

FIT5201 - Data analysis algorithms

Module 2: Linear Models for Regression

Part A:

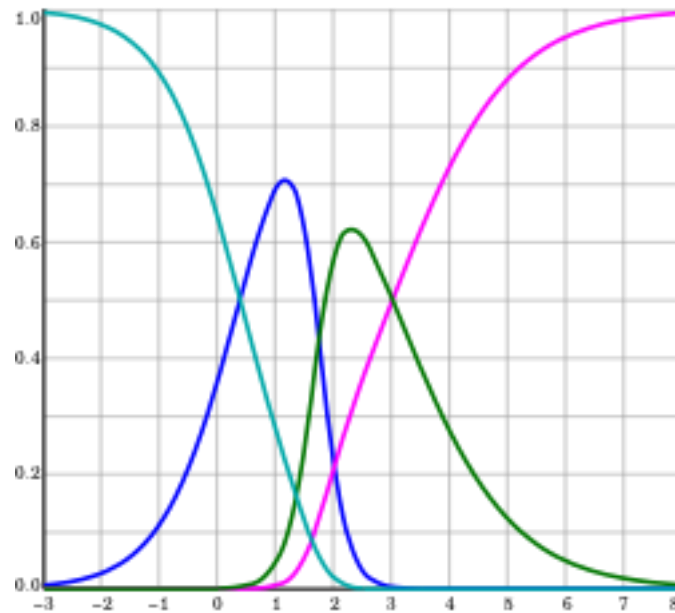
Provide a deep understanding of Linear Regression Models

- **Basis Function Models**
- **Learning Model Parameters**
- **Regularisation**



Part A

Basis Functions for Regression



Basis Functions

□ Regression

- Aim: predict the value of **continuous** target variables **t**

□ Linear Regression

- Regression problem
- Being linear to the **parameters** to be learned

Basis Functions

One of the simplest parametric approach is linear regression.

To predict the underlying function: $\sin(2\pi x)$, we used several polynomial functions:

$$y(\mathbf{x}, \mathbf{w}) = w_0 + w_1 x + w_2 x^2 + \dots + w_M x^M$$

1. $y(\mathbf{x}, \mathbf{w})$ has to be a linear function of the parameters \mathbf{w}
2. $y(\mathbf{x}, \mathbf{w})$ does not have to be a linear function of the variable x (represent the data)

$$y(x, \mathbf{w}) = w_0 + w_1 \phi_1(x) + w_2 \phi_2(x) + \dots + w_{M-1} \phi_{M-1}(x)$$

where $\phi_j(x)$ are called the basis functions

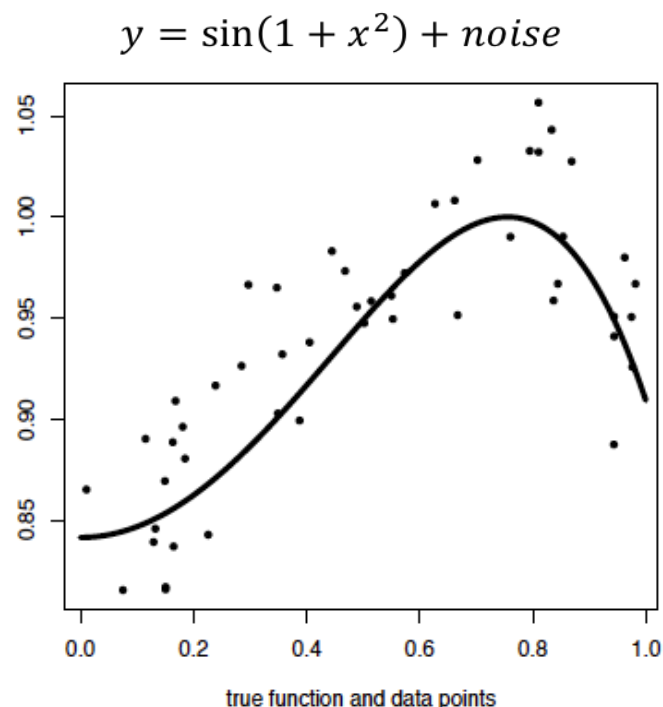
- If $\phi_j(x)$ is linear of x , it's called linear basis function
- Otherwise, it's called as non-linear basis function

Motivation for Basis Functions

We build the model being **linear of the parameters** to **simplify** the optimization process.

We **don't expect** the best predictor for t to be a **linear function** of x .

So, we need to **allow for a non-linear function of x** , but we don't have any theory that says what form this function should take.



Basis Functions

Consider the following function:

$$y(\mathbf{x}, \mathbf{w}) = w_0 + w_1x + w_2x^2 + \dots + w_M x^M = w_0 + \sum_{j=1}^M w_j x^j$$

Basis functions provide a way of making a linear regression model nonlinear. In general, we define $\phi(\mathbf{x})$ where $\phi(\cdot)$ is a nonlinear function of \mathbf{x} . This give the following prediction model:

