DEEP LEARNING: INTRODUCTION

Caio Corro

ABOUT THE COURSE

http://caio-corro.fr/class_dl.html

All information will be on this website

Summary of the course

- Introduction to neural networks
- Efficient training
- Convolutional neural networks and a few other architectures
- Generative models (GAN, VAE)
- Introduction to Pytorch

Teachers

- Lectures: Michèle Sebag and Caio Corro
- Lab exercises: Caio Corro

Grading scheme

- 50% : Lab exercises
- 50% : Exam

REQUIREMENTS

Background

- Basics in machine learning
- Derivative computation
- Python

Programming

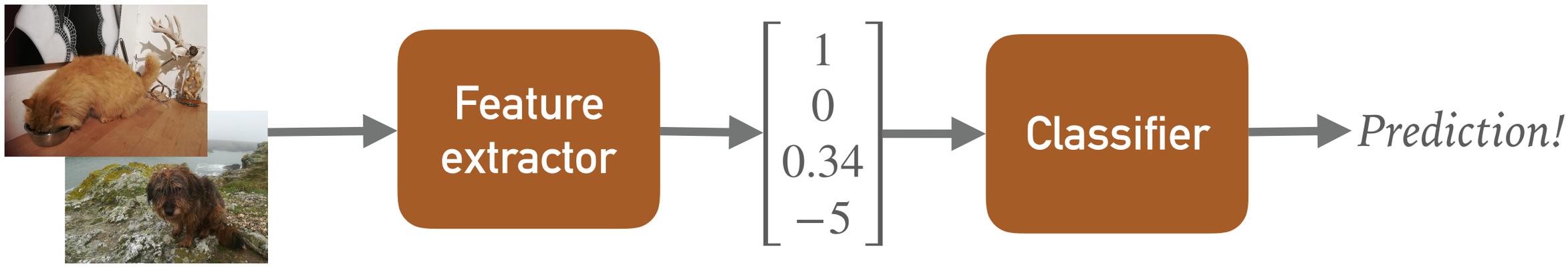
- Python 3 + Jupyter notebook
- Libraries: numpy, matplotlib, pytorch

Other

- Pen + paper!

PRE-DEEP LEARNING ERA

The « old school » machine learning pipeline

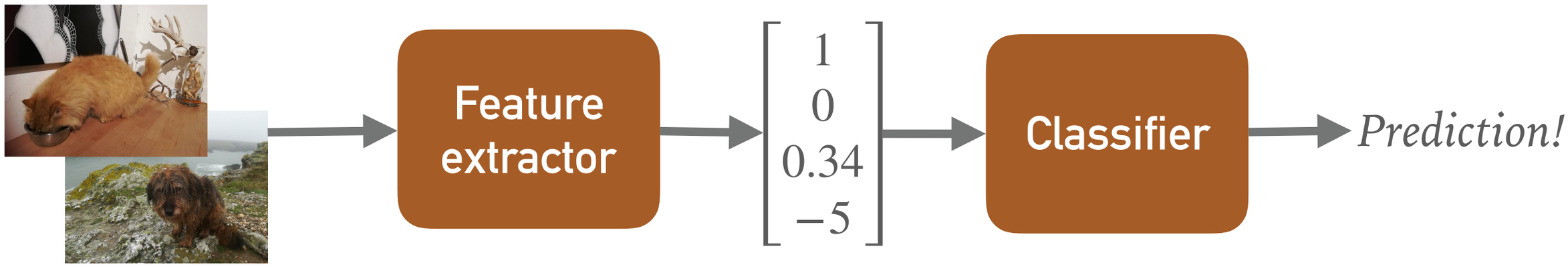


Feature extraction

- Problem dependent
  - Images : SIFT features, invariant to translation, scaling, etc.
  - Text : Stemming, lemmatisation
- Automatic or manual
- Raw data (sometimes…)

PRE-DEEP LEARNING ERA

The « old school » machine learning pipeline

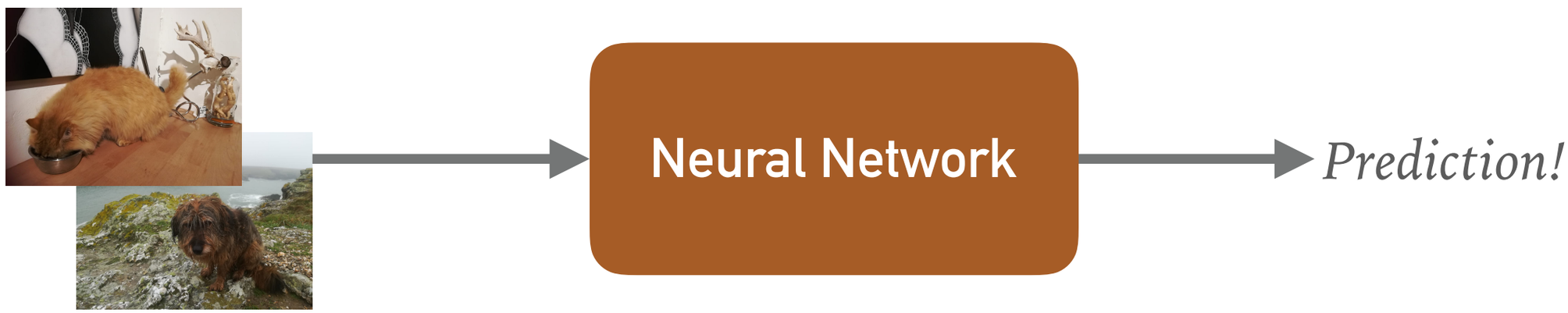


Example of classifiers

- Decision Tree:
  - Make a decision considering a limited number of features
  - Use conjunction of features to make a prediction
- K-nearest neighbors:
  - All features are used and considered equals
- Perceptron/linear classifier:
  - Weight features so they are more or less important to make a decision

DEEP LEARNING

The deep learning « pipeline »



What's the difference?

- No (or limited) feature extraction: use raw data as input!
- Complicated classifier: a neural network is (really) big non-convex function

Neural architecture design

- What kind of parameterized mathematical functions?
  - Image input: Convolutions? or others.
  - Text input: Recurrent neural networks? or others.
- How many parameters?
- How many layers?

Equivariant to translation

Take into account the sequential nature of the input

BUILDING NEURAL NETWORKS

Architecture design

Neural network = complicated parameterized function

- Inductive bias: take into account the data properties to design the architectures
- Time complexity/speed
- Mathematical properties for efficient training:
  differentiability, prevent vanishing/exploding gradient, ...

Parameter optimization

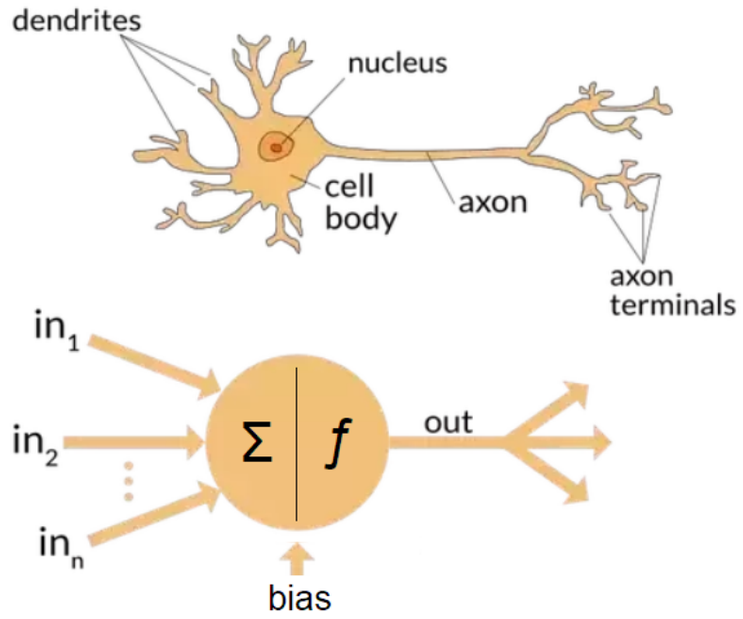
- Efficient optimization algorithms (i.e. first order gradient-based methods)
- Prevent overfitting
- Parallelized training (outside the scope of this course)

PERCEPTRON / LINEAR CLASSIFICATION

BIOLOGICAL INSPIRATION OF THE PERCEPTRON

Roughly

- A (brain) neuron receives signals
- Depending on the value of the signals, it triggers or not an output



Bio-inspired classifier

- Perceptron/neural network neurons have most probably been inspired from biology
- BUT they are different and work differently
- **The biology comparison is limited and misleading**
- **99.99% of Machine Learning is unrelated to human brain**
  => don't use human brain motivation/inspiration/comparaison (PLEASE DON'T)

Picture stolen from: https://www.quora.com/What-is-the-differences-between-artificial-neural-network-computer-science-and-biological-neural-network

MAIN IDEA

Classifier

Parameterized function $f_\theta : \mathcal{X} \to \mathcal{Y}$

Input space

Parameters

Score/output space

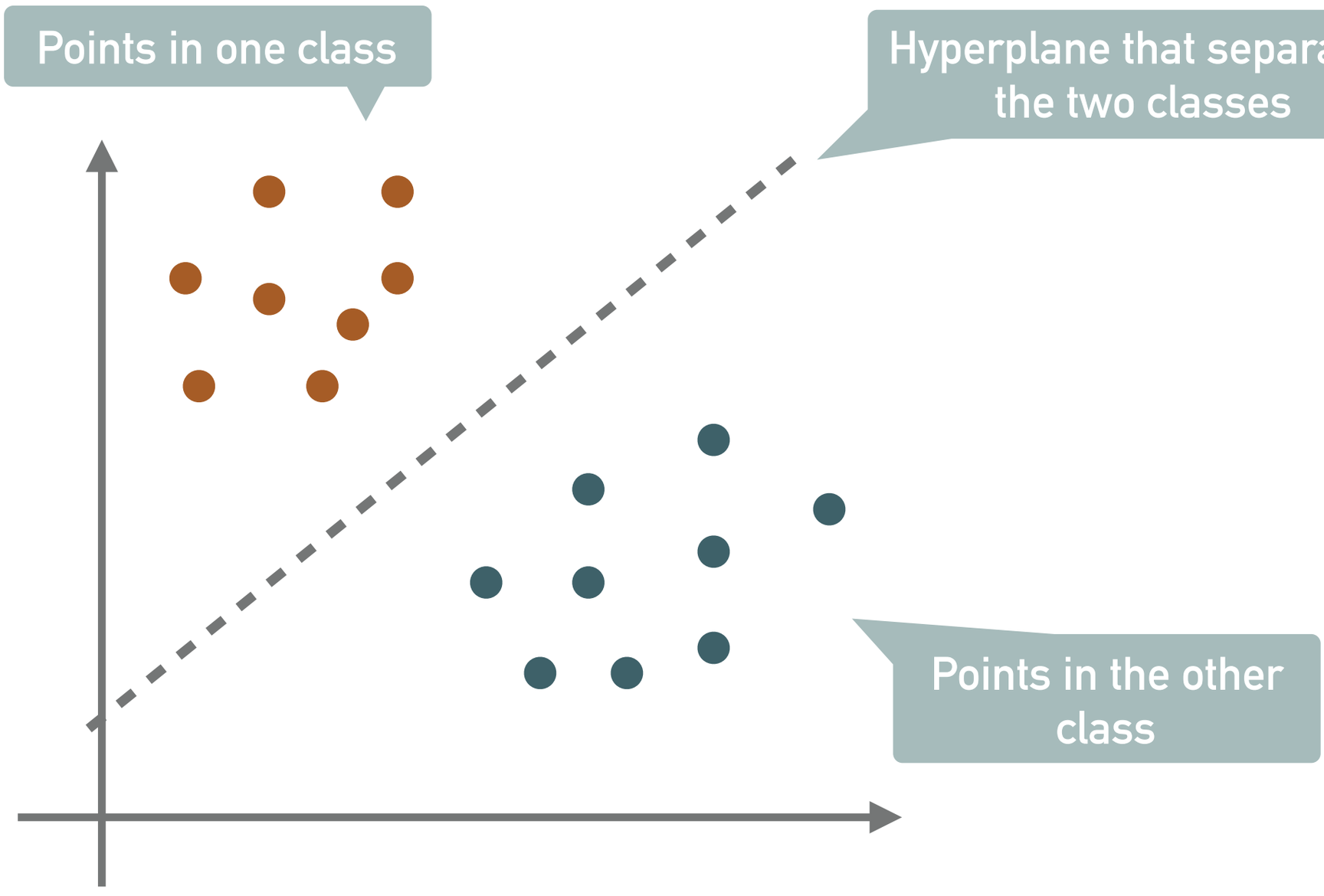
Different types of prediction

- Regression: predict a scalar value
- Binary classification: yes/no, cat/dog, etc
- Multiclass classification: cat/dog/mice/…

Training/learning

- Search for the best parameters θ,
  that is the parameters that make the function predict the correct output

BINARY LINEAR CLASSIFIER: INTUITION



DOT PRODUCT 1/2

Let $a \in \mathbb{R}^n$ and $a \in \mathbb{R}^n$ be two vectors.

$$a = \begin{bmatrix} a_1 \\ a_2 \\ \dots \\ a_n \end{bmatrix} \qquad x = \begin{bmatrix} x_1 \\ x_2 \\ \dots \\ x_n \end{bmatrix}$$

The dot product is defined as: $a^\top x = \langle a, x \rangle = \sum_{i=1}^{n} a_i \times x_i$

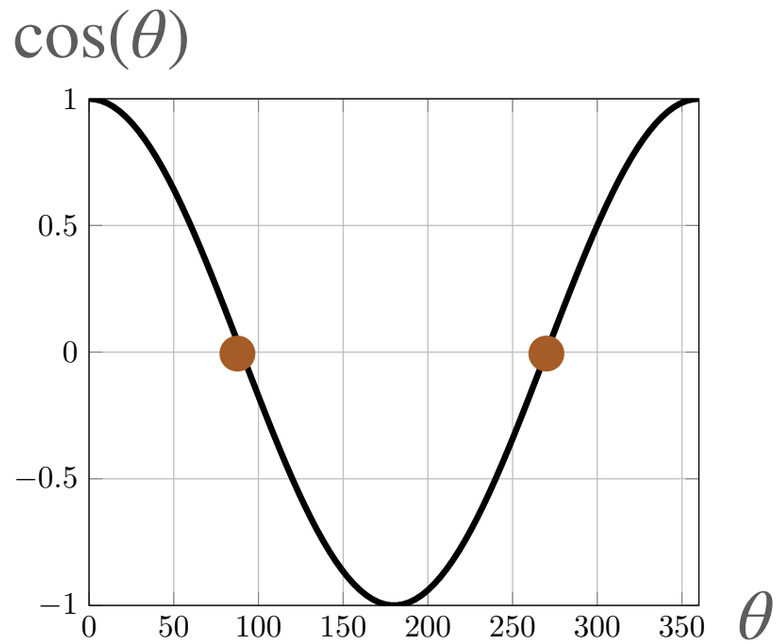
Transpose and matrix multiplication

Properties

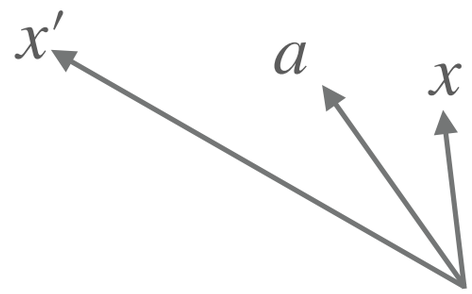
- $a^\top x = \|a\| \|x\| \cos\theta$ where $\|w\|$ is the magnitude of the vector: $\|a\| = \sqrt{\sum_{i=1}^{n} a_i^2}$
- $a^\top x = 0$ if and only if vectors a and x are orthogonal

DOT PRODUCT 2/2

- $a^\top x = \|a\| \|x\| \cos\theta$
- $\|a\| \geq 0$

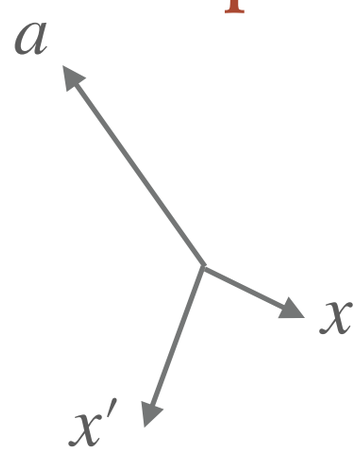


Positive dot product



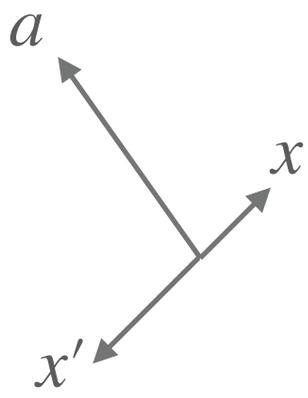
$a^\top x > 0 \quad a^\top x' > 0$

Negative dot product



$a^\top x < 0 \quad a^\top x' < 0$

Null dot product



$a^\top x = 0 \quad a^\top x' = 0$

BINARY LINEAR CLASSIFIER: DEFINITION

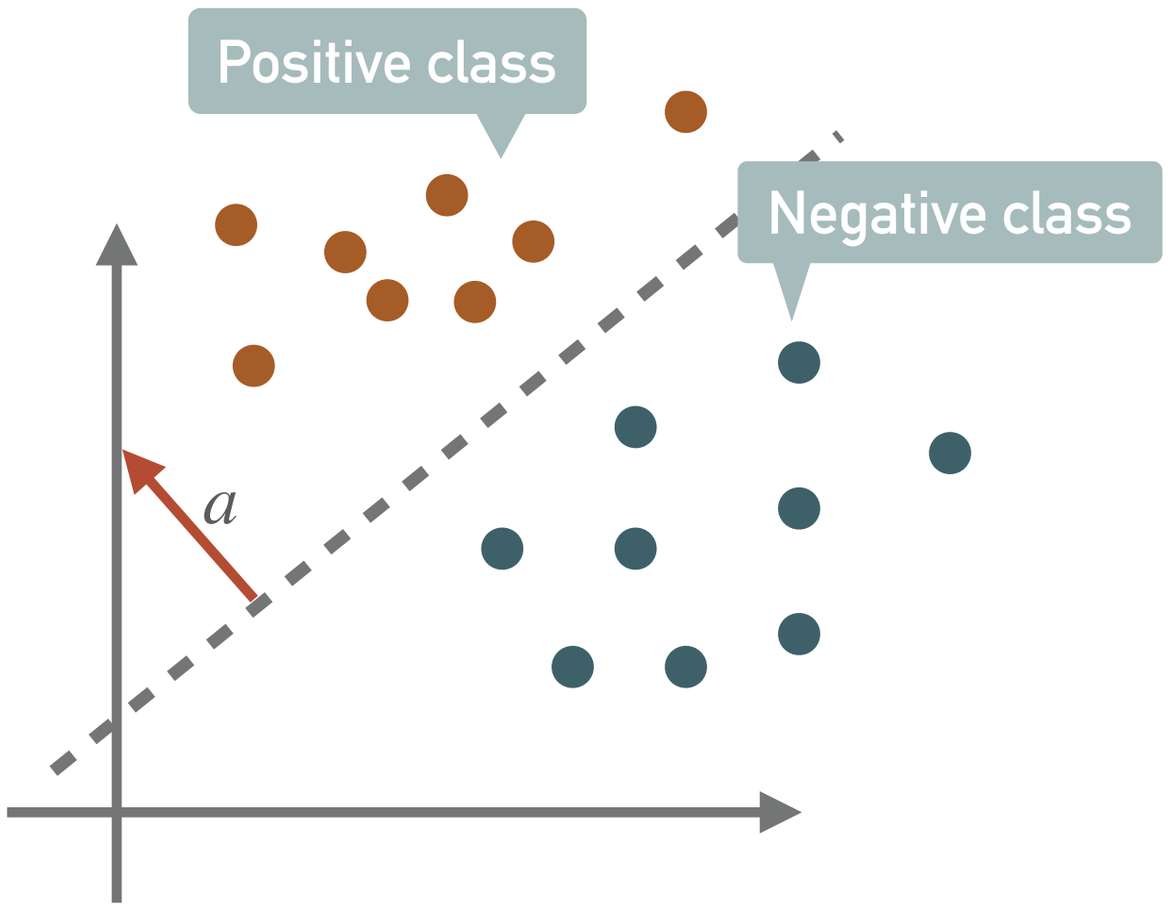
Classification function

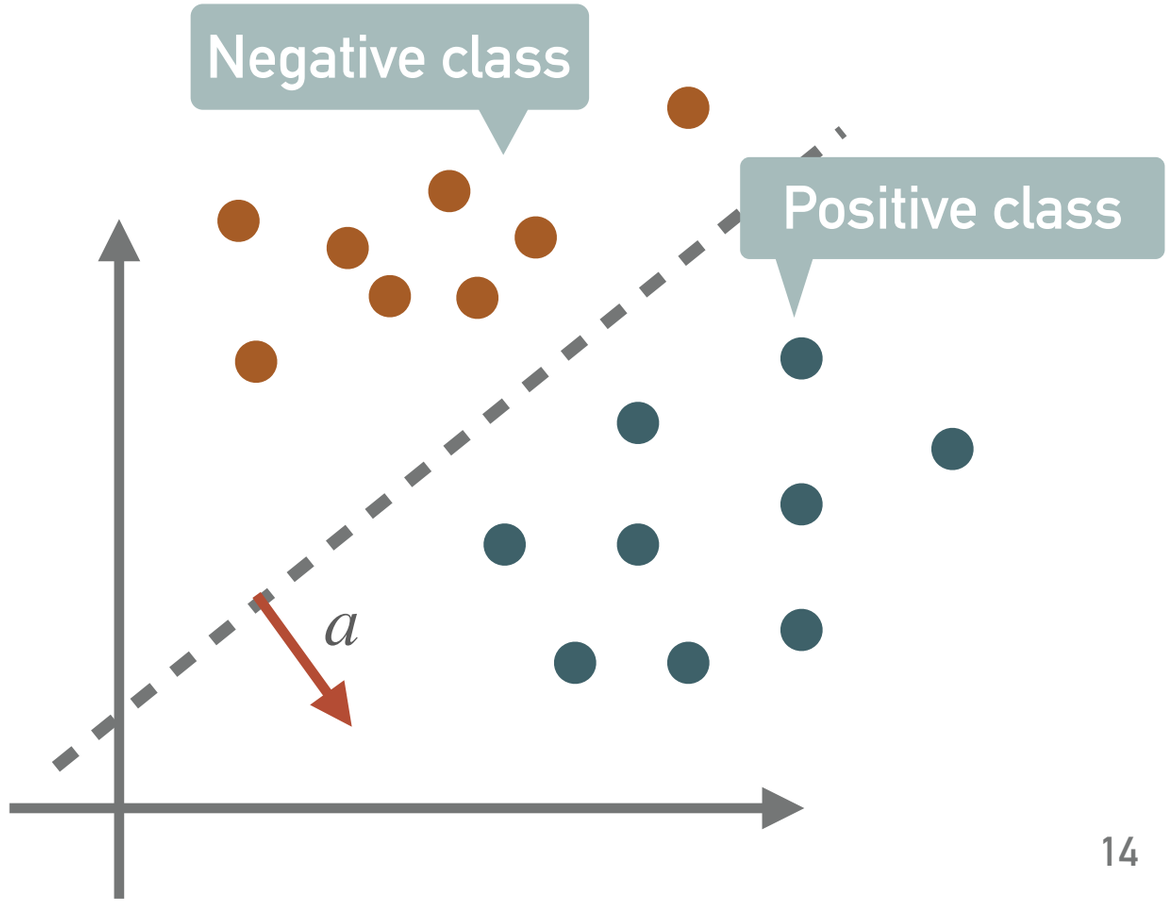
- In general: $f_\theta : \mathcal{X} \to \mathcal{Y}$
- Binary case: $f_\theta : \mathbb{R}^n \to \{-1, 1\}$

Perceptron

- Let the parameters be $\theta = \{a, b\}$
- Classification function:

$$f_\theta(x) = \begin{cases} -1 & if \quad a^\top x + b \leq 0, \\ 1 & if \quad a^\top x + b > 0. \end{cases}$$

