Member-only story

I thought I knew how LLMs learnt until I learnt this.

Meet the algorithm that secretly taught every AI to think

9 min read · 4 days ago

Nikhil Anand Follow

Listen Share More

All my articles are free to read. *Non-members can read for free by clicking this link.*

· · ·

Credits: Leonardo AI

Most people flinch when they hear the word *math*.

But the truth is, behind every equation hides a picture that makes everything suddenly click.

And nowhere is that truer than in AI.

In this blog, I'll take you through the algorithm that powers the training of every AI model.

Gradient Descent.

Most importantly, I'll show you the pictures hiding behind the math, so you'll never look at equations the same way again.

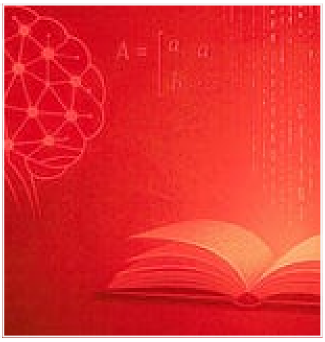
. . .

If you like learning AI concepts through easy-to-understand diagrams, I've created a free resource that organises all my work in one place — feel free to check it out!

Your free one-stop guide to AI in 2025

The only guide you'll ever need to master AI and LLMs.

makingaieasy.substack.com



. . .

Before we begin...

Last week, I wrote a blog about exactly what this field of optimisation is all about.

I'd encourage you to check out that previous blog out if you need a refresher on what "optimisation" means:

The learning engine inside every model

A straightforward guide to optimisation and the math of getting better

open.substack.com

Last time, we talked about how these problems aren't always well-defined.

That's exactly why we need **iterative methods**, which can handle uncertainty and gradual improvement where exact solutions don't exist.

This time, we'll focus on those iterative methods, and why they're especially important in AI.

Framing the problem

Remember our temperature problem from last time?

We're going to revisit that with a new perspective.

If you don’t remember, we had had a function that determined the temperature of every point on a 2D sheet.

For example, if the temperature function is $T = x^2 + y^2$, then given the (x, y) coordinates of a point on the sheet, I can tell you its temperature.

If you look at the function, it’s obvious that the temperature is minimised at (0, 0) where $T=0$.

That’s because both x^2 and y^2 are positive, so the overall function can’t be less than zero.

But sometimes it isn’t this obvious. Today, we’ll solve this same problem using the “iterative” method, so you get a feel of what that method entails for problems that are less obvious.

What exactly are iterative methods?

The core idea is simple.

We want to iteratively improve the function we're optimising by finding a direction to move in that reduces the function's value.

For instance, in the previous example, we can draw the **contours** of the function, which are just boundaries along which the value of the function is constant.

We see that to reduce the function's value, we have to move inwards.

The larger the circle, the larger the temperature of the point, so reducing the value of the function means we have to move inwards. (Image by author)

For example, if we start at (5, 5), we want to find a point where T is less than 50 (that is, 5^2+5^2).

The point (4, 4) is a possible candidate for that, since $4^2+4^2 = 32$.

If we keep taking steps along the decreasing temperature direction, we eventually reach the minimum.

How do we formalise these methods?

The more complex the function and the larger the dimensionality of the variable space, the harder it is to visualise this.

So we need a way of showing this analytically.

Luckily, we have vector notation to help us with that.

Here’s how we formalise it:

Taking a step along u_imp from the point x should lead us to a point where the function’s value is lesser. (Image by author)

Let’s break this down step by step.

Think of it this way.

The function f is just some mapping from some variable space to a scalar.

Earlier, we said $f(x,y) = x^2 + y^2$.

So we can think of that as a map from any pair of numbers to a single number. (5,5) maps to 50, (4,4) maps to 32, and so on.

Coming back to our formalisation:

Let's assume x is a point in 3D space.

We can’t visualise concepts in larger than 3 dimensions, so simplifying it to 3 makes it easier to picture what’s going on.

Now *x* takes a step along direction *u_imp,* which is also some 3D vector representing the direction we want to move in.

What our rule says is that anywhere along that line, the value of the objective *f* is lesser than its value at the initial point.

u_imp is a direction along which f is decreasing. (Image by author)

Let’s take a step back and remember what we want:

Given some objective function *f,* we want to minimise it.

So if we have this “decreasing direction”, we just need to keep taking steps along that direction to eventually reach the minimum!

But how do we find this decreasing direction?

That’s where a super powerful math concept called the derivative comes in.

The concept of derivatives

A derivative basically represents the “slope” of a function.

