Artificial Intelligence 2: Maximum Likelihood (I)

Shan He

School for Computer Science
University of Birmingham

Outline of Topics

Questions

- Question 1: Given some observations (observed data points), what is the simplest mathematical (statistical) model you can build to describe the observations?
- Question 2: If we know the observations from some probability distribution, e.g., normal distribution, what is the best way to describe the observations

- Question 1
- Answer 1: To describe or summarise the data, we can use **Descriptive Statistics:** a summary that quantitatively describe our data in hand, the data we currently have. The simplest method is called Univariate analysis
  - Central tendency: **expectation** (mean), median, mode, etc.
  - Dispersion: the range and quartiles of the dataset,
  - Spread: **variance** and standard deviation
  - Shape of the distribution: skewness and kurtosi
- Question 2
- Answer 2: In this case, we should use probability distribution function as the statistical model. now the task is to estimate the parameters of the probability distribution function.
- Today we are going to learn systematic statistical inference method called maximum likelihood estimation (MLE) for parameter estimation.

Balls in urns

Example: Suppose we have an urn that contains only black and white balls, but we have no idea about the proportion of each. You pull a number of balls with replacement and get the sequence $\boldsymbol{x}$: BBWBWWBWWWBW.

Question: what is the proportion of black balls that is most likely to draw that sequence?

- To give you the idea behind MLE let us look at an example. Suppose we have an urn that contains only black and white balls, but we have no idea about the proportion of each.
- You pull a number of balls with replacement, that is, after drawing a ball, you put it back to the urn, so that the proportion of black and white balls does not change.
- From a statistical perspective, when we sample with replacement, the sample values, in our case, the colours are **independent and identically distributed** (the so-called IID) . Practically, this means that the ball we pull on the first one doesn't affect what we pull on the second.
- Let's say the sequence $\boldsymbol{x}$ is BBWBWWBWWWBW. The question is, given the observation $\boldsymbol{x}$, **(READ SLIDES)**

Formulating the urn problem

Denote the proportion of black ball, or the probability of drawing a black ball as θ, we know the probability follows the **Bernoulli distribution** : $P(B) = \theta$ and $P(W) = 1 - \theta$ for a white ball.

Since each draw is I.I.D, we calculate the probability of getting that sequence $\boldsymbol{x}$ as the product of the probabilities of each of the individual draws, which is equivalent to the :

$$L(\theta|\boldsymbol{x}) = \prod_{i=1}^{N} \begin{cases} \theta & \text{if } x_i = B \\ 1-\theta & \text{if } x_i = W \end{cases}$$

This probability is called **likelihood** function L, which is a function of θ, given the samples (observation) $\boldsymbol{x}$.

Likelihood: The probability of observing your data given a particular model, i.e., the Bernoulli distribution. The question is how to find θ.

- The sequence $\boldsymbol{x}$ is a Bernoulli process, and we know $\theta = 5/12$ since we saw 5 black balls out of 12 observations. We can easily obtain the value of θ because this problem is simple. However, do we have a more general method for more complex problems, for example, high dimensional samples x that are not binary?
- To derive a more general method, we first need to formulate the problem as a mathematical model.
- Let us denote the proportion of black ball, i.e., the parameter we want to estimate, as θ
- This θ is also equivalent to the probability of drawing a black ball, i.e.,$P(B) = \theta$. Now, for drawing a white ball, the probability is $P(W) = 1 - \theta$.
- Since each draw is Independent and identically distributed (I.I.D), we can calculate the probability of getting that sequence $\boldsymbol{x}$ as the product of the probabilities of each of the individual draws
- This probability is called **likelihood** function L, which is a function of θ, given the samples $\boldsymbol{x}$,
- Informally, a Likelihood function measures the goodness of fit of the function and the observation by holding the observation $\boldsymbol{x}$ constant and varying θ