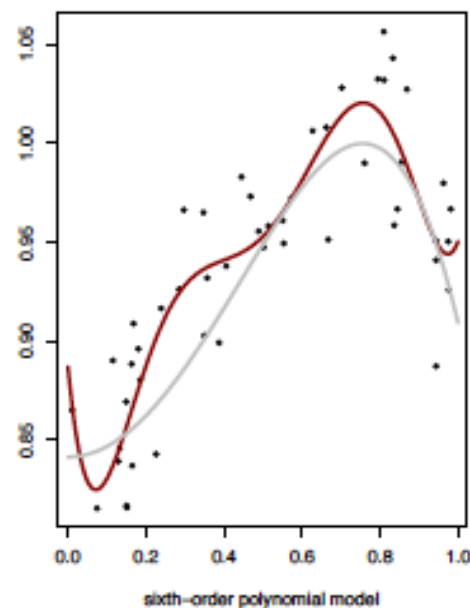
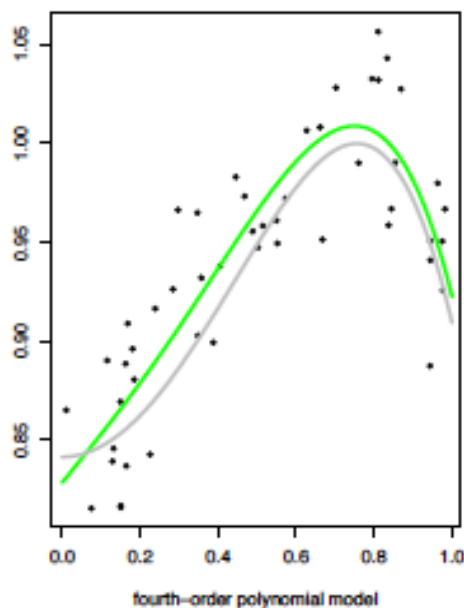
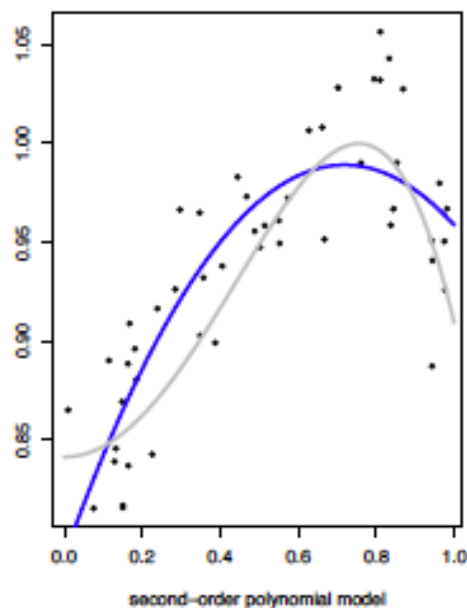
$$y(x, \mathbf{w}) = w_0 + \sum_{j=1}^{M-1} w_j \phi_j(x) = \sum_{j=0}^{M-1} w_j \phi_j(x)$$

where $\phi_j(\mathbf{x})$ is known as a **basis function**.

Example Regression Models

Regression curves for y having the form of

$$y(\mathbf{x}, \mathbf{w}) = w_0 + w_1 \phi_1(x) + w_2 \phi_2(x) + \cdots + w_M \phi_M(x)$$



These polynomial models with increasing order can be viewed as basis function models with $\phi_j(x) = x^j$:

Popular Basis Functions

□ Gaussian basis functions

$$\phi_j(x) := \exp\left(-\frac{(x-\mu_j)^2}{2s^2}\right)$$

□ Sigmoidal basis function

$$\phi_j(x) := \sigma\left(\frac{x-\mu_j}{s}\right)$$

$$\sigma(a) := \frac{1}{1+\exp(-a)} \text{ logistic sigmoid function}$$

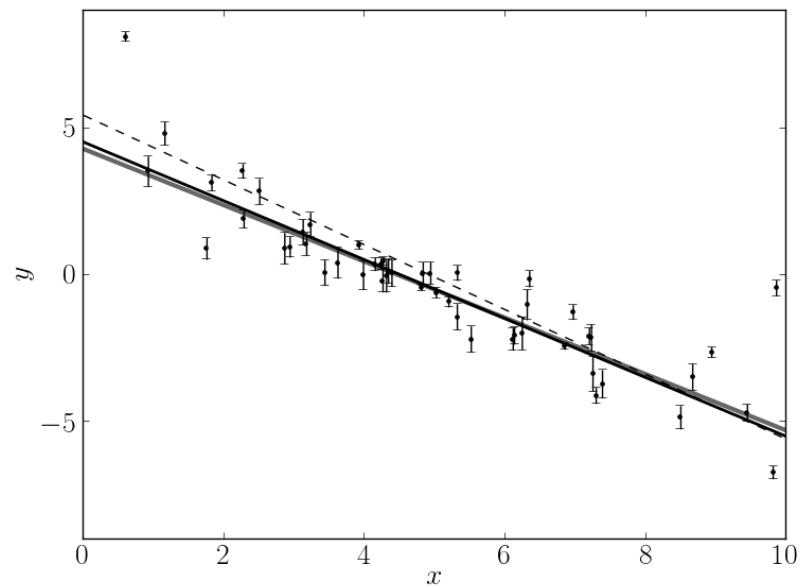
□ Hyperbolic tangent function

$$\tanh(a) = \frac{1-e^{-2a}}{1+e^{-2a}}$$

□ Why these functions? How to choose?

Part A

Learning Model Parameters



Loss Function

Recall how do we fit a regression model to the training data?

By minimising an error (or cost) function that measures the misfit between the predictions $y(\mathbf{x}, \mathbf{w})$ for each data training point x_n and the target value t_n .

How did we calculate the misfit?

The misfit is calculated using "the sum of the squares of the errors"

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

What is the objective of linear regression?

