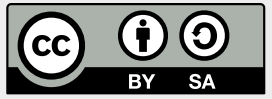
Linear Regression II, SGD

Jindřich Libovický (reusing materials by Milan Straka)

October 10, 2023

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



Today's Lecture Objectives

After this lecture you should be able to

- Reason about **overfitting** in terms of **model capacity**.
- Use L^2**-regularization** to control model capacity.
- Explain what the difference between **parameters and hyperparameters** is.
- Tell what the **basic probability concepts** are (joint, marginal, conditional probability; expected value, mean, variance).
- Mathematically describe and implement the **stochastic gradient descent** algorithm.
- Use both **numerical and categorical features** in linear regression.

Linear Regression

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

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

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

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

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

Various minimization approaches exist, but for linear regression an explicit solution exists:

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

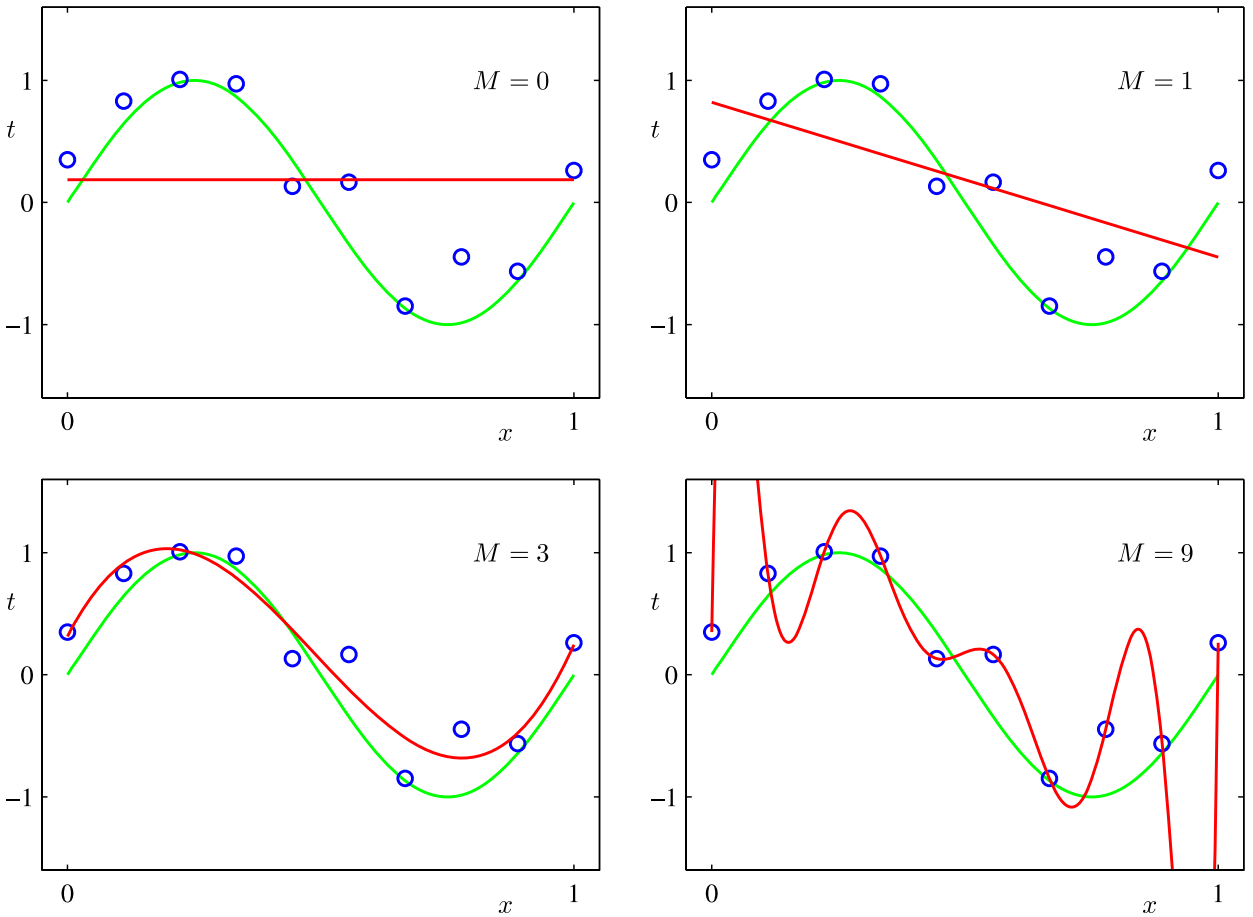
Linear Regression Example

Assume we want to predict a $t \in \mathbb{R}$ for a given $x \in \mathbb{R}$. If we train the linear regression with "raw" input vectors $\boldsymbol{x} = (x)$, only straight lines could be modeled.

However, if we consider input vectors $\boldsymbol{x} = (x^0, x^1, \ldots, x^M)$ for a given $M \geq 0$, the linear regression can model polynomials of degree M, because the prediction is then computed as

$$w_0 x^0 + w_1 x^1 + \ldots + w_M x^M.$$

The weights are coefficients of a polynomial of degree M.



Linear Regression Example

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

The displayed error illustrates two main challenges in machine learning:

- *underfitting*
- *overfitting*

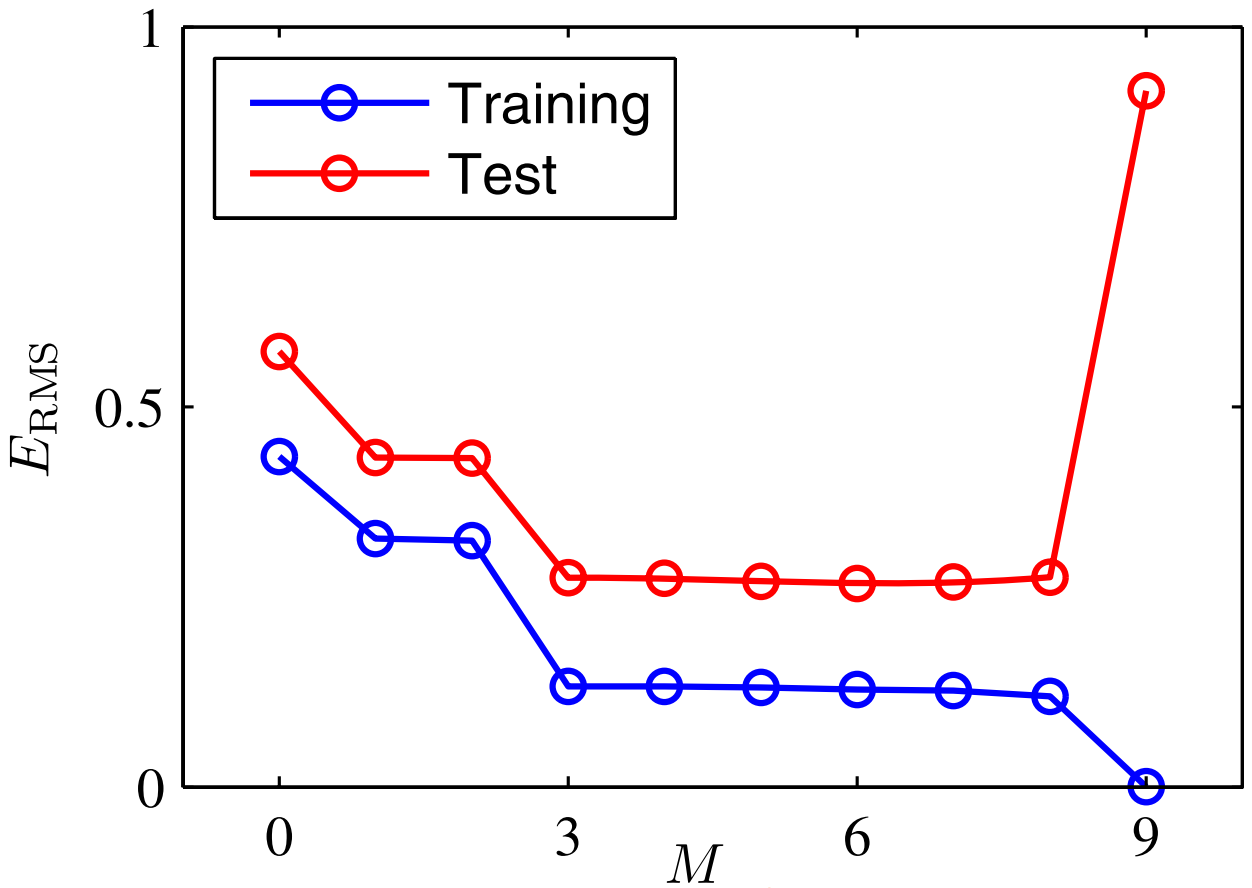


Figure 1.5 of Pattern Recognition and Machine Learning.

