Transformer architecture

Nicolas.Hernandez@univ-nantes.fr

ATAL - 8 novembre 2024

Nantes Université

Laboratoire des Sciences du Numérique de Nantes

TALN

Introduction

Language modeling: definition

Modeling: predict what event happens next given some description of "context", i.e., the current state of things

In language, an event is a linguistic unit (text, sentence, token, symbol)

Language Models (LMs) estimate the probability of different linguistic units: symbols, tokens, token sequences

Credit: lena-voita

Overview

Overview

- Probabilistic language model
- Contextual word representations
- Transformer

Probabilistic language model

Probabilistic Language model

Allow to estimate if a sequence of words exists in a given language

Various use cases:

- Spelling/grammar correction
  - P(how old are you) > P(how old is you)
- Translation
  - translate_fr_to_en (“quel age as-tu”): P(“how old are you”) > P (“what age have you”)
- Speech recognition:
  - P(I have not a question) > P(I have a knot question)

Probabilistic Language model

Objective:

calculate the probability of the sentence s, made of the words sequence $w_1 \dots w_N$

$$P(s) = P(w_1 \dots w_2 \dots w_N)$$

Use the chain rule to rewrite this joint probability in product of conditional probabilities:

$$P(s) = P(w_1) \times P(w_2|w_1) \times \dots \times P(w_N|w_1 \dots w_{N-1}) = \prod_{i=1}^{N} P(w_i|w_1 \dots w_{i-1})$$

with $h_i = w_1 \dots w_{i-1}$ *historical* of word w_i

Parameters $P(w_i|h_i)$ can be estimated on a corpus

Problem: there is no corpus which allows to observe such historical sub sequences

N-gram language model

In n-gram models, historical h_i of word w_i can be approximated by the n−1 preceding words (based on **Markov hypothesis**).

L'historique devient $h_i = w_{i-n+1} \ldots w_{i-1}$: deux historiques se terminant par les mêmes n−1 mots appartiennent alors à la même classe *d'équivalence*.

Une séquence de n mots est appelée un **n-gramme** et n correspond à **l'ordre** du modèle n-gramme.

Probability of a sentence can be reformulated in:

$$P(s) = \prod_{i=1}^{N} P(w_i|h_i) = \prod_{i=1}^{N} P(w_i|w_{i-n+1}^{i-1})$$

Conditional probability estimation

Probability $P(w_i|w_{i-n+1})$ can be calculated thanks to the **Maximum Likelihood estimation**, i.e. by calculating the relative frequencies of n-grams in a training corpus:

$$P(w_i|w_{i-n+1}^{i-1}) = \frac{c(w_{i-n+1}^{i-1} w_i)}{\sum_{w_j \in V} c(w_{i-n+1}^{i-1} w_j)}$$

- n : n-gram model order ;
- V : vocabulary;
- c(.) : counter of ngram occurrences in the training corpus.

Example

Corpus

<s> Antoine écoute Thom </s>
<s> Denis écoute une autre chanson </s>
<s> Elle écoute une chanson de Lionel </s>

Common n values:

- Unigram model (n = 1) :

  P(<s> il fait beau </s>) = P(<s>) × P(il) × P(fait) × P(beau) × P(</s>)

- Bigram model (n = 2) :

  P(<s> il fait beau </s>) = P(<s>) × P(il|<s>) × P(fait|il) × P(beau|fait) × P(</s>|beau)

- Trigram (n = 3) :

  P(<s> il fait beau </s>) = P(<s>) × P(il|<s>) × P(fait|<s> il) × P(beau|il fait) × P(</s>|fait beau)

Estimation de P(Antoine|<s>) selon le modèle bigramme :

$$P(Antoine|<s>) = \frac{c(<s>\ Antoine)}{\sum_{w_j \in V} c(<s> w_j)} = \frac{1}{3}$$

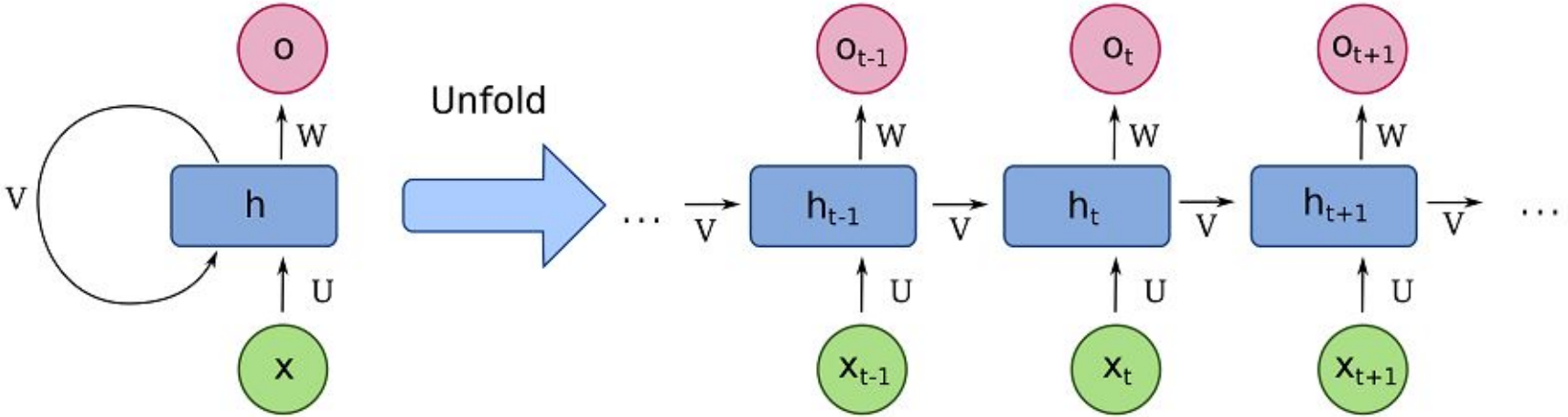
Contextual word representations

ELMO, BERT, etc.

Recurrent Neural Network (RNN)

While they show good results for shorter sequences (~100 words/steps), in longer ones the distance between the relevant words can be too long for results to be acceptable.

"Every week, I repetitively wash my clothes, and I do grocery shopping every day. Starting next week, I will … ?"

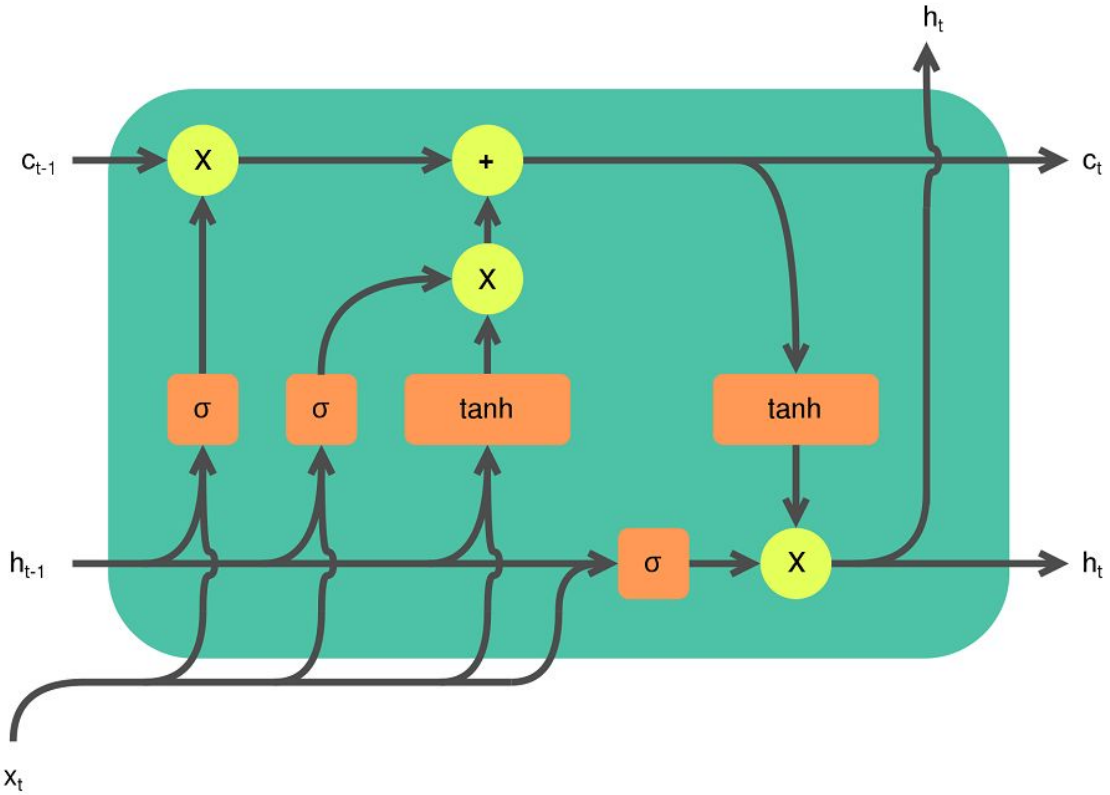


Long Short-Term Memory (LSTM)

(Hochreiter and Schmidhuber, 1997)

Several gates are available for partially processing the previous state as well as new inputs.

