



# Machine Learning

*Creative Machine Learning - Course 01*

---

Pr. Philippe Esling  
[esling@ircam.fr](mailto:esling@ircam.fr)

---



# Machine learning

---

## Problem statement

We witness a phenomenon  
Through a set of observations

$$(x_1, y_1) \ (x_2, y_2) \ \dots \ (x_n, y_n)$$

We know there exists a relation  
between  $\mathcal{X}$  and  $\mathcal{Y}$

But we do not know its nature

$$f : \mathcal{X} \rightarrow \mathcal{Y}$$

Machine learning constructs  
approximations

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

such that

$$y_i \approx \bar{y}_i = f_\theta(x_i)$$



# Regression and classification

---

The two most classical machine learning problems

Navigation icons: back, forward, search, etc.

# Machine learning

---

Formal problem statement

We aim to model the relationship between  $\mathcal{X}$  and  $\mathcal{Y}$

$$f : \mathcal{X} \rightarrow \mathcal{Y}$$

We collect a **dataset** that is representative of that relation

$$\mathcal{D} = \{(\mathbf{x}_1, \mathbf{y}_1) \cdots (\mathbf{x}_n, \mathbf{y}_n)\}$$

We will construct a **parametric approximation**  $f_\theta \in \mathcal{F}_\Theta$

$$f_\theta : \mathcal{X} \rightarrow \mathcal{Y} \quad \bar{\mathbf{y}} = f_\theta(\mathbf{x})$$

We want to find the best  $f_\theta^*$  so that  $\bar{\mathbf{y}} \approx \mathbf{y}$

So we need to find the best parameters  $\theta^* \in \Theta$

Need to compute the errors (**loss**) of our model

$$\mathcal{L}(\bar{\mathbf{y}}, \mathbf{y} \mid f_\theta, \theta)$$

And minimize (**optimize**) this amount of errors

$$\theta^* = \operatorname{argmin}_\theta \mathcal{L}(\bar{\mathbf{y}}, \mathbf{y} \mid f_\theta, \theta)$$

# Key elements of our problem definition

---

**Dataset** |  $\mathcal{D} = \{(\mathbf{x}_1, \mathbf{y}_1) \cdots (\mathbf{x}_n, \mathbf{y}_n)\}$   
The **dataset** has to be **representative** of the relation  $f : \mathcal{X} \rightarrow \mathcal{Y}$

**Model** |  $f_\theta : \mathcal{X} \rightarrow \mathcal{Y} \quad \bar{\mathbf{y}} = f_\theta(\mathbf{x})$   
The choice of **parametric family**  $f_\theta \in \mathcal{F}_\Theta$  is critical

**Loss** |  $\mathcal{L}(\bar{\mathbf{y}}, \mathbf{y} \mid f_\theta, \theta)$   
How do we measure the errors (**loss**) of our model

**Optimization** | Search for best  $\theta^* \in \Theta$  so  $f_{\theta^*}$  minimizes the loss ( $\bar{\mathbf{y}} \approx \mathbf{y}$ )  
$$\theta^* = \operatorname{argmin}_{\theta} \mathcal{L}(\bar{\mathbf{y}}, \mathbf{y} \mid f_\theta, \theta)$$

How to choose the parametric family ?

# Choice of approximation family (model)

Quality depends heavily on family  $\mathcal{F}_\theta$

Notion of *model capacity*



## Approximation families

Constant

$$\mathcal{F}_0(x) = \theta_0$$

Linear

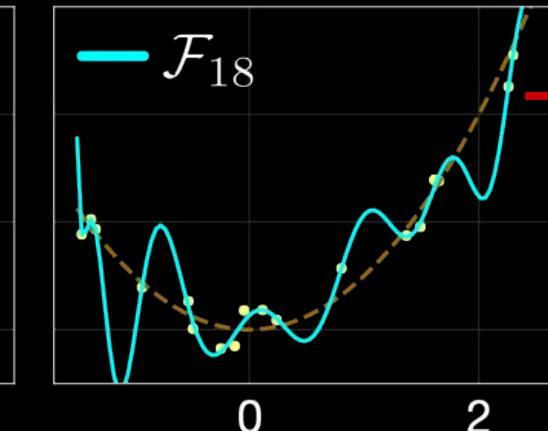
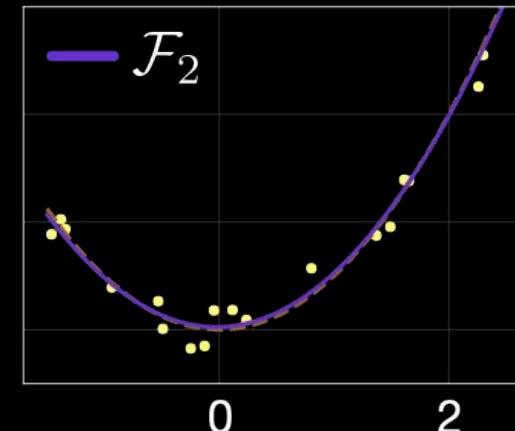
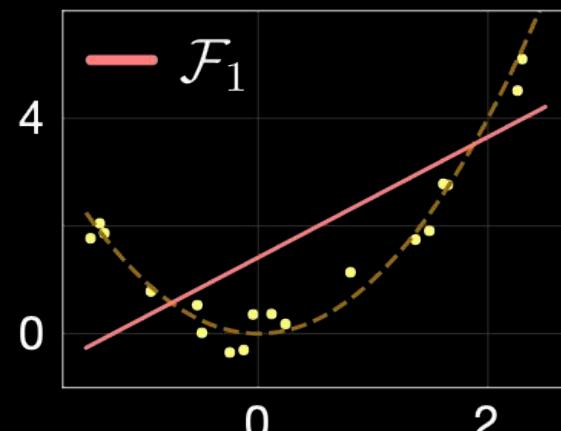
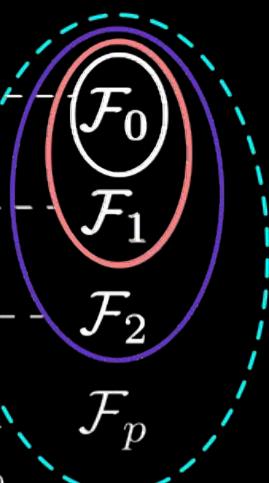
$$\mathcal{F}_1(x) = \theta_1 x + \theta_0$$

Square

$$\mathcal{F}_2(x) = \theta_2 x^2 + \theta_1 x + \theta_0$$

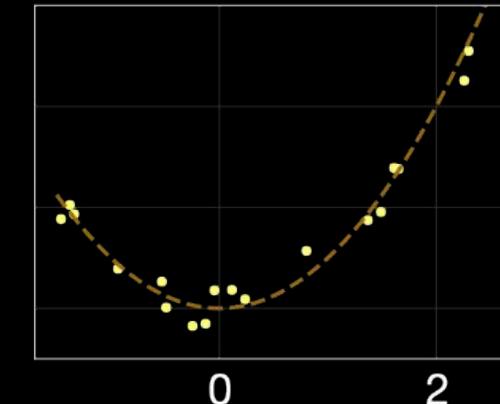
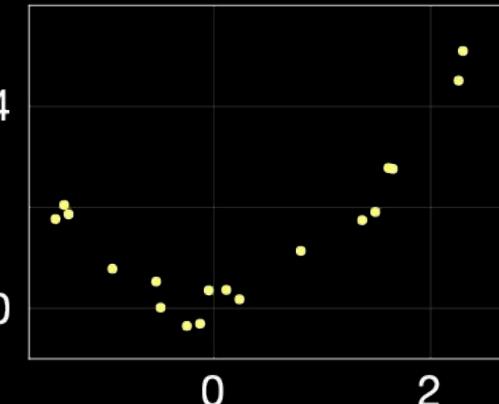
Polynomial

$$\mathcal{F}_p(x) = \theta_p x^p + \dots + \theta_1 x + \theta_0$$



--- Ground truth

● Training points



## Overfitting

("Because it can")

We will see how we can solve this through regularization

# Linear regression

---

## Supervised learning algorithm

Model the relationship between **one** output (target) and one or more inputs (features).

$$\mathbf{x} \in \mathbb{R}^n \quad \mathcal{X} \xrightarrow{f_{\theta}} \mathcal{Y} \quad y \in \mathbb{R}$$

*Input (feature)*      *Output (target)*

## Regression

# Linear regression

## Supervised learning algorithm

Model the relationship between **one** output (target) and one or more inputs (features).

$$\mathbf{x} \in \mathbb{R}^n \quad \mathcal{X} \xrightarrow{f_{\theta}} \mathcal{Y} \quad y \in \mathbb{R}$$

*Input (feature)*                                   *Output (target)*

**Goal** Find best-fitting *hyperplane (line)* to predict output.

**Equation**  $y = w_0 + w_1x_1 + w_2x_2 + \dots + w_nx_n + \epsilon$

**A note on the residual error**  $\epsilon$  Modeling the observation noise always present

**Model**  $f_{\theta}(x) = \bar{y} = w_0 + w_1x_1 + \dots + w_nx_n$

Modify parameters  $\theta = \{w_0, \dots, w_n\}$  so that  $y \approx \bar{y}$

**Best-fitting** line ? *Minimize differences between real and predicted output*

**Various application in inference and forecasting.**

$y$  : Output (target)  
 $x_i$  : Input (feature)  
 $w_i$  : Coefficients (weights)  
 $w_0$  : Intercept (bias term)  
 $\epsilon$  : Residual error

# Simple linear regression

## Principle

Linear regression with a **single input** variable.

$$x \in \mathbb{R} \rightarrow y \in \mathbb{R}$$

Model relationship of *single* input and output (*target*).