Model Capacity

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

- representational capacity
- effective capacity

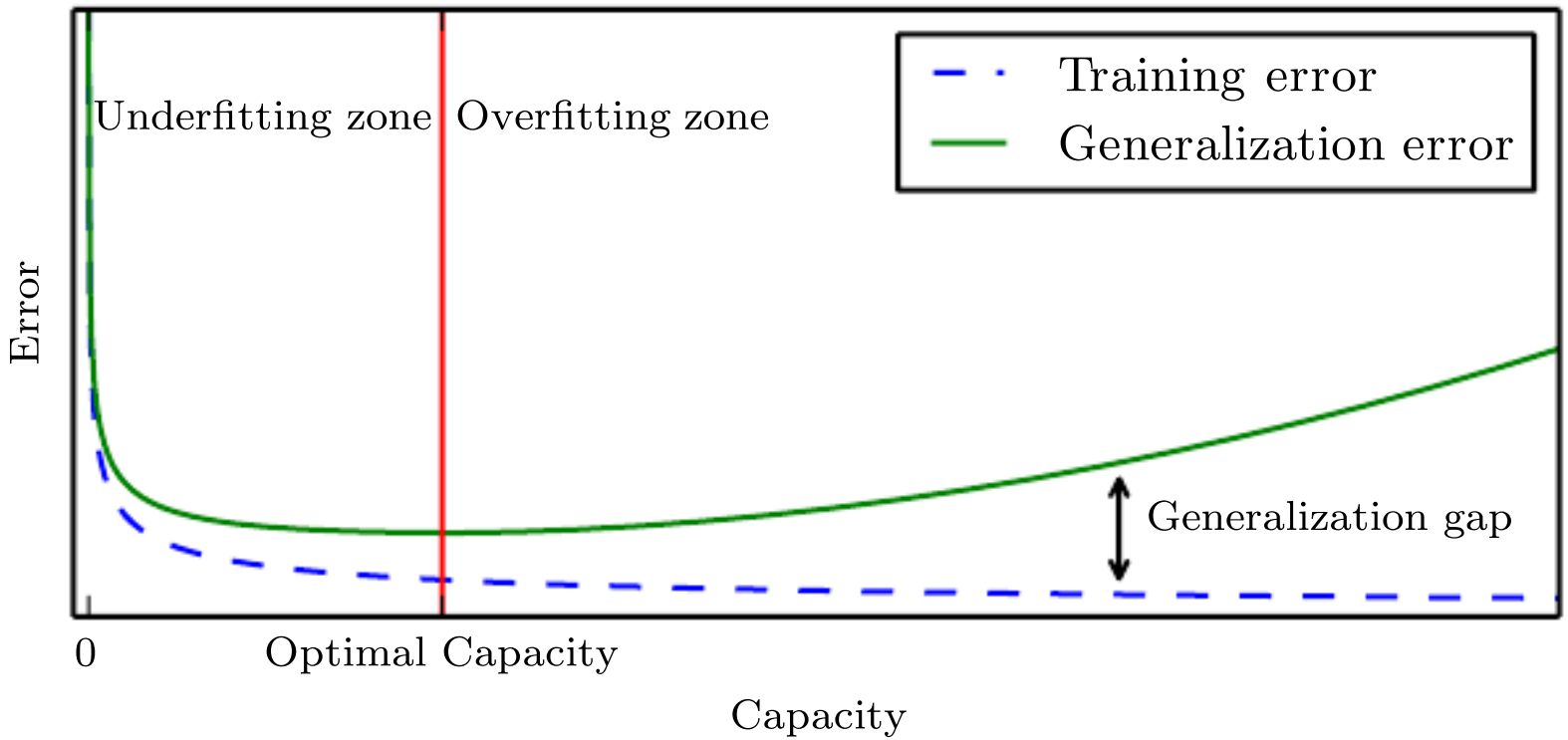
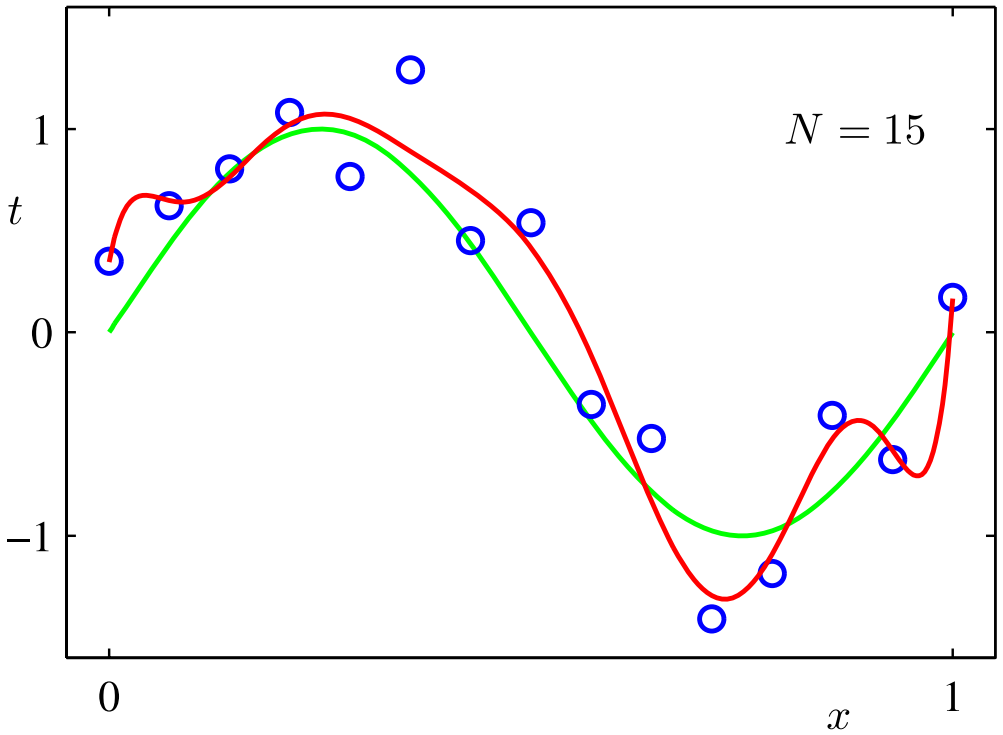


Figure 5.3 of "Deep Learning" book, https://www.deeplearningbook.org

Linear Regression Overfitting

Employing more data usually alleviates overfitting (the relative capacity of the model is decreased).



