Transformers

Grigory Sapunov  CTO / Intento

HSE / #12 / 10.12.2019

Self-Attention Neural Networks (SAN): Transformer Architecture

Attention Is All You Need

Ashish Vaswani* Google Brain avaswani@google.com

Noam Shazeer* Google Brain noam@google.com

Niki Parmar* Google Research nikip@google.com

Jakob Uszkoreit* Google Research usz@google.com

Llion Jones* Google Research llion@google.com

Aidan N. Gomez* † University of Toronto aidan@cs.toronto.edu

Łukasz Kaiser* Google Brain lukaszkaiser@google.com

Illia Polosukhin* ‡ illia.polosukhin@gmail.com

Abstract

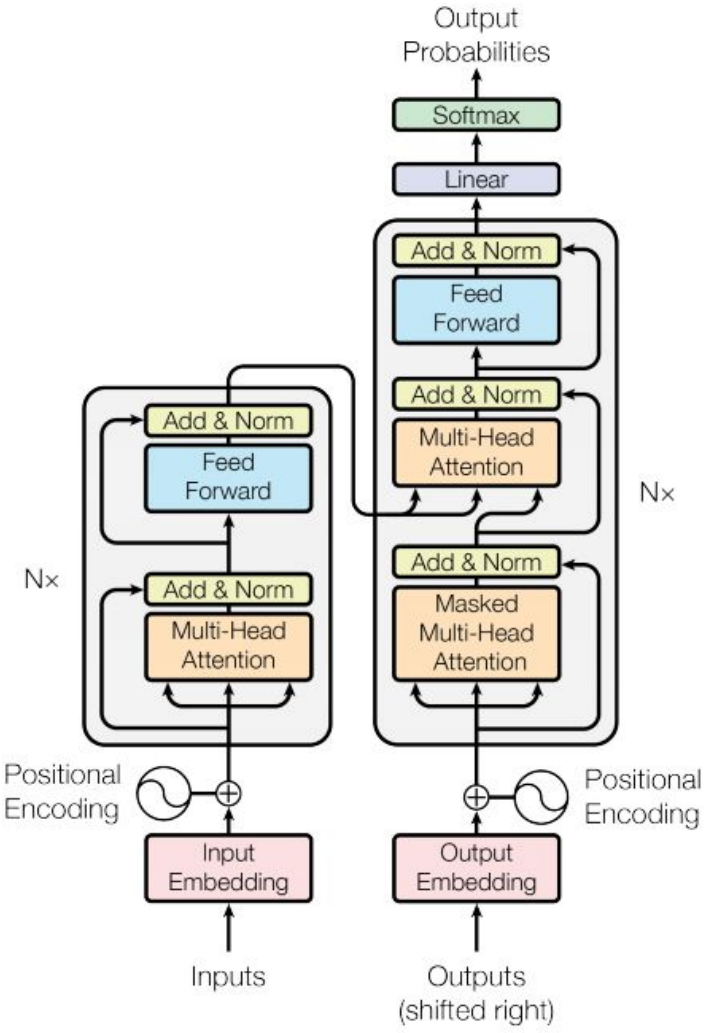
The dominant sequence transduction models are based on complex recurrent or convolutional neural networks that include an encoder and a decoder. The best performing models also connect the encoder and decoder through an attention mechanism. We propose a new simple network architecture, the Transformer, based solely on attention mechanisms, dispensing with recurrence and convolutions entirely. Experiments on two machine translation tasks show these models to be superior in quality while being more parallelizable and requiring significantly less time to train. Our model achieves 28.4 BLEU on the WMT 2014 English-to-German translation task, improving over the existing best results, including ensembles, by over 2 BLEU. On the WMT 2014 English-to-French translation task, our model establishes a new single-model state-of-the-art BLEU score of 41.8 after training for 3.5 days on eight GPUs, a small fraction of the training costs of the best models from the literature. We show that the Transformer generalizes well to other tasks by applying it successfully to English constituency parsing both with large and limited training data.

Attention Is All You Need, https://arxiv.org/abs/1706.03762

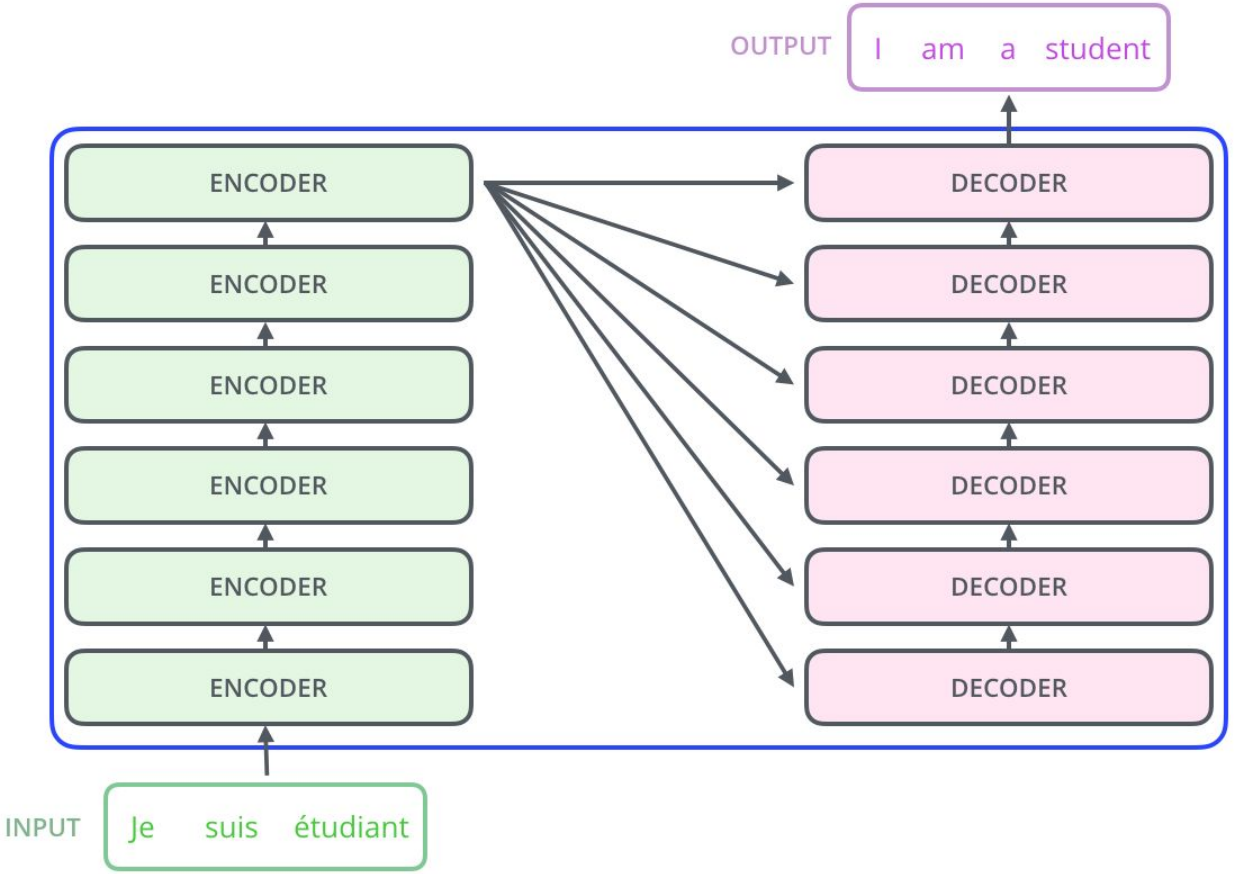
Transformer

A new simple network architecture,
the Transformer:

- Is a Encoder-Decoder architecture
- Based solely on attention mechanisms (no RNN/CNN)
- The major component in the transformer is the unit of multi-head self-attention mechanism.
- Fast: only matrix multiplications
- Strong results on standard WMT datasets



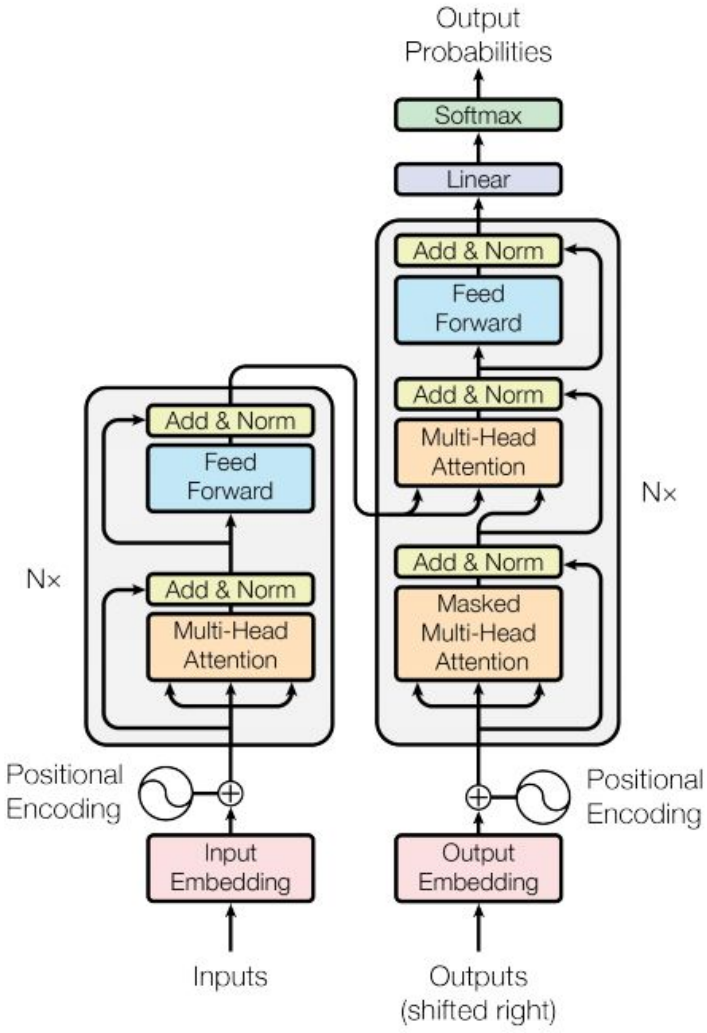
Working pipeline



http://jalammar.github.io/illustrated-transformer/

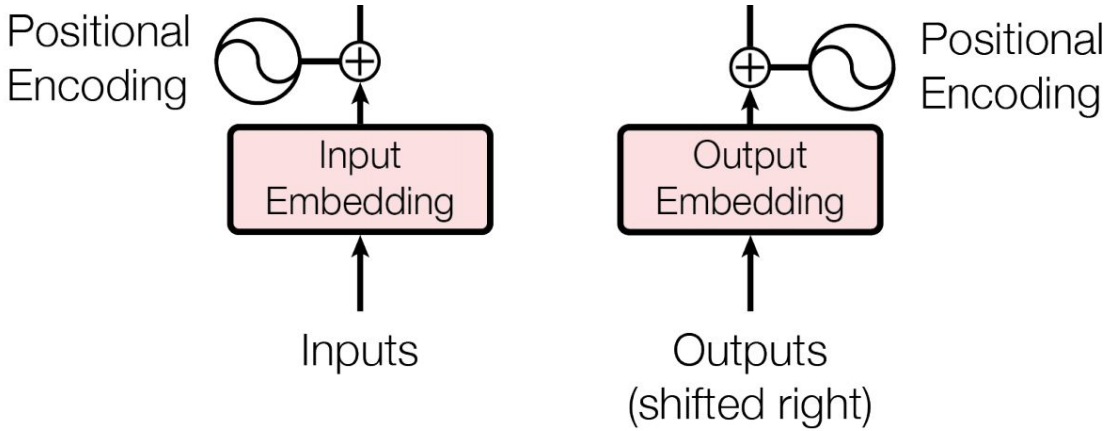
Inputs

The initial inputs to the encoder are the embeddings of the input sequence, and the initial inputs to the decoder are the embeddings of the outputs up to that point.

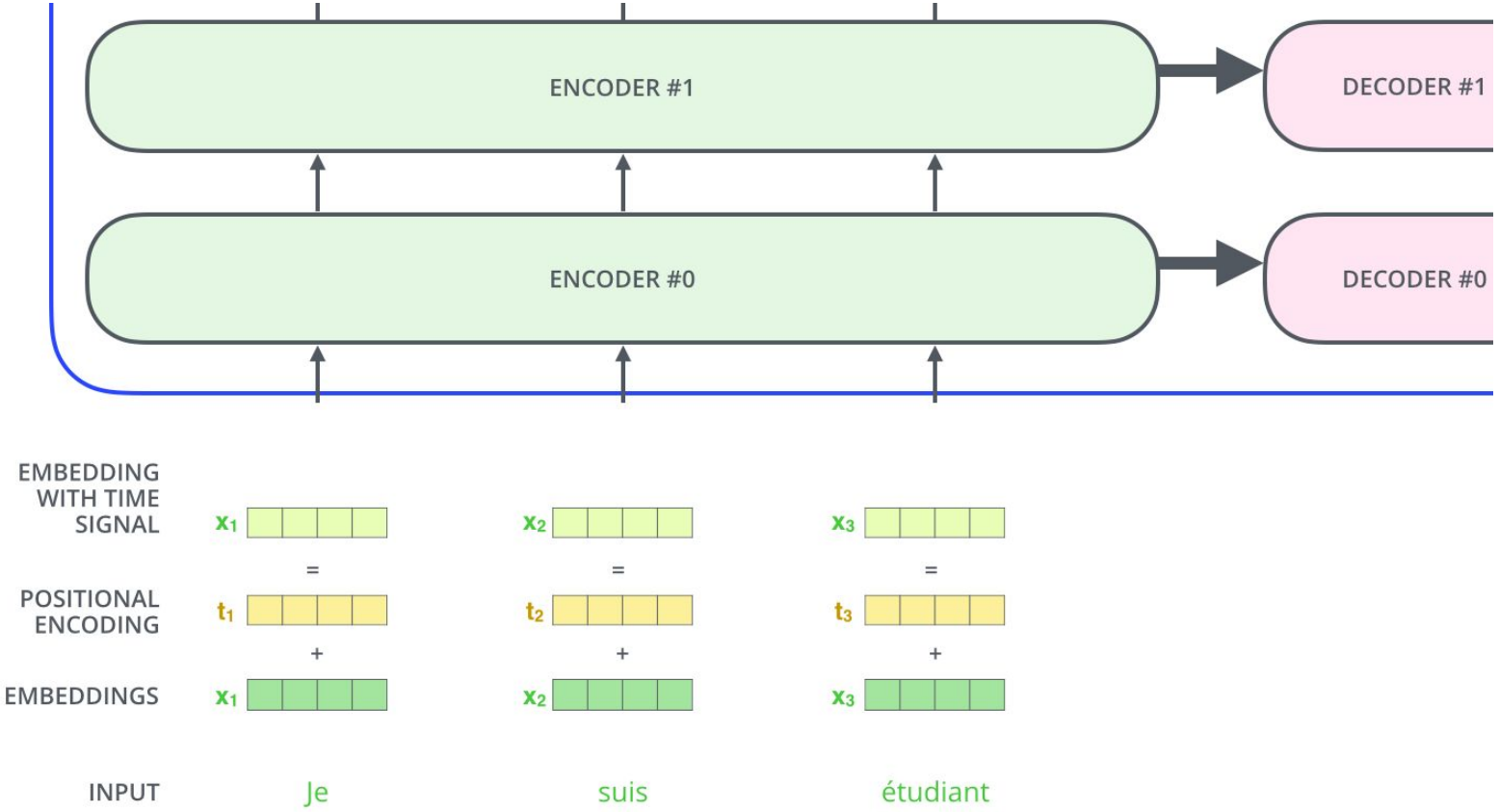


Embeddings

Similarly to other sequence transduction models, we use **learned embeddings** to convert the input tokens and output tokens to vectors of dimension d_{model}. We share the same weight matrix between the two embedding layers.



Embeddings



http://jalammar.github.io/illustrated-transformer/

Positional Encoding

Since our model contains no recurrence and no convolution, in order for the model to make use of the order of the sequence, we must inject some information about the relative or absolute position of the tokens in the sequence. To this end, we add “positional encodings” to the input embeddings at the bottoms of the encoder and decoder stacks. The positional encodings have the same dimension d_{model} as the embeddings, so that the two can be summed.

In this work, we use sine and cosine functions of different frequencies:

$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{\text{model}}})$$

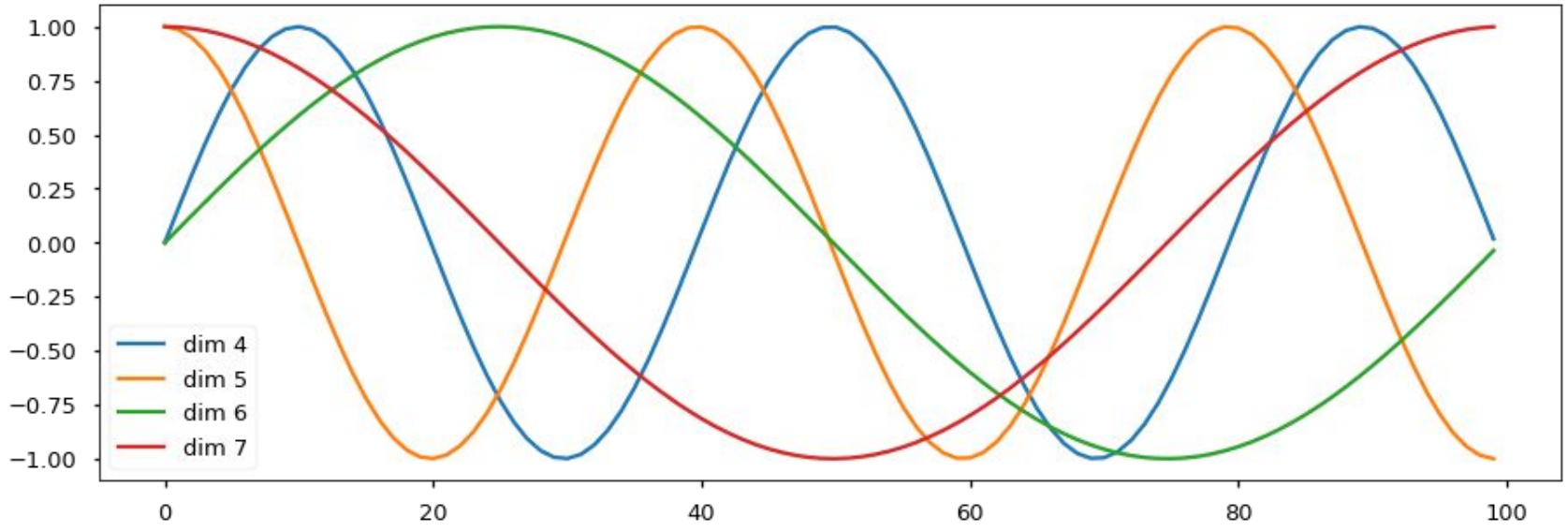
$$PE_{(pos,2i+1)} = cos(pos/10000^{2i/d_{\text{model}}})$$

where *pos* is the position and *i* is the dimension.

