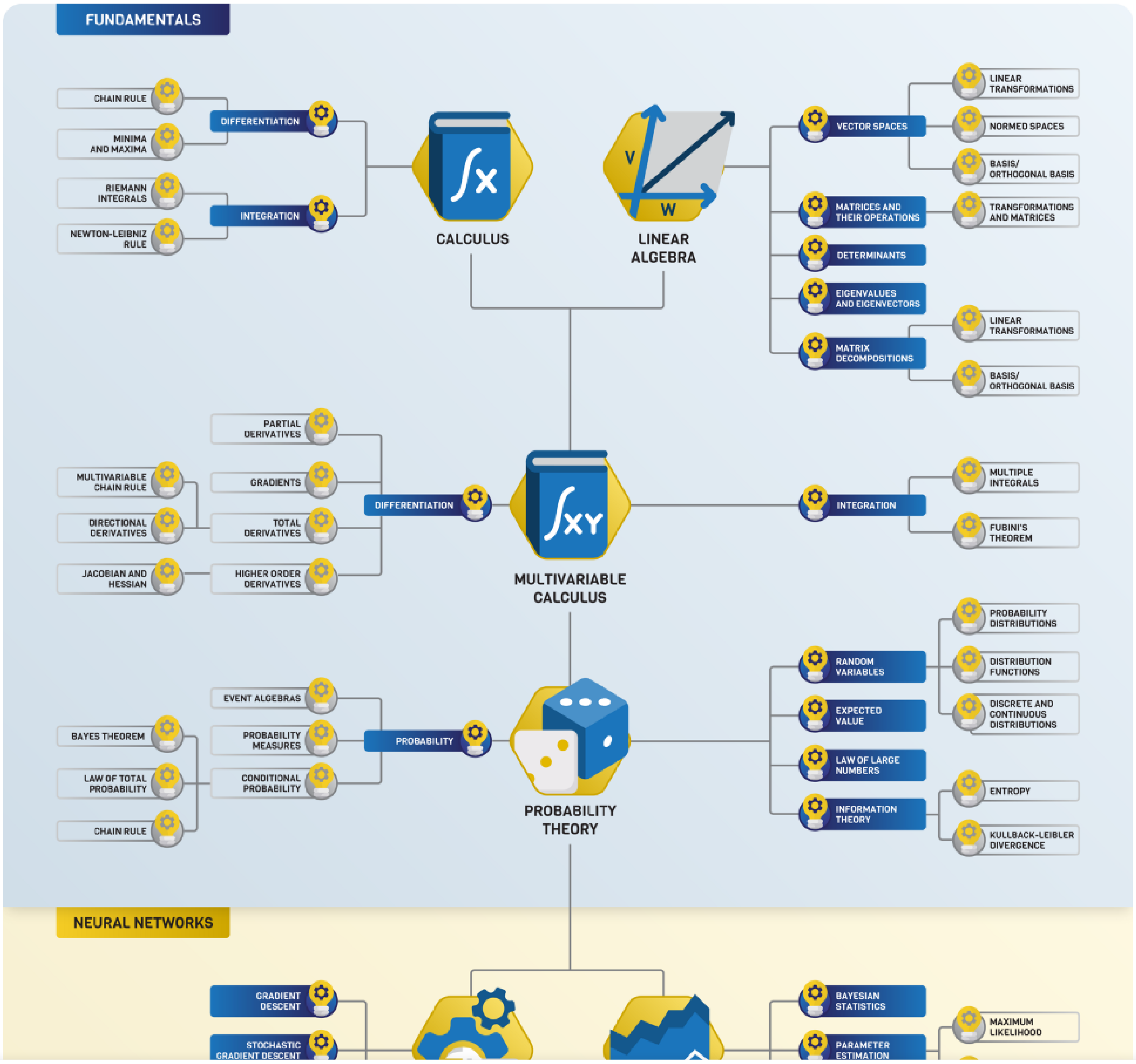
Open in app Get started

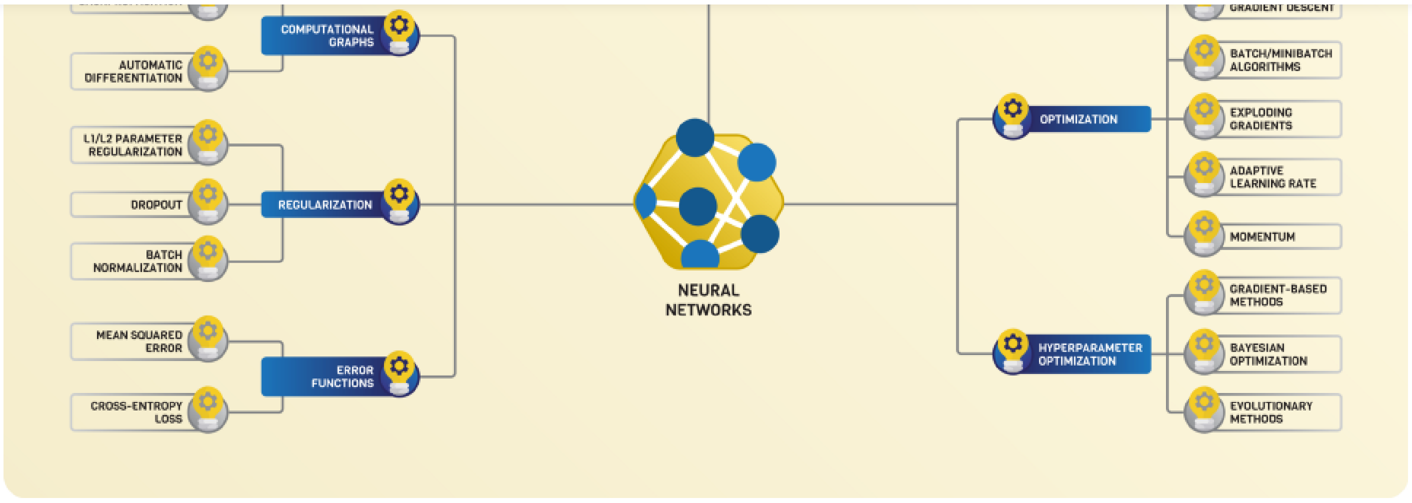
Published in Towards Data Science

Tivadar Danka Follow
Oct 23, 2020 · 19 min read ★ · Listen

Save



Open in app Get started



Images by Author unless otherwise indicated

The Roadmap of Mathematics for Machine Learning

Understanding the inner workings of neural networks from the ground-up

Knowing the mathematics behind machine learning algorithms is a superpower. If you have ever built a model for a real-life problem, you probably experienced that being familiar with the details can go a long way if you want to move beyond baseline performance. This is especially true when you want to push the boundaries of state of the art.

However, most of this knowledge is hidden behind layers of advanced mathematics. Understanding methods like stochastic gradient descent might seem difficult since it is built on top of multivariable calculus and probability theory.

