

Linear Regression II, SGD

Milan Straka

 October 12, 2020



Charles University in Prague
Faculty of Mathematics and Physics
Institute of Formal and Applied Linguistics



unless otherwise stated

Given an input value $\mathbf{x} \in \mathbb{R}^D$, **linear regression** computes predictions as:

$$y(\mathbf{x}; \mathbf{w}, b) = x_1 w_1 + x_2 w_2 + \dots + x_D w_D + b = \sum_{i=1}^D x_i w_i + b = \mathbf{x}^T \mathbf{w} + b.$$

The *bias* b can be considered one of the *weights* \mathbf{w} if convenient.

We train the weights by minimizing an **error function** between the real target values and their predictions, notably *sum of squares*:

$$\frac{1}{2} \sum_{i=1}^N (y(\mathbf{x}_i; \mathbf{w}) - t_i)^2$$

There are several ways how to minimize it, but in our case, there exists an explicit solution:

$$\mathbf{w} = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{t}.$$

Linear Regression Example

Assume our input vectors comprise of $\mathbf{x} = (x^0, x^1, \dots, x^M)$, for $M \geq 0$.

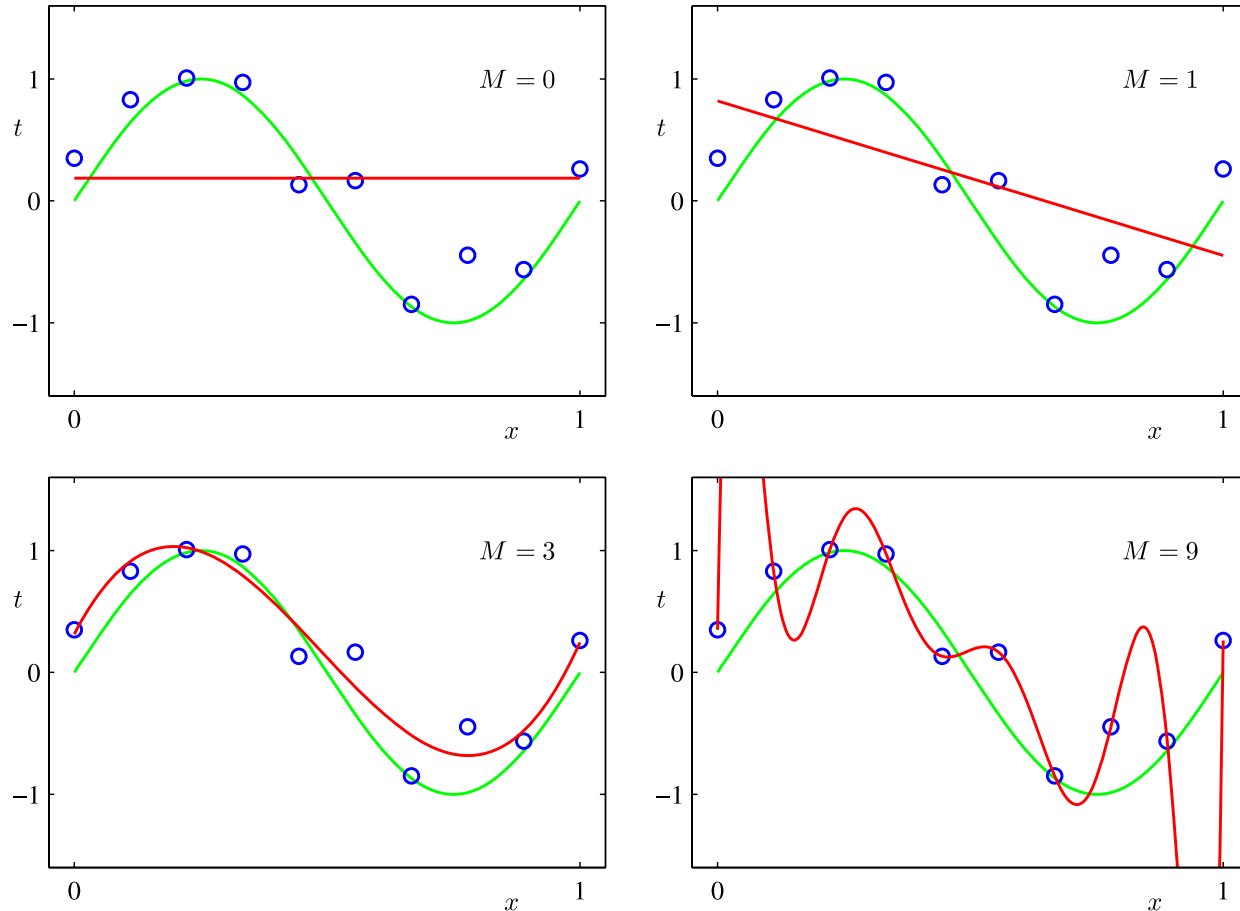


Figure 1.4 of Pattern Recognition and Machine Learning.

Linear Regression Example

To plot the error, the *root mean squared error* $\text{RMSE} = \sqrt{\text{MSE}}$ is frequently used.

The displayed error nicely illustrates two main challenges in machine learning:

- *underfitting*
- *overfitting*

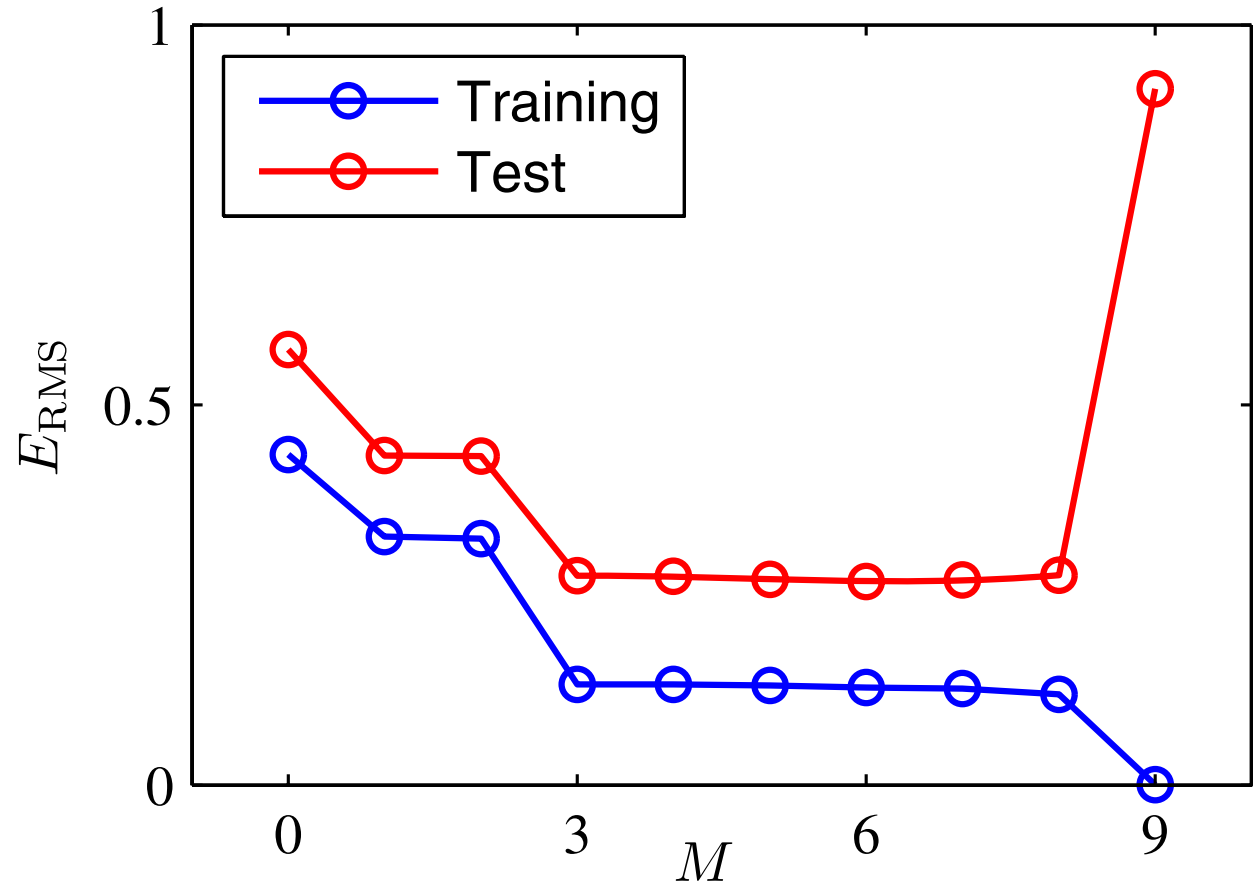


Figure 1.5 of Pattern Recognition and Machine Learning.

Model Capacity

We can control whether a model underfits or overfits by modifying its *capacity*.

- representational capacity
- effective capacity

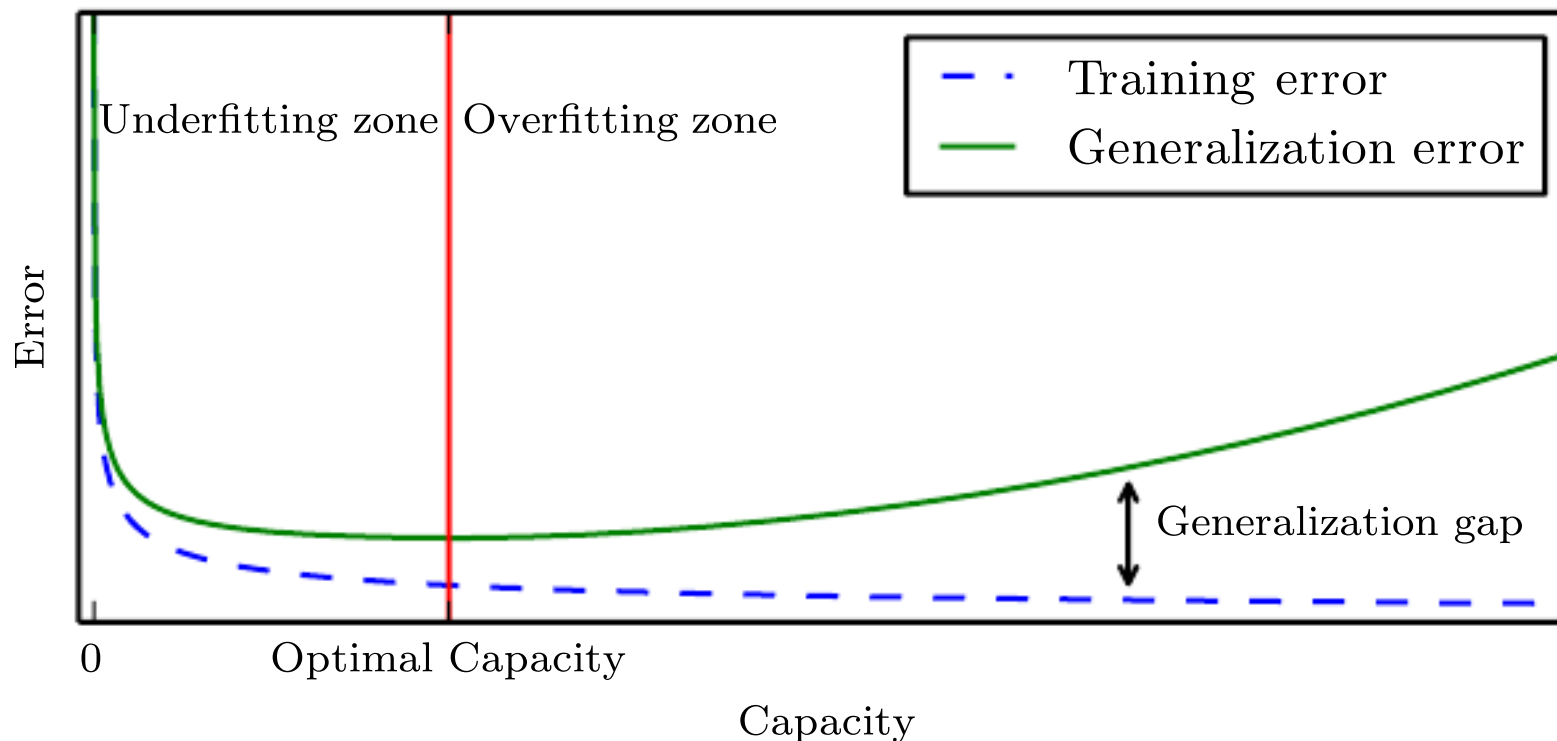


Figure 5.3, page 115 of Deep Learning Book, <http://deeplearningbook.org>

Linear Regression Overfitting

Note that employing more data also usually alleviates overfitting (the relative capacity of the model is decreased).