**Examples:** Predicting house prices based on sizes, predicting temperature based on year

Model

$$\bar{y} = w_0 + w_1 x \quad \theta = \{w_0, w_1\}$$

Based on our collected dataset of examples  $\mathcal{D} = \{(x_1, y_1), \dots, (x_n, y_n)\}$

We can compute the errors made by our model **on each example**

$$\epsilon_i = |y_i - \bar{y}_i| = |y_i - (w_0 + w_1 x_i)| \text{ —— } L_1 \text{ loss}$$

**$L_2$  loss function**  
Mean-Squared Error (MSE)

$$\mathcal{L}_{MSE}(\bar{\mathbf{y}}, \theta) = \sum_{i=1}^n |y_i - \bar{y}_i|^2 = \sum_{i=1}^n |y_i - (w_0 + w_1 x_i)|^2 \text{ —— } \begin{array}{l} \text{More sensitive to} \\ \text{outliers (square)} \end{array}$$

**whole dataset**

**Goal**  $\underset{\theta}{\operatorname{argmin}} \mathcal{L}(\bar{\mathbf{y}}, \theta)$  Minimize the MSE based on parameters  $\theta = \{w_0, w_1\}$

# Gradient descent

---

**Intuition** How to find the parameters that minimize our loss function ?

$$\min_{\theta} \mathcal{L}(\bar{\mathbf{y}}, \theta) \longrightarrow \frac{\partial \mathcal{L}}{\partial \theta} = 0 \quad \text{A minimum exists where the gradient of our function is null}$$

## Problems

1. Explicit solution ? Might **require prohibitive computation**
2. Finding minima ? There might be **multiple local minima**
3. **Gradient can change** widely depending on the examples

**Gradient descent** Optimization algorithm to minimize the loss iteratively.  
Compute partial derivatives of the loss w.r.t each parameter.

$$\nabla \mathcal{L}(\mathbf{w}) = \left[ \frac{\partial \mathcal{L}}{\partial w_1}, \frac{\partial \mathcal{L}}{\partial w_2}, \dots, \frac{\partial \mathcal{L}}{\partial w_n} \right]$$

Represents the **direction of the steepest increase** of the loss function.  
Adjusting parameters in that direction is the steepest decrease of the loss

# Gradient descent

---

## Algorithm

### 1 - Initialize parameters

Start from random point  $\mathbf{w} \sim \mathcal{N}(\mu_0, .01)$

### 2 - Compute partial derivatives

Derive the loss based on all parameters

$$\nabla \mathcal{L}(\mathbf{w}) = \left[ \frac{\partial \mathcal{L}}{\partial w_1}, \frac{\partial \mathcal{L}}{\partial w_2}, \dots, \frac{\partial \mathcal{L}}{\partial w_n} \right]$$

### 3 - Iterative updates

In each iteration update the weights

$$\mathbf{w}^{t+1} = \mathbf{w}^t - \eta \nabla \mathcal{L}(\mathbf{w}^t)$$

Move in the opposite direction of the gradient

Learning rate  $\eta$  controls the step size of the updates.

### 4 - Convergence

- $\mathcal{L}(\mathbf{w}) \rightarrow 0$

Stopping criteria

- $\nabla \mathcal{L}(\mathbf{w}) \rightarrow 0$
- $t > n_{epochs}$

# Understanding the learning rate

## Gradient descent optimization

$$\nabla \mathcal{L}(\mathbf{w}) = \left[ \frac{\partial \mathcal{L}}{\partial w_1}, \frac{\partial \mathcal{L}}{\partial w_2}, \dots, \frac{\partial \mathcal{L}}{\partial w_n} \right]$$

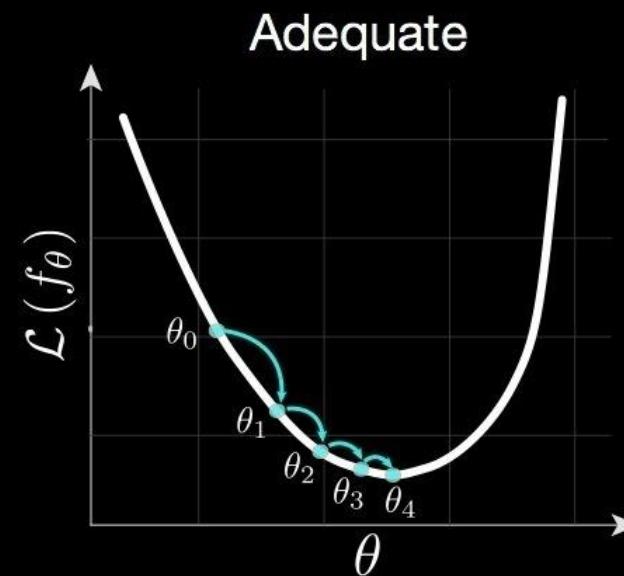
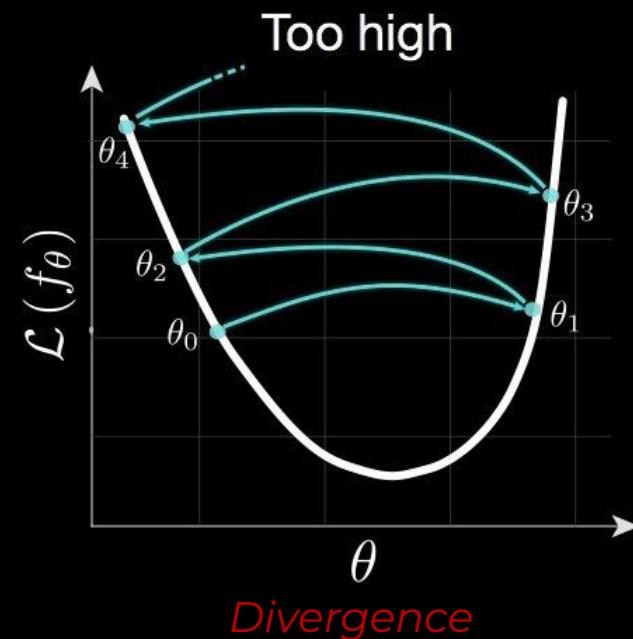
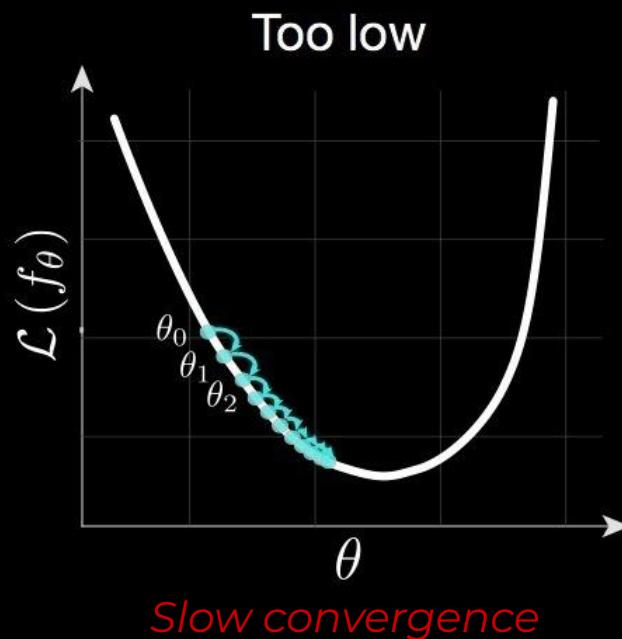
In our previous update equation

$$\mathbf{w}^{t+1} = \mathbf{w}^t - \boxed{\eta} \nabla \mathcal{L}(\mathbf{w}^t)$$

Learning rate  $\eta$

## Impact of the learning rate

Problem is that we do not know the **magnitude** of the gradient



# Applying gradient descent for regression

**Goal** We want to minimize our loss  $\mathcal{L}_{MSE}(\bar{\mathbf{y}}, \theta) = \sum_{i=1}^n (y_i - \bar{y}_i)^2 = \sum_{i=1}^n (y_i - (w_1 x_i + w_0))^2$

**Compute**  $\nabla \mathcal{L}(\mathbf{w}) = \left[ \frac{\partial \mathcal{L}}{\partial w_0}, \frac{\partial \mathcal{L}}{\partial w_1} \right]$

$$\frac{\partial \mathcal{L}}{\partial w_0} = \frac{\partial}{\partial w_0} \left( \sum_{i=1}^N (y_i - (w_1 x_i + w_0))^2 \right)$$
$$\frac{\partial \mathcal{L}}{\partial w_1} = \frac{\partial}{\partial w_1} \left( \sum_{i=1}^N (y_i - (w_1 x_i + w_0))^2 \right)$$

$$\frac{\partial \mathcal{L}}{\partial w_0} = -2 \sum_{i=1}^N (y_i - (w_1 x_i + w_0))$$
$$\frac{\partial \mathcal{L}}{\partial w_1} = -2 \sum_{i=1}^N x_i (y_i - (w_1 x_i + w_0))$$

**Complete algorithm** **Initialize parameters**  $w_1^0, w_0^0 \sim \mathcal{N}(0, 1)$

1. Compute our current **approximation**  $\bar{y} = w_1^t x + w_0^t$
2. Compute the current value of the **loss function**  $\mathcal{L}_{MSE}(\bar{\mathbf{y}}, \theta) = \sum_{i=1}^n (y_i - (w_1^t x_i + w_0^t))^2$
3. Compute **gradients** of the loss function

**Loop**

$$\nabla \mathcal{L}(\mathbf{w}) = \left[ \frac{\partial \mathcal{L}}{\partial w_0^t}, \frac{\partial \mathcal{L}}{\partial w_1^t} \right] = \left[ -2 \sum_{i=1}^N (y_i - (w_1^t x_i + w_0^t)), -2 \sum_{i=1}^N x_i (y_i - (w_1^t x_i + w_0^t)) \right]$$

4. **Update our parameters** based on the gradients

$$\mathbf{w}^{t+1} = \mathbf{w}^t - \eta \nabla \mathcal{L}(\mathbf{w}^t)$$

# Gradient descent variants

---

So far discussed **batch gradient descent** - compute gradient for the entire dataset

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \eta \frac{1}{N} \sum_i \nabla \mathcal{L}_i(\mathbf{w}_t)$$

**Can be slow (or even impossible)  
for large datasets.**

Variants can improve computational efficiency and even convergence properties.

## Stochastic GD

Computes gradient with **single data point** each time

Faster than batch-GD, but high variance in updates

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \eta \underline{\nabla \mathcal{L}_i(\mathbf{w}_t)}$$

---

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \eta \frac{1}{b} \sum_{i=1}^b \nabla \mathcal{L}_i(\mathbf{w}_t)$$

---

## Mini-batch GD

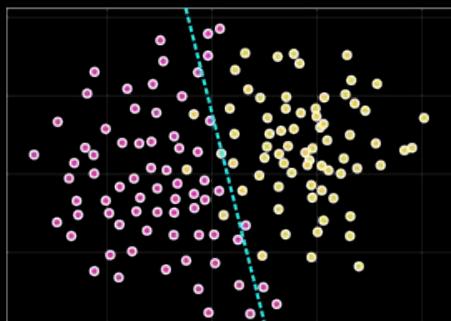
Use a **random subset (mini-batch)** of the dataset.  
Trade-off for balancing speed and variance

Adaptive Moment Estimation (ADAM)  
(cf. deep learning)