Likelihood function: general definition

Let $X_1, X_2, X_3, ..., X_n$, denoted compactly using a vector $\boldsymbol{X} = (X_1, X_2, X_3, ..., X_n)$ be a random sample from a distribution with a parameter θ. Suppose we have observed that $X_1 = x_1, X_2 = x_2, X_3 = x_3, ..., X_n = x_n$, denote a vector $\boldsymbol{x} = (x_1, x_2, x_3, ..., x_n)$, we can define the likelihood function as

1. If X_i's are discrete:

$$L(\theta|\boldsymbol{x}) = L(\theta|x_1, x_2, \ldots, x_n) \tag{1}$$
$$= P_{\boldsymbol{X}}(\boldsymbol{x}; \theta), \tag{2}$$

   where $P_{\boldsymbol{X}}(\boldsymbol{x}; \theta)$ is the PMF of $\boldsymbol{X}$ parametrised by θ

2. If X_i's are continuous:

$$L(\theta|x) = L(\theta|x_1, x_2, \ldots, x_n) \tag{3}$$
$$= f_{\boldsymbol{X}}(\mathbf{x}|\theta) \tag{4}$$

Note: In general, θ can be a vector, $\boldsymbol{\theta} = (\theta_1, \theta_2, \ldots, \theta_k)$.

- Let's define the likelihood function more formally, **READ SLIDES**
- Please note that, in some more complicated distributions, there are multiple parameters θs'. Therefore, we should use a vector of k parameters, from θ_1 to θ_k

