Recurrent Neural Networks (RNN) from Basic to Advanced

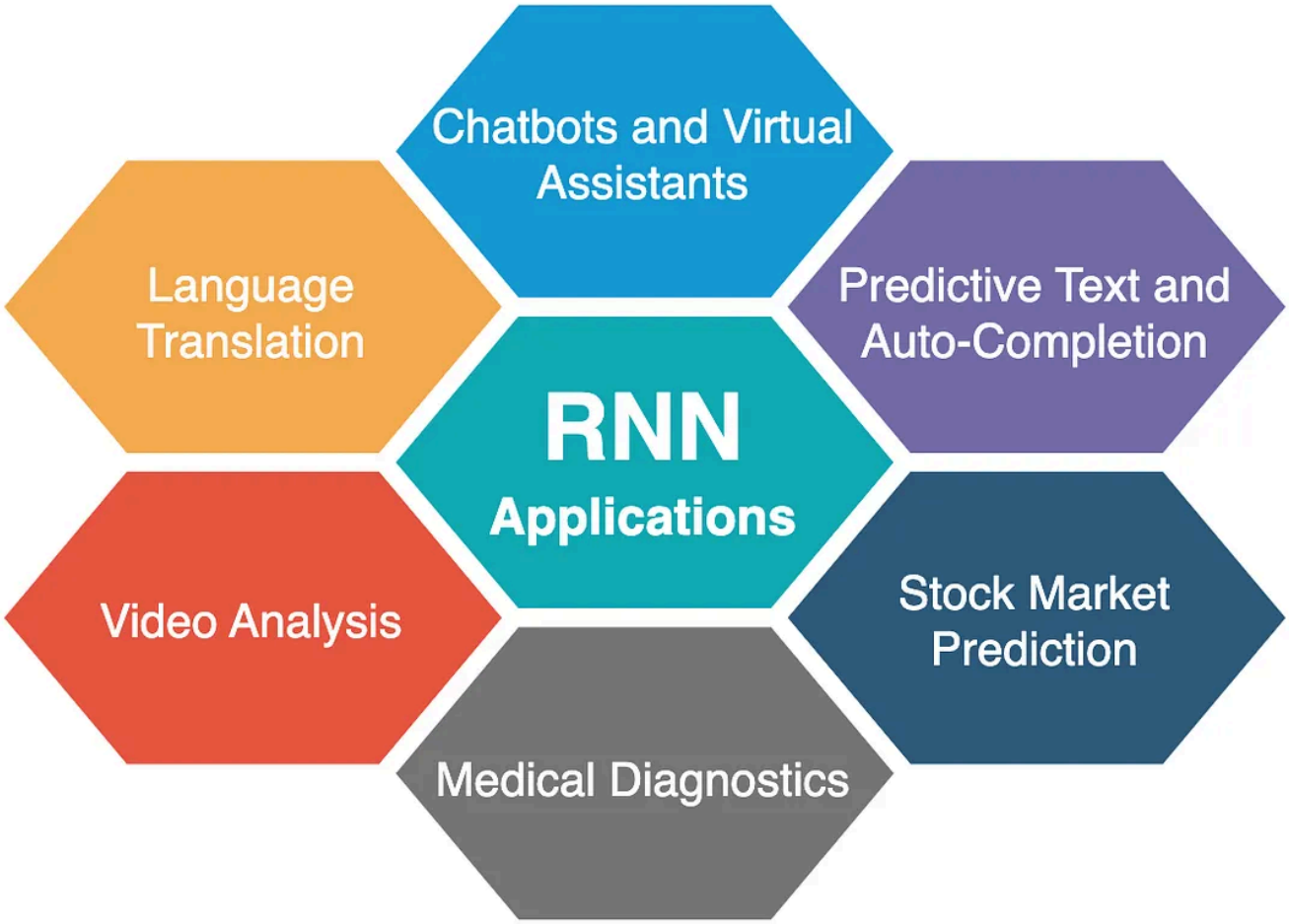
9 min read · Mar 25, 2024

Sachin Soni Follow

Listen Share

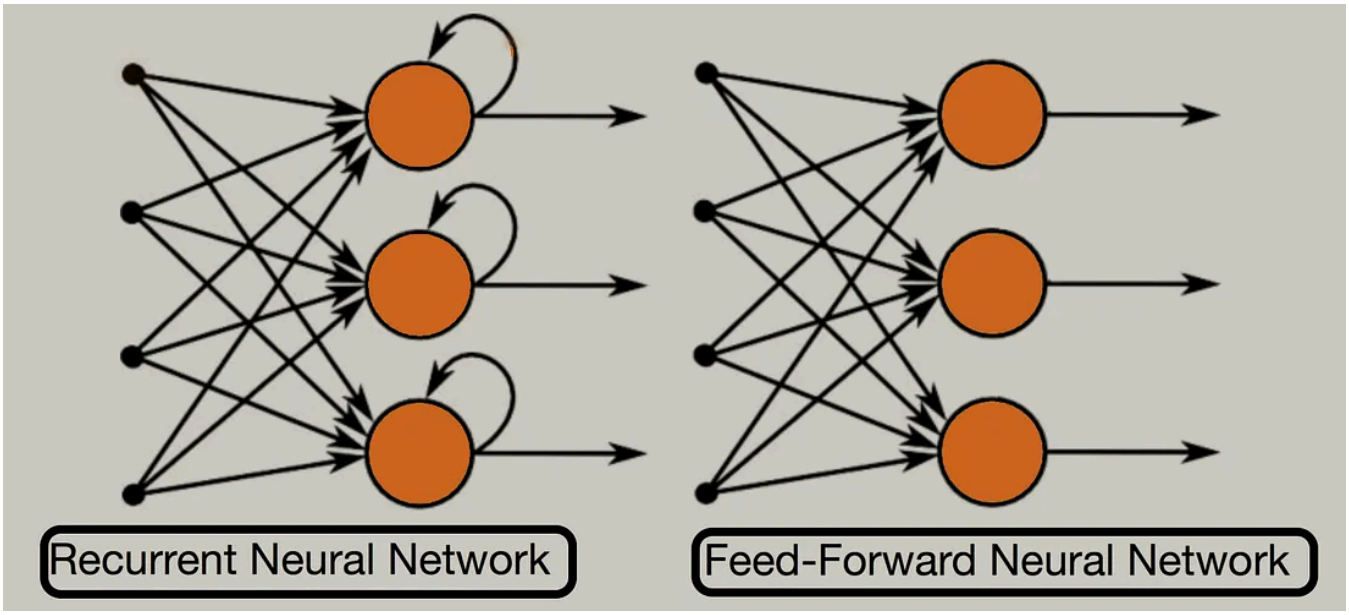
Ever wonder how your phone translates languages or predicts the next word you're typing? It's all thanks to something called a recurrent neural network, or RNN for short. Think of RNNs as super-smart machines that can learn from sequences, like sentences or music.

In this blog you will learn RNN basics that will helps you for learning Large Language Model to be more engaging and informative.



What is RNN ?

Recurrent neural networks (RNNs) are a type of artificial neural network **specifically designed to handle sequential data.**



ANNs and CNNs are example of Feed-Forward Neural Network while In RNNs , Information can flow back and forth through internal loops, allowing the network to consider past information when processing the current input.

Unlike traditional neural networks where each input is independent, RNNs can access and process information from previous inputs. This makes them particularly useful for tasks that involve sequences, like text, speech, or time series data.

What is Sequential data ?

Sequential data is information that has a specific order and where the order matters. Each piece of data in the sequence is related to the ones before and after it, and this order provides context and meaning to the data as a whole.

Here's an example to illustrate:

Imagine a sentence like "The quick brown fox jumps over the lazy dog." Each word in the sentence is a piece of data. The order of the words is crucial because it determines the meaning of the sentence. "Fox brown quick the jumps over lazy dog" wouldn't make much sense, right?

