

Deep learning

2.1. Loss and risk

François Fleuret

<https://fleuret.org/dlc/>



UNIVERSITÉ
DE GENÈVE

The general objective of machine learning is to capture regularity in data to make predictions.

In our regression example, we modeled age and blood pressure as being linearly related, to predict the latter from the former.

There are multiple types of inference that we can roughly split into three categories:

- Classification (e.g. object recognition, cancer detection, speech processing),
- regression (e.g. customer satisfaction, stock prediction, epidemiology), and
- density estimation (e.g. outlier detection, data visualization, sampling/synthesis).

The standard formalization for classification and regression considers a measure of probability

$$\mu_{X,Y}$$

over the observation/value of interest, and i.i.d. training samples

$$(x_n, y_n), \quad n = 1, \dots, N,$$

and for density estimation

$$\mu_X$$

and

$$x_n, \quad n = 1, \dots, N.$$

Intuitively, for classification a often intuitive interpretation is

$$\mu_{X,Y}(x,y) = \mu_{X|Y=y}(x) P(Y=y)$$

that is, draw Y first, and given its value, generate X .

So the conditional distribution

$$\mu_{X|Y=y}$$

stands for the distribution of the observable signal for the class y (e.g. “sound of an /ē/”, “image of a cat”).

For regression, one would interpret the joint law more naturally as

$$\mu_{X,Y}(x,y) = \mu_{Y|X=x}(y) \mu_X(x)$$

which would be: first, generate X , and given its value, generate Y .

In the simple cases

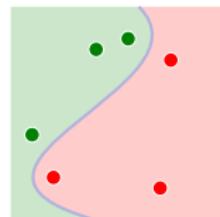
$$Y = f(X) + \epsilon$$

where f is the deterministic dependency between x and y (e.g. affine), and ϵ is a random noise, independent of X (e.g. Gaussian).

With such a probabilistic perspective, we can more precisely define the three types of inferences we introduced before:

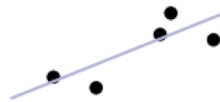
Classification,

- (X, Y) random variables on $\mathcal{Z} = \mathbb{R}^D \times \{1, \dots, C\}$,
- we want to estimate $\text{argmax}_y P(Y = y | X = x)$.



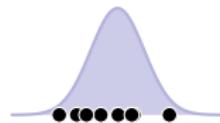
Regression,

- (X, Y) random variables on $\mathcal{Z} = \mathbb{R}^D \times \mathbb{R}$,
- we want to estimate $E(Y | X = x)$.



Density estimation,

- X random variable on $\mathcal{Z} = \mathbb{R}^D$,
- we want to estimate μ_X .



The boundaries between these categories are fuzzy:

- Regression allows to do classification through class scores.
- Density models allow to do classification thanks to Bayes' law.

etc.

Risk, empirical risk

Learning consists of finding in a set \mathcal{F} of functionals a “good” f^* (or its parameters’ values) usually defined through a loss

$$\ell : \mathcal{F} \times \mathcal{Z} \rightarrow \mathbb{R}$$

such that $\ell(f, z)$ increases with how wrong f is on z . For instance

- for classification:

$$\ell(f, (x, y)) = \mathbf{1}_{\{f(x) \neq y\}},$$

- for regression:

$$\ell(f, (x, y)) = (f(x) - y)^2,$$

- for density estimation:

$$\ell(q, z) = -\log q(z).$$

The loss may include additional terms related to f itself.

We are looking for an f with a small **expected risk**

$$R(f) = \mathbb{E}_Z (\ell(f, Z)),$$

which means that our learning procedure would ideally choose

$$f^* = \operatorname*{argmin}_{f \in \mathcal{F}} R(f).$$

Although this quantity is unknown, if we have i.i.d. training samples

$$\mathcal{D} = \{Z_1, \dots, Z_N\},$$

we can compute an estimate, the **empirical risk**:

$$\hat{R}(f; \mathcal{D}) = \hat{\mathbb{E}}_{\mathcal{D}} (\ell(f, Z)) = \frac{1}{N} \sum_{n=1}^N \ell(f, Z_n).$$

We have

$$\begin{aligned}\mathbb{E}_{Z_1, \dots, Z_N} (\hat{R}(f; \mathcal{D})) &= \mathbb{E}_{Z_1, \dots, Z_N} \left(\frac{1}{N} \sum_{n=1}^N \ell(f, Z_n) \right) \\ &= \frac{1}{N} \sum_{n=1}^N \mathbb{E}_{Z_n} (\ell(f, Z_n)) \\ &= \mathbb{E}_Z (\ell(f, Z)) \\ &= R(f).\end{aligned}$$

The empirical risk is an **unbiased estimator** of the expected risk.

Finally, given \mathcal{D} , \mathcal{F} , and ℓ , “learning” aims at computing

$$f^* = \operatorname{argmin}_{f \in \mathcal{F}} \hat{R}(f; \mathcal{D}).$$

- Can we bound $R(f)$ with $\hat{R}(f; \mathcal{D})$?

Yes if f is not chosen using \mathcal{D} . Since the Z_n are independent, we just need to take into account the variance of $\hat{R}(f; \mathcal{D})$.

- Can we bound $R(f^*)$ with $\hat{R}(f^*; \mathcal{D})$?



Unfortunately not simply, and not without additional constraints on \mathcal{F} .

For instance if $|\mathcal{F}| = 1$, we can!

Note that in practice, we call “loss” both the functional

$$\ell : \mathcal{F} \times \mathcal{Z} \rightarrow \mathbb{R}$$

and the empirical risk minimized during training

$$\mathcal{L}(f) = \frac{1}{N} \sum_{n=1}^N \ell(f, z_n).$$

Deep learning

2.2. Over and under fitting

François Fleuret

<https://fleuret.org/dlc/>



UNIVERSITÉ
DE GENÈVE

You want to hire someone, and you evaluate candidates by asking them ten technical yes/no questions.

Would you feel confident if you interviewed one candidate and they make a perfect score?

What about interviewing ten candidates and picking the best? What about interviewing one thousand?

Here the candidates are our models and the questions are the training examples used to pick the best one.

With

$$Q_k^n \sim \mathcal{B}(0.5), \quad n = 1, \dots, 1000, \quad k = 1, \dots, 10,$$

independent standing for “candidate n answers question k correctly”, we have

$$\forall n, \quad P(\forall k, Q_k^n = 1) = \frac{1}{1024}$$

and

$$P(\exists n, \forall k, Q_k^n = 1) \simeq 0.62.$$

So there is 62% chance that among 1,000 candidates answering completely at random, at least one will score perfectly.

Selecting a candidate based on a statistical estimator biases the said estimator for that candidate. And you need a greater number of “competence checks” if you have a larger pool of candidates.

Over and under-fitting, capacity. K -nearest-neighbors

A simple classification procedure is the “ K -nearest neighbors.”

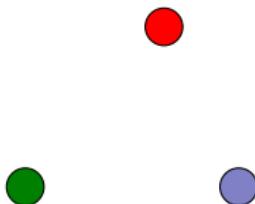
Given

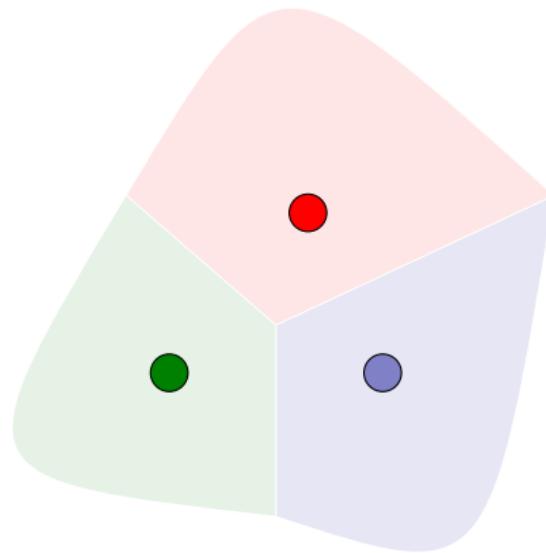
$$(x_n, y_n) \in \mathbb{R}^D \times \{1, \dots, C\}, \quad n = 1, \dots, N$$

to predict the y associated to a new x , take the y_n of the closest x_n :

$$\begin{aligned} n^*(x) &= \underset{n}{\operatorname{argmin}} \|x_n - x\| \\ f^*(x) &= y_{n^*(x)}. \end{aligned}$$

This recipe corresponds to $K = 1$, and makes the empirical training error zero.



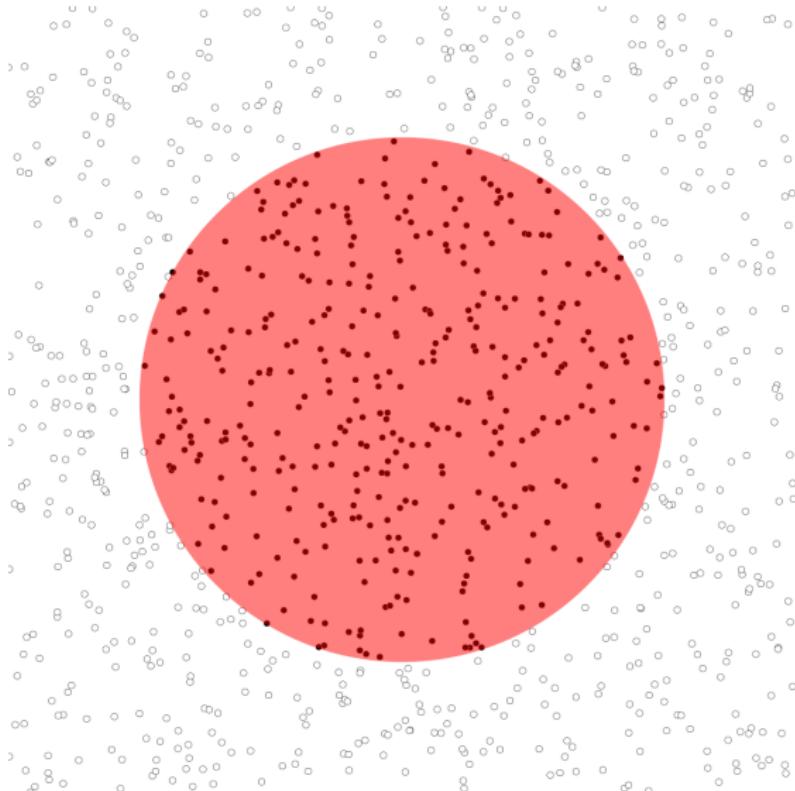


$$K = 1$$

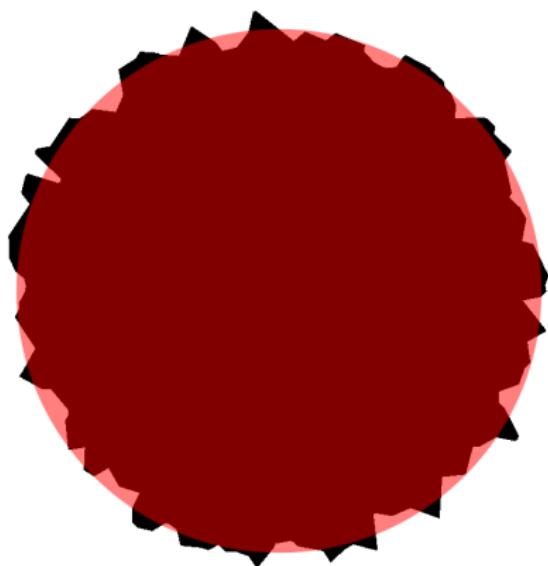
Under mild assumptions of regularities of $\mu_{X,Y}$, for $N \rightarrow \infty$ the asymptotic error rate of the 1-NN is less than twice the (optimal!) Bayes' Error rate.

It can be made more stable by looking at the $K > 1$ closest training points, and taking the majority vote.

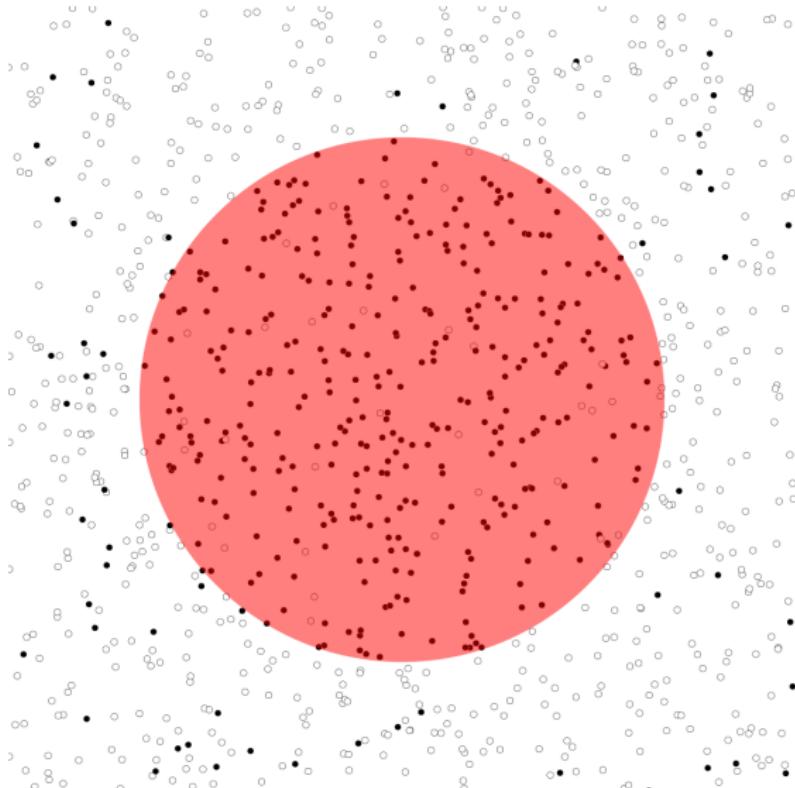
If we let also $K \rightarrow \infty$ “not too fast”, the error rate is the (optimal!) Bayes' Error rate.



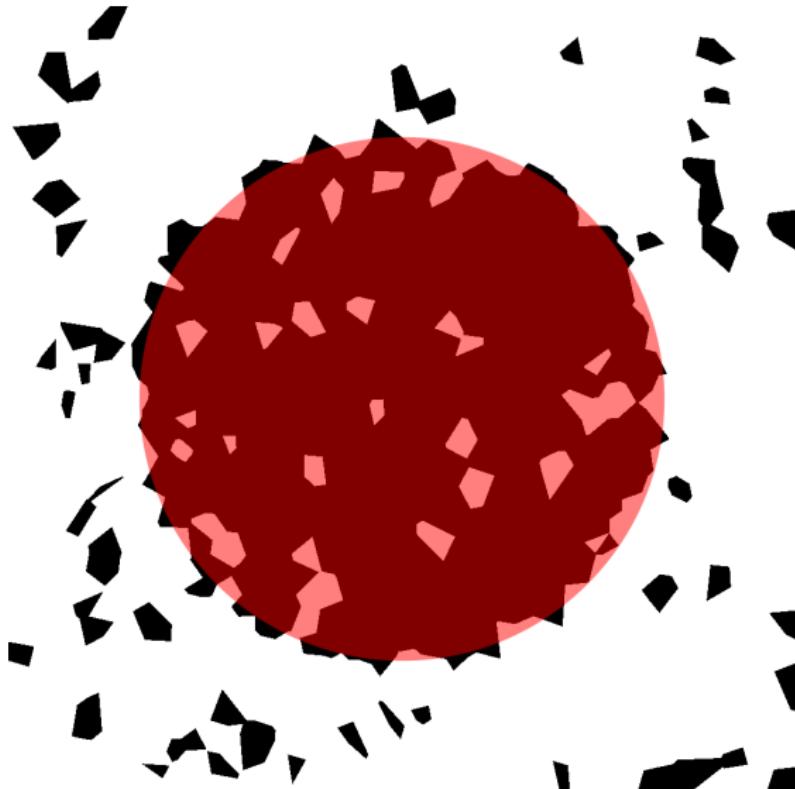
Training set



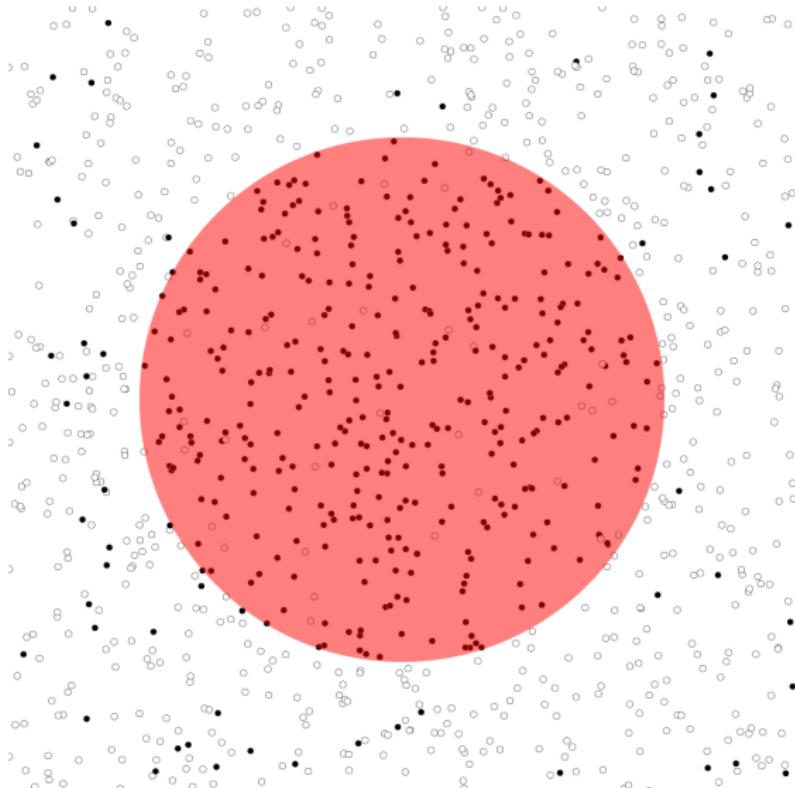
Prediction ($K=1$)



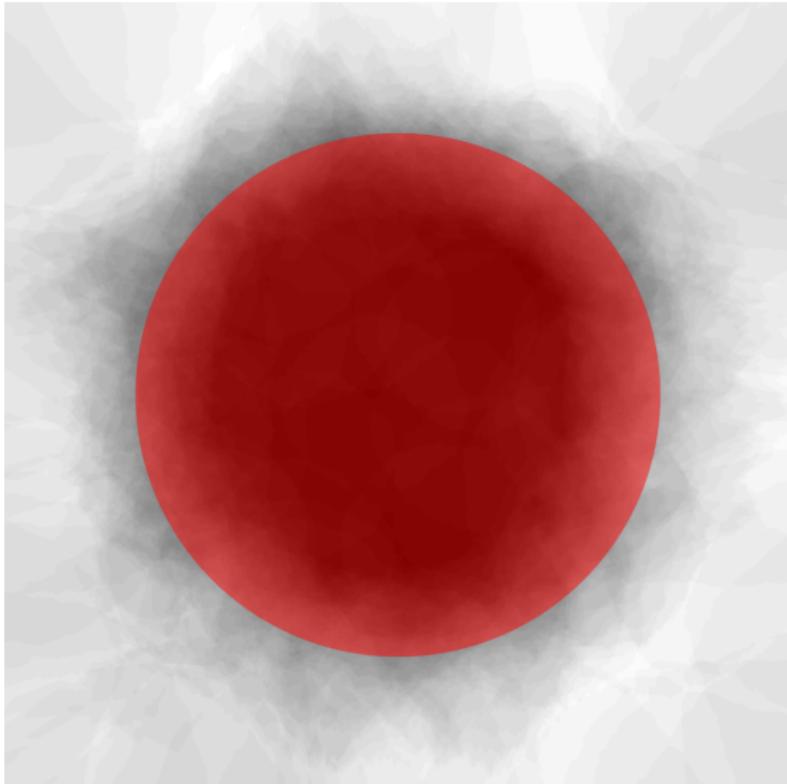
Training set



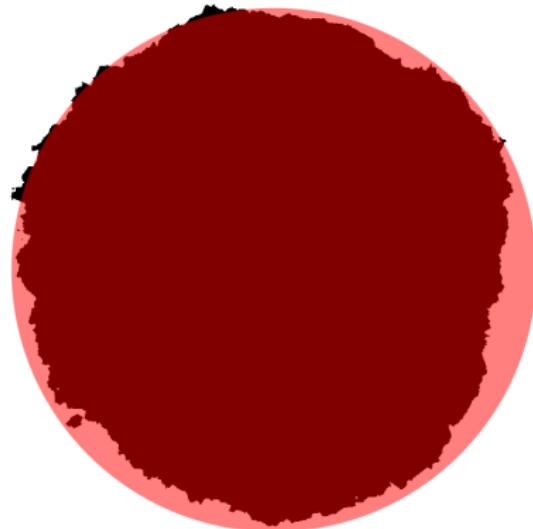
Prediction ($K=1$)



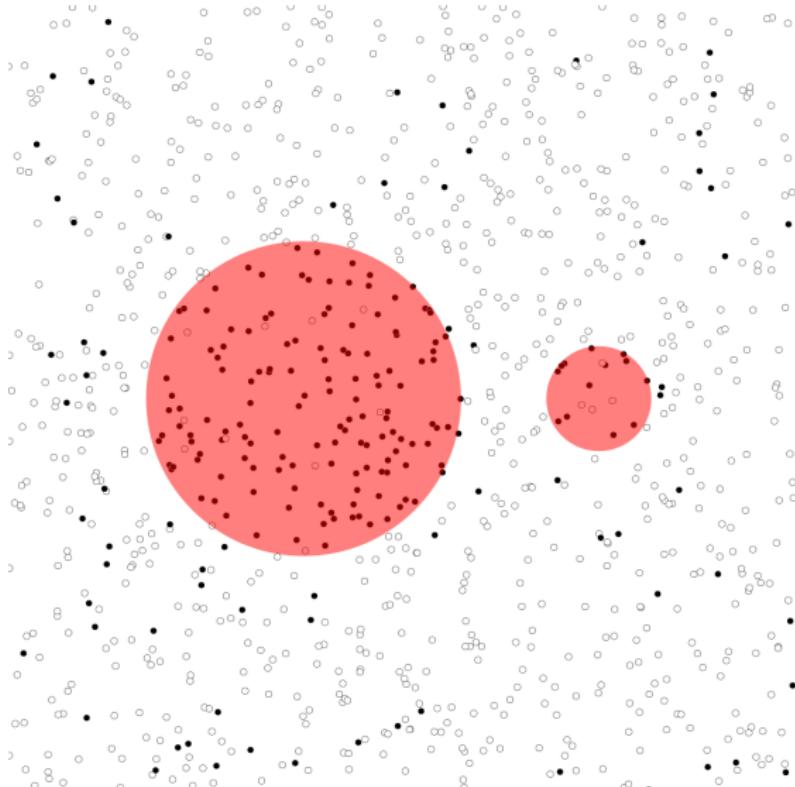
Training set



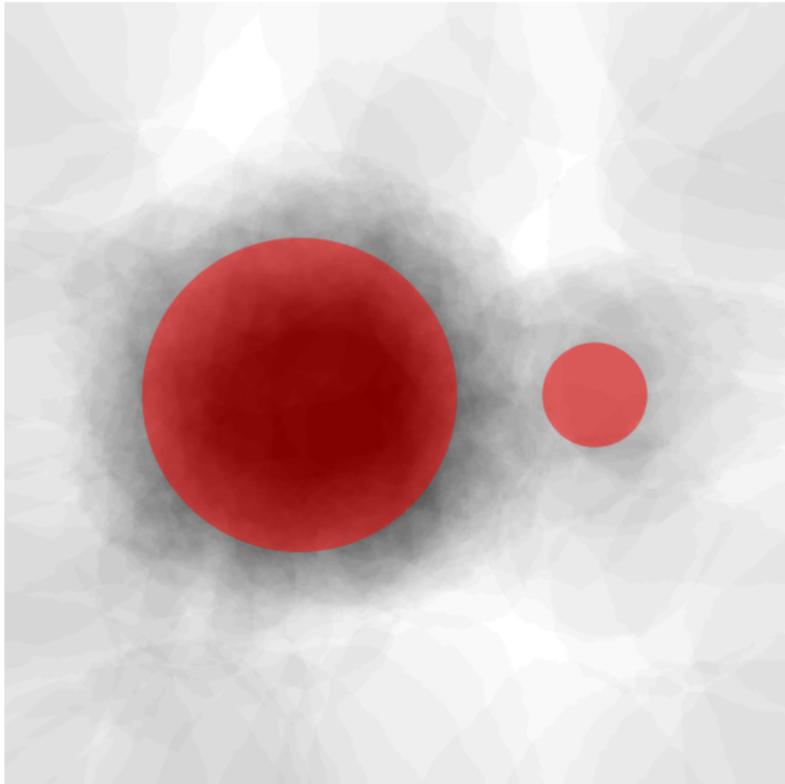
Votes (K=51)



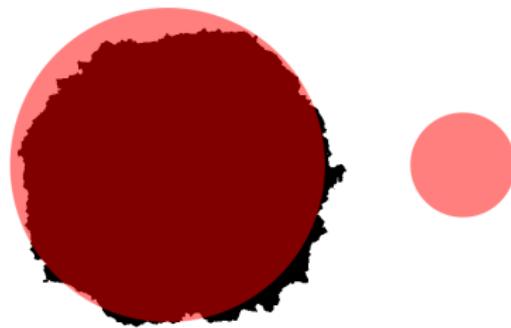
Prediction (K=51)



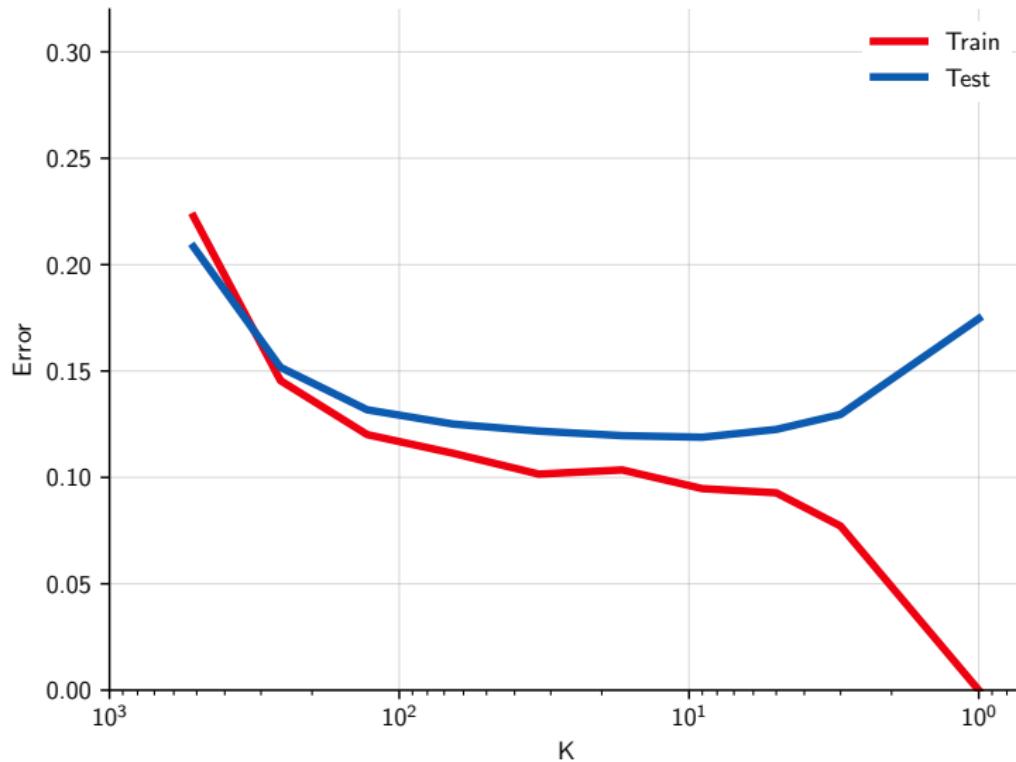
Training set

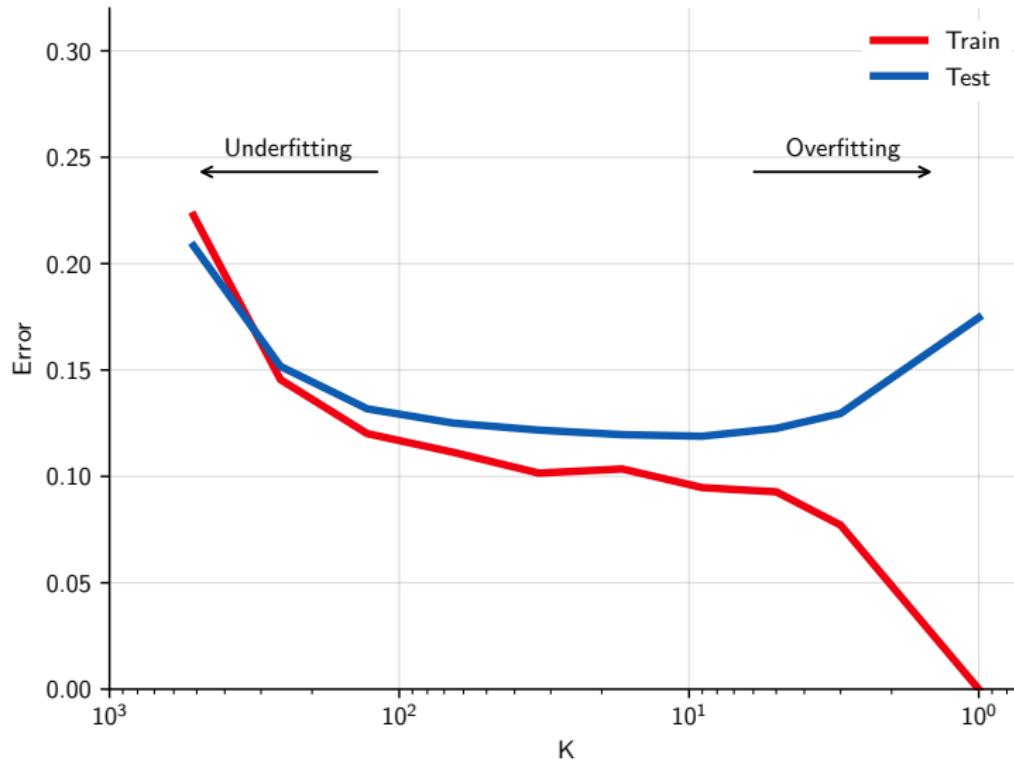


Votes ($K=51$)



Prediction ($K=51$)





Over and under-fitting, capacity, polynomials

Given a polynomial model

$$\forall x, \alpha_0, \dots, \alpha_D \in \mathbb{R}, f(x; \alpha) = \sum_{d=0}^D \alpha_d x^d.$$

and training points $(x_n, y_n) \in \mathbb{R}^2, n = 1, \dots, N$, the quadratic loss is

$$\begin{aligned}\mathcal{L}(\alpha) &= \sum_n (f(x_n; \alpha) - y_n)^2 \\ &= \sum_n \left(\sum_{d=0}^D \alpha_d x_n^d - y_n \right)^2 \\ &= \left\| \begin{pmatrix} x_1^0 & \dots & x_1^D \\ \vdots & & \vdots \\ x_N^0 & \dots & x_N^D \end{pmatrix} \begin{pmatrix} \alpha_0 \\ \vdots \\ \alpha_D \end{pmatrix} - \begin{pmatrix} y_1 \\ \vdots \\ y_N \end{pmatrix} \right\|^2.\end{aligned}$$

Hence, minimizing this loss is a standard quadratic problem, for which we have efficient algorithms.

