ECS171: Machine Learning

L4 Optimization and generalization of ML models

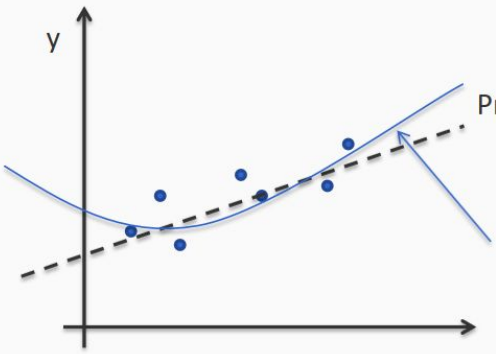
Instructor: Prof. Maike Sonnewald
TAs: Pu Sun & Devashree Kataria

MOCO Amsterdam garden

Intended Learning Outcomes

- Understand and be able to apply Least Mean Squares regression via Gradient Descent
  - Qualitatively appreciate the impact of choices e.g. the learning rate
- Describe and apply the maximum likelihood estimate for simple examples
  - Qualitatively appreciate how the maximum likelihood estimate relates to probability distribution functions
- Be able to describe what regularization does with example figures
- Describe strategies and make estimates of choosing the ‘best’ model, including with cross-validation (e.g. k-fold) and hold-out data with different metrics to quantify the error
- Describe and apply scaling methods and how they impact a models ability to find an optimum

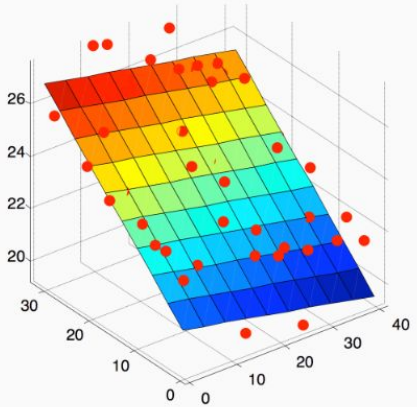
Problem setting reminder: We want to fit a function to data



Predict using $y = w_1 + w_2 x_2$

The linear function is not our only choice. We could have tried to fit the data as another polynomial

One dimensional input



Two dimensional input

Predict using $y = w_1 + w_2 x_2 + w_3 x_3$

Recap: What is the best weight vector?

Question: How do we know which weight vector is the best one for a training set?

For an input ($\mathbf{x}_i$, y_i) in the training set, the cost of a mistake is:

$$J(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{n}(y_i - \mathbf{w}^T\mathbf{x}_i)^2$$

We learn via optimization. We want to minimize the error by determining optimal weights **w:**

$$J(\mathbf{w}) = \underbrace{\min_{W}}_{\text{minimize}} \underbrace{\frac{1}{2}\sum_{i=1}^{n}(\underbrace{y_i - \mathbf{w}^T\mathbf{x}_i}_{\text{error}})^2}_{\text{mean squared error}}$$

Least Mean Squares regression via Gradient Descent

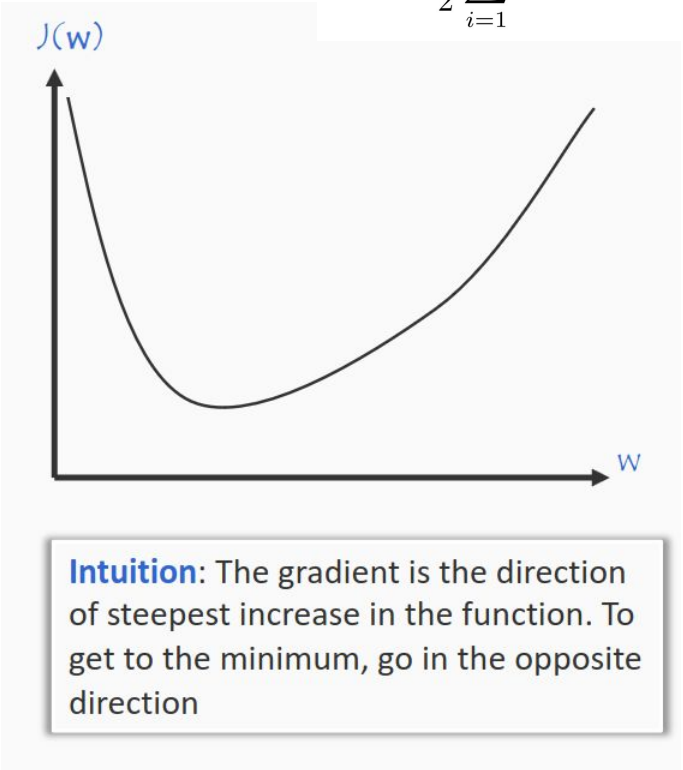
$$J(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{n}(y_i - \mathbf{w}^T\mathbf{x}_i)^2$$

Note: Analytical solution exists (lecture 3), so this optimization approach is merely a demonstration of the principle.

General strategy for minimizing the cost function J(**w**):

1. Start with an initial guess for entries in **w**, let's call this $\mathbf{w}^0$

2. Calculate the gradient, and update **w**t in the direction the gradient has the steepest increase in the function. To get to the minimum, go in the opposite direction.
    - Compute the gradient of the gradient of J($\mathbf{w}^t$)
    - Update $\mathbf{w}^t$ to get $\mathbf{w}^{t+1}$ by taking a "step" in the pposite direction of the gradient



Least Mean Squares regression via Gradient Descent

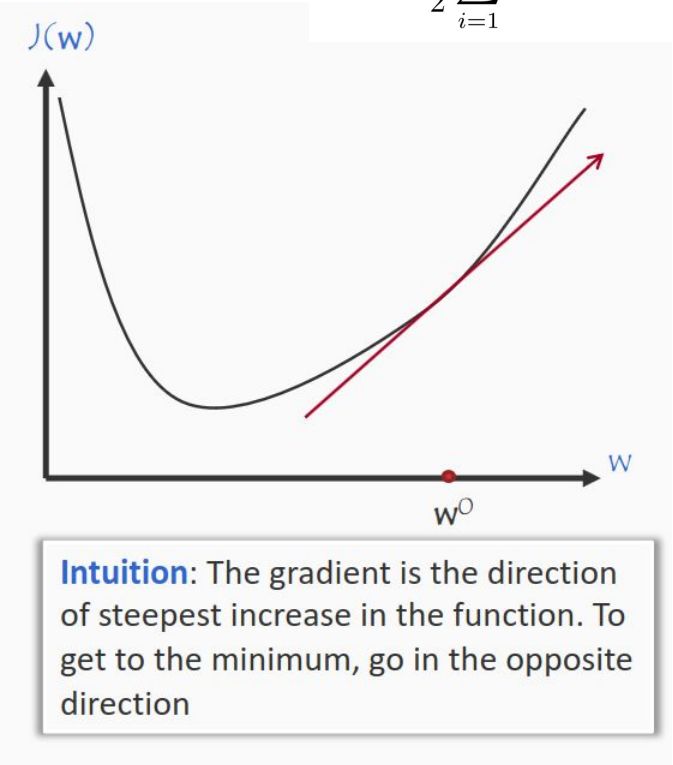
$$J(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{n}(y_i - \mathbf{w}^T\mathbf{x}_i)^2$$

Note: Analytical solution exists (lecture 3), so this optimization approach is merely a demonstration of the principle.

General strategy for minimizing the cost function J(**w**):

1. Start with an initial guess for entries in **w**, let's call this $\mathbf{w}^0$

2. Calculate the gradient, and update **w**t in the direction the gradient has the steepest increase in the function. To get to the minimum, go in the opposite direction.
    - Compute the gradient of the gradient of J($\mathbf{w}^t$)
    - Update $\mathbf{w}^t$ to get $\mathbf{w}^{t+1}$ by taking a "step" in the pposite direction of the gradient



Least Mean Squares regression via Gradient Descent

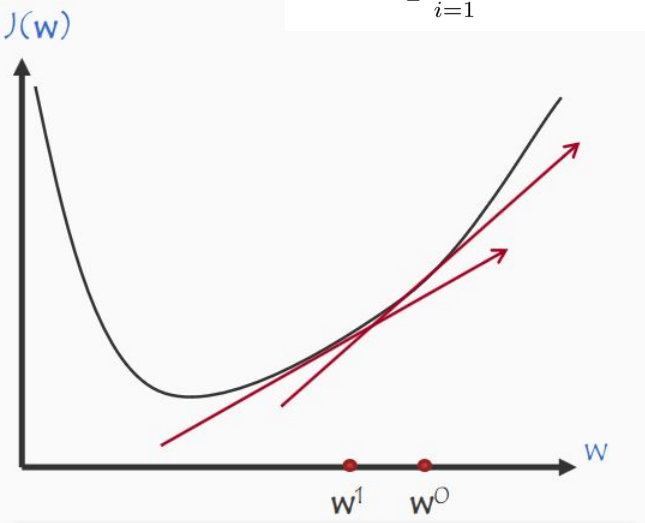
$$J(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{n}(y_i - \mathbf{w}^T\mathbf{x}_i)^2$$

Note: Analytical solution exists (lecture 3), so this optimization approach is merely a demonstration of the principle.

General strategy for minimizing the cost function J(**w**):

1. Start with an initial guess for entries in **w**, let's call this $\mathbf{w}^0$

2. Calculate the gradient, and update **w**t in the direction the gradient has the steepest increase in the function. To get to the minimum, go in the opposite direction.
    - Compute the gradient of the gradient of J($\mathbf{w}^t$)
    - Update $\mathbf{w}^t$ to get $\mathbf{w}^{t+1}$ by taking a "step" in the pposite direction of the gradient



Least Mean Squares regression via Gradient Descent

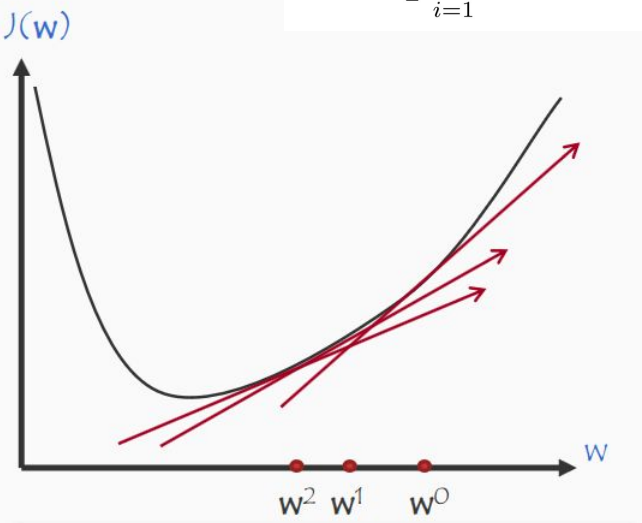
$$J(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{n}(y_i - \mathbf{w}^T\mathbf{x}_i)^2$$

Note: Analytical solution exists (lecture 3), so this optimization approach is merely a demonstration of the principle.