Here are some other common types of sequential data:

- **Time series data:** This refers to data points collected at regular intervals over time. Examples include stock prices, temperature readings, or website traffic. The order of the data points matters because it shows how the value changes over time.
- **Natural language text:** All written language is sequential. The order of words in a sentence or paragraph is essential for conveying meaning and understanding the relationships between ideas.
- **Speech signals:** Spoken language is another example of sequential data. The order of sounds like phonemes, syllables, and words is crucial for understanding the spoken message.

Why use RNN ?

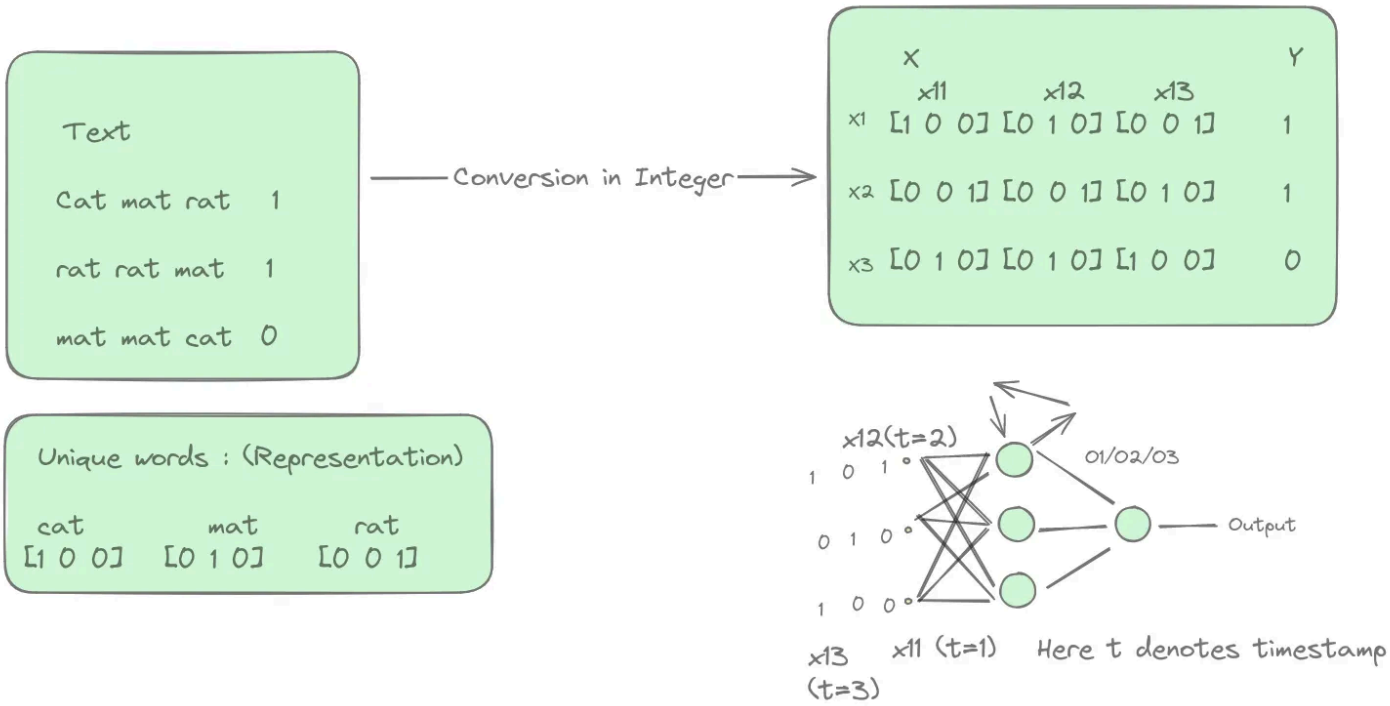
1. Traditional Artificial Neural Networks (ANNs) are powerful tools, but they struggle with sequential data like text because they require fixed-size inputs. Each input in an ANN is treated independently, making them unsuitable for tasks where the order and relationships between elements are crucial.
2. Suppose if we use zero padding concept in which Shorter sequences are padded with zeros at the end to reach the length of the longest sequence in the batch. These zeros act as placeholders and don't carry any meaningful information.

Padding introduces irrelevant zeros that the network needs to process alongside the actual data, increasing computational burden.

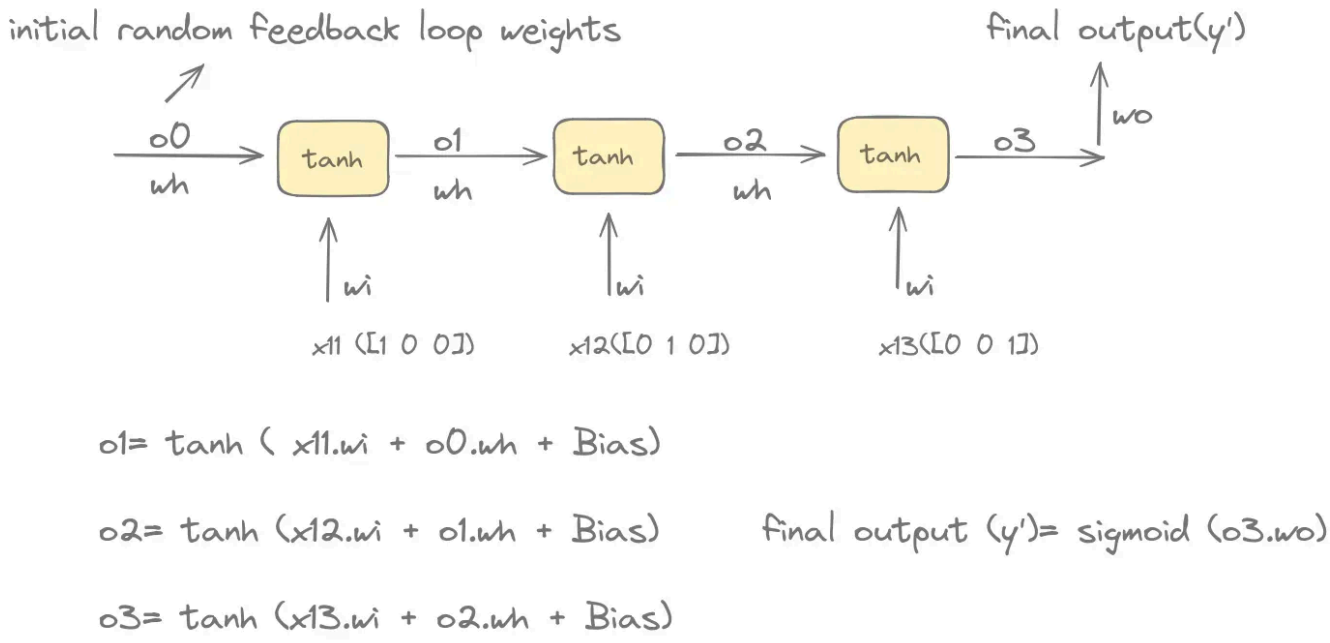
3. And also due to no sequences in passing input in ANN, we loss the context or sequential information. Apart from this if any user gives the input length of higher size that we expect then in that scenario we can nothing. For ex. we set our input size is 5 words but any user gives it 15 words at a time then in that case we can't handle it with ANN.

How RNNs works ?

The diagram depicts a simplified sentiment analysis process using a Recurrent Neural Network (RNN). Sample text is assigned sentiment labels (0/1 for positive/negative). Unique words are converted to numbers for the RNN to understand. These numbers are fed into the RNN one by one, with each word considered a single time step in the sequence. This demonstrates how RNNs can analyze sequential data like text to predict sentiment.



Let's say input weights are denoted by **wi**, output weights with **wo**, feedback loop weights with **wh** and biases for first hidden layer is **bi** and bias for output layer is **bo**, then mathematical formulation for above is given by :



o1= tanh (x11.wi + o0.wh + Bias)

o2= tanh (x12.wi + o1.wh + Bias)

o3= tanh (x13.wi + o2.wh + Bias)

final output (y')= sigmoid (o3.wo)

Types of RNN Architecture :

There are 3 types of RNN architecture which are :

1. One to Many
2. Many to One
3. Many to Many

1. One to Many :

In recurrent neural networks (RNNs), a “one-to-many” architecture represents a scenario where the network receives a **single input** but generates a **sequence of outputs**. This is the opposite of a many-to-one RNN.