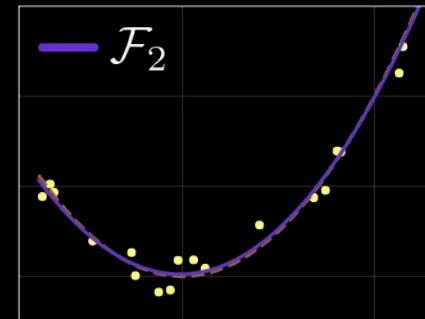
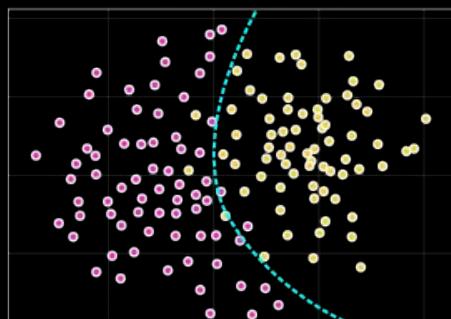
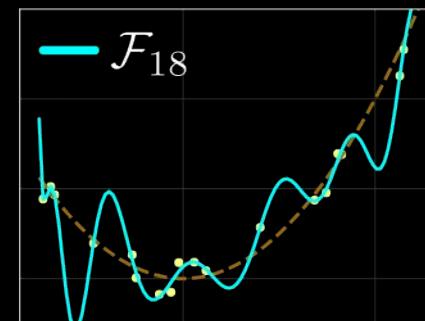
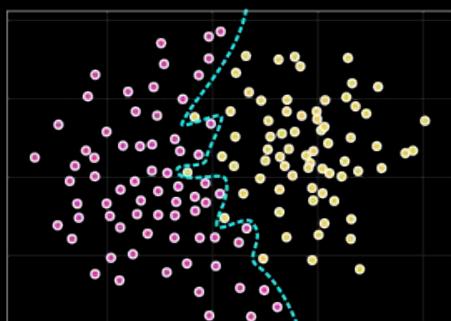
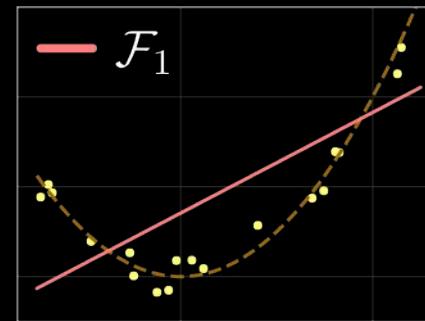
ADAM adapts the learning rate for each parameter  
Maintains decaying average of past and squared gradients.  
More robust and efficient than standard Gradient Descent.

# Understanding capacity

## Classification



## Regression



## Underfitting

- Function family is **too poor**
- Capacity is **too low** for this problem

## Overfitting

- Function family is **too rich**
- Capacity is **too high** for this problem

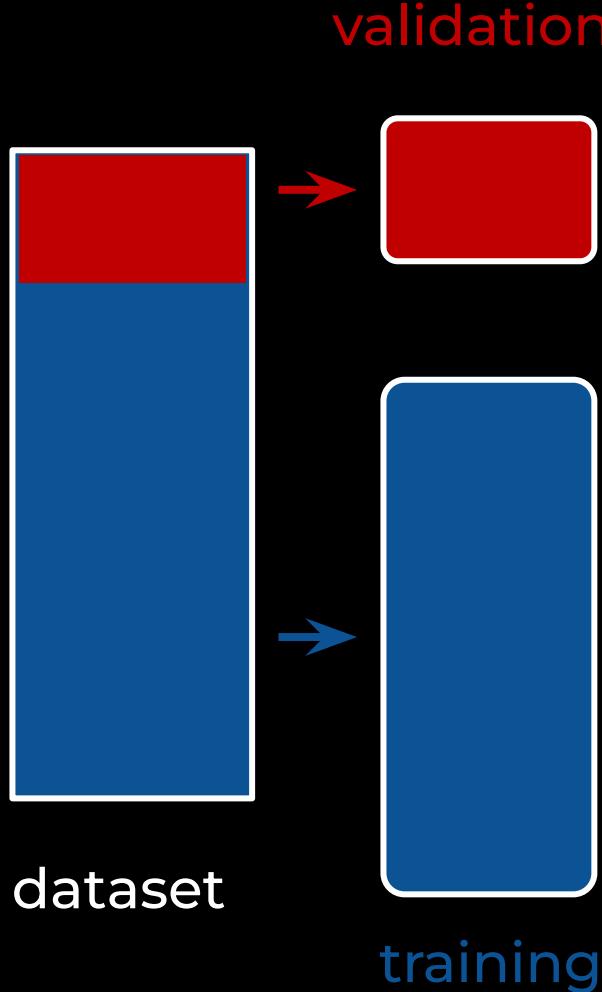
## Optimal capacity

- Function family is **adequate**
- **Optimal capacity** for this problem

# Cross-validation for overfitting

---

How do we aim to avoid overfitting ?



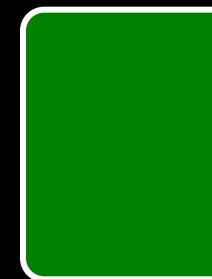
Split your complete dataset into two distinct set

- **Training** set (~80%) for learning your model
- **Validation** set (~20%) for evaluating overfitting

The validation set emulates an unknown dataset  
Allows to evaluate the quality on unseen data

**Final performance evaluation**

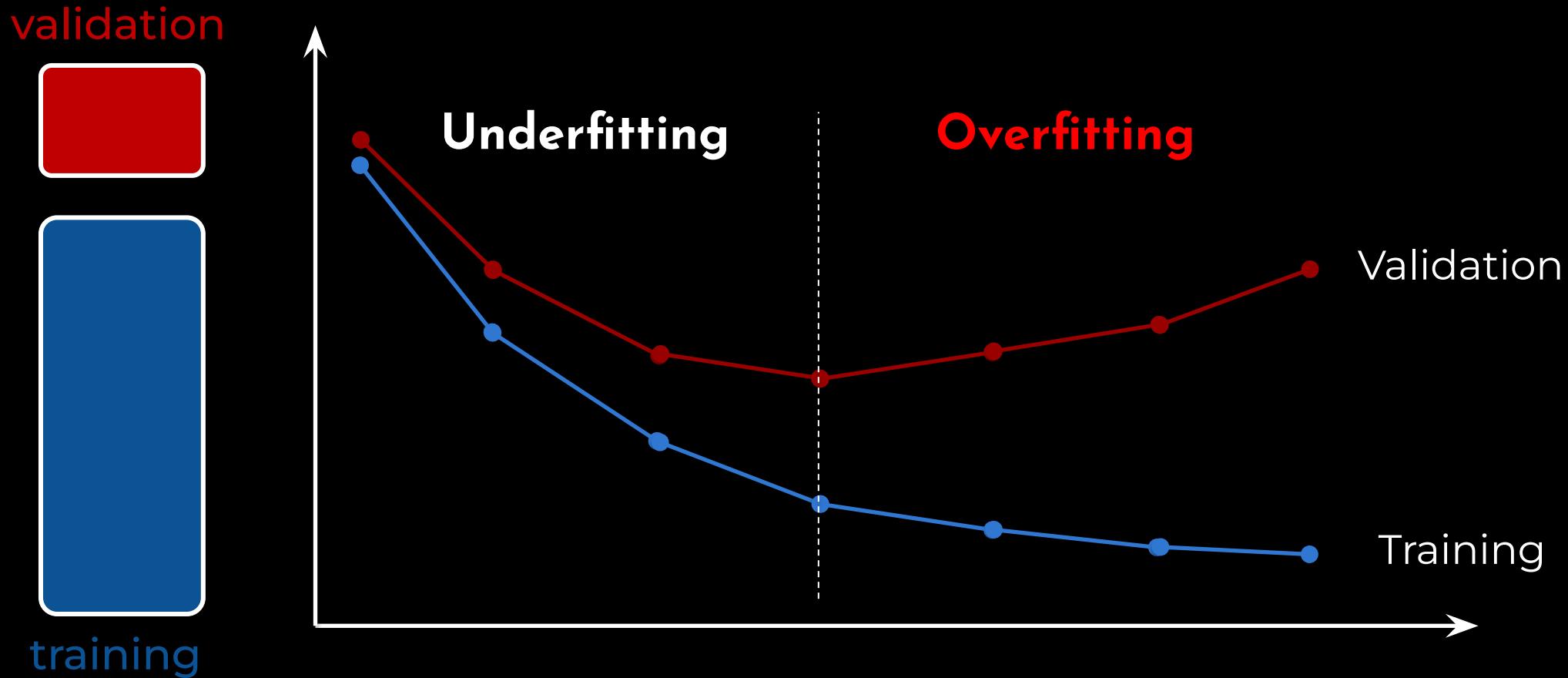
Performed on a *separate independent dataset*  
Ideally this set is *out-of-distribution (o.o.d)* to prevent bias



testing

# Early stopping

Stop the learning when the error increases on the validation set  
*(Approximates the generalization power)*



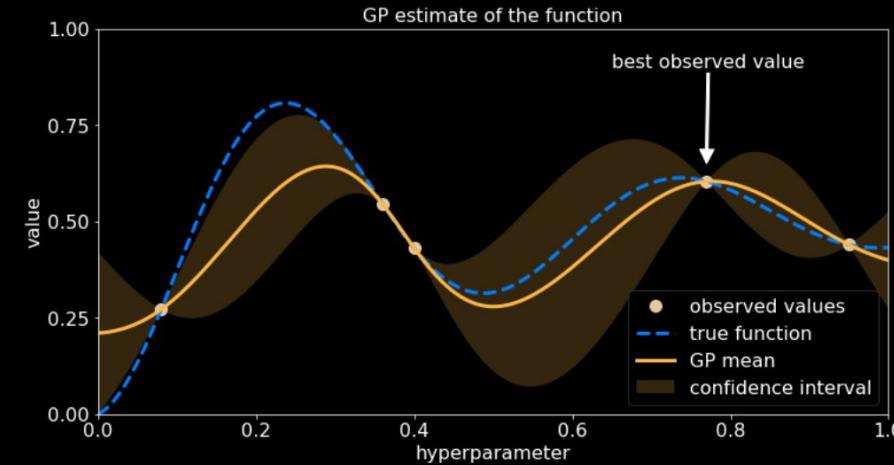
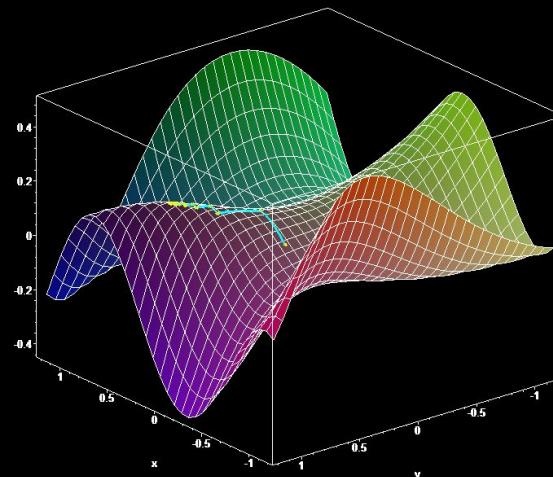
# A quick note on optimization

---

Not all ML approaches use gradient descent

A wide part of the **optimization** field is centered on search

- Depth-first (lexical order) - Breadth-first
- Heuristic – genetic algorithms



- Unfortunately, we will not spend too much time on *optimization* aspects
  - This would require a course in itself!
- But still explain everything that you need to perform everything yourselves