General strategy for minimizing the cost function J(**w**):

1. Start with an initial guess for entries in **w**, let's call this $\mathbf{w}^0$

2. Calculate the gradient, and update **w**t in the direction the gradient has the steepest increase in the function. To get to the minimum, go in the opposite direction.
   - Compute the gradient of the gradient of J($\mathbf{w}^t$)
   - Update $\mathbf{w}^t$ to get $\mathbf{w}^{t+1}$ by taking a "step" in the pposite direction of the gradient



Intuition: The gradient is the direction of steepest increase in the function. To get to the minimum, go in the opposite direction

Least Mean Squares regression via Gradient Descent

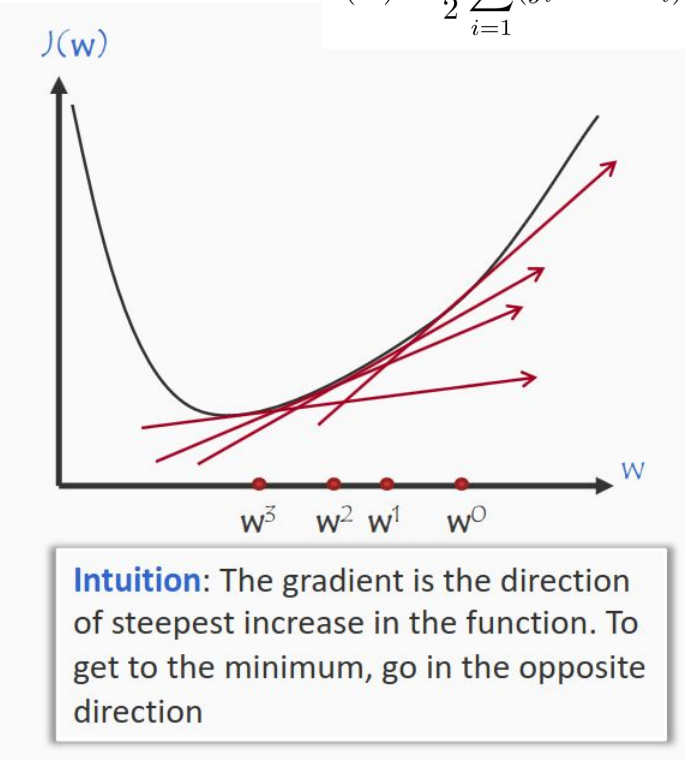
$$J(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{n}(y_i - \mathbf{w}^T\mathbf{x}_i)^2$$

Note: Analytical solution exists (lecture 3), so this optimization approach is merely a demonstration of the principle.

General strategy for minimizing the cost function J(**w**):

1. Start with an initial guess for entries in **w**, let's call this $\mathbf{w}^0$

2. Calculate the gradient, and update **w**t in the direction the gradient has the steepest increase in the function. To get to the minimum, go in the opposite direction.
   - Compute the gradient of the gradient of J($\mathbf{w}^t$)
   - Update $\mathbf{w}^t$ to get $\mathbf{w}^{t+1}$ by taking a "step" in the pposite direction of the gradient



Least Mean Squares regression via Gradient Descent

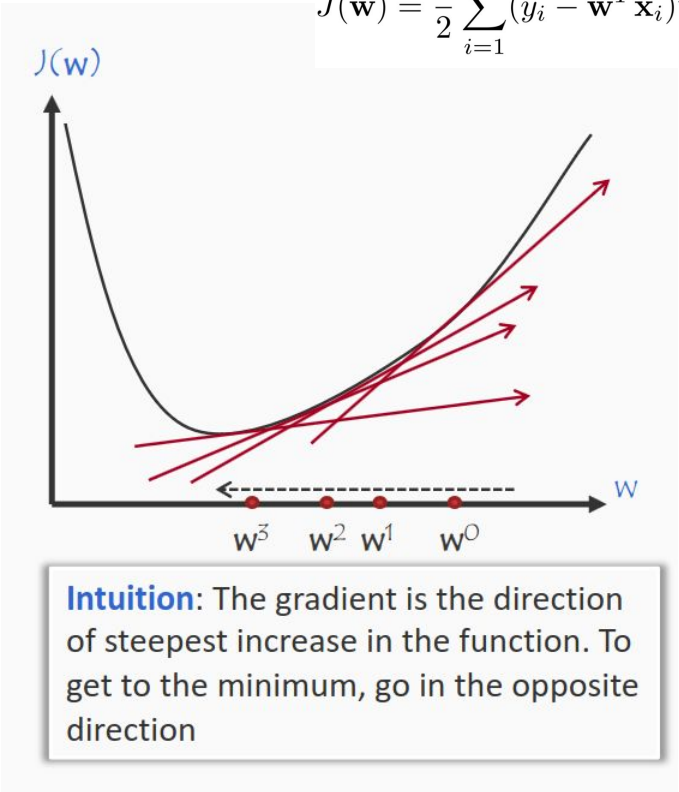
$$J(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{n}(y_i - \mathbf{w}^T\mathbf{x}_i)^2$$

Note: Analytical solution exists (lecture 3), so this optimization approach is merely a demonstration of the principle.

General strategy for minimizing the cost function J(**w**):

1. Start with an initial guess for entries in **w**, let's call this $\mathbf{w}^0$

2. Calculate the gradient, and update **w**t in the direction the gradient has the steepest increase in the function. To get to the minimum, go in the opposite direction.
    - Compute the gradient of the gradient of J($\mathbf{w}^t$)
    - Update $\mathbf{w}^t$ to get $\mathbf{w}^{t+1}$ by taking a "step" in the pposite direction of the gradient



Gradient descent for LMS

1. Initialize $\mathbf{w}^0$
2. For t = 0, 1, 2, ….
    1. Compute gradient of J($\mathbf{w}$) at $\mathbf{w}^t$. Call it ∇J($\mathbf{w}^t$)
    2. Update w as follows:

$$\mathbf{w}^{t+1} = \mathbf{w}^t - \lambda \nabla J(\mathbf{w}^t)$$

Here, the λ is the learning rate (covered further later in lecture)

Computing the gradient of the cost function

- The gradient is in the form: $\nabla J(\mathbf{w}^t) = \left[\frac{\partial J}{\partial w_1}, \frac{\partial J}{\partial w_2}, ..., \frac{\partial J}{\partial w_p}\right]$

- Recall, that : $\mathbf{w} = [w_1, w_2, ..., w_j, ..., w_n]$ with j indicating we have some w within **w** of length n

Computing the gradient of the cost function

- The gradient is in the form: $\nabla J(\mathbf{w}^t) = \left[\frac{\partial J}{\partial w_1}, \frac{\partial J}{\partial w_2}, ..., \frac{\partial J}{\partial w_p}\right]$

$$\frac{\partial J}{\partial w_j} = \frac{\partial J}{\partial w_j} \frac{1}{2} \sum_{i=1}^{n} (y_i - \mathbf{w}^T \mathbf{x}_i)^2$$

Computing the gradient of the cost function

- The gradient is in the form: $\nabla J(\mathbf{w}^t) = \left[\frac{\partial J}{\partial w_1}, \frac{\partial J}{\partial w_2}, ..., \frac{\partial J}{\partial w_p}\right]$

$$\frac{\partial J}{\partial w_j} = \frac{\partial J}{\partial w_j}\frac{1}{2}\sum_{i=1}^{n}(y_i - \mathbf{w}^T\mathbf{x}_i)^2$$

$$= \frac{1}{2}\sum_{i=1}^{n}\frac{\partial J}{\partial w_j}(y_i - \mathbf{w}^T\mathbf{x}_i)^2$$

Computing the gradient of the cost function

- The gradient is in the form: $\nabla J(\mathbf{w}^t) = \left[\frac{\partial J}{\partial w_1}, \frac{\partial J}{\partial w_2}, ..., \frac{\partial J}{\partial w_p}\right]$

$$\frac{\partial J}{\partial w_j} = \frac{\partial J}{\partial w_j}\frac{1}{2}\sum_{i=1}^{n}(y_i - \mathbf{w}^T\mathbf{x}_i)^2$$

$$= \frac{1}{2}\sum_{i=1}^{n}\frac{\partial J}{\partial w_j}(y_i - \mathbf{w}^T\mathbf{x}_i)^2$$

$$= \frac{1}{2}\sum_{i=1}^{n}2(y_i - \mathbf{w}^T\mathbf{x}_i)\frac{\partial J}{\partial w_j}(y_i - w_1x_{i1} - ... - w_jx_{ij}$$