Here’s a breakdown of how it works:

- **Single Input:** The RNN takes in a single piece of information as input. This could be an image, a musical note, a short sentence, or any data point that serves as a starting point for the network.
- **Multiple Outputs:** The RNN processes the input and generates a sequence of outputs over time. This sequence can vary in length depending on the specific task.

Here are some common applications of one to many RNNs:

- **Image Captioning:** Based on a single image input, the RNN generates a sentence or paragraph describing the image content (multiple outputs).
- **Music Generation:** The RNN receives a single starting note or short melody as input and produces a sequence of musical notes forming a complete piece (multiple outputs).

2. Many to One :

In recurrent neural networks (RNNs), a "many-to-one" architecture refers to a specific type of RNN where the network processes a **sequence of inputs** but produces a **single output**.

Here's a breakdown of how it works:

- **Many Inputs:** The RNN takes in a sequence of data points over time. This sequence could be words in a sentence, sensor readings over a period, or financial data points for multiple days.
- **Single Output:** After processing the entire sequence, the RNN generates a single output value. This output could be a classification (positive/negative sentiment), a prediction (next value in the time series), or a summary of the information in the sequence.

Here are some common applications of many-to-one RNNs:

- **Sentiment Analysis:** Given a sentence or review text (sequence of words), classify its overall sentiment (positive, negative, or neutral) as the single output.
- **Spam Detection:** Analyze an email's content (sequence of words) to determine if it's spam (single output).

3. Many to Many :

In recurrent neural networks (RNNs), a "many-to-many" architecture describes a scenario where the network processes a **sequence of inputs** and generates a corresponding **sequence of outputs**. This means both the input and output have multiple elements processed over time steps.

Here's a breakdown of the concept:

- **Multiple Inputs:** The RNN takes in a sequence of data points, similar to many-to-one RNNs. This sequence could be words in a sentence, sensor readings, or financial data points.
- **Multiple Outputs:** The RNN generates a new sequence of data points, with a length that may or may not be the same as the input sequence.

There are two main categories of many-to-many RNNs:

1. **Fixed Length:** In this case, the number of elements in the output sequence is the same as the number of elements in the input sequence. A common application is **named entity recognition (NER)**, where the network identifies and classifies each word in a sentence (input sequence) as a specific entity type (i.e. parts of speech) (output sequence).
2. **Variable Length:** Here, the output sequence can have a different length than the input sequence. This is particularly useful for tasks like **machine translation**, where the source sentence (input sequence) in one language might be translated into a longer or shorter sentence (output sequence) in another language.

Common Applications of Many-to-Many RNNs:

- **Machine Translation:** Translate text from one language to another (e.g., English to French).
- **Video Captioning:** Generate captions describing the content of a video (sequence of video frames as input, sequence of words as output).
- **Text Summarization:** Summarize a long document into a shorter version with key points (sequence of sentences as input, shorter sequence of sentences as output).

Network Architectures for Many-to-Many RNNs:

A common approach for many-to-many RNNs is the **encoder-decoder architecture**. Here, the network is divided into two parts:

- **Encoder:** This part processes the input sequence and captures its meaning in a hidden representation.
- **Decoder:** This part takes the hidden representation from the encoder and generates the output sequence element by element.

> ***Important note :-*** *One to One architecture is not a type of RNN because* ***a one-to-one situation involves a single input and a single output****, with no concept of sequence. However, for this specific task (single input to single output), both Convolutional Neural Networks (CNNs, for example image classification) and Artificial Neural Networks (ANNs) can be suitable candidates, depending on the nature of the input data.*

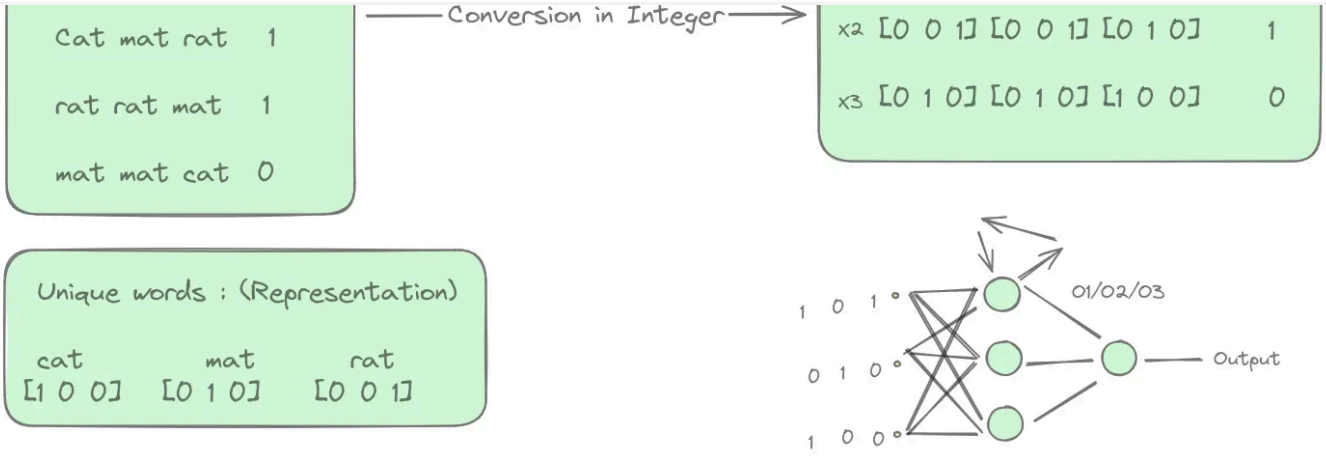
Backpropagation in RNNs :

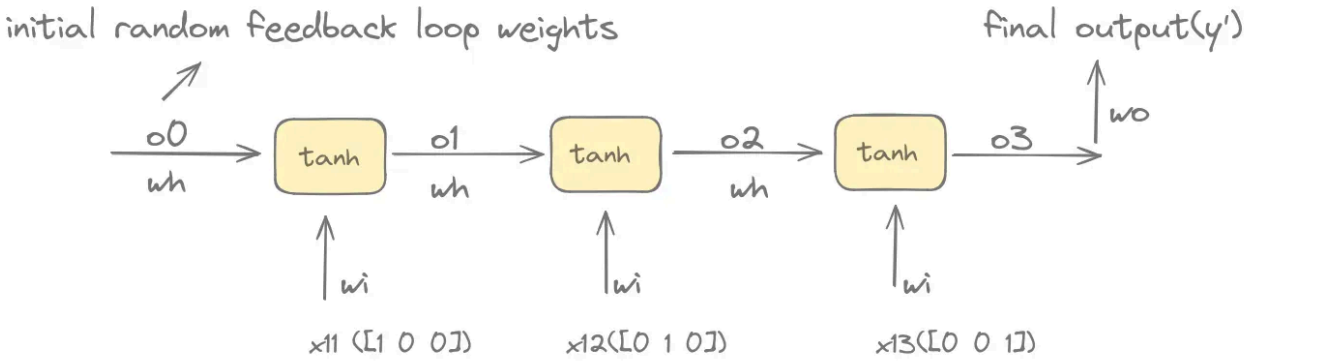
Backpropagation in RNNs, also known as Backpropagation Through Time (BPTT), is a crucial technique for training these networks. It allows them to learn and improve their performance on tasks involving sequential data.

Let's take an example to understand the back propagation in many to one RNNs architecture :

1. Forward Pass:

- The RNN processes a sequence of inputs one step at a time. At each step, the current input and the previous hidden state are fed into the network's activation function to generate a new hidden state and an output.





o1= tanh (x11.wi + o0.wh + Bias)

o2= tanh (x12.wi + o1.wh + Bias)

o3= tanh (x13.wi + o2.wh + Bias)

final output (y')= sigmoid (o3.wo)

2. Error Calculation:

- After processing the entire sequence, the difference (error) between the network's final output and the desired output for the sequence is calculated.

