How Transformers Work

The Neural Network used by OpenAI and DeepMind

Transformers are a type of neural network architecture that have been gaining popularity. Transformers were recently used by OpenAI in their language models, and also used recently by DeepMind for AlphaStar — their program to defeat a top professional Starcraft player.

Transformers were developed to solve the problem of **sequence transduction,** or **neural machine translation.** That means any task that transforms an input sequence to an output sequence. This includes speech recognition, text-to-speech transformation, etc.

This image represents sequence transduction. The input is represented in green, the model is represented in blue, and the output is represented in purple.
For models to perform **sequence transduction,** it is necessary to have some sort of memory. For example let's say that we are translating the following sentence to another language (French):

> *The Transformers are a Japanese [[hardcore punk]] band. The band was formed in 1968, during the height of Japanese music history*

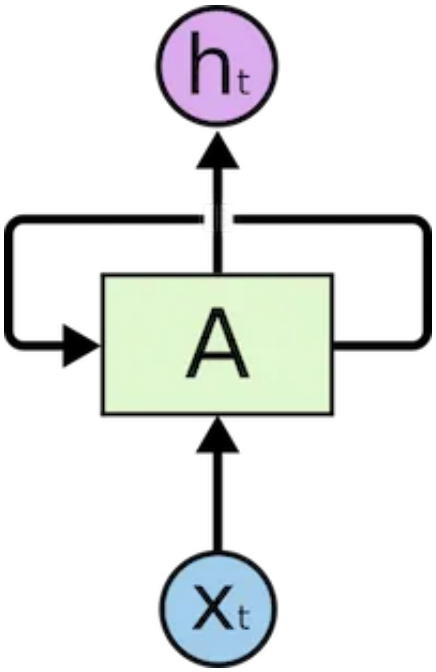
In this example, the word "the band" in the second sentence refers to the band "The Transformers" introduced in the first sentence. When you read about the band in the second sentence, you know that it is referencing to the "The Transformers" band. That may be important for translation. There are many examples, where words in some sentences refer to words in previous sentences.

For translating sentences like that, a model needs to figure out these sort of dependencies and connections. Recurrent Neural Networks (RNNs) and Convolutional Neural Networks (CNNs) have been used to deal with this problem because of their properties. Let's go over these two architectures and their drawbacks.

Recurrent Neural Networks

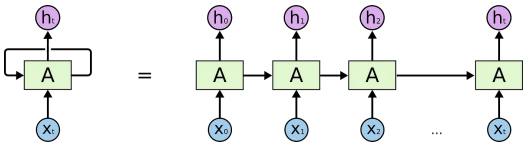
Recurrent Neural Networks have loops in them, allowing information to persist.

The input is represented as x_t



In the figure, we see part of the neural network, **A,** processing some input x_t and outputs h_t. A loop allows information to be passed from one step to the next.

The loops can be thought in a different way. A Recurrent Neural Network can be thought of as multiple copies of the same network, **A**, each network passing a message to a successor. Consider what happens if we unroll the loop:



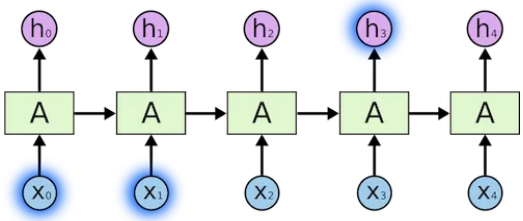
An unrolled recurrent neural network

This chain-like nature shows that recurrent neural networks are clearly related to sequences and lists. In that way, if we want to translate some text, we can set each input as the word in that text. The Recurrent Neural Network passes the information of the previous words to the next network that can use and process that information.
The following picture shows how usually a sequence to sequence model works using Recurrent Neural Networks. Each word is processed separately, and the resulting sentence is generated by passing a hidden state to the decoding stage that, then, generates the output.

