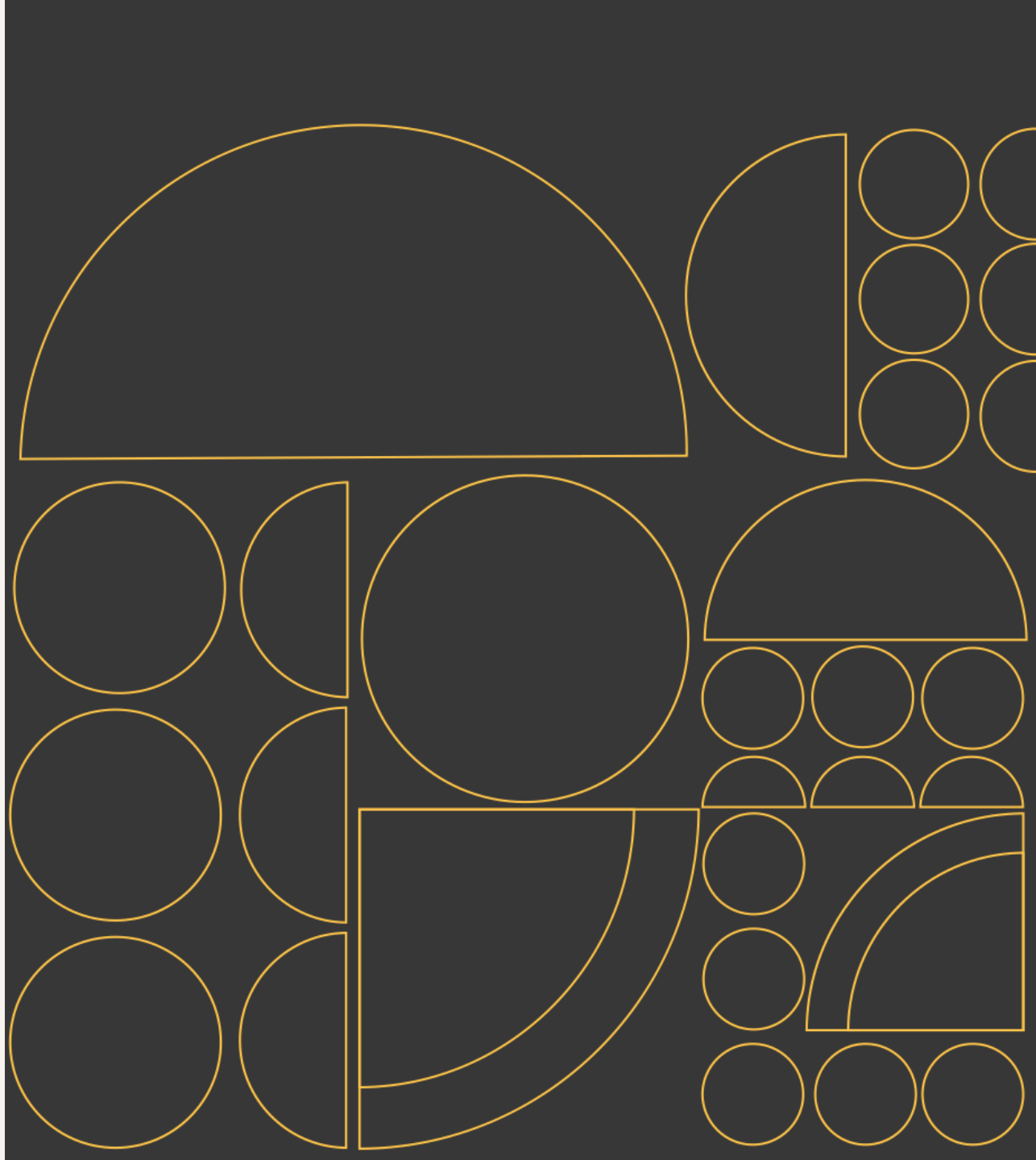


ChEESE

Artificial Intelligence and Machine Learning for Geosciences

Barcelona November 2024

Léonard Seydoux, Hugo Frezat,
Geneviève Moguilny & Alexandre Fournier

cheese-ai-for-geosciences

Project funded by EuroHPC under the grant agreement No 101093038.

Goal: Learn about statistical inference and machine learning

1. **Identify** data-related scientific problems
2. **Define** the problem and design a solution
3. **Learn** from examples in the litterature
4. **Criticize** the litterature
5. **Train** on real geoscience problems

RESEARCH

REVIEW SUMMARY

GEOPHYSICS

Machine learning for data-driven discovery in solid Earth geoscience

Karianne J. Bergen, Paul A. Johnson, Maarten V. de Hoop, Gregory C. Beroza*

BACKGROUND: The solid Earth, oceans, and atmosphere together form a complex interacting geosystem. Processes relevant to understanding Earth's geosystem behavior range in spatial scale from the atomic to the planetary, and in temporal scale from milliseconds to billions of years. Physical, chemical, and biological processes interact and have substantial influence on this complex geosystem, and humans interact with it in ways that are increasingly consequential to the future of both the natural world and civilization as the finiteness of Earth becomes increasingly apparent and limits on available energy, mineral resources, and fresh water increasingly affect the human condition. Earth is subject to a variety of geohazards that are poorly understood, yet increasingly impactful as our exposure grows through increasing urbanization, particularly in hazard-prone areas. We have a fundamental need to develop the best possible predictive understanding of how the geosystem works, and that understanding must be informed by both the present and the deep past. This understanding will come through the analysis of increasingly large geo-datasets and from computationally intensive simulations, often connected through inverse problems. Geoscientists are faced with the challenge of extracting as much useful information as possible and gaining new insights from these data, simulations, and the interplay between the two. Techniques from the rapidly evolving field of machine learning (ML) will play a key role in this effort.

ADVANCES: The confluence of ultrafast computers with large memory, rapid progress in ML algorithms, and the ready availability of large datasets place geoscience at the threshold of dramatic progress. We anticipate that this progress will come from the application of ML across three categories of research effort: (i) automation to perform a complex prediction task that cannot easily be described by a set of explicit commands; (ii) modeling and inverse problems to create a representation that approximates numerical simulations or captures relationships; and (iii) discovery to reveal new and often unanticipated patterns, structures, or relationships. Examples of automation include geologic mapping using remote-sensing data, characterizing the topology of fracture systems to model subsurface transport, and classifying volcanic ash particles to infer eruptive mechanism. Examples of modeling include approximating the viscoelastic response for complex rheology, determining wave speed models directly from tomographic data, and classifying diverse seismic events. Examples of discovery include predicting laboratory slip events using observations of acoustic emissions, detecting weak earthquake signals using similarity search, and determining the connectivity of subsurface reservoirs using groundwater tracer observations.

ON OUR WEBSITE

Read the full article at http://dx.doi.org/10.1126/science.aau0323

OUTLOOK: The use of ML in solid Earth geosciences is growing rapidly, but is still in its early stages and making uneven progress. Much remains to be done with existing datasets from long-standing data sources, which in many cases are largely unexplored. Newer, unconventional data sources such as light detection and ranging (LiDAR), fiber-optic sensing, and crowd-sourced measurements may demand new approaches through both the volume and the character of information that they present.

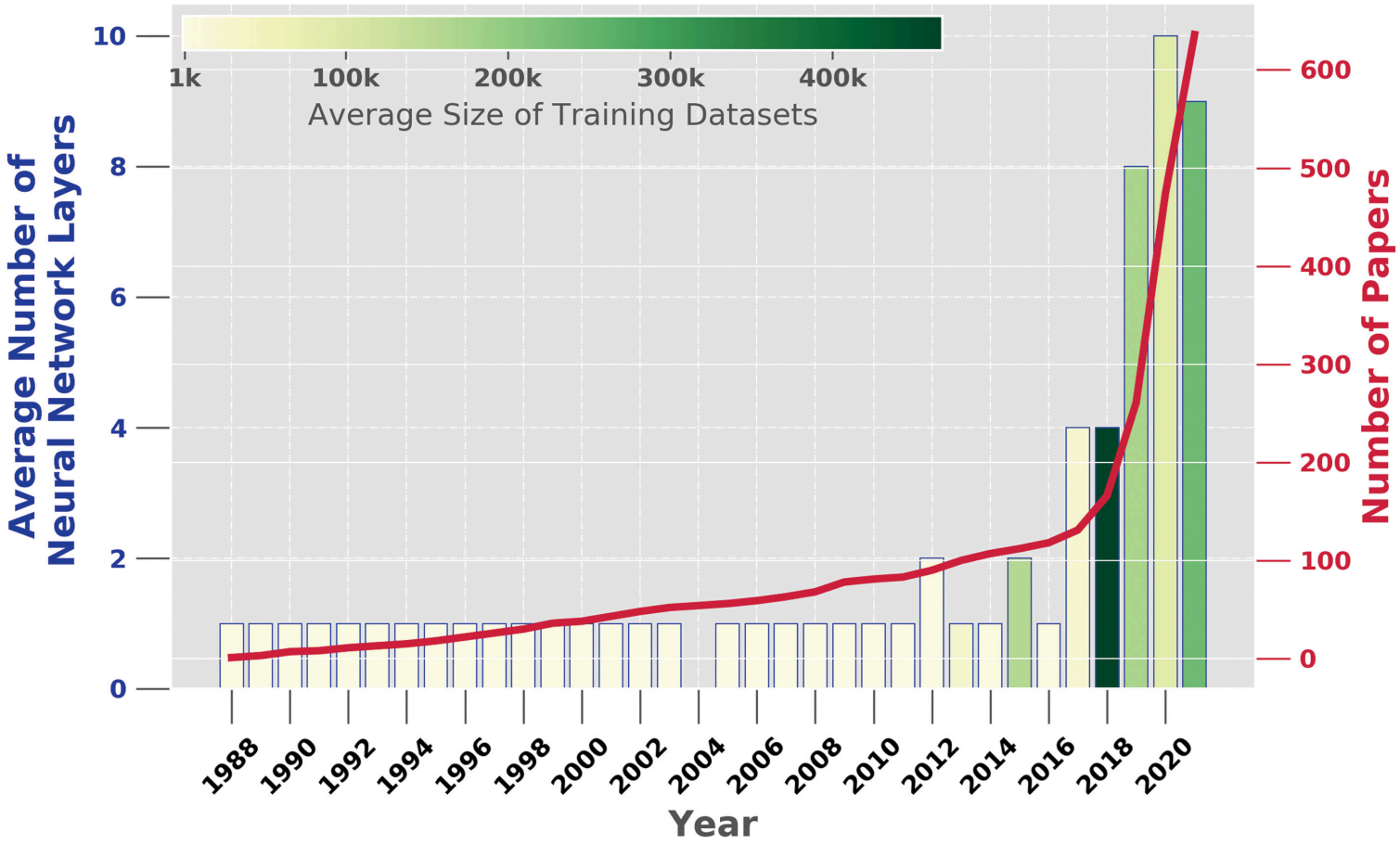
Practical steps could accelerate and broaden the use of ML in the geosciences. Wider adoption of open-science principles such as open source code, open data, and open access will better position the solid Earth community to take advantage of rapid developments in ML and artificial intelligence. Benchmark datasets and challenge problems have played an important role in driving progress in artificial intelligence research by enabling rigorous performance comparison and could play a similar role in the geosciences. Testing on high-quality datasets produces better models, and benchmark datasets make these data widely available to the research community. They also help recruit expertise from allied disciplines. Close collaboration between geoscientists and ML researchers will aid in making quick progress in ML geoscience applications. Extracting maximum value from geoscientific data will require new approaches for combining data-driven methods, physical modeling, and algorithms capable of learning with limited, weak, or biased labels. Funding opportunities that target the intersection of these disciplines, as well as a greater component of data science and ML education in the geosciences, could help bring this effort to fruition. ■

Digital geology. Digital representation of the geology of the conterminous United States. [Geology of the Conterminous United States at 1:2,500,000 scale; a digital representation of the 1974 P. B. King and H. M. Beikman map by P. G. Schruben, R. E. Arndt, W. J. Bawiec]

The list of author affiliations is available in the full article online.
*Corresponding author. Email: beroza@stanford.edu
Cite this article as K. J. Bergen et al., Science 363, eaau0323 (2019). DOI: 10.1126/science.aau0323

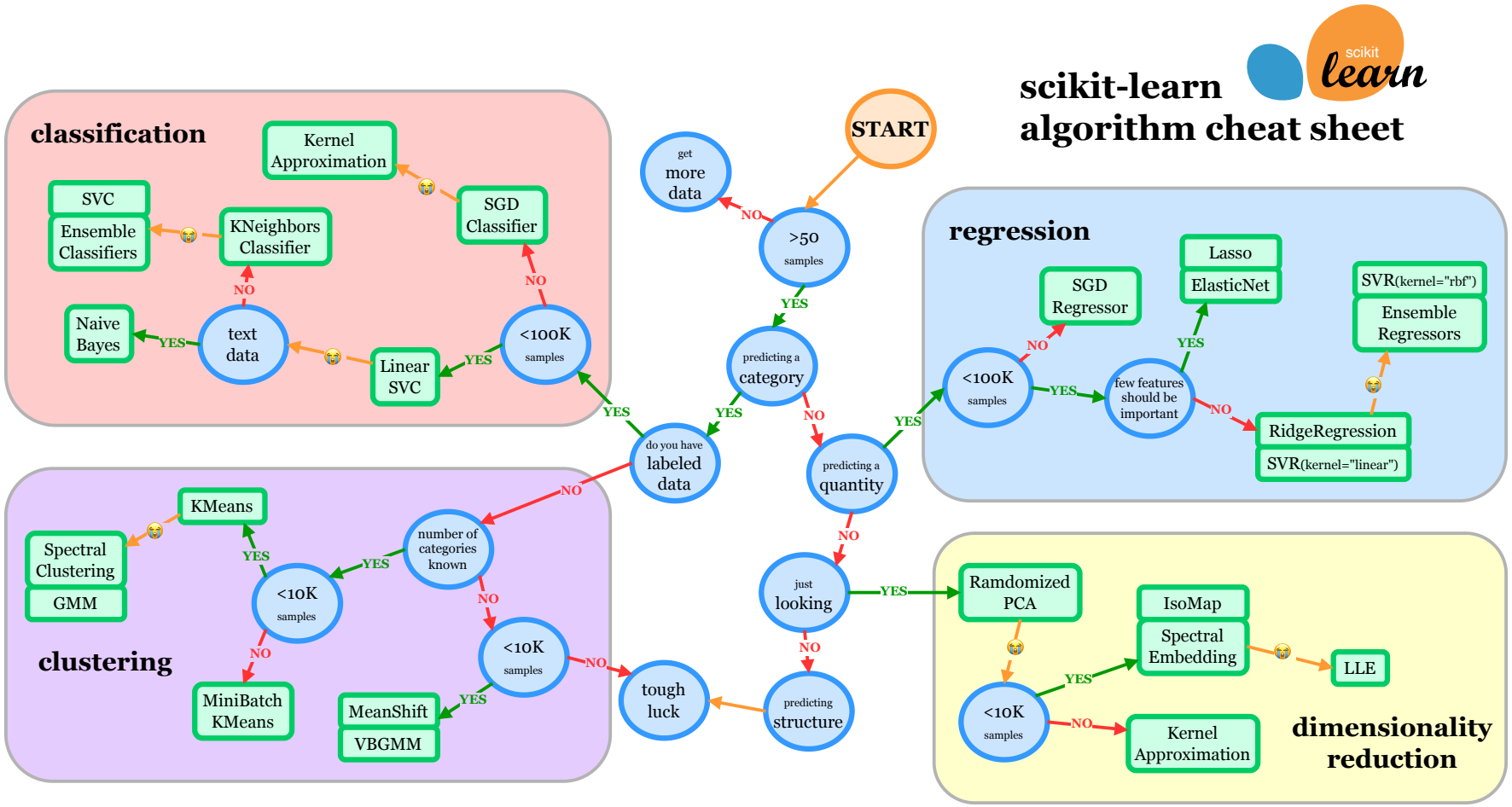
Bergen et al. (2019)

Goal: Keep up with the ongoing pace



Mousavi et al. (2022)

Contents of this class make use of the scikit-learn library



Contents of this class make use of the scikit-learn library

scikit-learn

Machine Learning in Python

Getting Started Release Highlights for 1.5

- Simple and efficient tools for predictive data analysis
- Accessible to everybody, and reusable in various contexts
- Built on NumPy, SciPy, and matplotlib
- Open source, commercially usable - BSD license

Classification

Identifying which category an object belongs to.

Applications: Spam detection, image recognition.
Algorithms: Gradient boosting, nearest neighbors, random forest, logistic regression, and more...

Regression

Predicting a continuous-valued attribute associated with an object.

Applications: Drug response, stock prices.
Algorithms: Gradient boosting, nearest neighbors, random forest, ridge, and more...

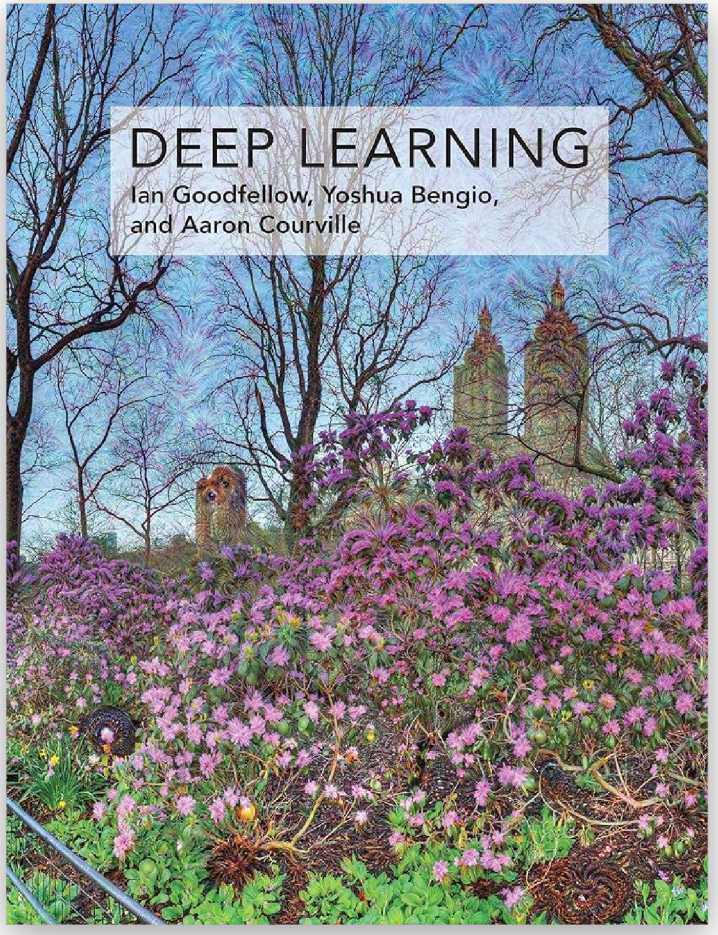
Clustering

Automatic grouping of similar objects into sets.

Applications: Customer segmentation, grouping experiment outcomes.
Algorithms: k-Means, HDBSCAN, hierarchical clustering, and more...

Lots of resources are also taken from the *Deep learning* book

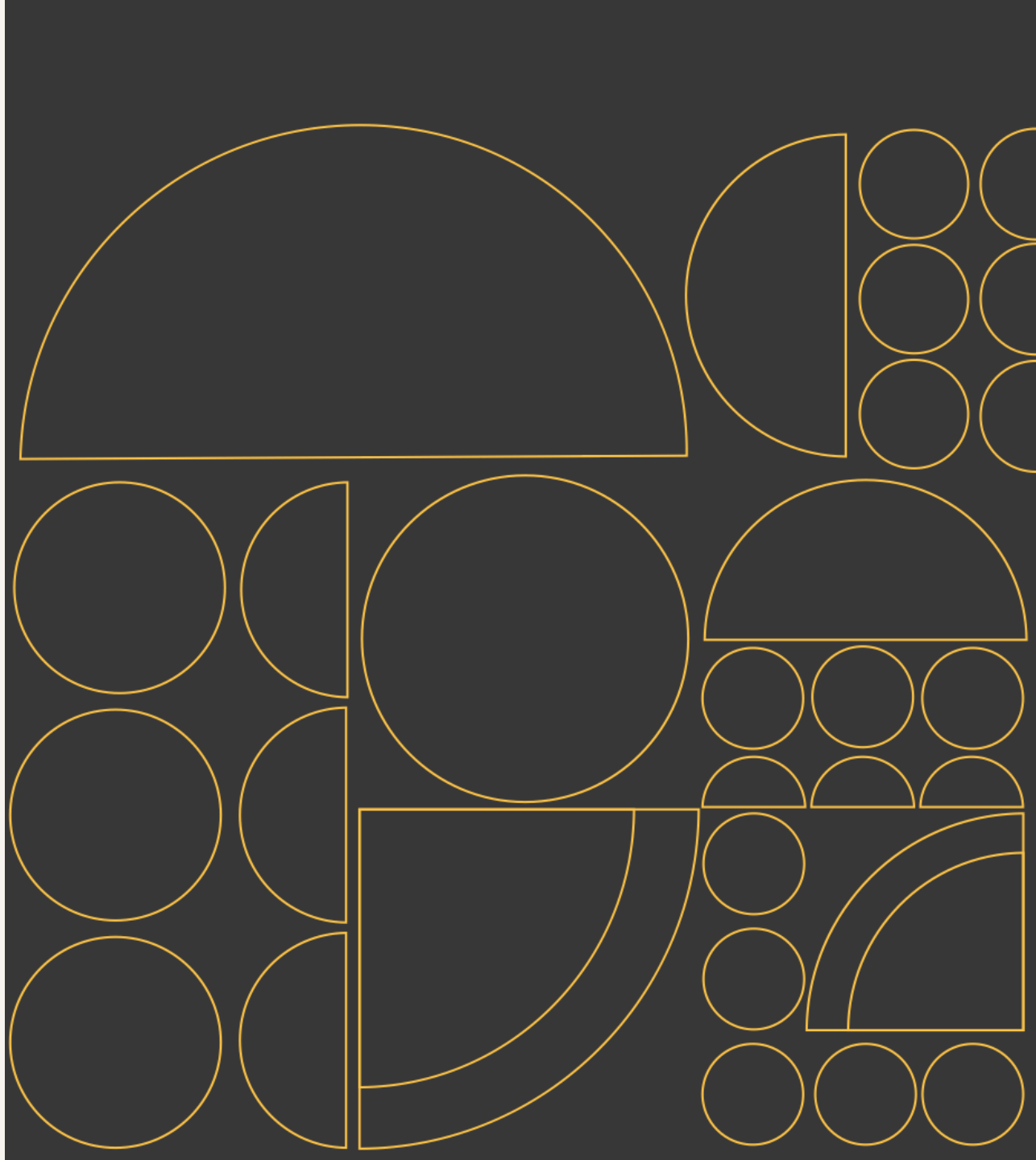
- Very complete introduction
- Historical aspects
- Starts from scratch (linear algebra)
- Covers machine/deep learning
- Illustrates with examples
- Online free access



ChEESE

1. Introduction

Machine learning for Earth science: Why, what, and how? Are any of those methods useful for your research? How to read papers that use machine learning?



How much time do you need to describe the following images?

Karpathy & Fei-Fei (2015)

How accurate are those descriptions?

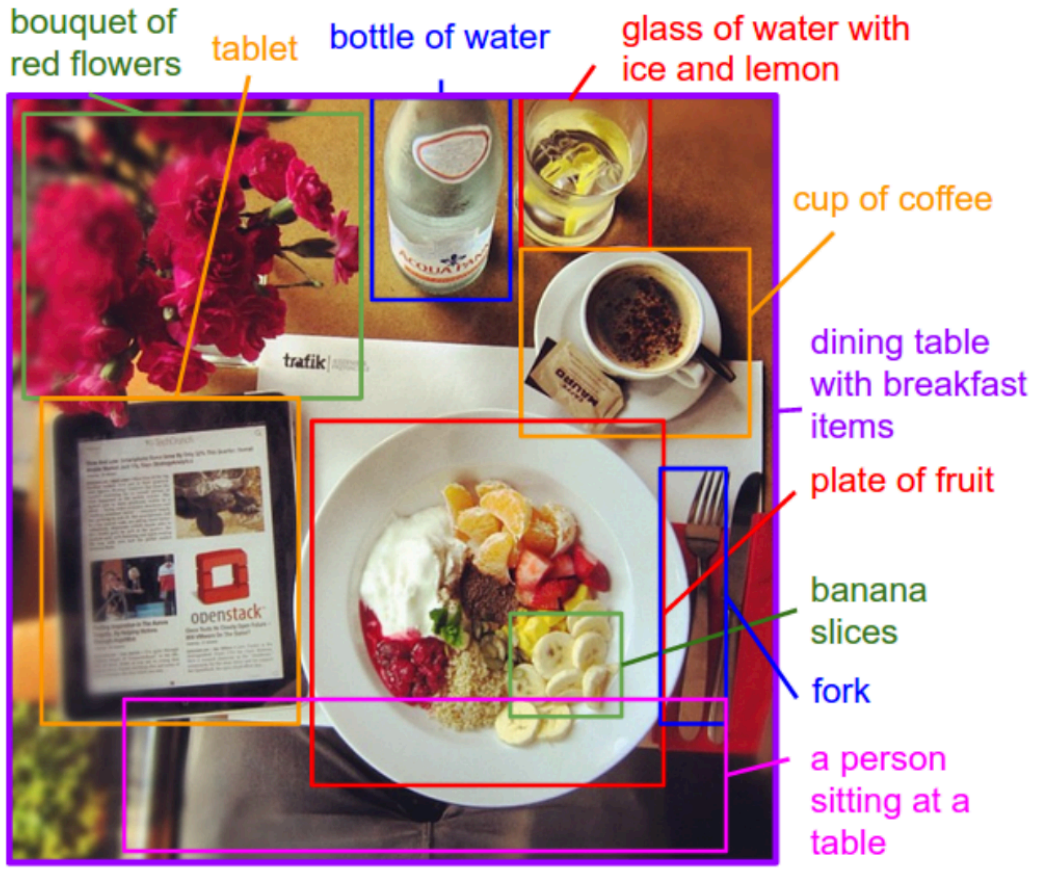
"man in black shirt is playing guitar."

"construction worker in orange safety vest is working on road."

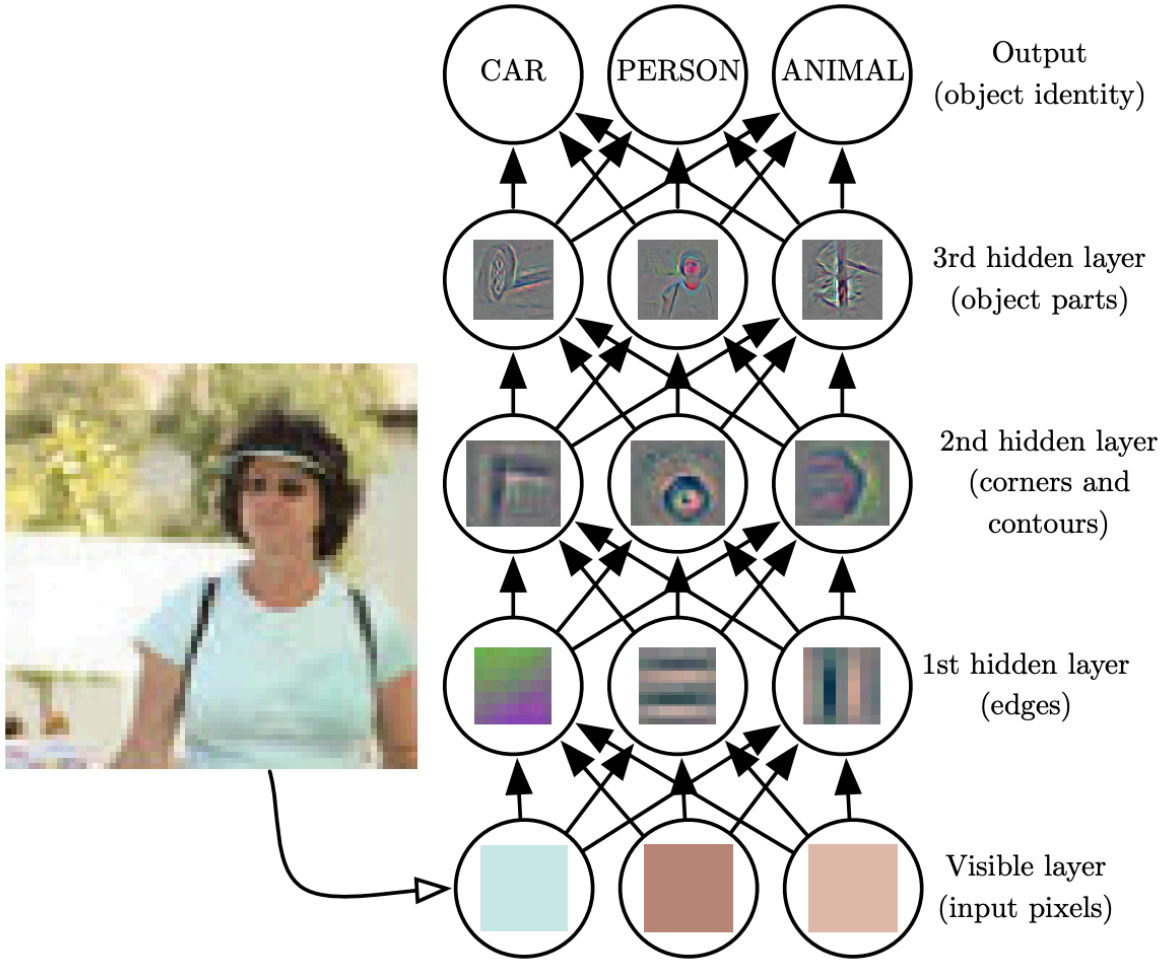
"two young girls are playing with lego toy."

How do we extract such high-level knowledge from data?

- Identify objects within image
- Recognize objects category
- Understand the link between objects
- Sort links by priority
- Generate text out of it

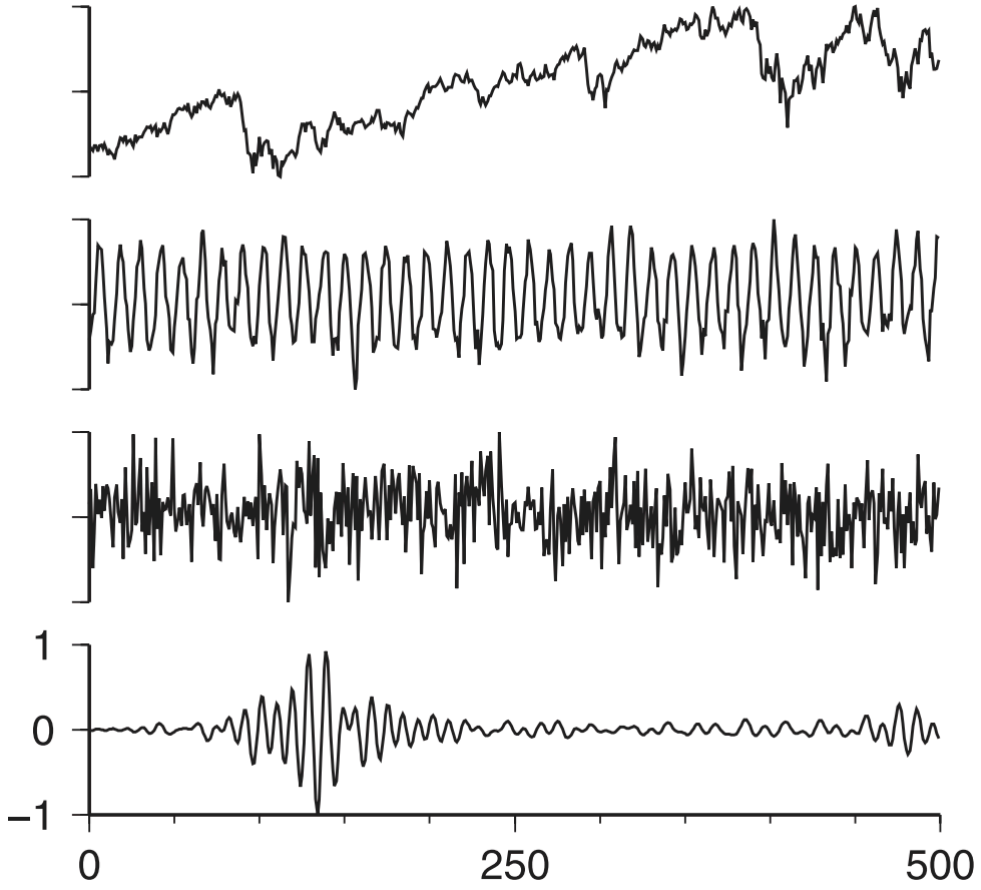


Ingredients: hierarchical knowledge extraction



Goodfellow et al. (2016)

Can you spot the seismogram?

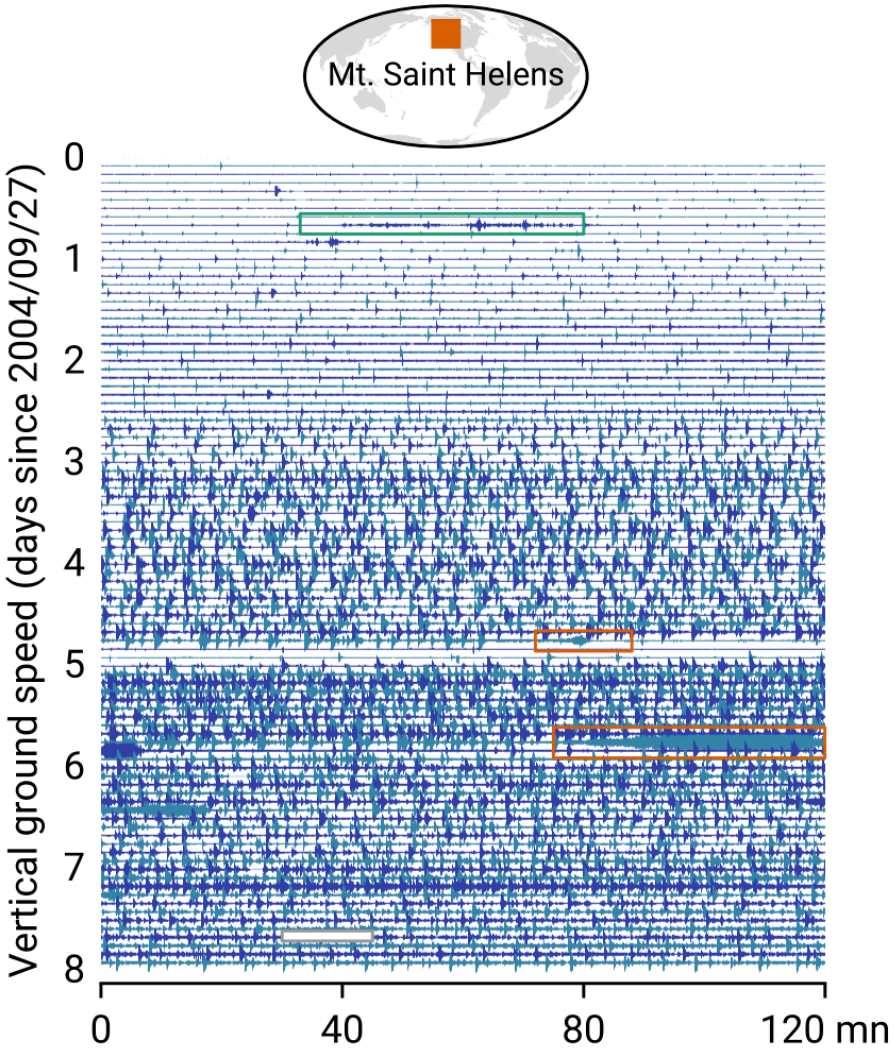


Valentine & Trampert (2012).

Top to bottom: UK stock exchange; Temperature in Central England; Gaussian noise; Long-period seismogram.

Detection and classification of events from seismograms

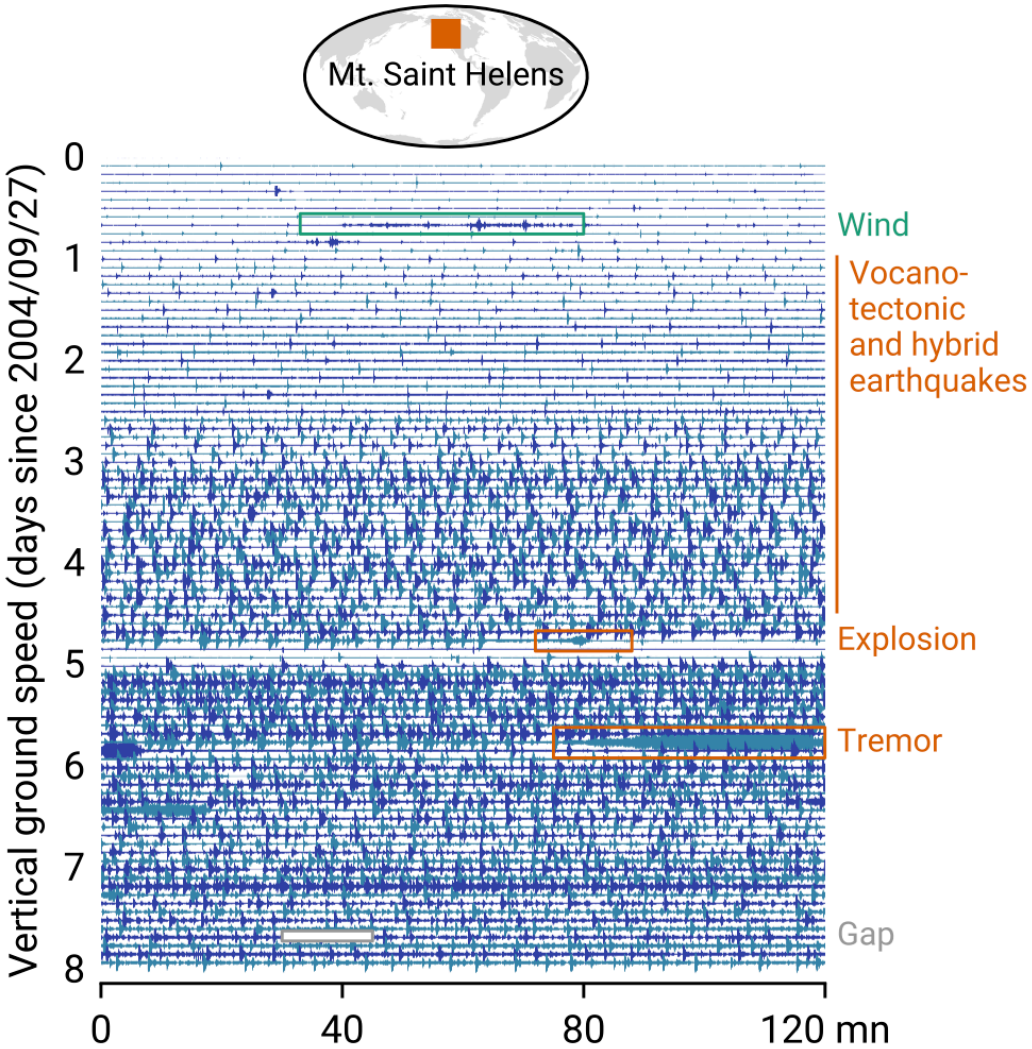
Most humans can pinpoint events.



Moran et al. (2008)

Detection and classification of events from seismograms

Most humans can pinpoint events.

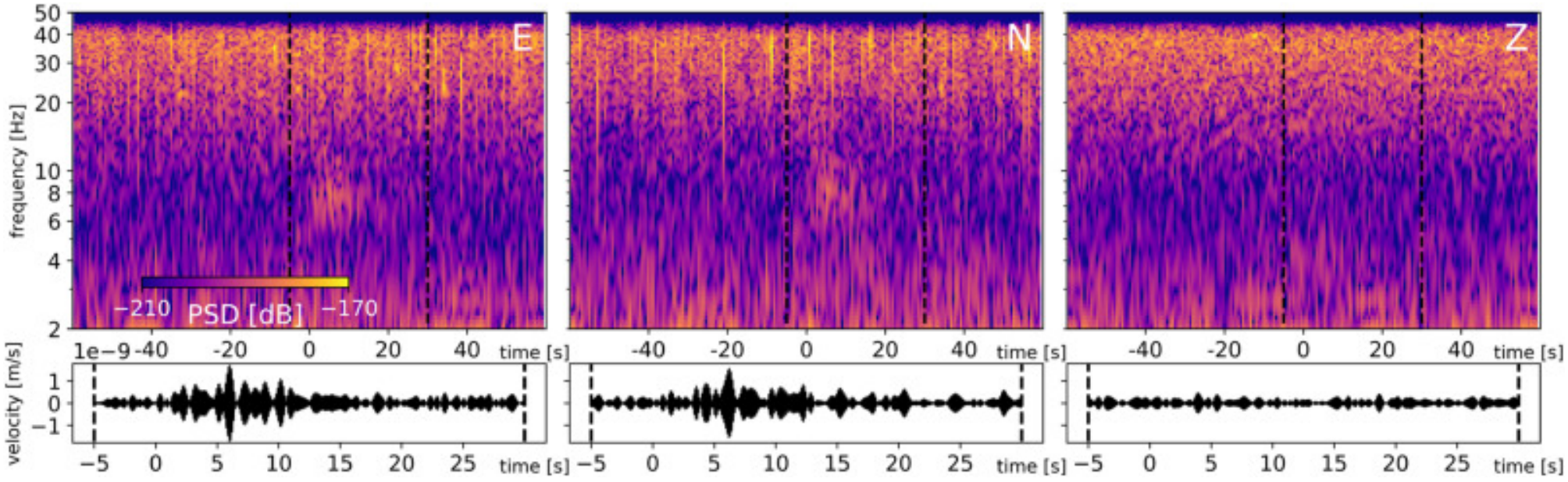


Experts can **classify** them.