And the loss function is calculated by using log loss

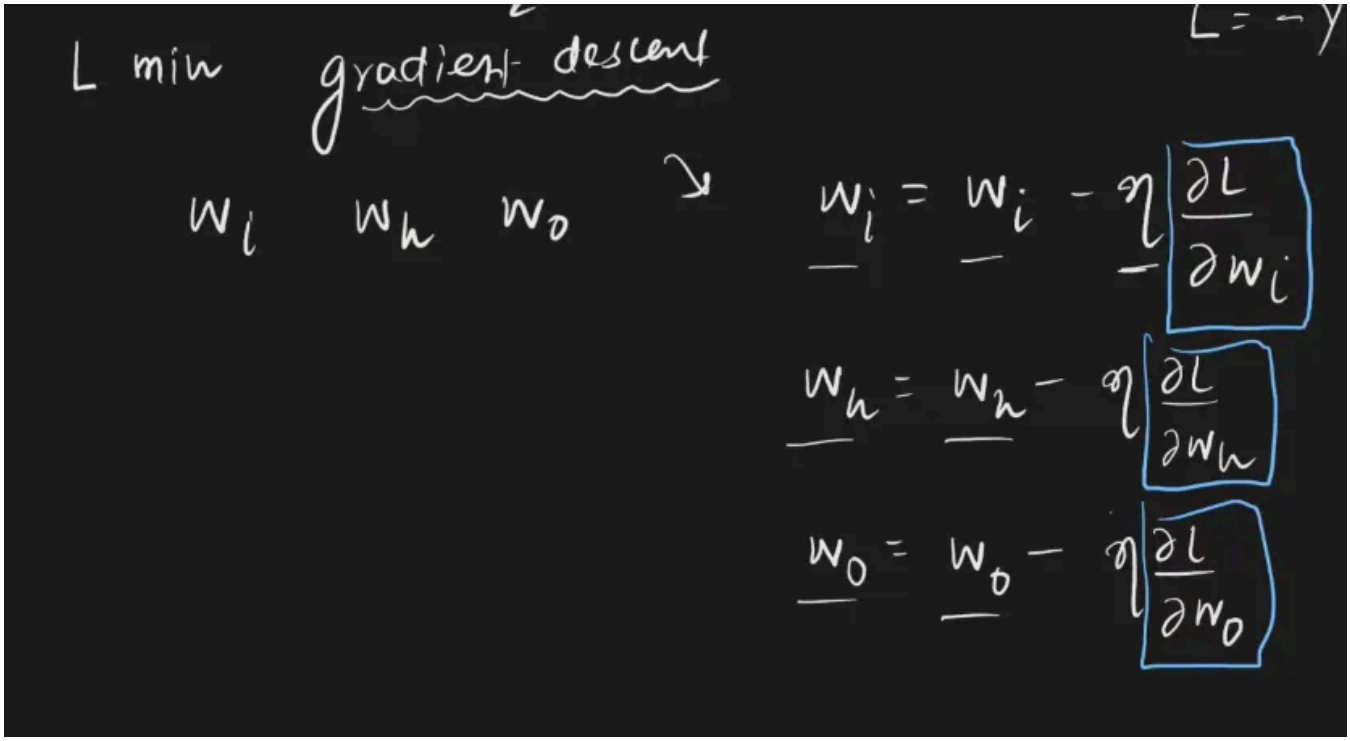
$$L = -y_i \log(y'_i) - (1-y_i)\log(1-y'_i)$$

