DEEP LEARNING

Master Positional Encoding: Part I

We present a "derivation" of the fixed positional encodi
Transformers, helping you get a full intuitive understan

Jonathan Kernes
Feb 15, 2021 21 min read

Hands-on Tutorials

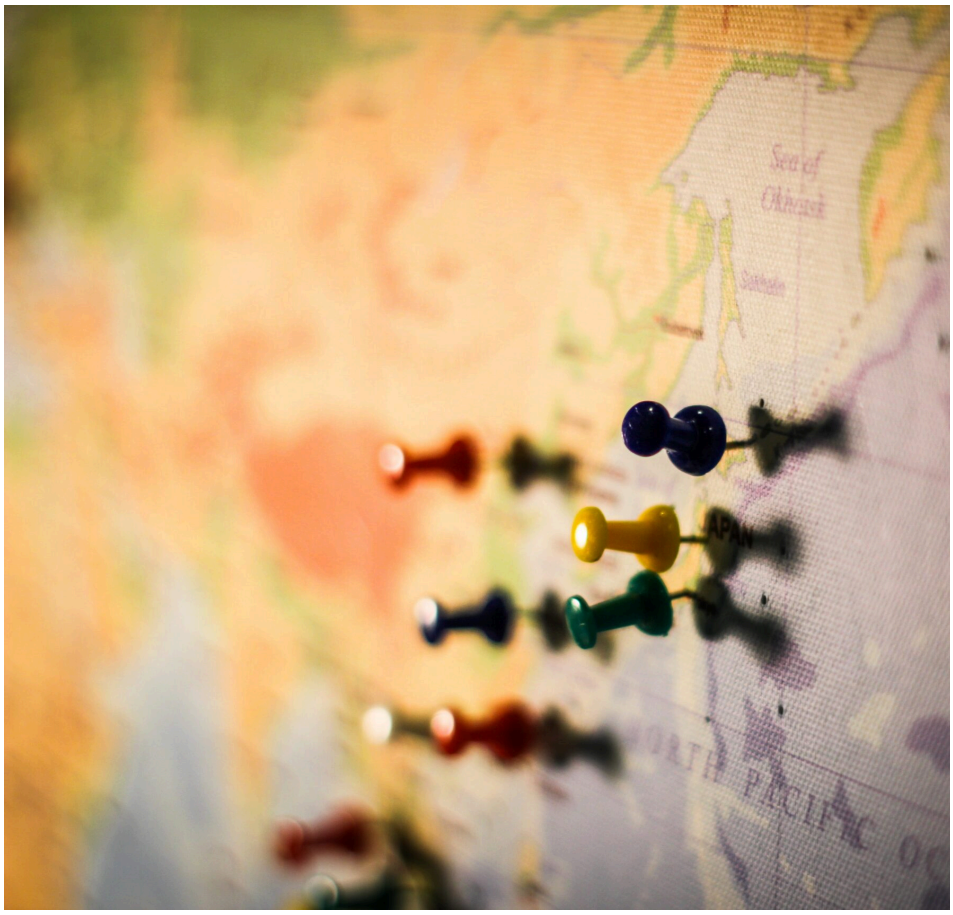


Photo by T.H. Chia on Unsplash

This is Part I of two posts on positional encoding
now available here!):

- **Part I:** the intuition and "derivation" of the fixed sinusoidal positional encoding.
- **Part II:** how do we, and how *should* we actually inject positional information into an attention model (or any other model that may need a positional embedding). This focuses on relative encodings.

As far as I know, there are some original insights i
positional encoding viewed as a manifold. But, th
so I doubt I' m the first to notice this; if anyone k
please let me know!

Part I: the fixed sinusoidal positiona

Introduction

Imagine you' ve finally managed to work your wa
understanding Transformers. You' ve drawn thos
matrices mapping keys to queries, and convinced
meaning of the logits tensor. You' ve even gone a
figured out how multi-head attention works, adm
and efficiency that a little tensor reshaping can pr
learned how to mask your input sequences by dir
logits and how to gracefully transfer sequences fr
decoder. You' re so satisfied with yourself you plo
couch, throw your feet up and treat yourself to a l
cookie... No. Two chocolate chip cookies. You' re

Contented with your newfound knowledge, you fi
embedding layer. "It' s easy," you muse to yourse
turning words into vectors since the backstreet bo
you turn your eyes towards the bottom of the mo
diagram, something unfamiliar catches your gaze.
encoding layer. You immediately stop eating your

falling to the floor, as you suddenly remember. *Positional information doesn't show up anywhere.*

Not yet panicking, you read ahead hoping you'll find see an easy solution. Instead all you get are a few sentences telling you about some fixed sinusoid function and the directions that when referring to the usual embedding layer "...the two can be summed." Furiously turning the page back and forth, you realize there

Having taken a physics class once before, you kno happened. You've been "it's trival"'ed.

Positional encoding is basically the "it's trivial" o One of many dreaded 2–3 word phrases like "it's matter", where the intent is most definitely the op you're saying. It's *not* fine, it *does* matter, and is *not* trivial.

Now, that we all acknowledge the situation, let's what positional encoding is. We are going to "der taking a physicist's approach. Come up with a fe theories/guesses for how to accomplish our task, upon them until we get an encoding that mostly s want. We will put off where and how to pop in the layer to Part II. For now, we are just trying to figur sequence some positional meaning.