Positional Encoding

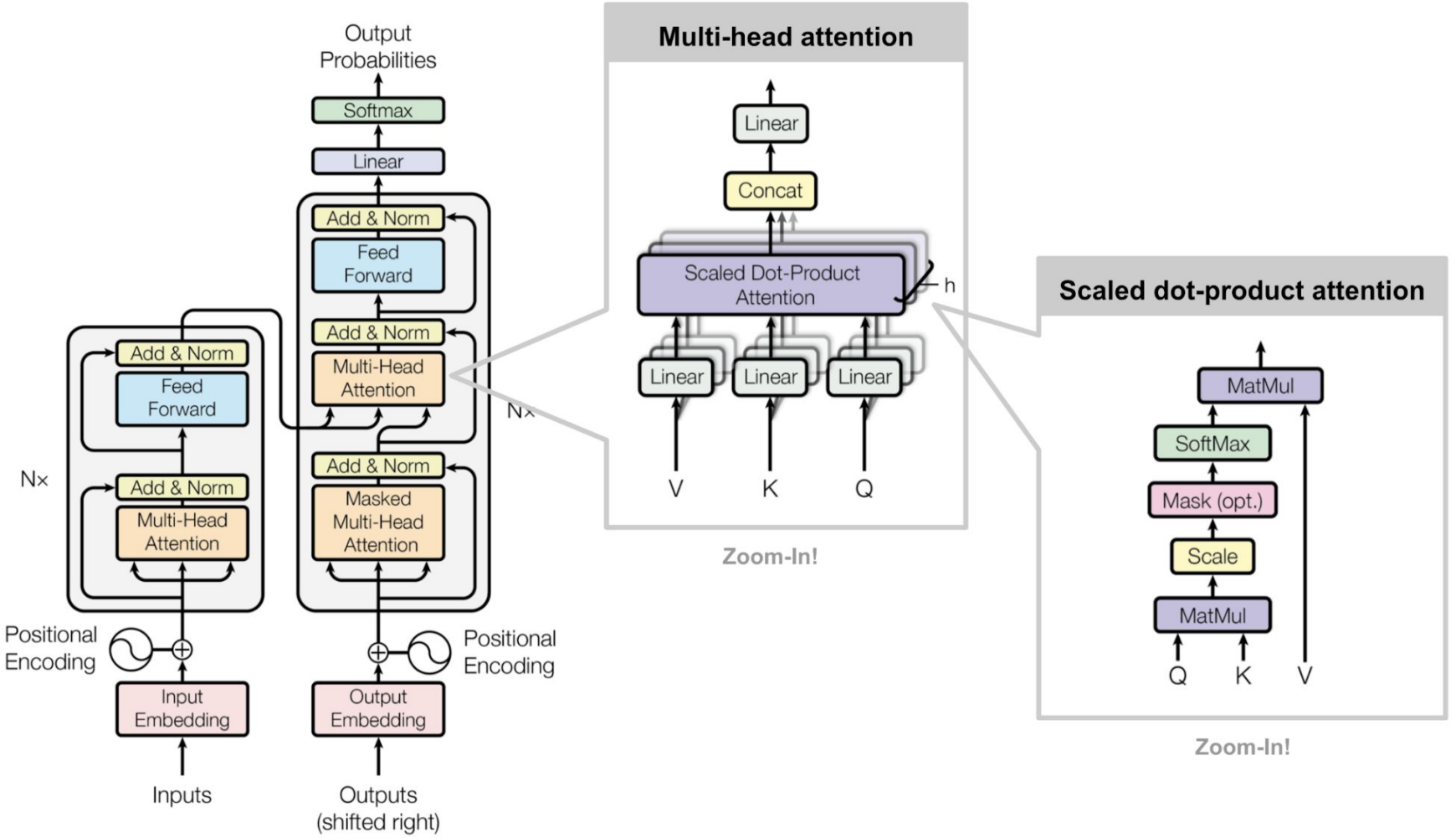
That is, each dimension of the positional encoding corresponds to a sinusoid. The wavelengths form a geometric progression from 2π to $10000 \cdot 2\pi$.

Below the positional encoding will add in a sine wave based on position. The frequency and offset of the wave is different for each dimension.



Positional Encoding

1. **Sinusoidal Position Encoding** (Vaswani et al, 2017, https://arxiv.org/abs/1706.03762)
   Use sine/cosine waves as previously described.
2. **Learned Position Encoding** (Gehring et al, 2017, https://arxiv.org/abs/1705.03122)
   Embed the absolute position of input elements. Can't extrapolate to lengths it has never seen during training.
3. **Relative Position Representations** [currently work the best] (Shaw et al, 2018, https://arxiv.org/abs/1803.02155)
   Model the input as a labeled, directed, fully-connected graph. Learn edge representation.



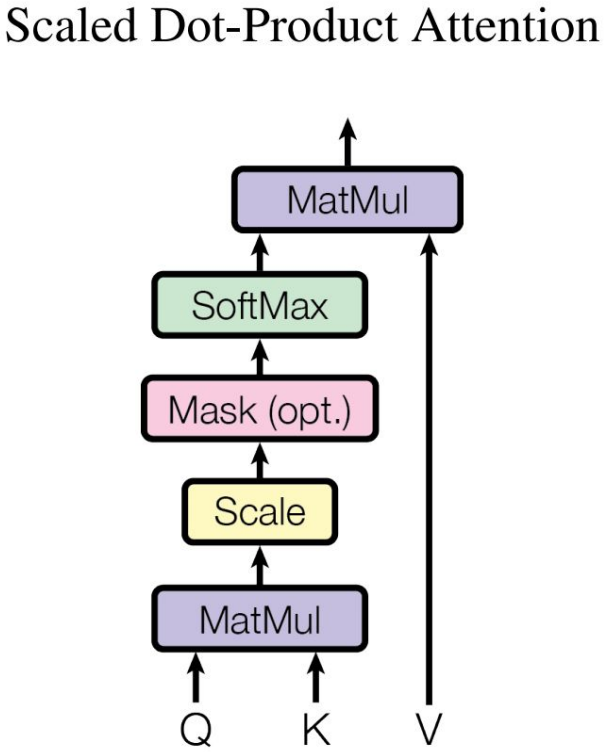
https://lilianweng.github.io/lil-log/2018/06/24/attention-attention.html

Scaled dot-product attention

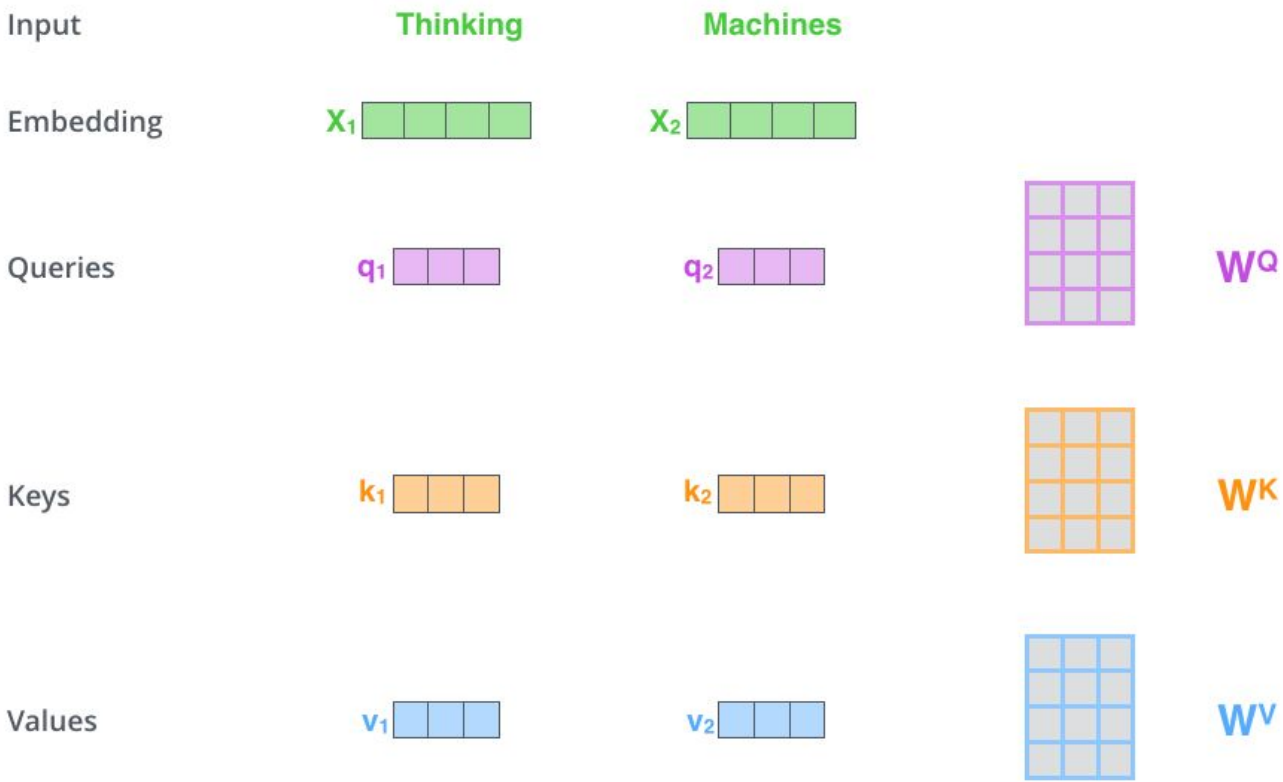
The transformer adopts the scaled dot-product attention: the output is a weighted sum of the values, where the weight assigned to each value is determined by the dot-product of the query with all the keys:

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{n}})\mathbf{V}$$

The input consists of queries and keys of dimension d_k, and values of dimension d_v.



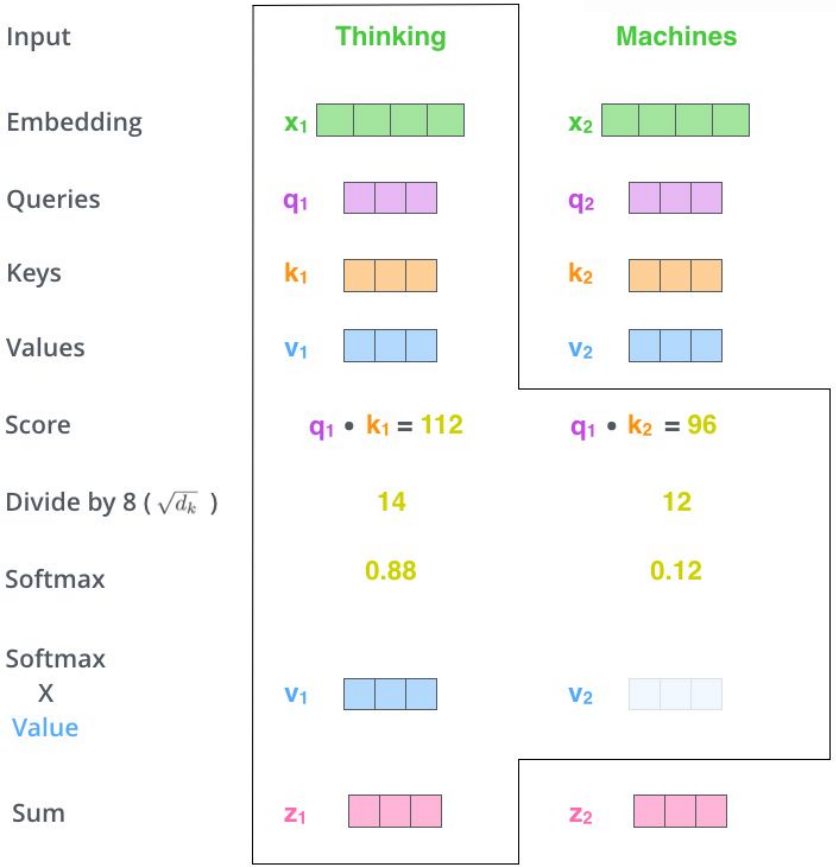
Scaled dot-product attention



http://jalammar.github.io/illustrated-transformer/

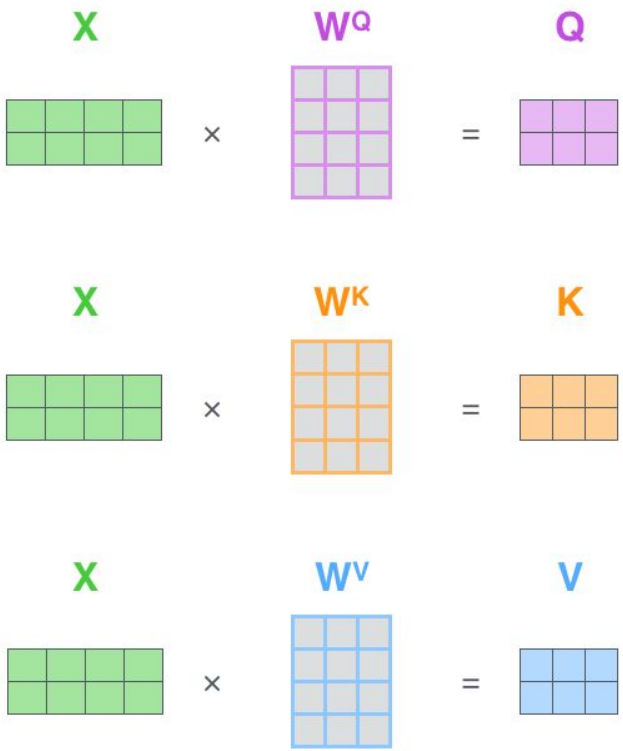
Scaled dot-product attention

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{n}})\mathbf{V}$$



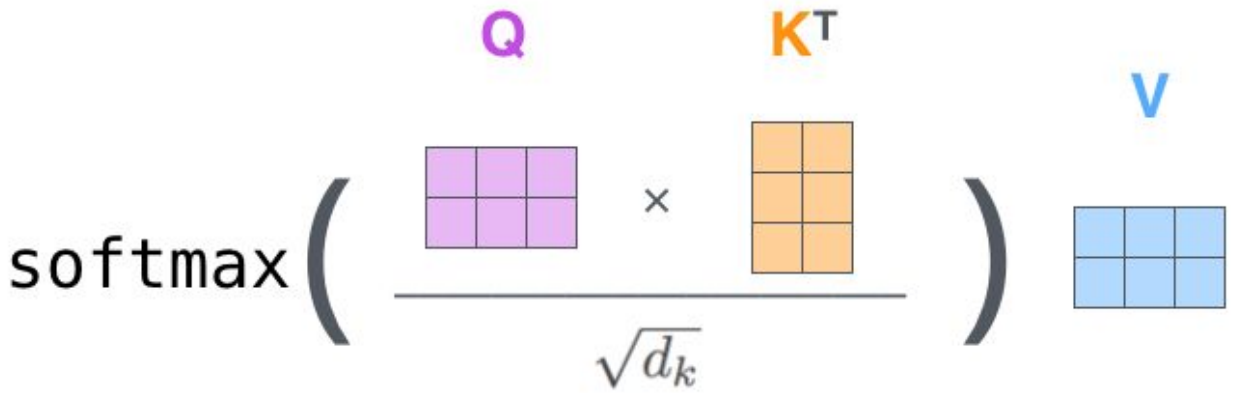
http://jalammar.github.io/illustrated-transformer/

Scaled dot-product attention: Matrix formulation



http://jalammar.github.io/illustrated-transformer/

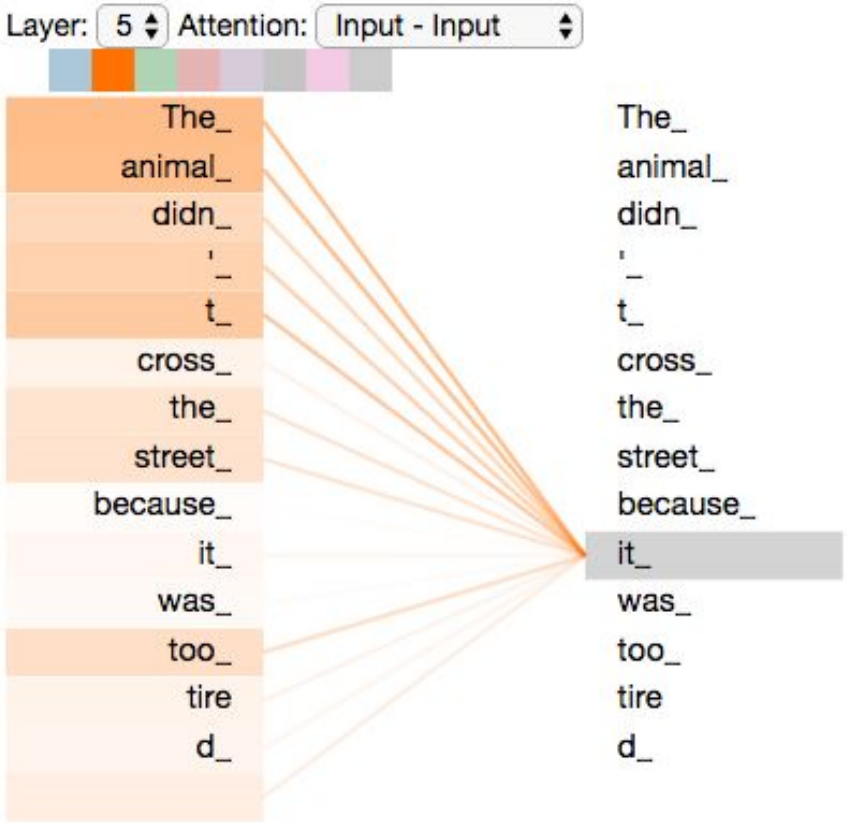
Scaled dot-product attention: Matrix formulation



$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{n}})\mathbf{V}$$

http://jalammar.github.io/illustrated-transformer/

Self-attention example

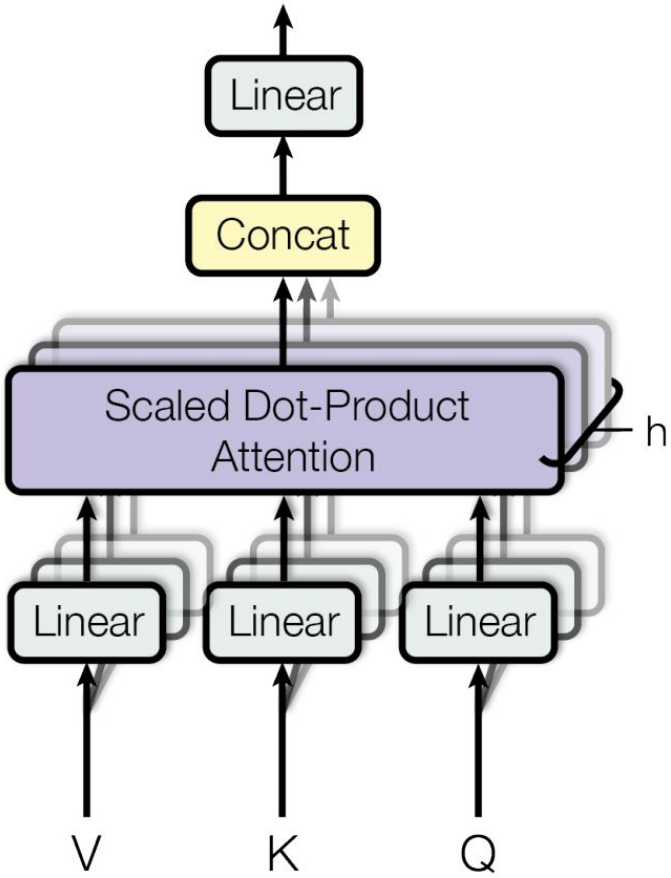


http://jalammar.github.io/illustrated-transformer/

Multi-head self-attention mechanism

Rather than only computing the attention once, the multi-head mechanism runs through the scaled dot-product attention multiple times in parallel. The independent attention outputs are simply concatenated and linearly transformed into the expected dimensions.