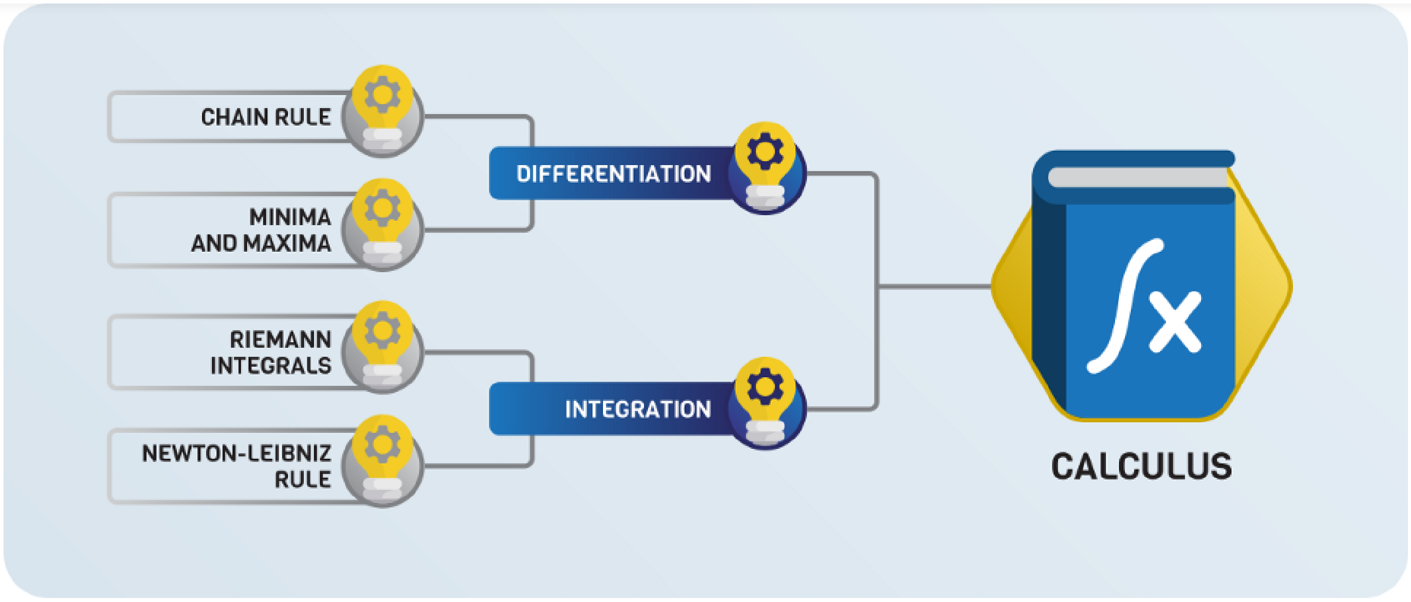
With proper foundations, though, most ideas can be seen as quite natural. If you are a beginner and don’t necessarily have formal education in higher mathematics, creating a curriculum for yourself is hard. In this post, my goal is to present a roadmap, taking you from absolute zero to a deep understanding of how neural networks work.

To keep things simple, the aim is not to cover everything. Instead, we will focus on getting our directions. This way, you will be able to study other topics without difficulties, if need be.

Instead of reading through in one sitting, I recommend using this article as a reference point through your studies. Go deep into a concept that is introduced, then check the roadmap and move on. I firmly believe that this is the best way to study: I will show you the road, but you must walk it.

The fundamentals

Most of machine learning is built upon three pillars: **linear algebra**, **calculus**, and **probability theory**. Since the last one builds on the first two, we should start with them. Calculus and linear algebra can be studied independently, as is usually the case in a standard curriculum.



Calculus

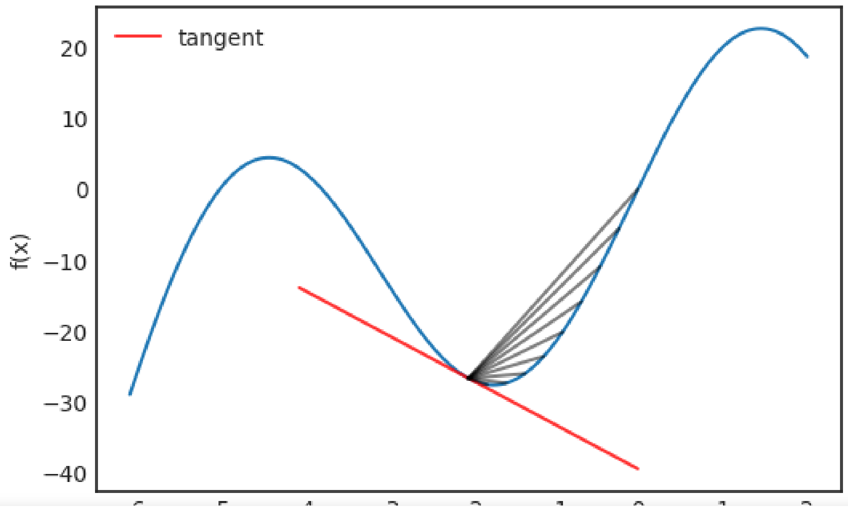
Calculus is the study of *differentiation* and *integration* of functions. Essentially, a neural network is a differentiable function, so calculus will be a fundamental tool to train neural networks, as we will see.

To familiarize yourself with the concepts, you should make things simple and study functions of a single variable for the first time. By definition, the derivative of a function is defined by

$$f'(x) = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

where the ratio for a given *h* is the slope of the line between the points *(x, f(x))* and *(x+h, f(x+h))*.

In the limit, this is essentially the slope of the tangent line at the point *x*. The figure below illustrates the concept.



Differentiation can be used to optimize functions: the derivative is zero at local maxima or minima. (However, this is not true in the other direction; see *f(x)* = x^3 at *0*.) Points where the derivative is zero, are called *critical points*. Whether a critical point is minima or maxima can be decided by looking at the *second derivative*:

$$f''(x_0) \begin{cases} > 0: & x_0 \text{ is a local minima,} \\ < 0: & x_0 \text{ is a local maxima,} \\ = 0: & \text{undetermined.} \end{cases}$$

There are several essential rules regarding differentiation, but probably the most important is the so-called *chain rule*:

$$\big(f(g(x))\big)' = f'(g(x))g'(x),$$

that tells us how to calculate the derivative of composed functions.