But the sequence is still parsed iteratively one word at a time.



Transformer: Attention is all you need…

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.

Context: Machine Translation task

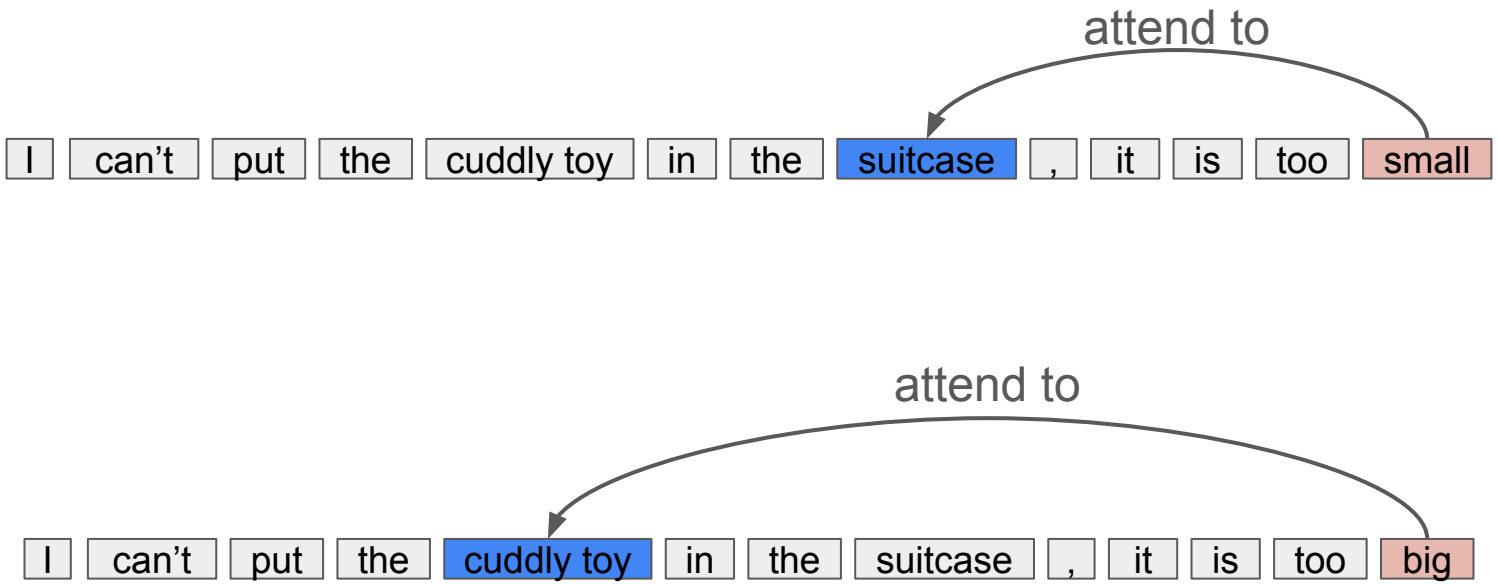
Architecture: **encoder-decoder**

Innovation: based solely on **(self-)attention mechanisms** ; no recurrence no convolution

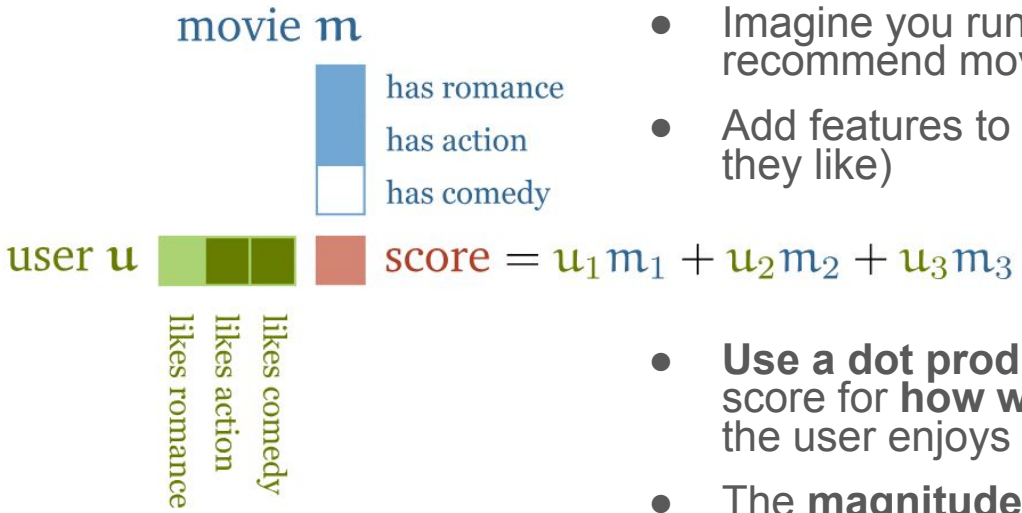
Some attention mechanisms already proposed e.g. (Bahdanau et al., 2014)

Attend to (pay attention to): basic intuition (1)

A degree of influence from one word to another



Attend to (pay attention to): basic intuition (2)



- Aims at **measuring the influence between two elements**
- Imagine you run a movie rental business and want to recommend movies to users that they **are likely to** enjoy
- Add features to movies (what they have) and users (what they like)
- **Use a dot product** between the two feature vectors to get a score for **how well** the attributes of the movie match what the user enjoys
- The **magnitudes** of the features indicate how much the feature should contribute to the total score

Credit: peterbloem.nl

Self-attention: simple formalization

- A **sequence-to-sequence** operation
  - a sequence of `L(engths)` vectors $x_{i \in \{1..L\}} \in \mathbb{R}^d$, `d` the dimension of the model, goes in and a sequence of `L` vectors $y_{i \in \{1..L\}} \in \mathbb{R}^d$ of same dimension `d` goes out (`d=512` originally)

- Each output vector y_i, a **weighted average over all the input vectors** x_i

$$\mathbf{y}_i = \sum_j w_{ij} \mathbf{x}_j$$

- The weight w'_{ij}, indicates **which vectors $\mathbf{x}_j$ influence $\mathbf{x}_i$** by the similarity of the vectors,
  - a **dot product** of the current x_i with all the x_j of the sequence

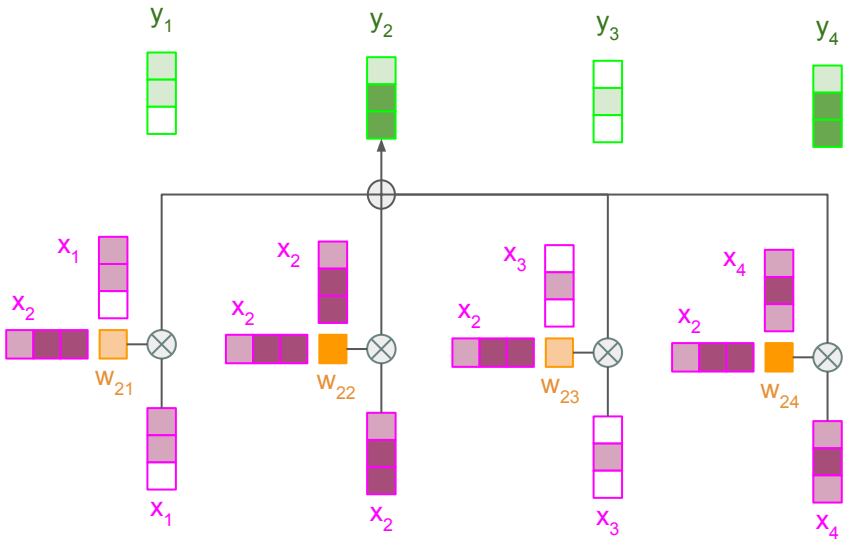
$$w'_{ij} = \mathbf{x}_i{}^{\mathsf{T}} \mathbf{x}_j$$