According to the paper, “multi-head attention allows the model to jointly attend to information from different representation subspaces at different positions. With a single attention head, averaging inhibits this.”



Multi-head self-attention mechanism

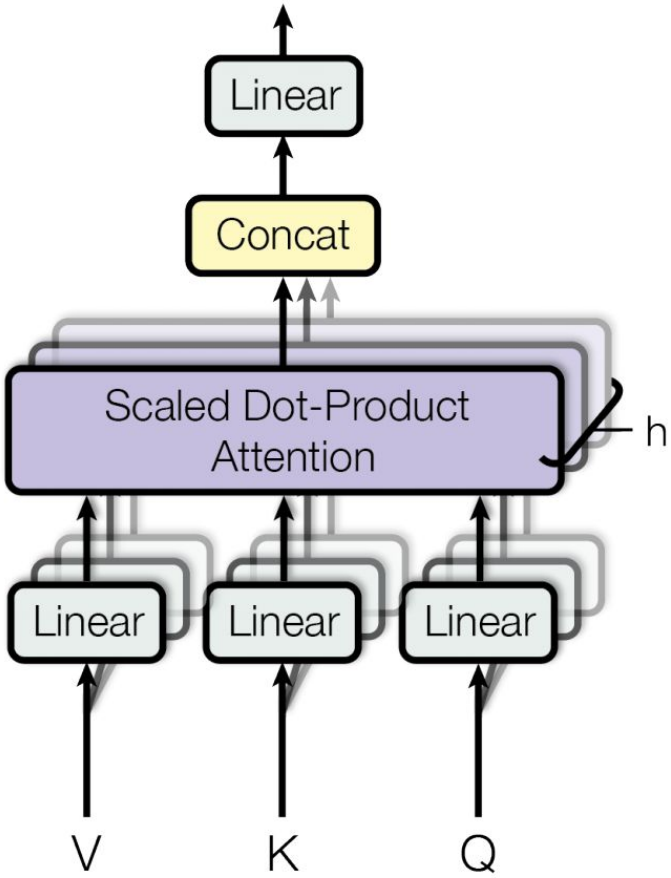
$$\text{MultiHead}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = [\text{head}_1; \ldots; \text{head}_h]\mathbf{W}^O$$
$$\text{where head}_i = \text{Attention}(\mathbf{Q}\mathbf{W}_i^Q, \mathbf{K}\mathbf{W}_i^K, \mathbf{V}\mathbf{W}_i^V)$$

$$W_i^Q \in \mathbb{R}^{d_{\text{model}} \times d_k}, W_i^K \in \mathbb{R}^{d_{\text{model}} \times d_k}, W_i^V \in \mathbb{R}^{d_{\text{model}} \times d_v}$$

$$W^O \in \mathbb{R}^{hd_v \times d_{\text{model}}}$$

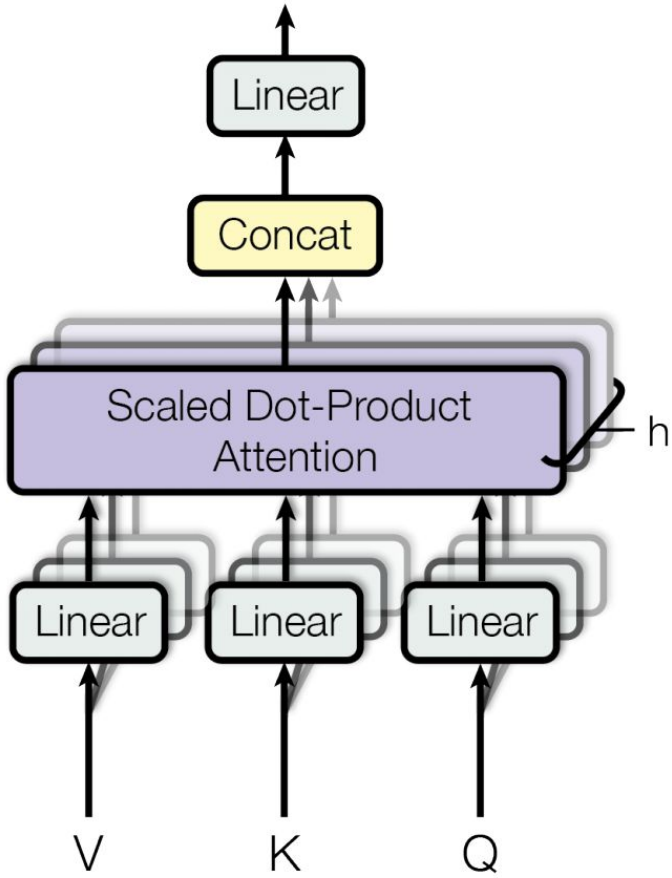
In this work we employ h=8 parallel attention layers, or heads. For each of these we use $d_k=d_v=d_{model}/h=64$.

Due to the reduced dimension of each head, the total computational cost is similar to that of single-head attention with full dimensionality.

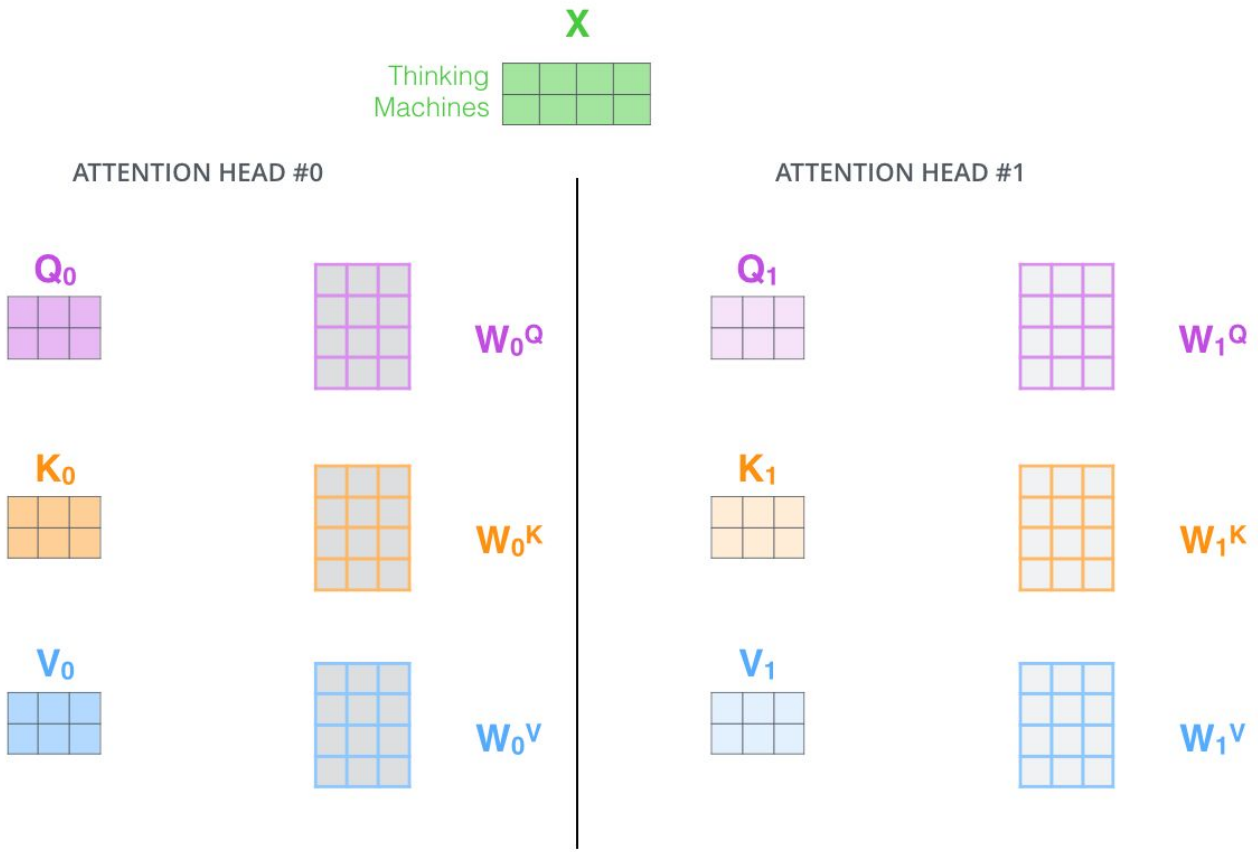


Multi-head self-attention mechanism

Essentially, the Multi-Head Attention is just several attention layers stacked together with different linear transformations of the same input.

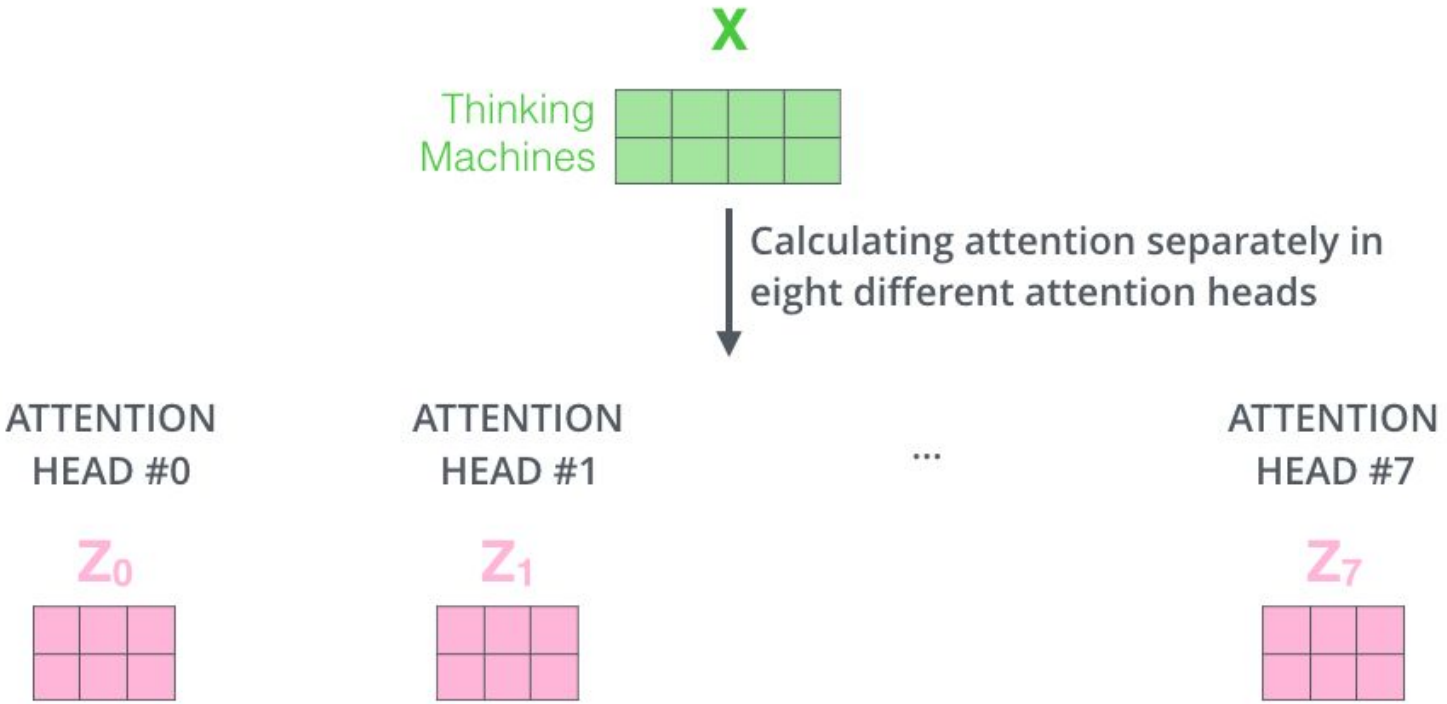


Multi-head self-attention mechanism



http://jalammar.github.io/illustrated-transformer/

Multi-head self-attention mechanism



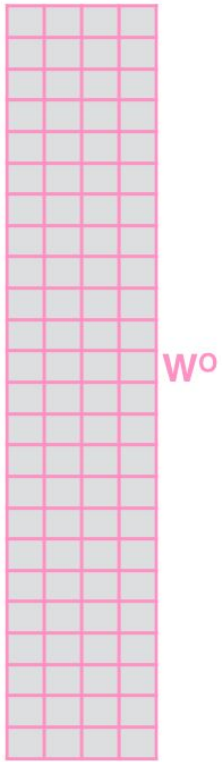
http://jalammar.github.io/illustrated-transformer/

Multi-head self-attention mechanism

1) Concatenate all the attention heads

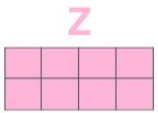
2) Multiply with a weight matrix W^O that was trained jointly with the model

X



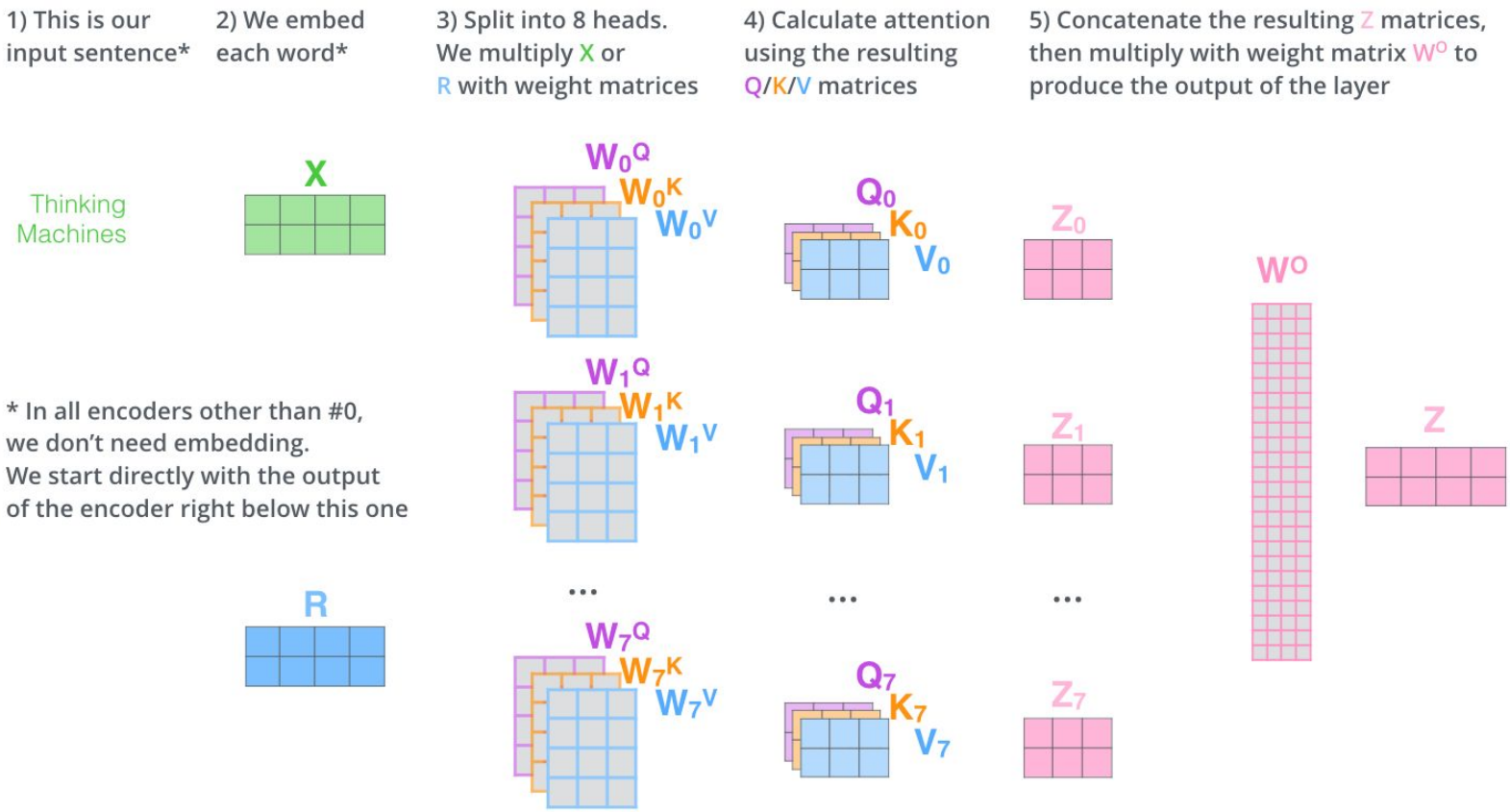
3) The result would be the Z matrix that captures information from all the attention heads. We can send this forward to the FFNN