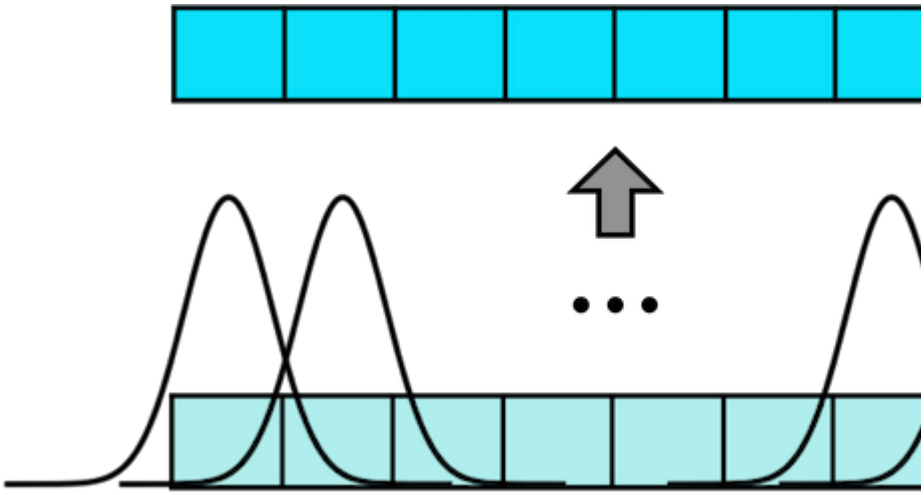
Discrete positional encoding

The most difficult part of this might just be trying exactly that we're trying to do. **If this subsectio much sense, just directly skip to the next one.** A the following goal:

> A positional encoding is a finite dimensional representation of the location or "position" of items in a sequence. Given some sequence A = [a*0, ..., a*{n-1}], the positional encoding must be some type of tensor that we can feed to a model to tell it where some value a_i is in the sequence A.

Admittedly, this is vague; this is because the "feed to a model" line is also vague. To fix this, let' s assume we want to tr the following task:

> Given a sequence A of length T, output another length T where the output for each index is the it' s neighbors.



Source: the current author. The bottom layer is our input encoding vector. The ov we feed this into, producing an output vector of the same size. Each dark blue s average of its lighter counterpart.

This model is meaningless. I repeat. This model is just to give us an idea of an example of one way y positional encoding to a neural network.

We are looking for a positional encoding *tensor* n encoding *network.* This means that that our posit tensor should be a representative of the position

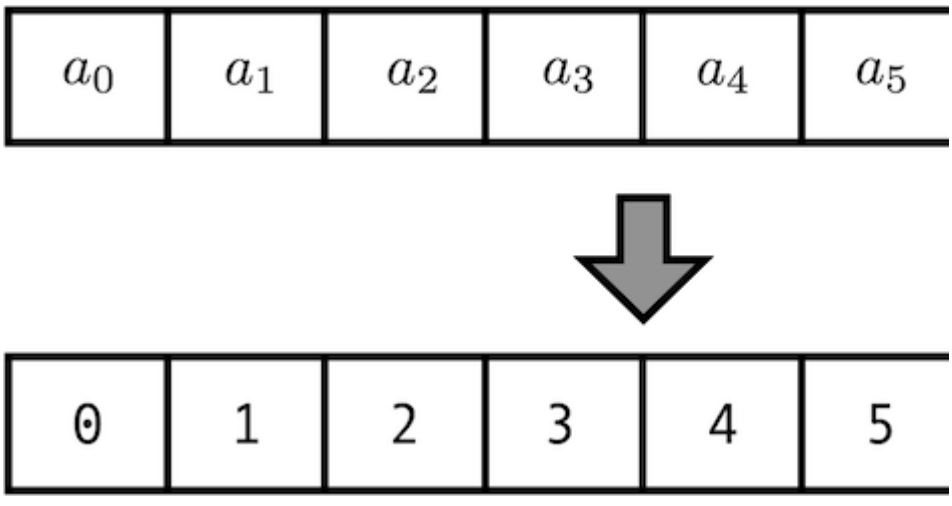
our network will take care of whatever operations need to be done to that.

This encoding is going to have the same dimension as the sequence length, which could be very long. Also, the sequence length is not even a fixed quantity. Feeding a variable length vector into a model is a big problem. We assume that there is some maximum value called the **max_position** for which our model just can't sequences. For all of our solutions we fix our encc and truncate down as needed for inputs.

Hopefully everything will be clear with an exampl

Guess #1: just count.

Let's start with the simplest positional encoding create. We just count! We map every every eleme the following encoding:



Source: current author. Sequence_length = 8. We simply create a positional enco number.

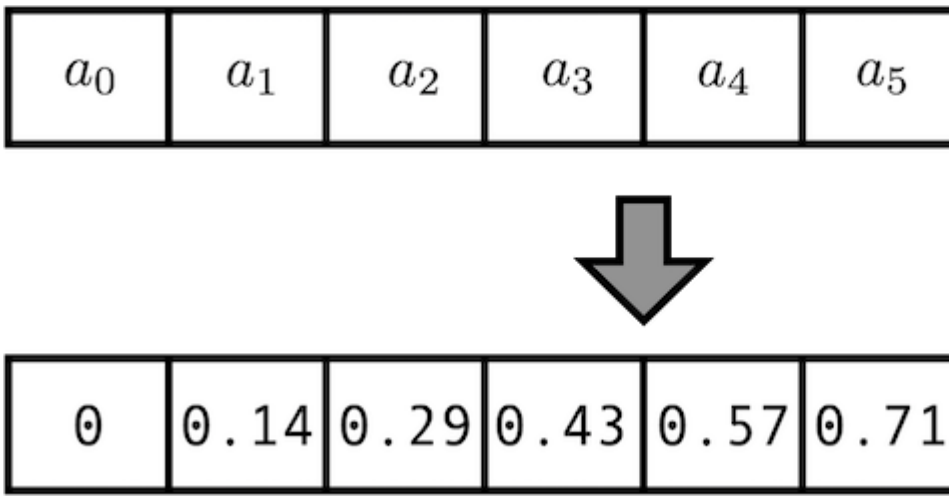
Pretty basic, we just created a new vector where e number. This is an **absolute positional encoding**. the sequence is labeled by its position relative to coordinates, assumed to be the start of the currer

continue to work with absolute encodings, as they can be used to derive **relative encodings**.

