CSE-E4810 Machine Learning and Neural Networks (5 cr)

Lecture 6: Model Assessment and Selection

Prof. Juha Karhunen

https://mycourses.aalto.fi/course/view.php?id=13086

Introduction

- In this lecture, we discuss quite important general issues in supervised learning.
- Namely model assessment and selection based on training and test error, as well as generalization, overfitting, and regularization.
- We also present the bias-variance decomposition.
- These important matters are discussed also in our two other machine learning courses:
  - T-61.3050 Machine Learning: Basic Principles
  - T-61.5140 Machine Learning: Advanced Probabilistic Methods.
- The material for this lecture has been taken from selected portions of several books.

- The main source is the graduate-level book C. Bishop, "Pattern Recognition and Machine Learning", Springer 2006.
- In Du's and Swamy's book "Neural Networks and Statistical Learning", Springer 2014, these matters are discussed briefly in:
  - Section 2.2 Learning and Generalization;
  - Section 2.3 Model Selection;
  - Section 2.4 Bias and Variance.
- We present these matters somewhat more in detail on these slides.
- But derivations are omitted for keeping the presentation simpler.
- We skip in our course most of the rather scattered and mixed topics in the remainder of Chapter 2 of Du's and Swamy's book.

Example: polynomial curve fitting

The problem and training data

- The purpose of this example is to illustrate major problems in model selection and to motivate a number of key concepts.
- Suppose we observe a real-valued scalar input variable x.
- We wish to use it for predicting the value of a real-valued scalar target variable t.
- Note: In neural networks, t is the desired response d.
- It is instructive to consider an artificial example using synthetically generated data.
- Because we then know the precise model which generated the data for comparison against any learned model.

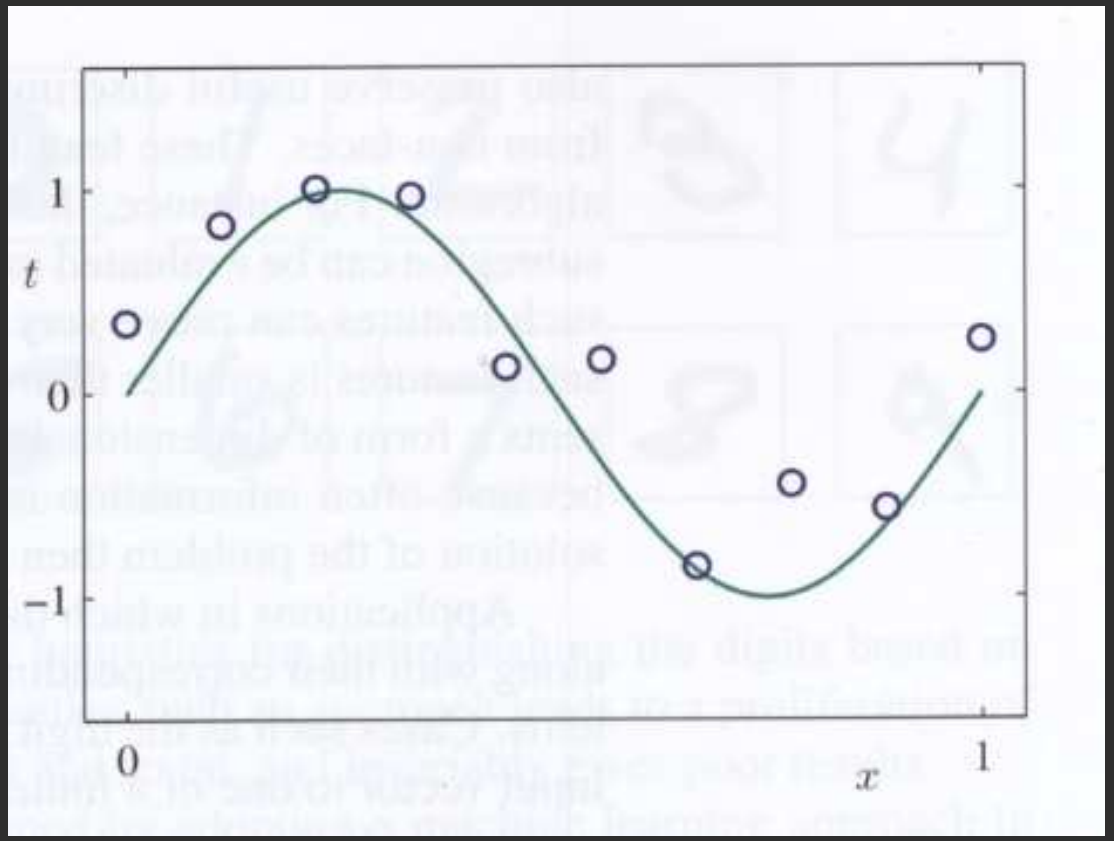

Figure 1: The example training set consisting of $N = 10$ measurements of the noisy sinusoid (1). The green curve shows the sinusoid $\sin(2\pi x_i)$.

- Suppose that we have a training set consisting of N input-target value pairs (x_i, t_i), $i = 1, 2, \ldots, N$.
- The inputs are collected to the input vector $\mathbf{x} = [x_1, x_2, \ldots, x_N]^T$.
- The corresponding target values are collected to the target vector $\mathbf{t} = [t_1, t_2, \ldots, t_N]^T$.
- Figure 1 shows a training set comprising $N = 10$ data points.
- The input values x_i are spaced uniformly in the interval $[0, 1]$.
- The corresponding target values t_i were generated from

$$t_i = \sin(2\pi x_i) + n_i \tag{1}$$

  where n_i is zero-mean Gaussian noise with standard deviation 0.3.
- That is, Figure 1 shows 10 samples of the noisy sinusoid (1).
- The data set generated in this way is similar to many real-world data

sets in that:

  - It has an underlying structure (regularity) that we wish to learn.
  - Individual observations are corrupted by random noise.

- The noise typically arises from inherent sources of variability that are themselves unobserved.
- Our goal is to exploit this training set in order to predict the value $\hat{t}$ of the target variable for some new value $\hat{x}$ of the input variable.
- Implicitly, we try to discover the function $\sin(2\pi x_i)$ underlying the data.
- This is intrinsically a difficult problem if we don't have any prior information on the data generation process (1).
- Small number of samples and noise make the problem even more difficult.

- In this example, we consider a simple curve fitting approach.
- For illustrating the arising problems, we shall fit the data using a polynomial function of the form

$$y(x, \mathbf{w}) = w_0 + w_1 x + w_2 x^2 + \ldots + w_M x^M = \sum_{j=0}^{M} w_j x^j \qquad (2)$$

- There M is the order of the polynomial, and x^j denotes x raised to power j.
- The vector $\mathbf{w} = [w_0, w_1, \ldots, w_M]^T$ contains the coefficients of the polynomial.
- Note that although $y(x, \mathbf{w})$ is a nonlinear function of x, it is a linear function of the coefficients $\mathbf{w}$.
- In general, models which are linear in the unknown parameters are called linear models.

The effect of model complexity

- For fitting the model(s) (2) to the training data, we use the familiar least-squares method.
- That is, we find such a coefficient vector $\mathbf{w}$ that minimizes the least-squares error function

$$E(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}[y(x_n, \mathbf{w}) - t_n]^2 \tag{3}$$

- For the linear model (2), the error function $E(\mathbf{w})$ is a quadratic function of the coefficients $\mathbf{w}$.
- Its derivatives with respect to the coefficients are linear functions in the elements of $\mathbf{w}$.
- And the resulting linear equations have a unique minimizing solution $\mathbf{w}^*$ which can be found in closed form.

- Provided that the number M of the coefficients is at most the number N of training pairs.
- See the exercises for details.
- There remains the problem of choosing the order M of the polynomial.
- This is an example of the important problem of model selection.
- Figure 2 shows four examples of the results of fitting polynomials to the data set of Figure 1.
- The polynomials fitted by the least-squares method have the orders $M = 0$, $M = 1$, $M = 3$, and $M = 9$.
- Note that because of the constant coefficient w_0, these polynomials have respectively 1, 2, 4, and 10 free parameters w_i.

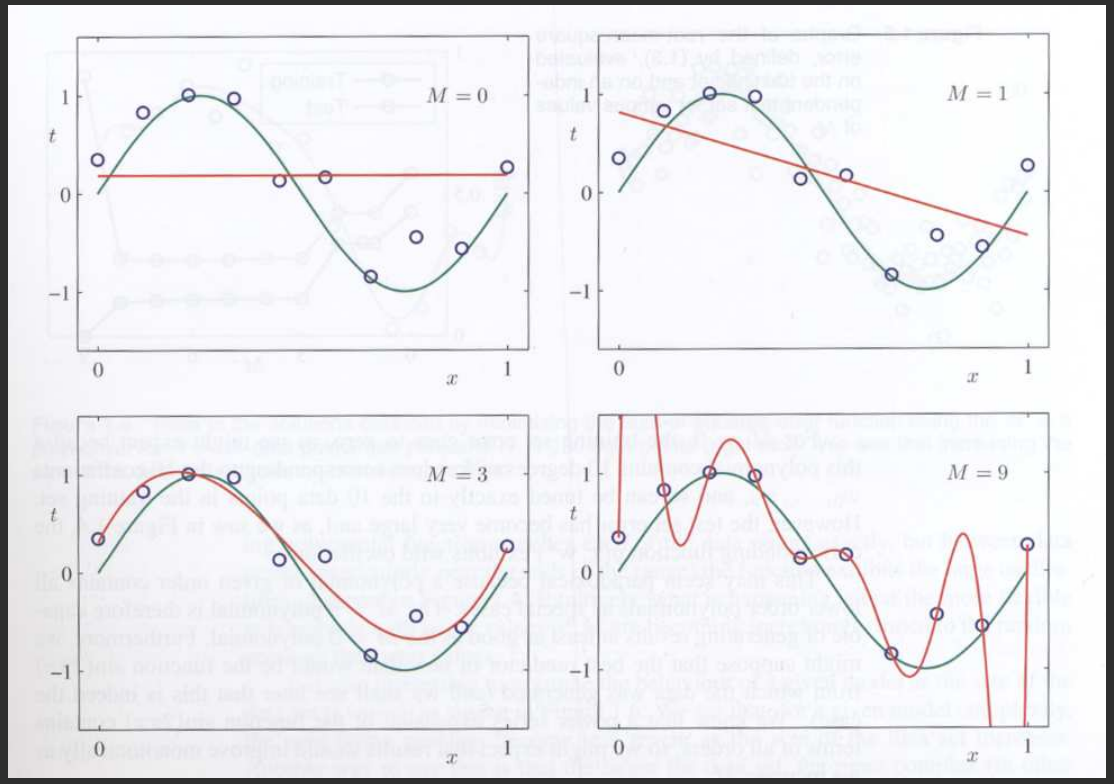

Figure 2: Plots of polynomials having various orders M (red curves) fitted to the data set of Figure 1. The green curve shows the true underlying sinusoid $\sin(2\pi x_i)$.