=



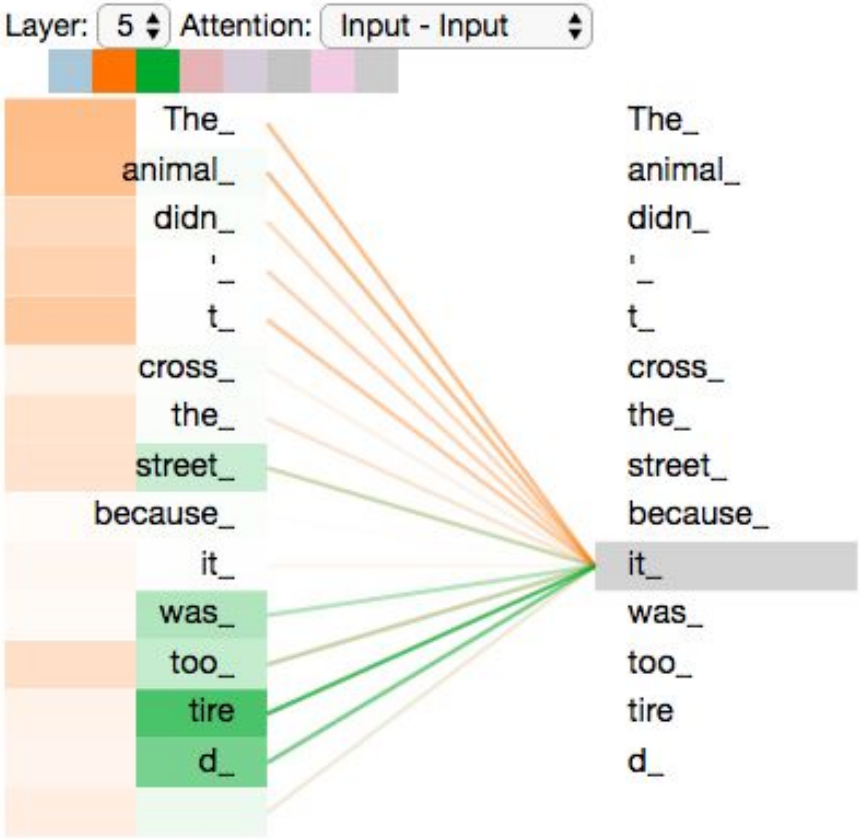
http://jalammar.github.io/illustrated-transformer/

Multi-head self-attention mechanism



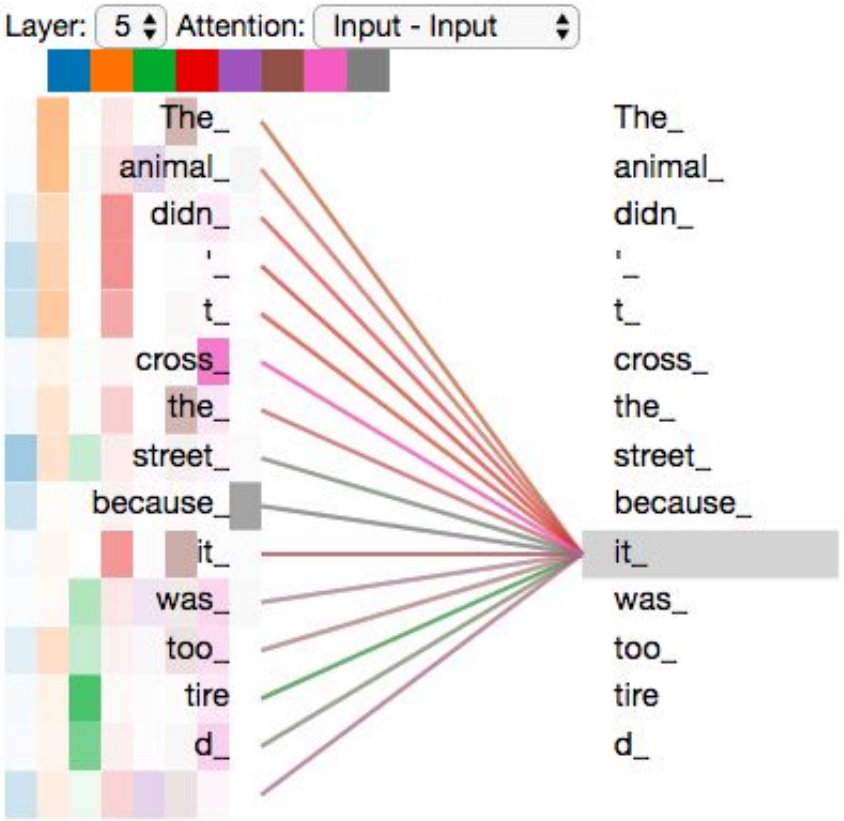
http://jalammar.github.io/illustrated-transformer/

Multi-head self-attention example (2 heads shown)



http://jalammar.github.io/illustrated-transformer/

Multi-head self-attention example (8 heads shown)



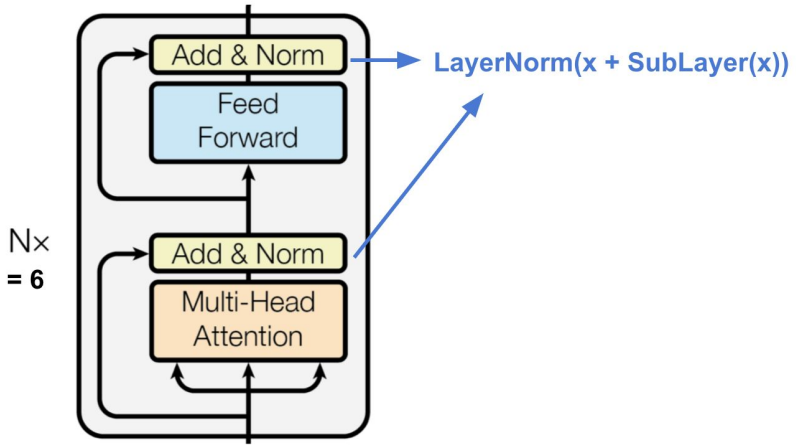
http://jalammar.github.io/illustrated-transformer/

Encoder

The encoder is composed of a stack of N=6 identical layers.

Each layer has two sub-layers. The first is a multi-head self-attention mechanism, and the second is a simple, position-wise fully connected feed-forward network.

To facilitate residual connections, all sub-layers in the model, as well as the embedding layers, produce outputs of dimension d_{model}=512.



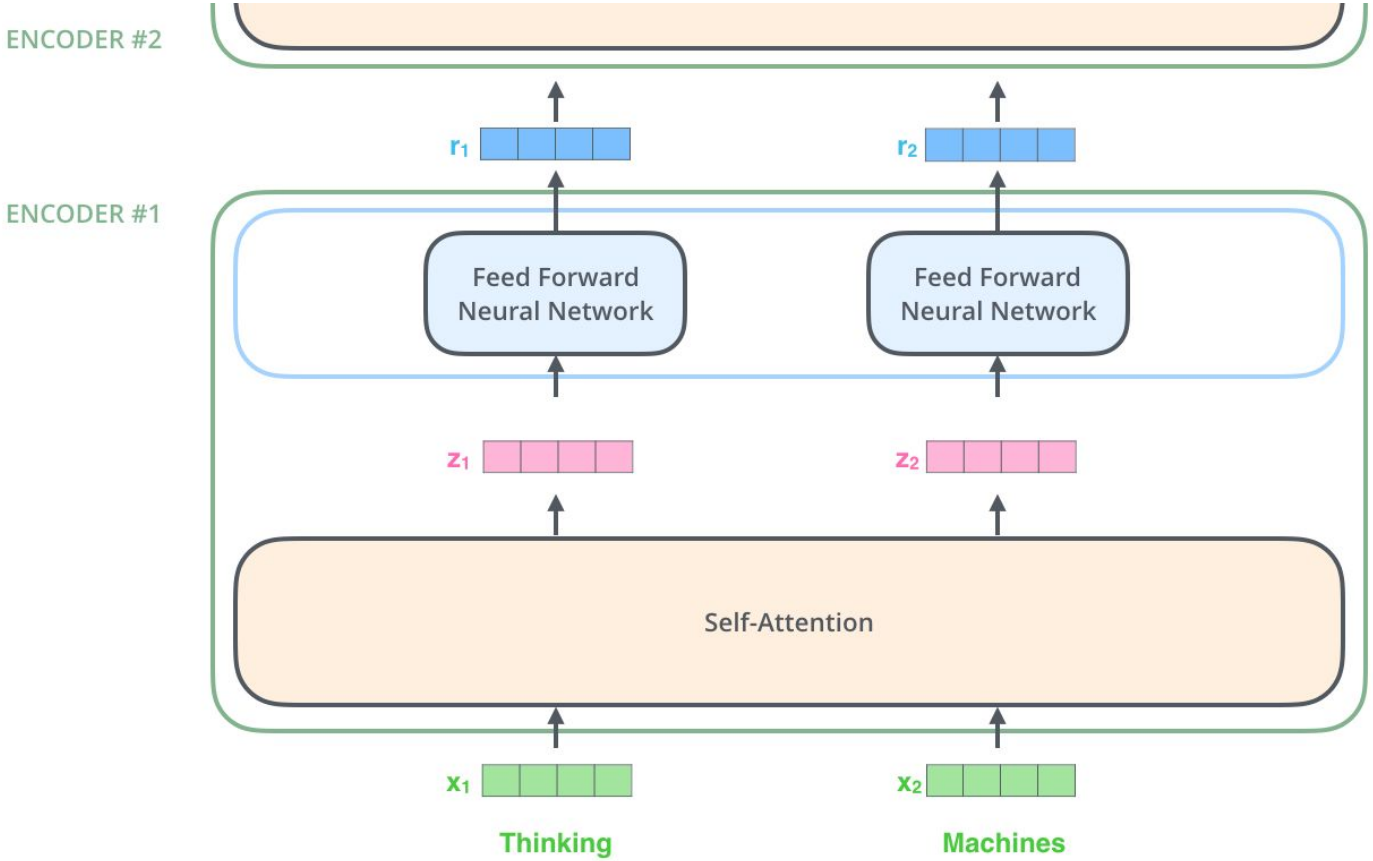
Position-wise Feed-Forward Networks

Each of the layers in our encoder and decoder contains a fully connected feed-forward network, which is applied to each position separately and identically. This consists of two linear transformations with a ReLU activation in between.

$$\mathrm{FFN}(x) = \max(0, xW_1 + b_1)W_2 + b_2$$

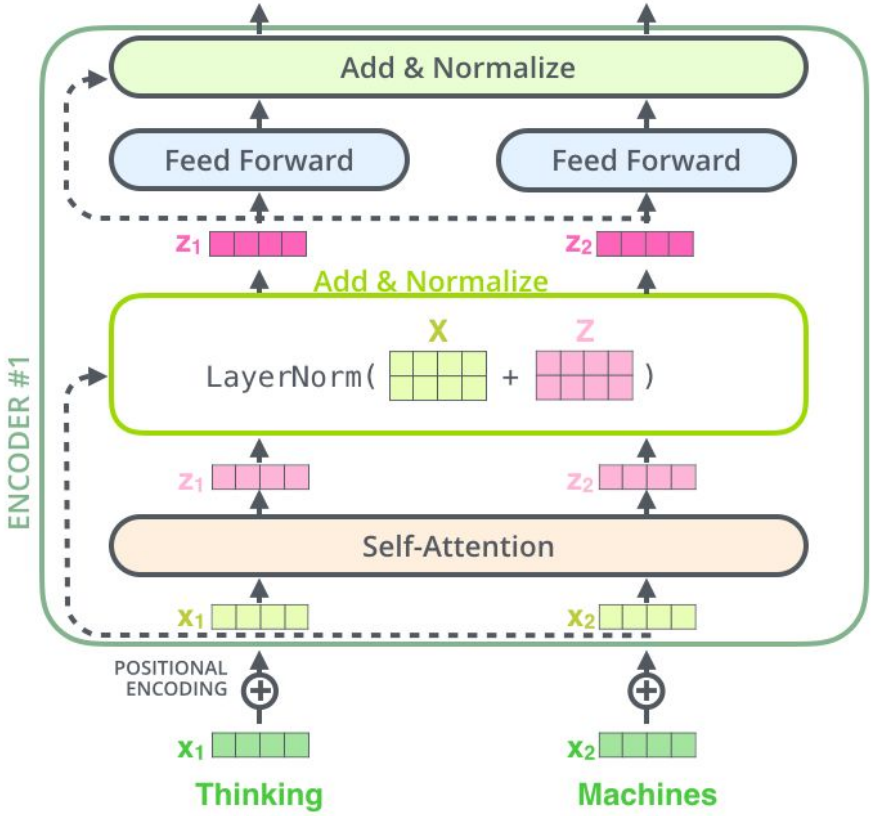
The dimensionality of input and output is d_{model}=512, and the inner-layer has dimensionality d_{ff}=2048.

Encoder



http://jalammar.github.io/illustrated-transformer/

Encoder



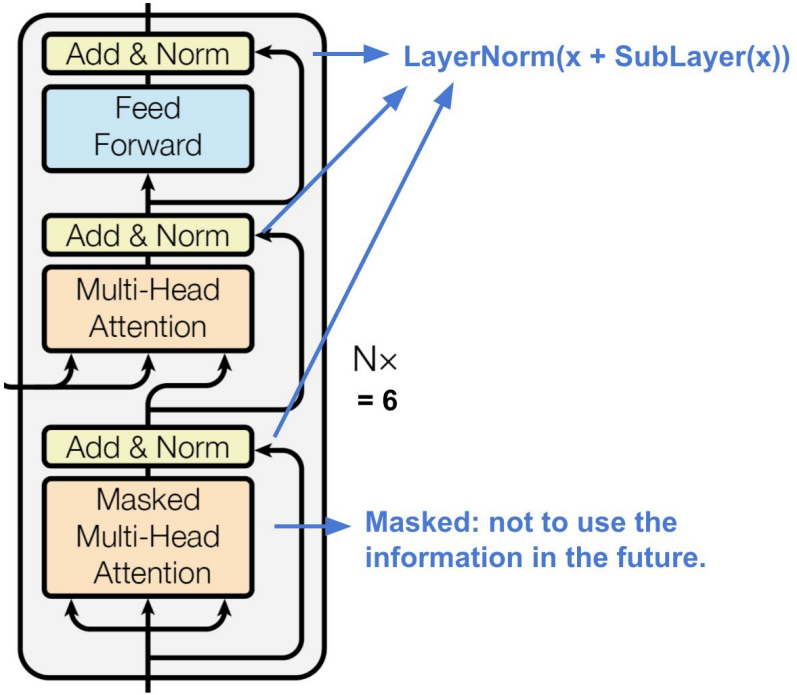
http://jalammar.github.io/illustrated-transformer/

Decoder

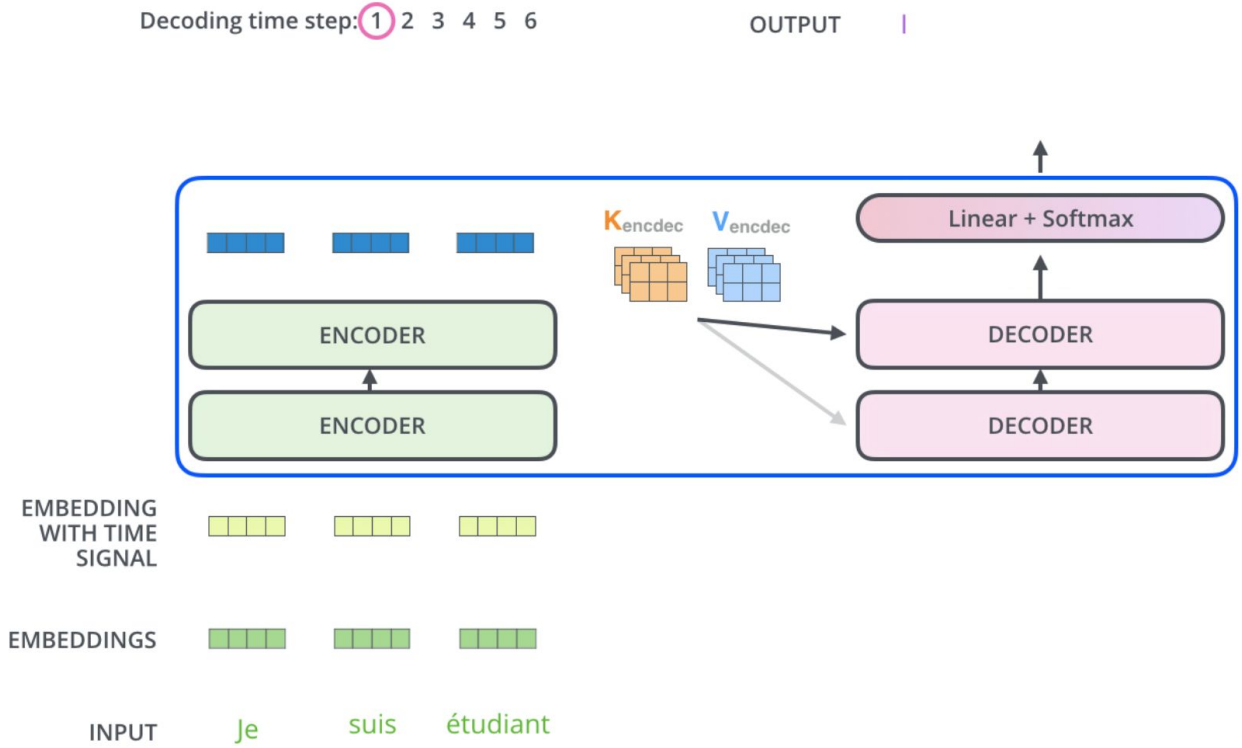
The decoder is also composed of a stack of N=6 identical layers.

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.

The Multi-Head Attention layer works just like multiheaded self-attention, except it creates its Queries matrix from the layer below it, and takes the Keys and Values matrix from the output of the encoder stack.