$$\underset{\alpha}{\operatorname{argmin}} \left\| \begin{pmatrix} x_1^0 & \dots & x_1^D \\ \vdots & & \vdots \\ x_N^0 & \dots & x_N^D \end{pmatrix} \begin{pmatrix} \alpha_0 \\ \vdots \\ \alpha_D \end{pmatrix} - \begin{pmatrix} y_1 \\ \vdots \\ y_N \end{pmatrix} \right\|^2$$

```
def fit_polynomial(D, x, y):
    # Broadcasting magic
    X = x[:, None] ** torch.arange(0, D + 1)[None]

    # Least square solution
    return torch.linalg.lstsq(X, y).solution
```

```

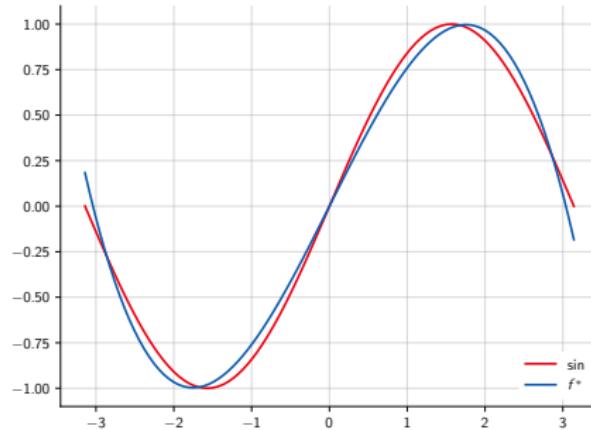
D, N = 4, 100
x = torch.linspace(-math.pi, math.pi, N)
y = x.sin()
alpha = fit_polynomial(D, x, y)

X = x[:, None] ** torch.arange(0, D + 1)[None]

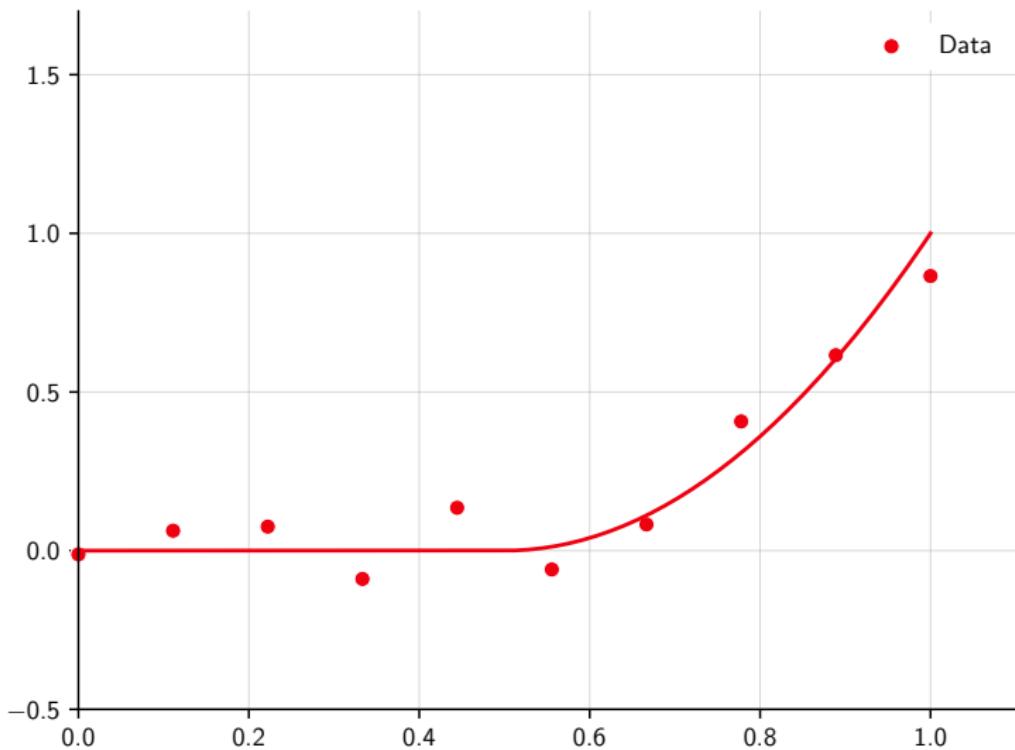
y_hat = X @ alpha

for k in range(N):
    print(x[k].item(), y[k].item(), y_hat[k].item())

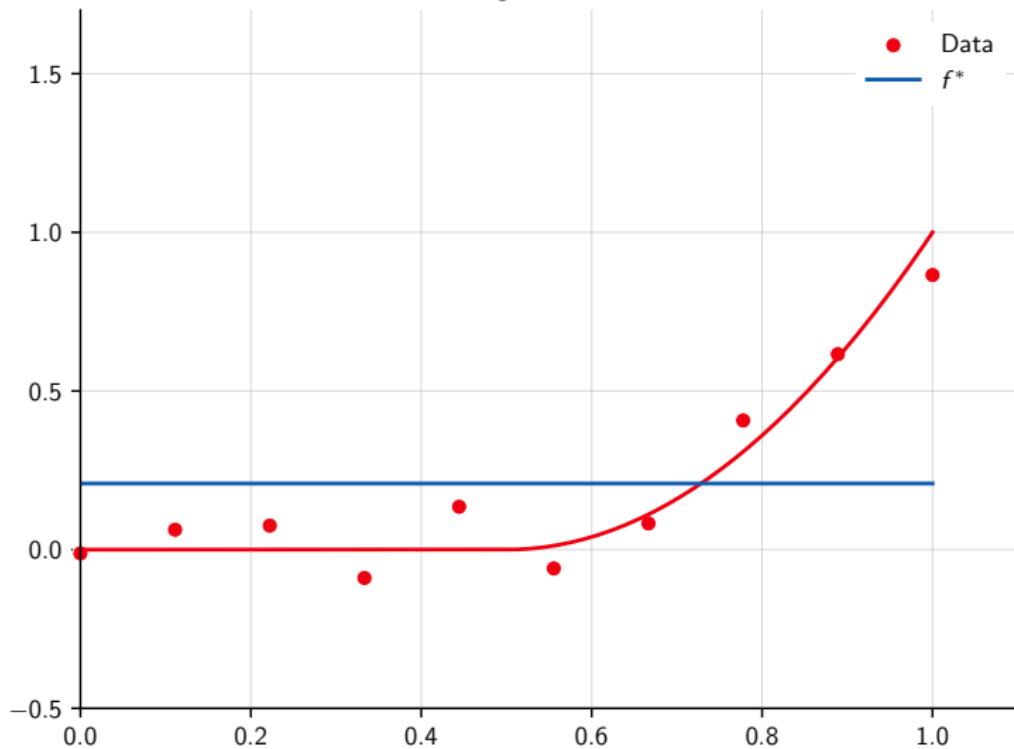
```



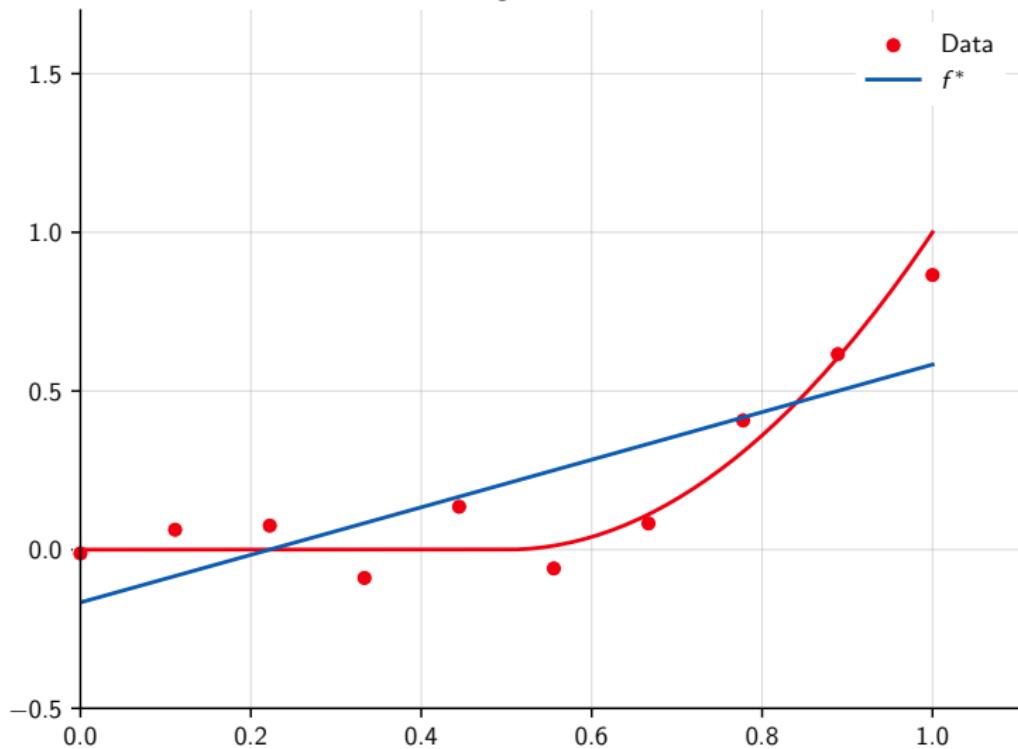
We can use this model to illustrate how the prediction changes when we increase the degree or the regularization.



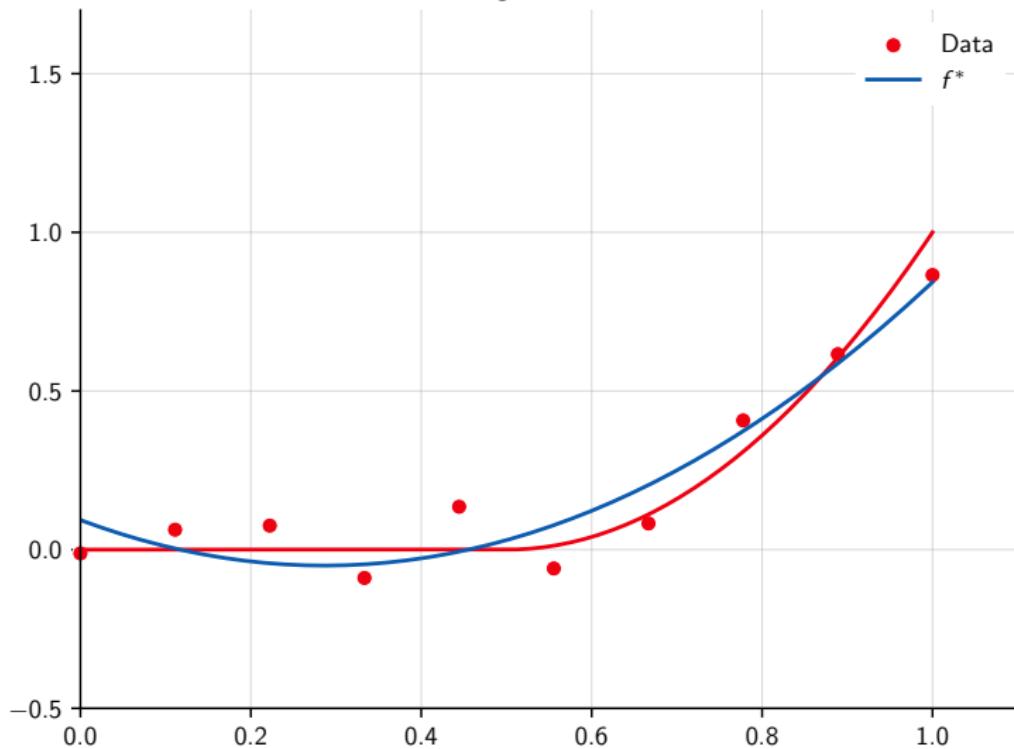
Degree $D = 0$



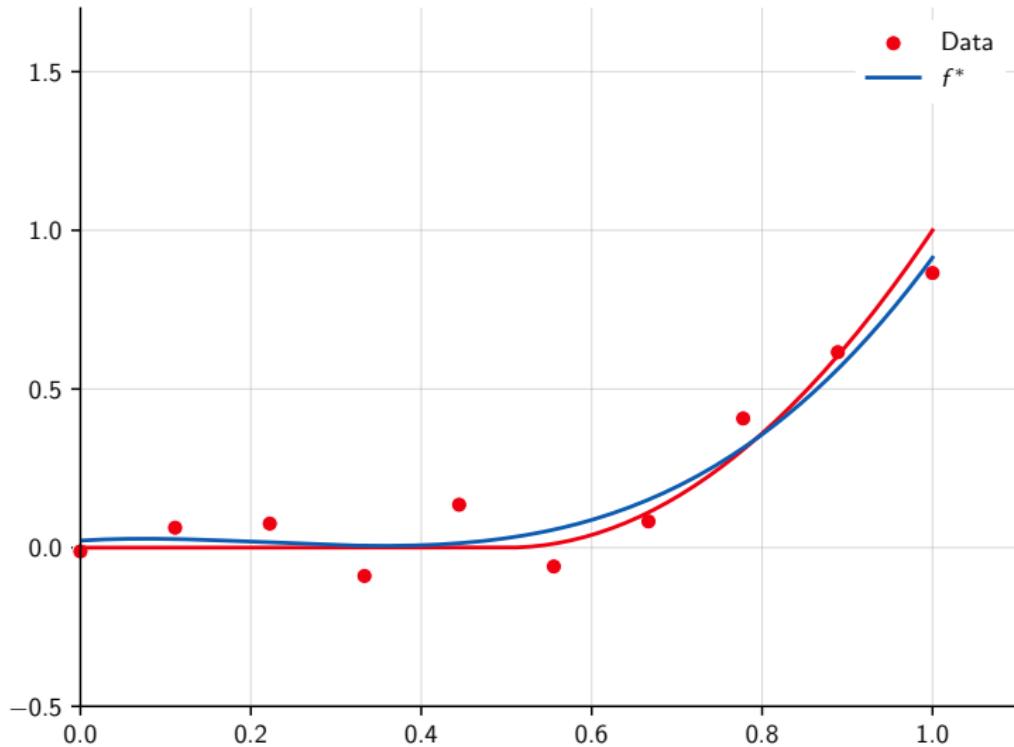
Degree $D = 1$



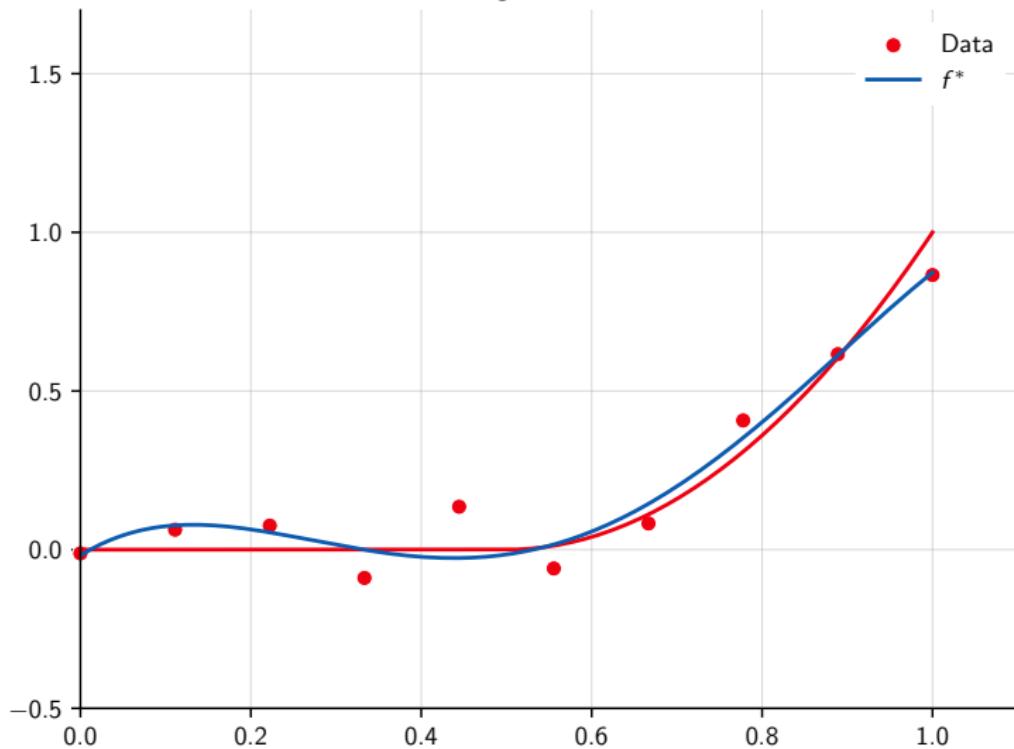
Degree $D = 2$



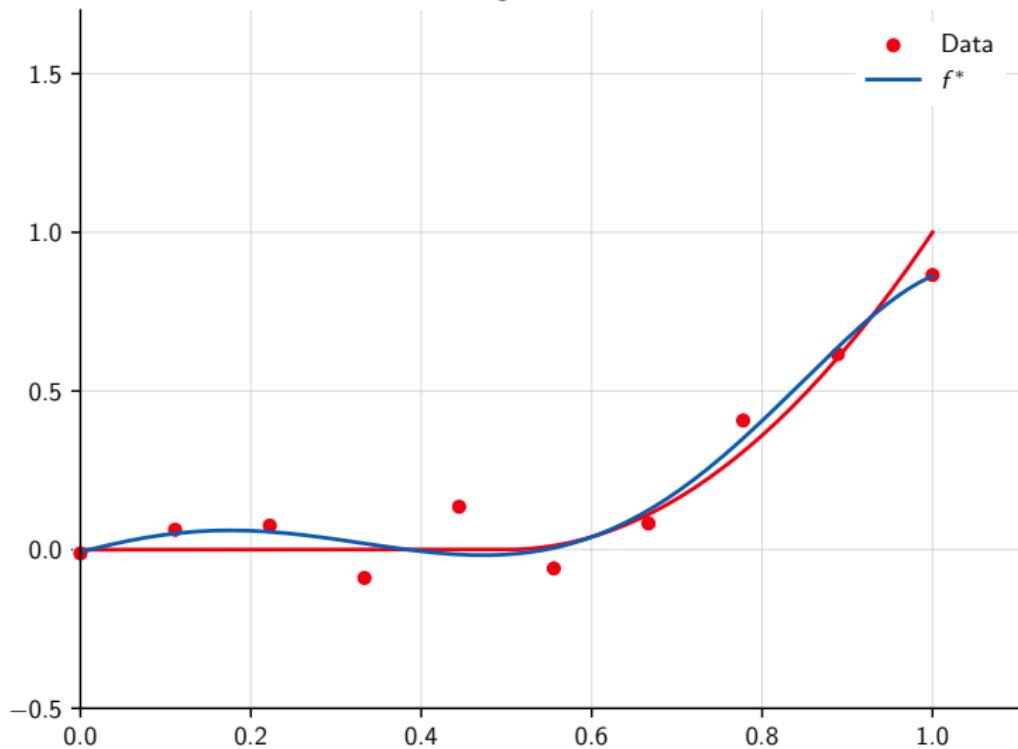
Degree $D = 3$



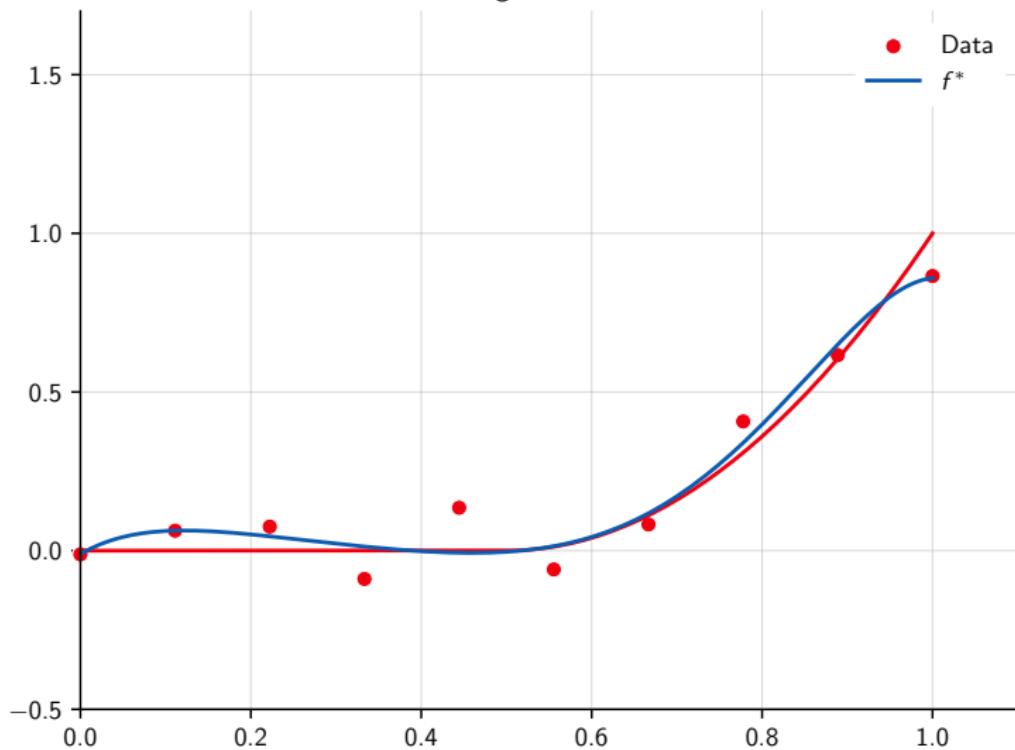
Degree $D = 4$



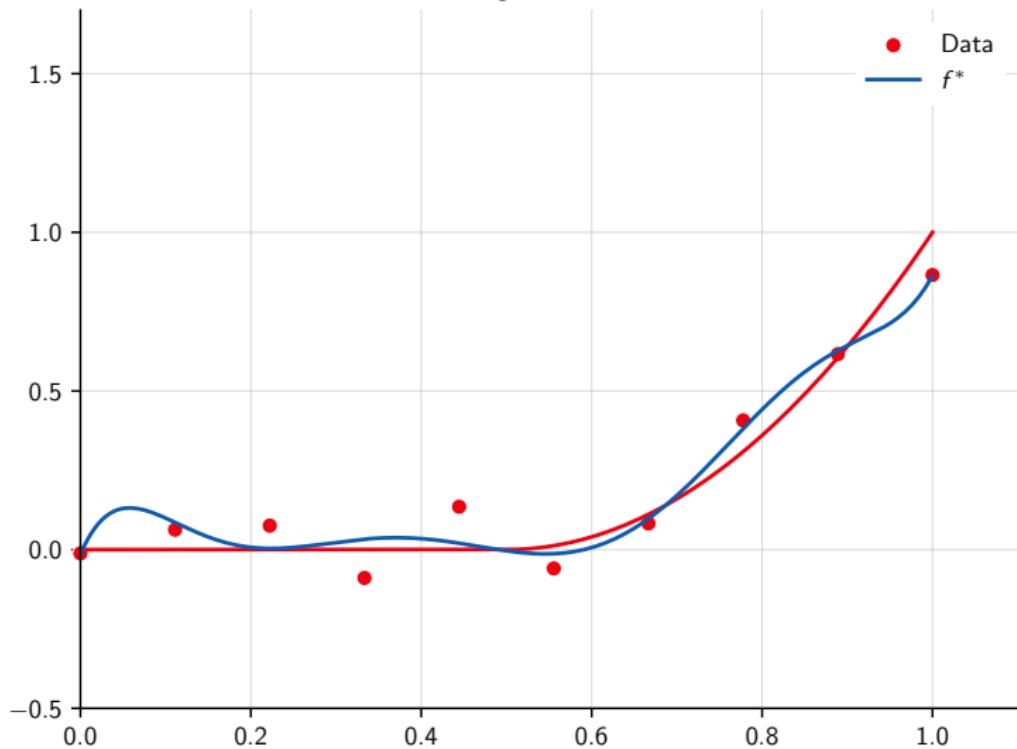
Degree $D = 5$



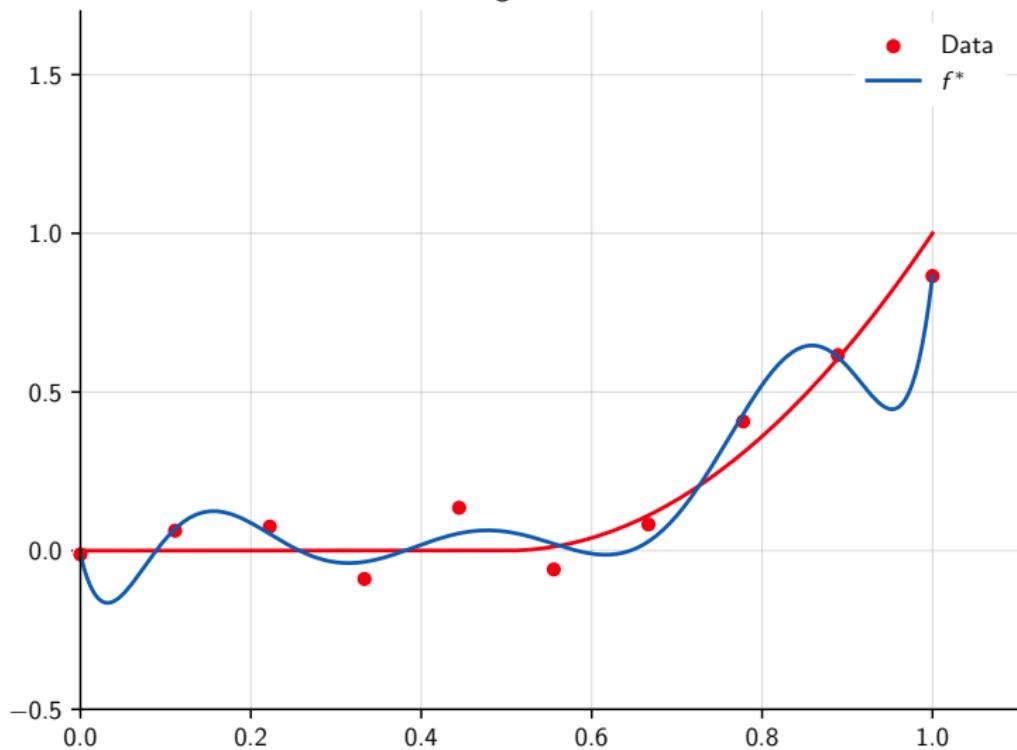
Degree $D = 6$



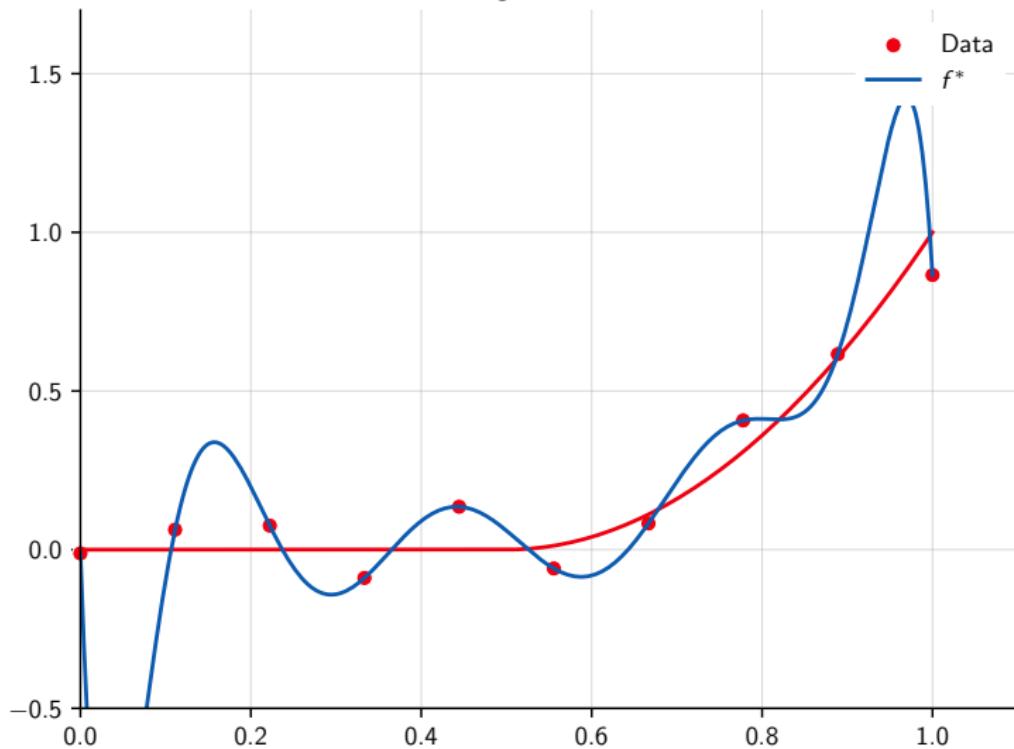
Degree $D = 7$

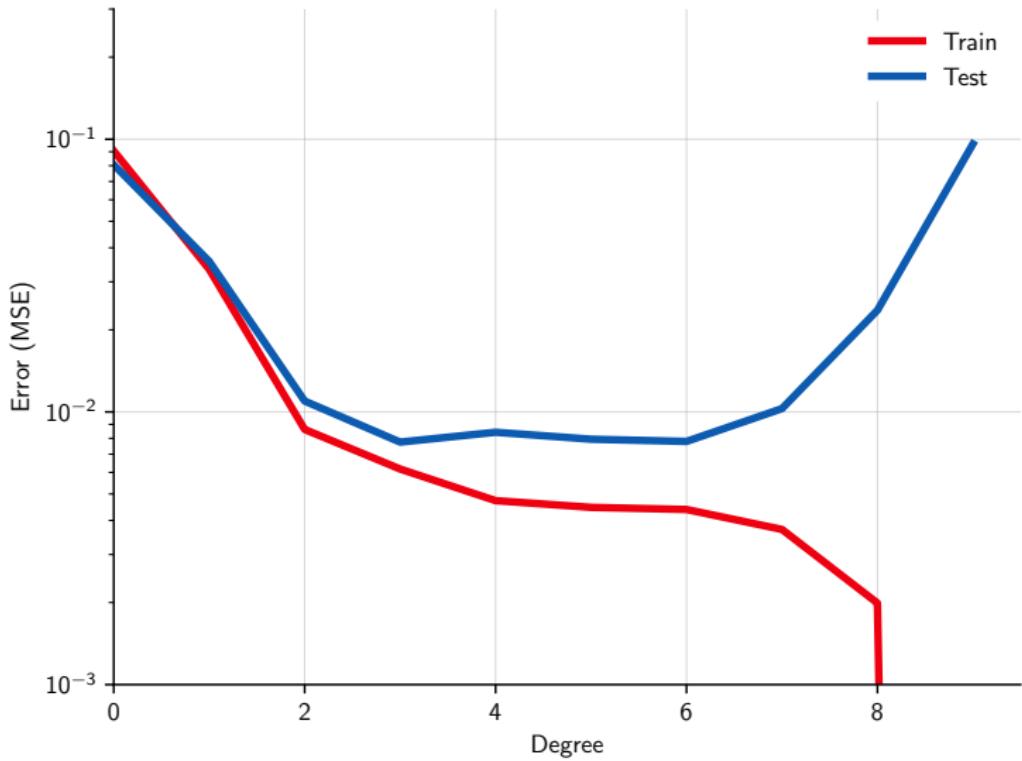


Degree $D = 8$



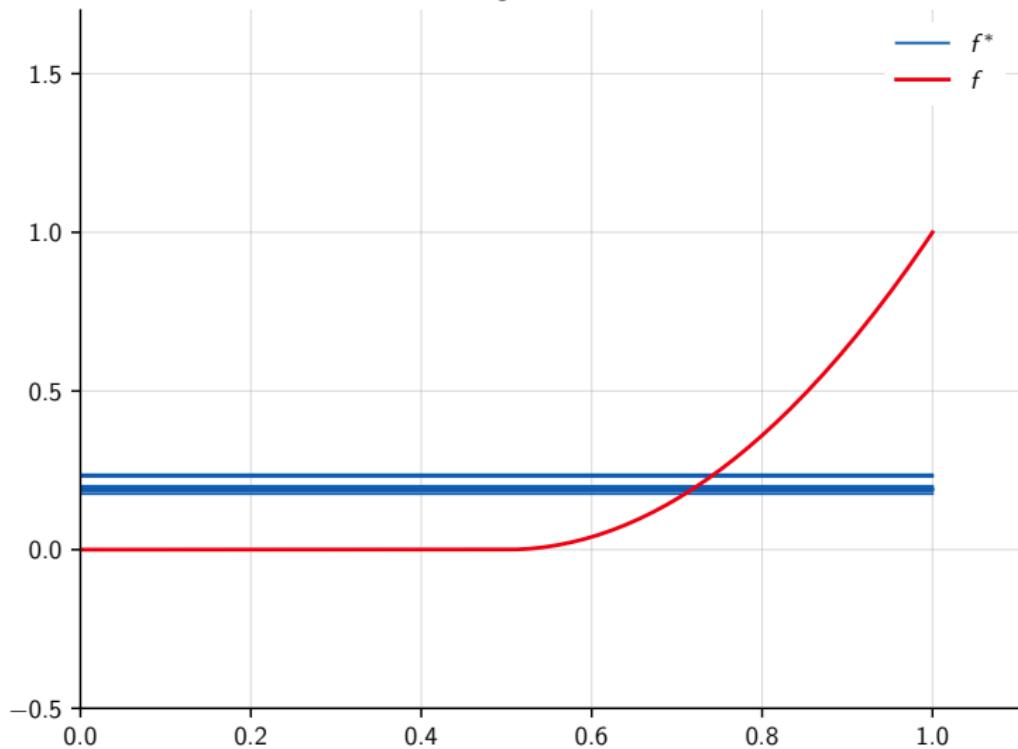
Degree $D = 9$



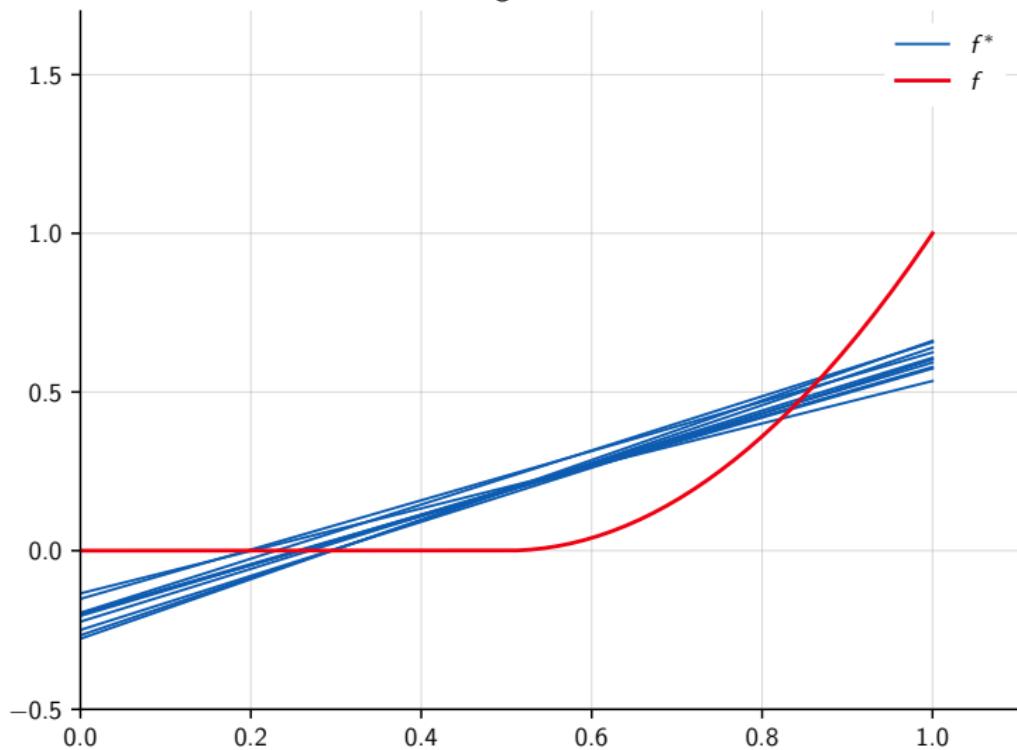


We can visualize the influence of the noise by generating multiple training sets $\mathcal{D}_1, \dots, \mathcal{D}_M$ with different noise, and training one model on each.

