CSC413/2516 Lecture 8:
Transformers and Autoregressive models

Bo Wang

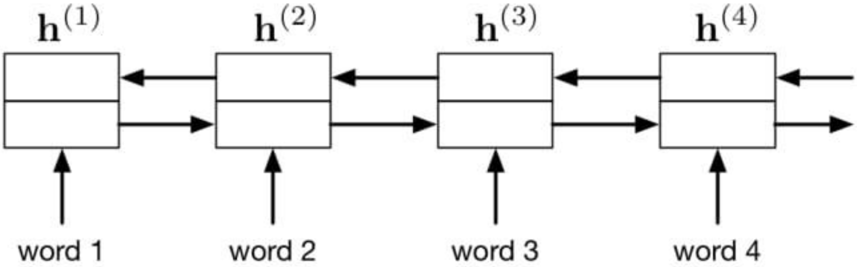
a.k.a. Attention is All You Need!
a.k.a. Good bye, RNN!

Overview

- We have seen a few RNN-based sequence prediction models.
- It is still challenging to generate long sequences, when the decoders only has access to the final hidden states from the encoder.
  - Machine translation: it's hard to summarize long sentences in a single vector, so let's allow the decoder peek at the input.
  - Vision: have a network glance at one part of an image at a time, so that we can understand what information it's using
- We also introduced attention coupled with RNN that drastically improves the performance on the long sequences.

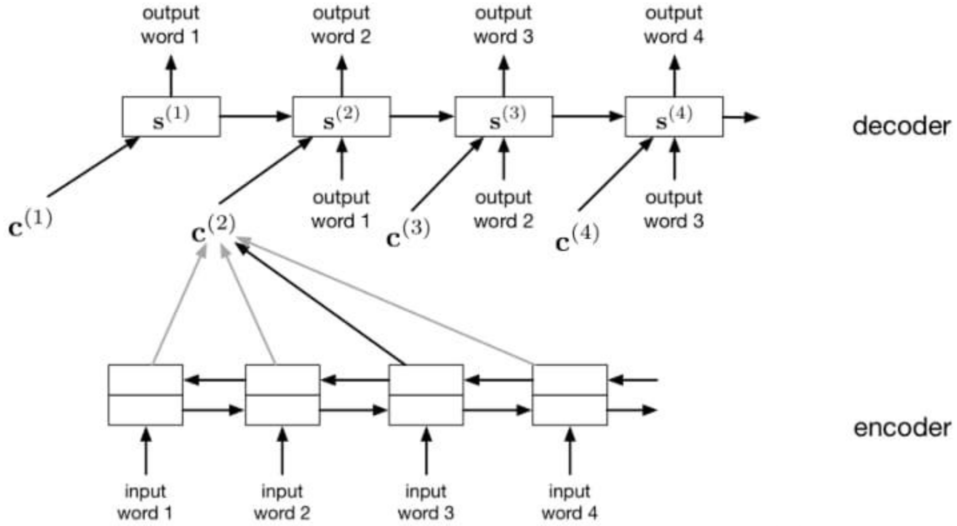
Overview: Attention-Based Machine Translation

- The model has both an encoder and a decoder. The encoder computes an annotation of each word in the input.
- It takes the form of a bidirectional RNN. This just means we have an RNN that runs forwards and an RNN that runs backwards, and we concatenate their hidden vectors.
  - The idea: information earlier or later in the sentence can help disambiguate a word, so we need both directions.
  - The RNN uses an LSTM-like architecture called gated recurrent units.



Overview: Attention-Based Machine Translation

- The decoder network is also an RNN. Like the encoder/decoder translation model, it makes predictions one word at a time, and its predictions are fed back in as inputs.
- The difference is that it also receives a context vector $\mathbf{c}^{(t)}$ at each time step, which is computed by attending to the inputs.



Overview: Attention-Based Machine Translation

- The context vector is computed as a weighted average of the encoder's annotations.

$$\mathbf{c}^{(i)} = \sum_j \alpha_{ij} h^{(j)}$$

- The attention weights are computed as a softmax, where the inputs depend on the annotation and the decoder's state:

$$\alpha_{ij} = \frac{\exp(\tilde{\alpha}_{ij})}{\sum_{j'} \exp(\tilde{\alpha}_{ij'})}$$

$$\tilde{\alpha}_{ij} = f(\mathbf{s}^{(i-1)}, \mathbf{h}^{(j)})$$

- Note that the attention function, f depends on the annotation vector, rather than the position in the sentence. This means it's a form of content-based addressing.
    - My language model tells me the next word should be an adjective. Find me an adjective in the input.

Limitations of Attention in RNN

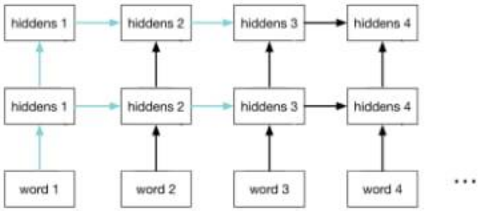
1. It is hard to generalize to long sequences.

2. It is difficult to parallel the computing.

3. It does not fully exploit the contextual information.

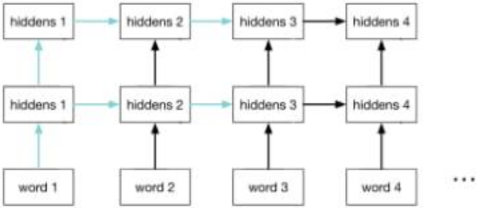
Attention is All You Need (Transformers)

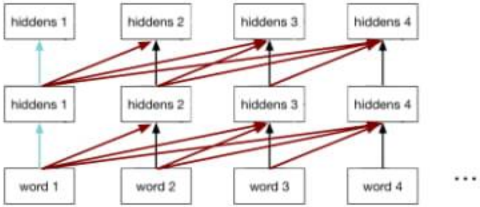
- We would like our model to have access to the entire history at the hidden layers.
- Previously we achieve this by having the recurrent connections.



Attention is All You Need (Transformers)

- We would like our model to have access to the entire history at the hidden layers.
- Previously we achieve this by having the recurrent connections.





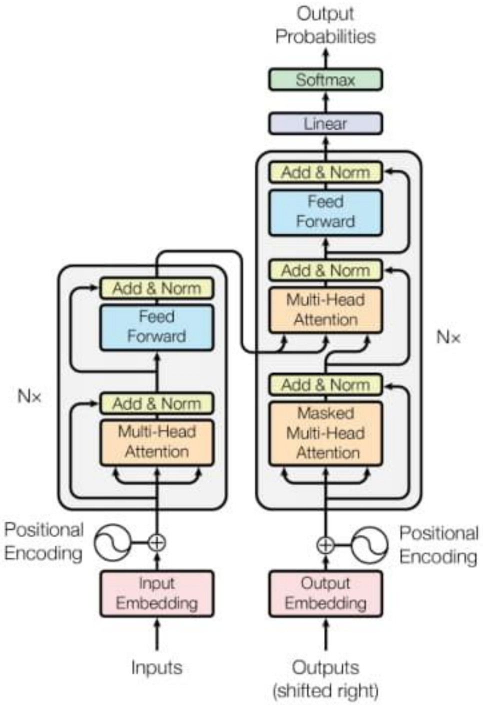
- **Core idea**: use attention to aggregate the context information by attending to one or a few important inputs from the past history.

Attention is All You Need

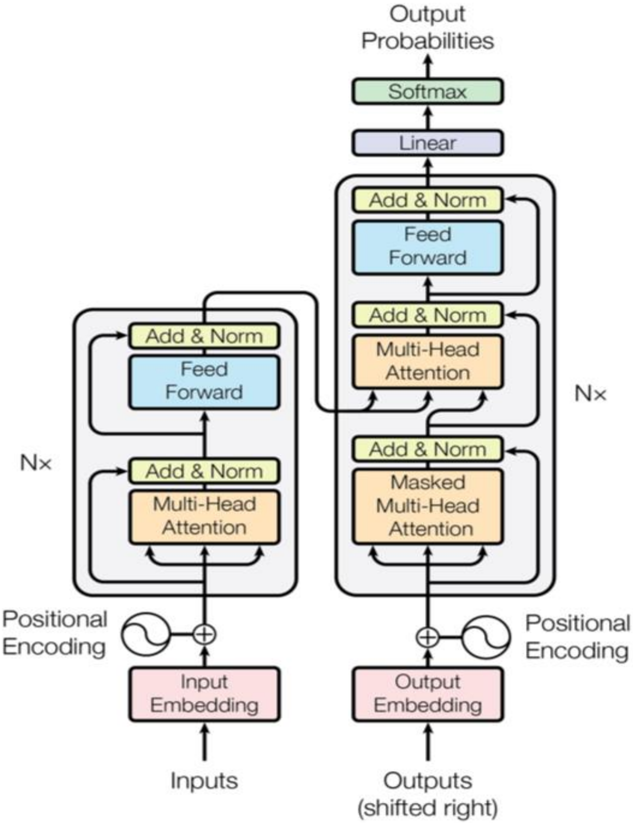
- We will now study a very successful neural network architecture for machine translation in the last few years:

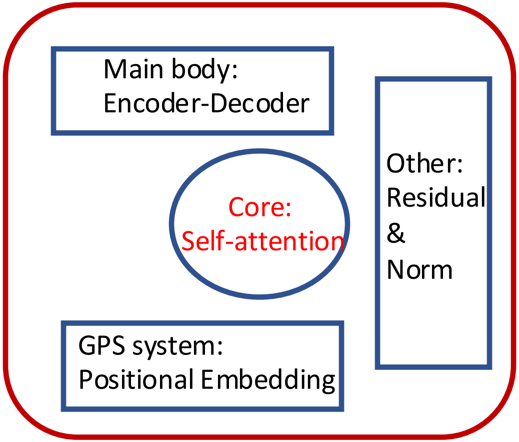
  Vaswani, Ashish, et al. "Attention is all you need." Advances in Neural Information Processing Systems. 2017.

- "Transformer" has a encoder-decoder architecture similar to the previous sequence-to-sequence RNN models.
  - except all the recurrent connections are replaced by the attention modules.



Disassemble the Transformers





Main body: Encoder-Decoder

Core: Self-attention

Other: Residual & Norm

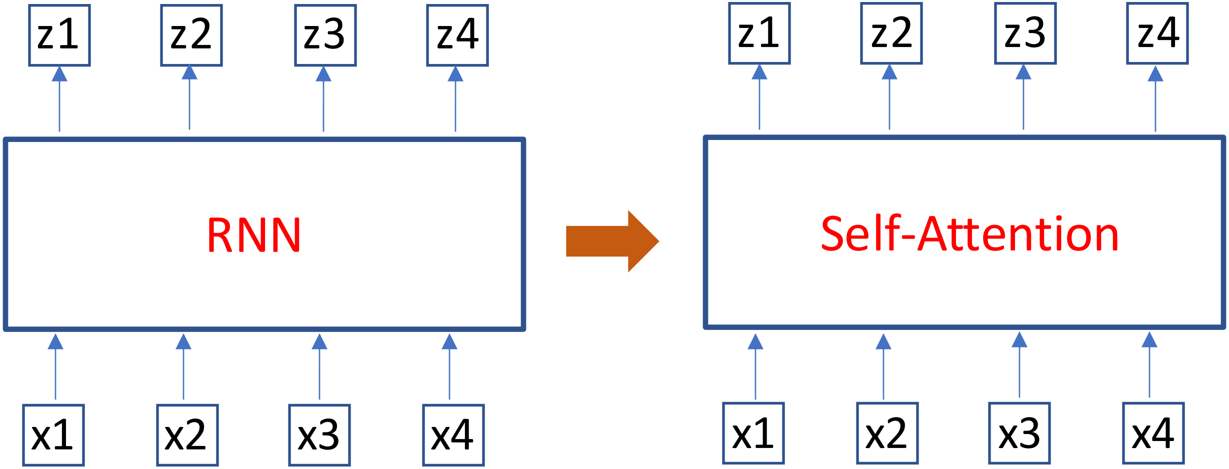
GPS system: Positional Embedding

The core of Transformers: Self-attention layer

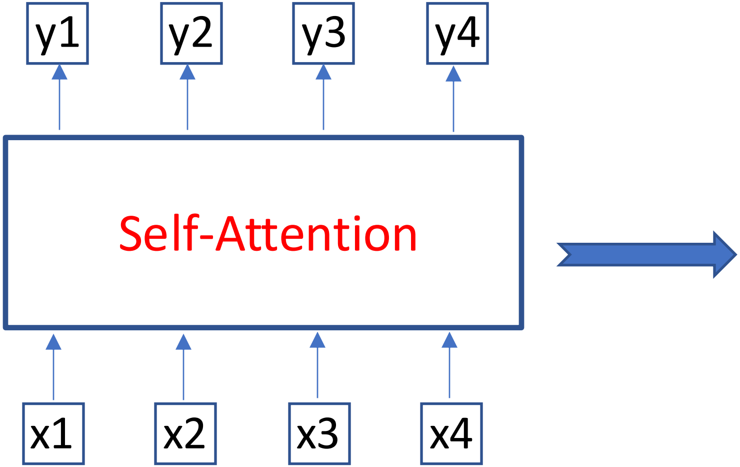
Sequence-to-Sequence Modeling

Key Difference:

1. **z** is obtained based on the whole input sequence
2. **z** can be computed parallelly



The core of Transformers: Self-attention layer



Important Concepts:

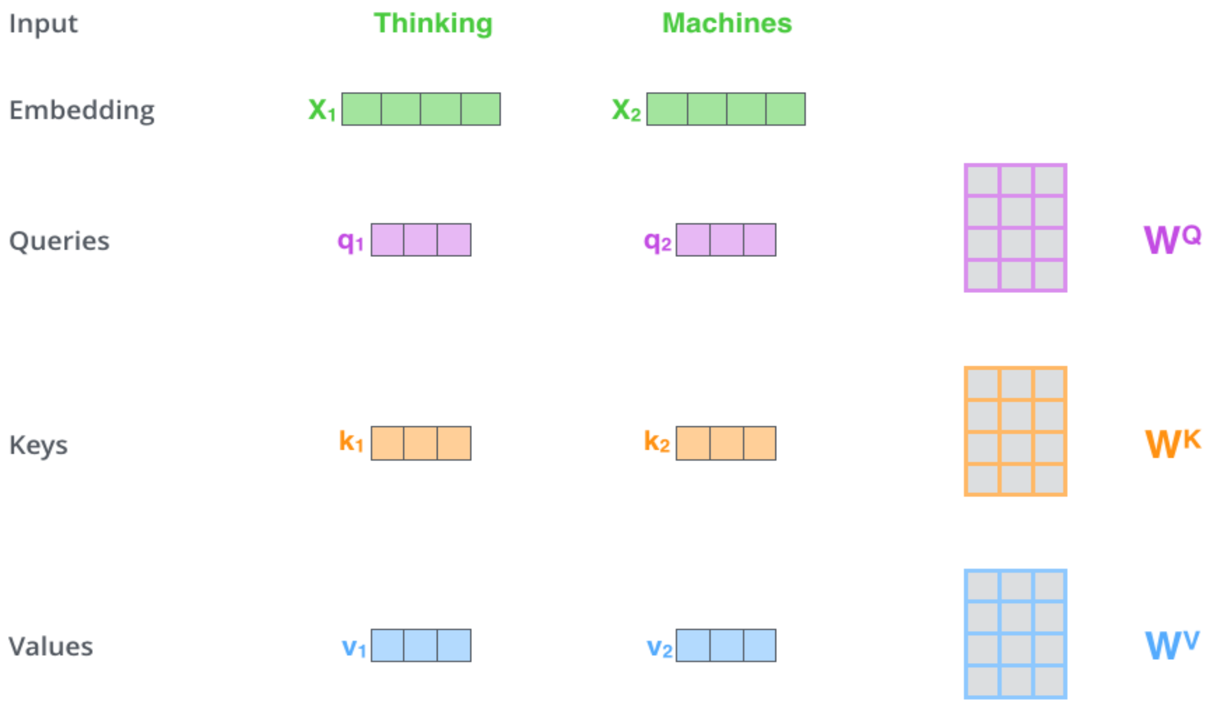
Key

Query

Value

Example Time: Self-attention layer

Step1: Calculate Queries, Keys, Values



Input — Thinking — Machines

Embedding — X_1 — X_2

Queries — q_1 — q_2 — W^Q

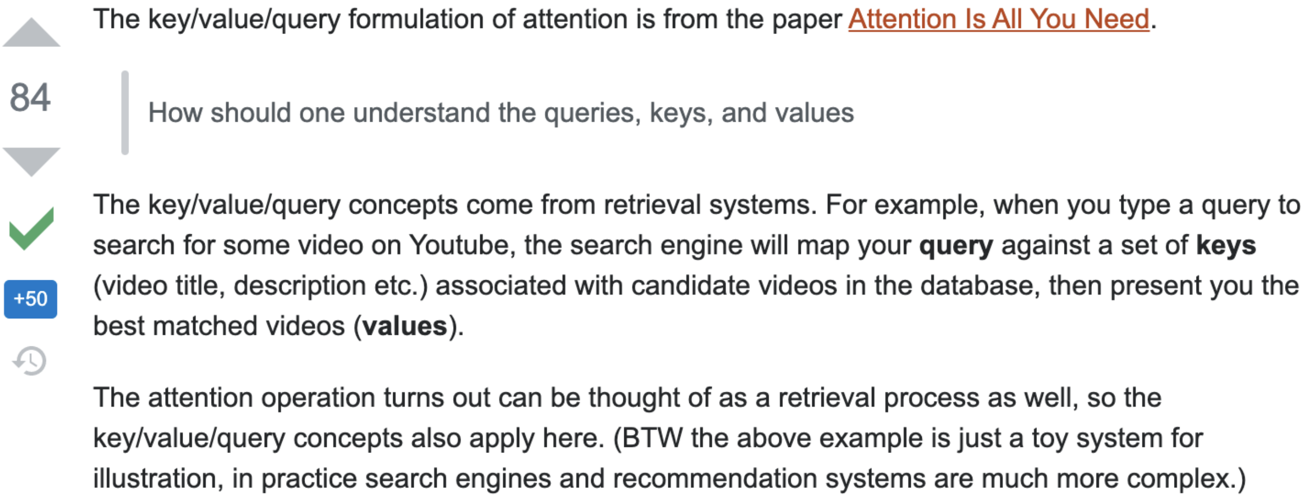
Keys — k_1 — k_2 — W^K

Values — v_1 — v_2 — W^V

Source: Jay Alammar

Example Time: Self-attention layer

How to understand the queries, keys and values in attention?



The key/value/query formulation of attention is from the paper Attention Is All You Need.

> How should one understand the queries, keys, and values

The key/value/query concepts come from retrieval systems. For example, when you type a query to search for some video on Youtube, the search engine will map your **query** against a set of **keys** (video title, description etc.) associated with candidate videos in the database, then present you the best matched videos (**values**).

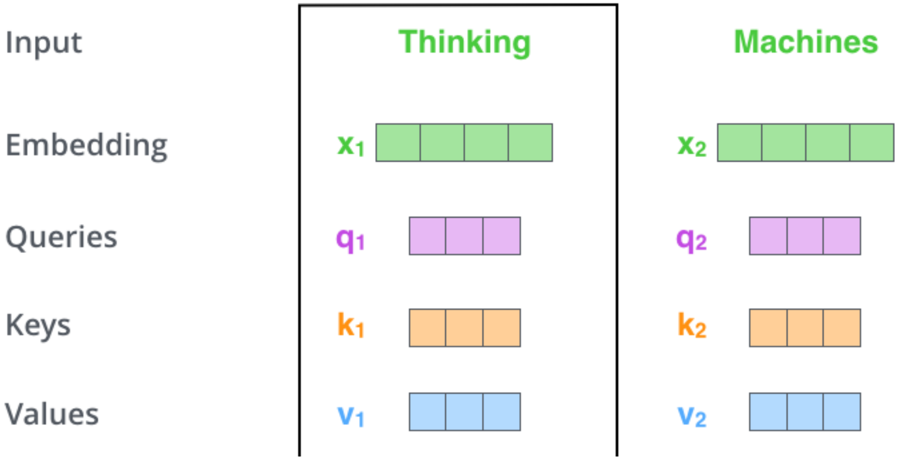
The attention operation turns out can be thought of as a retrieval process as well, so the key/value/query concepts also apply here. (BTW the above example is just a toy system for illustration, in practice search engines and recommendation systems are much more complex.)

https://stats.stackexchange.com/questions/421935/what-exactly-are-keys-queries-and-values-in-attention-mechanisms

Example Time: Self-attention layer

$$\text{attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_K}}\right)V$$

Step2: Calculate 'Scaled Dot-Product Attention' scores



Input

Embedding

Queries

Keys

Values

Thinking

Machines

x_1 x_2

q_1 q_2

k_1 k_2

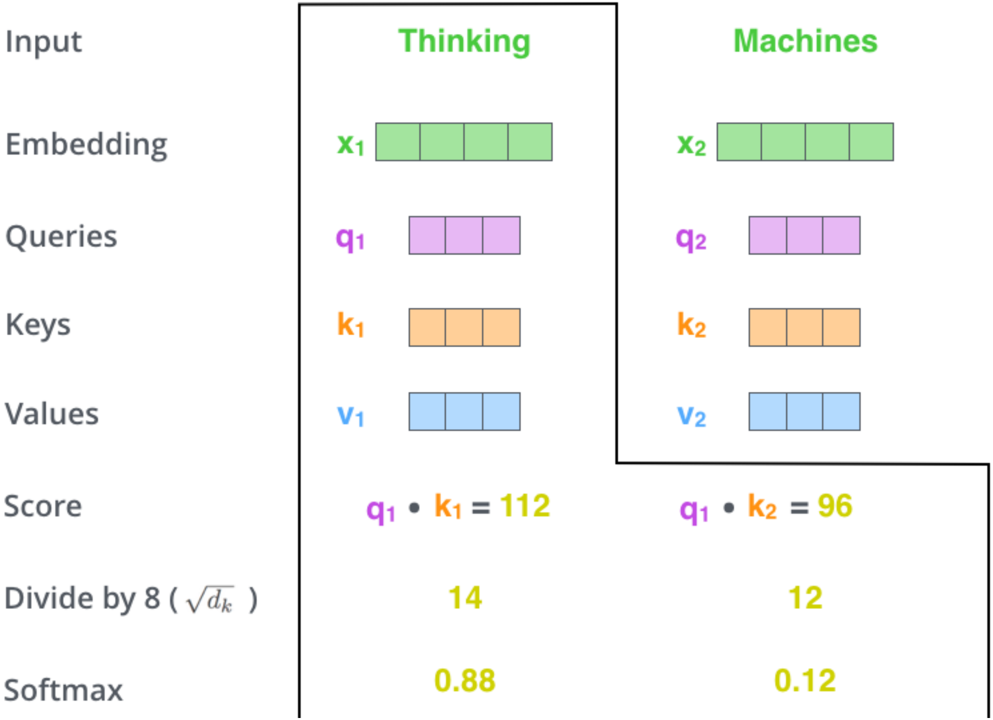
v_1 v_2

Source: Jay Alammar

Example Time: Self-attention layer

$$\text{attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_K}}\right) V$$

Step2: Calculate 'Scaled Dot-Product Attention' scores



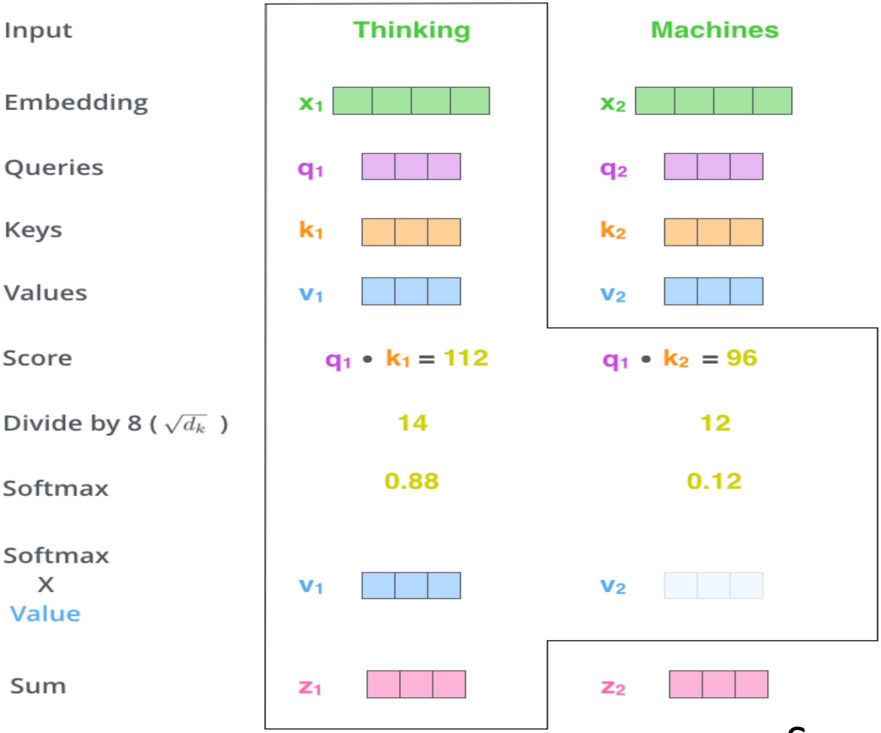
Input	Thinking	Machines
Embedding	x_1	x_2
Queries	q_1	q_2
Keys	k_1	k_2
Values	v_1	v_2
Score	$q_1 \bullet k_1 = 112$	$q_1 \bullet k_2 = 96$
Divide by 8 ($\sqrt{d_k}$)	14	12
Softmax	0.88	0.12

Source: Jay Alammar

Example Time: Self-attention layer

$$\text{attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_K}}\right) V$$

Step3: Calculate the output of self-attention layer



Input: Thinking | Machines

Embedding: x_1 | x_2

Queries: q_1 | q_2

Keys: k_1 | k_2

Values: v_1 | v_2

Score: $q_1 \bullet k_1 = 112$ | $q_1 \bullet k_2 = 96$

Divide by 8 ($\sqrt{d_k}$): 14 | 12

Softmax: 0.88 | 0.12

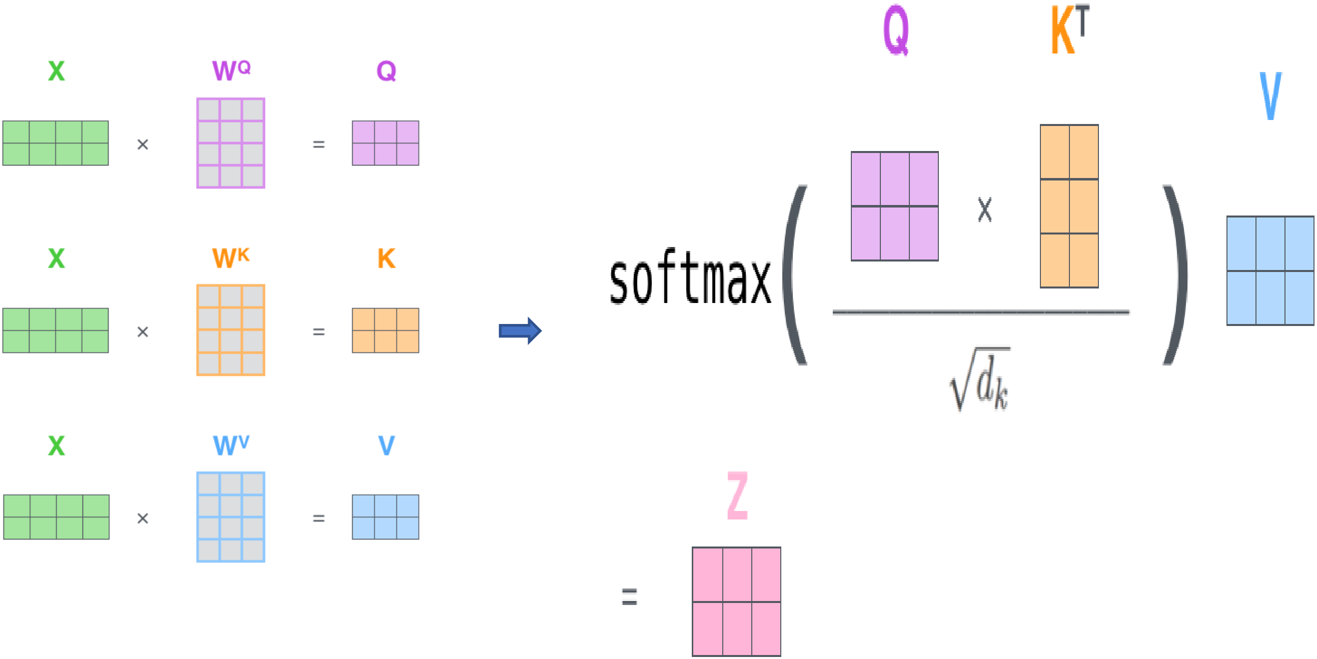
Softmax X Value: v_1 | v_2

Sum: z_1 | z_2

Source: Jay Alammar

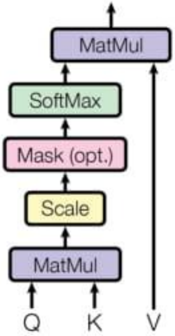
Example Time: Self-attention layer

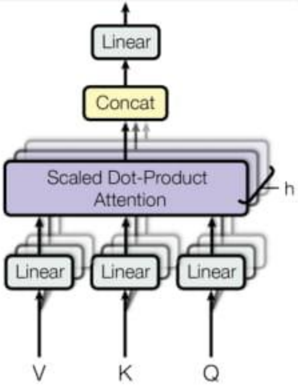
Matrix Form of Self-Attention



Source: Jay Alammar

Attention is All You Need





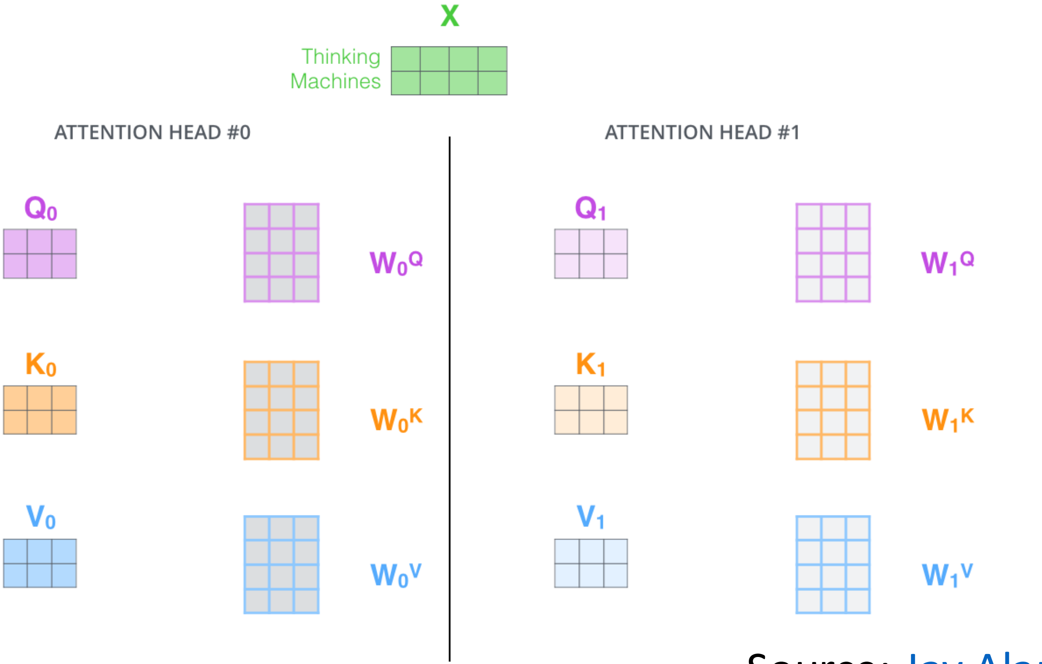
- The Scaled Dot-Product Attention attends to one or few entries in the input key-value pairs.
  - Humans can attend to many things simultaneously.
- The idea: apply Scaled Dot-Product Attention multiple times on the linearly transformed inputs.