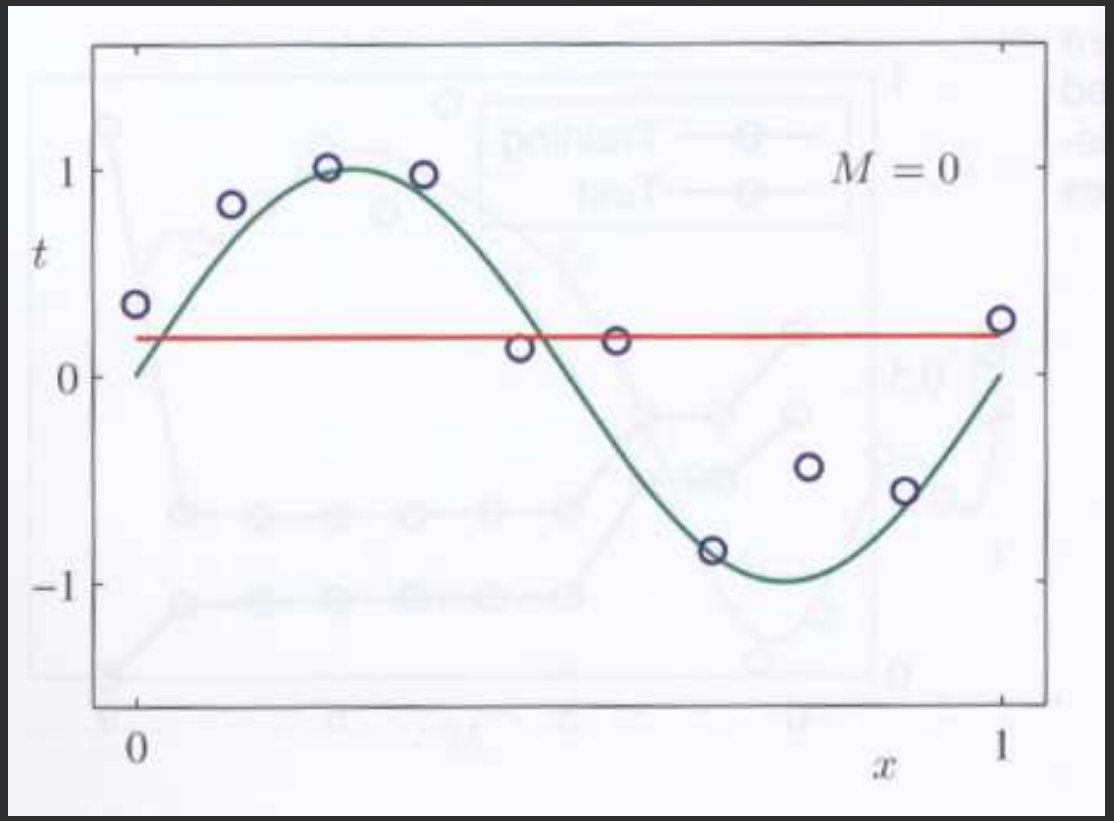

Figure 3: Red curve: polynomial of order $M = 0$ (constant) fitted to the data set of Figure 1. Green curve: the true sinusoid $\sin(2\pi x_i)$.

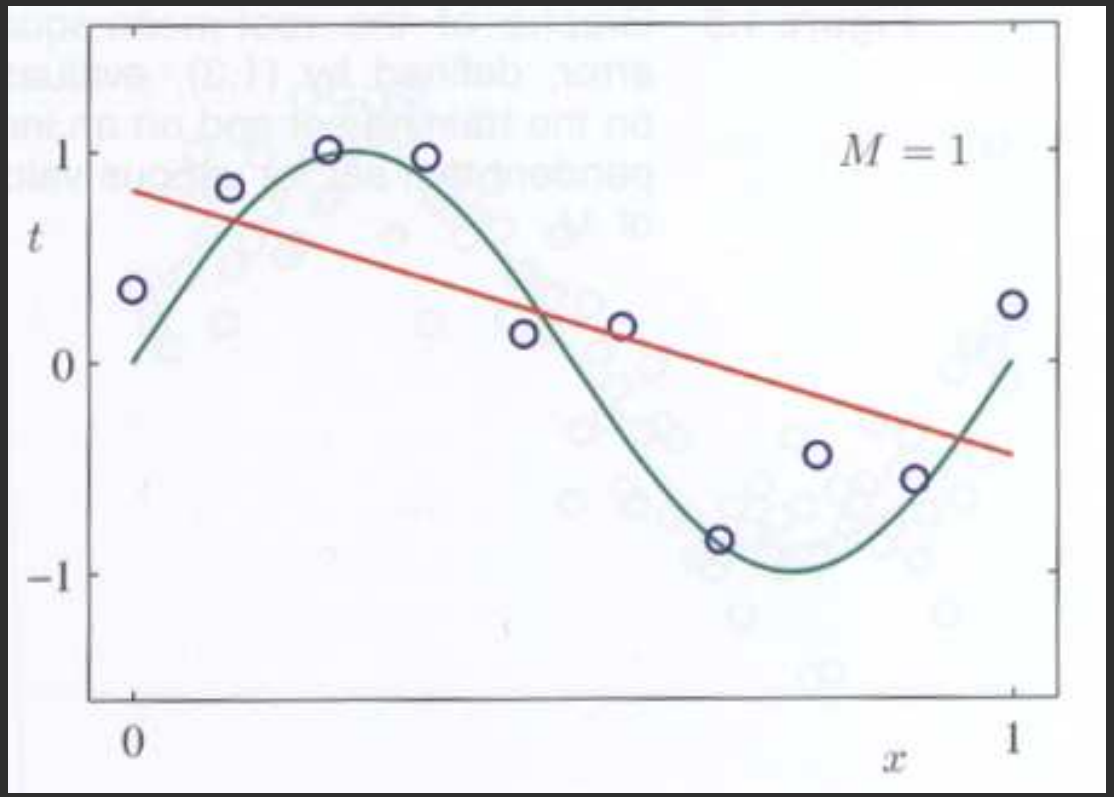

Figure 4: Red curve: polynomial of order $M = 1$ (straight line) fitted to the data set of Figure 1. Green curve: the true sinusoid $\sin(2\pi x_i)$.

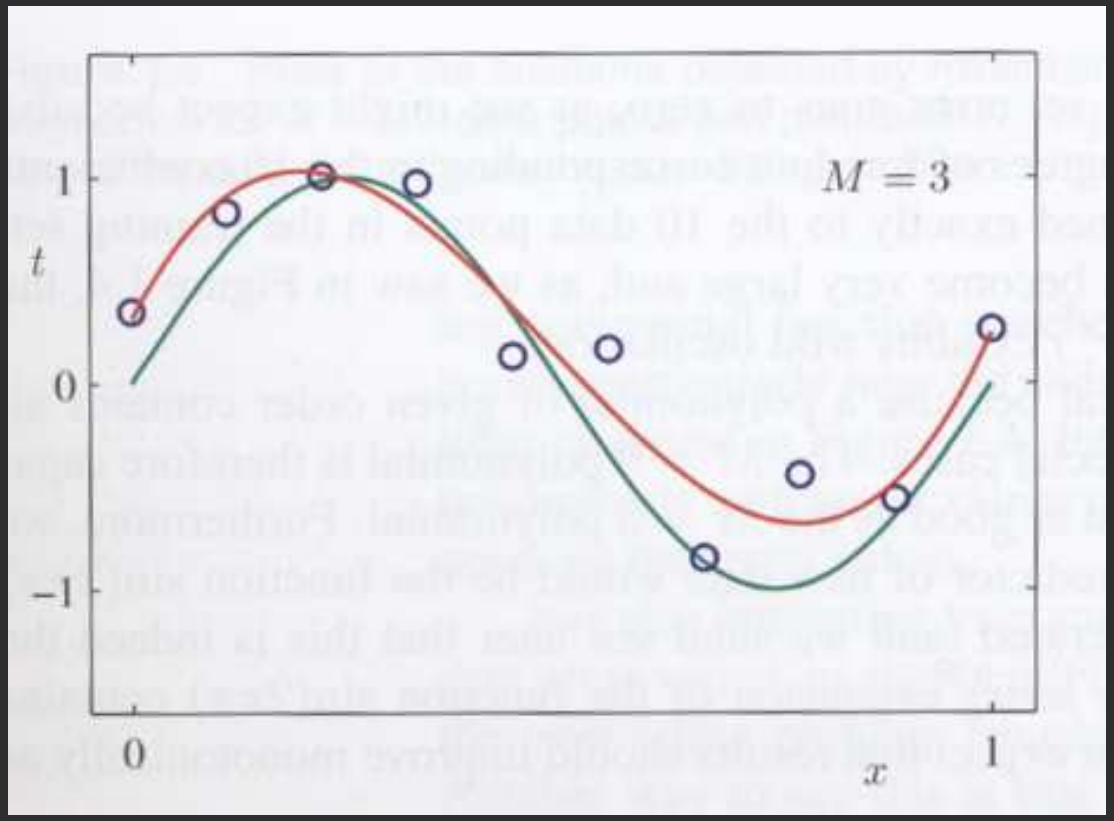

Figure 5: Red curve: polynomial of order $M = 3$ fitted to the data set of Figure 1. Green curve: the true sinusoid $\sin(2\pi x_i)$.

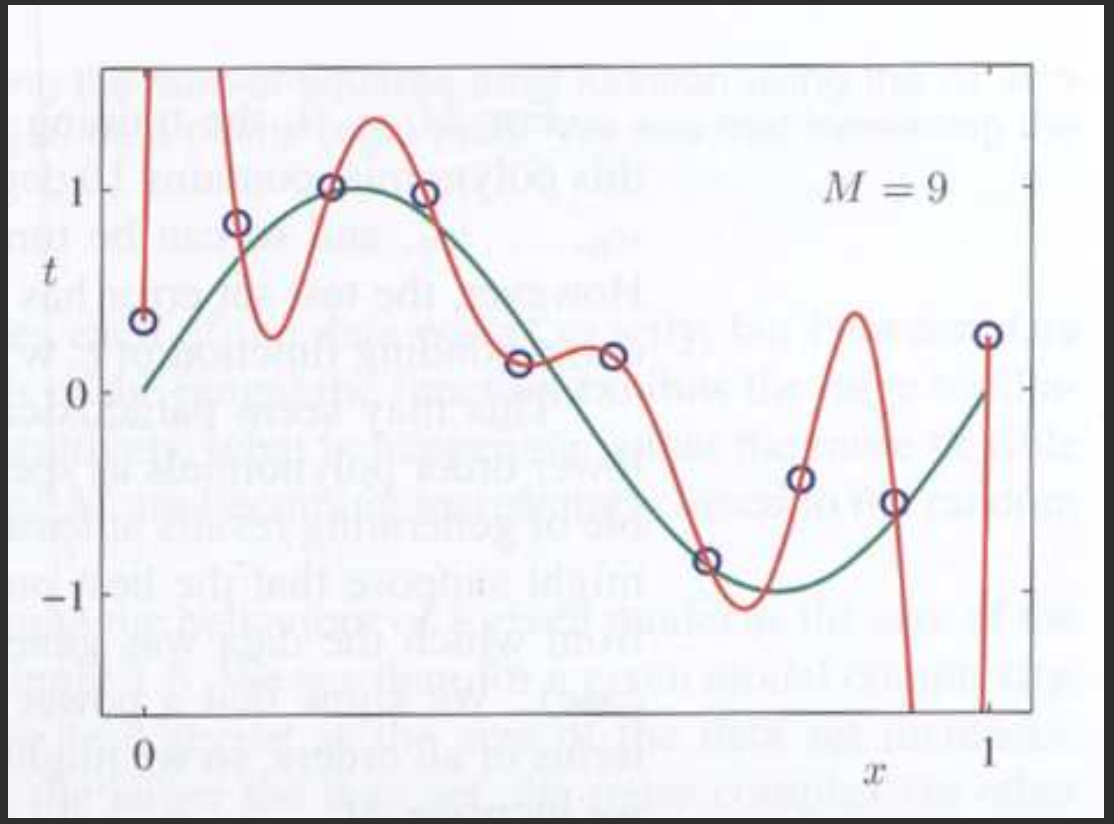

Figure 6: Red curve: polynomial of order $M = 9$ fitted to the data set of Figure 1. Green curve: the true sinusoid $\sin(2\pi x_i)$.