The objective of linear regression is to minimise the cost (i.e. error) function: Find \mathbf{w} that satisfies this function $\mathbf{w} := \arg \min_{\mathbf{w}} E(\mathbf{w})$

Loss Function

- ❑ Sum of squares of errors: Rigorous choice
- ❑ How? Maximum likelihood solution under Gaussian noise assumption
- ❑ Data $D = \{(x_n, t_n)\}_{n=1}^N$ generation
 - Function: $y(x, \mathbf{w})$
 - Noise: $\epsilon = \mathcal{N}(0, \sigma^2)$
 - $t_n = y(x, \mathbf{w}) + \epsilon$
 - $p(\mathbf{t}|\mathbf{x}, \mathbf{w}, \sigma^2) = \mathcal{N}(\mathbf{t}|y(\mathbf{x}, \mathbf{w}), \sigma^2)$

Loss Function

- ❑ Assume the training data points are drawn independently from $p(t|\mathbf{x}, \mathbf{w}, \sigma^2) = \mathcal{N}(t|y(\mathbf{x}, \mathbf{w}), \sigma^2)$

- ❑ Likelihood function

$$p(t_1, \dots, t_N | \mathbf{x}_1, \dots, \mathbf{x}_N, \mathbf{w}, \sigma) = \prod_{n=1}^N \mathcal{N}(t_n | y(\mathbf{x}_n, \mathbf{w}), \sigma^2)$$

- ❑ Log-likelihood function

$$\begin{aligned} \mathcal{L}(\mathbf{w}) &:= \log p(t_1, \dots, t_N | \mathbf{x}_1, \dots, \mathbf{x}_N, \mathbf{w}, \sigma) \\ &= \log \prod_{n=1}^N \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{1}{2\sigma^2} [t_n - \mathbf{w} \cdot \phi(\mathbf{x}_n)]^2\right) \\ &= \underbrace{N \log \frac{1}{\sqrt{2\pi}\sigma}}_{\text{constant wrt } \mathbf{w}} - \underbrace{\frac{1}{2\sigma^2} \sum_{n=1}^N [t_n - \mathbf{w} \cdot \phi(\mathbf{x}_n)]^2}_{\text{squared error}} \end{aligned}$$

Loss Function

- ❑ Sum of squares of errors: Rigorous choice
- ❑ How? Maximum likelihood solution under Gaussian noise assumption
- ❑ We know the loss function, then how to find the optimal parameters?
 - ❑ Loss function: $\frac{1}{2} \sum_{n=1}^N [t_n - \mathbf{w} \cdot \phi(\mathbf{x}_n)]^2$
 - ❑ **Any idea?**

Optimal Solutions

- Find the stationary points by setting **all partial derivatives** to 0

- Loss function:

$$\frac{1}{2\sigma^2} \sum_{n=1}^N [t_n - \mathbf{w} \cdot \phi(\mathbf{x}_n)]^2$$

- Setting partial derivatives with respect to **all parameters** to zero:

$$\frac{\partial}{\partial w_i} \mathcal{L}(\mathbf{w}) = \frac{1}{\sigma^2} \sum_{n=1}^N (t_n - \mathbf{w} \cdot \phi(\mathbf{x}_n)) \phi_i(\mathbf{x}_n) = 0 \quad \forall i$$

- The stationary point for every partial derivative is related with all the other parameters
- We need to **solve multiple equations** to get the **optimal solution**

$$\nabla \mathcal{L}(\mathbf{w}) = \begin{bmatrix} \frac{\partial \mathcal{L}(\mathbf{w})}{\partial w_0} \\ \vdots \\ \frac{\partial \mathcal{L}(\mathbf{w})}{\partial w_{M-1}} \end{bmatrix} = \begin{bmatrix} 0 \\ \vdots \\ 0 \end{bmatrix}$$

- Optional: can do this with matrix operations, elegant but might be expensive

$$(\Phi^T \Phi) \mathbf{w} = \Phi^T \mathbf{t} \Rightarrow \mathbf{w} = (\Phi^T \Phi)^{-1} \Phi^T \mathbf{t}$$

Optimal Solutions

□ Find the stationary points by setting **all partial derivatives** to 0

□ Loss function:

$$\frac{1}{2\sigma^2} \sum_{n=1}^N [t_n - \mathbf{w} \cdot \boldsymbol{\phi}(\mathbf{x}_n)]^2$$

□ Setting partial derivatives with respect to **all parameters** to zero:

$$\frac{\partial}{\partial w_i} \mathcal{L}(\mathbf{w}) = \frac{1}{\sigma^2} \sum_{n=1}^N (t_n - \mathbf{w} \cdot \boldsymbol{\phi}(\mathbf{x}_n)) \phi_i(\mathbf{x}_n) = 0 \quad \forall i$$

- The stationary point for every partial derivative is related with all the other parameters
- We need to solve **multiple equations** to get the **optimal solution**

$$\Phi = \begin{pmatrix} \phi_0(\mathbf{x}_1) & \phi_1(\mathbf{x}_1) & \dots & \phi_{M-1}(\mathbf{x}_1) \\ \phi_0(\mathbf{x}_2) & \phi_1(\mathbf{x}_2) & \dots & \phi_{M-1}(\mathbf{x}_2) \\ \vdots & \vdots & \ddots & \vdots \\ \phi_0(\mathbf{x}_N) & \phi_1(\mathbf{x}_N) & \dots & \phi_{M-1}(\mathbf{x}_N) \end{pmatrix}$$

- Optimal solution: but might be expensive

$$(\Phi^T \Phi) \mathbf{w} = \Phi^T \mathbf{t} \Rightarrow \mathbf{w} = (\Phi^T \Phi)^{-1} \Phi^T \mathbf{t}$$

Iterative optimization algorithms

- ❑ Motivation:

- ❑ Hard or expensive to find the optimal solutions directly

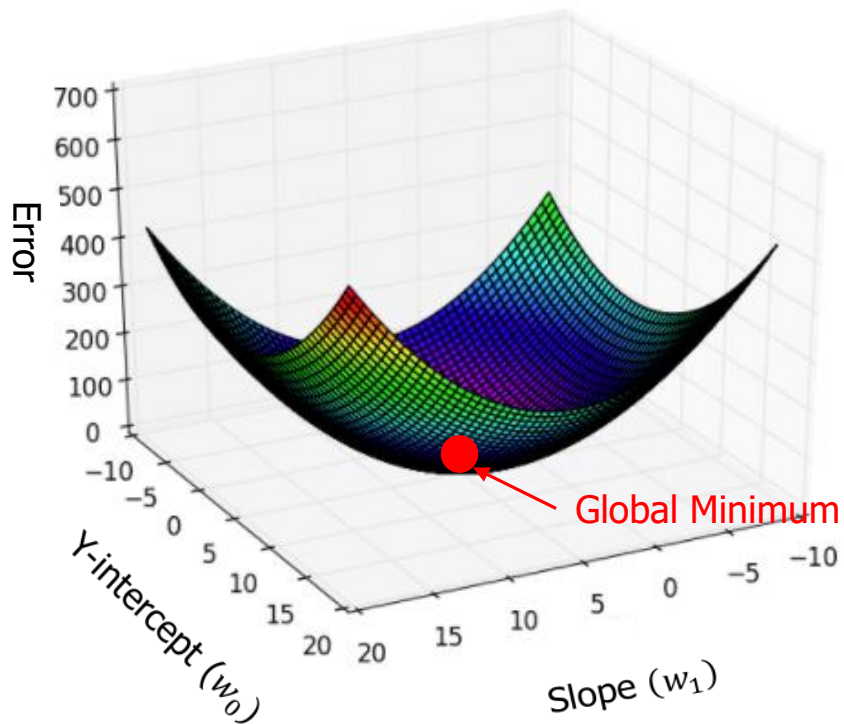
- ❑ Basic idea

- ❑ Initialize the parameters randomly
 - ❑ For each iteration, move towards the negative direction of the function gradient

- ❑ Methodologies

- ❑ Gradient descent algorithms
 - ❑ Stochastic gradient descent algorithms

Iterative optimization algorithms



start from a random location on this surface and move downhill to find the line with the lowest error.

Gradient Descent Algorithm

What is the gradient descent algorithm?

Gradient descent is an algorithm that minimizes functions.

Gradient descent starts with an initial set of parameter values and iteratively moves towards optimal parameters.

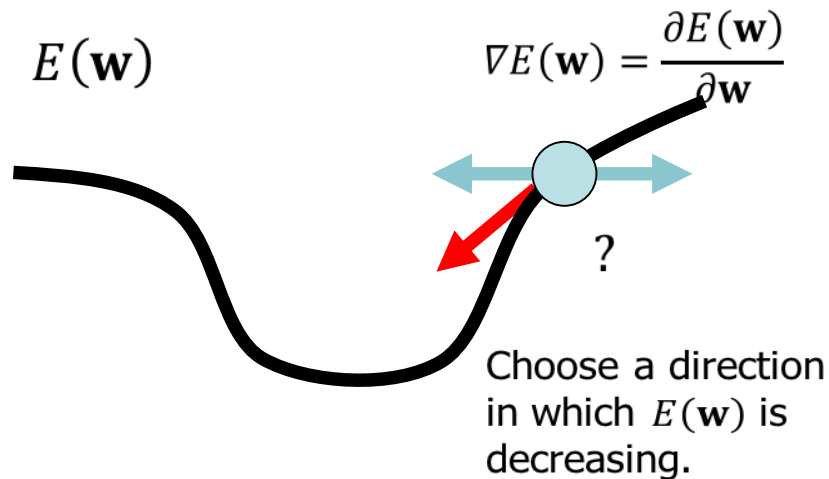
This iterative minimization can be achieved using calculus.

Demonstrate how gradient descent can be used to solve linear regression problems

Gradient Descent Algorithm

What does it mean to move downhill on the error function?

That is, how to change the parameter vector \mathbf{w} to improve $E(\mathbf{w})$?



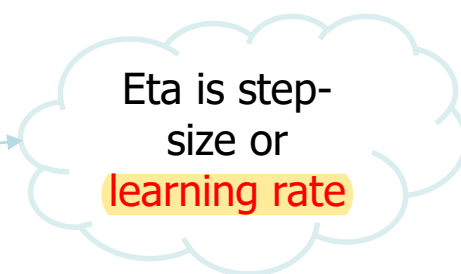
Gradient:

1. a multi-variable generalization of the derivative.
2. Vector-valued function while derivative being scalar-valued
3. Direction means that you will get a largest number towards this direction
4. Left figure is a special case for one variable

Gradient Descent Algorithm

We need to compute the **gradient of the error function**. The gradient will act like a **compass** and always point us **downhill**.