Imagine you’re in a valley between two mountains.

The slope is steeper further from the centre, but as you move towards the centre, the slope decreases.

It turns out this valley’s equation is $y = x^2$, and the slope at any point is $2x$.

The larger x is in either direction (positive or negative), the steeper it is.

We draw an arrow pointing along the positive x-axis if the slope is positive and along the negative x-axis if the slope is negative. (Image by author)

Clearly, the arrow only points along the direction where *f* is increasing. And moving in the opposite direction causes *f* to decrease.

Similarly, consider a constant function, which is a function whose value is constant regardless of the value of *x*.

For example, let’s look at the function f(*x*)=2.

The rate of change of the function is zero everywhere (since the function is constant), so the derivative of a constant function is zero.

Moving to multiple variables

Similarly, we can think of how this changes when we have 2 variables.

Consider $f(x, y) = x^2 + y^2$.

Now we have to consider a concept called the **partial derivative**, which just means:

> *Differentiate with respect to x keeping y constant, and differentiate with respect to y keeping x constant.*

We keep y constant while finding the partial derivative wrt x, and keep x constant while finding the partial derivative wrt y. (Image by author)

Each partial derivative tells us the slope along that specific axis.

The partial derivative wrt x represents the rate of change of the function parallel to the x-axis. (Image by author)

For instance, let’s take the point $(x, y) = (1, 2)$.

The partial derivatives are $\partial f/\partial x = 2(1) = 2$ and $\partial f/\partial y = 2(2) = 4$.

What if we add those two vectors of the partial derivatives ($2\hat{i}$ and $4\hat{j}$)? We get a net vector ($2\hat{i} + 4\hat{j}$).

For example, if we take the point (1, 2), the gradient of the function can be written by evaluating the derivatives at that point. (Image by author)

Turns out this is called the **gradient**, which happens to also be the direction of steepest ascent of the function.

The gradient

No matter how big the function is, you can find its gradient with a simple rule:

Partially differentiating it wrt all the variables of the function in vector form gives us the gradient. (Image by author)

> *And the **gradient is the direction of steepest ascent of any N-dimensional function.***

Let’s work through an example.

Let’s say we have a 3D function $f = x^2 + y^2 + z^3$ and we need the direction of steepest ascent at (1, 1, 2).

The gradient is:

We get a clear vector representing the steepest ascent. So we need to move opposite that direction if we want to reduce the value of the function.

The descent direction

If ∇f is the steepest ascent direction, the descent direction is -∇f.

Finding this descent direction and moving along it is what allows us to optimise and reduce the value of the function.

And guess what?

We just discovered the **gradient descent algorithm**, a super famous algorithm in machine learning.

What really is gradient descent?

Gradient descent just says this:

If we have a current point x_t and we know the function *f*, we can optimise the function by moving along the negative gradient from x_t.

The formula of gradient descent. If we can find the gradient of the function, this iterative method asks us to move along the negative gradient to obtain the next estimate of the minimum value of the function. (Image by author)

Here, α is called the **learning rate** or **step size**. It controls how big each step is.

Let's take a real-life example with the same temperature function we saw earlier.

Walking through the temperature example

Assume $f = x_1^2 + x_2^2$.

That's the same function we were looking into in the temperature example.

Let the point $x = [x_1, x_2]$, then:

We're writing the temperature function in terms of the 2D vector point rather than its components. (Image by author)

That means we can write this as:

The gradient is:

By taking the scalar 2 outside the column vector, we get a simplified expression for grad f. (Image by author)

So we can write the gradient descent expression like this:

Since gradf = 2x_t. (Image by author)

Assuming α = 0.1:

This is a super simple formula that represents the recursive update of each point x_t.

Let’s assume that we start at (1, 2), like in the previous example. That means:

So we can use the recursive formula to define any value of *x* in terms of the previous value and we can write iteratively that:

Thus:

But note that the larger *t* gets, the closer (0.8)^t gets to 0.

The calculator shows us why the larger t gets, the closer our fraction goes to zero. (Image by author)

That means x_t goes closer and closer to zero.

Which means (0, 0) is the minimum point of the function, and it’s the same minimum we found earlier.

But now we found it through a systematic, repeatable algorithm.

Why is gradient descent so important?

Everything in machine learning is optimisation.

LLMs, for example, have billions of parameters. Imagine a parameter vector like this:

Weight vectors in LLMs have billions of parameters to optimise. (Image by author)

For an LLM, we want to reduce the error in the words the LLM outputs. That becomes the objective we want to minimise.