where, y_i = Actual ground truth value

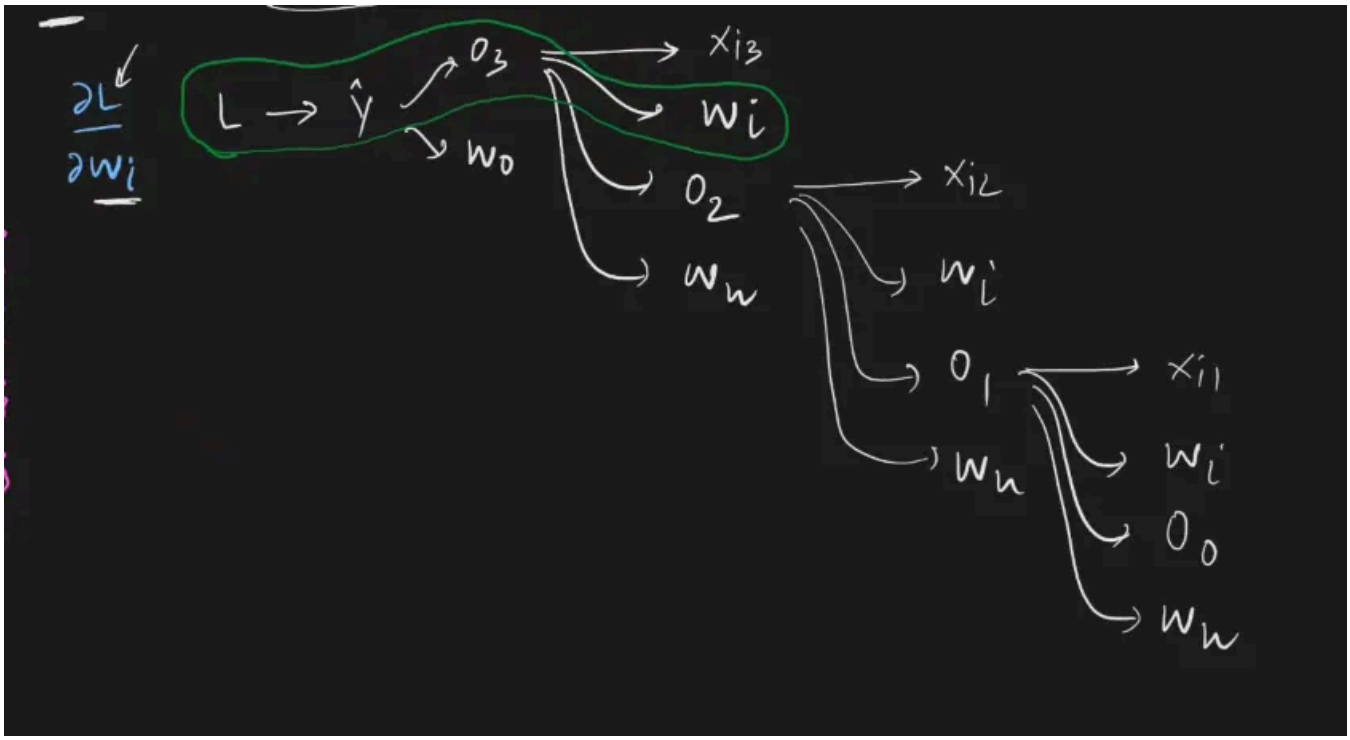
y'_i = Predicted value by model

3. Weight Update:

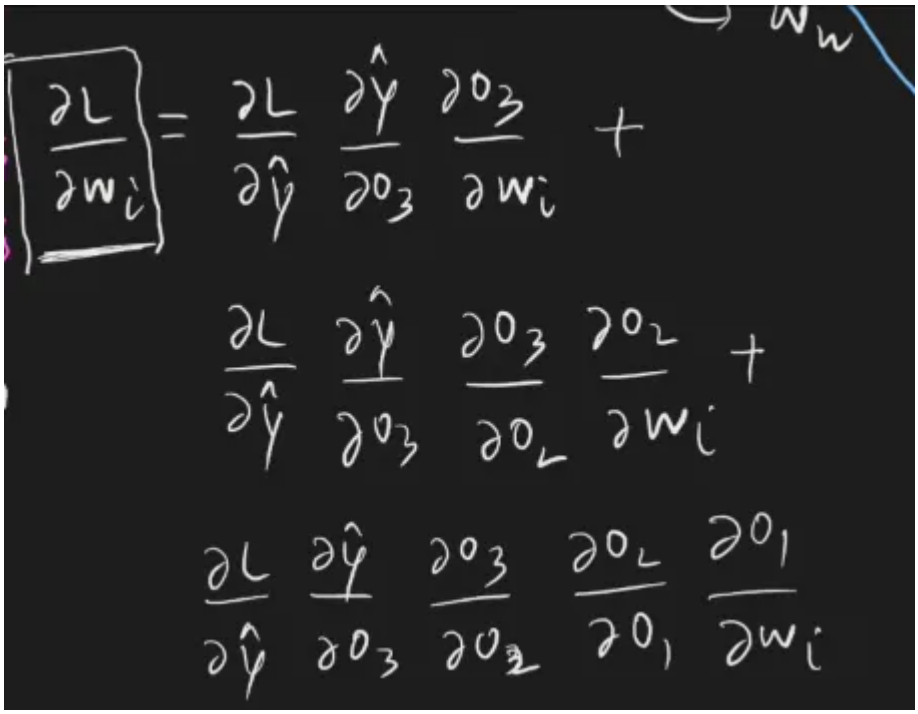
- Using the gradients, the weights in the actual RNN are adjusted in a way that will minimize the error in future predictions. Weights that significantly contributed to the error are adjusted more than those with minimal influence. Update of weights are calculated as shown in the figure :



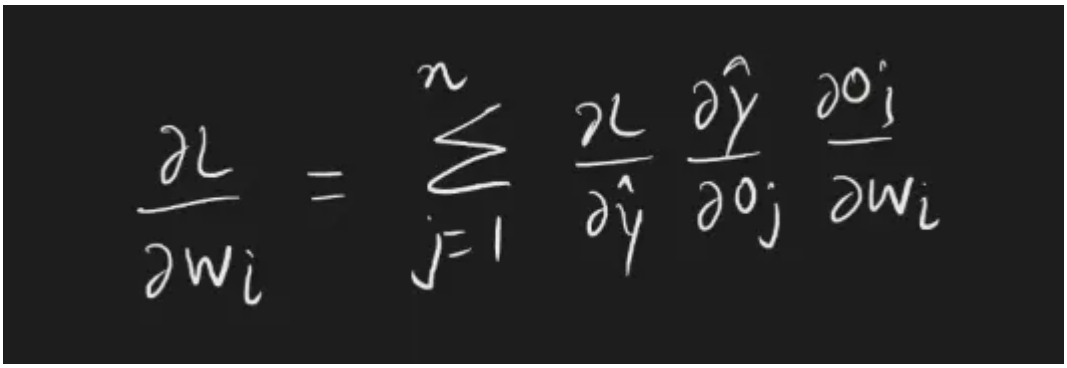
For calculating gradient, the following dependency needs to be understand :



And after understanding above dependency we can write the gradient w.r.t. to **wi** in the following way :



We can also write above equation in the following way also :



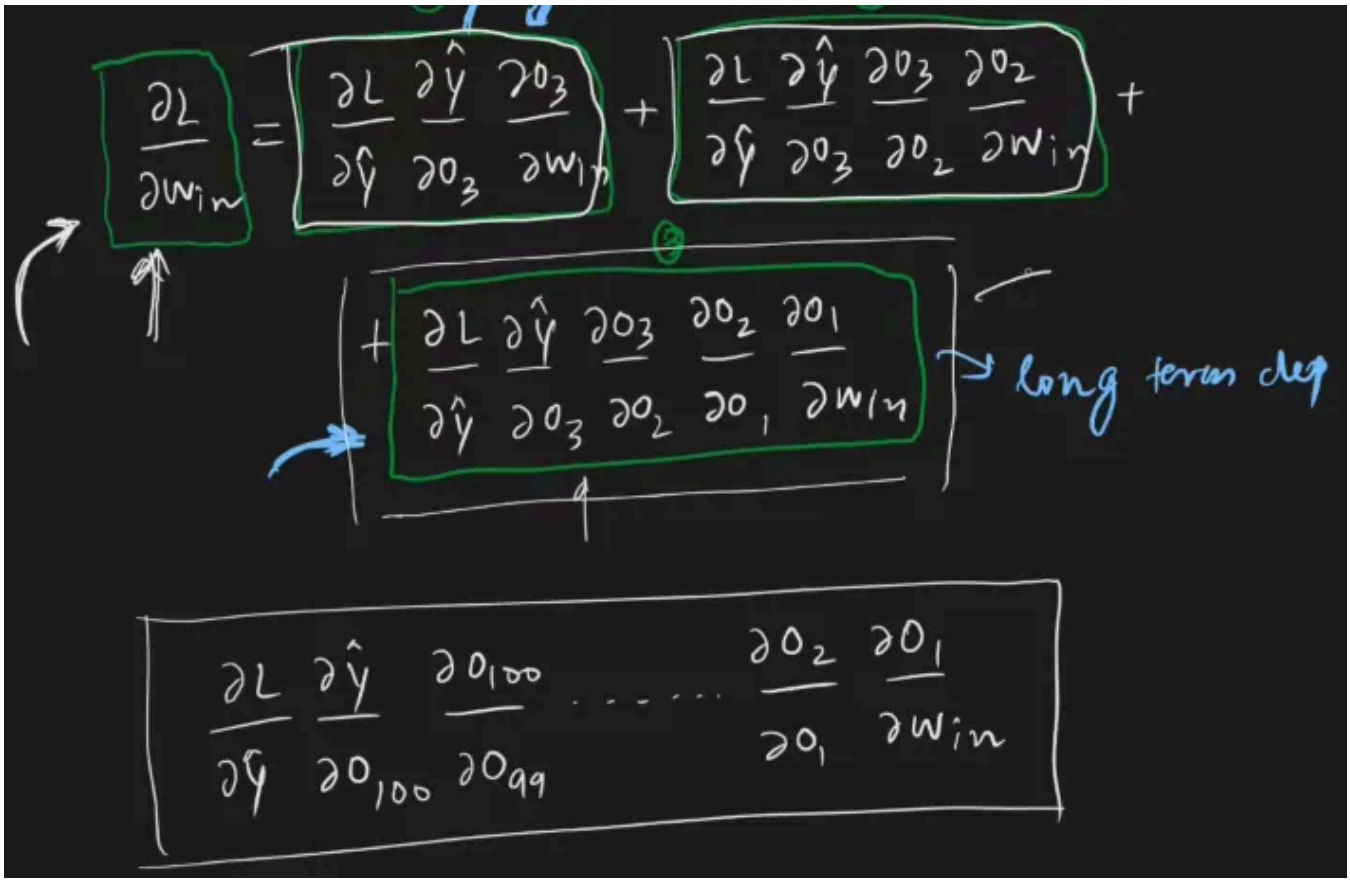
Problems with RNNs :

Vanishing Gradient Problem:

This is a major hurdle for RNNs, especially vanilla RNNs. It occurs when gradients (signals used to update weights during training) become very small or vanish as they propagate backward through the network during BPTT. This makes it difficult for the network to learn long-term dependencies in sequences, as information from earlier time steps can fade away.

In RNNs, information from previous time steps is used to influence the current output. This is achieved by feeding the output of a layer back as input to the next layer in the sequence. However, during back propagation through these layers:

- Gradients are multiplied by weights at each step.



- If these weights are all less than 1 in absolute value, this product becomes progressively smaller as we go back in time. This shrinks the error gradients for earlier time steps, making it difficult for the network to learn long-term dependencies present in the sequence.

RNNs struggle to learn from long sequences due to vanishing gradients. This makes them unsuitable for tasks like predicting future events based on long passages. However, RNNs excel at analyzing recent inputs, which is perfect for short-term predictions like suggesting the next word on a mobile keyboard.

I hope this blog has enhanced your fundamental knowledge of RNNs basics concept. If you've gained value from this content, consider following me for more insightful posts. Appreciate your time in reading this article. Thank you!

Follow

Written by Sachin Soni

846 followers · 21 following

Data Scientist at Tata Elxsi

Responses (6)

Write a response

What are your thoughts?