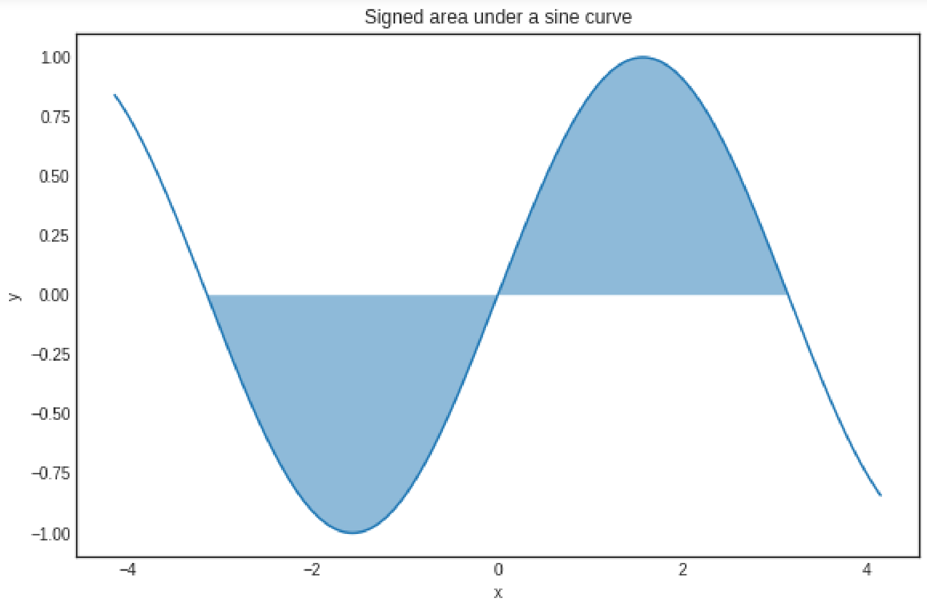
Integration is often called the inverse of differentiation. This is true because

$$F(x) = f(x_0) + \int_{x_0}^{x} f(x)dx$$
$$F'(x) = f(x)$$

which holds for any integrable function *f(x)*. The integral of a function can also be thought of as the signed area under the curve. For instance,

$$\int_{-\pi}^{\pi} \sin(x)dx = 0,$$

because when the function is negative, the area there also has a negative sign.



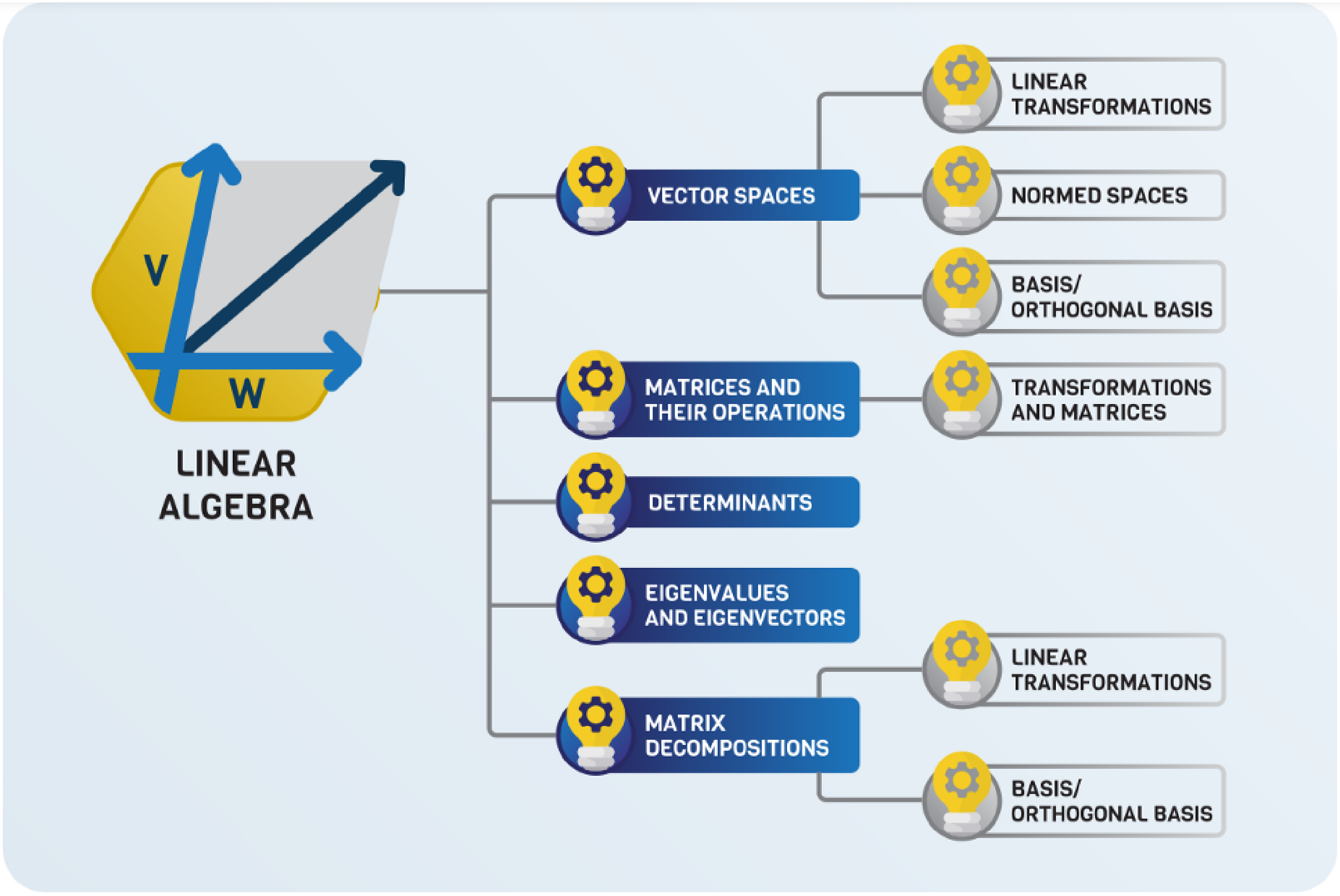
The signed area under a sine curve between -π and π

Integration itself plays a role in understanding the concept of expected value. For instance, quantities like entropy and Kullback-Leibler divergence are defined in terms of integrals.

Further study

I would recommend the Single Variable Calculus course from MIT. (In general, online courses from MIT are always excellent learning resources.) If you are more of a book person, there are many textbooks available. The Calculus book by Gilbert Strang accompanying the previously mentioned course is again a great resource, completely free of charge.

Open in app Get started



Linear algebra

As I mentioned, neural networks are essentially functions, which are trained using the tools of calculus. However, they are *described* with linear algebraic concepts like matrix multiplication.

Linear algebra is a vast subject with many essential aspects of machine learning, so this will be a significant segment.