- For clarity, the previous four figures show these fitting results separately for each example polynomial.
- Clearly, the constant ($M = 0$) and straight line ($M = 1$) models in Figures 3 and 4 are too simple.
- They give rather poor fits to the data, and cannot capture anything of the underlying nonlinear function $\sin(2\pi x_i)$.
- The third order polynomial ($M = 3$) in Figure 5 seems to give the best fit to the function $\sin(2\pi x_i)$ of the 4 polynomials tried.
- The highest order polynomial ($M = 9$) in Figure 6 gives an excellent fit to the training data.
- In fact, it passes exactly through each data point, and $E(\mathbf{w}) = 0$.
- This is because it has 10 free parameters which can be fitted perfectly to the 10 training samples.

- However, the fitted red curve in Figure 6 oscillates wildly, and gives a very poor representation of the function $\sin(2\pi x_i)$.
- This phenomenon is known as overfitting.

Training and test error

- The goal is to achieve good generalization by making accurate predictions for new data.
- We can study the dependence of the generalization performance on the model order M by using a separate test set.
- In this case, the test set contains 100 new data points generated using the model (1).
- For each choice of M, we can then evaluate the least-squares error $E(\mathbf{w}^*)$ in (3) for both the training data and test data.
- It is sometimes more convenient to use the root-mean-square (RMS)

error defined by

$$E_{RMS} = \sqrt{2E(\mathbf{w}^*)/N} \tag{4}$$

- The division by N allows comparison of different sizes of data sets on an equal footing.
- The square root ensures that E_{RMS} is measured on the same scale and in the same units as the target variable t.
- Figure 7 shows the root-mean-square error E_{RMS} for various values of the polynomial order M.
- The test error measures how well the model can predict new unseen data points.
- From Figure 7, we can see that small values of M give relatively large values for both the training and test error.

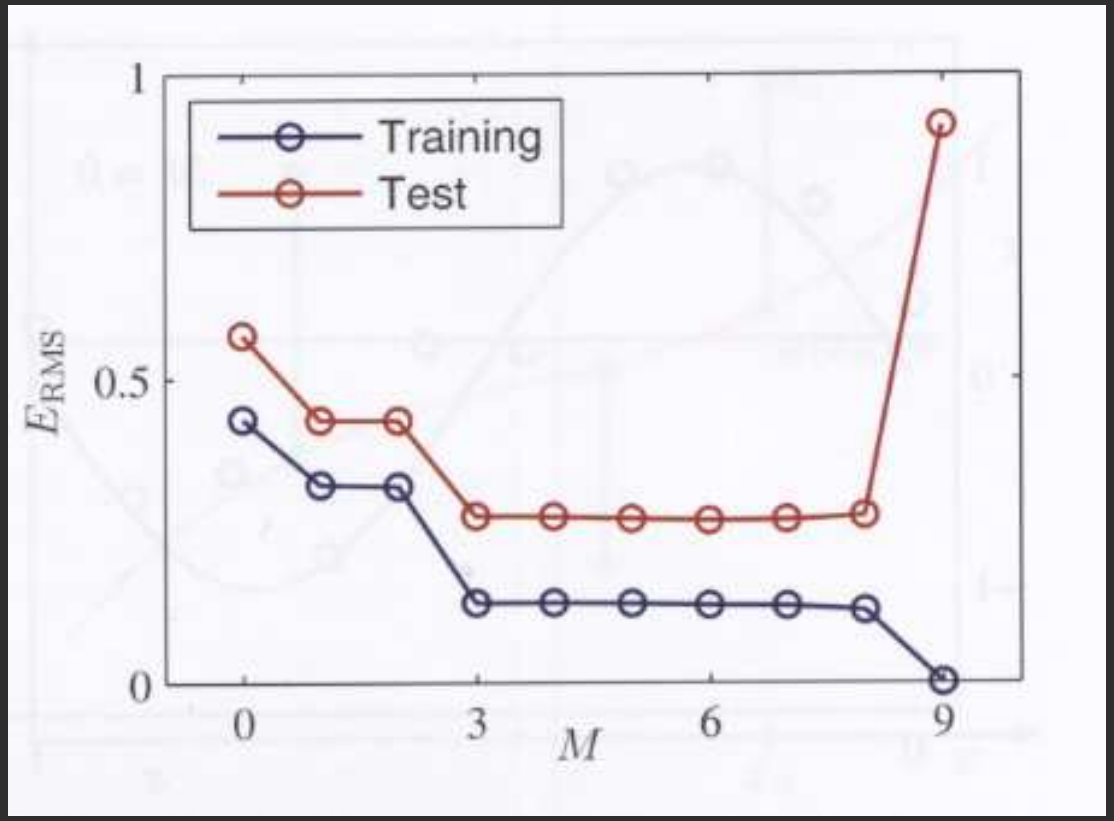

Figure 7: The root-mean-square error of the training set and independent test set for various polynomial orders M

- This is because small-order polynomial are too simple for capturing the oscillations of the sinusoid $\sin(2\pi x_i)$.
- Values in the range $3 \leq M \leq 8$ give small values of the test error, and also reasonably good representations of the generating function $\sin(2\pi x_i)$.
- For $M = 9$, the training error goes to zero, but the test error becomes very large.
- Because the corresponding function $y(x, \mathbf{w}^*)$ exhibits wild oscillations in Figure 6.
- What is actually happening is that the more flexible polynomials with larger values of M become increasingly tuned to the random noise on the target values.

Data size and model complexity

- It is also interesting to examine the behaviour of a given model as the size of the data set is varied.
- Figures 8 and 9 show the optimal least-squares solutions $y(x, \mathbf{w}^*)$ for the $M = 9$ degree polynomial when the number of the data points is $N = 15$ and $N = 100$, respectively.
- We see that, for a given model complexity, the overfitting problem becomes less severe as the size of the data set increases.
- The larger the data set, the more complicated model we can fit to it without overfitting.
- A rough rule of thumb says that the number of data points should be at least 5 to 10 times larger than the number of free parameters in the model to avoid overfitting.

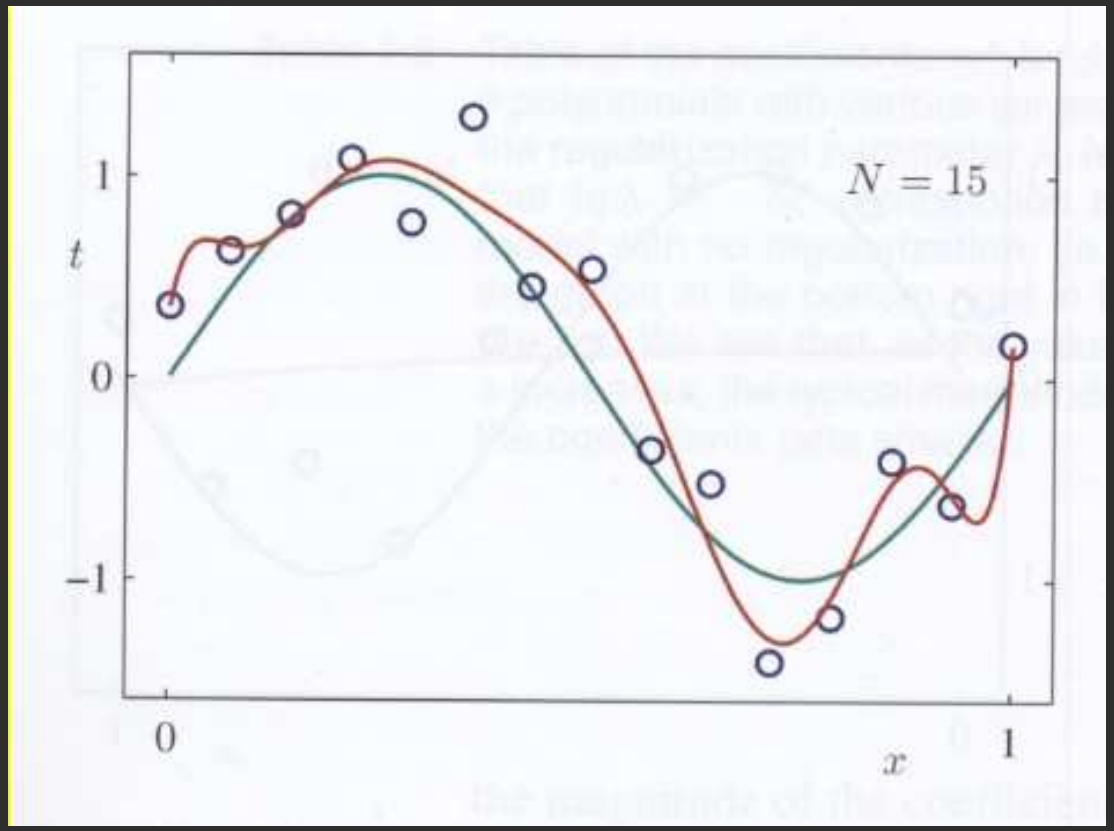

Figure 8: The optimal least-squares solution (red curve) for a $M = 9$ order polynomial for $N = 15$ data points.

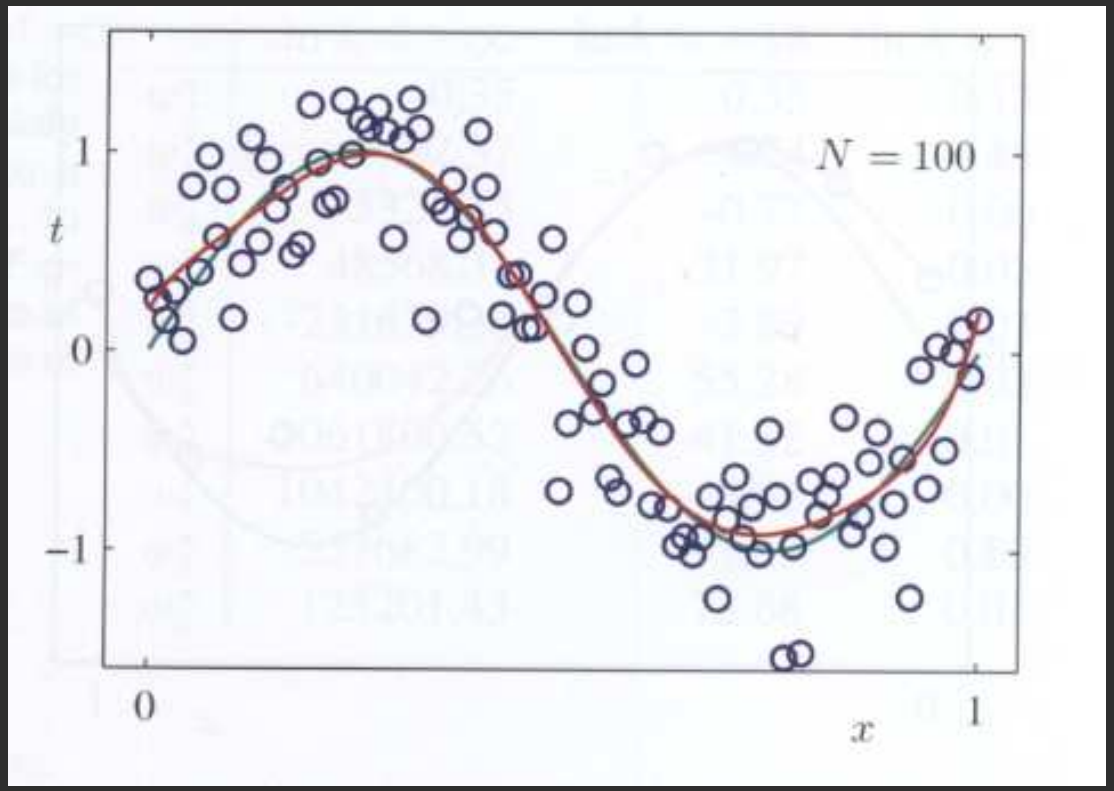

Figure 9: The optimal least-squares solution (red curve) for a $M = 9$ order polynomial for $N = 100$ data points.

