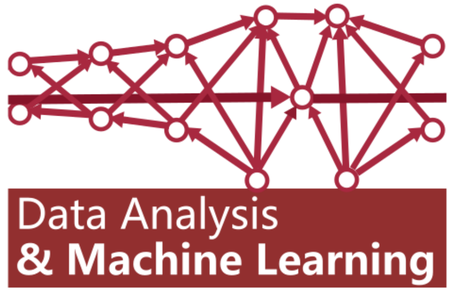


Data Analysis and Machine Learning 4

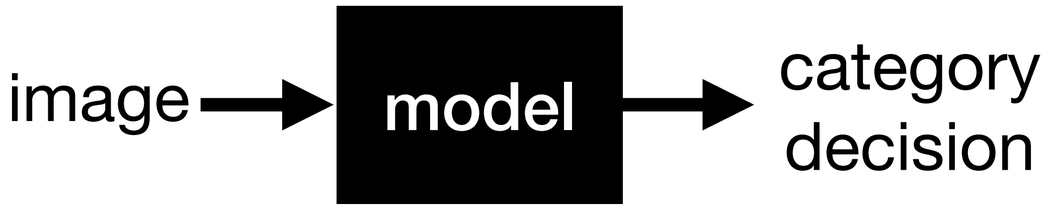
Week 5: Linear Regression

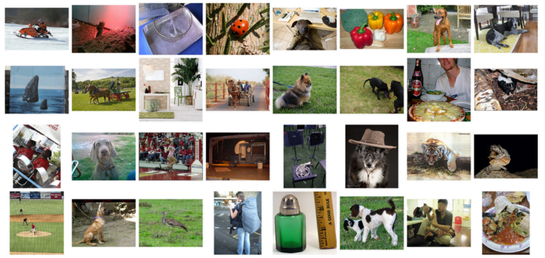
Elliot J. Crowley, 13th February 2023

THE UNIVERSITY *of* EDINBURGH

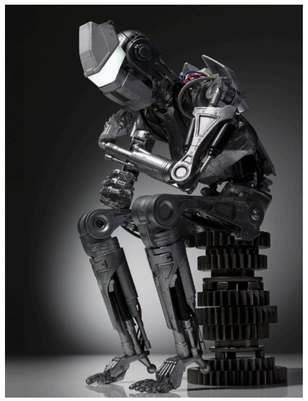
Recap

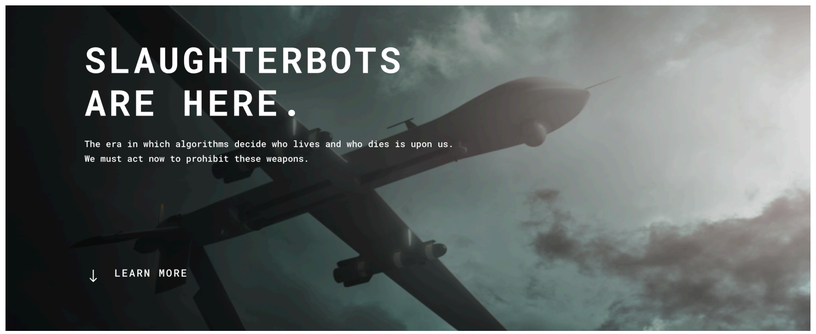
- We learned about supervised learning and looked at some examples

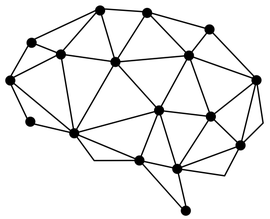




- We considered ethical issues that can arise when applying ML in society



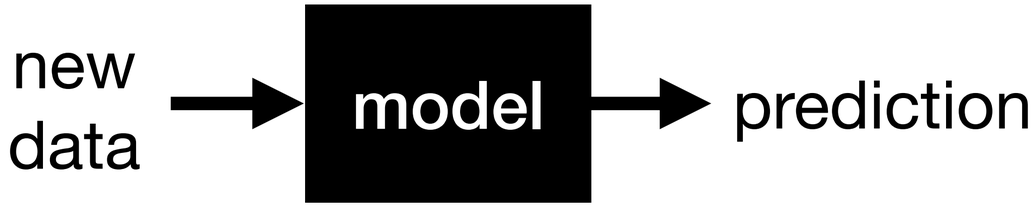




Cambridge
Analytica

Supervised Learning

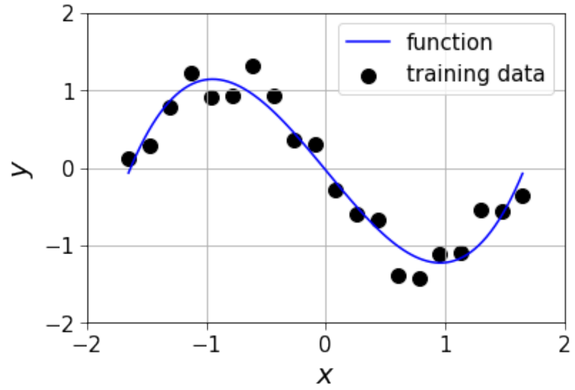
- We want a model that takes in a new data point and outputs a prediction



- For the model to be accurate it must first learn from training data
- Often, models are parameterised functions and learning = finding the best parameters
- Training data is a set of existing data points that have been **labelled**
- The label says what the prediction for that data point **should be**

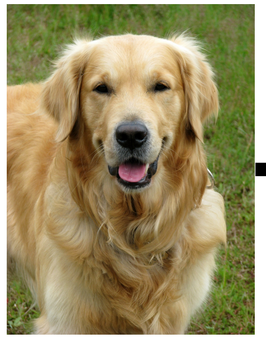
Two canonical problems in supervised learning

- **Regression**: Given input data, predict a continuous output



- **Classification**: Given input data, predict a distinct category

cat



dog

Linear Regression

The regression problem

- Our training set consists of N data point-target pairs $\{\mathbf{x}^{(n)}, y^{(n)}\}_{n=0}^{N-1}$
- Data points $\mathbf{x} \in \mathbb{R}^D$ are column vectors, targets (/labels) $y \in \mathbb{R}^1$ are scalar
- We can use matrix/vector notation as in Week 3

$$\mathbf{X} = \begin{bmatrix} \mathbf{x}^{(0)\top} \\ \mathbf{x}^{(1)\top} \\ \mathbf{x}^{(2)\top} \\ \vdots \\ \mathbf{x}^{(N-1)\top} \end{bmatrix} = \begin{bmatrix} x_0^{(0)} & x_1^{(0)} & \cdots & x_{D-1}^{(0)} \\ x_0^{(1)} & x_1^{(1)} & \cdots & x_{D-1}^{(1)} \\ x_0^{(2)} & x_1^{(2)} & \cdots & x_{D-1}^{(2)} \\ \cdots & \cdots & \ddots & \vdots \\ x_0^{(N-1)} & x_1^{(N-1)} & \cdots & x_{D-1}^{(N-1)} \end{bmatrix} \qquad \mathbf{y} = \begin{bmatrix} y^{(0)} \\ y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(N-1)} \end{bmatrix}$$

- **Objective:** We want some function f such that $f(\mathbf{x}^{(n)}) = y^{(n)}$ for each training point. This function is our regression model

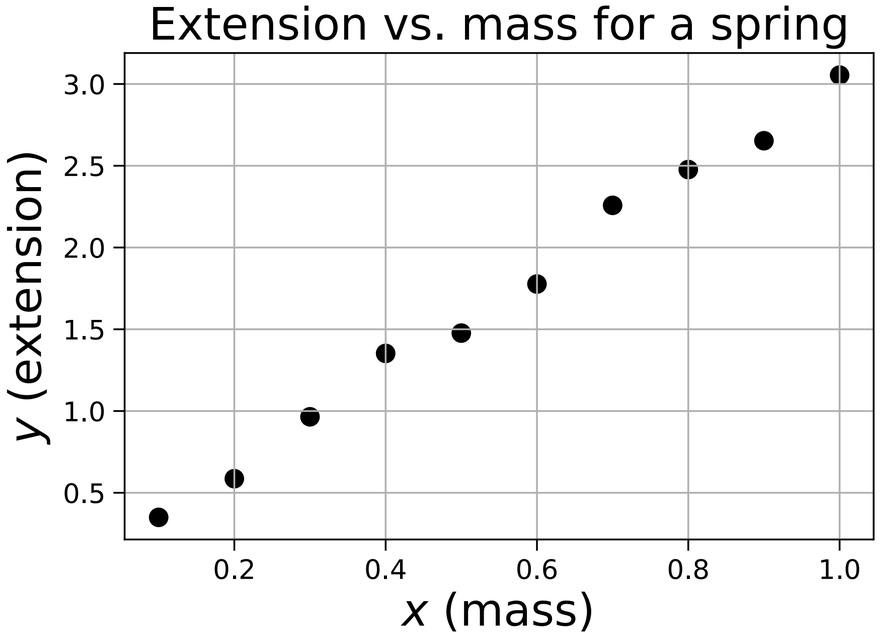
Simple linear regression

- We have 1D measurements of mass-extension pairs $\{x^{(n)}, y^{(n)}\}_{n=0}^{N-1}$
- We want a regression model represented by f s.t. $f(x^{(n)}) = y^{(n)}$ for each point
- Let's use a function that is linear in x and denote its outputs as $\hat{y}$

$$f(x) = \hat{y} = wx + b$$

w and b are the parameters of the model

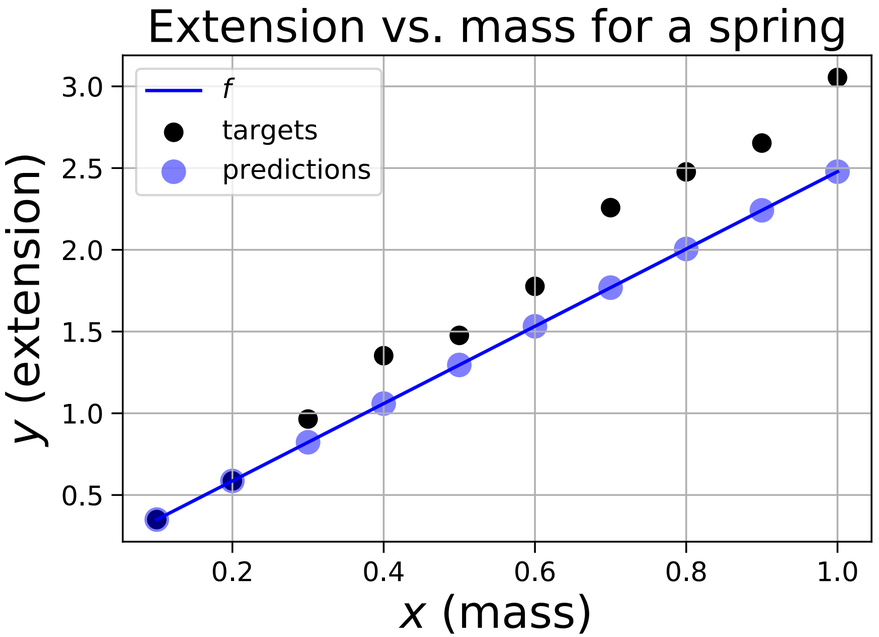
w is called the weight and b is called the bias



Our function predicts the targets

- $\hat{y}^{(0)}, \hat{y}^{(1)}, \ldots, \hat{y}^{(N-1)}$ are **predictions** of our **targets** $y^{(0)}, y^{(1)}, \ldots, y^{(N-1)}$
- We wanted a model f such that $\hat{y}^{(n)} = y^{(n)}$ for each point
- But we can't achieve this: a line can't perfectly fit the data here
- Can we relax our objective?

