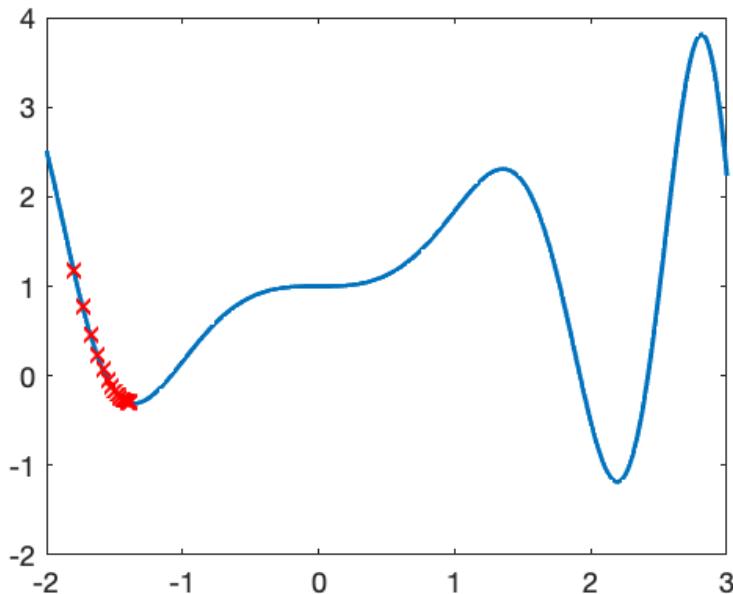
- The same iterative algorithm as before can be used



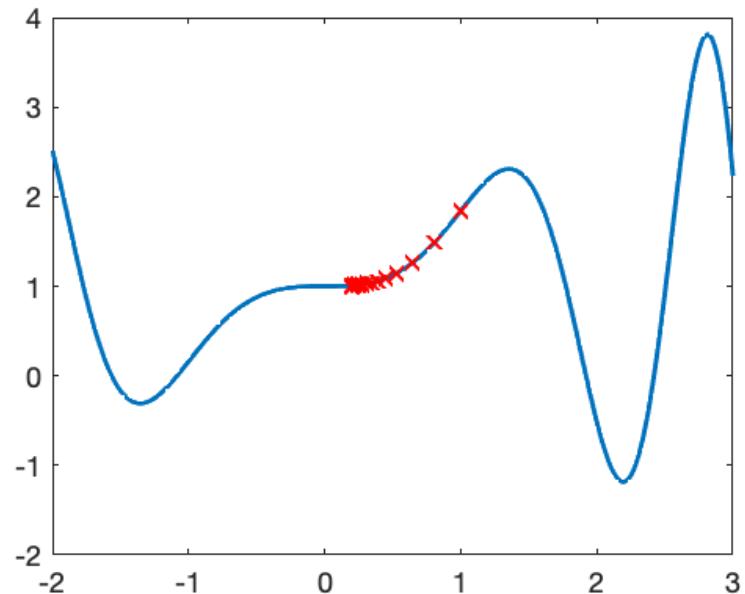
Minimizing a non-convex function

- However, different initializations yield different results
 - Before, $w_0 = 1.5$ and optimization reached the global minimum

$w_0 = -1.8$
Local minimum



$w_0 = 1$
Saddle point

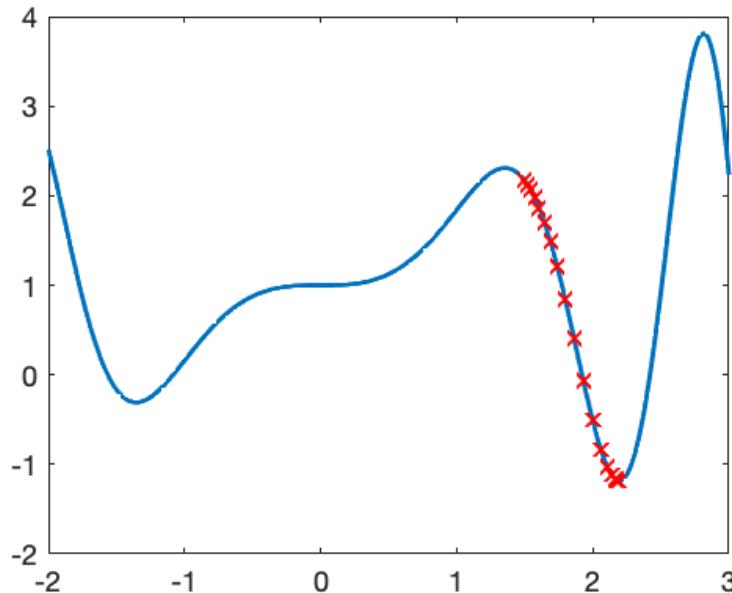


Minimizing a non-convex function

- And the results remain sensitive to the learning rate η
 - Note that η is problem-dependent and the values used here are very different from those used in the convex example before

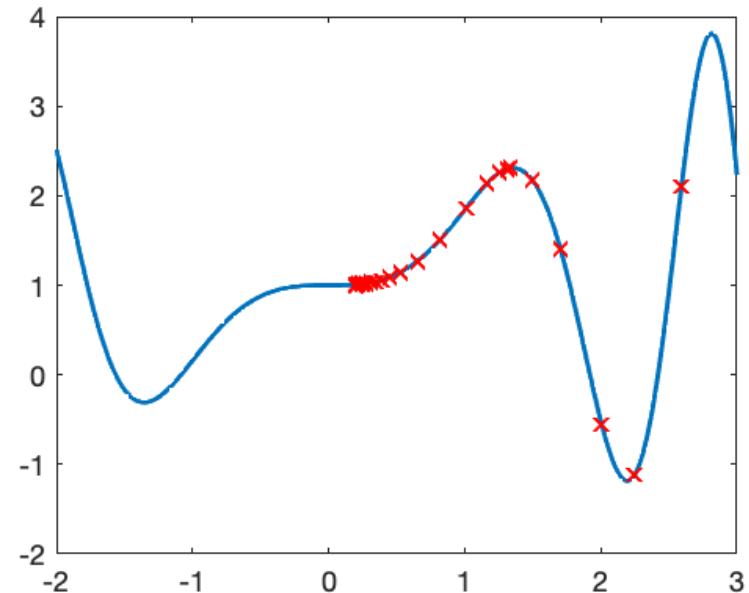
$$\eta = 0.01, w_0 = 1.5$$

Global minimum



$$\eta = 0.1, w_0 = 1.5$$

Saddle point



Recap: Gradient

- In practice, we have more than one parameter, i.e., $\mathbf{w} \in \mathbb{R}^D$, and we thus use the function gradient:

$$\nabla R(\mathbf{w}) = \begin{bmatrix} \frac{\partial R}{\partial w^{(1)}} \\ \frac{\partial R}{\partial w^{(2)}} \\ \vdots \\ \frac{\partial R}{\partial w^{(D)}} \end{bmatrix} \in \mathbb{R}^D$$

where $\frac{\partial R}{\partial w^{(j)}}$ denotes the partial derivative w.r.t. variable $w^{(j)}$

Recap: Properties of gradient

- The gradient at a point \mathbf{w} has the direction of greatest increase of the function at \mathbf{w}
- Its magnitude is the rate of increase in that direction

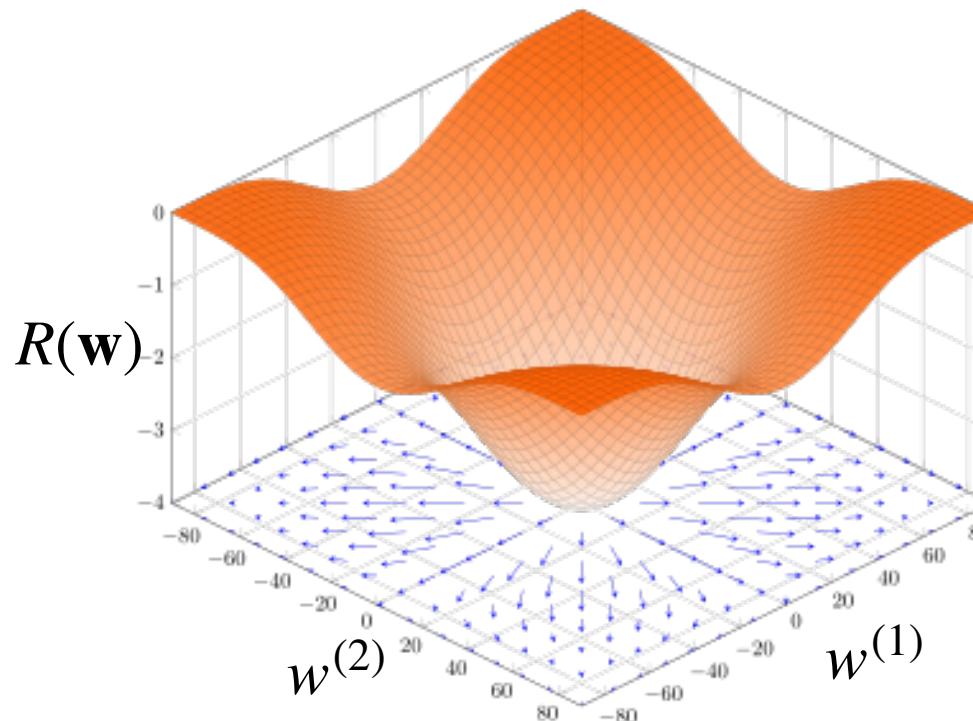


Figure from Wikipedia

Minimizing multi-variate functions

- As in the 1D case, to minimize a function that has no closed-form solution, we can exploit the fact that the gradient points in the direction of maximum function increase
- To decrease the function, we therefore want to move in the direction opposite to the gradient
- This yields an iterative algorithm that generalizes the one-variable one we saw before to multiple variables
 - This is called *gradient descent* (or steepest descent)

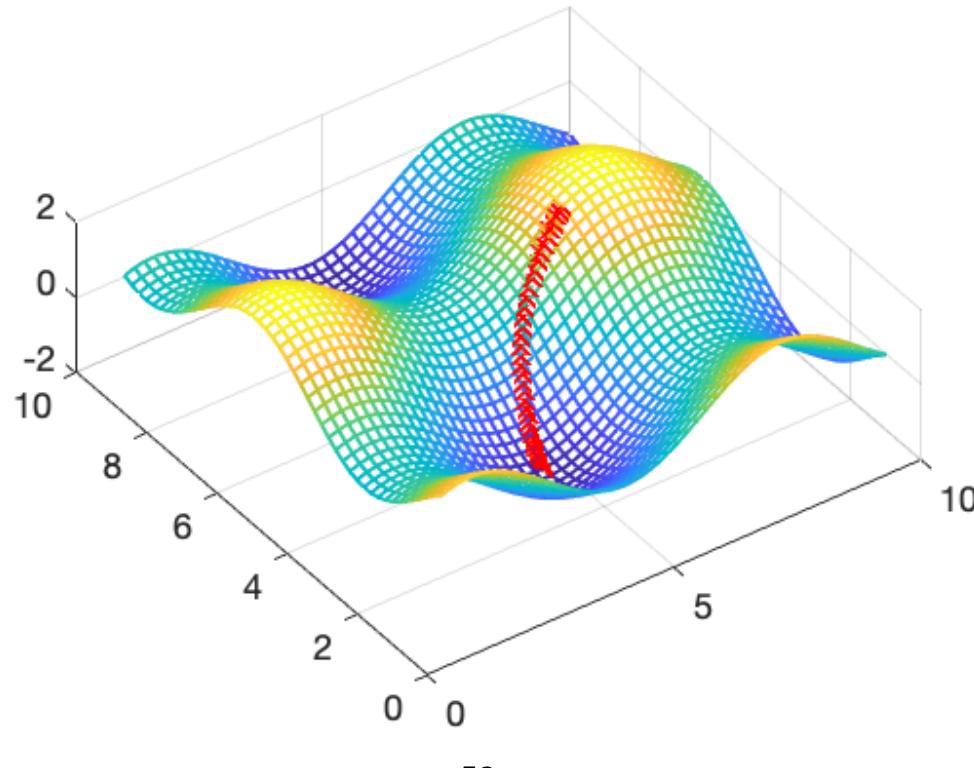
Gradient descent

- Algorithm:
 1. Initialize \mathbf{w}_0 (e.g., randomly)
 2. While not converged
 - 2.1. Update $\mathbf{w}_k \leftarrow \mathbf{w}_{k-1} - \eta \nabla R(\mathbf{w}_{k-1})$
- The learning rate η has the same effect as in the 1D case
- Convergence can again be measured by
 - Thresholding the change in function value
 - Thresholding the change in parameters, e.g., $\|\mathbf{w}_{k-1} - \mathbf{w}_k\| < \delta_w$

Minimizing a non-convex function

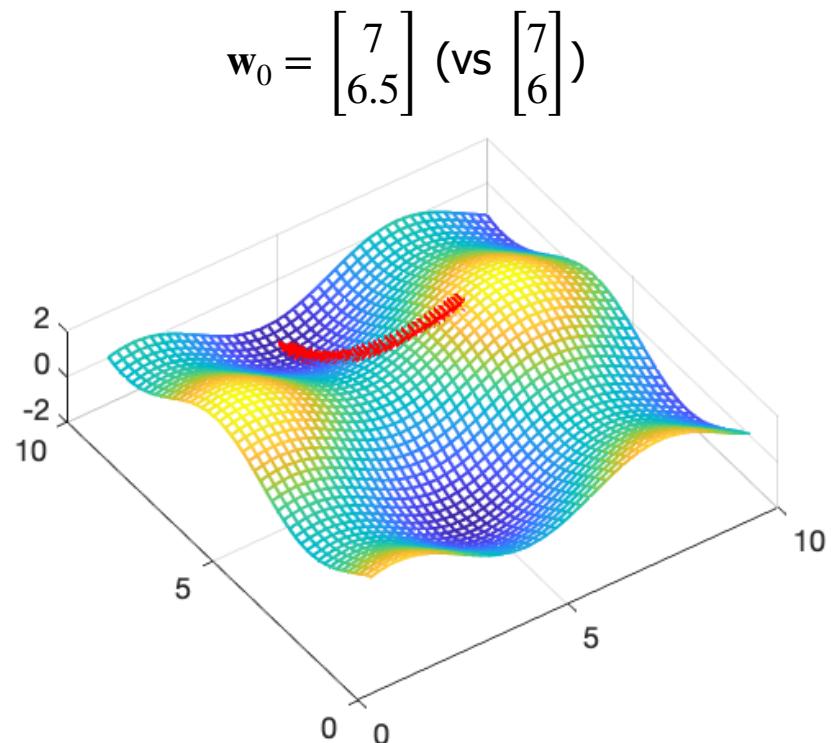
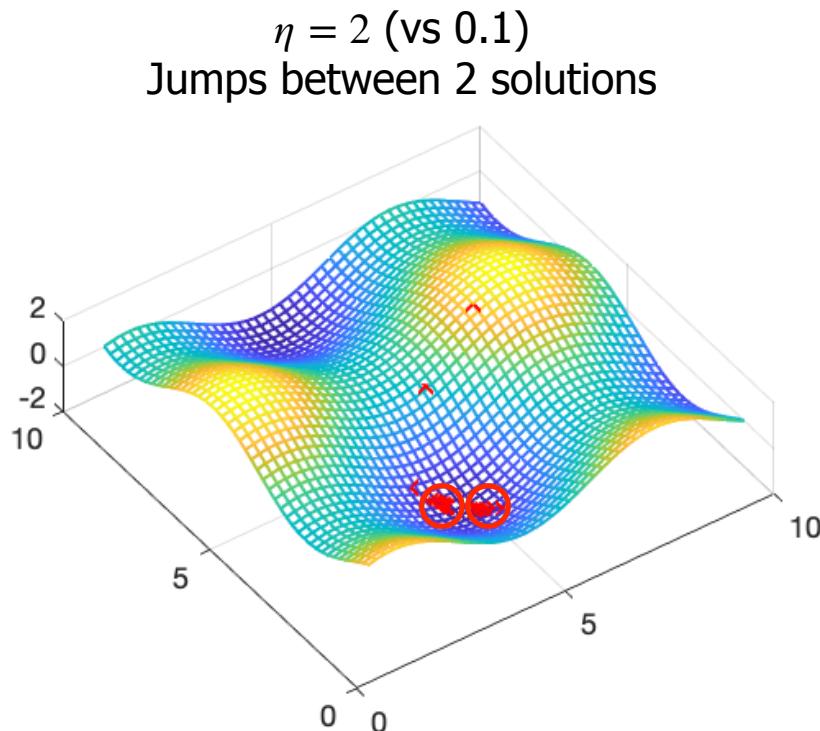
- Example: Sine- and cosine-based function of 2 variables

$$R(\mathbf{w}) = \sin(w^{(1)}) + \cos(w^{(2)}) \quad \nabla R(\mathbf{w}) = \begin{bmatrix} \cos(w^{(1)}) \\ -\sin(w^{(2)}) \end{bmatrix}$$



Gradient descent

- The same issues as in the 1D case can arise:
 - Different learning rates can yield different results
 - Different initializations can yield different results



End of the interlude

Back to Logistic Regression

Logistic regression: Training

- With binary ground-truth labels y_i , the cross-entropy can be written as

$$R(\mathbf{w}) = - \sum_{i \in \text{positive samples}} \ln(\hat{y}_i(\mathbf{w})) - \sum_{i \in \text{negative samples}} \ln(1 - \hat{y}_i(\mathbf{w}))$$

- To minimize it via gradient descent, we need to compute its gradient w.r.t. \mathbf{w}

Logistic regression: Gradient

- To compute the gradient w.r.t. \mathbf{w} , let us first look at the derivative of the logistic function

$$\sigma(a) = \frac{1}{1 + \exp(-a)}$$

- One can derive that

$$\frac{d\sigma(a)}{da} = \sigma(a) \cdot (1 - \sigma(a))$$

Logistic regression: Gradient

- Then, looking at the positive samples and following the chain rule, we have

$$\begin{aligned}\nabla \left(- \sum_{i \in \text{positive}} \ln \sigma(\mathbf{w}^T \mathbf{x}_i) \right) &= - \sum_{i \in \text{positive}} \frac{\partial \ln \sigma}{\partial \sigma} \frac{\partial \sigma(a_i)}{\partial a_i} \frac{\partial a_i}{\partial \mathbf{w}} \\ &= - \sum_{i \in \text{positive}} \frac{1}{\sigma(\mathbf{w}^T \mathbf{x}_i)} \sigma(\mathbf{w}^T \mathbf{x}_i) (1 - \sigma(\mathbf{w}^T \mathbf{x}_i)) \mathbf{x}_i \\ &= - \sum_{i \in \text{positive}} (1 - \hat{y}_i) \mathbf{x}_i = \sum_{i \in \text{positive}} (\hat{y}_i - 1) \mathbf{x}_i\end{aligned}$$

- Similarly, for the negative examples, we have

$$\nabla \left(- \sum_{i \in \text{negative}} \ln(1 - \sigma(\mathbf{w}^T \mathbf{x})) \right) = \sum_{i \in \text{negative}} \hat{y}_i \mathbf{x}_i$$

Logistic regression: Gradient

- Finally, this can be put back together to obtain

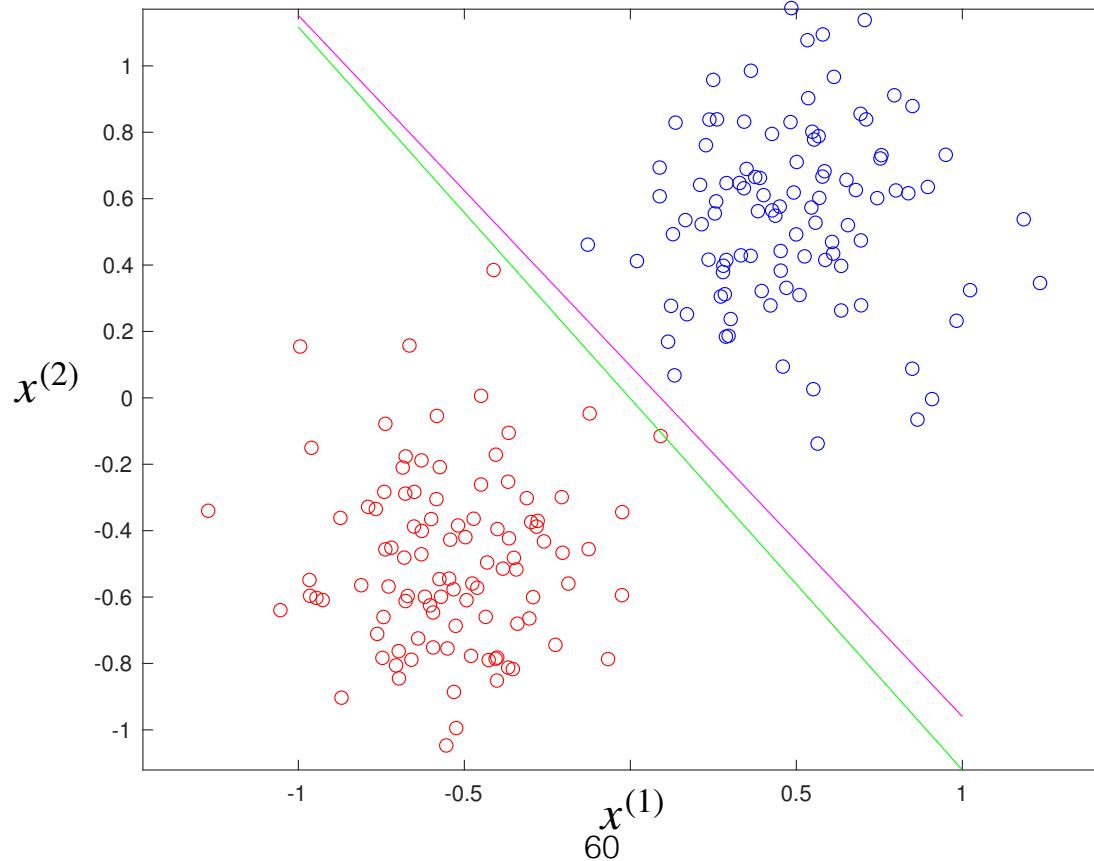
$$\nabla R(\mathbf{w}) = \sum_{i=1}^N (\hat{y}_i - y_i) \mathbf{x}_i$$

where y_i is the true label for sample i , i.e., 1 for positive samples and 0 for negative samples

- Intuitively, the gradient is given by the error of the current prediction $(\hat{y}_i - y_i)$, multiplied by the input

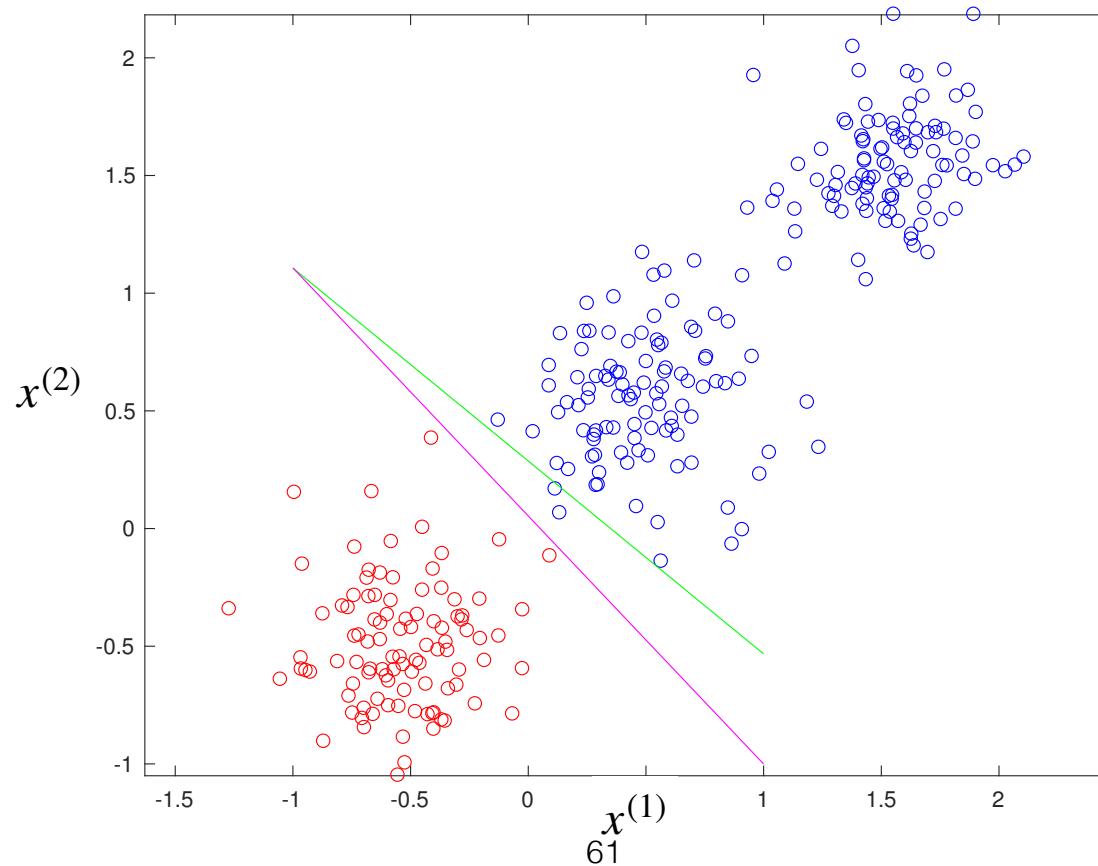
Logistic regression

- Example: 2D inputs, 2 classes
 - Least-square classification in green, logistic regression in magenta



Logistic regression

- Now, with additional samples, the boundary remains correct
 - Least-square classification in green, logistic regression in magenta

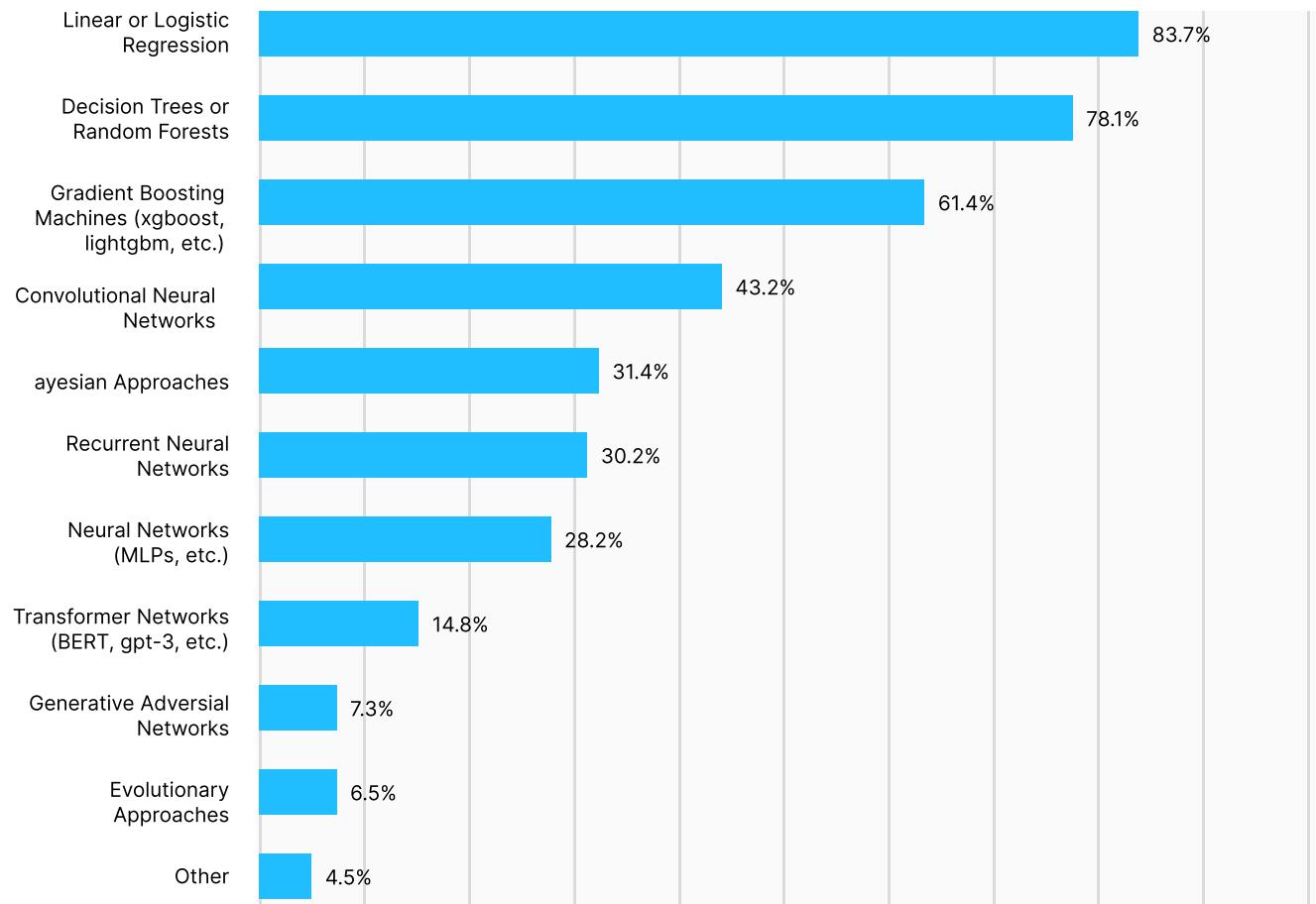


Logistic regression: Demo

- <https://playground.tensorflow.org/#activation=sigmoid&batchSize=10&dataset=gauss®Dataset=reg-plane&learningRate=0.03®ularizationRate=0&noise=0&networkShape=&seed=0.92907&showTestData=false&discretize=false&percTrainData=50&x=true&y=true&xTimesY=false&xSquared=false&ySquared=false&cosX=false&sinX=false&cosY=false&sinY=false&collectStats=false&problem=classification&initZero=false&hideText=false>

Logistic regression in practice

- According to a 2020 survey among data scientist, the answer to the question “What kind of data science methods are used at work” gave the statistics



Exercises

- Describe two main differences between linear regression and logistic regression.
- Describe one similarity between linear regression and logistic regression.

Recap: Model evaluation

- Once an ML model is trained, one would typically understand how well it performs on unseen test data
 - At this stage, the parameters of the model are fixed
 - Recall that the training and testing data must be separated!
- During this evaluation, one compares the predictions of the model with the true annotations of the test data
 - In contrast to the training stage, the model parameters are *not* updated
 - The evaluation metric may directly be the loss function, but may also differ from it

Classification evaluation

- Confusion matrix: Binary case (e.g., spam vs non-spam email)

True class → Hypothesized class V	Pos	Neg
Yes	TP	FP
No	FN	TN
	$P = TP + FN$	$N = FP + TN$

TP: True positive (number of samples correctly classified as positive)

FP: False positive (number of samples incorrectly classified as positive)

TN: True negative (number of samples correctly classified as negative)

FN: False negative (number of samples incorrectly classified as negative)

P: Total number of positive samples

N: Total number of negative samples

Classification evaluation

- Confusion matrix: Binary case
 - Numerical example for a dataset with 500 positive and 500 negative samples

True class →	Pos	Neg
Yes	200	100
No	300	400
	P=500	N=500

Classification metrics

- **Accuracy:** Percentage of correctly classified samples

$$Acc = \frac{TP + TN}{P + N}$$

- Exercise: Compute the accuracy for the following data

True class →	Pos	Neg
Yes	200	100
No	300	400
	P=500	N=500

What's wrong with accuracy?

True class →	Pos	Neg
Yes	200	100
No	300	400
	P=500	N=500

True class →	Pos	Neg
Yes	400	300
No	100	200
	P=500	N=500

- The two classifiers (matrices) above yield the same accuracy
- But their behavior is very different
 - Left: Weak positive recognition rate, but strong negative recognition rate
 - Right: Strong positive recognition rate, but weak negative recognition rate

Classification metrics

- **Precision:** Percentage of samples classified as positives that are truly positives

$$Precision = \frac{TP}{TP + FP}$$

- Exercise: Compute the precision for this data

True class →	Pos	Neg
Yes	200	100
No	300	400
	P=500	N=500

Classification metrics

- **Recall:** Percentage of positives samples that are correctly classified as positives

$$Recall = \frac{TP}{P}$$

- Exercise: Compute the recall for this data

True class →	Pos	Neg
Yes	200	100
No	300	400
	P=500	N=500

What's wrong with precision/recall?

True class →	Pos	Neg
Yes	200	100
No	300	400
	P=500	N=500

True class →	Pos	Neg
Yes	200	100
No	300	0
	P=500	N=100

- The two classifiers (matrices) above yield the same precision and recall (0.667 and 0.4) (The datasets are different)
- But their behavior is very different
 - Same positive recognition rate
 - Very different negative recognition rates (strong on left, but nil on right)
- Accuracy would clearly show this

Classification metrics

- Precision: Percentage of samples classified as positives that are truly positives

$$Precision = \frac{TP}{TP + FP}$$

- Exercise: Compute the precision for this data

True class →	Pos	Neg
Yes	200	100
No	300	400
	P=500	N=500

Classification metrics

- Recall: Percentage of positives samples that are correctly classified as positives

$$Recall = \frac{TP}{P}$$

- Exercise: Compute the recall for this data

True class →	Pos	Neg
Yes	200	100
No	300	400
	P=500	N=500

ROC curve

- Many classifiers output a score/confidence for their predictions (e.g., \hat{y} is a continuous value between 0 and 1 in logistic regression)
- The decision between positive and negative can be computed by thresholding this score:

$$\text{label}_i = 1 \text{ if } \hat{y}_i \geq \tau$$

where τ is a given threshold

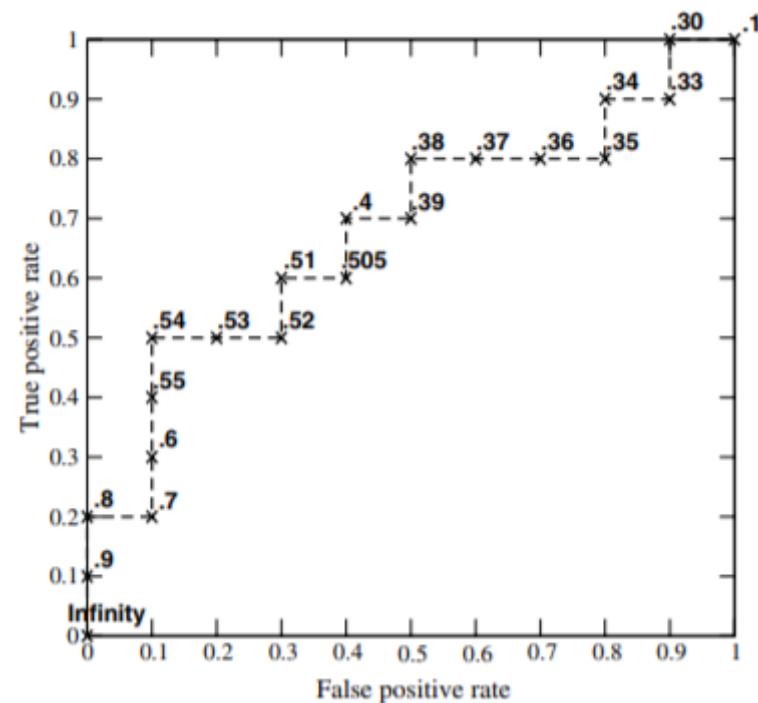
- The Receiver Operating Characteristic (ROC) curve plots the true positive rate (recall) as a function of the false positive rate, obtained by varying the score threshold

Constructing an ROC curve

- Example with 10 positive and 10 negative samples

Inst#	Class	Score	Inst#	Class	Score
1	p	.9	11	p	.4
2	p	.8	12	n	.39
3	n	.7	13	p	.38
4	p	.6	14	n	.37
5	p	.55	15	n	.36
6	p	.54	16	n	.35
7	n	.53	17	p	.34
8	n	.52	18	n	.33
9	p	.51	19	p	.30
10	n	.505	20	n	.1

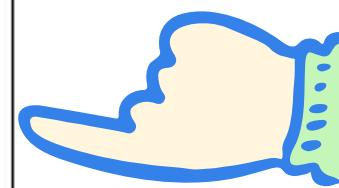
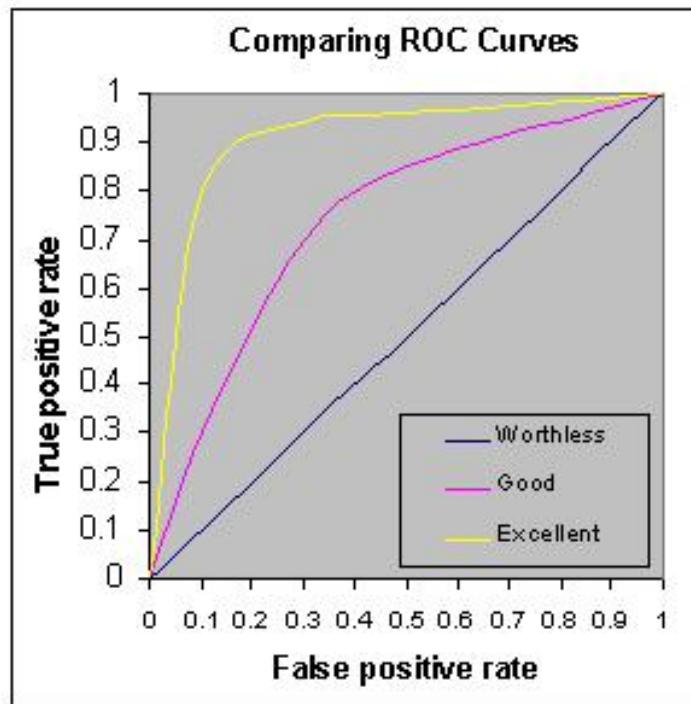
Samples sorted by score
(Class denotes the true label)



ROC curve

Classification metrics

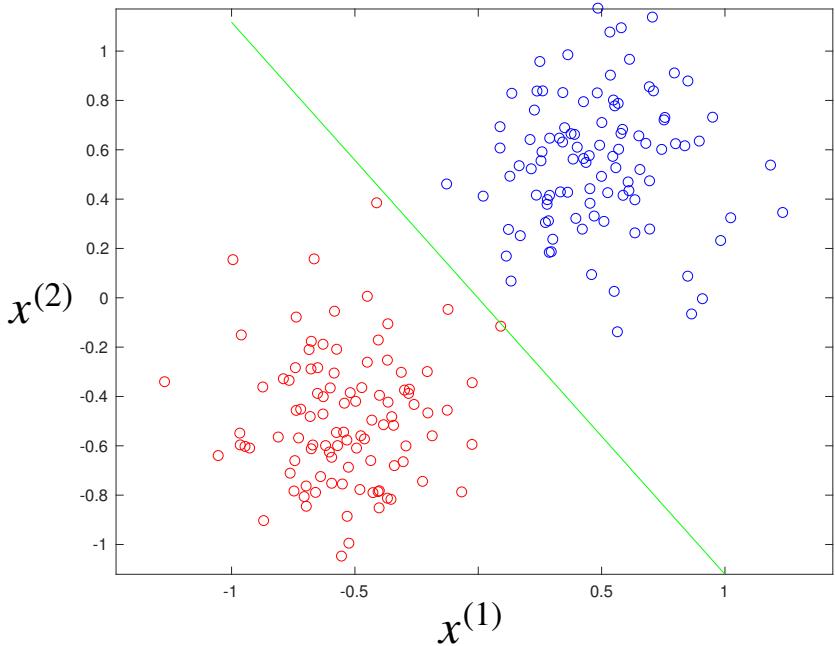
- ROC curves



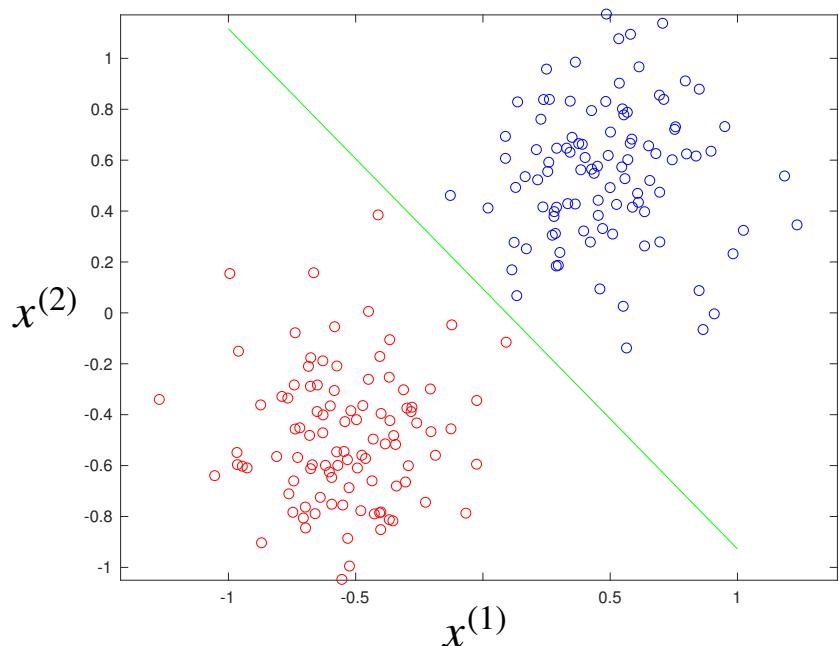
- The Area Under the ROC Curve (AUC) can act as a metric
 - The maximum AUC is 1
 - Random predictions give an AUC of 0.5

Decision boundaries

- Different classifiers have different decision boundaries



Least-square classification

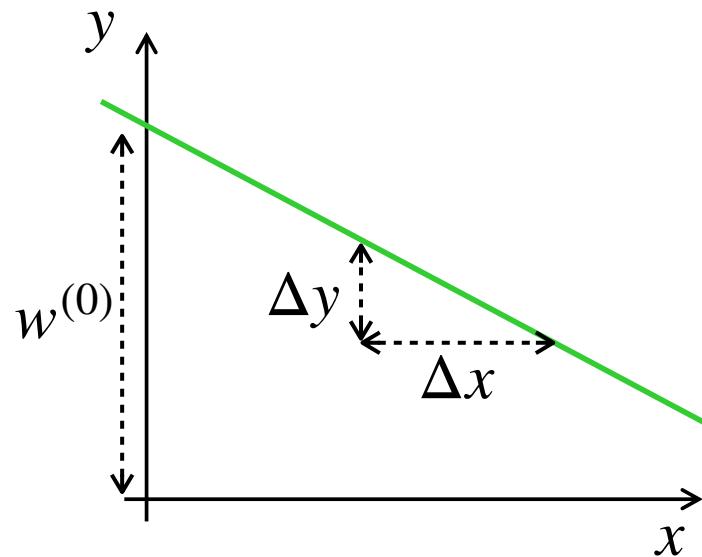


Logistic regression

- Can we define what a good decision boundary is?
 - We will discuss this further next week

Lecture 4: Linear Models for Classification (Part 2)

Recap: A simple parametric model: The line



- Defined by 2 parameters
 - The y -intercept $w^{(0)}$
 - The slope $w^{(1)} = \frac{\Delta y}{\Delta x}$
- Mathematically, a line is expressed as

$$y = w^{(1)}x + w^{(0)}$$

Recap: Hyperplane

- This can be generalized to higher dimensions
- In dimension D , we can write

$$y = w^{(0)} + w^{(1)}x^{(1)} + w^{(2)}x^{(2)} + \dots + w^{(D)}x^{(D)} = \mathbf{w}^T \begin{bmatrix} x^{(1)} \\ x^{(2)} \\ \vdots \\ x^{(D)} \\ 1 \end{bmatrix}$$

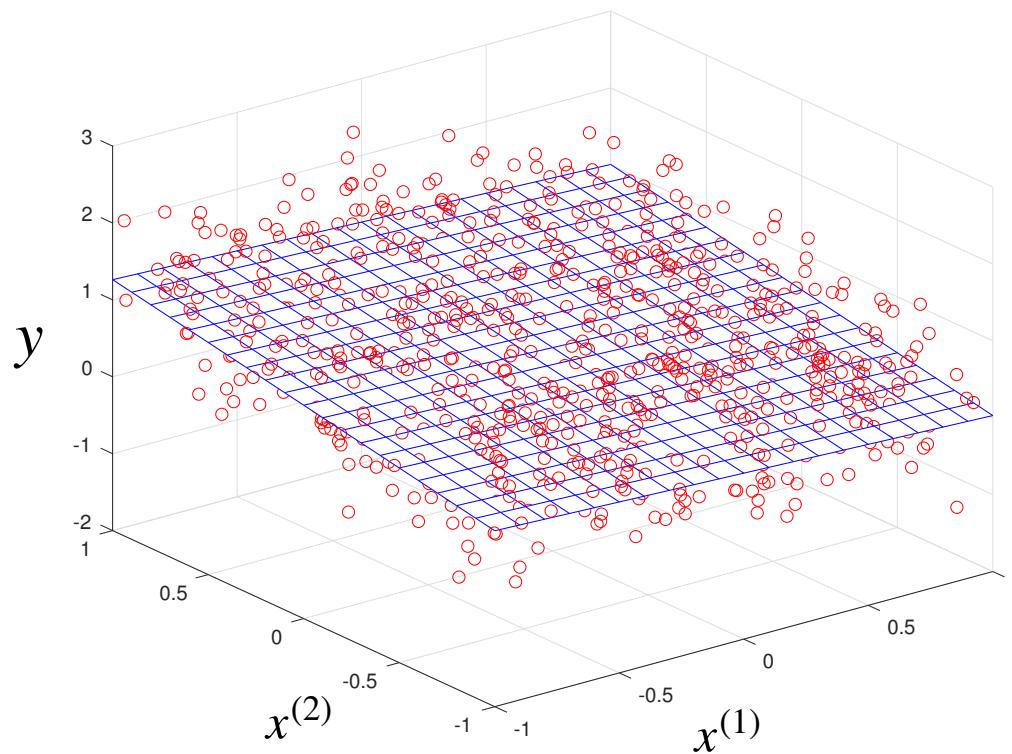
- Ultimately, whatever the dimension, we can write

$$y = \mathbf{w}^T \mathbf{x}$$

with $\mathbf{x} \in \mathbb{R}^{D+1}$, where the extra dimension contains a 1 to account for $w^{(0)}$

Recap: Linear regression

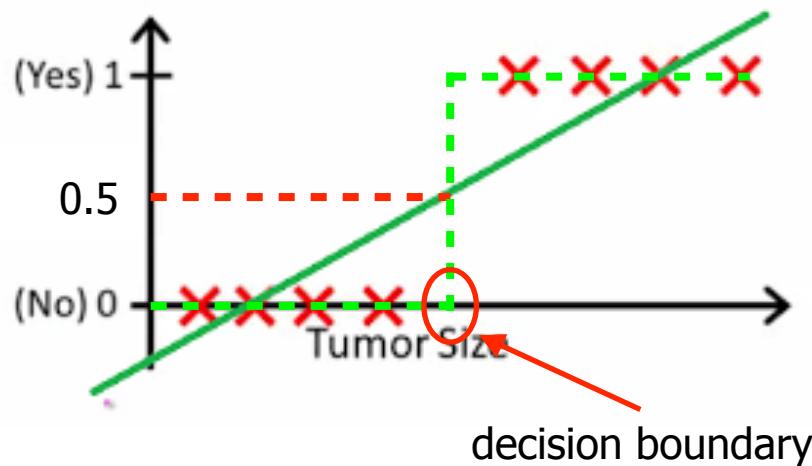
- Given N noisy pairs $\{(\mathbf{x}_i, y_i)\}$, where $\mathbf{x}_i \in \mathbb{R}^D$ (below, $D = 2$), find the parameters \mathbf{w}^* that best fits these observations



Recap: Binary classification as regression

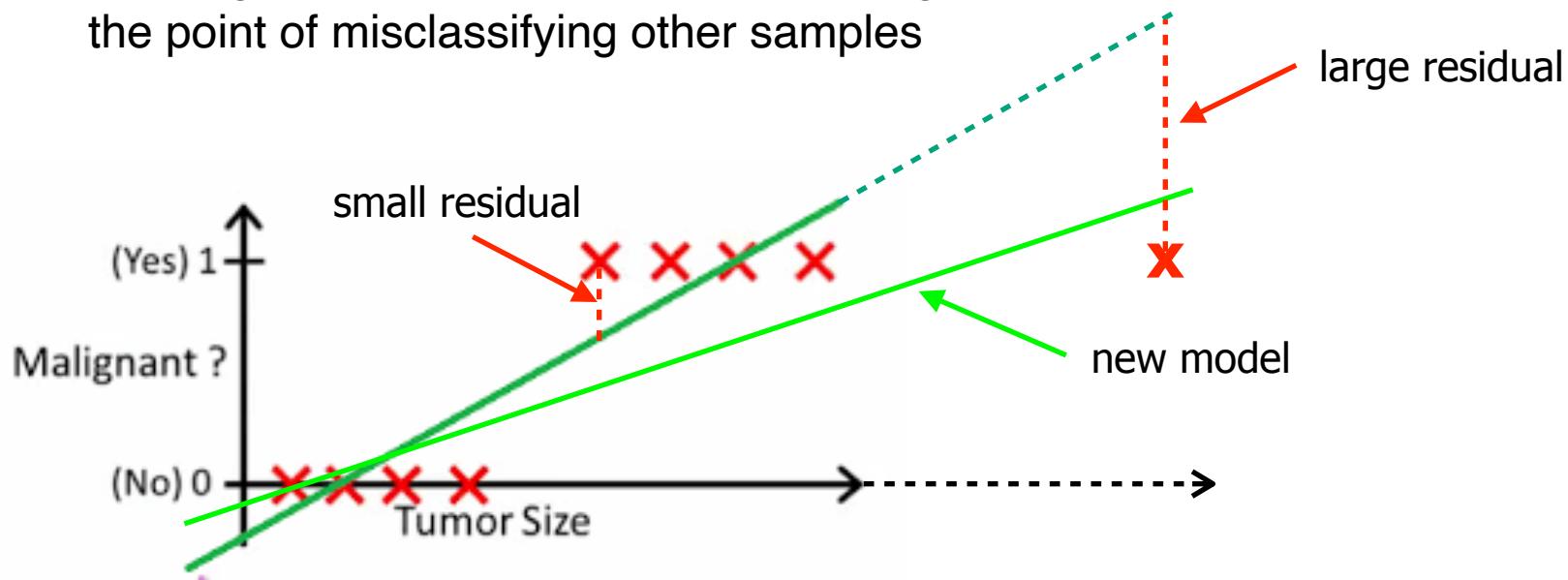
- Example (from Andrew Ng's course): Classify a tumor as benign vs malignant
 - 1D input (tumor size), 1D output (malignant vs benign)
 - Using a linear model, we can predict the label as $\hat{y} = w^{(1)}x + w^{(0)}$

$$\text{label} = \begin{cases} 1 & \text{if } \hat{y} \geq 0.5 \\ 0 & \text{otherwise} \end{cases}$$



Recap: Failures

- Back to the tumor prediction example
 - The residuals for “normal” samples are quite small
 - However, a sample with a much larger tumor size (with true label 1) would have a much larger residual
 - Training with such a sample would strongly affect the model parameters, to the point of misclassifying other samples

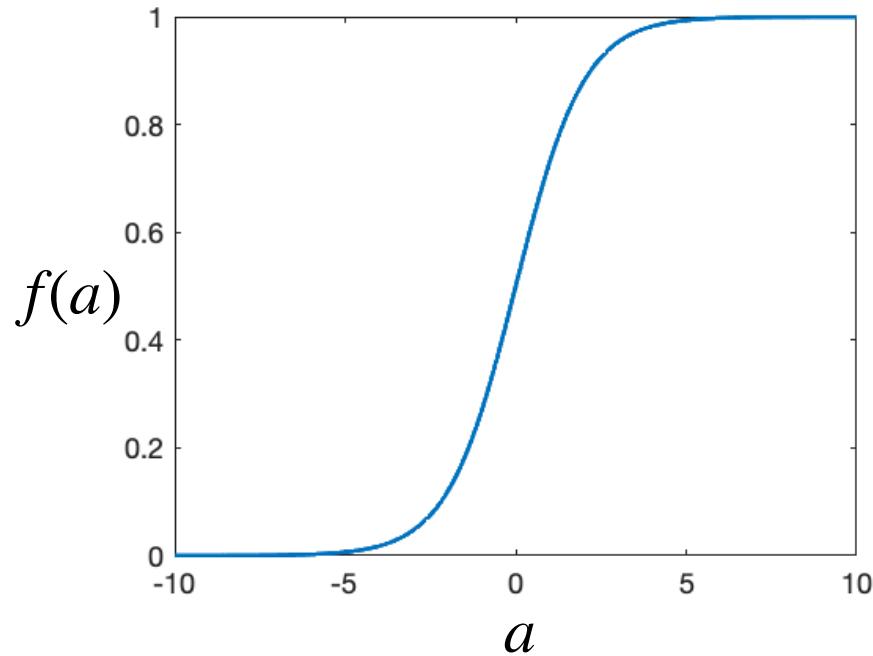


- This is because least-square classification fits a line to non-linear data

Recap: Adding non-linearity

- To model a classification problem, one can use a smooth approximation of the step function, such as the *logistic sigmoid function*

$$f(a) = \frac{1}{1 + \exp(-a)}$$



Recap: Probabilistic interpretation

- In the binary case, with D dimensional inputs, we can write the prediction of the model as

$$\hat{y}(\mathbf{x}) = \sigma(\mathbf{w}^T \mathbf{x}) = \frac{1}{1 + \exp(-\mathbf{w}^T \mathbf{x})}$$

- Such a prediction can be interpreted as the probability that \mathbf{x} belongs to the positive class (or as a score for the positive class)
- The probability to belong to the negative class (or score for the negative class) is then computed as $1 - \hat{y}(\mathbf{x})$

Recap: Logistic regression: Training

- Given N training samples, to learn the parameters \mathbf{w} of the logistic regression model, we minimize the cross-entropy

$$R(\mathbf{w}) = - \sum_{i=1}^N (y_i \ln(\hat{y}_i) + (1 - y_i) \ln(1 - \hat{y}_i))$$

- Since the true y_i is either 0 or 1, the cross-entropy can be re-written as

$$R(\mathbf{w}) = - \sum_{i \in \text{positive samples}} \ln(\hat{y}_i) - \sum_{i \in \text{negative samples}} \ln(1 - \hat{y}_i)$$

- Minimization is done via gradient descent

Exercises: Solution

- Describe two main differences between linear regression and logistic regression.
 1. Linear regression is used for regression tasks, i.e., to predict continuous values, whereas logistic regression is used for classification, i.e., to predict class labels.
 2. Logistic regression adds a nonlinearity to the output of the linear model.
- Describe one similarity between linear regression and logistic regression.
 - They internally rely on the same linear model and thus, for a given input dimension, have the same number of parameters.

Recap: Evaluating a classifier

- Confusion matrix: Binary case (e.g., spam vs non-spam email)

True class → Hypothesized class V	Pos	Neg
Yes	TP	FP
No	FN	TN
	P=TP+FN	N=FP+TN

TP: True positive (number of samples correctly classified as positive)

FP: False positive (number of samples incorrectly classified as positive)

TN: True negative (number of samples correctly classified as negative)

FN: False negative (number of samples incorrectly classified as negative)

P: Total number of positive samples

N: Total number of negative samples

Recap: Classification metrics

- Last time, we covered several metrics:
 - Accuracy: Percentage of correctly classified samples
 - Precision: Percentage of samples classified as positives that are truly positives
 - Recall: Percentage of positive samples that are correctly classified as positives
 - Area under the ROC curve
- Let me add two metrics that I skipped last time

Classification metrics

- False positive rate: Percentage of negative samples that are incorrectly classified as positives

$$FP \text{ rate} = \frac{FP}{N}$$

- Example:

True class →	Pos	Neg
Yes	200	100
No	300	400
	P=500	N=500

$$FP \text{ rate} = \frac{100}{500} = 0.2$$

Classification metrics

- F1 score: Single number that combines precision and recall

$$F1 = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall}$$

the higher the better !

- Example:

True class →	Pos	Neg
Yes	200	100
No	300	400
	P=500	N=500

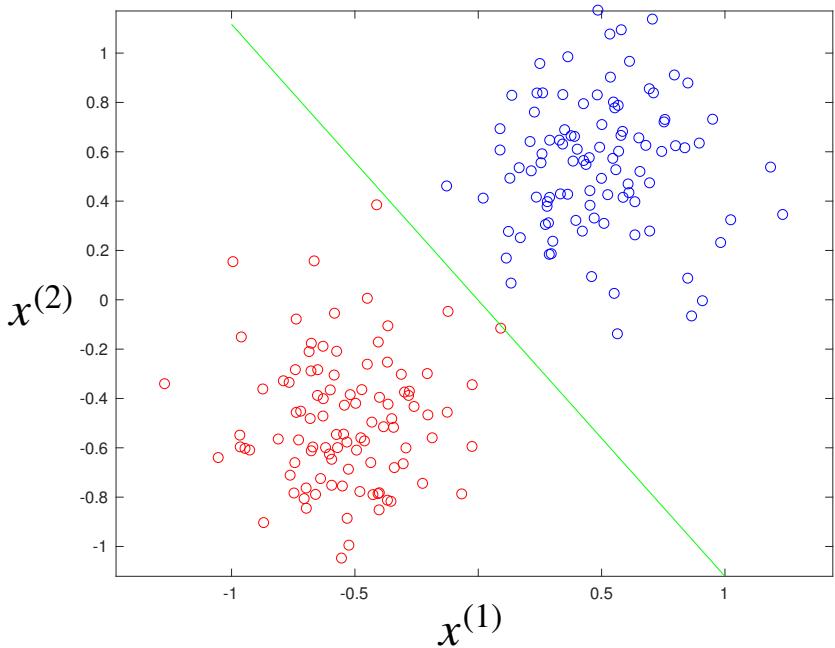
$$F1 = 2 \cdot \frac{0.667 \cdot 0.4}{0.667 + 0.4} = 0.5$$

Goals of today's lecture

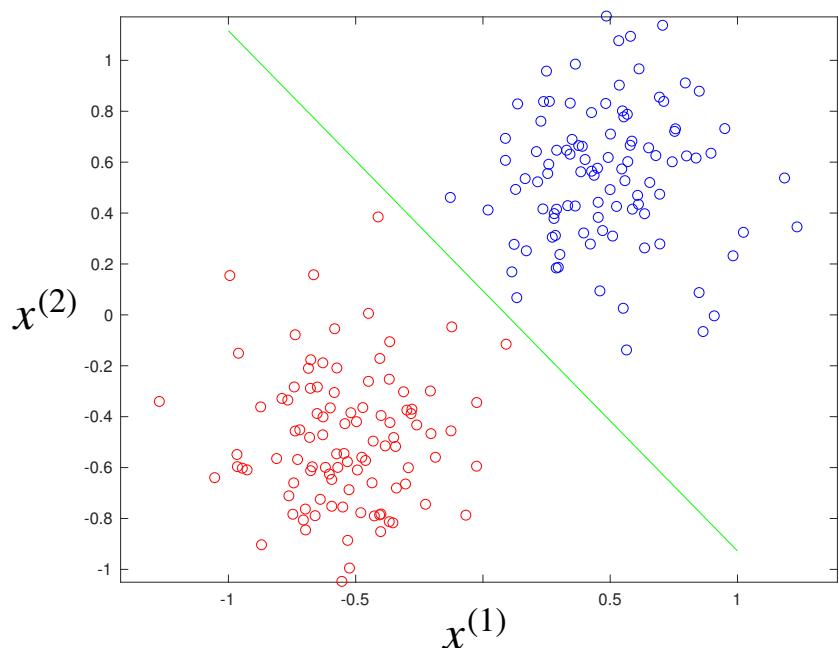
- Introduce the notion of margin between classes
- Derive the formulation for a maximum margin classifier (SVM), first for linearly separable data and then for non-linearly separable data
- Introduce basic notions of constrained optimization

Decision boundaries

- Different classifiers have different decision boundaries



Least-square classification



Logistic regression

- Can we define what a good decision boundary is?

Decision boundary properties

- The decision boundary is related to the parameters \mathbf{w} in the following manner (NB: today, the negative label is -1, not 0)
 - Let $\tilde{\mathbf{w}}$ be the vector of parameters without $w^{(0)}$

- Two points $\mathbf{x}_1, \mathbf{x}_2$ on the decision boundary have the same prediction ($\hat{y}_1 = \hat{y}_2 = 0$), thus

$$\tilde{\mathbf{w}}^T(\mathbf{x}_1 - \mathbf{x}_2) = \hat{y}_1 - w^{(0)} - (\hat{y}_2 - w^{(0)}) = 0$$

and so $\tilde{\mathbf{w}}$ is orthogonal to the decision boundary

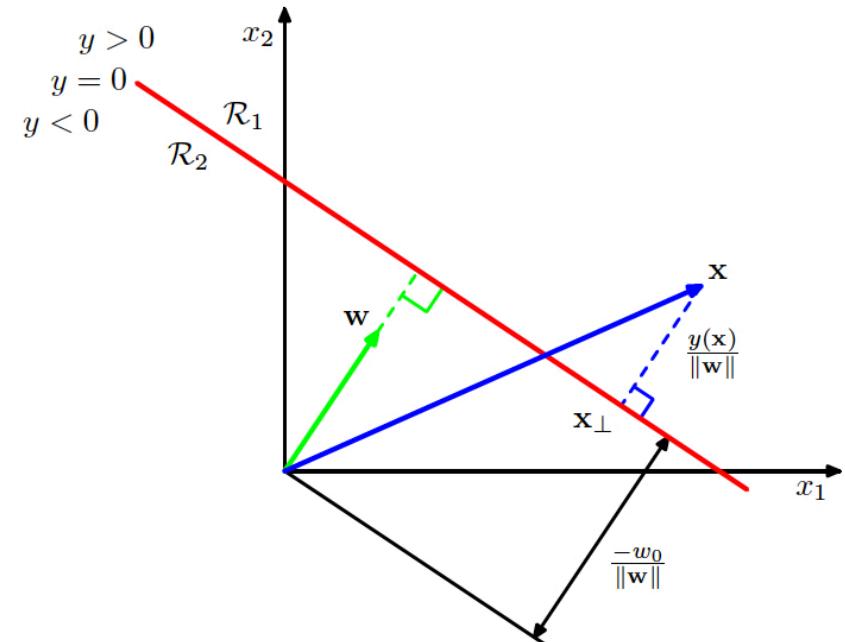


Figure from Bishop, Chapter 4.1.1
(\mathbf{w} in the fig. corresponds to $\tilde{\mathbf{w}}$)

Decision boundary properties

- Let \mathbf{x} be an arbitrary point, and \mathbf{x}_\perp its orthogonal projection on the decision boundary
- Because $\tilde{\mathbf{w}}$ is orthogonal to the decision boundary, we can write

$$\mathbf{x} = \mathbf{x}_\perp + r \frac{\tilde{\mathbf{w}}}{\|\tilde{\mathbf{w}}\|}$$

Multiplying on both sides by $\tilde{\mathbf{w}}^T$ yields

$$\tilde{\mathbf{w}}^T \mathbf{x} = \tilde{\mathbf{w}}^T \mathbf{x}_\perp + r \|\tilde{\mathbf{w}}\|$$

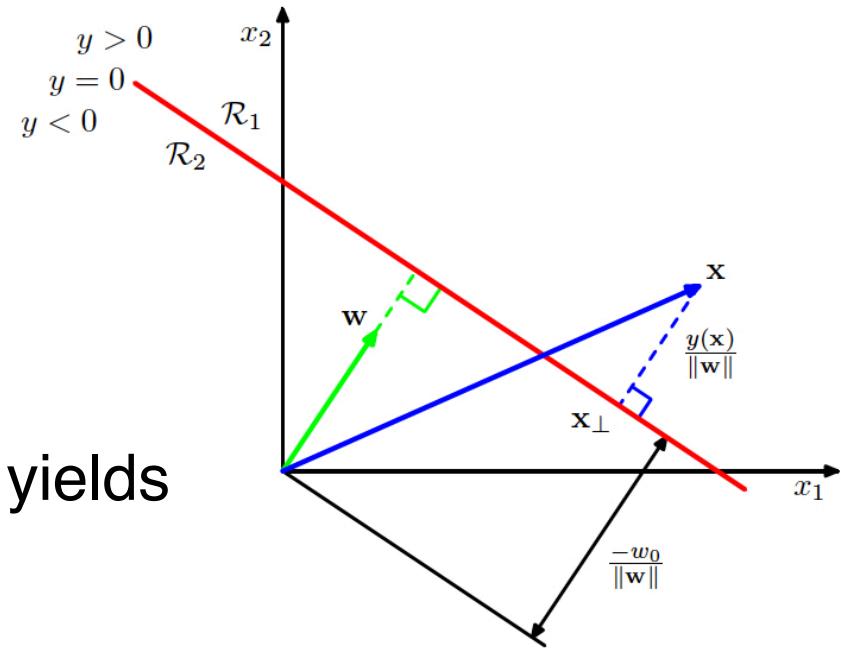


Figure from Bishop, Chapter 4.1.1

(\mathbf{w} in the fig. corresponds to $\tilde{\mathbf{w}}$)

Decision boundary properties

- Because \mathbf{x}_\perp is on the boundary, $\hat{y}(\mathbf{x}_\perp) = 0$, and so

$$\tilde{\mathbf{w}}^T \mathbf{x} = \tilde{\mathbf{w}}^T \mathbf{x}_\perp + r\|\tilde{\mathbf{w}}\| = \hat{y}(\mathbf{x}_\perp) - w^{(0)} + r\|\tilde{\mathbf{w}}\| = -w^{(0)} + r\|\tilde{\mathbf{w}}\|$$

- Passing $w^{(0)}$ on the lefthand side yields

$$\tilde{\mathbf{w}}^T \mathbf{x} + w^{(0)} = \hat{y}(\mathbf{x}) = r\|\tilde{\mathbf{w}}\|$$

And so, ultimately

$$r = \frac{\hat{y}(\mathbf{x})}{\|\tilde{\mathbf{w}}\|}$$

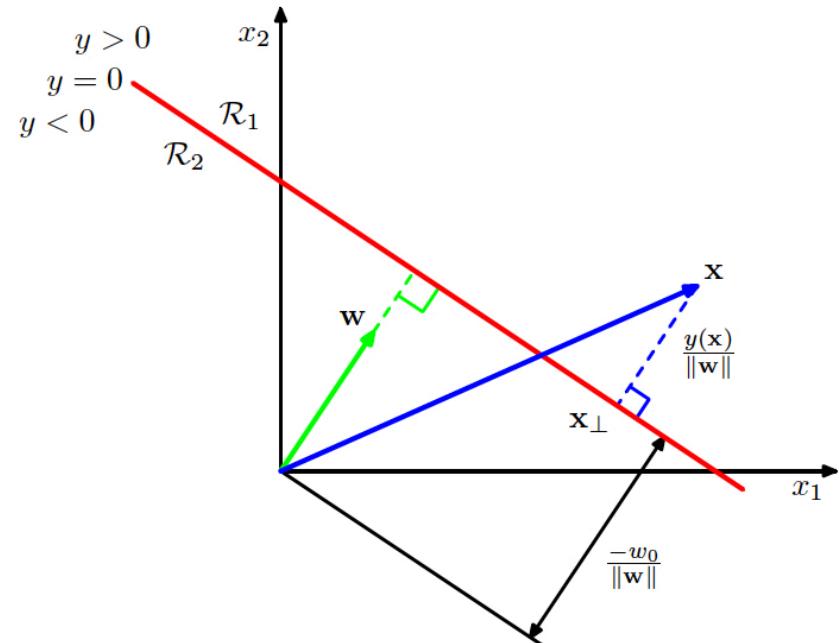


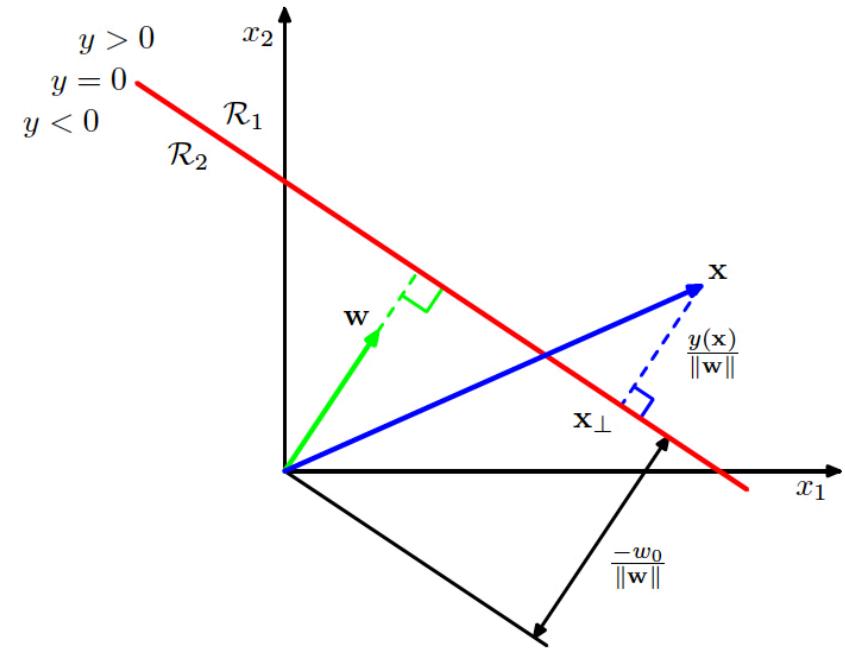
Figure from Bishop, Chapter 4.1.1

(\mathbf{w} in the fig. corresponds to $\tilde{\mathbf{w}}$)

Distance to the decision boundary

- In words: The signed distance from a point \mathbf{x} to the boundary is given by its prediction $\hat{y}(\mathbf{x})$ and the parameters $\tilde{\mathbf{w}}$ as

$$r = \frac{\hat{y}(\mathbf{x})}{\|\tilde{\mathbf{w}}\|}$$



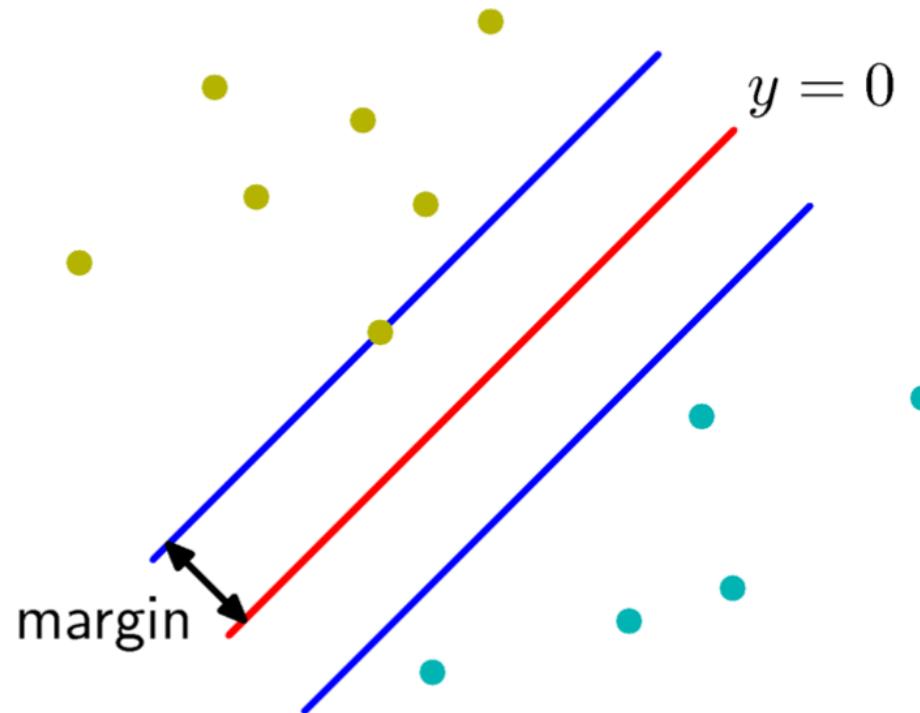
- We can use this to find a classifier whose decision boundary is as far as possible from all the points

Figure from Bishop, Chapter 4.1.1

(\mathbf{w} in the fig. corresponds to $\tilde{\mathbf{w}}$)

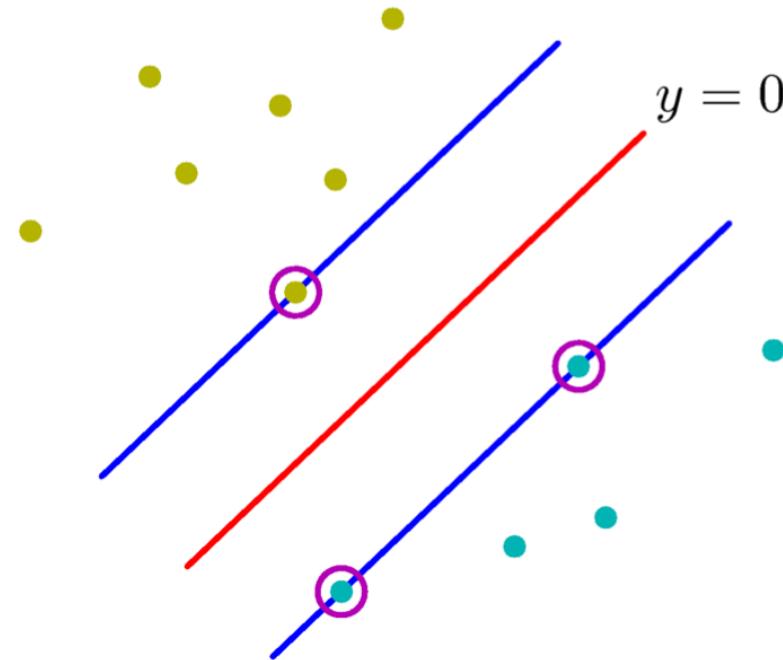
Margin (Chapter 7.1 in Bishop's book)

- The orthogonal distance between the boundary and the nearest sample is called *margin*



Support vectors

- A good decision boundary should **maximize the margin**
 - Such a boundary is determined by a subset of the data points, named *support vectors* (indicated with circles)



Maximum margin classifier

- For a solution where all the points are correctly classified

$$y_i \cdot \hat{y}_i = y_i \cdot (\tilde{\mathbf{w}}^T \mathbf{x}_i + w^{(0)}) > 0$$

- Making use of the definition of the signed distance to the boundary, we can write the (unsigned) distance as

$$\tilde{r}_i = \frac{y_i \cdot (\tilde{\mathbf{w}}^T \mathbf{x}_i + w^{(0)})}{\|\tilde{\mathbf{w}}\|}$$

- A maximum margin classifier then aims to maximize this distance for the point closest to the boundary, i.e., maximize the minimum such distance

Maximum margin classifier

- Mathematically, this can be expressed as

$$\{\tilde{\mathbf{w}}^*, w^{(0)*}\} = \operatorname{argmax}_{\tilde{\mathbf{w}}, w^{(0)}} \min_{i=1}^N \left(\frac{y_i \cdot (\tilde{\mathbf{w}}^T \mathbf{x}_i + w^{(0)})}{\|\tilde{\mathbf{w}}\|} \right)$$

Margin: Distance
between the boundary
and the nearest sample

- Unfortunately, solving this optimization problem is difficult
- We will therefore convert it to an equivalent problem that is easier to solve

Maximum margin classifier: Scale

- Note that the distance to the boundary is invariant to scaling the parameters

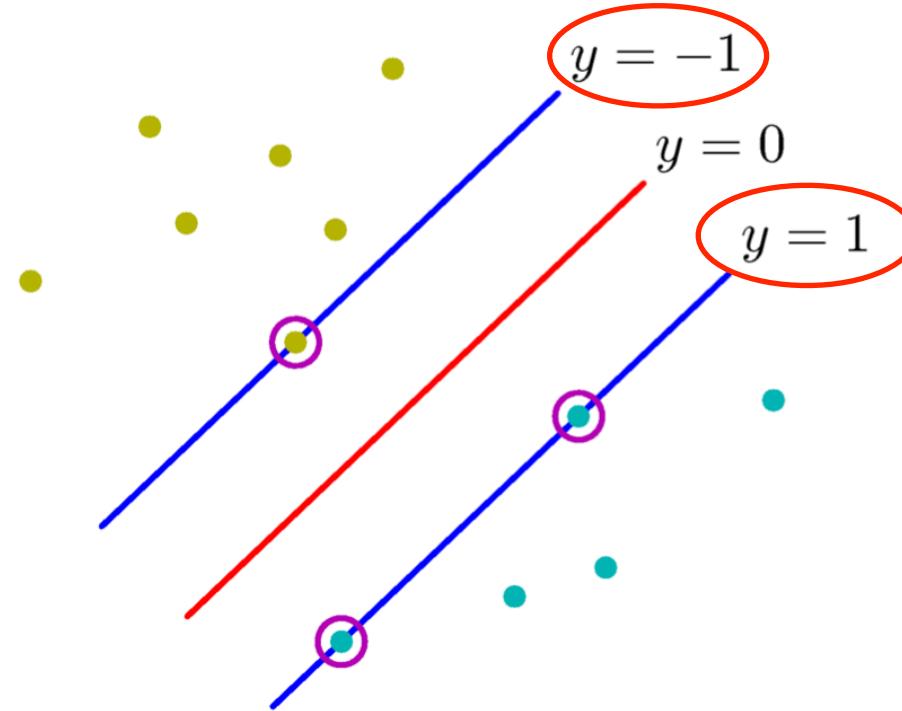
$$\frac{y_i \cdot (\tilde{\mathbf{w}}^T \mathbf{x}_i + w^{(0)})}{\|\tilde{\mathbf{w}}\|} = \frac{\lambda \cdot y_i \cdot (\tilde{\mathbf{w}}^T \mathbf{x}_i + w^{(0)})}{\lambda \cdot \|\tilde{\mathbf{w}}\|} = \frac{y_i \cdot (\lambda \cdot \tilde{\mathbf{w}}^T \mathbf{x}_i + \lambda \cdot w^{(0)})}{\|\lambda \cdot \tilde{\mathbf{w}}\|}$$

- This means that there are an infinite number of equivalent solutions for $\tilde{\mathbf{w}}$ and $w^{(0)}$
- And we can constrain the parameters to be such that, for the point j that is closest to the boundary,

$$y_j \cdot (\tilde{\mathbf{w}}^T \mathbf{x}_j + w^{(0)}) = 1$$

Maximum margin classifier: Scale

- In other words, with such a fixed scale, the support vectors will lie on hyperplanes at a distance 1 from the decision boundary



Maximum margin classifier

- This means that any point i must satisfy

$$y_i \cdot (\tilde{\mathbf{w}}^T \mathbf{x}_i + w^{(0)}) \geq 1$$

- For the points for which the equality holds, the constraints are said to be *active*; for the other ones, they are *inactive*
 - There will always be at least one active constraint, because there is always one closest point to the boundary
 - Once the margin is maximized, there will always be at least two active constraints, one from each class
- Because of these constraints, we have that

$$\min_i (y_i \cdot (\tilde{\mathbf{w}}^T \mathbf{x}_i + w^{(0)})) = 1$$

Maximum margin classifier

- This means that we can, to maximize the margin, we only need to maximize $1/\|\tilde{\mathbf{w}}\|$
- This is equivalent to minimizing $\|\tilde{\mathbf{w}}\|^2$, so we can write the solution to a max-margin classifier as

$$\min_{\tilde{\mathbf{w}}, w^{(0)}} \frac{1}{2} \|\tilde{\mathbf{w}}\|^2$$

subject to $y_i \cdot (\tilde{\mathbf{w}}^T \mathbf{x}_i + w^{(0)}) \geq 1 \quad \forall i$

- This formulation is called *Support Vector Machine*
 - The factor $1/2$ was added for later convenience and does not affect the solution

Support Vector Machine

- The SVM formulation

$$\min_{\tilde{\mathbf{w}}, w^{(0)}} \frac{1}{2} \|\tilde{\mathbf{w}}\|^2$$

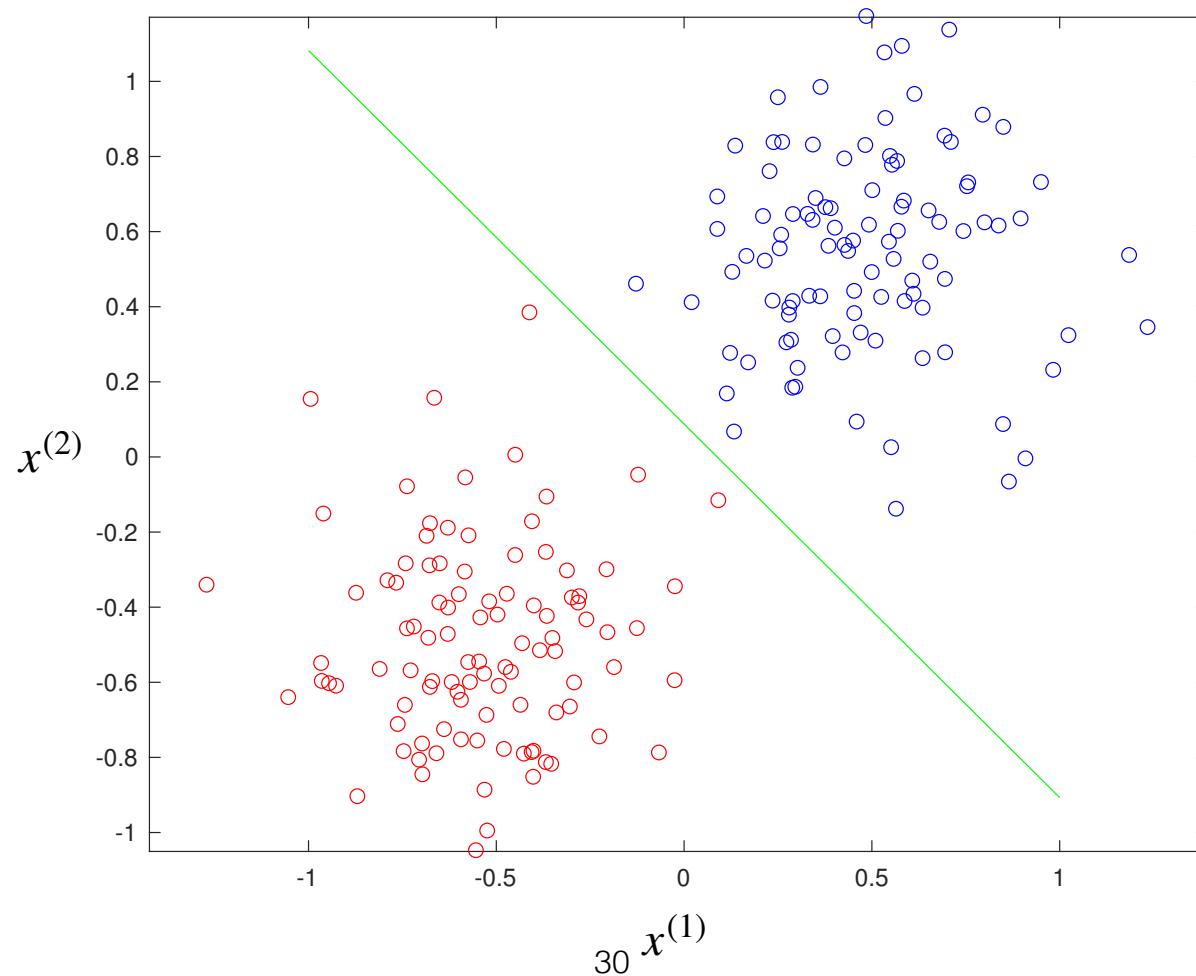
subject to $y_i \cdot (\tilde{\mathbf{w}}^T \mathbf{x}_i + w^{(0)}) \geq 1 , \forall i$

is a quadratic program

- Quadratic objective function, with linear constraints
- Here, the problem is convex, and can be solved to optimality with standard iterative solvers

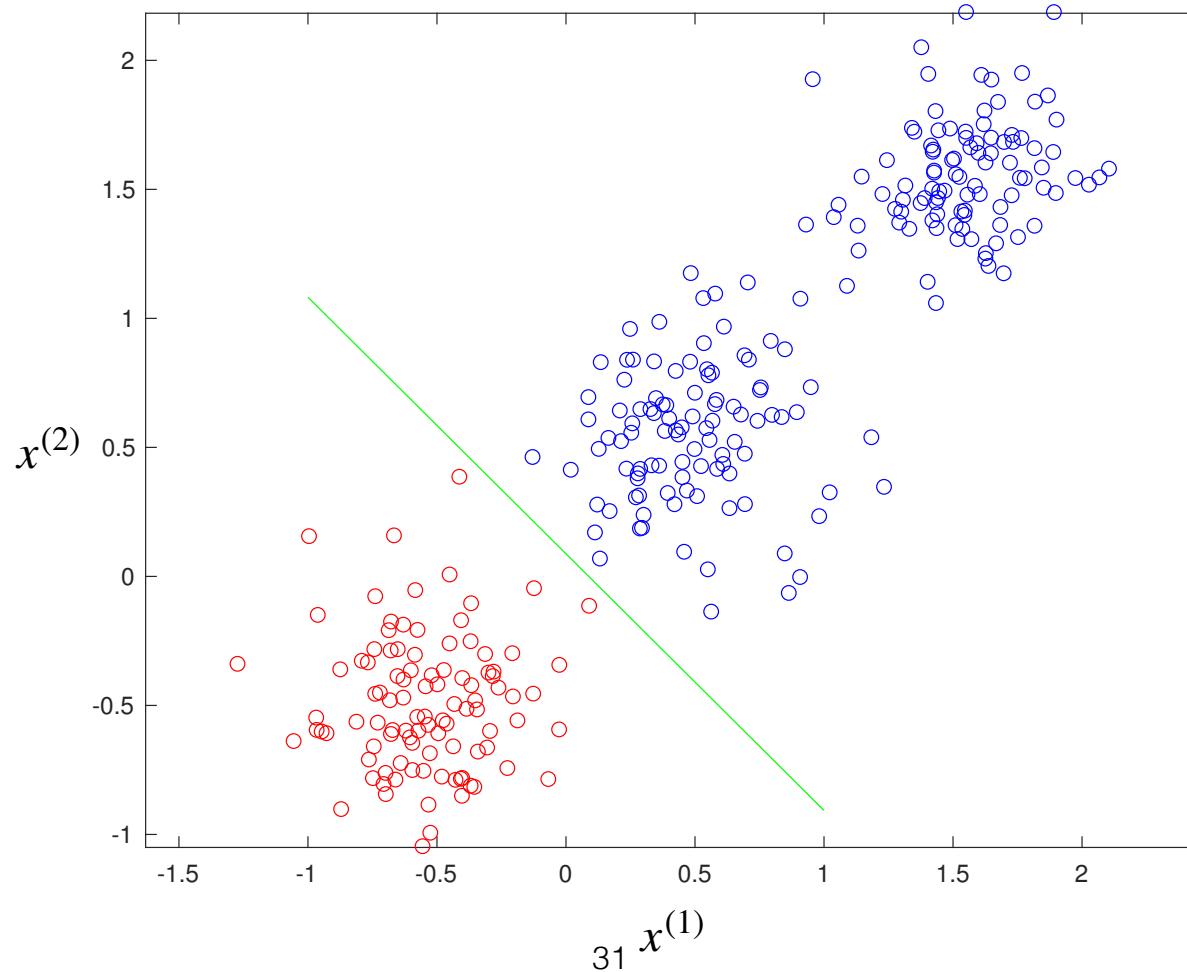
SVM Example

- Back to our toy data (2D samples from 2 classes)



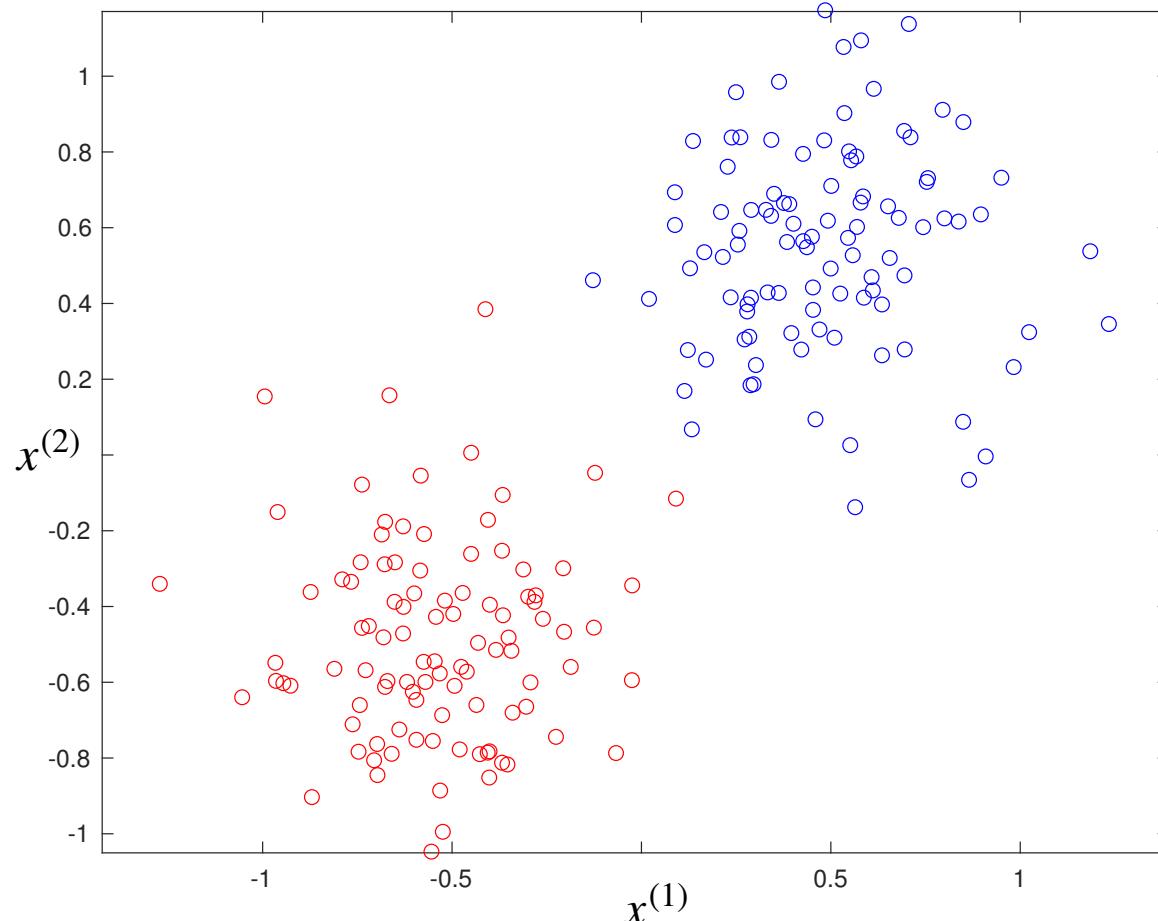
SVM Example

- In the presence of additional samples, it still works



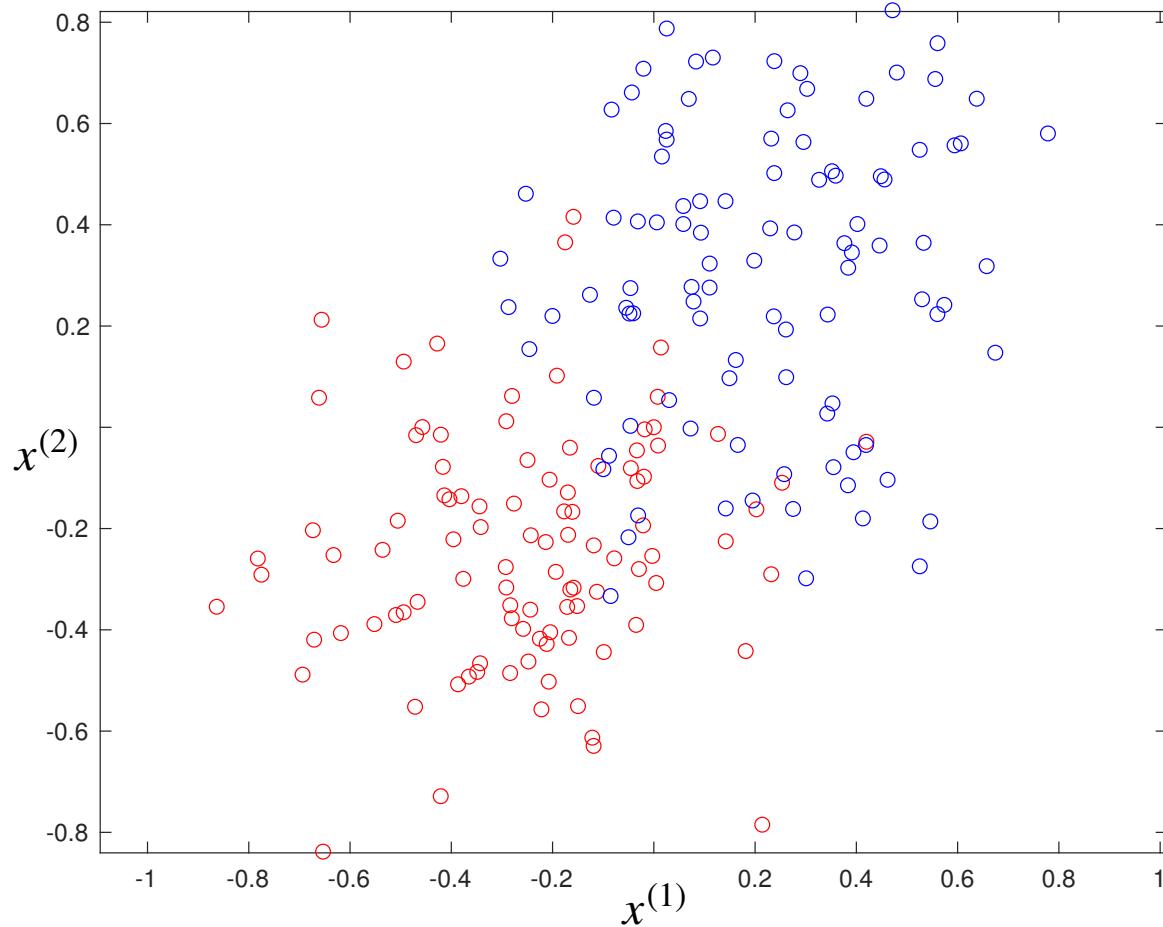
Overlapping classes

- In practice, the data essentially never looks like this



Overlapping classes

- but rather like this



Support Vector Machine

- This means that the constraints in the SVM formulation

$$\min_{\tilde{\mathbf{w}}, w^{(0)}} \frac{1}{2} \|\tilde{\mathbf{w}}\|^2$$

$$\text{subject to } y_i \cdot (\tilde{\mathbf{w}}^T \mathbf{x}_i + w^{(0)}) \geq 1, \quad \forall i$$

cannot all be satisfied

- So the problem simply cannot be solved as such

- For some samples, we need to allow $y_i \cdot (\tilde{\mathbf{w}}^T \mathbf{x}_i + w^{(0)})$ to be less than 1
- Note that for the other classifiers we have seen so far, this is not a real problem, because they do not have hard constraints; this would simply translate to a higher loss value

Slack variables

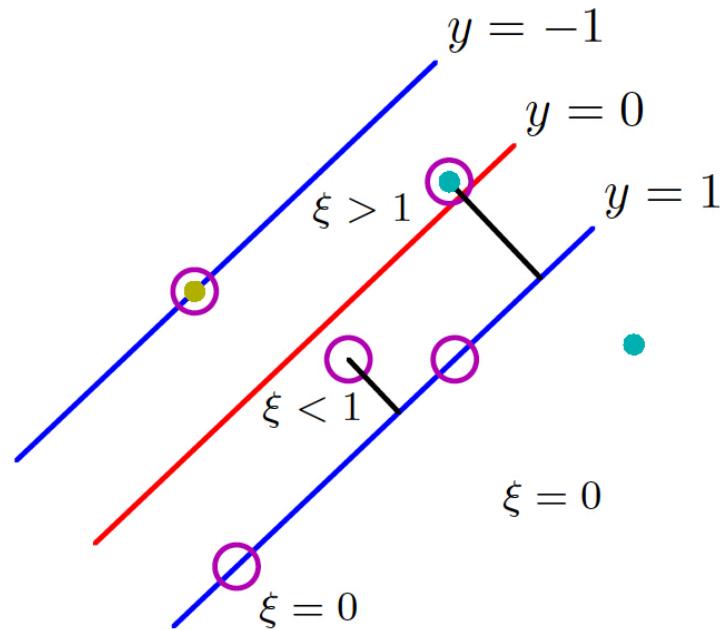
- To overcome this, we introduce an additional slack variable ξ_i for each sample
- We then re-write the constraints as

$$y_i \cdot (\tilde{\mathbf{w}}^T \mathbf{x}_i + w^{(0)}) \geq 1 - \xi_i$$

- With $\xi_i \geq 0$, this allows violating the original constraints

Slack variables

- Visually, the slack variables represent how much one violates the original constraints
 - If $0 < \xi_i \leq 1$, sample i lies inside the margin, but is still correctly classified
 - If $\xi_i \geq 1$, then sample i is misclassified



Slack variables

- During training, we then optimize the slack variables jointly with the other parameters
- Naively, we could thus think of writing the problem as

$$\min_{\tilde{\mathbf{w}}, w^{(0)}, \{\xi_i\}} \frac{1}{2} \|\tilde{\mathbf{w}}\|^2$$

subject to $y_i \cdot (\tilde{\mathbf{w}}^T \mathbf{x}_i + w^{(0)}) \geq 1 - \xi_i , \quad \forall i$

$$\xi_i \geq 0 , \quad \forall i$$

- However, this would simply allow the model to violate all the original constraints at no cost, and would make a useless classifier

Slack variables

- To prevent this, we encourage the slack variables to remain close to zero
- We therefore write the SVM problem as

$$\min_{\tilde{\mathbf{w}}, w^{(0)}, \{\xi_i\}} \frac{1}{2} \|\tilde{\mathbf{w}}\|^2 + C \sum_{i=1}^N \xi_i$$

subject to $y_i \cdot (\tilde{\mathbf{w}}^T \mathbf{x}_i + w^{(0)}) \geq 1 - \xi_i , \forall i$

$$\xi_i \geq 0 , \forall i$$

where C sets the influence of the regularizer on the slack variables

Exercise

- The slack variables allow SVM to handle the case where the classes are not truly linearly separable but still close to being so. Describe two reasons why this can happen in practice.

Support vector machine: Demo

- <http://www.cristiandima.com/basics-of-support-vector-machines/>

SVM: Solution

- The SVM optimization problem can be solved using standard quadratic programming frameworks
- The formulation we have seen so far is referred to as the primal problem
- Nevertheless, one can instead derive the dual SVM problem
 - We will see later today and in a future lecture why this can be useful

Interlude

Constrained optimization & Lagrange duality

Constrained optimization

- In general, an optimization problem can be written

$$\min_{\mathbf{w}} R(\mathbf{w})$$

subject to $f_i(\mathbf{w}) \leq 0, \quad i = 1, \dots, M$ (M inequality constraints)

$h_i(\mathbf{w}) = 0, \quad i = 1, \dots, P$ (P equality constraints)

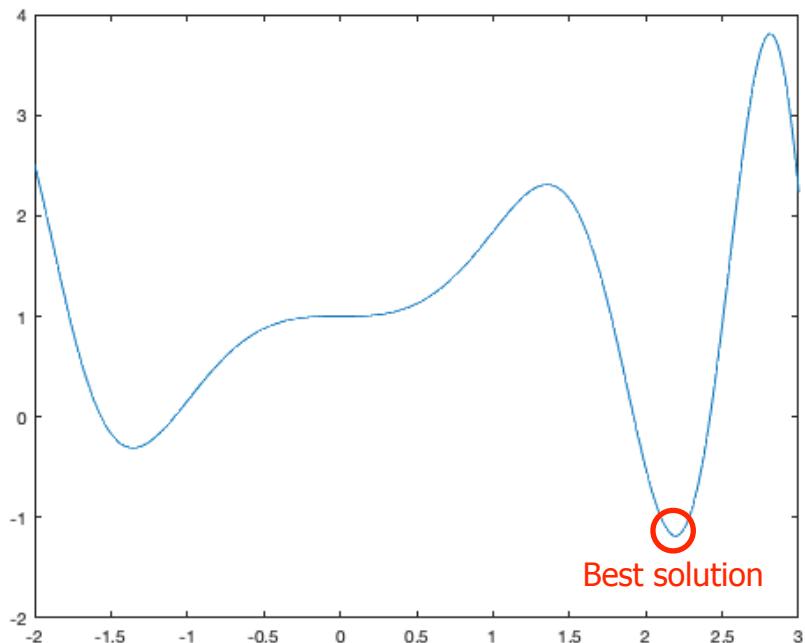
- Note that the constrained problem will typically not have the same solution as the unconstrained one

Adding constraints: Example

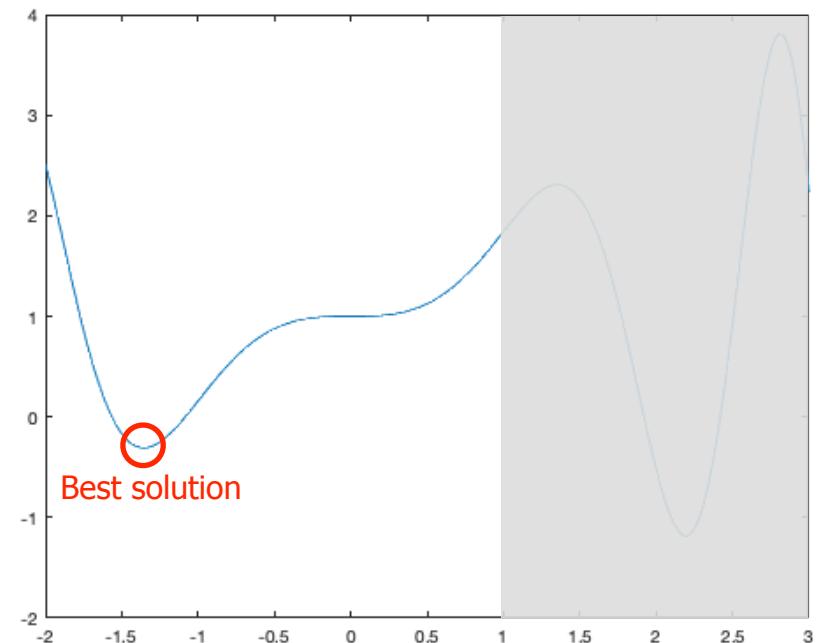
- Inequality: 1D sine-based function seen before

$$R(w) = w \sin(w^2) + 1$$

Unconstrained



Constrained
 $f_1(w) = w - 1 \leq 0$

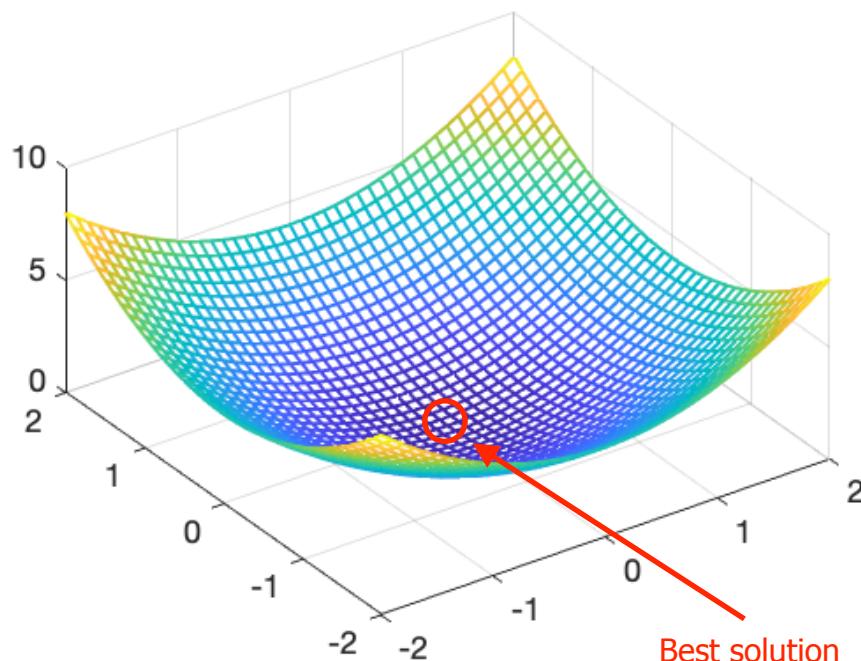


Adding constraints: Example

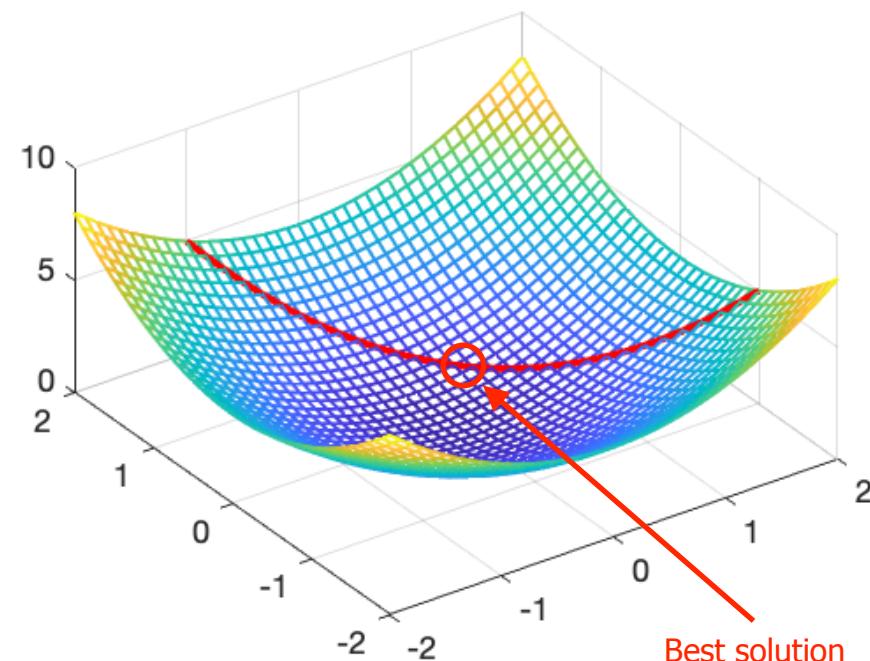
- Equality: 2D quadratic function

$$R(\mathbf{w}) = (w^{(1)})^2 + (w^{(2)})^2$$

Unconstrained



Constrained
 $h_1(\mathbf{w}) = w^{(1)} + w^{(2)} - 1 = 0$



Lagrangian

- The **Lagrangian** of an optimization problem

$$\min_{\mathbf{w}} R(\mathbf{w})$$

subject to $f_i(\mathbf{w}) \leq 0, \quad i = 1, \dots, M$
 $h_i(\mathbf{w}) = 0, \quad i = 1, \dots, P$

is expressed as

$$L(\mathbf{w}, \lambda, \nu) = R(\mathbf{w}) + \sum_{i=1}^M \lambda_i f_i(\mathbf{w}) + \sum_{i=1}^P \nu_i h_i(\mathbf{w})$$

- λ_i is the Lagrange multiplier associated with $f_i(\mathbf{w}) \leq 0$
- ν_i is the Lagrange multiplier associated with $h_i(\mathbf{w}) = 0$

Lagrange dual function

- The Lagrange dual function is expressed as

$$\begin{aligned} g(\lambda, \nu) &= \inf_{\mathbf{w}} L(\mathbf{w}, \lambda, \nu) \\ &= \inf_{\mathbf{w}} \left(R(\mathbf{w}) + \sum_{i=1}^M \lambda_i f_i(\mathbf{w}) + \sum_{i=1}^P \nu_i h_i(\mathbf{w}) \right) \end{aligned}$$

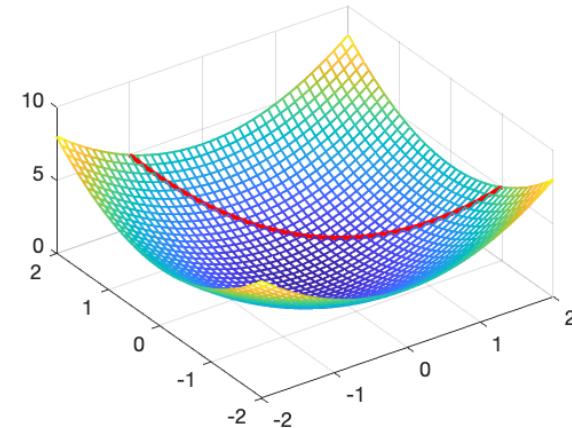
- For simplicity, think of \inf as \min
- In other words, to compute the Lagrange dual function for given values (λ, ν) , we need to minimize the Lagrangian w.r.t. \mathbf{w} while keep (λ, ν) fixed
- $g(\lambda, \nu)$ is a concave function (opposite of convex, e.g., single maximum)
- It forms a lower bound to the optimal value R^* of the original problem
 - if $\lambda \geq 0$, then $g(\lambda, \nu) \leq R^*$ for any (λ, ν)

Lagrange dual function: Example

- Quadratic function with equality constraint

$$\min_{\mathbf{w}} (\mathbf{w}^{(1)})^2 + (\mathbf{w}^{(2)})^2$$

subject to $\mathbf{w}^{(1)} + \mathbf{w}^{(2)} - 1 = 0$



$$L(\mathbf{w}, \nu_1) = (\mathbf{w}^{(1)})^2 + (\mathbf{w}^{(2)})^2 + \nu_1 (\mathbf{w}^{(1)} + \mathbf{w}^{(2)} - 1)$$

- To minimize $L(\mathbf{w}, \nu_1)$ w.r.t. \mathbf{w} , we can set the gradient to 0

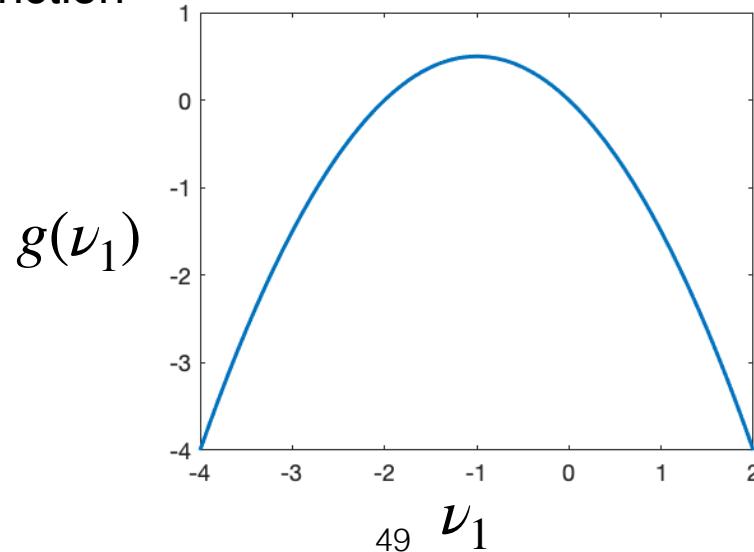
$$\nabla_{\mathbf{w}} L(\mathbf{w}, \nu_1) = \begin{bmatrix} 2\mathbf{w}^{(1)} + \nu_1 \\ 2\mathbf{w}^{(2)} + \nu_1 \end{bmatrix} = \mathbf{0} \Leftrightarrow \mathbf{w}^* = \begin{bmatrix} -\nu_1/2 \\ -\nu_1/2 \end{bmatrix}$$

Lagrange dual function: Example

- We can then plug this result in L to obtain

$$\begin{aligned} g(\nu_1) &= L \left(\begin{bmatrix} -\nu_1/2 \\ -\nu_1/2 \end{bmatrix}, \nu_1 \right) = \frac{1}{4}\nu_1^2 + \frac{1}{4}\nu_1^2 + \nu_1 \left(-\frac{1}{2}\nu_1 - \frac{1}{2}\nu_1 - 1 \right) \\ &= -\frac{1}{2}\nu_1^2 - \nu_1 \end{aligned}$$

- This is a concave function



Lagrange dual problem

- Since the Lagrange dual function is a lower bound to the optimal function value, one can aim to maximize it

$$\max_{\lambda, \nu} g(\lambda, \nu)$$

subject to $\lambda \geq 0$

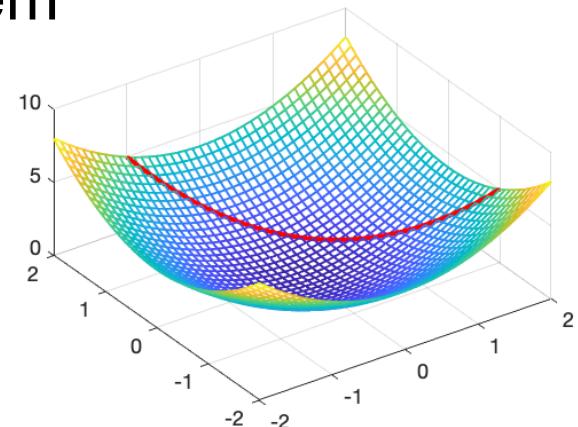
- However, this bound is not always tight
 - The maximum value of the dual problem does not always reach the minimum value of the primal (original) one
 - Nevertheless, under some conditions, it does

Lagrange dual problem: Example

- As shown before, for the convex problem

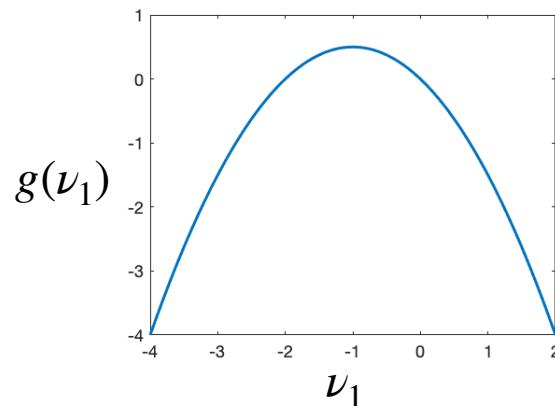
$$\min_{\mathbf{w}} (\mathbf{w}^{(1)})^2 + (\mathbf{w}^{(2)})^2$$

subject to $\mathbf{w}^{(1)} + \mathbf{w}^{(2)} - \mathbf{1} = 0$



- the dual function is

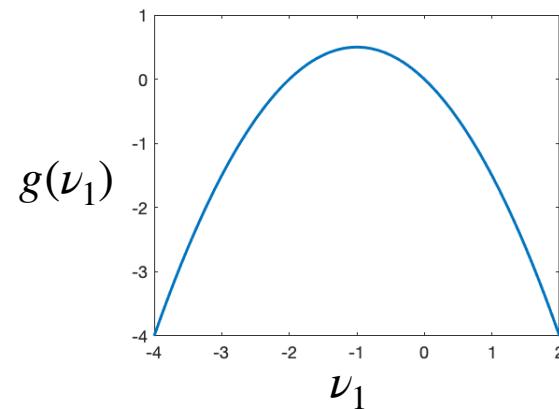
$$g(\nu_1) = -\frac{1}{2}\nu_1^2 - \nu_1$$



Lagrange dual problem: Example

- We can maximize the dual function by zeroing out its derivative

$$-\nu_1 - 1 = 0 \Leftrightarrow \nu_1 = -1$$



- Then, we have that

$$\mathbf{w}^* = \begin{bmatrix} -\nu_1/2 \\ -\nu_1/2 \end{bmatrix} = \begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix}$$

- And the maximum value is $g(-1) = 0.5$

Lagrange dual problem: Example

- Alternatively, we can rewrite the convex problem

$$\min_{\mathbf{w}} (\mathbf{w}^{(1)})^2 + (\mathbf{w}^{(2)})^2$$

subject to $\mathbf{w}^{(1)} + \mathbf{w}^{(2)} - 1 = 0$

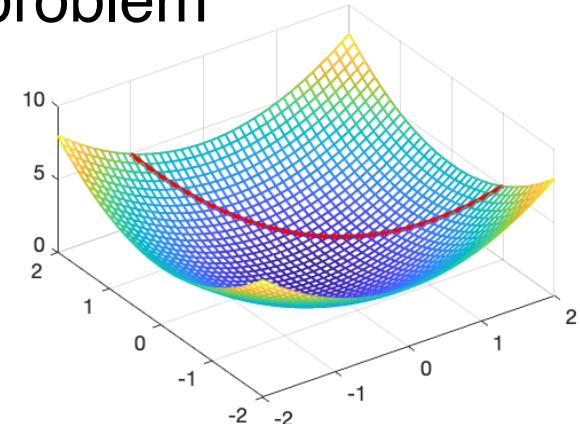
- as

$$\min_{w^{(1)}} (\mathbf{w}^{(1)})^2 + (1 - w^{(1)})^2 = 2(\mathbf{w}^{(1)})^2 - 2w^{(1)} + 1$$

- By zeroing out its derivative, we have

$$4w^{(1)} - 2 = 0 \Leftrightarrow w^{(1)} = 0.5$$

- And so $w^{(2)} = 0.5$, and the optimal function value is 0.5
 - So the bound given by the dual function in this case is tight



End of the interlude

Back to SVM

SVM: Lagrangian

- Let us first consider the case without slack variables
- As seen before, the SVM primal formulation is given by

$$\min_{\tilde{\mathbf{w}}, w^{(0)}} \frac{1}{2} \|\tilde{\mathbf{w}}\|^2$$

subject to $y_i \cdot (\tilde{\mathbf{w}}^T \mathbf{x}_i + w^{(0)}) \geq 1 , \forall i$

- The **Lagrangian of the SVM problem** is

$$L(\tilde{\mathbf{w}}, w^{(0)}, \{\alpha_i\}) = \frac{1}{2} \|\tilde{\mathbf{w}}\|^2 - \sum_{i=1}^N \alpha_i (y_i \cdot (\tilde{\mathbf{w}}^T \mathbf{x}_i + w^{(0)}) - 1)$$

where α_i is the Lagrange multiplier for constraint i , subject to the constraint $\alpha_i \geq 0$

Lagrange dual function

- We can then express the Lagrange dual function as

$$g(\{\alpha_i\}) = \min_{\tilde{\mathbf{w}}, w^{(0)}} L(\tilde{\mathbf{w}}, w^{(0)}, \{\alpha_i\})$$

which, as mentioned before, is a concave function that provides a lower bound to the original problem

- Because the SVM problem is convex, this lower bound is tight, so we can obtain its optimal solution by solving

$$\max_{\{\alpha_i\}} g(\{\alpha_i\})$$

subject to $\alpha_i \geq 0, \forall i$

Lagrange dual problem

- To solve this Lagrange dual problem, we first need to get the solution to the inner minimization

$$\min_{\tilde{\mathbf{w}}, w^{(0)}} L(\tilde{\mathbf{w}}, w^{(0)}, \{\alpha_i\}) = \min_{\tilde{\mathbf{w}}, w^{(0)}} \left(\frac{1}{2} \|\tilde{\mathbf{w}}\|^2 - \sum_{i=1}^N \alpha_i (y_i \cdot (\tilde{\mathbf{w}}^T \mathbf{x}_i + w^{(0)}) - 1) \right)$$

- This can be achieved by zeroing out the derivatives of $L(\cdot)$ w.r.t. $\tilde{\mathbf{w}}$ and $w^{(0)}$

Derivative w.r.t. $\tilde{\mathbf{w}}$

- We have

$$\frac{\partial L}{\partial \tilde{\mathbf{w}}} = \tilde{\mathbf{w}} - \sum_{i=1}^N \alpha_i y_i \mathbf{x}_i$$

- Setting the derivative to 0 gives

$$\tilde{\mathbf{w}} = \sum_{i=1}^N \alpha_i y_i \mathbf{x}_i$$

Derivative w.r.t. w_0

- We have

$$\frac{\partial L}{\partial w^{(0)}} = - \sum_{i=1}^N \alpha_i y_i$$

- Setting the derivative to 0 gives the condition

$$\sum_{i=1}^N \alpha_i y_i = 0$$

Lagrange dual problem

- By re-inserting the solution for $\tilde{\mathbf{w}}$ in the Lagrangian, we eventually get the dual problem

$$\max_{\{\alpha_i\}} \sum_{i=1}^N \alpha_i - \frac{1}{2} \sum_{i=1}^N \sum_{j=1}^N \alpha_i \alpha_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j$$

subject to

$$\sum_{i=1}^N \alpha_i y_i = 0$$

$$\alpha_i \geq 0, \forall i$$

Standard condition on the Lagrange multipliers for inequality constraints

Condition from the derivative w.r.t. $w^{(0)}$

Lagrange dual problem

- The dual problem is also a quadratic program
 - Note that it has N variables instead of $D + 1$ in the primal formulation
- Because it is a concave maximization problem, it can be solved to optimality
- At the solution, it can be shown that

$$\alpha_i^* \left(y_i \cdot ((\tilde{\mathbf{w}}^*)^T \mathbf{x}_i + w^{(0)*}) - 1 \right) = 0$$

- This means that, for every sample, either $\alpha_i^* = 0$, or
$$y_i \cdot ((\tilde{\mathbf{w}}^*)^T \mathbf{x}_i + w^{(0)*}) = 1$$
 - By definition, a point that satisfies the latter constraint is at a margin 1 from the decision boundary, and is thus a *support vector*

Prediction

- Once the optimal solution $\{\alpha_i^*\}$ is obtained, one can use them to predict the label for a new sample \mathbf{x} as

$$\hat{y}(\mathbf{x}) = (\tilde{\mathbf{w}}^*)^T \mathbf{x} + w^{(0)*} = \sum_{i=1}^N \alpha_i^* y_i \mathbf{x}_i^T \mathbf{x} + w^{(0)*}$$

- Because for all samples that are not support vectors, we have $\alpha_i^* = 0$, the sum can be computed on the support vectors only, i.e., with \mathcal{S} the set of support vector indices,

$$\hat{y}(\mathbf{x}) = \sum_{i \in \mathcal{S}} \alpha_i^* y_i \mathbf{x}_i^T \mathbf{x} + w^{(0)*}$$

- N.B. An attentive observer may have noticed that $w^{(0)}$ disappeared when computing the derivatives of the Lagrangian. To understand how to compute $w^{(0)*}$, I therefore invite the interested reader to look at Section 7.1 in Bishop's book

SVM with slack variables: Lagrangian

- When using slack variables, the **Lagrangian becomes**

$$L(\mathbf{w}, \{\xi_i\}, \{\alpha_i\}, \{\mu_i\}) = \frac{1}{2} \|\tilde{\mathbf{w}}\|^2 + C \sum_{i=1}^N \xi_i - \sum_{i=1}^N \alpha_i (y_i \cdot (\tilde{\mathbf{w}}^T \mathbf{x}_i + w^{(0)}) - 1 + \xi_i) - \sum_{i=1}^N \mu_i \xi_i$$

Annotations:

- New Lagrange multipliers for the slack variables
- Same as before
- Regularizer on the slack variables
- Similar to before, with the additional slack variable
- Constraints on the slack variables

- Deriving the dual problem then follows a similar strategy to that in the case without slack variables

Lagrange dual problem with slack variables

- Ultimately, the $\{\mu_i\}$ can be suppressed, and we obtain the dual problem

$$\max_{\{\alpha_i\}} \sum_{i=1}^N \alpha_i - \frac{1}{2} \sum_{i=1}^N \sum_{j=1}^N \alpha_i \alpha_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j$$

subject to $\sum_{i=1}^N \alpha_i y_i = 0$

$$0 \leq \alpha_i \leq C, \quad \forall i$$

It only differs from
the previous one in
these constraints

Support vectors & prediction with slack variables

- With slack variables, at the solution, we have

$$\alpha_i^* \left(y_i \cdot ((\tilde{\mathbf{w}}^*)^T \mathbf{x}_i + w^{(0)*}) - 1 + \xi_i^* \right) = 0$$

- This means that, for every point, either $\alpha_i^* = 0$, or $y_i \cdot ((\tilde{\mathbf{w}}^*)^T \mathbf{x}_i + w^{(0)*}) = 1 - \xi_i^*$
- The prediction has the same form as before, and the sum can again be computed only on the samples for which $\alpha_i^* \neq 0$, i.e.,

$$\hat{y}(\mathbf{x}) = \sum_{i \in \mathcal{S}} \alpha_i^* y_i \mathbf{x}_i^T \mathbf{x} + w^{(0)*}$$

Slack variables: Optimal values

- By re-writing the constraints, we have

$$\xi_i \geq 1 - y_i \cdot (\tilde{\mathbf{w}}^T \mathbf{x}_i + w^{(0)})$$

- After optimization, because we minimize $C \sum_{i=1}^N \xi_i$, we have
 - For the samples i that satisfy the original constraints $y_i \cdot (\tilde{\mathbf{w}}^*{}^T \mathbf{x}_i + w^{(0)*}) \geq 1$, $\xi_i^* = 0$ (because $\xi_i \geq 0$)
 - For the samples i that don't, $\xi_i^* = 1 - y_i \cdot (\tilde{\mathbf{w}}^*{}^T \mathbf{x}_i + w^{(0)*})$

SVM: Loss function

- This observation allows us to re-write the SVM problem as

$$\min_{\tilde{\mathbf{w}}, w^{(0)}} \frac{1}{2C} \|\tilde{\mathbf{w}}\|^2 + \sum_{i=1}^N \max(0, 1 - y_i \cdot (\tilde{\mathbf{w}}^T \mathbf{x}_i + w^{(0)}))$$

- The term

$$\max(0, 1 - y_i \cdot (\tilde{\mathbf{w}}^T \mathbf{x}_i + w^{(0)}))$$

is referred to as the *hinge loss*

Linear models

- We have seen 3 ways to use a linear model for classification:
 - Least-square classification
 - Logistic regression
 - Support Vector Machine
- All three methods rely on the **same** linear model, yet they yield different classifiers
- Question: Why? What is the main difference between them?

Lecture 5: Linear Models for Classification (Part 3)

Recap: Linear model

- In dimension D , we can write

$$y = w^{(0)} + w^{(1)}x^{(1)} + w^{(2)}x^{(2)} + \dots + w^{(D)}x^{(D)} = \mathbf{w}^T \begin{bmatrix} x^{(1)} \\ x^{(2)} \\ \vdots \\ x^{(D)} \\ 1 \end{bmatrix}$$

- In short, we can write

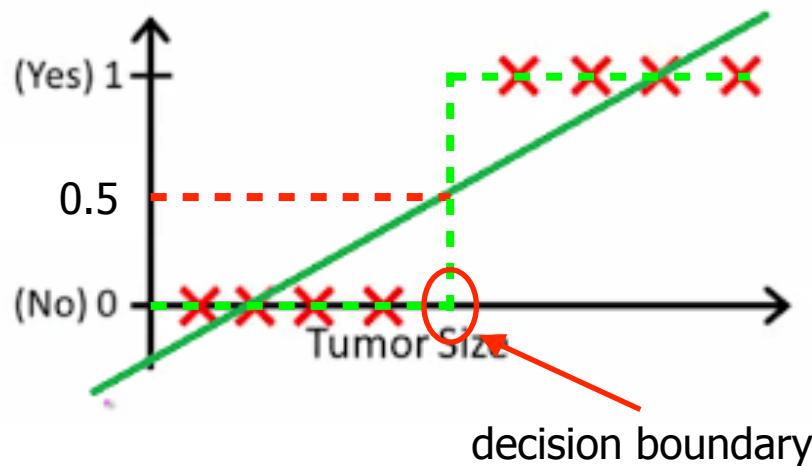
$$y = \mathbf{w}^T \mathbf{x}$$

with $\mathbf{x} \in \mathbb{R}^{D+1}$, where the extra dimension contains a 1 to account for $w^{(0)}$

Recap: Binary classification as regression

- Example (from Andrew Ng's course): Classify a tumor as benign vs malignant
 - 1D input (tumor size), 1D output (malignant vs benign)
 - Using a linear model, we can predict the label as $\hat{y} = w^{(1)}x + w^{(0)}$

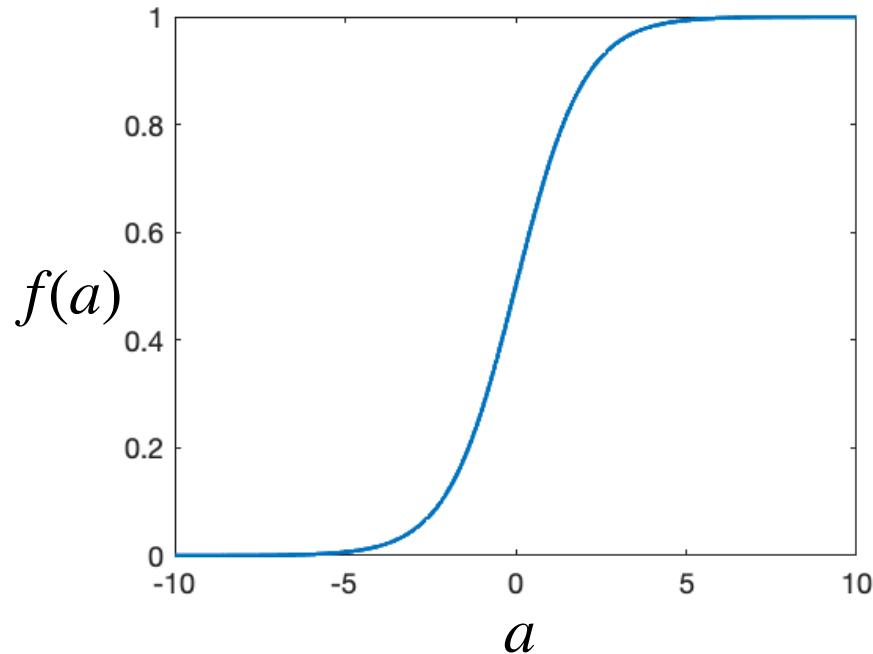
$$\text{label} = \begin{cases} 1 & \text{if } \hat{y} \geq 0.5 \\ 0 & \text{otherwise} \end{cases}$$



Recap: Adding non-linearity

- To model a classification problem, one can use a smooth approximation of the step function, such as the *logistic sigmoid function*

$$f(a) = \frac{1}{1 + \exp(-a)}$$



Recap: Logistic regression

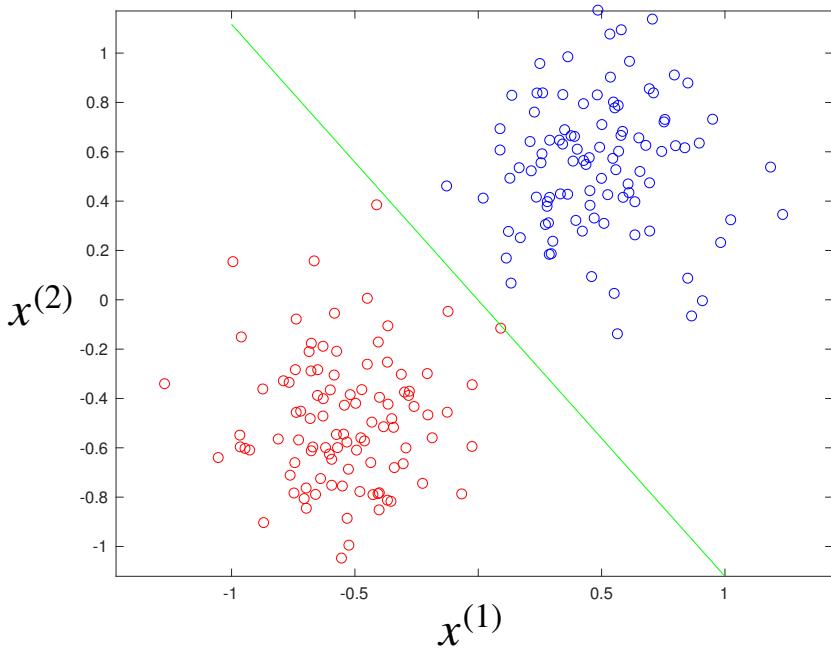
- To better model the underlying binary classification problem, we can pass the output of the linear model through a logistic function

$$\hat{y}(\mathbf{x}) = \sigma(\mathbf{w}^T \mathbf{x}) = \frac{1}{1 + \exp(-\mathbf{w}^T \mathbf{x})}$$

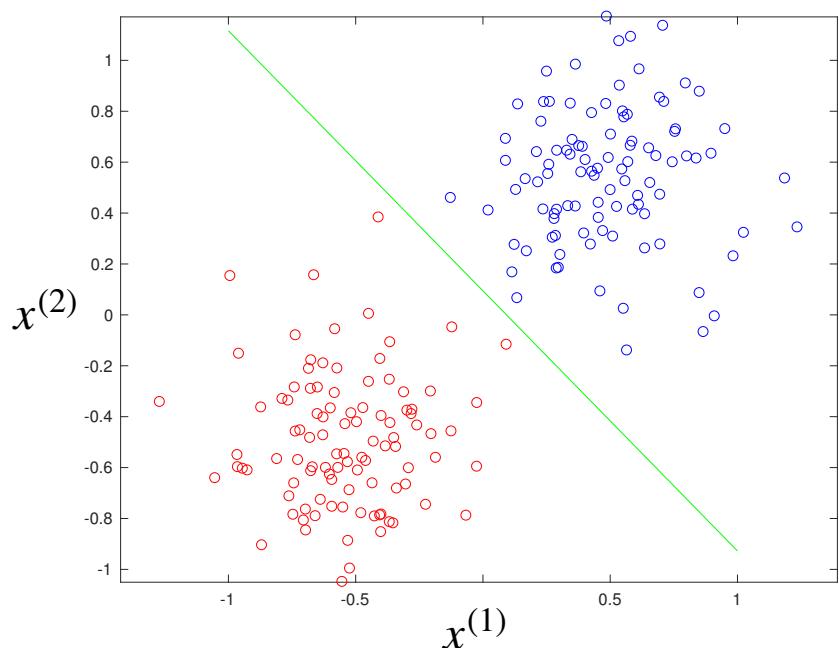
- Such a prediction can be interpreted as the probability that \mathbf{x} belongs to the positive class (or as a score for the positive class)
- The probability to belong to the negative class (or score for the negative class) is then computed as $1 - \hat{y}(\mathbf{x})$

Recap: Decision boundaries

- Different classifiers have different decision boundaries



Least-square classification



Logistic regression

- Can we define what a good decision boundary is?