Bogpurple

Aug 1, 2024

that's a great explanation thanks!

5 Reply

Aman Panchal
Jan 28, 2025 (edited)

Very amazing explanation. **Impressive work !**

Thanks for sharing this knowledge.

1 Reply

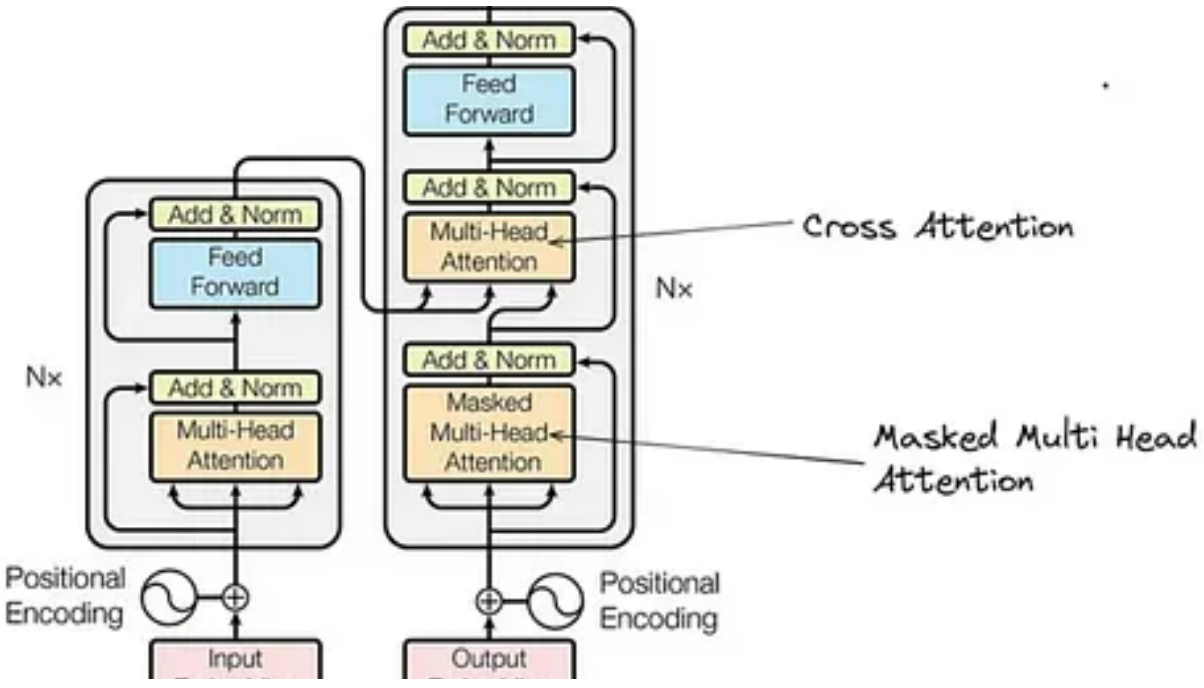
Xiaodi Fu
Sep 22, 2025

good work

Reply

See all responses

More from Sachin Soni

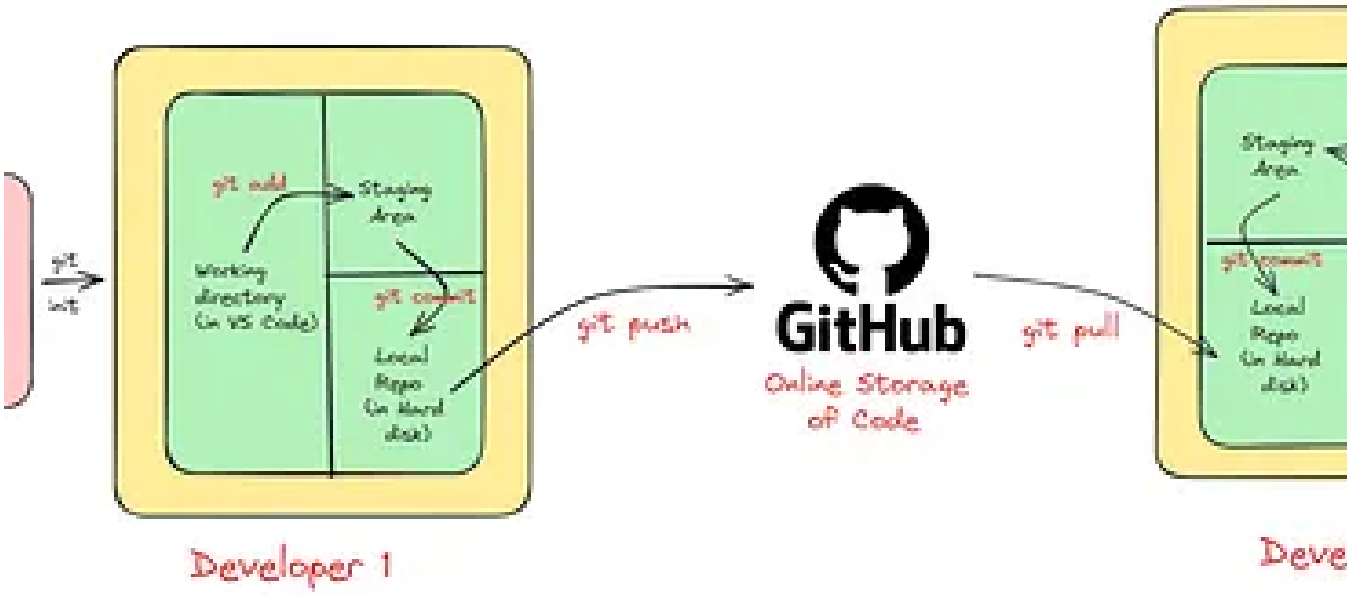


Sachin Soni

Cross Attention in Transformer

Cross attention is a key component in transformers, where a sequence can attend to another sequence's information, making it essential for...

Sep 6, 2024 184 3

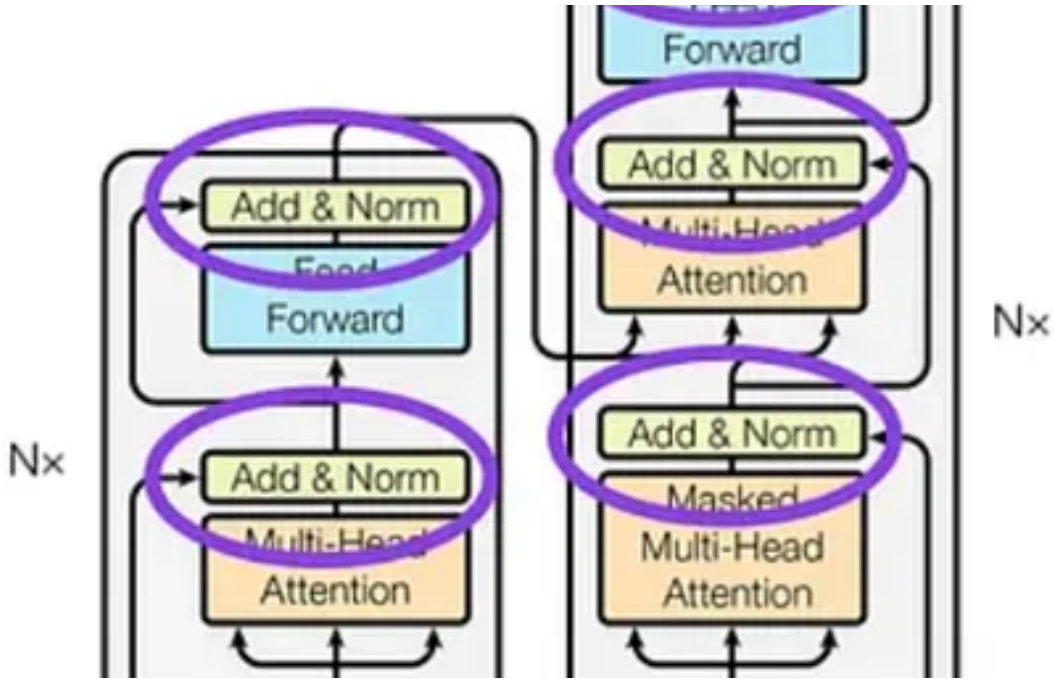


Sachin Soni

Complete Tutorial of Git and GitHub for Basic to Advanced

Have you ever wondered how teams of computer programmers keep their projects organized and working smoothly? It's all thanks to something...