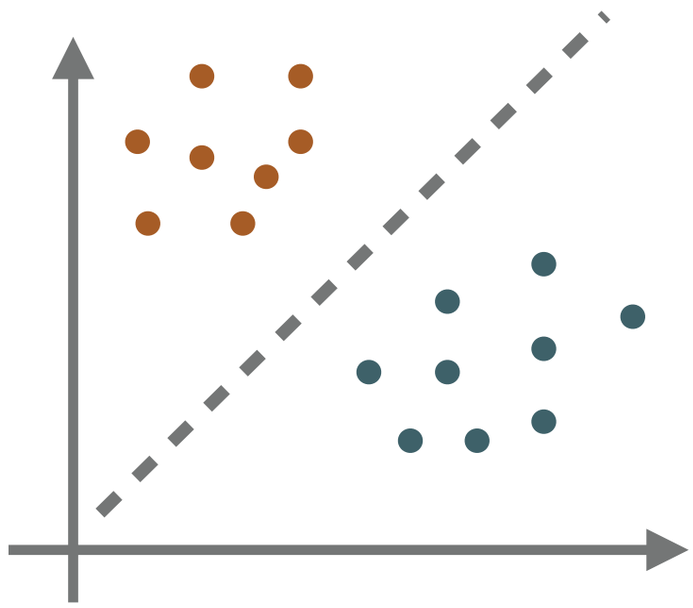




PERCEPTRON FOR BINARY CLASSIFICATION

In a nutshell

$$f_\theta(x) = \begin{cases} -1 & if \quad a^\top x + b \leq 0, \\ 1 & if \quad a^\top x + b > 0. \end{cases}$$



- Parameters: $\theta = \{a, b\}$
- Decision boundary is the set of points that solves:

$$a^\top x + b = 0$$

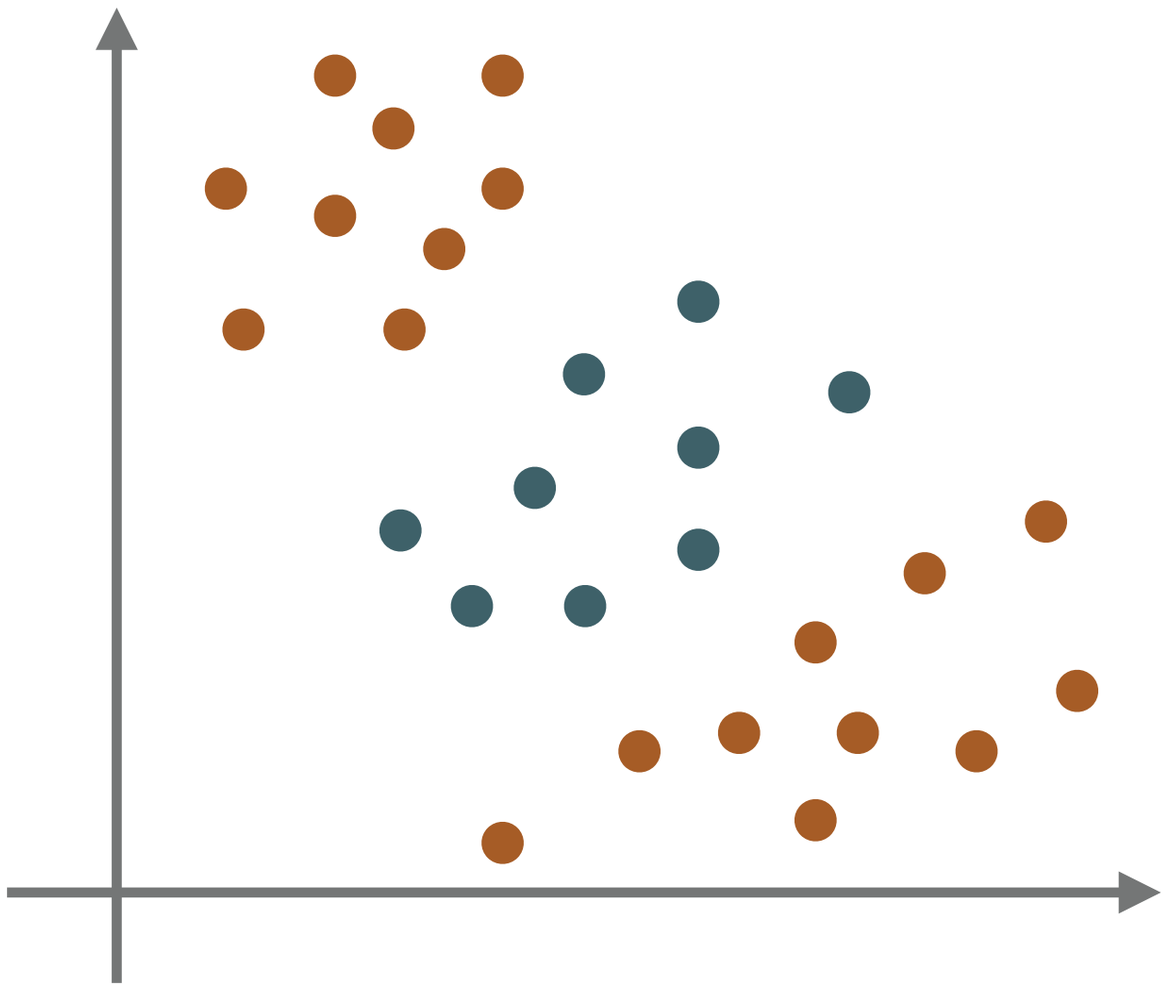
- The decision boundary is an hyperplane

Remaining questions

- Does an hyperplane that separates data always exists?
- How do we find this hyperplane, i.e. how do we compute w and b?

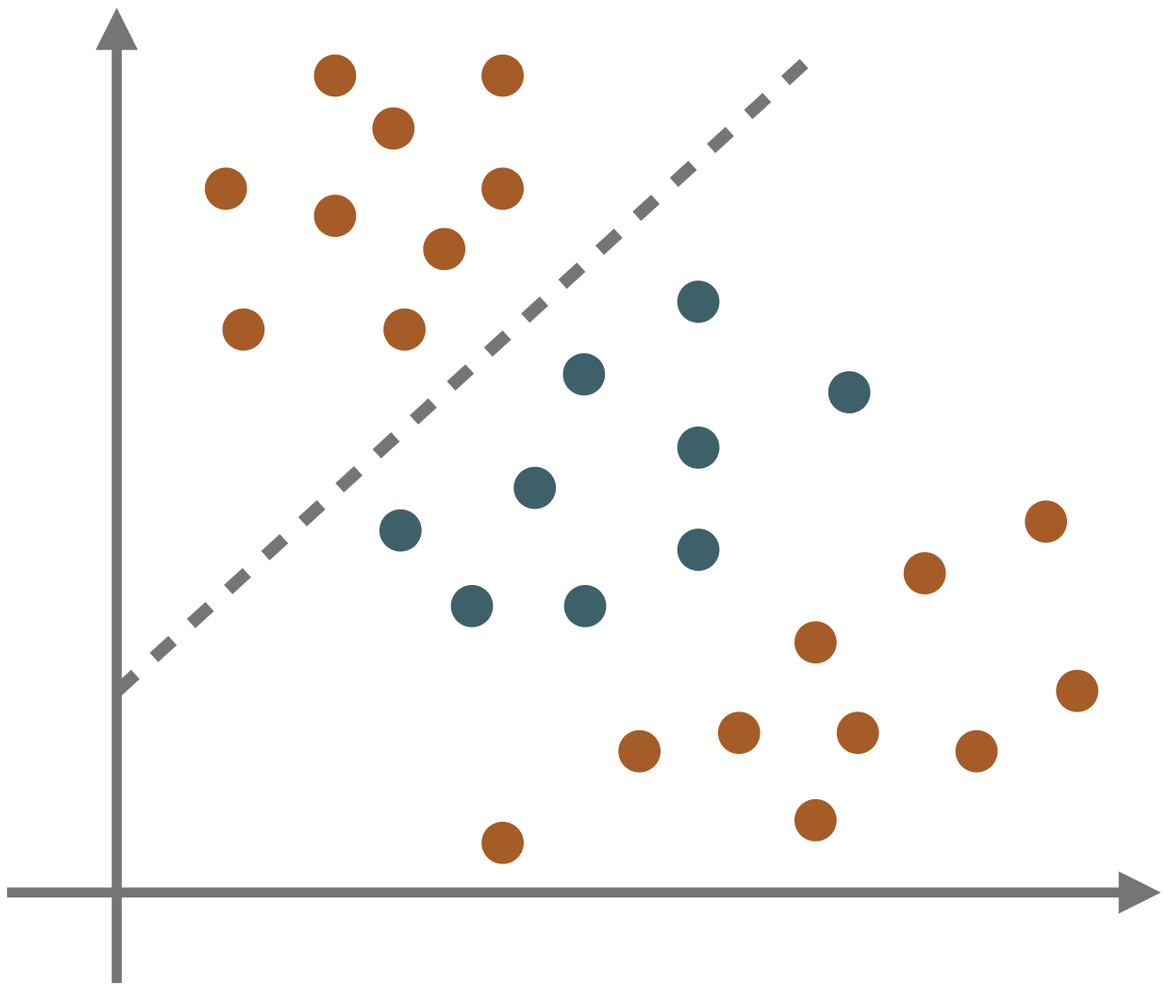
PROBLEMATIC CASES

- Can we always find a hyperplane that separate classes? **NO**
- Can we characterize formally in which cases we can? **YES**



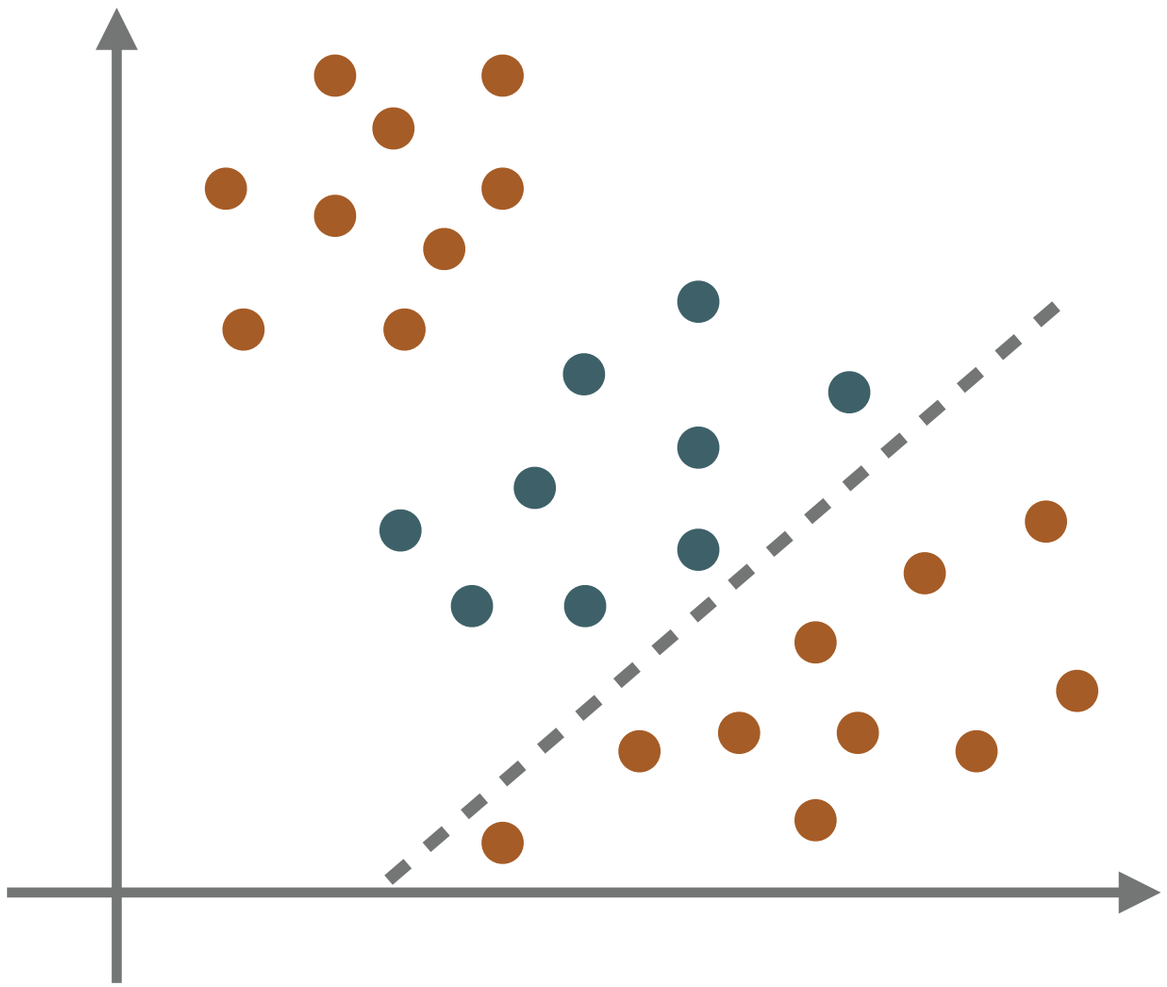
PROBLEMATIC CASES

- Can we always find a hyperplane that separate classes? **NO**
- Can we characterize formally in which cases we can? **YES**



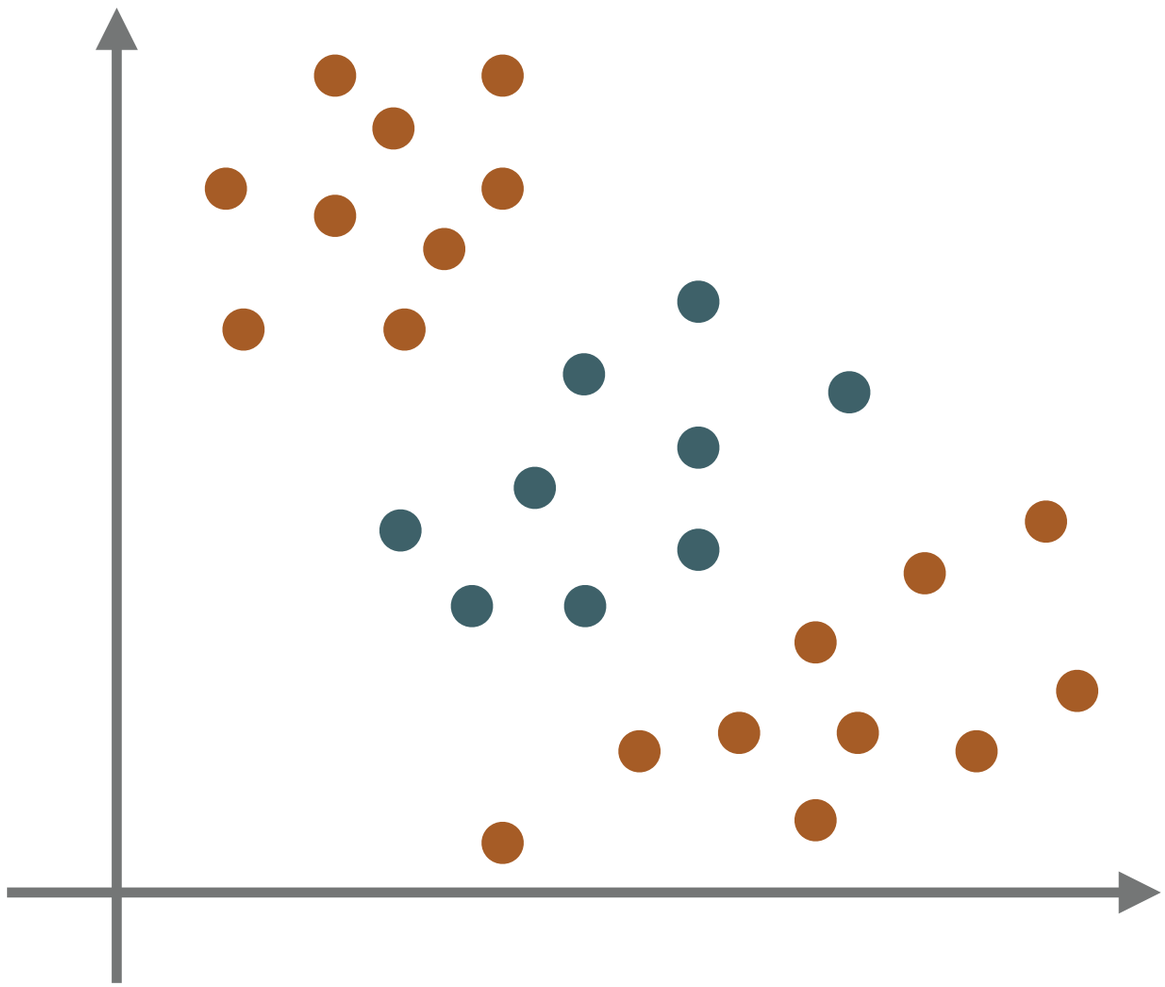
PROBLEMATIC CASES

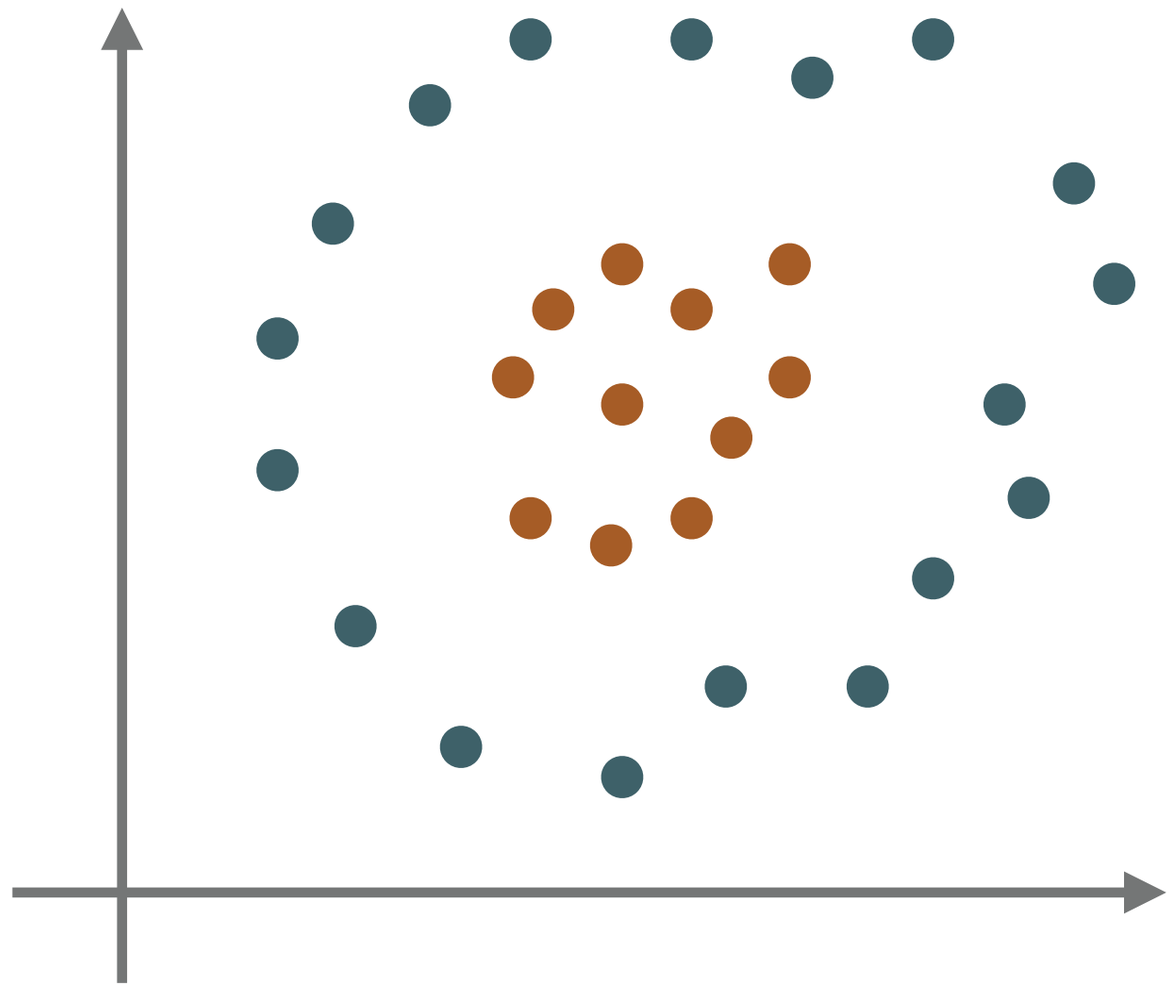
- Can we always find a hyperplane that separate classes? **NO**
- Can we characterize formally in which cases we can? **YES**



PROBLEMATIC CASES

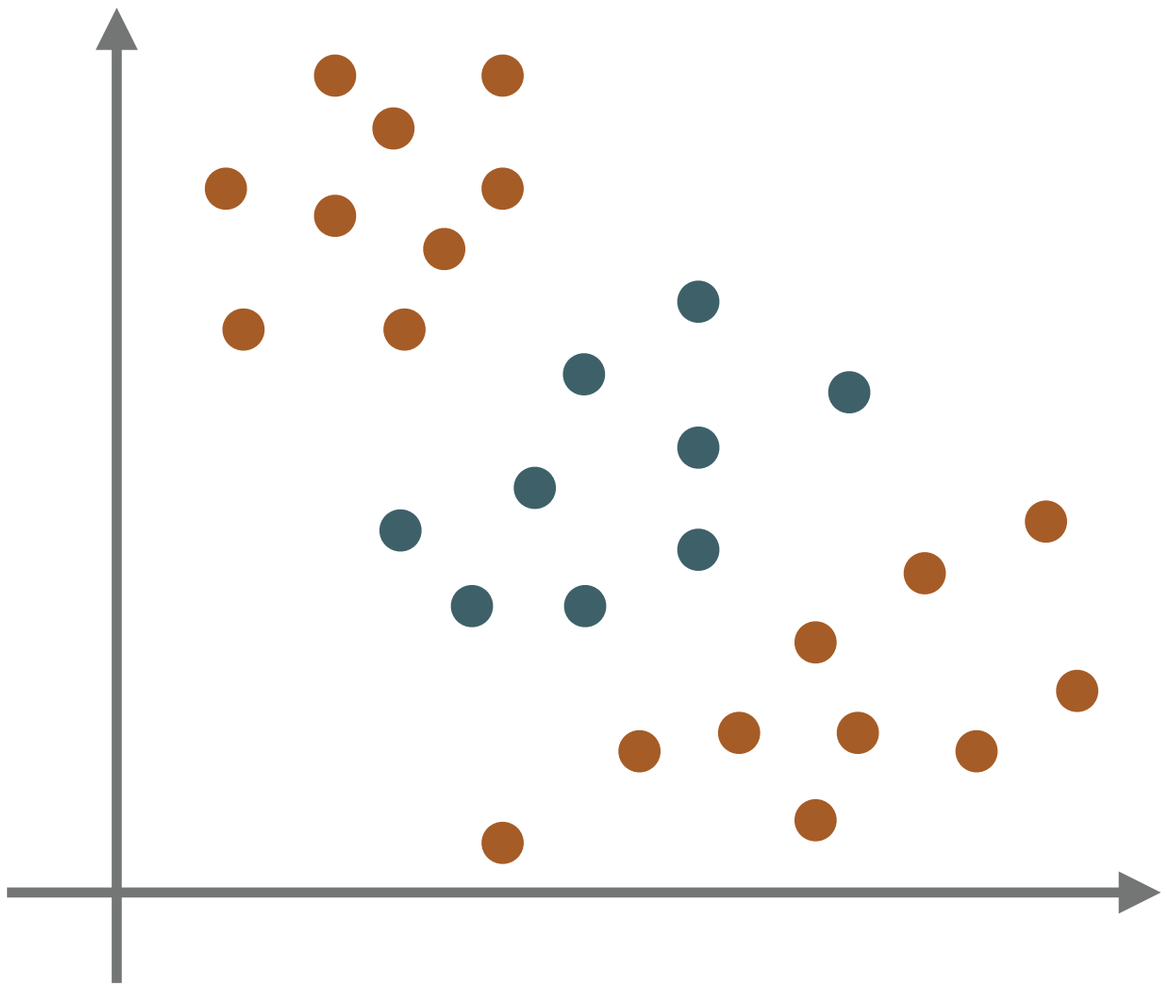
- Can we always find a hyperplane that separate classes? **NO**
- Can we characterize formally in which cases we can? **YES**