Recap: Distance to the decision boundary

- The signed distance from a point \mathbf{x} to the boundary is given by its prediction $\hat{y}(\mathbf{x})$ and the parameters $\tilde{\mathbf{w}}$ as

$$r = \frac{\hat{y}(\mathbf{x})}{\|\tilde{\mathbf{w}}\|}$$

- We can use this to find a classifier whose decision boundary is as far as possible from all the points

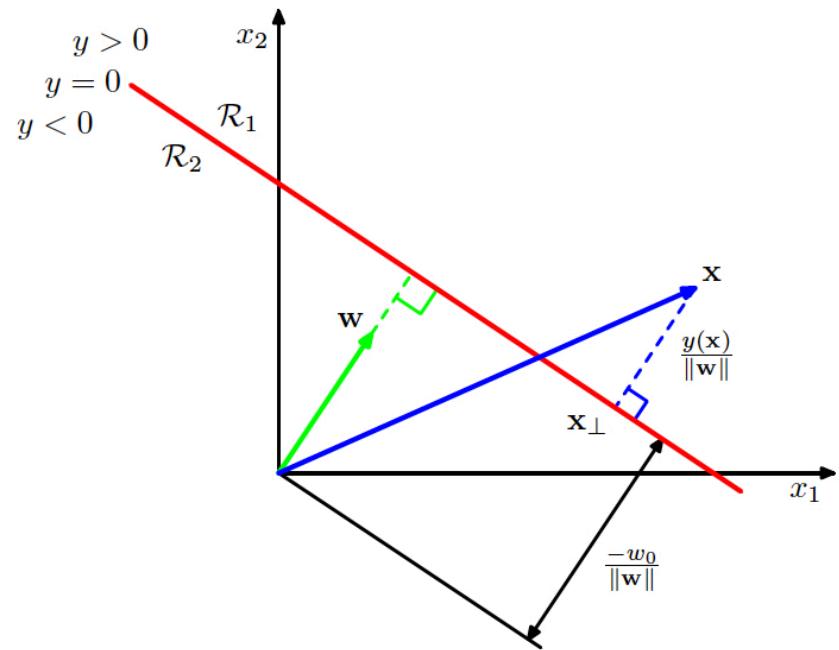


Figure from Bishop, Chapter 4.1.1

(\mathbf{w} in the fig. corresponds to $\tilde{\mathbf{w}}$)

Recap: Support Vector Machine

- The SVM formulation

$$\min_{\tilde{\mathbf{w}}, w^{(0)}} \frac{1}{2} \|\tilde{\mathbf{w}}\|^2$$

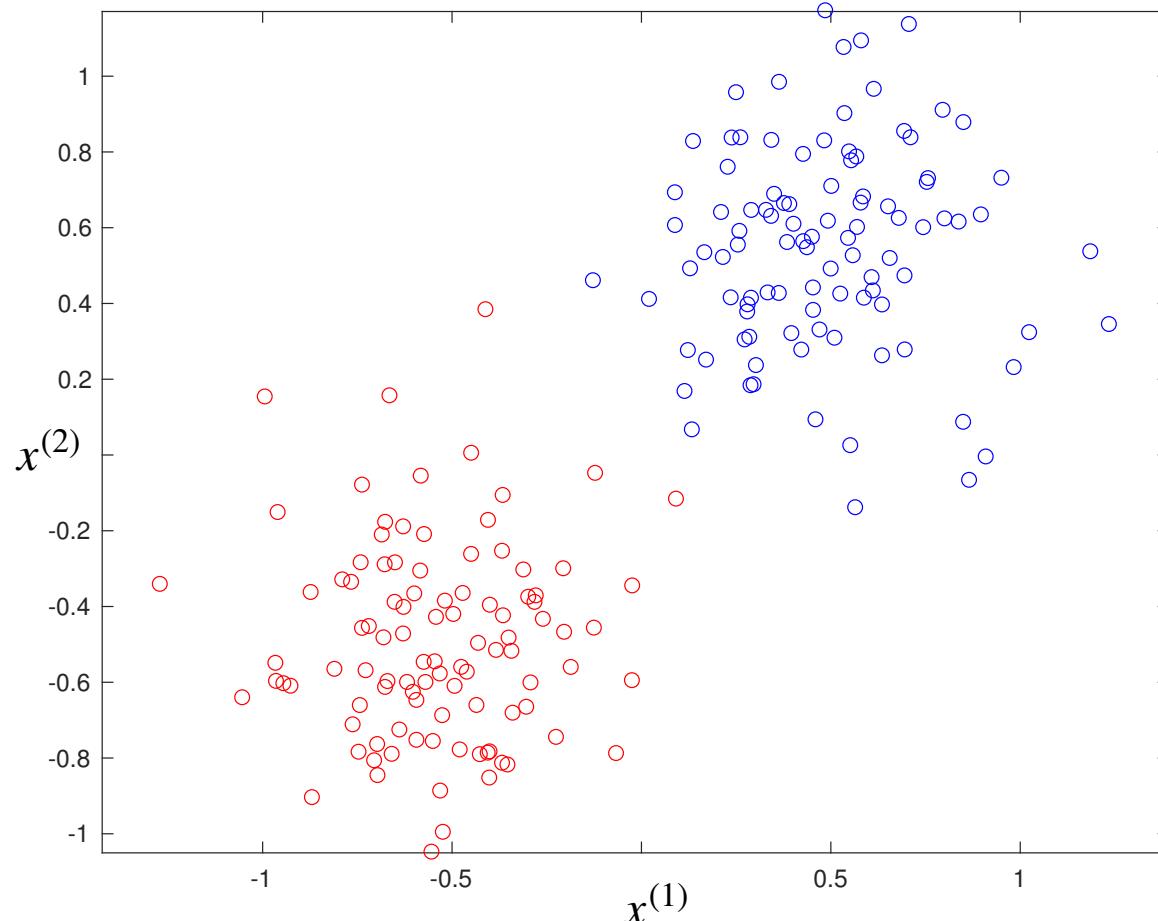
$$\text{subject to } y_i \cdot (\tilde{\mathbf{w}}^T \mathbf{x}_i + w^{(0)}) \geq 1, \quad \forall i$$

is a quadratic program

- Quadratic objective function, with linear constraints
- Here, the problem is convex, and can be solved to optimality with standard iterative solvers

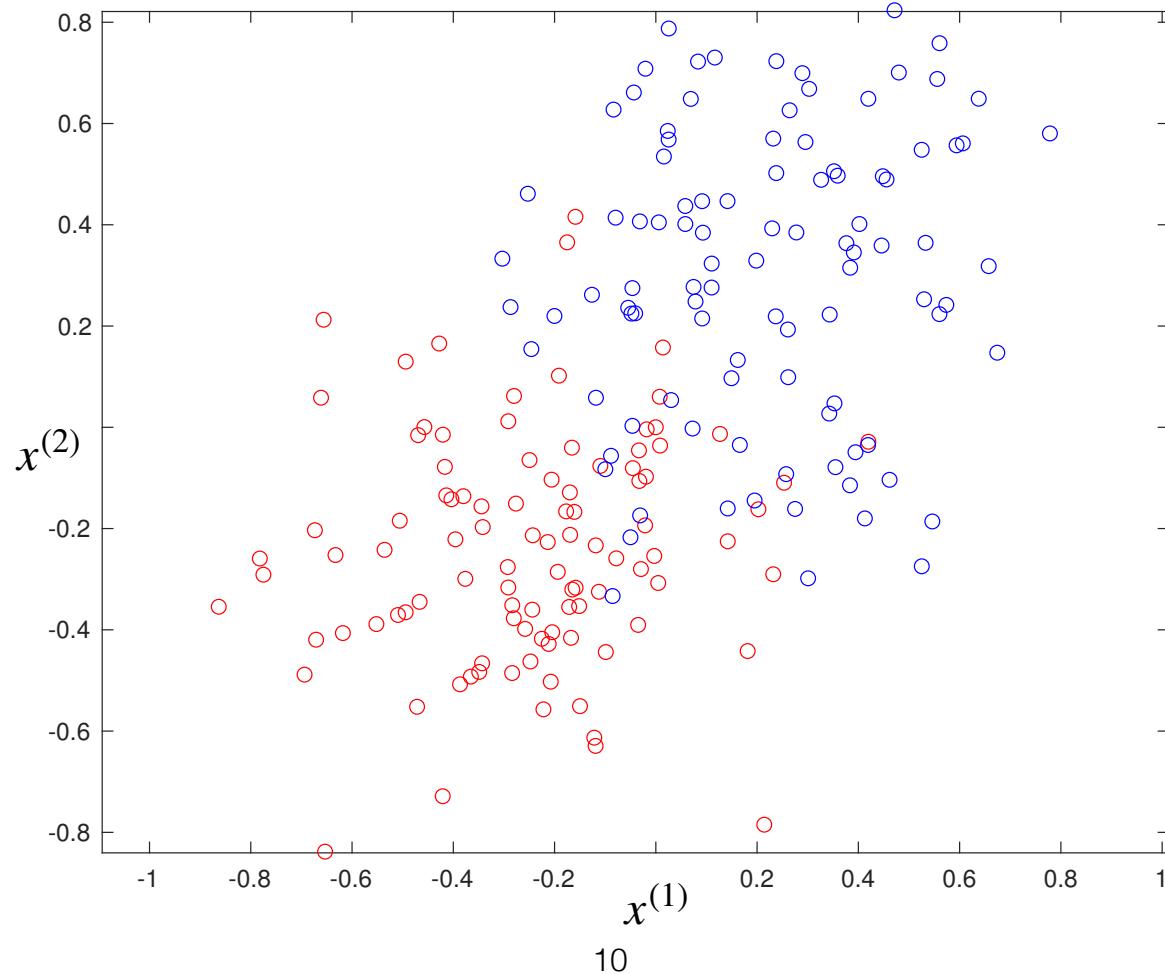
Recap: Overlapping classes

- In practice, the data essentially never looks like this



Recap: Overlapping classes

- but rather like this



Recap: Slack variables

- To handle this, we introduce an additional slack variable ξ_i for each sample
- We then write the SVM problem as

$$\begin{aligned} & \min_{\tilde{\mathbf{w}}, w^{(0)}, \{\xi_i\}} \frac{1}{2} \|\tilde{\mathbf{w}}\|^2 + C \sum_{i=1}^N \xi_i \\ & \text{subject to } y_i \cdot (\tilde{\mathbf{w}}^T \mathbf{x}_i + w^{(0)}) \geq 1 - \xi_i, \quad \forall i \end{aligned}$$

$$\xi_i \geq 0, \quad \forall i$$

where C sets the influence of the regularizer on the slack variables

Recap: SVM Prediction

- The SVM problem can be solved in its primal form or via its dual formulation in terms of the dual variables $\{\alpha_i\}$, $1 \leq i \leq N$
- Once the optimal solution $\{\alpha_i^*\}$ is obtained, one can use them to predict the label for a new sample \mathbf{x} as

$$\hat{y}(\mathbf{x}) = (\tilde{\mathbf{w}}^*)^T \mathbf{x} + w^{(0)*} = \sum_{i=1}^N \alpha_i^* y_i \mathbf{x}_i^T \mathbf{x} + w^{(0)*}$$

- Because for all samples that are not support vectors, we have $\alpha_i^* = 0$, the sum can be computed on the support vectors only, i.e., with \mathcal{S} the set of support vector indices,

$$\hat{y}(\mathbf{x}) = \sum_{i \in \mathcal{S}} \alpha_i^* y_i \mathbf{x}_i^T \mathbf{x} + w^{(0)*}$$

Recap: SVM: Loss function

- The SVM problem can also be written with the following loss

$$\min_{\tilde{\mathbf{w}}, w^{(0)}} \frac{1}{2C} \|\tilde{\mathbf{w}}\|^2 + \sum_{i=1}^N \max(0, 1 - y_i \cdot (\tilde{\mathbf{w}}^T \mathbf{x}_i + w^{(0)}))$$

- The term

$$\max(0, 1 - y_i \cdot (\tilde{\mathbf{w}}^T \mathbf{x}_i + w^{(0)}))$$

is referred to as the *hinge loss*

Exercise: Solutions

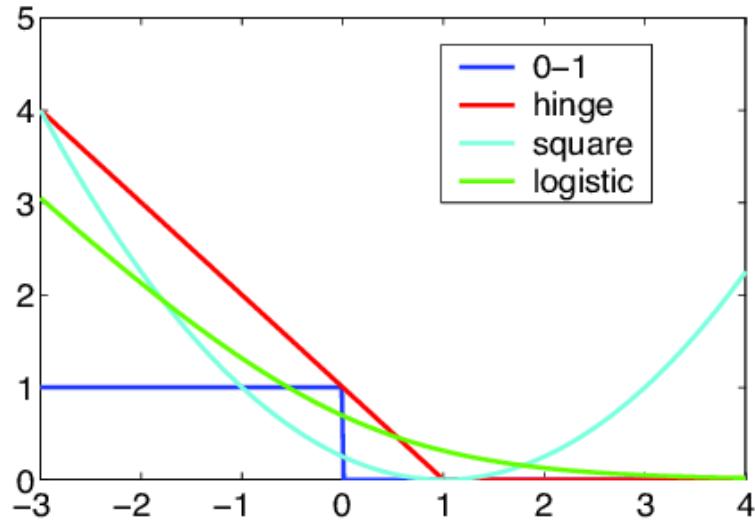
- The slack variables allow SVM to handle the case where the classes are not truly linearly separable but still close to being so. Describe two reasons why this can happen in practice.
 1. There is some noise in the data that makes samples close to the decision boundary move to the wrong side
 2. There are some mistakes in the ground-truth annotations of the training data

Linear models

- We have seen 3 ways to use a linear model for classification:
 - Least-square classification
 - Logistic regression
 - Support Vector Machine
- All three methods rely on the **same** linear model, yet they yield different classifiers
- Question: Why? What is the main difference between them?

Linear models

- The real difference between these methods is the loss function



- Square: Least-square classification
- Logistic: Logistic regression
- Hinge: SVM
- 0-1: Ideal

- Choosing an appropriate loss function has an impact on the resulting algorithm

Goals of today's lecture

- Move from binary to multiclass classification problems
- Understand how to represent multiple classes
- Extend the formulation of the linear classifiers seen so far to the multiclass scenario

From binary to multiclass classification

- So far, the three methods we have studied can only handle two classes: positive vs negative
- In general, one would typically like to discriminate between more than two categories. E.g., for image recognition



Label 1



Label 2

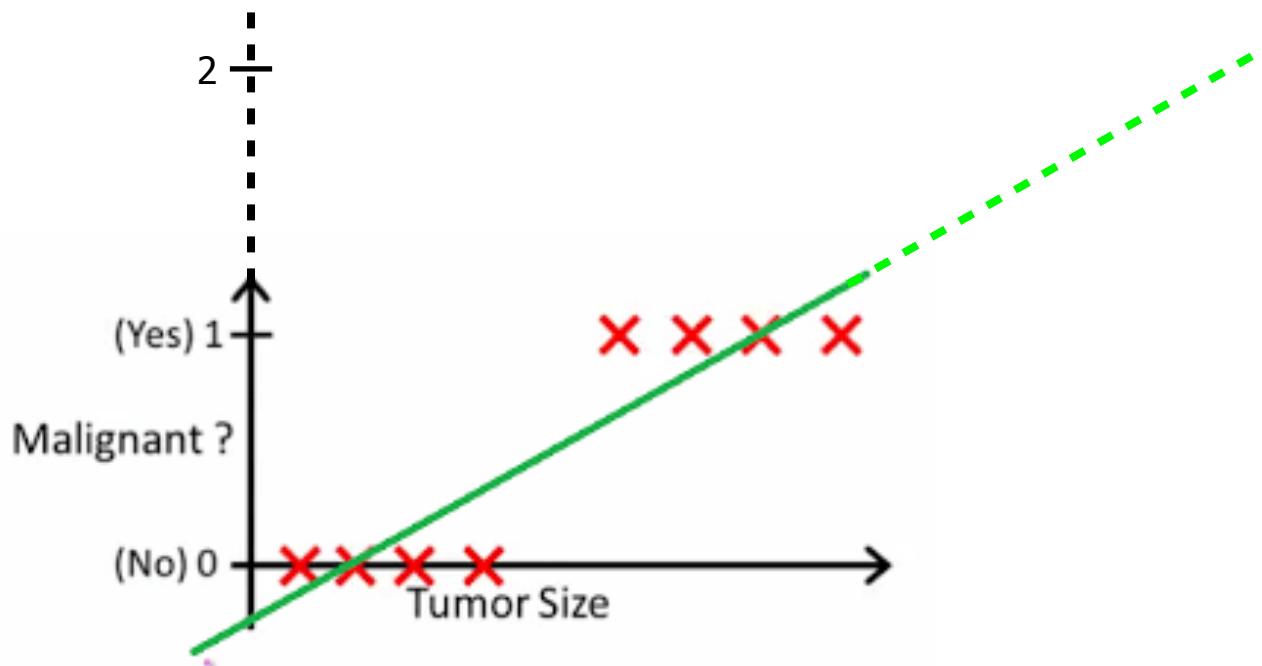


Label 3

- Let us know see how our three linear methods can handle this scenario
 - We will start with least-square classification

Least-square classification

- In essence, least-square classification treats the problem as a regression task
 - The model predicts a continuous value $\hat{y} = w^{(1)}x + w^{(0)}$
 - There is nothing that explicitly makes this value stop at 1, so we could continue the line further



Multiclass least-square classification: Naive approach

- The simplest way to go from the binary case to handling multiple classes would then be to use one integer value for each category



Label 1



Label 2



Label 3

- And round the prediction of the model to the nearest integer to obtain the predicted class value
- The same framework as for least-square binary classification could then directly apply

Multiclass least-square classification: Naive approach

- This, however, makes no sense, because:
 - There is no natural order between the different categories
 - E.g., mistaking a car for a tree would be less strongly penalized than mistaking it for a cat
 - This is because the least-square loss would be $(3 - 2)^2 = 1$ in the first case and $(3 - 1)^2 = 4$ in the second



Label 1



Label 2



Label 3

Dealing with multiple classes

- The most common approach to modeling a multi-class problem consists of encoding the class label as a vector with a single 1 at the index corresponding to the category and 0s elsewhere
 - In a 5-class problem, a sample in class 2 is represented as

$$\mathbf{y} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

- This is referred to as *one-hot encoding* (or 1-of-C encoding)

Multi-output linear model

- Predicting such a one-hot encoding is then similar to making the model output multiple values, as we saw for multi-output regression
- We therefore again use a matrix $\mathbf{W} \in \mathbb{R}^{(D+1) \times C}$, such that

$$\hat{\mathbf{y}}_i = \mathbf{W}^T \mathbf{x}_i = \begin{bmatrix} \mathbf{w}_{(1)}^T \\ \mathbf{w}_{(2)}^T \\ \vdots \\ \mathbf{w}_{(C)}^T \end{bmatrix} \mathbf{x}_i$$

where each $\mathbf{w}_{(j)}$ is a $(D + 1)$ -dimensional vector, used to predict one output dimension, i.e., the score for one class

Multi-class least-square classification

- Multi-class classification is then similar to a multi-output regression problem
- We can then write training as

$$\min_{\mathbf{W}} \sum_{i=1}^N \|\mathbf{W}^T \mathbf{x}_i - \mathbf{y}_i\|^2$$

where each \mathbf{y}_i is now a C -dimensional vector with a single one at the index corresponding to the class label, and zeros everywhere else

Multi-class least-square classification: Solution

- We can again group the inputs and the outputs in two matrices, as

$$\mathbf{X} = \begin{bmatrix} \mathbf{x}_1^T \\ \mathbf{x}_2^T \\ \vdots \\ \mathbf{x}_N^T \end{bmatrix} \in \mathbb{R}^{N \times D+1} \quad \mathbf{Y} = \begin{bmatrix} \mathbf{y}_1^T \\ \mathbf{y}_2^T \\ \vdots \\ \mathbf{y}_N^T \end{bmatrix} \in \mathbb{R}^{N \times C}$$

- Then, the solution has the same form as in the regression case (obtained by zeroing out the gradient w.r.t. \mathbf{W}), i.e.,

$$\mathbf{W}^* = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{Y} = \mathbf{X}^\dagger \mathbf{Y}$$

Multi-class least-square classification

- Given the optimal parameters, the prediction for a new sample is computed as

$$\hat{\mathbf{y}} = (\mathbf{W}^*)^T \mathbf{x}$$

- The sample is then assigned to class k if

$$\hat{\mathbf{y}}^{(k)} > \hat{\mathbf{y}}^{(j)}, \quad \forall j \neq k$$

- Or in other words the final label k is obtained as

$$k = \operatorname{argmax}_j \hat{\mathbf{y}}^{(j)}$$

Exercises

- You are training a multi-class least-square classifier to recognize cat, dog and car images. For the two images, \mathbf{x}_1 , \mathbf{x}_2 , below, what do the ground-truth vectors, \mathbf{y}_1 , \mathbf{y}_2 , look like numerically?
- Assuming the images are color images of size 32×32 , what is the size of the parameter matrix \mathbf{W} ?



\mathbf{x}_1



\mathbf{x}_2

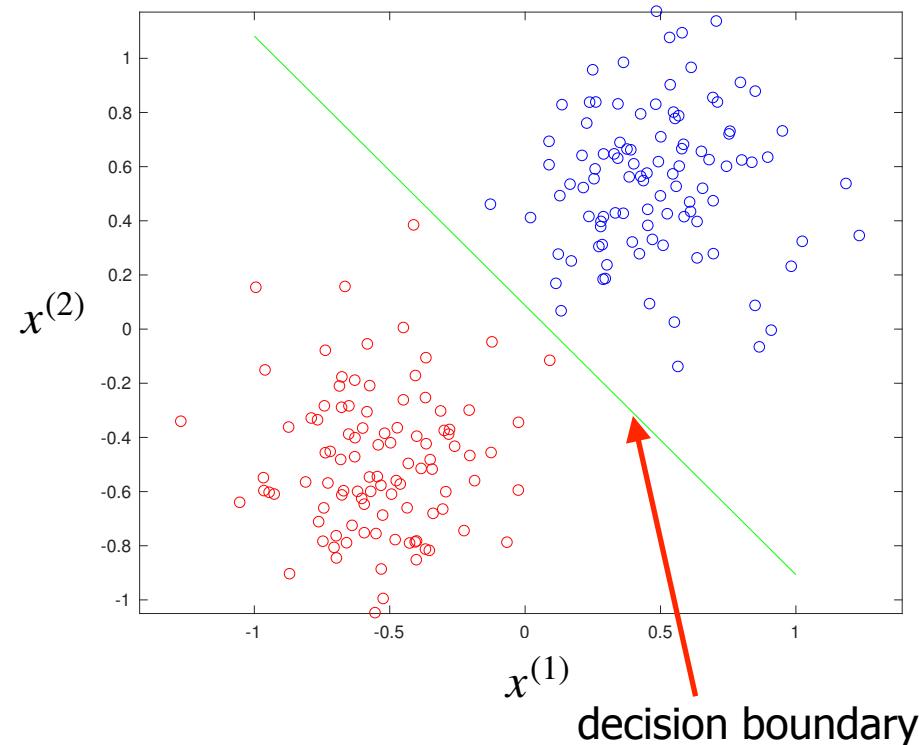
Interlude

Decision Boundary

(The following applies to all linear classification models, not just least-square classification)

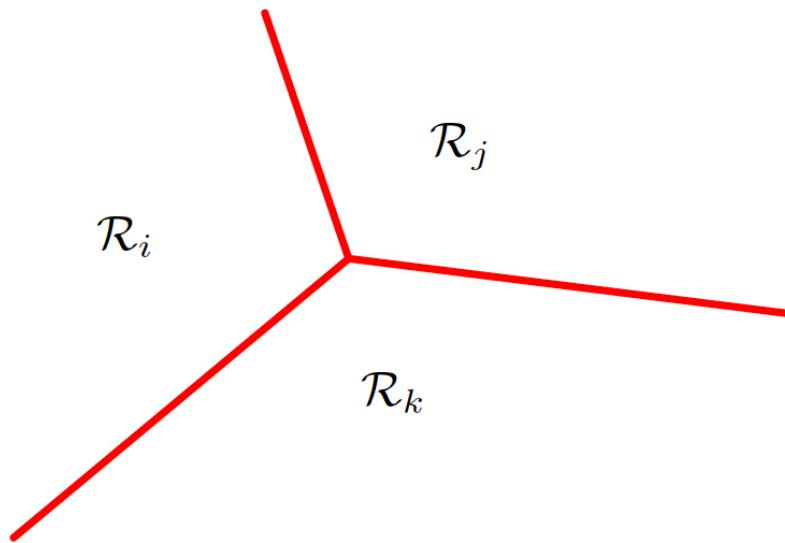
Recap: Decision boundary

- In the binary case, the point where the predicted label changes from 0 to 1 forms a *decision boundary*
 - With 2D inputs, it is a line



Decision boundaries: General definition

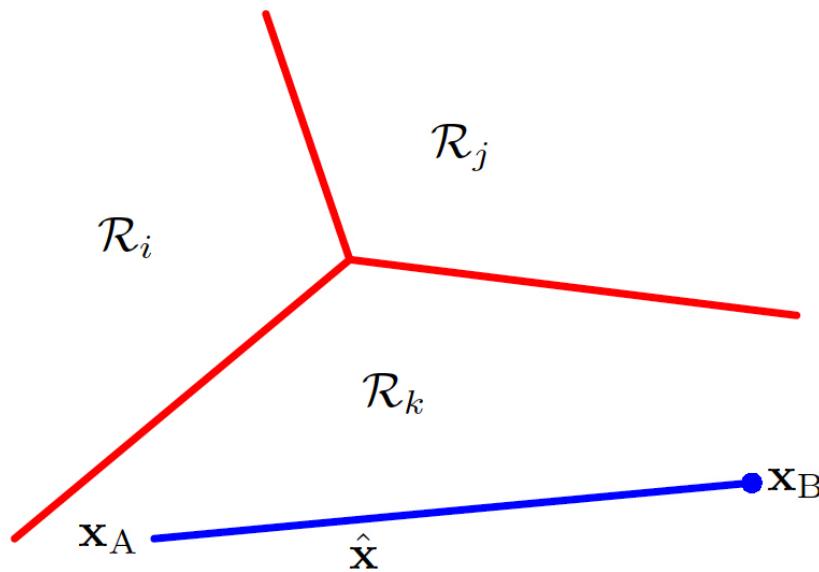
- The decision boundary between two classes k and j is defined as the set of points \mathbf{x} such that $\hat{y}^{(k)}(\mathbf{x}) = \hat{y}^{(j)}(\mathbf{x})$
- This corresponds to the $(D - 1)$ -dimensional hyperplane



- The binary case is a special case of this definition

Decision boundaries

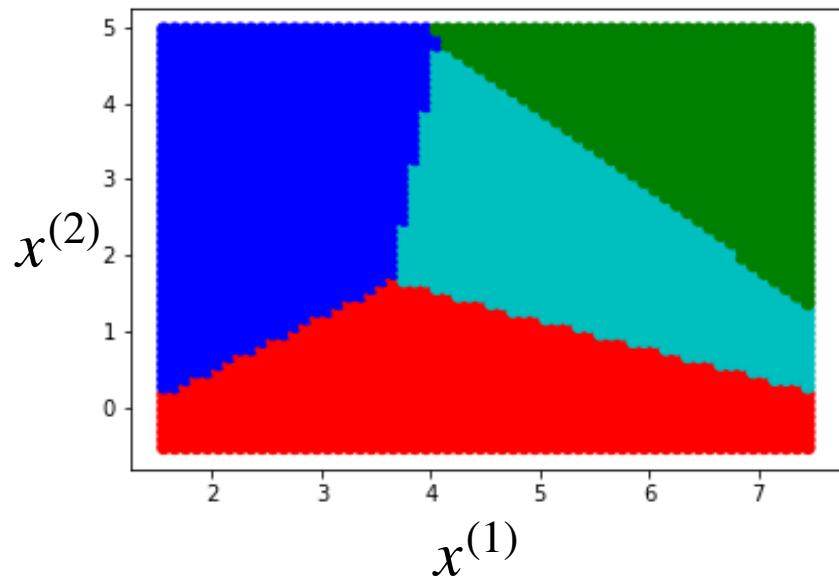
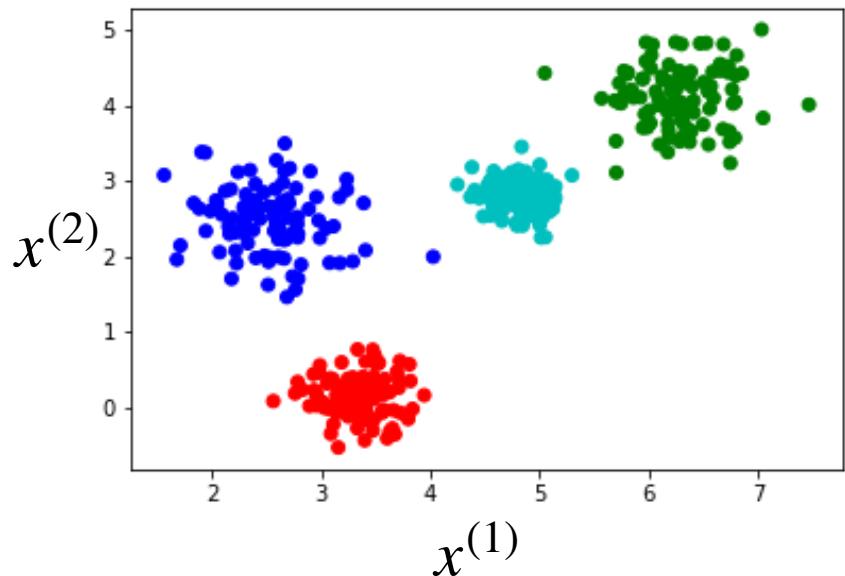
- The decision regions are singly-connected and convex



- Any point $\hat{\mathbf{x}}$ between two points \mathbf{x}_A and \mathbf{x}_B in the same region (e.g., k) will also be in the same region

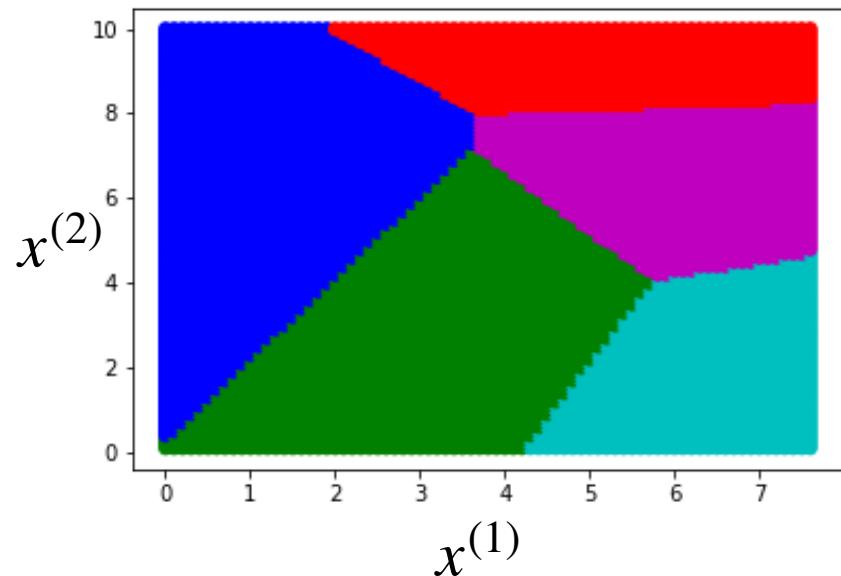
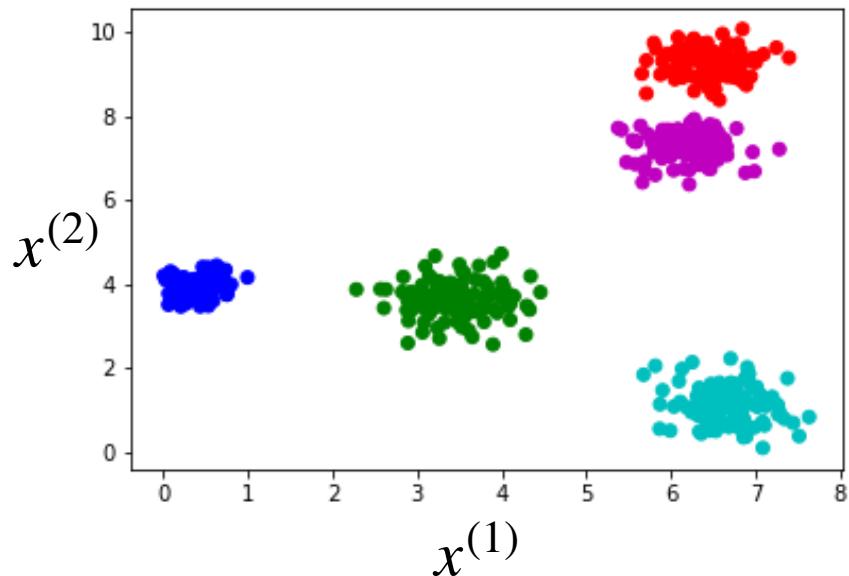
Decision boundaries for $C > 3$ classes

- $C = 4$ classes



Decision boundaries for $C > 3$ classes

- $C = 5$ classes



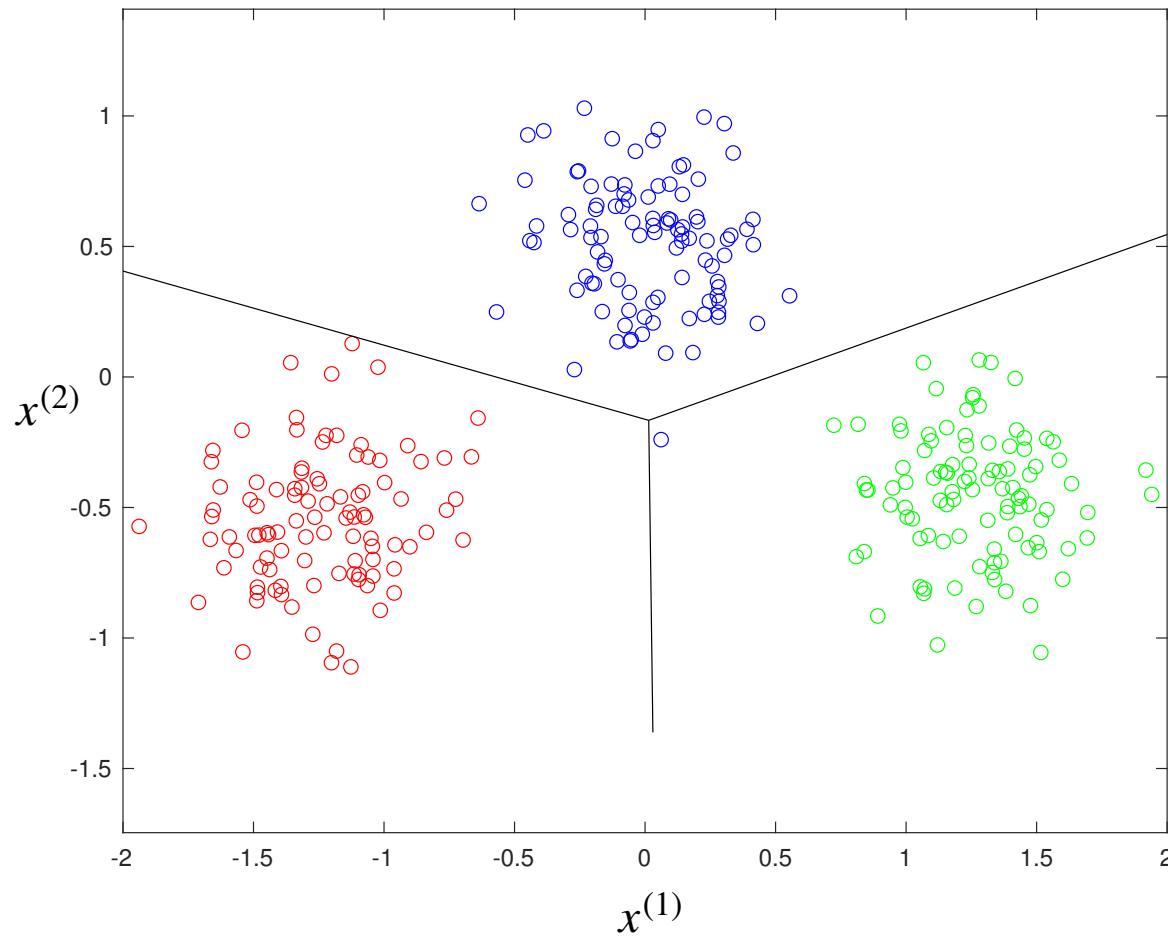
Code available on the Moodle page

End of the interlude

Let's look at a few least-square
classification examples

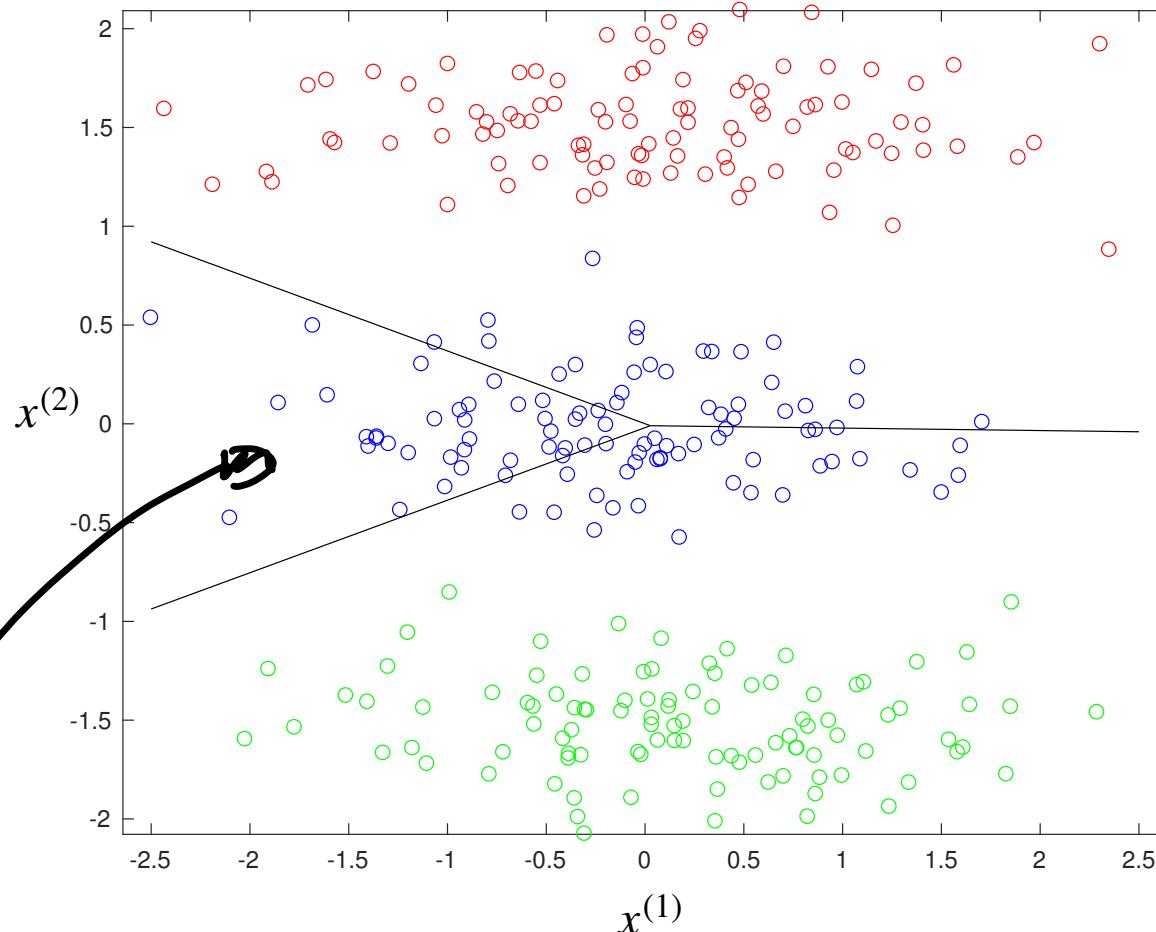
Multi-class least-square classification

- Example: 2D inputs, 3 classes



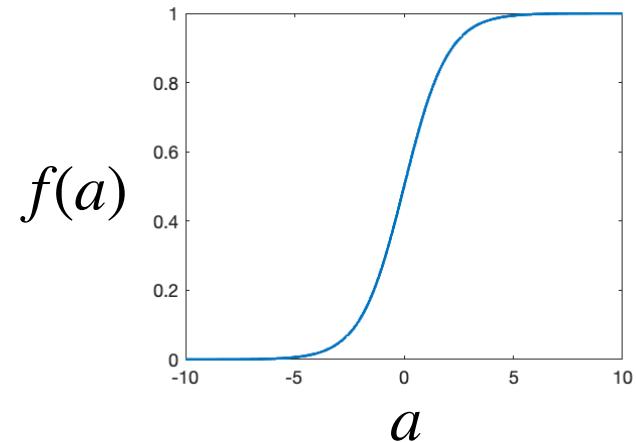
Multi-class least-square classification

- Failure case



Multi-class least-square classification: Drawbacks

- As in the binary case, multi-class least-square classification suffers from the fact that the loss function does not reflect the underlying classification task
- In the binary case, we addressed this by adding a nonlinearity to the output of the linear model via the logistic function, leading to logistic regression
- Let us now see how this extends to the multi-class scenario



Multi-class logistic regression

- As in the least-square classification case, we can use one-hot encodings to represent multi-class labels
- Then, we again need to use a matrix $\mathbf{W} \in \mathbb{R}^{(D+1) \times C}$ to represent the model parameters
- In this case, the probability for a class k is given by the softmax function

$$\hat{y}^{(k)}(\mathbf{x}) = \frac{\exp(\mathbf{w}_{(k)}^T \mathbf{x})}{\sum_{j=1}^C \exp(\mathbf{w}_{(j)}^T \mathbf{x})}$$

Multi-class logistic regression

- The empirical risk can then be derived from the multi-class version of the cross entropy, which yields

$$R(\mathbf{W}) = - \sum_{i=1}^N \sum_{k=1}^C y_i^{(k)} \ln \hat{y}_i^{(k)}$$

- As in the binary case, a single $y_i^{(k)}$ is 1 for every sample i , so we really have one term per training sample
- As in the binary case, training is done by minimizing this loss using gradient descent

Recap: Gradient descent

- Algorithm:
 1. Initialize \mathbf{w}_0 (e.g., randomly)
 2. While not converged
 - 2.1. Update $\mathbf{w}_k \leftarrow \mathbf{w}_{k-1} - \eta \nabla R(\mathbf{w}_{k-1})$
- η is a single value referred to as the learning rate
- Convergence can again be measured by
 - Thresholding the change in function value
 - Thresholding the change in parameters, e.g., $\|\mathbf{w}_{k-1} - \mathbf{w}_k\| < \delta_w$

Gradient descent: Dealing with a matrix

- With multiple classes, the parameters are shaped as a matrix $\mathbf{W} \in \mathbb{R}^{(D+1) \times C}$, not as a vector anymore
- This does not affect the algorithm:
 - It just means that the gradient itself will also be a matrix, i.e.

$$\nabla R(\mathbf{W}_{k-1}) \in \mathbb{R}^{(D+1) \times C}$$

- So the whole algorithm works with matrix entities
 - Convergence check can be done based on the Frobenius norm:
 $\|\mathbf{W}_{k-1} - \mathbf{W}_k\|_F < \delta_w$

Multi-class logistic regression: Gradient

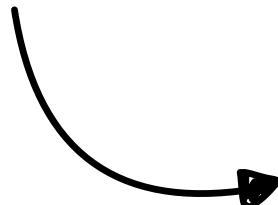
- By following the chain rule and using the gradient of the softmax function, we can derive the gradient of the multi-class cross-entropy as (details in Bishop Chap. 4.3.4)

$$\nabla R(\mathbf{W}) = \sum_{i=1}^N \mathbf{x}_i (\hat{\mathbf{y}}_i - \mathbf{y}_i)^T$$

- The gradient follows the same intuition as in the binary case: It multiplies the input by the error of the current prediction for each class $(\hat{y}_i^{(k)} - y_i^{(k)})$

Multi-class logistic regression: Prediction

- At test time, given a new input sample \mathbf{x} , the probability for any class k is computed via the softmax function as

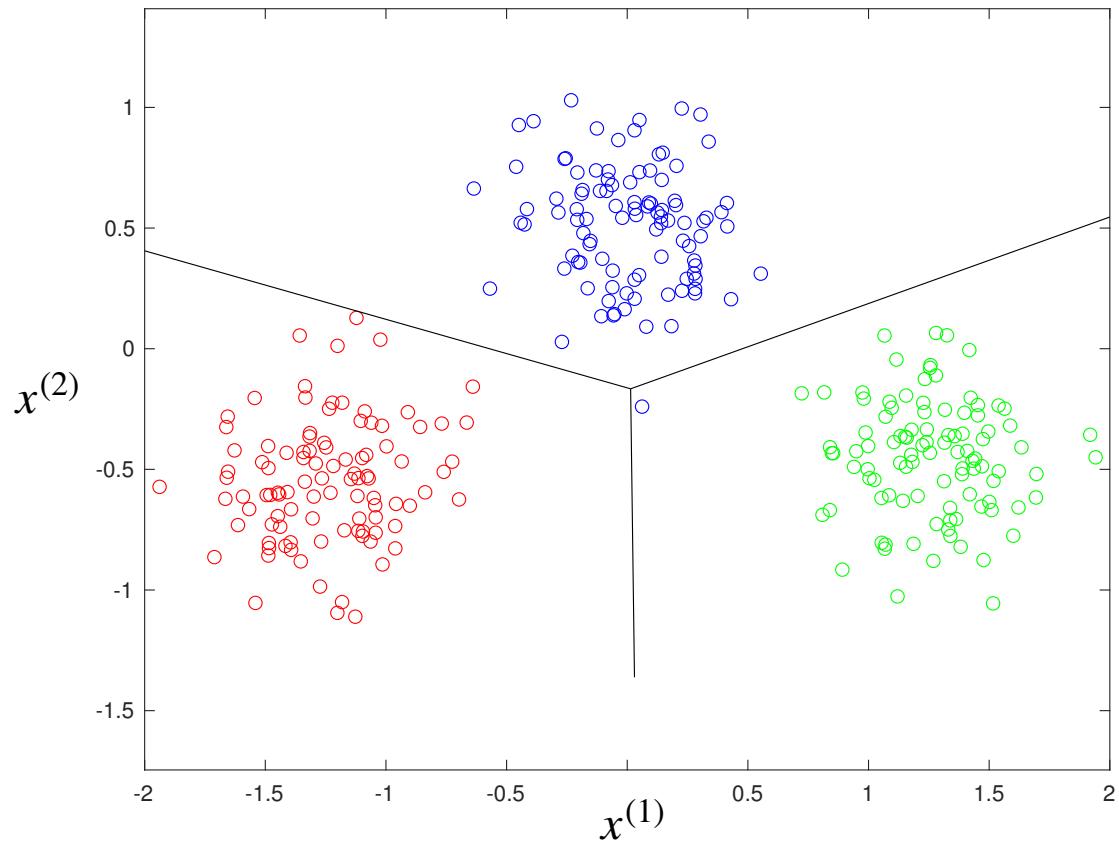

$$\hat{y}^{(k)}(\mathbf{x}) = \frac{\exp((\mathbf{w}_{(k)}^*)^T \mathbf{x})}{\sum_{j=1}^C \exp((\mathbf{w}_{(j)}^*)^T \mathbf{x})}$$

- The final class label is then predicted as

$$k = \operatorname{argmax}_j \hat{y}^{(j)}(\mathbf{x})$$

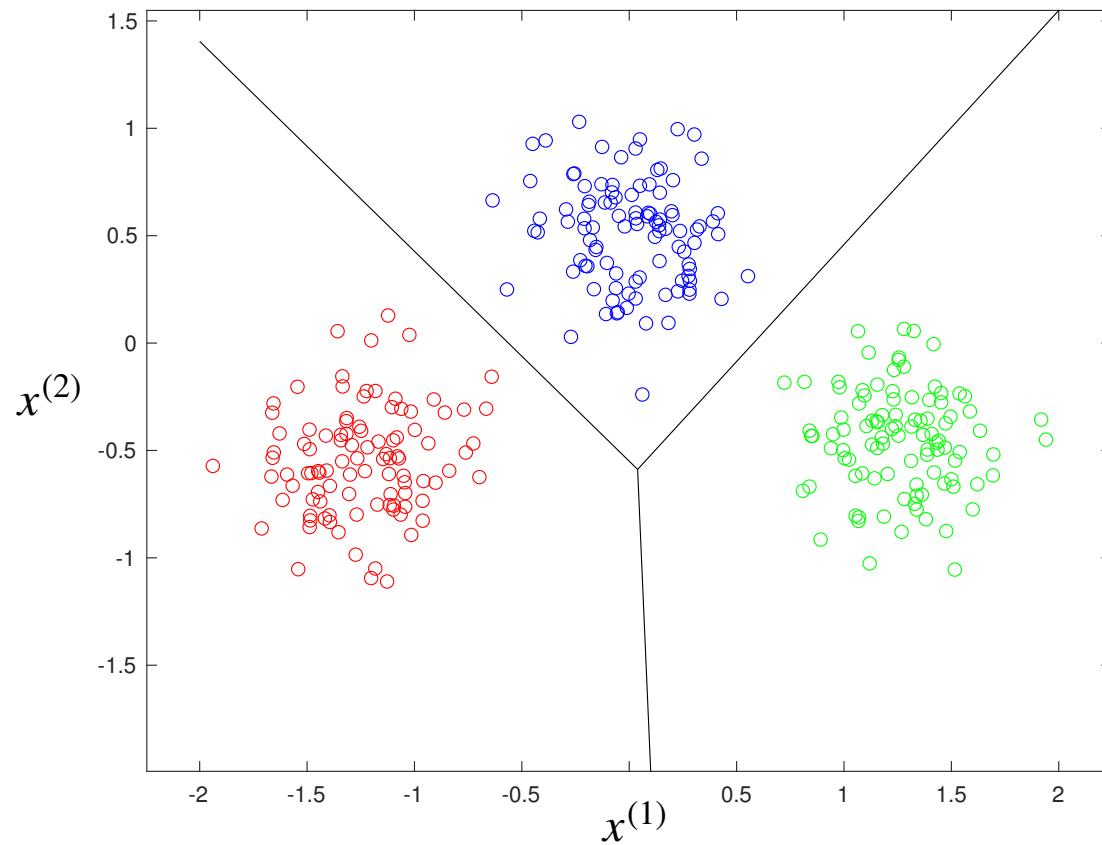
Multi-class logistic regression: Example

- 2D inputs, 3 classes: Least-square classification results



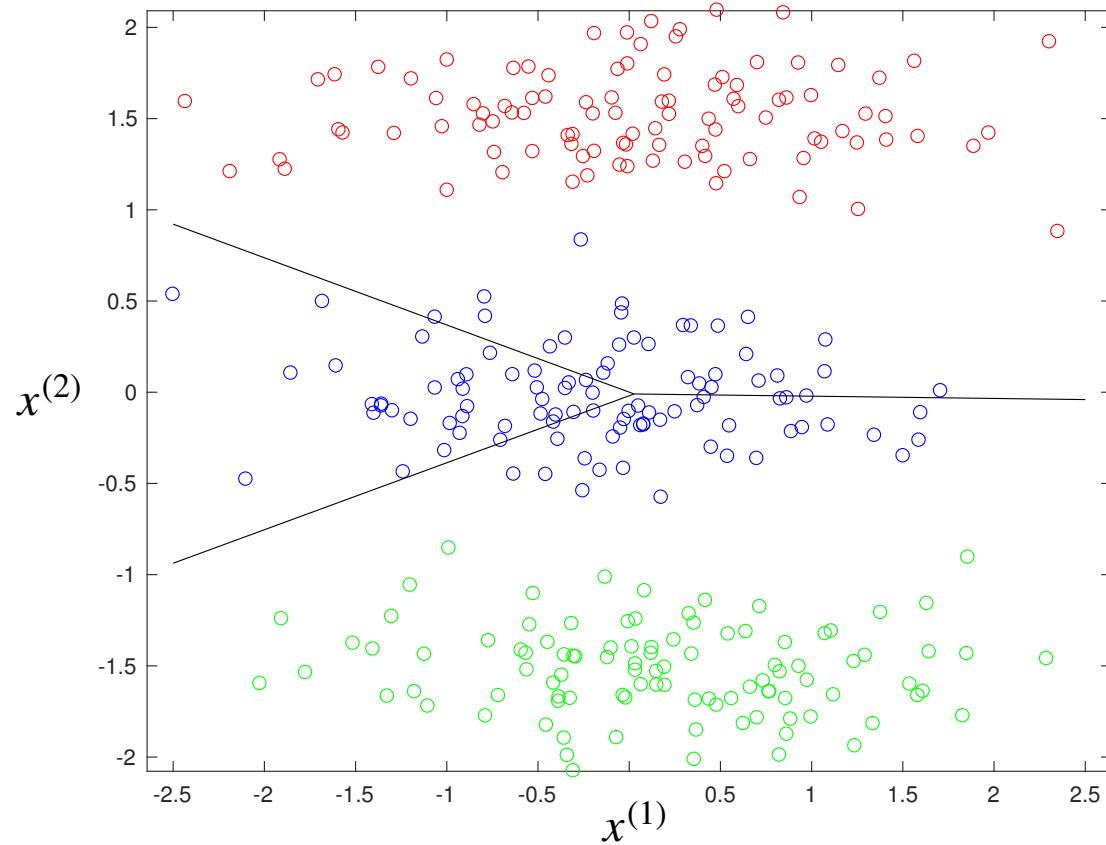
Multi-class logistic regression: Example

- 2D inputs, 3 classes: Logistic regression results



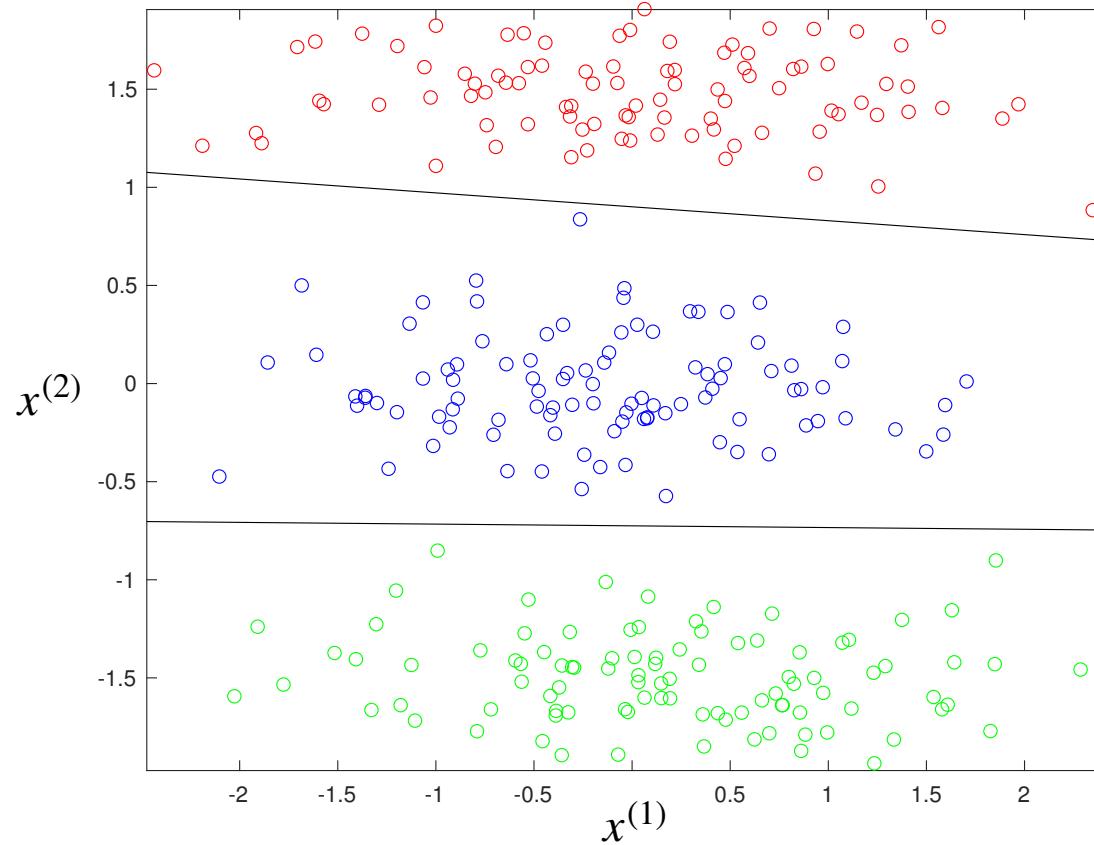
Multi-class logistic regression: Example

- 2D inputs, 3 classes: Least-square classification results



Multi-class logistic regression

- 2D inputs, 3 classes: Logistic regression results



Interlude

Evaluation metrics for multi-class classification

Recap: Classification evaluation

- Confusion matrix: Binary case (e.g., spam vs non-spam email)

True class → Hypothesized class V	Pos	Neg
Yes	TP	FP
No	FN	TN
	P=TP+FN	N=FP+TN

TP: True positive (number of samples correctly classified as positive)

FP: False positive (number of samples incorrectly classified as positive)

TN: True negative (number of samples correctly classified as negative)

FN: False negative (number of samples incorrectly classified as negative)

P: Total number of positive samples

N: Total number of negative samples

Recap: Classification metrics

- In the binary case, we saw the following metrics that can be computed from the confusion matrix:
 - Accuracy: Percentage of correctly classified samples
 - Precision: Percentage of samples classified as positives that are truly positives
 - Recall: Percentage of positives samples that are correctly classified as positives
 - False positive rate: Percentage of negative samples that are incorrectly classified as positives
 - F1 score: Single number that combines precision and recall

Classification evaluation

- Confusion matrix: Multi-class case
 - Numerical example for a dataset with 3 classes (UCI Iris dataset)
 - Note that the matrix is transposed (predicted labels as columns and actual labels as rows), which can also be used in practice

		Predicted		
		<i>Iris-setosa</i>	<i>Iris-versicolor</i>	<i>Iris-virginica</i>
Actual	<i>Iris-setosa</i>	12	1	1
	<i>Iris-versicolor</i>	0	16	0
	<i>Iris-virginica</i>	0	1	16

F
the correct predictions are on the diagonal

Metrics for multi class problems

- Accuracy:
 - A global accuracy can be obtained by summing the correct predictions for all classes (diagonal values of the confusion matrix), and dividing this by the total number of samples (sum of all entries in the confusion matrix)
- Precision, recall, FP-rate and F1-score:
 - The simplest approach to handling these metrics is to compute them for each individual class, and then average over the classes
 - In this case, each individual class is in turn considered as the “positive” class while the others act as the “negative” class
 - This does not account for class imbalance (some classes may have many more samples than others; more about this in a future lecture)
 - This can be handled by weighting the metric for each class by the proportion of samples from that class



End of the interlude

Let's look at a practical multi-class logistic regression example

Multi-class logistic regression: Application

- Predict the state of a fetus from cardiotocography: UCI Cardiotocography Dataset

- 21 input features
- Predict fetal state as Normal vs Suspect vs Pathologic

S. number	Name of the features	Description
1	LB	FHR baseline (beats per minute)
2	AC	Number of accelerations per second
3	FM	Number of fetal movements per second
4	UC	Number of uterine contractions per second
5	DL	Number of light decelerations per second
6	DS	Number of severe decelerations per second
7	DP	Number of prolonged decelerations per second
8	ASTV	Percentage of time with abnormal short term variability
9	MSTV	Mean value of short term variability
10	ALTV	Percentage of time with abnormal long term variability
11	MLTV	Mean value of long term variability
12	Width	Width of FHR histogram
13	Min	Minimum of FHR histogram
14	Max	Maximum of FHR histogram
15	Nmax	Number of histogram peaks
16	Nzeros	Number of histogram zeros
17	Mode	Histogram mode
18	Mean	Histogram mean
19	Median	Histogram median
20	Variance	Histogram variance
21	Tendency	Histogram tendency: -1 = left asymmetric; 0 = symmetric; 1 = right asymmetric

Multi-class logistic regression: Application

- Predict the state of a fetus from cardiotocography: UCI Cardiotocography Dataset
 - Logistic regression yields an accuracy of 92.1%
 - Confusion matrix

		Predicted label		
		0	1	2
True label	0	404	2	4
	1	24	48	0
	2	2	10	38

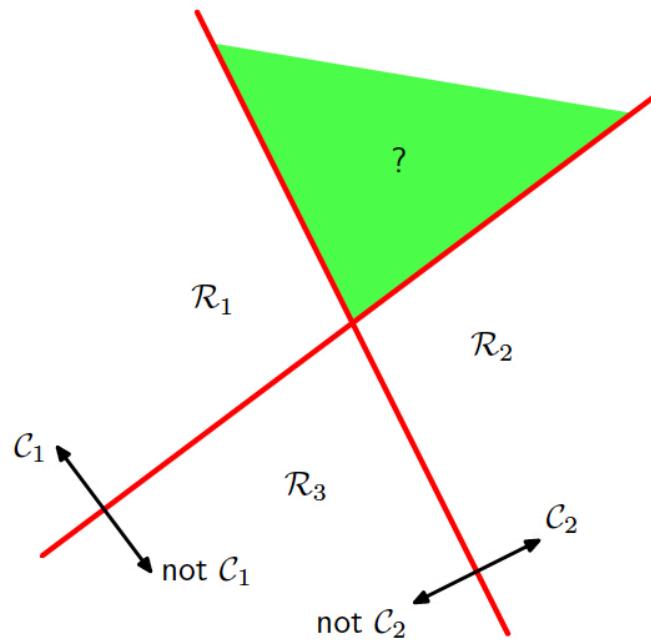
- Still 26 suspicious/pathologic samples classified as normal!

Dealing with multiple classes

- So far, we have seen how to extend binary least-square classification and binary logistic regression to the multi-class scenario. How about SVM?
- Unfortunately, there is no ideal solution to handle multiple classes in SVM
- Instead, one typically relies on one of the two solutions described in the next slides
 - Note that these solutions are quite general, not specific to SVM

Dealing with multiple classes

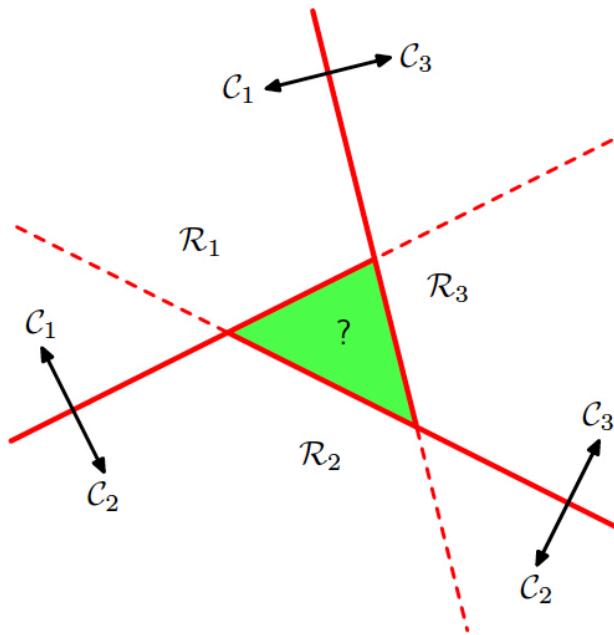
- To deal with multiple classes, one can use several binary classifiers
 - E.g., one-vs-rest: Classify each individual category vs all the other ones jointly



- This, however, leaves ambiguities (e.g., a sample can be classified as being from both class 1 and class 2)

Dealing with multiple classes

- To deal with multiple classes, one can use several binary classifiers
 - E.g., **one-vs-one**: Train one classifier for every pair of classes



- This also may result in **inconsistencies** (e.g., a sample can be classified as being from class 1, class 2 and class 3)

Multi-class SVM: Application

- Thyroid disease classification:
 - 30 attributes
 - 3 classes:
 - Hyperthyroid (excessive hormone production)
 - Hypothyroid (insufficient hormone production)
 - Normal

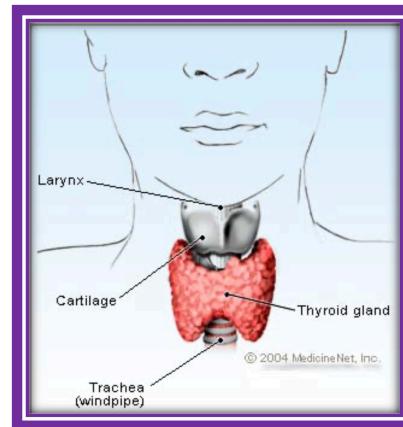


Image from Geetha et al., Global Journal of Computer Science and Technology , 2016

Multi-class SVM: Application

- Comparison of one-against-all (OAASVM) with one-against-one (OAOSVM)
 - Results from Chamasemani & Singh, International Conference on Bio-Inspired Computing: Theories and Applications, 2011

Method	Accuracy		Computation Time	
	Training	Test	Training	Test
OAASVM	95.6%	95.3%	212.4s	227.8s
OAOSVM	96.1%	94.48%	79.3s	81s

Multi-class SVM: Application

- Evaluation of the influence of the hyper-parameter C (strength of the regularizer on the slack variables)
 - Results from Chamasemani & Singh, International Conference on Bio-Inspired Computing: Theories and Applications, 2011

C	Accuracy		Computation Time	
	Training	Test	Training	Test
10	93.8%	92.5%	58.8s	52.9s
40	94.1%	93.8%	92.2s	93.3s
60	97.2%	96.4%	40.74s	40.21s
80	93.4%	92.9%	123.9s	134.6s

Comparing linear classifiers

- We have now seen 3 different linear (multi-class) classifiers:
 - Least-square classification
 - Logistic regression
 - SVM
- In practice, one almost never uses the least-square one
 - It does not reflect the underlying classification task, but is useful from a pedagogical standpoint
- Let us now compare logistic regression and SVM
 - The following results come from <https://martin-thoma.com/comparing-classifiers/>

Comparison on the Iris dataset

- UCI Iris dataset
 - 150 samples
 - 4 attributes (petal and sepal length and width)
 - 3 iris species (i.e., 3 classes)

	Accuracy	Training time	Testing time
SVM, linear	88.00%	0.0006s	0.0002s
Logistic Regression (C=1)	88.00%	0.0011s	0.0001s
Logistic Regression (C=1000)	92.00%	0.0010s	0.0002s

The value C is a hyper-parameter encoding “regularization” strength. We will talk about this in a future lecture; let’s ignore it for now.

- These results may suggest that logistic regression is better, however...

Comparison on the MNIST dataset

- MNIST handwritten digit recognition dataset
 - 60'000 training samples, 10'000 test samples
 - 28×28 images
 - 10 digits: 0,...,9 (i.e., 10 classes)

	Accuracy	Training time	Testing time
Linear SVM	94.16%	168.6950s	158.0101s
Logistic Regression (C=1)	91.47%	272.1309s	0.0531s
Logistic Regression (C=10000)	91.23%	1807.0624s	0.0529s

The value C is a hyper-parameter encoding “regularization” strength. We will talk about this in a future lecture; let’s ignore it for now

- The choice of the best classifier depends on the data

SVM confusion matrix on MNIST

- The errors are reasonably evenly spread across the classes
 - If it were not the case, one could try to re-weight the samples in the empirical risk to focus more strongly on the weaker classes

Linear SVM ↴

```
Classifier: linear SVM
Training time: 168.6950s
Testing time: 158.0101s
Confusion matrix:
```

```
[[2226    0     9     2     6    12     8     3    11     1]
 [ 1 2537   18     3     3     1     1     7    17     0]
 [ 12    16  2158    25    24     6    27    19    25     2]
 [  3     7    46  2188     4    47     3    18    27     5]
 [  2     5    19     1  2117     1     8     6     3    49]
 [ 18    13    11    73    20  1872    31     0    26     5]
 [ 20     6    22     1    10    30  2179     0     3     0]
 [  5    10    32    11    30     5     0  2268     5    51]
 [ 11    39    26    47    10    40     7     7  2018    10]
 [ 11     9     9    24    64     8     0    61    14  2189]]
```

```
Accuracy: 0.9416
```

Exercise

- You have trained a multi-class logistic regression classifier to recognize cat, dog and car images. For the two test images, \mathbf{x}_1 , \mathbf{x}_2 , below, assuming that the model predicts the correct class, what would you expect the outputs of the model, $\hat{\mathbf{y}}_1$, $\hat{\mathbf{y}}_2$, to look like numerically? Explain why.



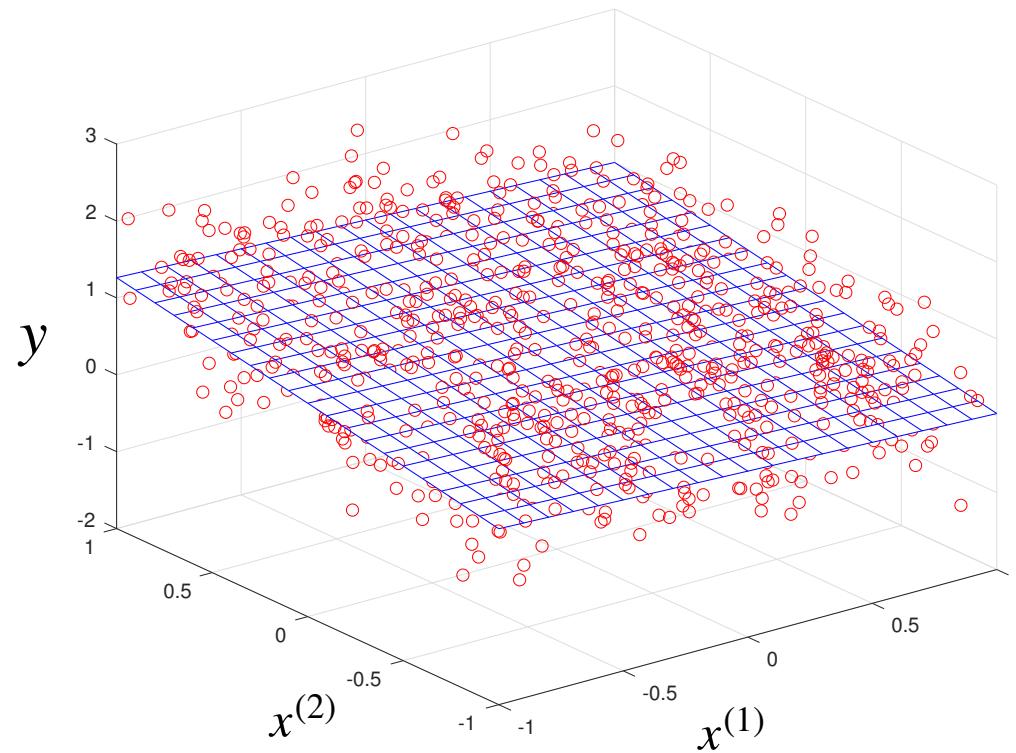
\mathbf{x}_1



\mathbf{x}_2

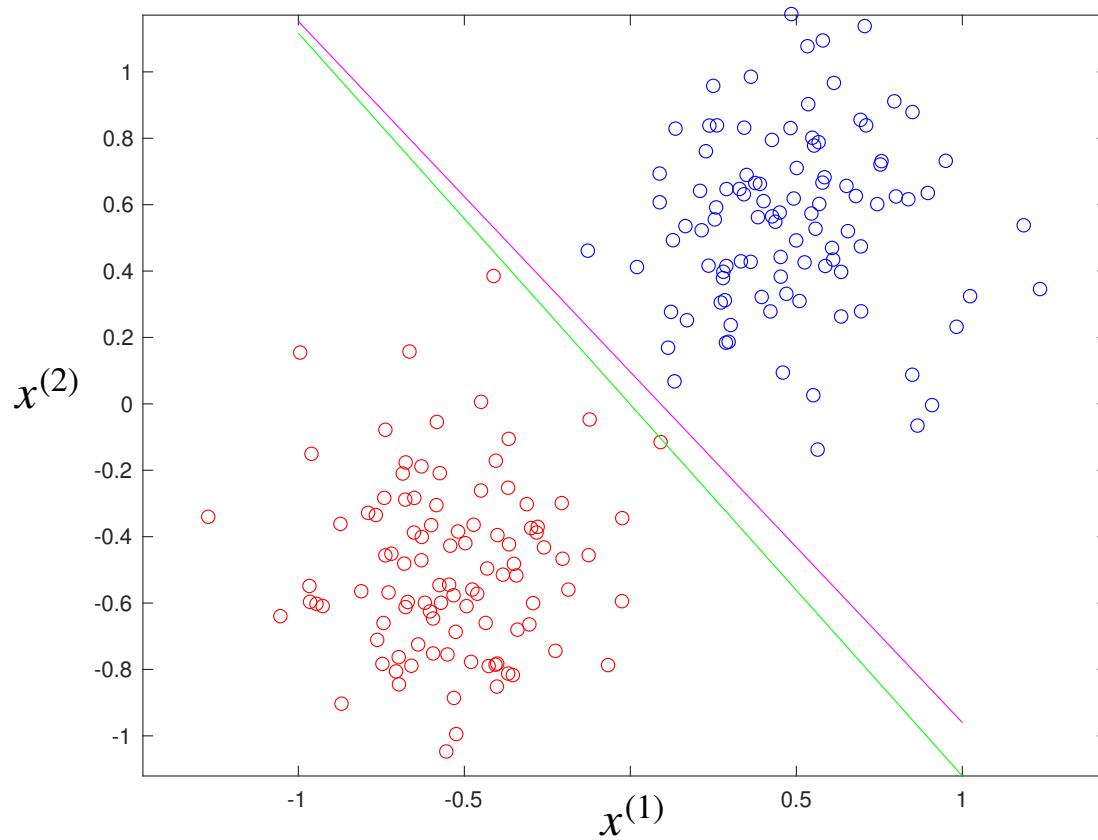
Until now

- Linear regression: Fit a linear model to the observations



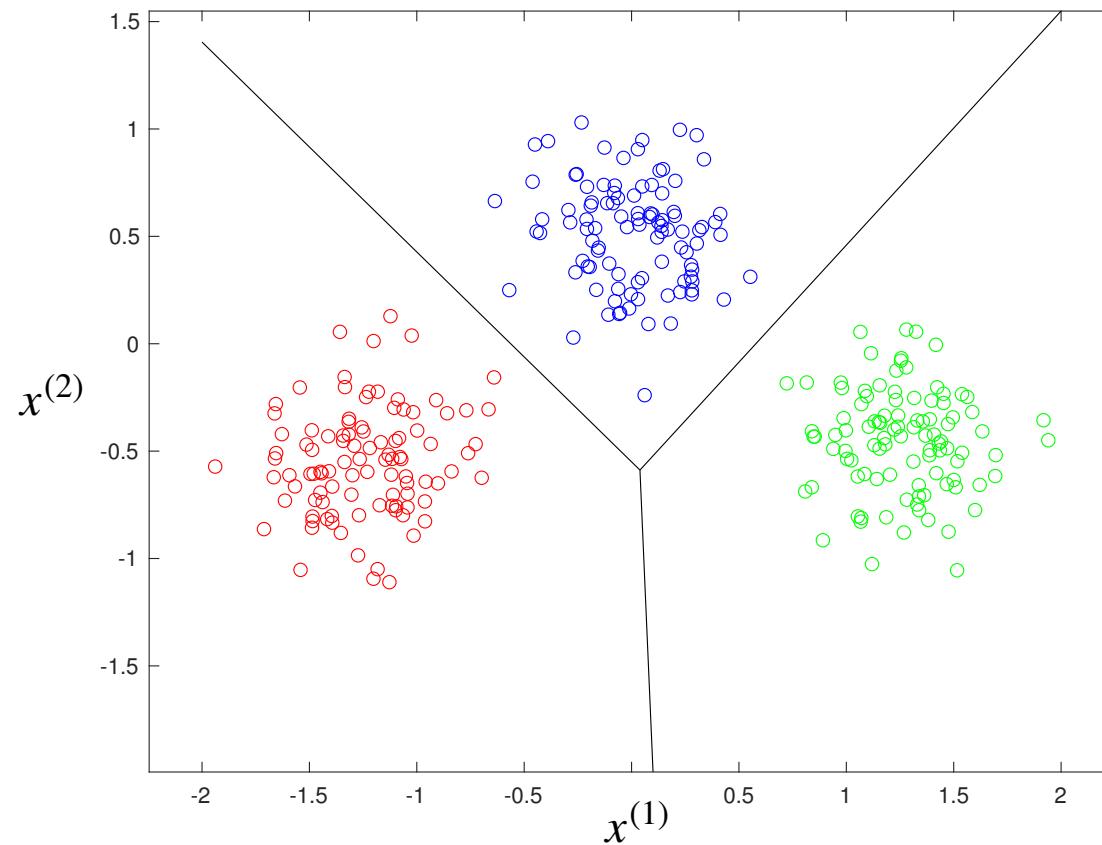
Until now

- Linear classification: Separate the classes by a linear model



Until now

- Linear classification: Separate the classes by a linear model
(multiple lines for the multi-class scenario)



From linear to nonlinear

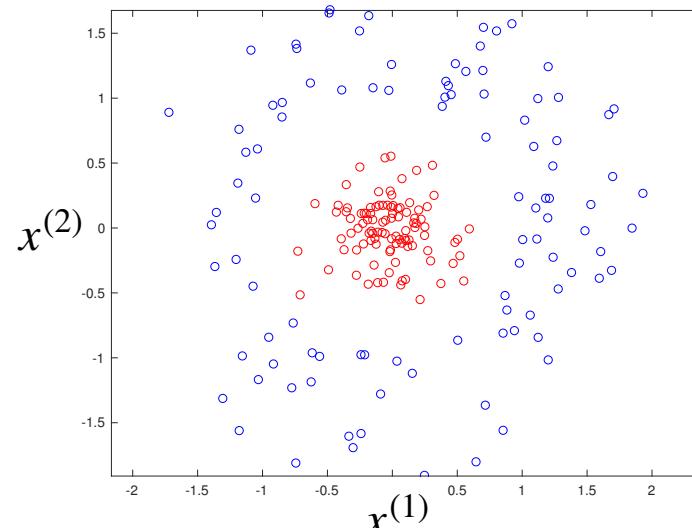
- Unfortunately, no linear model can directly fit this data

- 1D input, 2 classes (colors)



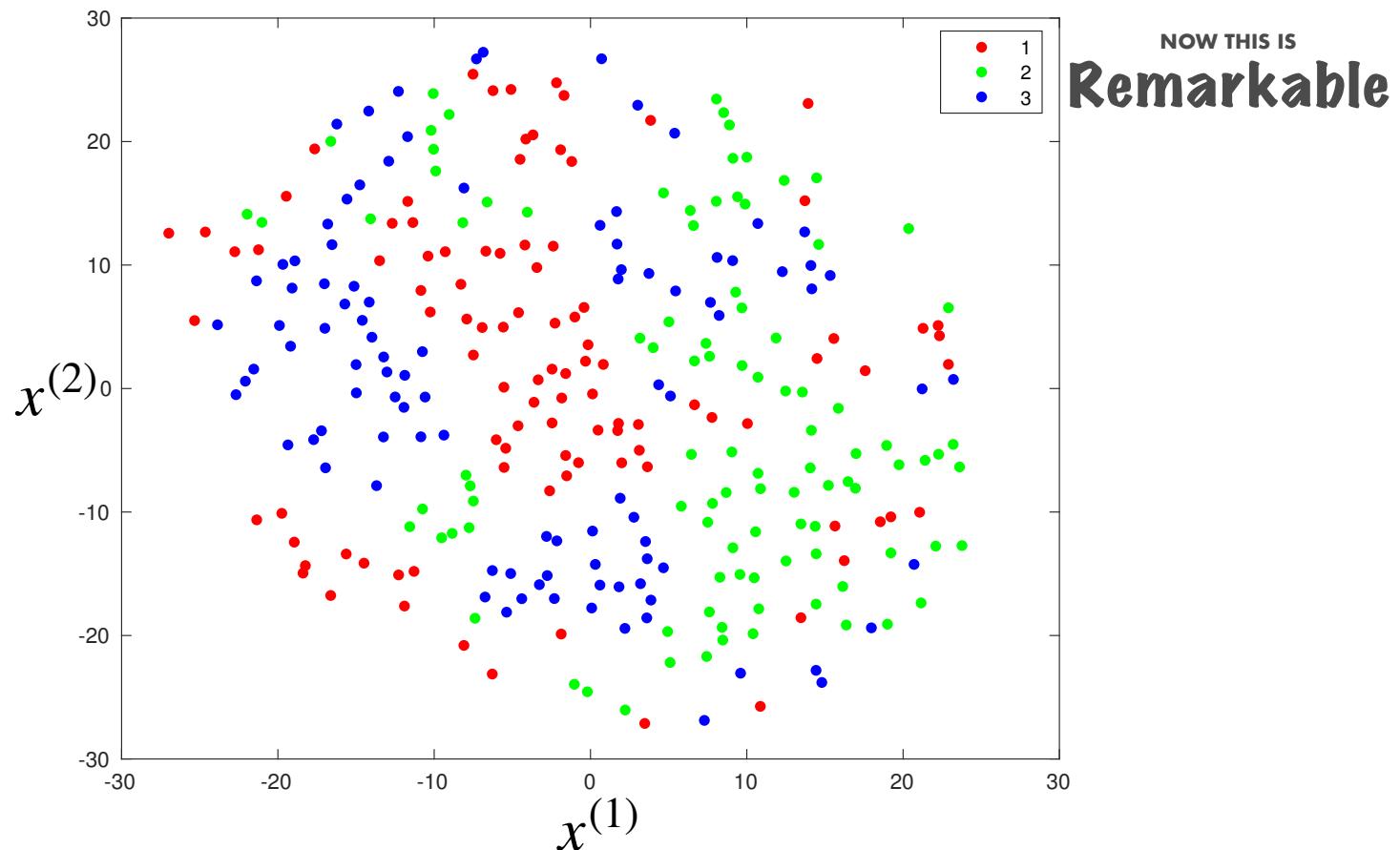
- Or this data

- 2D input, 2 classes (colors)



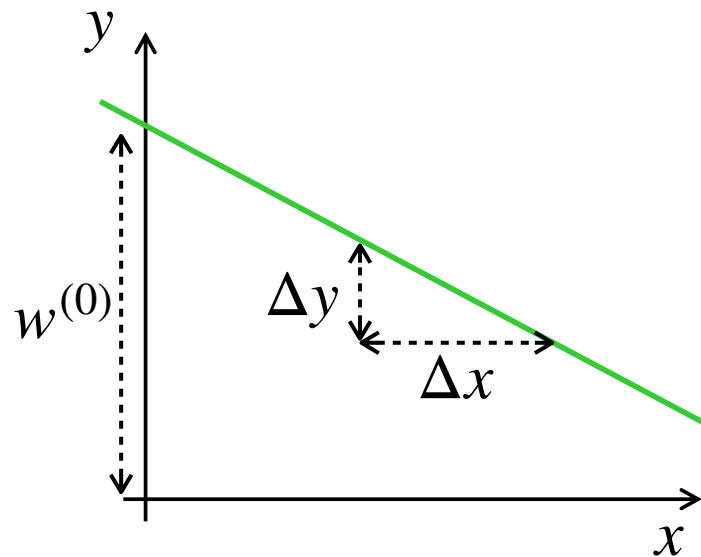
From linear to nonlinear

- Or this even more complicated data



Lecture 6: k-Nearest Neighbors & Feature Expansion

Recap: A simple parametric model: The line



- Defined by 2 parameters
 - The y -intercept $w^{(0)}$
 - The slope $w^{(1)} = \frac{\Delta y}{\Delta x}$
- Mathematically, a line is expressed as

$$y = w^{(1)}x + w^{(0)}$$

Recap: Hyperplane

- This can be generalized to higher dimensions
- In dimension D , we can write

$$y = w^{(0)} + w^{(1)}x^{(1)} + w^{(2)}x^{(2)} + \dots + w^{(D)}x^{(D)} = \mathbf{w}^T \begin{bmatrix} x^{(1)} \\ x^{(2)} \\ \vdots \\ x^{(D)} \\ 1 \end{bmatrix}$$

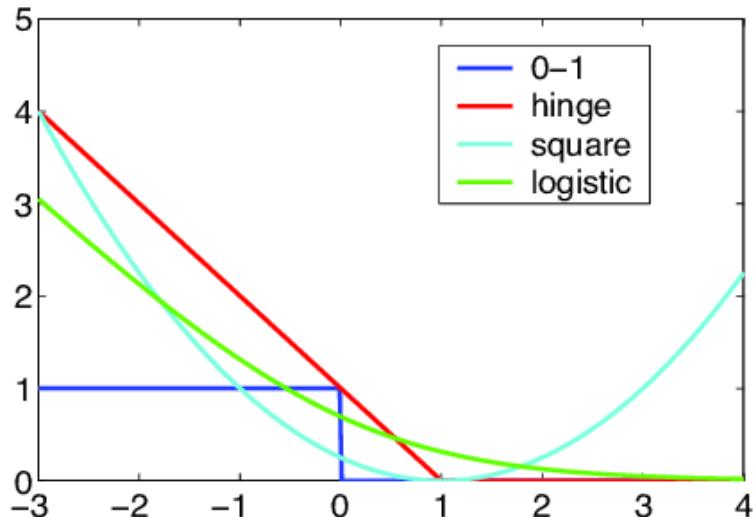
- Ultimately, whatever the dimension, we can write

$$y = \mathbf{w}^T \mathbf{x}$$

with $\mathbf{x} \in \mathbb{R}^{D+1}$, where the extra dimension contains a 1 to account for $w^{(0)}$

Recap: Linear models

- We have seen 3 algorithms that use the same linear model
- The real difference between these methods is the loss function

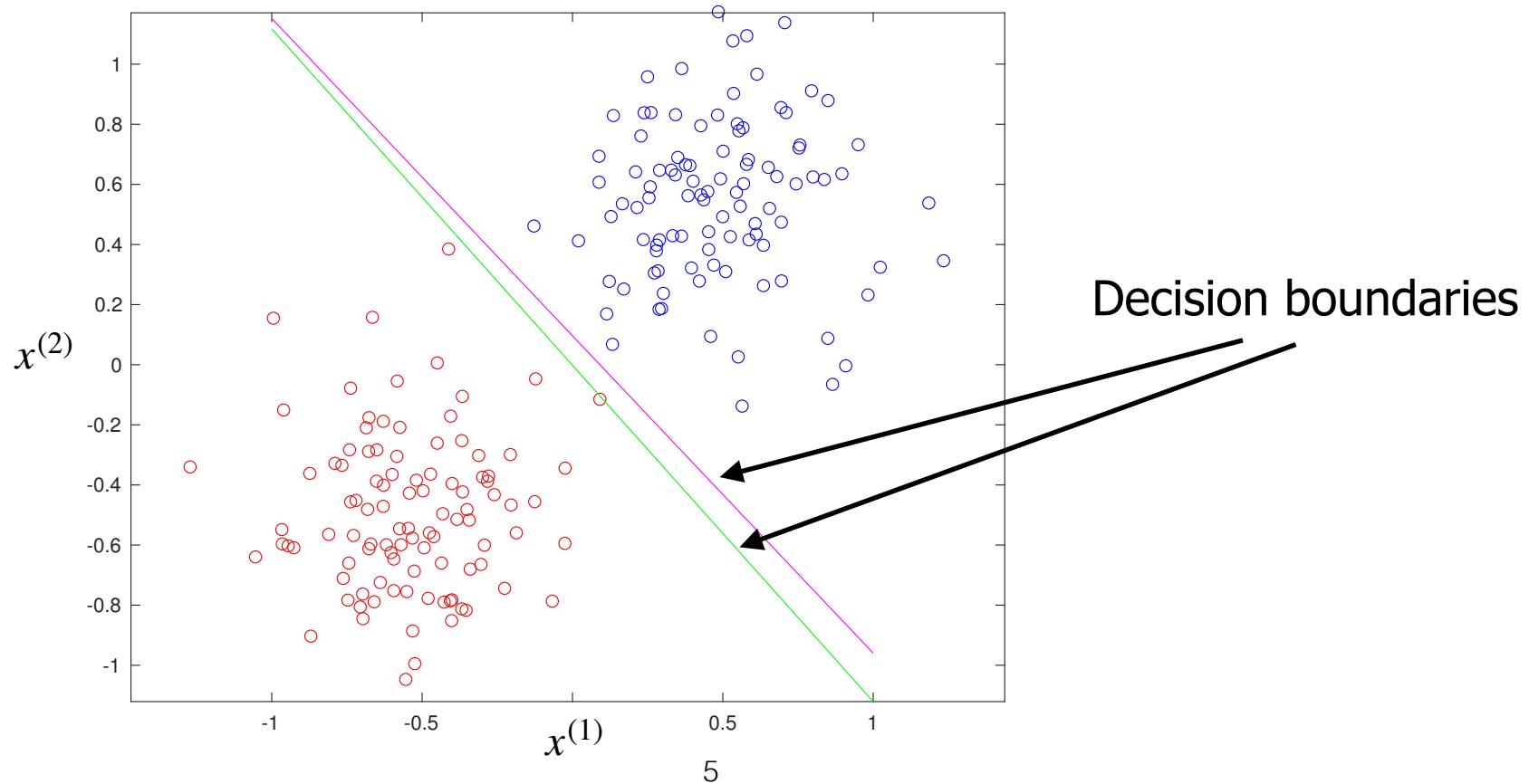


- Square: Least-square classification
- Logistic: Logistic regression
- Hinge: SVM
- 0-1: Ideal

- Choosing an appropriate loss function has an impact on the resulting algorithm

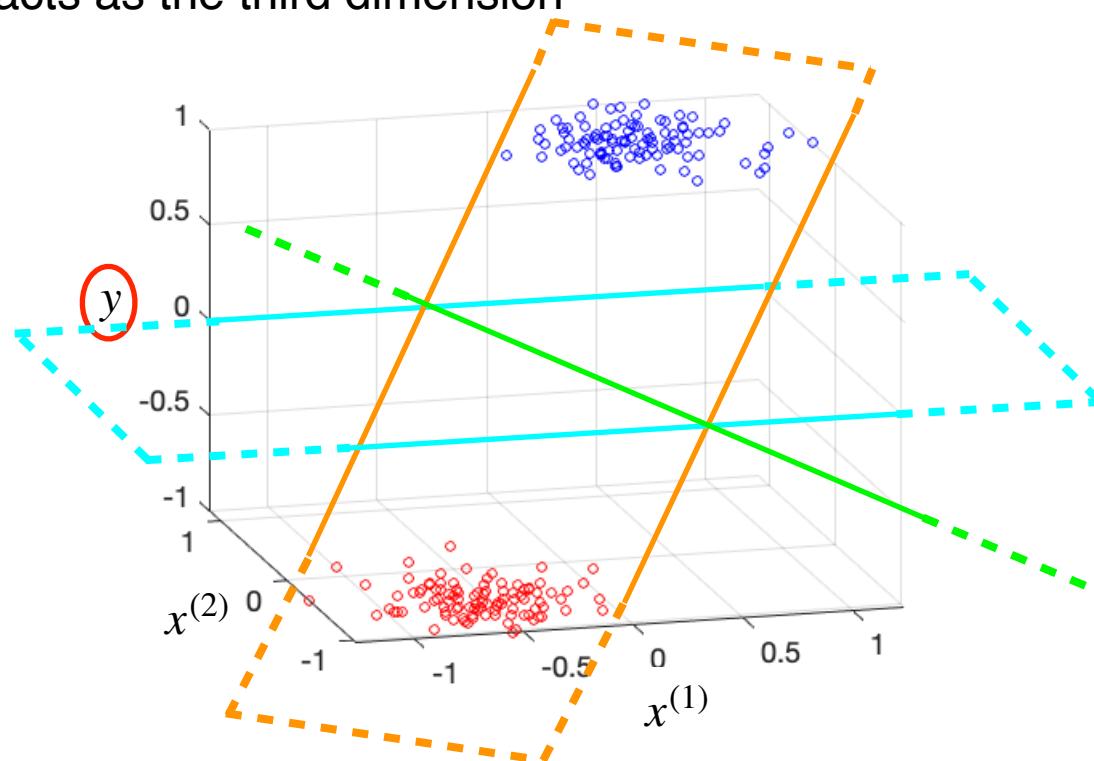
Linear model vs decision boundary

- Example: 2D inputs, 2 classes
 - Least-square classification in green, logistic regression in magenta



Linear model vs decision boundary

- The same data as before can also be seen in 3D
 - The label acts as the third dimension



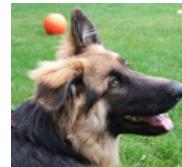
- The linear model encodes the plane that passes through the samples (orange plane)
- The decision boundary (green line) is the intersection of this plane with the plane $y = 0$ (or $y = 0.5$ if the negative label is 0) (cyan plane)

Recap: From binary to multiclass classification

- Initially, the three methods we have studied could only handle two classes: positive vs negative
- In general, one would typically like to discriminate between more than two categories. E.g., for image recognition



Label 1



Label 2



Label 3

- We then extended our three linear methods can handle this scenario

Recap: Dealing with multiple classes

- The most common approach to modeling a multi-class problem consists of encoding the class label as a vector with a single 1 at the index corresponding to the category and 0s elsewhere
 - In a 5-class problem, a sample in class 2 is represented as

$$\mathbf{y} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

- This is referred to as *one-hot encoding* (or 1-of-C encoding)

Recap: Multi-output linear model

- Predicting such a one-hot encoding is then similar to making the model output multiple values, as we saw for multi-output regression
- We therefore again use a matrix $\mathbf{W} \in \mathbb{R}^{(D+1) \times C}$, such that

$$\hat{\mathbf{y}}_i = \mathbf{W}^T \mathbf{x}_i = \begin{bmatrix} \mathbf{w}_{(1)}^T \\ \mathbf{w}_{(2)}^T \\ \vdots \\ \mathbf{w}_{(C)}^T \end{bmatrix} \mathbf{x}_i$$

where each $\mathbf{w}_{(j)}$ is a $(D + 1)$ -dimensional vector, used to predict one output dimension, i.e., the score for one class

Recap: Multi-class least-square classification

- Multi-class least-square classification is then similar to a multi-output regression problem
- We can then write training as

$$\min_{\mathbf{W}} \sum_{i=1}^N \|\mathbf{W}^T \mathbf{x}_i - \mathbf{y}_i\|^2$$

where each \mathbf{y}_i is now a C -dimensional vector with a single one at the index corresponding to the class label, and zeros everywhere else

Recap: Multi-class logistic regression

- As in the least-square classification case, for logistic regression, we can use one-hot encodings to represent multi-class labels
- Then, we again need to use a matrix $\mathbf{W} \in \mathbb{R}^{(D+1) \times C}$ to represent the model parameters
- In this case, the probability for a class k is given by the *softmax* function

$$\hat{y}^{(k)}(\mathbf{x}) = \frac{\exp(\mathbf{w}_{(k)}^T \mathbf{x})}{\sum_{j=1}^C \exp(\mathbf{w}_{(j)}^T \mathbf{x})}$$

Recap: Multi-class logistic regression

- The empirical risk can then be derived from the multi-class version of the cross entropy, which yields

$$R(\mathbf{W}) = - \sum_{i=1}^N \sum_{k=1}^C y_i^{(k)} \ln \hat{y}_i^{(k)}$$

- As in the binary case, a single $y_i^{(k)}$ is 1 for every sample i , so we really have one term per training sample
- As in the binary case, training is done by minimizing this loss using gradient descent

Recap: Dealing with multiple classes

- So far, we have seen how to extend binary least-square classification and binary logistic regression to the multi-class scenario. How about SVM?
- Unfortunately, there is no ideal solution to handle multiple classes in SVM
- Instead, one typically relies on using several binary classifiers
 - This can be done in a one-vs-rest or one-vs-one manner
 - Note that these two solutions are quite general, not specific to SVM

Exercises: Solutions

- You are training a multi-class least-square classifier to recognize cat, dog and car images. For the two images, \mathbf{x}_1 , \mathbf{x}_2 , below, what do the ground-truth vectors, \mathbf{y}_1 , \mathbf{y}_2 , look like numerically?

$$\cdot \mathbf{y}_1 = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}, \quad \mathbf{y}_2 = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

- Assuming the images are color images of size 32×32 , what is the size of the parameter matrix \mathbf{W} ?
 - \mathbf{W} is of size $(D + 1) \times C$. $D = 32 \cdot 32 \cdot 3 = 3072$ (3 for the number of color channels). $C = 3$ (number of categories).



\mathbf{x}_1



\mathbf{x}_2

Exercise: Solution

- You have trained a multi-class logistic regression classifier to recognize cat, dog and car images. For the two test images, \mathbf{x}_1 , \mathbf{x}_2 , below, assuming that the model predicts the correct class, what would you expect the outputs of the model, $\hat{\mathbf{y}}_1$, $\hat{\mathbf{y}}_2$, to look like numerically? Explain why.



\mathbf{x}_1



\mathbf{x}_2

- Solution: Let us assumed that the classes are ordered as cat, dog and car. Then, I would expect:

$$\cdot \hat{\mathbf{y}}_1 \approx \begin{bmatrix} 0.65 \\ 0.3 \\ 0.05 \end{bmatrix}, \text{ because cats and dogs are more similar than cats and cars}$$

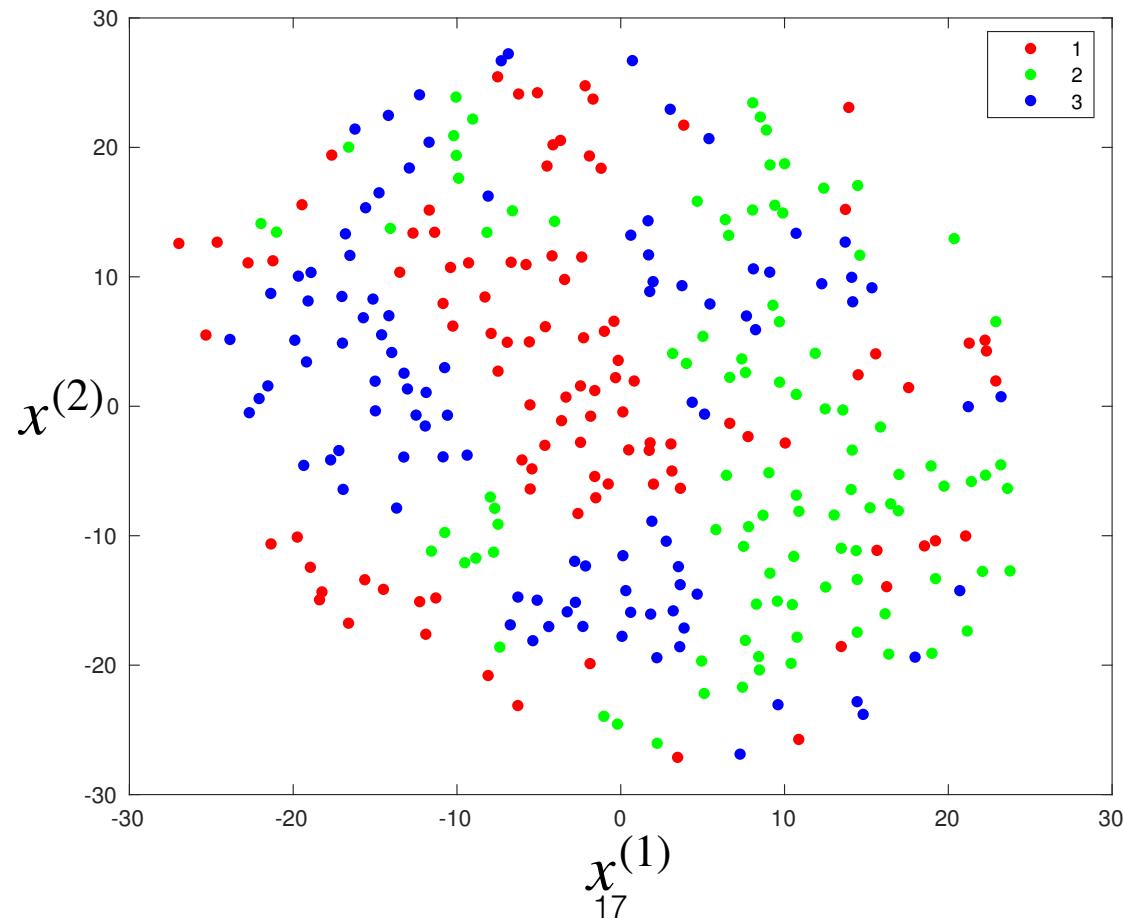
$$\cdot \hat{\mathbf{y}}_2 \approx \begin{bmatrix} 0.05 \\ 0.05 \\ 0.9 \end{bmatrix}, \text{ because a car looks very different from both cats and dogs}$$

Goals of today's lecture

- Move from linear to nonlinear machine learning
- Introduce the k-Nearest Neighbor method
- Introduce the idea of feature expansion

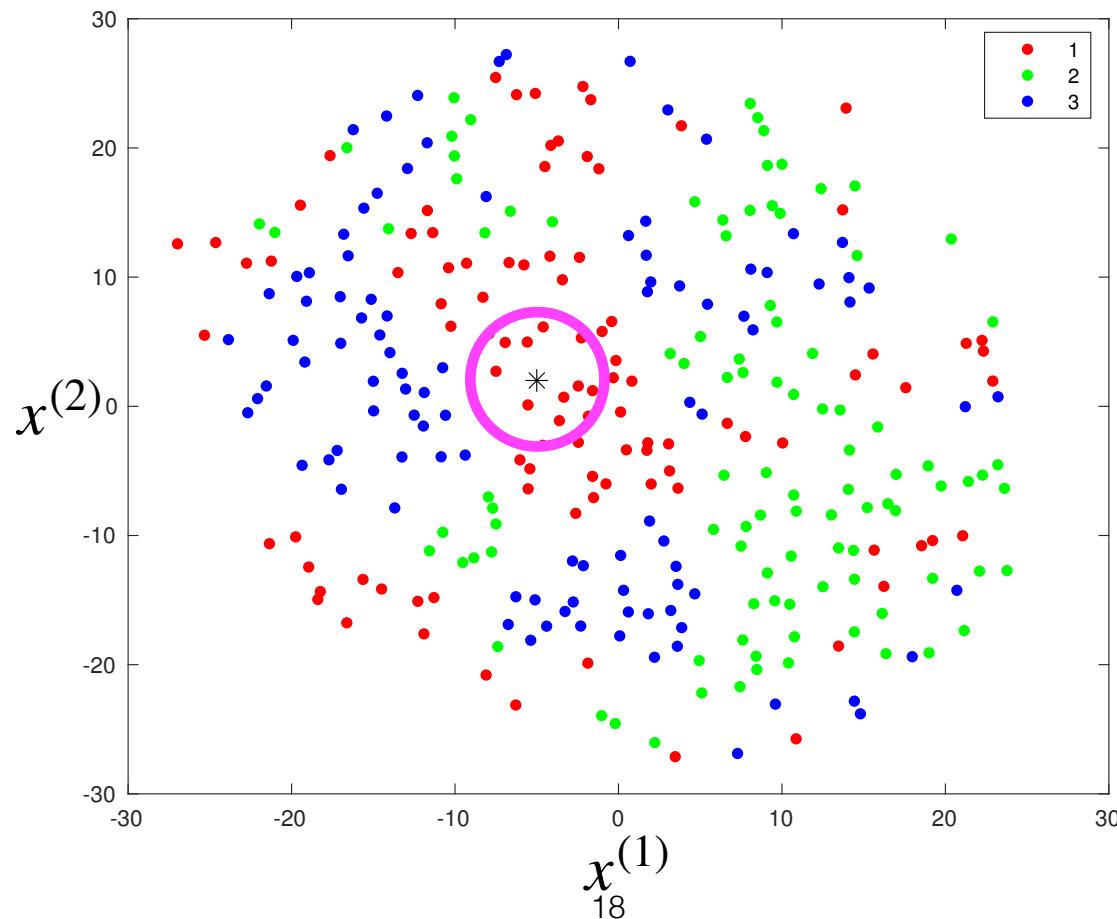
From linear to nonlinear

- Unfortunately, no linear model can directly fit this data



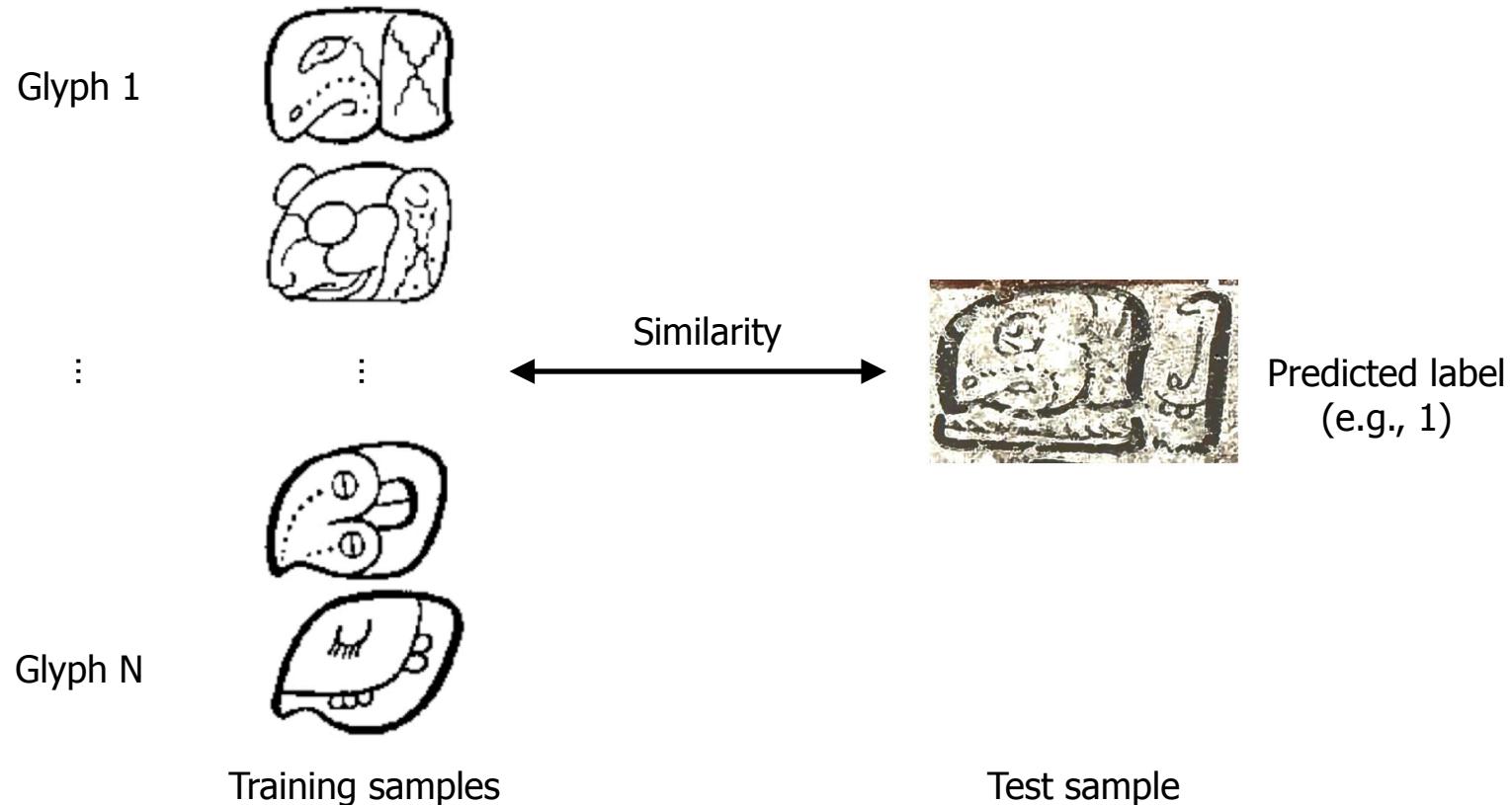
The simplest nonlinear ML algorithm

- However, assigning a class to this test sample (black star) seems very intuitive



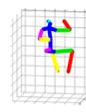
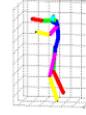
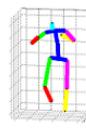
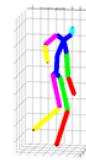
Nearest neighbor method (Bishop 2.5.2)

- Classification: Similar data samples have the same label
 - Example: Maya glyph recognition (Hu et al., 2017)



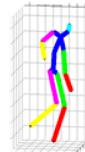
Nearest neighbor method

- Regression: Similar data samples have the same associated value
 - Example: Human pose estimation



Training samples

Similarity



Predicted pose

Test sample

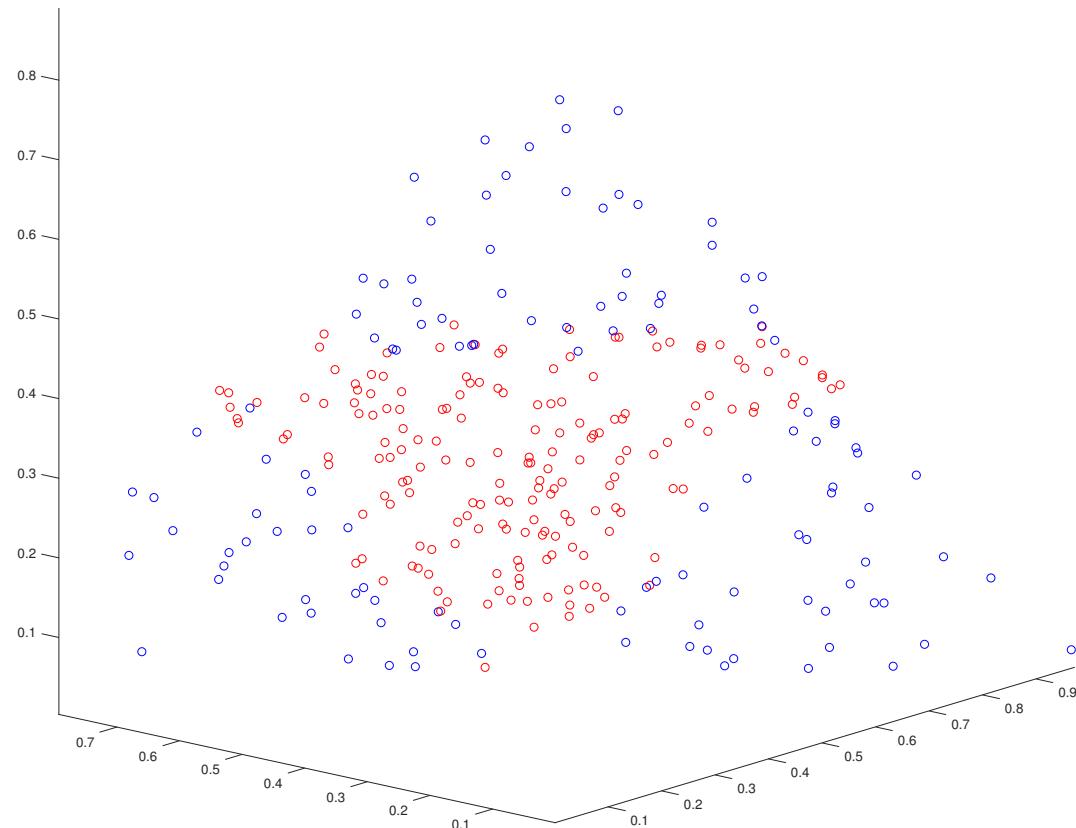
NN: Algorithm

1. Compute the distance between the test sample \mathbf{x} and all training samples $\{\mathbf{x}_i\}$
2. Find the sample \mathbf{x}_{NN} with minimum distance
3. Assign the corresponding label/value \mathbf{y}_{NN} to the test sample



NN: Properties

- The results may change for different distance functions
 - Toy example with 3D histograms belonging to 2 classes (colors)



Interlude

Histograms

Histograms

- A histogram in \mathbb{R}^D is a D -dimensional vector such that

$$x^{(d)} \in [0,1] , \forall 1 \leq d \leq D$$

$$\sum_{d=1}^D x^{(d)} = 1$$

- Given a vector in \mathbb{R}^D whose values are all non-negative (and at least one is strictly positive), one can obtain a histogram by dividing each value by the sum of values

Histograms: Numerical example

- Let the original vector in \mathbb{R}^9 be

$$\tilde{\mathbf{x}} = [1 \ 2 \ 1 \ 2 \ 1 \ 1 \ 0 \ 0 \ 0]^T$$

- The sum of its values is

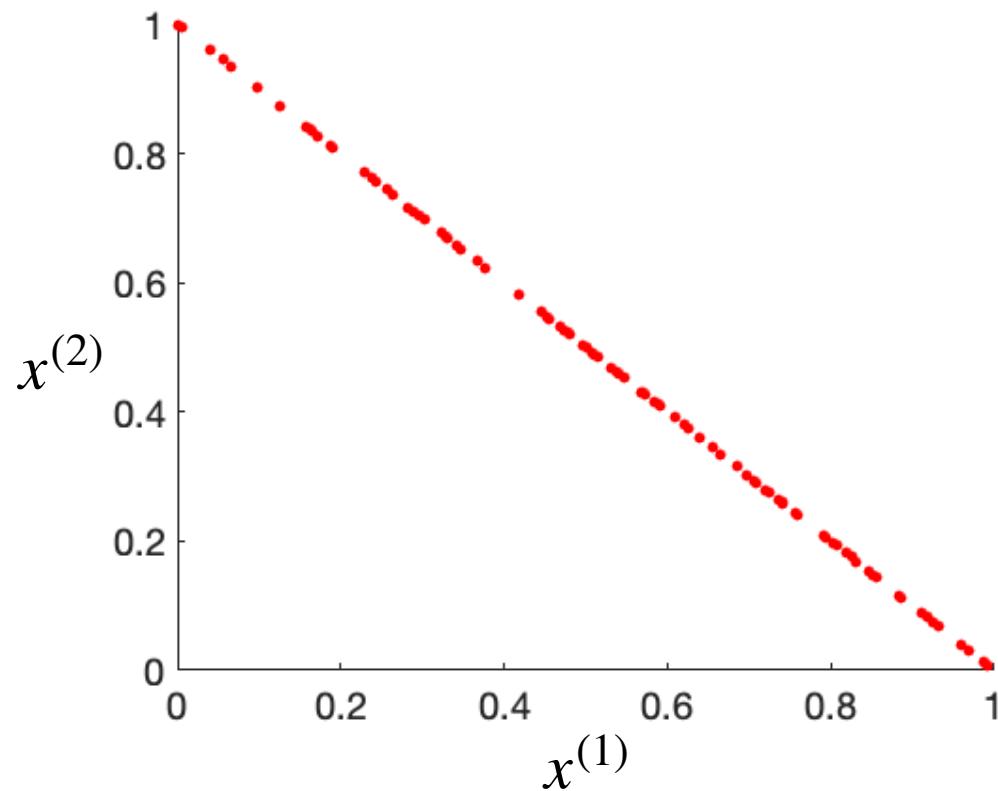
$$\sum_{d=1}^D \tilde{x}^{(d)} = 8$$

- Then, a histogram can be computed as

$$\mathbf{x} = [1/8 \ 1/4 \ 1/8 \ 1/4 \ 1/8 \ 1/8 \ 0 \ 0 \ 0]^T$$

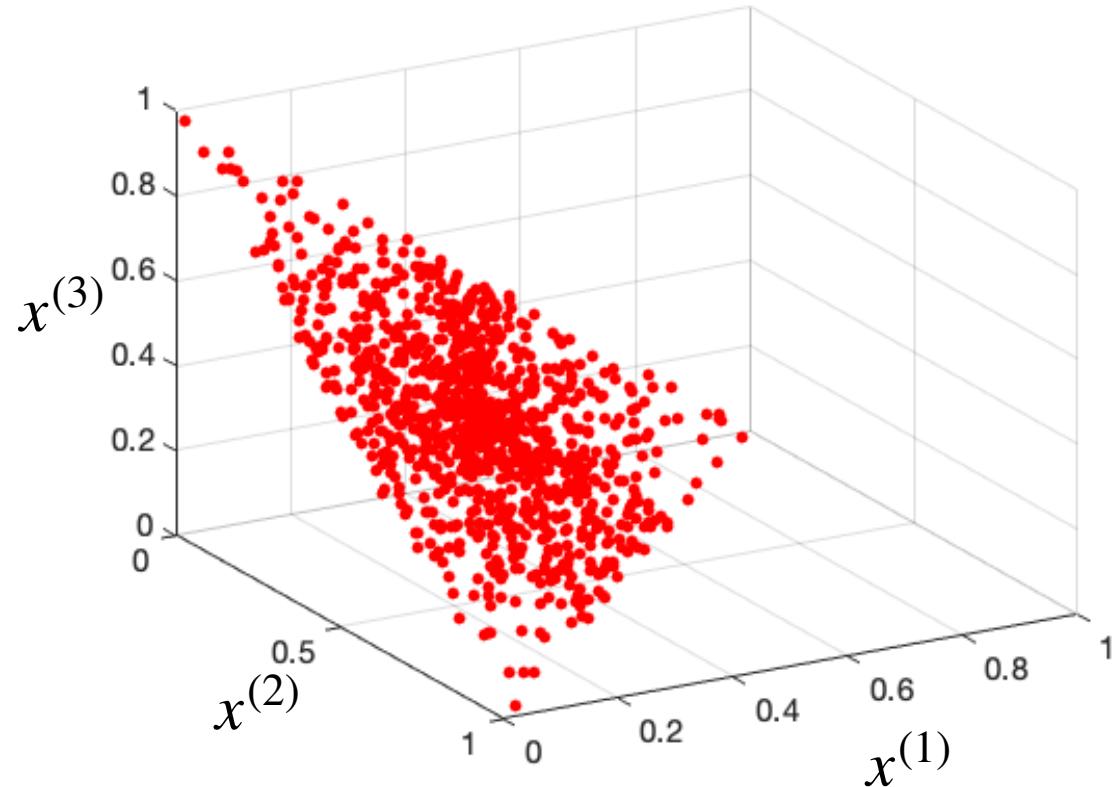
Histograms in 2D

- $N = 100$ samples in dimension $D = 2$



Histogram in 3D

- $N = 1000$ samples in dimension $D = 3$

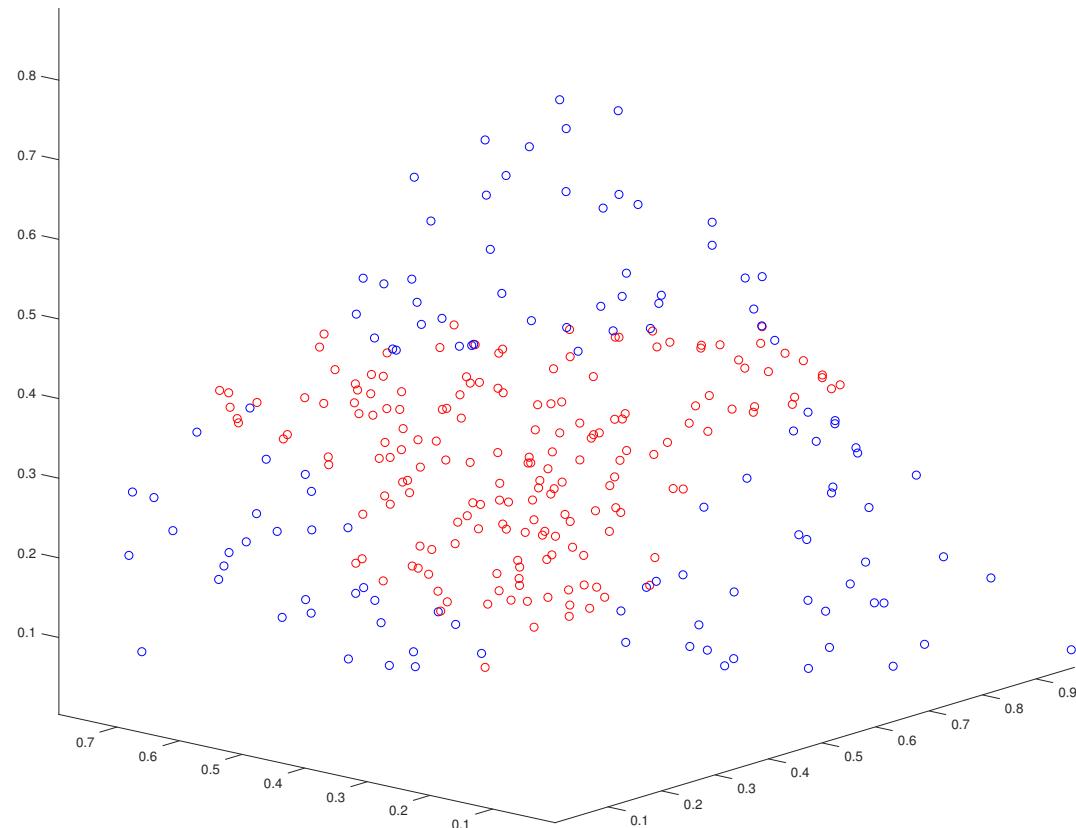


End of the interlude

Back to the NN algorithm

NN: Properties

- The results may change for different distance functions
 - Toy example with 3D histograms belonging to 2 classes (colors)



NN Example: 3D Histograms

- Default distance: Euclidean distance

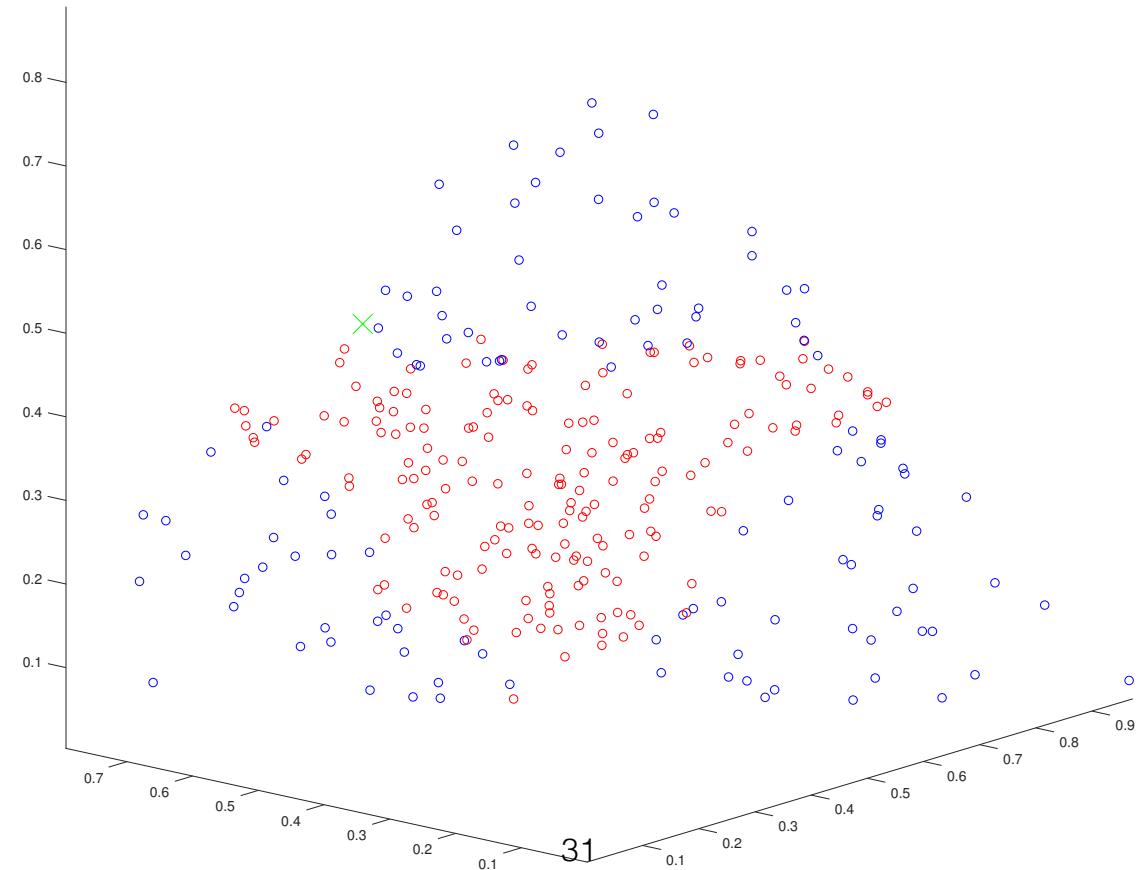
$$d(\mathbf{x}_i, \mathbf{x}) = \sqrt{\sum_{d=1}^D (x_i^{(d)} - x^{(d)})^2}$$

- Distance often used for histograms: Chi-square distance

$$d(\mathbf{x}_i, \mathbf{x}) = \chi^2(\mathbf{x}_i, \mathbf{x}) = \sqrt{\sum_{d=1}^D \frac{(x_i^{(d)} - x^{(d)})^2}{x_i^{(d)} + x^{(d)}}}$$

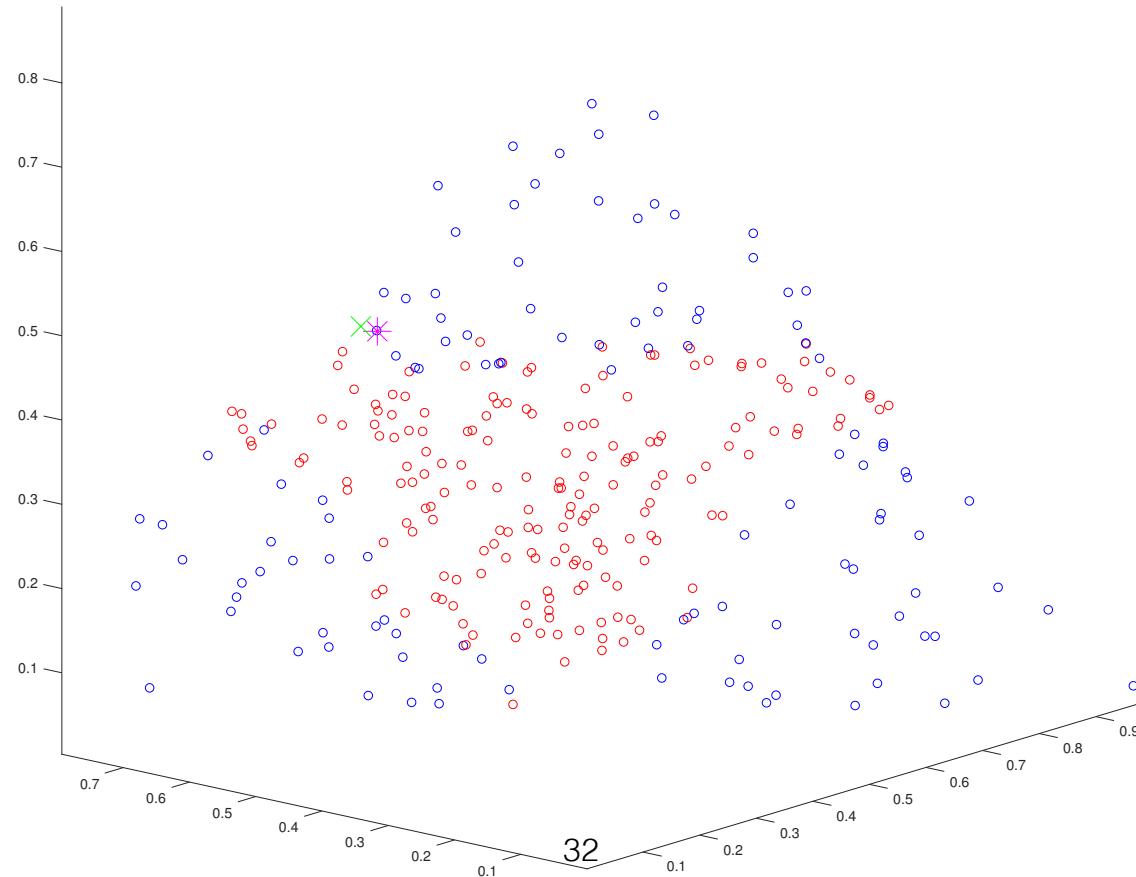
NN Example: 3D Histograms

- Test point: Green cross



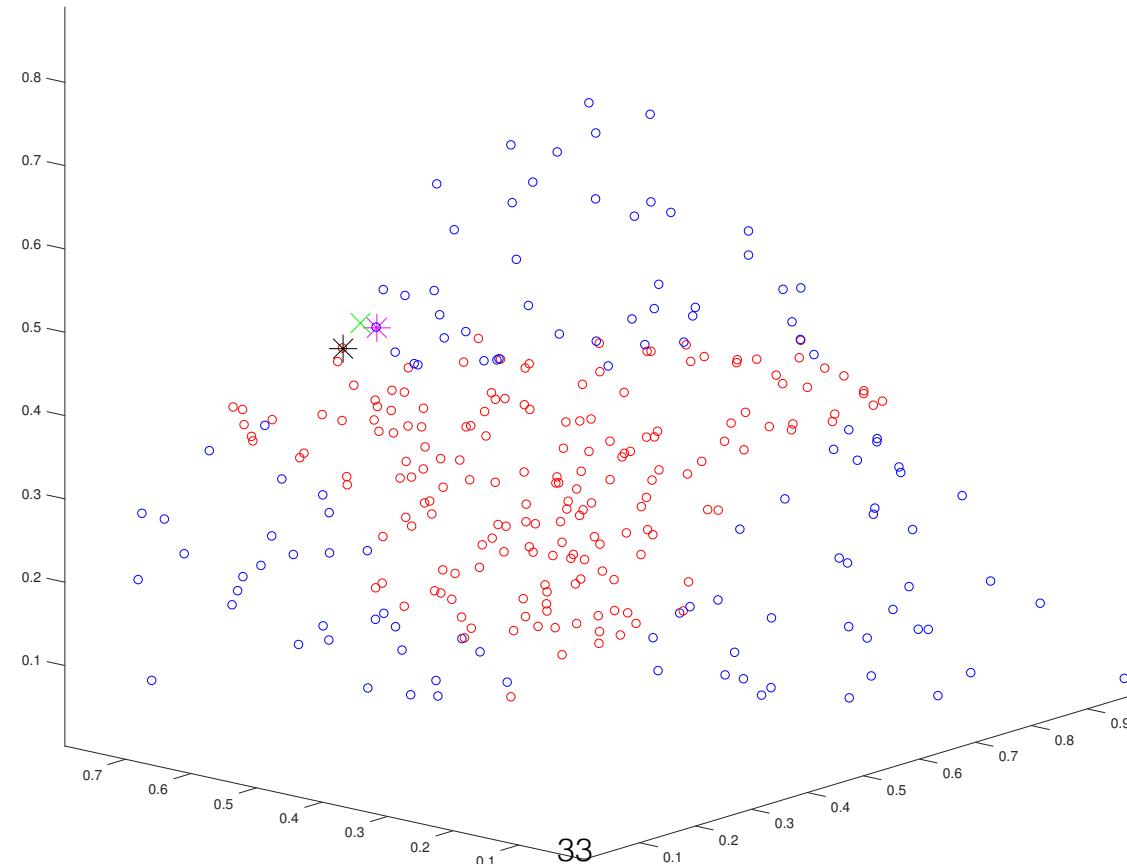
NN Example: 3D Histograms

- Test point: Green cross
- NN with Euclidean distance: Magenta star (class blue)



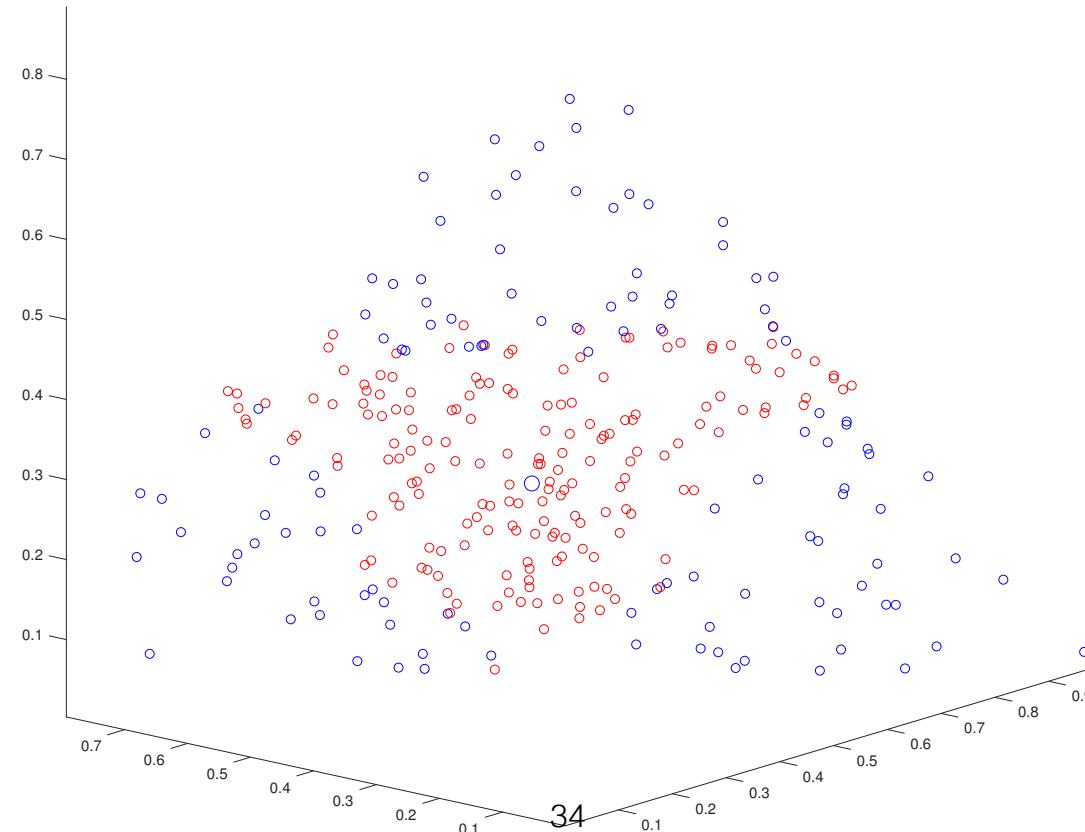
NN Example: 3D Histograms

- Test point: Green cross
- NN with Euclidean distance: Magenta star (class blue)
- NN with Chi-square distance: Black star (class red)



NN: Properties

- The results may be sensitive to outliers
 - A point close to the outlier (blue circle in the center) will be misclassified
 - Solution: Look at multiple neighbors instead of just one



k-Nearest Neighbors

KNN classif.