Regularization

- Due to overfitting, one must limit the number of free parameters in a model according to the size of the available training set.
- It would be more reasonable to choose the complexity of the model according to the complexity of the problem being solved.
- Regularization techniques can be used to combat overfitting.
- They allow using relatively complex and flexible models for data sets of limited size.
- In regularization, the standard cost function $E(\mathbf{w})$ of a neural network or an optimization problem is modified as

$$\widetilde{E}(\mathbf{w}) = E(\mathbf{w}) + \lambda R(\mathbf{w}) \tag{5}$$

- There the regularizing term $R(\mathbf{w})$ typically penalizes for the complexity of the model.

- Its inclusion imposes on the solution prior knowledge that we may have on the models being considered.
- The regularization parameter λ determines the relative importance of the complexity penalty term $R(\mathbf{w})$ with respect to the standard cost function $E(\mathbf{w})$.
- If λ is very small, the model is determined almost completely by the available training samples via minimization of the standard cost $E(\mathbf{w})$.
- On the other hand, if λ is very large, the model is determined almost completely by the regularization term $R(\mathbf{w})$.
- This means that the training samples are considered to be unreliable.
- In practice, the regularization parameter λ is assigned a value somewhere between these two extremes.

Weight decay

- The simplest and most popular regularizer is called weight decay in neural networks.
- Its modified cost function is defined by

$$\widetilde{E}(\mathbf{w}) = E(\mathbf{w}) + \frac{\lambda}{2}\mathbf{w}^T\mathbf{w} \tag{6}$$

- There the vector $\mathbf{w}$ contains all the weights of the neural network or the parameters of the model studied.
- The quadratic regularizing term $R(\mathbf{w}) = \mathbf{w}^T\mathbf{w}/2$ discourages the weights to reach large values.
- As we shall see from an example slightly later, overlearning often manifests itself as large values of the weights.
- In statistics, this type of techniques are called shrinkage methods

because they reduce the values of the coefficients w_i.

- The particular case (6) is called there ridge regression.
- In fact, it can be shown that the regularizer $\frac{\lambda}{2}\mathbf{w}^T\mathbf{w}$ corresponds to assuming a Gaussian prior distribution for the weight vector $\mathbf{w}$.
- This Gaussian prior distribution is given by

$$p(\mathbf{w} \mid \lambda) = \mathcal{N}(\mathbf{w}; \mathbf{0}; \lambda^{-1}\mathbf{I}) \tag{7}$$

- That is, it has zero mean vector and covariance matrix $\lambda^{-1}\mathbf{I}$.
- The regularizer $\frac{\lambda}{2}\mathbf{w}^T\mathbf{w}$ arises as the negative logarithm of this prior distribution in context with maximum likelihood estimation.
- The derivation is given in Section 3.3.1 in the Bishop's book mentioned below.
- More sophisticated alternative regularization methods for neural

networks are discussed for example in the books:

1. C. Bishop, "Pattern Recognition and Machine Learning", Springer 2006.
2. S. Haykin, "Neural Networks - A Comprehensive Foundation", 2nd ed., Prentice-Hall 1998.

- We already encountered one example on regularization when discussing the Levenberg-Marquardt algorithm.
- There a small matrix $\mu\mathbf{I}$ was added to the matrix $\mathbf{J}^T\mathbf{J}$ for preventing it to become ill-conditioned.
- In fact, corresponds to using weight decay in the least-squares method; see the exercises.
- Regularization is often used also in radial-basis function networks to be discussed later on.

Example on regularization

- We now continue our previous example on polynomial curve fitting by studying the effect of regularization.
- Take first a look at the coefficients of the polynomials fitted using the standard least-squares criterion (3).
- The optimal coefficient vector $\mathbf{w}^* = [w_0^*, w_1^*, \ldots, w_M^*]^T$ is for the first-order polynomial ($M = 1$)

$$\mathbf{w}^* = [0.82,\ -1.27]$$

- For the third-order polynomial ($M = 3$), it is

$$\mathbf{w}^* = [0.31,\ 7.99,\ -25.43,\ 17.37]$$

- And for the ninth-order ($M = 9$) polynomial, it is

$$\begin{aligned}\mathbf{w}^* = [&0.35,\ 232.4,\ -5321.8,\ 48568.3,\ -231639.3,\\ &640042.3,\ -1061800.5,\ 1042400.2,\ -557683.0,\ 125201.4]\end{aligned}$$

- We see that the magnitudes of the coefficients of the fitted polynomials grow dramatically with the order M of the polynomial.
- Therefore, weight decay should be a suitable regularization technique in this problem.
- The modified cost function (6) becomes for the least-squares fitting

$$\widetilde{E}(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}[y(x_n, \mathbf{w}) - t_n]^2 + \frac{\lambda}{2}\mathbf{w}^T\mathbf{w} \tag{8}$$

- Figures 10 and 11 show the fitting results for the $M = 9$ order polynomial for two values of the regularization parameter λ.

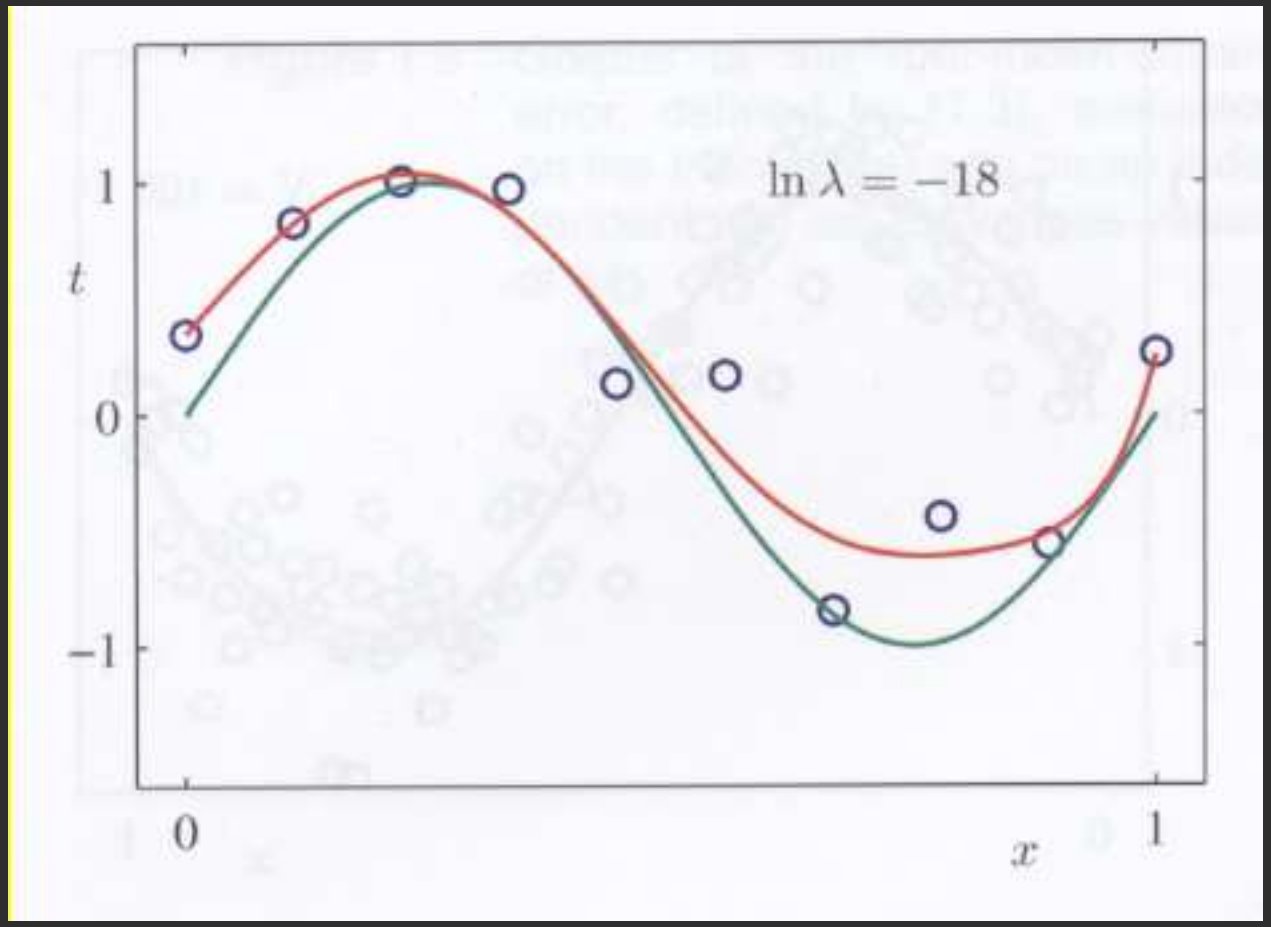

Figure 10: The regularized least-squares solution (red curve) for a $M = 9$ order polynomial using the parameter value $\ln \lambda = -18$.