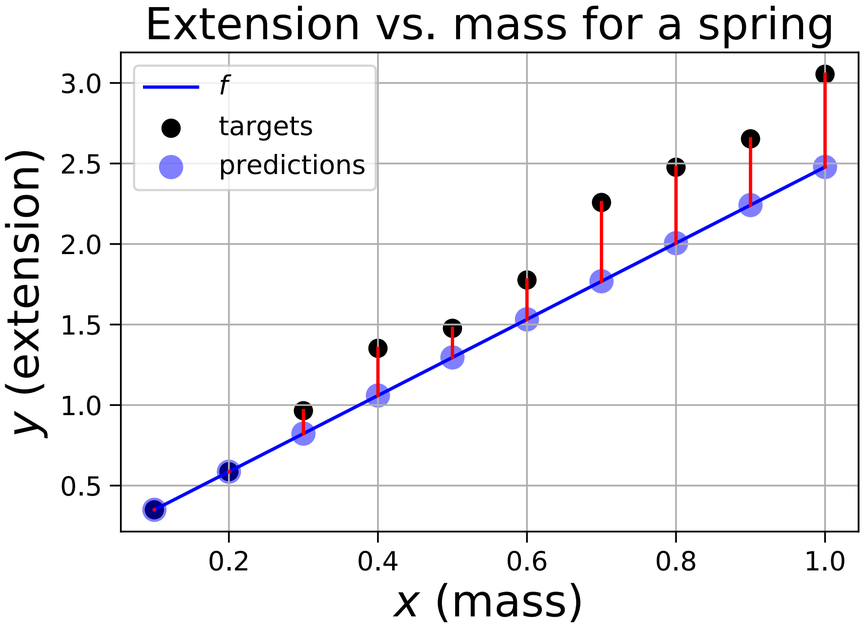
$$f(x) = \hat{y} = wx + b$$



The squared error loss function

- Let's instead minimise the square distance between every $\hat{y}^{(n)}$ and $y^{(n)}$: $(y^{(n)} - \hat{y}^{(n)})^2$
- In ML, given an objective, we typically construct a loss function
- This is a function of the model parameters and the data

$$L_{SE} = \sum_{n} (y^{(n)} - \hat{y}^{(n)})^2$$



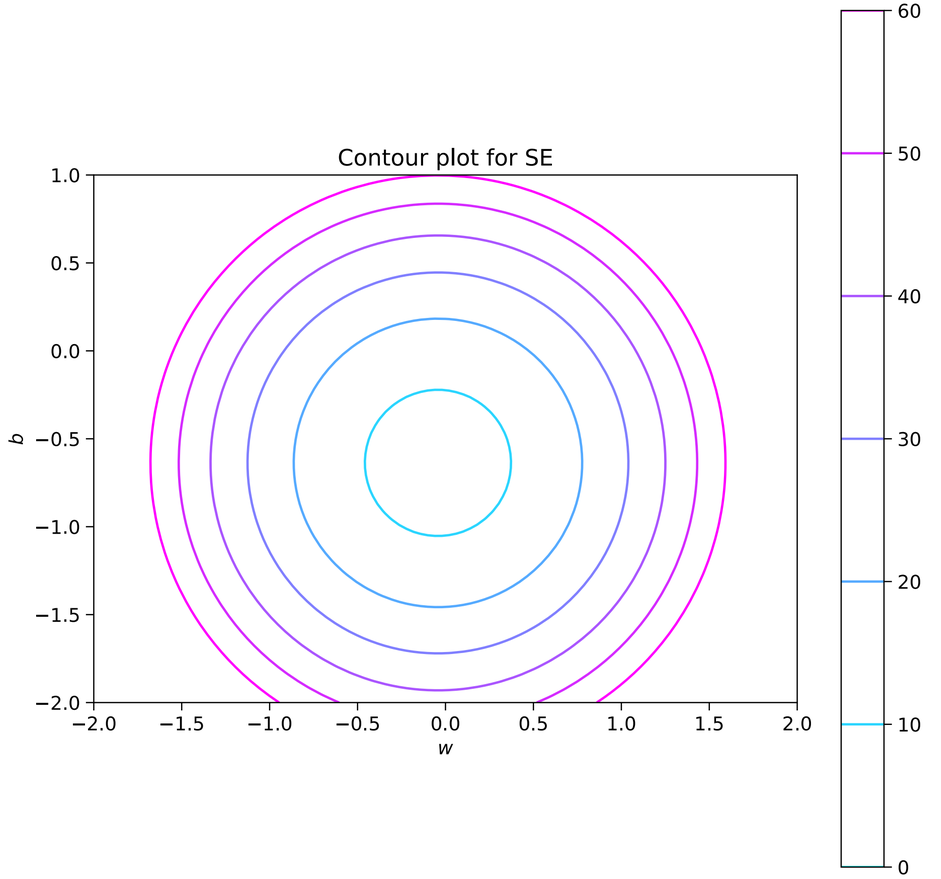
Our objective is achieved when the loss Function is minimised

Minimising squared error

- We can plug $\hat{y} = wx + b$ into the SE equation and assume fixed training data

$$L_{SE}(w, b) = \sum_n (y^{(n)} - wx^{(n)} - b)^2$$

- We want the w and b that minimise SE
- These occur when $\nabla L_{SE} = \begin{bmatrix} \frac{\partial L_{SE}}{\partial w} \\ \frac{\partial L_{SE}}{\partial b} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$



This function is **convex:** it only has one extremum which is a minimum

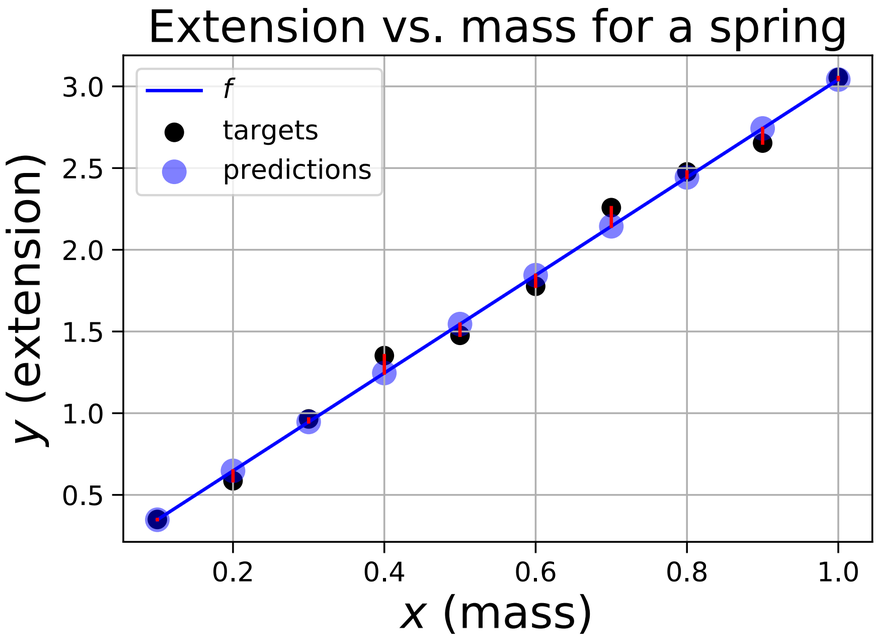
A line of best fit

After some fairly tedious algebra, we get

$$w = \frac{\frac{1}{N}\Sigma_n x^{(n)} y^{(n)} - \bar{x}\bar{y}}{\frac{1}{N}\Sigma_n x^{(n)2} - \bar{x}^2} \qquad b = \bar{y} - w\bar{x}$$

Where $\bar{x} = \frac{1}{N}\sum_n x^{(n)}$ and $\bar{y} = \frac{1}{N}\sum_n y^{(n)}$

Plug in the training data and we get a line that minimises the distances between target and predictions



Multiple linear regression

- We just performed simple linear regression, mapping $\mathbb{R}^1 \rightarrow \mathbb{R}^1$
- Multiple linear regression maps $\mathbb{R}^{D>1} \rightarrow \mathbb{R}^1$
- Let's predict petal width from the other three measurements in the iris dataset

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa
...	...	...	...	...	...
145	6.7	3.0	5.2	2.3	virginica
146	6.3	2.5	5.0	1.9	virginica
147	6.5	3.0	5.2	2.0	virginica
148	6.2	3.4	5.4	2.3	virginica
149	5.9	3.0	5.1	1.8	virginica

$$\mathbf{x} = [x_0 \quad x_1 \quad x_2]^\top$$

Sepal length Sepal width Petal length

y ← Petal width

Writing our function as a dot product

- Our function has a weight for each variable and an intercept (bias)

$$f(\mathbf{x}) = \hat{y} = w_0 x_0 + w_1 x_1 + w_2 x_2 + b$$

- We can write this function as a dot product of vectors by:

1. Writing $\mathbf{w} = \begin{bmatrix} b & w_0 & w_1 & w_2 \end{bmatrix}^\top$
2. Defining $\phi(\mathbf{x}) = \begin{bmatrix} 1 & \mathbf{x}^\top \end{bmatrix}^\top = \begin{bmatrix} 1 & x_0 & x_1 & x_2 \end{bmatrix}^\top$

$$f(\mathbf{x}) = \hat{y} = \mathbf{w}^\top \phi(\mathbf{x})$$

Minimising squared error

- We want to find $\mathbf{w}$ that minimises SE

$$L_{SE}(\mathbf{w}) = \sum_n (y^{(n)} - \mathbf{w}^\top \boldsymbol{\phi}(\mathbf{x}^{(n)}))^2$$

- We can express this loss as a vector norm with some rewriting:

$$\mathbf{y} = \begin{bmatrix} y^{(0)} \\ y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(N-1)} \end{bmatrix} \quad \boldsymbol{\Phi} = \begin{bmatrix} \boldsymbol{\phi}(\mathbf{x}^{(0)})^\top \\ \boldsymbol{\phi}(\mathbf{x}^{(1)})^\top \\ \boldsymbol{\phi}(\mathbf{x}^{(2)})^\top \\ \vdots \\ \boldsymbol{\phi}(\mathbf{x}^{(N-1)})^\top) \end{bmatrix} \qquad L_{SE}(\mathbf{w}) = \|\mathbf{y} - \boldsymbol{\Phi}\mathbf{w}\|^2$$

Vector calculus to the rescue

$$L_{SE}(\mathbf{w}) = \|\mathbf{y} - \mathbf{\Phi w}\|^2 = (\mathbf{y} - \mathbf{\Phi w})^\top(\mathbf{y} - \mathbf{\Phi w})$$

- Take the gradient and set to zero to get minimum

This function is **convex:** it only has one extremum which is a minimum

$$\nabla_{\mathbf{w}} L_{MSE} = -2\mathbf{\Phi}^\top(\mathbf{y} - \mathbf{\Phi w}) = 0$$

- And rearrange

$$\mathbf{w} = (\mathbf{\Phi}^\top\mathbf{\Phi})^{-1}\mathbf{\Phi}^\top\mathbf{y}$$

You are not required to do any vector or matrix calculus by hand on this course. https://www.math.uwaterloo.ca/~hwolkowi/matrixcookbook.pdf is a useful reference for this however.