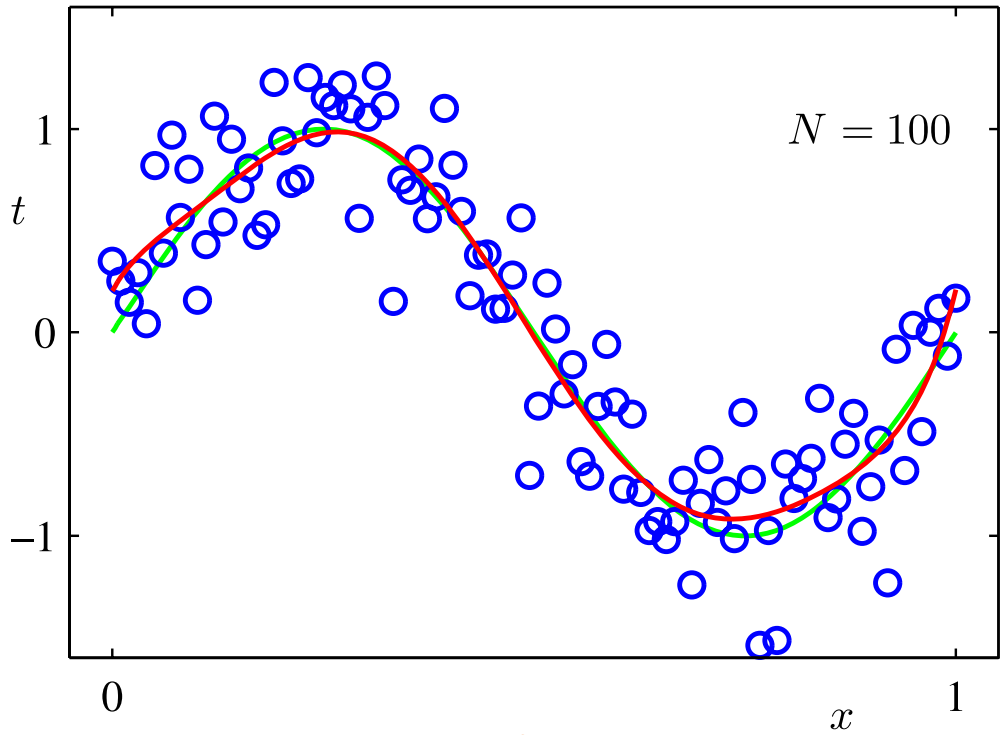
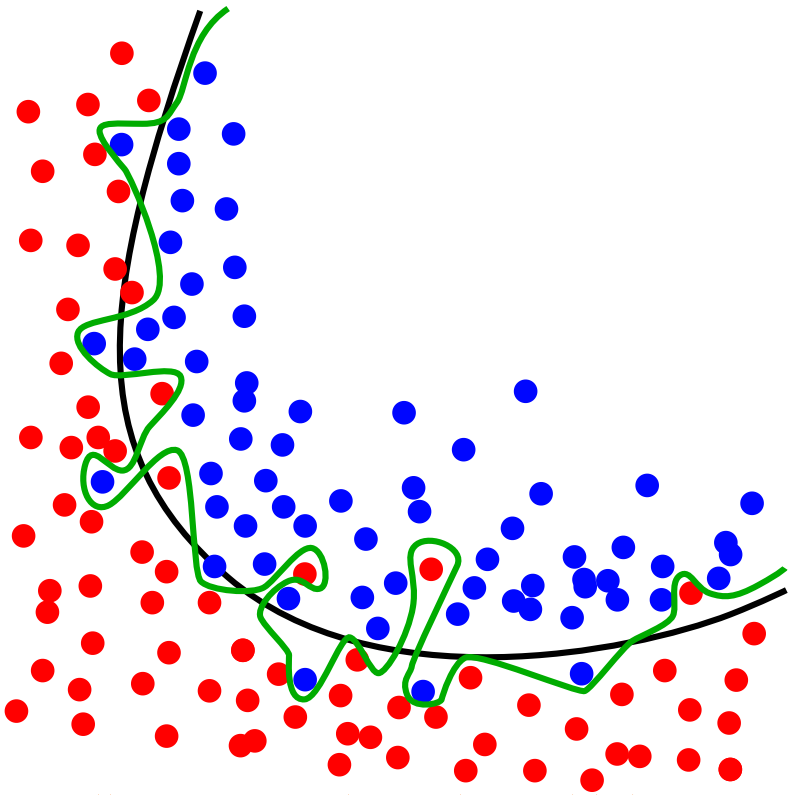


Figure 1.6 of Pattern Recognition and Machine Learning.

Regularization

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

We already saw that **limiting model capacity** can work as regularization.



https://upload.wikimedia.org/wikipedia/commons/1/19/Overfitting.svg

L2 Regularization

L^2-regularization: one of the oldest regularization techniques; tries to prefer "simpler" models by endorsing models with **smaller weights**.

L^2-regularization (also called **weight decay**) penalizes models with large weights by utilizing the following error function:

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

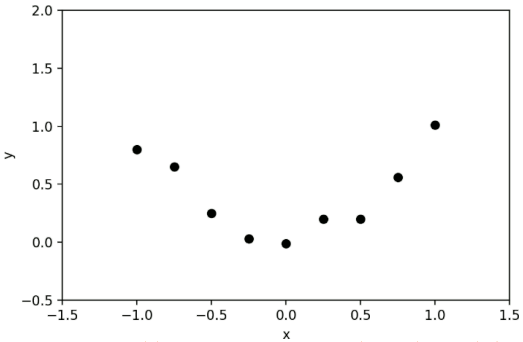
The L^2-regularization is usually not applied to the *bias*, only to the "proper" weights, because we cannot really overfit via the bias. Without penalizing the bias, linear regression with L^2-regularization is invariant to shifts (i.e., adding a constant to all the targets results in the same solution, only with the bias increased by that constant; if the bias were penalized, this would not hold).

We will not explicitly exclude the bias from the L^2-regularization penalty in the slides.

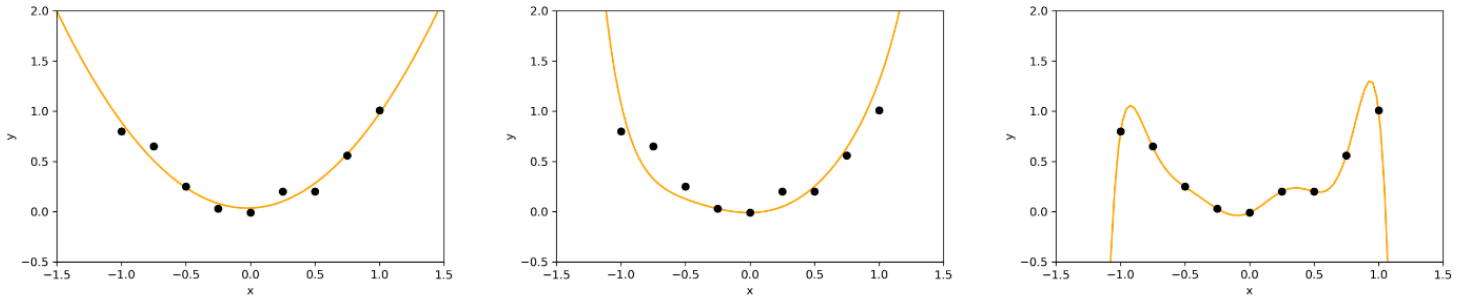
L2 Regularization

One way to look at L^2-regularization: it promotes smaller changes of the model (the gradient of linear regression with respect to the inputs are exactly the weights, i.e., $\nabla_{\boldsymbol{x}} y(\boldsymbol{x}; \boldsymbol{w}) = \boldsymbol{w}$).

