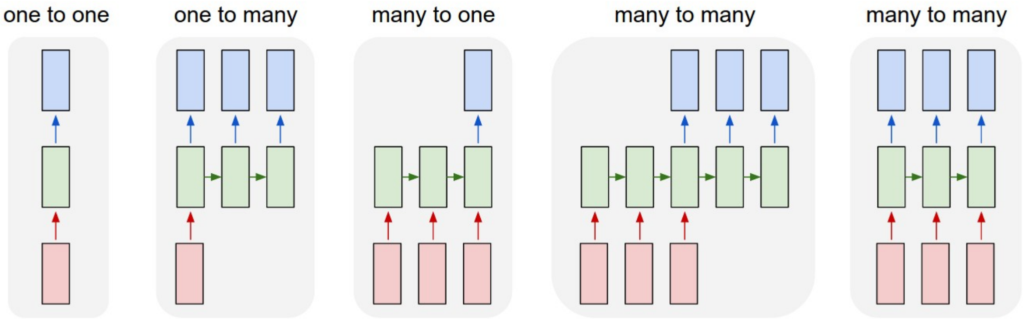
7 Language Models

- Sequence Modeling
  - Recurrent Neural Networks (RNN)
  - Long Short-Term Memory (LSTM)
  - Example: LSTMs in Machine Translation
  - Embeddings From Language Models (ELMo)
- Transformers
  - Elements of Transformers
  - Example: Step-By-Step Calculation
- Bidirectional Encoder Representations from Transformers (BERT)
  - Training BERT
  - Fine-Tuning BERT
- BERT-alike Models
  - Sentence-BERT
  - BERTopic
  - Decoding-enhanced BERT with Disentangled Attention (DeBERTa)
- Large Language Models

Sequence modeling

- e.g., sentiment classification for movie reviews
- word2vec still BOW
- sequence (order) (in most implementations) neglected

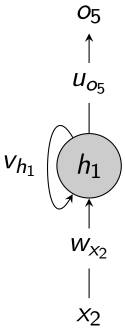


http://karpathy.github.io/2015/05/21/rnn-effectiveness/

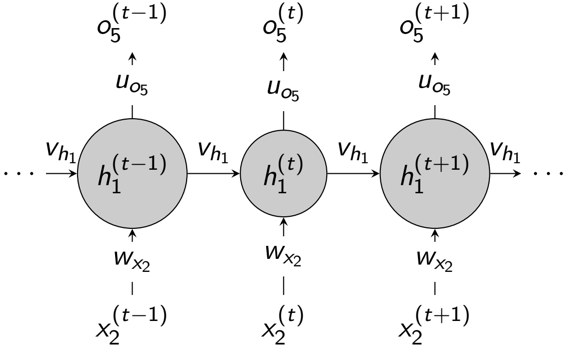
Recurrent neural network (RNN)

- idea: pass information from previous inputs into current calculation
- below: just a snippet of one stream of information, not a full NN!

