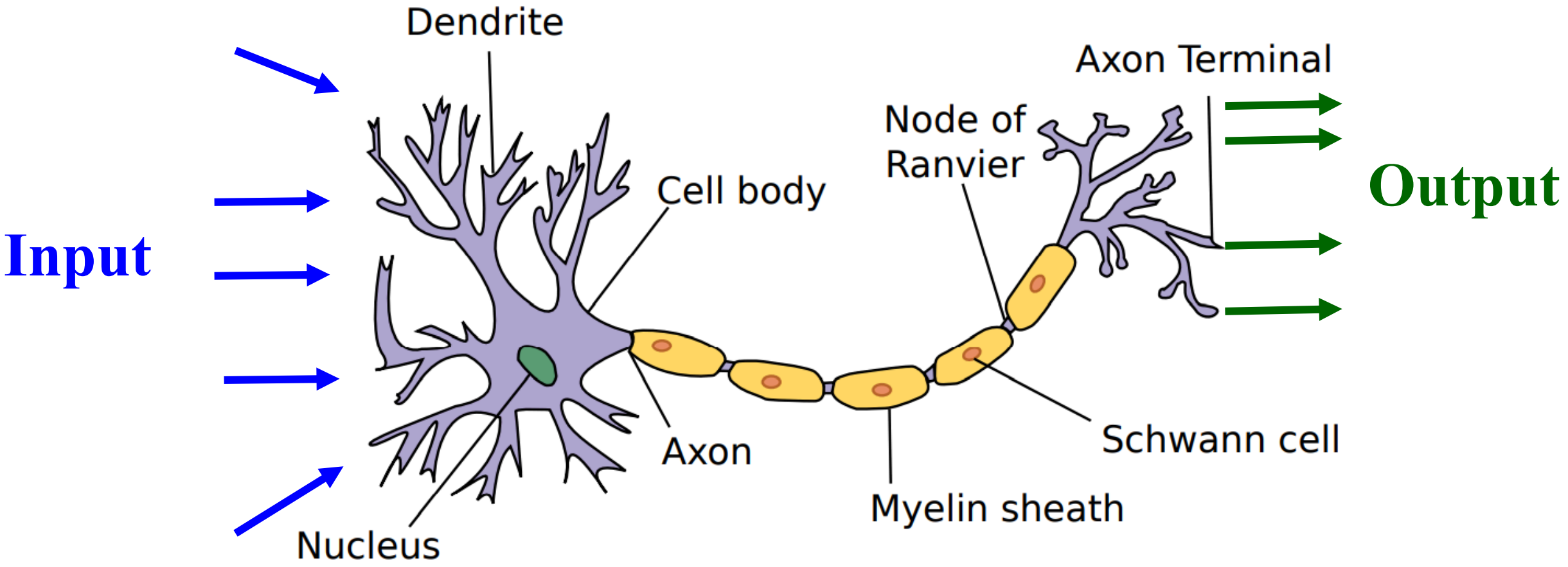
Regression Using Single-Neuron Perceptron

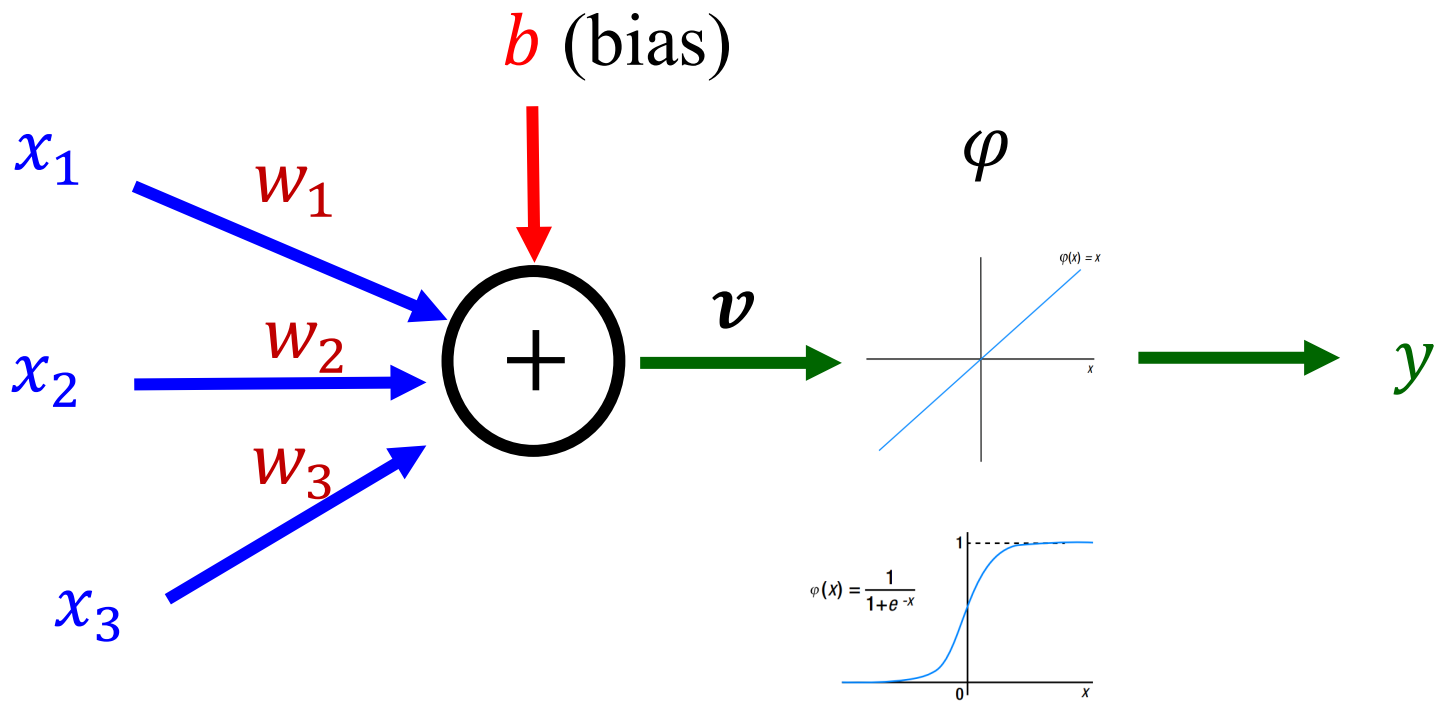
生醫光電所 吳育德

Paolo Perrotta, Programming Machine Learning: From Coding to Deep Learning (2020)

A biological neuron



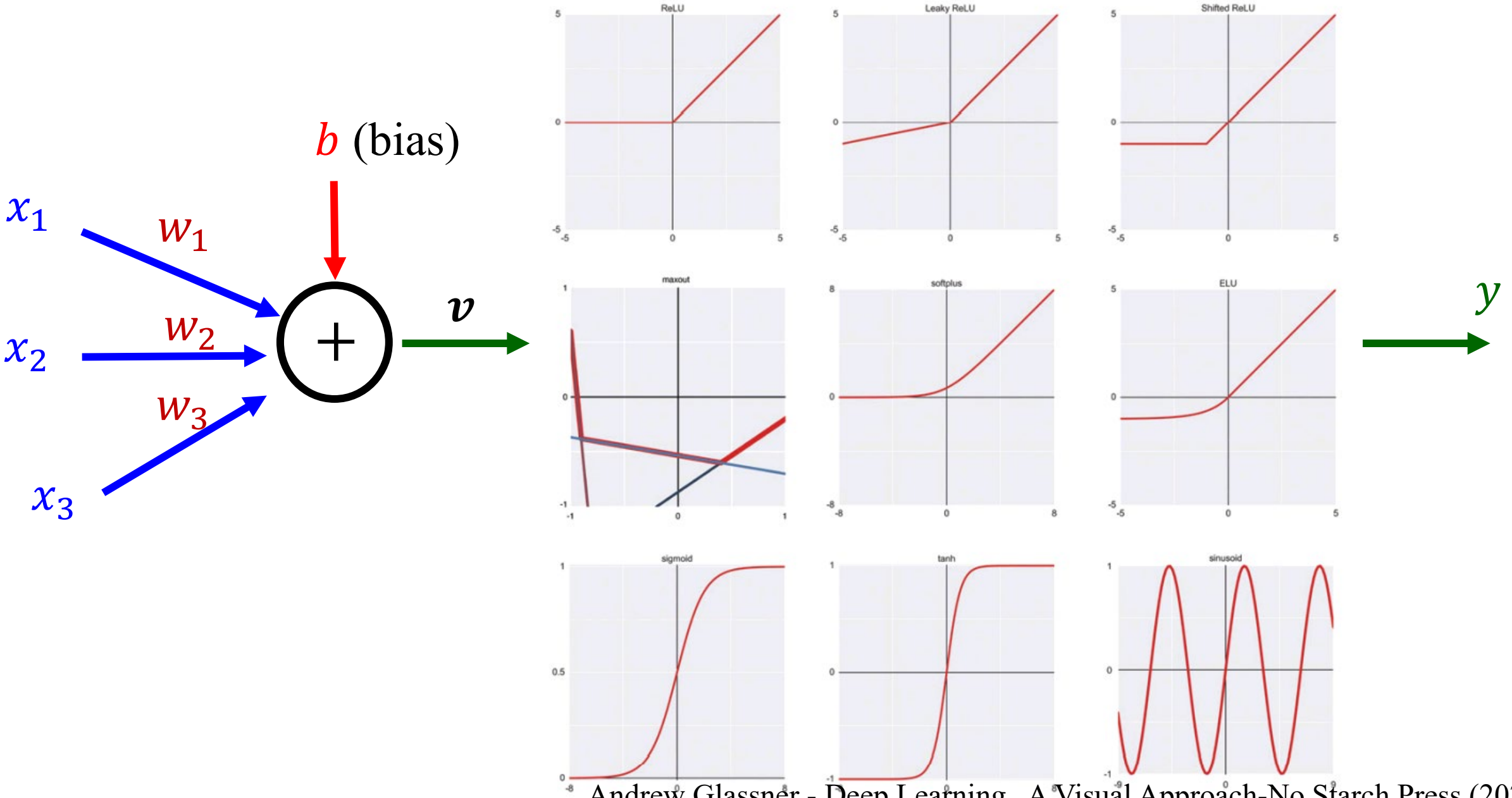
A simplified model receives three inputs



$$v = w_1x_1 + w_2x_2 + w_3x_3 + b$$

$y = \varphi(v)$, φ is the activation function

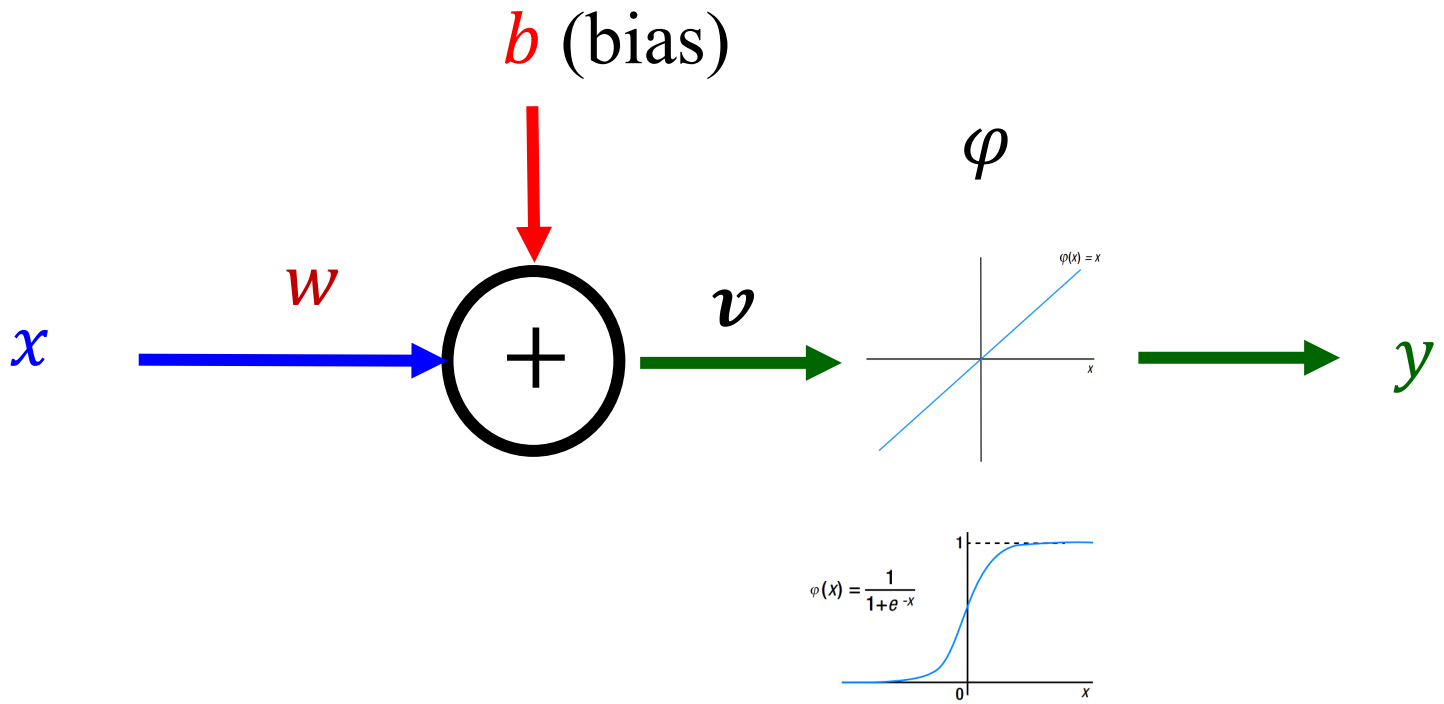
Activation Function Gallery



Outline

- Use linear regression to predict one variable from another.
- Upgrade the learning program with a faster and more efficient algorithm: gradient descent.
- Implement multiple linear regression with multiple inputs using gradient descent, least square loss, and maximal likelihood
- Nonlinear regression
- Determine the optimal order of regression model using cross validation

Perceptron

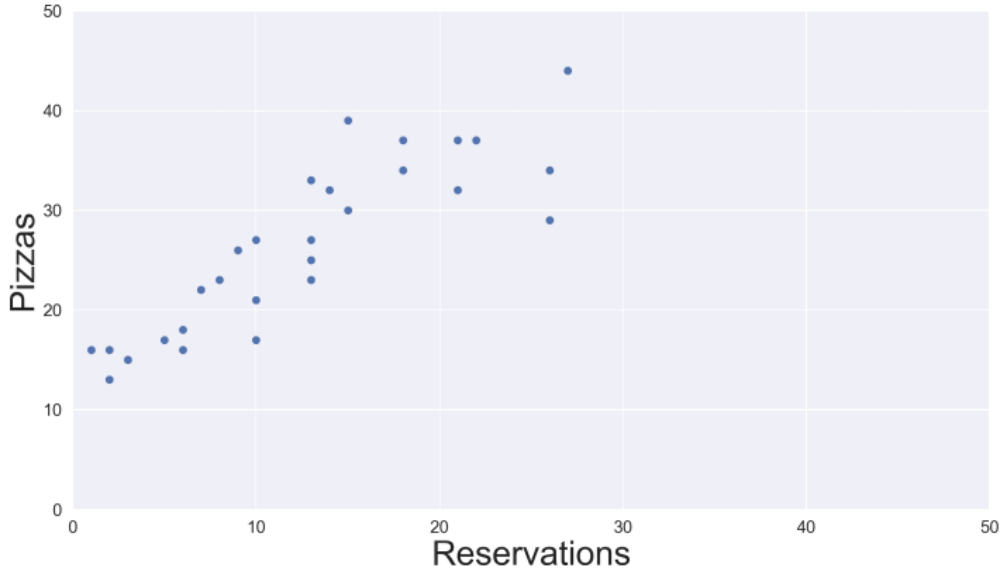


$$v = xw + b$$

$y = \varphi(v)$, φ is the activation function

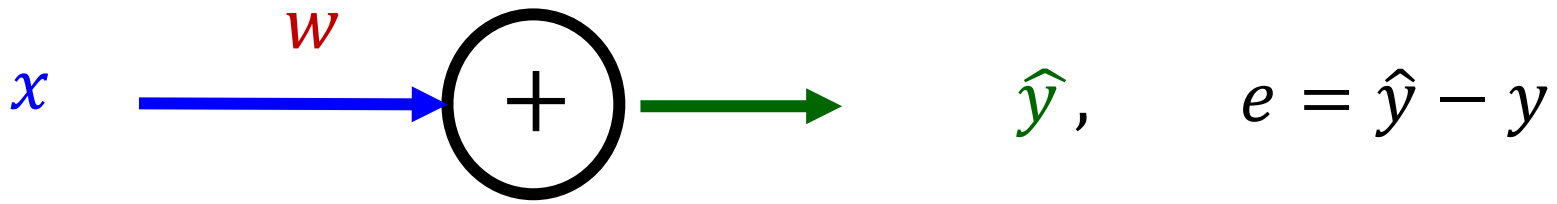
Supervised Pizza

- Roberto dreams of a program that looks at historical data, grasps the relation between reserved seats and pizzas, and uses it to forecast tonight's pizza sales from today's reservations.
- We can solve Roberto's pizza forecasting problem by training a supervised learning algorithm with a bunch of labeled examples.



Reservations	Pizzas
13	33
2	16
14	32
23	51
13	27
1	16
18	34
10	17
26	29
3	15
3	15
21	32
7	22
22	37
2	13
27	44
6	16
10	21
18	37
15	30
9	26
26	34
8	23
15	39
10	27
21	37
5	17
6	18
13	25
13	23

When $b = 0$ and $y = v$: Regression



$$\hat{y}_1 = x_1 w \Rightarrow \quad e_1 = \hat{y}_1 - y_1$$
$$\hat{y}_2 = x_2 w \Rightarrow \quad e_2 = \hat{y}_2 - y_2$$
$$\vdots$$
$$\hat{y}_N = x_N w \Rightarrow \quad e_N = \hat{y}_N - y_N$$

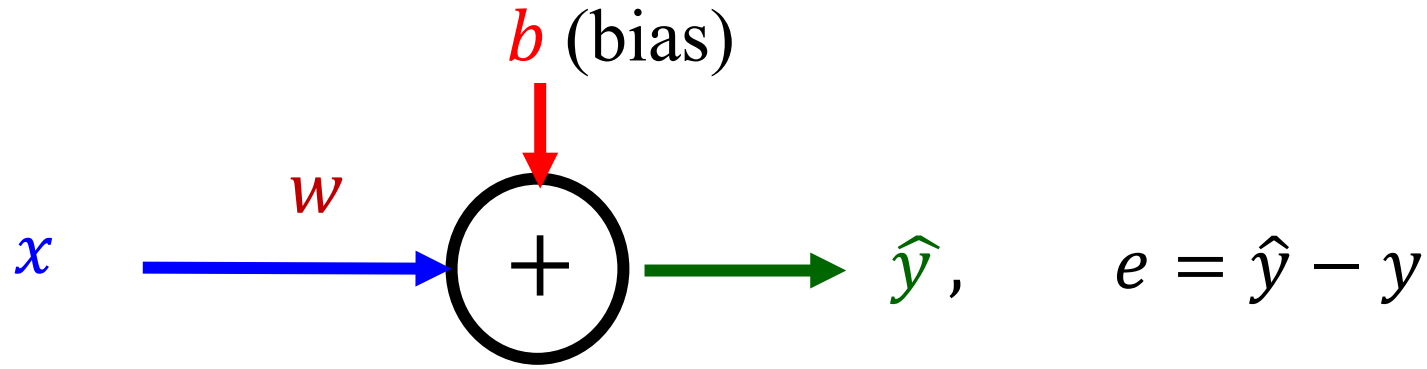
$$\text{Loss}(w) = \frac{1}{N}\sum_{n=1}^{N} {e_n}^2 = \frac{1}{N}\sum_{n=1}^{N} (x_n w - y_n)^2$$

```
def loss(X, Y, w):
    return np.average((predict(X, w) - Y) ** 2)
```

Iterative algorithm

```
def loss(X, Y, w):
    return np.average((predict(X, w) - Y) ** 2)

def train(X, Y, iterations, lr):
    w = 0
    for i in range(iterations):
        current_loss = loss(X, Y, w)
        print("Iteration %4d => Loss: %.6f" % (i, current_loss))

        if loss(X, Y, w + lr) < current_loss:
            w += lr
        elif loss(X, Y, w - lr) < current_loss:
            w -= lr
        else:
            return w

    raise Exception("Couldn't converge within %d iterations" % iterations)
```

When $b \neq 0$ and $y = v$: Regression



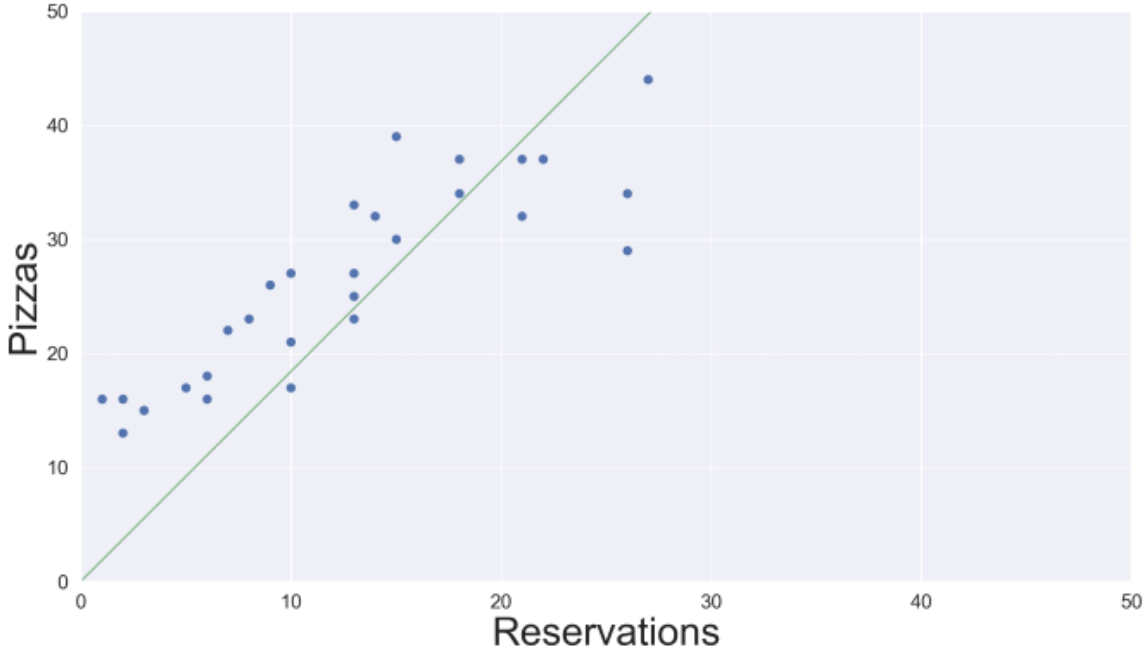
$$\hat{y}_1 = x_1 w + b \Rightarrow e_1 = \hat{y}_1 - y_1$$
$$\hat{y}_2 = x_2 w + b \Rightarrow e_2 = \hat{y}_2 - y_2$$
$$\vdots$$
$$\hat{y}_N = x_N w + b \Rightarrow e_N = \hat{y}_N - y_N$$

$$\text{Loss}(w, b) = \frac{1}{N}\sum_{n=1}^{N} {e_n}^2 = \frac{1}{N}\sum_{n=1}^{N} (x_n w + b - y_n)^2$$

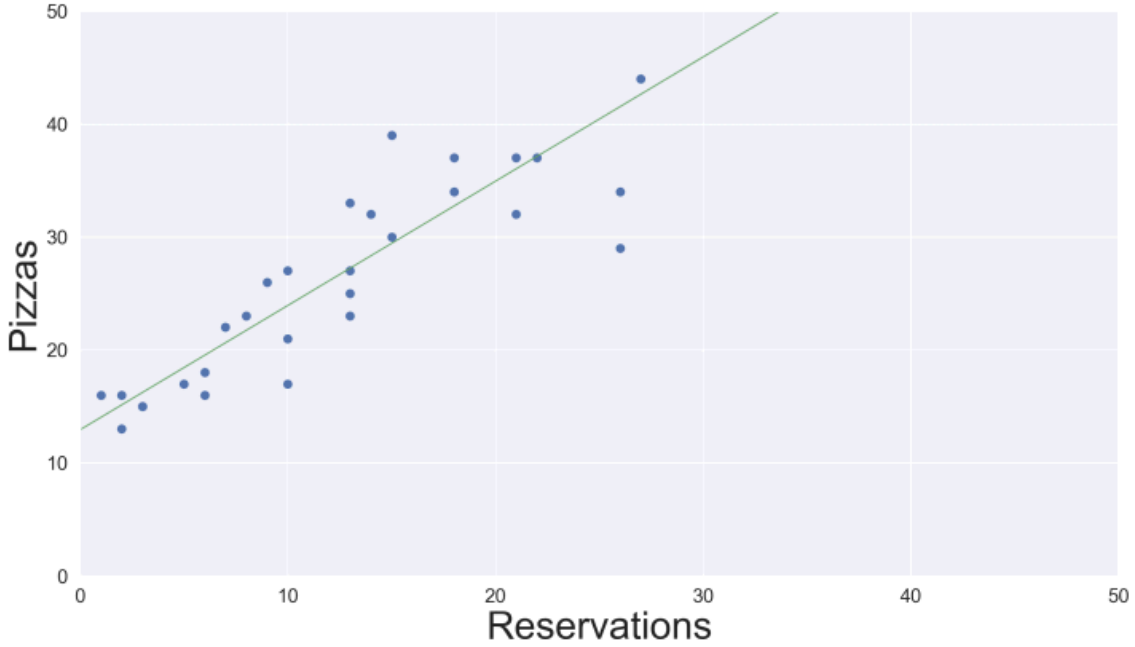
```python
def predict(X, w, b):
    return X * w + b

def loss(X, Y, w, b):
    return np.average((predict(X, w, b) - Y) ** 2)

def train(X, Y, iterations, lr):
    w = b = 0
    for i in range(iterations):
        current_loss = loss(X, Y, w, b)
        if i % 100 == 0:
              print(f'Iteration {i:4d} => Loss: {current_loss:.3f}')

        if loss(X, Y, w + lr, b) < current_loss:
            w += lr
        elif loss(X, Y, w - lr, b) < current_loss:
            w -= lr
        elif loss(X, Y, w, b + lr) < current_loss:
            b += lr
        elif loss(X, Y, w, b - lr) < current_loss:
            b -= lr
        else:
            return w, b

    raise Exception("Couldn't converge within %d iterations" % iterations)
```

Regression Results

$b = 0$



$b \neq 0$



Minimize Mean Squared Error Using Gradient Descent

- The loss function for output

$$\text{Loss}(w, b) = \frac{1}{N}\sum_{n=1}^{N}(x_n w + b - y_n)^2$$

- Minimize the loss function *Loss* w.r.t w and b

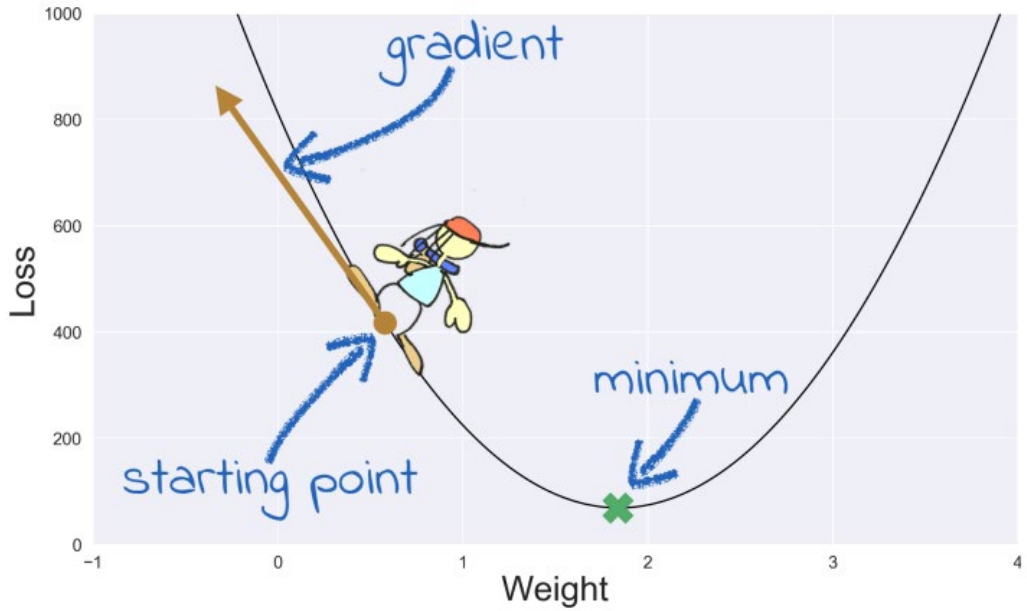
$$\frac{\partial Loss}{\partial w} = \frac{2}{N}\sum_{n=1}^{N}(x_n w + b - y_n)x_n = \frac{2}{N}\sum_{n=1}^{N} e_n x_n$$

$$\frac{\partial Loss}{\partial b} = \frac{2}{N}\sum_{n=1}^{N}(x_n w + b - y_n) = \frac{2}{N}\sum_{n=1}^{N} e_n$$

- The Batch Gradient Decent method

$$w^{(k+1)} = w^{(k)} - \alpha\frac{\partial Loss}{\partial w}$$

$$b^{(k+1)} = b^{(k)} - \alpha\frac{\partial Loss}{\partial b}$$



Minimize Mean Squared Error Using Gradient Descent

```
import numpy as np

def predict(X, w, b):
    return X * w + b

def loss(X, Y, w, b):
    return np.average((predict(X, w, b) - Y) ** 2)

def gradient(X, Y, w, b):
    w_gradient = 2 * np.average(X * (predict(X, w, b) - Y))
    b_gradient = 2 * np.average(predict(X, w, b) - Y)
    return (w_gradient, b_gradient)

def train(X, Y, iterations, lr):
    w = b = 0
    for i in range(iterations):
        print("Iteration %4d => Loss: %.10f" % (i, loss(X, Y, w, b)))
        w_gradient, b_gradient = gradient(X, Y, w, b)
        w -= w_gradient * lr
        b -= b_gradient * lr
    return w, b
```

Algorithm

1. Initialize the weights at adequate values.

2. Take the "input" from the training data of { input, correct output } and enter it to the neural network. Calculate the error $e_n = \hat{y}_n - y_n$

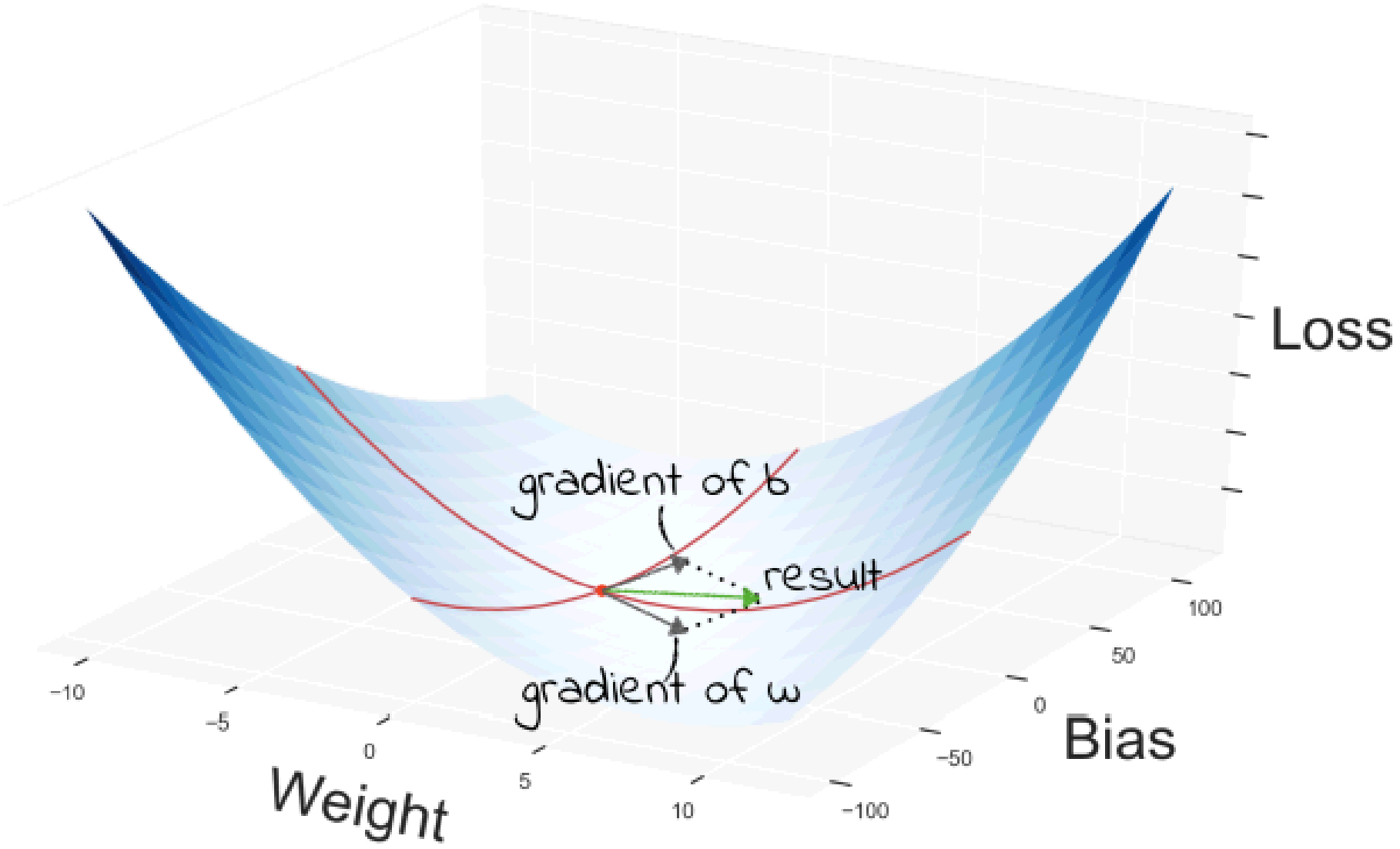
3. Calculate the $\Delta w = \alpha \frac{2}{N} \sum_{n=1}^{N} e_n x_n$, $\Delta b = \alpha \frac{2}{N} \sum_{n=1}^{N} e_n$

