EE-559 – Deep learning

2.2. Over and under fitting

François Fleuret

https://fleuret.org/ee559/

Sun Nov 25 20:12:22 UTC 2018

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

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 he makes a perfect score?

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 he makes a perfect score?

What about interviewing ten candidates and picking the best?

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 he makes a perfect score?

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

With

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

independent standing for "candidate n answere question k correctly", we have

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

and

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

With

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

independent standing for "candidate n answere question k correctly", we have

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

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, one will score perfectly.

With

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

independent standing for "candidate n answere question k correctly", we have

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

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, 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."

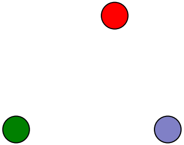
Given

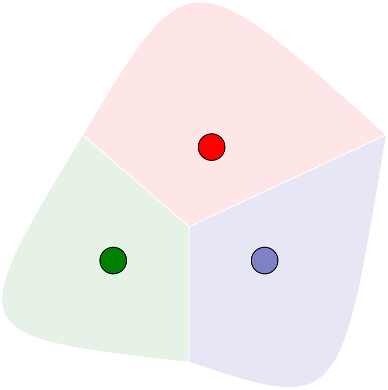
$$(x_n, y_n) \in \mathbb{R}^D \times \{1, \dots, C\},\ n = 1, \dots, N$$

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

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

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.

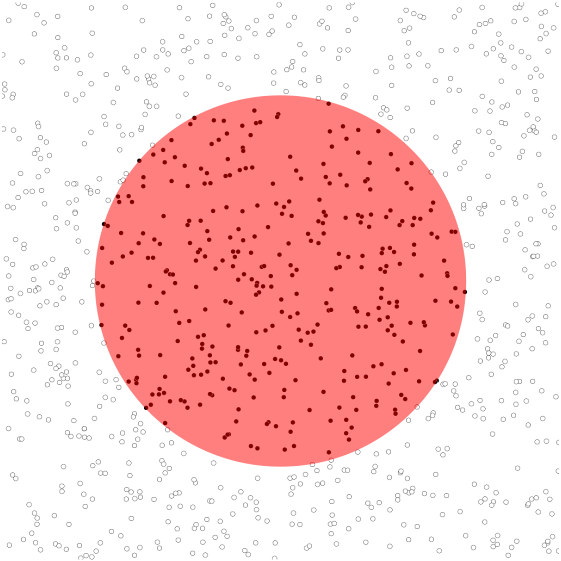
Under mild assumptions of regularities of $\mu_{X,Y}$, for $N \rightarrow \infty$ the asymptotic error rate of the 1-NN is less than twice the (optimal!) Bayes' Error rate.

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

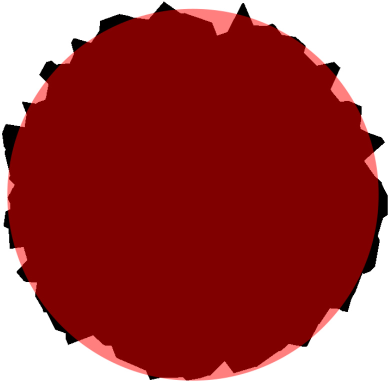
Under mild assumptions of regularities of $\mu_{X,Y}$, for $N \rightarrow \infty$ the asymptotic error rate of the 1-NN is less than twice the (optimal!) Bayes' Error rate.

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

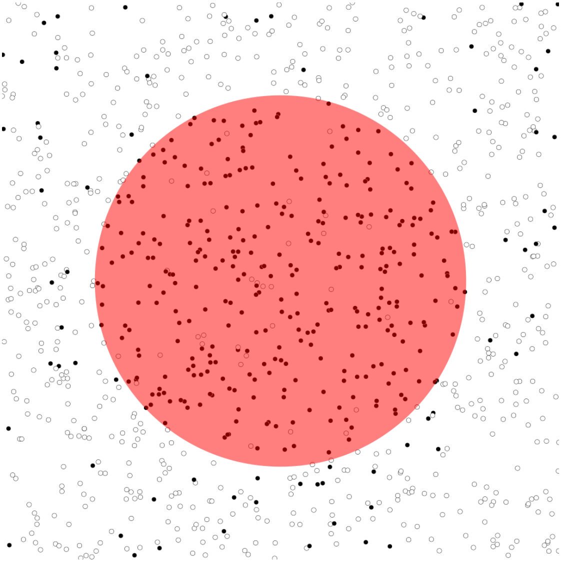
If we let also $K \rightarrow \infty$ "not too fast", the error rate is the (optimal!) Bayes' Error rate.



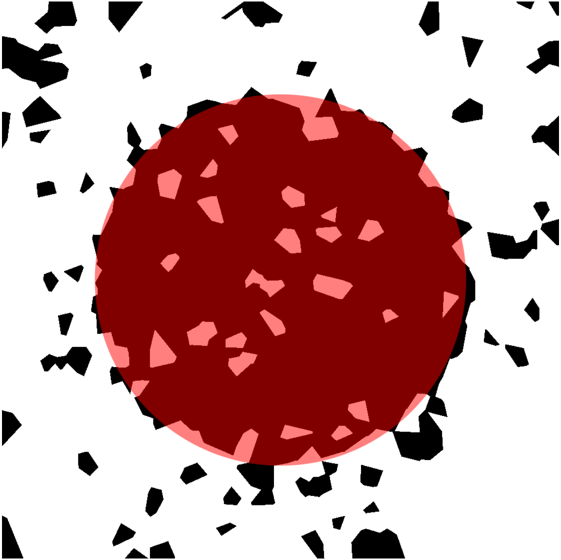
Training set



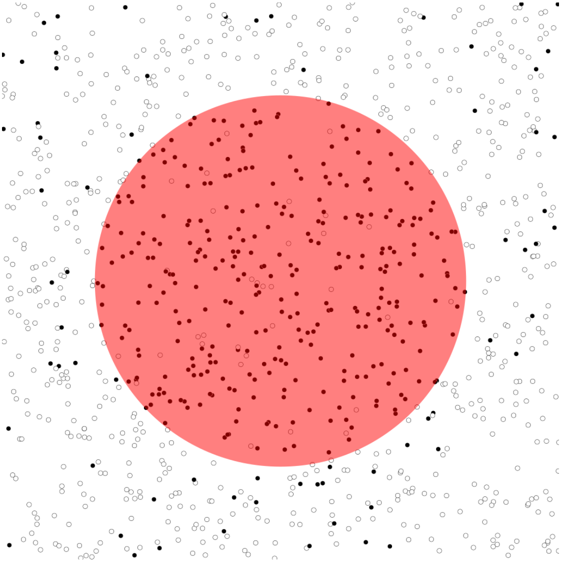
Prediction (K=1)



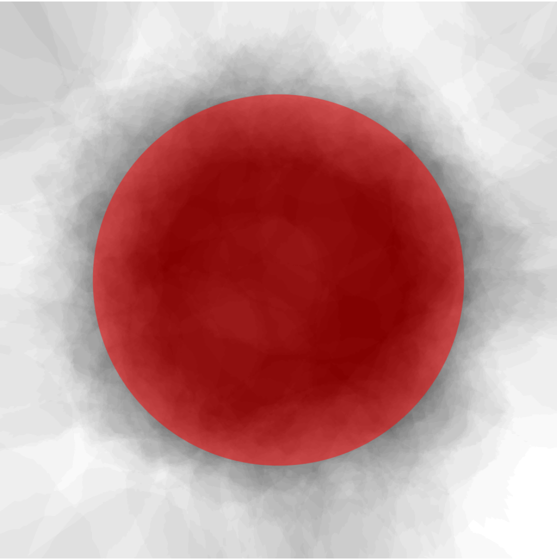
Training set



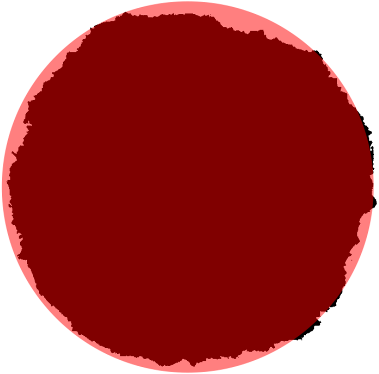
Prediction (K=1)



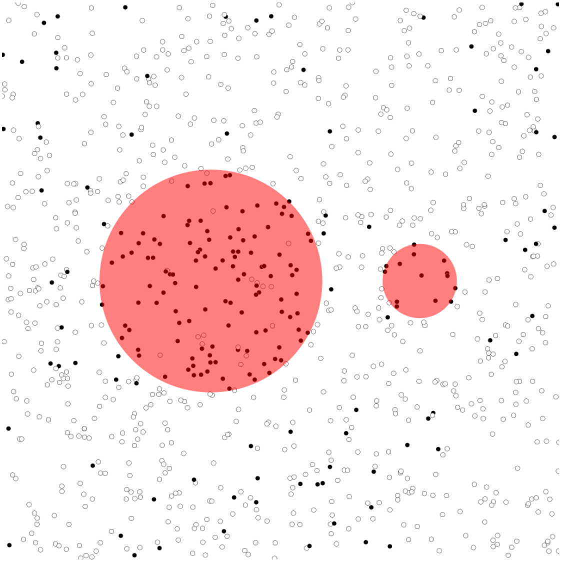
Training set



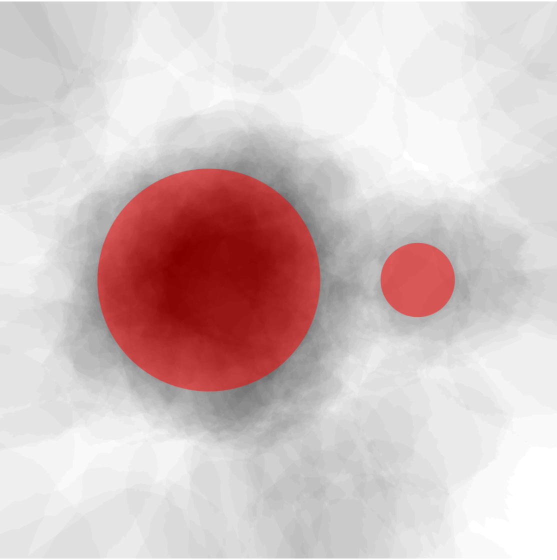
Votes (K=51)