- **Normalisation** of the weights and **construction of a probability distribution** over all the x_j by applying a **softmax**
  - Map the dot product range values from (-∞,+∞) to [0,1]) and make sure that the sum of all the w'_{ij} elements is equal to 1

$$w_{ij} = \frac{\exp w'_{ij}}{\sum_j \exp w'_{ij}}$$

Simple formalization: self-attention

- x_i a sequence of vectors e.g. words
- y_i output vector at the same position as the input vector x_i
- Here focus on calculation of y_2, softmax not represented
- Each output vector, an entirely new series of dot products, and a different weighted sum
- Attention operation, computed simultaneously, packed together into a matrix



Adapted from: peterbloem.nl

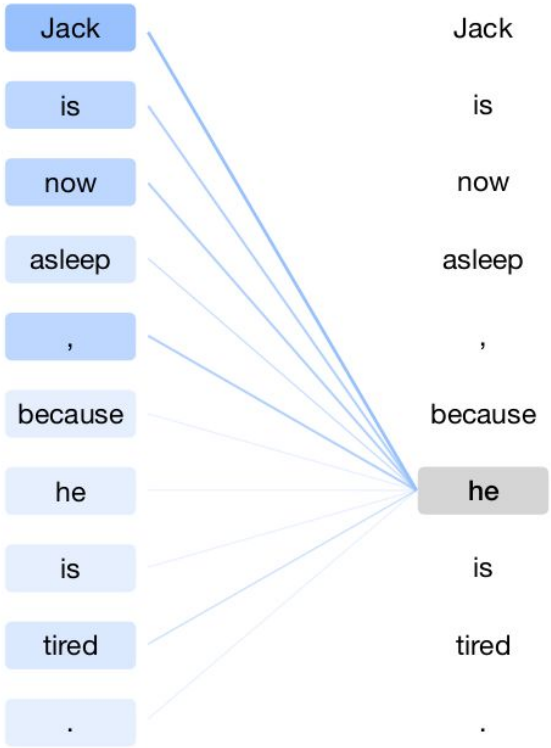


Figure 6: An illustration of the self-attention mechanism of Transformer. The figure shows the self-attention results when encoding the word "he", where the darker the color of the square is, the larger the corresponding attention score is.

Attention operation

Linear transformation

No parameters (yet)

Ignores the sequential nature of the input (it is **permutation equivariant**)

The only one in the architecture that propagates information between vectors through interactions between them

A family of attention mechanisms

Below is a summary table of several popular attention mechanisms and corresponding alignment score functions:

Name	Alignment score function	Citation
Content-base attention	$\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \text{cosine}[\boldsymbol{s}_t, \boldsymbol{h}_i]$	Graves2014
Additive(*)	$\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \mathbf{v}_a^\top \tanh(\mathbf{W}_a[\boldsymbol{s}_{t-1}; \boldsymbol{h}_i])$	Bahdanau2015
Location-Base	$\alpha_{t,i} = \text{softmax}(\mathbf{W}_a \boldsymbol{s}_t)$ Note: This simplifies the softmax alignment to only depend on the target position.	Luong2015
General	$\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \boldsymbol{s}_t^\top \mathbf{W}_a \boldsymbol{h}_i$ where $\mathbf{W}_a$ is a trainable weight matrix in the attention layer.	Luong2015
Dot-Product	$\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \boldsymbol{s}_t^\top \boldsymbol{h}_i$	Luong2015
Scaled Dot-Product(^)	$\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \frac{\boldsymbol{s}_t^\top \boldsymbol{h}_i}{\sqrt{n}}$ Note: very similar to the dot-product attention except for a scaling factor; where n is the dimension of the source hidden state.	Vaswani2017

(*) Referred to as "concat" in Luong, et al., 2015 and as "additive attention" in Vaswani, et al., 2017.
(^) It adds a scaling factor $1/\sqrt{n}$, motivated by the concern when the input is large, the softmax function may have an extremely small gradient, hard for efficient learning.

Credit: lilianweng

A family of attention mechanisms

Here are a summary of broader categories of attention mechanisms:

Name	Definition	Citation
Self-Attention(&)	Relating different positions of the same input sequence. Theoretically the self-attention can adopt any score functions above, but just replace the target sequence with the same input sequence.	Cheng2016
Global/Soft	Attending to the entire input state space.	Xu2015
Local/Hard	Attending to the part of input state space; i.e. a patch of the input image.	Xu2015; Luong2015

(&) Also, referred to as "intra-attention" in Cheng et al., 2016 and some other papers.

Credit: lilianweng

Additional tricks

- queries, keys and values
- scaling
- multi-head attention

Additional tricks: 1) Queries, keys and values

Three different parts of the self attention and every input vector x_i used in three different roles:

- query
  - to compare to every other vector to establish the weights for its own output y_i
- key
  - to compare to every other vector to establish the weights for the output of the j-th vector y_j
- value
  - as part of the weighted sum to compute each output vector once the weights have been established

Names comes from an analogy for the data structure of a key-value store

Additional tricks: 1) Queries, keys and values

Three additional weight matrices :

- $\mathbf{W}_Q \in \mathbb{R}^{d \times d_k}$, the query weight matrix. $d_q = d_k$. Without multiple heads $d_k = d$.
- $\mathbf{W}_K \in \mathbb{R}^{d \times d_k}$, the key weight matrix
- $\mathbf{W}_V \in \mathbb{R}^{d \times d_v}$, the value weight matrix. Often $d_v = d_k$

to compute three linear transformations of each x_i:

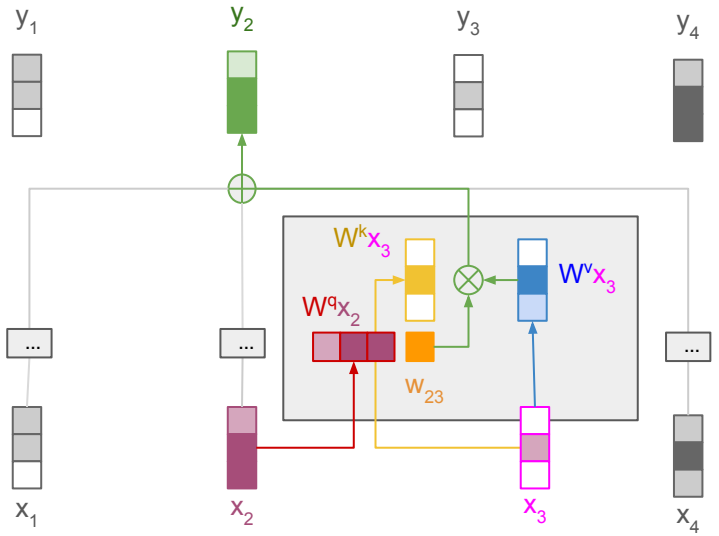
$$q_i = \mathbf{W}_Q x_i \qquad k_i = \mathbf{W}_K x_i \qquad v_i = \mathbf{W}_V x_i$$

$$w'_{ij} = q_i^T k_j$$

$$w_{ij} = softmax(w'_{ij})$$

$$y_i = sum_j (w_{ij} v_j)$$

Still (x_2, y_2) in focus:



Derived from: peterbloem.nl

Scaling

Additional tricks: 2) Scaling the dot product

High value of the embedding dimension `d` and/or large value of dot product pushes the softmax function to have several values near to zero (which can lead to the problem of vanishing gradient during training)

Solution: scale the dot product by a factor related to the size of the embeddings, via diving by the square root of the dimensionality `d` of the query and key vectors

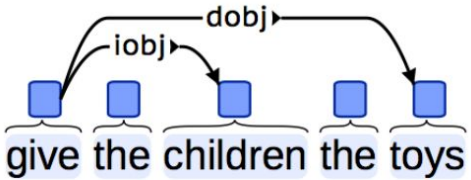
$$w'_{ij} = \frac{q_i^T k_j}{\sqrt{d}}$$

The sqrt function comes from the fact that the Euclidian length of a vector of `d` real values `c` is `sqrt(d*c)`

See: peterbloem.nl

Multi-head attention

Additional tricks: 3) Multi-head attention



Many phenomena to attend in the sequence

E.g. A word can mean different things to different neighbours : *children* expresses the recipient and *toys* what is given

$x_{children}$ and x_{toys} can influence y_{give} by different amounts, depending on their dot product with x_{give} but they cannot influence it in different ways !

