Lecture 4:

- Kernel Regression (cont’d.)
- Curse of Dimensionality
- Linear Regression
- Optimization
- Generalization
- Model complexity
- Regularization
Kernel Regression (cont’d.)
Administrative

- **Assignment 1** is out today!
- It is due **March 4** (i.e. in two weeks).
- It includes
  - Pencil-and-paper derivations
  - Implementing kNN classifier
  - Implementing linear regression
  - numpy/Python code
Assignment 1 Practical Problem

• Movie Rating and Genre Prediction

• The data has:
  - **Train**: 700 movies
  - **Test**: 300 movies

• You may want to split the training set into train and validation

• **Do not** use test data for training or parameter tuning

• Each movie has lots of available meta-data, e.g.:
  - Cast
  - Director(s)
  - Writer(s)
  - Year of release
  - Storyline (short description of the movie)
  - Plot (longer description of the movie)
  - Box-office information (try not using this for rating prediction)
  - Keywords (try not using this for genre prediction)

• Report performance on test data

Guess the rating of this movie!!!

adapted from Sanja Fidler
Four things make a memory based learner:

- **A distance metric**
  - Euclidean (and others)

- **How many nearby neighbors to look at?**
  - All of them

- **A weighting function (optional)**
  - \( w_i = \exp\left(-\frac{d(x_i, \text{query})^2}{\sigma^2}\right) \)
  - Nearby points to the query are weighted strongly, far points weakly. The \( \sigma \) parameter is the Kernel Width. Very important.

- **How to fit with the local points?**
  - Predict the weighted average of the outputs
    \[ \text{predict} = \frac{\Sigma w_i y_i}{\Sigma w_i} \]
Weighting/Kernel functions

\[ w_i = \exp(-d(x_i, \text{query})^2 / \sigma^2) \]

(Our examples use Gaussian)
Effect of Kernel Width

- What happens as $\sigma \to \infty$?
- What happens as $\sigma \to 0$?
Problems with Instance-Based Learning

• Expensive
  - No Learning: most real work done during testing
  - For every test sample, must search through all dataset – very slow!
  - Must use tricks like approximate nearest neighbour search

• Doesn’t work well when large number of irrelevant features
  - Distances overwhelmed by noisy features

• Curse of Dimensionality
  - Distances become meaningless in high dimensions
Curse of Dimensionality

• Consider applying a KNN classifier/regressor to data where the inputs are uniformly distributed in the $D$-dimensional unit cube.

• Suppose we estimate the density of class labels around a test point $x$ by “growing” a hyper-cube around $x$ until it contains a desired fraction $f$ of the data points.

• The expected edge length of this cube will be $e_D(f) = f^{1/D}$.

• If $D = 10$, and we want to base our estimate on 10% of the data, we have $e_{10}(0.1) = 0.8$, so we need to extend the cube 80% along each dimension around $x$.

• Even if we only use 1% of the data, we find $e_{10}(0.01) = 0.63$. — no longer very local
Linear Regression
Simple 1-D Regression

- Circles are data points (i.e., training examples) that are given to us.
- The data points are uniform in $x$, but may be displaced in $y$
  \[ t(x) = f(x) + \varepsilon \]
- In green is the “true” curve that we don’t know.
- Goal: We want to fit a curve to these points.

with $\varepsilon$ some noise.
Simple 1-D Regression

- **Key Questions:**
  - How do we parametrize the **model** (the curve)?
  - What **loss (objective) function** should we use to judge fit?
  - How do we optimize fit to unseen test data (**generalization**)?
Example: Boston House Prizes

- Estimate median house price in a neighborhood based on neighborhood statistics
- Look at first (of 13) attributes: per capita crime rate
- Use this to predict house prices in other neighborhoods
- Is this a good input (attribute) to predict house prices?

https://archive.ics.uci.edu/ml/datasets/Housing
Represent the data

- Data described as pairs \( D = \{(x^{(1)}, t^{(1)}), (x^{(2)}, t^{(2)}), \ldots, (x^{(N)}, t^{(N)})\} \)
  - \( x \) is the **input feature** (per capita crime rate)
  - \( t \) is the **target output** (median house price)
  - \( ^{(i)} \) simply indicates the training examples (we have \( N \) in this case)

- Here \( t \) is continuous, so this is a **regression problem**

- Model outputs \( y \), an estimate of \( t \)
  \[
y(x) = w_0 + w_1 x
\]

- What type of **model** did we choose?

