Calculus for Machine Learning

StartOnAI Programming and Mathematics Tutorial 3

1 Topic Summary

Calculus is the mathematical study of continuous change, a core concept of machine learning. With two major branches, differential calculus (the study of rates of change) and integral calculus (the study of infinitesimal quantities), it is used in virtually every method of machine learning, from regression to classification to dimensionality reduction.

Most, if not all of the operations and methods found in this tutorial will always be automatically performed by a computer in the context of machine learning, meaning details regarding manually performing such are *not* included in this tutorial. Manual examples are only provided for the sake of comprehension.

This tutorial will provide you with *only* the essentials of calculus necessary to embark further on your machine learning journey through our higher-level tutorials, provided that you have a sufficient mathematical background (up to and including precalculus; this tutorial also requires knowledge of linear algebra, which can be obtained through Tutorial 2). Rereading chapters is also advised due to the concise nature of this tutorial.

2 Main Concepts

2.1 Derivatives

Given the continuous function $f(x) = x^2$, we are tasked with finding the *average rate of change* from $x = a$ and $x = b$. To do this, we can just subtract $f(a)$ from $f(b)$ and divide that by $b - a$, leaving us with the slope formula

$$\frac{f(b) - f(a)}{b - a} \tag{1}$$

However, to find the *instantaneous rate of change* (that is, the rate of change at a single point) requires what is called a *limit*. Taking the limit of an expression involves finding what value the expression approaches as one or more variables approach a certain value(s). For instance, the limit of $f(x) = x^2$ as x approaches 3 is as follows:

$$\lim_{x \to 3} f(x) = 9 \tag{2}$$

If we apply the concept of a limit to our previously derived formula for the average rate of change by adding h, which represents an extremely small value, or *infinitesimal*, we arrive at the limit definition for the instantaneous rate of change at x (assuming that f is continuous or continuous at x), which is termed a *derivative*:

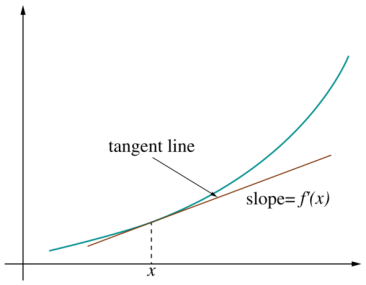
$$\lim_{h \to 0} \frac{f(x + h) - f(x)}{h} \tag{3}$$

If we substitute $f(x)$ for x^2 in the above formula, we will find that the derivative of $f(x) = x^2$ is $2x$, meaning that the rate of change at any point x is equal to $2x$. In proper terms, we would say that the derivative of $f(x)$ with respect to x (meaning that x is taken as a reference frame) is $2x$. In Lagrange's notation for derivatives, the above formula would be more compactly written as one of the following:

$$\frac{d}{dx} f(x) \text{ or } \frac{df}{dx} \tag{4}$$

We can also visualize a derivative geometrically as a line tangent to a single point of a function:

Figure 1: Visualization of a derivative as the line tangent to a single point of a function. (Lumen Learning)



Among the standard differentiation rules is the *chain rule*, which is generally useful in deep learning. The chain rule is used to calculate the derivatives of composite functions (functions composed of functions within other function(s)). Given a composite function $h(x) = f(g(x))$, we can calculate the derivative of $h(x)$ through the chain rule:

$$\frac{dh}{dx} = \frac{df}{dg} \cdot \frac{dg}{dx} \tag{5}$$

For instance, say that $f(x) = x^3$ and $g(x) = x^2$. To calculate the derivative of $h(x) = (x^2)^3$, we would apply the chain rule by differentiating every function with respect to its inner function (g(x) is the inner function in this case):

$$\frac{dg}{dx} = 2x; \frac{df}{dg} = 3(g(x))^2 \tag{6}$$

We can then substitute $g(x)$ for the actual function, leaving us with

$$\frac{dh}{dx} = 3(x^2)^2 \cdot 2x \tag{7}$$

which, after simplifying and evaluating, leaves us with the derivative of $h(x)$ with respect to x:

$$\frac{dh}{dx} = 6x^5 \tag{8}$$

The other differentiation rules are not quite as applicable as the chain rule in machine learning, but are nevertheless useful to know for the sake of future examples:

$$\text{Constant Rule: } \frac{d}{dx}[c] = 0 \tag{9}$$

$$\text{Power Rule: } \frac{d}{dx}x^n = nx^{n-1} \tag{10}$$

$$\text{Sum Rule: } \frac{d}{dx}[f(x) + g(x)] = \frac{d}{dx}f(x) + \frac{d}{dx}g(x) \tag{11}$$

$$\text{Product Rule: } \frac{d}{dx}[f(x)g(x)] = \frac{d}{dx}f(x)g(x) + f(x)\frac{d}{dx}g(x) \tag{12}$$

$$\text{Quotient Rule: } \frac{d}{dx}[\frac{f(x)}{g(x)}] = \frac{g(x)\frac{d}{dx}f(x) - f(x)\frac{d}{dx}g(x)}{[g(x)]^2} \tag{13}$$

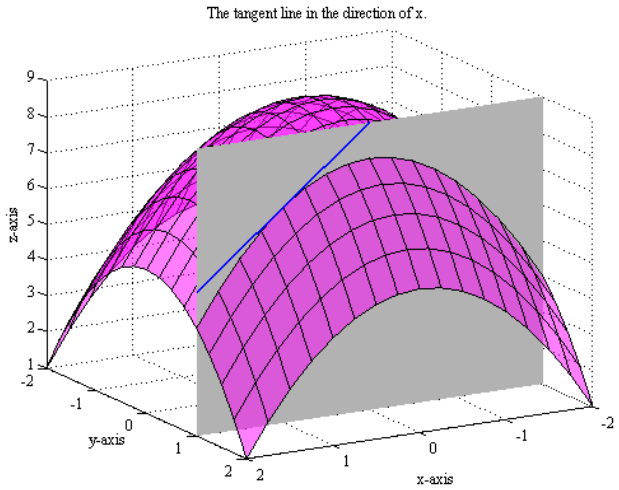
2.2 Partial Derivatives and Gradients

2.2.1 Gradients

A *gradient* is a vector that stores the *partial derivatives* of multivariable functions, which are functions whose inputs and/or outputs are composed of several numbers, allowing us to calculate the rate of change at a specific point on a curve for such functions.

To calculate the gradient of a multivariable function, we need to isolate each variable to determine how it impacts the output on its own by calculating a partial derivative for each variable. By iterating through each of the variables and calculating the derivative of the function after holding all other variables constant, we can calculate and store all of the partial derivatives of a given function in its gradient.

Figure 2: Visualization of a partial derivative of a multivariable function.



Given the multivariable function $f(x, z) = 2z^3x^2$, we can calculate its gradient by calculating each of the partial derivatives with respect to each of the input variables and storing those in a vector (note that partial derivatives are denoted with a "rounded d" symbol, del):

$$\nabla f(x, z) = \begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial z} \end{bmatrix} \tag{14}$$

To calculate $\frac{\partial f}{\partial x}$, we hold z fixed by swapping it with a constant b:

$$f(x, z) = bx^2 \tag{15}$$

We can then calculate the partial derivative of f with respect to x as we would for a single-variable function by using the power rule:

$$\frac{\partial f}{\partial x} = 2bx \tag{16}$$

Swapping out b for $2z^3$ leaves us with:

$$\frac{\partial f}{\partial x} = 4z^3x \tag{17}$$

By going through a similar process for $\frac{\partial f}{\partial z}$, we are left with the gradient of f:

$$\nabla f(x, z) = \begin{bmatrix} 4z^3x \\ 6x^2z^2 \end{bmatrix} \tag{18}$$

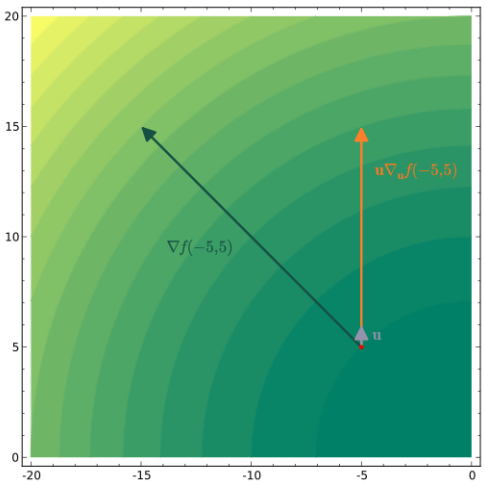
2.2.2 Directional Derivatives

An important property of gradients that is especially useful in deep learning is the fact that the gradient will always be the direction of steepest "ascent" in a multivariable function. That is, the gradient of a multivariable function at a given point will produce a vector pointing in the direction that will result in the greatest increase in the function's output.