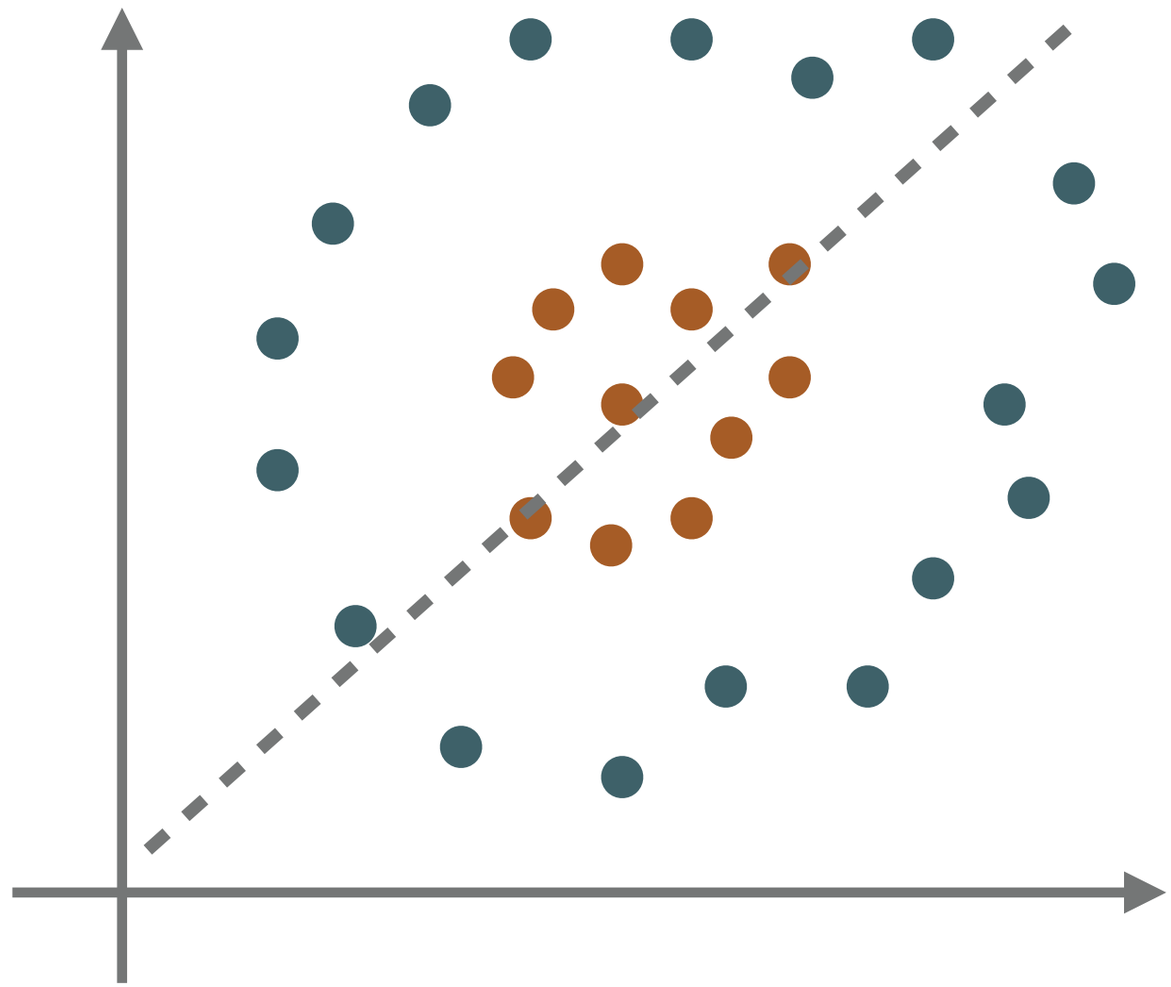




PROBLEMATIC CASES

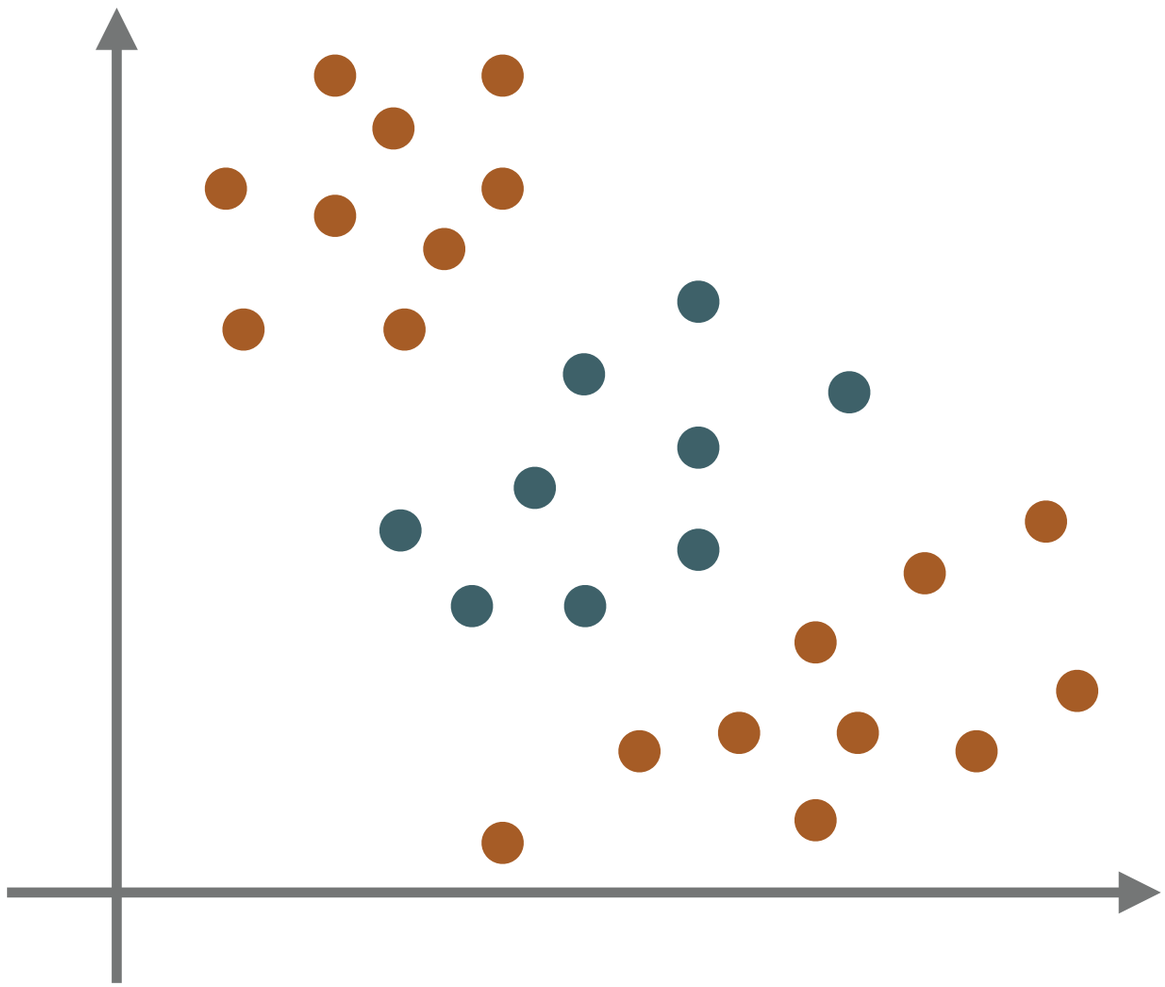
- Can we always find a hyperplane that separate classes? **NO**
- Can we characterize formally in which cases we can? **YES**

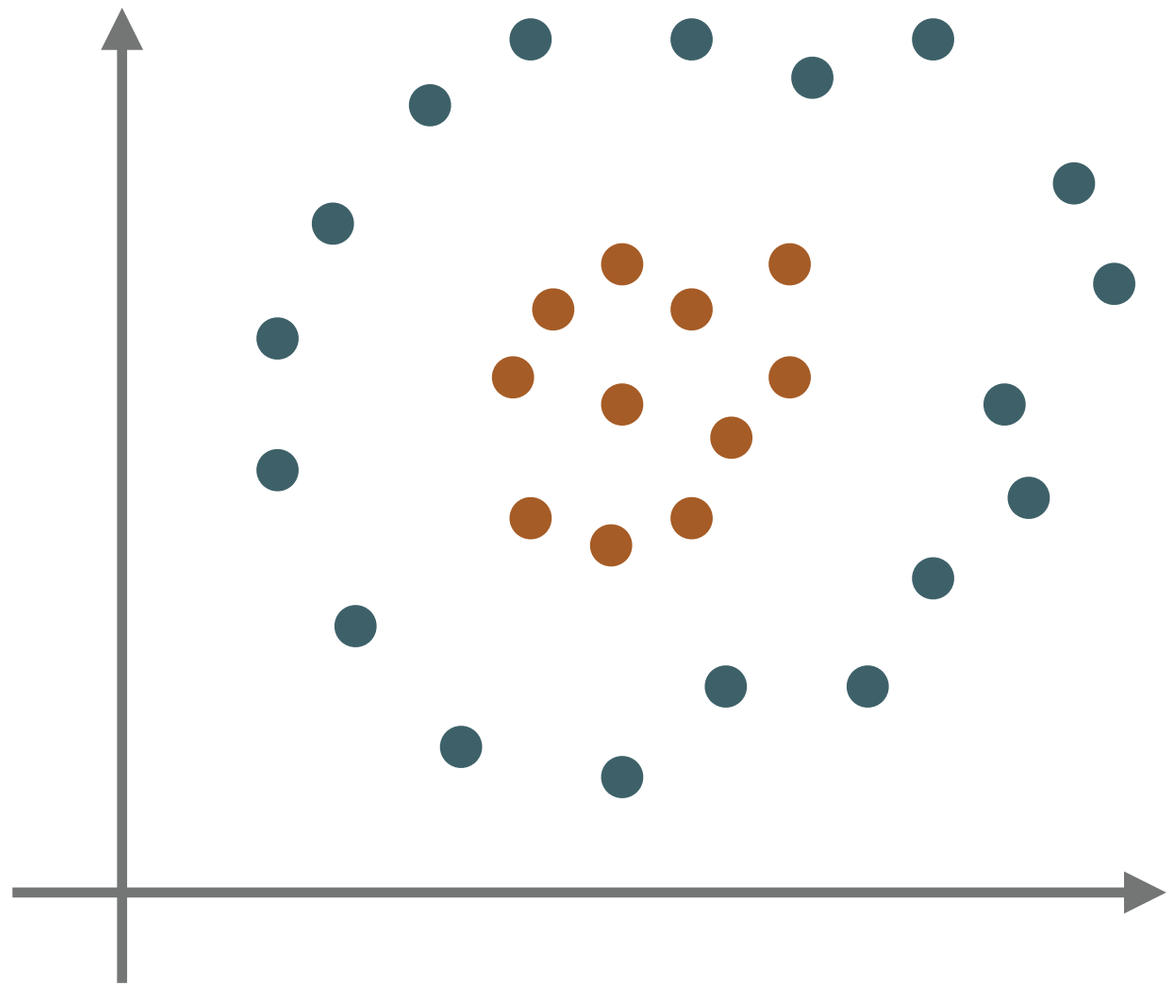




PROBLEMATIC CASES

- Can we always find a hyperplane that separate classes? **NO**
- Can we characterize formally in which cases we can? **YES**





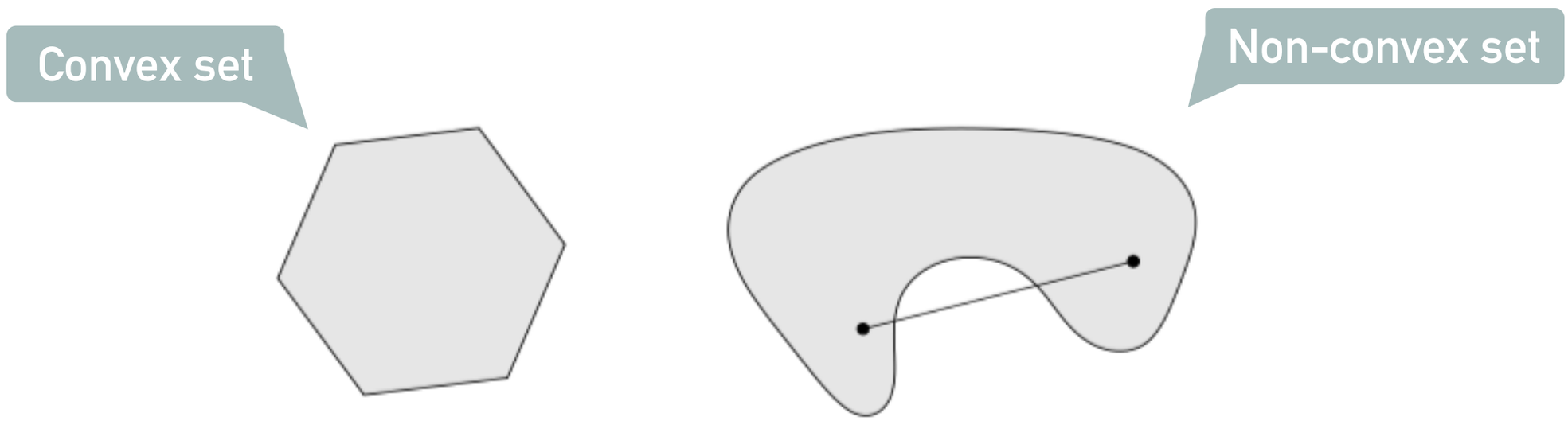
CONVEX SET

Definition

Let $C \in \mathbb{R}^n$ be a set of points. C is convex if and only if:

$$\forall x, y \in C, \epsilon \in [0,1] : \epsilon \times x + (1 - \epsilon) \times y \in C$$

Or, in other words, for every couple of points in C, their convex combination must also be in C.



(Picture from Convex Optimization, Boyd and Vandenberghe)

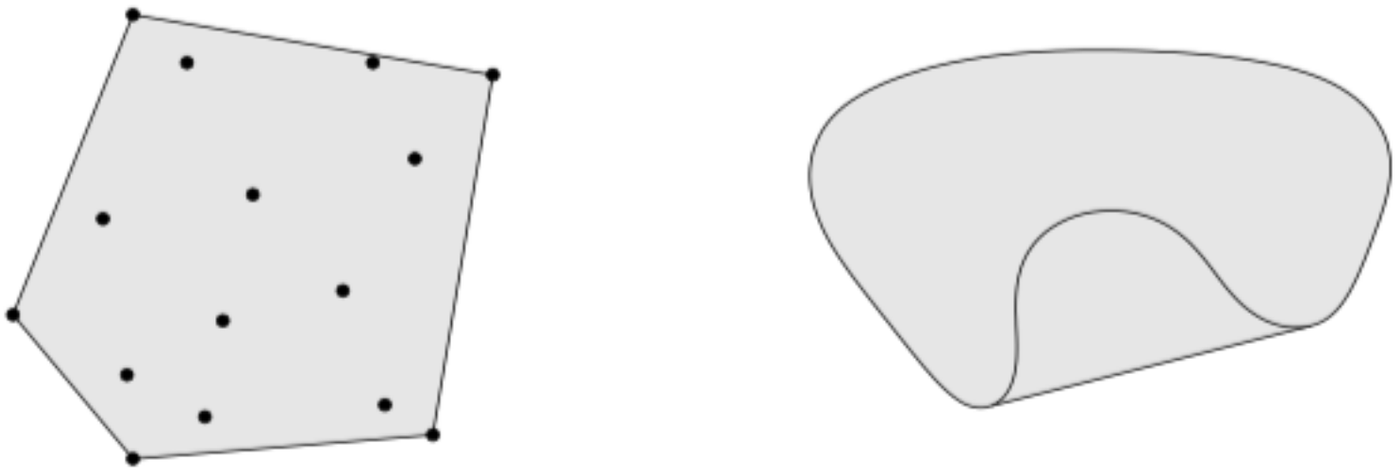
CONVEX HULL

Definition

The **convex hull** of a set$C \in \mathbb{R}^n$ is the set of all convex combinations of points in C:

$$\text{conv } C = \{\epsilon_1 x_1 + \ldots + \epsilon_k x_k \quad | \quad \forall i = 1...k : x_i \in C, \epsilon_i \geq 0, \epsilon_1 + \ldots + \epsilon_k = 1\}$$

Or, in other words, it is the smallest convex set that contains S



(Picture from Convex Optimization, Boyd and Vandenberghe)

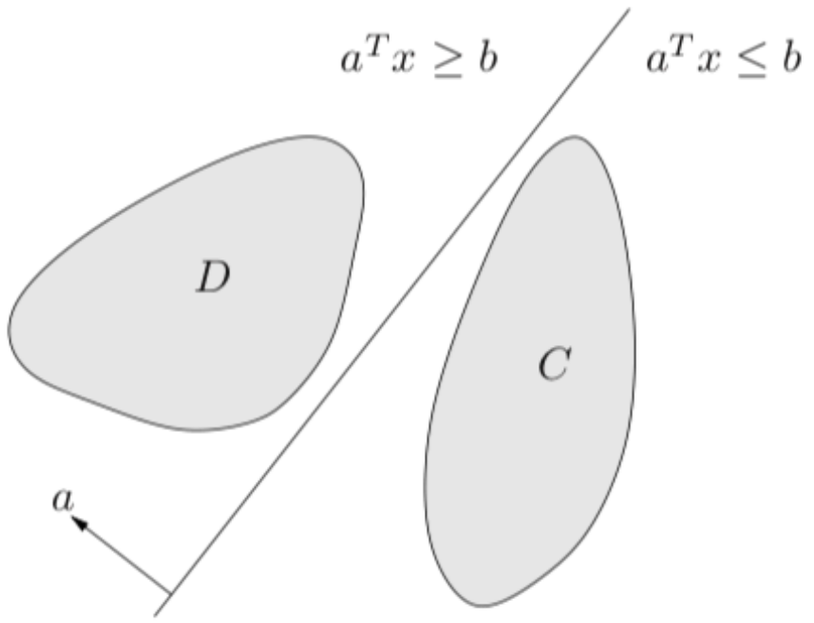
SEPARATING HYPERPLANE

Theorem

Let $C \in \mathbb{R}^n$ and $D \in \mathbb{R}^n$ be two convex sets.
If C and D does not intersect, i.e. $C \cap D = \emptyset$ then there exist a separating hyperplane such that:

$$\forall x \in C: \quad a^\top x + b \geq 0$$
$$\forall x \in D: \quad a^\top x + b \leq 0$$

where w and b parameterize the separation hyperplane.

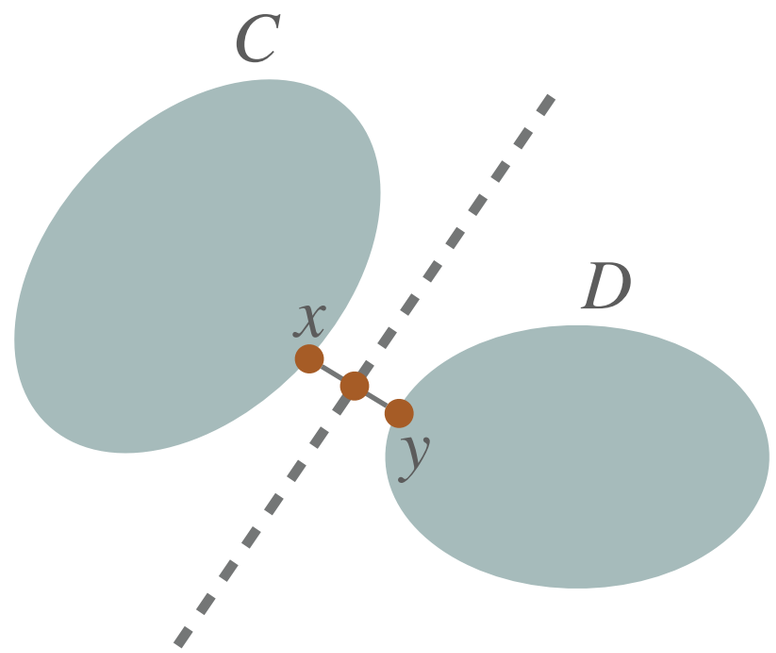


(Picture from Convex Optimization, Boyd and Vandenberghe)

PARAMETER OF THE SEPARATING HYPERPLANE

Closed form solution

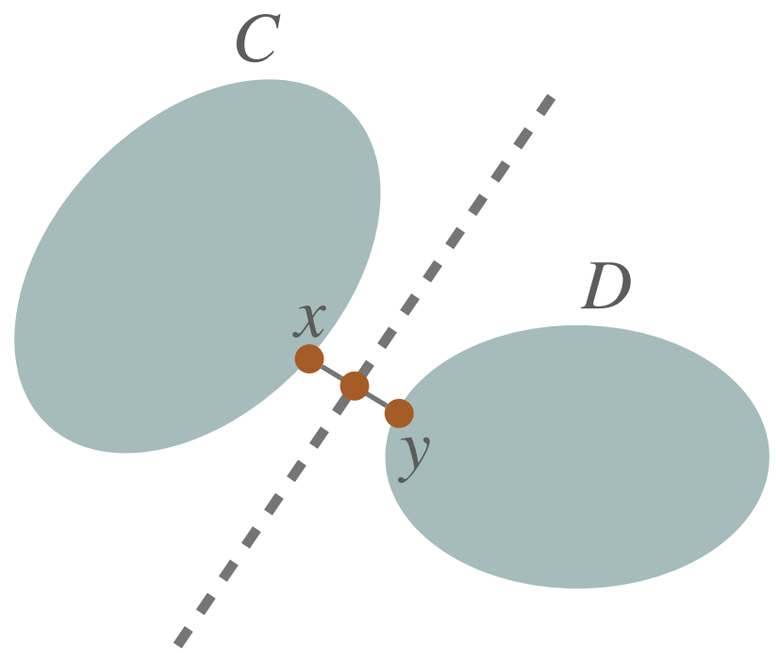
See Convex Optimization (Boyd and Vandenberghe) section 2.5.1.



PARAMETER OF THE SEPARATING HYPERPLANE

Closed form solution

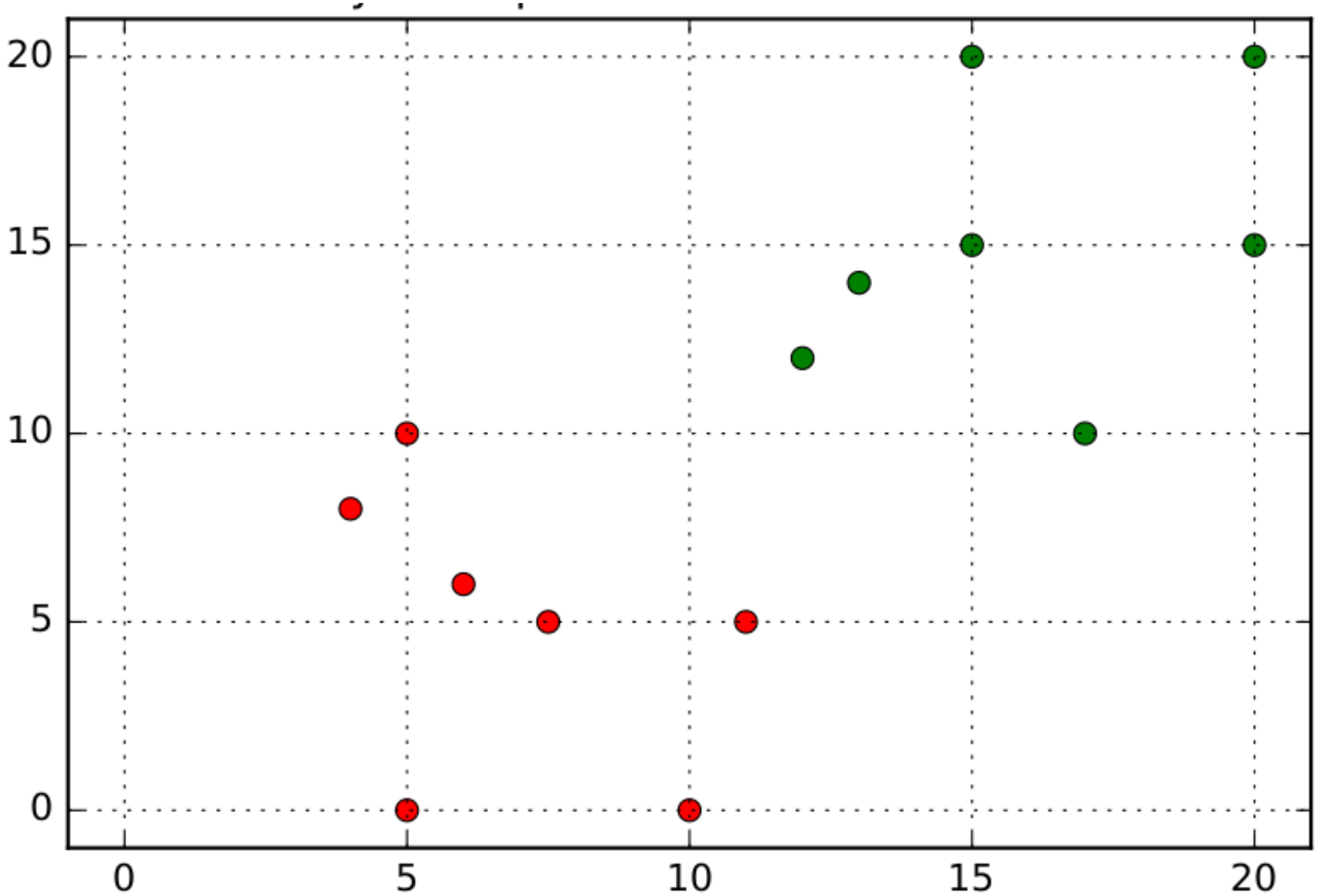
See Convex Optimization (Boyd and Vandenberghe) section 2.5.1.