- Classification:

1. Compute the distance between the test sample \mathbf{x} and all training samples $\{\mathbf{x}_i\}$
 2. Find the k samples $\{\mathbf{x}_{NN1}, \dots, \mathbf{x}_{NNk}\}$ with minimum distances
 3. Find the most common label among these k nearest neighbors (majority vote)
 4. Assign the corresponding label \mathbf{y}_{MV} to the test sample
-
- If several labels appear the same number of times among the k -NN, one strategy consists of taking the one whose samples have the smallest average distance to the test sample

k-Nearest Neighbors

- Regression:

1. Compute the distance between the test sample \mathbf{x} and all training samples $\{\mathbf{x}_i\}$
2. Find the k samples $\{\mathbf{x}_{NN1}, \dots, \mathbf{x}_{NNk}\}$ with minimum distances
3. Compute the value $\hat{\mathbf{y}}$ for the test sample based on that of these k -NN

- Two strategies:

- Use the average of the k -NN values $\{\mathbf{y}_{NNi}\}$

$$\hat{\mathbf{y}} = \frac{1}{k} \sum_{i=1}^k \mathbf{y}_{NNi}$$

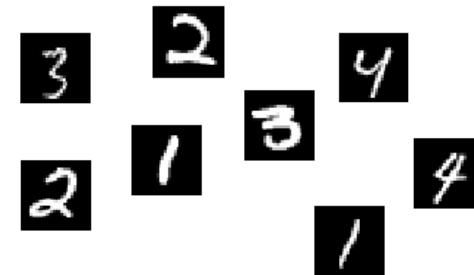
- Question: How else would you compute the predicted value $\hat{\mathbf{y}}$?

k-Nearest Neighbors: Demo

- <http://vision.stanford.edu/teaching/cs231n-demos/knn/>

k-Nearest Neighbors: Example

- MNIST classification



Method	Preprocessing	Error	Reference
K-nearest-neighbors, Euclidean (L2)	none	5.0	LeCun et al. 1998
K-nearest-neighbors, Euclidean (L2)	none	3.09	Kenneth Wilder, U. Chicago
K-nearest-neighbors, L3	none	2.83	Kenneth Wilder, U. Chicago
K-nearest-neighbors, Euclidean (L2)	deskewing	2.4	LeCun et al. 1998
K-nearest-neighbors, Euclidean (L2)	deskewing, noise removal, blurring	1.80	Kenneth Wilder, U. Chicago
K-nearest-neighbors, L3	deskewing, noise removal, blurring	1.73	Kenneth Wilder, U. Chicago
K-nearest-neighbors, L3	deskewing, noise removal, blurring, 1 pixel shift	1.33	Kenneth Wilder, U. Chicago
K-nearest-neighbors, L3	deskewing, noise removal, blurring, 2 pixel shift	1.22	Kenneth Wilder, U. Chicago
K-NN with non-linear deformation (IDM)	shiftable edges	0.54	Keysers et al. IEEE PAMI 2007
K-NN with non-linear deformation (P2DHMDM)	shiftable edges	0.52	Keysers et al. IEEE PAMI 2007
K-NN, Tangent Distance	subsampling to 16x16 pixels	1.1	LeCun et al. 1998
K-NN, shape context matching	shape context feature extraction	0.63	Belongie et al. IEEE PAMI 2002

k-Nearest Neighbors: Example

- Regression on UCI datasets:
 - Howley & Madden, AICS 2007 (k -NN with data-driven distance metric)
 - Abalone dataset:
 - Predict the age of abalone from 11 attributes, such as sex, length, diameter, height,...

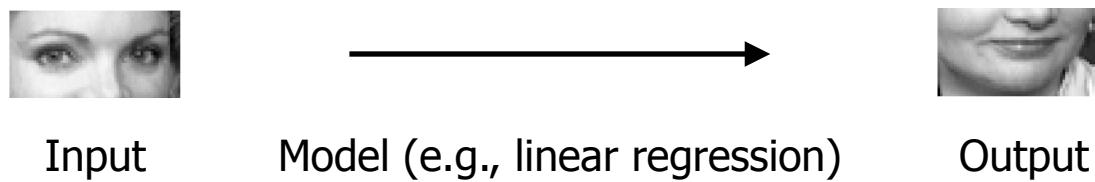


	No. Samples	No. Attributes	Average Test Error	
			Euclidean	KTree
Abalone*	4177	11	2.69	2.60

Recap: Multi-output linear regression: Example

- Face completion:

- Example from https://scikit-learn.org/stable/auto_examples/plot_multioutput_face_completion.html
- Task: Given the top half image of a face, predict the bottom half



- Dataset: Olivetti faces
 - 40 subjects (35 for training, 5 for testing)
 - 10 images per subject

Multi-output linear regression: Example

- Face completion:

- Results:

Linear regression



K-nn



true faces

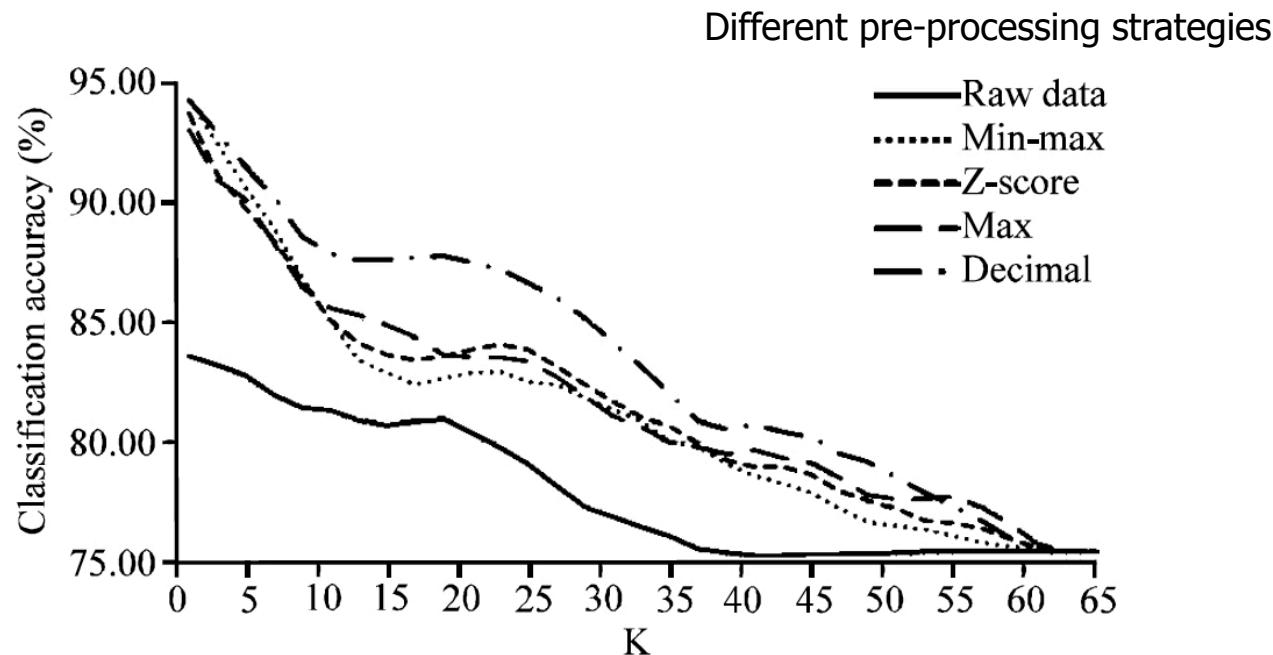


k-Nearest Neighbors: Properties

- No learning per se! What we need is
 - A good data representation
 - a distance function
 - a given value k
- This is referred to as a *non-parametric* learning method
 - No parameters to optimize during training
 - k is said to be a *hyper-parameter*, set manually (or by validation, which we will discuss in a future lecture)
- Nevertheless, k -NN can give very good results
 - E.g., MNIST digit recognition: 0.52% error

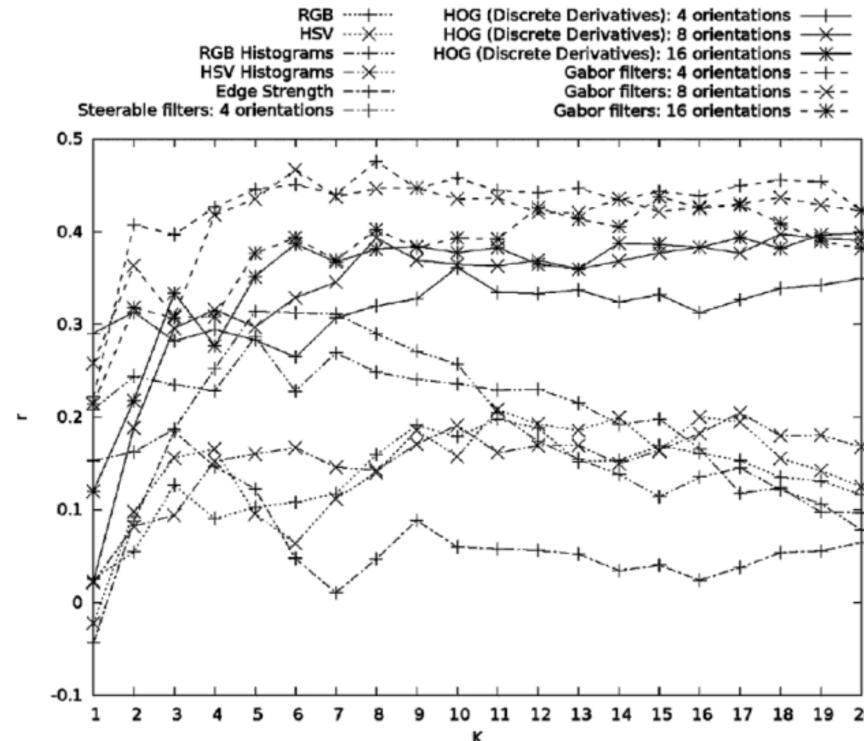
k -Nearest Neighbors: Properties

- The results depend on the value k
 - E.g., Ma et al., Journal of Applied Sciences, 2014
 - UCI Parkinson dataset: Predict if a patient suffers from Parkinson's disease from several biomedical voice measurements



k-Nearest Neighbors: Properties

- The results depend on the value k
 - E.g., Dee et al., “Visual digital humanities: using image data to derive approximate metadata”, 2016
 - Predict the year of a painting from an image



- Solution: Cross-validation → Next week

k-Nearest Neighbors: Properties

- Computationally expensive
 - For each test sample, one needs to compute the distances to all training samples
- Solution: Approximate nearest neighbor search
 - Hashing
 - kd-trees

Approximate k-Nearest Neighbors: Example

- Human pose estimation: Shakhnarovitch et al., ICCV 2003
 - Use k -NN with hashing to retrieve the most similar training image

Synthetic training data (for which you know the true pose)



Real test sample



Top match



k-Nearest Neighbors: Properties

- Curse of dimensionality:
 - In the real world, data representations tend to be high dimensional:
 - E.g., vectorizing an image yields a huge vector

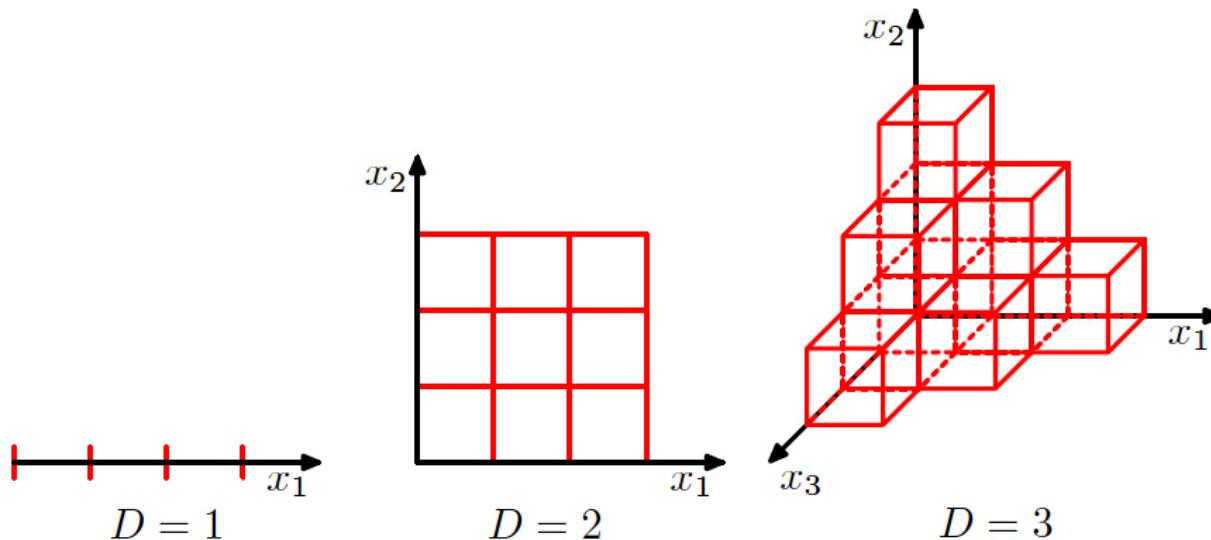


$$\mathbf{x}_i \in \mathbb{R}^{536 \cdot 356 \cdot 3} = \mathbb{R}^{572448}$$

- In such high dimension, all points tend to be far apart

Curse of dimensionality (Bishop 1.4)

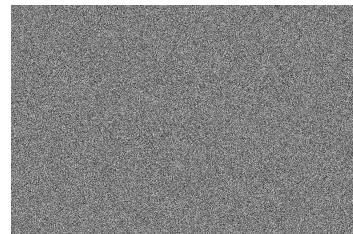
- Let us divide the input space into regular cells
- For dimensions $D = 1, 2, 3$ this gives



- The number of cells grows exponentially
- k-NN would then require an exponential growth of the data to cover the entire space

Curse of dimensionality

- Handling high-dimensional data also makes k -NN memory inefficient
 - We need to store all the training data to compute distances
- Fortunately, real data often lies in much smaller subspaces
 - E.g., the two images below lie in the same space, but we are more likely to observe the left one in the real world



- In a few weeks, we will talk about how to find the low-dimensional subspace of a given dataset

k-Nearest Neighbors: Summary

- Advantages:
 - Simple method
 - Effective to handle nonlinear data
 - Only requires defining one parameter (k)
- Drawbacks:
 - The results depend on k
 - Becomes expensive as N and/or D grow
 - May be unreliable with a large D
- Let us now look at a different way to handle nonlinear data

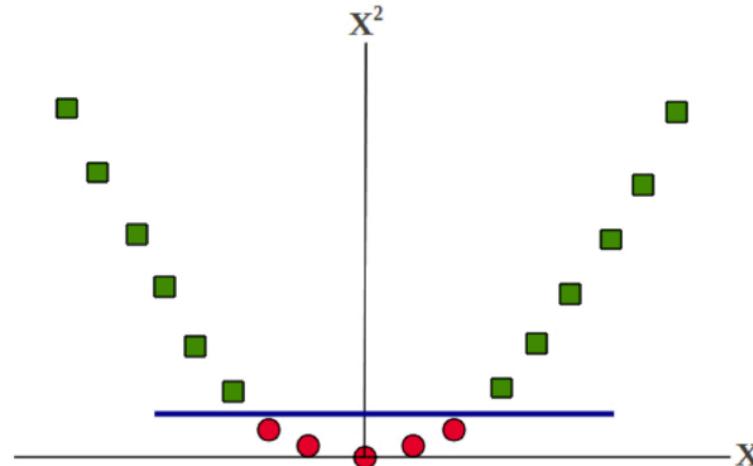
Nonlinear classification

- How can we handle this data?
 - 1D input, 2 classes (colors)



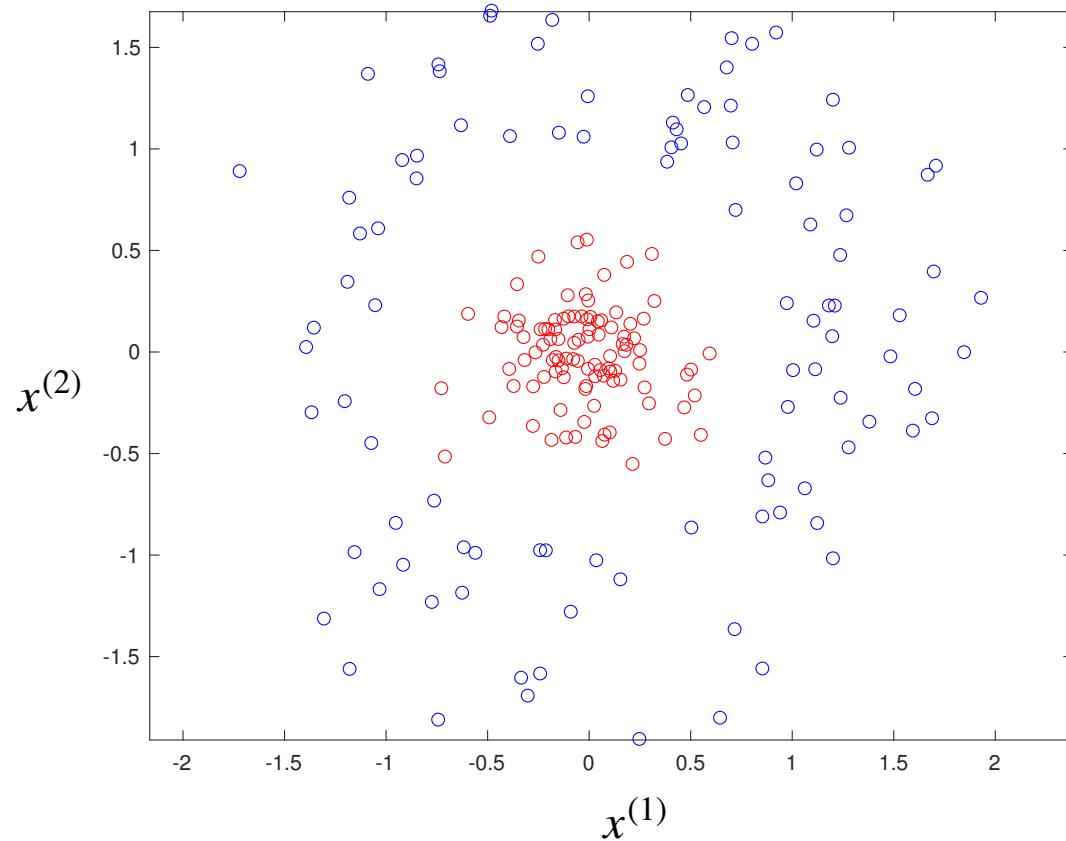
- How about transforming the input?

$$x \rightarrow [x, x^2]$$



Nonlinear classification

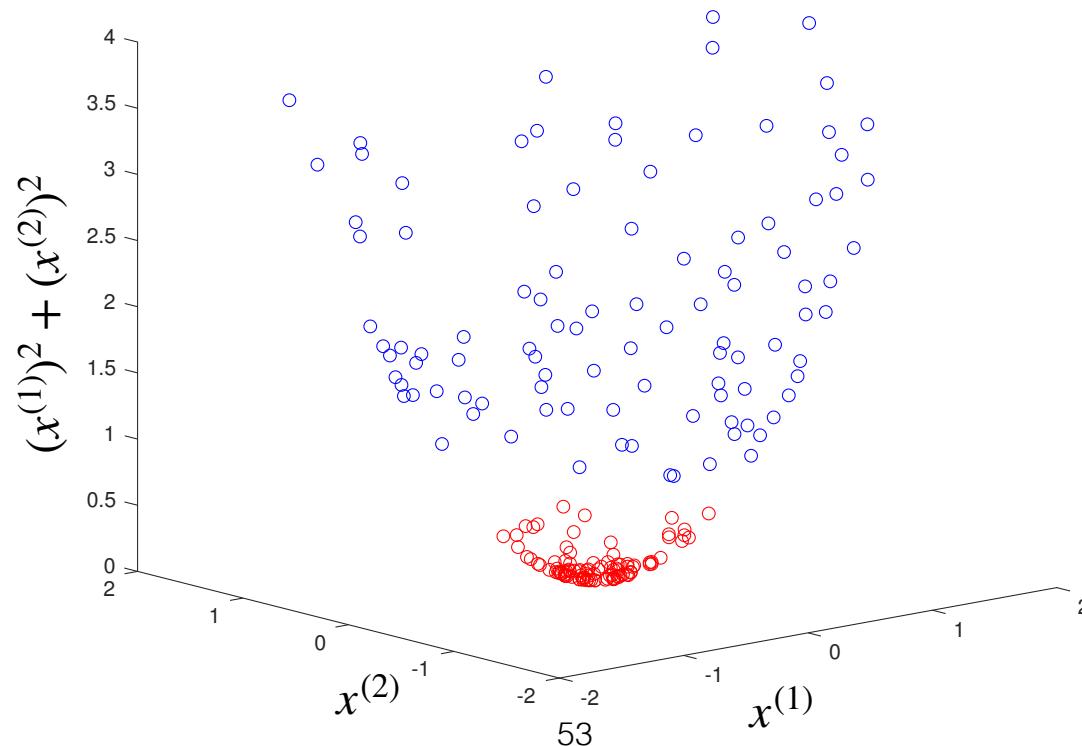
- How about this data?
 - 2D input, 2 classes (colors)



Nonlinear classification

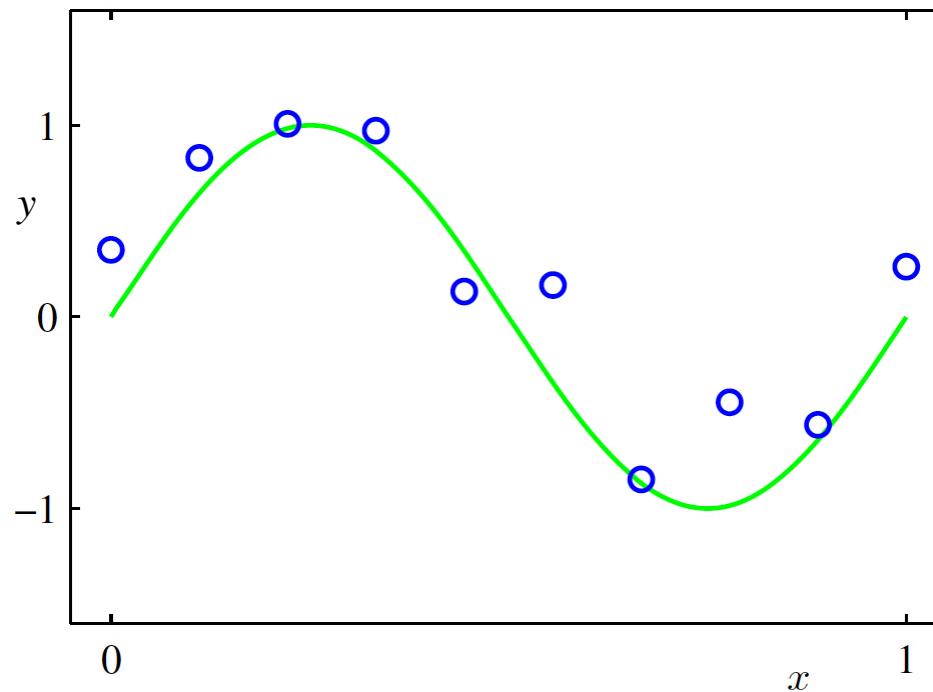
- Again, how about transforming the input?

$$\begin{bmatrix} x^{(1)} \\ x^{(2)} \end{bmatrix} \rightarrow \begin{bmatrix} x^{(1)} \\ x^{(2)} \\ (x^{(1)})^2 + (x^{(2)})^2 \end{bmatrix}$$



Polynomial curve fitting

- We have noisy training observations (blue circles, 1D input, 1D output) coming from the true green curve



Polynomial curve fitting

- We seek to find a polynomial function that approximates the true curve using these observations
- Such a polynomial function of degree M can be written as

$$y = w^{(0)} + w^{(1)}x + w^{(2)}x^2 + \dots + w^{(M)}x^M = \sum_{j=0}^M w^{(j)}x^j$$

where the $\{w^{(j)}\}$ are the coefficients of the different terms, but x^j represents x to the power j

- For example:
 - For $M = 0$, we have the constant function $y = w^{(0)}$
 - For $M = 1$, we have the line $y = w^{(0)} + w^{(1)}x$
 - For $M = 2$, we have the quadratic function $y = w^{(0)} + w^{(1)}x + w^{(2)}x^2$

Polynomial feature expansion

- In essence, in the three examples, we performed a change of variables
 - In the first case: $x \rightarrow \phi(x) = [x, x^2]$

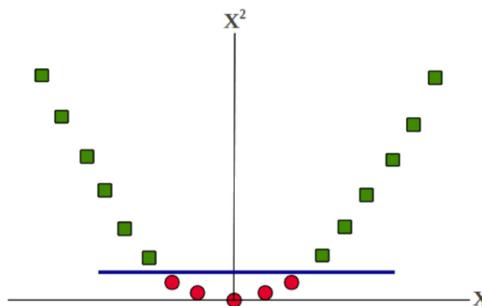
- In the second case: $\mathbf{x} \rightarrow \phi(\mathbf{x}) = \begin{bmatrix} x^{(1)} \\ x^{(2)} \\ (x^{(1)})^2 + (x^{(2)})^2 \end{bmatrix}$

- In the last (polynomial) case: $x \rightarrow \phi(x) = \begin{bmatrix} 1 \\ x \\ x^2 \\ \vdots \\ x^M \end{bmatrix}$

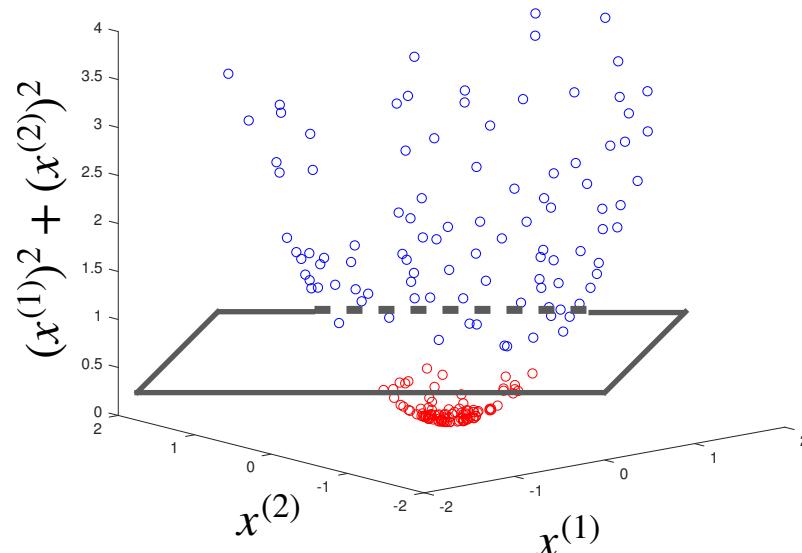
Polynomial feature expansion

- We can then apply a linear model to the resulting variables

- In the first case:



- In the second case:



Polynomial feature expansion

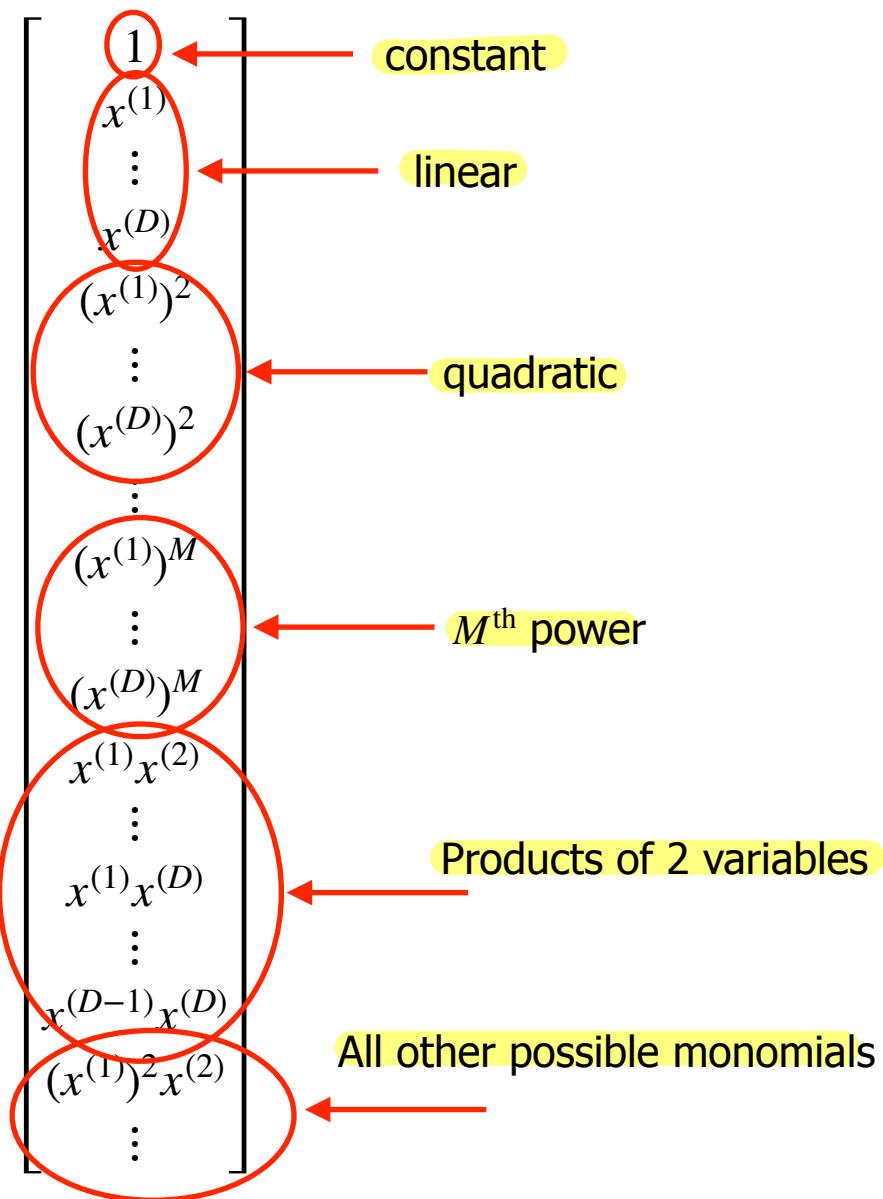
- We can then apply a linear model to the resulting variables
 - In the last case, we have

$$\sum_{j=0}^M w^{(j)} x^j = \mathbf{w}^T \begin{bmatrix} 1 \\ x \\ x^2 \\ \vdots \\ x^M \end{bmatrix}$$

Polynomial feature expansion

- In general, for D -dimensional inputs, we can obtain a large $\phi(\mathbf{x})$
- The resulting expanded features can then be used in any linear algorithm that we have seen previously
 - Let us look at how this would work for linear regression

$$\phi(\mathbf{x}) =$$



Working with expanded features

- In essence, polynomial feature expansion uses a mapping from $\mathbf{x} \in \mathbb{R}^D$ to another representation $\phi(\mathbf{x}) \in \mathbb{R}^F$, where $F \gg D$
- We can then use a linear model in this new space and write

$$\hat{y}_i = \mathbf{w}^T \phi(\mathbf{x}_i)$$

where now $\mathbf{w} \in \mathbb{R}^F$

(assuming that the 1 accounting for the bias has been incorporated in $\phi(\mathbf{x}_i)$)

Working with expanded features: Training

- For linear regression, we used the least-square loss

$$\min_{\mathbf{w}} \sum_{i=1}^N (\mathbf{w}^T \mathbf{x}_i - y_i)^2$$

- With our expanded features, we can re-write this as

$$\min_{\mathbf{w}} \sum_{i=1}^N (\mathbf{w}^T \phi(\mathbf{x}_i) - y_i)^2$$

Note that the \mathbf{w} vectors in the two equations above are different:

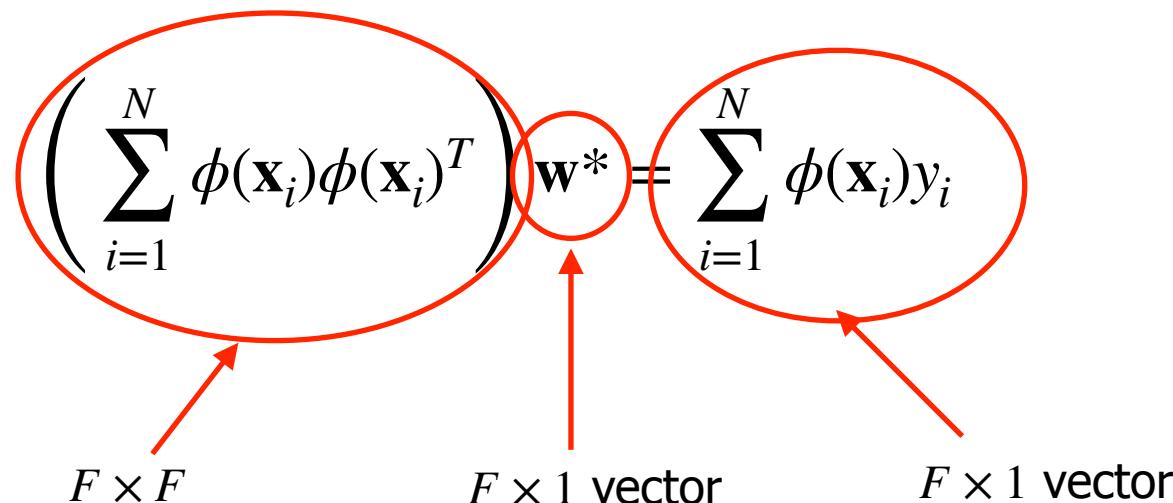
- In the first, $\mathbf{w} \in \mathbb{R}^{D+1}$
- In the second, $\mathbf{w} \in \mathbb{R}^F$

Working with expanded features: Gradient

- This new function is still convex, and its gradient is given by

$$\nabla E = 2 \sum_{i=1}^N \phi(\mathbf{x}_i) (\phi(\mathbf{x}_i)^T \mathbf{w} - y_i)$$

- Setting it to zero means that



Working with expanded features: Solution

- We can group the transformed inputs $\{\phi(\mathbf{x}_i)\}$ and the outputs $\{y_i\}$ in a matrix and vector of the form

$$\Phi = \begin{bmatrix} \phi(\mathbf{x}_1)^T \\ \phi(\mathbf{x}_2)^T \\ \vdots \\ \phi(\mathbf{x}_N)^T \end{bmatrix} \in \mathbb{R}^{N \times F} \quad \mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix} \in \mathbb{R}^N$$

- Then, we have

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

Polynomial feature expansion: Demo

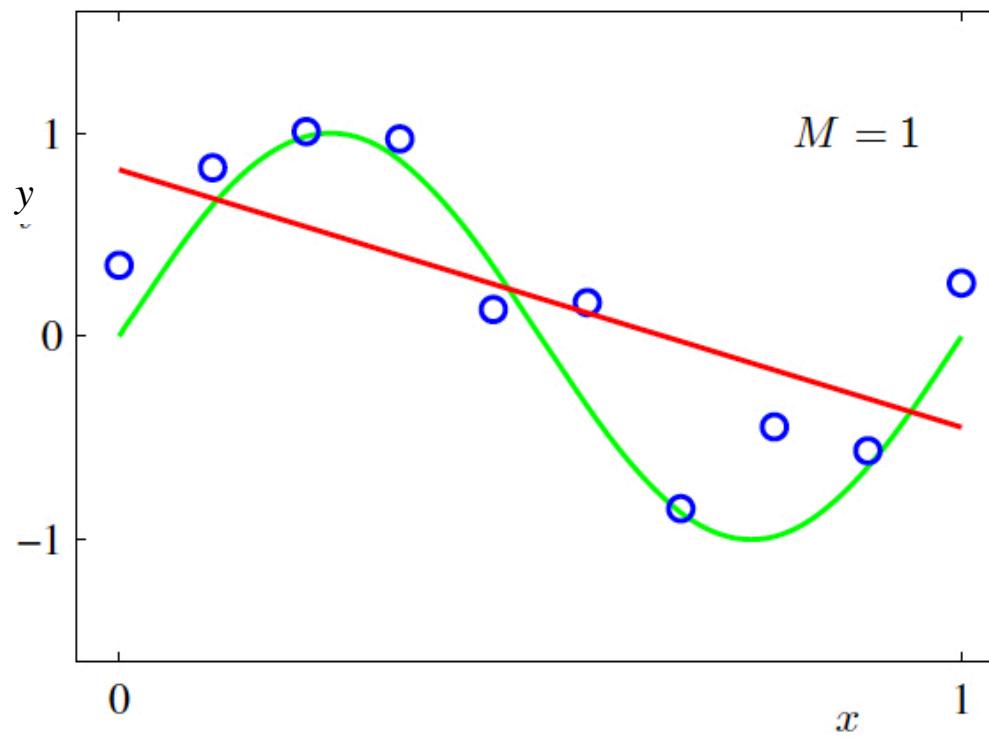
- <https://playground.tensorflow.org/#activation=linear&batchSize=10&dataset=gauss®Dataset=reg-plane&learningRate=0.03®ularizationRate=0&noise=0&networkShape=&seed=0.49340&showTestData=false&discretize=false&percTrainData=50&x=true&y=true&xTimesY=false&xSquared=false&ySquared=false&cosX=false&sinX=false&cosY=false&sinY=false&collectStats=false&problem=classification&initZero=false&hideText=false>

Mapping to a higher-dimensional space

- In all the examples considered before, the mapping $\phi(\cdot)$ increases the dimensionality of the input to the linear model
 - From 1 to 2 in the first toy example
 - From 2 to 3 in the second one
 - From 1 to $M + 1$ in the polynomial fitting case
 - From 2 to 5 in the previous demo
- Discussion: Why is higher dimension good?

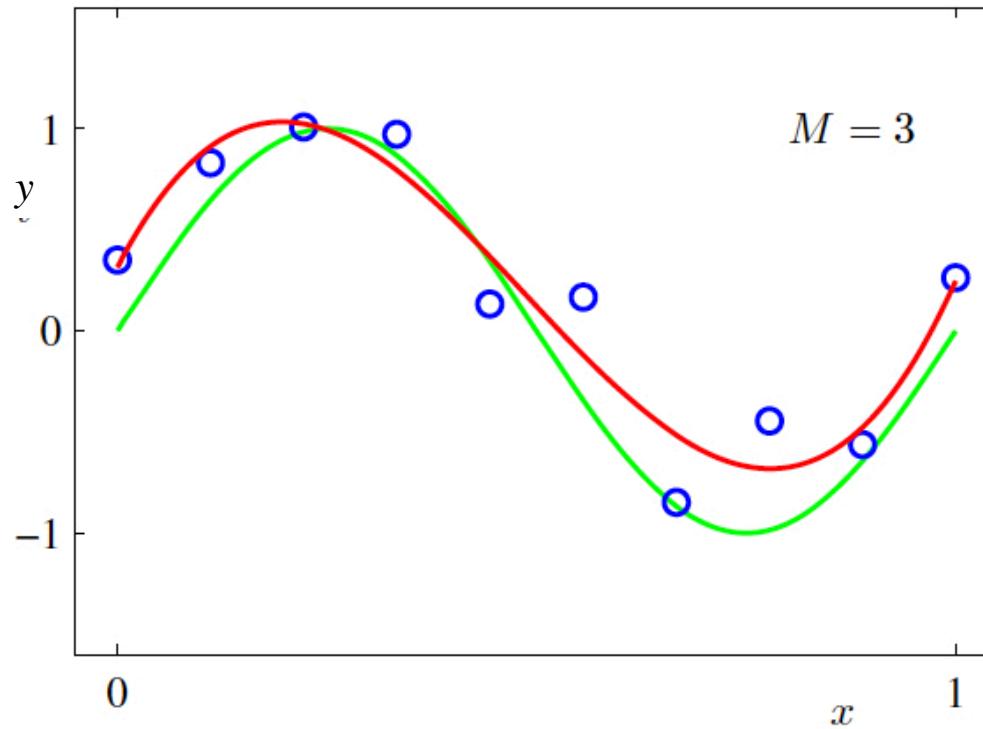
Feature expansion: Properties

- Effective, but requires choosing the degree of the polynomial
 - In Bishop's polynomial curve fitting example, with $M = 1$



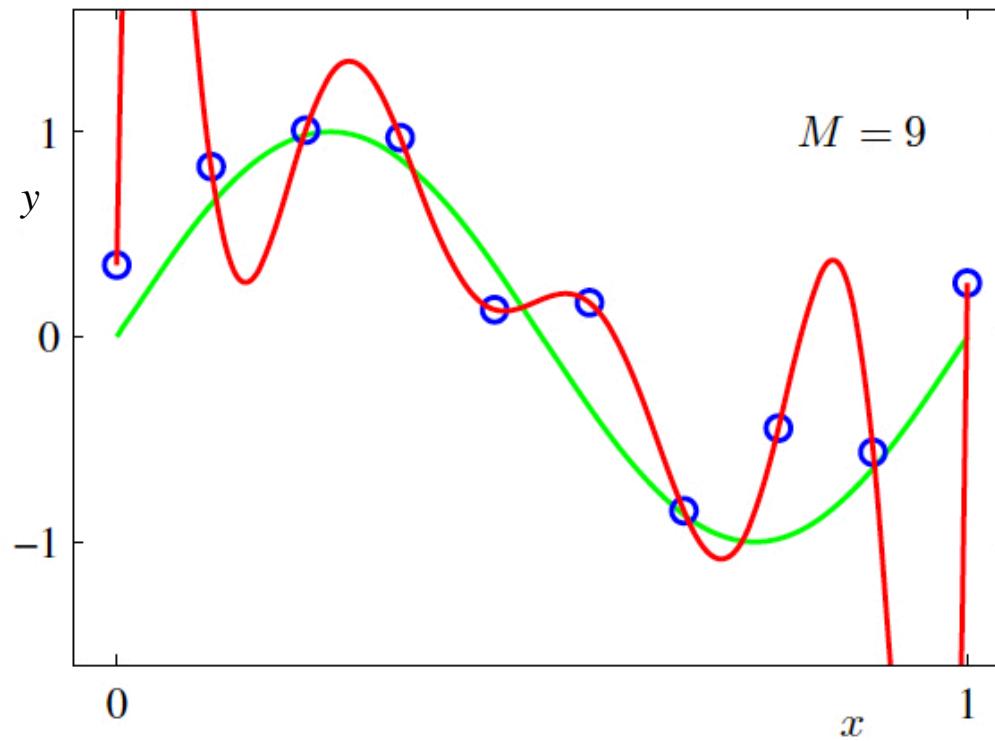
Feature expansion: Properties

- Effective, but requires choosing the degree of the polynomial
 - In Bishop's polynomial curve fitting example, with $M = 3$



Feature expansion: Properties

- Effective, but requires choosing the degree of the polynomial
 - In Bishop's polynomial curve fitting example, with $M = 9$ 



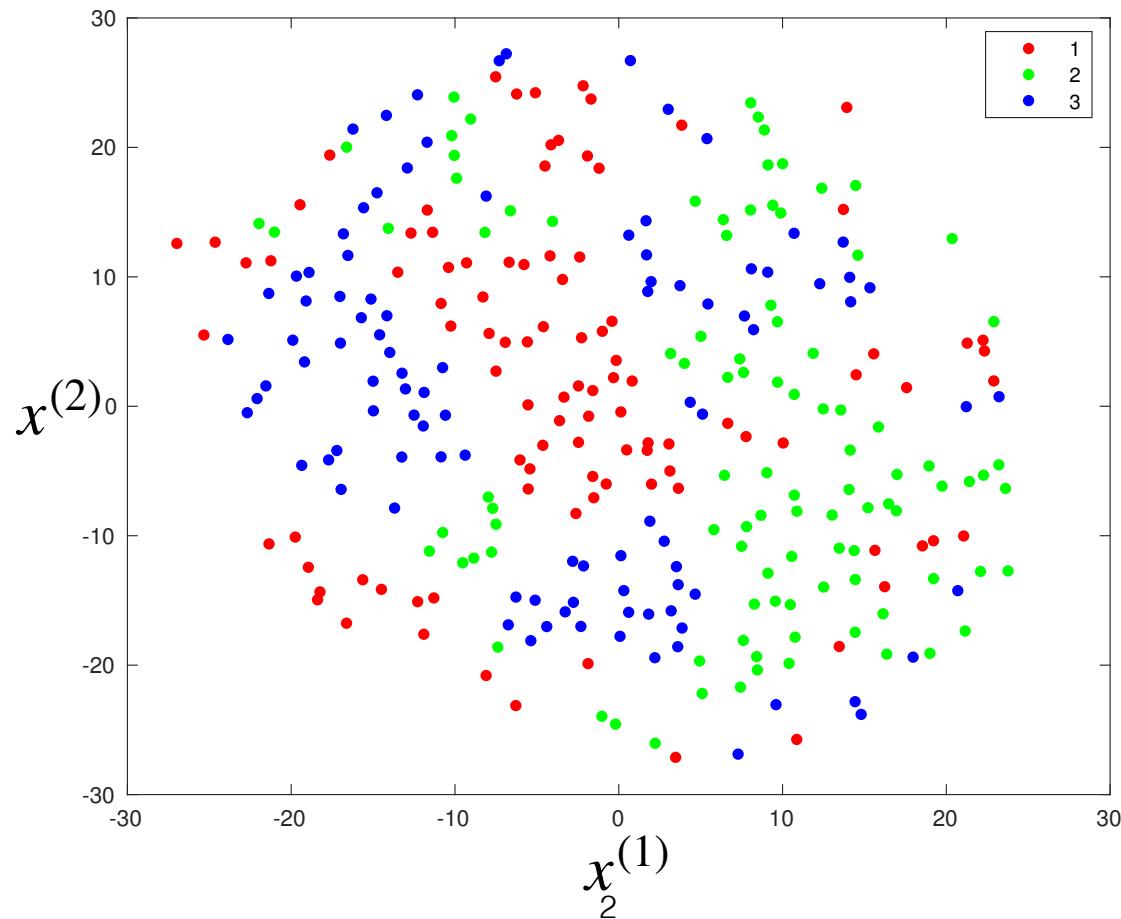
Feature expansion: Properties

- Why limit ourselves to polynomials?
 - Any function of the input could be used, e.g.,
 - sine and cosine
 - exponential and logarithm
 - ...
- In fact, in the demo...
 - <https://playground.tensorflow.org/#activation=linear&batchSize=10&dataset=gauss®Dataset=reg-plane&learningRate=0.03®ularizationRate=0&noise=0&networkShape=&seed=0.49340&showTestData=false&discretize=false&percTrainData=50&x=true&y=true&xTriesY=false&xSquared=false&ySquared=false&cosX=false&sinX=false&cosY=false&sinY=false&collectStats=false&problem=classification&initZero=false&hideText=false>
- Then, how do we choose which functions to use?
 - Solution: Kernels: In 2 weeks

Lecture 7: Overfitting, Cross-validation and Regularization

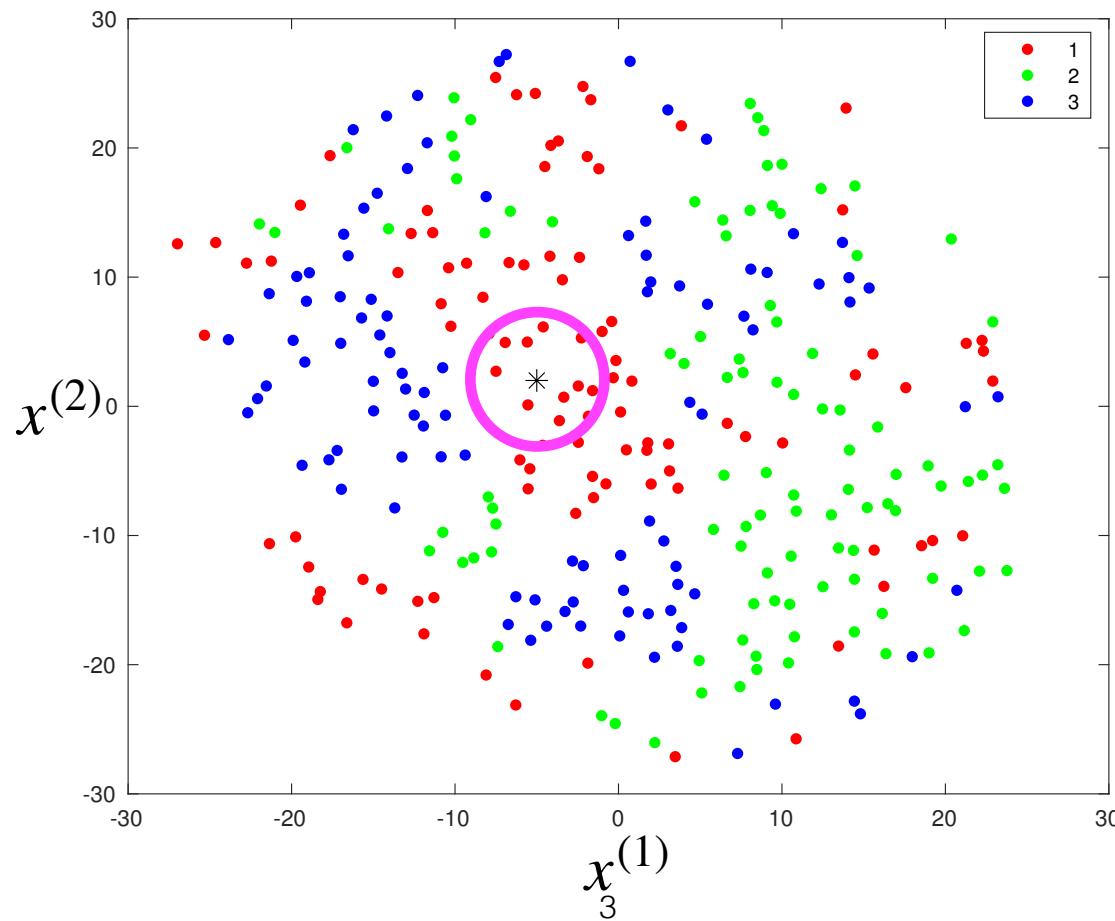
Recap: From linear to nonlinear

- Although we spent much time on linear models, no linear model can directly fit this data



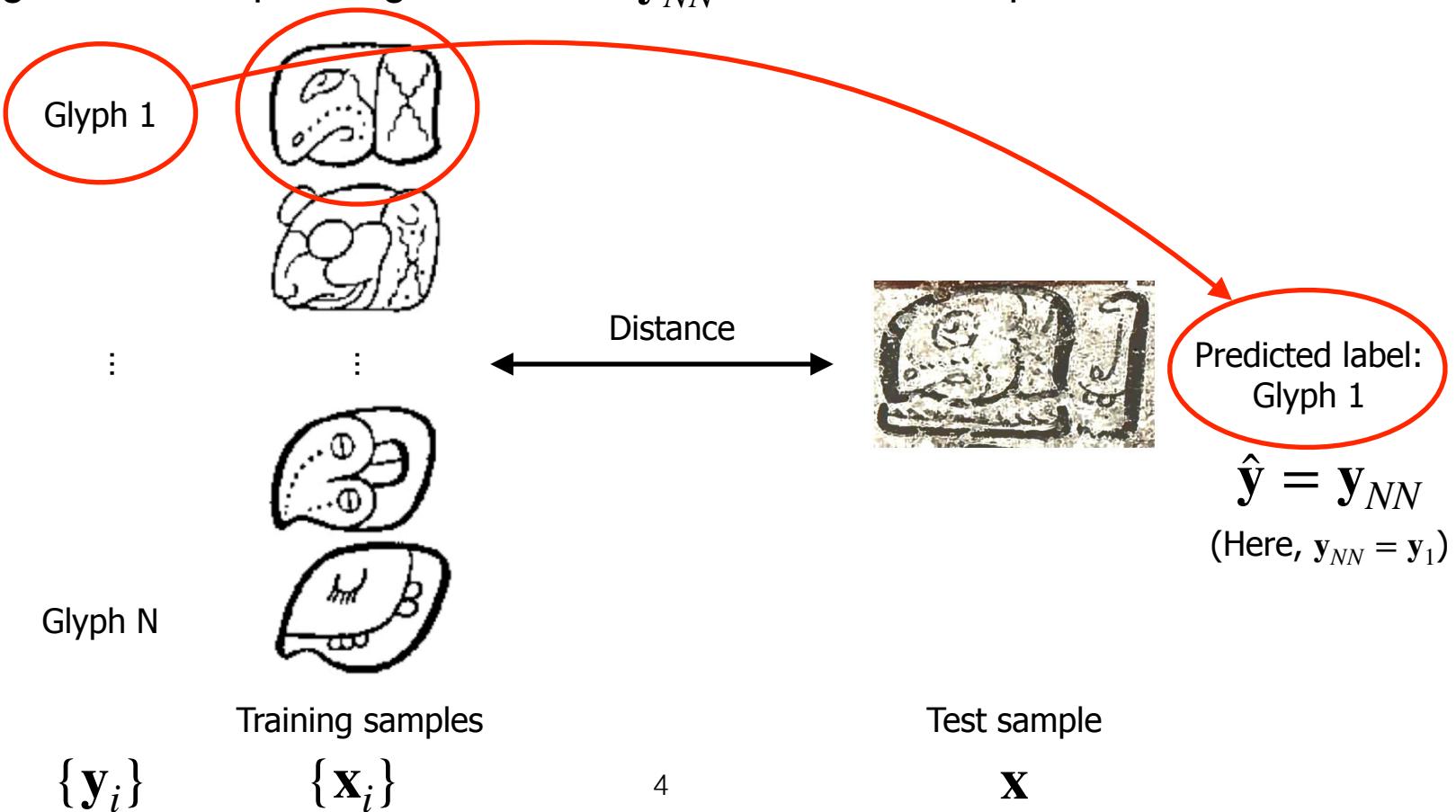
Recap: The simplest nonlinear ML algorithm

- However, assigning a class to this test sample (black star) seems very intuitive



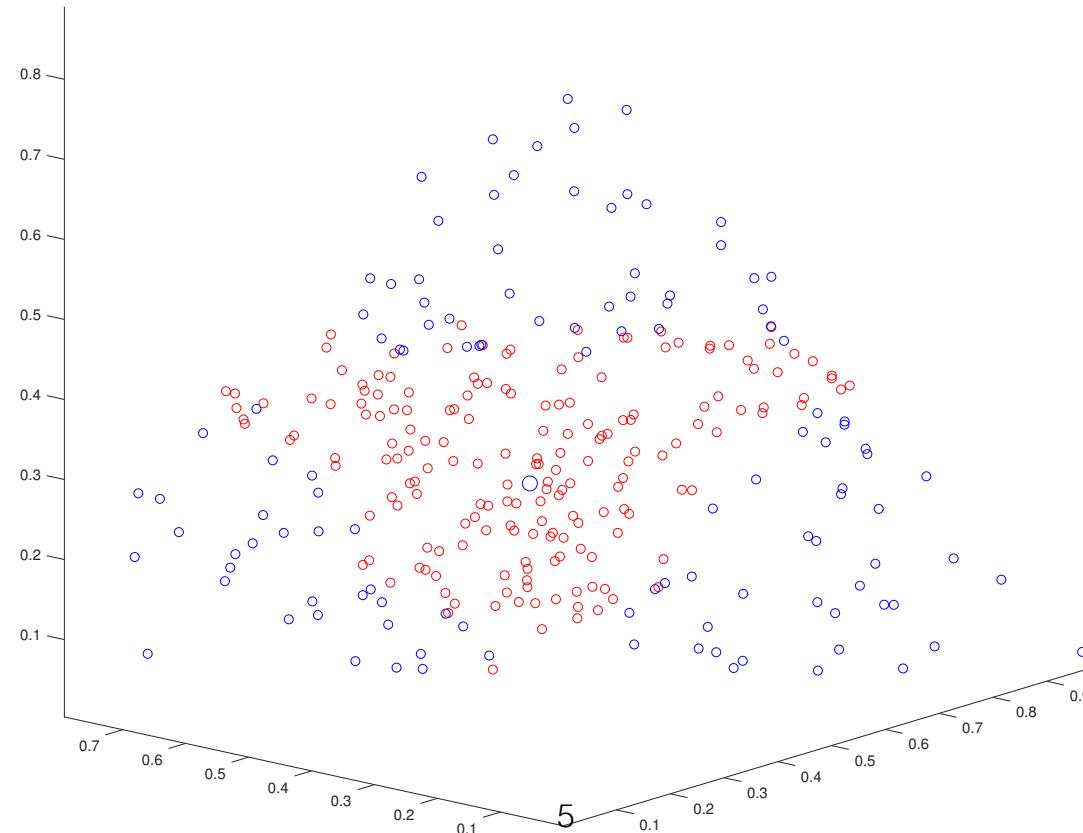
Recap: Nearest neighbor

1. Compute the distance between the test sample \mathbf{x} and all training samples $\{\mathbf{x}_i\}$
2. Find the sample \mathbf{x}_{NN} with minimum distance
3. Assign the corresponding label/value \mathbf{y}_{NN} to the test sample



Recap: NN: Properties

- The results may be sensitive to outliers
 - A point close to the outlier (blue circle in the center) will be misclassified
 - Solution: Look at multiple neighbors instead of just one



Recap: k-Nearest Neighbors

- Classification:
 1. Compute the distance between the test sample \mathbf{x} and all training samples $\{\mathbf{x}_i\}$
 2. Find the k samples $\{\mathbf{x}_{NN1}, \dots, \mathbf{x}_{NNk}\}$ with minimum distances
 3. Find the most common label among these k nearest neighbors (majority vote)
 4. Assign the corresponding label \mathbf{y}_{MV} to the test sample
- If several labels appear the same number of times among the k -NN, one strategy consists of taking the one whose samples have the smallest average distance to the test sample

Recap: k-Nearest Neighbors

- Regression:
 1. Compute the distance between the test sample \mathbf{x} and all training samples $\{\mathbf{x}_i\}$
 2. Find the k samples $\{\mathbf{x}_{NN1}, \dots, \mathbf{x}_{NNk}\}$ with minimum distances
 3. Compute the value \hat{y} for the test sample based on that of these k -NN
- Two strategies:
 - Use the average of the k -NN values $\{\mathbf{y}_{NNi}\}$
$$\hat{y} = \frac{1}{k} \sum_{i=1}^k \mathbf{y}_{NNi}$$
 - Use a weighted average of the k -NN values, based on the inverse distances

$$\hat{y} = \frac{1}{\sum_{i=1}^k w_{NNi}} \sum_{i=1}^k w_{NNi} \mathbf{y}_{NNi} \quad \text{with } w_{NNi} = \frac{1}{d(\mathbf{x}_{NNi}, \mathbf{x})}$$

Recap: k-Nearest Neighbors: Summary

- Advantages:
 - Simple method
 - Effective to handle nonlinear data
 - Only requires defining one parameter (k)
- Drawbacks:
 - The results depend on k
 - Becomes expensive as N and/or D grow
 - May be unreliable with a large D
- Let us now look at a different way to handle nonlinear data

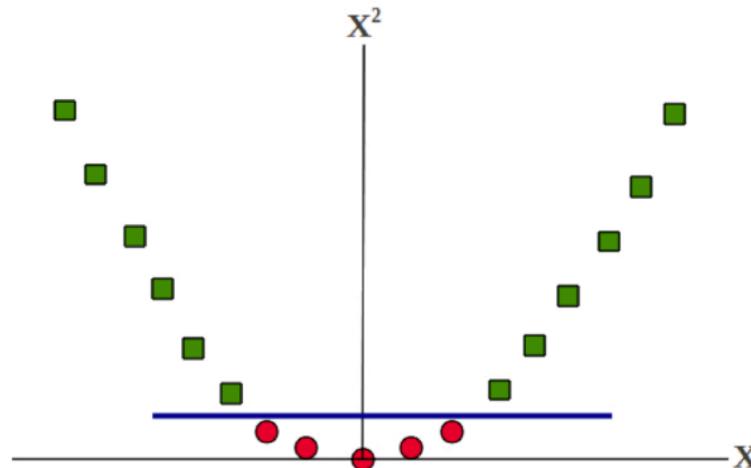
Recap: Nonlinear classification

- How can we handle this data?
 - 1D input, 2 classes (colors)



- How about transforming the input?

$$x \rightarrow [x, x^2]$$



Recap: Polynomial curve fitting

- We seek to find a polynomial function that approximates a true curve using some observations
- Such a polynomial function of degree M can be written as

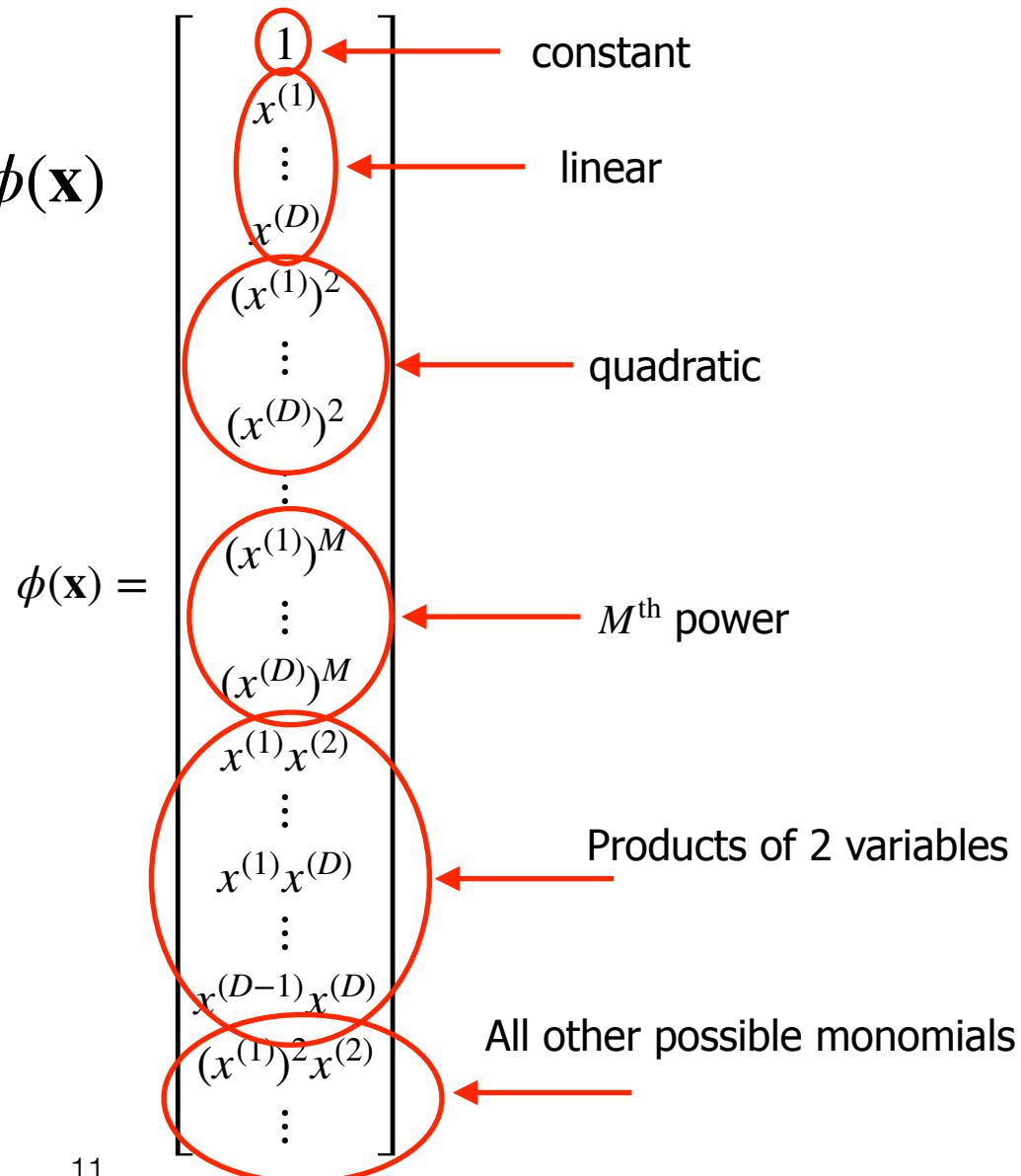
$$y = w^{(0)} + w^{(1)}x + w^{(2)}x^2 + \dots + w^{(M)}x^M = \sum_{j=0}^M w^{(j)}x^j$$

where the $\{w^{(j)}\}$ are the coefficients of the different terms

- For example:
 - For $M = 0$, we have the constant function $y = w^{(0)}$
 - For $M = 1$, we have the line $y = w^{(0)} + w^{(1)}x$
 - For $M = 2$, we have the quadratic function $y = w^{(0)} + w^{(1)}x + w^{(2)}x^2$

Recap: Polynomial feature expansion

- In general, for D -dimensional inputs, we can obtain a large $\phi(\mathbf{x})$
- The resulting expanded features can then be used in any linear algorithm
 - Let us look at how this would work for linear regression



Recap: Working with expanded features

- In essence, polynomial feature expansion uses a mapping from $\mathbf{x} \in \mathbb{R}^D$ to another representation $\phi(\mathbf{x}) \in \mathbb{R}^F$, where $F \gg D$
- We can then use a linear model in this new space and write

$$\hat{y}_i = \mathbf{w}^T \phi(\mathbf{x}_i)$$

where now $\mathbf{w} \in \mathbb{R}^F$

(assuming that the 1 accounting for the bias has been incorporated in $\phi(\mathbf{x}_i)$)

- Given training data, we can then find the optimal parameter via standard linear regression:

$$\min_{\mathbf{w}} \sum_{i=1}^N (\mathbf{w}^T \phi(\mathbf{x}_i) - y_i)^2$$

Discussion: Why is higher dimension good?

- Intuitively, for classification, increasing the dimensionality gives more freedom for the data to be separable by a hyperplane
- More formally: Cover's theorem:

A complex pattern-classification problem, cast in a high-dimensional space nonlinearly, is more likely to be linearly separable than in a low-dimensional space, provided that the space is not densely populated

- T.M. Cover, Geometrical and statistical properties of systems of linear inequalities with applications in pattern recognition, IEEE Trans Electron Comput, 1965

Goals of today's lecture

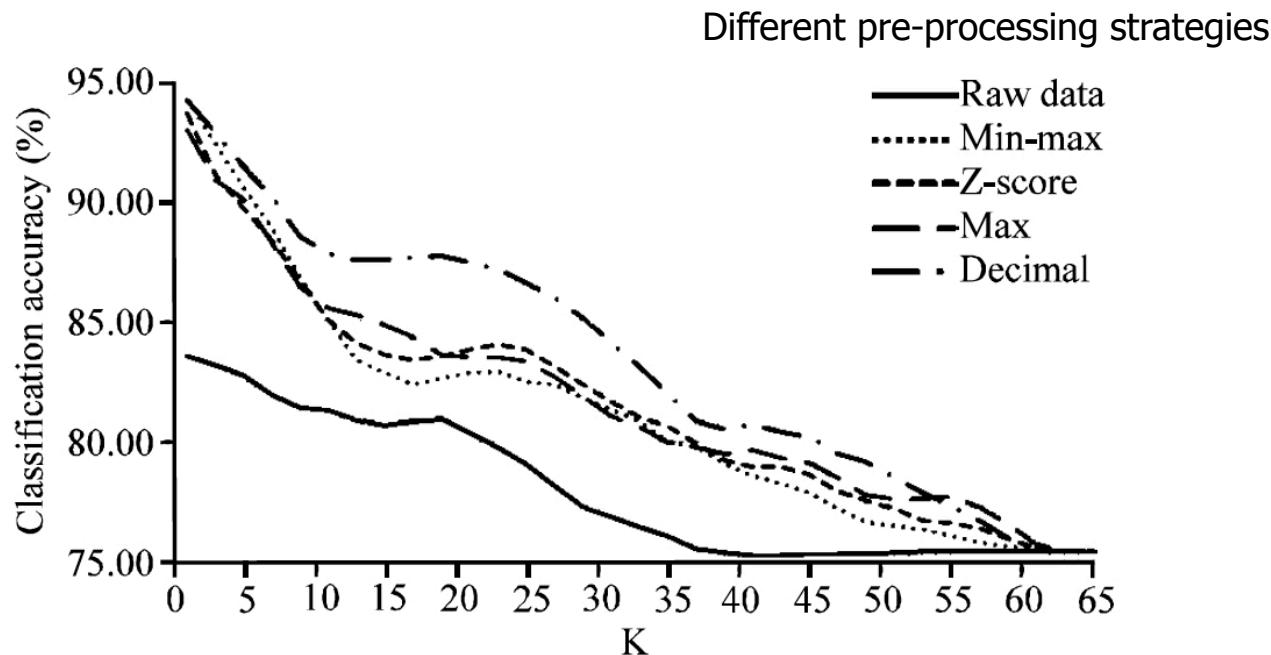
- Understand the notions of model complexity and overfitting
- Introduce several strategies to select a model that does not overfit
- Introduce the notion of regularization to prevent overfitting

Model complexity

- As we have moved from linear models to nonlinear ones, we have obtained increasing flexibility
 - We do not longer assume that the output linearly depends on the input
 - We can handle classification problems in which the classes are nonlinearly separable
- This flexibility, however, may sometimes make the model too complex for the task at hand
 - Let us look at this for two different models: k-Nearest Neighbors and polynomial regression

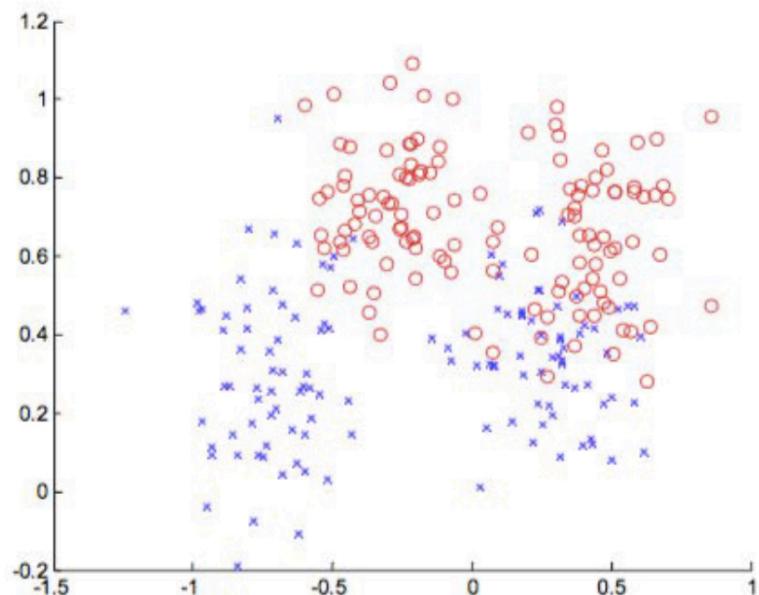
Recap: k-Nearest Neighbors

- Results depend on the value k
 - E.g., Ma et al., Journal of Applied Sciences, 2014
 - UCI Parkinson dataset: Predict if a patient suffers from Parkinson's disease from several biomedical voice measurements

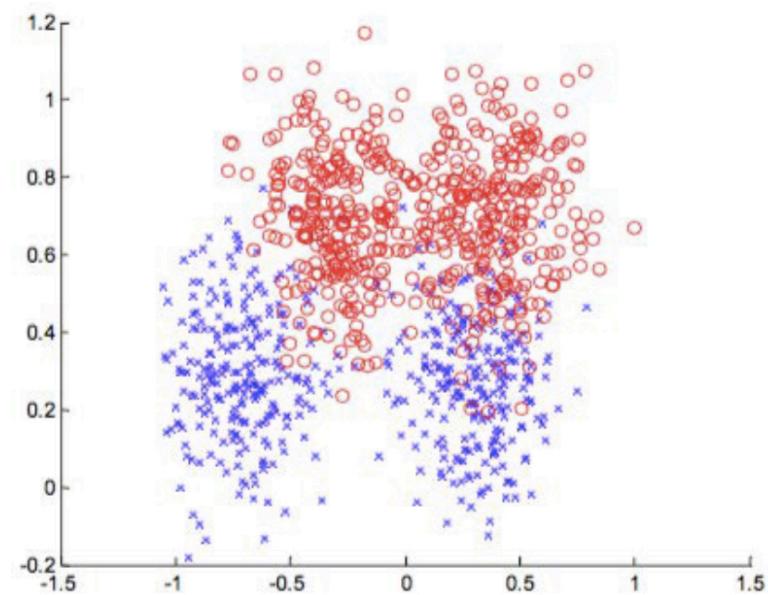


k -NN: Training vs test

- Toy example: 2D inputs from 2 classes (colors)



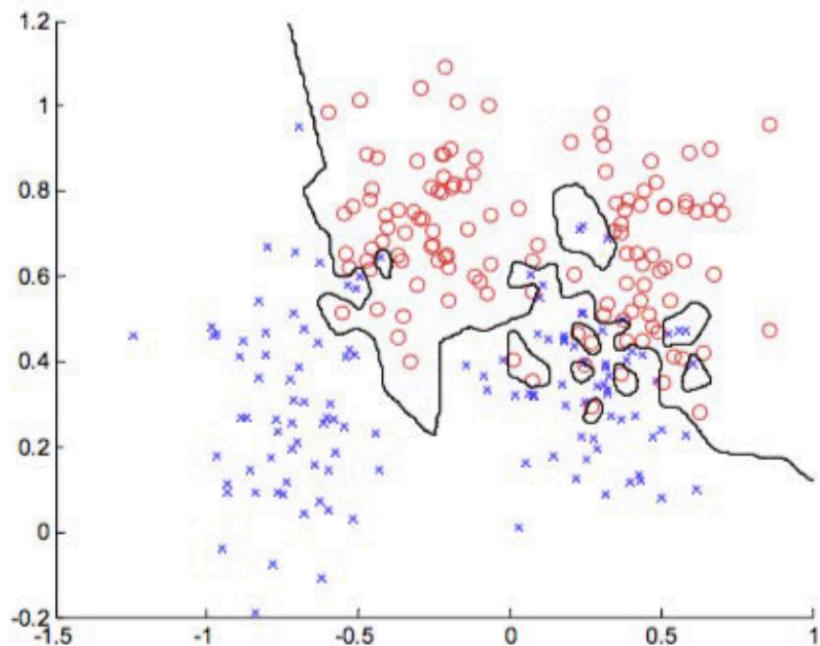
Training data



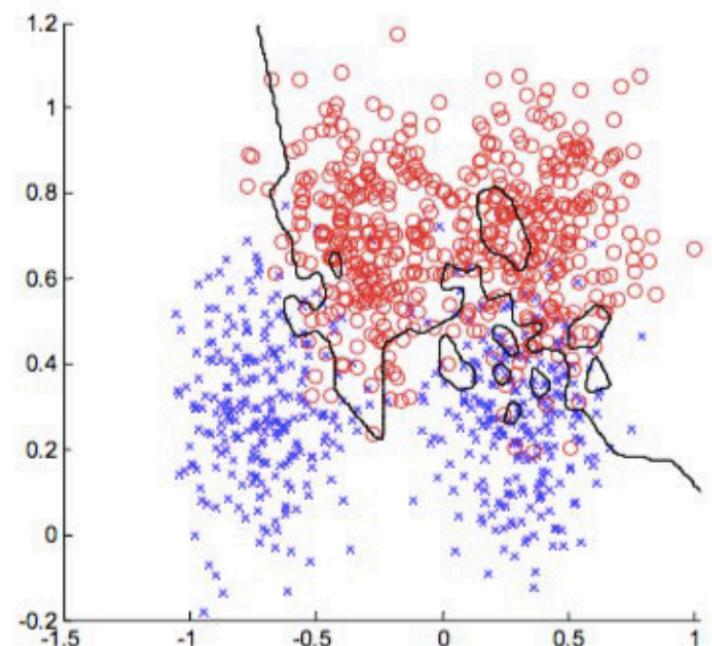
Test data

k -NN: Training vs test

- $k = 1$



Training data
error = 0.0

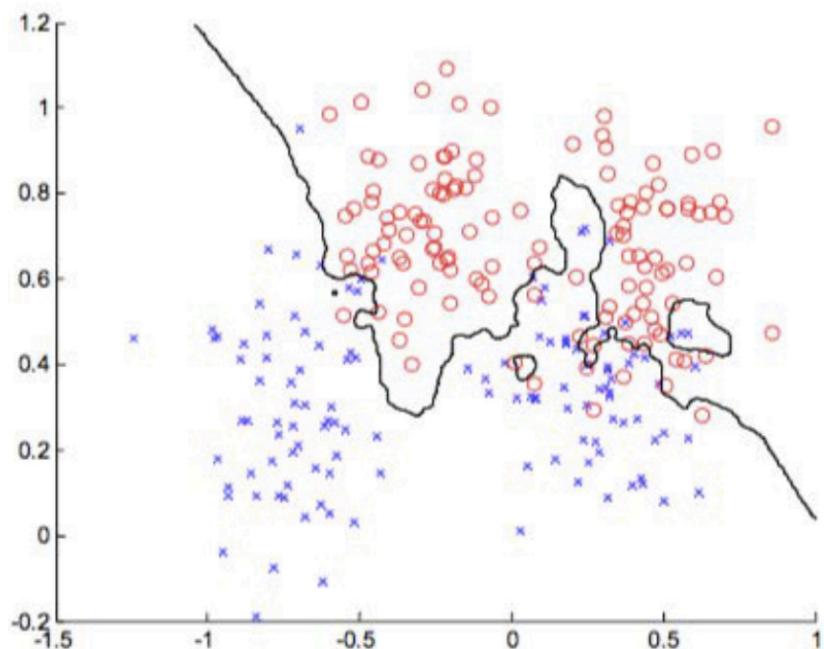


Test data
error = 0.15

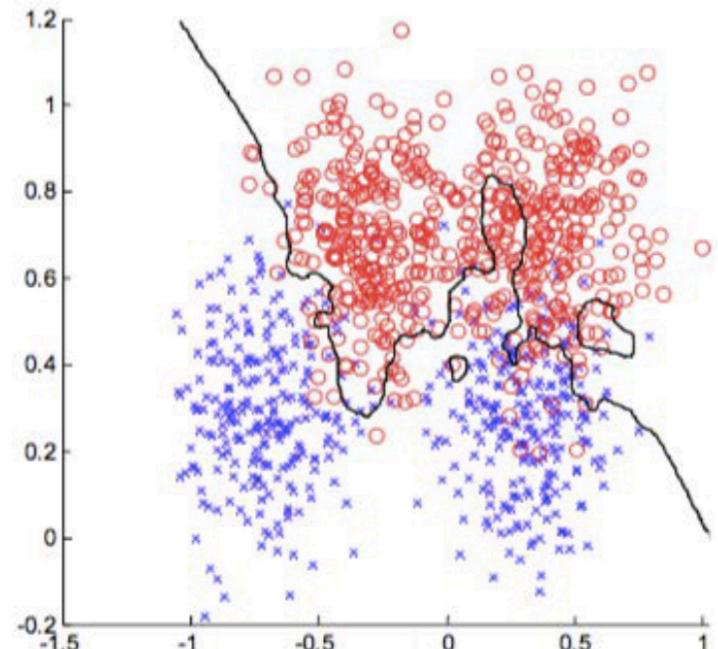
The training samples are classified perfectly (as expected), but not the test samples

k -NN: Training vs test

- $k = 3$



Training data
error = 0.076

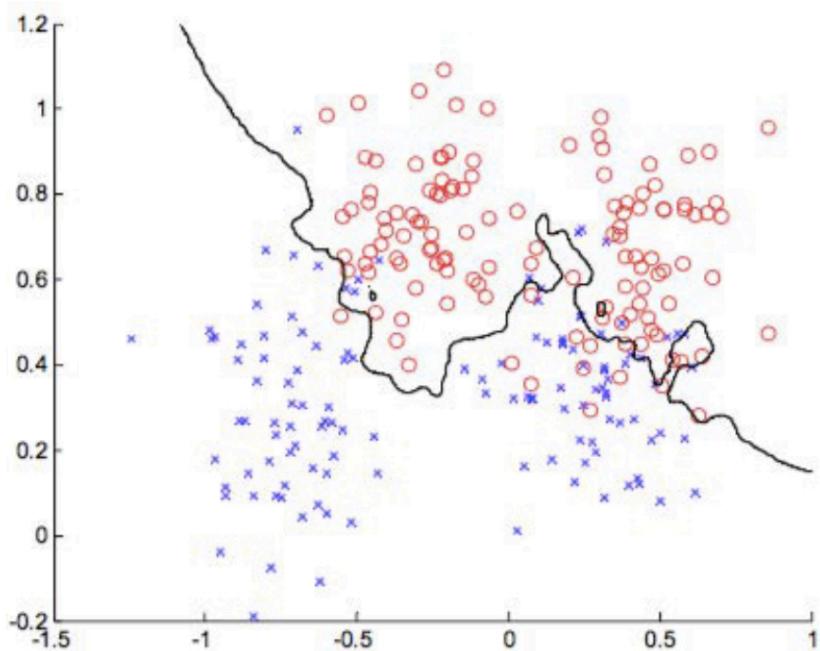


Test data
error = 0.134

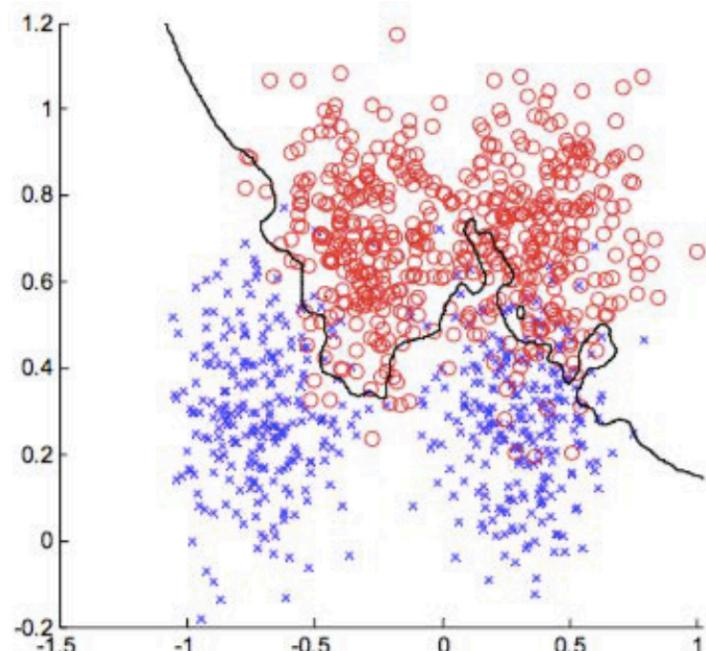
The boundary becomes smoother. The training error increases, but the test error decreases

k -NN: Training vs test

- $k = 7$



Training data
error = 0.132

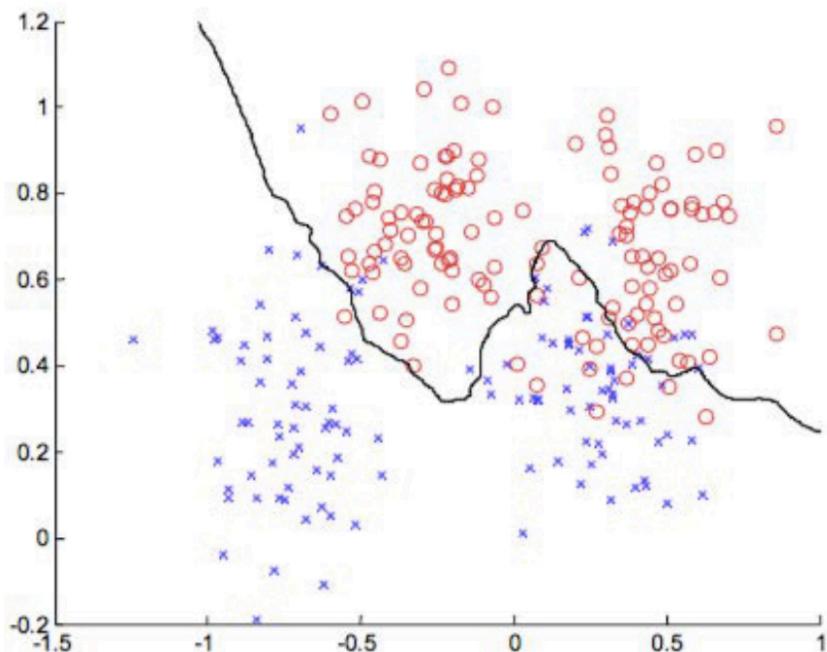


Test data
error = 0.111

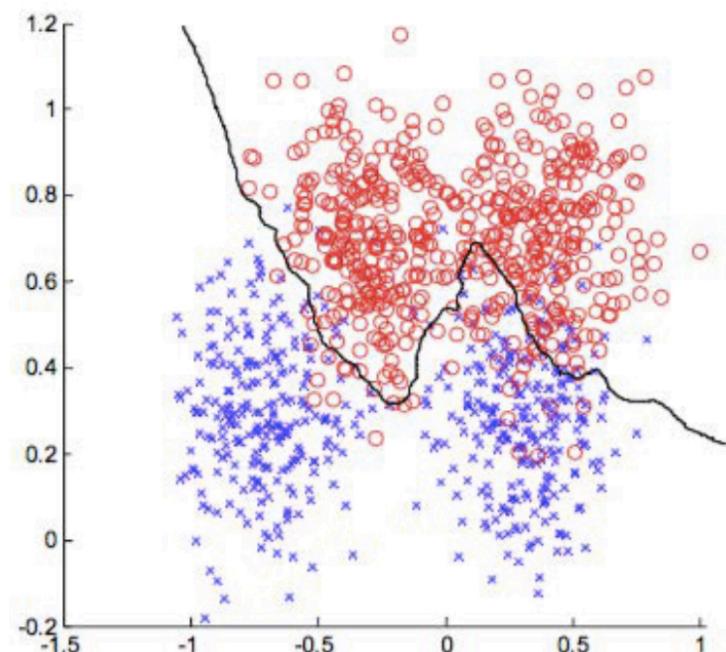
The boundary becomes smoother. The training error increases, but the test error decreases

k -NN: Training vs test

- $k = 21$



Training data
error = 0.112

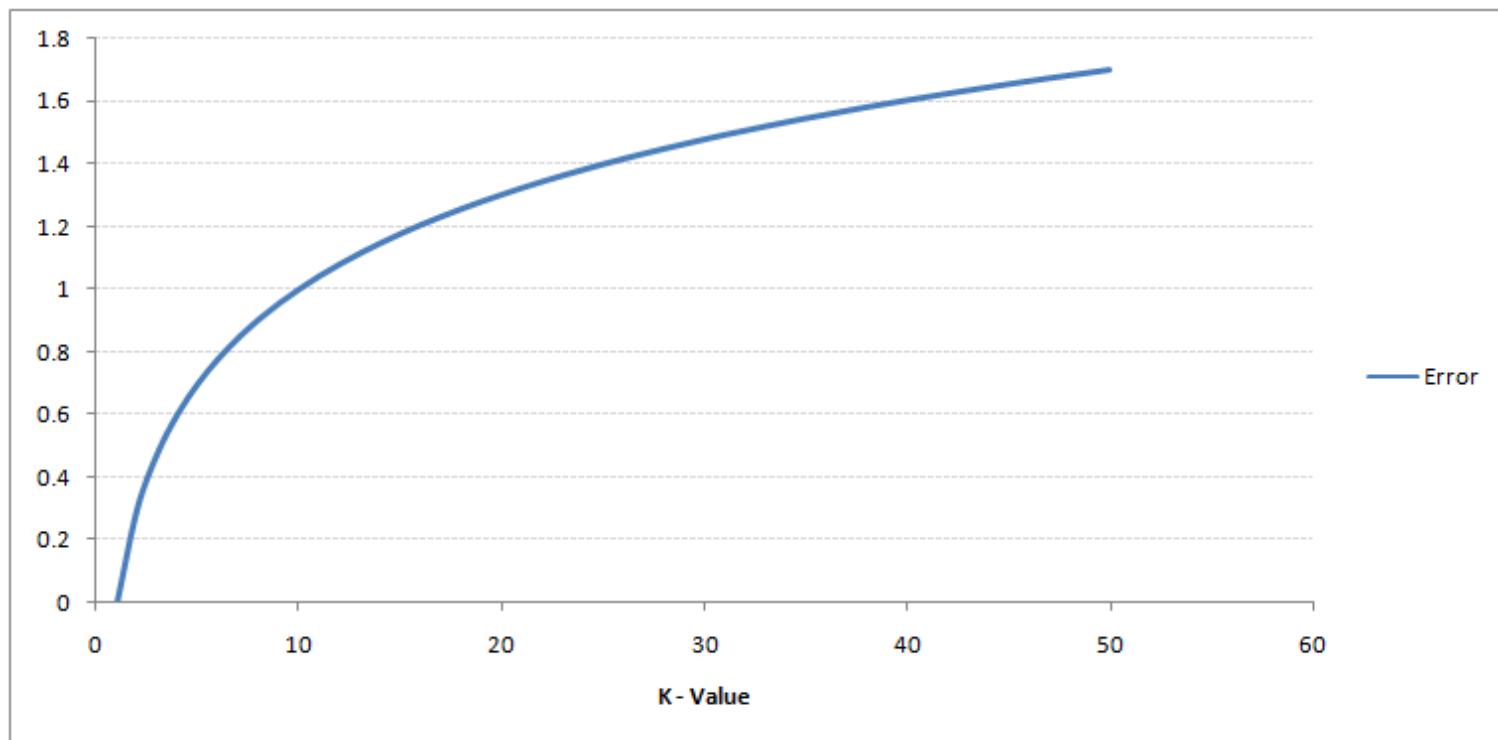


Test data
error = 0.092

The test error decreases again but would start increasing if we chose k even larger

k -NN: Training vs test

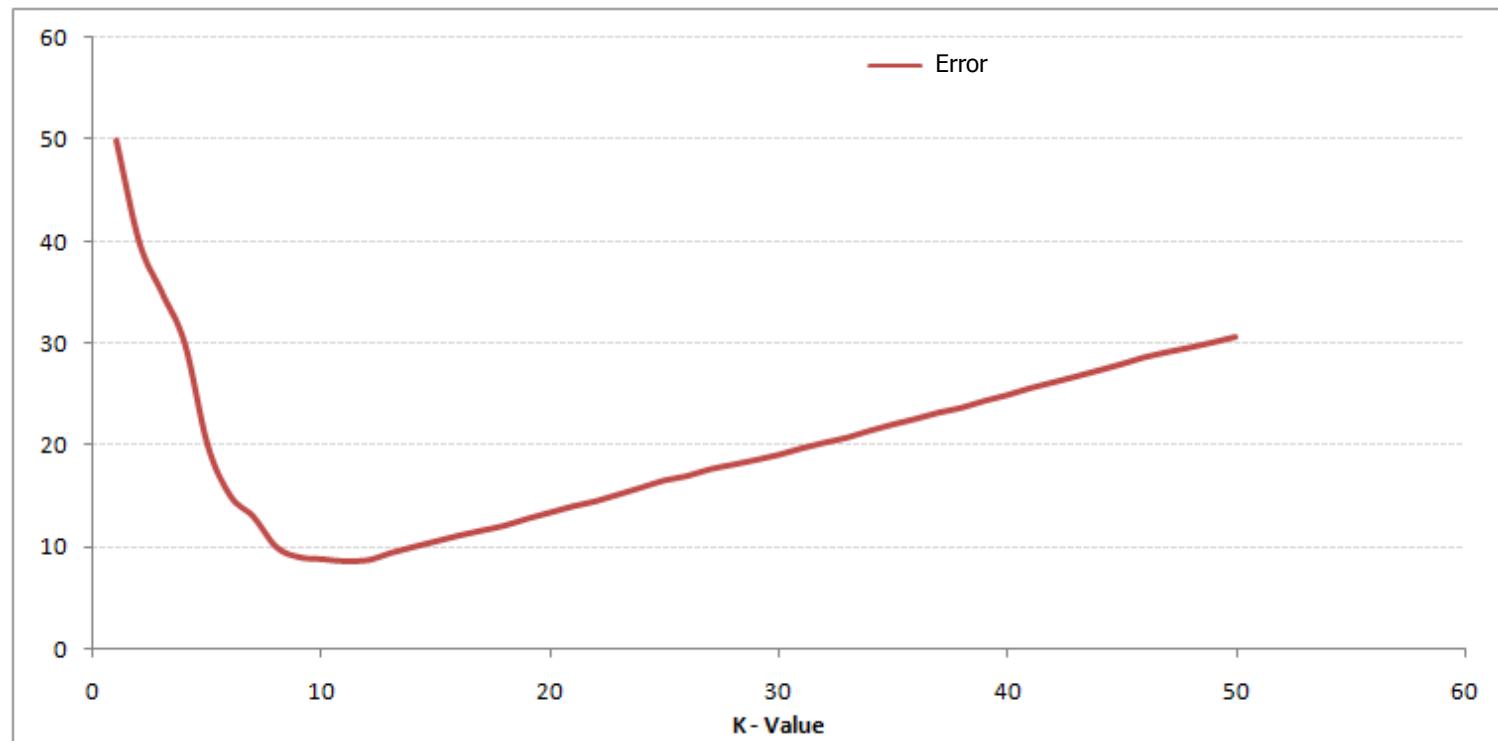
- Let's look at the training and test curves (from T. Srivastava @ AnalyticsVidhya)



Training error

k -NN: Training vs test

- Let's look at the training and test curves (from T. Srivastava @ AnalyticsVidhya)



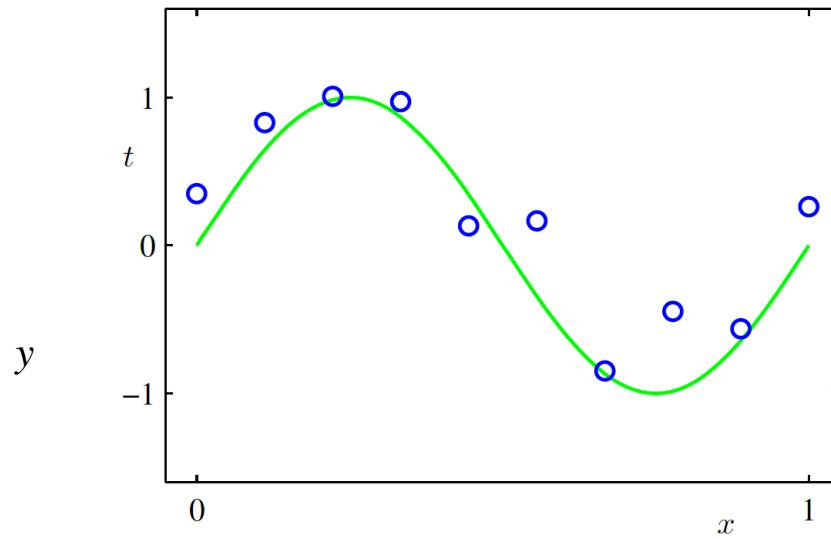
Test error

Overfitting vs Underfitting

- For $k = 1$, the error is 0 on the training data, but large on the test data. This is called **overfitting**.
- For large k values, the error is large on both the training and the test data. This is called **underfitting**.
- This behavior is **not** specific to k -NN. It can happen in most ML algorithms.

E.g., Polynomial curve fitting (Bishop 1.1)

- We have noisy training observations (blue circles, 1D input, 1D output) coming from the true green curve



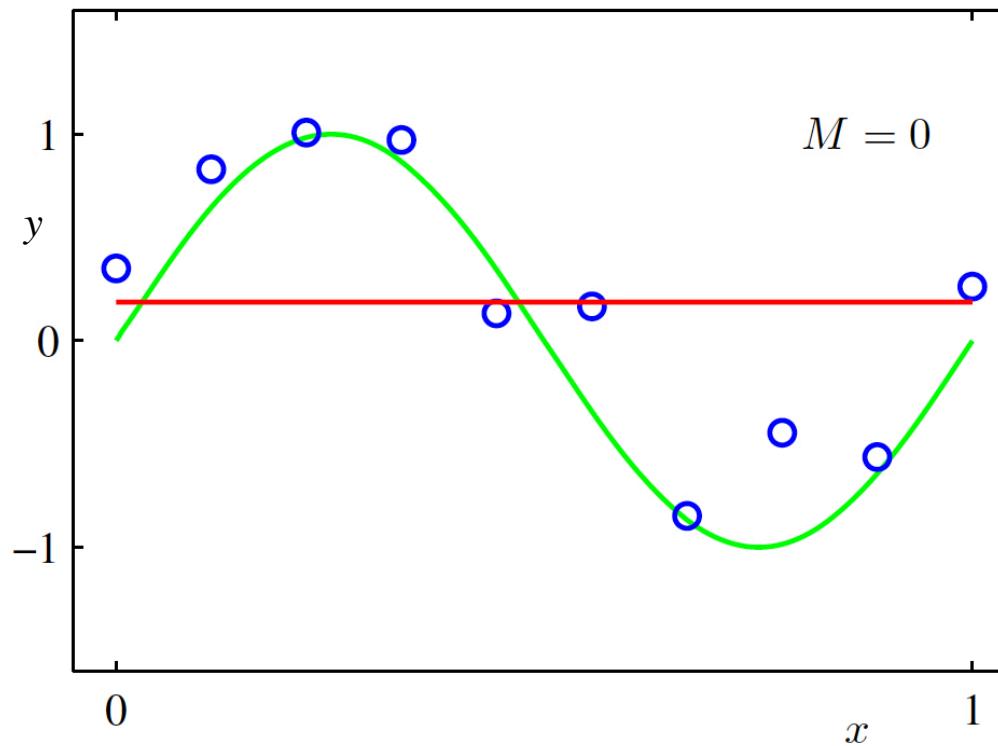
- We seek to approximate these observations with a polynomial function of degree M

$$y = w^{(0)} + w^{(1)}x + w^{(2)}x^2 + \dots + w^{(M)}x^M = \sum_{j=0}^M w^{(j)}x^j$$

where the $\{w^{(j)}\}$ are the coefficients of the different terms

E.g., Polynomial curve fitting (Bishop 1.1)

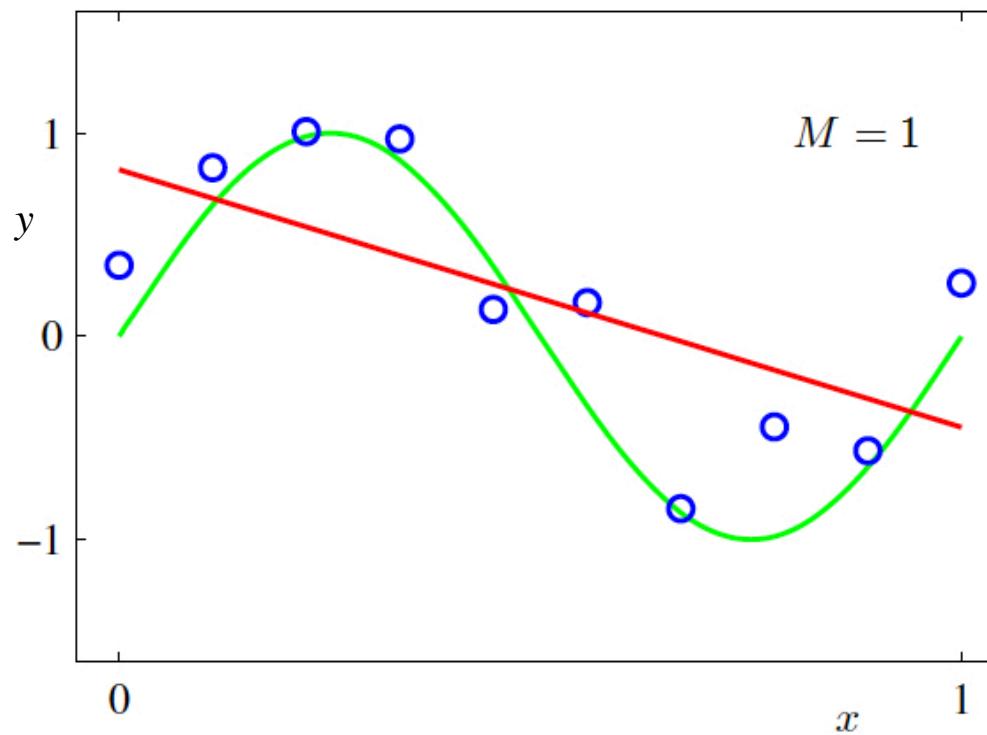
- $M = 0$



Underfitting: Approximates the observations poorly and is far from the (unknown) true curve

E.g., Polynomial curve fitting (Bishop 1.1)

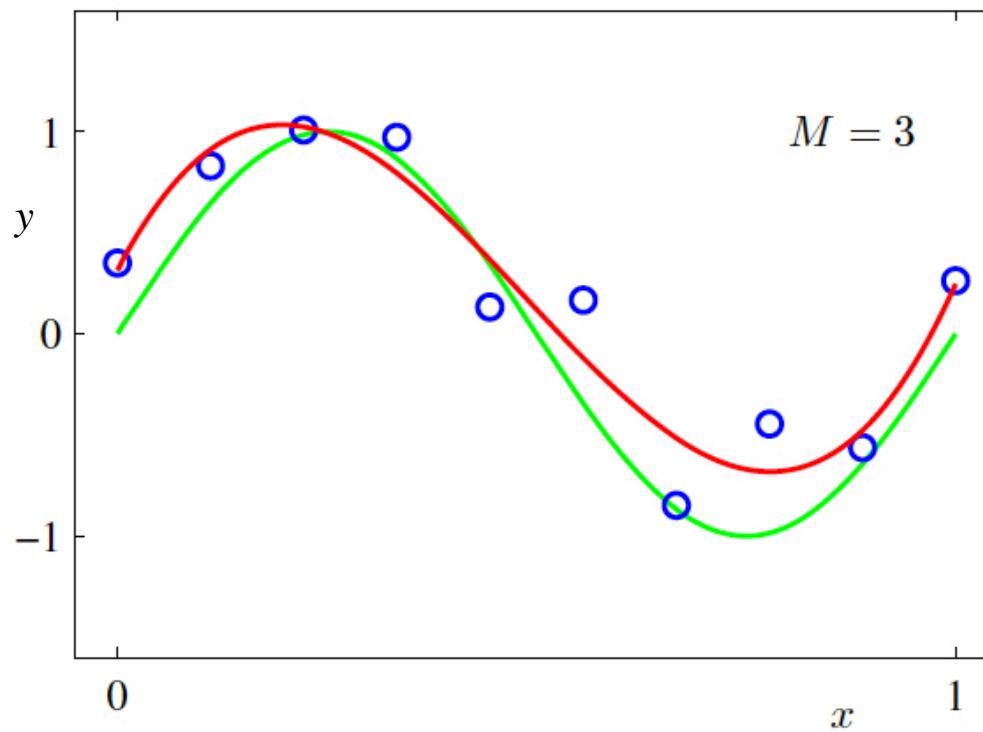
- $M = 1$



Underfitting: Approximates the observations poorly and is far from the (unknown) true curve

E.g., Polynomial curve fitting (Bishop 1.1)

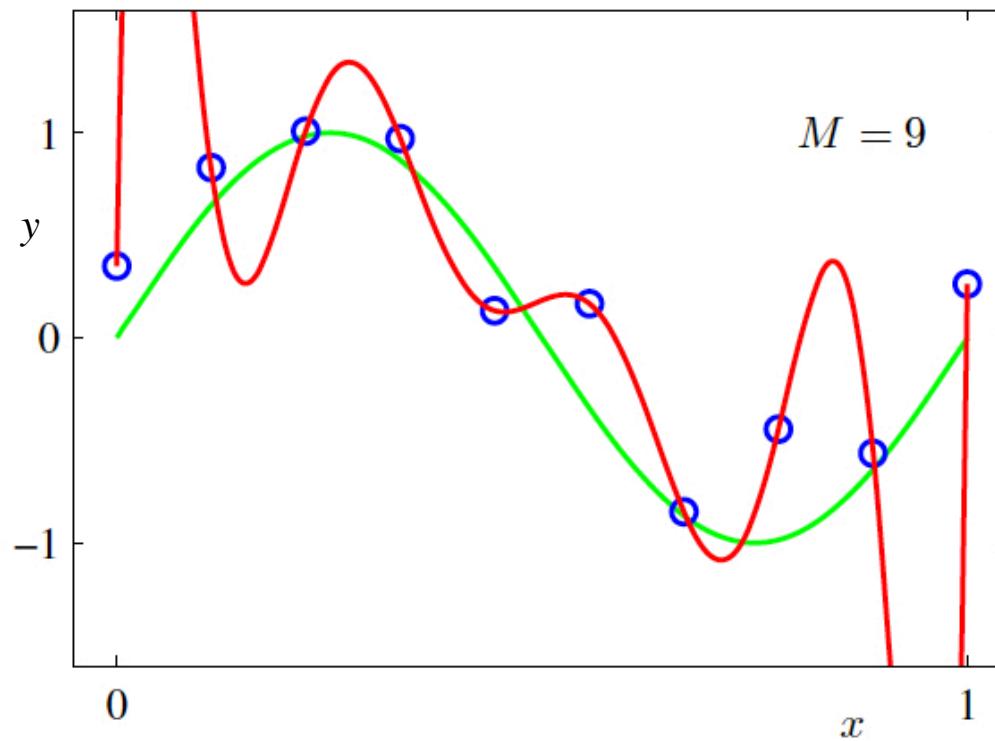
- $M = 3$



Fitting: Approximates the observations well and is close to the (unknown) true curve

E.g., Polynomial curve fitting (Bishop 1.1)

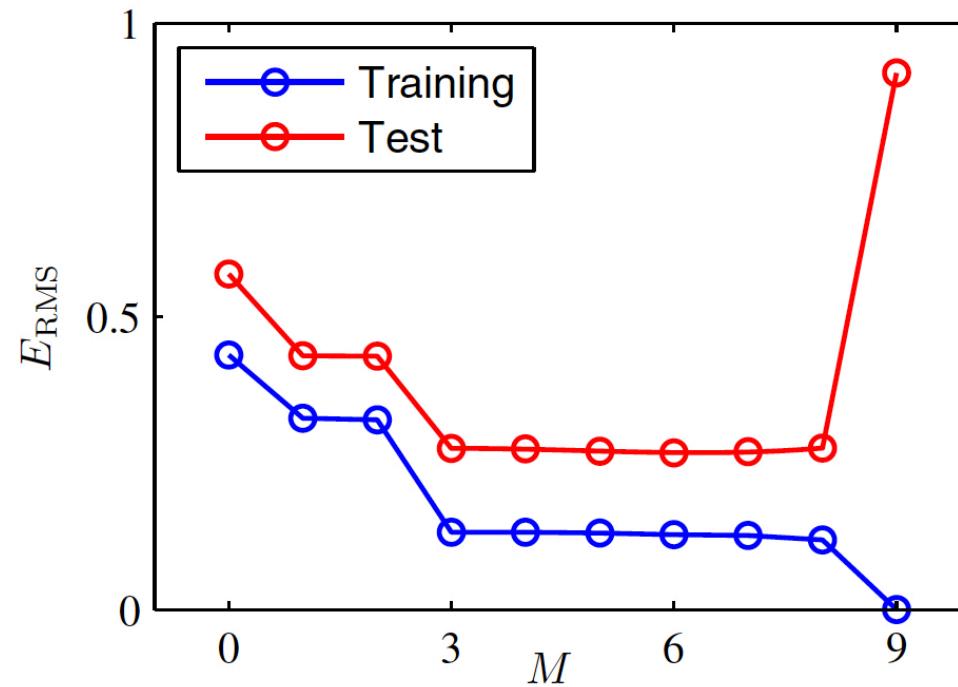
- $M = 9$



Overfitting: Approximates the observations perfectly but is far from the (unknown) true curve

E.g., Polynomial curve fitting (Bishop 1.1)

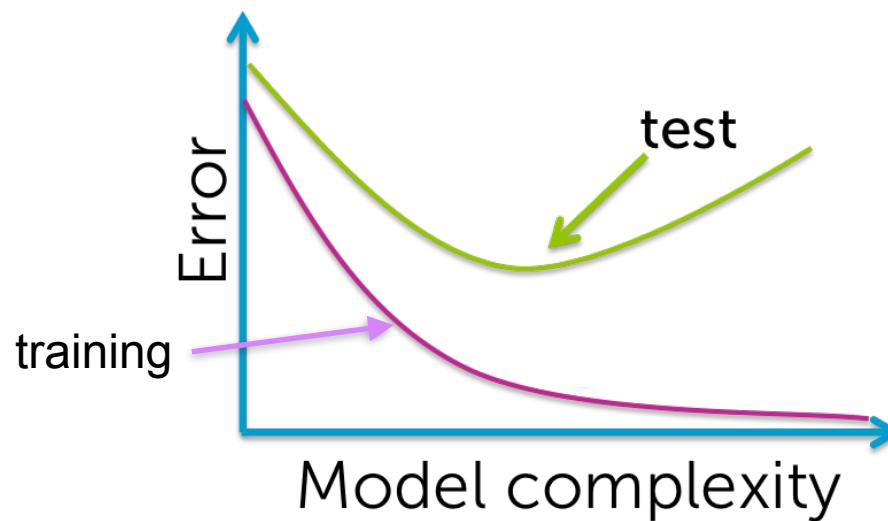
- Let's look at the training and test error curves



- While the training curve always decreases as the degree increases, the test curve eventually increases again

Overfitting vs Underfitting

- General phenomenon

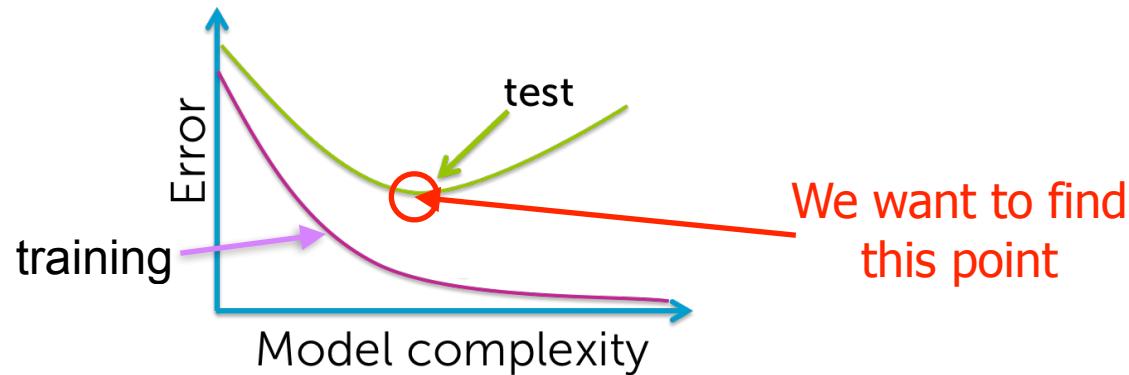


- Model complexity, e.g.
 - A high degree polynomial is more complex than a low degree one
 - In k -NN, $k = 1$ is more complex (the boundary is less smooth) than $k = 21$

Model selection

- The question then is:

Since we don't have access to the test data, **how do we choose the right complexity for the model?**

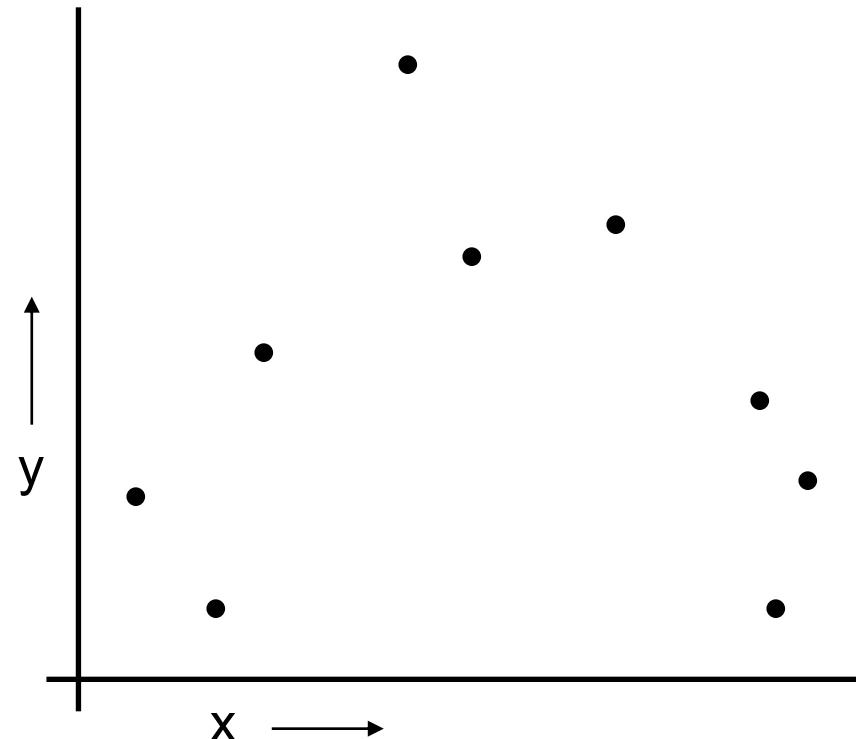


- Solution: **Cross-validation**

- The following material was adapted from Andrew Moore's cross-validation slides

Cross-validation: Toy example

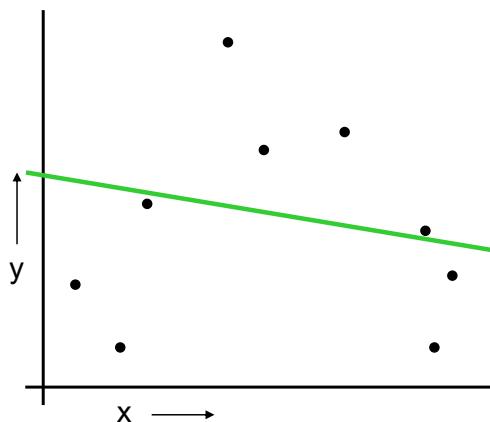
- Regression problem: 1D input x , 1D output y
 - We are given the following training observations



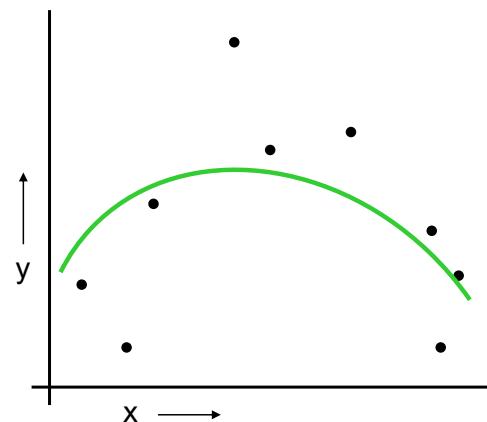
Cross-validation: Toy example

- We consider 3 different models (of increasing complexity)

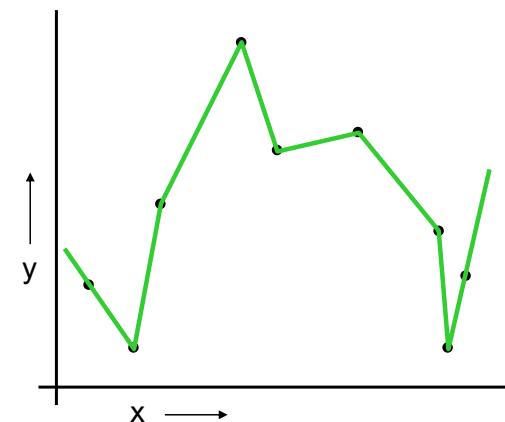
a linear function (line)



a quadratic function



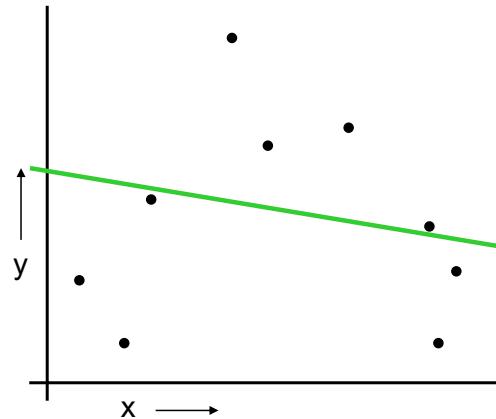
a piecewise linear function (join-the-dots)



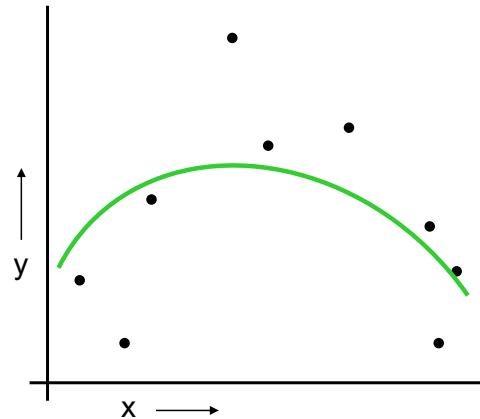
- Which is best?

Cross-validation: Toy example

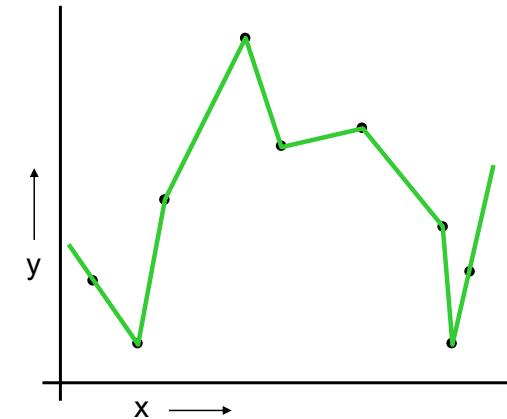
a linear function (line)



a quadratic function

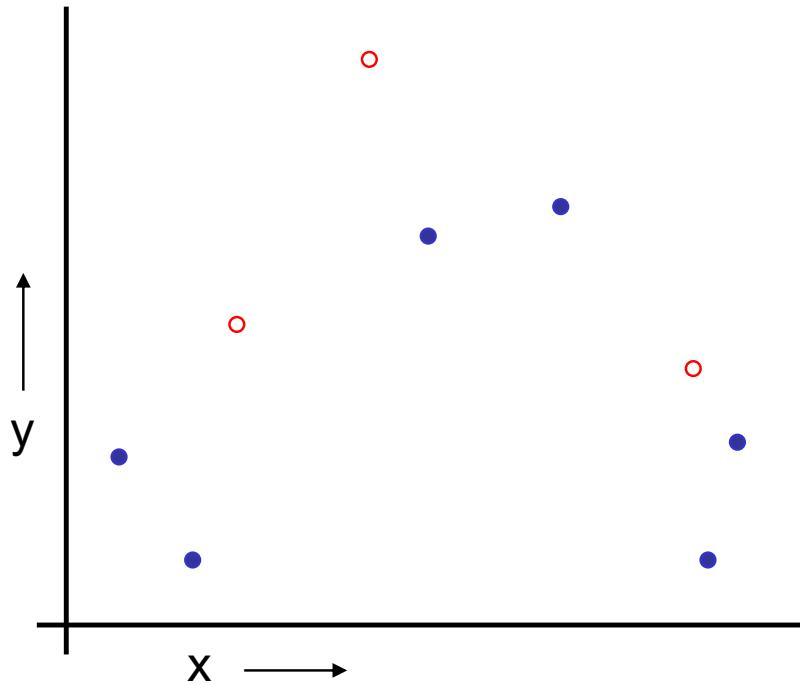


a piecewise linear function (join-the-dots)



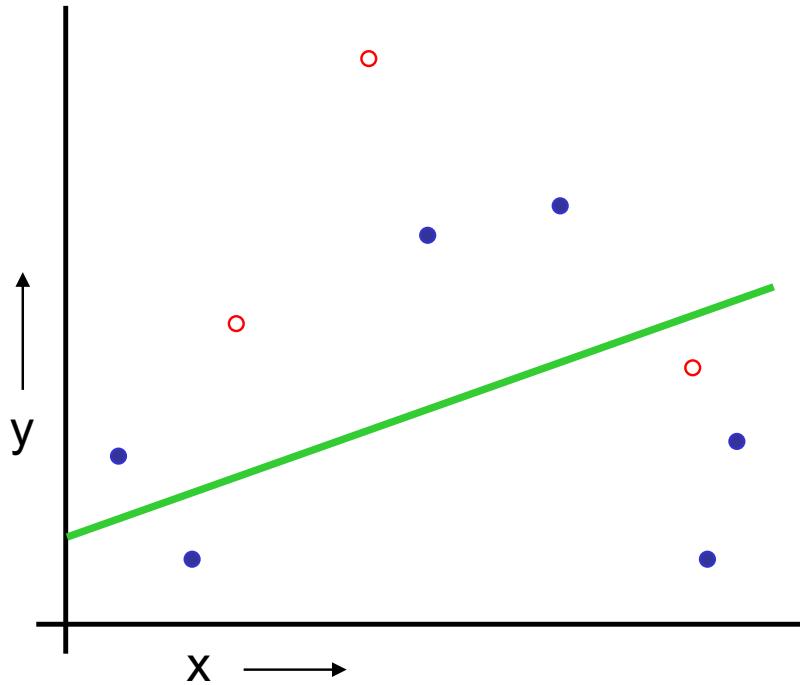
- Why not choose the one that best fits the data?
- What we truly care about is
How well is the model going to predict future data drawn from the same distribution?
- Let us look at 3 ways to simulate this with the training data

The validation set method



1. Randomly choose, e.g., 30% of the data to form a **validation set**
2. The remainder then acts as your new **training set**

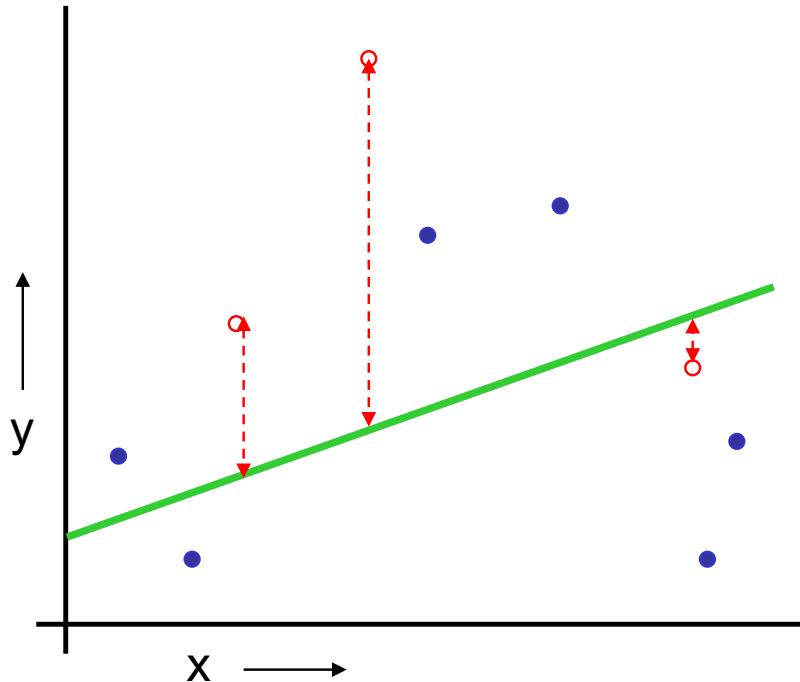
The validation set method



E.g., Fit a line

1. Randomly choose, e.g., 30% of the data to form a **validation set**
2. The remainder then acts as your new **training set**
3. Perform regression on this new training set only

The validation set method

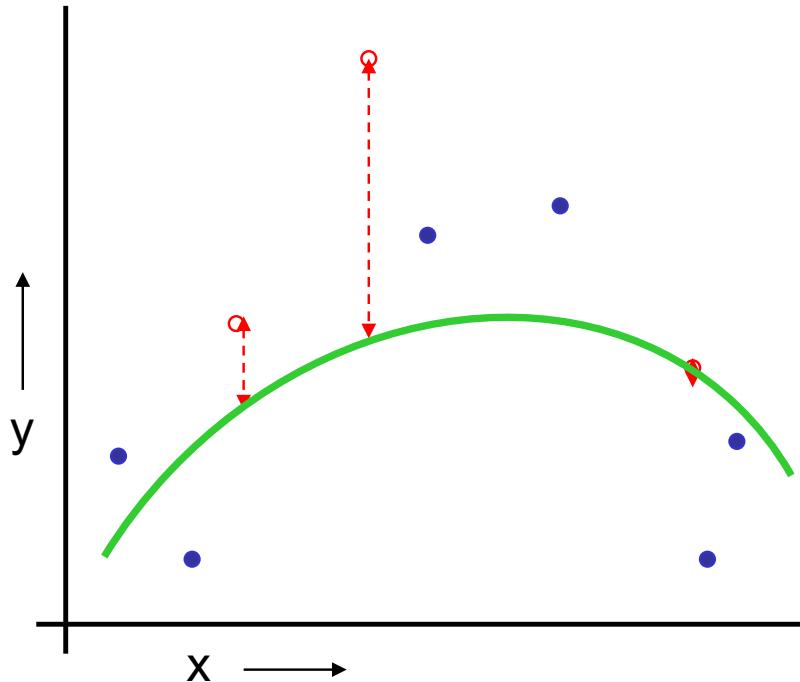


E.g., Fit a line

Mean squared error: 2.4

1. Randomly choose, e.g., 30% of the data to form a **validation set**
2. The remainder then acts as your new **training set**
3. Perform regression on this new training set only
4. Estimate the performance on the test data using the validation set

The validation set method

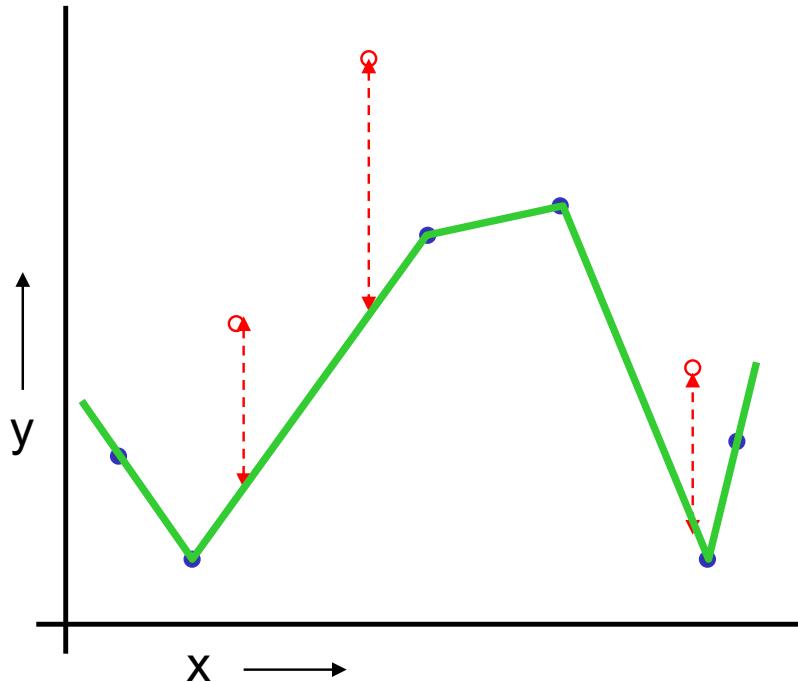


1. Randomly choose, e.g., 30% of the data to form a **validation set**
2. The remainder then acts as your new **training set**
3. Perform regression on this new training set only
4. Estimate the performance on the test data using the validation set

E.g., Fit a quadratic function

Mean squared error: 0.9

The validation set method



E.g., Join-the-dots function

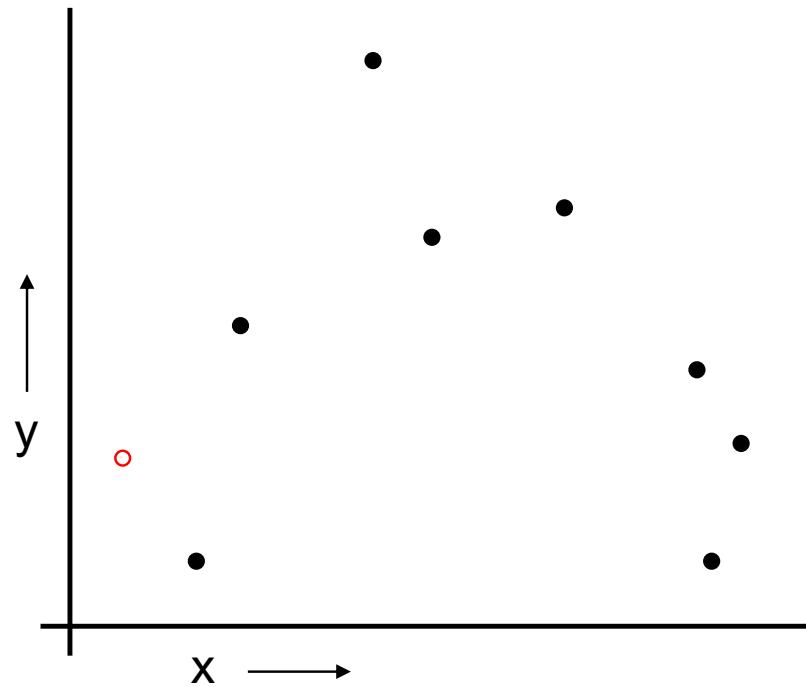
Mean squared error: 2.2

1. Randomly choose, e.g., 30% of the data to form a **validation set**
2. The remainder then acts as your new **training set**
3. Perform regression on this new training set only
4. Estimate the performance on the test data using the validation set

The validation set method

- Advantages:
 - Very simple
 - We can choose the model based on the validation error
- Question: What do you see as drawbacks?