Computing the gradient of the cost function

- The gradient is in the form: $\nabla J(\mathbf{w}^t) = \left[\frac{\partial J}{\partial w_1}, \frac{\partial J}{\partial w_2}, ..., \frac{\partial J}{\partial w_p}\right]$

$$\frac{\partial J}{\partial w_j} = \frac{\partial J}{\partial w_j}\frac{1}{2}\sum_{i=1}^{n}(y_i - \mathbf{w}^T\mathbf{x}_i)^2$$

$$= \frac{1}{2}\sum_{i=1}^{n}\frac{\partial J}{\partial w_j}(y_i - \mathbf{w}^T\mathbf{x}_i)^2$$

$$= \frac{1}{2}\sum_{i=1}^{n}2(y_i - \mathbf{w}^T\mathbf{x}_i)\frac{\partial J}{\partial w_j}(y_i - w_1x_{i1} - ... - w_jx_{ij}$$

$$= \frac{1}{2}\sum_{i=1}^{n}2(y_i - \mathbf{w}^T\mathbf{x}_i)(-x_{ij})$$

Computing the gradient of the cost function

- The gradient is in the form: $\nabla J(\mathbf{w}^t) = \left[\frac{\partial J}{\partial w_1}, \frac{\partial J}{\partial w_2}, ..., \frac{\partial J}{\partial w_p}\right]$

$$\frac{\partial J}{\partial w_j} = \frac{\partial J}{\partial w_j}\frac{1}{2}\sum_{i=1}^{n}(y_i - \mathbf{w}^T\mathbf{x}_i)^2$$

$$= \frac{1}{2}\sum_{i=1}^{n}\frac{\partial J}{\partial w_j}(y_i - \mathbf{w}^T\mathbf{x}_i)^2$$

$$= \frac{1}{2}\sum_{i=1}^{n}2(y_i - \mathbf{w}^T\mathbf{x}_i)\frac{\partial J}{\partial w_j}(y_i - w_1x_{i1} - ... - w_jx_{ij}$$

$$= \frac{1}{2}\sum_{i=1}^{n}2(y_i - \mathbf{w}^T\mathbf{x}_i)(-x_{ij})$$

$$= -\sum^{n}(y_i - \mathbf{w}^T\mathbf{x}_i)x_{ij}$$

One element of the gradient vector

Differentiation is key concept

Differentiation Rule	f(x)	f'(x)
constant rule	$y = 5$	$\frac{\partial y}{\partial x} = 0$
power rule	$y = x^5$	$\frac{\partial y}{\partial x} = 5x^4$
constant multiple rule	$y = 4x^3$	$\frac{\partial y}{\partial x} = 12x^2$
sum rule	$y = x^6 + x^3$	$\frac{\partial y}{\partial x} = 6x^5 + 3x^2$
product rule	$y = e^{3x} sinx$	$\frac{\partial y}{\partial x} = e^{3x}(3sinx + cosx)$

https://www.mathsisfun.com/calculus/derivatives-rules.html

Gradient descent for LMS

This example has an increasing first derivative, making it appear to bend upwards: convex

1. Initialize $\mathbf{w}^0$

2. For t = 0, 1, 2, …. [until error is below a threshold]

    1. Compute gradient of J($\mathbf{w}$) at $\mathbf{w}^t$. Call it $\nabla J(w^t)$.
    Evaluate the function for each training example to compute the error and construct the gradient vector:

$$\frac{\partial J}{\partial w_j} = -\sum_{i=1}^{n}(y_i - \mathbf{w}^T\mathbf{x}_i)x_{ij}$$

One element of $\nabla J(w^t)$

    2: Update w as follows:

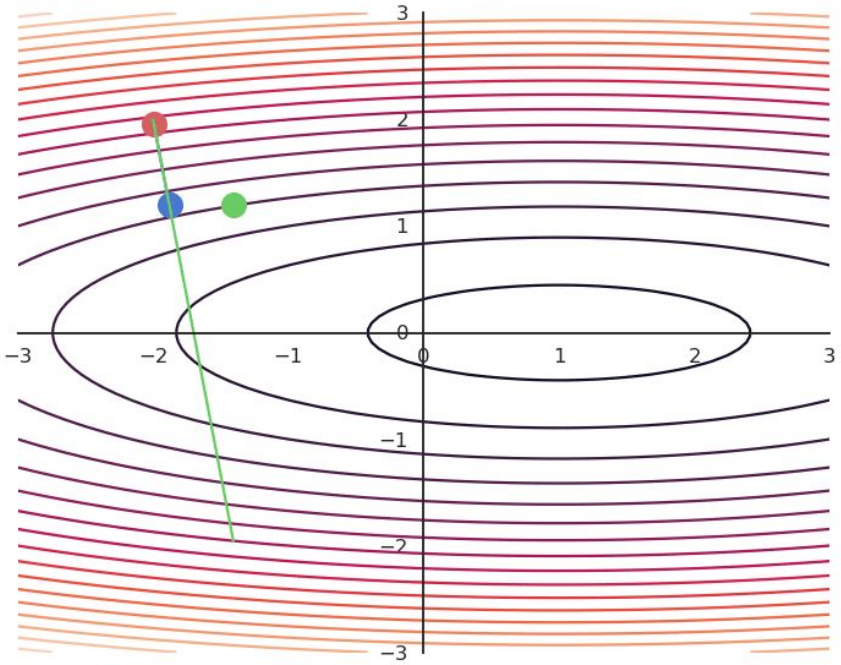
$$\mathbf{w}^{t+1} = \mathbf{w}^t - \lambda \nabla J(\mathbf{w}^t)$$

Here, the λ is the learning rate…

This algorithm is guaranteed to converge to the minimum of J if r is small enough.
Why? The objective J is a ***convex*** function

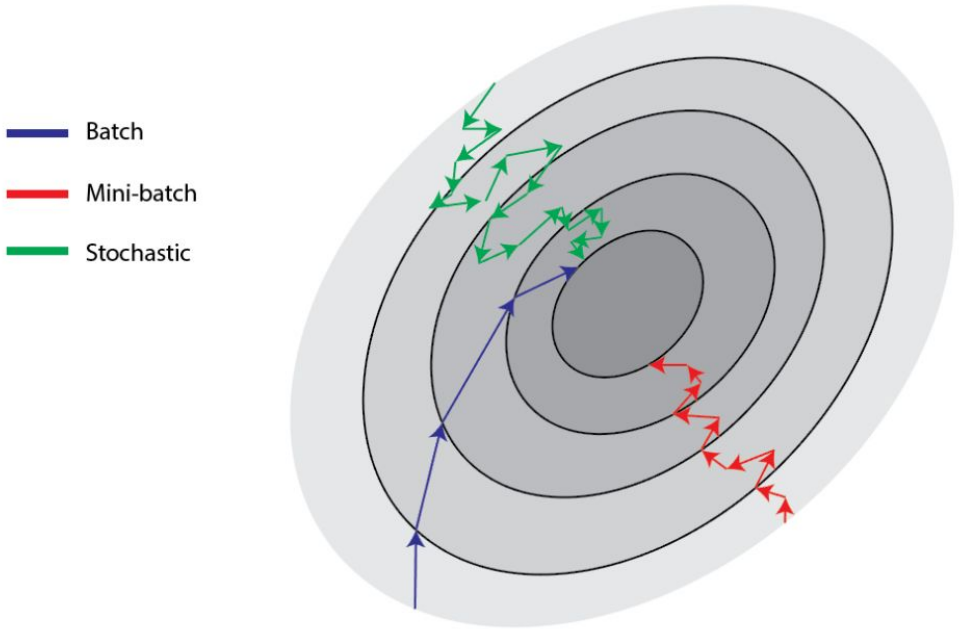
The impact of the learning rate is significant

- The learning rate has a big impact on how fast and if a model converges
- Red: small steps and converges slowly
- Green: big steps, could miss
- Blue: in-between
- Not knowing what the ‘landscape’ looks like makes estimating the learning rate an important question



https://julien-vitay.net/lecturenotes-neurocomputing/2-linear/1-Optimization.html

Approaches to updating weights can have a large impact



baeldung.com

Polynomial models

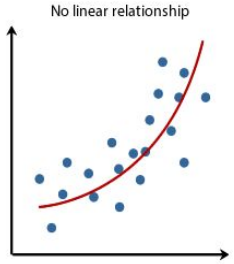
- Not all functions are convex and ‘guaranteed to converge’
- The world is full on non-linear relationships

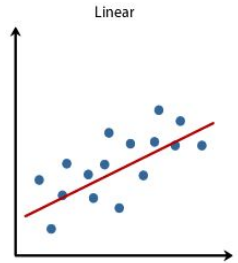
- We can create a more complicated model by defining input variables that are combinations of components of **x**

- Example: A polynomial function of p-th order of one dimensional feature x:

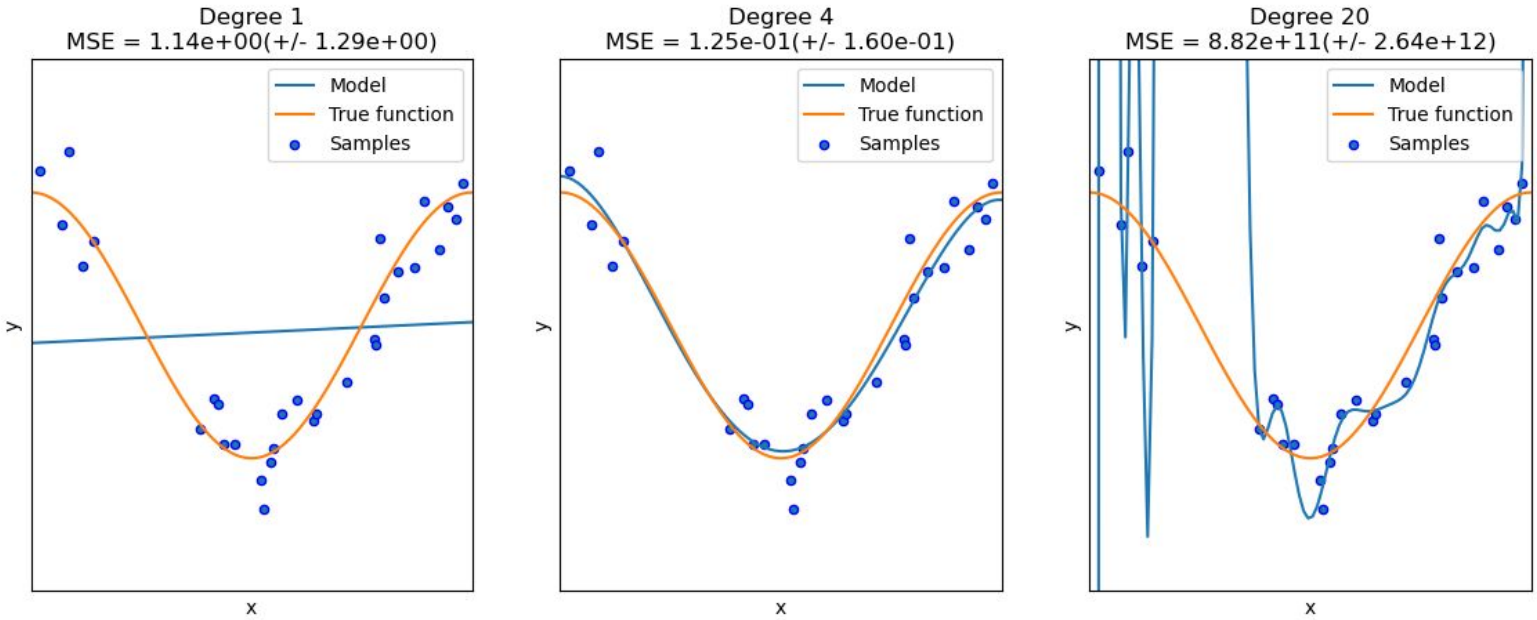
$$y(x, \mathbf{w}) = w_0 + w_1 x + w_2 x^2 + \ldots + w_p x_p^p = \sum_{j=0}^{p} w_j x^j$$