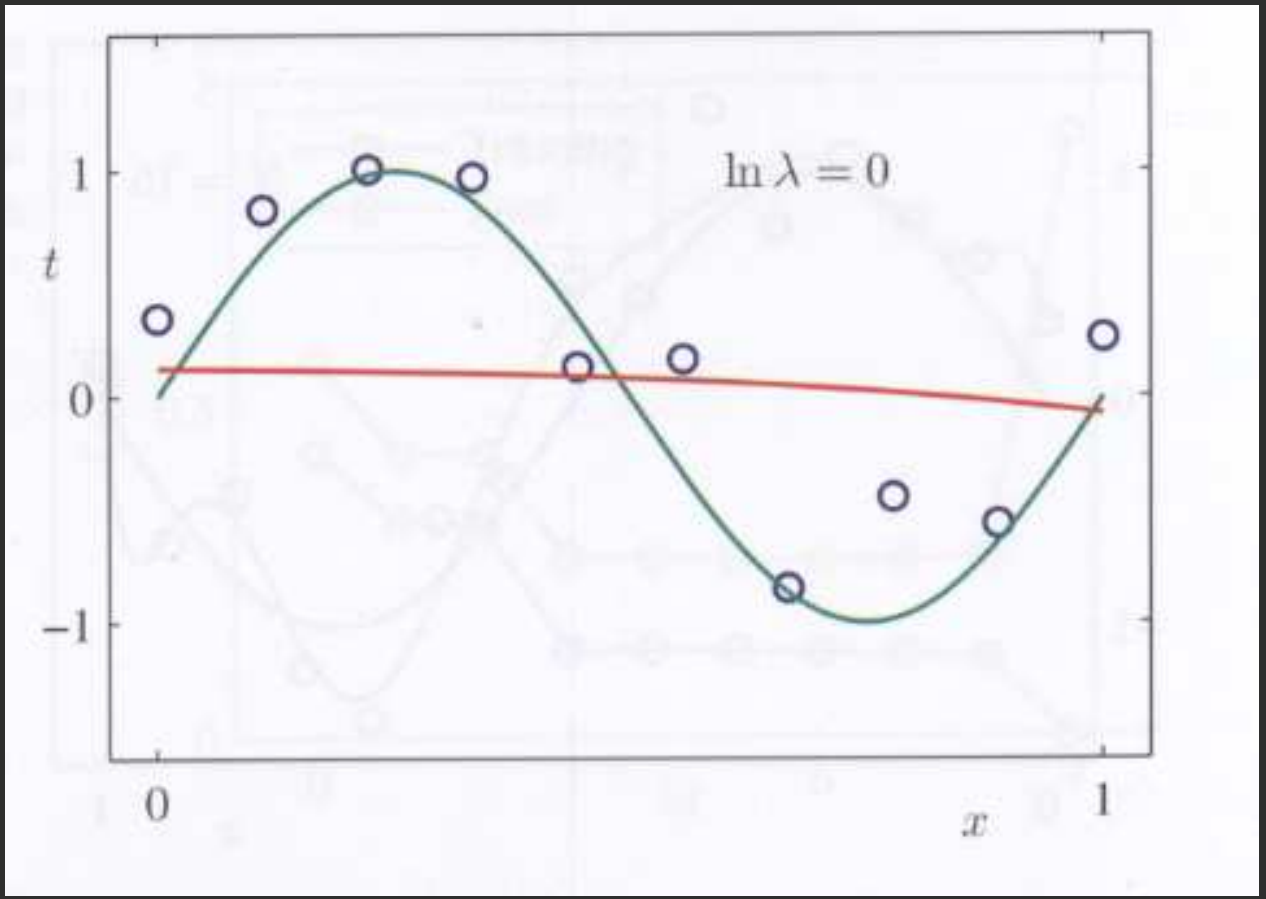

Figure 11: The regularized least-squares solution (red curve) for a $M = 9$ order polynomial using the parameter value $\ln \lambda = 0$.

- The data consists of $N = 10$ points shown in Figure 1.
- The case of no regularization, corresponding to $\lambda = 0$ and $\ln \lambda = -\infty$, is shown already in Figure 6.
- We see from Figure 10 that for a value of $\ln \lambda = -18$, the overfitting has suppressed.
- And we obtain now a much closer representation of the underlying function $\sin(2\pi x_i)$.
- But if we use too large value for λ we again obtain a poor fit, as shown in Figure 11 for $\ln \lambda = 0$.
- For the regularization parameter $\ln \lambda = -18$, the coefficients of the ninth-order ($M = 9$) polynomial are

$$\begin{aligned} \mathbf{w}^* &= [0.35,\ 4.74,\ -0.77,\ -31.97,\ -3.89, \\ &\quad 55.26,\ 41.32,\ -45.95,\ -91.53,\ 72.68] \end{aligned}$$

- And for the parameter $\ln \lambda = 0$, they are very small:

$$\begin{aligned} \mathbf{w}^* &= [0.13,\ -0.05,\ -0.06,\ -0.05,\ -0.03, \\ &\quad -0.02,\ -0.01,\ -0.00,\ 0.00,\ 0.01] \end{aligned}$$

- The impact of the regularization term on the generalization error in this example problem can be seen from Figure 12.
- It depicts the root-mean-square (RMS) error for both training and test sets against $\ln \lambda$ for the $M = 9$ polynomial.
- The good thing is that the test error is close to its minimum on a relatively large range $-35 < \ln \lambda < -25$.
- Using regularization, one can get rid of overfitting.
- But the problem of determining a suitable value to the regularization parameter λ remains.

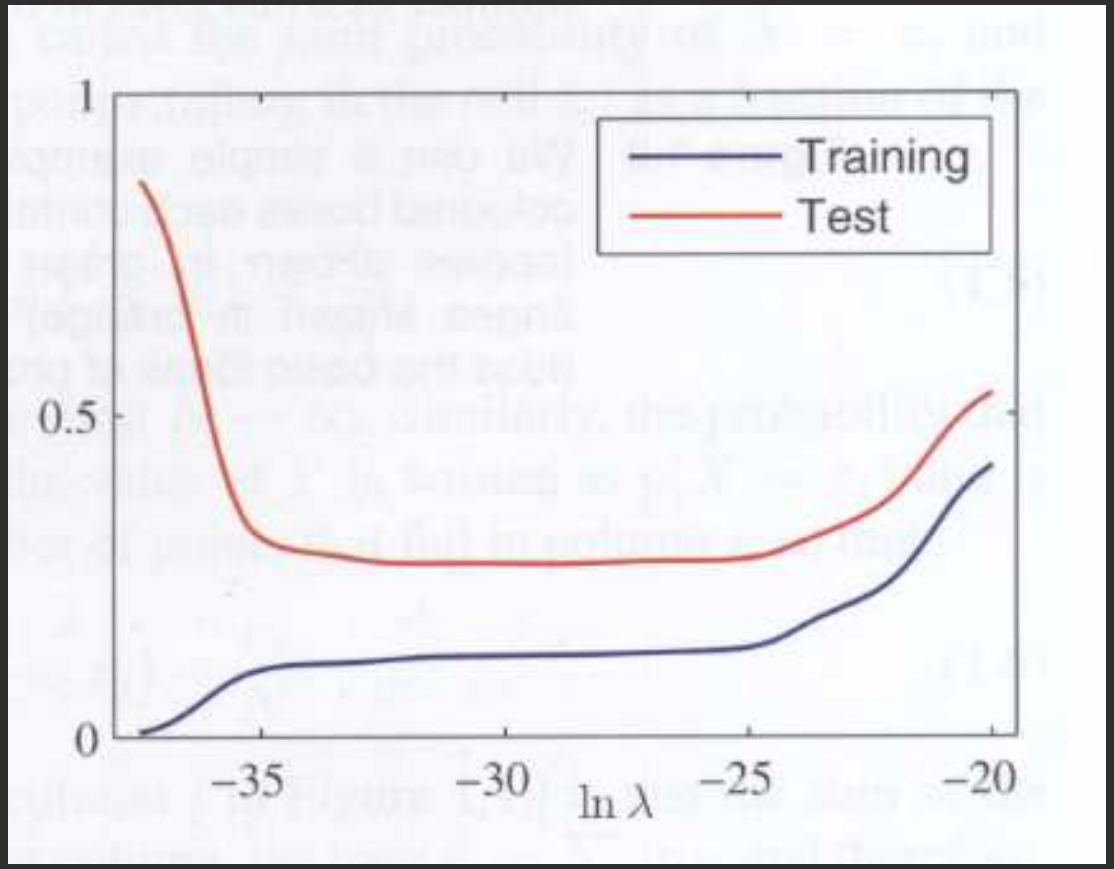

Figure 12: The RMS training and test errors as a function of $\ln \lambda$ for the $M = 9$ polynomial example.

The bias-variance decomposition

- Consider now more generally estimation and learning theory.
- Let us denote by $y(\mathbf{x})$ a specific estimate of the target value t for each input vector $\mathbf{x}$.
- Usually $y(\mathbf{x})$ depends on the parameters $\mathbf{w}$ of the chosen model family or on the weights $\mathbf{w}$ of the chosen neural network.
- But we shall not need the parameter vector $\mathbf{w}$ in our discussion here, and therefore drop it out for convenience.
- For simplicity, we also assume that both $y(\mathbf{x})$ and the target variable t are scalar valued.
- But the results can easily be extended for vector-valued $\mathbf{y}$ and $\mathbf{t}$.
- A widely used performance criterion in estimation theory is the

mean-square estimation error

$$\text{MSE} = \text{E}[\{y(\mathbf{x}) - t\}^2] = \int \int \{y(\mathbf{x}) - t\}^2 p(\mathbf{x}, t) d\mathbf{x} dt \qquad (9)$$

- Note that E denotes the mathematical expectation, while we previously used the symbol E for the cost function.
- The mean-square estimation error (9) is different from the deterministic least-squares criterion (3).
- It is probabilistic, and involves an integration over the joint distribution $p(\mathbf{x}, t)$ of the data vectors $\mathbf{x}$ and target values t.
- If we assume a completely flexible function $y(\mathbf{x})$, it is easy to show that the minimum of the MSE error is given by

$$h(\mathbf{x}) = y^*(\mathbf{x}) = \int t p(t \mid \mathbf{x}) dt = \text{E}_t[t \mid \mathbf{x}] \qquad (10)$$

- There $\mathrm{E}_t[t \mid \mathbf{x}]$ is the conditional expectation of t given the data $\mathbf{x}$.
- Theoretically, this is a very important result in estimation theory.
- It holds for all the distribution $p(\mathbf{x}, t)$ of the data vectors $\mathbf{x}$ and target values t.
- The problem is that in practice the conditional expectation (10) cannot usually computed, and it must be approximated in some way.
- This is because the posterior density is usually not known, and the required integrations cannot be performed analytically.
- For any (generally non-optimal) estimate $y(\mathbf{x})$, the mean-square estimation error (9) can be decomposed into two terms:

$$\mathrm{MSE} = \int \{y(\mathbf{x}) - h(\mathbf{x})\}^2 p(\mathbf{x}) d\mathbf{x} + \int \{h(\mathbf{x}) - t\}^2 p(\mathbf{x}, t) d\mathbf{x} dt \quad (11)$$

- The second term is independent of the estimate $y(\mathbf{x})$.

- It represents the minimum achievable value of the MSE error.
- It arises from the intrinsic variability of the target data, and can be regarded as noise:

$$\text{noise} = \int \{h(\mathbf{x}) - t\}^2 p(\mathbf{x}, t) d\mathbf{x} dt \tag{12}$$

- The first term in (11) is nonnegative.
- The best that we can hope is to make it zero.
- In principle, this is possible if we have an unlimited amount of data (and unlimited computational resources).
- In practice, we have a data set D containing only a finite number N of data points.
- Suppose that we had a large number of such independent data sets of size N.

- For any given data set D, we can run our learning algorithm and obtain an estimate $y(\mathbf{x}; D)$.
- Different data sets will give different estimates and consequently different values of the mean-square error.
- The performance of a particular learning algorithm can then be assessed by taking the average over this ensemble of data sets.
- The integrand of the first term in (11) is for a particular data set D

$$\{y(\mathbf{x}; D) - h(\mathbf{x})\}^2 \tag{13}$$

- We can now evaluate its expectation

$$\mathsf{E}_D[\{y(\mathbf{x}; D) - h(\mathbf{x})\}^2] \tag{14}$$

  over D and insert the result into (11).
- The derivation is given in the Bishop's book; we skip it.