Now let's be critical. Here's what's wrong with our initial guess: the scale of these numbers is huge. If we have a sequence of 500 tokens, we'll end up with a 500 in our vector. In general, neural nets like their weights to hover around zero, and usually be equally balanced positive and negative. If not, you open y sorts of problems, like exploding gradients and ur

Guess #2: normalize the "just count" guess

This seems like the obvious solution → Just divide largest integer so all of the values are in [0,1]. This



Source: current author. We normalize each entry by dividing by the sequer

But oh no! we've introduced another problem. W deal with arbitrary sequence length. That's beca entries is divided by the sequence length. A positi of say 0.8 means a totally different thing to a sequ than it does for one of length 20. (For length 5, 0. would be the 4th element. For sequence length 2(0.8 represents the 16th element!). **Naively norma work with variable sequence lengths**.

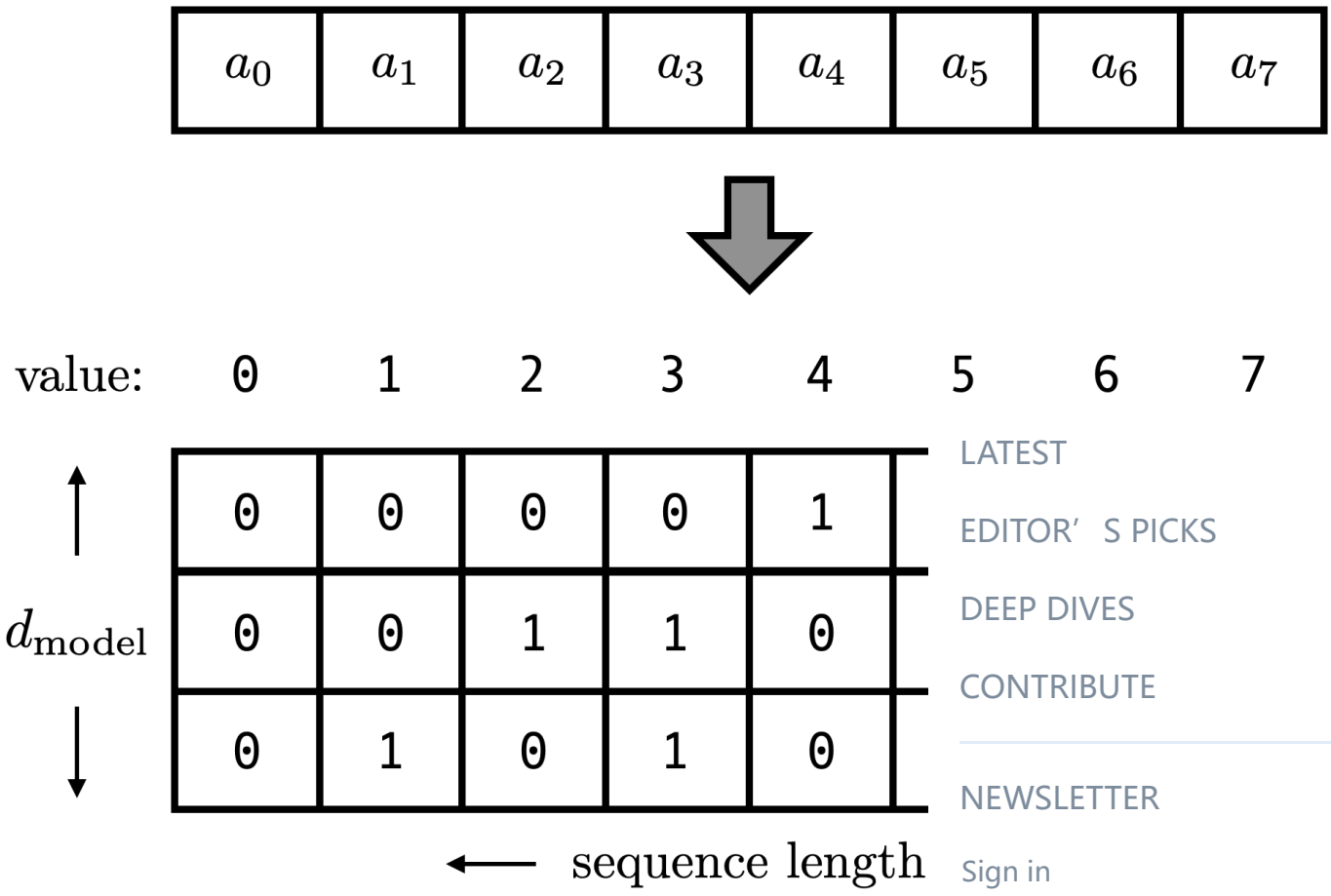
What we need to be able to do, is find a way to count without ever using numbers greater than 1. That's a big clue, and you may be able to guess the answer.

Use binary numbers.

Guess #3: just count! but using binary instead of decimal

Instead of writing say 35 for the 35th element, we represent it via its binary form 100011. Cool, ever one. But, in our excitement, we forgot that the nu whether we use binary or decimal; it's always jus gain anything... Instead, we actually need to do tw our integer to binary, and 2) convert our scalar int

Whoa, scalar to a vector? What does that mean? I positional encoding *vector* now becomes a positi Each number gets its own **binary vector** as oppos This is a pretty big conceptual shift, but by increas dimensionality, we are able to both keep arbitrary while also restricting numbers to the range [0,1]! looks like this:



Source: current author. The matrix version of positional encoding. Each binary vec
binary. e.g. index 5 holds 100 in binary, which is the v

We have some freedom. We can choose the dime
embedding space that holds the binary vector to
The only thing this dimension does is tell us how
can be, 2^dim to be precise. For simplicity, let' s
to _d*model,* the embedding dimension that our m
make sure we understand the vector representatic
would be represented as the binary vector

35 ← → [0, ..., 0, 1, 0, 0, 0, 1, 1]

Analysis time. What are some problems with our

1. We still haven' t fully normalized. Remember
   be positive and negative roughly equally. This
   rescale [0,1] ->[-1,1] via f(x) = 2x-1.