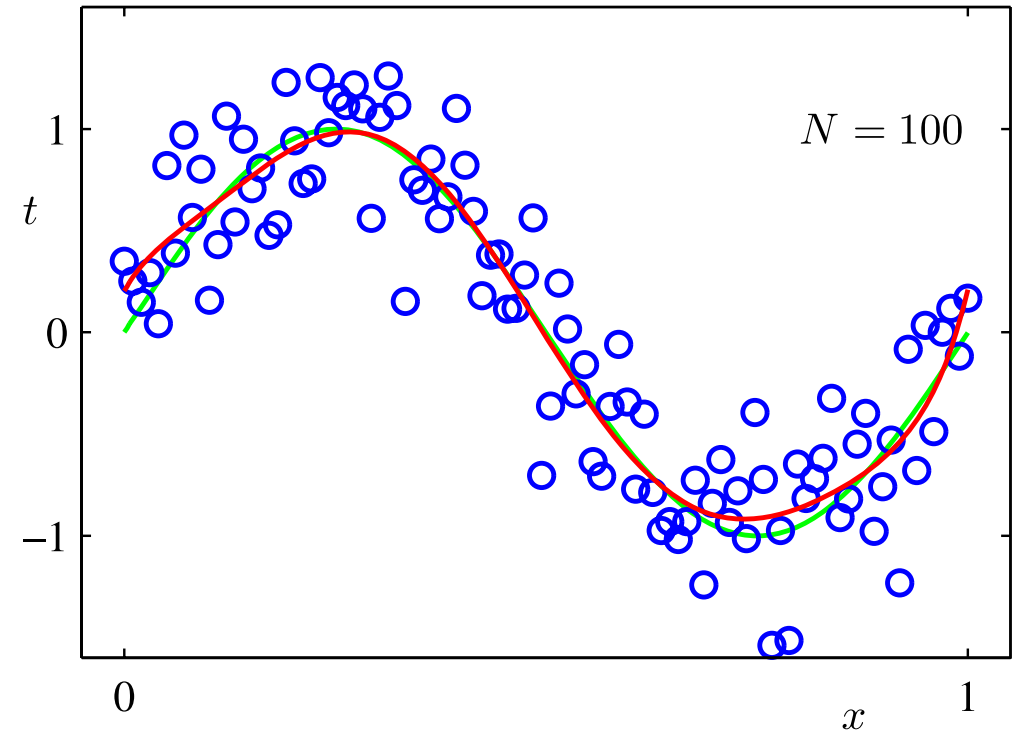
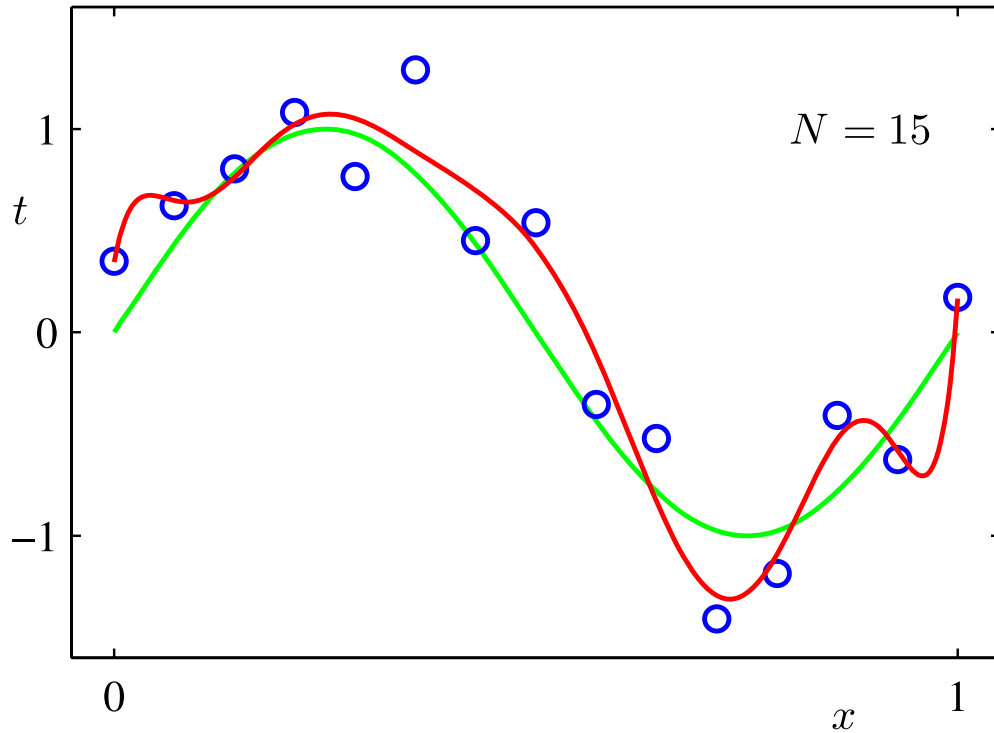


Figure 1.6 of Pattern Recognition and Machine Learning.

Regularization

Regularization in a broad sense is any change in a machine learning algorithm that is designed to *reduce generalization error* (but not necessarily its training error).

L_2 regularization (also called weighted decay) penalizes models with large weights:

$$\frac{1}{2} \sum_{i=1}^N (y(\mathbf{x}_i; \mathbf{w}) - t_i)^2 + \frac{\lambda}{2} \|\mathbf{w}\|^2$$

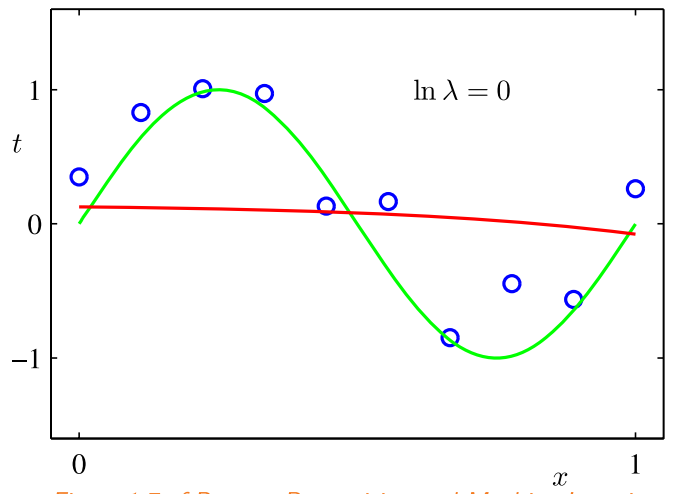
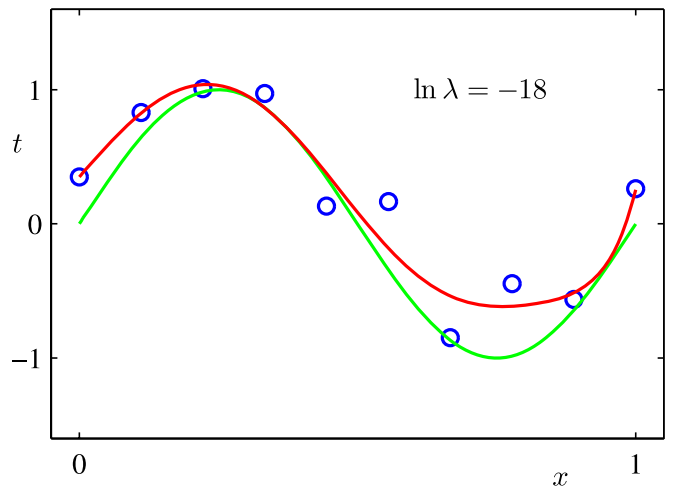


Figure 1.7 of Pattern Recognition and Machine Learning.