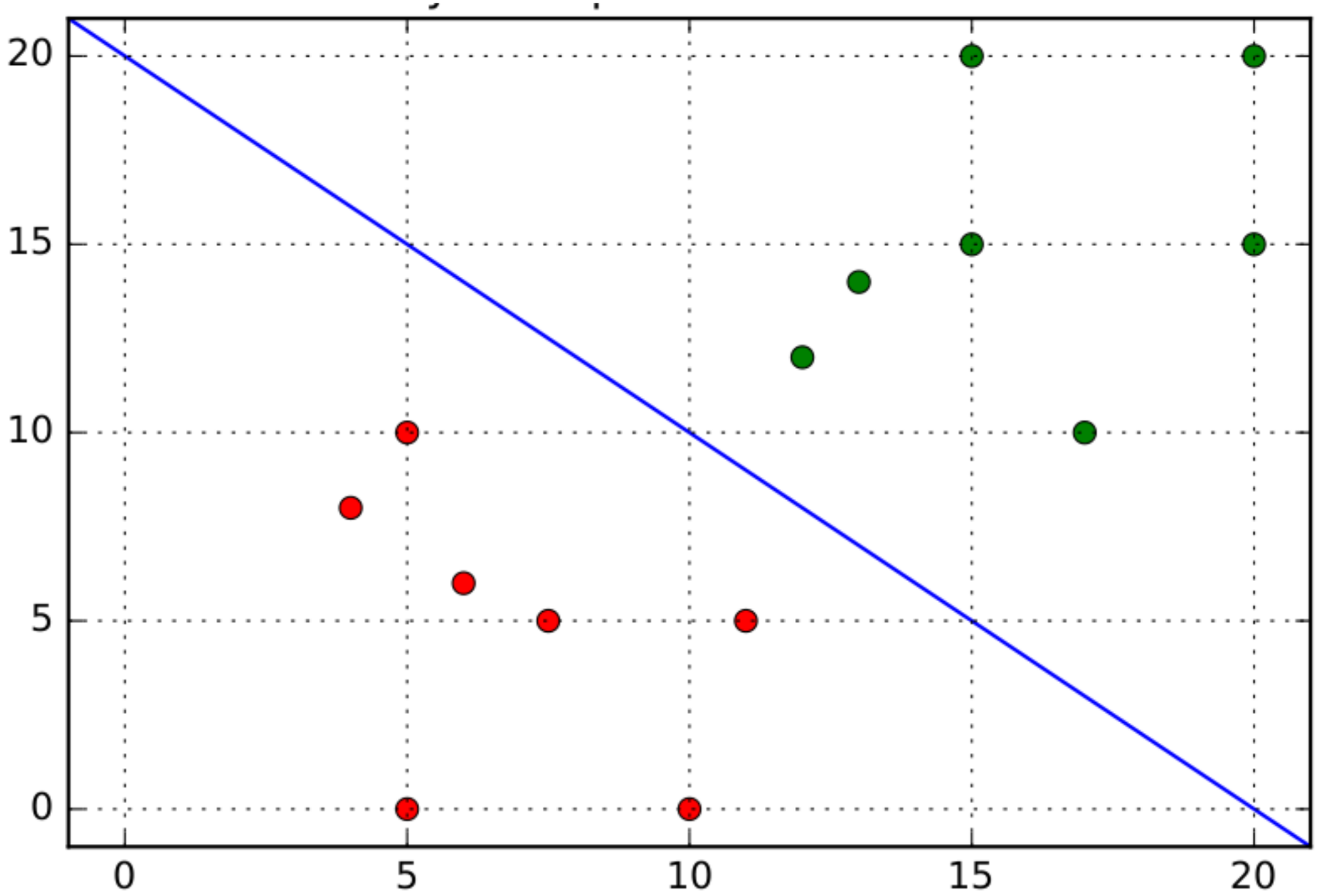
In practice

- Data is not linearly separable (i.e. such a hyperplane does not exists)
- Computing global solutions can be very expensive with big datasets
- Online algorithm are preferable

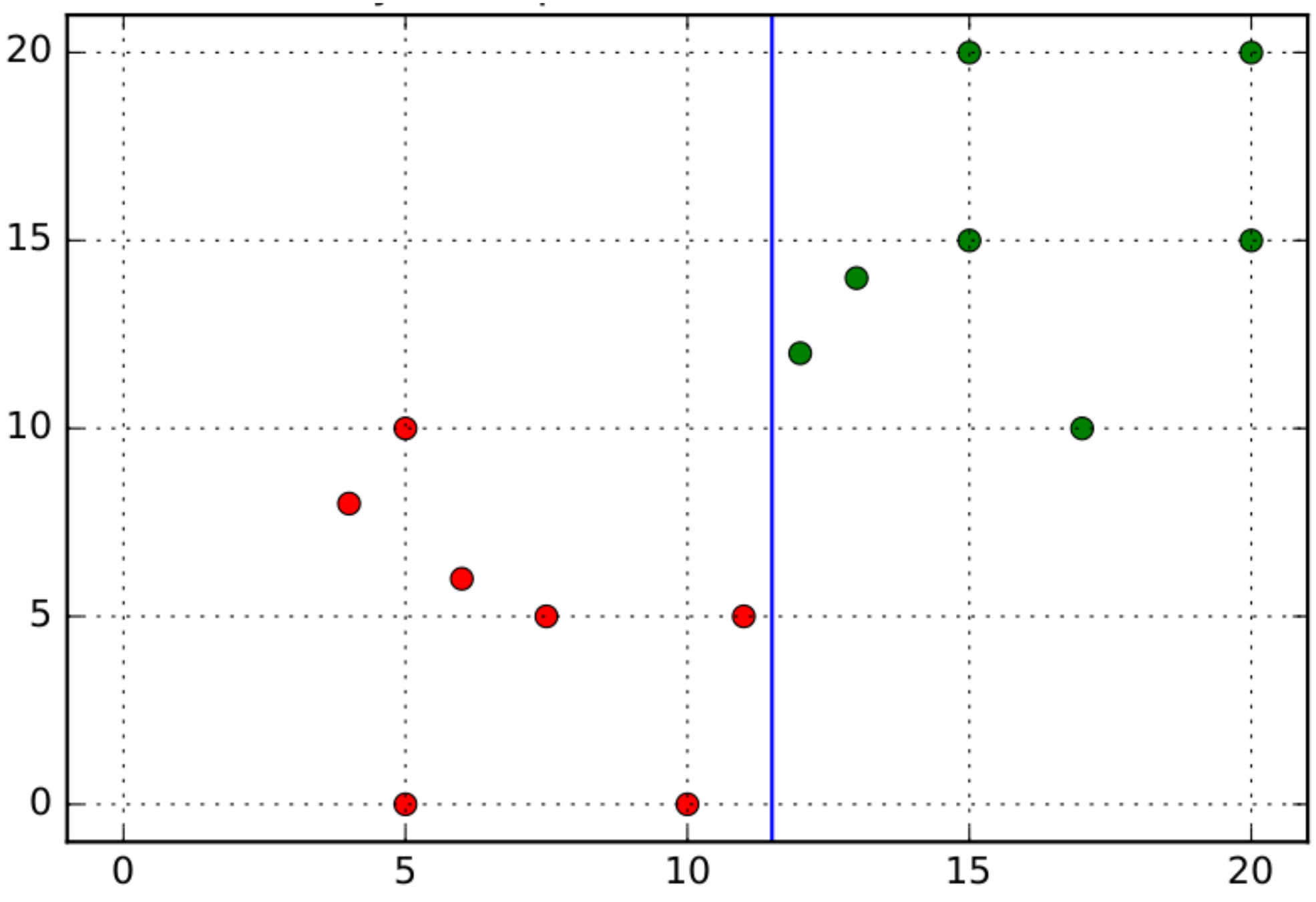
HOW TO SEPARATE THE DATA?



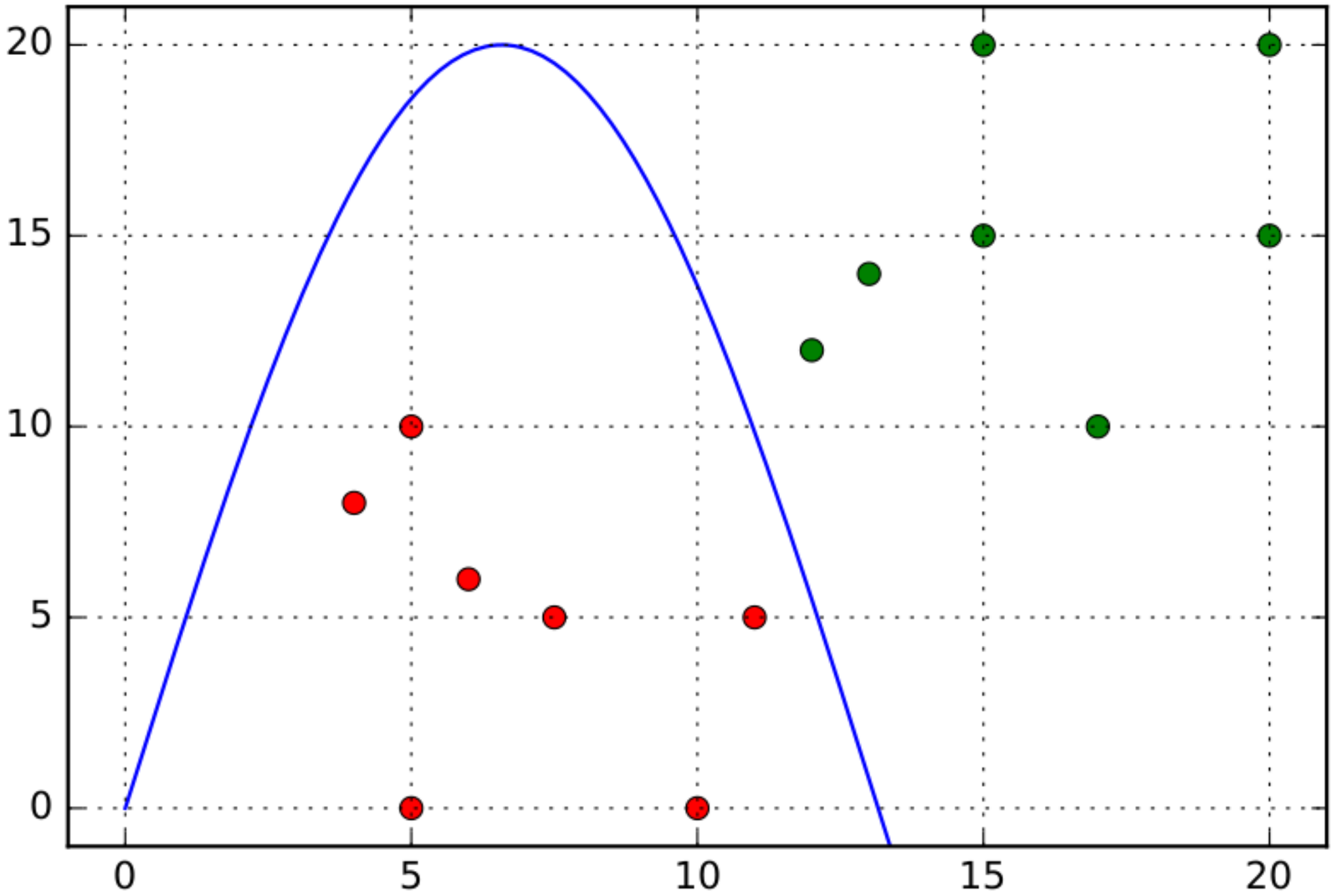
HOW TO SEPARATE THE DATA?



HOW TO SEPARATE THE DATA?



HOW TO SEPARATE THE DATA?



MULTI-LAYER PERCEPTRON

MAIN IDEA

Classifier

Parameterized function $f_\theta : \mathcal{X} \to \mathcal{Y}$

Feature space

Score/output space

Parameters

How to deal with non-separable inputs?

- Manually transform the inputs :(
- Learn automatically a transformation?

Intuition behind multi-layer perceptrons

- Compute « latent » hidden representations so that classes are linearly separable
- Use non-linear activation units so the transformation is not convex

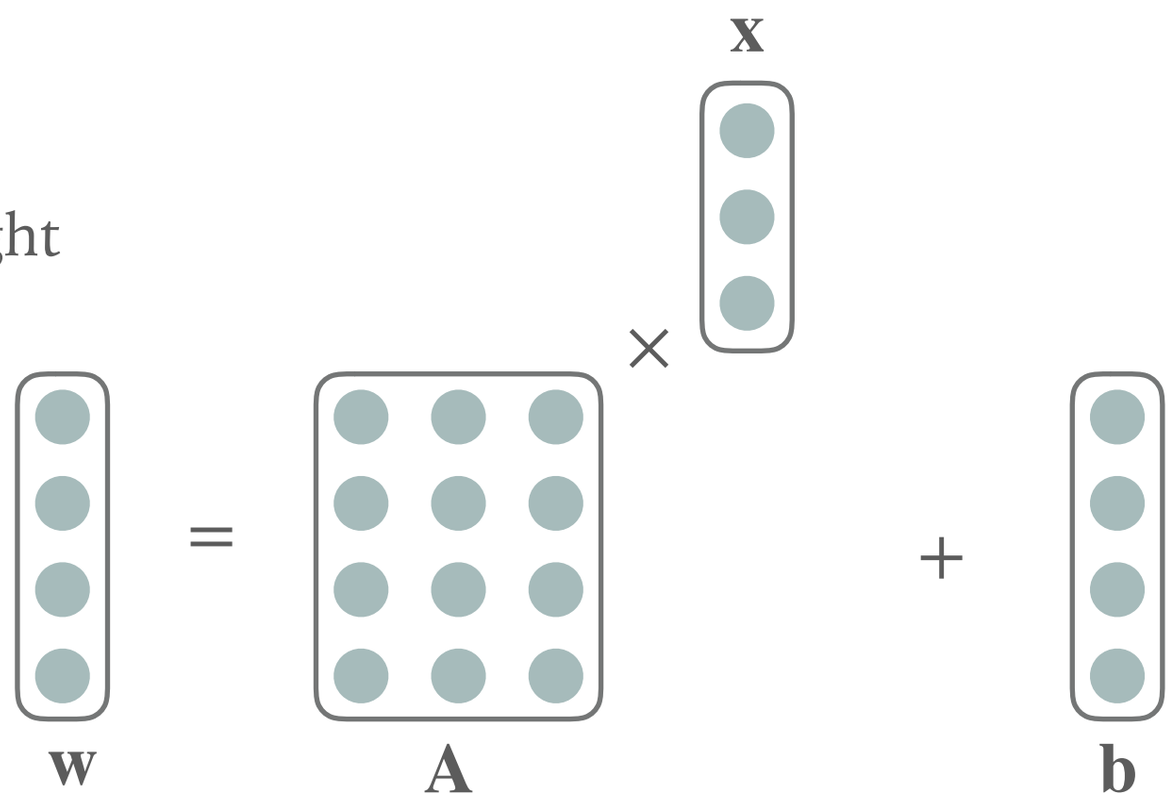
LINEAR CLASSIFIER FOR MULTI-CLASS CLASSIFICATION

Problem

- Input: features
- Output: 1-in-k prediction

Linear classifier $\mathbf{w} = \mathbf{A}\mathbf{x} + \mathbf{b}$

- Input dim: 3
- Output dim: k=4 classes
- Prediction: class with maximum weight



MULTILAYER PERCEPTRON 1/2

$$\mathbf{z}^{(1)} = \sigma\left(\mathbf{A}^{(1)}\mathbf{x} + \mathbf{b}^{(1)}\right)$$

First hidden layer

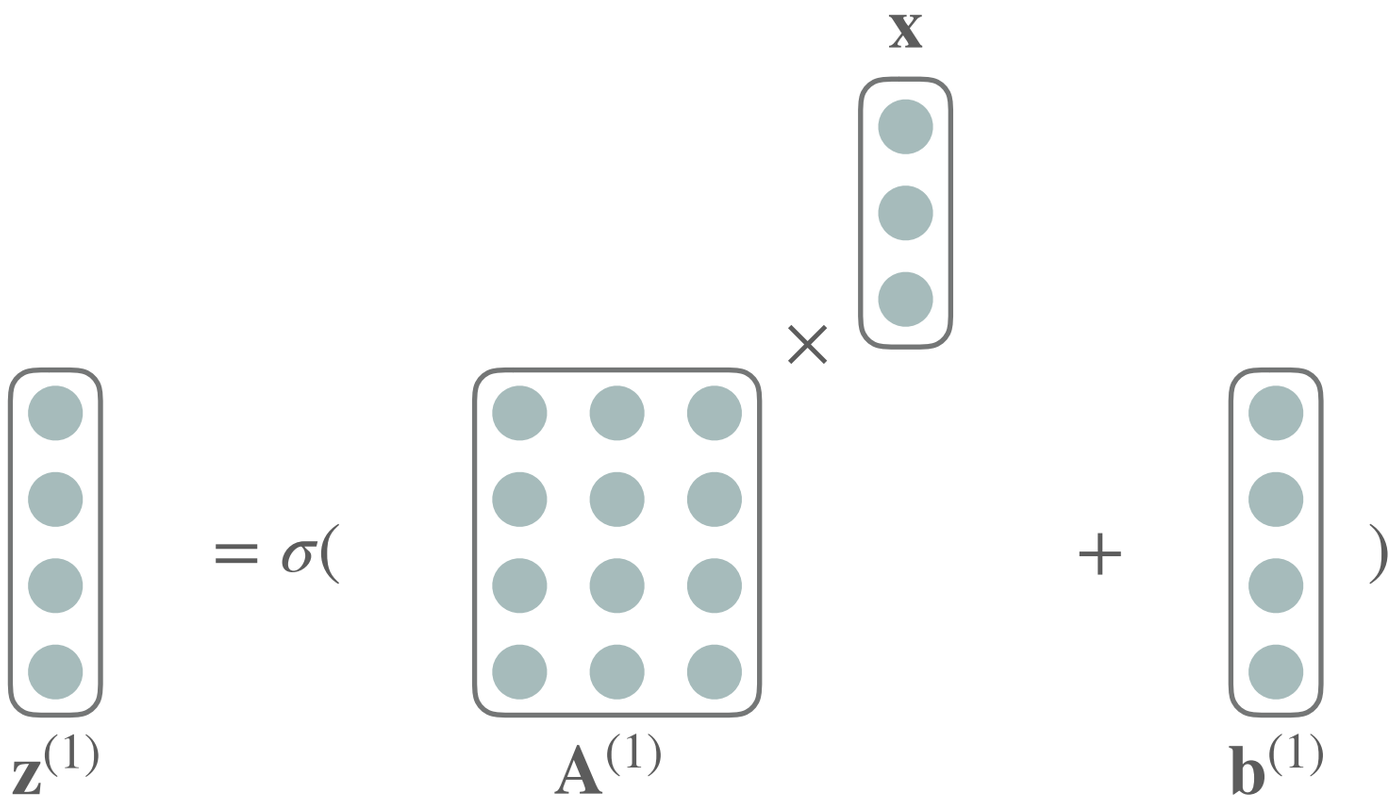
$$\mathbf{z}^{(2)} = \sigma\left(\mathbf{A}^{(2)}\mathbf{z}^{(1)} + \mathbf{b}^{(2)}\right)$$

Second hidden layer

$$\mathbf{w} = \mathbf{A}^{(3)}\mathbf{z}^{(2)} + \mathbf{b}^{(3)}$$

Output projection

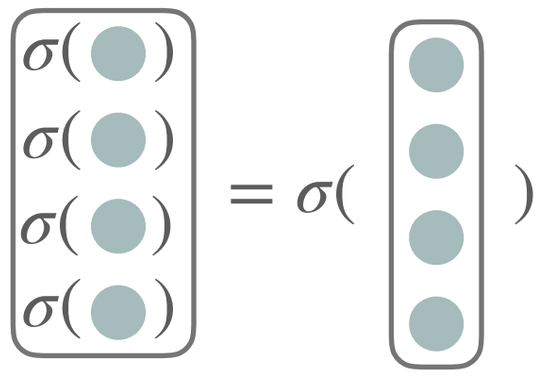
- $\mathbf{x}$: input features
- $\mathbf{z}^{(i)}$: hidden representations
- $\mathbf{w}$: output logits
- $\theta = \{\mathbf{A}^{(1)}, \mathbf{b}^{(1)}, \dots\}$: trainable parameters
- σ : piecewise non-linear activation function



NON-LINEAR ACTIVATION FUNCTIONS 1/2

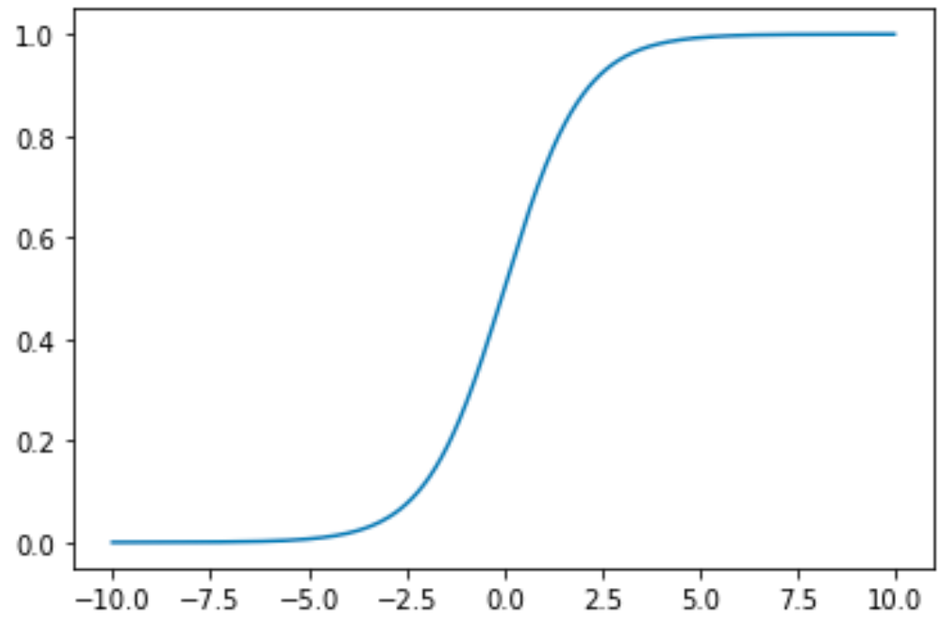
Main idea

- Apply a non-linear transformation
- Piecewise (so its fast to compute)
- There are many possibilities
  (I'll just present 3 of them)



Sigmoid

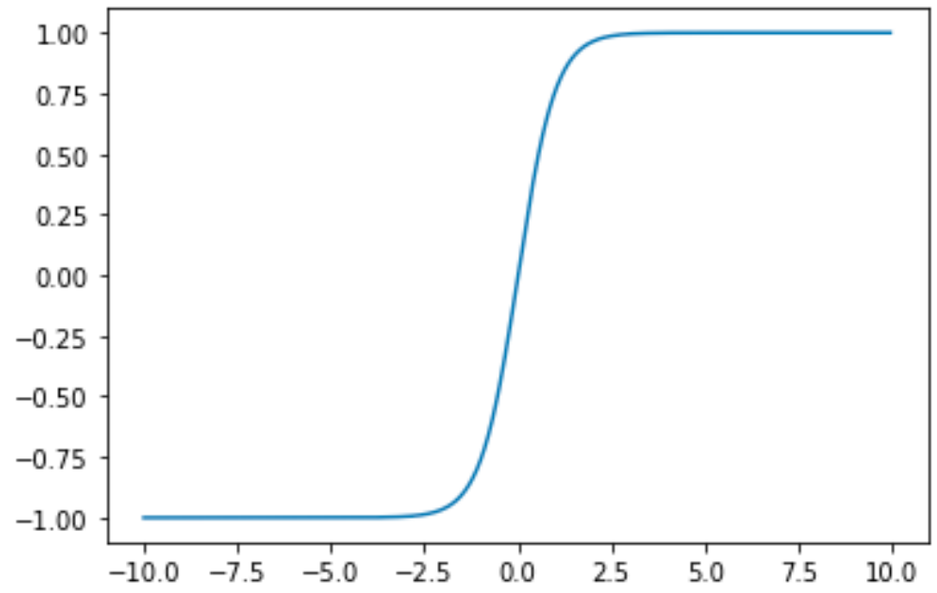
$$\sigma(u) = \frac{\exp(u)}{1 + \exp(u)} = \frac{1}{1 + \exp(-u)}$$