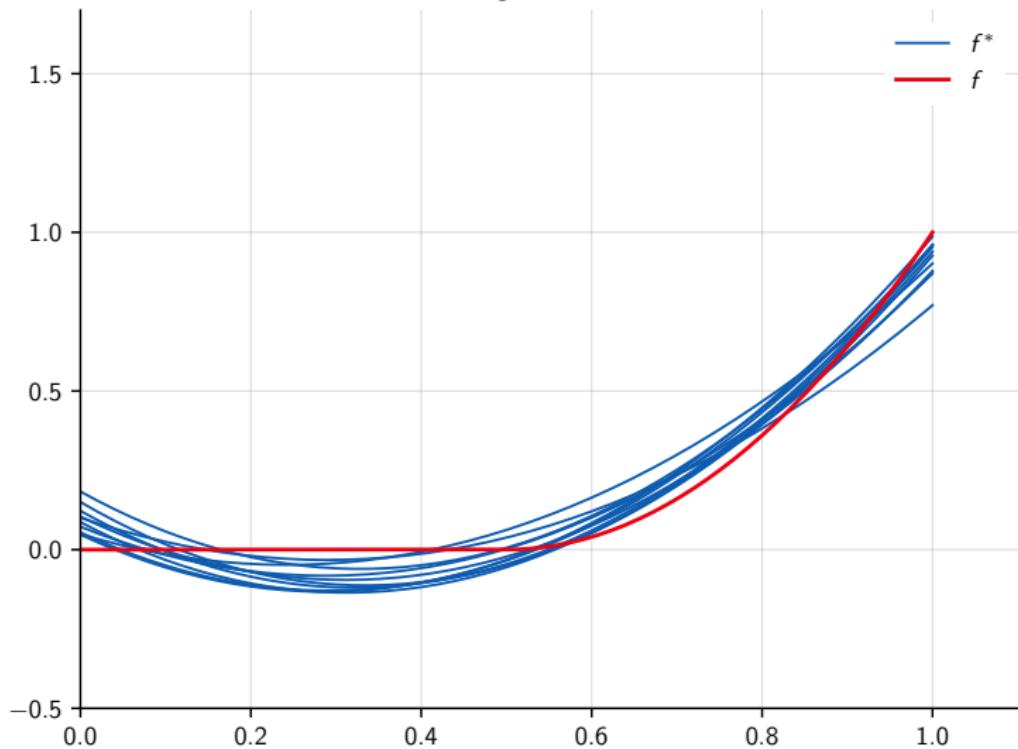
Degree $D = 0$



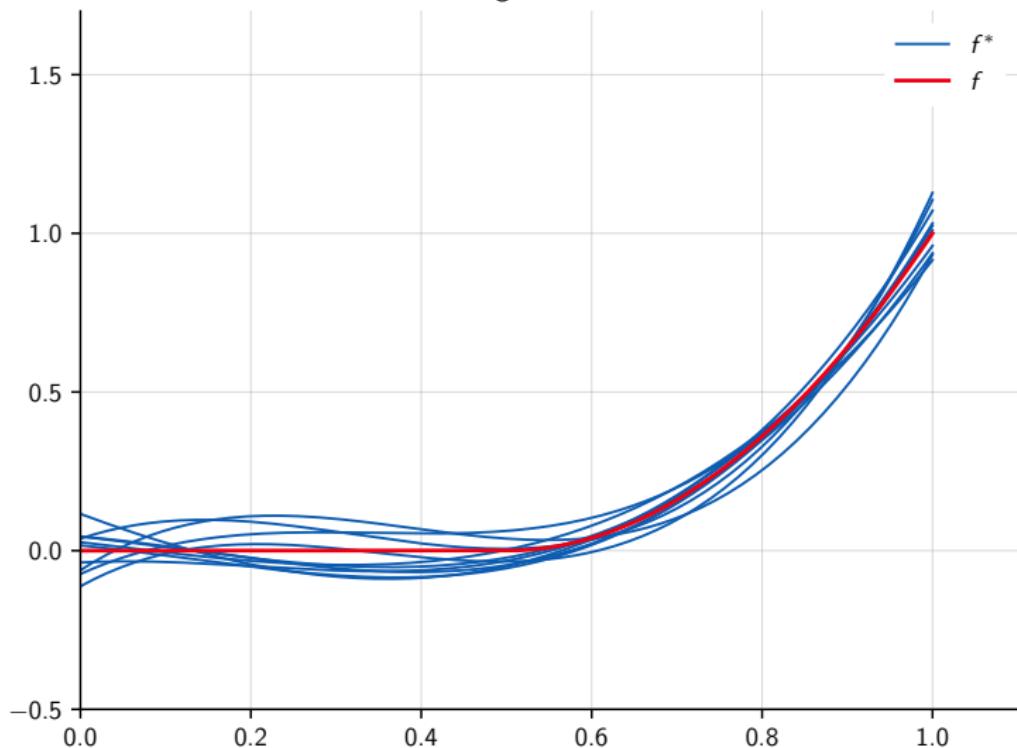
Degree $D = 1$



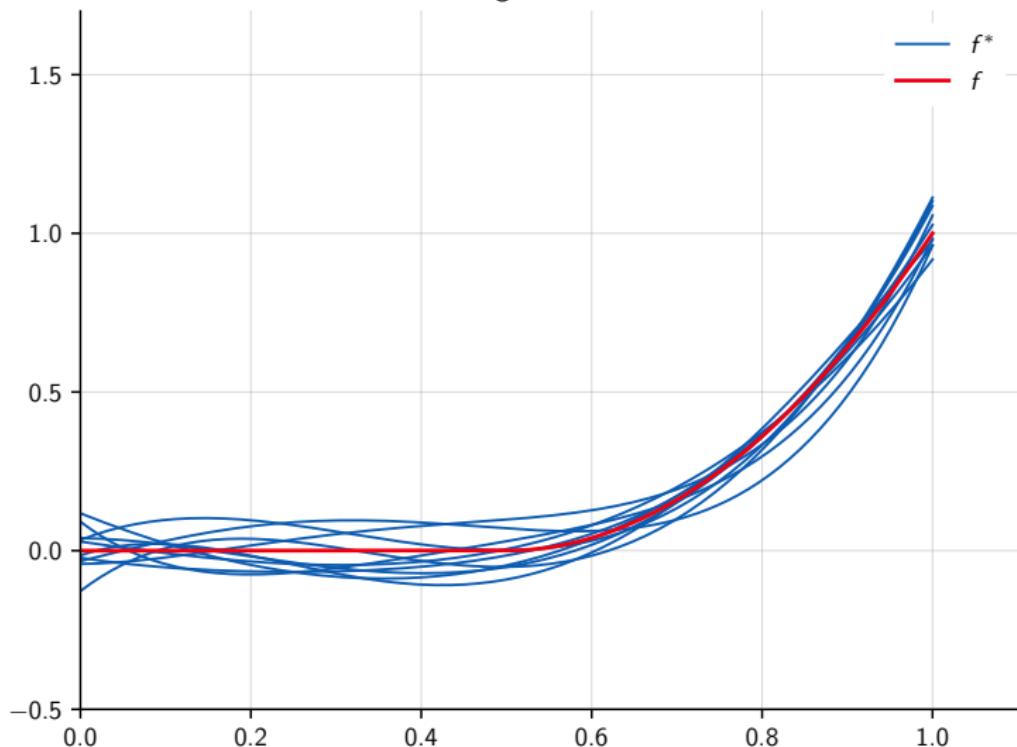
Degree $D = 2$



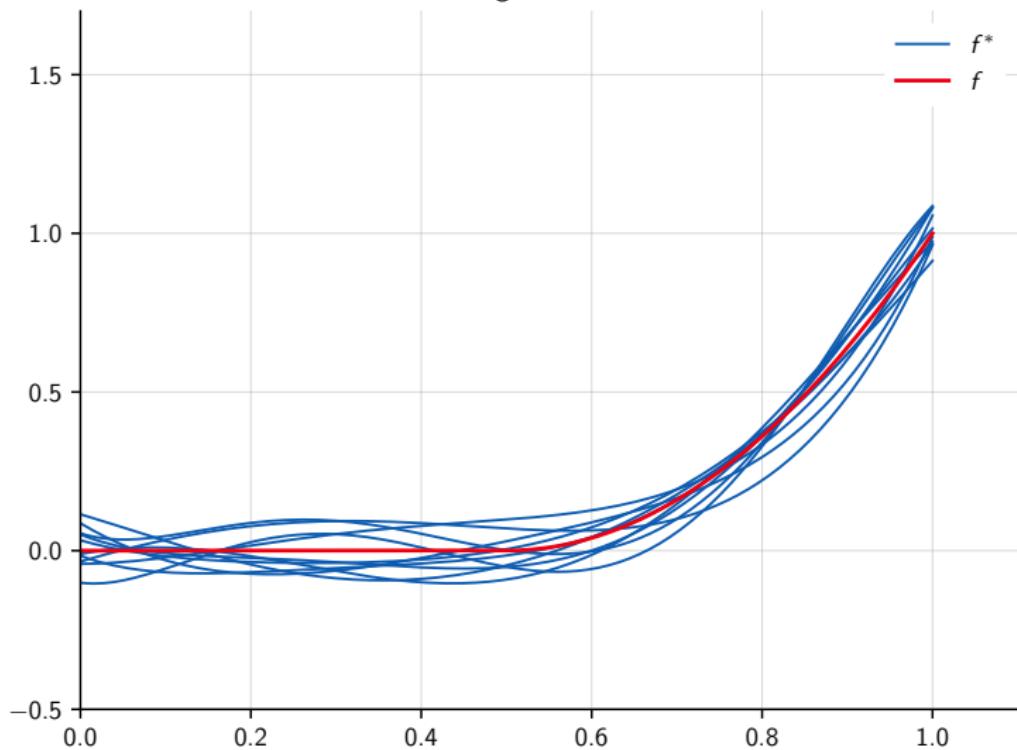
Degree $D = 3$



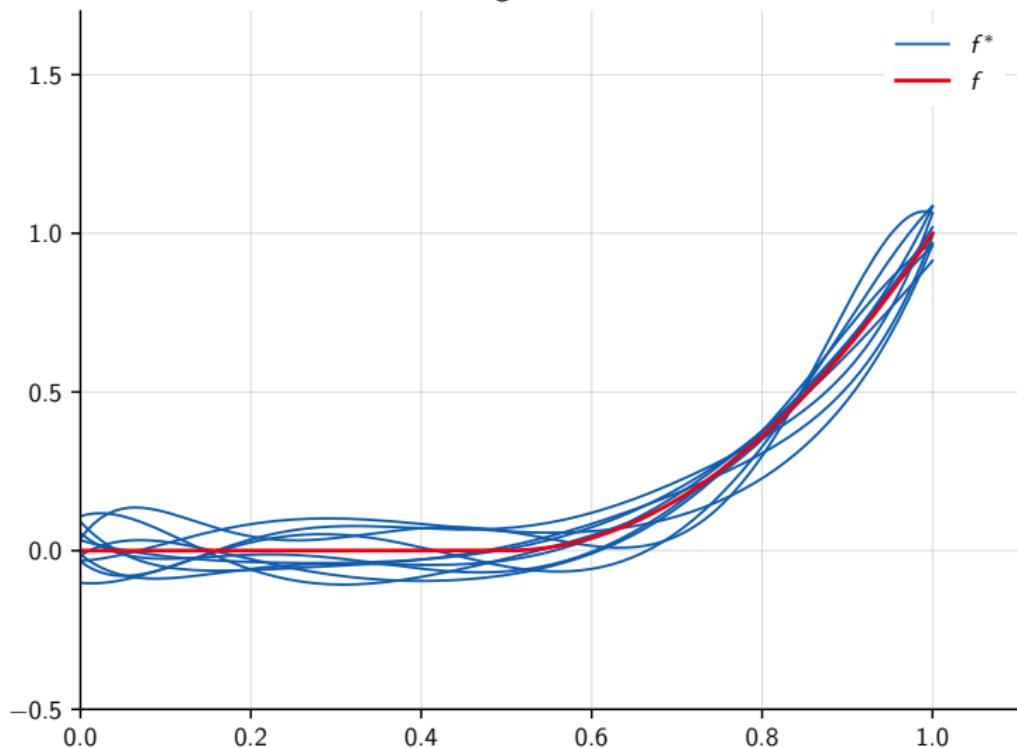
Degree $D = 4$



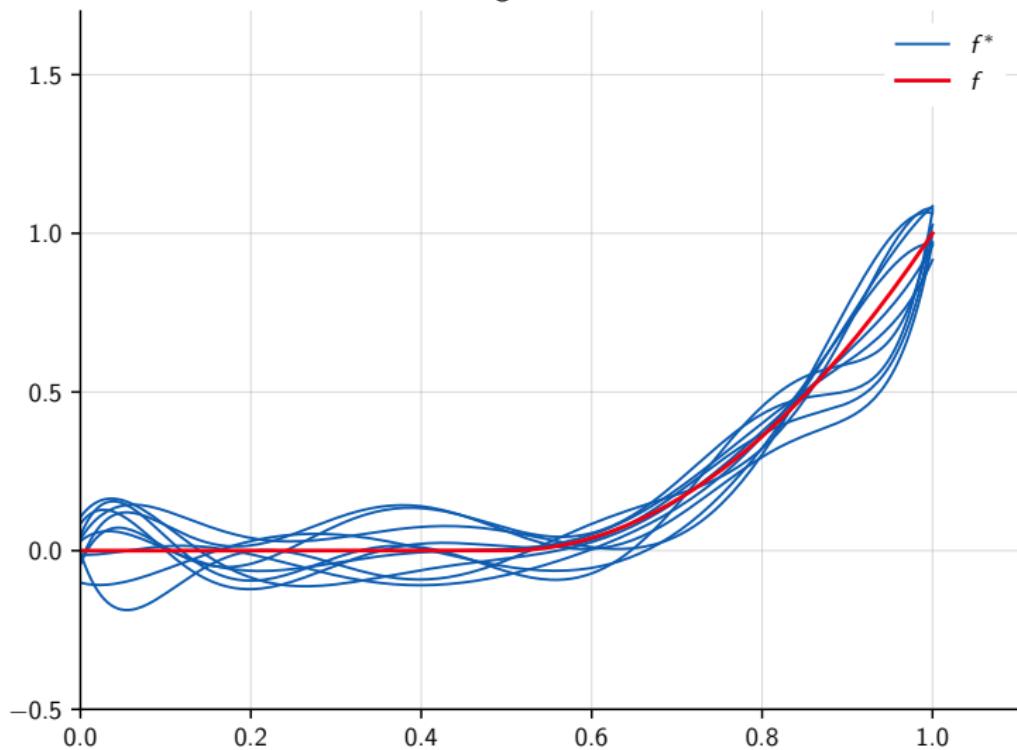
Degree $D = 5$



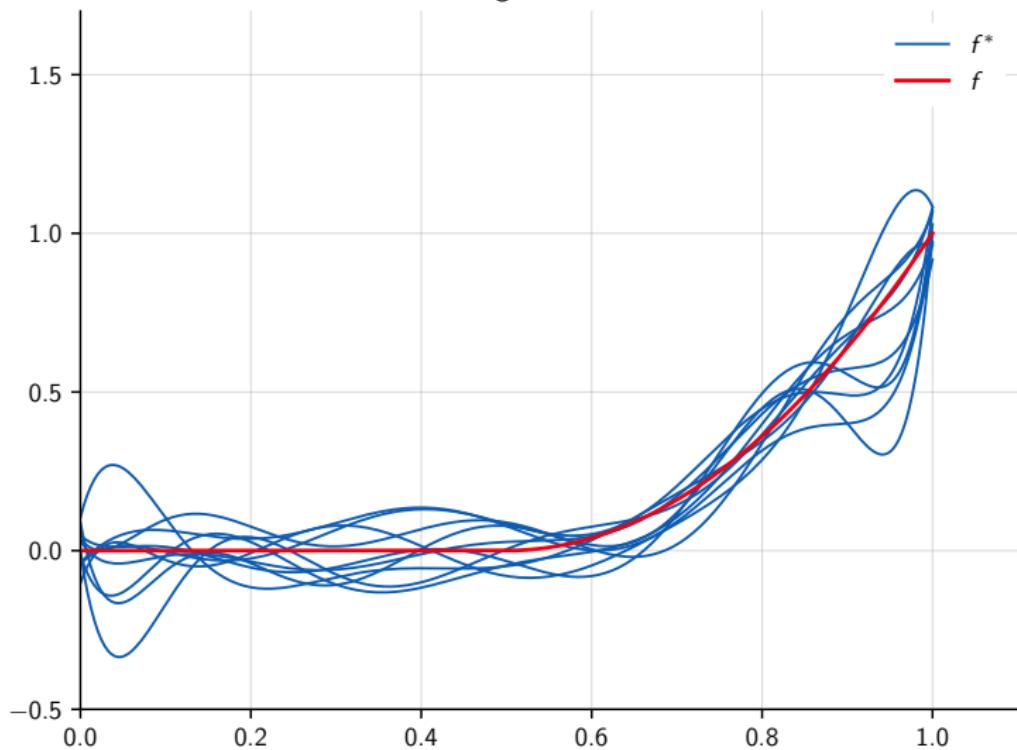
Degree $D = 6$



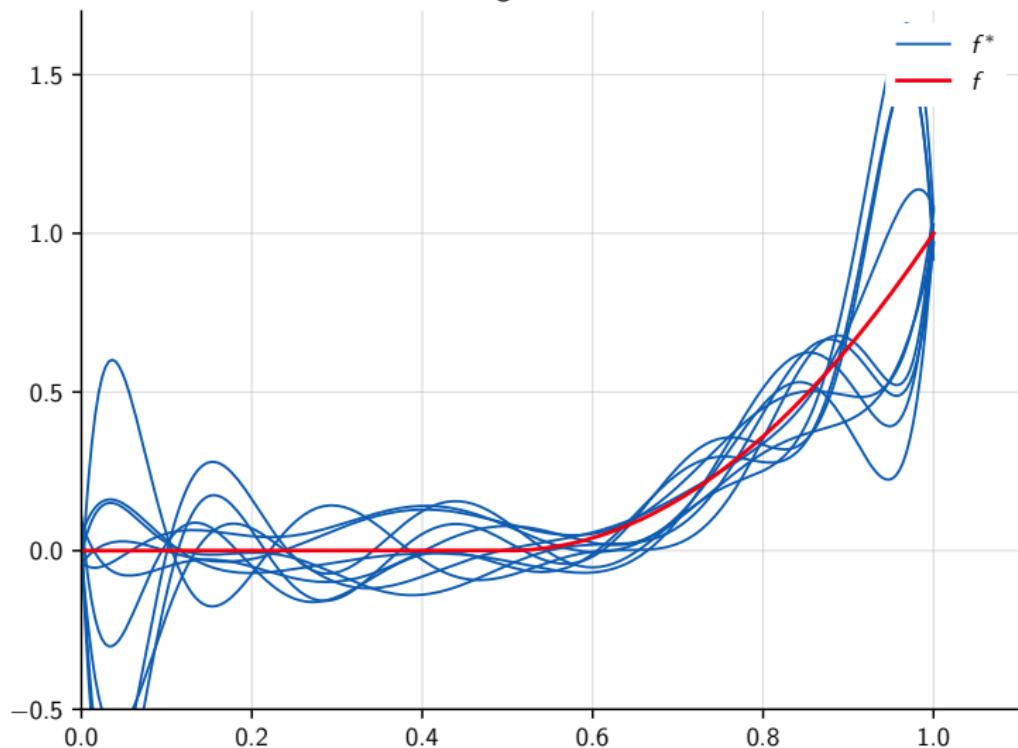
Degree $D = 7$



Degree $D = 8$



Degree $D = 9$



We can reformulate this control of the degree with a penalty

$$\mathcal{L}(\alpha) = \sum_n (f(x_n; \alpha) - y_n)^2 + \sum_d I_d(\alpha_d)$$

where

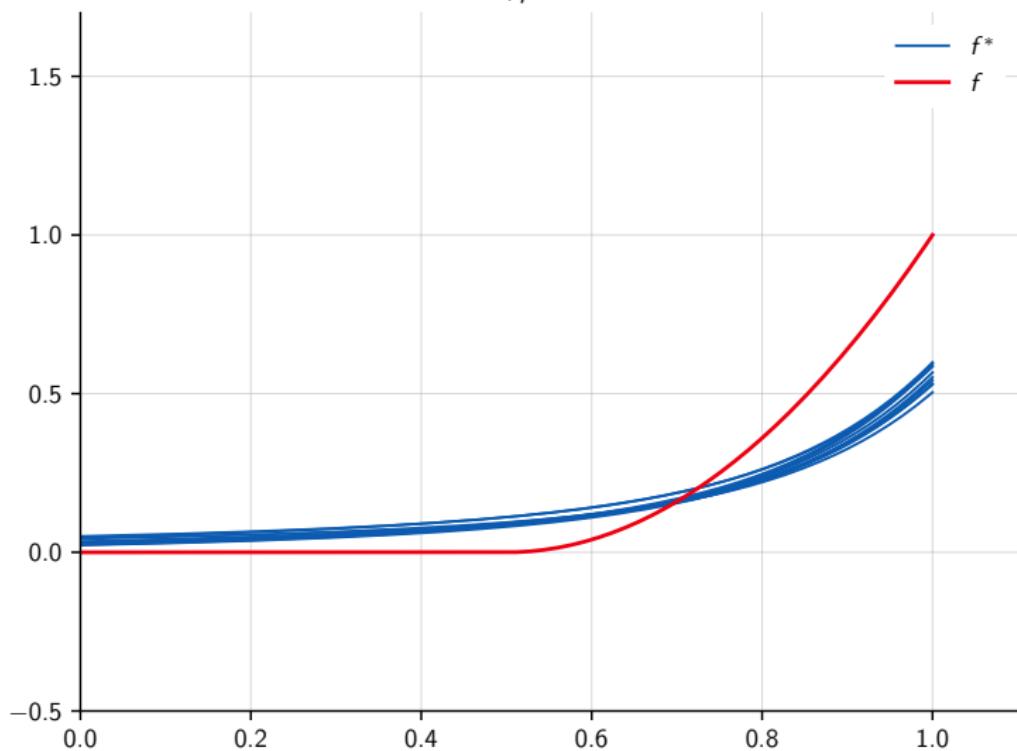
$$I_d(\alpha) = \begin{cases} 0 & \text{if } d \leq D \text{ or } \alpha = 0 \\ +\infty & \text{otherwise.} \end{cases}$$

Such a penalty kills any term of degree $> D$.

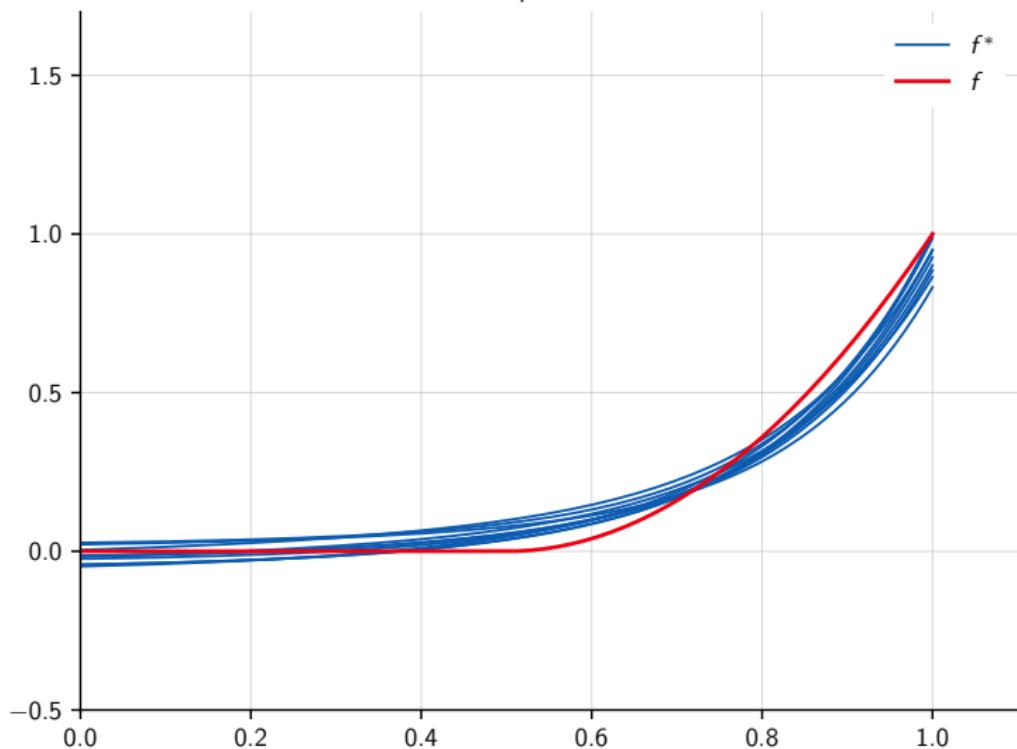
This suggests more subtle variants. For instance, to keep all this quadratic

$$\mathcal{L}(\alpha) = \sum_n (f(x_n; \alpha) - y_n)^2 + \rho \sum_d \alpha_d^2.$$