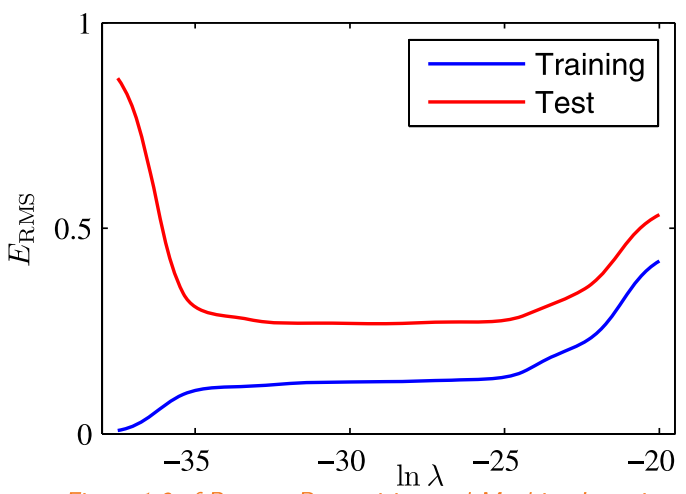


Figure 1.8 of Pattern Recognition and Machine Learning.

Regularizing Linear Regression

In matrix form, regularized sum of squares error for linear regression amounts to

$$\frac{1}{2} \|\mathbf{X}\mathbf{w} - \mathbf{t}\|^2 + \frac{\lambda}{2} \|\mathbf{w}\|^2.$$

When repeating the same calculation as in the unregularized case, we arrive at

$$(\mathbf{X}^T \mathbf{X} + \lambda \mathbf{I})\mathbf{w} = \mathbf{X}^T \mathbf{t},$$

where \mathbf{I} is an identity matrix.

Input: Dataset $(\mathbf{X} \in \mathbb{R}^{N \times D}, \mathbf{t} \in \mathbb{R}^N)$, constant $\lambda \in \mathbb{R}^+$.

Output: Weights $\mathbf{w} \in \mathbb{R}^D$ minimizing MSE of regularized linear regression.

- $\mathbf{w} \leftarrow (\mathbf{X}^T \mathbf{X} + \lambda \mathbf{I})^{-1} \mathbf{X}^T \mathbf{t}.$

Choosing Hyperparameters

Hyperparameters are not adapted by the learning algorithm itself.

Usually a **validation set** or **development set** is used to estimate the generalization error, allowing to update hyperparameters accordingly. If there is not enough data (well, there is **always** not enough data), more sophisticated approaches can be used.

So far, we have seen two hyperparameters, M and λ .

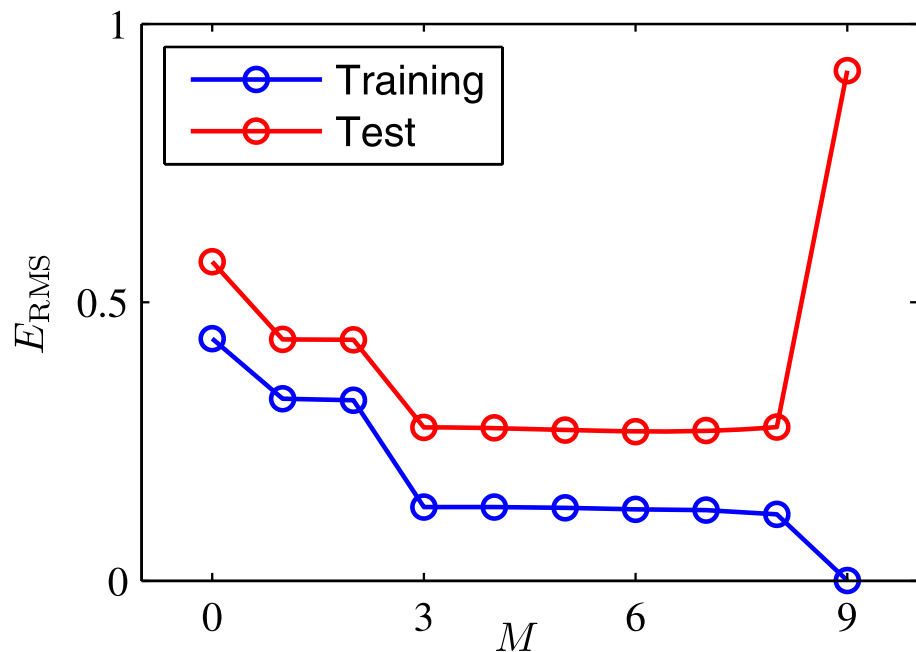


Figure 1.5 of Pattern Recognition and Machine Learning.

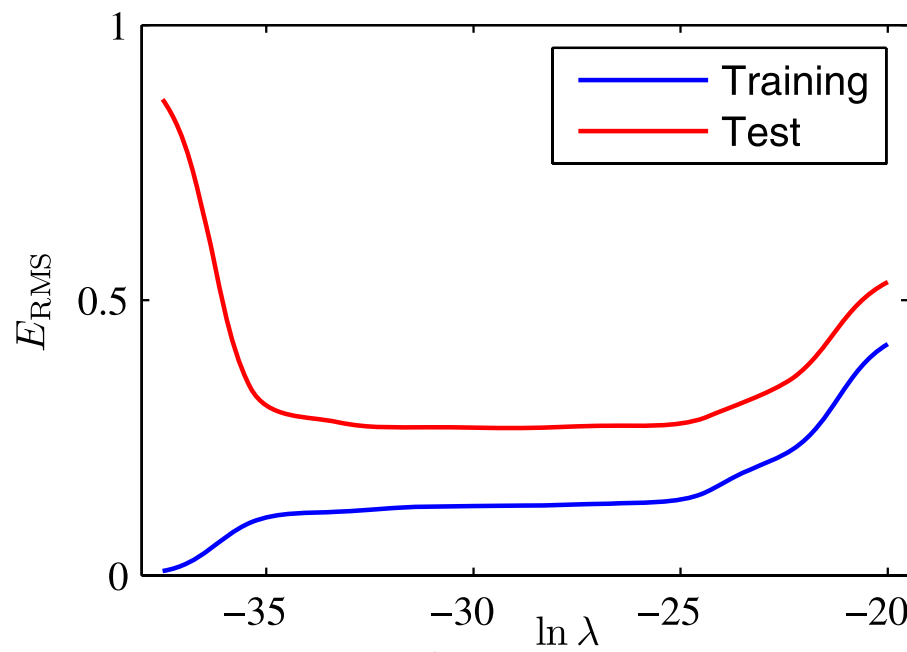


Figure 1.8 of Pattern Recognition and Machine Learning.