- The result is an important decomposition of the mean-square error into three terms:

$$\text{MSE} = (\text{bias})^2 + \text{variance} + \text{noise} \tag{15}$$

- The third noise term (12) is the irreducible error due to the intrinsic variation of the data.
- The first term,

$$(\text{bias})^2 = \int \{\text{E}_D[y(\mathbf{x}; D)] - h(\mathbf{x})\}^2 p(\mathbf{x}) d\mathbf{x} \tag{16}$$

  called the squared bias, represents the systematic error made by the estimator $y(\mathbf{x}; D)$.
- It measures how much the average prediction $\text{E}_D[y(\mathbf{x}; D)]$ over all the data sets differs from the desired regression function $h(\mathbf{x})$.

- The second term in (15),

$$\text{variance} = \int \mathrm{E}_D[\{y(\mathbf{x}; D) - \mathrm{E}_D[y(\mathbf{x}; D)]\}^2] p(\mathbf{x}) d\mathbf{x} \qquad (17)$$

  called the variance, measures how much the solutions for individual data sets vary around their average $\mathrm{E}_D[y(\mathbf{x}; D)]$.

- Hence the variance measures how sensitive the estimate $y(\mathbf{x}; D)$ is to the particular choice of data set D.
- Our goal is to minimize the mean-square error (15).
- It turns out that there is trade-off between bias and variance.
- Very flexible models with many free parameters have a low bias and high variance.
- While rigid models have high bias and low variance.
- This is often called the bias-variance dilemma.

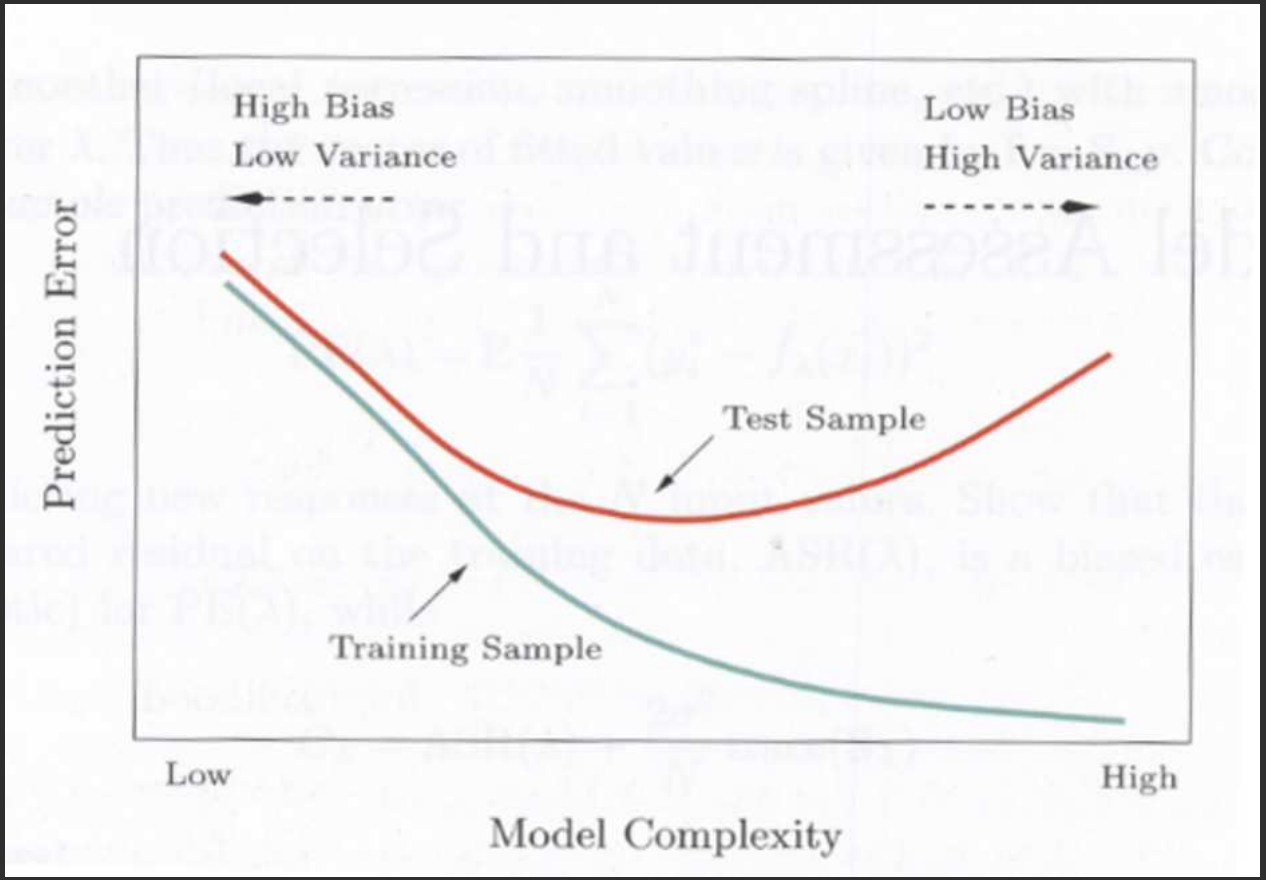

Figure 13: Test and training error as a function of model complexity.

- The model with the optimal estimation (prediction) capability is the one that leads to the best balance between bias and variance.
- If we use the mean-square error criterion (15) in supervised learning tasks, the optimal model minimizes the MSE error for the test data.
- The general situation is illustrated in the important Figure 13.
- The prediction error there is just the MSE error (15).

Example: sinusoid in noise

- We illustrate these theoretical results on the bias and variance by using the familiar noisy sinusoidal data

$$t_i = h(x_i) + n_i = \sin(2\pi x_i) + n_i \tag{18}$$

- Now 100 independent data sets (realizations) each containing $N = 25$ data points were generated from the sinusoidal curve $h(x) = \sin(2\pi x)$.

- These data sets are indexed by $l = 1, 2, \ldots, L$ where $L = 100$.
- We model the data in a similar manner as we did earlier using polynomials.
- But now we use 'Gaussian' basis functions

$$\phi_j(x) = \exp[-\frac{(x - \mu_j)^2}{2s^2}] \tag{19}$$

  where the parameter μ_j ("mean") defines the center of the function, and the parameter s ("standard deviation") determines its width.
- The target values t are modeled as linear combinations

$$y(x, \mathbf{w}) = \sum_{j=0}^{M-1} w_j \phi_j(x) \tag{20}$$

  of the Gaussian basis functions.
- Note that the coefficient w_0 introduces the bias term corresponding

to the 'dummy' basis function $\phi_0(x) = 1$.

- For each data set $D^{(l)}$ we fit a model (20) with 24 Gaussian basis functions.
- The total number of free parameters is 25, including the bias w_0.
- The fitting is done my minimizing the regularized least-squares error criterion

$$\widetilde{E}(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}[y(x_n, \mathbf{w}) - t_n]^2 + \frac{\lambda}{2}\mathbf{w}^T\mathbf{w} \qquad (21)$$

- Each data set yields its own prediction function $y^{(l)}(x)$.
- The parameter vector $\mathbf{w}^{(l)}$ has here been dropped out for simplifying the notation.
- The following figures illustrate the bias and variance for three values of the regularization parameter λ.

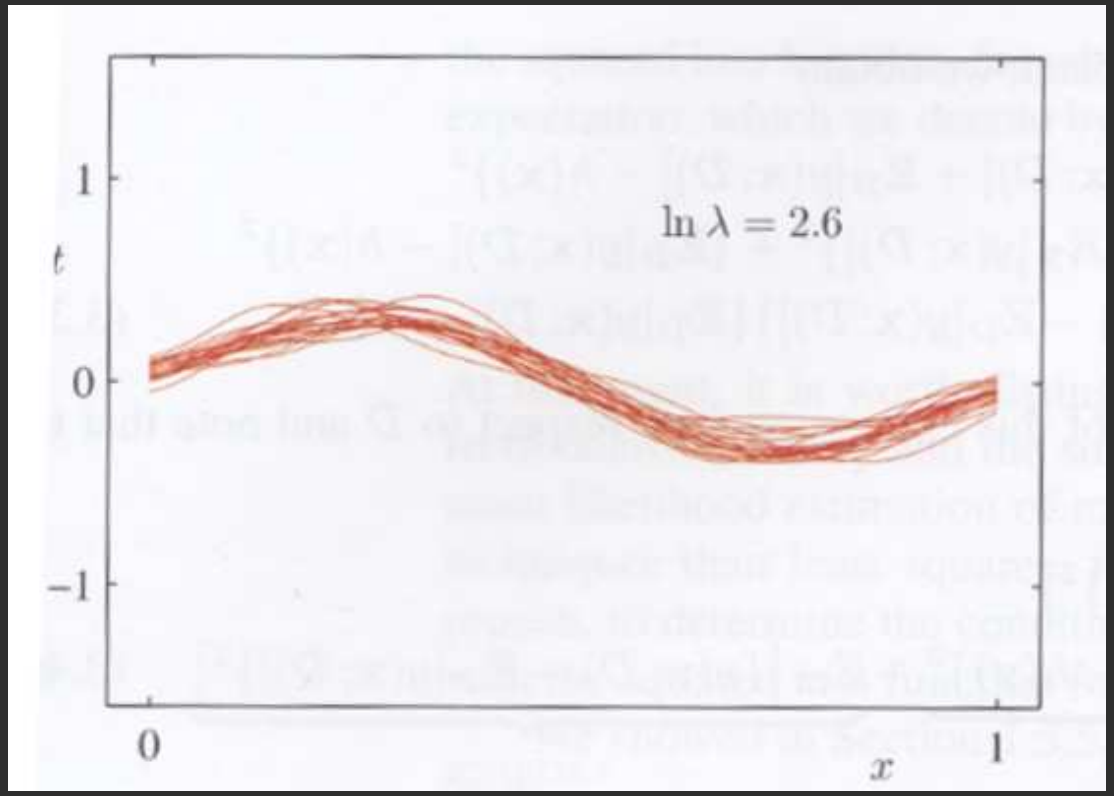

Figure 14: 20 regularized least-squares prediction functions $y^{(l)}(x)$ for the parameter value $\ln \lambda = 2.6$.