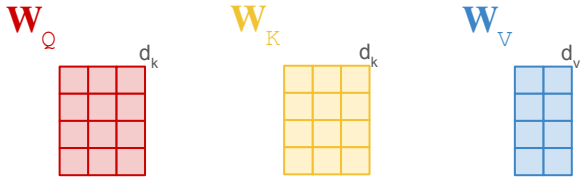
Need for several, h, attention heads of d/h dimensions each (8 of 64 originally) to capture various aspects; each of them with their own matrices $\mathbf{W}_{Qh}$, $\mathbf{W}_{Kh}$, $\mathbf{W}_{Vh}$ to project in subspaces

Applied in parallel

Result concatenated and linearly transformed to Y through matrice $\mathbf{W}_{O}$

Single head matrix calculation

Weights



$$W_Q \in \mathbb{R}^{d \times d_k}; W_K \in \mathbb{R}^{d \times d_k}; W_V \in \mathbb{R}^{d \times d_v}$$

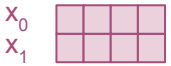
d_k, d_k and d_v equal to d when single attention head

Input

thinking machine

Embeddings

X



Queries

$Q=XW_Q$

Keys

$K=XW_K$

Values

$V=XW_V$

$$X \in \mathbb{R}^{L \times d}$$

$$Q = XW_Q; K = XW_K; V = XW_V$$

$$Q \in \mathbb{R}^{L \times d_k}; K \in \mathbb{R}^{L \times d_k}; V \in \mathbb{R}^{L \times d_v}$$

Attention

$A=softmax(QK^T/\sqrt{d_k})$

$$A \in \mathbb{R}^{L \times L}$$

Information captured or propagated also called attention (Q, K, V)

$Y=AV$

$$Y \in \mathbb{R}^{L \times d_v}$$

Illustration inspired from: jalammar

Multi head matrix calculation

Weights

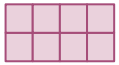
Input

Embeddings

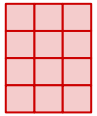
X

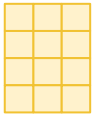
thinking machine

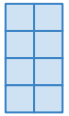
x_0
x_1



Attention head #0

W_{Q_0} 

W_{K_0} 

W_{V_0} 

Queries

$Q_0 = XW_{Q\_0}$

q_{00}
q_{01}

Keys

$K_0 = XW_{K\_0}$

k_{00}
k_{01}

Values

$V_0 = XW_{V\_0}$

v_{00}
v_{01}

Attention

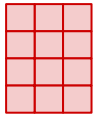
$A_0 = softmax(Q_0 K_0^T / \sqrt{d_k})$

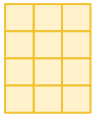
$Y_0 = A_0 V_0$

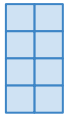
y_{00}
y_{01}

Information captured or propagated also called attention (Q, K, V)

Attention head #1

W_{Q_1} 

W_{K_1} 

W_{V_1} 

Queries

$Q_1 = XW_{Q\_1}$

q_{10}
q_{11}

Keys

$K_1 = XW_{K\_1}$

k_{10}
k_{11}

Values

$V_1 = XW_{V\_1}$

v_{10}
v_{11}

$A_1 = softmax(Q_1 K_1^T / \sqrt{d_k})$

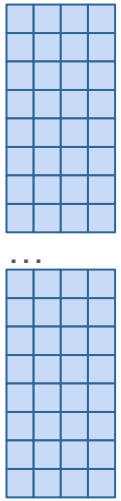
$Y_1 = A_1 V_1$

y_{10}
y_{11}

Illustration inspired from: jalammar

Concatenation then projection to d dimensions

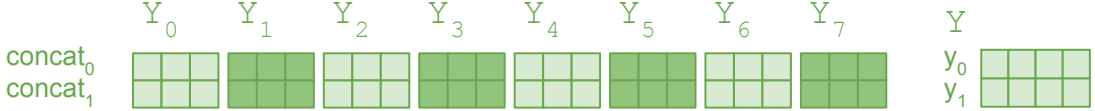
W_O, Output weight matrix



$$W_O \in \mathbb{R}^{h*d_v \times d}$$

$$Y = \text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_{i1}, \ldots \text{head}_h)W_O$$

where $\text{head}_i = \text{Attention}\ (QW_{Qi}, KW_{Ki}, VW_{Vi})$

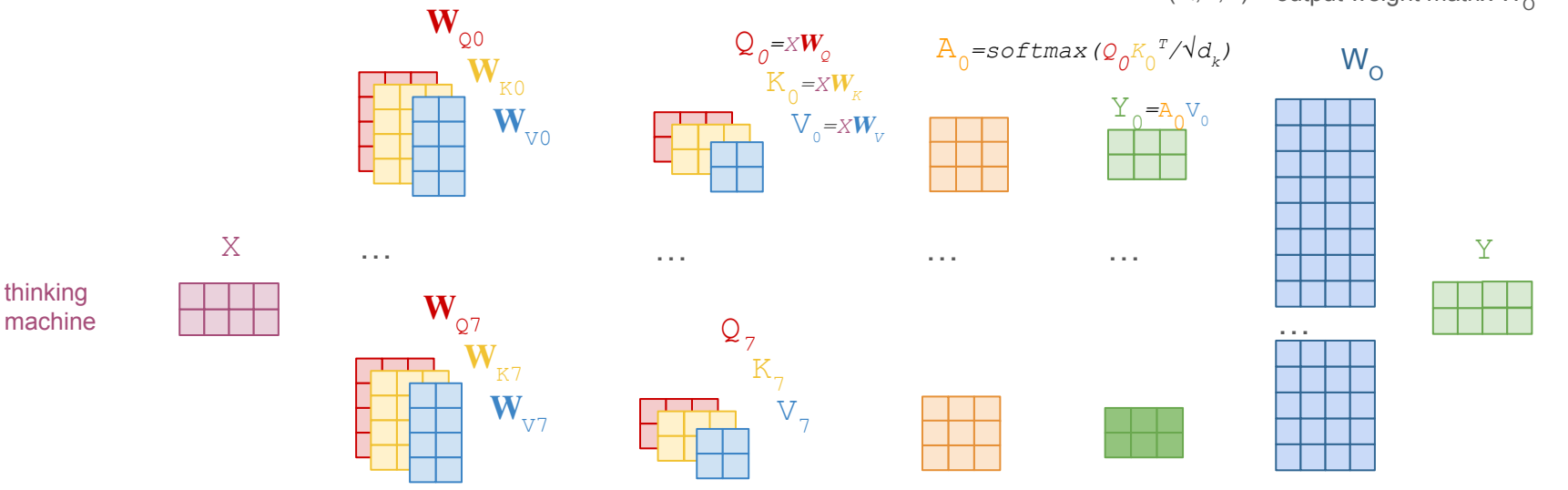


$$Y \in \mathbb{R}^{L \times d_v}$$

Self attention: all in one

Illustration inspired from: jalammar

1. tokenize the input sequence
2. embed each token in d_{model}
3. split into 8 heads (project into) by multiplying X with the weight matrices of each head
4. calculate the attention
5. calculate the information propagated also called Attn(Q,K,V)
6. concatenate and project in d_{model} by multiplying with the output weight matrix W_O