Considering the data points on the right, we present mean squared errors and L^2 norms of the weights for three linear regression models:



https://miro.medium.com/max/2880/1*0-fsK9RkqL3rogo2SnZPCg.png



(a) #params = 3
MSE = 0.006
L2 norm = 0.90
L1 norm = 0.98

(b) #params = 9
MSE = 0.035
L2 norm = 1.06
L1 norm = 2.32

(c) #params = 9
MSE = 0
L2 norm = 32.69
L1 norm = 70.03

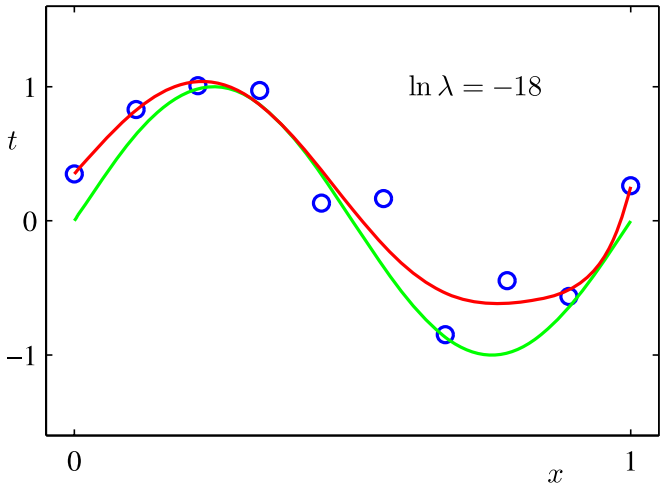
https://miro.medium.com/max/2880/1*DVFYChNDMNlS_7CVq2PhSQ.png

Figure a: $\hat{y} = 0.04 + 0.04x + 0.9x^2$
Figure b: $\hat{y} = -0.01 + 0.01x + 0.8x^2 + 0.5x^3 - 0.1x^4 - 0.1x^5 + 0.3x^6 - 0.3x^7 + 0.2x^8$
Figure c: $\hat{y} = -0.01 + 0.57x + 2.67x^2 - 4.08x^3 - 12.25x^4 + 7.41x^5 + 24.87x^6 - 3.79x^7 - 14.38x^8$

https://miro.medium.com/max/2880/1*UolRlKXikCz7SFsPfSZrYQ.png

L2 Regularization

The effect of L^2-regularization can be seen as limiting the *effective capacity* of the model.



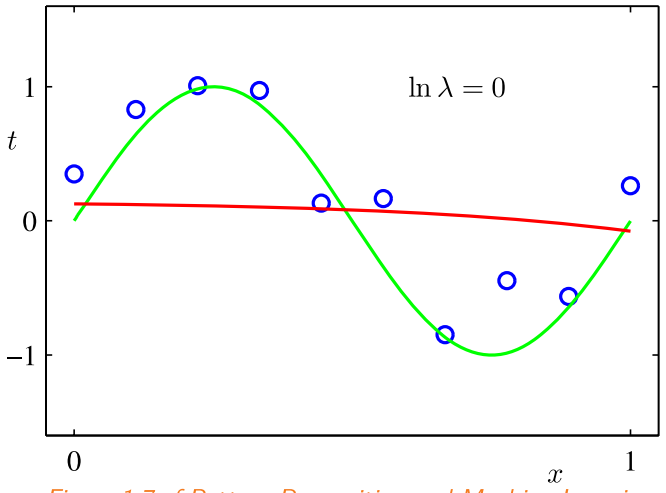


Figure 1.7 of Pattern Recognition and Machine Learning.

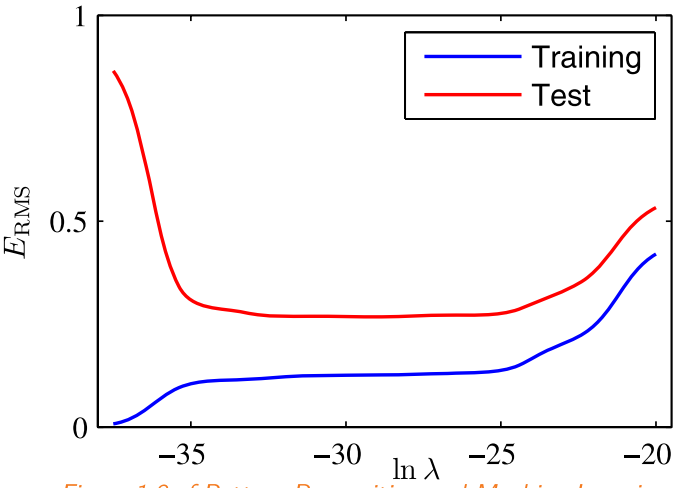


Figure 1.8 of Pattern Recognition and Machine Learning.

Regularizing Linear Regression

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

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

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

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

where $\boldsymbol{I}$ is an identity matrix.

Input: Dataset ($\boldsymbol{X} \in \mathbb{R}^{N\times D}$, $\boldsymbol{t} \in \mathbb{R}^N$), constant $\lambda \in \mathbb{R}^+$.
Output: Weights $\boldsymbol{w} \in \mathbb{R}^D$ minimizing MSE of regularized linear regression.

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

The matrix $\boldsymbol{X}^T\boldsymbol{X} + \lambda\boldsymbol{I}$ is always regular for $\lambda > 0$ (you can show that the matrix is positive definite).

Choosing Hyperparameters

Hyperparameters are not adapted by the learning algorithm itself.

A **validation set** or **development set** is used to estimate the generalization error, allowing us 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: two hyperparameters, M and λ.

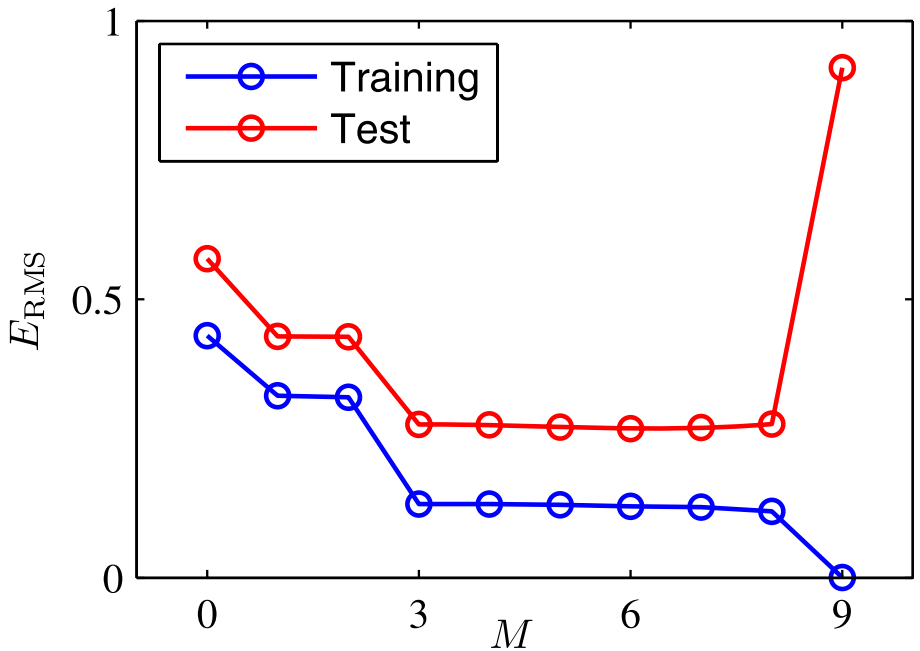


Figure 1.5 of Pattern Recognition and Machine Learning.