- Initialise the parameters to $\mathbf{w}^{(0)}$ and $t = 1$
- while a stopping condition is *not* met do:
 - $\mathbf{w}^{(t)} := \mathbf{w}^{(t-1)} - \eta \nabla E(\mathbf{w}^{(t-1)})$
 - $t = t + 1$



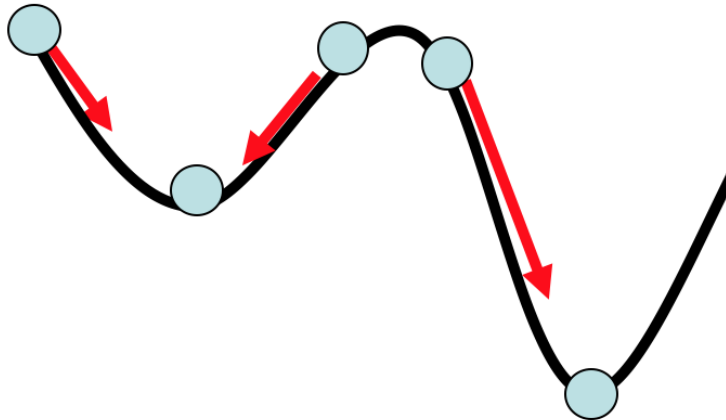
Eta is step-size or
learning rate

Key problems:

1. How to initialize
2. How to get partial derivatives
3. How to decide the learning rate

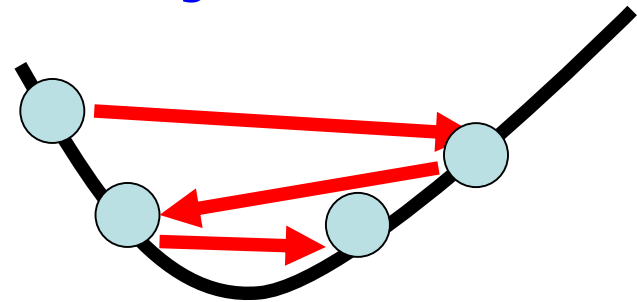
Gradient Descent (GD) Algorithm

- General and useful algorithm
- Sensitive to starting point: local minima

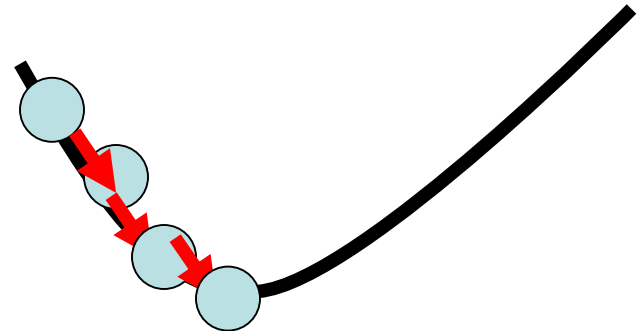


- Learning rate size?

Too large?

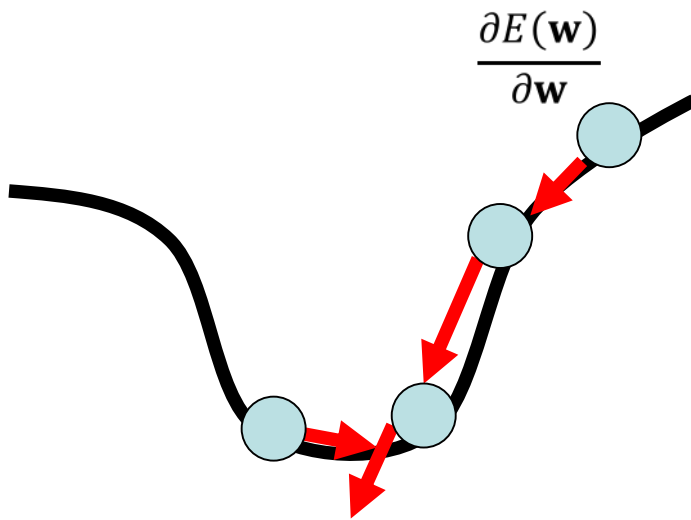


Too small?



Gradient Descent Algorithm

How does the gradient descent algorithm work?



- Initialise the parameters to $\mathbf{w}^{(0)}$ and $t = 1$
- while a stopping condition is not met do:
 - $\eta' = \eta$
 - while $\eta' > \epsilon$ do
 - $\mathbf{w} := \mathbf{w}^{(t-1)} - \eta' \nabla E(\mathbf{w}^{(t-1)})$
 - if $E(\mathbf{w}) < E(\mathbf{w}^{(t-1)})$ then break
 - $\eta' = \eta'/2$
 - $\mathbf{w}^{(t)} := \mathbf{w}$
 - $t = t + 1$

Fundamental Calculus

In order to compute the slopes we need to know some calculus:

Fundamental Calculus:

- o **Derivative** is a measurement of how a function changes when the values of its inputs change
- o **Partial derivative** (denoted by the symbol ∂) of a function of several variables is its derivative with respect to one of those variables with the others held constant.
- o The following is the only things we need to know for linear regression:
 1. **Derivative calculation:**
 - $\frac{\partial}{\partial x} x = 1, \frac{\partial}{\partial x} 2x = 2, \frac{\partial}{\partial x} x^2 = 2x$
 2. **Chain Rule:** calculate the derivative of a composition of two or more functions:
 - $\frac{\partial}{\partial x} g(f(x)) = \frac{\partial}{\partial x} g(f(x)) \frac{\partial}{\partial x} f(x)$
 - e.g: $\frac{\partial}{\partial x} (x^2 + 1)^2 = 2(x^2 + 1)(2x)$

Stochastic Gradient Descent (SGD) Algorithm

□ Motivation

- Batch-based techniques, such as gradient descent algorithm, involve processing the entire training set in one go

In the case of linear regression, the error function to minimise is:

$$E(\mathbf{w}) := \frac{1}{2} \sum_{n=1}^N [t_n - \mathbf{w} \cdot \phi(\mathbf{x}_n)]^2$$

where $\mathcal{D} := \{(\mathbf{x}_n, t_n)\}_{n=1}^N$ is the training data. The gradient of the training objective is:

$$\nabla E(\mathbf{w}) = - \sum_{n=1}^N [t_n - \mathbf{w} \cdot \phi(\mathbf{x}_n)] \phi(\mathbf{x}_n)$$



Why?