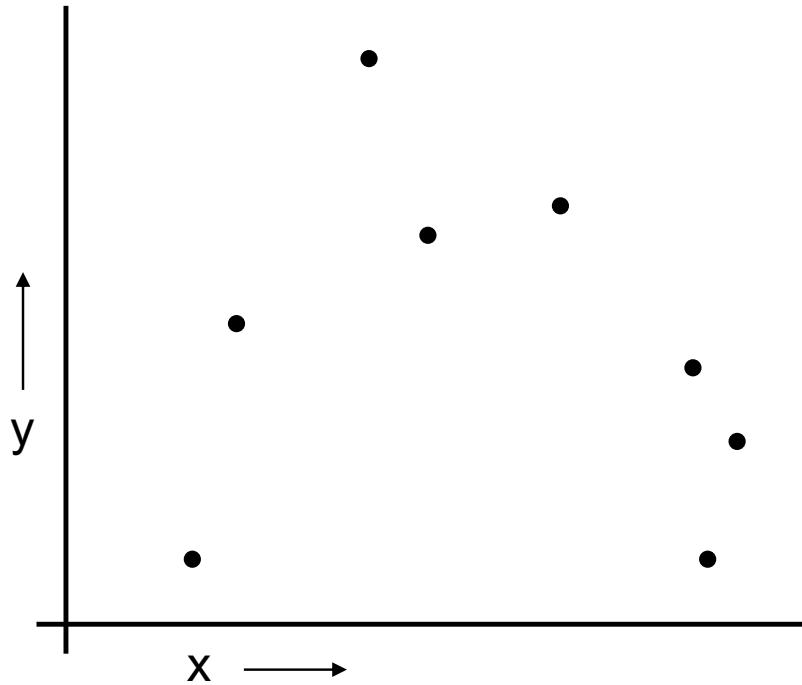
Leave-one-out cross-validation (LOOCV)



For $i = 1, \dots, N$

1. Let (x_i, y_i) be the i^{th} sample

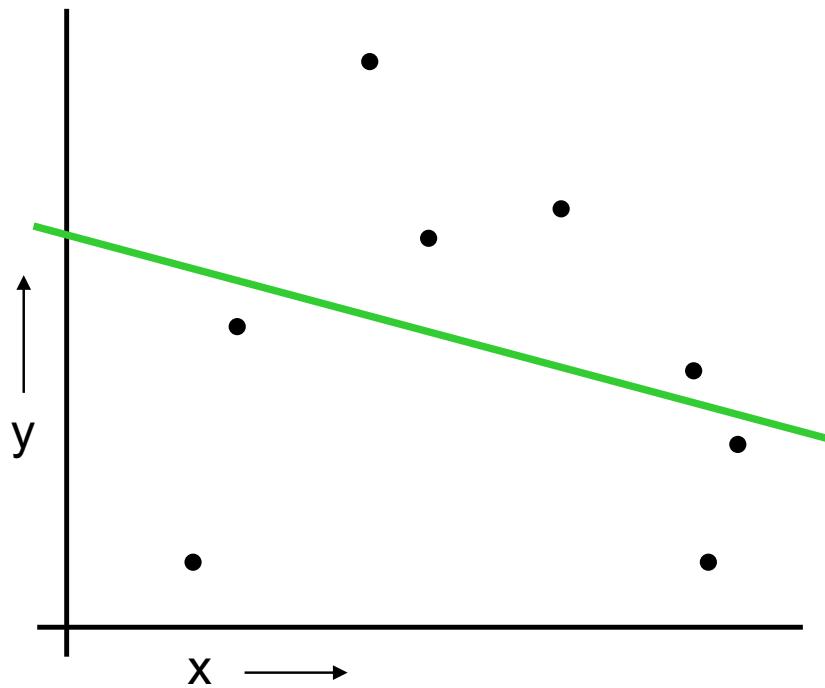
Leave-one-out cross-validation (LOOCV)



For $i = 1, \dots, N$

1. Let (x_i, y_i) be the i^{th} sample
2. Temporarily remove (x_i, y_i) from the training set

Leave-one-out cross-validation (LOOCV)

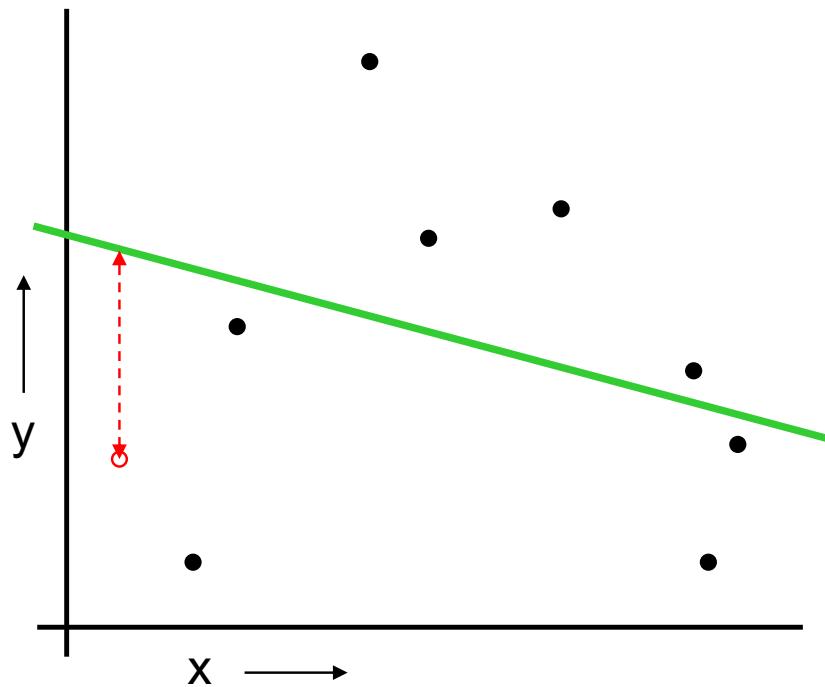


E.g., Fit a line

For $i = 1, \dots, N$

1. Let (x_i, y_i) be the i^{th} sample
2. Temporarily remove (x_i, y_i) from the training set
3. Perform regression on the remaining $N - 1$ samples

Leave-one-out cross-validation (LOOCV)

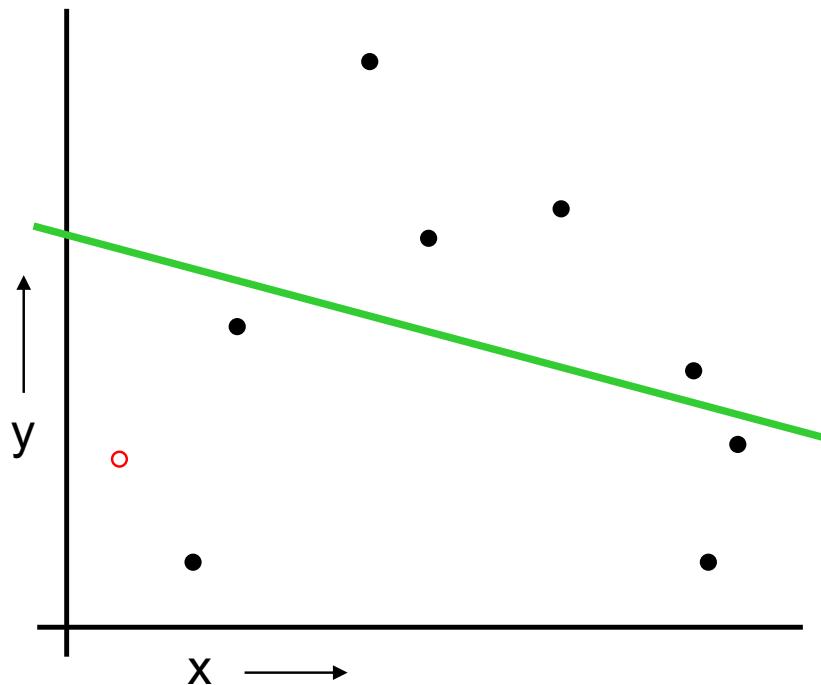


E.g., Fit a line

For $i = 1, \dots, N$

1. Let (x_i, y_i) be the i^{th} sample
2. Temporarily remove (x_i, y_i) from the training set
3. Perform regression on the remaining $N - 1$ samples
4. Compute the error on (x_i, y_i)

Leave-one-out cross-validation (LOOCV)

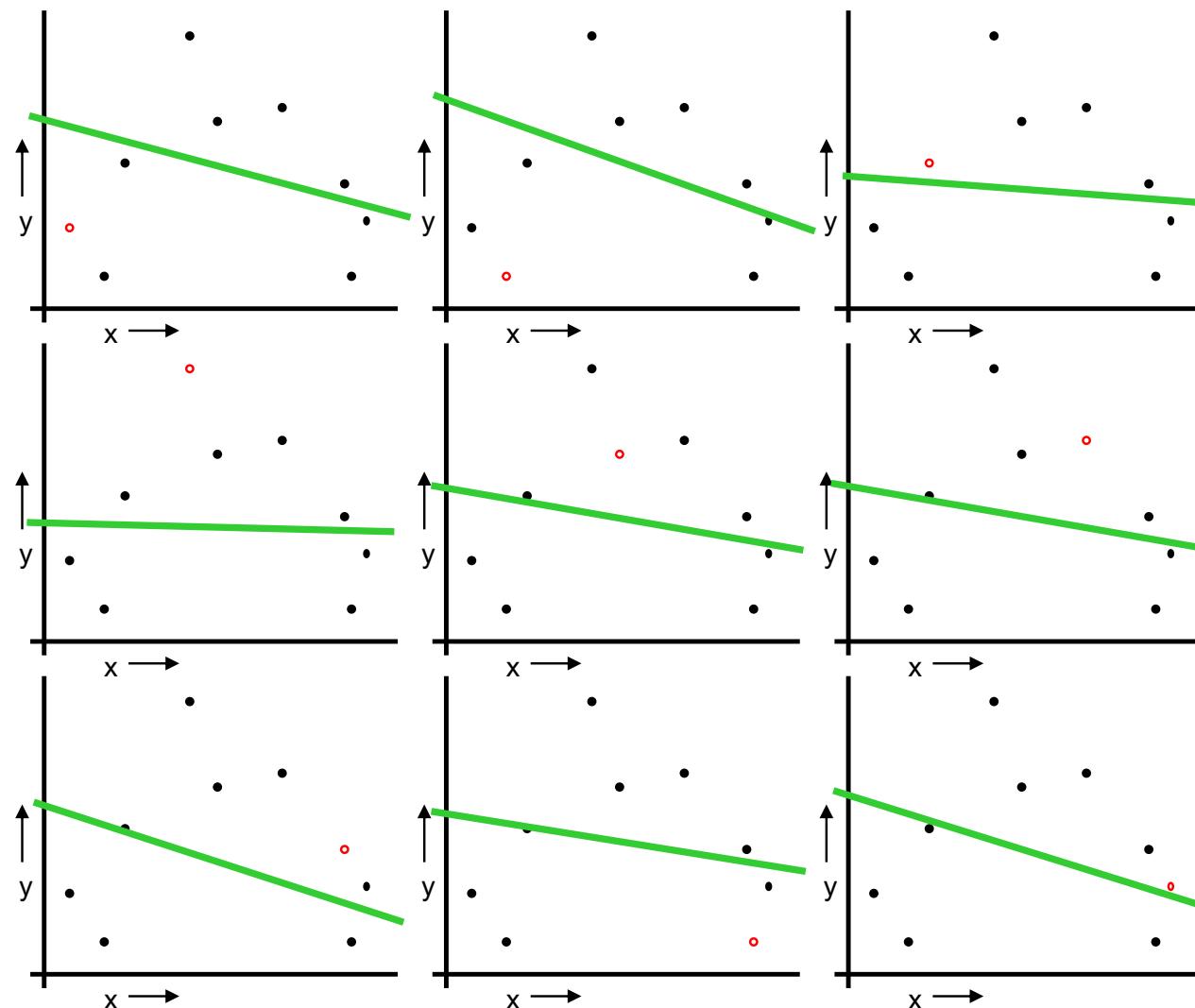


For $i = 1, \dots, N$

1. Let (x_i, y_i) be the i^{th} sample
2. Temporarily remove (x_i, y_i) from the training set
3. Perform regression on the remaining $N - 1$ samples
4. Compute the error on (x_i, y_i)

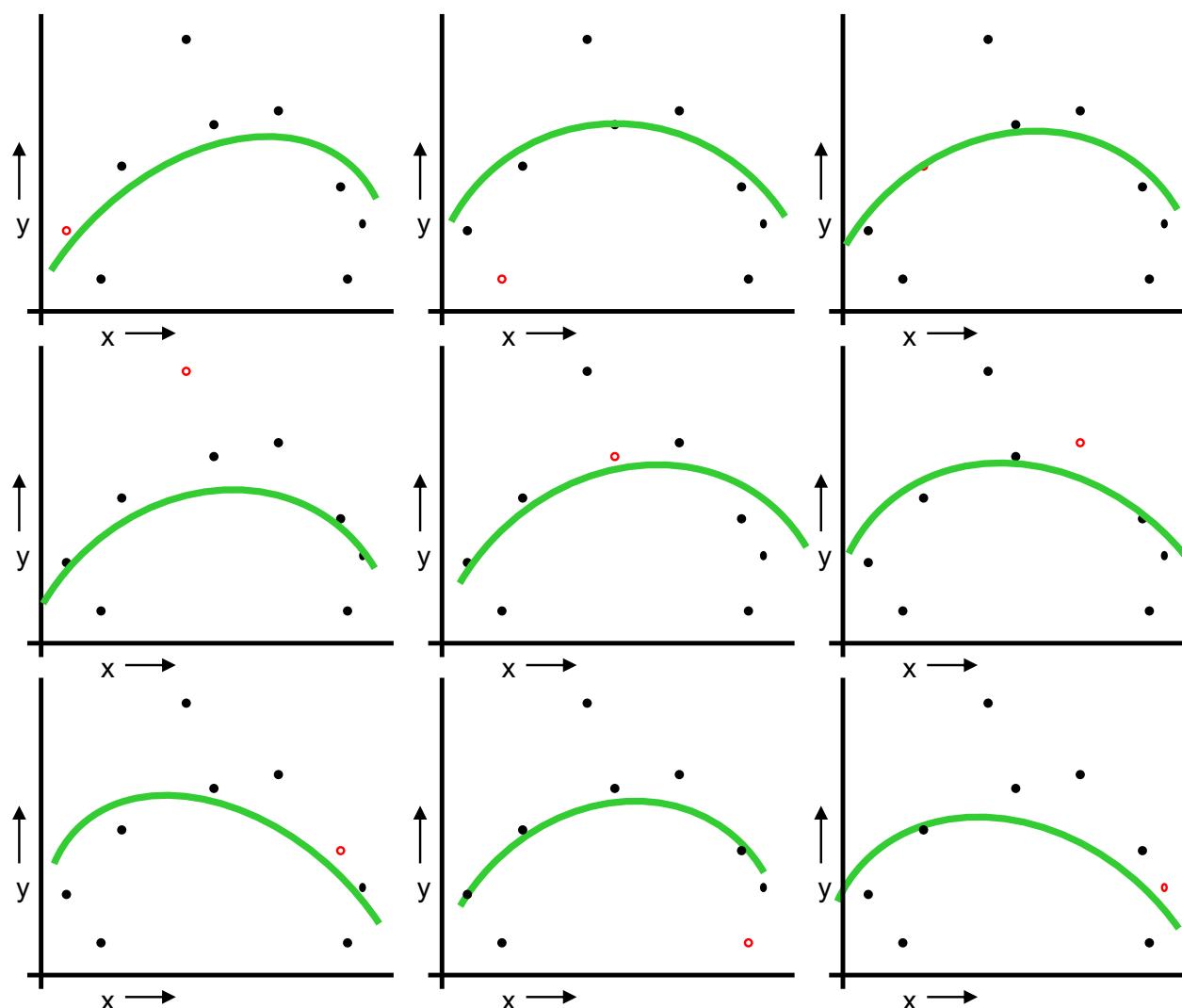
When you have done all points, report the average error

Leave-one-out cross-validation (LOOCV)



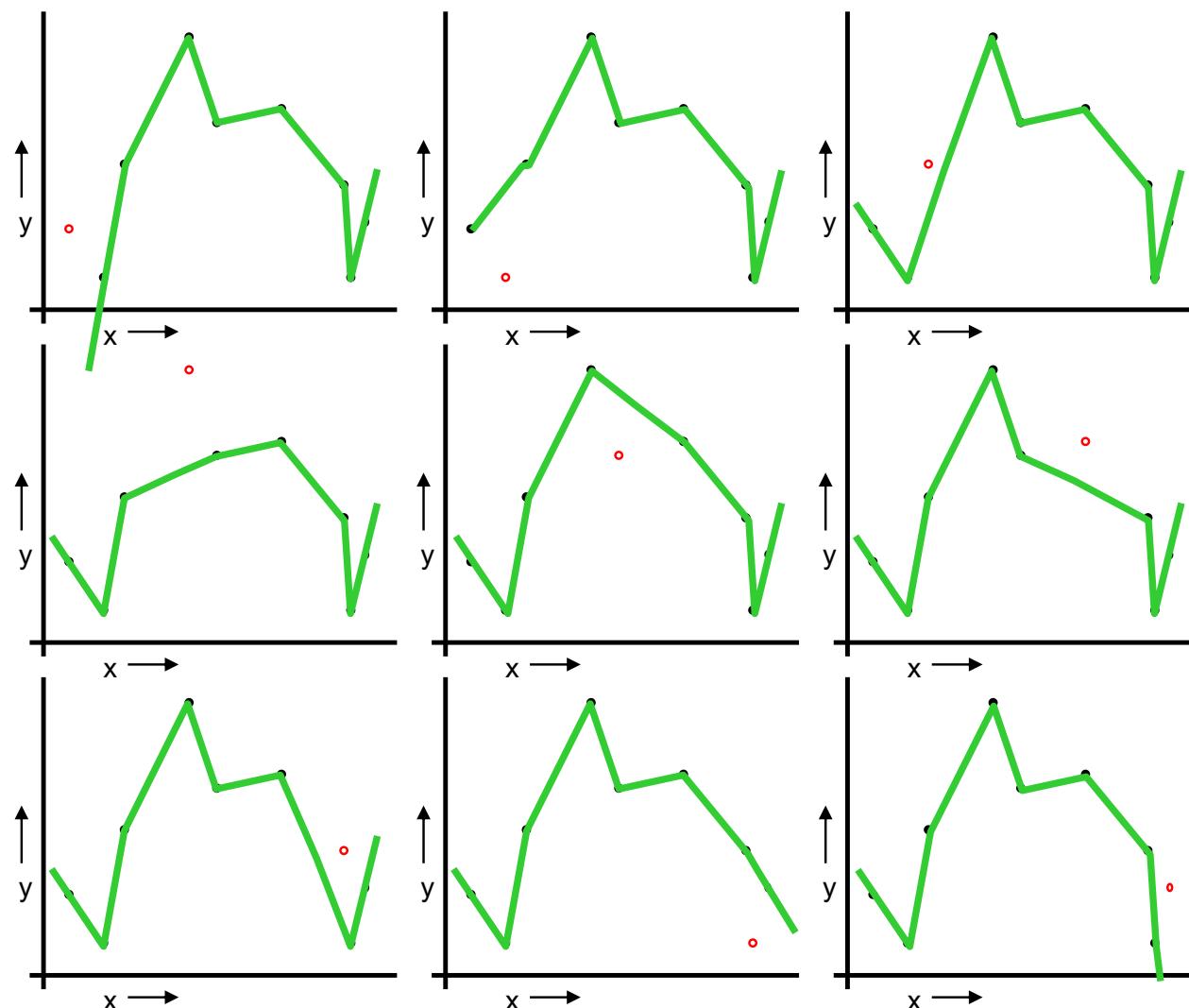
Mean squared error: 2.12
47

Leave-one-out cross-validation (LOOCV)



Mean squared error: 0.962

Leave-one-out cross-validation (LOOCV)



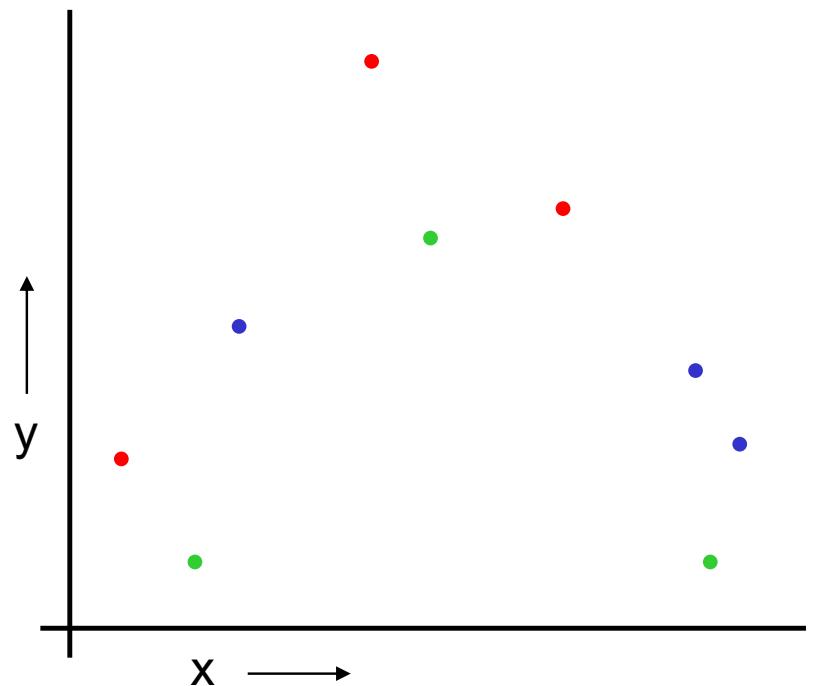
Mean squared error: 3.33
49

Which kind of cross-validation?

	Downside	Upside
Validation-set	Unreliable estimate of future performance	Cheap
Leave-one-out	Expensive	Doesn't waste data

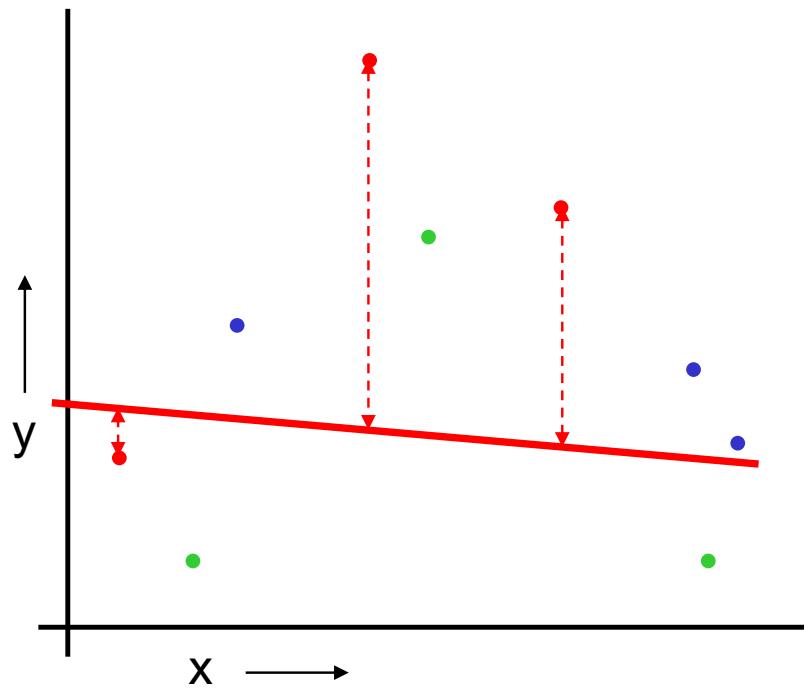
- Can we get the best of both worlds?

k -fold cross-validation



Randomly split the dataset into k partitions (e.g., $k = 3$, shown with different colors)

k -fold cross-validation

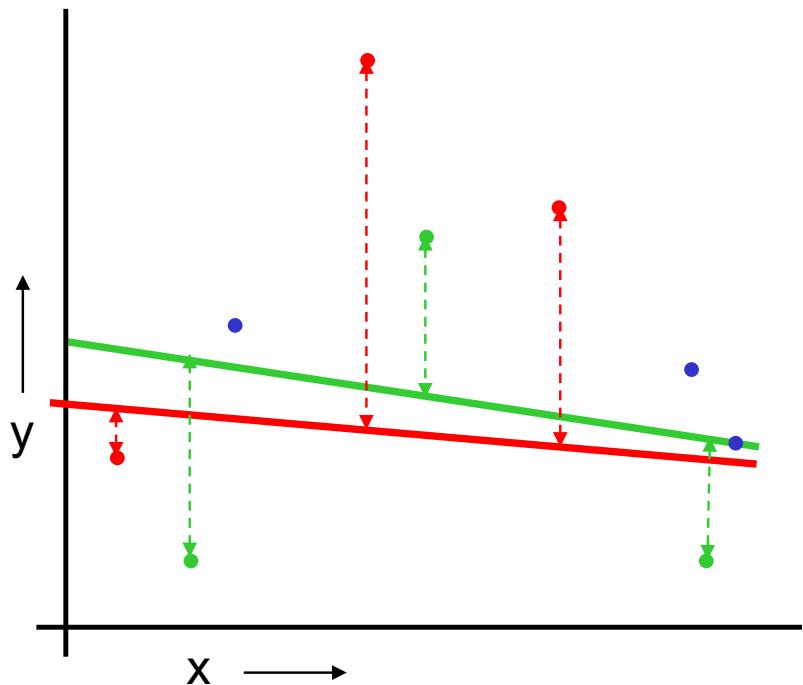


Randomly split the dataset into k partitions (e.g., $k = 3$, shown with different colors)

For the red partition, train on all the points **not** in the red partition. Compute the errors on the red points.

E.g., Fit a line

k -fold cross-validation



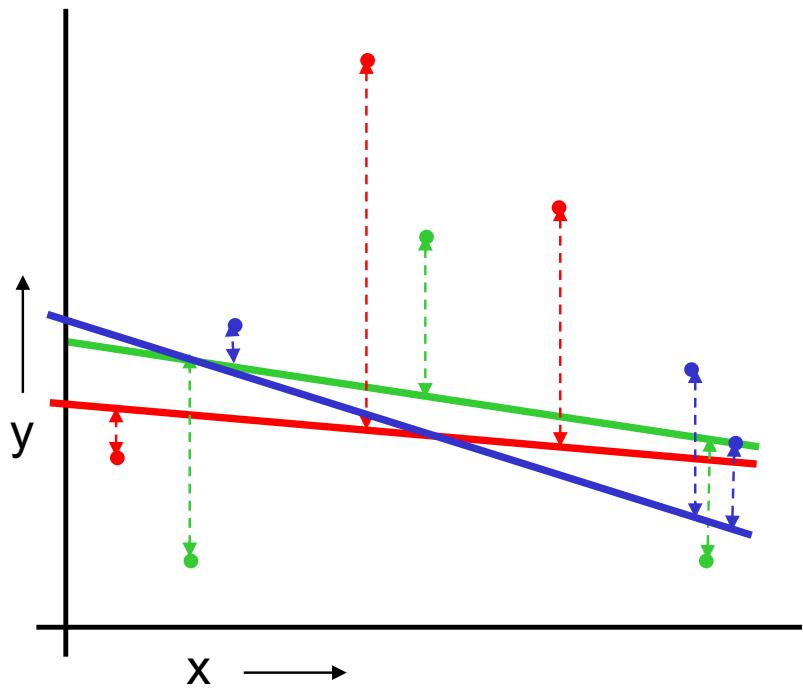
E.g., Fit a line

Randomly split the dataset into k partitions (e.g., $k = 3$, shown with different colors)

For the red partition, train on all the points **not** in the red partition. Compute the errors on the red points.

For the green partition, train on all the points **not** in the green partition. Compute the errors on the green points.

k -fold cross-validation



E.g., Fit a line

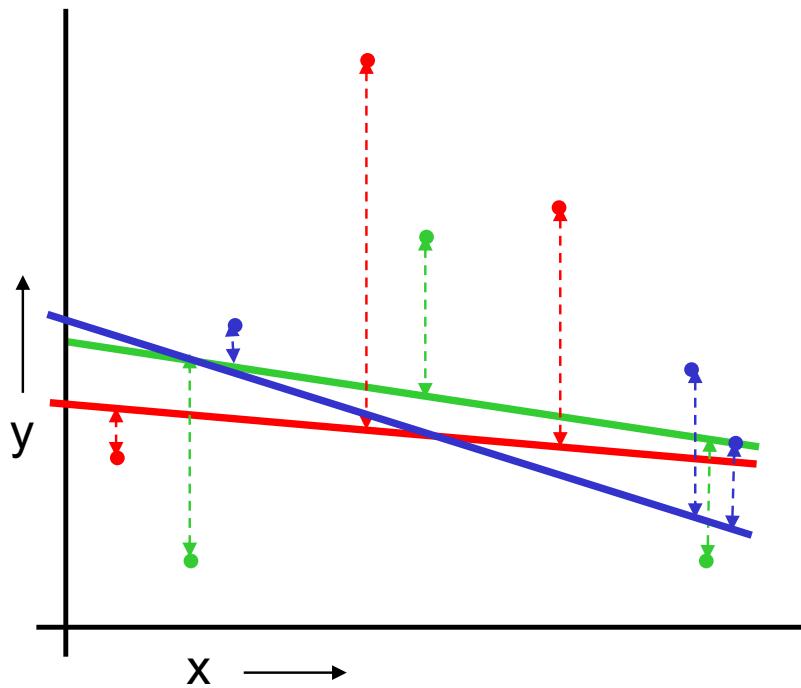
Randomly split the dataset into k partitions (e.g., $k = 3$, shown with different colors)

For the red partition, train on all the points **not** in the red partition. Compute the errors on the red points.

For the green partition, train on all the points **not** in the green partition. Compute the errors on the green points.

For the blue partition, train on all the points **not** in the blue partition. Compute the errors on the blue points.

k -fold cross-validation



E.g., Fit a line

Mean squared error: 2.05

Randomly split the dataset into k partitions (e.g., $k = 3$, shown with different colors)

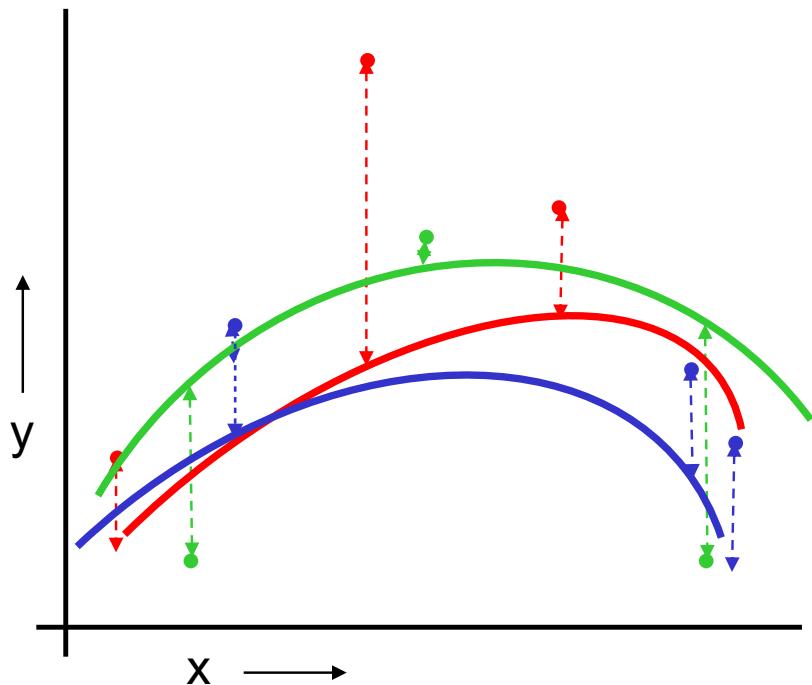
For the red partition, train on all the points **not** in the red partition. Compute the errors on the red points.

For the green partition, train on all the points **not** in the green partition. Compute the errors on the green points.

For the blue partition, train on all the points **not** in the blue partition. Compute the errors on the blue points.

Report the average error on all points

k -fold cross-validation



E.g., Fit a quadratic function

Mean squared error: 1.11

Randomly split the dataset into k partitions (e.g., $k = 3$, shown with different colors)

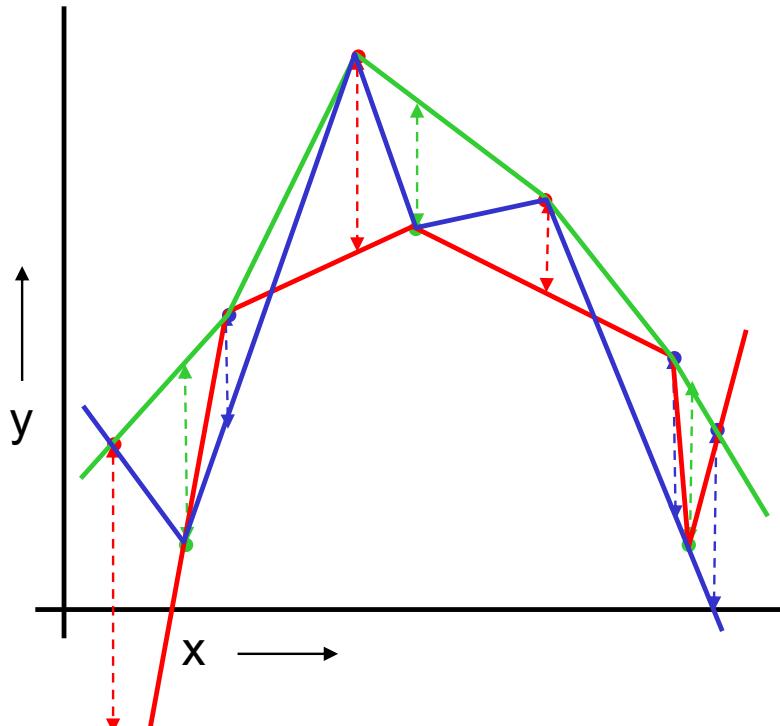
For the red partition, train on all the points **not** in the red partition. Compute the errors on the red points.

For the green partition, train on all the points **not** in the green partition. Compute the errors on the green points.

For the blue partition, train on all the points **not** in the blue partition. Compute the errors on the blue points.

Report the average error on all points

k -fold cross-validation



Mean squared error: 2.93

Randomly split the dataset into k partitions (e.g., $k = 3$, shown with different colors)

For the red partition, train on all the points **not** in the red partition. Compute the errors on the red points.

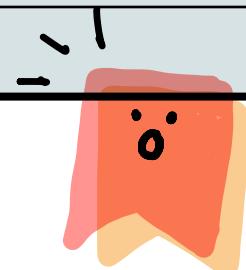
For the green partition, train on all the points **not** in the green partition. Compute the errors on the green points.

For the blue partition, train on all the points **not** in the blue partition. Compute the errors on the blue points.

Report the average error on all points

Which kind of cross-validation?

	Downside	Upside
Validation-set	Unreliable estimate of future performance	Cheap
Leave-one-out	Expensive	Doesn't waste data
10-fold	Wastes 10% of the data. 10 times more expensive than validation-set	Only wastes 10%. Only 10 times more expensive instead of N times
3-fold	Wastier than 10-fold. Expensivier than validation-set	Slightly better than validation-set
N-fold	Identical to Leave-one-out	



Cross-validation

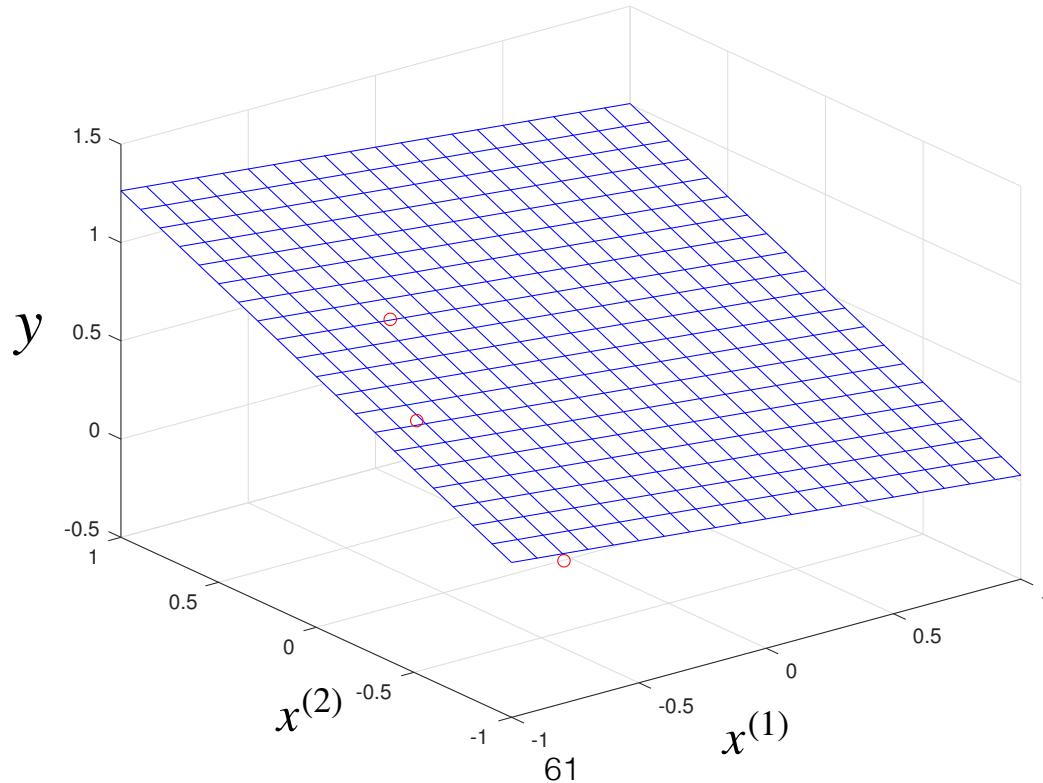
- Cross-validation can help to
 - Select the best model among a small number of candidates
 - Find the best value for one hyper-parameter (e.g., k in kNN)
 - For this, one treats each value of the hyper-parameter as a separate model
- However, it becomes impractical with too many candidates
 - For example, one cannot choose which combination of functions to use for feature expansion among a very large set
- Let us now look at another way to prevent overfitting by penalizing model complexity via regularization
 - To this end, we will go back to the linear model (with expanded features)

Overfitting with a linear model

- Because the linear model is simple, overfitting occurs fairly rarely
- Nevertheless, if there are fewer training samples than input dimensions ($N \leq D + 1$), then one can always fit a hyperplane that passes exactly through the training samples
 - This may seem a rare situation, but it can occur more easily when using feature expansion

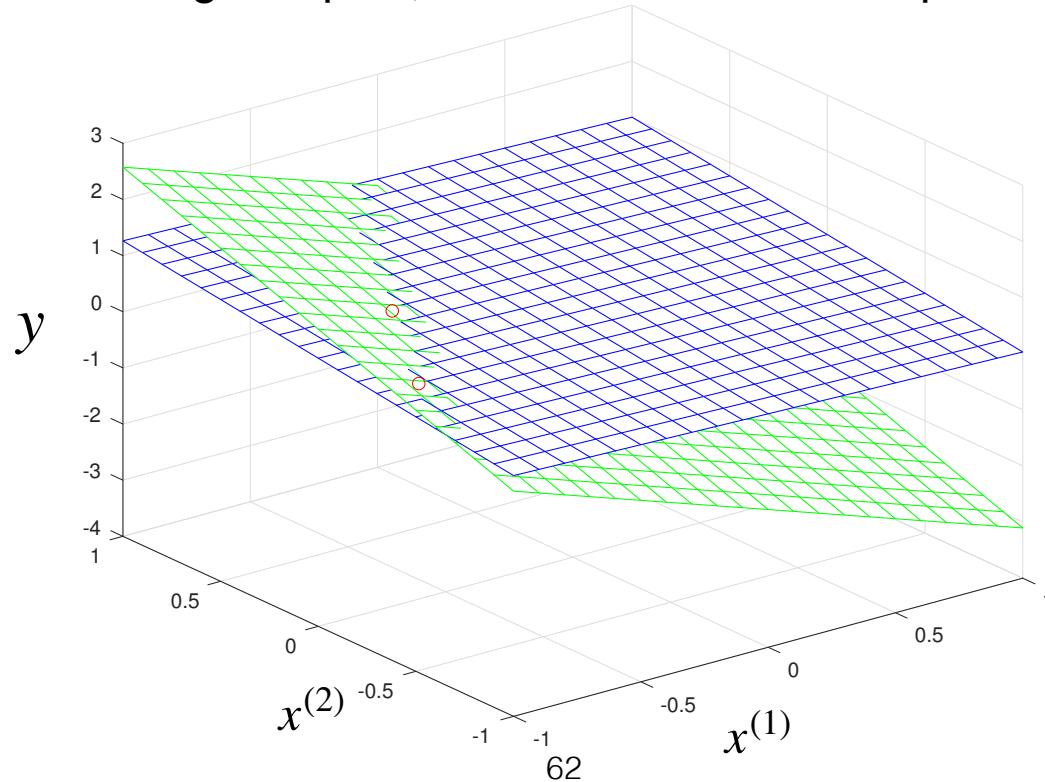
Overfitting with a linear model: Example

- Example: $N = 3, D = 2$
 - 3 noisy samples (red circles) originating from the true blue plane. They are not on the plane because of noise



Overfitting with a linear model: Example

- Example: $N = 3, D = 2$
 - 3 noisy samples (red circles) originating from the true blue plane. They are not on the plane because of noise
 - Fitting a plane to these samples yield the green plane. It passes exactly through the training samples, but is far from the true plane



Penalizing overfitting

- The standard way to penalize the complexity of a model is to add a regularizer to the empirical risk
- This leads to a new training objective function of the form

$$E(\mathbf{w}) = R(\mathbf{w}) + \lambda E_w(\mathbf{w})$$

where $R(\mathbf{w})$ is the empirical risk

$E_w(\mathbf{w})$ is the regularizer

λ is a hyper-parameter that defines the influence of the regularizer ($\lambda > 0$)

Regularized linear regression

- One of the most common regularizer is the sum of squares of the weights

$$E_w(\mathbf{w}) = \mathbf{w}^T \mathbf{w}$$

- This lets us write regularized linear regression as

$$\min_{\mathbf{w}} \sum_{i=1}^N (\phi(\mathbf{x}_i)^T \mathbf{w} - y_i)^2 + \lambda \mathbf{w}^T \mathbf{w}$$

- This remains a convex function (for $\lambda \geq 0$) and so we can still find the solution by zeroing out the gradient

Regularized linear regression: Gradient

- We have

$$\nabla E = \nabla R + \nabla E_w$$
$$= 2 \sum_{i=1}^N \phi(\mathbf{x}_i)(\phi(\mathbf{x}_i)^T \mathbf{w} - y_i) + 2\lambda \mathbf{w}$$

Same as what we saw in earlier lectures

- Setting the gradient to zero yields

$$\left(\sum_{i=1}^N \phi(\mathbf{x}_i)\phi(\mathbf{x}_i)^T + \lambda \mathbf{I}_F \right) \mathbf{w}^* = \sum_{i=1}^N \phi(\mathbf{x}_i)y_i$$

where \mathbf{I}_F is the $F \times F$ identity matrix (1 on the diagonal and 0 elsewhere)

Regularized linear regression: Solution

- Grouping the inputs in a matrix Φ and the outputs in a vector \mathbf{y} , as before, gives the final solution

$$\mathbf{w}^* = (\Phi^T \Phi + \lambda \mathbf{I}_F)^{-1} \Phi^T \mathbf{y}$$

- Note that now we cannot use the pseudo-inverse anymore
- Regularized linear regression is also referred to as *ridge regression*
 - Note that the regularizer affects the optimal value of the parameters, but not the mathematical form of the prediction, which remains

$$\hat{y} = (\mathbf{w}^*)^T \phi(\mathbf{x})$$

Regularized linear regression: Demo

- <https://playground.tensorflow.org/#activation=linear&batchSize=10&dataset=circle®Dataset=reg-plane&learningRate=0.03®ularizationRate=0&noise=0&networkShape=&seed=0.45772&showTestData=false&discretize=false&percTrainData=50&x=true&y=true&xTimesY=false&xSquared=false&ySquared=false&cosX=false&sinX=false&cosY=false&sinY=false&collectStats=false&problem=regression&initZero=false&hideText=false>

Multi-output ridge regression

- When dealing with multiple output dimensions, the parameters are expressed in matrix form
- We can then write regularized multi-output linear regression as

$$\min_{\mathbf{W}} \sum_{i=1}^N \|\mathbf{W}^T \phi(\mathbf{x}_i) - \mathbf{y}_i\|^2 + \lambda \|\mathbf{W}\|_F^2$$

where $\|\mathbf{W}\|_F^2$ is the square Frobenius norm, equivalent to the sum of the square of each value in the matrix \mathbf{W}

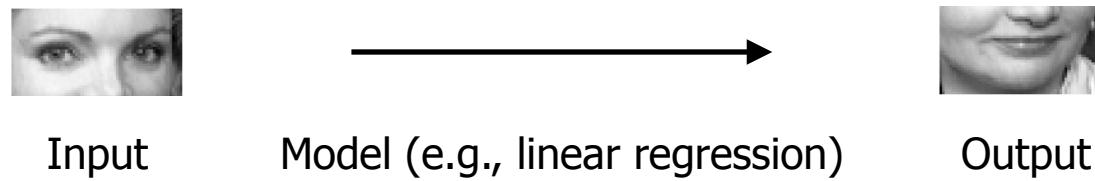
- By zeroing out the gradient, we obtain the solution

$$\mathbf{W}^* = (\Phi^T \Phi + \lambda \mathbf{I}_F)^{-1} \Phi^T \mathbf{Y}$$

where \mathbf{Y} is an $N \times C$ matrix (with C output dimensions)

Multi-output ridge regression: Example

- Face completion:
 - Example from https://scikit-learn.org/stable/auto_examples/plot_multioutput_face_completion.html
 - Task: Given the top half image of a face, predict the bottom half



- Dataset: Olivetti faces
 - 40 subjects (35 for training, 5 for testing)
 - 10 images per subject

Multi-output ridge regression: Example

- Face completion:

- Results:

Linear regression



K-nn



Ridge



true faces



Regularization and logistic regression

- Regularization may also be added to the cross-entropy loss used to train the logistic regression model
 - With a convex regularizer, such as the sum of squares, the overall regularized risk remains convex
 - The scikit-learn implementation of logistic regression has such an L^2 regularizer by default
- In fact, the following demo allows one to add a regularizer
 - <https://playground.tensorflow.org/#activation=sigmoid&batchSize=10&dataset=gauss®Dataset=reg-plane&learningRate=0.03®ularizationRate=0&noise=0&networkShape=&seed=0.92907&showTestData=false&discretize=false&percTrainData=50&x=true&y=true&xTimesY=false&xSquared=false&ySquared=false&cosX=false&sinX=false&cosY=false&sinY=false&collectStats=false&problem=classification&initZero=false&hideText=false>

Regularization and SVM

- How about incorporating regularization into SVM?
- Recall that the SVM problem is expressed as

$$\begin{aligned} & \min_{\tilde{\mathbf{w}}, w^{(0)}, \{\xi_i\}} \frac{1}{2} \|\tilde{\mathbf{w}}\|^2 + C \sum_{i=1}^N \xi_i \\ & \text{subject to } y_i \cdot (\tilde{\mathbf{w}}^T \mathbf{x}_i + w^{(0)}) \geq 1 - \xi_i, \quad \forall i \end{aligned}$$

$$\xi_i \geq 0, \quad \forall i$$

In a sense, the regularizer is already incorporated into SVM, via the margin maximization term. Note that this does not involve $w^{(0)}$. In practice, $w^{(0)}$ may also be removed from the regularization in least-square or logistic regression

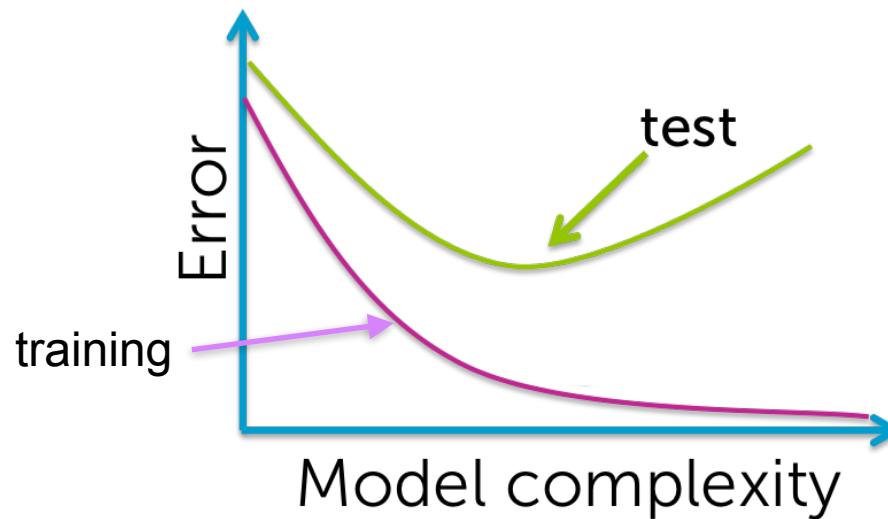
Finding the right regularization strength

- Note that the use of a regularizer leads to another hyper-parameter to the model, i.e., λ
 - Question: Can you optimize it in the same way as \mathbf{w} ?
 - Question: If not, how would you set its value?
- Note that other regularizers than the sum of squares can be used
 - Another popular one is the L^1 norm of \mathbf{w} , which encourages sparsity
 - This, however, goes beyond the scope of this course

Lecture 8: Kernel Methods

Recap: Overfitting vs Underfitting

- General phenomenon

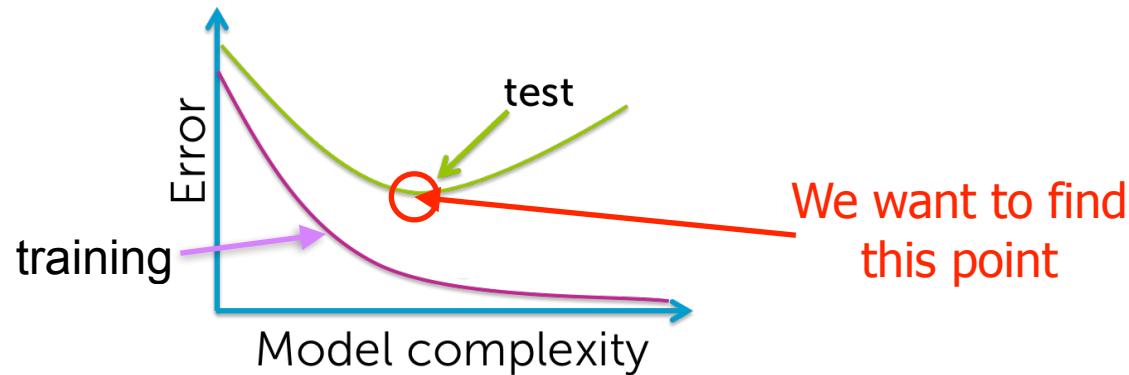


- Model complexity, e.g.
 - A high degree polynomial is more complex than a low degree one
 - In k -NN, $k = 1$ is more complex (the boundary is less smooth) than $k = 21$

Recap: Model selection

- The question then is:

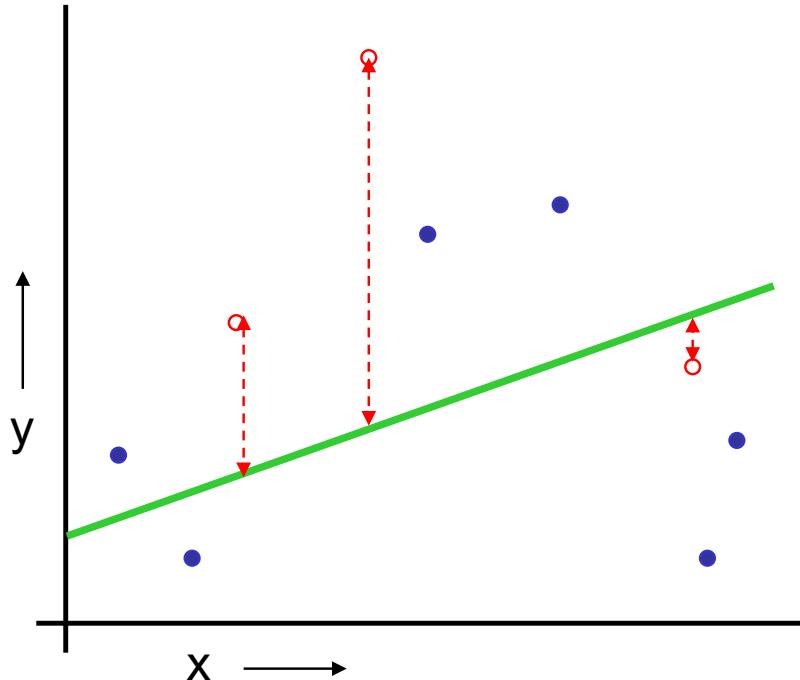
Since we don't have access to the test data, how do we choose the right complexity for the model?



- Solution: Cross-validation

- The following material was adapted from Andrew Moore's cross-validation slides

Recap: The validation set method



E.g., Fit a line

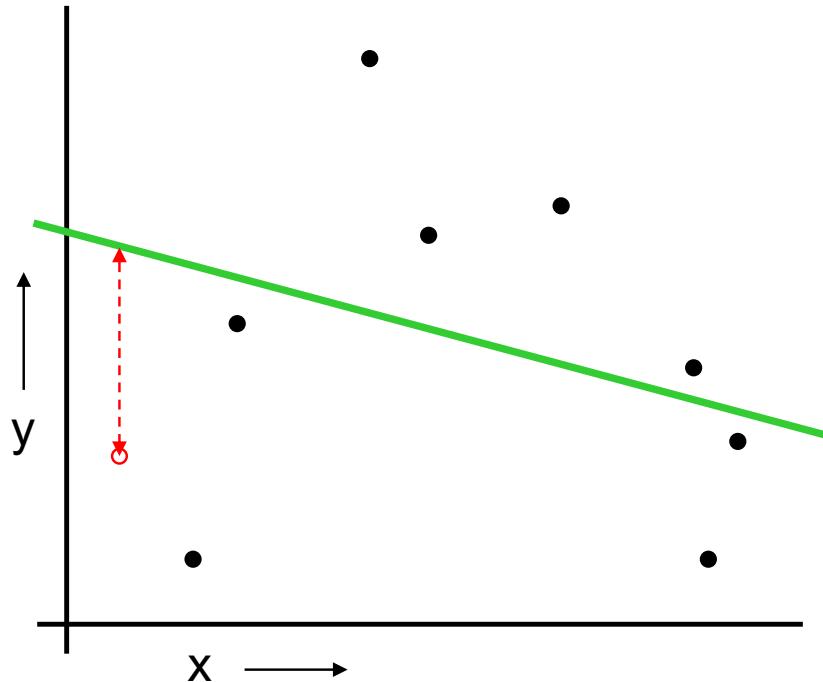
Mean squared error: 2.4

1. Randomly choose, e.g., 30% of the data to form a validation set
2. The remainder then acts as your new training set
3. Perform regression on this new training set only
4. Estimate the performance on the test data using the validation set

Recap: The validation set method

- Advantages:
 - Very simple
 - We can choose the model based on the validation error
- Drawbacks:
 - Wastes data: the best model estimate was obtained with 30% less data
 - If we don't have much data, the validation set might be lucky or unlucky

Recap: LOOCV



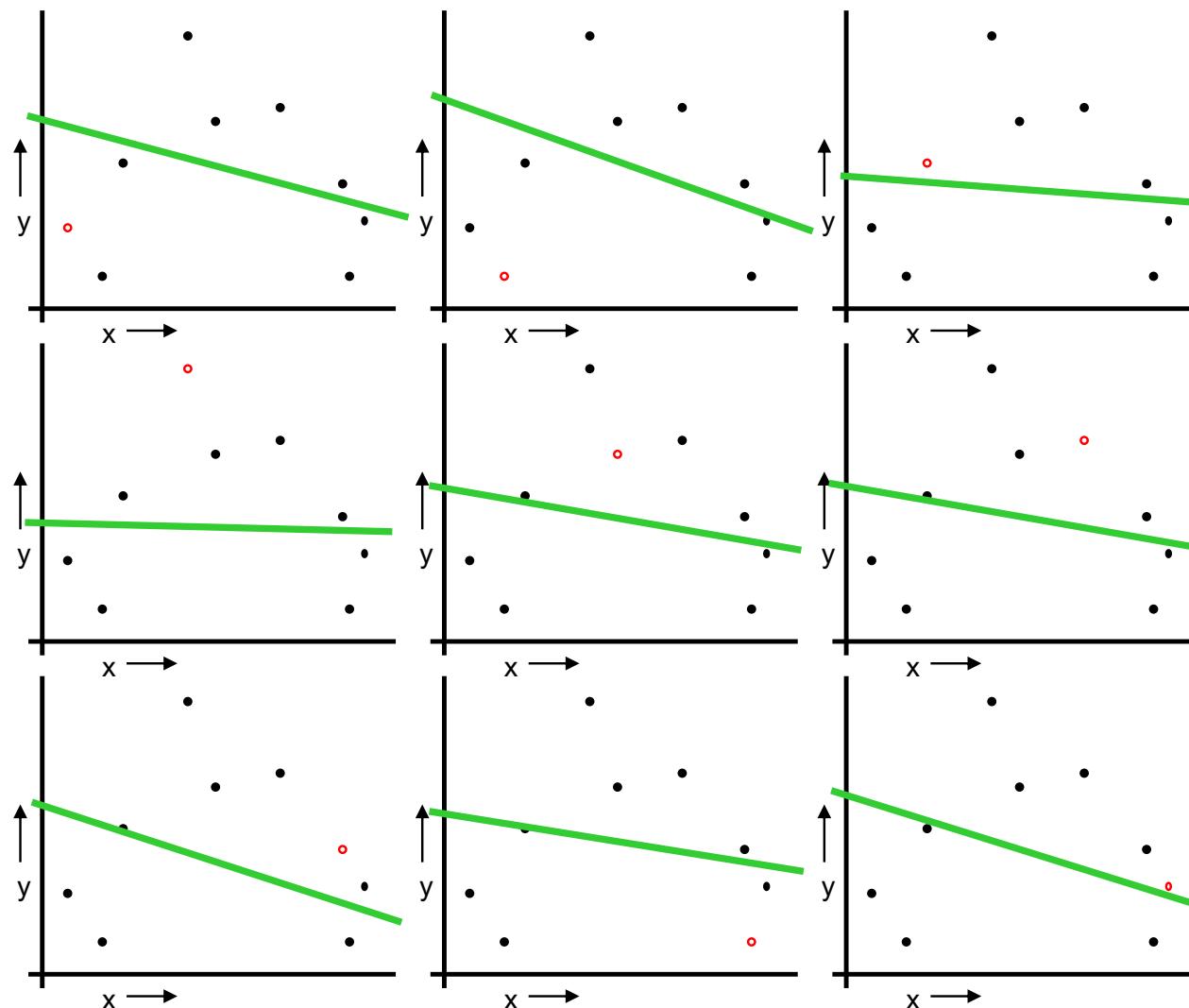
E.g., Fit a line

For $i = 1, \dots, N$

1. Let (x_i, y_i) be the i^{th} sample
2. Temporarily remove (x_i, y_i) from the training set
3. Perform regression on the remaining $N - 1$ samples
4. Compute the error on (x_i, y_i)

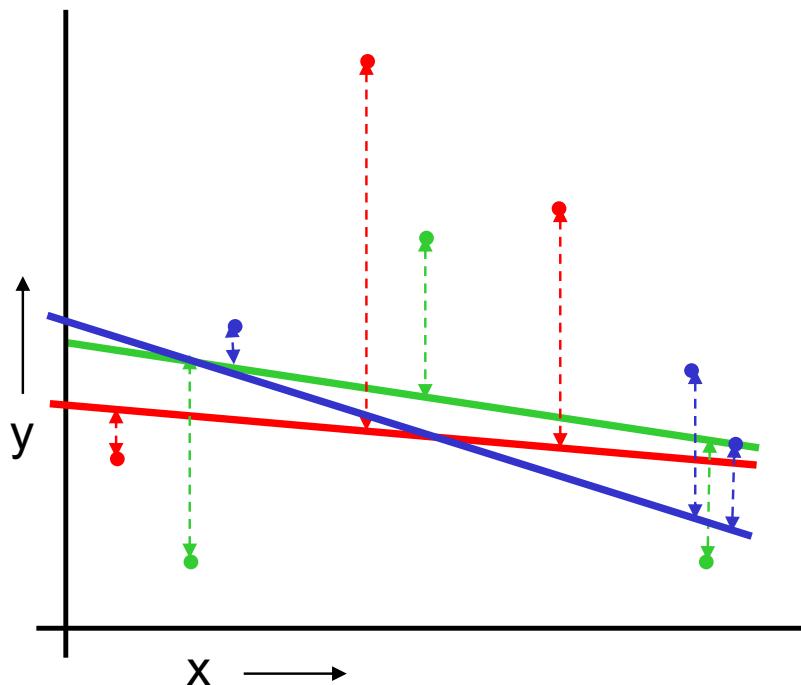
When you have done all points, report the average error

Recap: LOOCV



Mean squared error: 2.12

Recap: k -fold cross-validation



E.g., Fit a line

Mean squared error: 2.05

Randomly split the dataset into k partitions (e.g., $k = 3$, shown with different colors)

For the red partition, train on all the points **not** in the red partition. Compute the errors on the red points.

For the green partition, train on all the points **not** in the green partition. Compute the errors on the green points.

For the blue partition, train on all the points **not** in the blue partition. Compute the errors on the blue points.

Report the average error on all points

Recap: Which kind of cross-validation?

	Downside	Upside
Validation-set	Unreliable estimate of future performance	Cheap
Leave-one-out	Expensive	Doesn't waste data
10-fold	Wastes 10% of the data. 10 times more expensive than validation-set	Only wastes 10%. Only 10 times more expensive instead of N times
3-fold	Wastier than 10-fold. Expensivier than validation-set	Slightly better than validation-set
N-fold	Identical to Leave-one-out	

Recap: Overfitting with a linear model

- Because the linear model is simple, overfitting occurs fairly rarely
- Nevertheless, if there are fewer training samples than input dimensions ($N \leq D + 1$), then one can always fit a hyperplane that passes exactly through the training samples
 - This may seem a rare situation, but it can occur more easily when using feature expansion

Recap: Penalizing overfitting

- The standard way to penalize the complexity of a model is to add a regularizer to the empirical risk
- This leads to a new training objective function of the form

$$E(\mathbf{w}) = R(\mathbf{w}) + \lambda E_w(\mathbf{w})$$

where $R(\mathbf{w})$ is the empirical risk

$E_w(\mathbf{w})$ is the regularizer

λ is a hyper-parameter that defines the influence of the regularizer ($\lambda > 0$)

Recap: Regularized linear regression

- One of the most common regularizer is the sum of squares of the weights

$$E_w(\mathbf{w}) = \mathbf{w}^T \mathbf{w}$$

- This lets us write regularized linear regression as

$$\min_{\mathbf{w}} \sum_{i=1}^N (\phi(\mathbf{x}_i)^T \mathbf{w} - y_i)^2 + \lambda \mathbf{w}^T \mathbf{w}$$

- This remains a convex function (for $\lambda \geq 0$) and so we can still find the solution by zeroing out the gradient

Recap: Finding the right regularization strength

- Note that the use of a regularizer leads to another hyper-parameter to the model, i.e., λ
 - This parameter cannot be optimized with \mathbf{w} , because, for the training data, the best solution would always be $\lambda = 0$
 - Instead, it can be set via cross-validation, e.g., train with different values of $\lambda \in \{1e-3, 1e-2, 1e-1, 1, 10, 100, 1000\}$ and choose the best model based on the performance on validation data
- Note that other regularizers than the sum of squares can be used
 - Another popular one is the L^1 norm of \mathbf{w} , which encourages sparsity
 - This, however, goes beyond the scope of this course

Goals of today's lecture

- Introduce the notion of kernel function and the kernel trick
- Derive kernelized versions of linear (and ridge) regression and of SVM

Back to the linear model

- In dimension D , we can write

$$y = w^{(0)} + w^{(1)}x^{(1)} + w^{(2)}x^{(2)} + \dots + w^{(D)}x^{(D)} = \mathbf{w}^T \begin{bmatrix} x^{(1)} \\ x^{(2)} \\ \vdots \\ x^{(D)} \\ 1 \end{bmatrix}$$

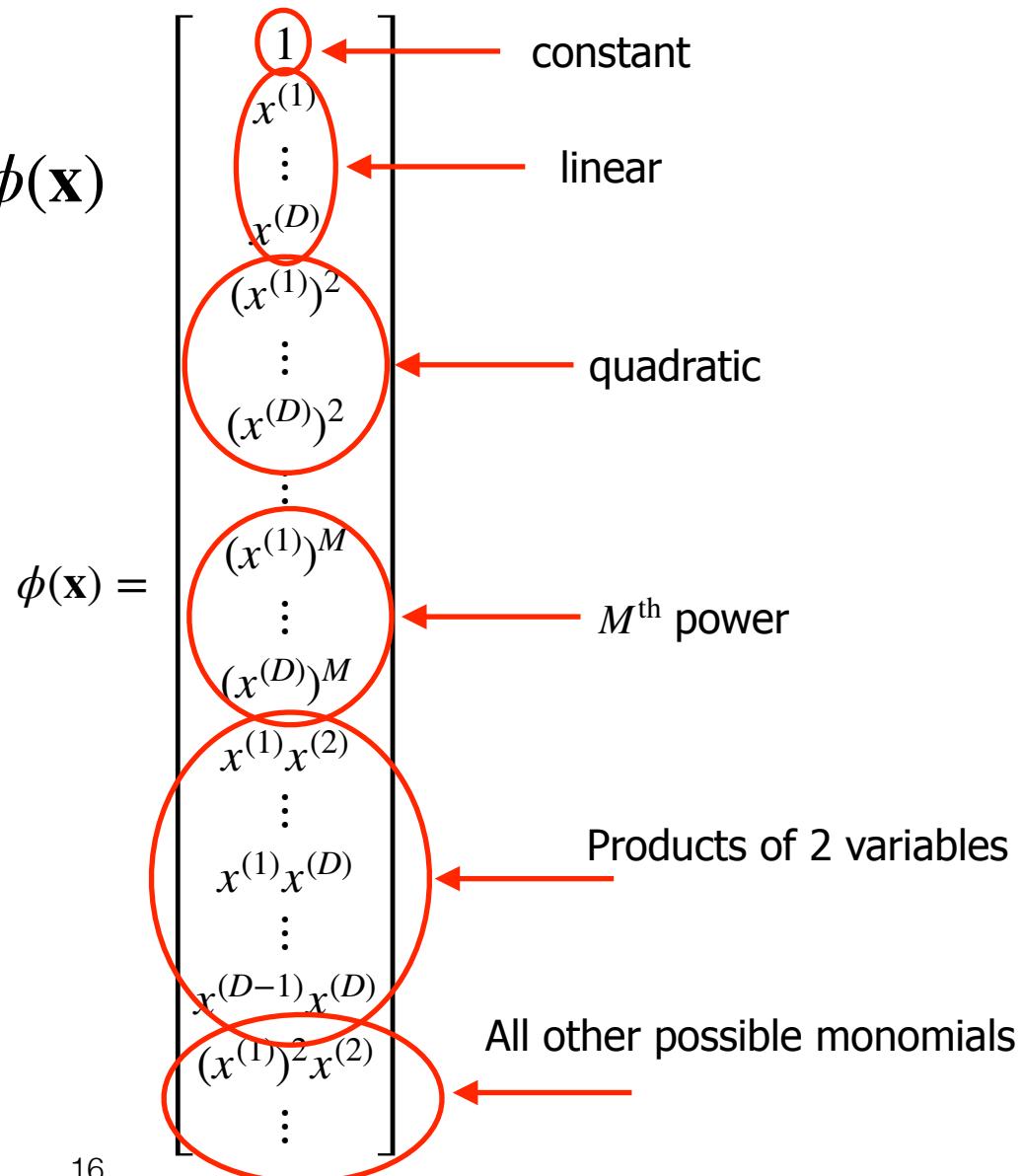
- Ultimately, whatever the dimension, we can write

$$y = \mathbf{w}^T \mathbf{x}$$

with $\mathbf{x} \in \mathbb{R}^{D+1}$, where the extra dimension contains a 1 to account for $w^{(0)}$

Recap: Polynomial feature expansion

- In general, for D -dimensional inputs, we can obtain a large $\phi(\mathbf{x})$
- The resulting expanded features can then be used in any linear algorithm
 - Let us look at how this would work for linear regression



Recap: Working with expanded features

- In essence, polynomial feature expansion uses a mapping from $\mathbf{x} \in \mathbb{R}^D$ to another representation $\phi(\mathbf{x}) \in \mathbb{R}^F$, where $F \gg D$
- We can then use a linear model in this new space and write

$$\hat{y}_i = \mathbf{w}^T \phi(\mathbf{x}_i)$$

where now $\mathbf{w} \in \mathbb{R}^F$

(assuming that the 1 accounting for the bias has been incorporated in $\phi(\mathbf{x}_i)$)

- Given training data, we can then find the optimal parameters via standard linear regression:

$$\min_{\mathbf{w}} \sum_{i=1}^N (\mathbf{w}^T \phi(\mathbf{x}_i) - y_i)^2$$

Recap: Working with expanded features: Solution

- We can group the transformed inputs $\{\phi(\mathbf{x}_i)\}$ and the outputs $\{y_i\}$ in a matrix and vector of the form

$$\Phi = \begin{bmatrix} \phi(\mathbf{x}_1)^T \\ \phi(\mathbf{x}_2)^T \\ \vdots \\ \phi(\mathbf{x}_N)^T \end{bmatrix} \in \mathbb{R}^{N \times F} \quad \mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix} \in \mathbb{R}^N$$

- Then, we have

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

Recap: Feature expansion: Properties

- Why limit ourselves to polynomials?
 - Any function of the input could be used, e.g.,
 - sine and cosine
 - exponential and logarithm
 - ...
- In fact, in the demo...
 - <https://playground.tensorflow.org/#activation=linear&batchSize=10&dataset=gauss®Dataset=reg-plane&learningRate=0.03®ularizationRate=0&noise=0&networkShape=&seed=0.49340&showTestData=false&discretize=false&percTrainData=50&x=true&y=true&xTriesY=false&xSquared=false&ySquared=false&cosX=false&sinX=false&cosY=false&sinY=false&collectStats=false&problem=classification&initZero=false&hideText=false>
- Then, how do we choose which functions to use?
 - Solution: Kernels: Let's circumvent this choice

Kernel function

- The solution to many ML algorithms ends up relying on a dot product between inputs, i.e., $\mathbf{x}_i^T \mathbf{x}_j$
- Because $\mathbf{x}_i^T \mathbf{x}_j \propto \cos\angle(\mathbf{x}_i, \mathbf{x}_j)$, this encodes a notion of similarity between two samples
- When doing feature expansion, we then have dot products of the form $\phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j)$, which also encodes a similarity between the two samples, but different from the original one
- Thus, instead of explicitly defining the mapping $\phi(\cdot)$, one can define a similarity function $k(\mathbf{x}_i, \mathbf{x}_j)$
- This is called a *kernel* function. It corresponds to some mapping $\phi(\cdot)$, i.e., $k(\mathbf{x}_i, \mathbf{x}_j) = \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j)$, but this $\phi(\cdot)$ may be unknown

Kernel: Example

- One example of commonly-used kernel function is the polynomial kernel

$$k(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i^T \mathbf{x}_j + c)^d$$

- It has two hyper-parameters:
 - The bias c (often set to 1)
 - The degree of the polynomial d (often set to 2)
- For such kernels, the corresponding mappings $\phi(\cdot)$ are known

Polynomial kernel vs feature expansion

- Consider the following mapping for a 2D input \mathbf{x}

$$\phi(\mathbf{x}) = \left[(x^{(1)})^2, (x^{(2)})^2, \sqrt{2}x^{(1)}x^{(2)}, \sqrt{2}x^{(1)}, \sqrt{2}x^{(2)}, 1 \right]^T$$

- The dot product between two samples then is

$$\begin{aligned}\phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j) &= (x_i^{(1)})^2(x_j^{(1)})^2 + (x_i^{(2)})^2(x_j^{(2)})^2 + 2x_i^{(1)}x_i^{(2)}x_j^{(1)}x_j^{(2)} + 2x_i^{(1)}x_j^{(1)} + 2x_i^{(2)}x_j^{(2)} + 1 \\ &= \left(x_i^{(1)}x_j^{(1)} + x_i^{(2)}x_j^{(2)} + 1 \right)^2\end{aligned}$$

- This corresponds to the quadratic kernel

$$k(\mathbf{x}_i, \mathbf{x}_j) = \left(\mathbf{x}_i^T \mathbf{x}_j + 1 \right)^2$$

- Note that computing the kernel value only requires 3 multiplications, versus 6 when computing the dot product $\phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j)$

Kernel: Example

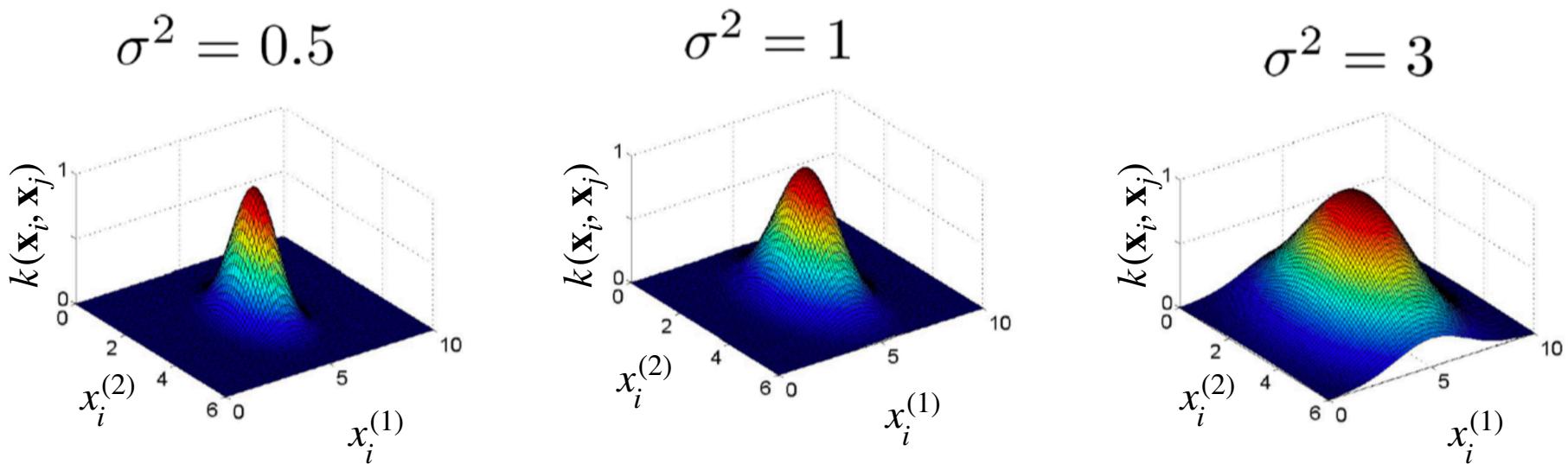
- Possibly the most common kernel is the Gaussian (or Radial Basis Function (RBF)) kernel

$$k(\mathbf{x}_i, \mathbf{x}_j) = \exp\left(-\frac{\|\mathbf{x}_i - \mathbf{x}_j\|^2}{2\sigma^2}\right)$$

- It has one hyper-parameter:
 - The bandwidth σ (which is data-dependent)
- For this kernel, there is no explicit corresponding mapping $\phi(\cdot)$
 - In theory, this kernel corresponds to a mapping to an infinite dimensional space

Gaussian kernel: Visualization

- With 2D inputs, we can visualize the effect of σ
- Let one sample be fixed as $\mathbf{x}_j = [5, 3]^T$, and vary \mathbf{x}_i



Kernel trick

- Since a kernel function corresponds to a similarity, as a dot product, one can replace any dot product between two samples in the solution of an ML algorithm with a kernel function
- This is applicable to many algorithms, and is referred to as *kernelizing* an algorithm
- Let us now look at how this works for linear regression

Kernel regression

- As seen before, the solution to linear regression is given by

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

- At first sight, it may seem that $\Phi^T \Phi$ contains the dot products that we want to replace with a kernel function
 - However, $\Phi^T \Phi$ is an $F \times F$ matrix encoding dot products between the different feature dimensions, not the different samples
 - A kernel matrix, containing the dot products between every pair of training samples would be of size $N \times N$, and correspond to $\Phi \Phi^T$

Representer theorem

- To kernelize linear regression we can make use of the Representer theorem
 - This will be useful to kernelize other algorithms
- In essence, this theorem states that the minimizer of a regularized empirical risk function can be represented as a linear combination of the points in the high dimensional space. That is, we can write

$$\mathbf{w}^* = \sum_{i=1}^N \alpha_i^* \phi(\mathbf{x}_i) = \Phi^T \alpha^*$$

Proven by Schölkopf et al., “A Generalized Representer Theorem”, Computational Learning Theory, 2001

Kernel regression

- This means that we can re-write the empirical risk in terms of the variables $\{\alpha_i\}$
- That is, the empirical risk

$$R(\mathbf{w}) = \sum_{i=1}^N (\mathbf{w}^T \phi(\mathbf{x}_i) - y_i)^2$$

- Can be re-written as

Now, we have the kind of dot products we need

$$\begin{aligned} R(\alpha) &= \sum_{i=1}^N (\alpha^T \Phi \phi(\mathbf{x}_i) - y_i)^2 \\ &= \sum_{i=1}^N (\alpha^T k(\mathbf{X}, \mathbf{x}_i) - y_i)^2 \end{aligned}$$

Vector obtained by evaluating the kernel function between all training samples and sample i

Kernel regression

- The gradient of the risk w.r.t. α is given by

$$\nabla R(\alpha) = 2 \sum_{i=1}^N k(\mathbf{X}, \mathbf{x}_i) (k(\mathbf{X}, \mathbf{x}_i)^T \alpha - y_i)$$

- Following the same reasoning as for linear regression, we can write this in matrix form as

$$\nabla_{\alpha} R(\alpha) = 2\mathbf{K}\mathbf{K}\alpha - 2\mathbf{K}\mathbf{y}$$

- \mathbf{K} is the $N \times N$ training kernel matrix, obtained by evaluating the kernel function between all pairs of training samples

Kernel regression

- Setting the gradient to 0 yields

$$2\mathbf{K}\mathbf{K}\alpha^* = 2\mathbf{K}\mathbf{y}$$

$$\Leftrightarrow \mathbf{K}\alpha^* = \mathbf{y}$$

- which gives the solution

$$\alpha^* = (\mathbf{K})^{-1} \mathbf{y}$$

Kernel regression

- If we put this solution back in the definition of \mathbf{w}^* , we have

$$\begin{aligned}\mathbf{w}^* &= \Phi^T \boldsymbol{\alpha}^* \\ &= \textcircled{\Phi^T} (\mathbf{K})^{-1} \mathbf{y}\end{aligned}$$

- This solution still depends on Φ , which we argued could be unknown
- In practice, this is not a problem, because our goal is to make predictions for new samples
 - We do not really care about having access to \mathbf{w}^*

Kernel regression: Prediction

- For a new sample \mathbf{x} , the prediction is given by

$$\begin{aligned}\hat{y} &= (\mathbf{w}^*)^T \phi(\mathbf{x}) = \mathbf{y}^T (\mathbf{K})^{-1} \Phi \phi(\mathbf{x}) \\ &= \mathbf{y}^T (\mathbf{K})^{-1} k(\mathbf{X}, \mathbf{x})\end{aligned}$$

where $k(\mathbf{X}, \mathbf{x})$ is the N -dimensional vector obtained by evaluating the kernel function between the training samples and the test one

- Note that the quantity $\mathbf{y}^T (\mathbf{K})^{-1}$ only depends on the training data, and thus can be pre-computed (i.e., does not need to be re-computed for every test sample)

Recap: Ridge regression

- To penalize overfitting, one can use a regularizer such as the sum of squares of the weights

$$E_w(\mathbf{w}) = \mathbf{w}^T \mathbf{w}$$

- Adding this to the linear regression empirical risk yields the ridge regression problem

$$\min_{\mathbf{w}} \sum_{i=1}^N (\phi(\mathbf{x}_i)^T \mathbf{w} - y_i)^2 + \lambda \mathbf{w}^T \mathbf{w}$$

- Such a regularizer can also be kernelized

Kernel ridge regression

- That is, with the Representer theorem, the regularized empirical risk

$$E(\mathbf{w}) = \sum_{i=1}^N (\mathbf{w}^T \phi(\mathbf{x}_i) - y_i)^2 + \lambda \|\mathbf{w}\|^2$$

As the loss function, the regularizer has the dot products we need

- Can be re-written as

$$\begin{aligned} E(\alpha) &= \sum_{i=1}^N (\alpha^T \Phi \phi(\mathbf{x}_i) - y_i)^2 + \lambda \alpha^T \Phi \Phi^T \alpha \\ &= \sum_{i=1}^N (\alpha^T k(\mathbf{X}, \mathbf{x}_i) - y_i)^2 + \lambda \alpha^T \mathbf{K} \alpha \end{aligned}$$

Kernel ridge regression

- Following the same reasoning as without regularizer, we can write the gradient of the risk in matrix form as

$$\nabla_{\alpha} E(\alpha) = 2\mathbf{K} (\mathbf{K} + \lambda \mathbf{I}_N)^{-1} \alpha - 2\mathbf{K}\mathbf{y}$$

- Setting it to 0 yields the solution

$$\alpha^* = (\mathbf{K} + \lambda \mathbf{I}_N)^{-1} \mathbf{y}$$

- Given α^* , one can then express \mathbf{w}^* mathematically as in the case without regularization, and eventually use this formulation to make predictions for the test samples

Gaussian process regression

- Roughly speaking, GP regression is a probabilistic version of kernel ridge regression
 - The mean prediction of a GP corresponds to the KRR prediction
- Demo: <http://www.tmpl.fi/gp/>

Exercise

- As we just saw in the previous demo, the results of kernel ridge regression strongly depend on the choice of kernel and of the hyperparameter values for a given kernel. How can you identify a good kernel and good hyperparameter values?

Recap: Multi-output linear regression

- To predict multiple values, we use a parameter matrix $\mathbf{W} \in \mathbb{R}^{(D+1) \times C}$, such that

$$\hat{\mathbf{y}}_i = \mathbf{W}^T \mathbf{x}_i = \begin{bmatrix} \mathbf{w}_{(1)}^T \\ \mathbf{w}_{(2)}^T \\ \vdots \\ \mathbf{w}_{(C)}^T \end{bmatrix} \mathbf{x}_i$$

where each $\mathbf{w}_{(j)}$ is a $(D + 1)$ -dimensional vector,
used to predict one output dimension

Kernel regression with multiple outputs

- To handle nonlinear problems, we can then again work in a different feature space $\phi(\cdot)$ and write the empirical risk

$$R(\mathbf{W}) = \sum_{i=1}^N \|\mathbf{W}^T \phi(\mathbf{x}_i) - \mathbf{y}_i\|^2$$

- By making use of the Representer theorem, we can again derive a solution that does not explicitly depend on $\phi(\cdot)$ but rather on kernel values $k(\mathbf{x}_i, \mathbf{x}_j)$

Kernel regression with multiple outputs

- Ultimately, the solution is given by

$$\mathbf{W}^* = \Phi^T (\mathbf{K})^{-1} \mathbf{Y}$$

where Φ is the same matrix concatenating the training inputs $\{\phi(\mathbf{x}_i)\}$ as in the 1D-output case, and $\mathbf{Y} \in \mathbb{R}^{N \times C}$ is the matrix stacking the training output vectors

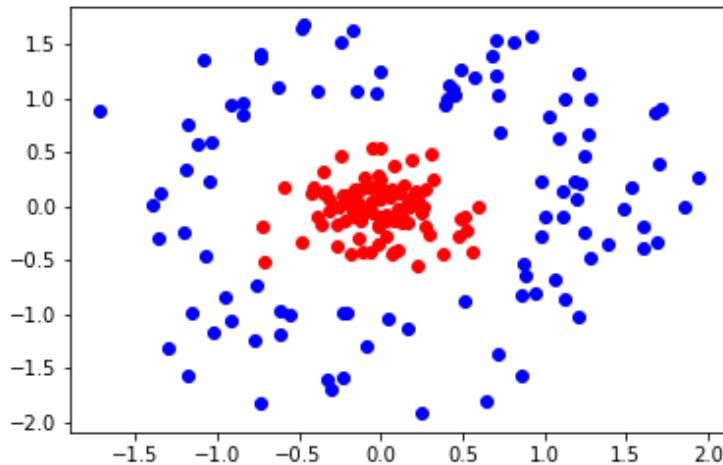
- The prediction vector for a test sample is then given as

$$\begin{aligned}\hat{\mathbf{y}} &= (\mathbf{W}^*)^T \phi(\mathbf{x}) = \mathbf{Y}^T (\mathbf{K})^{-1} \Phi \phi(\mathbf{x}) \\ &= \mathbf{Y}^T (\mathbf{K})^{-1} k(\mathbf{X}, \mathbf{x})\end{aligned}$$

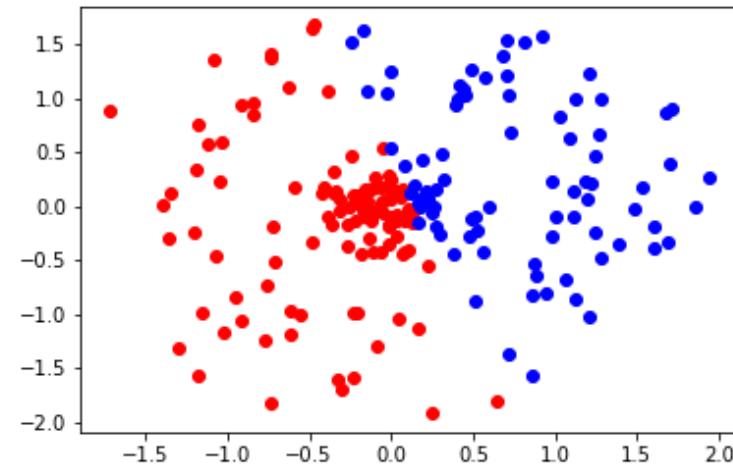
Kernel regression: Example 1

- 2D inputs from 2 classes (colors)

True labels



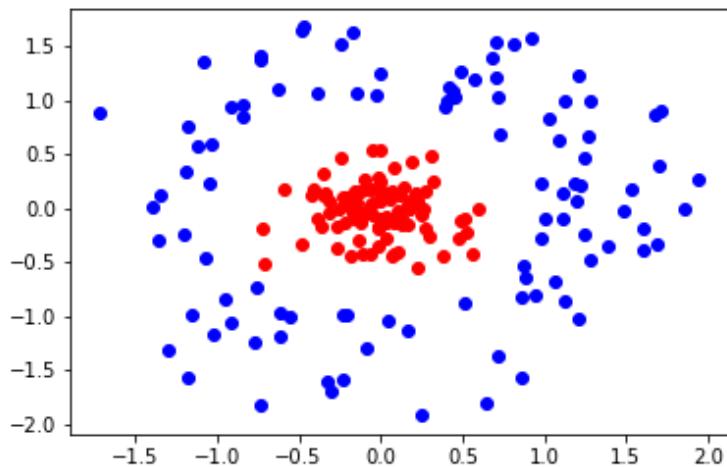
Fitting a linear model



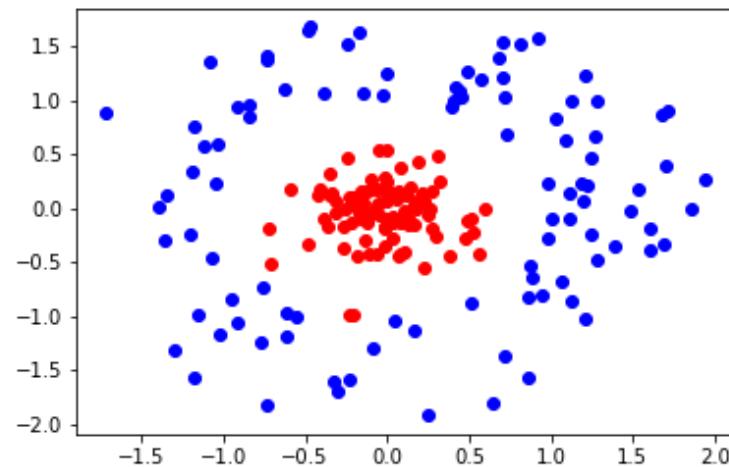
Kernel regression: Example 1

- 2D inputs from 2 classes (colors)

True labels



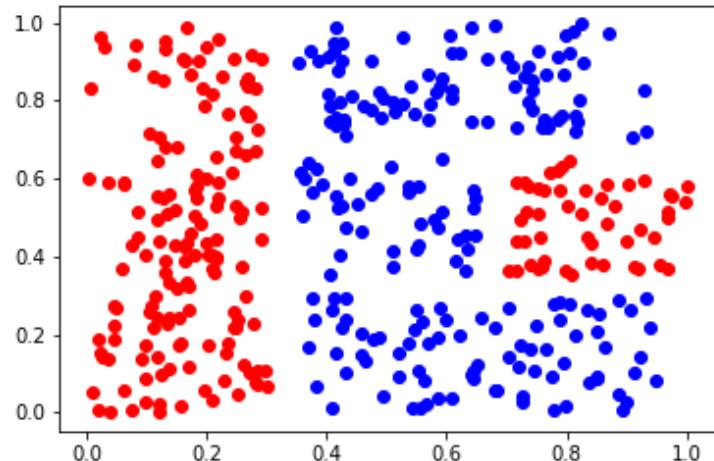
Fitting a nonlinear model
(RBF kernel)



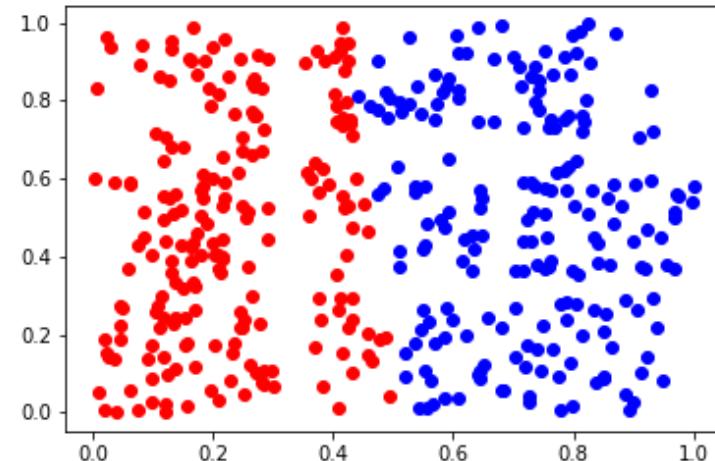
Kernel regression: Example 2

- 2D inputs from 2 classes (colors)

True labels



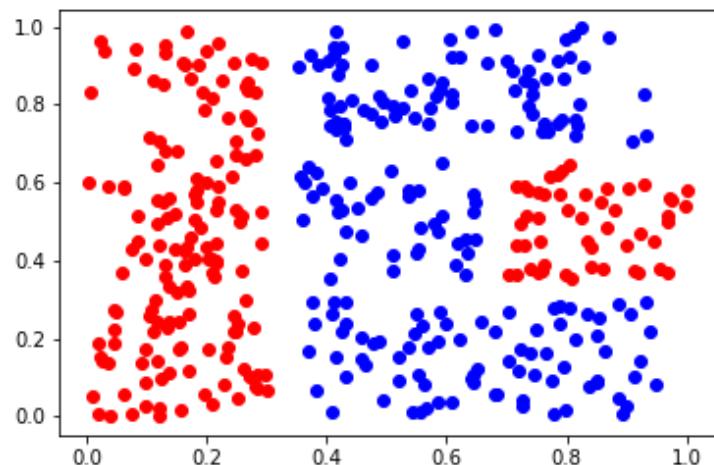
Fitting a linear model



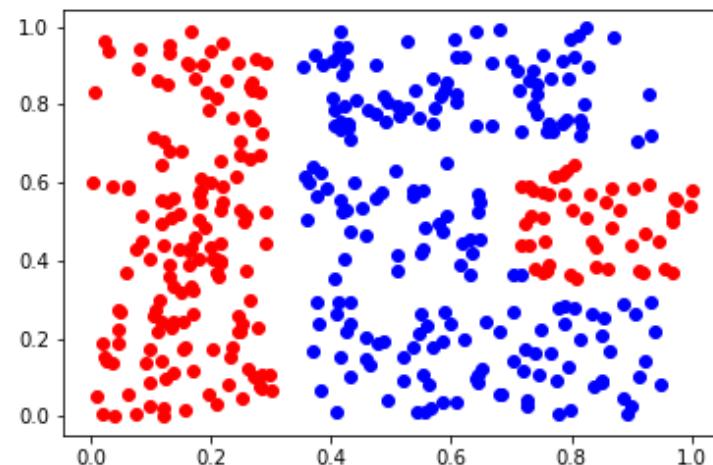
Kernel regression: Example 2

- 2D inputs from 2 classes (colors)

True labels



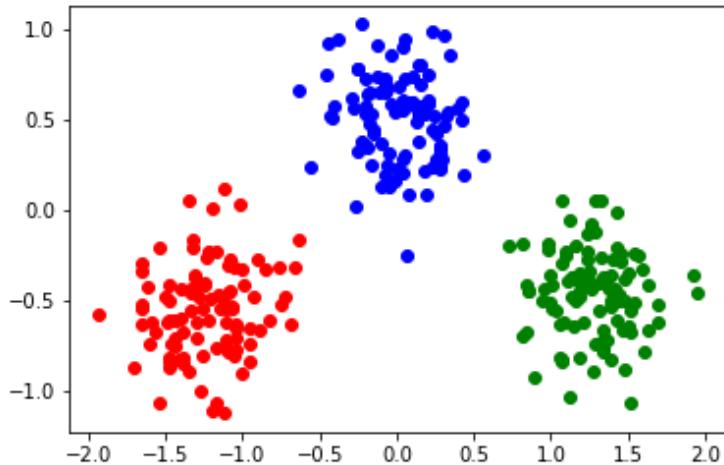
Fitting a nonlinear model
(RBF kernel)



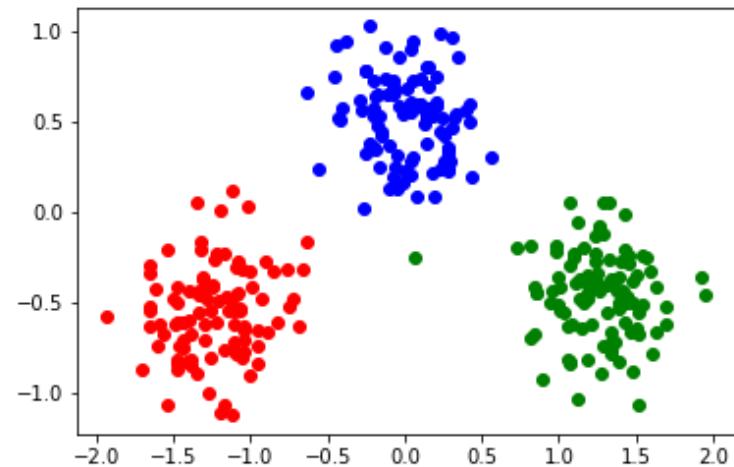
Kernel regression: Example 3

- Kernel regression also applies to linearly separable data
 - 2D inputs from 3 classes (colors)
 - One can use a linear kernel: $k(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i^T \mathbf{x}_j$

True labels



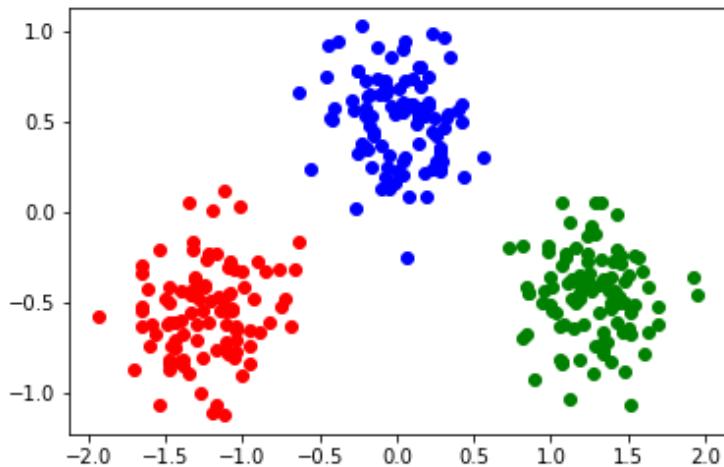
Fitted linear model
(KR with linear kernel)



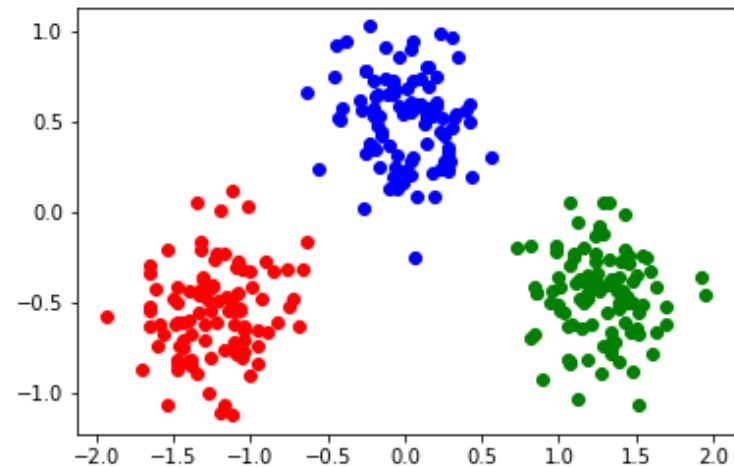
Kernel ridge regression: Example 3

- KRR also applies to linearly separable data
 - 2D inputs from 3 classes (colors)
 - Or an RBF kernel (here with a small λ)

True labels



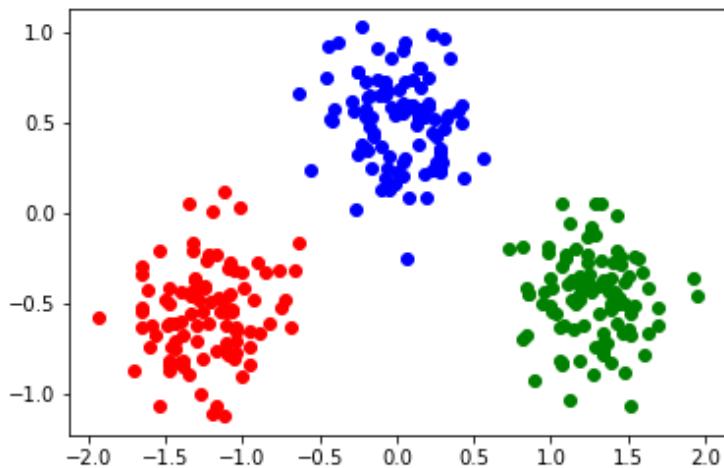
Fitted nonlinear model
(KRR with RBF kernel)



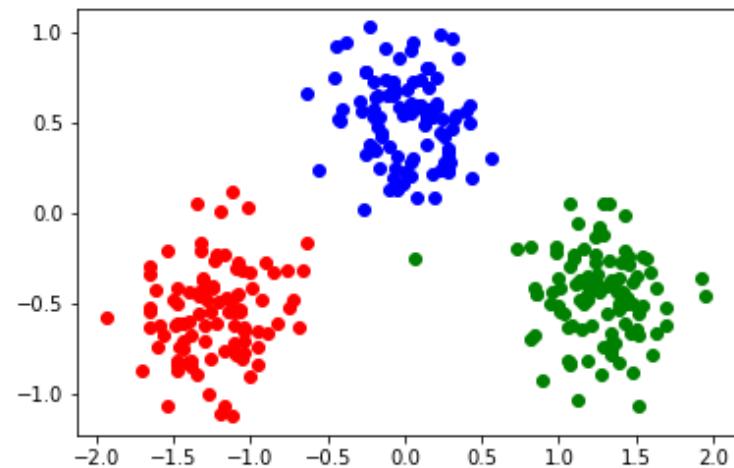
Kernel ridge regression: Example 3

- KRR also applies to linearly separable data
 - 2D inputs from 3 classes (colors)
 - Or an RBF kernel (here with a larger $\lambda = 1$)

True labels



Fitted nonlinear model
(KRR with RBF kernel)



Kernelization

- The idea of kernelizing an algorithm does not apply only to linear regression
 - Let us now study this for SVM
 - In a few weeks, we will also do this for dimensionality reduction
 - Note that logistic regression can also be kernelized, although this is less commonly done

Recap: SVM Dual formulation

- Instead of solving the original (primal) problem in terms of \mathbf{w} , SVM can also be expressed in a dual form (coming from Lagrange duality)
- When using slack variables to account for overlapping classes, the dual problem can be expressed as

$$\max_{\{\alpha_i\}} \sum_{i=1}^N \alpha_i - \frac{1}{2} \sum_{i=1}^N \sum_{j=1}^N \alpha_i \alpha_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j$$

subject to $\sum_{i=1}^N \alpha_i y_i = 0$

$$0 \leq \alpha_i \leq C, \quad \forall i$$

- This is a concave problem, which can be solved to optimality

Working with expanded features

- Writing the dual problem with expanded features yields

$$\max_{\{\alpha_i\}} \sum_{i=1}^N \alpha_i - \frac{1}{2} \sum_{i=1}^N \sum_{j=1}^N \alpha_i \alpha_j y_i y_j \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j)$$

subject to $\sum_{i=1}^N \alpha_i y_i = 0$

$0 \leq \alpha_i \leq C, \forall i$

We directly have the dot products we need!

Kernel SVM

- Therefore kernel SVM can be written as

$$\max_{\{\alpha_i\}} \sum_{i=1}^N \alpha_i - \frac{1}{2} \sum_{i=1}^N \sum_{j=1}^N \alpha_i \alpha_j y_i y_j k(\mathbf{x}_i, \mathbf{x}_j)$$

subject to $\sum_{i=1}^N \alpha_i y_i = 0$

$$0 \leq \alpha_i \leq C, \quad \forall i$$

- This remains a concave problem

Recap: Prediction

- Once the optimal $\{\alpha_i^*\}$ are obtained, one can use them to predict the label for a new sample \mathbf{x}
- It can be shown that, for the linear case

$$\tilde{\mathbf{w}}^* = \sum_{i=1}^N \alpha_i^* y_i \mathbf{x}_i$$

- This remains true with expanded features, i.e.,

$$\tilde{\mathbf{w}}^* = \sum_{i=1}^N \alpha_i^* y_i \phi(\mathbf{x}_i)$$

Prediction

- Therefore, the prediction for a new sample \mathbf{x} is given by

$$\begin{aligned}\hat{y}(\mathbf{x}) &= (\tilde{\mathbf{w}}^*)^T \phi(\mathbf{x}) + w^{(0)*} = \sum_{i=1}^N \alpha_i^* y_i \phi(\mathbf{x}_i)^T \phi(\mathbf{x}) + w^{(0)*} \\ &= \sum_{i=1}^N \alpha_i^* y_i k(\mathbf{x}_i, \mathbf{x}) + w^{(0)*}\end{aligned}$$

- Recall that, in both the linear and kernel case, for all samples that are not support vectors, we have $\alpha_i^* = 0$
- Therefore the sum can be computed on the support vectors only, i.e., with \mathcal{S} the set of support vector indices,

$$\hat{y}(\mathbf{x}) = \sum_{i \in \mathcal{S}} \alpha_i^* y_i k(\mathbf{x}_i, \mathbf{x}) + w^{(0)*}$$

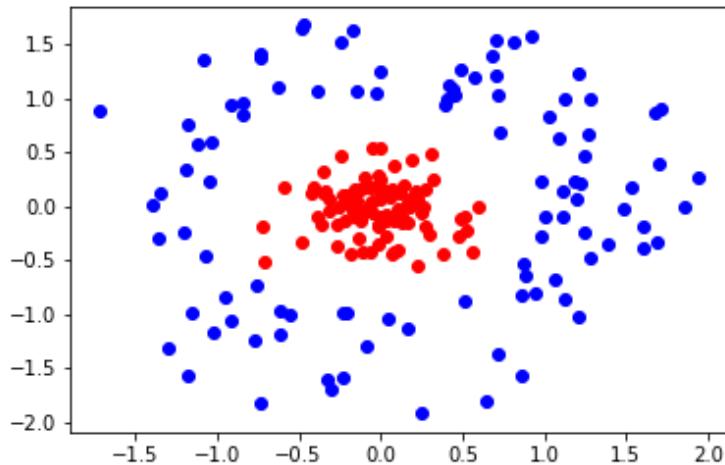
Side note: How about $w^{(0)*}$?

- As mentioned in the SVM lecture, $w^{(0)}$ disappeared from the dual formulation
- To understand how to compute $w^{(0)*}$, I invite the interested reader to look at Section 7.1 in Bishop's book
 - This is of course optional

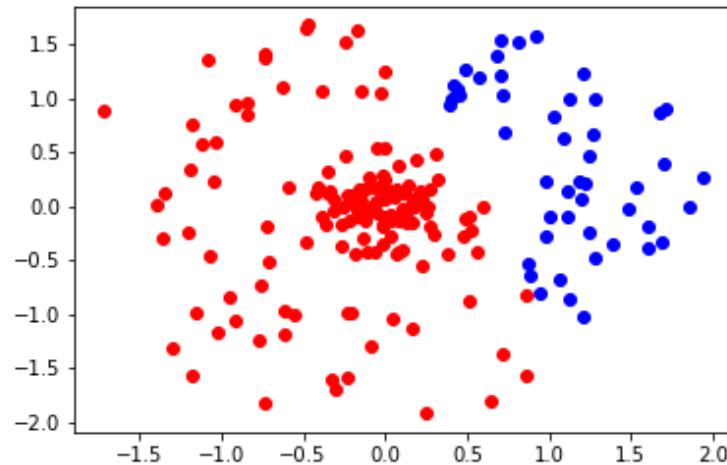
Kernel SVM: Example 1

- 2D inputs from 2 classes (colors)

True labels

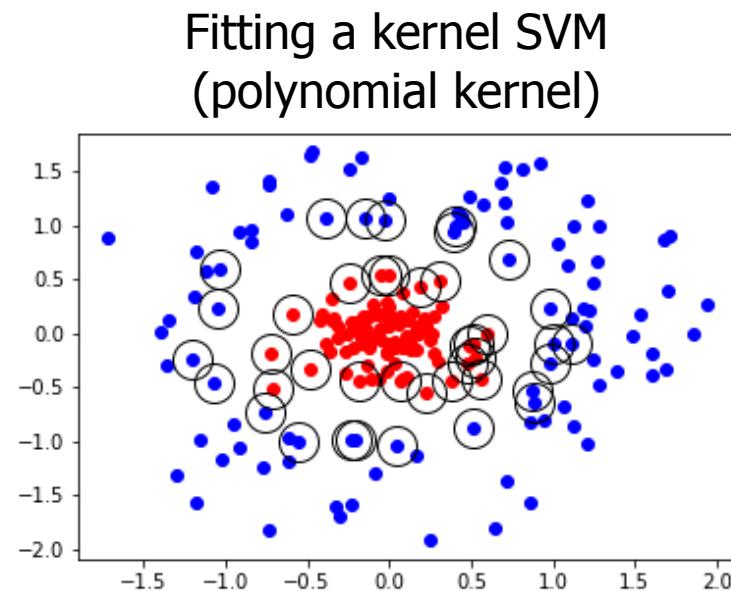
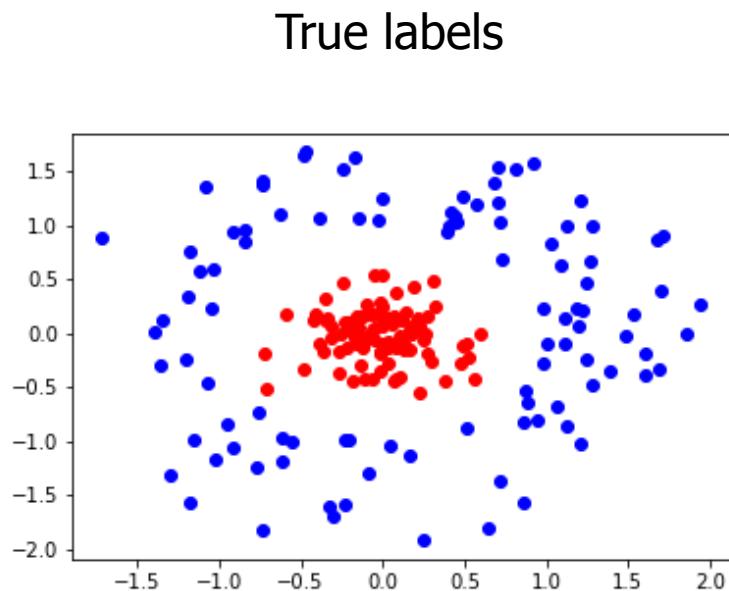


Fitting a linear SVM



Kernel SVM: Example 1

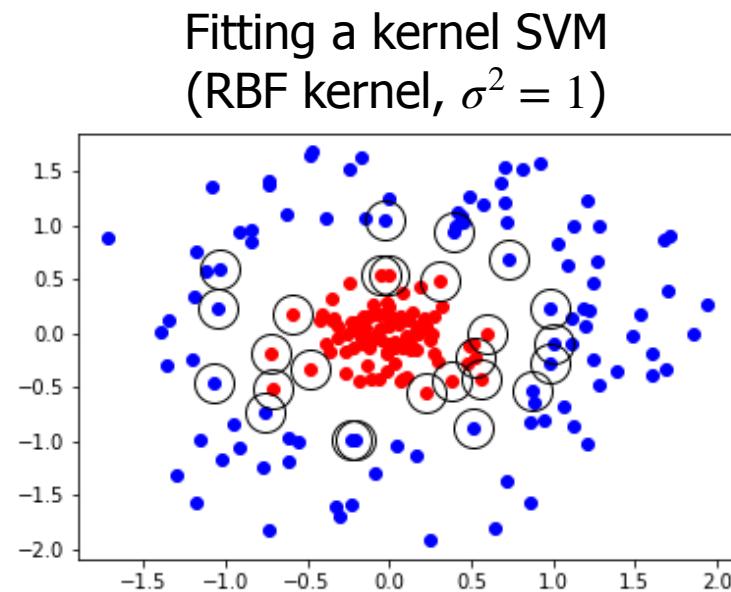
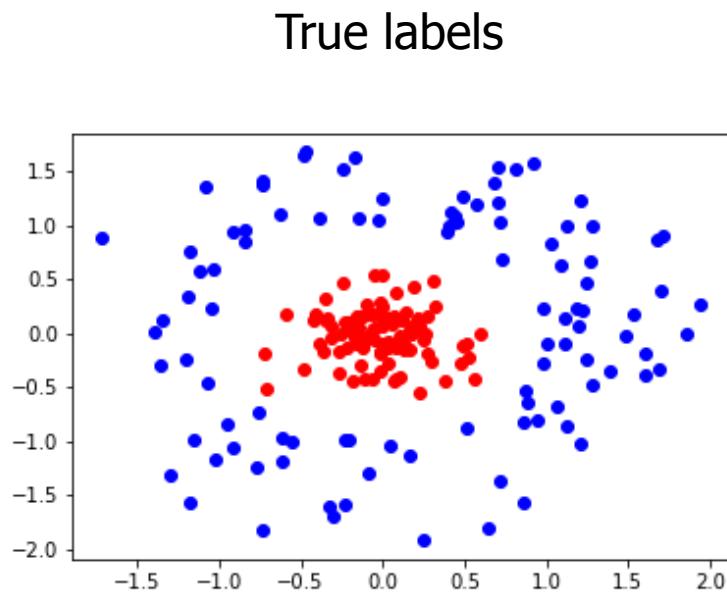
- 2D inputs from 2 classes (colors)



Circles indicate support vectors

Kernel SVM: Example 1

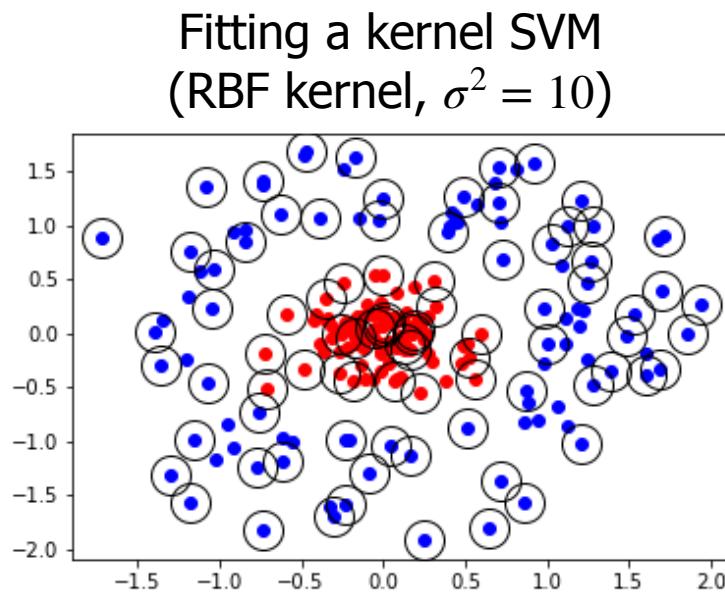
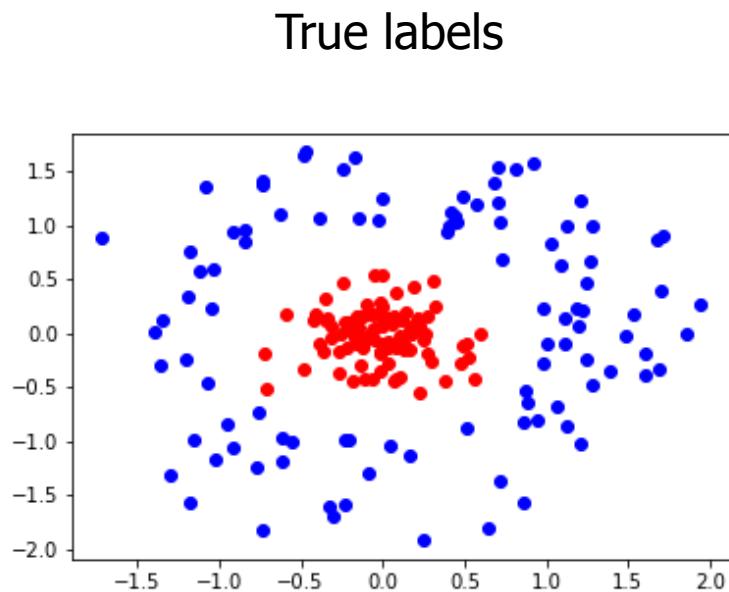
- 2D inputs from 2 classes (colors)



Circles indicate support vectors

Kernel SVM: Example 1

- 2D inputs from 2 classes (colors)

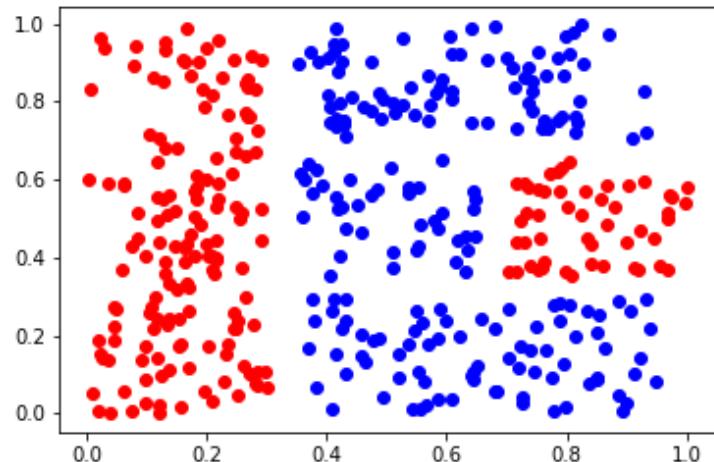


Circles indicate support vectors

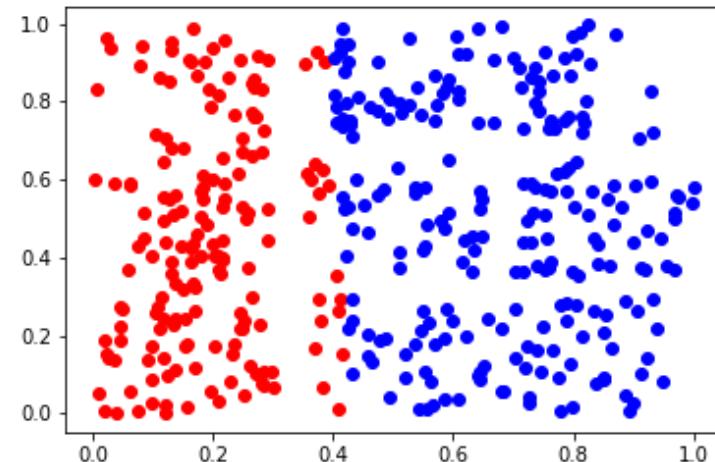
Kernel SVM: Example 2

- 2D inputs from 2 classes (colors)

True labels

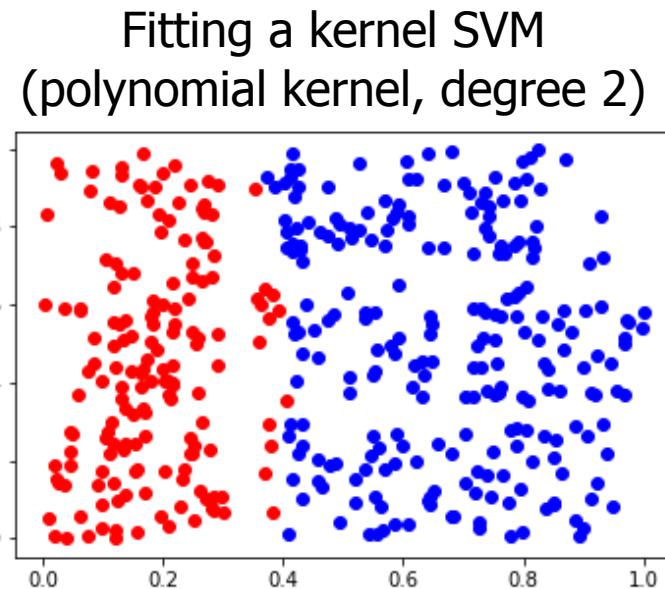
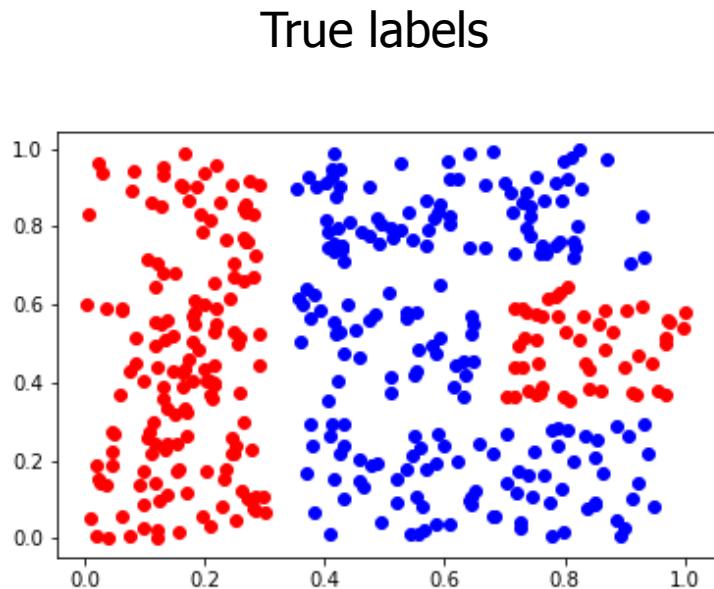


Fitting a linear SVM



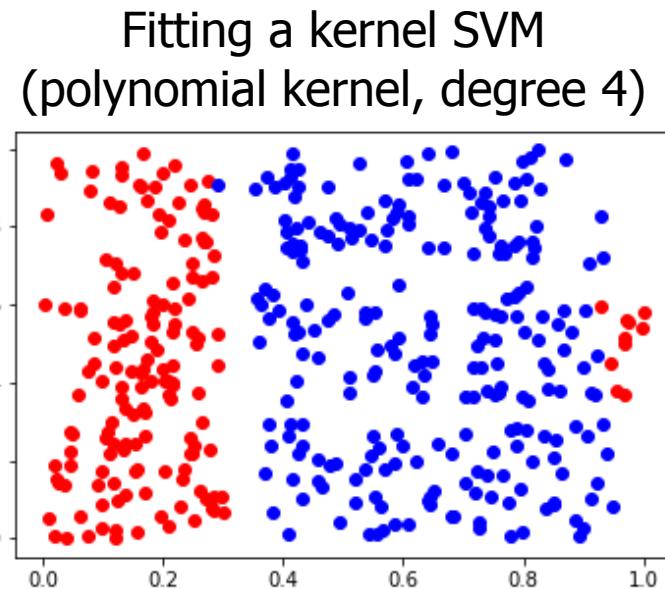
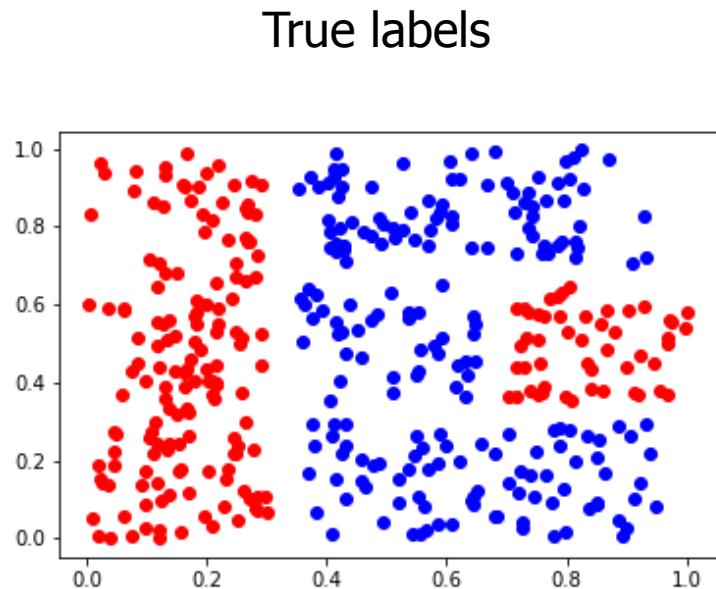
Kernel SVM: Example 2

- 2D inputs from 2 classes (colors)



Kernel SVM: Example 2

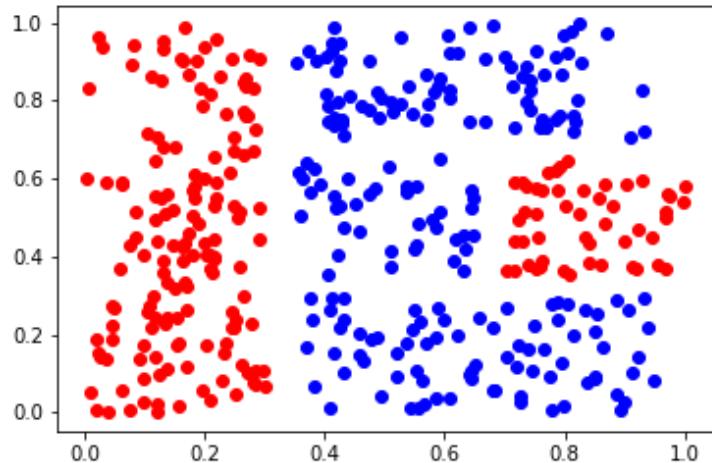
- 2D inputs from 2 classes (colors)



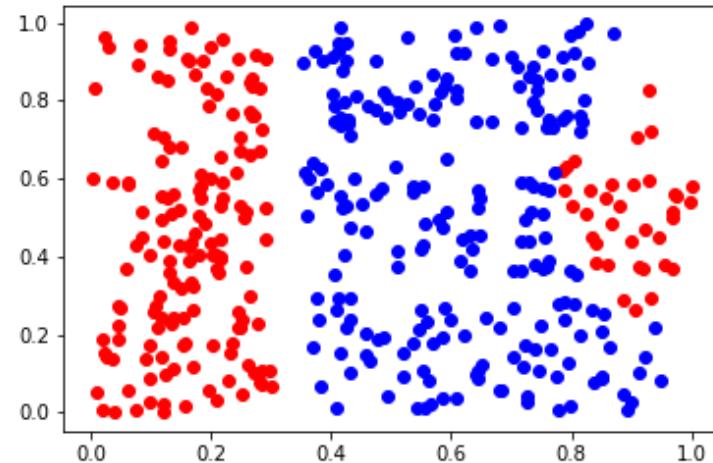
Kernel SVM: Example 2

- 2D inputs from 2 classes (colors)

True labels

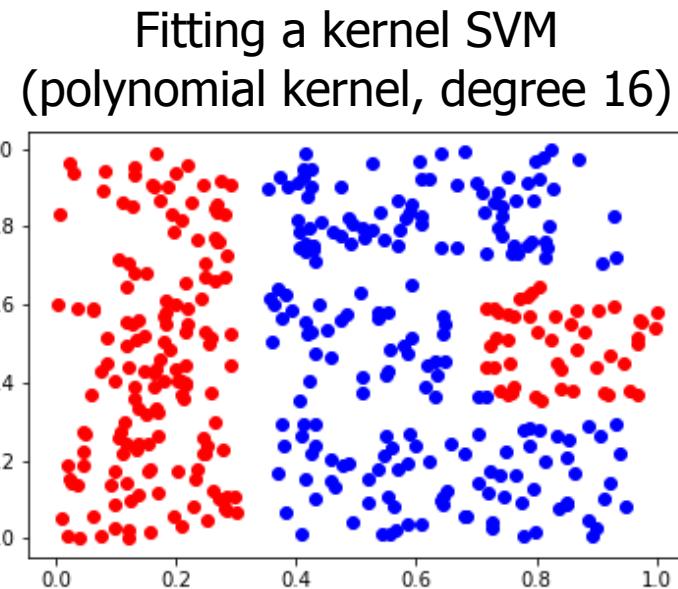
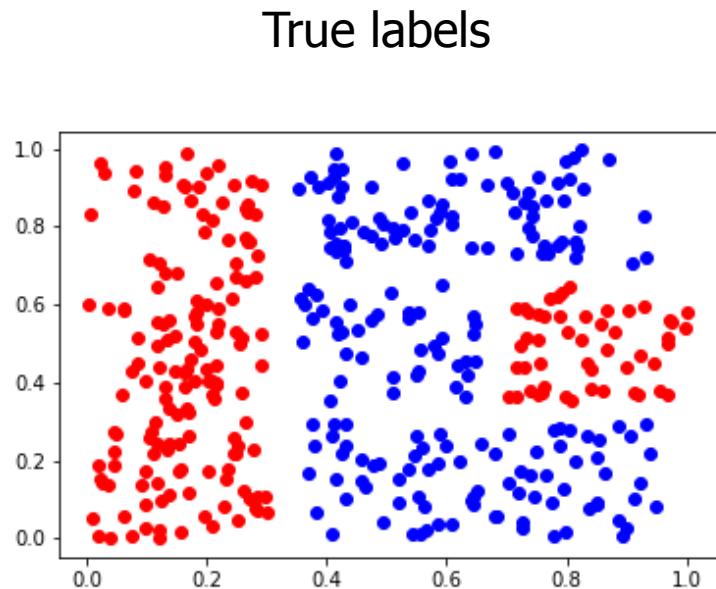


Fitting a kernel SVM
(polynomial kernel, degree 8)



Kernel SVM: Example 2

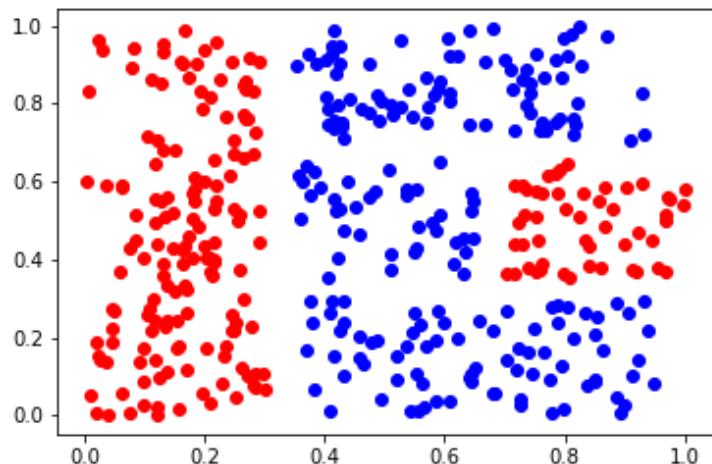
- 2D inputs from 2 classes (colors)



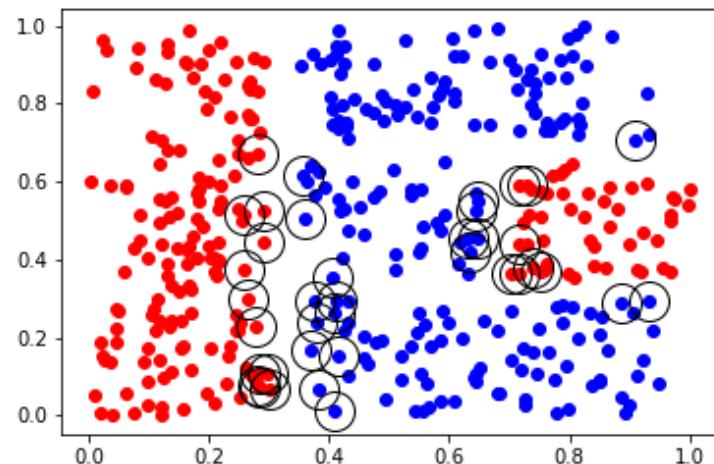
Kernel SVM: Example 2

- 2D inputs from 2 classes (colors)

True labels



Fitting a kernel SVM
(polynomial kernel, degree 16)

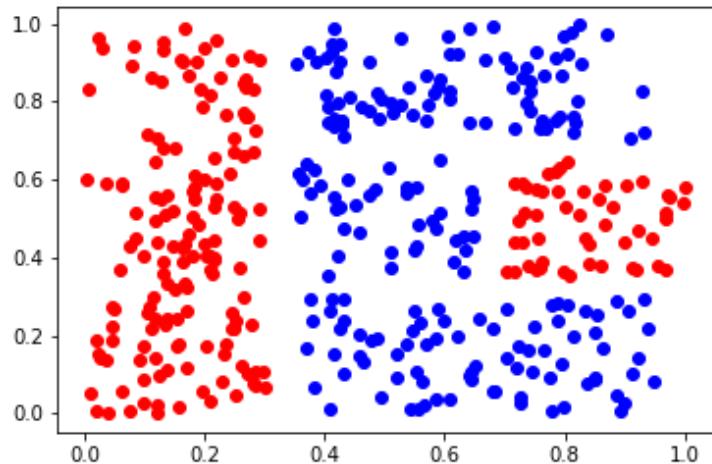


Circles indicate support vectors

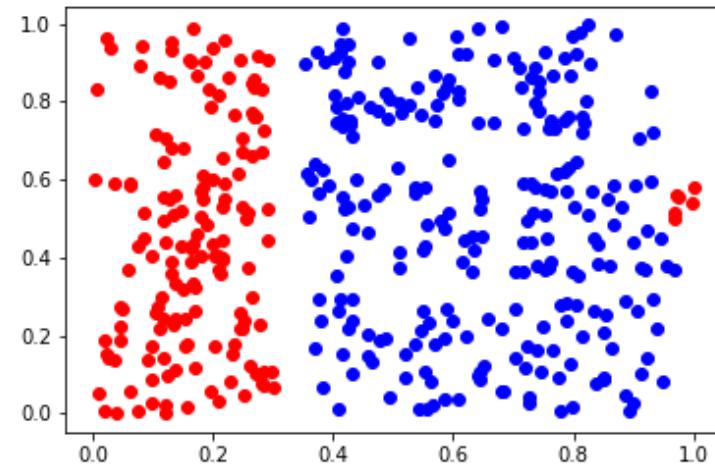
Kernel SVM: Example 2

- 2D inputs from 2 classes (colors)

True labels



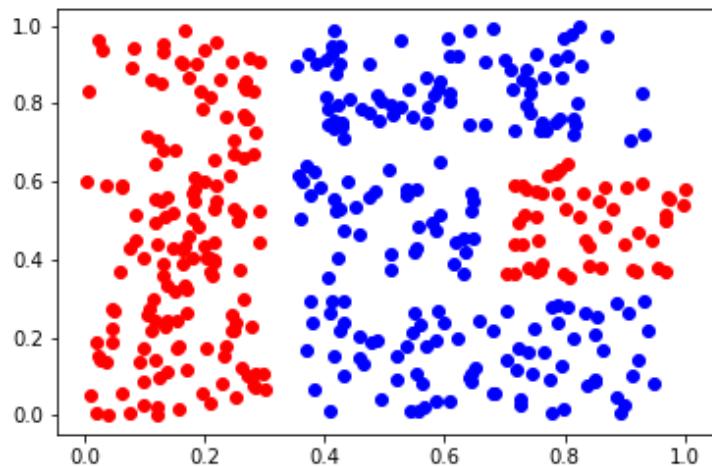
Fitting a kernel SVM
(RBF kernel, $\sigma^2 = 1$)



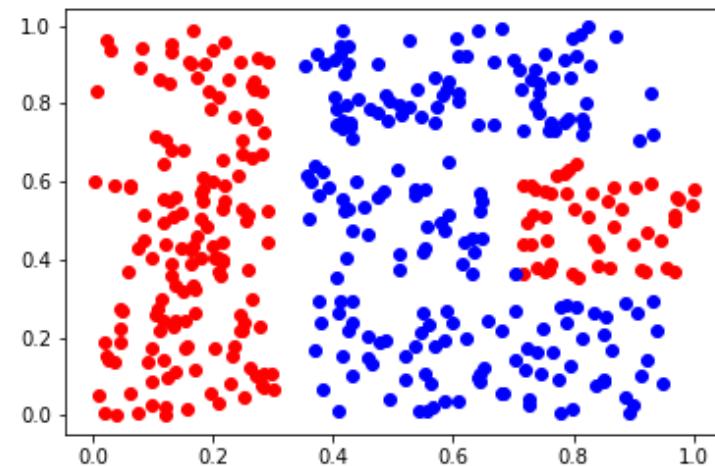
Kernel SVM: Example 2

- 2D inputs from 2 classes (colors)

True labels



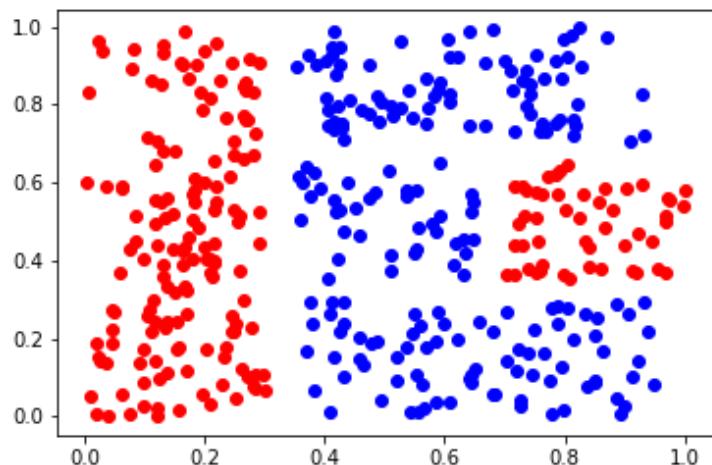
Fitting a kernel SVM
(RBF kernel, $\sigma^2 = 10$)



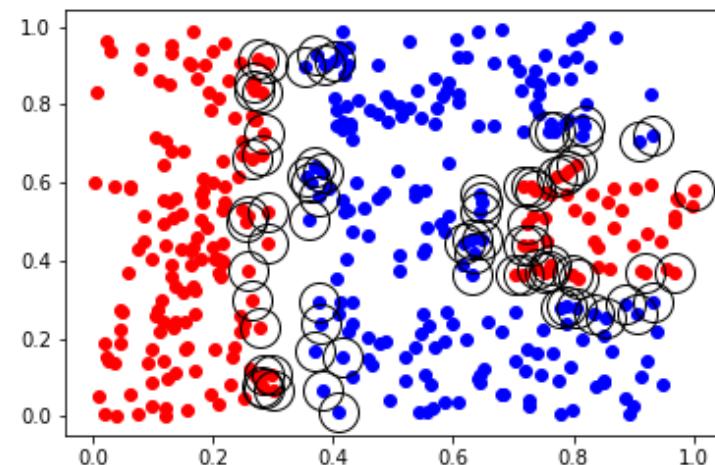
Kernel SVM: Example 2

- 2D inputs from 2 classes (colors)

True labels



Fitting a kernel SVM
(RBF kernel, $\sigma^2 = 10$)



Circles indicate support vectors

Kernel SVM: Demo

- <http://www.cristiandima.com/basics-of-support-vector-machines/>

Kernel SVM on MNIST

- Digit classification



Method	Preprocessing	Error	Reference
K-NN with non-linear deformation (P2DHMDM)	shiftable edges	0.52	Keyser et al. IEEE PAMI 2007

SVMs			
SVM, Gaussian Kernel	none	1.4	
SVM deg 4 polynomial	deskewing	1.1	LeCun et al. 1998
Reduced Set SVM deg 5 polynomial	deskewing	1.0	LeCun et al. 1998
Virtual SVM deg-9 poly [distortions]	none	0.8	LeCun et al. 1998
Virtual SVM, deg-9 poly, 1-pixel jittered	none	0.68	DeCoste and Scholkopf, MLJ 2002
Virtual SVM, deg-9 poly, 1-pixel jittered	deskewing	0.68	DeCoste and Scholkopf, MLJ 2002
Virtual SVM, deg-9 poly, 2-pixel jittered	deskewing	0.56	DeCoste and Scholkopf, MLJ 2002

Exercises

- Discuss two advantages and two drawbacks of kernel methods

Lecture 9: Data Representations & Processing

Goals of today's lecture

- Introduce the idea of data representations and one general method to represent data
- Study different data processing techniques
- Introduce the problem of class imbalance and solutions to address it

Heterogeneous data

- Text data: 20 Newsgroup dataset
 - Three samples

Although I realize that principle is not one of your strongest points, I would still like to know why do do not ask any question of this sort about the Arab countries.