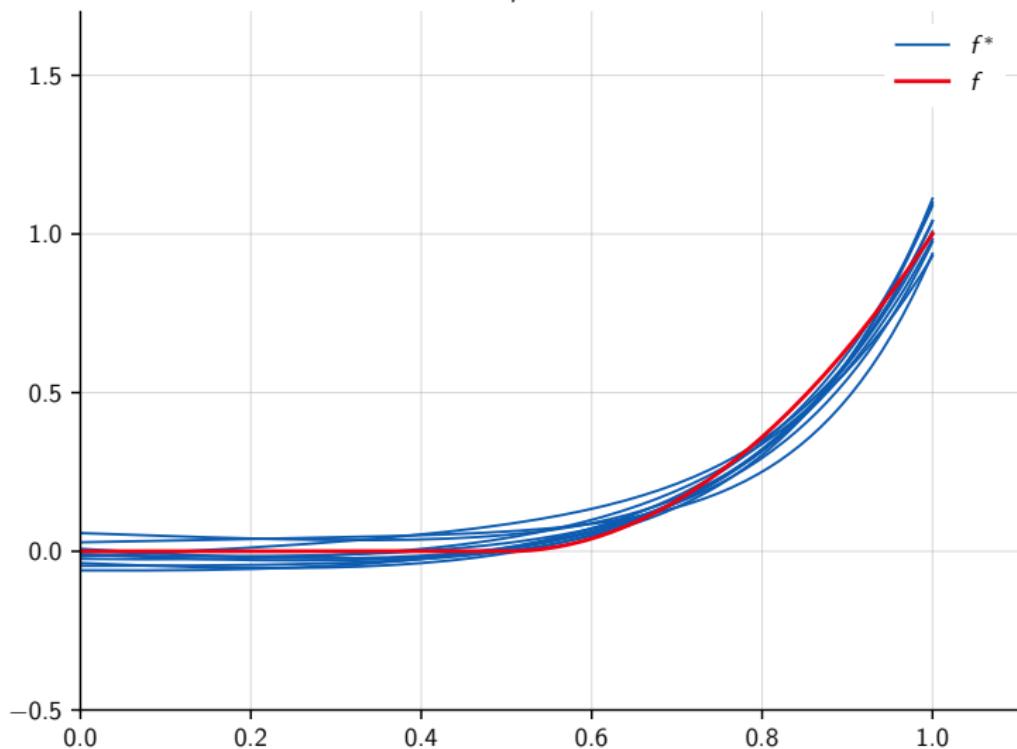
$$D = 9, \rho = 1 \times 10^1$$



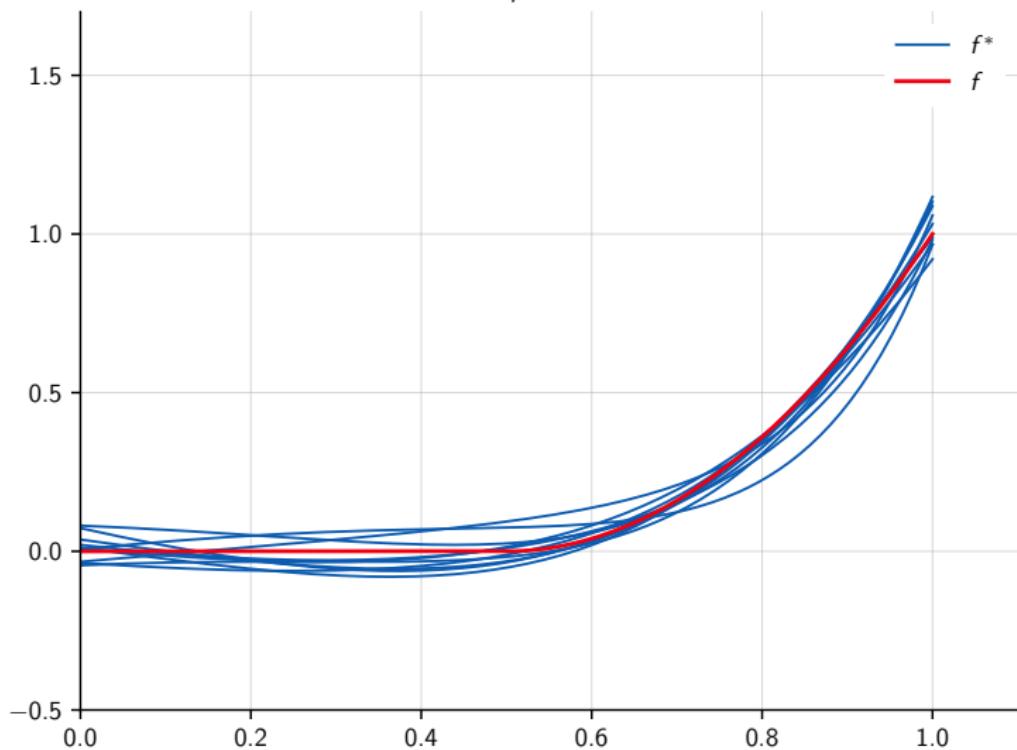
$$D = 9, \rho = 1 \times 10^0$$



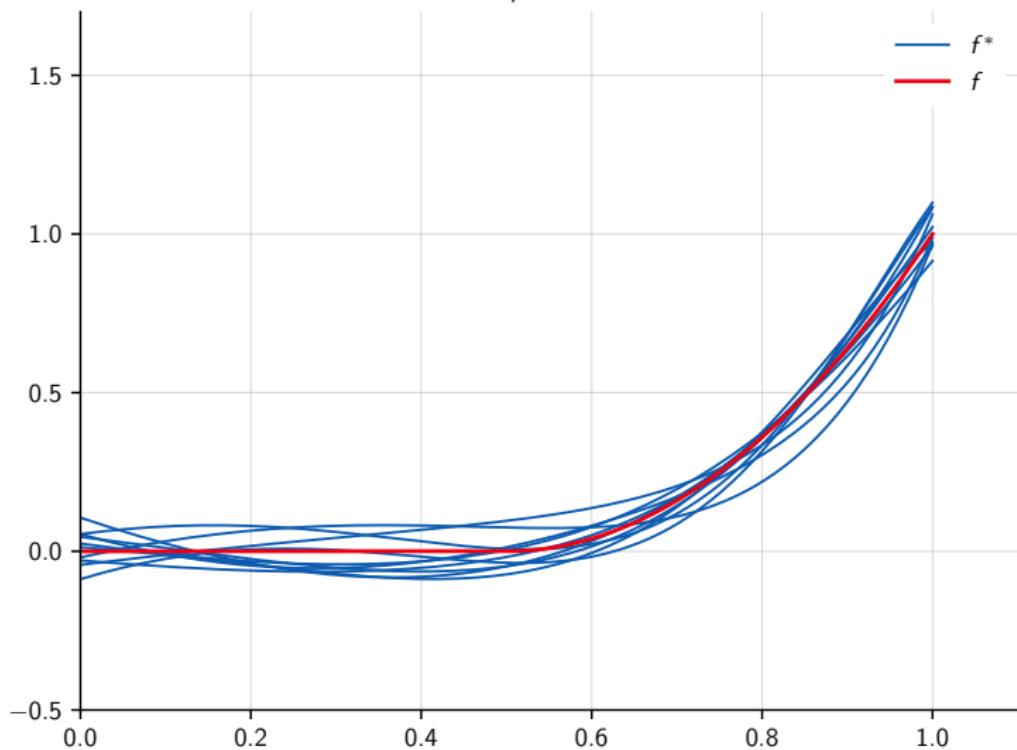
$$D = 9, \rho = 1 \times 10^{-1}$$



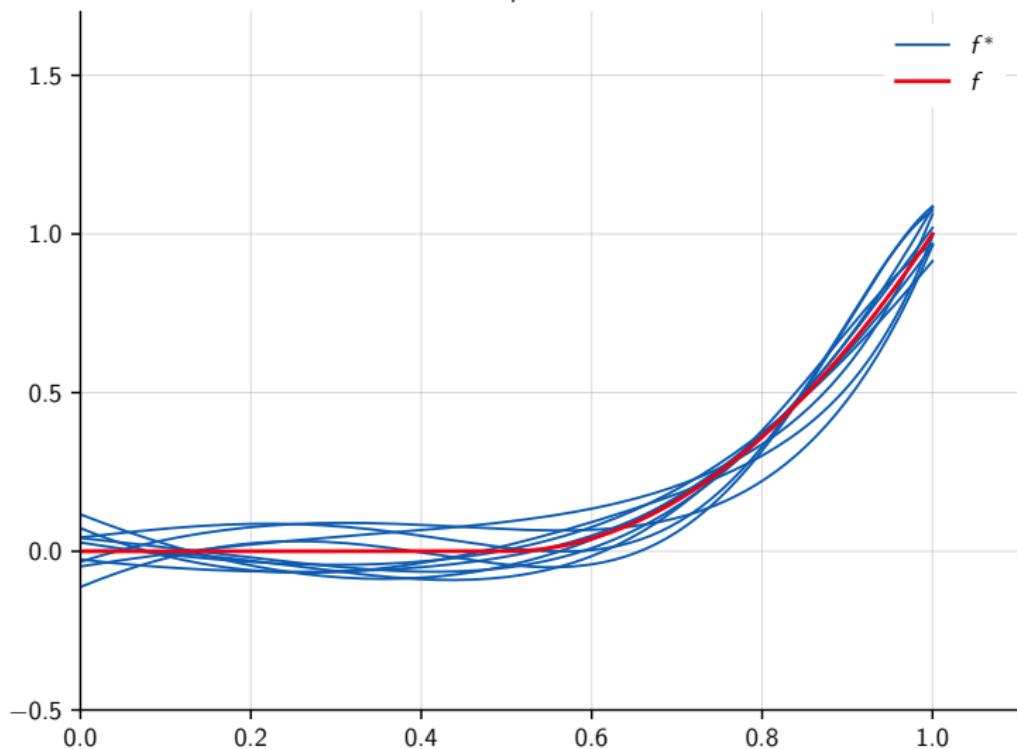
$$D = 9, \rho = 1 \times 10^{-2}$$



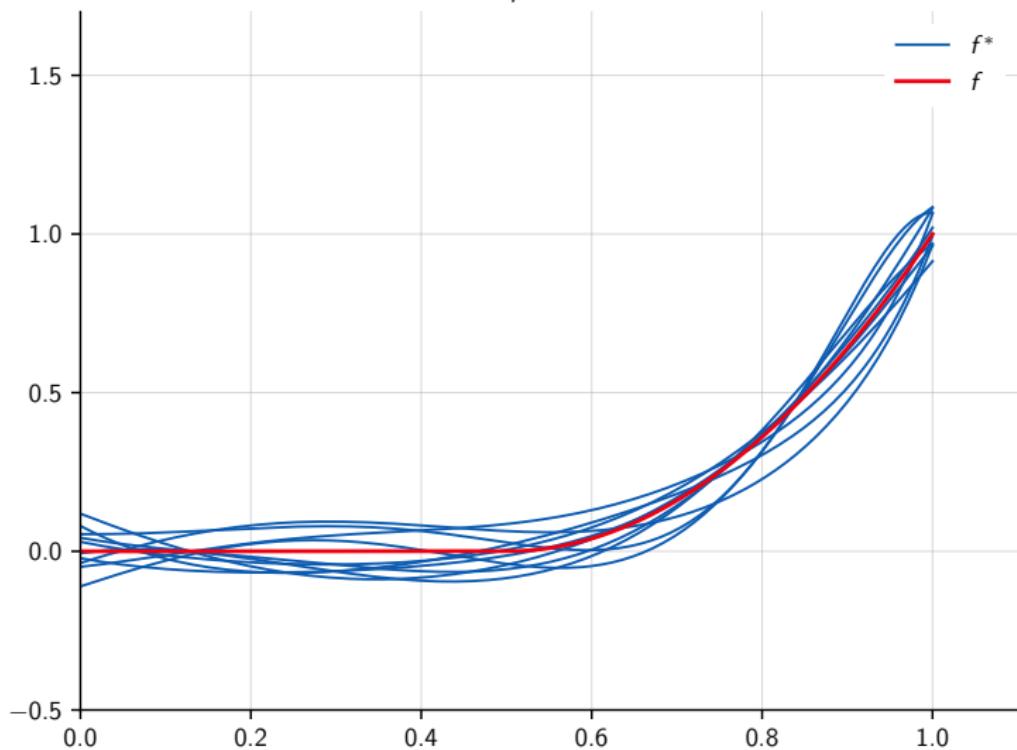
$$D = 9, \rho = 1 \times 10^{-3}$$



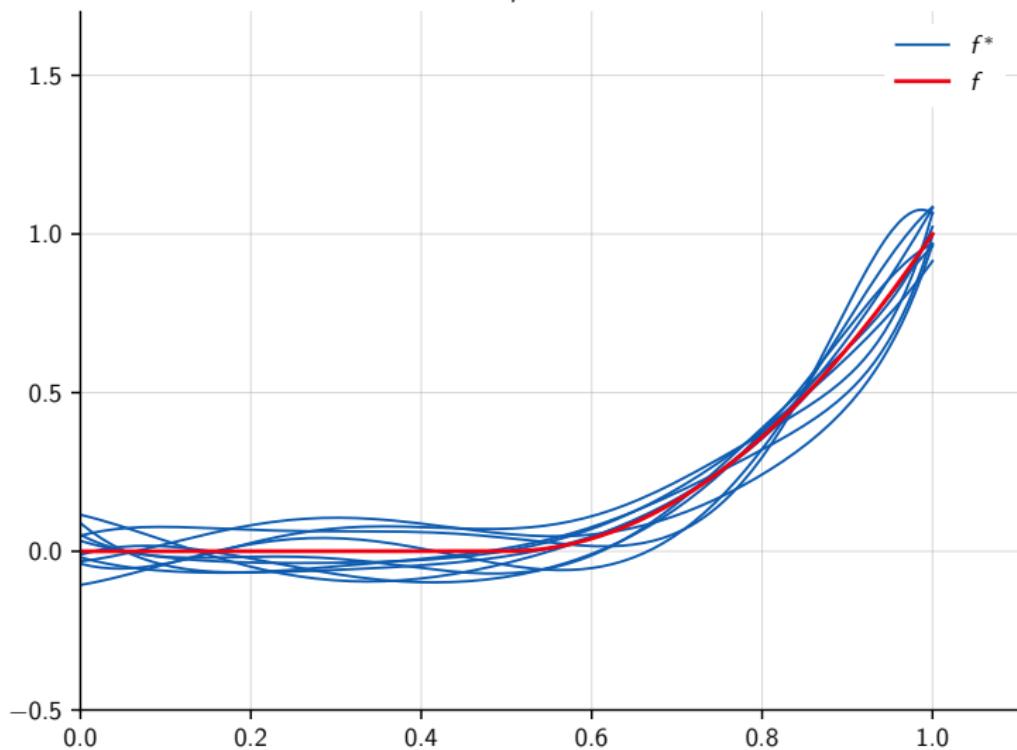
$$D = 9, \rho = 1 \times 10^{-4}$$



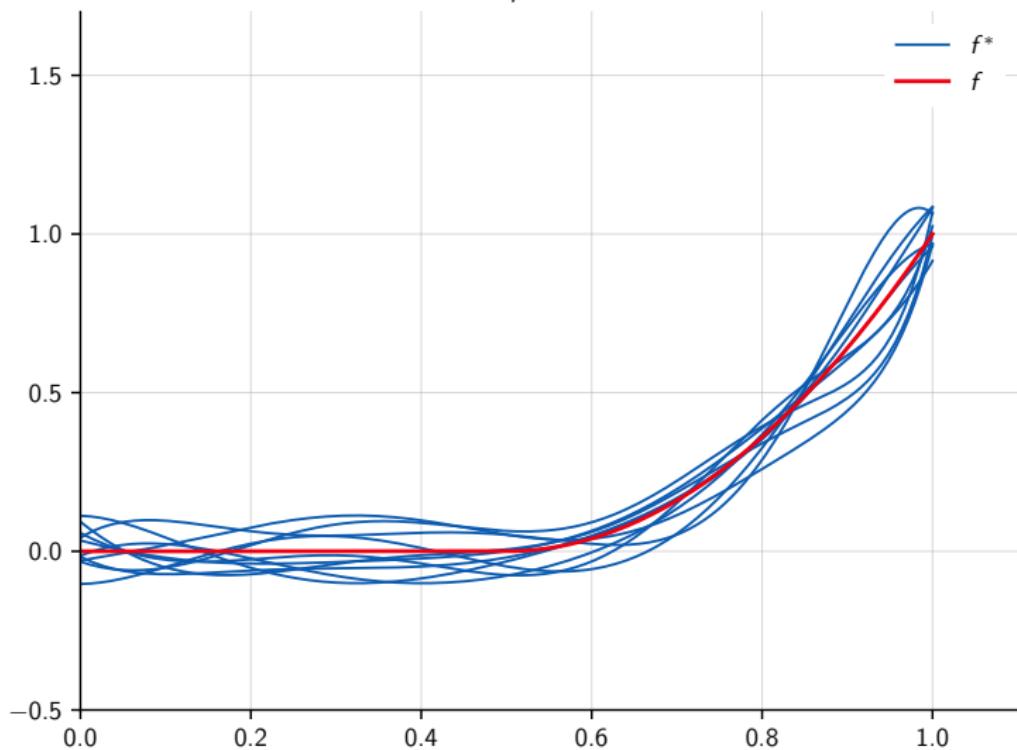
$$D = 9, \rho = 1 \times 10^{-5}$$



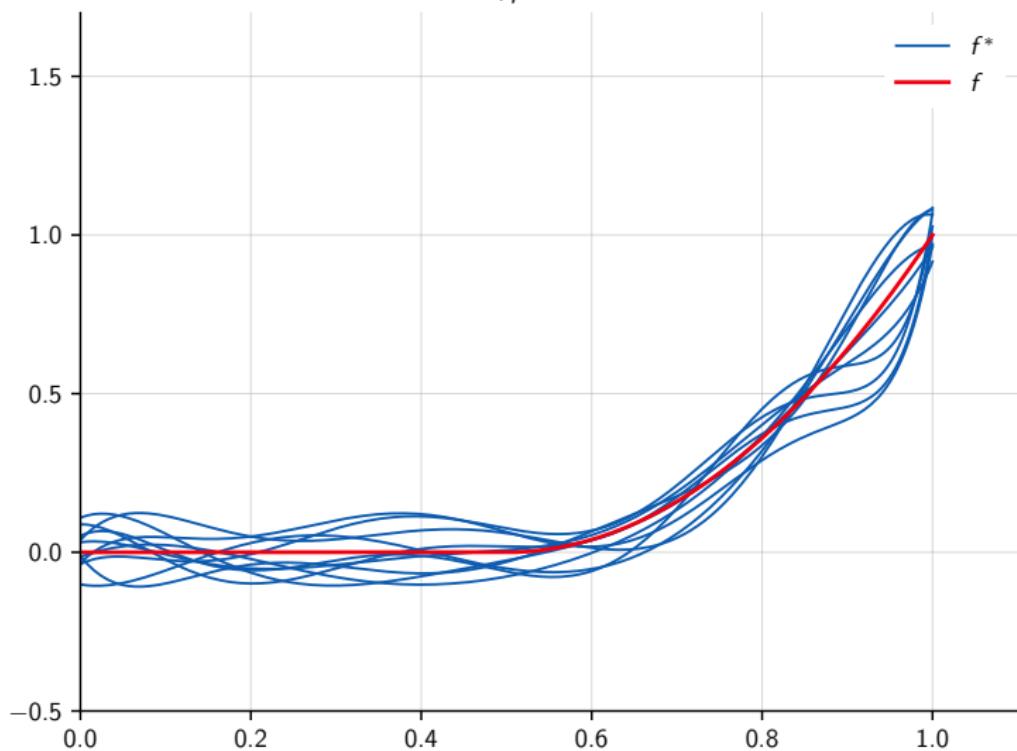
$$D = 9, \rho = 1 \times 10^{-6}$$



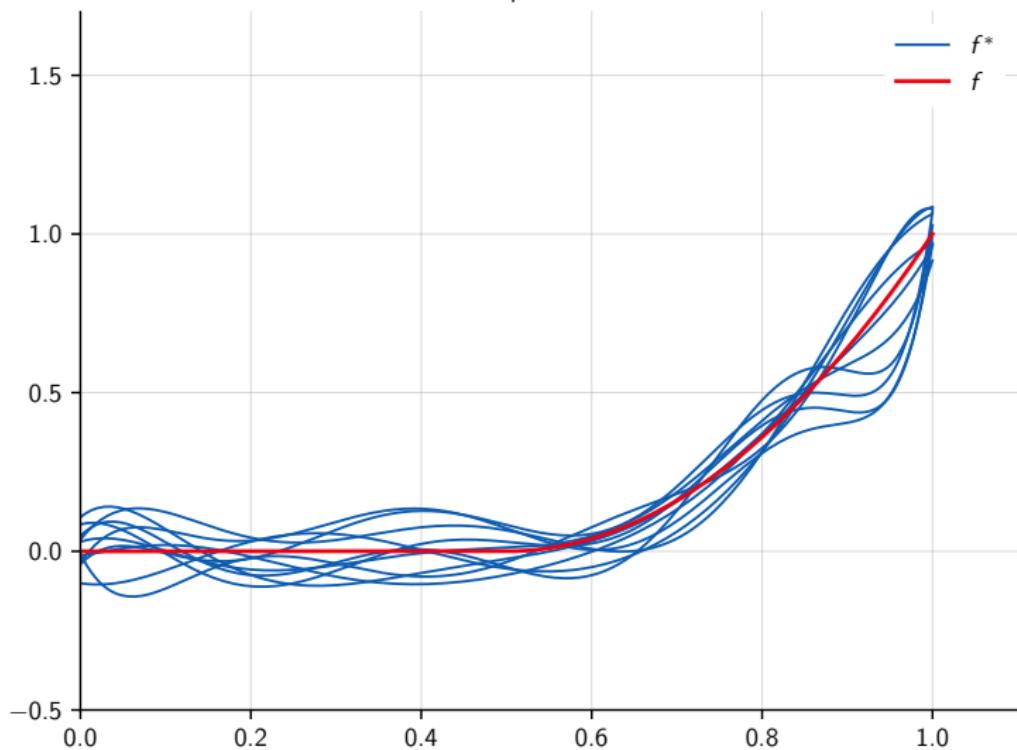
$$D = 9, \rho = 1 \times 10^{-7}$$



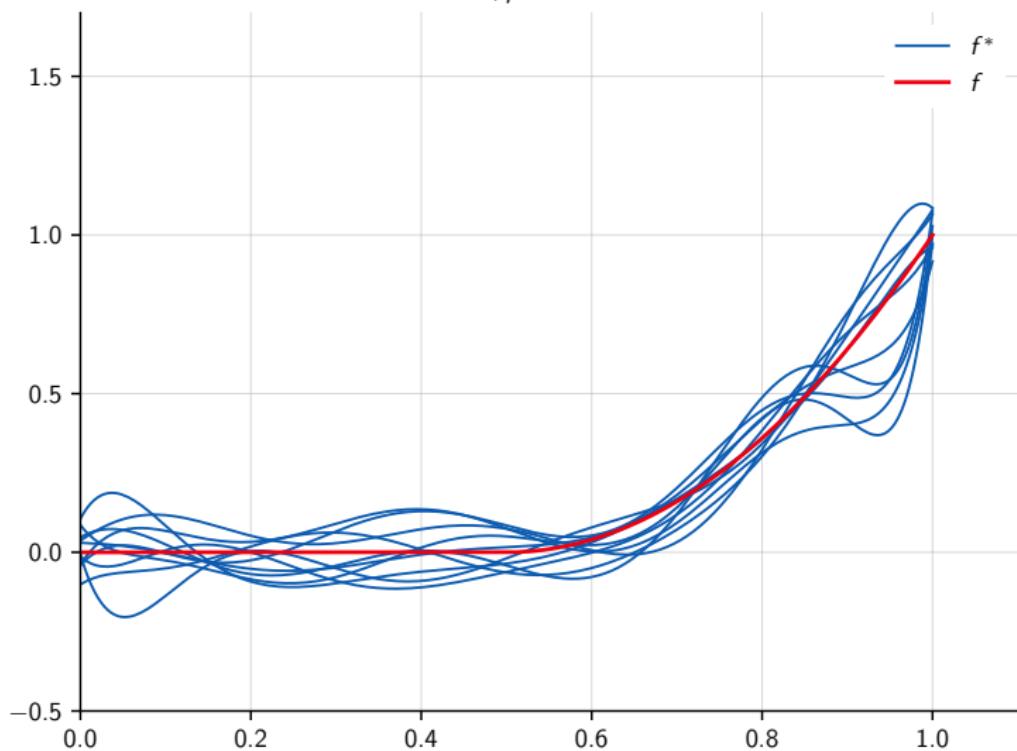
$$D = 9, \rho = 1 \times 10^{-8}$$



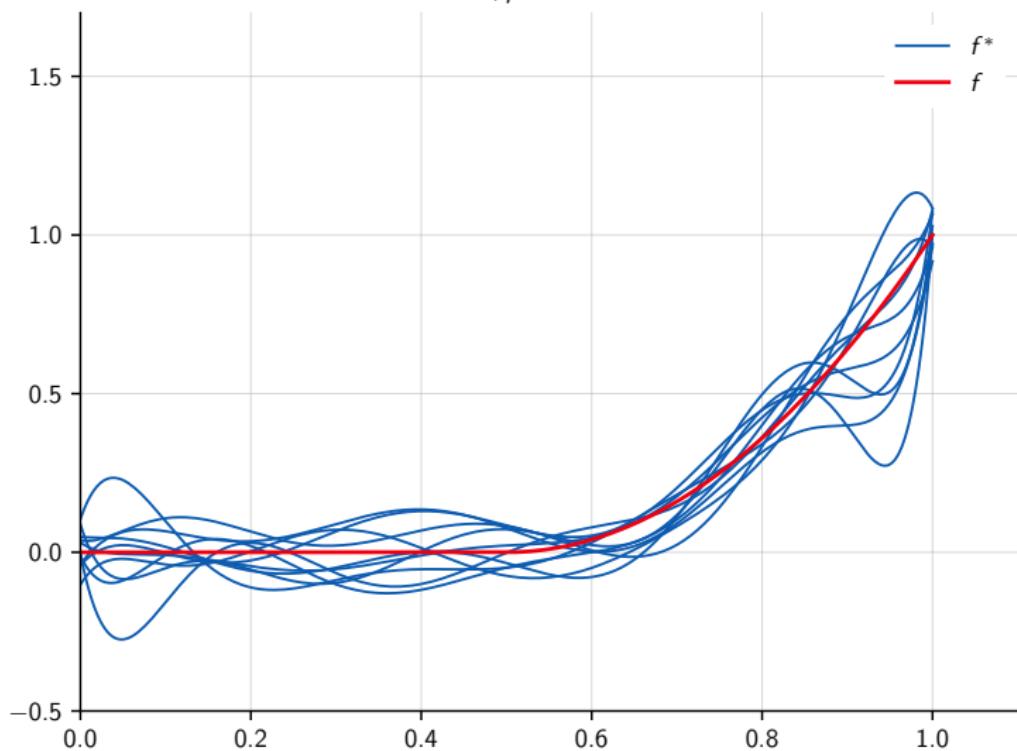
$$D = 9, \rho = 1 \times 10^{-9}$$



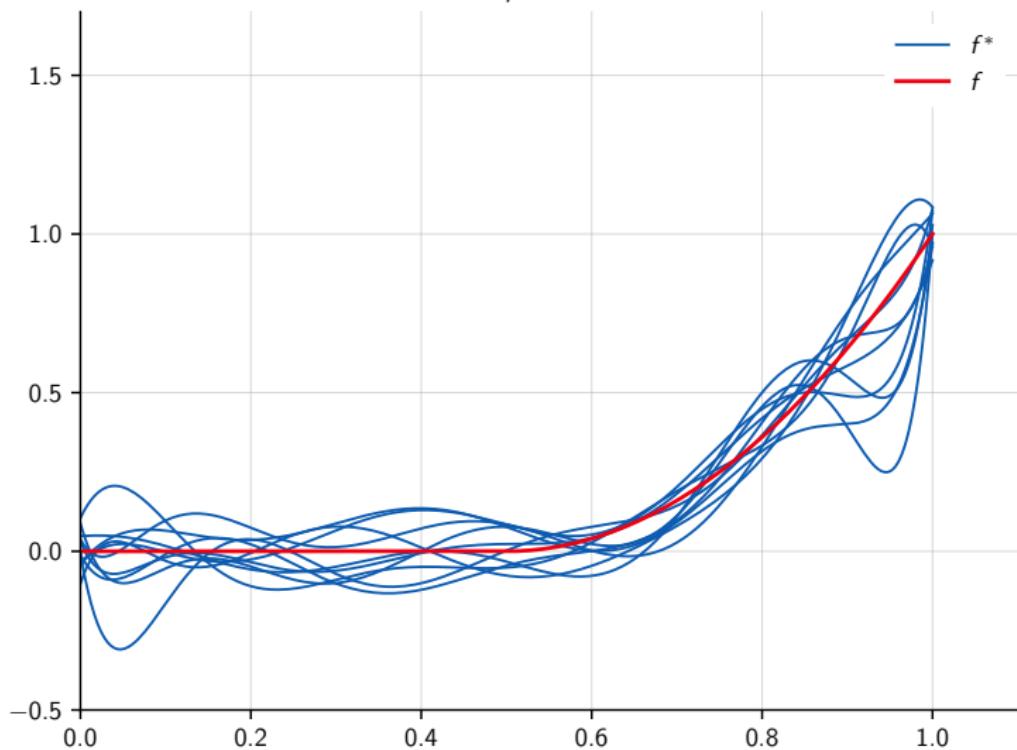
$$D = 9, \rho = 1 \times 10^{-10}$$



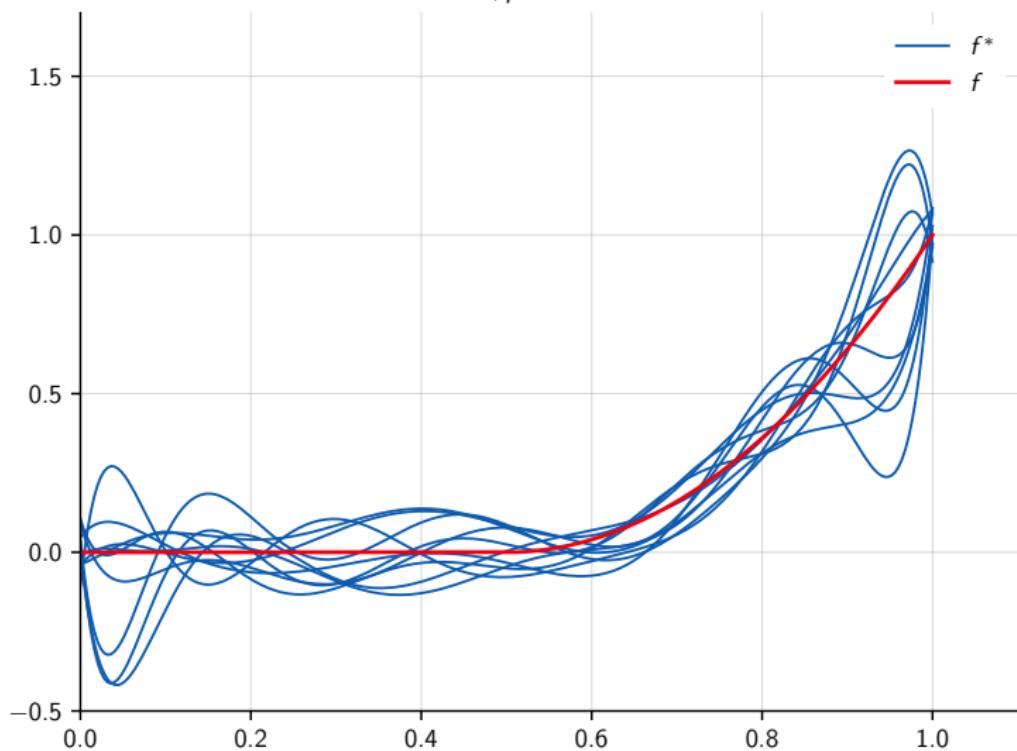
$$D = 9, \rho = 1 \times 10^{-11}$$



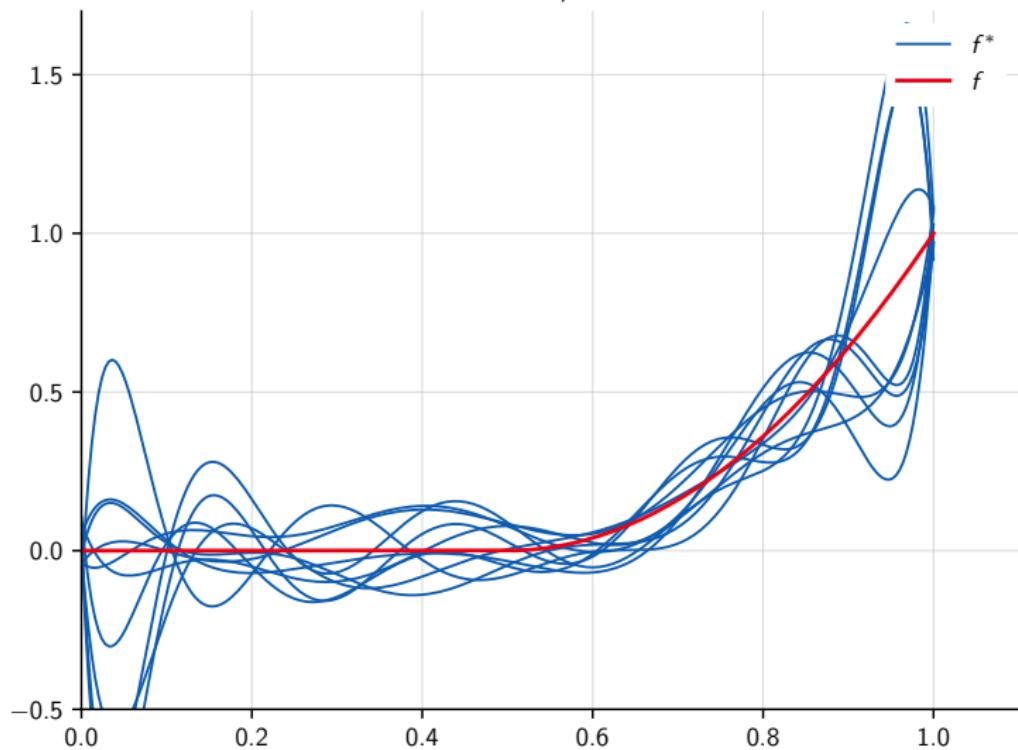
$$D = 9, \rho = 1 \times 10^{-12}$$

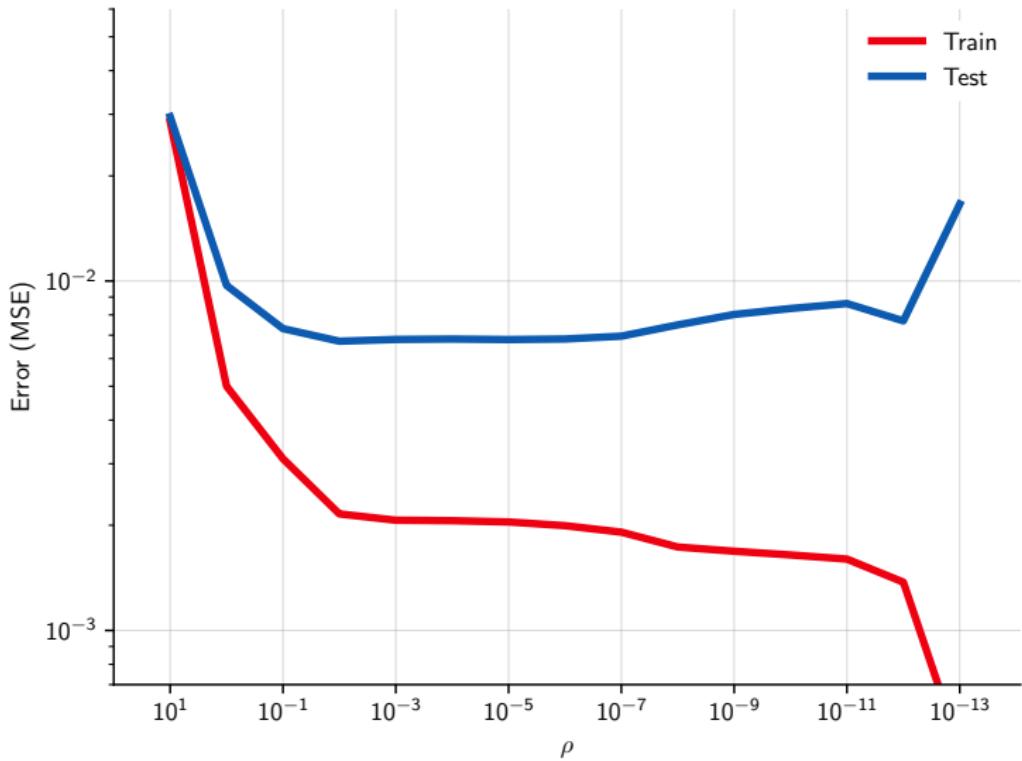


$$D = 9, \rho = 1 \times 10^{-13}$$



$$D = 9, \rho = 0$$





We define the **capacity** of a set of predictors as its ability to model an arbitrary functional. This is a vague definition, difficult to make formal.

A mathematically precise notion is the Vapnik–Chervonenkis dimension of a set of functions, which, in the Binary classification case, is the cardinality of the largest set that can be labeled arbitrarily (Vapnik, 1995).

It is a very powerful concept, but is poorly adapted to neural networks. We will not say more about it in this course.

