Why does the transformer do better than RNN and LSTM in long-range context dependencies?

Asked 2 years, 5 months ago Modified 8 months ago Viewed 52k times

48

24

I am reading the article How Transformers Work where the author writes

> Another problem with RNNs, and LSTMs, is that it's hard to parallelize the work for processing sentences, since you have to process word by word. Not only that but there is no model of **long and short-range dependencies**.

Why exactly does the transformer do better than RNN and LSTM in **long-range context dependencies**?

machine-learning natural-language-processing recurrent-neural-networks long-short-term-memory transformer

Share Improve this question Follow

edited Apr 7, 2020 at 16:08
nbro
34.9k 11 81 149

asked Apr 7, 2020 at 12:05
DRV
1,303 2 10 16

1 I think it's incorrect to say that LSTMs cannot capture long-range dependencies. Well, it depends on what you mean by "long-range". They certainly can capture *certain* long-range dependencies. Also, when the author of that article says "there is no model of long and short-range dependencies.", this probably answers the question. The transformer probably has a "model" (whatever the author means by that) that models long-range dependencies explicitly (whatever "explicitly" means). Honestly, I am currently not familiar with the details of the transformer, so I cannot provide a formal answer now. – nbro Apr 7, 2020 at 15:46

4 Answers

Sorted by: Highest score (default)

42

I'll list some bullet points of the main innovations introduced by transformers , followed by bullet points of the main characteristics of the other architectures you mentioned, so we can then

42 compared them.

Transformers

Transformers (Attention is all you need) were introduced in the context of machine translation with the purpose to avoid recursion in order to allow parallel computation (to reduce training time) and also to reduce drops in performance due to long dependencies. The main characteristics are:

- **Non sequential**: sentences are processed as a whole rather than word by word.
- **Self Attention**: this is the newly introduced 'unit' used to compute similarity scores between words in a sentence.
- **Positional embeddings**: another innovation introduced to replace recurrence. The idea is to use fixed or learned weights which encode information related to a specific position of a token in a sentence.

The first point is the main reason why transformer do not suffer from long dependency issues. The original transformers do not rely on past hidden states to capture dependencies with previous words. They instead process a sentence as a whole. That is why there is no risk to lose (or "forget") past information. Moreover, multi-head attention and positional embeddings both provide information about the relationship between different words.

RNN / LSTM

Recurrent neural networks and Long-short term memory models, for what concerns this question, are almost identical in their core properties:

- **Sequential processing**: sentences must be processed word by word.
- **Past information retained through past hidden states**: sequence to sequence models follow the Markov property: each state is assumed to be dependent only on the previously seen state.

The first property is the reason why RNN and LSTM can't be trained in parallel. In order to encode the second word in a sentence I need the previously computed hidden states of the first word, therefore I need to compute that first. The second property is a bit more subtle, but not hard to grasp conceptually. Information in RNN and LSTM are retained thanks to previously computed hidden states. The point is that the encoding of a specific word is retained only for the next time step, which means that the encoding of a word strongly affects only the representation of the next word, so its influence is quickly lost after a few time steps. LSTM (and also GruRNN) can boost a bit the dependency range they can learn thanks to a deeper processing of the hidden states through specific units (which comes with an increased number of parameters to train) but nevertheless the problem is inherently related to recursion. Another way in which people mitigated this problem is to use bi-directional models. These encode the same sentence from the start to end, and from the end to the start, allowing words at the end of a sentence to have stronger influence in the creation of the hidden representation. However, this is just a workaround rather than a real solution for very long dependencies.

CNN

Also convolutional neural networks are widely used in nlp since they are quite fast to train and effective with short texts. The way they tackle dependencies is by applying different kernels to the same sentence, and indeed since their first application to text (Convolutional Neural Networks for Sentence Classification) they were implement as multichannel CNN. Why do different kernels allow to learn dependencies? Because a kernel of size 2 for example would learn relationships between pairs of words, a kernel of size 3 would capture relationships between triplets of words and so on. The evident problem here is that the number of different kernels required to capture dependencies among all possible combinations of words in a sentence would be enormous and unpractical because of the exponential growing number of combinations when increasing the maximum length size of input sentences.

To summarize, Transformers are better than all the other architectures because they totally avoid recursion, by processing sentences as a whole and by learning relationships between words thanks to multi-head attention mechanisms and positional embeddings. Nevertheless, it must be pointed out that also transformers can capture only dependencies within the fixed input size used to train them, i.e. if I use as a maximum sentence size 50, the model will not be able to capture dependencies between the first word of a sentence and words that occur more than 50 words later, like in another paragraph. New transformers like Transformer-XL tries to overcome exactly this issue, by kinda re-introducing recursion by storing hidden states of already encoded sentences to leverage them in the subsequent encoding of the next sentences.

Share Improve this answer Follow

edited Jan 26 at 9:18
Hymns For Disco
103 4

answered Apr 7, 2020 at 16:12
Edoardo Guerriero
4,273 8 22

4 It's not clear from this answer what you mean when you say that "transformers process sentences as a whole". Do you mean that first you create word embeddings that represent whole sentences rather than words? I think that you should have mentioned the encoder-decoder architecture with LSTMs, because it seems that this "processing sentences as a whole" already existed in other machine translation architectures with LSTMs (like neural machine translation and stuff like that), so that would not be a novel thing. – nbro Sep 17, 2020 at 13:12

9

Let's start with RNN. A well known problem is vanishin/exploding gradients, which means that the model is biased by most recent inputs in the sequence, or in other words, older inputs have practically no effect in the output at the current step.