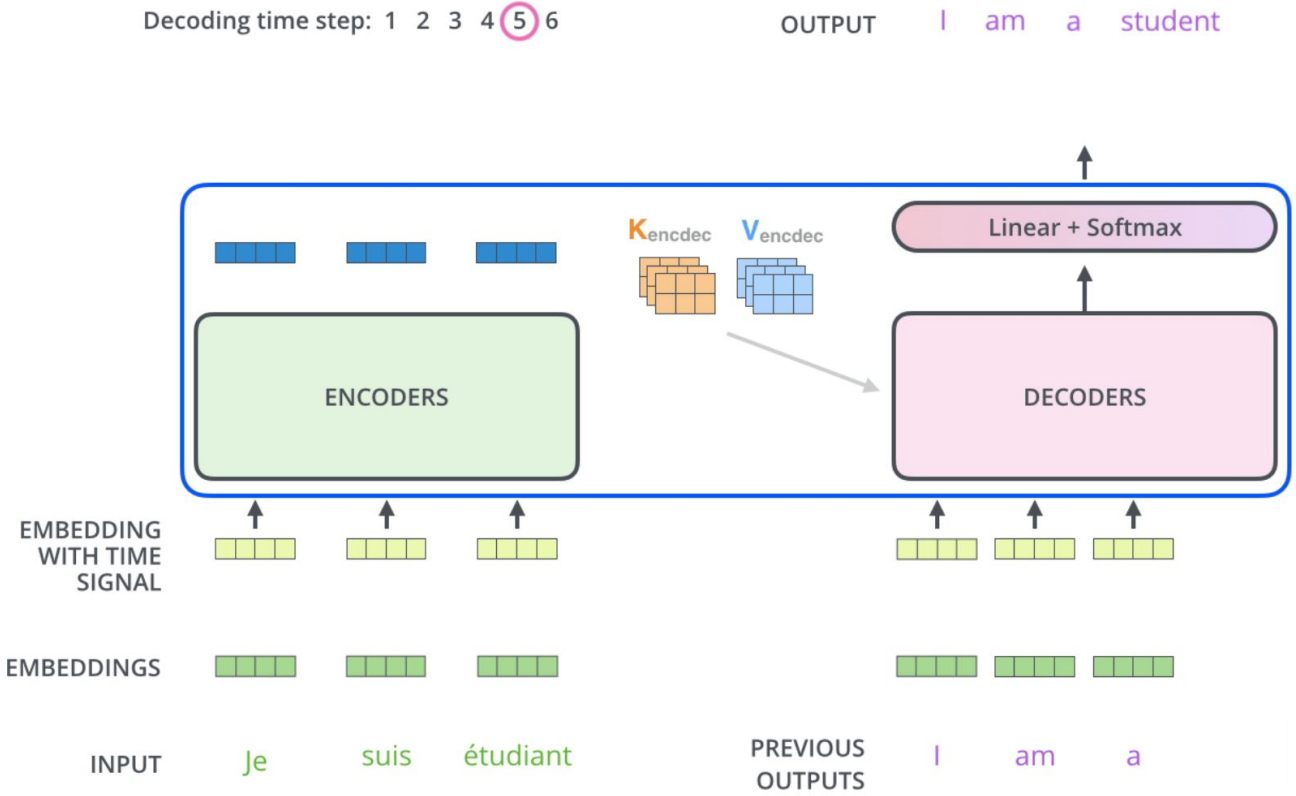
Decoder



After finishing the encoding phase, we begin the decoding phase. Each step in the decoding phase outputs an element from the output sequence (the English translation sentence in this case).

http://jalammar.github.io/illustrated-transformer/

Decoder



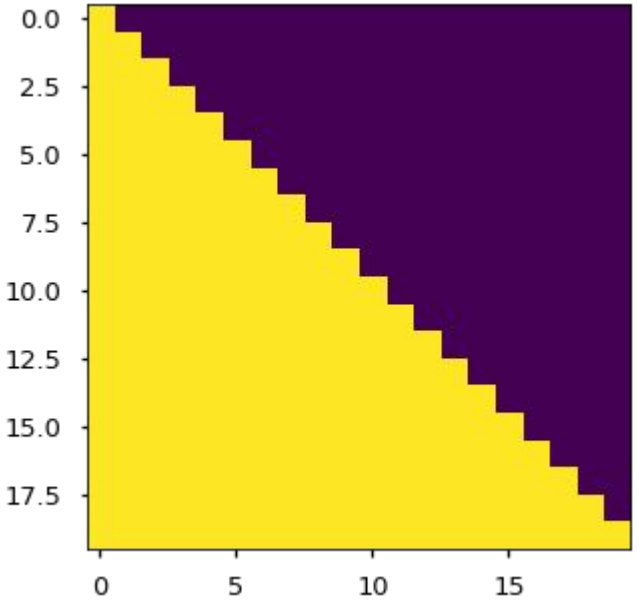
http://jalammar.github.io/illustrated-transformer/

Masking in Decoder

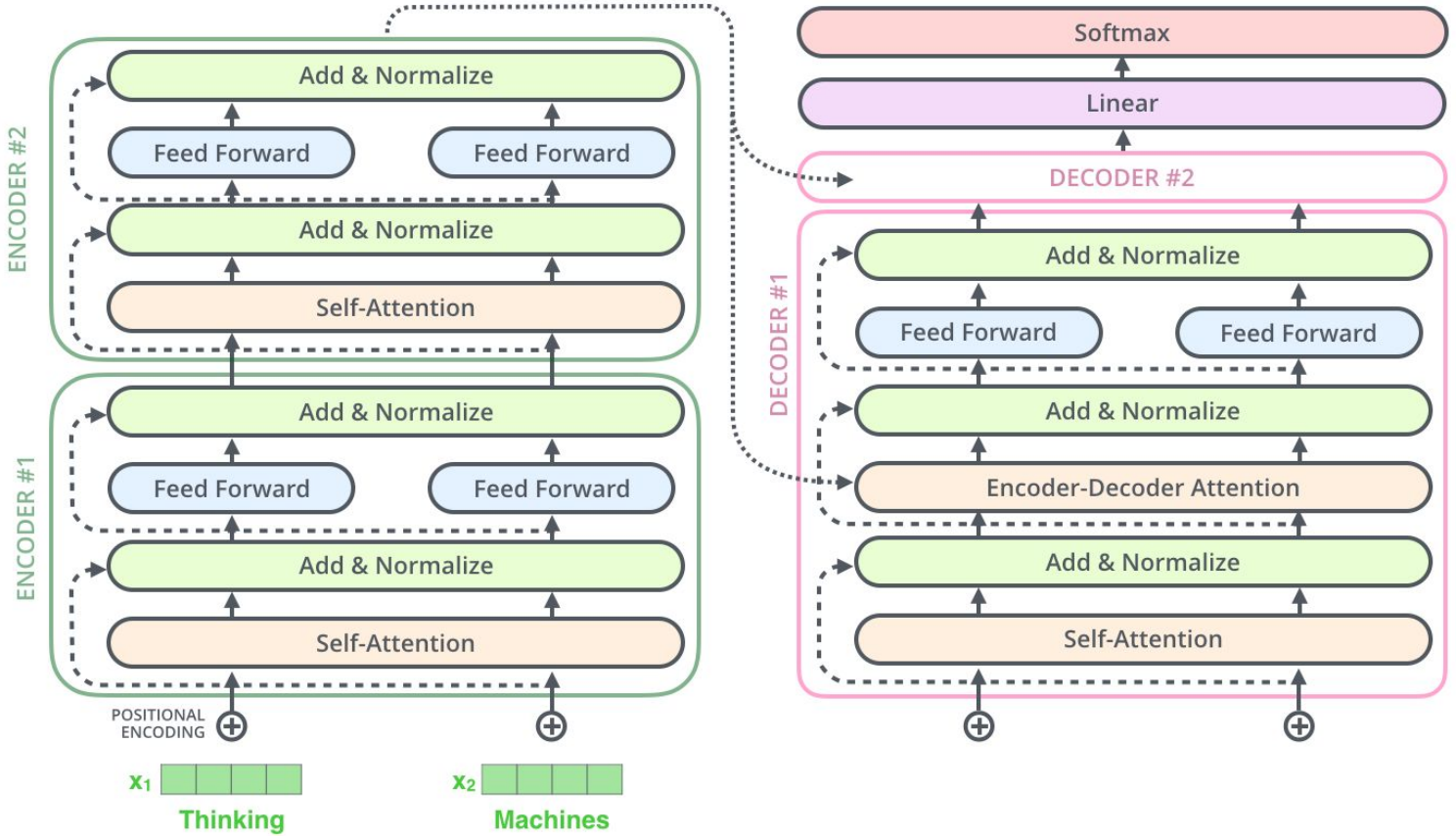
We also modify the self-attention sub-layer in the decoder stack to prevent positions from attending to subsequent positions.

In the decoder, the self-attention layer is only allowed to attend to earlier positions in the output sequence. This is done by masking future positions (setting them to -inf) before the softmax step in the self-attention calculation.

The attention mask shows the position each tgt word (row) is allowed to look at (column). Words are blocked for attending to future words during training.



Working pipeline



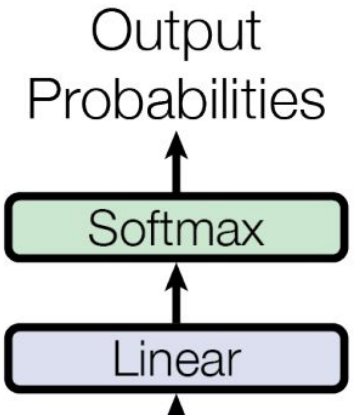
http://jalammar.github.io/illustrated-transformer/

Attention in the model

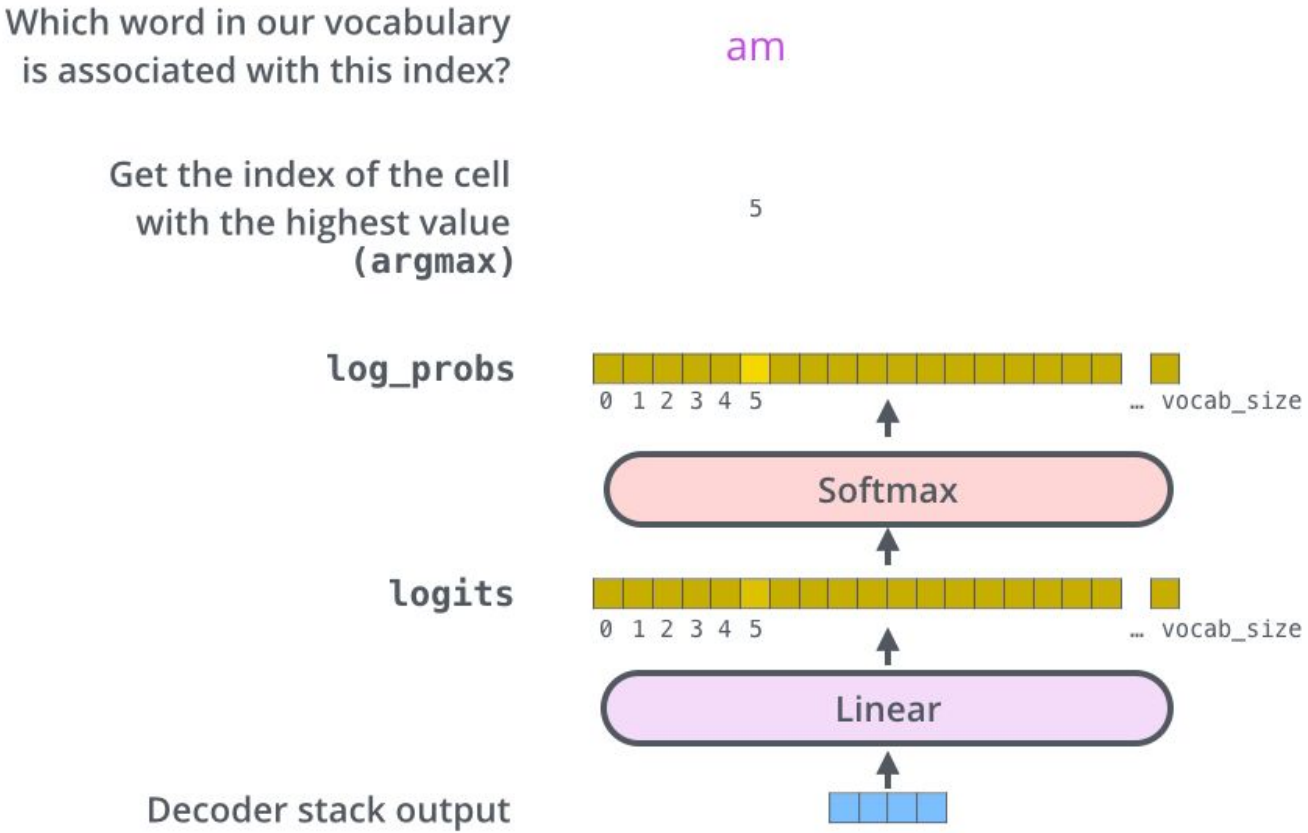
1. In “encoder-decoder attention” layers, the queries come from the previous decoder layer, and the memory keys and values come from the output of the encoder. This allows every position in the decoder to attend over all positions in the input sequence. This mimics the typical encoder-decoder attention mechanisms in sequence-to-sequence models.
2. The encoder contains self-attention layers. In a self-attention layer all of the keys, values and queries come from the same place, in this case, the output of the previous layer in the encoder. Each position in the encoder can attend to all positions in the previous layer of the encoder.
3. Similarly, self-attention layers in the decoder allow each position in the decoder to attend to all positions in the decoder up to and including that position. We need to prevent leftward information flow in the decoder to preserve the auto-regressive property (by masking out all values in the input of the softmax which correspond to illegal connections).

The Final Linear and Softmax Layer

We use the usual learned linear transformation and softmax function to convert the decoder output to predicted next-token probabilities.



The Final Linear and Softmax Layer



http://jalammar.github.io/illustrated-transformer/

Attention visualization

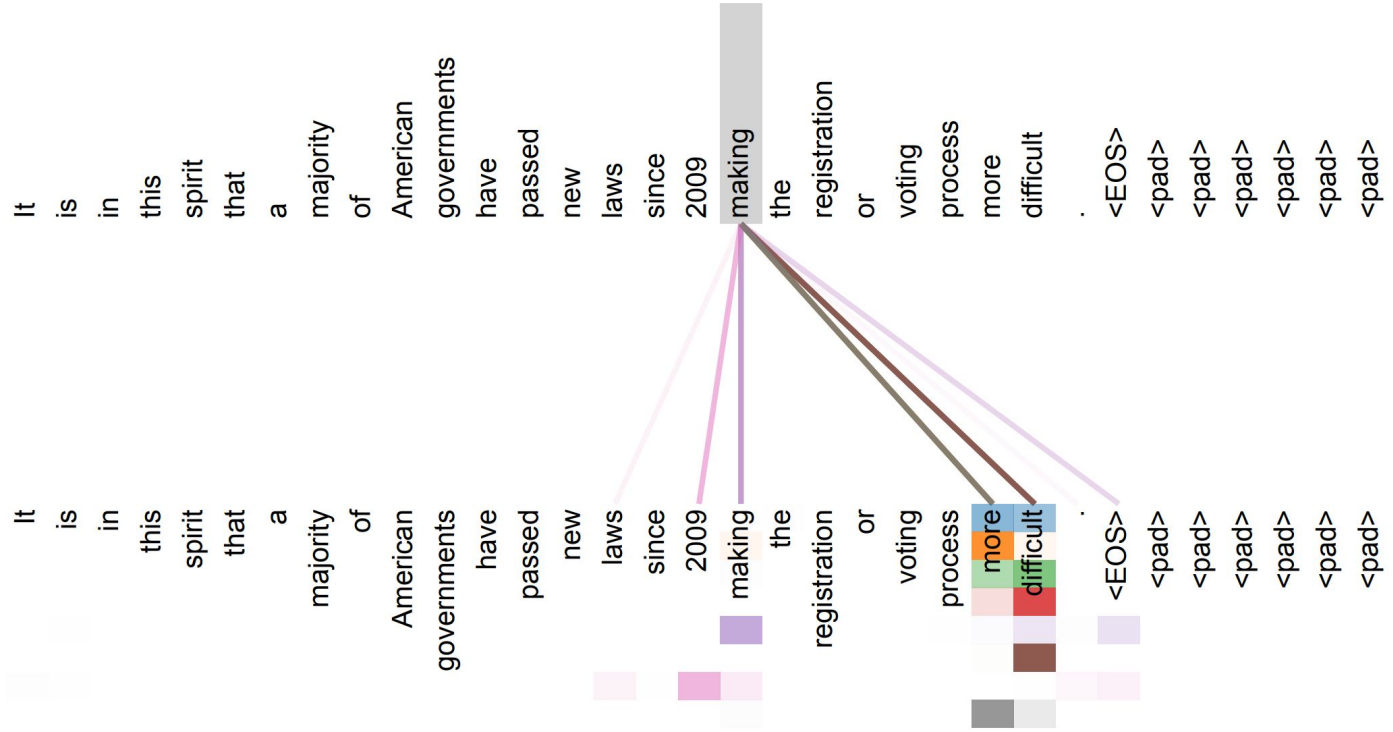


Figure 3: An example of the attention mechanism following long-distance dependencies in the encoder self-attention in layer 5 of 6. Many of the attention heads attend to a distant dependency of the verb ‘making’, completing the phrase ‘making...more difficult’. Attentions here shown only for the word ‘making’. Different colors represent different heads. Best viewed in color.

Applying the Transformer to machine translation

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

Different architectures for NMT

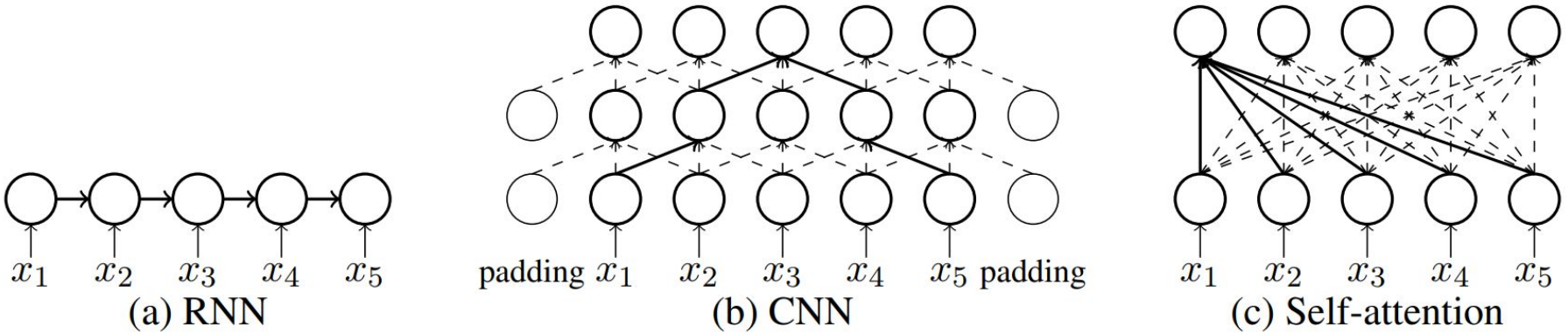


Figure 1: Architectures of different neural networks in NMT.