Stochastic Gradient Descent (SGD) Algorithm

❑ Motivation

- ❑ Reason: we try to optimize the function to fit to all the training data
 - In other words, we need to minimize the prediction error over all the training data
- ❑ Really computationally costly for large data sets
- ❑ May not have access to all of the training data
- ❑ Not incremental

Stochastic Gradient Descent (SGD) Algorithm

Revisit the GD algorithm:

Initialise \mathbf{w}^0 ;

Do {

$$\mathbf{w} = \mathbf{w}^{(k-1)} - \eta' \nabla E(\mathbf{w}^{(k-1)});$$

} while $(\nabla E(\mathbf{w}) > \epsilon)$

Need to process entire training set
each iteration: computational cost
is very expensive for large datasets

SGD's idea:

$E(\mathbf{w}) = \sum_n E_n(\mathbf{w})$, where
 $E_n(\mathbf{w}) = \frac{1}{2}(y_n - t_n)^2$. Then,
instead of visiting all data
points in the training set,
visiting a single data point in
each iteration.

Stochastic Gradient Descent (SGD) Algorithm

The GD algorithm:

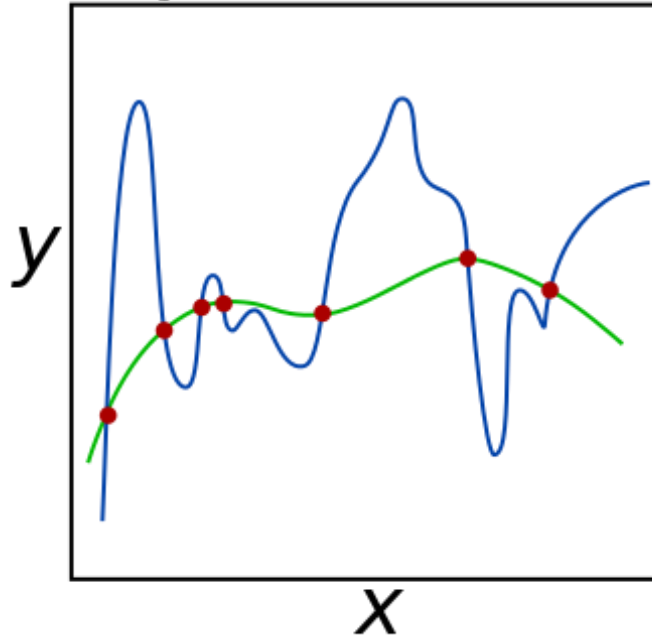
```
Initialise  $\mathbf{w}^0$ ;  
Do {  
     $\mathbf{w} = \mathbf{w}^{(k-1)} - \eta' \nabla E(\mathbf{w}^{(k-1)})$ ;  
} while  $(\nabla E(\mathbf{w}) > \epsilon)$ 
```

The SGD algorithm:

```
Initialise  $\mathbf{w}^0$ ;  
Do {  
    for each training point  $\tau$   
         $\mathbf{w}^\tau = \mathbf{w}^{(\tau-1)} - \eta' \nabla E_n(\mathbf{w}^{(\tau-1)})$ ;  
} while  $(\nabla E(\mathbf{w}) > \epsilon)$ 
```

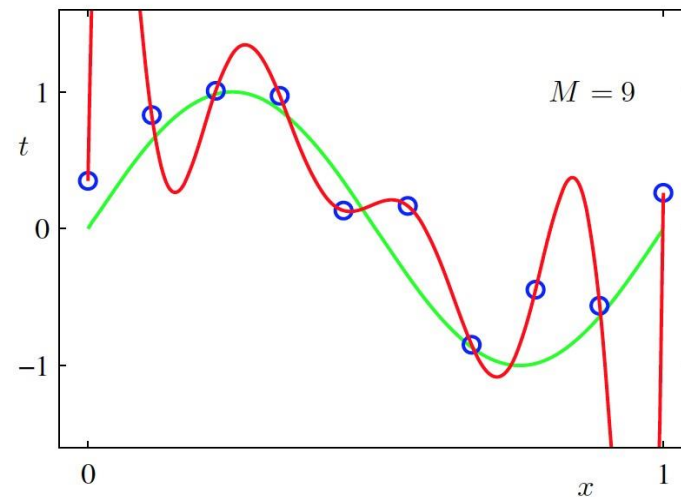
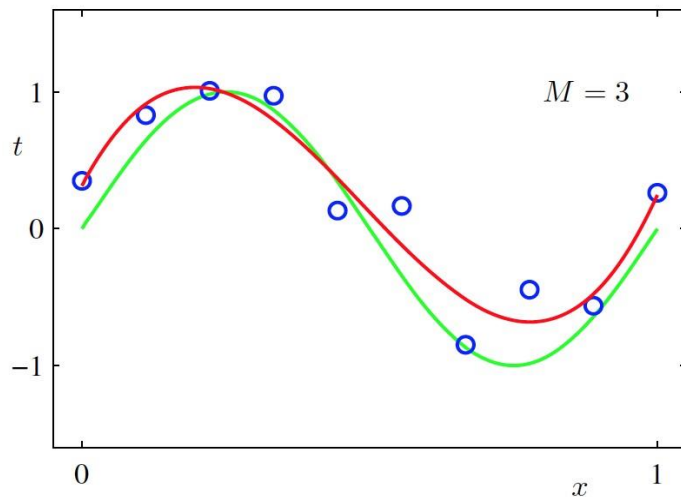
Part A

Regularisation



Regularisation

Remember how to prevent overfitting?
By adding penalties to the values of parameters.