LSTMs/GRUs mainly try to solve this problem, by including a separate memory (cell) and/or extra gates to learn when to let go of past/current information. Check these series of lectures for more in-depth discussion. Also check the interactive parts of this article for some intuitive understanding of dependency on past elements.

Now, given all this, information from past steps *still* goes through a sequence of computations and we're relying on these new gate/memory mechanisms to pass information from old steps to the current one.

One major advantage of the transformer architecture, is that at each step we have **direct** access to all the other steps (self-attention), which practically leaves no room for information loss, as far as message passing is concerned. On top of that, we can look at both future and past elements at the same time, which also brings the benefit of bidirectional RNNs, without the 2x computation needed. And of course, all this happens in parallel (non-recurrent), which makes both training/inference much faster.

The self-attention with every other token in the input means that the processing will be in the order of $\mathcal{O}(N^2)$ (glossing over details), which means that it's going to be costly to apply transformers on long sequences, compared to RNNs. That's probably one area that RNNs still have an advantage over transformers.

Share Improve this answer Follow

edited Apr 7, 2020 at 16:25
nbro
34.9k 11 81 149

answered Apr 7, 2020 at 16:20
olix20
276 1 2

Why can you look at both the future and past elements at the same time without 2x computation? – nbro Apr 7, 2020 at 16:26

3 A bidirectional RNN has a forward and a backward RNN, which means we process the whole sequence 2 times. It's basically a hack so that we have some way to look at the future using the backward RNN. In transformers, this is done in "one" operation, which is just calculating the result of `softmax(QK) x V` – olix20 Apr 7, 2020 at 16:35

1

First: RNN is one part of the Neural Network family for processing sequential data. The way in which RNN is able to store information from the past is to loop in its architecture, which automatically keeps information from the past stored. Second: sltm / gru is a component of regulating the flow of information referred to as the gate and GRU has 2 gates, namely reset gate and gate update. If we want to make a decision to eat like the analogy above, resetting the gate on the GRU will determine how to combine new inputs with past information, and update the gate, will determine how much past information should be kept. source : https://link.springer.com/article/10.1007/s00500-019-04281-z

Share Improve this answer Follow

answered Apr 21, 2020 at 14:25
lutfizain
21 1

3 Hi @Lutfizain, welcome to AI stack exchange! Please see the tour and look around to see how this site

works. I think you aren't answer the question, OP is asking why transformer (kind of NN architecture) performs better than RNN/LSTM on that context, and you didn't mention it – malioboro Apr 22, 2020 at 0:41

0

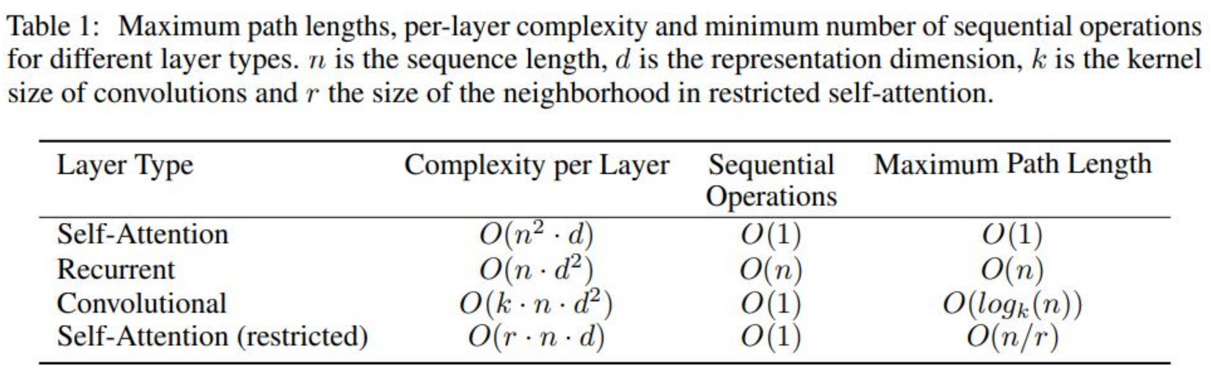

Table 1: Maximum path lengths, per-layer complexity and minimum number of sequential operations for different layer types. n is the sequence length, d is the representation dimension, k is the kernel size of convolutions and r the size of the neighborhood in restricted self-attention.

Layer Type	Complexity per Layer	Sequential Operations	Maximum Path Length
Self-Attention	$O(n^2 \cdot d)$	$O(1)$	$O(1)$
Recurrent	$O(n \cdot d^2)$	$O(n)$	$O(n)$
Convolutional	$O(k \cdot n \cdot d^2)$	$O(1)$	$O(log_k(n))$
Self-Attention (restricted)	$O(r \cdot n \cdot d)$	$O(1)$	$O(n/r)$

table is from arxiv:1706.03762 Attention Is All You Need

It's because of the path length. If you have a sequence of length n. Then a transformer will have access to each element with O(1) sequential operations where a recurrent neural network will need at most O(n) sequential operations to access an element.
Very long sequences gives you problem with exploding and vanishing gradients because of the chain rule in backprop.
Transformers don't have this problem as the distance to each element in the sequence is always O(1) sequential operations away. (as it selects the right element instead of trying to "remember" it)

(arxiv:2103.05487) UnICORNN: A recurrent model for learning very long time dependencies

"The design of RNNs that can accurately handle sequential inputs with long-time dependencies is very challenging.
This is largely on account of the exploding and vanishing gradient problem (EVGP)."

Share Improve this answer Follow

answered Jan 26 at 11:24
hal9000
349 2 7