If you want to continue this think tank charade of yours, your fixation on Israel must stop. You might have to start asking the same sort of questions of Arab countries as well. You realize it would not work, as the Arab countries' treatment of Jews over the last several decades is so bad that your fixation on Israel would begin to look like the biased attack that it is.

Everyone in this group recognizes that your stupid 'Center for Policy Research' is nothing more than a fancy name for some bigot who hates Israel.

I'd like to see this info as well. As for wavelength, I think you're primarily going to find two - 880 nM +/- a bit, and/or 950 nM +/- a bit. Usually it is about 10 nm either way. The two most common I have seen were 880 and 950 but I have also heard of 890 and 940. I'm not sure that the 10 nm one way or another will make a great deal of difference.

Another suggestion - find a brand of TV that uses an IR remote, and go look at the SAMS photofact for it. You can often find some very detailed schematics and parts list for not only the receiver but the transmitter as well, including carrier freq. specs. and tone decoding specs. if the system uses that.

Whoops!! Wrong group. Soooooooooooooorry folks..

- How can we leverage text documents of different length?
 - With a linear model, the number of parameters directly depends on the input dimension

More general question

- How can we represent data?

Why do we need data representations?

- Data **heterogeneity**
 - E.g., Vectorization of images



Size: 536x356x3

$$\mathbf{x}_i \in \mathbb{R}^{536 \cdot 356 \cdot 3} = \mathbb{R}^{572448}$$



Size: 500x523x3

$$\mathbf{x}_j \in \mathbb{R}^{500 \cdot 523 \cdot 3} = \mathbb{R}^{784500}$$

- We **cannot compare** (e.g., subtract) **vectors of different dimensions**
- Same problem for speech samples of different length

Why do we need data representations?

- Data size
 - E.g., Vectorization of an image



$$\mathbf{x}_i \in \mathbb{R}^{536 \cdot 356 \cdot 3} = \mathbb{R}^{572448}$$

- A single, fairly small image is already a very long vector
- How about the size of a complete dataset, e.g., ImageNet (1M images)



Why do we need data representations?

- **Data noisiness**

- Not every piece of data encodes valuable information
- E.g., Image recognition: Not all pixels are indicative of the class
- E.g., Text categorization: Not all words are indicative of the class



I'd like to see this info as well. As for wavelength, I think you're primarily going to find two - 880 nm +/- a bit, and/or 950 nm +/- a bit. Usually it is about 10 nM either way. The two most common I have seen were 880 and 950 but I have also heard of 890 and 940. I'm not sure that the 10 nM one way or another will make a great deal of difference.

Another suggestion - find a brand of TV that uses an IR remote, and go look at the SAMS photofact for it. You can often find some very detailed schematics and parts list for not only the receiver but the transmitter as well, including carrier freq. specs. and tone decoding specs. if the system uses that.

Why do we need data representations?

- Some representations are better suited to certain tasks than others
 - E.g., MNIST classification

Method	Preprocessing	Error	Reference
K-nearest-neighbors, Euclidean (L2)	none	5.0	LeCun et al. 1998
K-nearest-neighbors, Euclidean (L2)	none	3.09	Kenneth Wilder, U. Chicago
K-nearest-neighbors, L3	none	2.83	Kenneth Wilder, U. Chicago
K-nearest-neighbors, Euclidean (L2)	deskewing	2.4	LeCun et al. 1998
K-nearest-neighbors, Euclidean (L2)	deskewing, noise removal, blurring	1.80	Kenneth Wilder, U. Chicago
K-nearest-neighbors, L3	deskewing, noise removal, blurring	1.73	Kenneth Wilder, U. Chicago
K-nearest-neighbors, L3	deskewing, noise removal, blurring, 1 pixel shift	1.33	Kenneth Wilder, U. Chicago
K-nearest-neighbors, L3	deskewing, noise removal, blurring, 2 pixel shift	1.22	Kenneth Wilder, U. Chicago
K-NN with non-linear deformation (IDM)	shiftable edges	0.54	Keysers et al. IEEE PAMI 2007
K-NN with non-linear deformation (P2DHMDM)	shiftable edges	0.52	Keysers et al. IEEE PAMI 2007
K-NN, Tangent Distance	subsampling to 16x16 pixels	1.1	LeCun et al. 1998
K-NN, shape context matching	shape context feature extraction	0.63	Belongie et al. IEEE PAMI 2002

Data representations: Attributes

- E.g., UCI Abalone dataset
 - Predict the age of abalone from measurements
 - Attributes



Name	Data Type	Measurement	Description
Sex	Categorical		M, F, and I (infant)
Length	Numeric (continuous)	mm	Longest shell measurement
Diameter	Numeric (continuous)	mm	perpendicular to length
Height	Numeric (continuous)	mm	with meat in shell
WT_Whole	Numeric (continuous)	grams	whole abalone
WT_Shuck	Numeric (continuous)	grams	weight of meat
WT_Vscra	Numeric (continuous)	grams	gut weight (after bleeding)
WT_Shell	Numeric (continuous)	grams	after being dried
Rings	Numeric (integer)		+1.5 gives the age in years

Different types of attributes

- Categorical

Name	Data Type	Measurement	Description
Sex	Categorical		M, F, and I (infant)
Length	Numeric (continuous)	mm	Longest shell measurement
Diameter	Numeric (continuous)	mm	perpendicular to length
Height	Numeric (continuous)	mm	with meat in shell
WT_Whole	Numeric (continuous)	grams	whole abalone
WT_Shuck	Numeric (continuous)	grams	weight of meat
WT_Vscra	Numeric (continuous)	grams	gut weight (after bleeding)
WT_Shell	Numeric (continuous)	grams	after being dried
Rings	Numeric (integer)		+1.5 gives the age in years

Different types of attributes

- Real (continuous)

Name	Data Type	Measurement	Description
Sex	Categorical		M, F, and I (infant)
Length	Numeric (continuous)	mm	Longest shell measurement
Diameter	Numeric (continuous)	mm	perpendicular to length
Height	Numeric (continuous)	mm	with meat in shell
WT_Whole	Numeric (continuous)	grams	whole abalone
WT_Shuck	Numeric (continuous)	grams	weight of meat
WT_Vscra	Numeric (continuous)	grams	gut weight (after bleeding)
WT_Shell	Numeric (continuous)	grams	after being dried
Rings	Numeric (integer)		+1.5 gives the age in years

Different types of attributes

- Integer (discrete)

Name	Data Type	Measurement	Description
Sex	Categorical		M, F, and I (infant)
Length	Numeric (continuous)	mm	Longest shell measurement
Diameter	Numeric (continuous)	mm	perpendicular to length
Height	Numeric (continuous)	mm	with meat in shell
WT_Whole	Numeric (continuous)	grams	whole abalone
WT_Shuck	Numeric (continuous)	grams	weight of meat
WT_Vscra	Numeric (continuous)	grams	gut weight (after bleeding)
WT_Shell	Numeric (continuous)	grams	after being dried
Rings	Numeric (integer)		+1.5 gives the age in years

From attributes to dictionaries

- With **attributes**, all samples have the same length (number of attributes)
- Can we extract something similar to attributes from text samples?

Although I realize that principle is not one of your strongest points, I would still like to know why do do not ask any question of this sort about the Arab countries.

If you want to continue this think tank charade of yours, your fixation on Israel must stop. You might have to start asking the same sort of questions of Arab countries as well. You realize it would not work, as the Arab countries' treatment of Jews over the last several decades is so bad that your fixation on Israel would begin to look like the biased attack that it is.

Everyone in this group recognizes that your stupid 'Center for Policy Research' is nothing more than a fancy name for some bigot who hates Israel.

I'd like to see this info as well. As for wavelength, I think you're primarily going to find two - 880 nM +/- a bit, and/or 950 nM +/- a bit. Usually it is about 10 nM either way. The two most common I have seen were 880 and 950 but I have also heard of 890 and 940. I'm not sure that the 10 nM one way or another will make a great deal of difference.

Another suggestion - find a brand of TV that uses an IR remote, and go look at the SAMS photofact for it. You can often find some very detailed schematics and parts list for not only the receiver but the transmitter as well, including carrier freq. specs. and tone decoding specs. if the system uses that.

Whoops!! Wrong group. Soooooooooooooorry folks..

From attributes to dictionaries

- All samples are written in the same language
- They relate to the same dictionary

Although I realize that principle is not one of your strongest points, I would still like to know why do do not ask any question of this sort about the Arab countries.

If you want to continue this think tank charade of yours, your fixation on Israel must stop. You might have to start asking the same sort of questions of Arab countries as well. You realize it would not work, as the Arab countries' treatment of Jews over the last several decades is so bad that your fixation on Israel would begin to look like the biased attack that it is.

Everyone in this group recognizes that your stupid 'Center for Policy Research' is nothing more than a fancy name for some bigot who hates Israel.

I'd like to see this info as well. As for wavelength, I think you're primarily going to find two - 880 nM +/- a bit, and/or 950 nM +/- a bit. Usually it is about 10 nM either way. The two most common I have seen were 880 and 950 but I have also heard of 890 and 940. I'm not sure that the 10 nM one way or another will make a great deal of difference.

Another suggestion - find a brand of TV that uses an IR remote, and go look at the SAMS photofact for it. You can often find some very detailed schematics and parts list for not only the receiver but the transmitter as well, including carrier freq. specs. and tone decoding specs. if the system uses that.

Whoops!! Wrong group. Soooooooooooooorry folks..

Bag of words

- Idea: For each sample, count the number of times each word in the dictionary appears
- Example (from Wikipedia):
 - 2 samples

(1) John likes to watch movies. Mary likes movies too.

(2) John also likes to watch football games.

- Corresponding counts:

```
BoW1 = {"John":1, "likes":2, "to":1, "watch":1, "movies":2, "Mary":1, "too":1};  
BoW2 = {"John":1, "also":1, "likes":1, "to":1, "watch":1, "football":1, "games":1};
```

Creating a dictionary

- Consider only the words that appear in the dataset (not the entire English dictionary)
 - Example

```
"John", "likes", "to", "watch", "movies", "Mary", "too", "also", "football", "games"  
-----> ----->  
From Sample 1           From Sample 2
```

- Do not consider the too common words, e.g., “the”, “a”, “to”, “it”, “is”, “for”, ...
 - Example (removed “to”)

```
"John", "likes", "watch", "movies", "Mary", "too", "also", "football", "games"
```

Vector representation of BoW

- With D words in the dictionary, each sample can be represented as a vector in \mathbb{R}^D
- Example:
 - With dictionary ($D = 9$)

"John", "likes", "watch", "movies", "Mary", "too", "also", "football", "games"

- the BoWs

```
BoW1 = {"John":1, "likes":2, "to":1, "watch":1, "movies":2, "Mary":1, "too":1};  
BoW2 = {"John":1, "also":1, "likes":1, "to":1, "watch":1, "football":1, "games":1};
```

- become

$$\tilde{\mathbf{x}}_1 = [1 \ 2 \ 1 \ 2 \ 1 \ 1 \ 0 \ 0 \ 0]$$

$$\tilde{\mathbf{x}}_2 = [1 \ 1 \ 1 \ 0 \ 0 \ 0 \ 1 \ 1 \ 1]$$

Histograms

- For text samples of different length, the actual word counts may not be relevant
 - Longer texts will just have larger values
- Instead, one can encode the proportion of each word
- This is achieved by dividing each vector element by the sum of all elements
 - Example:

$$\tilde{\mathbf{x}}_1 = [1 \ 2 \ 1 \ 2 \ 1 \ 1 \ 0 \ 0 \ 0] \quad \sum_{d=1}^D \tilde{x}_1^{(d)} = 8$$

Final representation: $\mathbf{x}_1 = [1/8 \ 1/4 \ 1/8 \ 1/4 \ 1/8 \ 1/8 \ 0 \ 0 \ 0]$

Histograms

- The final BoW representation

$$\mathbf{x}_1 = [1/8 \ 1/4 \ 1/8 \ 1/4 \ 1/8 \ 1/8 \ 0 \ 0 \ 0]$$

- is a histogram
- It is such that

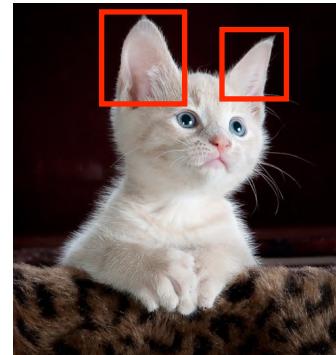
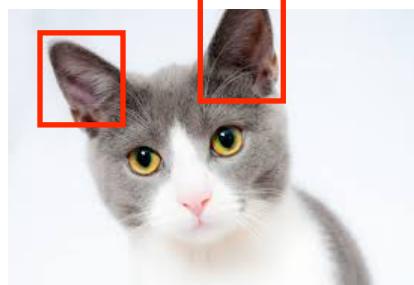
$$x_1^{(d)} \in [0,1]$$

$$\sum_{d=1}^D x_1^{(d)} = 1$$

- In this way, each sample is represented with a vector in \mathbb{R}^D

How about images?

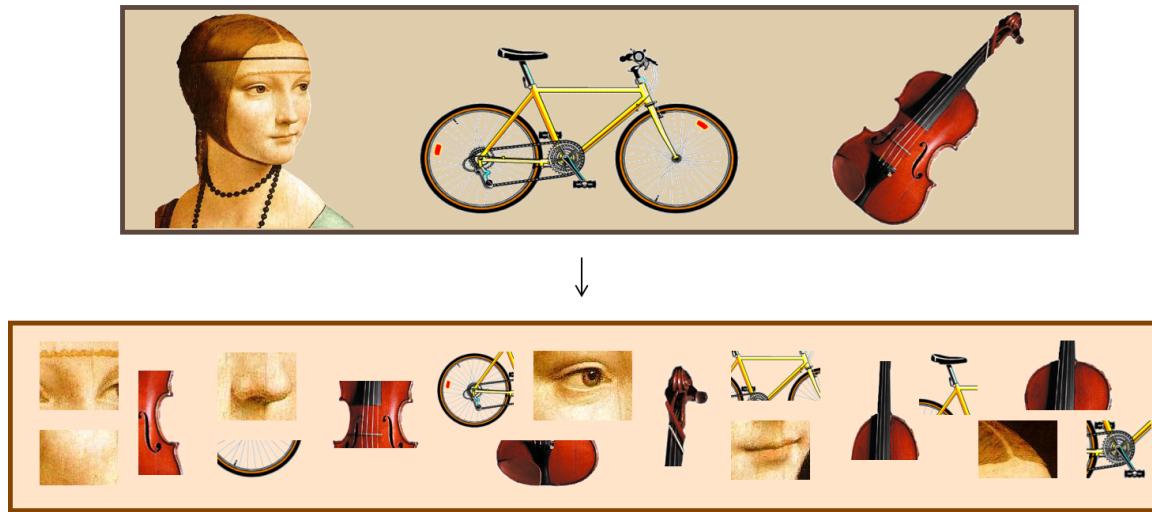
- No direct notion of “words” for image data



- However, objects are made of common parts/patches
 - They can be thought of as visual words

Visual dictionary (Example by K. Grauman)

- Extract patches from all images



- Each patch is unique, so it cannot be considered as a word
 - However, some patches look similar

Visual dictionary (Example by K. Grauman)

- Group the patches based on similarity to obtain prototypes that act as dictionary words



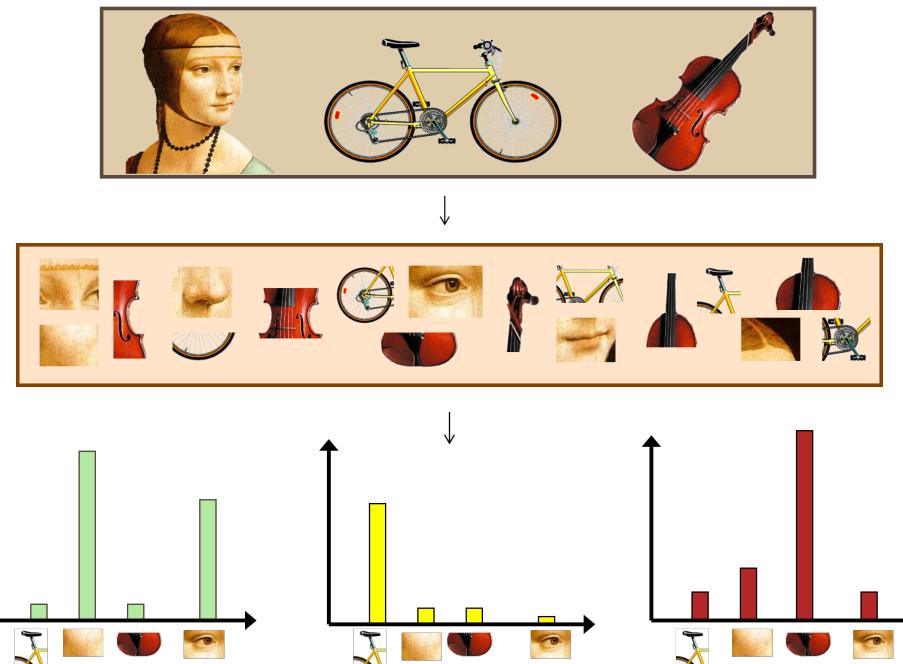
Clustering
algorithm
(more detail in a
future lecture)



- The center of each cluster (group) is taken as a visual word

Bag of visual words

- For each patch in an image
 - find the nearest visual word
 - add one to the corresponding element in a BoW vector
- Normalize the BoW vectors to obtain histograms



Local patch representation

- This still requires comparing local image patches
 - The patches can all have the same size
 - They can be small enough for this to be manageable
- Other local image representations can be used
 - This goes beyond the scope of this course

Exercise

- We have seen that text and images can be represented as Bags of Words. What other type of data can you represent in this way? How would you do it?

Data representation

- Large, heterogeneous data can be made compact and homogenous
 - E.g., via BoW that yield histograms of the same length
- The resulting representations can be directly used as input to an ML algorithm
 - This is applicable to all the methods we have seen so far
 - But in general to most machine learning algorithms

Data pre-processing

- Even with a good representation, the **data might benefit from being pre-processed**
- This is because, e.g.,
 - Some attribute values are missing
 - Some attribute values are noisy
 - The feature dimensions are of incommensurate magnitudes

Missing data

- Some attribute values were not recorded
 - E.g., the Air Quality UCI dataset contains missing values, which are tagged with a -200 value
- Potential solutions
 - Ignore the corresponding sample
 - Fill in the missing values with a global constant
 - Fill in the missing values with the corresponding attribute average over the dataset
 - Fill in the missing values with the corresponding attribute average over the samples belonging to the same class

Noisy data

- **Noise:**
 - Random measurement error
- **Incorrect values**
 - Faulty data collection
 - Data entry problems
 - Data transmission problems
- **Other problems**
 - **Duplicate entries**
 - **Inconsistent data**

Cleaning noisy data

- **Binning**

For each attribute:

1. Sort the data
2. Partition it into bins
3. Smooth the data by taking the bin means, or medians

- **Data :** 0, 4, 12, 16, 16, 18, 24, 26, 28

- **Equal width**

- Bin 1: 0, 4 [-, 10)
- Bin 2: 12, 16, 16, 18 [10, 20)
- Bin 3: 24, 26, 28 [20, +) .

- **Equal frequency**

- Bin 1: 0, 4, 12 [-, 14)
- Bin 2: 16, 16, 18 [14, 21)
- Bin 3: 24, 26, 28 [21, +)

Cleaning noisy data

- Clustering
 - Detect outliers as the clusters with too few elements and remove these samples
- Discussion: What other solution to cleaning noisy data can you think of?

Data normalization

- Goal: Scale each individual attribute/feature dimension to fall within a specified range
- Min-max normalization
 - For each feature dimension d , compute the normalized feature

$$\tilde{x}_i^{(d)} = \frac{x_i^{(d)} - x_{min}^{(d)}}{x_{max}^{(d)} - x_{min}^{(d)}}$$

where $x_{min}^{(d)} = \min_{i=1}^N x_i^{(d)}$

$$x_{max}^{(d)} = \max_{i=1}^N x_i^{(d)}$$

Data normalization

- z-score normalization
 - For each feature dimension d , compute the normalized feature

$$\tilde{x}_i^{(d)} = \frac{x_i^{(d)} - \mu^{(d)}}{\sigma^{(d)}}$$

where $\mu^{(d)} = \frac{1}{N} \sum_{i=1}^N x_i^{(d)}$ \rightarrow mean

$$\sigma^{(d)} = \sqrt{\frac{1}{N} \sum_{i=1}^N (x_i^{(d)} - \mu^{(d)})^2}$$
 \rightarrow Standard deviation

Data normalization

- max normalization
 - For each feature dimension d , compute the normalized feature

$$\tilde{x}_i^{(d)} = \frac{x_i^{(d)}}{x_{max}^{(d)}}$$

where $x_{max}^{(d)} = \max_{i=1}^N |x_i^{(d)}|$

Data normalization

- Normalization by decimal scaling
 - For each feature dimension d , compute the normalized feature

$$\tilde{x}_i^{(d)} = \frac{x_i^{(d)}}{10^k}$$

where k is the smallest integer such that $\max_{i=1}^N |\tilde{x}_i^{(d)}| \leq 1$

- The normalized (or cleaned) features are then used as input to an ML algorithm
 - Normalization is not always necessary, and may or may not help depending on the data at hand and on the chosen ML algorithm

From data sample to data set

- Data representations and data cleaning/normalization acts on the individual samples
 - Although the normalization statistics come from the whole training set
- Nevertheless, even with good representations, the data set itself might make learning more difficult
 - Let us look at some of these situations and how they can be addressed
 - The following material is adapted from slides by Aude Billard @ EPFL and Enrique Garcia Ceja @ University of Oslo on the topic of imbalanced data

Imbalanced data: Example

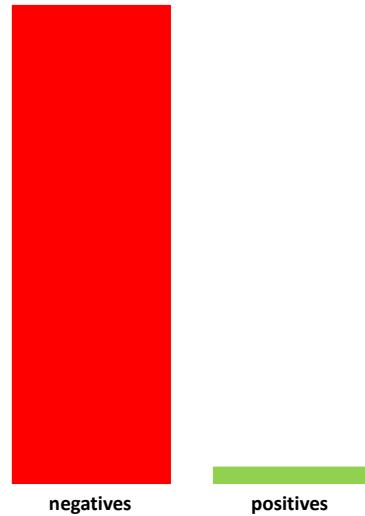
- You train a classifier to predict if a subject suffers from cancer (positive) or not (negative) from image data
- On validation data, your classifier gives you 99.1% accuracy
 - This sounds fantastic! We are ready to deploy it in the real world
- Out of curiosity (or thoroughness), you still look at the confusion matrix

		Predicted	
		Negative	Positive
Actual	Negative	990	0
	Positive	9	1

You missed almost all the cancerous cases!

Imbalanced data: Example

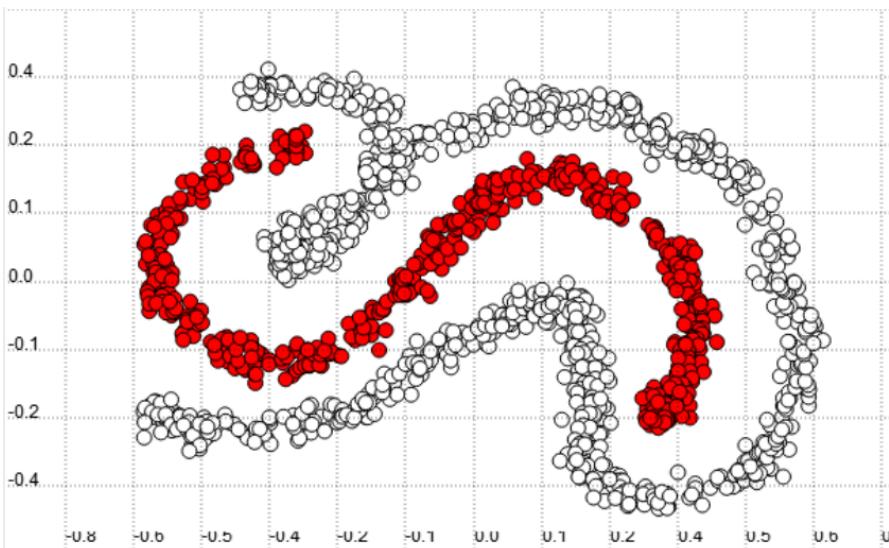
- What happened?
 - Let's look at the statistics of the training data



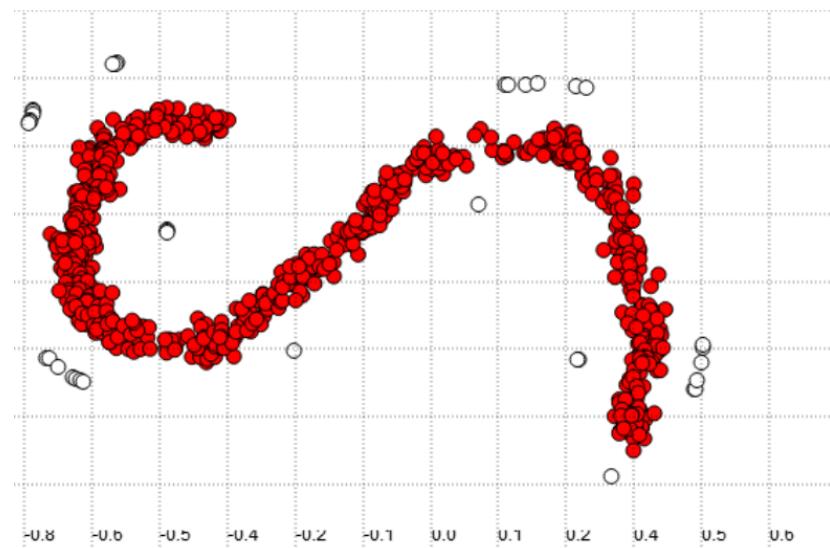
- Most of the samples are negative
 - So the classifier just learned to predict “negative” most of the time

Imbalanced vs balanced data

- Between-class imbalance



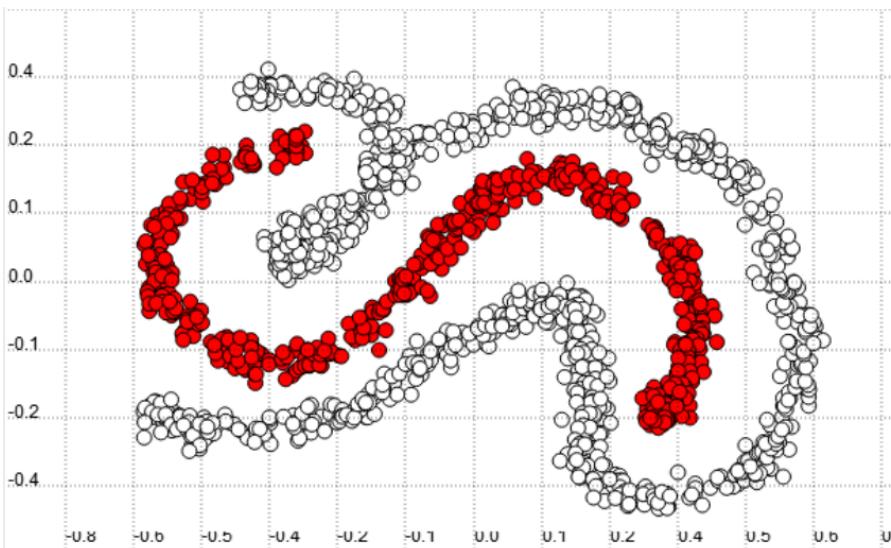
Balanced data



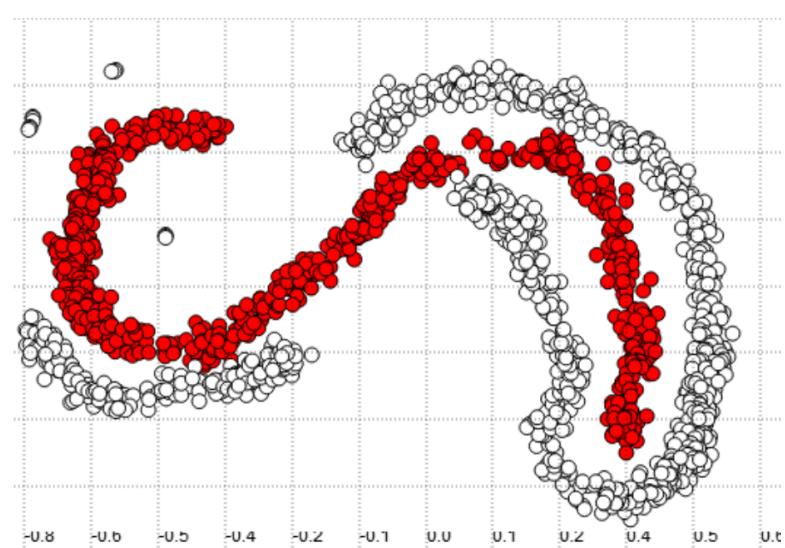
Imbalanced data

Imbalanced vs balanced data

- Within-class imbalance

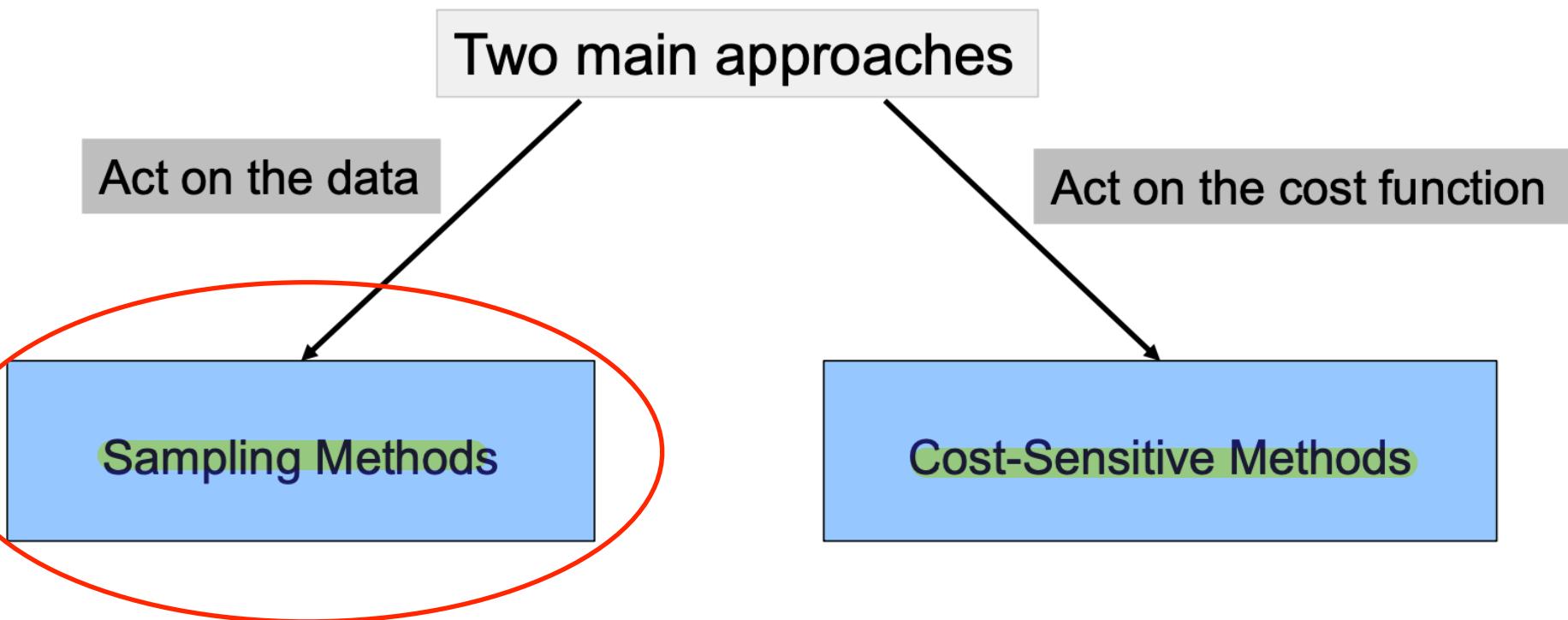


Balanced data

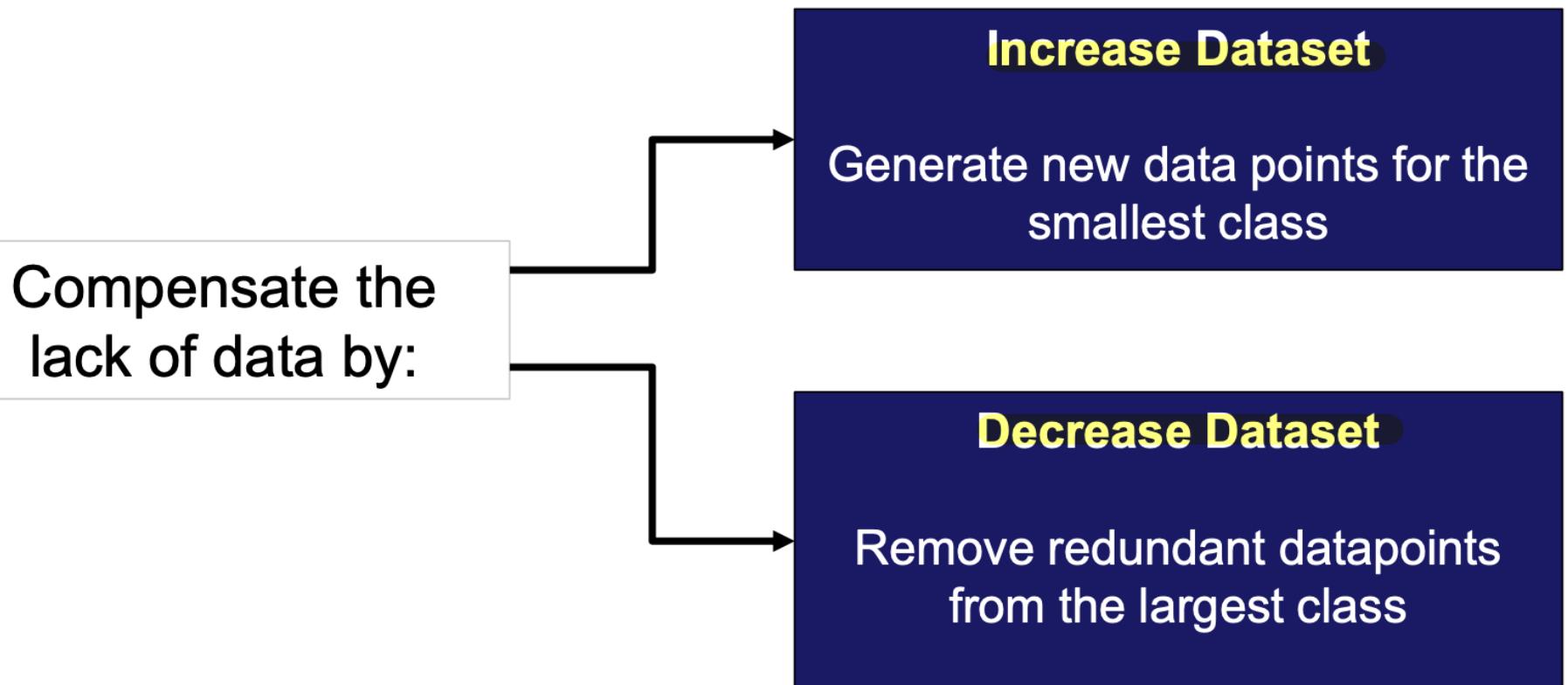


Imbalanced data

Learning with imbalanced data

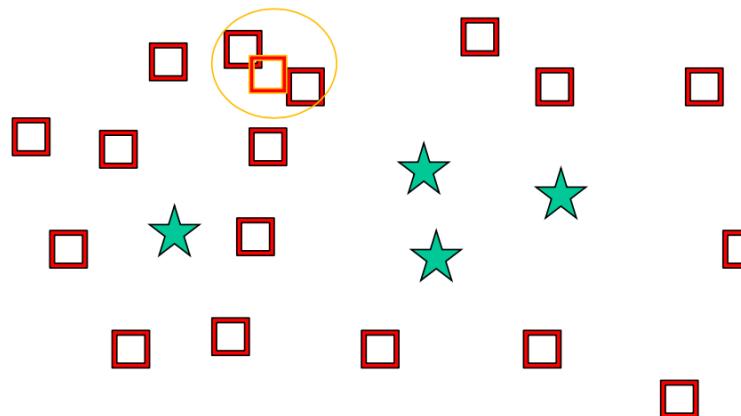


Sampling methods



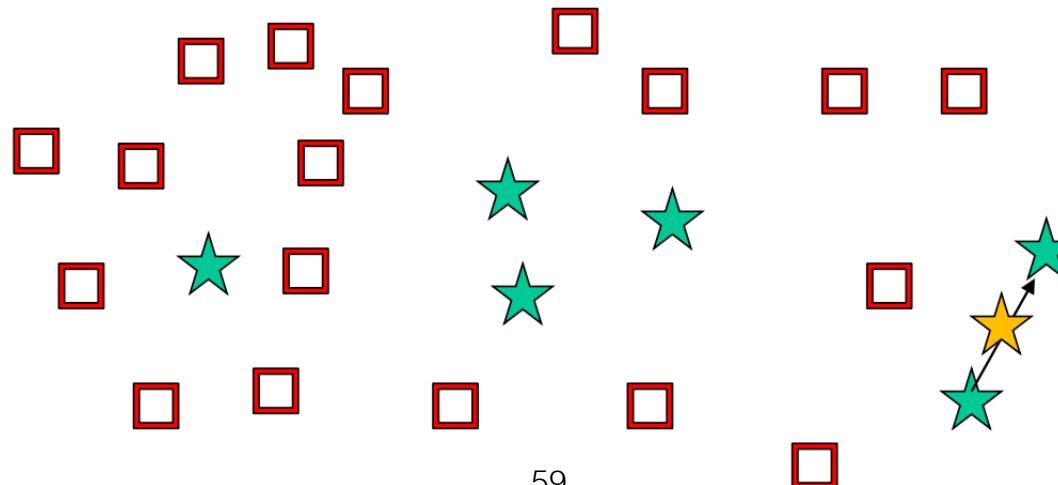
Decrease the data: Undersampling

- Naive (random) undersampling:
 - Randomly remove samples from the largest class until you obtain a balanced data set
 - Drawback: May remove important samples
- More clever undersampling:
 - Remove the redundant samples
 - Good if the class to undersample is densely populated (e.g., low-dimensional representation because of the curse of dimensionality)



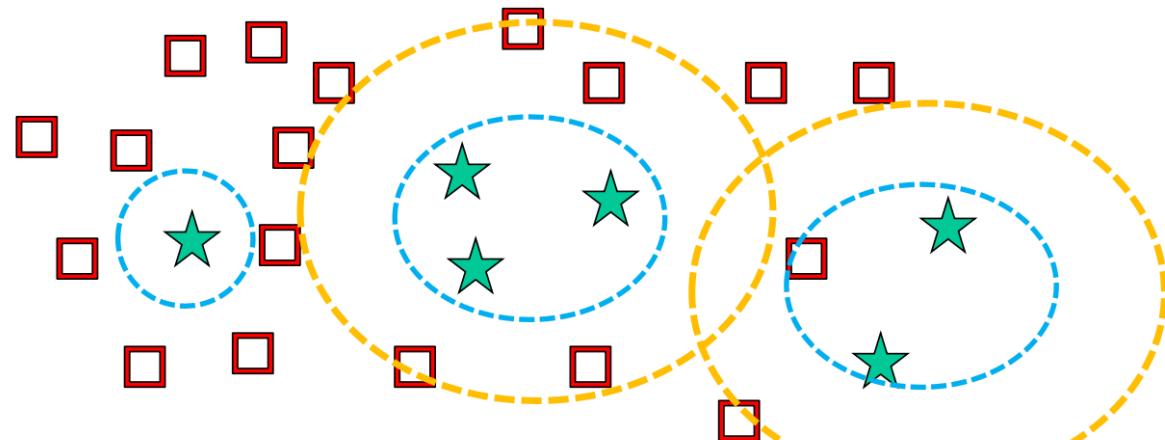
Increase the data: Oversampling

- Naive (random) oversampling:
 - Replicate the data samples from the smallest class until you obtain a balanced data set
 - Drawback: May lead to overfitting
- More clever oversampling:
 - Create new data samples based on neighboring existing ones
 - This may create noisy samples



Oversampling

- SMOTE: Synthetic minority oversampling technique (Chawla et al., JAIR 2002)
- AdaSyn: Adaptive synthetic sampling (He et al., IJCNN 2008)
- Reason about the density of data points and the border with the other class



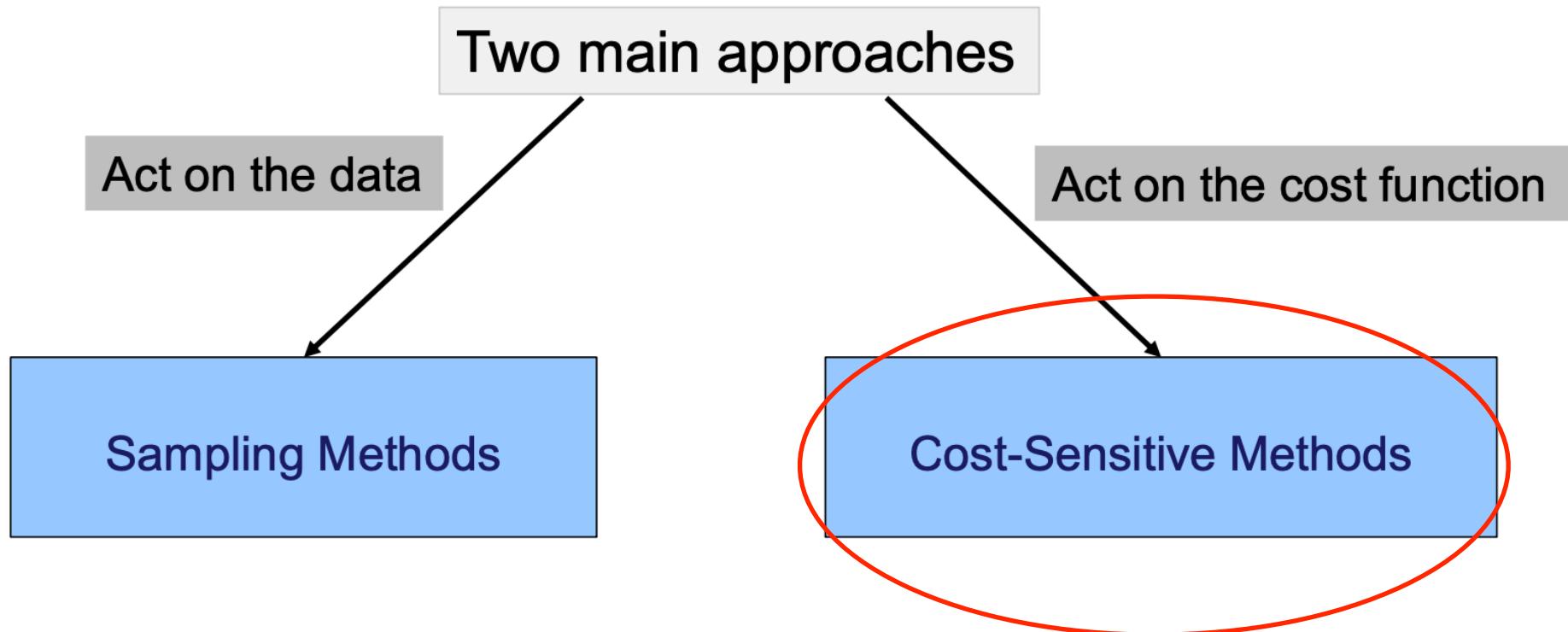
No Neighbors of the
same class → **noise**

Several Neighbors
of the same class

Surrounded by the
other class
→ **in danger**

Surrounded only on one
side by the other class
→ **safe**

Recap: Learning with imbalanced data



Empirical risk

- Given N training samples $\{(\mathbf{x}_i, y_i)\}$, the empirical risk is defined as

$$R(\{\mathbf{x}_i\}, \{y_i\}, \mathbf{w}) = \frac{1}{N} \sum_{i=1}^N \ell(\hat{y}_i, y_i)$$

where $\{\mathbf{x}_i\}$ and $\{y_i\}$ are the sets of training inputs and labels, respectively

- In general, each loss term $\ell(\hat{y}_i, y_i)$ has the same influence
- With imbalanced data, this means that most of the empirical risk comes from the largest class

Sample re-weighting

- A simple solution to this problem consist of re-weighting the samples, to put more emphasis on the rare class
- This gives an empirical risk of the form

$$R(\{\mathbf{x}_i\}, \{y_i\}, \mathbf{w}) = \frac{1}{N} \sum_{i=1}^N \beta_i \ell(\hat{y}_i, y_i)$$

where β_i is the weight for sample i

- Note that β_i cannot be optimized, because the optimal solution would simply be to set all of them to 0
- Instead, they are set manually, e.g., inversely proportional to the class frequency:
$$\beta_i = 1 - \frac{N_k}{N}$$
, if sample i belongs to class k , which contains N_k samples

Sample re-weighting

- This strategy is quite general
- E.g., it applies to least-square classification:

$$\min_{\mathbf{W}} \sum_{i=1}^N \beta_i \|\mathbf{W}^T \mathbf{x}_i - \mathbf{y}_i\|^2 + \lambda \|\mathbf{W}\|_F^2$$

- logistic regression:

$$\min_{\mathbf{W}} = - \sum_{i=1}^N \beta_i \sum_{k=1}^C y_i^{(k)} \ln(\mathbf{w}_{(k)}^T \mathbf{x}_i)$$

Sample re-weighting

- and support vector machines:

$$\begin{aligned} & \min_{\tilde{\mathbf{w}}, w^{(0)}, \{\xi_i\}} \frac{1}{2} \|\tilde{\mathbf{w}}\|^2 + C \sum_{i=1}^N \beta_i \xi_i \\ & \text{subject to } y_i \cdot (\tilde{\mathbf{w}}^T \mathbf{x}_i + w^{(0)}) \geq 1 - \xi_i, \quad \forall i \\ & \quad \xi_i \geq 0, \quad \forall i \end{aligned}$$

- In the three cases, since the weights are fixed during optimization, the respective optimization problems remain convex and can thus be solved to optimality

From handcrafted representations to learned ones

- The representations and data processing strategies we have discussed so far were all manually designed
- There is, however, no guarantee that they are the best representations for the task at hand
- Ideally, one would therefore like to learn the representation jointly with the classifier
 - This will bring us to the topic of artificial neural networks

Lecture 10: Deep Learning (Part 1)

Recap: Data representations

- Data heterogeneity
- Data size
 - E.g., Vectorization of images



Size: 536x356x3

$$\mathbf{x}_i \in \mathbb{R}^{536 \cdot 356 \cdot 3} = \mathbb{R}^{572448}$$



Size: 500x523x3

$$\mathbf{x}_j \in \mathbb{R}^{500 \cdot 523 \cdot 3} = \mathbb{R}^{784500}$$

- Data noisiness
 - Not every piece of data encodes valuable information

Recap: Attributes

- E.g., UCI Abalone dataset
 - Predict the age of abalone from measurements



Name	Data Type	Measurement	Description
Sex	Categorical		M, F, and I (infant)
Length	Numeric (continuous)	mm	Longest shell measurement
Diameter	Numeric (continuous)	mm	perpendicular to length
Height	Numeric (continuous)	mm	with meat in shell
WT_Whole	Numeric (continuous)	grams	whole abalone
WT_Shuck	Numeric (continuous)	grams	weight of meat
WT_Vscra	Numeric (continuous)	grams	gut weight (after bleeding)
WT_Shell	Numeric (continuous)	grams	after being dried
Rings	Numeric (integer)		+1.5 gives the age in years

- With attributes, all samples have the same length

Recap: Bag of words

- Idea: For each sample, count the number of times each word in the dictionary appears
- Example (from Wikipedia):
 - 2 samples

(1) John likes to watch movies. Mary likes movies too.

(2) John also likes to watch football games.

- Corresponding counts:

```
BoW1 = {"John":1, "likes":2, "to":1, "watch":1, "movies":2, "Mary":1, "too":1};  
BoW2 = {"John":1, "also":1, "likes":1, "to":1, "watch":1, "football":1, "games":1};
```

Recap: Bag of words

- With D words in the dictionary, each sample can be represented as a vector in \mathbb{R}^D
- Example:
 - With dictionary ($D = 9$)

"John", "likes", "watch", "movies", "Mary", "too", "also", "football", "games"

- the BoWs

```
BoW1 = {"John":1, "likes":2, "to":1, "watch":1, "movies":2, "Mary":1, "too":1};  
BoW2 = {"John":1, "also":1, "likes":1, "to":1, "watch":1, "football":1, "games":1};
```

- become

$$\tilde{\mathbf{x}}_1 = [1 \ 2 \ 1 \ 2 \ 1 \ 1 \ 0 \ 0 \ 0]$$
$$\tilde{\mathbf{x}}_2 = [1 \ 1 \ 1 \ 0 \ 0 \ 0 \ 1 \ 1 \ 1]$$

Recap: Histograms

- After normalization, the final BoW representation

$$\mathbf{x}_1 = [1/8 \ 1/4 \ 1/8 \ 1/4 \ 1/8 \ 1/8 \ 0 \ 0 \ 0]$$

- is called a **histogram**

- It is such that

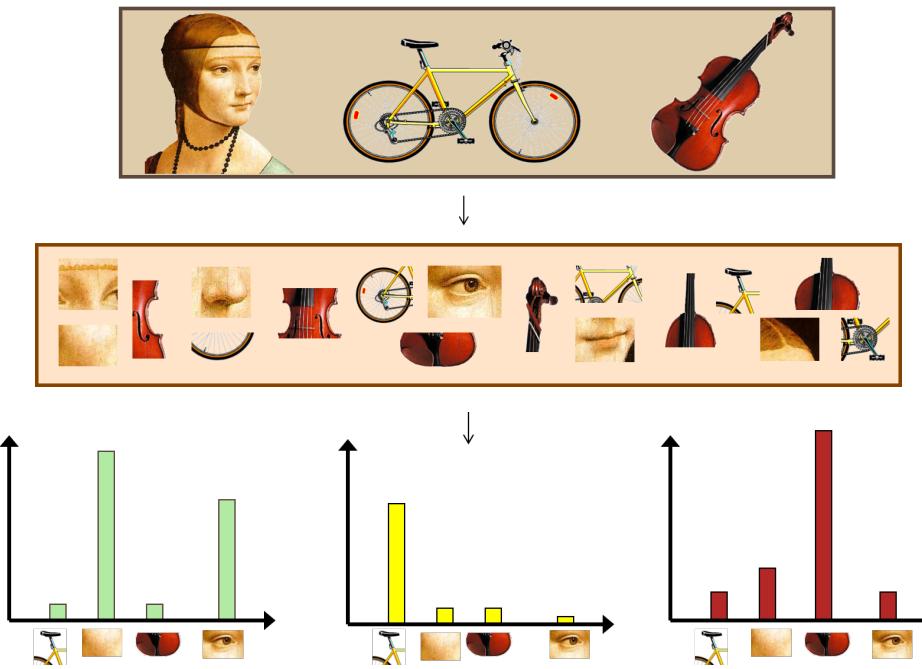
$$x_1^{(d)} \in [0,1]$$

$$\sum_{d=1}^D x_1^{(d)} = 1$$

- In this way, each sample is represented with a vector in \mathbb{R}^D

Recap: Bag of visual words

- For each patch in an image
 - find the nearest visual word
 - add one to the corresponding element in a BoW vector
- Normalize the BoW vectors to obtain histograms



Exercise: Solution

- We have seen that text and images can be represented as Bags of Words. What other type of data can you represent in this way? How would you do it?
 - For example, sound/speech. One can build a dictionary by breaking the training samples into short sound snippets and clustering them based on a notion of similarity. Then, each training sample can be expressed as a bag of audio word by associating each snippet to one cluster.

Recap: Data pre-processing

- Even with a good representation, the data might benefit from being pre-processed
- This is because, e.g.,
 - Some attribute values are missing
 - Some attribute values are noisy
 - The feature dimensions are of incommensurate magnitudes

Recap: Missing data

- Some attribute values were not recorded
 - E.g., the Air Quality UCI dataset contains missing values, which are tagged with a -200 value
- Potential solutions
 - Ignore the corresponding sample
 - Fill in the missing values with a global constant
 - Fill in the missing values with the corresponding attribute average over the dataset
 - Fill in the missing values with the corresponding attribute average over the samples belonging to the same class

Recap: Noisy data

- **Noise:**
 - Random measurement error
- **Incorrect values**
 - Faulty data collection
 - Data entry problems
 - Data transmission problems
- **Other problems**
 - Duplicate entries
 - Inconsistent data
- **Solution: Clean the data**

Recap: Cleaning noisy data

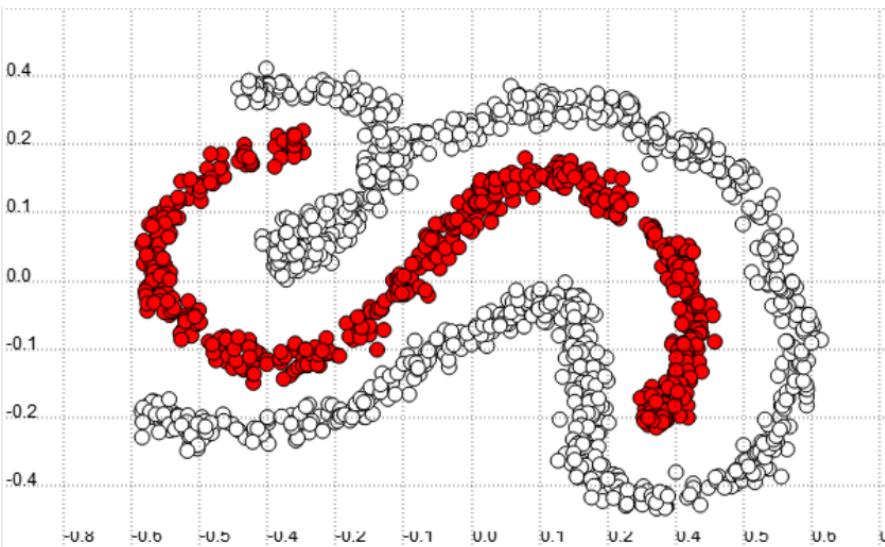
- **Binning**
 - Partition the data into bins and smooth it by replacing the data values with the mean or median of their respective bin
- **Clustering**
 - Detect outliers as the clusters with too few elements and remove these samples
- What other solution: **Leverage human knowledge**
 - Some values are not possible for certain attributes. E.g.,
 - Blood pressure cannot be 0
 - Salary cannot be negative

Recap: Data normalization

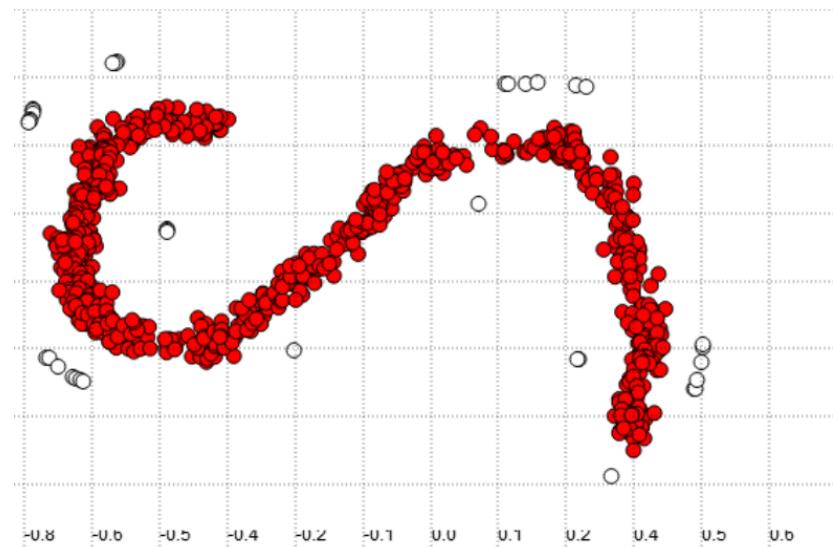
- Goal: Scale each individual attribute/feature dimension to fall within a specified range
- Min-max normalization
- z-score normalization
- Max normalization
- Normalization by decimal scaling

Recap: Imbalanced vs balanced data

- Between-class imbalance



Balanced data



Imbalanced data

- Two solutions:
 - Act on the data
 - Act on the empirical risk

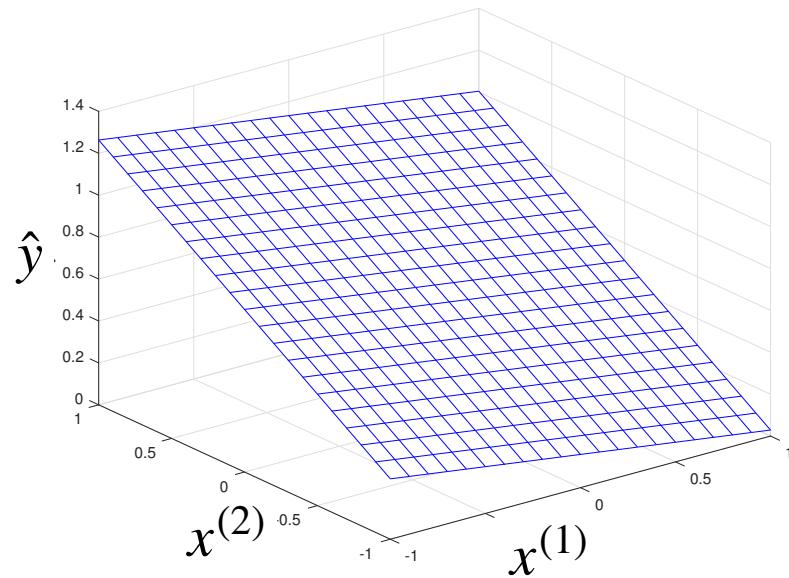
Goals of today's lecture

- Introduce the multilayer perceptron (MLP) model
- Explain the training algorithm for an MLP

Lessons learned until now

- Linear models:

$$\hat{y}(\mathbf{x}) = \mathbf{w}^T \mathbf{x}$$



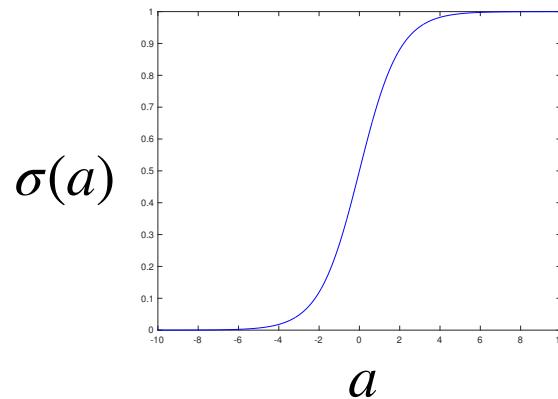
- Lesson learned: Simple to use and often effective

Lessons learned until now

- Logistic regression:

$$\hat{y}(\mathbf{x}) = \sigma(\mathbf{w}^T \mathbf{x})$$

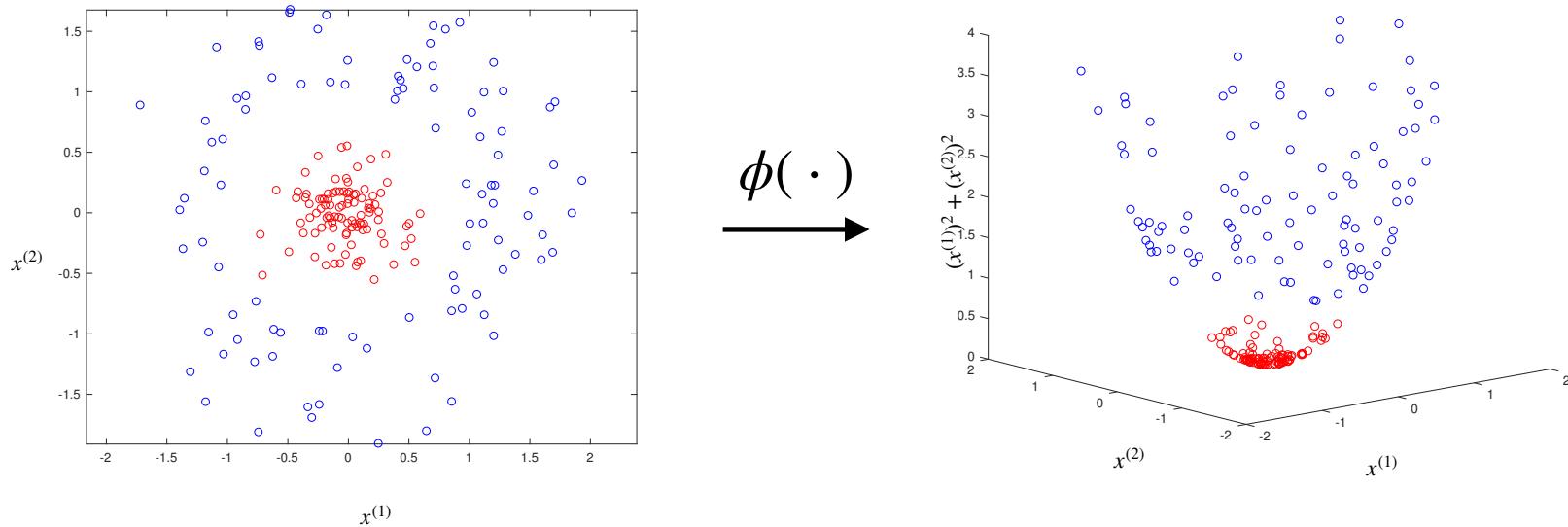
with $\sigma(\cdot)$ the sigmoid function



- Lesson learned: Nonlinearity can help

Lessons learned until now

- Kernel methods inherently map the data to a different representation



- Lesson learned: Transforming the input can help

From handcrafted representations to learned ones

- The representations and data processing strategies we have discussed so far were all manually designed
 - The transformation corresponding to a kernel is still defined manually
- There is, however, no guarantee that they are the best representations for the task at hand
- Ideally, one would therefore like to learn the representation jointly with the classifier
 - Artificial neural networks are a way to do so

Let's put all our findings together

- A simple artificial neural network

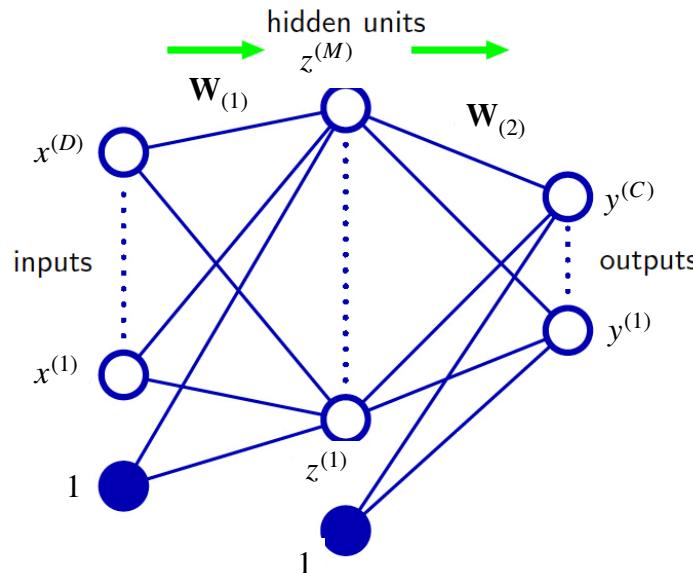
- We first map the input \mathbf{x} to a different representation (hidden units)

$$\mathbf{z} = f_{(1)}(\mathbf{W}_{(1)}^T \mathbf{x})$$

- The outputs are then obtained by another mapping

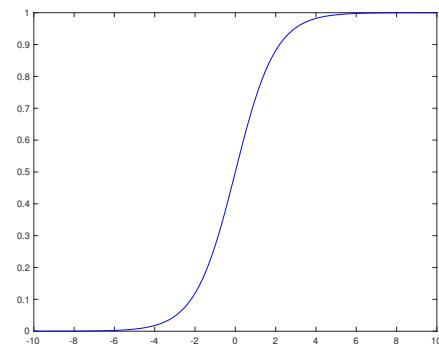
$$\mathbf{y} = f_{(2)}(\mathbf{W}_{(2)}^T \mathbf{z})$$

- Each mapping relies on a nonlinear function $f_{(j)}(\cdot)$, called *activation function*

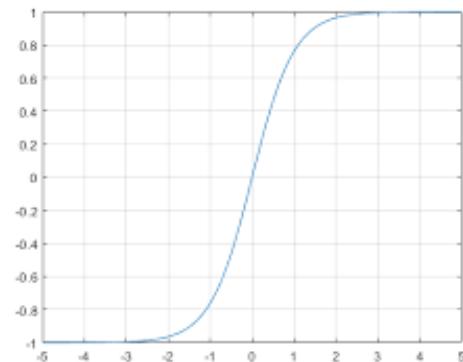


Activation function: Hidden layer

- Sigmoid: $f(a) = \frac{1}{1 + \exp(-a)}$

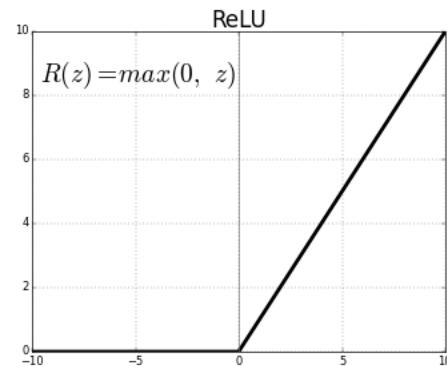


- Hyperbolic tangent: $f(a) = \tanh(a)$



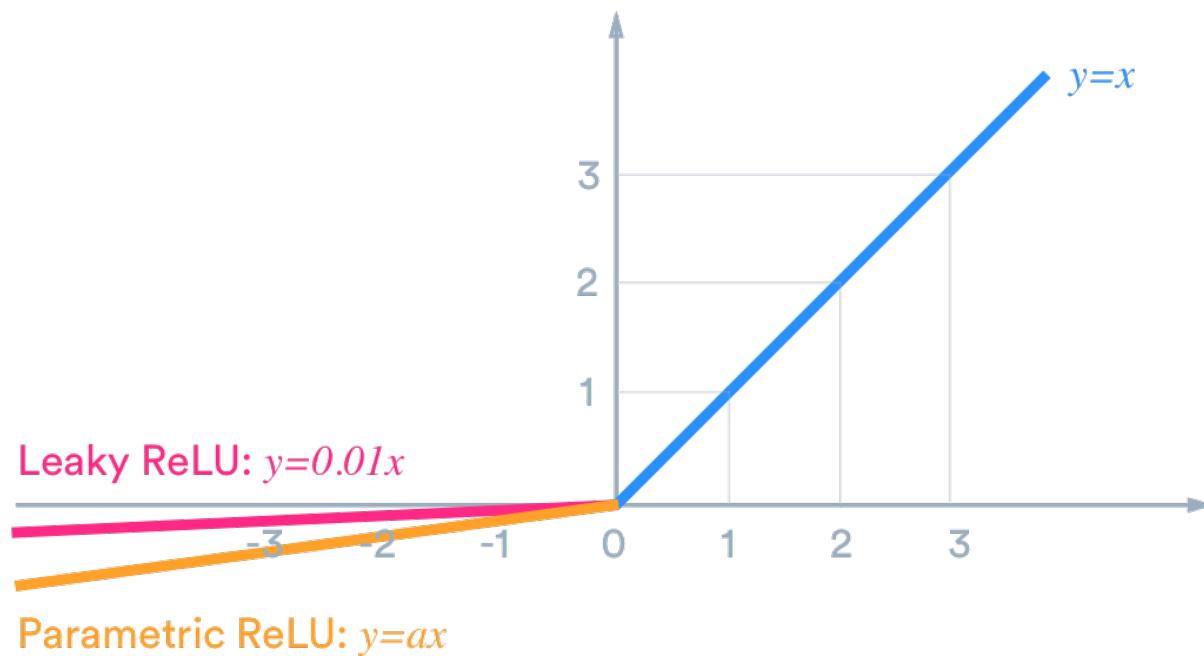
- Rectified Linear Unit (ReLU)

$$f(a) = \begin{cases} a, & \text{if } a > 0 \\ 0, & \text{otherwise} \end{cases} = \max(a, 0)$$



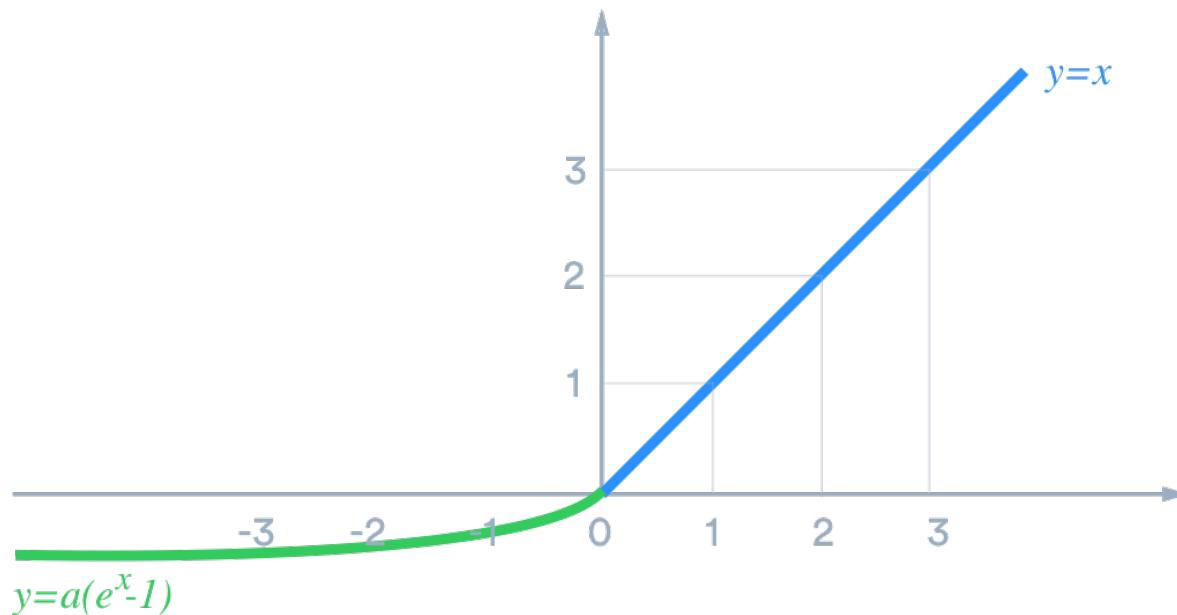
Activation function: Variants of ReLU

- Leaky ReLU & Parametric ReLU (PReLU)



Activation function: Variants of ReLU

- Exponential linear (ELU & SELU)



Activation function: Final (output) layer

- For classification, one typically relies on the softmax function (as in multi-class logistic regression)

$$y^{(k)} = \frac{\exp(\mathbf{w}_{(2)(k)}^T \mathbf{z})}{\sum_{j=1}^C \exp(\mathbf{w}_{(2)(j)}^T \mathbf{z})}$$

where $\mathbf{w}_{(2)(j)}$ indicates the j^{th} column of $\mathbf{W}_{(2)}$

- For regression, one typically relies on a linear activation function (i.e., no activation function, $\mathbf{y} = \mathbf{W}_{(2)}^T \mathbf{z}$)

1D regression

- If we consider a single output dimension and no final activation function, then, for M hidden units, the output can be written as

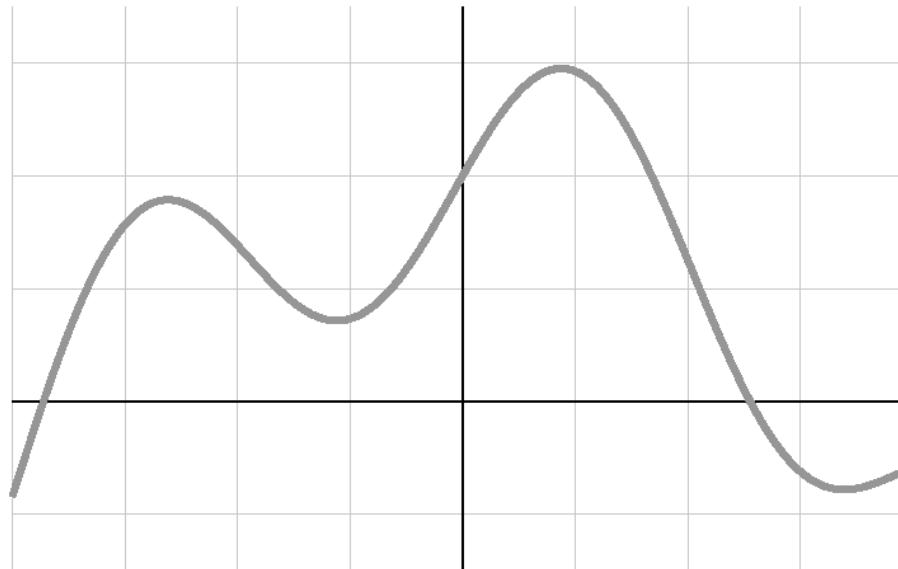
$$\hat{y} = w_{(2)(1)}f_{(1)}(\mathbf{w}_{(1)(1)}^T \mathbf{x}) + w_{(2)(2)}f_{(1)}(\mathbf{w}_{(1)(2)}^T \mathbf{x}) + \dots$$

$$= \sum_{j=1}^M w_{(2)(j)}f_{(1)}(\mathbf{w}_{(1)(j)}^T \mathbf{x})$$

- In essence, we have a simple linear combination of nonlinear functions of the input
 - How far can we go with this?

Universal approximation

- We can approximate any continuous function with a linear combination of translated/scaled ReLU functions



Universal approximation

- We can approximate any continuous function with a linear combination of translated/scaled ReLU functions $f(\cdot)$

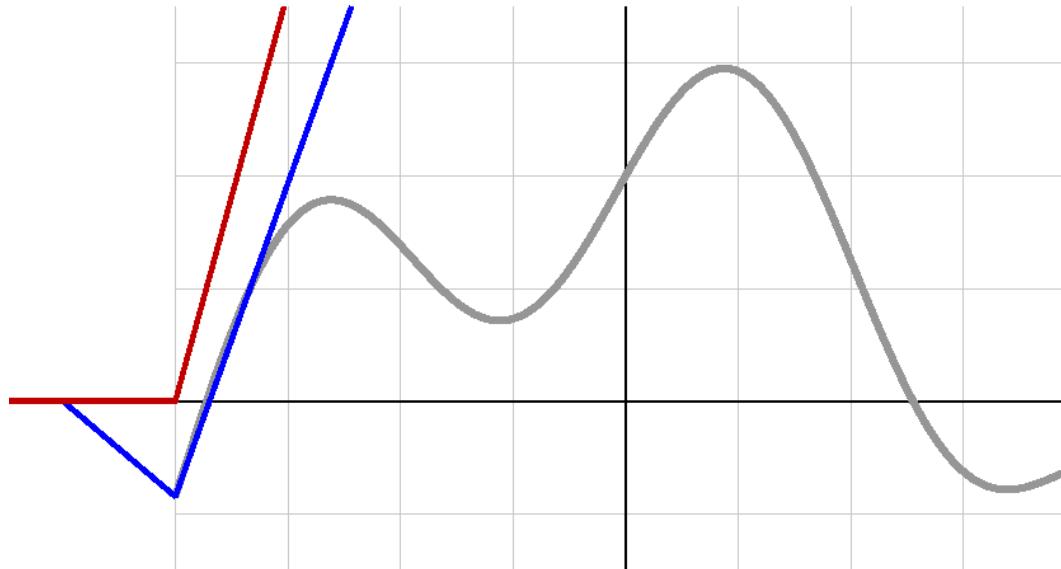
$$\hat{y} = f(w_{(1)(1)}^{(1)}x + w_{(1)(1)}^{(0)})$$



Universal approximation

- We can approximate any continuous function with a linear combination of translated/scaled ReLU functions $f(\cdot)$

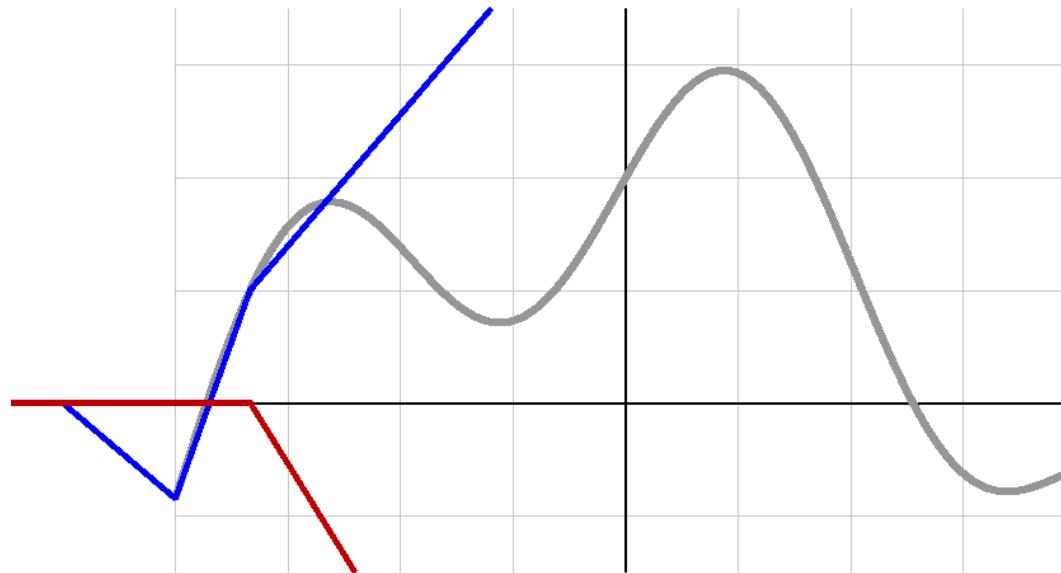
$$\hat{y} = f(w_{(1)(1)}^{(1)}x + w_{(1)(1)}^{(0)}) + f(w_{(1)(2)}^{(1)}x + w_{(1)(2)}^{(0)})$$



Universal approximation

- We can approximate any continuous function with a linear combination of translated/scaled ReLU functions $f(\cdot)$

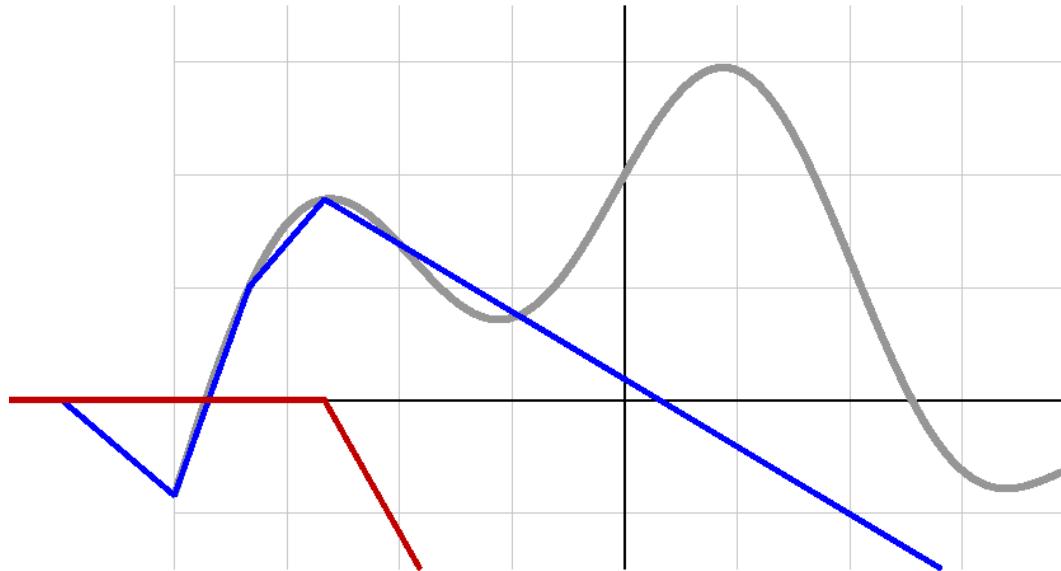
$$\hat{y} = f(w_{(1)(1)}^{(1)}x + w_{(1)(1)}^{(0)}) + f(w_{(1)(2)}^{(1)}x + w_{(1)(2)}^{(0)}) + f(w_{(1)(3)}^{(1)}x + w_{(1)(3)}^{(0)})$$



Universal approximation

- We can approximate any continuous function with a linear combination of translated/scaled ReLU functions $f(\cdot)$

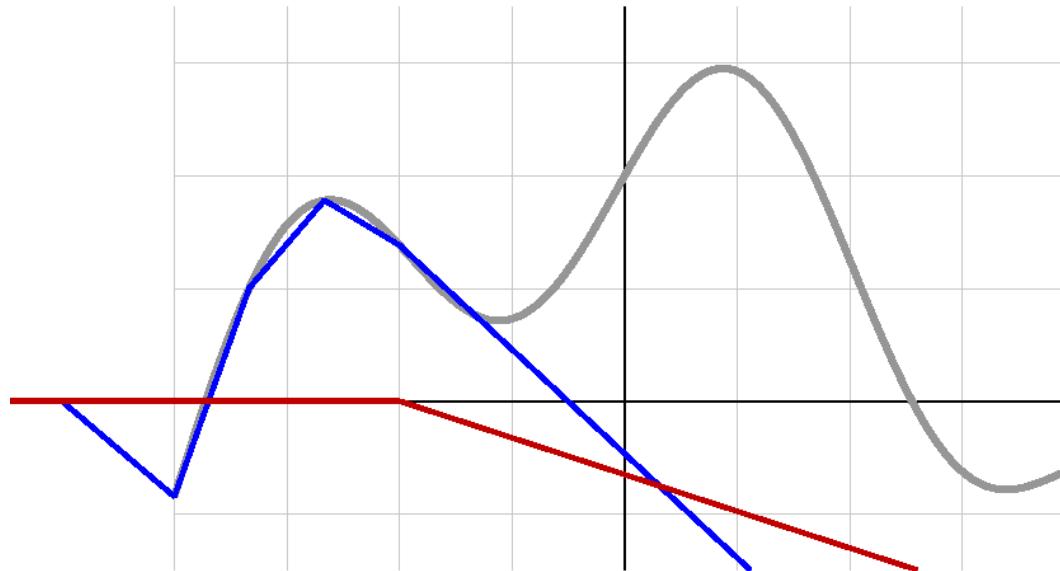
$$\hat{y} = f(w_{(1)(1)}^{(1)}x + w_{(1)(1)}^{(0)}) + f(w_{(1)(2)}^{(1)}x + w_{(1)(2)}^{(0)}) + f(w_{(1)(3)}^{(1)}x + w_{(1)(3)}^{(0)}) + \dots$$



Universal approximation

- We can approximate any continuous function with a linear combination of translated/scaled ReLU functions $f(\cdot)$

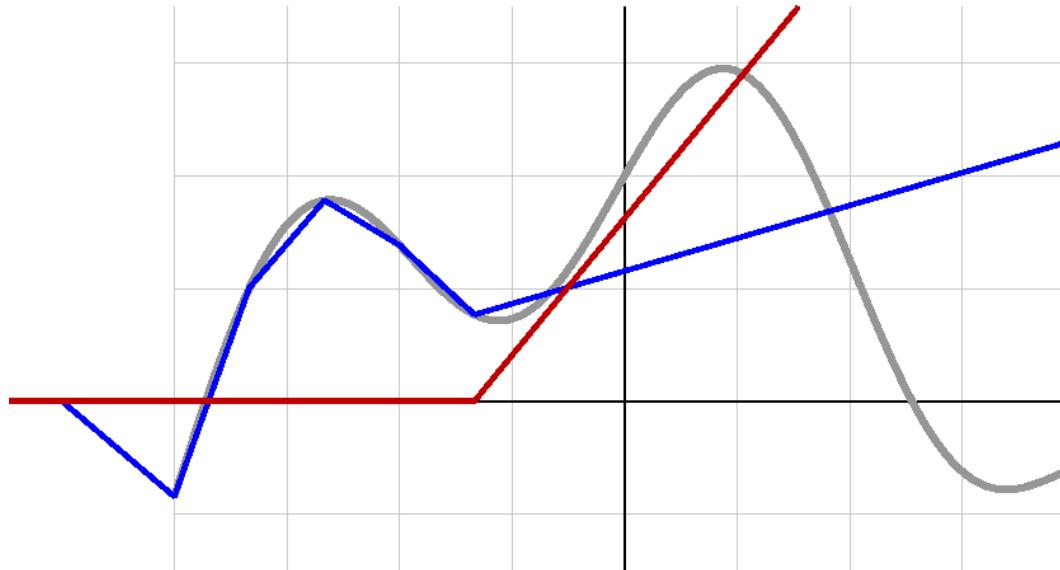
$$\hat{y} = f(w_{(1)(1)}^{(1)}x + w_{(1)(1)}^{(0)}) + f(w_{(1)(2)}^{(1)}x + w_{(1)(2)}^{(0)}) + f(w_{(1)(3)}^{(1)}x + w_{(1)(3)}^{(0)}) + \dots$$



Universal approximation

- We can approximate any continuous function with a linear combination of translated/scaled ReLU functions $f(\cdot)$

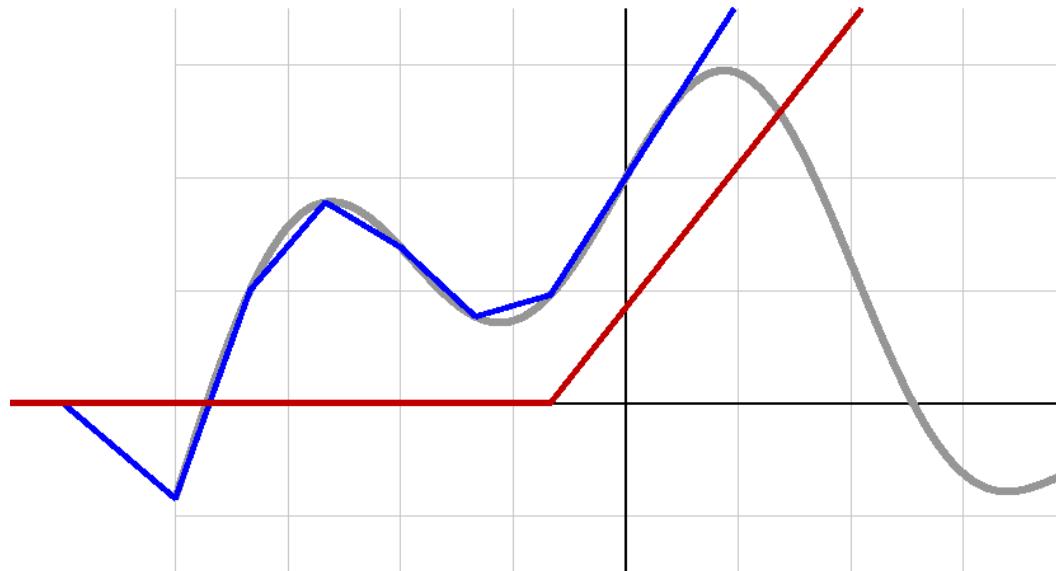
$$\hat{y} = f(w_{(1)(1)}^{(1)}x + w_{(1)(1)}^{(0)}) + f(w_{(1)(2)}^{(1)}x + w_{(1)(2)}^{(0)}) + f(w_{(1)(3)}^{(1)}x + w_{(1)(3)}^{(0)}) + \dots$$



Universal approximation

- We can approximate any continuous function with a linear combination of translated/scaled ReLU functions $f(\cdot)$

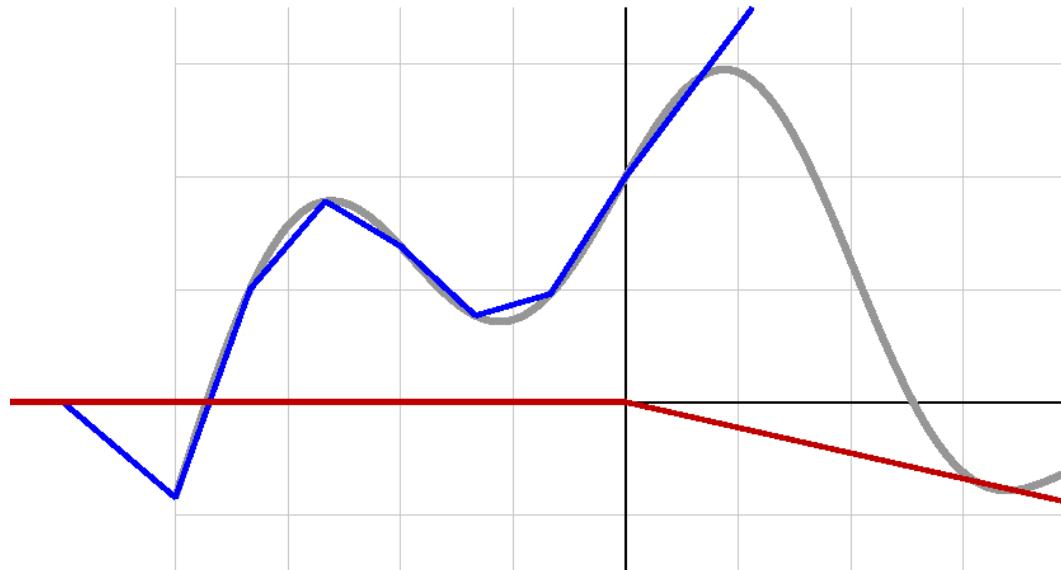
$$\hat{y} = f(w_{(1)(1)}^{(1)}x + w_{(1)(1)}^{(0)}) + f(w_{(1)(2)}^{(1)}x + w_{(1)(2)}^{(0)}) + f(w_{(1)(3)}^{(1)}x + w_{(1)(3)}^{(0)}) + \dots$$



Universal approximation

- We can approximate any continuous function with a linear combination of translated/scaled ReLU functions $f(\cdot)$

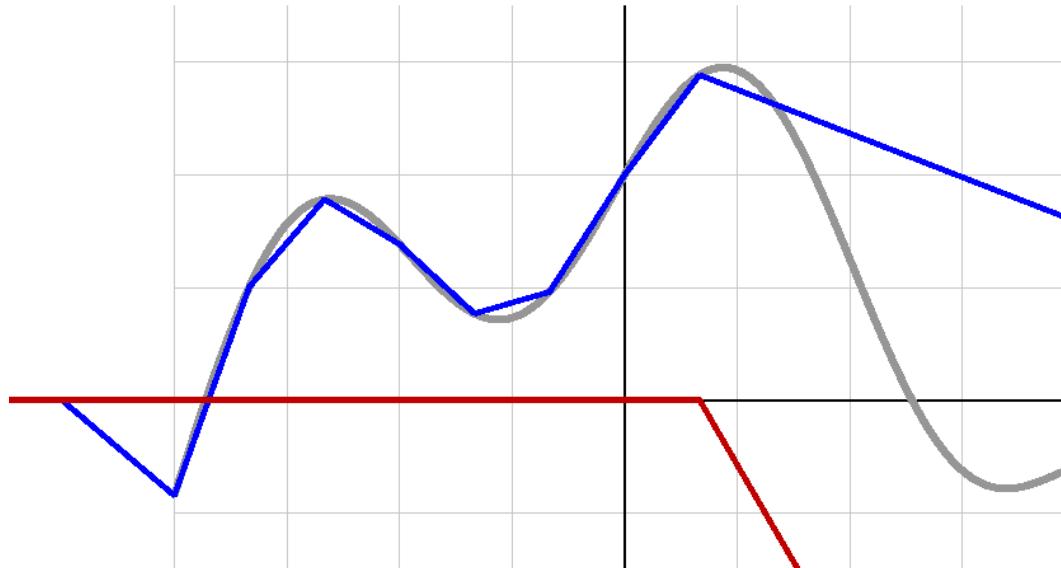
$$\hat{y} = f(w_{(1)(1)}^{(1)}x + w_{(1)(1)}^{(0)}) + f(w_{(1)(2)}^{(1)}x + w_{(1)(2)}^{(0)}) + f(w_{(1)(3)}^{(1)}x + w_{(1)(3)}^{(0)}) + \dots$$



Universal approximation

- We can approximate any continuous function with a linear combination of translated/scaled ReLU functions $f(\cdot)$

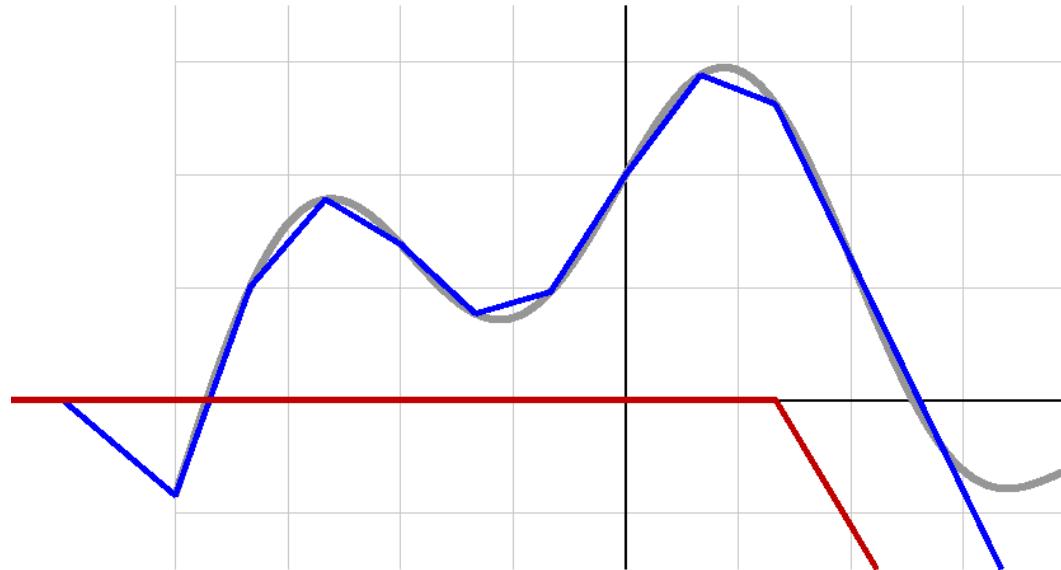
$$\hat{y} = f(w_{(1)(1)}^{(1)}x + w_{(1)(1)}^{(0)}) + f(w_{(1)(2)}^{(1)}x + w_{(1)(2)}^{(0)}) + f(w_{(1)(3)}^{(1)}x + w_{(1)(3)}^{(0)}) + \dots$$



Universal approximation

- We can approximate any continuous function with a linear combination of translated/scaled ReLU functions $f(\cdot)$

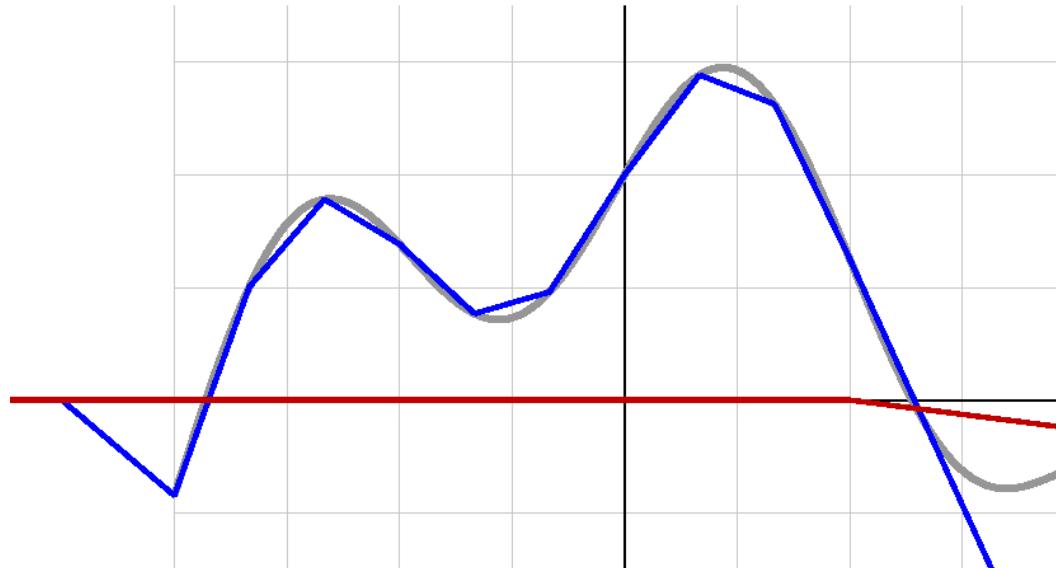
$$\hat{y} = f(w_{(1)(1)}^{(1)}x + w_{(1)(1)}^{(0)}) + f(w_{(1)(2)}^{(1)}x + w_{(1)(2)}^{(0)}) + f(w_{(1)(3)}^{(1)}x + w_{(1)(3)}^{(0)}) + \dots$$



Universal approximation

- We can approximate any continuous function with a linear combination of translated/scaled ReLU functions $f(\cdot)$

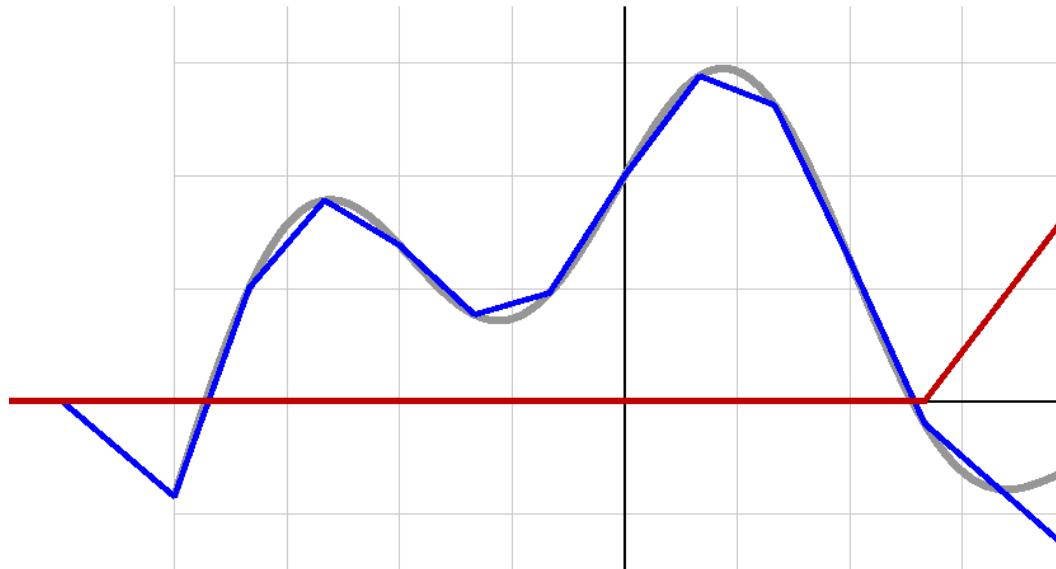
$$\hat{y} = f(w_{(1)(1)}^{(1)}x + w_{(1)(1)}^{(0)}) + f(w_{(1)(2)}^{(1)}x + w_{(1)(2)}^{(0)}) + f(w_{(1)(3)}^{(1)}x + w_{(1)(3)}^{(0)}) + \dots$$



Universal approximation

- We can approximate any continuous function with a linear combination of translated/scaled ReLU functions $f(\cdot)$

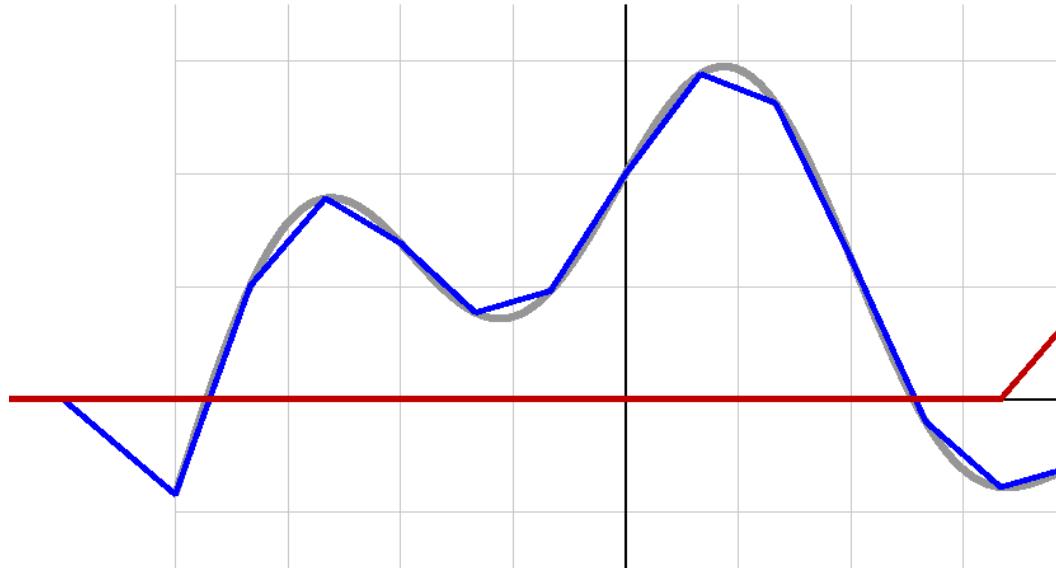
$$\hat{y} = f(w_{(1)(1)}^{(1)}x + w_{(1)(1)}^{(0)}) + f(w_{(1)(2)}^{(1)}x + w_{(1)(2)}^{(0)}) + f(w_{(1)(3)}^{(1)}x + w_{(1)(3)}^{(0)}) + \dots$$



Universal approximation

- We can approximate any continuous function with a linear combination of translated/scaled ReLU functions $f(\cdot)$

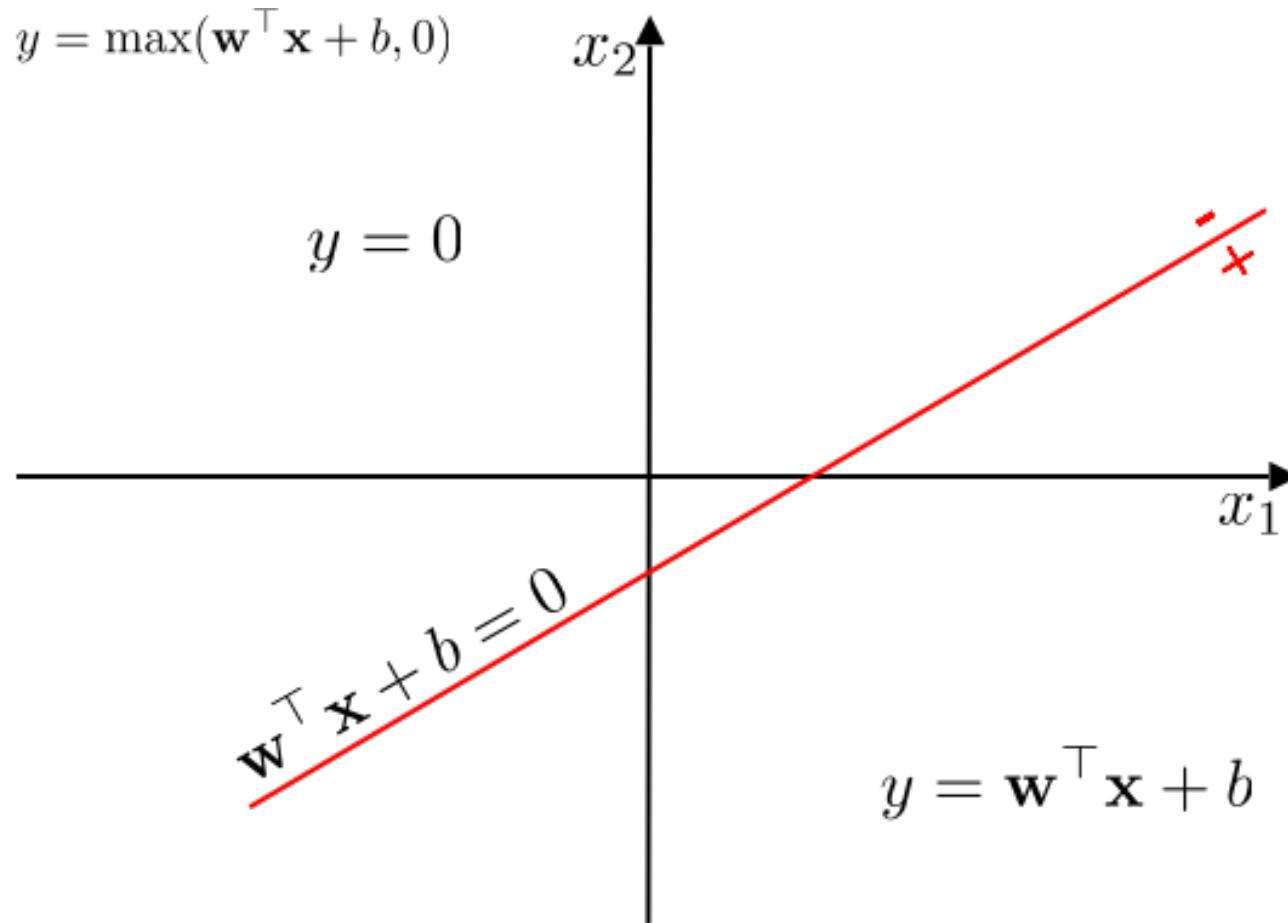
$$\hat{y} = f(w_{(1)(1)}^{(1)}x + w_{(1)(1)}^{(0)}) + f(w_{(1)(2)}^{(1)}x + w_{(1)(2)}^{(0)}) + f(w_{(1)(3)}^{(1)}x + w_{(1)(3)}^{(0)}) + \dots$$



- This is also true for other activation functions under mild assumptions

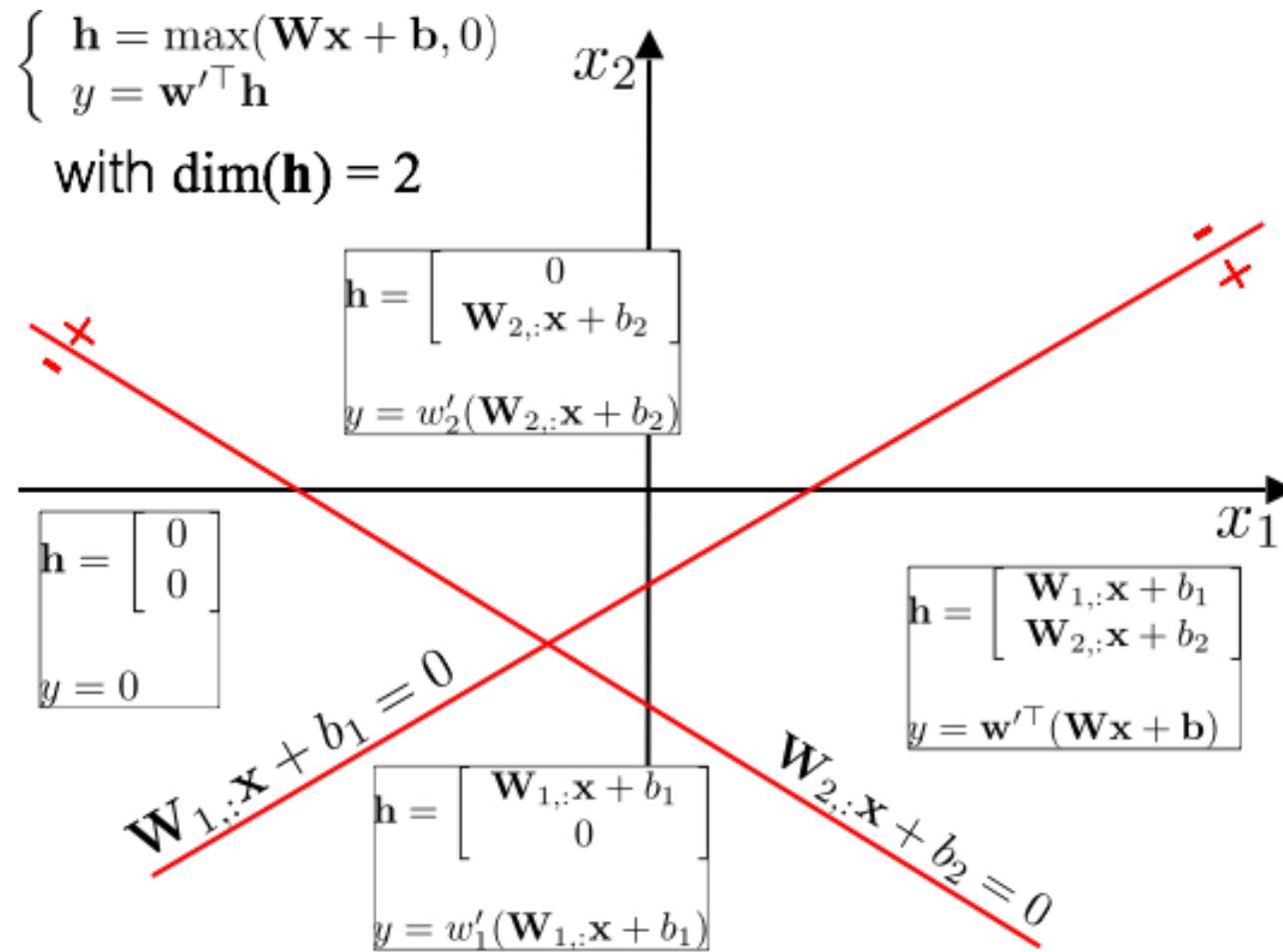
Simple network: Visualization

- Output of ReLU activation (b in the figure corresponds to $w^{(0)}$)



Simple network: Visualization

- Two hidden units with ReLU activation (\mathbf{h} corresponds to \mathbf{z} in Slide 20)

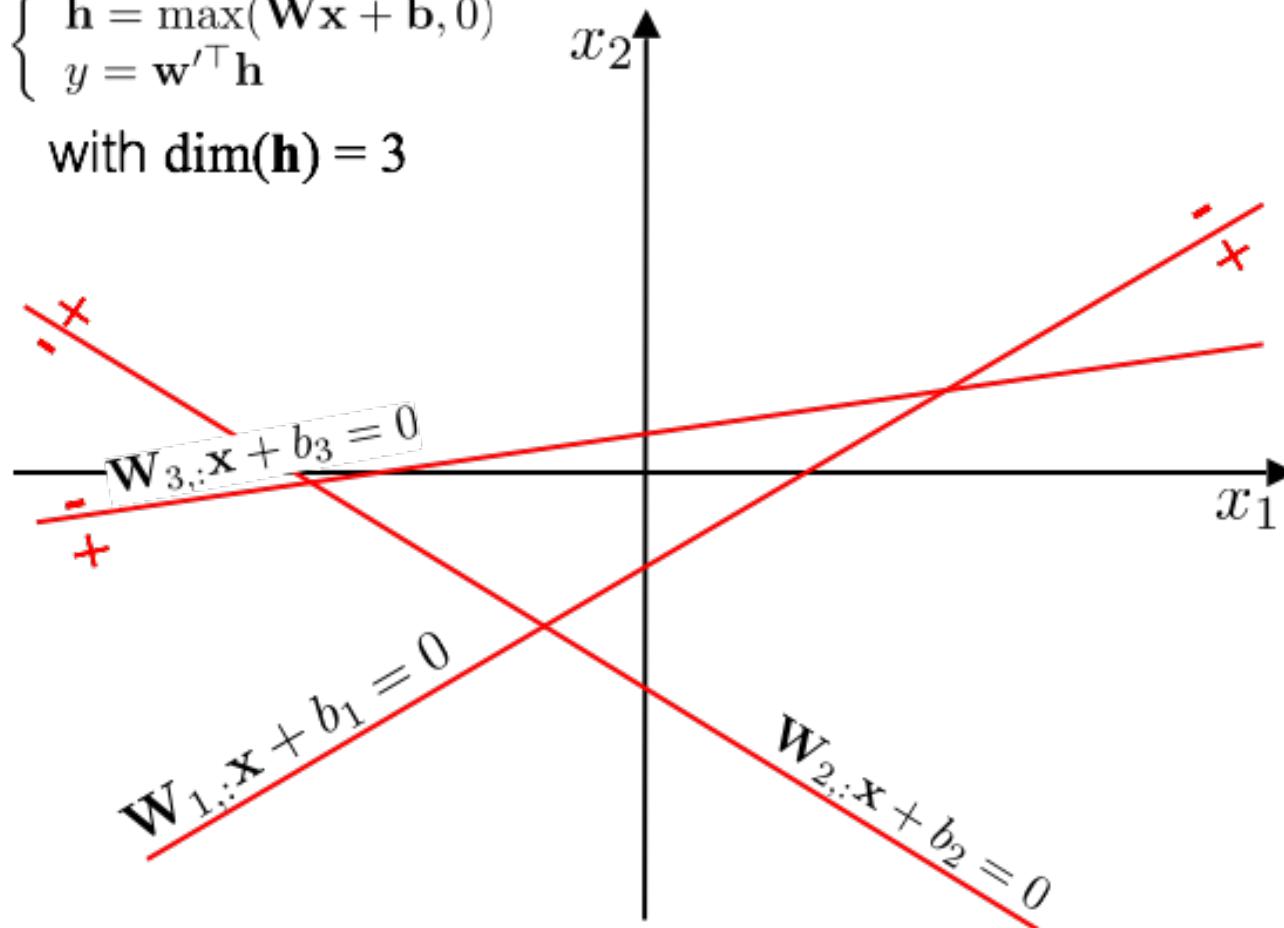


Simple network: Visualization

- Three hidden units with ReLU activation (\mathbf{h} corresponds to \mathbf{z} in Slide 20)

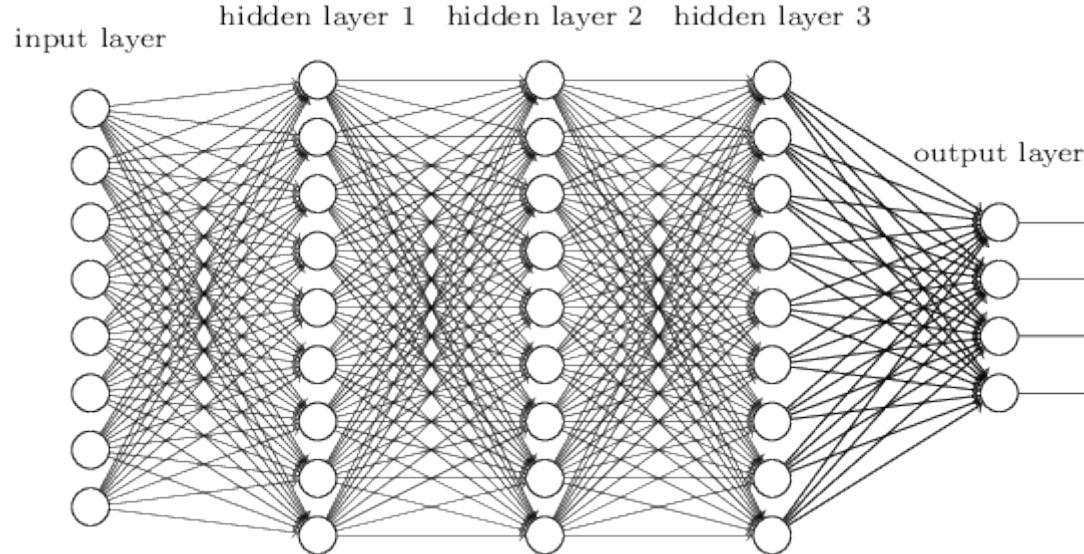
$$\begin{cases} \mathbf{h} = \max(\mathbf{W}\mathbf{x} + \mathbf{b}, 0) \\ y = \mathbf{w}'^\top \mathbf{h} \end{cases}$$

with $\dim(\mathbf{h}) = 3$



Multilayer Perceptron

- With a single hidden layer, one can already model complicated functions
- However, why stop at just one hidden layer?