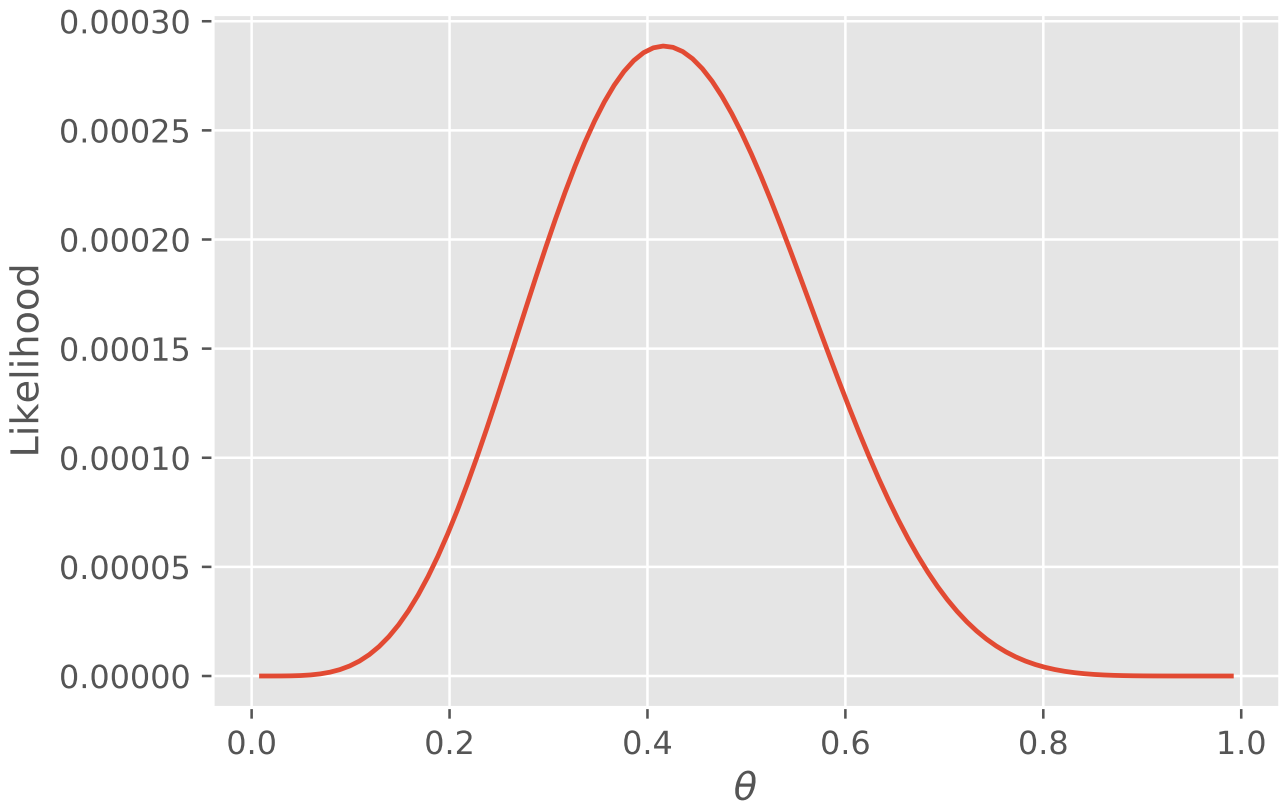

Figure: Likelihood function $L(\theta|\boldsymbol{x})$, $\theta \in [0, 1]$

- Note that the likelihood is not a probability function - the area under the likelihood curve does not have to sum to one.
- Let's try some examples using Python

Difference between the Probability and Likelihood

Probability: **a number** $p \in [0, 1]$ between 0 to 1 to describe how likely an event is to occur, or how likely it is that a proposition is true, assuming we know the distribution of the data.

Likelihood: **a function** that measures the goodness of fit of a statistical model to a sample of data for given values of the unknown parameters. It is a function of the unknown parameters, e.g., θ.

- So the fundamental difference between Probability and Likelihood is their aims.
- The aim of probability calculation is to find a number $p \in [0, 1]$ between 0 to 1 to describe how likely an event is to occur, or how likely it is that a proposition is true, assuming we know the distribution of the data.
- For example, in Lecture 4, we can calculate the probability of the location of the dot randomly placed on a line of length 1, i.e., $[0, 1]$. However, the calculation is based on the assumption that the line is a uniform distribution. Such assumption might not hold if we are not in an Euclidean space.
- In contrast, to define a likelihood function, we assume the we know the data follow some distribution but we do not know the parameters of the distribution function.
- We therefore define the likelihood function as a function that measures the goodness of fit of the distribution function to a sample of data for given values of the unknown parameters. A likelihood funtion is a function of the unknown parameters, e.g., θ.
- the aim of Likelihood calculation is to find the best distribution of the data.

Solving the urn problem: Maximum likelihood Estimator

Question: Given the above plot, what is this value θ for the urn?
Answer This is value of θ at the peak of the likelihood graph above, which is $\theta = 5/12$. Mathematically, this is defined as
Maximum likelihood estimation: Informally, based solely on the data, Maximum likelihood estimation searches the best parameters of a probability distribution that makes the data most likely.

- As we can see from the example, changing the value of θ will change the value of the function $L(\theta|\boldsymbol{x})$
- The bigger the value, the better the model fit
- In fact, after plotting, we can eyeball the graph and estimate the value of θ_{MLE} at the peak of the likelihood graph.
- But as mentioned, this rudimental method only works for simple and low-dimensional problems, e.g., 2 or 3 dimensional problems.
- We need a general and mathematical way to deal with more complex and high-dimensional problems.
- To achieve this, we first formulate the problem as an optimisation problem, which aims to find θ MLE. This can be written as the arguments of the maxima (argmax) of the likelihood function L.