4. Adjust the weights as: $w \leftarrow w - \Delta w$, $b \leftarrow b - \Delta b$

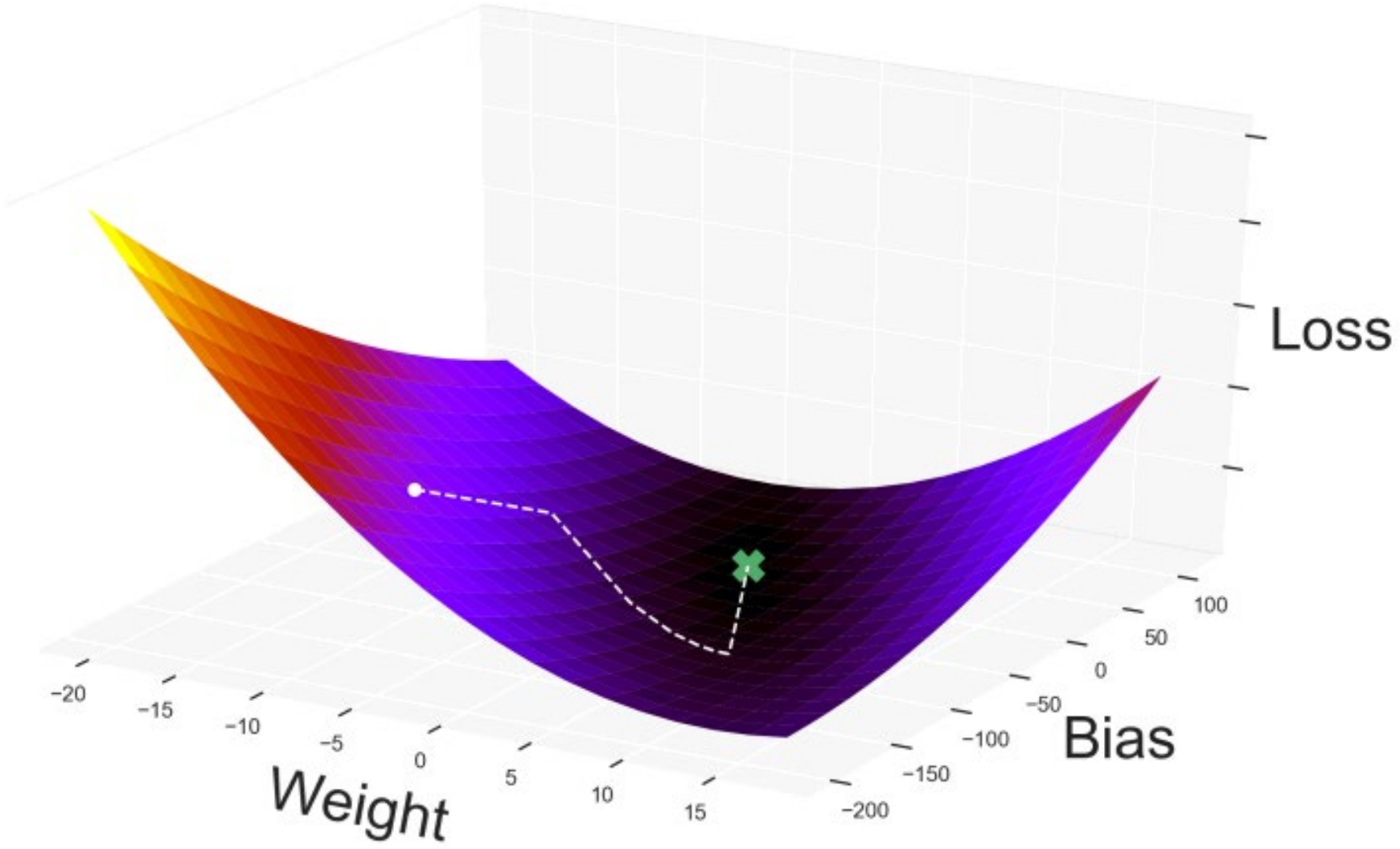
5. Perform Steps 2~4 for all training data.

6. Repeat Steps 2~5 until the error reaches an acceptable tolerance level.

Minimize Mean Squared Error Using Gradient Descent

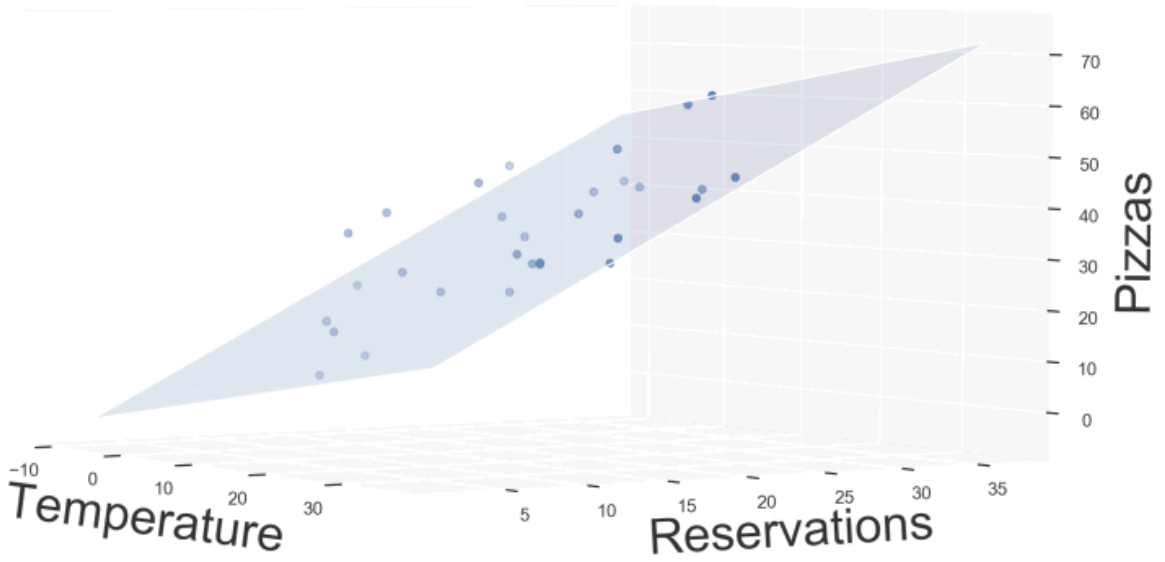


Minimize Mean Squared Error Using Gradient Descent

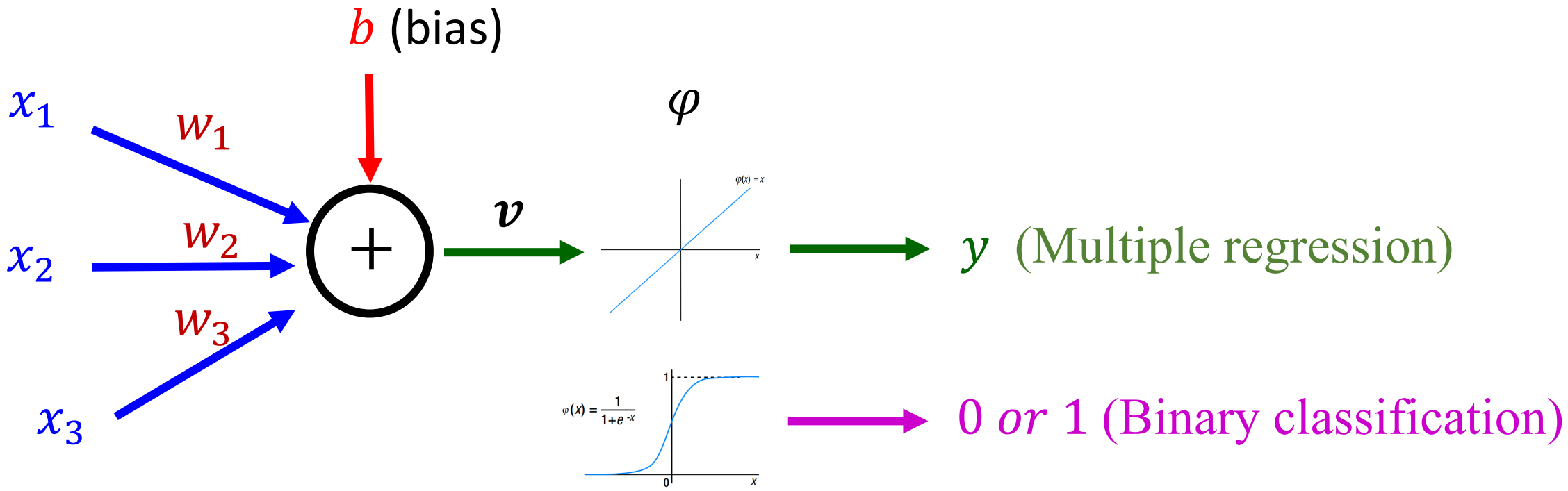


Hyperspace: Multiple inputs, single output

Reservations	Temperature	Tourists	Pizzas
13	26	9	44
2	14	6	23
14	20	3	28
23	25	9	60
13	24	8	42
1	12	2	5
18	23	9	51
10	18	10	44
26	24	3	42
3	14	1	9
3	12	3	14
21	27	5	43
7	17	3	22
22	21	1	34
2	12	4	16
27	26	2	46
6	15	4	26
10	21	7	33
18	18	3	29
15	26	8	43
9	20	6	37
26	25	9	62
8	21	10	47
15	22	7	38
10	20	2	22
21	21	1	29
5	12	7	34
6	14	9	38
13	19	4	30
13	20	3	28



Hyperspace: Multiple inputs, single output



$$v = x_1 w_1 + x_2 w_2 + x_3 w_3 + b = \boldsymbol{x}\boldsymbol{w} + b$$

where $\boldsymbol{x} = [x_1\ x_2\ x_3]$, $\boldsymbol{w} = \begin{bmatrix} w_1 \\ w_2 \\ w_3 \end{bmatrix}$

$y = \varphi(v)$, φ is the activation function

Features

Samples

$$X =$$

x_1	x_2	x_3	y
Reservations	Temperature	Tourists	Pizzas
13	26	9	44
2	14	6	23
14	20	3	28
23	25	9	60
13	24	8	42
1	12	2	5
18	23	9	51
10	18	10	44
26	24	3	42
3	14	1	9
3	12	3	14
21	27	5	43
7	17	3	22
22	21	1	34
2	12	4	16
27	26	2	46
6	15	4	26
10	21	7	33
18	18	3	29
15	26	8	43
9	20	6	37
26	25	9	62
8	21	10	47
15	22	7	38
10	20	2	22
21	21	1	29
5	12	7	34
6	14	9	38
13	19	4	30
13	20	3	28

When $b = 0$ and $y = v$: Multiple Regression

$$\hat{y}_1 = [x_{11}\ x_{12}\ x_{13}] \begin{bmatrix} w_1 \\ w_2 \\ w_3 \end{bmatrix} \Rightarrow \quad e_1 = \hat{y}_1 - y_1$$

$$\hat{y}_2 = [x_{21}\ x_{22}\ x_{23}] \begin{bmatrix} w_1 \\ w_2 \\ w_3 \end{bmatrix} \Rightarrow \quad e_2 = \hat{y}_2 - y_2$$

$$\vdots$$

$$\hat{y}_N = [x_{N1}\ x_{N2}\ x_{N3}] \begin{bmatrix} w_1 \\ w_2 \\ w_3 \end{bmatrix} \Rightarrow \quad e_N = \hat{y}_N - y_N$$

$$\text{Loss}(\boldsymbol{w}) = \frac{1}{N}\sum_{n=1}^{N}(\hat{y}_n - y_n)^2 \,, \hat{\boldsymbol{y}}_{N\times 1} = \boldsymbol{X}_{N\times 3}\boldsymbol{w}_{3\times 1}$$

```
import numpy as np
x1, x2, x3, y = np.loadtxt("pizza_3_vars.txt", skiprows=1, unpack=True)

x1.shape  # => (30, )
```

That dangling comma is NumPy's way of saying that these arrays have just one dimension. They are "array," as opposed to "matrix."

```
X = np.column_stack((x1, x2, x3))
X.shape  # => (30, 3)

Y = y.reshape(-1, 1)

Y.shape  # => (30, 1)
```

The ' -1 ' in y.reshape(-1, 1) means that if one dimension is -1, then NumPy will set it to whatever makes the other dimensions fit.

Minimize Mean Squared Error Using Gradient Descent

- The loss function for output

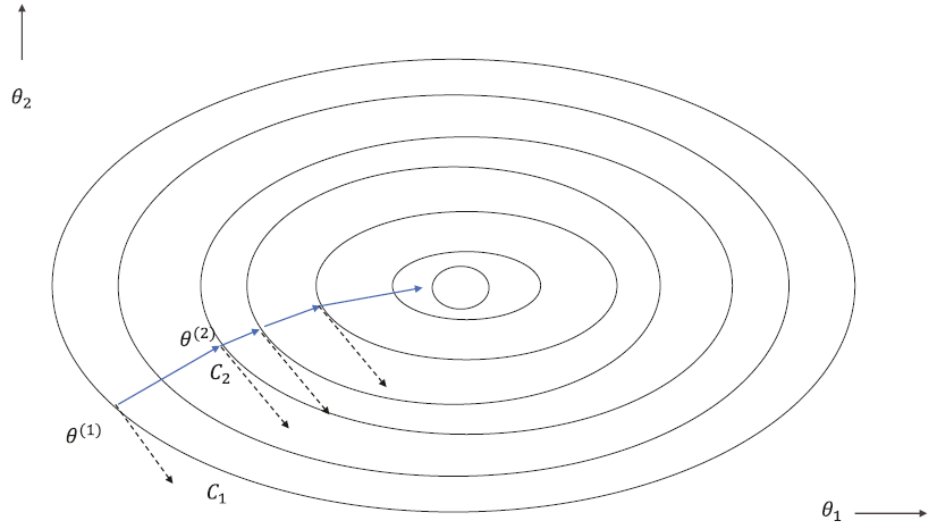
$$\text{Loss}(\boldsymbol{w}) = \frac{1}{N}\sum_{n=1}^{N}(x_{n1}w_1 + x_{n2}w_2 + x_{n3}w_3 - y_n)^2 = \frac{1}{N}\sum_{n=1}^{N} e_N^2$$

- Minimize the loss function *Loss* w.r.t w_1, w_2 and w_3

$$\frac{\partial Loss}{\partial \boldsymbol{w}} = \begin{bmatrix} \frac{\partial Loss}{\partial w_1} \\ \frac{\partial Loss}{\partial w_2} \\ \frac{\partial Loss}{\partial w_3} \end{bmatrix} = \begin{bmatrix} \frac{2}{N}\sum_{n=1}^{N} e_n x_{n1} \\ \frac{2}{N}\sum_{n=1}^{N} e_n x_{n2} \\ \frac{2}{N}\sum_{n=1}^{N} e_n x_{n3} \end{bmatrix} = \frac{2}{N} X_{3\times \mathrm{N}}^{\mathrm{T}} \times e_{N\times 1}$$

- The Batch Gradient Decent method

$$\boldsymbol{w}^{(k+1)} = \boldsymbol{w}^{(k)} - \alpha \frac{\partial Loss}{\partial \boldsymbol{w}}$$



$$X^{T}_{3\times N} \times e_{N\times 1}$$

x_1

Reservations
13
2
14
23
13
1
18
10
26
3
3
21
7
22
2
27
6
10
18
15
9
26
8
15
10
21
5
6
13
13

x_2

Temperature
26
14
20
25
24
12
23
18
24
14
12
27
17
21
12
26
15
21
18
26
20
25
21
22
20
21
12
14
19
20

x_3

Pizzas
44
23
28
60
42
5
51
44
42
9
14
43
22
34
16
46
26
33
29
43
37
62
47
38
22
29
34
38
30
28

$$\begin{bmatrix} e_1 \\ e_2 \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ e_{N-1} \\ e_N \end{bmatrix}$$

```python
import numpy as np

def predict(X, w):
    return np.matmul(X, w)

def loss(X, Y, w):
    return np.average((predict(X, w) - Y) ** 2)

def gradient(X, Y, w):
    return 2 * np.matmul(X.T, (predict(X, w) - Y)) / X.shape[0]

def train(X, Y, iterations, lr):
    w = np.zeros((X.shape[1], 1))
    for i in range(iterations):
        print("Iteration %4d => Loss: %.20f" % (i, loss(X, Y, w)))
        w -= gradient(X, Y, w) * lr
    return w

x1, x2, x3, y = np.loadtxt("pizza_3_vars.txt", skiprows=1, unpack=True)
X = np.column_stack((x1, x2, x3))
Y = y.reshape(-1, 1)
w = train(X, Y, iterations=100000, lr=0.001)
```

```
a_number = loss(X, Y, w)
a_number.shape  # => ()
```

this is a scalar, so it has no dimensions

```
w.shape  # => (3, 1)
```

When $b \equiv w_1 \neq 0$ and $y = v$: Multiple Regression

pizza_3_vars.txt

Bias	Reservations	Temperature	Tourists	Pizzas
1	13	26	9	44
1	2	14	6	23
1	14	20	3	28
...	...	...	...	...
1	13	20	3	28

$$\hat{y}_1 = [1\ x_{12}\ x_{13}\ x_{14}] \begin{bmatrix} w_1 \\ w_2 \\ w_3 \\ w_4 \end{bmatrix} \Rightarrow \quad e_1 = \hat{y}_1 - y_1$$

$$\vdots$$

$$\hat{y}_N = [1\ x_{N2}\ x_{N3}\ x_{N4}] \begin{bmatrix} w_1 \\ w_2 \\ w_3 \\ w_4 \end{bmatrix} \Rightarrow \quad e_N = \hat{y}_N - y_N$$

$$\text{Loss}(\boldsymbol{w}) = \frac{1}{N}\sum_{n=1}^{N}(\hat{y}_n - y_n)^2 \,, \hat{\boldsymbol{y}}_{N\times 1} = [\boldsymbol{1}_{N\times 1} \quad \boldsymbol{X}_{N\times 3}] \times \boldsymbol{w}_{4\times 1}$$

```
x1, x2, x3, y = np.loadtxt("pizza_3_vars.txt", skiprows=1, unpack=True)
X = np.column_stack((np.ones(x1.size), x1, x2, x3))
Y = y.reshape(-1, 1)
w = train(X, Y, iterations=100000, lr=0.001)

print("\nWeights: %s" % w.T)
print("\nA few predictions:")
for i in range(5):
    print("X[%d] -> %.4f (label: %d)" % (i, predict(X[i], w), Y[i]))
```

Minimize Mean Squared Error Using Gradient Descent

- The loss function for output

$$\text{Loss}(\boldsymbol{w}) = \frac{1}{N}\sum_{n=1}^{N}(1 \times w_1 + x_{n2}w_2 + x_{n3}w_3 + x_{n4}w_4 - y_n)^2 = \frac{1}{N}\sum_{n=1}^{N} e_N^2$$

- Minimize the loss function *Loss* w.r.t w_1, w_2, w_3 and w_4

$$\frac{\partial Loss}{\partial \boldsymbol{w}} = \begin{bmatrix} \frac{\partial Loss}{\partial w_1} \\ \frac{\partial Loss}{\partial w_2} \\ \frac{\partial Loss}{\partial w_3} \\ \frac{\partial Loss}{\partial w_4} \end{bmatrix} = \begin{bmatrix} \frac{2}{N}\sum_{n=1}^{N} e_n \\ \frac{2}{N}\sum_{n=1}^{N} e_n x_{n2} \\ \frac{2}{N}\sum_{n=1}^{N} e_n x_{n3} \\ \frac{2}{N}\sum_{n=1}^{N} e_n x_{n4} \end{bmatrix} = \frac{2}{N}[\mathbf{1}_{N\times 1} \quad \boldsymbol{X}_{N\times 3}]^{\mathrm{T}}_{4\times \mathrm{N}} \times e_{N\times 1}$$

- The Batch Gradient Decent method

$$\boldsymbol{w}^{(k+1)} = \boldsymbol{w}^{(k)} - \alpha \frac{\partial Loss}{\partial \boldsymbol{w}}$$

$$[\mathbf{1}_{N\times 1} \quad \boldsymbol{X}_{N\times 3}]^{\mathrm{T}}_{4\times \mathrm{N}} \quad \times \quad e_{N\times 1}$$

x_1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

x_2

Reservations
13
2
14
23
13
1
18
10
26
3
3
21
7
22
2
27
6
10
18
15
9
26
8
15
10
21
5
6
13
13

x_3

Temperature
26
14
20
25
24
12
23
18
24
14
12
27
17
21
12
26
15
21
18
26
20
25
21
22
20
21
12
14
19
20

x_4

Pizzas
44
23
28
60
42
5
51
44
42
9
14
43
22
34
16
46
26
33
29
43
37
62
47
38
22
29
34
38
30
28

$$\begin{bmatrix} e_1 \\ e_2 \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ e_{N-1} \\ e_N \end{bmatrix}$$

```
import numpy as np

def predict(X, w):
    return np.matmul(X, w)

def loss(X, Y, w):
    return np.average((predict(X, w) - Y) ** 2)

def gradient(X, Y, w):
    return 2 * np.matmul(X.T, (predict(X, w) - Y)) / X.shape[0]

def train(X, Y, iterations, lr):
    w = np.zeros((X.shape[1], 1))
    for i in range(iterations):
        print("Iteration %4d => Loss: %.20f" % (i, loss(X, Y, w)))
        w -= gradient(X, Y, w) * lr
    return w

x1, x2, x3, y = np.loadtxt("pizza_3_vars.txt", skiprows=1, unpack=True)
X = np.column_stack((x1, x2, x3))
Y = y.reshape(-1, 1)
w = train(X, Y, iterations=100000, lr=0.001)
```

Linear Modelling:
A Least Squares Approach

Chap1. A First Course in Machine Learning, 2ed, Simon Rogers and Mark Girolami, 2017

Lecture Outline

- Minimize the average squared loss function
- Vector/Matrix notation
- Non-linear response from a linear model
- Generalization and over-fitting
- K-fold cross-validation
- Regularized least squares

The Average Squared Loss Function

- Given the dataset $(x_1, t_1),(x_2, t_2), \ldots ,(x_N, t_N)$
- We want to minimize the average squared loss function across the whole dataset

$$L = \frac{1}{N}\sum_{n=1}^{N}(t_n - f(x_n; \omega_0; \omega_1))^2$$

$$= \frac{1}{N}\sum_{n=1}^{N}(t_n - (\omega_0 + \omega_1 x_n))^2$$

$$= \frac{1}{N}\sum_{n=1}^{N}(\omega_1^2 x_n^2 + 2\omega_1 x_n \omega_0 - 2\omega_1 x_n t_n + \omega_0^2 - 2\omega_0 t_n + t_n^2)$$

$$= \frac{1}{N}\sum_{n=1}^{N}(\omega_1^2 x_n^2 + 2\omega_1 x_n(\omega_0 - t_n) + \omega_0^2 - 2\omega_0 t_n + t_n^2)$$

Minimize the Average Squared Loss Function

$$\frac{\partial L}{\partial \omega_0} = 2\omega_0 + 2\omega_1 \frac{1}{N}\left(\sum_{n=1}^{N} x_n\right) - \frac{2}{N}\left(\sum_{n=1}^{N} t_n\right) = 0$$

Define $\quad \bar{t} \equiv \frac{1}{N}\left(\sum_{n=1}^{N} t_n\right), \qquad \bar{x} \equiv \frac{1}{N}\left(\sum_{n=1}^{N} x_n\right)$

$$\Rightarrow \hat{\omega}_0 = \frac{1}{N}\left(\sum_{n=1}^{N} t_n\right) - \omega_1 \frac{1}{N}\left(\sum_{n=1}^{N} x_n\right) = \bar{t} - \omega_1 \bar{x}$$

Minimize the Average Squared Loss Function

Define $\quad \overline{x^2} \equiv \frac{1}{N}\left(\sum_{n=1}^{N} x_n^2\right), \qquad \overline{xt} \equiv \frac{1}{N}\left(\sum_{n=1}^{N} x_n\, t_n\right)$

$$\frac{\partial L}{\partial \omega_1} = 2\omega_1 \frac{1}{N}\left(\sum_{n=1}^{N} x_n^2\right) + \frac{2}{N}\left(\sum_{n=1}^{N} x_n(\hat{\omega}_0 - t_n)\right)$$

$$= 2\omega_1 \overline{x^2} + \frac{2}{N}\left(\sum_{n=1}^{N} x_n(\bar{t} - \omega_1 \bar{x} - t_n)\right)$$

$$= 2\omega_1 \overline{x^2} + 2\bar{x}\bar{t} - 2\omega_1 \bar{x}\bar{x} - 2\overline{xt} = 0$$

$$\Rightarrow \hat{\omega}_1 = \frac{\overline{xt} - \bar{x}\bar{t}}{\overline{x^2} - \bar{x}^2}$$

Ensure a Minimum

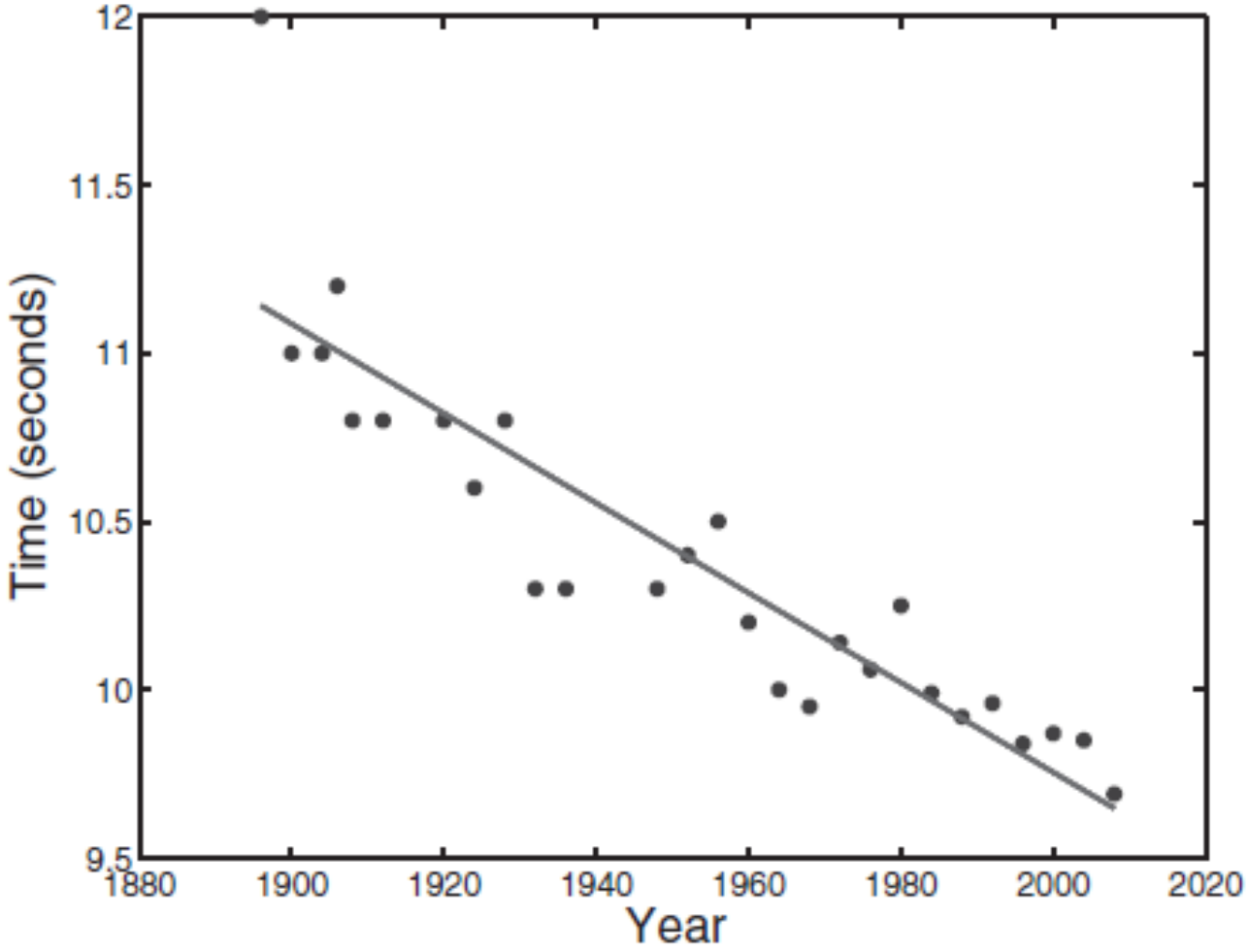
- Examine the second derivatives to ensure a minimum

$$\frac{\partial^2 L}{\partial \omega_0^2} = 2 > 0, \qquad \frac{\partial^2 L}{\partial \omega_1^2} = \frac{1}{N}\left(\sum_{n=1}^{N} x_n^2\right) > 0$$

$\Rightarrow$ a minimum of the loss function

Least Squares Fit to the Olympic men's 100m Data

n	x_n	t_n	$x_n t_n$	x_n^2
1	1896	12.00	22752.0	3.5948×10^6
2	1900	11.00	20900.0	3.6100×10^6
3	1904	11.00	20944.0	3.6252×10^6
4	1906	11.20	21347.2	3.6328×10^6
5	1908	10.80	20606.4	3.6405×10^6
6	1912	10.80	20649.6	3.6557×10^6
7	1920	10.80	20736.0	3.6864×10^6
8	1924	10.60	20394.4	3.7018×10^6
9	1928	10.80	20822.4	3.7172×10^6
10	1932	10.30	19899.6	3.7326×10^6
11	1936	10.30	19940.8	3.7481×10^6
12	1948	10.30	20064.4	3.7947×10^6
13	1952	10.40	20300.8	3.8103×10^6
14	1956	10.50	20538.0	3.8259×10^6
15	1960	10.20	19992.0	3.8416×10^6
16	1964	10.00	19640.0	3.8573×10^6
17	1968	9.95	19581.6	3.8730×10^6
18	1972	10.14	19996.1	3.8888×10^6
19	1976	10.06	19878.6	3.9046×10^6
20	1980	10.25	20295.0	3.9204×10^6
21	1984	9.99	19820.2	3.9363×10^6
22	1988	9.92	19721.0	3.9521×10^6
23	1992	9.96	19840.3	3.9681×10^6
24	1996	9.84	19640.6	3.9840×10^6
25	2000	9.87	19740.0	4.0000×10^6
26	2004	9.85	19739.4	4.0160×10^6
27	2008	9.69	19457.5	4.0321×10^6
$(1/N)\sum_{n=1}^{N}$	1952.37	10.39	20268.1	3.8130×10^6