$$\text{MultiHead}(Q, K, V) = \text{concat}\left(\mathbf{c}_1, \cdots, \mathbf{c}_h\right) W^O,$$

$$\mathbf{c}_i = \text{attention}(QW_i^Q, KW_i^K, VW_i^V).$$

A bit of extension: Multi-Head Attention

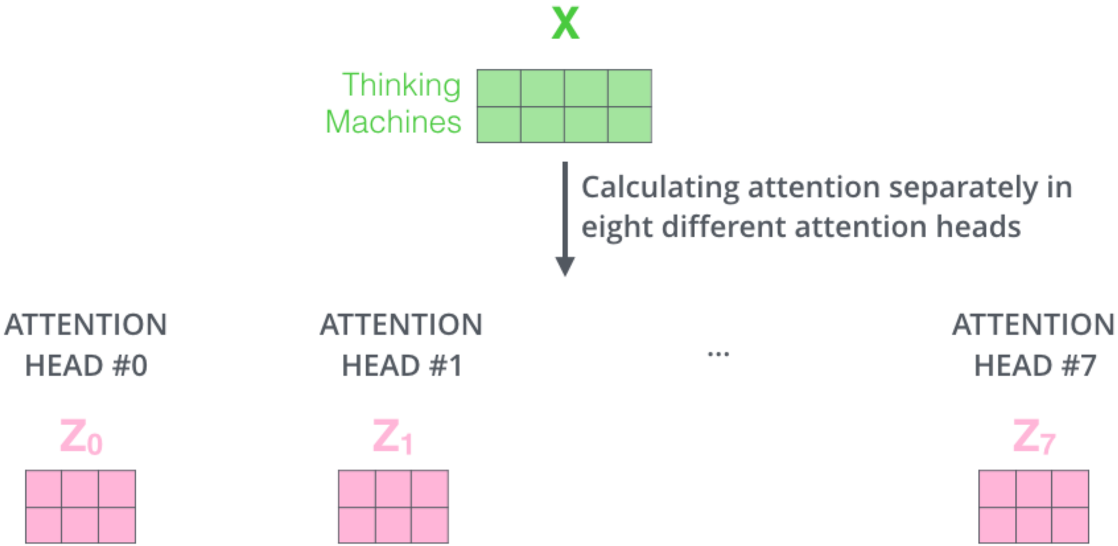
The beast with multiple heads --- Step 1 : Calculate multiple keys, queries, and values



Source: Jay Alammar

A bit of extension: Multi-Head Attention

The beast with multiple heads --- Step 2 : Calculate multiple attentions



X

Thinking Machines

Calculating attention separately in eight different attention heads

ATTENTION HEAD #0

ATTENTION HEAD #1

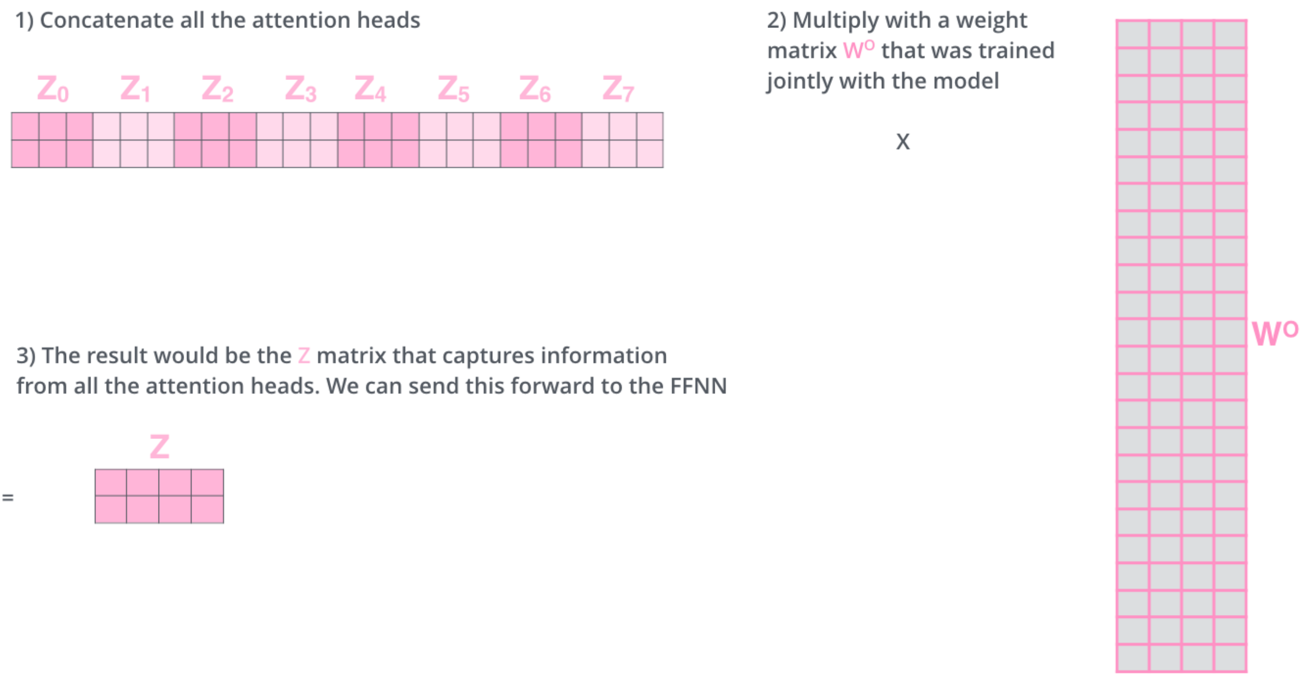
...

ATTENTION HEAD #7

Source: Jay Alammar

A bit of extension: Multi-Head Attention

The beast with multiple heads --- Step 3 : Concatenate and Compress



1) Concatenate all the attention heads

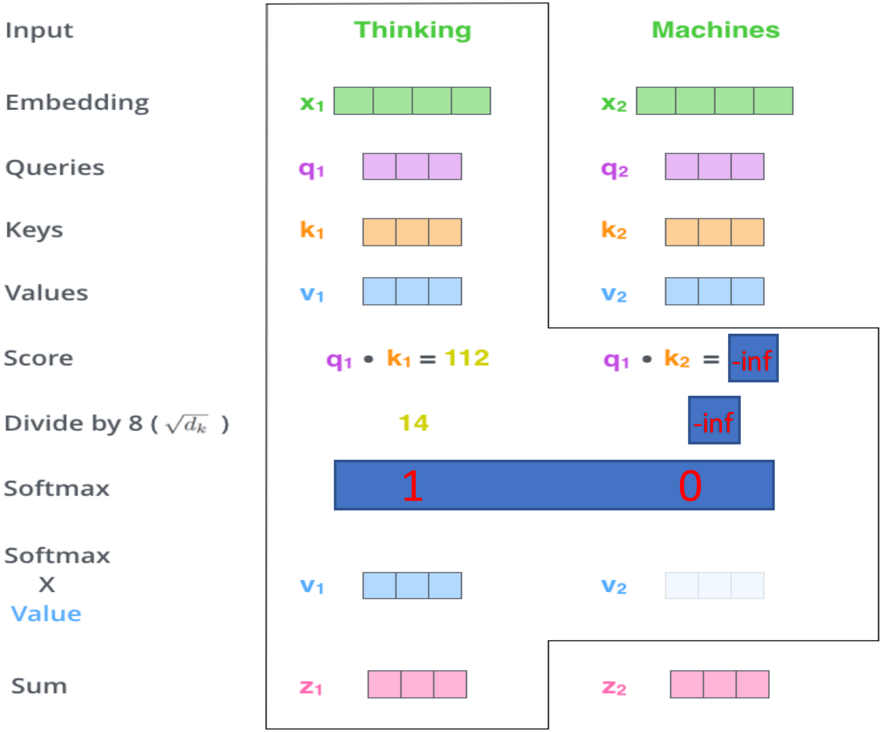
2) Multiply with a weight matrix W^O that was trained jointly with the model

3) The result would be the Z matrix that captures information from all the attention heads. We can send this forward to the FFNN

Source: Jay Alammar

A bit more of extension: Masked Multi-Head Attention

Sometimes you don't want to attend to future sequences, just mask them.



Input — Thinking — Machines

Embedding — x_1 — x_2

Queries — q_1 — q_2

Keys — k_1 — k_2

Values — v_1 — v_2

Score — $q_1 \bullet k_1 = 112$ — $q_1 \bullet k_2 =$ -inf

Divide by 8 ($\sqrt{d_k}$) — 14 — -inf

Softmax — 1 — 0

Softmax X Value — v_1 — v_2

Sum — z_1 — z_2

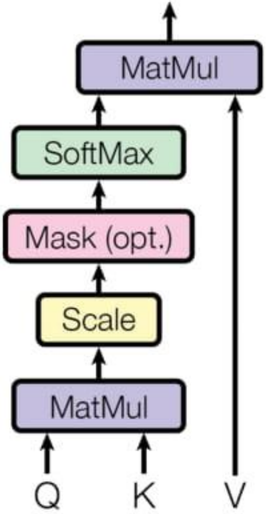
Summary: Self-attention layer

- In general, Attention mappings can be described as a function of a query and a set of key-value pairs.
- Transformers use a "Scaled Dot-Product Attention" to obtain the context vector:

$$\mathbf{c}^{(t)} = \text{attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_K}}\right) V,$$

scaled by square root of the key dimension d_K.

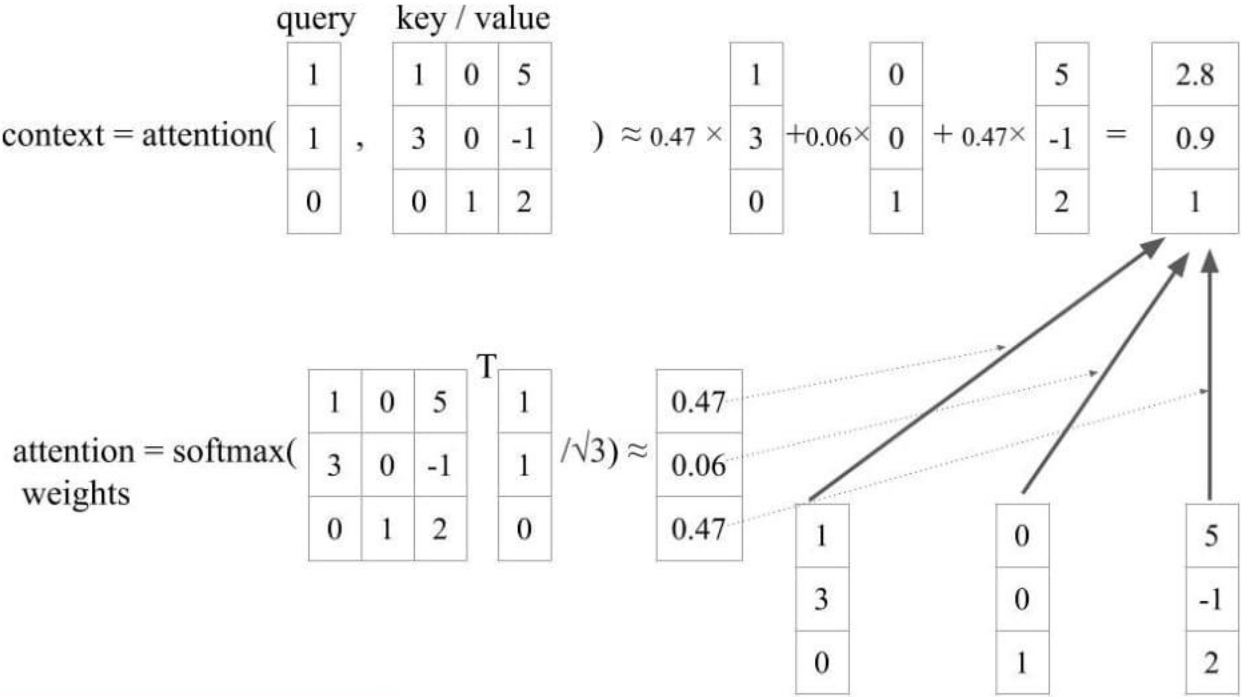
- Invalid connections to the future inputs are masked out to preserve the autoregressive property.