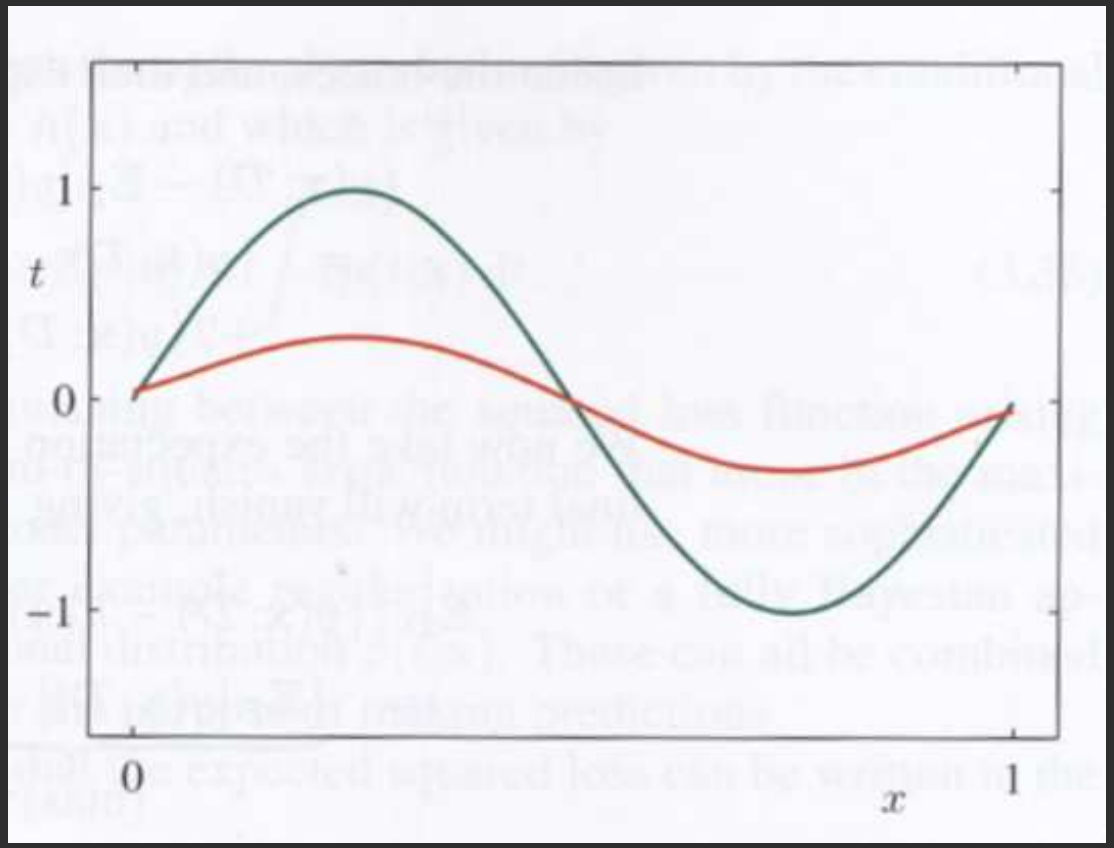

Figure 15: The sinusoid $\sin(2\pi x)$ (green curve) and the average prediction $\bar{y}(x)$ (red curve) for the regularization parameter $\ln \lambda = 2.6$.

- They correspond to models having different flexibility.
- In each of the "variance" figures, only 20 prediction functions $y^{(l)}(x)$ are shown for clarity.
- While in the "bias" figures, the green curve is the true sinusoid $h(x) = \sin(2\pi x)$.
- The red curve there is the average prediction

$$\bar{y}(x) = \frac{1}{L}\sum_{l=1}^{L} y^{(l)}(x) \tag{22}$$

  of all the $L = 100$ data sets.
- We see that for the regularization parameter $\ln \lambda = 2.6$, the variance in Figure 14 is small, but the bias in Figure 15 is quite large.
- The results show that the regularization parameter $\ln \lambda = 2.6$ is too large, leading to too rigid a model.

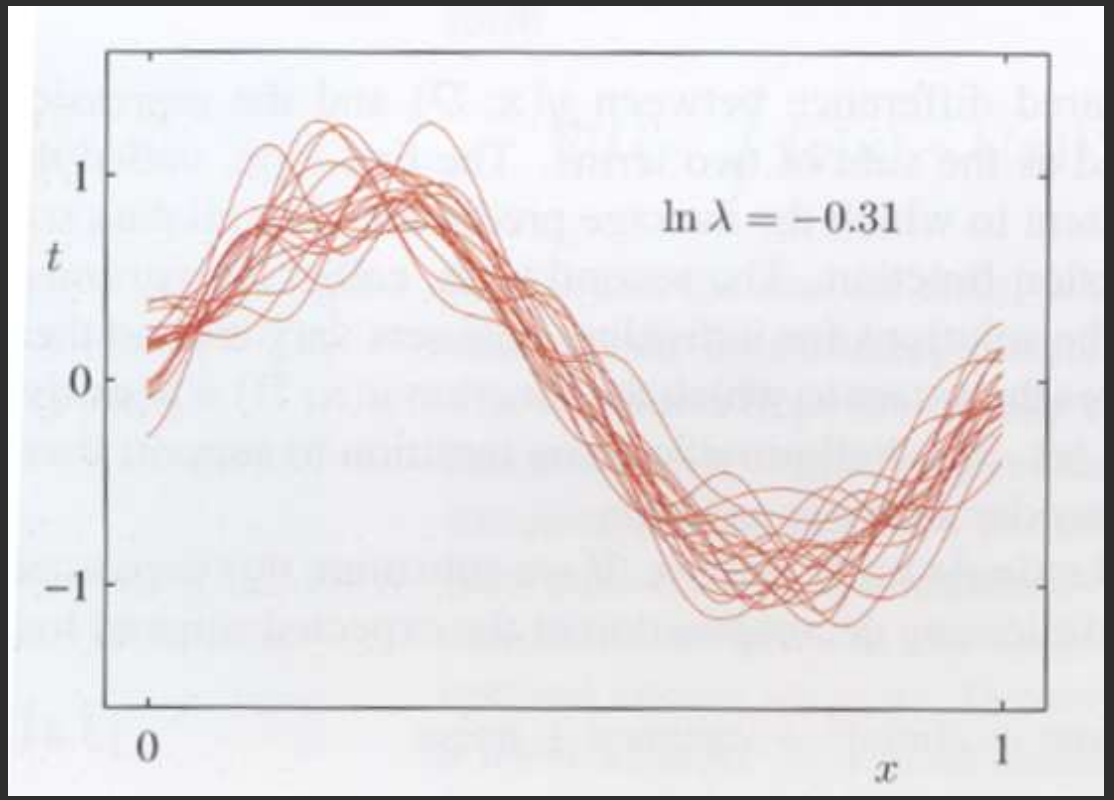

Figure 16: 20 regularized least-squares prediction functions $y^{(l)}(x)$ for the parameter value $\ln \lambda = -0.31$.

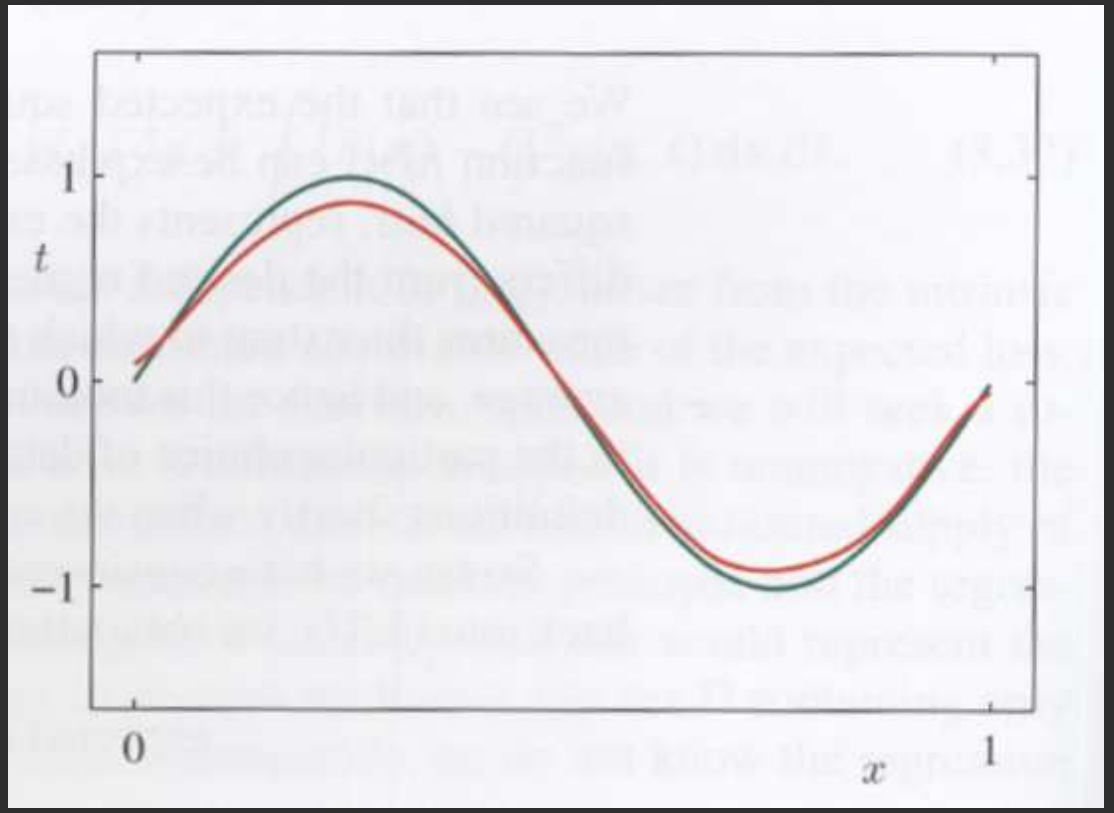

Figure 17: The sinusoid $\sin(2\pi x)$ (green curve) and the average prediction $\bar{y}(x)$ (red curve) for the regularization parameter $\ln \lambda = -0.31$.

- For the parameter value $\ln \lambda = -0.31$, the variance in Figure 16 is larger, but the bias in Figure 17 is rather small.
- While for $\ln \lambda = -2.4$ the variance in Figure 18 is fairly large, but the bias in Figure 19 quite small.
- In this example, the regularization parameter $\ln \lambda = -0.31$ provides the best compromise between the bias and variance.
- Note also how averaging improves the results, reducing the variance dramatically.
- In fact, averaging over $L = 100$ data sets of $N = 25$ samples corresponds roughly to using $100 \times 25 = 2500$ samples for estimation.
- Then the most flexible model in Figure 19 would perform the best.

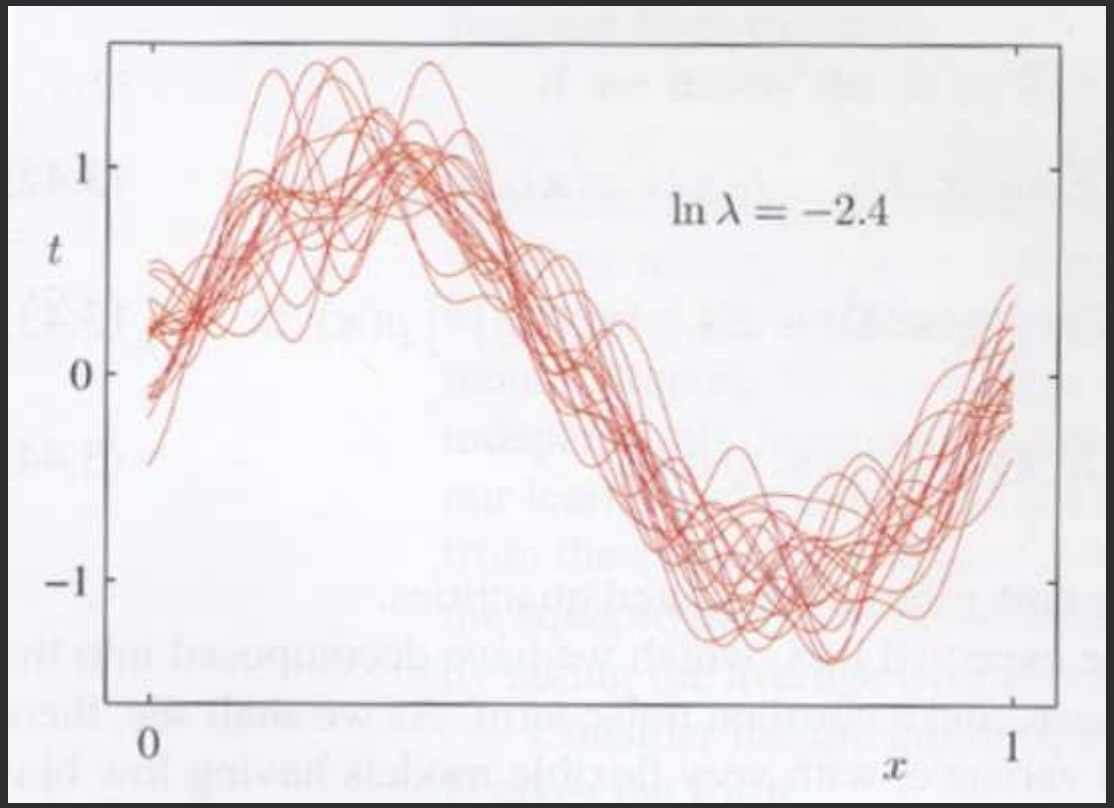

Figure 18: 20 regularized least-squares prediction functions $y^{(l)}(x)$ for the parameter value $\ln \lambda = -2.4$.

