Connectionist Temporal Classification(CTC)

Abhishek Kumar Pandey · Follow
7 min read · May 26, 2024

Easy:

Alright, let's imagine you're watching a movie, but instead of just enjoying the story, you're also trying to predict what happens next. This isn't easy because movies can be long, and there are lots of twists and turns. Sometimes, you might guess right, and sometimes, you'll be surprised. But the cool thing is, you're getting better at it as you watch more movies!

Now, imagine if you could use a magic tool that helps you understand patterns in the movie — like how certain characters always end up together, or how music plays a big role in setting the mood. This magic tool would help you make even better guesses about what's going to happen next.

This magic tool is kind of like what computers do when they use something called "Connectionist Temporal Classification" (CTC). Computers don't watch movies, but they deal with lots of data over time, like listening to speech or analyzing patterns in music. They use CTC to understand the flow of information through time, similar to how you're learning to predict movie plots.

Here's the really cool part: CTC helps computers remember patterns from the past to make better predictions about the future. It's like training a super-smart friend who gets better at guessing what comes next the more it learns from the data it's analyzing.

So, in short, Connectionist Temporal Classification is a way for computers to understand and predict patterns in data over time, much like how you're getting better at predicting what happens next in movies

A movie

Moderate:

Connectionist Temporal Classification (CTC) is a technique used in machine learning, particularly in tasks involving sequences of data such as speech recognition, handwriting recognition, and time series prediction. It allows models to handle variable-length input sequences and makes predictions based on temporal dynamics within those sequences. Here's a breakdown of how it works:

Understanding Sequences

First, let's clarify what we mean by "sequences." In the context of CTC, sequences refer to ordered sets of data points. For example, a sentence spoken in a recording is a sequence of sounds, and a handwritten word is a sequence of pen strokes. These sequences can vary in length — some sentences may be longer than others, and different people write words in different numbers of strokes.

The Challenge

The challenge with sequences, especially variable-length ones, is aligning the input data with the model's architecture. Traditional neural network architectures expect fixed-size inputs, which doesn't work well with sequences because their lengths can differ significantly.

How CTC Solves This Problem

CTC introduces a way to train neural networks on sequences without needing to know the alignment between the input sequence and the desired output sequence. Here's how it does that:

1. **Softmax Output Layer:** At the end of the network, instead of having a single output node per time step, CTC uses a softmax layer that outputs probabilities for all possible tokens (e.g., letters, sounds) at every time step simultaneously. This means the network predicts what could possibly come next at any given moment, considering all potential outputs.
2. **Alignment Path:** To train the network, CTC defines an "alignment path" that maps each output token to one or more input time steps. This path is not known beforehand; instead, it's optimized during training to find the best match between the predicted and actual output sequences. The goal is to minimize the difference between what the network predicts and what actually occurs.
3. **Loss Function:** The loss function used in CTC measures how well the predicted alignment paths match the target sequences. It penalizes the model for incorrect alignments, encouraging it to improve its predictions over time.
4. **Blanks:** A unique "blank" token is introduced into the output space. This token represents the absence of an output at a particular time step, allowing the model to indicate gaps in the prediction where no output is needed. This is crucial for handling variable-length sequences, as it enables the model to account for differences in the length of input and output sequences.

Practical Applications

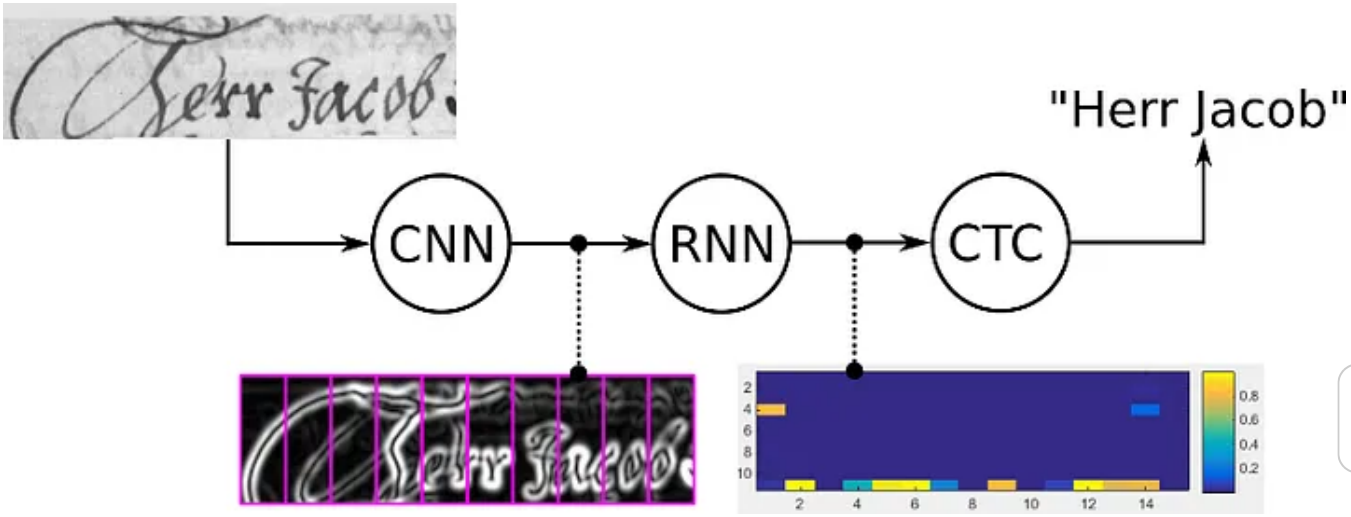
CTC has been highly successful in applications where understanding the timing and order of events is critical, such as:

- **Speech Recognition:** Transcribing spoken language into written text accurately requires recognizing the sequence of sounds and mapping them to the

corresponding letters.

- **Handwriting Recognition:** Recognizing handwritten text involves interpreting varying stroke orders and durations to identify the intended characters.
- **Time Series Prediction:** Predicting future values in time series data (like stock prices or weather patterns) benefits from modeling the temporal dependencies between observations.

In summary, Connectionist Temporal Classification provides a powerful framework for dealing with sequential data, enabling machines to understand and predict the flow of information over time.



Hard:

Connectionist Temporal Classification (CTC) is an algorithm used to train recurrent neural networks (RNNs) for sequence-to-sequence mapping problems, especially where the alignment between input and output sequences is unknown. It's particularly useful in automatic speech recognition (ASR) and handwriting recognition.

Key Concepts of CTC:

1. **Sequence Alignment Problem:**
   Traditional methods assume that each input step directly corresponds to an output step, which is not the case in many real-world applications.

For example, in speech recognition, the number of spoken frames and the number of letters in the transcribed text are different and not directly aligned.

2. **Blank Tokens:**
   CTC introduces a special "blank" token that allows the model to predict that no output character should be emitted at a particular time step.
   This helps the model to handle varying lengths of input and output sequences.

3. **Many-to-One Mapping:**
   The CTC allows many input frames to map to a single output character.
   For instance, multiple frames of the sound "llll" in "hello" can all map to a single "l" in the final output.

How CTC Works:

1. **Input Sequence:**
   The model takes an input sequence (e.g., frames of audio) and processes it to produce a probability distribution over possible output labels (including the blank token) for each frame.

2. **Output Probabilities:**
   For each frame, the model predicts the probability of each possible output character and the blank token.

3. **CTC Path:**
   A path through the probability distributions is generated, where each path is a potential alignment of the input frames to the output sequence.
   These paths can include repeated characters and blank tokens to align varying input lengths to the output sequence.

4. **Loss Calculation:**
   The CTC loss function sums over all valid paths that correspond to the correct output sequence.
   It calculates the probability of the correct transcription by summing the probabilities of all possible alignments that could lead to the correct output.

The Problem CTC Solves:

Imagine you have an audio recording of someone saying "hello" and you want your RNN to transcribe it. The problem is that the audio signal is much longer than the

text “hello”. There’s no clear one-to-one mapping between individual audio frames and letters.

- Traditional approaches would require manually segmenting the audio into phonemes or characters, which is tedious and error-prone.
- CTC solves this by allowing the RNN to output a sequence of labels that can be *decoded* into the final transcription.

How CTC Works:

1. **Blank Label:** CTC introduces a special “blank” label (often denoted as “-”) representing “no output” at a given timestep.
2. **Possible Alignments:** The RNN outputs a probability distribution over all possible labels (including the blank) at each timestep. This creates a matrix of probabilities representing all possible alignments between the input sequence and the output labels.
3. **Decoding the Output:** CTC uses a decoding algorithm to transform the matrix of probabilities into the most likely output sequence. This involves:
   Merging consecutive repeated labels (e.g., “hhheeelllooo” becomes “hello”).
   Removing blank labels.