Moran et al. (2008)

Diving into previously unseed data

Expert-detected marsquake within continuous insight data



Clinton et al. (2021)

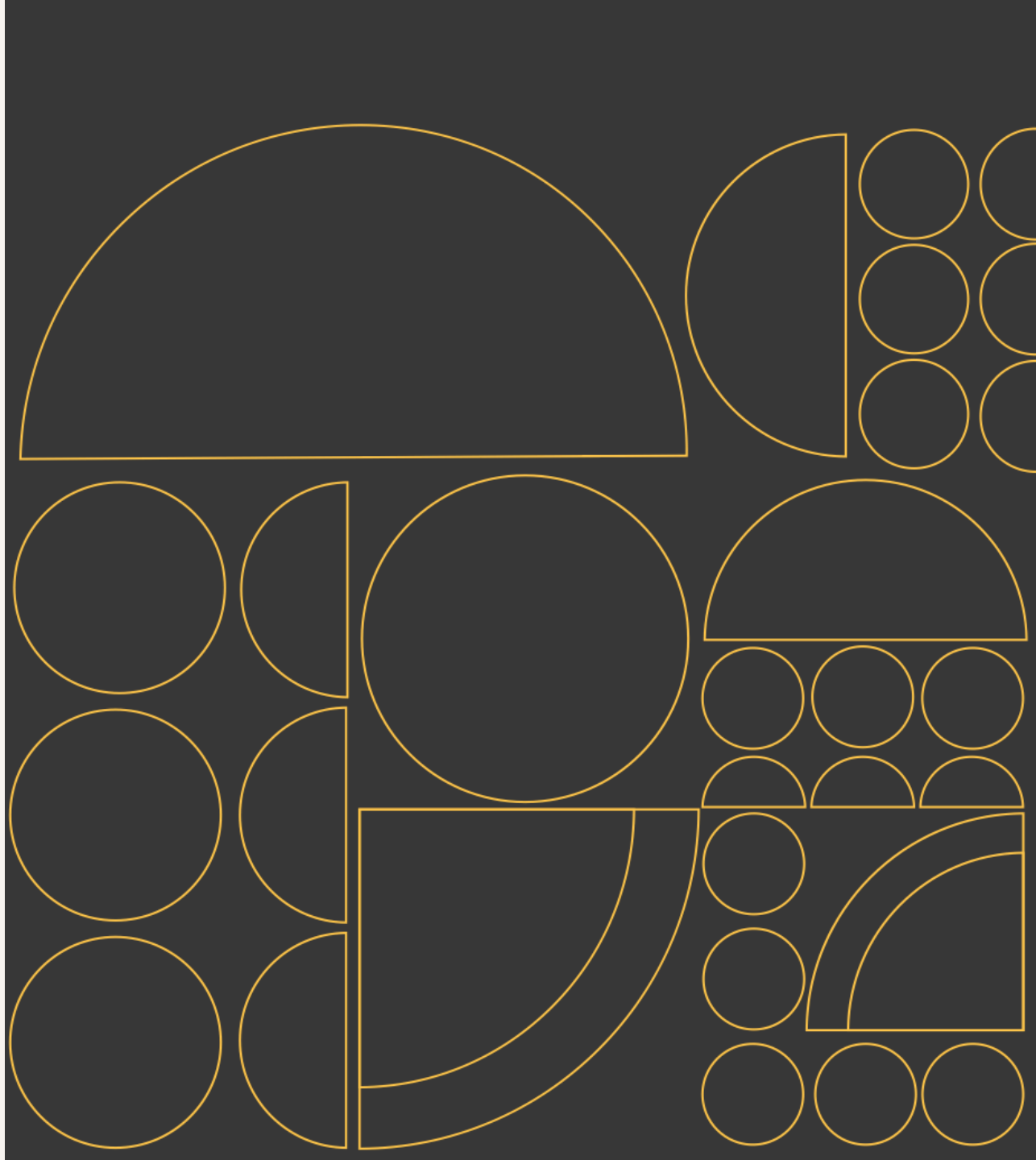
Target tasks of machine learning

- Time-consuming tasks
- Hard-to-describe tasks
- Exploration of new data

ChEESE

2. Definitions

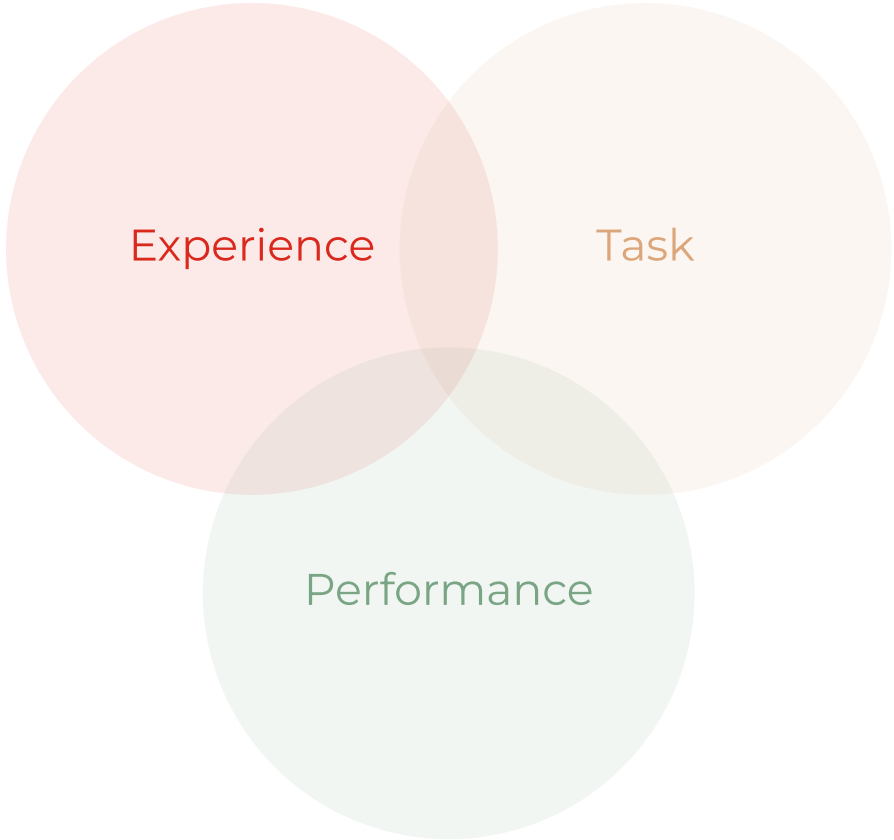
Machine learning is a field of study in artificial intelligence of statistical algorithms that can effectively generalize and thus perform tasks without explicit instructions.



General definition of machine learning

An algorithm learns from experience with respect to a task and performance, if its performance at solving the task improves with experience.

All three elements are required.



The data, the model, and the loss

🙊

the data

A set of samples $\mathbf{x}_i$ and labels $\mathbf{y}_i$ to learn from:

$$\mathcal{D} = \{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^{N}$$

🙉

the model

A parametric function f_θ that maps data $\mathbf{x}$ to $\hat{\mathbf{y}}$

$$f_\theta : \mathbf{x} \mapsto \hat{\mathbf{y}}$$

🙈

the loss

A measurement of the model performance

$$\mathcal{L}(\hat{\mathbf{y}}, \mathbf{y})$$

Learning = find the optimal parameters θ^* that minimize the loss $\mathcal{L}$ function

$$\theta^* = \underset{\theta}{\operatorname{argmin}}\, \mathcal{L}\Big(f_\theta(\mathbf{x}), \mathbf{y}\Big)$$

Useful vocabulary and symbols

Symbol	Name
$\{\mathbf{x}_i \in \mathbb{X}\}_{i=1\ldots N}$	Collection of **samples**
$\{\mathbf{y}_i \in \mathbb{Y}\}_{i=1\ldots N}$	Collection of **labels**
$\mathbf{x} = (x_1, \ldots, x_F)$	Set of sample **features**
$\mathbf{y} = (y_1, \ldots, y_T)$	Set of label **targets**
N	Dataset size
F	Feature space dimensions
T	Target space dimension
$\mathbb{X}$	Data space
$\mathbb{Y}$	Label space

An image is a sample $\mathbf{x}$ with

$$\mathbf{x} \in \mathbb{X} = \mathbb{R}^{H \times W \times C}$$

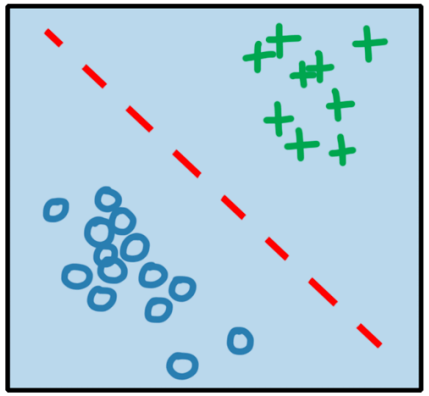
H is the height, W the width, and C the channels. The labels are a category y with

$$y \in \mathbb{Y} = \{0, 1, \ldots, K\}$$

with K the number of categories.
Note that y is scalar in this case.

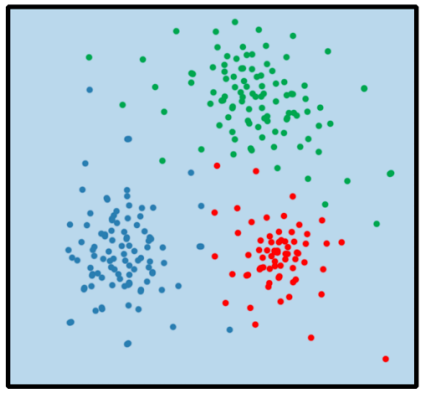
Main types of learning

supervised learning



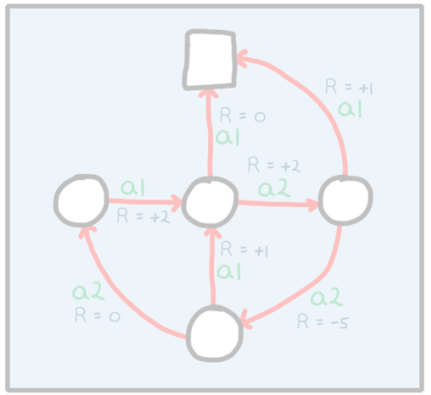
Predict $\mathbf{y}$ from $\mathbf{x}$ (regression, classification).

unsupervised learning



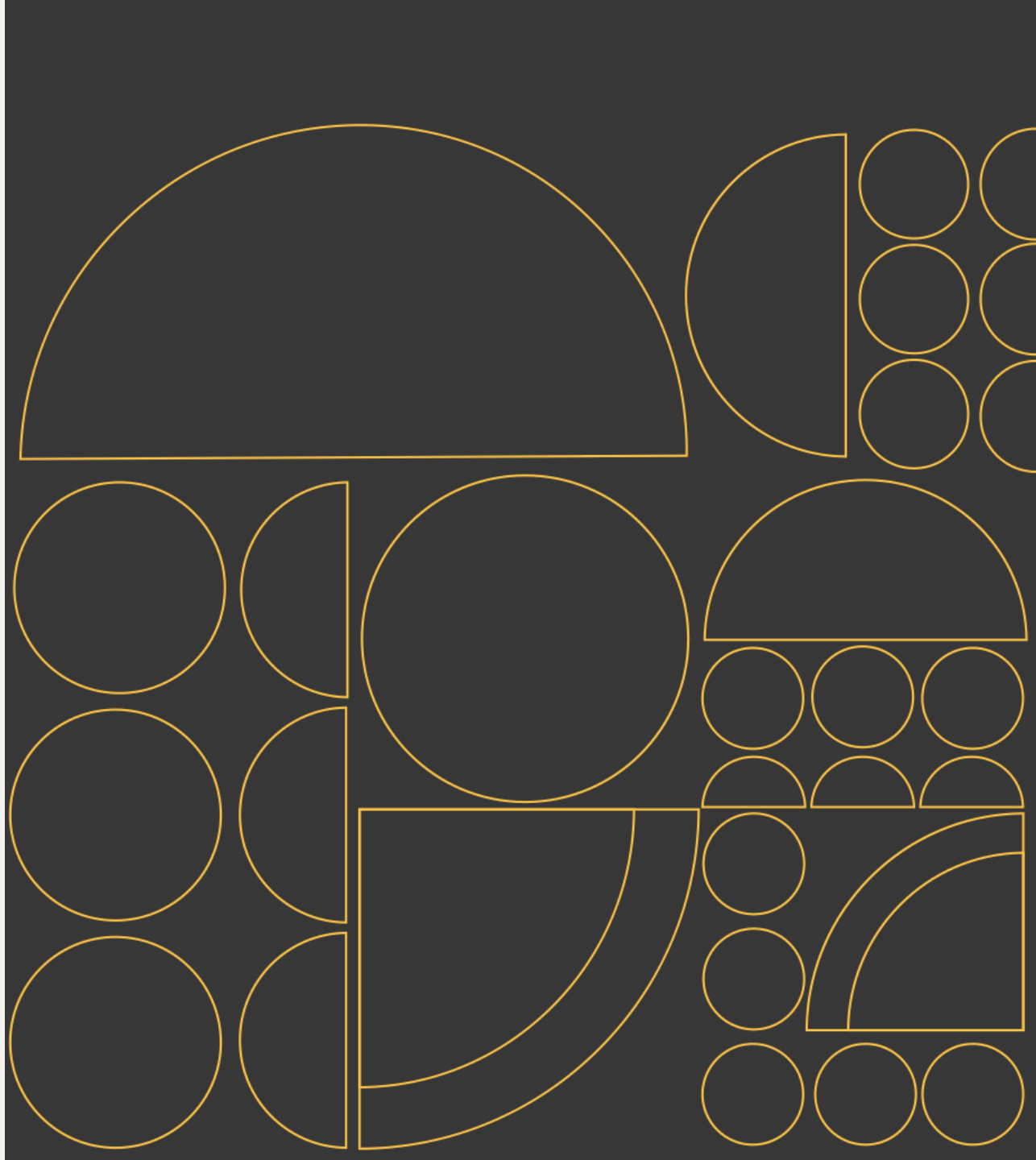
Learn some distribution $p(\mathbf{x})$ (clustering, reduction).

reinforcement learning



Learn a policy to maximize a reward (gaming, robotics).

from mathworks.com



ChEESE

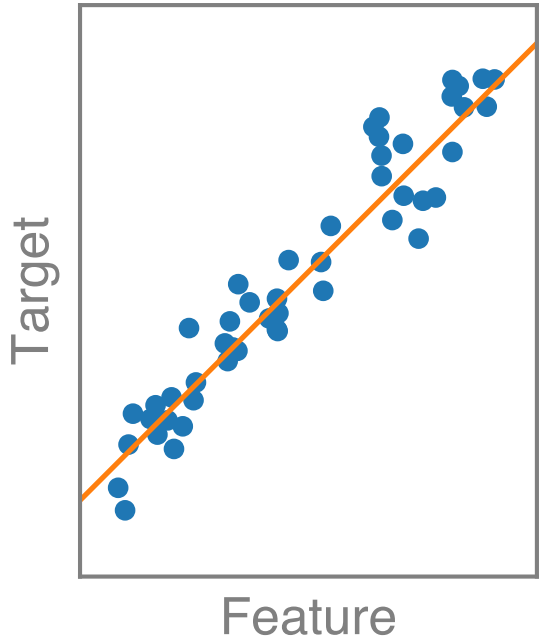
3. Regression

How to solve a regression or classification task with machine learning?

The two main tasks of supervised learning

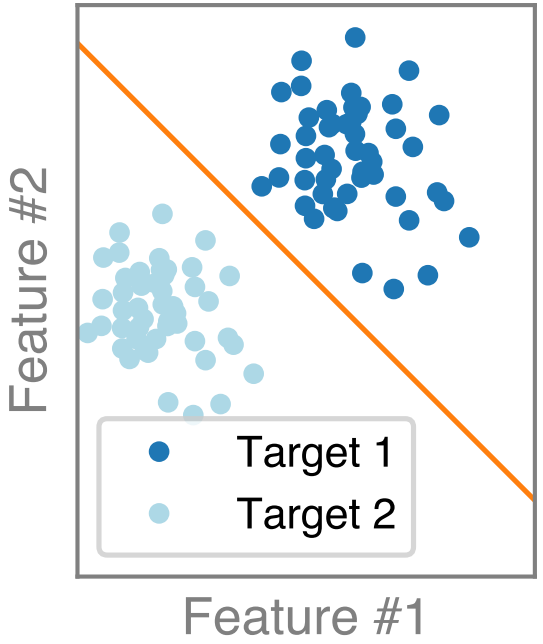
Regression

x and y are continuous



Classification

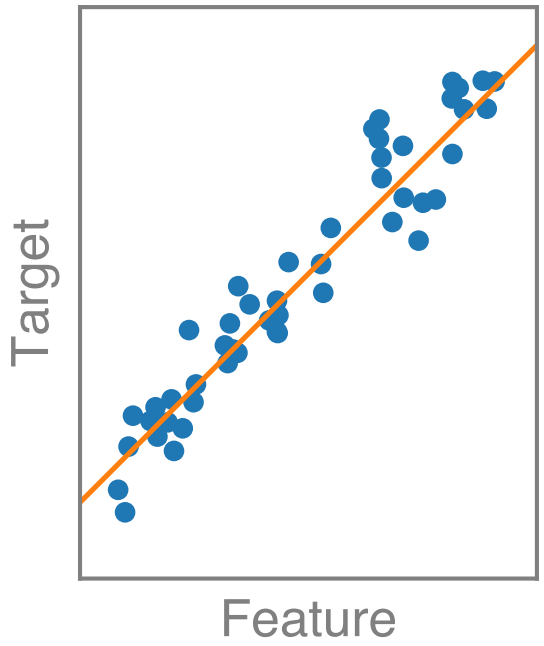
x is continuous and y is descrete



The two main tasks of supervised learning

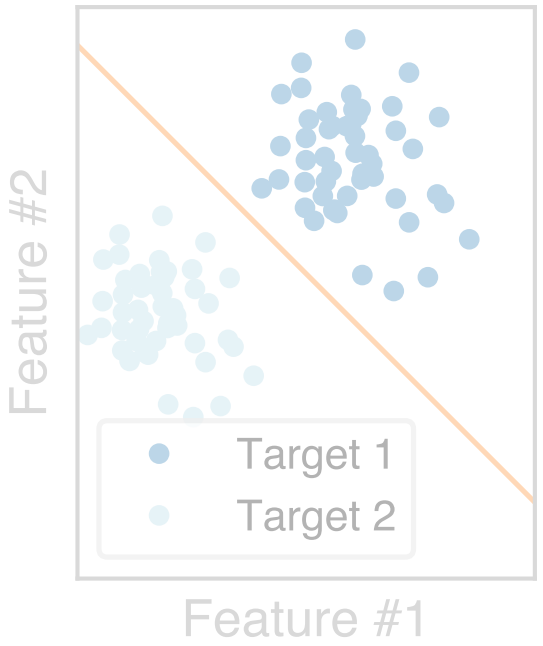
Regression

x and y are continuous



Classification

x is continuous and y is descrete



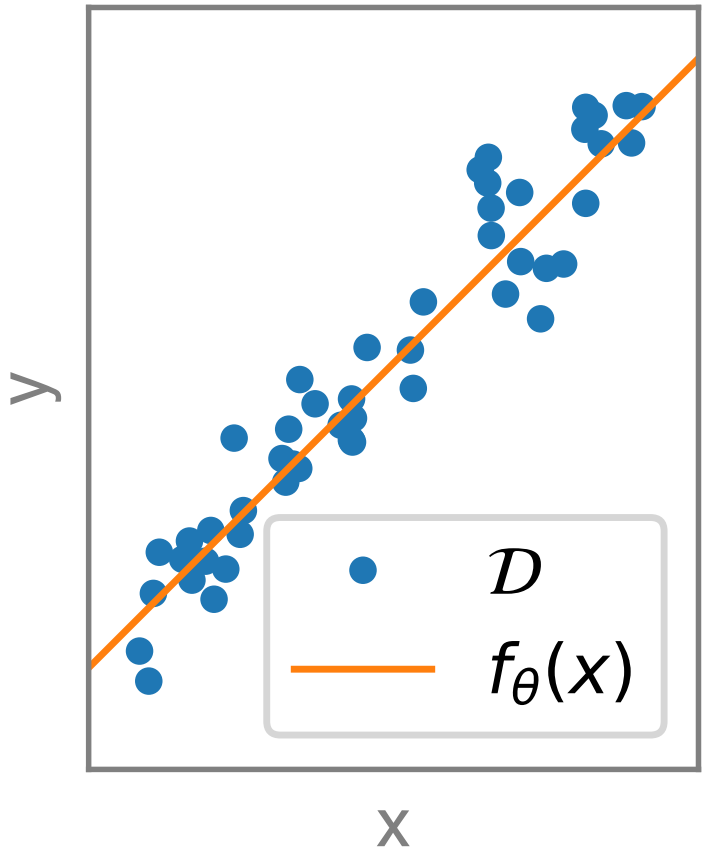
The regression task

Dataset: set of N samples x_i and corresponding labels y_i such as

$$\mathcal{D} = \{(x_i, y_i)\}_{i=1}^{N}$$

Regression: optimize the parameters θ of a function f_θ to predict y from x. Find the optimal parameters θ^* that minimize $\mathcal{L}$, such as

$$\theta^* = \underset{\theta}{\operatorname{argmin}} \mathcal{L}\Big(f_\theta(x), y\Big).$$



The linear regression

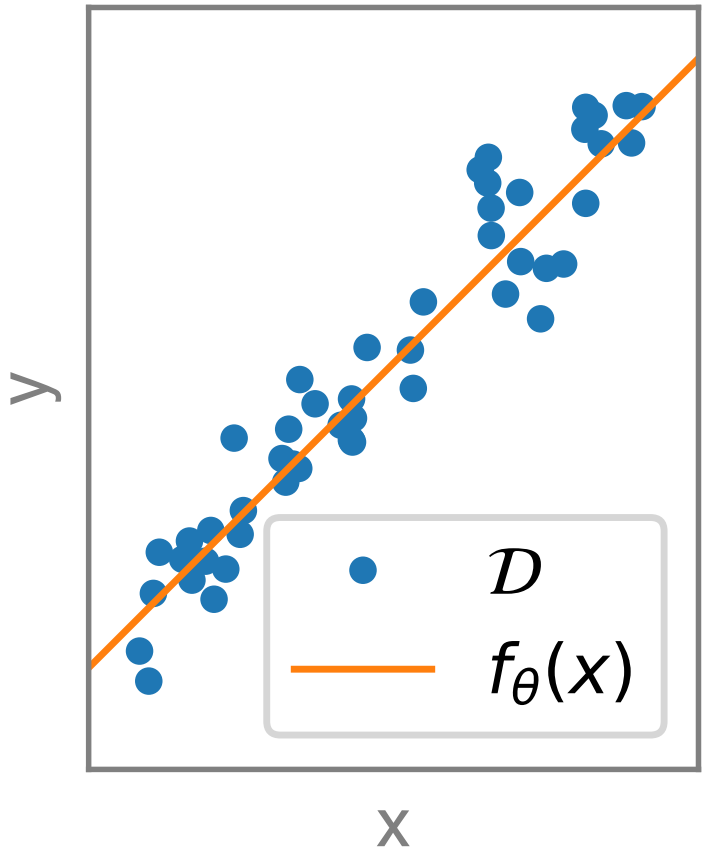
Linear model: coefficients $\theta = (a, b) \in \mathbb{R}^2$ that map x to y with

$$f_\theta : x \mapsto y = ax + b.$$

Loss function: mean squared error (example), given by

$$\mathcal{L}(\theta) = \frac{1}{N} \sum_{i=1}^{N} \left(f_\theta(x_i) - y_i \right)^2 .$$

How do we minimize the loss?



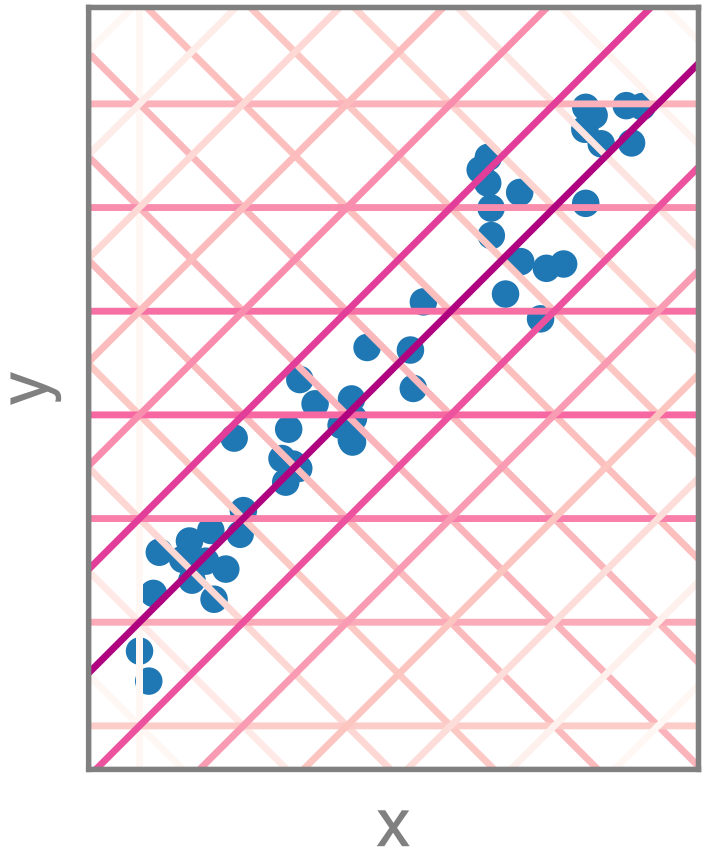
Naive attempt: grid search

Grid search: find θ^* among tested values of θ.

Pros: easy to implement, exhaustive search, uncertainty estimation.

Cons: unscalable. If 0.1s / evaluation, then 2 parameters with 100 values each takes 1/4 hour. *For 5 parameters it takes more than 30 years!*

Any smarter idea?

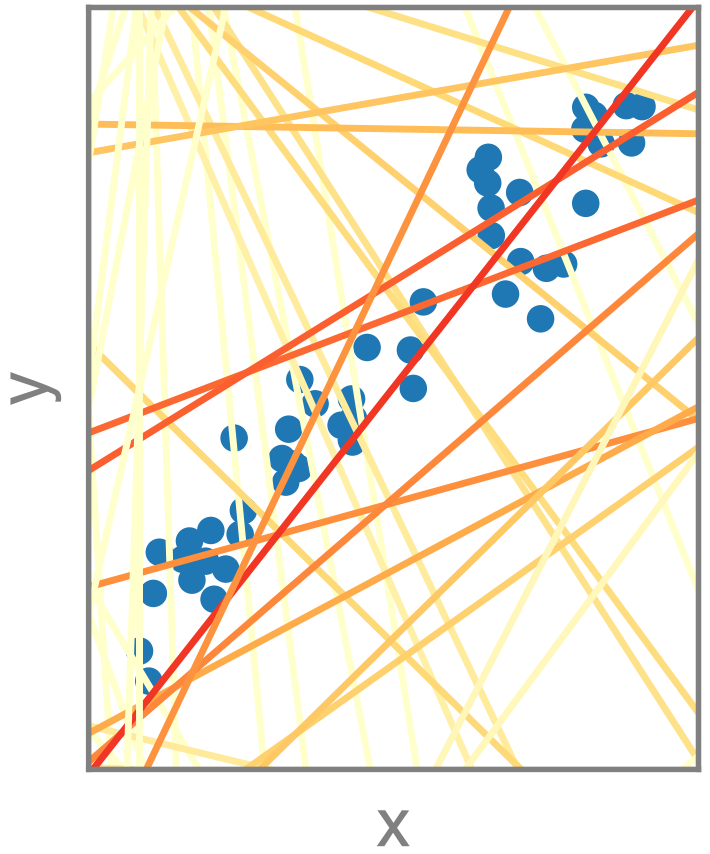


Random search

Random search to find θ^*.

Pros: easy to implement, scalable, uncertainty estimation, can include prior knowledge.

Cons: not exhaustive, can be slow to converge.

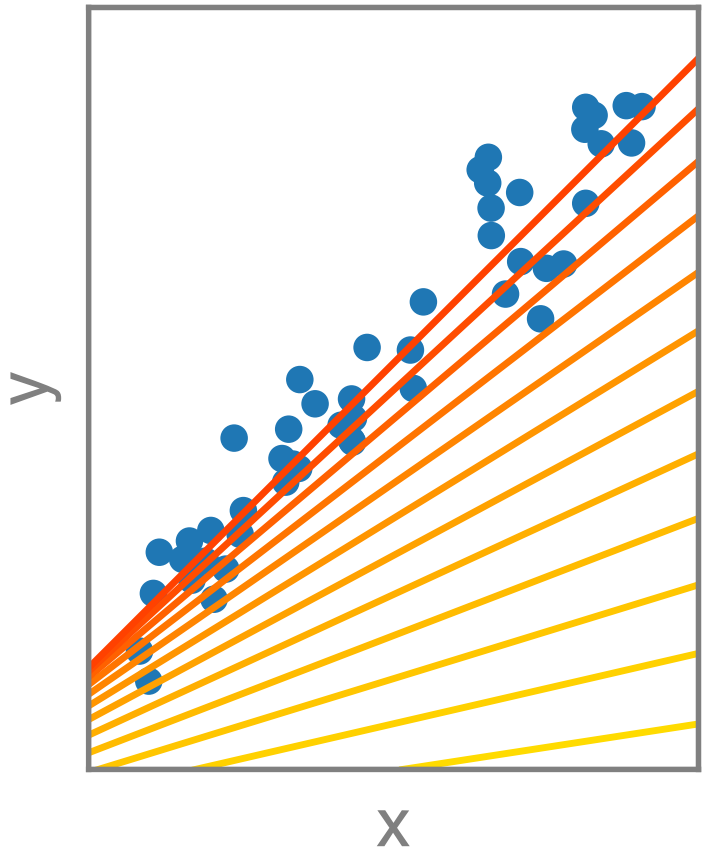


Gradient descent

Idea: estimate the gradient of $\mathcal{L}$ w.r.t. the parameters θ, update the parameters towards gradient descent.

Pros: converges faster than random search.

Cons: gets stuck in local minima, slow to converge, needs differentiability.



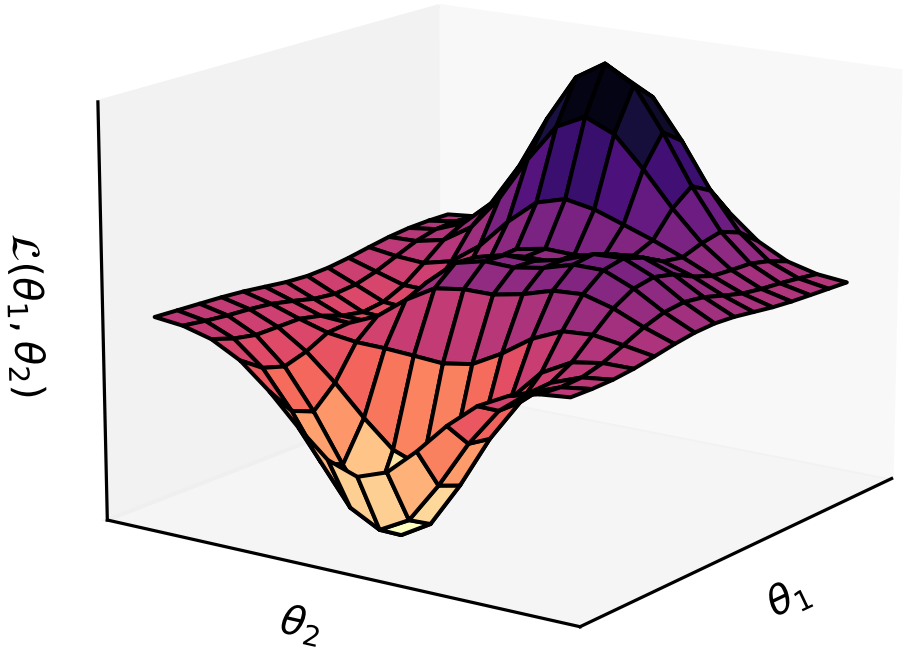
Gradient descent

Recipe

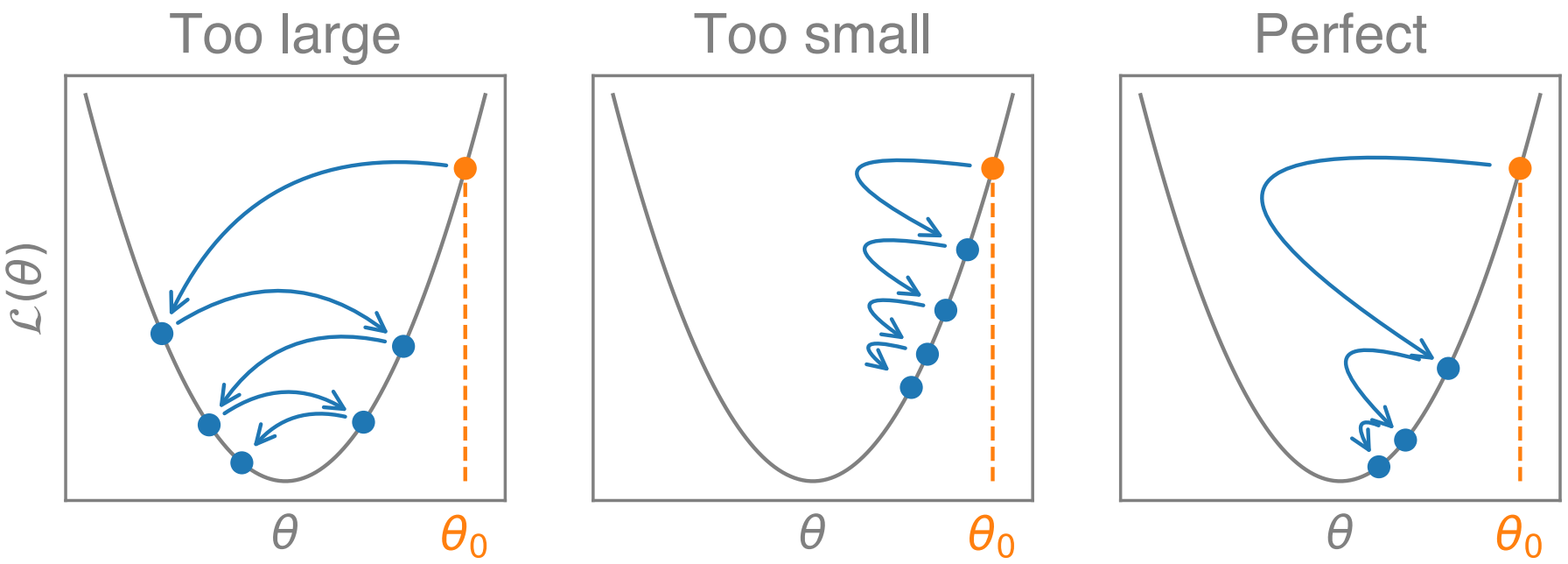
1. Initial model $\theta = (a_0, b_0)$
2. Compute the gradient $\nabla \mathcal{L}(\theta)$
3. Update the model $\theta \leftarrow \theta - \eta \nabla \mathcal{L}(\theta)$
4. Repeat until convergence

Hyperparameters

The **learning rate** η is the update step.



How to deal with learning rate?



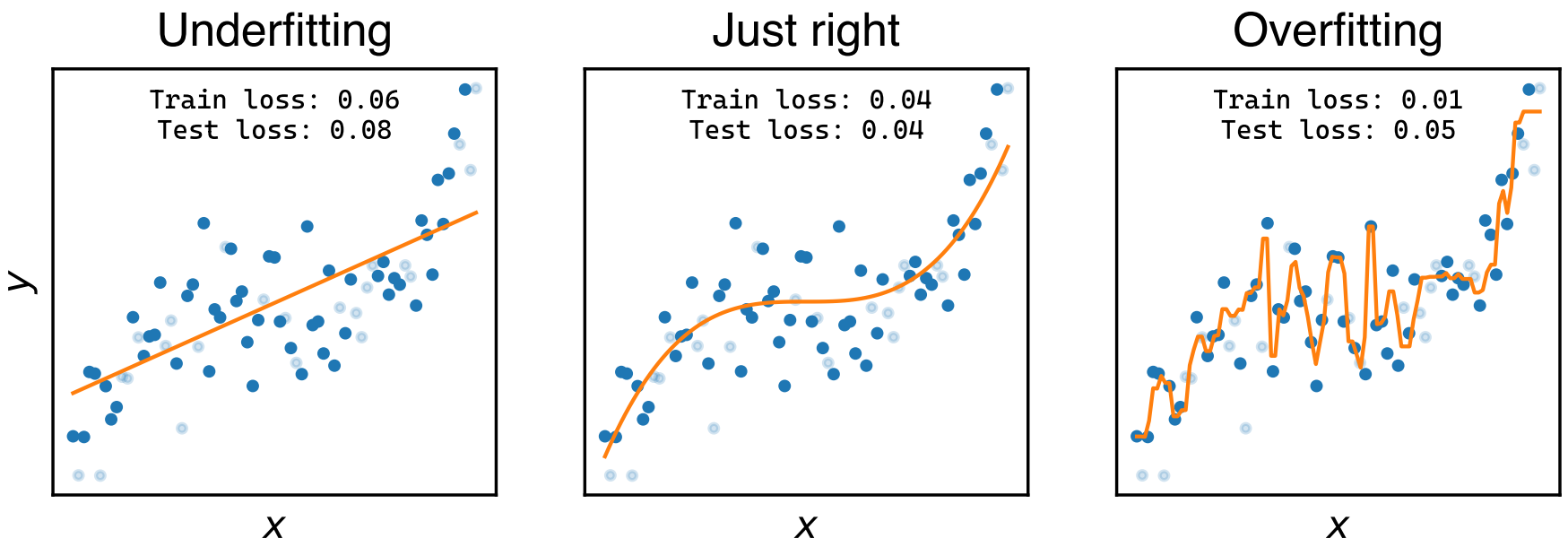
That's part of the **hyperparameters tuning**.

More about that in the deep learning lectures.

The problem of overfitting

Having a loss close to 0 does not mean that the model **generalizes** well.

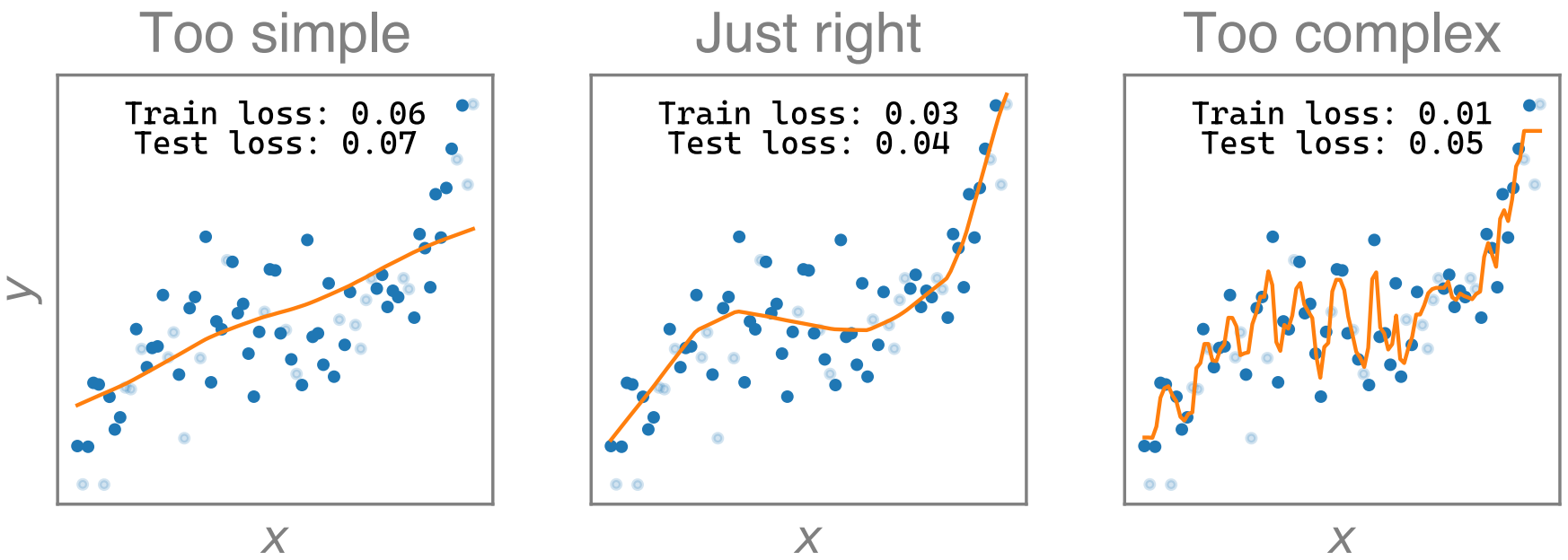
Key concepts to prevent overfitting: split the dataset



By splitting the dataset into a **training** and a **testing** set,
we evaluate the performance on unseen (but **similar**) data.

Key concepts to prevent overfitting: regularization

Add a penalty term $\mathcal{R}$ to the loss $\mathcal{L}_{\mathcal{R}} = \mathcal{L} + \lambda\mathcal{R}$, with λ the regularization strength

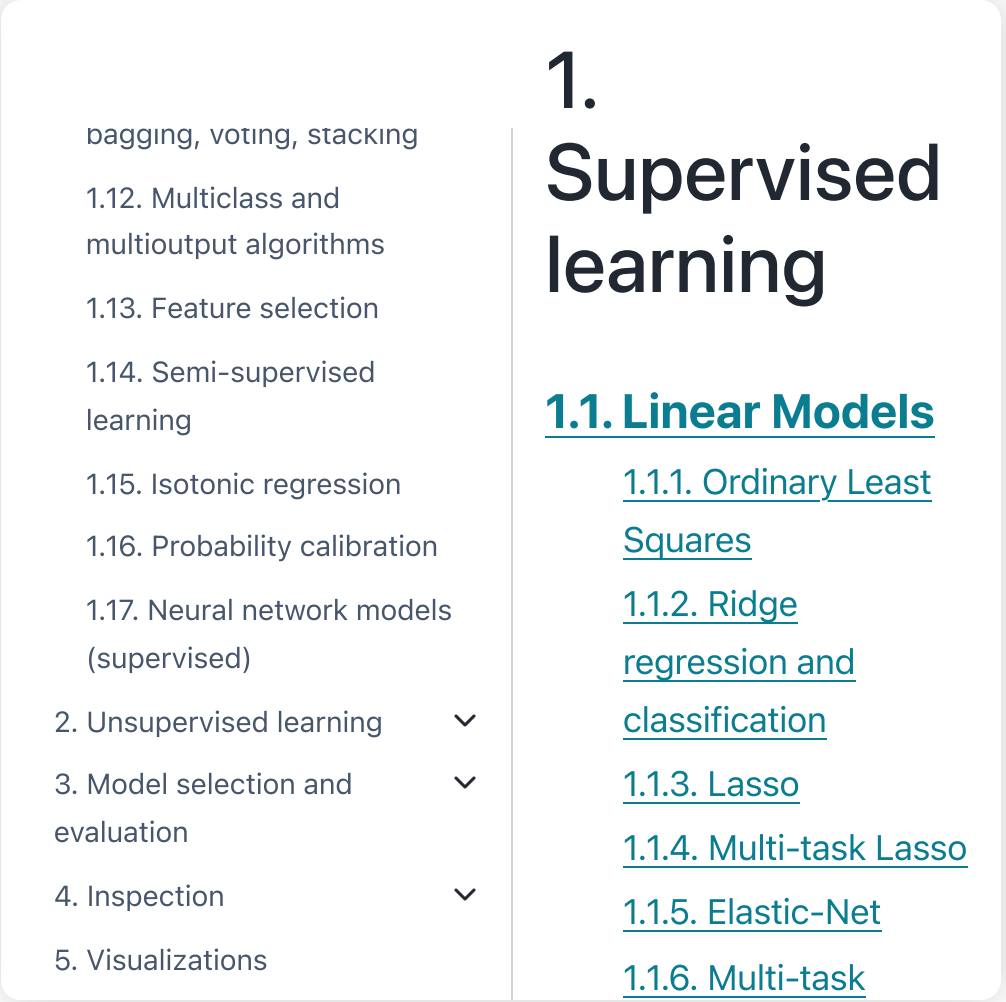


The regularization penalizes the model's complexity.