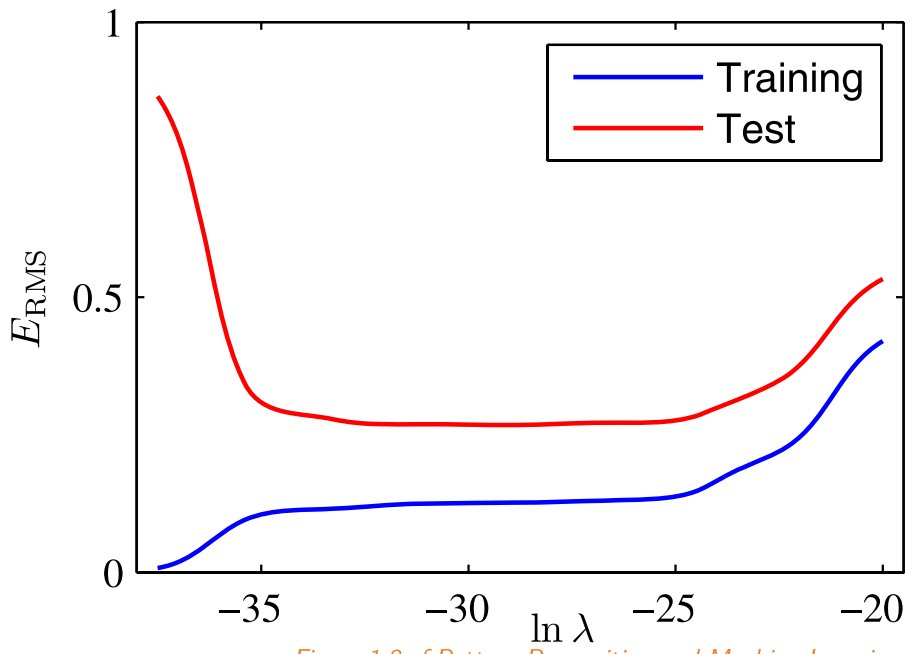
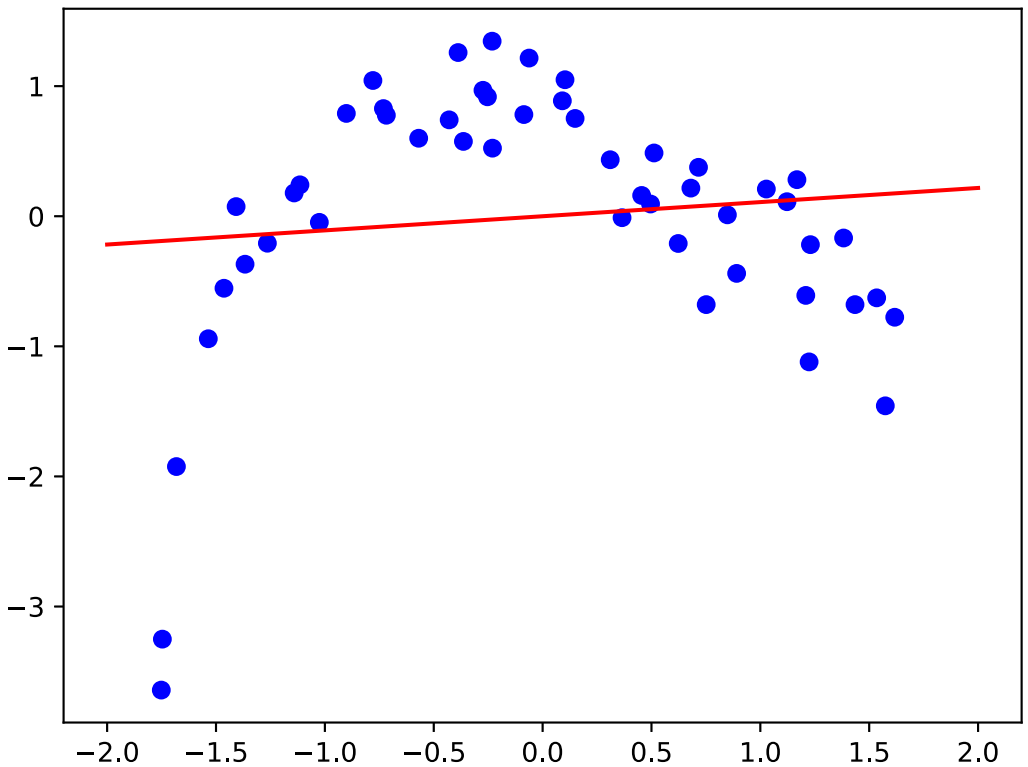


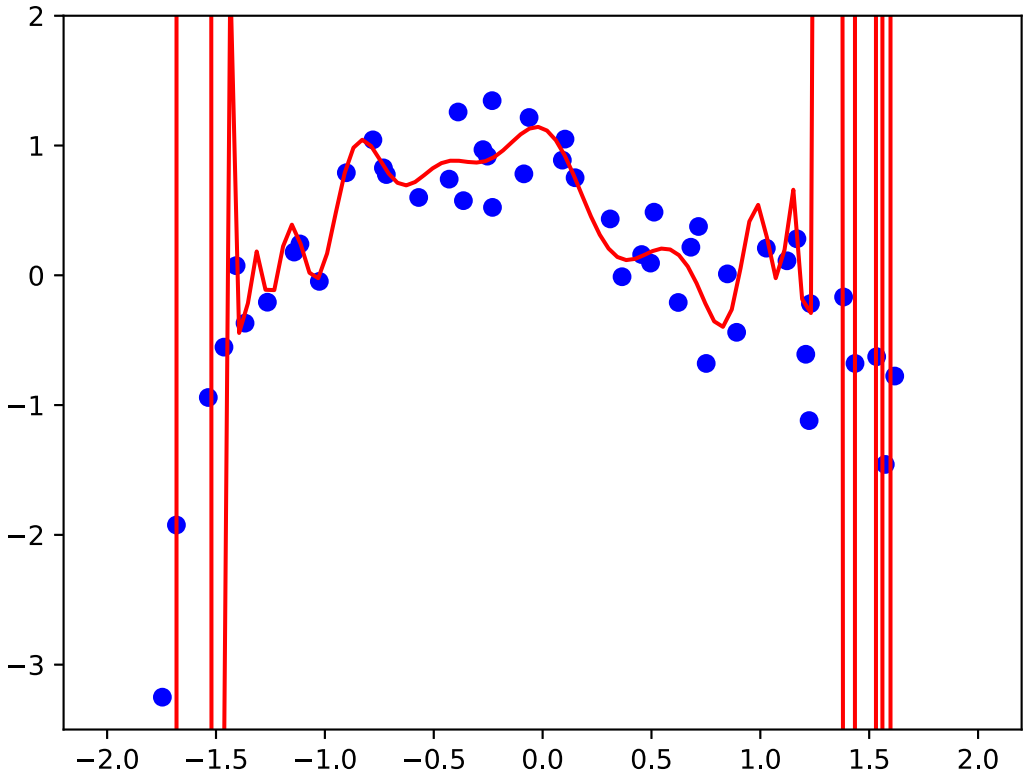
Figure 1.8 of Pattern Recognition and Machine Learning.

What went wrong? (1/5)



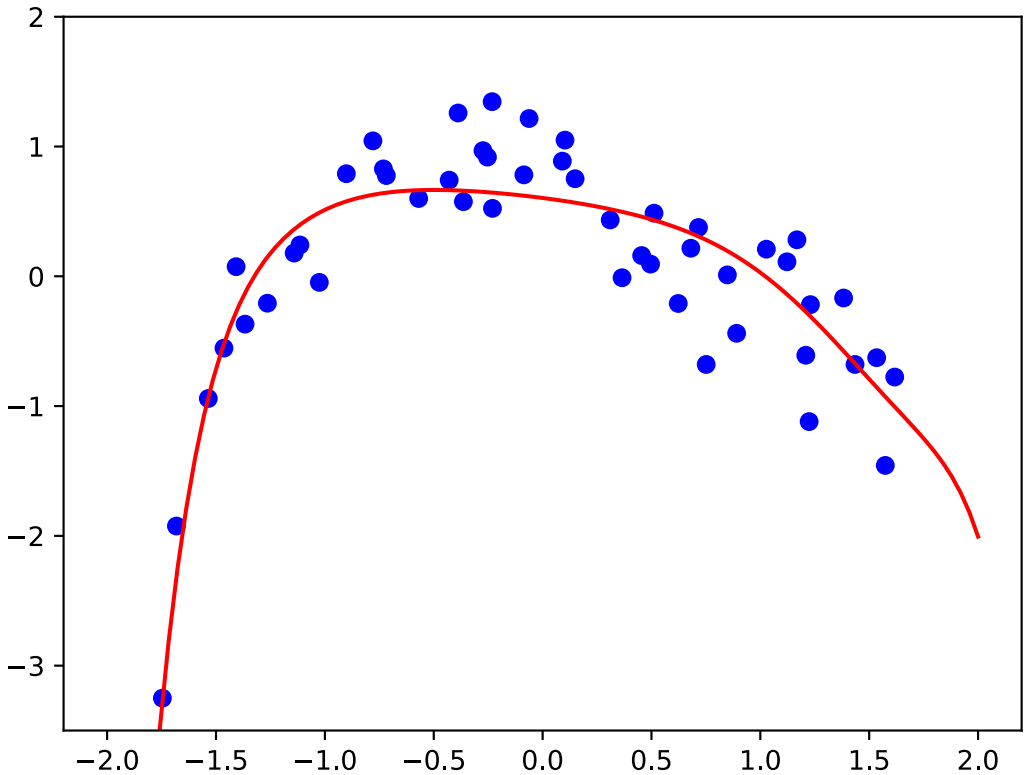
Too small model capacity.

