Attention is all you need

**Introduction**

Important note on recurrent neural networks, as they state that parallelization is precluded (not possible), because of the sequential nature of recurrent neural networks:

“This inherently sequential nature precludes parallelization within training examples, which becomes critical at longer sequence lengths, as memory constraints limit batching across examples.”

The transformer architecture is said to *eschew* recurrentness in the language model, meaning that they will completely avoid recurrentness in the trained model. This therefore allows for better parallelization, and therefore faster training time.

“In this work we propose the Transformer, a model architecture eschewing recurrence and instead relying entirely on an attention mechanism to draw global dependencies between input and output. The Transformer allows for significantly more parallelization and can reach a new state of the art in translation quality after being trained for as little as twelve hours on eight P100 GPUs”

QUESTION: what part of recurrent neural networks makes it hard for parallelization training?

Is parallelization understood correctly, as a method of using several GPU’s to work together, to train the data, instead of only having a singular GPU. Meaning that the transformer learns approximately 8 times faster, as it uses 8 GPU’s.

**Background**

Feel like this is a core part of what transformers is, but I’m not sure what it means.

“The goal of reducing sequential computation also forms the foundation of the Extended Neural GPU [16], ByteNet [18] and ConvS2S [9], all of which use convolutional neural networks as basic building block, computing hidden representations in parallel for all input and output positions.”

Specifically, the part “computing hidden representations in parallel for all input and output positions.”

QUESTION: is this the correct thought pattern to have, the gist of what I am reading, is that they are treating text processing as a space complexity problem instead of a time complexity problem, meaning that they look at the entirety of the text simultaneously, as we would with a conv. NN. Which therefore enables connections to being made between different words in constant time, instead of connections being harder to reach, based on how far two-word connections is from each other in the sentence.

**Multi-headed Attention:**

A way of accelerating the speed at which to arbitrary words can be connected in a string, without multiheaded attention the time is constant for the attention mechanism.

Self-attention is also sometimes called **intra-attention.**

The following quote makes it sound a lot, like the attention model is as huge a significance as RNN’s and Convolutions. While also stating that this design is nothing like the others. Therefore, not some alternative RNN or Convolution method.

“*To the best of our knowledge, however, the Transformer is the first transduction model relying entirely on self-attention to compute representations of its input and output without using sequencealigned RNNs or convolution*”

**The part with the transformer model**

**Diagram

Description automatically generated**

**Encoder:**

*“The decoder is also composed of a stack of N = 6 identical layers. In addition to the two sub-layers in each encoder layer, the decoder inserts a third sub-layer, which performs multi-head attention over the output of the encoder stack. Similar to the encoder, we employ residual connections around each of the sub-layers, followed by layer normalization”*

Each stack is based on two sublayers. One which is the multi-headed attention mechanism and the other which is a feed forward network. Feedforward meaning the most basic form of hidden layer in the neural network.

Another thing that can be seen, is that for both the multi-headed self-attention layer, and the feed forward layer, a residual connection is used, meaning that the input is added into the output, making it easier for the parameters of the layers to converge into the correct parameters. These values are added and then normalized in the (“Add & norm”) part of the layer.

The reason for the normalization, is to make sure the input for the next layers, is not completely blown out of proportions by the addition statement.

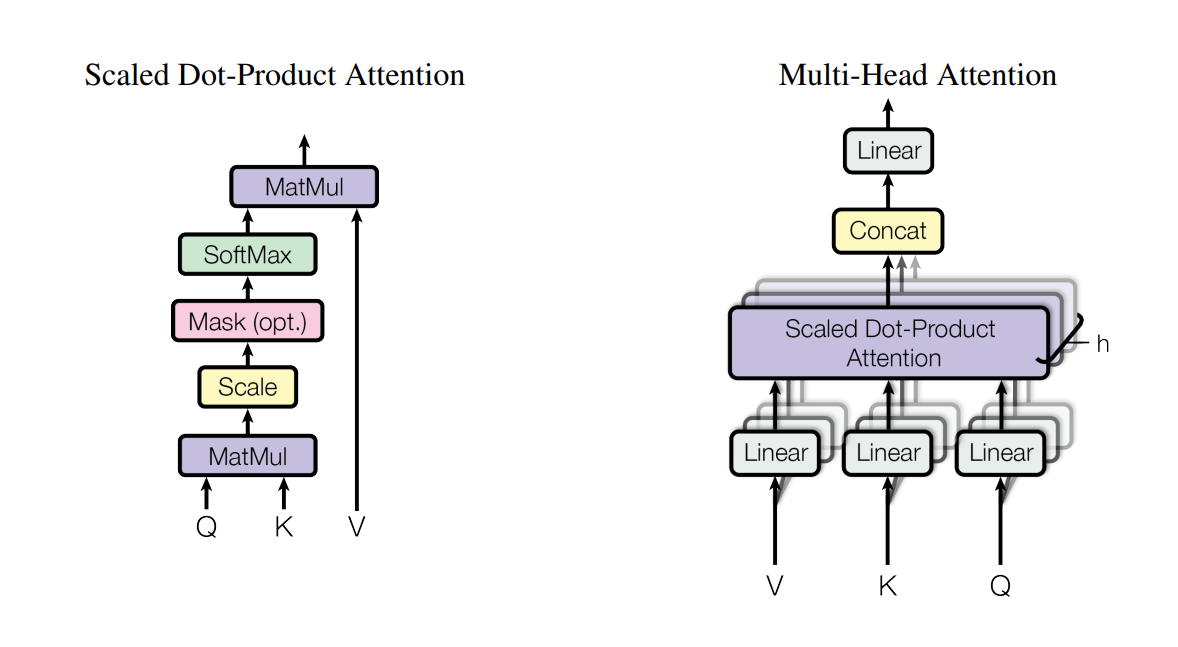
Though I am not completely sure how the normalization is made? I know how normalization works for a single list of values, but how does normalization work for several lists of values?