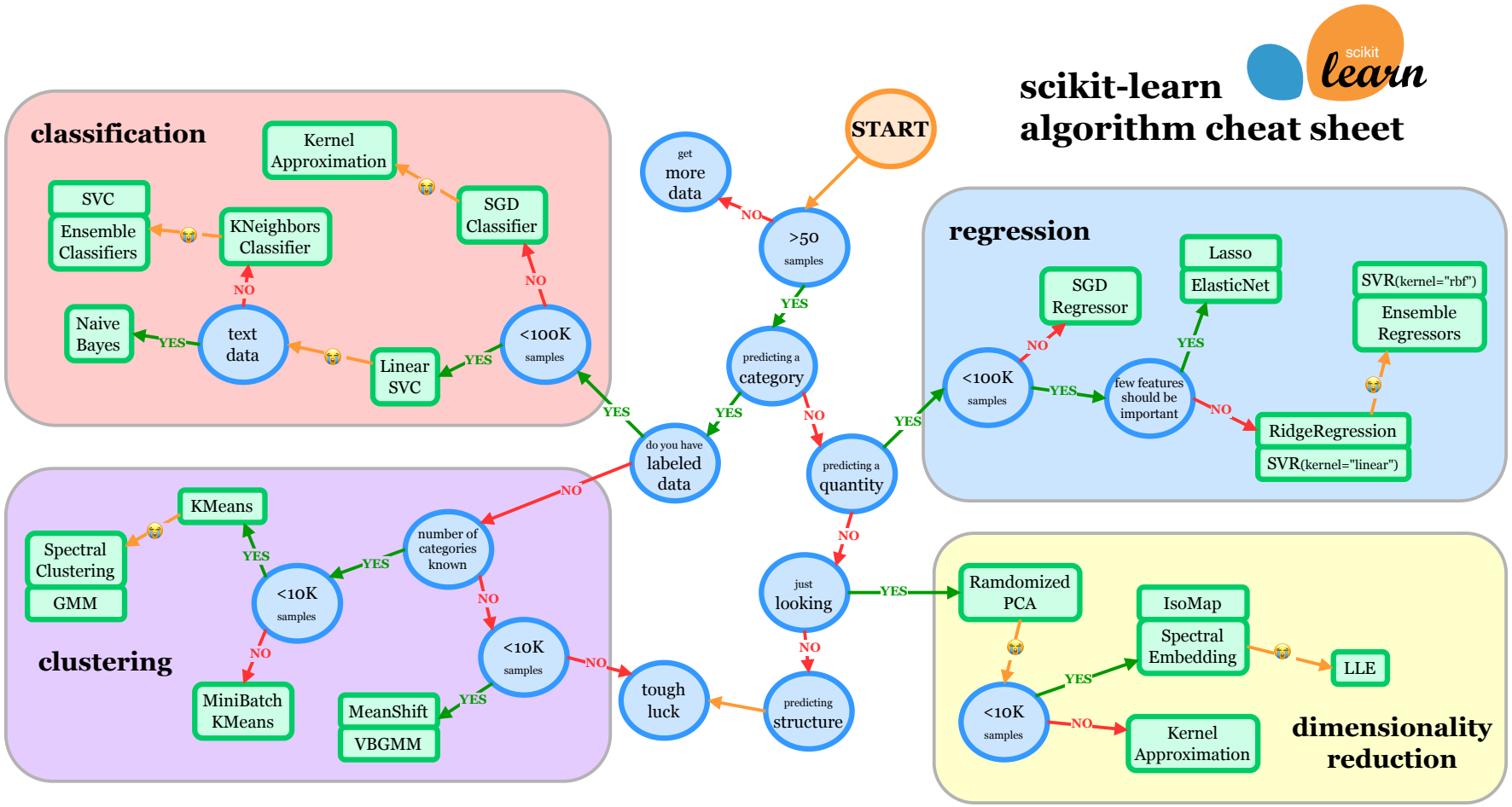
Why so many regression algorithms?

Because of combination of models, losses, and regularizations. The scikit-learn.org website provides a unified interface in a `greybox style`.

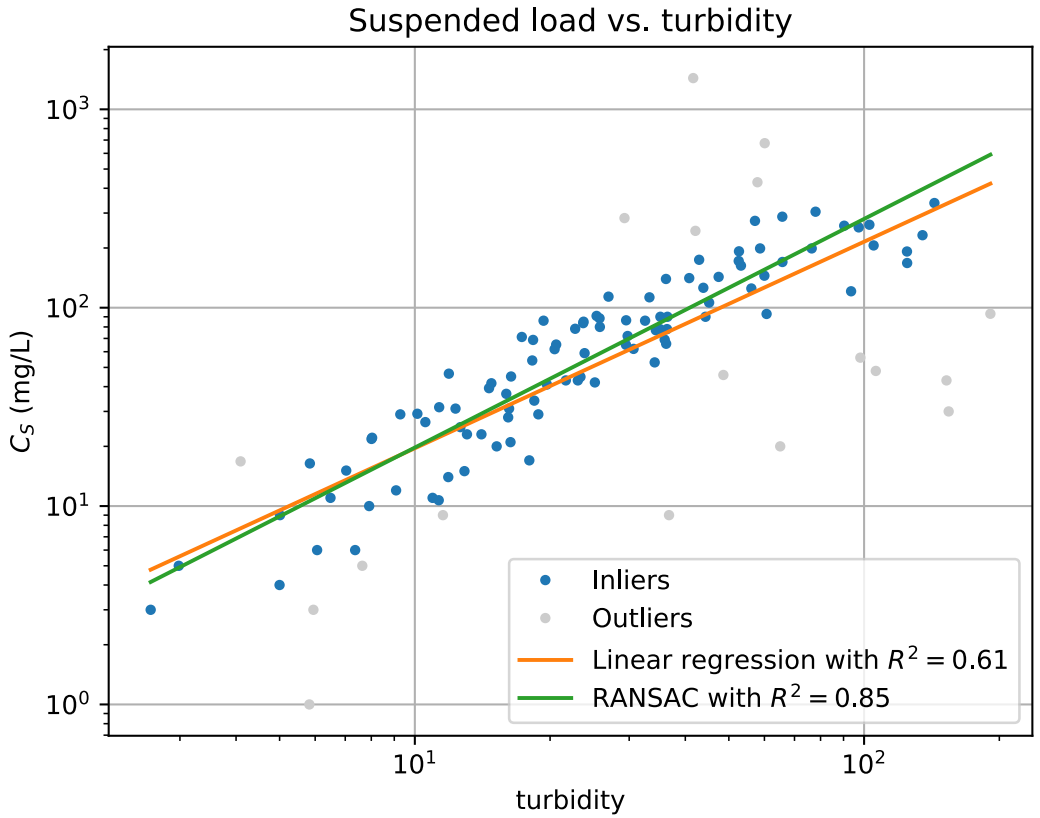
The model selection is made by experience or **trial and error**.



Guidelines for exploring relevant models

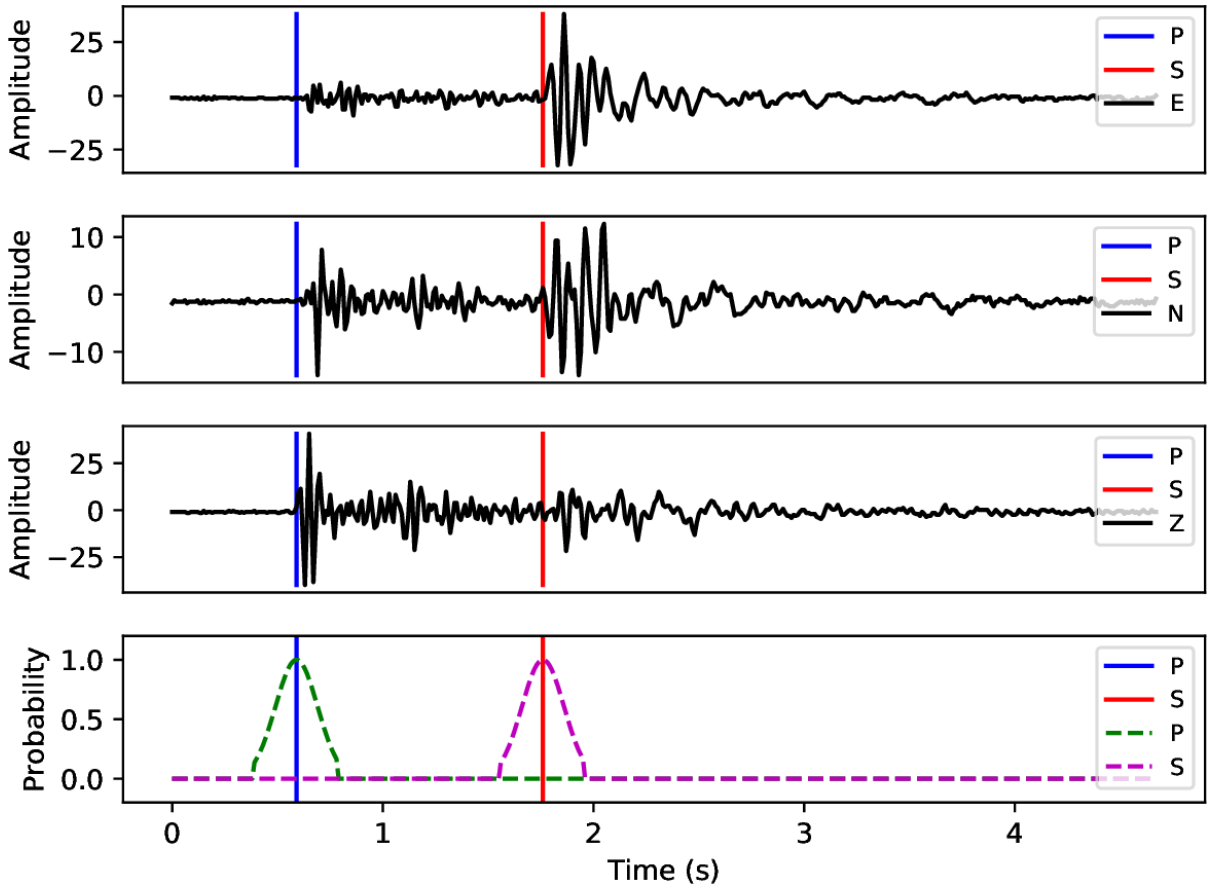


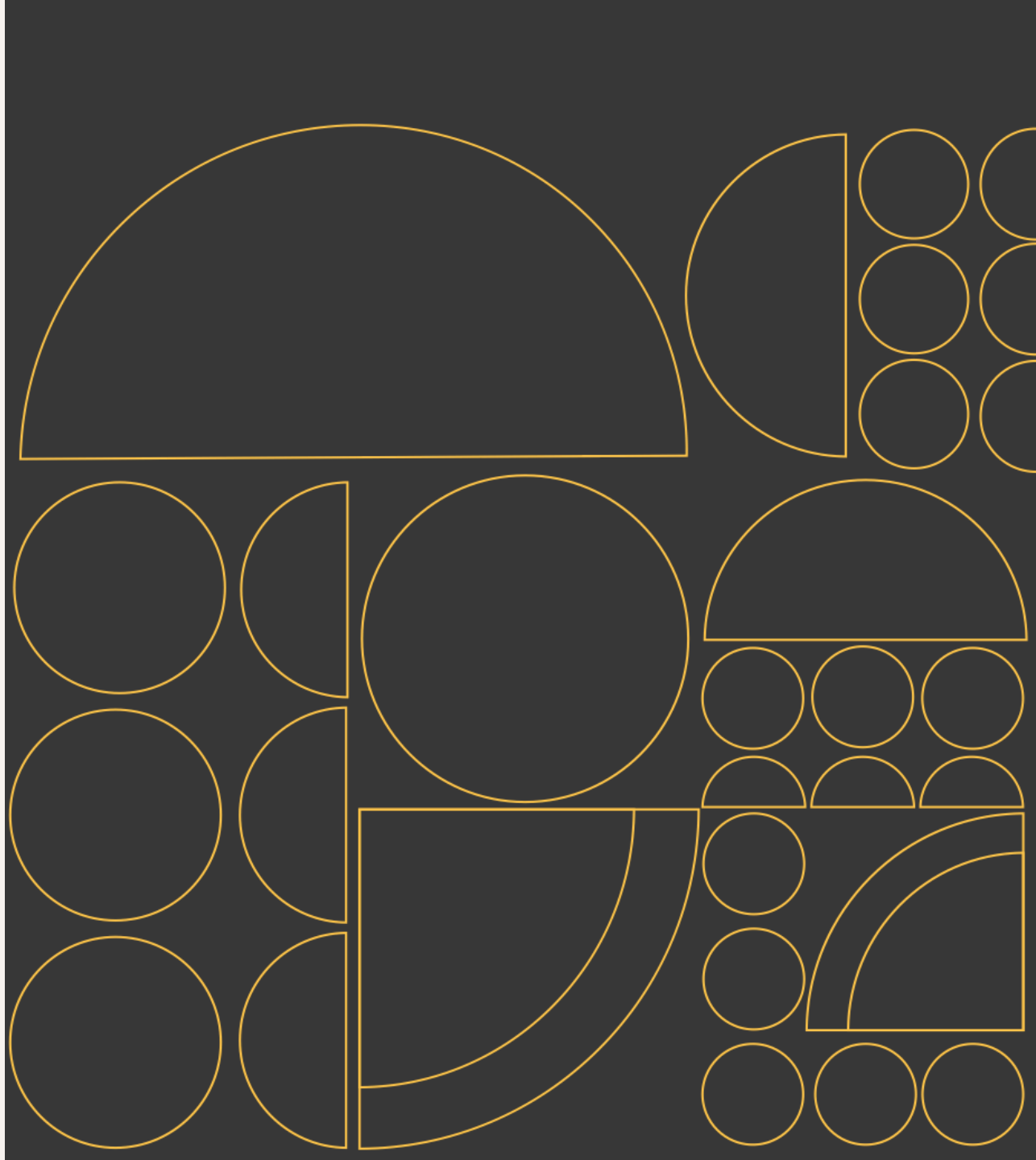
Notebook: expansive data from cheap sensors



Find out *P* and *S* waves within continuous seismograms

How do you addess this regression problem?





ChEESE

4. Supervised machine learning: classification

How to solve a regression or classification task with machine learning?

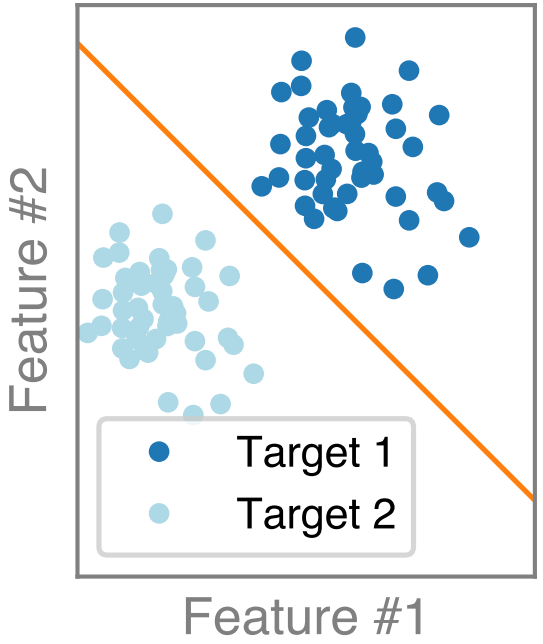
The two main tasks of supervised machine learning

Regression

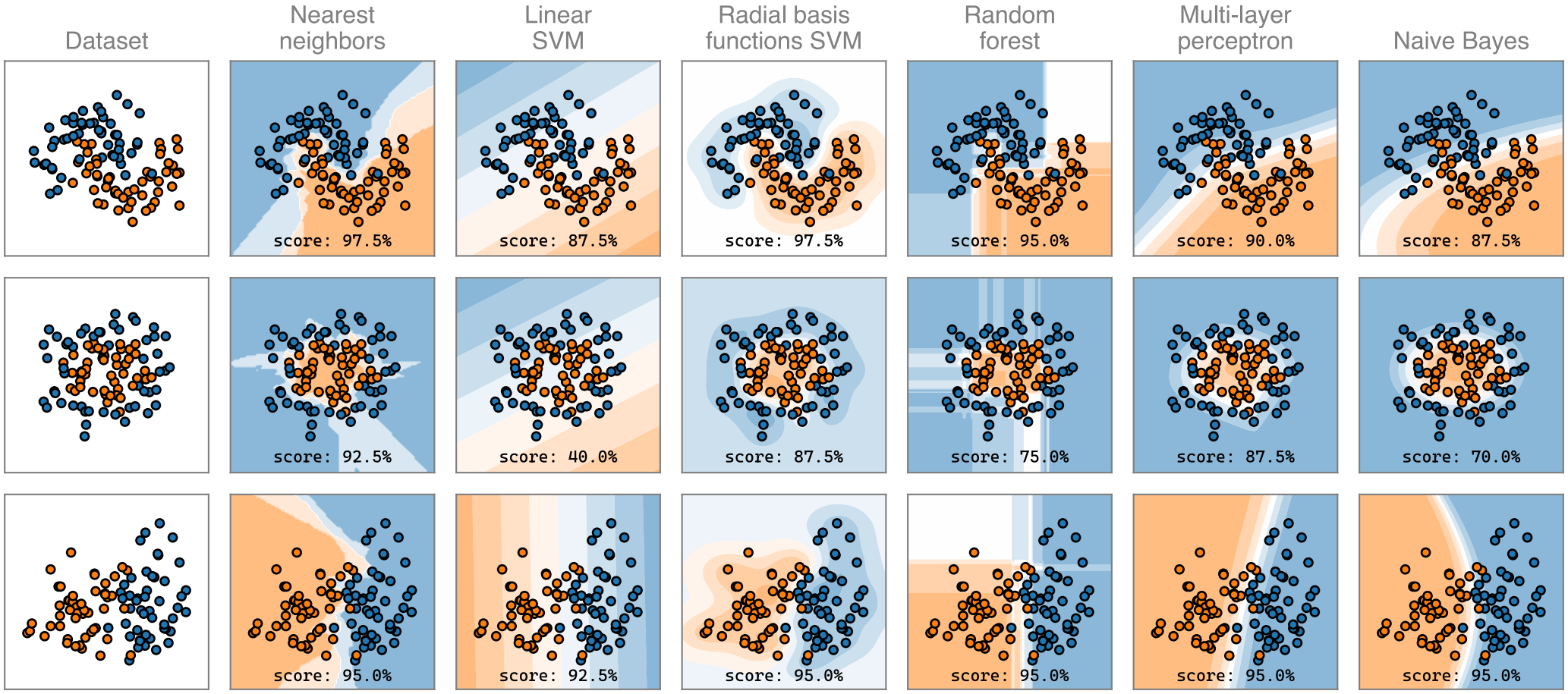
x and y are continuous

Classification

x is continuous and y is descrete



The classification task



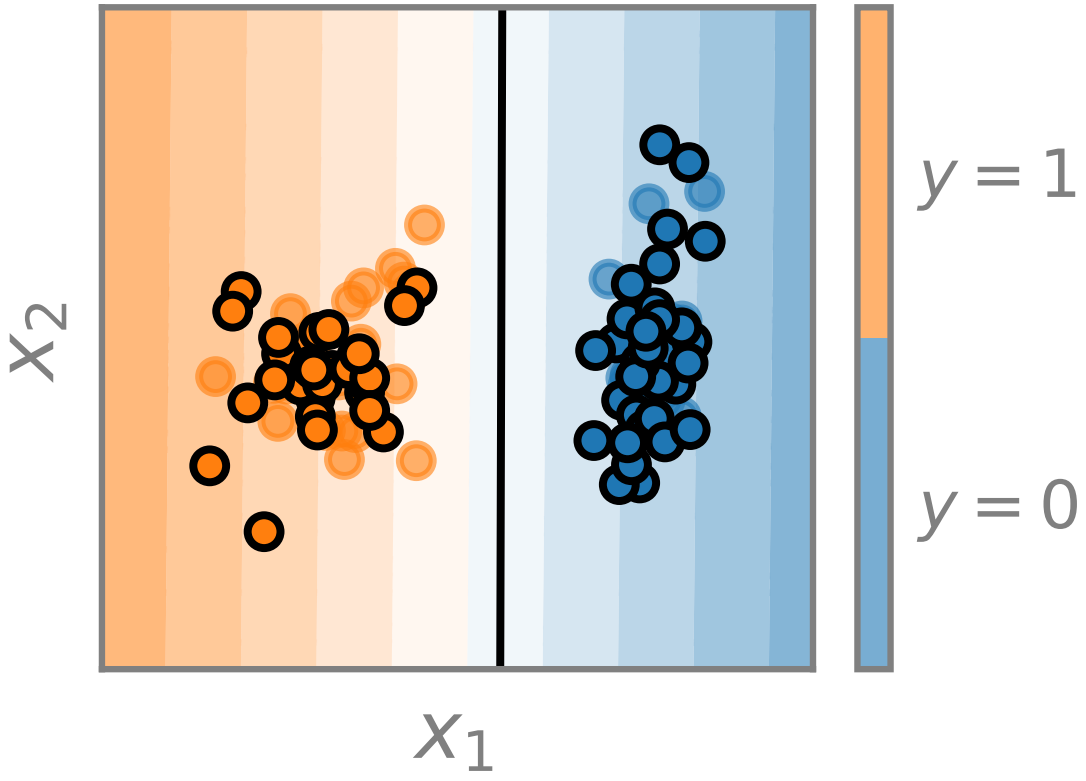
Here again, we have many possibilities.

The classification task

Experience: labels $y \in \{0, 1\}$ for two features $\mathbf{x} \in \mathbb{R}^2$.

Task: predict $\hat{y}$ of each sample $\mathbf{x}$.

Performance: how should we measure the performance of a classifier?

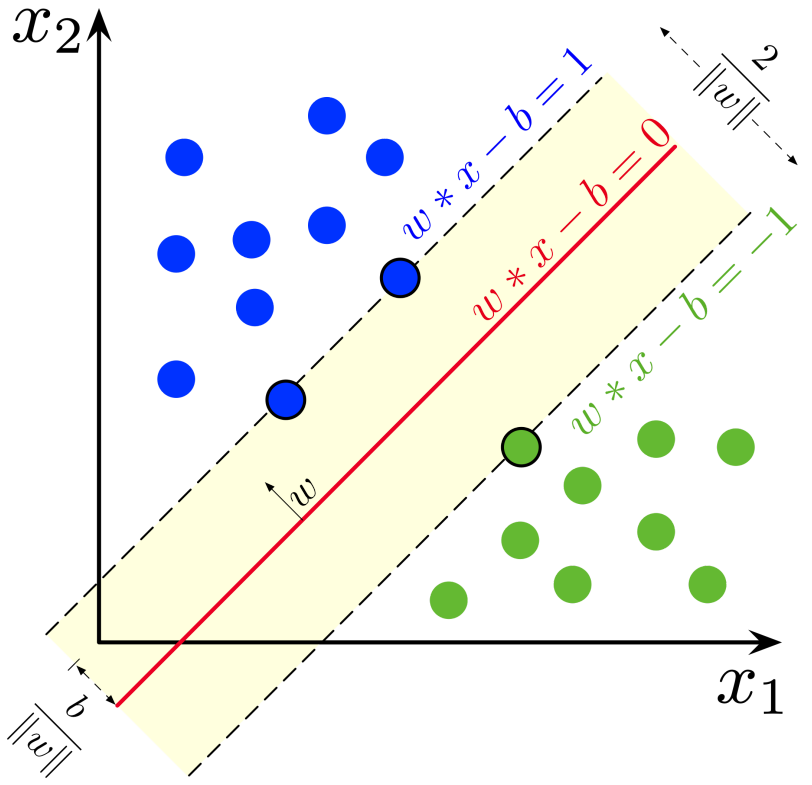


The classification task with support vector machines (SVM)

Support vector machines search the hyperplane of normal vector $\mathbf{w}$ and bias b that split the classes.

> Note: in 2D, a hyperplane is a line.

The support vectors are the samples that are closest to the other class.



https://en.wikipedia.org/wiki/Support_vector_machine

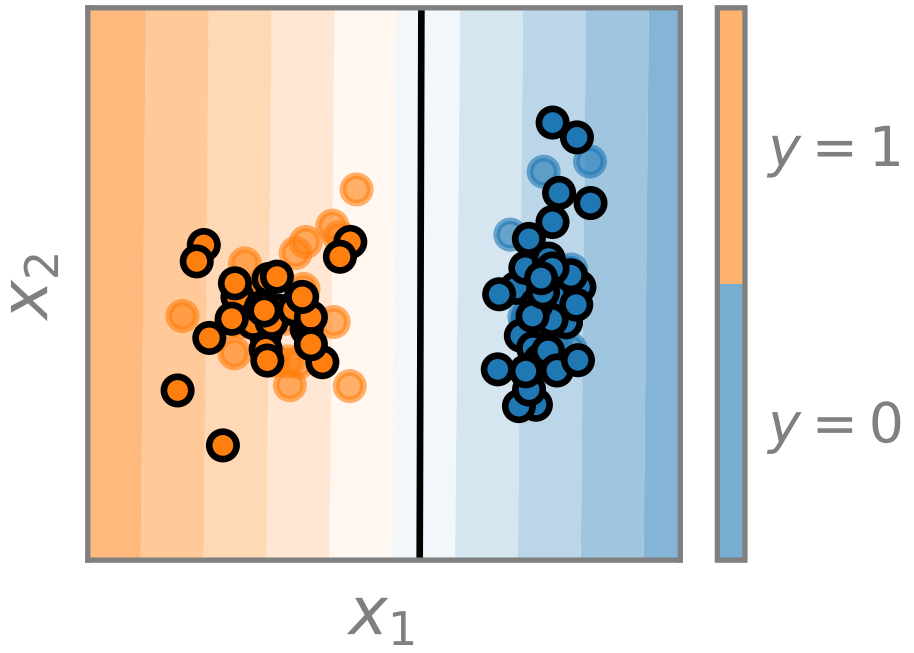
The classification task with support vector machines (SVM)

The decision function $f(\mathbf{x})$ dependson the sign of the linear combination of the normal vector and the sample:

$$f(\mathbf{x}) = \mathbf{w} \cdot \mathbf{x} + b$$

The quantity to minimize is the **Hinge loss**:

$$\mathcal{L}(\mathbf{w}, b) = \frac{1}{N} \sum_{i=1}^{N} \max\left(0, 1 - y_i \left(\mathbf{w} \cdot \mathbf{x}_i + b\right)\right)$$



The classification task with support vector machines (SVM)

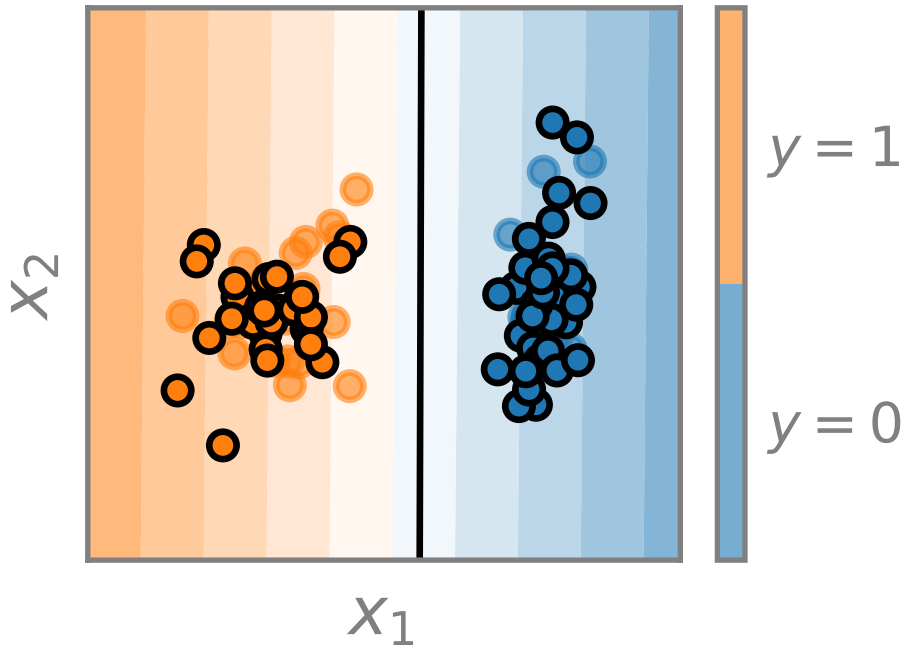
The decision function $f(\mathbf{x})$ depends on the sign of the linear combination of the normal vector and the sample:

$$f(\mathbf{x}) = \mathbf{w} \cdot \mathbf{x} + b$$

The quantity to minimize is the **hinge loss**:

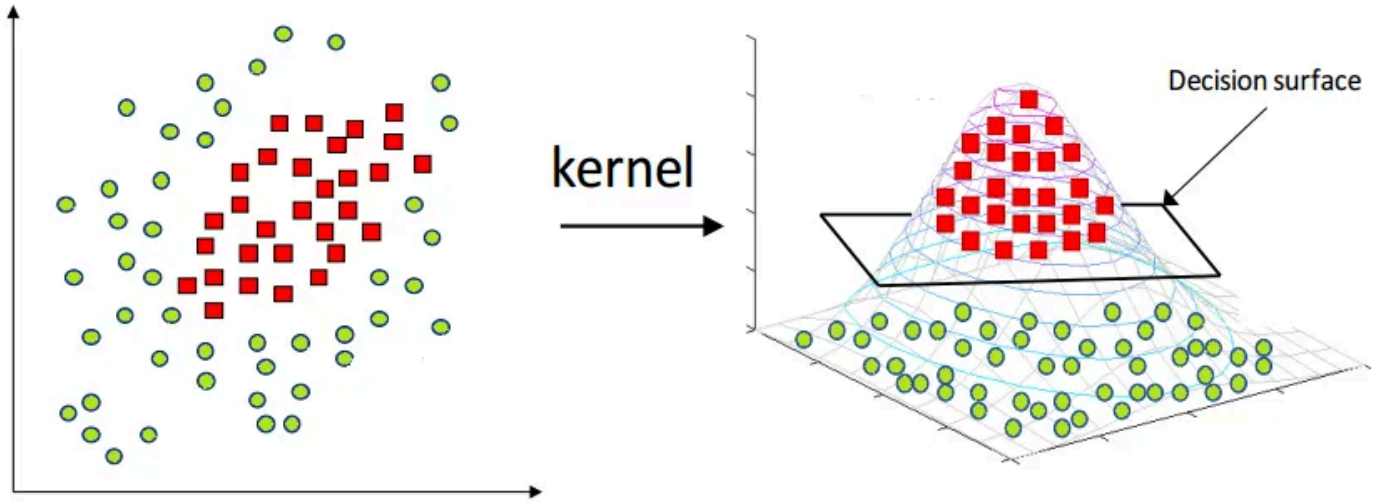
$$\mathcal{L}(\mathbf{w}, b) = \frac{1}{N} \sum_{i=1}^{N} \max\left(0, 1 - y_i \left(\mathbf{w} \cdot \mathbf{x}_i + b\right)\right)$$

What about non linear problems?



The kernel trick for non linear classification problems

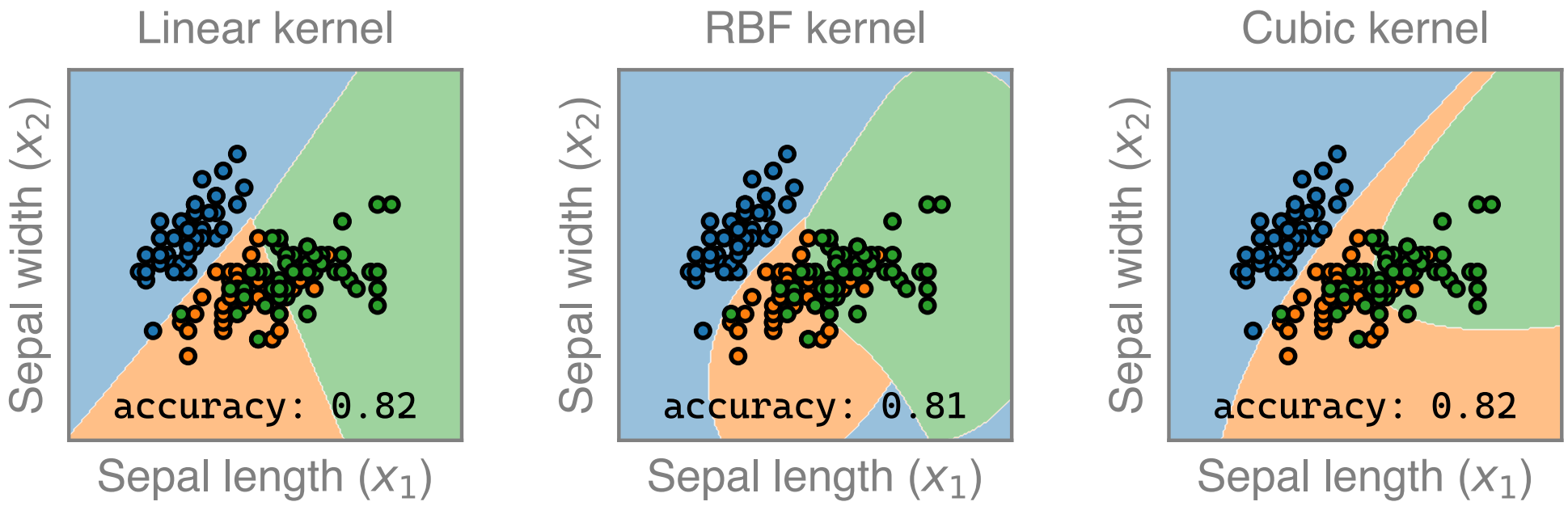
The kernel trick allows to map the data to a **higher dimensional** space made from the input features where the problem is **linearly separable**.



The **Radial Basis Functions** (RBF) is an infinite kernel $K(\mathbf{x}, \mathbf{x}') = \exp\left(-\frac{\|\mathbf{x}-\mathbf{x}'\|^2}{2\sigma^2}\right)$

Generalization of the SVM: the support vector classifier (SVC)

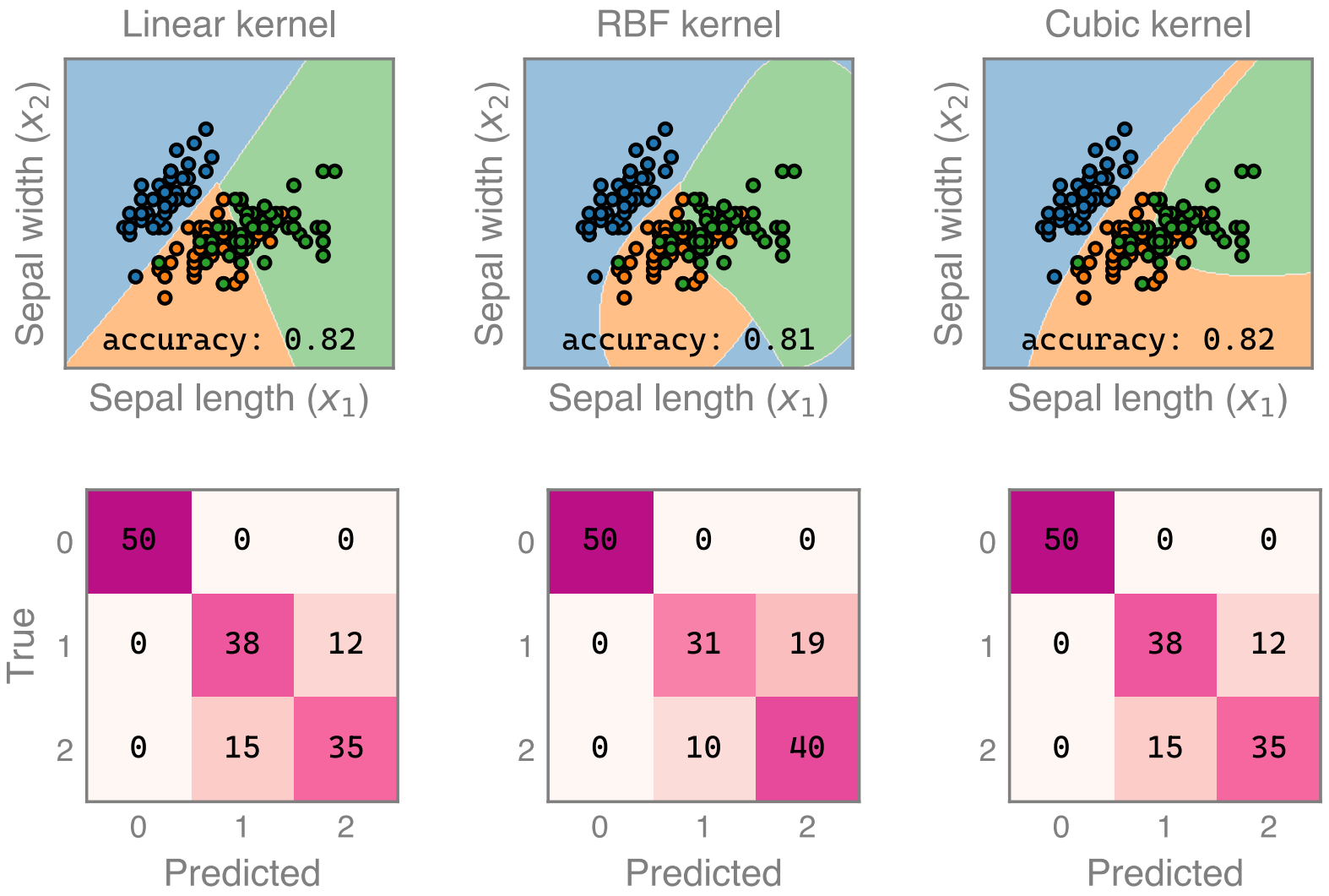
The SVC is a generalization of the SVM that digests more than two classes.



The decision function is linear in the kernel space only.

We can project it back to the data space to inspect it.

Various classifcation metrics from the confusion matrix



Various classifcation metrics: accuracy, precision, recall

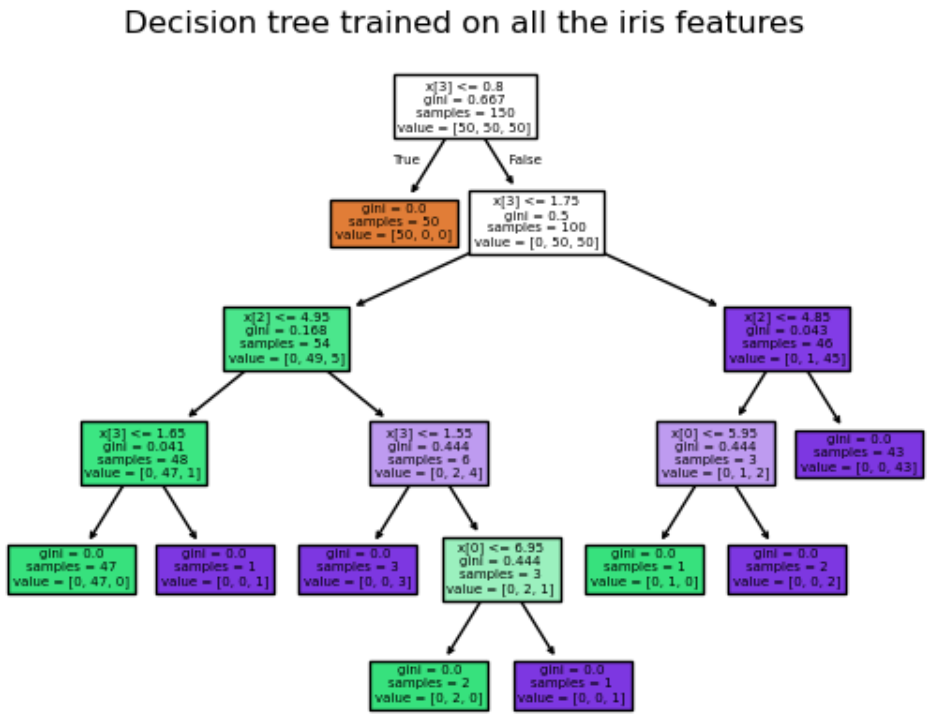
Ma et al. (2019)

Decision trees and random forests

Decision trees learn to predict y with feature splitting.

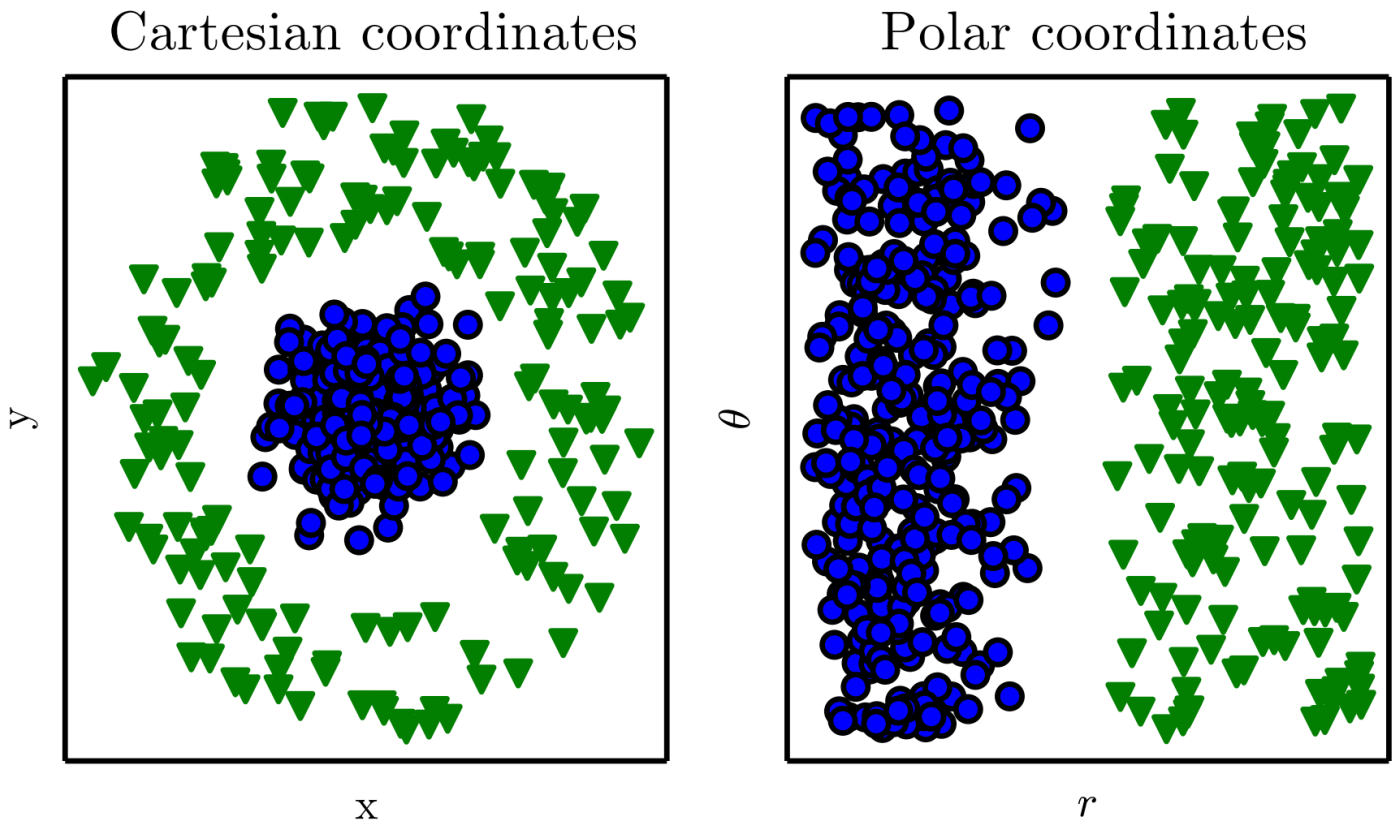
Random forests are ensembles of decision trees that vote for y.