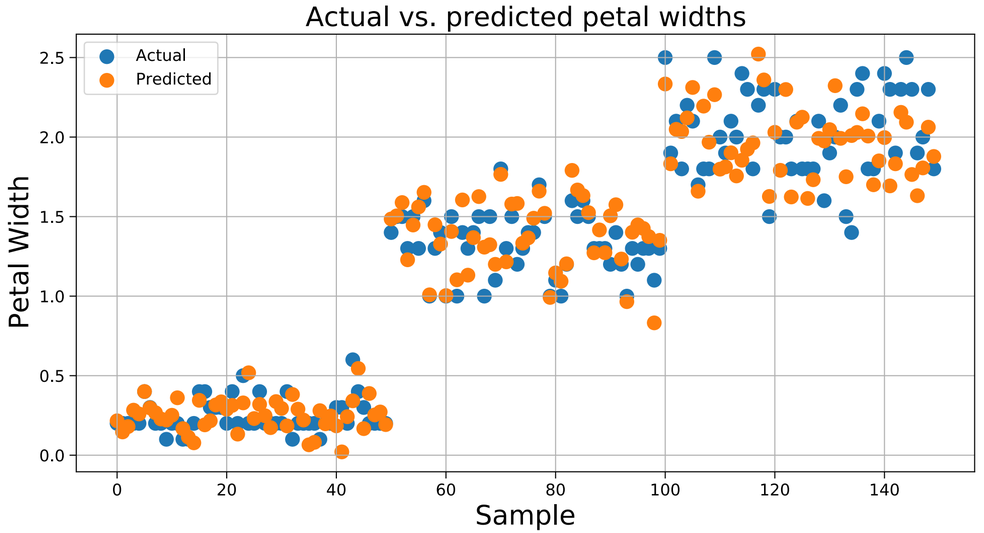
How do we evaluate the model?

- Can compute the mean SE (MSE): 0.03586 (small which is good)
- Can compute R^2: 0.93785 (high which is good — can be 1 at most)
- Can compare predicted petal widths $\hat{y} = \mathbf{w}^\top \phi(\mathbf{x})$ to actual petal widths y
- But we want to apply our models to **new data**…

$$R^2 = 1 - \frac{\sum_n (y^{(n)} - \hat{y}^{(n)})^2}{\sum_n (y^{(n)} - \bar{y})^2}$$

Mean y

R^2 is the fraction of the variance of y that is explained by the model

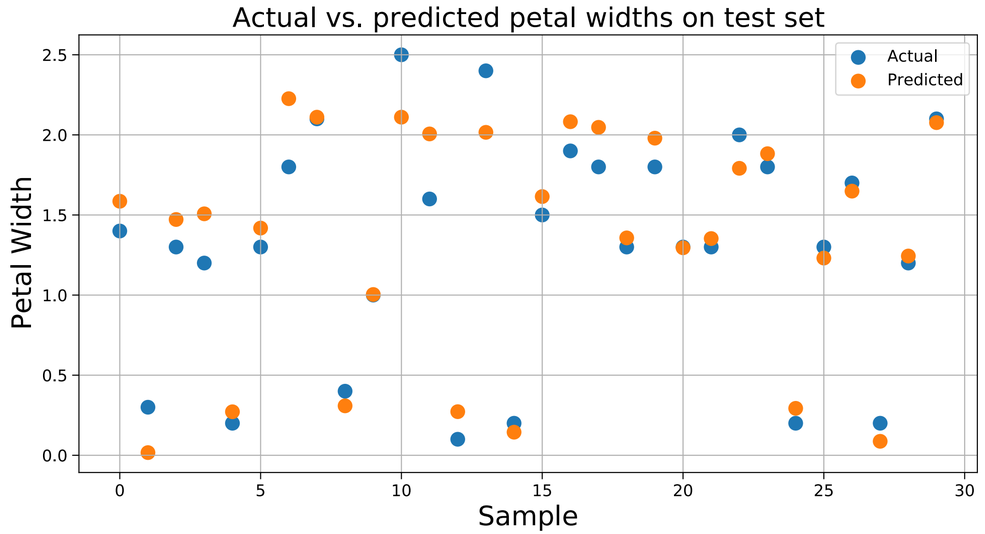


Test set

- We ultimately want our models to do well on **new data**
- Models should be evaluated on data that **wasn't used for training**
- **Solution:** Evaluate model on a test set (can split dataset into train/test)
- A model that can perform well on test is able to **generalise**
- **The test set must never be used to fit the model**

Evaluation

- Let's split the iris dataset into 80% training and 20% test at random
- Learn weights on train, apply to test
- Train MSE: 0.03536 and Test MSE: 0.03906
- Train R^2: 0.9409 and Test R^2: 0.9179



How do we interpret the model?

$$\mathbf{w} = \begin{bmatrix} b \\ w_0 \\ w_1 \\ w_2 \end{bmatrix} = \begin{bmatrix} -0.32 \\ -0.18 \\ 0.21 \\ 0.52 \end{bmatrix}$$

$$\hat{y} = -0.18x_0 + 0.21x_1 + 0.52x_2 - 0.32$$

Petal length — Sepal length — Sepal width — Petal length

- With linear models, the weights tell you the contribution of each variable to the prediction
- **But this isn't simple to interpret if the data isn't standardised**

Variables have their own scales!

Standardised results

- We compute the variable means and standard deviations **on the training set**
- Then apply these **to the training set and the test set!**
- The learnt weights are now simple to interpret

$$\hat{y} = -0.15x_0 + 0.09x_1 + 0.92x_2 + 1.17$$

Petal length

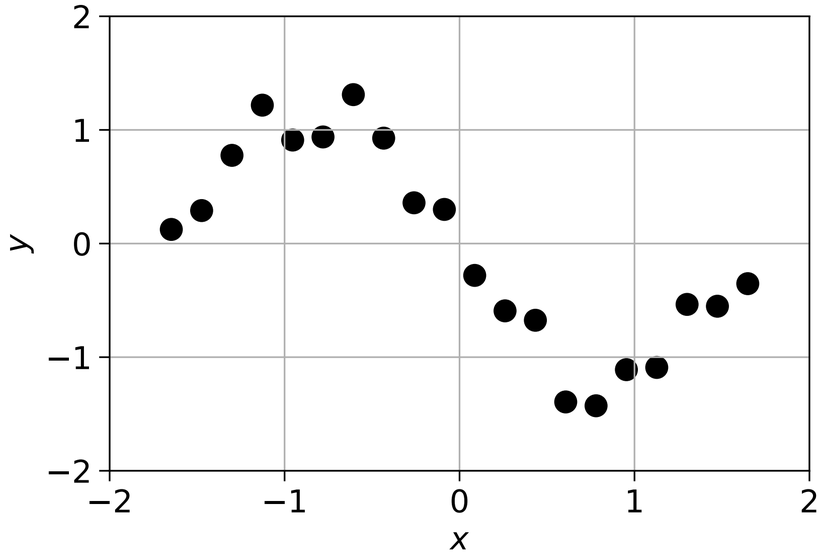
Standardised Sepal length

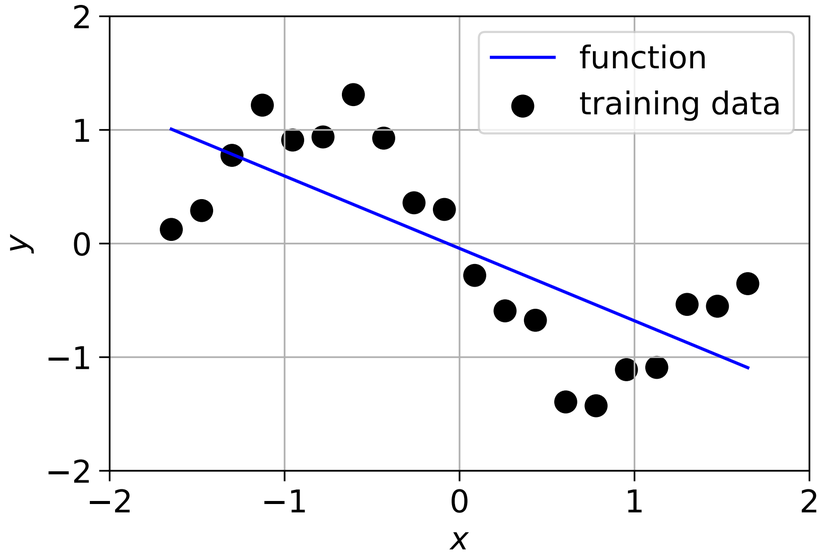
Standardised Sepal width

Standardised Petal length

Polynomial regression

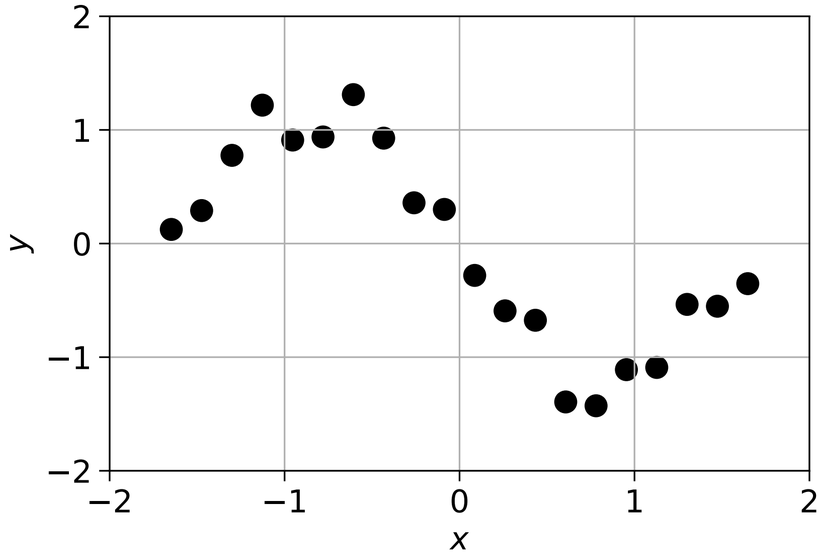
- Consider the 1D training set of data-target pairs below $\{x^{(n)}, y^{(n)}\}_{n=0}^{N-1}$
- The relationship between data and targets is curvilinear
- Simple linear regression produces a model that **underfits** to the data
- The model doesn't have the **capacity** to capture the way the data varies

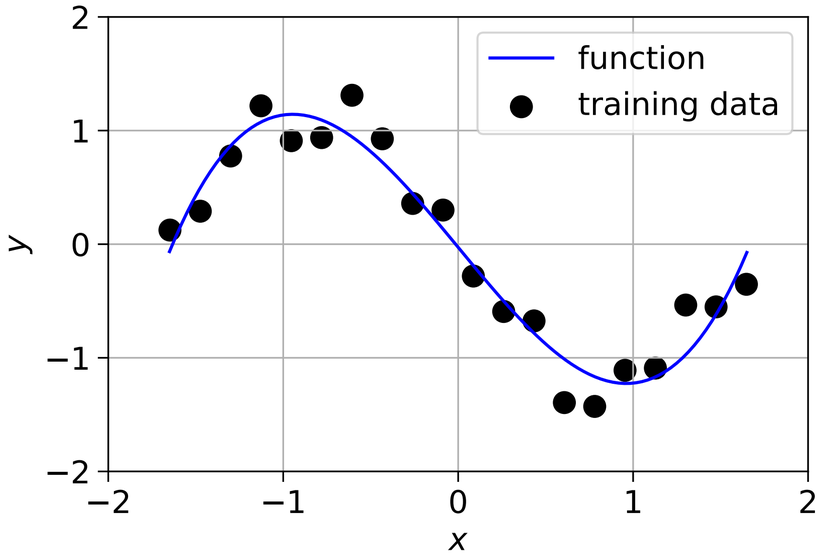




Polynomial regression

- We can use an M^{th} degree polynomial as our model $\hat{y} = f(x) = \sum_{m=0}^{M} w_m x^m$
- Using $M = 3$ (i.e. a cubic) gives us a good fit
- **This is still linear regression** as the model is linear in the weights!