More Example: Scaled Dot-Product

$$\text{attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_K}}\right) V$$

Scale the un-normalized attention weights by the square root of the vector length:



query key / value

$$\text{context} = \text{attention}\left(\begin{array}{|c|}\hline 1 \\ \hline 1 \\ \hline 0 \\ \hline \end{array}, \begin{array}{|c|c|c|}\hline 1 & 0 & 5 \\ \hline 3 & 0 & -1 \\ \hline 0 & 1 & 2 \\ \hline \end{array} \right) \approx 0.47 \times \begin{array}{|c|}\hline 1 \\ \hline 3 \\ \hline 0 \\ \hline \end{array} + 0.06 \times \begin{array}{|c|}\hline 0 \\ \hline 0 \\ \hline 1 \\ \hline \end{array} + 0.47 \times \begin{array}{|c|}\hline 5 \\ \hline -1 \\ \hline 2 \\ \hline \end{array} = \begin{array}{|c|}\hline 2.8 \\ \hline 0.9 \\ \hline 1 \\ \hline \end{array}$$

$$\begin{array}{l}\text{attention} \\ \text{weights}\end{array} = \text{softmax}\left(\begin{array}{|c|c|c|}\hline 1 & 0 & 5 \\ \hline 3 & 0 & -1 \\ \hline 0 & 1 & 2 \\ \hline \end{array}^{T} \begin{array}{|c|}\hline 1 \\ \hline 1 \\ \hline 0 \\ \hline \end{array} / \sqrt{3} \right) \approx \begin{array}{|c|}\hline 0.47 \\ \hline 0.06 \\ \hline 0.47 \\ \hline \end{array}$$

$$\begin{array}{|c|}\hline 1 \\ \hline 3 \\ \hline 0 \\ \hline \end{array} \quad \begin{array}{|c|}\hline 0 \\ \hline 0 \\ \hline 1 \\ \hline \end{array} \quad \begin{array}{|c|}\hline 5 \\ \hline -1 \\ \hline 2 \\ \hline \end{array}$$

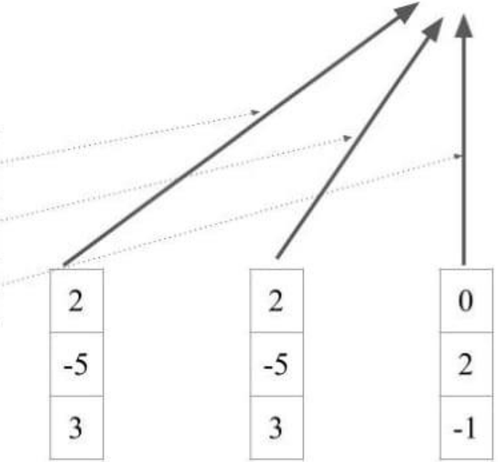
More Example: Different Keys and

$$\text{attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_K}}\right) V$$

When the key and the value vectors are different:

$$\text{context} = \text{attention}\left(\overset{\text{query}}{\begin{bmatrix}1\\1\\0\end{bmatrix}}, \overset{\text{key}}{\begin{bmatrix}1&0&5\\3&0&-1\\0&1&2\end{bmatrix}}, \overset{\text{value}}{\begin{bmatrix}2&2&0\\-5&-5&2\\3&3&-1\end{bmatrix}}\right) \approx 0.47\times\begin{bmatrix}2\\-5\\3\end{bmatrix} + 0.06\times\begin{bmatrix}2\\-5\\3\end{bmatrix} + 0.47\times\begin{bmatrix}0\\2\\-1\end{bmatrix} = \begin{bmatrix}1.06\\-1.71\\1.12\end{bmatrix}$$

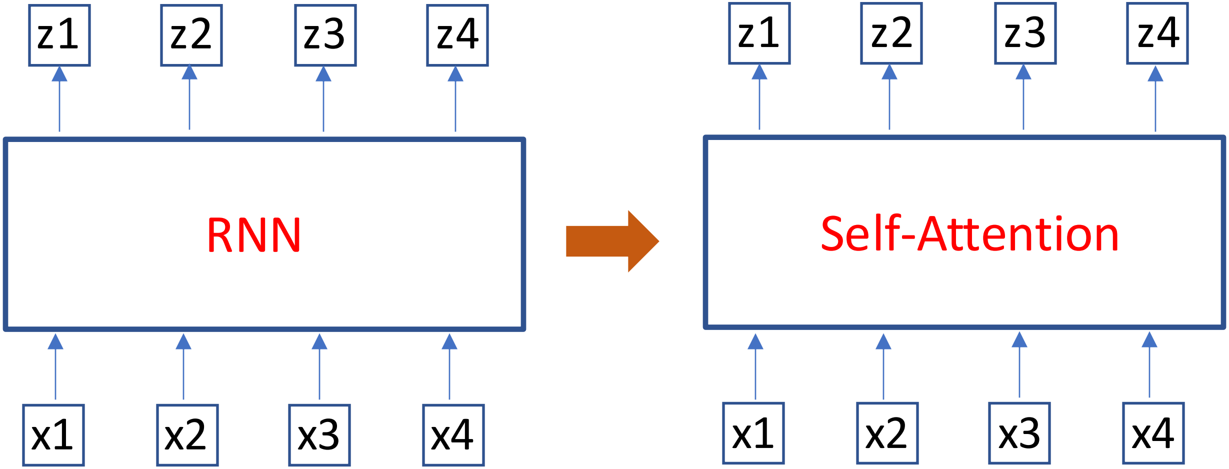
$$\begin{array}{l}\text{attention}\\ \text{weights}\end{array} = \text{softmax}\left(\begin{bmatrix}1&0&5\\3&0&-1\\0&1&2\end{bmatrix}^{T}\begin{bmatrix}1\\1\\0\end{bmatrix} / \sqrt{3}\right) \approx \begin{bmatrix}0.47\\0.06\\0.47\end{bmatrix}$$



The core of Transformers: Self-attention layer

Key Takeaway:

1. Self-attention learns how/where to attend within the whole input sequences.
2. Self-attention can be computed parallelly
3. F(S) = F(R) * (1 + epi)



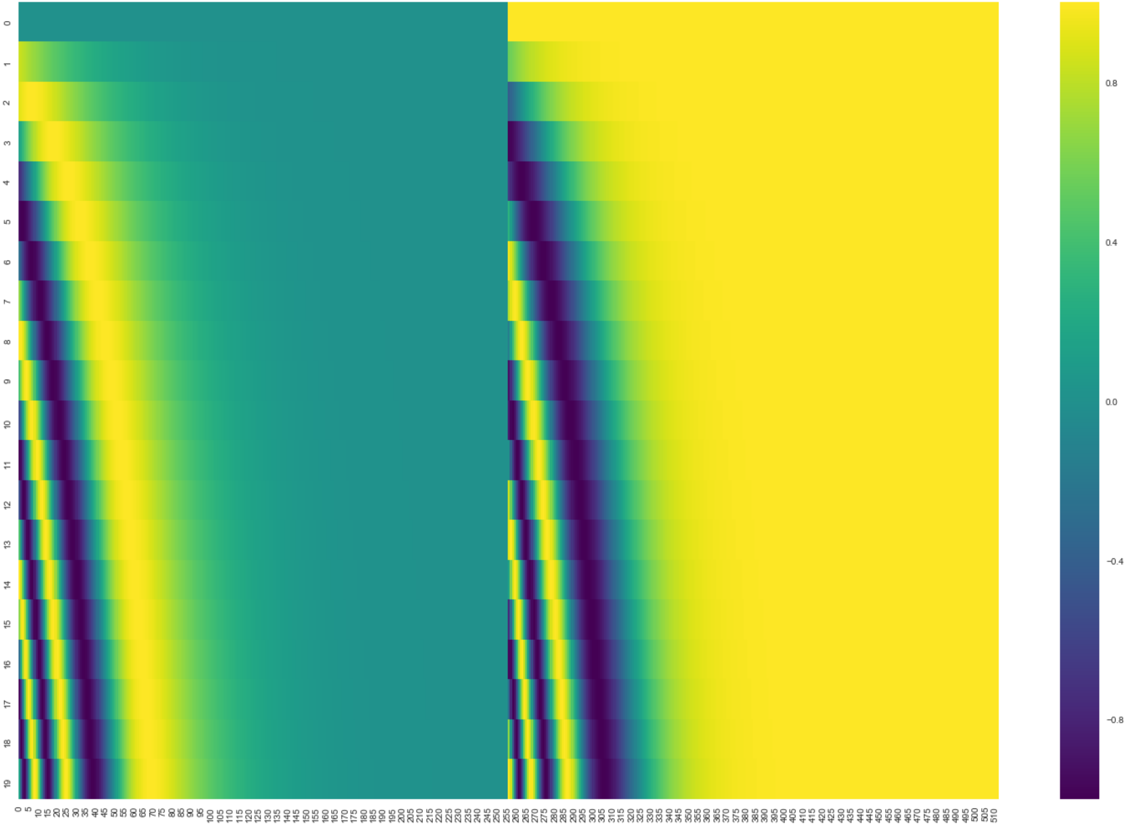
GPS system: Positional Encoding

- Unlike RNNs and CNNs encoders, the attention encoder outputs do not depend on the order of the inputs. (Why?)
- The order of the sequence conveys important information for the machine translation tasks and language modeling.
- The idea: add positional information of a input token in the sequence into the input embedding vectors.

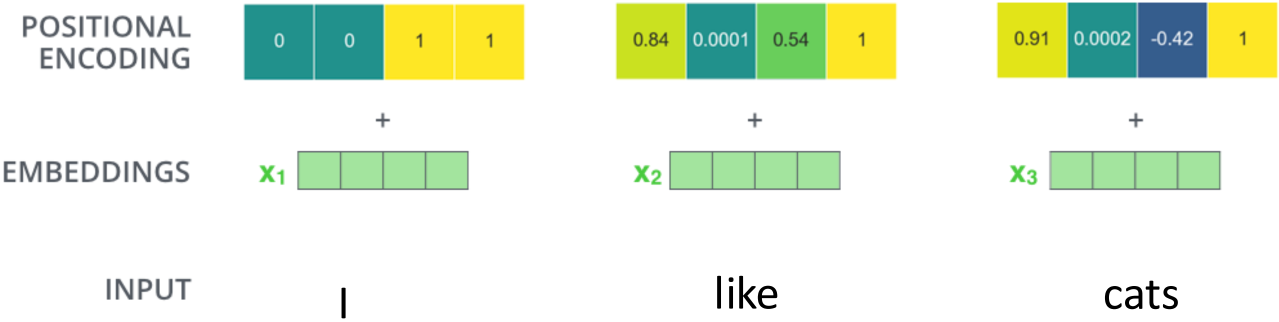
$$\mathrm{PE}_{pos,2i} = sin(pos/10000^{2i/d_{emb}}),$$
$$\mathrm{PE}_{pos,2i+1} = cos(pos/10000^{2i/d_{emb}}),$$

- The final input embeddings are the concatenation of the learnable embedding and the postional encoding.

GPS system: Positional Encoding



GPS system: Positional Encoding



POSITIONAL ENCODING

0 0 1 1

0.84 0.0001 0.54 1

0.91 0.0002 -0.42 1

+ + +

EMBEDDINGS

x_1 x_2 x_3

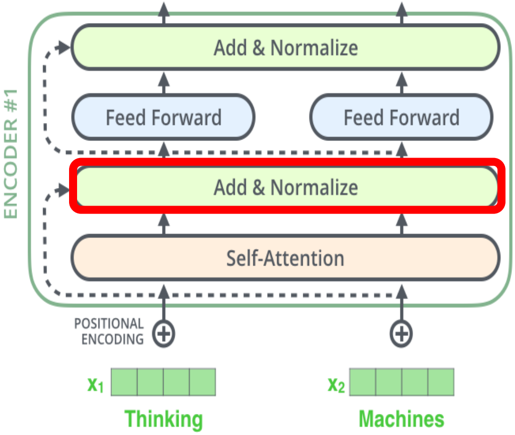
INPUT

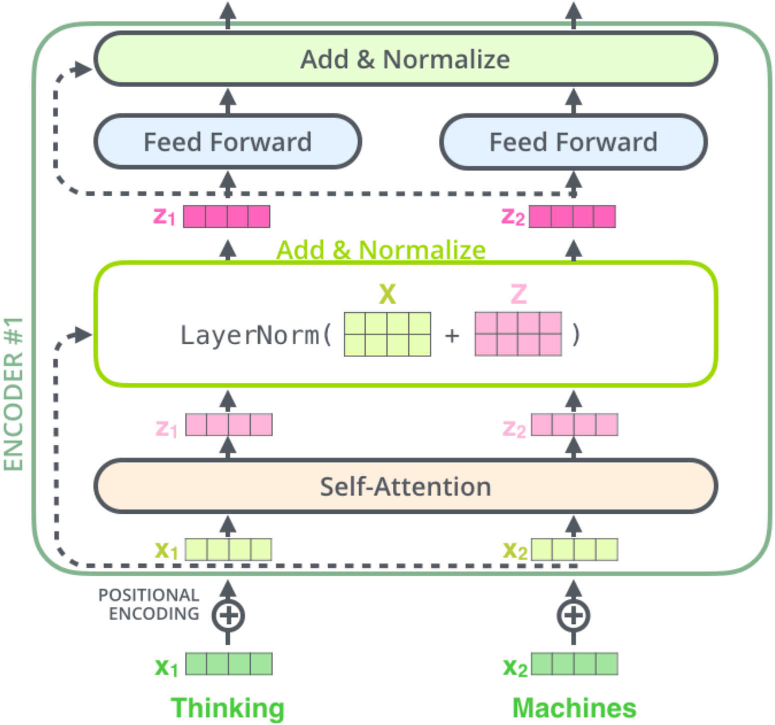
I like cats

A real example of positional encoding with a toy embedding size of 4

Source: Jay Alammar

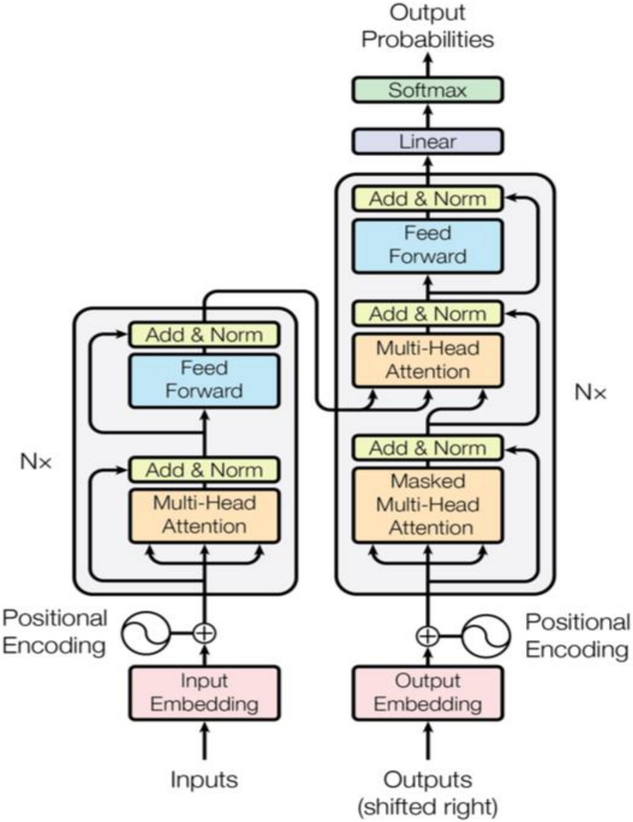
Nothing Fancy: Skip Connections and Normalization



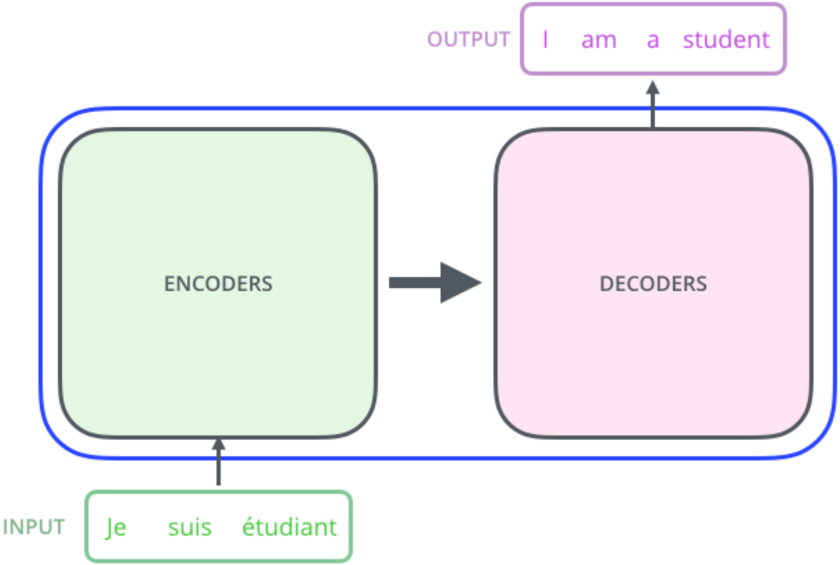


Source: Jay Alammar

Let's assemble the transformer!