Regularisation

Remember how to prevent overfitting?
By adding penalties to the values of parameters.

	$M = 0$	$M = 1$	$M = 6$	$M = 9$
w_0^*	0.19	0.82	0.31	0.35
w_1^*		-1.27	7.99	232.37
w_2^*			-25.43	-5321.83
w_3^*			17.37	48568.31
w_4^*				-231639.30
w_5^*				640042.26
w_6^*				-1061800.52
w_7^*				1042400.18
w_8^*				-557682.99
w_9^*				125201.43

Regularisation

□ Total error function

$$E_{\mathcal{D}}(\mathbf{w}) + \lambda\Omega(\mathbf{w})$$

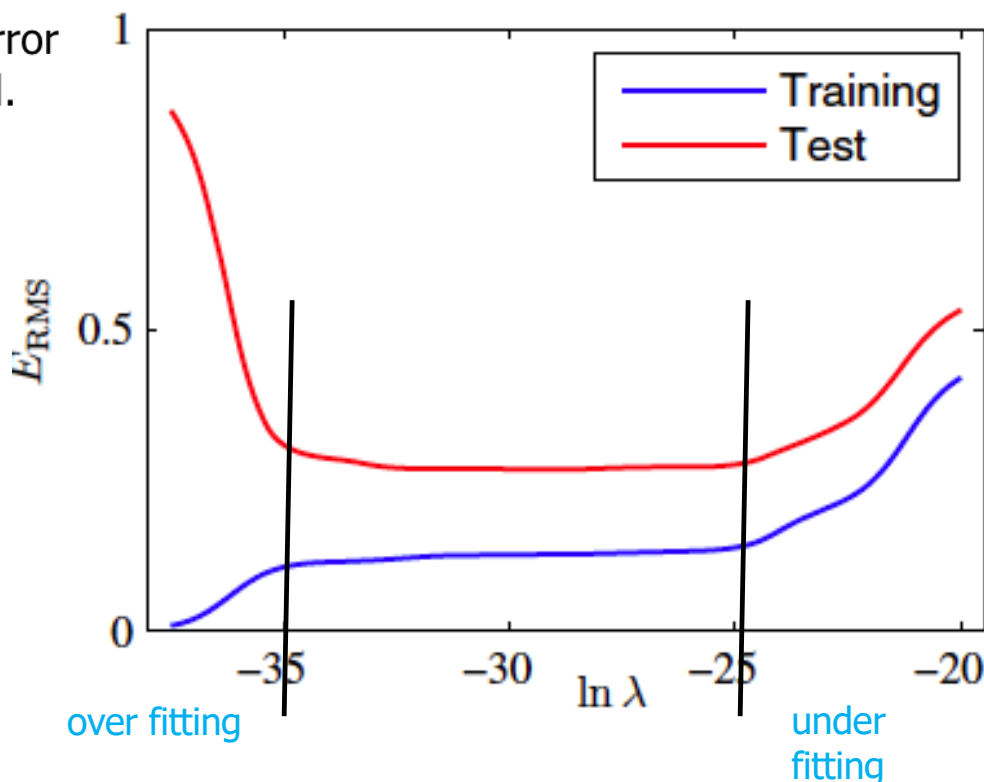
- **How to find a suitable regularization parameter?**
- **How to define a suitable regularization function?**

How does regularisation work?

For a large λ , models with high complexity can be ruled out. For a small λ , models with high training errors can be ruled out. The optimal solution lies somewhere in the middle.

Graph of the root-mean-square error vs. $\ln(\lambda)$ for the $M = 9$ polynomial.

- 1 λ increases, model becomes less complex, training error increases
- 2 λ is very small, model too complex, test error high (over fitting)
- 3 λ increases, model becomes less complex, testing error decreases
- 4 λ is too large, model too simple, testing error increases (under fitting)



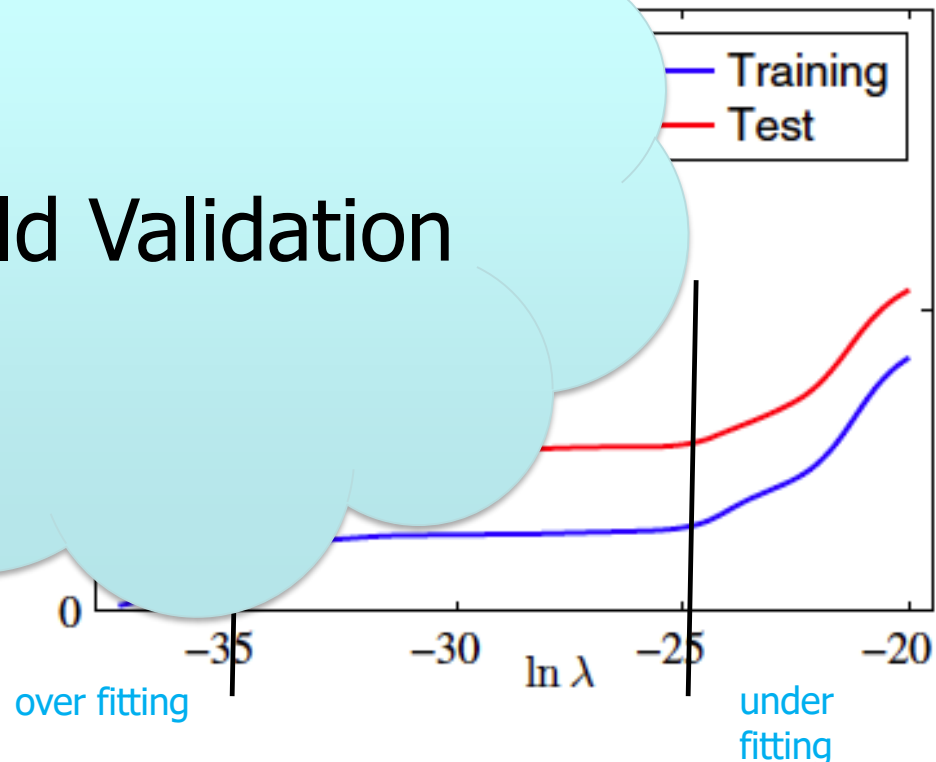
How does regularisation work?