When training a linear regression model, we minimized the sum of squares error function by computing its gradient (partial derivatives with respect to all weights) and found solution when it is equal to zero, arriving at the following equation for optimal weights:

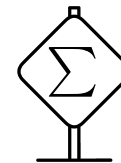
$$\mathbf{X}^T \mathbf{X} \mathbf{w} = \mathbf{X}^T \mathbf{t}.$$

If $\mathbf{X}^T \mathbf{X}$ is regular, we can invert it and compute the weights as $\mathbf{w} = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{t}$.

If you recall that $\text{rank}(\mathbf{X}) = \text{rank}(\mathbf{X}^T \mathbf{X})$, matrix $\mathbf{X}^T \mathbf{X} \in \mathbb{R}^{D \times D}$ is regular if and only if \mathbf{X} has rank D , which is equivalent to the columns of \mathbf{X} being linearly independent.

SVD Solution of Linear Regression

Now consider the case that $\mathbf{X}^T \mathbf{X}$ is singular. We will show that $\mathbf{X}^T \mathbf{X} \mathbf{w} = \mathbf{X}^T \mathbf{t}$ is still solvable, but it does not have a unique solution. Our goal in this case will be to find the \mathbf{w} with minimum $\|\mathbf{w}\|^2$ fulfilling the equation.



We now consider *singular value decomposition (SVD)* of \mathbf{X} , writing $\mathbf{X} = \mathbf{U} \mathbf{\Sigma} \mathbf{V}^T$, where

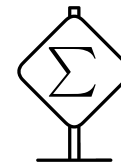
- $\mathbf{U} \in \mathbb{R}^{N \times N}$ is an orthogonal matrix, i.e., $\mathbf{u}_i^T \mathbf{u}_j = [i = j]$,
- $\mathbf{\Sigma} \in \mathbb{R}^{N \times D}$ is a diagonal matrix,
- $\mathbf{V} \in \mathbb{R}^{D \times D}$ is again an orthogonal matrix.

Assuming the diagonal matrix $\mathbf{\Sigma}$ has rank r , we can write it as

$$\mathbf{\Sigma} = \begin{bmatrix} \mathbf{\Sigma}_r & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{bmatrix},$$

where $\mathbf{\Sigma}_r \in \mathbb{R}^{r \times r}$ is a regular diagonal matrix. Denoting \mathbf{U}_r and \mathbf{V}_r the matrix of first r columns of \mathbf{U} and \mathbf{V} , respectively, we can write $\mathbf{X} = \mathbf{U}_r \mathbf{\Sigma}_r \mathbf{V}_r^T$.

Using the decomposition $\mathbf{X} = \mathbf{U}_r \mathbf{\Sigma}_r \mathbf{V}_r^T$, we can rewrite the goal equation as



$$\mathbf{V}_r \mathbf{\Sigma}_r^T \mathbf{U}_r^T \mathbf{U}_r \mathbf{\Sigma}_r \mathbf{V}_r^T \mathbf{w} = \mathbf{V}_r \mathbf{\Sigma}_r^T \mathbf{U}_r^T \mathbf{t}.$$

A transposition of an orthogonal matrix is its inverse. Therefore, our submatrix \mathbf{U}_r fulfills that $\mathbf{U}_r^T \mathbf{U}_r = \mathbf{I}$, because $\mathbf{U}_r^T \mathbf{U}_r$ is a top left submatrix of $\mathbf{U}^T \mathbf{U}$. Analogously, $\mathbf{V}_r^T \mathbf{V}_r = \mathbf{I}$.

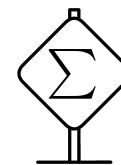
We therefore simplify the goal equation to

$$\mathbf{\Sigma}_r \mathbf{\Sigma}_r \mathbf{V}_r^T \mathbf{w} = \mathbf{\Sigma}_r \mathbf{U}_r^T \mathbf{t}$$

Because the diagonal matrix $\mathbf{\Sigma}_r$ is regular, we can divide by it and obtain

$$\mathbf{V}_r^T \mathbf{w} = \mathbf{\Sigma}_r^{-1} \mathbf{U}_r^T \mathbf{t}.$$

We have $\mathbf{V}_r^T \mathbf{w} = \mathbf{\Sigma}_r^{-1} \mathbf{U}_r^T \mathbf{t}$. If the original matrix $\mathbf{X}^T \mathbf{X}$ was regular, then $r = D$ and \mathbf{V}_r is a square regular orthogonal matrix, in which case



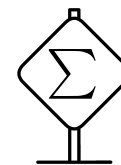
$$\mathbf{w} = \mathbf{V}_r \mathbf{\Sigma}_r^{-1} \mathbf{U}_r^T \mathbf{t}.$$

If we denote $\mathbf{\Sigma}^+ \in \mathbb{R}^{D \times N}$ the diagonal matrix with $\Sigma_{i,i}^{-1}$ on diagonal, we can rewrite to

$$\mathbf{w} = \mathbf{V} \mathbf{\Sigma}^+ \mathbf{U}^T \mathbf{t}.$$

Now if $r < D$, $\mathbf{V}_r^T \mathbf{w} = \mathbf{y}$ is undetermined and has infinitely many solutions. To find the one with smallest norm $\|\mathbf{w}\|$, consider the full product $\mathbf{V}^T \mathbf{w}$. Because \mathbf{V} is orthogonal, $\|\mathbf{V}^T \mathbf{w}\| = \|\mathbf{w}\|$, and it is sufficient to find \mathbf{w} with smallest $\|\mathbf{V}^T \mathbf{w}\|$. We know that the first r elements of $\|\mathbf{V}^T \mathbf{w}\|$ are fixed by the above equation – the smallest $\|\mathbf{V}^T \mathbf{w}\|$ can be therefore obtained by setting the last $D - r$ elements to zero. Finally, we note that $\mathbf{\Sigma}^+ \mathbf{U}^T \mathbf{t}$ is exactly $\mathbf{\Sigma}_r^{-1} \mathbf{U}_r^T \mathbf{t}$ padded with $D - r$ zeros, obtaining the same solution $\mathbf{w} = \mathbf{V} \mathbf{\Sigma}^+ \mathbf{U}^T \mathbf{t}$.