Vector spaces

To have a good understanding of linear algebra, I would suggest starting with vector spaces. It is better if we talk about a special case first. You can think of each point in the plane as a tuple

$$x = (x_1, x_2) \in \mathbb{R}^2.$$

These are essentially vectors pointing from zero to *(x1, x2)*. You can add these vectors together and multiply them with scalars:

$$(x_1, x_2) + (y_1, y_2) = (x_1 + y_1, x_2 + y_2)$$

properties hold:

$$\begin{aligned}
&(1) \quad x+y=y+x \quad \text{for all } x,y\in V,\\
&(2) \quad x+(y+z)=(x+y)+z \quad \text{for all } x,y,z\in V,\\
&(3) \quad \text{there is an element } 0\in V \text{ such that for all } x\in V, \text{ we have } v+0=v,\\
&(4) \quad \text{for every } x\in V, \text{ there is an } y\in V \text{ such that } x+y=0,\\
&(5) \quad a(bx)=(ab)x \text{ for all } a,b\in\mathbb{R}, x\in V,\\
&(6) \quad 1x=x \text{ for all } x\in V,\\
&(7) \quad (a+b)x=ax+bx \text{ for all } a,b\in\mathbb{R}, x\in V,\\
&(8) \quad a(x+y)=ax+ay \text{ for all } a\in\mathbb{R}, x,y\in V.
\end{aligned}$$

Don't panic! I know that this looks very scary (at least it looked that to me when I was a freshman student with a math major), but it really isn't. These just guarantee that vectors can be added together and scaled just as you would expect. When thinking about vector spaces, it helps if you mentally model them as

$$\mathbb{R}^2.$$

Normed spaces

If you feel like you have a good understanding of vector spaces, the next step would be to understand how to measure a vector's magnitude. By default, a vector space in itself gives no tools for this. How would you do this on the plane? You have probably learned that there, we have

$$\|x\| = \sqrt{(x_1)^2+(x_2)^2}.$$

This is a special case of a *norm*. In general, a vector space *V* is normed if there is a function

$$\|\cdot\| : V \to [0,\infty)$$

called norm such that

$$\begin{aligned}
&(1) \quad \|x\|\geq 0, \quad x\in V,\\
&(2) \quad \|x\|=0 \quad \text{if and only if} \quad x=0,\\
&(3) \quad \|ax\|=|a|\|x\| \quad a\in\mathbb{R}, x\in V,\\
&(4) \quad \|x+y\|\leq\|x\|+\|y\|, \quad x,y\in V.
\end{aligned}$$

$$\|x\|_p = \Big(\sum_{i=1}^{n} (x_i)^p \Big)^{1/p}, \quad p \in [1, \infty), x \in \mathbb{R}^n$$

(with *p = 2* we obtain the special case mentioned above) and the *supremum norm*

$$\|x\|_\infty = \sup\{x_1, \dots, x_n\}, \quad x \in \mathbb{R}^n.$$

Sometimes, like for *p = 2*, the norm comes from a so-called *inner product*, which is a bilinear function

$$\langle \cdot, \cdot \rangle : V \times V \to [0, \infty),$$

such that

$$\begin{aligned}
&(1) \quad \langle x, y \rangle = \langle y, x \rangle, \\
&(2) \quad \langle ax + y, z \rangle = a\langle x, z \rangle + \langle y, z \rangle, \\
&(3) \quad \langle x, x \rangle \geq 0, \\
&(4) \quad \langle x, x \rangle = 0 \iff x = 0.
\end{aligned}$$

A vector space with an inner product is called *inner product space*. An example is the classical Euclidean product

$$\langle x, y \rangle = x_1 y_1 + x_2 y_2.$$

Every inner product can be turned into a norm by

$$\|x\| = \sqrt{\langle x, x \rangle}.$$

When the inner product for two vectors is zero, we say that the vectors are *orthogonal* to each other. (Try to come up with some concrete examples on the plane to understand the concept deeper.)

Basis and orthogonal/orthonormal basis

Although vector spaces are infinite (in our case), you can find a finite set of vectors that can be used to express *all* vectors in the space. For example, on the plane, we have

$$x = (x_1, x_2) = x_1 e_1 + x_2 e_2,$$

where

In general, *basis* is a minimal set of vectors

$$b_1, b_2, \ldots, b_n \in V$$

such that their linear combinations span the vector space:

$$V = \left\{ \sum_{i=1}^{n} \alpha_i b_i : a_i \in \mathbb{R} \right\}.$$

A basis always exists for any vector space. (It may not be a finite set, but that shouldn't concern us now.) Without a doubt, a basis simplifies things greatly when talking about linear spaces.

When the vectors in a basis are orthogonal to each other, we call it an *orthogonal basis*. If each basis vector's norm is 1 for an orthogonal basis, we say it is *orthonormal*.

Linear transformations

One of the key objects related to vector spaces are *linear transformations*. If you have seen a neural network before, you know that one of the fundamental building blocks are layers of the form

$$f(x) = \sigma(Ax + b),$$

where *A* is a matrix, *b* and *x* are vectors, and *σ* is the sigmoid function. (Or any activation function, really.) Well, the part *Ax* is a linear transformation. In general, the function

$$L : V \mapsto W$$

is a linear transformation between vector spaces *V* and *W* if

$$L(ax + y) = aL(x) + L(y)$$

holds for all *x, y* in *V*, and all *a* real number.

To give a concrete example, rotations around the origin in the plane are linear transformations.

Undoubtedly, the most crucial fact about linear transformations is that they can be represented with matrices, as you'll see next in your studies.

Matrices and their operations

If linear transformations are clear, you can turn to the study of matrices. (Linear algebra courses often start with matrices, but I would recommend it this way for reasons to be explained later.)

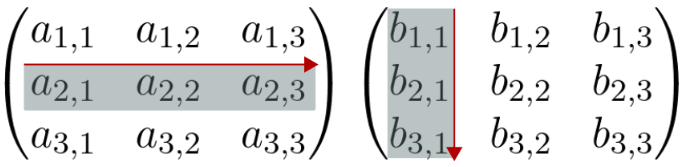
$$A = (a_{i,j})_{i,j=1}^{n,m} \in \mathbb{R}^{n \times m},$$
$$B = (b_{i,j})_{i,j=1}^{m,l} \in \mathbb{R}^{m \times l},$$

then their product can be obtained by

$$AB = \Big(\sum_{k=1}^{m} a_{i,k} b_{k,j} \Big)_{i,j=1}^{n,l}.$$

This might seem difficult to comprehend, but it is actually quite straightforward. Take a look at the figure below, demonstrating how to calculate the element in the 2nd row, 1 st column of the product.

$$a_{2,1}b_{1,1} + a_{2,2}b_{2,1} + a_{2,3}b_{3,1}$$



The reason why matrix multiplication is defined the way it is because matrices represent linear transformations between vector spaces. Matrix multiplication is the composition of linear transformations.

If you would like to read more about this, there is a great article right here at Towards Data Science, explaining things in detail.

Why is Linear Algebra Taught So Badly?