# Multiple Linear Regression

**Principle**

Extension of simple linear regression for **two or more variables** as input

**Examples:** Predicting temperature based on year, CO<sub>2</sub> gas, human population, location

**Equation**

$$f_{\theta}(\mathbf{x}) = \bar{y} = w_0 + w_1x_1 + \cdots + w_mx_m \quad \theta = \{w_0, \dots, w_m\}$$

Find the best-fitting **hyperplane** still with MSE loss

$$\mathcal{L}_{MSE}(\bar{\mathbf{y}}, \theta) = \sum_{i=1}^N \left| y_i - (w_0 + \sum_{j=1}^m w_j x_{i,j}) \right|^2$$

**Matrix formulation**

We should compute  $\nabla \mathcal{L}(\mathbf{w}) = \left[ \frac{\partial \mathcal{L}}{\partial w_1}, \frac{\partial \mathcal{L}}{\partial w_2}, \dots, \frac{\partial \mathcal{L}}{\partial w_m} \right]$

Deriving and updating each weight  
is tedious and time-consuming

This problem can be expressed in **matrix form**

$$\bar{\mathbf{y}} = \mathbf{X}\mathbf{w} + \mathbf{w}_0 \quad \mathcal{L}_{MSE}(\bar{\mathbf{y}}, \theta) = (\mathbf{y} - (\mathbf{X}\mathbf{w} + \mathbf{w}_0))$$

$$\begin{array}{ll} \mathbf{y} \in \mathbb{R}^{N \times 1} & \mathbf{w}_0 \in \mathbb{R}^{N \times 1} \\ \mathbf{X} \in \mathbb{R}^{N \times M} & \mathbf{w} \in \mathbb{R}^{M \times 1} \end{array}$$

We obtain simpler expressions of the gradients

$$\frac{\partial \mathcal{L}}{\partial \mathbf{w}} = \mathbf{X}^T(\mathbf{y} - (\mathbf{X}\mathbf{w} + \mathbf{w}_0)) \quad \frac{\partial \mathcal{L}}{\partial \mathbf{w}_0} = (\mathbf{y} - (\mathbf{X}\mathbf{w} + \mathbf{w}_0))$$

Can go further to integrate bias  $\mathbf{w} = [w_1, w_2, \dots, w_m, w_0] \quad \mathbf{x} = [x_1, x_2, \dots, x_m, 1]$

More compact representation and efficient computation.

# Linear classification

---

General problem

$$\mathbf{x} \in \mathbb{R}^n$$

*Input (feature)*

$$\mathcal{X} - f_{\theta} \rightarrow \mathcal{Y}$$

$$y \in \{0, 1\}$$

*Output class*

Find how to assign data points into distinct classes

*Linearly separable* problems means classes can be separated by a line

---

# Linear classification

General problem

$$\mathbf{x} \in \mathbb{R}^n_{\text{Input (feature)}}$$

$$\mathcal{X} - f_{\theta} \rightarrow \mathcal{Y}$$

$$y \in \{0, 1\}_{\text{Output class}}$$

Find how to assign data points into distinct classes

*Linearly separable* problems means classes can be separated by a line

Approach  
(linear)

Use a weighted sum of input features  $f_{\theta}(\mathbf{x}) = \mathbf{w}^T \mathbf{x} + b$

Represented in matrix form with bias term included

$$\mathbf{w} = [w_1, w_2, \dots, w_n, b] \quad \mathbf{x} = [x_1, x_2, \dots, x_n, 1]$$

Decision rule

Functions that separate classes using a linear decision boundary

$$f_{\theta}(\mathbf{x}) \geq 0 \text{ — Class 0}$$

$$f_{\theta}(\mathbf{x}) < 0 \text{ — Class 1}$$

Loss function

Hinge loss function for linear classification:

$$\mathcal{L}(\mathbf{w}, \mathbf{x}_i, y_i) = \max(0, 1 - y_i(\mathbf{w}^T \mathbf{x}_i))$$

[Exercise 1: Perform derivation and implement in NumPy]

# Polynomial regression

## Problem

Linear models obviously only represent **linear relationships**

We are looking to increase the complexity (**capacity**) of our model

## Polynomial regression

$$f_{\theta}(x) = \bar{y} = w_0 + w_1 x^1 + w_2 x^2 + \dots + w_m x^m$$

Allows to model more complex (non-linear) relationships

Note that still

$$x \in \mathbb{R}$$

**Loss function** Slightly modifying the MSE loss  $\mathcal{L}_{MSE}(\bar{\mathbf{y}}, \theta) = \sum_{i=1}^N \left| y_i - (w_0 + \sum_{j=1}^m w_j x_i^j) \right|^2$

**Gradient descent** Actually very similar to the previous expressions  
Remember that we are deriving with respect to  $\frac{\partial \mathcal{L}}{\partial w_i}$

**Implementation tip** Prepare “higher-order features”  $\mathbf{x}_{poly} = [\mathbf{x}, \mathbf{x}^2, \mathbf{x}^3, \dots, \mathbf{x}^m]$

**Regularization** To prevent overfitting, add regularization  $\mathcal{L}_{L_1}(\hat{\mathbf{y}}, \theta) = \mathcal{L}_{MSE}(\hat{\mathbf{y}}, \theta) + \lambda \sum_{i=1}^m |w_i|$

**[Exercise 2: Perform derivation and implement in NumPy]**

# Course exercises

---

Exercises for this lesson and how do they differ from what we have seen together ?

## General framework

1. Implement a slightly different task in NumPy
2. Produce the same implementation in JAX
3. Compare both versions

1 - Linear classification Identical model form and setup as linear regression

**Major difference: loss function (and domain)**

$$y \in \{0, 1\}$$

$$\mathcal{L}(\mathbf{w}, \mathbf{x}_i, y_i) = \max(0, 1 - y_i(\mathbf{w}^T \mathbf{x}_i))$$

2 - Polynomial regression Identical task and setup as linear regression

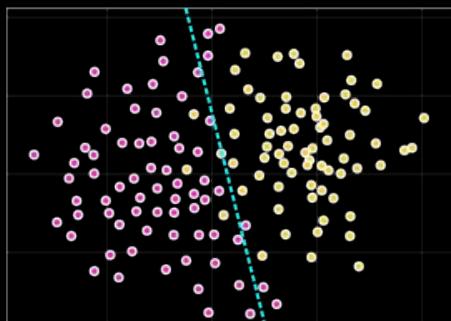
**Major difference: model used**

$$f_{\theta}(x) = \bar{y} = w_0 + w_1 x^1 + w_2 x^2 + \dots + w_m x^m$$

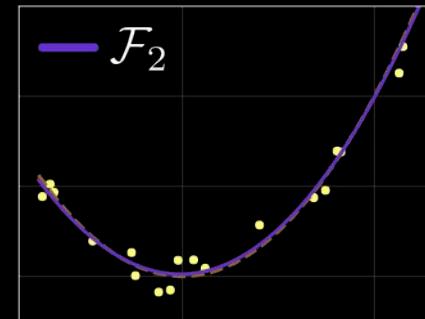
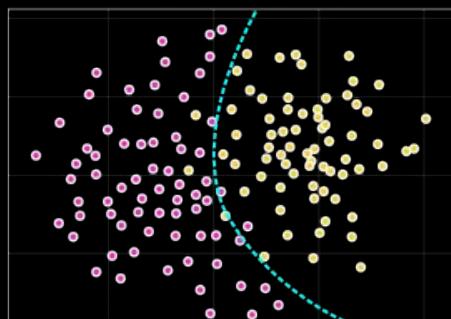
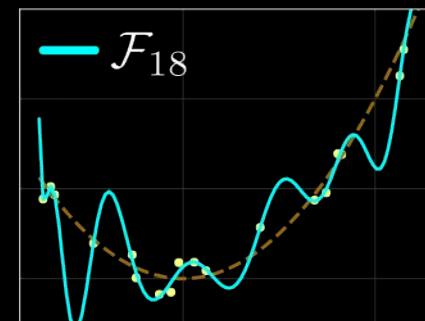
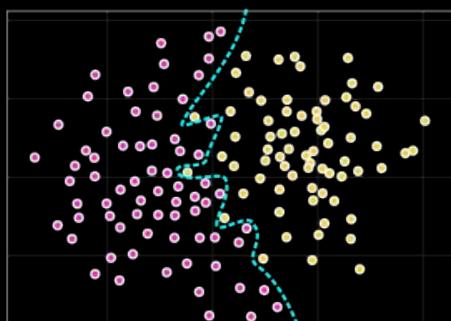
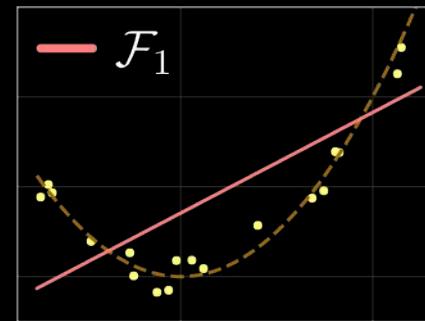
**Also no base code for this exercise (learn to define it yourself)**

# Understanding capacity

## Classification



## Regression



## Underfitting

- Function family is **too poor**
- Capacity is **too low** for this problem

## Overfitting

- Function family is **too rich**
- Capacity is **too high** for this problem

## Optimal capacity

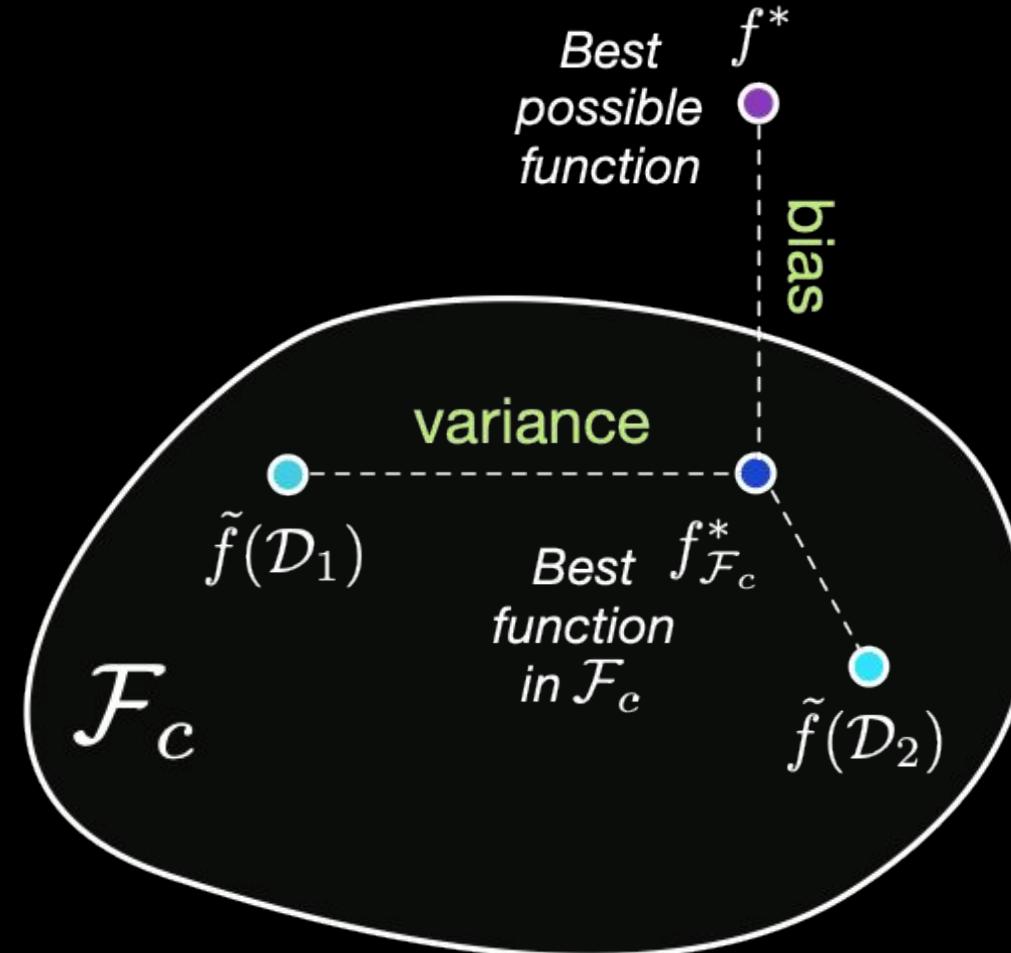
- Function family is **adequate**
- **Optimal capacity** for this problem

