Open in app ↗

⬤❚⬤   🔍 Search   ✍ Write   🔔   👤

✦ **Jump-start your best year yet:** Become a member and get 25% off the first year   ✕

Understanding Backpropagation Algorithm

Learn the nuts and bolts of a neural network's most important ingredient

👤 Simeon Kostadinov · Follow
   Published in Towards Data Science · 8 min read · Aug 8, 2019

👏 2.6K   💬 19   🔖⁺   ▶   ↥   •••

"A man is running on a highway" — photo by Andrea Leopardi on Unsplash

Backpropagation algorithm is probably the most fundamental building block in a neural network. It was first introduced in 1960s and almost 30 years later (1989) popularized by Rumelhart, Hinton and Williams in a paper called *"Learning representations by back-propagating errors"*.

The algorithm is used to effectively train a neural network through a method called chain rule. In simple terms, after each forward pass through a network, backpropagation performs a backward pass while adjusting the model's parameters (weights and biases).

In this article, I would like to go over the mathematical process of training and optimizing a simple 4-layer neural network. *I believe this would help the*

reader understand how backpropagation works as well as realize its importance.

Define the neural network model

The 4-layer neural network consists of 4 neurons for the **input layer,** 4 neurons for the **hidden layers** and 1 neuron for the **output layer**.
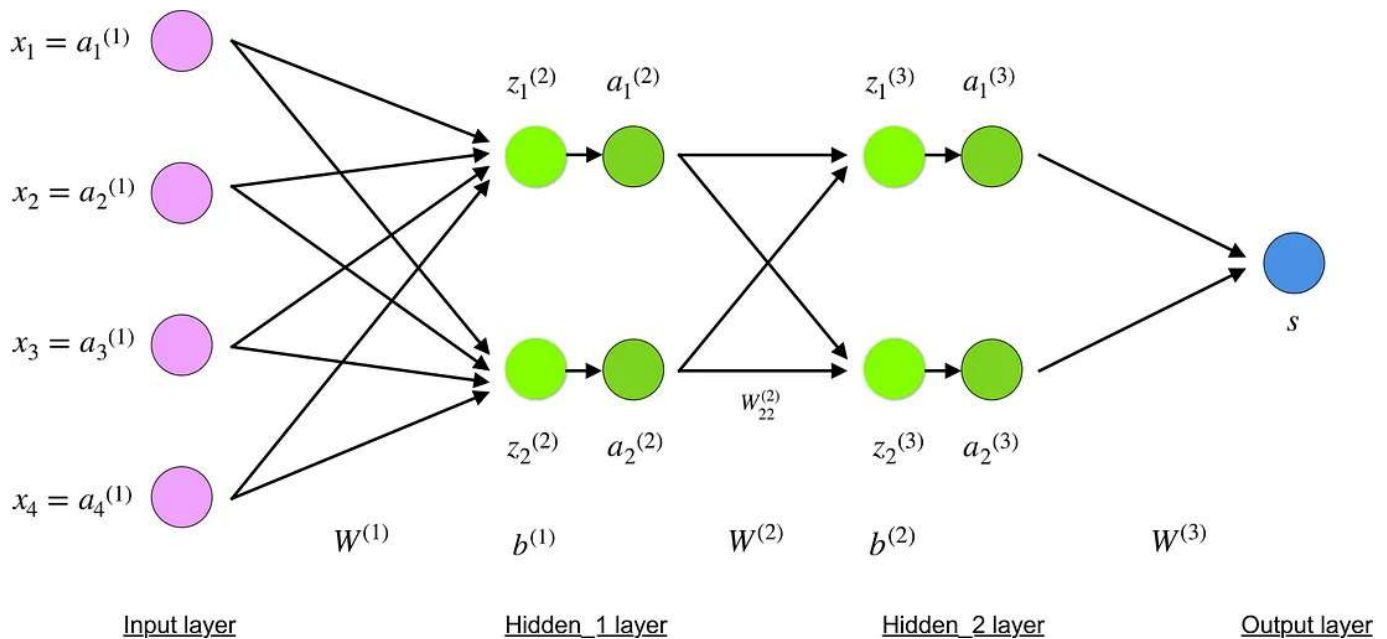
Simple 4-layer neural network illustration

