



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



The Time Series Transformer

All you need to know about the state of the art Transformer Neural Network Architecture, adapted to Time Series Tasks. Keras code included.



Theodoros Ntakouris · Follow

Published in Towards Data Science · 6 min read · Jan 26, 2021



1.2K



13

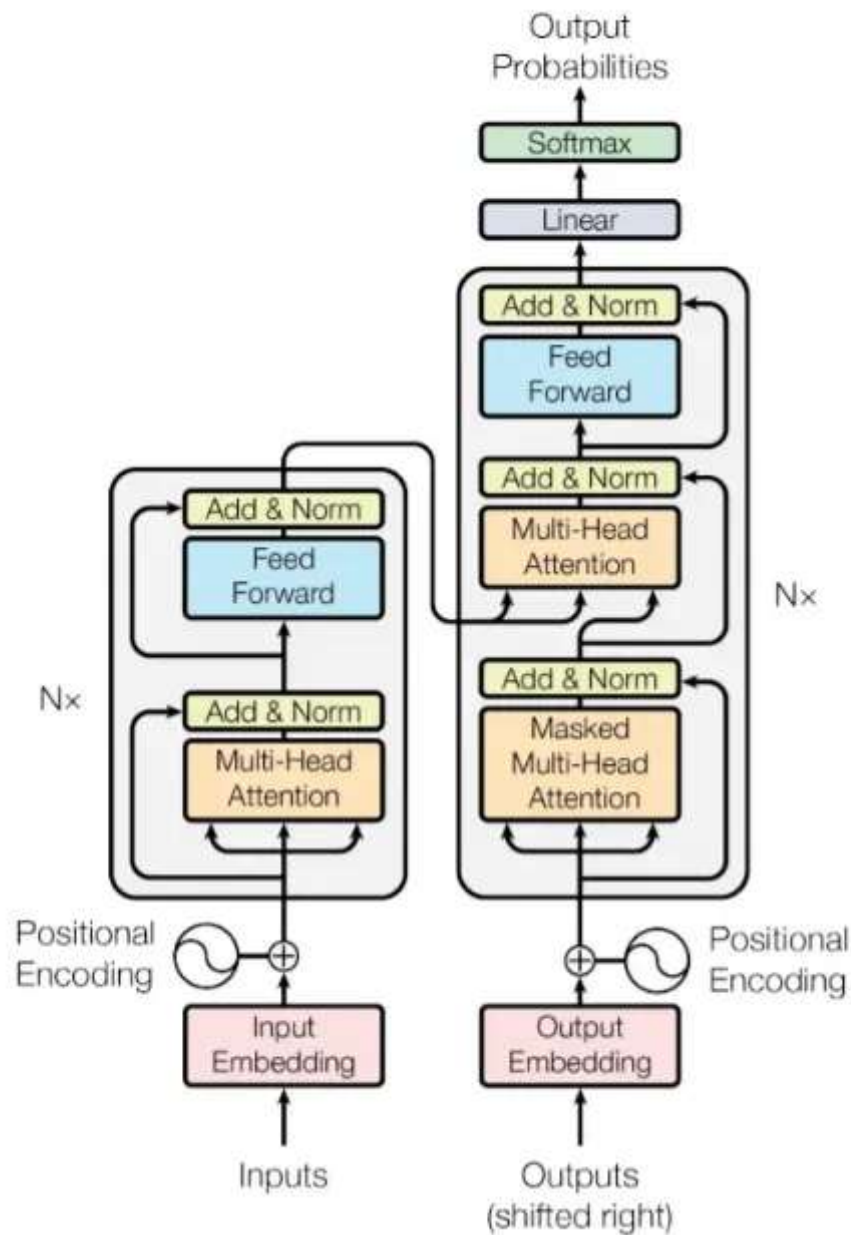


Table of Contents

- Introduction
- Preprocessing
- Learnable Time Representation (Time 2 Vec)
- Architecture
- Bag Of Tricks (things to consider when training Transformers)

Introduction

Attention Is All You Need they said. Is it a more robust convolution? Is it just a hack to squeeze more learning capacity out of fewer parameters? Is it supposed to be sparse? How did the original authors come up with this architecture?