Here’s the parallel:

The objective and the parameters are the components of any optimisation problem, whether it’s math or ML. (Image by author)

We need to optimise a billion weights. So the vector *x* here is a billion-dimensional vector representing all those weights.

But the exact same gradient descent idea works.

Now we try to optimise a billion-parameter vector down using the gradient direction of the function we want to optimise. (Image by author)

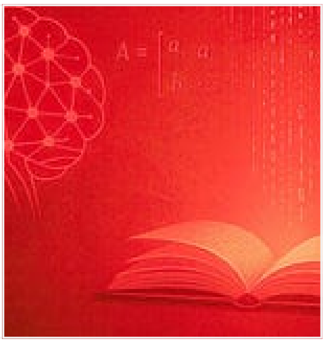
That’s the power of the analytical vector way of looking at it. The same simple formula works whether you have 2 parameters or 2 billion parameters.

If you like learning AI concepts through easy-to-understand diagrams, I’ve created a free resource that organises all my work in one place — feel free to check it out!

Your free one-stop guide to AI in 2025

The only guide you’ll ever need to master AI and LLMs.

makingaieasy.substack.com



Wrapping up

In this blog, we walked through the mathematical foundation of how AI systems learn.

We started with a simple temperature optimisation problem and built our way up to understanding gradient descent, the algorithm that powers every modern AI model.

What's exciting about this is the universality of it.

Whether you're training a model to predict temperatures, recognise images, or generate human-like text, you're fundamentally doing the same thing:

Iteratively moving in the direction that reduces your error.

Understanding this core algorithm is the first step to understanding how any AI system learns. Everything else is built on top of this foundation.

Acknowledgements

All diagrams were made by me on Canva.

Mathematics Artificial Intelligence Machine Learning Gradient Descent Differential Equations

Responses (9)

Bgerby

What are your thoughts?

Lauri Viisanen
4 days ago

> Meet the algorithm that secretly taught every AI to think.

Think? AI? Ellaborate.

15 1 reply Reply

AIPPV
2 days ago

Thanks, very well explained.

12 Reply

Rocky Wilkes
15 hours ago

Cool piece. The math behind AI might feel scary until you see it as a sketch, then it starts making sense.

2 Reply

See all responses

More from Nikhil Anand and AI Advances

In AI Advances by Nikhil Anand

I wasted months running slow LLMs before learning this

Why your LLM is running at just 10% of its potential speed

Oct 10 841 11

In AI Advances by Kuriko Iwai

Mastering Autoencoders (AEs) for Advanced Unsupervised Learning

Explore the core mechanics of AEs with essential regularization techniques and various layer architectures

Oct 2 456 3

In AI Advances by Jose Crespo, PhD

AI Is Producing More Garbage Code Than Ever

Soon, companies will be forced to rehire real human programmers and stop burning cash on AI.

Oct 6 572 31

In AI Advances by Nikhil Anand

If you still think AI is magic, read this.

AI might seem mysterious and weird, but it’s a lot simpler than you think.

Oct 14 543 16

See all from Nikhil Anand

See all from AI Advances

Recommended from Medium

In Towards AI by Teja Kusireddy

We Spent $47,000 Running AI Agents in Production. Here’s What Nobody Tells You About A2A and MCP.

Multi-agent systems are the future. Agent-to-Agent (A2A) communication and Anthropic’s Model Context Protocol (MCP) are revolutionary. But...

Oct 16 995 15

In AI Software Engineer by Joe Njenga

This Viral DeepSeek OCR Model Is Changing How LLMs Work

This DeepSeek OCR model hit an overnight success not seen in any other release—4k+ GitHub stars in less than 24 hours and more than 100k...

Oct 23 272 4

In Data Science Collective by Simon Greenman

The AI Vibe Coding Paradox: Why Experience Matters More Than Ever

AI can now code faster than any developer. But without experienced leadership, it breaks down in record time

4d ago 153 12

Ignacio de Gregorio

The Whale Strikes Again

DeepSeek-OCR is much more than it seems

Oct 23 1.1K 19

In Towards Deep Learning by Sumit Pandey

Why Everyone Will Want DGX Spark on Their Desk — Yes, Everyone

I just saw this picture today and was amazed, I've been waiting for this moment for a long time. (No, it's not Elon.) It's that tiny...

Oct 14 68 11

In Artificial Intelligence in Plain English by DrSwarnenduAI

A Geometric Interpretation of Attention: The Linear Algebra of Finding Your Match

Good day and Welcome back 😎 !

6d ago 110 3

See more recommendations