The Maximum Likelihood Estimator (MLE)

Formally, Let $\boldsymbol{X} = (X_1, X_2, X_3, ..., X_n)$ be a random sample from a distribution with a parameter θ. Suppose we have observed the values of $\boldsymbol{X}$ as $\boldsymbol{x} = (x_1, x_2, x_3, ..., x_n)$, a maximum likelihood estimate of θ, denoted as $\hat{\theta}_{MLE}$ is a value of θ that maximises the likelihood function, i.e.,

$$\hat{\theta}_{MLE} = \underset{\theta}{\operatorname{argmax}}\, L(\theta|\boldsymbol{x}) \tag{5}$$

Maximum Likelihood Estimator (MLE)

A maximum likelihood estimator (MLE) of the parameter θ, denoted as $\hat{\Theta}_{MLE}$ is a random variable $\hat{\Theta}_{MLE} = \hat{\Theta}_{MLE}(\boldsymbol{X})$ whose value when $X_1 = x_1, X_2 = x_2, \ldots, X_n = x_n$ is given by $\hat{\theta}_{MLE}$.

- Here is the formal definition of Maximum Likelihood Estimation method.

Solving the urn problem: Maximum likelihood

Question: Given the optimisation problem as formulated in equation 5, how to solve it?

Option 1: Exhaustive search – only works for low dimensional problems such as this.

- Grid search: usually used for tuning hyper-parameters of a machine learning model. See Hyperparameter optimization - Grid Search

Option 2: Optimization algorithms – a more general way to solve the problem.

- One naive way is to sweep all the values of parameter θ to find one that generate the maximum likelihood function value, or to make the model mostly likely to generate the observation $\boldsymbol{x}$
- In fact, in computer science, this is called exhaustive search, i.e., systematically enumerating all possible values of θ and checking which gives us the maximum likelihood.
- A better way is called grid search, which is a popular method for tuning hyper-parameters of a machine learning model. Hyper parameters are parameters for controlling the learning process.
- For a grid search method, instead of evaluating all possible values, it exhaustively evaluate the likelihood function at **a set of** values of θ, which are spaced at small intervals, then pick the value of θ that gives you the maximum likelihood.
- However, this naive exhaustive search and grid search can not scale up for high dimensional problems. We need some general way to solve the problem, optimisation algorithms – in this lecture, we shall learn the most basic one, gradient descent, which your learn in AI 1.
- But before we learn those optimisation algorithms, we need to talk about a transformation of the likelihood function, called the cost function, which is an important concept in optimisation.

Cost functions

Cost function: A function that maps a set of events into a number that represents the "cost" of that event occurring. Also known as the **loss function** or **objective function**.

Cost function for likelihood: a general one-to-one mapping with likelihood – the negative logarithm of the likelihood function:

$$J(\theta, D) = -\log(L(\theta|D))$$

Question: Why use the negative logarithm of the likelihood function as the cost function, i.e., $-\log(L(\theta|D))$

- As the name suggest, a cost function is a function that maps a set of events into a number that represents the "cost" of that event occurring.
- The cost function, also known as the **loss function** or **objective function**, is the function that optimization works over – we want to find the optimal set of events to minimize the "cost" of those events.
- Specifically for our maximum likelihood problem, we define the cost function as the negative logarithm of the likelihood function
- But why should we use this cost function?
- We can maximise the likelihood function. Why should we transform it to the its negative logarithm?

Why should we use the cost function?

Question: Why use the negative logarithm of the likelihood function as the cost function, i.e., $-\log(L(\theta|D))$

Answers:

- **Convention**: By convention, many optimisation problems are minimisation problems
- **Convenience**: Taking the logarithm changes multiplication to addition, i.e., $\log(AB) = \log(A) + \log(B)$, which is easier to differentiate
- **Numerically stable**: Product of θ, which is a probability will converge quickly to zero, which might cause problems for computers who are limited by machine precision.

- The cost function is negative is because of convention. Many of these optimization techniques originated from physics, where the objective is to minimize the energy of a system. Therefore optimization will by default try to minimize functions.
- We take the logarithm of the likelihood is because it changes multiplication to addition, which makes the differentiation easier,
- Taking the logarithm also avoid problems caused by the limit of computer precision.
- This is because if we use the product of θ-s, of which each is a probability, i.e., the value is between 0 and 1, their product tend to converge quickly to zero.
- However, taking the logarithm changes the product to sums, which makes the numerical calculation more stable.
- This is very important if you are going to solve this maximum likelihood problem using a computer because the precision of that computer is limited, which does not distinguish a very small number from zero.