2. Our binary vectors come from a discrete function, and not a discretization of a continuous function.

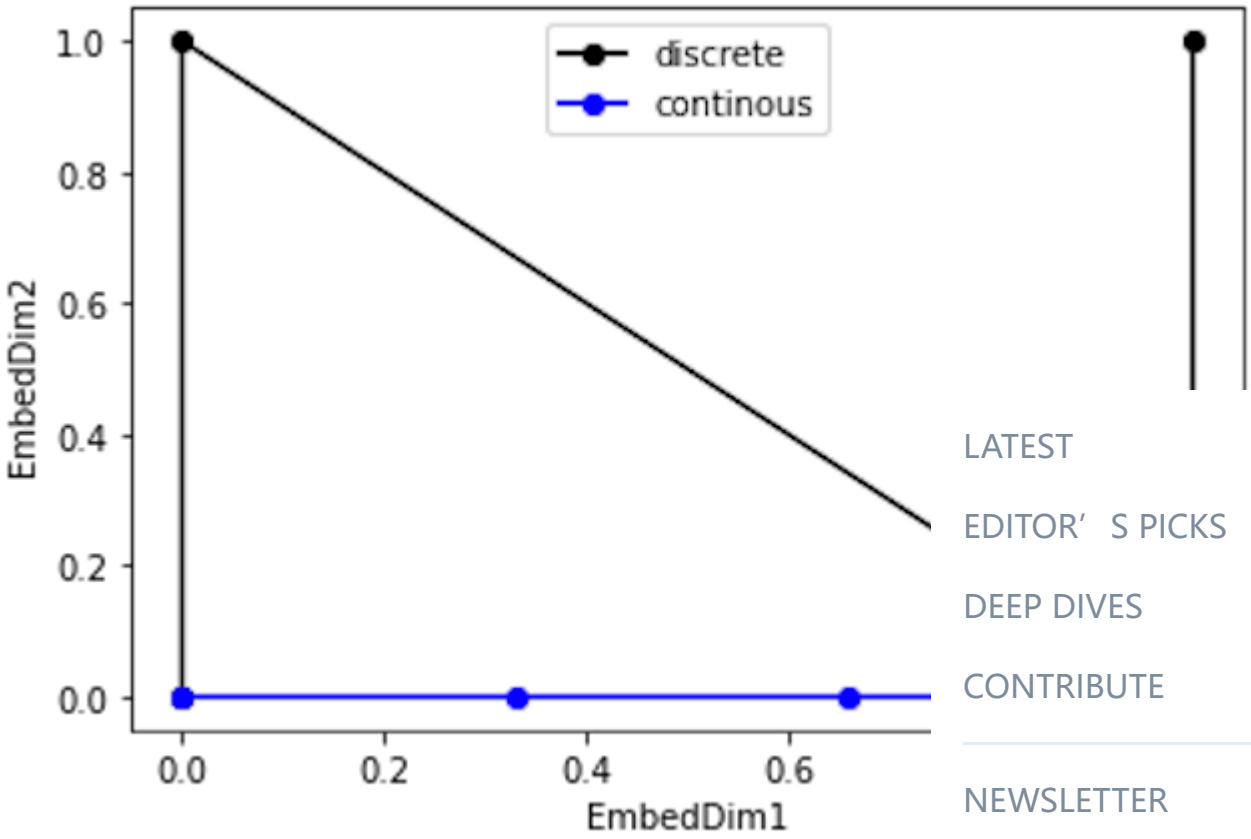
The latter point is subtle, and it has to do with using a *vector* to represent a *scalar* position. Let's compare two encodings, that try to measure a distance from 0 →3. The first encoding will be really simple: only use the x-axis dimension and put the position there, represented by a variable x in [0,3]

Continuous encoding: [x, 0]

Now let's compare it to our binary vector discret
this is discrete, we can only represent 4 positions:
give 4 points:

Binary vector encoding: [0,0], [0,1], [1,0], [1,1]

Any time we discretize something, we have inclina
that something can be **interpolated,** meaning tha
continuous function by connecting the dots. Take
interpolations for our two encodings:



Source: current author. The space curve showing flow of position. The discrete cur
(0,1)->(1,1), and is much more jagged than the smooth continu

You can see that the continuous encoding curve i
easy to estimate intermediate positions. For the b
thing is a mess. This will lead us to our next hint: **f**
the binary vector a discretization of something