Input layer

The neurons, colored in **purple**, represent the input data. These can be as simple as scalars or more complex like vectors or multidimensional matrices.

$$x_i = a_i^{(1)}, i \in 1,2,3,4$$

Equation for input x_i

The first set of activations (*a*) are equal to the input values. *NB: "activation" is the neuron's value after applying an activation function. See below.*

Hidden layers

The final values at the hidden neurons, colored in **green,** are computed using z^l — weighted inputs in layer l, and a^l— activations in layer l. For layer 2 and 3 the equations are:

- $l = 2$

$$z^{(2)} = W^{(1)}x + b^{(1)}$$

$$a^{(2)} = f(z^{(2)})$$

Equations for z² and a²

- $l = 3$

$$z^{(3)} = W^{(2)}a^{(2)} + b^{(2)}$$

$$a^{(3)} = f(z^{(3)})$$

Equations for z³ and a³

W^2 and W^3 are the weights in layer 2 and 3 while b² and b³ are the biases in those layers.

Activations a^2 and a^3 are computed using an activation function f. Typically, this **function f is non-linear** (e.g. <u>sigmoid</u>, <u>ReLU</u>, <u>tanh</u>) and allows the network to learn complex patterns in data. We won't go over the details of how activation functions work, but, if interested, I strongly recommend reading <u>this great article</u>.

Looking carefully, you can see that all of x, z^2, a^2, z^3, a^3, W^1, W^2, b^1 and b^2 are missing their subscripts presented in the 4-layer network illustration above. **The reason is that we have combined all parameter values in matrices, grouped by layers.** This is the standard way of working with neural networks and one should be comfortable with the calculations. However, I will go over the equations to clear out any confusion.

Let's pick layer 2 and its parameters as an example. The same operations can be applied to any layer in the network.

- W^1 is a weight matrix of shape *(n, m)* where n is the number of output neurons (neurons in the next layer) and m is the number of input neurons (neurons in the previous layer). For us, *n = 2* and *m = 4*.

$$W^{(1)} = \begin{bmatrix} W_{11}^{(1)} & W_{12}^{(1)} & W_{13}^{(1)} & W_{14}^{(1)} \\ W_{21}^{(1)} & W_{22}^{(1)} & W_{23}^{(1)} & W_{24}^{(1)} \end{bmatrix}$$

Equation for W^1

NB: The first number in any weight's subscript matches the index of the neuron in the next layer (in our case this is the *Hidden_2 layer*) **and the second number matches the index of the neuron in previous layer** (in our case this is the *Input layer*).

- x is the input vector of shape *(m, 1)* where m is the number of input neurons. For us, *m = 4*.

$$x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix}$$

Equation for x

- b^1 is a bias vector of shape *(n , 1)* where n is the number of neurons in the current layer. For us, $n = 2$.