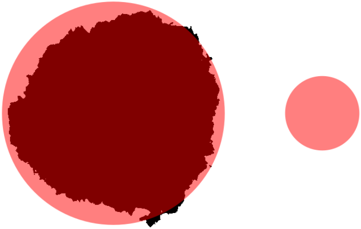
Prediction (K=51)



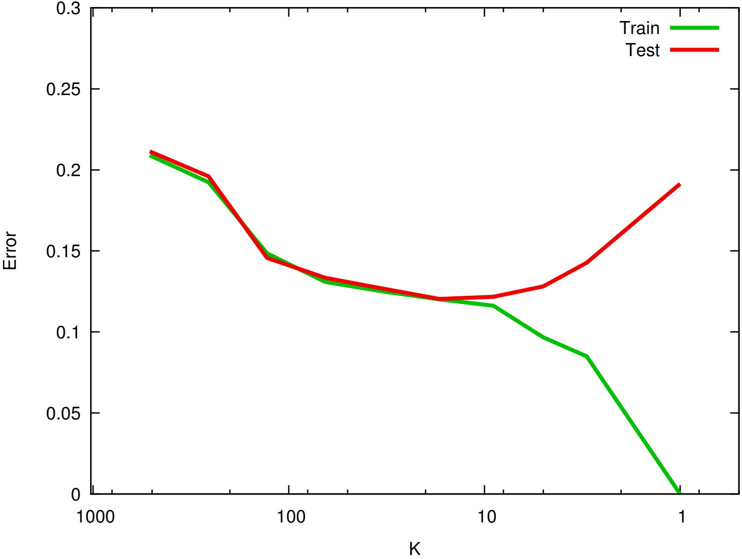
Training set

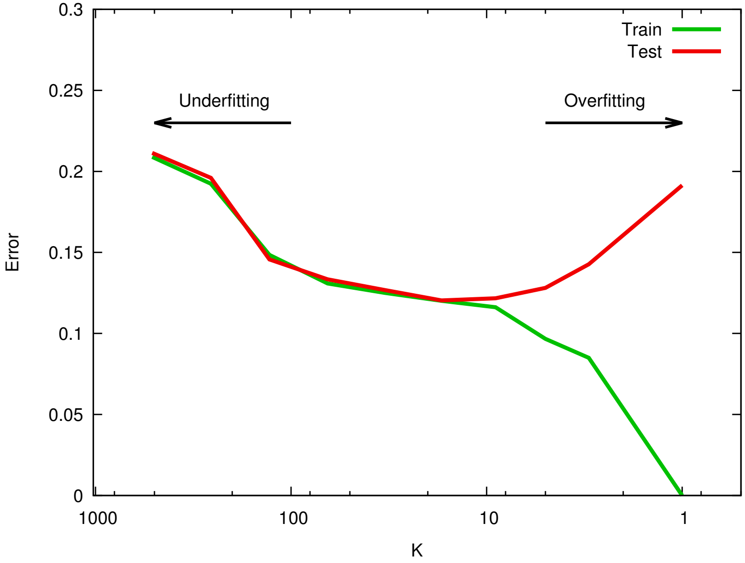


Votes (K=51)



Prediction (K=51)


Train
Test
Error
K
0.3
0.25
0.2
0.15
0.1
0.05
0
1000
100
10
1


Train
Test
Underfitting
Overfitting
Error
0.3
0.25
0.2
0.15
0.1
0.05
0
1000
100
10
1
K

Over and under-fitting, capacity, polynomials

Given a polynomial model

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

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

$$\mathscr{L}(\alpha) = \sum_n (f(x_n;\alpha) - y_n)^2$$

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

$$\begin{aligned}\mathscr{L}(\alpha) &= \sum_n \left(f(x_n;\alpha) - y_n\right)^2 \\ &= \sum_n \left(\sum_{d=0}^{D} \alpha_d x_n^d - y_n\right)^2\end{aligned}$$

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

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

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

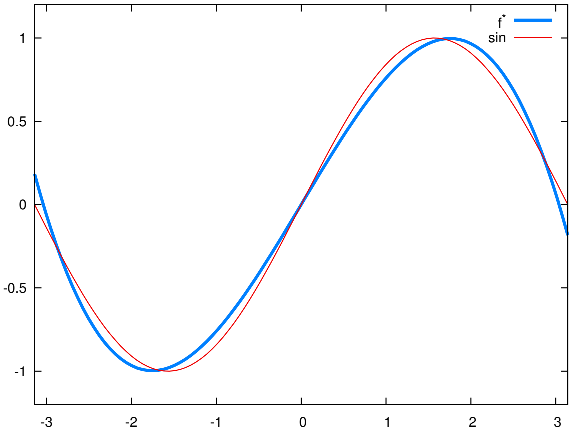
$$\begin{aligned}
\mathscr{L}(\alpha) &= \sum_n (f(x_n; \alpha) - y_n)^2 \\
&= \sum_n \left(\sum_{d=0}^{D} \alpha_d x_n^d - y_n \right)^2 \\
&= \left\| \begin{pmatrix} x_1^0 & \dots & x_1^D \\ \vdots & & \vdots \\ x_N^0 & \dots & x_N^D \end{pmatrix} \begin{pmatrix} \alpha_0 \\ \vdots \\ \alpha_D \end{pmatrix}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

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

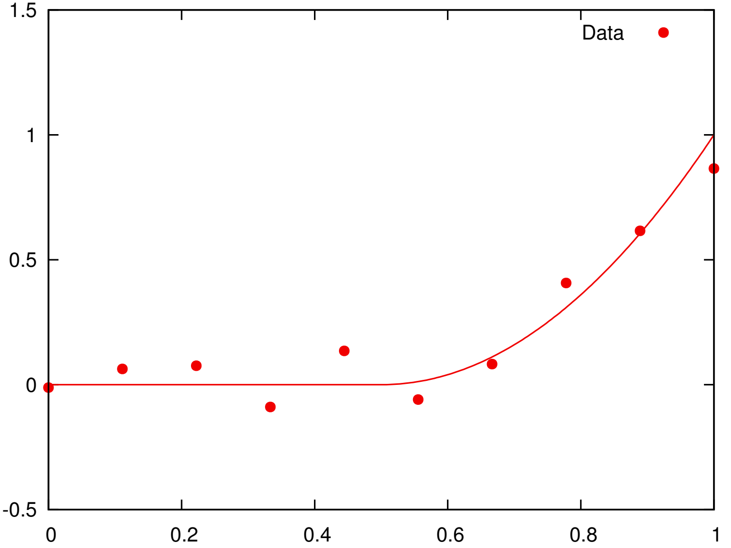
$$\underset{\alpha}{\operatorname{argmin}} \left\| \begin{pmatrix} x_1^0 & \dots & x_1^D \\ \vdots & & \vdots \\ x_N^0 & \dots & x_N^D \end{pmatrix} \begin{pmatrix} \alpha_0 \\ \vdots \\ \alpha_D \end{pmatrix} - \begin{pmatrix} y_1 \\ \vdots \\ y_N \end{pmatrix} \right\|^2$$

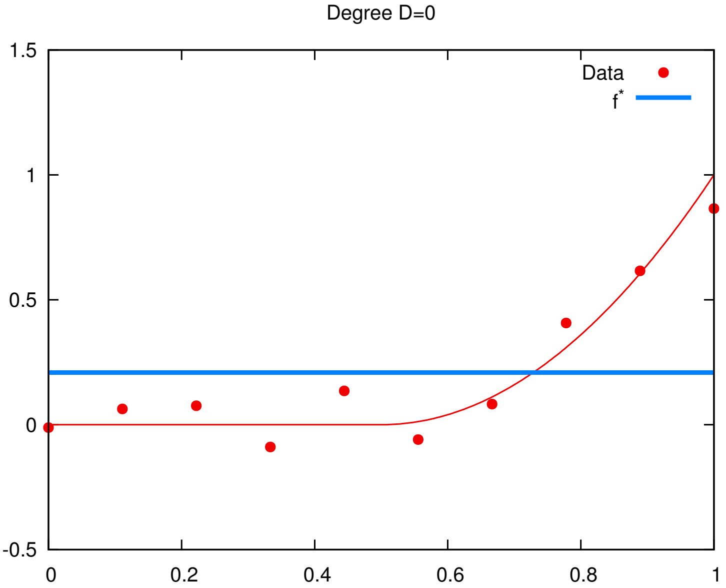
```
def fit_polynomial(D, x, y):
    X = torch.empty(x.size(0), D + 1)
    for d in range(D + 1):
        X[:, d] = x.pow(d)

    # gels expects a matrix for target
    Y = y.view(-1, 1)

    # LAPACK's GEneralized Least-Square
    alpha, _ = torch.gels(Y, X)

    return alpha[:D+1, 0]
```