# Bias and variance decomposition

This problem lies in the **bias** and **variance** of our approximation

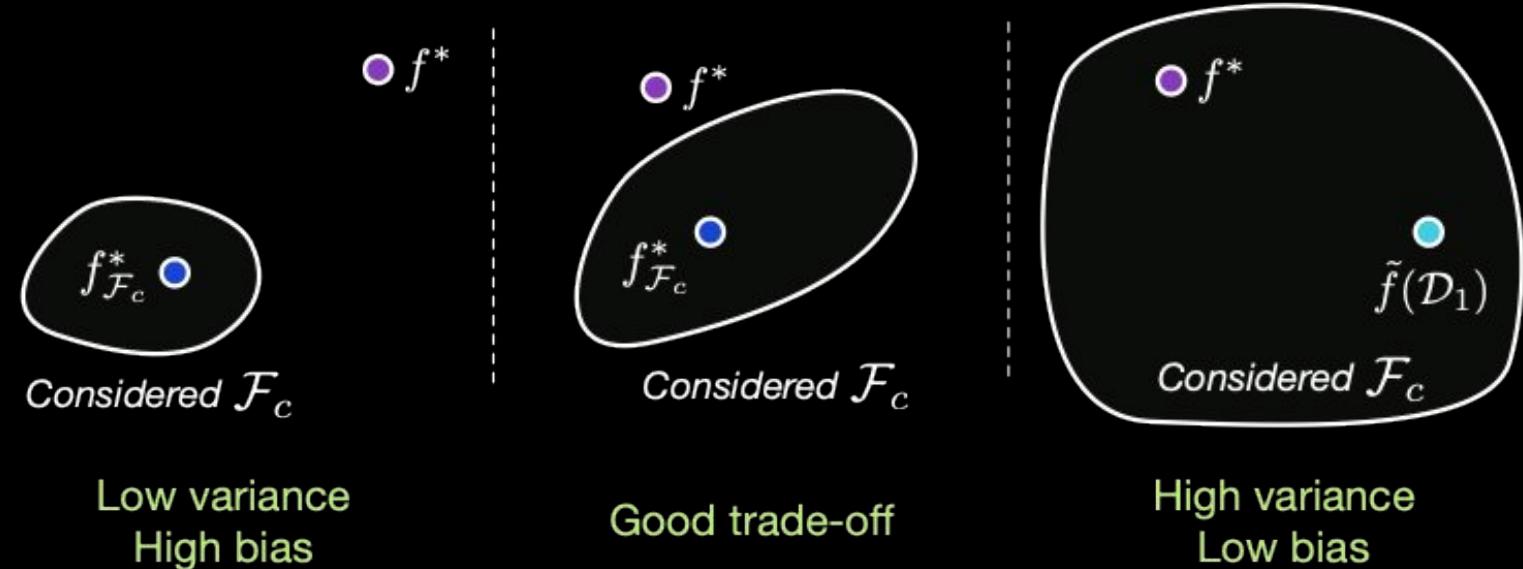
Space of all  
possible  
functions

*Considered  
function  
family*



# Bias and variance - details

So how can we adequately choose the family ?



Problem in the variance aspect is the discrepancy lying between

**Expected risk**  
(what we want)

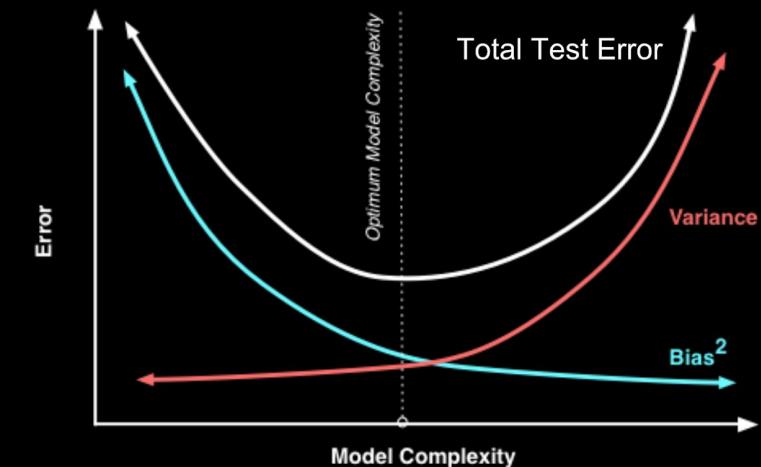
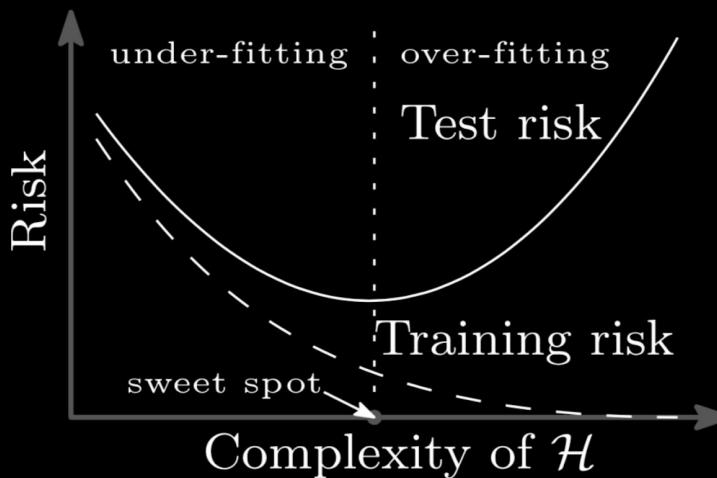
$$\mathcal{R}(f_\theta) = \mathbb{E}_{(\mathbf{x}, \mathbf{y}) \sim p(\mathbf{x}, \mathbf{y})} [\mathcal{L}(f_\theta(\mathbf{x}), \mathbf{y})]$$

**Empirical risk**  
(what we have)

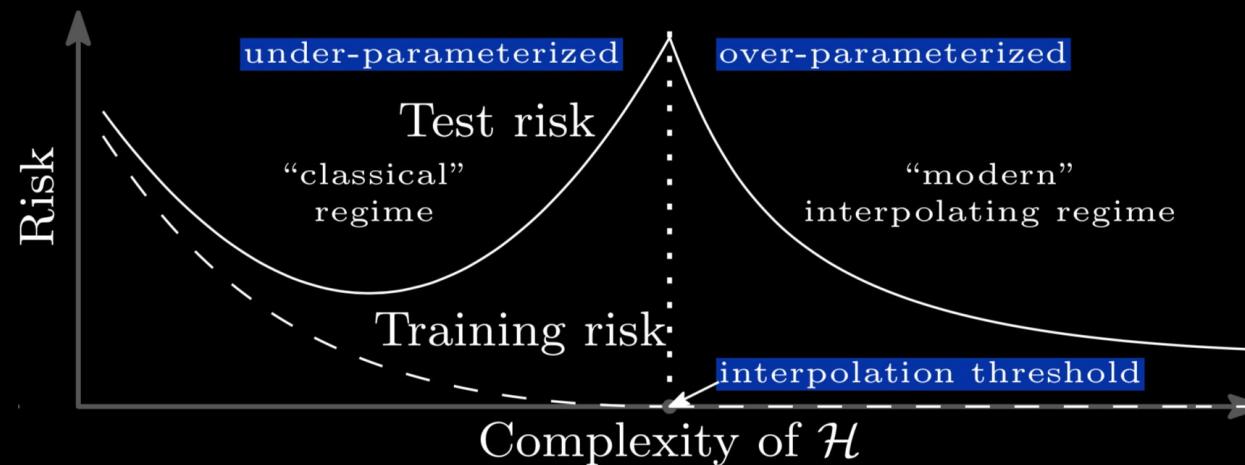
$$\hat{\mathcal{R}}(f_\theta, \mathcal{D}) = \frac{1}{|\mathcal{D}|} \sum_{(\mathbf{x}, \mathbf{y}) \in \mathcal{D}} \mathcal{L}(f_\theta(\mathbf{x}), \mathbf{y})$$

# Modern risk curves for deep learning

We used to think about  
**overfitting** (risk curves)



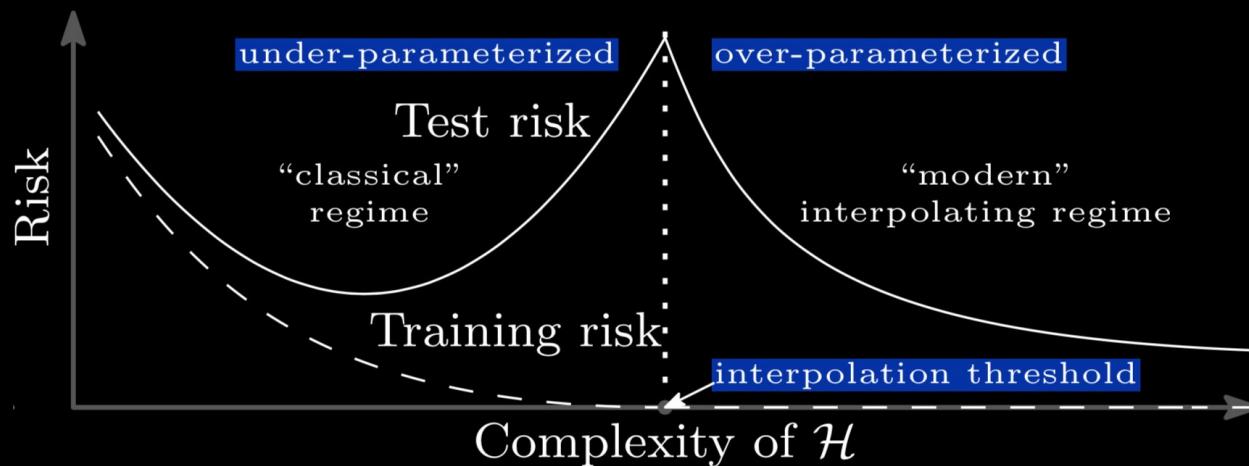
But **overparameterized** models have an **entirely different behavior**