Mar 16, 2024 115 6

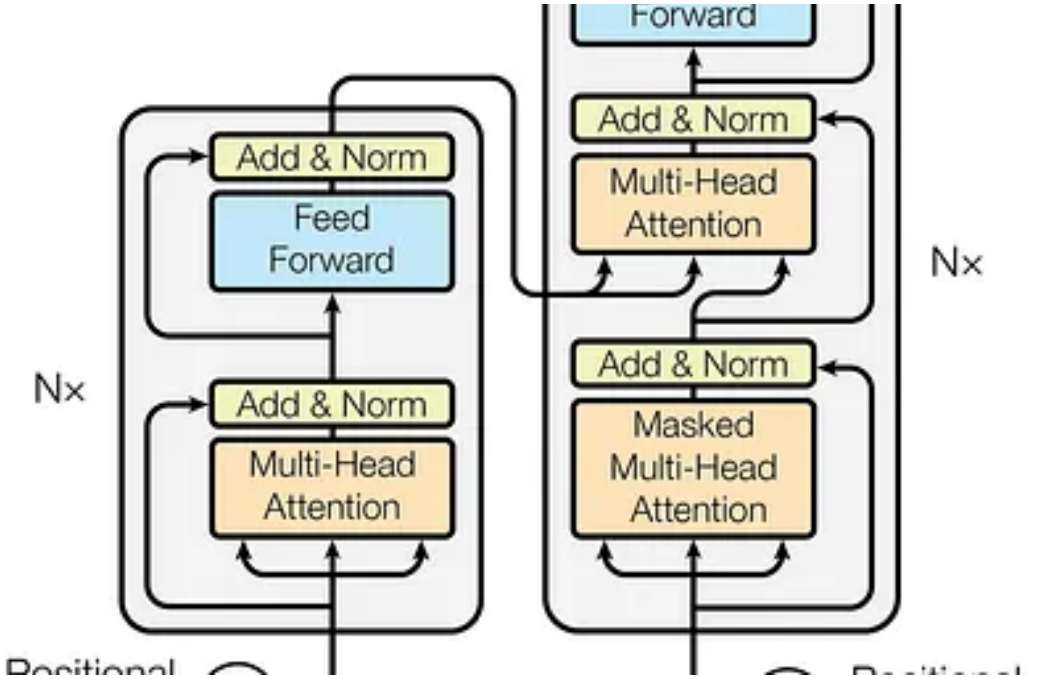


Sachin Soni

Layer Normalization in Transformer

Layer normalization is a crucial technique in transformer models that helps stabilize and accelerate training by normalizing the inputs to...

Sep 4, 2024 163 4



In Towards AI by Sachin Soni

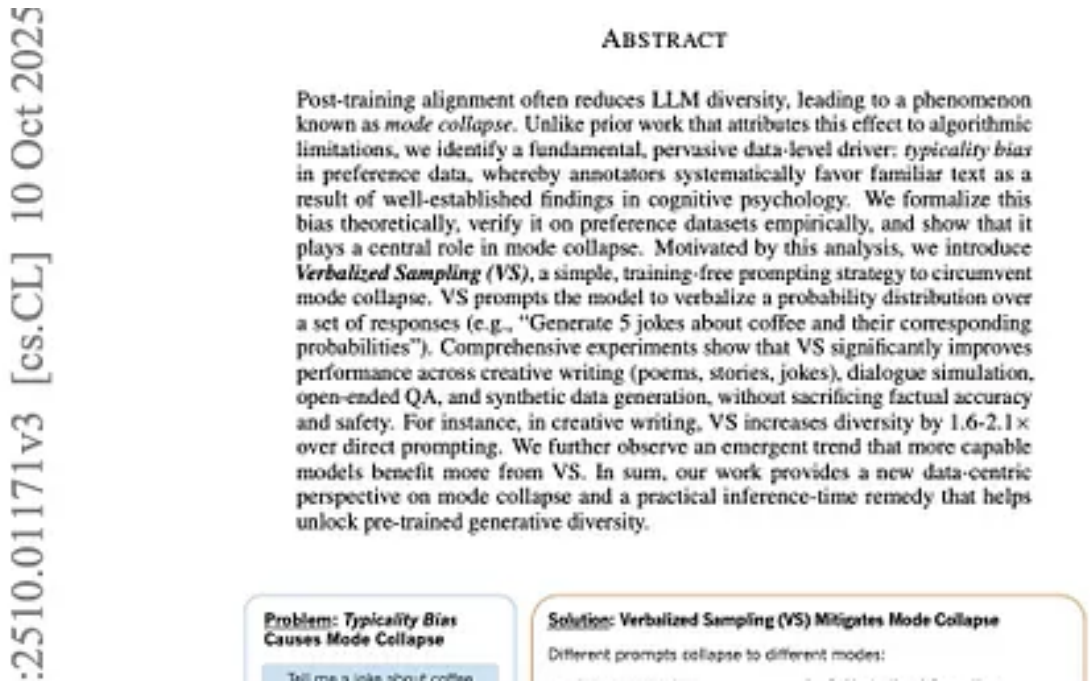
Transformer Architecture Part -1

In recent years, transformers have revolutionized the world of deep learning, powering everything from language models to vision tasks. If...

Sep 5, 2024 149

See all from Sachin Soni

Recommended from Medium

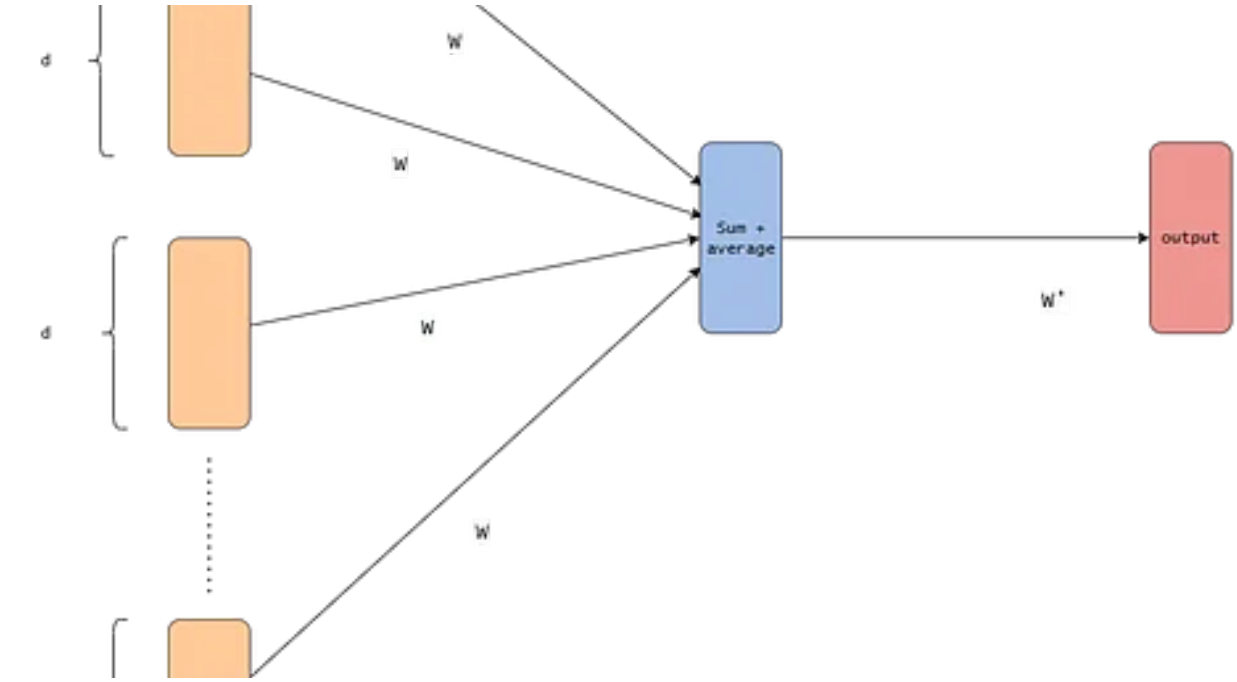


In Generative AI by Adham Khaled

Stanford Just Killed Prompt Engineering With 8 Words (And I Can't Believe It Worked)

ChatGPT keeps giving you the same boring response? This new technique unlocks 2× more creativity from ANY AI model—no training required...