Although the capacity is hard to define precisely, it is quite clear in practice how to modulate it for a given class of models.

In particular one can control over-fitting either by

- Reducing the space \mathcal{F} (less functionals, constrained or degraded optimization), or
- Making the choice of f^* less dependent on data (penalty on coefficients, margin maximization, ensemble methods).

Deep learning

2.3. Bias-variance dilemma

François Fleuret
<https://fleuret.org/dlc/>

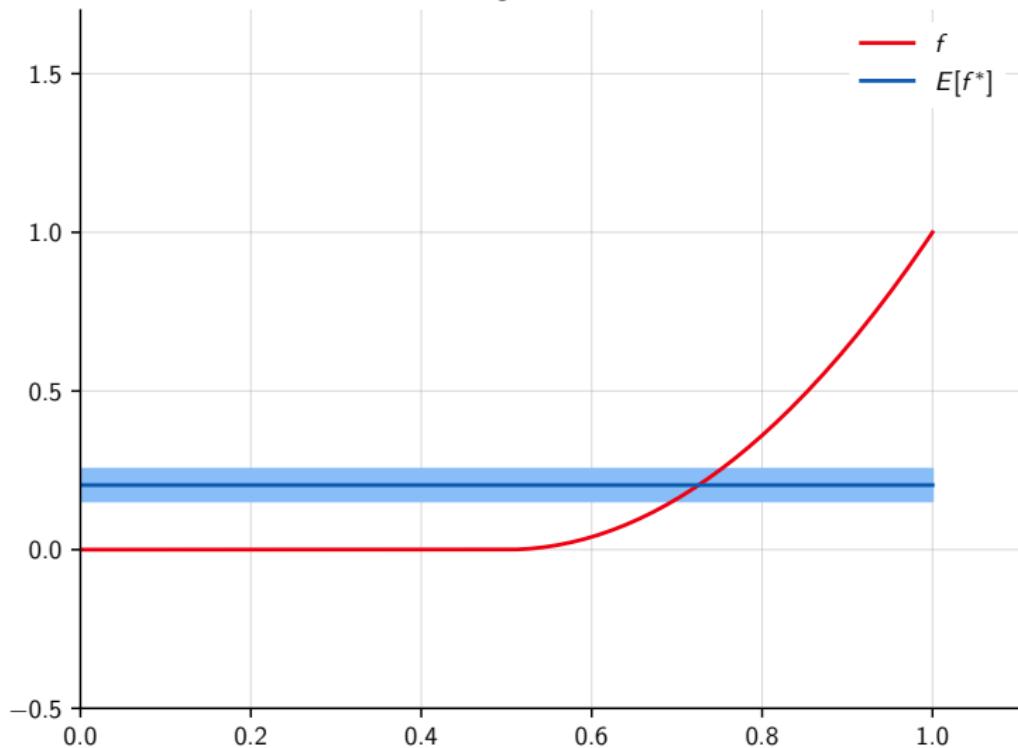


UNIVERSITÉ
DE GENÈVE

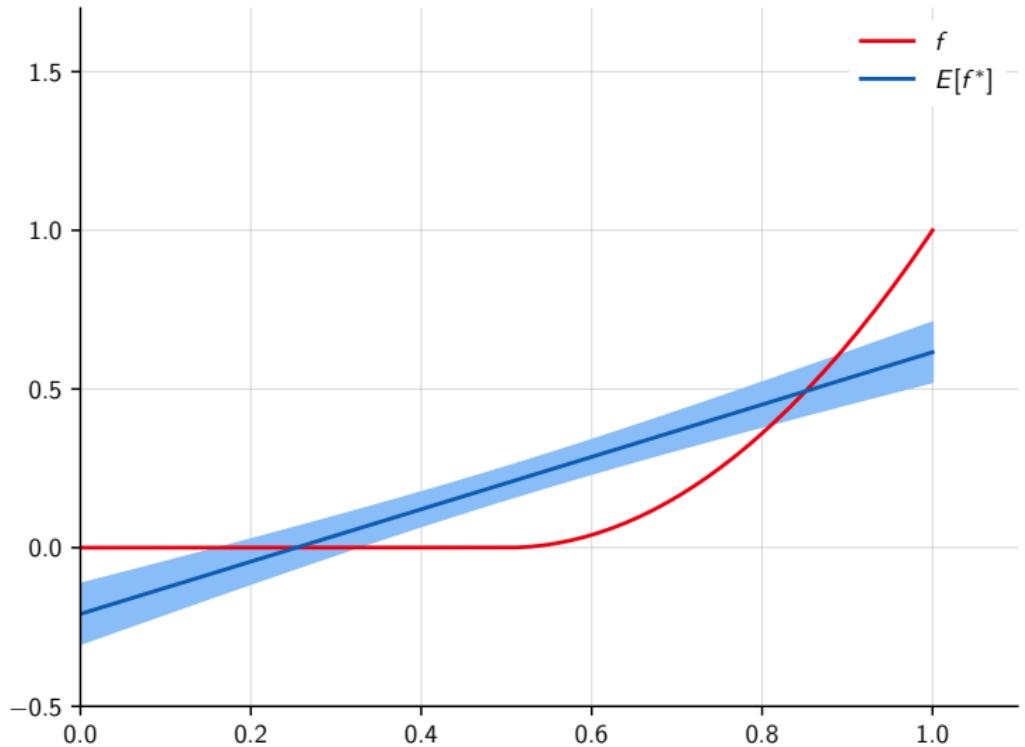
We can visualize over-fitting for our polynomial regression by generating multiple training sets $\mathcal{D}_1, \dots, \mathcal{D}_M$, training as many models f_1, \dots, f_M , and computing empirically the mean and standard deviation of the prediction at every point.

As we will see, when the capacity increases, or the regularization decreases, the mean of the predicted value gets right on target, but the prediction varies more across runs.