These algorithms are extremely powerful.



from scikit-learn.org

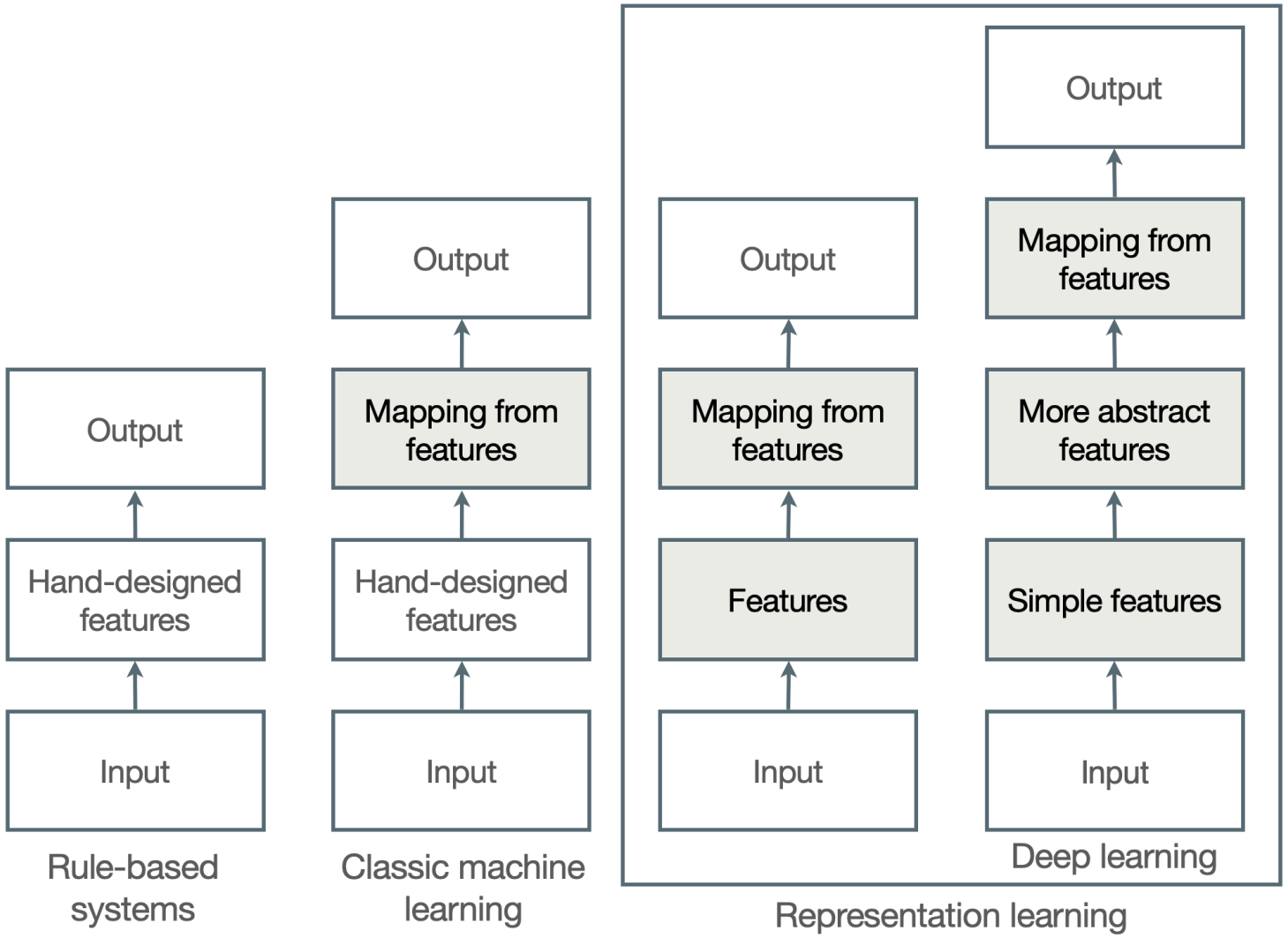
Representation matters!



There is no need for a complex model if you have a good **representation** of the data.

Learning strategies depending on the task complexity

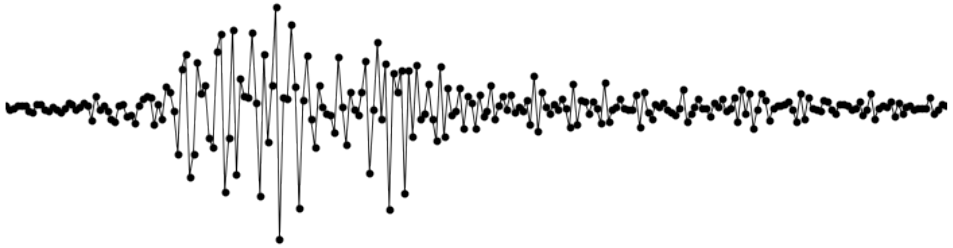
Hand-designed or learned features?



Goodfellow et al. (2016)

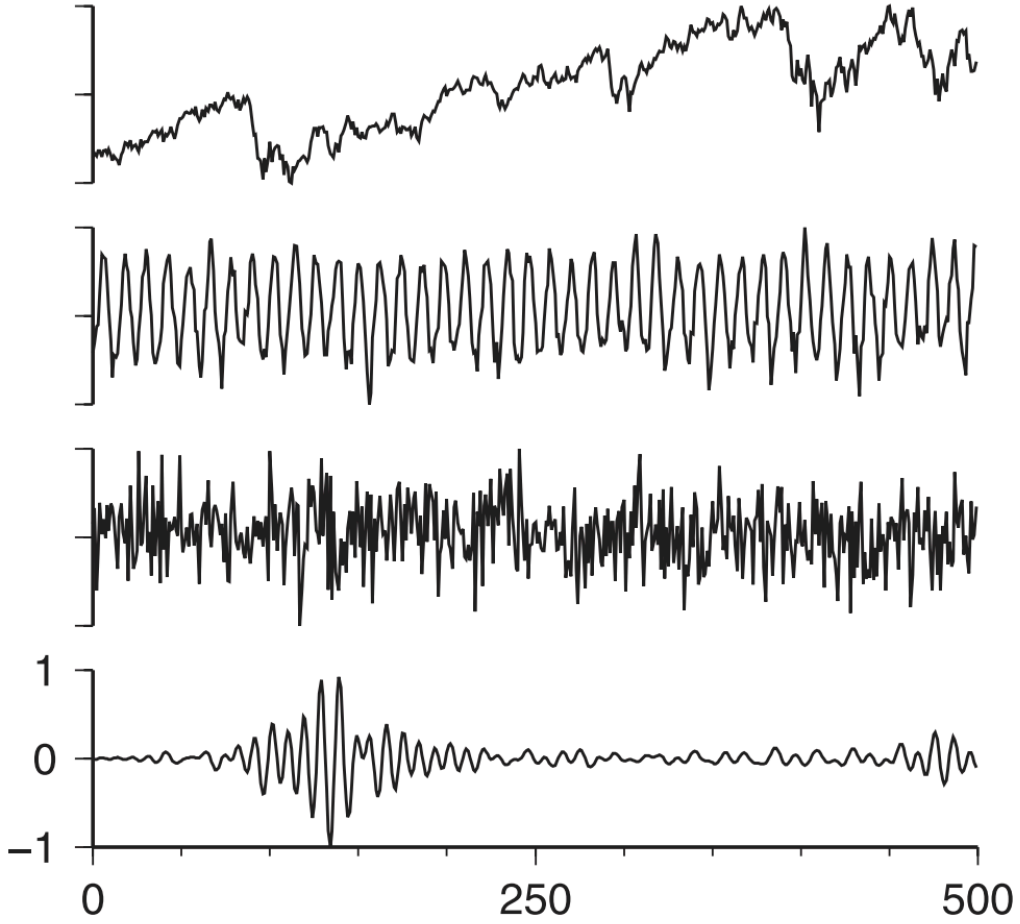
Why would waveforms not be the best representation of ground motion?

We can see waveforms $\mathbf{x} \in \mathbb{R}^N$ as points of a N-dimensional space



Yet, seismic waveform do not occupy this space fully, likely very sparse.

Dimension > Information

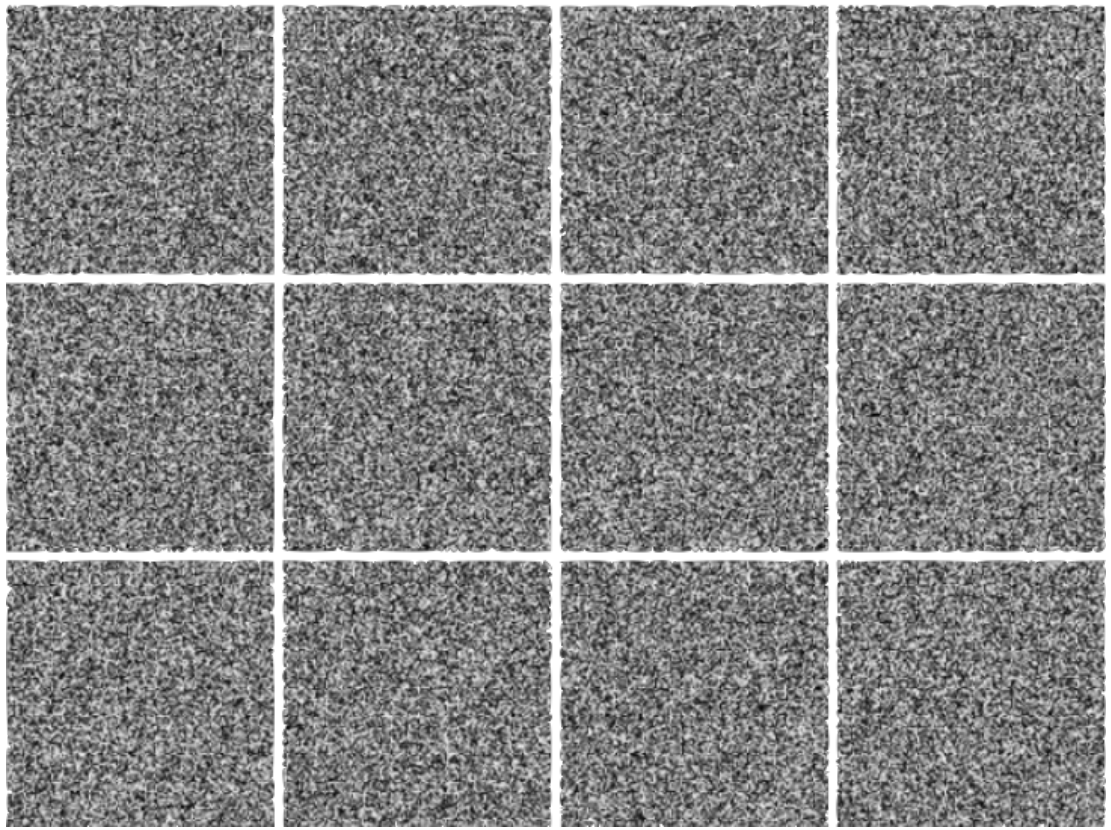


still from Valentine & Trampert (2012)

Actually, natural data spaces may be often not fully occupied

Random sampling of the pixels of a face. What is the likelihood that the reshuffled image *is* a face?

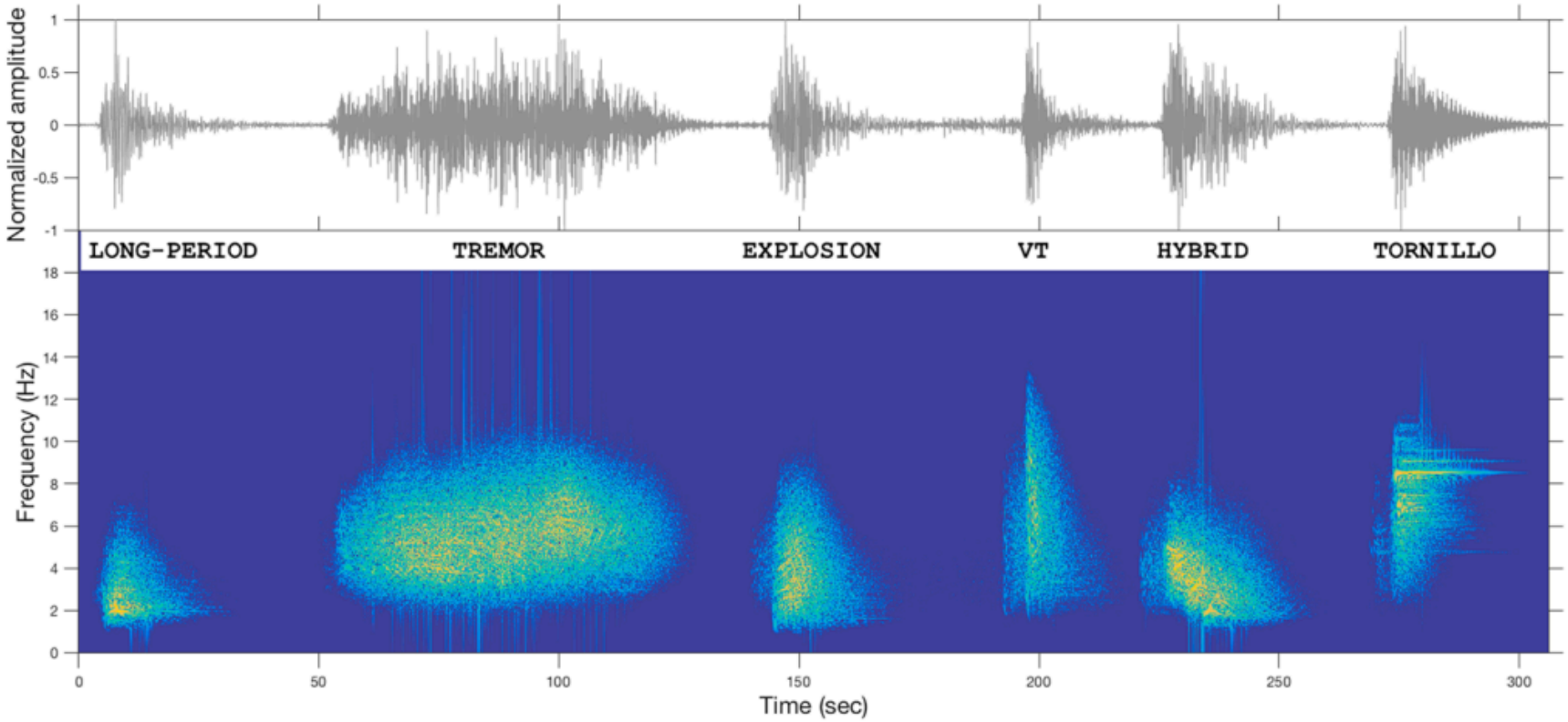
Like waveforms, **images are living on a manifold.**



modified from Goodfellow et al. (2016)

Supervised learning for sismo-volcanic signal classification

Supervised learning experiences a set of examples containing features $\mathbf{x}_i \in \mathbb{X}$ associated with labels $\mathbf{y} \in \mathbb{Y}$ to be predicted from the features (here, classification).

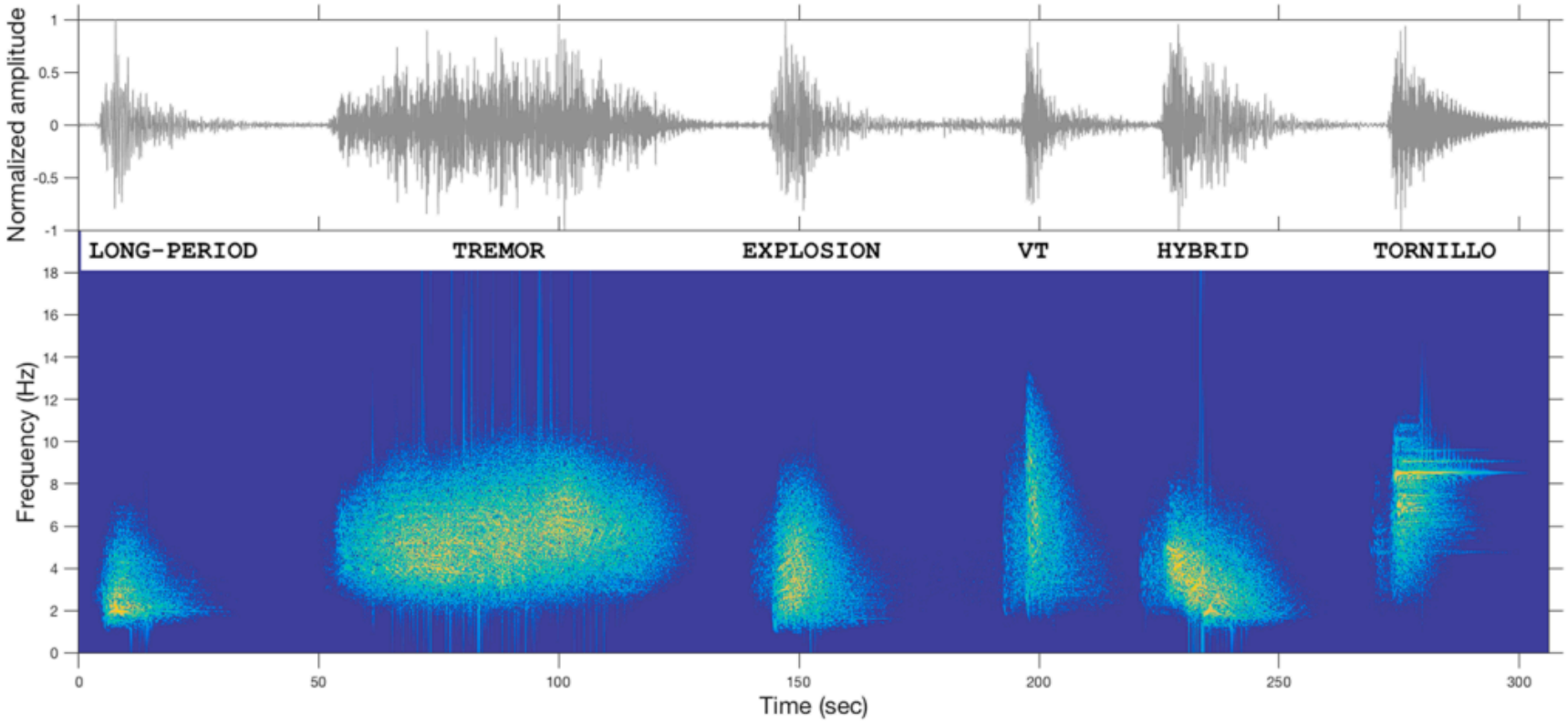


Malfante et al. (2018)

Supervised learning for sismo-volcanic signal classification

In this case, $\mathbf{x}$ lies in $\mathbb{R}^{3\times N}$, and $\mathbf{y}$ in $[0, \ldots, 5]$.

Which **representation** of $\mathbf{x}$ works best?



Malfante et al. (2018)

Handcrafted features for classical machine learning

We need to find relevant descriptors of our data, used as features $\mathbf{x}$.

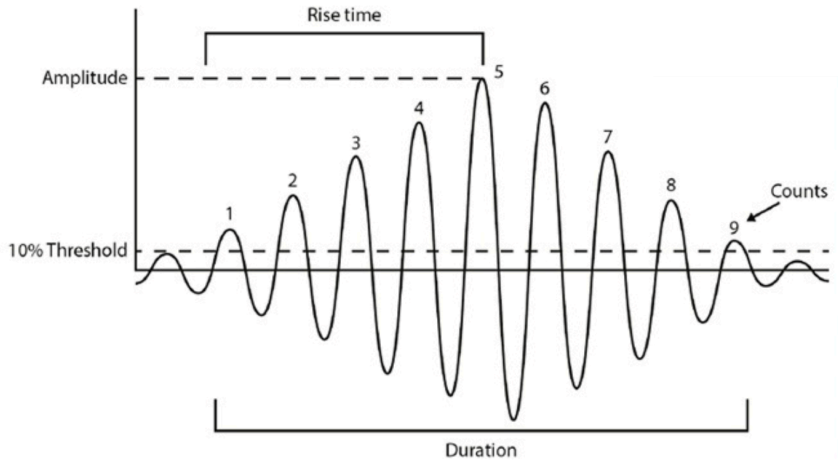


Table 1
List of Features

Statistic features

Feature	Definition	Used in	Ref.
Length	$n = \text{length}(s)$	Tucker and Brown (2005)	1
Mean	$\mu_s = \frac{1}{n}\sum_i s[i]$	Tucker and Brown (2005)	2
Standard deviation	$\sigma_s = \sqrt{\frac{1}{(n-1)}\sum_i (s[i]-\mu_s)^2}$		3
Skewness	$\frac{1}{n}\sum_i \left(\frac{s[i]-\mu_s}{\sigma_s}\right)^3$	Langet (2014) and Hibert et al. (2014)	4
Kurtosis	$\frac{1}{n}\sum_i \left(\frac{s[i]-\mu_s}{\sigma_s}\right)^4$	Langet (2014) and Hibert et al. (2014)	5
i of central energy	$\bar{i} = \frac{1}{E}\cdot\sum_i E_i \cdot i$	(Tucker & Brown, 2005)	6
RMS bandwidth	$B_i = \sqrt{\frac{1}{E}\sum_i i^2 \cdot E_i - \bar{i}^2}$	Tucker and Brown (2005)	7
Mean skewness	$\sqrt{\frac{\sum_i (i-\bar{i})^3 E_i}{E\cdot B_i^3}}$	Tucker and Brown (2005)	8
Mean kurtosis	$\sqrt{\frac{\sum_i (i-\bar{i})^4 E_i}{E\cdot B_i^4}}$	Tucker and Brown (2005)	9

Entropy features (with $p(s_j)$ the probability of amplitude level s_j)

Feature	Definition	Ref.	
Shannon entropy[a]	$-\sum_j p(s_j)\log_2\left(p(s_j)\right)$	Esmaili et al. (2004) and Han et al. (2011)	10 to 12
Rényi *entropy*[b]	$\frac{1}{1-\alpha}\cdot\log_2\left(\sum_j p(s_j)^\alpha\right)$	Han et al. (2011)	13 to 18

Shape descriptor features

Feature	Definition	Ref.	
Rate of attack	$\max_i\left(\frac{s[i]-s[i-1]}{n}\right)$	Tucker and Brown (2005)	19
Rate of decay	$\min_i\left(\frac{s[i]-s[i+1]}{n}\right)$	Tucker and Brown (2005)	20
Ratios	min/mean and max/mean	Langet (2014) and Hibert et al. (2014)	21 to 22
Energy descriptors	Signal energy, maximum, average, standard deviation, skewness, and kurtosis	Tucker and Brown (2005)	23 to 28
Specific values	min, max, *i* of min, *i* of max, threshold crossing rate, and silence ratio	Tucker and Brown (2005)	29 to 34

Note. Features computed for a signal $s[i]_{i=1}^{n}$ (in which *i* might correspond to a temporal, frequency, or cepstral sample). $E = \sum_{i=1}^{n} s[i]^2$ and $E_i = s[i]^2$ describe the signal energy and the energy at sample *i*, respectively. Some features have a dimension greater than others; e.g., entropy measurements are made on three different estimations of the amplitude probability (i.e., different histogram bin numbers).
[a]Bin numbers for probability estimation: 5, 30, and 500. [b]Bin numbers for probability estimation: 5, 30, 500, $\alpha = 2$, and inf.

Jasperson et al. (2022)

Performance measure, and what can we learn from it?

Accuracy of the predictions measures the model's performance (= confusion matrix)

Table 3
Confusion Matrix

		True Class (ground truth)						
		LP	TR	VT	EXP	HYB	TOR	Precision
	LP	**58,363**	627	8	0	5	1	98.9%
	TR	3,000	**4,584**	0	1	2	0	60.4%
Predicted	VT	478	11	**475**	5	11	3	48.3%
Class	EXP	15	16	2	**29**	0	0	47.8%
	HYB	131	3	28	13	**125**	0	41.7%
	TOR	43	4	3	0	0	**28**	35.9%
Accuracy		**94.1%**	**87.4%**	**92.2%**	**59.8%**	**87.1%**	**84.6%**	

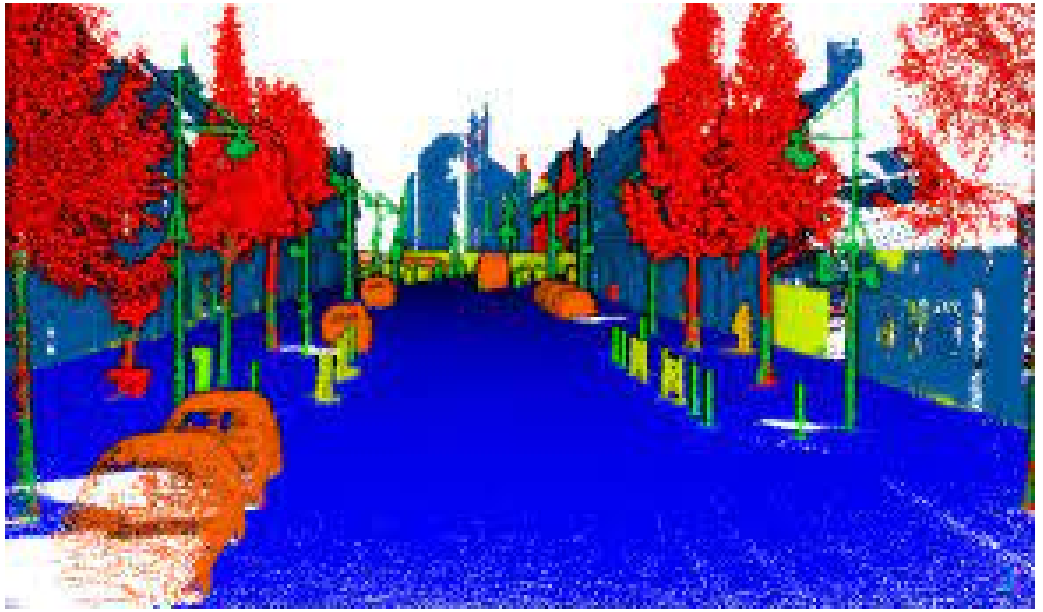
What is the guarantee that the features we choose are the best ones?

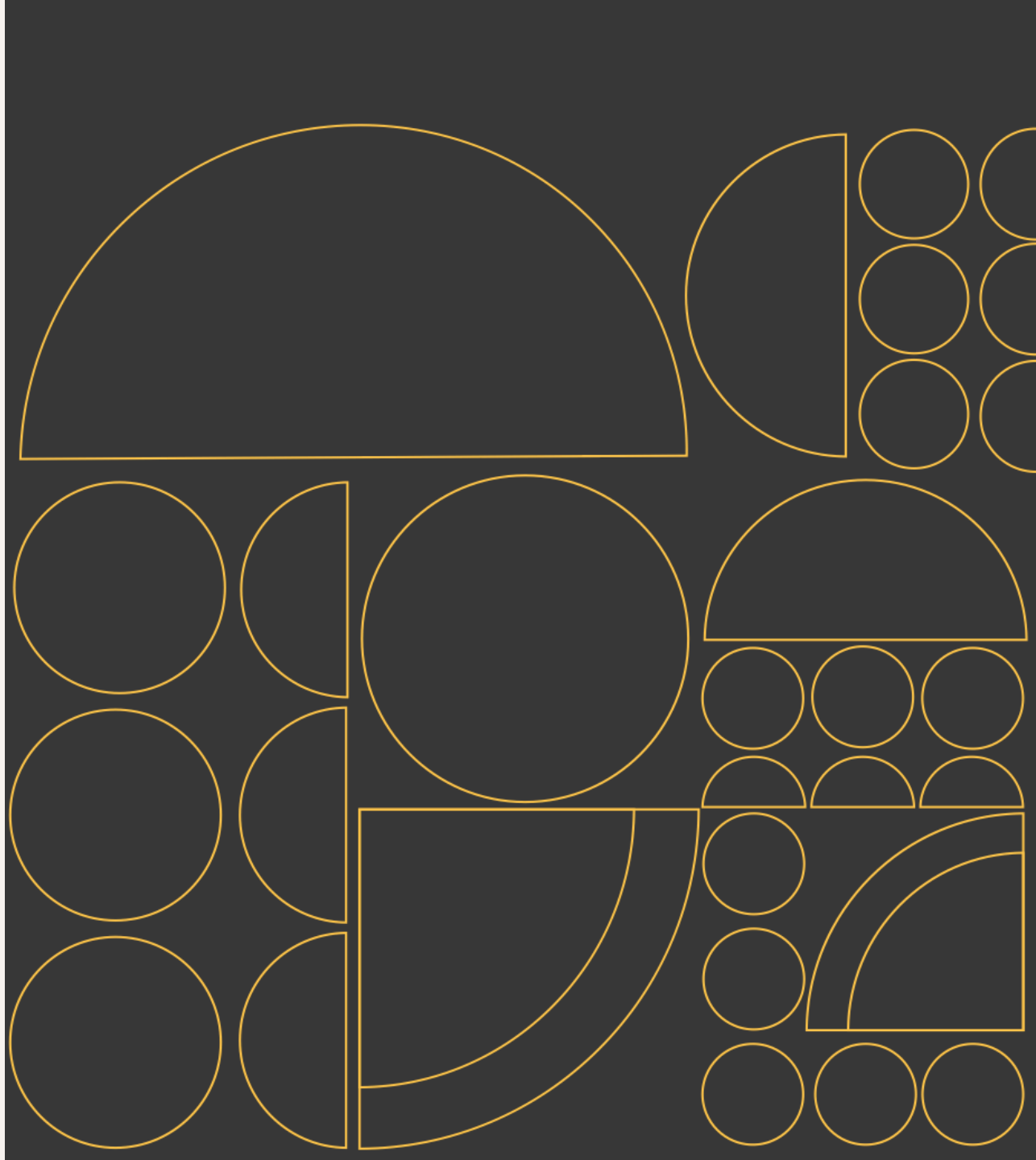
Malafante et al. (2018)

Notebook: Lidar point cloud classification

Problem: automate the identification of objects in a lidar cloud from labeled subset.

Objectives: supervised learning, classification, non-linear models, multi-scale features.





ChEESE

5. Deep learning: multi-layer perceptrons

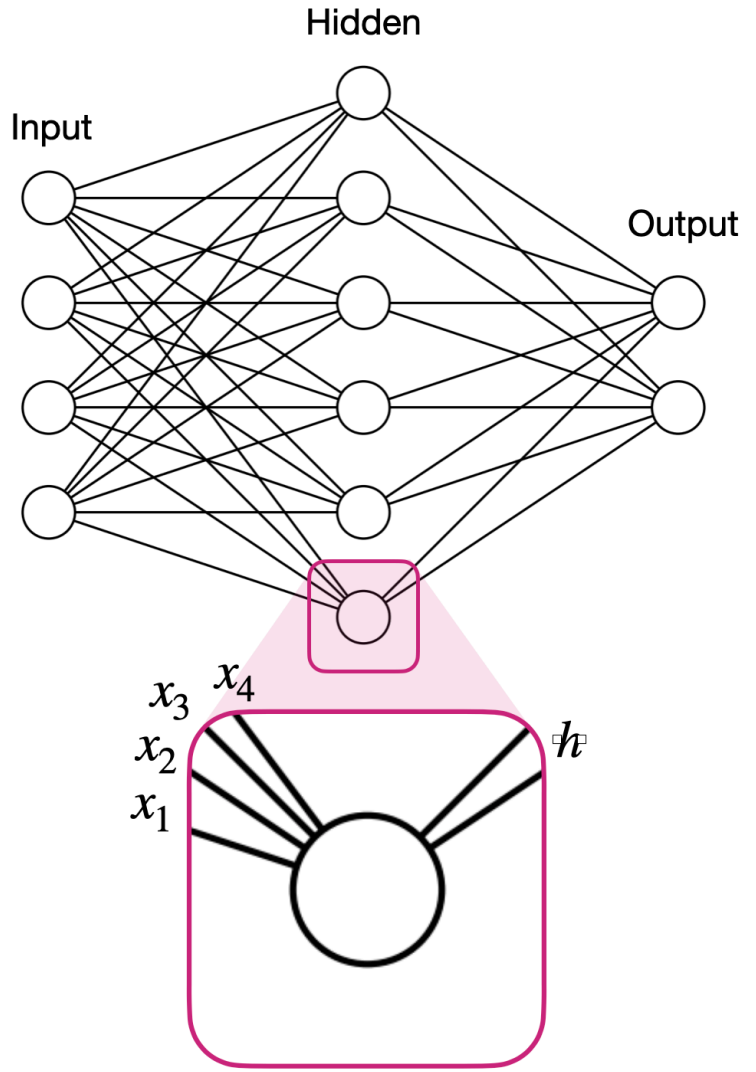
How deep learning works? What is a neural network? How to train it, and what for?

A general form of an artificial neuron

A **neuron**, or unit, takes a set of inputs $\mathbf{x}$ and outputs an activation value h, as

$$h = \varphi\left(\sum_{i=1}^{N} w_i x_i + b\right)$$

with w_i the weights, b the bias, φ is the activation function, and N is the number of inputs.



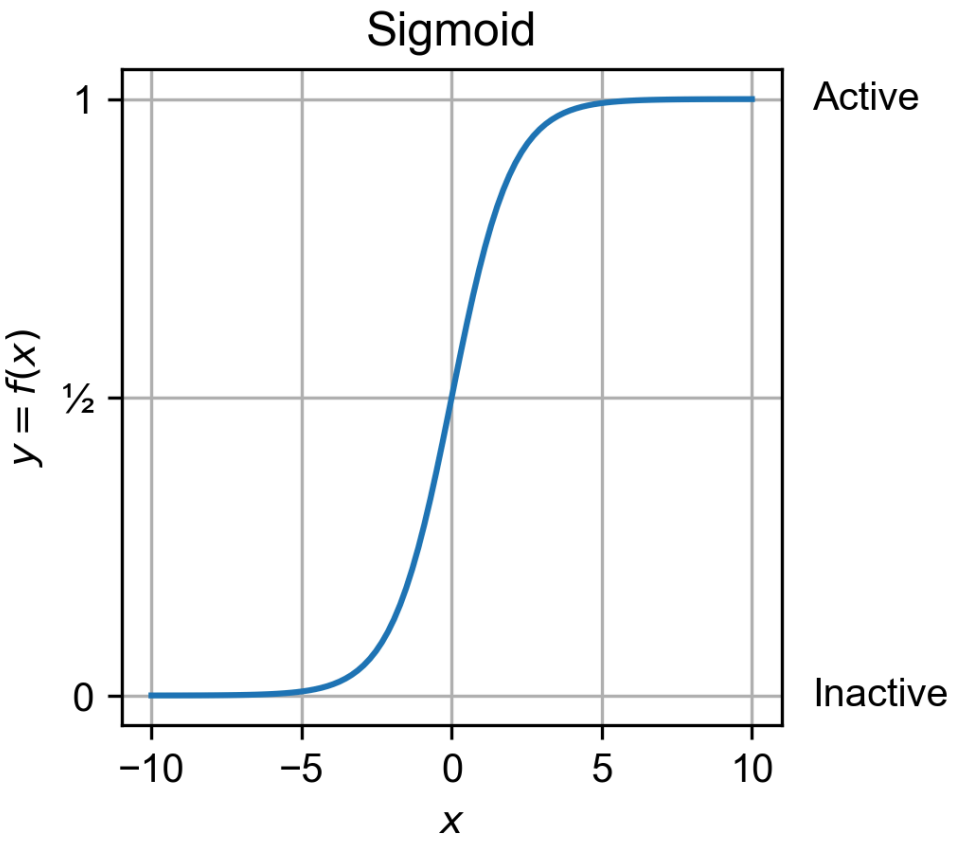
A famous neuron: the sigmoid unit

A **neuron**, or unit, transforms a set of inputs $\mathbf{x}$ into an output h, as

$$h = \varphi\left(\sum_{i=1}^{N} w_i x_i + b\right)$$

with w_i the weights, b the bias, φ is the activation function, and N is the number of inputs. Common activation functions include the **sigmoid** function, defined as

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



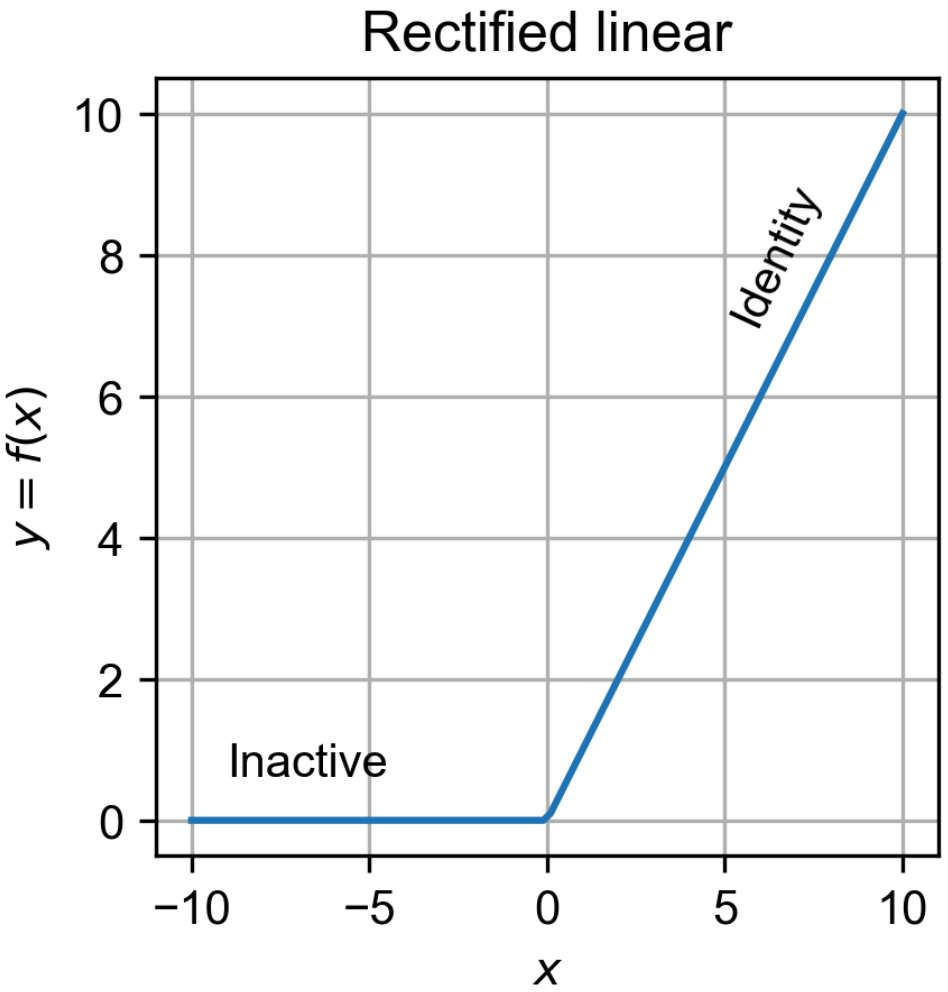
An even more famous one: the rectified linear unit

A **neuron**, or unit, transforms a set of inputs $\mathbf{x}$ into an output h, as

$$h = \varphi\left(\sum_{i=1}^{N} w_i x_i + b\right)$$

with w_i the weights, b the bias, φ is the activation function, and N is the number of inputs. Common activation functions include the **rectified linear unit** (ReLU), defined as

$$\varphi(z) = \max(0, z)$$



ReLU are empirically preferred to sigmoid units for computational efficiency no saturation when x is large.