Oct 20, 2025 21K 547

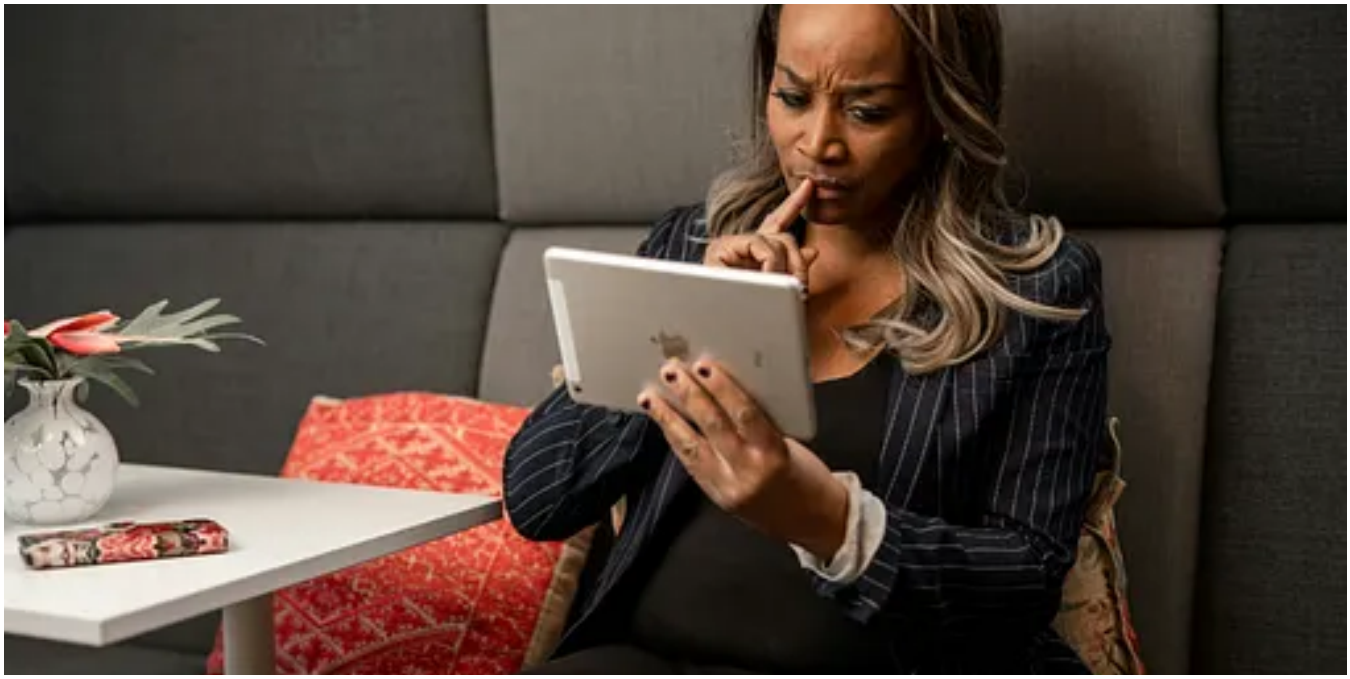


In Python in Plain English by Helene Kegel

Understanding Word2Vec—CBOW and Skip-Gram

In this article, we will take a look at the word embedding model called Word2Vec, which is a model that learns vector representations (i.e...

Aug 19, 2025



In Dev Genius by Rashawndoyley

LoRA and QLoRA: The Secret to Fine-Tuning LLMs Without Breaking the Bank (or Your GPU)

What is LoRA... or its way more appealing and slimmer sister QLoRA? LOL.

Dec 5, 2025 1

Will Lockett

The AI Bubble Is About To Burst, But The Next Bubble Is Already Growing

Techbros are preparing their latest bandwagon.

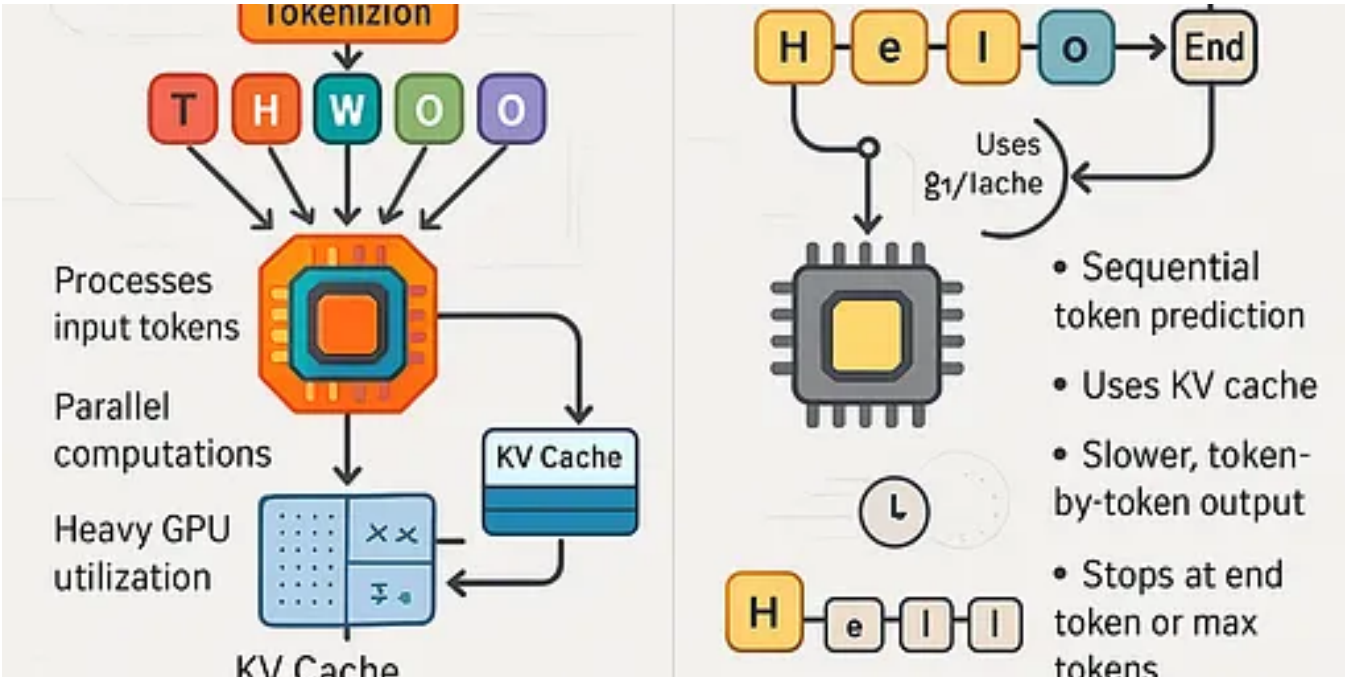
Sep 15, 2025 20K 811

In How To Profit AI® by Mohamed Bakry

Your ChatGPT History Just Went Public on Google. Here's What I Did in 10 Mins to Fix It.

Safety/Privacy Check Prompt Template Is Included

Dec 28, 2025 14.5K 463



Saiii

Understanding the Two Key Stages of LLM Inference: Prefill and Decode

Prefill Stage and Decode Stage in Inference

Jul 26, 2025 28