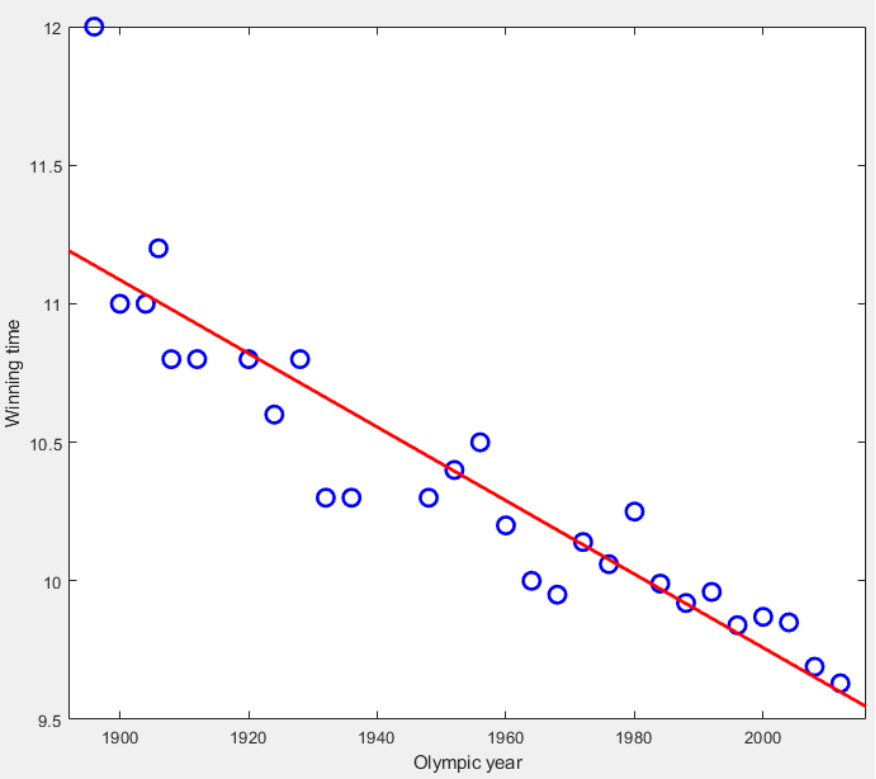


Least Squares Fit to the Olympic men's 100m Data

$$\hat{\omega}_1 = \frac{\overline{xt} - \bar{x}\bar{t}}{\overline{x^2} - \bar{x}^2} = \frac{20268.1 - 1952.37 \times 10.39}{3.813 \times 10^6 - 1952.37 \times 1952.37} = -0.0133$$

$$\hat{\omega}_0 = \bar{t} - \omega_1 \bar{x} = 10.39 - (-0.0133) \times 1952.37 = 35.416$$

$$f(x; \omega_0; \omega_1) = \omega_0 + \omega_1 x = 35.416 - 0.0133x$$

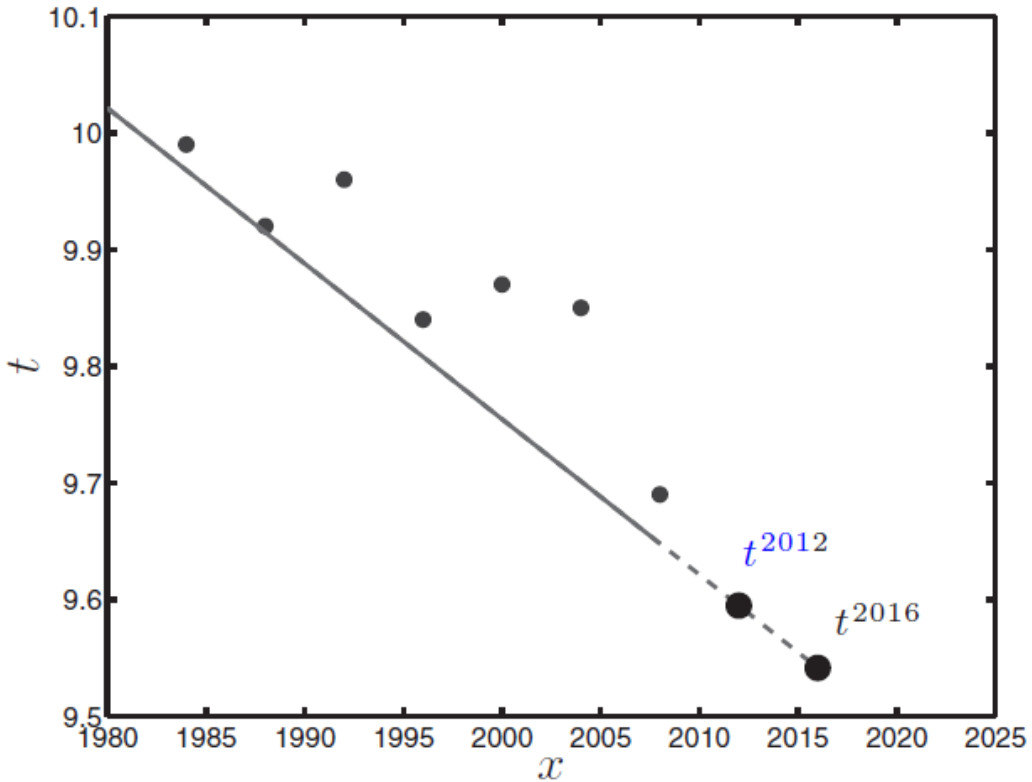


Making Predictions

- Predict the winning times at the 2012 and 2016 Olympics,

$$t^{2012} = f(2012; \omega_0; \omega_1) = 35.416 - 0.0133 \times 2012 = 9.595$$

$$t^{2016} = f(2016; \omega_0; \omega_1) = 35.416 - 0.0133 \times 2016 = 9.541$$



Vector/Matrix Notation

- Define $\boldsymbol{\omega} = \begin{bmatrix} \omega_0 \\ \omega_1 \end{bmatrix}, \boldsymbol{x}_n = \begin{bmatrix} 1 \\ x_n \end{bmatrix} \Rightarrow f(x_n; \omega_0; \omega_1) = \omega_0 + \omega_1 x_n = \boldsymbol{\omega}^T \boldsymbol{x}_n$

- Define $\boldsymbol{X} = \begin{bmatrix} \boldsymbol{x}_1^T \\ \boldsymbol{x}_2^T \\ \vdots \\ \boldsymbol{x}_N^T \end{bmatrix} = \begin{bmatrix} 1 & x_1 \\ 1 & x_2 \\ \vdots & \vdots \\ 1 & x_N \end{bmatrix}, \boldsymbol{t} = \begin{bmatrix} t_1 \\ t_2 \\ \vdots \\ t_N \end{bmatrix} \Rightarrow \boldsymbol{X}\boldsymbol{\omega} = \begin{bmatrix} \omega_0 + \omega_1 x_1 \\ \omega_0 + \omega_1 x_2 \\ \vdots \\ \omega_0 + \omega_1 x_N \end{bmatrix}$

- Therefore

$$L = \frac{1}{N} \sum_{n=1}^{N} (t_n - f(x_n; \omega_0; \omega_1))^2$$

$$= \frac{1}{N} \sum_{n=1}^{N} (t_n - (\omega_0 + \omega_1 x_n))^2 = \frac{1}{N} (\boldsymbol{t} - \boldsymbol{X}\boldsymbol{\omega})^T (\boldsymbol{t} - \boldsymbol{X}\boldsymbol{\omega})$$

Vector/Matrix Notation

- $$L = \frac{1}{N}(\boldsymbol{t} - \boldsymbol{X\omega})^T(\boldsymbol{t} - \boldsymbol{X\omega}) = \frac{1}{N}(\boldsymbol{X\omega} - \boldsymbol{t})^T(\boldsymbol{X\omega} - \boldsymbol{t})$$
$$= \frac{1}{N}\left((\boldsymbol{X\omega})^T - \boldsymbol{t}^T\right)(\boldsymbol{X\omega} - \boldsymbol{t})$$
$$= \frac{1}{N}(\boldsymbol{X\omega})^T\boldsymbol{X\omega} - \frac{1}{N}\boldsymbol{t}^T\boldsymbol{X\omega} - \frac{1}{N}(\boldsymbol{X\omega})^T\boldsymbol{t} + \frac{1}{N}\boldsymbol{t}^T\boldsymbol{t}$$
$$= \frac{1}{N}\boldsymbol{\omega}^T\boldsymbol{X}^T\boldsymbol{X\omega} - \frac{2}{N}\boldsymbol{\omega}^T\boldsymbol{X}^T\boldsymbol{t} + \frac{1}{N}\boldsymbol{t}^T\boldsymbol{t}$$

- $$\Rightarrow \quad \frac{\partial L}{\partial \boldsymbol{\omega}} = \frac{2}{N}\boldsymbol{X}^T\boldsymbol{X\omega} - \frac{2}{N}\boldsymbol{X}^T\boldsymbol{t} = \boldsymbol{0} \Rightarrow \boldsymbol{(X^TX)\omega = X^Tt}$$

$$\Rightarrow \quad \boldsymbol{\hat{\omega} = (X^TX)^{-1}X^Tt}$$

- Making predictions: $t_{new} = \hat{\boldsymbol{\omega}}^T\boldsymbol{x}_{new}$

$f(\mathbf{w})$	$\frac{\partial f}{\partial \mathbf{w}}$
$\mathbf{w}^\top\mathbf{x}$	$\mathbf{x}$
$\mathbf{x}^\top\mathbf{w}$	$\mathbf{x}$
$\mathbf{w}^\top\mathbf{w}$	$2\mathbf{w}$
$\mathbf{w}^\top\mathbf{C}\mathbf{w}$	$2\mathbf{C}\mathbf{w}$

Non-linear Response from a Linear Model

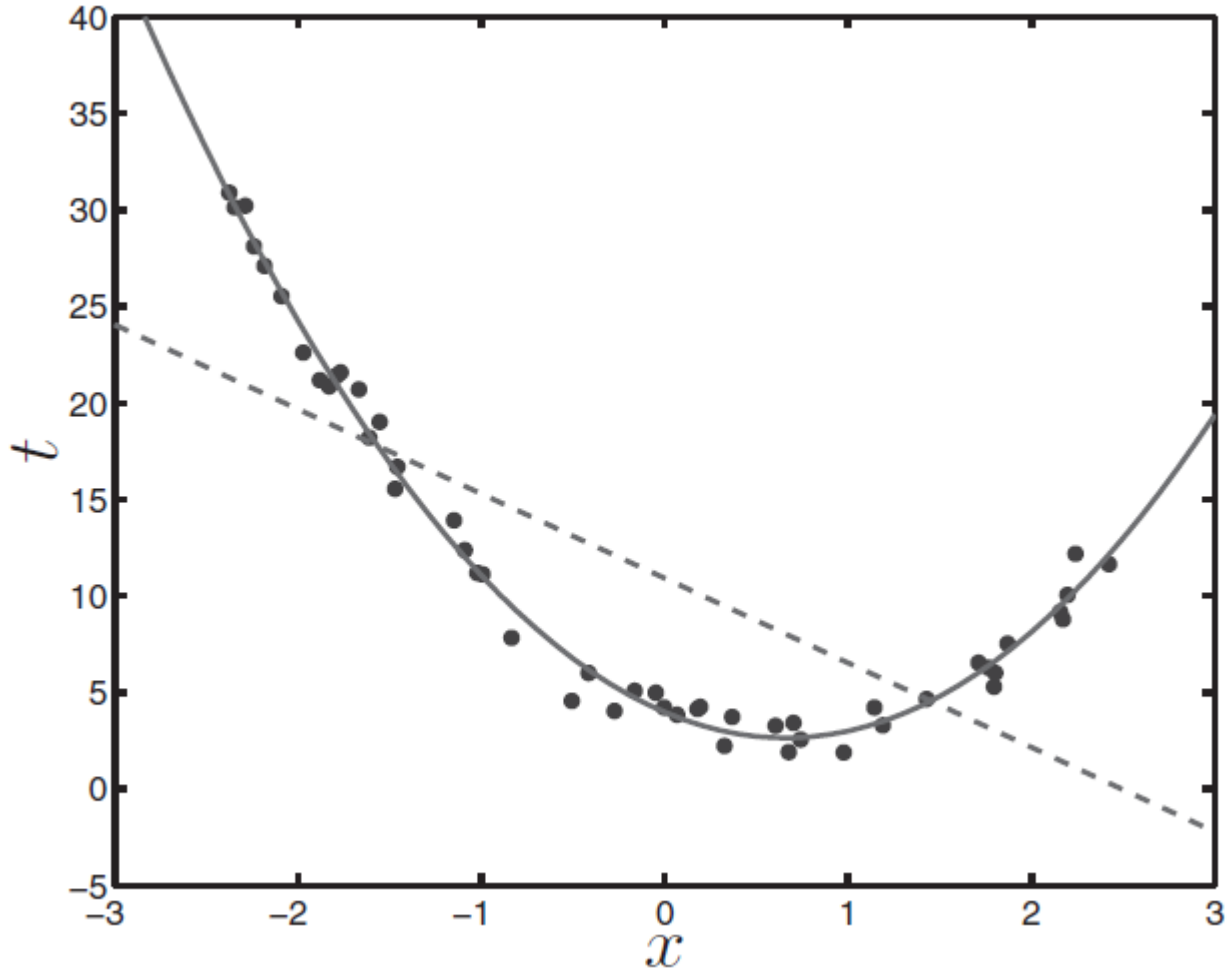
- Let a model be still linear in the parameters, but the function we are fitting is quadratic in the data:

$$\boldsymbol{x}_n = \begin{bmatrix} 1 \\ x_n \\ x_n^2 \end{bmatrix}, \boldsymbol{X} = \begin{bmatrix} \boldsymbol{x}_1^T \\ \boldsymbol{x}_2^T \\ \vdots \\ \boldsymbol{x}_N^T \end{bmatrix} = \begin{bmatrix} 1 & x_1 & x_1^2 \\ 1 & x_2 & x_2^2 \\ \vdots & \vdots & \vdots \\ 1 & x_N & x_N^2 \end{bmatrix}, \boldsymbol{\omega} = \begin{bmatrix} \omega_0 \\ \omega_1 \\ \omega_2 \end{bmatrix},$$

$$\Rightarrow f(x_n; \omega_0; \omega_1; \omega_2) = \omega_0 + \omega_1 x_n + \omega_2 x_n^2 = \boldsymbol{\omega}^T \boldsymbol{x}_n$$

Non-linear Response from a Linear Model

Example of linear (dashed) and quadratic (solid) models fitted to a dataset generated from a quadratic function.



Non-linear Response from a Linear Model

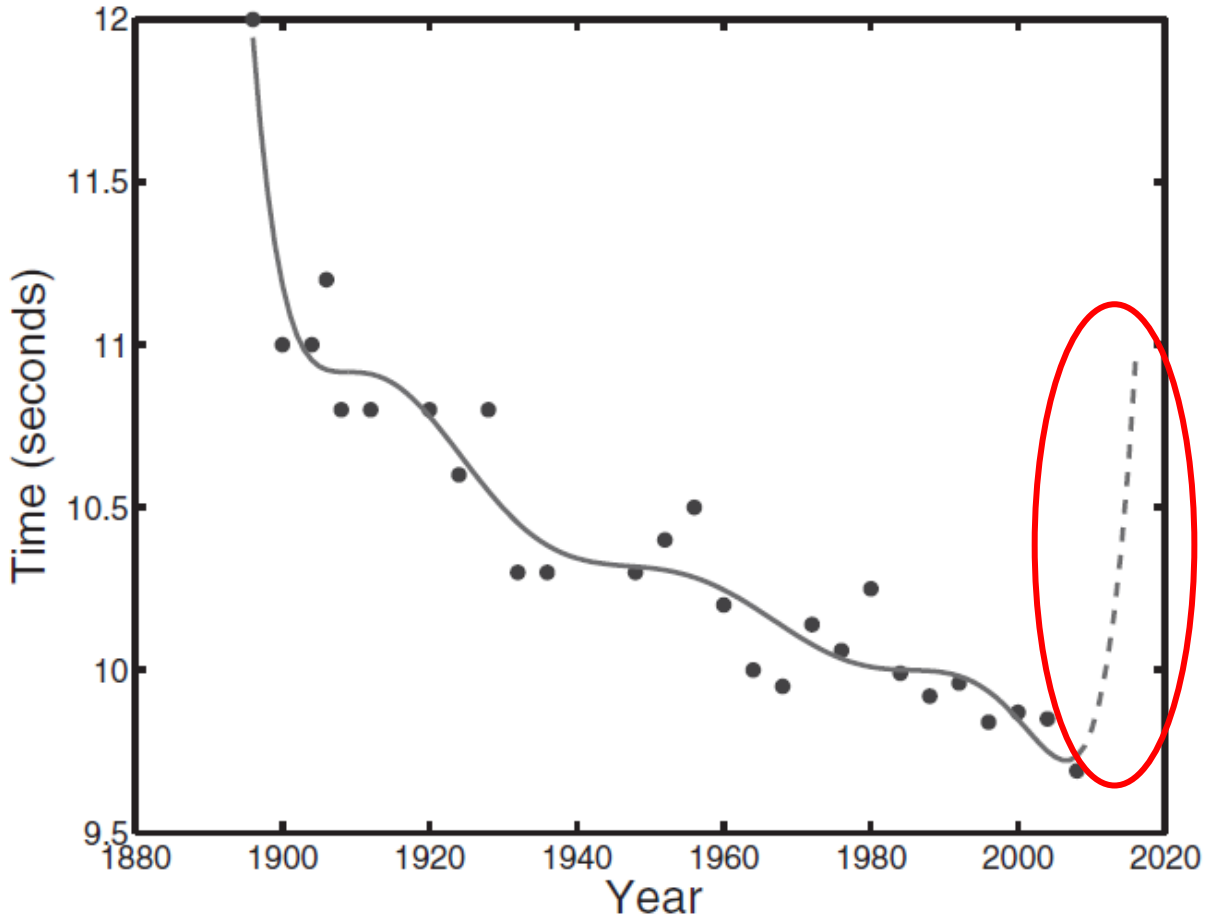
- More generally, for a Kth order polynomial, the augmented data matrix

$$\boldsymbol{x}_n = \begin{bmatrix} x_n^0 \\ x_n^1 \\ \vdots \\ x_n^K \end{bmatrix}, x_n^0 = 1, \boldsymbol{X} = \begin{bmatrix} \boldsymbol{x}_1^T \\ \boldsymbol{x}_2^T \\ \vdots \\ \boldsymbol{x}_N^T \end{bmatrix} = \begin{bmatrix} x_1^0 & x_1^1 & x_1^2 & \cdots & x_1^K \\ x_2^0 & x_2^1 & x_2^2 & \cdots & x_2^K \\ \vdots & \vdots & \vdots & \cdots & \vdots \\ x_N^0 & x_N^1 & x_N^2 & & x_N^K \end{bmatrix}$$

$$\Rightarrow f(x_n; \omega_0; \omega_1; \cdots; \omega_K) = \omega_0 x_n^0 + \omega_1 x_n^1 + \cdots + \omega_K x_n^K = \sum_{k=0}^{K} \omega_k x_n^k$$

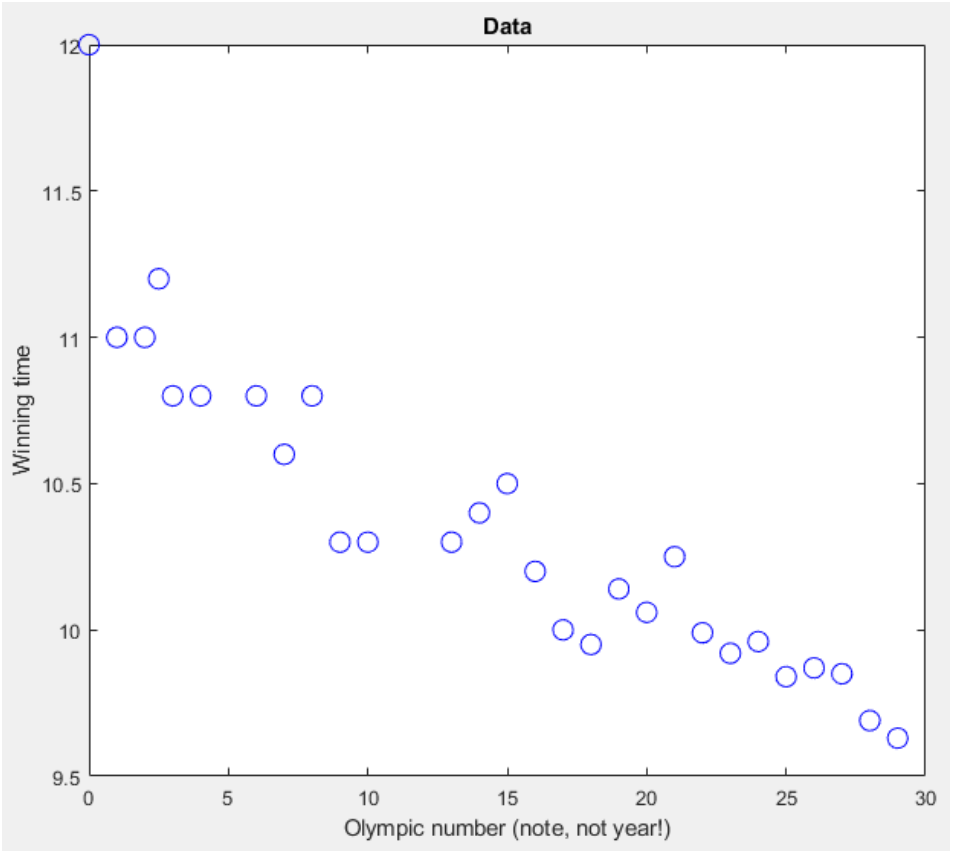
Eighth-order Polynomial Fitted to the Olympic 100m men's Data

- The 8th-order polynomial gets closer to the observed data than the 1st-order polynomial (original model): a lower value of the loss function: $L^8 = 0.459 < L^1 = 1.358$
- Predictions (dashed line) do not look sensible, particularly outside the range of the observed data.

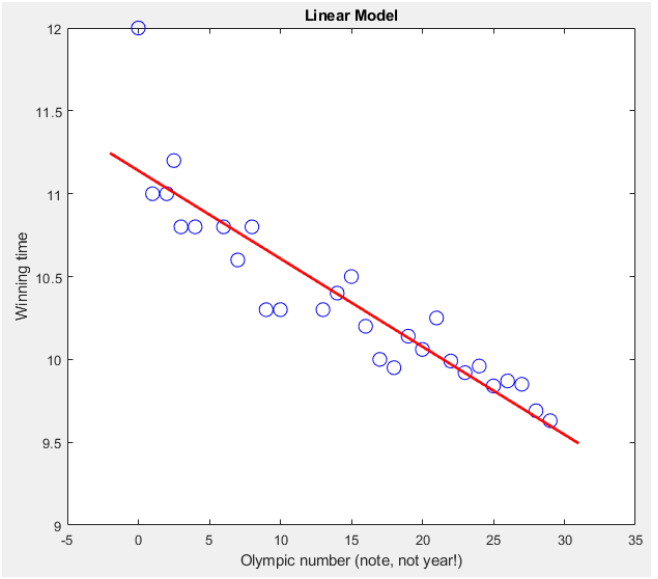


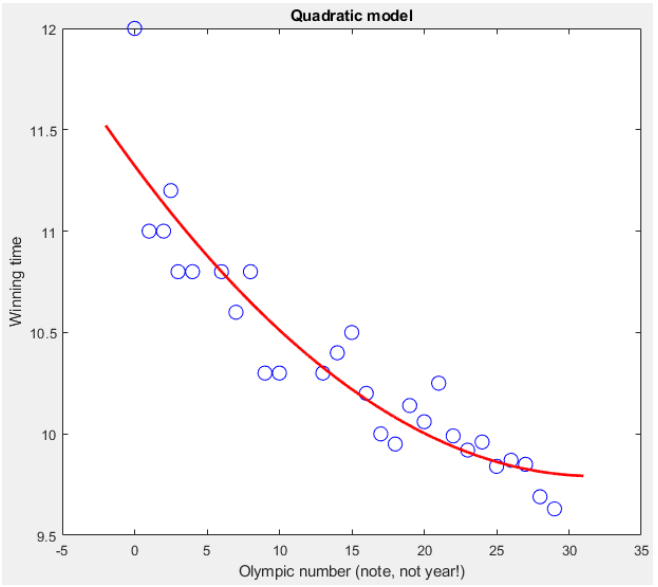
MATLAB script: olymppoly.m

```
%% olymppoly.m  From A First Course in Machine Learning, Chapter 1.
clc;clear all;close all;
%% Load the Olympic data and extract the mens 100m data
load ../data/olympics/male100.csv
x = male100(:,1); % Olympic years
t = male100(:,2); % Winning times
% Rescale x for numerical reasons
x = x - x(1); x = x./4;
% Plot the data
plot(x,t,'bo','markersize',10);
xlabel('Olympic number (note, not year!)');
ylabel('Winning time');
title('Data')
pause(3)
%% Linear model
plotx = [x(1)-2:0.01:x(end)+2]'; X = [];
plotX = [];
for k = 0:1
    X = [X x.^k];
    plotX = [plotX plotx.^k];
end
w = (X'*X)\X'*t;
```

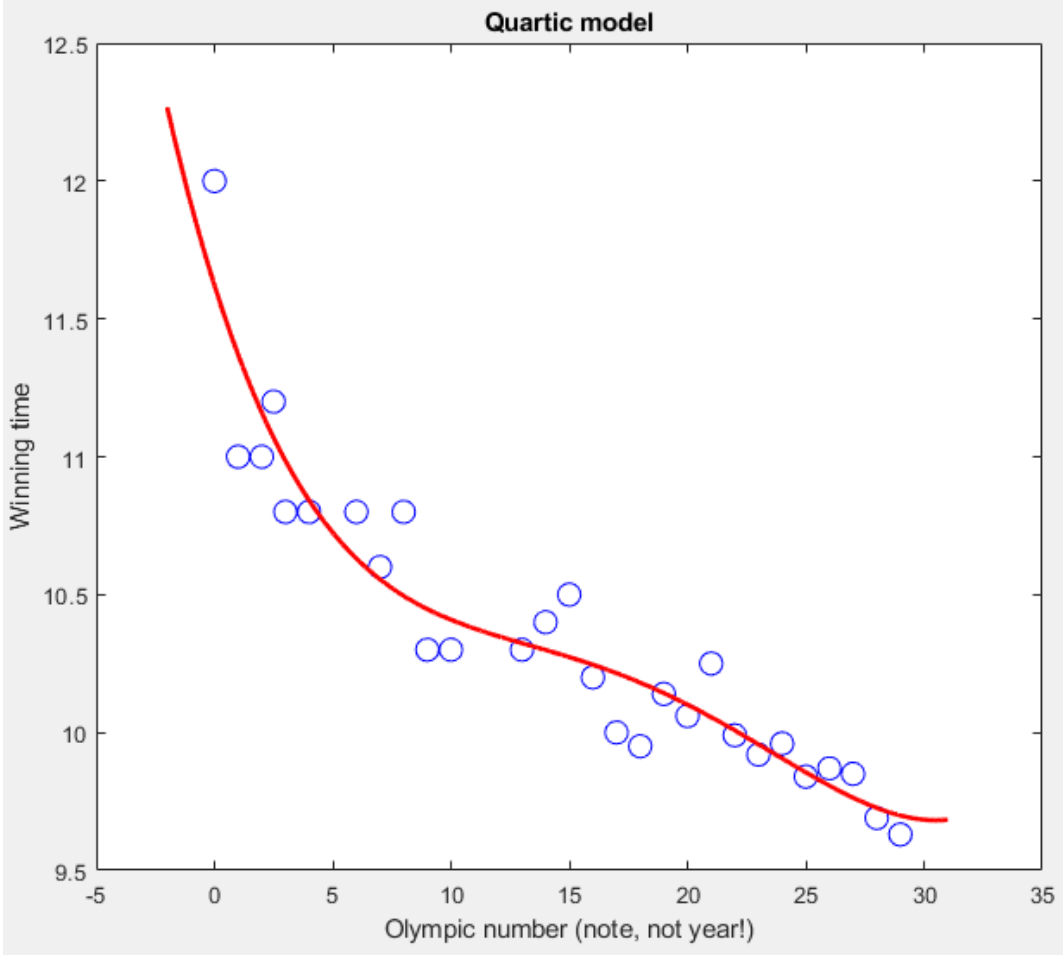


```
% Plot the model
figure(1);hold off
plot(x,t,'bo','markersize',10);
xlabel('Olympic number (note, not year!)');
ylabel('Winning time');
hold on
plot(plotx,plotX*w,'r','linewidth',2)
title('Linear Model')
pause(3)
 %% Quadratic model
plotx = [x(1)-2:0.01:x(end)+2]'; X = [];
plotX = [];
for k = 0:2
    X = [X x.^k];
    plotX = [plotX plotx.^k];
end
w = (X'*X)\X'*t;
% Plot the model
figure(1);hold off
plot(x,t,'bo','markersize',10);
xlabel('Olympic number (note, not year!)'); ylabel('Winning time');
hold on
plot(plotx,plotX*w,'r','linewidth',2)title('Quadratic model')
pause(3)
```

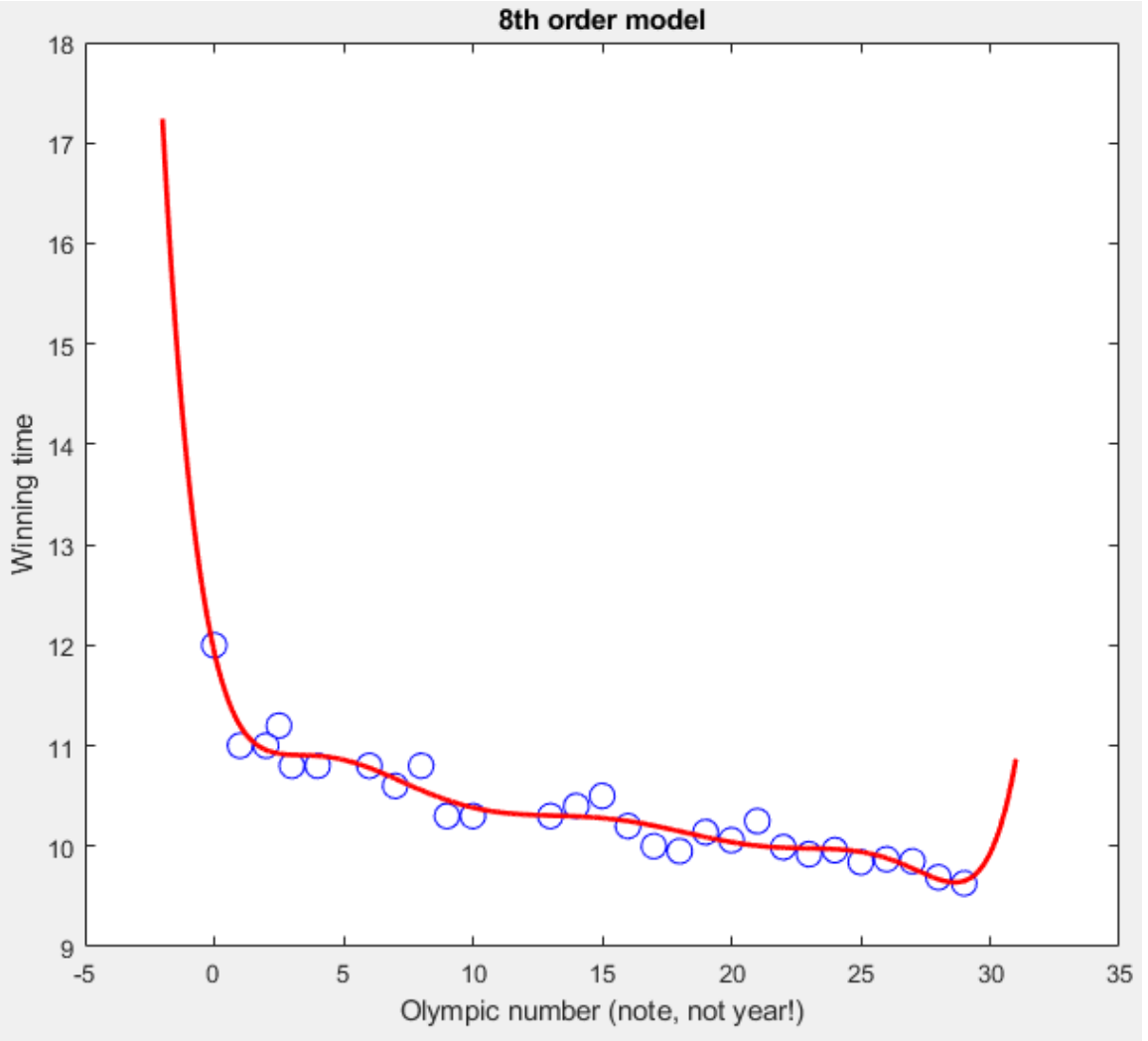




```
%% Quartic model
plotx = [x(1)-2:0.01:x(end)+2]';
X = [];
plotX = [];
for k = 0:4
    X = [X x.^k];
    plotX = [plotX plotx.^k];
end
w = (X'*X)\X'*t;
% Plot the model
figure(1);hold off
plot(x,t,'bo','markersize',10);
xlabel('Olympic number (note, not year!)');
ylabel('Winning time');
hold on
plot(plotx,plotX*w,'r','linewidth',2)
title('Quartic model')
pause(3)
```



```
%% 8th order model
plotx = [x(1)-2:0.01:x(end)+2]';
X = [];
plotX = [];
for k = 0:8
    X = [X x.^k];
    plotX = [plotX plotx.^k];
end
w = (X'*X)\X'*t;
% Plot the model
%figure(1);hold off
figure(1);hold off
plot(x,t,'bo','markersize',10);
xlabel('Olympic number (note, not year!)');
ylabel('Winning time');
hold on
plot(plotx,plotX*w,'r','linewidth',2)
title('8th order model')
```



