RNN

Unlike normal neural networks, RNNs are designed to take a series of inputs with no predetermined limit on size. The term “series” here denotes that each input of that sequence has some relationship with its neighbors or has some influence on them.

Basic feed-forward networks “remember” things too, but they remember the things they learned during training. Although RNNs learn similarly during training, they also remember things learned from prior input(s) while generating output(s).

RNNs can be used in multiple types of models.

1. Vector-Sequence Models — Take fixed-sized vectors as input and output vectors of any size. For example, in image captioning, the image is the input and the output describes the image.

2. Sequence-Vector Model — Take a vector of any size and output a vector of fixed size. For example, sentiment analysis of a movie rates the review of any movie, positive or negative, as a fixed size vector.

3. Sequence-to-Sequence Model — The most popular and most used variant, this takes a sequence as input and outputs another sequence with variant sizes. An example of this is language translation for time series data for stock market prediction.

An RNN has two major disadvantages, however:

For example, in the sentence “The clouds are in the ____.” the next word should obviously be sky, as it is linked with the clouds. If the distance between clouds and the predicted word is short, so the RNN can predict it easily.

Consider another example, however: “I grew up in Germany with my parents, I spent many years there and have proper knowledge about their culture. That’s why I speak fluent ____.”

Here the predicted word is German, which is directly connected with Germany. The distance between Germany and the predicted word is longer in this case, however, so it’s difficult for the RNN to predict.

So, unfortunately, as that gap grows, RNNs become unable to connect as their memory fades with distance.

RNN

برخلاف شبکه‌های عصبی معمولی، RNNها به گونه‌ای طراحی شده‌اند که مجموعه‌ای از ورودی‌ها را بدون محدودیت اندازه از پیش تعیین‌شده دریافت کنند. اصطلاح "سری" در اینجا نشان می دهد که هر ورودی از آن دنباله رابطه ای با همسایگان خود دارد یا تأثیری بر آنها دارد.

شبکه‌های اصلی پیش‌خور نیز چیزهایی را «به خاطر می‌آورند»، اما چیزهایی را که در طول آموزش آموخته‌اند به خاطر می‌آورند. اگرچه RNNها در طول آموزش به طور مشابه یاد می گیرند، اما در حین تولید خروجی، چیزهایی را که از ورودی(های) قبلی آموخته اند نیز به خاطر می آورند.

RNNها را می توان در انواع مختلفی از مدل ها استفاده کرد.

.۱مدل های توالی برداری - بردارهای با اندازه ثابت را به عنوان بردار ورودی و خروجی با هر اندازه در نظر بگیرید. به عنوان مثال، در زیرنویس تصویر، تصویر ورودی است و خروجی تصویر را توصیف می کند.

.۲مدل توالی بردار - یک بردار با هر اندازه ای بگیرید و یک بردار با اندازه ثابت خروجی بگیرید. به عنوان مثال، تجزیه و تحلیل احساسات یک فیلم، بررسی هر فیلم، مثبت یا منفی، را به عنوان یک بردار اندازه ثابت رتبه بندی می کند.

.۳مدل - Sequence-to-Sequence محبوب ترین و پرکاربردترین نوع، این یک دنباله را به عنوان ورودی می گیرد و دنباله دیگری را با اندازه های مختلف خروجی می دهد. نمونه ای از این ترجمه زبان برای داده های سری زمانی برای پیش بینی بازار سهام است.

با این حال، RNNدو عیب عمده دارد:

به عنوان مثال، در جمله "ابرها در ____ هستند." بدیهی است که کلمه بعدی باید آسمان باشد، زیرا با ابرها پیوند خورده است. اگر فاصله بین ابرها و کلمه پیش بینی شده کوتاه باشد، بنابراین RNN می تواند آن را به راحتی پیش بینی کند.

با این حال، مثال دیگری را در نظر بگیرید: «من در آلمان با پدر و مادرم بزرگ شدم، سال‌های زیادی را در آنجا گذراندم و شناخت درستی از فرهنگ آنها دارم. به همین دلیل است که من روان ____ صحبت می کنم.

در اینجا کلمه پیش بینی شده آلمانی است که مستقیماً با آلمان مرتبط است. فاصله بین آلمان و کلمه پیش‌بینی‌شده در این مورد طولانی‌تر است، بنابراین پیش‌بینی برای RNN دشوار است.

بنابراین، متأسفانه، با افزایش این شکاف، RNNها قادر به اتصال نیستند زیرا حافظه آنها با فاصله کم می شود.

LSTM

Long short-term memory is a special kind of RNN, specially made for solving vanishing gradient problems. They are capable of learning long-term dependencies. In fact, remembering information for long periods of time is practically their default behavior, not something they struggle to learn!

LSTM neurons, unlike the normal version, have a branch that allows passing information to skip the long processing of the current cell. This branch allows the network to retain memory for a longer period of time. It improves the vanishing gradient problem but not terribly well: It will do fine until 100 words, but around 1,000 words, it starts to lose its grip.

Further, like the simple RNN, it is also very slow to train, and perhaps even slower. These systems take input sequentially one by one, which doesn't use up GPUs very well, which are designed for parallel computation. Later, I'll address how we can parallelize sequential data. For now, we are dealing with two issues:

- Vanishing gradient
- Slow training

Solving the Vanishing Gradient Issue

Attention answers the question of what part of the input we should focus on. I'm going to explain attention via a hypothetical scenario:

Suppose someone gave us a book on machine learning and asked us to compile all the information about categorical cross-entropy. There are two ways of doing such a task. First, we could read the whole book and come back with the answer. Second, we could skip to the index, find the chapter on losses, go to the cross-entropy part and just read the relevant information on categorical cross-entropy.