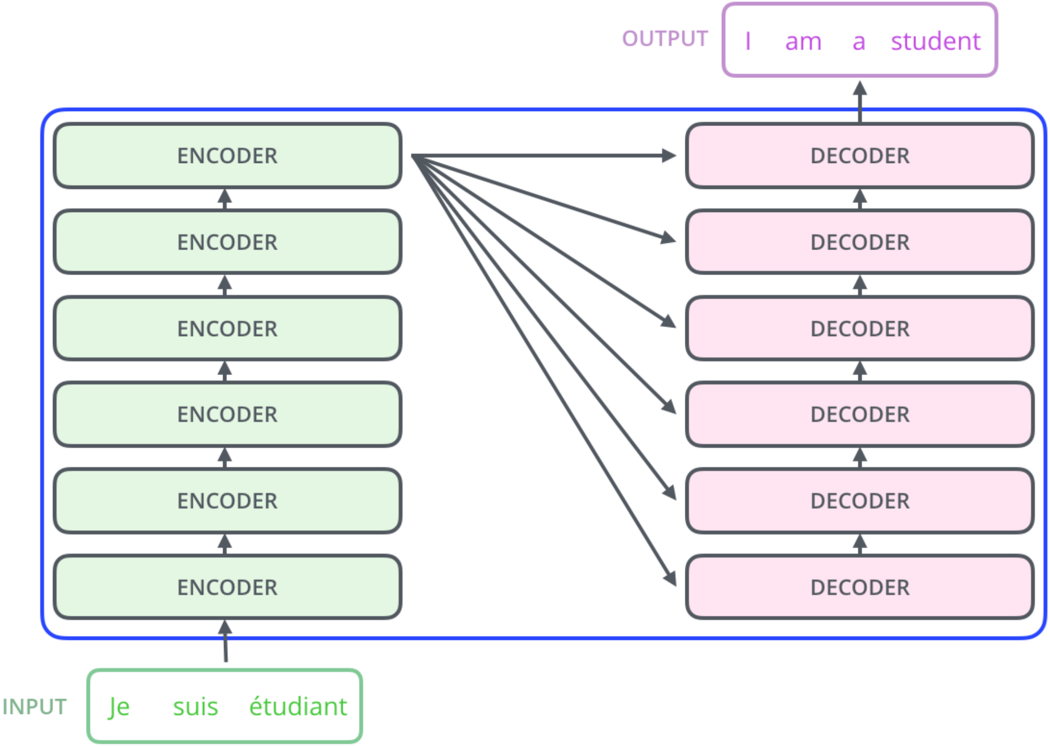


Main Body: Encoder-Decoder



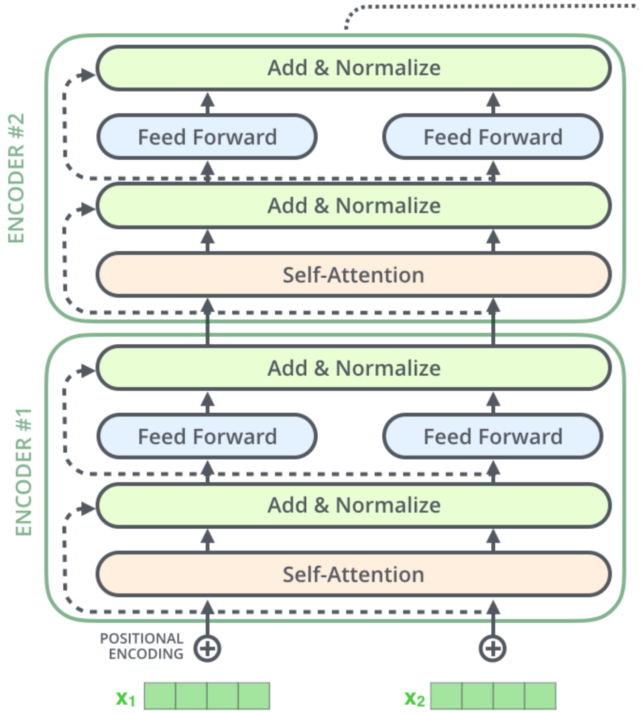
Source: Jay Alammar

Main Body: Encoder-Decoder



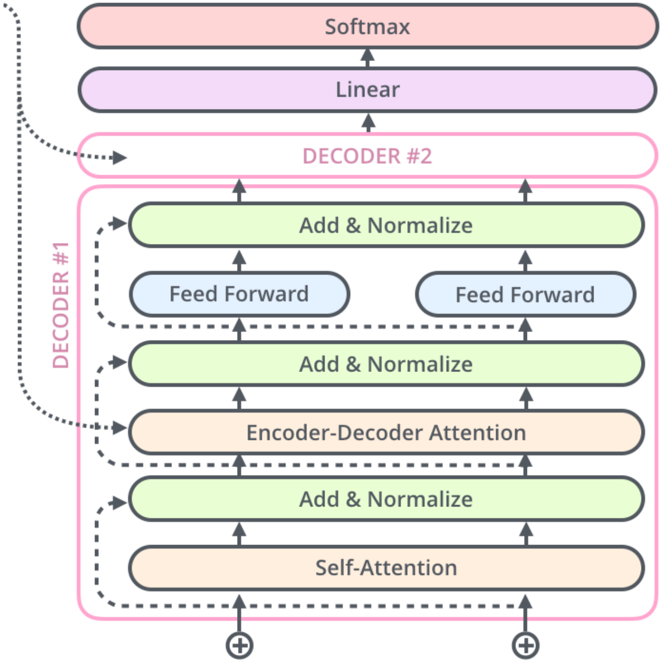
Source: Jay Alammar

Encoder



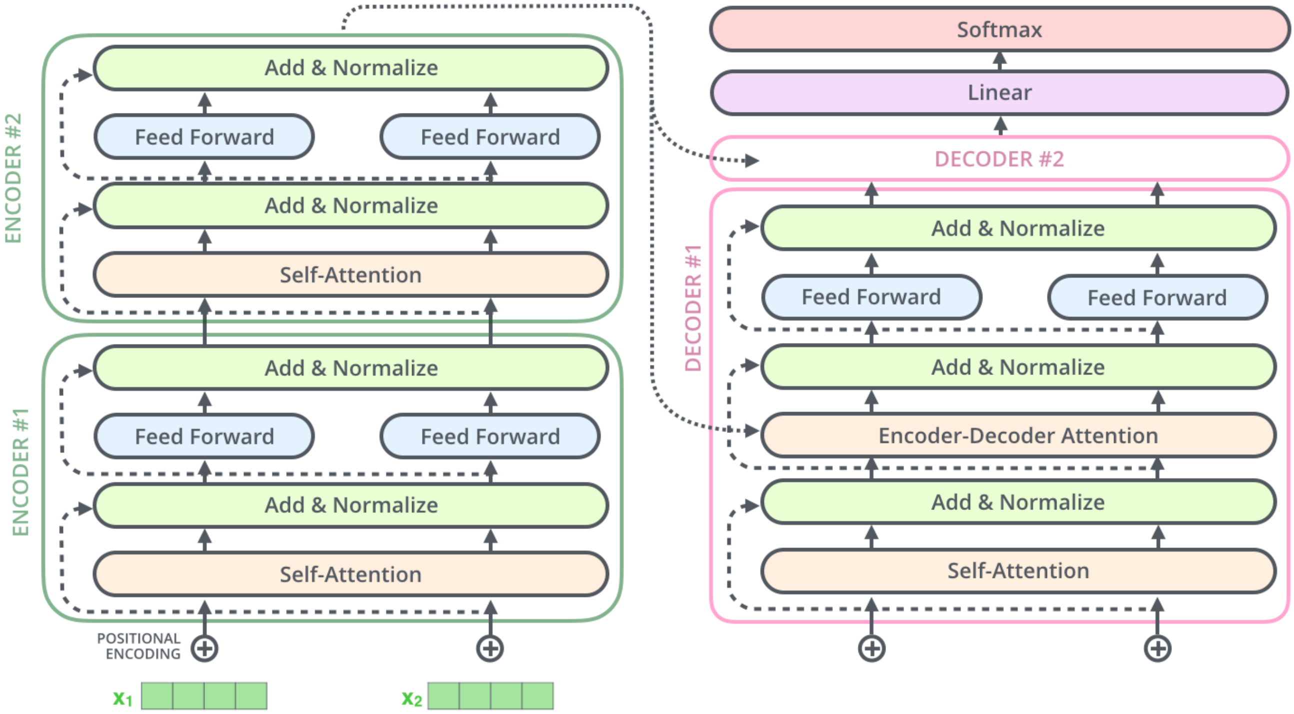
Source: Jay Alammar

Decoder



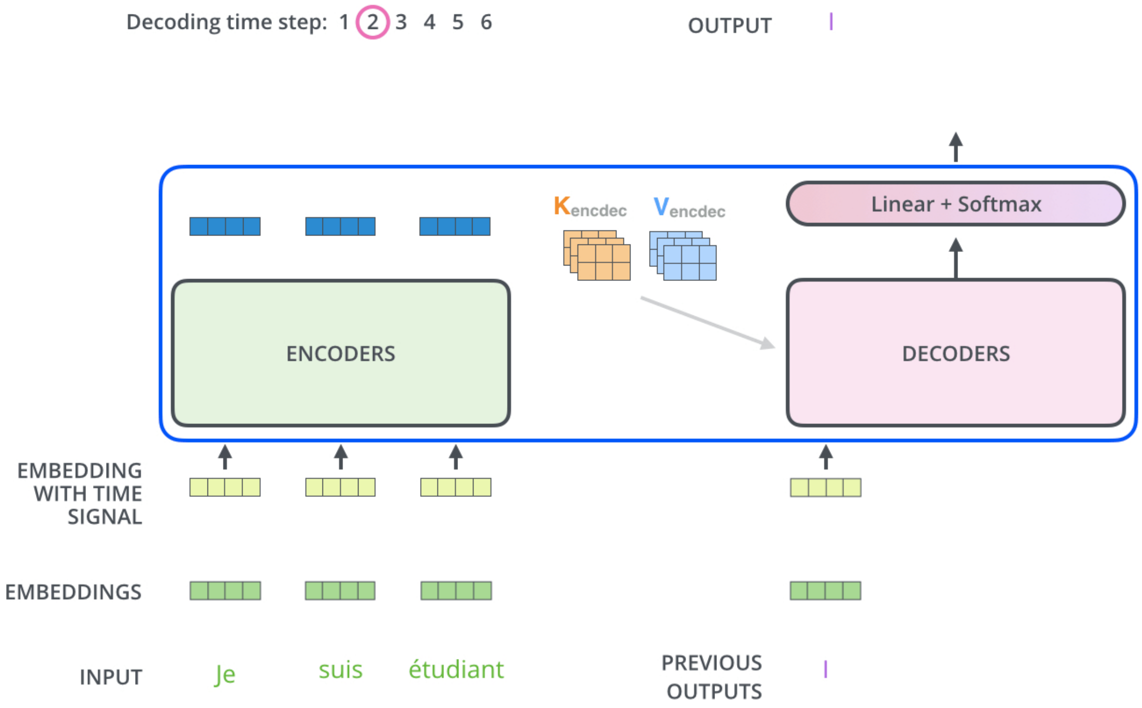
Source: Jay Alammar

Main Body: Encoder-Decoder



Source: Jay Alammar

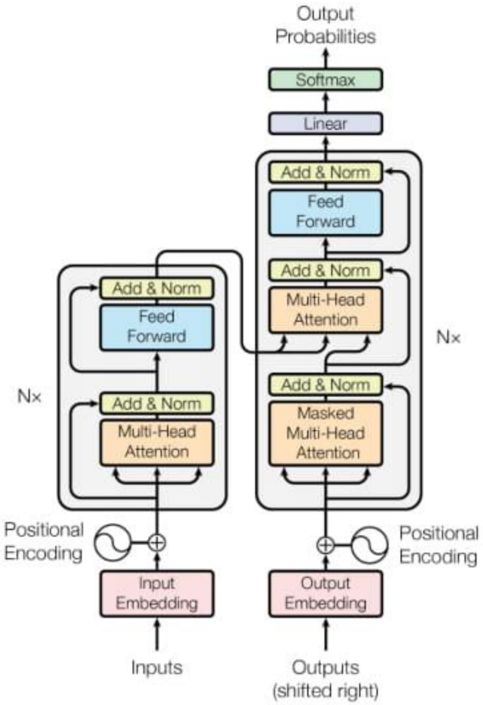
Main Body: Encoder-Decoder Training



Source: Jay Alammar

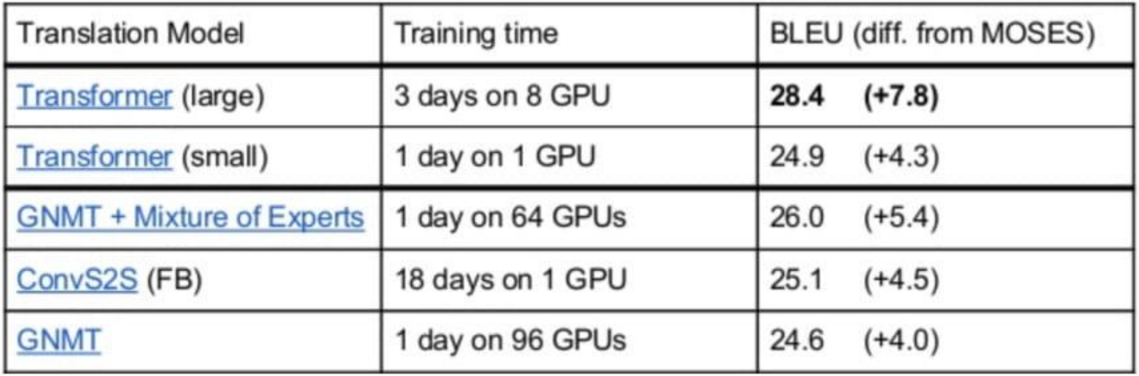
Transformer Machine Translation

- Transformer has a encoder-decoder architecture similar to the previous RNN models.
  - except all the recurrent connections are replaced by the attention modules.
- The transfomer model uses N stacked self-attention layers.
- Skip-connections help preserve the positional and identity information from the input sequences.