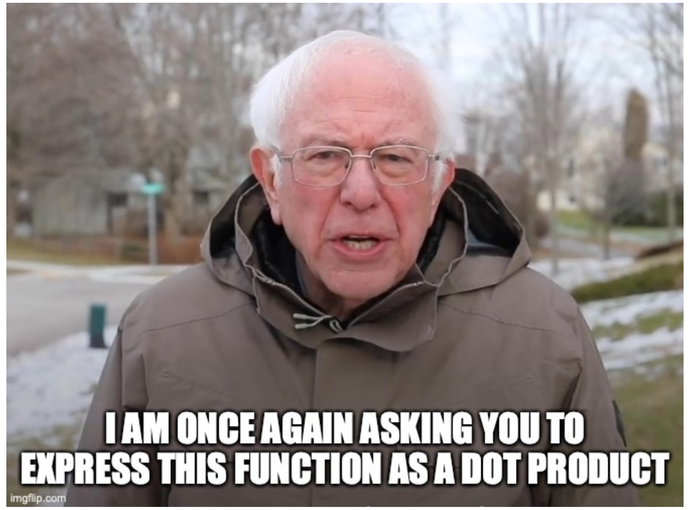




How do we fit this function?

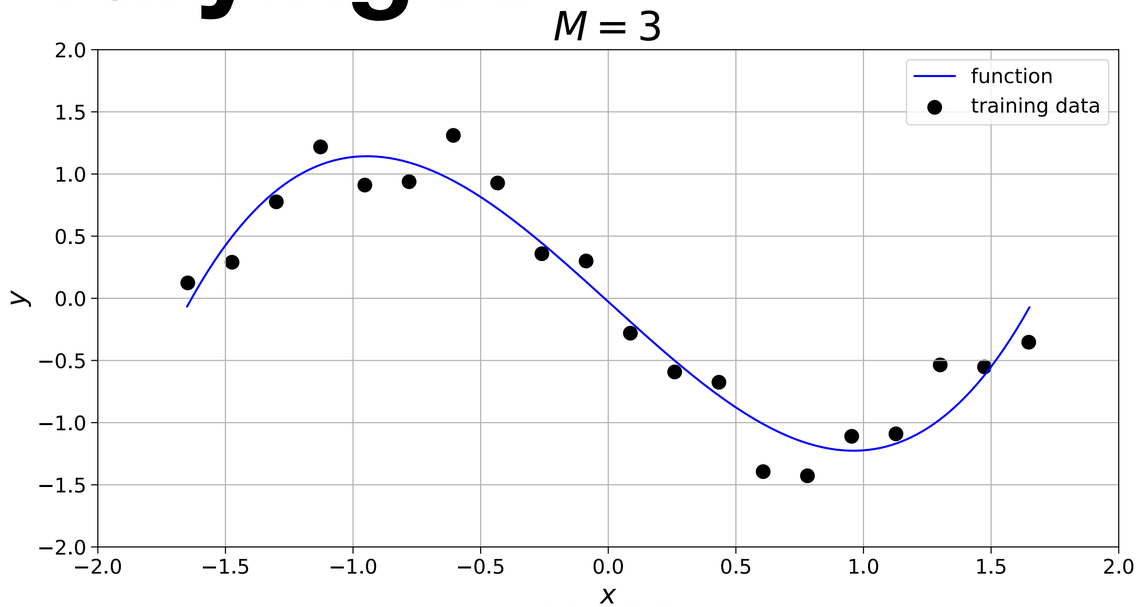
- Our function is $\hat{y} = f(x) = \sum_{m=0}^{M} w_m x^m$

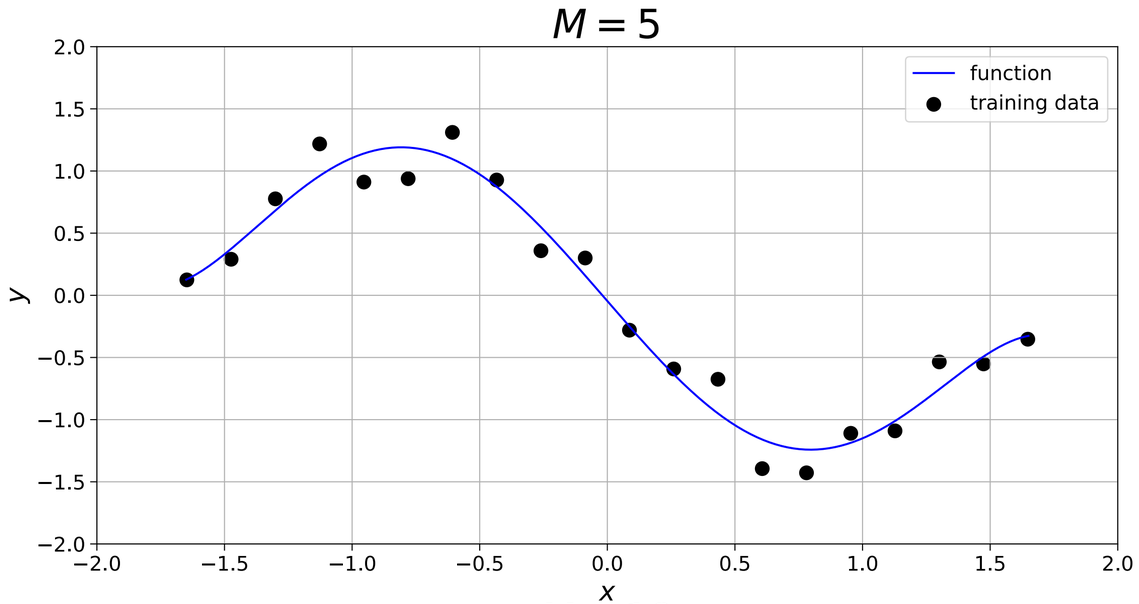
  1. (Re)define $\phi(\mathbf{x}) = \begin{bmatrix} 1 & x & x^2 & \dots & x^M \end{bmatrix}^\top$
  2. Write $\mathbf{w} = \begin{bmatrix} w_0 & w_1 & w_2 & \dots & w^M \end{bmatrix}^\top$

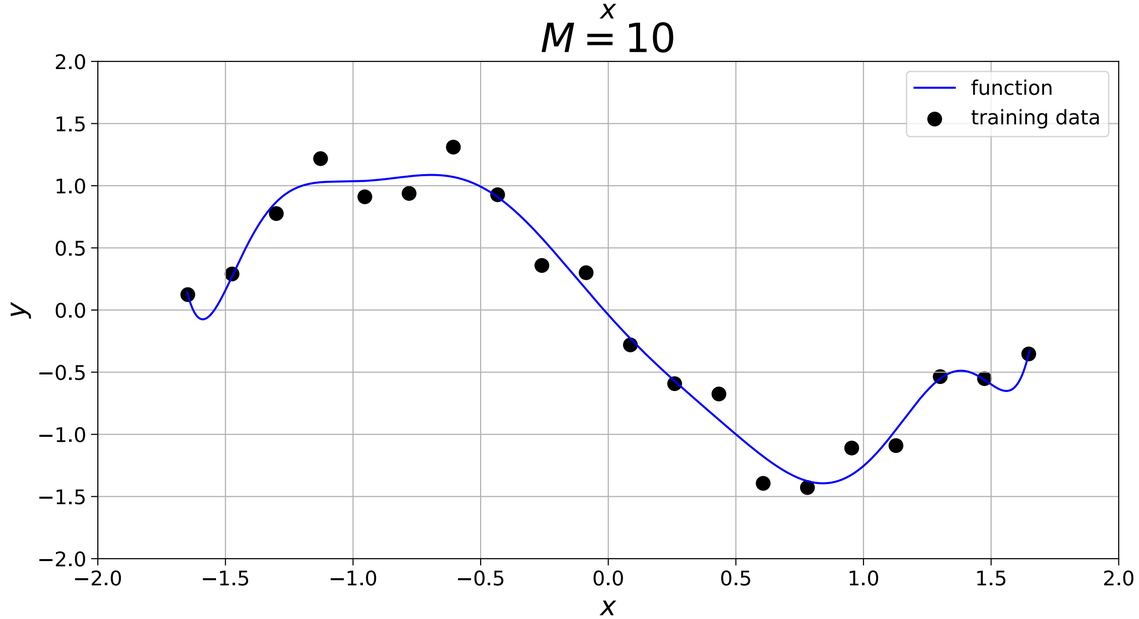


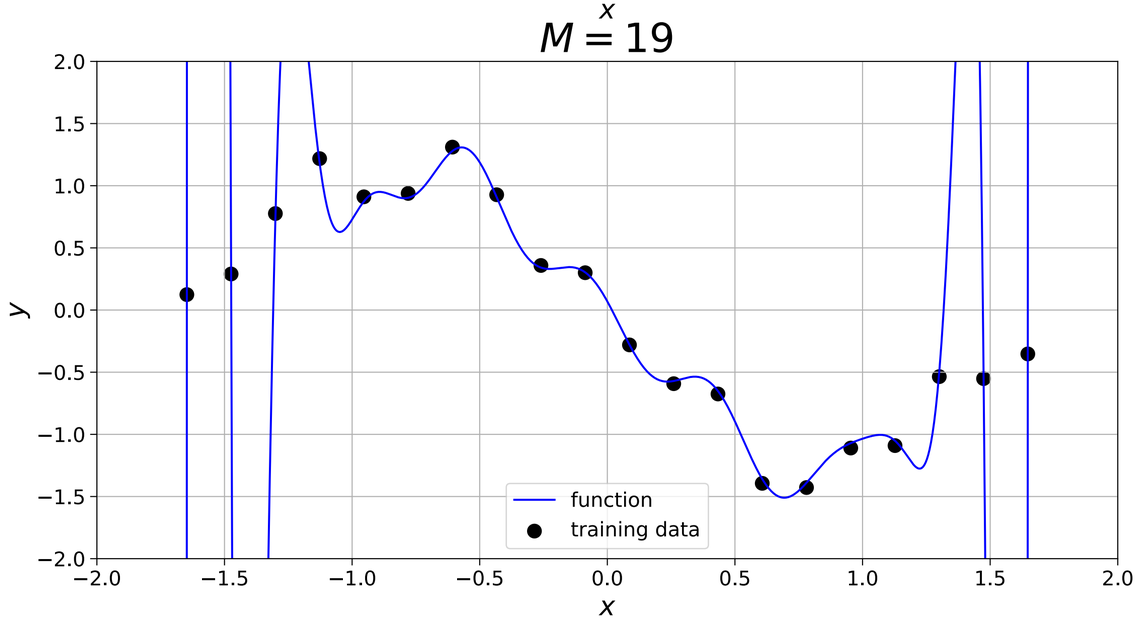
- We get $f(\mathbf{x}) = \hat{y} = \mathbf{w}^\top \phi(\mathbf{x})$ and we can again write $L_{SE}(\mathbf{w}) = \|\mathbf{y} - \mathbf{\Phi w}\|^2$
- We can minimise this with $\mathbf{w} = (\mathbf{\Phi}^\top \mathbf{\Phi})^{-1} \mathbf{\Phi}^\top \mathbf{y}$ as before