Example:

Let’s say the input audio has 5 timesteps, and the possible labels are “h”, “e”, “l”, “o”, and “-”. The RNN might output the following probability matrix:

| Timestep | h | e | l | o | — |

| — — — — — | — — -| — — -| — — -| — — -| — — -|

| 1 | 0.2 | 0.1 | 0.1 | 0.1 | 0.5 |

| 2 | 0.7 | 0.1 | 0.1 | 0.0 | 0.1 |

| 3 | 0.1 | 0.8 | 0.0 | 0.0 | 0.1 |

| 4 | 0.1 | 0.1 | 0.7 | 0.0 | 0.1 |

| 5 | 0.1 | 0.0 | 0.1 | 0.7 | 0.1 |

The decoding algorithm would then identify the most likely path through this matrix, considering the rules of merging repetitions and removing blanks. In this case, the most likely output would be “hello”.

Advantages of CTC:

- **No need for pre-segmented data:** CTC learns the alignment between input and output sequences automatically.
- **Handles variable-length sequences:** Both input and output sequences can have different lengths.
- **End-to-end trainable:** The entire system (RNN + CTC) can be trained jointly, optimizing for the final transcription accuracy.

Disadvantages of CTC:

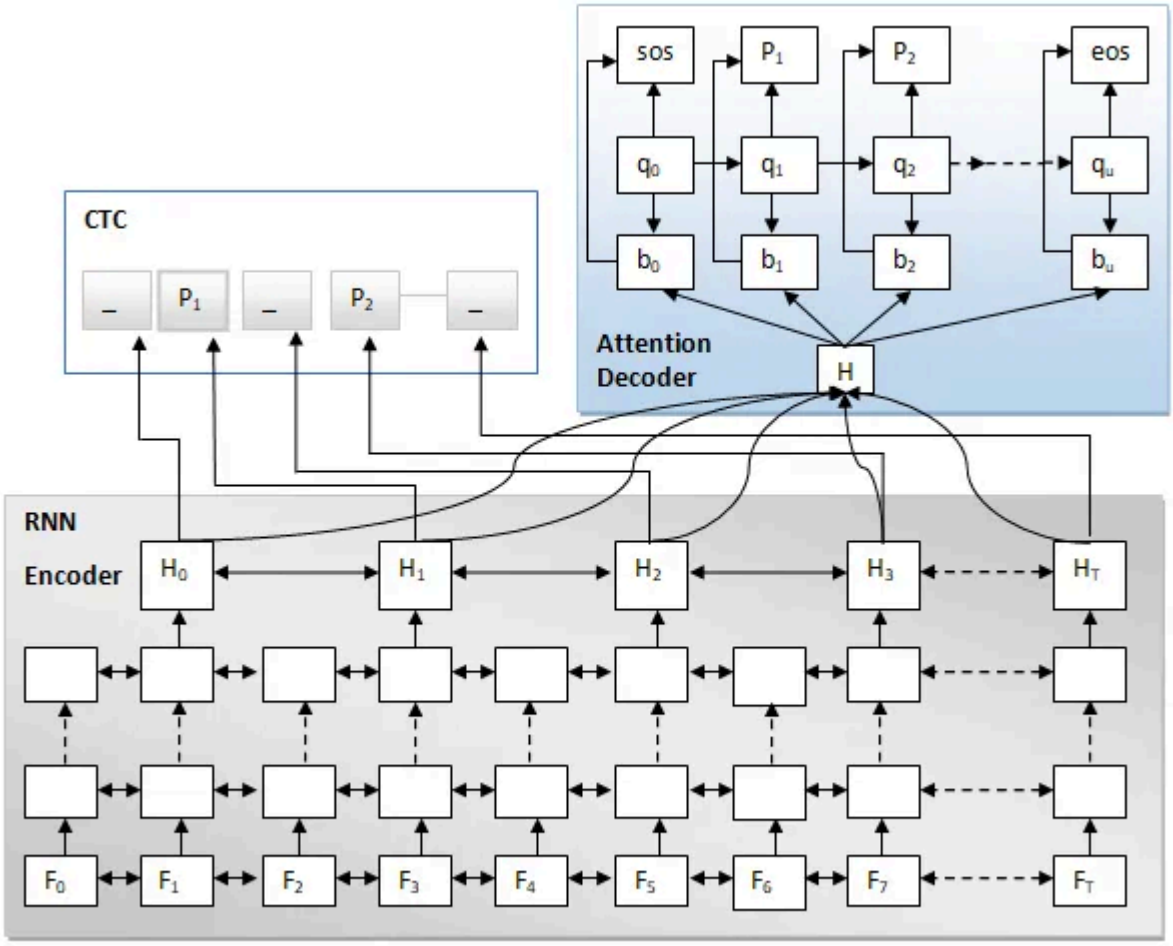
- **Decoding complexity:** The decoding process can be computationally expensive for very long sequences.
- **Conditional independence assumption:** CTC assumes that the output at each timestep is conditionally independent of previous outputs, which might not always hold.