The solution to a linear regression with sum of squares error function is tightly connected to matrix pseudoinverses. If a matrix \mathbf{X} is singular or rectangular, it does not have an exact inverse, and $\mathbf{X}\mathbf{w} = \mathbf{b}$ does not have an exact solution.



However, we can consider the so-called *Moore-Penrose pseudoinverse*

$$\mathbf{X}^+ \stackrel{\text{def}}{=} \mathbf{V}\mathbf{\Sigma}^+\mathbf{U}^T$$

to be the closest approximation to an inverse, in the sense that we can find the best solution (with smallest MSE) to the equation $\mathbf{X}\mathbf{w} = \mathbf{b}$ by setting $\mathbf{w} = \mathbf{X}^+\mathbf{b}$.

Alternatively, we can define the pseudoinverse of a matrix \mathbf{X} as

$$\mathbf{X}^+ = \arg \min_{\mathbf{Y} \in \mathbb{R}^{D \times N}} \|\mathbf{X}\mathbf{Y} - \mathbf{I}_N\|_F = \arg \min_{\mathbf{Y} \in \mathbb{R}^{N \times D}} \|\mathbf{Y}\mathbf{X} - \mathbf{I}_D\|_F$$

which can be verified to be the same as our SVD formula.

A random variable \mathbf{x} is a result of a random process. It can be discrete or continuous.

Probability Distribution

A probability distribution describes how likely are individual values a random variable can take.

The notation $\mathbf{x} \sim P$ stands for a random variable \mathbf{x} having a distribution P .

For discrete variables, the probability that \mathbf{x} takes a value x is denoted as $P(x)$ or explicitly as $P(\mathbf{x} = x)$. All probabilities are non-negative and sum of probabilities of all possible values of \mathbf{x} is $\sum_x P(\mathbf{x} = x) = 1$.

For continuous variables, the probability that the value of \mathbf{x} lies in the interval $[a, b]$ is given by $\int_a^b p(x) dx$.

Expectation

The expectation of a function $f(x)$ with respect to discrete probability distribution $P(x)$ is defined as:

$$\mathbb{E}_{x \sim P}[f(x)] \stackrel{\text{def}}{=} \sum_x P(x) f(x)$$

For continuous variables it is computed as:

$$\mathbb{E}_{x \sim p}[f(x)] \stackrel{\text{def}}{=} \int_x p(x) f(x) dx$$

If the random variable is obvious from context, we can write only $\mathbb{E}_P[x]$ or even $\mathbb{E}[x]$.

Expectation is linear, i.e.,

$$\mathbb{E}_x[\alpha f(x) + \beta g(x)] = \alpha \mathbb{E}_x[f(x)] + \beta \mathbb{E}_x[g(x)]$$

Variance

Variance measures how much the values of a random variable differ from its mean $\mu = \mathbb{E}[x]$.

$$\text{Var}(x) \stackrel{\text{def}}{=} \mathbb{E} \left[(x - \mathbb{E}[x])^2 \right], \text{ or more generally}$$

$$\text{Var}(f(x)) \stackrel{\text{def}}{=} \mathbb{E} \left[(f(x) - \mathbb{E}[f(x)])^2 \right]$$

It is easy to see that

$$\text{Var}(x) = \mathbb{E} \left[x^2 - 2x\mathbb{E}[x] + (\mathbb{E}[x])^2 \right] = \mathbb{E} [x^2] - (\mathbb{E}[x])^2,$$

because $\mathbb{E} [2x\mathbb{E}[x]] = 2(\mathbb{E}[x])^2$.

Variance is connected to $\mathbb{E}[x^2]$, a *second moment* of a random variable – it is in fact a *centered* second moment.

Estimators and Bias

An *estimator* is a rule for computing an estimate of a given value, often an expectation of some random value(s).

For example, we might estimate *mean* of random variable by sampling a value according to its probability distribution.

Bias of an estimator is the difference of the expected value of the estimator and the true value being estimated:

$$\text{bias} = \mathbb{E}[\text{estimate}] - \text{true estimated value.}$$

If the bias is zero, we call the estimator *unbiased*, otherwise we call it *biased*.

Estimators and Bias

If we have a sequence of estimates, it also might happen that the bias converges to zero. Consider the well known sample estimate of variance. Given x_1, \dots, x_n independent and identically distributed random variables, we might estimate mean and variance as

$$\hat{\mu} = \frac{1}{n} \sum_i x_i, \quad \hat{\sigma}^2 = \frac{1}{n} \sum_i (x_i - \hat{\mu})^2.$$

Such estimate is biased, because $\mathbb{E}[\hat{\sigma}^2] = (1 - \frac{1}{n})\sigma^2$, but the bias converges to zero with increasing n .

Also, an unbiased estimator does not necessarily have small variance – in some cases it can have large variance, so a biased estimator with smaller variance might be preferred.

Gradient Descent

Sometimes it is more practical to search for the best model weights in an iterative/incremental/sequential fashion. Either because there is too much data, or the direct optimization is not feasible.

Assuming we are minimizing an error function

$$\arg \min_{\mathbf{w}} E(\mathbf{w}),$$

we may use *gradient descent*:

$$\mathbf{w} \leftarrow \mathbf{w} - \alpha \nabla_{\mathbf{w}} E(\mathbf{w})$$

The constant α is called a *learning rate* and specifies the “length” of a step we perform in every iteration of the gradient descent.