Non-linear Response from a Linear Model

- We are free to define any set of K functions of x, $h_k(x)$:

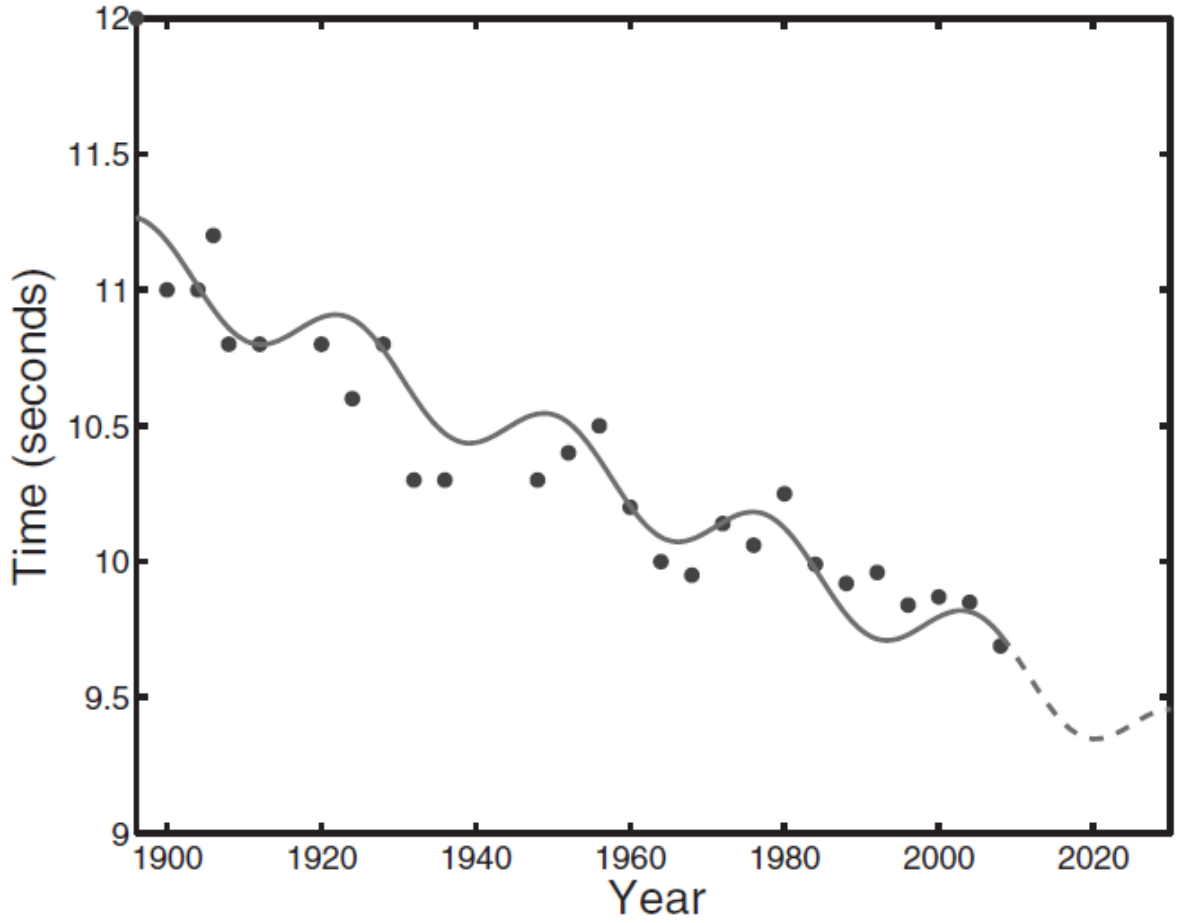
$$\mathbf{X} = \begin{bmatrix} h_1(x_1) & h_2(x_1) & \cdots & h_K(x_1) \\ h_1(x_2) & h_2(x_2) & \cdots & h_K(x_2) \\ \vdots & \vdots & \cdots & \vdots \\ h_1(x_N) & h_2(x_N) & \cdots & h_K(x_N) \end{bmatrix}$$

- A suitable set of functions for 100m data might be:

$$\begin{aligned} h_1(x) &= 1 \\ h_2(x) &= x \\ h_3(x) &= \sin\left(\frac{x-a}{b}\right) \\ f(x;\mathbf{w}) &= w_0 + w_1 x + w_2 \sin\left(\frac{x-a}{b}\right) \end{aligned}$$

Least squares fit of $f(x;\boldsymbol{\omega}) = \omega_0 + \omega_1 x + \omega_2 \sin(\frac{x-a}{b})$ to 100m data

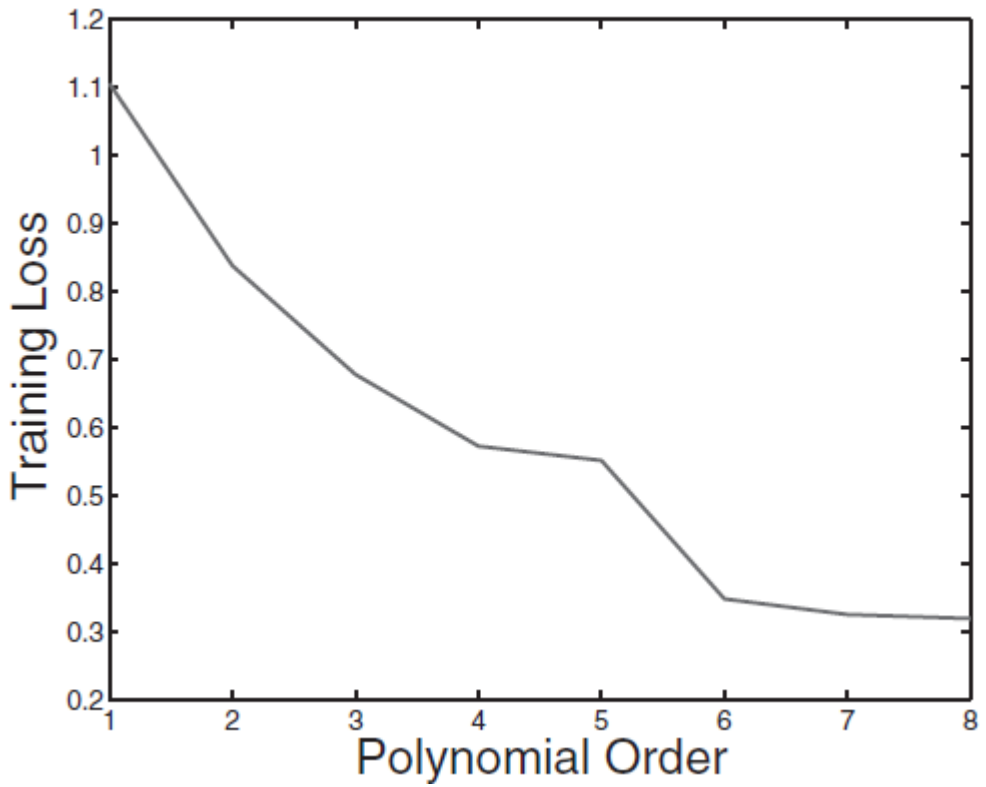
- There appears to be a slight periodic trend in the 100m data
- Assume $a = 2660$ and $b = 4.3$ are fixed
- $\omega_0 = 36.610, \omega_1 = -0.013, \omega_2 = -0.133, L = 1.1037$

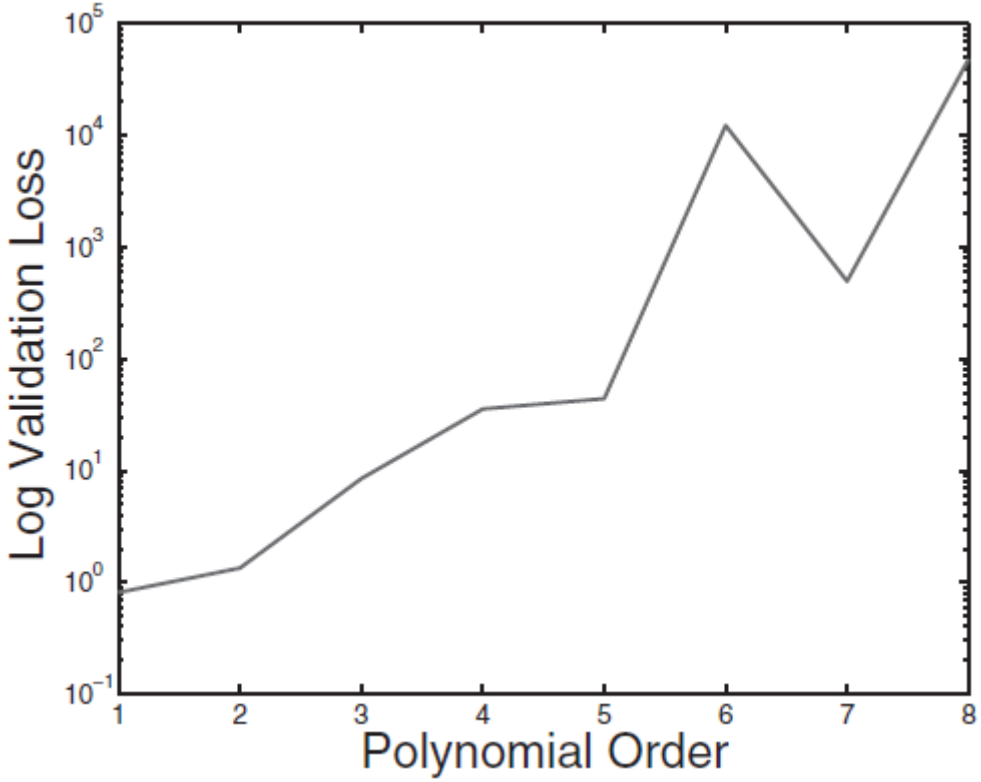


Generalization and Over-fitting

- In 100m data, the 8th-order polynomial model pays too much attention to the training data (it overfits) and does not generalize well to new data.
- Determining the optimal model complexity such that it is able to generalize well without over-fitting is very challenging.
- This tradeoff is often referred to as the bias-variance tradeoff.
- One common way to overcome this problem is to use a second dataset, referred to as a validation set. It is used to validate the predictive performance of our model.
- For example, in 100m data, we could remove all Olympics since 1980 from the training set and make these the validation set.

Validation data



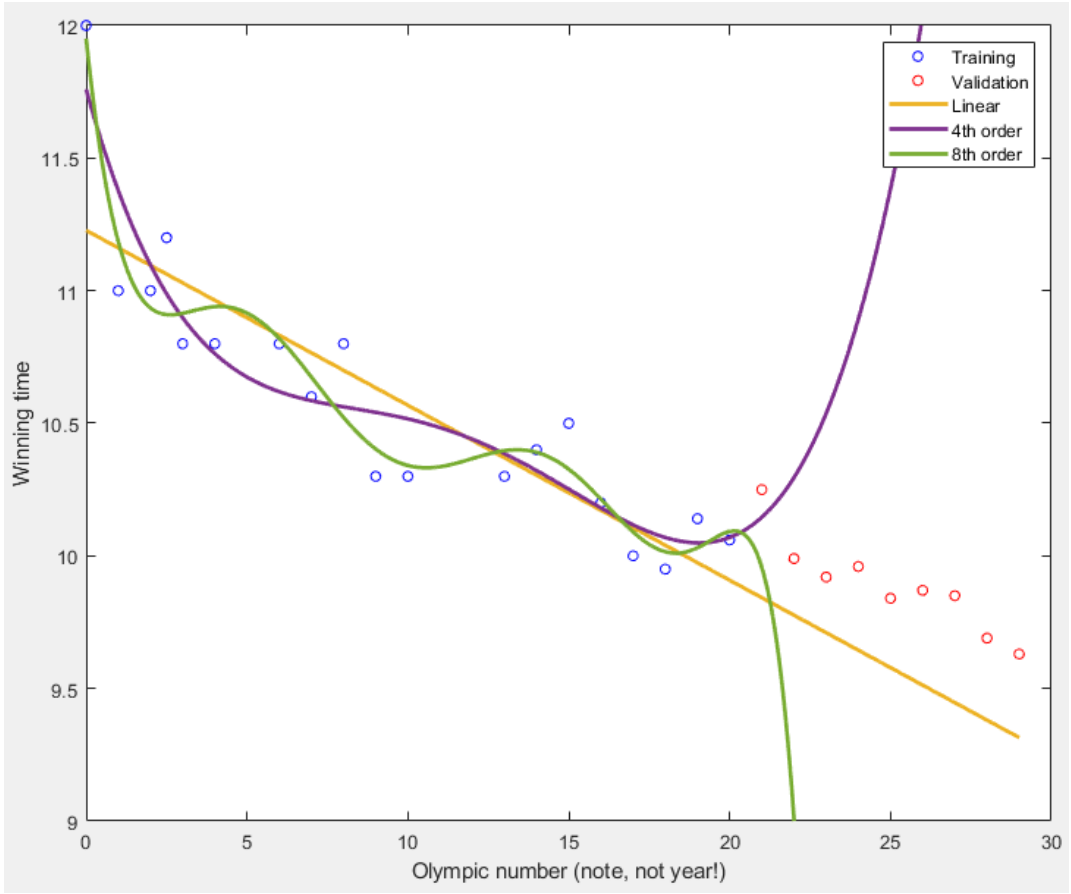


- Left: Training loss for the Olympic men's 100m data (before 1980)
- Right: Log validation loss for the Olympic men's 100m data. When using the squared predictive error (squared loss) and measures how close the predicted values are to the true values. Note that the log loss is plotted as the value increases so rapidly.

MATLAB script: olympval.m

```
%% olympval.m From A First Course in Machine Learning, Chapter 1.
clc;clear all;close all;
%% Load the Olympic data and extract the training and validation data
load ../data/olympics/male100.csv
x = male100(:,1); % Olympic years
t = male100(:,2); % Winning times
pos = find(x>1979);
% Rescale x for numerical reasons
x = x - x(1); x = x./4;
valx = x(pos:end); valt = t(pos:end);
x(pos:end) = []; t(pos:end) = [];
%% Fit the different models and plot the results
orders = [1 4 8]; %We shall fit models of these orders
% Plot the data
figure(1);hold off
plot(x,t,'bo','markersize',5); hold all
plot(valx,valt,'ro','markersize',5);
plotx = [min(x):0.01:max(valx)]';
```

```
for i = 1:length(orders)
    X = [];plotX = [];valX = [];
    for k = 0:orders(i)
        X = [X x.^k];
        valX = [valX valx.^k];
        plotX = [plotX plotx.^k];
    end
    % Compute w
    w = (X'*X)\X'*t;
    plot(plotx,plotX*w,'linewidth',2);
    % Compute validation loss
    val_loss(i) = mean((valX*w - valt).^2);
end
ylim([9 12]);
legend('Training','Validation','Linear','4th order','8th order')
%% Display the validation losses
for i = 1:length(orders)
 fprintf('\n Model order: %g, Validation loss: %g',...
        orders(i),val_loss(i));
end
xlabel('Olympic number (note, not year!)');
ylabel('Winning time');
```



Cross-Validation

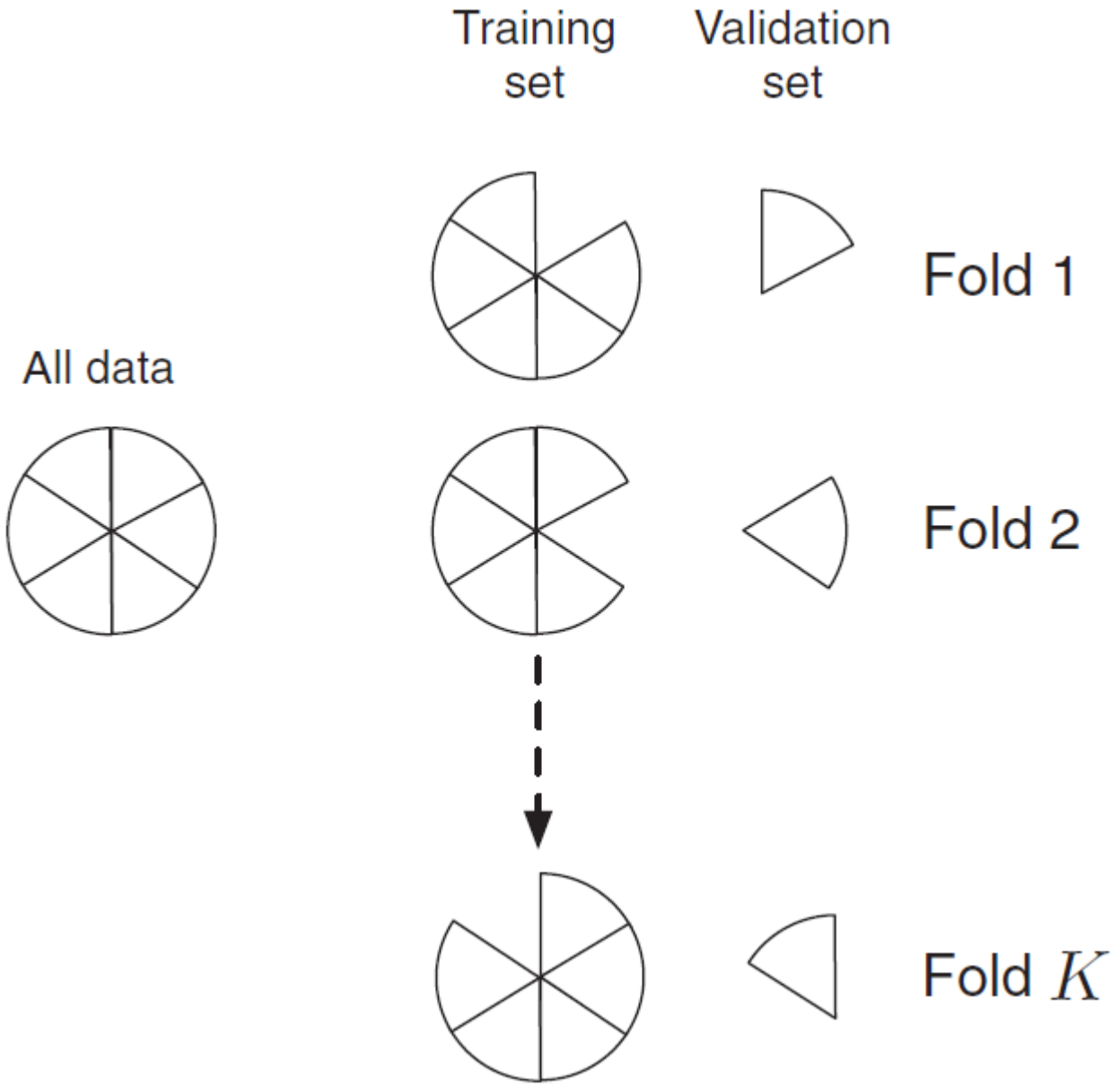
- The loss from validation data will be sensitive to the choice of data in our validation set. This is particularly problematic if our dataset (and hence our validation set) is small.

- Cross-validation is a technique that allows us to make more efficient use of the data we have.

- K-fold cross-validation splits the data into K equally sized blocks. Each block takes its turn as a validation set for a training set comprised of the other K-1 blocks.

- Averaging over the resulting K loss values gives us our final loss value.

Cross-Validation



Leave-One-Out Cross-Validation (LOOCV)

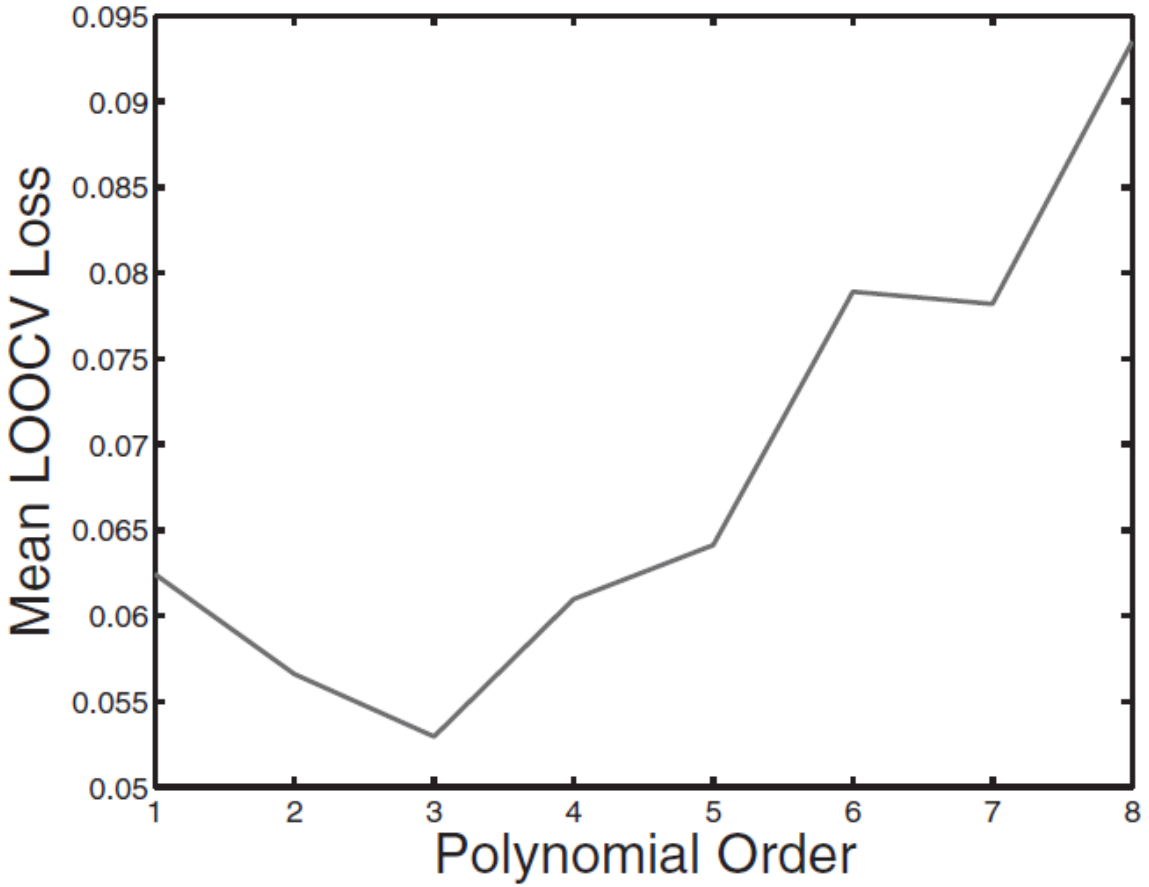
- An extreme case of K-fold cross-validation is where K = N: each data observation is held out in turn and used to test a model trained on the other N-1 objects

- The average squared validation loss for LOOCV is

$$L^{CV} = \frac{1}{N}\sum_{n=1}^{N}(t_n - \widehat{\omega}_{\_n}^{T}x_n)^2$$

where $\widehat{\omega}_{\_n}$ is the estimated parameters without the nth training example.

The mean LOOCV error for the Olympic men's 100m



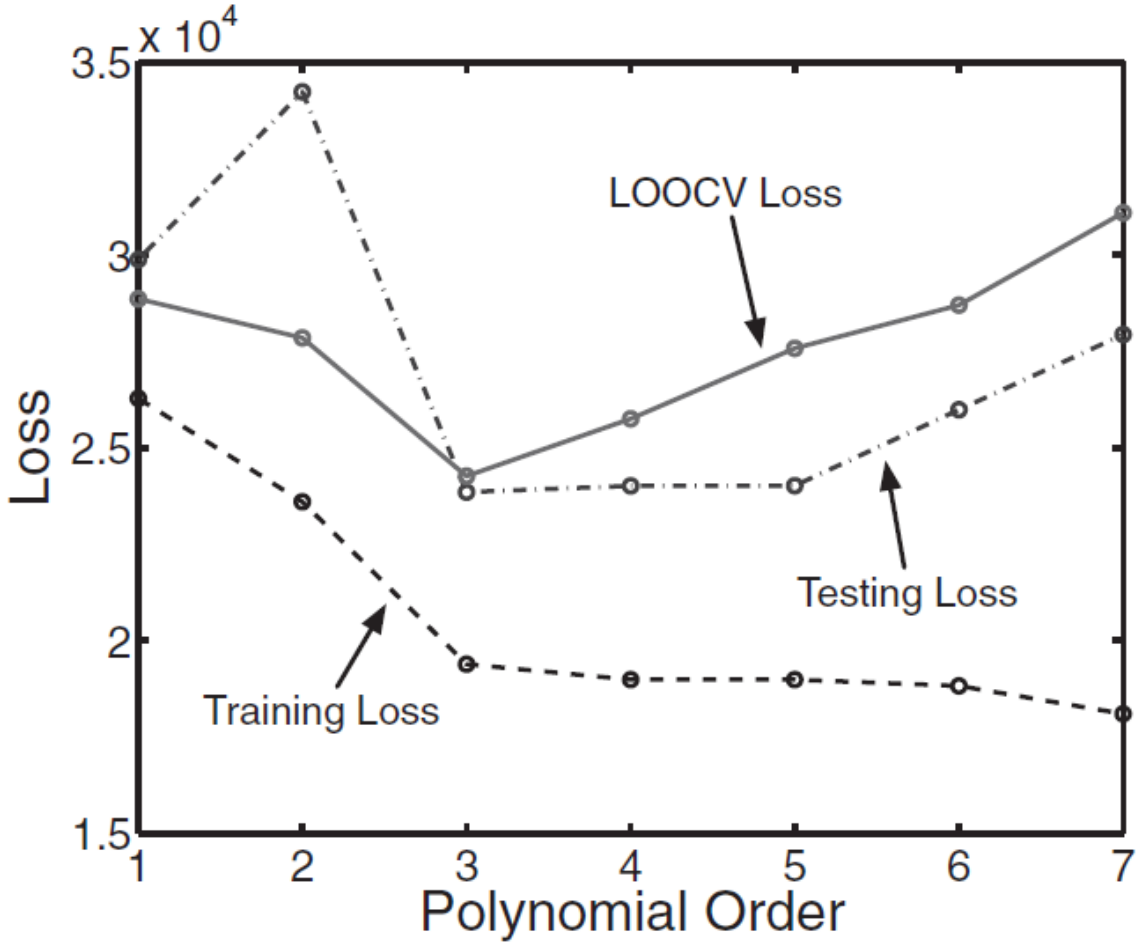
This plot suggests that a third-order polynomial would be best and disagrees with the value obtained from using the last few data points as a validation set. However, the two methods do agree on one thing- the model certainly shouldn't be sixth-order or above.

Cross-Validation: A Synthetic dataset

- 50 input target pairs were generated from a noisy 3rd-order polynomial function.
- A further 1000 input-target pairs were generated from the true function and are used as an independent test set to compute an additional, independent loss.
- The LOOCV loss and the test loss decrease as the order is increased to 3 and then increase as the order is increased further.
- Unfortunately, we will rarely be able to call upon 1000 independent points from outside our training set and will heavily rely on a cross-validation scheme.

Cross-Validation: A Synthetic dataset

The training, testing and leave-one-out loss curves obtained for a noisy cubic function where a sample size of 50 is available for training and LOOCV estimation. The test error is computed using 1000 independent samples.

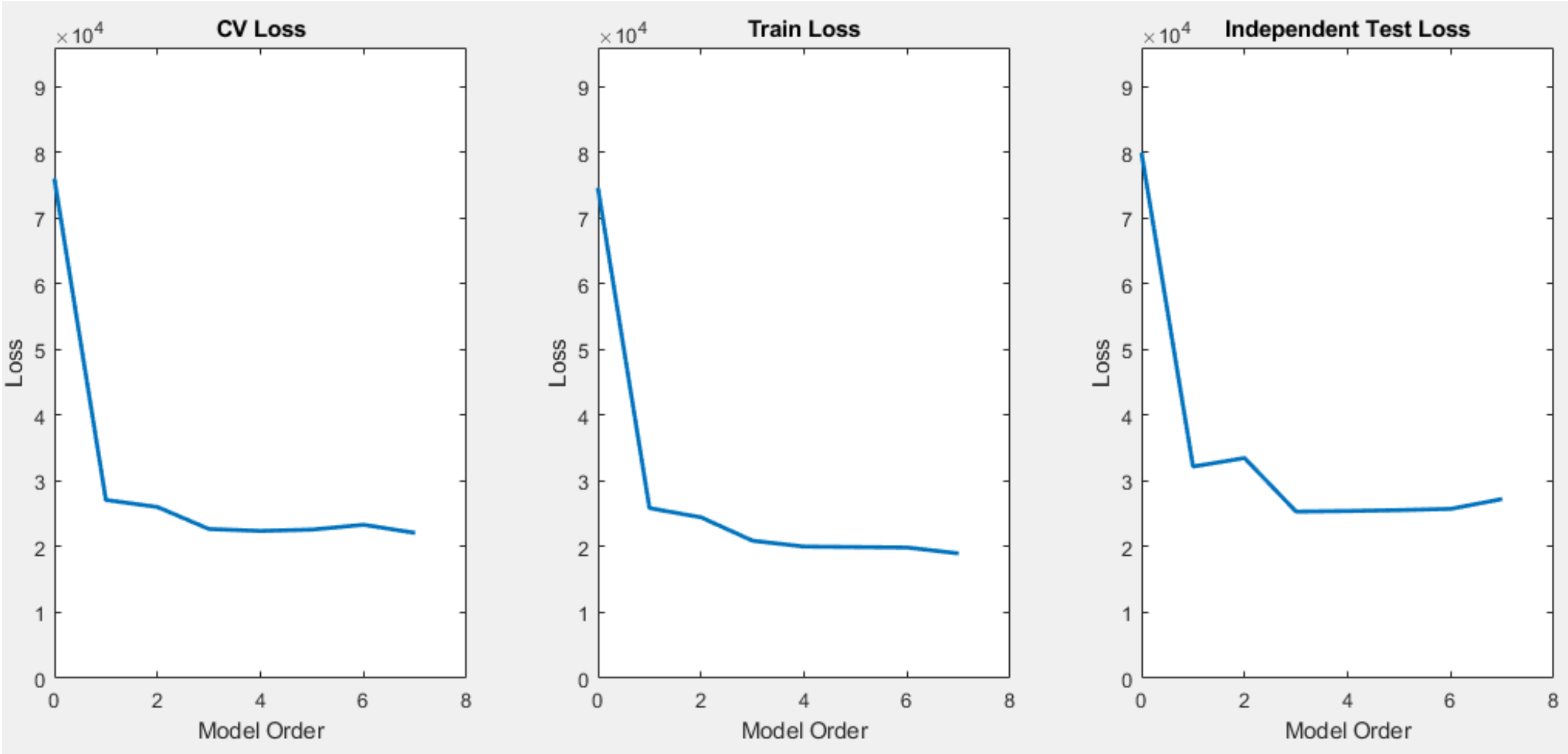


MATLAB script: cv_demo.m

```
%% cv_demo.m From A First Course in Machine Learning, Chapter 1.
clc;clear all;close all;
rng(1);
%% Generate some data
% Generate x between -5 and 5
N = 100;  x = 10*rand(N,1) - 5;
t = 5*x.^3  - x.^2 + x + 150*randn(size(x));
testx = [-5:0.01:5]'; % Large, independent test set
testt = 5*testx.^3 - testx.^2 + testx + 150*randn(size(testx));

%% Run a cross-validation over model orders
maxorder = 7;
X = [];
testX = [];
K = 9 %K-fold CV
sizes = repmat(floor(N/K),1,K);
sizes(end) = sizes(end) + N - sum(sizes);
csizes = [0 cumsum(sizes)];
```

```
% Note that it is often sensible to permute the data objects before
% performing CV.  It is not necessary here as x was created randomly.
% If it were necessary, the following code would work:
% order = randperm(N); x = x(order); Or: X = X(order,:) if it is multi-dimensional.
% t = t(order);
for k = 0:maxorder
    X = [X x.^k]; testX = [testX testx.^k];
    for fold = 1:K
        % Partition the data foldX contains the data for just one fold
        % trainX contains all other data

        foldX = X(csizes(fold)+1:csizes(fold+1),:);
        trainX = X; trainX(csizes(fold)+1:csizes(fold+1),:) = [];
        foldt = t(csizes(fold)+1:csizes(fold+1));
        traint = t; traint(csizes(fold)+1:csizes(fold+1)) = [];

        w = (trainX'*trainX)\trainX'*traint; fold_pred = foldX*w;
        cv_loss(fold,k+1) = mean((fold_pred-foldt).^2);
        ind_pred = testX*w;
        ind_loss(fold,k+1) = mean((ind_pred - testt).^2);
        train_pred = trainX*w;
        train_loss(fold,k+1) = mean((train_pred - traint).^2);
    end
end
```

```
%% Plot the results
figure(1);
subplot(131)
plot(0:maxorder,mean(cv_loss,1),'linewidth',2)
xlabel('Model Order');
ylabel('Loss');
title('CV Loss');
ylim(1.2*[0,max(max(max(mean(cv_loss,1),mean(train_loss,1)),mean(ind_loss,1)))])
subplot(132)
plot(0:maxorder,mean(train_loss,1),'linewidth',2)
xlabel('Model Order');
ylabel('Loss');
title('Train Loss');
ylim(1.2*[0,max(max(max(mean(cv_loss,1),mean(train_loss,1)),mean(ind_loss,1)))])
subplot(133)
plot(0:maxorder,mean(ind_loss,1),'linewidth',2)
xlabel('Model Order');
ylabel('Loss');
title('Independent Test Loss')
ylim(1.2*[0,max(max(max(mean(cv_loss,1),mean(train_loss,1)),mean(ind_loss,1)))])
```


CV Loss
Train Loss
Independent Test Loss
×10^4
Loss
Model Order
0
1
2
3
4
5
6
7
8
9
0
2
4
6
8

Computational Scaling of K-fold Cross-Validation

- To implement LOOCV, we need to train our model N times, which will take roughly N times longer than training it once on all of the data

- In 10-fold cross-validation, we leave out 10% of the data for validation and use the remaining 90% for training.
    - This reduces the number of training loops from N to 10: a considerable saving if N >> 10.