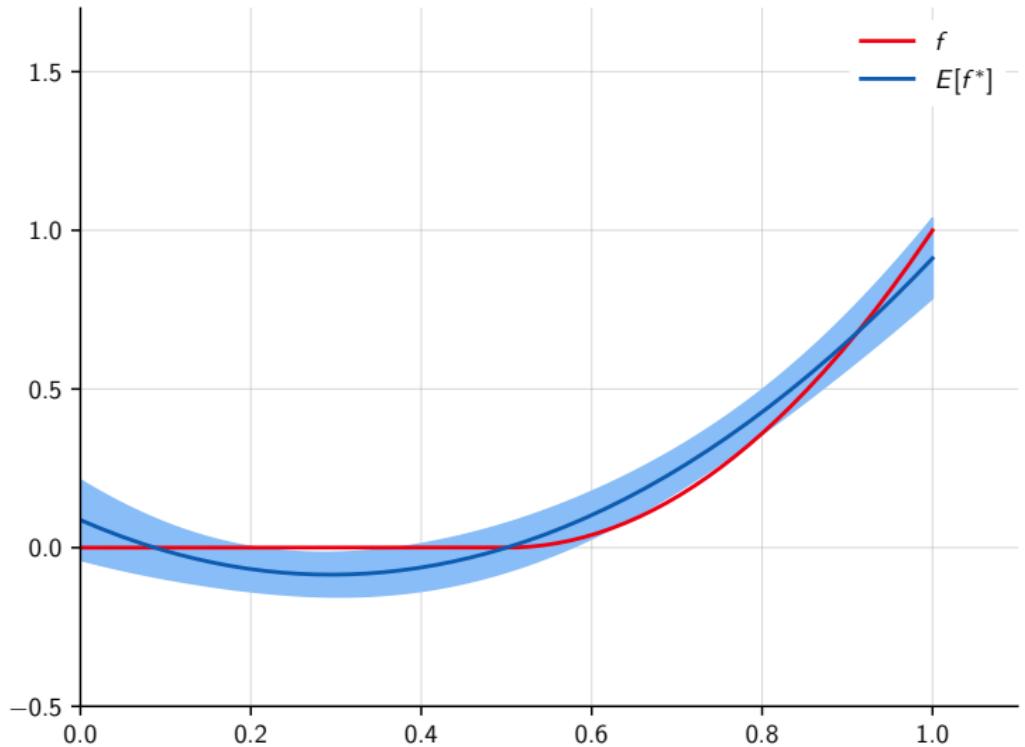
Degree $D = 0$



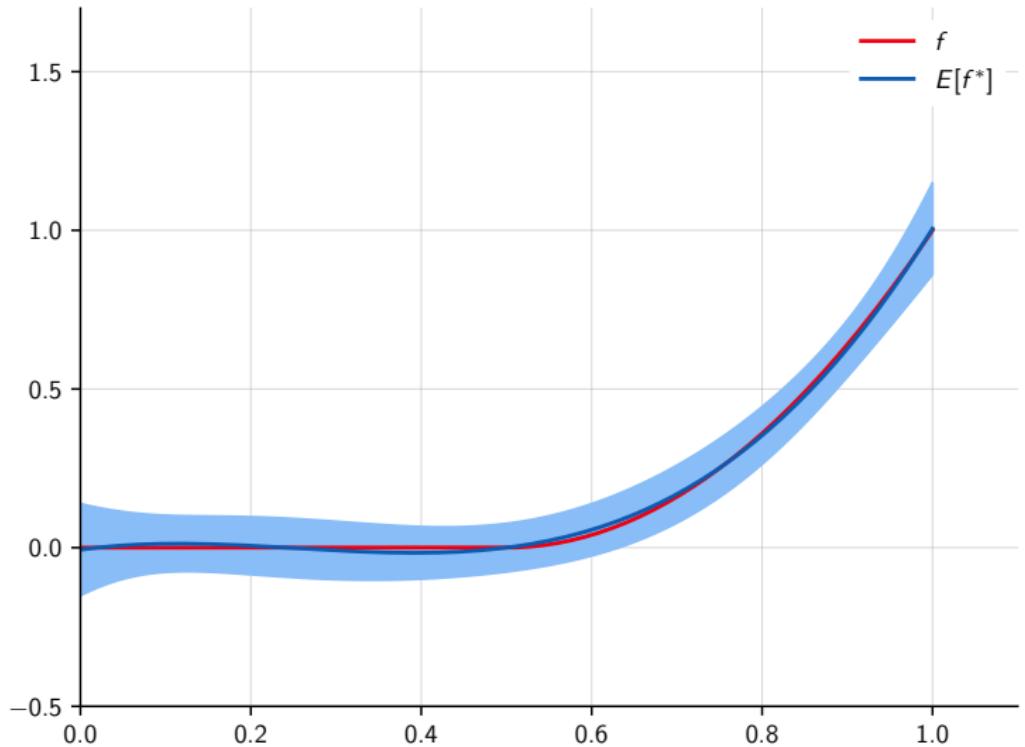
Degree $D = 1$



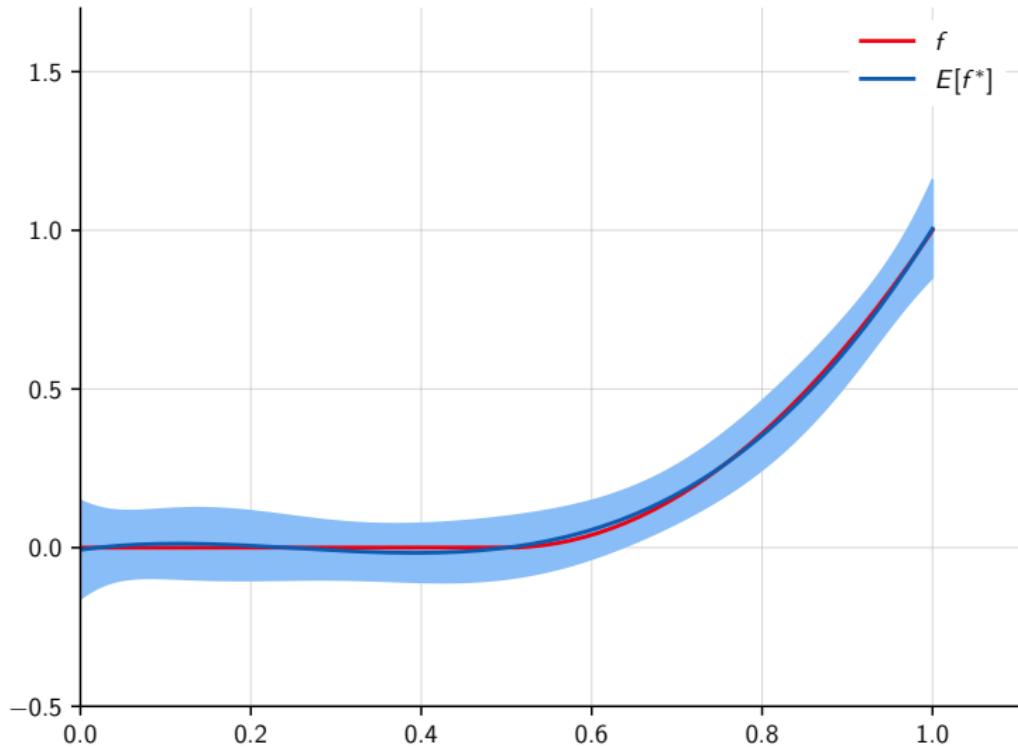
Degree $D = 2$



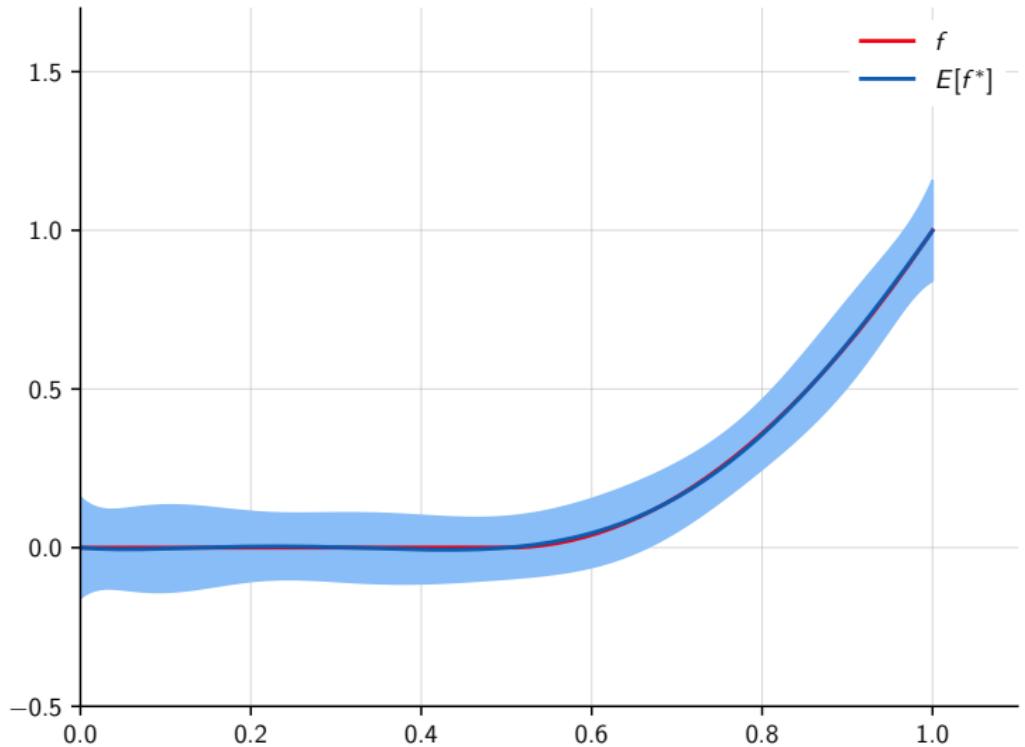
Degree $D = 3$



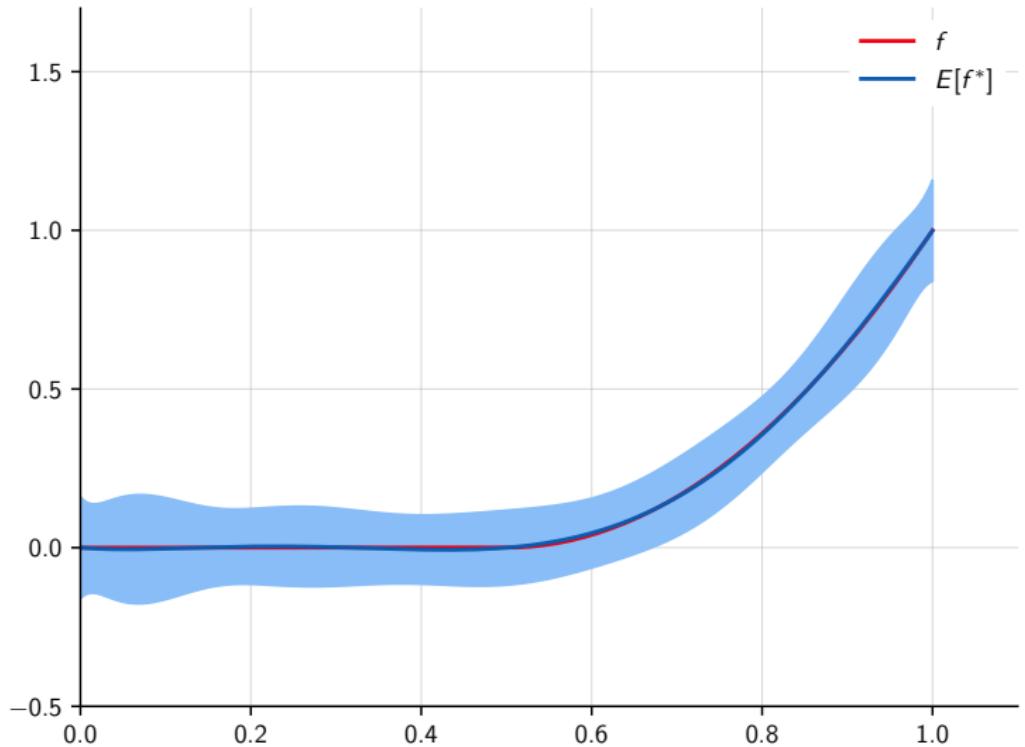
Degree $D = 4$



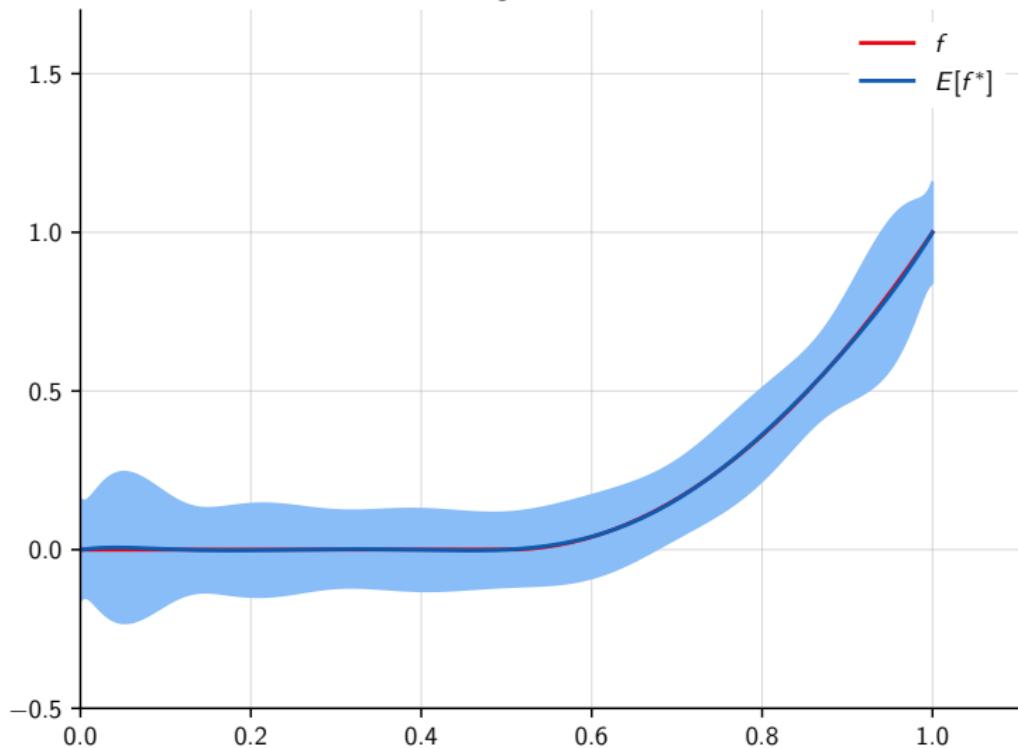
Degree $D = 5$



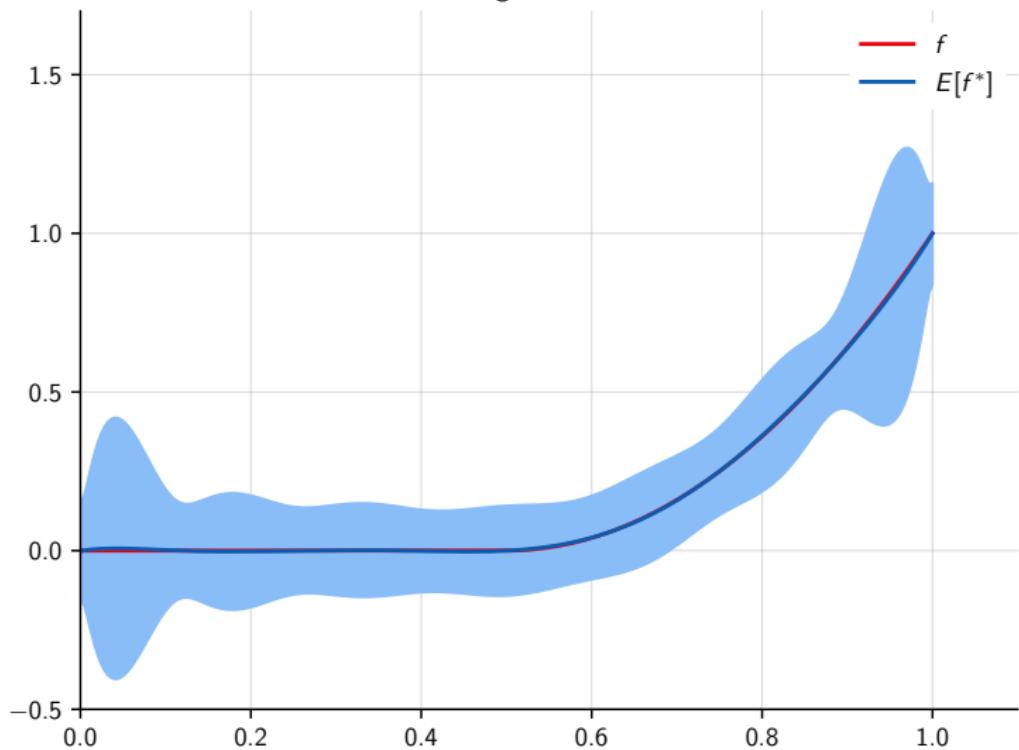
Degree $D = 6$



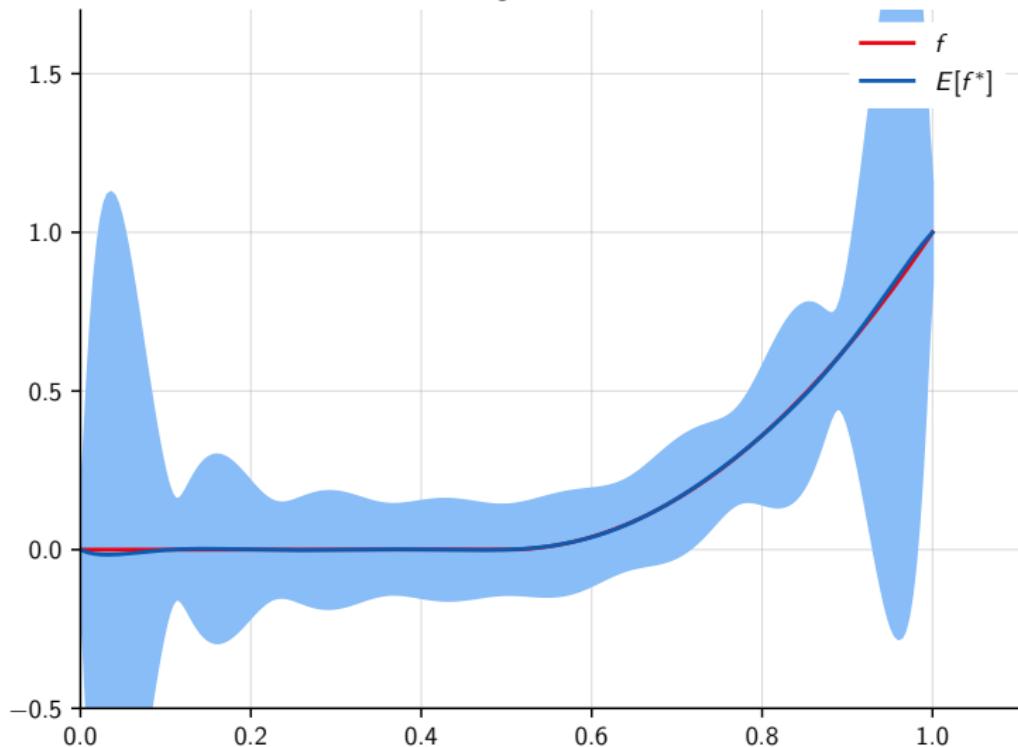
Degree $D = 7$



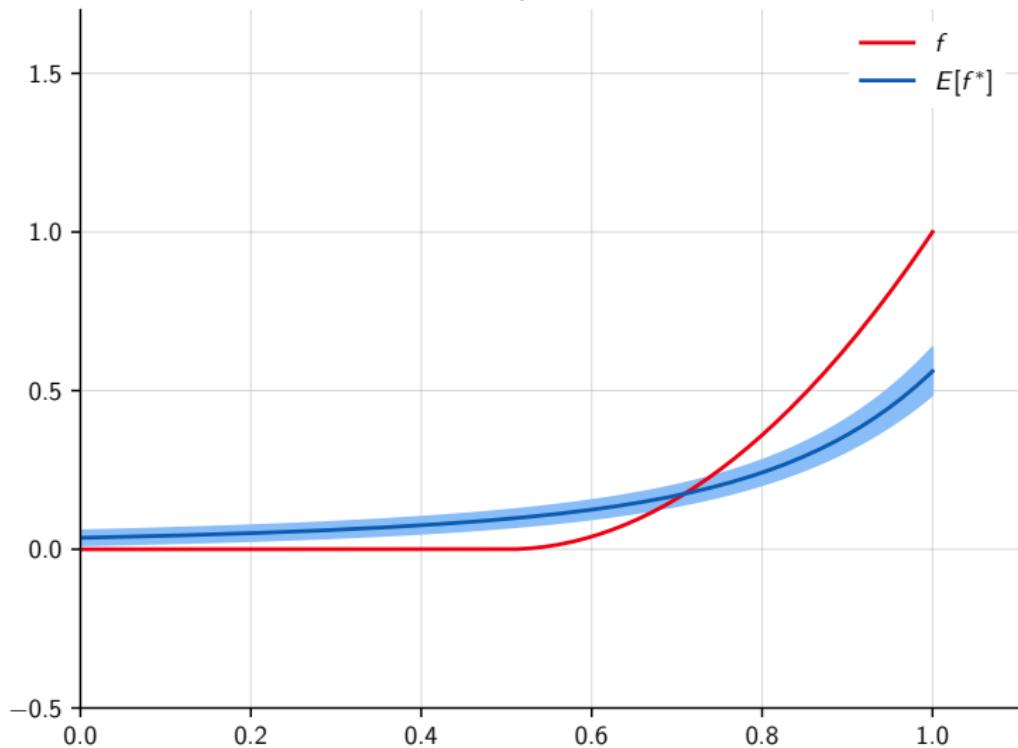
Degree $D = 8$



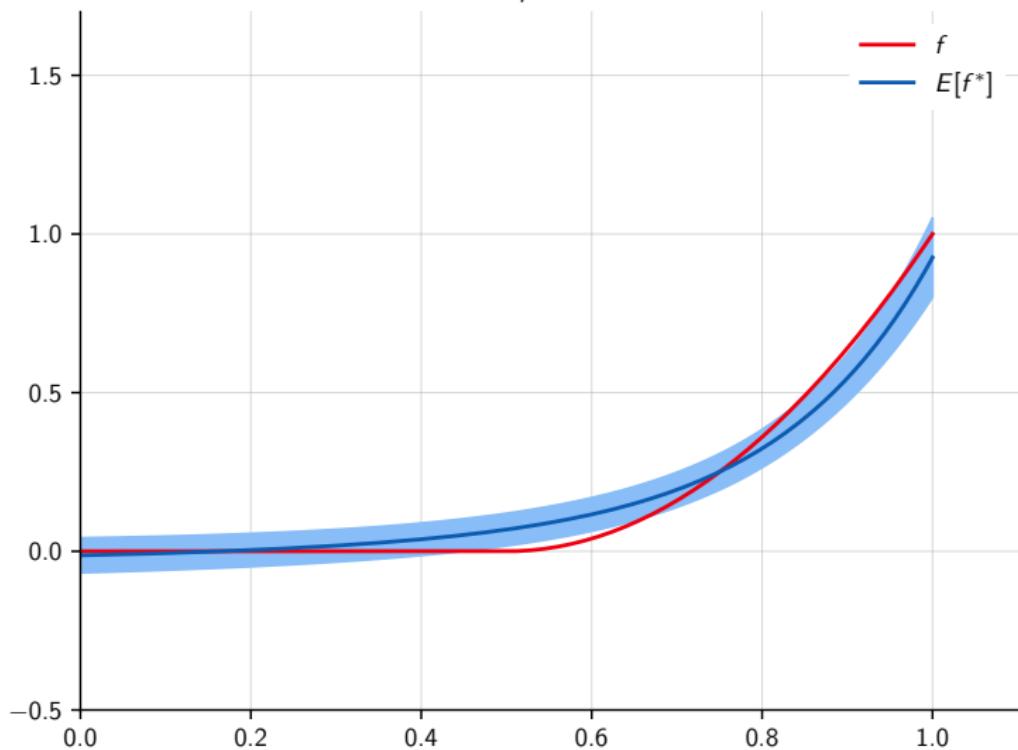
Degree $D = 9$



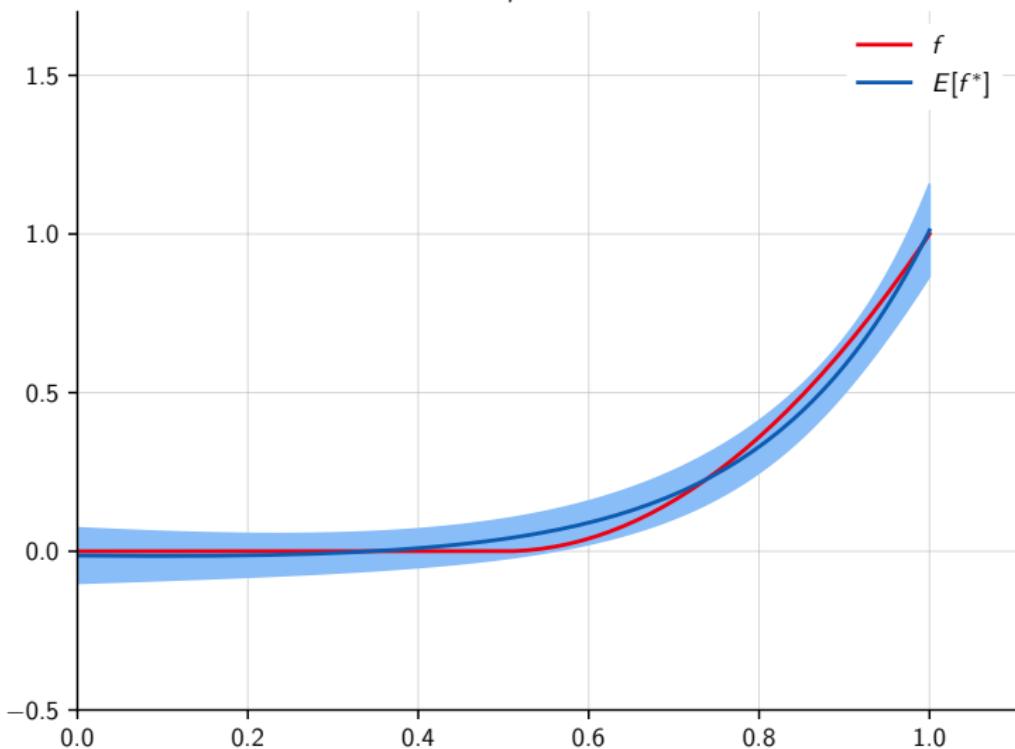
$$D = 9, \rho = 1 \times 10^1$$



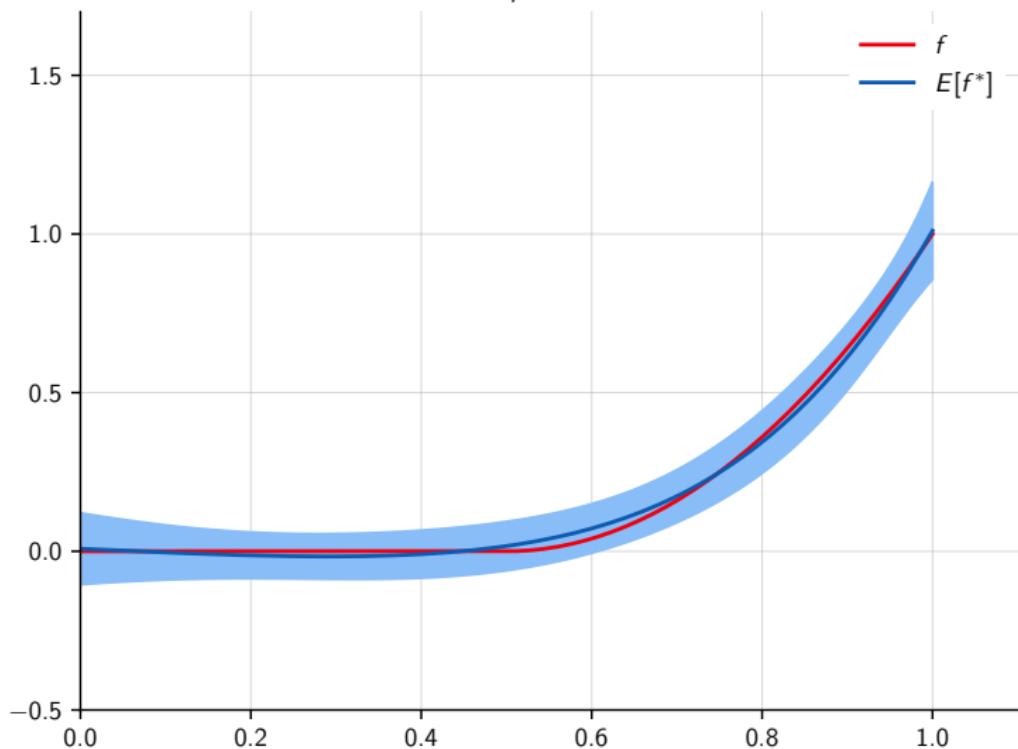
$$D = 9, \rho = 1 \times 10^0$$



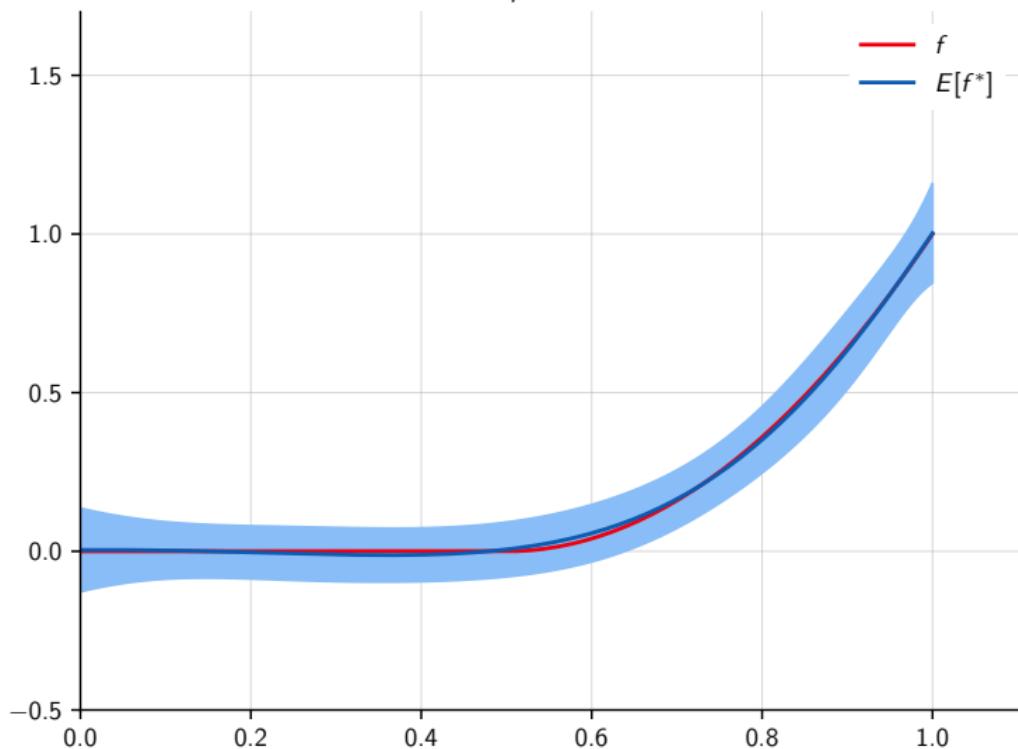
$$D = 9, \rho = 1 \times 10^{-1}$$



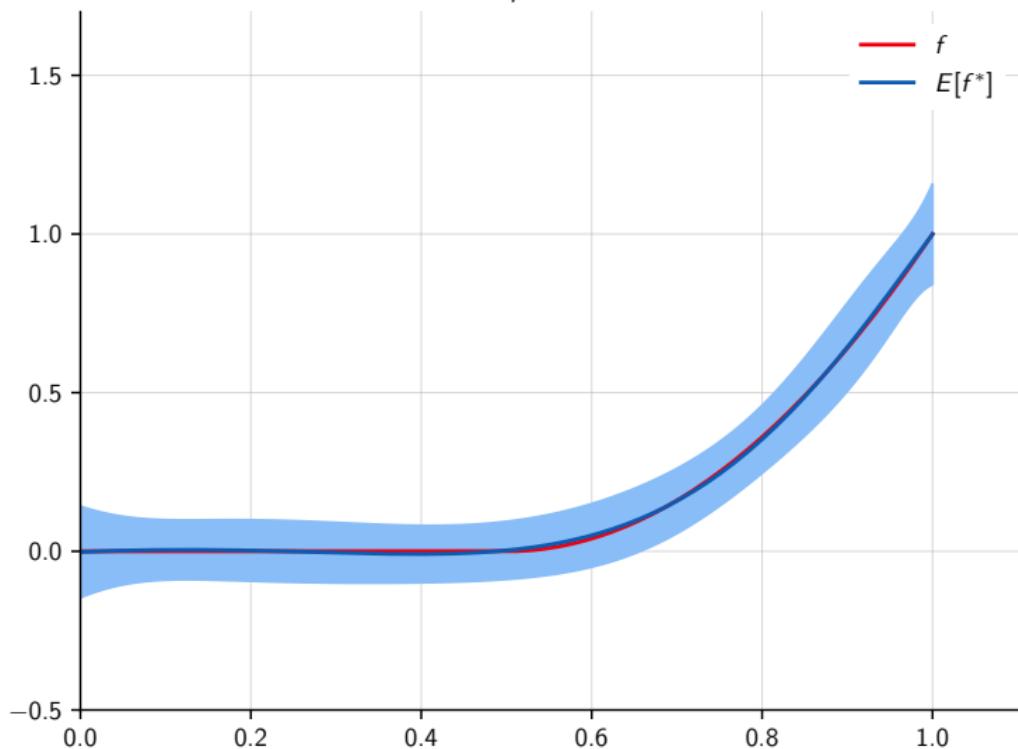
$$D = 9, \rho = 1 \times 10^{-2}$$



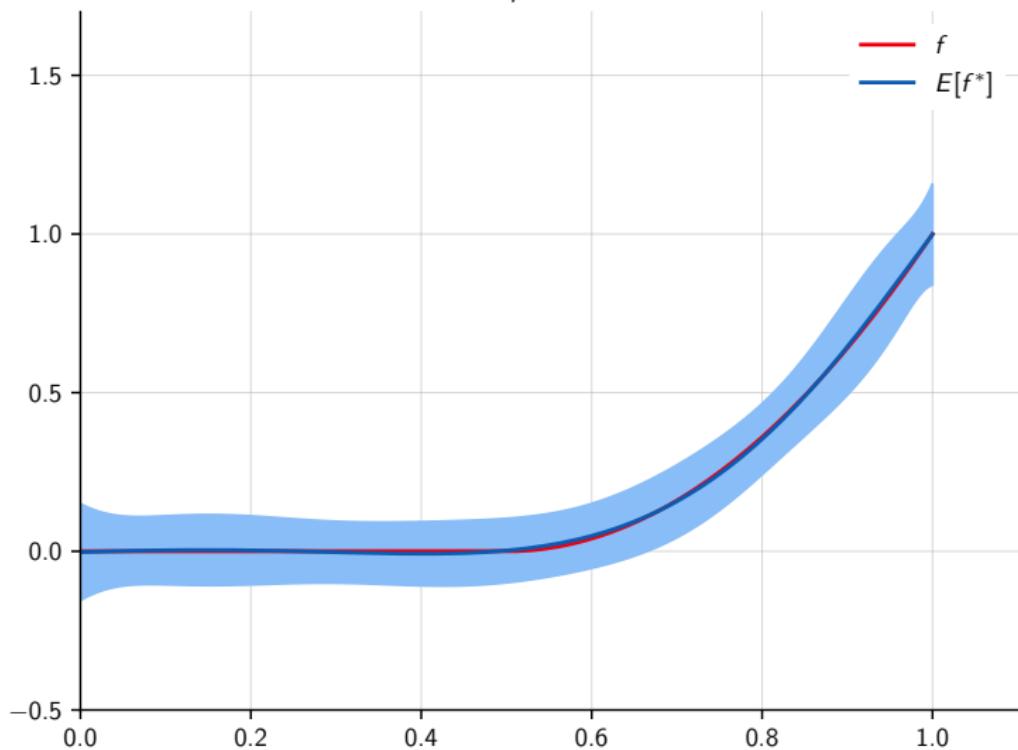
$$D = 9, \rho = 1 \times 10^{-3}$$



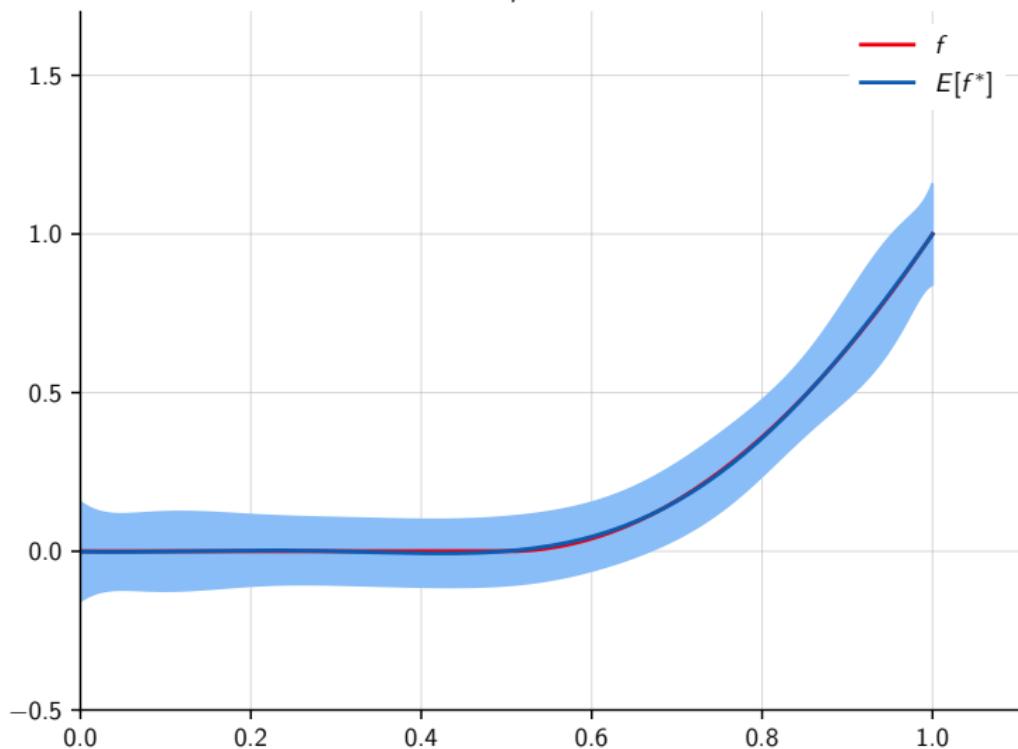
$$D = 9, \rho = 1 \times 10^{-4}$$



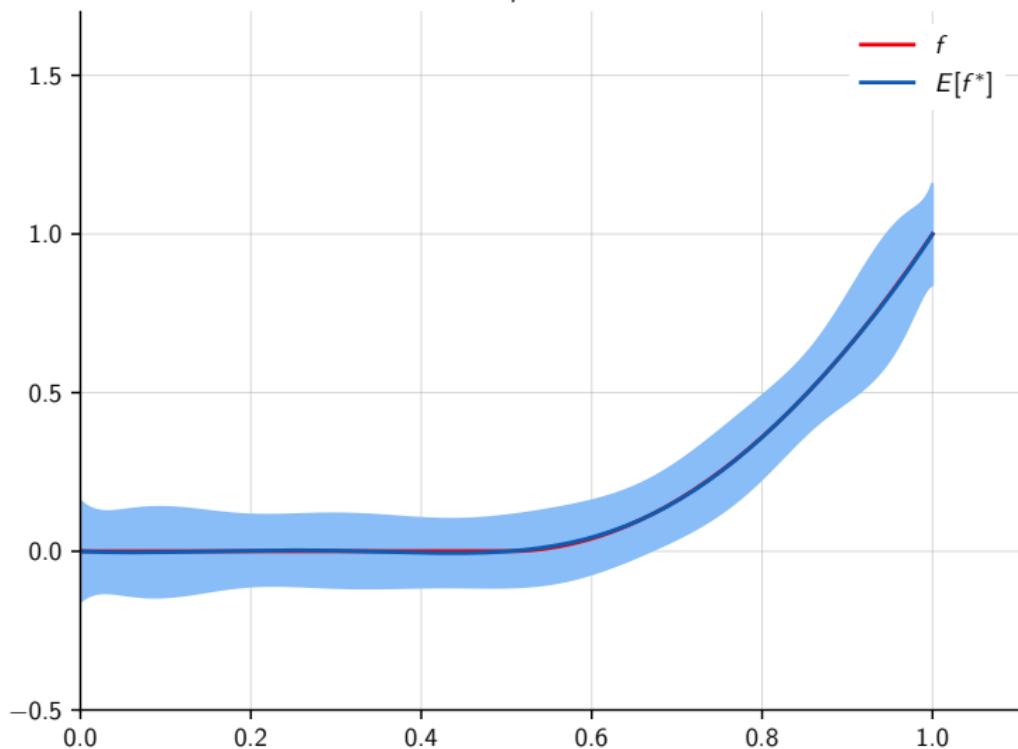
$$D = 9, \rho = 1 \times 10^{-5}$$



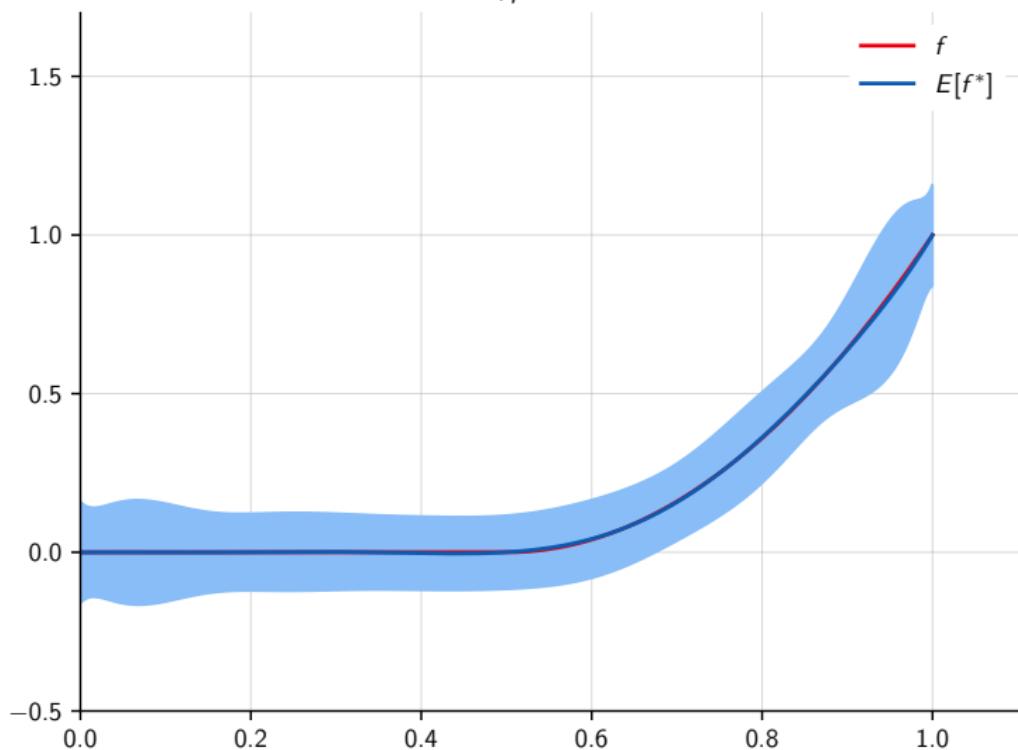
$$D = 9, \rho = 1 \times 10^{-6}$$



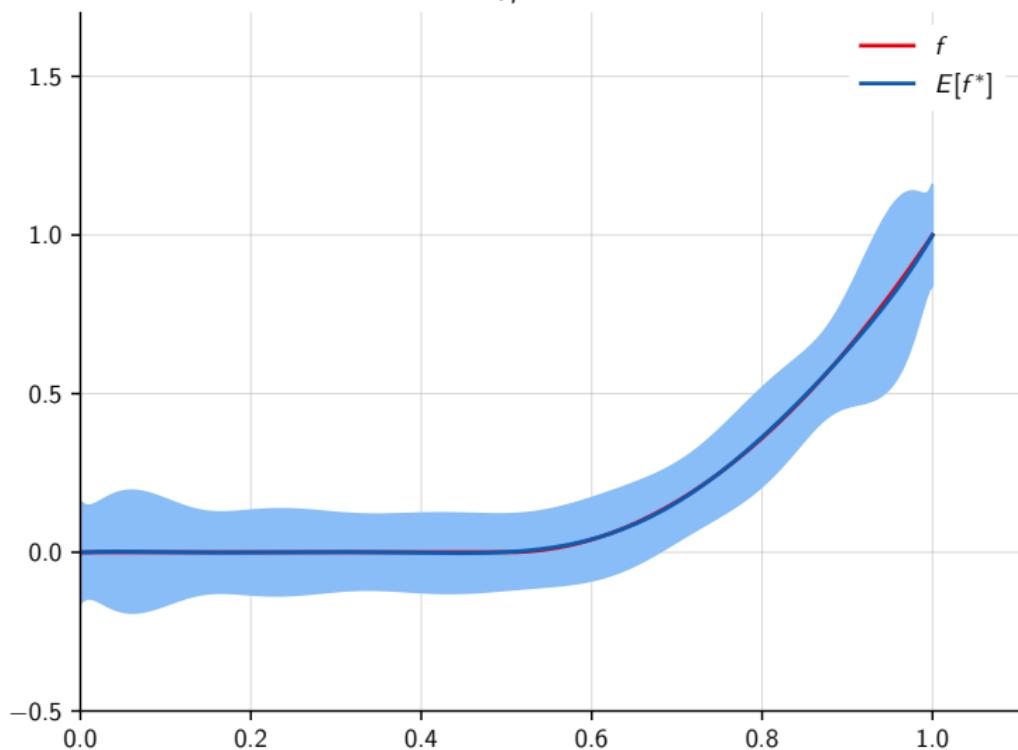
$$D = 9, \rho = 1 \times 10^{-7}$$



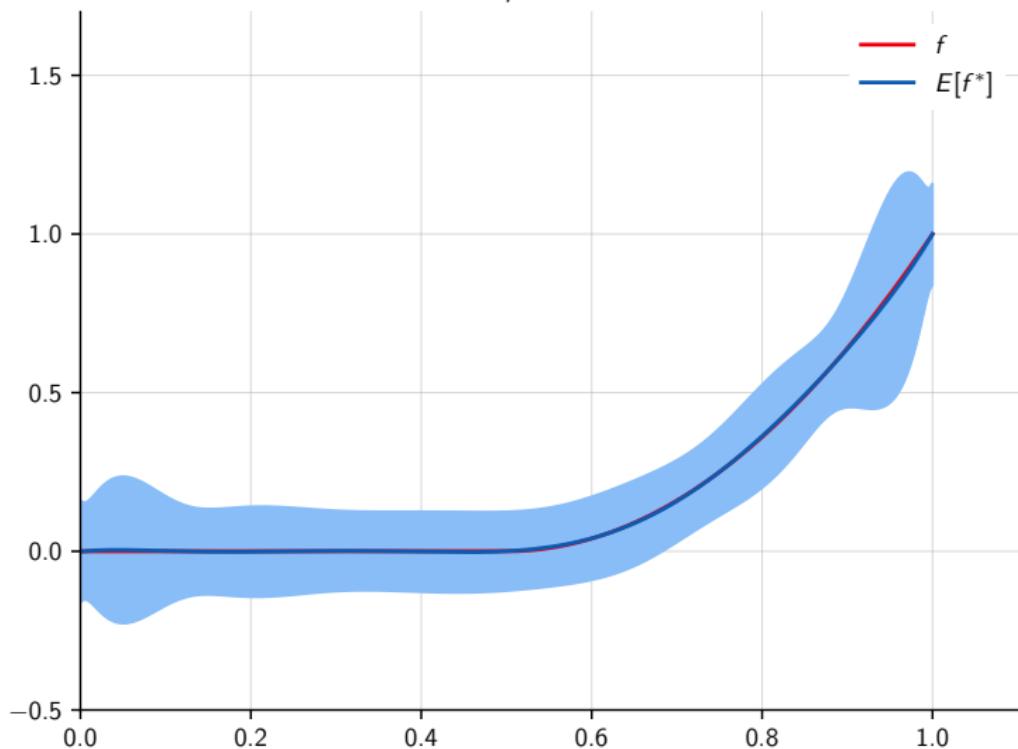
$$D = 9, \rho = 1 \times 10^{-8}$$



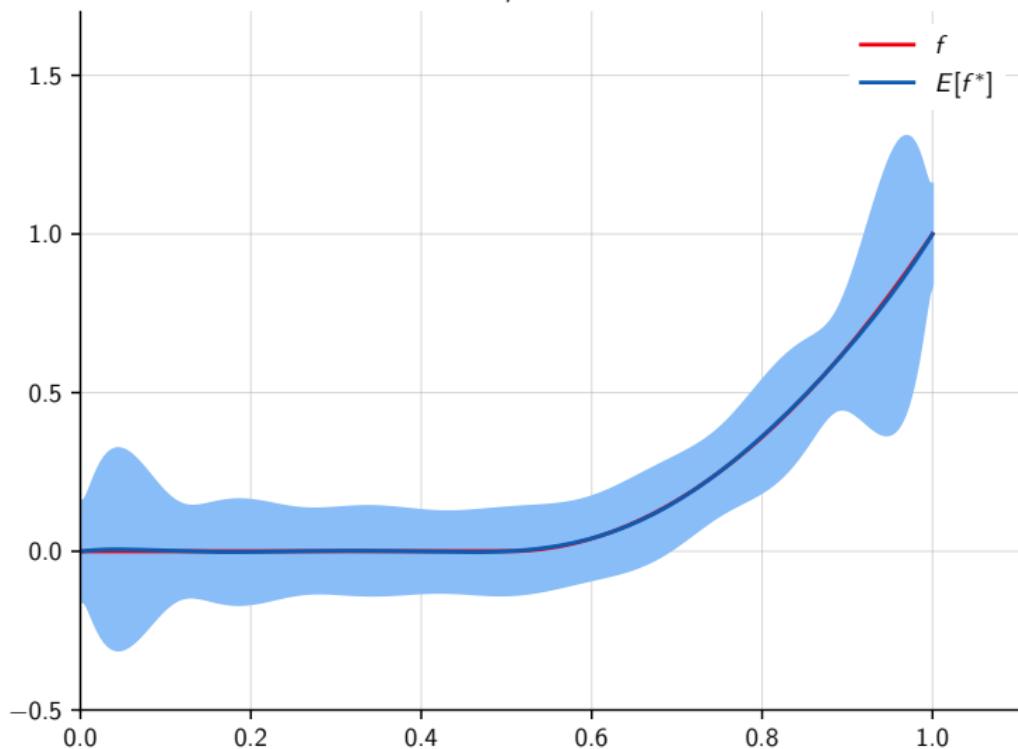
$$D = 9, \rho = 1 \times 10^{-9}$$



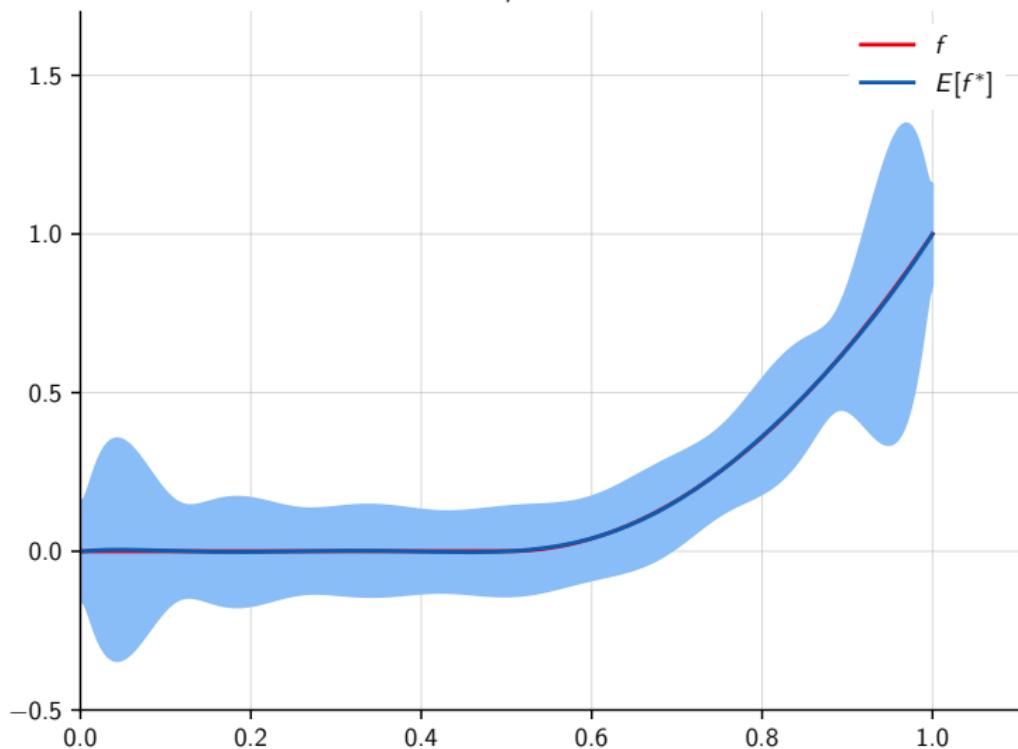
$$D = 9, \rho = 1 \times 10^{-10}$$



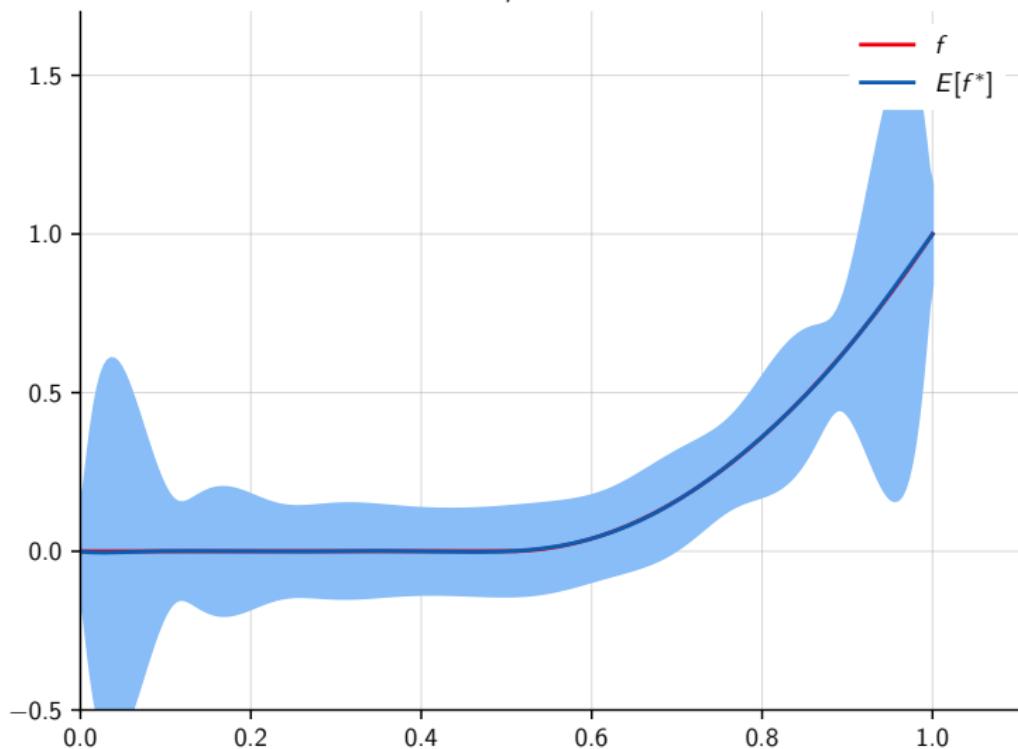
$$D = 9, \rho = 1 \times 10^{-11}$$



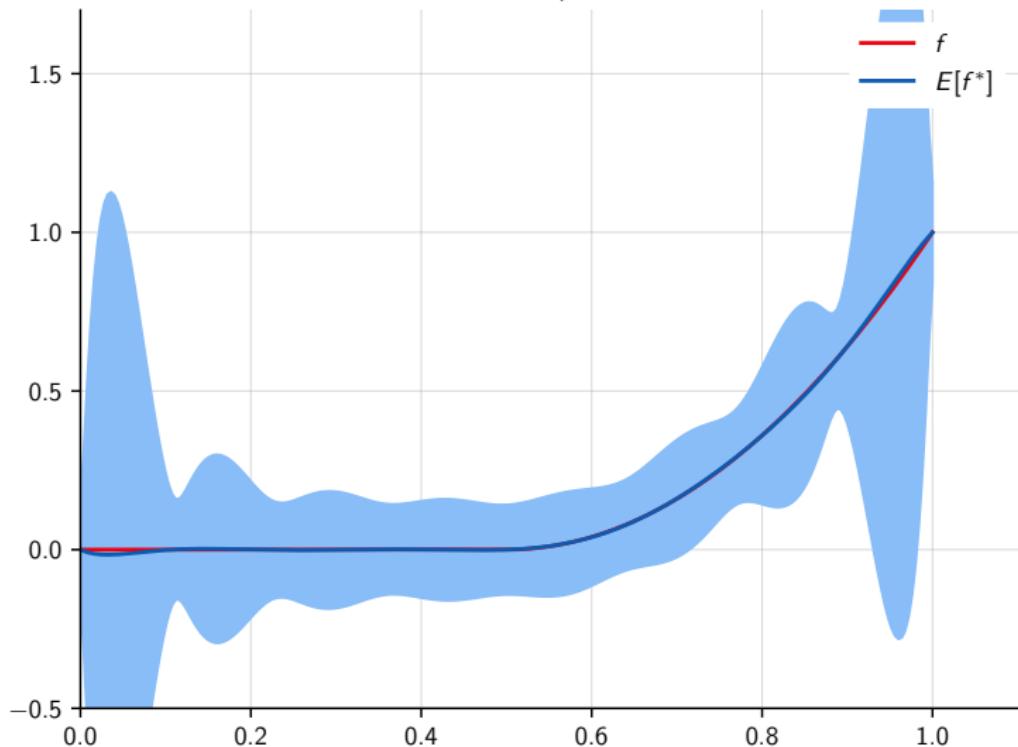
$$D = 9, \rho = 1 \times 10^{-12}$$



$$D = 9, \rho = 1 \times 10^{-13}$$



$$D = 9, \rho = 0$$



We can formalize these observations as follows:

Let x be fixed, y the “true” value associated to it, f^* the predictor we learned from the data-set \mathcal{D} , and $Y = f^*(x)$ be the value we predict at x .

If we consider that the training set \mathcal{D} is a random quantity, then f^* is random, and consequently Y is.