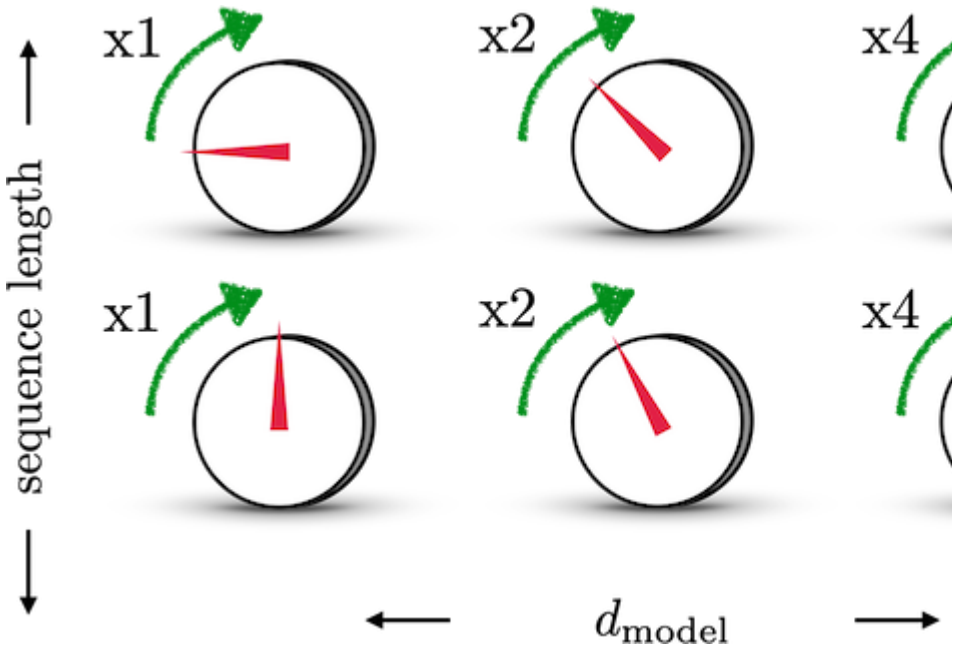
So with pictures, we want a function that connect
smooth way that looks natural. For anyone who h
what we are really doing is finding an embedding

Yes, if you use positional encoding, you can sounc
people that you' re using manifolds. What this m
terms, is that we are looking for a curve in the _d*
space, that as you walk along it slowly increases y
continuous way. Doing this properly brings us to

Guess #4: use a *continuous* binary vector

To make our binary vector continuous, we need a function that interpolates a back and forth cycle 0 →1 →0 →1 →0... etc. Cycles you say? Time to bring in trig functions. Let' s use a sine function. Also, a sine function exists on [-1,1] so it' s also properly normalized, which is an added bonus!

Here' s the image/analogy you should now have. Every matrix entry in the positional encoding matrix represents a "di Typically your dial might do something like adjust (volume is analogous to position). Like normal dia dials we use in the encoding matrix can continuou to on (1). To get a precise volume (position), we u (vector) of dials with different sensitivity. The first volume ever so slightly, perhaps by 1 unit, a barel The second will be more powerful, adjusting volu third will adjust by 4 units, the fourth by 8 units, e



Source: current author. Each position in the sequence (column) is represented by a that be visualized a setting of a bunch of dials. Dials are depict matrix elements. As more sensitive, and increase position more.

In this way, you can get a precise volume *over vas* strength increases exponentially. Instead of buildi that contains say 512 sound levels, we can build 8

as much precision (2^8=256 if you're wondering why 8 dials). Super efficient huh!

NOTATION WARNING: we transposed the positional encoding matrix relative to the previous section. We now have dim M = (sequence_length, d_model), which is the usual way this is written.

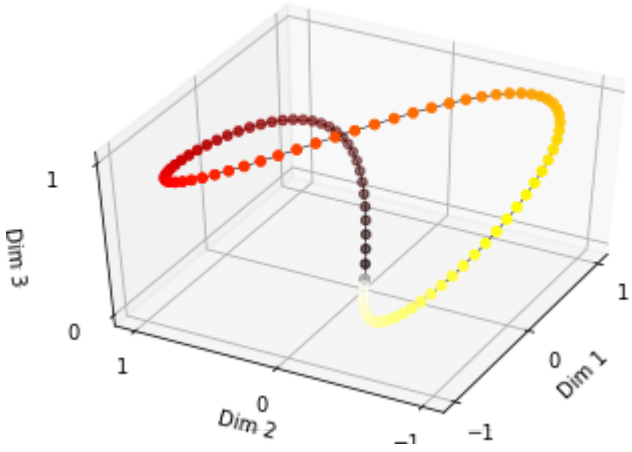
We now want our sine functions to undergo one r
the right moments. For the first entry, it should sh
every time we move one step in the sequence. Th
frequency of pi/2 for the first dial, pi/4 for the sec
third, and on and on...

We now have our first prototype positional encod
of a **matrix denoted by M**, where the y-axis of th
discrete position _x*i* in the sequence (0,1, ..., n-1),
vector embedding dimension. Elements of the ma

$$M_{ij} = \sin(2\pi i/2^j) = \sin(x_i\omega_j)$$

Here, omega is the dial frequency, which is monotonically decreasing with respect
integer giving the sequence position.

Let's now compare our continuous-discretized e
plain old discretized encoding.



Source: current author. The colors indicate the absolute position that that coord
omega=pi/2, pi/4, pi/8 for dimensions 1, 2 and 3 respe

Look, it’ s all continuous-y and manifold-y.

Analysis time. We are very close to the final answer. There are two things we still want to fix.

First, there is a discontinuity problem. The above plot is a smooth curve like we wanted. However, it is also a closed curve. Consider moving from position *n*, to *n+1.* This would be th that transitions yellow →brown. By construction, t equivalent to position 1. This was necessary to ma continuous, but it’ s also not what we want! *n+1* apart, not close together. For anyone who has stu transforms, this problem should be familiar. Only frequencies are unique.

To fix this, we observe that there’ s no reason tha our embedding vector line up with a binary vecto frequencies (omega_0, omega_1, ..., omega_d_mo so long as they are *monotonically increasing,* whi increase, and only increase. We can lower all of th this way keep ourselves far away from the bounda

$$M_{ij} = \sin(x_i \omega_0^{j/d_{\text{model}}})$$

The slowed down dial version. omega_0 is the min frequency. Check the edge cas j=d_model gives the smallest.

The original authors choose a minimum frequenc purpose. they then increase frequencies exponent omega_min^{range(d)/d} to achieve the monotor choose 10,000? I truly have no idea, I’ m guessin made this work the best.

The second problem isn’ t a problem insomuch a There is no simple operation that allows us to adc

positional units from a position vector. Such an operation, known as translation, would allow us to write the positional encoding vector of index *i* as a function of the vector for index *j*.