- Slightly misleading name:
 - The **perceptron** uses a step function as activation function
 - **MLPs typically rely on continuous functions** (e.g., sigmoid, ReLU)

Multilayer Perceptron

- In essence, each hidden layer takes as input the output of the previous layer
- For the first hidden layer, we still have

$$\mathbf{z}_{(1)} = f_{(1)}(\mathbf{W}_{(1)}^T \mathbf{x})$$

- For any subsequent layer l , we have

$$\mathbf{z}_{(l)} = f_{(l)}(\mathbf{W}_{(l)}^T \mathbf{z}_{l-1})$$

- Finally, for a network of L layers, we compute the output as

$$\mathbf{y} = f_{(L)}(\mathbf{W}_{(L)}^T \mathbf{z}_{L-1})$$

- The process of going from the input to the output is called the *forward pass* of the network

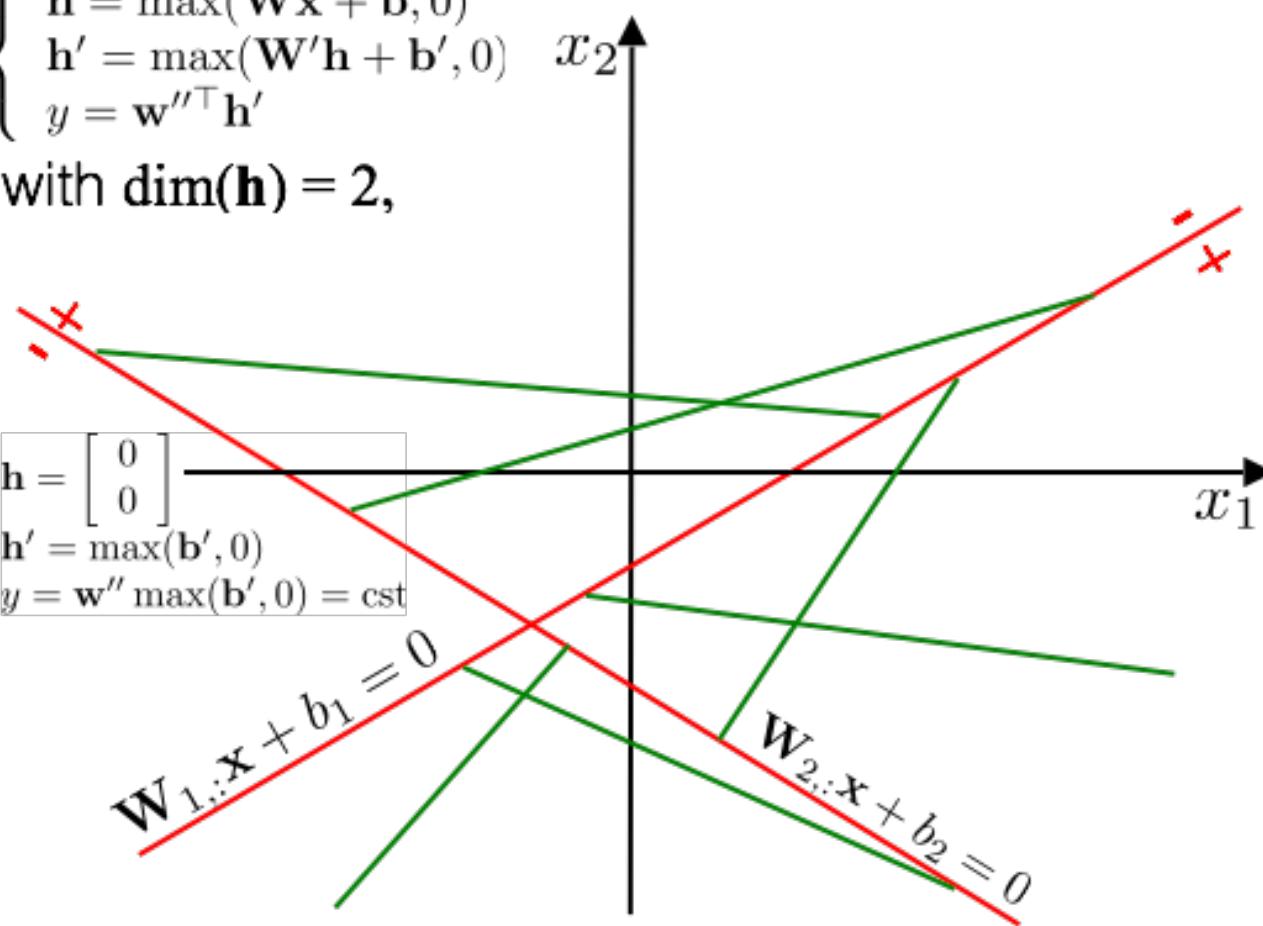
Two-layer network: Visualization

- Two layers with two units each, with ReLU activation

$$\begin{cases} \mathbf{h} = \max(\mathbf{W}\mathbf{x} + \mathbf{b}, 0) \\ \mathbf{h}' = \max(\mathbf{W}'\mathbf{h} + \mathbf{b}', 0) \\ y = \mathbf{w}''^\top \mathbf{h}' \end{cases}$$

with $\dim(\mathbf{h}) = 2$,

$$\begin{aligned} \mathbf{h} &= \begin{bmatrix} 0 \\ 0 \end{bmatrix} \\ \mathbf{h}' &= \max(\mathbf{b}', 0) \\ y &= \mathbf{w}'' \max(\mathbf{b}', 0) = \text{cst} \end{aligned}$$



Training an MLP

- As usual, training is done by minimizing an empirical risk w.r.t. the parameters, here all the $\{\mathbf{W}_{(l)}\}$
- For regression, the loss function is typically the square loss

$$R(\{\mathbf{W}_{(l)}\}) = \frac{1}{N} \sum_{i=1}^N \|\hat{\mathbf{y}}_i - \mathbf{y}_i\|^2$$

- For classification, one usually relies on the cross-entropy

$$R(\{\mathbf{W}_{(l)}\}) = -\frac{1}{N} \sum_{i=1}^N \sum_{k=1}^C y_i^{(k)} \ln \hat{y}_i^{(k)}$$

Training an MLP

- Because an MLP stacks multiple layers, each of which has a nonlinear activation function, the problem does not have a closed-form solution and is not convex
- Learning is therefore done by gradient descent
 - Question: Can you think of a drawback of this strategy?
- While the gradient might seem complicated, it can in fact be computed in a nice manner
 - This is called backpropagation, which we will discuss now

Gradient-based learning

- Because the empirical risk is an average over the training samples, its gradient will also be an average of gradients
- Thus, we can focus on a single loss term $\ell(\hat{\mathbf{y}}_i, \mathbf{y}_i)$, where

$$R(\{\mathbf{W}_{(l)}\}) = \frac{1}{N} \sum_{i=1}^N \ell(\hat{\mathbf{y}}_i, \mathbf{y}_i) = \frac{1}{N} \sum_{i=1}^N \ell_i$$

- For simplicity, I will assume that all hidden layers use the same type of activation function, denoted by $f(\cdot)$
- You do not need to know all the details of the following derivation, but I would like you to have an intuitive understanding of the algorithm

Gradient-based learning

- We can write the k^{th} dimension of the output of any layer l as

$$z_{(l)}^{(k)} = f(a_{(l)}^{(k)}) = f\left(\sum_j W_{(l)}^{(k,j)} z_{(l-1)}^{(j)}\right)$$

- Following the chain rule, we have

$$\frac{\partial \ell_i}{\partial W_{(l)}^{(k,j)}} = \frac{\partial \ell_i}{\partial a_{(l)}^{(k)}} \frac{\partial a_{(l)}^{(k)}}{\partial W_{(l)}^{(k,j)}}$$

- Furthermore, based on the equation at the top, we have

$$\frac{\partial a_{(l)}^{(k)}}{\partial W_{(l)}^{(k,j)}} = z_{(l-1)}^{(j)}$$

Gradient-based learning

- Let us define

$$\delta_{(l)}^{(k)} = \frac{\partial \ell_i}{\partial a_{(l)}^{(k)}}$$

- Now, we have

$$\frac{\partial \ell_i}{\partial W_{(l)}^{(k,j)}} = \delta_{(l)}^{(k)} z_{(l-1)}^{(j)}$$

- For the last layer (Layer L), $\delta_{(L)}^{(k)}$ can be computed explicitly as the partial derivative of the loss function w.r.t. the network's output. E.g., for the square loss, we have

$$\delta_{(L)}^{(k)} = 2(\hat{y}_i^{(k)} - y_i^{(k)})$$

Gradient-based learning

- Let us now look at Layer $L - 1$
- Following the chain rule, we have

$$\delta_{(L-1)}^{(k)} = \frac{\partial \ell_i}{\partial a_{(L-1)}^{(k)}} = \sum_m \frac{\partial \ell_i}{\partial a_{(L)}^{(m)}} \frac{\partial a_{(L)}^{(m)}}{\partial a_{(L-1)}^{(k)}}$$

- Now, recall that

$$a_{(L)}^{(m)} = \mathbf{w}_{(L)(m)}^T f(\mathbf{a}_{(L-1)}) = \sum_k W_{(L)}^{(m,k)} f(a_{(L-1)}^{(k)})$$

- This means that

$$\frac{\partial a_{(L)}^{(m)}}{\partial a_{(L-1)}^{(k)}} = W_{(L)}^{(m,k)} f'(a_{(L-1)}^{(k)})$$

where $f'(\cdot)$ is the derivative of the activation function w.r.t. its argument

Backpropagation

- So we can write

$$\delta_{(L-1)}^{(k)} = \sum_m \delta_{(L)}^{(m)} \frac{\partial a_{(L)}^{(m)}}{\partial a_{(L-1)}^{(k)}} = f'(a_{(L-1)}^{(k)}) \sum_m W_{(L)}^{(m,k)} \delta_{(L)}^{(m)}$$

- This derivation is true for any layer:
 - Given $\delta_{(l+1)}$, one can automatically compute $\delta_{(l)}$
- In turn, this gives us a way to compute the gradient

$$\frac{\partial \ell_i}{\partial W_{(l)}^{(k,j)}} = \delta_{(l)}^{(k)} z_{(l-1)}^{(j)}$$

at any layer, starting from the last one

Backpropagation: Algorithm

1. Propagate \mathbf{x}_i forward through the network
2. Compute $\delta_{(L)}$ depending on the loss
3. Propagate $\delta_{(L)}$ backward to obtain each $\delta_{(l)}$
4. At each layer, compute the partial derivatives

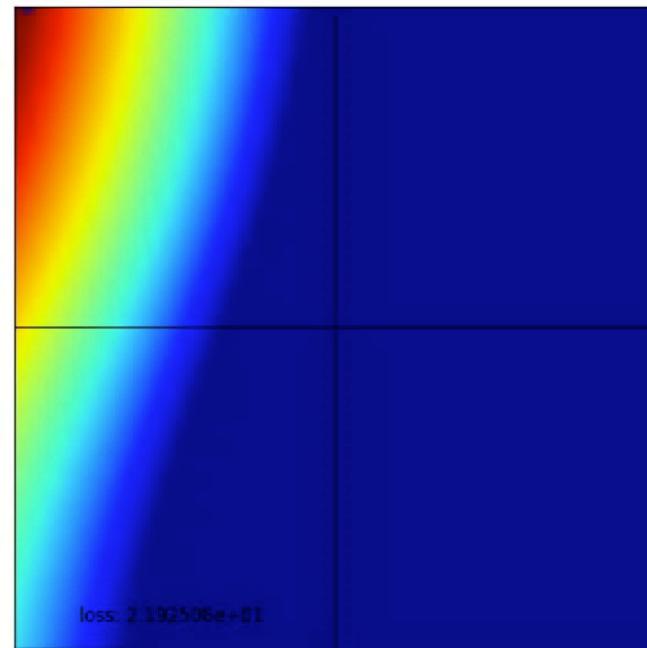
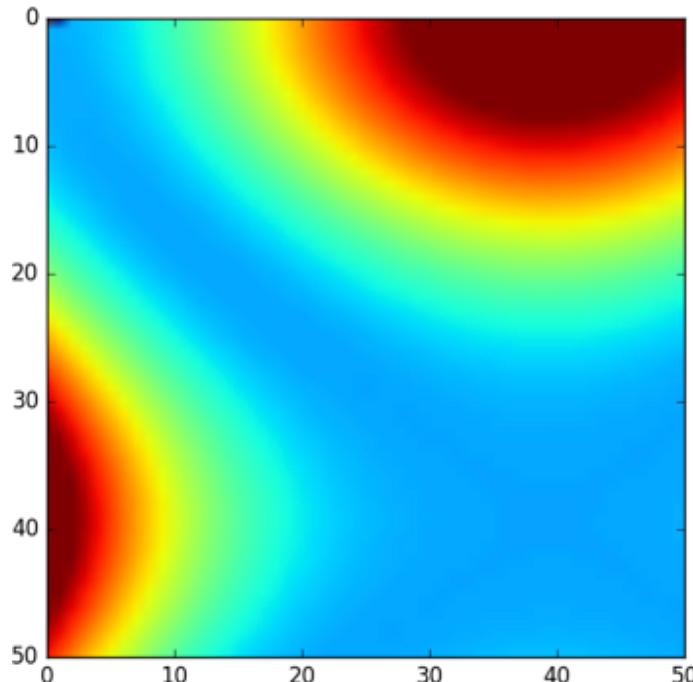
$$\frac{\partial \ell_i}{\partial W_{(l)}^{(k,j)}} = \delta_{(l)}^{(k)} z_{(l-1)}^{(j)}$$

where $\mathbf{z}_{(l-1)}$ has been computed during the forward pass

To understand

Simple network training: Regression example

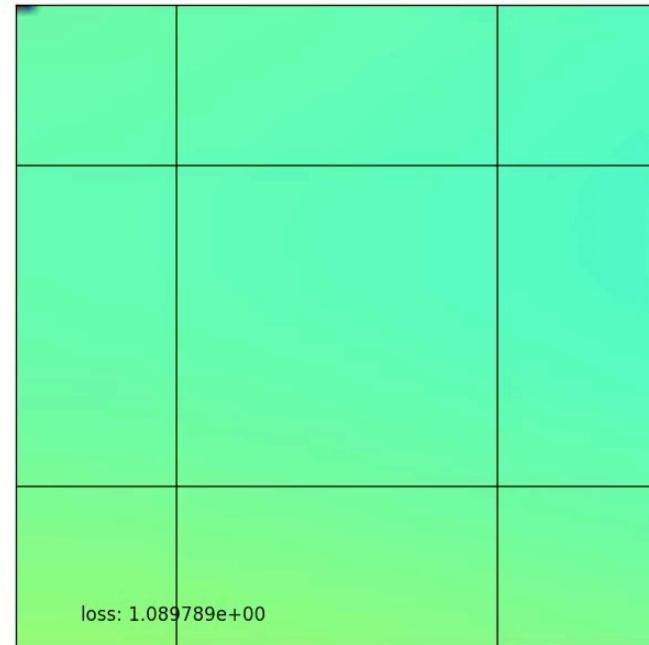
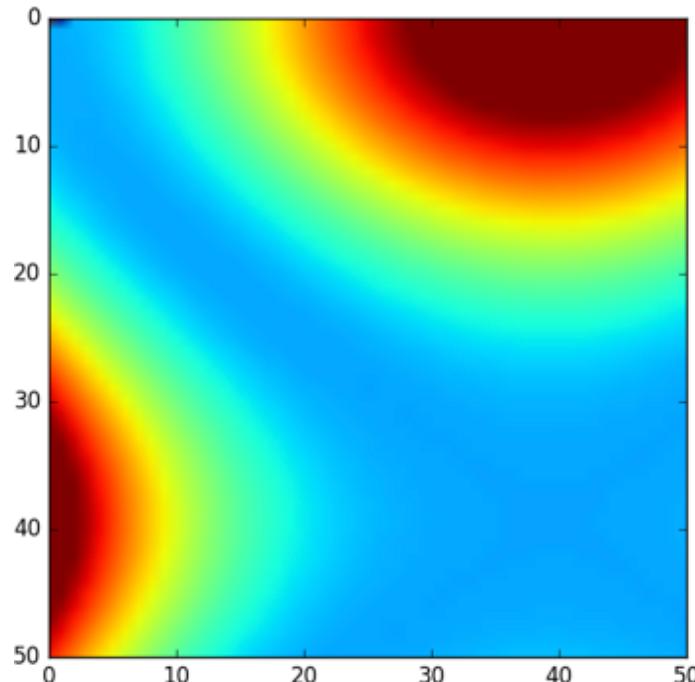
- Interpolate a surface
 - Input: 2D coordinate $(x^{(1)}, x^{(2)})$
 - Output: 1D value ($y = 100(x^{(2)} - (x^{(1)})^2)^2 + (1 - x^{(1)})^2$, shown as color)



3 hidden units

Simple network training: Regression example

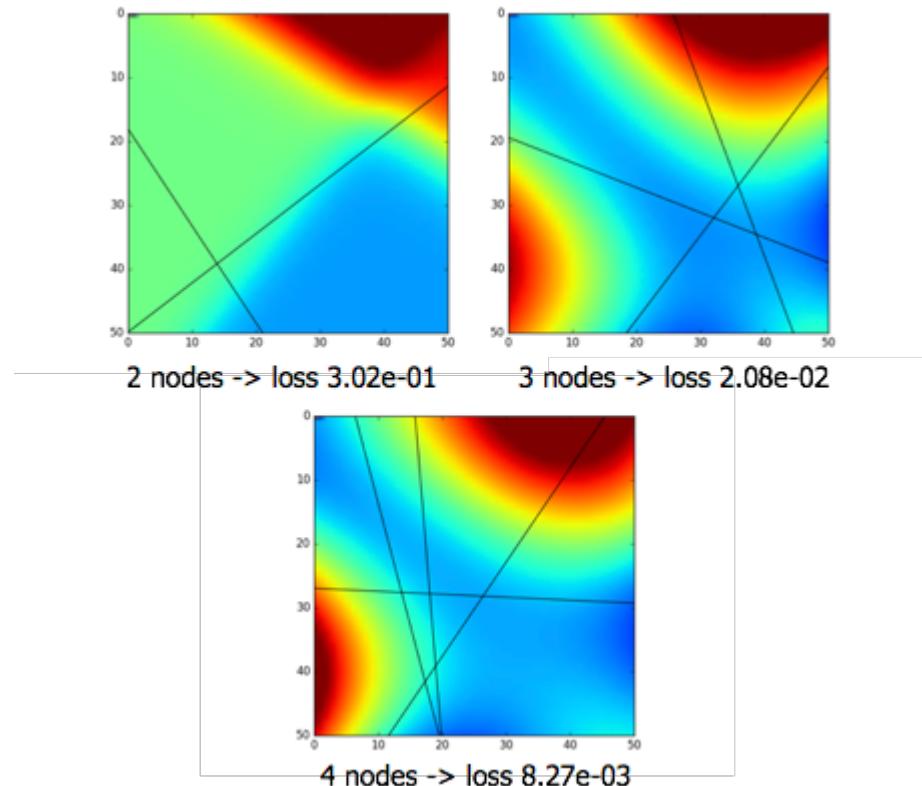
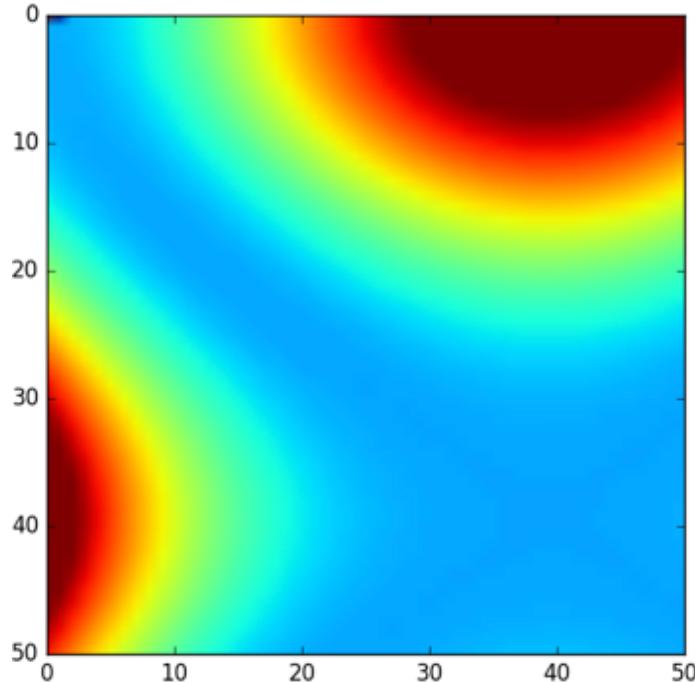
- Interpolate a surface
 - Input: 2D coordinate $(x^{(1)}, x^{(2)})$
 - Output: 1D value ($y = 100(x^{(2)} - (x^{(1)})^2)^2 + (1 - x^{(1)})^2$, shown as color)



4 hidden units

Simple network training: Regression example

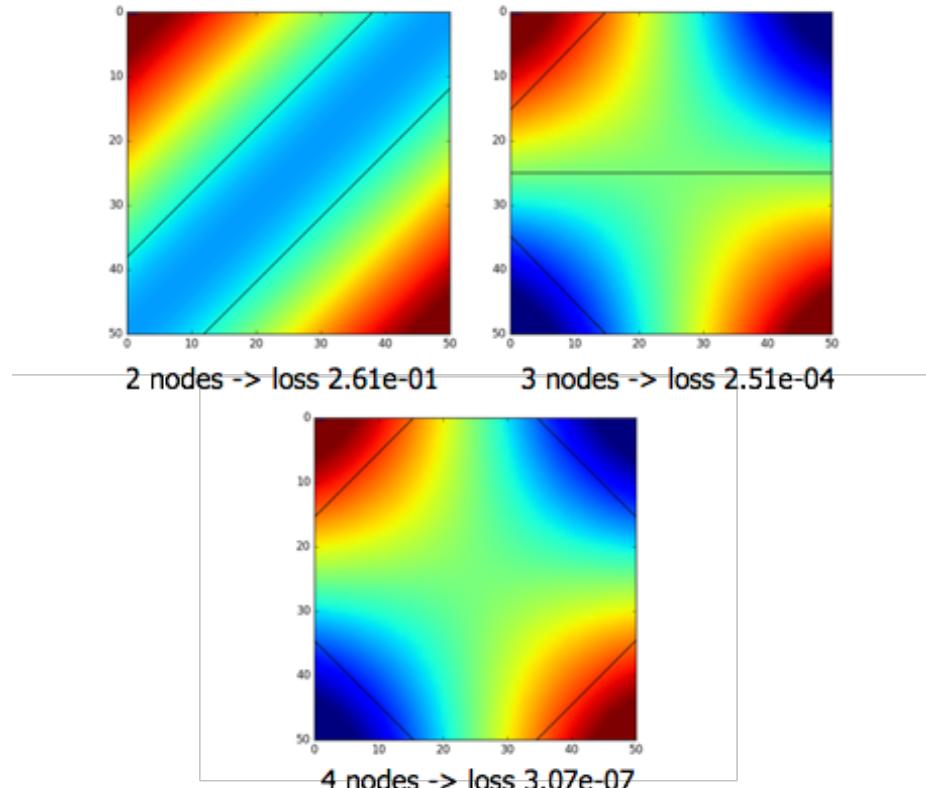
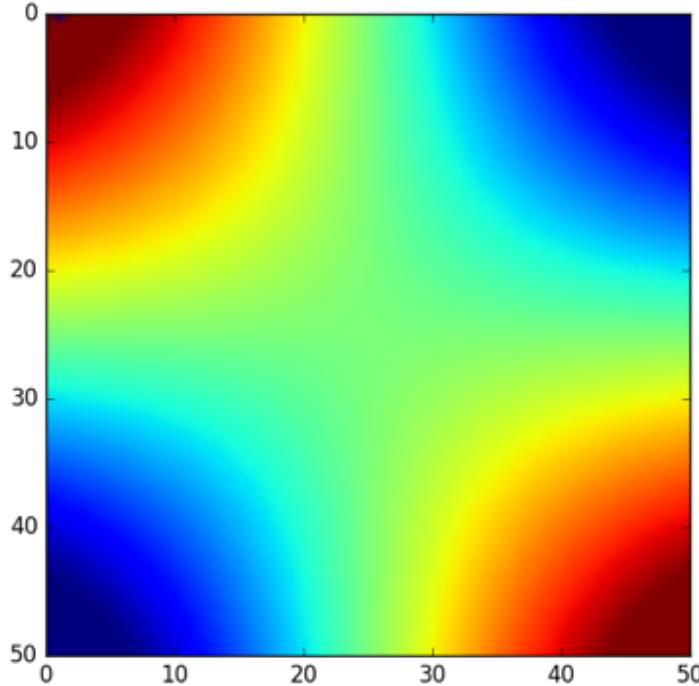
- Interpolate a surface
 - Input: 2D coordinate $(x^{(1)}, x^{(2)})$
 - Output: 1D value ($y = 100(x^{(2)} - (x^{(1)})^2)^2 + (1 - x^{(1)})^2$, shown as color)



Simple network training: Regression example

- Interpolate a surface

- Input: 2D coordinate $(x^{(1)}, x^{(2)})$
- Output: 1D value ($y = \sin(x^{(1)})\sin(x^{(2)})$, shown as color)



Gradient descent

- With standard gradient the update at iteration k is computed as

$$\mathbf{W}_k \leftarrow \mathbf{W}_{k-1} - \eta \nabla R(\mathbf{W}_{k-1})$$

where \mathbf{W} encompasses the parameters of all layers

- To compute this gradient, one needs to pass each training sample through the network and backpropagate its error
 - This becomes expensive for large training sets
 - This does not account for the incremental nature of the gradient

$$\nabla R(\mathbf{W}_{k-1}) = \sum_{i=1}^N \nabla \ell_i(\mathbf{W}_{k-1})$$

When computing $\nabla \ell_i$, we have already computed $\nabla \ell_1, \dots, \nabla \ell_{i-1}$, which we could have used to have a better estimate of \mathbf{W}

Stochastic gradient descent

- *Stochastic* gradient descent updates the parameters \mathbf{W} based on the gradient of a single sample in each iteration

$$\mathbf{W}_k \leftarrow \mathbf{W}_{k-1} - \eta \nabla \ell_{i(k)}(\mathbf{W}_{k-1})$$

At each iteration, we use the gradient of a single loss term.
The index of the sample depends on the iteration number k

- In practice, one then iterates over the training samples, either in a fixed order or in a random one

SGD with mini-batches

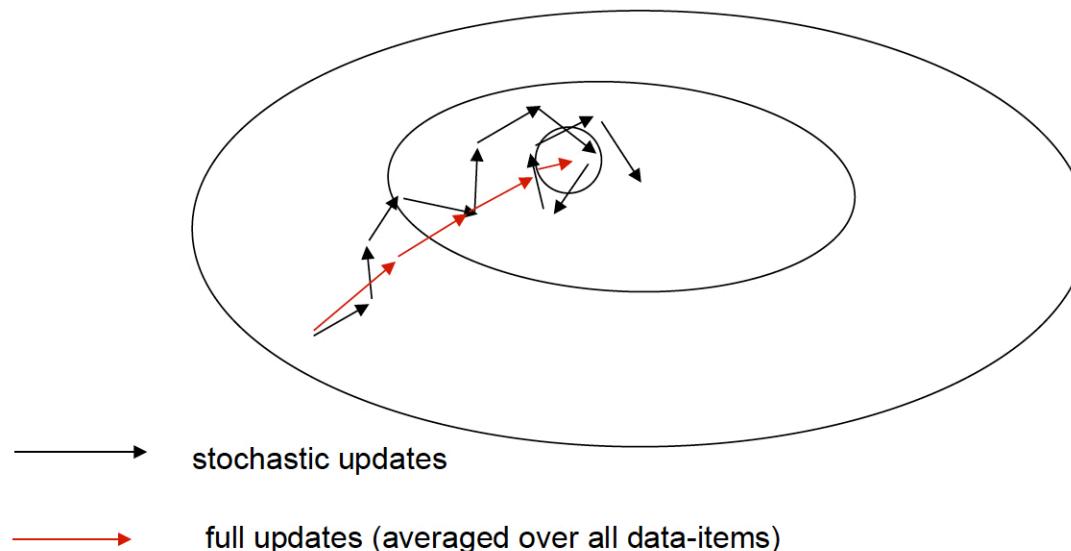
- The gradient obtained by a single sample might be a poor approximation of the true, complete gradient
 - E.g., an individual sample might benefit from the increase of one specific parameters, whereas most of the other samples would benefit from a decrease of this parameter
- To obtain a more reliable gradient estimate, one typically uses mini-batches of B samples
 - The parameters are then updated as

$$\mathbf{W}_k \leftarrow \mathbf{W}_{k-1} - \eta \sum_{b=1}^B \nabla \ell_{i(b,k)}(\mathbf{W}_{k-1})$$

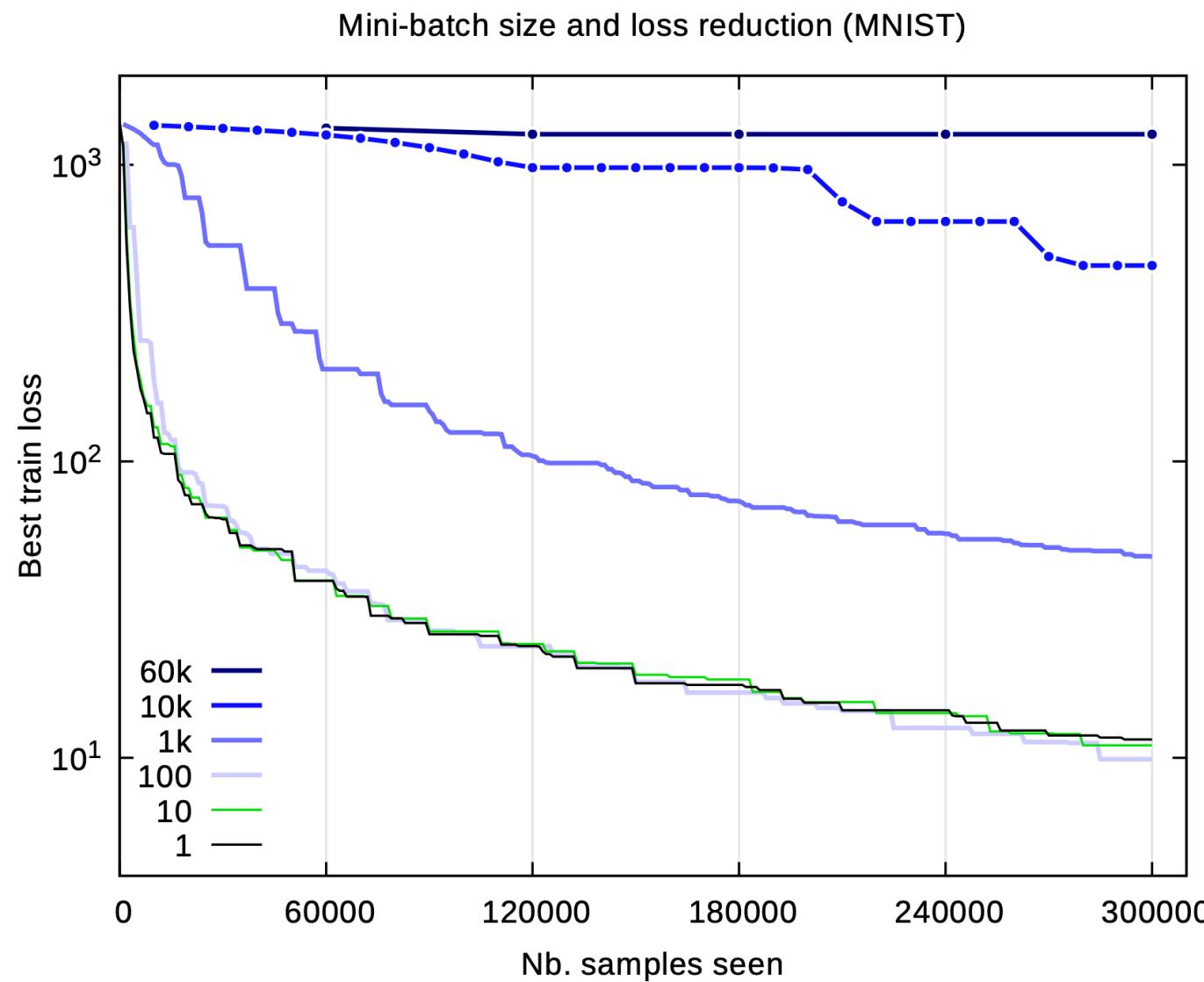
The index of the samples used for the update now depends on the iteration number k and on the index b of the samples in the mini-batch

SGD behavior

- Because it approximates the full gradient, SGD will move the parameters less directly towards a local minimum
 - The parameters dance around the minimum, which may require decreasing the learning rate
 - Nevertheless, SGD can help to avoid bad local minima

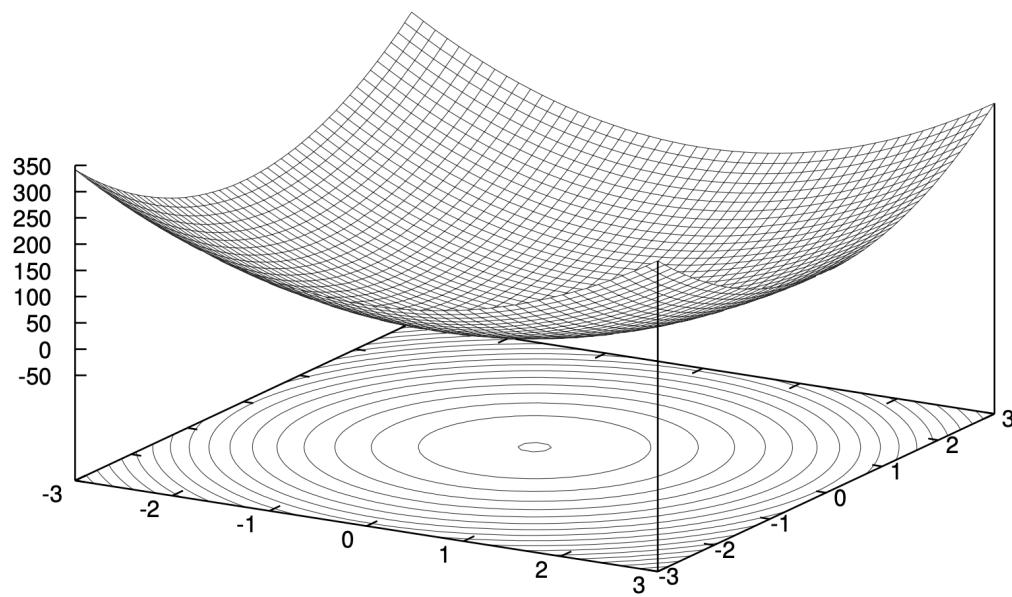


SGD behavior

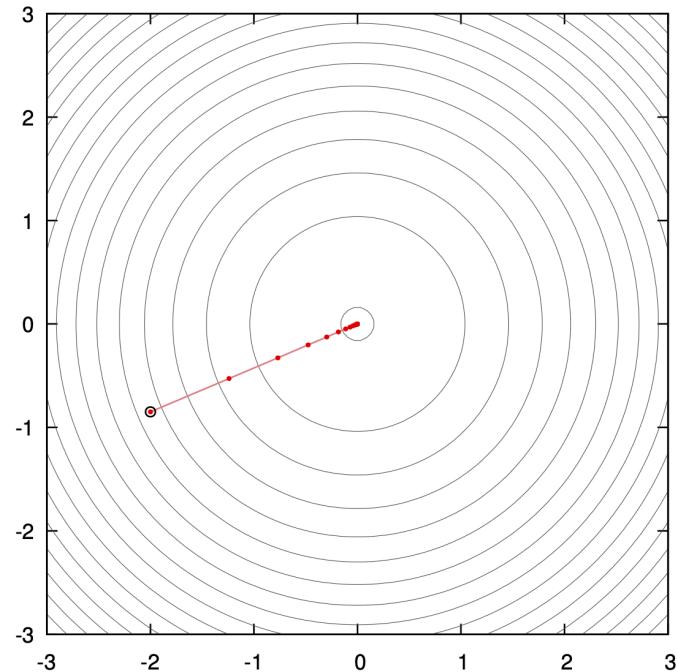


Gradient descent: Limitations

- Gradient descent makes strong assumption about the magnitude and isotropy of the local curvature so that the same step size can be used in all directions



$$\eta = 1.0e - 2$$



Gradient descent: Limitations

- Gradient descent makes strong assumption about the magnitude and isotropy of the local curvature so that the same step size can be used in all directions

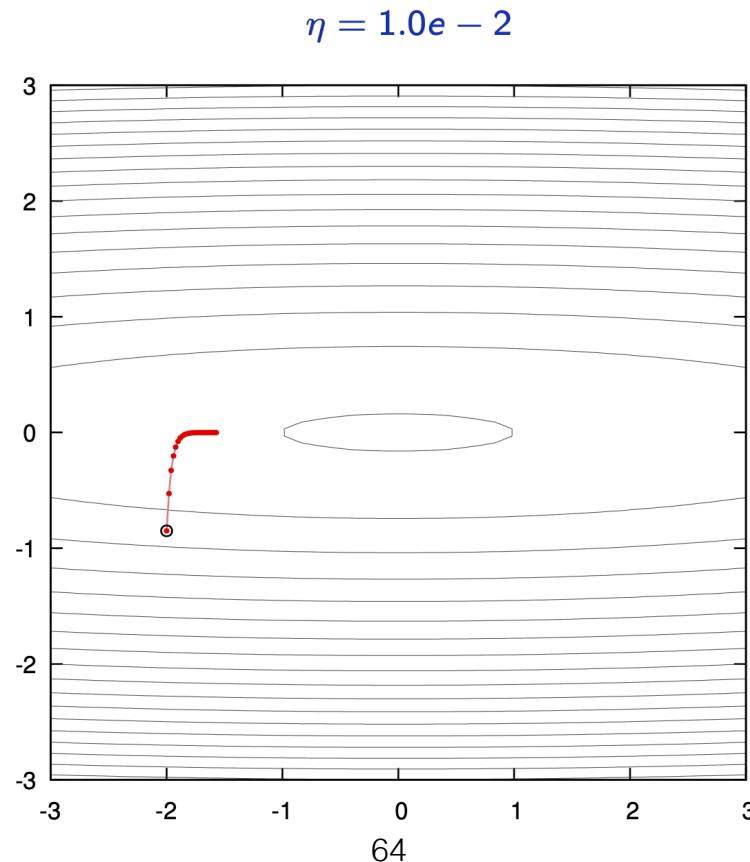


Figure by François Fleuret

Gradient descent: Limitations

- Gradient descent makes strong assumption about the magnitude and isotropy of the local curvature so that the same step size can be used in all directions

$$\eta = 2.0e - 2$$

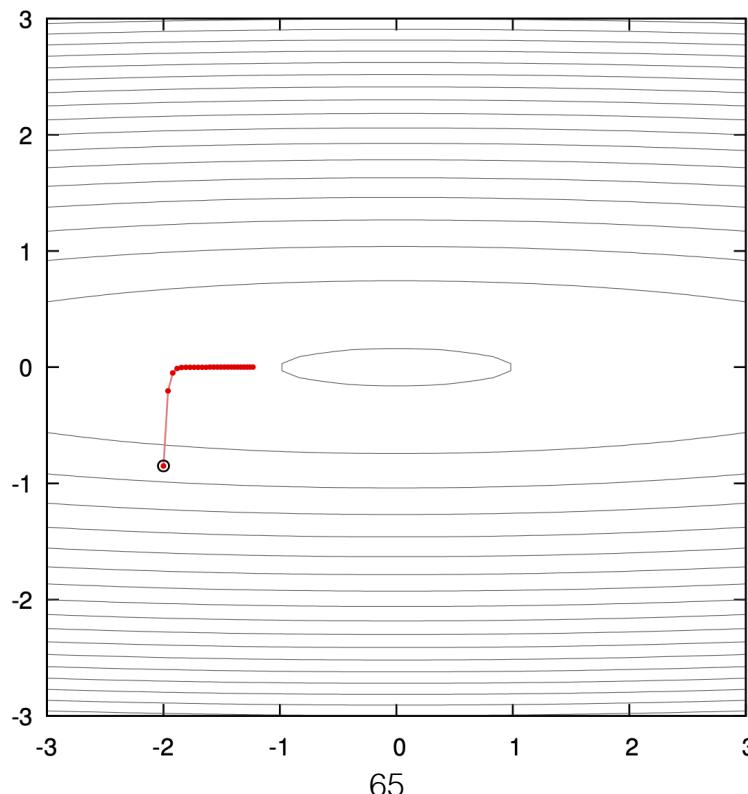


Figure by François Fleuret

Gradient descent: Limitations

- Gradient descent makes strong assumption about the magnitude and isotropy of the local curvature so that the same step size can be used in all directions

$$\eta = 4.0e - 2$$

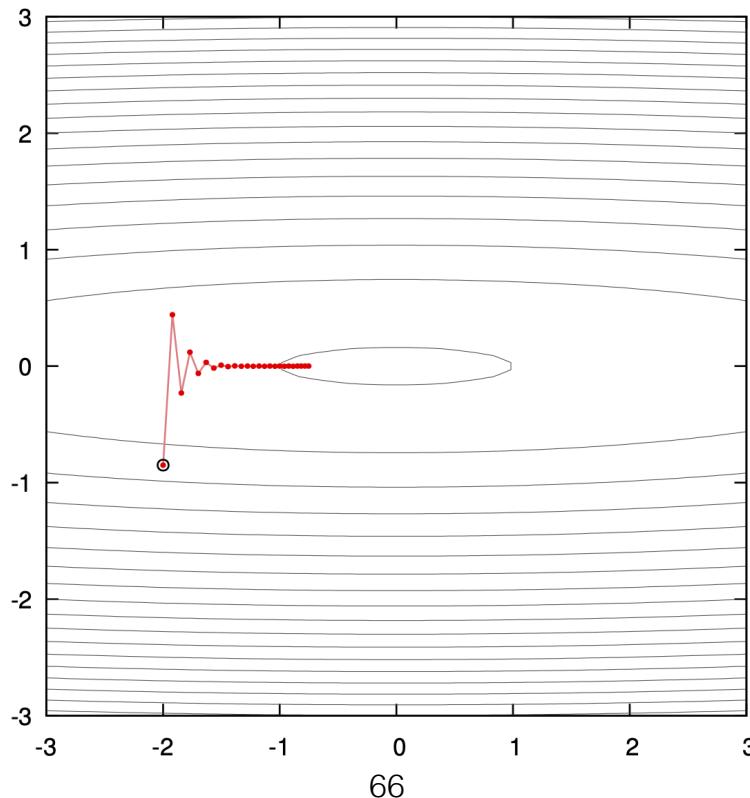
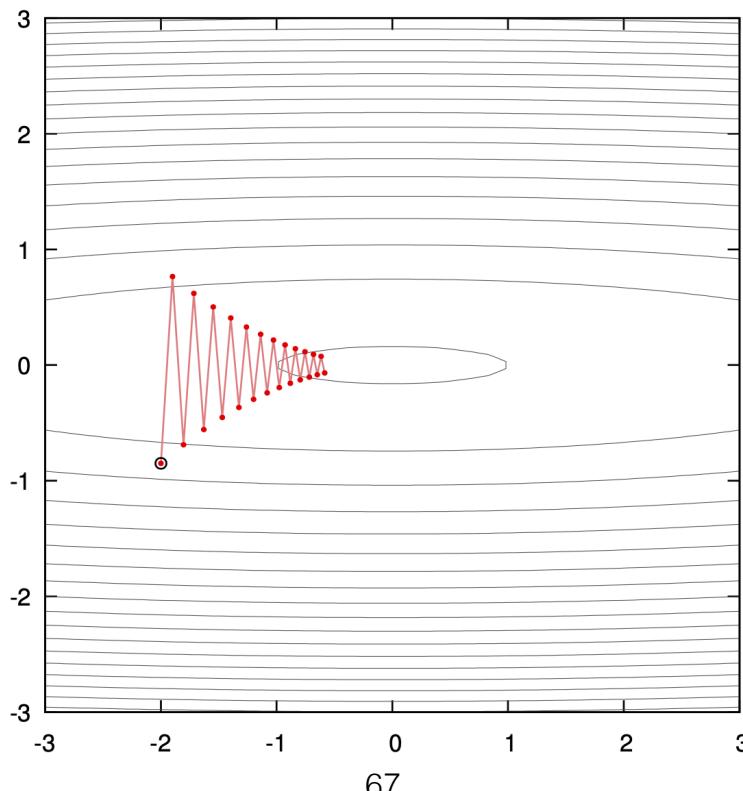


Figure by François Fleuret

Gradient descent: Limitations

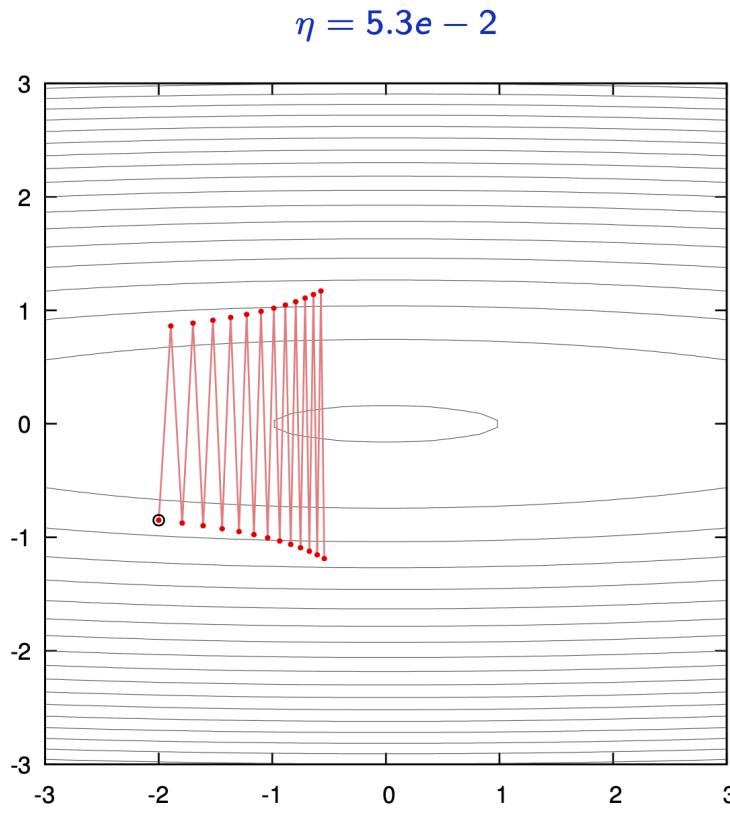
- Gradient descent makes strong assumption about the magnitude and isotropy of the local curvature so that the same step size can be used in all directions

$$\eta = 5.0e - 2$$



Gradient descent: Limitations

- Gradient descent makes strong assumption about the magnitude and isotropy of the local curvature so that the same step size can be used in all directions



SGD Extensions

- **Momentum:**

- Add inertia to the parameter updates

$$\nu \leftarrow \gamma\nu + \eta \nabla R(\mathbf{W})$$

$$\mathbf{W} \leftarrow \mathbf{W} - \nu$$

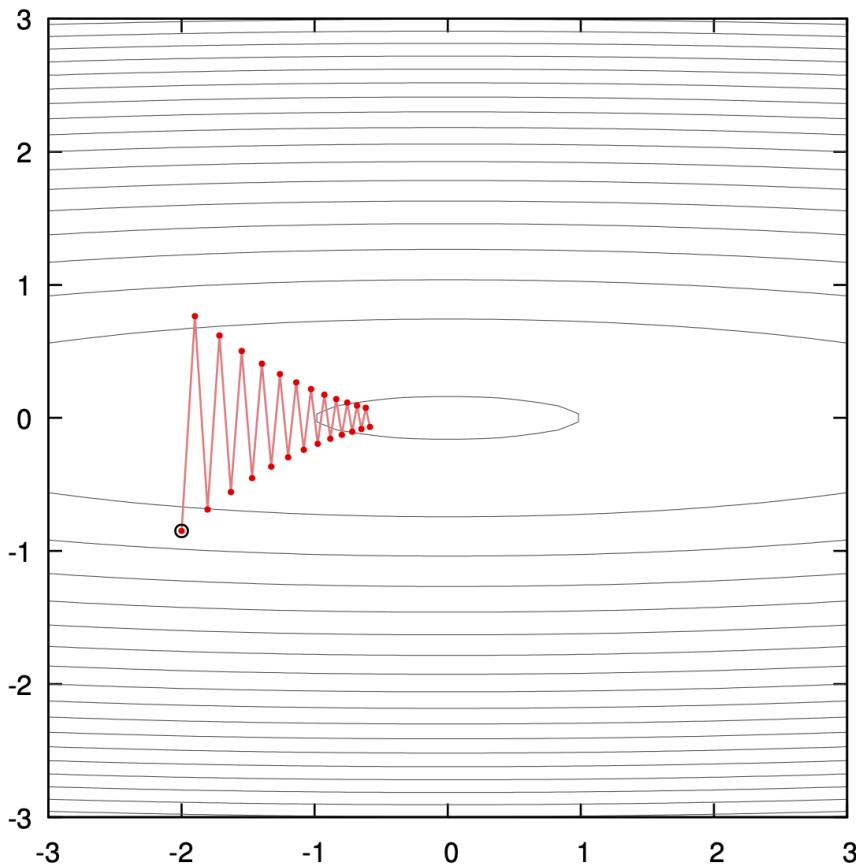
- With $\gamma > 0$, momentum

- Accelerates if the gradient does not change too much
- Dampens the oscillations in narrow valleys

Momentum

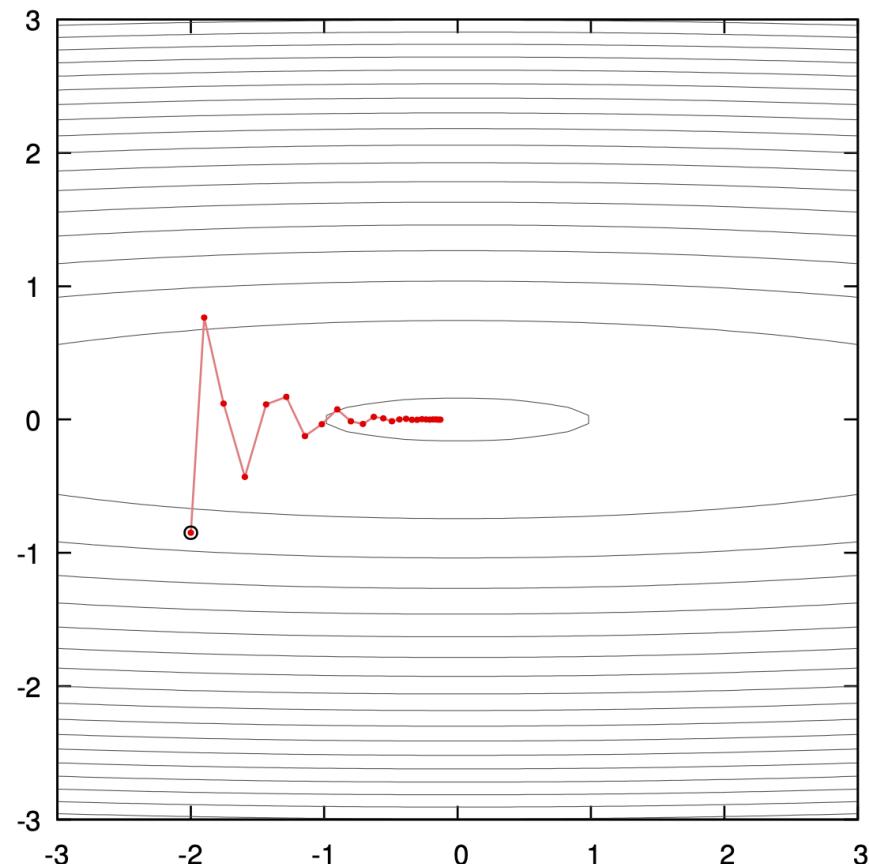
Without momentum

$$\eta = 5.0e - 2, \gamma = 0$$



With momentum

$$\eta = 5.0e - 2, \gamma = 0.5$$



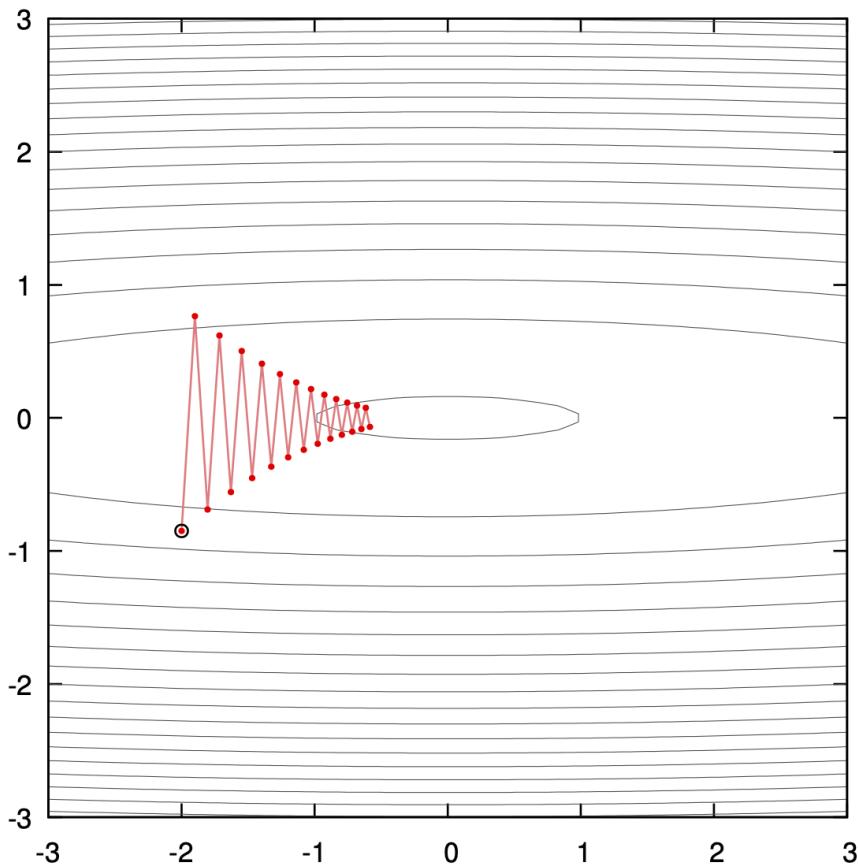
SGD Extensions

- Adaptive learning rate:
 - Adagrad (Duchi et al., JMLR 2011)
 - Adadelta (Zeiler, 2012)
 - RMSprop (Hinton, 2012)
 - ADAM (Kingma et al., ICLR 2015)
- For example, ADAM uses a moving average of each coordinate to rescale each coordinate separately

Momentum

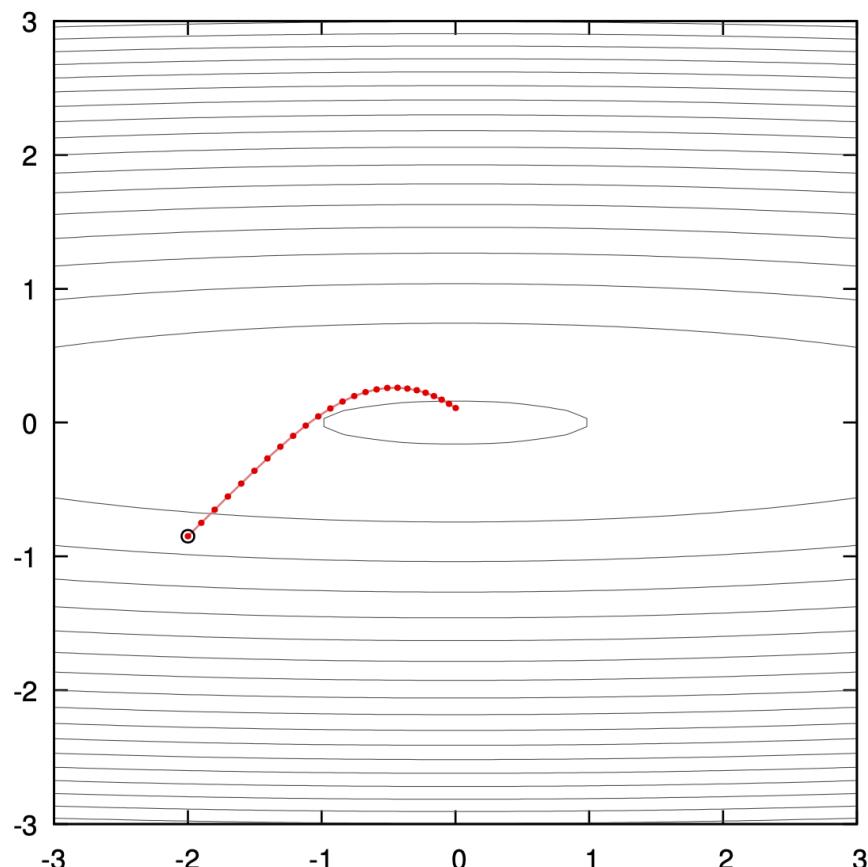
Without ADAM

$$\eta = 5.0e - 2, \gamma = 0$$



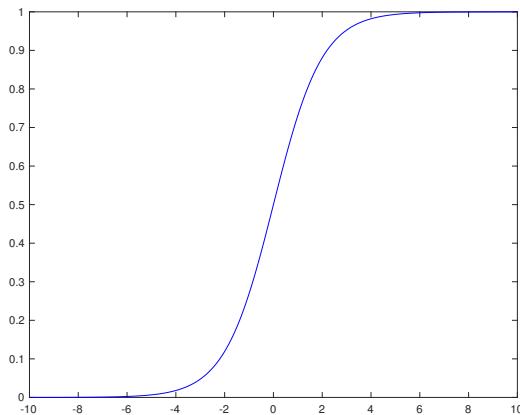
With ADAM

ADAM has several hyper-parameters that I will not list here

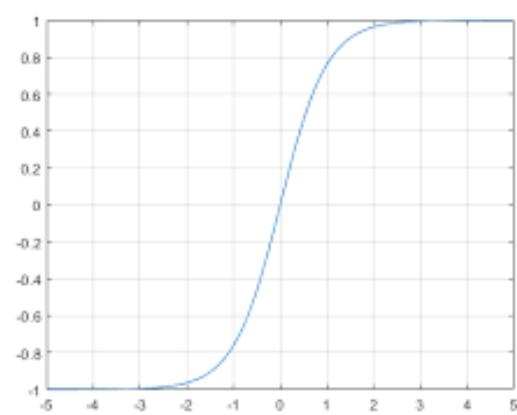


Gradient vanishing & initialization

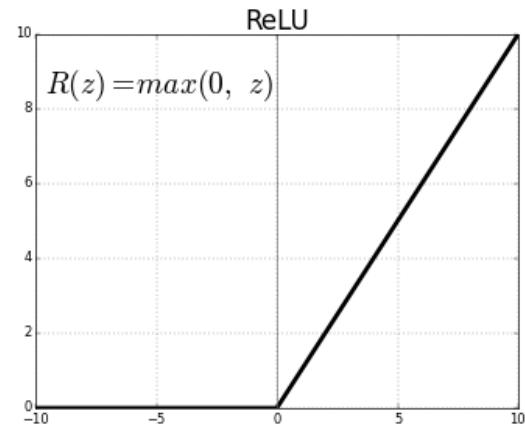
- The activation functions often have saturating regions where the gradient becomes (close to) 0



Sigmoid



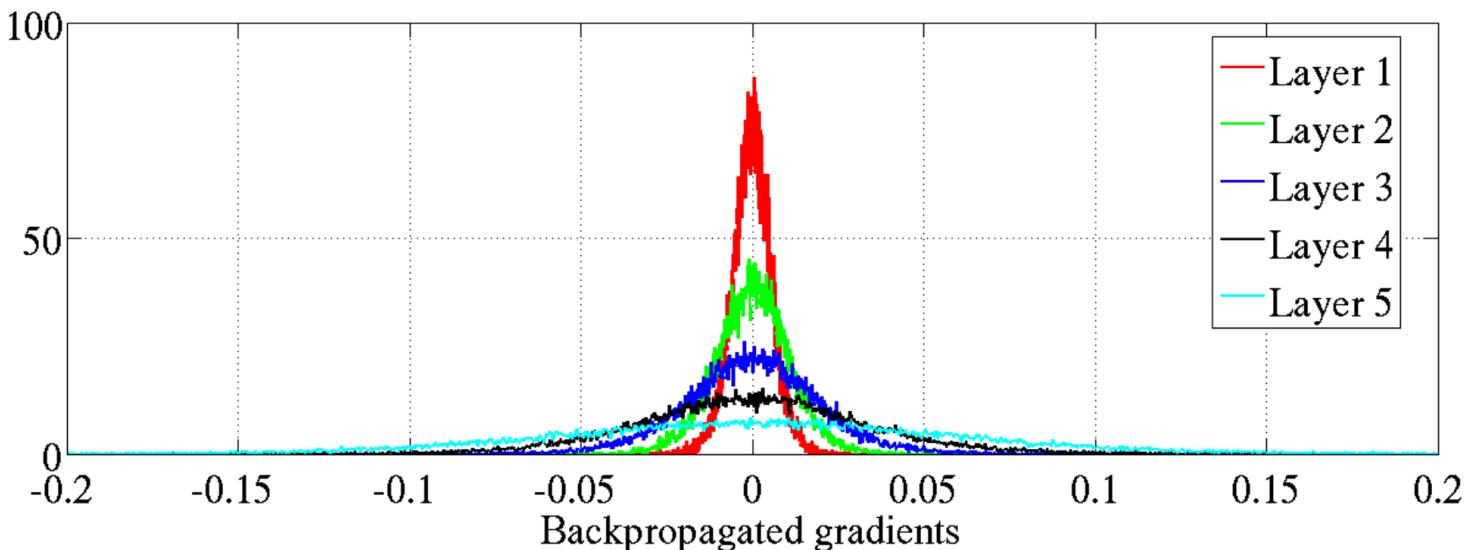
tanh



ReLU

Gradient vanishing & initialization

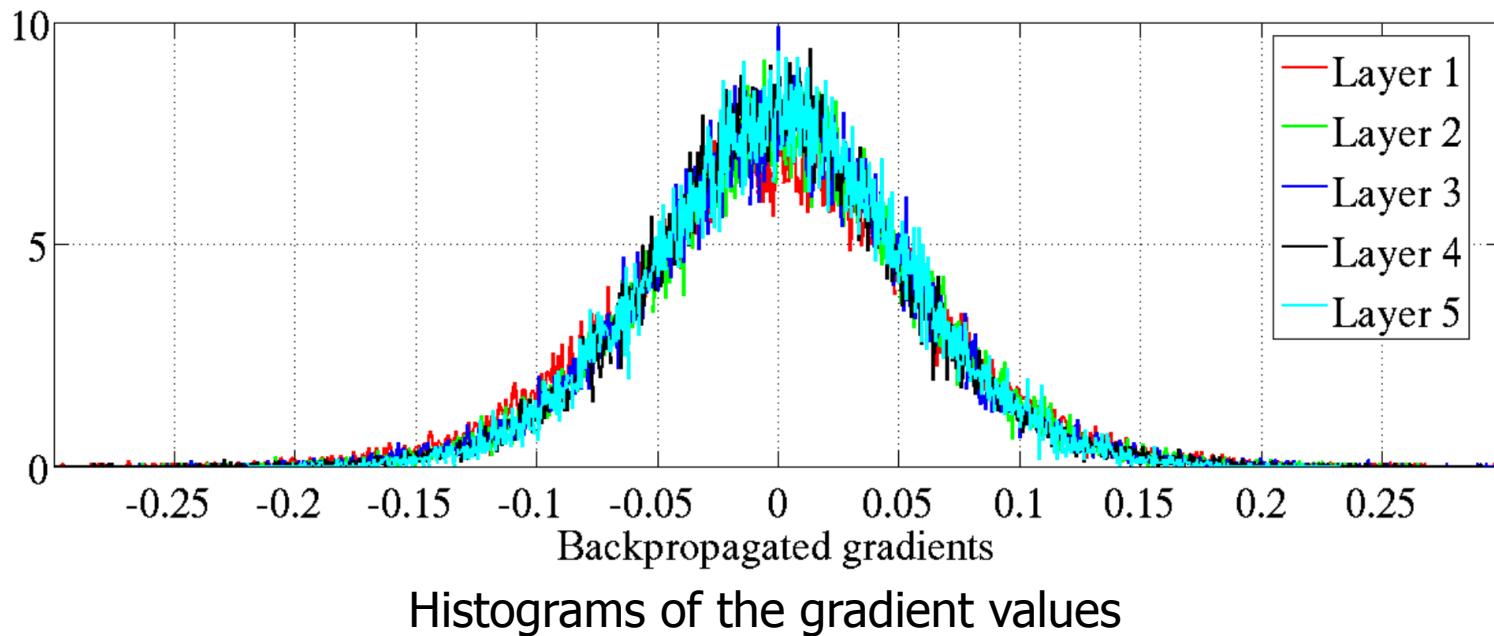
- Glorot & Bengio observed that the gradient vanishes exponentially with the (inverse) depth if the weights are such that they tend to often saturate the activation functions
 - X. Glorot & Y. Bengio, Understanding the difficulty of training deep feedforward neural networks, AISTATS, 2010



Histograms of the gradient values

Gradient vanishing & initialization

- To overcome this, Glorot & Bengio introduced a weight initialization strategy where the range of the initial, random values of the weights is layer dependent
 - This is referred to as Xavier (or Glorot) initialization



Regularization

- Deep networks have many parameters and can therefore overfit
 - In practice, this seems to happen surprisingly rarely
- One strategy to address this is called Dropout (Srivastava et al., JMLR 2014):
 - Randomly remove units and the connections during training
- Based on what we have learned so far, can you think of two other strategies to prevent overfitting?

MLP demos

- <http://playground.tensorflow.org/#activation=tanh&batchSize=10&dataset=circle®Dataset=reg-plane&learningRate=0.03®ularizationRate=0&noise=0&networkShape=4&seed=0.20666&showTestData=false&discretize=false&percTrainData=50&x=true&y=true&xTimesY=false&xSquared=false&ySquared=false&cosX=false&sinX=false&cosY=false&sinY=false&collectStats=false&problem=classification&initZero=false&hideText=false>
- <http://scs.ryerson.ca/%7Eaharley/vis/fc/>

Working with images

- Treating an image as a big vector seems counter-intuitive
 - An image of a reasonable size yields a huge vector, thus a huge amount of parameters in an MLP
 - Small image regions have similar property and should not be processed independently
 - A translation should not affect the results



$$\mathbf{x}_i \in \mathbb{R}^{536 \cdot 356 \cdot 3} = \mathbb{R}^{572448}$$

- Next week, we will see how we can reduce the number of parameters and improve invariance

Lecture 11: Deep Learning (Part 2)

Recap: Putting our findings together

- A simple artificial neural network

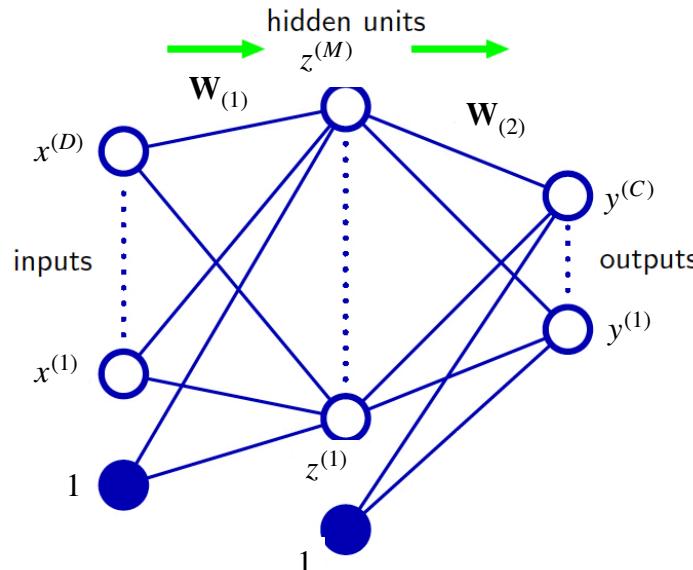
- We first map the input \mathbf{x} to a different representation (hidden units)

$$\mathbf{z} = f_{(1)}(\mathbf{W}_{(1)}^T \mathbf{x})$$

- The outputs are then obtained by another mapping

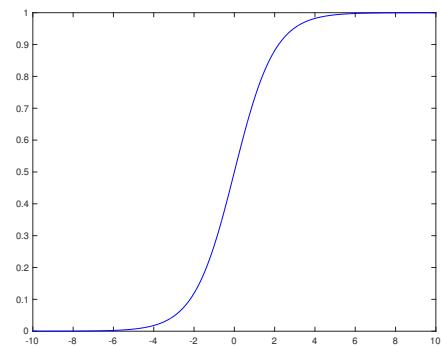
$$\mathbf{y} = f_{(2)}(\mathbf{W}_{(2)}^T \mathbf{z})$$

- Each mapping relies on a nonlinear function $f_{(j)}(\cdot)$, called *activation function*

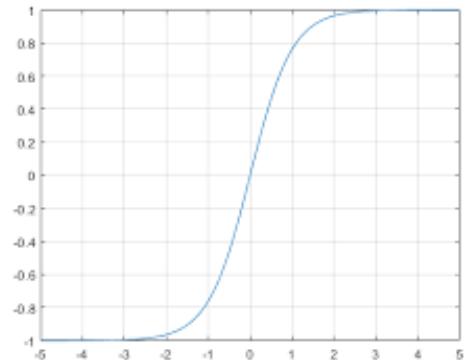


Recap: Activation function: Hidden layer

- . Sigmoid: $f(a) = \frac{1}{1 + \exp(-a)}$

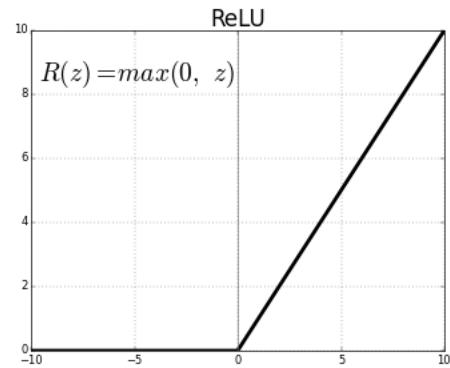


- . Hyperbolic tangent: $f(a) = \tanh(a)$



- . Rectified Linear Unit (ReLU)

$$f(a) = \begin{cases} a, & \text{if } a > 0 \\ 0, & \text{otherwise} \end{cases} = \max(a, 0)$$



Recap: Activation function: Final (output) layer

- For classification, one typically relies on the softmax function (as in multi-class logistic regression)

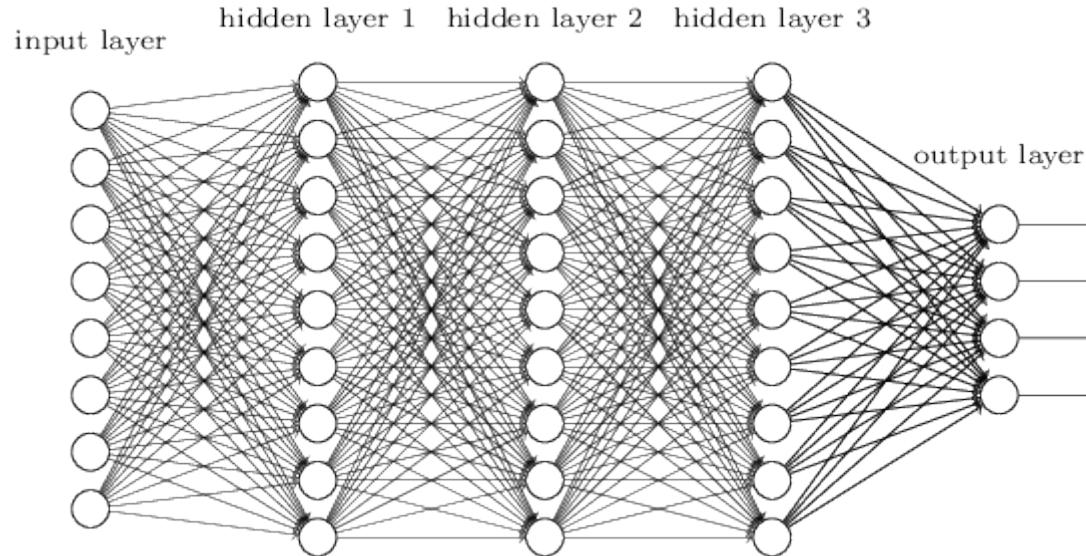
$$y^{(k)} = \frac{\exp(\mathbf{w}_{(2)(k)}^T \mathbf{z})}{\sum_{j=1}^C \exp(\mathbf{w}_{(2)(j)}^T \mathbf{z})}$$

where $\mathbf{w}_{(2)(j)}$ indicates the j^{th} column of $\mathbf{W}_{(2)}$

- For regression, one typically relies on a linear activation function (i.e., no activation function, $\mathbf{y} = \mathbf{W}_{(2)}^T \mathbf{z}$)

Recap: Multilayer Perceptron

- With a single hidden layer, one can already model complicated functions
- However, why stop at just one hidden layer?



- Slightly misleading name:
 - The perceptron uses a step function as activation function
 - MLPs typically rely on continuous functions (e.g., sigmoid, ReLU)

Recap: Training an MLP

- As usual, training is done by minimizing an empirical risk w.r.t. the parameters, here all the $\{\mathbf{W}_{(j)}\}$
- For regression, the loss function is typically the square loss

$$R(\{\mathbf{W}_{(j)}\}) = \frac{1}{N} \sum_{i=1}^N \|\hat{\mathbf{y}}_i - \mathbf{y}_i\|^2$$

- For classification, one usually relies on the cross-entropy

$$R(\{\mathbf{W}_{(j)}\}) = -\frac{1}{N} \sum_{i=1}^N \sum_{k=1}^C y_i^{(k)} \ln \hat{y}_i^{(k)}$$

Recap: Training an MLP

- Because an MLP stacks multiple layers, each of which has a nonlinear activation function, the problem does not have a closed-form solution and is not convex
- Learning is therefore done by gradient descent
 - Question: Can you think of a drawback of this strategy?
 - Answer: The solution will depend on the initial parameter values
- While the gradient might seem complicated, it can in fact be computed in a nice manner
 - This is called backpropagation

Recap: Backpropagation: Algorithm

1. Propagate \mathbf{x}_i forward through the network
2. Compute $\delta_{(L)}$ depending on the loss
3. Propagate $\delta_{(L)}$ backward to obtain each $\delta_{(l)}$
4. At each layer, compute the partial derivatives

$$\frac{\partial \ell_i}{\partial W_{(l)}^{(k,j)}} = \delta_{(l)}^{(k)} z_{(l-1)}^{(j)}$$

where $\mathbf{z}_{(l-1)}$ has been computed during the forward pass

Recap: Stochastic Gradient Descent

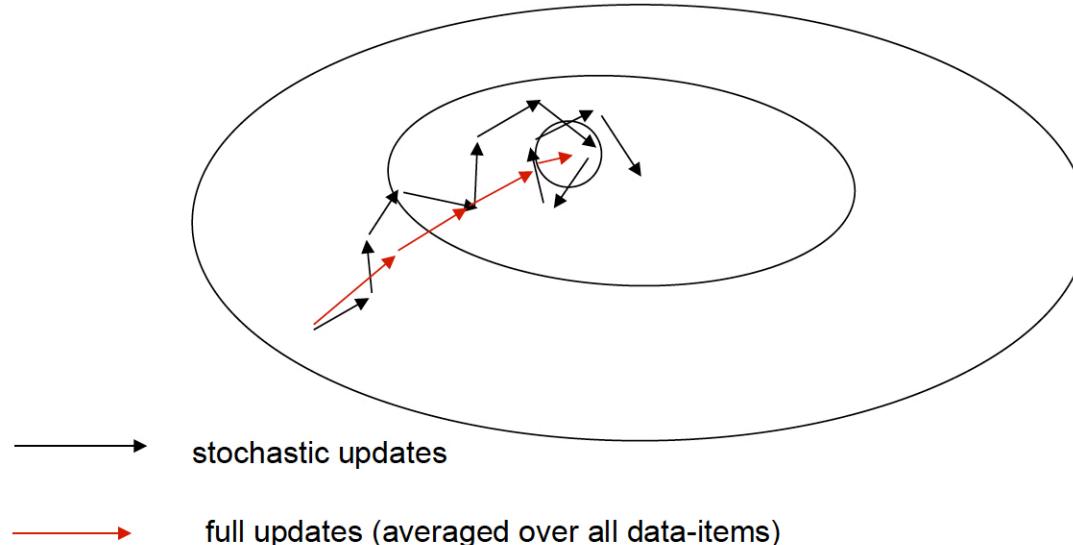
- Instead of relying on the full gradient, which is expensive to compute, *Stochastic* gradient descent updates the parameters \mathbf{W} based on the gradient of a single sample in each iteration
- To obtain a more reliable gradient estimate, one typically uses mini-batches of B samples
 - The parameters are then updated as

$$\mathbf{W}_k \leftarrow \mathbf{W}_{k-1} - \eta \sum_{b=1}^B \nabla \ell_{i(b,k)}(\mathbf{W}_{k-1})$$

The index of the samples used for the update now depends on the iteration number k and on the index b of the samples in the mini-batch

Recap: SGD behavior

- Because it approximates the full gradient, SGD will move the parameters less directly towards a local minimum
 - The parameters dance around the minimum, which may require decreasing the learning rate
 - Nevertheless, SGD can help to avoid bad local minima



Recap: Regularization

- Deep networks have many parameters and can therefore overfit
 - In practice, this seems to happen surprisingly rarely
- Strategies to address this include
 - Early stopping (use a validation set to check when the loss stops decreasing)
 - Weight decay:
 - Add an L_2 regularizer on the parameters, as seen in linear regression
 - Dropout (Srivastava et al., JMLR 2014):
 - Randomly remove units and their connections during training

Today: Working with images

- Treating an image as a big vector seems counter-intuitive
 - An image of a reasonable size yields a huge vector, thus a huge amount of parameters in an MLP
 - Small image regions have similar property and should not be processed independently
 - A translation should not affect the results



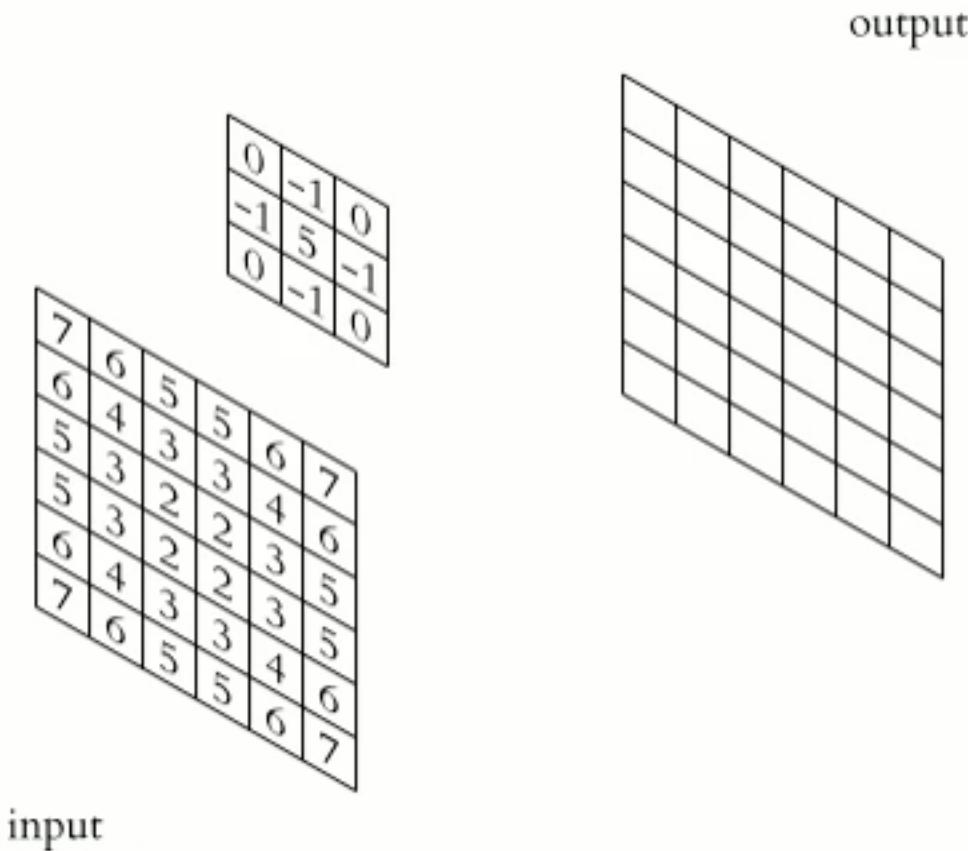
$$\mathbf{x}_i \in \mathbb{R}^{536 \cdot 356 \cdot 3} = \mathbb{R}^{572448}$$

- Can we reduce the number of parameters and improve robustness?

Goals of today's lecture

- Introduce the Convolutional Neural Network (CNN) model and its different types of layers
- Introduce the idea of transposed convolutions for fully-convolutional networks

2D convolutions



Convolutional layer

- Fewer parameters than fully-connected layers
 - the parameters of the filter (5x5 below) are learned
 - the parameters are shared across different spatial locations

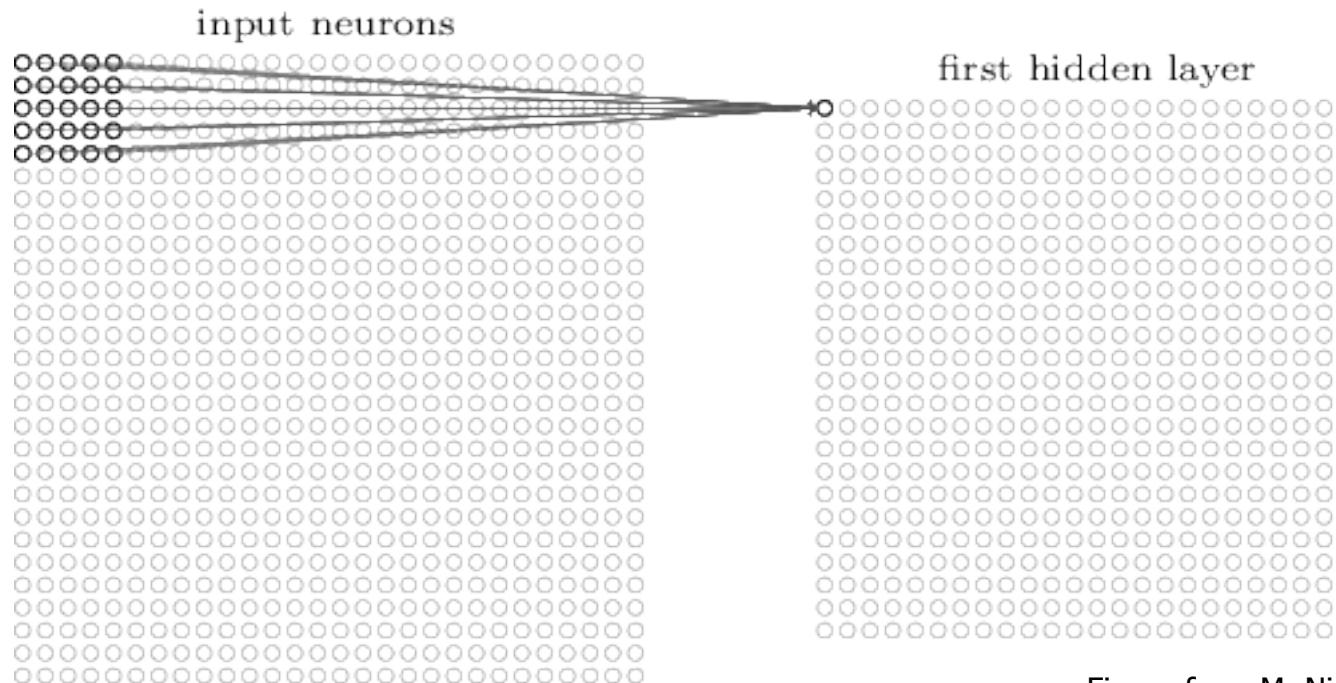


Figure from M. Nielsen's book
<http://neuralnetworksanddeeplearning.com>

Convolutional layer

- Fewer parameters than fully-connected layers
 - the parameters of the filter (5x5 below) are learned
 - the parameters are shared across different spatial locations
- Equivariance to translation

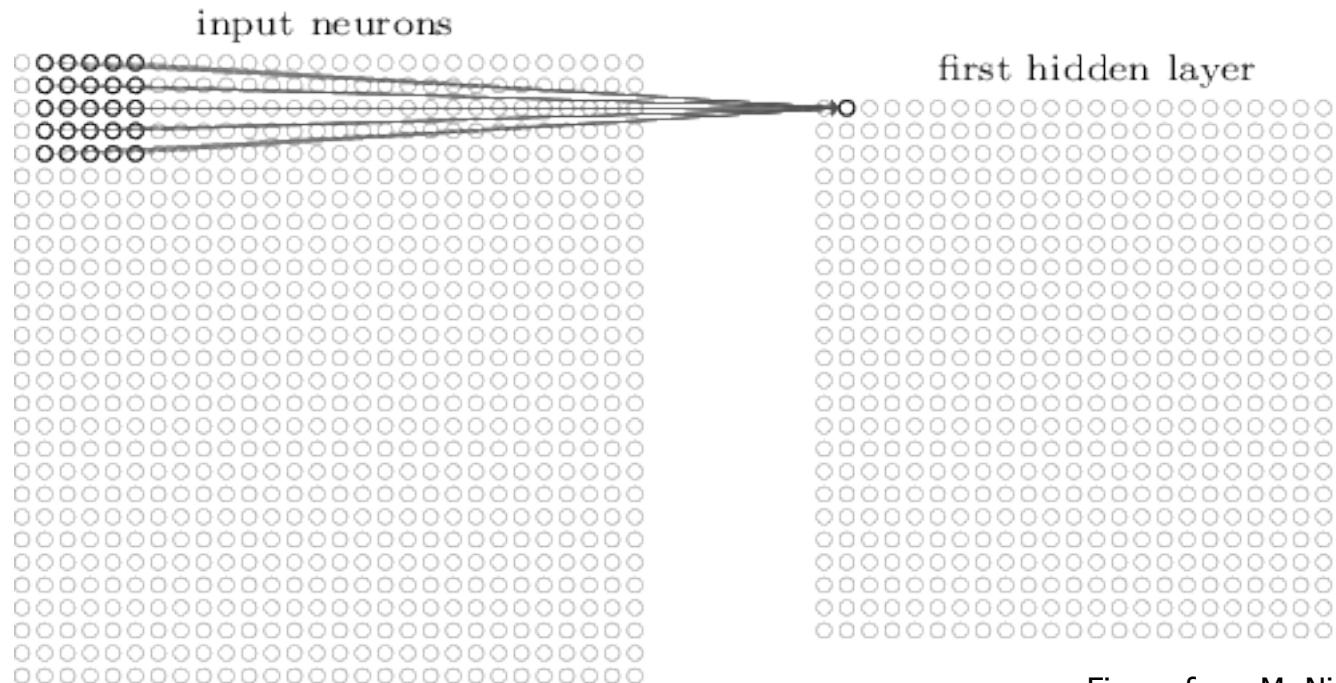
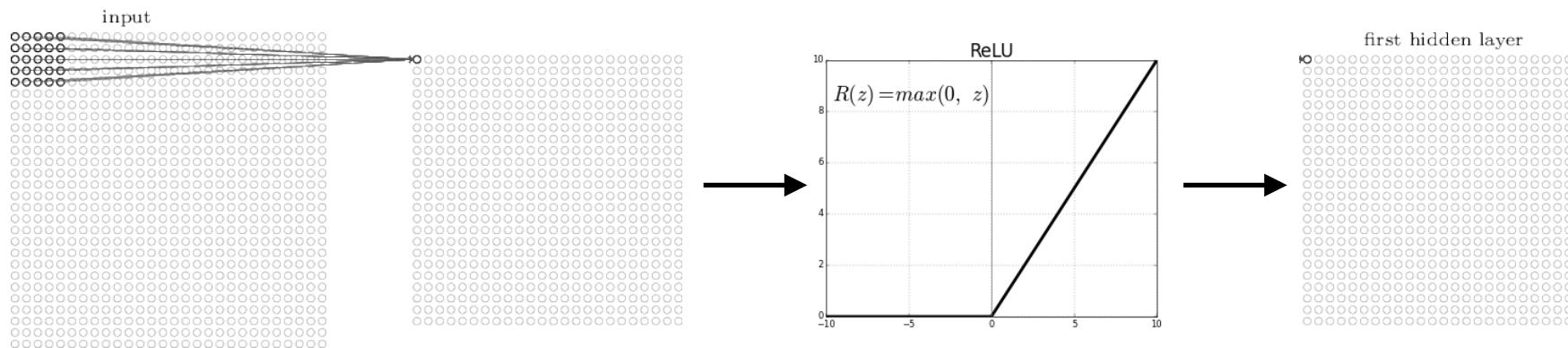


Figure from M. Nielsen's book
<http://neuralnetworksanddeeplearning.com>

Convolutional layer: Nonlinearity

- Note that each convolutional output is passed in an activation function. So, in fact, we have



- The activation function is applied in an element-wise manner, i.e., to each spatial location independently

Convolutional layer

- We can then use multiple filters to create multiple channels (3 in the example below)
 - Each filter has its own set of learnable parameters (in the example, we have 3 (5x5) filters, i.e., 75 learnable parameters)

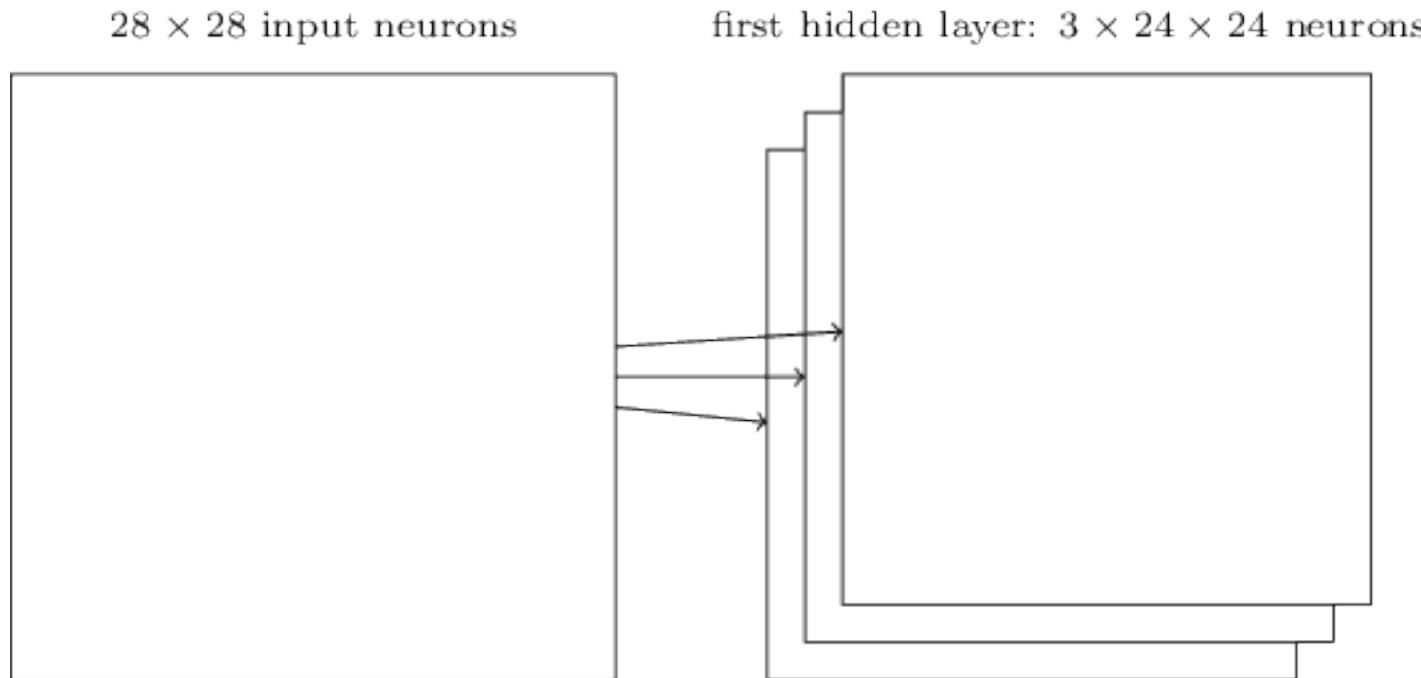


Figure from M. Nielsen's book
<http://neuralnetworksanddeeplearning.com>

Pooling layer

- From local information to a more global representation
- Different pooling strategies:
 - Max pooling
 - Average pooling

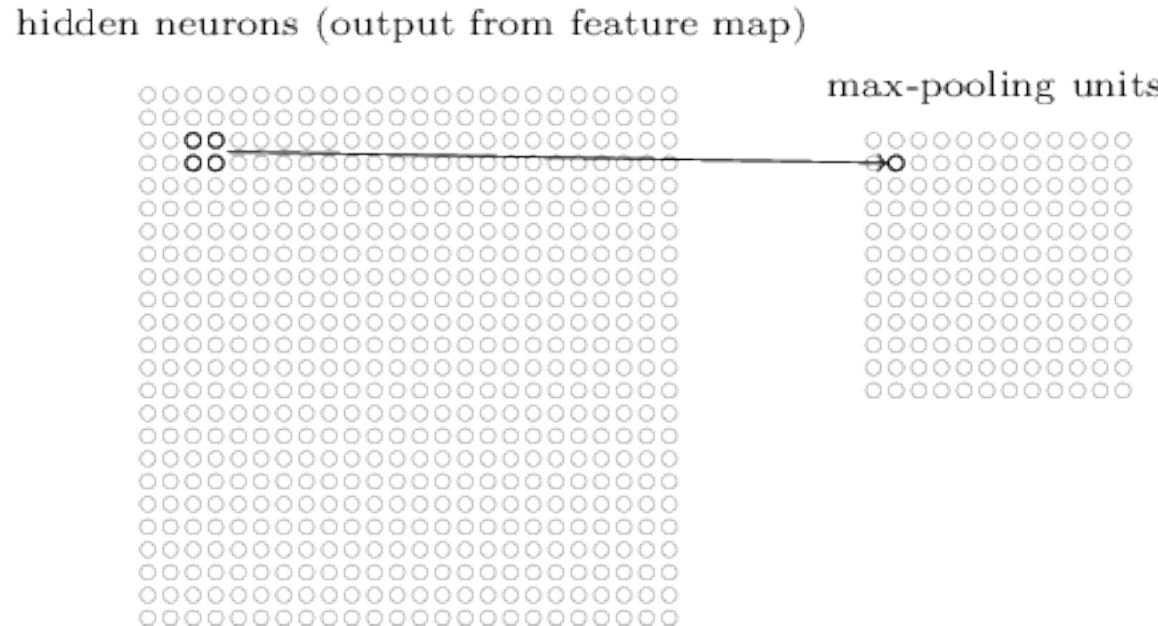


Figure from M. Nielsen's book
<http://neuralnetworksanddeeplearning.com>

Convolutional Neural Network (CNN)

- We can then stack these operations

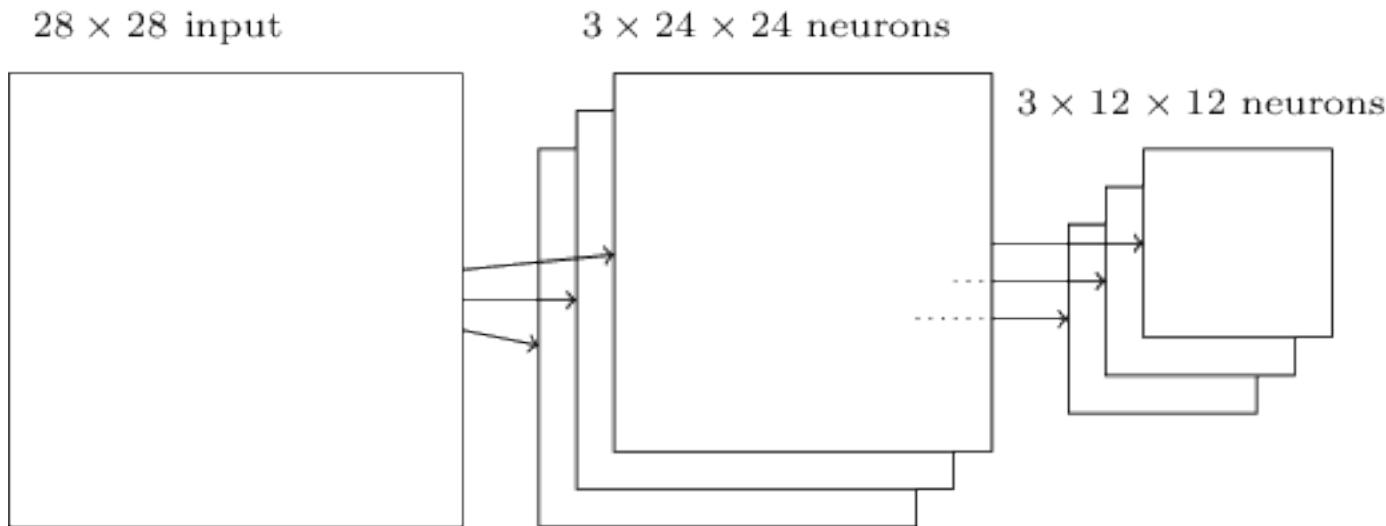


Figure from M. Nielsen's book
<http://neuralnetworksanddeeplearning.com>

Convolutional Neural Network (CNN)

- And then stack multiple such blocks
 - The number of channels can vary (below: first 3, then 6)
 - The size of the filters can vary (below: first 5×5 , then 3×3)

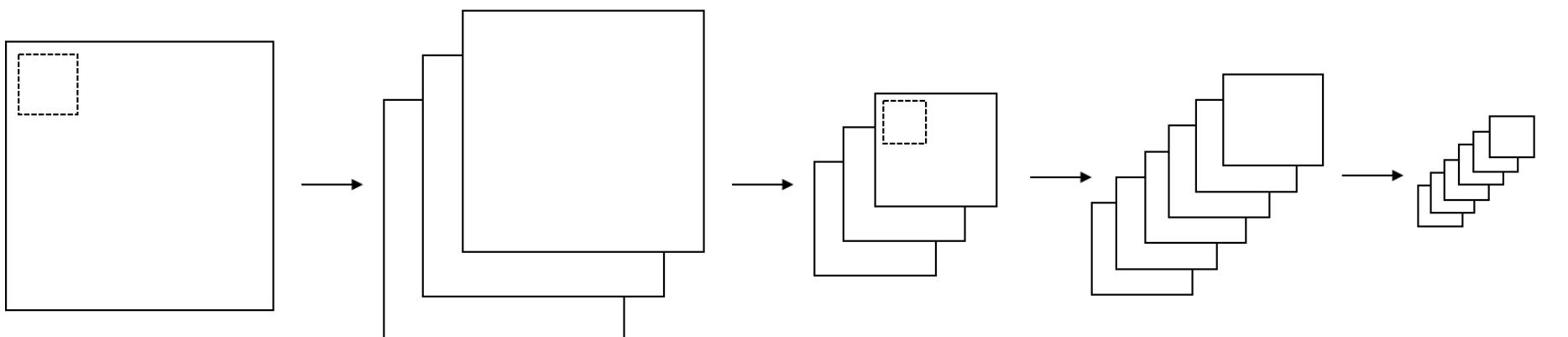
28 x 28

3 x 24 x 24

3 x 12 x 12

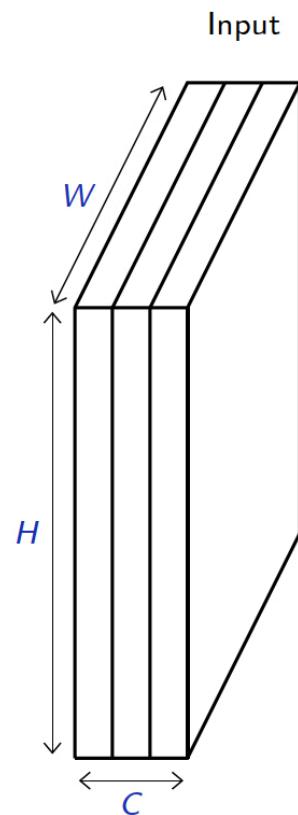
6 x 10 x 10

6 x 5 x 5



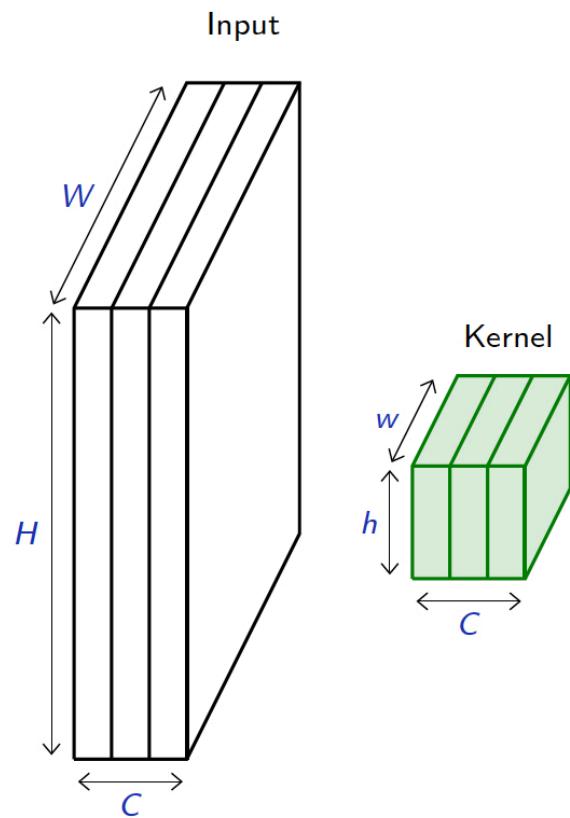
Convolutional Neural Network (CNN)

- Note that, with multiple *input channels*, the convolutions are in fact **done in 3D**



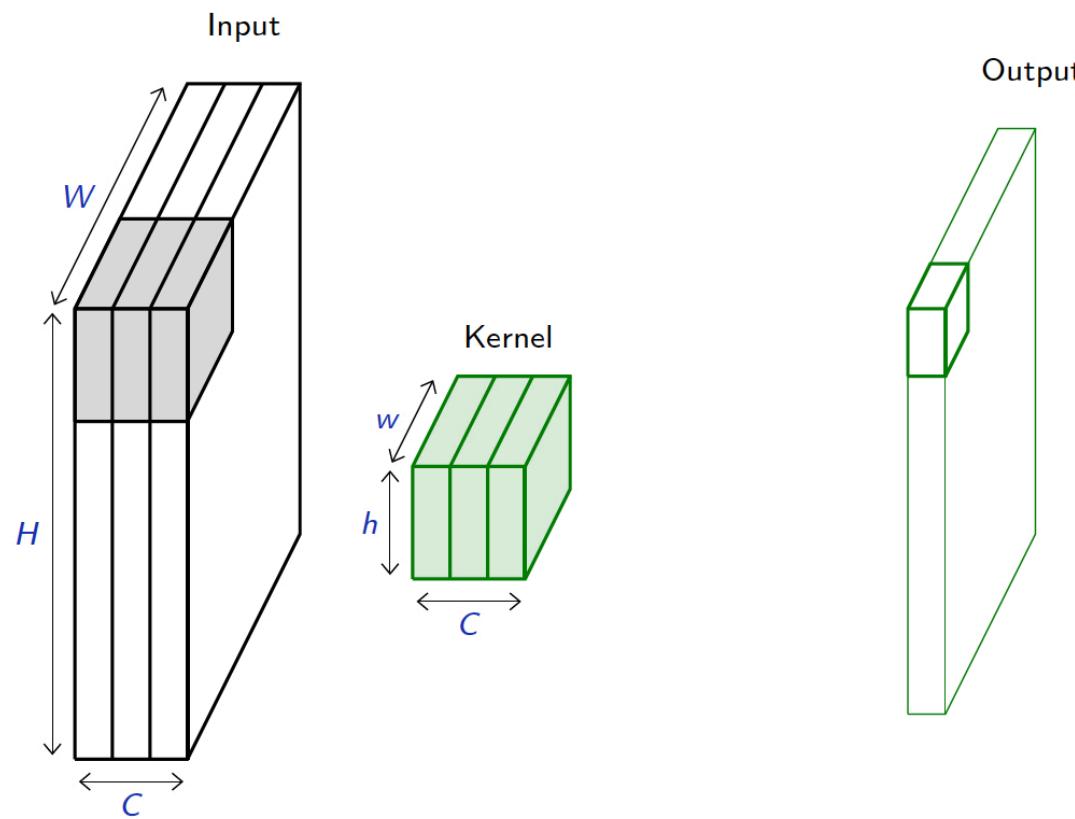
Convolutional Neural Network (CNN)

- Note that, with multiple *input* channels, the convolutions are in fact done in 3D



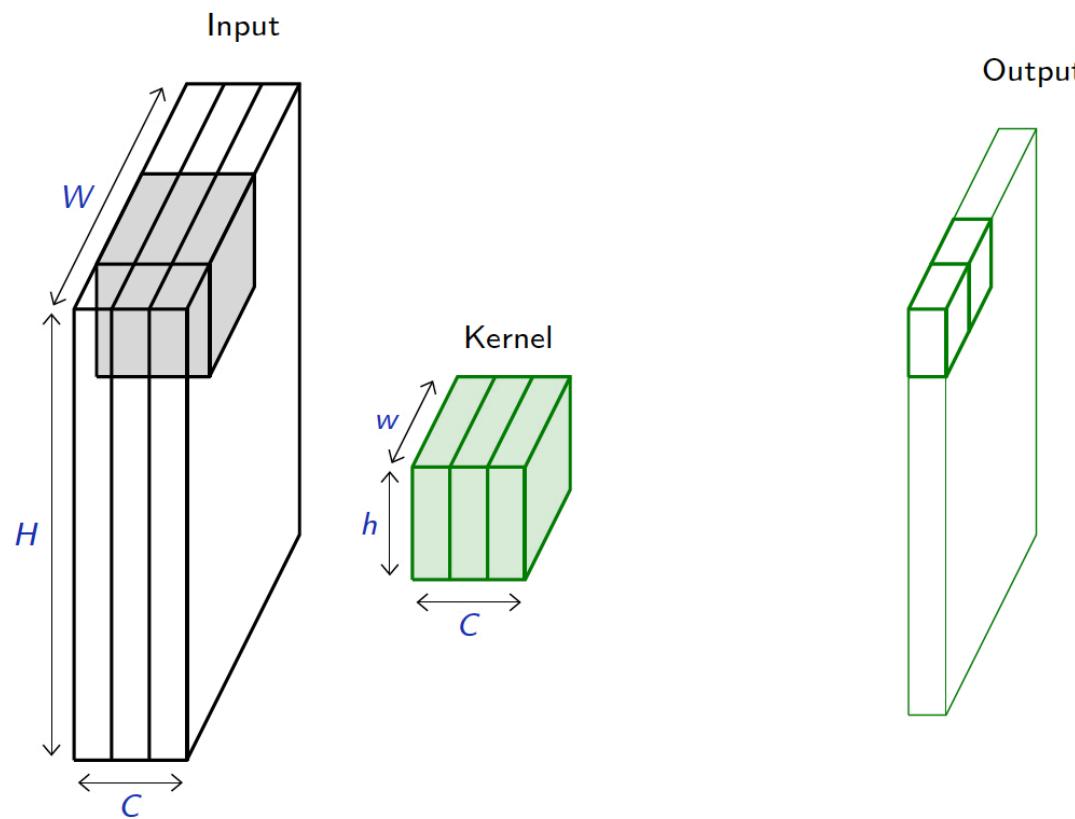
Convolutional Neural Network (CNN)

- Note that, with multiple *input* channels, the convolutions are in fact done in 3D



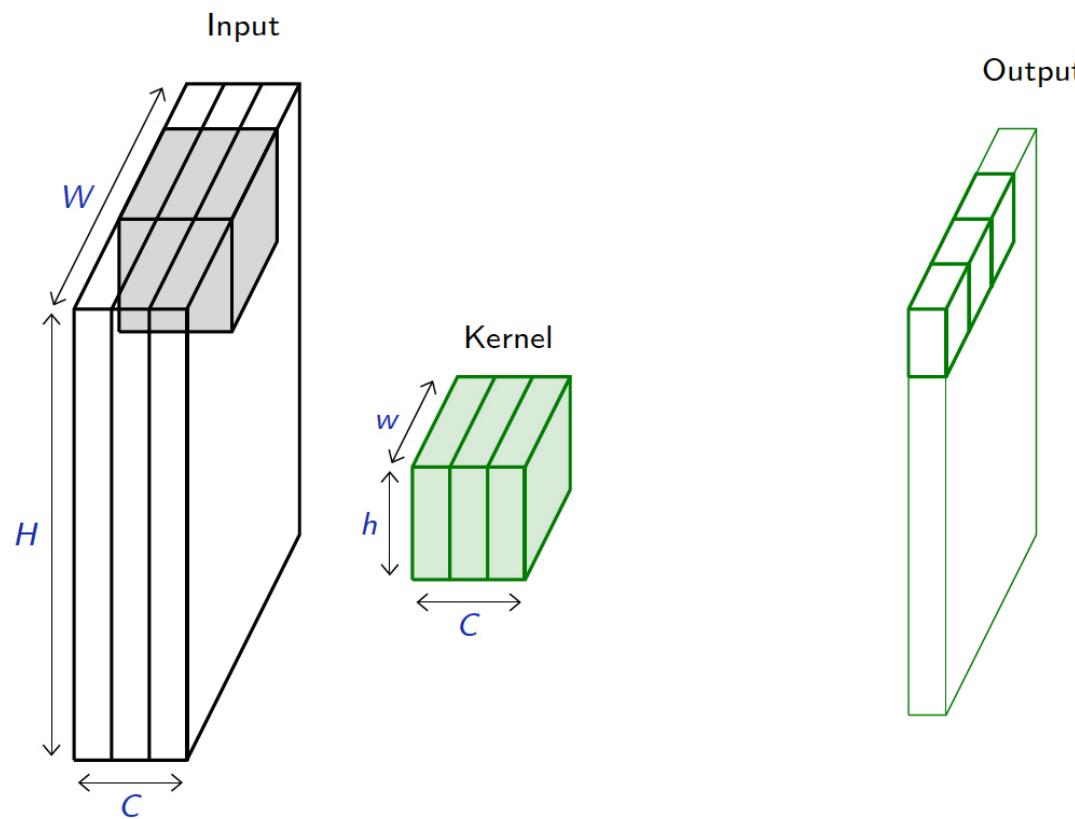
Convolutional Neural Network (CNN)

- Note that, with multiple *input* channels, the convolutions are in fact done in 3D



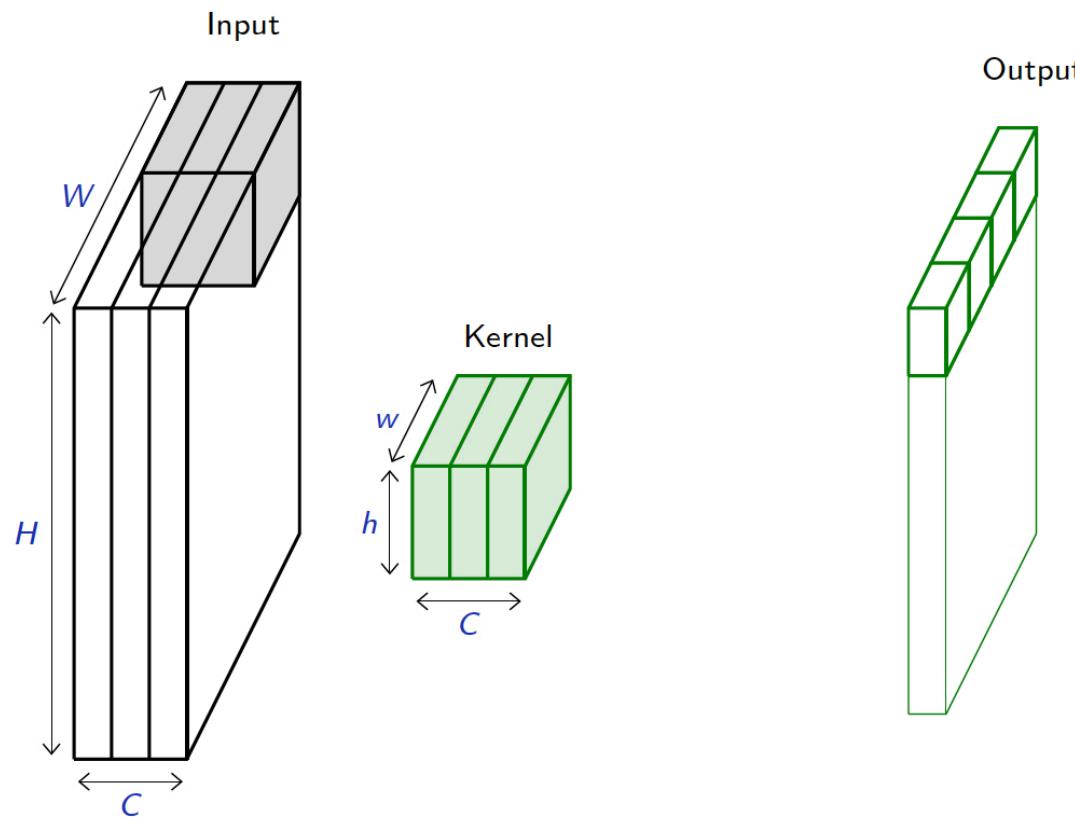
Convolutional Neural Network (CNN)

- Note that, with multiple *input* channels, the convolutions are in fact done in 3D



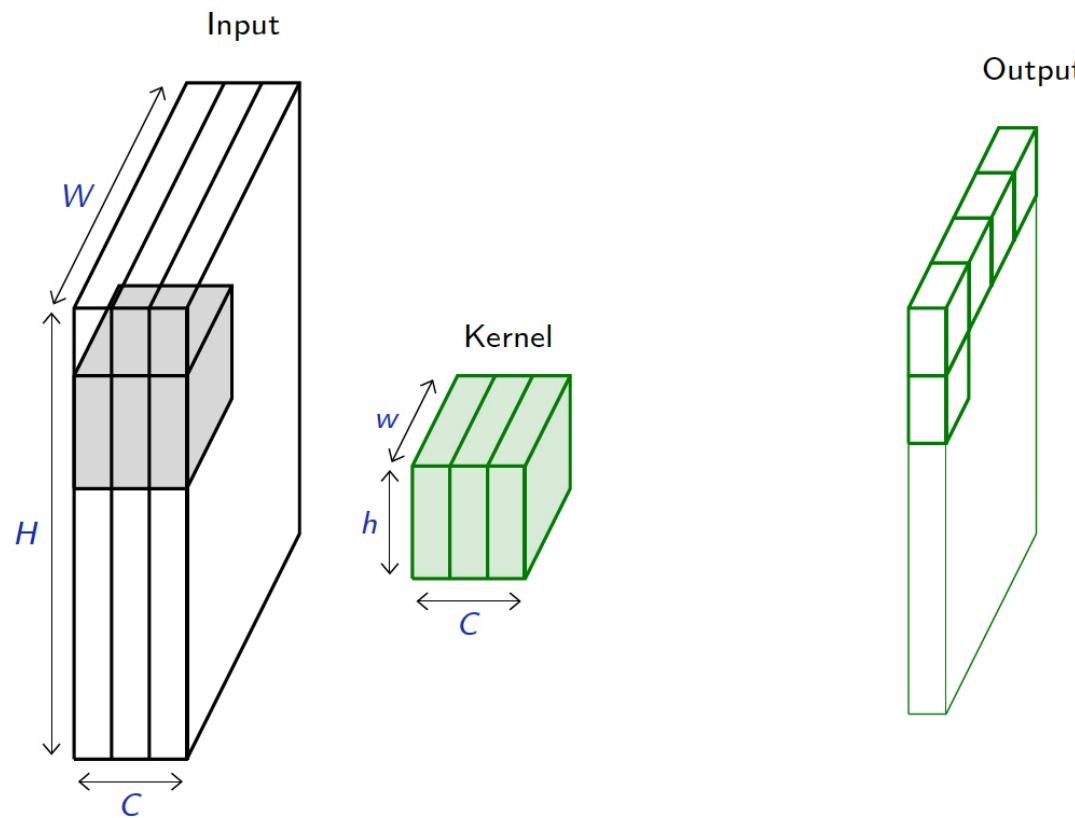
Convolutional Neural Network (CNN)

- Note that, with multiple *input* channels, the convolutions are in fact done in 3D



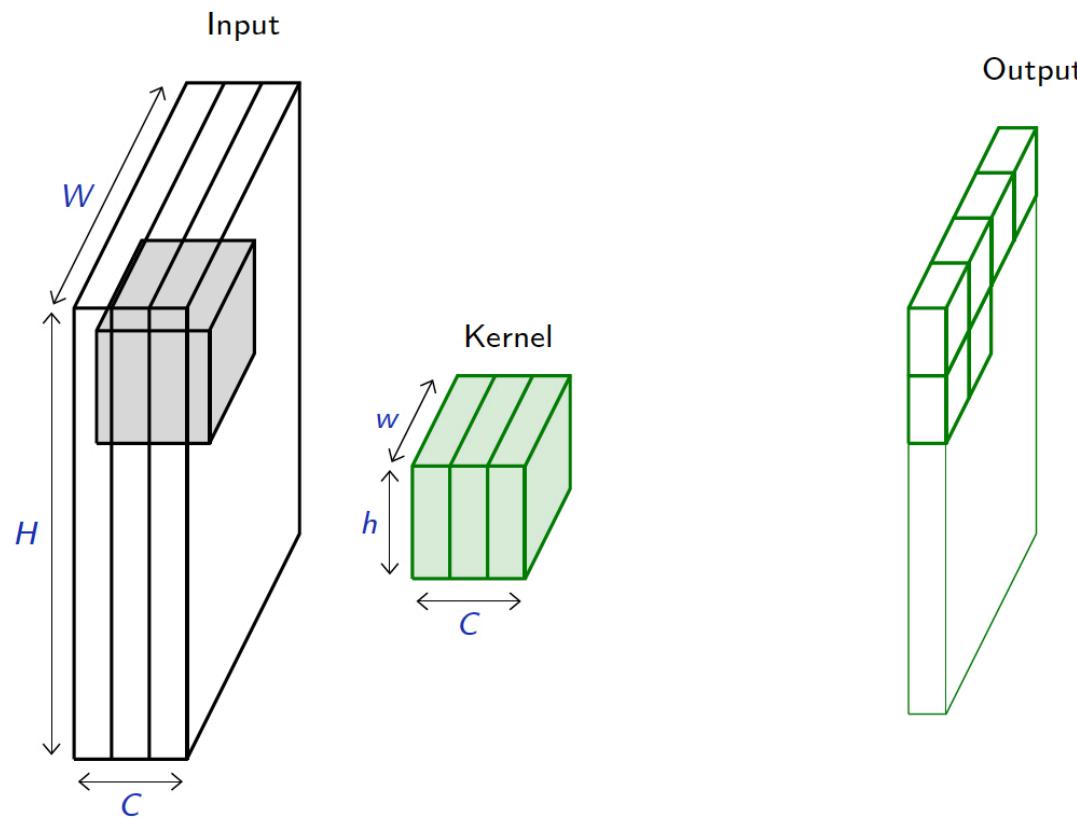
Convolutional Neural Network (CNN)

- Note that, with multiple *input* channels, the convolutions are in fact done in 3D



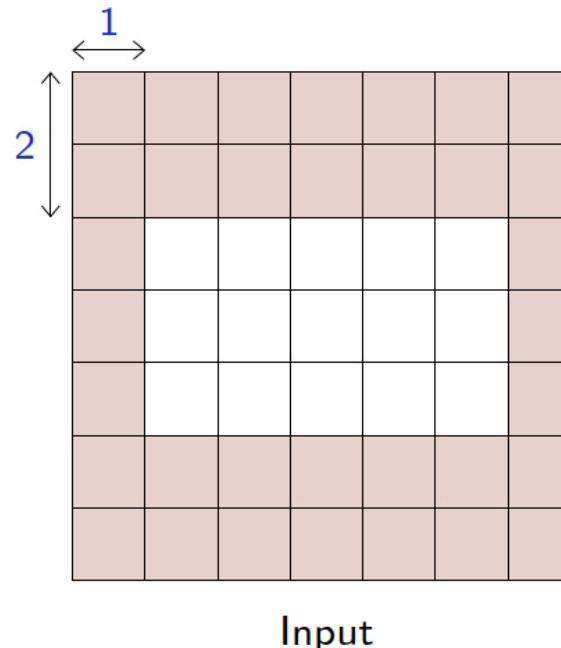
Convolutional Neural Network (CNN)

- Note that, with multiple *input* channels, the convolutions are in fact done in 3D



Padding

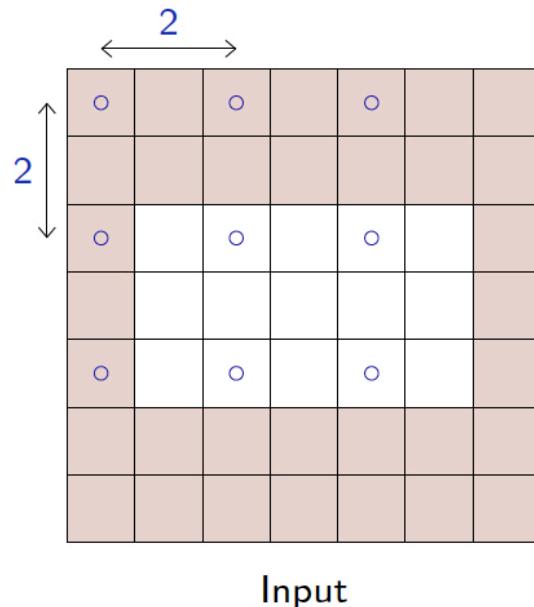
- To be able to center the convolution kernel on the **boundary locations**, padding extends the input
 - This can be done by mirroring or copying the boundary values
 - E.g., $C \times 3 \times 5$ input (in white) with a padding of (2,1)



Original input in white
Padded region in red

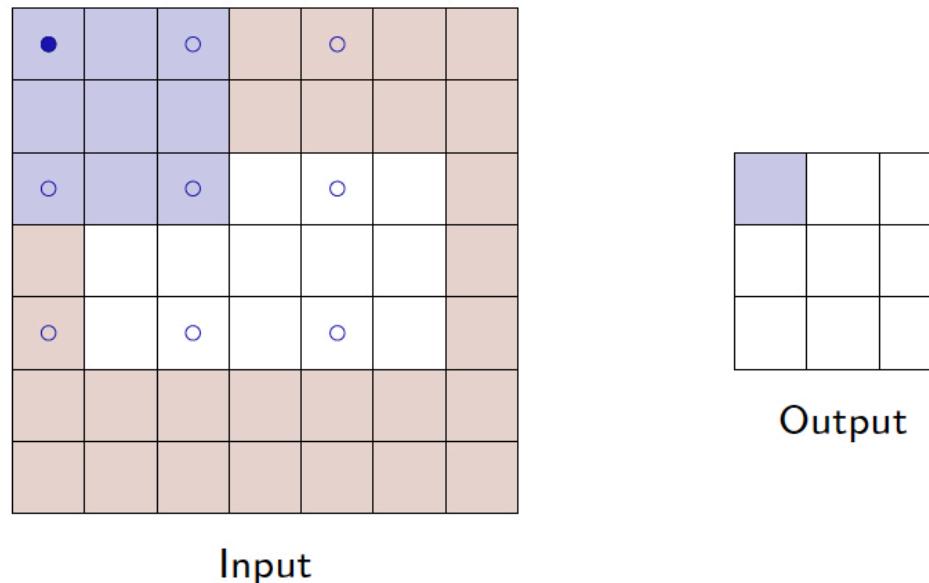
Convolution stride

- In a standard convolution, the filter is shifted pixel by pixel
- The shifts can be made larger by increasing the stride
 - E.g., stride of (2,2) on the previously-padded input



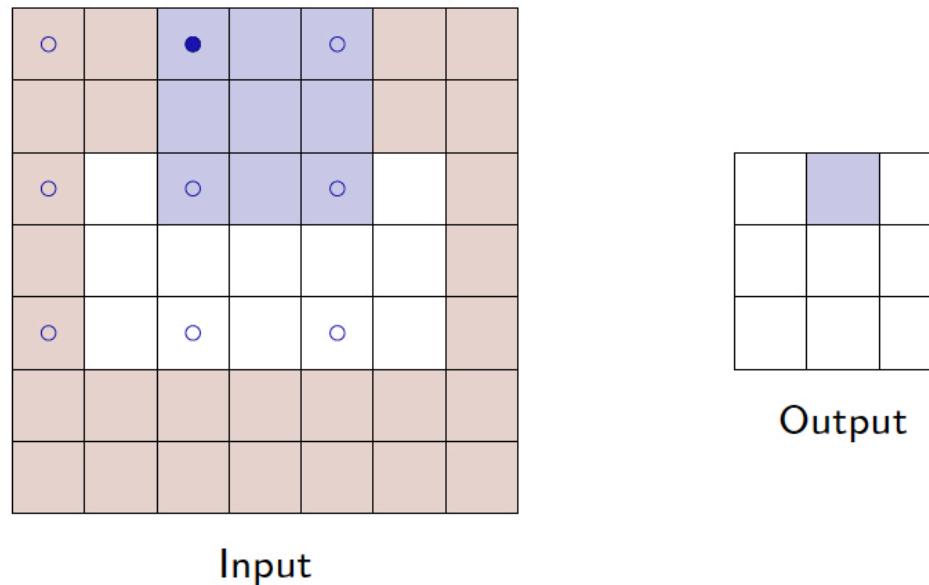
Convolution stride

- In a standard convolution, the filter is shifted pixel by pixel
- The shifts can be made larger by increasing the stride
 - E.g., stride of (2,2) on the previously-padded input
 - Applying a kernel of size 3×3



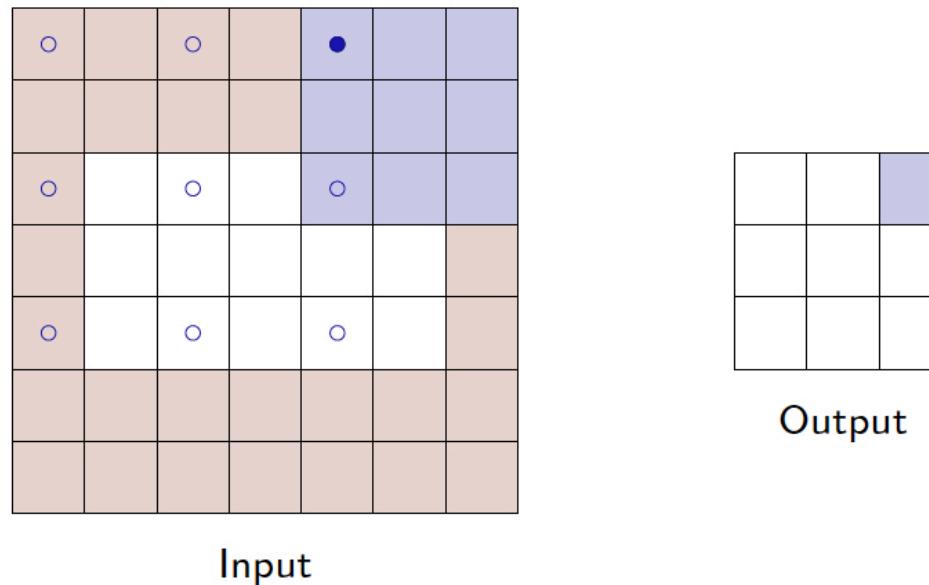
Convolution stride

- In a standard convolution, the filter is shifted pixel by pixel
- The shifts can be made larger by increasing the stride
 - E.g., stride of (2,2) on the previously-padded input
 - Applying a kernel of size 3×3



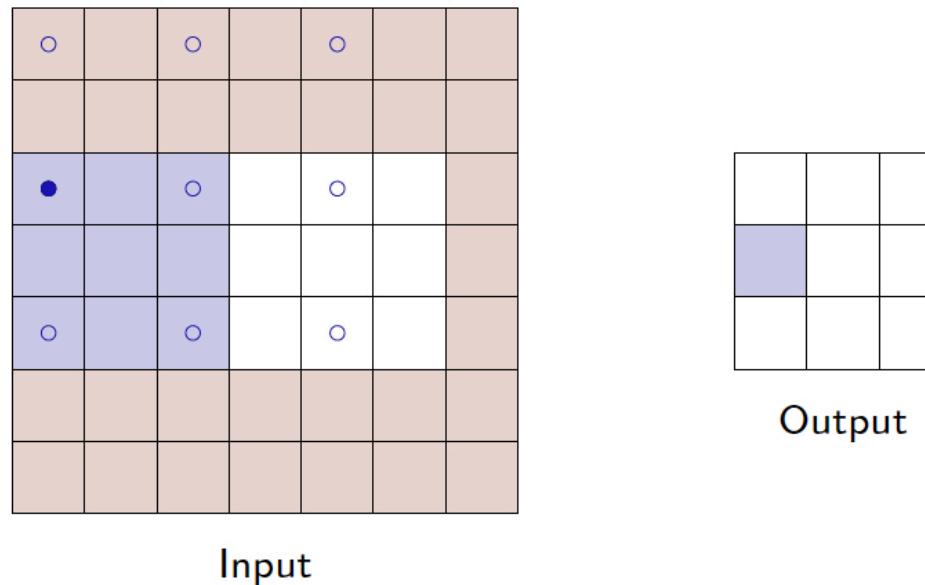
Convolution stride

- In a standard convolution, the filter is shifted pixel by pixel
- The shifts can be made larger by increasing the stride
 - E.g., stride of (2,2) on the previously-padded input
 - Applying a kernel of size 3×3



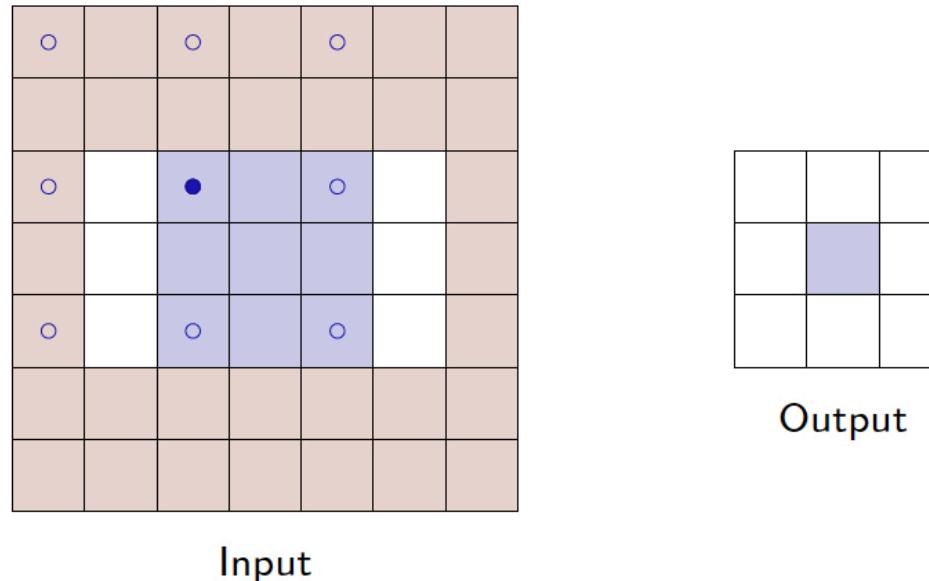
Convolution stride

- In a standard convolution, the filter is shifted pixel by pixel
- The shifts can be made larger by increasing the stride
 - E.g., stride of (2,2) on the previously-padded input
 - Applying a kernel of size 3×3



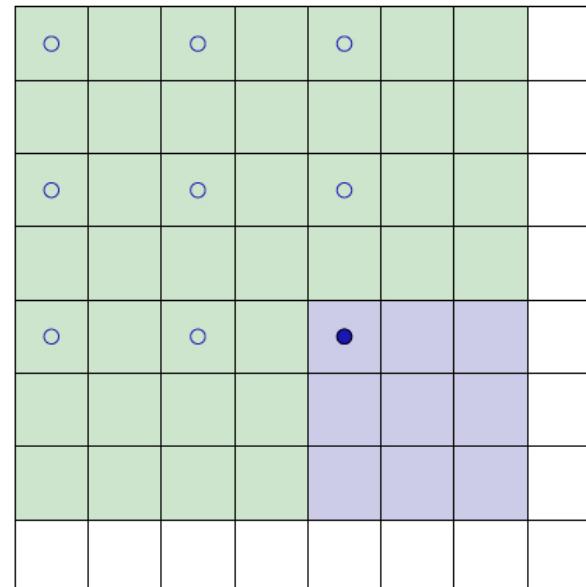
Convolution stride

- In a standard convolution, the filter is shifted pixel by pixel
- The shifts can be made larger by increasing the stride
 - E.g., stride of (2,2) on the previously-padded input
 - Applying a kernel of size 3×3



Convolution stride

- Warning: A convolution with a stride greater than 1 may not cover the input map completely, hence ignoring some of the input values



Convolutional Neural Network (CNN)

- Eventually, one typically wants to **output a vector**, e.g.,
 - a vector of class probabilities for classification
 - a vector of 3D joint coordinates for human pose estimation
- This can be achieved by **appending fully-connected layer(s)**
 - This requires vectorizing the last convolutional output first (e.g., $3 \times 12 \times 12 \rightarrow 432$ -dimensional vector)

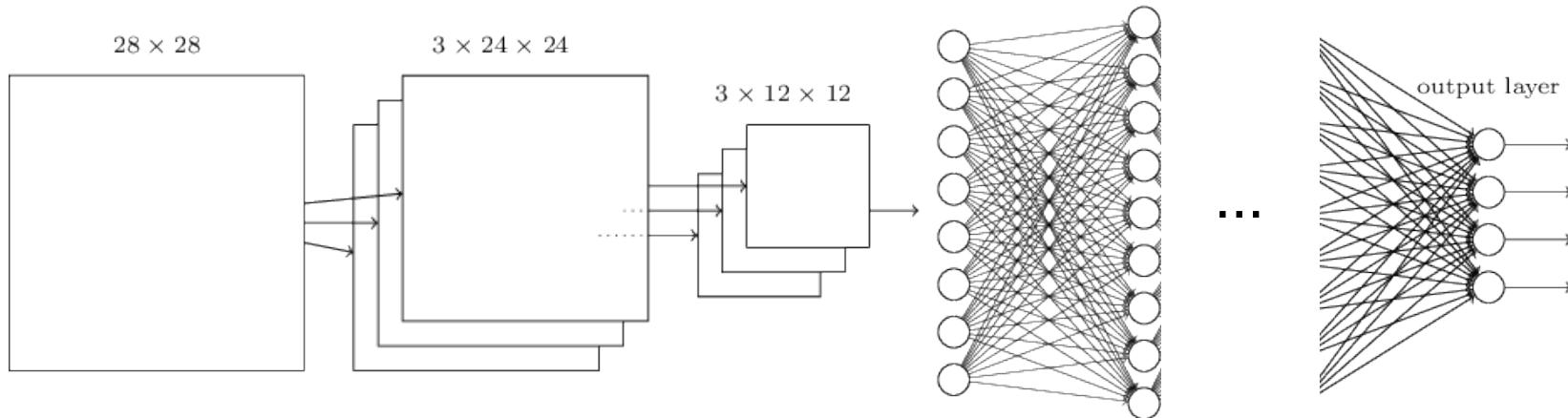
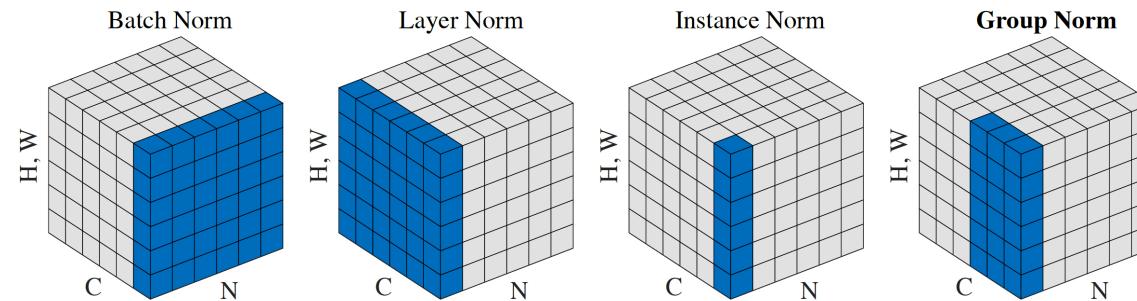


Figure adapted from M. Nielsen's book
<http://neuralnetworksanddeeplearning.com>

Normalization layers

- Inspired by the fact that data normalization can help, several layers have been developed to normalize the feature maps
- These layers include
 - Batch normalization (Ioffe & Szegedy, ICML 2015)
 - Layer normalization (Ba et al., 2016)
 - Instance normalization (Ulyanov et al., 2016)
 - Group normalization (Wu & He, 2018)



N : Number of samples in a mini-batch; C : Number of channels;
 (H, W) : Size of the feature map

CNN: Learning

- Learning the weights of the CNN layers is done in the same manner as for MLPs
 - Using stochastic gradient descent with mini-batches
- The same idea of error backpropagation that we discussed last time is applicable to convolutional layers
- The gradient of pooling layers depends on the strategy:
 - For average pooling, the gradient is backpropagated to all neurons used during pooling
 - For max pooling, the gradient is backpropagated only to the neuron that corresponded to the maximum value

CNN: Learning

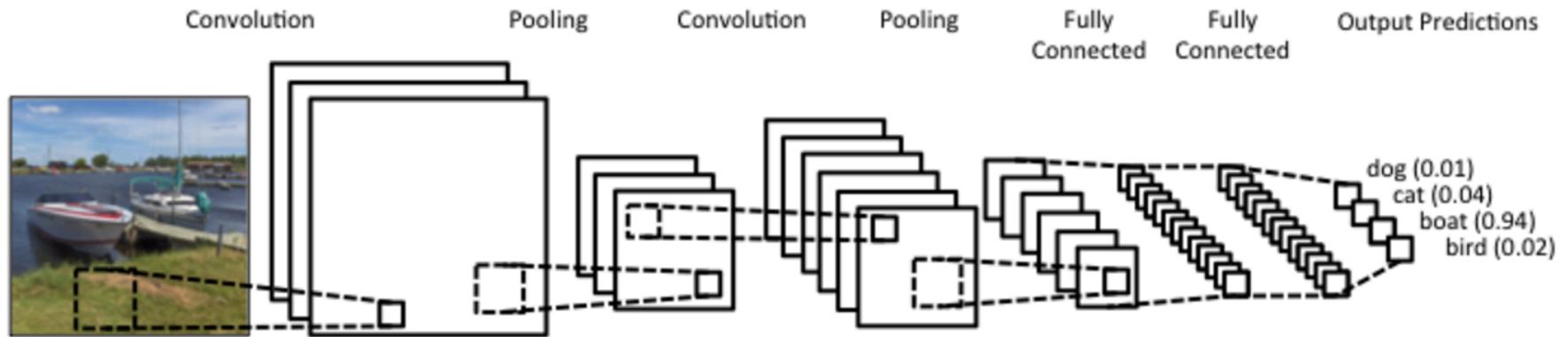
- The same SGD extensions as in MLP can be used:
 - Momentum
 - Adaptive learning rates
- The same regularization techniques as in MLP can be used:
 - Early stopping
 - Weight decay
 - Dropout

CNN: Demo

- <http://scs.ryerson.ca/~aharley/vis/conv/>

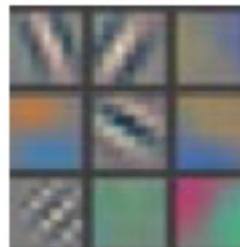
CNN: Interpretation

- A CNN can be thought of as jointly learning the data representation and the classifier (or regressor)



CNN: Interpretation

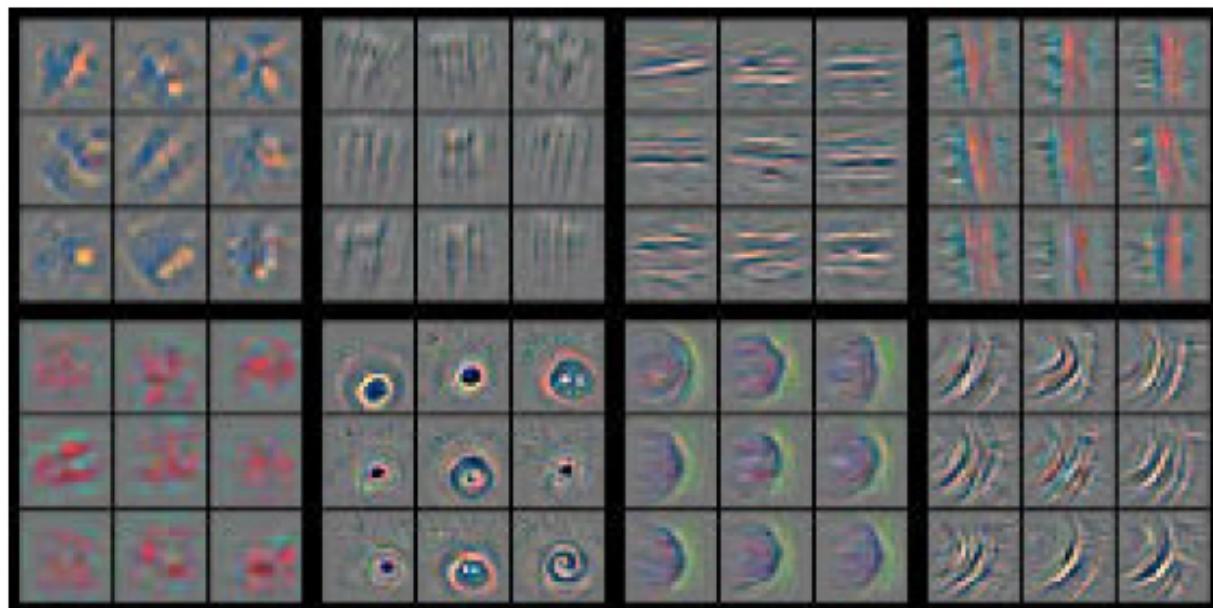
- One can visualize the features extracted at different layers
 - Zeiler & Fergus, ECCV 2014



Layer 1

CNN: Interpretation

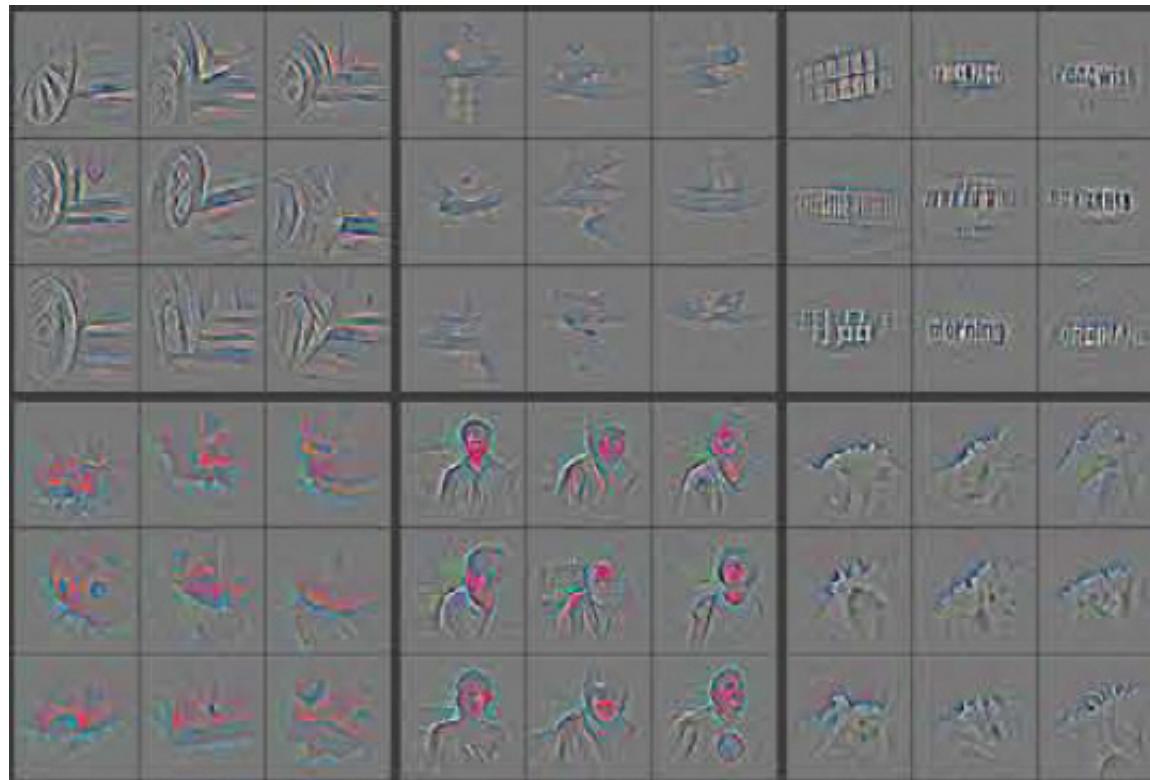
- One can visualize the features extracted at different layers
 - Zeiler & Fergus, ECCV 2014



Layer 2

CNN: Interpretation

- One can visualize the features extracted at different layers
 - Zeiler & Fergus, ECCV 2014



Layer 3

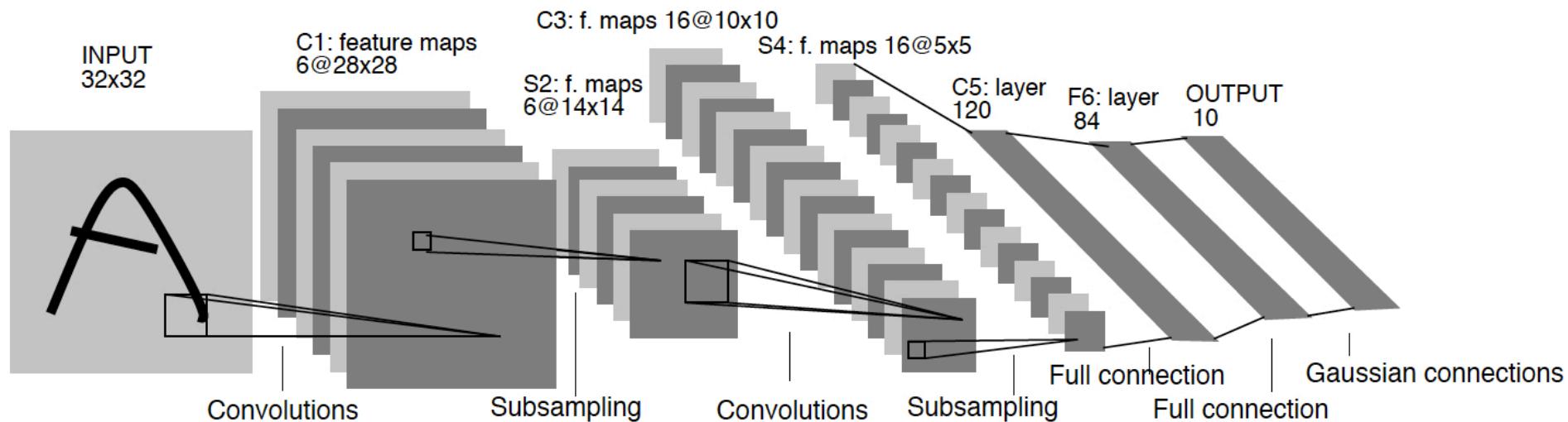
Visualization demo

- <https://cs.stanford.edu/people/karpathy/convnetjs/>

Standard architectures

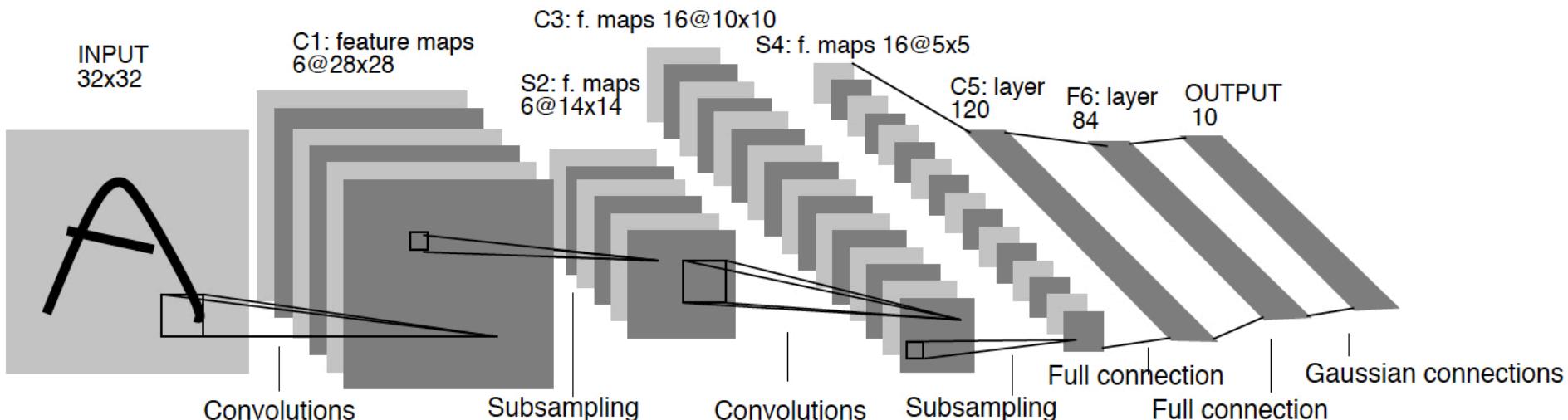
- **LeNet-5**

- LeCun et al., Gradient-Based Learning Applied to Document Recognition, 1998



Exercise

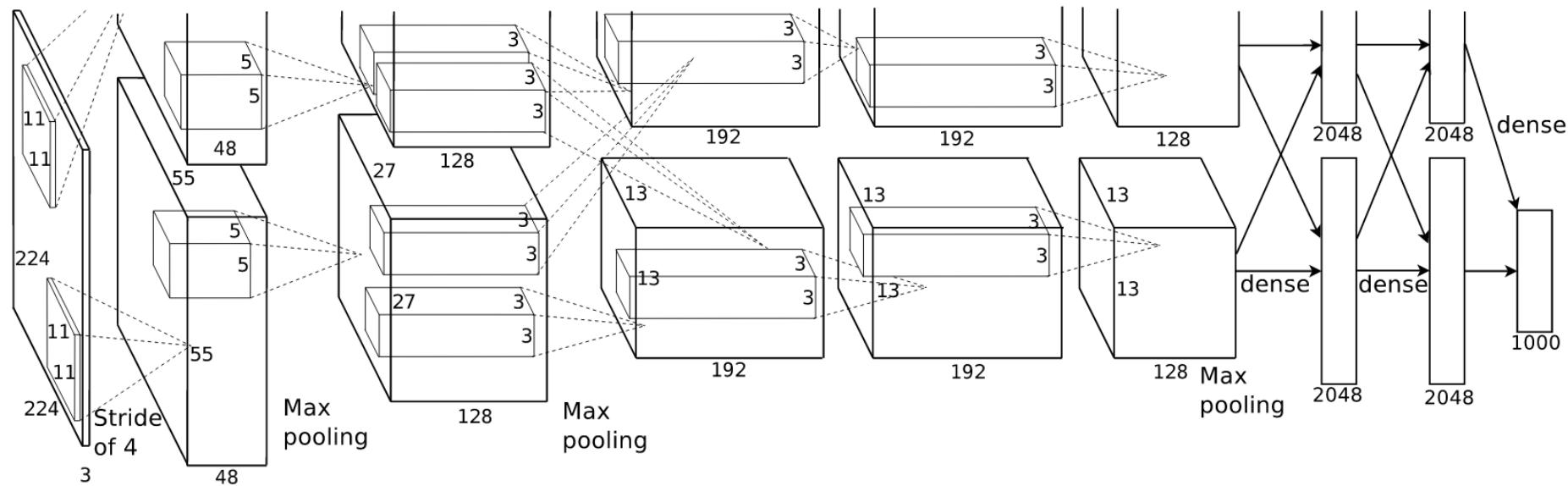
- In the LeNet-5 architecture, how many learnable parameters are there in
 - the first convolutional layer (from Input to C1)?
 - the pooling layer (from C1 to S2)?
 - the second convolutional layer (from S2 to C3)?
 - the last fully-connected layer (from F6 to Output)?