Cost function for likelihood

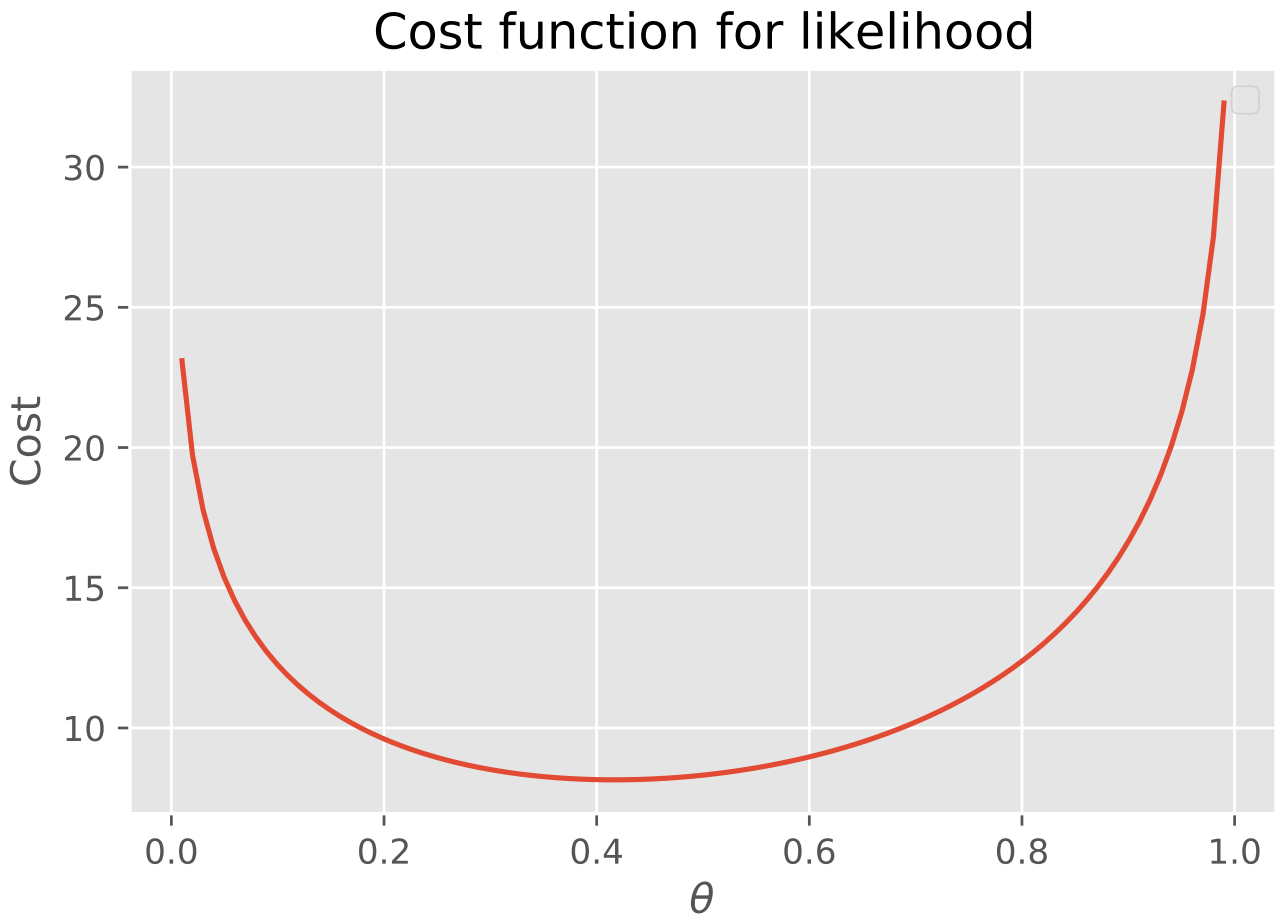

Figure: Cost function for likelihood

- As you can see from this graph, the minimum value is this cost function is obtained around 0.41
- Please note of the equivalence here: the point that was the maximum of the likelihood function is also the point that is the minimum of the cost function
- this is because the log-likelihood is a strictly increasing function of the likelihood, so decreasing the negative log-likelihood will always increase the likelihood.