- Divide the dataset into training and testing examples
  - Use the training examples to construct hypothesis, or function approximator, that maps \( x \) to predicted \( y \)
  - Evaluate hypothesis on test set
A simple model typically does not exactly fit the data — lack of fit can be considered noise.

Sources of noise:
- Imprecision in data attributes (input noise, e.g. noise in per-capita crime)
- Errors in data targets (mislabeled, e.g. noise in house prices)
- Additional attributes not taken into account by data attributes, affect target values (latent variables). In the example, what else could affect house prices?
- Model may be too simple to account for data targets
Least-Squares Regression

Define a model

Linear:

\[ y(x) = w_0 + w_1 x \]

Standard loss/cost/objective function measures the squared error between \( y \) and the true value \( t \).

Linear model:

\[
(l(w)) = \sum_{n=1}^{N} (t(n) - (w_0 + w_1 x(n)))^2
\]

For a particular hypothesis (\( y(x) \) defined by a choice of \( \mathbf{w} \)), what does the loss represent geometrically?

How do we obtain weights \( \mathbf{w} = (w_0, w_1) \)?

For the linear model, what kind of a function is \( l(w) \)?
Least-Squares Regression

- Define a model

  Linear: \( y(x) = \text{function}(x, w) \)

- Standard loss/cost/objective function measures the squared error between \( y \) and the true value \( t \).

- For a particular hypothesis \( (y(x) \text{ defined by a choice of } w, \text{ drawn in red}) \), what does the loss represent geometrically?

- How do we obtain weights \( w = (w_0, w_1) \)?

- For the linear model, what kind of a function is \( (w) \)?
Least-Squares Regression

- Define a model

Linear: \( y(x) = w_0 + w_1 x \)
Least-Squares Regression

• Define a model

  Linear: \( y(x) = w_0 + w_1 x \)

• Standard loss/cost/objective function measures the squared error between \( y \) and the true value \( t \)

\[
\ell(w) = \sum_{n=1}^{N} \left[ t^{(n)} - y(x^{(n)}) \right]^2
\]
Least-Squares Regression

- Define a model
  Linear: \( y(x) = w_0 + w_1 x \)

- Standard loss/cost/objective function measures the squared error between \( y \) and the true value \( t \)

Linear model: \[ \ell(w) = \sum_{n=1}^{N} \left[ t^{(n)} - (w_0 + w_1 x^{(n)}) \right]^2 \]
Least-Squares Regression

• Define a model

Linear: \[ y(x) = w_0 + w_1 x \]

• Standard loss/cost/objective function measures the squared error between \( y \) and the true value \( t \)

Linear model: \[ \ell(w) = \sum_{n=1}^{N} [t^{(n)} - (w_0 + w_1 x^{(n)})]^2 \]

• For a particular hypothesis \((y(x)\) defined by a choice of \(w\), drawn in red), what does the loss represent geometrically?
Least-Squares Regression

- Define a model
  
  **Linear:** \( y(x) = w_0 + w_1 x \)
  
- Standard loss/cost/objective function measures the squared error between \( y \) and the true value \( t \)

  Linear model:
  
  \[
  \ell(w) = \sum_{n=1}^{N} \left[ t^{(n)} - (w_0 + w_1 x^{(n)}) \right]^2
  \]

- The loss for the red hypothesis is the **sum of the squared vertical errors** (squared lengths of green vertical lines)
Least-Squares Regression

- Define a model
  Linear: \( y(x) = w_0 + w_1 x \)

- Standard loss/cost/objective function measures the squared error between \( y \) and the true value \( t \)