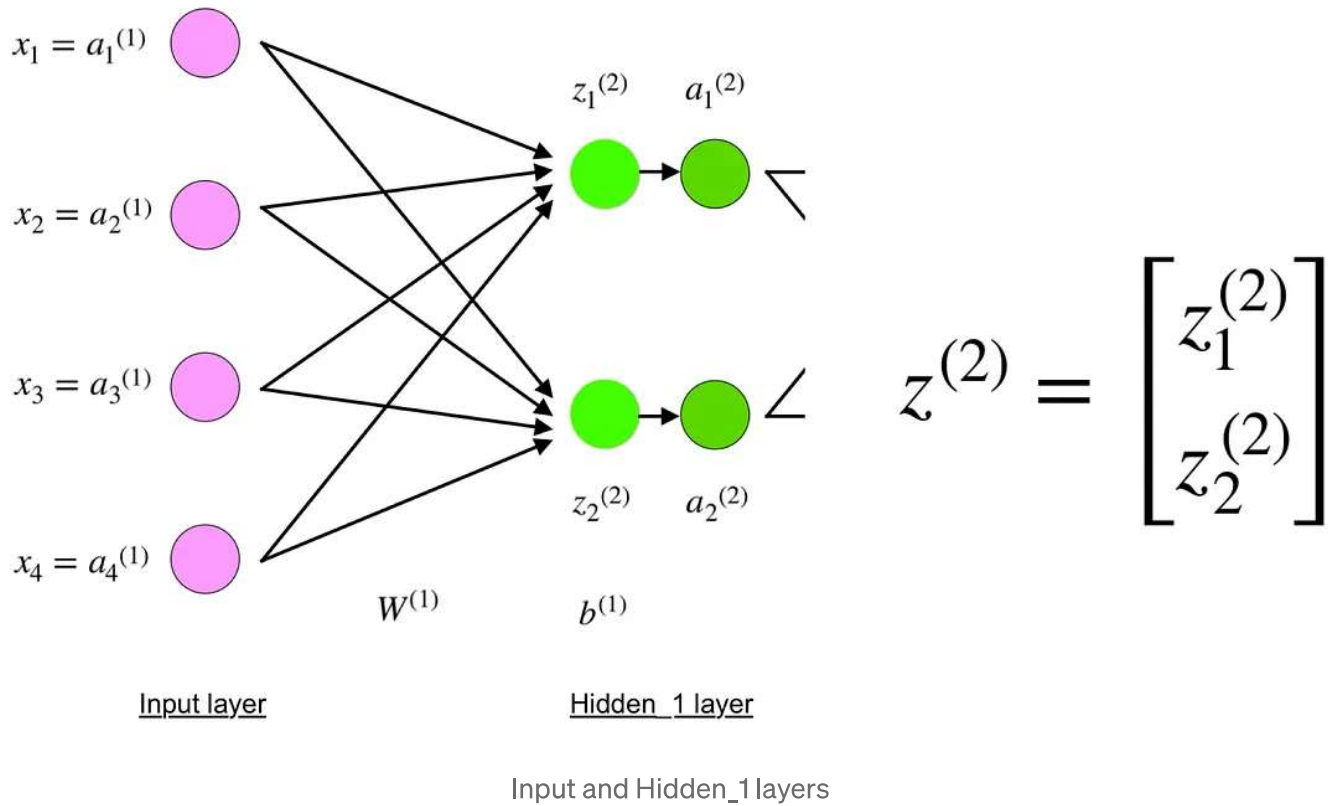$$b^{(1)} = \begin{bmatrix} b_1^{(1)} \\ b_2^{(1)} \end{bmatrix}$$

Equation for b¹

Following the equation for z^2, we can use the above definitions of W^1, x and b^1 to derive "*Equation for z^2*":

$$z^{(2)} = \begin{bmatrix} W_{11}^{(1)}x_1 + W_{12}^{(1)}x_2 + W_{13}^{(1)}x_3 + W_{14}^{(1)}x_4 \\ W_{21}^{(1)}x_1 + W_{22}^{(1)}x_2 + W_{23}^{(1)}x_3 + W_{24}^{(1)}x_4 \end{bmatrix} + \begin{bmatrix} b_1^{(1)} \\ b_2^{(1)} \end{bmatrix}$$

Equation for z²

Now carefully observe the neural network illustration from above.

Input and Hidden_1 layers

You will see that z^2 can be expressed using $(z\_1)^2$ and $(z\_2)^2$ where $(z\_1)^2$ and $(z\_2)^2$ are the sums of the multiplication between every input x_i with the corresponding weight $(W\_ij)^1$.

This leads to the same "*Equation for z^2*" and proofs that the matrix representations for z^2, a^2, z^3 and a^3 are correct.

Output layer

The final part of a neural network is the output layer which produces the predicated value. In our simple example, it is presented as a single neuron, colored in **blue** and evaluated as follows:

$$s = W^{(3)}a^{(3)}$$

Equation for output s

Again, we are using the matrix representation to simplify the equation. One can use the above techniques to understand the underlying logic. **Please leave any comments below if you find yourself lost in the equations — I would love to help!**

Forward propagation and evaluation

The equations above form network's forward propagation. Here is a short overview:

$$x = a^{(1)} \qquad \textit{Input layer}$$

$$z^{(2)} = W^{(1)}x + b^{(1)} \qquad \textit{neuron value at } Hidden_1 \textit{ layer}$$

$$a^{(2)} = f(z^{(2)}) \qquad \textit{activation value at } Hidden_1 \textit{ layer}$$

$$z^{(3)} = W^{(2)}a^{(2)} + b^{(2)} \qquad \textit{neuron value at } Hidden_2 \textit{ layer}$$

$$a^{(3)} = f(z^{(3)}) \qquad \textit{activation value at } Hidden_2 \textit{ layer}$$

$$s = W^{(3)}a^{(3)} \qquad \textit{Output layer}$$

Overview of forward propagation equations colored by layer

The final step in a forward pass is to evaluate the **predicted output s** against an **expected output y.**

The output y is part of the training dataset (x, y) where x is the input (as we saw in the previous section).