Which do you think is the faster method?

The first approach may take a whole week, whereas the second should just take a few minutes. Furthermore, our results from the first method will be vaguer and full of too much information. The second approach will more accurately meet the requirement.

What did we do differently here?

LSTM

حافظه کوتاه مدت یک نوع خاص از RNN است که به طور ویژه برای حل مشکلات گرادیان ناپدید شده ساخته شده است. آنها قادر به یادگیری وابستگی های طولانی مدت هستند. در واقع، به خاطر سپردن اطلاعات برای مدت طولانی عملاً رفتار پیش فرض آنهاست، نه چیزی که برای یادگیری آن تلاش می کنند!

نورون هایLSTM ، بر خلاف نسخه معمولی، دارای شاخه ای هستند که به انتقال اطلاعات اجازه می دهد تا پردازش طولانی سلول فعلی را نادیده بگیرد. این شاخه به شبکه اجازه می دهد تا حافظه را برای مدت زمان طولانی تری حفظ کند. مشکل گرادیان ناپدید شدن را بهبود می بخشد، اما نه خیلی خوب: تا ۱۰۰ کلمه خوب عمل می کند، اما حدود ۱۰۰۰ کلمه، شروع به از دست دادن قدرت خود می کند.

علاوه بر این، مانند RNN ساده، آموزش آن نیز بسیار کند است و شاید حتی کندتر. این سیستم‌ها ورودی‌های متوالی را یکی یکی دریافت می‌کنند، که از پردازنده‌های گرافیکی که برای محاسبات موازی طراحی شده‌اند، به خوبی استفاده نمی‌کنند. بعداً به نحوه موازی سازی داده های متوالی خواهم پرداخت. در حال حاضر ما با دو موضوع سروکار داریم:

- شیب ناپدید شدن
- تمرین آهسته

حل مشکل ناپدید شدن گرادیان

توجه به این سوال پاسخ می دهد که باید روی چه بخشی از ورودی تمرکز کنیم. من قصد دارم توجه را از طریق یک سناریوی فرضی توضیح دهم:

فرض کنید شخصی کتابی در مورد یادگیری ماشین به ما داد و از ما خواست که تمام اطلاعات مربوط به آنتروپی متقاطع طبقه‌ای را گردآوری کنیم. دو راه برای انجام چنین کاری وجود دارد. ابتدا می‌توانیم کل کتاب را بخوانیم و با جواب برگردیم. دوم، ما می‌توانیم به فهرست پرش کنیم، فصل تلفات را پیدا کنیم، به قسمت آنتروپی متقاطع برویم و فقط اطلاعات مربوطه را در مورد آنتروپی متقاطع طبقه‌ای بخوانیم.

به نظر شما کدام روش سریعتر است؟

رویکرد اول ممکن است یک هفته کامل طول بکشد، در حالی که روش دوم باید فقط چند دقیقه طول بکشد. علاوه بر این، نتایج ما از روش اول مبهم و پر از اطلاعات بیش از حد خواهد بود. رویکرد دوم با دقت بیشتری نیاز را برآورده می کند.

اینجا چه کار متفاوتی انجام دادیم؟

In the former case, we didn't zero in on any one part of the book. In the latter method, however, we focused our attention on the losses chapter and more specifically on the part where the concept of categorical cross-entropy is explained. This second version is the way most of us humans would actually do this task.

Attention in neural networks is somewhat similar to what we find in humans. It means they focus on certain parts of the inputs while the rest gets less emphasis. Let's say we are making an NMT (neural machine translator). This animation shows how a simple seq-to-seq model works.

We see that, for each step of the encoder or decoder, the RNN is processing its inputs and generating output for that time step. In each time step, the RNN updates its hidden state based on the inputs and previous outputs it has seen. In the animation, we see that the hidden state is actually the context vector we pass along to the decoder.

Time for Attention

The context vector turns out to be problematic for these types of models, which struggle when dealing with long sentences. Or they may have been facing the vanishing gradient problem in long sentences. So, a solution came along in a paper that introduced attention. It highly improved the quality of machine translation as it allows the model to focus on the relevant part of the input sequence as necessary.

This attention model is different from the classic seq-to-seq model in two ways. First, as compared to a simple seq-to-seq model, here, the encoder passes a lot more data to the decoder. Previously, only the final, hidden state of the encoding part was sent to the decoder, but now the encoder passes all the hidden states, even the intermediate ones. The decoder part also does an extra step before producing its output. This step proceeds like this:

It checks each hidden state that it received as every hidden state of the encoder is mostly associated with a particular word of the input sentence. It gives each hidden state a score.

Each score is multiplied by its respective SoftMax score, thus amplifying hidden states with high scores and drowning out hidden states with low scores. A clear visualization is available here. This scoring exercise happens at each time step on the decoder side.

در مورد اول، ما هیچ بخشی از کتاب را صفر نکردیم. با این حال، در روش اخیر، ما توجه خود را بر فصل ضرر و به طور خاص بر بخشی که مفهوم آنتروپی متقاطع طبقه ای توضیح داده می شود، متمرکز کردیم. این نسخه دوم روشی است که اکثر ما انسان ها واقعاً این کار را انجام می دهیم.