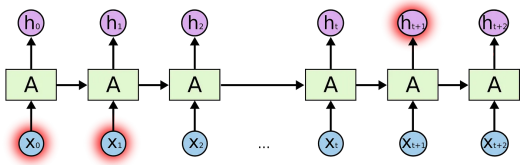
The Problem of Long-Term Dependencies

Consider a language model that is trying to predict the next word based on the previous ones. If we are trying to predict the next word of the sentence **“the clouds in the sky”**, we don’t need further context. It’s pretty obvious that the next word is go-

ing to be **sky.**
In this case where the difference between the relevant information and the place that is needed is small, RNNs can learn to use past information and figure out what is the next word for this sentence.



But there are cases where we need more context. For example, let's say that you are trying to predict the last word of the text: **"I grew up in France… I speak fluent …".** Recent information suggests that the next word is probably a language, but if we want to narrow down which language, we need context of France, that is further back in the text.



RNNs become very ineffective when the gap between the relevant information and the point where it is needed become very large. That is due to the fact that the information is passed at each step and the longer the chain is, the more probable the information is lost along the chain.

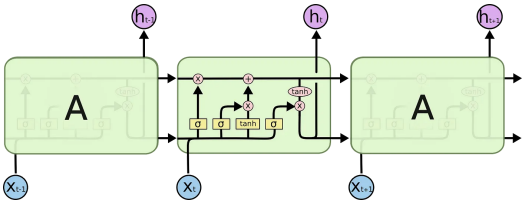
In theory, RNNs could learn this long-term dependencies. In practice, they don't seem to learn them. LSTM, a special type of RNN, tries to solve this kind of problem.

Long-Short Term Memory (LSTM)

When arranging one's calendar for the day, we prioritize our appointments. If there is anything important, we can cancel some of the meetings and accommodate what is important.

RNNs don't do that. Whenever it adds new information, it transforms existing information completely by applying a function. The entire information is modified, and there is no consideration of what is important and what is not.

LSTMs make small modifications to the information by multiplications and additions. With LSTMs, the information flows through a mechanism known as cell states. In this way, LSTMs can selectively remember or forget things that are important and not so important.
Internally, a LSTM looks like the following:



Each cell takes as inputs **x_t** (a word in the case of a sentence to sentence translation), the **previous cell state** and the **output of the previous cell**. It manipulates these inputs and based on them, it generates a new cell state, and an output. I won't go into detail on the mechanics of each cell.

With a cell state, the information in a sentence that is important for translating a word may be passed from one word to another, when translating.

The Problem with LSTMs

The same problem that happens to RNNs generally, happen with LSTMs, i.e. when sentences are too long LSTMs still don't do too well. The reason for that is that the probability of keeping the context from a word that is far away from the current word being processed decreases exponentially with the distance from it.
That means that when sentences are long, the model often forgets the content of distant positions in the sequence. Another problem with RNNs, and LSTMs, is that it's hard to parallelize the work for processing sentences, since you are have to process word by word. Not only that but there is no model of long and short range dependencies. To summarize, LSTMs and RNNs present 3 problems:

– Sequential computation inhibits parallelization
– No explicit modeling of long and short range dependencies
– "Distance" between positions is linear

Attention

To solve some of these problems, researchers created a technique for paying attention to specific words.

When translating a sentence, I pay special attention to the word I'm presently translating. When I'm transcribing an audio recording, I listen carefully to the segment I'm actively writing down. And if you ask me to describe the room I'm sitting in, I'll glance around at the objects I'm describing as I do so.

Neural networks can achieve this same behavior using ***attention***, focusing on part of a subset of the information they are given. For example, an RNN can attend over the output of another RNN. At every time step, it focuses on different positions in the other RNN.
To solve these problems, **Attention** is a technique that is used in a neural network. For RNNs, instead of only encoding the whole sentence in a hidden state, each word has a corresponding hidden state that is passed all the way to the decoding stage. Then, the hidden states are used at each step of the RNN to decode. The following gif shows how that happens.