Evaluation between *s* and *y* happens through a **cost function.** This can be as simple as <u>MSE</u> (mean squared error) or more complex like <u>cross-entropy</u>.

We name this cost function *C* and denote it as follows:

$$C = cost(s, y)$$

Equation for cost function C

were *cost* can be equal to MSE, cross-entropy or <u>any other cost function</u>.

Based on *C*'s value, the model "knows" how much to adjust its parameters in order to get closer to the expected output *y*. This happens using the backpropagation algorithm.

Backpropagation and computing gradients

According to the paper from 1989, backpropagation:

> *repeatedly adjusts the weights of the connections in the network so as to minimize a measure of the difference between the actual output vector of the net and the desired output vector.*

and

> *the ability to create useful new features distinguishes back-propagation from earlier, simpler methods...*

In other words, **backpropagation aims to minimize the cost function by adjusting network's weights and biases.** The level of adjustment is

determined by the gradients of the cost function with respect to those parameters.

One question may arise — **why computing gradients**?

To answer this, we first need to revisit some calculus terminology:

- *Gradient of a function C(x_1, x_2, …, x_m) in point x is a vector of the <u>partial derivatives</u> of C in x.*

$$\frac{\partial C}{\partial x} = [\frac{\partial C}{\partial x_1}, \frac{\partial C}{\partial x_2}, \ldots, \frac{\partial C}{\partial x_m}]$$

Equation for derivative of C in x

- <u>*The derivative of a function C measures the sensitivity to change of the function value (output value) with respect to a change in its argument x (input value).*</u> *In other words, the derivative tells us the direction C is going.*

- *The gradient shows how much the parameter x needs to change (in positive or negative direction) to minimize C.*

Compute those gradients happens using a technique called <u>chain rule</u>.

For a single weight *(w_jk)^l,* the gradient is:

$$\frac{\partial C}{\partial w_{jk}^l} = \frac{\partial C}{\partial z_j^l}\frac{\partial z_j^l}{\partial w_{jk}^l} \qquad chain\ rule$$

$$z_j^l = \sum_{k=1}^{m} w_{jk}^l a_k^{l-1} + b_j^l \qquad by\ definition$$

$$m \ - \ number\ of\ neurons\ in\ l-1\ layer$$

$$\frac{\partial z_j^l}{\partial w_{jk}^l} = a_k^{l-1} \qquad by\ differentiation\ (calculating\ derivative)$$

$$\frac{\partial C}{\partial w_{jk}^l} = \frac{\partial C}{\partial z_j^l}a_k^{l-1} \qquad final\ value$$

Equations for derivative of C in a single weight (w_jk)^l

Similar set of equations can be applied to *(b_j)^l*:

$$\frac{\partial C}{\partial b_j^l} = \frac{\partial C}{\partial z_j^l}\frac{\partial z_j^l}{\partial b_j^l} \qquad chain\ rule$$

$$\frac{\partial z_j^l}{\partial b_j^l} = 1 \qquad by\ differentiation\ (calculating\ derivative)$$

$$\frac{\partial C}{\partial b_j^l} = \frac{\partial C}{\partial z_j^l}1 \qquad final\ value$$

Equations for derivative of C in a single bias (b_j)^l

The common part in both equations is often called *"local gradient"* and is expressed as follows:

$$\delta_j^l = \frac{\partial C}{\partial z_j^l} \qquad local\ gradient$$

Equation for local gradient

The *"local gradient"* can easily be determined using the chain rule. I won't go over the process now but if you have any questions, please comment below.

The gradients allow us to optimize the model's parameters:

$$while\ (termination\ condition\ not\ met)$$

$$w := w - \epsilon \frac{\partial C}{\partial w}$$

$$b := b - \epsilon \frac{\partial C}{\partial b}$$

$$end$$

Algorithm for optimizing weights and biases (also called "Gradient descent")

- Initial values of w and b are randomly chosen.

- Epsilon (e) is the <u>learning rate</u>. It determines the gradient's influence.