NON-LINEAR ACTIVATION FUNCTIONS 2/2

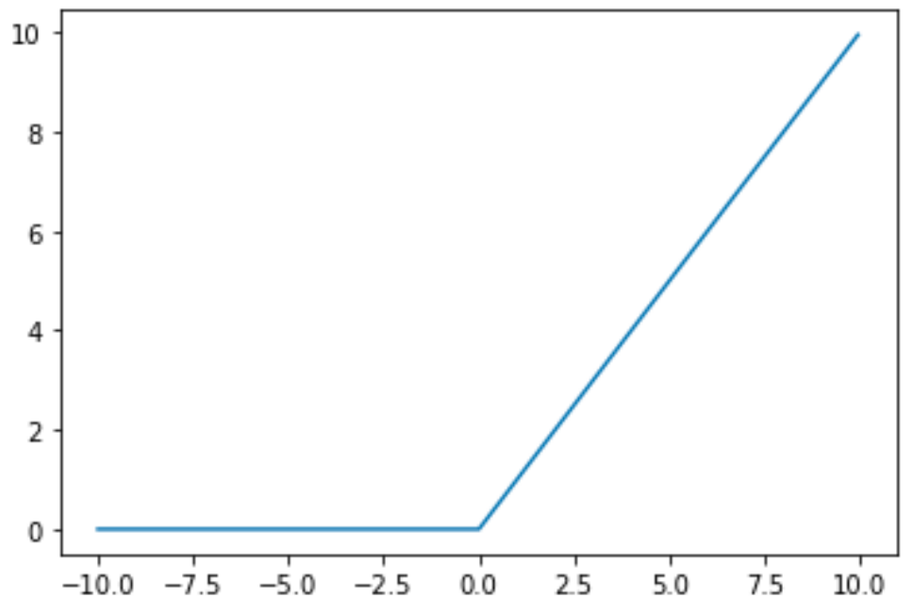
Hyperbolic tangent (tanh)

$$\tanh(u) = \frac{\exp(2u) - 1}{\exp(2u) + 1}$$



Rectified Linear Unit (relu)

$$\text{relu}(u) = \max(0,u)$$



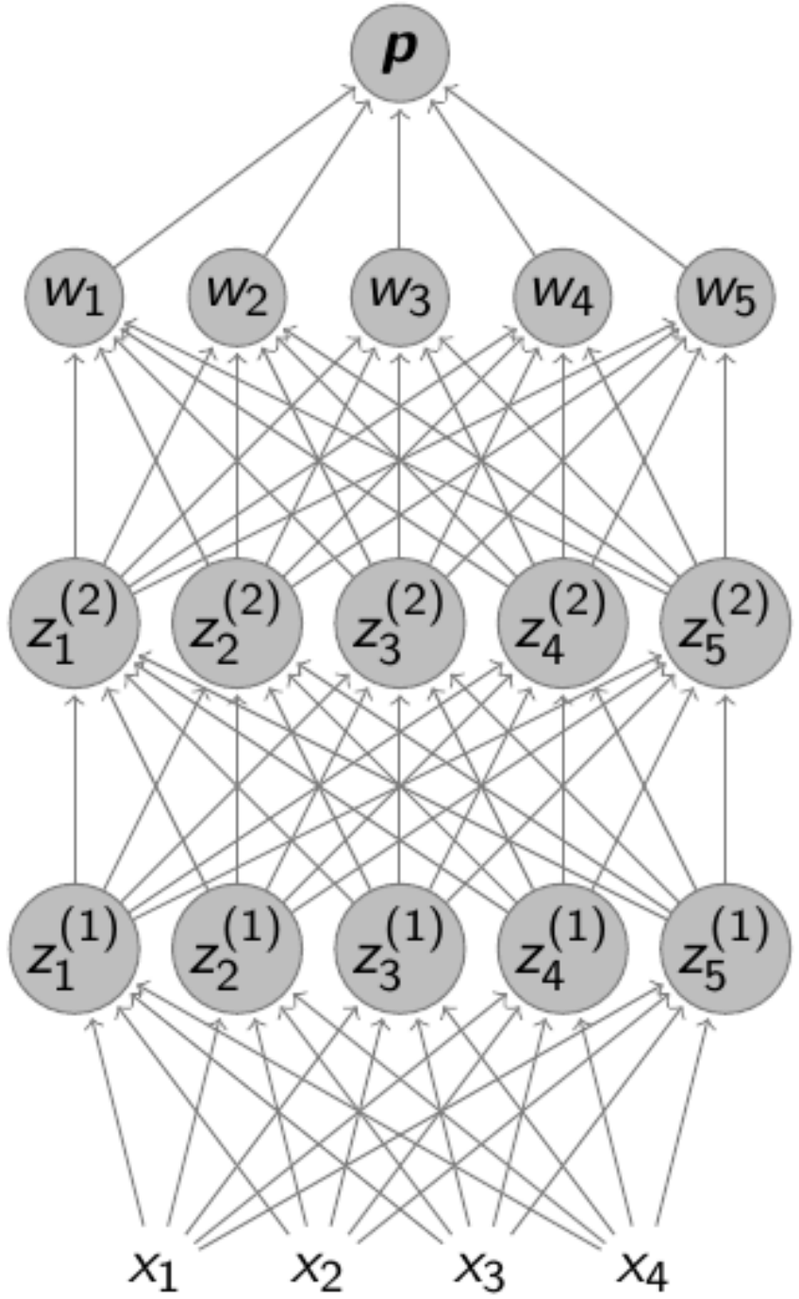
- $\boldsymbol{x}$: input features
- $\boldsymbol{z}^{(1)}, \boldsymbol{z}^{(2)}$: hidden representation
- $\boldsymbol{w}$: output logits or class weights
- $\boldsymbol{p}$: probability distribution over classes
- $\theta = \{\boldsymbol{A}^{(1)}, \boldsymbol{b}^{(1)}, ...\}$: parameters
- σ: non-linear activation function

$$\boldsymbol{z}^{(1)} = \sigma\left(\boldsymbol{A}^{(1)}\boldsymbol{x} + \boldsymbol{b}^{(1)}\right)$$

$$\boldsymbol{z}^{(2)} = \sigma\left(\boldsymbol{A}^{(2)}\boldsymbol{z}^{(1)} + \boldsymbol{b}^{(2)}\right)$$

$$\boldsymbol{w} = \sigma\left(\boldsymbol{A}^{(3)}\boldsymbol{z}^{(2)} + \boldsymbol{b}^{(3)}\right)$$

$$\boldsymbol{p} = \text{Softmax}(\boldsymbol{w}) \quad \text{i.e.} \quad p_i = \frac{\exp(w_i)}{\sum_j \exp(w_j)}$$



Graphical or mathematical representation?

- Use a graphical representation only if required
- Alway prefer the mathematical description!

Code example!

PREDICTION FUNCTION

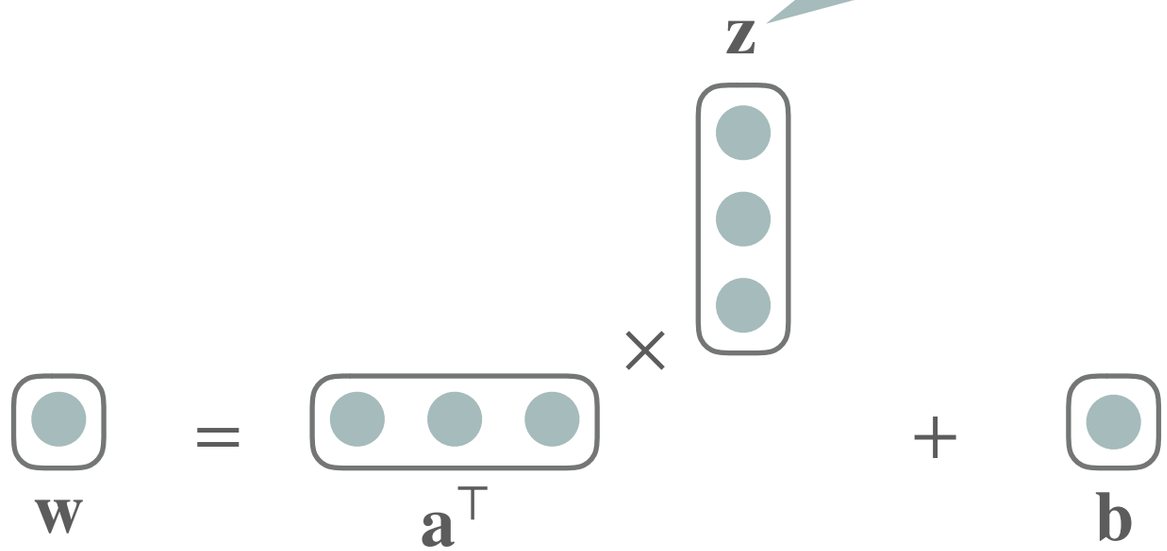
Vocabulary issue

The term "prediction function" can refer to both the "full model" or only the function that transforms the class weights/logits/scores to an actual output. :(

DO NOT CONFUSE

- The (non-linear) activation function (inside the neural network)
- The function that transforms weights/logits/scores into an output (at the output of the neural network)

BINARY CLASSIFICATION



Prediction functions

- 0 / 1 prediction :

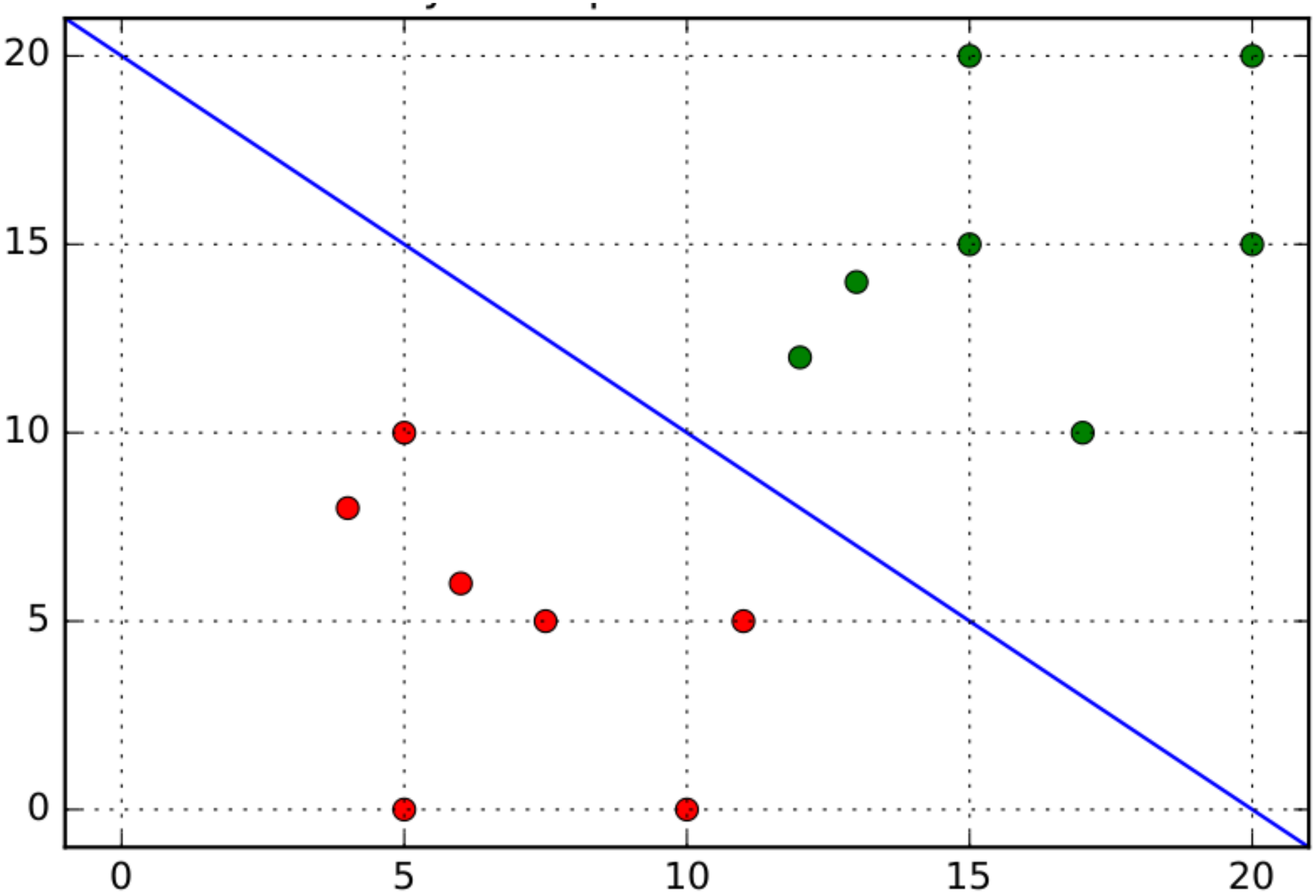
$$\hat{\mathbf{y}}(w) = \begin{cases} 1 & if\ w \geq 0 \\ 0 & otherwise \end{cases}$$

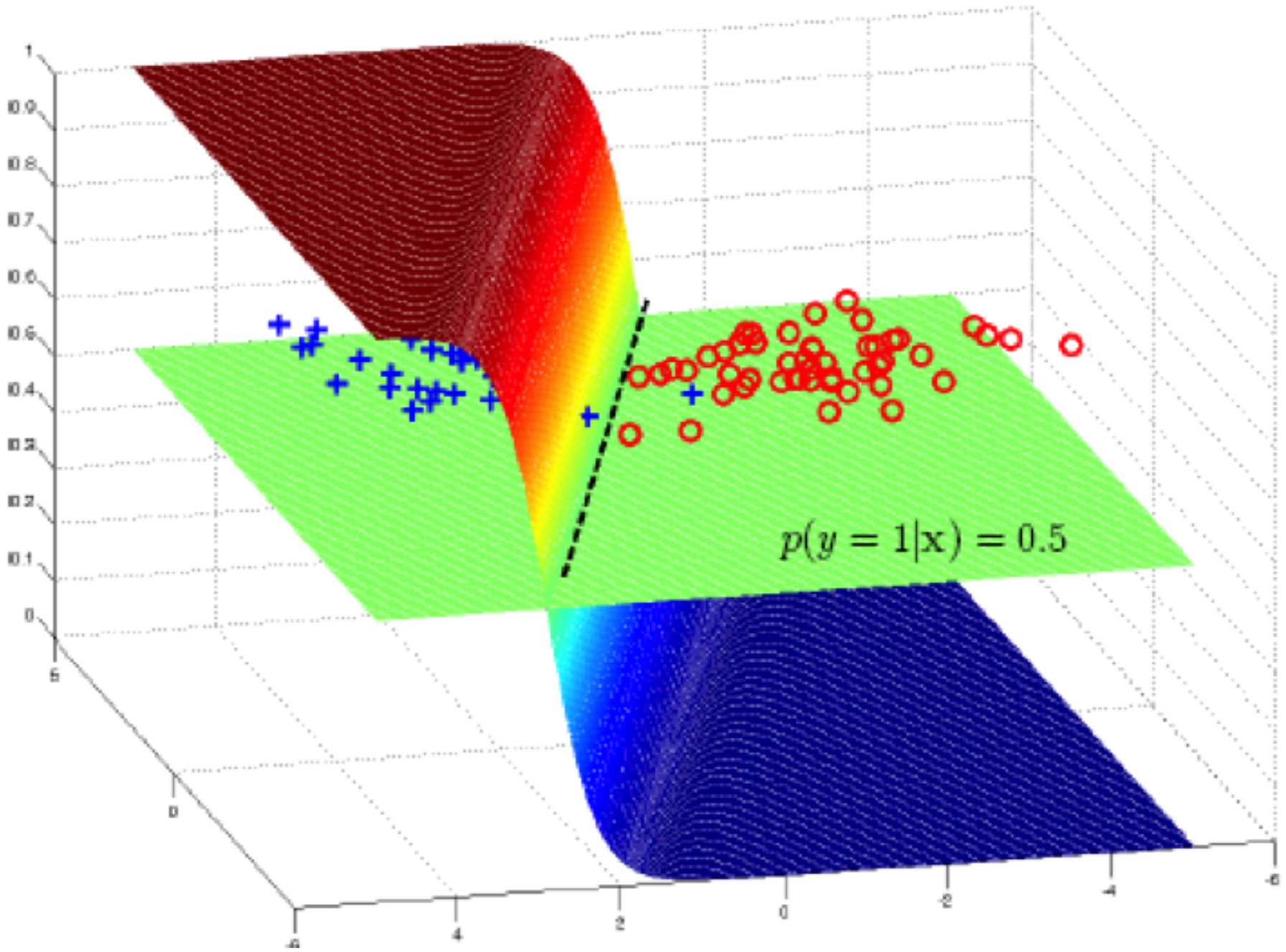
- -1 / 1 prediction

$$\hat{\mathbf{y}}(w) = \begin{cases} 1 & if\ w \geq 0 \\ -1 & otherwise \end{cases}$$

- Probabilistic loss (output is the parameter of a Bernoulli distribution)

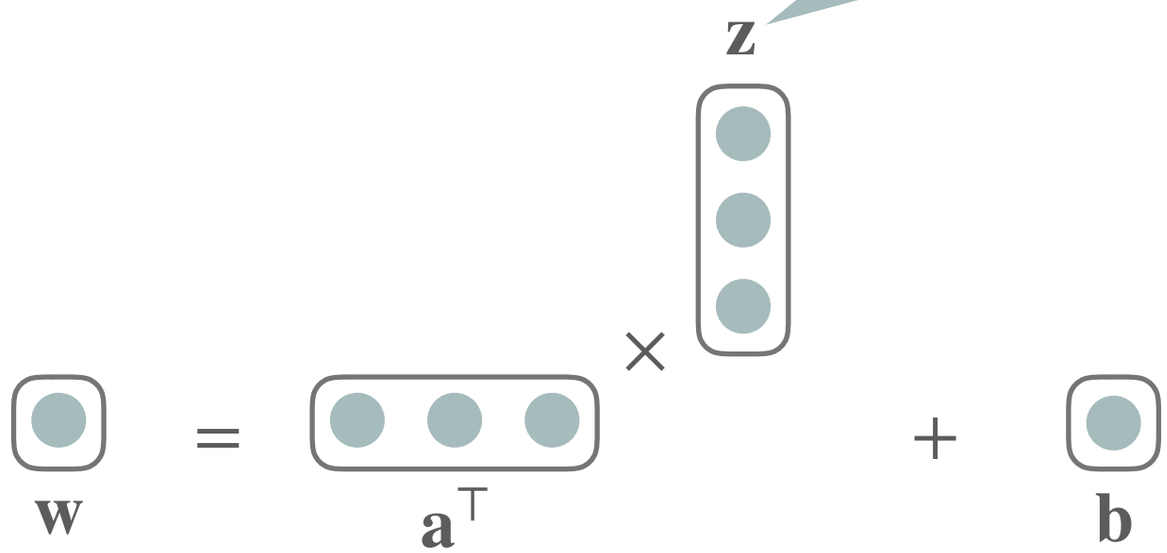
$$\hat{\mathbf{y}}(w) = \sigma(w) = \frac{\exp(w)}{1 + \exp(w)}$$


20
15
10
5
0
0
5
10
15
20


1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
5
0
-5
4
2
0
-2
-4
-6
$p(y = 1|\mathbf{x}) = 0.5$

BINARY CLASSIFICATION

hidden representation computed by the neural network



Loss functions

- Hinge loss (gold is 0/1) : $\ell(y, w) = \max(0,\ 1 - (2y - 1)w)$
- Hinge loss (gold is -1/1) : $\ell(y, w) = \max(0,\ 1 - yw)$
- Negative log-likelihood (or cross-entropy) (gold is 0/1) :

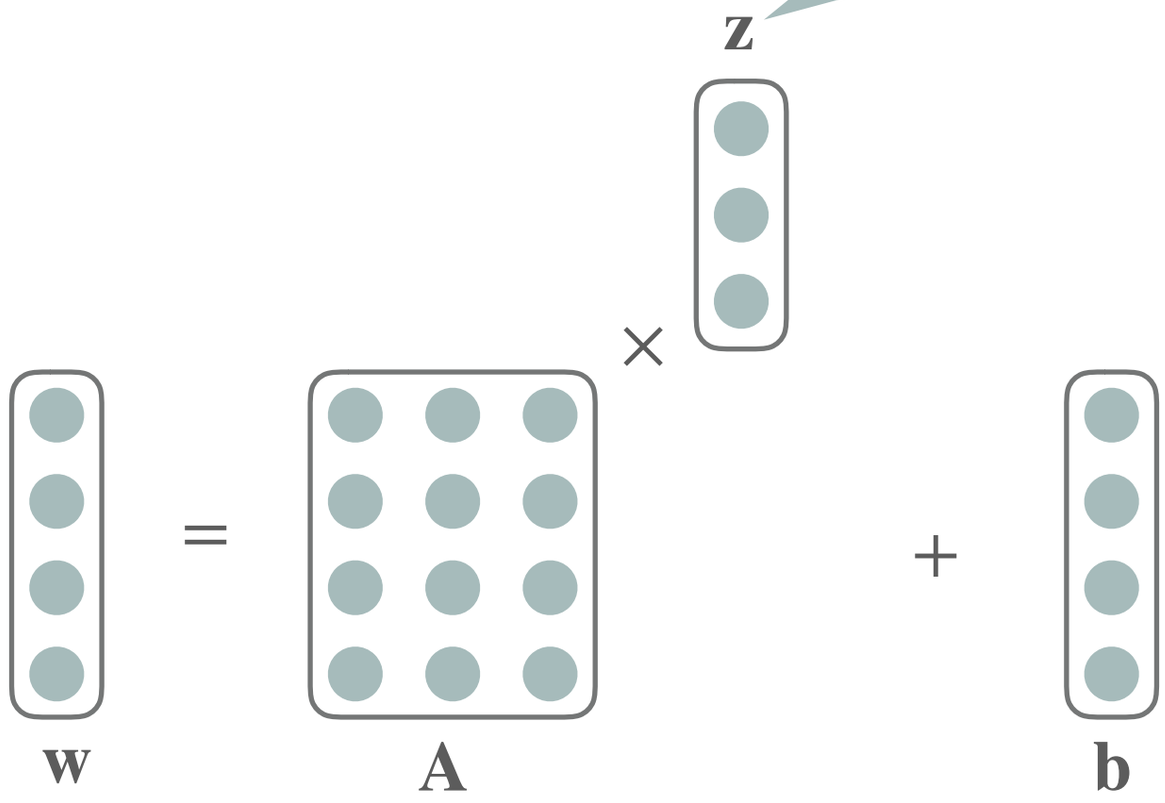
$$\ell(y, w) = -yw + \log(1 + \exp(w))$$

- Negative log-likelihood (or cross-entropy) (gold is -1/1)

$$\ell(y, w) = \log(1 + \exp(-yw))$$

MULTICLASS CLASSIFICATION

hidden representation computed by the neural network



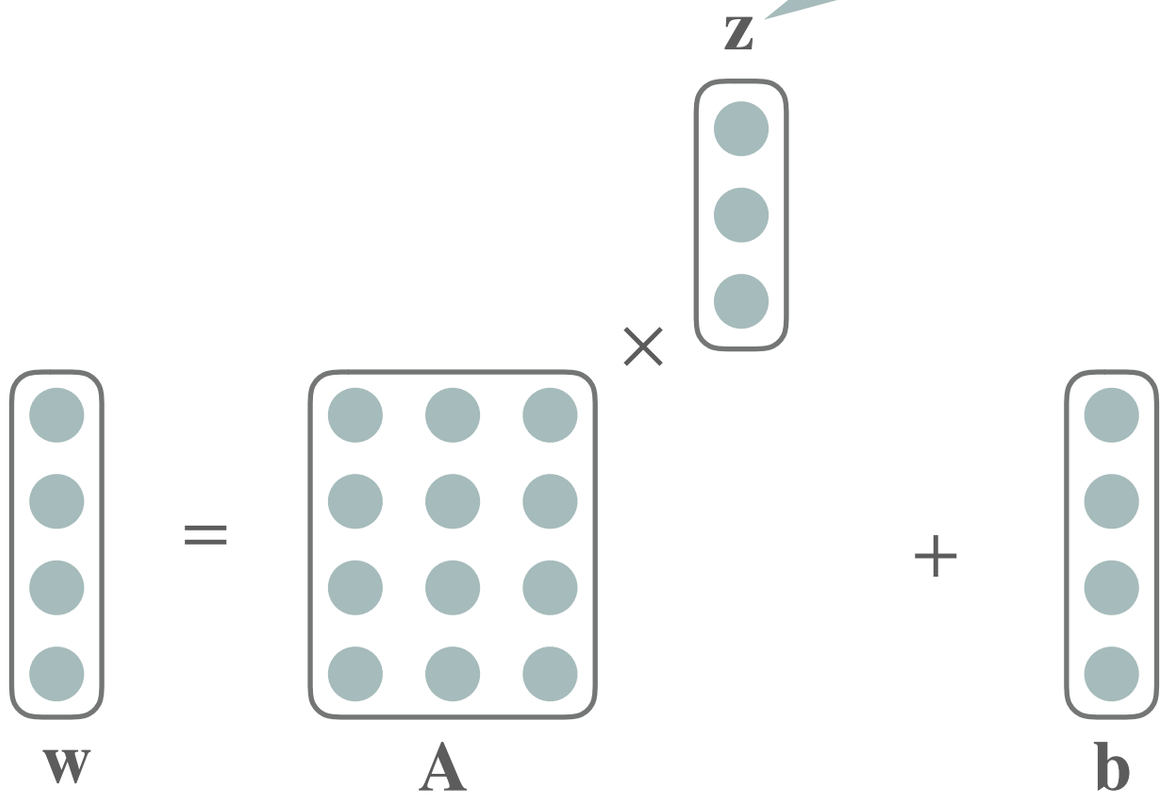
Prediction functions

- Integer output : $\hat{\mathbf{y}}(\mathbf{w}) = \text{argmax}_{i \in \{1,\ldots,k\}} \; w_i$
- One-hot vector output : $\hat{\mathbf{y}}(\mathbf{w}) = \text{argmax}_{\mathbf{y} \in E(k)} \; \langle \; \mathbf{y}, \mathbf{w} \; \rangle$
- Probabilistic output (i.e. distribution over classes) : $\hat{\mathbf{y}}(\mathbf{w}) = \text{softmax}(\mathbf{w})$