$X \in \mathbb{R}^{L \times d}$

$W_{Q_i}, W_{K_i} \in \mathbb{R}^{d \times d_k/h}$

$W_{V_i} \in \mathbb{R}^{d \times d_v/h}$

$Q_i, K_i \in \mathbb{R}^{L \times d_k/h}$

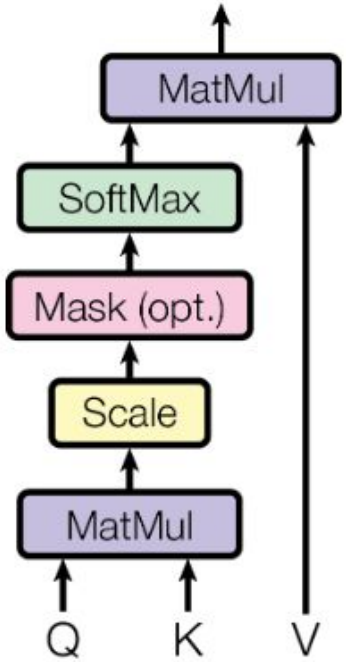
$V_i \in \mathbb{R}^{L \times d_v/h}$

$A_i \in \mathbb{R}^{L \times L}$

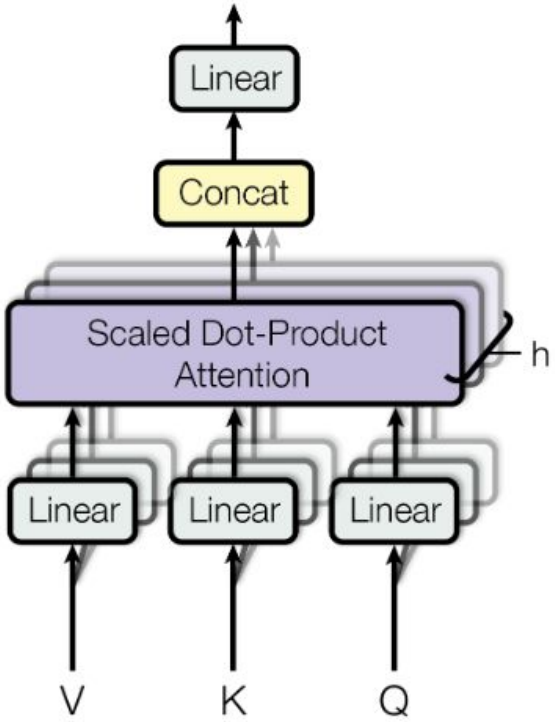
$W_O \in \mathbb{R}^{h*d_v \times d}$

$Y \in \mathbb{R}^{L \times d_v}$

Scaled Dot-Product Attention



Multi-Head Attention



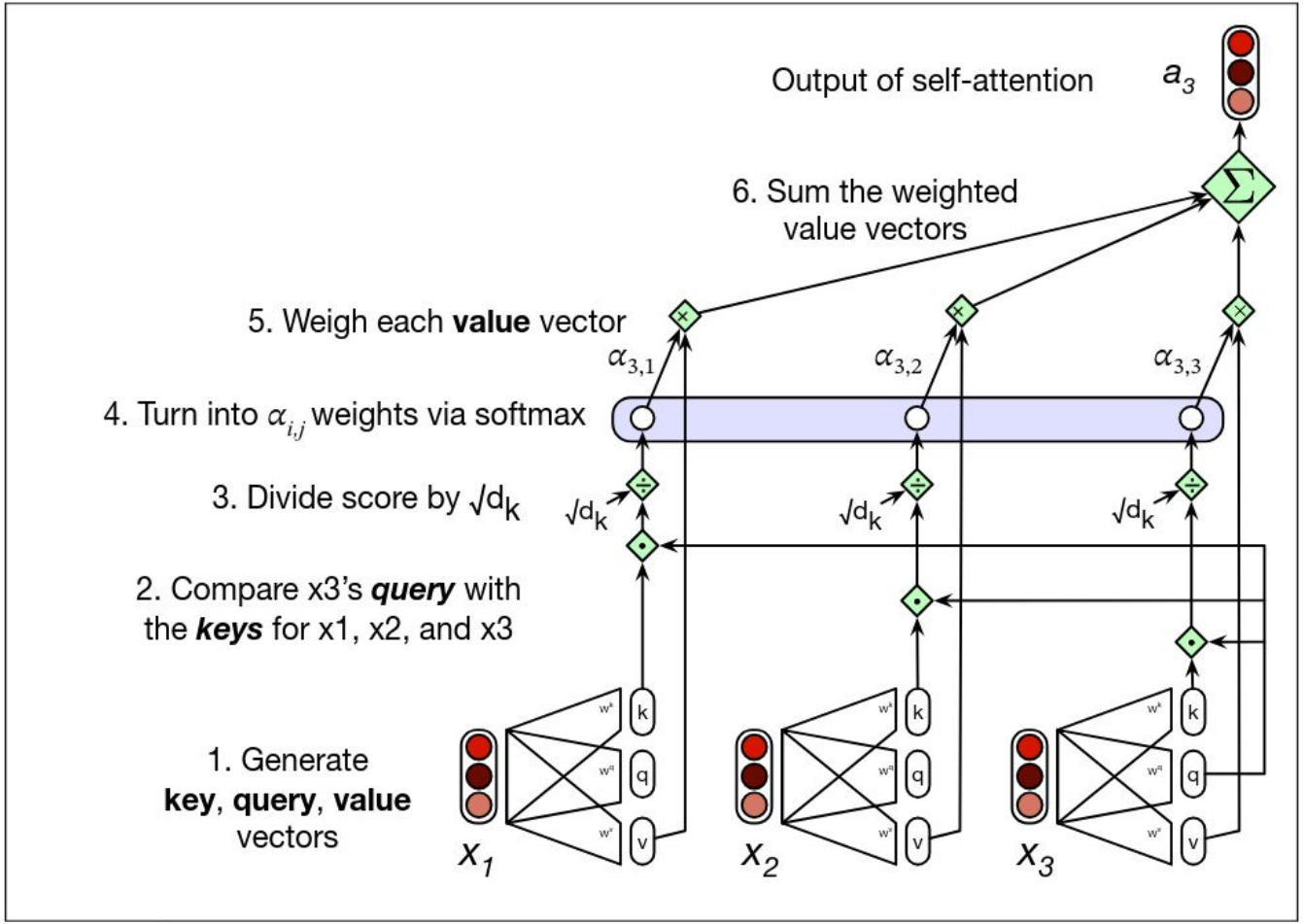
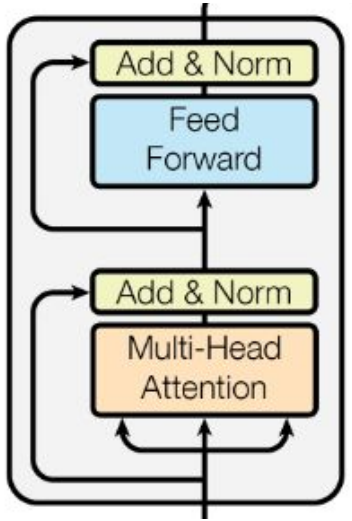


Figure 9.4 Calculating the value of $\mathbf{a}_3$, the third element of a sequence using causal (left-to-right) self-attention.

The transformer block

The Transformer block: a seq2seq block



Applies in sequence (bottom-up):

4. another normalization layer with addition of residual connections
3. a feed forward layer
2. a normalization layer with addition of residual connections
1. a multiple head of self attention layer

Normalization and residual connections (Add: $x \mapsto f(x)+x$): standard **tricks to stabilize the training and convergence** of deep neural networks

FFN is a Multi-Layer Perceptron of

- two linear transformations which expand-and-contract
- and one hidden layer between with a ReLU activation max (0, x) which **introduces non-linearity** and prevents vanishing gradients
- “position-wise” because the same transformation across different positions

Can be seen as 2 sub blocks: multi-head attention block and feed forward block

Originally: FFN dim(input) = dim(output) = 512 and dim(inner layer) = 2048 ; 6 blocks

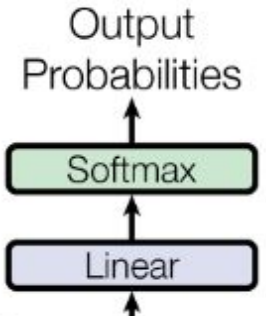
Output

Output

A common way to build a classifier:

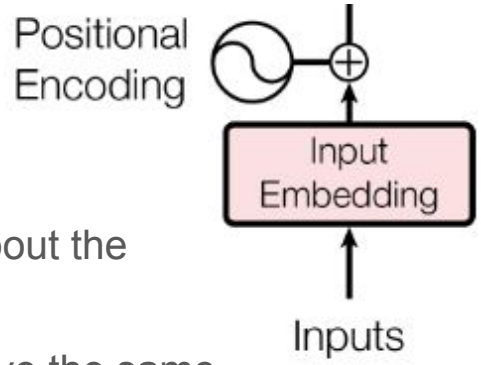
- apply global average pooling
- map to a softmax class vector (one element per class and probability score)

E.g. softmax over a vocabulary



Positional encoding

Positional encoding



Since attention mechanism is permutation equivariant, no information about the order of the sequence

Solution: add a position embedding to each input embedding (should have the same number of dimension d as the tokens to be summed)

Can be fixed, learned or resulting of a mapping function from the position

The original paper defines a mapping function which uses sine and cos functions and map a position `pos` among `L` (sequence size) and a rank `i` among `d` (model dimensions) to a real value: $PE(pos, i) \in \mathbb{R}^{L \times d}$