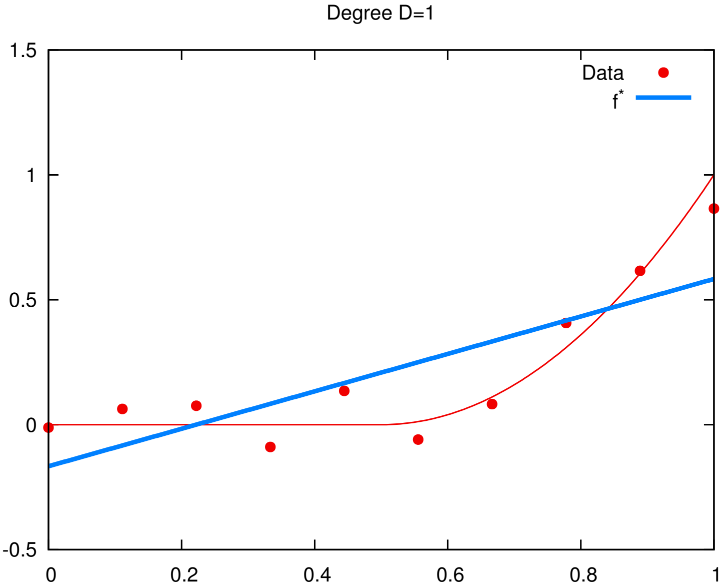
```
D, N = 4, 100
x = torch.linspace(-math.pi, math.pi, N)
y = x.sin()
alpha = fit_polynomial(D, x, y)

X = torch.empty(N, D + 1)
for d in range(D + 1):
    X[:, d] = x.pow(d)

yhat = X.mv(alpha)

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

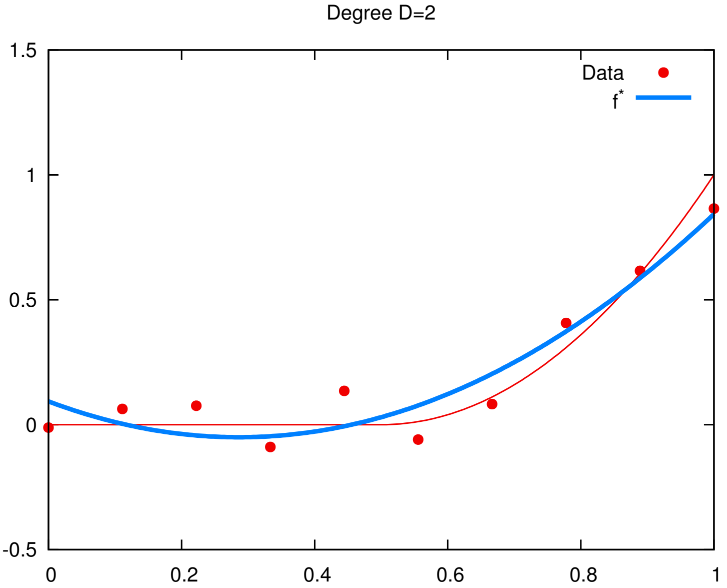


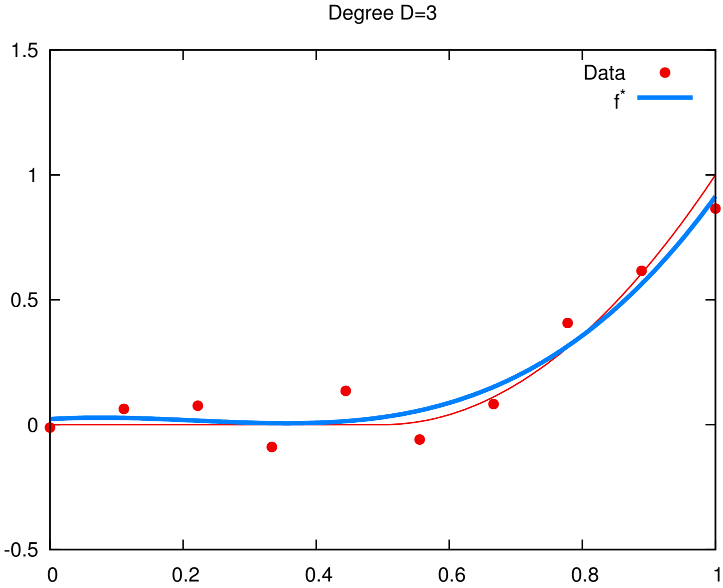
We can use that model to illustrate how the prediction changes when we increase the degree or the regularization.

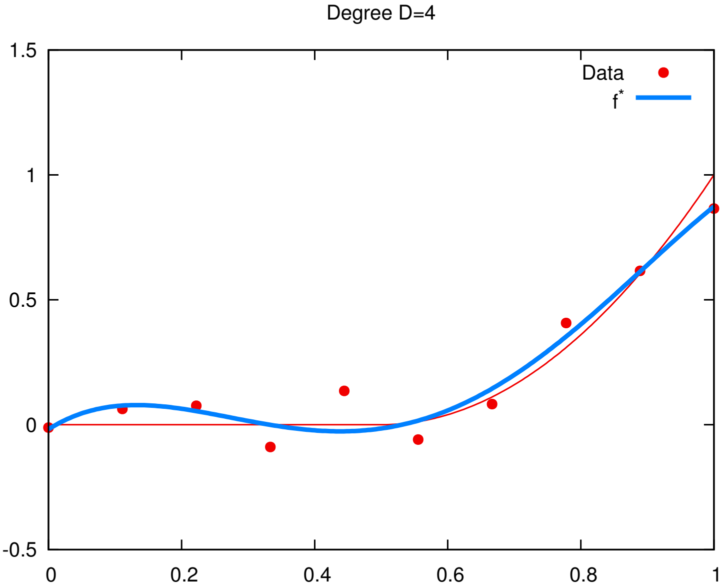

Data ●
1.5
1
0.5
0
-0.5
0
0.2
0.4
0.6
0.8
1

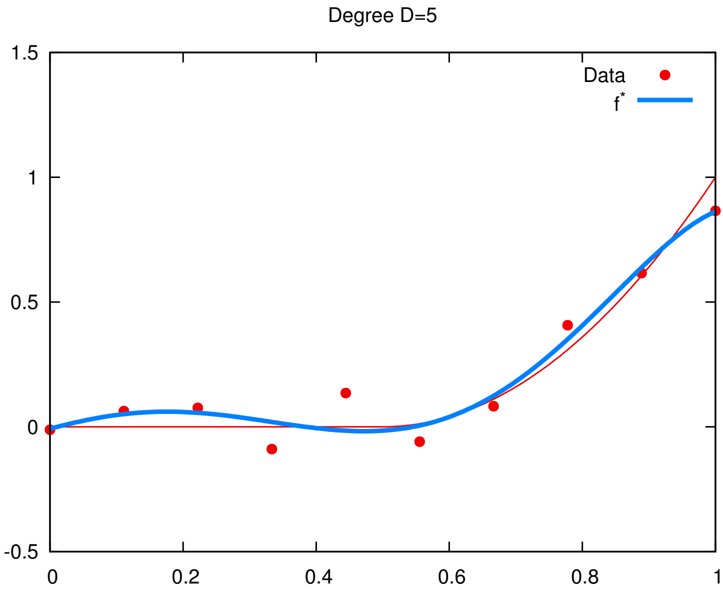

Degree D=0
Data ●
f*
1.5
1
0.5
0
-0.5
0
0.2
0.4
0.6
0.8
1

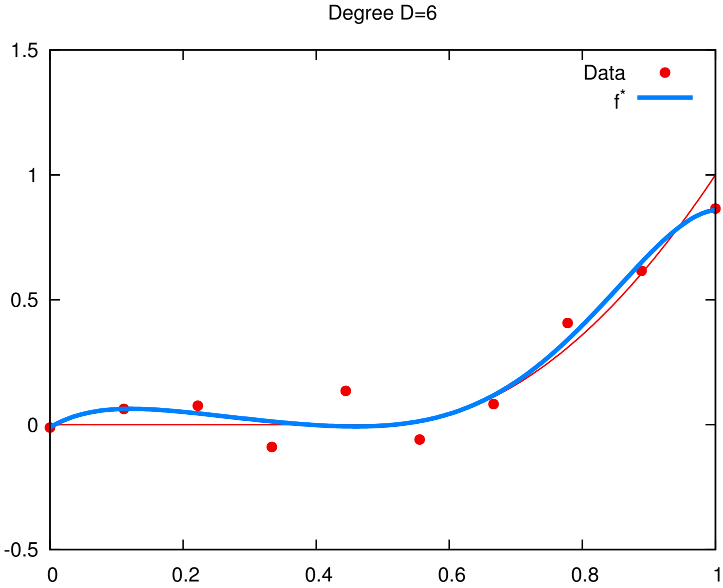

Degree D=1
Data
f*
1.5
1
0.5
0
-0.5
0
0.2
0.4
0.6
0.8
1

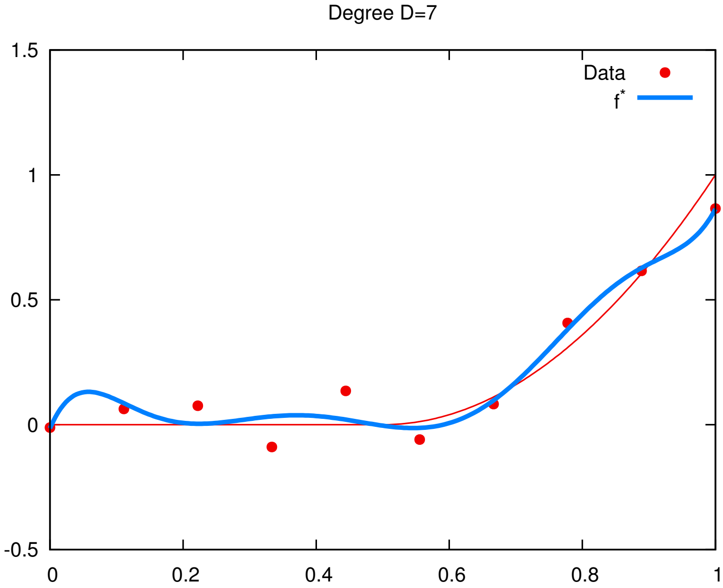

Degree D=2
Data
f*
1.5
1
0.5
0
-0.5
0
0.2