However, if we want to "change directions" (that is, move in the direction of some vector v while moving through point x), we will need a *directional derivative*, which will allow us to calculate the instantaneous rate of change of a function at point x in direction of vector $\vec{v}$.

Intuitively, we can view directional derivatives as a tool allowing us to find the "steepness" of the "hills" of a function if we want to move in a direction other than the direction specified by the gradient.

Figure 3: Visualization of the gradient of $f(x, y) = x^2 + y^2$ scaled by the directional derivative in direction of $\hat{j}$, which is represented by the orange vector. (Bach, 2013)



The directional derivative is computed by taking the dot product of the gradient of function f and a unit vector $\vec{v}$ of "tiny nudges" representing the desired direction. The output of this calculation is a scalar number representing how much f will change if the current input moves with vector $\vec{v}$:

$$\nabla_{\vec{v}} f(x, y) = \begin{bmatrix} a \\ b \end{bmatrix} \cdot \begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{bmatrix} = \vec{v} \cdot \nabla f(x, y) \tag{19}$$

Another important property of gradients to note is that the gradient will always be zero at a local maximum or minimum of a function (points where the surrounding points are of lower/greater value than the point itself), which is quite useful for knowing if we get "stuck" during optimization.

The differentiation rules (sum rule, product rule, chain rule, etc) that we covered in the previous chapter still apply to partial derivatives. However, when we compute gradients involving vectors and matrices, we need to pay attention to our operations, as matrix multiplication is not commutative.

2.2.3 Gradients of Vector-Valued Functions

So far, we've covered partial derivatives and gradients of functions whose input space is multidimensional. We will now generalize the concept of the gradient to vector-valued functions (functions with an n-dimensional input and an m-dimensional output)

For a function $f : \mathbb{R}^n \to \mathbb{R}^m$ (a function that maps from an n-dimensional vector to an m-dimensional vector) and a vector $x = \begin{bmatrix} x_1, \ldots, x_n \end{bmatrix}^T \in \mathbb{R}^n$ (a vector of dimension n), the corresponding output vector is defined as

$$f(x) = \begin{bmatrix} f_1(x) \\ \vdots \\ f_m(x) \end{bmatrix} \in \mathbb{R}^m \tag{20}$$

Writing the vector-valued function in this way allows us to view said function as a vector of functions $\begin{bmatrix} f_1, \ldots, f_m \end{bmatrix}^T$, where the standard differentiation rules can be applied to every f_i in that vector.

Therefore, the partial derivative of a vector-valued function $f : \mathbb{R}^n \to \mathbb{R}^m$ with respect to x_i $(i = 1, \ldots, n)$ is given as the vector

$$\frac{\partial f}{\partial x_i} = \begin{bmatrix} \frac{\partial f_1}{\partial x_i} \\ \vdots \\ \frac{\partial f_m}{\partial x_i} \end{bmatrix} \tag{21}$$

The above is conceptualized as a vector containing the all of the partial derivatives for each f within the function vector representing our vector-valued function f with respect to x_i, a value within the vector x.

If we use the above to calculate the partial derivatives of f with respect to each value x_i within the vector x (which

of dimension n), we obtain the gradient of the vector-valued function f, which is an n by m matrix:

$$\nabla f(x) = \left[\frac{\partial f(x)}{\partial x_1} \cdots \frac{\partial f(x)}{\partial x_n}\right] = \begin{bmatrix} \frac{\partial f_1(x)}{\partial x_1} & \cdots & \frac{\partial f_1(x)}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_m(x)}{\partial x_1} & \cdots & \frac{\partial f_m(x)}{\partial x_n} \end{bmatrix} \tag{22}$$

The collection of all first-order (one iteration of differentiation) partial derivatives of a vector-valued function f is called the *Jacobian*.

2.2.4 Higher Order Derivatives

So far, we have covered gradients (i.e. first-order derivatives). Some machine learning methods and algorithms (e.g. Newton's Method for optimization) require higher order derivatives. Consider a function f that maps from $\mathbb{R}^{\nvDash}$ to $\mathbb{R}$ of two variables, x and y. We will use the following notation for higher-order partial derivatives and gradients:

- $\frac{\partial^2 f}{\partial x^2}$ is the second partial derivative of f with respect to x.
- $\frac{\partial^n f}{\partial x^n}$ is the nth partial derivative of f with respect to x.
- $\frac{\partial^2 f}{\partial y \partial x} = \frac{\partial}{\partial y}(\frac{\partial f}{\partial x})$ is the partial derivative obtained by first partial differentiating with respect to x and then with respect to y.
- $\frac{\partial^2 f}{\partial x \partial y}$ is the partial derivative obtained by first partial differentiating by y and then x.