The Transformer Architecture

- It's better than RNNs because it's not recurrent and can use previous time step features without a loss in detail
- It's the top performer architecture on plethora of tasks, including but not limited to: NLP, Vision, Regression (it scales)

It is pretty easy to switch from an existing RNN model to the Attention architecture. Inputs are of the same shape!

Preprocessing

Using Transformers for Time Series Tasks is different than using them for NLP or Computer Vision. We neither tokenize data, nor cut them into 16x16 image chunks. Instead, we follow a more classic / old school way of preparing data for training.

One thing that is definitely true is that we have to feed data in the same value range as input, to eliminate bias. This is typically on the $[0, 1]$ or $[-1, 1]$ range. In general, it is recommended to apply the same kind of preprocessing pipeline on all of your input features to eliminate this bias. Individual use cases may be exempt from this, different models and data are unique! Think about the origin of your data for a moment.

Popular time series preprocessing techniques include:

- Just scaling to $[0, 1]$ or $[-1, 1]$
- Standard Scaling (removing mean, dividing by standard deviation)
- Power Transforming (using a power function to push the data to a more normal distribution, typically used on skewed data / where outliers are present)
- Outlier Removal
- Pairwise Differencing or Calculating Percentage Differences
- Seasonal Decomposition (trying to make the time series stationary)

- Engineering More Features (automated feature extractors, bucketing to percentiles, etc)
- Resampling in the time dimension
- Resampling in a feature dimension (instead of using the time interval, use a predicate on a feature to re-arrange your time steps — for example when recorded quantity exceeds N units)
- Rolling Values
- Aggregations
- Combinations of these techniques

Again, preprocessing decisions are tightly coupled to the problem and data at hand, but this is a nice list to get your started.

If your time series can become stationary by doing preprocessing such as seasonal decomposition, you could get good quality predictions by using smaller models (that also get trained way faster and require less code and effort), such as NeuralProphet or Tensorflow Probability.