Applications:

- **Automatic Speech Recognition:** Transcribing speech audio into text.
- **Handwriting Recognition:** Converting handwritten text images into digital text.
- **Keyword Spotting:** Detecting specific keywords within an audio stream.

In Conclusion:

Connectionist Temporal Classification is a powerful technique for training RNNs on sequence-to-sequence problems where the alignment between input and output is unknown. It has revolutionized areas like automatic speech recognition by simplifying the training process and achieving impressive accuracy.



If you want you can support me: https://buymeacoffee.com/abhi83540

If you want such articles in your email inbox you can subscribe to my newsletter: https://abhishekkumarpandey.substack.com/

A few books on deep learning that I am reading:

Book 1

Book 2

Book 3

Deep Learning | Machine Learning | Data Science | NLP | Computer Vision

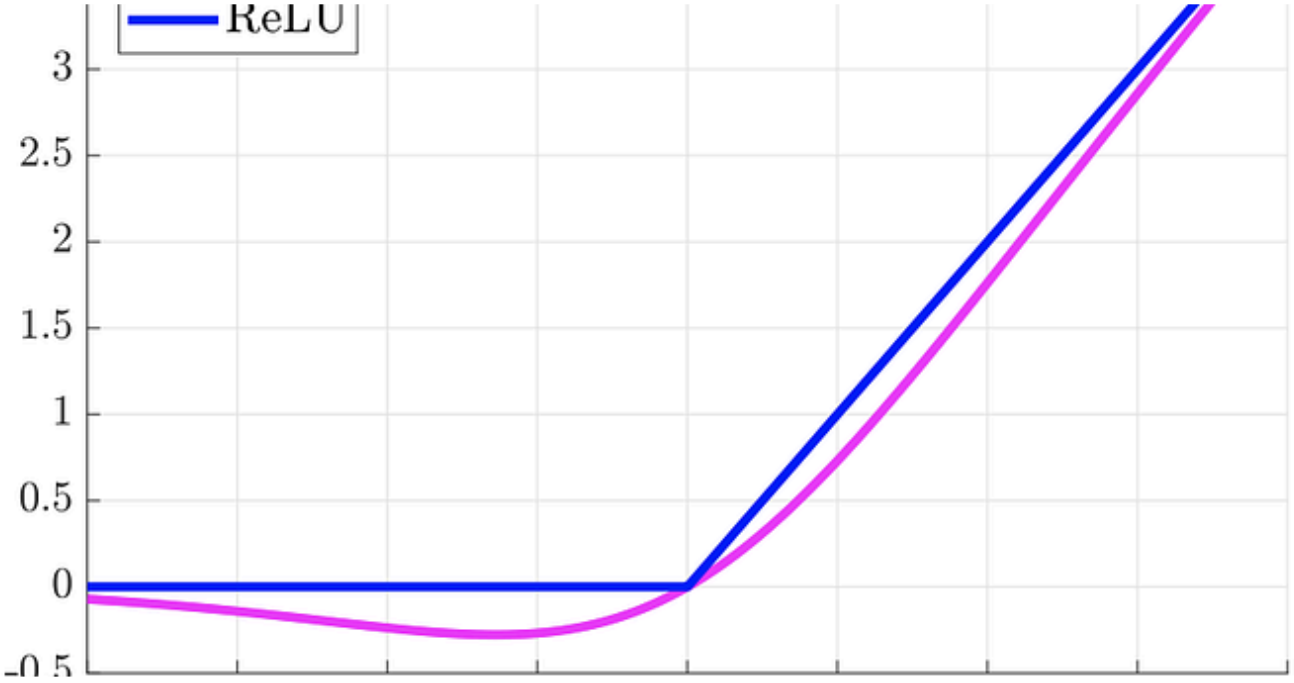
Follow

Written by Abhishek Kumar Pandey

12 Followers

I am a deep learning researcher. I write about technology. https://www.buymeacoffee.com/abhi83540

More from Abhishek Kumar Pandey



Abhishek Kumar Pandey

SiLU (Sigmoid Linear Unit) activation function

Easy:

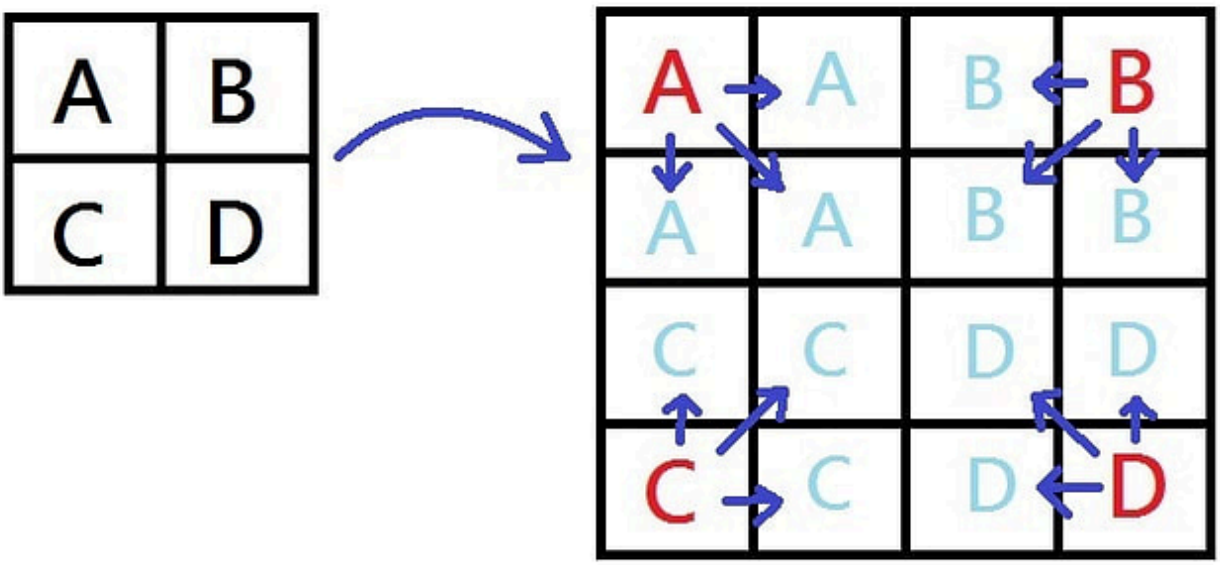
Apr 6 7

Abhishek Kumar Pandey

Adaptive Average Pooling Layer

Easy

Apr 13 3

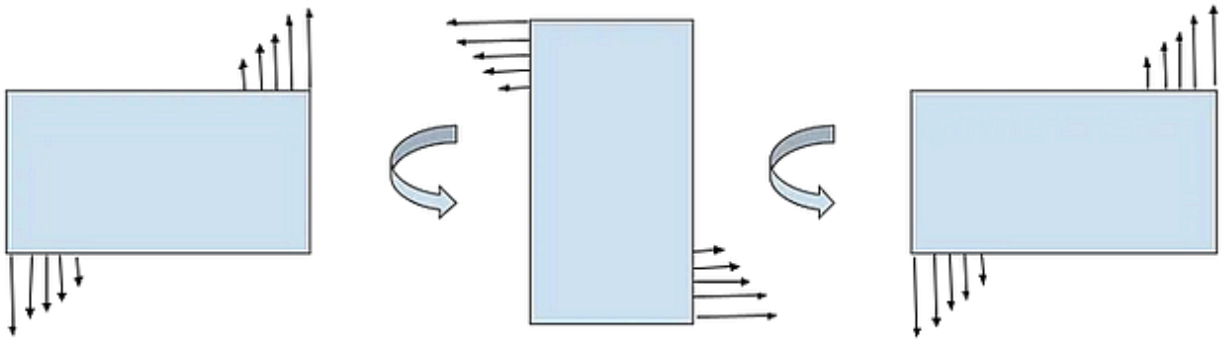


Abhishek Kumar Pandey

Nearest Neighbor Interpolation

Easy:

Mar 28 4



Created by Abhishek using Google Docs

Farneback algorithm

Abhishek Kumar Pandey in Python's Gurus

Farneback Algorithm

Easy:

Mar 1 108

See all from Abhishek Kumar Pandey

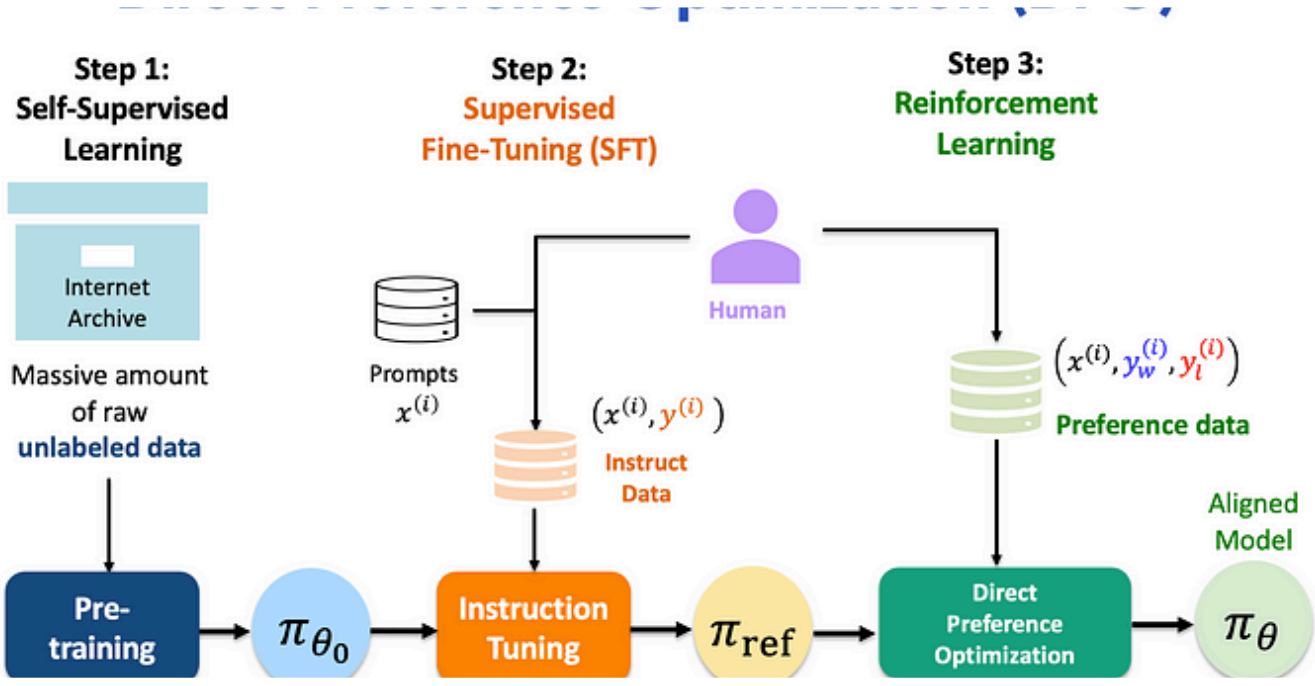
Recommended from Medium

Vincent Lee

Journey to Mamba🐍— Part 1: State space models

Introduction

Mar 20 6



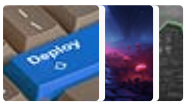
LM Po

Direct Preference Optimization (DPO) of LLMs: A Paradigm Shift

Large language models (LLMs) have made tremendous progress in recent years, but aligning them with human preferences remains a critical...