For a given `i`, gives the values for the i^{th} and $i+1^{th}$ dimensions (with a step of 2):

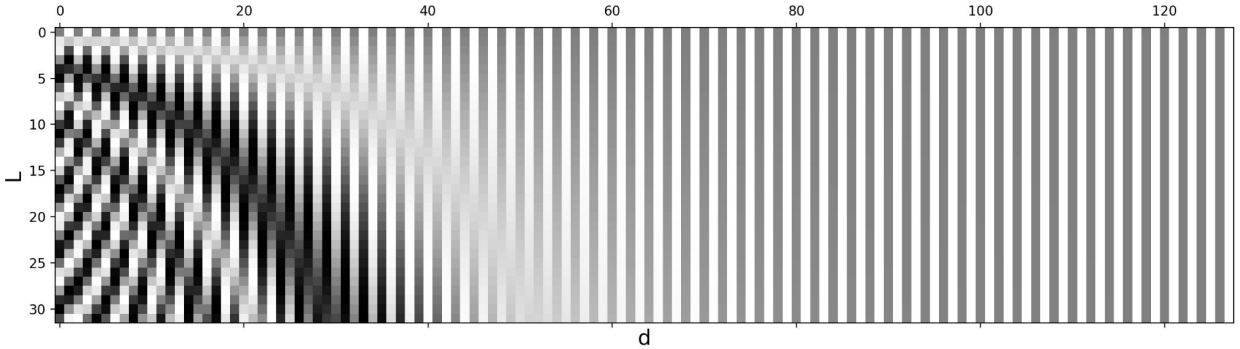
- $PE(pos, 2i) = \sin(pos/10000^{2i/d})$
- $PE(pos, 2i+1) = \cos(pos/10000^{2i/d})$

Positional encoding

Each dimension of the positional encoding corresponds to a sinusoid of different wavelengths in different dimensions, from 2π to $10000*2\pi$

Sequence	Index of token, *pos*	Positional Encoding Matrix with d=4			
		i=0	i=0	i=1	i=1
I	0	$P_{00}=\sin(0)$ = 0	$P_{01}=\cos(0)$ = 1	$P_{02}=\sin(0)$ = 0	$P_{03}=\cos(0)$ = 1
am	1	$P_{10}=\sin(1/1)$ = 0.84	$P_{11}=\cos(1/1)$ = 0.54	$P_{12}=\sin(1/10)$ = 0.10	$P_{13}=\cos(1/10)$ = 1.0
a	2	$P_{20}=\sin(2/1)$ = 0.91	$P_{21}=\cos(2/1)$ = -0.42	$P_{22}=\sin(2/10)$ = 0.20	$P_{23}=\cos(2/10)$ = 0.98
Robot	3	$P_{30}=\sin(3/1)$ = 0.14	$P_{31}=\cos(3/1)$ = -0.99	$P_{32}=\sin(3/10)$ = 0.30	$P_{33}=\cos(3/10)$ = 0.96

Credit: Mehreen Saeed



Credit: lilianweng

Sinusoidal positional encoding with L=32 and d=128

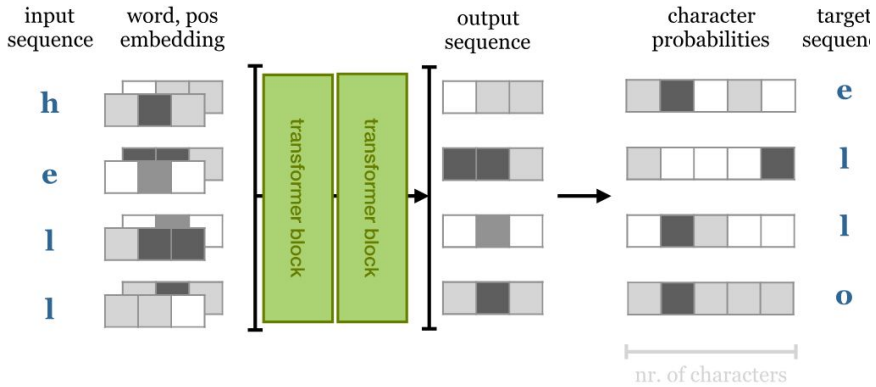
Decoder side:

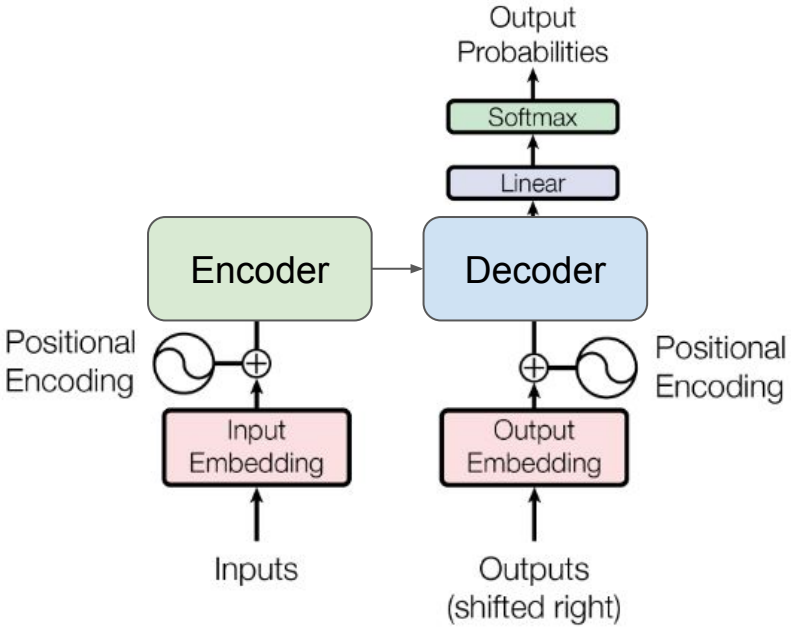
- attention masking
- cross/ encoder-decoder attention

Decoder: masking

Autoregressive model : learn to predict the next element at each point in a sequence

the target output is the same sequence shifted one character to the left





Decoder: masking

With a transformer, the output depends on the entire input sequence, so prediction of the next character becomes vacuously easy, just retrieve it from the input

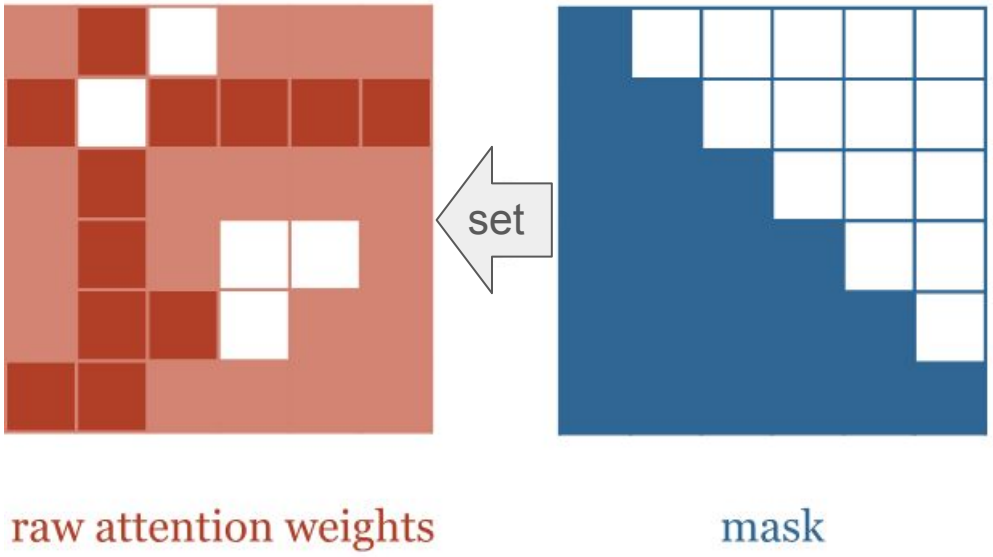
To use self-attention as an autoregressive model, ensure it cannot look forward into the sequence