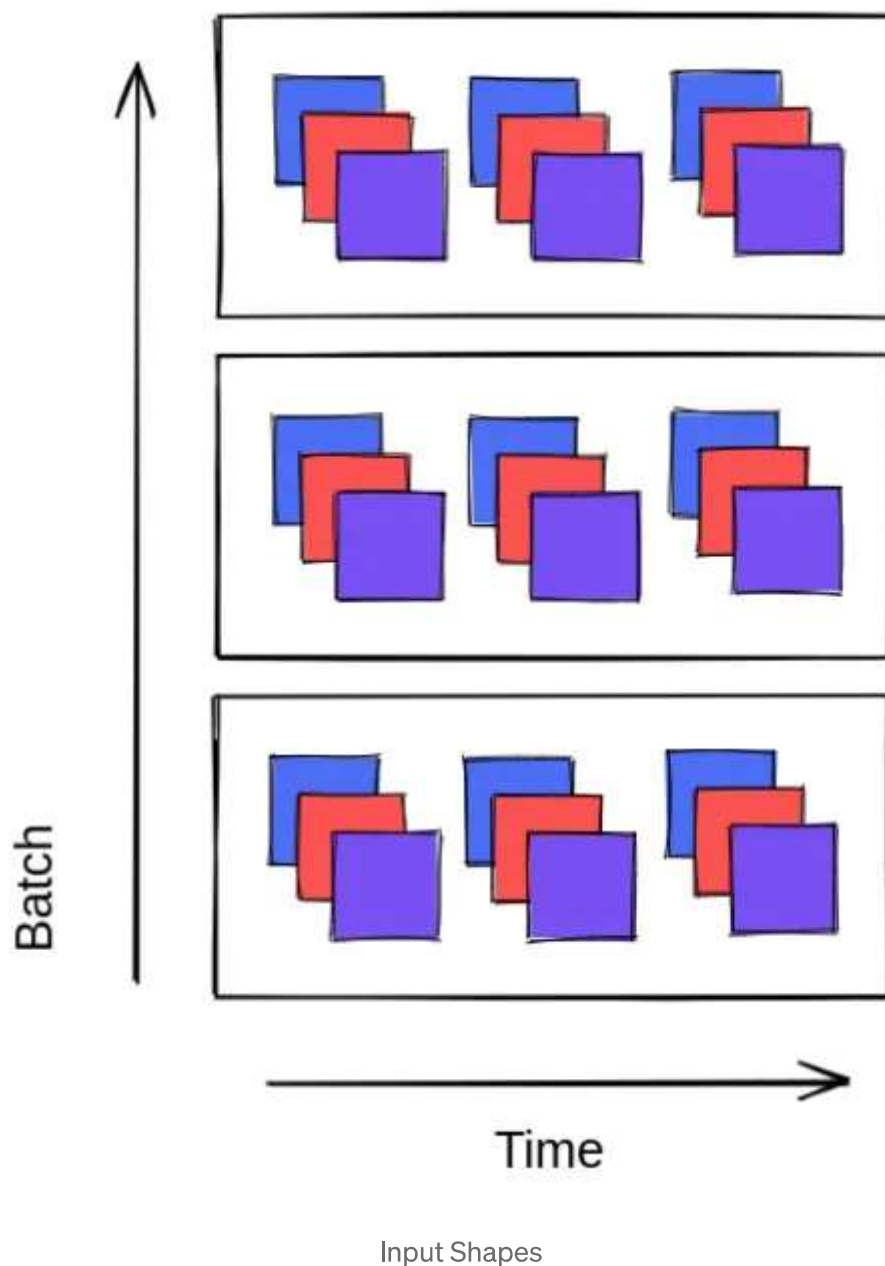
Deep Neural Networks can learn linear and periodic components on their own, during training (we will use Time 2 Vec later). That said, I would advise against seasonal decomposition as a preprocessing step.

Other decisions such as calculating aggregates and pairwise differences, depend on the nature of your data, and what you want to predict.

Treating sequence length as a hyper parameter, this leads us to an input tensor shape that is similar to RNNs: (batch size, sequence length,

features) .

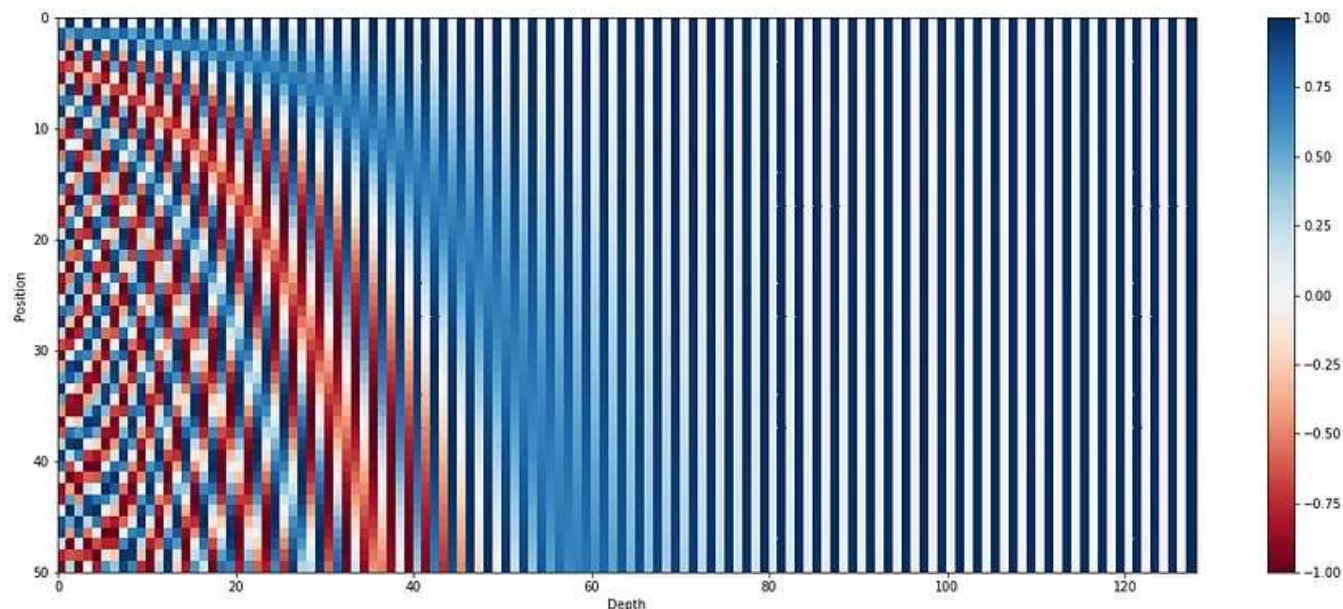
Here is a drawing for all the dimensions set to 3.