توجه در شبکه های عصبی تا حدودی شبیه به آنچه در انسان می یابیم است. این بدان معنی است که آنها روی بخش های خاصی از ورودی ها تمرکز می کنند در حالی که بقیه تأکید کمتری دارند. فرض کنید در حال ساخت یک NMT (مترجم ماشین عصبی) هستیم. این انیمیشن نشان می دهد که چگونه یک مدل seq-to-seq ساده کار می کند.

می بینیم که برای هر مرحله از رمزگذار یا رمزگشا، RNN ورودی های خود را پردازش می کند و خروجی آن مرحله زمانی را تولید می کند. در هر مرحله زمانی، RNN وضعیت پنهان خود را بر اساس ورودی ها و خروجی های قبلی که دیده است به روز می کند. در انیمیشن می بینیم که حالت پنهان در واقع همان بردار زمینه ای است که به رمزگشا ارسال می کنیم.

زمان توجه

بردار زمینه برای این نوع مدل‌ها مشکل‌ساز است، که هنگام برخورد با جملات طولانی مشکل دارند. یا ممکن است در جملات طولانی با مشکل گرادیان ناپدید شدن مواجه شده باشند. بنابراین، راه حلی در مقاله ای آمد که توجه را معرفی کرد. کیفیت ترجمه ماشینی را بسیار بهبود بخشید زیرا به مدل اجازه می‌دهد تا در صورت لزوم روی قسمت مربوطه از دنباله ورودی تمرکز کند.

این مدل توجه از دو جهت با مدل کلاسیک seq-to-seq متفاوت است. اول، در مقایسه با یک مدل سادهseq-to-seq ، در اینجا، رمزگذار داده های بسیار بیشتری را به رمزگشا ارسال می کند. قبلاً فقط حالت نهایی و پنهان قسمت رمزگذاری به رمزگشا ارسال می شد، اما اکنون انکودر از تمام حالت های پنهان حتی حالت های میانی عبور می کند.

قسمت رمزگشا نیز یک مرحله اضافی را قبل از تولید خروجی انجام می دهد. این مرحله به این صورت پیش می رود:

هر حالت پنهان دریافت شده را بررسی می کند زیرا هر حالت پنهان رمزگذار بیشتر با کلمه خاصی از جمله ورودی مرتبط است.

به هر حالت پنهان یک امتیاز می دهد.

هر امتیاز در امتیاز سافت مکس مربوطه ضرب می‌شود، بنابراین حالت‌های پنهان با امتیازات بالا تقویت می‌شوند و حالت‌های پنهان با امتیازات پایین غرق می‌شوند. یک تجسم واضح در اینجا موجود است. این تمرین امتیاز دهی در هر مرحله زمانی در سمت رمزگشا انجام می شود.

Now, when we bring the whole thing together:

The attention decoder layer takes the embedding of the <END> token and an initial decoder hidden state. The RNN processes its inputs and produces an output and a new hidden state vector (h4).

Now, we use encoder hidden states and the h4 vector to calculate a context vector, C4, for this time step. This is where the attention concept is applied, giving it the name the attention step.

We concatenate (h4) and C4 in one vector.

Now, this vector is passed into a feed-forward neural network. The output of the feed-forward neural networks indicates the output word of this time step.

These steps get repeated for the next time steps. A clear visualization is available here.

So, this is how attention works. For further clarification, you can see its application to an image captioning problem here.

Now, remember earlier I mentioned parallelizing sequential data? Here comes our ammunition for doing just that.

حالا، وقتی همه چیز را با هم جمع می کنیم:

لایه رمزگشای توجه، جاسازی نشانه <END> و یک حالت پنهان رمزگشای اولیه را می گیرد RNN .ورودی های خود را پردازش می کند و یک خروجی و یک بردار حالت پنهان جدید (h۴) تولید می کند.

حال، از حالت‌های پنهان رمزگذار و بردار h۴ برای محاسبه بردار زمینه، C۴، برای این مرحله زمانی استفاده می‌کنیم. اینجاست که مفهوم توجه به کار می رود و نام آن را مرحله توجه می گذارد.

ما (h۴) و C۴ را در یک بردار به هم متصل می کنیم.

اکنون، این بردار به یک شبکه عصبی پیش‌خور منتقل می‌شود. خروجی شبکه‌های عصبی پیش‌خور، کلمه خروجی این مرحله زمانی را نشان می‌دهد.

این مراحل برای مراحل بعدی تکرار می شوند. یک تجسم واضح در اینجا موجود است.

بنابراین، توجه اینگونه عمل می کند. برای توضیح بیشتر، می‌توانید کاربرد آن را برای مشکل نوشتن تصویر در اینجا ببینید.

حالا، به یاد دارید که قبلاً به موازی کردن داده های متوالی اشاره کردم؟ در اینجا مهمات ما برای انجام این کار آمده است.

Transformers

مبدل ها

A paper called “Attention Is All You Need,” published in 2017, introduced an encoder-decoder architecture based on attention layers, which the authors called the transformer.

One main difference is that the input sequence can be passed parallelly so that GPU can be used effectively and the speed of training can also be increased. It is also based on the multi-headed attention layer, so it easily overcomes the vanishing gradient issue. The paper applies the transformer to an NMT.

So, both of the problems that we highlighted before are partially solved here.

For example, in a translator made up of a simple RNN, we input our sequence or the sentence in a continuous manner, one word at a time, to generate word embeddings. As every word depends on the previous word, its hidden state acts accordingly, so we have to feed it in one step at a time.