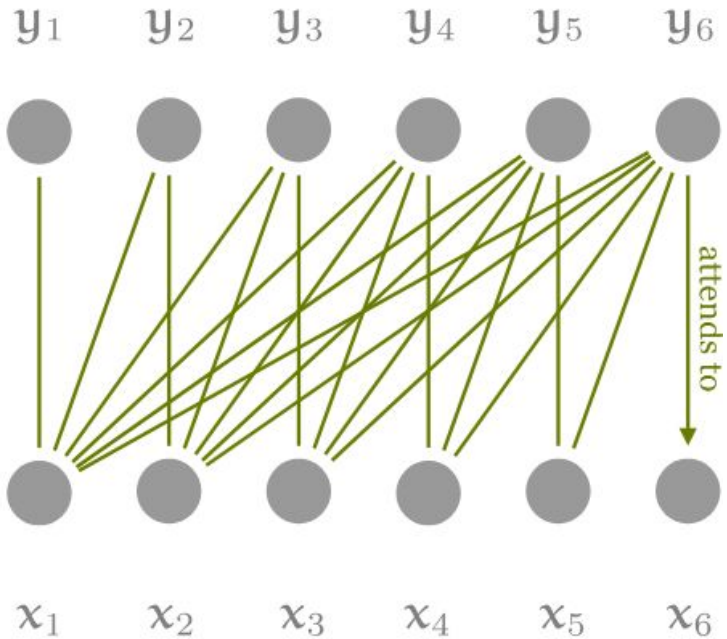
Apply a mask to the matrix of dot products, before the softmax is applied

We actually set the masked out elements (the white squares) to $-\infty$

These elements to be zero after the softmax,

Decoder: masking



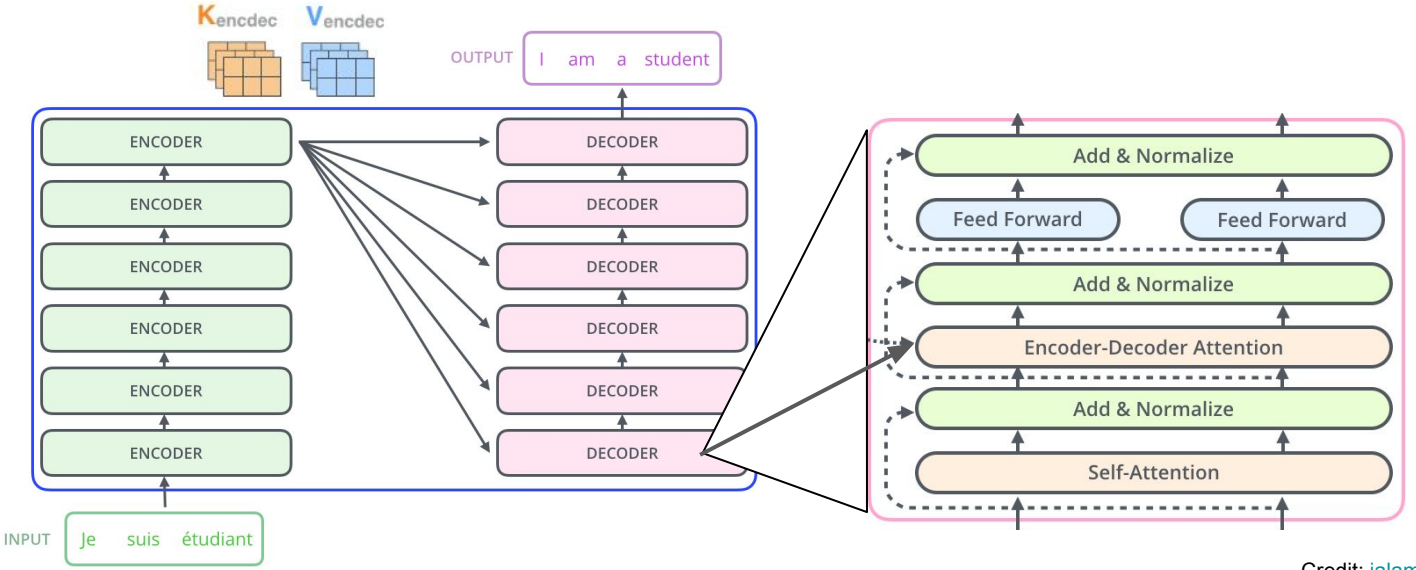


Credit: peterbloem.nl

Cross-/Encoder-Decoder Attention

Encoder output

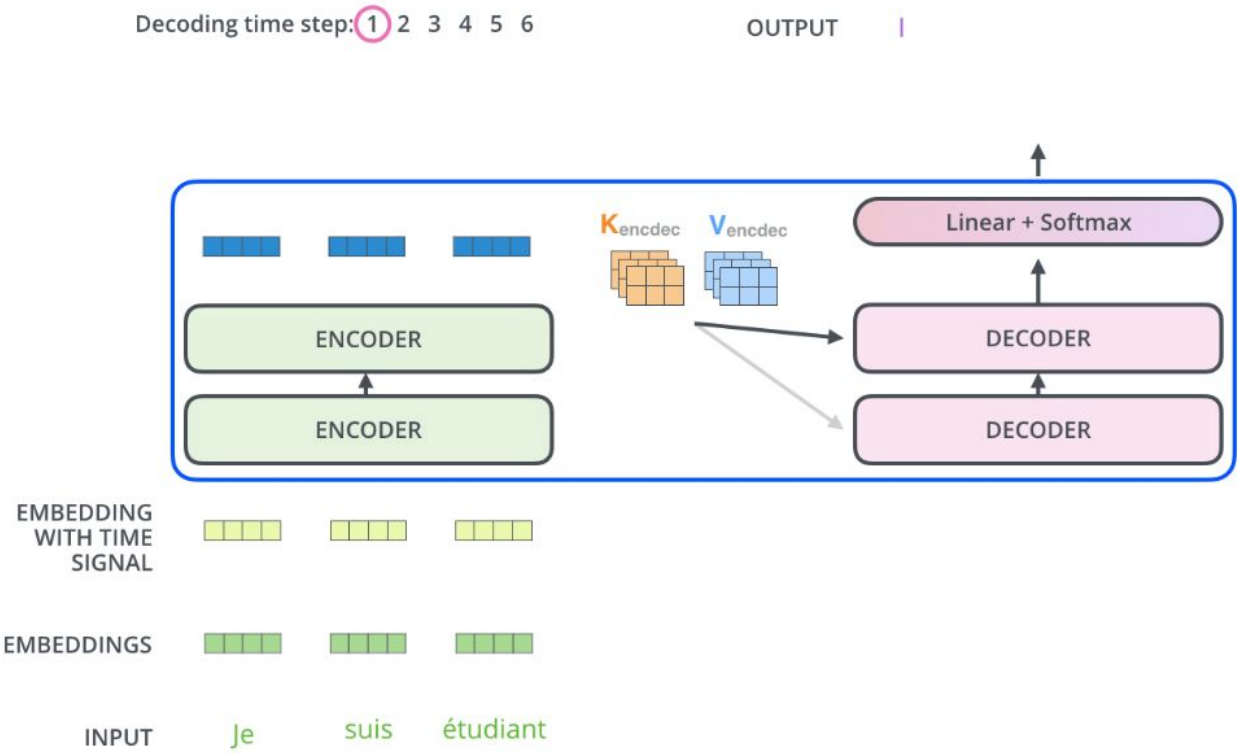
- feeds the **cross attention blocks** of **all decoder blocks** from bottom-up
- under the form of a Key and Value which is used with a Query coming from the decoder