Learnable Time Representation

For Attention to work, you need to attach the meaning of time to your input features. In the original NLP model, a collection of superimposed sinusoidal

functions were added to each input embedding. We need a different representation now that our inputs are scalar values and not distinct words/tokens.



Positional Encoding Visualization from [kazemnejad's blog](#).

The Time 2 Vec paper comes in handy. It's a learnable and complementary, model-agnostic representation of time. If you've studied Fourier Transforms in the past, this should be easy to understand.

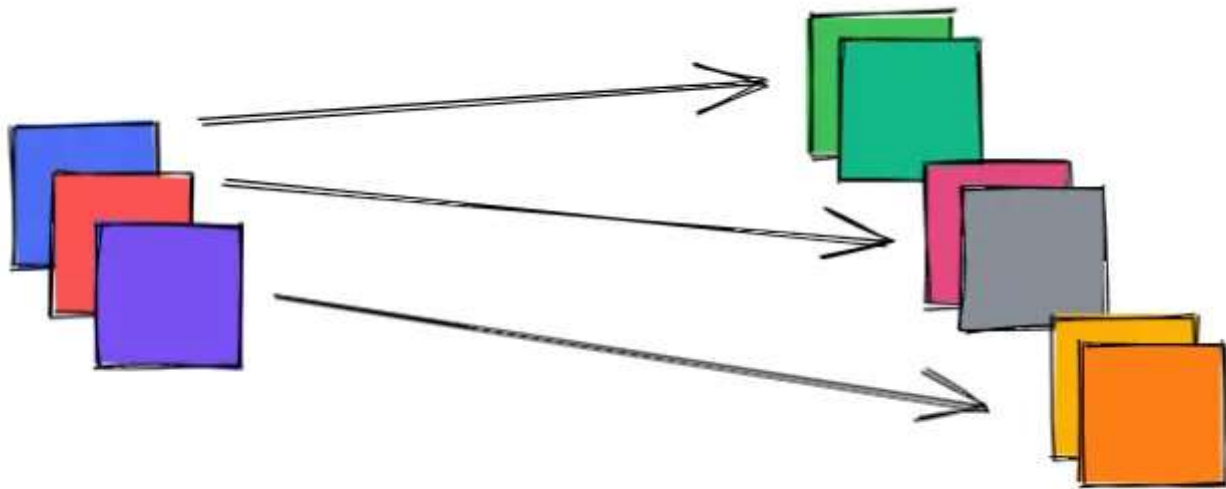
Just break down each input feature to a linear component (a line) and as many periodic (sinusoidal) components you wish. By defining the decomposition as a function, we can make the weights learnable through back propagation.

$$\mathbf{t2v}(\tau)[i] = \begin{cases} \omega_i \tau + \varphi_i, & \text{if } i = 0. \\ \mathcal{F}(\omega_i \tau + \varphi_i), & \text{if } 1 \leq i \leq k. \end{cases}$$

For each input feature, we apply the same layer in a time-independent (time-distributed layer) manner. This learnable embedding does not depend on time! Finally, concatenate the original inputs.