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





The order of the polynomial impacts fit



Example in Discussion

Probabilistic models for linear regression

- Maximum Likelihood Estimation (MLE) for Linear Regression
- A probabilistic framework to estimate parameters of a linear regression model
- We find an optimal way to fit a *distribution* to the data
- We look for the parameter values of a statistical model that maximises the likelihood of observing the given data distribution
- What follows if a brief introduction we will build on in a later lecture
- Likelihood estimation is a stepping stone towards Bayesian modelling covered later

Maximum Likelihood: A coin example

cuemath.com

- Flip a coin: Number of heads (N_H) number of tails (N_T)
- Goal: What is the probability of heads if we flip again
- The behaviour is summarized with a parameter θ, the probability of heads
- Data of flips D= (x^1,...,x^{100}) are independent Bernoulli random variables with parameter θ
  - Individual flips are independent and identically distributed (i.i.d.)
- Likelihood L(θ) function is: $L(\theta) = p(\mathcal{D}) = \theta^{N_H}(1-\theta)^{N_T}.$
-
- L(θ) is usually small so we work with the log: $\ell(\theta) = \log L(\theta) = N_H \log\theta + N_T \log(1-\theta)$
- We now wish to choose a θ which maximises $\ell(\theta)$. For our coin example:

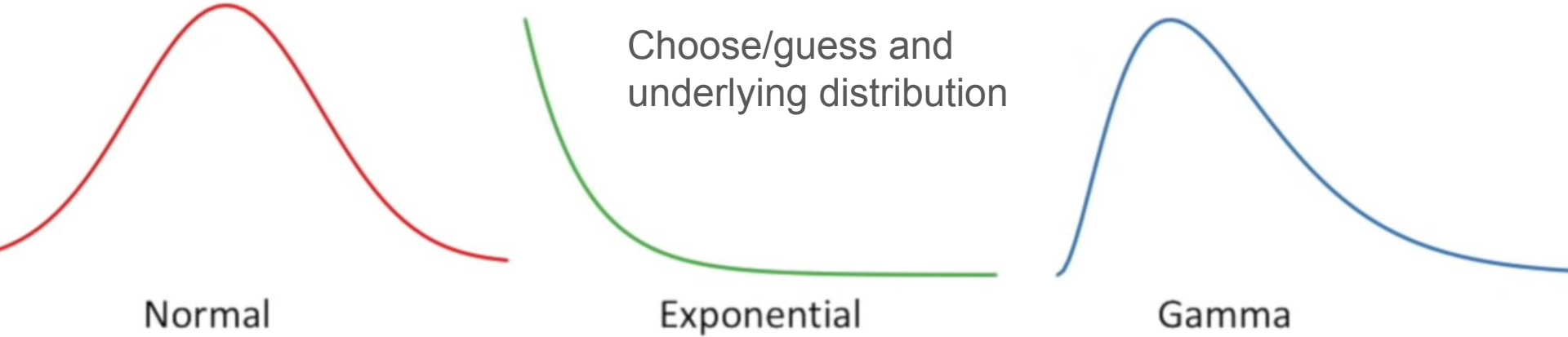
$$\begin{aligned}\frac{\mathrm{d}\ell}{\mathrm{d}\theta} &= \frac{\mathrm{d}}{\mathrm{d}\theta}\left(N_H \log\theta + N_T \log(1-\theta)\right) \\ &= \frac{N_H}{\theta} - \frac{N_T}{1-\theta}\end{aligned}$$

- Set this to zero we have the maximum likelihood estimate: $\hat{\theta}_{\mathrm{ML}} = \dfrac{N_H}{N_H + N_T}$