Why Self-Attention? A Targeted Evaluation of Neural Machine Translation Architectures, https://arxiv.org/abs/1808.08946

Resources

- The Annotated Transformer
  http://nlp.seas.harvard.edu/2018/04/03/attention.html
- Attention? Attention!
  https://lilianweng.github.io/lil-log/2018/06/24/attention-attention.html
- The Illustrated Transformer
  http://jalammar.github.io/illustrated-transformer/
- Paper Dissected: “Attention is All You Need” Explained
  http://mlexplained.com/2017/12/29/attention-is-all-you-need-explained/
- The Transformer – Attention is all you need.
  https://mchromiak.github.io/articles/2017/Sep/12/Transformer-Attention-is-all-you-need/
- When Recurrent Models Don't Need to be Recurrent
  https://bair.berkeley.edu/blog/2018/08/06/recurrent/
- Self-Attention Mechanisms in Natural Language Processing,
  https://www.alibabacloud.com/blog/self-attention-mechanisms-in-natural-language-processing_593968

Code

- https://github.com/tensorflow/tensor2tensor
- Running the Transformer with Tensor2Tensor https://cloud.google.com/tpu/docs/tutorials/transformer
- https://ai.googleblog.com/2017/06/accelerating-deep-learning-research.html
- https://github.com/pytorch/fairseq

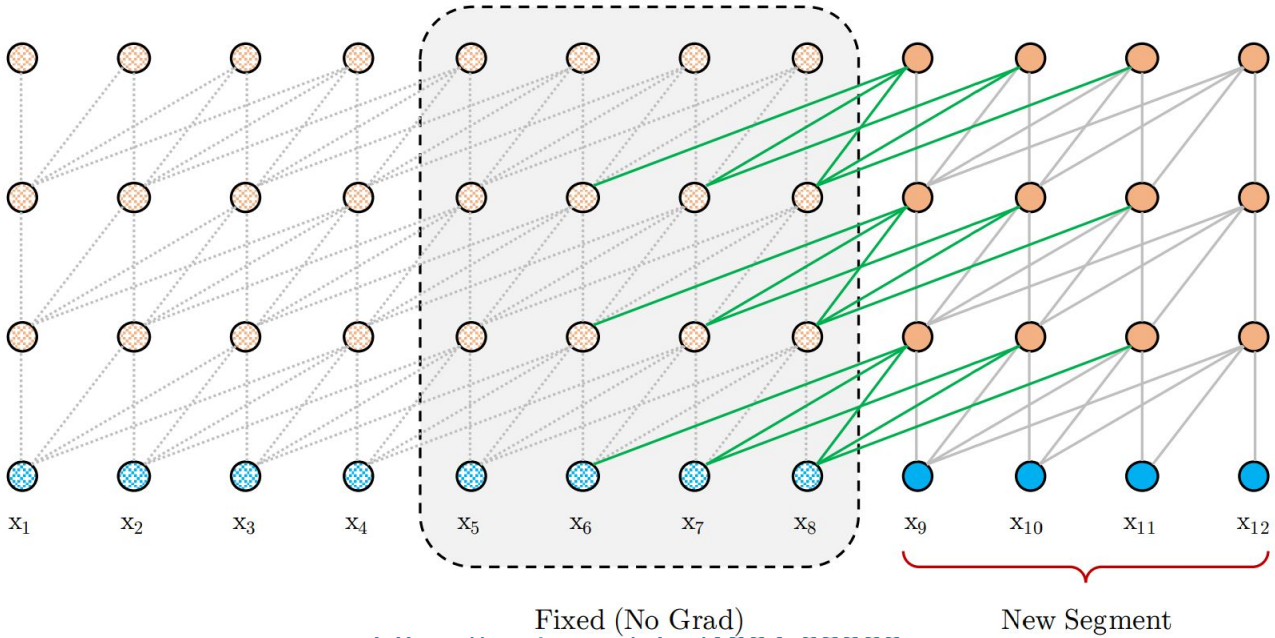
Transformer: The next steps

Другие виды трансформеров

- Image Transformer
- Music Transformer
- Universal Transformer
- Transformer-XL
- Sparse Transformer
- ...

Transformer-XL

Модификация LM на обычном трансформере (декодере). Добавляется возможность посмотреть на репрезентации предыдущего сегмента → можно выучивать более длинные зависимости.



https://arxiv.org/abs/1901.02860

Image Transformer

In this work, we generalize a recently proposed model architecture based on self-attention, the Transformer, to a sequence modeling formulation of image generation with a tractable likelihood. By restricting the self-attention mechanism to attend to local neighborhoods we significantly increase the size of images the model can process in practice, despite maintaining significantly larger receptive fields per layer than typical convolutional neural networks.

Image Transformer, https://arxiv.org/abs/1802.05751

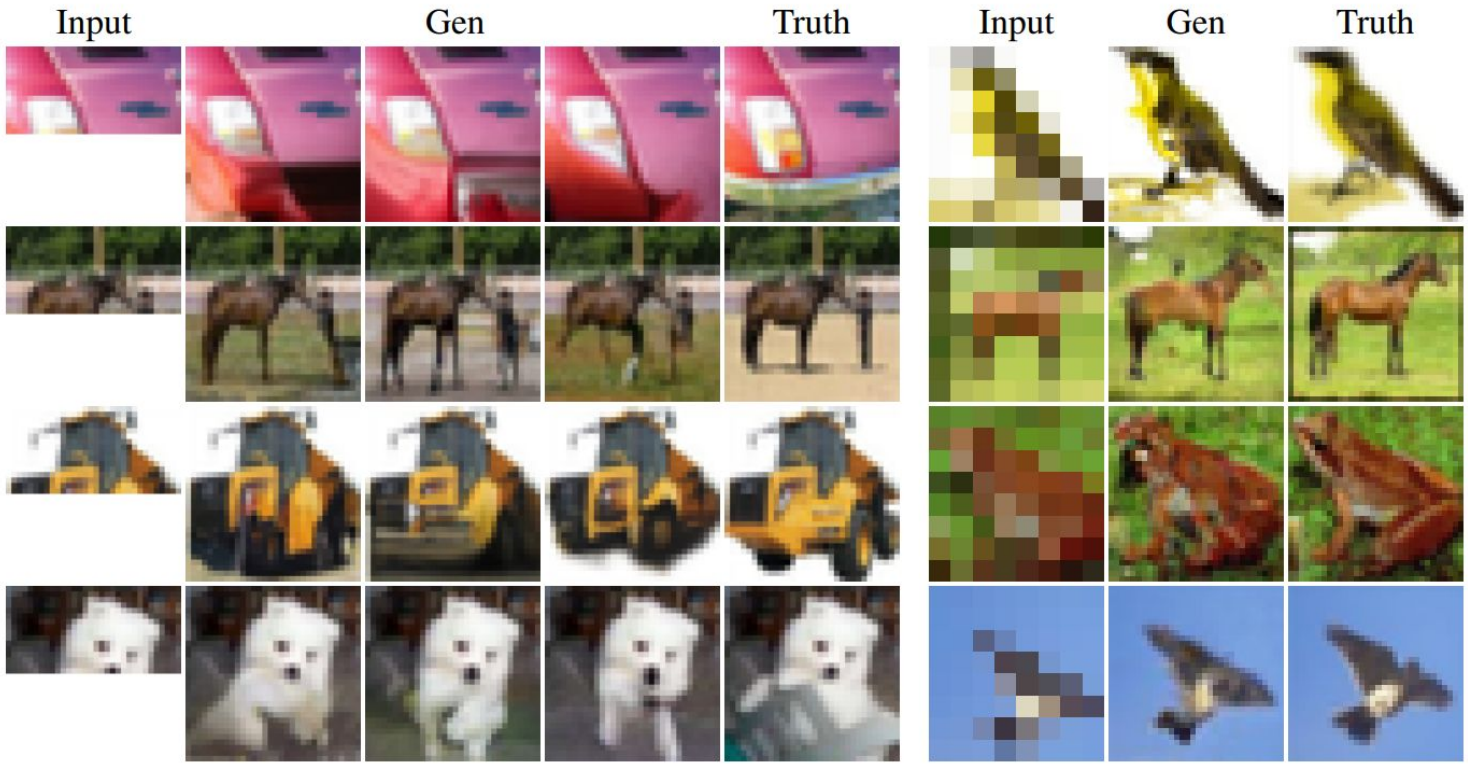


Table 2. On the left are image completions from our best conditional generation model, where we sample the second half. On the right are samples from our four-fold super-resolution model trained on CIFAR-10. Our images look realistic and plausible, show good diversity among the completion samples and observe the outputs carry surprising details for coarse inputs in super-resolution.

Image Transformer, https://arxiv.org/abs/1802.05751

Universal Transformer

In “Universal Transformers” we extend the standard Transformer to be computationally universal (**Turing complete**) using a novel, efficient flavor of parallel-in-time recurrence which yields stronger results across a wider range of tasks.

Crucially, where an RNN processes a sequence symbol-by-symbol (left to right), the Universal Transformer processes all symbols at the same time (like the Transformer), but then refines its interpretation of every symbol in parallel over a variable number of recurrent processing steps using self-attention.

This parallel-in-time recurrence mechanism is both faster than the serial recurrence used in RNNs, and also makes the Universal Transformer more powerful than the standard feedforward Transformer.

Moving Beyond Translation with the Universal Transformer
https://ai.googleblog.com/2018/08/moving-beyond-translation-with.html

Universal Transformer

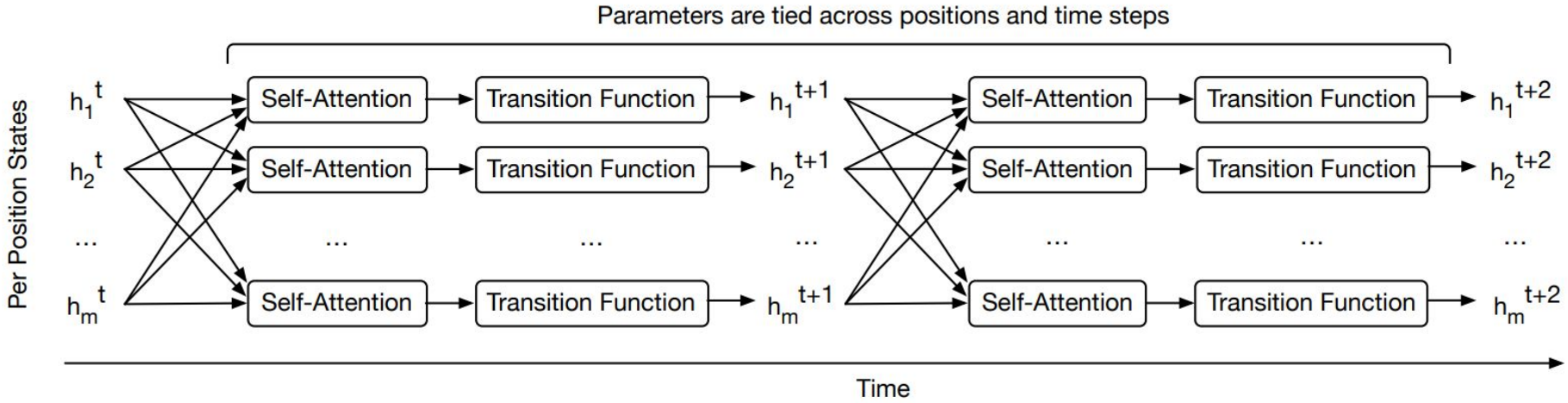


Figure 1: The Universal Transformer repeatedly refines a series of vector representations for each position of the sequence in parallel, by combining information from different positions using self-attention and applying a recurrent transition function. We show this process over two recurrent time-steps. Arrows denote dependencies between operations. Initially, h^0 is initialized with the embedding for each symbol in the sequence. h_i^t represents the representation for input symbol $1 \leq i \leq m$ at recurrent time-step t.

Universal Transformers, https://arxiv.org/abs/1807.03819

Universal Transformer

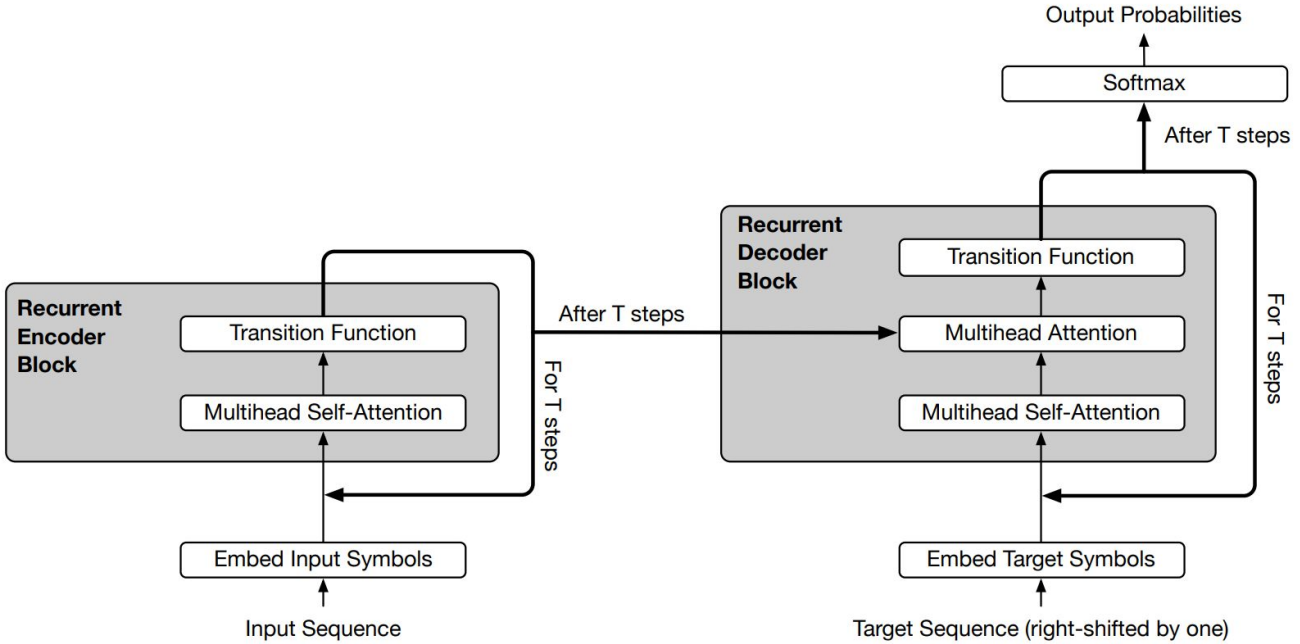


Figure 2: The recurrent blocks of the Universal Transformer encoder and decoder. This diagram omits position and time-step encodings as well as dropout, residual connections and layer normalization. A complete version can be found in the appendix. The Adaptive Universal Transformer dynamically determines the number of steps T for each position using ACT.

Universal Transformers, https://arxiv.org/abs/1807.03819

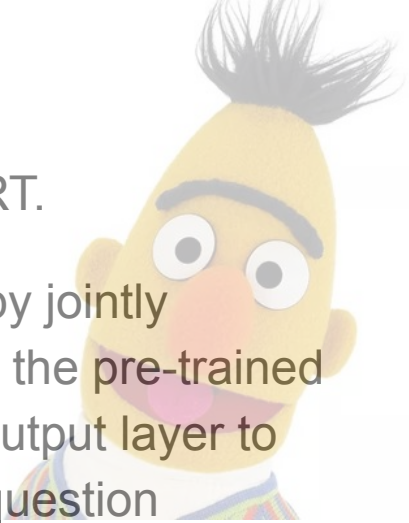
BERT & Co

BERT

Bidirectional Encoder Representations from Transformers, or BERT.

BERT is designed to pre-train deep bidirectional representations by jointly conditioning on both left and right context in all layers. As a result, the pre-trained BERT representations can be fine-tuned with just one additional output layer to create state-of-the-art models for a wide range of tasks, such as question answering and language inference, without substantial task-specific architecture modifications.

BERT uses only the encoder part of the Transformer.



Open Sourcing BERT: State-of-the-Art Pre-training for Natural Language Processing,
https://ai.googleblog.com/2018/11/open-sourcing-bert-state-of-art-pre.html
Best NLP Model Ever? Google BERT Sets New Standards in 11 Language Tasks
https://medium.com/syncedreview/best-nlp-model-ever-google-bert-sets-new-standards-in-11-language-tasks-4a2a189bc155

BERT

Bidirectional Encoder Representations from Transformers, or BERT

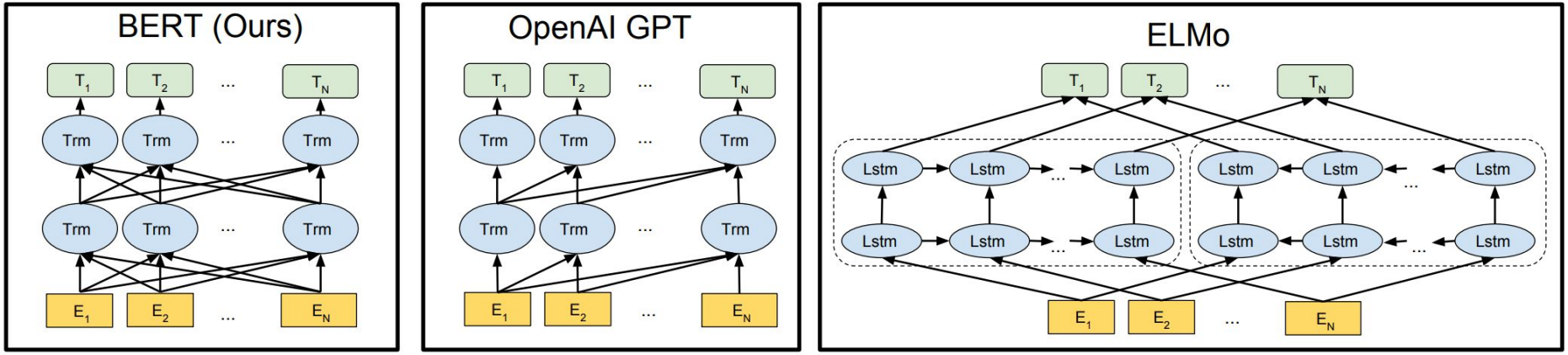


Figure 1: Differences in pre-training model architectures. BERT uses a bidirectional Transformer. OpenAI GPT uses a left-to-right Transformer. ELMo uses the concatenation of independently trained left-to-right and right-to-left LSTM to generate features for downstream tasks. Among three, only BERT representations are jointly conditioned on both left and right context in all layers.

BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding,
https://arxiv.org/abs/1810.04805

BERT

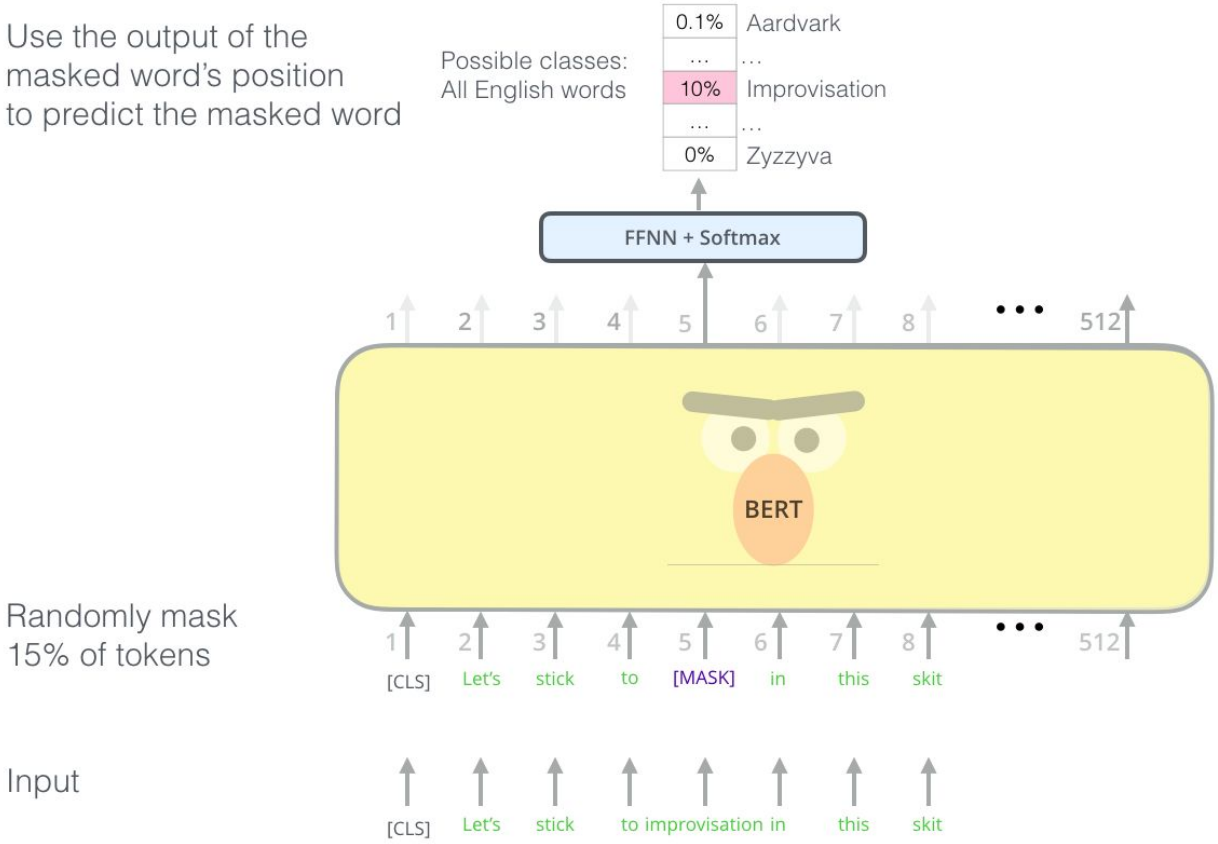
Pre-training tasks:

- **Masked Language Model**: predict random words from within the sequence, not the next word for a sequence of words.
- **Next Sentence Prediction**: give the model two sentences and ask it to predict if the second sentence follows the first in a corpus or not.

Input =

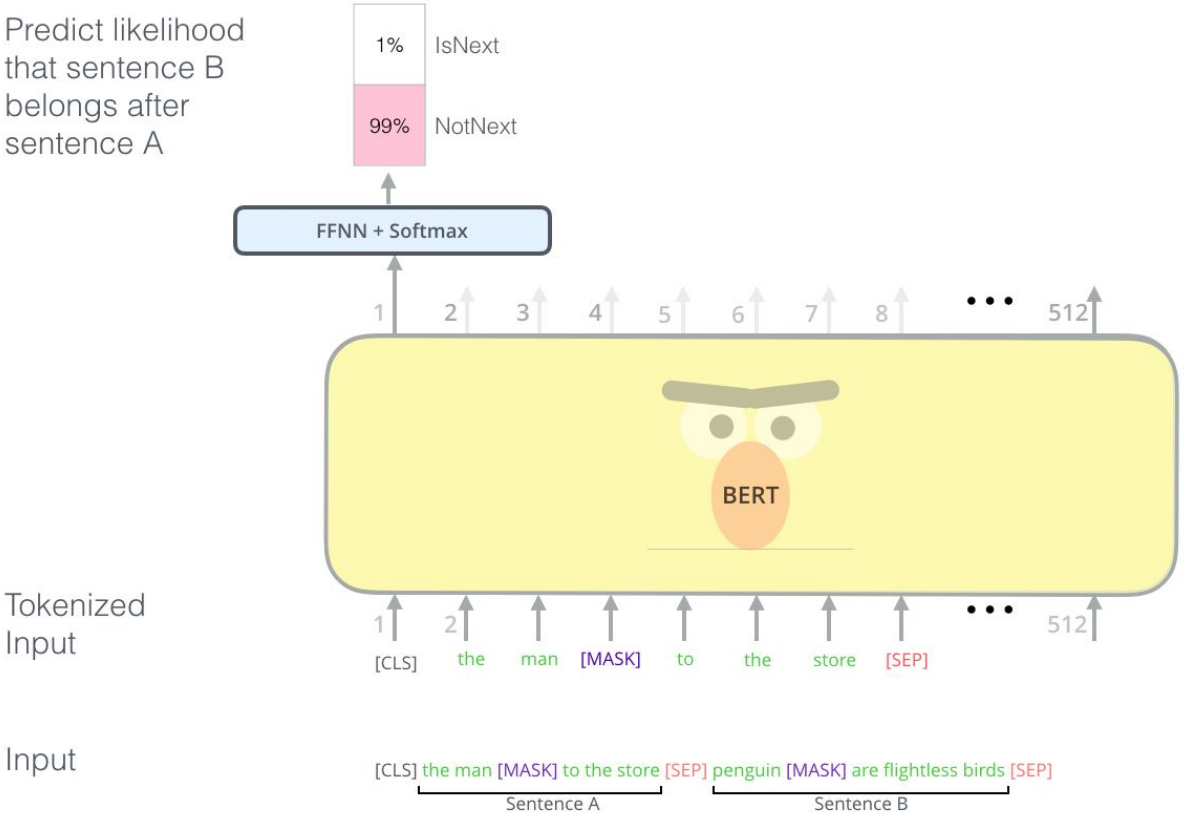
[CLS] the man went to [MASK] store [SEP] he bought a gallon [MASK] milk [SEP]

BERT: masked language model



https://jalammar.github.io/illustrated-bert/

BERT: next sentence prediction



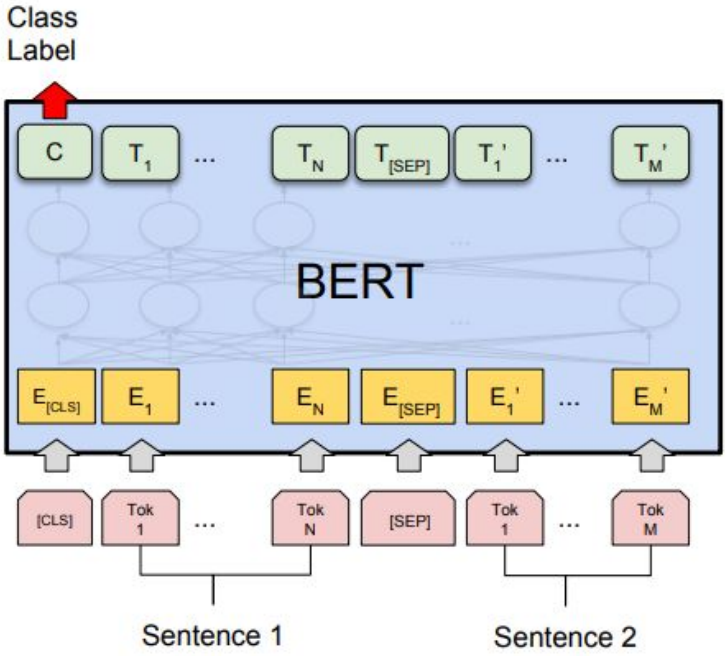
https://jalammar.github.io/illustrated-bert/

BERT

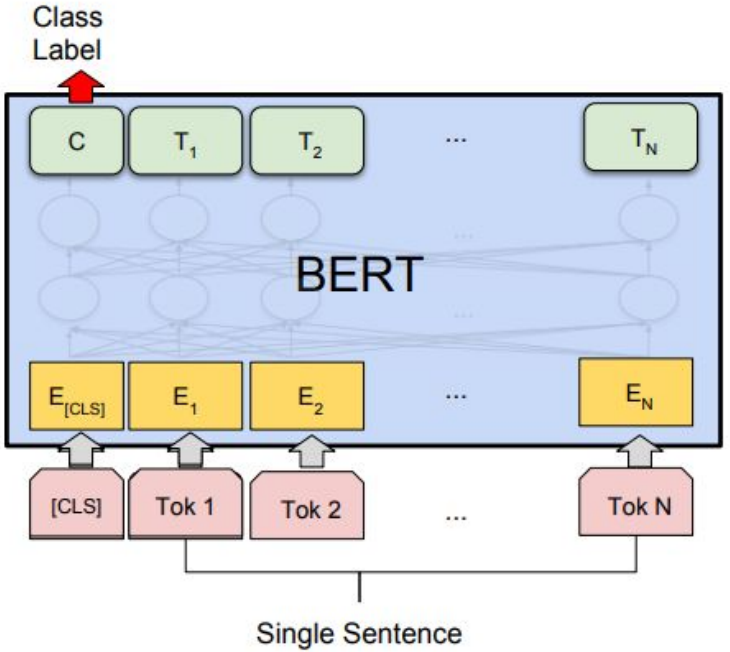
How to use:

- **Fine-tuning approach**: pre-train some model architecture on a LM objective before fine-tuning that same model for a supervised downstream task.
  - Our task specific models are formed by incorporating BERT with one additional output layer, so a minimal number of parameters need to be learned from scratch.
- **Feature-based approach**: learned representations are typically used as features in a downstream model.
  - Not all NLP tasks can be easily be represented by a Transformer encoder architecture, and therefore require a task-specific model architecture to be added.
  - There are major computational benefits to being able to pre-compute an expensive representation of the training data once and then run many experiments with less expensive models on top of this representation

BERT: using fine-tuning approach

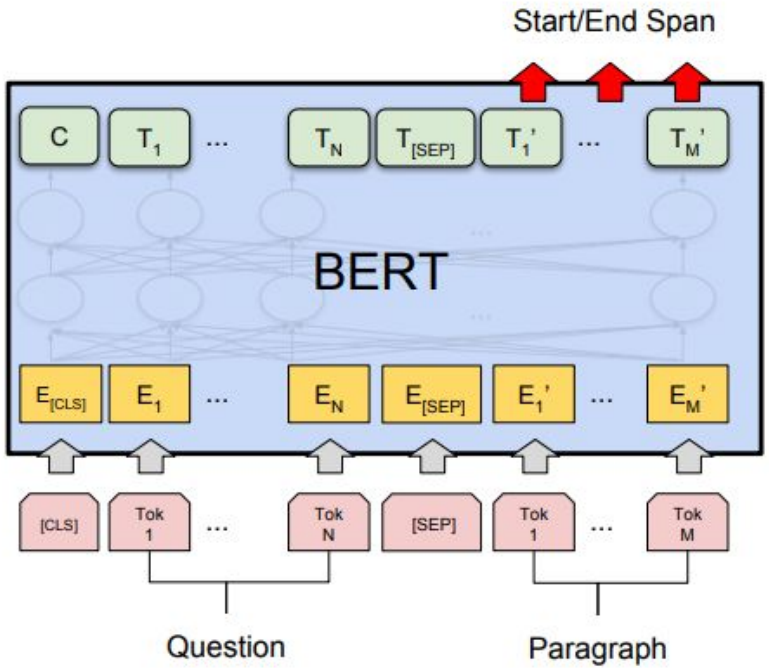


(a) Sentence Pair Classification Tasks: MNLI, QQP, QNLI, STS-B, MRPC, RTE, SWAG

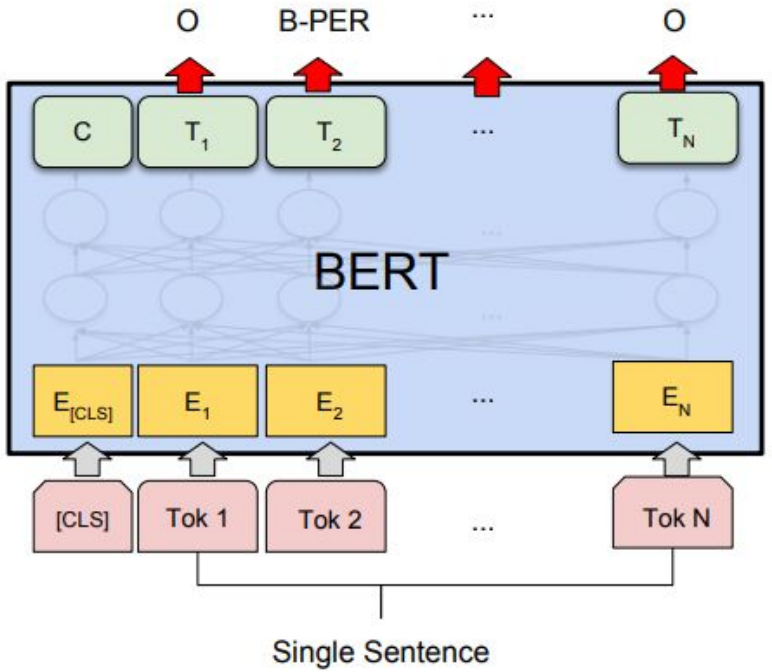


(b) Single Sentence Classification Tasks: SST-2, CoLA

BERT: using fine-tuning approach

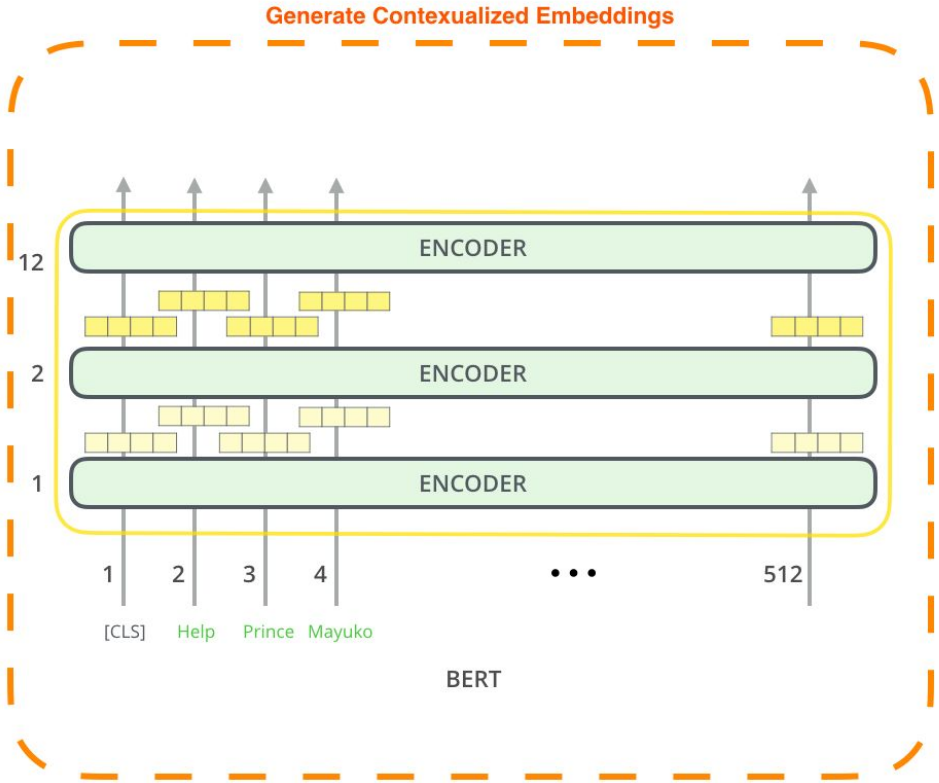


(c) Question Answering Tasks: SQuAD v1.1

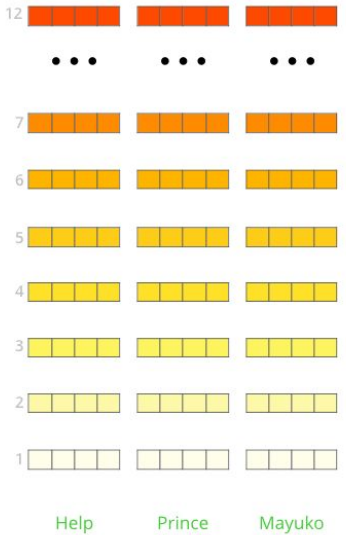


(d) Single Sentence Tagging Tasks: CoNLL-2003 NER

BERT: feature extraction



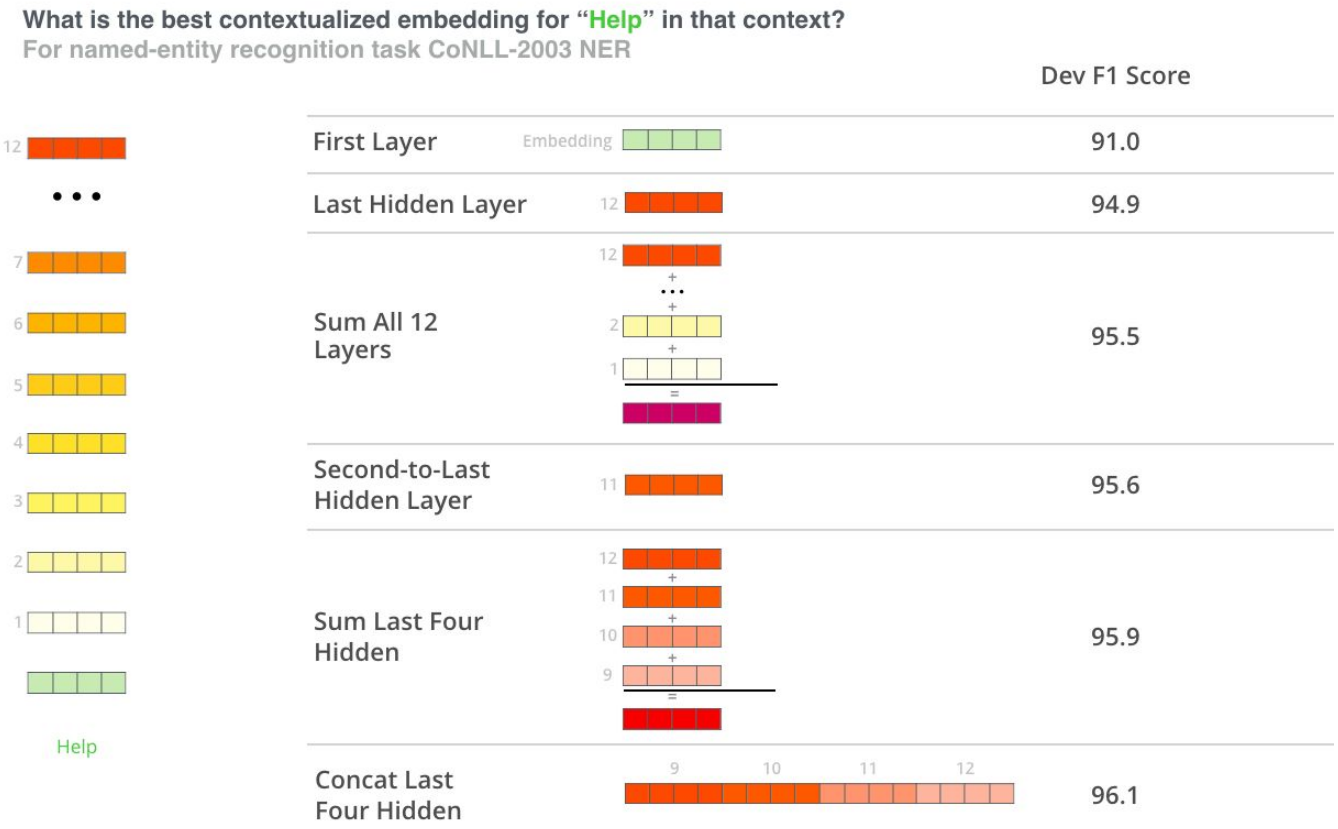
The output of each encoder layer along each token's path can be used as a feature representing that token.