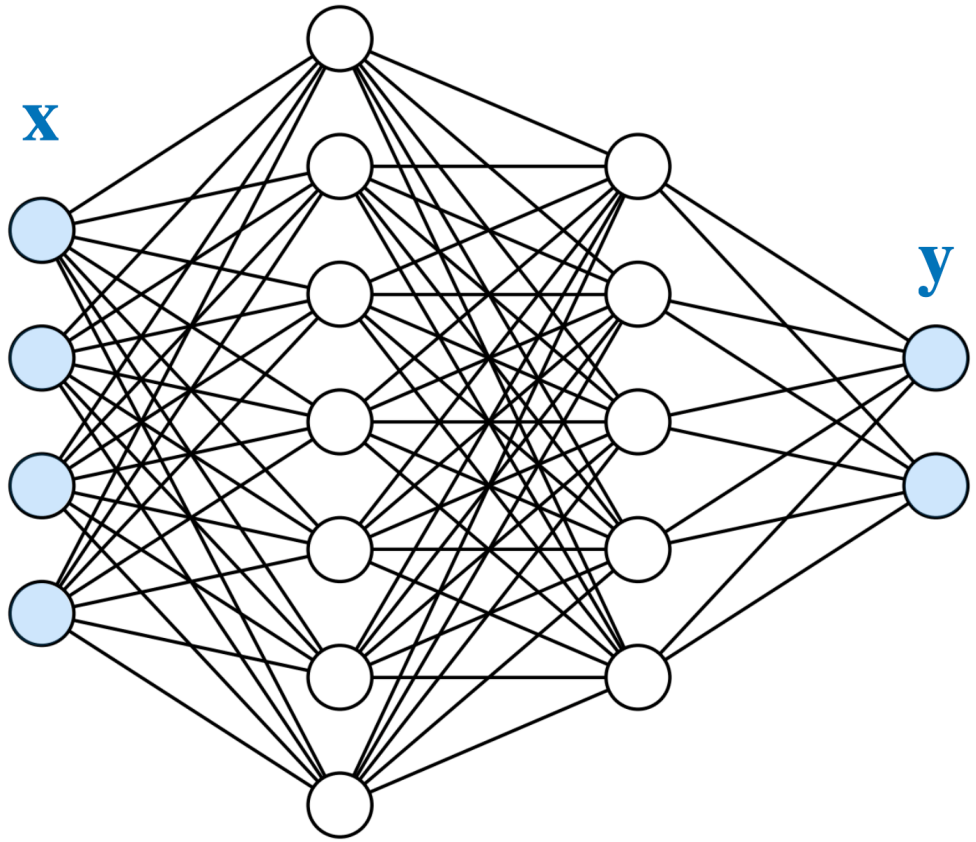
The multilayer perceptron (MLP)

A **multilayer perceptron** is a neural network with multiple hidden layers:

$$h_i^{(1)} = \varphi^{(1)}\left(\sum_j w_{ij}^{(1)} x_j + b_i^{(1)}\right)$$

$$h_i^{(2)} = \varphi^{(2)}\left(\sum_j w_{ij}^{(2)} h_j^{(1)} + b_i^{(2)}\right)$$

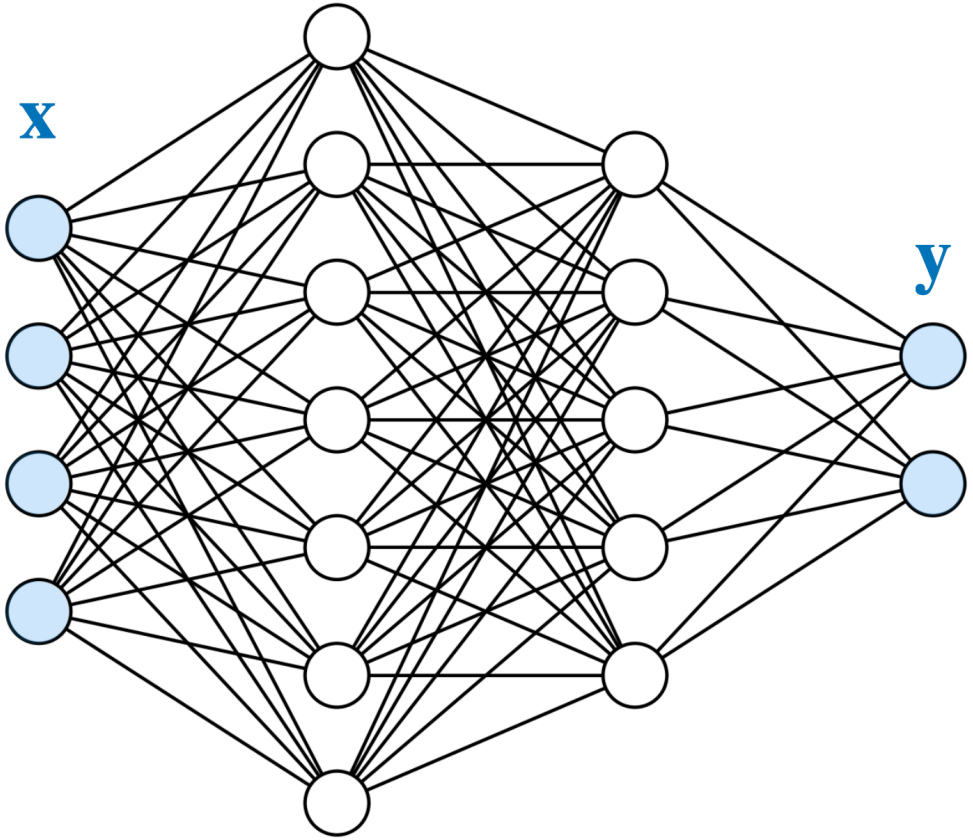
$$y_i = \varphi^{(3)}\left(\sum_j w_{ij}^{(3)} h_j^{(2)} + b_i^{(3)}\right)$$



The multilayer perceptron (MLP)

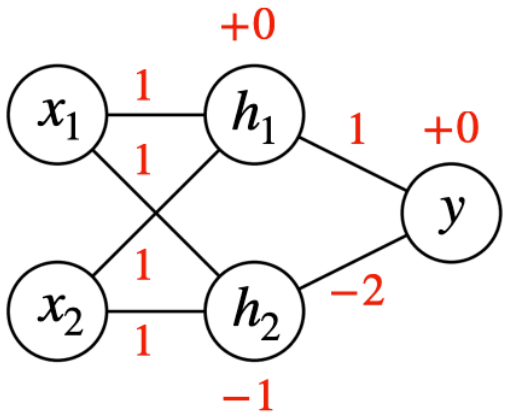
A **multilayer perceptron** is a neural network with multiple hidden layers. Generally speaking (omitting the biases):

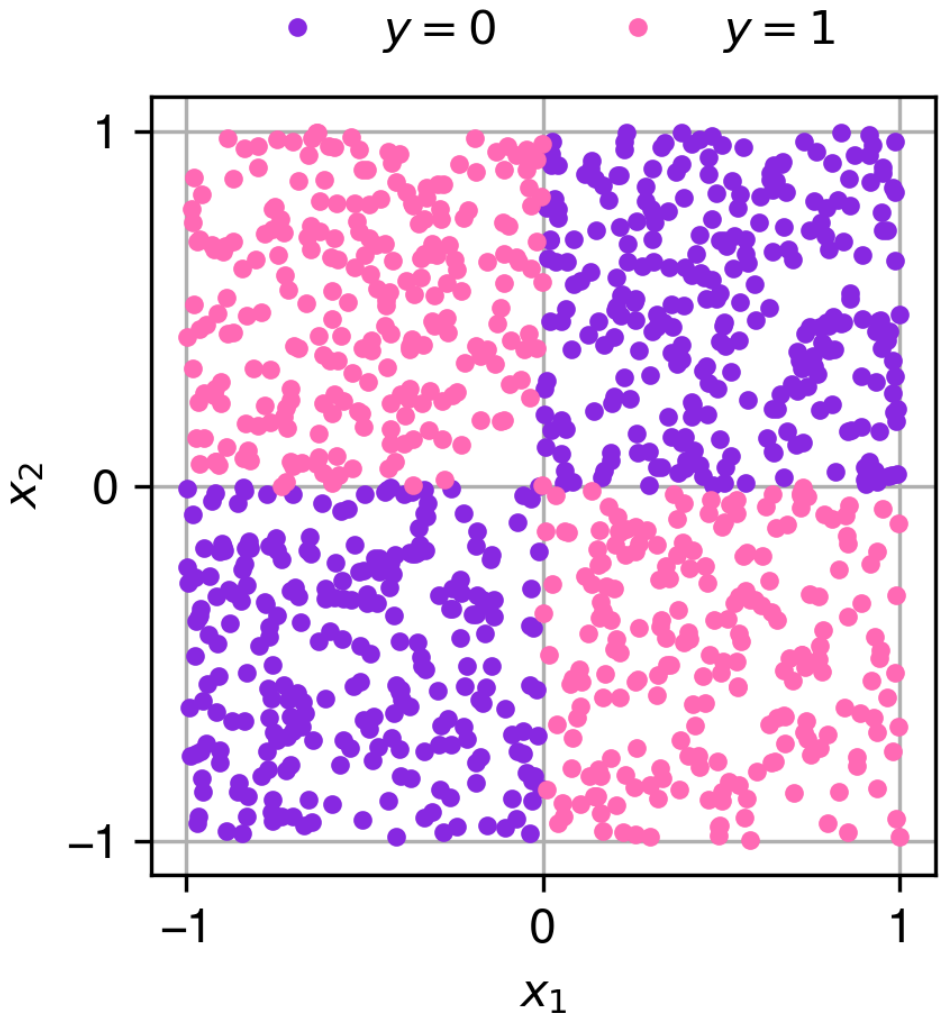
$$y = \varphi^{(\ell)}\left(\mathbf{W}^{(\ell)}\varphi^{(\ell-1)}\left(\mathbf{W}^{(\ell-1)}\dots\varphi^{(1)}\left(\mathbf{W}^{(1)}\mathbf{x}\right)\dots\right)\right)$$



Quick example for solving the XOR problem

Multi-layer perceptrons that solves the XOR problem with binary activations:



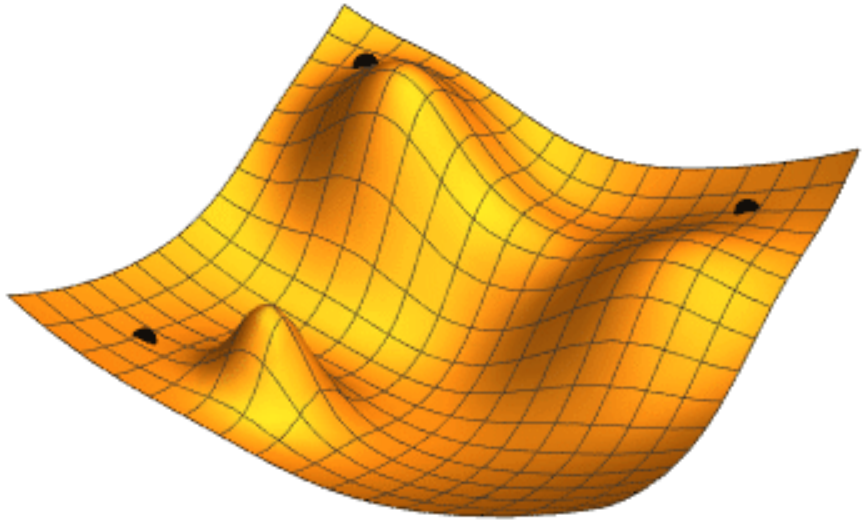


See Section 6.1 of Goodfellow et al. (2016)

Gradient descent for neural networks

We note $f_\theta(x) : x \mapsto y$ the model, where θ are the parameters of the model (including biases and weights).

1. **Learning** is the process of finding the parameters θ^* that minimize the loss $\mathcal{L}$.
2. The **backpropagation** computes the loss function gradient with respect to θ.
3. The **gradient descent** updates θ in the direction of the steepest descent.



Gradient computation with backpropagation

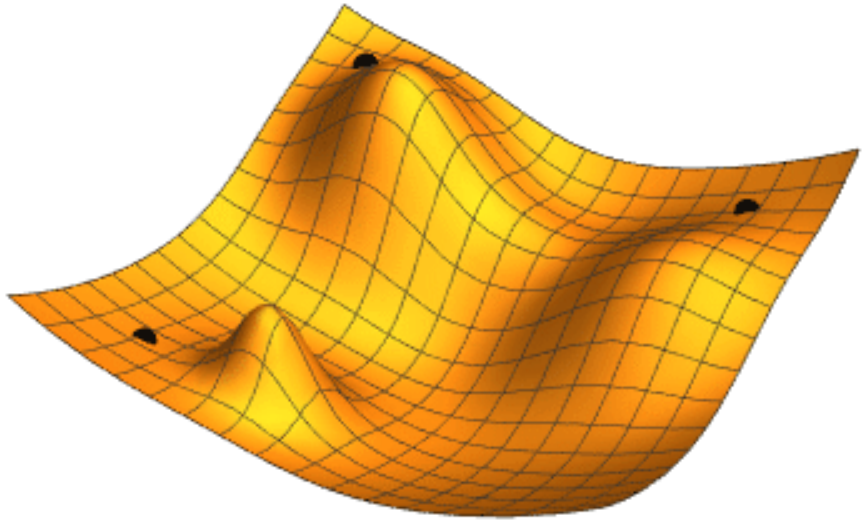
1. **Initialization**: the weights are initialized randomly, the biases to zero
2. **Feed forward**: the input is propagated through the network to compute the output
3. **Loss**: the loss is computed between the output and the target
4. **Back propagation**: computation of the gradient from the loss to the input
5. **Gradient descent**: update the parameters in the direction of the steepest descent

Gradient-based optimization

Once the gradient is computed, the parameters are updated using the **gradient descent** algorithm:

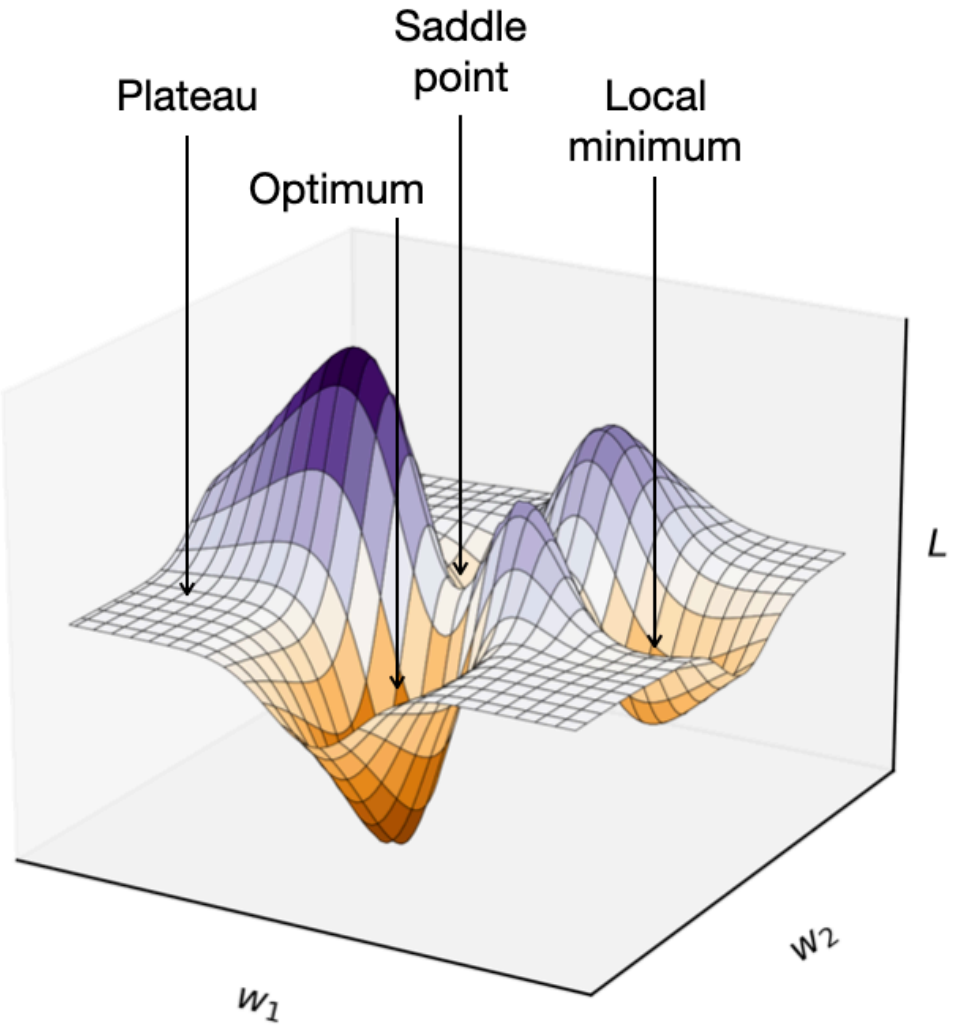
$$\theta \leftarrow \theta - \eta \frac{\partial \mathcal{L}}{\partial \theta}$$

where is η the **learning rate** that controls the size of the update.

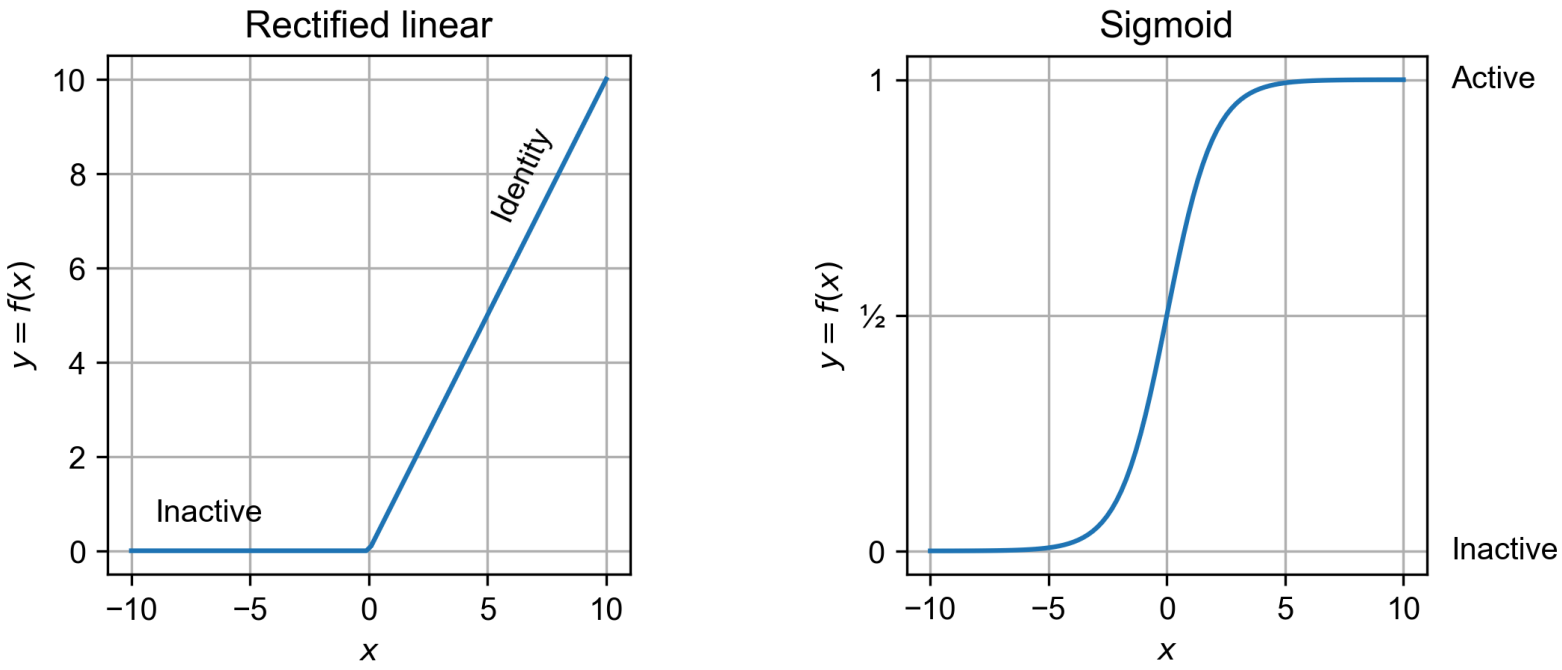


Gradient descent common issues

- **Local minima**: getting stuck in a local minimum.
- **Sattling points**: behaves as a local minimum but is not.
- **Plateau**: flat loss function, vanishing gradient, slow convergence.



Gradient descent common issues with plateau



Plateau are flat regions of the loss function where the gradient is zero. This can happen with activation functions such as the sigmoid function with saturation. It can also happen with the ReLU function for inputs with negative values.

Gradient-descent tricks to avoid issue

- **Learning rate**: set up, and maybe adapt it.
- **Momentum**: use the gradient of the previous iteration to update the parameters.
- **Normalization**: normalize the inputs of each layer.
- **Stochastic gradient descent**: use a mini-batch of samples to compute the gradient.
- **Dropout**: randomly drop some neurons during training.

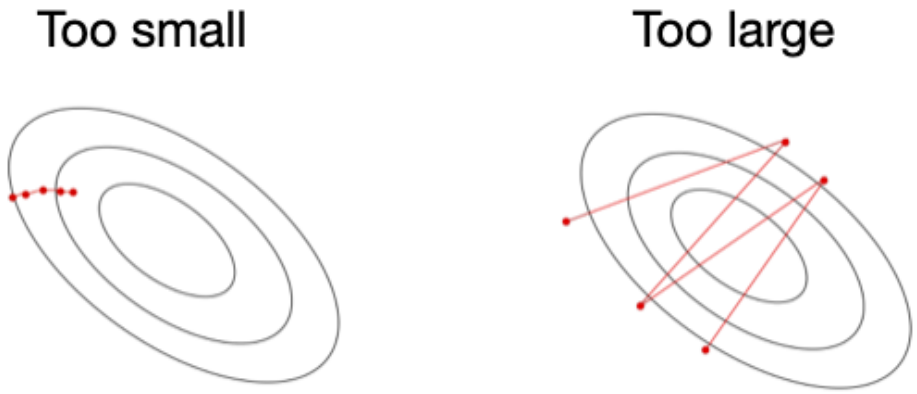
Gradient descent and learning rate

The **learning rate** is a hyperparameter that controls the size of the update of the parameters:

$$\theta \leftarrow \theta - \eta \frac{\partial \mathcal{L}}{\partial \theta}$$

We must look for a learning rate to avoid local minima while still converging fast enough, without diverging.

> We can also **adapt** the learning rate.

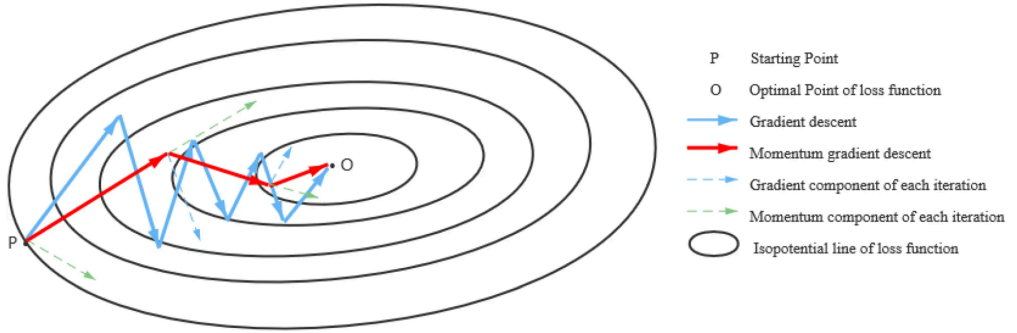


Gradient descent and momentum

The **momentum** is a technique to accelerate the gradient descent by adding a fraction of the gradient of the previous iteration:

$$p \leftarrow \alpha p - \eta \frac{\partial \mathcal{L}}{\partial \theta}$$
$$\theta \leftarrow \theta + p$$

where α is the a damping parameter, and v_i is the **velocity**. Lower values of α give more weight to the current gradient, higher values give more weight to the previous gradients.



From Zhang et al. (2021)

Data normalization

To avoid getting in the saturation of sigmoidal activation functions, it is important to normalize the data. This can be done by **normalizing the input and the features**:

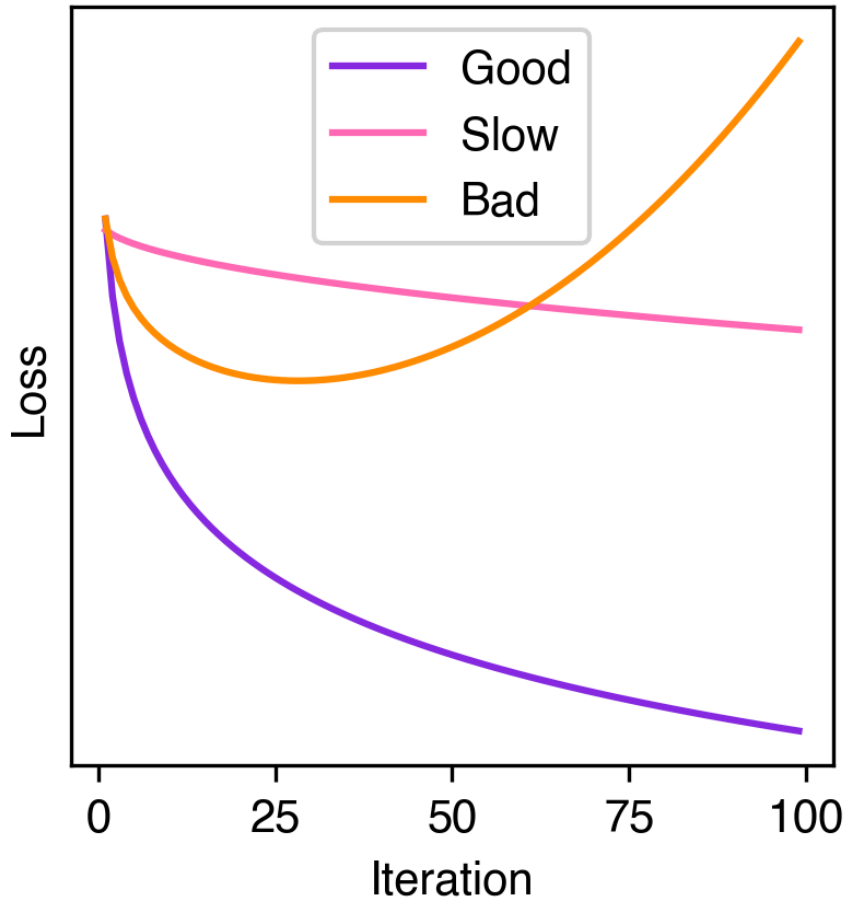
$$\hat{x}_i = \frac{x_i - \mu_i}{\sqrt{\sigma_i^2 + \epsilon}}$$

where μ_i is the mean of the input, σ_i is the standard deviation of the input, and ϵ is a small constant to avoid division by zero. You can also apply the normalization after the activation function.

Monitor the training curves

The **training curves** are a good way to monitor the training of a model.

- Slow: increase the learning rate.
- Growing: decrease the learning rate.
- Cross-validation: within 0.0001 to 0.1

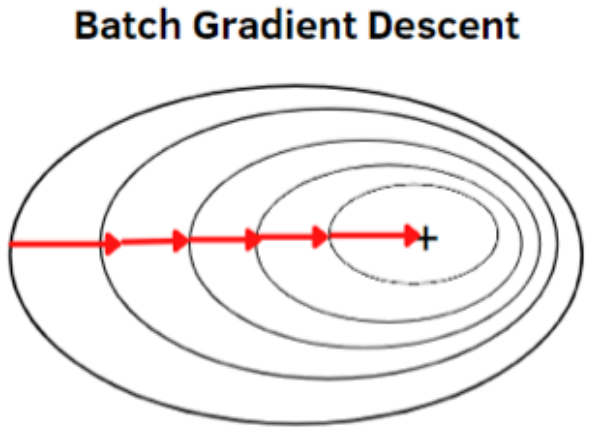


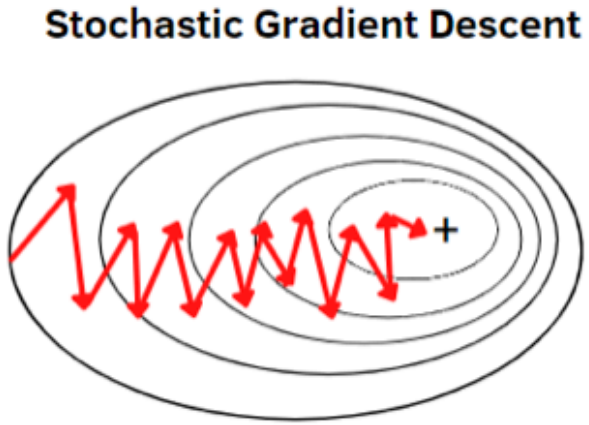
Stochastic gradient descent

The gradient of the loss function with respect to the parameters θ is computed using the **full-batch gradient descent** equal to:

$$\frac{\partial \mathcal{L}}{\partial \theta} = \frac{1}{N} \sum_{i=1}^{N} \frac{\partial \mathcal{L}^{(i)}}{\partial \theta}$$

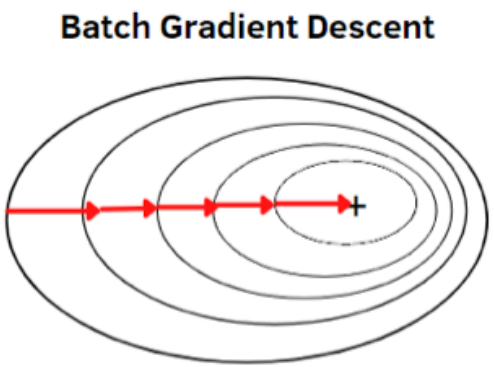
The **stochastic gradient descent** is a technique to compute the loss gradient from every sample in the dataset at each iteration.

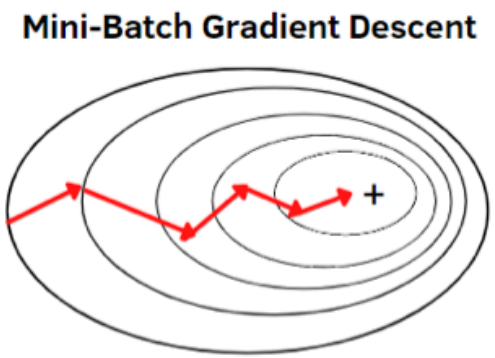


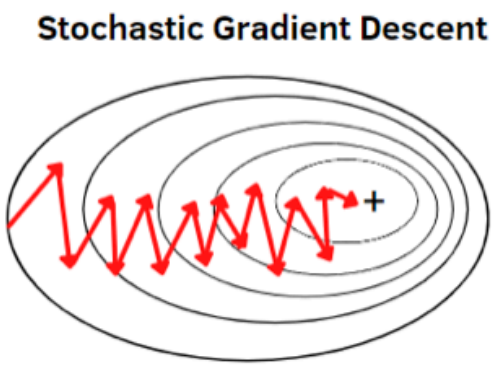


Mini-batch gradient descent

The **mini-batch gradient descent** is a technique to compute the gradient of the loss function with respect to a subset of the dataset. It is a compromise between the full-batch gradient descent and the stochastic gradient descent.



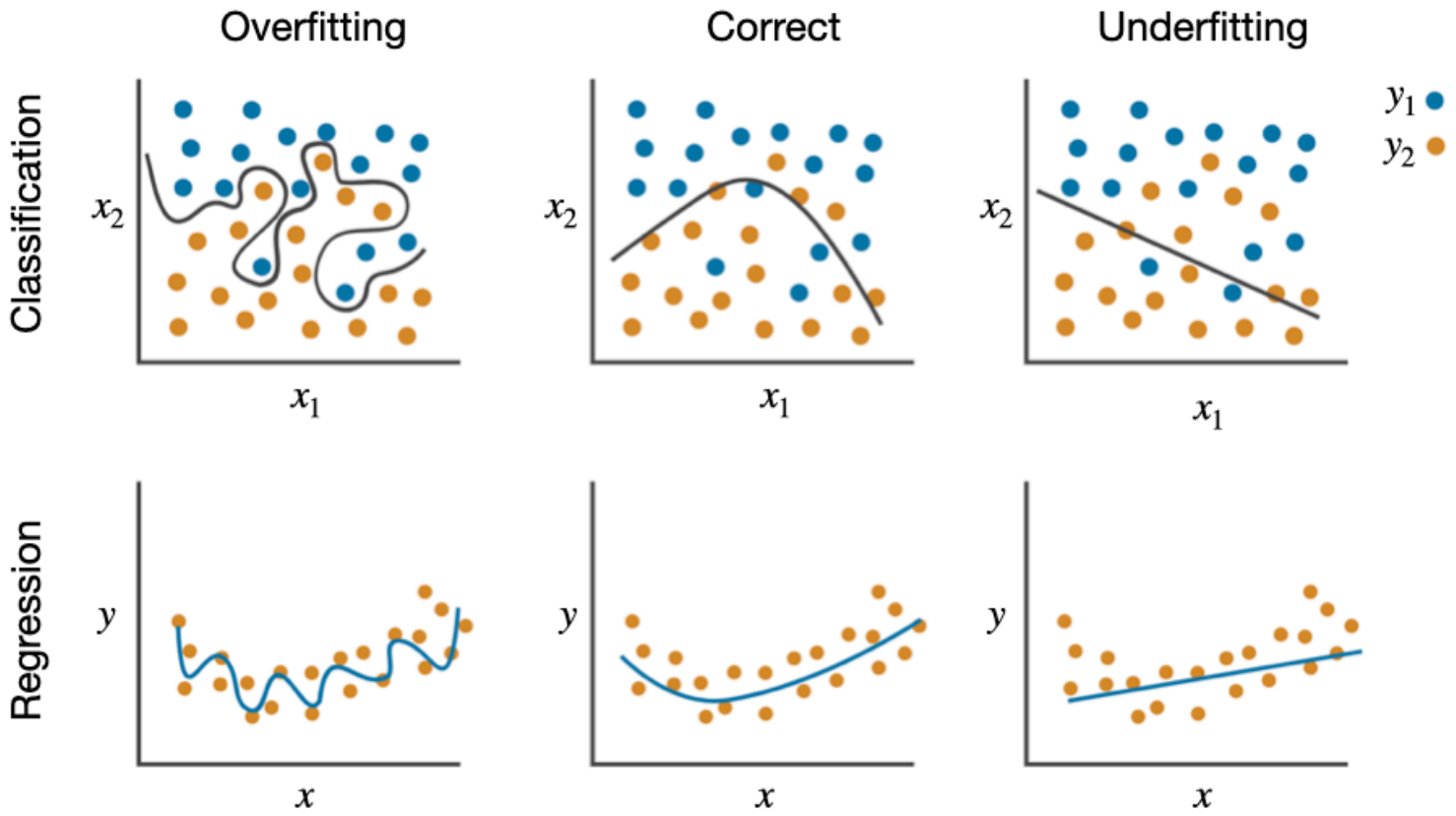




Overfitting and underfitting

Overfitting: too complex model, does not generalize to new data.

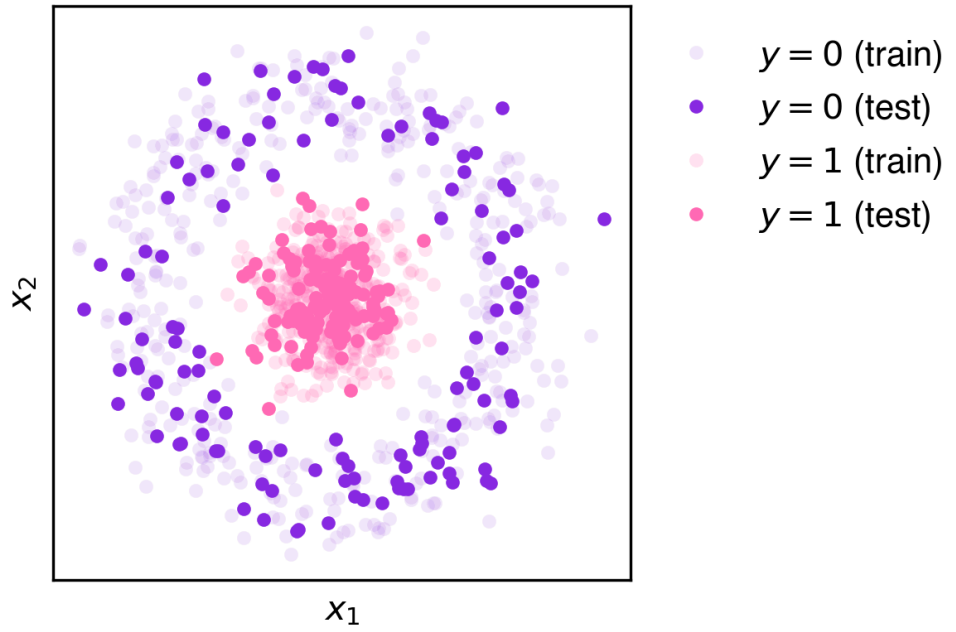
Underfitting: too simple model, does not capture the data structure.



Splitting the dataset into train and test sets

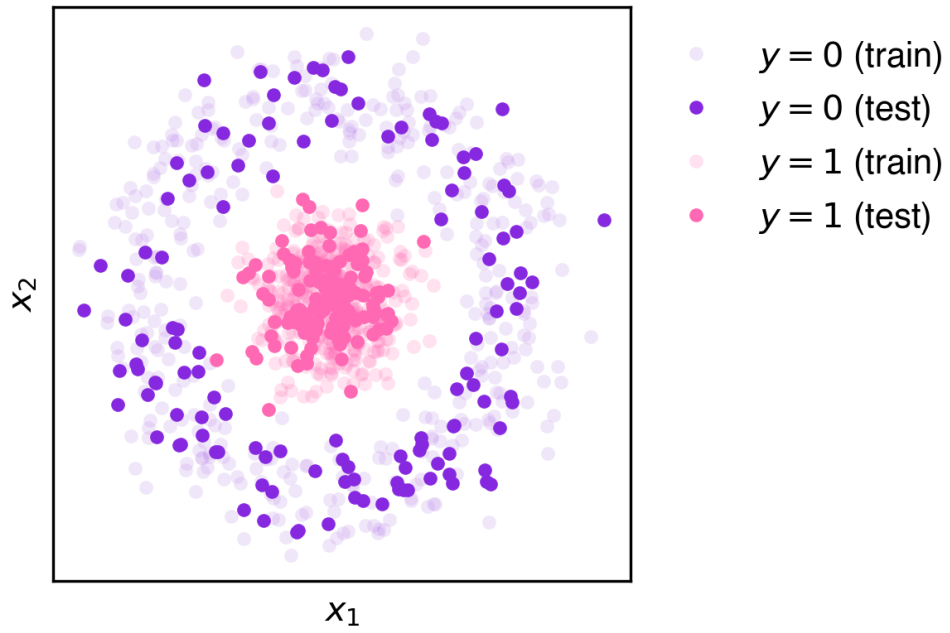
The **training set** is used to train the model. The **test set** is used to evaluate the model generalization error on unseen data.

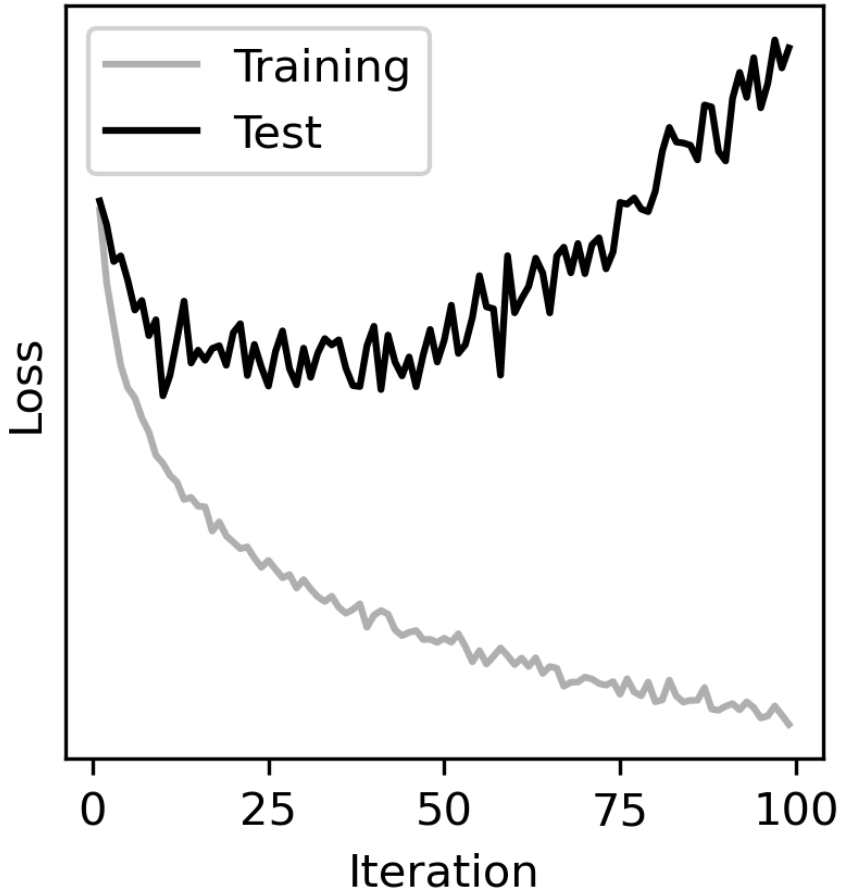
> The typical split is 80% for the training set and 20% for the test set.



Training and test learning curves

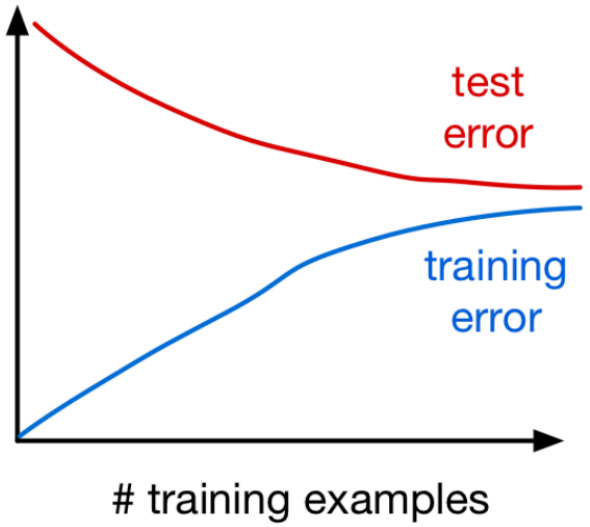
We must ensure that both the training and test losses decrease. If the training loss is much lower than the test loss, the model **overfits** the training set.

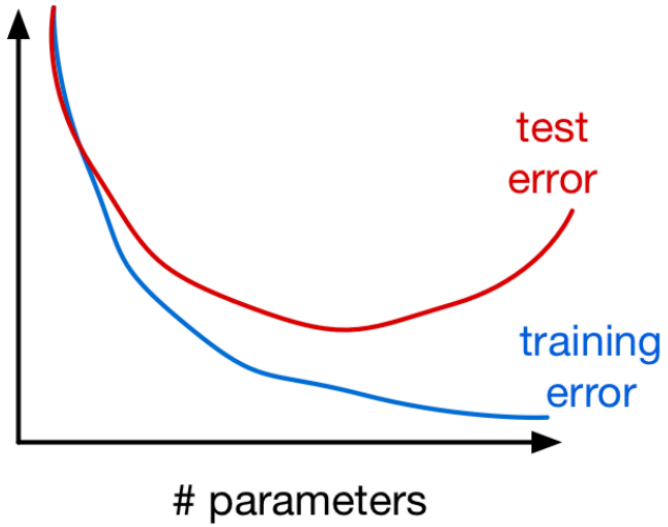




Targetting the right model complexity

The **model complexity** is roughly the number of parameters of the model. The **model generalization error** is the error on the test set.

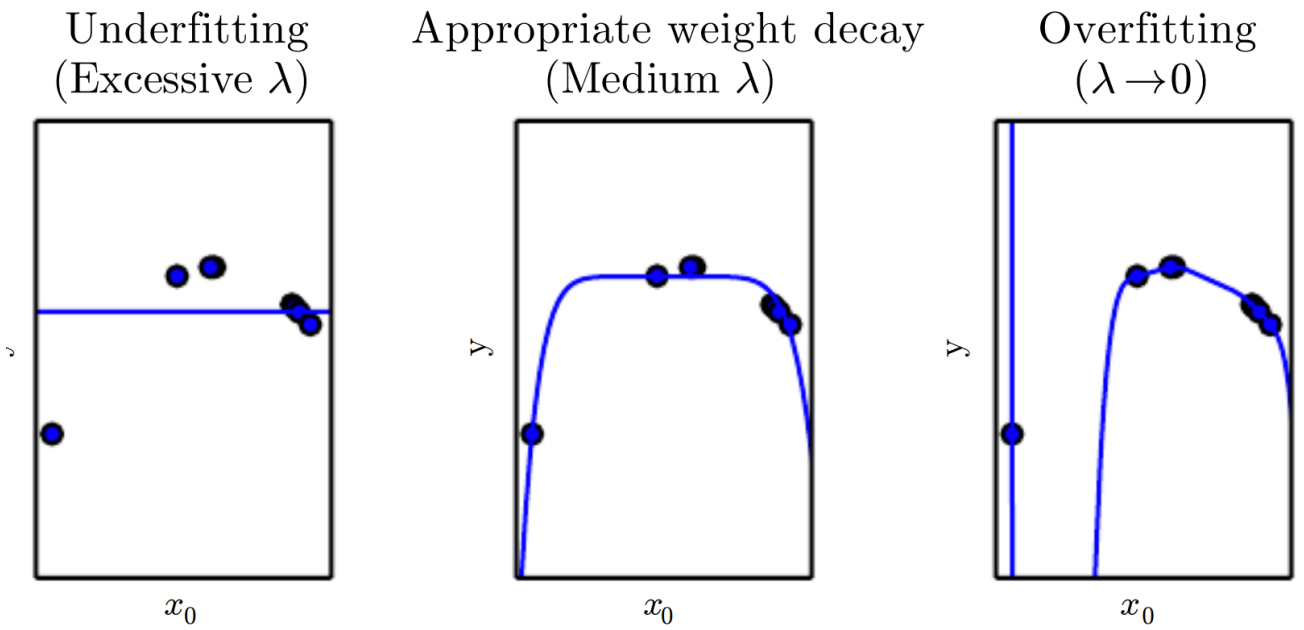




Regularization

Regularization is a technique to control overfitting by adding a penalty term $\mathcal{R}$ to the loss function. The **regularization parameter** λ controls the strength of the regularization.