Transformer Machine Translation

- BLEU scores of state-of-the-art models on the WMT14 English-to-German translation task



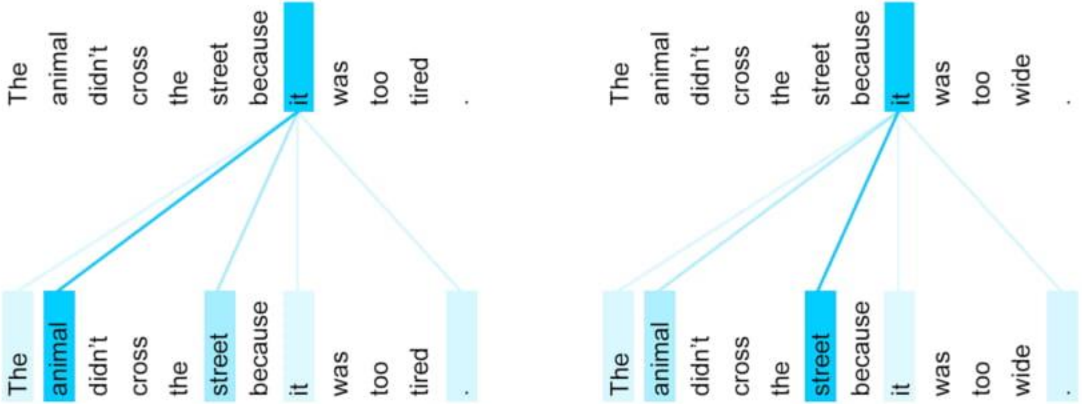
Translation Model	Training time	BLEU (diff. from MOSES)
Transformer (large)	3 days on 8 GPU	**28.4 (+7.8)**
Transformer (small)	1 day on 1 GPU	24.9 (+4.3)
GNMT + Mixture of Experts	1 day on 64 GPUs	26.0 (+5.4)
ConvS2S (FB)	18 days on 1 GPU	25.1 (+4.5)
GNMT	1 day on 96 GPUs	24.6 (+4.0)

Vaswani, Ashish, et al. "Attention is all you need." Advances in Neural Information Processing Systems. 2017.

Source: Jay Alammar

Transformer Machine Translation

- Self-attention layers learnt "it" could refer to different entities in the different contexts.

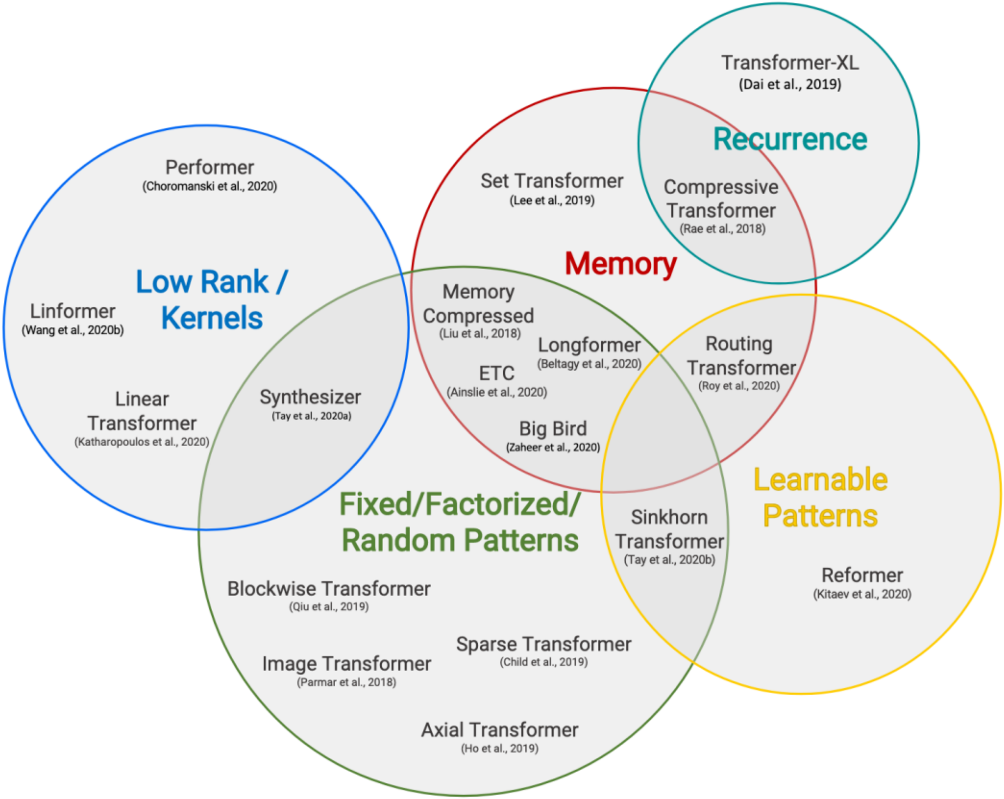


- Visualization of the 5th to 6th self-attention layer in the encoder.

`https://ai.googleblog.com/2017/08/transformer-novel-neural-network.html`

Transformer Machine Translation

F(S) = F(R) * (1 + epi)



Transformer-XL
(Dai et al., 2019)

Recurrence

Compressive Transformer
(Rae et al., 2018)

Set Transformer
(Lee et al., 2019)

Memory

Performer
(Choromanski et al., 2020)

Low Rank / Kernels

Linformer
(Wang et al., 2020b)

Linear Transformer
(Katharopoulos et al., 2020)

Synthesizer
(Tay et al., 2020a)

Memory Compressed
(Liu et al., 2018)

Longformer
(Beltagy et al., 2020)

ETC
(Ainslie et al., 2020)

Big Bird
(Zaheer et al., 2020)

Routing Transformer
(Roy et al., 2020)

Learnable Patterns

Sinkhorn Transformer
(Tay et al., 2020b)

Reformer
(Kitaev et al., 2020)

Fixed/Factorized/ Random Patterns

Blockwise Transformer
(Qiu et al., 2019)

Sparse Transformer
(Child et al., 2019)

Image Transformer
(Parmar et al., 2018)

Axial Transformer
(Ho et al., 2019)

After the break

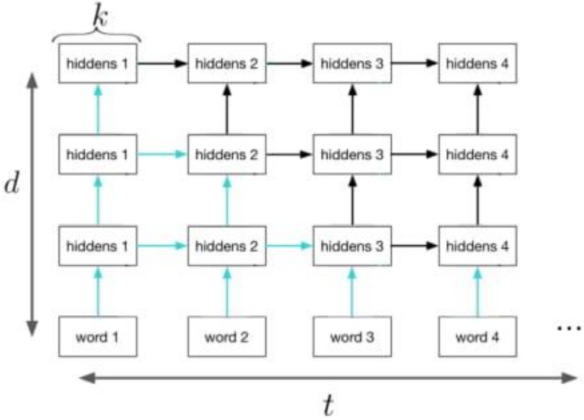
After the break: **Computational Cost**

Computational Cost and Parallelism

- There are a few things we should consider when designing an RNN.
- Computational cost:
  - **Number of connections.** How many add-multiply operations for the forward and backward pass.
  - **Number of time steps.** How many copies of hidden units to store for Backpropgation Through Time.
  - **Number of sequential operations.** The computations cannot be parallelized. (The part of the model that requires a for loop).
- **Maximum path length across time**: the shortest path length between the first encoder input and the last decoder output.
  - It tells us how easy it is for the RNN to remember / retreive information from the input sequence.

Computational Cost and Parallelism

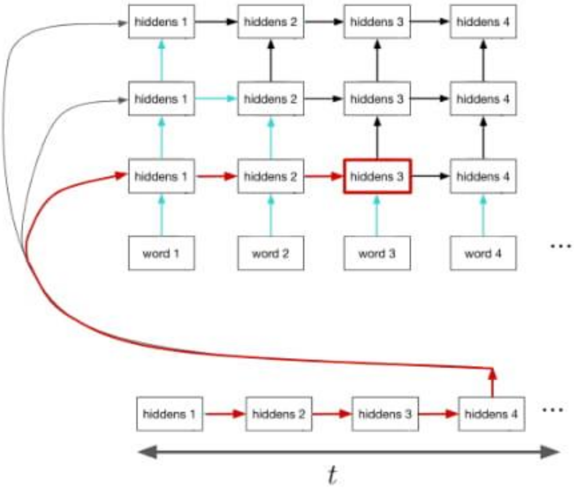
- Consider a standard d layer RNN from Lecture 7 with k hidden units, training on a sequence of length t.

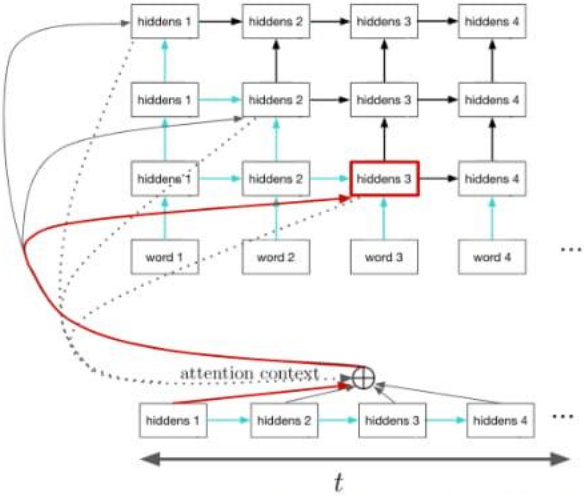


- There are k^2 connections for each hidden-to-hidden connection. A total of $t \times k^2 \times d$ connections.
- We need to store all $t \times k \times d$ hidden units during training.
- Only $k \times d$ hidden units need to be stored at test time.

Computational Cost and Parallelism

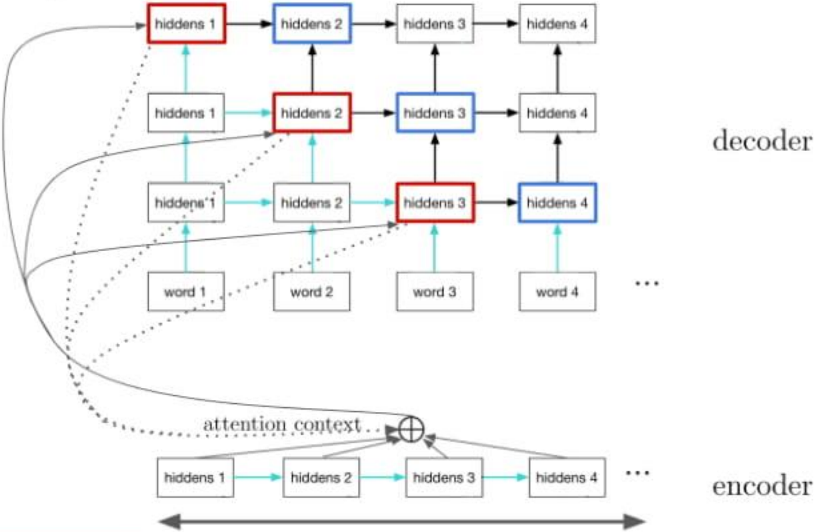
- During backprop, in the standard encoder-decoder RNN, the maximum path length across time is the number of time steps.
- Attention-based RNNs have a constant path length between the encoder inputs and the decoder hidden states.
  - Learning becomes easier. Why?





Computational Cost and Parallelism

- During forward pass, attention-based RNNs achieves efficient content-based addressing at the cost of re-computing context vectors at each time step.
  - *Bahdanau et. al.* computes context vector over the entire input sequence of length t using a neural network of k^2 connections.
  - Computing the context vectors adds a $t \times k^2$ cost at each time step.



Computational Cost and Parallelism

- In summary:
  - t: sequence length, d: # layers and k: # neurons at each layer.

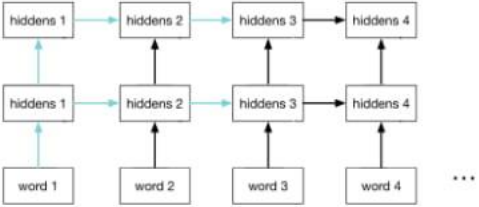
	training	training	test	test
Model	complexity	memory	complexity	memory
RNN	$t \times k^2 \times d$	$t \times k \times d$	$t \times k^2 \times d$	$k \times d$
RNN+attn.	$t^2 \times k^2 \times d$	$t^2 \times k \times d$	$t^2 \times k^2 \times d$	$t \times k \times d$

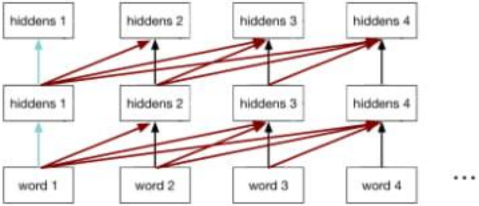
- Attention needs to re-compute context vectors at every time step.
- Attention has the benefit of reducing the maximum path length between long range dependencies of the input and the target sentences.

	sequential	maximum path
Model	operations	length across time
RNN	t	t
RNN+attn.	t	1

Improve Parallelism

- RNNs are sequential in the sequence length t due to the number hidden-to-hidden lateral connections.
  - RNN architecture limits the parallelism potential for longer sequences.
- Improve parallelism: remove the lateral connections. We will have a deep autoregressive model, where the hidden units depends on all the previous time steps.

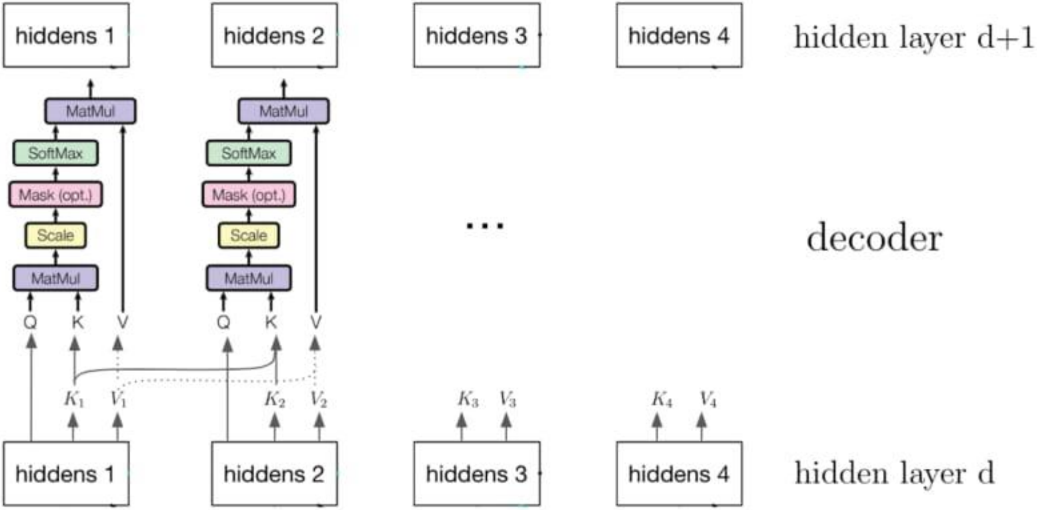




  - Benefit: the number of sequential operations is now linear in the depth d, but is independent of the sequence length t. (usually $d << t$.)

Computational Cost and Parallelism

- Self-attention allows the model to learn to access information from the past hidden layer, but decoding is very expensive.
- When generating sentences, the computation in the self-attention decoder grows as the sequence gets longer.



Computational Cost and Parallelism

$\text{attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_K}}\right)V$

- t: sequence length, d: # layers and k: # neurons at each layer.