Linear algebra is one of the cornerstones of machine learning. It's more intuitive than you might think

towardsdatascience.com

Determinants

In my opinion, determinants are hands down one of the most challenging concepts to grasp in linear algebra. Depending on your learning resource, it is usually defined by either a recursive definition or a sum that iterates through all permutations. None of them are tractable without significant experience in mathematics.

To understand this concept, watch this video below. Trust me, it is magic.

To summarize, the determinant of a matrix describes how the volume of an object scales under the corresponding linear transformation. If the transformation changes orientations, the sign of the determinant is negative.

You will eventually need to understand how to calculate the determinant, but I wouldn't worry about it now.

Eigenvalues, eigenvectors, and matrix decompositions

A standard first linear algebra course usually ends with eigenvalues/eigenvectors and some special matrix decompositions like the Singular Value Decomposition.

Let's suppose that we have a matrix *A*. The number *λ* is an eigenvalue of *A* if there is a vector *x* (called eigenvector) such that

$$Ax = \lambda x$$

holds. In other words, the linear transformation represented by *A* is a scaling by *λ* for the vector *x*. This concept plays an essential role in linear algebra. (And practically in every field which uses linear algebra extensively.)

At this point, you are ready to familiarize yourself with a few matrix decompositions. If you think about it for a second, what type of matrices are the best from a computational perspective? Diagonal matrices! If a linear transformation has a diagonal matrix, it is trivial to compute its value on an arbitrary vector.

$$\begin{pmatrix} \lambda_1 & 0 & 0 & \dots & 0 \\ 0 & \lambda_2 & 0 & \dots & 0 \\ 0 & 0 & \lambda_3 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & 0 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} \lambda_1 x_1 \\ \lambda_2 x_2 \\ \lambda_3 x_3 \\ \vdots \\ \lambda_n x_n \end{pmatrix}$$

matrices *U*, *V,* and a diagonal matrix *Σ* such that

$$A = U\Sigma V$$

holds. (*U* and *V* are so-called *unitary matrices*, which I don't define here, suffice to know that it is a special family of matrices.)

SVD is also used to perform Principal Component Analysis, one of the simplest and most well-known dimensionality reduction methods.

Further study

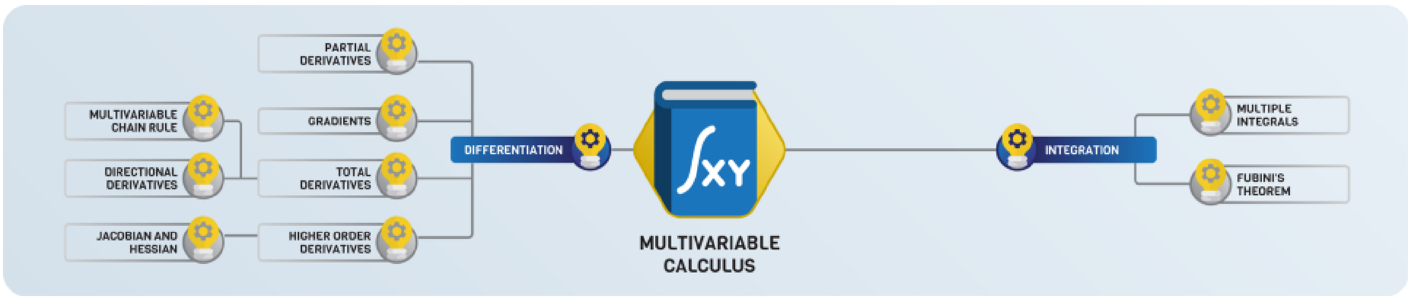
Linear algebra can be taught in many ways. The path I outlined here was inspired by the textbook Linear Algebra Done Right by Sheldon Axler. For an online lecture, I would recommend the Linear Algebra course from MIT OpenCourseWare, an excellent resource.

If a course might be too much, there are great articles out there, for instance, the following.

Mathematics for AI: Linear Algebra and How to Understand It Better

How it works, where it is used and how to learn it faster.

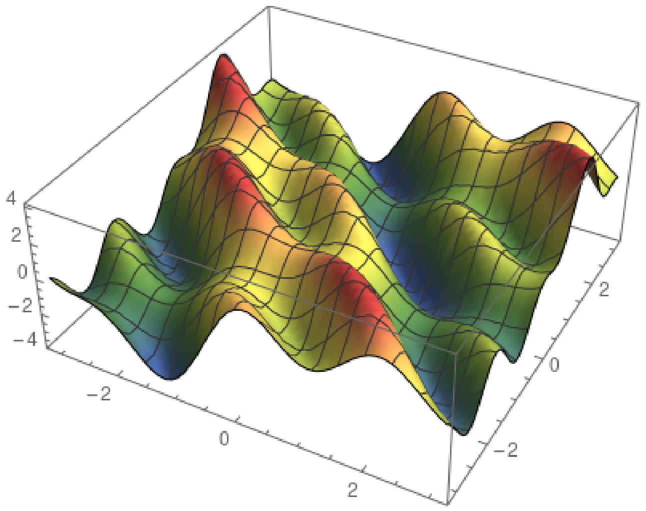
towardsdatascience.com



Multivariable calculus

This is the part where linear algebra and calculus come together, laying the foundations for the primary tool to train neural networks: gradient descent. Mathematically speaking, a neural network is simply a function of multiple variables. (Although, the number of variables can be in the millions.)

Similar to univariable calculus, the two main topics here are differentiation and integration. Let's suppose that we have a function mapping vectors to real numbers. In two dimensions (that is, for $n = 2$), you can imagine its plot as a surface. (Since humans don't see higher than three dimensions, it is hard to visualize functions with more than two real variables.)



Graph of a function of two variables.

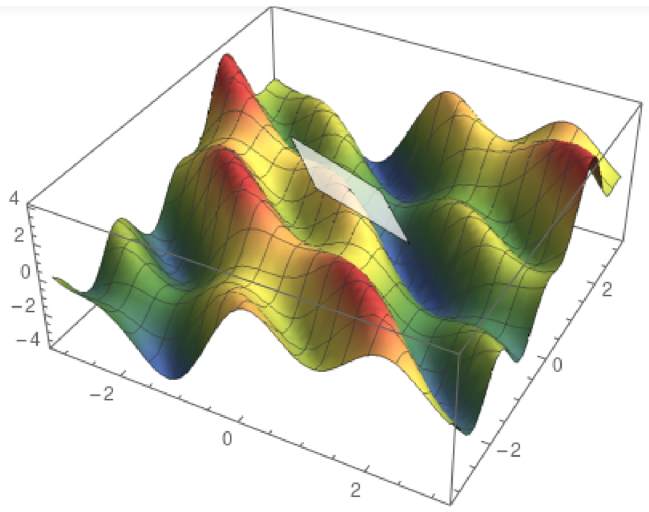
Differentiation in multiple variables

In a single variable, the derivative was the slope of the tangent line. How would you define tangents here? A point on the surface has several tangents, not just one. However, there are two special tangents: one is parallel to the *x*-*z* plane, while the other is to the *y*-*z* plane. Their slope is determined by the *partial derivatives*, defined by

$$\begin{aligned}\frac{\partial f}{\partial x} &:= \lim_{h \to 0} \frac{f(x+h, y) - f(x, y)}{h} \\ \frac{\partial f}{\partial y} &:= \lim_{h \to 0} \frac{f(x, y+h) - f(x, y)}{h}.\end{aligned}$$

That is, you take the derivative of the functions obtained by fixing all but one variable. (The formal definition is identical for ≥ 3 variables, just more complicated notation.)