We have

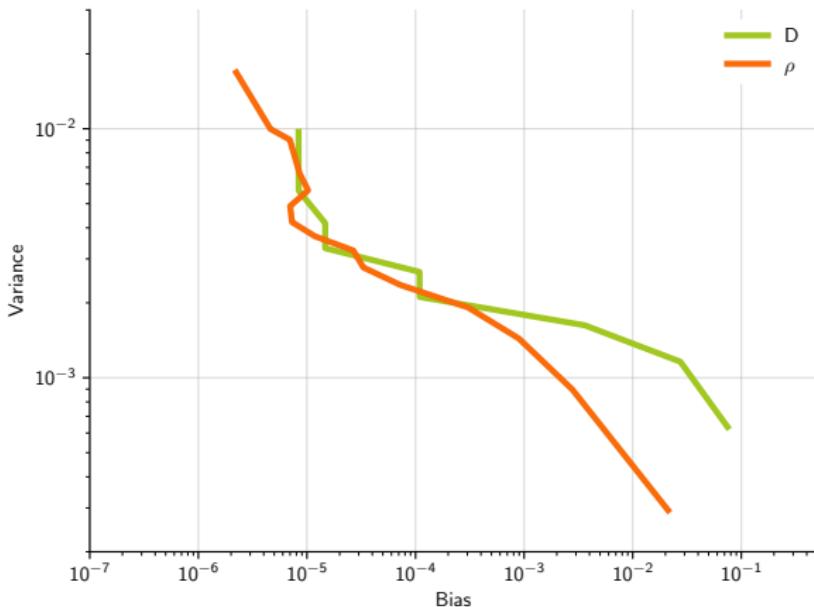
$$\begin{aligned}\mathbb{E}_{\mathcal{D}} ((Y - y)^2) &= \mathbb{E}_{\mathcal{D}} (Y^2 - 2Yy + y^2) \\&= \mathbb{E}_{\mathcal{D}} (Y^2) - 2\mathbb{E}_{\mathcal{D}} (Y)y + y^2 \\&= \underbrace{\mathbb{E}_{\mathcal{D}} (Y^2)}_{V_{\mathcal{D}}(Y)} - \underbrace{\mathbb{E}_{\mathcal{D}} (Y)^2}_{(\mathbb{E}_{\mathcal{D}} (Y) - y)^2} + \underbrace{\mathbb{E}_{\mathcal{D}} (Y)^2 - 2\mathbb{E}_{\mathcal{D}} (Y)y + y^2}_{(\mathbb{E}_{\mathcal{D}} (Y) - y)^2} \\&= \underbrace{(\mathbb{E}_{\mathcal{D}} (Y) - y)^2}_{\text{Bias}} + \underbrace{V_{\mathcal{D}} (Y)}_{\text{Variance}}\end{aligned}$$

This is the **bias-variance decomposition**:

- the bias term quantifies how much the model fits the data on average,
- the variance term quantifies how much the model changes across data-sets.

(Geman and Bienenstock, 1992)

From this comes the **bias-variance tradeoff**:



Reducing the capacity makes f^* fit the data less on average, which increases the bias term. Increasing the capacity makes f^* vary a lot with the training data, which increases the variance term.

Is all this probabilistic?

Conceptually model-fitting and regularization can be interpreted as Bayesian inference.

This approach consists of **modeling the parameters A of the model themselves as random quantities following a prior distribution μ_A .**

By looking at the data \mathcal{D} , we can estimate a posterior distribution for the said parameters,

$$\mu_A(\alpha | \mathcal{D} = \mathbf{d}) \propto \mu_{\mathcal{D}}(\mathbf{d} | A = \alpha) \mu_A(\alpha),$$

and from that their most likely values.

So instead of a penalty term, we define a prior distribution, which is usually more intellectually satisfying.

For instance, consider a polynomial model with Gaussian prior, that is

$$\forall n, \quad Y_n = \sum_{d=0}^D A_d X_n^d + \Delta_n,$$

where

$$\forall d, \quad A_d \sim \mathcal{N}(0, \xi), \quad \forall n, \quad X_n \sim \mu_X, \quad \Delta_n \sim \mathcal{N}(0, \sigma)$$

all independent.

For clarity, let $A = (A_0, \dots, A_D)$ and $\alpha = (\alpha_0, \dots, \alpha_D)$.

Remember that $\mathcal{D} = \{(X_1, Y_1), \dots, (X_N, Y_N)\}$ is the (random) training set and $\mathbf{d} = \{(x_1, y_1), \dots, (x_N, y_N)\}$ is a realization.

$$\begin{aligned}
& \log \mu_A(\alpha \mid \mathcal{D} = \mathbf{d}) \\
&= \log \frac{\mu_{\mathcal{D}}(\mathbf{d} \mid A = \alpha) \mu_A(\alpha)}{\mu_{\mathcal{D}}(\mathbf{d})} \\
&= \log \mu_{\mathcal{D}}(\mathbf{d} \mid A = \alpha) + \log \mu_A(\alpha) - \log Z \\
&= \log \prod_n \mu(x_n, y_n \mid A = \alpha) + \log \mu_A(\alpha) - \log Z \\
&= \log \prod_n \mu(y_n \mid X_n = x_n, A = \alpha) \underbrace{\mu(x_n \mid A = \alpha)}_{= \mu(x_n)} + \log \mu_A(\alpha) - \log Z \\
&= \log \prod_n \mu(y_n \mid X_n = x_n, A = \alpha) + \log \mu_A(\alpha) - \log Z' \\
&= - \underbrace{\frac{1}{2\sigma^2} \sum_n \left(y_n - \sum_d \alpha_d x_n^d \right)^2}_{\text{Gaussian noise on } Y} - \underbrace{\frac{1}{2\xi^2} \sum_d \alpha_d^2}_{\text{Gaussian prior on } A} - \log Z''.
\end{aligned}$$

Taking $\rho = \sigma^2/\xi^2$ gives the penalty term of the previous slides.

Regularization seen through that prism is intuitive: The stronger the prior, the more evidence you need to deviate from it.

Deep learning

2.4. Proper evaluation protocols

François Fleuret

<https://fleuret.org/dlc/>



UNIVERSITÉ
DE GENÈVE

Learning algorithms, in particular deep-learning ones, require the tuning of many meta-parameters.

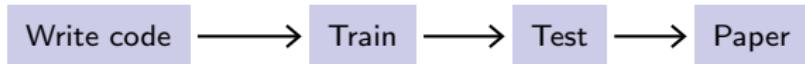
These parameters have a strong impact on the performance, resulting in a “meta” over-fitting through experiments.

We must be extra careful with performance estimation.

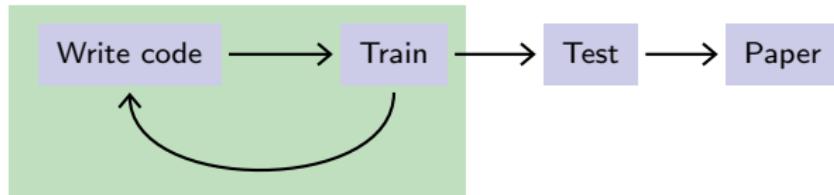
Running 100 times the MNIST experiment, with randomized weights, we get:

Worst	Median	Best
1.3%	1.0%	0.82%

The ideal development cycle is

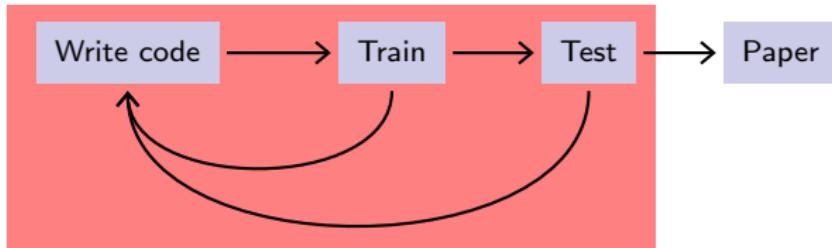


or in practice something like

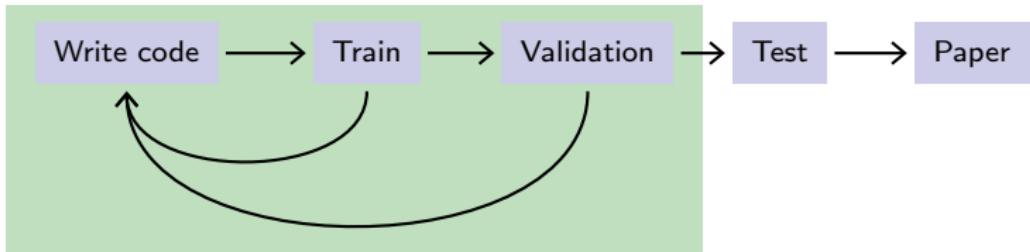


There may be over-fitting, but it does not bias the final performance evaluation.

Unfortunately, it often looks like



This should be avoided at all costs. The standard strategy is to have a separate validation set for the tuning.

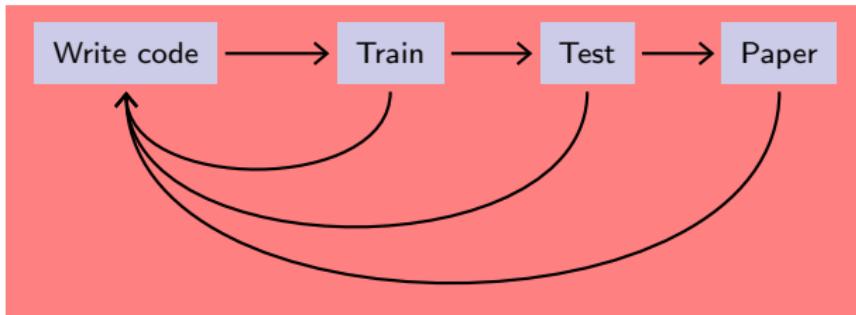


When data is scarce, one can use cross-validation: average through multiple random splits of the data in a train and a validation sets.

There is no unbiased estimator of the variance of cross-validation valid under all distributions (Bengio and Grandvalet, 2004).

Some data-sets (MNIST!) have been used by thousands of researchers, over millions of experiments, in hundreds of papers.

The global overall process looks more like



“Cheating” in machine learning, from bad to “are you kidding?”:

- “Early evaluation stopping”,
- meta-parameter (over-)tuning,
- data-set selection,
- algorithm data-set specific clauses,
- seed selection.

Top-tier conferences are demanding regarding experiments, and are biased against “complicated” pipelines.

The community pushes toward accessible implementations, reference data-sets, leader boards, and constant upgrades of benchmarks.

Deep learning

2.5. Basic clusterings and embeddings

François Fleuret

<https://fleuret.org/dlc/>



UNIVERSITÉ
DE GENÈVE

Deep learning models combine embeddings and dimension reduction operations.

They parametrize and re-parametrize multiple times the input signal into representations that get more and more invariant and noise free.

To get an intuition of how this is possible, we consider here two standard algorithms:

- K -means, and
- Principal Component Analysis (PCA).

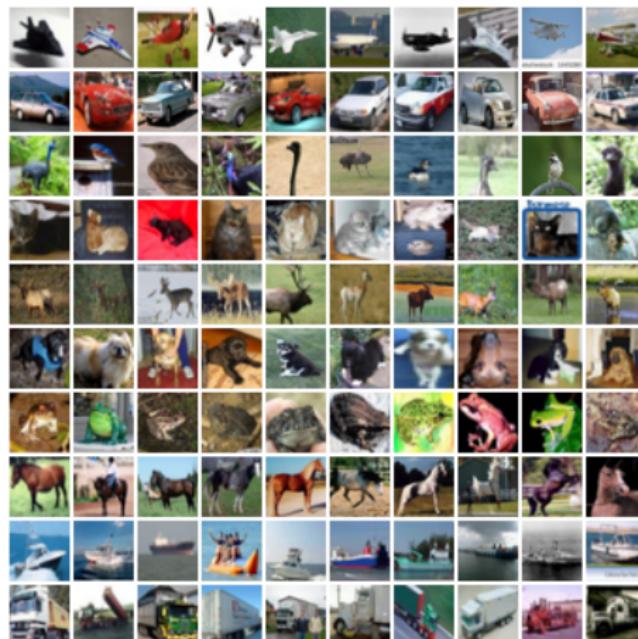
We will illustrate these methods on our two favorite data-sets.

MNIST data-set

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

28 × 28 grayscale images, 60k train samples, 10k test samples.

CIFAR10 data-set



32 × 32 color images, 50k train samples, 10k test samples.

(Krizhevsky, 2009, chap. 3)

Given

$$x_n \in \mathbb{R}^D, \quad n = 1, \dots, N,$$

and a fixed number of clusters $K > 0$, K -means tries to find K “centroids” that span uniformly the training population.

Given a point, the index of its closest centroid is a good coding.

Formally, [Lloyd's algorithm for] K -means (approximately) solves

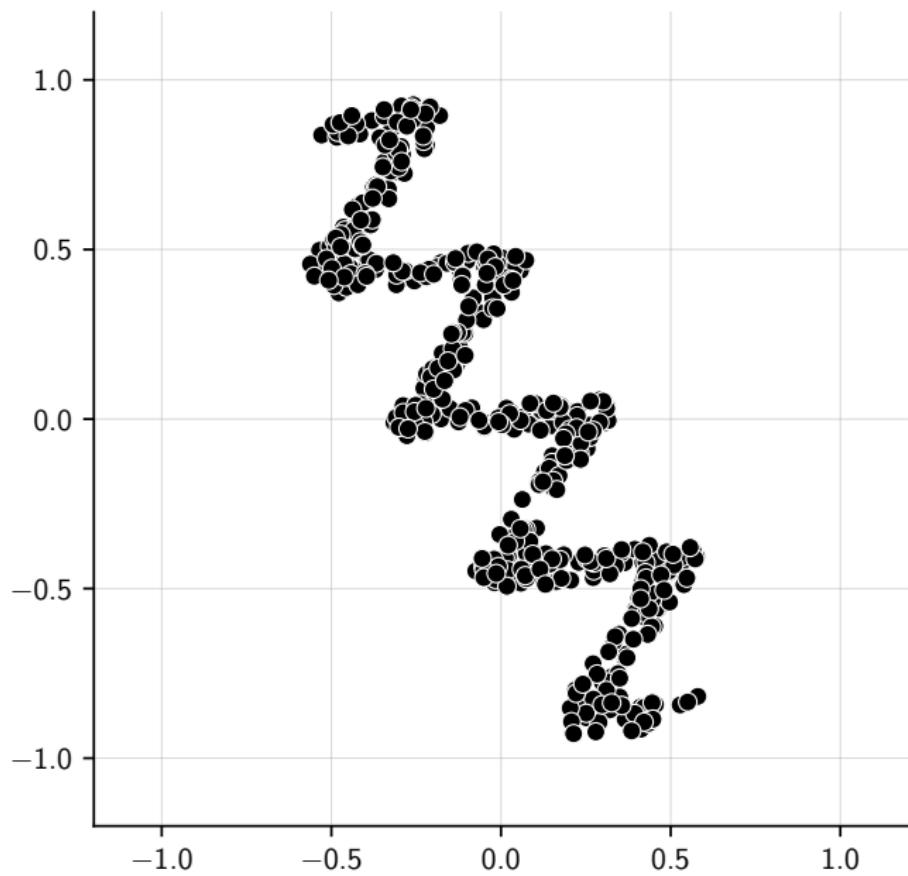
$$\underset{c_1, \dots, c_K \in \mathbb{R}^D}{\operatorname{argmin}} \sum_n \min_k \|x_n - c_k\|^2.$$

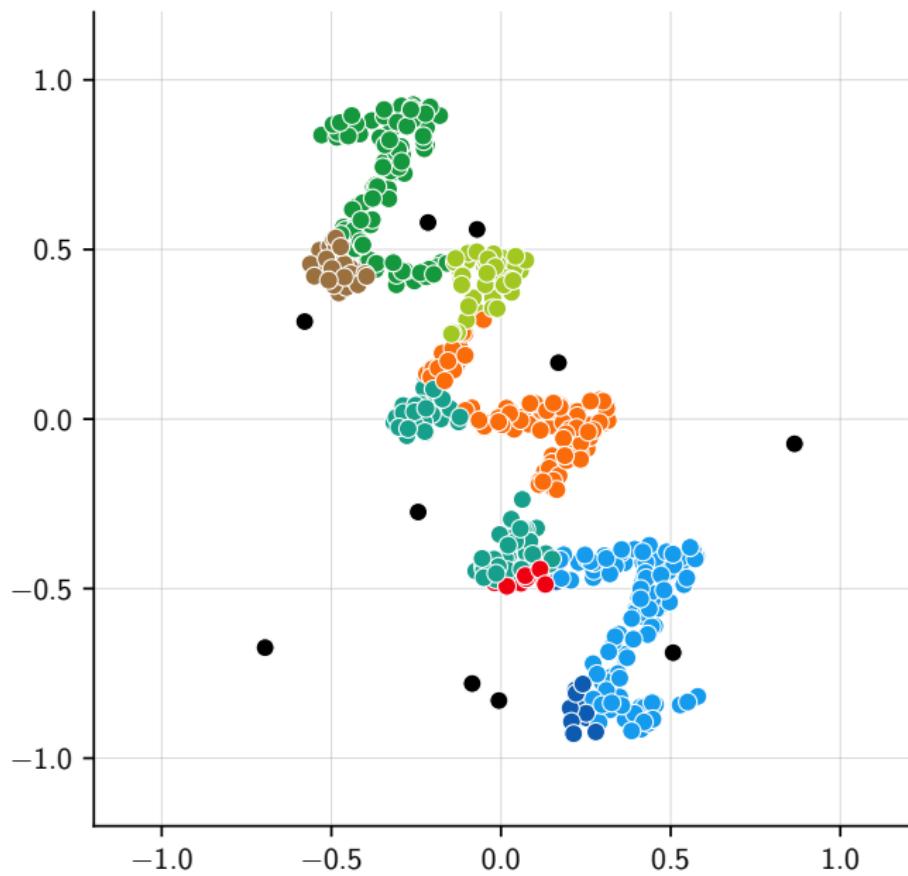
This is achieved with a random initialization of c_1^0, \dots, c_K^0 followed by repeating until convergence:

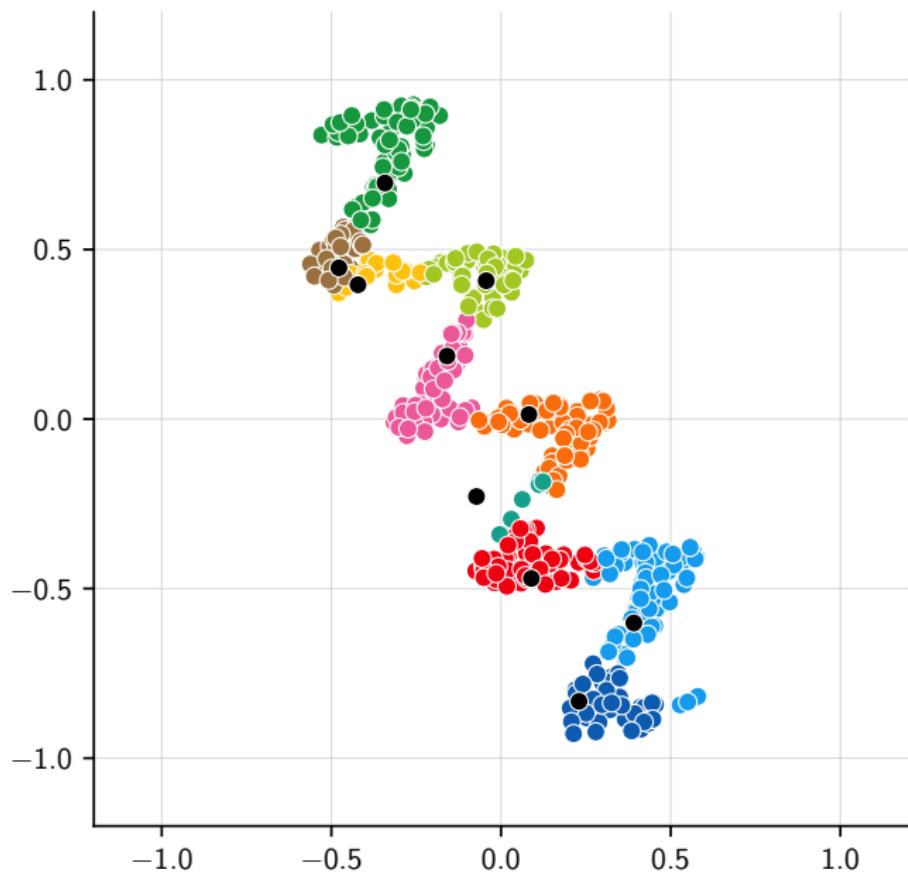
$$\forall n, k_n^t = \underset{k}{\operatorname{argmin}} \|x_n - c_k^t\| \quad (1)$$

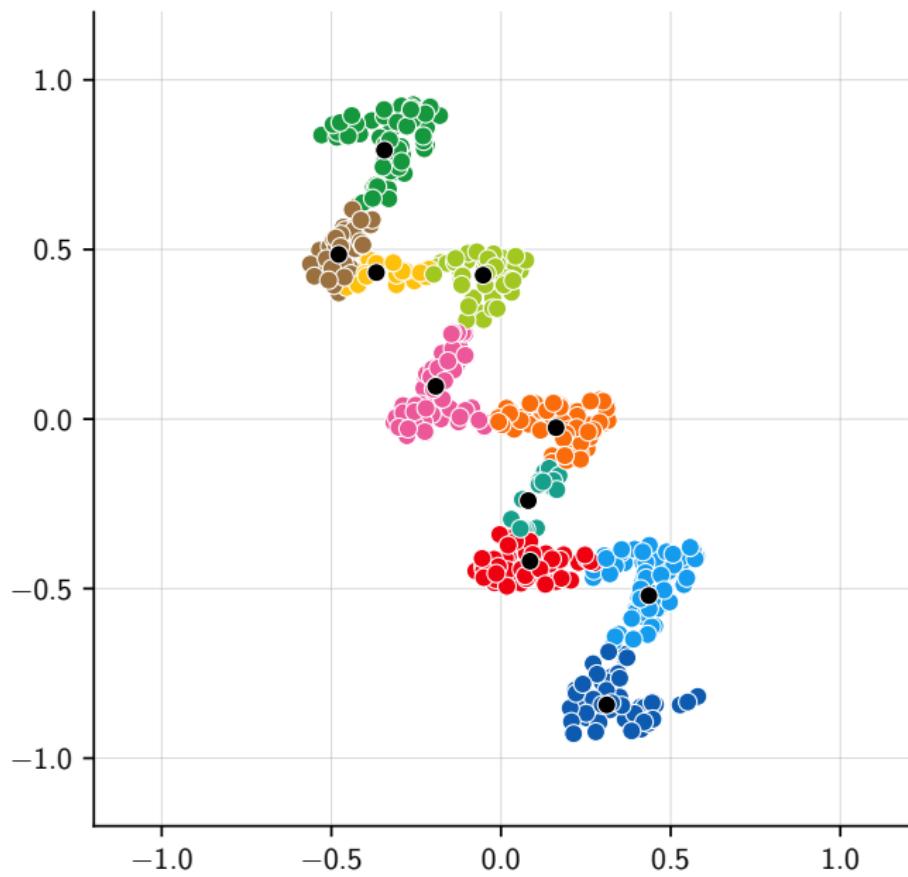
$$\forall k, c_k^{t+1} = \frac{1}{|n : k_n^t = k|} \sum_{n : k_n^t = k} x_n \quad (2)$$

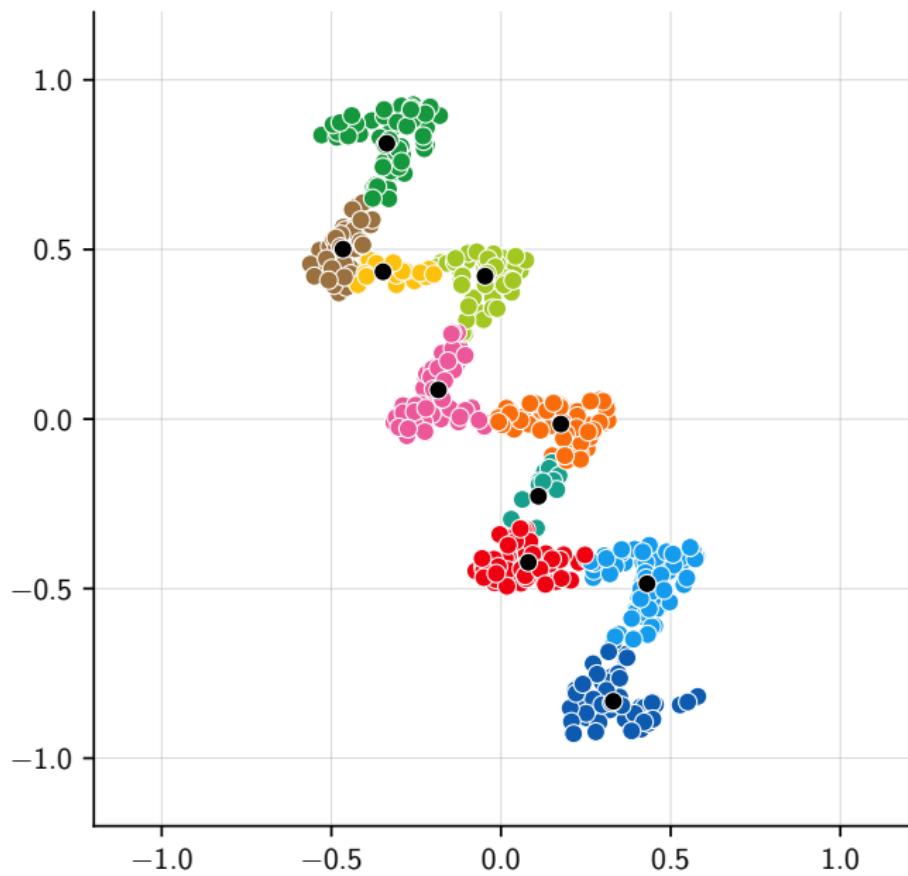
At every iteration, (1) each sample is associated to its closest centroid's cluster, and (2) each centroid is updated to the average of its cluster.