The green step is called the **encoding stage** and the purple step is the **decoding stage.**

Convolutional Neural Networks

Convolutional Neural Networks help solve these problems. With them we can

- Trivial to parallelize (per layer)
- Exploits local dependencies
- Distance between positions is logarithmic

Some of the most popular neural networks for sequence transduction, Wavenet and Bytenet, are Convolutional Neural Networks.

Wavenet, model is a Convolutional Neural Network (CNN).

The reason why Convolutional Neural Networks can work in parallel, is that each word on the input can be processed at the same time and does not necessarily depend on the previous words to be translated. Not only that, but the “distance” between the output word and any input for a CNN is in the order of **log(N)** — that is the size of the height of the tree generated from the output to the input (you can see it on the GIF above. That is much better than the distance of the output of a RNN and an input, which is on the order of **N.**

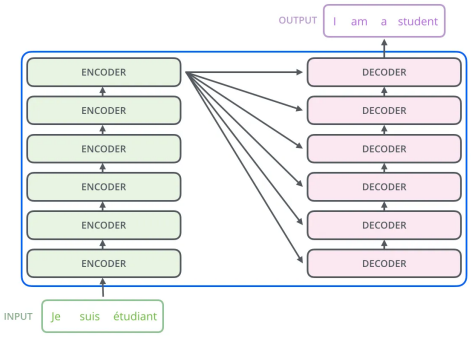
The problem is that Convolutional Neural Networks do not necessarily help with the problem of figuring out the problem of dependencies when translating sentences. That’s why **Transformers** were created, they are a combination of both CNNs with attention.

Transformers

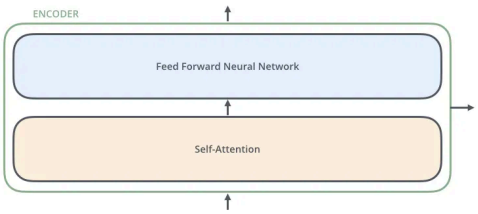
To solve the problem of parallelization, Transformers try to solve the problem by using Convolutional Neural Networks together with **attention models.** Attention boosts the speed of how fast the model can translate from one sequence to another.

Let's take a look at how **Transformer** works. Transformer is a model that uses **attention** to boost the speed. More specifically, it uses **self-attention.**

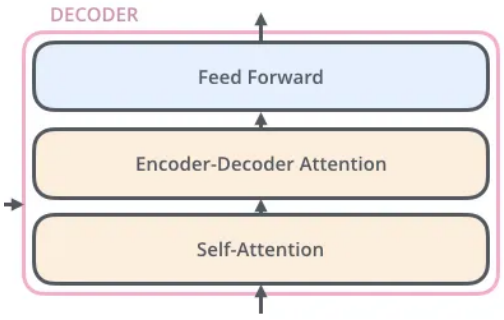
Internally, the Transformer has a similar kind of architecture as the previous models above. But the Transformer consists of six encoders and six decoders.



Each encoder is very similar to each other. All encoders have the same architecture. Decoders share the same property, i.e. they are also very similar to each other. Each encoder consists of two layers: **Self-attention** and a feed Forward Neural Network.



The encoder's inputs first flow through a **self-attention** layer. It helps the encoder look at other words in the input sentence as it encodes a specific word. The decoder has both those layers, but between them is an attention layer that helps the decoder focus on relevant parts of the input sentence.



Positional Encoding

Another important step on the Transformer is to add positional encoding when encoding each word. Encoding the position of each word is relevant, since the position of each word is relevant to the translation.

Bibliography

1. The Unreasonable Effectiveness of Recurrent Neural Networks
2. Understanding LSTM Networks
3. Visualizing A Neural Machine Translation Model
4. The Illustrated Transformer
5. The Transformer — Attention is all you need
6. The Annotated Transformer
7. Attention is all you need attentional neural network models
8. Self-Attention For Generative Models
9. OpenAI GPT-2: Understanding Language Generation through Visualization
10. WaveNet: A Generative Model for Raw Audio
11. How Transformers Work