Maximum Likelihood Estimation

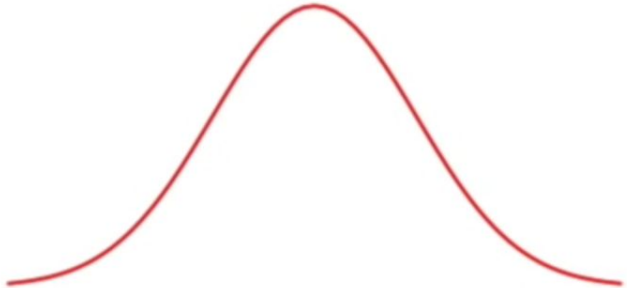
Data: Weight of mice

Maximum Likelihood Estimation

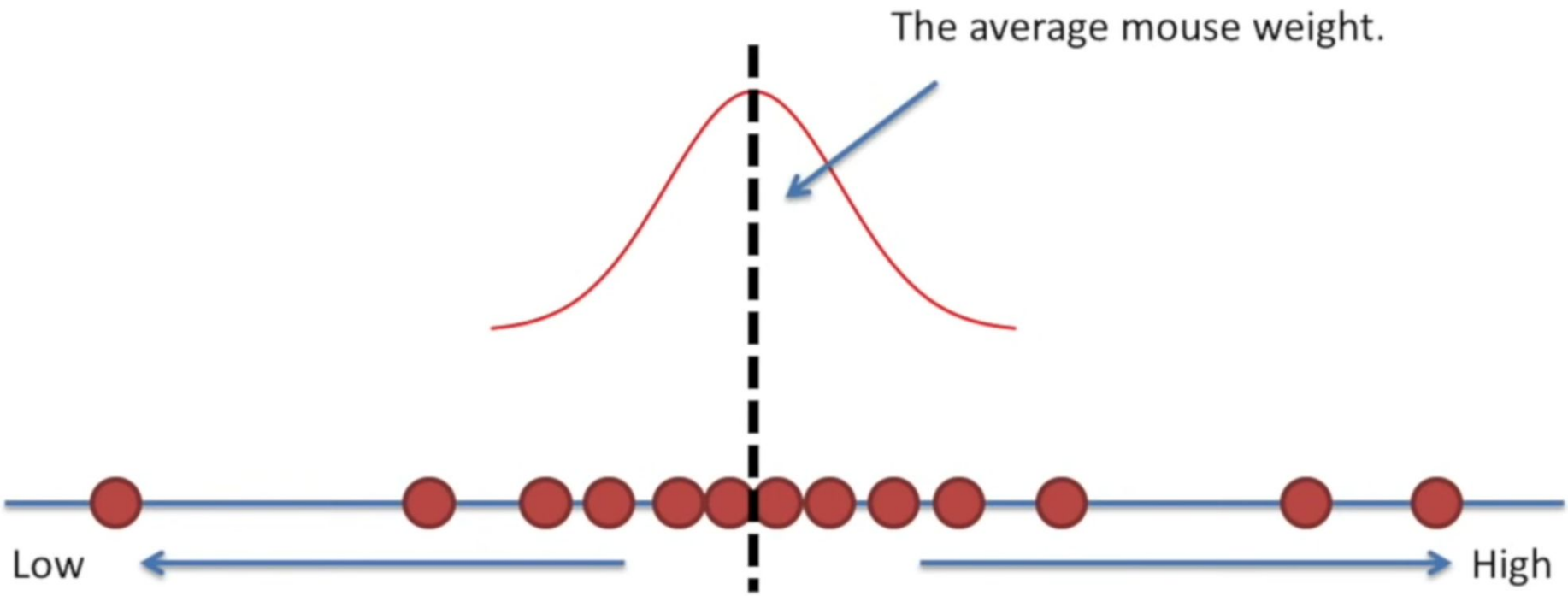


Maximum Likelihood Estimation

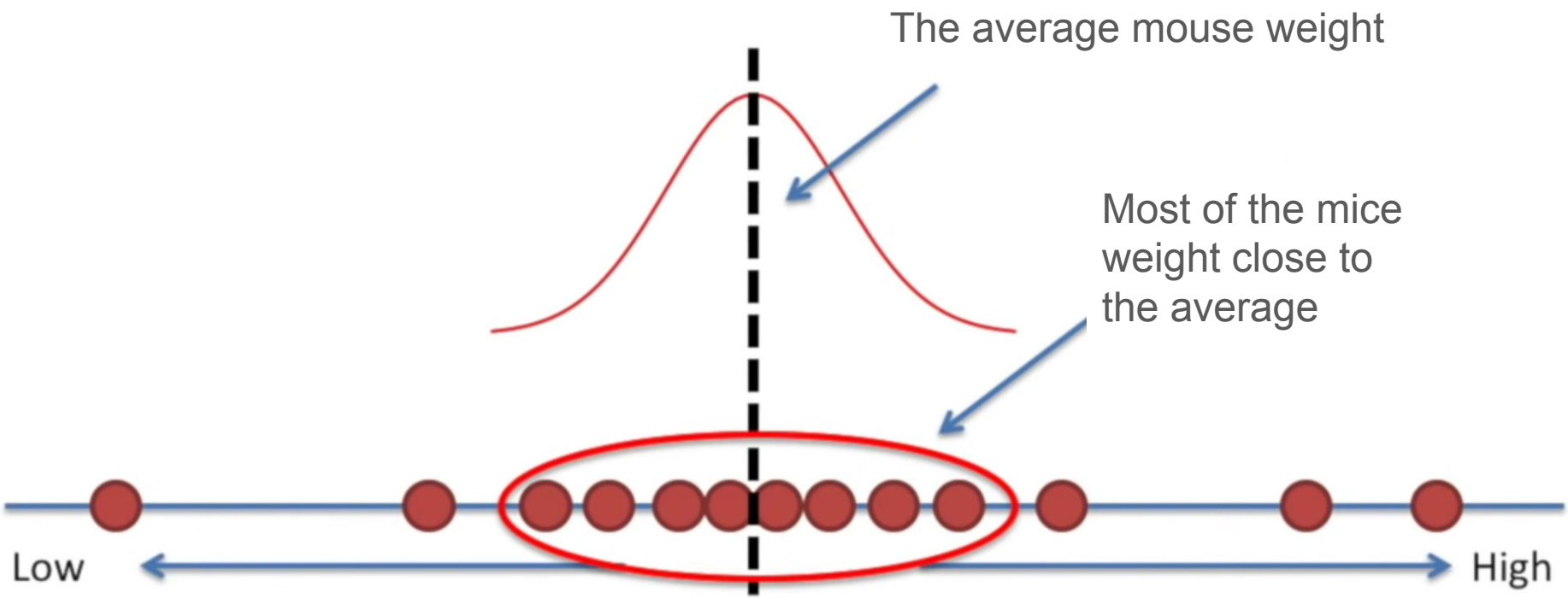
Let’s try a ‘normal’ distribution



Maximum Likelihood Estimation

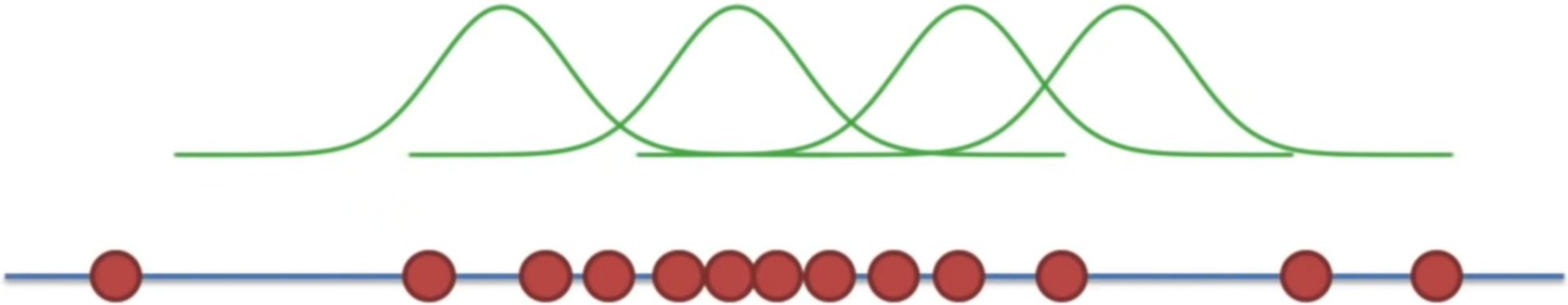


Maximum Likelihood Estimation



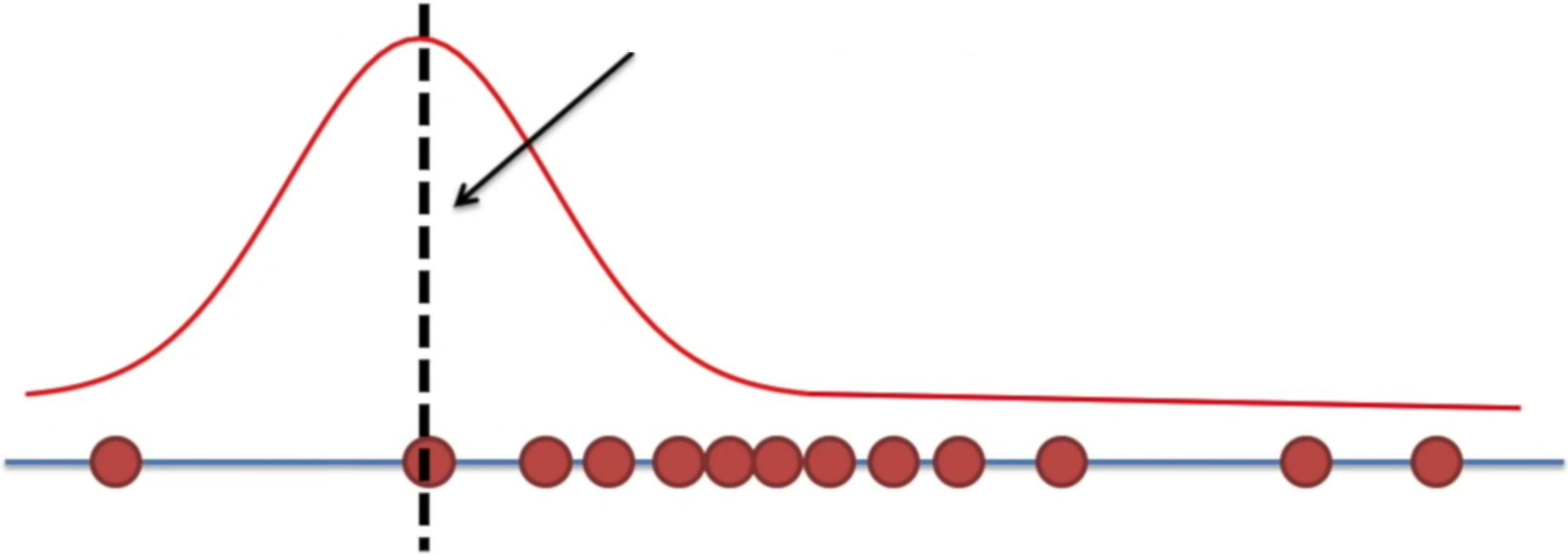
Maximum Likelihood Estimation

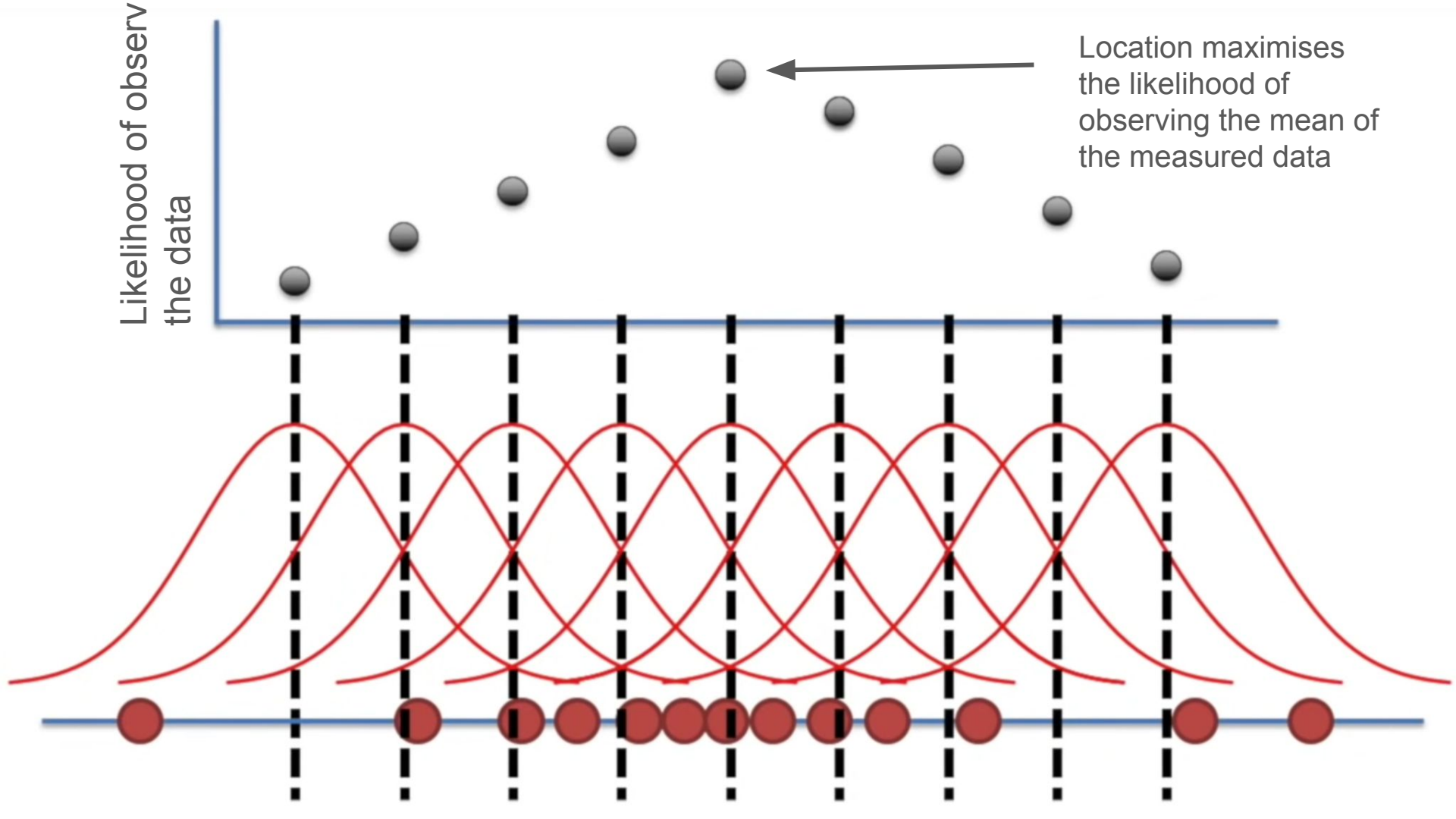
Once shape is determined, we must determine the optimal location that maximises the probability of being similar to the observations