Here is a an illustration of the learned time embeddings, which are **different**, for each input feature category (1 learned linear component and 1 learned periodic component per feature).

This does not mean that each time — step will carry the same embedding value, because the computation of the time2vec embeddings depend on the input values!



And, in the end, we concatenate these all together to form the input for the attention blocks.


```

1  class Time2Vec(keras.layers.Layer):
2      def __init__(self, kernel_size=1):
3          super(Time2Vec, self).__init__(trainable=True, name='Time2VecLayer')
4          self.k = kernel_size
5
6      def build(self, input_shape):
7          # trend
8          self.wb = self.add_weight(name='wb', shape=(input_shape[1],), initializer='uniform', trainable=True)
9          self.bb = self.add_weight(name='bb', shape=(input_shape[1],), initializer='uniform', trainable=True)
10         # periodic
11         self.wa = self.add_weight(name='wa', shape=(1, input_shape[1], self.k), initializer='uniform', trainable=True)
12         self.ba = self.add_weight(name='ba', shape=(1, input_shape[1], self.k), initializer='uniform', trainable=True)
13         super(Time2Vec, self).build(input_shape)
14
15     def call(self, inputs, **kwargs):
16         bias = self.wb * inputs + self.bb
17         dp = K.dot(inputs, self.wa) + self.ba
18         wgts = K.sin(dp) # or K.cos(.)
19
20         ret = K.concatenate([K.expand_dims(bias, -1), wgts], -1)
21         ret = K.reshape(ret, (-1, inputs.shape[1]*(self.k+1)))
22         return ret
23
24     def compute_output_shape(self, input_shape):
25         return (input_shape[0], input_shape[1]*(self.k + 1))