Standard architectures

- **AlexNet**

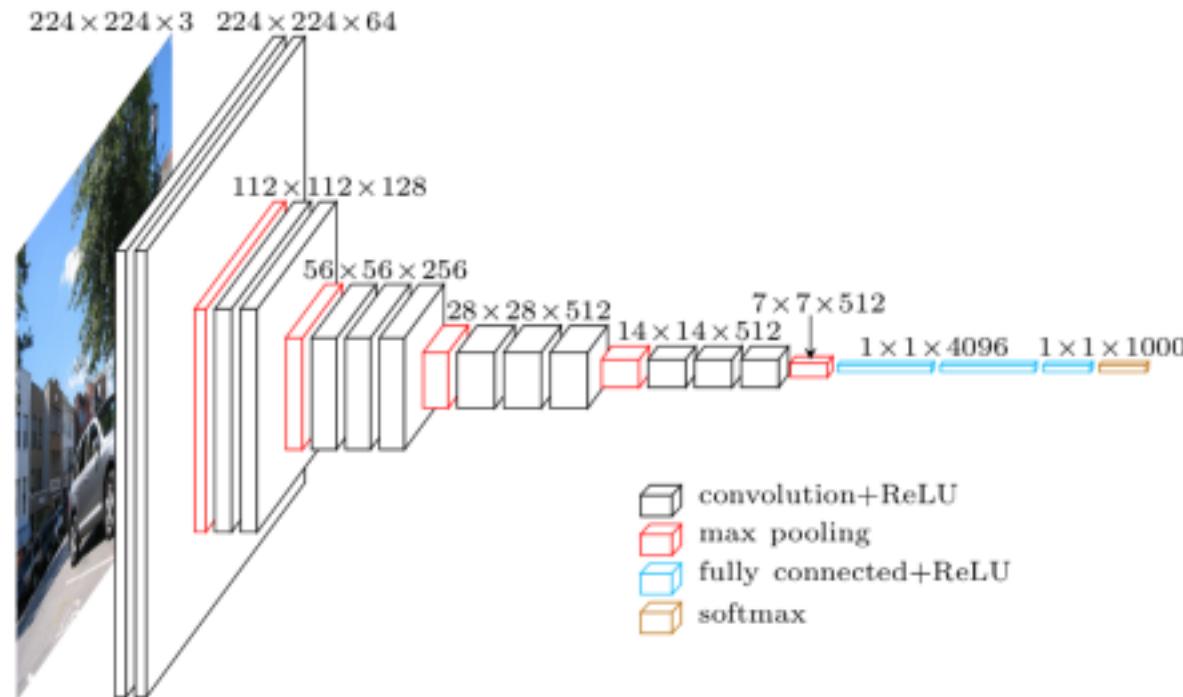
- Krizhevsky et al., ImageNet Classification with Deep Convolutional Neural Networks, NIPS 2012



Standard architectures

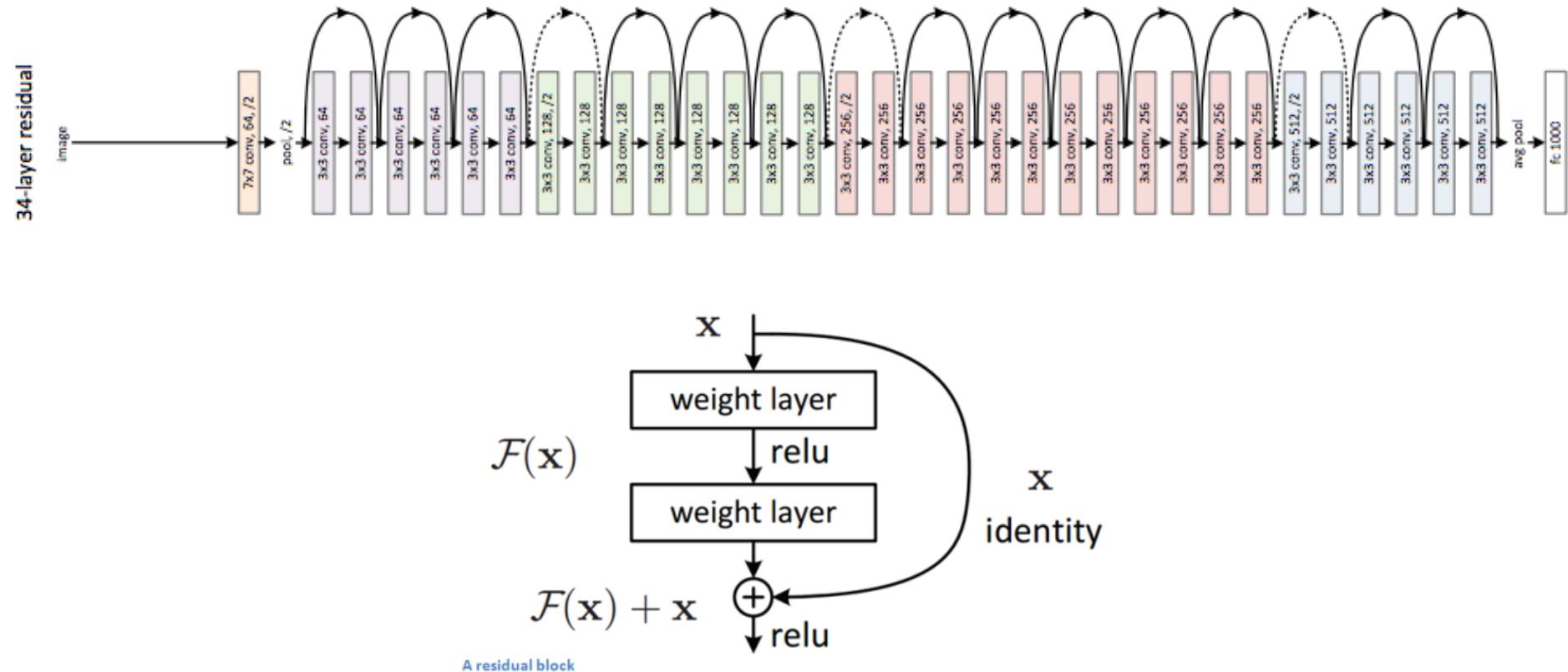
- VGG

- Simonyan & Zisserman, Very Deep Convolutional Networks for Large-Scale Image Recognition, 2014



Standard architectures

- ResNet
 - He et al., Deep Residual Learning for Image Recognition, CVPR 2016

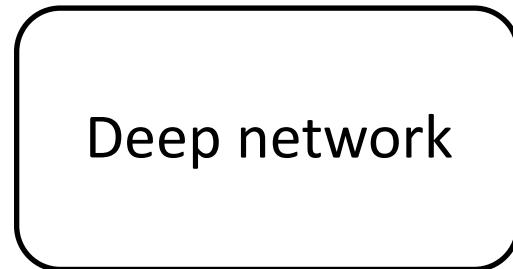


Tackling different tasks

- When working with images, one does not necessarily want to simply classify them
- E.g., semantic segmentation is the task of assigning a label to every pixel in the input image



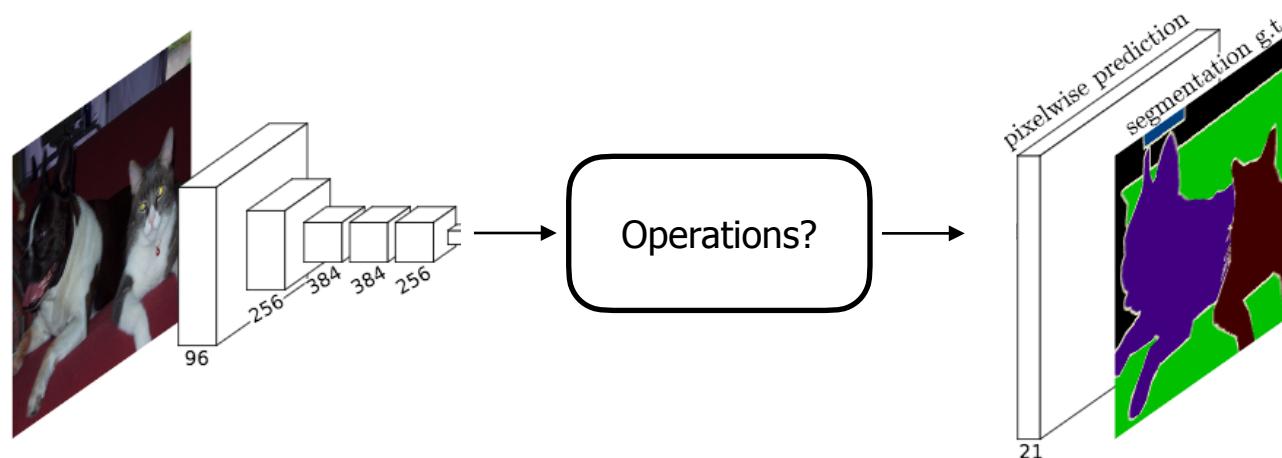
Input



Output

CNNs for semantic segmentation

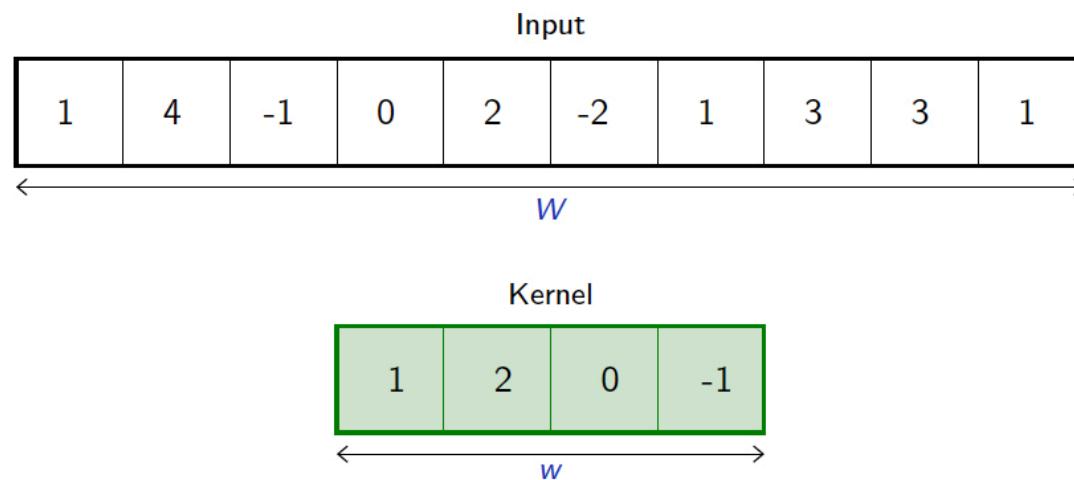
- Problem:
 - With convolutions (without padding) and pooling layers, the size of the feature maps decreases at every layer
 - With the operations seen before, we have no way of increasing this size back to the original resolution



- Solution: Transposed convolutions

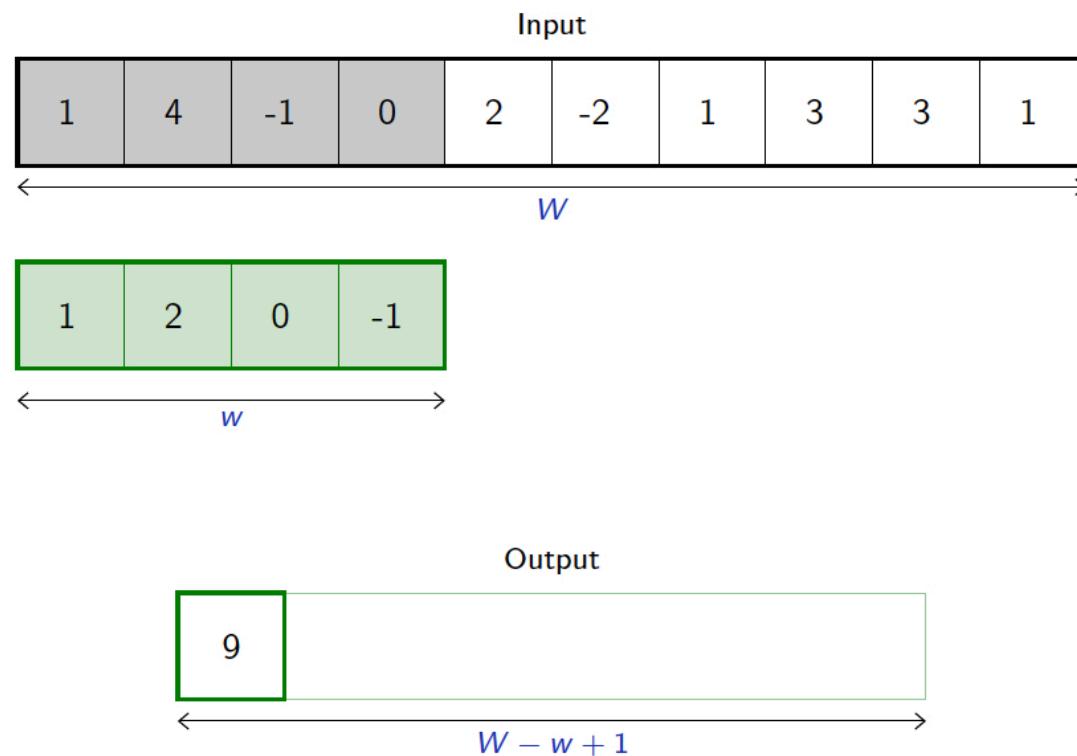
1D convolution

- Before we look into transposed convolutions, let us look at a normal, 1D convolution



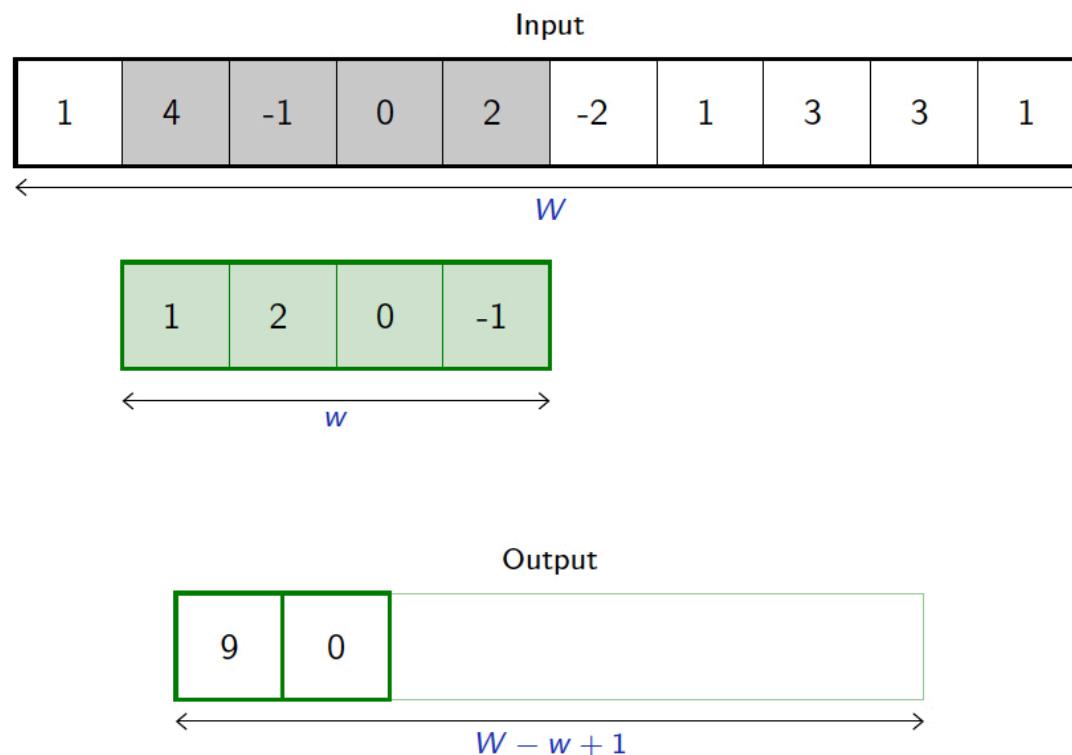
1D convolution

- Before we look into transposed convolutions, let us look at a normal, 1D convolution



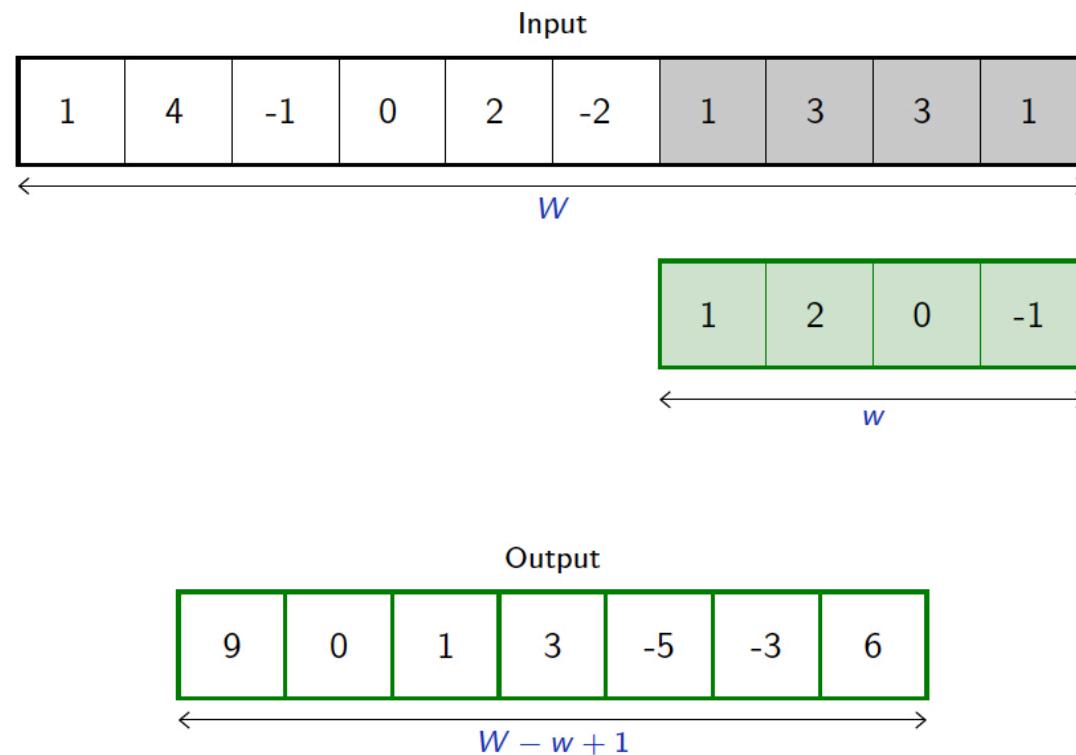
1D convolution

- Before we look into transposed convolutions, let us look at a normal, 1D convolution



1D convolution

- Before we look into transposed convolutions, let us look at a normal, 1D convolution



1D convolution

- In essence, a 1D convolution maps a vector (of dimension W in the example) to another, smaller vector (of dimension $W - w + 1$ for a kernel of size w)
- This can also be achieved via a matrix product

$$\mathbf{z} = \begin{bmatrix} 9 \\ 0 \\ 1 \\ 3 \\ -5 \\ -3 \\ 6 \end{bmatrix} = \begin{bmatrix} 1 & 2 & 0 & -1 \\ 0 & 1 & 2 & 0 & -1 \\ 0 & 0 & \ddots & & \\ & & & \ddots & \\ & & & & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 4 \\ -1 \\ 0 \\ 2 \\ -2 \\ 1 \\ 3 \\ 3 \\ 1 \end{bmatrix}$$

Kernel

The diagram illustrates the computation of a 1D convolution. On the left, the input vector \mathbf{z} is shown as a column vector with elements [9, 0, 1, 3, -5, -3, 6]. To its right is the kernel matrix, which has dimensions $W \times W - w + 1$. The kernel is highlighted with red ovals. Red arrows point from the input vector to the kernel matrix, indicating the receptive field of each output element. The result of the multiplication is a column vector on the far right.

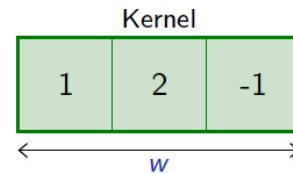
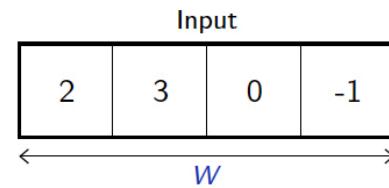
Transposed convolution

- Since a convolution can be represented by a rectangular matrix going from dimension W to dimension $W - w + 1$, increasing the dimensionality can be achieved by using the transposed matrix

$$\begin{bmatrix} w_1 & w_2 & w_3 & 0 & 0 & 0 \\ 0 & w_1 & w_2 & w_3 & 0 & 0 \\ 0 & 0 & w_1 & w_2 & w_3 & 0 \\ 0 & 0 & 0 & w_1 & w_2 & w_3 \end{bmatrix}^T = \begin{bmatrix} w_1 & 0 & 0 & 0 \\ w_2 & w_1 & 0 & 0 \\ w_3 & w_2 & w_1 & 0 \\ 0 & w_3 & w_2 & w_1 \\ 0 & 0 & w_3 & w_2 \\ 0 & 0 & 0 & w_3 \end{bmatrix}$$

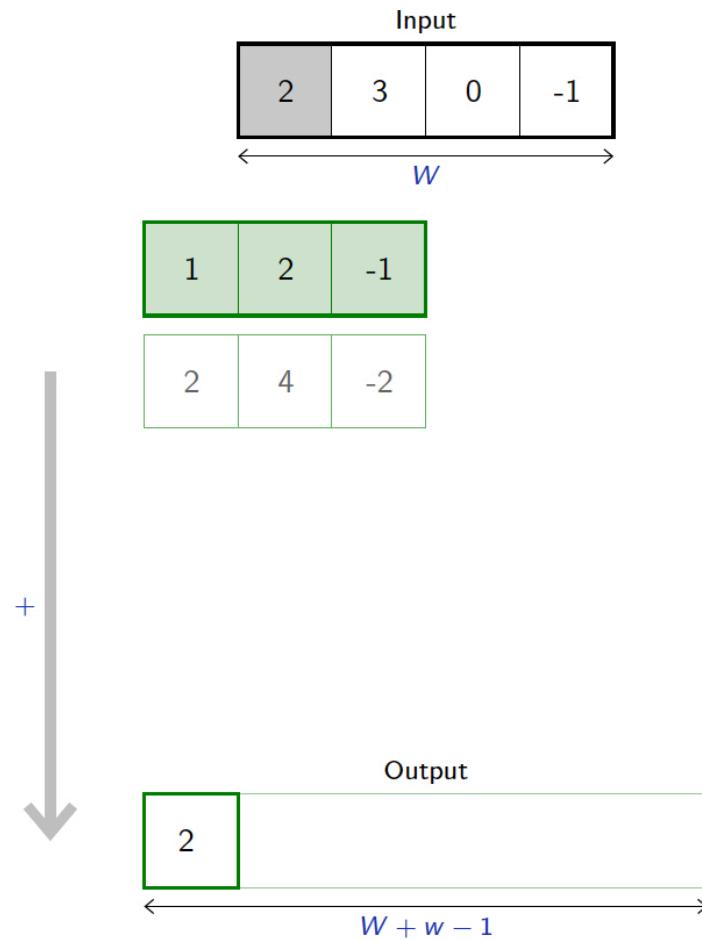
Transposed convolution: Example

- Let us look at a 1D transposed convolution



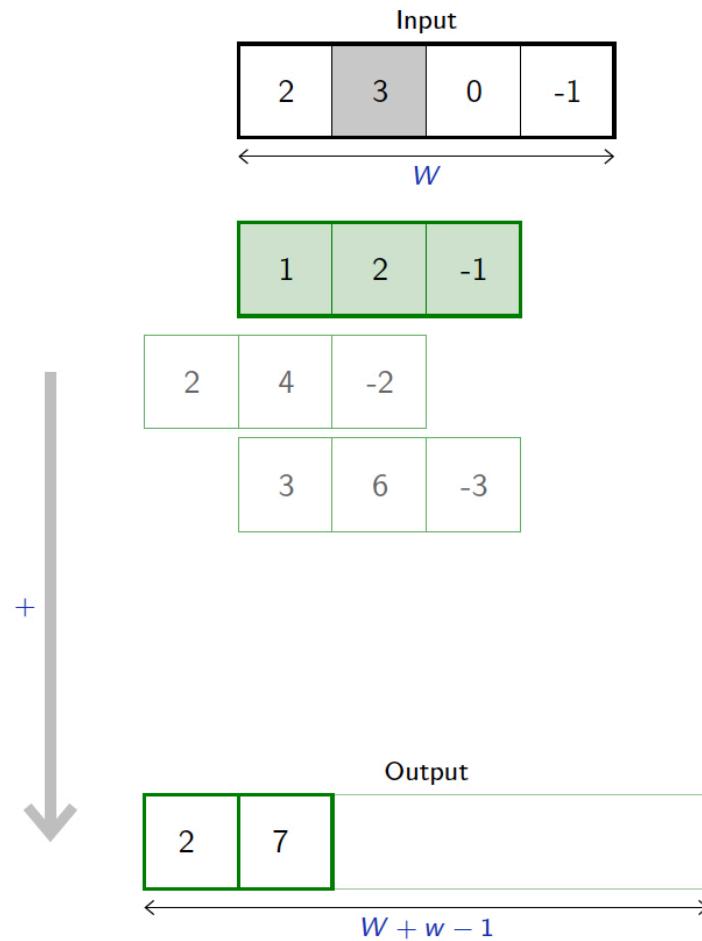
Transposed convolution: Example

- Let us look at a 1D transposed convolution



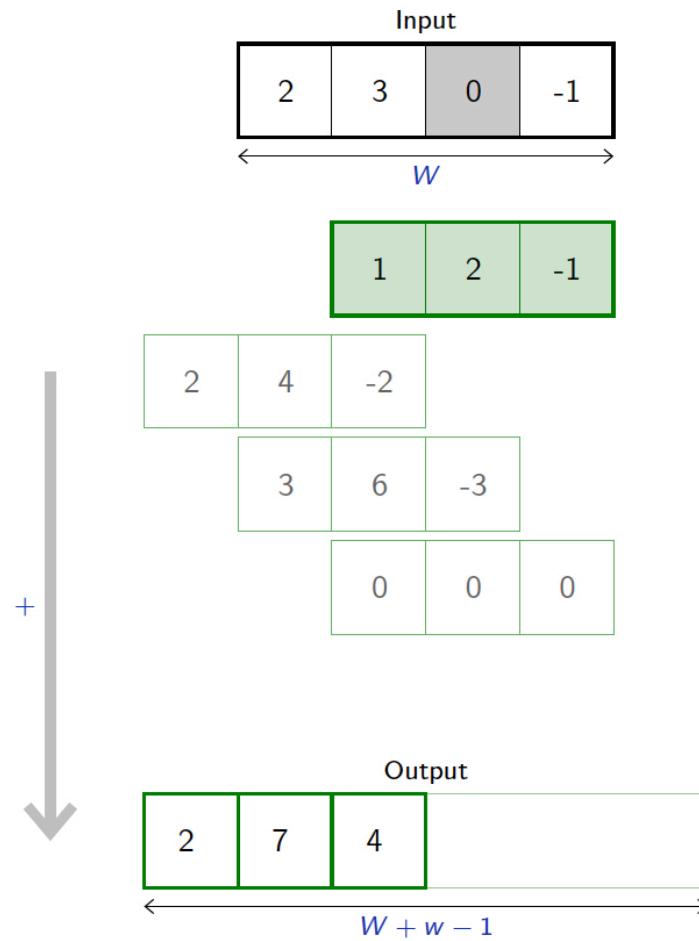
Transposed convolution: Example

- Let us look at a 1D transposed convolution



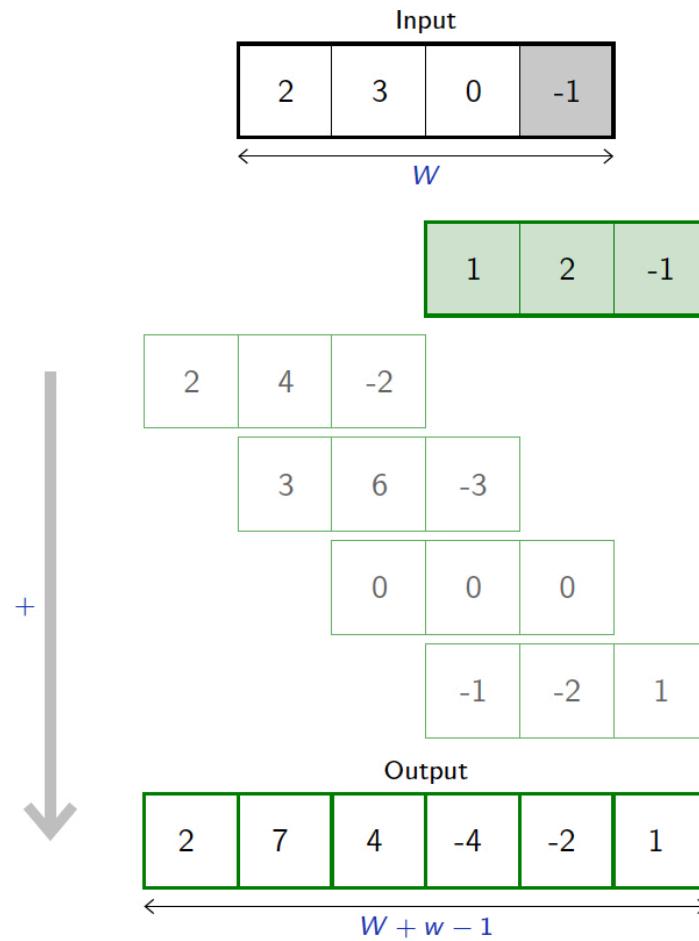
Transposed convolution: Example

- Let us look at a 1D transposed convolution



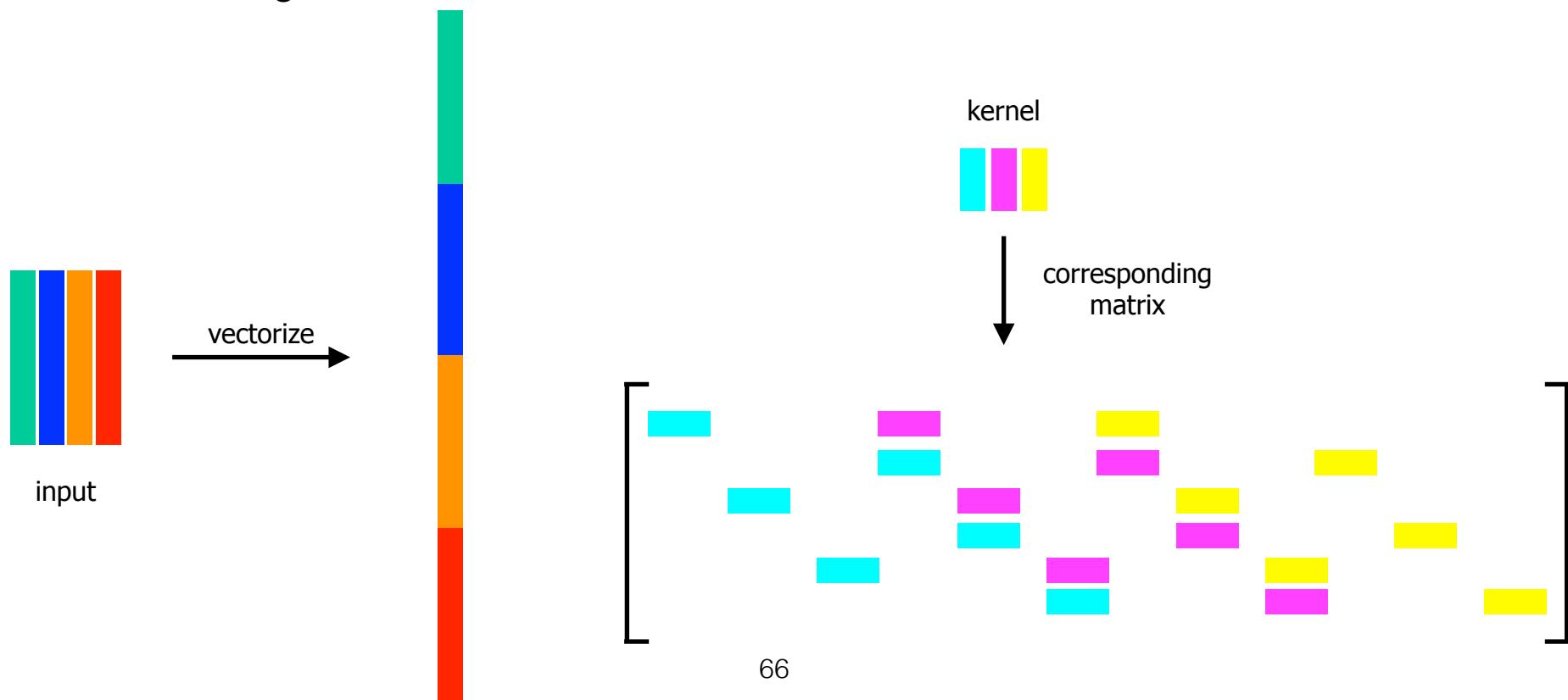
Transposed convolution: Example

- Let us look at a 1D transposed convolution



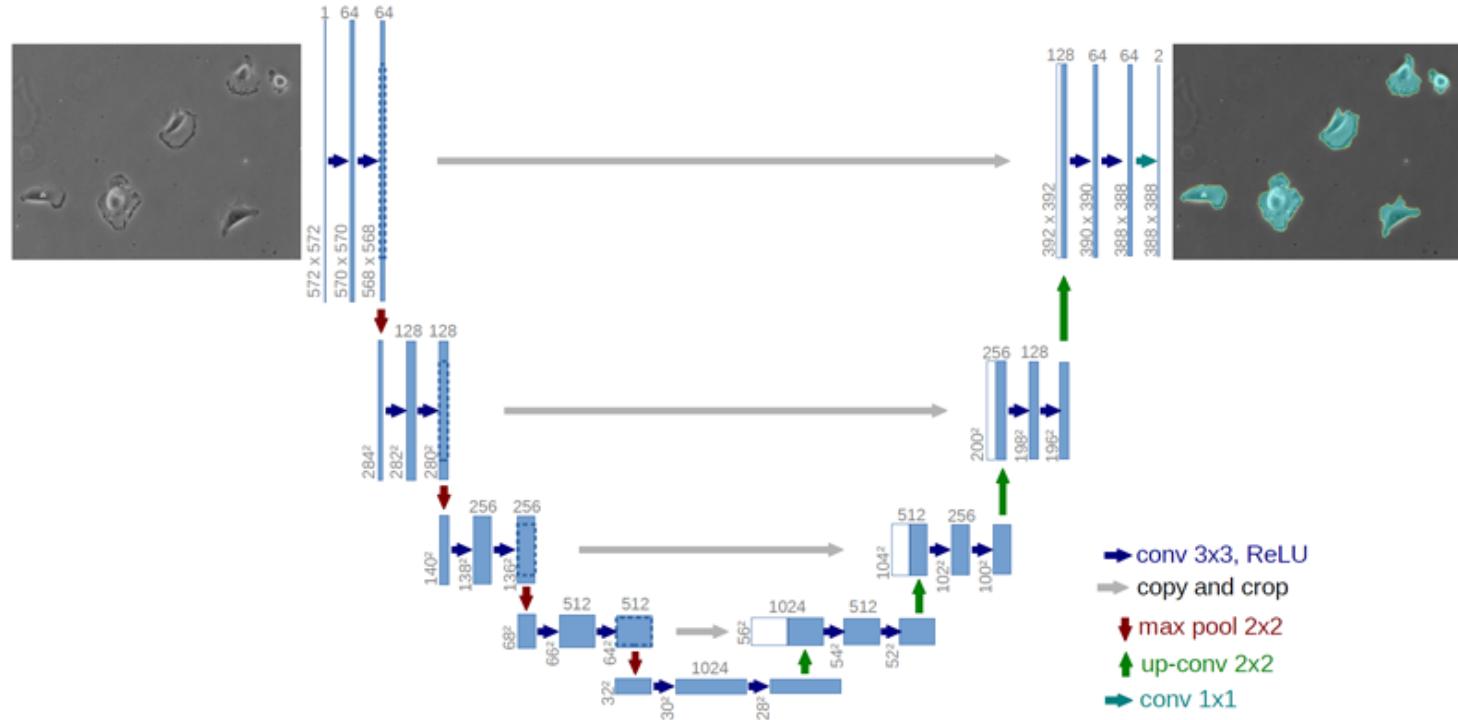
2D convolution as matrix product

- A similar matrix representation can be obtained for 2D convolutions
 - The input is vectorized, e.g., by concatenating its columns
 - The matrix is obtained by spreading the different columns of the kernel at the matching locations



Fully-convolutional networks

- Stacking transposed convolutions after regular ones lets us go back to the original spatial resolution
 - E.g., for binary semantic segmentation, the output is a binary mask of the same size as the image (here overlaid on top of the input image)
 - Ronneberger et al., U-Net: Convolutional Networks for Biomedical Image Segmentation, MICCAI 2015

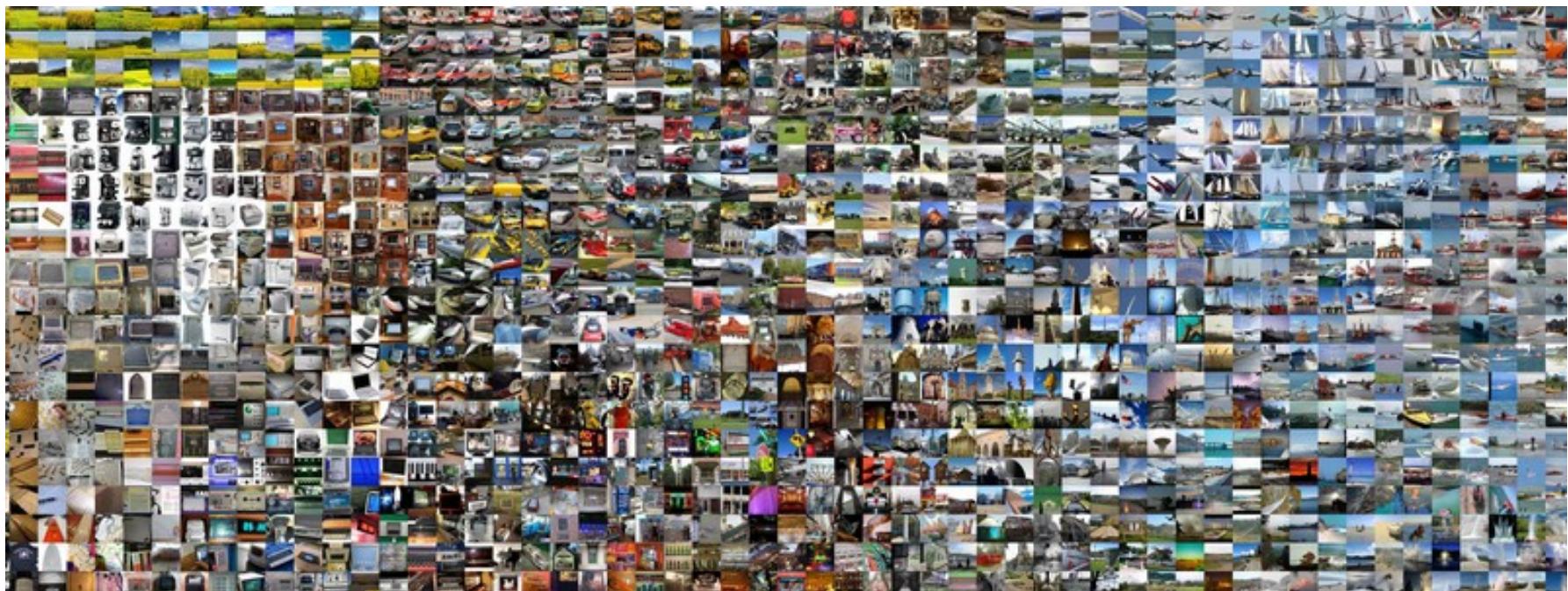


Semantic segmentation demo

- <http://deepscore.cs.uni-freiburg.de>

Deep Learning: The key to the success

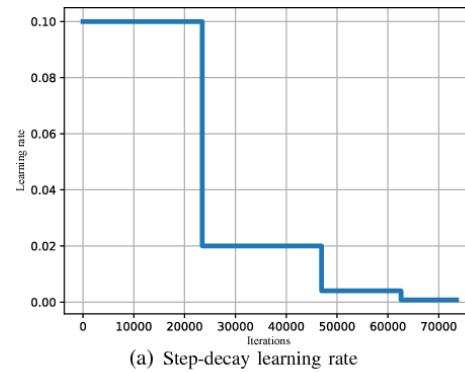
- Big data
 - E.g., ImageNet, Russakovsky et al., 2015



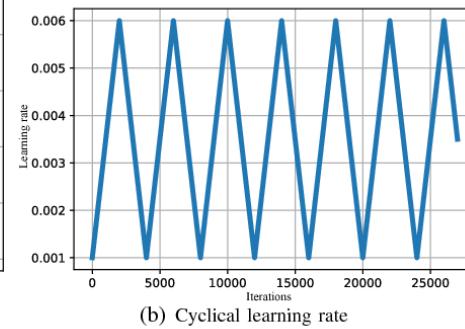
- GPUs

Deep Learning: Tricks of the trade

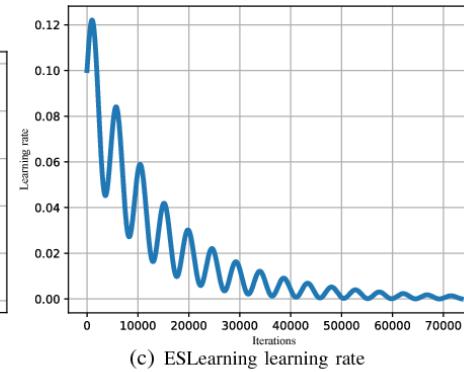
- Pre-training
 - People often start with a standard architecture trained on ImageNet
- Learning rate scheduling
 - Decrease the learning rate after a pre-defined number of epochs
 - Use a cyclic learning rate



(a) Step-decay learning rate



(b) Cyclical learning rate



(c) ESLearning learning rate

Figure from An et al., VCIP 2017

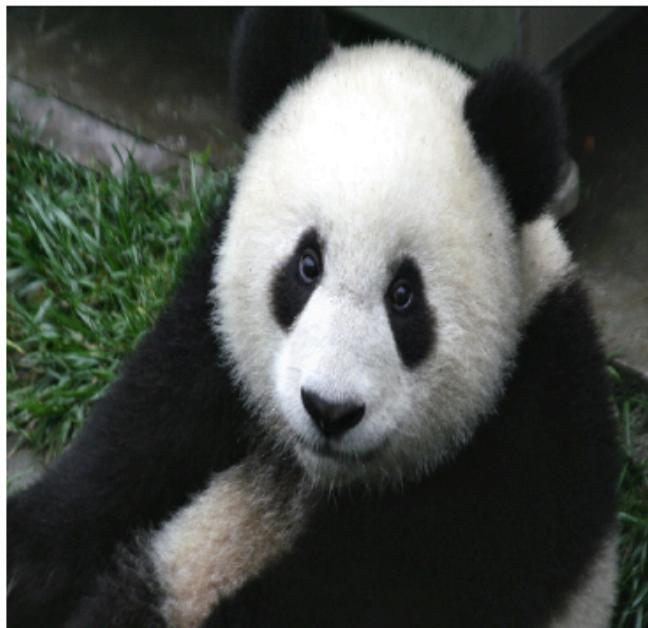
- Data augmentation

- E.g., for images, flip, rotate, affine transform,...

Despite the success...

- Li & Li, Adversarial Examples Detection in Deep Networks with Convolutional Filter Statistics, 2016

Giant Panda (99.32% confidence)

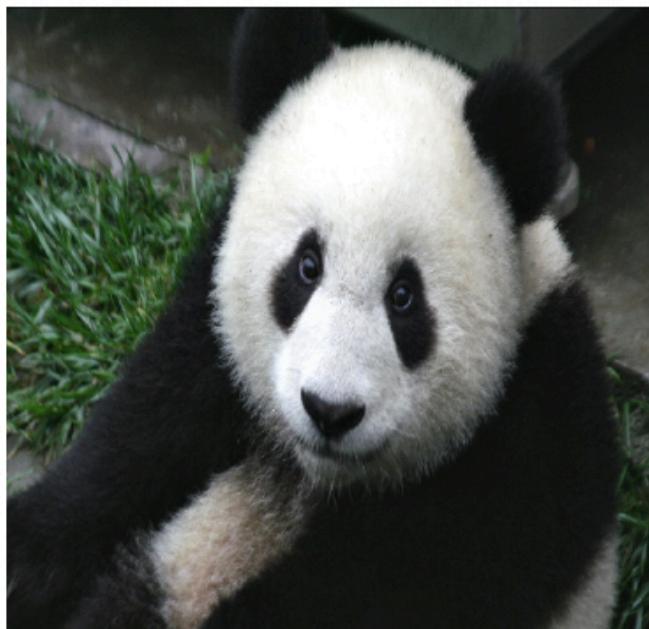


I

Despite the success...

- Li & Li, Adversarial Examples Detection in Deep Networks with Convolutional Filter Statistics, 2016

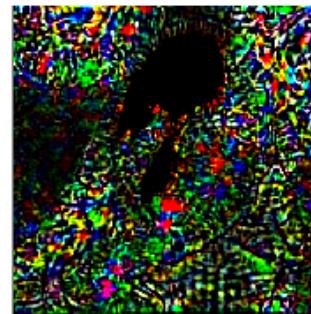
Giant Panda (99.32% confidence)



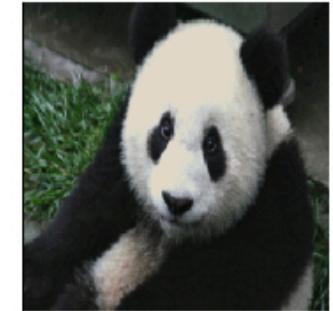
I

+0.03

ΔI



Shark (93.89% confidence)

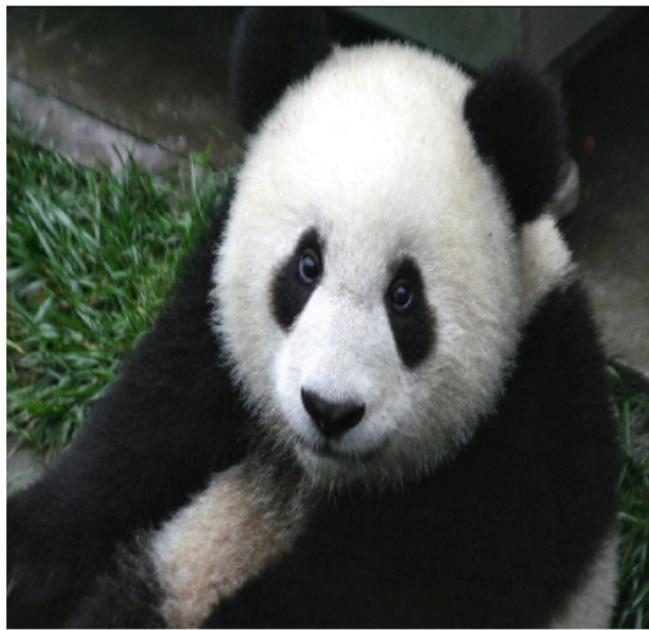


=

Despite the success...

- Li & Li, Adversarial Examples Detection in Deep Networks with Convolutional Filter Statistics, 2016

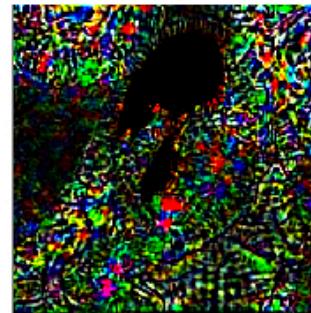
Giant Panda (99.32% confidence)



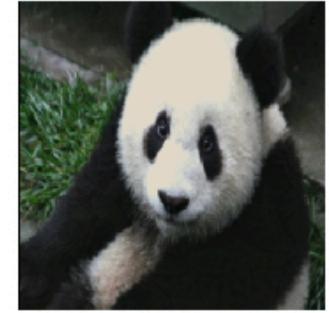
I

+0.03

ΔI



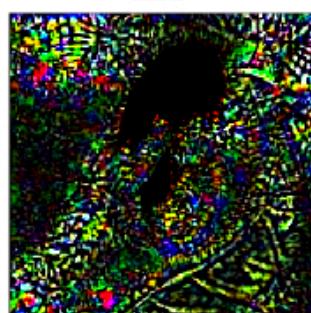
Shark (93.89% confidence)



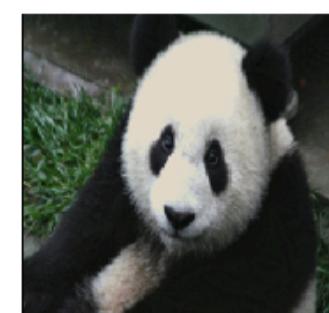
=

+0.03

ΔI



Goldfish (95.15% confidence)



=

Discussion

- Intuitively, can you explain what is happening with such adversarial examples?

Lecture 12: Dimensionality Reduction

Recap: Artificial Neural Networks

- A simple artificial neural network

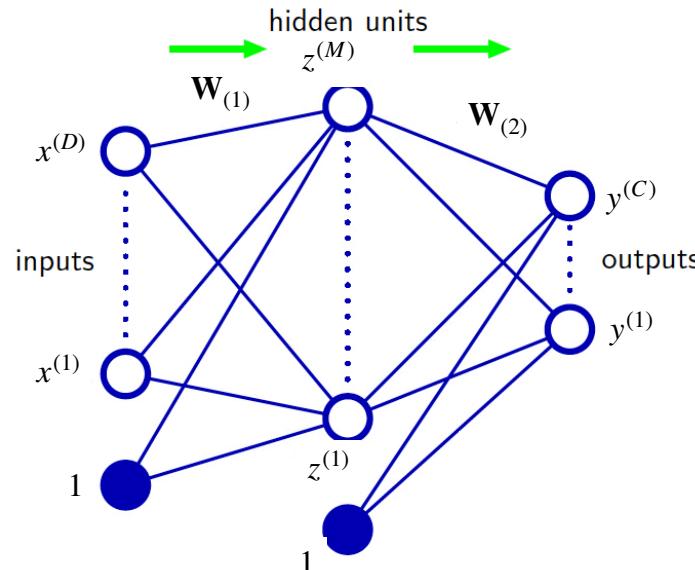
- We first map the input \mathbf{x} to a different representation (hidden units)

$$\mathbf{z} = f_{(1)}(\mathbf{W}_{(1)}^T \mathbf{x})$$

- The outputs are then obtained by another mapping

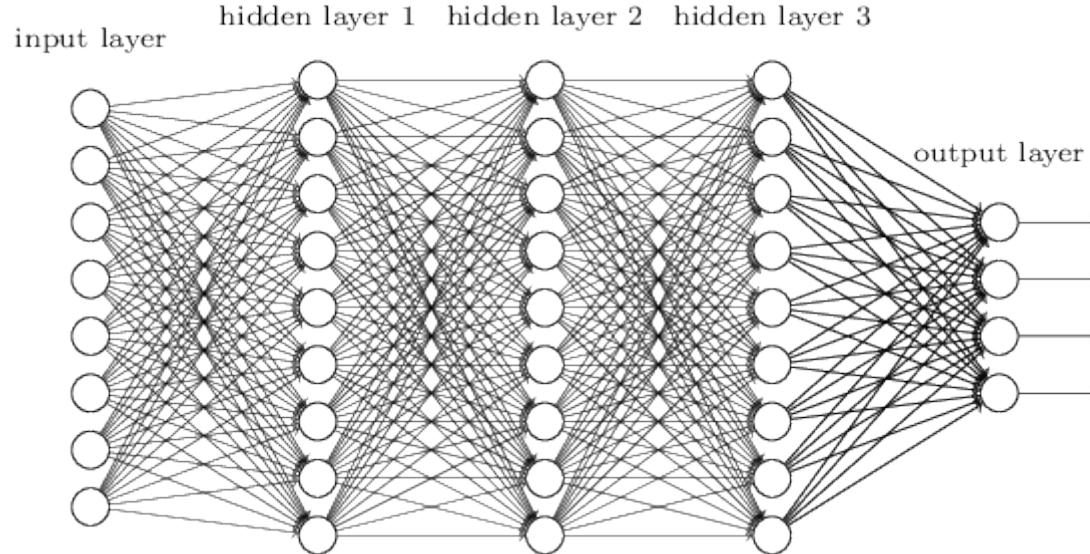
$$\mathbf{y} = f_{(2)}(\mathbf{W}_{(2)}^T \mathbf{z})$$

- Each mapping relies on a nonlinear function $f_{(j)}(\cdot)$, called *activation function*



Recap: Multilayer Perceptron

- With a single hidden layer, one can already model complicated functions
- Why stop at just one hidden layer?



- Slightly misleading name:
 - The perceptron uses a step function as activation function
 - MLPs typically rely on continuous functions (e.g., sigmoid, ReLU)

Recap: Working with images

- Treating an image as a big vector seems counter-intuitive
 - An image of a reasonable size yields a huge vector, thus huge amount of parameters in an MLP
 - Small image regions have similar property and should not be processed independently
 - A translation should not affect the results



$$\mathbf{x}_i \in \mathbb{R}^{536 \cdot 356 \cdot 3} = \mathbb{R}^{572448}$$

- Can we reduce the number of parameters and improve robustness?

Recap: Convolutional layer

- Fewer parameters than fully-connected layers
 - the parameters are shared across different spatial locations
- Equivariance to translation

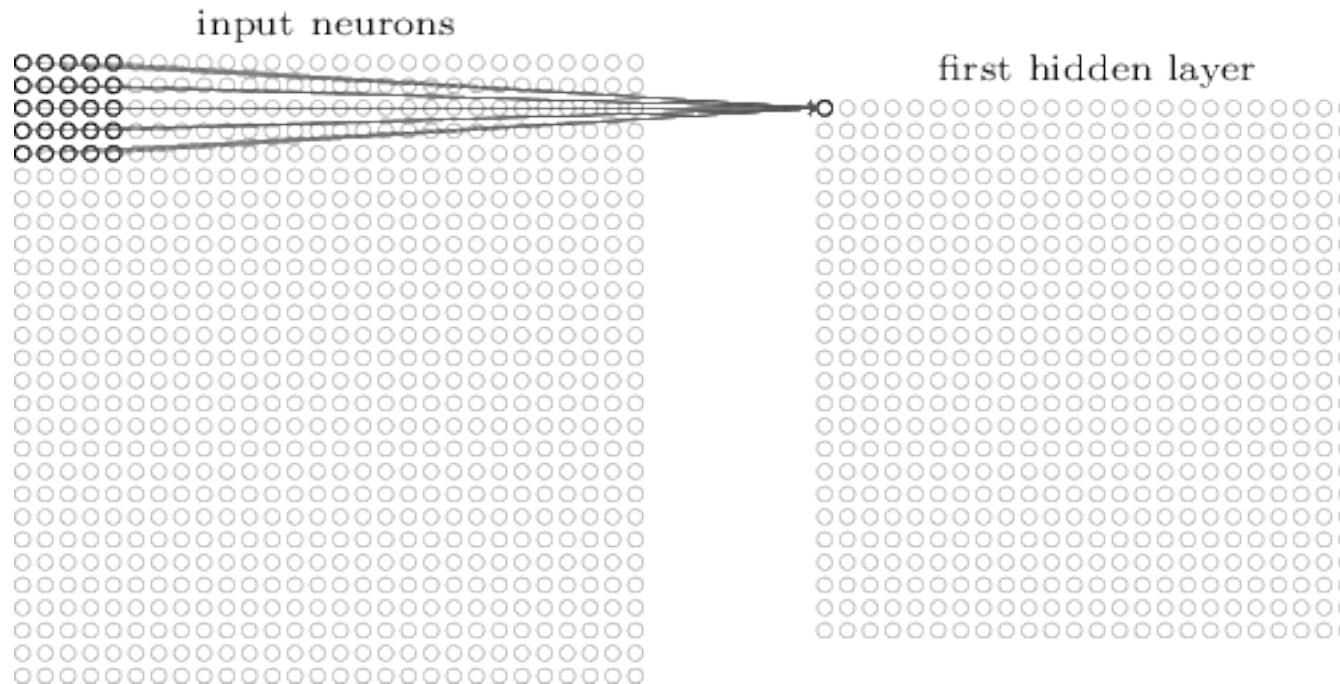
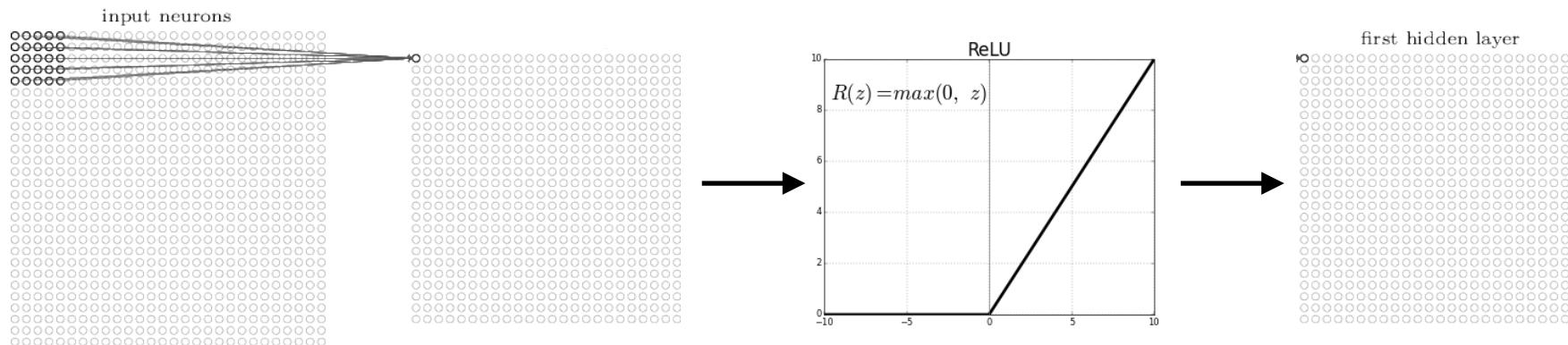


Figure from M. Nielsen's book
<http://neuralnetworksanddeeplearning.com>

Recap: Convolutional layer: Nonlinearity

- Note that each convolutional output is passed in an activation function. So, in fact, we have



Recap: Convolutional layer

- We can then use multiple filters to create multiple channels (3 in the example below)

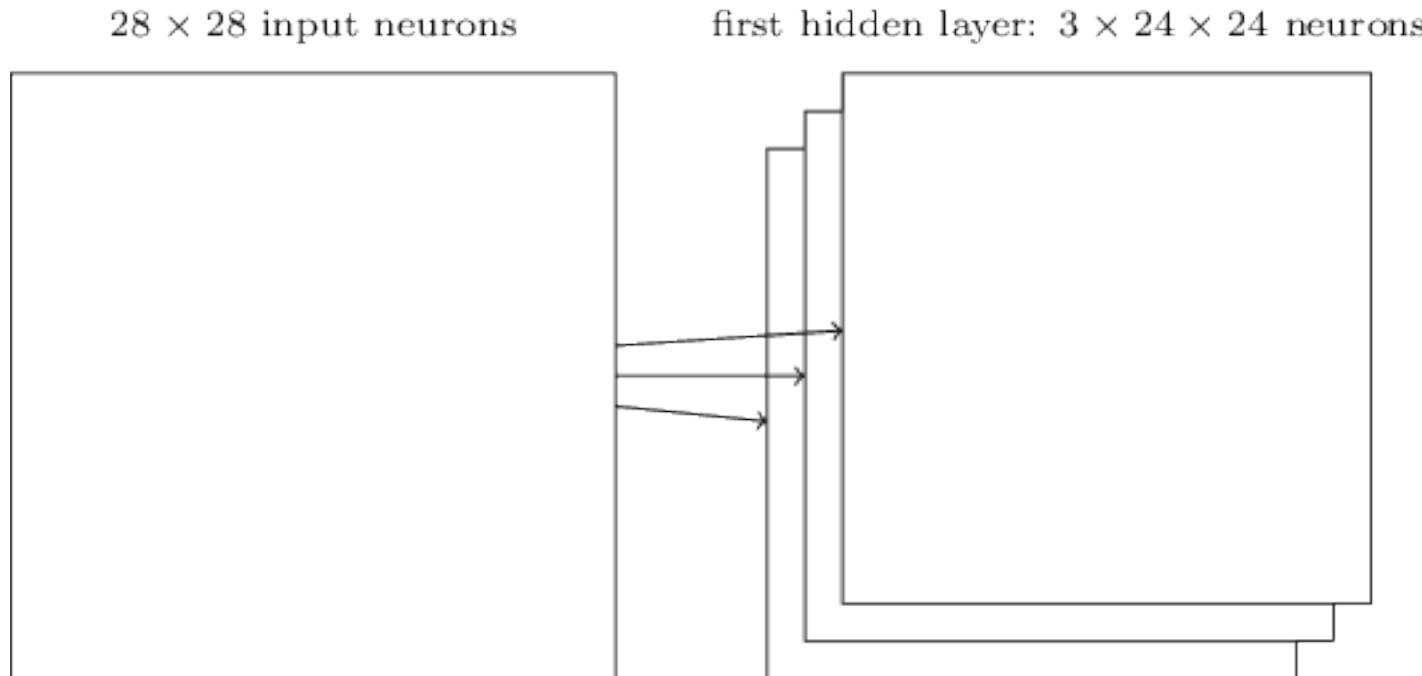


Figure from M. Nielsen's book
<http://neuralnetworksanddeeplearning.com>

Recap: Pooling layer

- From local information to global a more global representation
- Different pooling strategies:
 - Max pooling
 - Average pooling

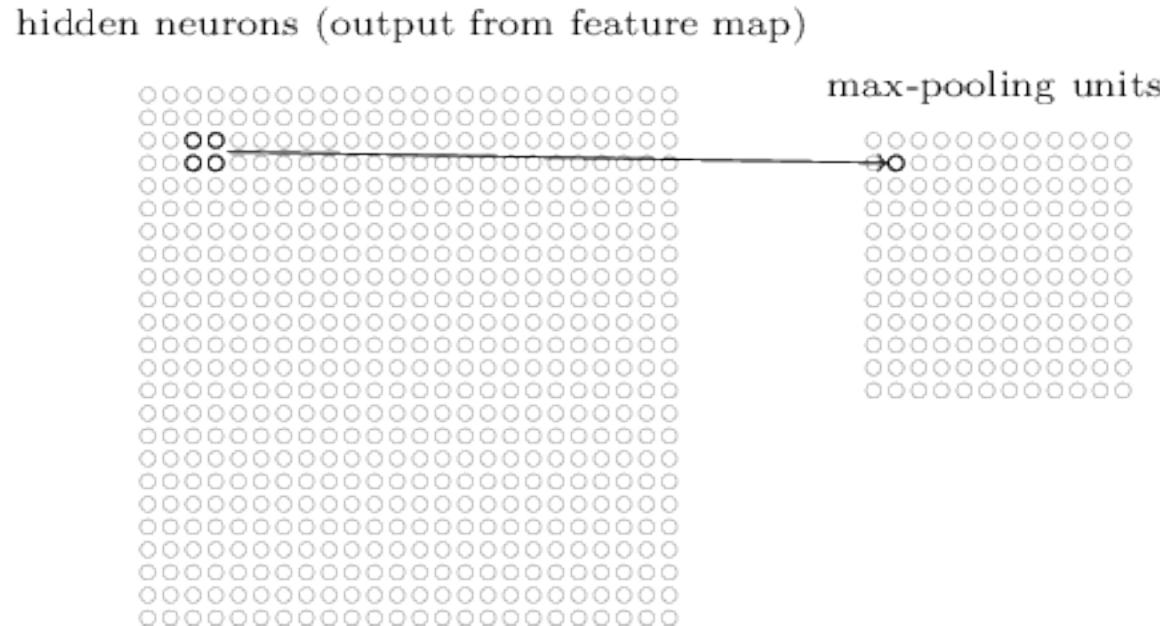
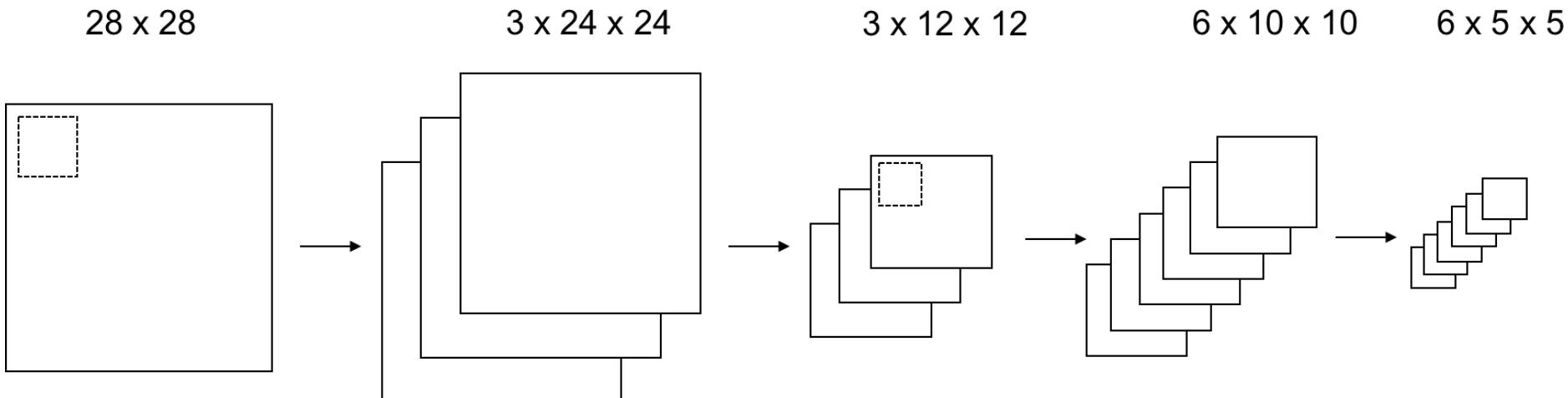


Figure from M. Nielsen's book
<http://neuralnetworksanddeeplearning.com>

Recap: Convolutional Neural Network (CNN)

- We can then stack multiple such operations
 - The number of channels can vary (below: first 3, then 6)
 - The size of the filters can vary (below: first 5×5 , then 3×3)



Recap: Convolutional Neural Network (CNN)

- Eventually, one typically wants to output a vector, e.g.,
 - a vector of class probabilities for classification
 - a vector of 3D joint coordinates for human pose estimation
- This can be achieved by appending fully-connected layer(s)
 - This requires vectorizing the last convolutional output first (e.g., $3 \times 12 \times 12 \rightarrow 432$ -dimensional vector)

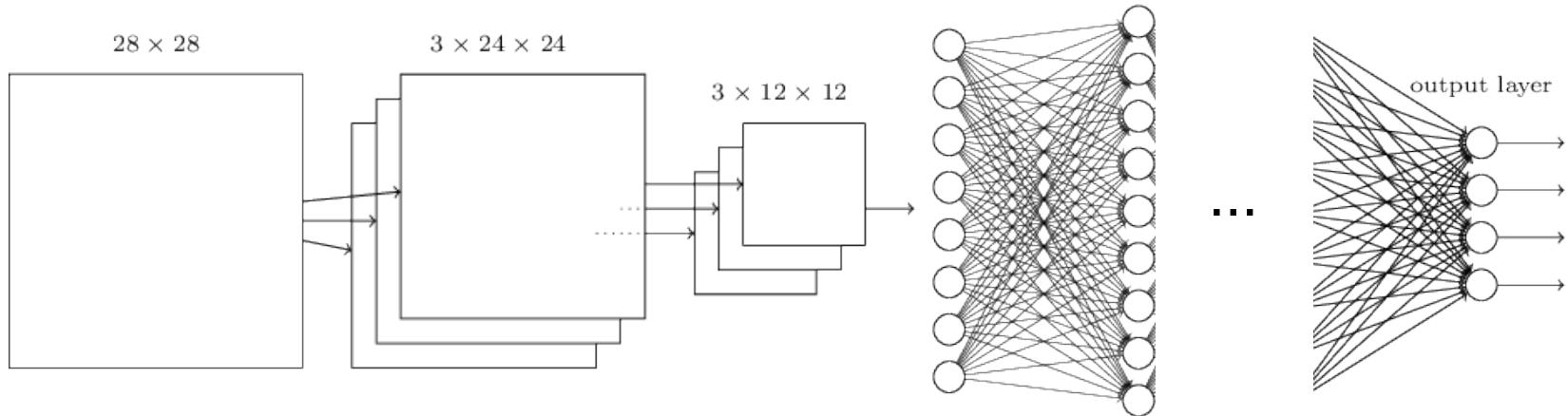


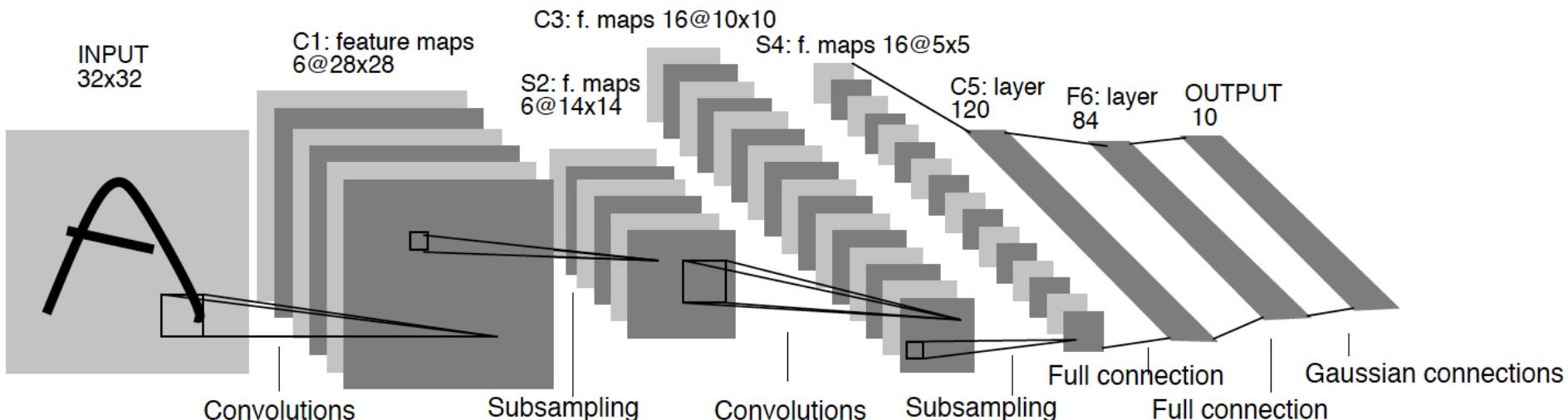
Figure adapted from M. Nielsen's book
<http://neuralnetworksanddeeplearning.com>

Recap: CNN: Learning

- Learning the weights of the CNN layers is done in the same manner as for MLPs
 - Using stochastic gradient descent with mini-batches
- The same idea of error backpropagation that we discussed last time is applicable to convolutional layers
- The gradient of pooling layers depends on the strategy:
 - For average pooling, the gradient is backpropagated to all neurons used during pooling
 - For max pooling, the gradient is backpropagated only to the neuron that corresponded to the maximum value

Exercise

- In the LeNet-5 architecture, how many learnable parameters are there in
 - the first convolutional layer (from Input to C1)?
 - the pooling layer (from C1 to S2)?
 - the second convolutional layer (from S2 to C3)?
 - the last fully-connected layer (from F6 to Output)?

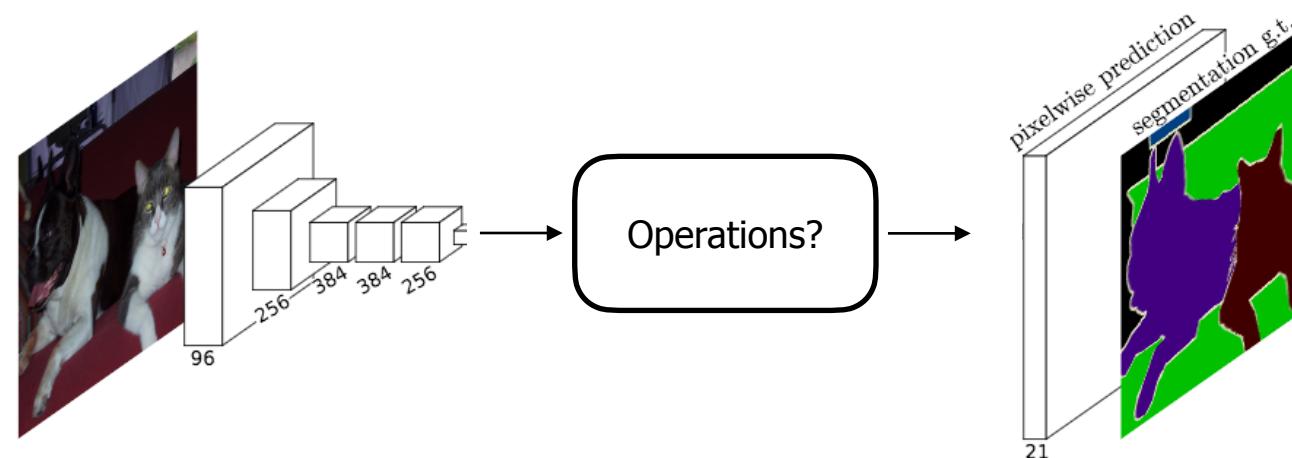


Exercise: Solution

- the first convolutional layer (from Input to C1)?
 - Because the size goes from 32 to 28, we have 5x5 filters (without padding, we “lose” 2 pixels on each side). We have 1 input channel, and 6 output channels. So we have $6 \times 5 \times 5 = 150$ parameters for the convolutions. We also have 1 additional bias per output channel, so 6 additional parameters.
- the pooling layer (from C1 to S2)?
 - There are no learnable parameters in a pooling layer
- the second convolutional layer (from S2 to C3)?
 - Because the size goes from 14 to 10, we have 5x5 filters. We have 6 input channels, and 16 output channels. So we have $16 \times 6 \times 5 \times 5 = 2400$ parameters. Furthermore, we have 16 additional bias parameters.
- the last fully-connected layer (from F6 to Output)?
 - We have 84 input units and 10 output units. Accounting for the bias, by concatenating a 1 to the input, we have $(84+1) \times 10 = 850$ parameters.

Recap: CNNs for semantic segmentation

- Problem:
 - With convolutions (without padding) and pooling layers, the size of the feature maps decreases at every layer
 - With the operations seen before, we have no way of increasing this size back to the original resolution



- Solution: Transposed convolutions

Discussion

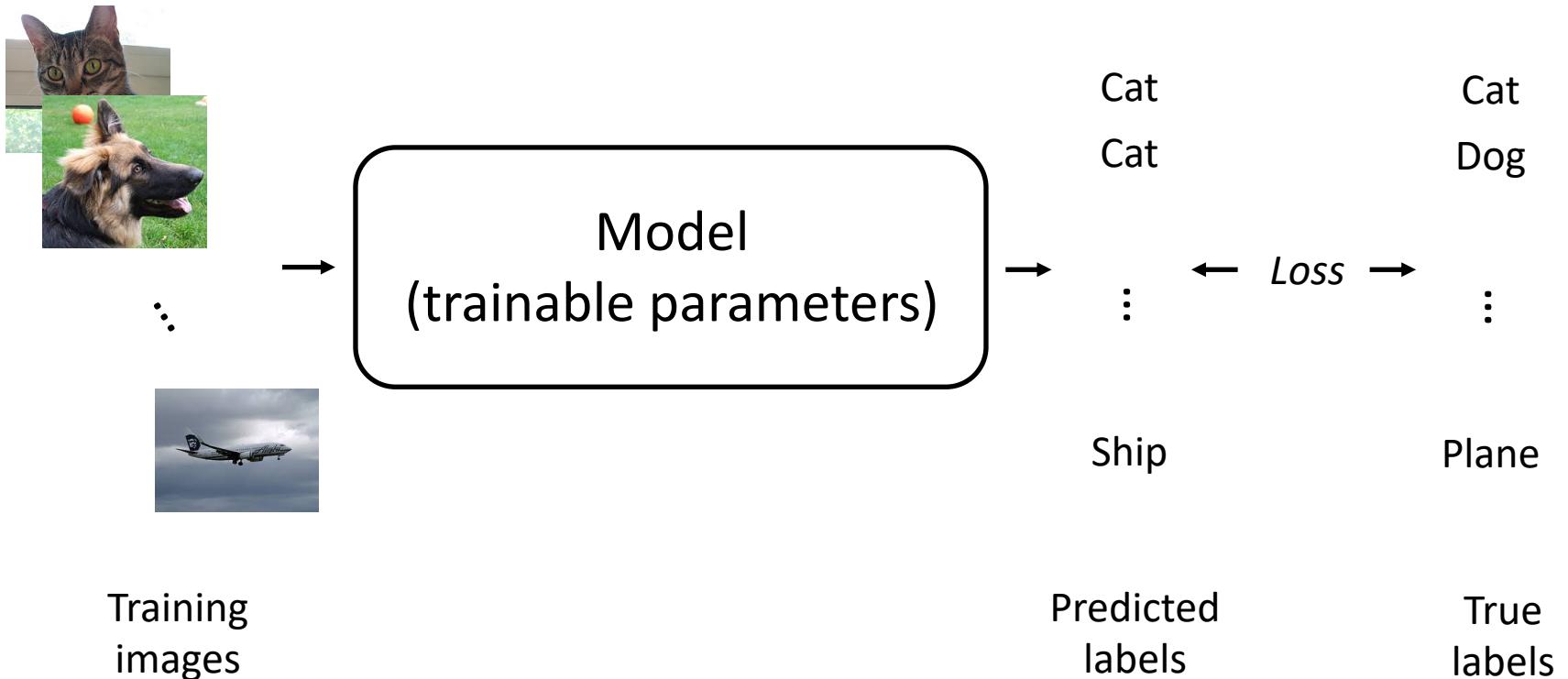
- Intuitively, can you explain what is happening with adversarial examples?
 - Because of the nonlinear transformations performed by deep neural networks, a small change in input values can lead to a large change in hidden representations. Furthermore, because of their many parameters, the decision boundary of deep neural networks is highly nonlinear. This makes it easier to change a hidden representation such that it crosses the decision boundary.

Goals of today's lecture

- Introduce our first unsupervised learning task: dimensionality reduction
- Introduce a linear dimensionality reduction technique, Principal Component Analysis, and its kernel extension
- Introduce autoencoders

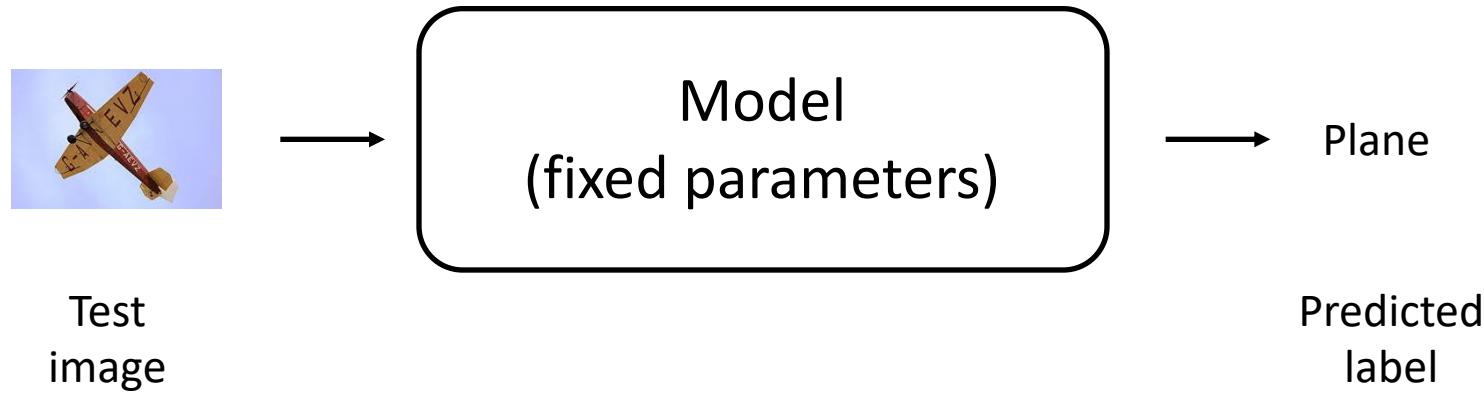
Until now: Supervised learning

- Stage 1: Training: Use data with ground-truth labels to optimize model parameters



Supervised learning

- Stage 2: Testing: Predict the output for a new data sample

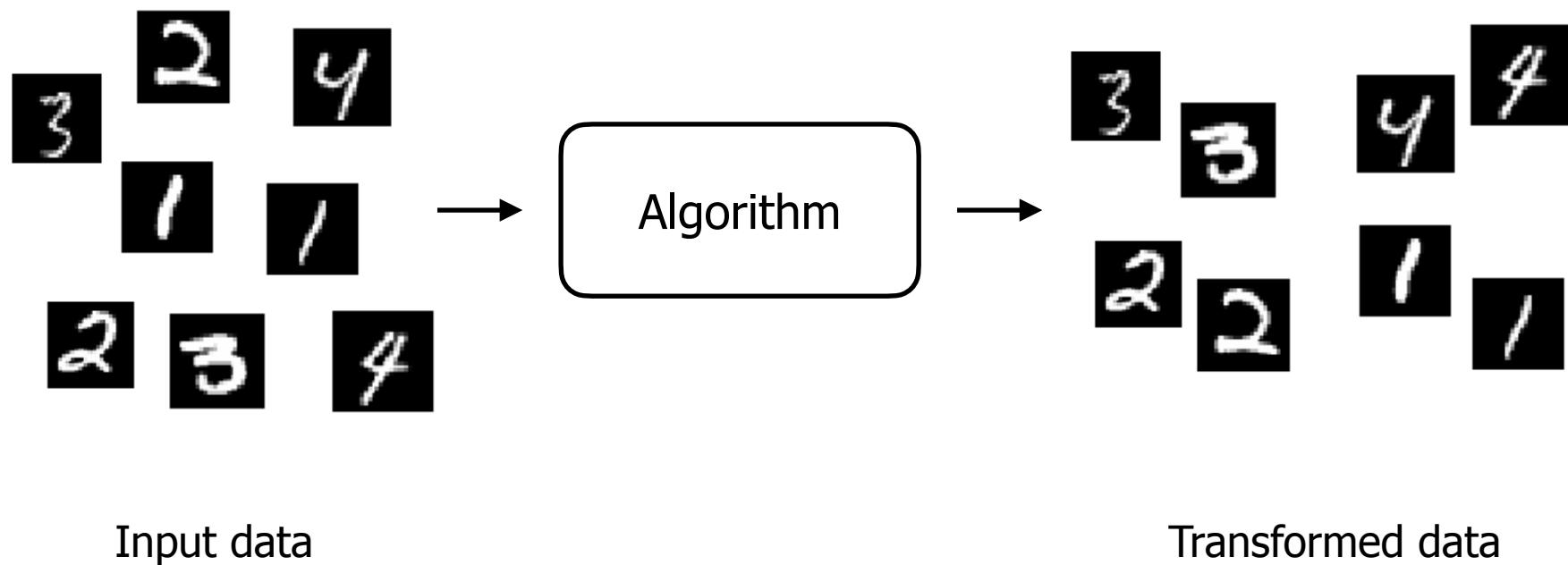


Supervised learning

- Good when you have
 - a concrete prediction task to solve
 - labeled data
- Not so good when you
 - do not have labeled data
 - want to get an understanding of the data

Today: No prediction

- A single stage: Transform the data for further analysis



Today: Dimensionality reduction

- Example: Data visualization
 - <http://colah.github.io/posts/2014-10-Visualizing-MNIST/>
 - <https://experiments.withgoogle.com/ai/drum-machine/view/>

Today: Dimensionality reduction

- Example: Data size reduction
 - E.g., vectorizing an image yields a huge vector



$$\mathbf{x}_i \in \mathbb{R}^{536 \cdot 356 \cdot 3} = \mathbb{R}^{572448}$$

- Dealing with such large data is cumbersome
- Can we compress it while keeping the relevant information?

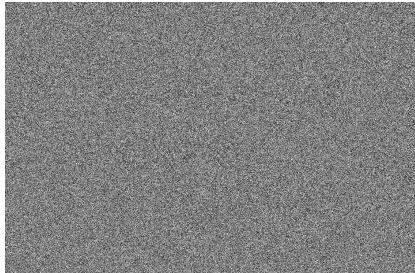
Dimensionality reduction: Intuition

- This image lies in a 572448 dimensional space



$$\mathbf{x}_i \in \mathbb{R}^{536 \cdot 356 \cdot 3} = \mathbb{R}^{572448}$$

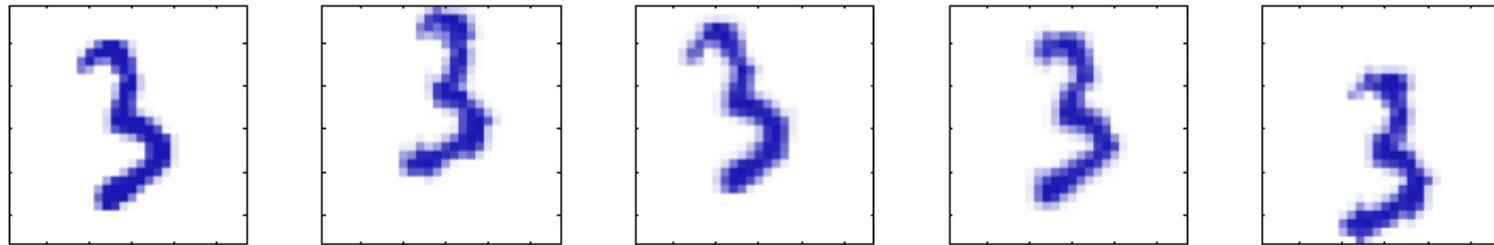
- So does this one
 - But we are much less likely to encounter it in the real world...



$$\mathbf{x}_j \in \mathbb{R}^{536 \cdot 356 \cdot 3} = \mathbb{R}^{572448}$$

Another example (Bishop Chapter 12)

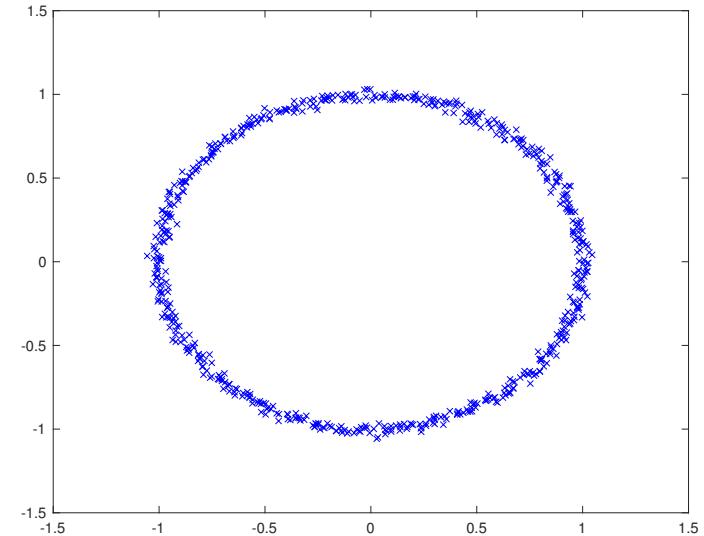
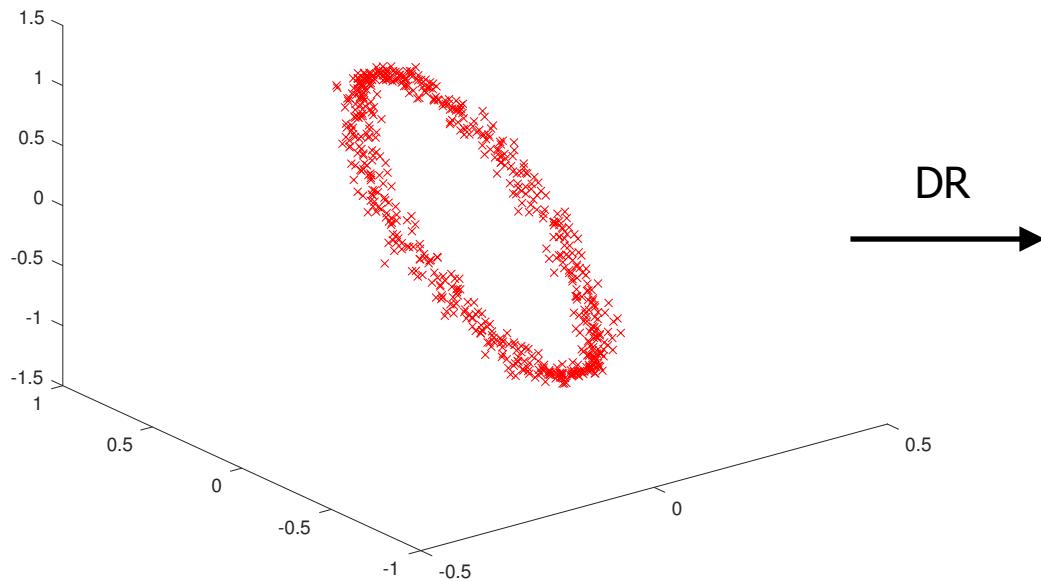
- Images created by translating and rotating the same digit 3



- The data lies in a high-dimensional space (32×32)
- But there are only 3 degrees of freedom (2 translations and 1 rotation)
- Inherently, this data lies in a 3-dimensional space
- In practice, the data often lies in a much lower-dimensional space, referred to as manifold

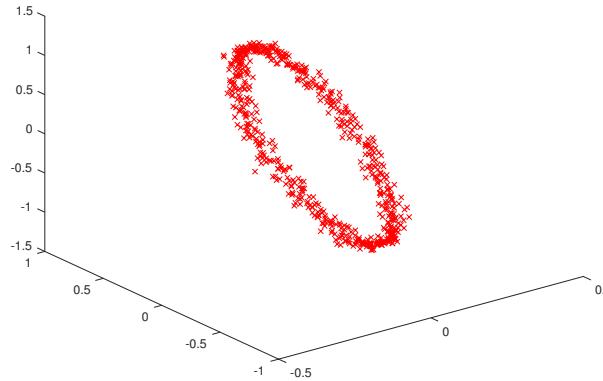
Dimensionality reduction

- Dimensionality reduction is the task of
 - discovering the data manifold
 - getting a low-dimensional representation of the data
 - sometimes at some loss of information (hopefully noise)

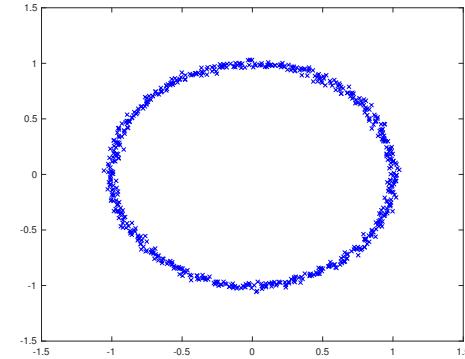


Dimensionality reduction

- $\mathbf{x}_i \in \mathbb{R}^D$: High-dimensional data sample
- $\mathbf{y}_i \in \mathbb{R}^d$: Low-dimensional representation



x



y

- Our goal is to find a mapping $\mathbf{y}_i = f(\mathbf{x}_i)$
- How about a linear one $\mathbf{y}_i = \mathbf{W}^T \mathbf{x}_i$

Principal Component Analysis (PCA) (Bishop 12.1)

- Pearson (1901), Hotelling (1933)
- Given N samples $\{\mathbf{x}_i\}$, PCA yields a projection of the form

$$\mathbf{y}_i = \mathbf{W}^T(\mathbf{x}_i - \bar{\mathbf{x}}) \quad \text{s.t. } \mathbf{W}^T \mathbf{W} = \mathbf{I}_d$$

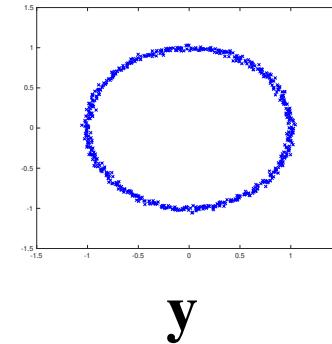
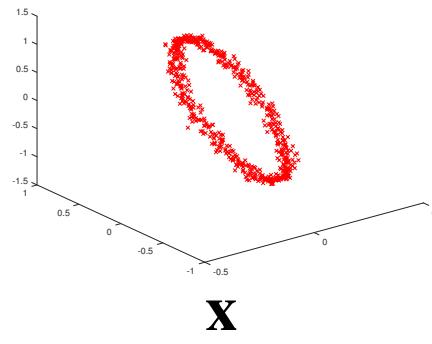
where $\bar{\mathbf{x}}$ is the mean of the data, i.e.,

$$\bar{\mathbf{x}} = \frac{1}{N} \sum_{i=1}^N \mathbf{x}_i$$

- The data is unsupervised:
 - What do we want this projection to achieve?

PCA: Objective

- We aim to keep most of the “important” signal
- Yet, we want to remove the noise



- This can be achieved by finding directions that have a large variance, i.e., for the j^{th} output dimension, we want to maximize

$$\text{var}(\{y_i^{(j)}\}) = \frac{1}{N} \sum_{i=1}^N (y_i^{(j)} - \bar{y}^{(j)})^2$$

where $\bar{y}^{(j)}$ is the mean of the j^{th} dimension of the data after projection

PCA: Variance maximization

- Let us look at the projection to a 1D space
 - . That is, we use a vector D -dimensional $\mathbf{w}_{(1)}$, s.t., $\mathbf{w}_{(1)}^T \mathbf{w}_{(1)} = 1$, instead of a matrix $\mathbf{W} \in \mathbb{R}^{D \times d}$
- In this case, the mean of the data after projection is

$$\begin{aligned}\bar{y} &= \frac{1}{N} \sum_{i=1}^N y_i = \frac{1}{N} \sum_{i=1}^N \mathbf{w}_{(1)}^T (\mathbf{x}_i - \bar{\mathbf{x}}) \\ &= \mathbf{w}_{(1)}^T \left(\frac{1}{N} \sum_{i=1}^N (\mathbf{x}_i - \bar{\mathbf{x}}) \right) = 0\end{aligned}$$

PCA: Variance maximization

- Then, the variance of the data after projection is

$$\begin{aligned}\text{var}(\{y_i\}) &= \frac{1}{N} \sum_{i=1}^N (y_i - \bar{y})^2 = \frac{1}{N} \sum_{i=1}^N \left(\mathbf{w}_{(1)}^T (\mathbf{x}_i - \bar{\mathbf{x}}) - 0 \right)^2 \\ &= \frac{1}{N} \sum_{i=1}^N \left(\mathbf{w}_{(1)}^T (\mathbf{x}_i - \bar{\mathbf{x}}) \right)^2 = \frac{1}{N} \sum_{i=1}^N \mathbf{w}_{(1)}^T (\mathbf{x}_i - \bar{\mathbf{x}}) (\mathbf{x}_i - \bar{\mathbf{x}})^T \mathbf{w}_{(1)} \\ &= \mathbf{w}_{(1)}^T \left(\frac{1}{N} \sum_{i=1}^N (\mathbf{x}_i - \bar{\mathbf{x}}) (\mathbf{x}_i - \bar{\mathbf{x}})^T \right) \mathbf{w}_{(1)} = \mathbf{w}_{(1)}^T \mathbf{C} \mathbf{w}_{(1)}\end{aligned}$$

where \mathbf{C} is the input data covariance matrix

$$\mathbf{C} = \frac{1}{N} \sum_{i=1}^N (\mathbf{x}_i - \bar{\mathbf{x}}) (\mathbf{x}_i - \bar{\mathbf{x}})^T$$

PCA: Variance maximization

- So, ultimately, we seek to solve the problem

$$\max_{\mathbf{w}_{(1)}} \mathbf{w}_{(1)}^T \mathbf{C} \mathbf{w}_{(1)}$$

$$\text{s.t. } \mathbf{w}_{(1)}^T \mathbf{w}_{(1)} = 1$$

- Following what we saw in previous lectures, we can write the Lagrangian of this problem:

$$L(\mathbf{w}_{(1)}, \lambda_1) = \mathbf{w}_{(1)}^T \mathbf{C} \mathbf{w}_{(1)} + \lambda_1 (1 - \mathbf{w}_{(1)}^T \mathbf{w}_{(1)})$$

PCA: Variance maximization

- Setting the gradient of the Lagrangian to 0 yields

$$\mathbf{C}\mathbf{w}_{(1)} = \lambda_1 \mathbf{w}_{(1)}$$

- This is the definition of an eigenvector
- So $\mathbf{w}_{(1)}$ must be an eigenvector of \mathbf{C} , with eigenvalue λ_1
- The question then is: Which eigenvector?

PCA: Variance maximization

- Multiplying both sides of the eigenvector equation from the left by $\mathbf{w}_{(1)}^T$ yields

$$\mathbf{w}_{(1)}^T \mathbf{C} \mathbf{w}_{(1)} = \lambda_1 \mathbf{w}_{(1)}^T \mathbf{w}_{(1)} = \lambda_1$$

because of the constraint on $\mathbf{w}_{(1)}$

- The resulting term on the lefthand side is the variance of the projected data
- Since we seek to maximize it, we should take $\mathbf{w}_{(1)}$ as the eigenvector corresponding to the largest eigenvalue λ_1
 - Because the equation above shows that the variance is equal to the eigenvalue

PCA: Dealing with more than 1D projections

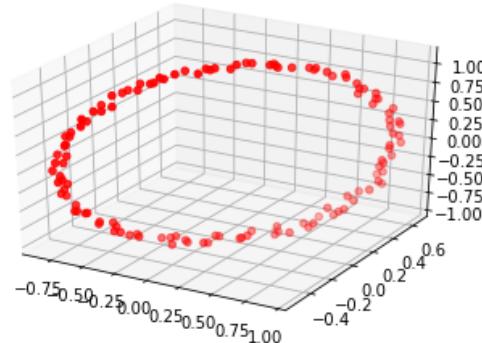
- To obtain an output representation that is more than 1D, i.e., $d > 1$, we can perform recursively
 - The second projection vector $\mathbf{w}_{(2)}$ corresponds to the eigenvector of \mathbf{C} with the second largest eigenvalue
 - The third vector $\mathbf{w}_{(3)}$ to the eigenvector with the third largest eigenvalue
 - ...
- The matrix \mathbf{W} is obtained by concatenating the resulting vectors, i.e.,

$$\mathbf{W} = [\mathbf{w}_{(1)} \quad \mathbf{w}_{(2)} \quad \cdots \quad \mathbf{w}_{(d)}] \in \mathbb{R}^{D \times d}$$

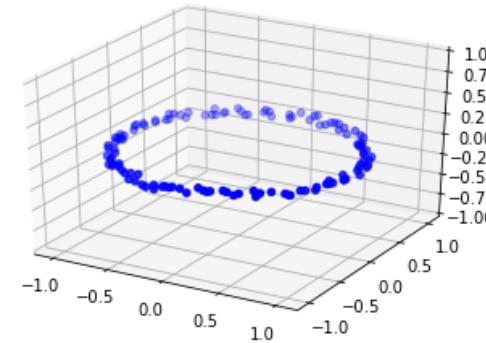
- This guarantees to satisfy the constraint $\mathbf{W}^T \mathbf{W} = \mathbf{I}_d$
 - Because the eigenvectors of a matrix are orthogonal and of norm 1

PCA

- In the limit, one can use all dimensions, i.e., set $d = D$
 - There is therefore no reduction of dimensionality
 - In 3D, you can think of this as a rotation of the data
 - This incurs no loss of information
 - The $d = D$ dimensions in the new space are uncorrelated



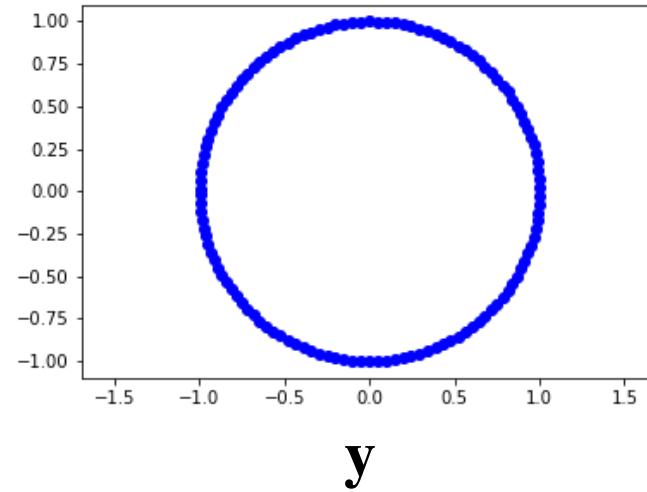
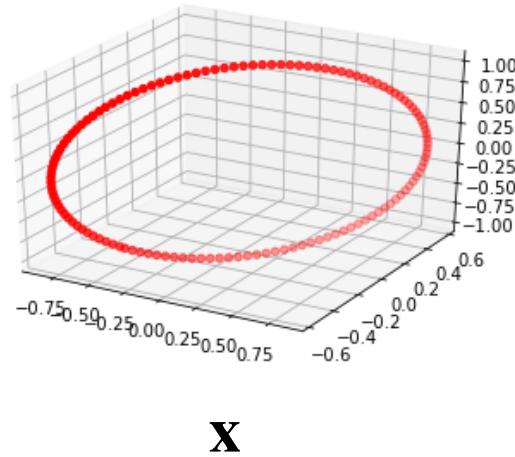
X



y

PCA

- Another option is to keep all the eigenvectors that correspond to non-zero eigenvalues
 - This means that the data is truly low-dimensional
 - E.g., this happens when we have fewer samples than dimensions ($N < D$)
 - The resulting $\{\mathbf{y}_i\}$ are lower dimensional ($d < D$)
 - But there is no loss of information



PCA

- In practice, one typically truncates the eigenvalues so as to discard some that are non-zero
 - This can be achieved by aiming to retain a pre-defined percentage of the data variance, measured as the sum of eigenvalues
 - E.g., to retain at least 90% of the variance, one can search for d such that

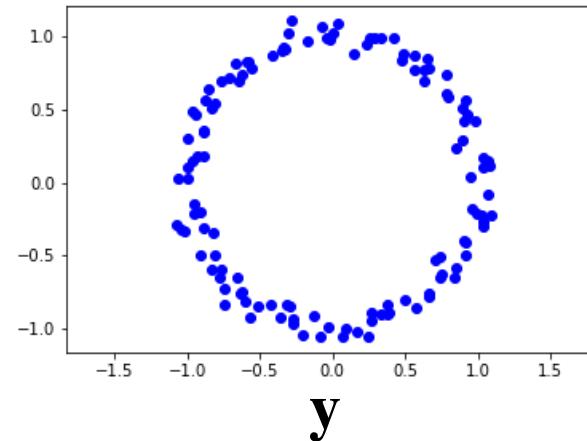
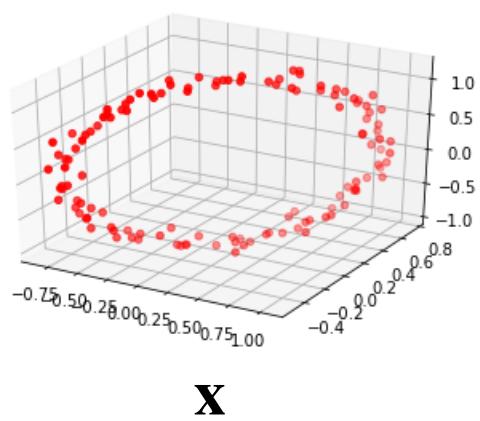
$$\sum_{j=1}^d \lambda_j \geq 0.9 \cdot \sum_{k=1}^D \lambda_k$$

assuming the eigenvalues are sorted in decreasing order

- The resulting $\{\mathbf{y}_i\}$ have an even lower dimension

PCA: Error minimization

- This incurs some loss of information



- However, the corresponding rectangular matrix \mathbf{W} is the orthogonal matrix that minimizes the reconstruction error

$$e_i = \|\hat{\mathbf{x}}_i - \mathbf{x}_i\|^2$$

where

$$\hat{\mathbf{x}}_i = \bar{\mathbf{x}} + \mathbf{W}\mathbf{y}_i = \bar{\mathbf{x}} + \mathbf{W}\mathbf{W}^T(\mathbf{x}_i - \bar{\mathbf{x}})$$

PCA: Demo

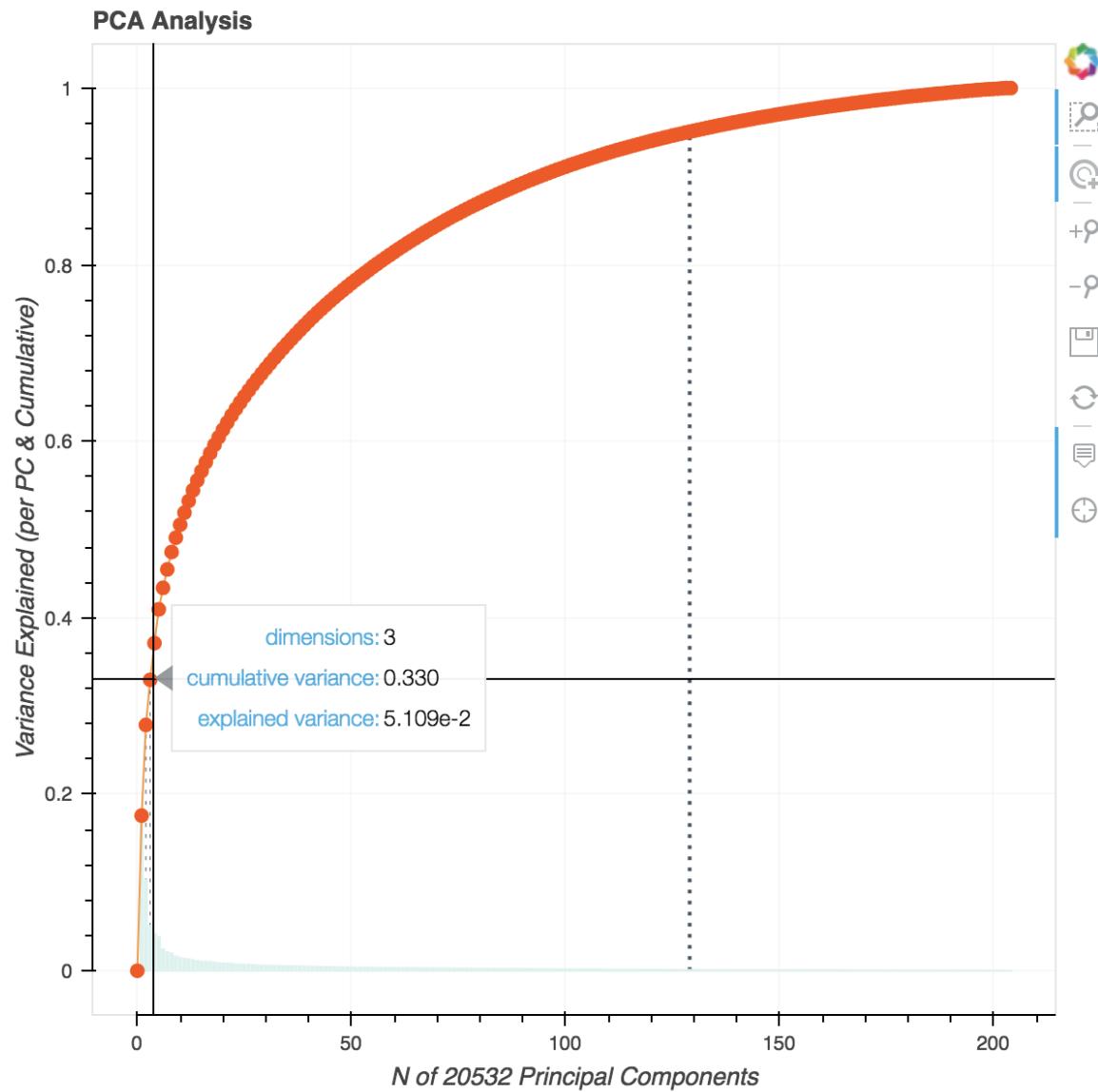
- <http://setosa.io/ev/principal-component-analysis/>

PCA: Example

- The Cancer Genome Atlas breast cancer RNA-Seq dataset:
 - Example from <https://medium.com/cascade-bio-blog/creating-visualizations-to-better-understand-your-data-and-models-part-1-a51e7e5af9c0>
 - Normal tissue vs primary tumor
 - 20532 features (genes for which an expression is measured)
 - 204 samples
- Question: What is the maximum dimensionality that we obtain after DR (if we remove all the truly irrelevant dimensions)?