  Linear model:
  \[
  \ell(w) = \sum_{n=1}^{N} \left[ t^{(n)} - (w_0 + w_1 x^{(n)}) \right]^2
  \]

- How do we obtain weights \( w = (w_0, w_1) \)?
Least-Squares Regression

- Define a model
  Linear: \( y(x) = w_0 + w_1 x \)

- Standard loss/cost/objective function measures the squared error between \( y \) and the true value \( t \)

  Linear model:
  \[
  \ell(w) = \sum_{n=1}^{N} \left[ t^{(n)} - (w_0 + w_1 x^{(n)}) \right]^2
  \]

- How do we obtain weights \( w = (w_0, w_1) \)? Find \( w \) that minimizes loss \( \ell(w) \)
Optimizing the Objective

• One straightforward method: gradient descent
  - initialize \( w \) (e.g., randomly)
  - repeatedly update \( w \) based on the gradient

\[
   w \leftarrow w - \lambda \frac{\partial \ell}{\partial w}
\]

• \( \lambda \) is the learning rate

• For a single training case, this gives the LMS update rule:

\[
   w \leftarrow w + 2\lambda \left( t^{(n)} - y(x^{(n)}) \right) x^{(n)}
\]

\[\text{error}\]

• Note: As error approaches zero, so does the update (\( w \) stops changing)
Optimizing the Objective
Optimizing the Objective
Effect of learning rate $\lambda$

- Large $\lambda$ => Fast convergence but larger residual error
  Also possible oscillations

- Small $\lambda$ => Slow convergence but small residual error
Optimizing Across Training Set

- Two ways to generalize this for all examples in training set:
  1. **Batch updates**: sum or average updates across every example \( n \), then change the parameter values
     \[
     \mathbf{w} \leftarrow \mathbf{w} + 2\lambda \left( t^{(n)} - y(x^{(n)}) \right) x^{(n)}
     \]
  2. **Stochastic/online updates**: update the parameters for each training case in turn, according to its own gradients

---

**Algorithm 1** Stochastic gradient descent

1: Randomly shuffle examples in the training set
2: for \( i = 1 \) to \( N \) do
3:     Update:
4:         \[
        \mathbf{w} \leftarrow \mathbf{w} + 2\lambda (t^{(i)} - y(x^{(i)}))x^{(i)}
        \]
5: end for

Underlying assumption: sample is independent and identically distributed (i.i.d.)
Optimizing Across Training Set

• Two ways to generalize this for all examples in training set:

1. **Batch updates:** sum or average updates across every example \( n \), then change the parameter values

\[
\mathbf{w} \leftarrow \mathbf{w} + 2\lambda \left( t^{(n)} - y(x^{(n)}) \right) x^{(n)}
\]

2. **Stochastic/online updates:** update the parameters for each training case in turn, according to its own gradients

• Underlying assumption: sample is independent and identically distributed (i.i.d.)
Analytical Solution

• For some objectives we can also find the optimal solution analytically

• This is the case for linear least-squares regression

• How?
Vectorization

• Consider our model:

\[ y(x) = w_0 + w_1 x \]

• Let

\[ \mathbf{w} = \begin{bmatrix} w_0 \\ w_1 \end{bmatrix} \quad \mathbf{x}^T = [1 \ x] \]

• Can write the model in vectorized form as

\[ y(x) = \mathbf{w}^T \mathbf{x} \]
Vectorization

• Consider our model with \( n \) instances:

\[
\begin{align*}
\mathbf{t} &= \left[ t^{(1)}, t^{(2)}, \ldots, t^{(N)} \right]^T \\
\mathbf{X} &= \begin{bmatrix} 1, x^{(1)} \\ 1, x^{(2)} \\ \vdots \\ 1, x^{(N)} \end{bmatrix}^T \\
\mathbf{w} &= \begin{bmatrix} w_0 \\ w_1 \end{bmatrix}
\end{align*}
\]