```

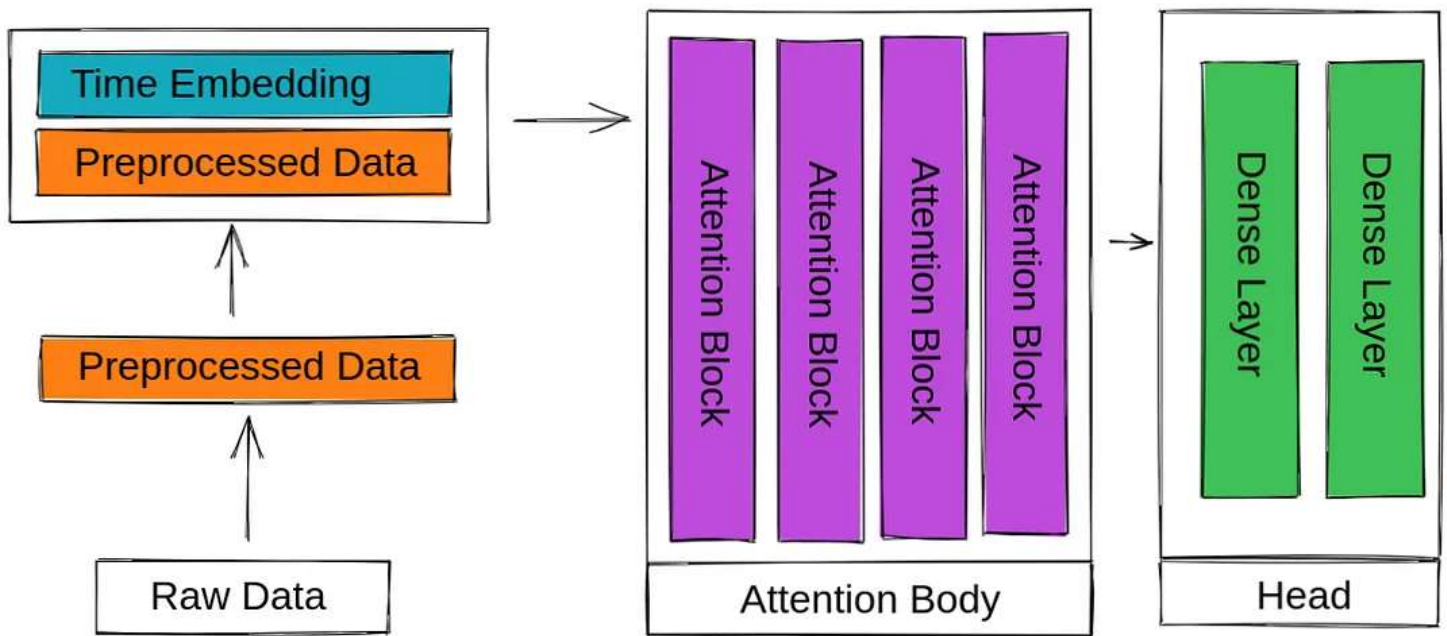
Time2Vec.py hosted with ❤ by GitHub

[view raw](#)

Architecture

We are going to use Multi-Head Self-Attention (setting Q, K and V to depend on the input through different dense layers/matrices). The next part is optional and depends on the scale of your model and data, but we are also going to ditch the decoder part completely. This means, that we are only going to use one or more attention block layers.

In the last part, we are going to use a few (one or more) Dense layers to predict whatever we want to predict.



Our Architecture

Each Attention Block consists of Self Attention, Layer Normalizations and a Feed — Forward Block. The input dimensions of each block are equal to it's output dimensions.

Optionally, before the head part, you can apply some sort of pooling (Global Average 1D for example).

Bag Of Tricks

Things to consider when using Transformers and Attention, to get the most out of your model.

- **Start Small**

Don't go crazy with hyperparameters. Start with a single, humble attention layer, a couple of heads and a low dimension. Observe results and adjust hyper parameters accordingly — don't overfit! Scale your model along with your data. Nevertheless, nothing is stopping you from scheduling a huge hyperparameter search job :).

- **Learning Rate Warmup**

A crucial part of the attention mechanism that leads to greater stability is learning-rate warmup. Start with a small learnign rate and gradually increase it till you reach the base one, then decrease again. You can go crazy with exponential — decaying schedules and sophisticated formulas, but I will just give you a simple example that you should be able to understand just by reading the following code out loud:

- **Use Adam (or variants)**

Non-accelerated gradient descent optimization methods do not work well with Transformers. Adam is a good initial optimizer choice to train with. Keep an eye out for newer (and possibly better) optimization techniques like AdamW or NovoGrad!

Thanks for reading all the way to the end!

References

[1] Attention Is All You Need, Ashish Vaswani and Noam Shazeer and Niki Parmar and Jakob Uszkoreit and Llion Jones and Aidan N. Gomez and Lukasz Kaiser and Illia Polosukhin, 2017

[2] Time2Vec: Learning a Vector Representation of Time, Seyed Mehran Kazemi and Rishab Goel and Sepehr Eghbali and Janahan Ramanan and Jaspreet Sahota and Sanjay Thakur and Stella Wu and Cathal Smyth and Pascal Poupart and Marcus Brubaker, 2019

Deep Learning

Machine Learning

Data Science

Time Series Forecasting

Artificial Intelligence



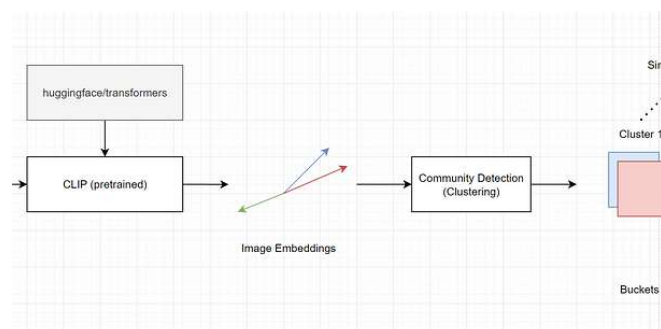
Written by Theodoros Ntakouris

476 Followers · Writer for Towards Data Science

Follow

<https://ntakour.is/>

More from Theodoros Ntakouris and Towards Data Science



 Theodoros Ntakouris

Image deduplication using OpenAI's CLIP and Community...