In a transformer, however, we can pass all the words of a sentence and determine the word embedding simultaneously. So, let’s see how it’s actually working:

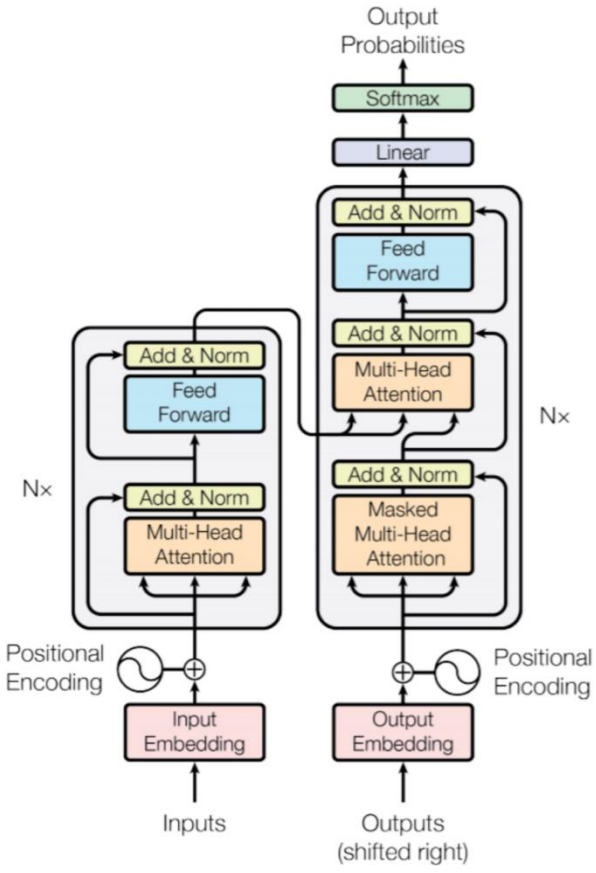
مقاله ای به نام «توجه همه آن چیزی است که نیاز دارید» که در سال ۲۰۱۷ منتشر شد، معماری رمزگذار-رمزگشا را بر اساس لایه های توجه معرفی کرد که نویسندگان آن را ترانسفورماتور نامیدند.

یکی از تفاوت های اصلی این است که توالی ورودی را می توان به صورت موازی عبور داد تا بتوان از GPU به طور موثر استفاده کرد و سرعت آموزش را نیز افزایش داد. همچنین بر اساس لایه توجه چند سر است، بنابراین به راحتی بر مشکل گرادیان ناپدید غلبه می کند. کاغذ ترانسفورماتور را به یک NMT اعمال می کند.

بنابراین، هر دو مشکلی که قبلاً برجسته کردیم تا حدی در اینجا حل شده است.

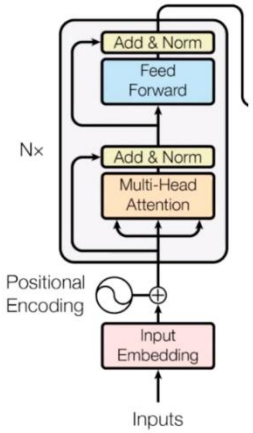
به عنوان مثال، در مترجمی که از یک RNN ساده تشکیل شده است، دنباله یا جمله خود را به صورت پیوسته، هر بار یک کلمه وارد می کنیم تا جاسازی های کلمه ایجاد شود. از آنجایی که هر کلمه به کلمه قبلی بستگی دارد، حالت پنهان آن مطابق با آن عمل می کند، بنابراین باید در یک مرحله آن را تغذیه کنیم.

اما در یک ترانسفورماتور می‌توانیم همه کلمات یک جمله را منتقل کنیم و کلمه embedding را به طور همزمان تعیین کنیم. بنابراین، بیایید ببینیم که در واقع چگونه کار می کند:



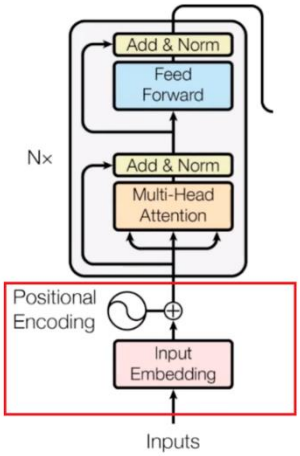
Encoder Block

بلوک رمزگذار



Computers don't understand words. Instead, they work on numbers, vectors or matrices. So, we need to convert our words to a vector. But how is this possible? Here's where the concept of embedding space comes into play. It's like an open space or dictionary where words of similar meanings are grouped together. This is called an embedding space, and here every word, according to its meaning, is mapped and assigned with a particular value. Thus, we convert our words into vectors.

کامپیوترها کلمات را نمی فهمند. در عوض، آنها بر روی اعداد، بردارها یا ماتریس ها کار می کنند. بنابراین، ما باید کلمات خود را به بردار تبدیل کنیم. ولی چطور این ممکن است؟ اینجا جایی است که مفهوم جاسازی فضا مطرح می شود. این مانند یک فضای باز یا فرهنگ لغت است که در آن کلمات با معانی مشابه در کنار هم قرار می گیرند. به این فضای تعبیه می گویند و در اینجا هر کلمه با توجه به معنای خود، نگاشت می شود و با مقدار خاصی تخصیص می یابد. بنابراین، ما کلمات خود را به بردار تبدیل می کنیم.



One other issue we will face is that, in different sentences, each word may take on different meanings. So, to solve this issue, we use positional encoders. These are vectors that give context according to the position of the word in a sentence.

Word → Embedding → Positional Embedding → Final Vector, framed as Context.