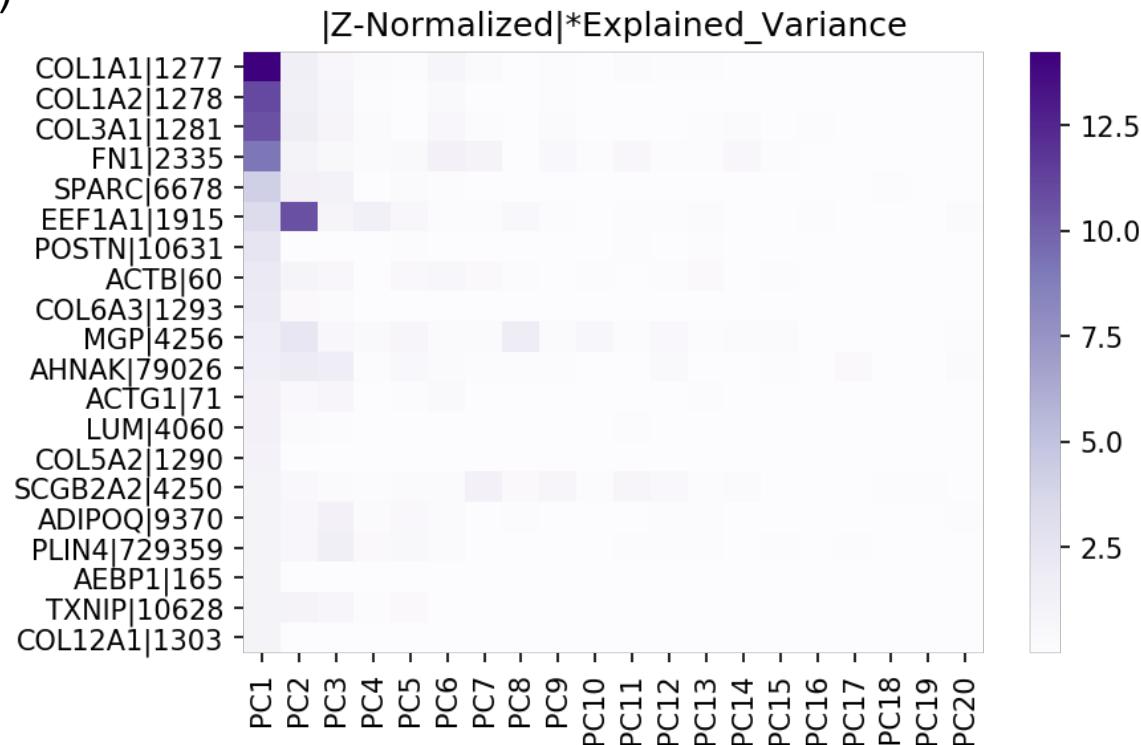
PCA: Example

- Breast cancer dataset:
 - Example from <https://medium.com/cascade-bio-blog/creating-visualizations-to-better-understand-your-data-and-models-part-1-a51e7e5af9c0>
 - Cumulative variance explained by the principal components



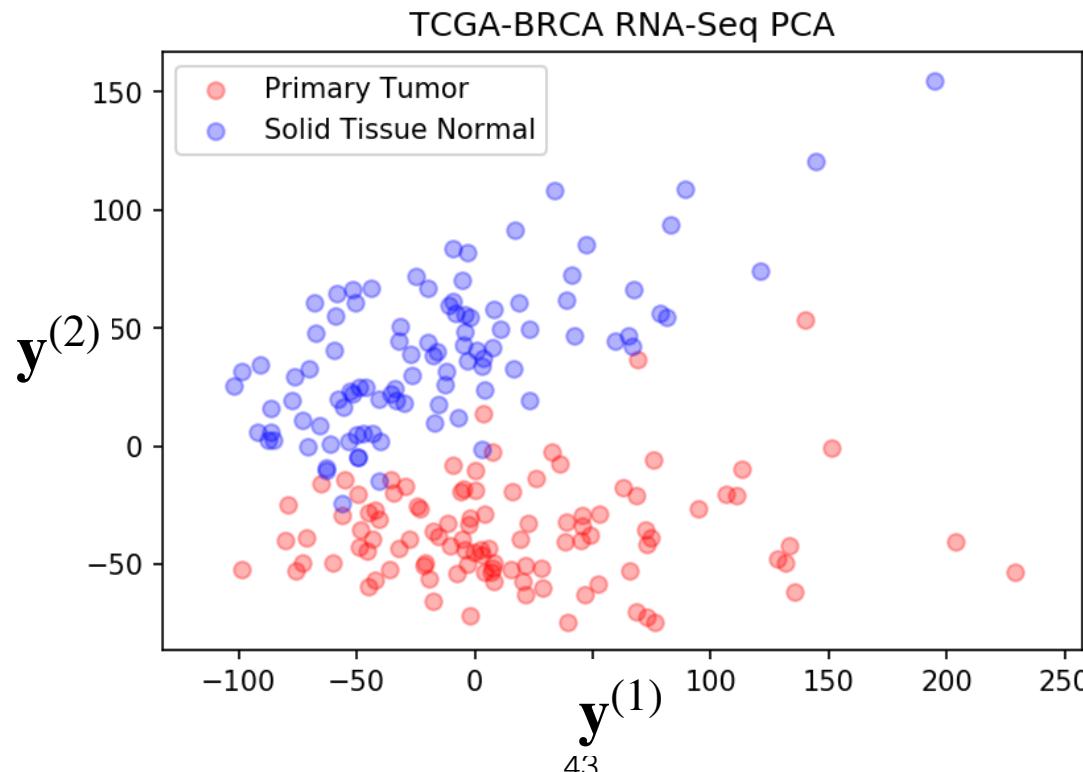
PCA: Example

- The Cancer Genome Atlas breast cancer RNA-Seq dataset:
 - Example from <https://medium.com/cascade-bio-blog/creating-visualizations-to-better-understand-your-data-and-models-part-1-a51e7e5af9c0>
 - Influence of each attribute on each component (re-normalized by the explained variance)



PCA: Example

- The Cancer Genome Atlas breast cancer RNA-Seq dataset:
 - Example from <https://medium.com/cascade-bio-blog/creating-visualizations-to-better-understand-your-data-and-models-part-1-a51e7e5af9c0>
 - Visualization of the samples projected in 2D



PCA: Mapping

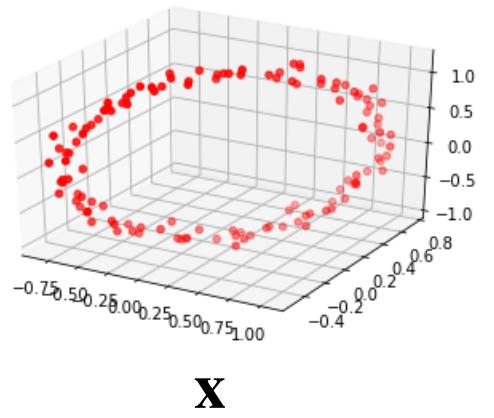
- PCA not only reduces the dimensionality of the original data. It provides a continuous mapping from the low-dimensional space to the high-dimensional one
- That is, for any $\mathbf{y} \in \mathbb{R}^d$, we can compute a point in the high-dimensional space as

$$\hat{\mathbf{x}} = \bar{\mathbf{x}} + \mathbf{W}\mathbf{y}$$

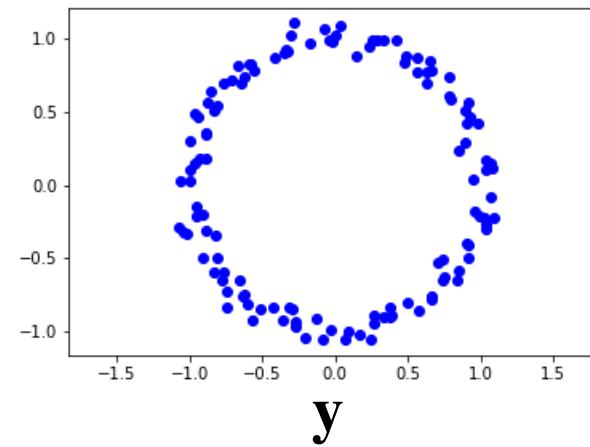
- This mapping constrains $\hat{\mathbf{x}}$ to lie in a subspace, and thus provides a form of regularization

PCA mapping: Example

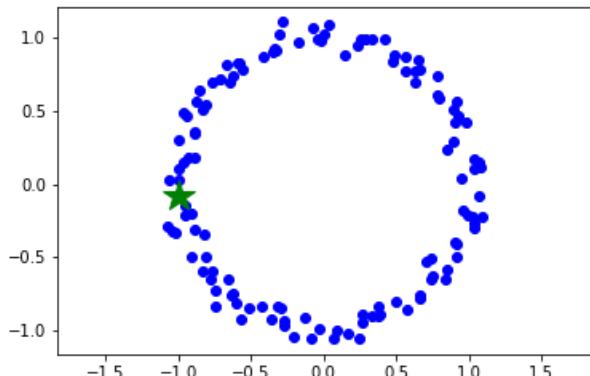
- Original data



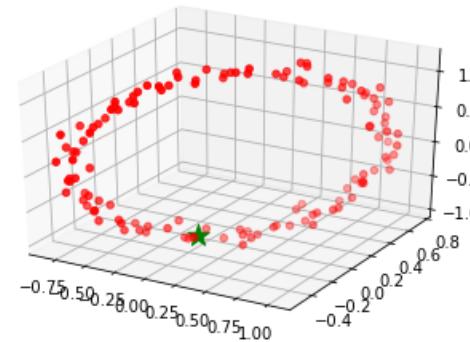
PCA
→



- New point (green star)

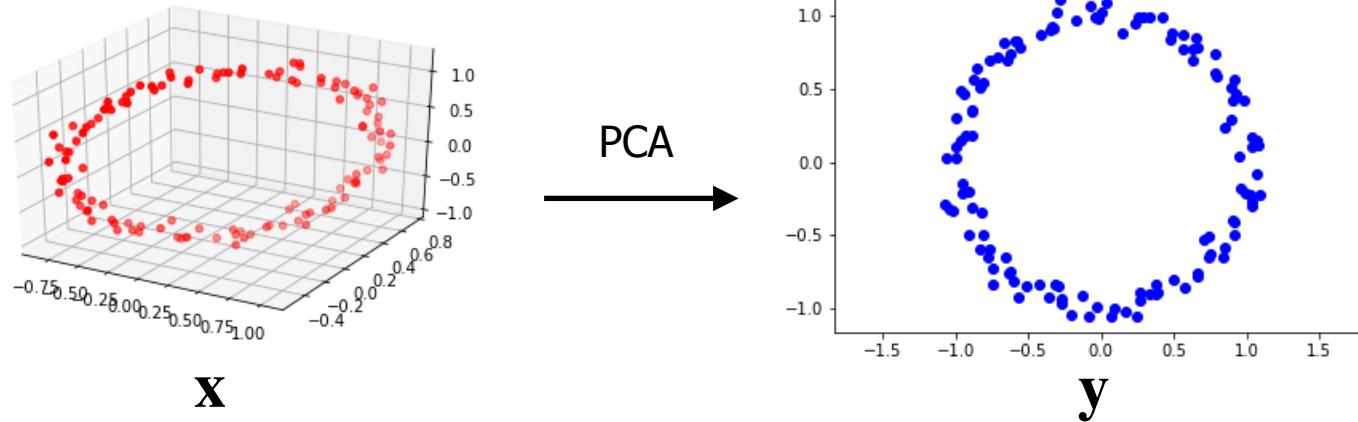


Mapping
→

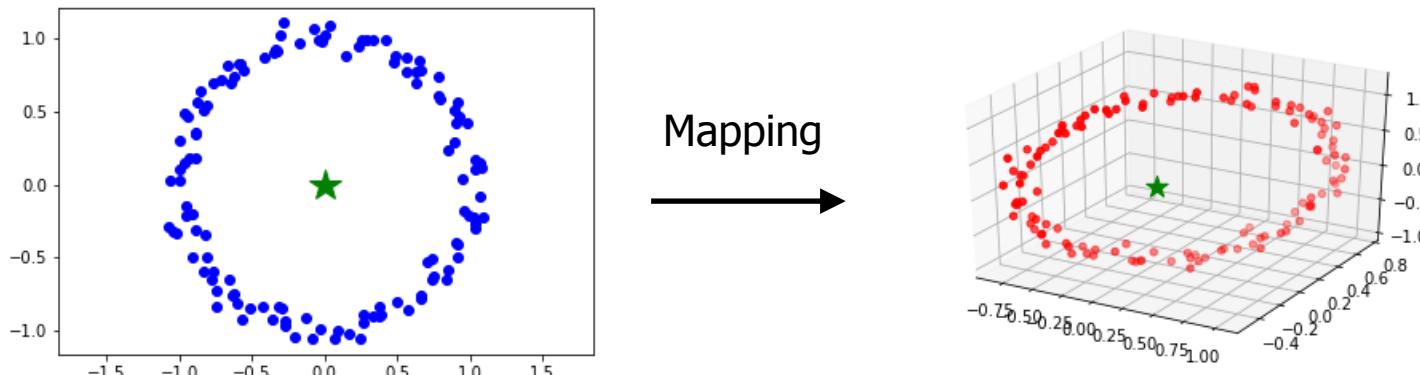


PCA mapping: Example

- Original data



- New point (green star)

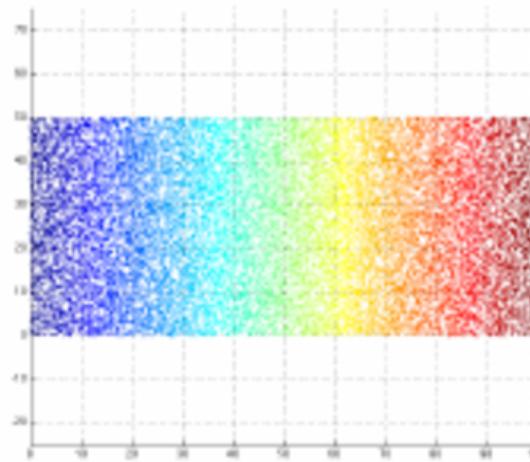
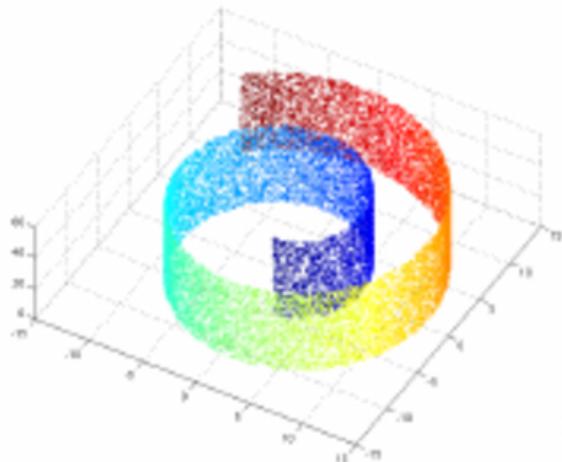


PCA mapping: Use case

- Morphable models: Blanz & Vetter, SIGGRAPH 1999
 - <http://gravis.dmi.unibas.ch/publications/Sigg99/siggraph99.mpg>

Dealing with nonlinear data

- PCA is a linear dimensionality reduction method
 - It just gives projections of the data
- How can we achieve nonlinear dimensionality reduction
 - E.g., a projection of the Swiss roll would squash it instead of unrolling it



Kernel PCA

- Schölkopf et al., 1999
- Same intuition as in supervised kernel methods:

- Map the data to a high-dimensional space

$$\mathbf{x}_i \rightarrow \phi(\mathbf{x}_i)$$

- Then perform PCA
- Our goal will again be to replace dot products with kernel values

$$k(\mathbf{x}_i, \mathbf{x}_j) = \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j)$$

Kernel PCA

- Let us first assume that the data is centered in feature space. That is,

$$\frac{1}{N} \sum_{i=1}^N \phi(\mathbf{x}_i) = \mathbf{0}$$

- Then, the covariance matrix can be written as

$$\mathbf{C} = \frac{1}{N} \sum_{i=1}^N \phi(\mathbf{x}_i) \phi(\mathbf{x}_i)^T$$

- Note that this is an $F \times F$ matrix, not an $N \times N$ one, so we still do not have kernel values here

Kernel PCA

- Applying PCA in feature space means that we need to solve the eigenvalue problem

$$\mathbf{C}\mathbf{w}_{(1)} = \lambda_1 \mathbf{w}_{(1)}$$

where $\mathbf{w}_{(1)}$ now is an F -dimensional vector

- From the previous definition of \mathbf{C} , this can be rewritten as

$$\left(\frac{1}{N} \sum_{i=1}^N \phi(\mathbf{x}_i) \phi(\mathbf{x}_i)^T \right) \mathbf{w}_{(1)} = \lambda_1 \mathbf{w}_{(1)}$$

Kernel PCA

- Recall that, according to the Representer theorem, $\mathbf{w}_{(1)}$ can be expressed as a linear combination of the $\{\phi(\mathbf{x}_i)\}$, i.e.,

$$\mathbf{w}_{(1)} = \sum_{j=1}^N a_j \phi(\mathbf{x}_j)$$

- Let us then substitute this alternative representation of $\mathbf{w}_{(1)}$ in our eigenvector equation. This yields

$$\left(\frac{1}{N} \sum_{i=1}^N \phi(\mathbf{x}_i) \phi(\mathbf{x}_i)^T \right) \sum_{j=1}^N a_j \phi(\mathbf{x}_j) = \lambda_1 \sum_{j=1}^N a_j \phi(\mathbf{x}_j)$$

Kernel PCA

- After some computations (skipped here), we can obtain a new eigenvalue problem

$$\mathbf{K}\mathbf{a} = \lambda_1 N\mathbf{a}$$

where $\mathbf{a} \in \mathbb{R}^N$ is the vector containing the new parameters of the problem, and \mathbf{K} is the kernel matrix

- Solving this problem gives us a vector \mathbf{a} , from which we could, in theory, obtain $\mathbf{w}_{(1)}$ as

$$\mathbf{w}_{(1)} = \sum_{i=1}^N a_i \phi(\mathbf{x}_i)$$

- Note, however, that it depends on $\{\phi(\mathbf{x}_i)\}$, which are unknown

Kernel PCA

- As in kernel ridge regression, this can be addressed by noting that, ultimately, we are not interested in $\mathbf{w}_{(1)}$, but rather in obtaining the low-dimensional projections $\{\mathbf{y}_i\}$
- For a 1D representation, these projections can be computed as

$$y_i = \phi(\mathbf{x}_i)^T \mathbf{w}_{(1)} = \sum_{j=1}^N a_j \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j) = \sum_{j=1}^N a_j k(\mathbf{x}_i, \mathbf{x}_j)$$

which depends on the kernel function $k(\cdot, \cdot)$

- As in the linear case, multiple output dimensions can be obtained by taking the d eigenvectors with largest eigenvalues

Kernel PCA: Non-centered data

- So far, we have assumed that the data was centered in feature space
- There is, however, no reason for this to be satisfied
 - Even if the original data is centered, the data after mapping to feature space might not be
- To handle the more realistic scenario where it isn't, let us define

$$\tilde{\phi}(\mathbf{x}_i) = \phi(\mathbf{x}_i) - \frac{1}{N} \sum_{j=1}^N \phi(\mathbf{x}_j)$$

Kernel PCA: Non-centered data

- The $(i, j)^{\text{th}}$ elements of the corresponding kernel matrix is then given by

$$\tilde{\mathbf{K}}_{i,j} = \left(\tilde{\phi}(\mathbf{x}_i) \right)^T \tilde{\phi}(\mathbf{x}_j)$$

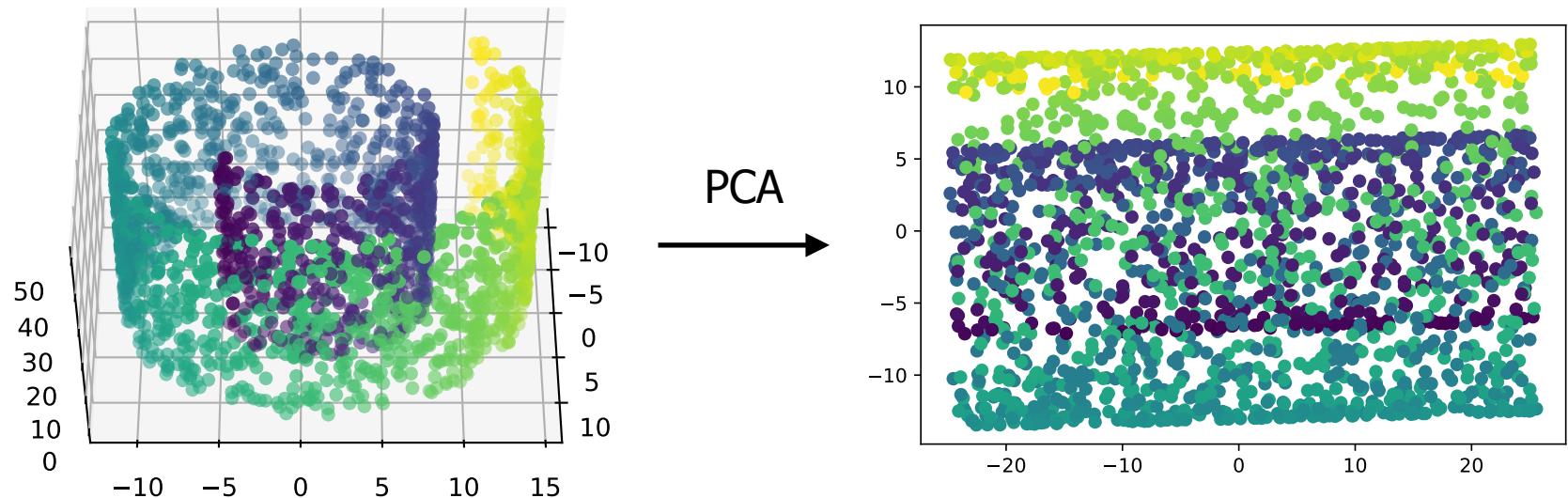
- This kernel matrix can in fact be computed based on the original one as

$$\tilde{\mathbf{K}} = \mathbf{K} - \mathbf{1}_N \mathbf{K} - \mathbf{K} \mathbf{1}_N + \mathbf{1}_N \mathbf{K} \mathbf{1}_N$$

where $\mathbf{1}_N$ is an $N \times N$ matrix with every element equal to $1/N$

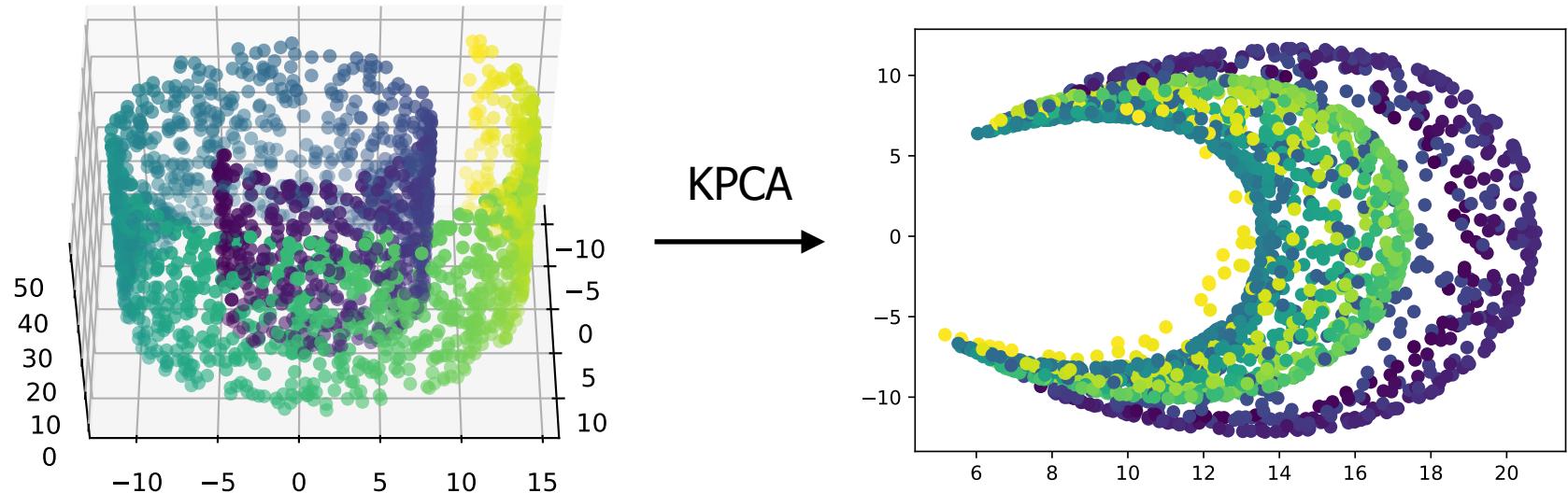
Kernel PCA: Example

- Swiss roll: PCA results



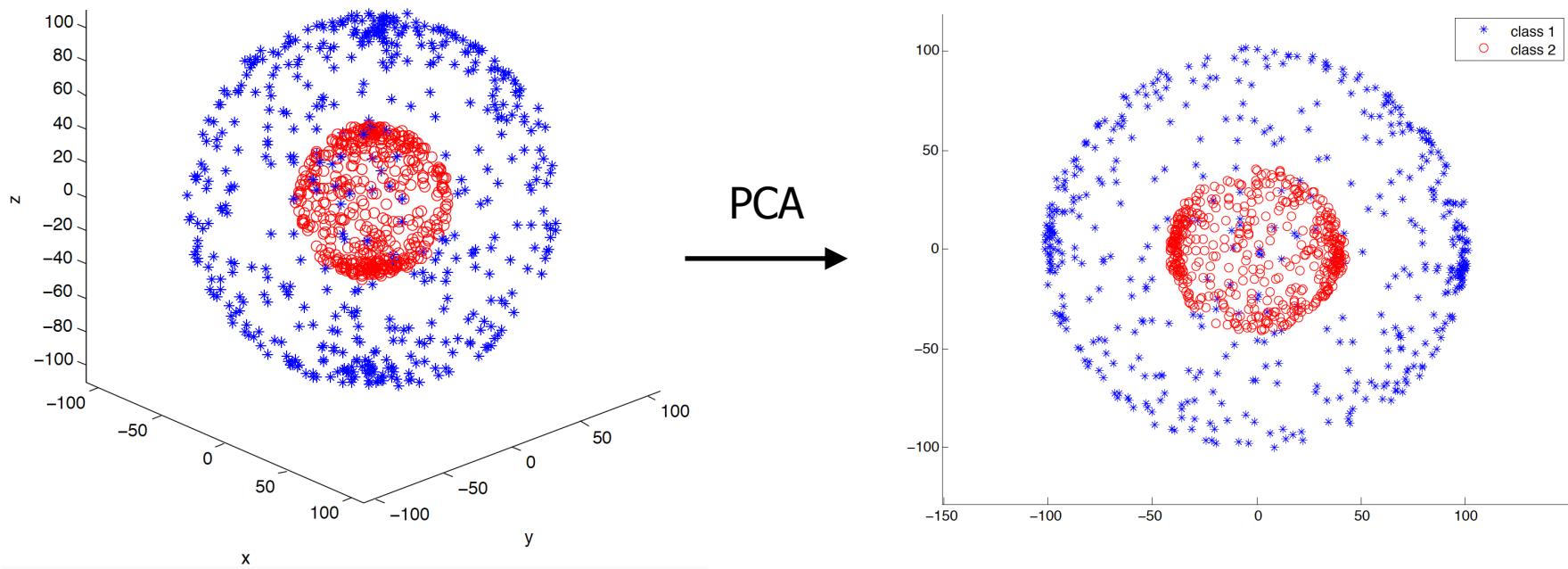
Kernel PCA: Example

- Swiss roll: Kernel PCA results (RBF kernel)



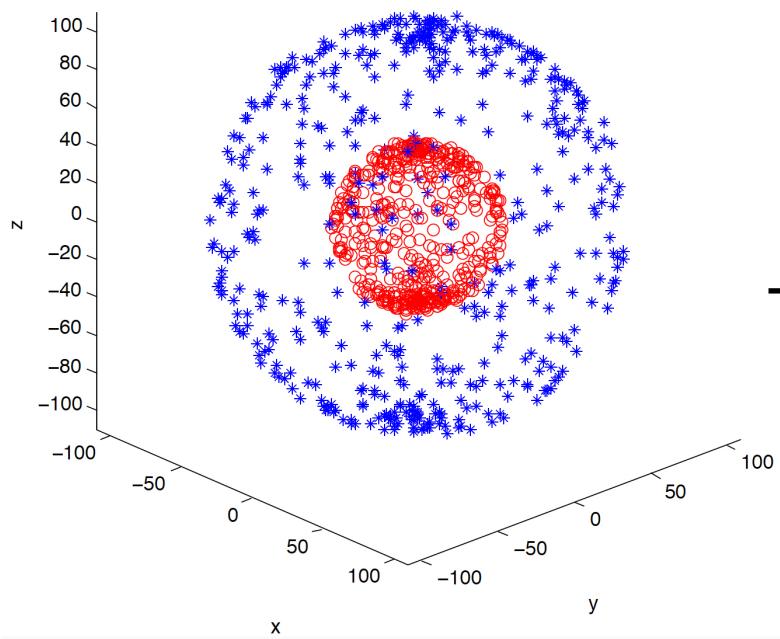
Kernel PCA: Example

- 3D inputs, 2 classes: PCA results

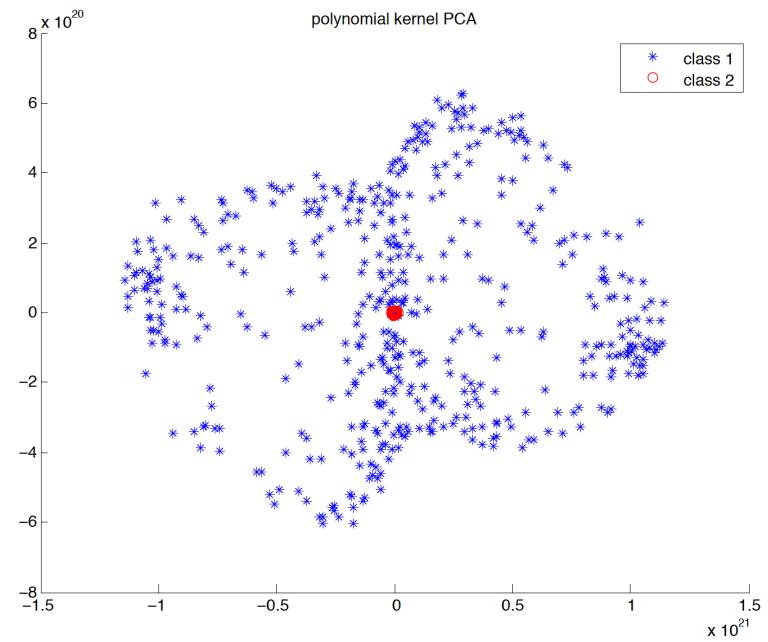


Kernel PCA: Example

- 3D inputs, 2 classes: Kernel PCA results (polynomial kernel)

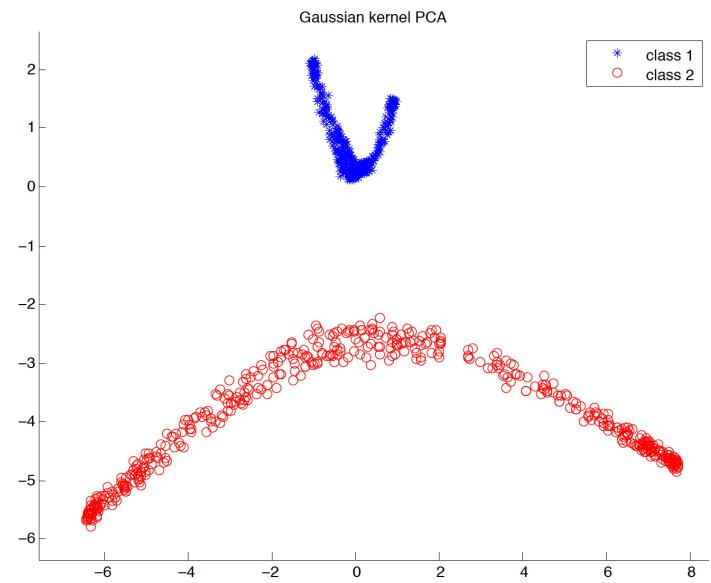
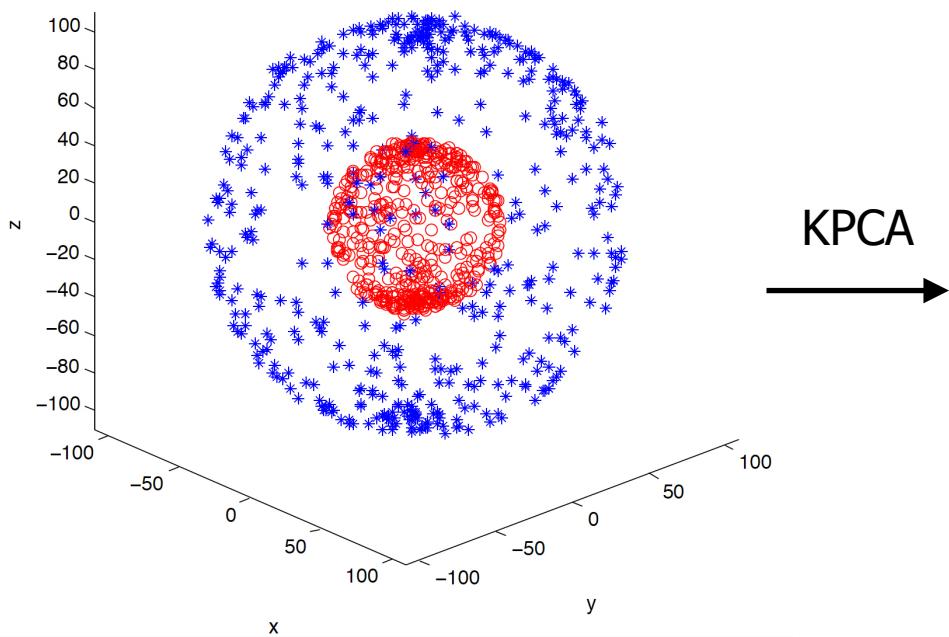


KPCA
→



Kernel PCA: Example

- 3D inputs, 2 classes: Kernel PCA results (RBF kernel)



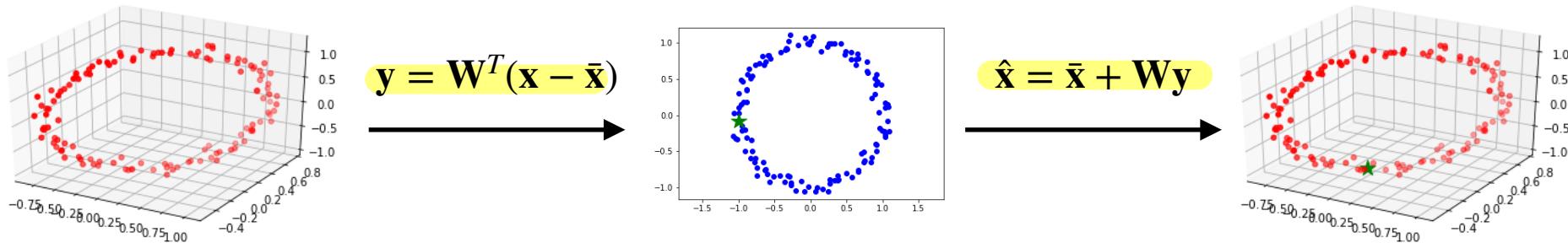
- Discussion: Why is an RBF kernel well suited to this data?

Kernel PCA vs PCA

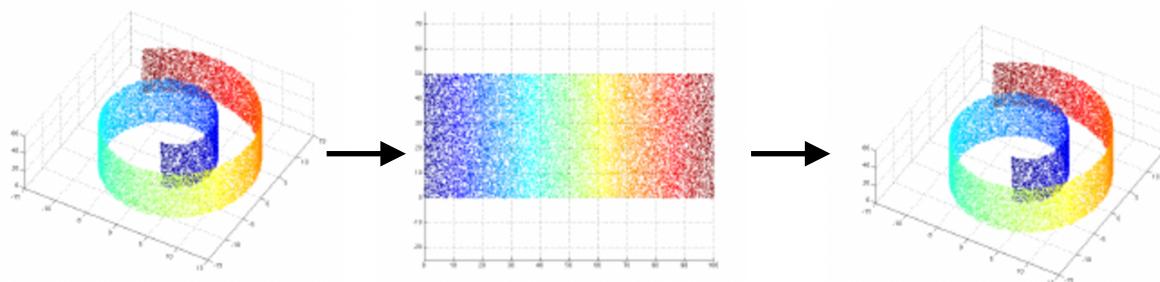
- As PCA, Kernel PCA defines a mapping from the high-dimensional data to the low-dimensional one
- However, in contrast to PCA, Kernel PCA does not provide a mapping from the low-dimensional space to the high-dimensional one
 - We cannot simply compute the high-dimensional version of any point in the low-dimensional space
- Can we nonetheless have this inverse mapping with a nonlinear method?
 - Solution: Let's rely on deep networks

Another look at PCA

- PCA provides a mapping from high to low dimension and back

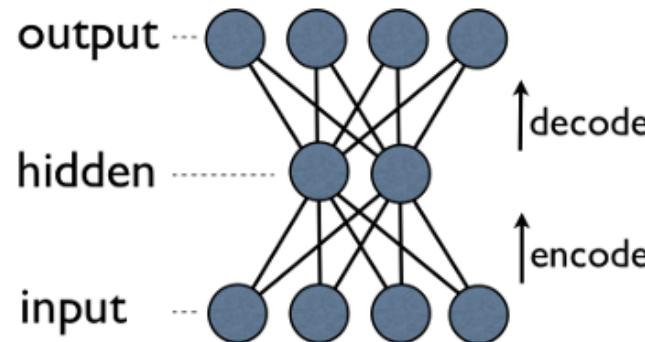


- Both mappings are linear, so PCA cannot handle cases like this



Autoencoder

- There is no reason to limit ourselves to linear mappings
- Following neural networks, the simplest extension consists of using a nonlinear activation function in the middle
- This is referred to as an *autoencoder*

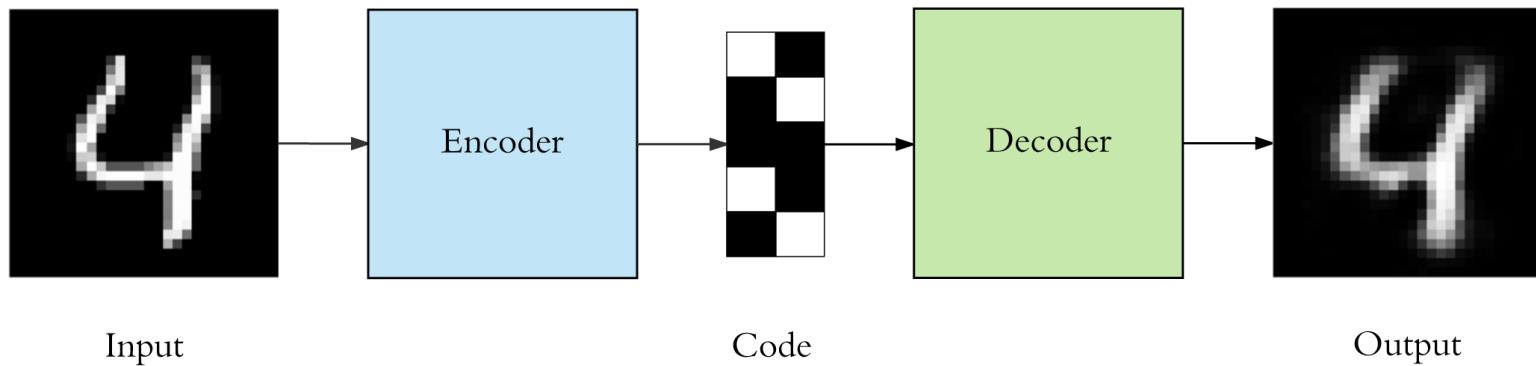


$$\hat{\mathbf{x}} = \mathbf{W}f(\mathbf{W}^T \mathbf{x})$$

with $f(\cdot)$ a nonlinear, element wise activation function

Autoencoder: A general form

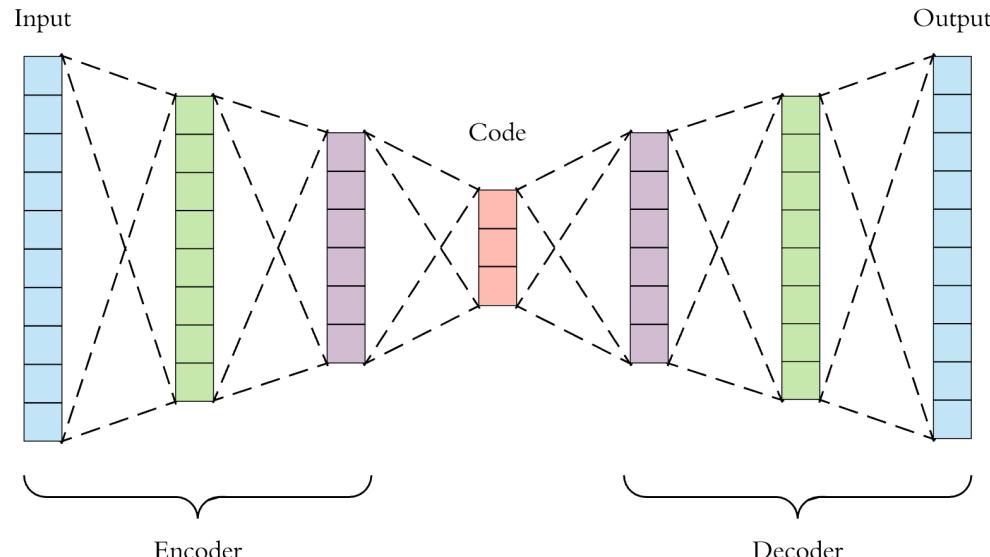
- In general, an autoencoder can be represented by two mappings:
 - The encoder going from the input to a latent representation
 - The decoder going from the latent representation to the output



- As in the PCA case, the output of the decoder is meant to be similar to the original input

Deep autoencoder

- There is no reason to restrict the encoder and decoder to consist of a single layer
- We can then create **deep autoencoders** by stacking **multiple layers**, each with an activation function
 - . Each layer gives a mapping of the form $\mathbf{z}_{(j)} = f_{(j)} \left(\mathbf{W}_{(j)}^T \mathbf{z}_{(j-1)} \right)$



Autoencoder training

- As mentioned before (Slide 38), with PCA, the reconstructed inputs $\{\hat{\mathbf{x}}_i\}$ minimize the least-square error

$$e_i = \|\hat{\mathbf{x}}_i - \mathbf{x}_i\|^2$$

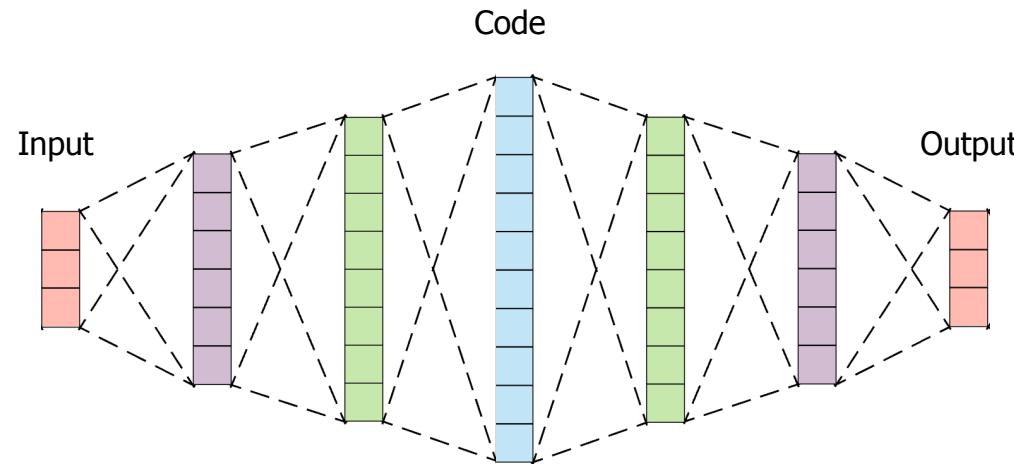
- Following this intuition, an autoencoder can be trained by minimizing the empirical risk

$$R(\{\mathbf{W}_{(j)}\}) = \frac{1}{N} \sum_{i=1}^N \|\hat{\mathbf{x}}_i(\{\mathbf{W}_{(j)}\}) - \mathbf{x}_i\|^2$$

- As for MLPs, this is non-convex and minimization is done via stochastic gradient descent

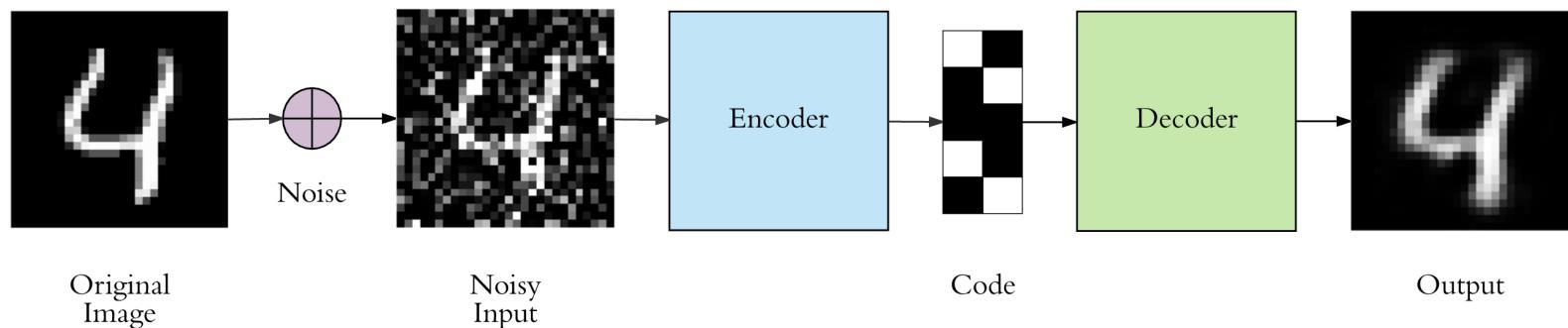
Dimensionality expansion

- Because the mapping is nonlinear, there is no reason to restrict the latent representation to be low-dimensional
 - One can then perform dimensionality expansion



Denoising autoencoders

- Having a low-dimensional latent representation encourages the autoencoder to learn an “intelligent” mapping
- With dimensionality expansion, one could simply learn to copy the input
- To avoid this, one can add noise to the input and aim to reconstruct a noise-free version of the input



Sparse autoencoder

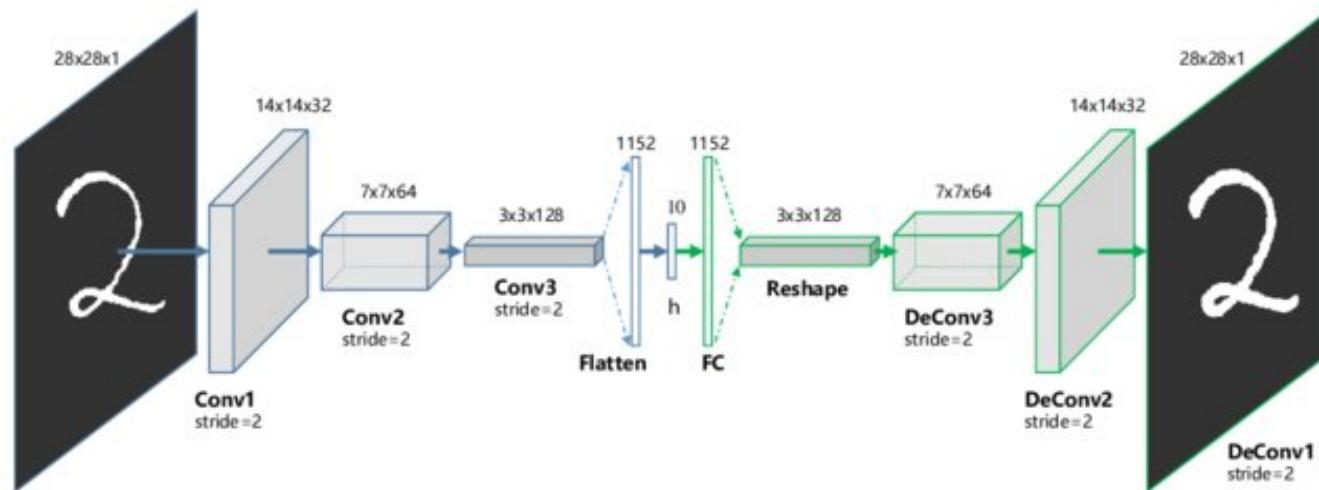
- Another approach to learn a meaningful latent representation consists of adding a regularizer
- Let \mathbf{y}_i denote the latent representation (or code) for sample i
- Then, one can encourage a large number of the elements $\{\mathbf{y}_i^{(k)}\}$ to go to zero
- This can be achieved by a sparsity-inducing regularizer, such as the L_1 norm

$$E_r(\{\mathbf{W}_{(j)}\}) = \sum_{i=1}^N \sum_{k=1}^M |\mathbf{y}_i^{(k)}|$$

with M the dimensionality of the latent representation

Convolutional autoencoder

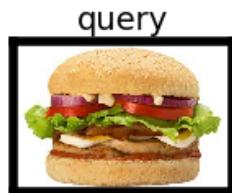
- With convolutions and transposed convolutions, one can also design convolutional autoencoders
 - Note that fully-connected layers can also be used in the middle
 - Example from Guo et al., ICONIP 2017



Autoencoder: Application

- A code provides a compact representation of the input
- It can be used for retrieval in a large data collection, e.g., via by k nearest neighbors
 - E.g., <https://towardsdatascience.com/find-similar-images-using-autoencoders-315f374029ea>

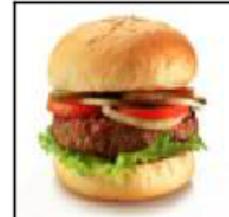
Image Retrieval ($k=5$)



Rank #1



Rank #2



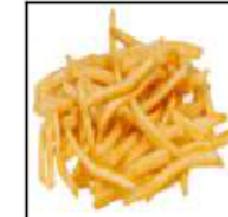
Rank #3



Rank #4

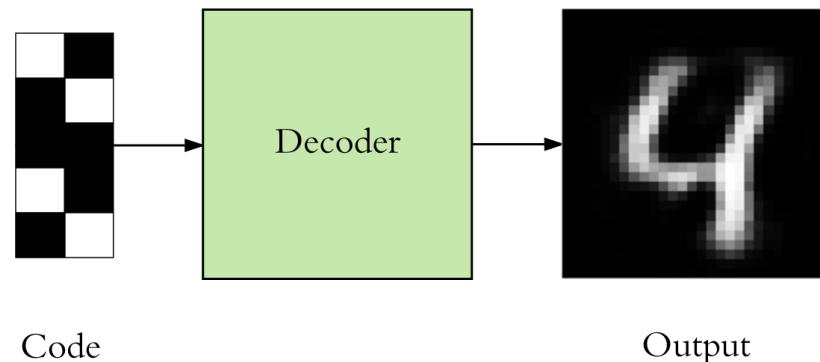


Rank #5



Autoencoder: Mapping

- As in PCA, the decoder provides a mapping from the latent space to the output (same as input) space
- One can thus generate new samples (e.g., images) by taking arbitrary points in the latent space



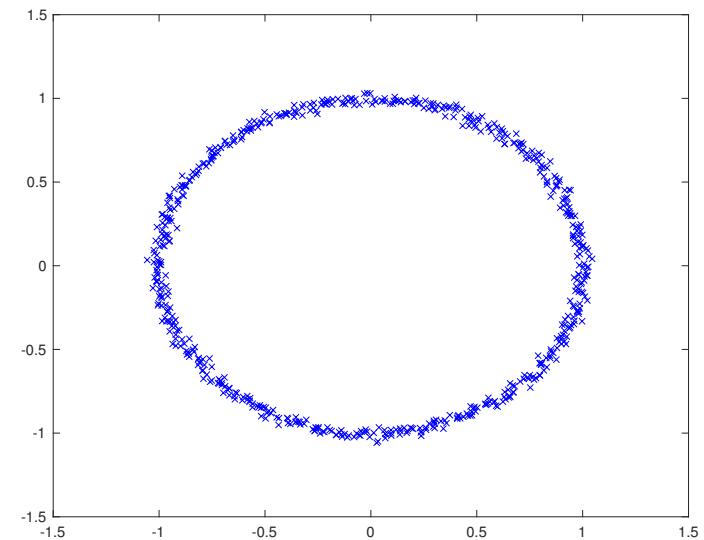
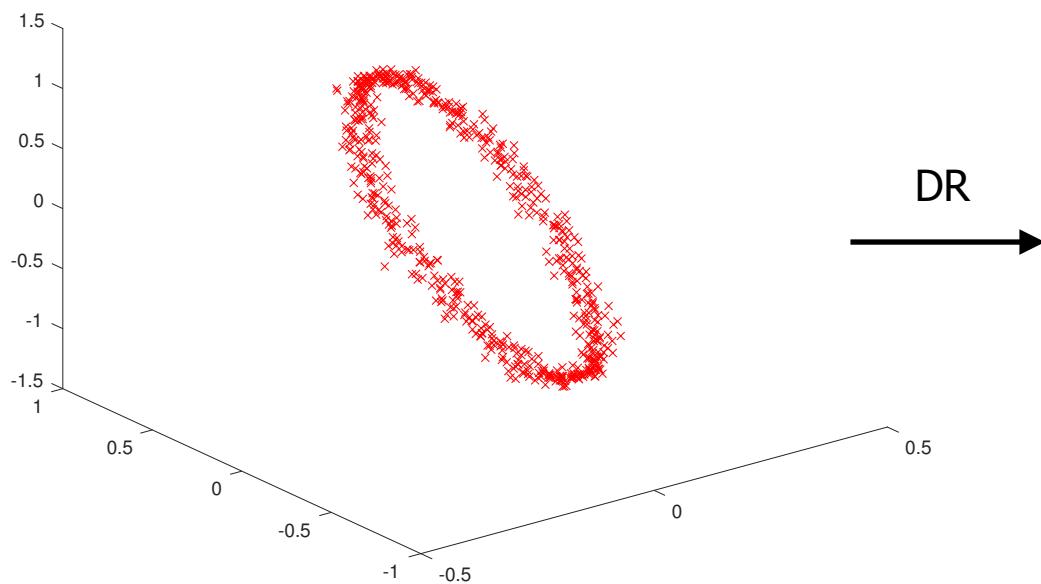
Autoencoder demo

- http://dpkingma.com/sgvb_mnist_demo/demo.html

Lecture 13: Clustering

Recap: Dimensionality reduction

- Dimensionality reduction is the task of
 - discovering the data manifold
 - getting a low-dimensional representation of the data
 - sometimes at some loss of information (hopefully noise)



Recap: Principal Component Analysis (PCA)

- Pearson (1901), Hotelling (1933)
- Given N samples $\{\mathbf{x}_i\}$, PCA yields a projection of the form

$$\mathbf{y}_i = \mathbf{W}^T(\mathbf{x}_i - \bar{\mathbf{x}}) \quad \text{s.t. } \mathbf{W}^T \mathbf{W} = \mathbf{I}_d$$

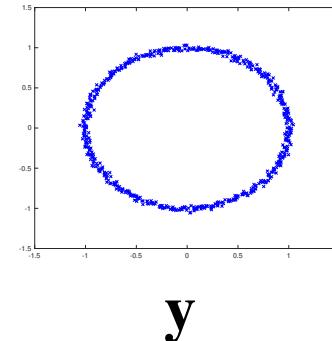
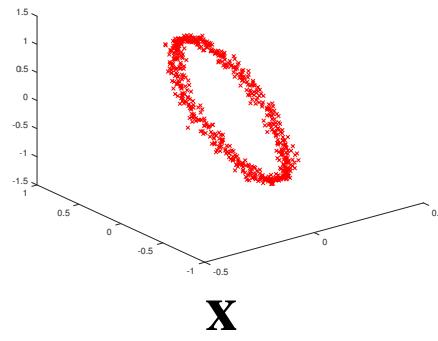
where $\bar{\mathbf{x}}$ is the mean of the data, i.e.,

$$\bar{\mathbf{x}} = \frac{1}{N} \sum_{i=1}^N \mathbf{x}_i$$

- The data is unsupervised:
 - What do we want this projection to achieve?

Recap: PCA objective

- We aim to keep most of the “important” signal
- Yet, we want to remove the noise



- This can be achieved by finding directions that have a large variance, i.e., for the j^{th} output dimension, we want to maximize

$$\text{var}(\{y_i^{(j)}\}) = \frac{1}{N} \sum_{i=1}^N (y_i^{(j)} - \bar{y}^{(j)})^2$$

where $\bar{y}^{(j)}$ is the mean of the j^{th} dimension of the data after projection

Recap: PCA: Variance maximization

- For a 1D projection, this can be achieved by taking the eigenvector $\mathbf{w}_{(1)}$ of the data covariance matrix \mathbf{C} with the largest eigenvalue
- To obtain an output representation that is more than 1D, i.e., $d > 1$, we can perform recursively
 - The second projection vector $\mathbf{w}_{(2)}$ corresponds to the eigenvector of \mathbf{C} with the second largest eigenvalue
 - The third vector $\mathbf{w}_{(3)}$ to the eigenvector with the third largest eigenvalue
 - ...
- The matrix \mathbf{W} is obtained by concatenating the resulting vectors, i.e.,

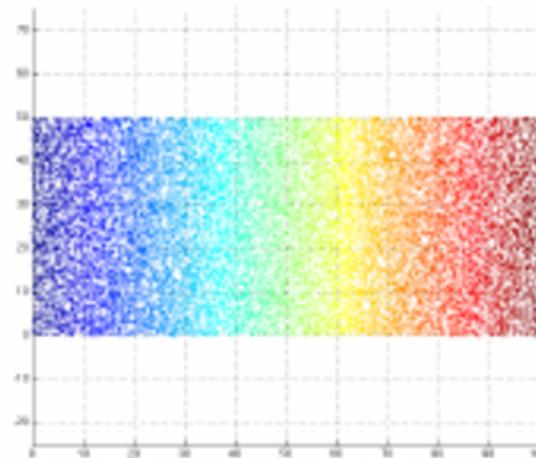
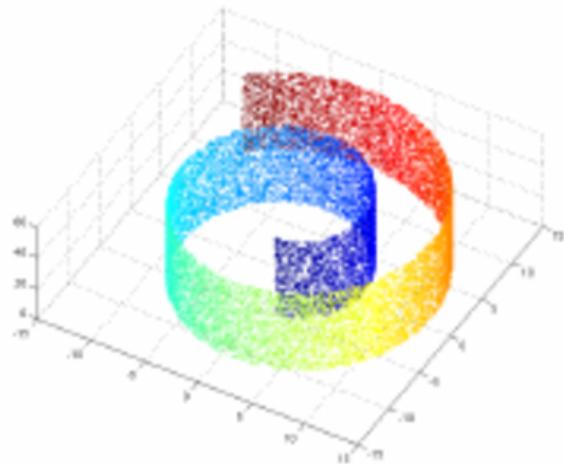
$$\mathbf{W} = [\mathbf{w}_{(1)} \quad \mathbf{w}_{(2)} \quad \cdots \quad \mathbf{w}_{(d)}] \in \mathbb{R}^{D \times d}$$

Recap: PCA Example

- The Cancer Genome Atlas breast cancer RNA-Seq dataset:
 - Example from <https://medium.com/cascade-bio-blog/creating-visualizations-to-better-understand-your-data-and-models-part-1-a51e7e5af9c0>
 - Normal tissue vs primary tumor
 - 20532 features (genes for which an expression is measured)
 - 204 samples
- Question: What is the maximum dimensionality that we obtain after DR (if we remove all the truly irrelevant dimensions)?
 - Answer: 20532 features means $D = 20532$, so d is at most 20532. However, because we have $N = 204$ samples, d is at most 204

Recap: Dealing with nonlinear data

- PCA and Fisher LDA are linear dimensionality reduction methods
 - They just give projections of the data
- How can we achieve nonlinear dimensionality reduction
 - E.g., a projection of the Swiss roll would squash it instead of unrolling it



Recap: Kernel PCA

- Schölkopf et al., 1999
- Same intuition as in for supervised kernel methods:
 - Map the data to a high-dimensional space

$$\mathbf{x}_i \rightarrow \phi(\mathbf{x}_i)$$

- Then perform PCA
- Our goal will again be to replace dot products with kernel values

$$k(\mathbf{x}_i, \mathbf{x}_j) = \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j)$$

Recap: Kernel PCA

- Let us assume that the data is centered in feature space (we saw that this could be relaxed). That is,

$$\frac{1}{N} \sum_{i=1}^N \phi(\mathbf{x}_i) = \mathbf{0}$$

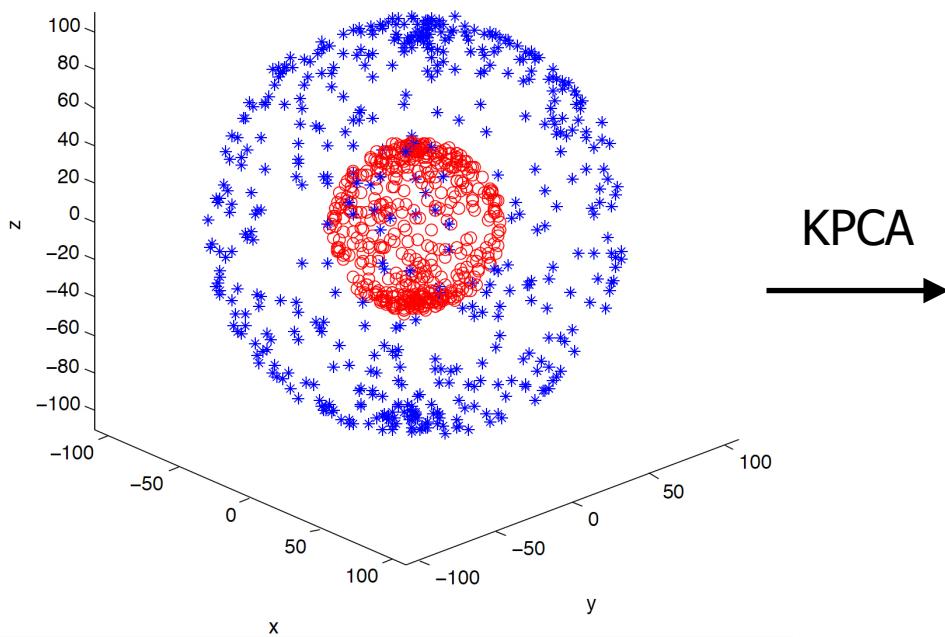
- Then, one can obtain coefficients \mathbf{a} by eigendecomposition of the kernel matrix \mathbf{K}
- These coefficients can then be used to obtain the 1D low-dimensional representation as

$$y_i = \sum_{j=1}^N a_j k(\mathbf{x}_i, \mathbf{x}_j)$$

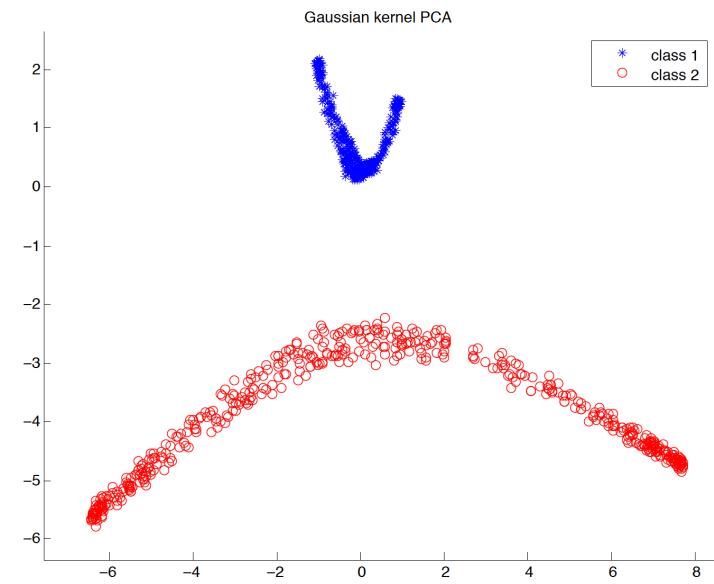
- For more than 1D, one can use more eigenvectors of \mathbf{K}

Recap: Kernel PCA: Example

- 3D inputs, 2 classes: Kernel PCA results (RBF kernel)



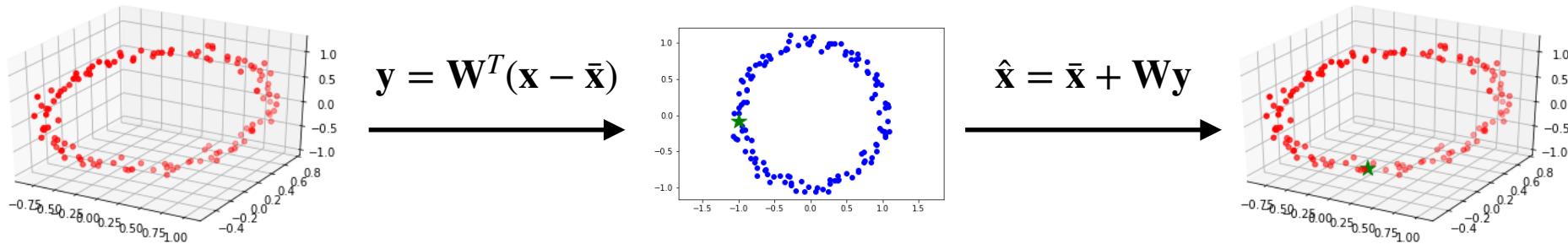
KPCA
→



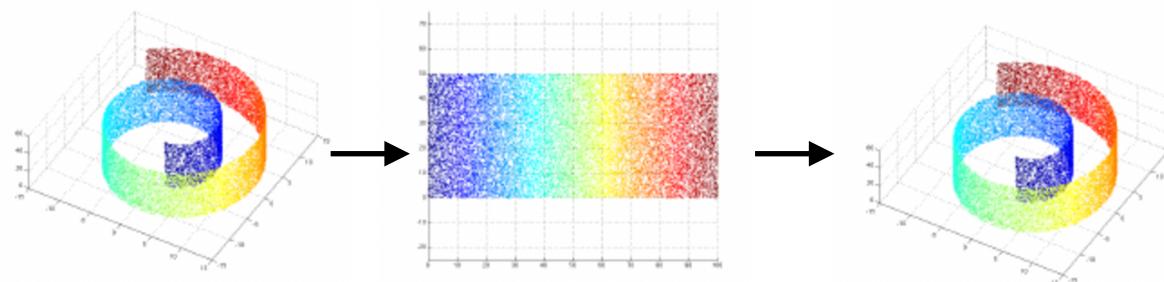
- Discussion: Why is an RBF kernel well suited to this data?
 - The similarity between points depends on their distance

Recap: Another look at PCA

- PCA provides a mapping from high to low dimension and back

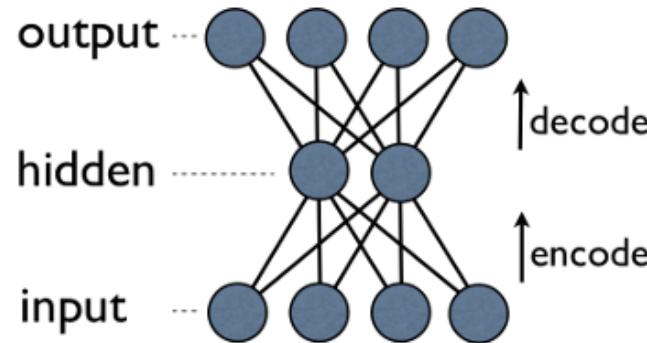


- Both mappings are linear, so PCA cannot handle cases like this



Recap: Autoencoder

- There is no reason to limit ourselves to linear mappings
- Following neural networks, the simplest extension consists of using a nonlinear activation function in the middle
- This is referred to as an *autoencoder*

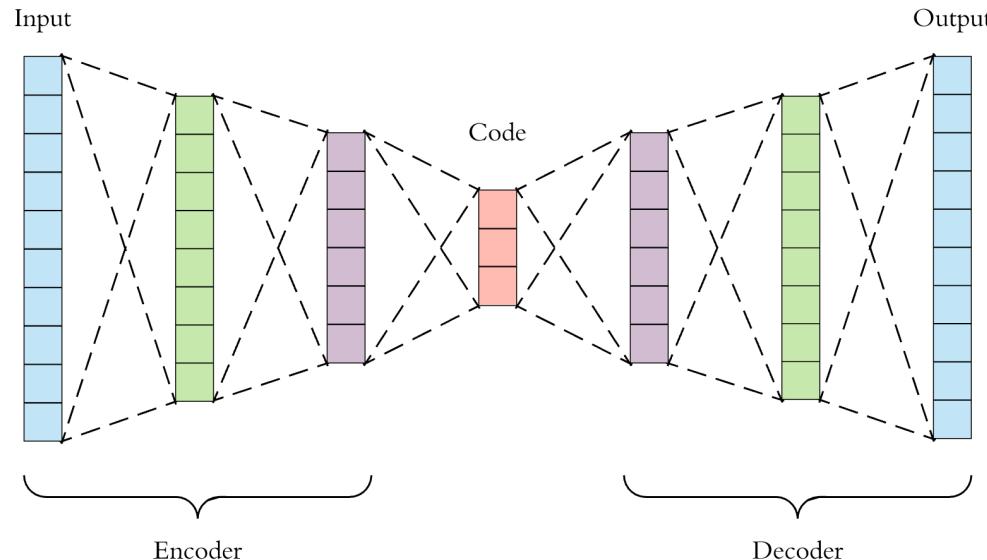


$$\hat{\mathbf{x}} = \mathbf{W}f(\mathbf{W}^T \mathbf{x})$$

with $f(\cdot)$ a nonlinear, element wise activation function

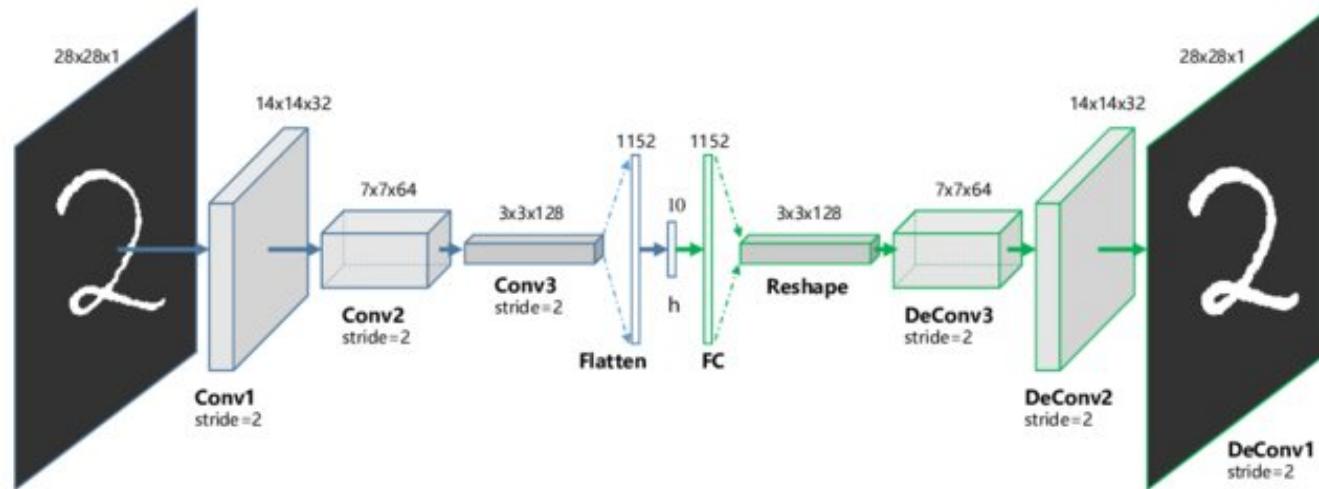
Recap: Deep autoencoder

- There is no reason to restrict the encoder and decoder to consist of a single layer
- We can then create deep autoencoders by stacking multiple layers, each with an activation function
 - . Each layer gives a mapping of the form $\mathbf{z}_{(j)} = f_{(j)} \left(\mathbf{W}_{(j)}^T \mathbf{z}_{(j-1)} \right)$



Recap: Convolutional autoencoder

- With convolutions and transposed convolutions, one can also design convolutional autoencoders
 - Note that fully-connected layers can also be used in the middle
 - Example from Guo et al., ICONIP 2017

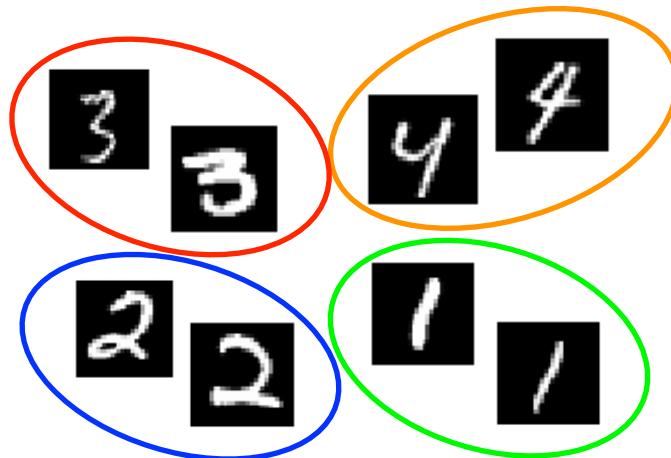


Goals of today's lecture

- Introduce another unsupervised learning task: clustering
- Introduce K-means clustering
- Introduce Fisher Linear Discriminant Analysis, which combines notions of clustering and of dimensionality reduction
- Introduce spectral clustering

Clustering: Another unsupervised learning task

- Given some input data



- Can we identify the groups, without performing any data transformation?
- This operation is called **clustering**

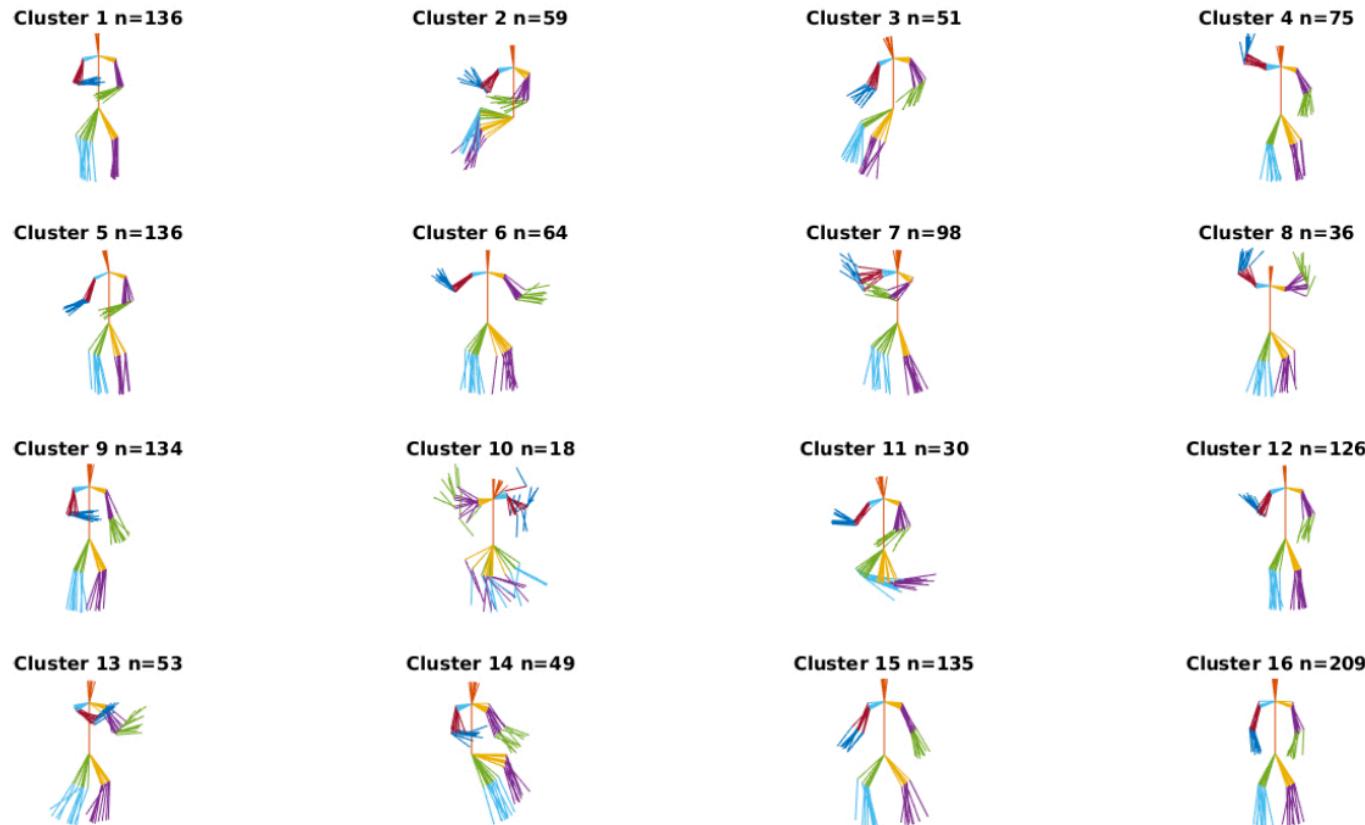
Motivating clustering example

- In Impett & Süsstrunk, Digital Humanities, 2017, it was observed that paintings by different artists representing the same scene depict the characters in the same pose



Motivating clustering example

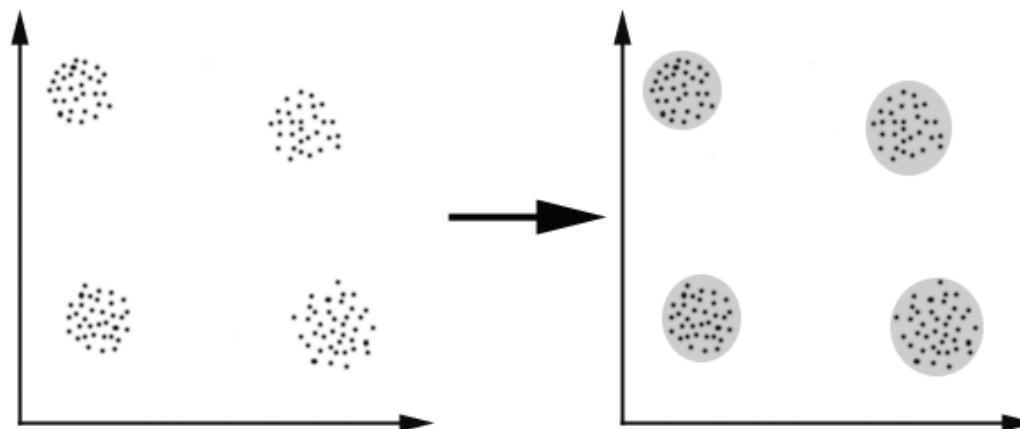
- One can then think of analyzing this phenomenon by clustering the poses observed in a collection of paintings



A simple clustering algorithm

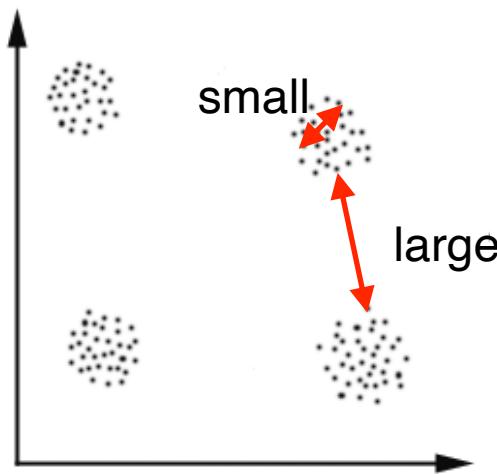
- **K-means clustering:**

- Given a set of input samples, group the samples into K clusters
- K is a hyper-parameter, assumed to be known/given
- In the toy example below, each data sample is a point in 2D. In our previous example, each data sample was a human pose



K-means clustering: Intuition

- The distances between the points within a cluster should be small
- The distances across clusters should be large



- This can be encoded via distances to cluster centers $\{\mu_1, \dots, \mu_K\}$

K-means clustering: Algorithm

1. Initialize $\{\mu_1, \dots, \mu_K\}$ (e.g., randomly)
2. While not converged
 - 2.1. Assign each point \mathbf{x}_i to the nearest center μ_k
 - 2.2. Update each center μ_k based on the points assigned to it

Algorithm: More details

- Step 2.1: Assign each point \mathbf{x}_i to the nearest center μ_k
 - For each point \mathbf{x}_i , compute the Euclidean distance to every center $\{\mu_1, \dots, \mu_K\}$
 - Find the smallest distance
 - The point is said to be assigned to the corresponding cluster (note that **each point is assigned to a single cluster**)
- Step 2.2: Update each center μ_k based on the points assigned to it
 - Recompute each center μ_k as the mean of the points that were assigned to it

Algorithm: More details

- Step 2: While not converged
 - The algorithm iteratively updates the cluster assignment for each point and the cluster centers. We need to eventually stop iterating
 - This could be achieved by a **fixed number of iterations**. However, setting this number is arbitrary and a too small number can lead to bad results
 - The algorithm is guaranteed to converge to a stable solution, where the cluster centers and the point assignments are fixed, i.e., do not change as more iterations are performed
 - The difference in assignments or center locations between two iterations can be used as criteria to stop the algorithm
- Importantly: **While the algorithm always converges, it does not always converge to the best (desired) solution**
 - The solution depends on the center initialization

K-means clustering: Demo

- <https://www.naftaliharris.com/blog/visualizing-k-means-clustering/>

K-means clustering: Example

- Color-based image segmentation (Bishop's book, Chap 9.1):
 - Consider a single image
 - Each pixel corresponds to one data point
 - Each pixel is encoded by its RGB color, i.e., $\mathbf{x}_i \in \mathbb{R}^3$
 - We seek to cluster the pixels according to their color



K-means clustering: Example

- Color-based image segmentation (Bishop's book, Chap 9.1):

Original image



$K = 2$



$K = 3$



$K = 10$



K-means clustering: Properties

- ✓ Simple algorithm guaranteed to converge
 - ✗ Does not always converge to the best solution
- Run the algorithm several times with different initialization
- For each solution, sum the distances of each point to its assigned cluster
 - Choose the solution with the smallest sum

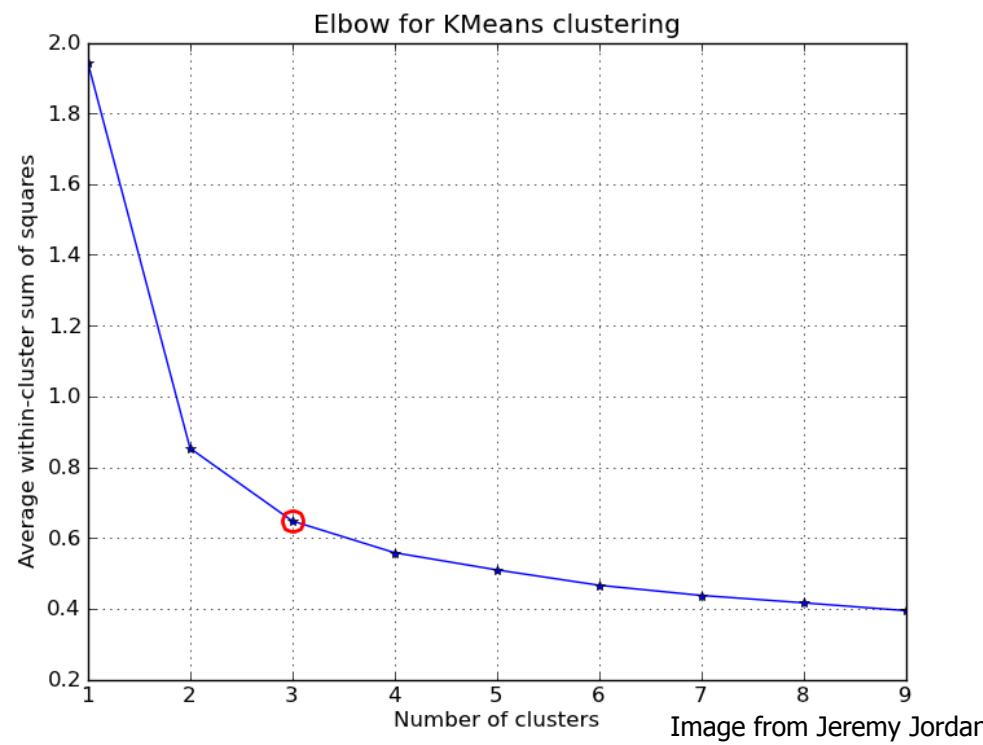
K-means clustering: Properties

- ✓ Returns exactly K clusters
- ✗ Some clusters might be empty (no samples assigned)
- ✗ Requires the user to determine K
 - This is unsupervised learning, so cross-validation is not an option

→ Elbow method:

- Run the algorithm with different values of K
- For each solution, sum the distances of each point to its assigned cluster
- Plot the resulting curve

K-means clustering: Elbow method



- The average within-cluster distance typically decreases towards zero
 - Using too many clusters make the results meaningless
- The elbow of the curve is where the drop in within-cluster distances becomes less significant

K-means clustering: Properties

✓ K-means clustering is unsupervised

- It only requires having observations, no additional annotations
- Without annotations, it requires human analysis to provide a semantic meaning to the clusters
- If annotations are available, they can be used to label each cluster, e.g., with a corresponding category label
 - For each cluster, find the number of samples of each category
 - Assign the cluster the category with the largest number of samples

K-means clustering: Properties

✓ The Euclidean distance works well for data with homogeneous dimensions. In the previous examples

- With the 2D toy data, both dimensions are of commensurate magnitude
- With the human pose, each dimension $d \in \{1, \dots, D\}$ represents the same type of information
- With the color image, each dimension represents a color channel and varies in the range [0,255]

✗ In practice, this is not always the case

- Different data dimensions can have completely different magnitudes
- They can encode completely different types of information

Example of data with heterogeneous dims

- Wine dataset from the UCI ML repository
 - 178 wines from 3 different producers
 - Each wine is represented by 13 attributes, such as quantity of alcohol, malic acid and magnesium
- Two samples from the dataset

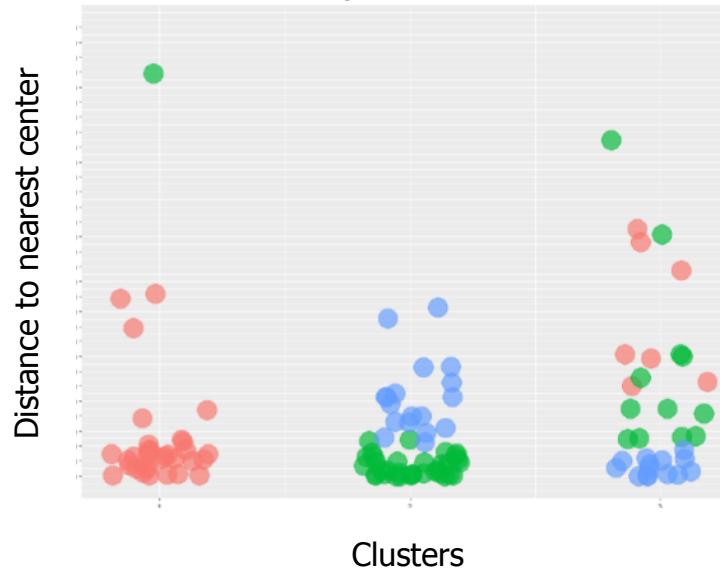
14.37	1.95	2.5	16.8	113	3.85	3.49	0.24	2.18	7.8	0.86 1.04	3.45	1480 735
13.24	2.59	2.87	21	118	2.8	2.69	0.39	1.82	4.32		2.93	

- Large values will contribute a lot to the Euclidean distance
- And small ones much less
- It does not mean that they are less meaningful for clustering!
- Solution: Normalization and/or different distance function

for heterogeneous dims

Wine example (from Dina Jankovic)

- Euclidean distance with raw data
 - The color represents the true producer (ideally all samples in one cluster should have the same color)



- Accuracy: 70.3%
 - Percentage of samples assigned to the correct cluster (producer)

Wine example (from Dina Jankovic)

- Euclidean distance with scaled data



- Accuracy: 93.2%

Wine example (from Dina Jankovic)

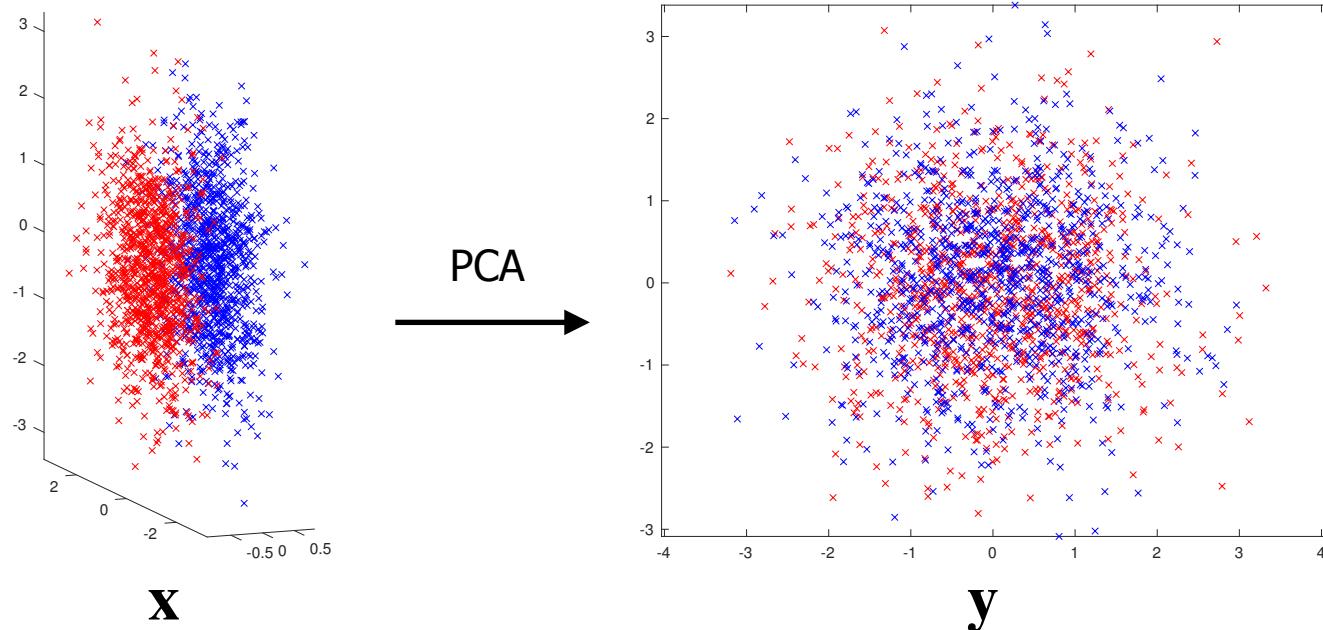
- L^1 distance with scaled data



- Accuracy: 94.5%

Clustering for dimensionality reduction

- The dimensionality reduction algorithms we saw last week were unsupervised and thus may not preserve category information
- Example with PCA: 3D data from 2 classes (colors)



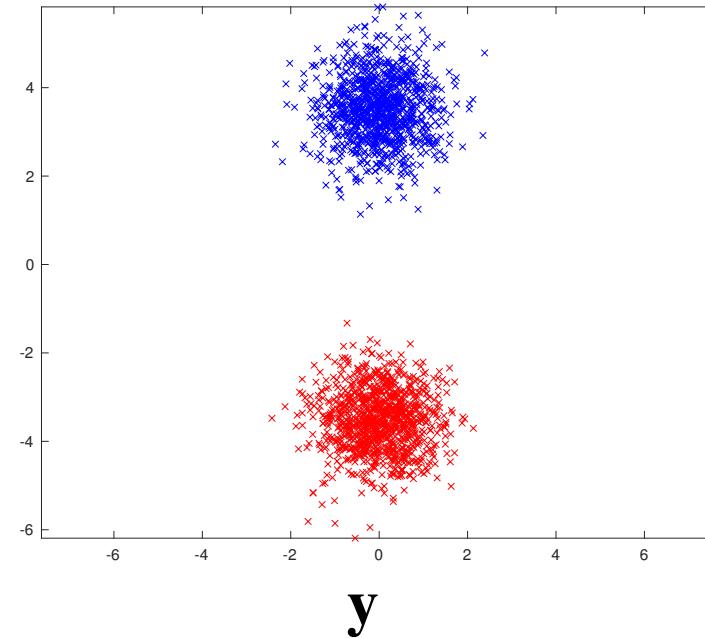
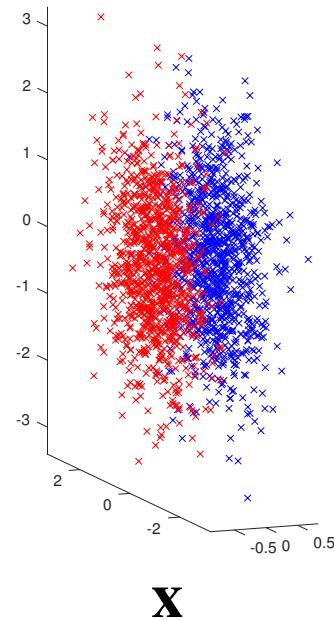
- How about making use of class labels during DR to encourage clustering in the low-dimensional space?

Fisher Linear Discriminant Analysis (LDA) (Bishop 4.1.6)

- Intuitively, we want:

- the samples from the same class to be clustered
- the different classes to be separated

Similar to the k-means clustering intuition



- Note that we are back to supervised learning

- However, the supervision is not given for the $\{y_i\}$ directly but via the class labels

Fisher LDA: Clustering the samples from the same class

- Mathematically, this means that we want a low variance within each class after projection
- For a 1D projection, encoded via a vector $\mathbf{w}_{(1)}$, and C classes, this can be expressed as aiming to minimize

$$E_{\mathbf{W}}(\mathbf{w}_{(1)}) = \sum_{c=1}^C \sum_{i \in c} (y_i - \nu_c)^2$$

where ν_c is the mean of the samples in class c after projection, and $i \in c$ indicates that sample i belongs to class c

Note that both y_i and ν_c depend on $\mathbf{w}_{(1)}$

Fisher LDA: Clustering the samples from the same class

- As in the PCA case, the variance after projection is equal to the projection of the covariance matrix
- This lets us rewrite the previous objective function as

$$E_W(\mathbf{w}_{(1)}) = \mathbf{w}_{(1)}^T \mathbf{S}_W \mathbf{w}_{(1)}$$

where

$$\mathbf{S}_W = \sum_{c=1}^C \sum_{i \in c} (\mathbf{x}_i - \boldsymbol{\mu}_c)(\mathbf{x}_i - \boldsymbol{\mu}_c)^T$$

with $\boldsymbol{\mu}_c$ the mean of the data in class c *before* projection

- \mathbf{S}_W is referred to as the within-class scatter matrix

Fisher LDA: Separating the different classes

- In addition to clustering the samples according to the classes, we want to separate the different clusters
- This can be achieved by pushing the means of the clusters away from each other
- Mathematically, this can be expressed as maximizing

$$E_B(\mathbf{w}_{(1)}) = \sum_{c=1}^C N_c (\nu_c - \bar{y})^2$$

where ν_c is defined as before, \bar{y} is the mean of all samples after projection, and N_c is the number of samples in class c

Fisher LDA: Separating the different classes

- Again, following the same reasoning as before, this can be re-written as

$$E_B(\mathbf{w}_{(1)}) = \mathbf{w}_{(1)}^T \mathbf{S}_B \mathbf{w}_{(1)}$$

where

$$\mathbf{S}_B = \sum_{c=1}^C N_c (\mu_c - \bar{\mathbf{x}})(\mu_c - \bar{\mathbf{x}})^T$$

with $\bar{\mathbf{x}}$ the mean of all the samples and $\{\mu_c\}$ the class-specific means

- \mathbf{S}_B is referred to as the between-class scatter matrix

Fisher LDA

- Altogether, we want to
 - minimize $E_W(\mathbf{w}_{(1)})$
 - maximize $E_B(\mathbf{w}_{(1)})$
- This can be achieved by maximizing

$$J(\mathbf{w}_{(1)}) = \frac{E_B(\mathbf{w}_{(1)})}{E_W(\mathbf{w}_{(1)})} = \frac{\mathbf{w}_{(1)}^T \mathbf{S}_B \mathbf{w}_{(1)}}{\mathbf{w}_{(1)}^T \mathbf{S}_W \mathbf{w}_{(1)}}$$

because, in general, minimizing a function $f(\cdot)$ can be achieved by maximizing $1/f(\cdot)$

Fisher LDA

- The previous objective function is invariant to scaling, i.e.,

$$J(\alpha \mathbf{w}_{(1)}) = J(\mathbf{w}_{(1)}) \quad \text{for any } \alpha \in \mathbb{R}$$

- So we can fix the scale by constraining $\mathbf{w}_{(1)}$ to be such that

$$\mathbf{w}_{(1)}^T \mathbf{S}_W \mathbf{w}_{(1)} = 1$$

- This yields the Fisher LDA formulation

$$\max_{\mathbf{w}_{(1)}} \mathbf{w}_{(1)}^T \mathbf{S}_B \mathbf{w}_{(1)}$$

$$\text{s.t. } \mathbf{w}_{(1)}^T \mathbf{S}_W \mathbf{w}_{(1)} = 1$$

Fisher LDA

- To solve this, we again rely on the Lagrangian, written as

$$L(\mathbf{w}_{(1)}, \lambda_1) = \mathbf{w}_{(1)}^T \mathbf{S}_B \mathbf{w}_{(1)} + \lambda_1 (1 - \mathbf{w}_{(1)}^T \mathbf{S}_W \mathbf{w}_{(1)})$$

- Zeroing out the gradient of $L(\cdot)$ w.r.t. $\mathbf{w}_{(1)}$ yields

$$\mathbf{S}_B \mathbf{w}_{(1)} = \lambda_1 \mathbf{S}_W \mathbf{w}_{(1)}$$

- This implies that $\mathbf{w}_{(1)}$ must be the solution to a generalized eigenvector problem
- Left-multiplying both sides by $\mathbf{w}_{(1)}^T$ and dividing by $\mathbf{w}_{(1)}^T \mathbf{S}_W \mathbf{w}_{(1)}$ tells us that $\mathbf{w}_{(1)}$ should be the eigenvector with largest eigenvalue

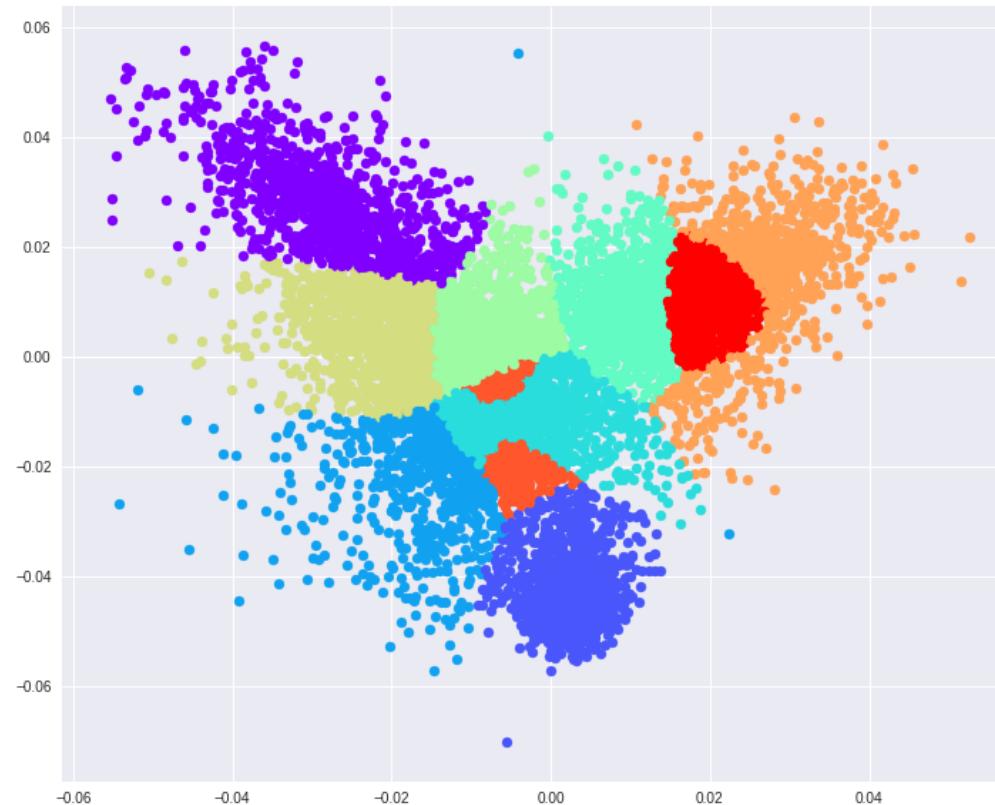
Fisher LDA: Dealing with $d > 1$

- To project the data to more than a single dimension, we can follow a recursive strategy similar to the PCA one
 - Ultimately, this consists of taking the d eigenvectors with largest eigenvalues
- Note that it can be shown that \mathbf{S}_B has rank at most $C - 1$
- Therefore, we can project the data only to at most $C - 1$ dimensions
 - The remaining eigenvalues will all be 0, and thus carry no information

Fisher LDA: Example

- Visualizing MNIST:

- Example from <https://sthalles.github.io/fisher-linear-discriminant/>
- In 2D

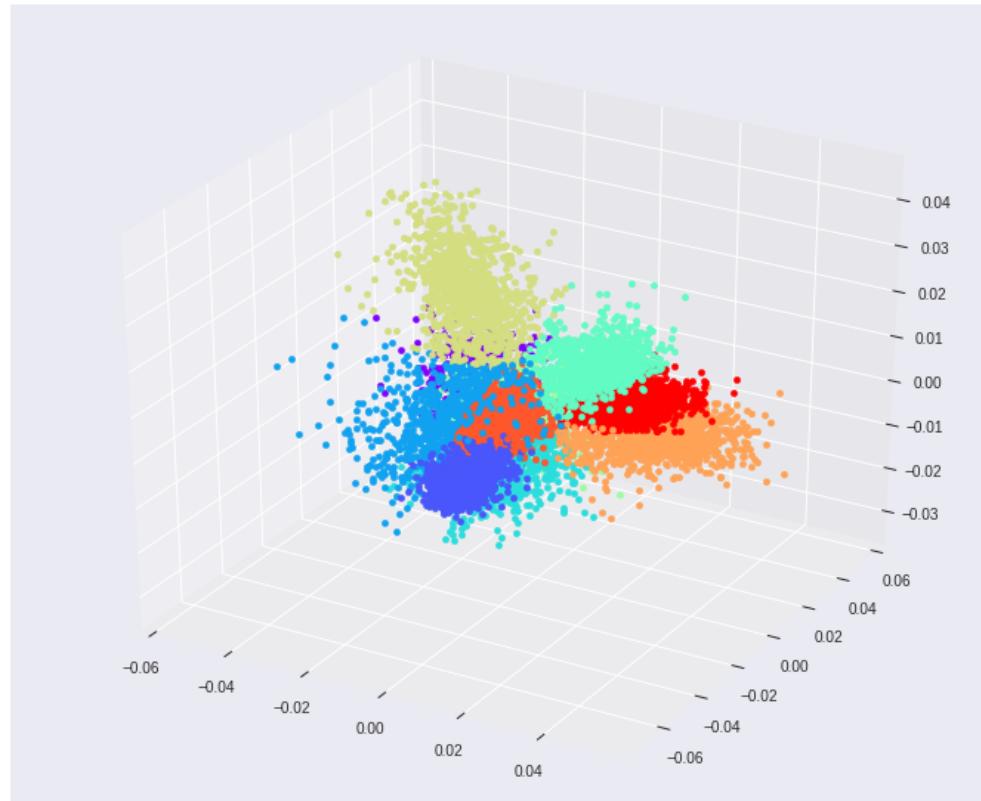


- Classification accuracy in the resulting space: 56%

Fisher LDA: Example

- Visualizing MNIST:

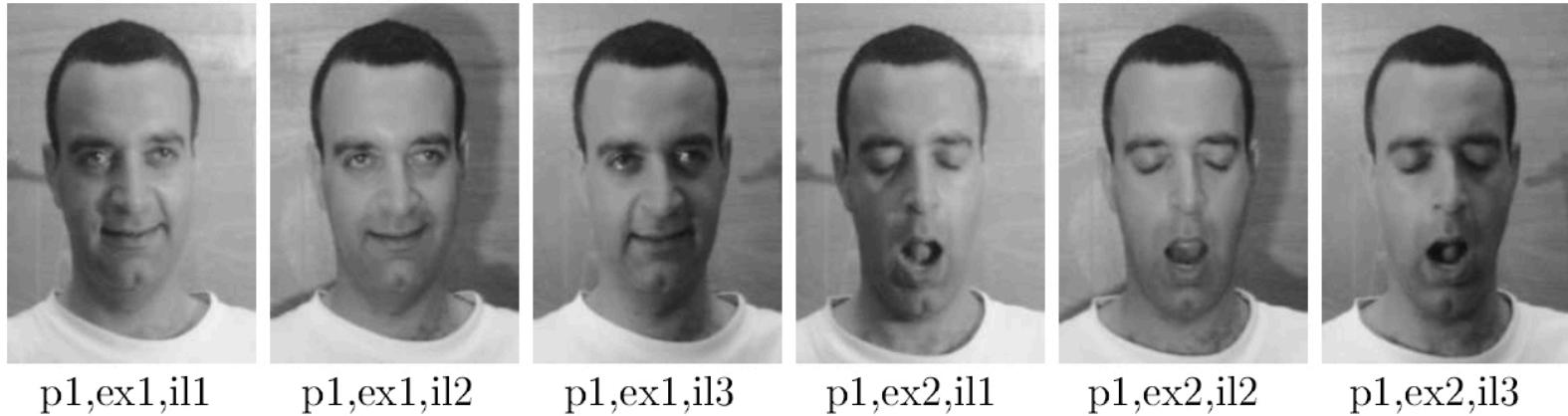
- Example from <https://sthalles.github.io/fisher-linear-discriminant/>
- In 3D



- Classification accuracy in the resulting space: 74%

Eigenfaces vs Fisherfaces

- Consider a dataset of face images
 - 2 different expressions (treated as categories)
 - several illumination conditions



- One can apply either PCA or LDA to these images
 - the resulting eigenvectors can also be thought of as images
 - they are called eigenfaces for PCA and fisherfaces for LDA

Eigenfaces

- The eigenfaces yield the lowest reconstruction error
 - As such, they contain information about the illumination conditions



Image reconstruction:



$$= \bar{\mathbf{x}} + y^{(1)} *$$



$$+ y^{(2)} *$$



$$+ y^{(3)} *$$



$$+ \dots$$

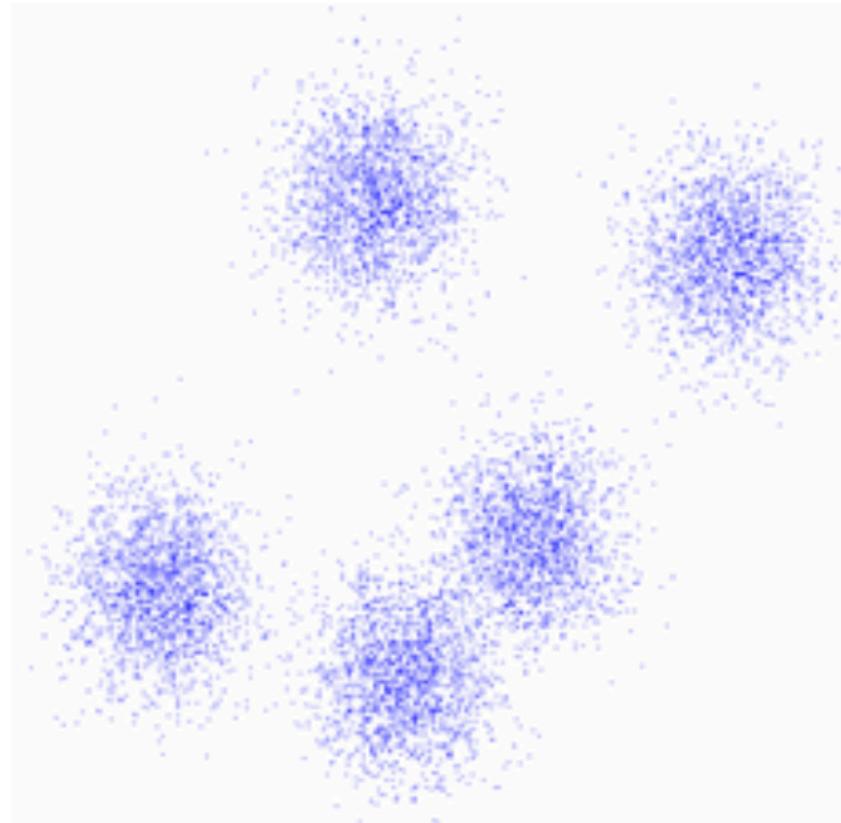
Fisherfaces

- The fisherfaces are those that are the most discriminative of the expression
 - As such, they discard illumination information, which is irrelevant to this task



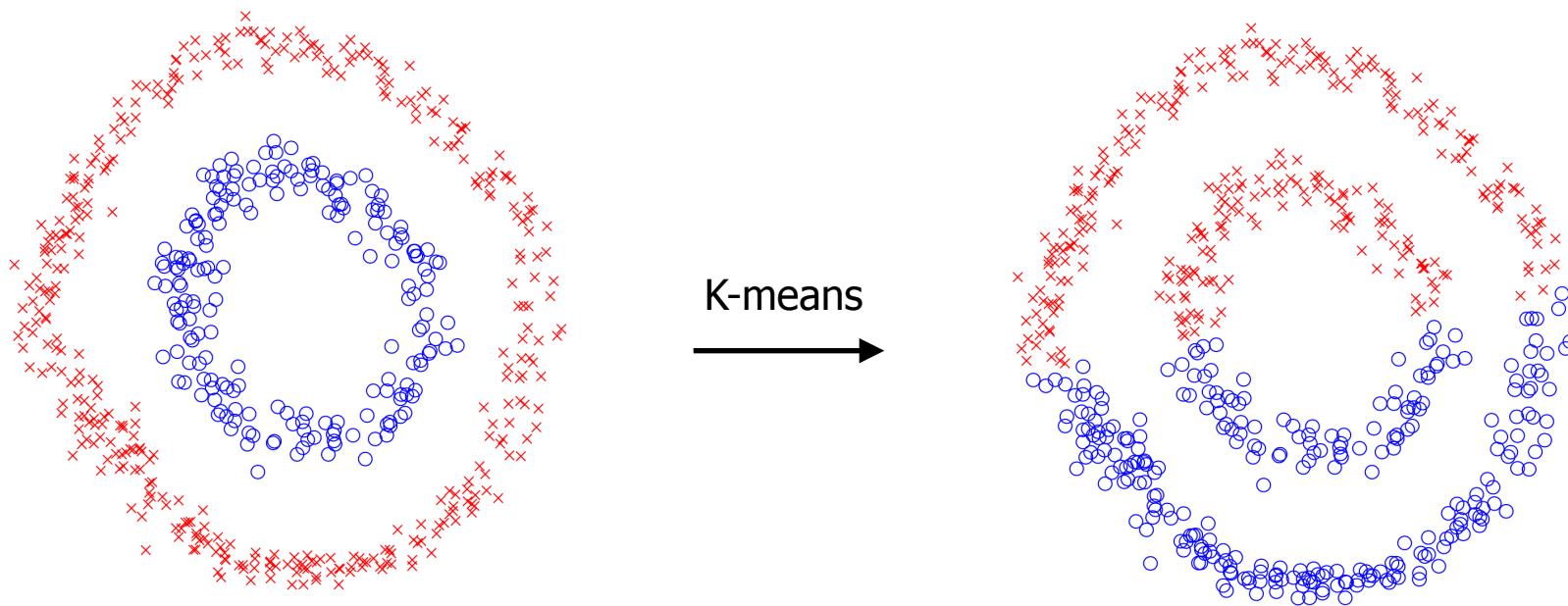
Clustering: Compactness

- K-means clustering (and Fisher LDA) exploit the notion of compactness of clusters



Clustering: Compactness

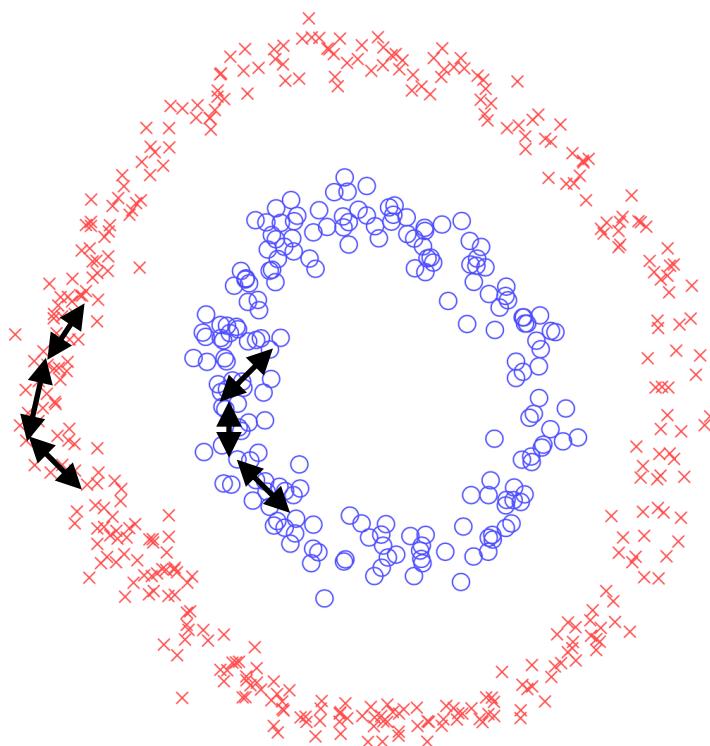
- What would happen if the data is not compact
 - E.g., two concentric circles



- Yet, we still clearly identify the two “true” clusters

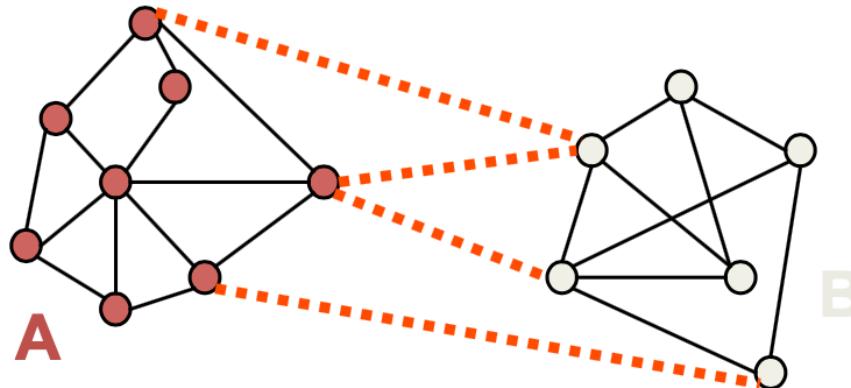
Clustering: Connectivity

- The two clusters we observe arise from the connectivity of the points



Spectral clustering: Graph-based connectivity

- Group the points based on edges in a graph
 - Strong connections indicate points that should be clustered
 - Weaker ones suggest that the graph can be cut into pieces/partitions



How to create the graph?

- Compute a notion of similarity between the points
 - E.g., the similarity between point i and point j can be taken as

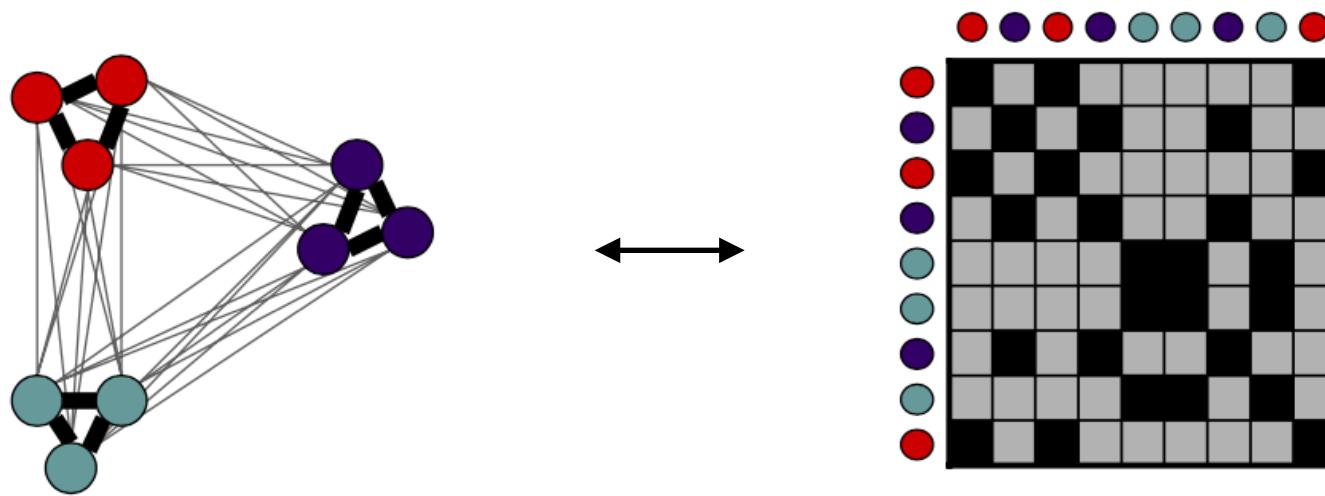
$$W_{ij} = \exp\left(\frac{-\|\mathbf{x}_i - \mathbf{x}_j\|^2}{\sigma^2}\right)$$

where σ is a hyper-parameter

- This would lead to a fully connected graph (an edge with a weight W_{ij} for every pair of points)
 - The graph can also be restricted to the K-nearest neighbor of each point

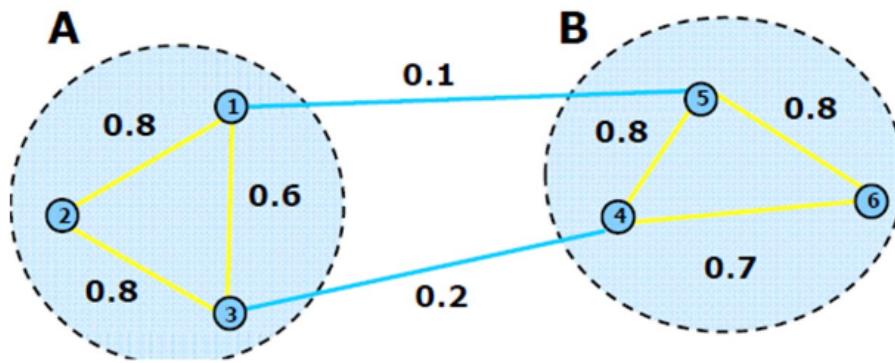
From graph to similarity matrix

- A graph can be equivalently represented by a similarity (or affinity) matrix



Graph cut

- Consider a partition of a graph into two parts A and B



- A cost for cut is obtained by summing the weight of the edges that connect the two groups

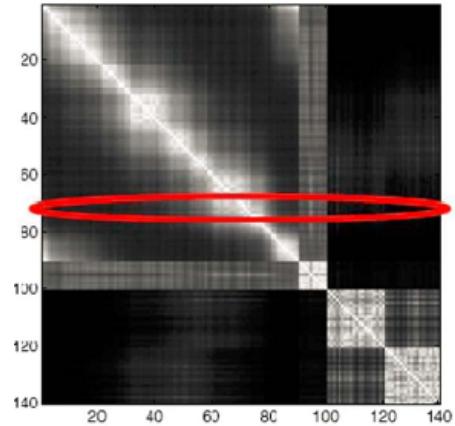
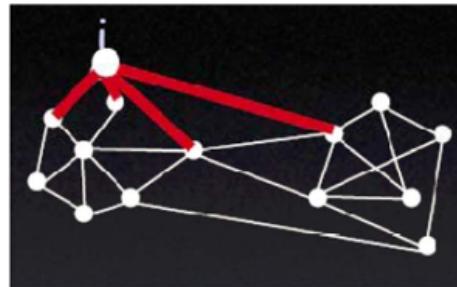
$$cut(A, B) = \sum_{i \in A, j \in B} W_{ij} \quad (=0.3 \text{ in this example})$$

- Intuitively, for clustering, we would like to find the partition that minimizes such a cut

Graph terminology

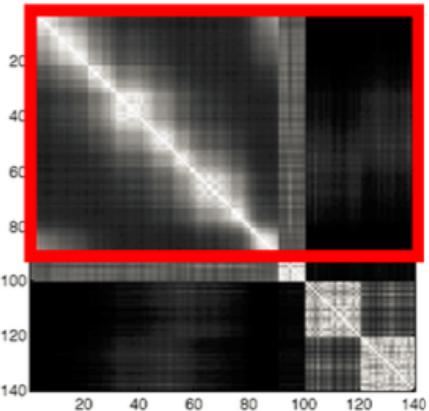
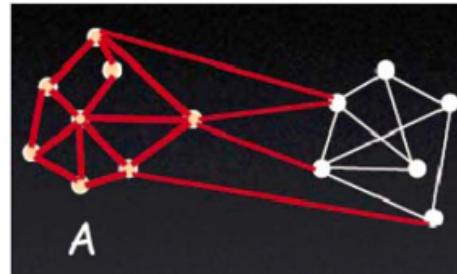
- The degree of a node in the graph is given by

$$d_i = \sum_j W_{ij}$$



- The volume of a set (partition) is given by

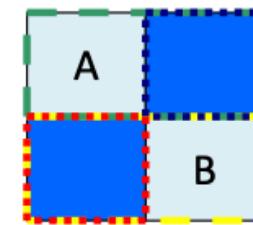
$$vol(A) = \sum_{i \in A} d_i$$



Normalized cut

- Minimizing the cut might favor imbalanced partitions
 - E.g., a single point in A and all the others in B
- Instead, we can use a normalized cut

$$Ncut(A, B) = \frac{cut(A, B)}{Vol(A)} + \frac{cut(A, B)}{Vol(B)}$$



where $Vol(A)$ is the volume of partition A , defined in the previous slide

Normalized cut: Transformed formulation

- Finding the partition that minimizes the normalized cut is NP hard
- This can be re-written as the optimization problem

$$\min_{\mathbf{y}} \frac{\mathbf{y}^T(\mathbf{D} - \mathbf{W})\mathbf{y}}{\mathbf{y}^T \mathbf{D} \mathbf{y}}$$

$$\text{s.t. } \mathbf{y}^T \mathbf{D} \mathbf{1} = 0$$

$$\mathbf{y}_i \in \{1, -b\}, \quad \forall 1 \leq i \leq N$$

where $\mathbf{W} \in \mathbb{R}^{N \times N}$ is the similarity matrix between all pairs of points

$\mathbf{D} \in \mathbb{R}^{N \times N}$ is the diagonal degree matrix, with $\mathbf{D}_{ii} = d_i$

\mathbf{y} indicates to which partition each point belongs

- However, this remains an NP hard problem

Normalized cut: Relaxation

- This problem can then be relaxed as

$$\min_{\mathbf{y}} \mathbf{y}^T (\mathbf{D} - \mathbf{W}) \mathbf{y}$$

$$\text{s.t. } \mathbf{y}^T \mathbf{D} \mathbf{y} = 1$$

- Using the Lagrangian, the solution can be expressed as a generalized eigenvalue problem

$$(\mathbf{D} - \mathbf{W}) \mathbf{y} = \lambda \mathbf{D} \mathbf{y}$$

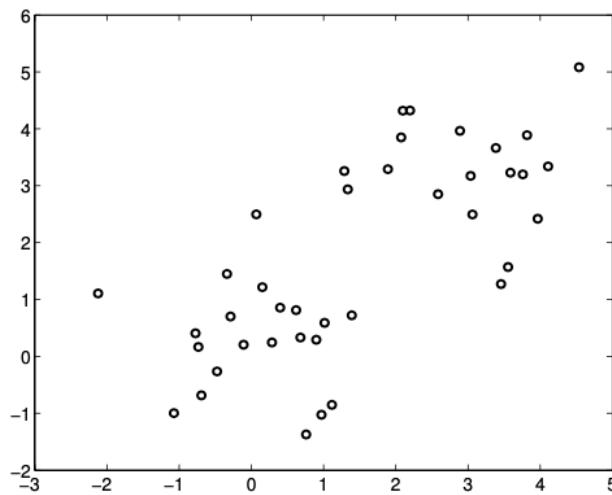
- Side note: $(\mathbf{D} - \mathbf{W})$ is referred to as the graph Laplacian

Normalized cut: Which eigenvector

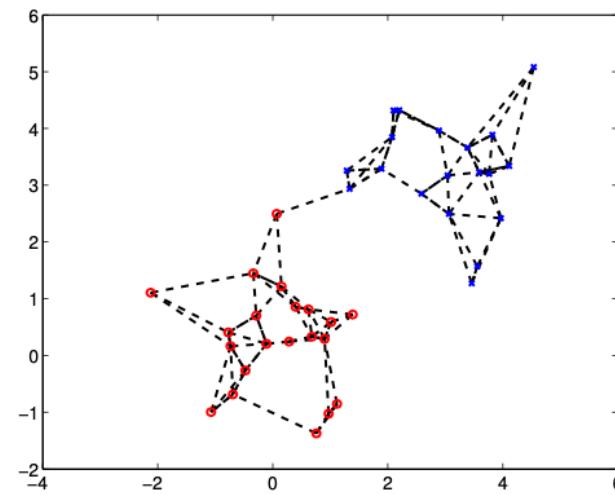
- The eigenvector with the smallest eigenvalue is a vector of all 1s
 - Its eigenvalue is 0
- The solution is then obtained by the eigenvector with the second smallest eigenvalue
 - Ideally, a positive value in this vector indicates that the corresponding point belongs to one partition, and a negative value to the other
 - Because of the relaxation, this is not so ideal; one then needs to threshold the values (e.g., by taking the median value as threshold for balanced data)

Normalized cut: Example

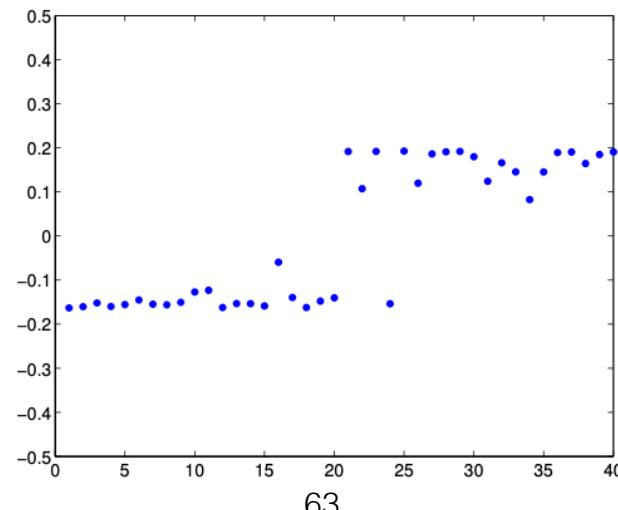
Input data



Graph and partition



2nd eigenvector

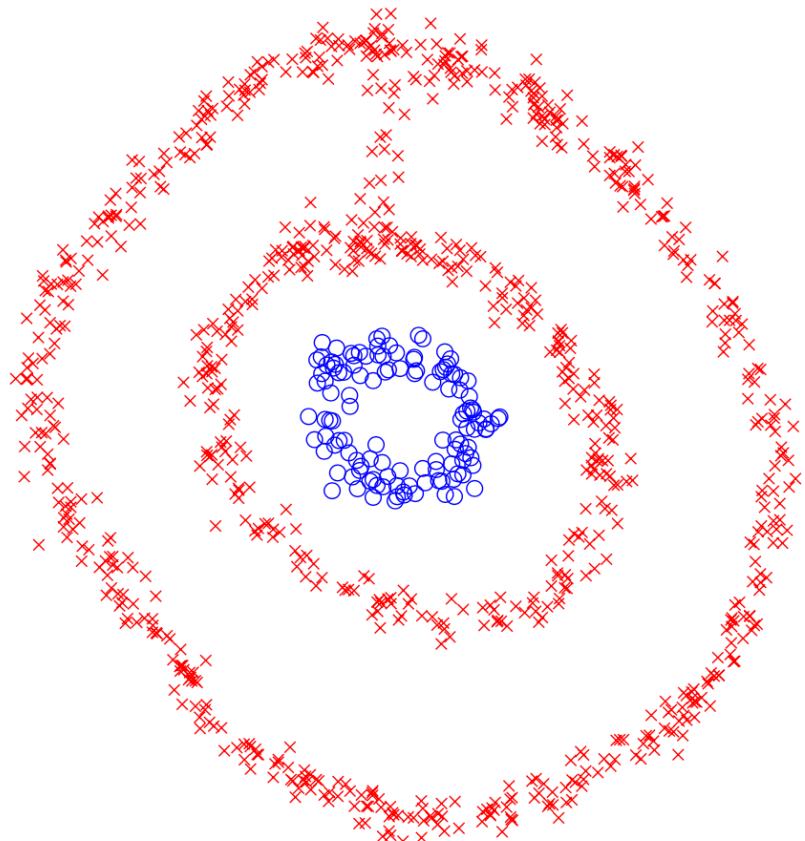


K-way partition

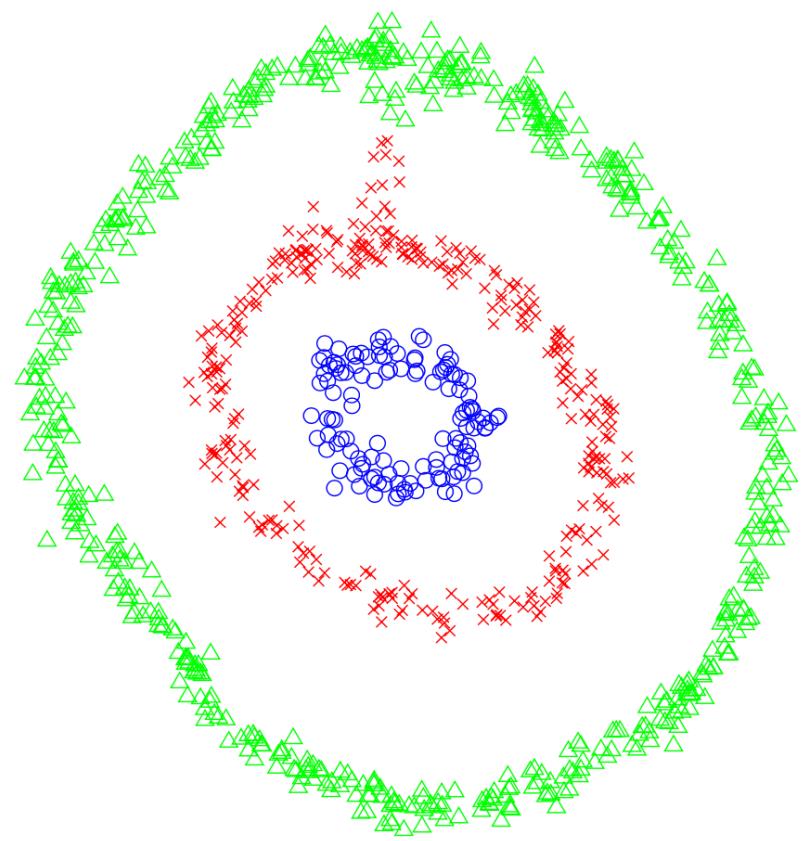
- To obtain more than 2 clusters, one can either
 1. Recursively apply the 2-way partitioning algorithm
 - Not efficient and unstable
 2. Use multiple (i.e., K) eigenvectors
 - Each point is represented as a K -dimensional vector
 - Apply K -means clustering to the resulting N vectors
 - Interpretation: Dimensionality reduction followed by K -means

K-way partition: Example

2 clusters

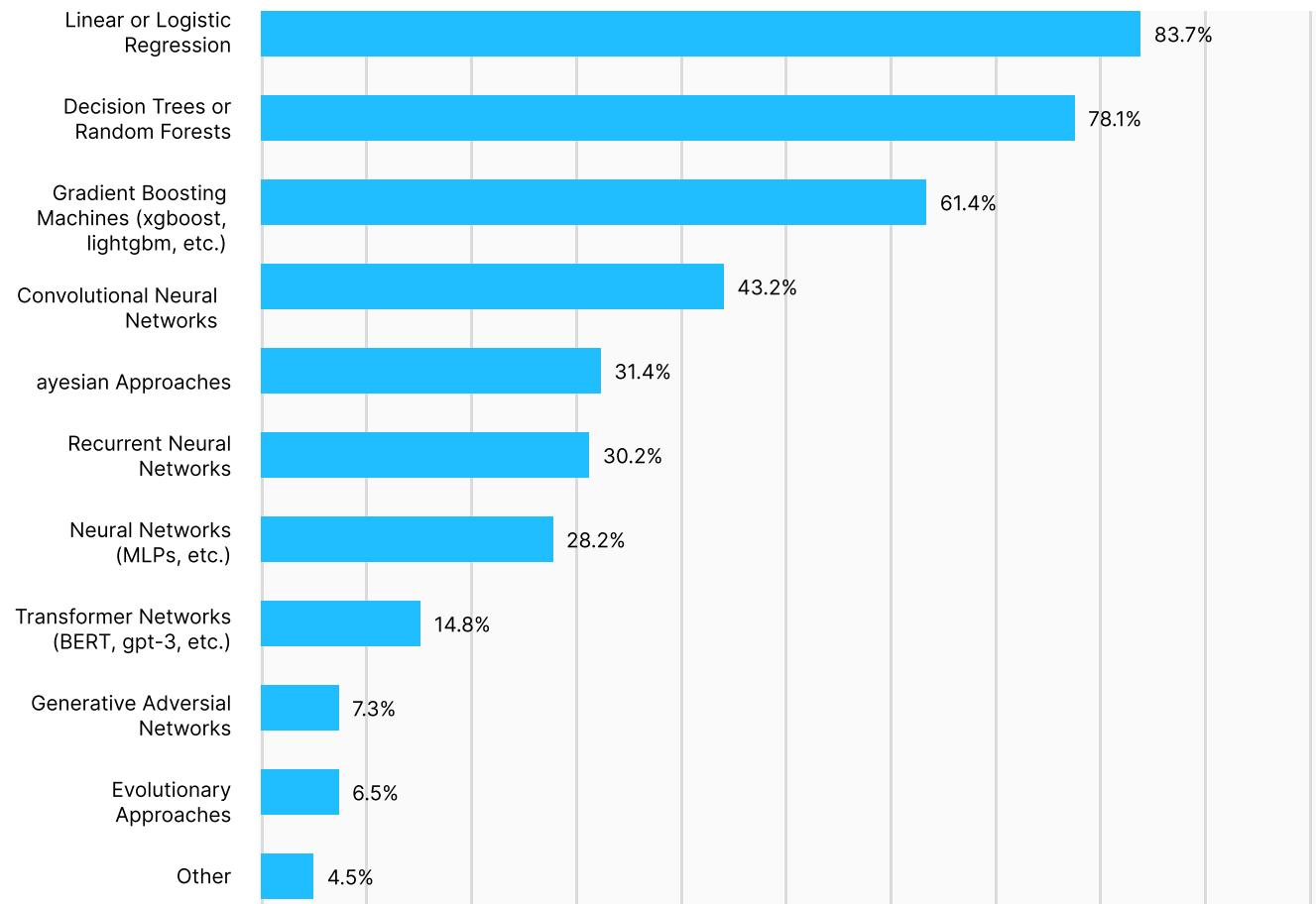


3 clusters



Machine Learning in practice

- According to a 2020 survey among data scientist, the answer to the question “What kind of data science methods are used at work” gave the statistics



Machine Learning in practice

- Here is the 2021 version of the same survey