- A popular choice is to use N-fold cross-validation, allowing averages to be taken across both folds and repetitions.

Regularized Least Squares

- Rather than just minimizing the average squared loss L, we could minimize a regularized loss L' made by adding previous loss and a term penalizing overcomplexity:

$$L' = L + \lambda \boldsymbol{\omega}^T \boldsymbol{\omega} = \frac{1}{N} (\boldsymbol{X}\boldsymbol{\omega} - \boldsymbol{t})^T (\boldsymbol{X}\boldsymbol{\omega} - \boldsymbol{t}) + \lambda \boldsymbol{\omega}^T \boldsymbol{\omega}$$

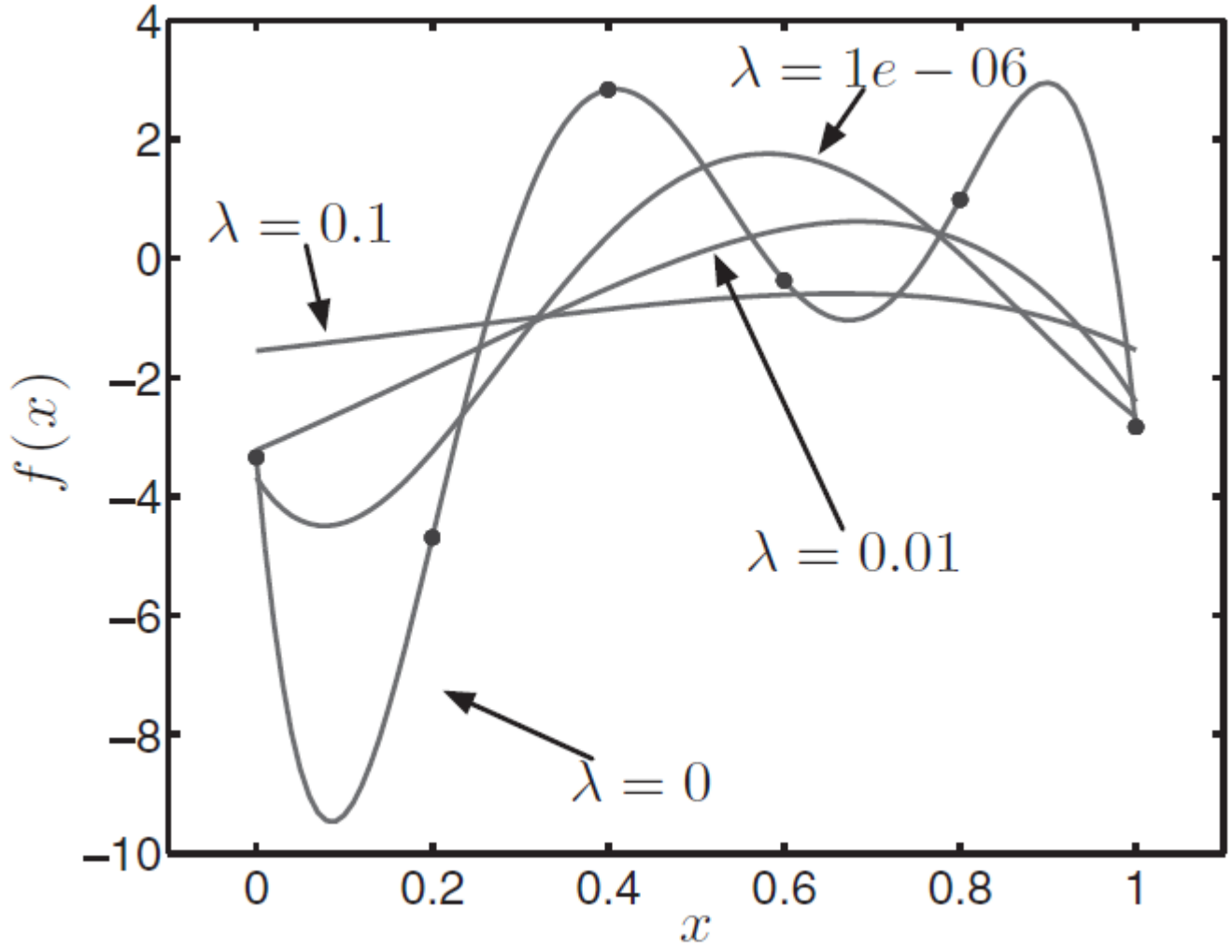
$$= \frac{1}{N} \boldsymbol{\omega}^T \boldsymbol{X}^T \boldsymbol{X} \boldsymbol{\omega} - \frac{2}{N} \boldsymbol{\omega}^T \boldsymbol{X}^T \boldsymbol{t} + \frac{1}{N} \boldsymbol{t}^T \boldsymbol{t} + \lambda \boldsymbol{\omega}^T \boldsymbol{\omega}$$

- $\Rightarrow \quad \frac{\partial L'}{\partial \boldsymbol{\omega}} = \frac{2}{N} \boldsymbol{X}^T \boldsymbol{X} \boldsymbol{\omega} - \frac{2}{N} \boldsymbol{X}^T \boldsymbol{t} + 2\lambda \boldsymbol{\omega} = \boldsymbol{0}$

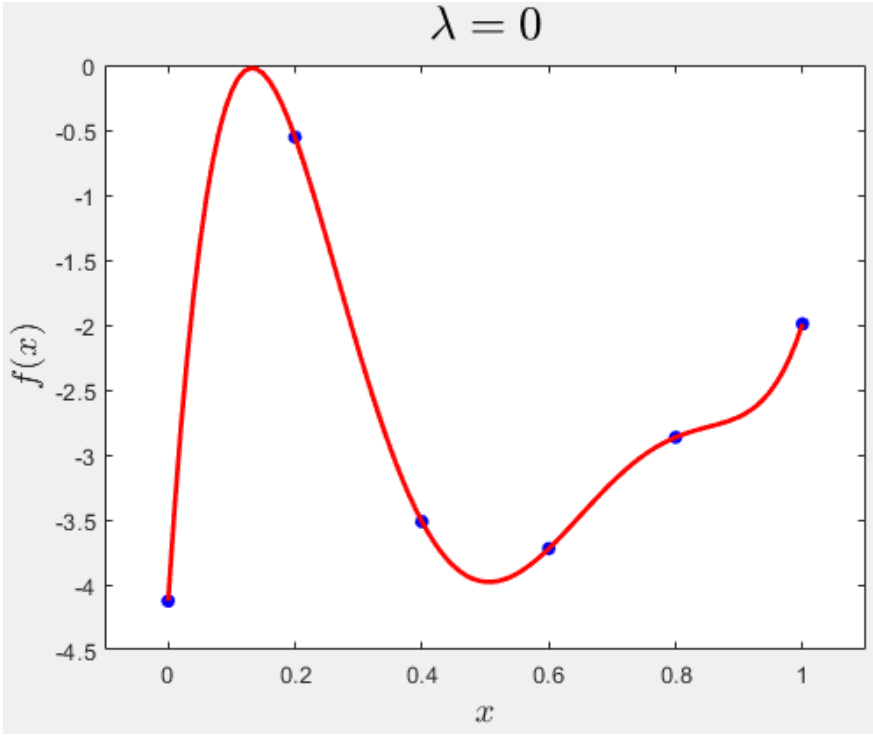
$\Rightarrow \quad (\boldsymbol{X}^T \boldsymbol{X} + N\lambda \mathbf{I}) \boldsymbol{\omega} = \boldsymbol{X}^T \boldsymbol{t}$

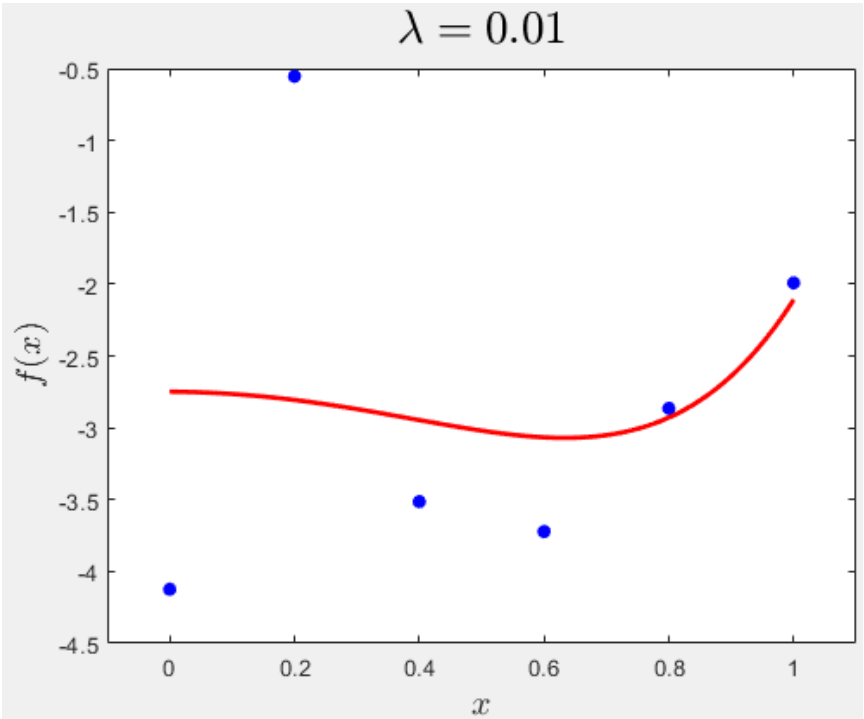
$\Rightarrow \quad \hat{\boldsymbol{\omega}} = (\boldsymbol{X}^T \boldsymbol{X} + N\lambda \mathbf{I})^{-1} \boldsymbol{X}^T \boldsymbol{t}$

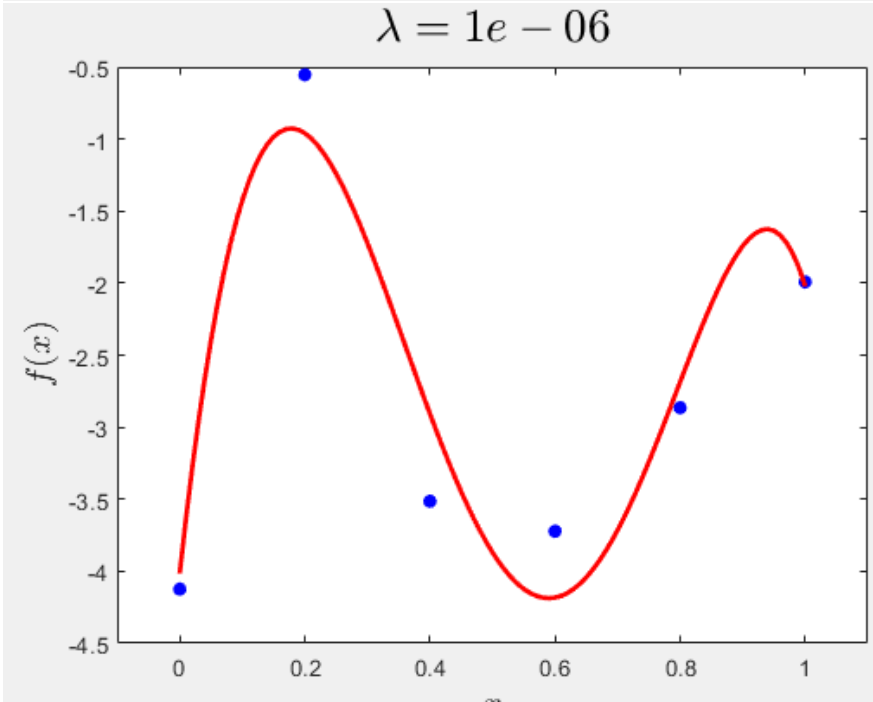
Varying the regularization parameter for a 5th-order polynomial

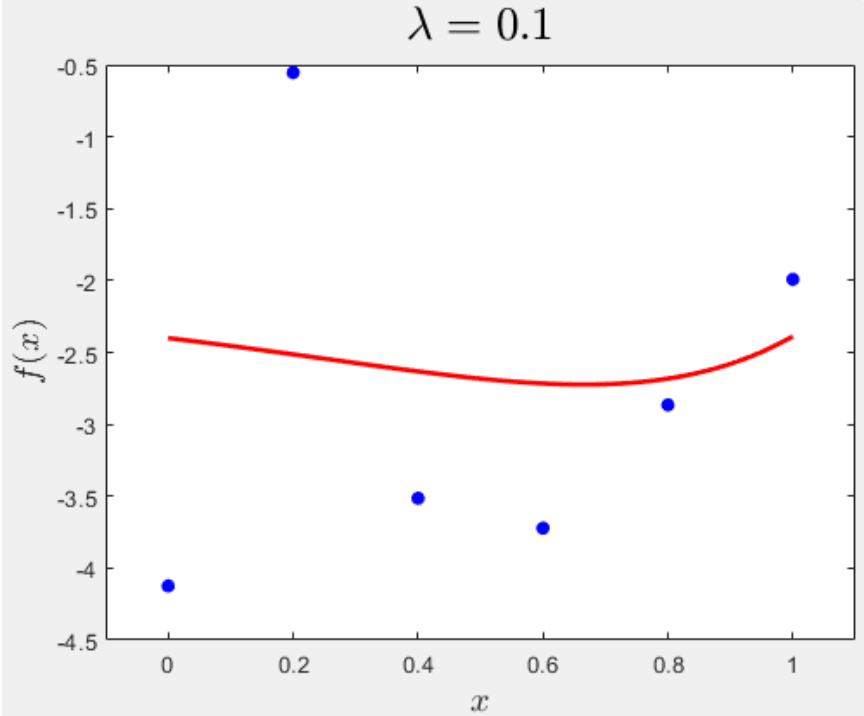


We can use cross-validation to choose the λ to give the best predictive performance


$\lambda = 0$
$f(x)$
x


$\lambda = 0.01$
$f(x)$
x


$\lambda = 1e-06$
$f(x)$
x

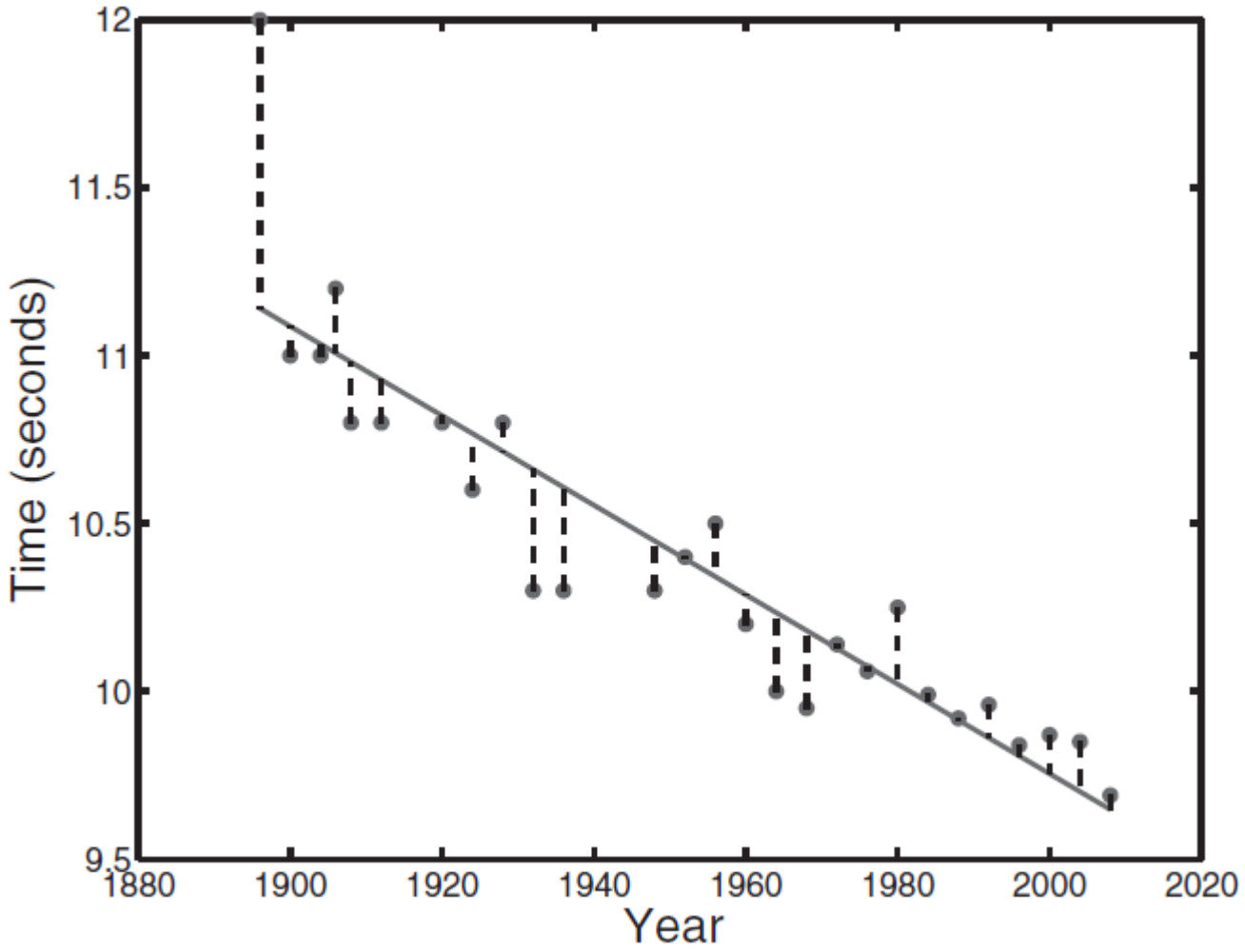

$\lambda = 0.1$
$f(x)$
x

Linear Modelling: A Maximum Likelihood Approach

Chap2. A First Course in Machine Learning, 2ed, Simon Rogers and Mark Girolami, 2017

Consider the Modelling Problem as a Generative One

- Linear fit to the Olympic men's 100 m data with errors highlighted.



- Model takes the form: $t_n = \boldsymbol{\omega}^T \boldsymbol{x}_n + \epsilon_n$ (a continuous random variable)

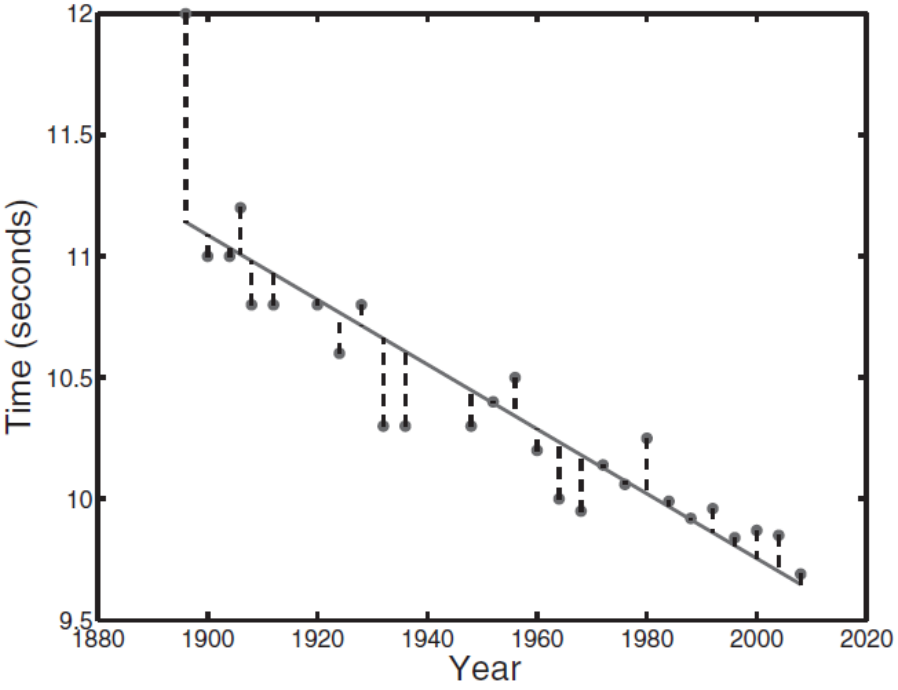
Assume ϵ_n's are independent Gaussian random variable

- Assume $\epsilon_1, \cdots, \epsilon_n$ are independent Gaussian random variable

$$p(\epsilon_1, \cdots, \epsilon_n) = \prod_{n=1}^{N} p(\epsilon_n), \qquad p(\epsilon_n) = N(\mu, \sigma^2)$$

- $t_n = \boldsymbol{\omega}^T \boldsymbol{x}_n + \epsilon_n \Rightarrow$ Deterministic $\boldsymbol{\omega}^T \boldsymbol{x}_n$ referred to as as trend or drift.

$\Rightarrow \epsilon_n$ referred to as noise

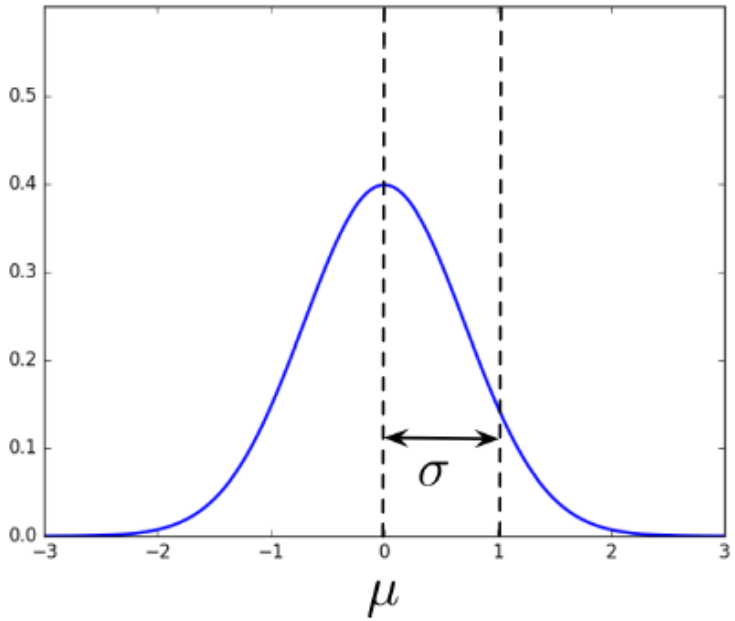


Gaussian Distribution

Gaussian Distribution

- Recall the Gaussian, or normal, distribution:

$$N(x;\mu,\sigma^2) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

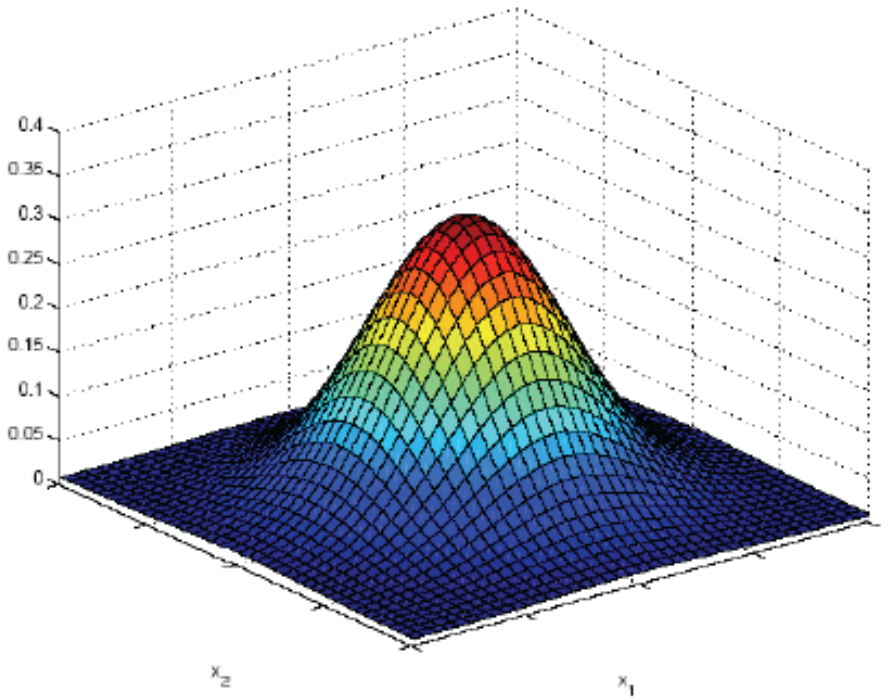


- In machine learning, we use Gaussians a lot because they make the calculations easy.

Multivariate Gaussian Distribution

- Multivariate Gaussian Distribution $\boldsymbol{x} \sim N(\mu, \sum)$, or $N(\boldsymbol{x}; \mu, \sum)$ is

$$p(\boldsymbol{x}) = \frac{1}{(2\pi)^{d/2}|\sum|^{1/2}} \exp[-\frac{1}{2}\ (\boldsymbol{x}-\mu)^T \sum^{-1} (\boldsymbol{x}-\mu)]$$



Multivariate parameters

- Mean: $E[\boldsymbol{x}] = [\mu_1, \cdots, \mu_d]^T$
- Covariance

$$\sum = \mathrm{Cov}(\boldsymbol{x}) = E[(\boldsymbol{x} - \mu)(\boldsymbol{x} - \mu)^T] = \begin{bmatrix} \sigma_1^2 & \sigma_{12} & \cdots & \sigma_{1d} \\ \sigma_{21} & \sigma_2^2 & \cdots & \sigma_{2d} \\ \vdots & \vdots & \ddots & \vdots \\ \sigma_{d1} & \sigma_{d2} & \cdots & \sigma_d^2 \end{bmatrix}$$

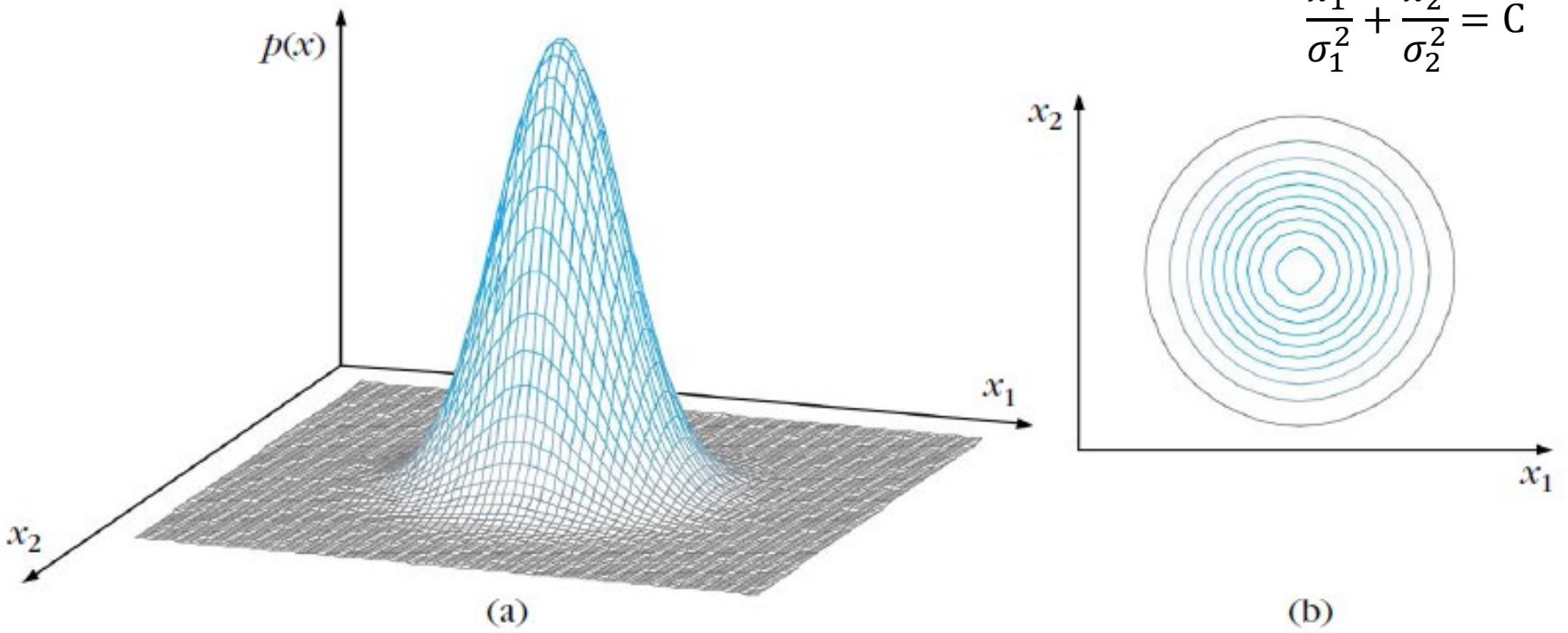
- For Gaussians - all you need to know is mean and covariance

2D Gaussian pdf, diagonal $\sum$ with $\sigma_1^2 = \sigma_2^2$

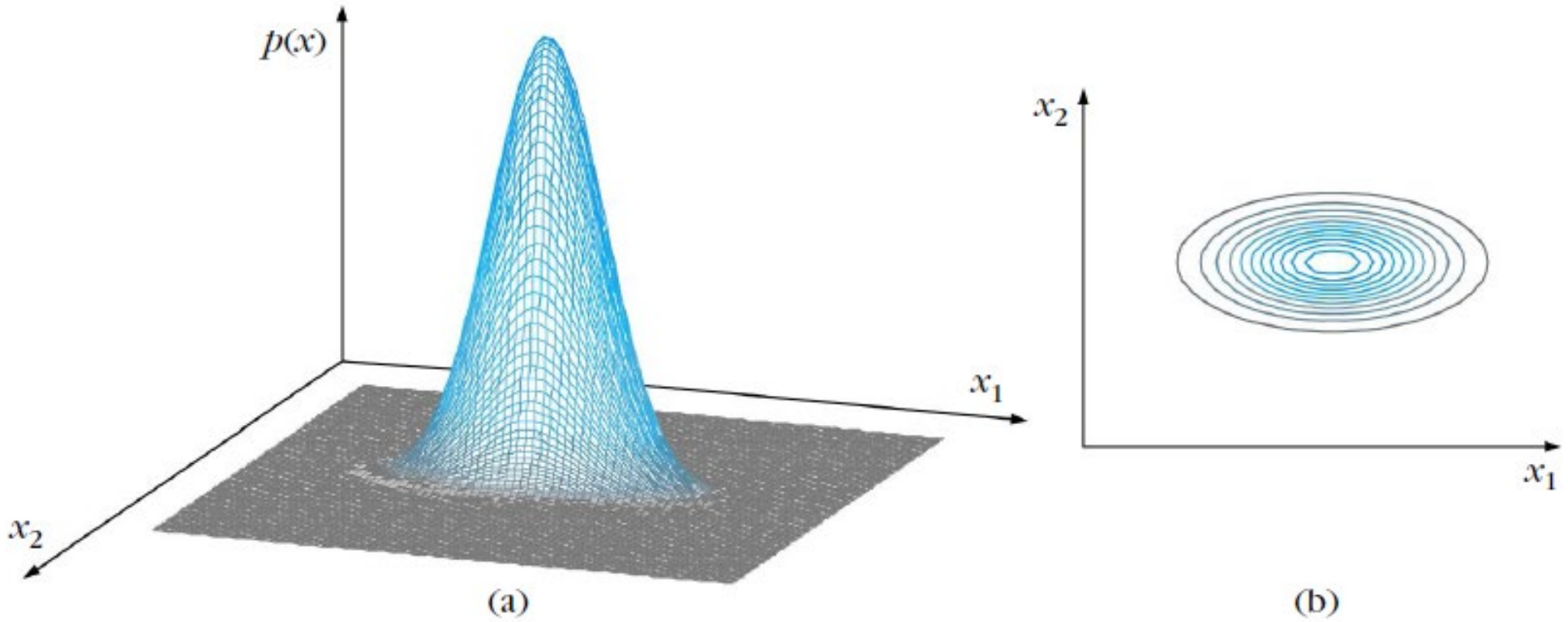
$$\Sigma = \begin{bmatrix} 3 & 0 \\ 0 & 3 \end{bmatrix}$$

$$(\boldsymbol{x} - \mu)^T \Sigma^{-1} (\boldsymbol{x} - \mu) = [x_1\ x_2] \begin{bmatrix} \frac{1}{\sigma_1^2} & 0 \\ 0 & \frac{1}{\sigma_2^2} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \mathrm{C}$$