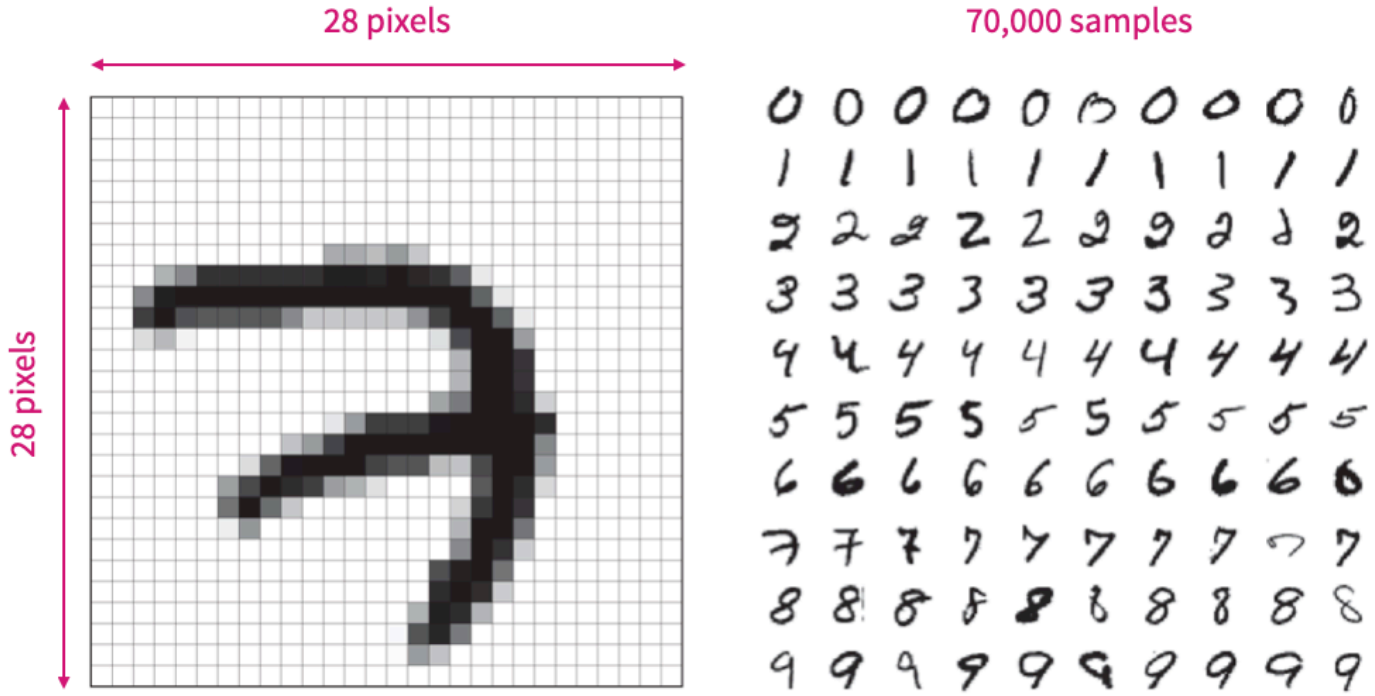
$$\mathcal{L}_{\text{reg}} = \mathcal{L} + \lambda\mathcal{R} = \mathcal{L} + \lambda\|\theta\|_2^2$$



From Goodfellow et al. (2016)

A fully connected network for solving the MNIST classification

Handwritten digits set of grayscale images $x \in \mathbb{R}^{28\times 28}$ and classes $y \in \{0, \ldots, 9\}$.



Goal: predict the number encoded in the pixels.

LeCun et al. (1998)

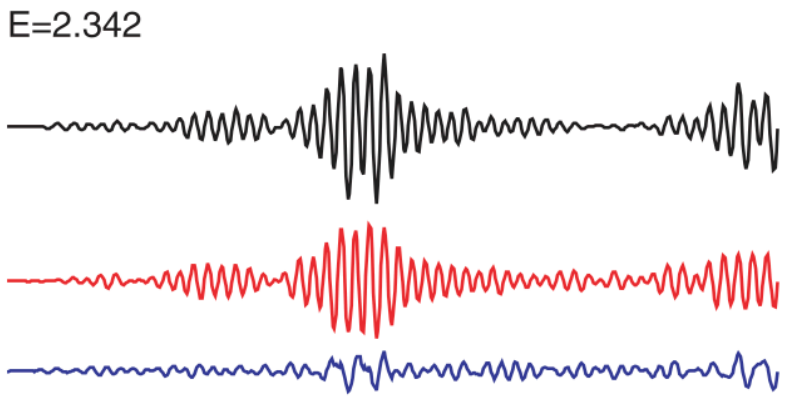
A fully connected network for solving the MNIST classification

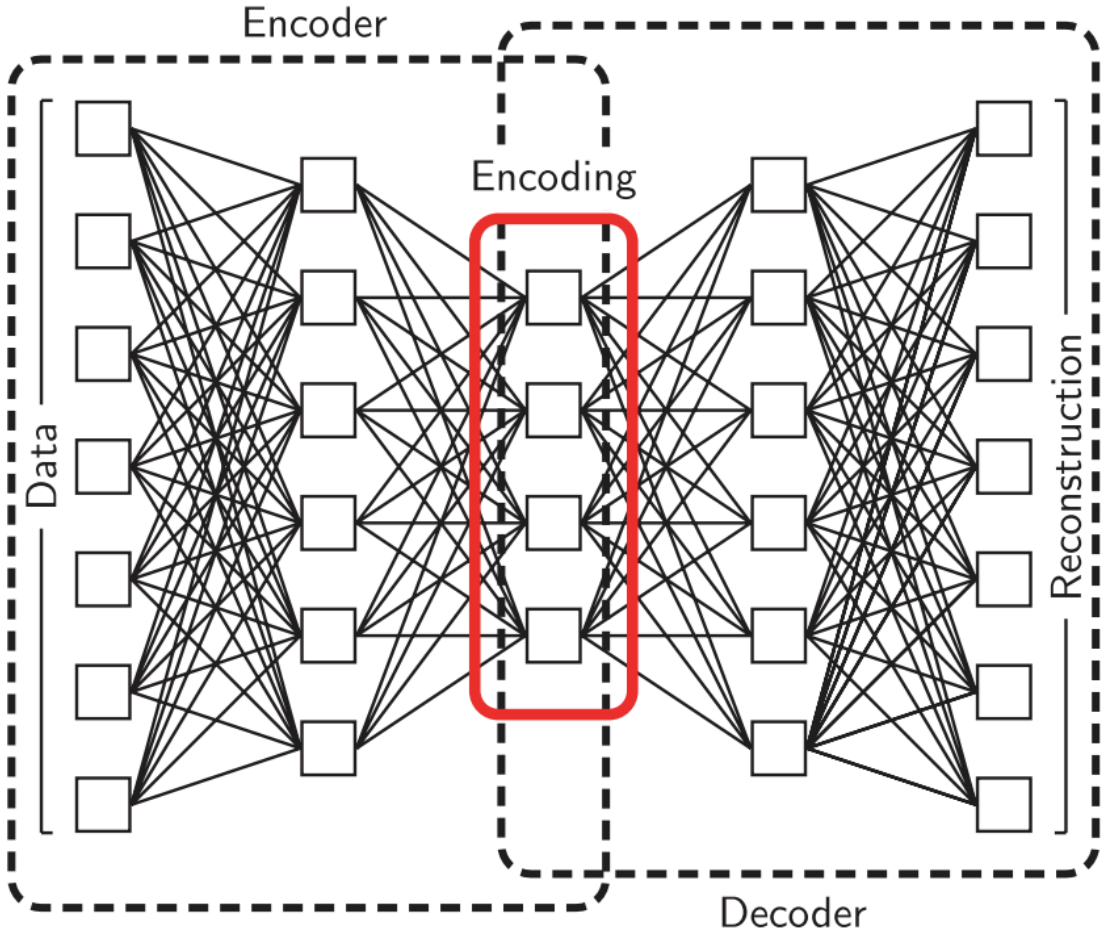
adamharley.com (Harley, 2015)

Example in seismology: fully-connected autoencoder

Training of a fully-connected autoencoder on real seismic data.

This is an **unsupervised** learning task: the input and output are the same.

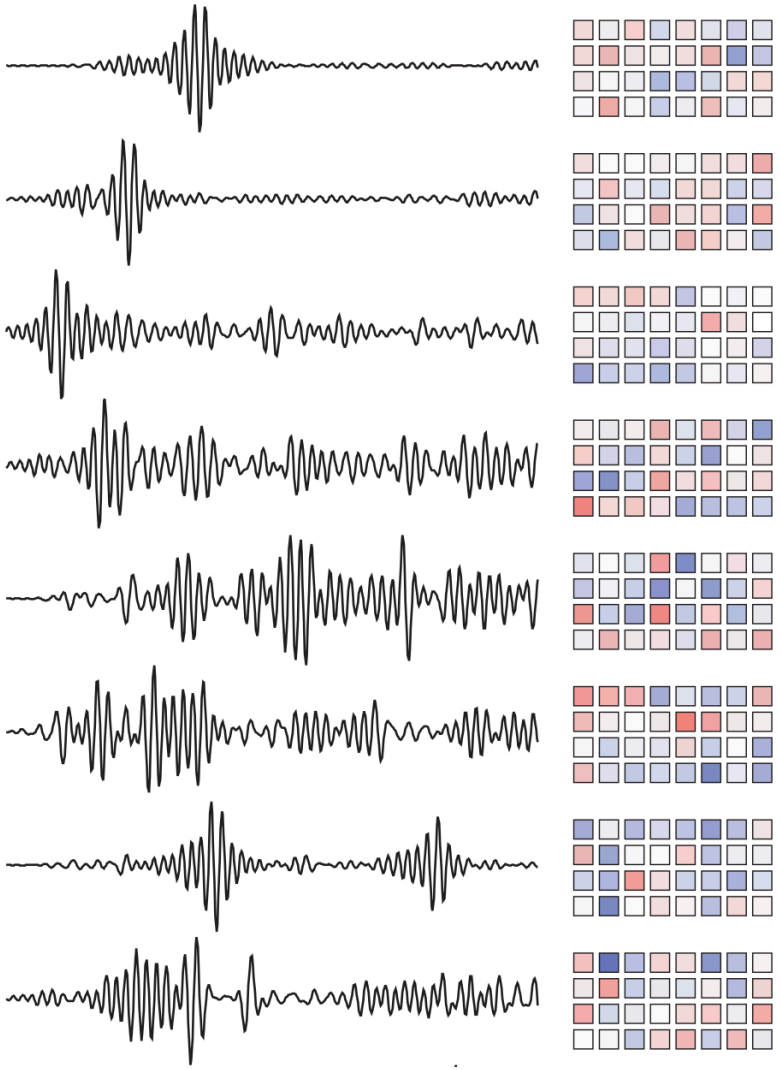




from Valentine and Trampert (2012)

Example in seismology: fully-connected autoencoder

We **learned** a low-dimensional representation for the seismic data.



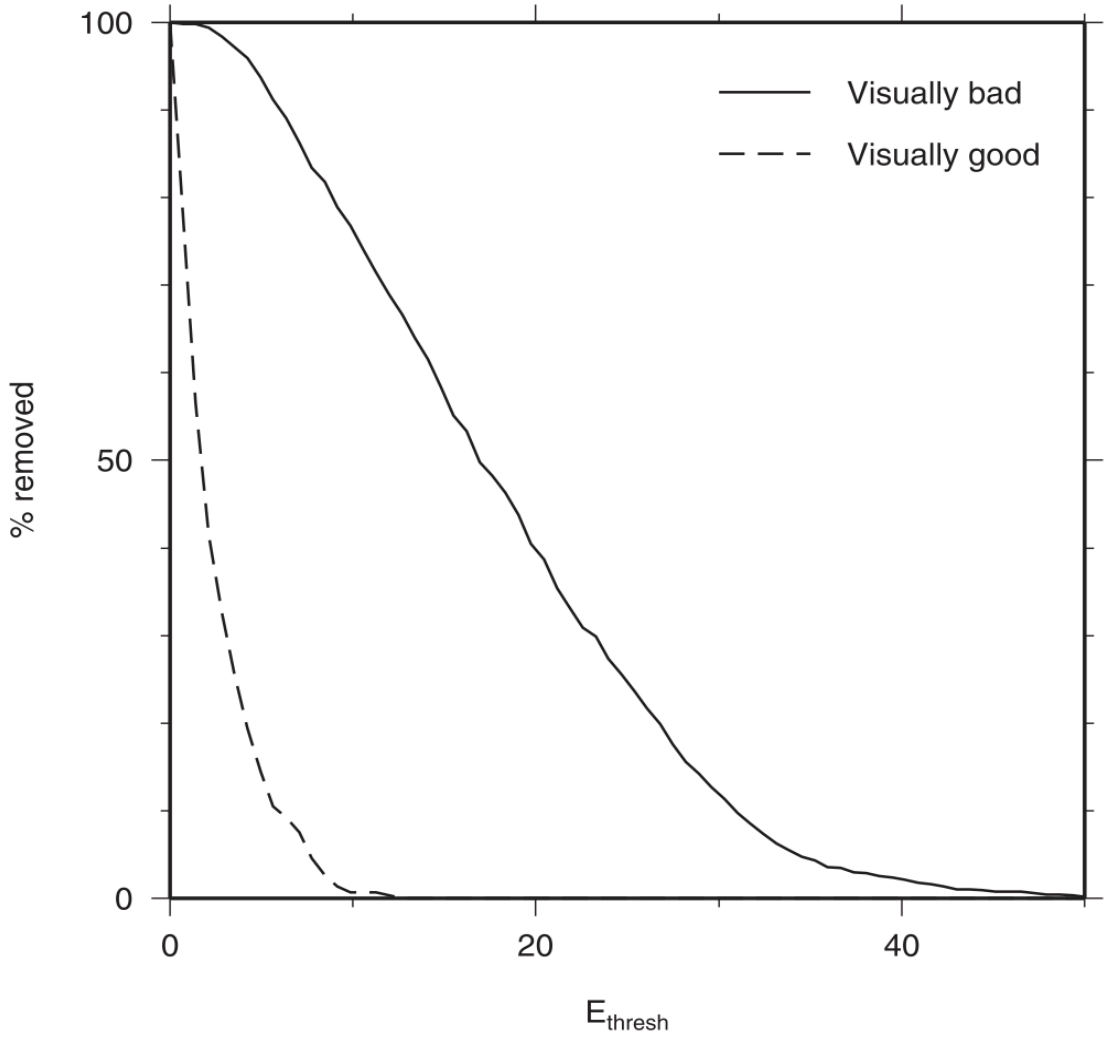
These are the **latent variables** of the autoencoder.

from Valentine and Trampert (2012)

Example in seismology: fully-connected autoencoder

Example applications:

- quality assessment
- compression

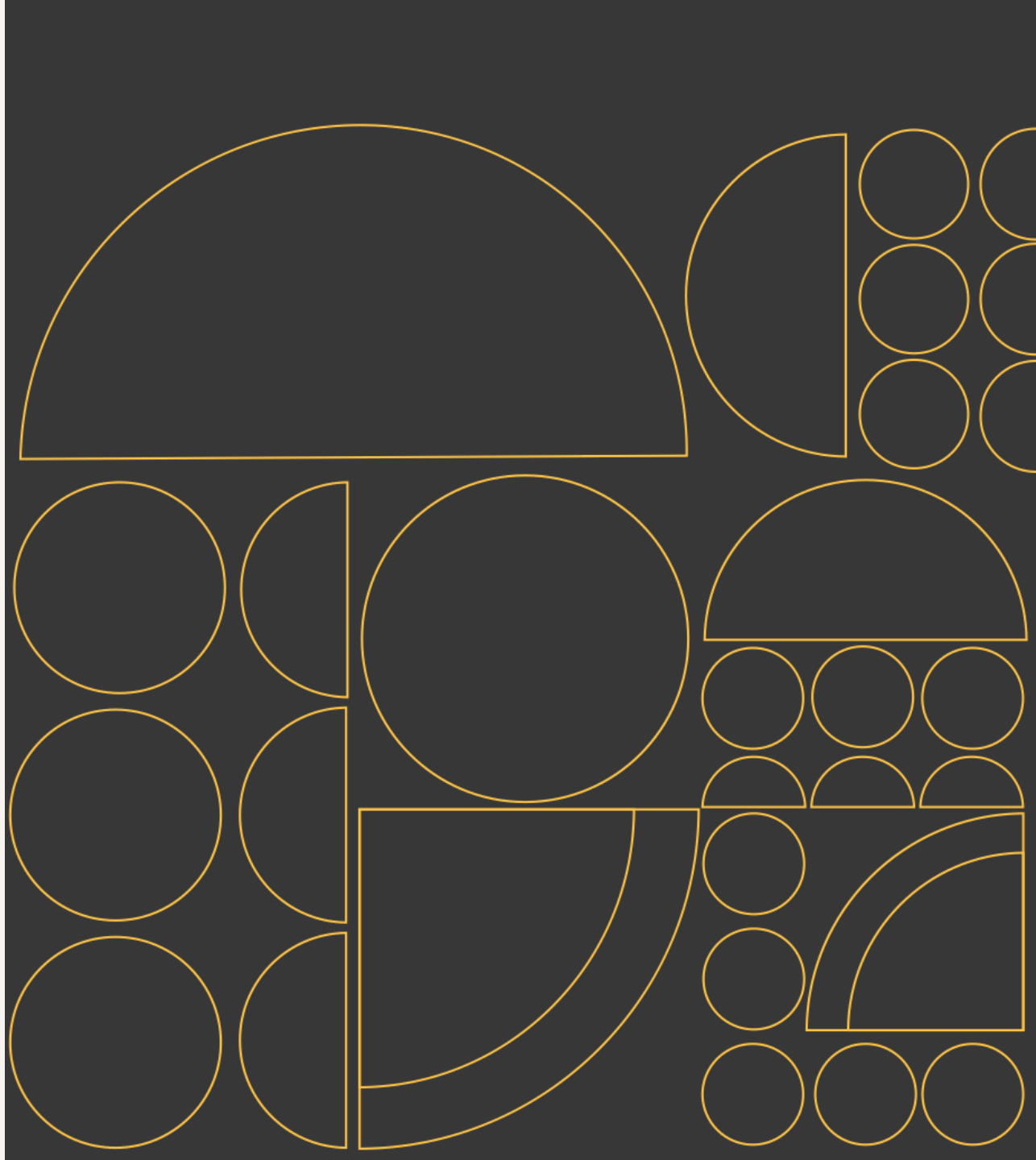


from Valentine and Trampert (2012)

ChEESE

6. Deep learning: convolutional neural networks

How deep learning works? What is a neural network? How to train it, and what for?

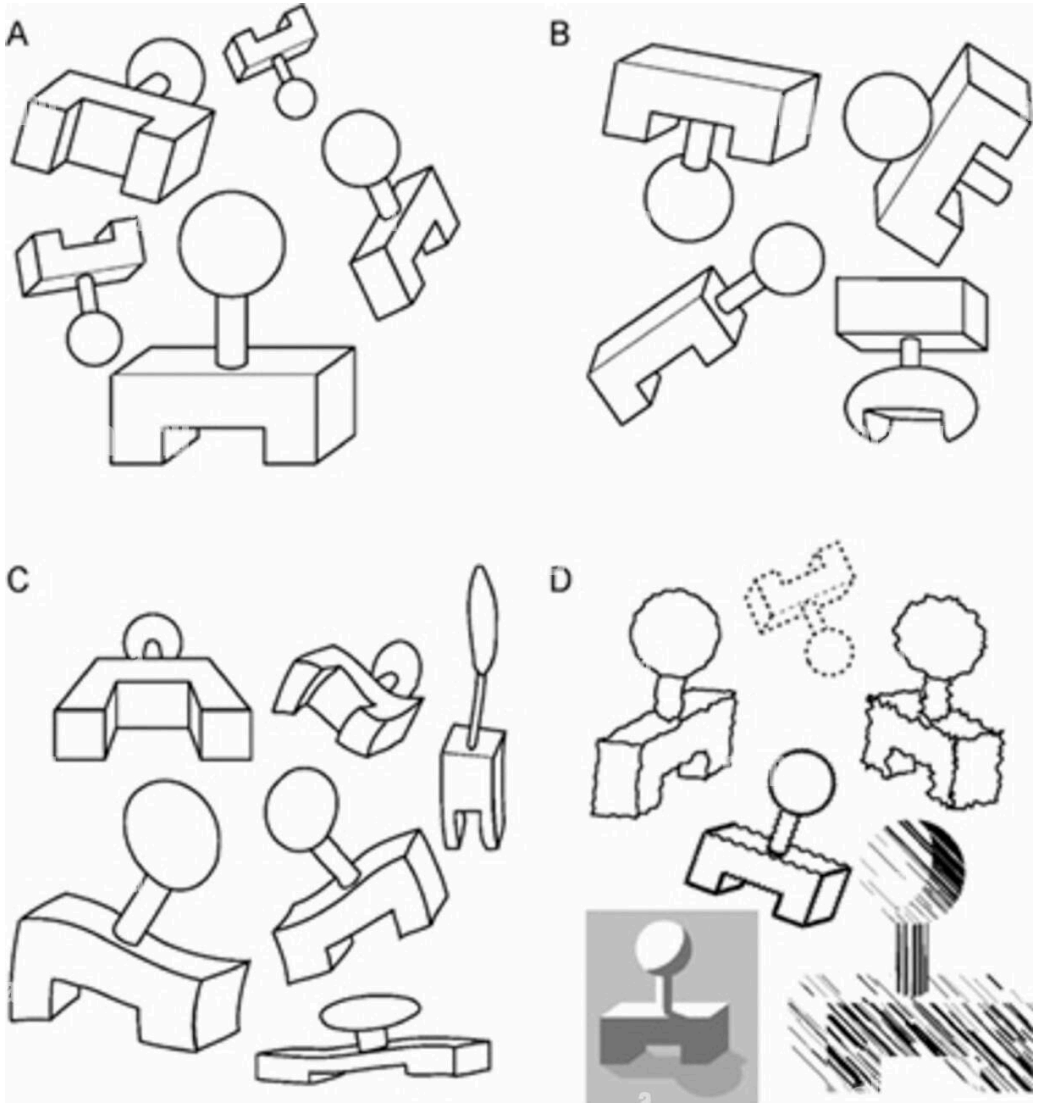


Limitations of fully connected networks

Vision is robust to a lot:

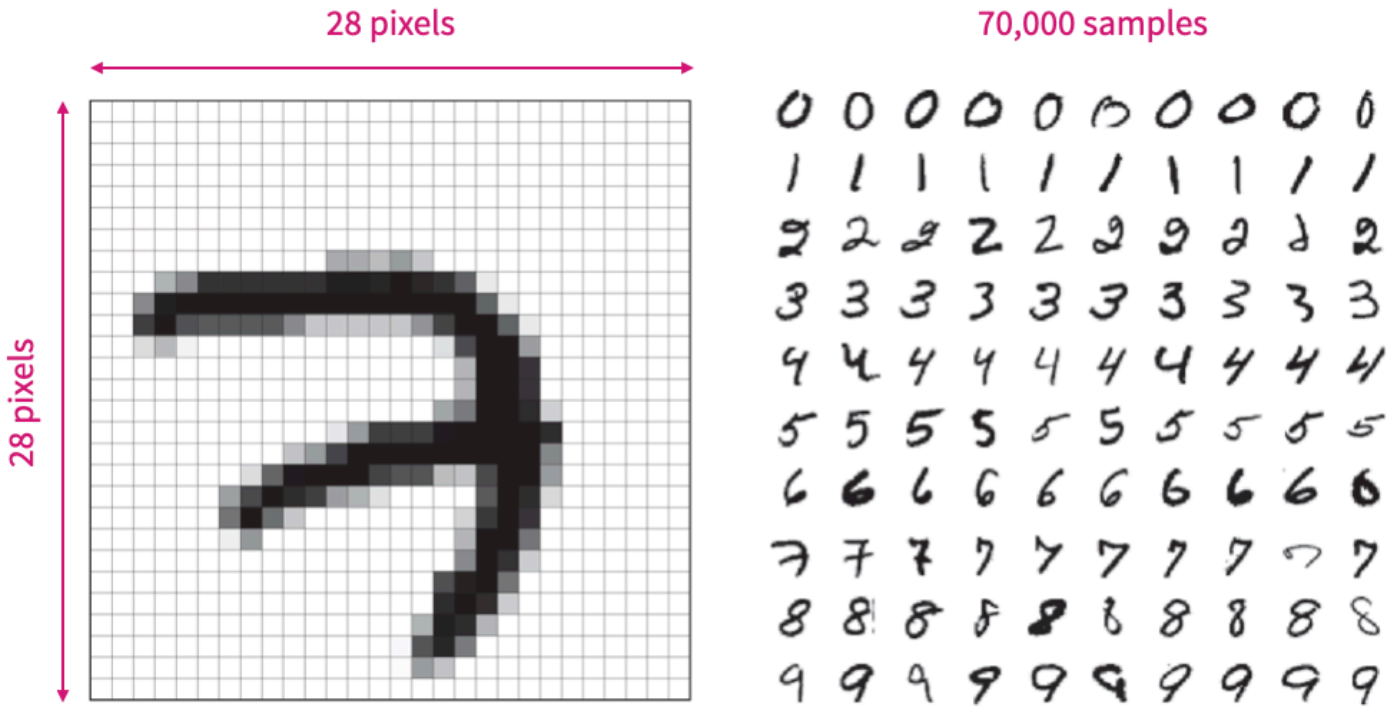
- Translation
- Rotation
- Scaling
- Shearing
- Illumination
- Occlusion

We need invariance to these transformations.



Example: the handwritten digits

Handwritten digits set of grayscale images $x \in \mathbb{R}^{28 \times 28}$ and classes $y \in \{0, \ldots, 9\}$.

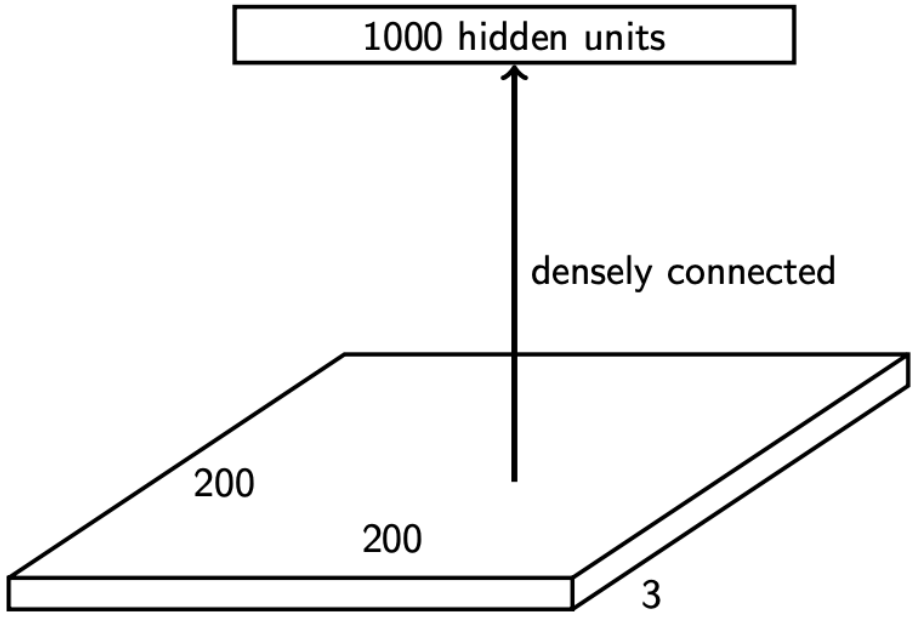


LeCun et *al.* (1998)

Limitations of fully connected networks

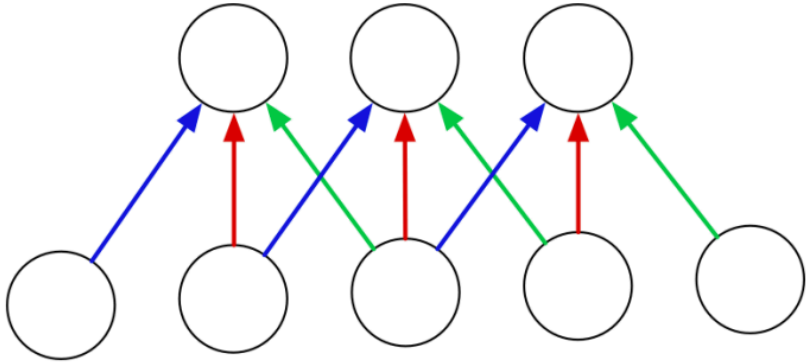
An image may be of 200×200 pixels $\times 3$ color channels. With a **fully connected network** with 1000 hidden units, we would have $N = 200 \times 200 \times 3 \times 1000 = 120$M parameters.

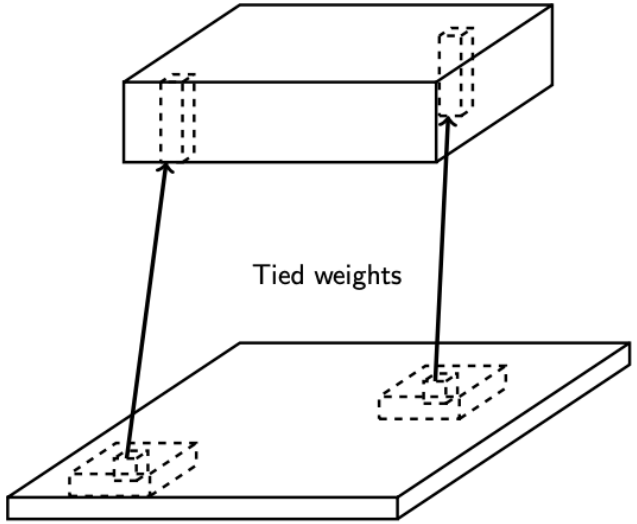
This clearly does not scale to large images.



Convolutional neural networks

Convolutional layers are a type of layer that are used in convolutional neural networks. They are composed of a set of learnable filters.





Each hidden unit look a local content from the input image, althought the weights are shared across the entire image.

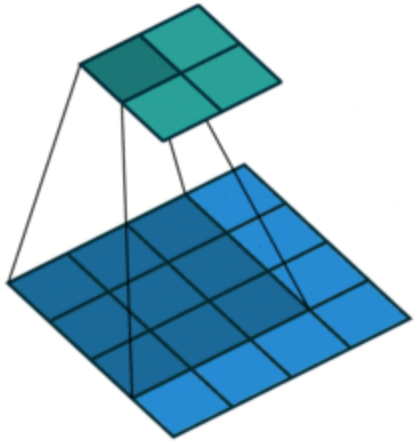
Convolutional neural networks

Discrete image convolution:

$$(A * B)_{ij} = \sum_n \sum_m A_{nm} B_{i-n,j-m}$$

where A is a input image, and B is a convolutional kernel (weights) to learn.

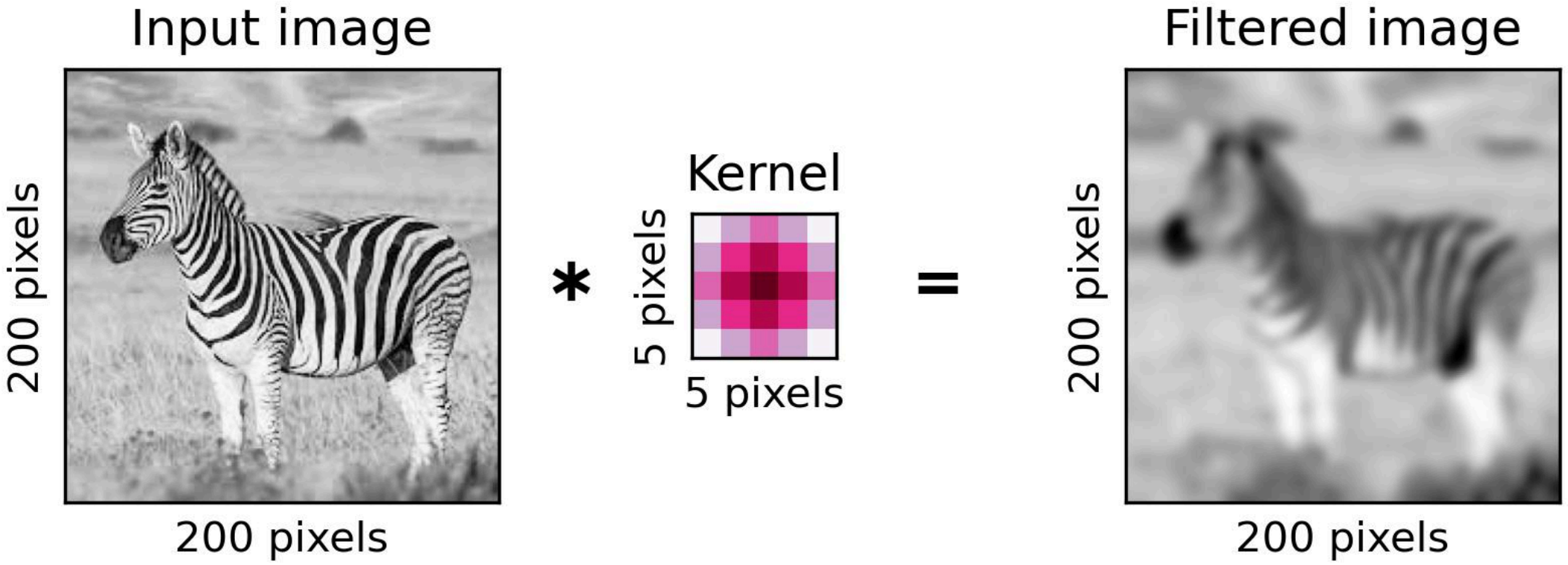
> Convolutional layers extract local features from the input image ≠ fully connected layers that extract global features.



From Vincent Dumoulin, Francesco Visin (2016)

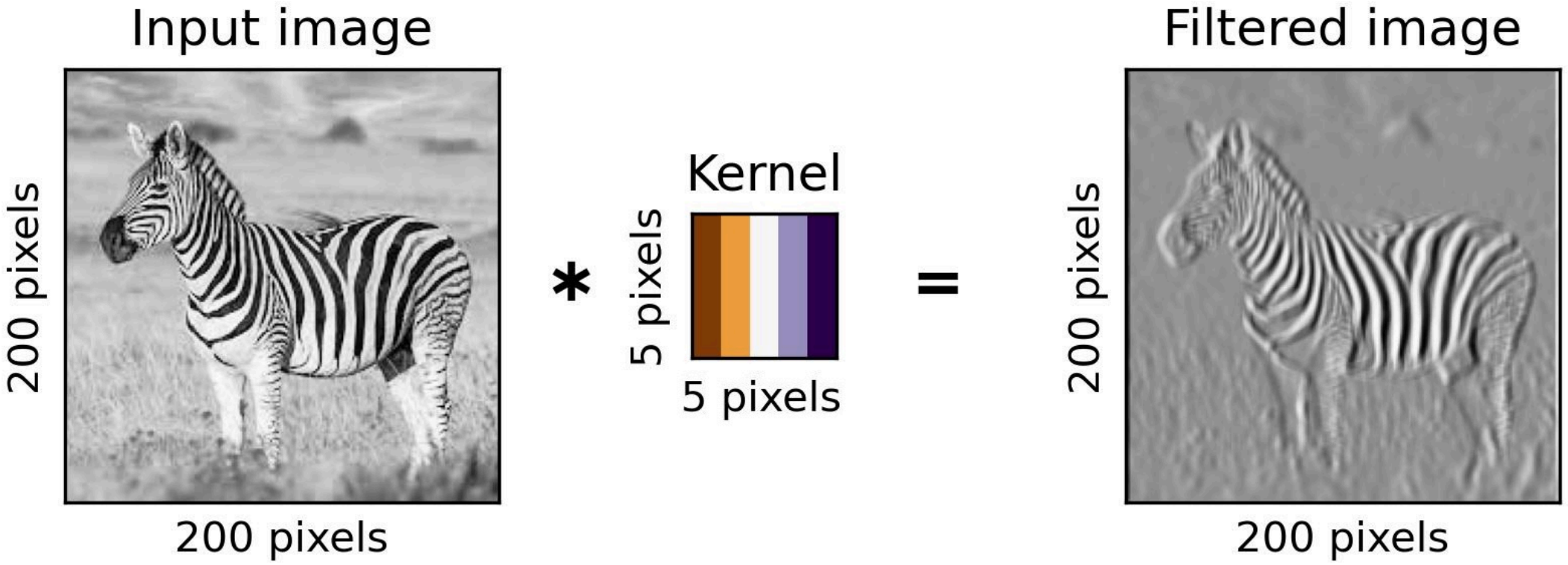
Convolution operation

$$(A * B)_{ij} = \sum_n \sum_m A_{nm} B_{i-n,j-m}$$

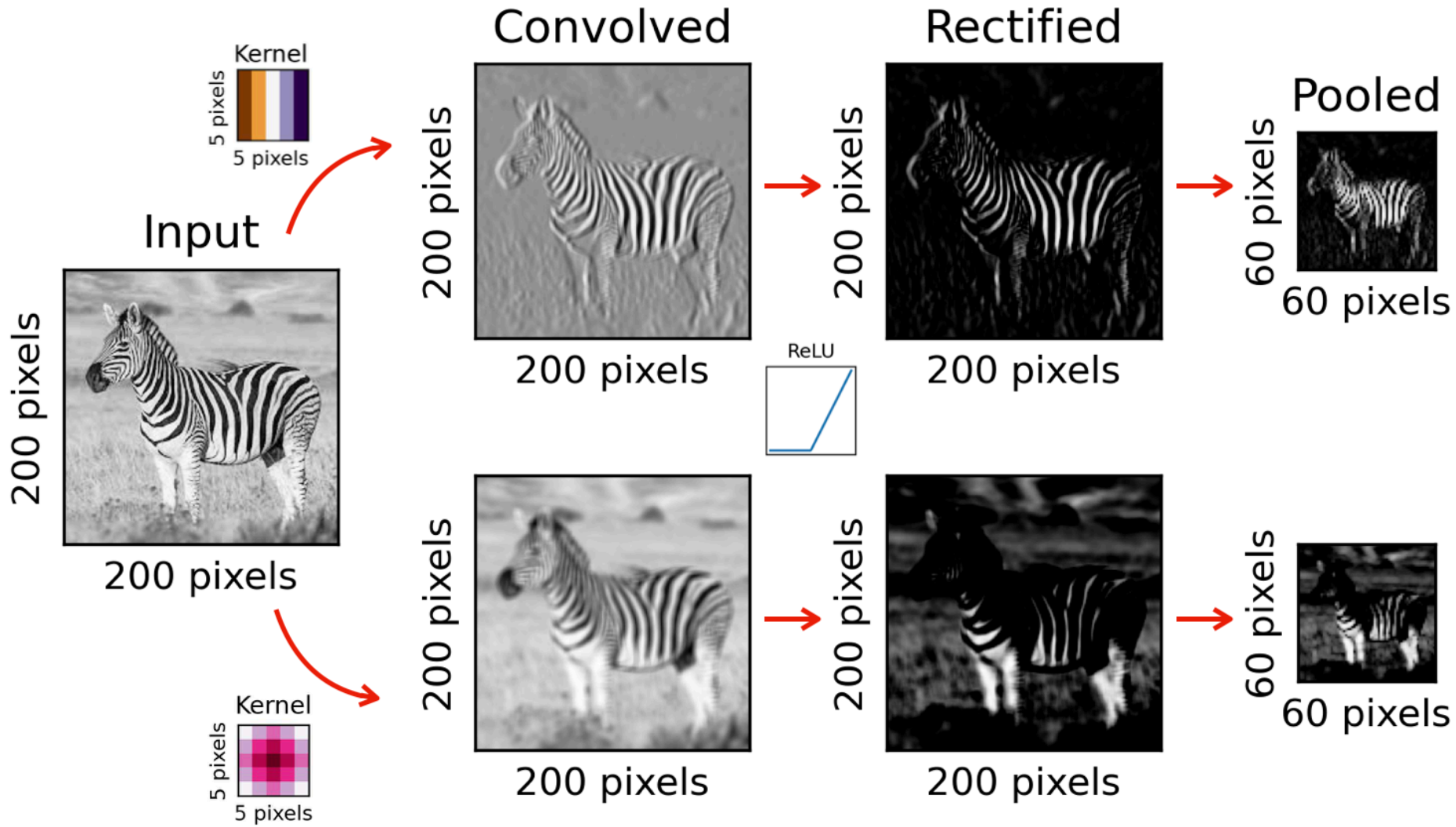


Convolution operation

$$(A * B)_{ij} = \sum_n \sum_m A_{nm} B_{i-n,j-m}$$

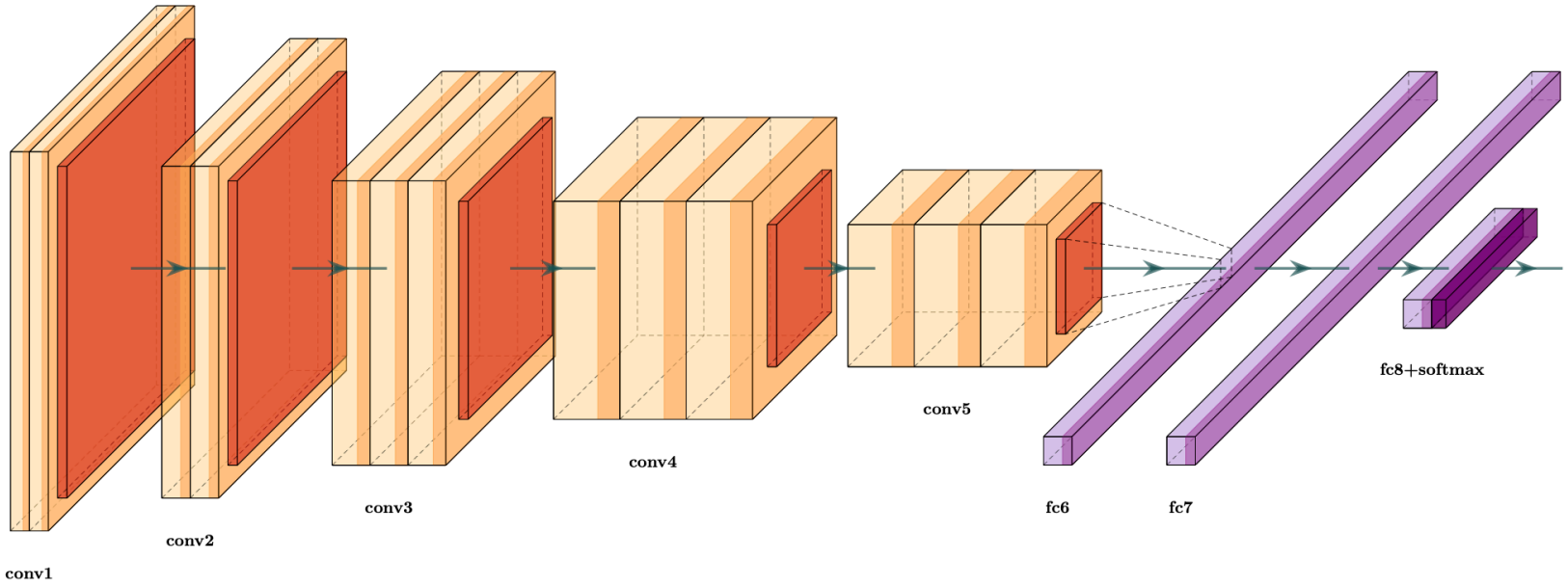


Convolution unit



Convolutional neural network: example with VGG16

Now, we can understand this winning architecture for image classification.