What went wrong? (2/5)



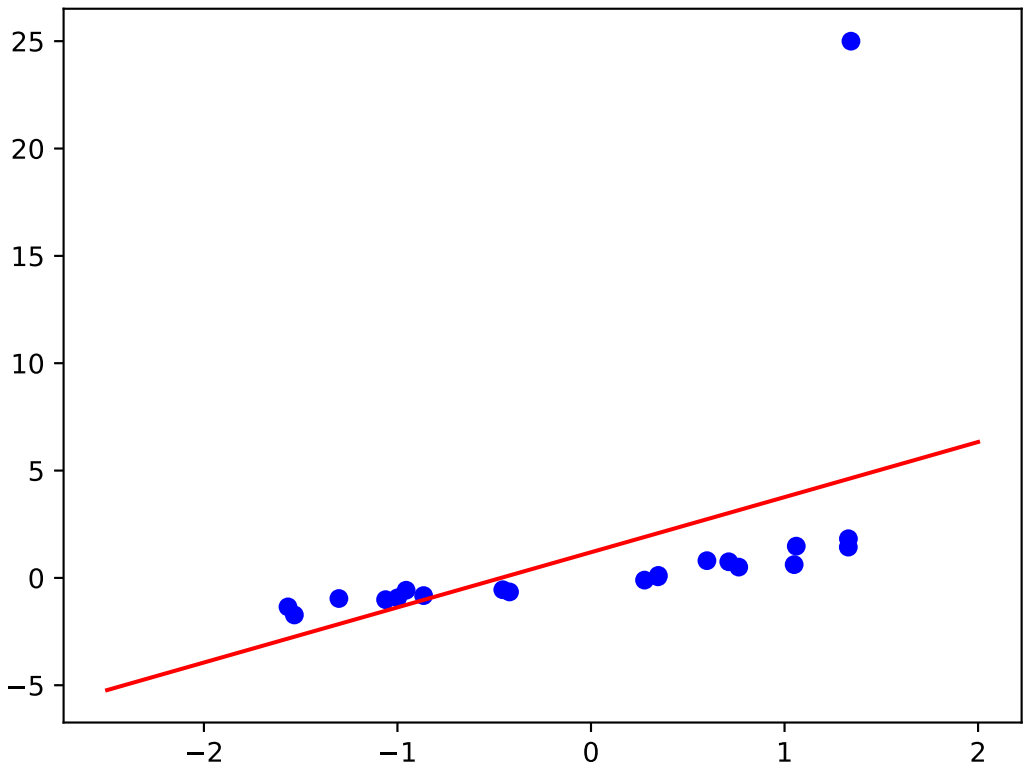
This is overfitting like crazy.
(We increased the capacity by adding polynomial features.)

What went wrong? (3/5)



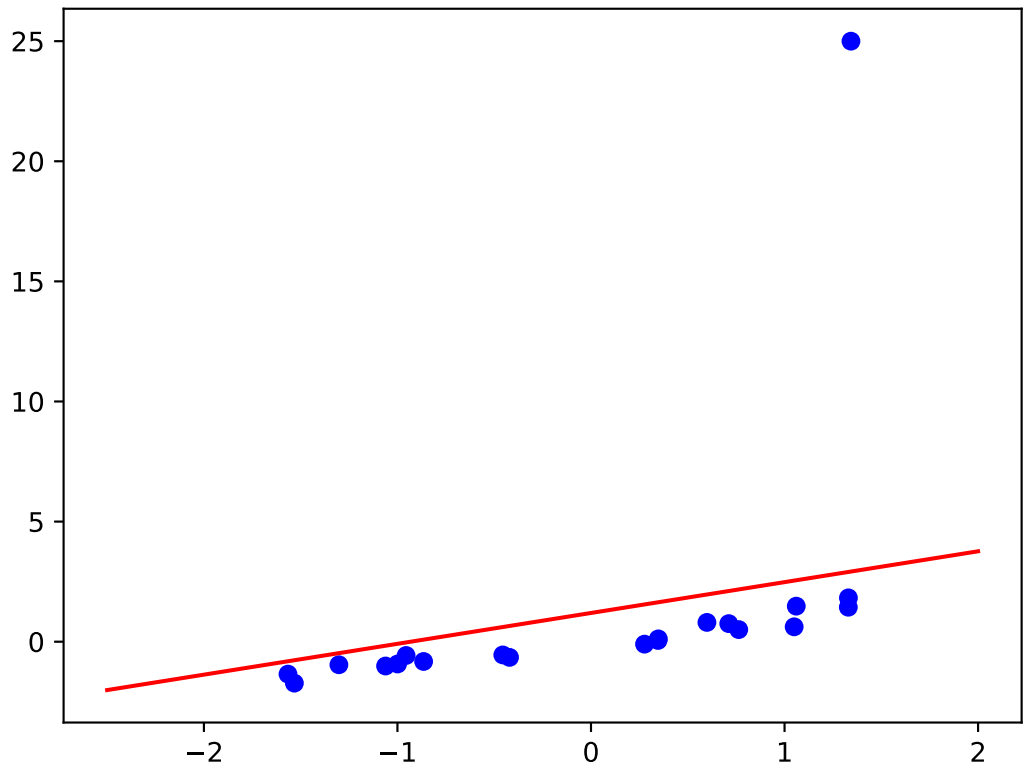
Nothing, this looks pretty good.
(We added L2 regularization to the previous model.)

What went wrong? (4/5)



The outlier steers the line away.

What went wrong? (5/5)



After adding L2 regularization, it is a little better.

A random variable $\mathbf{x}$ is a result of a random process, and it can be either discrete or continuous.

Probability Distribution

A probability distribution describes how likely are the individual values that 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 nonnegative, and the sum of the 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 $\int_a^b p(x)\,\mathrm{d}x$, where $p(x)$ is the *probability density function*, which is always nonnegative and integrates to 1 over the range of all values of $\mathbf{x}$.

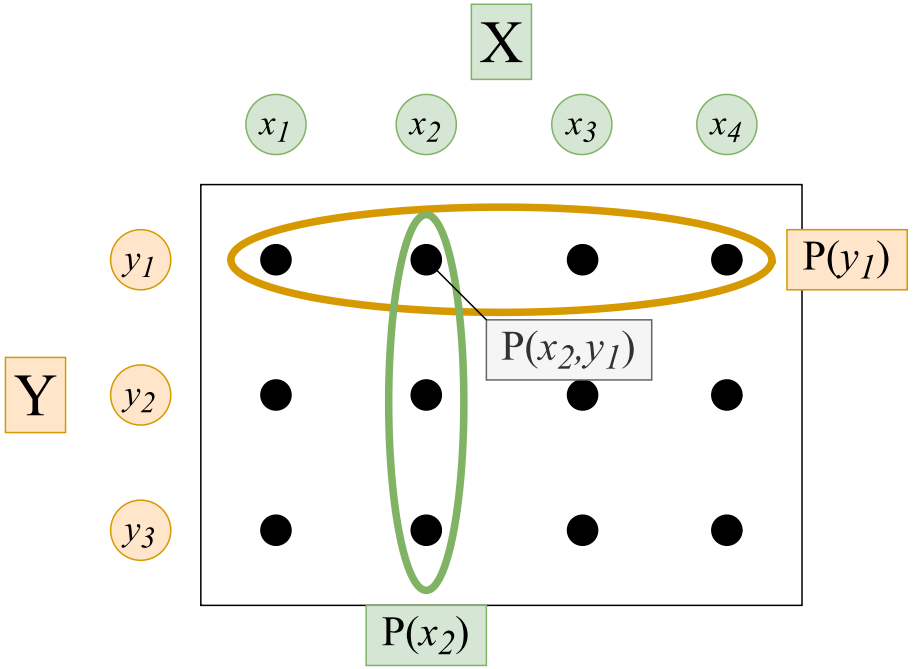
Joint, Conditional, Marginal Probability