A short guide on how to use image embeddings from OpenAI's CLIP and...

3 min read · Apr 9, 2023

 4 

 ...



 Sheila Teo in Towards Data Science

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 28, 2023

 9.93K  118

 ...



 Thu Vu in Towards Data Science

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



 Theodoros Ntakouris in Towards Data Science

Advanced Tensorflow Data Input Pipelines: Handling Time Series

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

🌟 · 12 min read · Jan 5



2.2K



24



133



Extracting labels, windowing multivariate
series, multiple TF Record file shards and...


4 min read · Sep 25, 2020

See all from Theodoros Ntakouris

See all from Towards Data Science

Recommended from Medium



 Mouna Labiadh

Forecasting book sales with Temporal Fusion Transformer

11 min read · Aug 2, 2023

 96  2



 Fareed Khan in Level Up Coding

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

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

13 min read · Dec 18, 2023

 1.5K  24

Lists



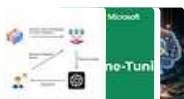
Predictive Modeling w/ Python

20 stories · 825 saves



Practical Guides to Machine Learning

10 stories · 953 saves



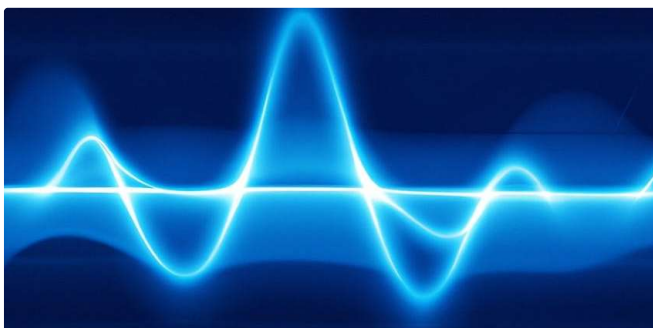
Natural Language Processing

1118 stories · 589 saves



data science and AI

39 stories · 49 saves





Nikos Kafritsas in Towards Data Science

Temporal Fusion Transformer: Time Series Forecasting with Dee...

Create accurate & interpretable predictions

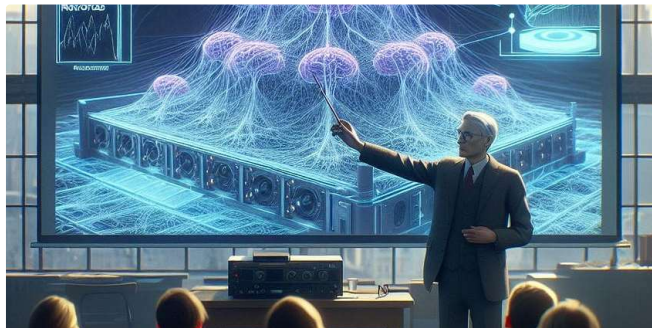
🌟 · 12 min read · Nov 4, 2022



1.9K



25



Austin Starks

Reinforcement Learning is Dead. Long Live the Transformer!

Large Language Models are more powerful than you imagine

7 min read · Jan 13



446



13



BearingPoint Data, Analytics in BearingPoint Data, Analytics

Transformers for Time-Series Data

By Lars Ødegaard Bentsen (Data Scientist) at BearingPoint

9 min read · Dec 19, 2023



58



Rokon

RNN vs. LSTM vs. Transformers: Unraveling the Secrets of...

In the realm of deep learning, sequential data processing is at the heart of many...

3 min read · Sep 24, 2023



2



See more recommendations