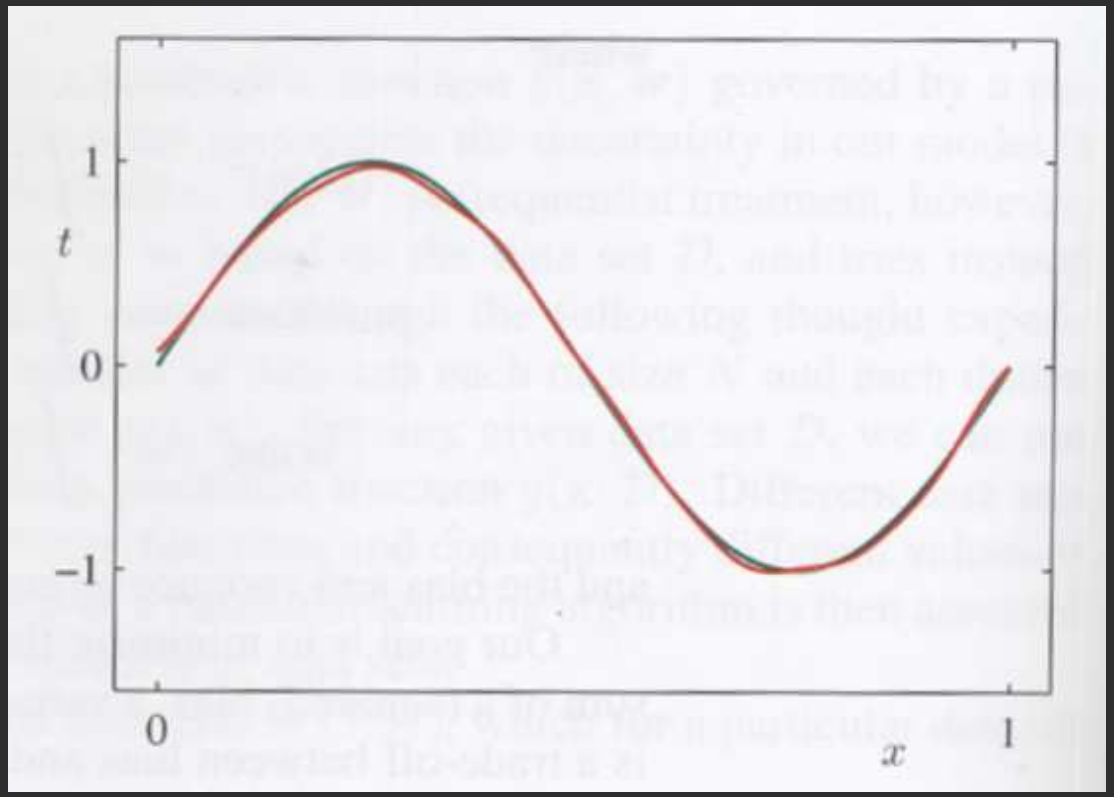

Figure 19: The sinusoid $\sin(2\pi x)$ (green curve) and the average prediction $\bar{y}(x)$ (red curve) for the regularization parameter $\ln \lambda = -2.4$.

- The bias in Eq. (16) and variance in Eq. (17) can be estimated numerically.
- This can be done by approximating the integral over x weighted by the distribution $p(x)$ by a finite sum over data points drawn from that distribution.
- This procedure yields the following sample estimates:

$$(\text{bias})^2 = \frac{1}{N}\sum_{n=1}^{N}[\bar{y}(x_n) - h(x_n)]^2 \tag{23}$$

$$\text{variance} = \frac{1}{N}\sum_{n=1}^{N}\frac{1}{L}\sum_{l=1}^{L}[y^{(l)}(x_n) - \bar{y}(x_n)]^2 \tag{24}$$

- The bias-variance decomposition provides interesting insights into the model complexity issue.
- But it is of limited practical value, because it is based on averages

over many data sets.

- In practice, we have only the single observed data set.
- If we had a large number of independent training sets of given size, it would be better to combine them into a single large training set.
- Using such a large data set reduces the level of overfitting for a given model complexity.

Validation and model order selection

- The number of inputs and outputs in a neural network is generally determined by the dimensionality of the data set.
- Whereas the number M of hidden units is a free parameter that can be adjusted to give the best predictive performance.
- One might expect that there will be an optimum value of M that gives the best generalization performance.

- Corresponding to the optimum balance between underfitting and overfitting.
- There are in fact two separate goals that we can have:
  1. Model selection: Estimation of the performance of different models in order to choose the (approximate) best one.
  2. Model assessment: Having chosen a final model, estimation of its prediction (generalization) error on new data.
- In a data-rich situation, the best approach for both problems is to randomly divide the dataset into three parts:
  - Training set, which is used to fit the models;
  - Validation set, which is used to estimate prediction error for model selection;
  - Test set, which is used for assessment of the generalization error of the final chosen model.

- Ideally, the test set should be kept in a "vault", and be brought out only at the end of the data analysis.
- In practice the validation and test sets are often combined to a single set called either validation or test set.
- If the model design is iterated many times using a limited data set, then some overfitting to the validation data can occur in this case.
- And the validation set error of the final chosen model will underestimate the true test error, sometimes substantially.
- It is difficult to give a general rule on how many observations there should be in each of these three parts.
- A typical split might be 50% for training, and 25% each for validation and testing.
- A problem in the above procedures is that the generalization error is not a simple function of M.

- Because of the presence of local minima in the error function.
- One can choose multiple random initializations for the weights for a range of values of M.
- And then choose the value of M that gives the overall best performance for the validation set.
- If many models are tried and tested, the computational load with teaching each model with many different initializations can grow quite large.
- Notice that the dataset is divided into training, validation, and test sets for model selection and performance assessment.
- After carrying out this procedure, these three data sets should be combined back to the original data set in real-world applications.
- And then this larger data set is used for final training of the selected model for getting best results.

Cross-validation

- In many applications, there is insufficient amount of data to split it into three parts as above.
- In order to build good models, we would like to use as much of the available data as possible for training.
- However, if the validation set is small, it will give a relatively noisy estimate of generalization (prediction) performance.
- Probably the simplest and most widely used method for estimating generalization error is cross-validation.
- In cross-validation, we split the available data to K roughly equal-sized parts.
- Typical choices of K are 5 or 10.
- If $K = 5$, we always use four of the data subsets for training, and

the remaining one for testing.

- The subset used for testing is varied so that each subset serves once as a test set.
- Thus first say the first subset serves as a test set, and the model is learned using the four other subsets put together as the training set.
- And finally the fifth subset is used for testing while the model is learned by using the first four subsets together as the training data.
- The K estimates of generalization error obtained in this way are combined typically by averaging them.
- This procedure partly overcomes the local minimum problem, too, if different random initializations are used in each case.
- In the extreme case the amount N of training data can be quite small.

- Then it may be appropriate to split the data into N sets containing each one data sample.
- This is called the leave-out technique.
- A major drawback of cross-validation is that the number of training runs required increases by a factor of K.
- This can prove problematic for models in which the training itself is computationally expensive.
- This is even more so if there are multiple complexity parameters for a single model.
- For example, several regularization parameters in addition to the number of neurons in the hidden layer(s).

Early stopping

- The early stopping method is an alternative to regularization for controlling the effective complexity of a neural network.
- For example, a MLP network trained with the back-propagation algorithm learns in stages.
- Moving from the realization of fairly simple to more complex mapping functions as the training session progresses.
- This shows up also in the behavior of the mean-square training error.
- During training, it starts off at a large value, decreases rapidly, and then continues to decrease slowly.
- It is very difficult to figure out when it is best to stop training if we monitor during the learning the training error curve only.
- If training is continued too long, the network may end up overfitting

the training data.

- We may use a variant of cross-validation here: the training set is split into an estimation subset and validation subset.
- The training session is stopped periodically, and the network is tested on validation subset after each period of training.
- The training error decreases fairly monotonically.
- But the validation error decreases first monotonically to a minimum.
- Then it starts to grow due to overfitting.
- Learning is stopped at the stage where the validation error gets its minimum.

Other model order selection methods

- Other model selection methods include information-theoretic methods and Bayesian methods.
- Information theoretic methods, such as Akaike's information criterion (AIC) and Schwartz's Bayesian information criterion (BIC), are typically sums of two parts.
- The first part, for example the mean-square error, decreases monotonically with the increasing model order.
- The second complexity penalizing part increases with the number of parameters in the model.
- When the sum attains its minimum, the model order is considered to be correct.
- AIC and BIC are discussed briefly in subsection 2.3.2 in Du's and

Swamy's book.

- They are more useful for simpler models than for neural networks.
- In fully Bayesian methods, one can estimate the probability of studied model, and then select the most probable model.
- Bayesian methods do not suffer markedly from the overfitting.
- But they are often computionally costly and conceptually demanding.
- Bayesian methods are discussed in detail in our course T-61.5140 Machine Learning: Advanced Probabilistic Methods.

Curse of dimensionality

- Curse of dimensionality means that high-dimensional data vectors cause often problems.

- The first problem is that the computation time needed may "explode" or grow intolerably large.
- Because the computational load of a method can be proportional for example to the 3rd or 4th power of the dimensionality d of the data.
- Other problems are that the number of parameters needed and the size of the neural network usually grow with the dimensionality d.
- Third problem is that as the dimensionality of the data d grows, the number of data vectors should be increased exponentially.
- For performing an adequate sampling of the data space.
- Otherwise the data set can be unbalanced and the test results quite wrong even though the training results were accurate.
- This problem arises in many real life situations because of the difficulty of obtaining enough data vectors.

- Two methods for combatting the curse of dimensionality:
  1. Data compression using for example principal component analysis.
  2. Variable selection: only the most informative components of the data vectors are used in computations.
- In variable selection, components of data vectors which are insignificant for getting good results are discarded.
- We shall not discuss variable selection methods in our course.
- Some papers on variable selection can be found on the publications subpage of the EIML group in Aalto University.
- See http://research.ics.aalto.fi/eiml/publications.shtml
- For example papers 3/2012, 10/2011, 9/2011, 23/2009 there.