Tangents in these special directions span the *tangent plane*.



Tangent plane.

The gradient

There is another special direction: the gradient, which is the vector defined by

$$\nabla f(x_1, x_2, \dots, x_n) = \left(\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \dots, \frac{\partial f}{\partial x_n} \right).$$

The gradient always points to the direction of the largest increase! So, if you would take a tiny step in this direction, your elevation would be the maximal among all the other directions you could have chosen. This is the basic idea of *gradient descent*, which is an algorithm to maximize functions. Its steps are the following.

1. Calculate the gradient in the point *x₀*, where you currently are.
2. Take a small step in the direction of the gradient to arrive at the point *x₁*. (The step size is called the *learning rate*.)
3. Go back to Step 1. and repeat the process until convergence.

Of course, there are several flaws in this basic algorithm, which was improved several times over the years. Modern gradient descent based optimizers employ many tricks like adaptive step size, momentum, and other methods, which we are not going to detail here.

Calculating the gradient in practice is difficult. Functions are often described by the composition of other functions, for instance, the familiar linear layer

$$f(x) = \sigma(Ax + b),$$

where *A* is a matrix, *b* and *x* are vectors, and *σ* is the sigmoid function. (Of course, there can be are other can always be written in terms of vector-scalar functions like

$$g(x) = \begin{pmatrix} g_1(x), \\ g_2(x), \\ \vdots, \\ g_m(x) \end{pmatrix}, \quad g_i : \mathbb{R}^n \to \mathbb{R}.$$

The gradient of *g* is defined by the matrix whose *k*-th row is the *k*-th component's gradient. That is,

$$\nabla g(x) = \left(\frac{\partial}{\partial x_i} g_j(x) \right)_{i,j=1}^{n,m}.$$

This matrix is called the *total derivative of g*.

In our example

$$f(x) = \sigma(Ax + b),$$

things become a bit more complicated because it is the composition of two functions:

$$l : \mathbb{R}^n \mapsto \mathbb{R}^m, \quad x \mapsto Ax + b$$

and

$$\sigma : \mathbb{R}^m \mapsto \mathbb{R}^m,$$

defined by applying the univariate sigmoid componentwise. The function *l* can be decomposed further to *m* functions mapping from the n-dimensional vector space to the space of real numbers:

$$l(x) = Ax + b = \begin{pmatrix} l_1(x) \\ l_2(x) \\ \vdots \\ l_m(x) \end{pmatrix},$$

where

$$l_i(x) = \sum^{n} a_{ij} x_j + b_i$$

$$\nabla f(x) = \nabla \sigma(l(x)) \cdot \nabla l(x).$$

This is the *chain rule for multivariate functions* in its full generality. Without it, there would be no easy way to calculate the gradient of a neural network, which is ultimately a composition of many functions.

Higher-order derivatives

Similarly to the univariate case, the gradient and the derivatives play a role in determining whether a given point in space is local minima or maxima. (Or neither.) To provide a concrete example, training a neural network is equivalent to minimizing the loss function on the parameters' training data. It is all about finding the optimal parameter configuration *w* for which the minimum is attained:

$$\min_{w} l(N(x_{\text{train}}, w)),$$

where

$$N: \mathbb{R}^n \to \mathbb{R}^m, \quad l: \mathbb{R}^m \to \mathbb{R}$$

are the neural network and the loss function, respectively.

For a general differentiable vector-scalar function of say *n* variables, there are n^2 second derivatives, forming the *Hessian matrix*

$$Hf(x) = \begin{pmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2^2} & \cdots & \frac{\partial^2 f}{\partial x_2 \partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1} & \frac{\partial^2 f}{\partial x_n \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_n^2} \end{pmatrix}.$$

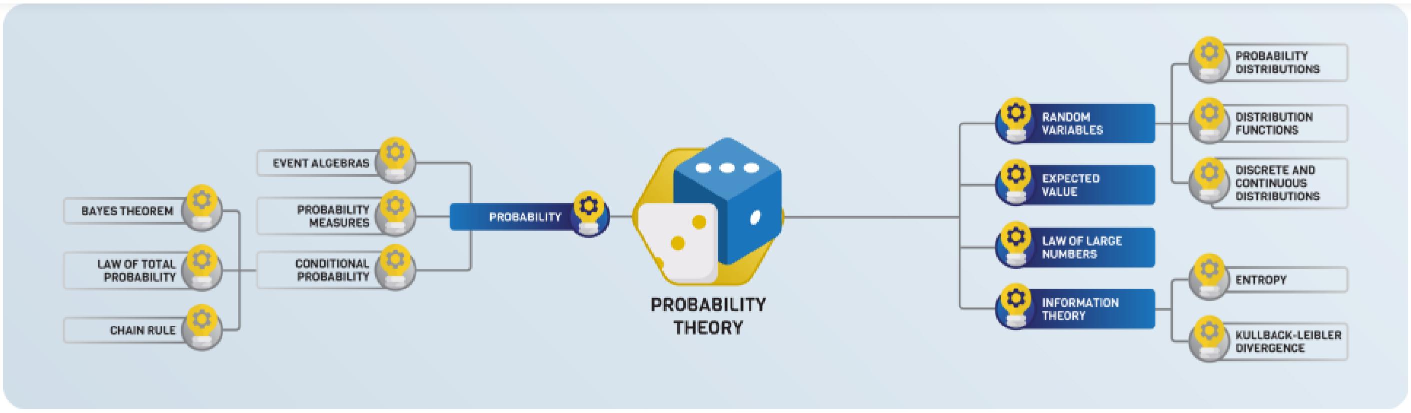
In multiple variables, the determinant of the Hessian takes the role of the second derivative. Similarly, it can be used to tell whether a critical point (that is, where all of the derivatives are zero) is minima, maxima, or just a saddle point.

Further study

There are a lot of awesome online courses on multivariable calculus. I have two specific recommendations:

- Khan Academy on multivariable calculus,
- MIT multivariable calculus.

Now we are ready to take on the final subject: probability theory!



Probability theory

Probability theory is the mathematically rigorous study of chance, fundamental to all fields of science.

Setting exact definitions aside, let's ponder a bit about what *probability* represents. Let's say I toss a coin, with a 50% chance (or *0.5 probability*) of it being heads. After repeating the experiment 10 times, how many heads did I get?