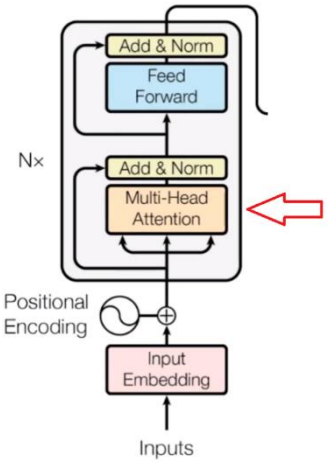
So, now that our input is ready, it goes to the encoder block.

یکی دیگر از مسائلی که با آن روبرو خواهیم شد این است که در جملات مختلف، هر کلمه ممکن است معانی مختلفی به خود بگیرد. بنابراین برای حل این مشکل از رمزگذارهای موقعیتی استفاده می کنیم. اینها بردارهایی هستند که با توجه به موقعیت کلمه در یک جمله زمینه را می دهند.

Word → Embedding → Positional Embedding → Final Vector، به عنوان Context قاب شده است.

بنابراین، اکنون که ورودی ما آماده است، به بلوک رمزگذار می رود.

MULTI-HEAD ATTENTION PART



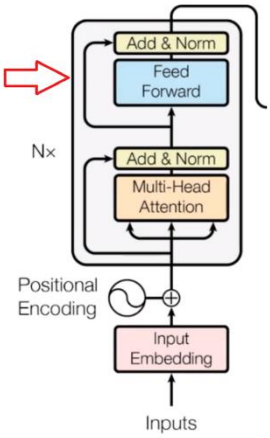
Now comes the main essence of the transformer: self-attention.

This focuses on how relevant a particular word is with respect to other words in the sentence. It is represented as an attention vector. For every word, we can generate an attention vector generated that captures the contextual relationship between words in that sentence.

The only problem now is that, for every word, it weighs its value much higher on itself in the sentence, but we want to know its interaction with other words of that sentence. So, we determine multiple attention vectors per word and take a weighted average to compute the final attention vector of every word.

As we are using multiple attention vectors, this process is called the multi-head attention block.

FEED-FORWARD NETWORK



بخش توجه چند سر

اکنون جوهر اصلی ترانسفورماتور می آید: توجه به خود.

این بر میزان مرتبط بودن یک کلمه خاص با سایر کلمات در جمله تمرکز می کند. به عنوان یک بردار توجه نشان داده می شود. برای هر کلمه، می‌توانیم یک بردار توجه ایجاد کنیم که رابطه متنی بین کلمات آن جمله را نشان می‌دهد.

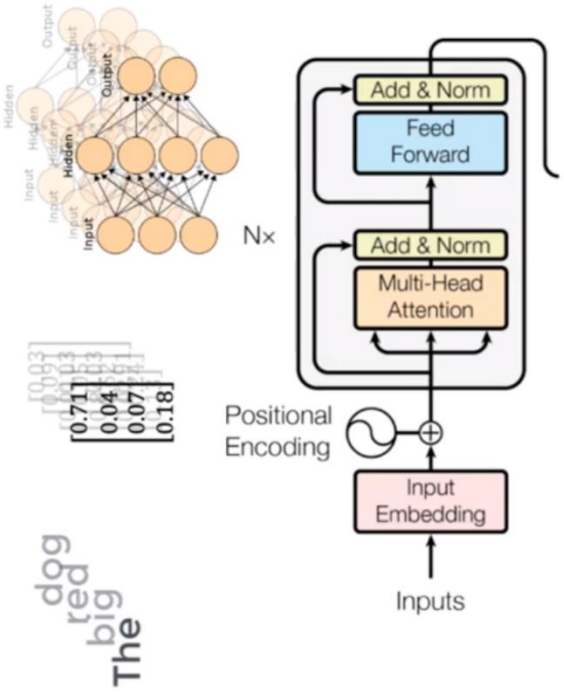
تنها مشکلی که اکنون وجود دارد این است که برای هر کلمه، ارزش خود را در جمله بسیار بیشتر می کند، اما ما می خواهیم تعامل آن را با سایر کلمات آن جمله بدانیم. بنابراین، ما بردارهای توجه متعدد را در هر کلمه تعیین می کنیم و یک میانگین وزنی برای محاسبه بردار توجه نهایی هر کلمه می گیریم.

از آنجایی که ما از بردارهای توجه چندگانه استفاده می کنیم، این فرآیند بلوک توجه چند سر نامیده می شود.

شبکه تغذیه به جلو

Now, the second step is the feed-forward neural network. A simple feed-forward neural network is applied to every attention vector to transform the attention vectors into a form that is acceptable to the next encoder or decoder layer.

در حال حاضر، مرحله دوم شبکه عصبی پیشخور است. یک شبکه عصبی پیش‌خور ساده برای هر بردار توجه اعمال می‌شود تا بردارهای توجه را به شکلی تبدیل کند که برای لایه رمزگذار یا رمزگشای بعدی قابل قبول باشد.

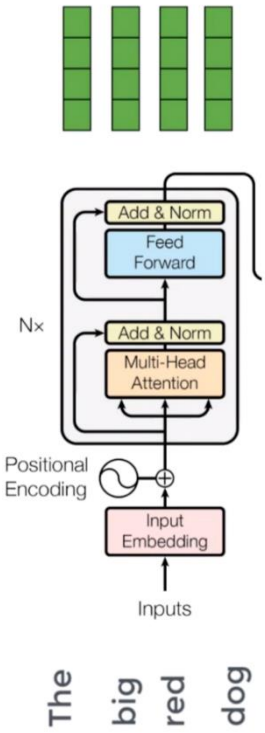


The feed-forward network accepts attention vectors one at a time. And the best thing here is, unlike the case of the RNN, each of these attention vectors is independent of one another. So, we can apply parallelization here, and that makes all the difference.