**Decoder:**

I am much more unsure how the decode works, compared to the encoder. The Decoder gits an output text, and the encoder output as its input, I am already confused here, as to where the output came from.

I get the idea, that the decoder attention mechanism is masked, such to make sure words don’t attend subsequent words in the list, but the part of all the outputs being shifted to the right is weird?

**Attention mechanism:**

****

You can somehow change the resolution of the attention mechanism by altering the size of the queries and keys dimensionality called dk. this phenomenon is used in the linear projection step of the multiheaded attention mechanism, to both use the attention mechanism on different representation subspaces, and scaling the queries and keys dimensionalities such to optimize computational power.

I do not understand why a fixed sinusoidal function can be used for positional embedding of words.

My best quess, is that it injects positional unique values to each word embedding, which the neural network will learn to understand, and process. But if uniqueness for each position is the key, why is a periodic function which repeats itself.

How does it calculate the first output word? With the first output word, there should’nt be any outputs that the decoder can use.

**Why self-attention**

“One is the total computational complexity per layer”

Why is this the case? The slowest part of the transformer model is the feed forward layers. I know that convolutional layers are faster, so its probably in comparison to recurrent layers. But why are recurrent layers slower than feedforward layers?

**A Gentle Introduction to Positional Encoding in Transformer Models, Part 1**

[A Gentle Introduction to Positional Encoding in Transformer Models, Part 1 - MachineLearningMastery.com](https://machinelearningmastery.com/a-gentle-introduction-to-positional-encoding-in-transformer-models-part-1/)

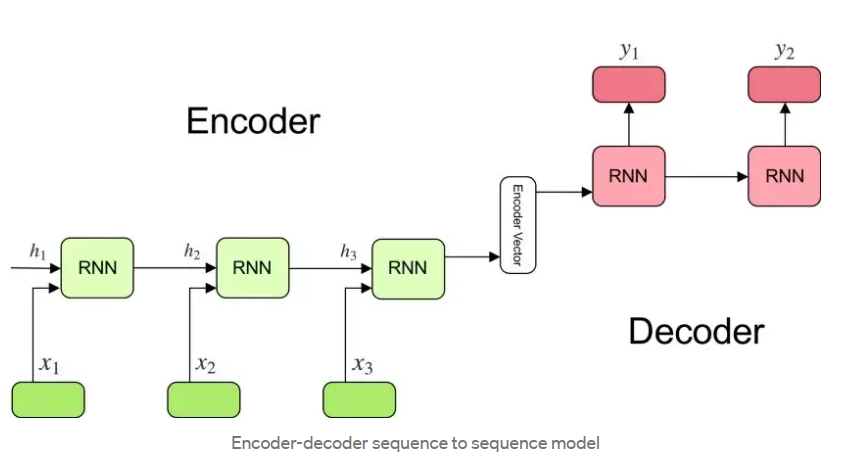
Instead of using a single value which can become too large, a vector of values are used, where the size of the sentence and the position itself is used to create a unique positional embedding vector value.

**Understanding encoder-Decoder sequence to sequence model**

[Understanding Encoder-Decoder Sequence to Sequence Model | by Simeon Kostadinov | Towards Data Science](https://towardsdatascience.com/understanding-encoder-decoder-sequence-to-sequence-model-679e04af4346)

In general, I would like to have a better understanding of the encoder decoder architecture:

As I have understood, a text is encoded in the encoder part, which then creates a vector of information. This vector will serve as the input for the decoder part, which is used to generate a response text. This is used, as the



Based on the following text:

*“For example, translating “What are you doing today?” from English to Chinese has input of 5 words and output of 7 symbols (今天你在做什麼？). Clearly, we can’t use a regular LSTM network to map each word from the English sentence to the Chinese sentence*”

It is thought, that a normal LSTM model will always have the same size output, as input. This is the primary reason as to why the encoder decoder model is better, as it enables arbitrary input to arbitrary output. Though I do not understand what part of the sequence makes this possible.

# Jannik blog

Seems like every word is passed into the encoder layer separately, but all the layers are the same for each word, making it extremely parralelizasionable.

Each word is passed through the encoder by itself which gives a representation of the singular word by itself in the other end. Each word is passed through the same way, making it look like many words are passed through.

They are not as intertwined as a thought they would be, but they are still one big lump in matrix form which is passed through together, they just don’t interact because the laws of matrix multiplication do not allow it.