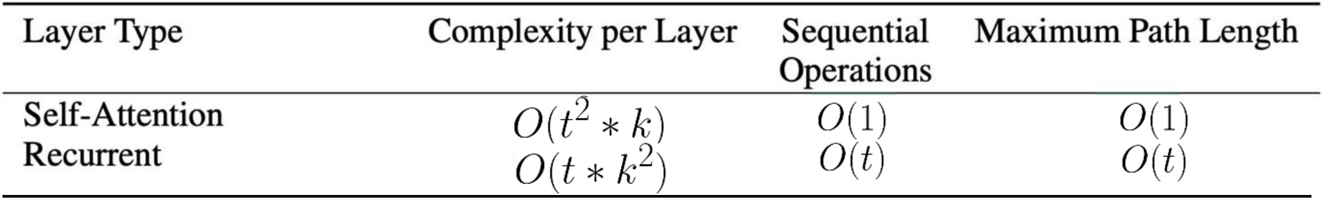
Model	training complexity	training memory	test complexity	test memory
RNN	$t \times k^2 \times d$	$t \times k \times d$	$t \times k^2 \times d$	$k \times d$
RNN+attn.	$t^2 \times k^2 \times d$	$t^2 \times k \times d$	$t^2 \times k^2 \times d$	$t \times k \times d$
transformer	$\boxed{t^2 \times k} \times d$	$t \times k \times d$	$t^2 \times k \times d$	$t \times k \times d$

- **Transformer vs RNN**: There is a trade-off between the sequencial operations and decoding complexity.
  - The sequential operations in transformers are independent of sequence length, but they are very expensive to decode.
  - Transformers can learn faster than RNNs on parallel processing hardwards for longer sequences.

Model	sequential operations	maximum path length across time
RNN	t	t
RNN+attn.	t	1
transformer	d	1

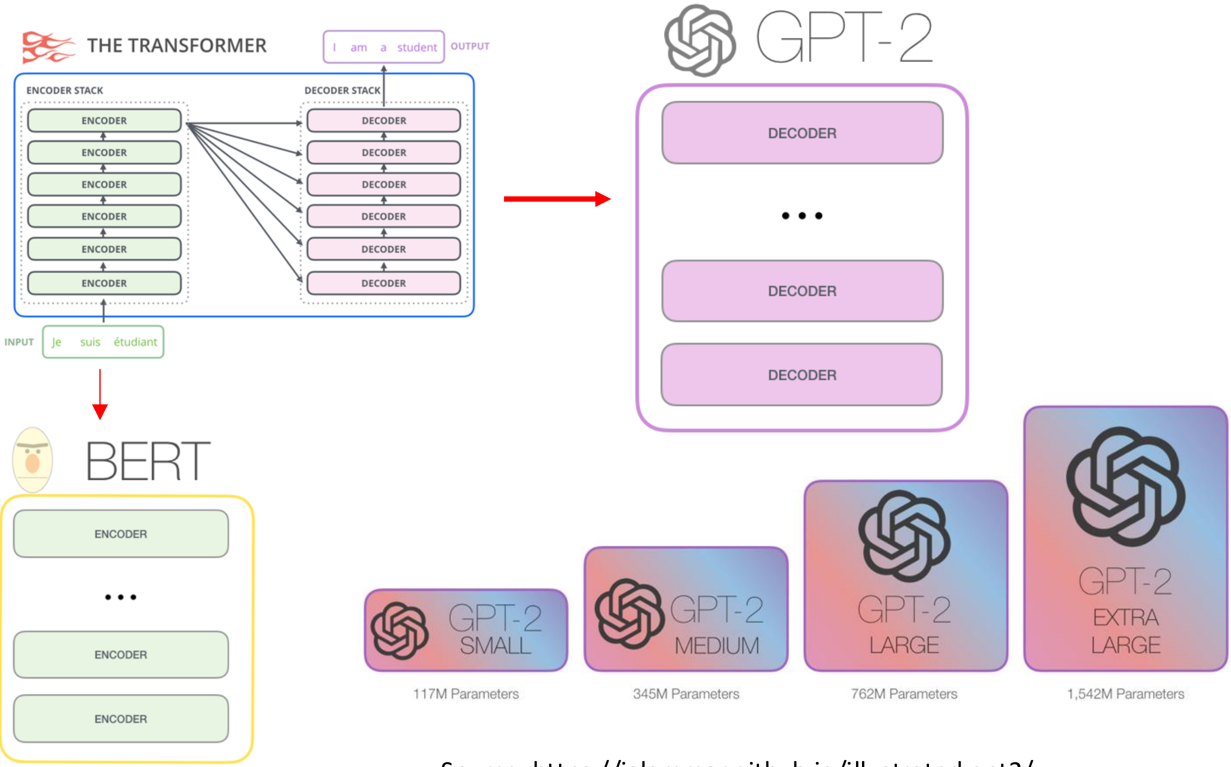
Computational Cost --- Quick Summary

Self-attention v.s. RNN



Layer Type	Complexity per Layer	Sequential Operations	Maximum Path Length
Self-Attention	$O(t^2 * k)$	$O(1)$	$O(1)$
Recurrent	$O(t * k^2)$	$O(t)$	$O(t)$

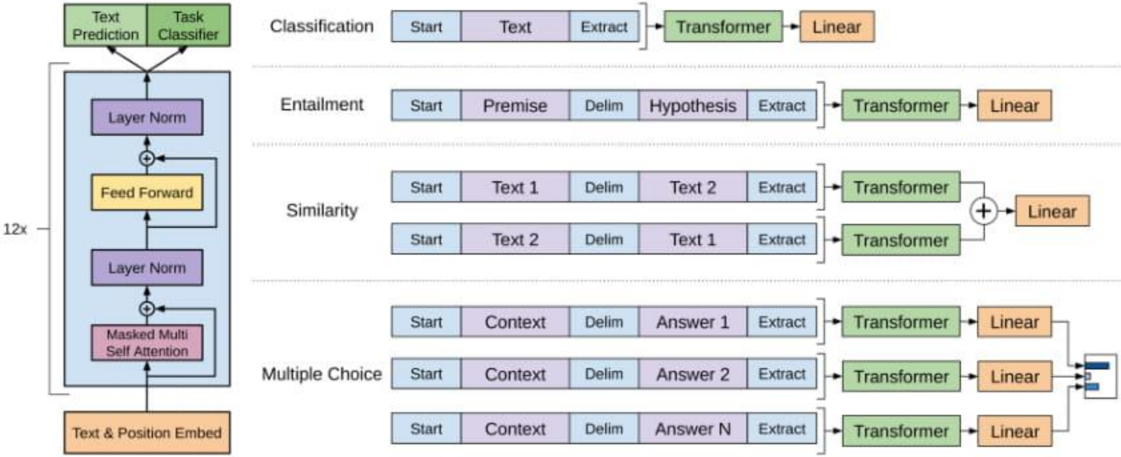
The Rise of Large Language Models



Source: https://jalammar.github.io/illustrated-gpt2/

Transformer Language Pre-training

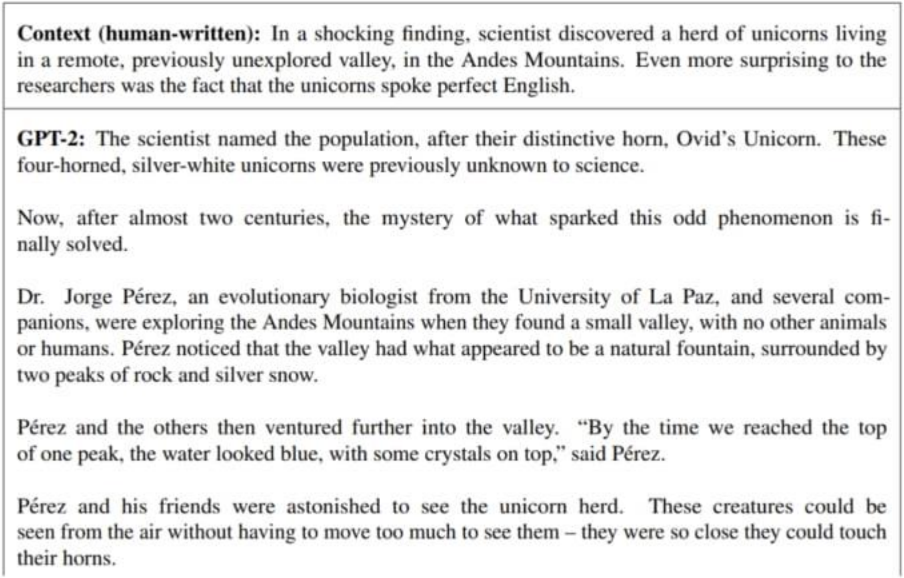
- Similar to pre-training computer vision models on ImageNet, we can pre-train a language model for NLP tasks.
  - The pre-trained model is then fine-tuned on textual entailment, question answering, semantic similarity assessment, and document classification.



Radford, Alec, et al. "Improving Language Understanding by Generative Pre-Training." 2018.

Transformer Language Pre-training

- Increasing the training data set and the model size has a noticible improvement on the transformer language model. Cherry picked generated samples from *Radford, et al., 2019*:



Context (human-written): In a shocking finding, scientist discovered a herd of unicorns living in a remote, previously unexplored valley, in the Andes Mountains. Even more surprising to the researchers was the fact that the unicorns spoke perfect English.

GPT-2: The scientist named the population, after their distinctive horn, Ovid's Unicorn. These four-horned, silver-white unicorns were previously unknown to science.

Now, after almost two centuries, the mystery of what sparked this odd phenomenon is finally solved.

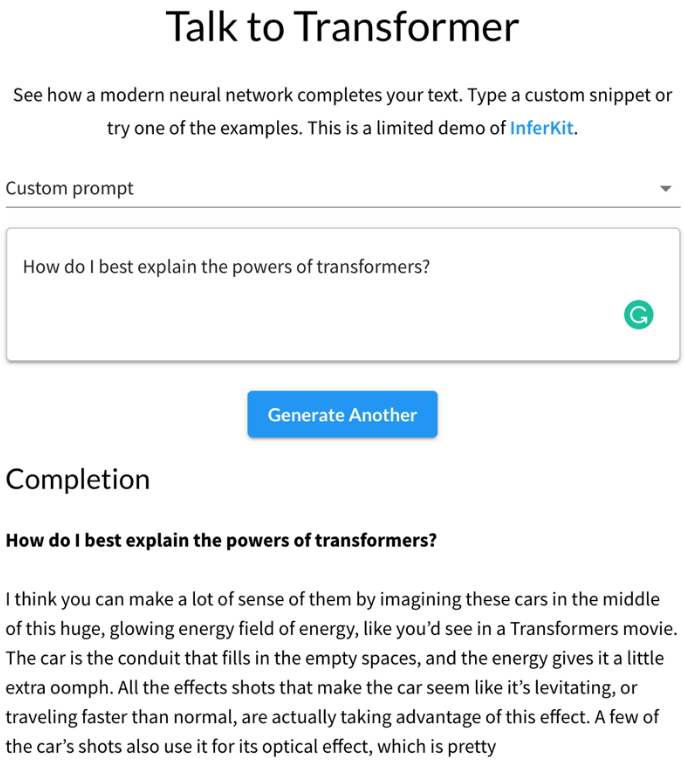
Dr. Jorge Pérez, an evolutionary biologist from the University of La Paz, and several companions, were exploring the Andes Mountains when they found a small valley, with no other animals or humans. Pérez noticed that the valley had what appeared to be a natural fountain, surrounded by two peaks of rock and silver snow.

Pérez and the others then ventured further into the valley. "By the time we reached the top of one peak, the water looked blue, with some crystals on top," said Pérez.

Pérez and his friends were astonished to see the unicorn herd. These creatures could be seen from the air without having to move too much to see them – they were so close they could touch their horns.

For the full text samples see Radford, Alec, et al. "Language Models are Unsupervised Multitask Learners." 2019.

Transformer Language Pre-training



Talk to Transformer

See how a modern neural network completes your text. Type a custom snippet or try one of the examples. This is a limited demo of InferKit.

Custom prompt

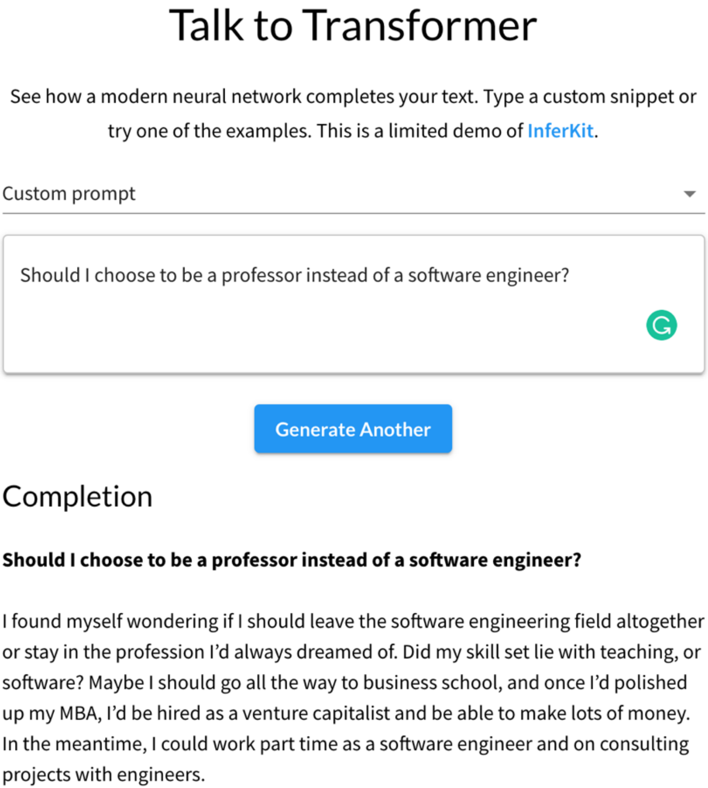
How do I best explain the powers of transformers?

Completion

How do I best explain the powers of transformers?

I think you can make a lot of sense of them by imagining these cars in the middle of this huge, glowing energy field of energy, like you'd see in a Transformers movie. The car is the conduit that fills in the empty spaces, and the energy gives it a little extra oomph. All the effects shots that make the car seem like it's levitating, or traveling faster than normal, are actually taking advantage of this effect. A few of the car's shots also use it for its optical effect, which is pretty

Transformer Language Pre-training



Talk to Transformer

See how a modern neural network completes your text. Type a custom snippet or try one of the examples. This is a limited demo of InferKit.

Custom prompt

Should I choose to be a professor instead of a software engineer?

Completion

Should I choose to be a professor instead of a software engineer?

I found myself wondering if I should leave the software engineering field altogether or stay in the profession I'd always dreamed of. Did my skill set lie with teaching, or software? Maybe I should go all the way to business school, and once I'd polished up my MBA, I'd be hired as a venture capitalist and be able to make lots of money. In the meantime, I could work part time as a software engineer and on consulting projects with engineers.

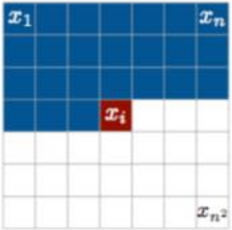
Why Next-Token Prediction Can Achieve AGI

After the break

After the break: CNN for speech analysis

Overview

- We've already looked at a few autoregressive models in this course:
  - Neural language models from Lecture 3
  - RNN language models (and decoders) from Lecture 7
  - Transformer decoders from Lecture 8
- We can push this further, and generate very long sequences.