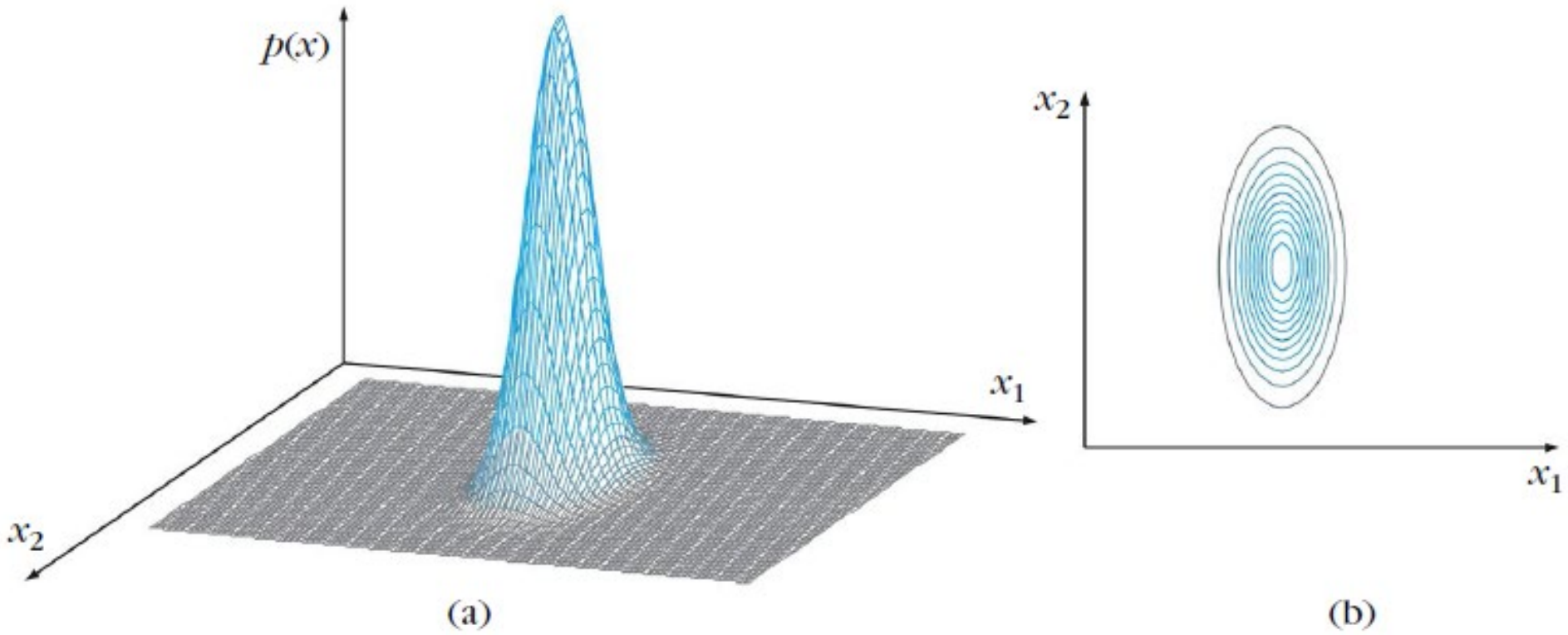
$$\frac{x_1^2}{\sigma_1^2} + \frac{x_2^2}{\sigma_2^2} = \mathrm{C}$$



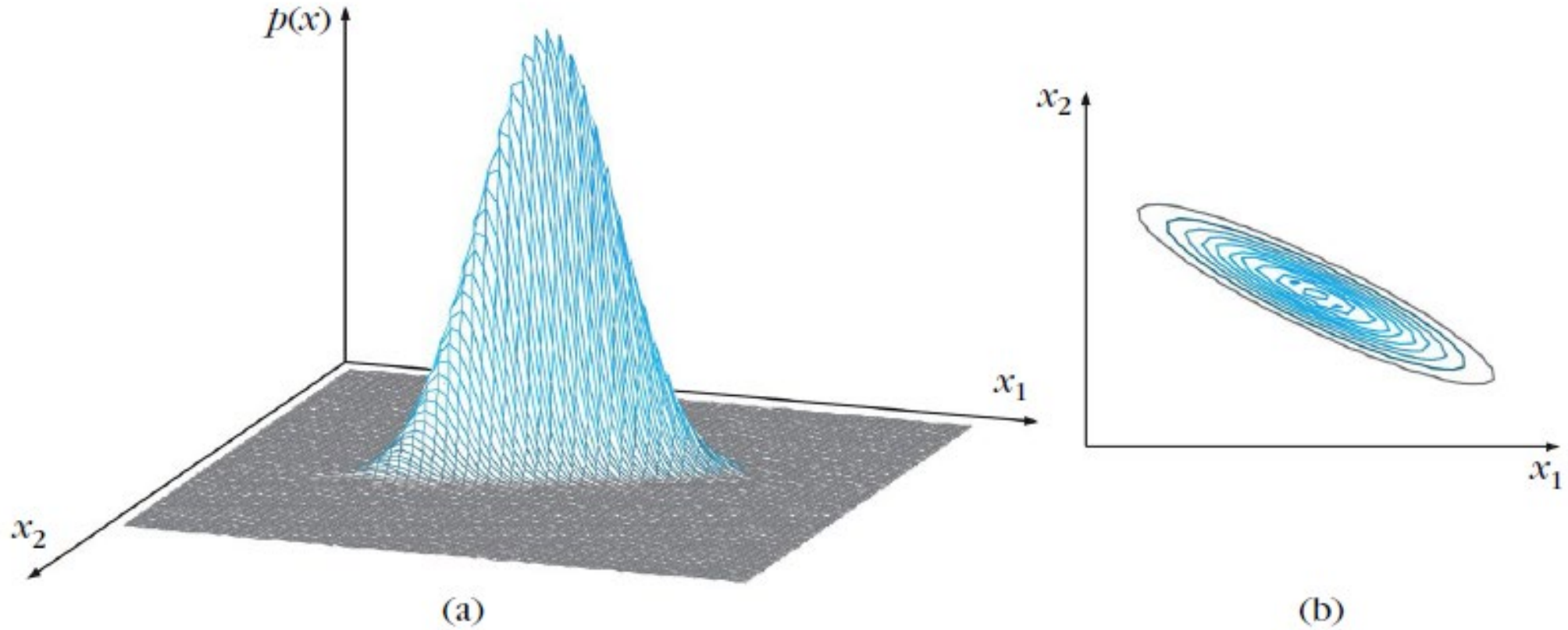
2D Gaussian pdf, diagonal $\sum$ with $\sigma_1^2 = 15 \gg \sigma_2^2 = 3$



2D Gaussian pdf, diagonal $\sum$ with $\sigma_1^2 = 3 \ll \sigma_2^2 = 15$



2D Gaussian pdf, non-diagonal ∑



Likelihood

- So far, the model is of the form $t_n = f(\boldsymbol{x}_n; \boldsymbol{\omega}) + \epsilon_n,\ \ \epsilon_n \sim N(0, \sigma^2)$

- t_n is now a random variable, no single value of t_n for a particular $\boldsymbol{x}_n$.
  - We cannot use the loss as a means of optimizing $\boldsymbol{\omega}$ and σ^2.

- The density function for $t_n = \boldsymbol{\omega}^T \boldsymbol{x}_n + \epsilon_n$ is $p(t_n | \boldsymbol{x}_n, \boldsymbol{\omega}) = N(\boldsymbol{\omega}^T \boldsymbol{x}_n, \sigma^2)$

- The density of t_n depends on particular values of $\boldsymbol{x}_n$ and $\boldsymbol{\omega}$ (they determine the mean) and σ^2 (the variance).

- How we can use the density of t_n to find the optimal values of $\boldsymbol{\omega}$ and σ^2 ?

Likelihood: Olympics Example

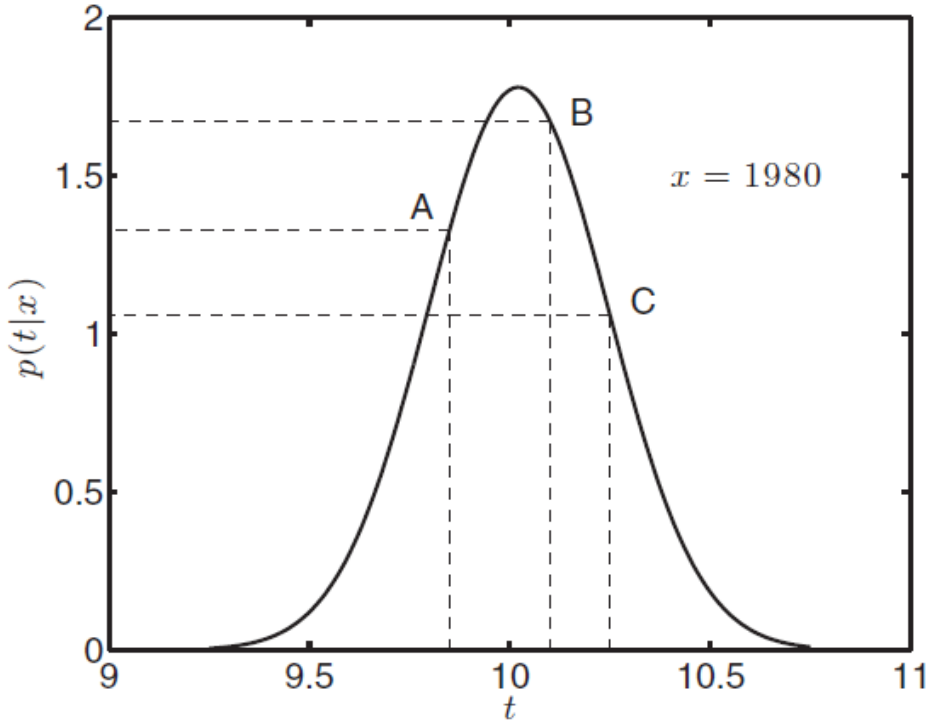
- Consider the year 1980 from Olympic dataset.
- Based on the model $\hat{\omega}_0 = 35.416$ and $\hat{\omega}_1 = -0.0133$ in the previous lecture and assume $\sigma^2 = 0.05$, we can plot $p(t_n|\boldsymbol{x}_n, \boldsymbol{\omega})$ w.r.t t_n

$$p\left(t_n|\boldsymbol{x}_n = \begin{bmatrix} 1 \\ 1980 \end{bmatrix}, \boldsymbol{\omega} = \begin{bmatrix} 35.416 \\ -0.0133 \end{bmatrix}, \sigma^2 = 0.05\right) = N(\boldsymbol{\omega}^T \boldsymbol{x}_n, \sigma^2)$$
$$= N(35.416 \times 1 - 0.0133 \times 1980 = 10.02, 0.05)$$

1. Right plot is likelihood function for the year 1980:
   - The most likely winning time is 10.02 seconds,
     Likelihood: mean > B > A > C

2. The 1980 actual winning time is C (t = 10.25)

3. We can change $\boldsymbol{\omega}$ and σ^2 to try and move the density so as to make the likelihood $p(t_n|\boldsymbol{x}_n, \boldsymbol{\omega})$ as high as possible at t = 10.25.



Dataset likelihood

- In general, we are not interested in the likelihood of a single data point but that of all of the data.

- Given *N* training samples, $\boldsymbol{x_1}, \boldsymbol{x_2}, \ldots, \boldsymbol{x_N}$, and note $t_n = \boldsymbol{\omega}^T \boldsymbol{x}_n + \epsilon_n$ is conditionally independent, i.e. given $\boldsymbol{\omega}$, the t_n's are independent, since we assume the noise at each data point is independent

$$p(\epsilon_1, \cdots, \epsilon_n) = \prod_{n=1}^{N} p(\epsilon_n), \epsilon_n \sim N(0, \sigma^2)$$

$$\Rightarrow L = p(t_1, \cdots, t_n | \boldsymbol{x}_n, \boldsymbol{\omega}, \sigma^2) = \prod_{n=1}^{N} p(t_n | \boldsymbol{x}_n, \boldsymbol{\omega}, \sigma^2) = \prod_{n=1}^{N} N(\boldsymbol{\omega}^T \boldsymbol{x}_n, \sigma^2)$$

Maximum Likelihood

- Using the monotonicity of log, we define the *log-likelihood function*

$$l(\boldsymbol{\omega}, \sigma^2) \equiv \log L = \log \prod_{n=1}^{N} p(t_n | \boldsymbol{x}_n, \boldsymbol{\omega}, \sigma^2) = \sum_{n=1}^{N} \log N(\boldsymbol{\omega}^T \boldsymbol{x}_n, \sigma^2)$$

$$= \sum_{n=1}^{N} \log \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(t_n - \boldsymbol{\omega}^T \boldsymbol{x}_n)^2}{2\sigma^2}}$$

$$= -\frac{N}{2}(\log(2\pi) + \log \sigma^2) - \frac{1}{2\sigma^2} \sum_{n=1}^{N} (t_n - \boldsymbol{\omega}^T \boldsymbol{x}_n)^2$$

Maximum Likelihood

- Define $\boldsymbol{\omega} = \begin{bmatrix} \omega_0 \\ \omega_1 \end{bmatrix}, \boldsymbol{x}_n = \begin{bmatrix} 1 \\ x_n \end{bmatrix} \Rightarrow \omega_0 + \omega_1 x_n = \boldsymbol{\omega}^T \boldsymbol{x}_n = \boldsymbol{x}_n{}^T \boldsymbol{\omega}$

- Define $\boldsymbol{X} = \begin{bmatrix} \boldsymbol{x}_1^T \\ \boldsymbol{x}_2^T \\ \vdots \\ \boldsymbol{x}_N^T \end{bmatrix} = \begin{bmatrix} 1 & x_1 \\ 1 & x_2 \\ \vdots & \vdots \\ 1 & x_N \end{bmatrix}, \boldsymbol{t} = \begin{bmatrix} t_1 \\ t_2 \\ \vdots \\ t_N \end{bmatrix} \Rightarrow \boldsymbol{X}\boldsymbol{\omega} = \begin{bmatrix} \omega_0 + \omega_1 x_1 \\ \omega_0 + \omega_1 x_2 \\ \vdots \\ \omega_0 + \omega_1 x_N \end{bmatrix}$

- Therefore

$$l(\boldsymbol{\omega}, \sigma^2) = -\frac{N}{2}(\log(2\pi) + \log\sigma^2) - \frac{1}{2\sigma^2}\sum_{n=1}^{N}(t_n - \boldsymbol{\omega}^T\boldsymbol{x}_n)^2$$

$$= -\frac{N}{2}(\log(2\pi) + \log\sigma^2) - \frac{1}{2\sigma^2}(\boldsymbol{t} - \boldsymbol{X}\boldsymbol{\omega})^T(\boldsymbol{t} - \boldsymbol{X}\boldsymbol{\omega})$$

Maximum Likelihood

- $$\frac{\partial l(\boldsymbol{\omega},\sigma^2)}{\partial \boldsymbol{\omega}} = 0 - \frac{1}{2\sigma^2}\frac{\partial}{\partial \boldsymbol{\omega}}\left((\boldsymbol{t}-\boldsymbol{X}\boldsymbol{\omega})^T(\boldsymbol{t}-\boldsymbol{X}\boldsymbol{\omega})\right)$$
$$= -\frac{1}{2\sigma^2}\frac{\partial}{\partial \boldsymbol{\omega}}(\boldsymbol{\omega}^T\boldsymbol{X}^T\boldsymbol{X}\boldsymbol{\omega} - 2\boldsymbol{\omega}^T\boldsymbol{X}^T\boldsymbol{t} + \boldsymbol{t}^T\boldsymbol{t}) = 0$$

$$\Rightarrow \quad \frac{\partial l(\boldsymbol{\omega},\sigma^2)}{\partial \boldsymbol{\omega}} = 2\boldsymbol{X}^T\boldsymbol{X}\boldsymbol{\omega} - 2\boldsymbol{X}^T\boldsymbol{t} = \boldsymbol{0}$$

$$\Rightarrow \quad \hat{\boldsymbol{\omega}} = (\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{t}$$

This is the maximum likelihood solution for $\boldsymbol{\omega}$ which is exactly the same with the least square solution in the previous lecture

- Minimizing the squared loss is equivalent to the maximum likelihood solution if the noise is assumed to be Gaussian.

Maximum Likelihood

- $\frac{\partial l(\boldsymbol{\omega},\sigma^2)}{\partial \sigma^2} = -\frac{N}{2}\frac{1}{\sigma^2} + \frac{1}{2\sigma^4}(\boldsymbol{t}-\boldsymbol{X\omega})^T(\boldsymbol{t}-\boldsymbol{X\omega}) = 0$

$$\Rightarrow \hat{\sigma}^2 = \frac{1}{N}(\boldsymbol{t}-\boldsymbol{X}\hat{\boldsymbol{\omega}})^T(\boldsymbol{t}-\boldsymbol{X}\hat{\boldsymbol{\omega}}) \qquad \left(=\frac{1}{N}\sum_{n=1}^{N}(t_n - \boldsymbol{x}_n{}^T\hat{\boldsymbol{\omega}})^2\right)$$

$$= \frac{1}{N}(\boldsymbol{t}-\boldsymbol{X}(\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{t})^T(\boldsymbol{t}-\boldsymbol{X}(\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{t})$$

$$= \frac{1}{N}(\boldsymbol{t}^T-\boldsymbol{t}^T\boldsymbol{X}(\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T)(\boldsymbol{t}-\boldsymbol{X}(\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{t})$$

$$= \frac{1}{N}\big(\boldsymbol{t}^T\boldsymbol{t}-\boldsymbol{t}^T\boldsymbol{X}(\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{t}-\boldsymbol{t}^T\boldsymbol{X}(\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{t}+$$
$$\boldsymbol{t}^T\boldsymbol{X}(\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{X}(\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{t}\big)$$

$$= \frac{1}{N}\big(\boldsymbol{t}^T\boldsymbol{t}-2\boldsymbol{t}^T\boldsymbol{X}(\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{t}+\boldsymbol{t}^T\boldsymbol{X}(\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{t}\big)$$

$$= \frac{1}{N}\big(\boldsymbol{t}^T\boldsymbol{t}-\boldsymbol{t}^T\boldsymbol{X}(\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{t}\big)$$

$$\Rightarrow \hat{\sigma}^2 = \frac{1}{N}\big(\boldsymbol{t}^T\boldsymbol{t}-\boldsymbol{t}^T\boldsymbol{X}\hat{\boldsymbol{\omega}}\big) = \frac{1}{N}\sum_{n=1}^{N}(t_n - \boldsymbol{x}_n{}^T\hat{\boldsymbol{\omega}})^2$$

The optimal variance is simply the average squared error.

Maximum Likelihood

- Using the Olympic 100m data, the optimal parameter values (for a first-order polynomial) are

$$\hat{\boldsymbol{\omega}} = \begin{bmatrix} 35.416 \\ -0.0133 \end{bmatrix}, \hat{\sigma}^2 = 0.0503$$

- $\hat{\boldsymbol{\omega}}$ is the same as the least squares solution provided in the previous lecture (they are both computed using the same expression).

- $\hat{\sigma}^2$ tells us the variance of the assumed Gaussian noise is used to corrupt our data.

- We will see that modelling the noise in this way provides several benefits over loss minimization.

Characteristics of the Maximum Likelihood Solution

Comment 2.6 – Hessian matrix: A Hessian matrix is square matrix containing all of the second-order partial derivatives of a function. For example, the Hessian matrix for a function $f(\mathbf{x}; \mathbf{w})$ with parameters $\mathbf{w} = [w_1, \ldots, w_K]^T$ would be

$$\mathbf{H} = \begin{bmatrix} \frac{\partial^2 f}{\partial w_1^2} & \frac{\partial^2 f}{\partial w_1 \partial w_2} & \cdots & \frac{\partial^2 f}{\partial w_1 \partial w_K} \\ \frac{\partial^2 f}{\partial w_2 \partial w_1} & \frac{\partial^2 f}{\partial w_2^2} & \cdots & \frac{\partial^2 f}{\partial w_2 \partial w_K} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial w_K \partial w_1} & \frac{\partial^2 f}{\partial w_K \partial w_2} & \cdots & \frac{\partial^2 f}{\partial w_K^2} \end{bmatrix}.$$

We can use the Hessian to tell us something about turning points in $f(\mathbf{x}; \mathbf{w})$. For example, if the Hessian is *negative definite* (see Comment 2.7) at some turning point $\widehat{\mathbf{w}}$, then we know that that turning point corresponds to a maximum.

Characteristics of the Maximum Likelihood Solution

- The Hessian matrix of second-order partial derivatives:

$$\frac{\partial^2 l(\boldsymbol{\omega}, \sigma^2)}{\partial \boldsymbol{\omega} \partial \boldsymbol{\omega}^T} = -\frac{1}{2\sigma^2} \frac{\partial}{\partial \boldsymbol{\omega}^T} (2\boldsymbol{X}^T \boldsymbol{X}\boldsymbol{\omega} - 2\boldsymbol{X}^T \boldsymbol{t}) = -\frac{1}{\sigma^2} \boldsymbol{X}^T \boldsymbol{X}$$

- To be sure this is a maximum, we need to determine whether or not this matrix is negative definite.

- We can show that $-\frac{1}{\sigma^2} \boldsymbol{z}^T \boldsymbol{X}^T \boldsymbol{X}\boldsymbol{z} < 0$, or $\boldsymbol{z}^T \boldsymbol{X}^T \boldsymbol{X}\boldsymbol{z} > 0$, $\forall \boldsymbol{z}$ (see textbook)

Comment 2.7 – Negative definite matrices: A real-valued matrix $\mathbf{H}$ is negative definite if

$$\mathbf{x}^\mathsf{T}\mathbf{H}\mathbf{x} < 0$$

for all vectors of real values $\mathbf{x}$.

Characteristics of the Maximum Likelihood Solution

- To ensure that $\hat{\sigma}^2 = \frac{1}{N}\left(\boldsymbol{t}^T\boldsymbol{t} - \boldsymbol{t}^T\boldsymbol{X}\hat{\boldsymbol{\omega}}\right)$ corresponds to a maximum of the likelihood,

$$\frac{\partial^2 l(\boldsymbol{\omega}, \sigma^2)}{\partial \sigma^2} = \frac{N}{\sigma^2} - \frac{3}{\sigma^4}(\boldsymbol{t} - \boldsymbol{X}\hat{\boldsymbol{\omega}})^T(\boldsymbol{t} - \boldsymbol{X}\hat{\boldsymbol{\omega}}) = \frac{N}{\hat{\sigma}^2} - \frac{3}{(\hat{\sigma}^2)^2}N\hat{\sigma}^2 = -\frac{2N}{\hat{\sigma}^2} < 0$$

Maximum Likelihood Favors Complex Models

- The maximum value of L will keep increasing as we decrease $\hat{\sigma}^2$ since

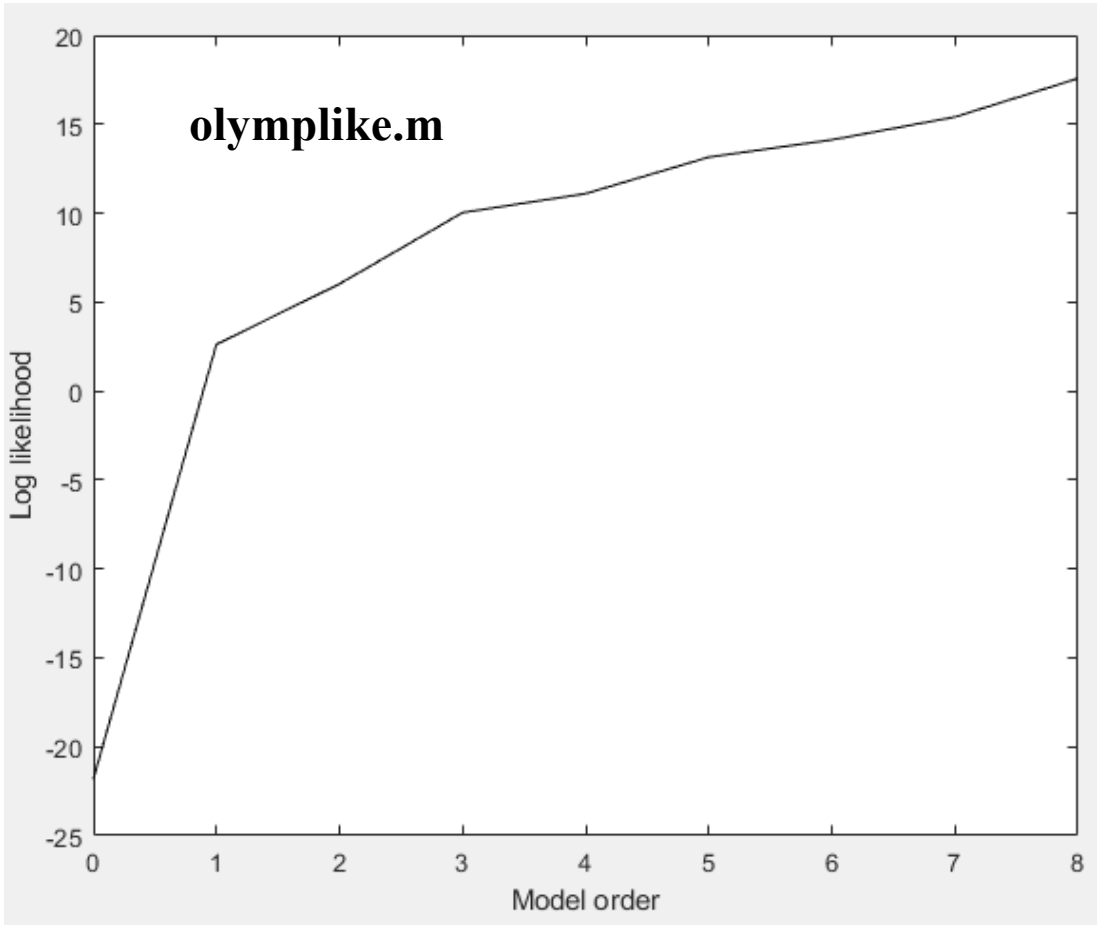
$$l(\boldsymbol{\omega}, \sigma^2) = -\frac{N}{2}(\log(2\pi) + \log\hat{\sigma}^2) - \frac{1}{2\hat{\sigma}^2}\boldsymbol{(t - X\omega)^T(t - X\omega)}$$

$$= -\frac{N}{2}(\log(2\pi) + \log\hat{\sigma}^2) - \frac{1}{2\hat{\sigma}^2}N\hat{\sigma}^2 = -\frac{N}{2}(\log(2\pi) + 1) - \frac{N}{2}\log\hat{\sigma}^2$$

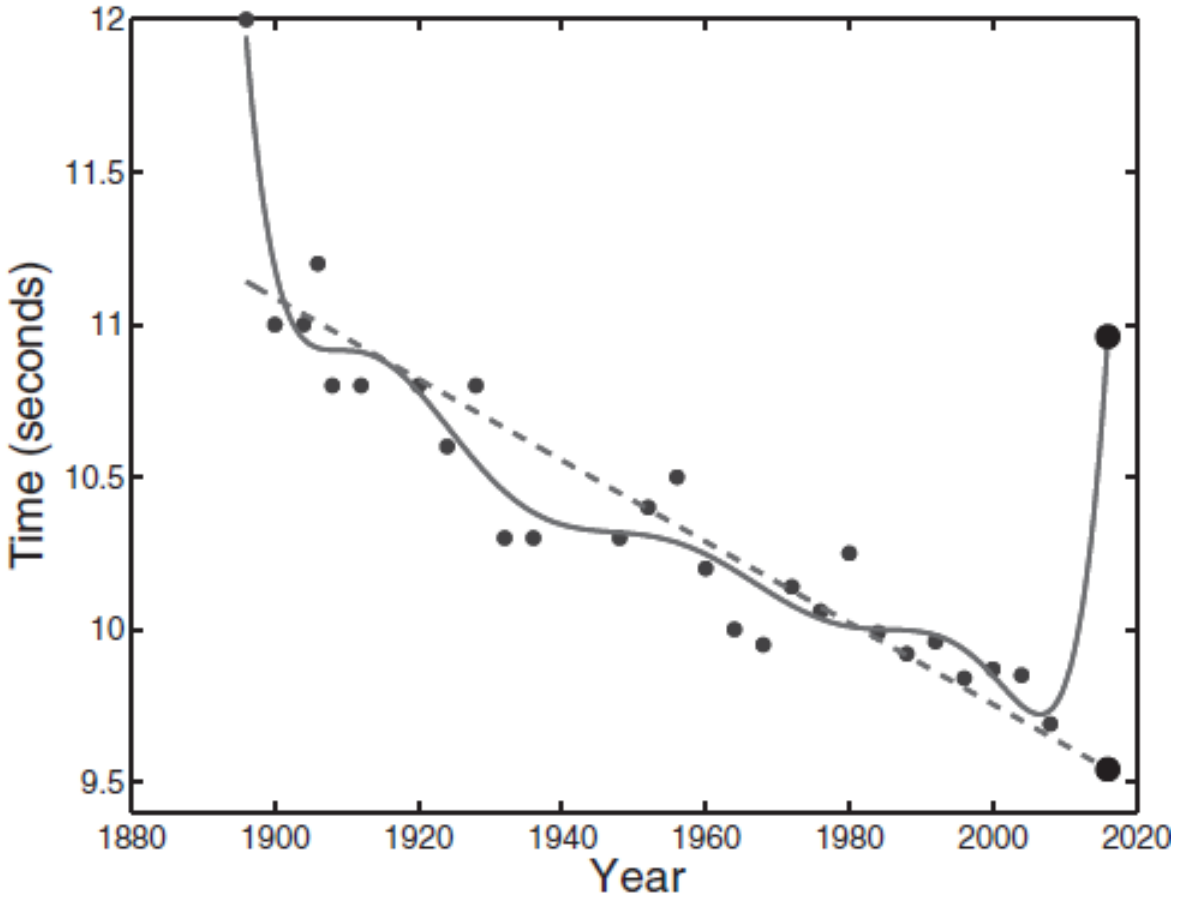
- One way to decrease $\hat{\sigma}^2$ is to modify $f(\boldsymbol{x}_n; \boldsymbol{\omega})$ so that it can capture more of the variability in the data

- Revisit the Olympic men's 100m data, the increase in likelihood as model complexity increases
  - Too simple will have a high bias (under-fitting)
  - The more complex model is over-fitting
  - There is a trade-off between generalization and over-fitting

Model Complexity Example with Olympic Men's 100m data



Increase in log-likelihood as the polynomial order increases.



First- and eighth- order polynomial functions fitted to the Olympic men's 100m data. Large dark circles correspond to predictions for the 2016 Olympics.