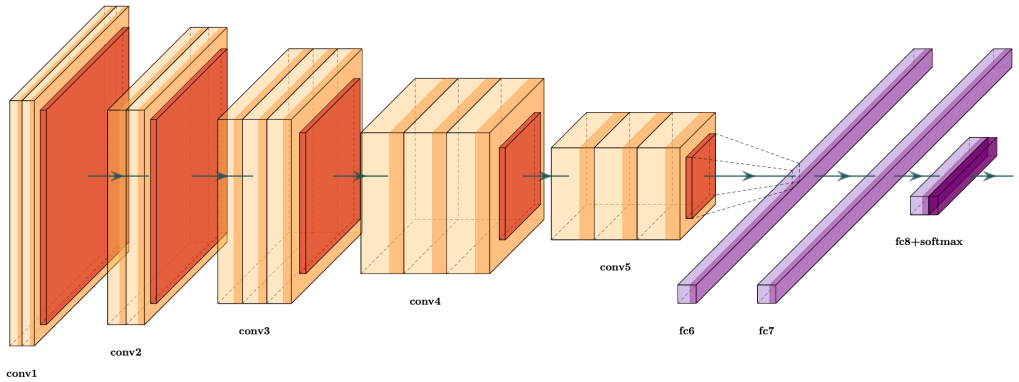


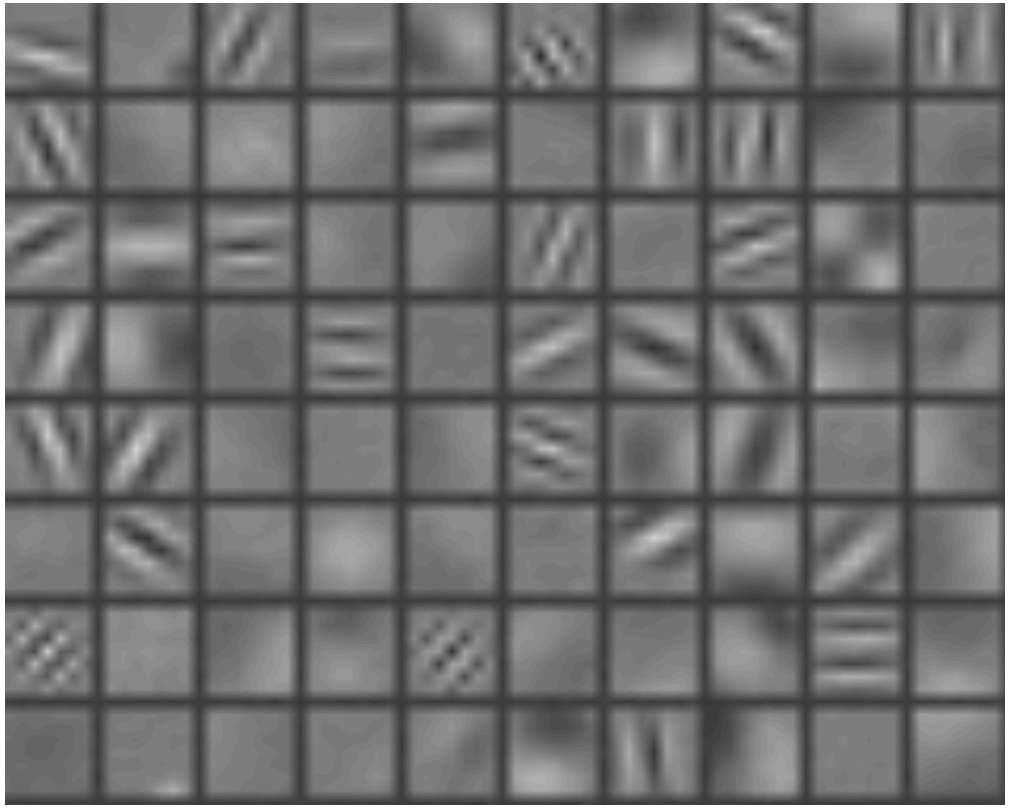
Note the last three layers are **fully connected**.

When extracting low-dimensional data from images, this is often needed.

Convolutional neural network: example with VGG16

Here are the **filters from the first layer** of VGG16 after training on 100k+ images. These filters collect various shapes, scales, colors, etc.





from Zeiler and Fergus (2013)

A convolutional network for solving the MNIST classification

FPS

Draw your number here

×

Downsampled drawing:

First guess:

Second guess:

Layer visibility

Input layer

Convolution layer 1

Downsampling layer 1

Convolution layer 2

Downsampling layer 2

Fully-connected layer 1

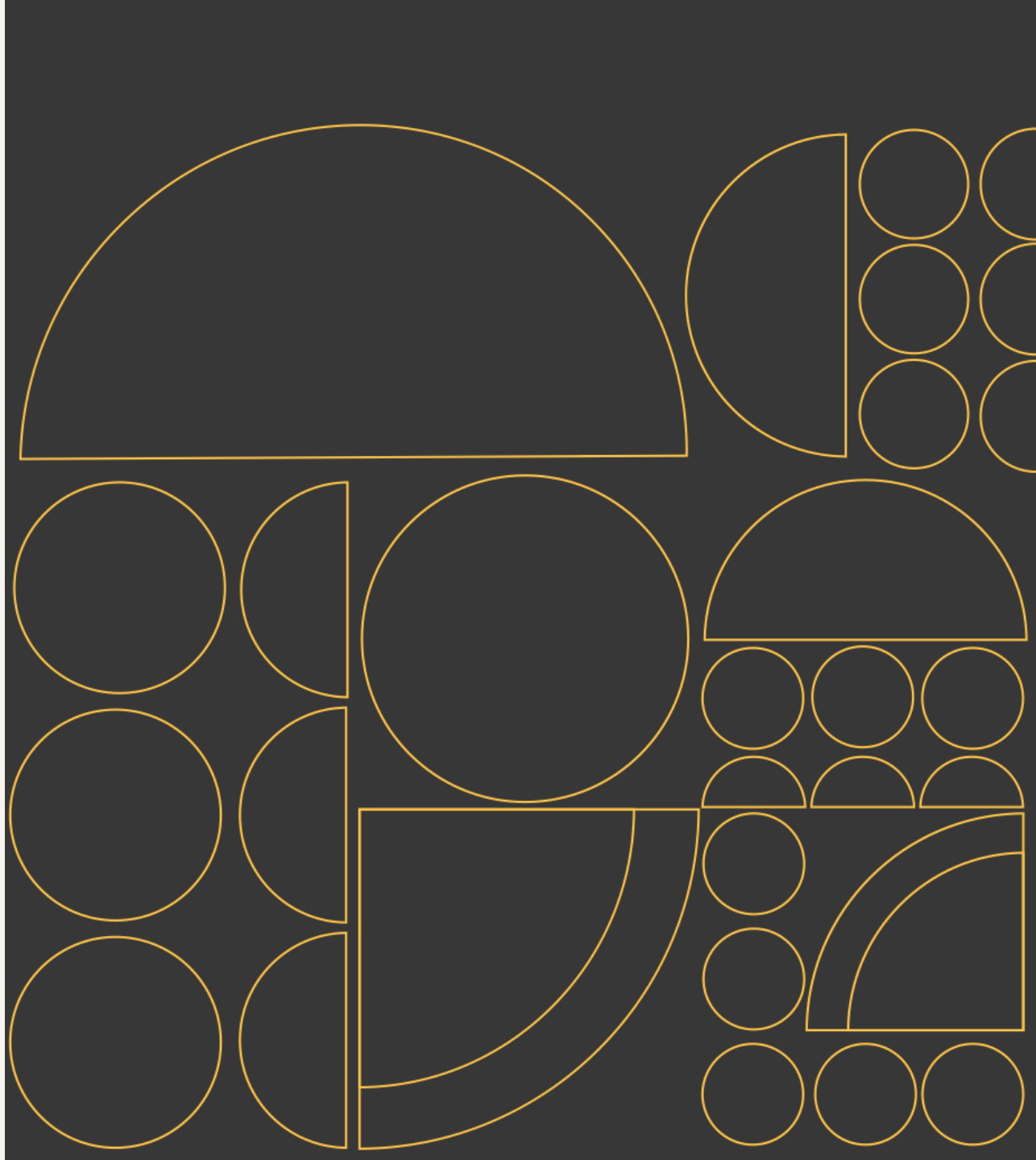
Fully-connected layer 2

Output layer

adamharley.com (Harley, 2015)

100

Made by Adam Harley. Project details.



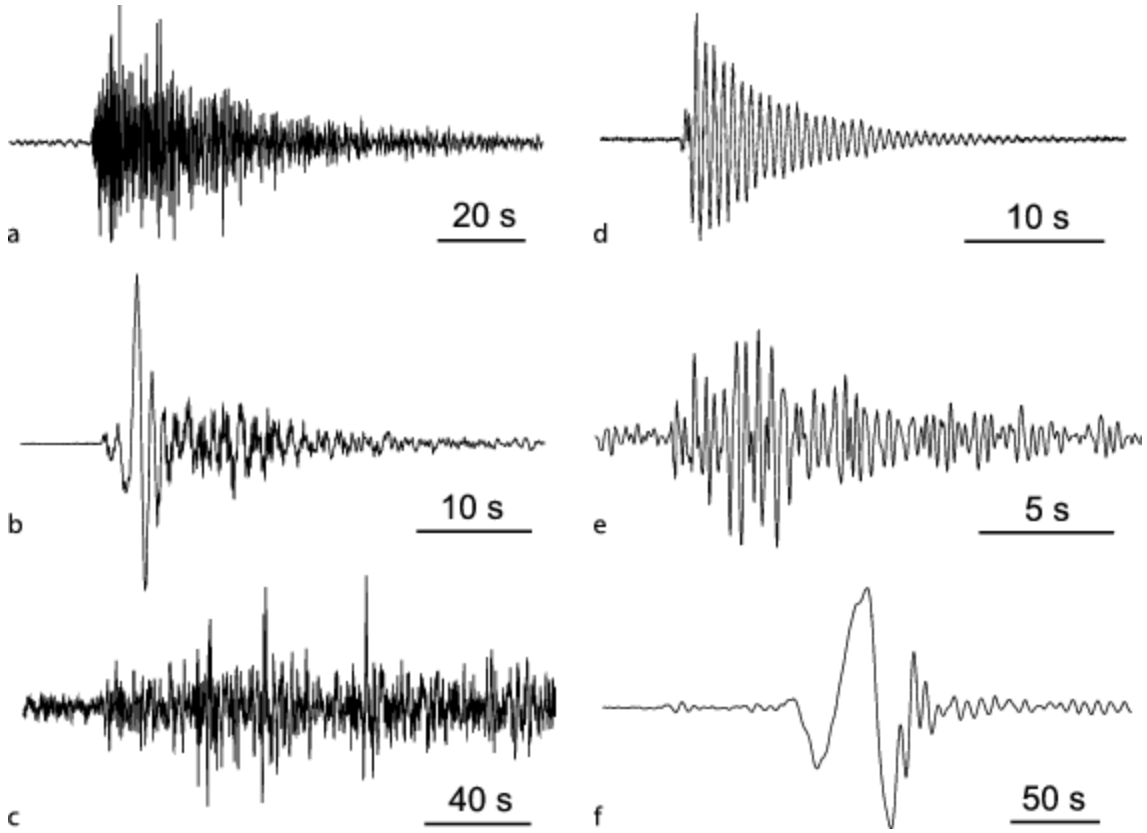
ChEESE

7. Applications

The illustration of the previous concepts with examples from seismology. And then you will be ready to apply these concepts to your own problems!

Deep-learning applications in seismology

- Signal detection, pattern recognition
- Classification
- Source localization from sparse or evolving datasets
- Denoising and compression



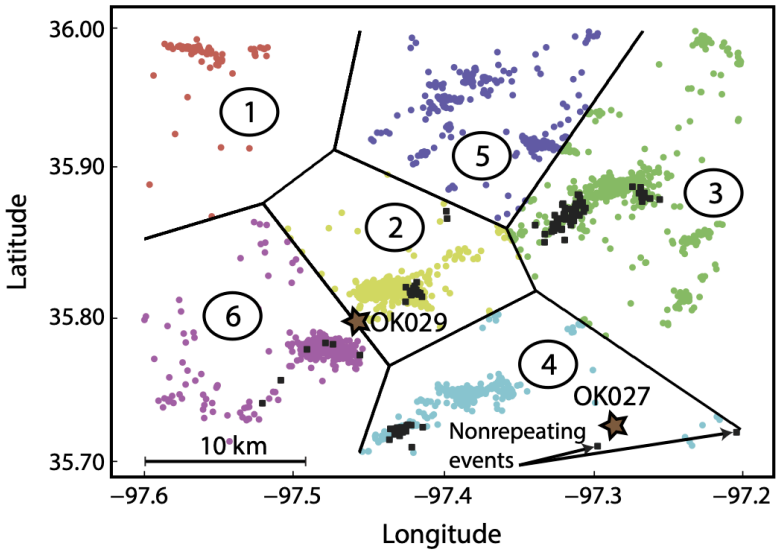
Earthquake detection and location with ConvNetQuake

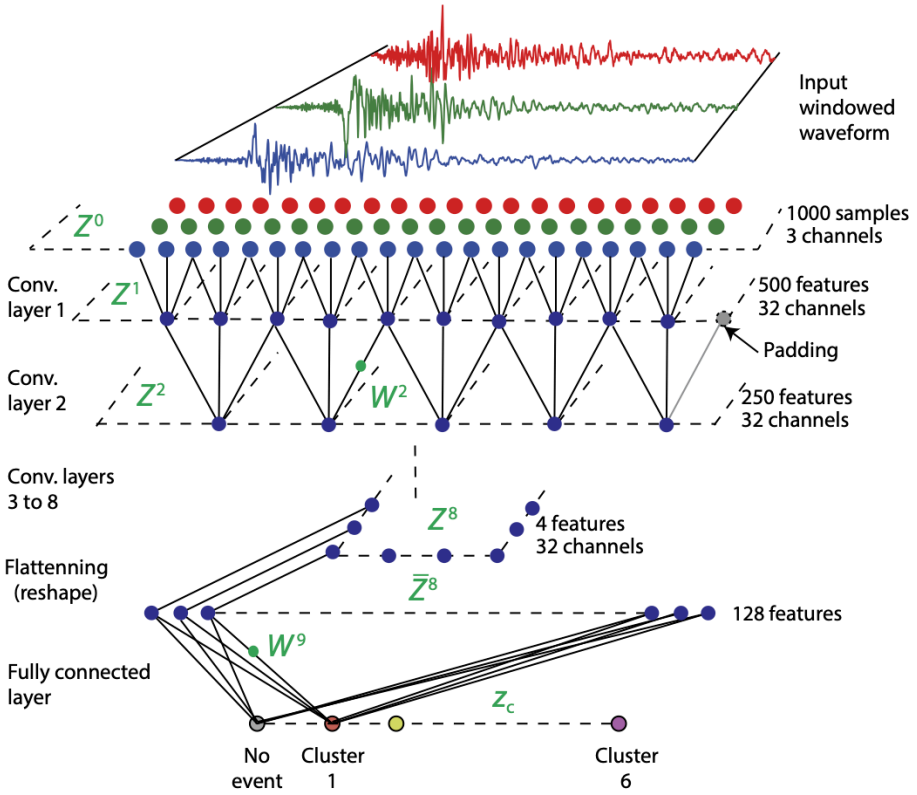
Features: 3-comp. waveform $x \in \mathbb{R}^{N \times 3}$

Target: prob. of event in cell 1 to 6

Loss: cross-entropy with regularization

$\mathcal{L} = -\sum_c q_c \log p_c + \lambda \|\mathbf{w}\|_2^2$





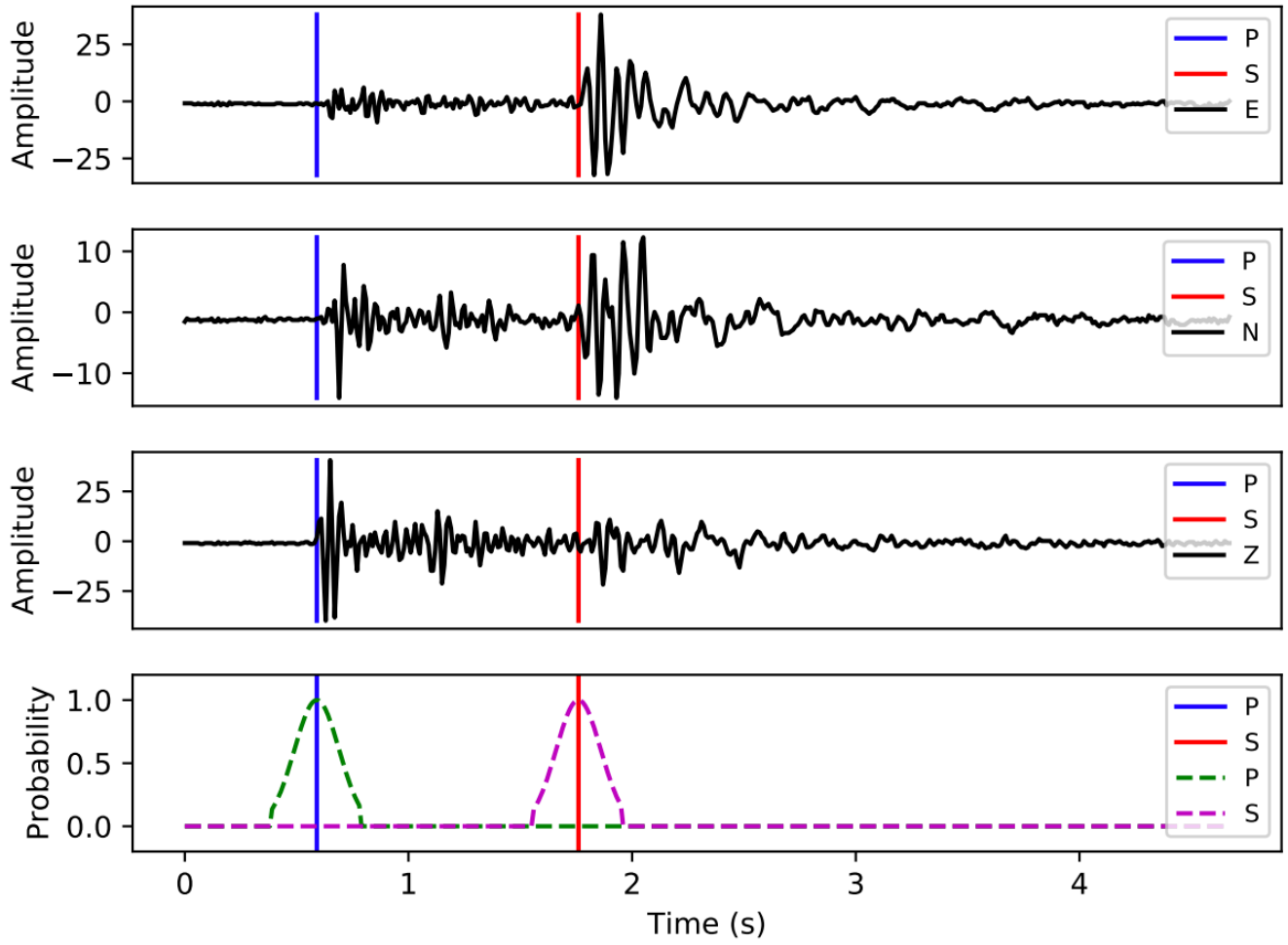
From Perol et al. (2016)

Seismic phase picking with PhaseNet

Features: 3-component seismic signal $x \in \mathbb{R}^{3000 \times 3}$

.

Targets: probabilities $p_i(x)$ of P, S, and Noise over time $= y \in \mathbb{R}^{3000 \times 3}$



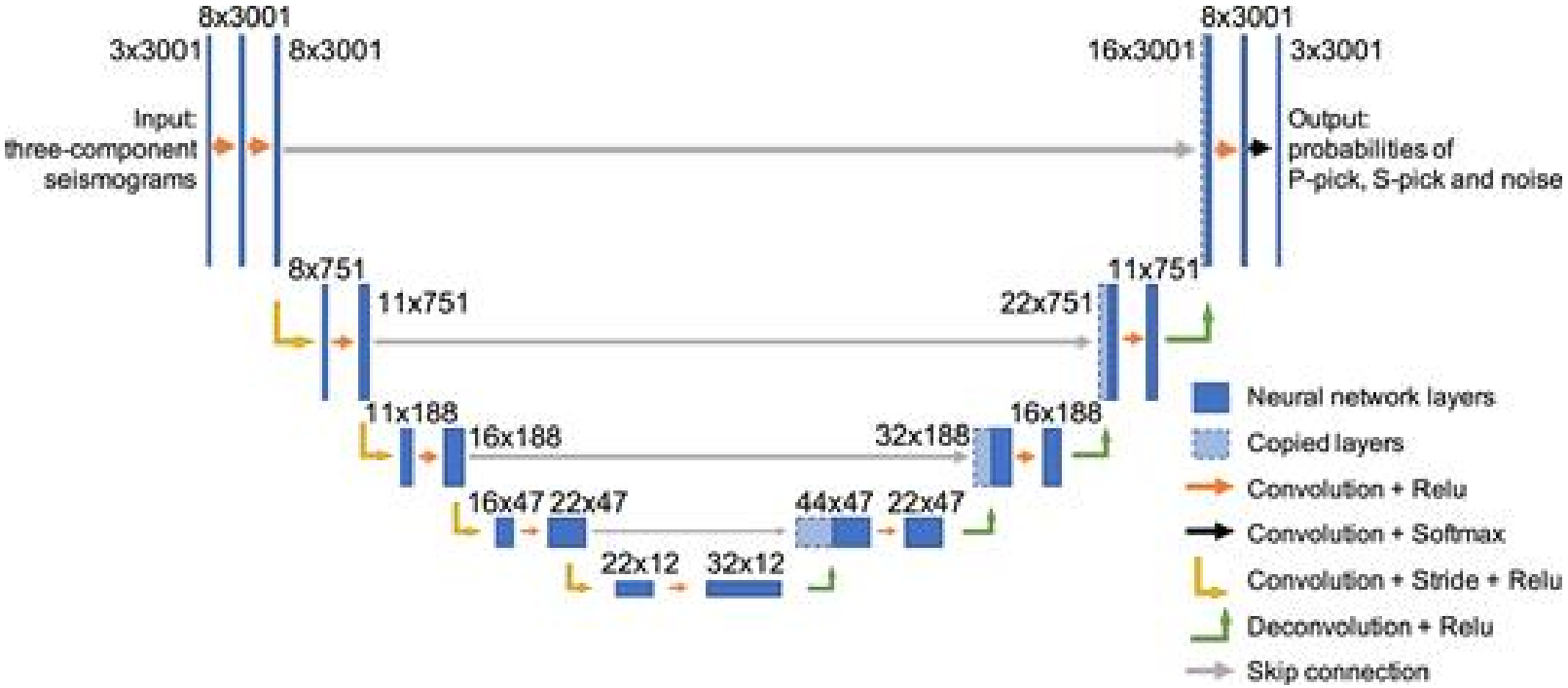
From Zhu et al. (2016)

Seismic phase picking with PhaseNet

Features: 3-component seismic signal $x \in \mathbb{R}^{3000 \times 3}$

Targets: probabilities $p_i(x)$ of P, S, and Noise over time $= y \in \mathbb{R}^{3000 \times 3}$

Loss: cross-entropy $\mathcal{L} = -\sum_i \sum_x p(x) \log(q(x))$

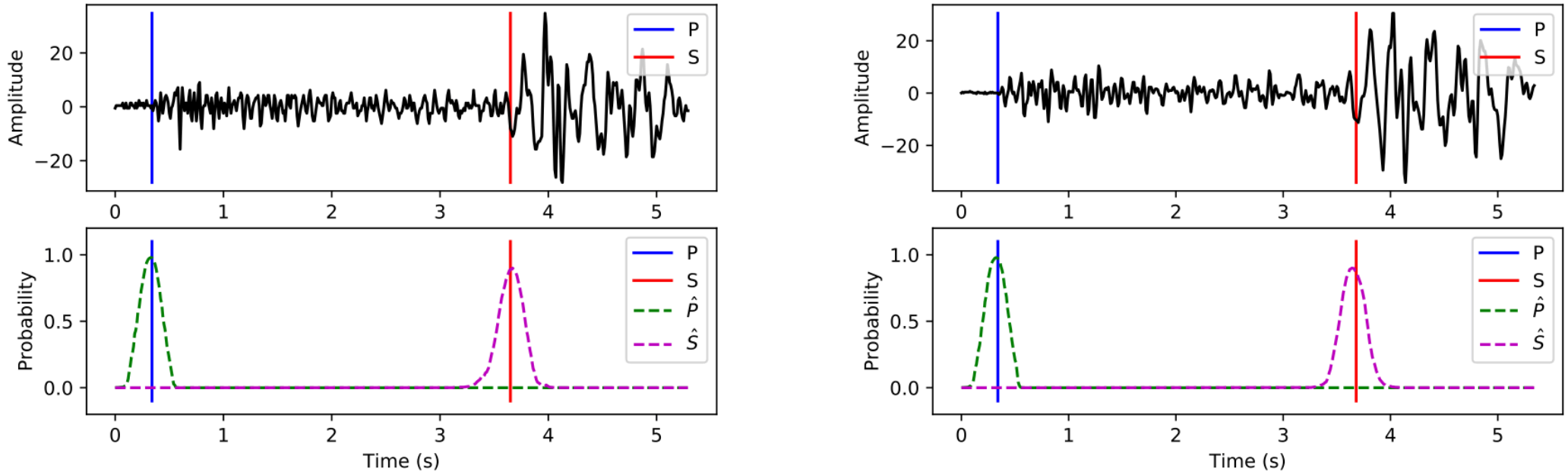


From Zhu et al. (2016)

Seismic phase picking with PhaseNet

Features: 3-component seismic signal $x \in \mathbb{R}^{3000 \times 3}$

Predictions: likelihood $q_i(x)$ of P, S, and Noise over time

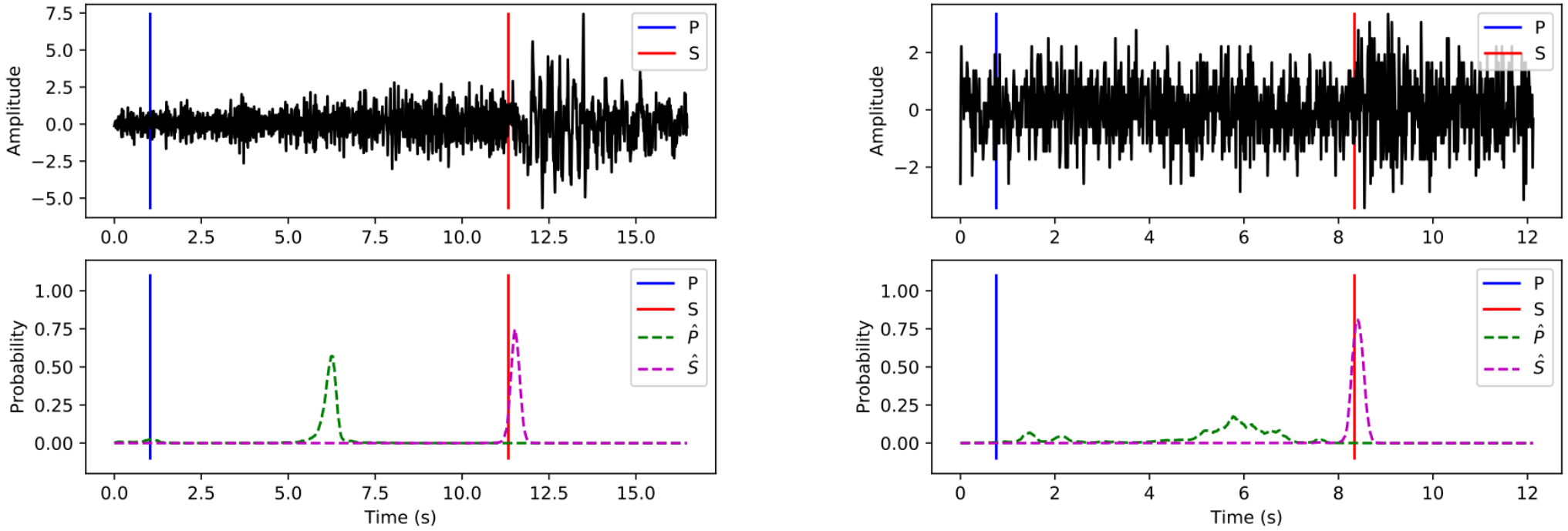


From Zhu et al. (2016)

Seismic phase picking with PhaseNet

Features: 3-component seismic signal $x \in \mathbb{R}^{3000 \times 3}$

Predictions: likelihood $q_i(x)$ of P, S, and Noise over time

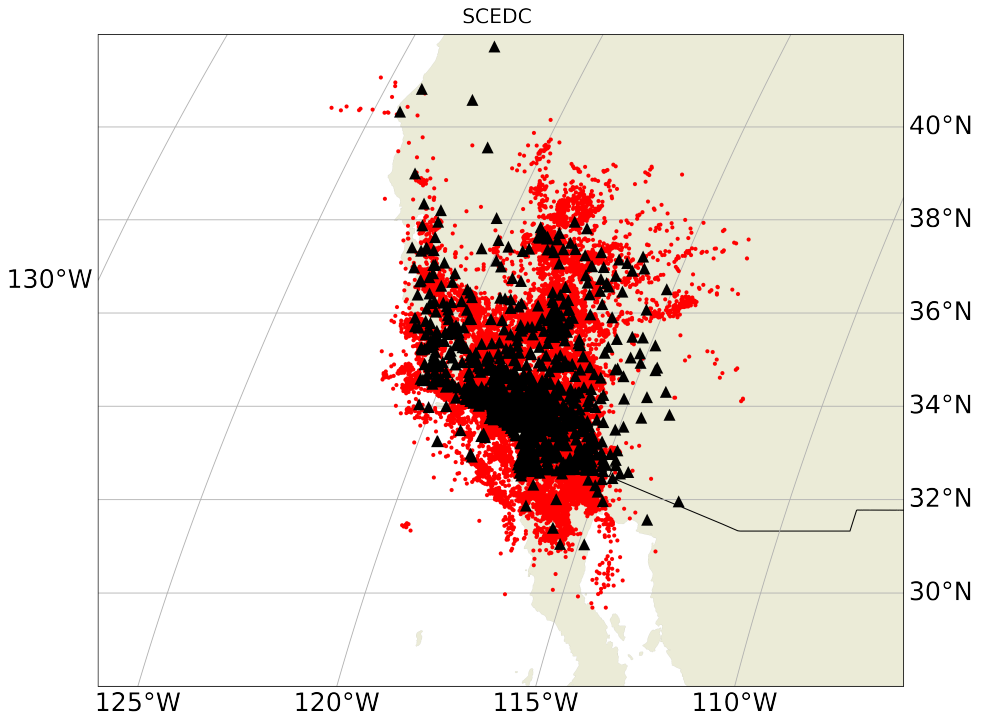


From Zhu et al. (2016)

Transfer learning and fine-tuning

Transfer learning is the use of a pre-trained model $f_\alpha = f_{\theta^*}$ on a new task as a initial point for training a new model $f_\alpha \rightarrow f_{\alpha^*}$.

Fine-tuning is the partial re-training of a pre-trained model on a new task, while keeping the weights of the pre-trained layers fixed.



Deep-learning libraries in Julia and Python

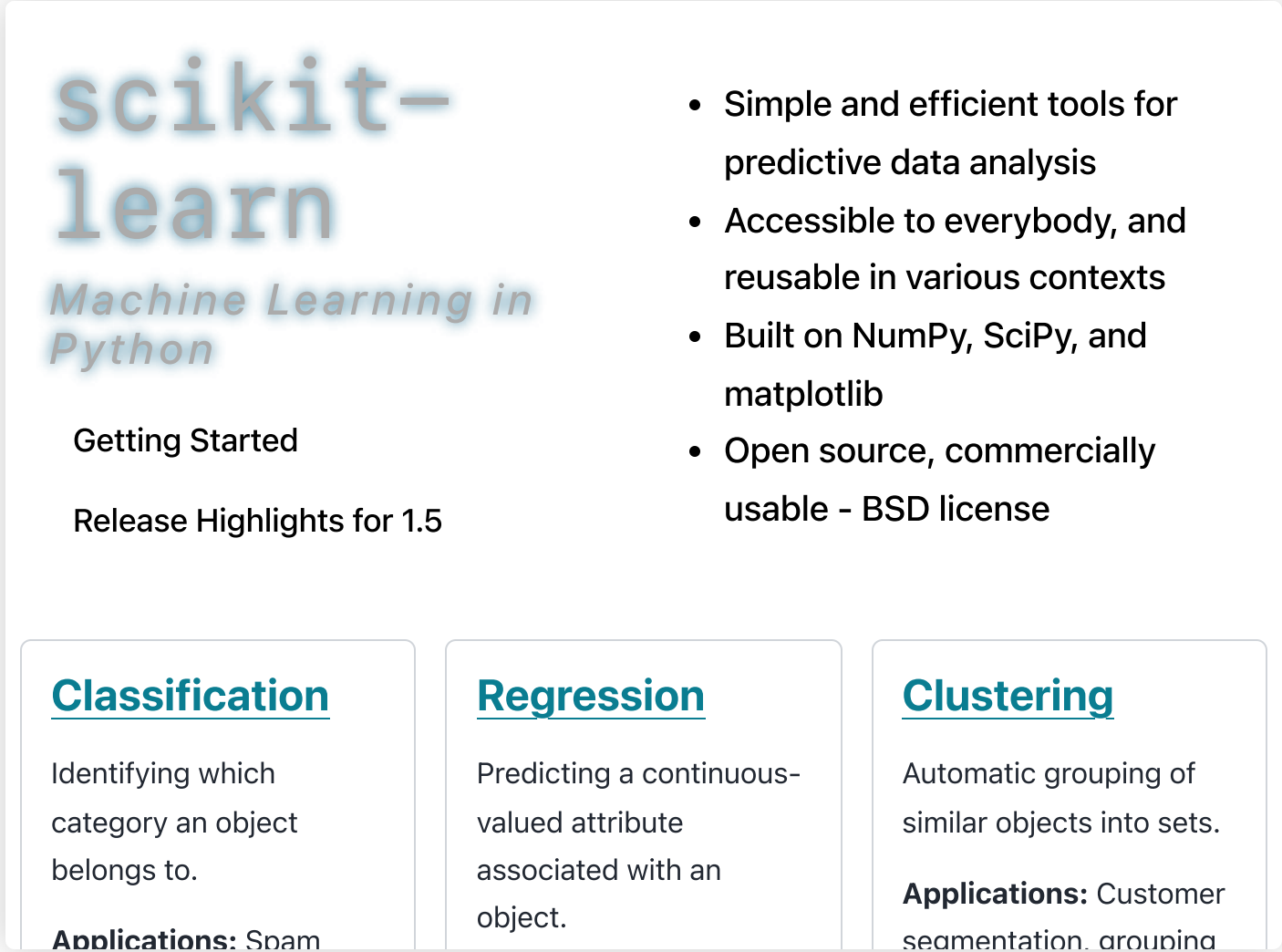
Warning

Libraries are constantly evolving, and the documentation is often incomplete.

Dive into the scikit-learn toolbox documentation

Scikit-Learn toolbox documentation

- Machine learning in Python
- Online examples
- Explanation of algorithms
- Grey-box models



Learning visually with the TensorFlow playground

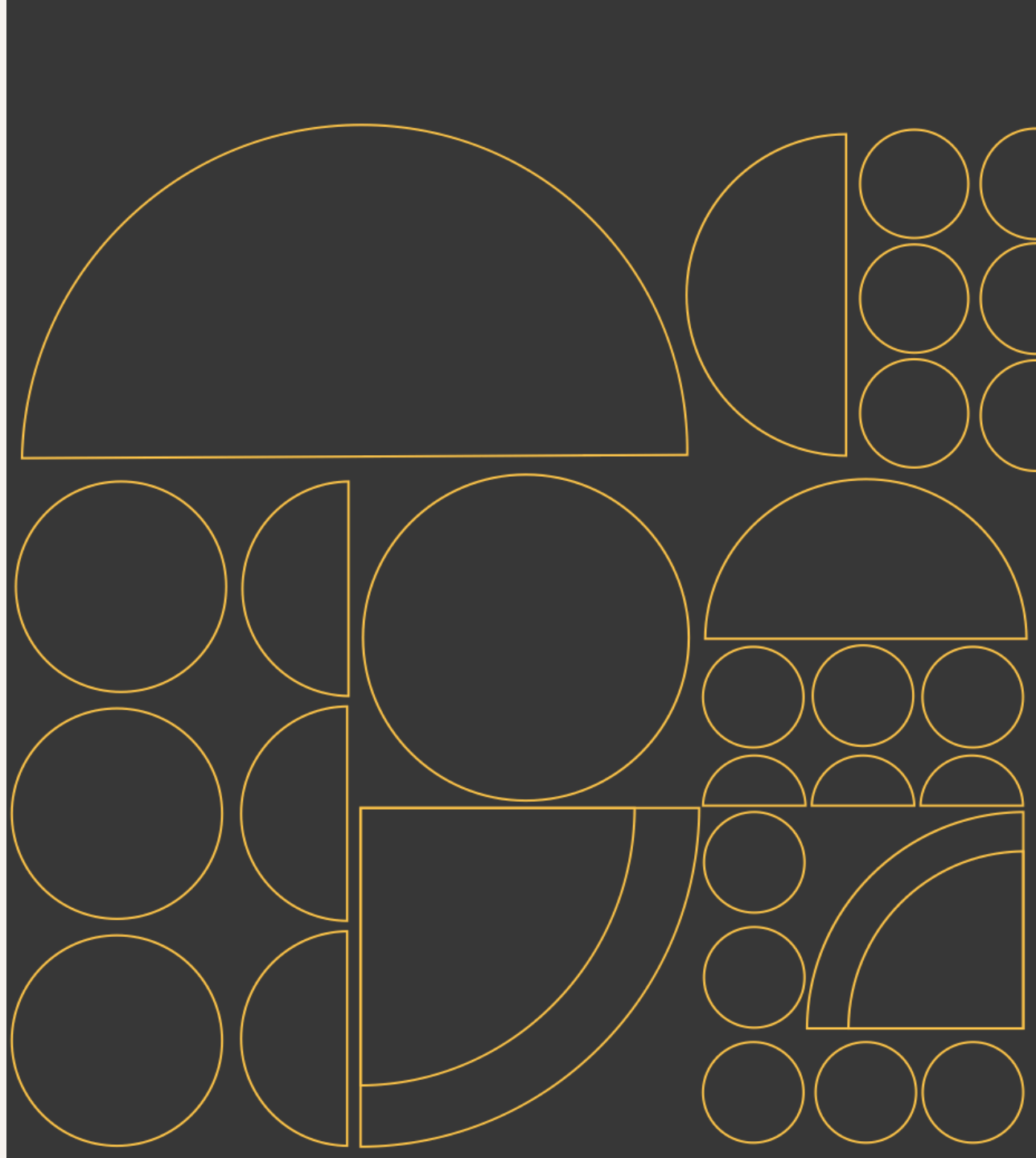
Interesting online projects

What do 50 million drawings look like?

Over 15 million players have contributed millions of drawings playing Quick, Draw! These doodles are a unique data set that can help developers train new neural networks, help researchers see patterns in how people around the world draw, and help artists create things we haven't begun to think of. That's why we're open-sourcing them, for anyone to play with.

Select a drawing

https://quickdraw.withgoogle.com/data

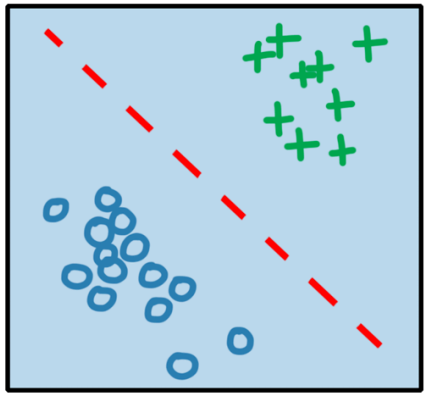


ChEESE

8. Unsupervised learning

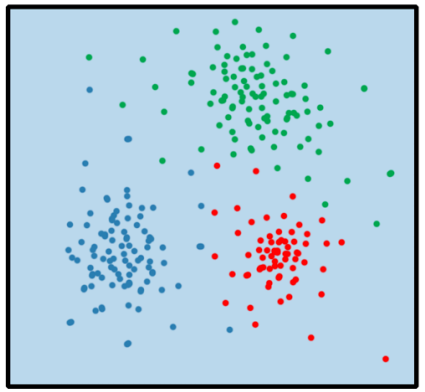
Main types of machine learning

supervised learning



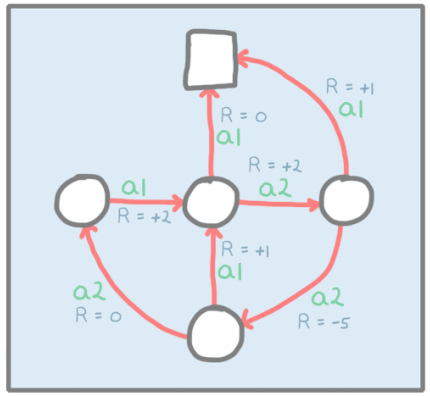
Predict some output $\mathbf{y}$ from input $\mathbf{x}$ (regression, classification).

unsupervised learning



Learn data distribution $p(\mathbf{x})$ or structure (clustering, reduction).