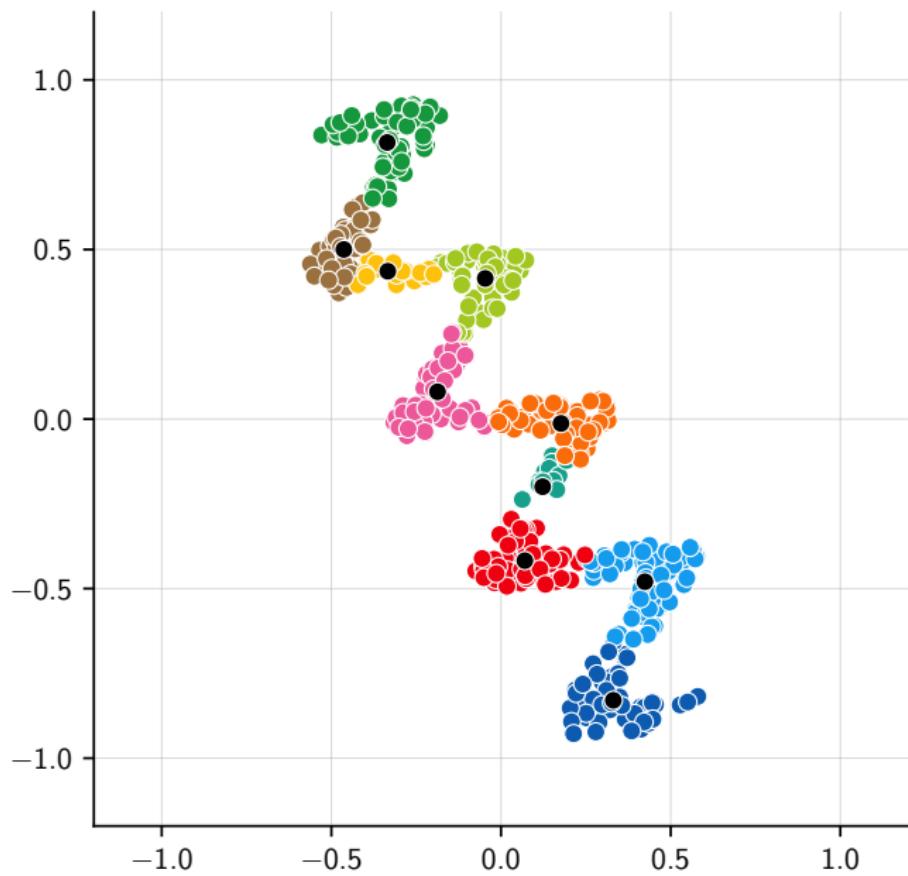


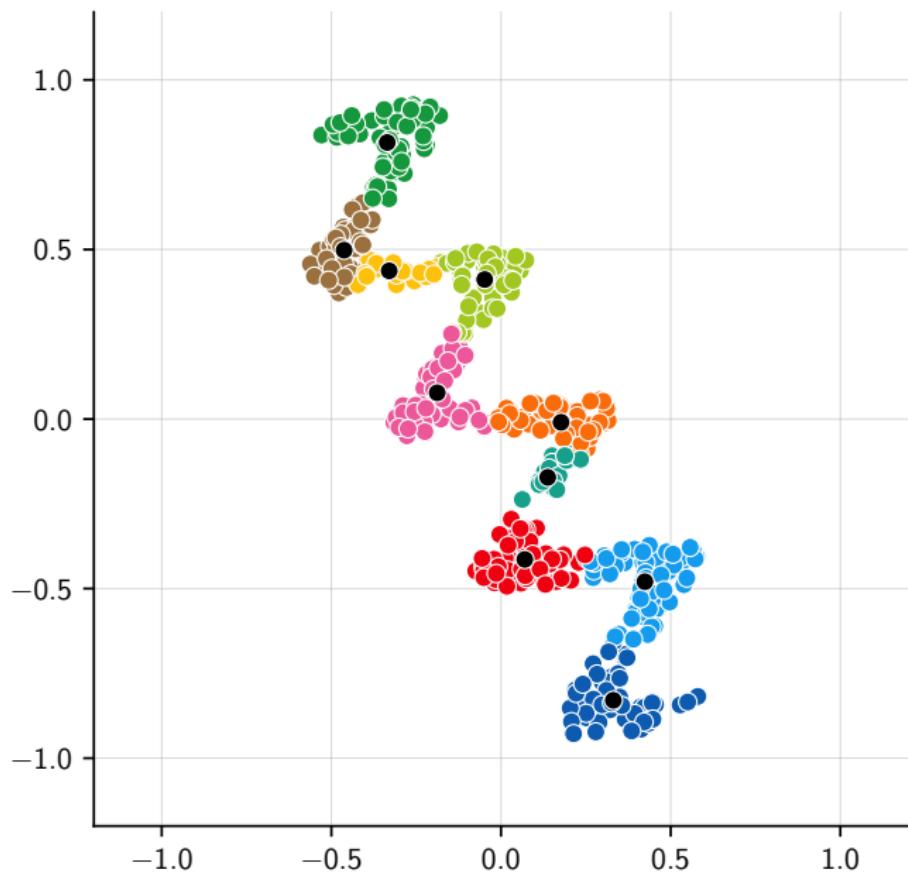


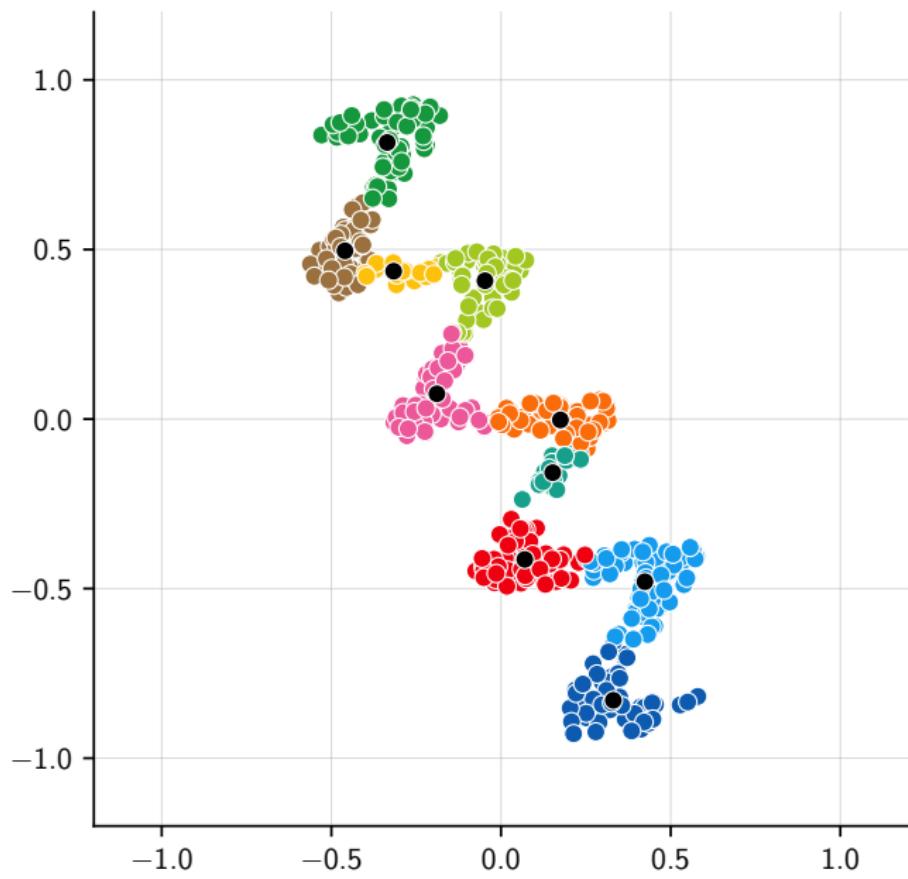


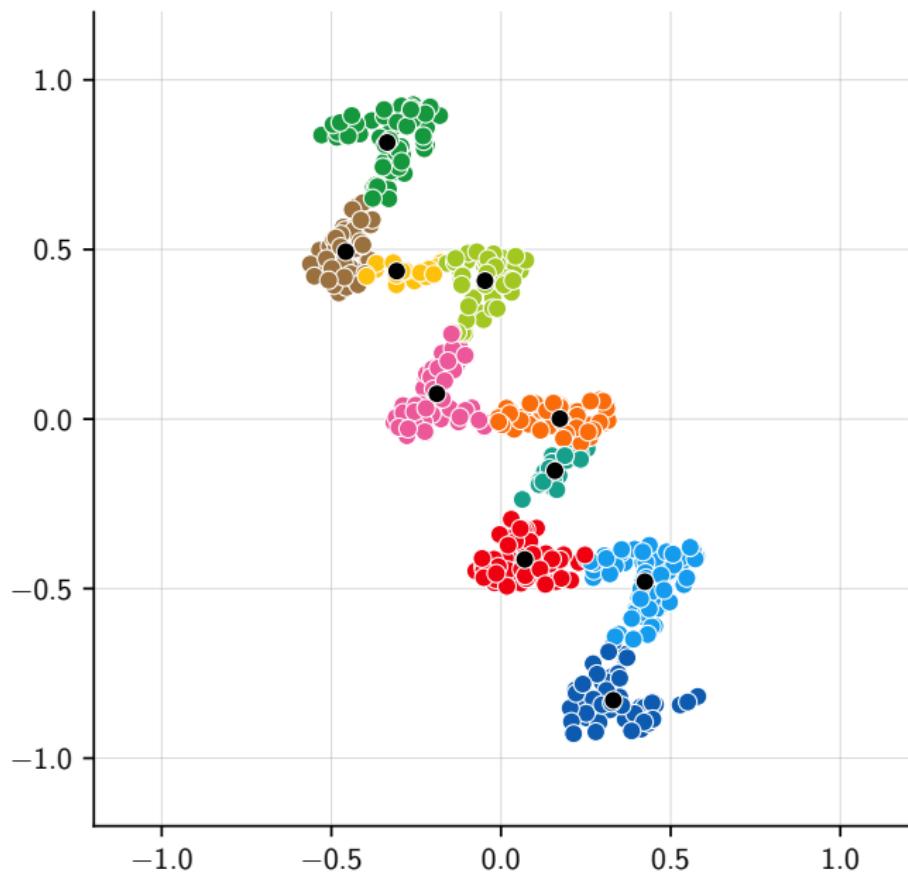


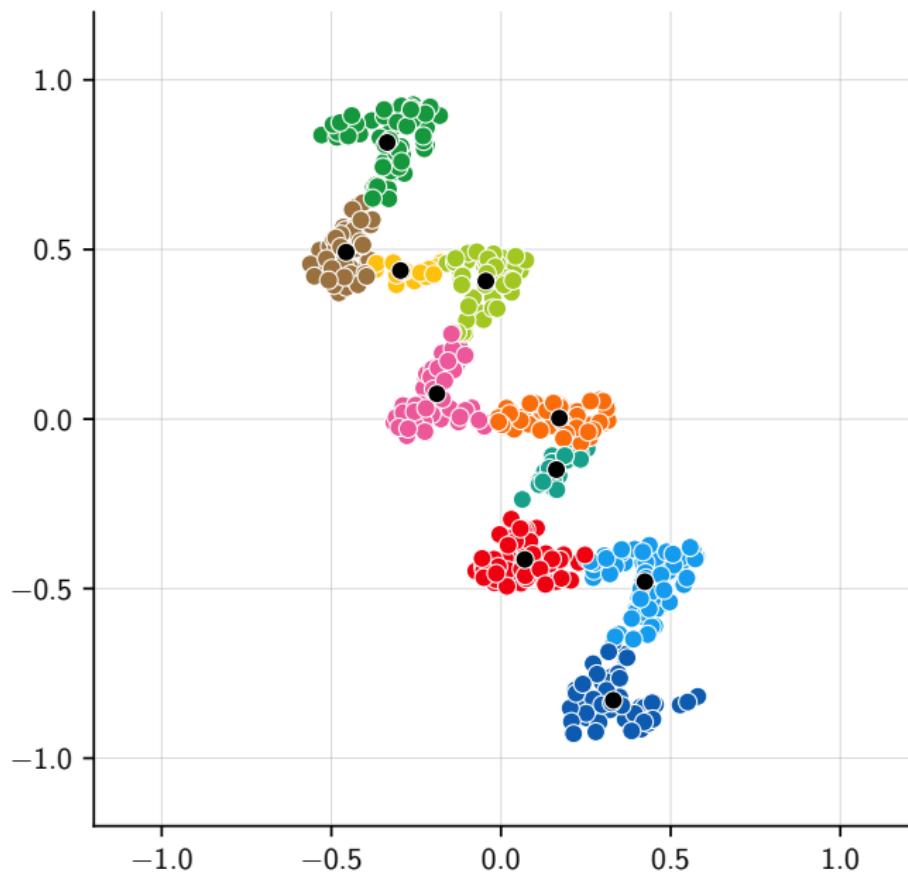


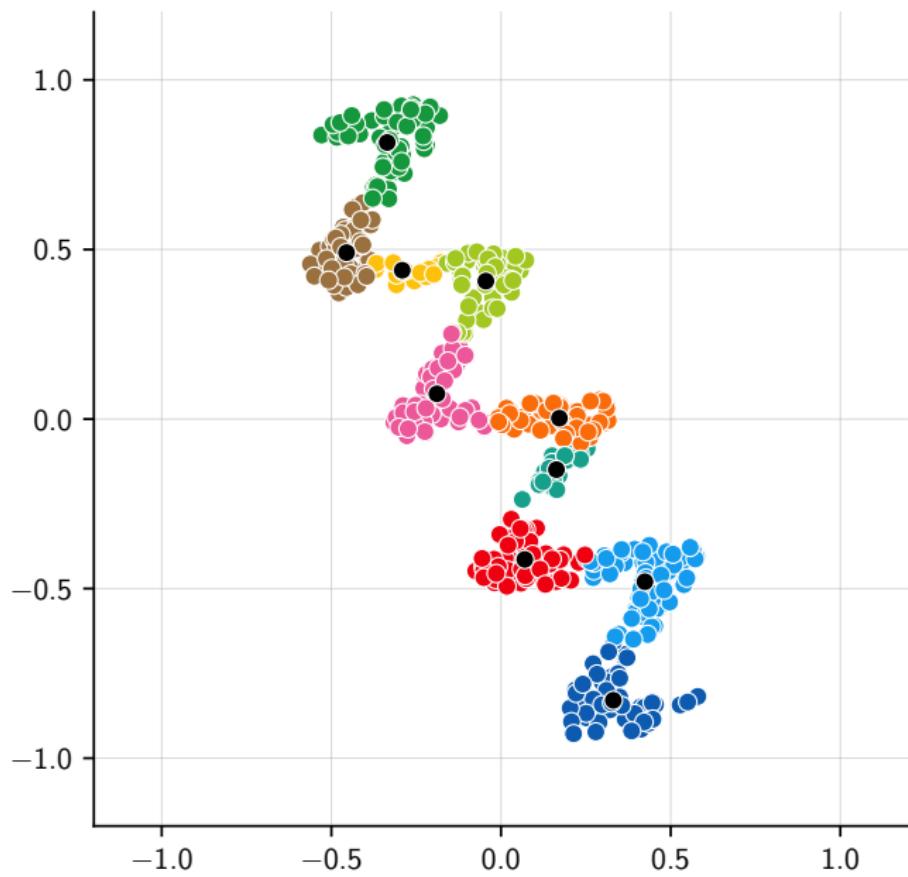


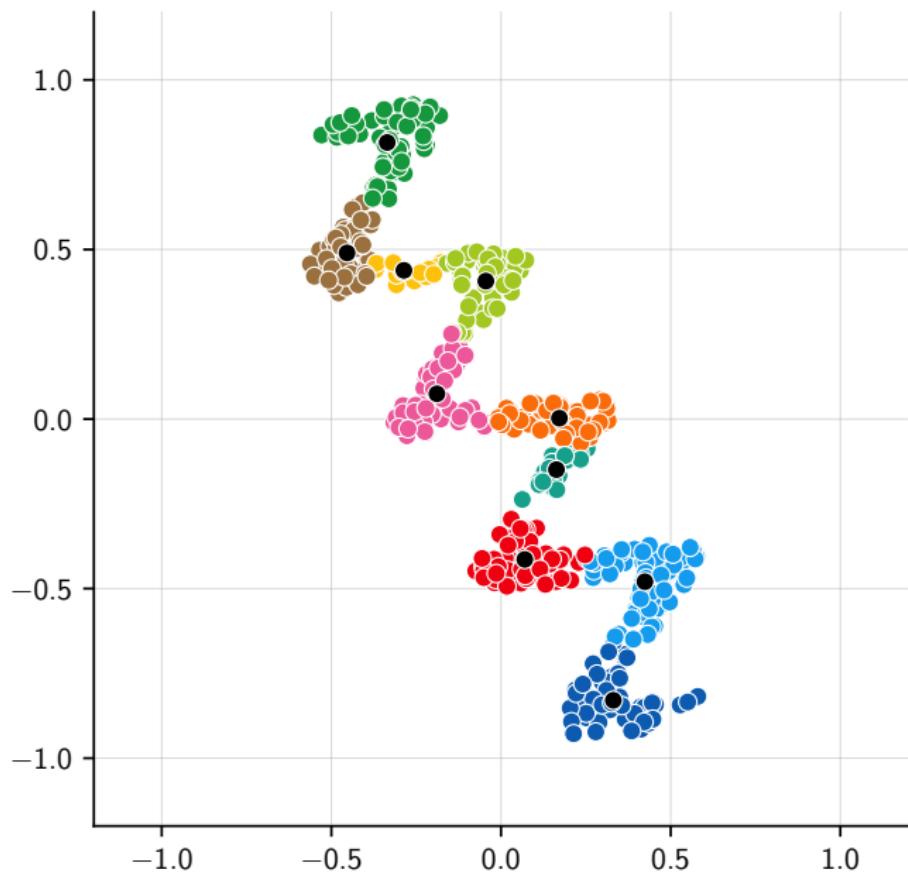












We can apply that algorithm to images from MNIST ($1 \times 28 \times 28$) or CIFAR10 ($3 \times 32 \times 32$) by considering them as vectors from \mathbb{R}^{784} and \mathbb{R}^{3072} respectively.

Centroids can similarly be visualized as images, and clustering can be done per-class, or for all the classes mixed.

0	00	0000	000000000
1	11	1111	111111111
2	22	2222	2222222222
3	33	3333	3333333333
4	44	4444	4444444444
5	55	5555	5555555555
6	66	6666	6666666666
7	77	7777	7777777777
8	88	8888	8888888888
9	99	9999	9999999999
0	09	3109	00988319

$K = 1$

$K = 2$

$K = 4$

$K = 8$

000000000000000000
111111111111111111
222222222222222222
333333333333333333
444444444444444444
555555555555555555
666666666666666666
777777777777777777
888888888888888888
999999999999999999
1095629729680310

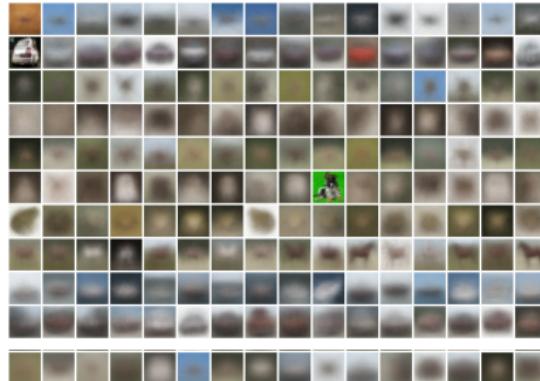
$K = 16$

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

$$K = 32$$



$K = 1 \quad K = 2 \quad K = 4 \quad K = 8$



$K = 16$



$K = 32$

The Principal Component Analysis (PCA) aims also at extracting an information in a L^2 sense. Instead of clusters, it looks for an “affine subspace”, i.e. a point and a basis, that spans the data.

Given data-points

$$x_n \in \mathbb{R}^D, \quad n = 1, \dots, N$$

(A) compute the average and center the data

$$\begin{aligned}\bar{x} &= \frac{1}{N} \sum_n x_n \\ \forall n, \quad x_n^{(0)} &= x_n - \bar{x}\end{aligned}$$

and then for $t = 1, \dots, D$,

(B) pick the direction and project the data

$$\begin{aligned}v_t &= \underset{\|v\|=1}{\operatorname{argmax}} \sum_n \left(v \cdot x_n^{(t-1)} \right)^2 \\ \forall n, \quad x_n^{(t)} &= x_n^{(t-1)} - \left(v_t \cdot x_n^{(t-1)} \right) v_t.\end{aligned}$$

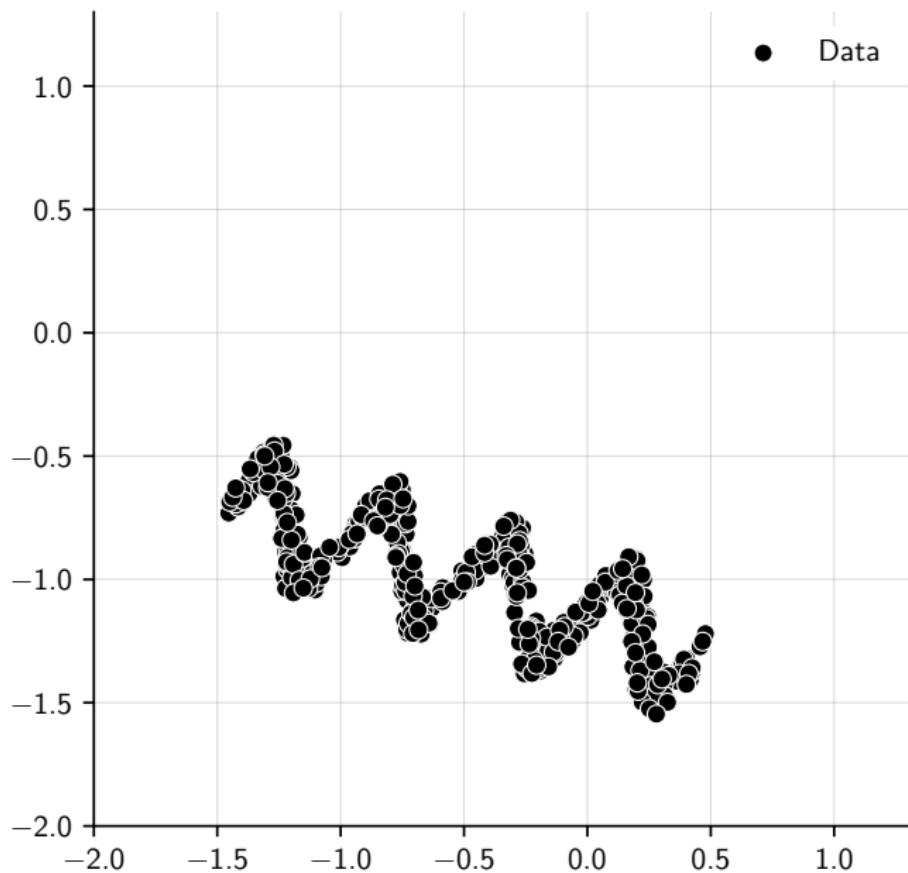
Although this is a simple way to envision PCA, standard implementations rely on an eigen decomposition. With

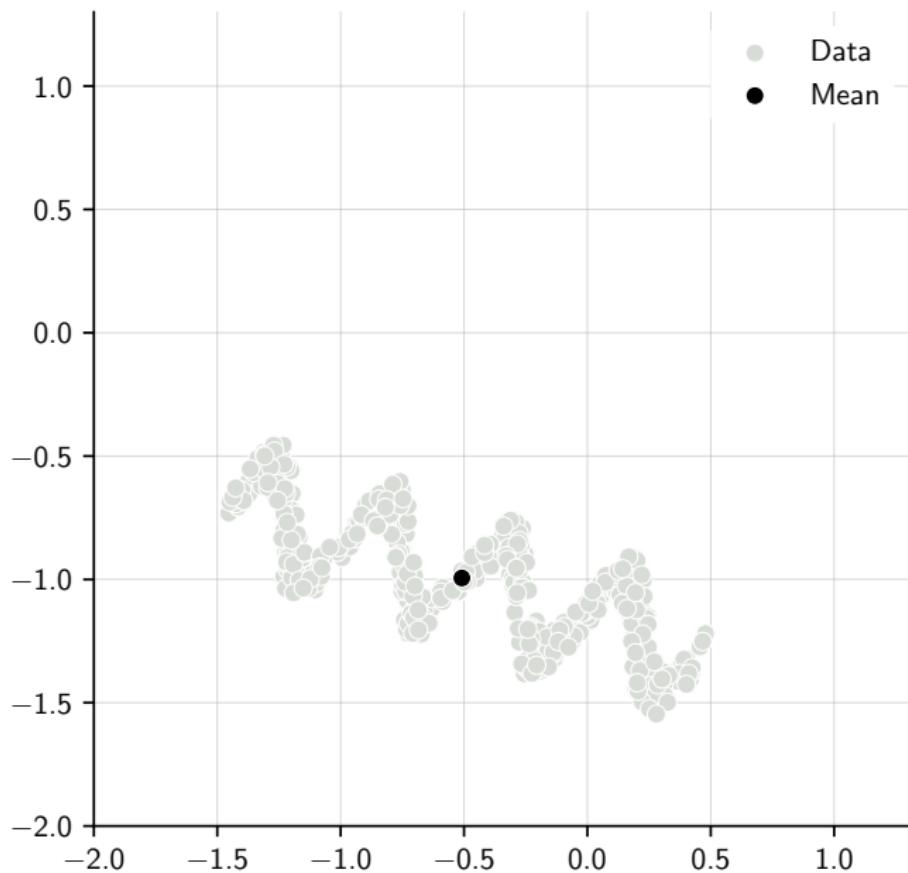
$$\mathbf{X} = \begin{pmatrix} - & \mathbf{x}_1 & - \\ & \vdots & \\ - & \mathbf{x}_N & - \end{pmatrix}$$

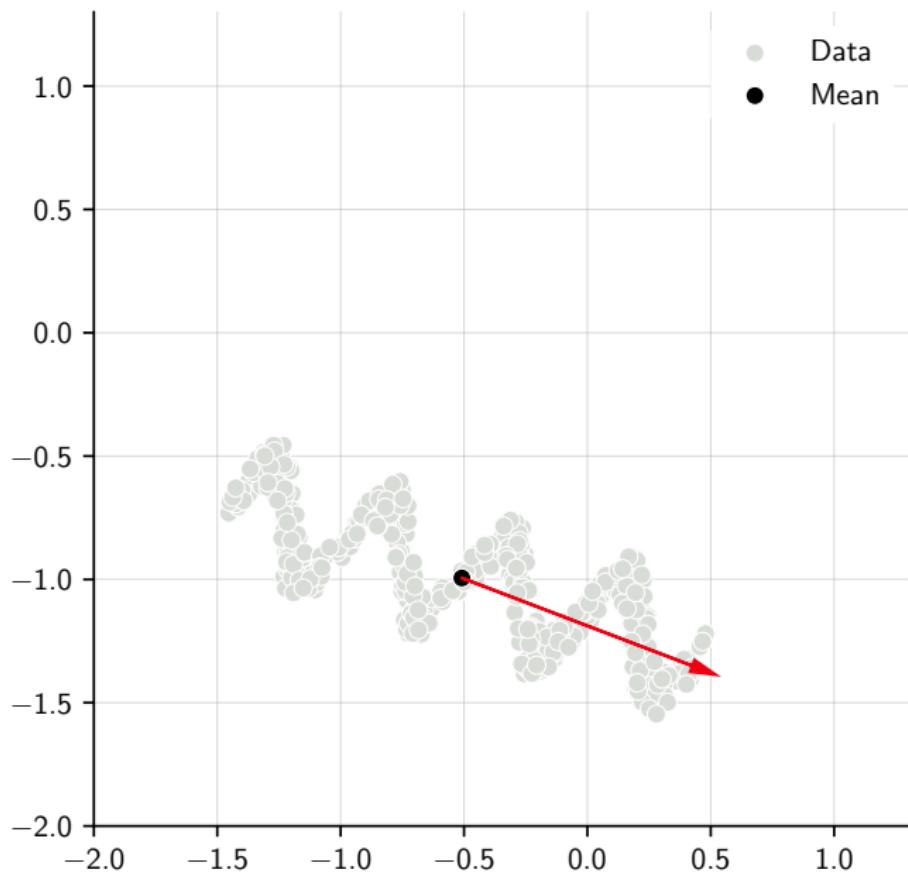
the centered data points, we have

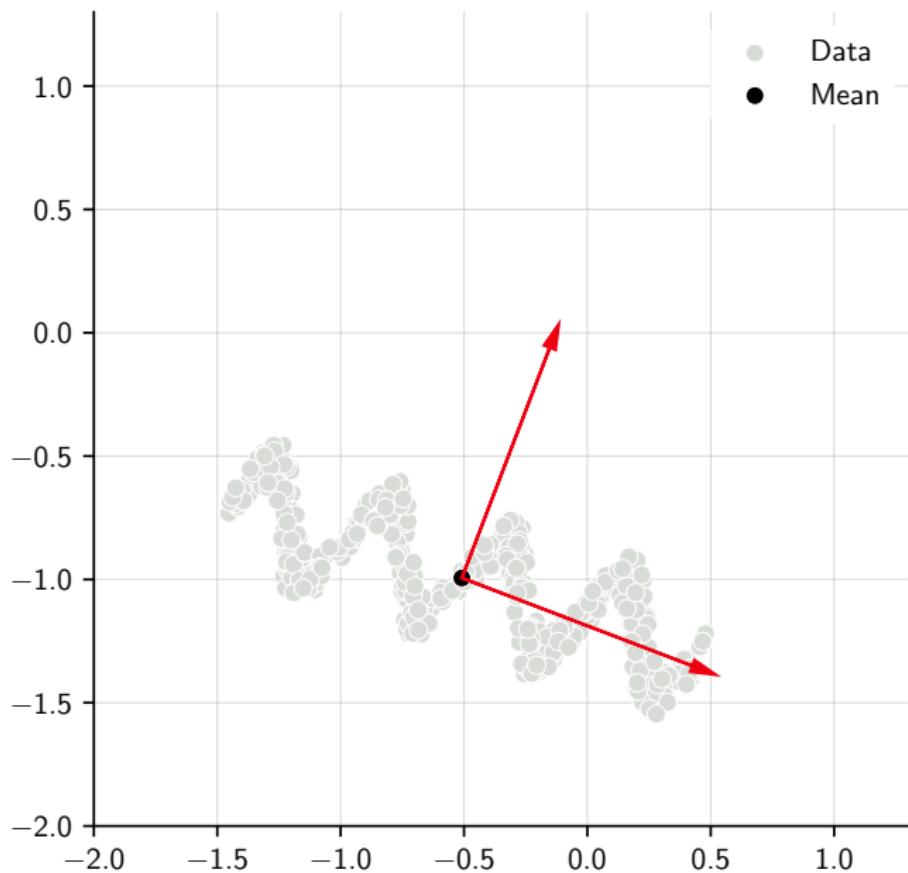
$$\begin{aligned}\sum_n (\mathbf{v} \cdot \mathbf{x}_n)^2 &= \left\| \begin{pmatrix} \mathbf{v} \cdot \mathbf{x}_1 \\ \vdots \\ \mathbf{v} \cdot \mathbf{x}_N \end{pmatrix} \right\|_2^2 \\ &= \left\| \mathbf{v} \mathbf{X}^\top \right\|_2^2 \\ &= (\mathbf{v} \mathbf{X}^\top) (\mathbf{v} \mathbf{X}^\top)^\top \\ &= \mathbf{v} (\mathbf{X}^\top \mathbf{X}) \mathbf{v}^\top.\end{aligned}$$

From this we can derive that $\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_D$ are the eigenvectors of $\mathbf{X}^\top \mathbf{X}$ ranked according to [the absolute values of] their eigenvalues.





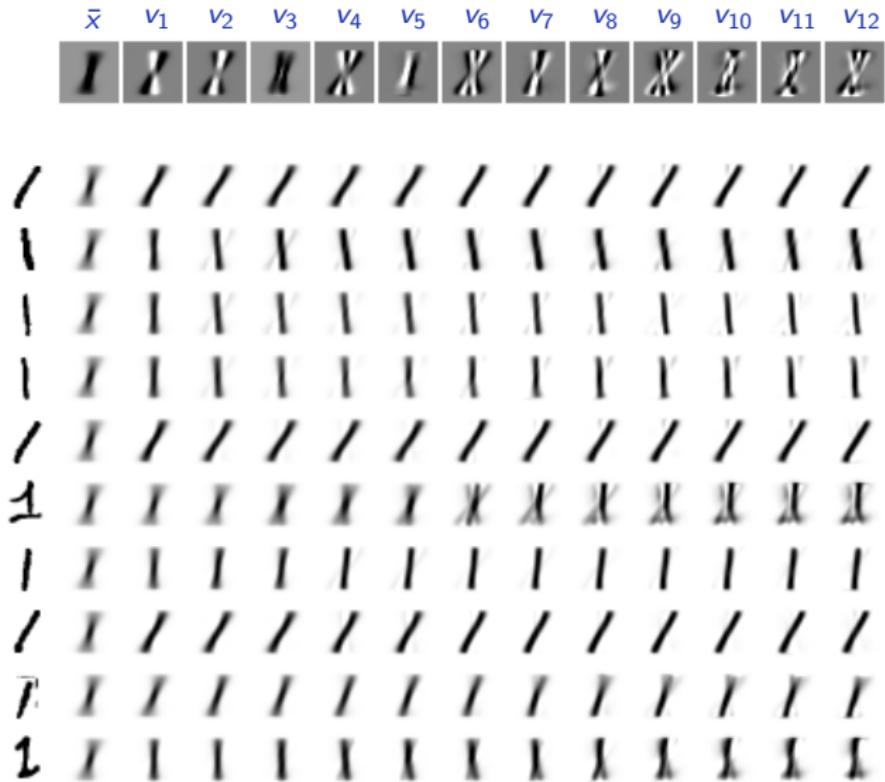




As for K -means, we can apply that algorithm to images from MNIST or CIFAR10 by considering them as vectors.

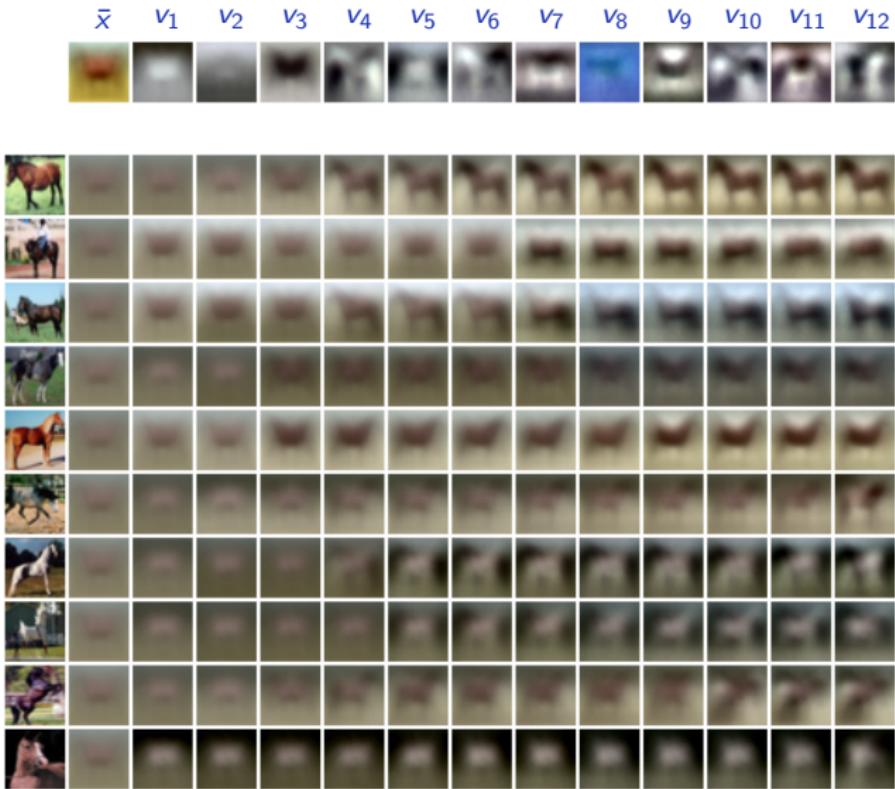
For any sample x and any T , we can compute a reconstruction using T vectors from the PCA basis, i.e.

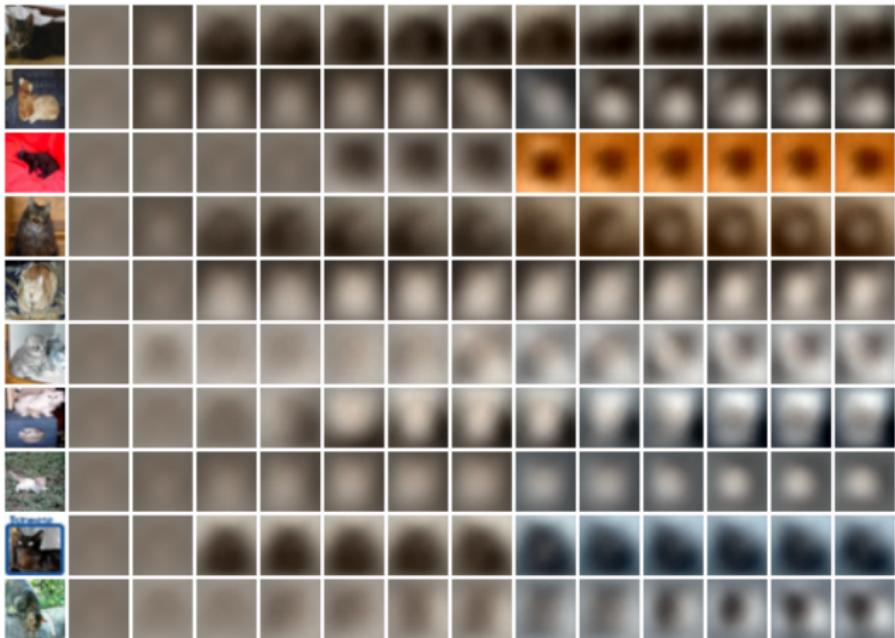
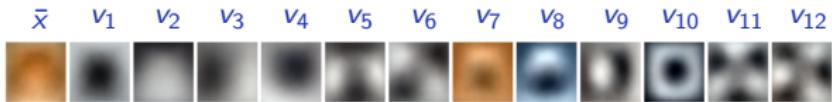
$$\bar{x} + \sum_{t=1}^T (v_t \cdot (x - \bar{x})) v_t.$$



\bar{x}	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8	v_9	v_{10}	v_{11}	v_{12}
9	9	9	9	9	9	9	9	9	9	9	9	9







These results show that even crude embeddings capture something meaningful. Changes in pixel intensity as expected, but also deformations in the “indexing” space (i.e. the image plan).

However, translations and deformations damage the representation badly, and “composition” (e.g. object on background) is not handled at all.

These strengths and shortcomings provide an intuitive motivation for “deep neural networks”, and the rest of this course.

We would like

- to use many encoding “of these sorts” for small local structures with limited variability,
- have different “channels” for different components,
- process at multiple scales.

Computationally, we would like to deal with large signals and large training sets, so we need to avoid super-linear cost in one or the other.

The end