Aug 24 1

Lists



Predictive Modeling w/ Python

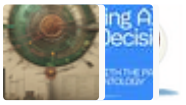
20 stories · 1490 saves

Practical Guides to Machine Learning

10 stories · 1819 saves

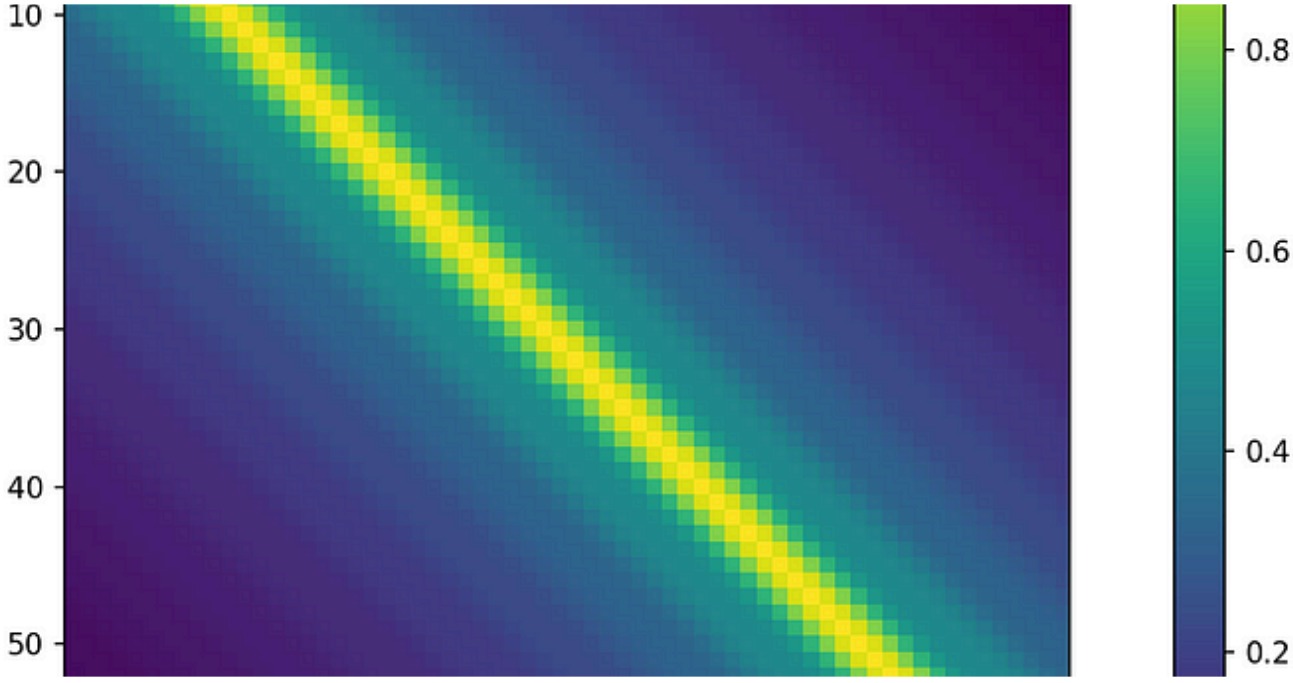
Natural Language Processing

1679 stories · 1252 saves



data science and AI

40 stories · 234 saves



Pranay Janupalli

Understanding Sinusoidal Positional Encoding in Transformers

In NLP, transformer architecture has emerged as a powerful architecture for handling sequential data. However, unlike recurrent neural...