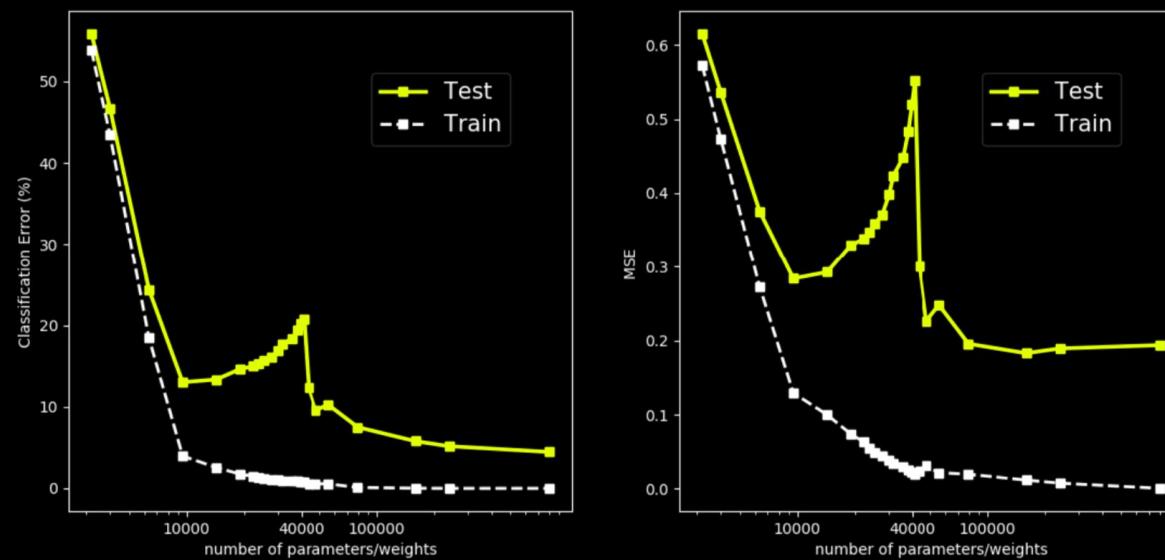
[Belkin, et al (2019). Reconciling modern machine-learning practice and the classical bias-variance trade-off.. PNAS, 116(32).]

# Modern risk curves for deep learning

This idealized curve  
can be validated



**Experimental validation**  
MNIST dataset  
Clear regime difference



[Belkin, et al (2019). Reconciling modern machine-learning practice and the classical bias-variance trade-off.. PNAS, 116(32).]

# Scaling laws and data

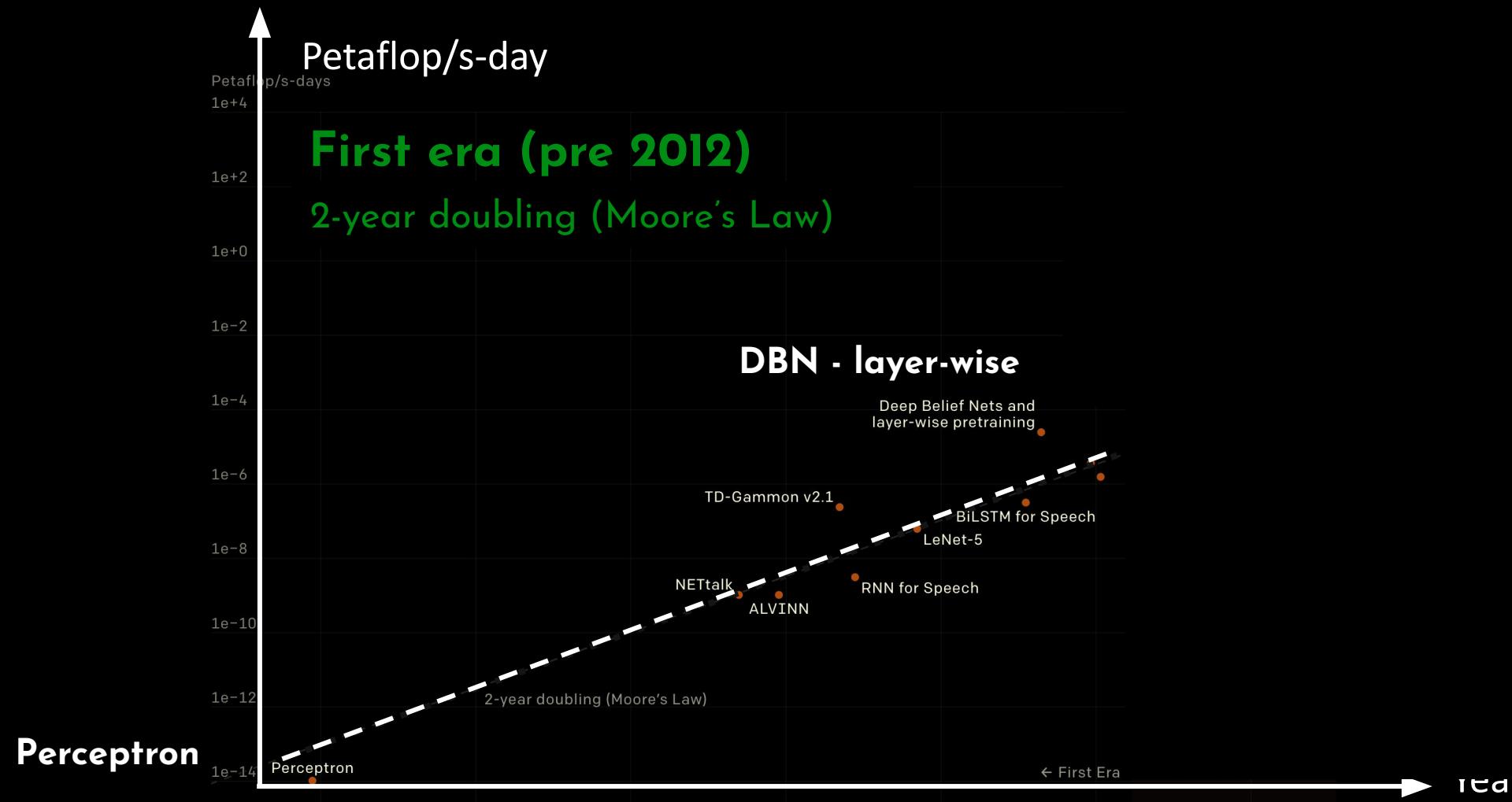
---

## — How data naturally regularizes complexity

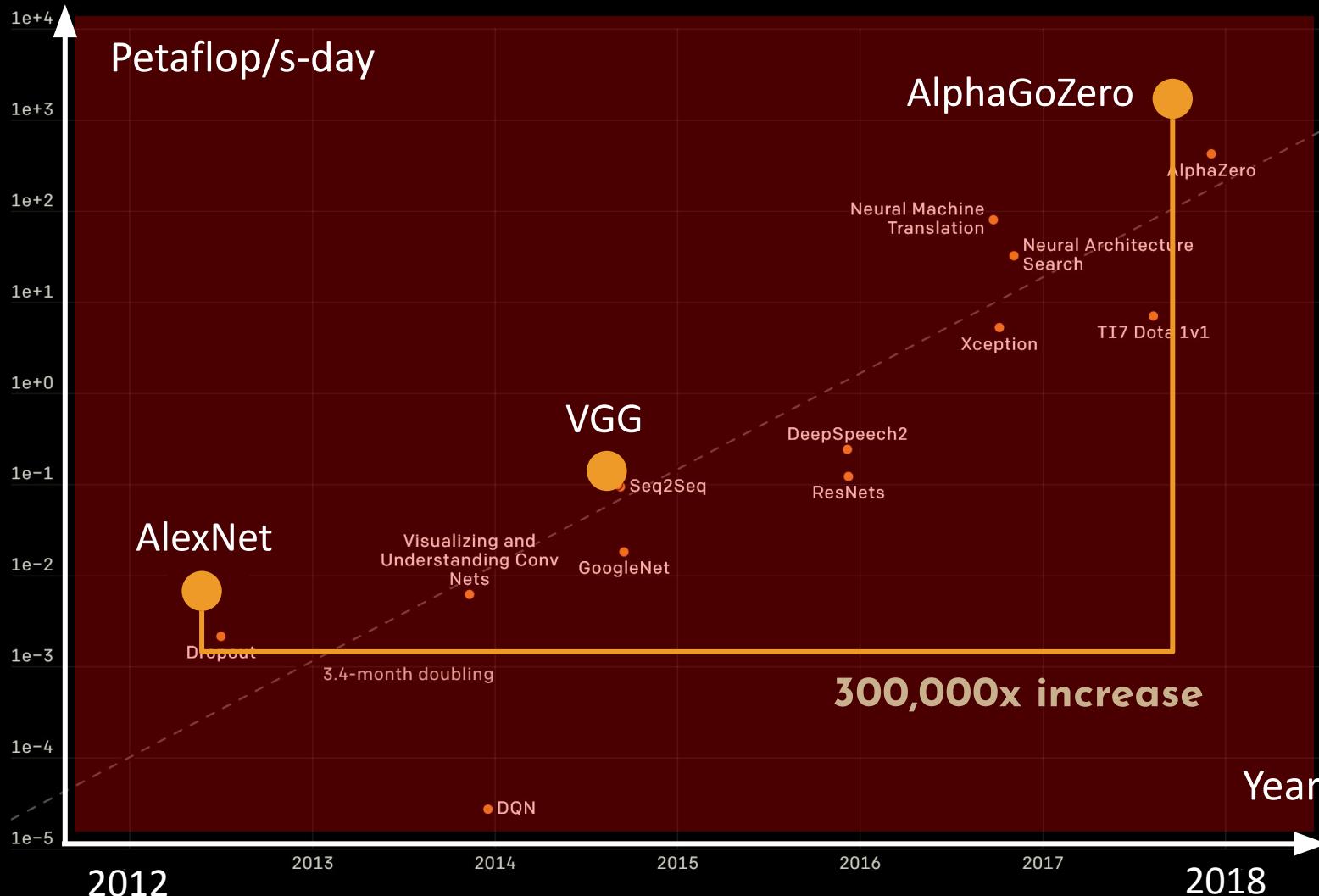
# Eras in machine learning

## Two distinct eras in ML

Dario Amodei and Danny Hernandez. AI and compute (2018)



# Eras in machine learning (zooming in)



Dario Amodei and Danny Hernandez. AI and compute (2018)

# Consequences and observations

---

## Direct consequences of this accuracy race

- Huge environmental issues
- Precludes the use in non-specialized (user-side) hardware
- Even less possible to embed such systems



*175 billion parameters, takes 355 years on a single GPU to train*

GPT-3

**Carbon footprint for training equivalent to driving to the moon and back [3]**



*Dr. James O'Donoghue using NASA imagery.*

# Observation #2 - Occam's razor and complexity

The well-known Occam's razor principle

« Entities should not be multiplied without necessity. »

Or equivalently ...