For a large λ , models with high complexity can be ruled out. For a small λ , models with high training errors can be ruled out. The optimal solution lies somewhere in the middle.

Graph of the root-mean-square error vs. $\ln(\lambda)$ for the $M = 9$ polynomial

- 1 λ increases, model becomes less complex, training error increases
- 2 λ is very small, model is very complex, test error increases (over fitting)
- 3 λ increases, model becomes less complex, testing error decreases
- 4 λ is too large, model too simple, testing error increases (under fitting)

K-fold Validation



Regularization function

- Depend on problems

$$\Omega(\mathbf{w}) := \frac{1}{2} \sum_{j=0}^{M-1} |w_j|^q$$

$$\frac{1}{2} \sum_{n=1}^N (t_n - \mathbf{w} \cdot \boldsymbol{\phi}(\mathbf{x}_n))^2 + \frac{\lambda}{2} \sum_{j=0}^{M-1} |w_j|^q$$

- Popular choice

- Ridge Regularization
- Lasso Regularization

Ridge vs Lasso Regularisation

What is Ridge regularisation?

Ridge Regularisation

- o Performs L2 regularisation: i.e. adds penalty equivalent to squares of the magnitude of parameters (i.e. L2-norm, ℓ_2 -norm)

$$\frac{1}{2} \sum_{n=1}^N (t_n - \mathbf{w} \cdot \boldsymbol{\phi}(\mathbf{x}_n))^2 + \frac{\lambda}{2} \sum_{j=0}^{M-1} w_j^2$$

Global Optimal reading: to find the optimal solution, we can use matrix operations

$$\mathbf{w} = (\lambda \mathbf{I} + \boldsymbol{\Phi}^T \boldsymbol{\Phi})^{-1} \boldsymbol{\Phi}^T \mathbf{t}$$

Ridge vs Lasso Regularisation

What is Lasso regularisation?

Lasso Regularisation

- o Performs L1 regularisation: i.e. adds penalty equivalent to absolute value of the magnitude of parameters (i.e. L1-norm, ℓ_1 -norm)

$$\frac{1}{2} \sum_{n=1}^N (t_n - \mathbf{w} \cdot \boldsymbol{\phi}(\mathbf{x}_n))^2 + \frac{\lambda}{2} \sum_{j=0}^{M-1} |w_j|$$

How to optimize? Non-differentiable
Sub-gradient

Key Differences: Ridge & Lasso

What are key differences between Ridge and Lasso regression

- Ridge: The main advantage is parameter shrinkage and reduce the model complexity
- Lasso: Along with shrinking parameters, it selects input variables as well. Some features can be excluded from the model.

When do we have to use Ridge or Lasso?

- Ridge: Mainly used to prevent overfitting, but not useful in case of high # of input variables.
- Lasso: It provides sparse solutions, but useful in case of high # of input variables as those ones with zero parameters can simply be ignored.

L1 : feature selection?

Shoe size	Weight	Height
2	1.8	4
3	3.1	6
...
4	3.9	8

h: height, the target variable

s: shoe size

w: weight

$$h = \alpha_1 s + \alpha_2 w$$



$$\alpha_1 + \alpha_2 = 2$$

$$\alpha_1=1, \alpha_2 = 1 \text{ or } \alpha_1 = 2, \alpha_2 = 0 \text{ or } \alpha_1 = 0, \alpha_2 = 2$$

L1 : feature selection?

Shoe size	Weight	Height
2	1.8	4
3	3.1	6
...
4	3.9	8

h: height, the target variable

s: shoe size

w: weight

$$h = \alpha_1 s + \alpha_2 w$$

$$E = \sum_{i=1}^n (h' - \alpha_1 s - \alpha_2 w)^2 + \lambda (\alpha_1^2 + \alpha_2^2) \xrightarrow{\text{dashed arrow}} \alpha_1 = 1, \alpha_2 = 1$$

$$E = \sum_{i=1}^n (h' - \alpha_1 s - \alpha_2 w)^2 + \lambda (|\alpha_1| + |\alpha_2|)$$



$$\alpha_1 = 2, \alpha_2 = 0 \text{ or } \alpha_1 = 0, \alpha_2 = 2$$

Conclusion

How would you be able to summarise what we have learned?

- ☐ Basis functions
- ☐ Linear regression with Gradient Descent (GD) algorithm
- ☐ Linear regression with Stochastic GD (SGD) algorithm
- ☐ Ridge regularisation algorithm
- ☐ Lasso regularisation algorithm

Tutorial (Week 3)

Run and understand how linear regression and regularisation work in R

What will we learn in Week 4

☐ **Part B in Module 2**

- ☐ What is Bias-Variance Analysis?
- ☐ How can we use this analysis to improve effectiveness of regression?