The *Hessian* is the collection of all second-order partial derivatives.

If $f(x, y)$ is a twice (continuously) differentiable function, then

$$\frac{\partial^2 f}{\partial x \partial y} = \frac{\partial^2 f}{\partial y \partial x} \tag{23}$$

and the order of differentiation doesn't matter, and the corresponding *Hessian matrix* (a square matrix of second-order partial derivatives that describes the local curvature of a multivariable function at a given point) is symmetric:

$$H = \begin{bmatrix} \frac{\partial^2 f}{\partial x^2} & \frac{\partial^2 f}{\partial x \partial y} \\ \frac{\partial^2 f}{\partial x \partial y} & \frac{\partial^2 f}{\partial y^2} \end{bmatrix} \tag{24}$$

The Hessian is denoted as $\nabla^2_{x,y} f(x, y)$. Generally, for $f : \mathbb{R}^n \to \mathbb{R}$ (a function that maps from an n-dimensional space to the set of all real numbers), the Hessian is an $n \times n$ matrix.

2.2.5 Automatic Differentiation

Computation wise, a computer can find the exact value of the gradient of a function via *automatic differentiation*, a set of techniques to numerically evaluate the gradient by working with intermediate variables and applying the chain rule. Automatic differentiation applies a series of elementary arithmetic operations (addition, multiplication, elementary functions). By applying the chain rule to these operations, the gradient of quite complicated functions can be computed automatically. Automatic differentiation applies to general computer programs and has forward and reverse modes.

Figure 4: A simple graph representing the data flow from inputs x to outputs y. (Deisenroth, Faisal, Ong; 2020)

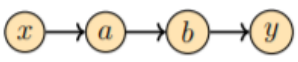


Figure 4 shows a simple graph representing the data flow from inputs x to outputs y via intermediate variables a and b ($x, y, a,$ and b can all be thought of as "functions").

If we want to find the derivative $\frac{dy}{dx}$, we would apply the chain rule:

$$\frac{dy}{dx} = \frac{dy}{db} \cdot \frac{db}{da} \cdot \frac{da}{dx} \tag{25}$$

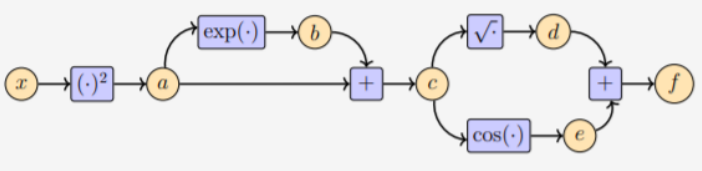
The reverse mode and forward mode differ in the order of multiplcation; In reverse mode, $\frac{dy}{db}$ and $\frac{db}{da}$ would be multiplied first, while $\frac{da}{dx}$ and $\frac{db}{da}$ would be multiplied first in forward mode. In the context of neural networks (a machine learning

method), the reverse mode is computationally significantly cheaper than the forward mode.

For instance, consider the function $f(x) = \sqrt{x^2 + exp(x^2)} + cos(x^2 + exp(x^2))$. If we were to implement function f in a program, we would use intermediate variables to save computation:

$$\begin{aligned} a &= x^2 \\ b &= exp(a) \\ c &= a + b \\ d &= \sqrt{c} \\ e &= cos(c) \\ f &= d + e \end{aligned} \tag{26}$$

Figure 5: A computation graph representing the order of computation among the intermediate variables. (Deisenroth, Faisal, Ong; 2020)



This is the same type of thinking process that occurs when applying the chain rule. Note that the preceding set of equations requires fewer operations than a direct implementation of the function. The set of equations that include intermediate variables can be thought of as a computation graph, a representation that is widely used in implementations of neural network software libraries.