Why is this a very desirable property to have? Imagine we have a network that is trying to translate the sentence "I am going to eat." The combination "is/am/are" +"going"+"to" +"verb" is a very common grammatical structure, with a fixed positional stru ends up at index 1, "to" at index 2, etc. In this case "verb", we may want the network to learn to pay a "I" that occurs before "am going to". "I" is located the left of "verb". Since our attention layer uses lir to form the keys, queries, and values, it would be positional encodings such that the linear transfori the position vector located 4 units to the left, so t the position vector of "verb". The query and key w perfectly.

In summary, we want to see if we can modify our can be translated via linear transformation.

Final guess, #5: Use both sine and cosine to ma

The math is going to seem a little heavy, but it' s just need to keep track of notation. First, let' s su positional encoding matrix

$$\mathrm{PE} = \begin{pmatrix} \mathbf{v}^{(0)} \\ \vdots \\ \mathbf{v}^{(\text{seq len}-1)} \end{pmatrix}$$

The row-vector **v** is a vector of sines evaluated at but with varying frequencies

$$\mathbf{v}^{(i)} = [\sin(\omega_0 x_i), \cdots, \sin(\omega_{n-1} x_i)]$$

Each row-vector represents the positional encoding vector of a single, discrete position. The positional encoding matrix **PE,** is a vector of these vectors, and so it is a (seq_len, n) dimensional matrix (seq_len is the sequence length). We now want to find a linear transformation **T(dx)** such that the following equation holds:

$$\mathrm{PE}(x + \Delta x) = \mathrm{PE}(x) \cdot \mathbf{T}(\Delta x)$$

How will we do this? First, notice that since all of t
are sines, the positions *x* are actually *angles*. From
know that any operation T that shifts the argumer
must be some kind of rotation. Rotations can fam
applying a linear transformation to a (cosine, sine
standard results for rotation matrices, we can mak
following identity:

$$\begin{pmatrix} \cos(\theta + \phi) \\ \sin(\theta + \phi) \end{pmatrix} = \begin{pmatrix} \cos\phi & -\sin\phi \\ \sin\phi & \cos\phi \end{pmatrix} \begin{pmatrix} \cos\theta \\ \sin\theta \end{pmatrix}$$

to build a rotation.

> Note, this formula has the linear operator actin
> our network, linear transformations are conven
> the right as we have so far shown. It' s concep
> the left operation so we will write that answer h
> mind that you will need to take the transpose c
> current **PE** matrix.

To build an encoding that supports translations vi
we do the following. Create a duplicate encoding
instead of sine. Now, build a new positional encoc
follows: alternating between the cosine and sine r
first column and append it to the final **PE** matrix.

replacing every sin(...) with a [cos(...) sin(...)] pair. In our new **PE** matrix, the row-vectors look like the following:

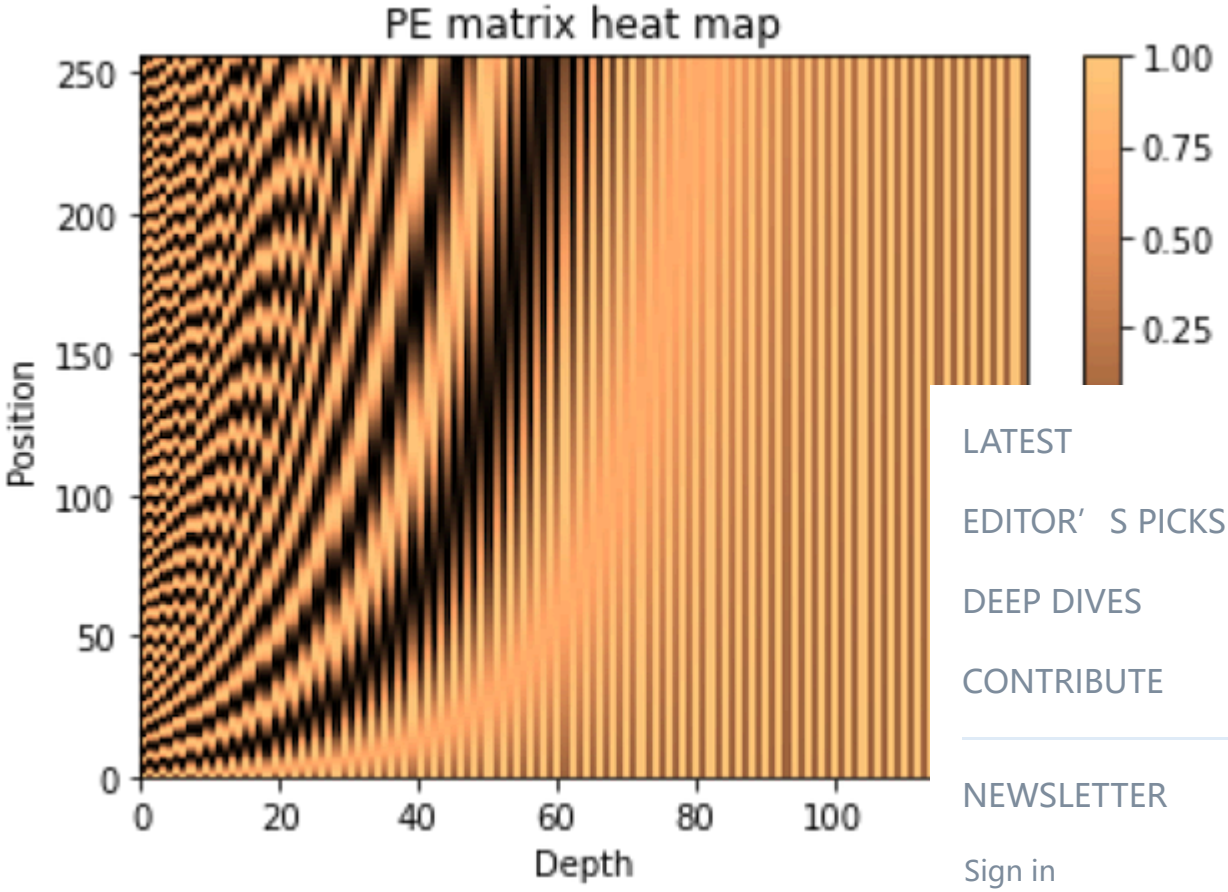
$$\mathbf{v}^{(i)} = [\cos(\omega_0 x_i), \sin(\omega_0 x_i), \ldots, \cos(\omega_{n-1} x_i), \sin(\omega_{n-1} x_i)]$$