Varying M


$M = 3$
function
training data
x
y

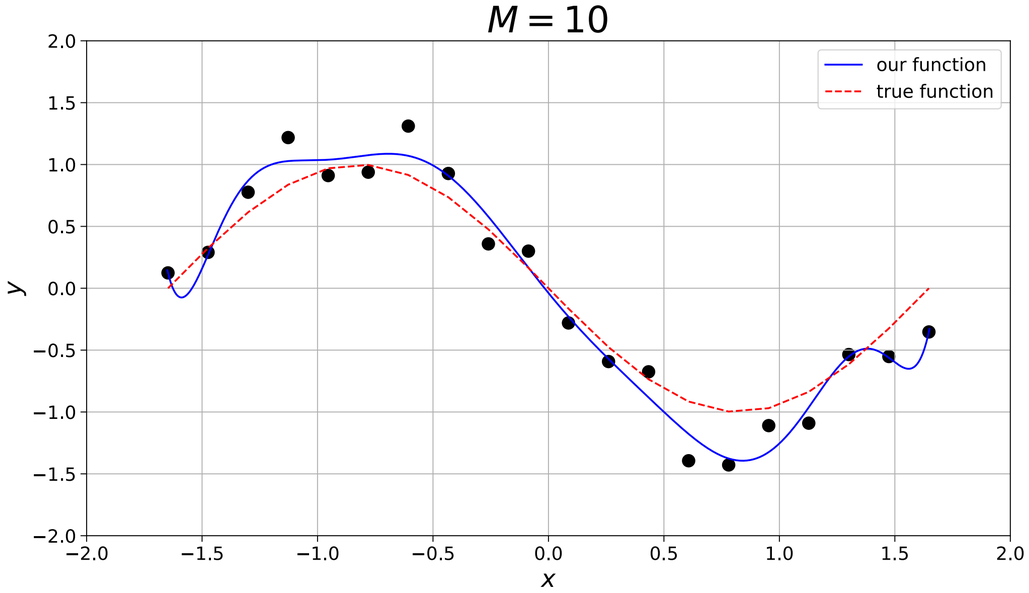

$M = 5$
function
training data
x
y

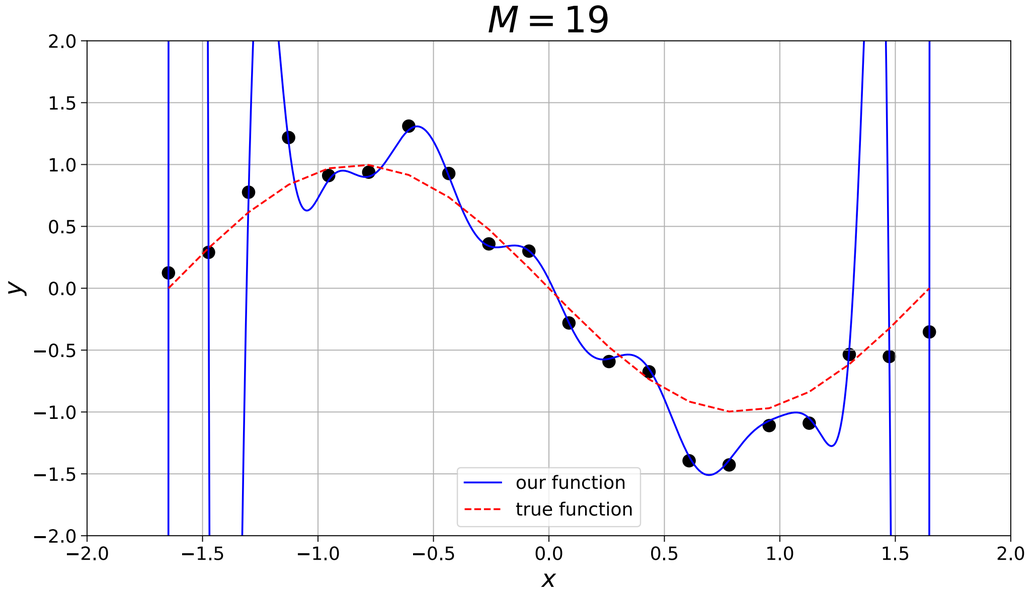

$M = 10$
function
training data
x
y


$M = 19$
function
training data
x
y

Overfitting

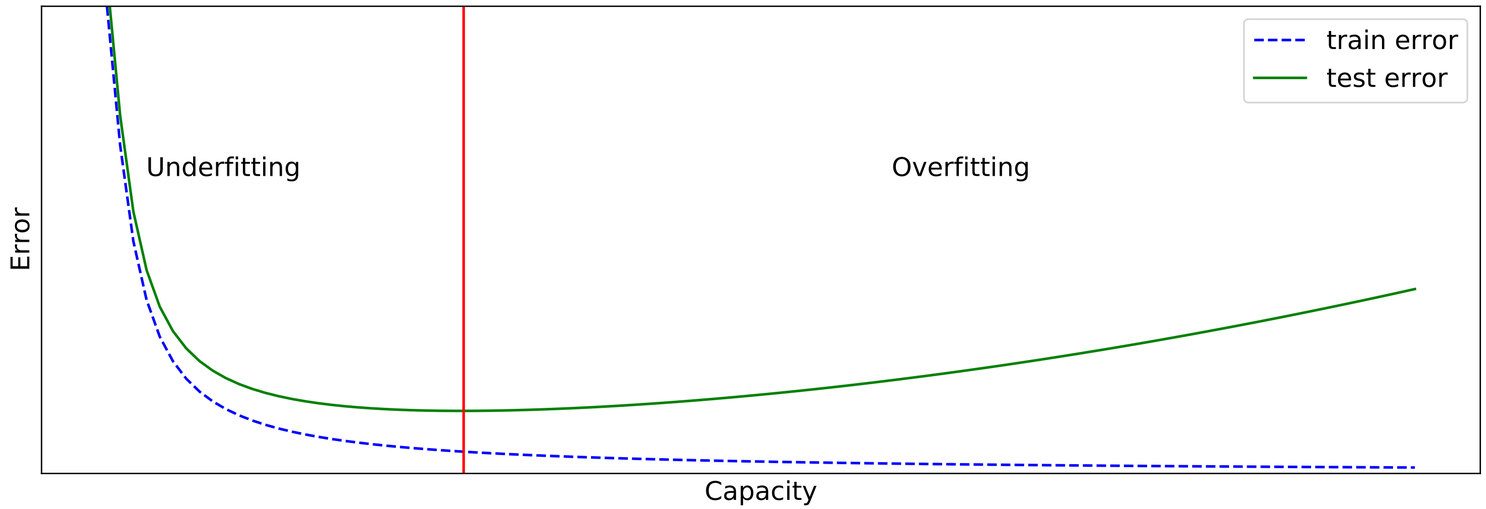
- These models have overfit to the training data
- We want our models to generalise to test data —these don't!
- **Spoilers**: $y = \sin((x - a)/b) + \mathcal{N}(0,0.25^2)$
- The models have too much **capacity,** and are latching on to the noise





Regularisation

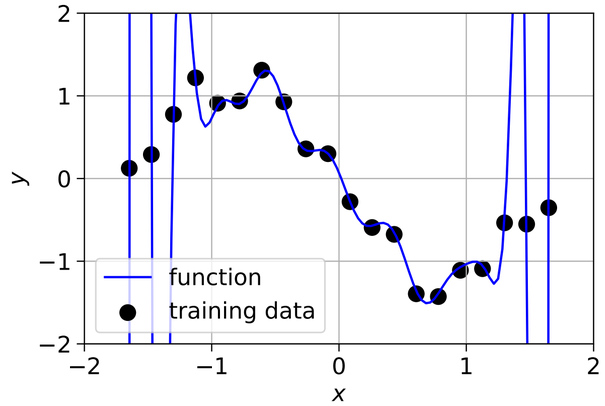
- We ultimately want to maximise test performance i.e. minimise test error
- The model should have the capacity to represent the function we care about
- But high capacity models tend to overfit
- **Regularisation** techniques combat overfitting by making the model *simpler*



This figure is my reproduction of Figure 5.3 from https://www.deeplearningbook.org/contents/ml.html

L2 regularisation

- Overfitted models tend to have large weights



$$y = 30.38x^{19} - 18.83x^{18} - 313.41x^{17} + \ldots$$

- We can regularise our model by penalising large weight values
- Let's add a term to our loss function that is small when weights are small

$$L_{ridge}(\mathbf{w}) = \underbrace{\|\mathbf{y} - \mathbf{\Phi w}\|^2}_{SE} + \underbrace{\lambda \|\mathbf{w}\|^2}_{regularisation}$$

Ridge regression

$$L_{ridge}(\mathbf{w}) = \underbrace{\|\mathbf{y} - \mathbf{\Phi w}\|^2}_{SE} + \underbrace{\lambda \|\mathbf{w}\|^2}_{regularisation} \quad \text{where } \|\mathbf{w}\|^2 = \mathbf{w}^\top \mathbf{w}$$

This function is **convex:** it only has one extremum which is a minimum

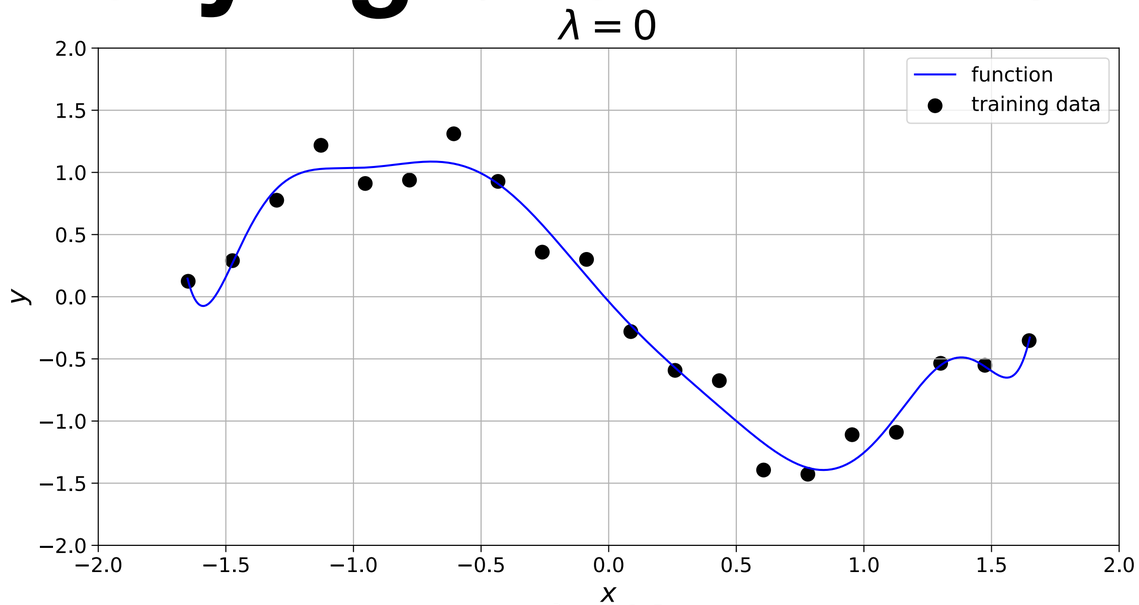
- λ is a hyperparameter that tells us how important regularisation is
- Let's take the gradient and set to zero to get the optimal weights