- w and b are matrix representations of the weights and biases. Derivative of C in w or b can be calculated using partial derivatives of C in the individual weights or biases.

- Termination condition is met once the cost function is minimized.

I would like to dedicate the final part of this section to a simple example in which we will calculate the gradient of C with respect to a single weight $(w\_22)^2$.

Let's zoom in on the bottom part of the above neural network:
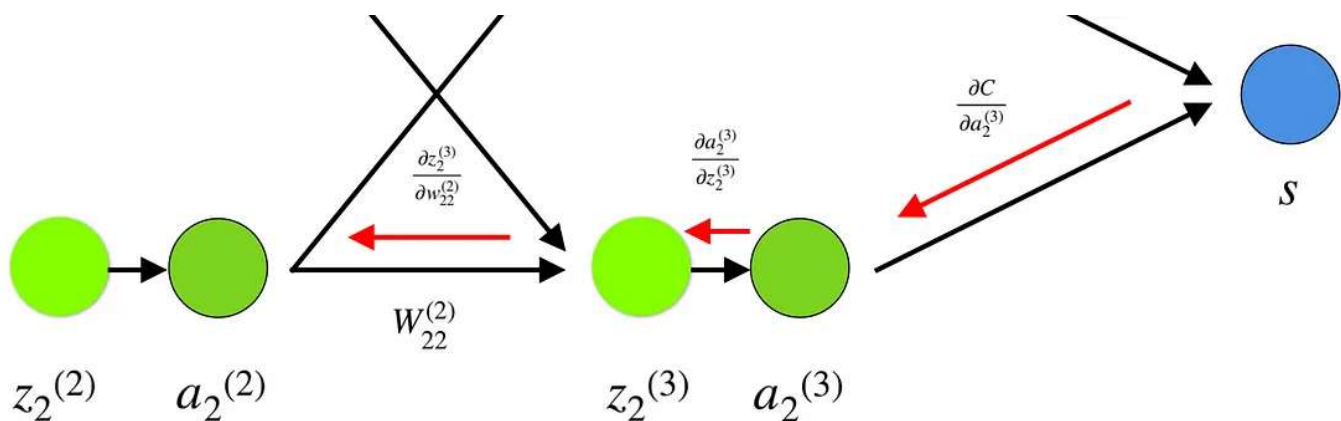
Visual representation of backpropagation in a neural network

Weight $(w\_22)^2$ connects $(a\_2)^2$ and $(z\_2)^2$, so computing the gradient requires applying the chain rule through $(z\_2)^3$ and $(a\_2)^3$:

$$\frac{\partial C}{\partial w_{22}^{(2)}} = \frac{\partial C}{\partial z_2^{(3)}} \cdot \frac{\partial z_2^{(3)}}{\partial w_{22}^{(2)}} = \frac{\partial C}{\partial a_2^{(3)}} \cdot \frac{\partial a_2^{(3)}}{\partial z_2^{(3)}} \cdot a_2^{(2)} = \frac{\partial C}{\partial a_2^{(3)}} \cdot f'(z_2^{(3)}) \cdot a_2^{(2)}$$

Equation for derivative of C in (w_22)²

Calculating the final value of derivative of C in $(a\_2)^3$ requires knowledge of the function C. Since C is dependent on $(a\_2)^3$, calculating the derivative should be fairly straightforward.

I hope this example manages to throw some light on the mathematics behind computing gradients. To further enhance your skills, I strongly recommend watching Stanford's NLP series where Richard Socher gives 4 great explanations of backpropagation.

Final remarks

In this article, I went through a detailed explanation of how backpropagation works under the hood using mathematical techniques like computing gradients, chain rule etc. *Knowing the nuts and bolts of this algorithm will fortify your neural networks knowledge and make you feel comfortable to take on more complex models. Enjoy your deep learning journey!*

Thank you for the reading. Hope you enjoyed the article 🤩 and I wish you a great day!

Artificial Intelligence   Deep Learning   Algorithms   Mathematics   Data Science

Written by Simeon Kostadinov

Follow

1.93K Followers · Writer for Towards Data Science

Obsessed with creating a positive impact. Love blogging about AI and reading books. For more content, follow me 👉 https://www.linkedin.com/in/simeonkostadinov/

More from Simeon Kostadinov and Towards Data Science

Simeon Kostadinov in Towards Data Science

Sheila Teo in Towards Data Science

Understanding GRU Networks

In this article, I will try to give a fairly simple and understandable explanation of one reall...