• Then:

\[
\ell(\mathbf{w}) = \sum_{n=1}^{N} \left[ \mathbf{w}^T \mathbf{x}^{(n)} - t^{(n)} \right]^2
\]

\[
= (\mathbf{X} \mathbf{w} - \mathbf{t})^T (\mathbf{X} \mathbf{w} - \mathbf{t})
\]
Analytical Solution

• Instead of using GD, solve for optimal \( w \) analytically
  - Notice the solution is when \( \frac{\partial}{\partial w} \ell(w) = 0 \)

• Derivation:

\[
\ell(w) = (Xw - t)^T(Xw - t)
= w^TX^TXw - t^TXw - w^TX^Tt + t^Tt
= w^TX^TXw - 2w^TX^Tt + t^Tt
\]

- Take derivative and set equal to 0, then solve for

\[
\frac{\partial}{\partial w} \left( w^TX^TXw - 2w^TX^Tt + t^Tt \right) = 0
\]

\[
(X^TX)w - X^Tt = 0
\]

\[
(X^TX)w = X^Tt
\]

**Closed Form Solution:** \( w = (X^TX)^{-1}X^Tt \)

If \( X^TX \) is not invertible (i.e., singular), may need to:
- Use pseudo-inverse instead of the inverse
  - In Python, `numpy.linalg.pinv(a)`
- Remove redundant (not linearly independent) features
- Remove extra features to ensure that \( d \leq N \)
Multi-dimensional Inputs

- One method of extending the model is to consider other input dimensions
  \[ y(x) = w_0 + w_1 x_1 + w_2 x_2 \]

- In the Boston housing example, we can look at the number of rooms

---

(Raw text content is repeated here for reference)

- One method of extending the model is to consider other input dimensions
  \[ y(x) = w_0 + w_1 x_1 + w_2 x_2 \]

- In the Boston housing example, we can look at the number of rooms
Linear Regression with Multi-dimensional Inputs

- Imagine now we want to predict the median house price from these multi-dimensional observations.

- Each house is a data point $n$, with observations indexed by $j$:

  $$\mathbf{x}^{(n)} = \left(x_1^{(n)}, \ldots, x_j^{(n)}, \ldots, x_d^{(n)}\right)$$

- We can incorporate the bias $w_0$ into $\mathbf{w}$, by using $x_0 = 1$, then

  $$y(\mathbf{x}) = w_0 + \sum_{j=1}^{d} w_j x_j = \mathbf{w}^T \mathbf{x}$$

- We can then solve for $\mathbf{w} = (w_0, w_1, \ldots, w_d)$. How?

- We can use gradient descent to solve for each coefficient, or compute $\mathbf{w}$ analytically (how does the solution change?)
More Powerful Models?

• What if our linear model is not good? How can we create a more complicated model?
Fitting a Polynomial

• What if our linear model is not good? How can we create a more complicated model?

• We can create a more complicated model by defining input variables that are combinations of components of $x$

• Example: an $M$-th order polynomial function of one dimensional feature $x$:

$$y(x, w) = w_0 + \sum_{j=1}^{M} w_j x^j$$

where $x^j$ is the $j$-th power of $x$

• We can use the same approach to optimize for the weights $w$

• How do we do that?
Some types of basis functions in 1-D

- **Sigmoidoids**
  \[ \phi_j(x) = \sigma \left( \frac{x - \mu_j}{s} \right) \]
  \[ \sigma(a) = \frac{1}{1 + \exp(-a)} \]

- **Gaussians**
  \[ \phi_j(x) = \exp \left\{ -\frac{(x - \mu_j)^2}{2s^2} \right\} \]

- **Polynomials**

---

*Equivalently, we can use the \( \phi \) or feature extraction, to the original data variables. If the original variables comprise the vector \( \phi \) and the parameter \( \theta \), then the features can be expressed in terms of the basis functions.*

---

*By using nonlinear basis functions, we allow the function to be non-linear. An example of polynomial regression considered in Chapter 1 is a polynomial function. These various choices of basis function are illustrated in Figure 3.1.*
Two types of linear model that are equivalent with respect to learning

\[ y(x, w) = w_0 + w_1 x_1 + w_2 x_2 + \ldots = w^T x \]

\[ y(x, w) = w_0 + w_1 \phi_1(x) + w_2 \phi_2(x) + \ldots = w^T \Phi(x) \]

• The first model has the same number of adaptive coefficients as the dimensionality of the data +1.