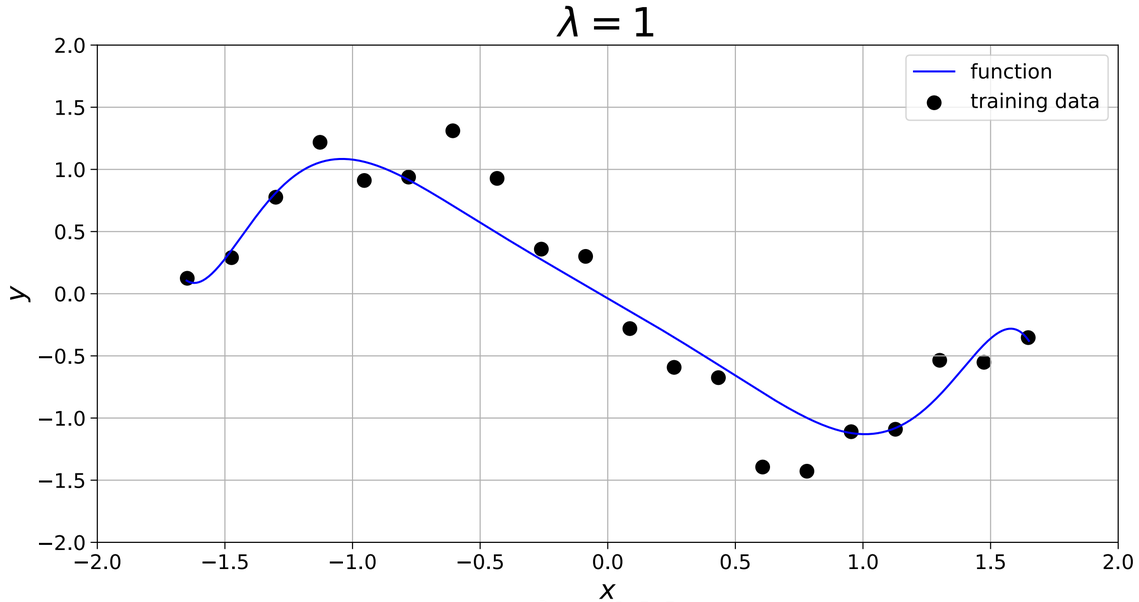
$$\nabla_{\mathbf{w}} L_{ridge} = -2\mathbf{\Phi}^\top(\mathbf{y} - \mathbf{\Phi w}) + 2\lambda \mathbf{w} = 0$$

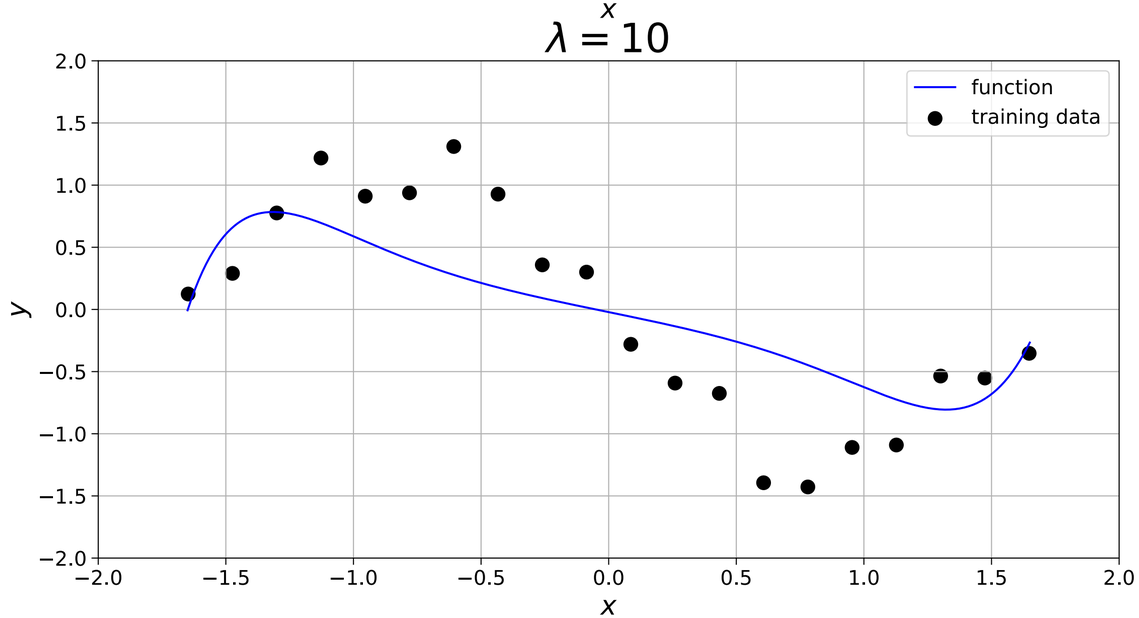
$$\mathbf{w} = (\mathbf{\Phi}^\top \mathbf{\Phi} + \lambda I)^{-1} \mathbf{\Phi}^\top \mathbf{y}$$

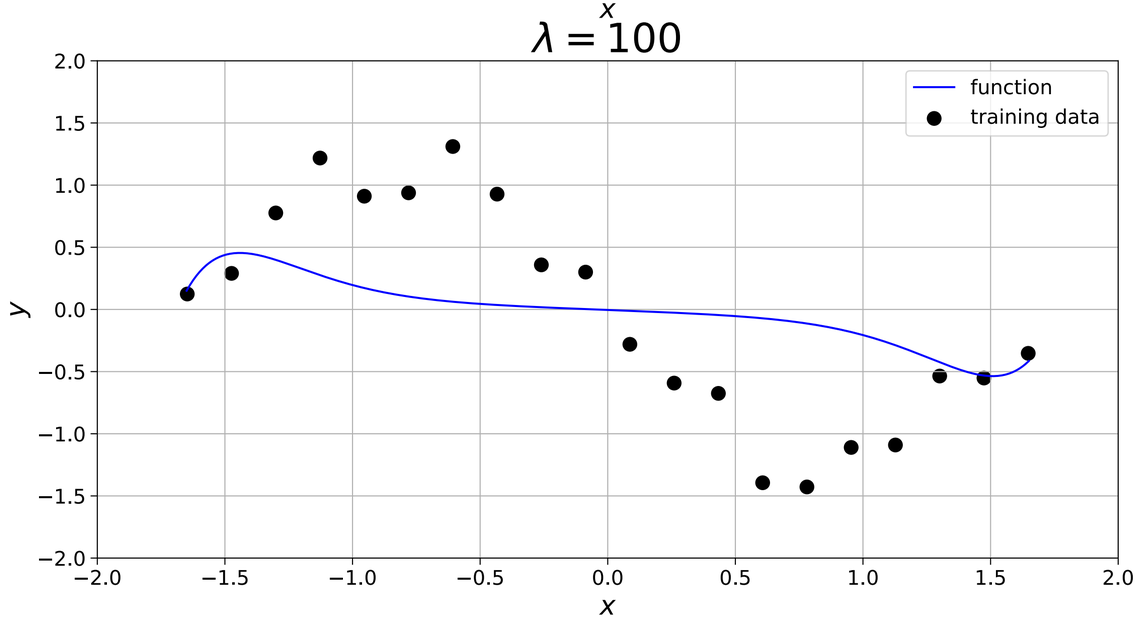
But in practice, we don't regularise the bias term!

Varying λ for $M = 10$









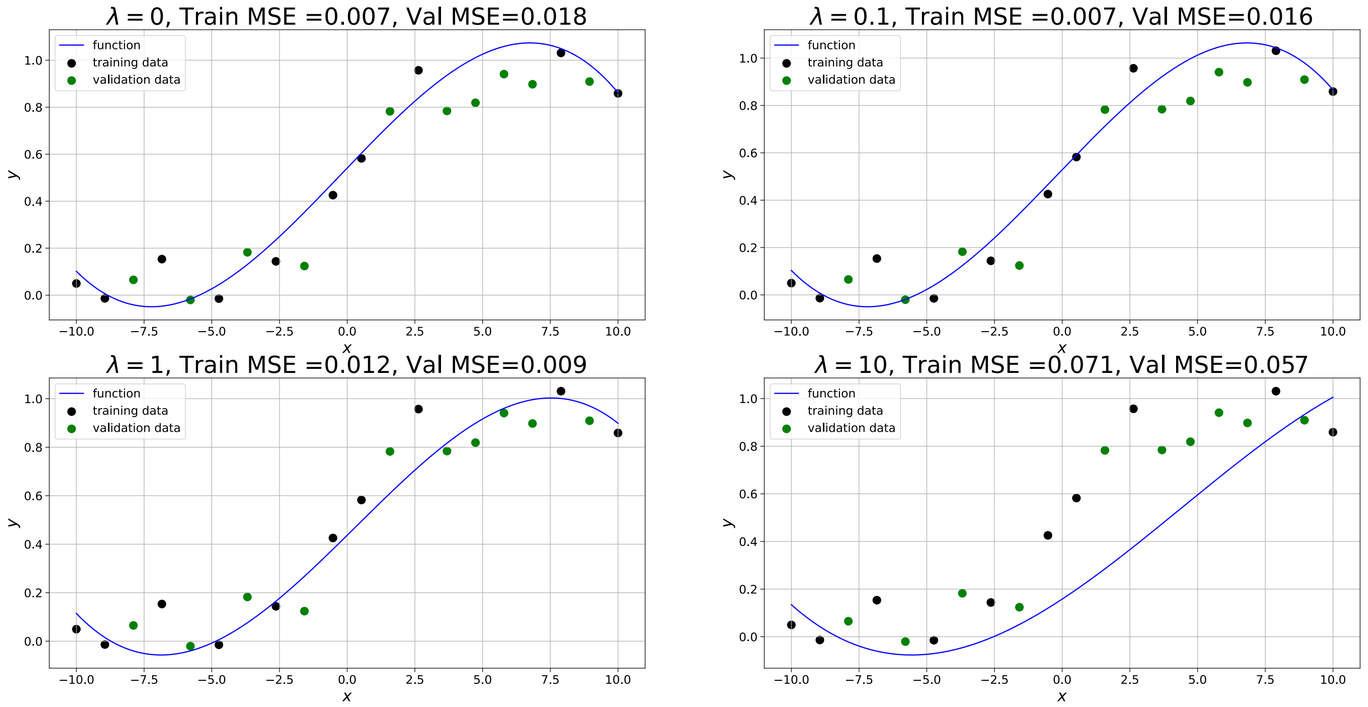
The validation set

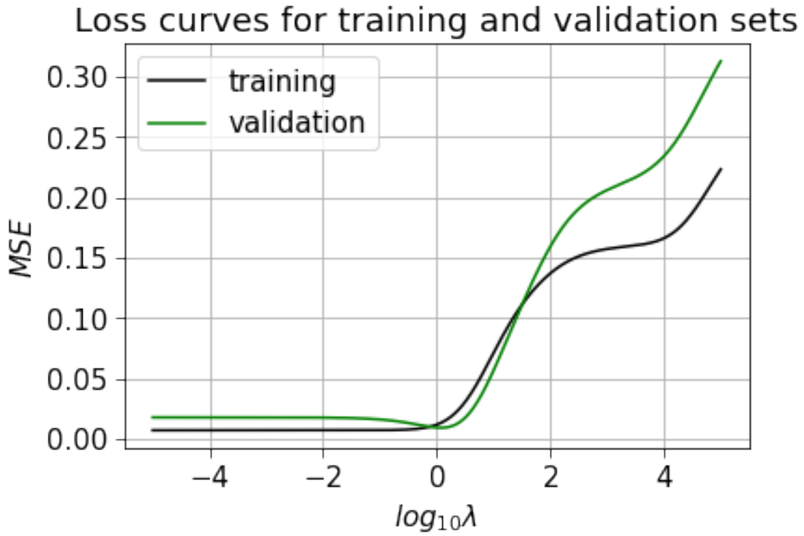
- Our goal is to perform well on the test set. Can we try different values of λ and pick the one that maximises test performance?
- **No! This would be using the test set to fit the model**
- Instead, we split the dataset three-ways: train, validation, test
- The validation set is used to tune hyperparameters

Sometimes we will just use default hyperparameter values!

Hyperparameter tuning with grid search

- Create a list of λ values and for each value fit a model on the training set
- Evaluate each model on the validation set (e.g. with MSE or R^2)
- Keep the model that performs best on validation **then** apply to test





Grid search

- We create a grid of possible values for each hyperparameter
- We then train a model for each grid element, and pick the model that performs best on validation**.** This is **model selection**
- With one hyperparameter, the grid is 1D, with two it's 2D and so on
- This can quickly get very expensive!