شبکه فید فوروارد بردارهای توجه را یکی یکی می پذیرد. و بهترین چیز در اینجا این است که برخلاف موردRNN ، هر یک از این بردارهای توجه مستقل از یکدیگر هستند. بنابراین، ما می توانیم موازی سازی را در اینجا اعمال کنیم، و این همه تفاوت را ایجاد می کند.

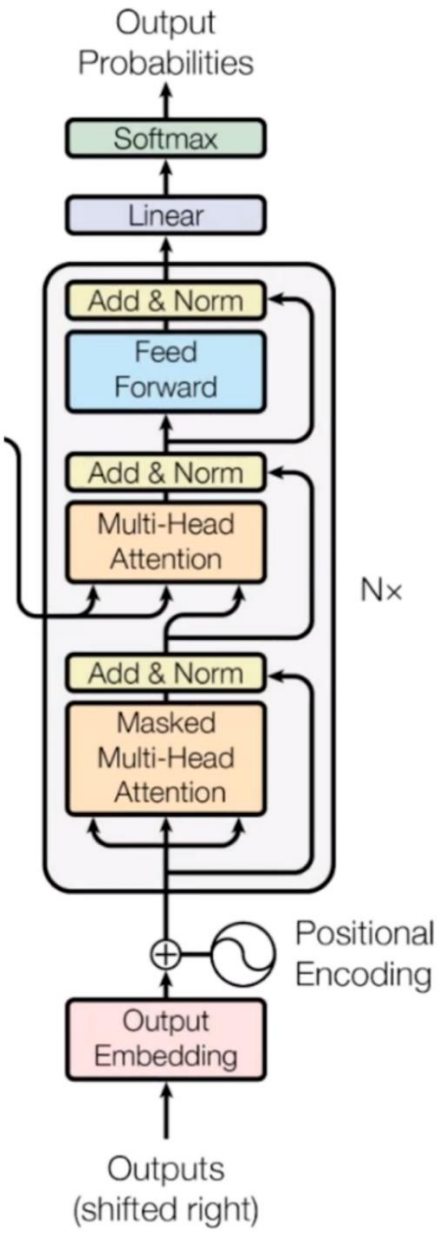
Now we can pass all the words at the same time into the encoder block and get the set of encoded vectors for every word simultaneously.

اکنون می‌توانیم همه کلمات را همزمان به بلوک رمزگذار منتقل کنیم و مجموعه بردارهای رمزگذاری شده را برای هر کلمه به طور همزمان دریافت کنیم.



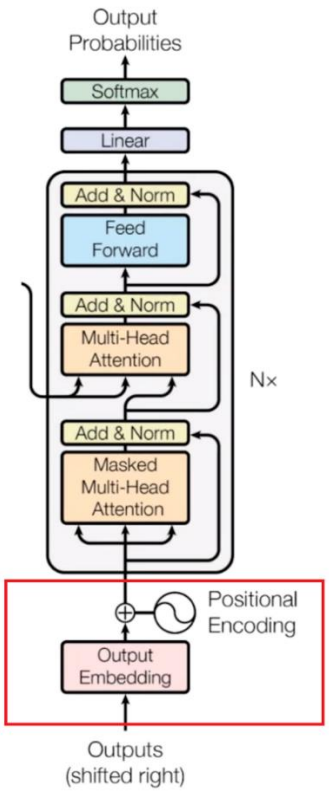
Decoder Block

بلوک رسیور



Now, if we're training a translator for English to French, for training, we need to give an English sentence along with its translated French version for the model to learn. So, our English sentences pass through encoder block, and French sentences pass through the decoder block.

حال، اگر برای آموزش یک مترجم انگلیسی به فرانسوی آموزش می‌دهیم، باید یک جمله انگلیسی به همراه نسخه فرانسوی ترجمه شده آن را ارائه کنیم تا مدل یاد بگیرد. بنابراین، جملات انگلیسی ما از بلوک رمزگذار عبور می کنند، و جملات فرانسوی از بلوک رمزگشا عبور می کنند.

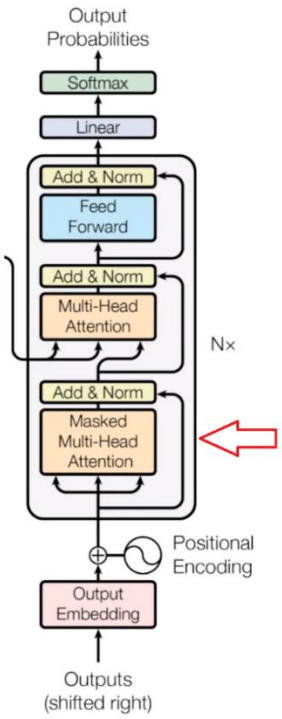


در ابتدا لایه embedding و قسمت رمزگذار موقعیتی را داریم که کلمات را به بردارهای مربوطه تبدیل می کند. این مشابه چیزی است که در قسمت رمزگذار مشاهده کردیم.

بخش توجه چند سر نقابدار

At first, we have the embedding layer and positional encoder part, which changes the words into respective vectors. This is similar to what we saw in the encoder part.