« The simplest explanation is most likely the right one »

More formal view is the Minimum Description Length (MDL) principle

« Comprehension is compression »

Given a set of models  $\mathcal{H}_1, \dots, \mathcal{H}_n$  that can explain the dataset  $\mathcal{D}$

The MDL principle states that

$$\mathcal{H}^{best} = \operatorname{argmin}_{\mathcal{H}} [L(\mathcal{H}) + L(\mathcal{D}|\mathcal{H})]$$

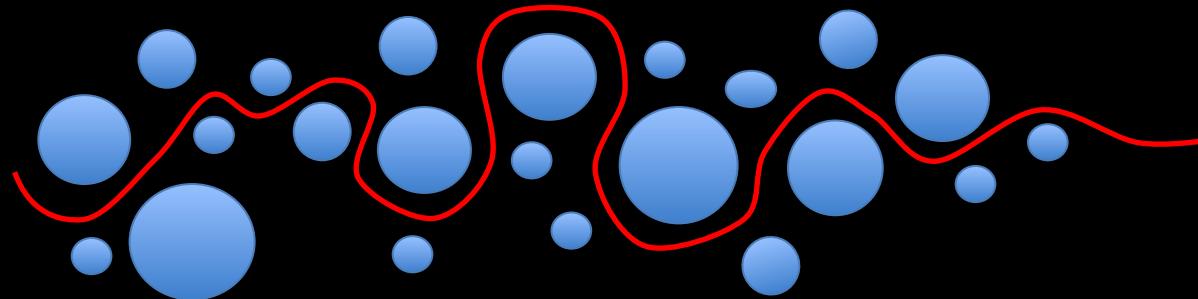
$L(\mathcal{H})$  : Length of the description of model  $\mathcal{H}$  in bits

$L(\mathcal{D}|\mathcal{H})$ : Length of describing data  $\mathcal{D}$  encoded by model  $\mathcal{H}$  in bits

# Observation #1 - Complexity of the problem

---

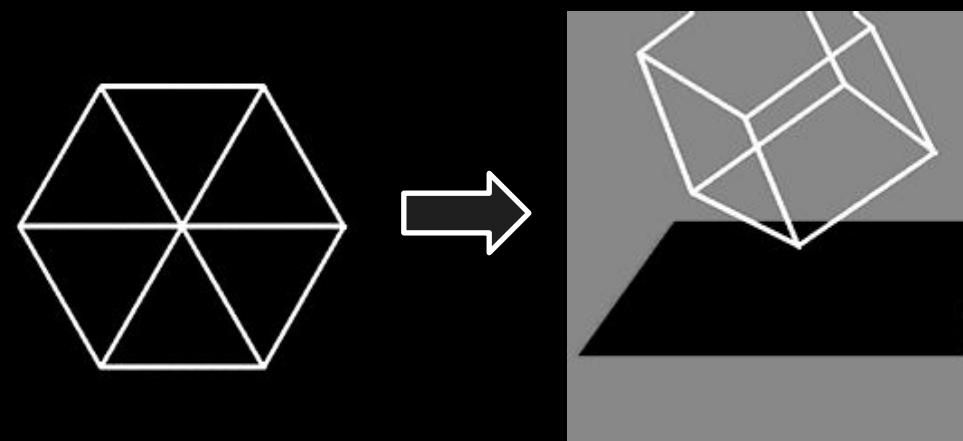
Imagine you observe  
the path of an ant



Seems complicated ... you could say the ant is extremely smart  
You look closer, there are rocks that the ant is just avoiding

**Complexity can be consequence of the environment not the program**

Now imagine you are trying to understand this behavior

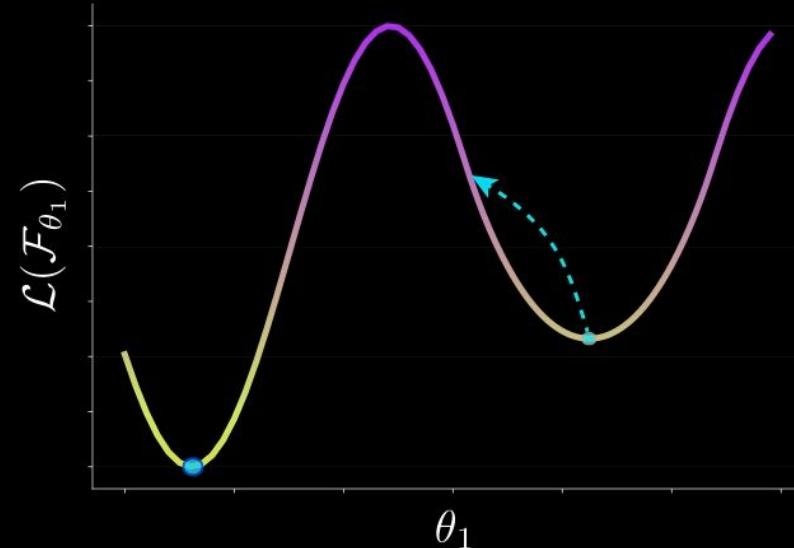


**Notion of  
problem  
dimensionality**

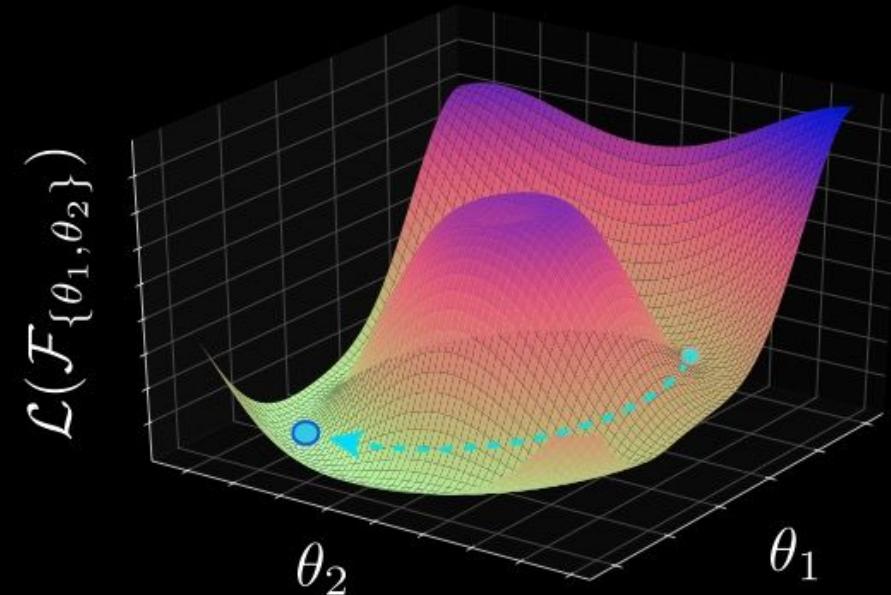
# Overparametrization

One potential explanation on why overparameterization works

Imagine the loss landscape (error given parameter values)



With a single parameter, hard to escape minima



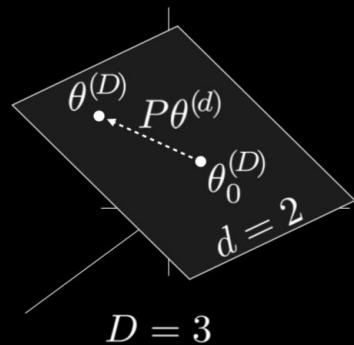
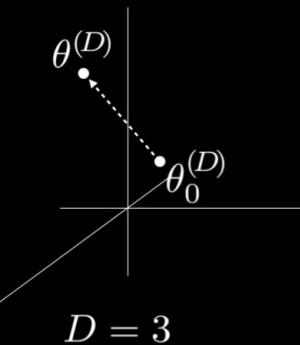
Extra parameter, we can « go around » (idealized)

Overparametrization **simplifies the training** ... allowing to find a **better solution faster**

Does not mean that the final solution needs all these parameters

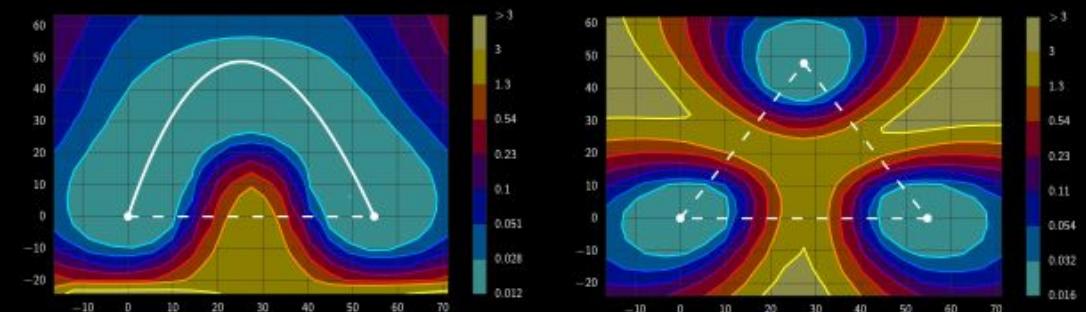
# Intriguing perspectives for understanding

## Mode connectivity

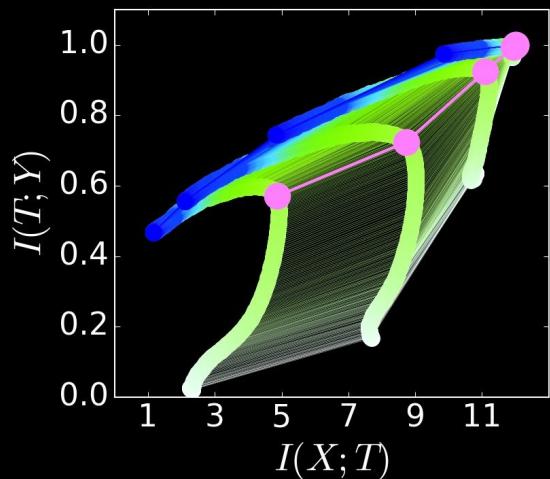


Mode (efficient solutions) connectivity  
How loss surfaces have « valleys »

Efficient solutions might live on a subspace  
Hence, we could simplify the models



Garipov, et al. (2018). Loss surfaces, mode connectivity, and fast ensembling of dnns. In Advances in Neural Information Processing Systems (pp. 8789-8798).