0.4
0.6
0.8
1

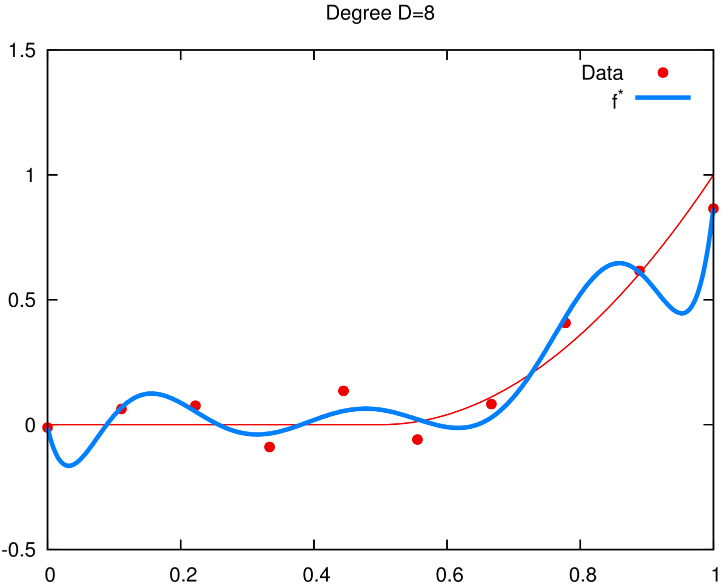

Degree D=3
Data ●
f^*
1.5
1
0.5
0
-0.5
0
0.2
0.4
0.6
0.8
1

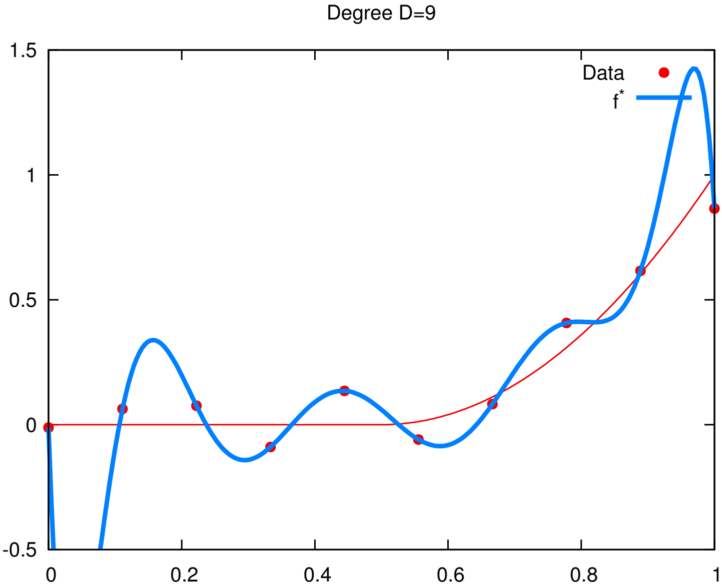

Degree D=4
Data
f^*
1.5
1
0.5
0
-0.5
0
0.2
0.4
0.6
0.8
1

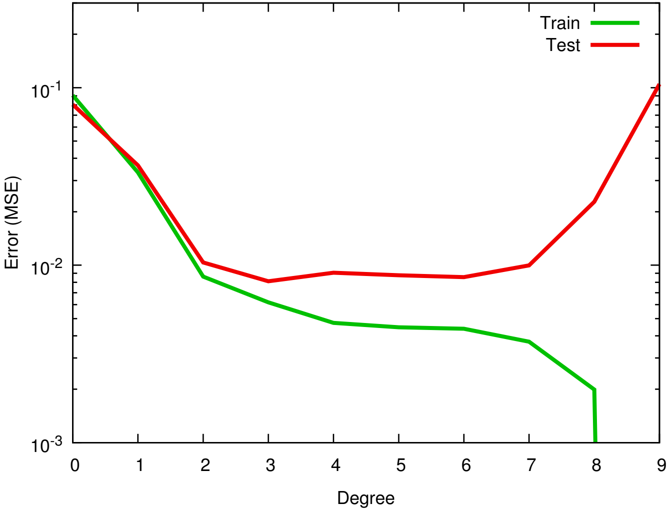

Degree D=5
Data
f*
1.5
1
0.5
0
-0.5
0
0.2
0.4
0.6
0.8
1

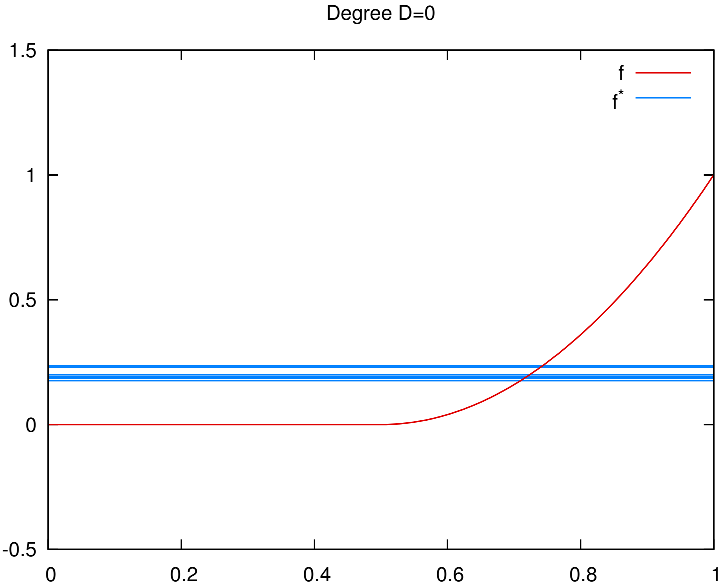

Degree D=6
Data
f^*
1.5
1
0.5
0
-0.5
0
0.2
0.4
0.6
0.8
1

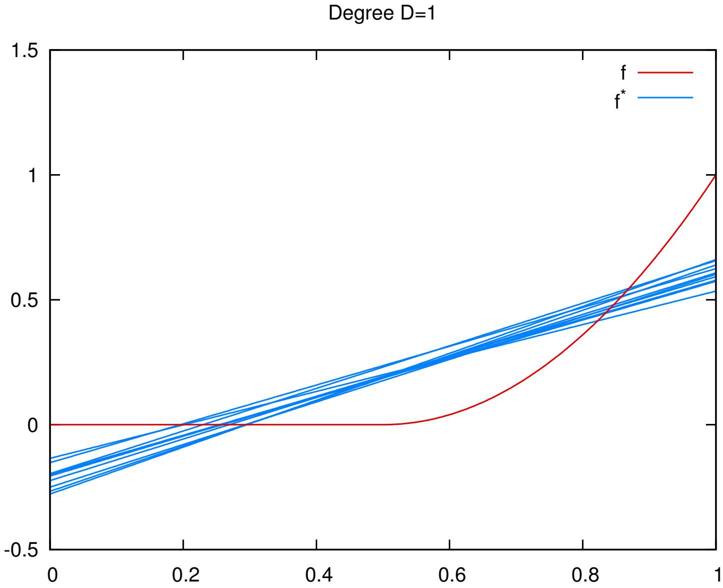

Degree D=7
Data
f^*
1.5
1
0.5
0
-0.5
0
0.2
0.4
0.6
0.8
1

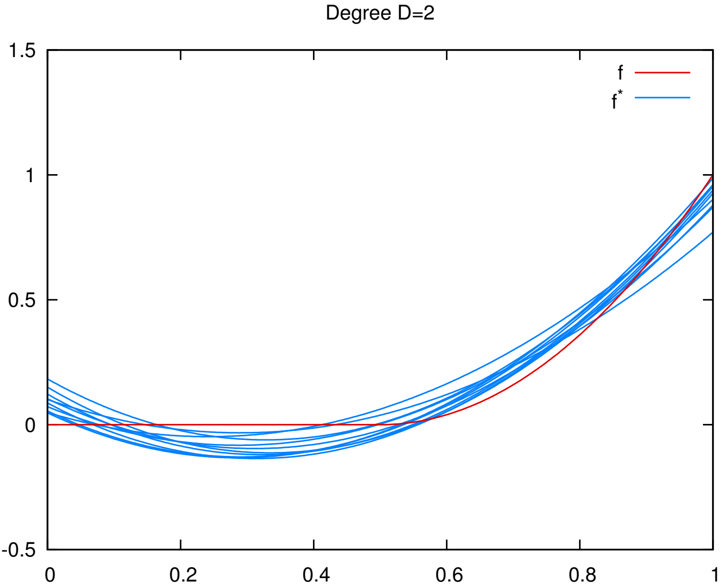

Degree D=8
Data
f*
1.5
1
0.5
0
-0.5
0
0.2
0.4
0.6
0.8
1

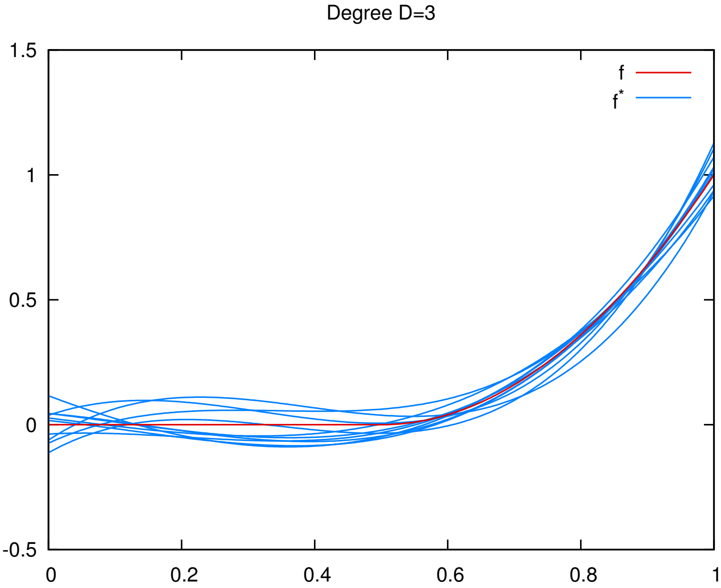

Degree D=9
Data
f*
1.5
1
0.5
0
-0.5
0
0.2
0.4
0.6
0.8
1

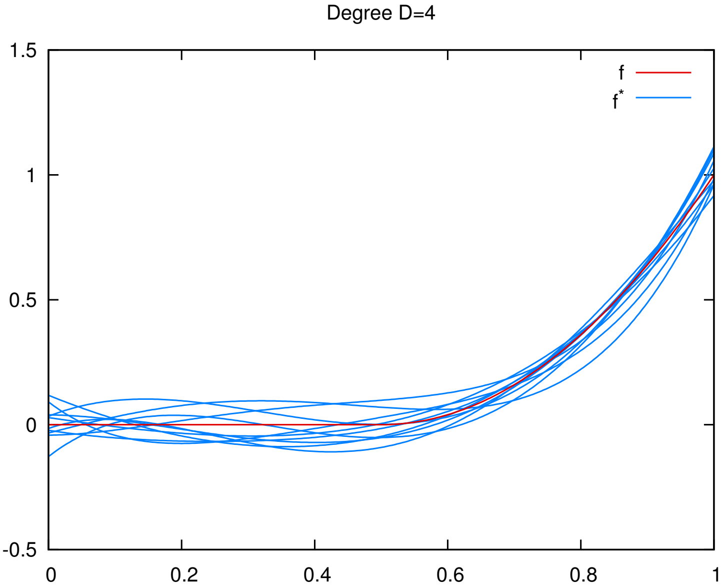

Train
Test
10^-1
10^-2
10^-3
Error (MSE)
0
1
2
3
4
5
6
7
8
9
Degree

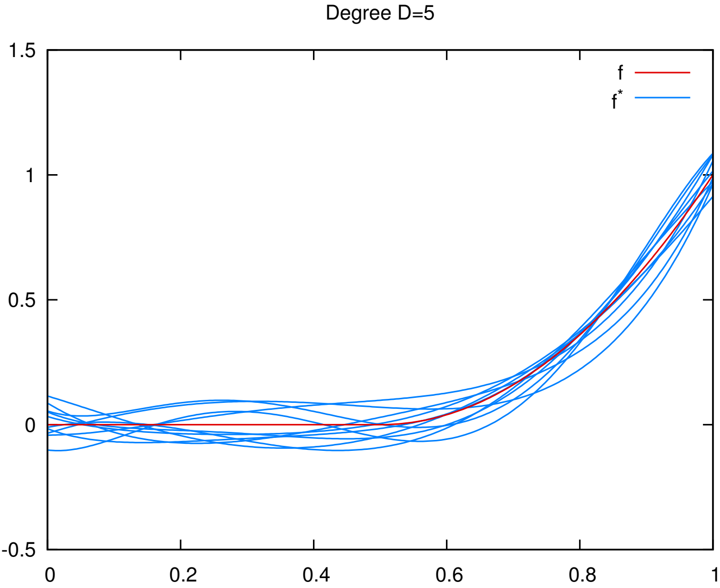