If you have answered 5, you are wrong. Heads being 0.5 probability doesn't guarantee that every second throw is heads. Instead, what it means that if you repeat the experiment *n* times where *n* is a *really large number*, the number of heads will be very close to *n/2*.

To get a good grip on probability, I would recommend my post below, which I wrote a while ago, to offer a concise yet mathematically correct presentation of the concept.

The mathematical foundations of probability

A measure-theoretic introduction

towardsdatascience.com

Besides the basics, there are some advanced things you need to understand, first and foremost, *expected value* and *entropy*.

Expected value

Supposed that you play a game with your friend. You toss a classical six-sided dice, and if the outcome is 1 or 2, you win 300 bucks. Otherwise, you lose 200. What are your average earnings per round if you play this game long enough? Should you be even playing this game?

Well, you win 100 bucks with probability 1/3, and you lose 200 with probability 2/3. That is if X is the random variable encoding the result of the dice throw, then

$$
\begin{aligned}
E[X] &= 300P\big(X \in \{1, 2\}\big) - 200P\big(X \in \{3, 4, 5, 6\}\big) \\
&= 300\frac{1}{3} - 200\frac{2}{3} \\
&= -\frac{100}{3}.
\end{aligned}
$$

This is the expected value, that is, the average amount of money you will receive per round in the long run. Since this is negative, you will lose money, so you should never play this game.

Generally speaking, the expected value is defined by

$$E[X] = \sum_{i=1}^{\infty} x_i P(X = x_i), \quad X \in \{x_1, x_2, \dots\}$$

for discrete random variables and

$$E[X] = \int_{-\infty}^{\infty} x f_X(x) dx$$

for real-valued continuous random variables.

In machine learning, *loss functions for training neural networks are expected values* in one way or another.

Law of large numbers

People often falsely attribute certain phenomena to the law of large numbers. For instance, gamblers who are on a losing streak believe that they should soon win because of the *law of large numbers*. This is totally wrong. Let's see what this is really!

Suppose that

$$X, X_1, X_2, \dots$$

are random variables representing the independent repetitions of the same experiment. (Say, rolling a dice or tossing a coin.)

Essentially, the law of large numbers states that

$$\lim_{n \to \infty} \frac{1}{n} \sum_{i=1}^{n} X_i = E[X].$$

An interpretation is that if a random event is repeated enough times, individual results might not matter. So, if you are playing in a casino with a game that has negative expected value (as they all do), it doesn't matter that you win occasionally. The law of large numbers implies that you *will lose money*.

To get a little bit ahead, LLN is going to be essential for *stochastic gradient descent*.

Information theory

Let's play a game. I have thought of a number between 1 and 1024, and you have to guess it. You can ask questions, but your goal is to use as few questions as possible. How much do you need?

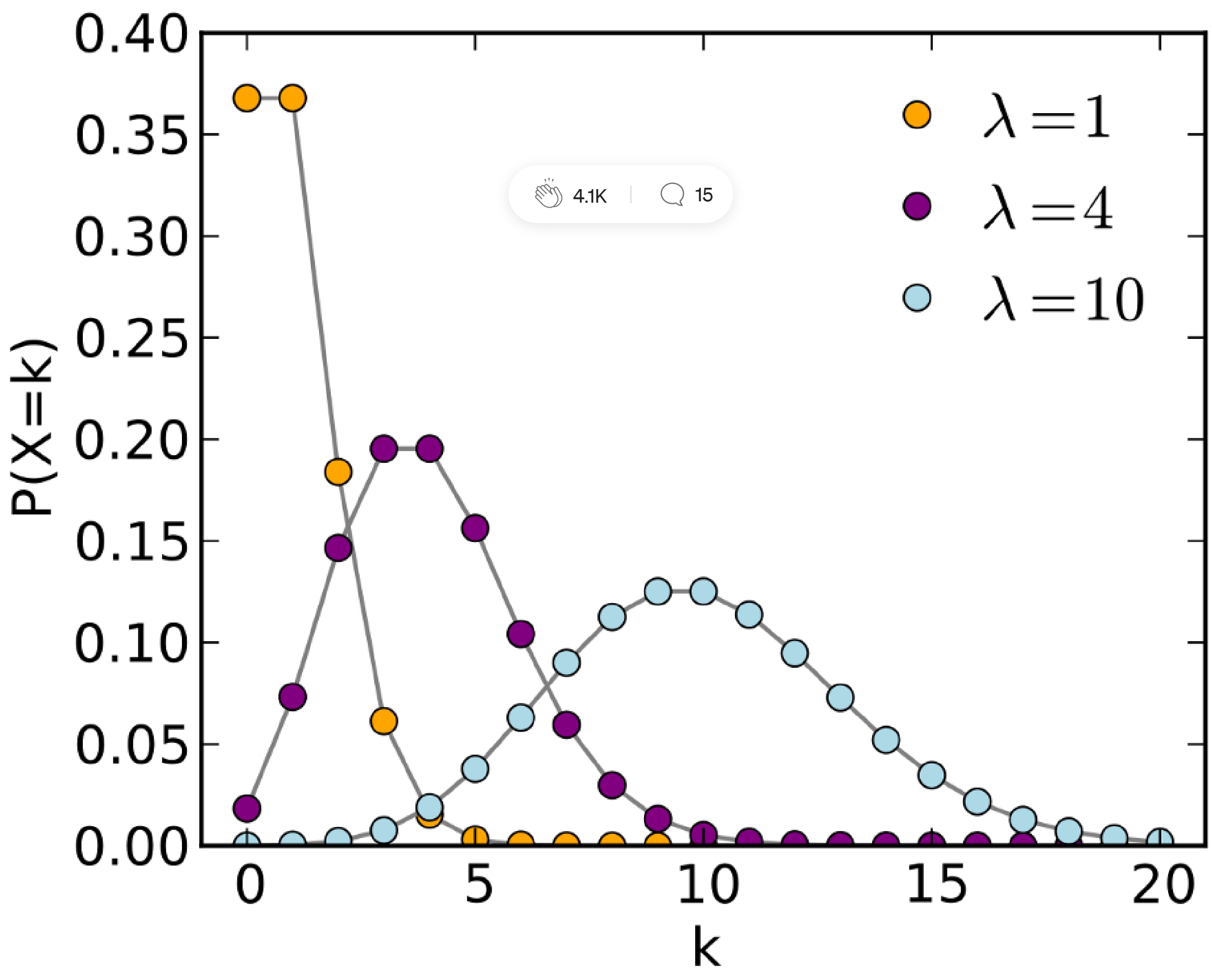
If you play it smart, you will perform a binary search with your questions. First you may ask: *is the number between 1 and 512?* With this, you have cut the search space in half. Using this strategy, you can figure out the answer in

$$\log_2 1024 = 10$$

questions.