Introduction to optimisation

Optimization: finding the best solution from among the set of all feasible solutions.

Optimisation procedure:

1. Constructing a Model ✓
2. Determining the Problem Type
3. Selecting an optimisation algorithm ✓

Optimization Taxonomy

- Now we shall look at optimisation. Many machine learning and inference problems can be formulated or re-formulated as optimisation problems.
- Optimisation a process to find the best solution from among the set of all feasible solutions.
- Basically, to apply optimisation to solve a real-world problem, we need to first construct an mathematical model, usually by making some assumptions.
- Generally, these assumptions are made to make our mathematical model tractable, but there is always an trade-off between tractability and complexity. The modelling itself is a vast and deep research area but we shall cover a tip of the iceberg in my lectures, including this lecture.
- The second step is to determine the problem type, whether it is a discrete problem, a continuous problem or a mix-variable problem? Is it constrained or unconstrained? Is the model involves uncertainty? etc.
- We then need to select optimisation algorithms according to the problem type.
- In this lecture, I will refresh you memory about gradient descent you

Introduction to optimisation

Machine learning $\Longrightarrow$ optimisation problems
Some examples:

- **Supervised learning**: Given some training data, we want to train a machine learning model to explain the data. The training process is essentially a process of finding a optimal set of parameters of this model and the optimality is defined by an objective function
- **Unsupervised learning**: Given some unlabelled samples, we aim to divide them into multiple clusters or groups, of which the samples in same group are as similar as possible but samples in different group are as different as possible.

- Let's talk about optimisation algorithms. Why they are important in machine learning?
- That's because many machine learning problems can be cast into optimisation problems.
- For example, supervised learning such as classification, is a process that, given some training data, we want to train a mathematical model to explain the data.
- The training process is essentially a process of finding a optimal set of parameters of this model, and the optimality is defined by an objective function – which is a optimisation problem.
- Another example is unsupervised learning, for example, clustering. **READ SLIDES** – this is can be naturally cast into a optimisation problem.
- There are also countless examples in AI, for example, Automated planning and scheduling, can be formulated as optimisation problems. From week 7, Dr Miqing Li will introduce some of the optimisation algorithms, esp. multi-objective optimisation algorithms .

The First-Order Optimality Condition

Optimisation: For a function $g(\boldsymbol{w})$ of N dimensional independent variables $\boldsymbol{w} \in \mathbb{R}^N$, the optimisation problem is

$$\underset{\boldsymbol{w}}{\operatorname{argmin}}\, g(\boldsymbol{w})$$

A $\boldsymbol{w}^*$ is the local minimum if it satisfies
First-order necessary condition for optimality:

$$\nabla_{\boldsymbol{w}} g(\boldsymbol{w}^*) = 0_{N\times 1} \tag{6}$$

The point which satisfy the condition is also called **stationary point**.
Note: A stationary point can be minimum, maximum or a saddle point

- An optimisation problem can be formally defined as (**Main screen)**
- This says formally 'look over every possible independent variable $\boldsymbol{w}$ and find the one that gives the smallest value of $g(\boldsymbol{w})$'.
- But how do we know $g(\boldsymbol{w})$ is the smallest?
- For functions with two or less variables, we can always plot, by hand or by a computer, then eyeball the plot to find the smallest value and the corresponding input.
- But for functions with more than two variables, we cannot visualise them. We need some analytical methods to do this – We have the first-order necessary condition for optimality
- That is, for this $\boldsymbol{w}^*$ to be a local minimum, the gradient, or the every partial derivative of g must be zero.
- Noticed that we call this the **necessary** condition for optimality, not **necessary and sufficient** condition, which means other points satisfy this condition but are not local minima, for example maxima and saddle points, which I shall explain later.
- We collectively call the points who satisfy the condition as **stationary points**. The condition is "first-order" because we only use the first order derivative.