6 min read · Dec 16, 2017

👏 6.5K   💬 28

How I Won Singapore's GPT-4 Prompt Engineering Competition

A deep dive into the strategies I learned for harnessing the power of Large Language...

✦ · 24 min read · Dec 29, 2023

👏 9.95K   💬 118

Thu Vu in Towards Data Science

Simeon Kostadinov in Towards Data Science

How to Learn AI on Your Own (a self-study guide)

Understanding Encoder-Decoder Sequence to Sequence Model

If your hands touch a keyboard for work, Artificial Intelligence is going to change your…

In this article, I will try to give a short and concise explanation of the sequence to…

✦ · 12 min read · Jan 5

5 min read · Feb 5, 2019

👏 2.2K   💬 24   🔖   •••

👏 2.9K   💬 12   🔖   •••

(See all from Simeon Kostadinov)   (See all from Towards Data Science)

Recommended from Medium
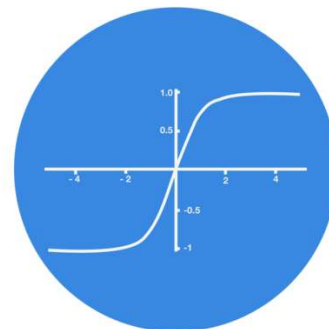
Puneeth Venugopal

Everton Gomede, PhD

The Chain Rule of Calculus: The Backbone of Deep Learning...

Introduction

Activation Functions and Loss Functions in Deep Learning:...

Introduction

4 min read · Oct 15, 2023

5 min read · Oct 8, 2023

👏 58   💬 1   🔖⁺   •••

👏 100   💬   🔖⁺   •••

Lists

Predictive Modeling w/ Python
20 stories · 828 saves

Natural Language Processing
1118 stories · 591 saves

ChatGPT prompts
34 stories · 1004 saves

ChatGPT
23 stories · 410 saves

btd

Koushik

Learning Paths: Understanding and Implementing Backpropagation i...

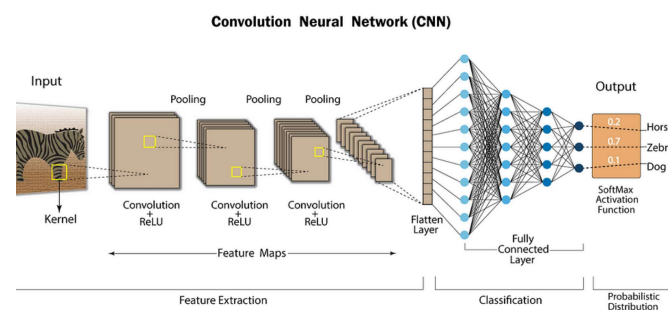Backpropagation, short for "backward propagation of errors," is a supervised...

Understanding Convolutional Neural Networks (CNNs) in Depth

Convolutional Neural Networks skillfully capturing and extracting patterns from data,...
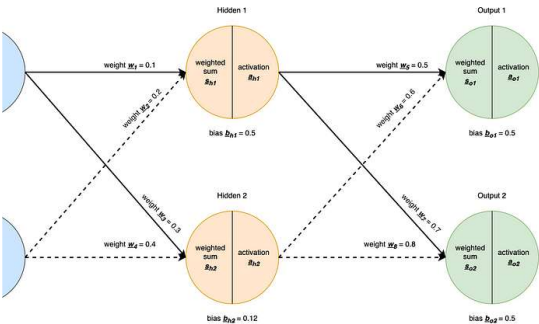
✨ · 5 min read · Nov 16, 2023

12 min read · Nov 28, 2023

👤 Long Nguyen   👤 Fareed Khan in Level Up Coding

Building a Neural Network from Scratch (with Backpropagation)

Solving Transformer by Hand: A Step-by-Step Math Example

Unveiling the magic of neural networks: from bare Python to TensorFlow. A hands-on…

Performing numerous matrix multiplications to solve the encoder and decoder parts of th…

11 min read   ·   Nov 18, 2023   13 min read   ·   Dec 18, 2023

See more recommendations