Imagine we have hyperparameters α and β. Let's search over $\alpha = \{0,1\}$ and $\beta = \{0.1,1,10\}$

	$\beta = 0.1$	$\beta = 1$	$\beta = 10$
$\alpha = 0$	$R^2_{val} = 0.46$	$R^2_{val} = 0.52$	$R^2_{val} = 0.39$
$\alpha = 1$	$R^2_{val} = 0.73$	$R^2_{val} = 0.87$	$R^2_{val} = 0.79$

A general framework for linear regression

All the models so far assume the form $f(\mathbf{x}) = \hat{y} = \mathbf{w}^\top \phi(\mathbf{x})$

- For simple linear regression $\phi(x) = \begin{bmatrix} 1 & x \end{bmatrix}^\top$
- For multiple linear regression $\phi(\mathbf{x}) = \begin{bmatrix} 1 & \mathbf{x}^\top \end{bmatrix}^\top$
- For polynomial regression $\phi(x) = \begin{bmatrix} 1 & x & x^2 & \dots & x^M \end{bmatrix}^\top$
- In each case $\mathbf{w}$ is a vector of weights with the same dimensionality as $\phi(x)$

$$L_{ridge}(\mathbf{w}) = \underbrace{\|\mathbf{y} - \mathbf{\Phi w}\|^2}_{SE} + \underbrace{\lambda \|\mathbf{w}\|^2}_{regularisation}$$

$$\mathbf{w} = (\mathbf{\Phi}^\top \mathbf{\Phi} + \lambda I)^{-1} \mathbf{\Phi}^\top \mathbf{y}$$

Basis functions and feature vectors

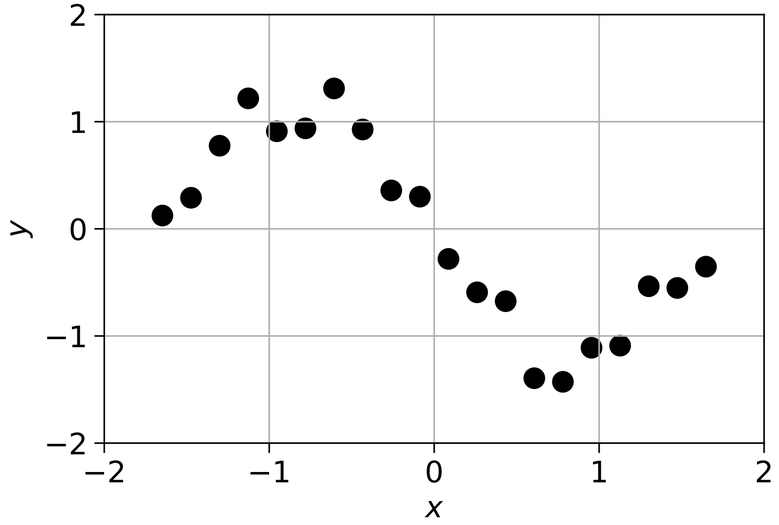
- $\phi(\mathbf{x})$ provides a basis for $\mathbf{x}$ (which can be non-linear e.g. polynomial)
- $\phi(\mathbf{x}) = \begin{bmatrix} a(\mathbf{x}) & b(\mathbf{x}) & c(\mathbf{x}) & \dots & z(\mathbf{x}) \end{bmatrix}^\top$
- These can be whatever!
- Our predictions are a just linear combination of the elements of $\phi(\mathbf{x})$
- These elements are called basis functions
- $\phi(\mathbf{x})$ is the **feature vector** for $\mathbf{x}$ and ϕ is a feature map

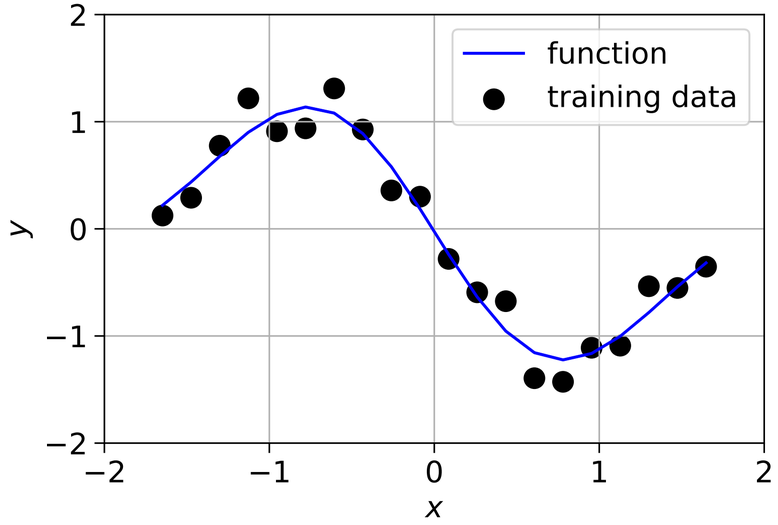
Gaussian basis functions

- We can design our own $\phi(\mathbf{x})$; $\mathbf{\Phi}$ is often referred to as the design matrix
- Each basis function could be a Gaussian centred on each training point

$$\phi(x) = \left[e^{-(x-x^{(0)})^2/\sigma^2} \quad e^{-(x-x^{(1)})^2/\sigma^2} \quad e^{-(x-x^{(2)})^2/\sigma^2} \quad \ldots \quad e^{-(x-x^{(N-1)})^2/\sigma^2}\right]^\top$$

- Here, σ is an additional hyperparameter





The pesky bias term

- We've been writing $f(\mathbf{x}) = \mathbf{w}^\top \boldsymbol{\phi}(\mathbf{x})$ where $\mathbf{w} = \begin{bmatrix} b & w_0 & w_1 & \dots \end{bmatrix}^\top$
- Models in sklearn are $f(\mathbf{x}) = \mathbf{w}^\top \boldsymbol{\phi}(\mathbf{x}) + b$ where $\mathbf{w} = [w_0 \quad w_1 \quad \dots]^\top$
- i.e. **the bias term is explicit**
- This means we don't need to include a constant term in $\boldsymbol{\phi}(\mathbf{x})$ in practice
- **Make sure you're happy with this** as we will start using $f(\mathbf{x}) = \mathbf{w}^\top \boldsymbol{\phi}(\mathbf{x}) + b$ when we consider linear models for classification

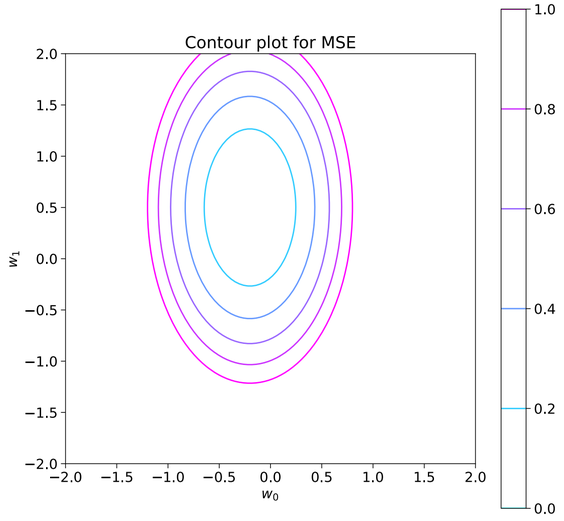
The maths for regression is a lot nicer if the bias is included with the weights

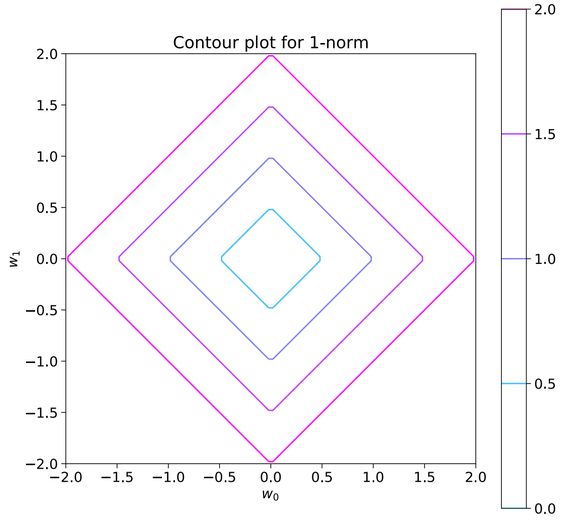
Lasso regression

$$L_{lasso}(\mathbf{w}) = \underbrace{\frac{1}{2N}\|\mathbf{y} - \mathbf{\Phi w}\|^2}_{MSE} + \underbrace{\lambda\,|\,\mathbf{w}\,|}_{regularisation}$$

- Very similar to ridge regression except the SE term has been scaled and the regularisation term is a 1-norm

- 1-norm encourages sparsity in $\mathbf{w}$ which is a form of variable selection





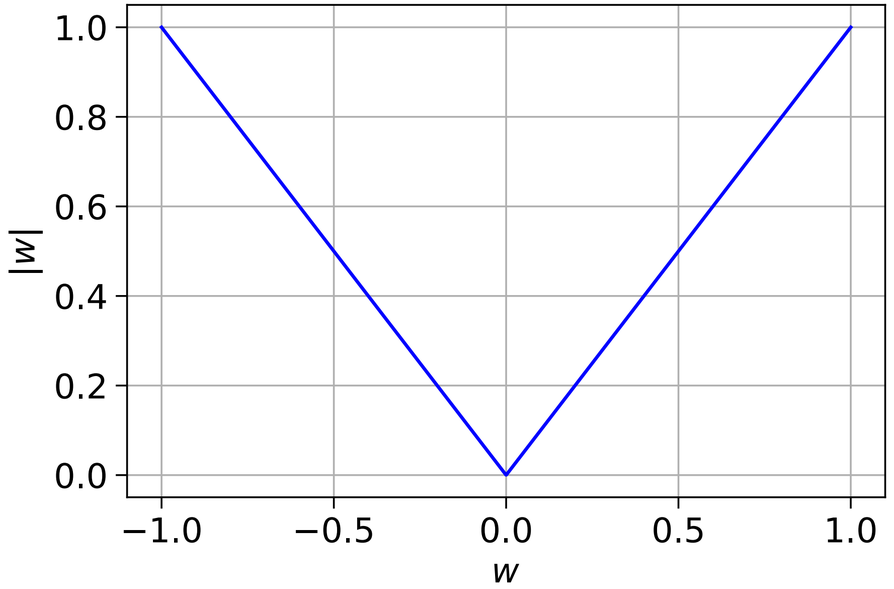
The minimum occurs at one of the points where the contours of the two terms are at a tangent

Such points are more likely to occur at the corners of The 1-norm contours

Optimisation

- Finding the weights that minimise a loss function on training data is an optimisation problem $\underset{\mathbf{w}}{\text{minimise}}\ L(\mathbf{w})$ with solution $\mathbf{w}^* = \arg\min_{\mathbf{w}} L(\mathbf{w})$
- This was simple for $L_{ridge}(\mathbf{w})$ which is convex and differentiable
- We just compute $\nabla_{\mathbf{w}} L_{ridge}$ and set to zero
- However, $L_{lasso}(\mathbf{w})$ is non-differentiable

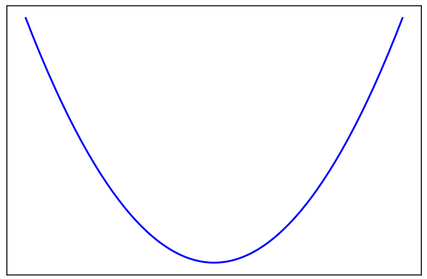
$$L_{lasso}(\mathbf{w}) = \underbrace{\frac{1}{2N}\|\mathbf{y} - \mathbf{\Phi w}\|^2}_{MSE} + \underbrace{\lambda\,|\,\mathbf{w}\,|}_{regularisation}$$



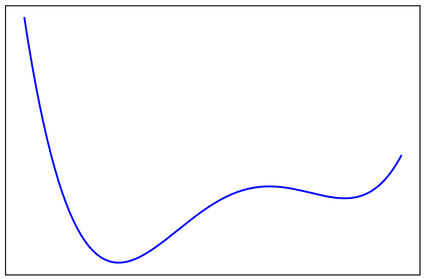
Convexity

- Convex functions have one extremum which is a minimum. This is very useful for optimisation!