MATLAB script: olymplike.m

```
%% olymplike.m
% From A First Course in Machine Learning, Chapter 2.
clc;clear all;close all;
%% Load the Olympic data
load ../data/olympics/male100.csv
x = male100(:,1); % Olympic years
t = male100(:,2); % Winning times
% Rescale x for numerical stability
x = x - x(1); x = x./4;
%% Fit different order models with maximum likelihood
orders = [0:8];X = [];
N = length(x);
for i = 1:length(orders)
    X = [X x.^orders(i)];
    w = (X'*X)\(X'*t);
    ss = (1/N)*(t'*t - t'*X*w);
    log_like(i) = sum(log(normpdf(t,X*w,sqrt(ss))));
end
%% Plot the model order versus the (log) likelihood
figure(1); hold off
plot(orders, log_like,'k');xlabel('Model order');ylabel('Log likelihood');
```

The Bias-Variance Trade-Off

- The trade-off between generalization and over-fitting discussed is also described as the bias-variance trade-off.
- Suppose we fit a model $\hat{f}(x)$ to some training data, and let (x_0, t_0) be a test observation. If the true model is $t = f(\boldsymbol{X}) + \epsilon$ $\left(\text{with } f(x) = E(t|\boldsymbol{X} = x)\right)$, then

$$E\left[\left(\boldsymbol{t} - \hat{f}(\boldsymbol{X})\right)^2 \middle| \boldsymbol{X} = x\right] = Var\left(\hat{f}(x_0)\right) + [Bias(\hat{f}(x_0))]^2 + Var(\epsilon).$$

↓	↓	↓
Total error	Reducible	Irreducible

- *Variance* refers to the amount by which $\hat{f}$ would change if we estimated it using a different training data set
- $[Bias(\hat{f}(x_0))] = E\left[\hat{f}(x_0)\right] - f(x_0)$.
- Complexity ↑ $\Rightarrow$ *Variance* ↑ , *Bias* ↓

Given a model space, let $f(\mathbf{x})$ be the approximation to the underlying process $t(\mathbf{x})$ generating the data. Note $f(\mathbf{x})$ depends on the data set used to calculate the model. The measurement is $t = t(\mathbf{x}) + \epsilon$. The expected squared error at $\mathbf{x}$ is:

$$
\begin{aligned}
\mathrm{E}\left[(t - f(\mathbf{x}))^2\right] &= \mathrm{E}\left[t^2 + f(\mathbf{x})^2 - 2tf(\mathbf{x})\right] \\
&= \mathrm{E}\left[t^2\right] + \mathrm{E}\left[f(\mathbf{x})^2\right] - 2\mathrm{E}\left[tf(\mathbf{x})\right] \\
&= \mathrm{Var}\left[t\right] + \mathrm{E}\left[t\right]^2 + \mathrm{Var}\left[f(\mathbf{x})\right] + \mathrm{E}\left[f(\mathbf{x})\right]^2 - 2\mathrm{E}\left[tf(\mathbf{x})\right] \\
&= \sigma^2 + t(\mathbf{x})^2 + \mathrm{Var}\left[f(\mathbf{x})\right] + \mathrm{E}\left[f(\mathbf{x})\right]^2 - 2t(\mathbf{x})\mathrm{E}\left[f(\mathbf{x})\right] \\
&= \sigma^2 + \underbrace{\mathrm{Var}\left[f(\mathbf{x})\right]}_{\textit{variance}} + \underbrace{\left(\mathrm{E}\left[f(\mathbf{x})\right] - t(\mathbf{x})\right)^2}_{\textit{bias}}.
\end{aligned}
$$

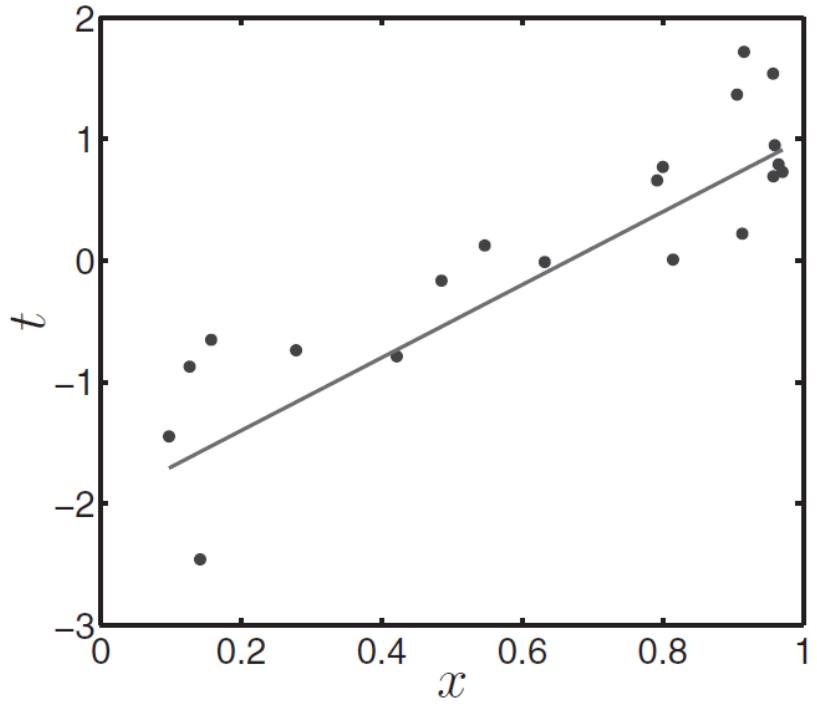
Effect of Noise on Parameter Estimates

- If there is a lot of noise (σ^2 is high), it is likely that we could tolerate reasonably large changes in $\hat{\boldsymbol{\omega}}$.

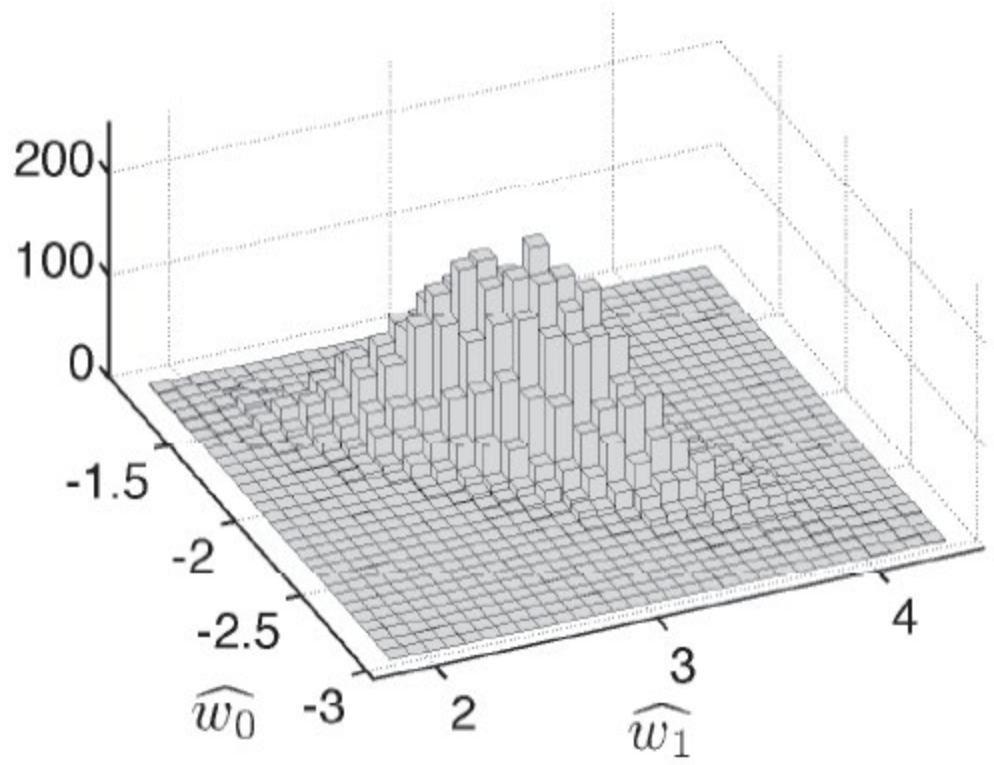
- If there is very little noise, the quality of the fit will deteriorate rapidly.

- Example: $t_n = \omega_0 + \omega_1 x_n + \epsilon_n = -2 + 3x_n + \epsilon_n, \epsilon_n \sim N(0, \sigma^2 = 0.5^2)$

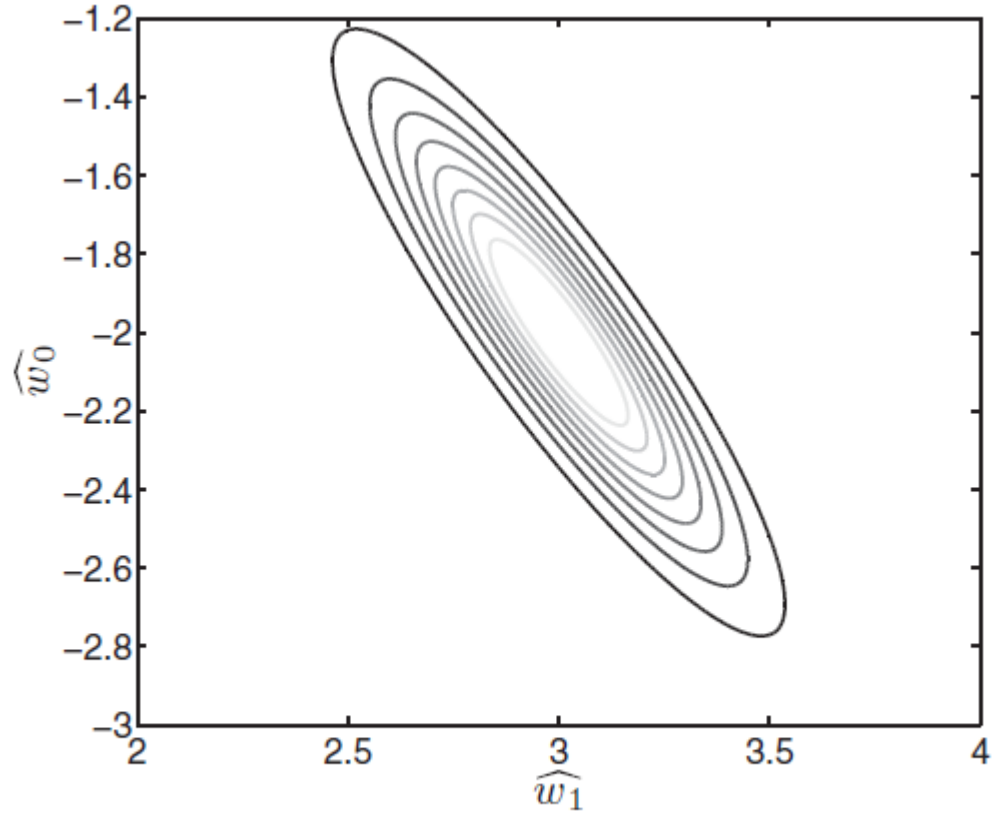
$x_n \sim U[0,1]$



Variability in $\hat{\omega}$ for 10000 Datasets Created from $t_n = -2 + 3x_n + \epsilon_n$



(a)

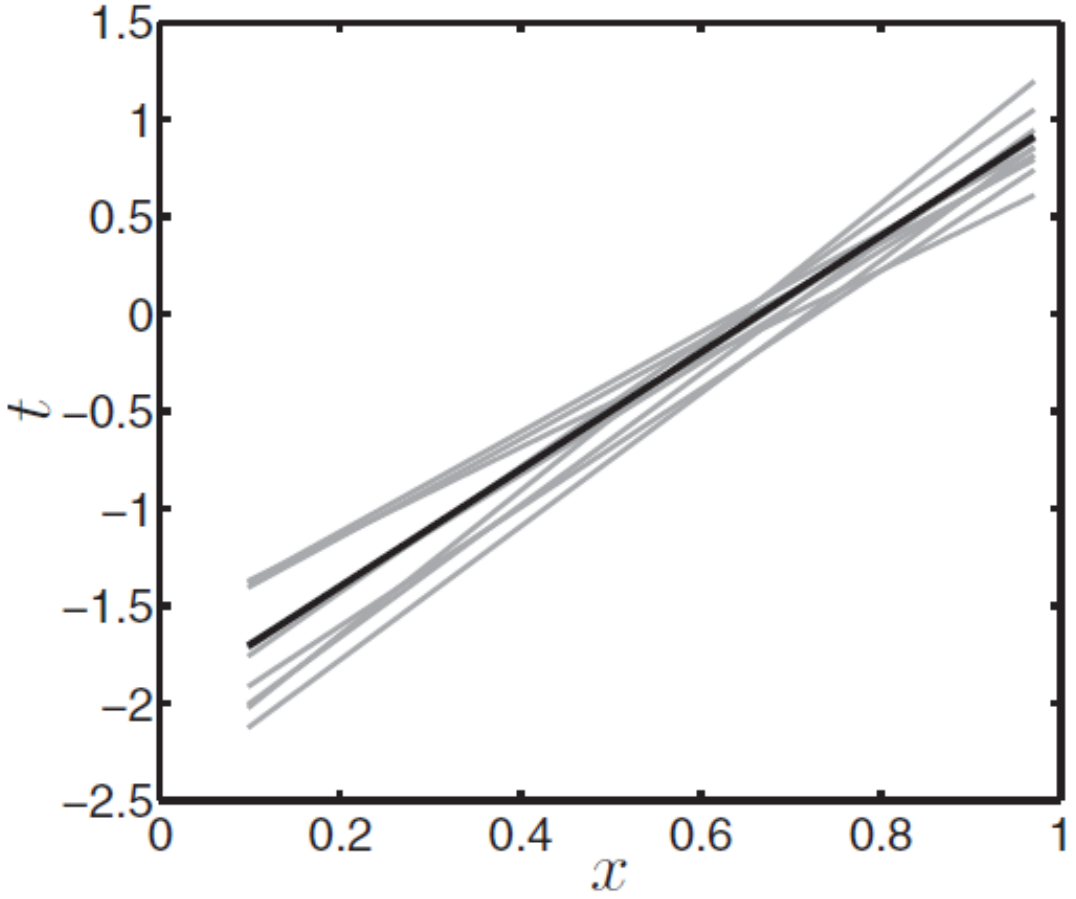


(b)

(a) Histogram: the height of each bar represents the number of datasets that resulted in parameter values within a particular range

(b) Contour plot of (a).

Functions Inferred from Ten datasets Created from $t_n = -2 + 3x_n + \epsilon_n$ as well as the true function (wider, darker line).



Uncertainty in Estimates $\hat{\boldsymbol{\omega}}$

- $\hat{\boldsymbol{\omega}}$ is strongly influenced by the particular noise values in the data.
- It would be useful to know how much uncertainty there was in $\hat{\boldsymbol{\omega}}$.
- Is this $\hat{\boldsymbol{\omega}}$ unique in explaining the data well or are there many that could do almost as well?
- Note $L = p(\boldsymbol{t}|\boldsymbol{X}, \boldsymbol{\omega}, \sigma^2) = \prod_{n=1}^{N} p(t_n|\boldsymbol{x}_n, \boldsymbol{\omega}, \sigma^2) = \prod_{n=1}^{N} N(\boldsymbol{\omega}^T \boldsymbol{x}_n, \sigma^2)$
  $= N(\boldsymbol{X}\boldsymbol{\omega}, \sigma^2 \mathbf{I})$
- Computing the expectation of $\hat{\boldsymbol{\omega}}$ w.r.t. the generating distribution will tell us what we expect $\hat{\boldsymbol{\omega}}$ on average to be.

Uncertainty in Estimates $\hat{\boldsymbol{\omega}}$

- $$E_{p(\boldsymbol{t}|\boldsymbol{X},\boldsymbol{\omega},\sigma^2)}\{\hat{\boldsymbol{\omega}}\} = \int \hat{\boldsymbol{\omega}} p(\boldsymbol{t}|\boldsymbol{X},\boldsymbol{\omega},\sigma^2) d\boldsymbol{t}$$
$$= (\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T \int \boldsymbol{t} p(\boldsymbol{t}|\boldsymbol{X},\boldsymbol{\omega},\sigma^2) d\boldsymbol{t}$$
$$= (\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T E_{p(\boldsymbol{t}|\boldsymbol{X},\boldsymbol{\omega},\sigma^2)}\{\boldsymbol{t}\}$$
$$= (\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T \boldsymbol{X}\boldsymbol{\omega}$$
$$= \boldsymbol{\omega}$$

- This result tells us that the expected value of $\hat{\boldsymbol{\omega}}$ is the true parameter value $\boldsymbol{\omega}$, it means that our estimator is unbiased: it is not, on average, too big or too small.

- The potential variability in the estimate of $\hat{\boldsymbol{\omega}}$ is encapsulated in its covariance matrix.

Covariance Matrix of $\hat{\boldsymbol{\omega}}$

- First note that $p(\boldsymbol{t}|\boldsymbol{X},\boldsymbol{\omega},\sigma^2) = N(\boldsymbol{X\omega},\sigma^2\mathbf{I})$

$$\Rightarrow \text{cov}\{\boldsymbol{t}\} = \sigma^2\mathbf{I} = E_{p(\boldsymbol{t}|\boldsymbol{X},\boldsymbol{\omega},\sigma^2)}\{\boldsymbol{tt}^T\} - E_{p(\boldsymbol{t}|\boldsymbol{X},\boldsymbol{\omega},\sigma^2)}\{\boldsymbol{t}\}E_{p(\boldsymbol{t}|\boldsymbol{X},\boldsymbol{\omega},\sigma^2)}\{\boldsymbol{t}\}^T$$

$$\Rightarrow E_{p(\boldsymbol{t}|\boldsymbol{X},\boldsymbol{\omega},\sigma^2)}\{\boldsymbol{tt}^T\} = E_{p(\boldsymbol{t}|\boldsymbol{X},\boldsymbol{\omega},\sigma^2)}\{\boldsymbol{t}\}E_{p(\boldsymbol{t}|\boldsymbol{X},\boldsymbol{\omega},\sigma^2)}\{\boldsymbol{t}\}^T + \sigma^2\mathbf{I}$$

$$= \boldsymbol{X\omega}(\boldsymbol{X\omega})^T + \sigma^2\mathbf{I} = \boldsymbol{X\omega\omega}^T\boldsymbol{X}^T + \sigma^2\mathbf{I}$$

- $\text{cov}\{\hat{\boldsymbol{\omega}}\} = E_{p(\boldsymbol{t}|\boldsymbol{X},\boldsymbol{\omega},\sigma^2)}\{\hat{\boldsymbol{\omega}}\hat{\boldsymbol{\omega}}^T\} - E_{p(\boldsymbol{t}|\boldsymbol{X},\boldsymbol{\omega},\sigma^2)}\{\hat{\boldsymbol{\omega}}\}E_{p(\boldsymbol{t}|\boldsymbol{X},\boldsymbol{\omega},\sigma^2)}\{\hat{\boldsymbol{\omega}}\}^T$

$$= E_{p(\boldsymbol{t}|\boldsymbol{X},\boldsymbol{\omega},\sigma^2)}\{\hat{\boldsymbol{\omega}}\hat{\boldsymbol{\omega}}^T\} - \boldsymbol{\omega\omega}^T, \quad \text{since } \hat{\boldsymbol{\omega}} \text{ is unbiased}$$

$$= E_{p(\boldsymbol{t}|\boldsymbol{X},\boldsymbol{\omega},\sigma^2)}\{(\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{t}((\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{t})^T\} - \boldsymbol{\omega\omega}^T$$

$$= E_{p(\boldsymbol{t}|\boldsymbol{X},\boldsymbol{\omega},\sigma^2)}\{(\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{tt}^T\boldsymbol{X}(\boldsymbol{X}^T\boldsymbol{X})^{-1}\} - \boldsymbol{\omega\omega}^T$$

$$= (\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T E_{p(\boldsymbol{t}|\boldsymbol{X},\boldsymbol{\omega},\sigma^2)}\{\boldsymbol{tt}^T\}\boldsymbol{X}(\boldsymbol{X}^T\boldsymbol{X})^{-1} - \boldsymbol{\omega\omega}^T$$

$$= (\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T(\boldsymbol{X\omega\omega}^T\boldsymbol{X}^T + \sigma^2\mathbf{I})\boldsymbol{X}(\boldsymbol{X}^T\boldsymbol{X})^{-1} - \boldsymbol{\omega\omega}^T$$

$$= \boldsymbol{\omega\omega}^T + \sigma^2(\boldsymbol{X}^T\boldsymbol{X})^{-1} - \boldsymbol{\omega\omega}^T = \sigma^2(\boldsymbol{X}^T\boldsymbol{X})^{-1}$$

Fisher Information Matrix ($\boldsymbol{I}$)

- Recall that $\frac{\partial^2 l(\boldsymbol{\omega},\sigma^2)}{\partial \boldsymbol{\omega} \partial \boldsymbol{\omega}^T} = -\frac{1}{2\sigma^2}\frac{\partial}{\partial \boldsymbol{\omega}^T}(2\boldsymbol{X}^T\boldsymbol{X}\boldsymbol{\omega} - 2\boldsymbol{X}^T\boldsymbol{t}) = -\frac{1}{\sigma^2}\boldsymbol{X}^T\boldsymbol{X}$

$\Rightarrow \text{cov}\{\hat{\boldsymbol{\omega}}\} = \sigma^2(\boldsymbol{X}^T\boldsymbol{X})^{-1} = -(\frac{\partial^2 l(\boldsymbol{\omega},\sigma^2)}{\partial \boldsymbol{\omega} \partial \boldsymbol{\omega}^T})^{-1}$

This result tells us that the certainty/uncertainty in $\hat{\boldsymbol{\omega}}$ (as described by $\text{cov}\{\hat{\boldsymbol{\omega}}\}$) is directly linked to the second derivative of the log-likelihood.

- The second derivative of the log-likelihood tells us about the curvature of the likelihood function.
  - Low curvature corresponds to a high level of uncertainty in $\hat{\boldsymbol{\omega}}$
  - High curvature corresponds to a low level of uncertainty in $\hat{\boldsymbol{\omega}}$.

$\Rightarrow$ We have an expression that tells us how much information our data gives us regarding our parameter estimates.

Fisher Information Matrix (I)

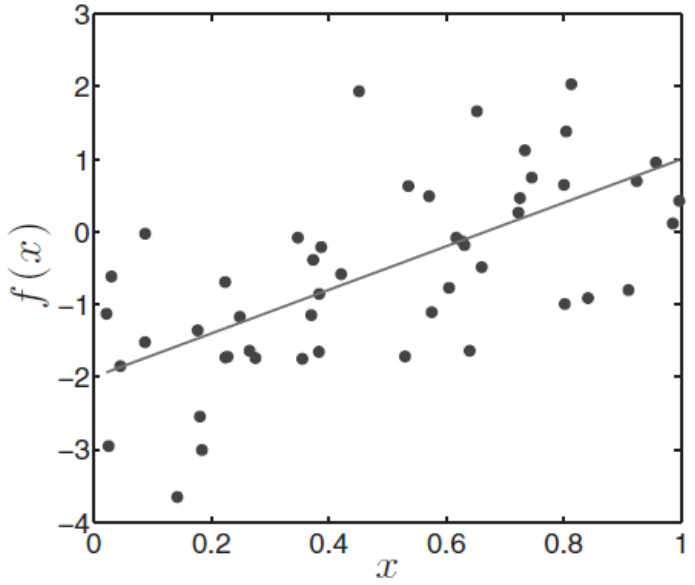
- The Fisher Information Matrix is defined as the expected value of the matrix of second derivatives of the log-likelihood:

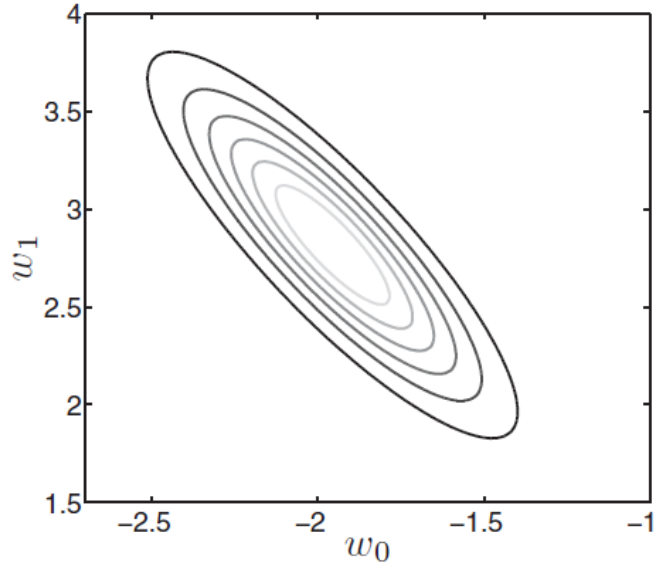
$$I \equiv E_{p(\boldsymbol{t}|\boldsymbol{X},\boldsymbol{\omega},\sigma^2)}\left\{-\frac{\partial^2 \log p(\boldsymbol{t}|\boldsymbol{X},\boldsymbol{\omega},\sigma^2)}{\partial \boldsymbol{\omega} \partial \boldsymbol{\omega}^T}\right\}$$

- Therefore $\text{cov}\{\hat{\boldsymbol{\omega}}\} = I^{-1}$ or $I = \frac{1}{\sigma^2}\boldsymbol{X}^T\boldsymbol{X}$

- In general, if the information content is high, the data can inform a very accurate parameter estimate and $\text{cov}\{\hat{\boldsymbol{\omega}}\}$ will be low. If the information content is low, the covariance will be high.

Two Example Datasets with Different Noise Levels and the Corresponding Likelihood Functions.

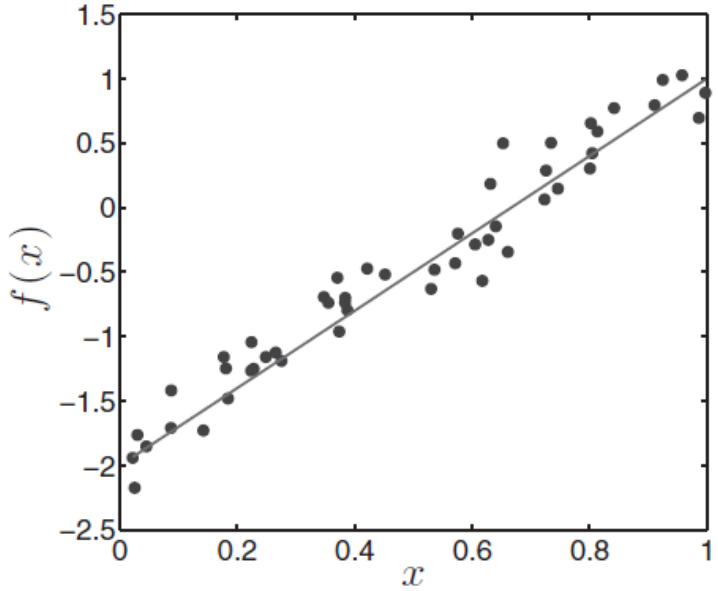


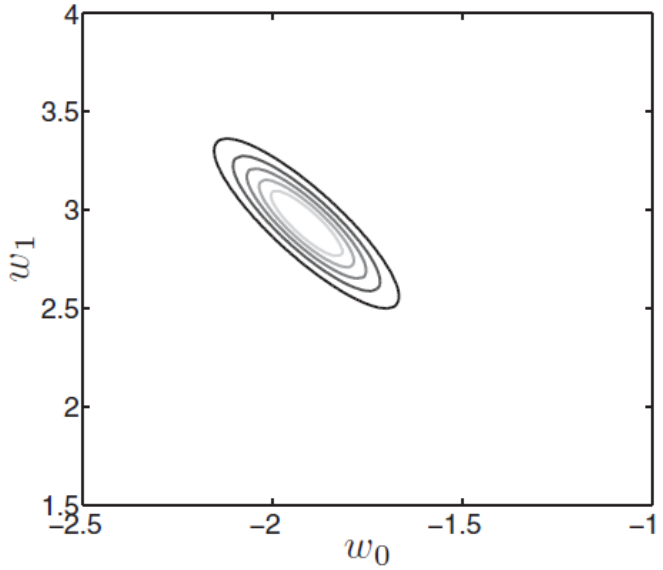


True function $t = -2 + 3x$

$$\mathcal{I} = \begin{bmatrix} 50.0000 & 24.3311 \\ 24.3311 & 15.8953 \end{bmatrix}$$

$$\text{cov}\{\widehat{\mathbf{w}}\} = \begin{bmatrix} 0.0784 & -0.1200 \\ -0.1200 & 0.2466 \end{bmatrix}$$





$$\mathcal{I} = \begin{bmatrix} 1.2500 \times 10^3 & 0.6083 \times 10^3 \\ 0.6083 \times 10^3 & 0.3974 \times 10^3 \end{bmatrix}$$

$$\text{cov}\{\widehat{\mathbf{w}}\} = \begin{bmatrix} 0.0031 & -0.0048 \\ -0.0048 & 0.0099 \end{bmatrix}$$

Comparison with Empirical Values

- Recall in men's Olympic 100m data, we obtain

$$t_n = \omega_0 + \omega_1 x_n + \epsilon_n, \hat{\boldsymbol{\omega}} = \begin{bmatrix} \hat{\omega}_0 \\ \hat{\omega}_1 \end{bmatrix} = \begin{bmatrix} 35.416 \\ -0.0133 \end{bmatrix}, \hat{\sigma}^2 = 0.0503$$

- We compute $\text{cov}\{\hat{\boldsymbol{\omega}}\} = \sigma^2 (\boldsymbol{X}^T \boldsymbol{X})^{-1} = \begin{bmatrix} 5.7972 & -0.003 \\ -0.003 & 1.5204 \times 10^{-6} \end{bmatrix}$

- The variance of $\hat{\omega}_0$ (5.7972) is much higher than the variance in $\hat{\omega}_1$ (1.5204×10^{-6}), suggesting that we could tolerate bigger changes in $\hat{\omega}_0$ than $\hat{\omega}_1$ and still be left with a reasonably good model.

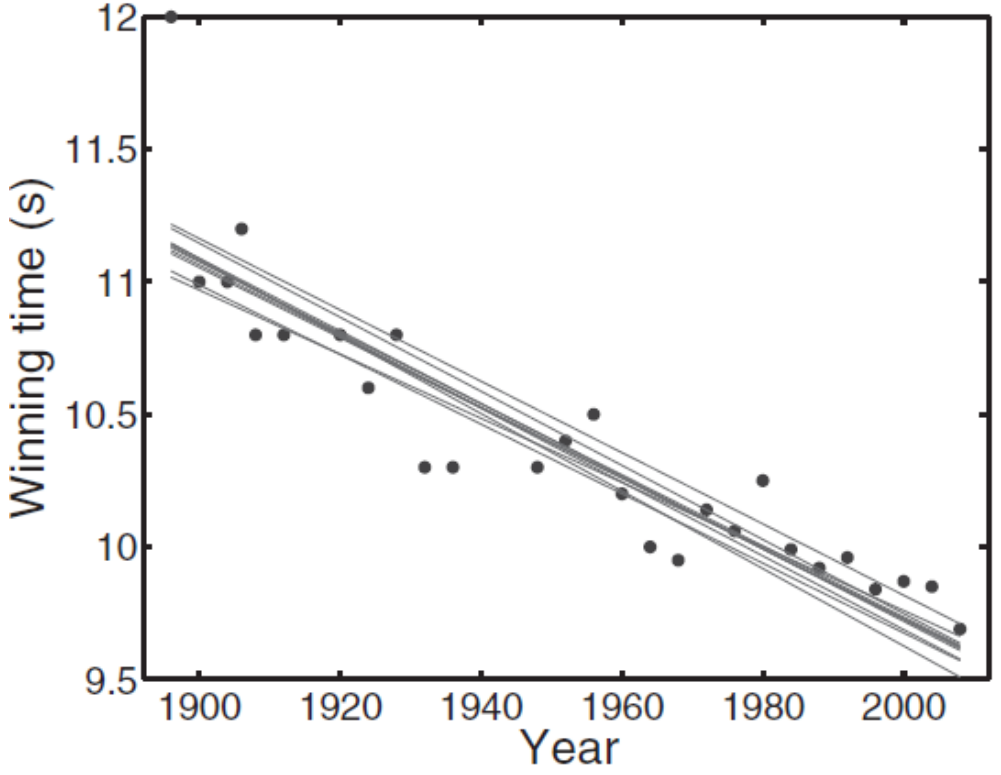
- The negativity of the off-diagonal elements tell us that, if we were to slightly increase either $\hat{\omega}_0$ or $\hat{\omega}_1$, we have to slightly decrease the other.

Comparison with Empirical Values

- Another way to get a feeling for the meaning of $\mathrm{cov}\{\hat{\boldsymbol{\omega}}\}$ is to look at the variability in models.

- To do this, we assume $\boldsymbol{\omega} \sim N(\hat{\boldsymbol{\omega}}, \mathrm{cov}\{\hat{\boldsymbol{\omega}}\})$ and sample several instances (say, 10, lower plot) of $\boldsymbol{\omega}$ and plot the resulting models

- Very little change in gradient (ω_1) across the ten samples but that this small gradient change, if we extrapolated back to year zero, result in quite a large change of ω_0.