E(k) is the set of one-hot vector of dim. k

MULTICLASS CLASSIFICATION

hidden representation computed by the neural network

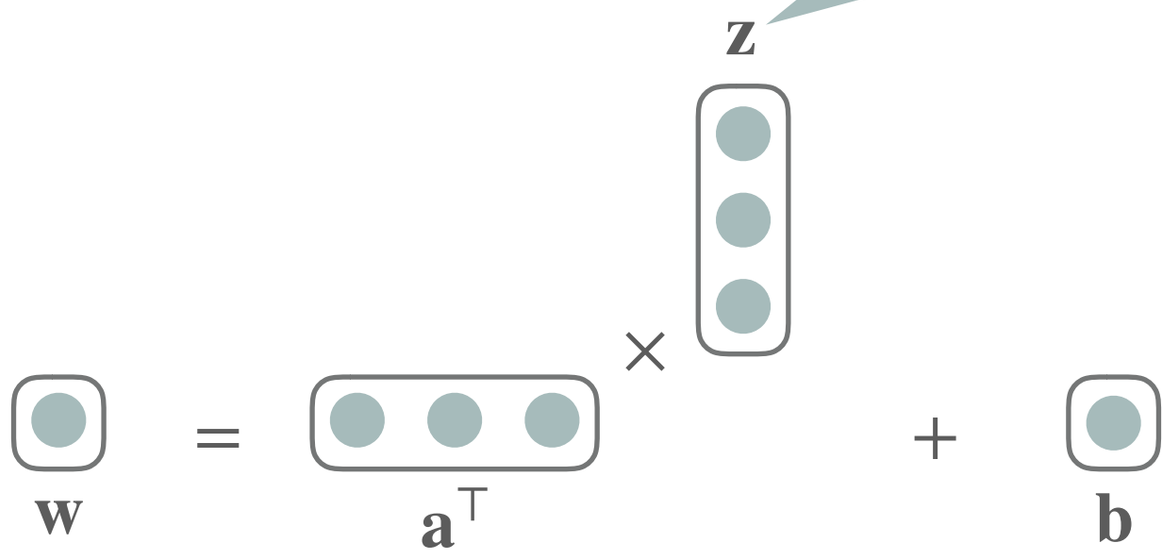


Loss functions

- hinge loss (m >= 0 is the margin) : $\ell(\mathbf{y}, \mathbf{w}) = \max(0,\ m - \langle \mathbf{y}, \mathbf{w} \rangle + \max_{\mathbf{y}' \in E(k) \setminus \{\mathbf{y}\}} \langle \mathbf{y}', \mathbf{w} \rangle$
- Negative log-likelihood : (also called cross-entropy) $\ell(\mathbf{y}, \mathbf{w}) = - \langle\ \mathbf{y}, \mathbf{w}\ \rangle + \log \sum_i \exp w_i$

REGRESSION

hidden representation computed by the neural network



Prediction functions

- Trivial : $\hat{\mathbf{y}}(w) = w$

Loss functions

- Quadratique error loss : $\ell(y, w) = (y - w)^2$
- Absolute error loss : $\ell(y, w) = |y - w|$

NEURAL ARCHITECTURES: A REALLY QUICK OVERVIEW

NEURAL ARCHITECTURE DESIGN

Neural network = complicated parameterized function

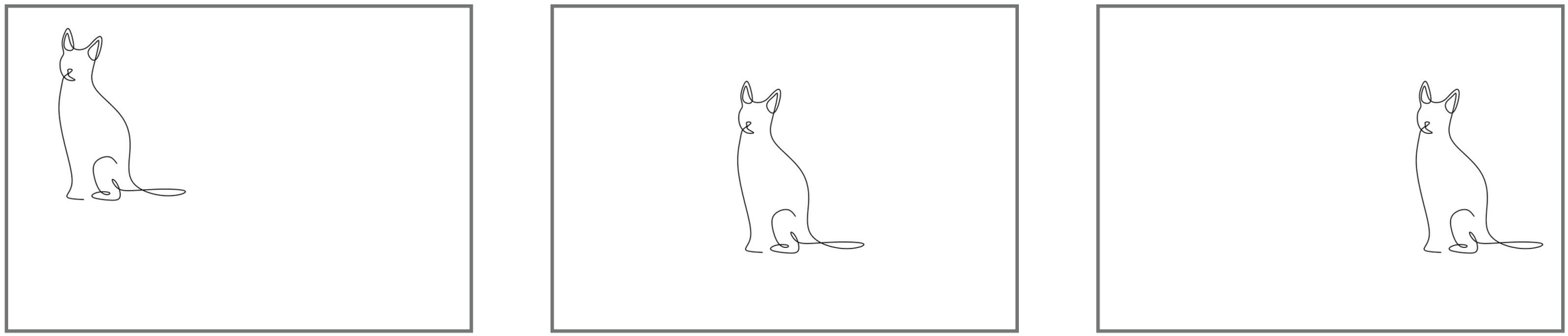
- Inductive bias: take into account the data to design the architectures
- Time complexity/speed
- Mathematical properties for efficient training:
  differentiability, prevent vanishing/exploding gradients

CONVOLUTIONAL NEURAL NETWORKS (CNN)

Intuition

No matter where the cat is in the picture, it is a cat

=> we want to encode this fact in the neural architecture!

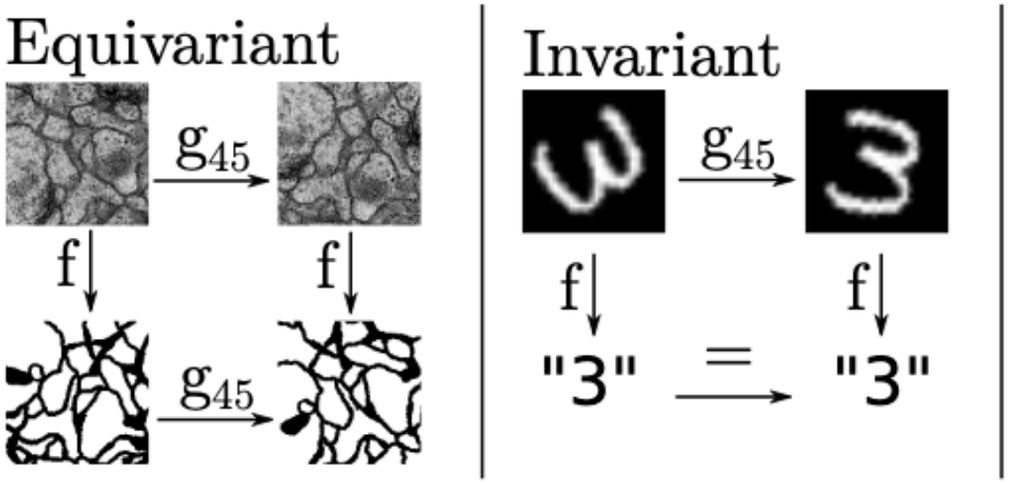


Equivariant function

If we apply a transformation on the input,
the output will be transformed in the « same » way

Invariant function

If we apply a transformation on the input,
the output will remain the same



EQUIVARIANT CONVOLUTIONS IN COMPUTER VISION

Translation equivariant convolution

Preserves the « translation structure »

- ➤ If the input is transposed
- ➤ The output is also transposed

+ pooling will make the model invariant

EQUIVARIANT CONVOLUTIONS IN COMPUTER VISION

Translation equivariant convolution

Preserves the « translation structure »

- ➤ If the input is transposed
- ➤ The output is also transposed

+ pooling will make the model invariant

Rotation equivariant convolution

Preserves the « rotation structure »

- ➤ If the input is rotated
- ➤ The output is also rotated

Standard convolution **is not** rotation equivariant

GROUP CONVOLUTIONS

[Cohen and Weiling, 2016]

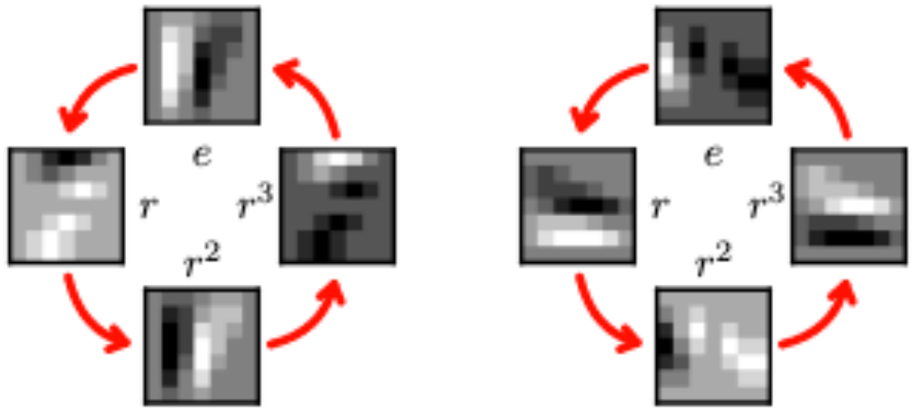


Figure 1. A p4 feature map and its rotation by r.

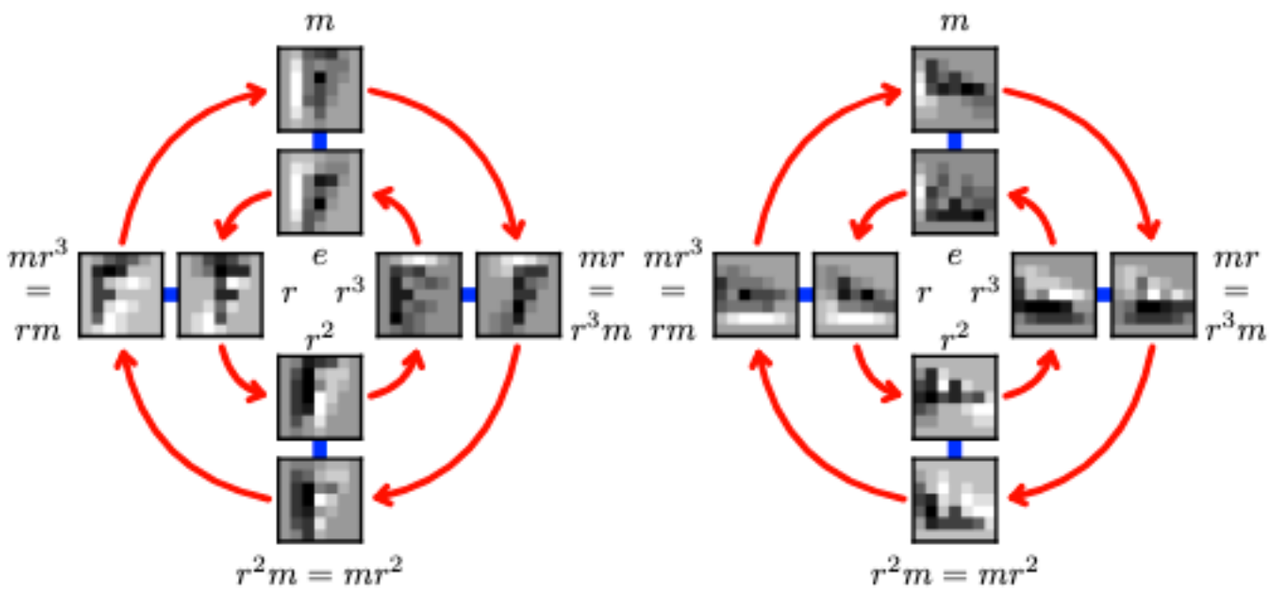
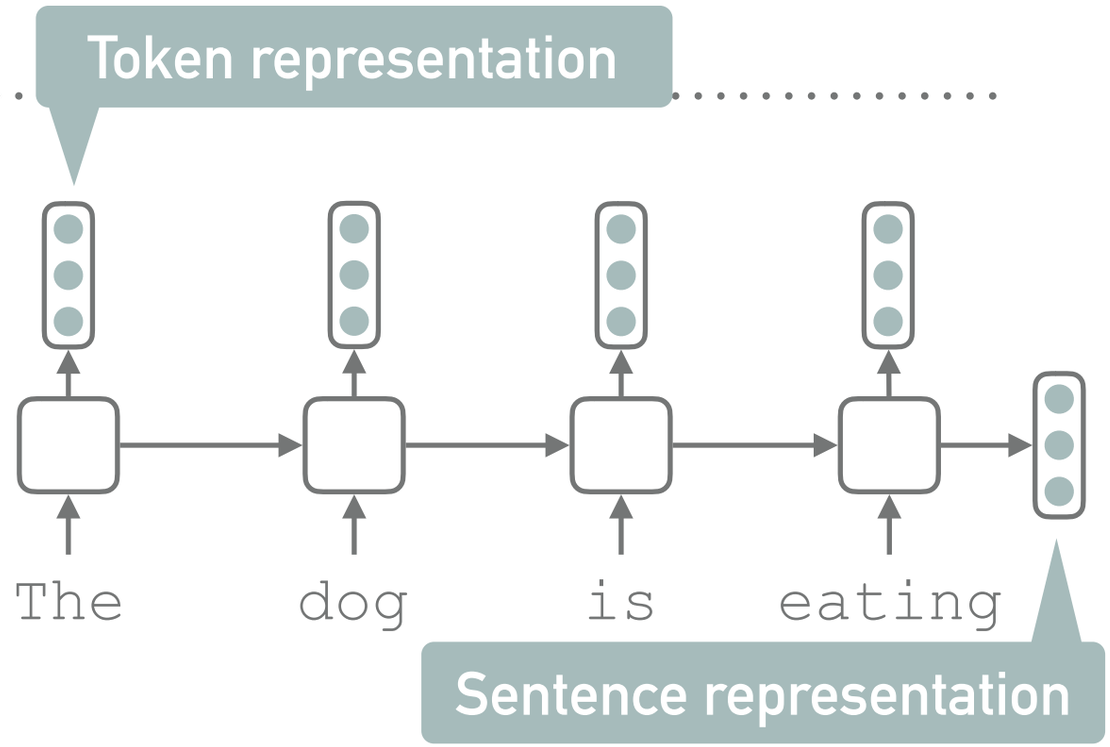


Figure 2. A p4m feature map and its rotation by r.

RECURRENT NEURAL NETWORKS

Recurrent neural networks

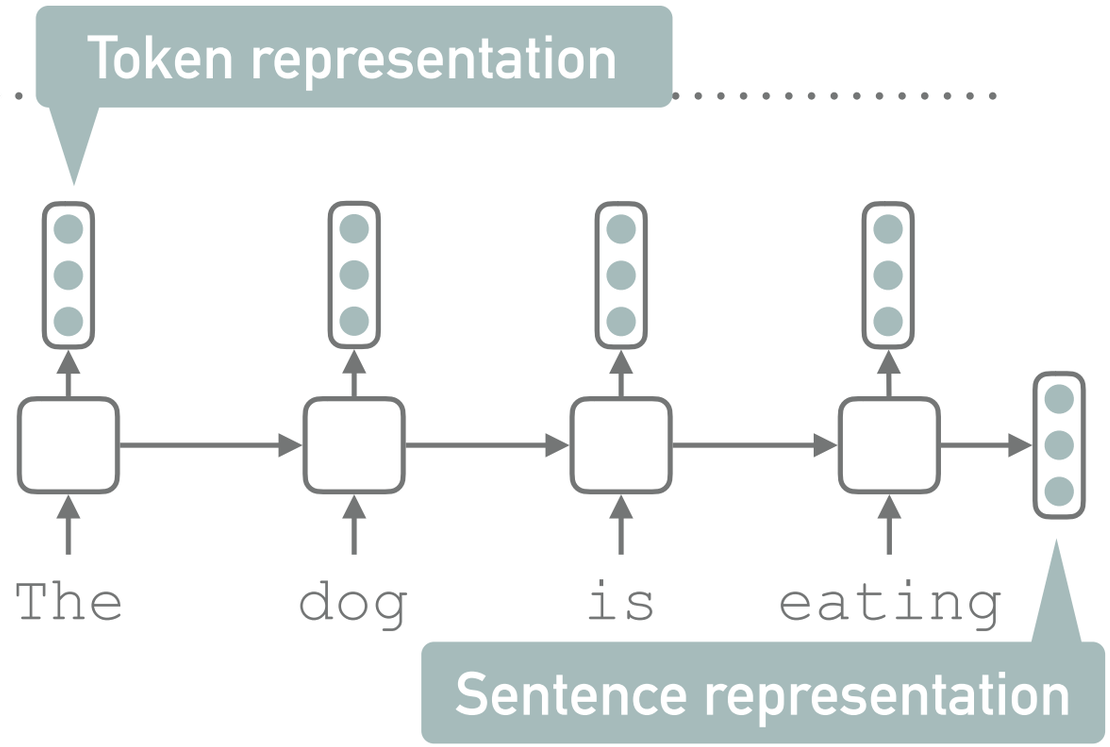
- Inputs are fed sequentially
- State representation updated at each input



RECURRENT NEURAL NETWORKS

Recurrent neural networks

- Inputs are fed sequentially
- State representation updated at each input



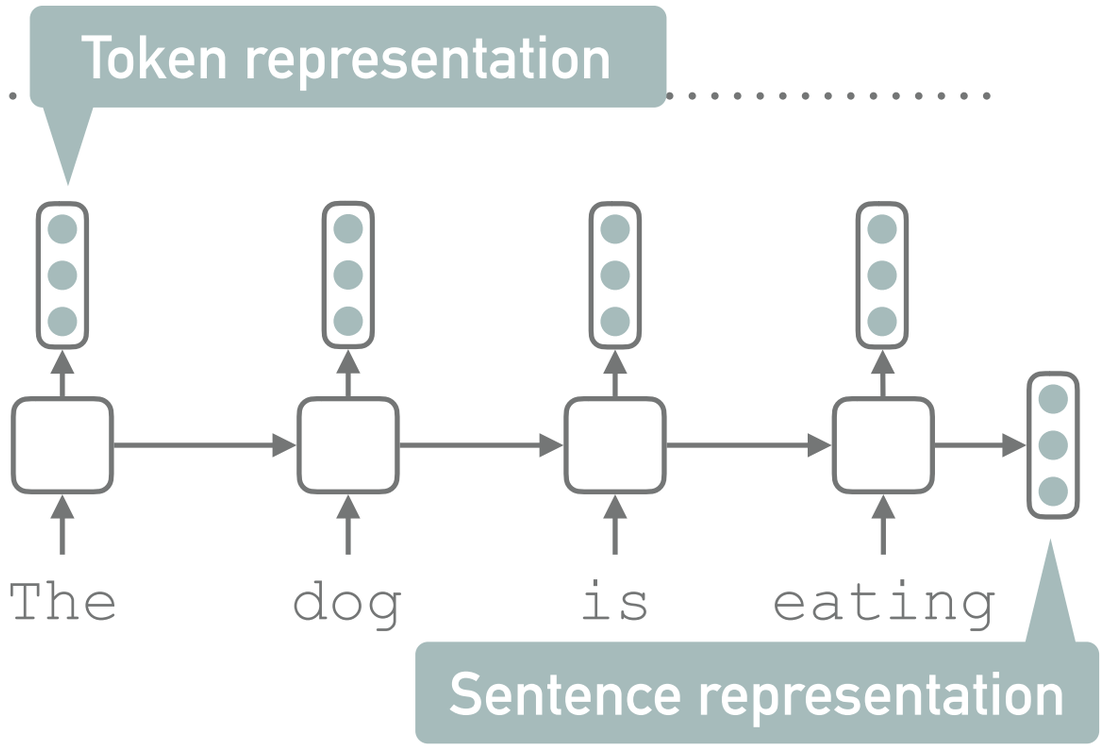
Intuition

Use two RNNs with **different trainable parameters**

RECURRENT NEURAL NETWORKS

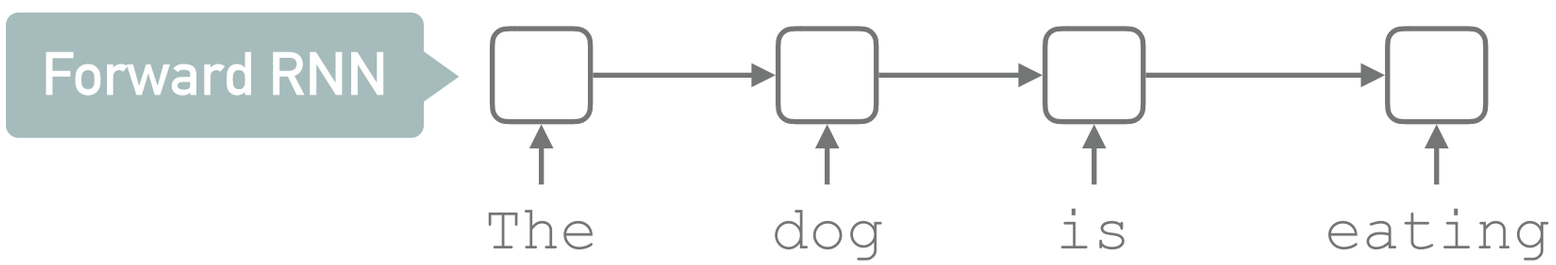
Recurrent neural networks

- Inputs are fed sequentially
- State representation updated at each input



Intuition

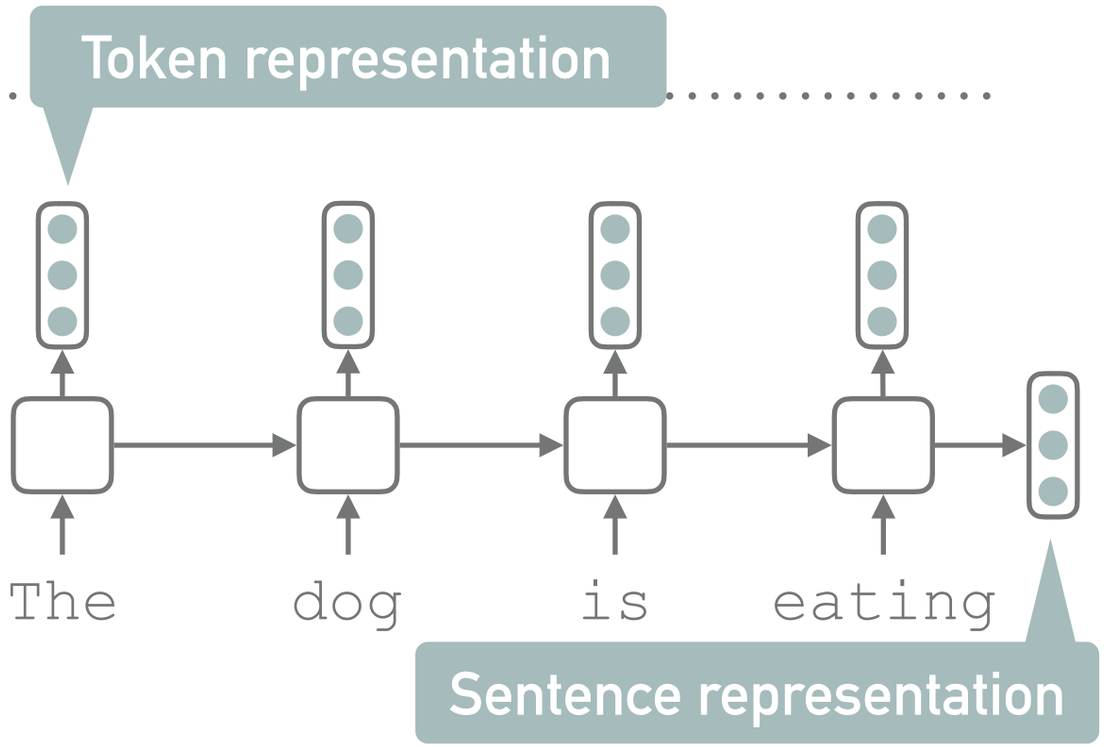
Use two RNNs with **different trainable parameters**