We now build the full linear transformation by using a bunch of block-diagonal linear transformations. Each block will have a different matrix, since the frequencies that the block acts on are di
in order to translate the _k_th dial with frequency
we would need a total angle shift of delta=omega
can now be written as:

$$\mathbf{T}(\Delta x) = \begin{pmatrix} \begin{bmatrix} \cos(\omega_0 \Delta x) & -\sin(\omega_0 \Delta x) \\ \sin(\omega_0 \Delta x) & \cos(\omega_0 \Delta x) \end{bmatrix} & \cdots \\ \vdots & \ddots \\ 0 & \cdots \end{pmatrix}$$

If you take the transpose of this, you can directly i
previous PE(x+dx)=PE(x)*T equation, thereby prov
the existence of a translation matrix!

That wraps up mostly everything. Let's now take
PE matrix actually looks like.



Source: current author.

Here' s how to interpret the above PE matrix hea

1. **Most of the PE matrix is not needed for enc** plot that are darker indicate dials that have be meaning that the sines and cosines in that reg away from their initial values of 0 and 1 respe how much of the embedding space is being u positional information. As you can see by foll curve-ish thing, activating a dial one step dee becomes exponentially more difficult (the bla e^x).
2. **Vertical lines at greater depth vary less thai depth.** Pick a fixed depth, then move upward color shifts light →dark →light… The frequen *decreases* with depth, showing our intuition e at greater depth are more sensitive.

Final answer. No more guesses

At this point we' ve done everything we need to do, so let' s summarize:

1. Positional encoding is represented by a matrix. For sequence length T and model depth D, this would be a (T, D) tensor.
2. The "positions" are just the indices in the seq
3. Each row of the PE matrix is a vector that repr interpolated position of the discrete value ass index
4. The row-vector is an alternating series of sine frequencies that decrease according to a geo
5. There exists a matrix multiplication that can s any row-vector we want.

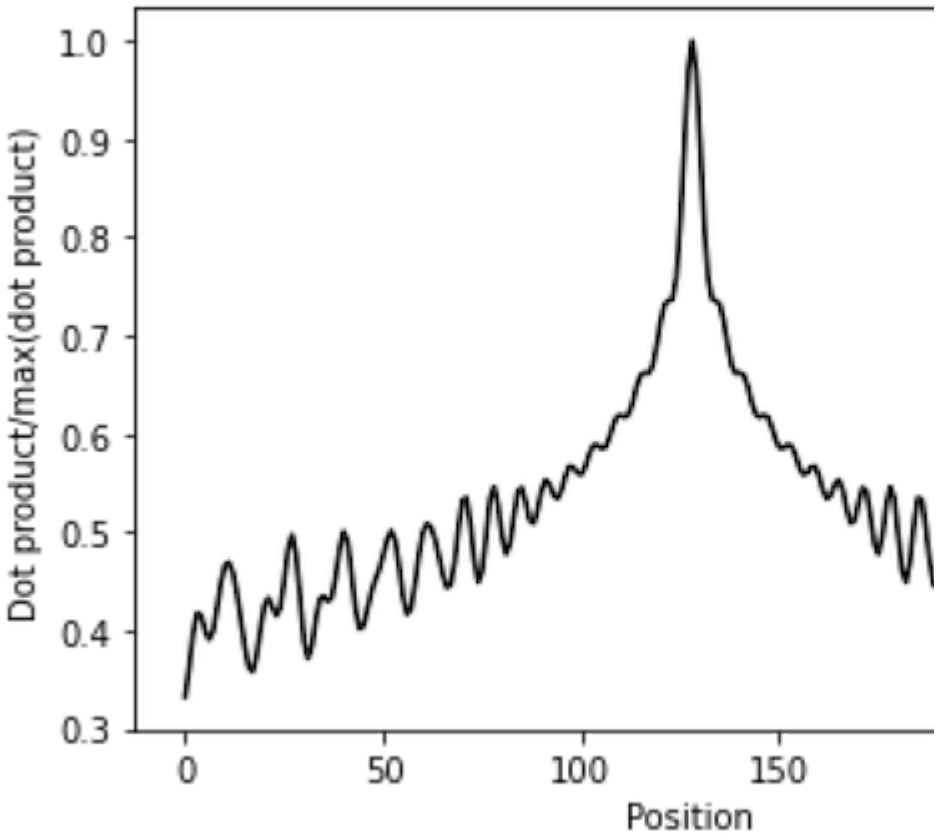
Here are two ways to create a positional encodinç numpy and one using only TensorFlow operators. encoding matrix can be initialized at the same tim uses it, this formulation isn' t of much use. But it' working with tensors.

A couple of notes to understand the code. First w broadcasting to create the PE matrix angles via ar positions and frequencies. Second, we directly ap to slices of the angle matrix, using subscripts, whi

Bonus property

There is an additional property that we would like encoding to satisfy, and it is related to how positi used in an actual attention model. Attention is pre that each embedding associated with each positic

has some representative key or query. These keys and queries allow us to quickly determine is the current position should "attend" on another position, by performing the dot product query_i@key_j. If the dot product is large, then it implies the key at *j* matches the query at *i*. In order for this to work with positional encoding vectors, we would like that positions close to each other return large key-query dot products, while those far away return small ones. [illegible] figure below:



Source: current author. We show the dot product of vector v^{(128)} with all othe
with parameters d_embed=128, max_position=256. Dot products are nor

This figure shows the dot product between a part
encoding vector representing the 128th position,
positional encoding vector. Notice, that without a
for this, we get exactly what we wanted. The dot p
when comparing v^{(128)} to itself, and steadily d
further away. A nice bonus feature!