MASKED MULTI-HEAD ATTENTION PART



Now it will pass through the self-attention block, where attention vectors are generated for every word in the French sentences to represent how much each word is related to every word in the same sentence, just like we saw in the encoder part.

But this block is called the masked multi-head attention block, which I am going to explain in simple terms. First, we need to know how the learning mechanism works. When we provide an English word, it will be translated into its French version using previous results. It will then match and compare with the actual French translation that we fed into the decoder block. After comparing both, it will update its matrix value. This is how it will learn after several iterations.

What we observe is that we need to hide the next French word so that, at first, it will predict the next word itself using previous results without knowing the real translated word. For learning to take place, it would make no sense if it already knows the next French word. Therefore, we need to hide (or mask) it.

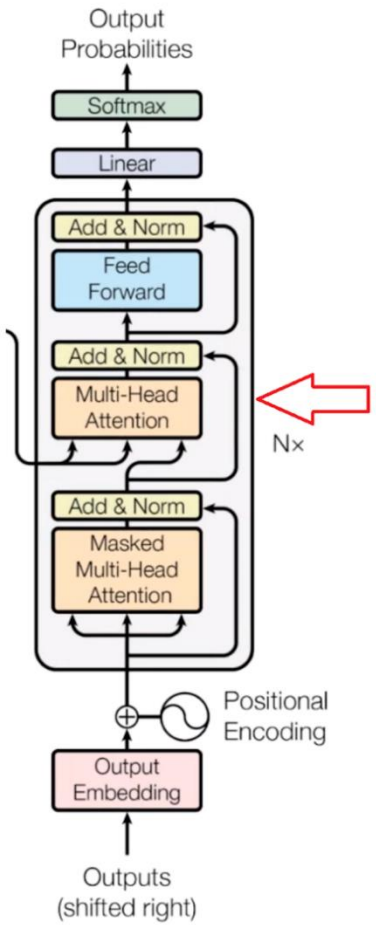
We can take any word from the English sentence, but we can only take the previous word of the French sentence for learning purposes. So, while performing parallelization with the matrix operation, we need to make sure that the matrix will mask the words appearing later by transforming them into zeroes so that the attention network can't use them.

اکنون از بلوک توجه به خود عبور می کند، جایی که بردارهای توجه برای هر کلمه در جملات فرانسوی ایجاد می شود تا نشان دهد که هر کلمه چقدر با هر کلمه در همان جمله مرتبط است، درست همانطور که در قسمت رمزگذار دیدیم.

اما این بلوک بلوک توجه چند سر ماسک شده نام دارد که قصد دارم آن را به زبان ساده توضیح دهم. ابتدا باید بدانیم مکانیسم یادگیری چگونه کار می کند. وقتی یک کلمه انگلیسی ارائه می کنیم، با استفاده از نتایج قبلی به نسخه فرانسوی آن ترجمه می شود. سپس با ترجمه فرانسوی واقعی که به بلوک رمزگشا وارد کردیم مطابقت و مقایسه می شود. پس از مقایسه هر دو، مقدار ماتریس خود را به روز می کند. به این ترتیب پس از چندین بار تکرار یاد می گیرد.

آنچه مشاهده می کنیم این است که باید کلمه فرانسوی بعدی را پنهان کنیم تا در ابتدا کلمه بعدی را با استفاده از نتایج قبلی بدون دانستن کلمه واقعی ترجمه شده پیش بینی کند. برای اینکه یادگیری اتفاق بیفتد، اگر از قبل کلمه فرانسوی بعدی را بداند، معنی ندارد. بنابراین، ما باید آن را پنهان کنیم (یا ماسک کنیم).

ما می توانیم هر کلمه ای را از جمله انگلیسی بگیریم، اما فقط می توانیم کلمه قبلی جمله فرانسوی را برای اهداف یادگیری استفاده کنیم. بنابراین، در حین انجام موازی سازی با عملیات ماتریس، باید مطمئن شویم که ماتریس کلماتی که بعدا ظاهر می شوند را با تبدیل آنها به صفر می پوشاند تا شبکه توجه نتواند از آنها استفاده کند.



Now, the resulting attention vectors from the previous layer and the vectors from the encoder block are passed into another multi-head attention block. This is where the results from the encoder block also come into the picture. In the diagram, the results from the encoder block also clearly come here. That's why it is called the encoder-decoder attention block.

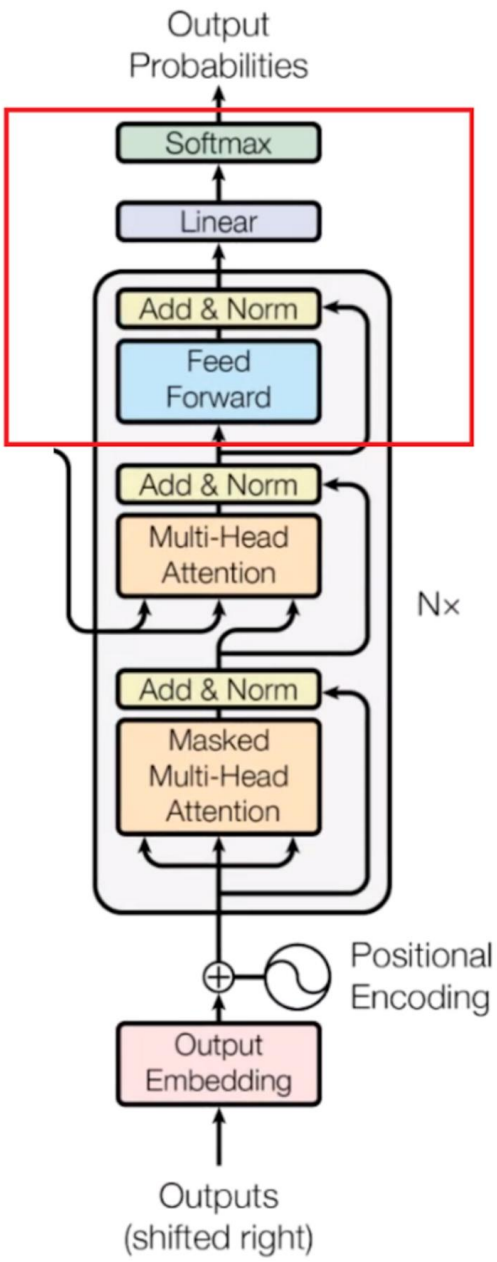
Since we have one vector of every word for each English and French sentence, this block actually does the mapping of English and French words and finds out the relation between them. So, this is the part where the main English to French word mapping happens.