A convex function

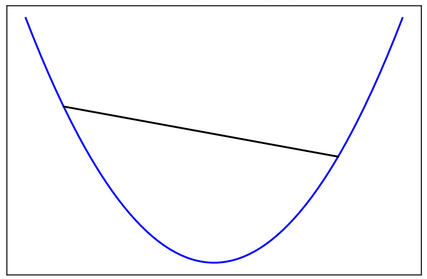


A non-convex function

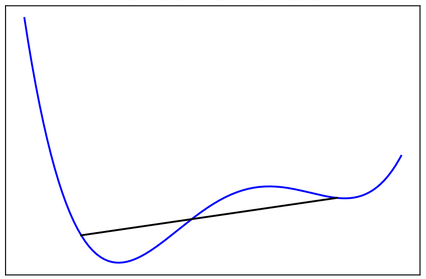


- A function of one variable is convex if a line drawn between any two points on the function doesn't fall below the function

A convex function

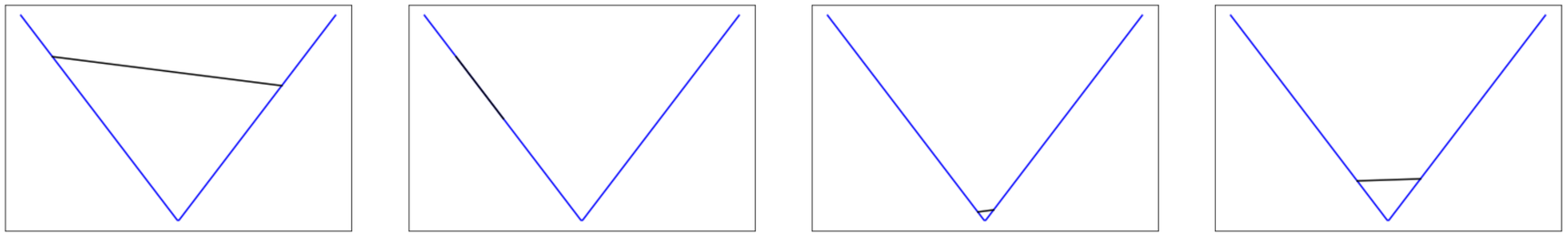


A non-convex function



Lasso is convex

- $|w|$ is convex (as is $|\mathbf{w}|$): it clearly has a minimum at $w = 0$.
- It not being differentiable doesn't change this



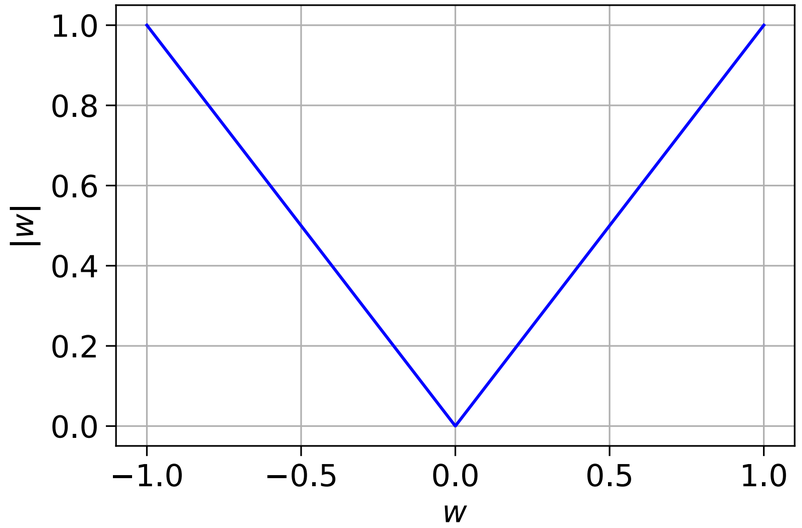
- The sum of two convex functions is convex

$$L_{lasso}(\mathbf{w}) = \underbrace{\frac{1}{2N}\|\mathbf{y} - \mathbf{\Phi w}\|^2}_{MSE} + \underbrace{\lambda|\mathbf{w}|}_{regularisation}$$

- $L_{lasso}(\mathbf{w})$ is convex, we just need to find its minimum

Subderivatives

- $g(w) = |w|$ is piecewise differentiable



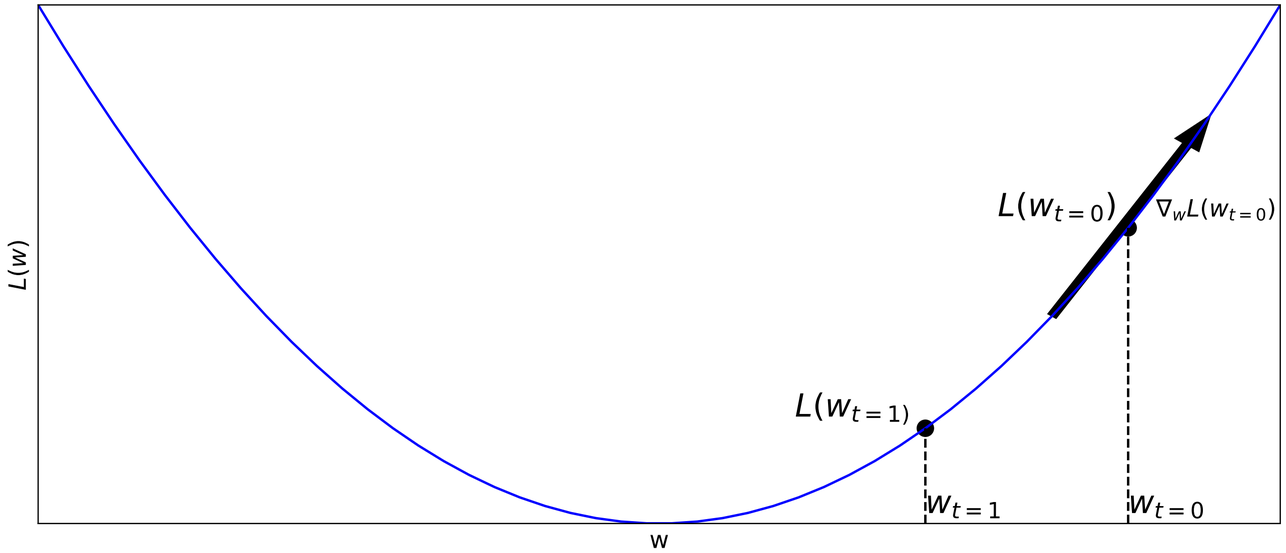
$$\frac{dg}{dw} = \begin{cases} 1 & \text{if } w > 0 \\ -1 & \text{if } w < 0 \end{cases}$$

- We can evaluate the gradient at any point (except $w = 0$)
- This is all we need to do to perform gradient descent (GD)

Gradient descent (GD) intuition

- We have a function $L(\mathbf{w})$ and we want to find $\mathbf{w}^* = \arg\min_{\mathbf{w}} L(\mathbf{w})$
- Let's initialise $\mathbf{w}$ at random and call it $\mathbf{w}_{t=0}$
- The gradient at $\mathbf{w}_{t=0}$: $\nabla_{\mathbf{w}} L(\mathbf{w}_{t=0})$ tells us **locally** the direction we can move $\mathbf{w}_{t=0}$ to most increase the function
- Move in the opposite direction!

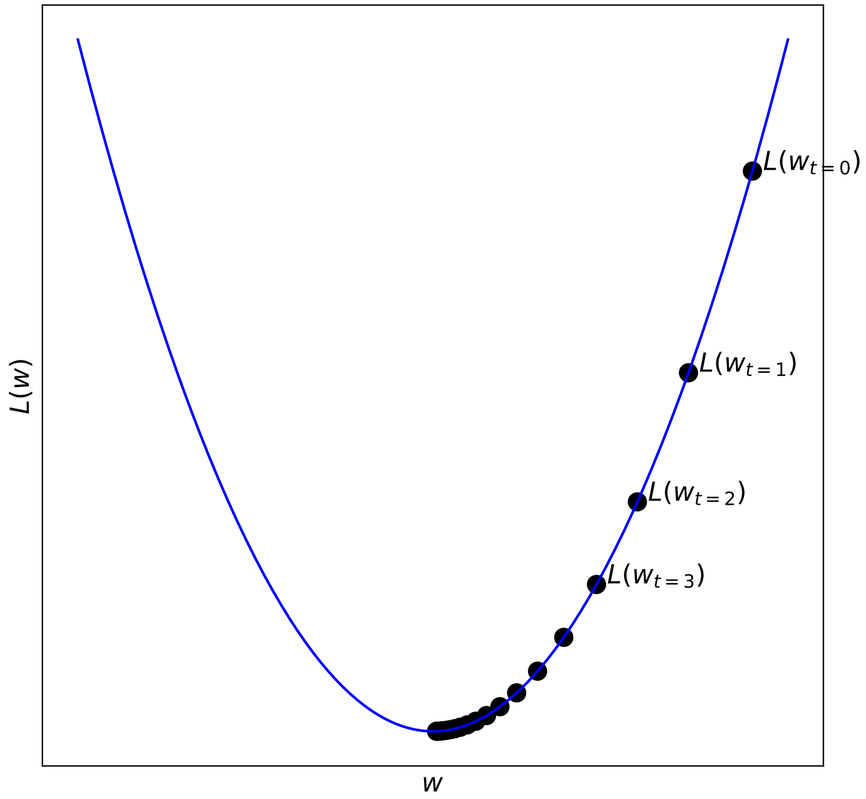
$$\mathbf{w}_{t=1} = \mathbf{w}_{t=0} - \alpha \nabla_{\mathbf{w}} L(\mathbf{w}_{t=0})$$



Gradient descent (GD) algorithm

Goal: We have a function $L(\mathbf{w})$ and we want to find $\mathbf{w}^* = \arg\min_{\mathbf{w}} L(\mathbf{w})$

- Initialise $\mathbf{w}$ as $\mathbf{w}_{t=0}$
- For i in range(T):
  1. Compute $\nabla_{\mathbf{w}} L(\mathbf{w}_{t=i})$
  2. Update $\mathbf{w}_{t=i+1} = \mathbf{w}_{t=i} - \alpha \nabla_{\mathbf{w}} L(\mathbf{w}_{t=i})$



α is called the step size, or *learning rate*

It is yet another hyperparameter

Optimisation algorithms

- The process of using an optimisation algorithm to learn the weights that minimise a loss function on training data is known as … **training!**
- There are many optimisation algorithms; some work better than others for different methods
- We will only detail variations of gradient descent on this course
- Sklearn will default to whatever optimiser tends to work best for a method
- Please be happy using optimisation algorithms that you haven't learnt about, and if you're not — go find out how they work!

Summary

- We have learnt about different types of linear regression
- We have reasoned the need for a test set for evaluation
- We have discovered how regularisers can prevent overfitting
- We have learnt how a validation set may be used to tune hyperparameters
- We have found out what convex functions are
- We have explored gradient descent for optimising convex functions