The reasons for this can be traced back to the orthogonality of sines and cosines. As we get further away, the frequencies between positional encoding vectors are no longer correlated, and the summation of random sines and cosines is known to tend to zero.

Conclusion

Thanks for sticking with it until the end. Like all th
things that are explained the least tend to be the
but oh well.

We learned that positional encoding is a means o
location of objects in a sequence into information
network (or other model) can understand and use
approach where we just give the model integers c
position, we learned that this awful for a number
numbers can be too large, there' s not an obviou
information, it doesn' t extend well to sequences

We next made a series of guesses, acting like phy
produce a working theory. By continually adjustin
guesses to incorporate more desired characteristi
landed on sinusoidal positional encoding matrix.
vectors to represent sequence positions. It can be
allows for translations, and deal with varying sequ

The details and intuition behind implementation
explored in a future article. For example, in that a
questions like "why do we just add positional enc
sequence embeddings?" Here, it seems like a dec
answer is that the PE matrix is sparse. Only a smal
so it seems like the model may treat the addition

concatenation, reserving some part of the embedding for position, and the rest for tokens.

References

1. Vaswani, Ashish, et al. "Attention is all you need." *arXiv preprint arXiv:1706.03762* (2017).

WRITTEN BY

Jonathan Kernes

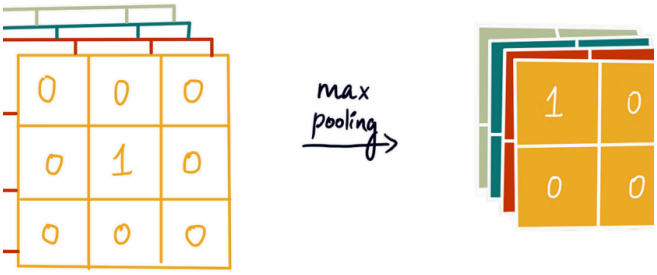
See all from Jonathan Kernes

Topics:

Deep Learning | Editors Pick | Hands On Tutorials | Transformers

Share this article:

Related Articles



ARTIFICIAL INTELLIGENCE

Implementing Convolutional Neural Networks in TensorFlow

Step-by-step code guide to building a Convolutional Neural Network

Shreya Rao

August 20, 2024 6 min read

DEEP LEARNING

Deep Dive ir xLSTMs by H

Explore the wisc xLSTMs - a probi present-day LLM

Srijanie Dey, PhD

July 9, 2024 13 r

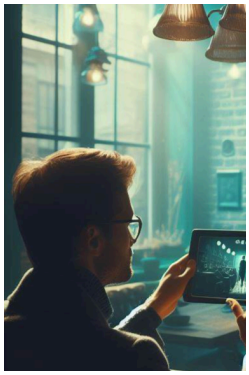
NLP

Check Your Biases

Symbolic Engines and Unexpected Results - A Personal Coding Experience

Veronica Villa

March 17, 2022 9 min read



CINEMA

Evaluating C Which synta features are

This article expl between a movi genre, leveragin

Christabelle Pabal

January 20, 2024

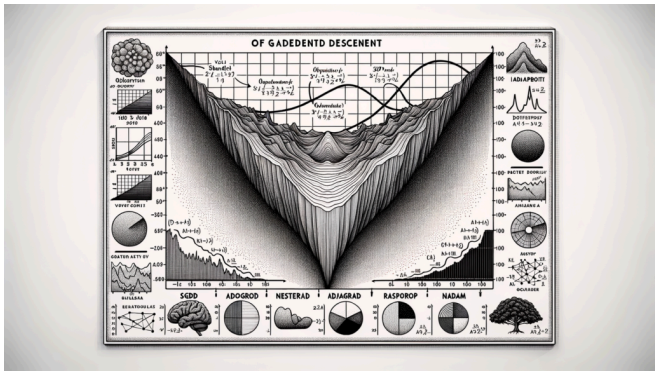
DEEP LEARNING

Speeding Up the Vision Transformer with BatchNorm

How integrating Batch Normalization in an encoder-only Transformer architecture can lead to reduced training time...

Anindya Dey, PhD

August 6, 2024 28 min read



DATA SCIENCE

The Math Be Optimizers: Understandi

This is a bit diffe say.

Peng Qian

August 17, 2024

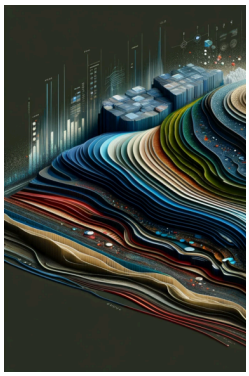
DATA SCIENCE

Latest picks: Time Series Forecasting with Deep Learning and Attention Mechanism

Your daily dose of data science

TDS Editors

November 4, 2020 1 min read



DATA SCIENCE

Stacked Ens Advanced Pr With H2O.ai

And how I place largest machine them!

Sheila Teo

December 18, 202

MACHINE LEARNING

Large Language Models, MirrorBERT - Transforming Models into Universal Lexical and Sentence...

Discover how mirror augmentation generates data and aces the BERT performance on semantic similarity tasks

Vyacheslav Efimov
December 12, 2023 7 min read

Email address*

First name*

Last name*

Job title*

Job level*

Please Select

Company name*

- I consent to receive newsletters and other communica Science publications.*

Subscribe Now