Maximum Likelihood Estimation

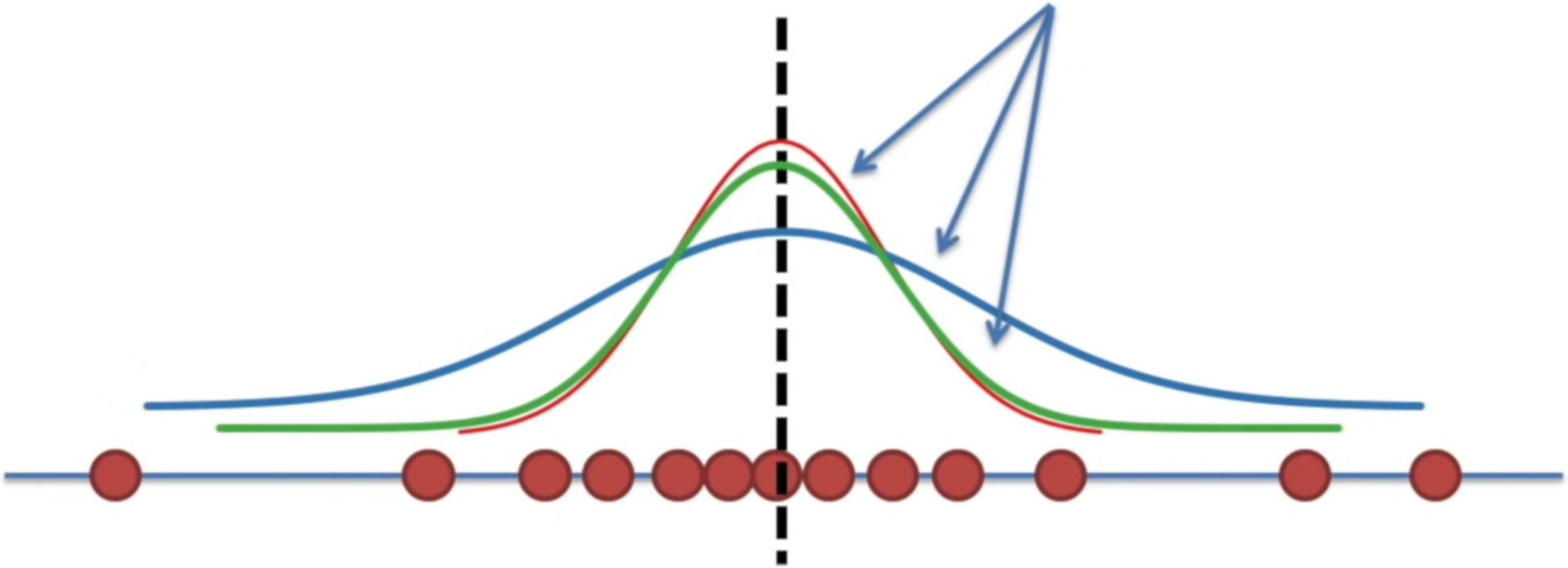
Most of the measured values
should be near the average

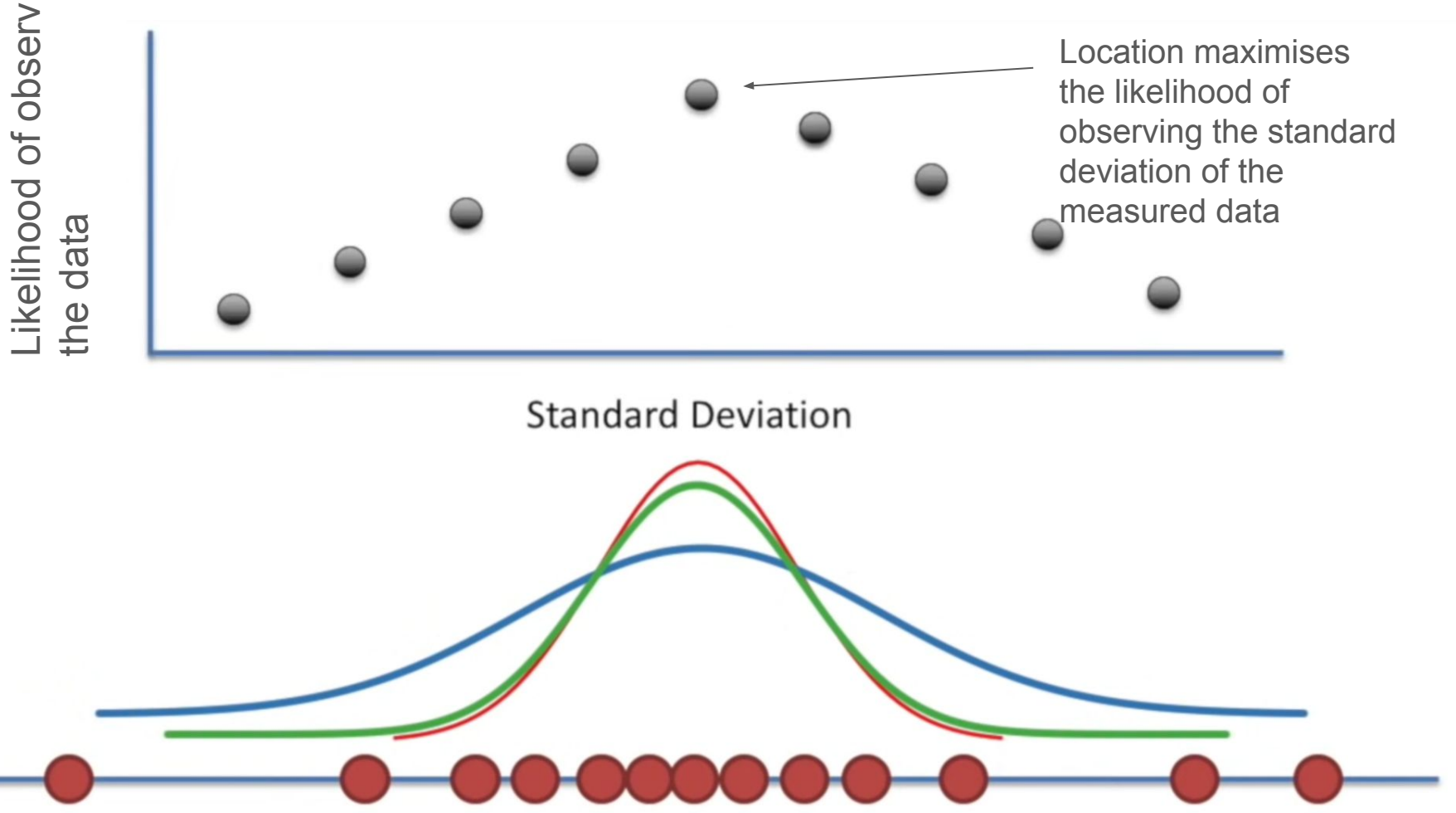



Likelihood of observ
the data
Location maximises
the likelihood of
observing the mean of
the measured data

Maximum Likelihood Estimation

Now we need the ‘maximum likelihood’ of the standard deviation




Likelihood of observ
the data
Location maximises the likelihood of observing the standard deviation of the measured data
Standard Deviation

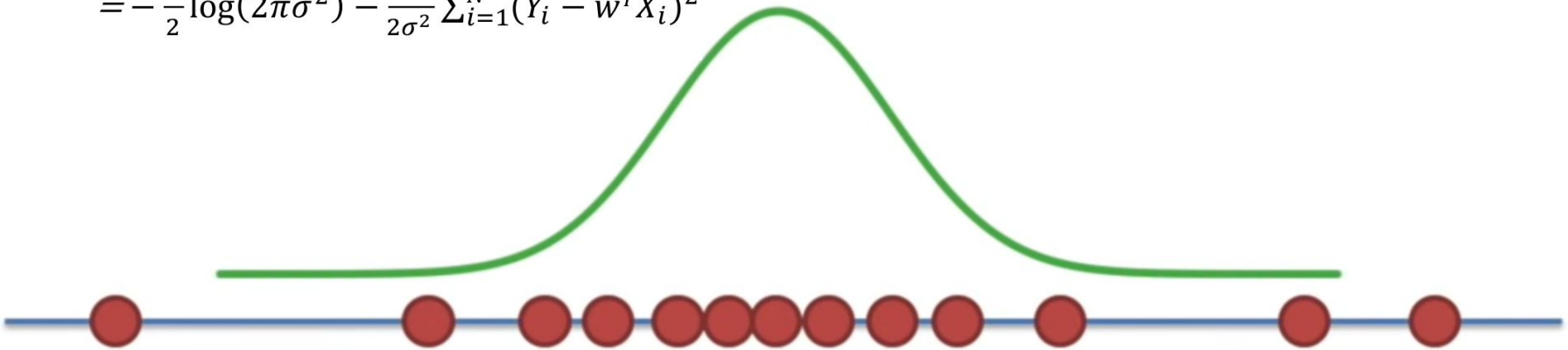
Maximum Likelihood Estimation

Here, θ is the likelihood

$$L(\theta) \triangleq \log L(\theta|X) = \sum_{i=1}^{N} \log L(\theta|X_i)$$

$$= \sum_{i=1}^{N} \log\left[\frac{1}{\sqrt{2\pi\sigma^2}} \times e^{-\frac{(Y_i - w^T X_i)^2}{2\sigma^2}}\right]$$

$$= -\frac{N}{2}\log(2\pi\sigma^2) - \frac{1}{2\sigma^2}\sum_{i=1}^{N}(Y_i - w^T X_i)^2$$



Regression key concepts

Data fits – is linear model best (model selection)?