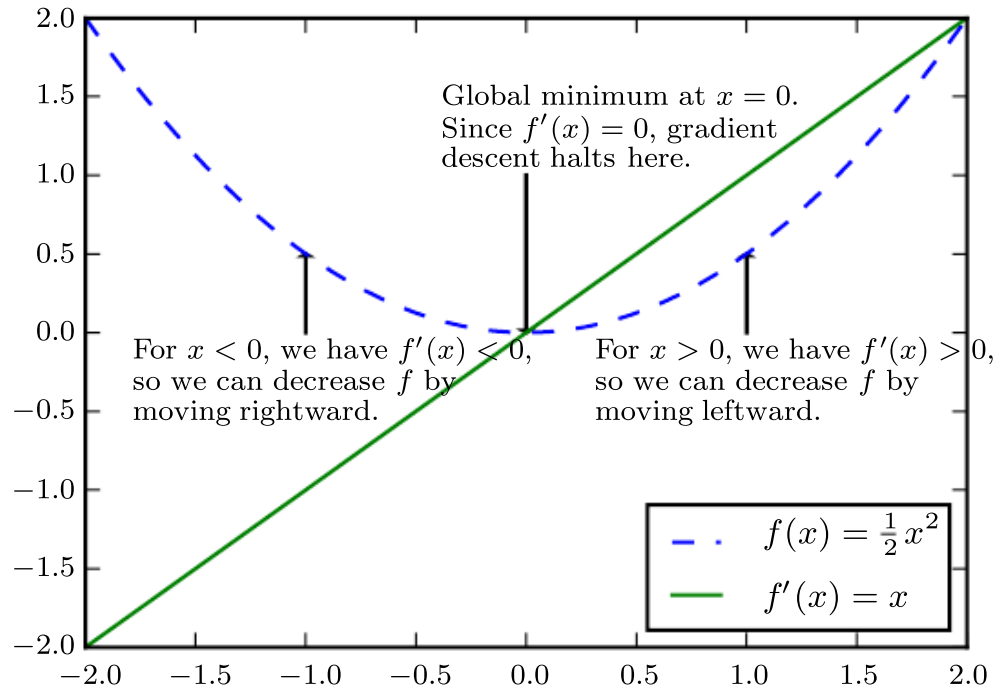


Figure 4.1, page 83 of Deep Learning Book, <http://deeplearningbook.org>

Gradient Descent Variants

Consider an error function computed as an expectation over the dataset:

$$\nabla_{\mathbf{w}} E(\mathbf{w}) = \nabla_{\mathbf{w}} \mathbb{E}_{(\mathbf{x}, t) \sim \hat{p}_{\text{data}}} L(y(\mathbf{x}; \mathbf{w}), t).$$

- **(Regular) Gradient Descent:** We use all training data to compute $\nabla_{\mathbf{w}} E(\mathbf{w})$ exactly.
- **Online (or Stochastic) Gradient Descent:** We estimate $\nabla_{\mathbf{w}} E(\mathbf{w})$ using a single random example from the training data. Such an estimate is unbiased, but very noisy.

$$\nabla_{\mathbf{w}} E(\mathbf{w}) \approx \nabla_{\mathbf{w}} L(y(\mathbf{x}; \mathbf{w}), t) \text{ for randomly chosen } (\mathbf{x}, t) \text{ from } \hat{p}_{\text{data}}.$$

- **Minibatch SGD:** The minibatch SGD is a trade-off between gradient descent and SGD – the expectation in $\nabla_{\mathbf{w}} E(\mathbf{w})$ is estimated using m random independent examples from the training data.

$$\nabla_{\mathbf{w}} E(\mathbf{w}) \approx \frac{1}{m} \sum_{i=1}^m \nabla_{\mathbf{w}} L(y(\mathbf{x}_i; \mathbf{w}), t_i) \text{ for randomly chosen } (\mathbf{x}_i, t_i) \text{ from } \hat{p}_{\text{data}}.$$

Gradient Descent Convergence

Assume that we perform a stochastic gradient descent, using a sequence of learning rates α_i , and using a noisy estimate $J(\mathbf{w})$ of the real gradient $\nabla_{\mathbf{w}} E(\mathbf{w})$:

$$\mathbf{w}_{i+1} \leftarrow \mathbf{w}_i - \alpha_i J(\mathbf{w}_i).$$

It can be proven (under some reasonable conditions; see Robbins and Monro algorithm, 1951) that if the loss function L is convex and continuous, then SGD converges to the unique optimum almost surely if the sequence of learning rates α_i fulfills the following conditions:

$$\forall i : \alpha_i > 0, \quad \sum_i \alpha_i = \infty, \quad \sum_i \alpha_i^2 < \infty.$$

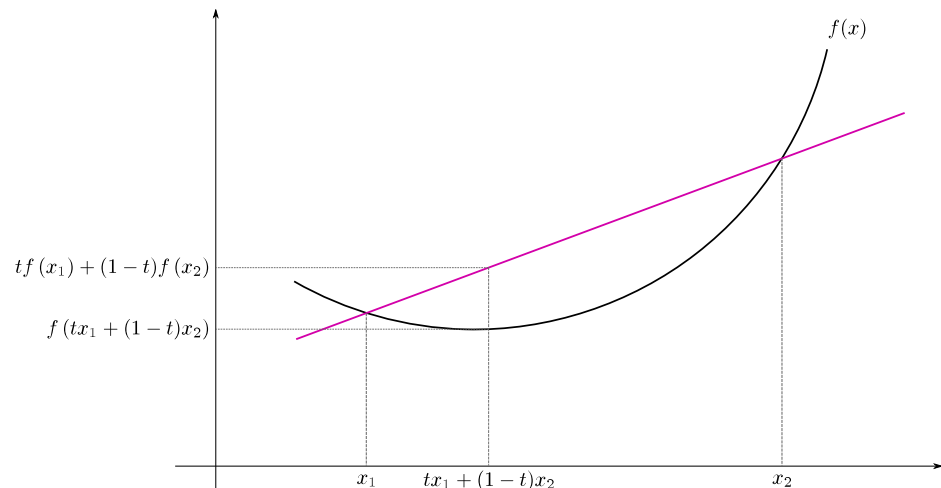
Note that the third condition implies that $\alpha_i \rightarrow 0$.

For non-convex loss functions, we can get guarantees of converging to a *local* optimum only. However, note that finding a global minimum of an arbitrary function is *at least NP-hard*.

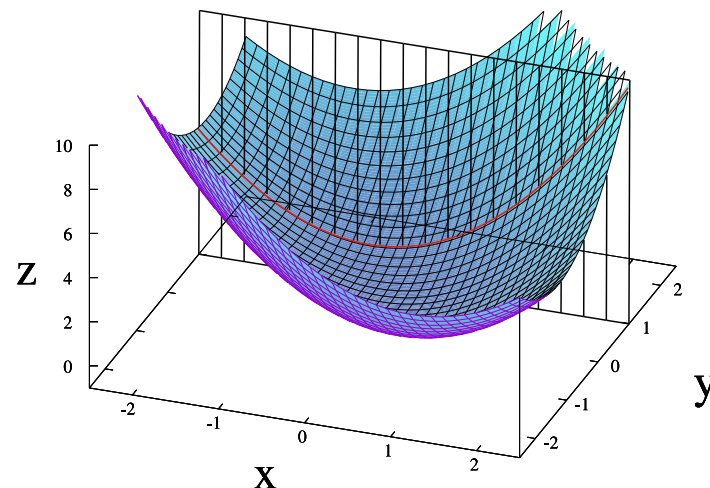
Gradient Descent Convergence

Convex functions mentioned on a previous slide are such that for x_1, x_2 and real $0 \leq t \leq 1$,

$$f(tx_1 + (1 - t)x_2) \leq tf(x_1) + (1 - t)f(x_2).$$



<https://upload.wikimedia.org/wikipedia/commons/c/c7/ConvexFunction.svg>



https://commons.wikimedia.org/wiki/File:Partial_func_eg.svg

A twice-differentiable function is convex iff its second derivative is always non-negative.