Credit: jalammar

Encoder - Decoder

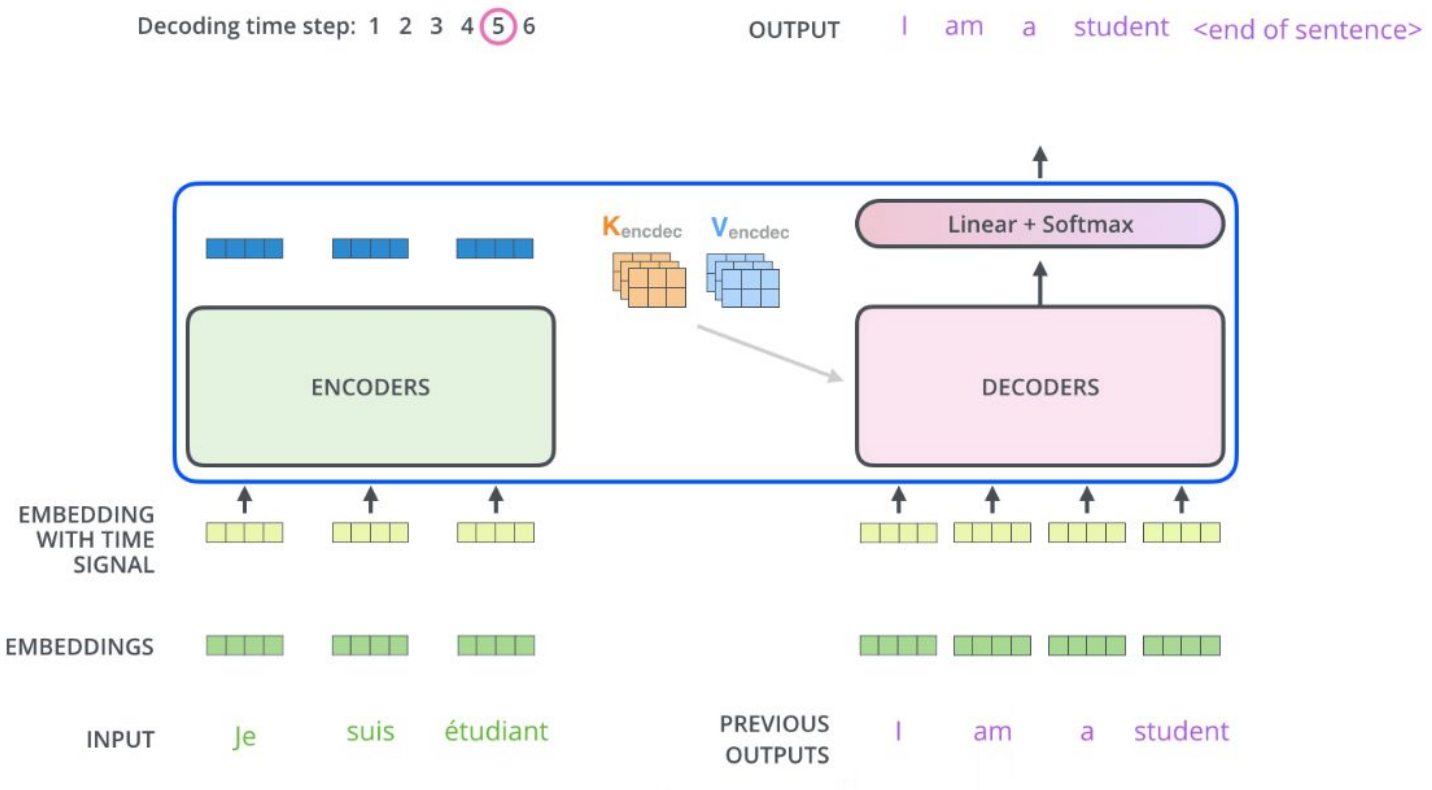
The encoder output a contextual representation for the input sequence (in a bottom up process)



Credit: jalammar

Encoder - Decoder

Using the same KV from encoder final layer, the ouput tokens are concatenated to the decoder input one at a time until the generation (output) of the <eos> token by the decoder



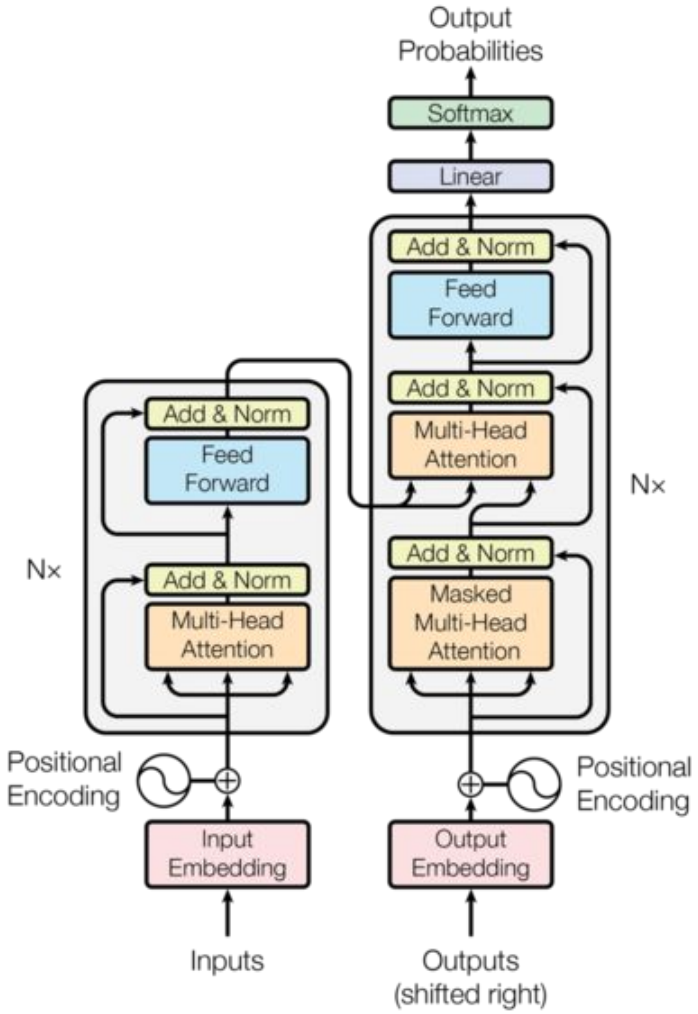
Credit: jalammar

Transformer conclusions

Capture distant dependencies

Allows **parallel processing** (no costly computations of RNN and LSTM)

Quadratic cost with dimension of the sequence



Summary

Summary

Attention main idea: include in the representation of a word the representation of words in the context that influence it

Blocks with FFN+ReLU to introduce non-linearity

3 variants of attention blocks: self attention, masked attention and crossed attention

Positional encoding since attention is position equivariant

Able to attend long range dependencies

Architecture which allows parallel computing but quadratic cost with the number of dimensions

Notations

d, the model size / hidden state dimension / positional encoding size

h, the number of heads in multi-head attention layer

L, the segment length of input sequence.

$X \in \mathbb{R}^{L \times d}$, the input sequence ; each element x_i mapped into an embedding vector of shape, same as the model size

$W_Q \in \mathbb{R}^{d \times d_k}$, the query weight matrix $d_q = d_k$. When $h=1$, $d_k = d$. d_k and (d_k) are written variants

$W_K \in \mathbb{R}^{d \times d_k}$, the key weight matrix

$W_V \in \mathbb{R}^{d \times d_v}$, the value weight matrix. Often we have $d_v = d_k$

$W_{Ki}, W_{Qi} \in \mathbb{R}^{d \times d_k/h}$; $W_{Vi} \in \mathbb{R}^{d \times d_v/h}$, the weight matrices per head i

$W_o \in \mathbb{R}^{h*d_v \times d}$, the output weight matrix

$Q = XW_Q \in \mathbb{R}^{L \times d_k}$, the query embedding inputs with elements q_i

$K = XW_K \in \mathbb{R}^{L \times d_k}$, the key embedding inputs with elements k_i

$V = XW_V \in \mathbb{R}^{L \times d_v}$, the value embedding inputs with elements v_i

$A = softmax(QK^T/\sqrt{d_k}) \in \mathbb{R}^{L \times L}$, the self-attention matrix between a input sequence of length L and itself

$a_{ij} \in A$ the scalar attention score between query q_i and key k_j

$P \in \mathbb{R}^{L \times d}$, position encoding matrix, where the i-th row p_i is the positional encoding for input x_i

References

Transformer

- Bahdanau, D., Cho, K., & Bengio, Y. (2014). Neural machine translation by jointly learning to align and translate. ArXiv:1409.0473.
- The Illustrated Transformer, Jay Alammar, Blog post, 2018
- Transformers from scratch, Peter Bloem, blog post with code, 18 Aug 2019
- UvA Deep Learning Tutorials, Phillip Lippe, 2024, (tutorial 6: Transformers and Multi-Head Attention)
- Speech and Language Processing, Dan Jurafsky and James H. Martin, 3rd ed. draft, Aug. 20, 2024, (chapter 9: The Transformer)
- Dive into Deep Learning, Zhang, Aston and Lipton, Zachary C. and Li, Mu and Smola, Alexander J., Cambridge University Press, 2023, (chapter 11: Attention Mechanisms and Transformers)
- The Transformer family, Lilian Weng, blog post, 2023 Deep Learning
- How large language models work, a visual intro to transformers (Chapter 5), Attention in transformers, visually explained (Chapter 6) and How might LLMs store facts (Chapter 7) 3blue1brown 2024