We can directly compute the derivatives of the intermediate variables with respect to their corresponding inputs by recalling the definition of the derivative of elementary functions, leaving us with the following:

$$\begin{aligned} \frac{\partial a}{\partial x} &= 2x \\ \frac{\partial b}{\partial a} &= exp(a) \\ \frac{\partial c}{\partial a} &= 1 = \frac{\partial c}{\partial b} \\ \frac{\partial d}{\partial c} &= \frac{1}{2\sqrt{c}} \\ \frac{\partial e}{\partial c} &= -sin(e) \\ \frac{\partial f}{\partial d} &= 1 = \frac{\partial f}{\partial e} \end{aligned} \tag{27}$$

By looking at the computation graph, we can work backward from the output to compute $\frac{\partial f}{\partial x}$:

$$\frac{\partial f}{\partial c} = \frac{\partial f}{\partial d}\frac{\partial d}{\partial c} + \frac{\partial f}{\partial e}\frac{\partial e}{\partial c} \tag{28}$$

$$\frac{\partial f}{\partial b} = \frac{\partial f}{\partial c}\frac{\partial c}{\partial b} \tag{29}$$

$$\frac{\partial f}{\partial a} = \frac{\partial f}{\partial b}\frac{\partial b}{\partial a} + \frac{\partial f}{\partial c}\frac{\partial c}{\partial a} \tag{30}$$

$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial a}\frac{\partial a}{\partial x} \tag{31}$$

Substituting the results of the derivatives of the elementary functions leaves us with

$$\frac{\partial f}{\partial c} = 1 \cdot \frac{1}{2\sqrt{c}} + 1 \cdot (-sin(c)) \tag{32}$$

$$\frac{\partial f}{\partial b} = \frac{\partial f}{\partial c} \cdot 1 \tag{33}$$

$$\frac{\partial f}{\partial a} = \frac{\partial f}{\partial b} exp(a) + \frac{\partial f}{\partial c} \cdot 1 \tag{34}$$

$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial a} \cdot 2x \tag{35}$$

2.3 Integrals

In the past two chapters, we've been given a function f and asked what its derivative/gradient is. In this chapter, we'll explore the concept of *integration*, the inverse of differentiation.

Given a function $f(x)$, an *antiderivative* of $f(x)$ is any function $F(x)$ such that $\frac{d}{dx}F(x) = f(x)$. The most general antiderivative of $f(x)$ is called an *indefinite integral* and denoted:

$$\int f(x)dx = F(x) + C \tag{36}$$

where $\int$ is called the *integral symbol*, $f(x)$ is called the *integrand*, x is called the *integration variable*, and the $+C$ is called the *constant of integration*, since the derivative of any constant is 0.

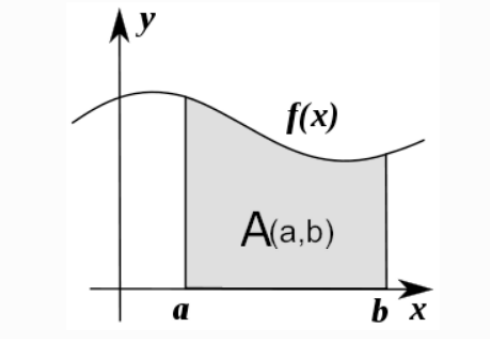
The process of finding the indefinite integral is called integration. If we use the above definition, we say that we are integrating $f(x)$ with respect to x. The dx in the above definition is a differential, which you can think of as a "closing parenthesis", with the integral symbol being the "opening parenthesis" for the integral.

If we apply the concept of an indefinite integral to a certain continuous interval, $[a, b]$, of a function f, we are left with the definition of a definite integral:

$$\int_a^b f(x)dx = F(b) - F(a) \tag{37}$$

To break down the above equation, which is known as the *fundamental theorem of calculus*, we can use the visual below:

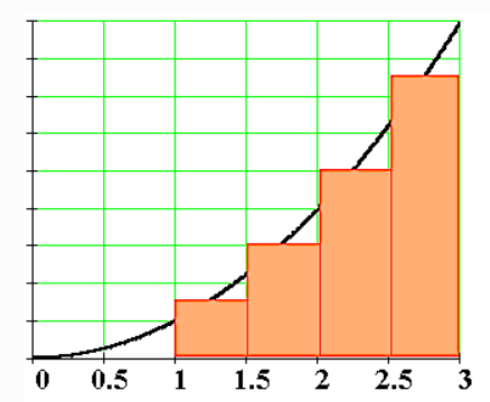
Figure 6: Visualization of a definite integral.



Intuitively, the value of $F(b)$ (the antiderivative of f with all variables substituted for b) represents the area under the curve of function f from the lower limit, 0, to the upper limit, b. Thus, if we subtract the value of $F(a)$ (the area under the curve from 0 to a) from $F(b)$, the resulting value will be the area under the curve from lower limit a to upper limit b.