Degree D=0
f
f*
1.5
1
0.5
0
-0.5
0
0.2
0.4
0.6
0.8
1

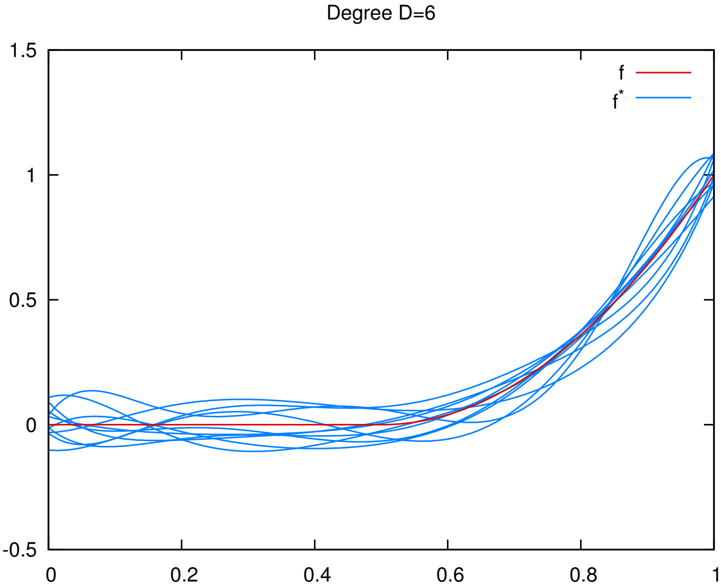

Degree D=1
f
f^*
1.5
1
0.5
0
-0.5
0
0.2
0.4
0.6
0.8
1

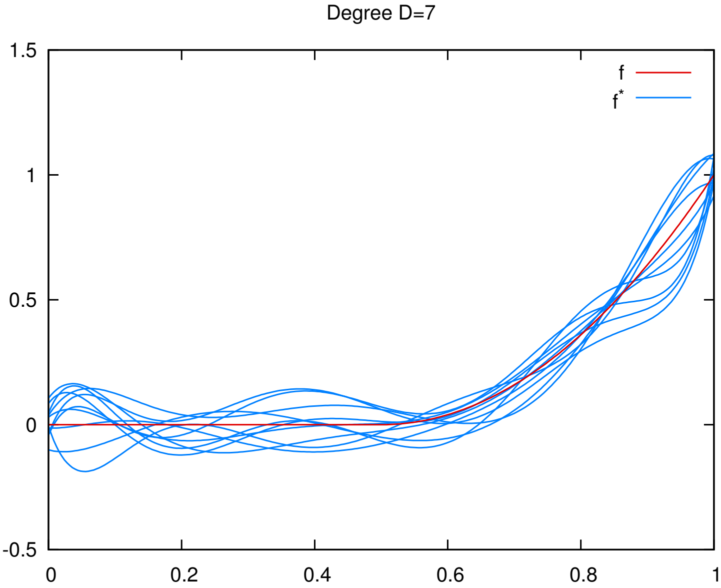

Degree D=2
f
f^*
1.5
1
0.5
0
-0.5
0
0.2
0.4
0.6
0.8
1

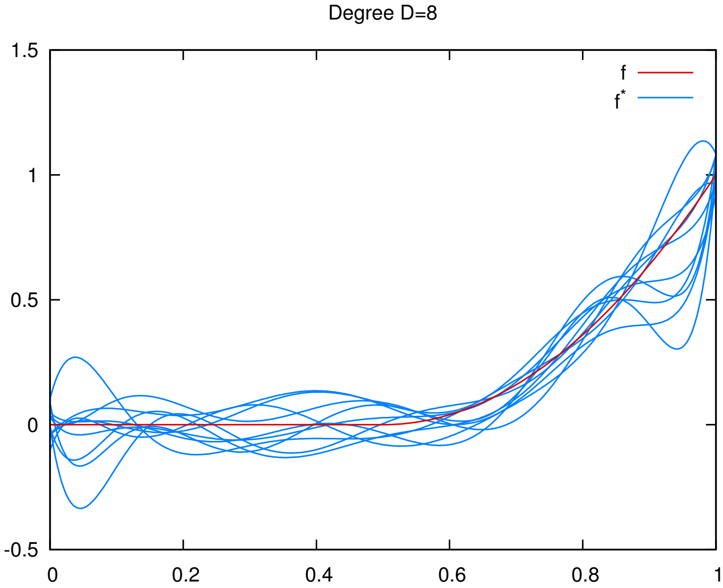

Degree D=3
f
f*
1.5
1
0.5
0
-0.5
0
0.2
0.4
0.6
0.8
1

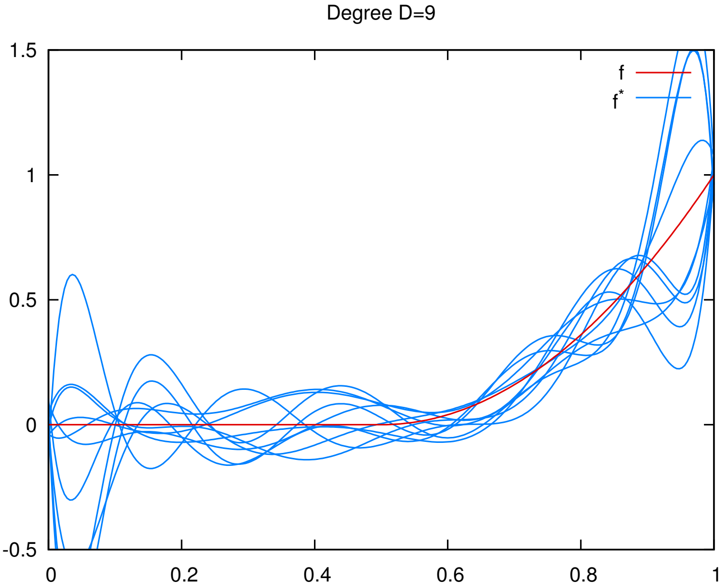

Degree D=4
f
f^*
1.5
1
0.5
0
-0.5
0
0.2
0.4
0.6
0.8
1


Degree D=5
f
f*
1.5
1
0.5
0
-0.5
0
0.2
0.4
0.6
0.8
1


Degree D=6
f
f*
1.5
1
0.5
0
-0.5
0
0.2
0.4
0.6
0.8
1


Degree D=7
f
f^*
1.5
1
0.5
0
-0.5
0
0.2
0.4
0.6
0.8
1


Degree D=8
f
f*
1.5
1
0.5
0
-0.5
0
0.2
0.4
0.6
0.8
1


Degree D=9
1.5
1
0.5
0
-0.5
0
0.2
0.4
0.6
0.8
1
f
f^*

We can reformulate this control of the degree with a penalty

$$\mathscr{L}(\alpha) = \sum_n \left(f(x_n;\alpha) - y_n\right)^2 + \sum_d l_d(\alpha_d)$$