Saddle Point

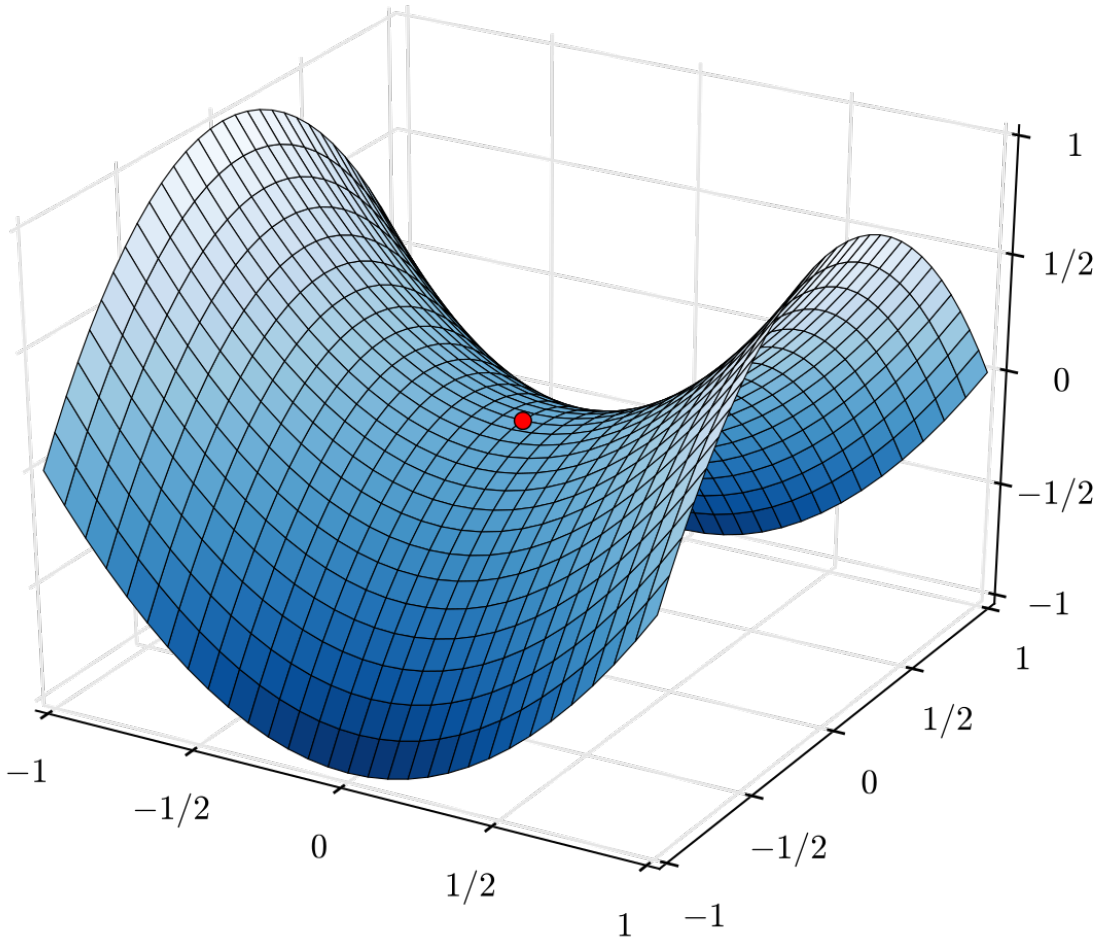

Figure: Saddle point By Nicoguaro - Own work, CC BY 3.0

a saddle point is also called a minimax point. As illustrated in this figure, this red point on the surface is the saddle point. Its slopes (derivatives) in orthogonal directions are all zero (a critical point), but it is neither a local maximum nor a local minimum of the function