Treat an image as a very long sequence using raster scan order

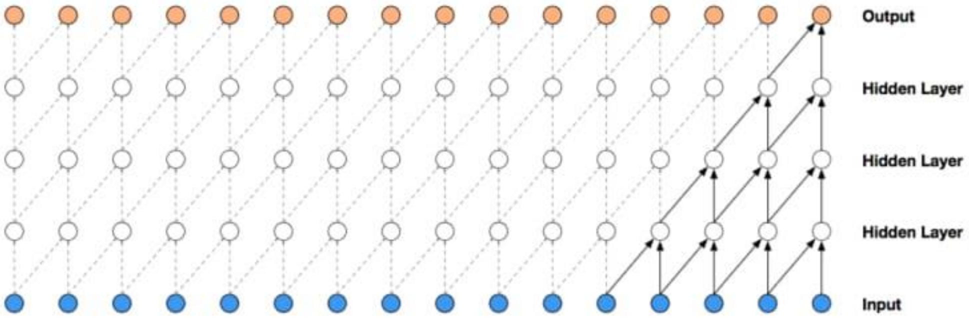
A speech signal can be represented as a waveform, with at least 16,000 samples per second.

- Problem:
  - Training an RNN to generate these sequences requires a sequential computation $> 10{,}000$ time steps.
  - Transformers are too expensive to train on 10,000 time steps.

Causal Convolution

Idea 1: causal convolution

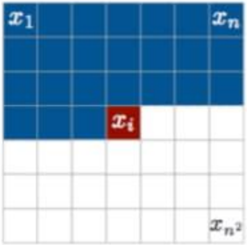
- For RNN language models, we used the training sequence as both the inputs and the outputs to the RNN.
  - We made sure the model was causal: each prediction depended only on inputs earlier in the sequence.
- We can do the same thing using a convolutional architecture.



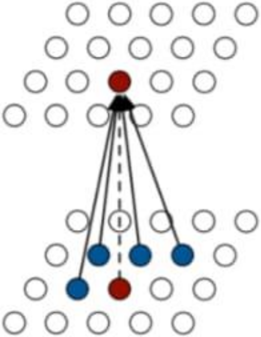
- No for loops! Processing each input sequence just requires a series of convolution operations.

Causal Convolution

Causal convolution for images:

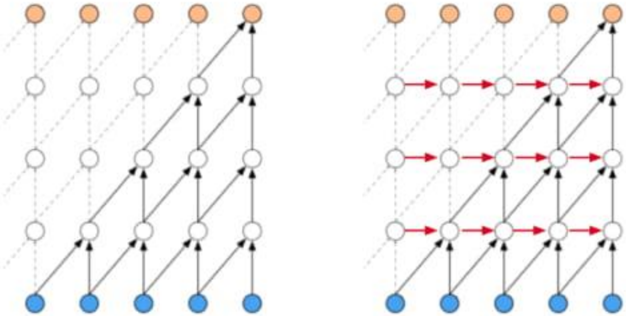


The image is treated as a very long sequence of pixels using raster scan order.



We can restrict the connectivity pattern in each layer to make it causal. This can be implemented by clamping some weights to zero.

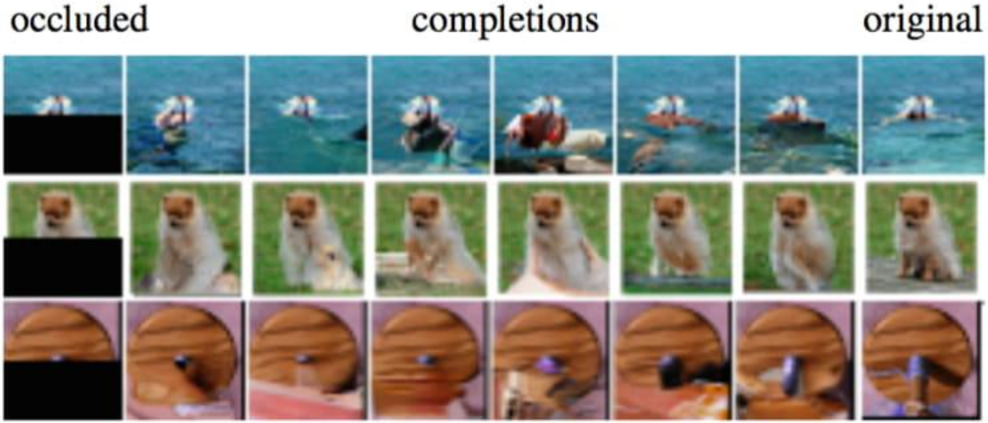
CNN vs. RNN



- We can turn a causal CNN into an RNN by adding recurrent connections. Is this a good idea?
  - The RNN has a memory, so it can use information from all past time steps. The CNN has a limited context.
  - But training the RNN is very expensive since it requires a for loop over time steps. The CNN only requires a series of convolutions.
  - Generating from both models is very expensive, since it requires a for loop. (Whereas generating from a GAN or a reversible model is very fast.)

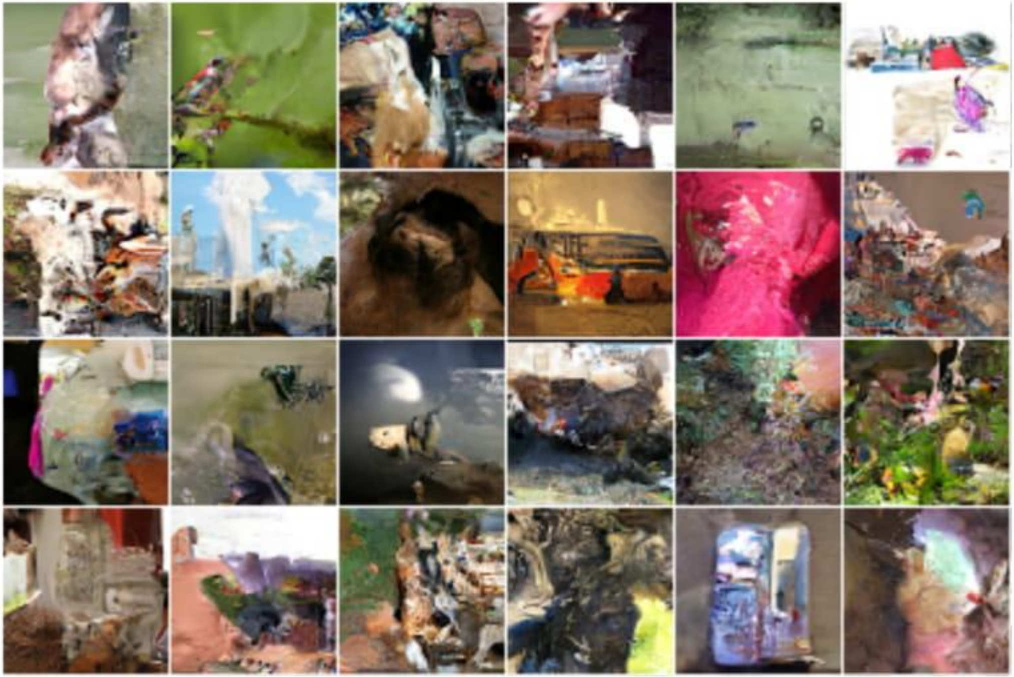
PixelCNN and PixelRNN

- Van den Oord et al., ICML 2016, "Pixel recurrent neural networks"
- This paper introduced two autoregressive models of images: the PixelRNN and the PixelCNN. Both generated amazingly good high-resolution images.
- The output is a softmax over 256 possible pixel intensities.
- Completing an image using an PixelCNN:

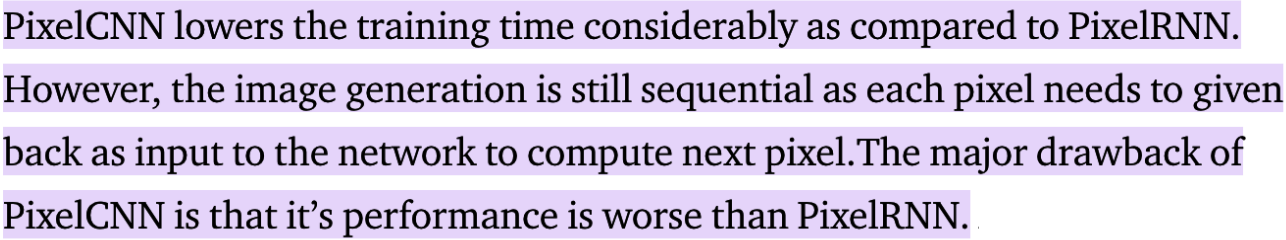


PixelCNN and PixelRNN

Samples from a PixelRNN trained on ImageNet:



PixelCNN and PixelRNN



PixelCNN lowers the training time considerably as compared to PixelRNN. However, the image generation is still sequential as each pixel needs to given back as input to the network to compute next pixel.The major drawback of PixelCNN is that it’s performance is worse than PixelRNN.

Source: Harshit Sharma

Dilated Convolution

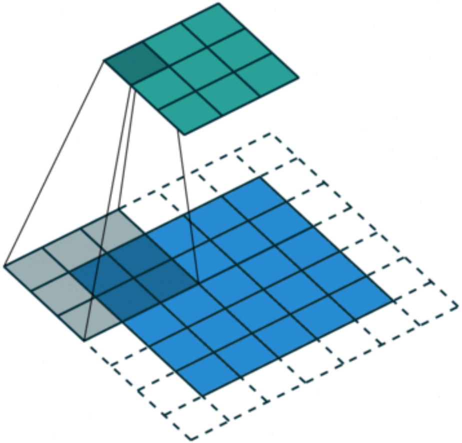
Idea 2: dilated convolution

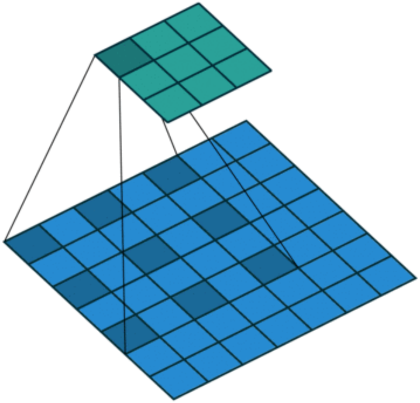
- The advantage of RNNs over CNNs is that their memory lets them learn arbitrary long-distance dependencies.
- But we can dramatically increase a CNN's receptive field using dilated convolution.

Dilated Convolution

Idea 2: dilated convolution

- The advantage of RNNs over CNNs is that their memory lets them learn arbitrary long-distance dependencies.
- But we can dramatically increase a CNN's receptive field using dilated convolution.

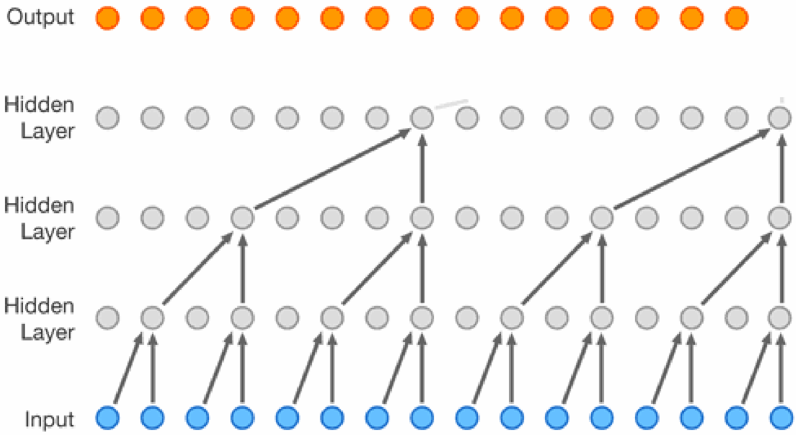




Dilated Convolution

Idea 2: dilated convolution

- The advantage of RNNs over CNNs is that their memory lets them learn arbitrary long-distance dependencies.
- But we can dramatically increase a CNN's receptive field using dilated convolution.

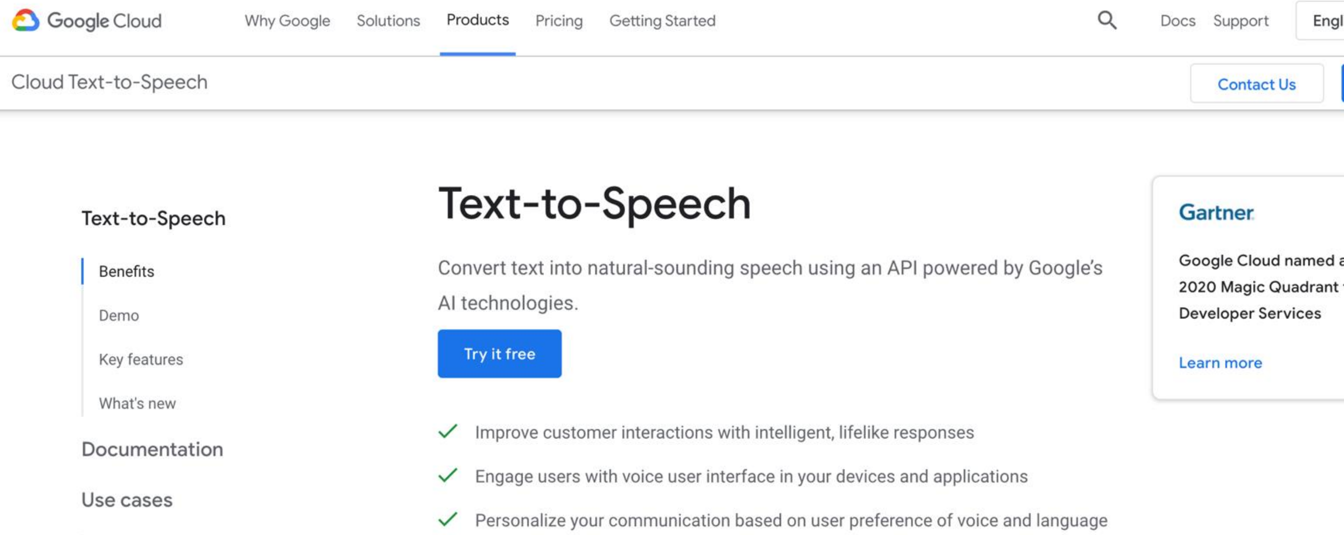


WaveNet

- WaveNet is an autoregressive model for raw audio based on causal dilated convolutions.
  - van den Oord et al., 2016. "WaveNet: a generative model for raw audio".
- Audio needs to be sampled at at least 16k frames per second for good quality. So the sequences are very long.
- WaveNet uses dilations of $1, 2, \ldots, 512$, so each unit at the end of this block as a receptive field of length 1024, or 64 milliseconds.
- It stacks several of these blocks, so the total context length is about 300 milliseconds.
- `https://deepmind.com/blog/wavenet-generative-model-raw-audio/`

WaveNet

Mean Opinion Scores



Text-to-Speech

Convert text into natural-sounding speech using an API powered by Google's AI technologies.

- ✓ Improve customer interactions with intelligent, lifelike responses
- ✓ Engage users with voice user interface in your devices and applications
- ✓ Personalize your communication based on user preference of voice and language

Gartner

Google Cloud named a 2020 Magic Quadrant Developer Services

The Bitter Lesson - Rich Sutton

The biggest lesson that can be read from 70 years of AI research is that general methods that leverage computation are ultimately the most effective, and by a large margin. The ultimate reason for this is Moore's law, or rather its generalization of continued exponentially falling cost per unit of computation.