RECURRENT NEURAL NETWORKS

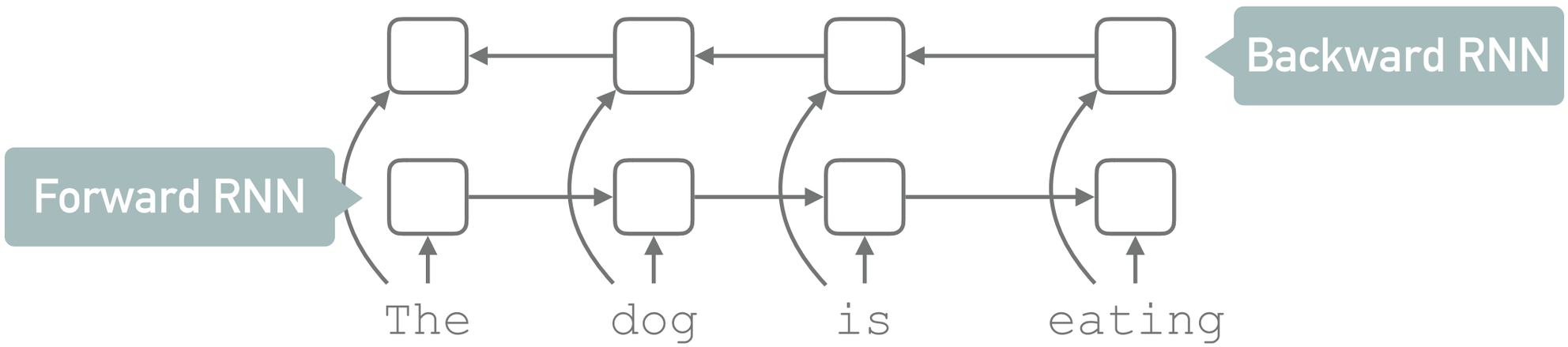
Recurrent neural networks

- Inputs are fed sequentially
- State representation updated at each input



Intuition

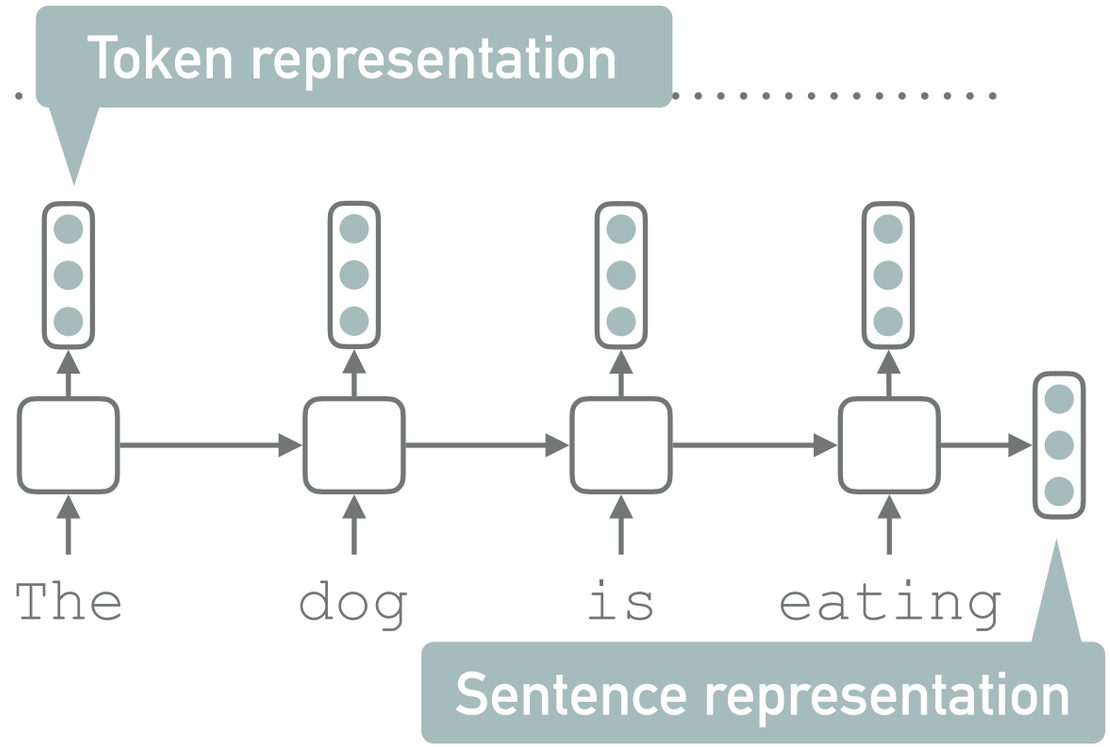
Use two RNNs with **different trainable parameters**



RECURRENT NEURAL NETWORKS

Recurrent neural networks

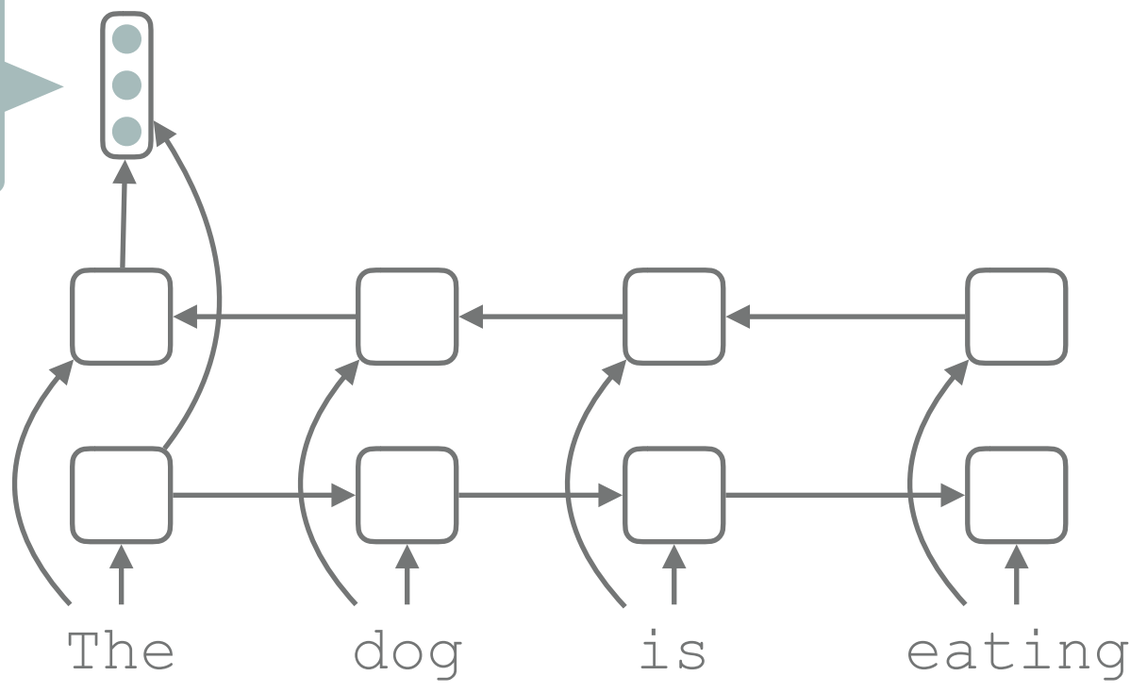
- Inputs are fed sequentially
- State representation updated at each input



Intuition

Use two RNNs with **different trainable parameters**

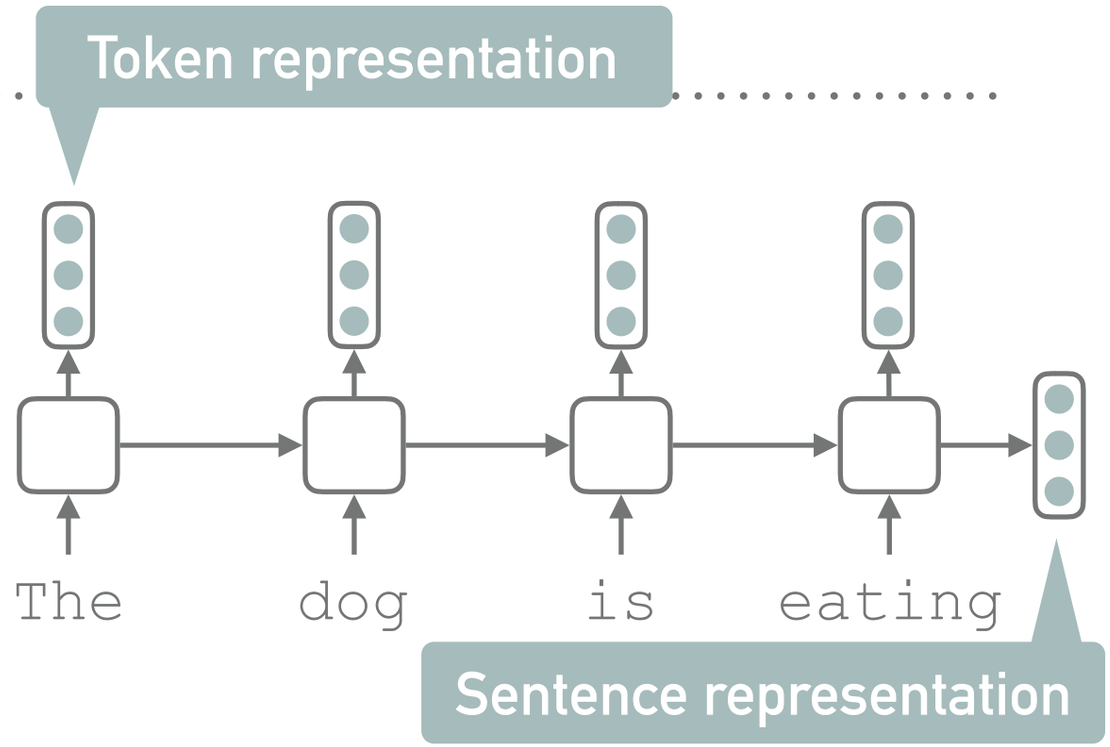
For token representation, we concatenate the output of each RNN



RECURRENT NEURAL NETWORKS

Recurrent neural networks

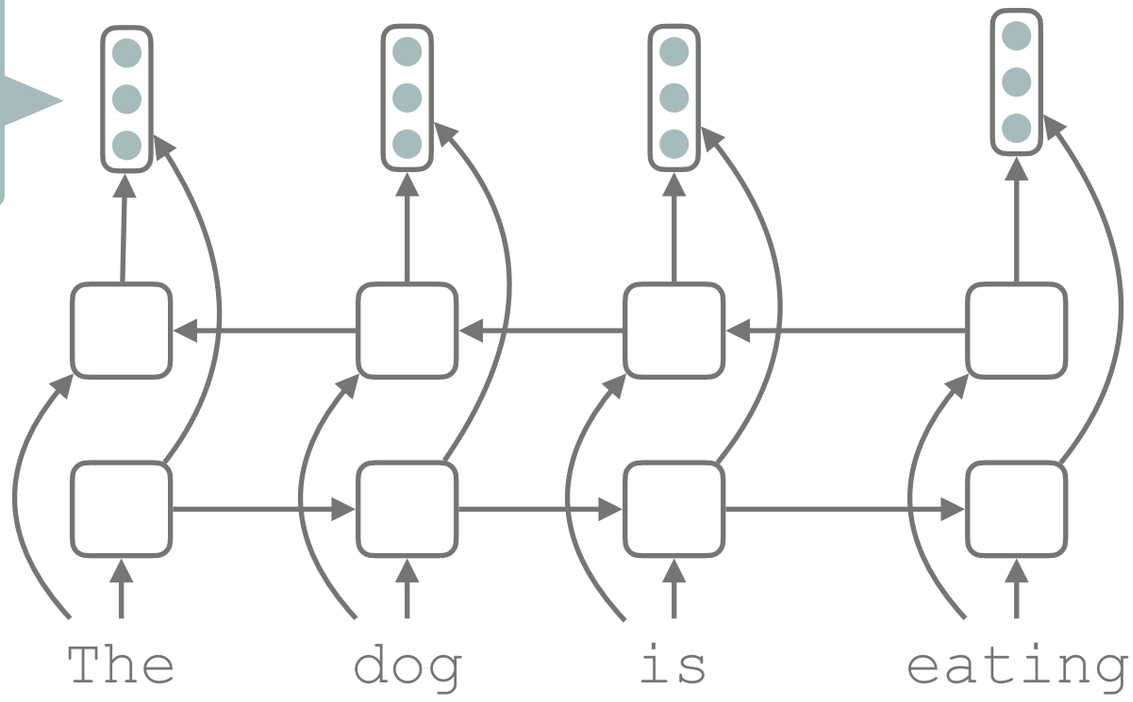
- Inputs are fed sequentially
- State representation updated at each input



Intuition

Use two RNNs with **different trainable parameters**

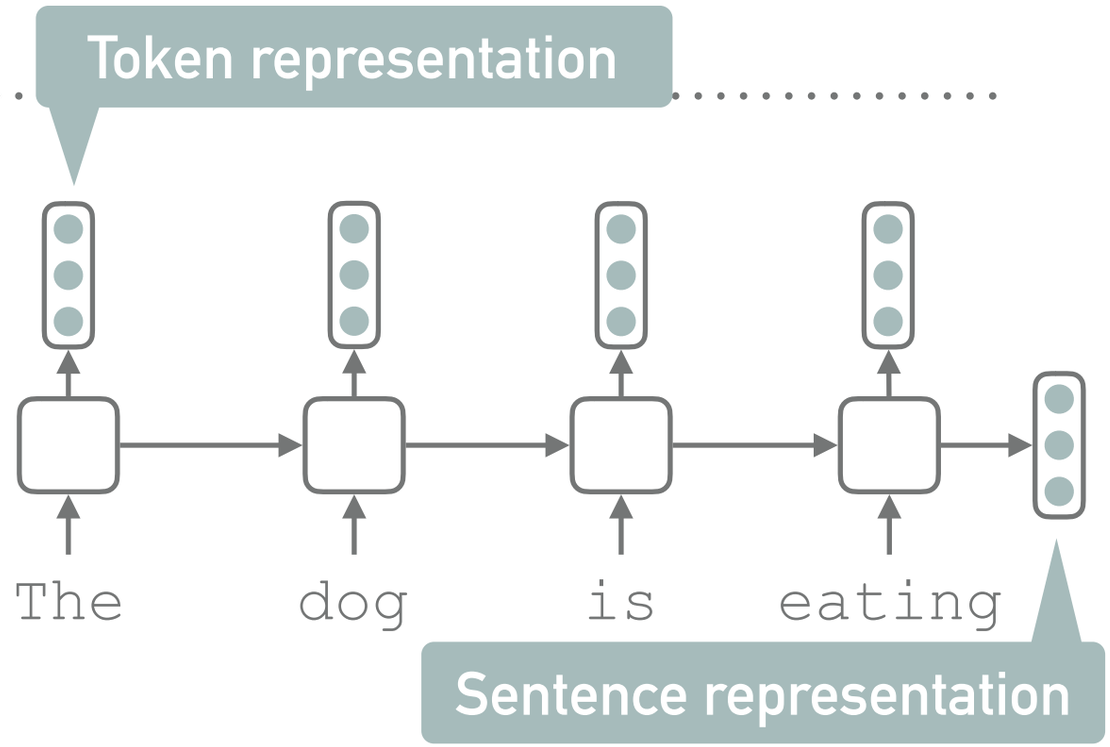
For token representation, we concatenate the output of each RNN



RECURRENT NEURAL NETWORKS

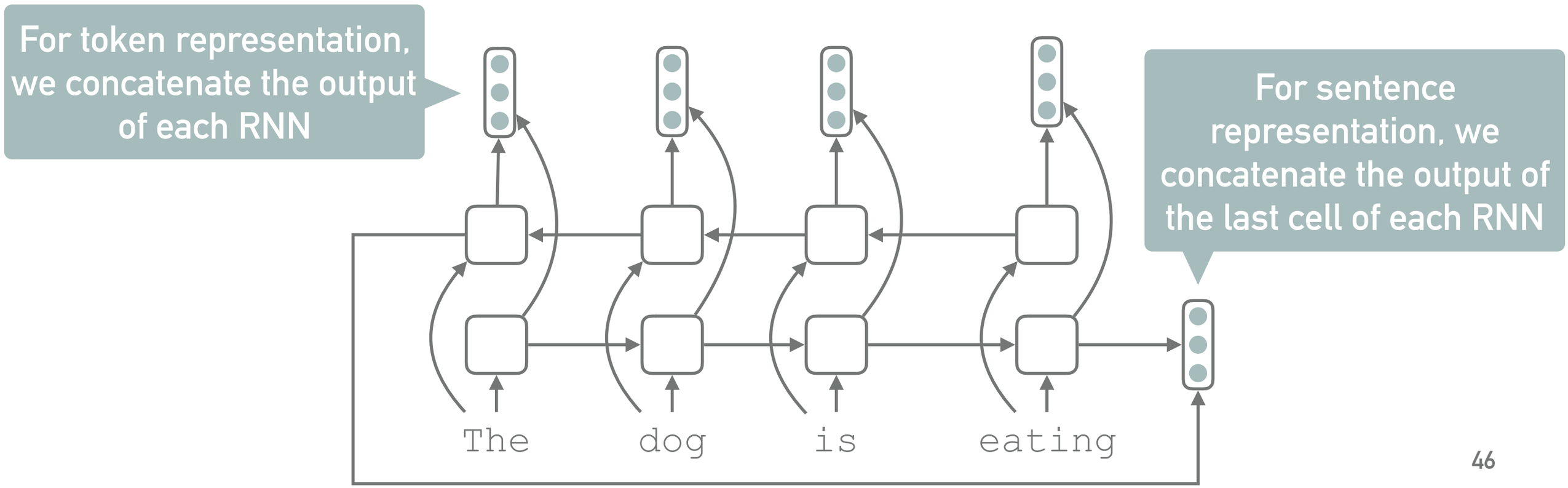
Recurrent neural networks

- Inputs are fed sequentially
- State representation updated at each input



Intuition

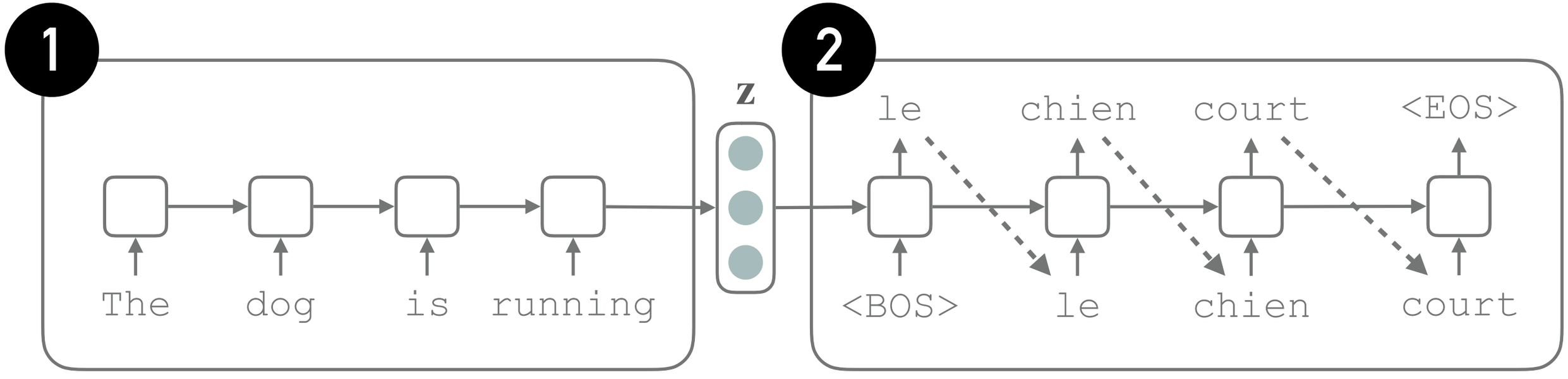
Use two RNNs with **different trainable parameters**



SEQUENCE TO SEQUENCE (SEQ2SEQ)

Intuition

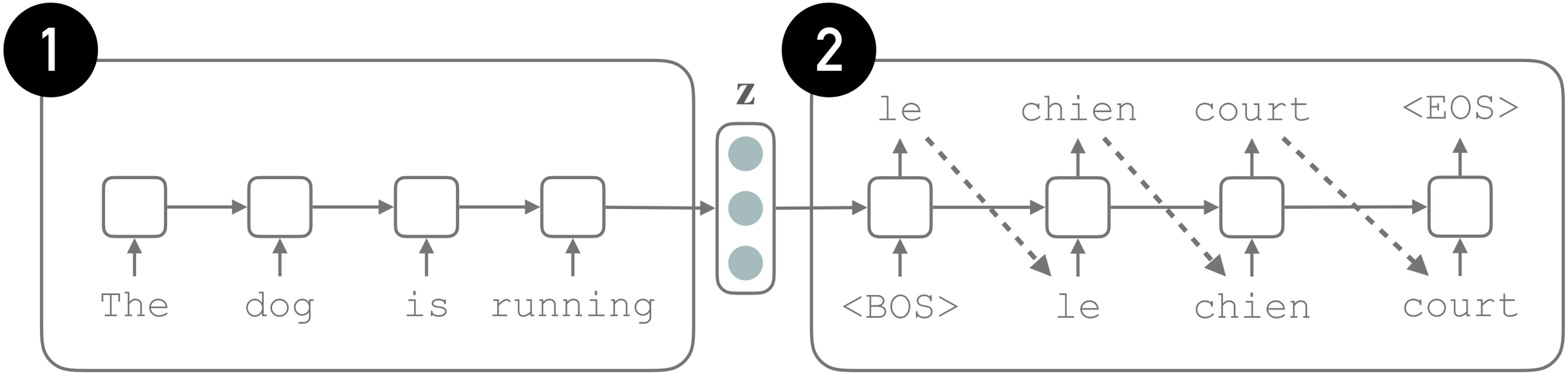
1. **Encoder:** encode the input sentence into a fixed size vector (sentence embedding)
2. **Decoder:** generate the translation auto-regressively (word by word) conditioned on the input sentence embedding



SEQUENCE TO SEQUENCE (SEQ2SEQ)

Intuition

1. **Encoder:** encode the input sentence into a fixed size vector (sentence embedding)
2. **Decoder:** generate the translation auto-regressively (word by word) conditioned on the input sentence embedding



The sentence embedding is a bottleneck, everything must be encoded inside!

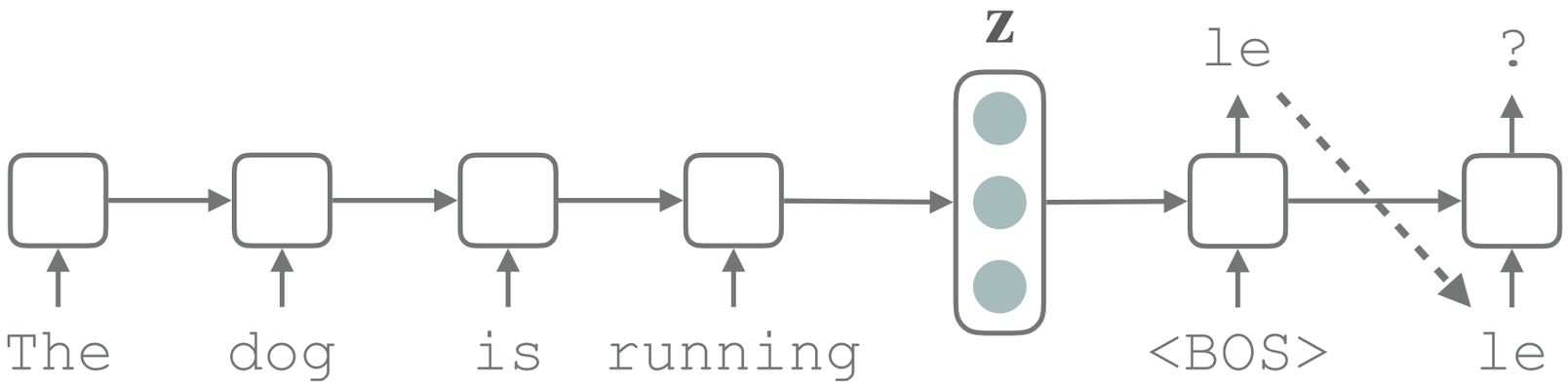
SEQ2SEQ WITH ATTENTION

[Bahdanau et al., 2014]

Intuition

- During decoding, we want to « look » at the input sentence
- Particularly, we want to focus on specific words

Here we need to generate « chien », so maybe we could look at « dog » in the input to help?