where

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

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

We can reformulate this control of the degree with a penalty

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

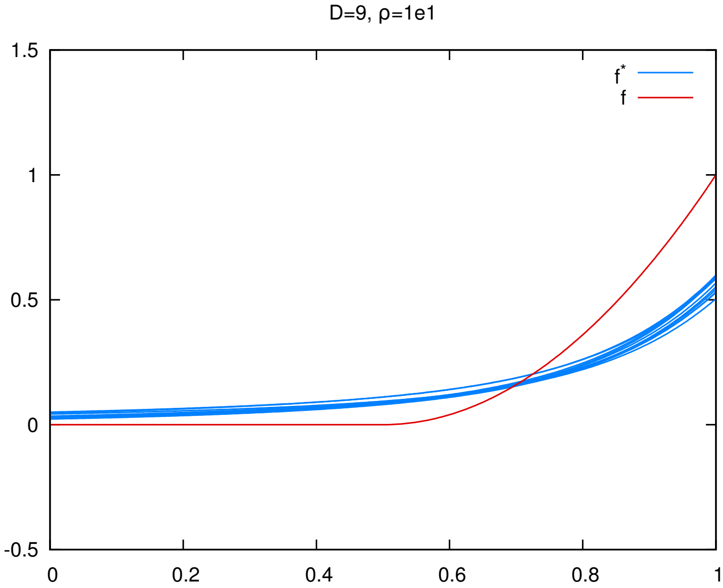
where

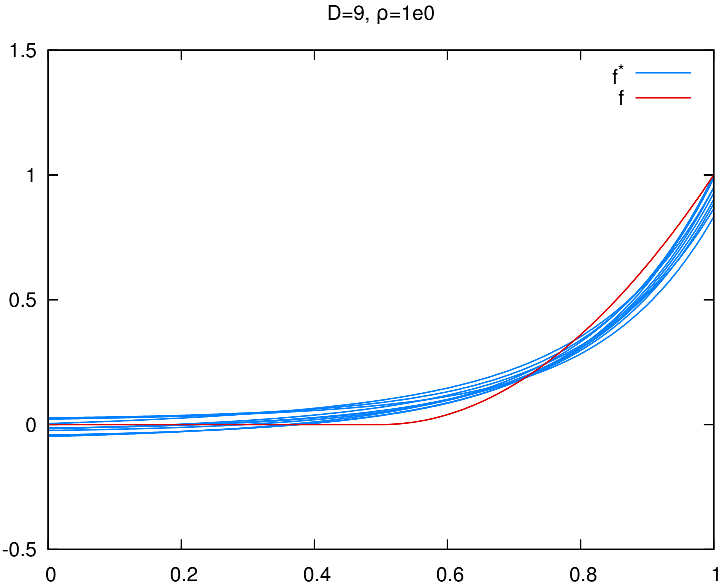
$$l_d(\alpha) = \begin{cases} 0 & \text{if } d \leq D \text{ or } \alpha = 0 \\ +\infty & \text{otherwise.} \end{cases}$$

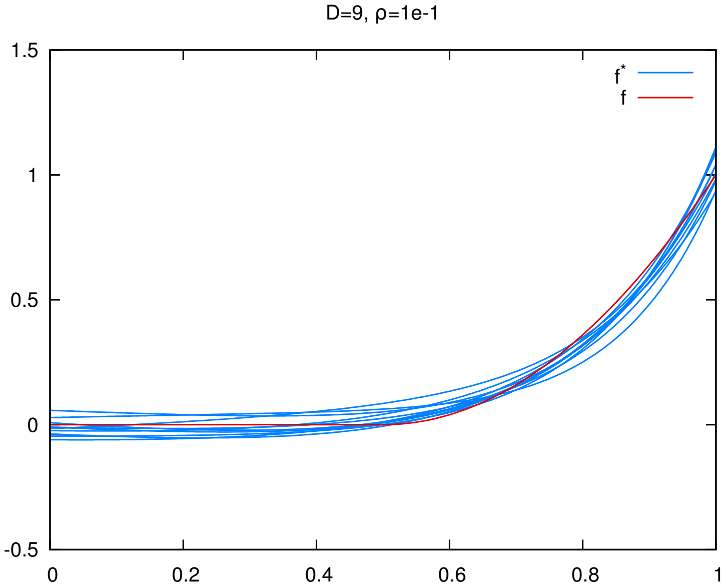
Such a penalty kills any term of degree $> D$.

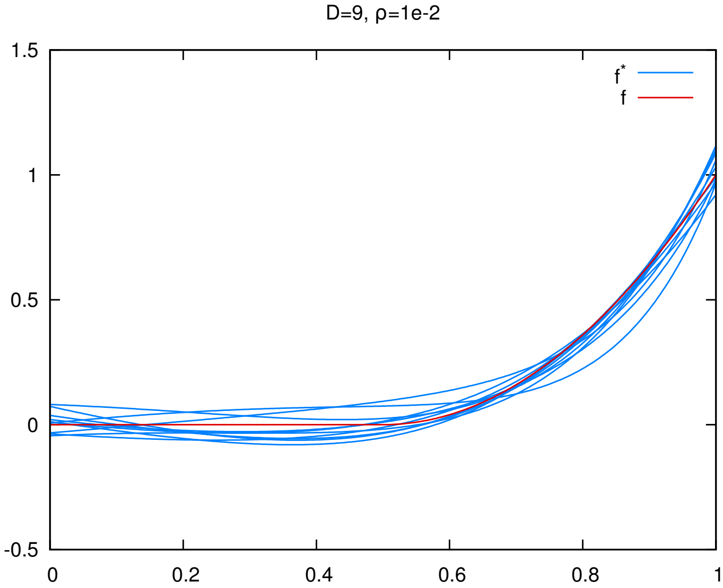
This motivates the use of more subtle variants. For instance, to keep all this quadratic

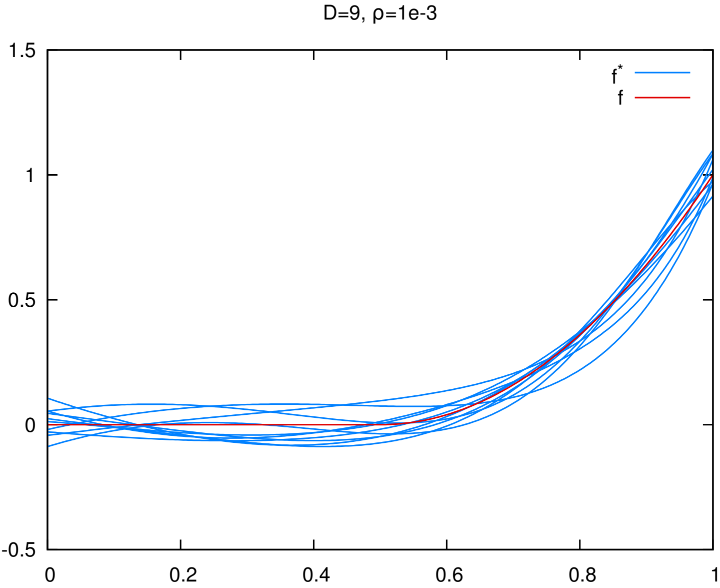
$$\mathscr{L}(\alpha) = \sum_n (f(x_n; \alpha) - y_n)^2 + \rho \sum_d \alpha_d^2.$$

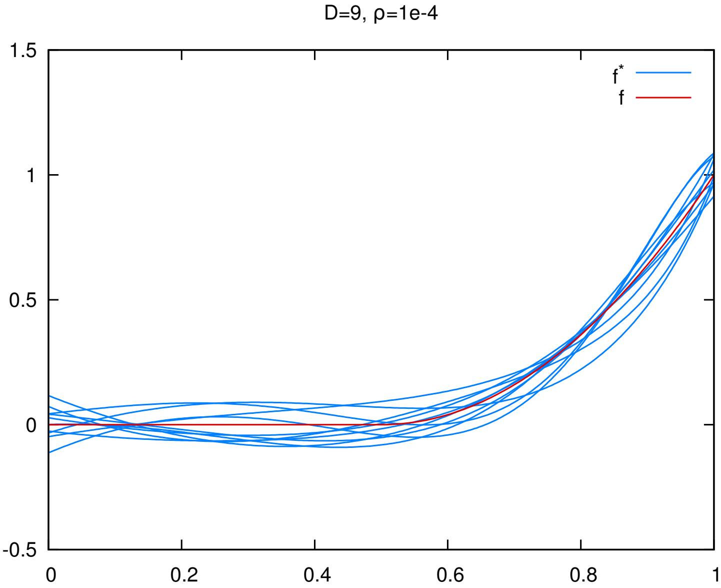

D=9, ρ=1e1
1.5
1
0.5