• The second model has the same number of adaptive coefficients as the number of basis functions +1.

• Once we have replaced the data by the outputs of the basis functions, fitting the second model is exactly the same problem as fitting the first model (unless we use the kernel trick).
General linear regression problem

- Using our new notations for the basis function linear regression can be written as

\[ y = \sum_{j=0}^{n} w_j \phi_j(x) \]

where \( \phi_j(x) \) can be either \( x_j \) for multivariate regression or one of the nonlinear basis we defined.

- Once again we can use “least squares” to find the optimal solution.
LMS for the general linear regression problem

Our goal is to minimize the following loss function:

\[ J(w) = \sum_{i} (y^i - \sum_{j} w_j \phi_j(x^i))^2 \]

Moving to vector notations we get:

\[ J(w) = \sum_{i} (y^i - w^T \phi(x^i))^2 \]

We take the derivative w.r.t \( w \)

\[ \frac{\partial}{\partial w} \sum_{i} (y^i - w^T \phi(x^i))^2 = 2 \sum_{i} (y^i - w^T \phi(x^i))\phi(x^i)^T \]

Equating to 0 we get

\[ 2 \sum_{i} (y^i - w^T \phi(x^i))\phi(x^i)^T = 0 \Rightarrow \]

\[ \sum_{i} y^i \phi(x^i)^T = w^T \left[ \sum_{i} \phi(x^i)\phi(x^i)^T \right] \]

\( y = \sum_{j=0}^{n} w_j \phi_j(x) \)

\( w \) – vector of dimension \( k+1 \)

\( \phi(x^i) \) – vector of dimension \( k+1 \)

\( y^i \) – a scaler
LMS for the general linear regression problem

We take the derivative w.r.t $w$

$$\frac{\partial}{\partial w} \sum_i (y^i - w^T \phi(x^i))^2 = 2 \sum_i (y^i - w^T \phi(x^i))\phi(x^i)^T$$

Equating to 0 we get

$$2 \sum_i (y^i - w^T \phi(x^i))\phi(x^i)^T = 0 \Rightarrow$$

$$\sum_i y^i \phi(x^i)^T = w^T \left[ \sum_i \phi(x^i)\phi(x^i)^T \right]$$

Define:

$$\Phi = \begin{pmatrix} 
\phi_0(x^1) & \phi_1(x^1) & \ldots & \phi_m(x^1) \\
\phi_0(x^2) & \phi_1(x^2) & \ldots & \phi_m(x^2) \\
\phi_0(x^n) & \phi_1(x^n) & \ldots & \phi_m(x^n) 
\end{pmatrix}$$

Then deriving $w$ we get:

$$w = (\Phi^T \Phi)^{-1} \Phi^T y$$
LMS for the general linear regression problem

Deriving \( w \) we get:

\[
\mathbf{w} = (\Phi^T \Phi)^{-1} \Phi^T \mathbf{y}
\]

This solution is also known as ‘psuedo inverse’

\[
J(w) = \sum_i (y_i - w^T \phi(x_i))^2
\]
0th order polynomial

\[ M = 0 \]
1\textsuperscript{st} order polynomial
3\textsuperscript{rd} order polynomial

\[ M = 3 \]
9th order polynomial

\[ M = 9 \]
Which Fit is Best?

from Bishop

\[ M = 0 \]

\[ M = 1 \]

\[ M = 3 \]

\[ M = 9 \]
Root Mean Square (RMS) Error

\[ E(w) = \frac{1}{2} \sum_{n=1}^{N} \left\{ y(x_n, w) - t_n \right\}^2 \]

\[ E_{\text{RMS}} = \sqrt{2E(w^*)/N} \]

The division by \( N \) allows us to compare different sizes of data sets on an equal footing, and the square root ensures that \( E_{\text{RMS}} \) is measured on the same scale (and in the same units) as the target variable \( t \).
Overfitting

Root Mean Square (RMS) Error:

\[ E_{\text{RMS}} = \sqrt{\frac{2E(w^*)}{N}} \]

\[ E(w) = \frac{1}{2} \sum_{n=1}^{N} (t_n - \phi(x_n)^T w)^2 = \frac{1}{2} \| t - \Phi w \|^2 \]

**Figure 1.5**
Graphs of the root-mean-square error, defined by (1.3), evaluated on the training set and on an independent test set for various values of \( M \).