Computation wise, we can "compute" (approximate very precisely) a definite integral in a program by splitting the region under the curve between lower limit a and upper limit b into tiny vertical strips of width h and taking the limit of the sum of the areas of those strips as h approaches 0.

Figure 7: Visualization of how one could "compute" a definite integral algorithmically.



2.4 Taylor and Maclaurin Series

A *sequence* is a list of numbers written in a specific order, which may or may not have an infinite numbers of terms. A term is denoted with a subscript (e.g. a_n), with the first term of a sequence being a_1.

If an infinite sequence approaches some value as n (the term number) approaches infinity, then we can say that that sequence *converges*. If the sequence does not approach any value and goes to infinity or negative infinity, the sequence can be said to *diverge*.

The concept that will eventually lead us to the main concepts of this chapter is the idea of a *series*, which is the sum of the terms of a sequence, which may or may not be infinite.

We use *summation notation*, $\sum_{i=a}^{n} x_i$, to represent this summation. The "i" represents the index of summation, the a(which can be any number) represent the lower limit of summation (think of it as a "starting point"), the n represents the upper limit of summation (think of it as a "stopping point"), and the "x_i represents an element of the sum.

One of the main concepts of this chapter, the *Taylor series*, is a representation of a function f as an infinite sum of terms. These terms are determined using derivatives of f evaluated at x_0.

The *Taylor polynomial* of degree n of $f : \mathbb{R} \to \mathbb{R}$ at x_0 is defined as

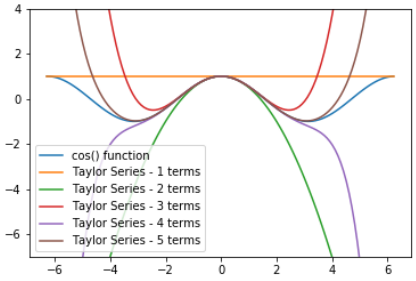
$$T_n(x) = \sum_{k=0}^{n} \frac{f^{(k)}(x_0)}{k!}(x - x_0)^k \tag{38}$$

where $f^{(k)}(x_0)$ is the kth derivative of f at x_0 (which we assume exists) and $\frac{f^{(k)}(x_0)}{k!}$ are the coefficients of the polynomial.

For a smooth function $f \in C^{\infty}$ (a function that is differentiable for all degrees of differentiation) and $f : \mathbb{R} \to \mathbb{R}$, the *Taylor series* of f at x_0 is defined as:

$$T_{\infty}(x) = \sum_{k=0}^{\infty} \frac{f^{(k)}(x_0)}{k!}(x - x_0)^k \tag{39}$$

Figure 8: Visualization of the Taylor series of $cos(x)$. As we add more terms to our series, our Taylor series models $cos(x)$ more closely around x_0.



For $x_0 = 0$, we obtain a special instance of the Taylor series known as the *Maclaurin series*.

In general, a Taylor polynomial of degree n is an approximation of a function (that does not necessarily need to be a polynomial), and is similar to f in a "neighborhood" around x_0.

The concept of a Taylor series/polynomial can also be extended to multivariable functions. Consider the function $f : \mathbb{R}^D \to \mathbb{R}$, with $x \mapsto f(x)$ and $x \in \mathbb{R}^D$ that is smooth at x_0. When we define the difference vector $\delta = x - x_0$, the multivariable Taylor series of f at x_0 is defined as:

$$f(x) = \sum_{k=0}^{\infty} \frac{D_x^k f(x_0)}{k!}\delta^k \tag{40}$$

where $D_x^k f(x_0)$ is the kth (total) derivative of f with respect to x, evaluated at x_0.

Likewise, the Taylor polynomial of degree n of f at x_0 is defined as:

$$T_n(x) = \sum_{k=0}^{n} \frac{D_x^k f(x_0)}{k!} \delta^k \tag{41}$$

3 Applications to Machine Learning

3.1 Partial Derivatives and Gradients

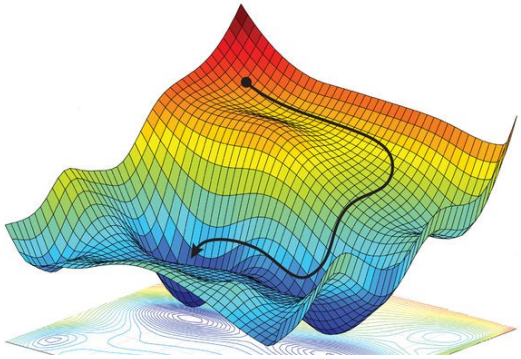
3.1.1 Optimization

Training a machine learning algorithm/model boils down to finding a "good" set of parameters, with the notion of "good" being determined by some objective function or the probabilistic model. Given an objective function that determines our model's performance, we can use optimization algorithms to find the best values.