0
-0.5
0
0.2
0.4
0.6
0.8
1
f^*
f

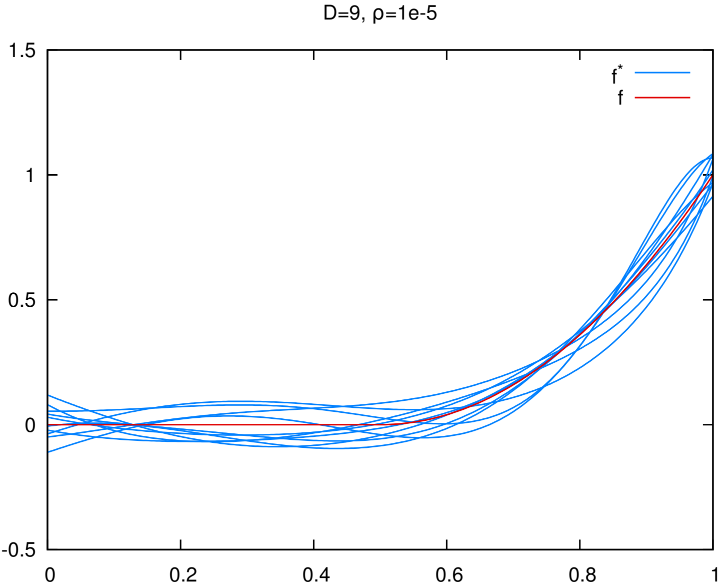

D=9, ρ=1e0
f*
f
1.5
1
0.5
0
-0.5
0
0.2
0.4
0.6
0.8
1

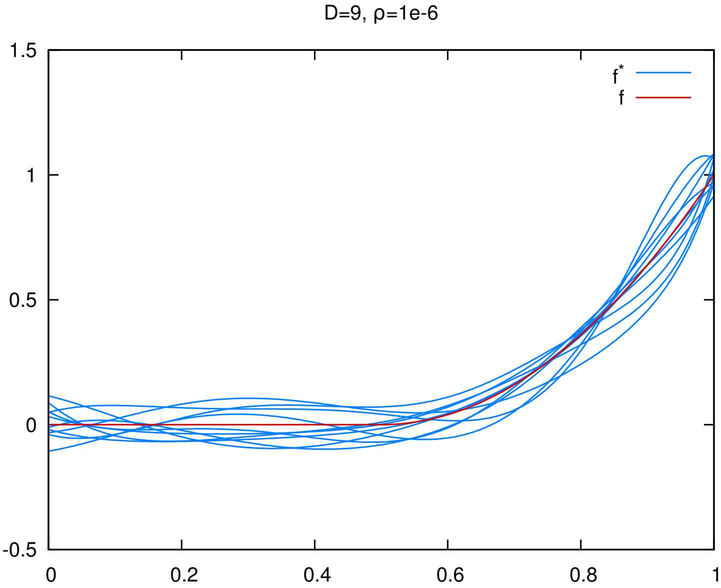

D=9, ρ=1e-1
1.5
1
0.5
0
-0.5
0
0.2
0.4
0.6
0.8
1
f^*
f

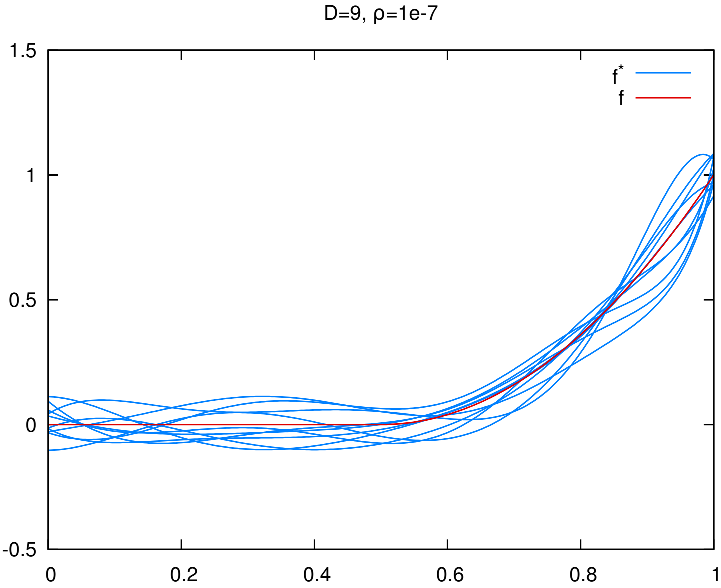

D=9, ρ=1e-2
1.5
1
0.5
0
-0.5
0
0.2
0.4
0.6
0.8
1
f*
f

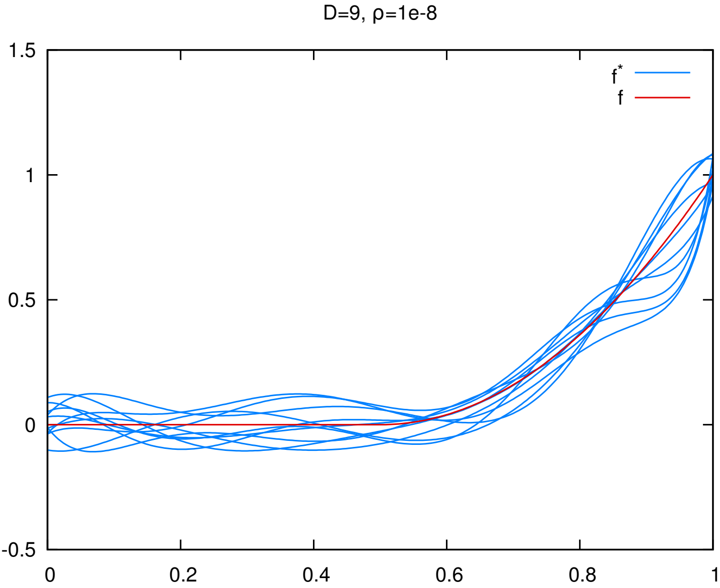

D=9, ρ=1e-3
f^*
f
1.5
1
0.5
0
-0.5
0
0.2
0.4
0.6
0.8
1

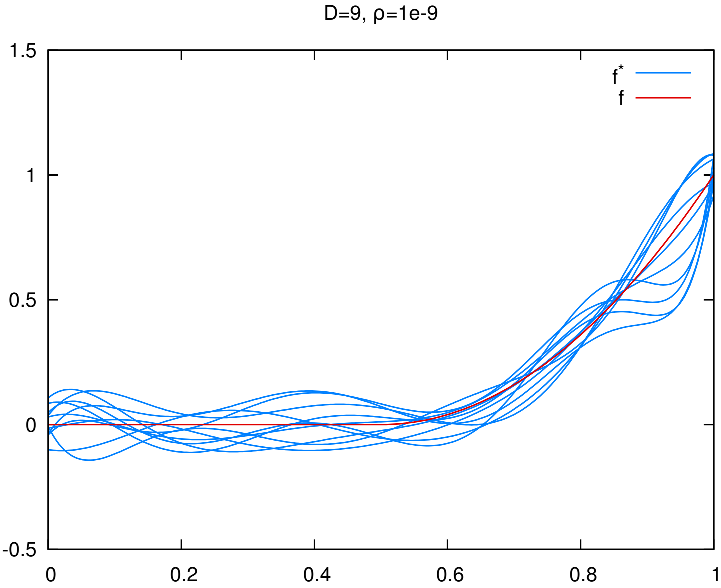

D=9, ρ=1e-4
1.5
1
0.5
0
-0.5
0
0.2
0.4
0.6
0.8
1
f*
f

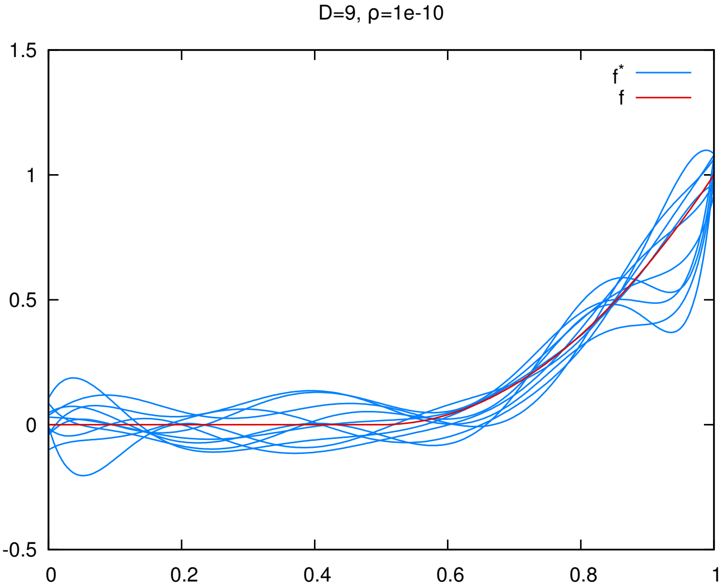

D=9, ρ=1e-5
1.5
1
0.5
0
-0.5
0
0.2
0.4
0.6
0.8
1
f*
f

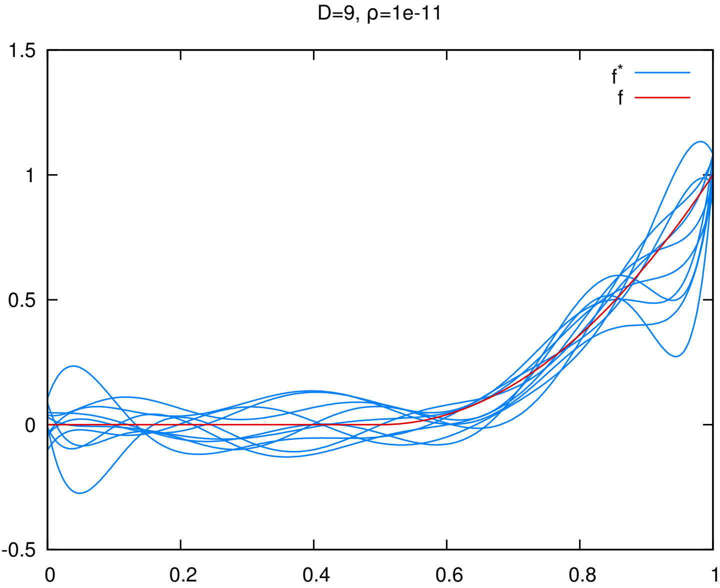

D=9, ρ=1e-6
f^*
f
1.5
1
0.5
0
-0.5
0
0.2
0.4
0.6
0.8
1

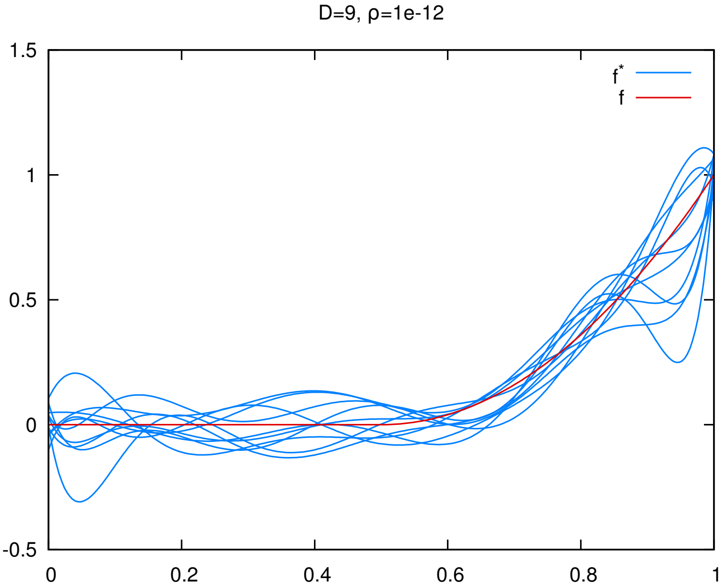

D=9, ρ=1e-7
f*
f
1.5
1
0.5
0
-0.5
0
0.2
0.4
0.6
0.8
1

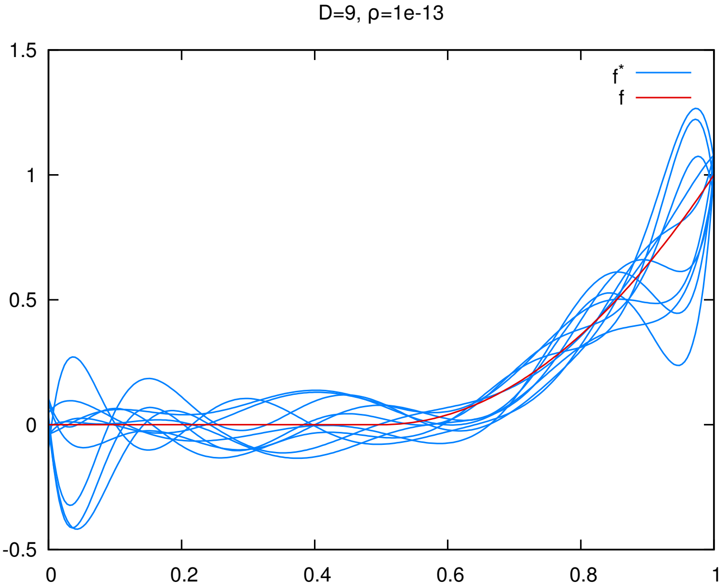

D=9, ρ=1e-8