The output of this block is attention vectors for every word in the English and French sentences. Each vector represents the relationship with other words in both languages.

اکنون، بردارهای توجه حاصل از لایه قبلی و بردارهای بلوک رمزگذار به بلوک توجه چند سر دیگری منتقل می شوند. در اینجاست که نتایج بلوک رمزگذار نیز به تصویر کشیده می شود. در نمودار، نتایج بلوک رمزگذار نیز به وضوح در اینجا آمده است. به همین دلیل است که به آن بلوک توجه رمزگذار - رمزگشا می گویند.

از آنجایی که برای هر جمله انگلیسی و فرانسوی یک بردار از هر کلمه داریم، این بلوک در واقع نگاشت کلمات انگلیسی و فرانسوی را انجام می دهد و رابطه بین آنها را پیدا می کند. بنابراین، این قسمتی است که نگاشت اصلی کلمات انگلیسی به فرانسوی در آن اتفاق می افتد.

خروجی این بلوک بردارهای توجه برای هر کلمه در جملات انگلیسی و فرانسوی است. هر بردار نشان دهنده رابطه با کلمات دیگر در هر دو زبان است.



Now, if we pass each attention vector into a feed-forward unit, it will make the output vectors into a form that is easily acceptable by another decoder block or a linear layer. A linear layer is another feed-forward layer that expands the dimensions into numbers of words in the French language after translation.

Now it is passed through a softmax layer that transforms the input into a probability distribution, which is human interpretable, and the resulting word is produced with the highest probability after translation.

حال، اگر هر بردار توجه را به یک واحد پیش‌خور منتقل کنیم، بردارهای خروجی را به شکلی تبدیل می‌کند که به راحتی توسط بلوک رمزگشای دیگر یا یک لایه خطی قابل قبول است. لایه خطی یکی دیگر از لایه‌های پیش‌خور است که بعد از ترجمه ابعاد را به تعداد کلمات در زبان فرانسوی گسترش می‌دهد.

اکنون از یک لایه softmax عبور داده می شود که ورودی را به یک توزیع احتمال تبدیل می کند که قابل تفسیر توسط انسان است و کلمه حاصل پس از ترجمه با بیشترین احتمال تولید می شود.

Here is an example from Google's AI blog. In the animation, the transformer starts by generating initial representations, or embeddings, for each word that are represented by the unfilled circles. Then, using self-attention, it aggregates information from all of the other words, generating a new representation per word informed by the entire context, represented by the filled balls. This step is then repeated multiple times in parallel for all words, successively generating new representations.

The decoder operates similarly, but generates one word at a time, from left to right. It attends not only to the other previously generated words but also to the final representations generated by the encoder.

The Takeaway

So, this is how the transformer works, and it is now the state-of-the-art technique in NLP. Its results, using a self-attention mechanism, are promising, and it also solves the parallelization issue. Even Google uses BERT, which uses a transformer to pre-train models for common NLP applications.

در اینجا نمونه ای از وبلاگ هوش مصنوعی گوگل آورده شده است. در انیمیشن، ترانسفورماتور با ایجاد نمایش های اولیه یا جاسازی ها برای هر کلمه که توسط دایره های پر نشده نمایش داده می شود، شروع می کند. سپس، با استفاده از توجه به خود، اطلاعات همه کلمات دیگر را جمع‌آوری می‌کند و به ازای هر کلمه یک نمایش جدید ایجاد می‌کند که توسط کل زمینه، که توسط توپ‌های پر شده نشان داده می‌شود. سپس این مرحله چندین بار به طور موازی برای همه کلمات تکرار می شود و به طور متوالی بازنمایی های جدیدی ایجاد می کند.

رمزگشا به طور مشابه عمل می کند، اما یک کلمه را در یک زمان، از چپ به راست تولید می کند. نه تنها به سایر کلمات تولید شده قبلی بلکه به نمایش های نهایی تولید شده توسط رمزگذار نیز توجه می کند.

غذای آماده

بنابراین، ترانسفورماتور اینگونه کار می کند و اکنون تکنیک پیشرفته در NLPاست. نتایج آن، با استفاده از مکانیزم توجه به خود، امیدوارکننده است، و همچنین مسئله موازی سازی را حل می کند. حتی گوگل از BERTاستفاده می‌کند که از یک ترانسفورماتور برای پیش‌آموزش مدل‌ها برای برنامه‌های رایج NLP استفاده می‌کند.