But what if I didn't use the uniform distribution when picking the number? For instance, I could have used a Poisson distribution.

Open in app Get started



Probability mass functions for Poisson distributions. Source: Wikipedia

Here, you would probably need fewer questions because you know that the distribution tends to concentrate around specific points. (Which depends on the parameter.)

In the extreme case, when the distribution is concentrated on a single number, you need *zero* questions to guess it correctly. Generally, the number of questions depends on the information carried by the distribution. The uniform distribution contains the least amount of information, while singular ones are pure information.

Entropy is a way to quantify this. It is defined by

$$H[X] = -\sum_{i=1}^{\infty} P(x_i) \log P(x_i)$$

for discrete random variables and

for continuous, real-valued ones. (The base of the logarithm is usually 2, *e*, or 10, but it doesn't really matter.)

If you have ever worked with classification models before, you probably encountered the *cross-entropy* loss, defined by

$$L[X] = -\sum_{i=1}^{\infty} P(x_i) \log \hat{P}(x_i),$$

where P is the ground truth (a distribution concentrated to a single class), while the hatted version represents the class predictions. This measures how much "information" the predictions have compared to the ground truth. When the predictions match, the cross-entropy loss is zero.

Another frequently used quantity is the *Kullback-Leibler divergence*, defined by

$$D_{KL}(P||Q) = -\sum_{i=1}^{\infty} P(x_i) \log \frac{Q(x_i)}{P(x_i)},$$

where *P* and *Q* are two probability distributions. This is essentially cross-entropy minus the entropy, which can be thought of as quantifying how different the two distributions are. This is useful, for instance, when training generative adversarial networks. Minimizing the Kullback-Leibler divergence guarantees that the two distributions are similar.

Further study

Here, I would recommend two books for you:

- Pattern Recognition and Machine Learning by Christopher Bishop,
- The Elements of Statistical Learning by Trevor Hastie, Robert Tibshirani, and Jerome Friedman.

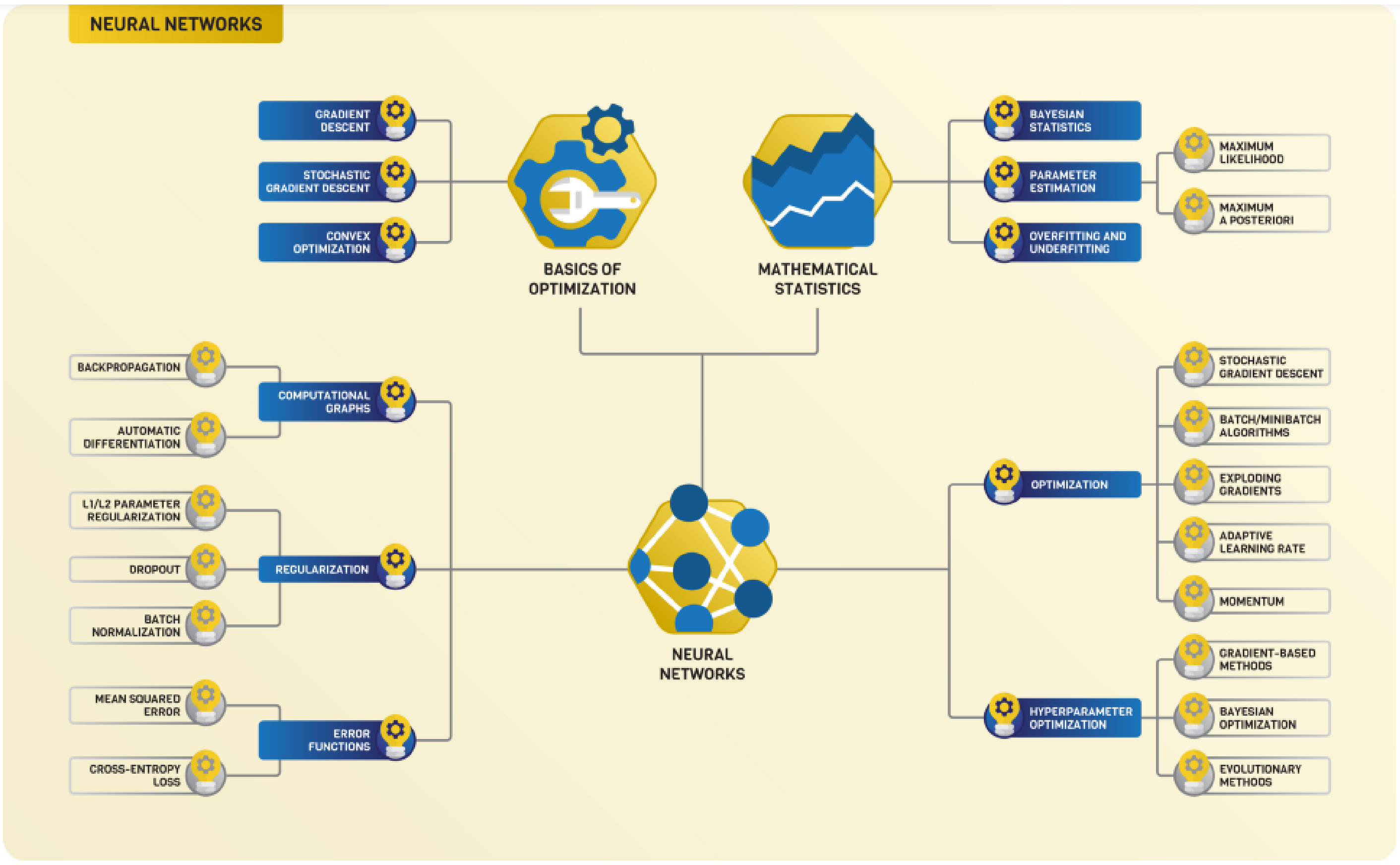
These are the two fundamental textbooks, and they teach you much more than probability theory. Both of them go way beyond the basics, but the corresponding chapters provide an excellent introduction.

Beyond the mathematical foundations

With this, we reviewed the necessary mathematics for understanding neural networks. Now, you are ready for the fun part: machine learning!

To really understand how neural networks work, you still have to learn some optimization and mathematical statistics. These subjects build upon the foundations we set. I won't go into details, since this is beyond the scope

Open in app Get started



If you would like to read more about some of these topics, check out some of my articles below!

The statistical foundations of machine learning

A look beyond function fitting

towardsdatascience.com

How to build a DIY deep learning framework in NumPy

Understanding the fine details of neural nets by building one from scratch

towardsdatascience.com

The mathematics of optimization for deep learning

A brief guide about how to minimize a function with millions of variables

towardsdatascience.com

If you love taking machine learning concepts apart and understanding what makes them tick, we have a lot in common. Check out my blog, where I frequently publish technical posts like this!