- Simple models may not capture all the important variations (signal) in the data: underfit
- More complex models may overfit the training data (fit not only the signal but also the noise in the data), especially if not enough data to constrain model
- Bias-variance trade-off is a key concept

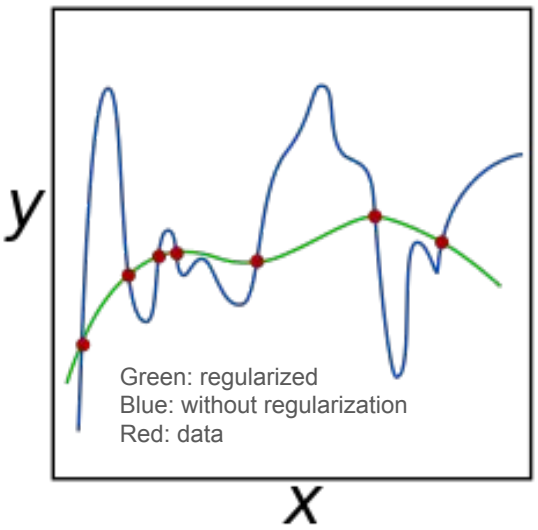
One method of assessing fit: test generalization = model's ability to predict the held out data

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

Regularization:
Curbing model complexity with the loss function

- Regularization is a process where the model has to choose what to emphasize
- The concept is used in the cost function, adding a ‘penalty’ term for more complicated models, for example:
  - A penalty for having a higher order polynomial versus a smaller one

wikipedia



Regularization: Three main types

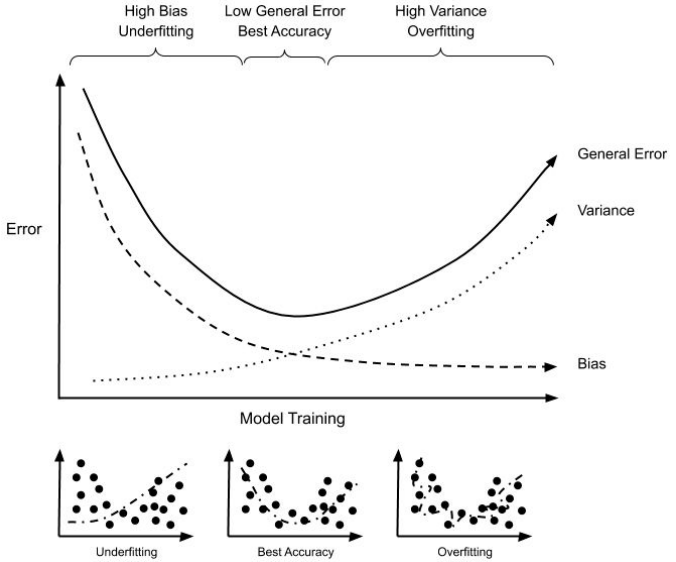
- Regularization methods are techniques that are used to calibrate ML models so as to minimize the adjusted loss function and prevent overfitting or underfitting
- Recall, in the absence of perfect data, simple models can have better ability to generalise (perform well on unseen data)
- Two main regularization techniques:
  - Lasso regression (L1)
  - Ridge regression (L2)
  - Elastic Net: Combination of L1 and L2
- We will come back to this concept for more ‘complicated’ cases in deep learning

L2: “squared magnitude” of the coefficient as a penalty term

$$\text{Cost} = \underbrace{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}_{\text{MSE}} + \underbrace{\lambda \sum_{i=1}^{m} w_i^2}_{\text{Penalty term}}$$

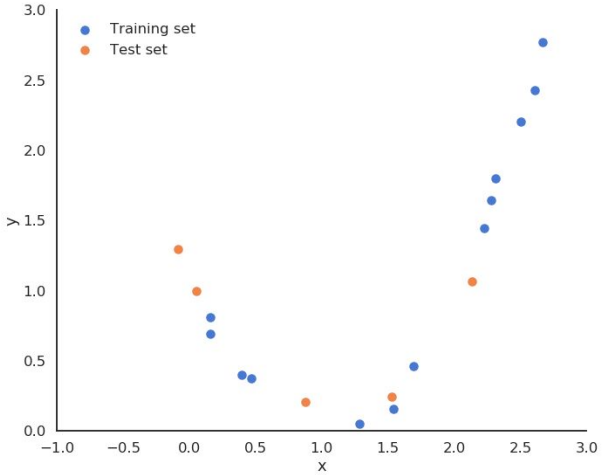
Optimization and generalisation

- We have now gone over a few types of models: Linear, polynomial, probabilistic and ways to train
- Recall, when we train a model, e.g. the linear, polynomial of various orders etc, we have a hypothesis of a representation of the ‘true’ function our samples come from in that model
- A **wrong** approach to selecting the ‘best’ model is to simply compare the errors
- Simply comparing errors least to overfitting



Generalisation: Test on held-out data

- We ‘hold out’ some of the data
- Train model on the ‘training data’, assess the convergence with the test data
- Can use additional ‘holdout’ set not used to select/tune the model
- Popular metrics: R2, RMSE

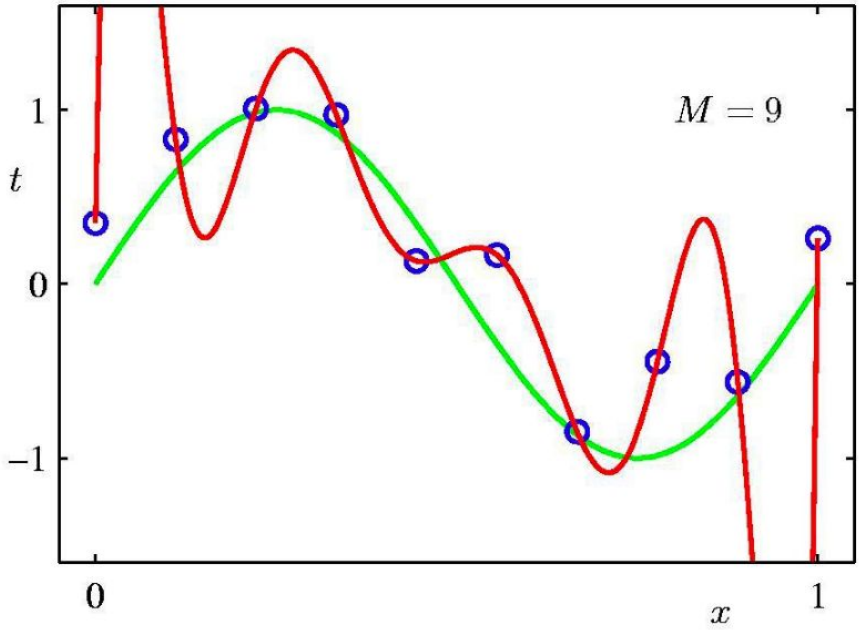


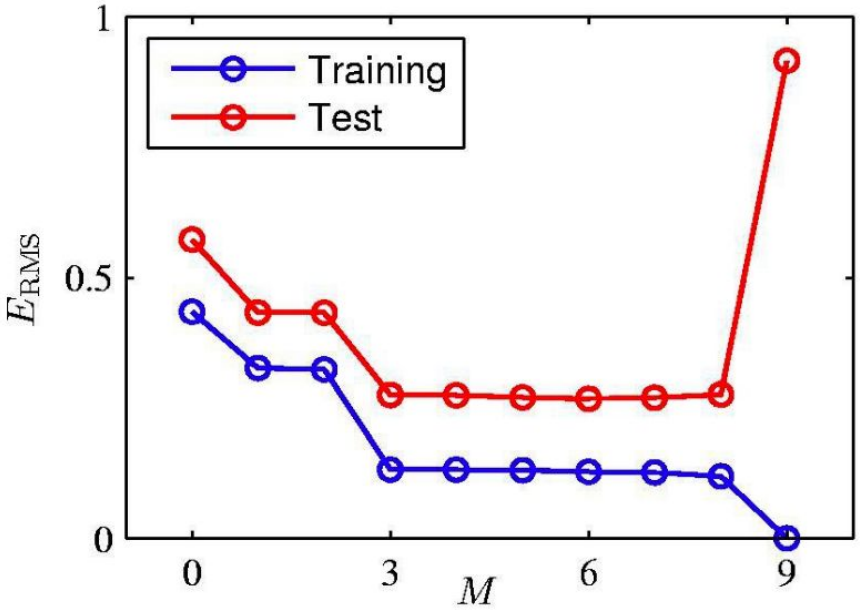
Example: constructing_polynomial_regression.ipynb

Bishop

Comparing test/train error over the order of a polynomial

- Zero Training Error



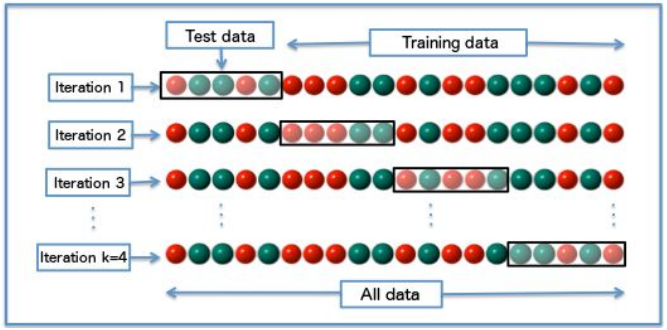


Root-Mean-Square (RMS) Error: $E_{\mathrm{RMS}} = \sqrt{2E(\mathbf{w}^\star)/N}$

RMSE tells us how concentrated the data is around the line of best fit.

Cross-validation

- Cross validation is used to compare models and prevent overfitting.
- A resampling procedure to help the model to generalize well
- Has a single parameter called k for the number of partitions
- Procedure for k-fold cross validation:
  1. Randomize the dataset and create k equal size partitions
  2. Use k-1 partitions for training the model
  3. Use the k^{th} partition for testing and evaluating the model
  4. Record the evaluation scores (such as MSE, R^2) in each iteration
  5. iterate k times with a different subset reserved for testing purpose each time.
  6. Select the model with the highest performance score and average the evaluation scores.



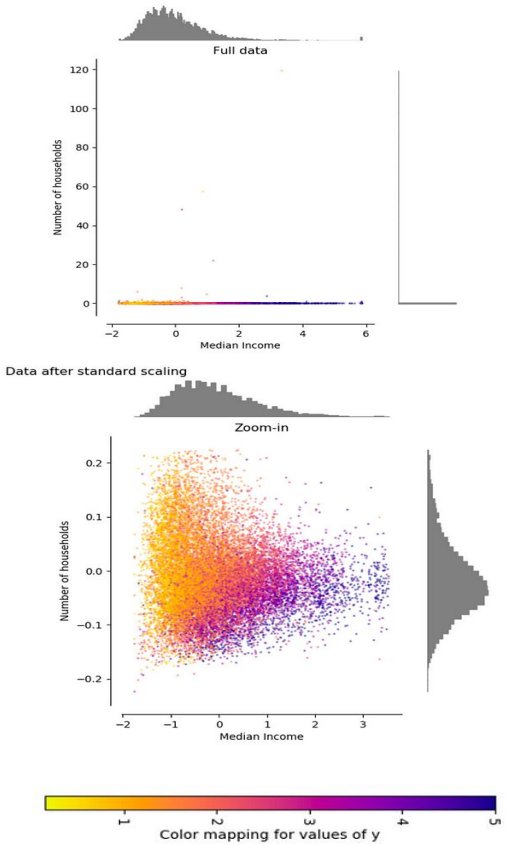
- Some commonly used variations on cross-validation are stratified k-fold, leave-one-out, and repeated k-fold are available in scikit-learn.