For \( M = 9 \), the training set error goes to zero, as we might expect because this polynomial contains 10 degrees of freedom corresponding to the 10 coefficients \( w_0, ..., w_9 \), and so can be tuned exactly to the 10 data points in the training set.

However, the test set error has become very large and, as we saw in Figure 1.4, the corresponding function \( y(x, w^*) \) exhibits wild oscillations.

This may seem paradoxical because a polynomial of given order contains all lower order polynomials as special cases. The \( M = 9 \) polynomial is therefore capable of generating results at least as good as the \( M = 3 \) polynomial. Furthermore, we might suppose that the best predictor of new data would be the function \( \sin(2\pi x) \) from which the data was generated (and we shall see later that this is indeed the case). We know that a power series expansion of the function \( \sin(2\pi x) \) contains terms of all orders, so we might expect that results should improve monotonically as we increase \( M \).

We can gain some insight into the problem by examining the values of the coefficients \( w^* \) obtained from polynomials of various order, as shown in Table 1.1.

**Table 1.1**
Table of the coefficients \( w^* \) for polynomials of various order.

<table>
<thead>
<tr>
<th>( M )</th>
<th>( w^*_0 )</th>
<th>( w^*_1 )</th>
<th>( w^*_2 )</th>
<th>( w^*_3 )</th>
<th>( w^*_4 )</th>
<th>( w^*_5 )</th>
<th>( w^*_6 )</th>
<th>( w^*_7 )</th>
<th>( w^*_8 )</th>
<th>( w^*_9 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.19</td>
<td>0.82</td>
<td>0.31</td>
<td>0.35</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>-1.27</td>
<td>7.99</td>
<td>232.37</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>-25.43</td>
<td>-5321.83</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>17.37</td>
<td>48568.31</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>-231639.30</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>640042.26</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>-1061800.52</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>1042400.18</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>-557682.99</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>125201.43</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Generalization

- **Generalization** = model’s ability to predict the held out data
- What is happening?
- Our model with $M = 9$ **overfits** the data (it models also noise)

![Graph showing root-mean-square error (RMS) for training and test sets with varying $M$ values.](image)

For $M = 9$, the training set error goes to zero, as we might expect because this polynomial contains 10 degrees of freedom corresponding to the 10 coefficients $w_0, \ldots, w_9$, and so can be tuned exactly to the 10 data points in the training set. However, the test set error has become very large and, as we saw in Figure 1.4, the corresponding function $y(x, w^\star)$ exhibits wild oscillations. This may seem paradoxical because a polynomial of given order contains all lower order polynomials as special cases. The $M = 9$ polynomial is therefore capable of generating results at least as good as the $M = 3$ polynomial. Furthermore, we might suppose that the best predictor of new data would be the function $\sin(2\pi x)$ from which the data was generated (and we shall see later that this is indeed the case). We know that a power series expansion of the function $\sin(2\pi x)$ contains terms of all orders, so we might expect that results should improve monotonically as we increase $M$.

We can gain some insight into the problem by examining the values of the coefficients $w^\star$ obtained from polynomials of various order, as shown in Table 1.1.