The First-Order Optimality Condition

The equation of first-order necessary condition (equation 6) can be written as a system of N first order equations:

$$\frac{\partial}{\partial w_1} g(\boldsymbol{w}^*) = 0$$
$$\frac{\partial}{\partial w_2} g(\boldsymbol{w}^*) = 0$$
$$\vdots$$
$$\frac{\partial}{\partial w_N} g(\boldsymbol{w}^*) = 0$$

Question: can we solve this system of N first-order equations to identify the minimum points of the function g?

- Equation 6 can be written out one equation at-a-time, and we can see that is exactly a system of N first-order equations
- It seems that if we can solve this system of N first-order equations, we can identify the minimum points of the function g. Is it true?
- The answer is no. This is because, first of all, it is difficult or even impossible to solve such system algebraically for 'closed form' solutions, esp. when the equations are non-linear.
- Secondly, as we just learned, the condition is just a necessary condition, the solutions also include many other stationary points such as maxima and saddle points.
- Because of the above two reasons, we need some iterative methods to solve the such a system to identify minimum points.
- OK, let me review gradient descent methods.

Gradient descent

Gradient descent: a first-order iterative optimization algorithm for finding a local minimum of a differentiable cost function.

Idea: to employ the **negative gradient** at each step to decreases the cost function.

Two ingredients: the negative gradient consists

- a direction – determined by the gradient at the point
- a magnitude, sometimes called step size

Intuition: start at any value of parameter θ, then change θ in the direction that decreases the cost function, and keep repeating until there is only the tiniest decrease in cost with each step.

- Now, let's introduce the first optimisation algorithm in this module – Gradient descent
- So, what is gradient descent?
- It is a first-order iterative optimization algorithm for finding a local minimum of a differentiable cost function.
- The essence of gradient descent is to employ the **negative gradient** at each step to decreases the cost function.
- As we learned, just like any vector, the negative gradient always consists
  - a direction – determined by the gradient at the point
  - a magnitude, usually called step size
- Let's go back to the cost function plot to explain it visually, which is better to get the intuition (**turn to page 14**)
- **Intuition**: you start at any value of parameter θ, then change your parameter in the direction that decreases the cost function, this can be obtained by computing the negative gradient, and keep repeating until there is only the tiniest decrease in cost with each step.

Gradient descent

Formally, we define the negative gradient of a cost function $J(\theta)$ as

$$-\nabla_\theta J = -\frac{dJ(\theta)}{d\theta},$$

then we need to choose a magnitude or step size parameter η (also called the learning rate) so that the update equation becomes:

$$\theta(t+1) = \theta(t) - \eta\nabla_\theta J(\theta(t)) = \theta(t) - \eta\frac{dJ(\theta(t))}{d\theta},$$

Question: we know that $J(\theta) = -\log(\theta^5(1-\theta)^7)$, apply the rules we learned to calculate the derivative.

- To formalise our intuition into mathematical equations, we define **the negative gradient** of a cost function $J(\theta)$ as the negative derivative of the cost function w.r.t θ, which gives us the direction.
- We then need to determine the **magnitude**, or the step size η, or the so-called learning rate to define the vector.
- Finally, we need the update equation to generate the new $\theta(t+1)$ is the sum of the current $\theta(t)$ and the negative gradient.

Gradient descent

Pseudo-code of gradient descent:

Initialisation: Start at any value of parameter θ

repeat

change the parameter θ in the direction that decreases the cost function $J(\theta)$

until the decrease in cost with each step is very small

end

- This simple gradient descent, which as mentioned, is a iterative method, can be summarised in this pseudo-code.
- Now, let's take a look at how to use Pythone to program gradient descent to solve maximum likelihood problem.