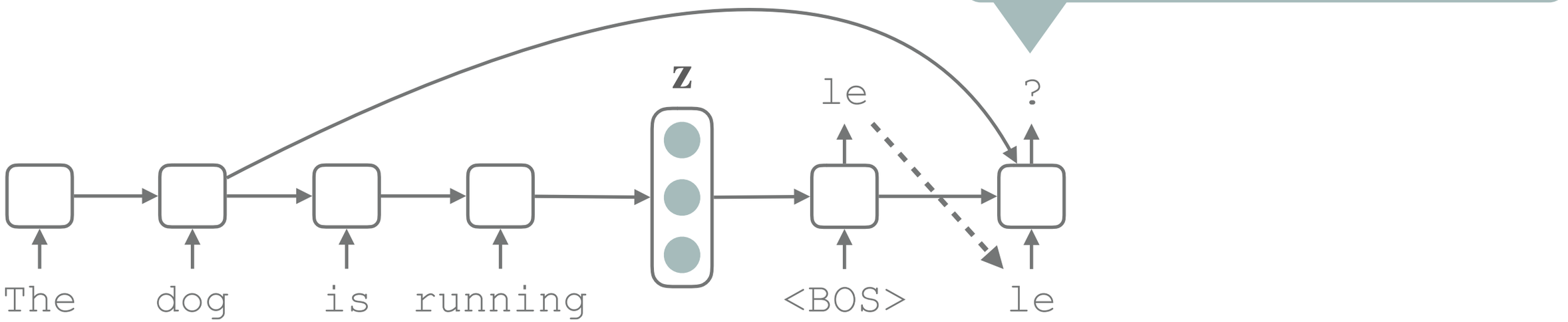


SEQ2SEQ WITH ATTENTION

[Bahdanau et al., 2014]

Intuition

- During decoding, we want to « look » at the input sentence
- Particularly, we want to focus on specific words



Attention mechanism

We had a « module » that wil learn to look at a word from the input

TAKEAWAY

You need to understand the problem you try to solve in order to build good neural architecture

- Michèle will present several neural architectures
- Other course: « Deep Learning for Natural Language Processing » (Master 2)
  Focus on specific neural architectures for text!

NEURAL NETWORK TRAINING

GRADIENT-BASED TRAINING

Neural network

Parameterized function $f_\theta : \mathcal{X} \rightarrow \mathcal{Y}$

Feature space

Output space

Parameters

Training

- Labeled example: features + « gold » answer
- Train set: $D = \{(x^{(i)}, y^{(i)})\}_{i=1}^{n}$
- Find parameters θ so that $f_\theta(x^{(i)}) \simeq y^{(i)}, \forall i$

End-to-end training

- In the old days: layer per layer training (in some kind of generative model)
- Nowadays: Train all parameters at the same time
  (+ unsupervised pretraining in some cases => DL4NLP course)

Testing / evaluation

- Test if the model **generalizes** to unseen data (i.e. disjoint set from the train set)

LOSS FUNCTION

Intuition

- Compare the output with the gold output (i.e. the expected output)
- The loss must be minimized (& bounded below by 0)
- Must be related to the evaluation function, but often slightly different

Learning objective

$$\theta^* = \text{argmin}_\theta \quad \frac{1}{n} \quad \sum_{i=1}^{n} \quad l(\ y^{(i)},\ f_\theta(\mathbf{x}^{(i)})\ )$$

- Modern machine learning is optimization

GRADIENT DESCENT

Problem

Solve: $\min_{\theta} g(\theta)$

Intuition

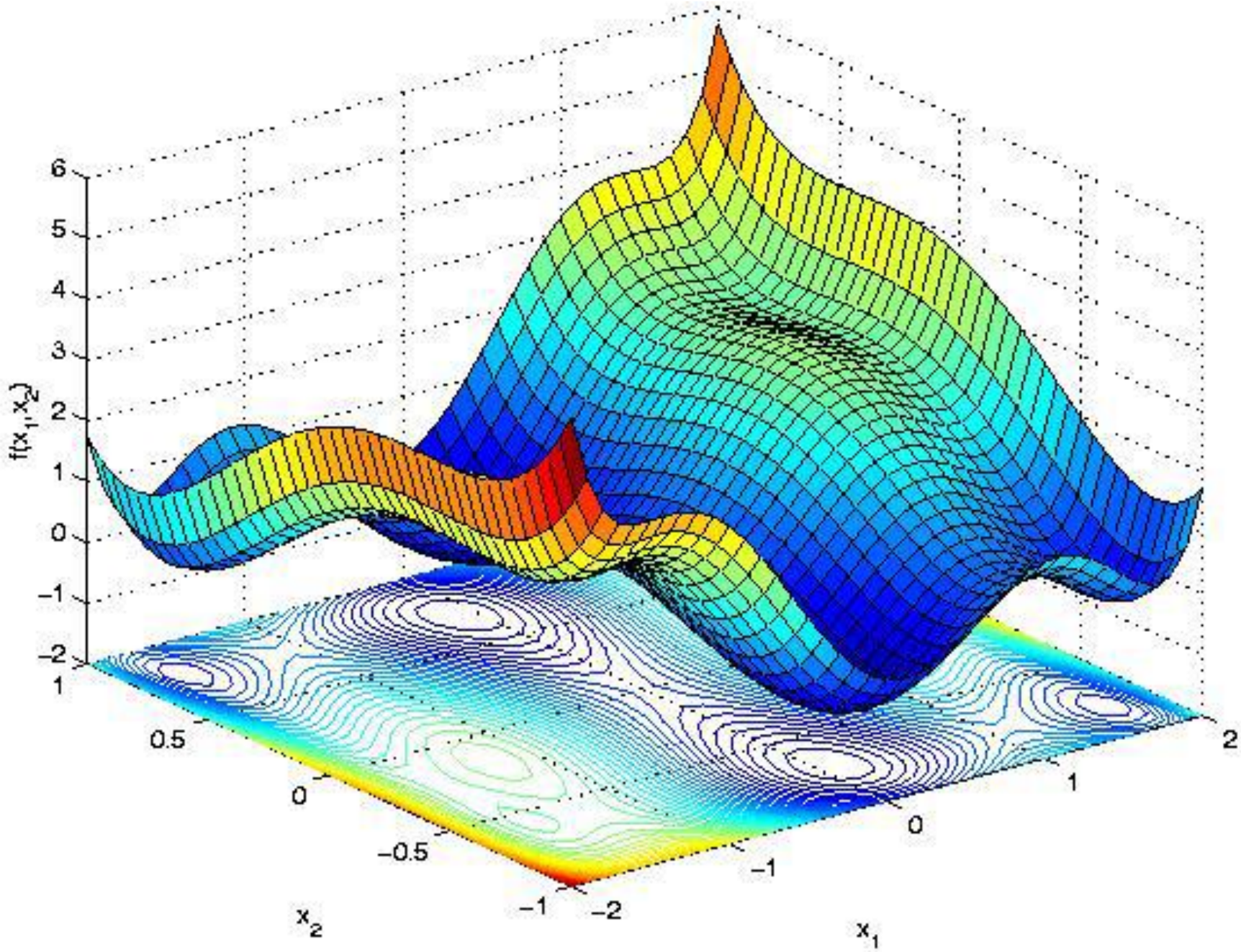
- All you can compute: evaluate the function and its gradient at a given point
- You can use gradient information to see in which direction the function is decreasing
- Therefore: just make a small step in this direction!
- In this course we won't differentiate between gradient and sub-gradient

Formally

- Choose an initial point randomly: $\theta^{(0)}$
- Make T iterations/steps: $\theta^{(t+1)} = \theta^{(t)} - \eta \times \nabla_{\theta} g(\theta)$

Stepsize

NON-CONVEX FUNCTION ILLUSTRATION



Many local minima! Do we care? NO

TRAIN/DEV/TEST

Parameters vs. hyper-parameters

- Parameters: the parameters of the function, which are learned during training
- Hyper-parameters: the parameters of the training algorithm and the neural architecture choice (number of layers, hidden representation dimensions, …)

Three datasets

- Train set:
  Used to compute the objective and its gradient
- Development / validation set:
  Used during training to choose hyper-parameters and to know when to stop training
- Test set:
  Used to evaluate the model!

THE OVERFITTING PROBLEM

GENERALIZATION

Overparameterized neural networks

- Networks where the number of parameters exceed the training dataset size.
- Can learn by heart the dataset,
  i.e. **overfit the data** -> does not generalize well to unseen data
- Are easier to optimize in practice

Monitoring the training process

- Loss should go down
- Training accuracy should go up
- Dev accuracy should go up

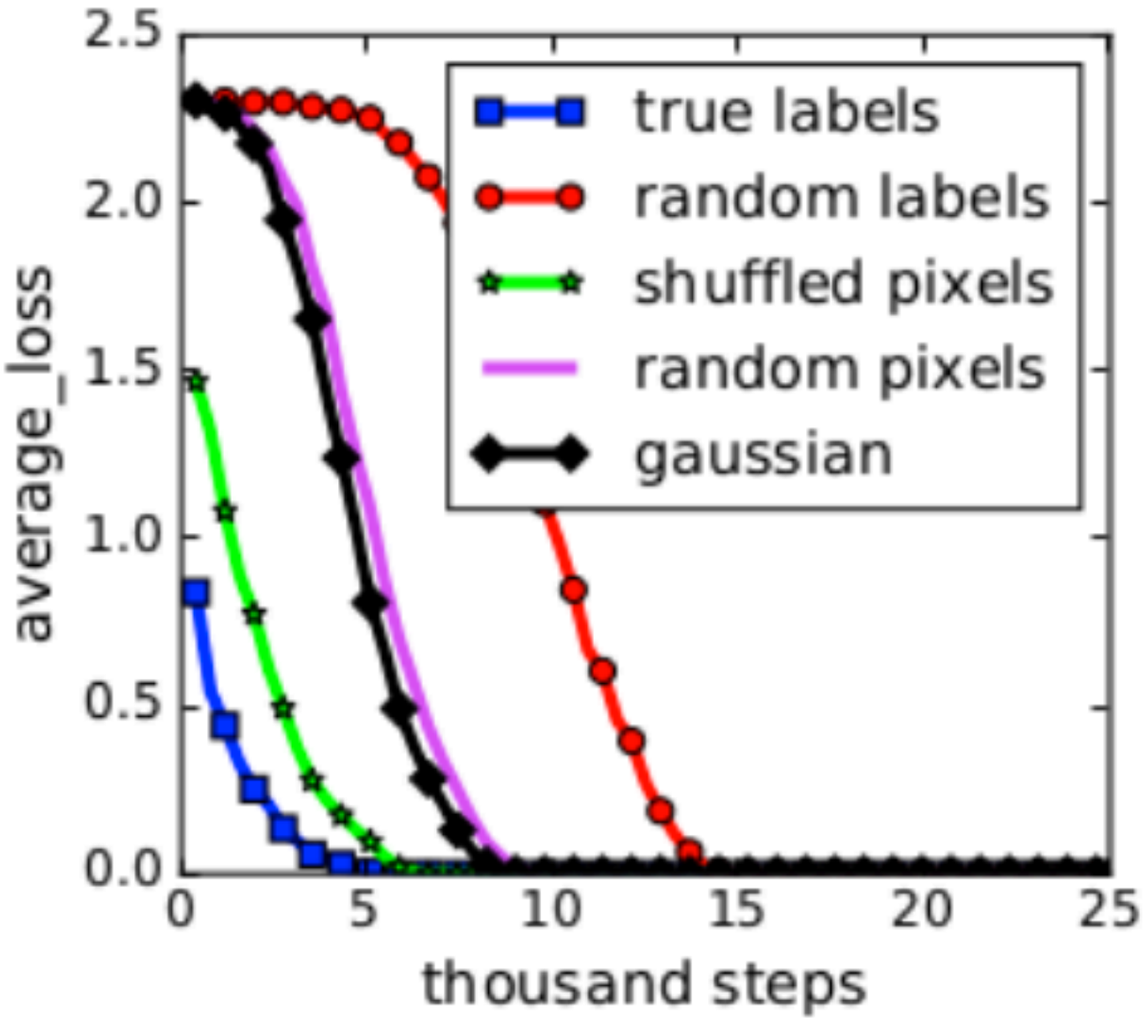
Regularization

Techniques to control parameters during learning and prevent overfitting

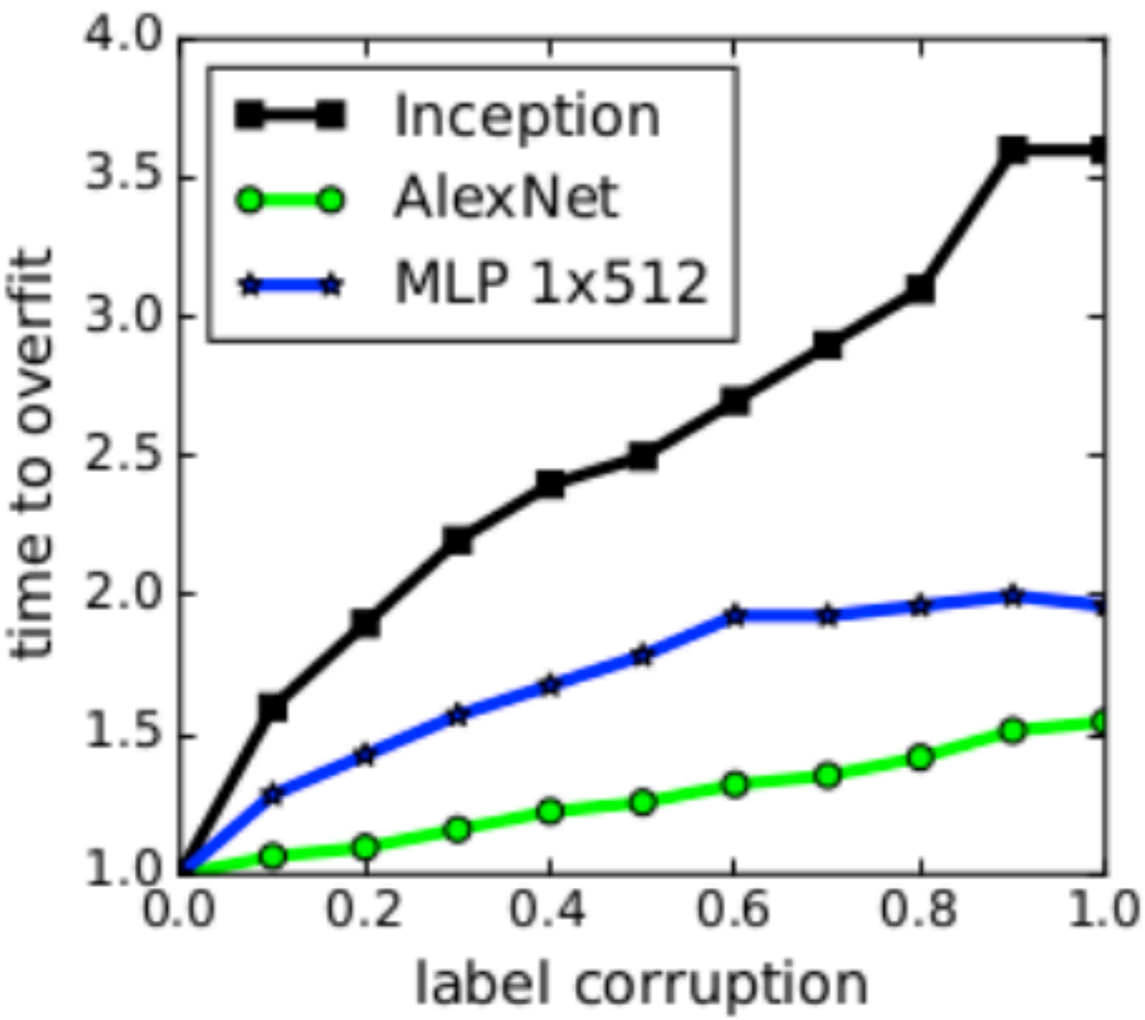
LEARNING WITH RANDOM INPUTS AND LABELS 1/2 [ZHANG ET AL., 2017]

model	# params	random crop	weight decay	train accuracy	test accuracy
Inception	1,649,402	yes	yes	100.0	89.05
		yes	no	100.0	89.31
		no	yes	100.0	86.03
		no	no	100.0	85.75
(fitting random labels)		no	no	100.0	9.78
Inception w/o BatchNorm	1,649,402	no	yes	100.0	83.00
		no	no	100.0	82.00
(fitting random labels)		no	no	100.0	10.12
Alexnet	1,387,786	yes	yes	99.90	81.22
		yes	no	99.82	79.66
		no	yes	100.0	77.36
		no	no	100.0	76.07
(fitting random labels)		no	no	99.82	9.86
MLP 3x512	1,735,178	no	yes	100.0	53.35
		no	no	100.0	52.39
(fitting random labels)		no	no	100.0	10.48
MLP 1x512	1,209,866	no	yes	99.80	50.39
		no	no	100.0	50.51
(fitting random labels)		no	no	99.34	10.61

LEARNING WITH RANDOM INPUTS AND LABELS 2/2 [ZHANG ET AL., 2017]



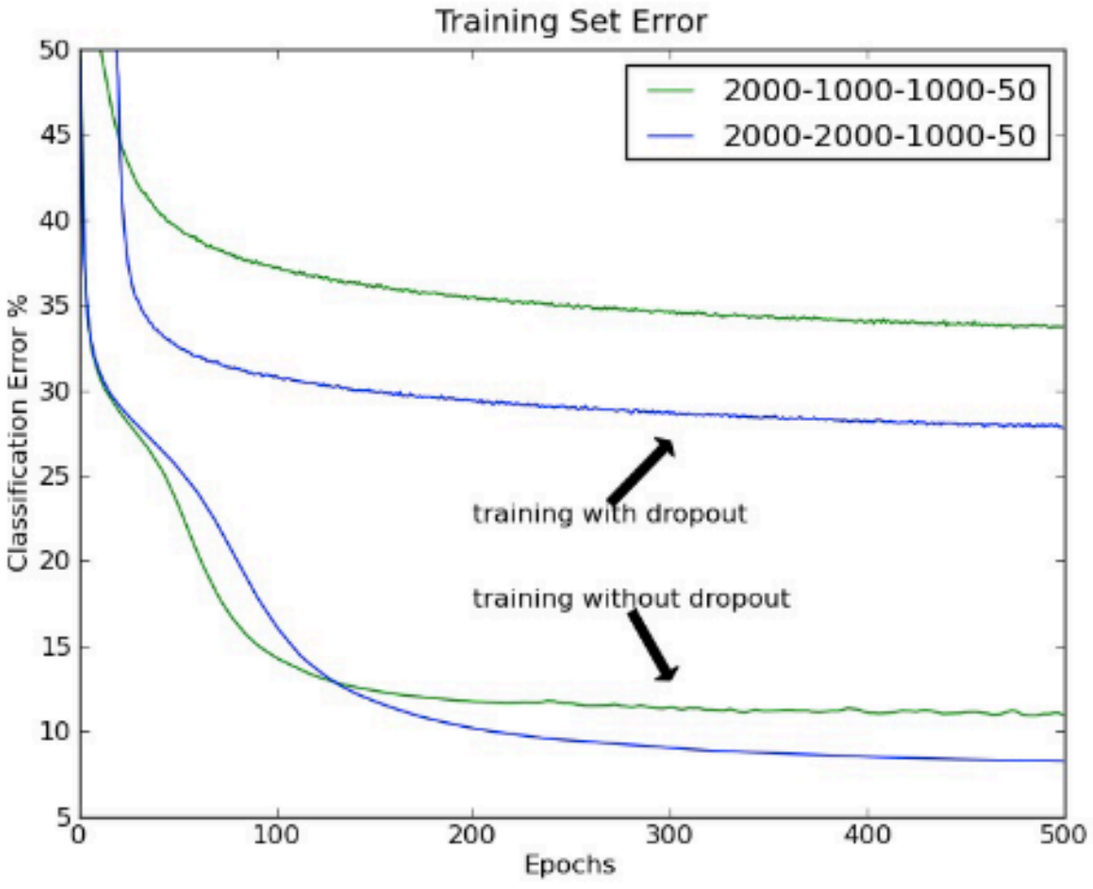
(a) learning curves

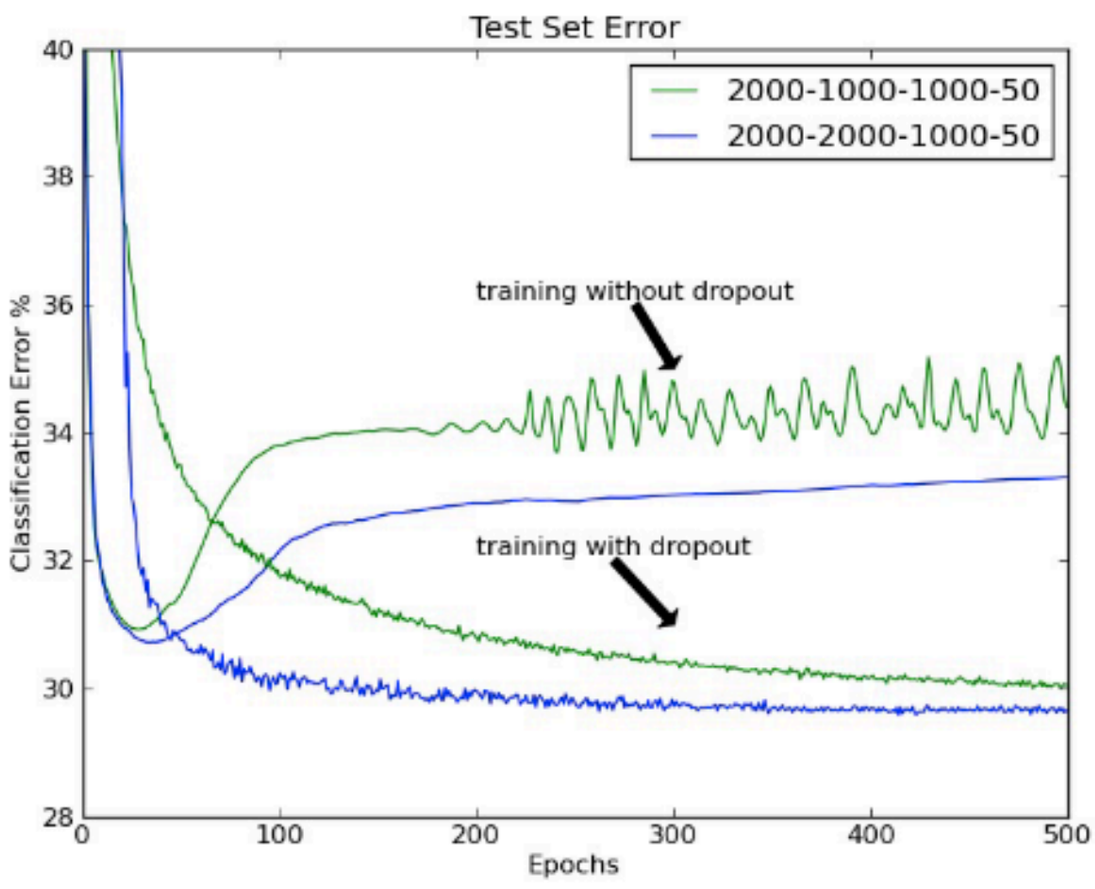


(b) convergence slowdown

SOLVING THE OVERFITTING PROBLEM VIA DROPOUT [HINTON ET AL., 2012]

(more details about dropout next week, just for illustration)





COMPUTATIONAL GRAPHS

MAIN IDEA BEHIND NN LIBRARIES

Problem

- We need the gradient of the objective for training
- We don't want to compute it by ourselves, too complicated

Back-propagation algorithm (next week!)

- Forward pass: define the function to compute (i.e. the objective)
- Backward pass: automatically compute the gradient wrt parameters :)

Computational graph

During the forward pass, we construct a computational graph that retain all operations used to compute the objective

A TYPOLOGY OF NEURAL NETWORK LIBRARIES

Static computational graphs

Defines the computation graph once for all, just update the inputs
(ex: Tensorflow, Dynet C++ API)

Dynamic computational graphs

Each time we need to compute a value, we have to rebuild the full graph

- Eager: computation are done immediately (ex: Pytorch 1, Tensorflow)
- Lazy: first define the computation, the execute it (ex: Dynet)
  => allows for forward pass optimization!