<table>
<thead>
<tr>
<th>$M$</th>
<th>$w^\star_0$</th>
<th>$w^\star_1$</th>
<th>$w^\star_2$</th>
<th>$w^\star_3$</th>
<th>$w^\star_4$</th>
<th>$w^\star_5$</th>
<th>$w^\star_6$</th>
<th>$w^\star_7$</th>
<th>$w^\star_8$</th>
<th>$w^\star_9$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.19</td>
<td>0.82</td>
<td>0.31</td>
<td>0.35</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>-1.27</td>
<td>7.99</td>
<td>232.37</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>-25.43</td>
<td>-5321.83</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>17.37</td>
<td>48568.31</td>
<td>-231639.30</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>640042.26</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>-1061800.52</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>1042400.18</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>-557682.99</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>125201.43</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

We see that, as $M$ increases, the magnitude of the coefficients typically gets larger. In particular for the $M = 9$ polynomial, the coefficients have become finely tuned to the data by developing large positive and negative values so that the corresponding function $y(x, w^\star)$ exhibits wild oscillations.
**Generalization**

- **Generalization** = model’s ability to predict the held out data
- What is happening?
  - Our model with $M = 9$ **overfits** the data (it models also noise)
  - Not a problem if we have lots of training examples
Generalization

- **Generalization** = model’s ability to predict the held out data
- What is happening?
- Our model with $M = 9$ **overfits** the data (it models also noise)
- Let’s look at the estimated weights for various $M$ in the case of fewer examples

<table>
<thead>
<tr>
<th></th>
<th>$M = 0$</th>
<th>$M = 1$</th>
<th>$M = 6$</th>
<th>$M = 9$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$w_0^*$</td>
<td>0.19</td>
<td>0.82</td>
<td>0.31</td>
<td>0.35</td>
</tr>
<tr>
<td>$w_1^*$</td>
<td></td>
<td>-1.27</td>
<td>7.99</td>
<td>232.37</td>
</tr>
<tr>
<td>$w_2^*$</td>
<td></td>
<td></td>
<td>-25.43</td>
<td>-5321.83</td>
</tr>
<tr>
<td>$w_3^*$</td>
<td></td>
<td></td>
<td>17.37</td>
<td>48568.31</td>
</tr>
<tr>
<td>$w_4^*$</td>
<td></td>
<td></td>
<td></td>
<td>-231639.30</td>
</tr>
<tr>
<td>$w_5^*$</td>
<td></td>
<td></td>
<td></td>
<td>640042.26</td>
</tr>
<tr>
<td>$w_6^*$</td>
<td></td>
<td></td>
<td></td>
<td>-1061800.52</td>
</tr>
<tr>
<td>$w_7^*$</td>
<td></td>
<td></td>
<td></td>
<td>1042400.18</td>
</tr>
<tr>
<td>$w_8^*$</td>
<td></td>
<td></td>
<td></td>
<td>-557682.99</td>
</tr>
<tr>
<td>$w_9^*$</td>
<td></td>
<td></td>
<td></td>
<td>125201.43</td>
</tr>
</tbody>
</table>
Generalization

- Generalization = model’s ability to predict the held out data
- What is happening?
  - Our model with $M = 9$ overfits the data (it models also noise)
  - Let’s look at the estimated weights for various $M$ in the case of fewer examples
  - The weights are becoming huge to compensate for the noise
  - One way of dealing with this is to encourage the weights to be small (this way no input dimension will have too much influence on prediction). This is called regularization.
1-D regression illustrates key concepts

• Data fits – is linear model best (model selection)?
  – Simplest models do not capture all the important variations (signal) in the data: underfit
  – More complex model may overfit the training data (fit not only the signal but also the noise in the data), especially if not enough data to constrain model

• One method of assessing fit: test generalization = model’s ability to predict the held out data

• Optimization is essential: stochastic and batch iterative approaches; analytic when available
Regularized Least Squares

- A technique to control the overfitting phenomenon
- Add a penalty term to the error function in order to discourage the coefficients from reaching large values

Ridge regression

\[
\tilde{E}(w) = \frac{1}{2} \sum_{n=1}^{N} \{y(x_n, w) - t_n\}^2 + \frac{\lambda}{2} \|w\|^2
\]

which is minimized by

\[
w = \left(\lambda I + \Phi^T \Phi\right)^{-1} \Phi^T t.
\]
The effect of regularization

Figure 1.7 Plots of \( M = 9 \) polynomials fitted to the data set shown in Figure 1.2 using the regularized error function (1.4) for two values of the regularization parameter \( \lambda \) corresponding to \( \ln \lambda = -18 \) and \( \ln \lambda = 0 \). The case of no regularizer, i.e., \( \lambda = 0 \), corresponding to \( \ln \lambda = -\infty \), is shown at the bottom right of Figure 1.4.