But which one should we use?

https://jalammar.github.io/illustrated-bert/

BERT: feature extraction



https://jalammar.github.io/illustrated-bert/

XLNet: Permutation Language Modeling

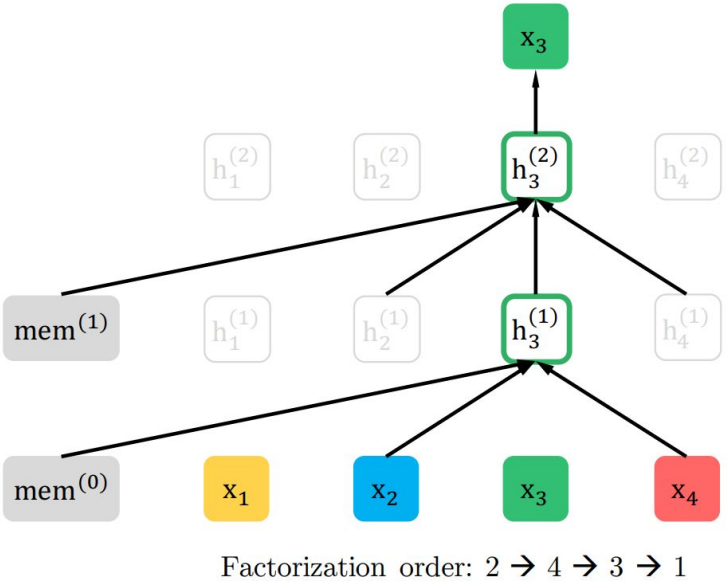
BERT — это, по сути, denoising autoencoder. Он не учитывает взаимозависимость маскируемых слов.

Традиционные LM не видят правый контекст.

Как устранить проблемы этих методов?

Берём в качестве левого контекста различные перестановки слов во всём предложении.

XLNet = Transformer-XL + PLM



XLNet: Generalized Autoregressive Pretraining for Language Understanding
https://arxiv.org/abs/1906.08237

RoBERTa: A Robustly Optimized BERT

BERT был “недотюнен”.

Улучшения:

- Взять больше данных, тренировать дольше
- Next sentence prediction лишний
- Более длинные предложения
- Большие батчи
- Динамическое маскирование

Результат: state-of-the-art на 4 из 9 задачах в GLUE.

https://arxiv.org/abs/1907.11692

GPT-2

Языковая модель на декодере трансформера.

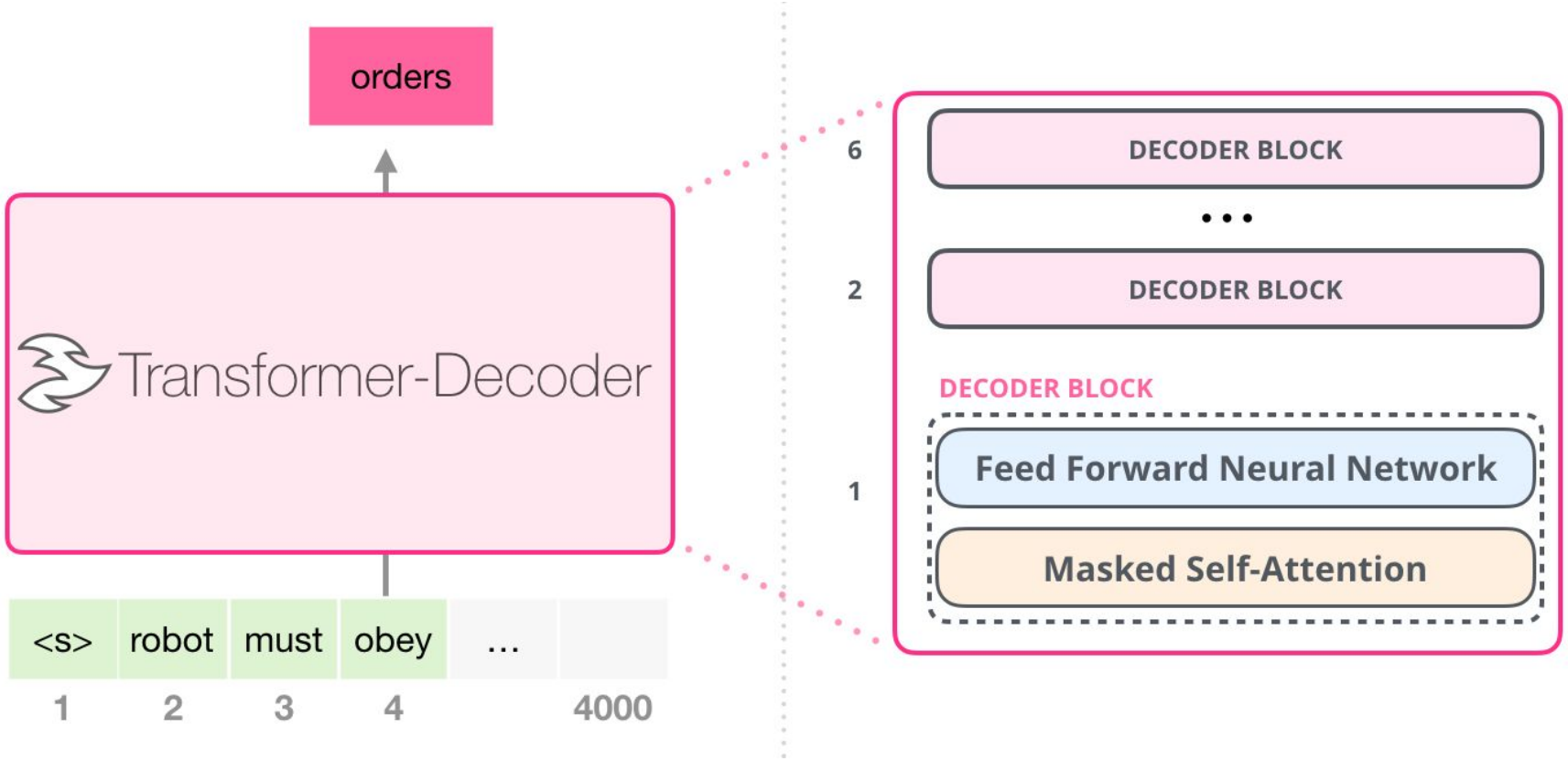
Умеет генерить продолжение текста. Настолько хорошо, что OpenAI отказался её публиковать и устроил мощный PR.

Публикует понемногу, начиная с маленьких моделей.

Разные языковые модели на трансформерах можно попробовать здесь: https://transformer.huggingface.co/

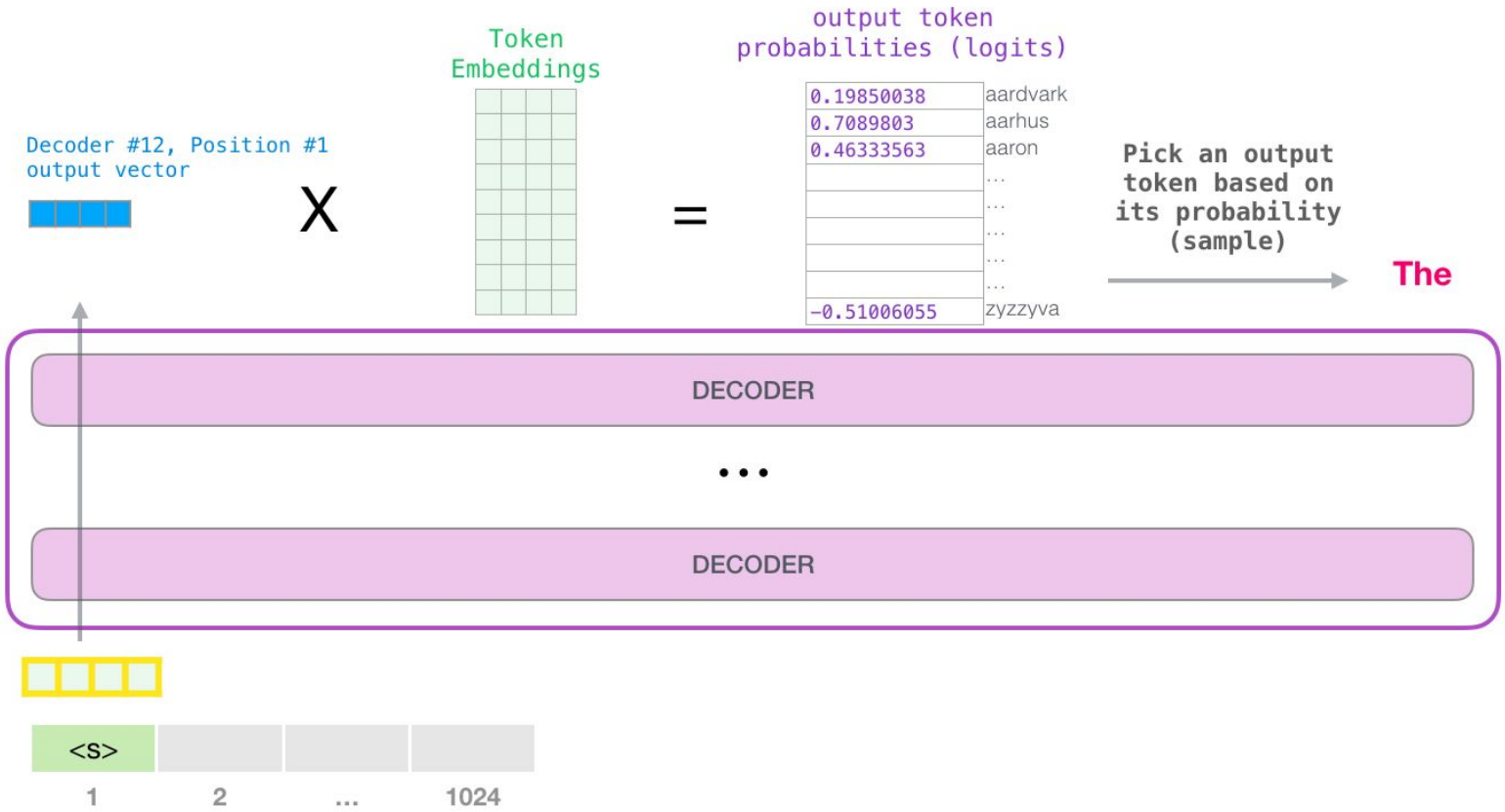
https://openai.com/blog/better-language-models/

GPT-2



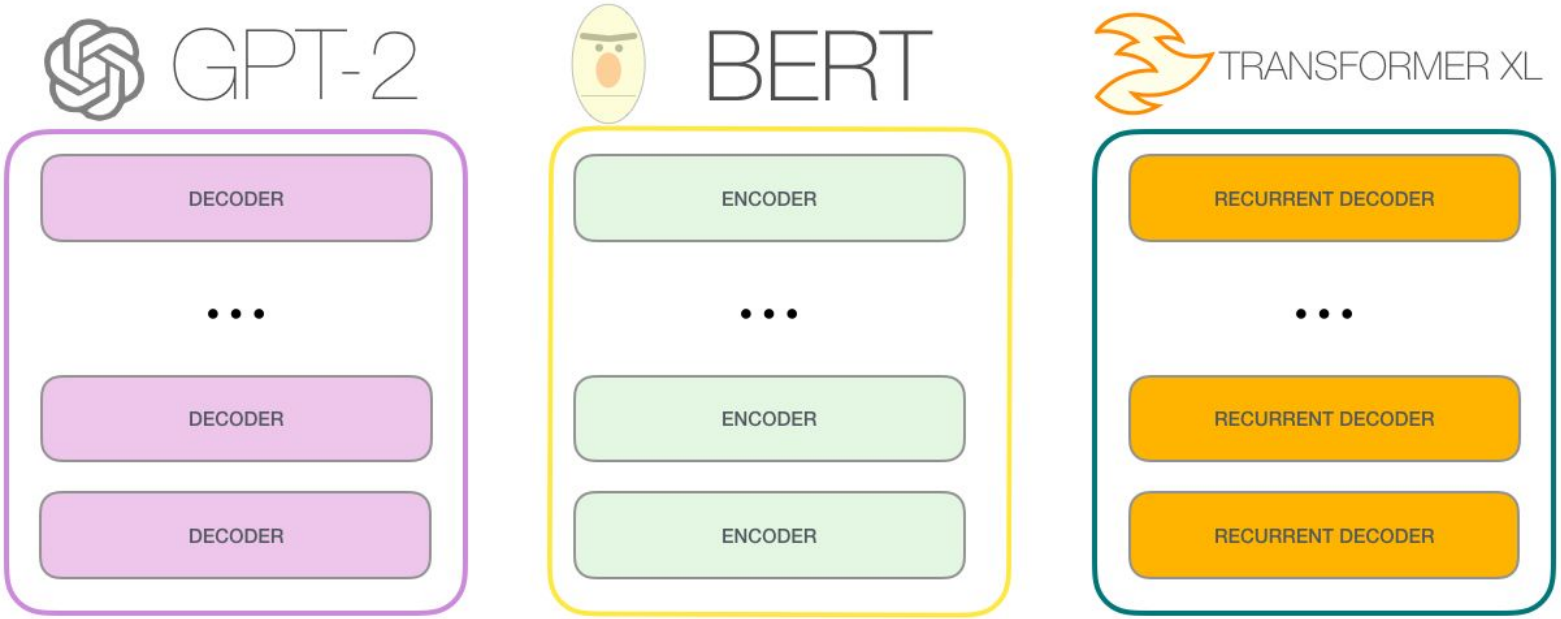
http://jalammar.github.io/illustrated-gpt2/

GPT-2



http://jalammar.github.io/illustrated-gpt2/

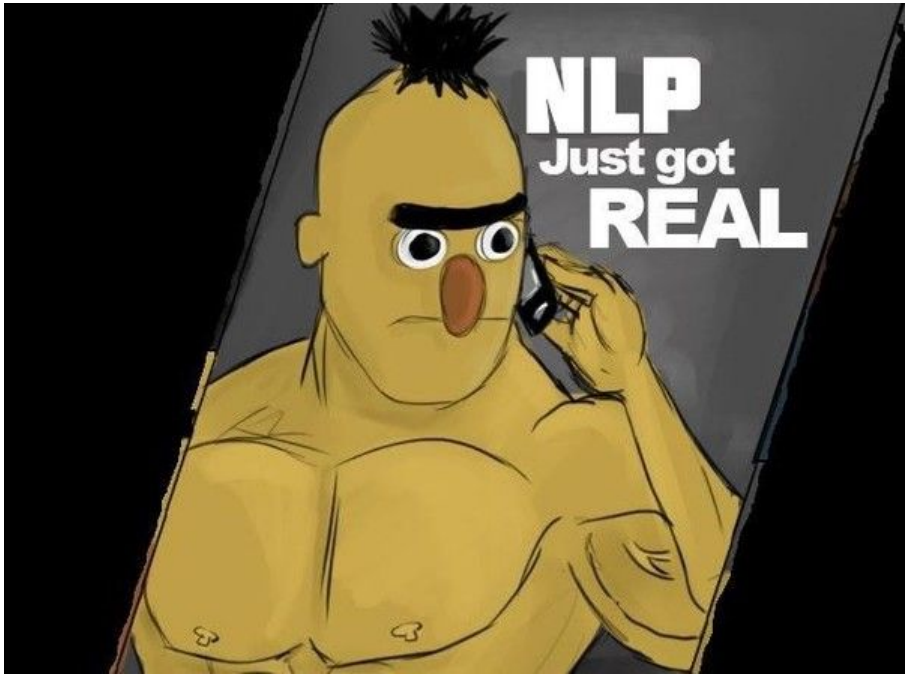
GPT-2 / BERT / Transformer-XL



http://jalammar.github.io/illustrated-gpt2/

Language Model Zoo

- ELMo
- ULMFiT
- GPT
- BERT (BioBERT, ClinicalBERT, …)
- ERNIE
- XLNet
- KERMIT
- ERNIE 2.0
- GPT-2
- ALBERT
- …



Resources

- Open Sourcing BERT: State-of-the-Art Pre-training for Natural Language Processing
  https://ai.googleblog.com/2018/11/open-sourcing-bert-state-of-art-pre.html
- Dissecting BERT Part 1: Understanding the Transformer
  https://medium.com/@mromerocalvo/dissecting-bert-part1-6dcf5360b07f
- Understanding BERT Part 2: BERT Specifics
  https://medium.com/dissecting-bert/dissecting-bert-part2-335ff2ed9c73
- Dissecting BERT Appendix: The Decoder
  https://medium.com/dissecting-bert/dissecting-bert-appendix-the-decoder-3b86f66b0e5f
- The Illustrated BERT, ELMo, and co. (How NLP Cracked Transfer Learning)
  https://jalammar.github.io/illustrated-bert/

Code

- TensorFlow code and pre-trained models for BERT
  https://github.com/google-research/bert
- State-of-the-art Natural Language Processing for TensorFlow 2.0 and PyTorch.
  https://github.com/huggingface/transformers
- DeepPavlov: An open source library for deep learning end-to-end dialog systems and chatbots
  https://github.com/deepmipt/DeepPavlov

Thanks!

grigory.sapunov@ieee.org
gs@inten.to