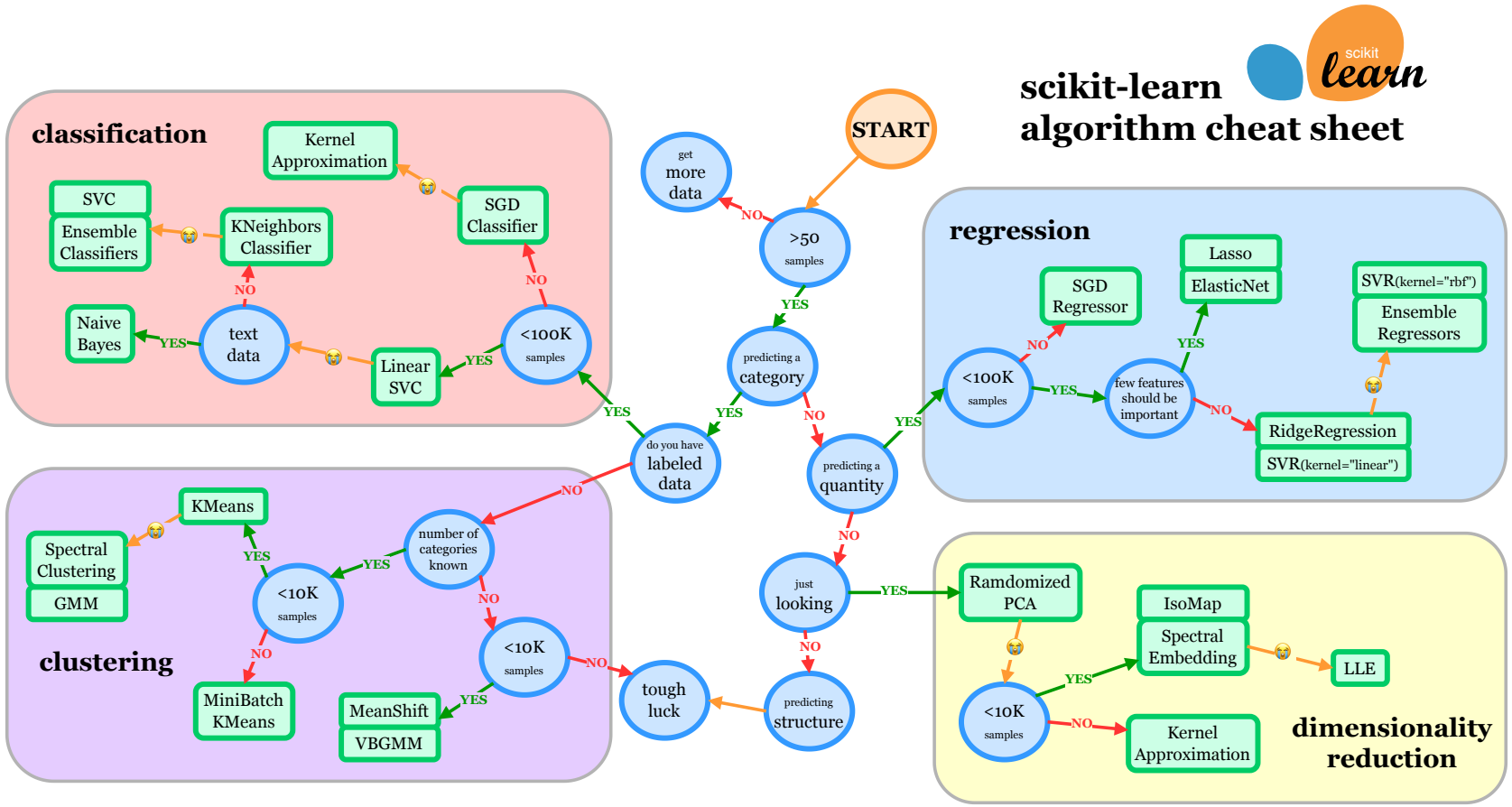
reinforcement learning



Learns a policy to maximize the reward (game playing, robotics).

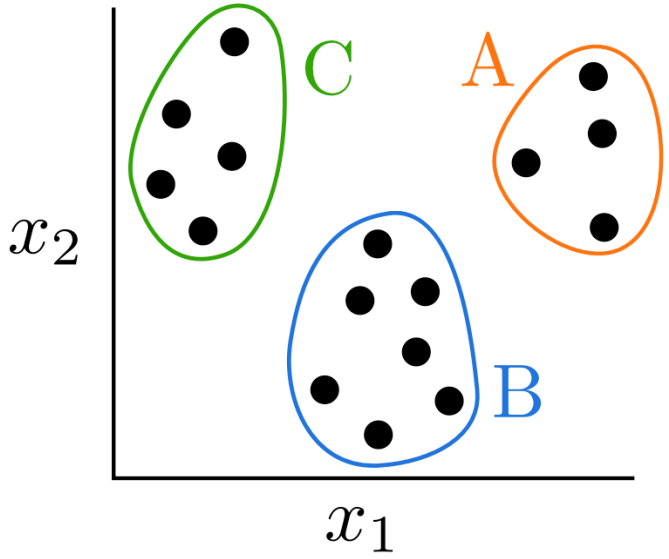
from mathworks.com

Contents of this class make use of the scikit-learn library



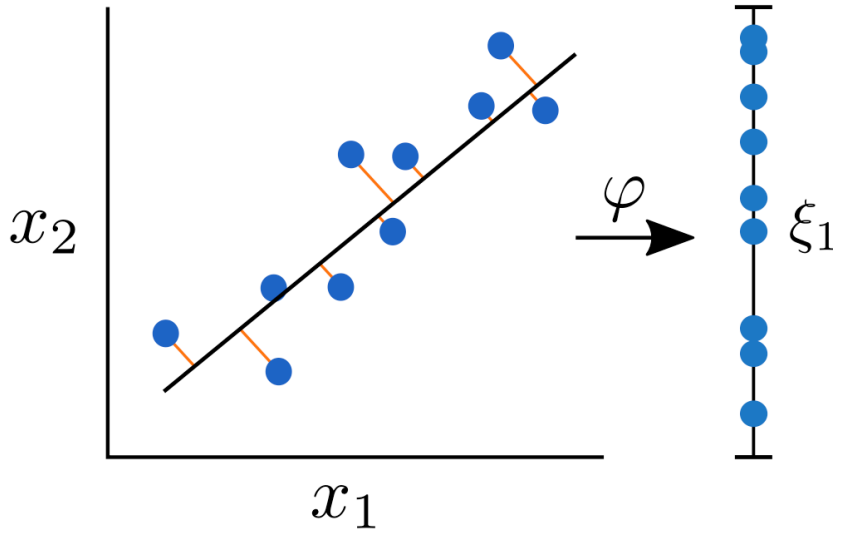
Unsupervised learning: learning the structure of the data without labels

Clustering



Group similar data points together based on some similarity measure.

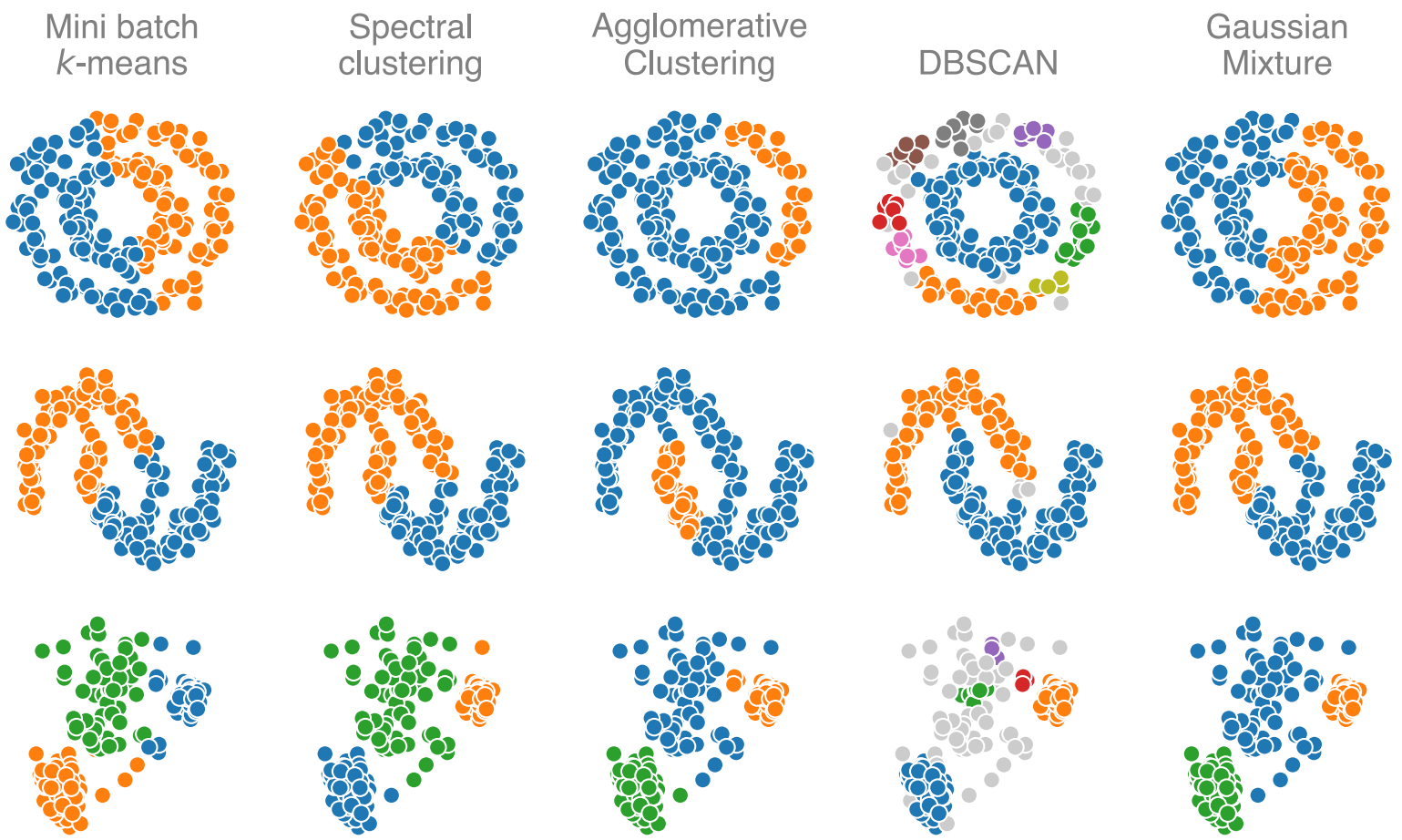
Dimensionality reduction



Find a low-dimensional representation of the data.

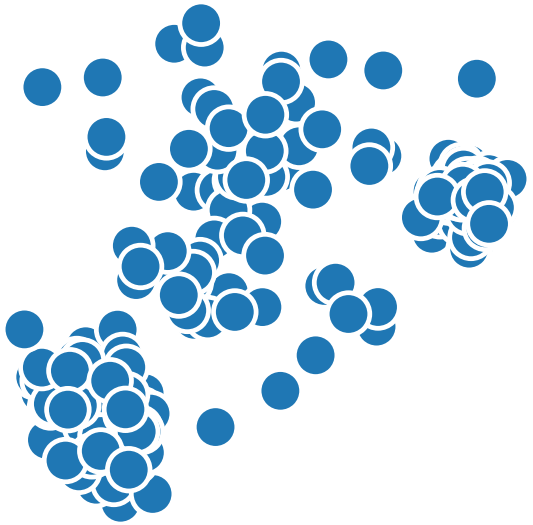
from researchgate.net

Clustering – class-membership identification without labels



from scikit-learn.org

Definitions of clustering

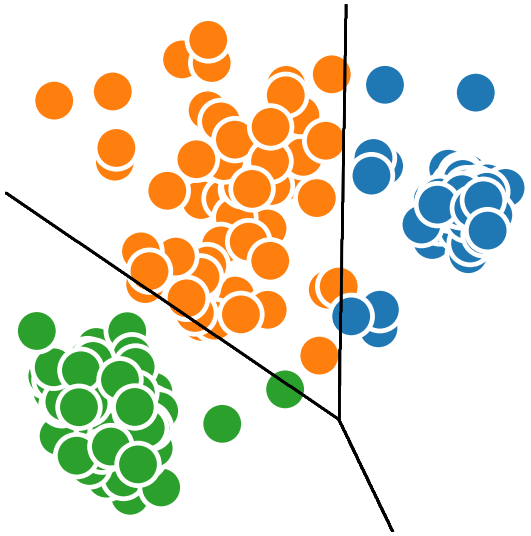


Two main definitions of clustering:

- Top-bottom: partition the heterogeneous data into **homogeneous** subsets.
- Bottom-up: group the data samples based on some criterion of **similarity**.

→ We need to provide a definition of **similarity** or **homogeneity**.

Example of clustering with *k*-means



k-means is a clustering algorithm that partitions the data into k clusters by minimizing the **inertia**:

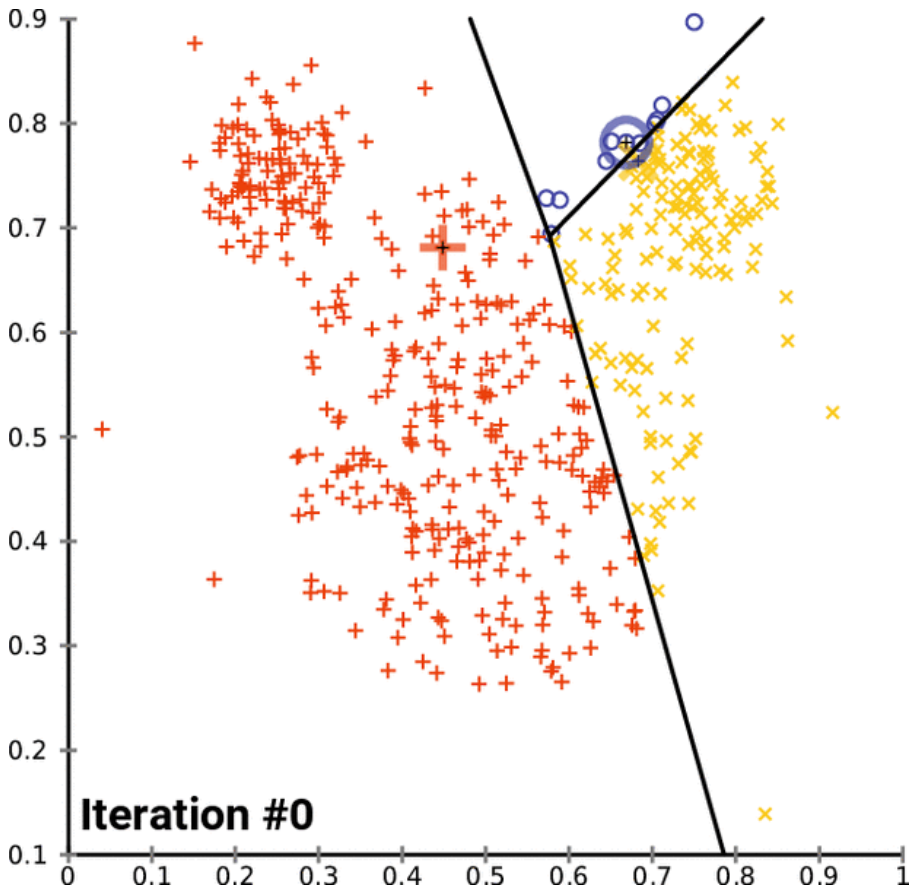
$$\arg\min_{\mathbf{C}} \sum_{i=1}^{k} \sum_{\mathbf{x}_j \in C_i} \|\mathbf{x}_j - \mu_i\|^2$$

where μ_i is the centroid of cluster C_i.

Example of clustering with *k*-means

In practice, we need to provide the number of clusters k, and the algorithm will find the best partition:

1. Initialize the centroids μ_i randomly.
2. Assign each sample to the closest centroid.
3. Update the centroids with the new samples.
4. Repeat until convergence.

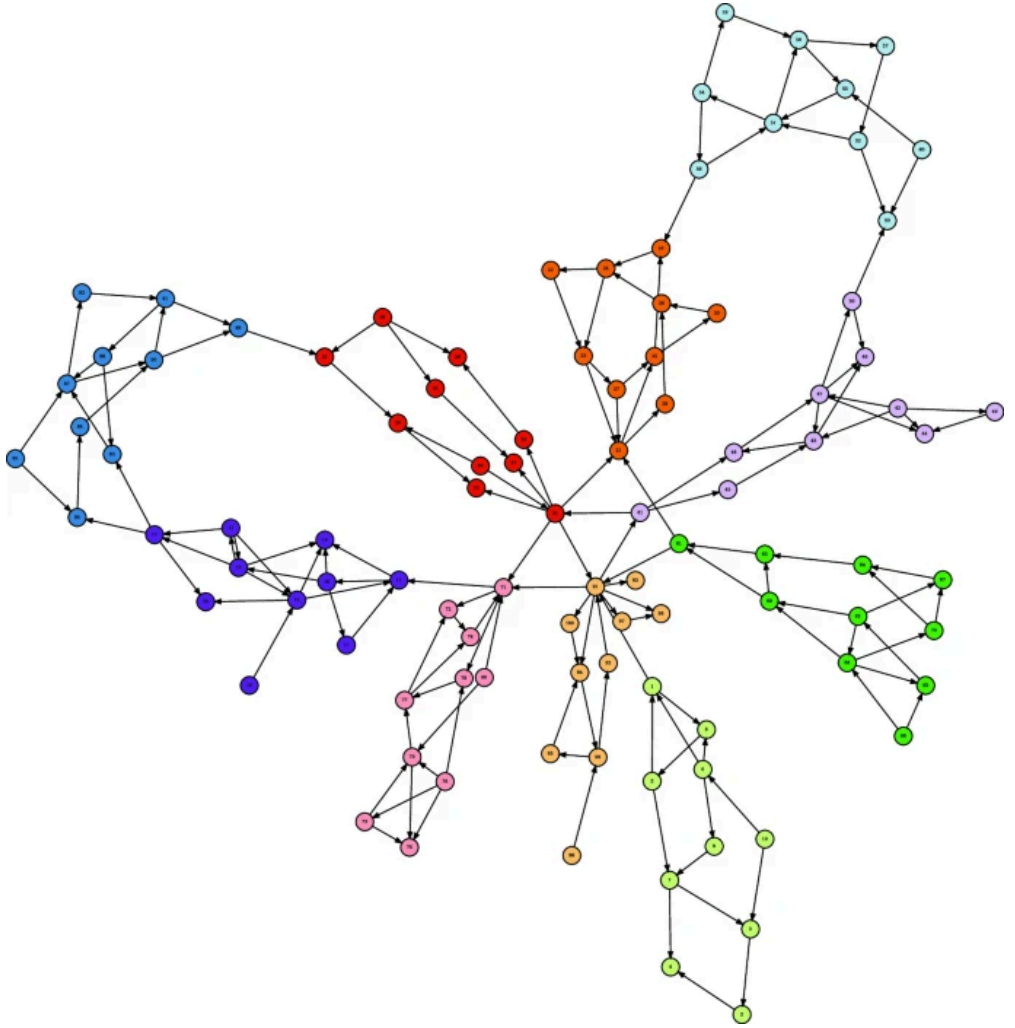


Wikipedia

Example of clustering with spectral clustering

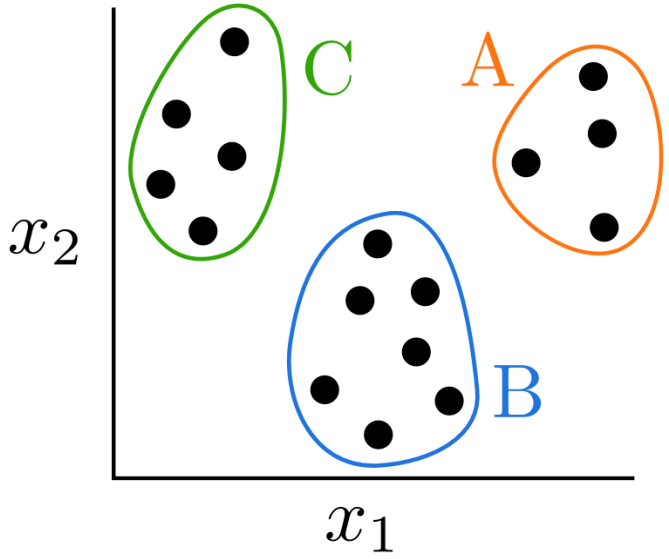
Spectral clustering uses the eigenvectors of the similarity matrix to find the clusters.

1. Compute the adjacency matrix.
2. Compute the Laplacian matrix.
3. Get the first m eigenvectors.
4. Cluster the data using k-means.



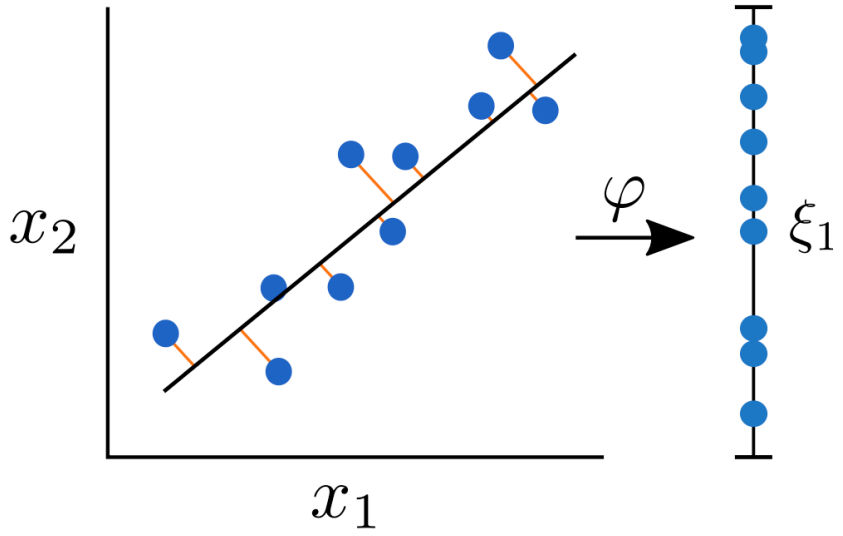
Unsupervised learning: learning the structure of the data without labels

Clustering



Group similar data points together based on some similarity measure.

Dimensionality reduction



Find a low-dimensional representation of the data.

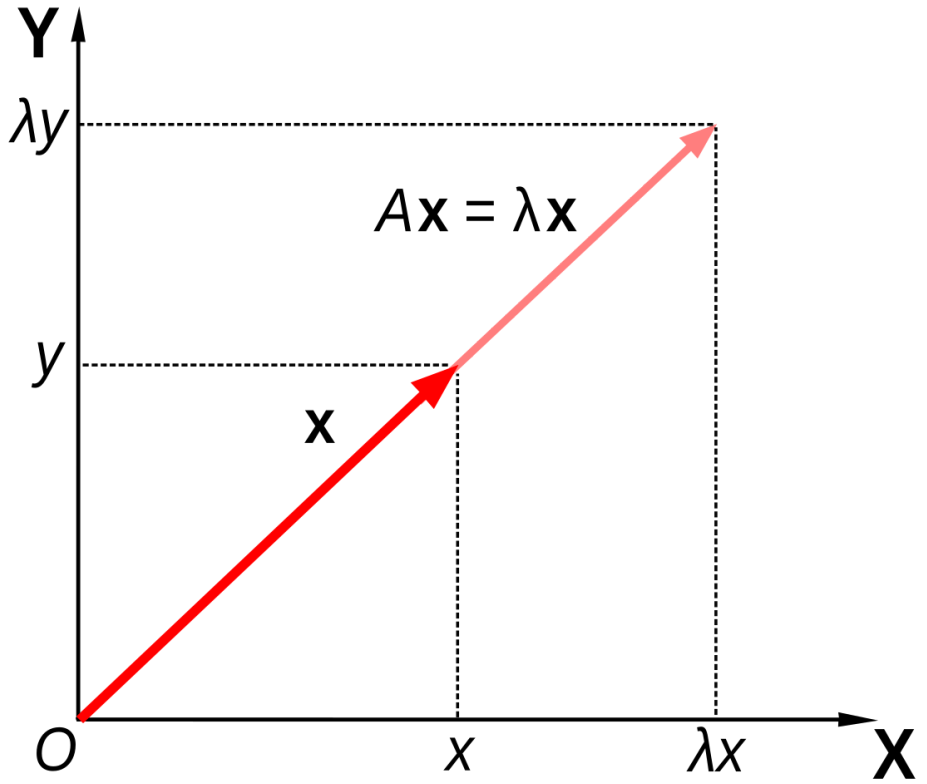
from researchgate.net

Eigenvectors of a matrix

The eigenvectors of a matrix are obtained by solving the eigenvalue problem:

$$\mathbf{Ax} = \lambda\mathbf{x}$$

where $\mathbf{A}$ is a square matrix, $\mathbf{x}$ is the eigenvector, and λ is the eigenvalue.

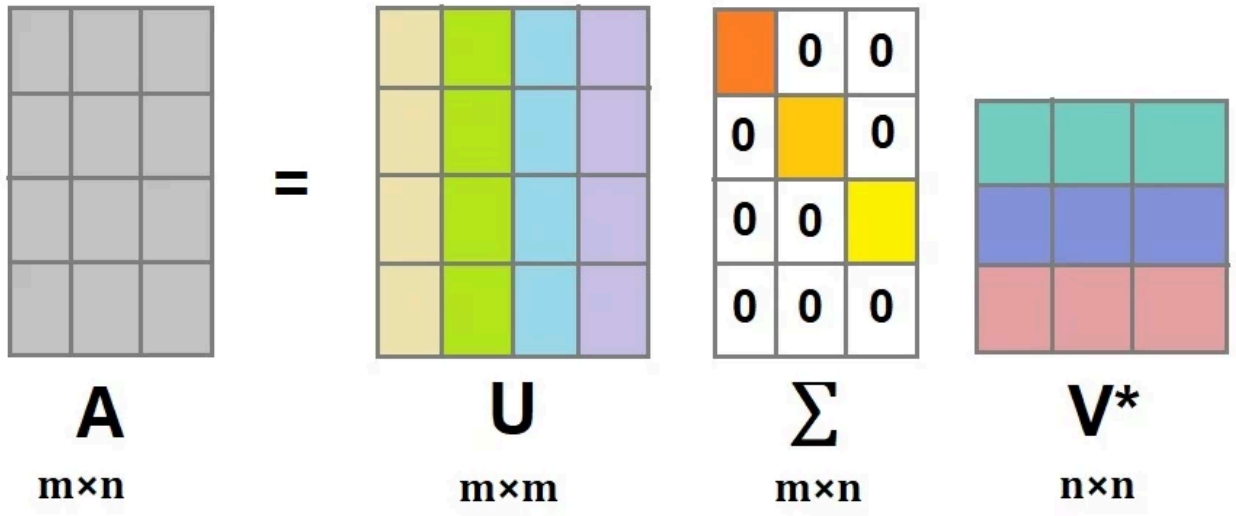


Singular value decomposition

The singular value decomposition (SVD) of a matrix $\mathbf{A}$ is defined as:

$$\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T$$

where $\mathbf{U}$ and $\mathbf{V}$ are orthogonal matrices, and $\mathbf{\Sigma}$ is a diagonal matrix.



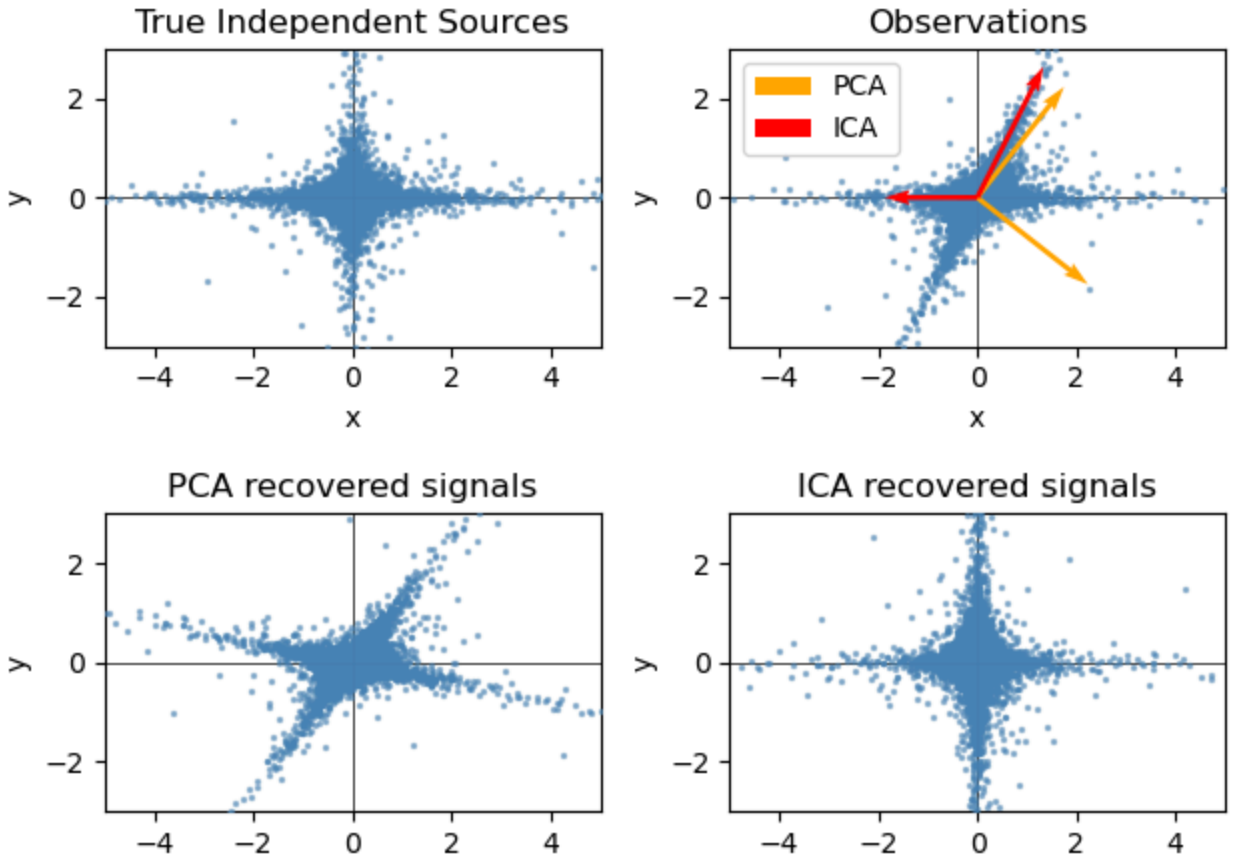
Example of dimensionality reduction with principal component analysis

Principal component analysis (PCA) is a dimensionality reduction technique that uses eigendecomposition:

1. Compute the data covariance matrix.
2. Compute the eigenvectors.
3. Project the data onto the eigenvectors.

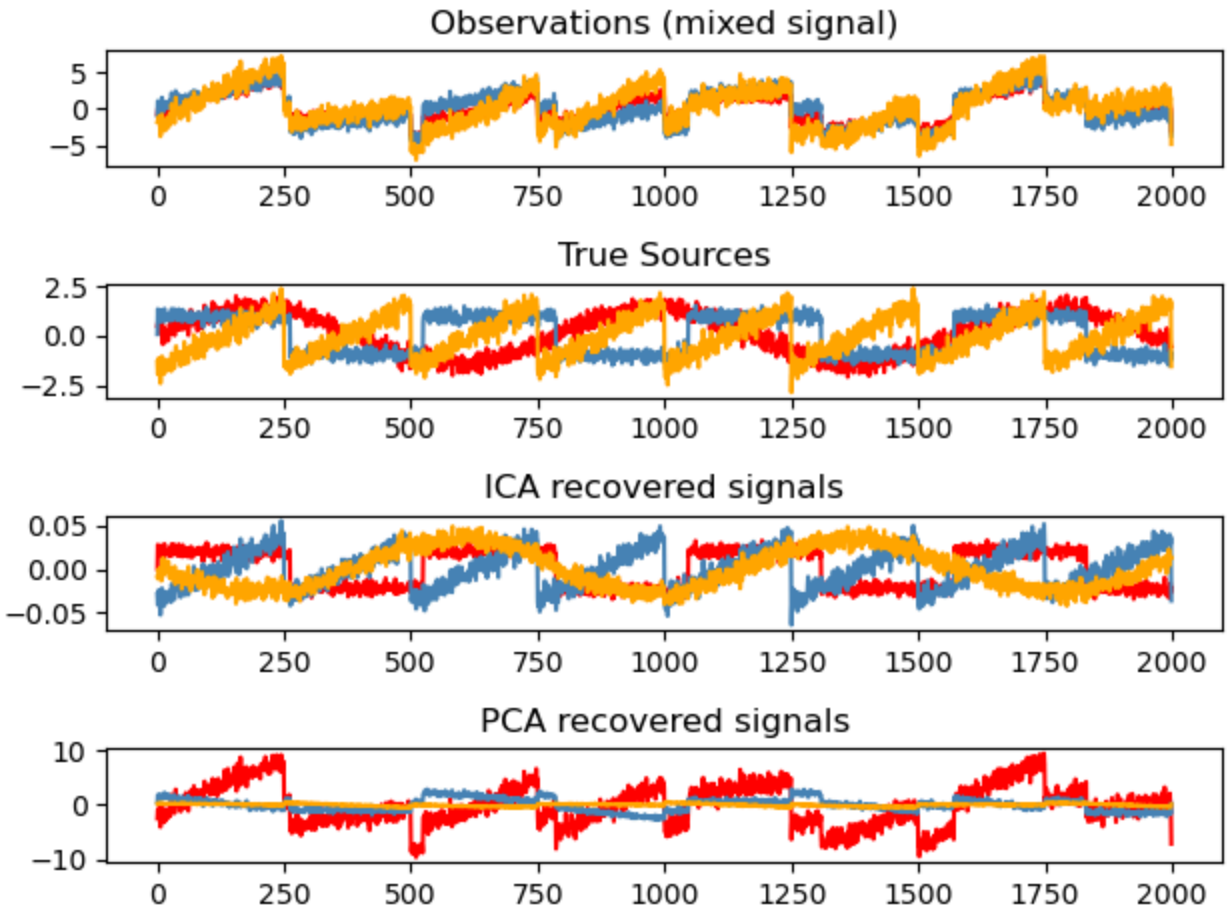
Example of dimensionality reduction with independent component analysis

Independent component analysis (ICA) is a dimensionality reduction technique that maximizes the independence of the components.



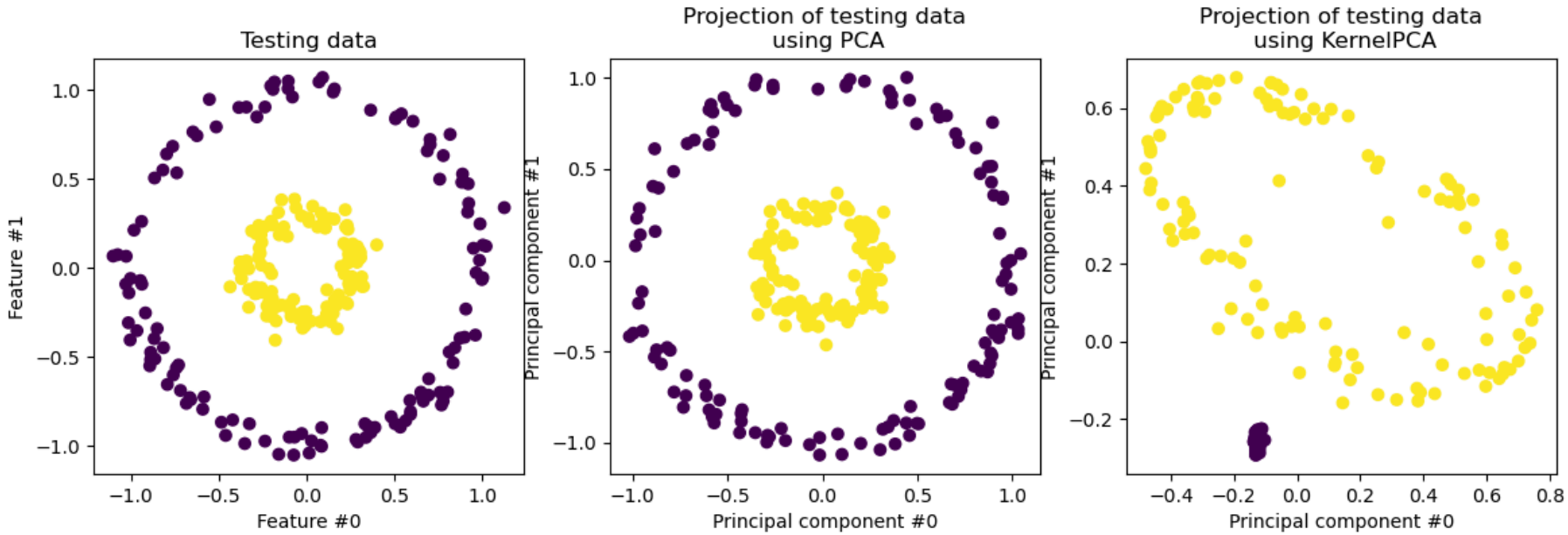
Principal vs. independent component analysis

For blind source separation, ICA is preferred over PCA.



Kernel PCA, of course

Kernel PCA is a non-linear dimensionality reduction technique that uses the kernel trick to project the data onto a higher-dimensional space.



Deep unsupervised learning

Deep neural networks can be used for unsupervised learning:

Autoencoders: learn a low-dimensional representation of the data.

Generative adversarial networks: learn a generative model of the data.

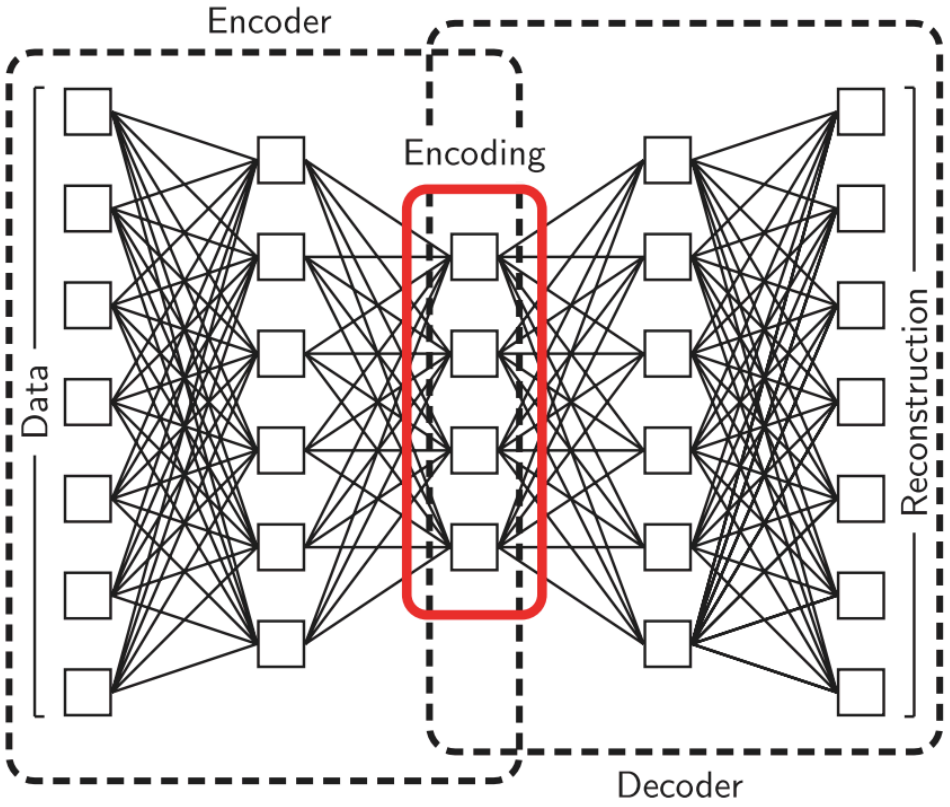
Autoencoders

Autoencoders are neural networks that learn a low-dimensional representation of the data. They are composed of an **encoder** and a **decoder**.

The input $\mathbf{x}$ is encoded into a latent representation $\mathbf{z}$, and decoded into $\mathbf{x}'$.

The loss function is the difference between the input and the reconstruction:

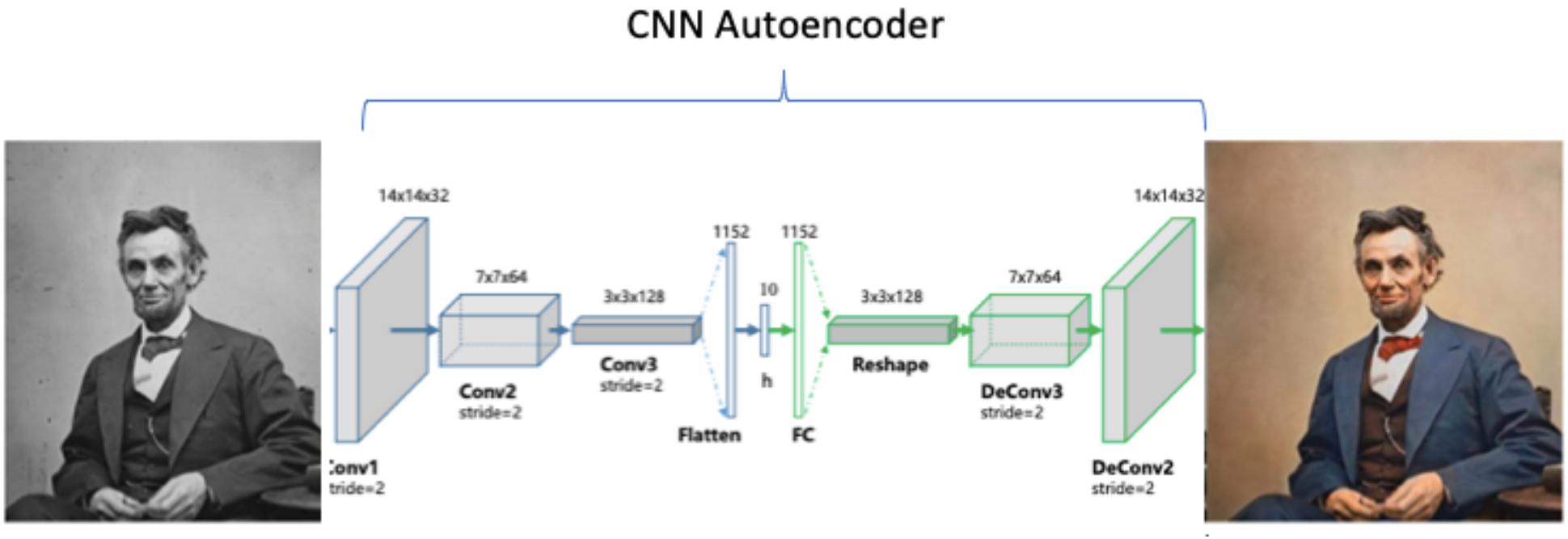
$$\mathcal{L} = \|\mathbf{x} - \mathbf{x}'\|^2$$

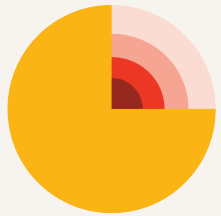


Convolutional autoencoders

Convolutional autoencoders use convolutional layers instead of fully connected layers.

They are used for image denoising, compression, and quality assessment.





ChEESE

The end!