One technique that is often used to control the over-fitting phenomenon in such cases is that of regularization, which involves adding a penalty term to the error function (1.2) in order to discourage the coefficients from reaching large values. The simplest such penalty term takes the form of a sum of squares of all of the coefficients, leading to a modified error function of the form

\[
\tilde{E}(w) = \frac{1}{2} N \sum_{n=1}^{N} \left( y(x_n, w) - t_n \right)^2 + \lambda \|w\|^2 \tag{1.4}
\]

where \( \|w\|^2 \equiv w^T w = w_0^2 + w_1^2 + \ldots + w_M^2 \), and the coefficient \( \lambda \) governs the relative importance of the regularization term compared with the sum-of-squares error term. Note that often the coefficient \( w_0 \) is omitted from the regularizer because its inclusion causes the results to depend on the choice of origin for the target variable (Hastie et al., 2001), or it may be included but with its own regularization coefficient (we shall discuss this topic in more detail in Section 5.5.1). Again, the error function in (1.4) can be minimized exactly in closed form. Techniques such as this are known in the statistics literature as shrinkage methods because they reduce the value of the coefficients. The particular case of a quadratic regularizer is called ridge regression (Hoerl and Kennard, 1970). In the context of neural networks, this approach is known as weight decay.
The effect of regularization

The corresponding coefficients from the fitted polynomials, showing that regularization has the desired effect of reducing the magnitude of the coefficients.

<table>
<thead>
<tr>
<th>ln λ = −∞</th>
<th>ln λ = −18</th>
<th>ln λ = 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>(w_0^*)</td>
<td>0.35</td>
<td>0.35</td>
</tr>
<tr>
<td>(w_1^*)</td>
<td>232.37</td>
<td>4.74</td>
</tr>
<tr>
<td>(w_2^*)</td>
<td>-5321.83</td>
<td>-0.77</td>
</tr>
<tr>
<td>(w_3^*)</td>
<td>48568.31</td>
<td>-31.97</td>
</tr>
<tr>
<td>(w_4^*)</td>
<td>-231639.30</td>
<td>-3.89</td>
</tr>
<tr>
<td>(w_5^*)</td>
<td>640042.26</td>
<td>55.28</td>
</tr>
<tr>
<td>(w_6^*)</td>
<td>-1061800.52</td>
<td>41.32</td>
</tr>
<tr>
<td>(w_7^*)</td>
<td>1042400.18</td>
<td>-45.95</td>
</tr>
<tr>
<td>(w_8^*)</td>
<td>-557682.99</td>
<td>-91.53</td>
</tr>
<tr>
<td>(w_9^*)</td>
<td>125201.43</td>
<td>72.68</td>
</tr>
</tbody>
</table>
A more general regularizer

\[
\frac{1}{2} \sum_{n=1}^{N} \left\{ t_n - w^T \phi(x_n) \right\}^2 + \frac{\lambda}{2} \sum_{j=1}^{M} |w_j|^q
\]

![Contours of the regularization term in (3.29) for various values of the parameter](image)

- \( q = 0.5 \)
- \( q = 1 \)
- \( q = 2 \)
- \( q = 4 \)
1-D regression illustrates key concepts

- Data fits – is linear model best (model selection)?
  - Simple models may not capture all the important variations (signal) in the data: underfit
  - More complex models may overfit the training data (fit not only the signal but also the noise in the data), especially if not enough data to constrain model

- One method of assessing fit: test generalization = model’s ability to predict the held out data

- Optimization is essential: stochastic and batch iterative approaches; analytic when available