Optimisation: Standardizing

- Pre-processing carefully can help the optimization of the model: Often overlooked!
- **Standardizing** (or Z-score normalization) refers to rescaling features to have the properties of a normal distribution. This is not only important for features with different units, but also for many algorithms that assume an underlying normal distribution (Euclidian distance measures). The mean (μ) and standard deviation (σ) should be 0 and 1 respectively. Scores (or z-scores) for the samples "x" are:

$$z = \frac{x - \mu}{\sigma}$$

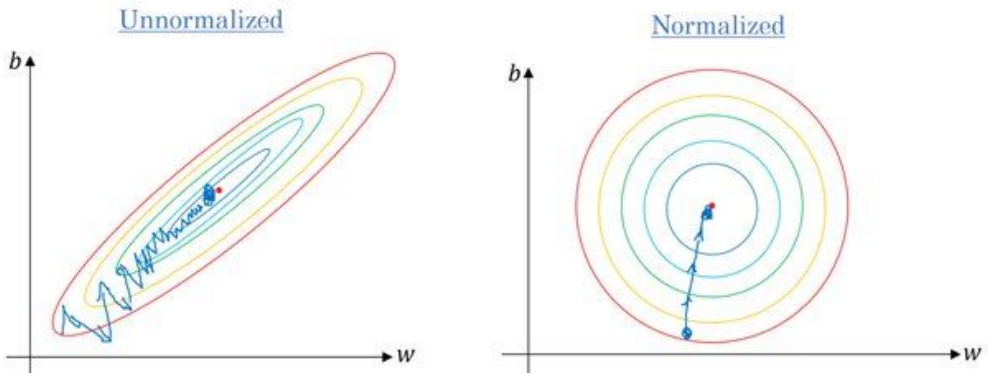
-



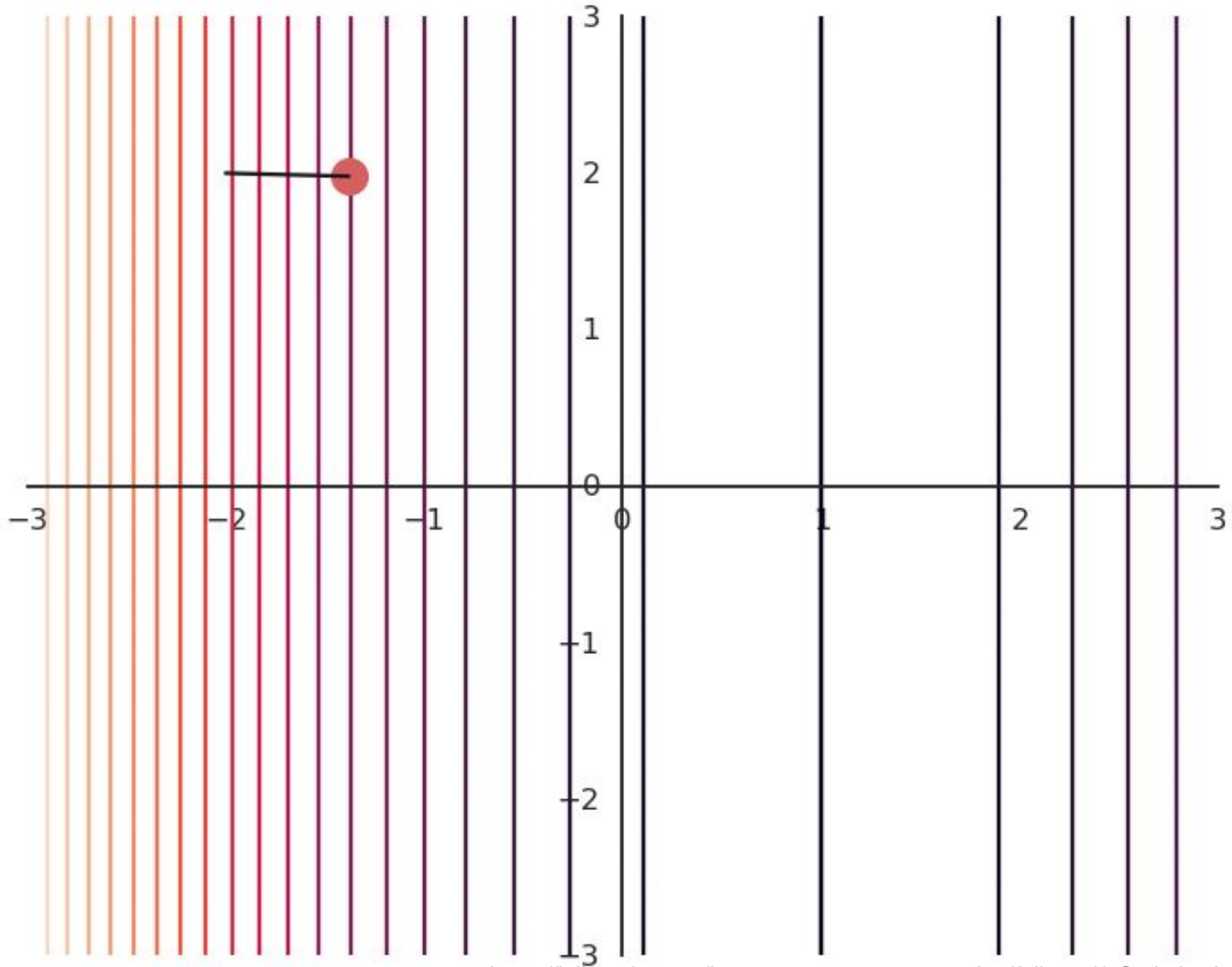
Optimization: Min-Max Scaling

- **Normalization/Min-Max scaling** does not center the data around zero (the name is misleading), but scales the data to a fixed range e.g. between 0 and 1. A bounded range will end up with smaller standard deviations. For the data x:

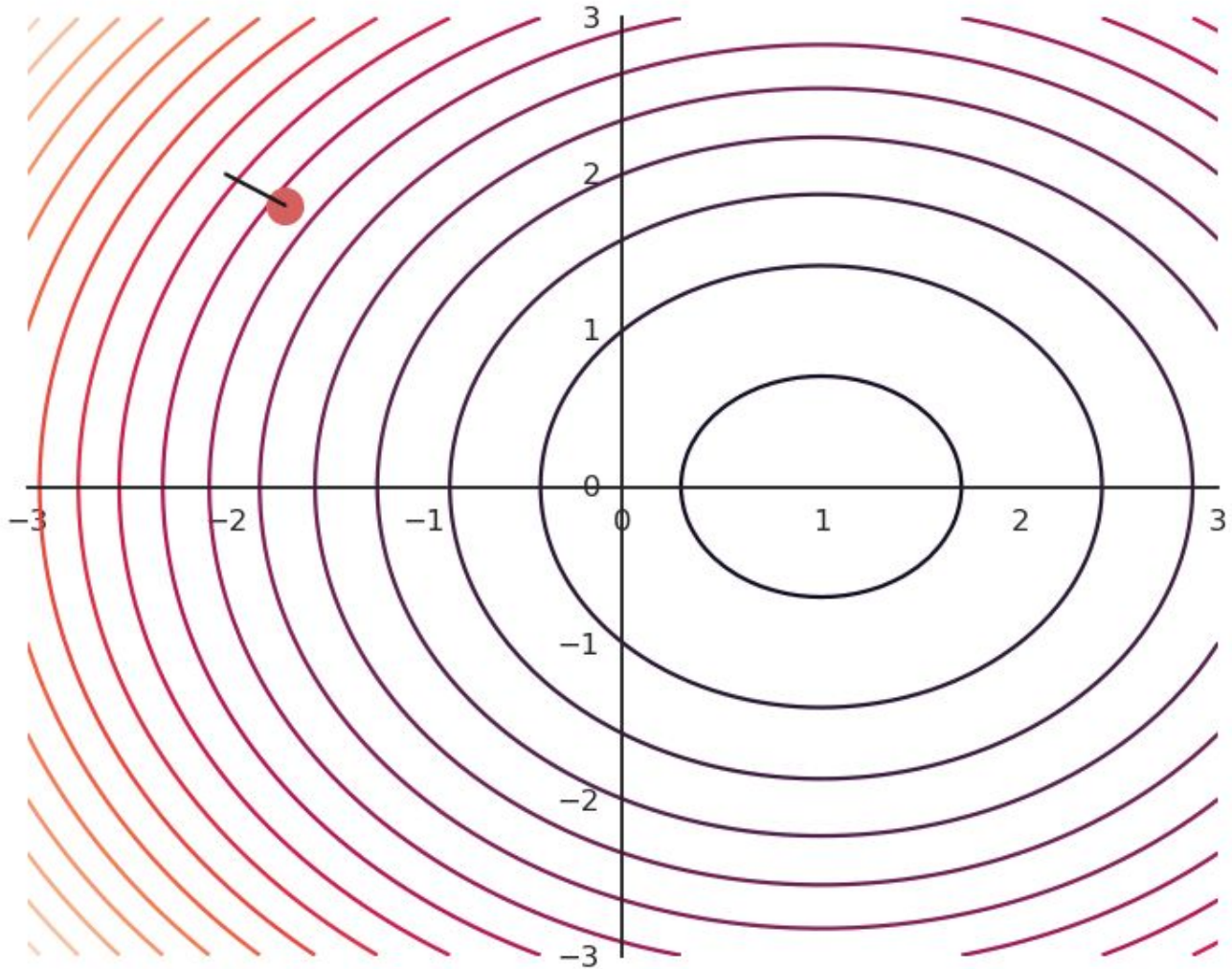
$$x_{norm} = \frac{x - x_{min}}{x_{max} - x_{min}}$$



quora.com



https://julien-vitay.net/lecturenotes-neurocomputing/2-linear/1-Optimization.html



https://julien-vitay.net/lecturenotes-neurocomputing/2-linear/1-Optimization.html