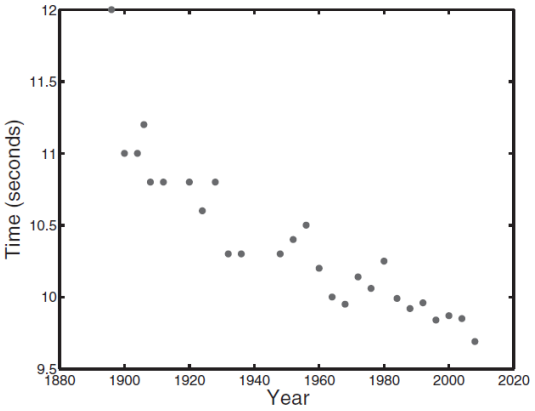


Variability in Predictions

- Suppose we observe a new set of attributes, $\boldsymbol{x}_{\text{new}}$. We want to predict $t_{\text{new}} = \hat{\boldsymbol{\omega}}^T \boldsymbol{x}_{\text{new}}$ and its variability $\sigma^2_{\text{new}} = \text{var}\{t_{\text{new}}\}$.
- $$\begin{aligned}\text{var}\{t_{\text{new}}\} &= E_{p(\boldsymbol{t}|\boldsymbol{X},\boldsymbol{\omega},\sigma^2)}\{t^2_{\text{new}}\} - (E_{p(\boldsymbol{t}|\boldsymbol{X},\boldsymbol{\omega},\sigma^2)}\{t_{\text{new}}\})^2 \\ &= E_{p(\boldsymbol{t}|\boldsymbol{X},\boldsymbol{\omega},\sigma^2)}\{(\hat{\boldsymbol{\omega}}^T \boldsymbol{x}_{\text{new}})^2\} - (E_{p(\boldsymbol{t}|\boldsymbol{X},\boldsymbol{\omega},\sigma^2)}\{\hat{\boldsymbol{\omega}}^T \boldsymbol{x}_{\text{new}}\})^2 \\ &= E_{p(\boldsymbol{t}|\boldsymbol{X},\boldsymbol{\omega},\sigma^2)}\{\boldsymbol{x}^T_{\text{new}} \hat{\boldsymbol{\omega}}\hat{\boldsymbol{\omega}}^T \boldsymbol{x}_{\text{new}}\} - (E_{p(\boldsymbol{t}|\boldsymbol{X},\boldsymbol{\omega},\sigma^2)}\{\hat{\boldsymbol{\omega}}^T\}\boldsymbol{x}_{\text{new}})^2 \\ &= \boldsymbol{x}^T_{\text{new}} E_{p(\boldsymbol{t}|\boldsymbol{X},\boldsymbol{\omega},\sigma^2)}\{\hat{\boldsymbol{\omega}}\hat{\boldsymbol{\omega}}^T\}\boldsymbol{x}_{\text{new}} - (\boldsymbol{\omega}\boldsymbol{x}_{\text{new}})^2, \ \hat{\boldsymbol{\omega}} \text{ is unbiased}\end{aligned}$$

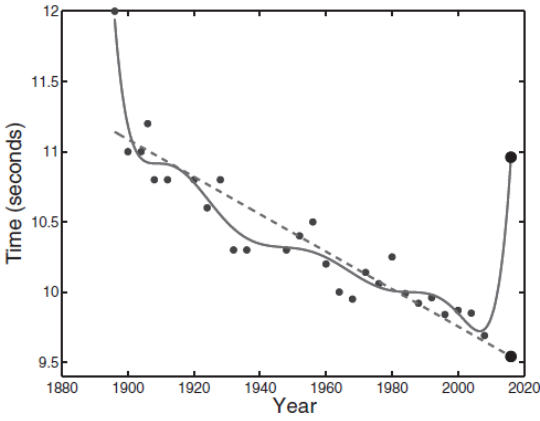
$$\begin{aligned}&= \boldsymbol{x}^T_{\text{new}} E_{p(\boldsymbol{t}|\boldsymbol{X},\boldsymbol{\omega},\sigma^2)}\{(\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{t}((\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{t})^T\}\boldsymbol{x}_{\text{new}} - \boldsymbol{x}^T_{\text{new}}\boldsymbol{\omega}\boldsymbol{\omega}^T\boldsymbol{x}_{\text{new}} \\ &= \boldsymbol{x}^T_{\text{new}} E_{p(\boldsymbol{t}|\boldsymbol{X},\boldsymbol{\omega},\sigma^2)}\{(\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{t}\boldsymbol{t}^T\boldsymbol{X}(\boldsymbol{X}^T\boldsymbol{X})^{-1}\}\boldsymbol{x}_{\text{new}} - \boldsymbol{x}^T_{\text{new}}\boldsymbol{\omega}\boldsymbol{\omega}^T\boldsymbol{x}_{\text{new}} \\ &= \boldsymbol{x}^T_{\text{new}}(\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T E_{p(\boldsymbol{t}|\boldsymbol{X},\boldsymbol{\omega},\sigma^2)}\{\boldsymbol{t}\boldsymbol{t}^T\}\boldsymbol{X}(\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{x}_{\text{new}} - \boldsymbol{x}^T_{\text{new}}\boldsymbol{\omega}\boldsymbol{\omega}^T\boldsymbol{x}_{\text{new}} \\ &= \boldsymbol{x}^T_{\text{new}}(\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T(\boldsymbol{X}\boldsymbol{\omega}\boldsymbol{\omega}^T\boldsymbol{X}^T + \sigma^2\mathbf{I})\boldsymbol{X}(\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{x}_{\text{new}} - \boldsymbol{x}^T_{\text{new}}\boldsymbol{\omega}\boldsymbol{\omega}^T\boldsymbol{x}_{\text{new}} \\ &= \boldsymbol{x}^T_{\text{new}}\boldsymbol{\omega}\boldsymbol{\omega}^T\boldsymbol{x}_{\text{new}} + \sigma^2\boldsymbol{x}^T_{\text{new}}(\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{x}_{\text{new}} - \boldsymbol{x}^T_{\text{new}}\boldsymbol{\omega}\boldsymbol{\omega}^T\boldsymbol{x}_{\text{new}} \\ &= \boldsymbol{x}^T_{\text{new}}\sigma^2(\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{x}_{\text{new}} = \boldsymbol{x}^T_{\text{new}}\text{cov}\{\hat{\boldsymbol{\omega}}\}\boldsymbol{x}_{\text{new}}\end{aligned}$$

Summary of the Fitting and Prediction of Olympic 100m data



- Given the dataset $(x_1, t_1),(x_2, t_2), \ldots ,(x_N, t_N)$,

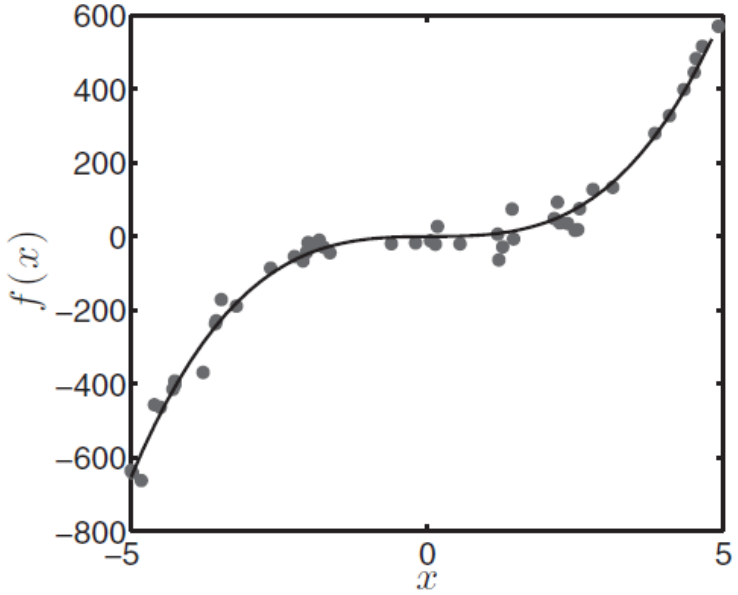
- $t_n = \omega_0 + \omega_1 x_n + \epsilon_n$, is random variable, $\epsilon_n \sim N(0, \sigma^2)$
  $\Rightarrow\ p(t_n|\boldsymbol{x}_n, \boldsymbol{\omega}) = N(\boldsymbol{\omega}^T \boldsymbol{x}_n, \sigma^2)$
  $\Rightarrow$ maximize $\log \prod_{n=1}^{N} p(t_n|\boldsymbol{x}_n, \boldsymbol{\omega}, \sigma^2) = \sum_{n=1}^{N} \log N(\boldsymbol{\omega}^T \boldsymbol{x}_n, \sigma^2)$
  $\Rightarrow \hat{\boldsymbol{\omega}} = \begin{bmatrix} \hat{\omega}_0 \\ \hat{\omega}_1 \end{bmatrix} = (\boldsymbol{X}^T \boldsymbol{X})^{-1} \boldsymbol{X}^T \boldsymbol{t} = \begin{bmatrix} 35.416 \\ -0.0133 \end{bmatrix}$,
  $\Rightarrow \hat{\sigma}^2 = \frac{1}{N}\left(\boldsymbol{t}^T \boldsymbol{t} - \boldsymbol{t}^T \boldsymbol{X} \hat{\boldsymbol{\omega}}\right) = \frac{1}{N} \sum_{n=1}^{N} (t_n - \boldsymbol{x}_n{}^T \hat{\boldsymbol{\omega}})^2 = 0.0503$



- $E_{p(\boldsymbol{t}|\boldsymbol{X},\boldsymbol{\omega},\sigma^2)}\{\hat{\boldsymbol{\omega}}\} = \boldsymbol{\omega}$, unbiased, $E_{p(\boldsymbol{t}|\boldsymbol{X},\boldsymbol{\omega},\sigma^2)}\{\hat{\sigma}^2\} = ??$
- $\text{cov}\{\hat{\boldsymbol{\omega}}\} = \sigma^2 (\boldsymbol{X}^T \boldsymbol{X})^{-1} = \begin{bmatrix} 5.7972 & -0.003 \\ -0.003 & 1.5204 \times 10^{-6} \end{bmatrix} = I^{-1}$ or $I = \frac{1}{\sigma^2} \boldsymbol{X}^T \boldsymbol{X}$

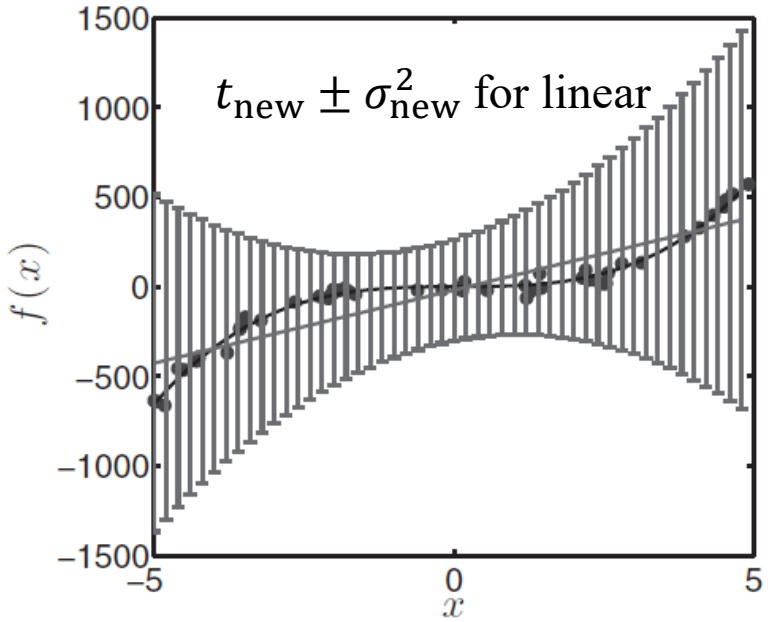
- $t_{\text{new}} = \hat{\boldsymbol{\omega}}^T \boldsymbol{x}_{\text{new}}$, $\sigma^2_{\text{new}} = \boldsymbol{x}^T_{\text{new}} \text{cov}\{\hat{\boldsymbol{\omega}}\} \boldsymbol{x}_{\text{new}}$

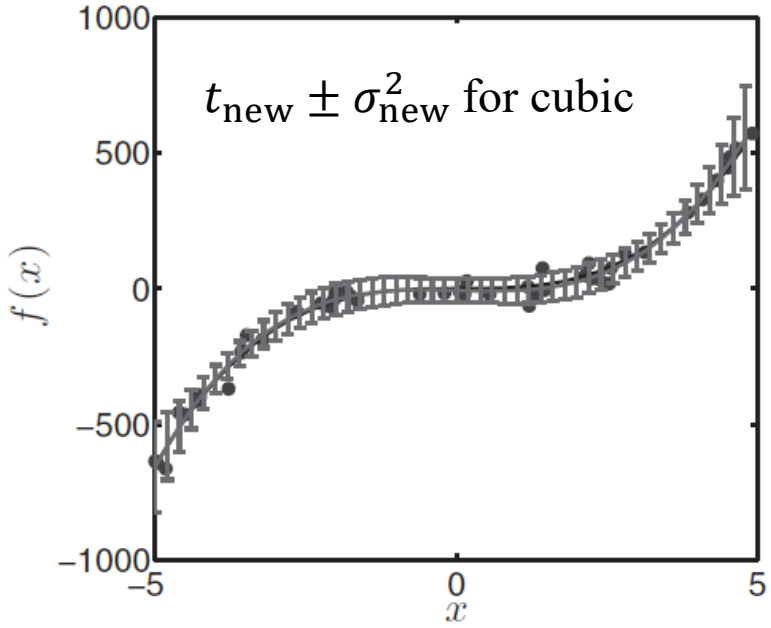
Predictive Variability: An Example

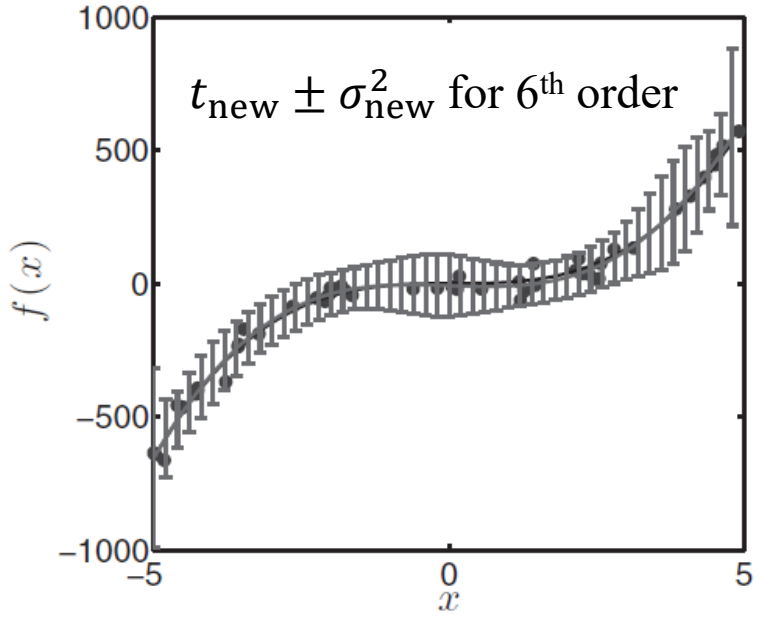


$\Leftarrow f(x) = 5x^3 - x^2 + x + \epsilon, \epsilon \sim \mathrm{N}(0,1000)$

- The linear model has very high predictive variance
- The cubic model is better able to model the trend and this is reflected in its much more confident predictions
- The sixth-order model is over-complex: it has too much freedom and fit the data well for a large range of parameter values

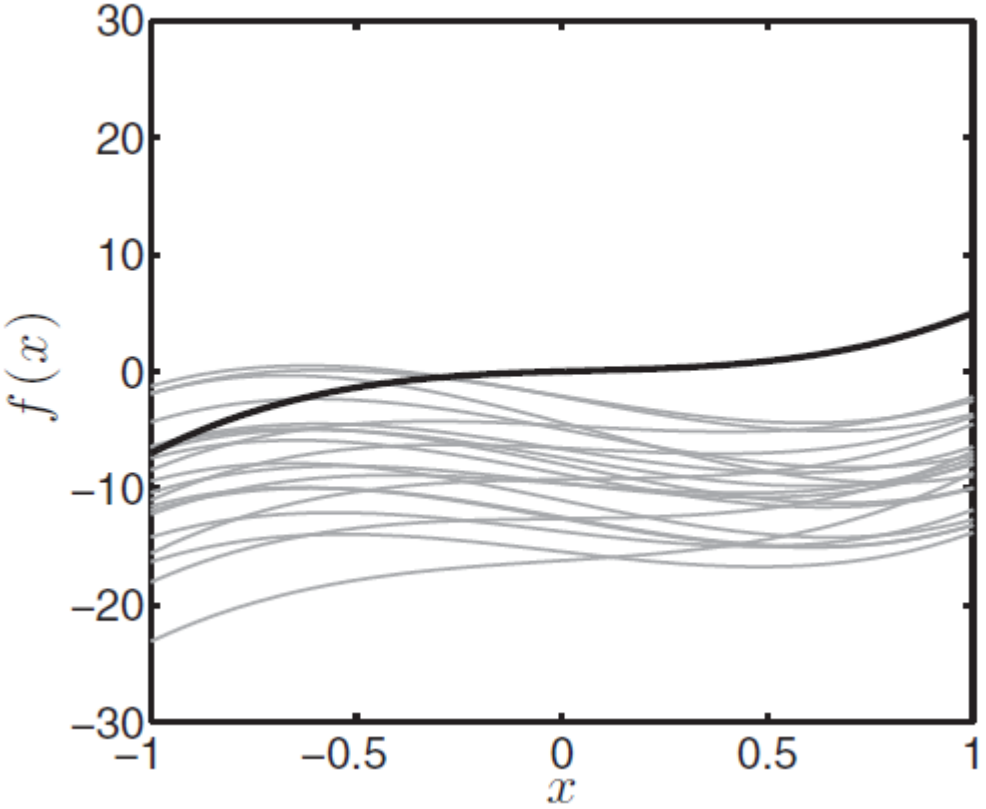


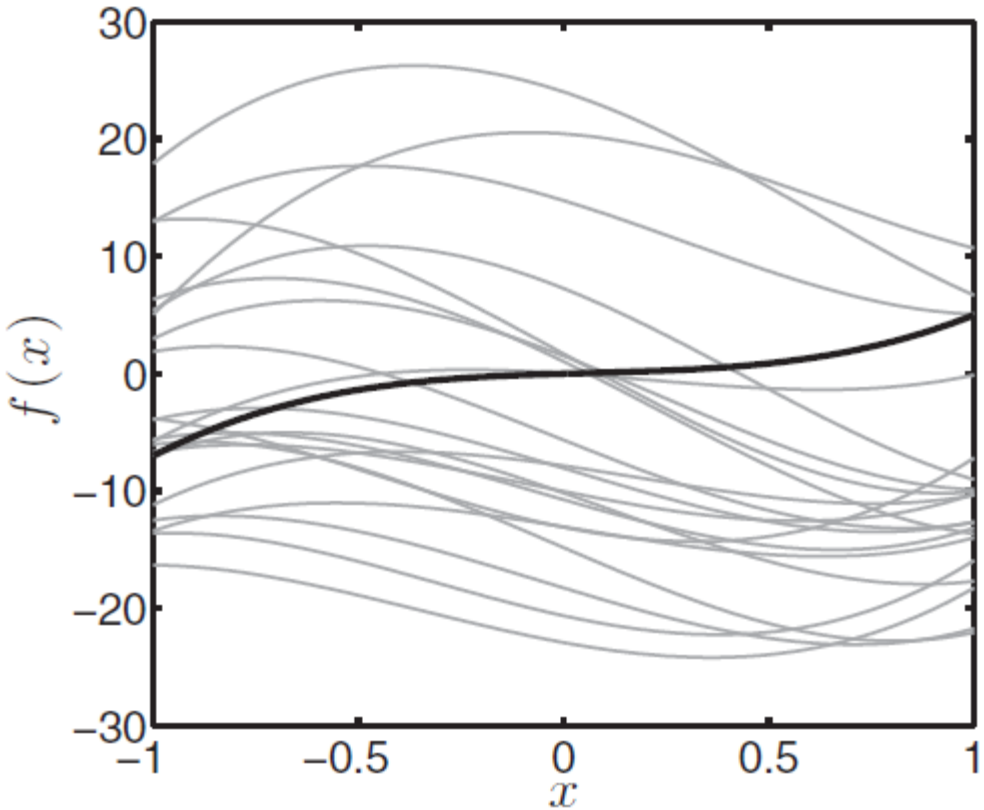




Predictive Variability: An Example

- 20 Examples of functions with parameters $\boldsymbol{\omega} \sim N(\hat{\boldsymbol{\omega}}, \text{cov}\{\hat{\boldsymbol{\omega}}\})$ for the third and sixth-order models (the plot is zoomed into a small region of x and the darker line shows the true function)
- The increased variability in possible functions caused by the increase in parameter uncertainty is clear for the sixth-order model.





Expected Values of the Estimators

- We showed $E_{p(\boldsymbol{t}|\boldsymbol{X},\boldsymbol{\omega},\sigma^2)}\{\hat{\boldsymbol{\omega}}\} = \boldsymbol{\omega}$ is unbiased
- How about $E_{p(\boldsymbol{t}|\boldsymbol{X},\boldsymbol{\omega},\sigma^2)}\{\hat{\sigma}^2\} = \sigma^2$??
- $$E_{p(\boldsymbol{t}|\boldsymbol{X},\boldsymbol{\omega},\sigma^2)}\{\hat{\sigma}^2\} = E_{p(\boldsymbol{t}|\boldsymbol{X},\boldsymbol{\omega},\sigma^2)}\left\{\frac{1}{N}\left(\boldsymbol{t}^T\boldsymbol{t} - \boldsymbol{t}^T\boldsymbol{X}\hat{\boldsymbol{\omega}}\right)\right\}$$
$$= E_{p(\boldsymbol{t}|\boldsymbol{X},\boldsymbol{\omega},\sigma^2)}\left\{\frac{1}{N}\left(\boldsymbol{t}^T\boldsymbol{t} - \boldsymbol{t}^T\boldsymbol{X}(\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{t}\right)\right\}$$
$$= \frac{1}{N}E_{p(\boldsymbol{t}|\boldsymbol{X},\boldsymbol{\omega},\sigma^2)}\{\boldsymbol{t}^T\boldsymbol{t}\} - \frac{1}{N}E_{p(\boldsymbol{t}|\boldsymbol{X},\boldsymbol{\omega},\sigma^2)}\{\boldsymbol{t}^T\boldsymbol{X}(\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{t}\}$$

- Note if $\boldsymbol{t}\sim N(\boldsymbol{\mu},\sum) \Rightarrow E_{p(\boldsymbol{t})}\{\boldsymbol{t}^T\mathrm{A}\boldsymbol{t}\} = \mathrm{Tr}(\mathrm{A}\sum) + \boldsymbol{\mu}^T\mathrm{A}\boldsymbol{\mu}$
- Since $p(\boldsymbol{t}|\boldsymbol{X},\boldsymbol{\omega},\sigma^2)\sim N(\boldsymbol{X}\boldsymbol{\omega},\sigma^2\mathbf{I}_N)$

$$\Rightarrow E_{p(\boldsymbol{t}|\boldsymbol{X},\boldsymbol{\omega},\sigma^2)}\{\boldsymbol{t}^T\mathbf{I}_N\boldsymbol{t}\} = \mathrm{Tr}(\mathbf{I}_N\sigma^2\mathbf{I}_N) + \boldsymbol{\omega}^T\boldsymbol{X}^T\mathbf{I}_N\boldsymbol{X}\boldsymbol{\omega} = N\sigma^2 + \boldsymbol{\omega}^T\boldsymbol{X}^T\boldsymbol{X}\boldsymbol{\omega}$$

and $E_{p(\boldsymbol{t}|\boldsymbol{X},\boldsymbol{\omega},\sigma^2)}\{\boldsymbol{t}^T\boldsymbol{X}(\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{t}\} = \mathrm{Tr}(\boldsymbol{X}(\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\sigma^2\mathbf{I}_N)$

$$+\boldsymbol{\omega}^T\boldsymbol{X}^T\boldsymbol{X}(\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{X}\boldsymbol{\omega}$$

Comment 2.8 – Matrix trace: The trace of a square matrix $\mathbf{A}$, denoted $\mathsf{Tr}(\mathbf{A})$, is the sum of the diagonal elements of $\mathbf{A}$. For example, if

$$\mathbf{A} = \begin{bmatrix} A_{11} & A_{12} & \cdots & A_{1D} \\ A_{21} & A_{22} & \cdots & A_{2D} \\ \vdots & \vdots & \ddots & \vdots \\ A_{D1} & A_{D2} & \cdots & A_{DD} \end{bmatrix},$$

then

$$\mathsf{Tr}(\mathbf{A}) = \sum_{d=1}^{D} A_{dd}.$$

It follows that, if $\mathbf{A} = \mathbf{I}_D$, i.e. the $D \times D$ identity matrix,

$$\mathsf{Tr}(\mathbf{I}_D) = \sum_{d=1}^{D} 1 = D.$$

A useful identity that we will often use is that

$$\mathsf{Tr}(\mathbf{AB}) = \mathsf{Tr}(\mathbf{BA}).$$

Also, the trace of a scalar is just equal to the scalar value (a scalar could be thought of as a 1×1 matrix), i.e.

$$\mathsf{Tr}(a) = a,$$

or, if $\mathbf{w} = [w_1, \ldots, w_D]^\mathsf{T}$,

$$\mathsf{Tr}(\mathbf{w}^\mathsf{T}\mathbf{w}) = \mathbf{w}^\mathsf{T}\mathbf{w}$$

because the result of $\mathbf{w}^\mathsf{T}\mathbf{w}$ is a scalar.

Expected Values of the Estimators

- $E_{p(\boldsymbol{t}|\boldsymbol{X},\boldsymbol{\omega},\sigma^2)}\{\hat{\sigma}^2\} = \frac{1}{N} E_{p(\boldsymbol{t}|\boldsymbol{X},\boldsymbol{\omega},\sigma^2)}\{\boldsymbol{t}^T\boldsymbol{t}\} - \frac{1}{N} E_{p(\boldsymbol{t}|\boldsymbol{X},\boldsymbol{\omega},\sigma^2)}\{\boldsymbol{t}^T\boldsymbol{X}(\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{t}\}$

  $= \sigma^2 + \frac{1}{N}\boldsymbol{\omega}^T\boldsymbol{X}^T\boldsymbol{X}\boldsymbol{\omega} - \frac{1}{N}\sigma^2\mathrm{Tr}(\boldsymbol{X}(\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T) - \frac{1}{N}\boldsymbol{\omega}^T\boldsymbol{X}^T\boldsymbol{X}\boldsymbol{\omega}$

  $= \sigma^2 - \frac{1}{N}\sigma^2\mathrm{Tr}(\boldsymbol{X}(\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T)$, note $\mathrm{Tr}(\mathrm{AB}) = \mathrm{Tr}(\mathrm{BA})$

  $= \sigma^2 - \frac{1}{N}\sigma^2\mathrm{Tr}((\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{X})$,

  $= \sigma^2 - \frac{1}{N}\sigma^2\mathrm{Tr}(\mathbf{I}_\mathrm{D})$, D is the number of columns in $\boldsymbol{X}$

  $= \sigma^2(1 - \frac{\mathrm{D}}{N})$, D is also the number of parameters

- Assume that $\mathrm{D} < N \Rightarrow E_{p(\boldsymbol{t}|\boldsymbol{X},\boldsymbol{\omega},\sigma^2)}\{\hat{\sigma}^2\} < \sigma^2$, biased estimator !

- Recall the synthetic example: $t_n = -2 + 3x_n + \epsilon_n$, $\epsilon_n \sim N(0, \sigma^2 = 0.5^2)$

  For this example, D = 2 and $N = 20$, $E_{p(\boldsymbol{t}|\boldsymbol{X},\boldsymbol{\omega},\sigma^2)}\{\hat{\sigma}^2\} = 0.5^2\left(1 - \frac{2}{20}\right) = 0.225$

Summary

- Assume the errors between the data and the proposed deterministic model are noise and Gaussian distributed, the least squares solution from previous lecture is equivalent to the solution obtained by maximizing the likelihood.

- The benefit of the likelihood approach is the ability to quantify the uncertainty in our parameter estimates and hence also, crucially, in our predictions. This allows us to move away from exact predictions to ranges of values (e.g. $t_{\text{new}} \pm \sigma^2_{\text{new}}$).

- Theoretical properties of the maximum likelihood parameter values show that estimate $\hat{\boldsymbol{\omega}}$ is unbiased, but $\hat{\sigma}^2$ is, on average, biased to be too low.

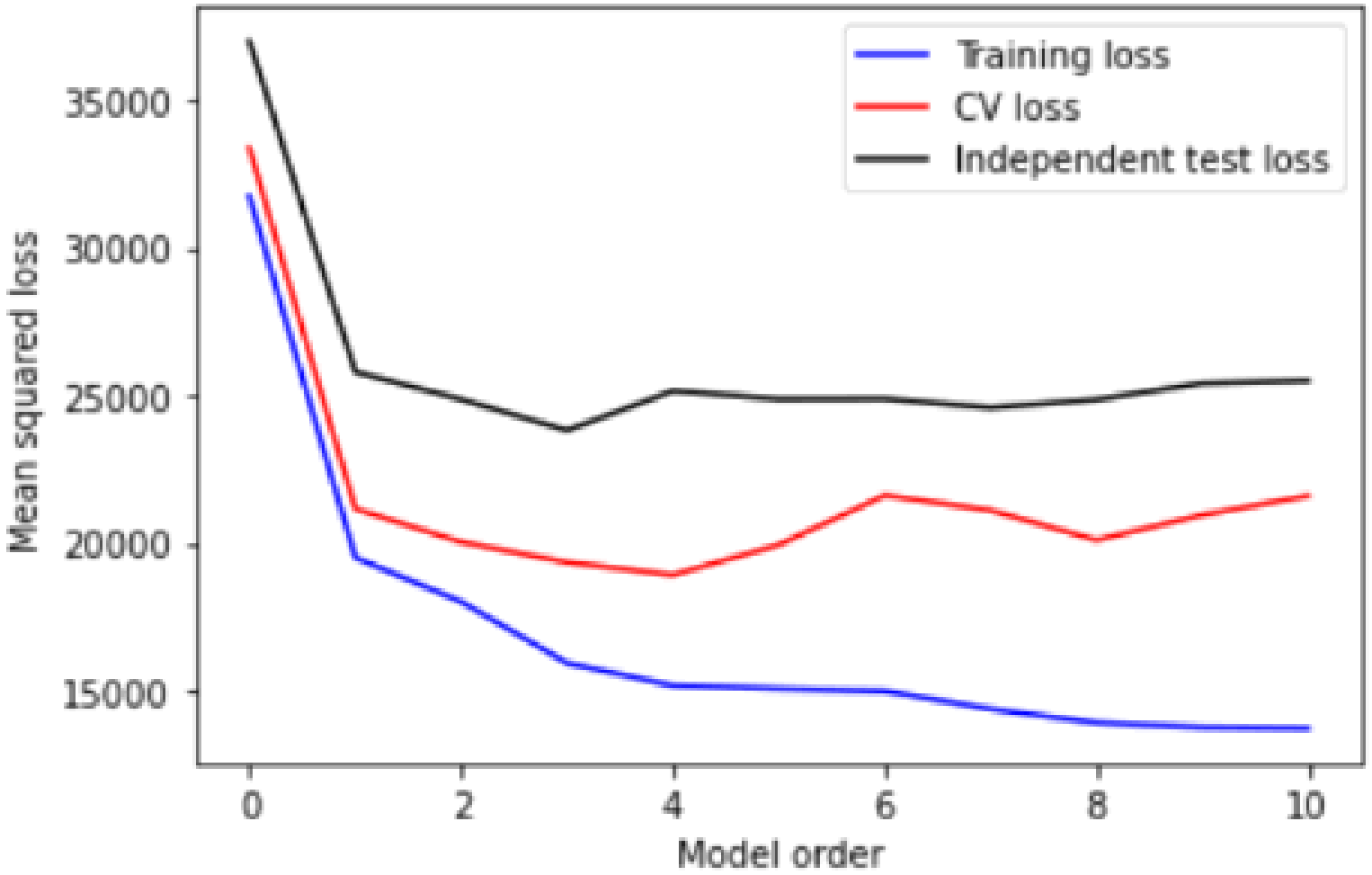
Homework #7

- Perform a 10-fold cross-validation to choose the optimal polynomial order for the synthetic data generated as follows:

```
import numpy as np
N = 70
np.random.seed(2022)
x = 10*np.random.rand(N,1) - 5
t = 5*x**3 - x**2 + x + 150*np.random.randn(N,1)
N_test = 200
x_test = np.linspace(-5,5,N_test)[:,None]
t_test = 5*x_test**3 - x_test**2 + x_test + 150*np.random.randn(N_test,1)
```

1. Note you need to partition the training data (x, t)'s into 10 folds, each fold consists of 7 samples
2. You need to test the polynomial order from 0 (only bias term) to 10 using the testing data
3. Plot the mean-squared training loss, CV loss, and independent test loss versus polynomial order

Homework #7



Deadline of Homework #7: 2022/11/14 3:30pm

$$\boldsymbol{X} = \begin{bmatrix} \boldsymbol{x}_1^T \\ \boldsymbol{x}_2^T \\ \vdots \\ \boldsymbol{x}_N^T \end{bmatrix} = \begin{bmatrix} 1 & x_1^1 & x_1^2 & \cdots & x_1^K \\ 1 & x_2^1 & x_2^2 & \cdots & x_2^K \\ \vdots & \vdots & \vdots & \cdots & \vdots \\ 1 & x_N^1 & x_N^2 & \cdots & x_N^K \end{bmatrix}, \boldsymbol{Y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix}, \hat{\boldsymbol{\omega}} = (\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{Y}$$