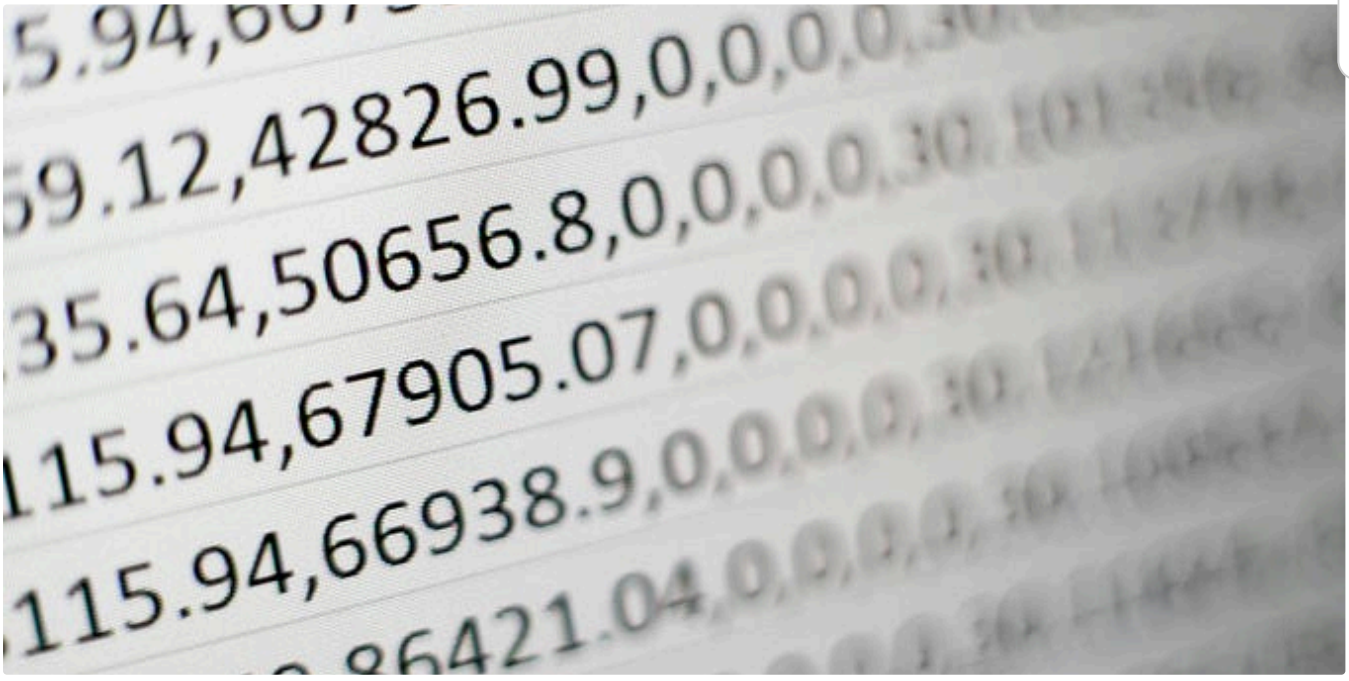
Apr 14 5

Isaac Berrios

Introduction to Beamforming: Part 3

How to derive and implement the Capon Beamformer

Aug 26

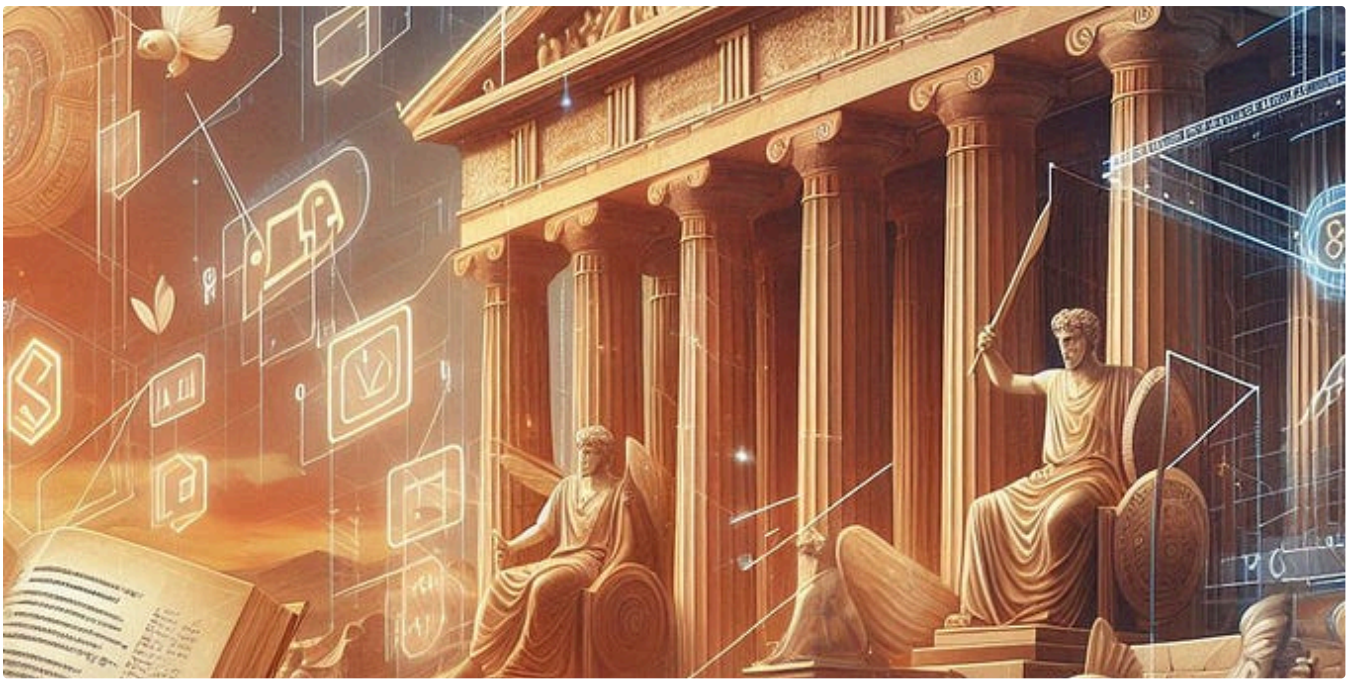


Deepanshu Bagotia

How to generate node-embeddings using encoder-decoder in Graphs

This is the third blog of the Graph ML Blogs series. If you have already followed here or your prior knowledge only asks for this, let's...

Apr 11 30



George Sarmonikas

The Importance of Tokenizers for Multilingual LLMs: A Case Study on Greek

Large language models (LLMs) have shown remarkable performance on a wide range of natural language processing tasks. However, most LLMs are...

Jun 12

See more recommendations