For two random variables, a **joint probability distribution** is a distribution of all possible pairs of outputs (and analogously for more than two):

$$P(\mathrm{x} = x_2, \mathrm{y} = y_1).$$

Marginal distribution is a distribution of one (or a subset) of the random variables and can be obtained by summing over the other variable(s):

$$P(\mathrm{x} = x_2) = \sum_y P(\mathrm{x} = x_2, \mathrm{y} = y).$$



Conditional distribution is a distribution of one (or a subset) of the random variables, given that another event has already occurred:

$$P(\mathrm{x} = x_2 | \mathrm{y} = y_1) = P(\mathrm{x} = x_2, \mathrm{y} = y_1) / P(\mathrm{y} = y_1).$$

If $P(\mathrm{x}, \mathrm{y}) = P(\mathrm{x}) \cdot P(\mathrm{y})$ or $P(\mathrm{x}|\mathrm{y}) = P(\mathrm{x})$, random variables x and y are **independent**.

Expectation

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

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

For continuous variables, the expectation is computed as:

$$\mathbb{E}_{\mathbf{x}\sim p}[f(x)] \stackrel{\text{def}}{=} \int_x p(x) f(x)\,\mathrm{d}x.$$

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

Expectation is linear, i.e., for constants $\alpha, \beta \in \mathbb{R}$:

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

Variance

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

$$\mathrm{Var}(x) \stackrel{\text{def}}{=} \mathbb{E}\left[\left(x - \mathbb{E}[x]\right)^2\right], \text{ or more generally,}$$
$$\mathrm{Var}_{\mathbf{x}\sim P}(f(x)) \stackrel{\text{def}}{=} \mathbb{E}\left[\left(f(x) - \mathbb{E}[f(x)]\right)^2\right].$$

It is easy to see that

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

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

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

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

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

we may use *gradient descent*:

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

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

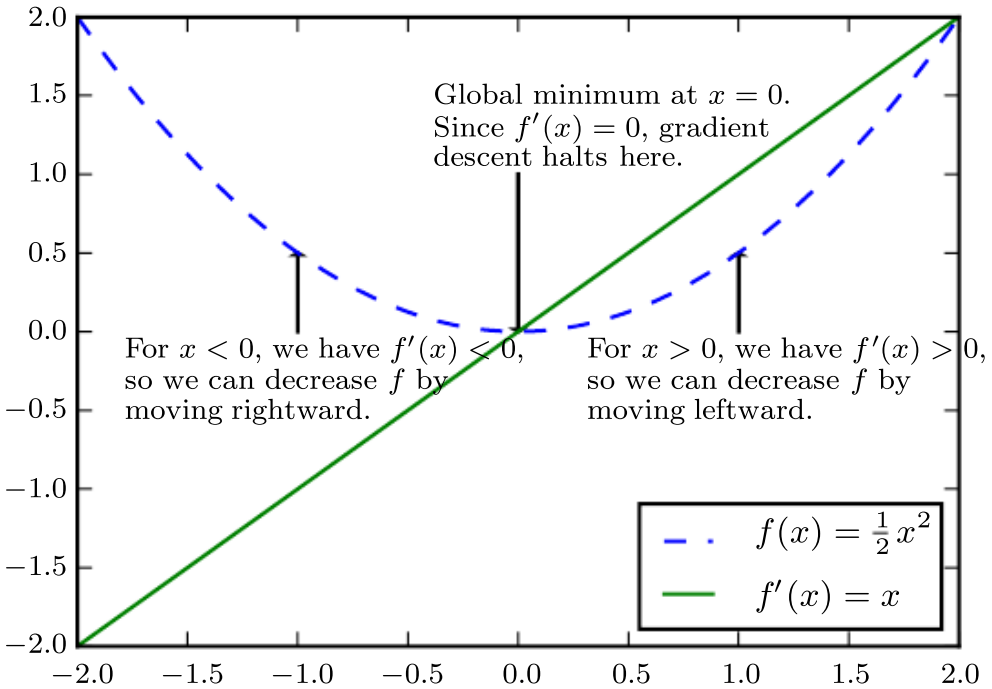


Figure 4.1 of "Deep Learning" book, https://www.deeplearningbook.org

Gradient Descent Variants

Let $\boldsymbol{X} \in \mathbb{R}^{N \times D}$, $\boldsymbol{t} \in \mathbb{R}^N$ be the training data, and denote $\hat{p}_{\text{data}}(\boldsymbol{x}, t) \stackrel{\text{def}}{=} \frac{|\{i:(\boldsymbol{x},t)=(\boldsymbol{x}_i,t_i)\}|}{N}$.
Assume that the error function can be computed as an expectation over the dataset:

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

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

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

- **Minibatch SGD**: Trade-off between gradient descent and SGD – the expectation in $\nabla_{\boldsymbol{w}} E(\boldsymbol{w})$ is estimated using B random independent examples from the training data.

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

Gradient Descent Convergence

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

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

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:

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

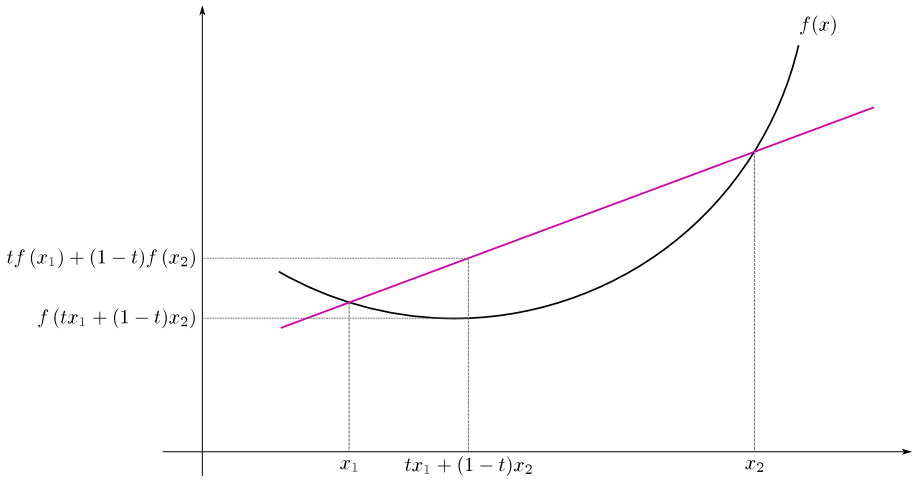
Note that the third condition implies that $\alpha_i \to 0$.

For nonconvex loss functions, we can get guarantees of converging to a *local* optimum only.

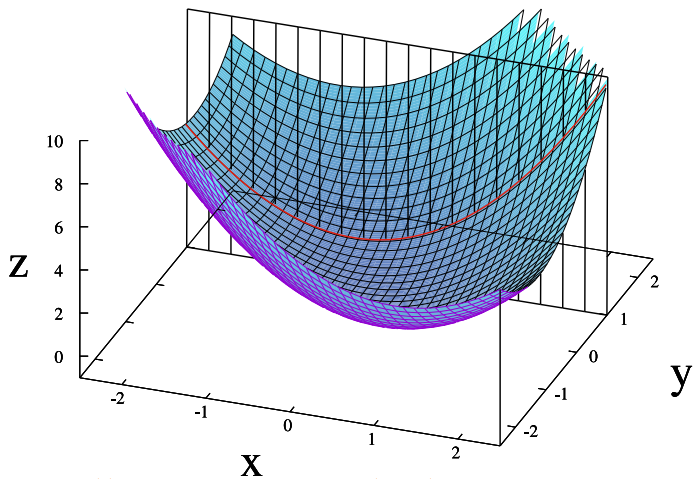
Gradient Descent Convergence

Convex functions mentioned on the previous slide are such that for $\boldsymbol{u}, \boldsymbol{v}$ and real $0 \leq t \leq 1$,

$$f(t\boldsymbol{u} + (1-t)\boldsymbol{v}) \leq tf(\boldsymbol{u}) + (1-t)f(\boldsymbol{v}).$$



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



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

A twice-differentiable function of a single variable is convex iff its second derivative is always nonnegative. (For functions of multiple variables, the Hessian must be positive semi-definite.)