f*
f
1.5
1
0.5
0
-0.5
0
0.2
0.4
0.6
0.8
1

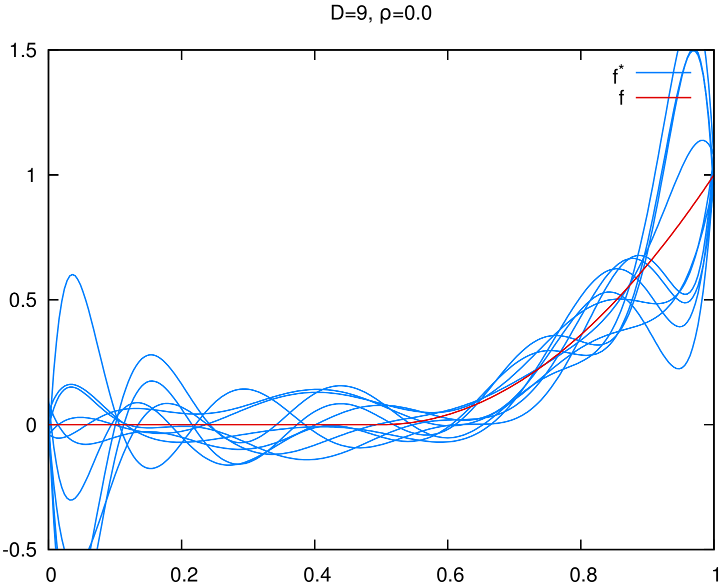

D=9, ρ=1e-9
1.5
1
0.5
0
-0.5
0
0.2
0.4
0.6
0.8
1
f*
f

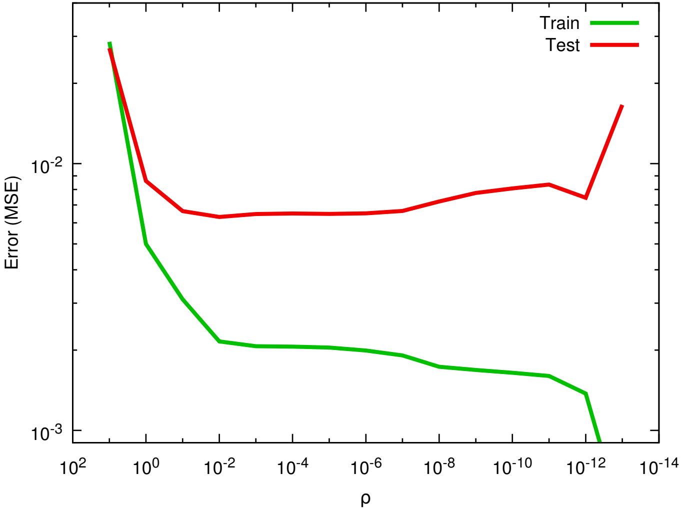

D=9, ρ=1e-10
f*
f
1.5
1
0.5
0
-0.5
0
0.2
0.4
0.6
0.8
1


D=9, ρ=1e-11
1.5
1
0.5
0
-0.5
0
0.2
0.4
0.6
0.8
1
f^*
f


D=9, ρ=1e-12
1.5
1
0.5
0
-0.5
0
0.2
0.4
0.6
0.8
1
f*
f


D=9, ρ=1e-13
f*
f
1.5
1
0.5
0
-0.5
0
0.2
0.4
0.6
0.8
1


D=9, ρ=0.0
f^*
f
1.5
1
0.5
0
-0.5
0
0.2
0.4
0.6
0.8
1


Train
Test
Error (MSE)
10^{-2}
10^{-3}
10^{2}
10^{0}
10^{-2}
10^{-4}
10^{-6}
10^{-8}
10^{-10}
10^{-12}
10^{-14}
ρ

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.

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

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

The end

References

V. N. Vapnik. *The Nature of Statistical Learning Theory*. Springer-Verlag, New York, 1995.