By convention, most objective functions in machine learning are intended to be minimized. As such, the best possible value is the global minimum value of our objective function.

From Chapter 2 (Partial Derivatives and Gradients) of Section 2, we know that the gradient of a multivariable function always "points" in the direction of steepest ascent. Therefore, what we should do is use the *negative* of the gradient to take "steps" in the direction of steepest *descent*, while also being careful to avoid "traps" such as local minima (where the function appears to maximally minimized according to the surrounding values, but is not actually the global minimum). Reaching the global minimum allows us to select the best parameters for our model, and therefore achieve the highest possible performance on a given task by iteratively updating our parameters.

Figure 9: Visualization of gradient descent on a multivariable objective function. (Amini, Rus)



This method of optimization is called *gradient descent*, and is covered in our third tutorial in our set of machine learning tutorials.

Many modifications (e.g. momentum, choosing the right set of hyperparameters, etc) to gradient descent are involved when training a machine learning model, but the above captures the core essence of gradient descent. In addition, specialized techniques (e.g. vectorization) are utilized by computers to train models with large ($> 10^6$) numbers of parameters.

However, we have not yet addressed the issue of actually computing the gradient of the objective function. In the case of artificial neural networks, a algorithm called *backpropagation* is used to do such by using the chain rule to calculate the gradient of the loss function (the objective function) with respect to each of the neural network's weight (parameters).

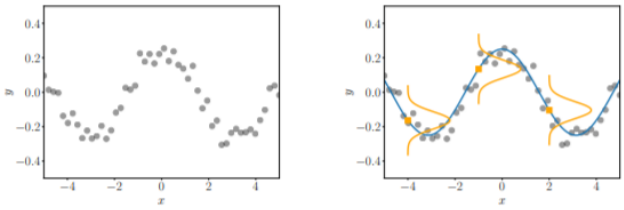
As versatile as gradient descent may seem, there are other optimization problems (e.g. convex optimization, quadratic optimization, constrained optimization, etc) which more specialized optimization algorithms are suited to.

3.1.2 Regression

In regression, we aim to find a continuous function f that maps inputs x to corresponding function values $f(x)$. We assume that we are given a set of training inputs x_n and corresponding observations (training outputs) $y_n = f(x_n) + \epsilon$, where ϵ is an independent and identically distributed random variable that describes measurement/observation noise.

We want to find a function that not only models the training data, but generalizes well to predicting function values that are not part of the training data. A typical example of a regression problem is shown below:

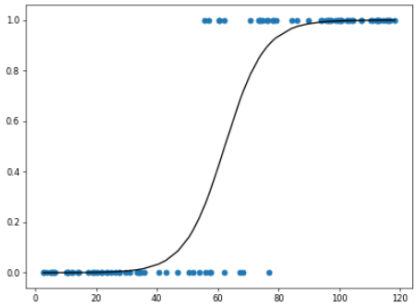
Figure 10: Visualization of a typical regression problem and one possible solution. (Deisenroth, Faisal, Ong; 2020)



Given this training data, we want to infer the function that generated this data. Finding this function requires solving multiple subproblems, including choosing a model & parameterization (what model best suits this data? what parameters should we use?), finding good parameters (optimization), overfitting (does our model fit the data *too* closely?), and uncertainty modelling (how well can our model generalize to data that it's never seen before?).

For instance, we can use *logistic regression* to model the probability of events with only two possible outcomes (alive/dead, pass/fail, etc). Given training data and an objective function (logistic loss) to minimize, we can use gradient descent to iteratively update the parameters of our model until we reach the global minimum of our objective function and therefore achieve the best possible set of parameters.