## Information Bottleneck theory

Two distinct period in training

- First one maximizes *mutual information*
- Second one performs compression

Shwartz-Ziv, R., & Tishby, N. (2017). Opening the black box of deep neural networks via information. arXiv preprint arXiv:1703.00810.

# Recalling our program

---

## 3 major types of learning

- **Supervised learning** is inferring a function from labeled training data
- **Unsupervised learning** is trying to find hidden structure in unlabeled data
- **Reinforcement learning** is acting to maximize a notion of cumulative reward

**Self-supervised learning** is using the data itself directly to define tasks to solve

## Detailed program

**Supervised**  
(discriminative)

**Unsupervised**  
(generative)

1. Introduction
2. Machine learning
3. Neural networks
4. Advanced networks
5. Deep learning
6. Probabilities and Bayesian inference
7. Latent models, EM and GMM
8. Approximate inference
9. Variational Auto-Encoders (VAEs) and flows
10. Adversarial learning (GANs)
11. Diffusion models

# Working with creative data (music)

## Real-world applications of this course

- Up to now we have talked on an abstract level
- We observe relationships between  $\mathcal{X}$  and  $\mathcal{Y}$

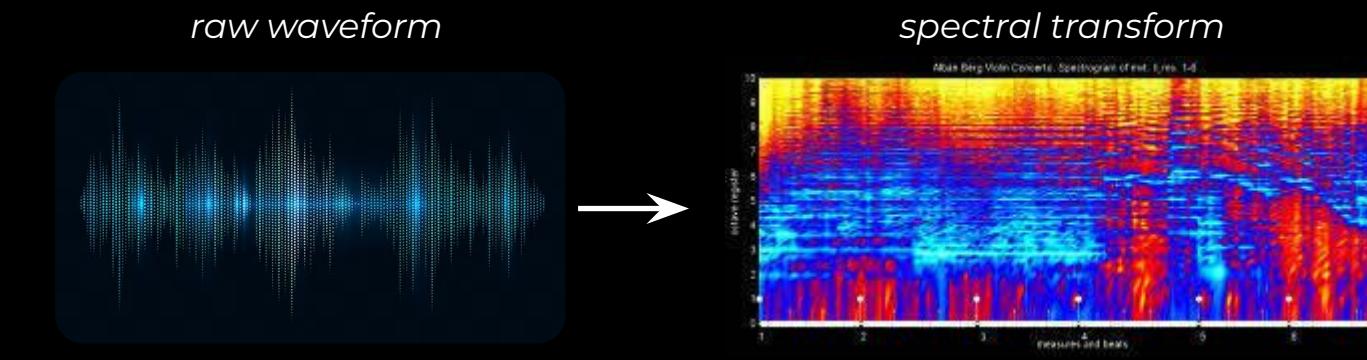
So what is the *nature* of this  $\mathcal{X}$  ?

This is defined by the **nature of the observations**

## Nature of audio observations

The major problem is the **dimensionality** of audio observations

In the case of waveform audio data, this problem is huge



5 seconds of 44.1k audio  
**220,500 dimensions**

2048 bands for 512 steps  
**220,500 dimensions**

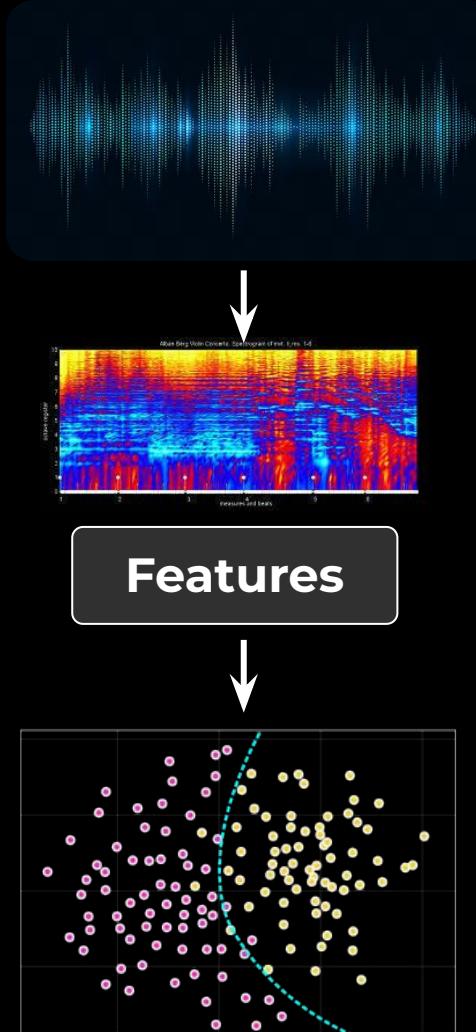
*known features*

- *Pitch*
- *Loudness*
- *Centroid*

Seminal approach of  
**feature-based learning**

# Feature-based learning

---



## Typical workflow prior to deep learning

How to define a well-optimizable problem

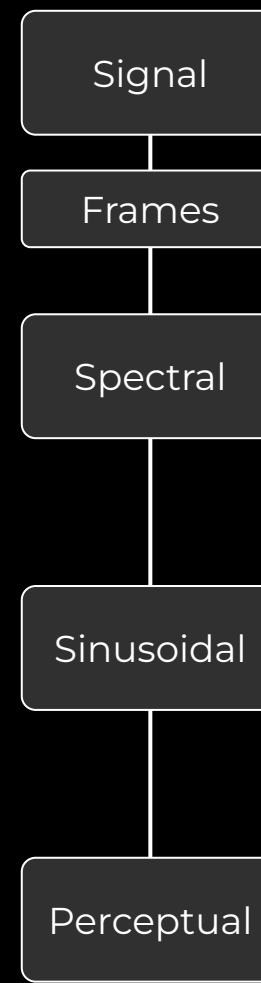
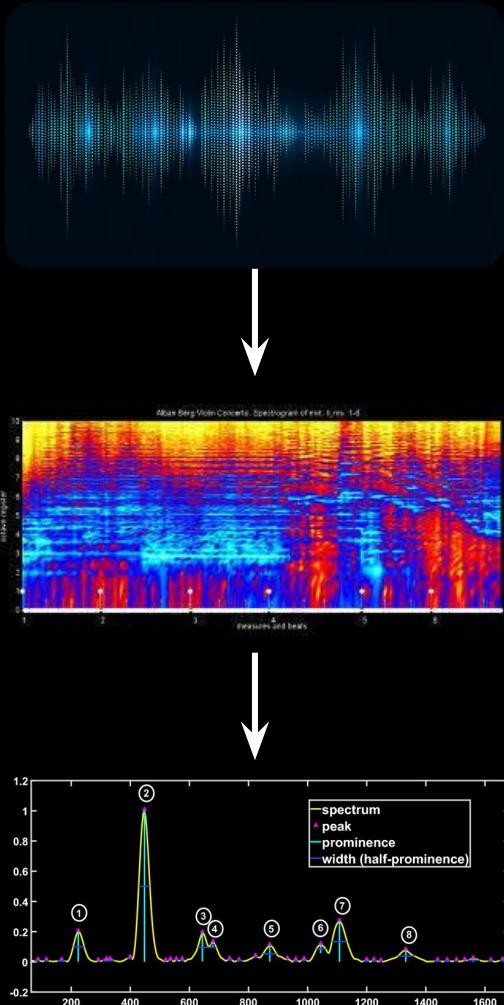
- First (pre-)process and normalize the observations
- Extract some features highly relevant to the problem
  - Highly problem-dependent
- Set of high-level features still associated to our classes
- Allows to use our simple learning algorithms

## Which features to use ?

- Highly depends on the task at hand !
- Years of research on audio features

# Audio features

Large available set of features to compute



## Instantaneous signal descriptors

- Energy envelope (RMS)
- Zero-crossing rate (Noisiness)
- Auto-correlation (pitch)

## Spectral transforms

- Mel-spectrogram
- Constant-Q transform
- Mel Frequency Cepstral Coefficients (MFCC)

## Harmonic descriptors

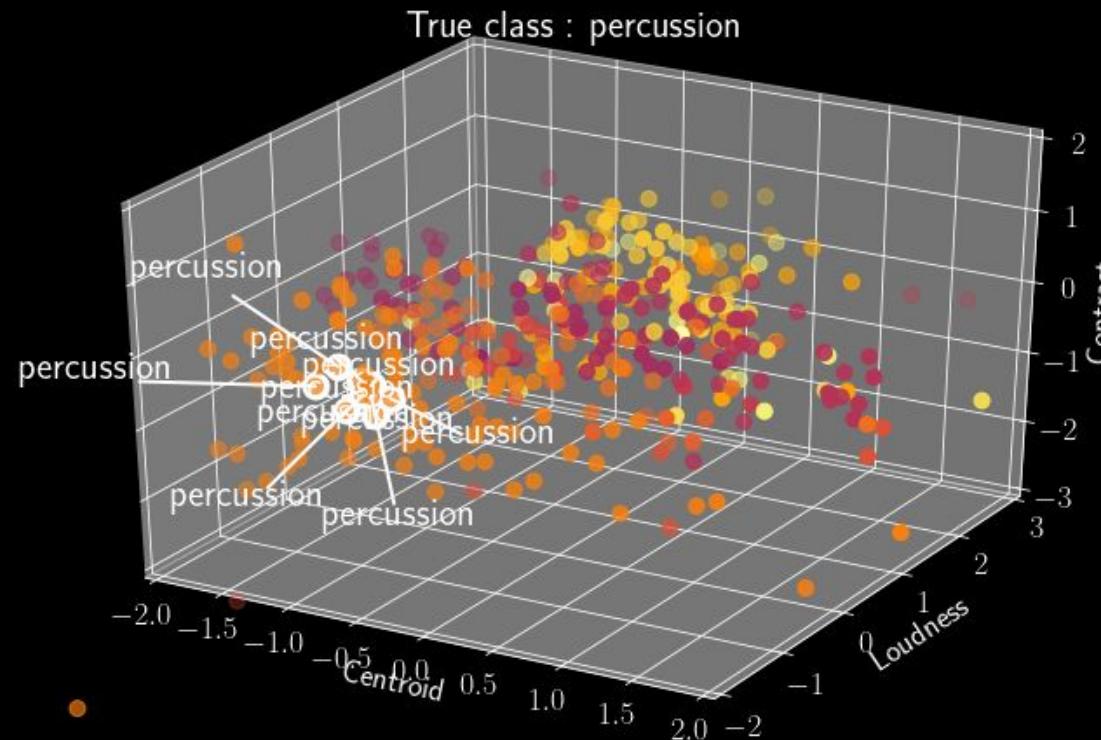
- Energy envelope (RMS)
- Zero-crossing rate (Noisiness)
- Auto-correlation (pitch)

## Perceptual descriptors

- Energy envelope (RMS)
- Zero-crossing rate (Noisiness)
- Auto-correlation (pitch)

# Feature-based learning

Typical audio feature spaces already provide interesting separation



- *Similar examples are usually grouped*
- Naïve way of addressing our classification = *look at neighbors*