A local minimum of a convex function is always the unique global minimum.

Well-known examples of convex functions are x^2 , e^x and $-\log x$.

Solving Linear Regression using SGD

To apply SGD on linear regression, we usually minimize one half of mean squared error:

$$E(\mathbf{w}) = \mathbb{E}_{(\mathbf{x}, t) \sim \hat{p}_{\text{data}}} \left[\frac{1}{2} (y(\mathbf{x}; \mathbf{w}) - t)^2 \right] = \mathbb{E}_{(\mathbf{x}, t) \sim \hat{p}_{\text{data}}} \left[\frac{1}{2} (\mathbf{x}^T \mathbf{w} - t)^2 \right].$$

If we also include L_2 regularization, we get

$$E(\mathbf{w}) = \mathbb{E}_{(\mathbf{x}, t) \sim \hat{p}_{\text{data}}} \left[\frac{1}{2} (\mathbf{x}^T \mathbf{w} - t)^2 \right] + \frac{\lambda}{2} \|\mathbf{w}\|^2.$$

We then estimate the expectation by a minibatch \mathbf{b} of examples as

$$\sum_{i \in \mathbf{b}} \frac{1}{|\mathbf{b}|} \left(\frac{1}{2} (\mathbf{x}_i^T \mathbf{w} - t_i)^2 \right) + \frac{\lambda}{2} \|\mathbf{w}\|^2,$$

which gives us an estimate of a gradient

$$\nabla_{\mathbf{w}} E(\mathbf{w}) \approx \sum_{i \in \mathbf{b}} \frac{1}{|\mathbf{b}|} \left((\mathbf{x}_i^T \mathbf{w} - t_i) \mathbf{x}_i \right) + \lambda \mathbf{w}.$$

Solving Linear Regression using SGD

The computed gradient allows us to formulate the following algorithm for solving linear regression with minibatch SGD.

Input: Dataset $(\mathbf{X} \in \mathbb{R}^{N \times D}, \mathbf{t} \in \mathbb{R}^N)$, learning rate $\alpha \in \mathbb{R}^+$, L_2 strength $\lambda \in \mathbb{R}$.

Output: Weights $\mathbf{w} \in \mathbb{R}^D$ which hopefully minimize regularized MSE of linear regression.

- $\mathbf{w} \leftarrow \mathbf{0}$
- repeat until convergence (or until our patience runs out):
 - sample a batch \mathbf{b} (either uniformly randomly; or we may want to process all training instances before repeating them, which can be implemented by generating a random permutation and then splitting it to batch-sizes chunks)
 - $\mathbf{w} \leftarrow \mathbf{w} - \alpha \sum_{i \in \mathbf{b}} \frac{1}{|\mathbf{b}|} ((\mathbf{x}_i^T \mathbf{w} - t_i) \mathbf{x}_i) - \alpha \lambda \mathbf{w}$

Features

Recall that the *input* instance values are usually the raw observations and are given. However, we might extend them suitably before running a machine learning algorithm, especially if the algorithm is linear or otherwise limited and cannot represent arbitrary function. Such instance representations is called *features*.

We already saw this in the example from the previous lecture, where even if our training examples were x and t , we performed the linear regression using features (x^0, x^1, \dots, x^M) :

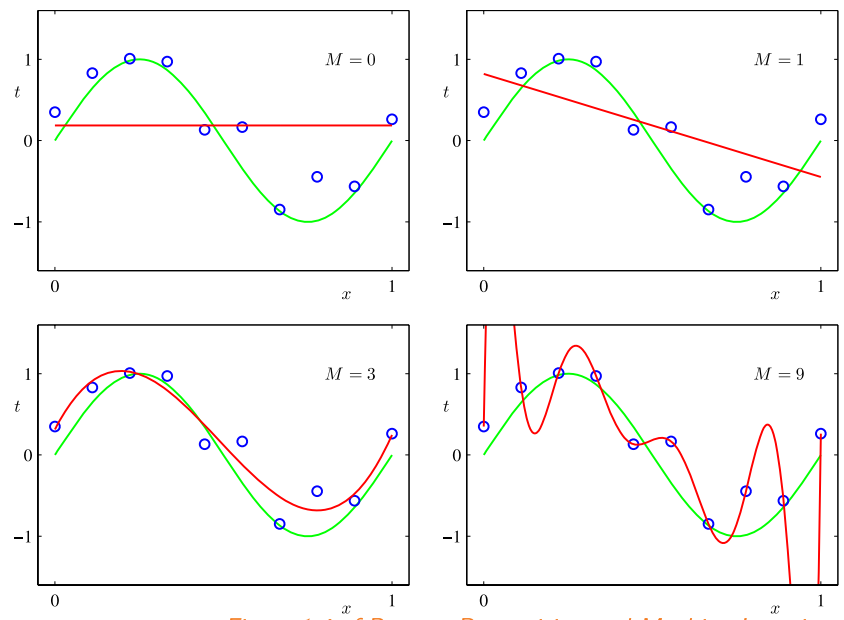


Figure 1.4 of Pattern Recognition and Machine Learning.

Generally, it would be best if we have machine learning algorithms processing only the raw inputs. However, many algorithms are capable of representing only a limited set of functions (for example linear ones), and in that case, *feature engineering* plays a major part in the final model performance. Feature engineering is a process of constructing features from raw inputs.

Commonly used features are:

- **polynomial features** of degree p : Given features (x_1, x_2, \dots, x_D) , we might consider *all* products of p input values. Therefore, polynomial features of degree 2 would consist of $x_i^2 \forall i$ and of $x_i x_j \forall i \neq j$.
- **categorical one-hot features**: Assume for example that a day in a week is represented on the input as an integer value of 1 to 7, or a breed of a dog is expressed as an integer value of 0 to 366. Using these integral values as input to linear regression makes little sense – instead it might be better to learn weights for individual days in a week or for individual dog breeds. We might therefore represent input classes by binary indicators for every class, giving rise to **one-hot** representation, where input integral value $1 \leq v \leq L$ is represented as L binary values, which are all zero except for the v -th one, which is one.