Figure 11: Visualization of the logistic regression algorithm. (OpenClassroom)

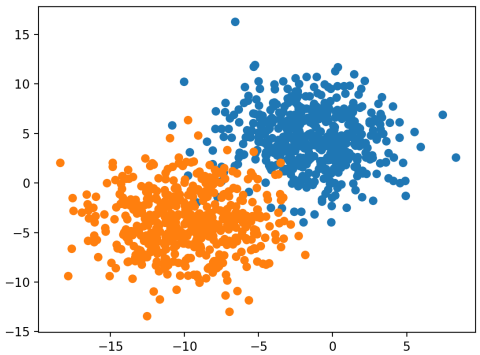


Regression is a fundamental problem in machine learning, and regression problems appear in a diverse range of research areas and applications, including time-series analysis, control systems, and deep learning.

3.1.3 Classification

In many situations, we want our machine learning algorithm to predict one of a number of outcomes. For instance, sorting received emails into the categories of spam and non-spam. There are usually a small number of outcomes and no additional structure on these outcomes. This a subset of this task in which we want to divide a dataset into two classes is called *binary classification*.

Figure 12: Visualization of a typical binary classification problem. (Brownlee, 2019)

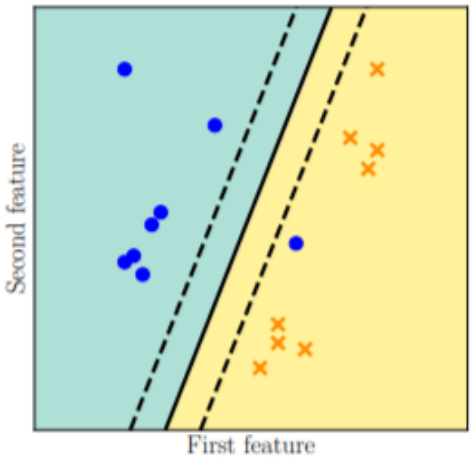


An approach known the *support vector machine* (SVM) solves the binary classification problem, though it is also suited for regression. Given a training set composed of input-output pairs, we want to find parameters for our SVM such that

our SVM will have the smallest classification error.

SVMs are based on the idea of finding a hyperplane that best divides a dataset into two classes. Simply put, we can define an optimization function to maximize a certain aspect(s) of the hyperplane. By using gradient ascent, we can maximize that aspect of our hyperplane over all of our training examples.

Figure 13: Visualization of a support vector machine's hyperplane. (Deisenroth, Faisal, Ong; 2020)



3.2 Integrals

3.2.1 Probability

The age of a person, the height of a person, their lifetime—each of these quantities have values that will range over an interval of integers, and because of this, we call these quantities *continuous random variables*.

Every continuous random variable has a *probability density function* $p(x)$, which satisfy the following conditions:

$$p(x) \geq 0 \text{ for all } x \tag{42}$$

$$\int_{-\infty}^{\infty} p(x)dx = 1 \tag{43}$$

Probability density functions can be used to determine the probability that a continuous random variable X lies between two values (say, a and b), which is denoted by $P(a \leq X \leq b)$ and is given by

$$P(a \leq X \leq b) = \int_{a}^{b} p(x)dx \tag{44}$$

3.2.2 Expected Value

The *expected value* of the random continuous variable X is computed using the formula

$$\mu = \int_{-\infty}^{\infty} x\, p(x) \tag{45}$$

and is a single number representing what value of X we can expect to obtain on average from the random variable X. This number is also called the *average* or *mean* of the random variable X.

3.2.3 Variance

The *variance* of the random variable X is defined as follows:

$$\sigma^2 = \int_{-\infty}^{\infty} (x-\mu)^2\, p(x) \tag{46}$$

This variance formula lets us compute the expectation of the squared distance of the random variable X from its expected value. The variance σ^2 gives us an indication of how clustered or spread the values of X are. A small variance indicates the outcomes of X are tightly clustered near the expected value μ, while a large variance indicates that the outcomes of X are widely spread. The square root of the variance is termed the *standard deviation* and is usually denoted σ.

The expected value and the variance are two central concepts in statistics because they allow us to characterize any random variable. The expected value is a measure of the *central tendency* of the random variable, while the variance measure its *dispersion*.

4 References

Deisenroth, M., Faisal, A., & Ong, C. (2020). Vector Calculus. In Mathematics for Machine Learning (pp. 120-151). Cambridge: Cambridge University Press. doi:10.1017/9781108679930.007

Goodfellow, I., Bengio, Y., Courville, A. (2016). Deep Learning. MIT Press

Calculus. (n.d.). Retrieved from https://ml-cheatsheet.readthedocs.io/en/latest/calculus.html