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

Well-known examples of convex functions are x^2, e^x, $-\log x$, and also the *sum of squares*.

Solving Linear Regression using SGD

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

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

If we also include L^2 regularization, we get

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

We then estimate the expectation by a minibatch of examples with indices $\mathbb{B}$ as

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

which gives us an estimate of a gradient

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

Solving Linear Regression using SGD

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

Input: Dataset ($\boldsymbol{X} \in \mathbb{R}^{N \times D}$, $\boldsymbol{t} \in \mathbb{R}^N$), learning rate $\alpha \in \mathbb{R}^+$, L^2 strength $\lambda \in \mathbb{R}$.
Output: Weights $\boldsymbol{w} \in \mathbb{R}^D$ hopefully minimizing the regularized MSE of a linear regression model.

- $\boldsymbol{w} \leftarrow \mathbf{0}$ or we initialize $\boldsymbol{w}$ randomly
- repeat until convergence (or until our patience runs out):
  - sample a minibatch of examples with indices $\mathbb{B}$
    - 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 into minibatch-sized chunks
      - the most common option; one pass through the data is called an **epoch**
  - $\boldsymbol{w} \leftarrow \boldsymbol{w} - \alpha \frac{1}{|\mathbb{B}|} \sum_{i \in \mathbb{B}} \left((\boldsymbol{x}_i^T \boldsymbol{w} - t_i) \boldsymbol{x}_i \right) - \alpha \lambda \boldsymbol{w}$

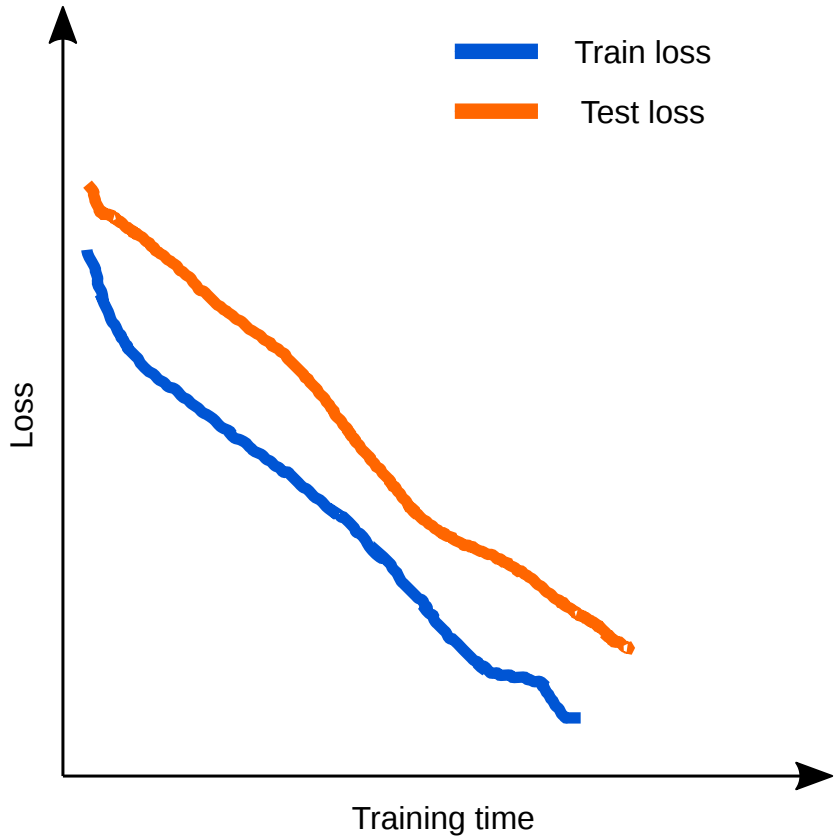
Training course

https://www.kaggle.com/code/ryanholbrook/stochastic-gradient-descent/tutorial

Illustrated Linear Regression

https://mlu-explain.github.io/linear-regression

What went wrong? (1/2)



The training did not converge yet.
We need to continue training.

What went wrong? (2/2)

The model is overfitting (zero training loss, increasing validation loss).
We should either regularize or do take the best validation checkpoint (i.e., **early stopping**).

Features

Recall that the *input* instance values are usually the raw observations and are given. However, we might extend them suitably before running a ML algorithm, especially if the algorithm cannot represent an arbitrary function (e.g., is linear). Such instance representations are called *features*.

Example from the previous lecture: even if our training examples were x and t, we performed the linear regression using features $(x^0, x^1, \ldots, x^M)$:

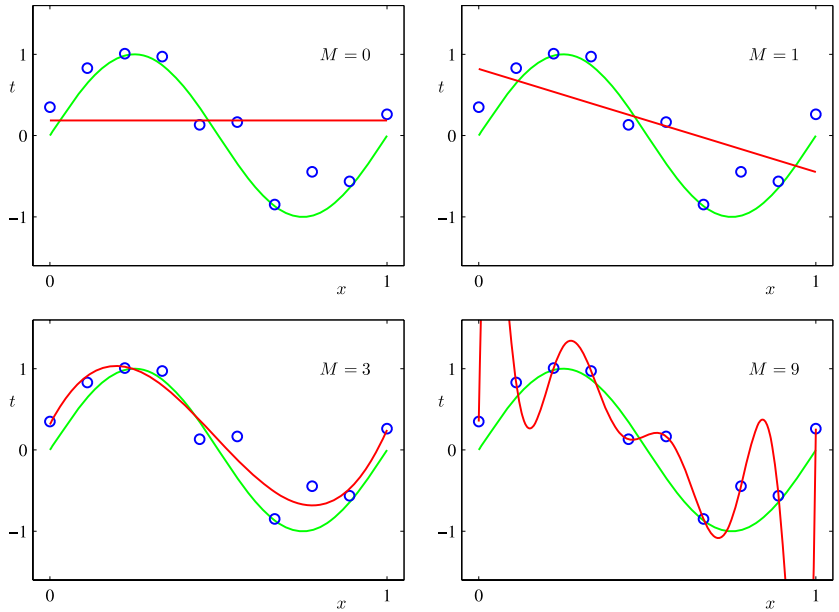


Figure 1.4 of Pattern Recognition and Machine Learning.

Feature Types

Generally, it would be best if the ML algorithms would process only the raw inputs. However, many algorithms can represent only a limited set of functions (e.g., linear), 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 $(x_1, x_2, \ldots, x_D)$, 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 that a day in a week is represented in 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 an 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 a **one-hot** representation, where an input integral value $0 \leq v < L$ is represented as L binary values, which are all zero except for the v^{th} one, which is one.

Feature Normalization

- Features in different scales would need different learning rates
- Common solution: normalize the features
  - Normalization: $x_{i,j}^{\text{norm}} = \frac{x_{i,j} - \min_k x_{k,j}}{\max_k x_{k,j} - \min_k x_{k,j}}$ (`MinMaxScaler` in Scikit-learn)
  - Standardization: $x_{i,j}^{\text{standard}} = \frac{x_{i,j} - \hat{\mu}_j}{\hat{\sigma} j}$ (`StandardScaler` in Scikit-learn)

Today's Lecture Objectives

After this lecture you should be able to

- Reason about **overfitting** in terms of **model capacity**.
- Use L^2-**regularization** to control model capacity.
- Explain what the difference between **parameters and hyperparameters** is.
- Tell what the **basic probability concepts** are (joint, marginal, conditional probability; expected value, mean, variance).
- Mathematically describe and implement the **stochastic gradient descent** algorithm.
- Use both **numerical and categorical features** in linear regression.