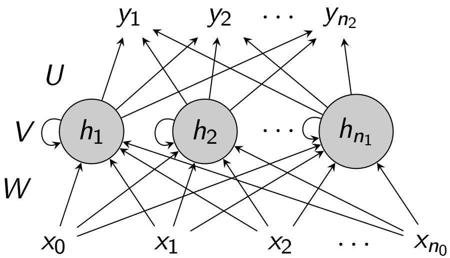
NN representation



unfolded representation

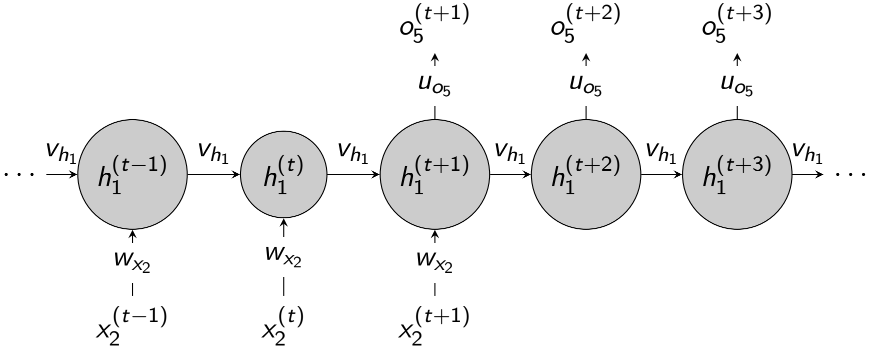


RNN in comparison to FF NN

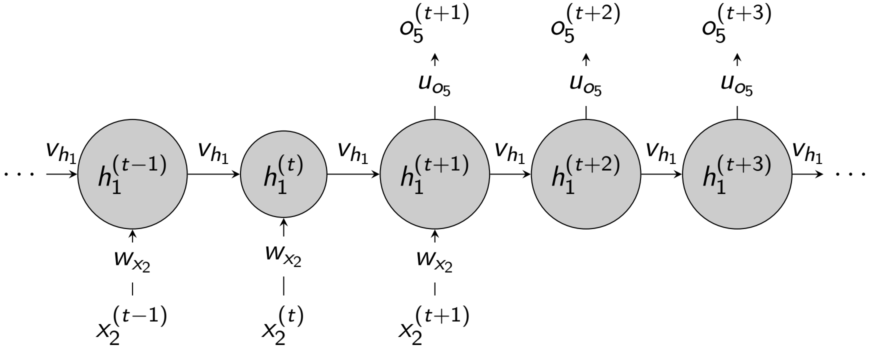


- traditional RNN structure; also known as
  - simple recurrent networks
  - Elman network
  - Jordan network
- more complex architectures possible

RNN architectures

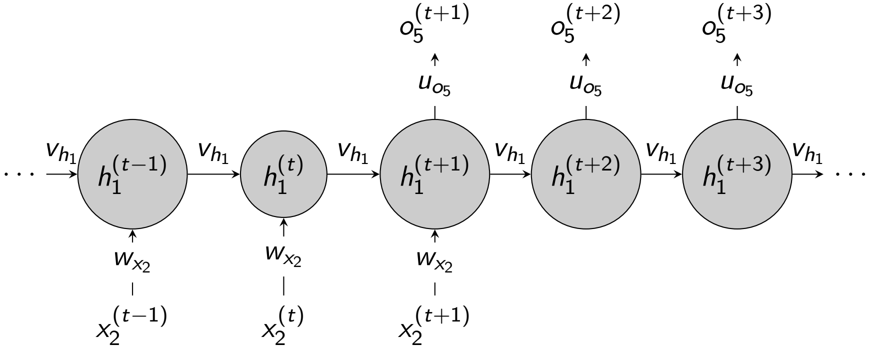


RNN architectures



- many to many, seq2seq
- advantage: short-term **memory**
- disadvantage:

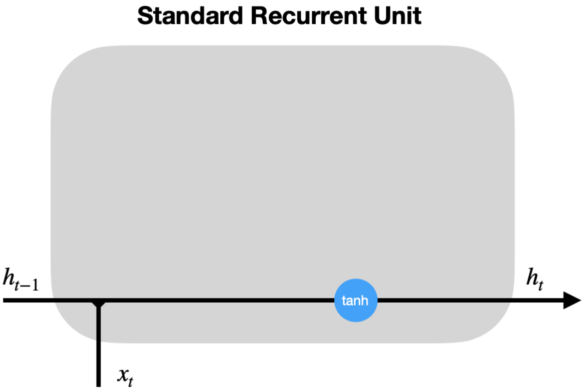
RNN architectures



- many to many, seq2seq
- advantage: short-term **memory**
- disadvantage: **short-term** memory

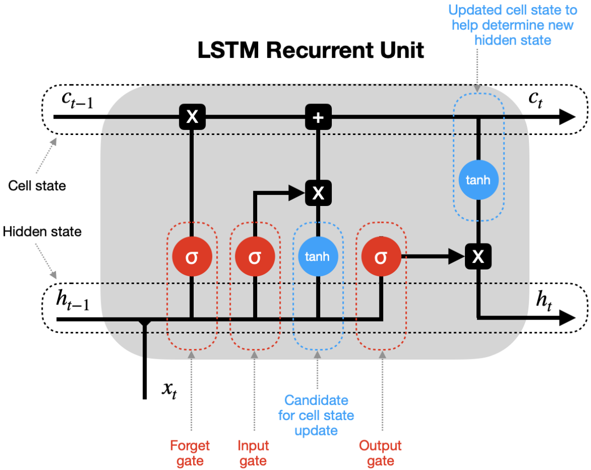
One recurrent unit

- simplify notation: $h_i^{(t)}, i = 1, \ldots, n_1 \quad \rightarrow h_t$
  - analogously we use x_t



https://towardsdatascience.com/lstm-recurrent-neural-networks-how-to-teach-a-network-to-remember-the-past-55e54c2ff22e

Long short-term memory (LSTM)



https://towardsdatascience.com/lstm-recurrent-neural-networks-how-to-teach-a-network-to-remember-the-past-55e54c2ff22e

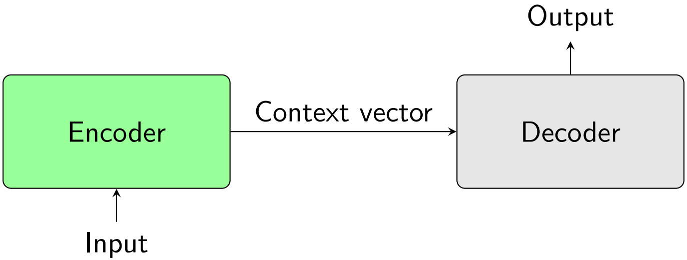
LSTM in real world

- Siri
- Apple's auto-completion
- AlphaGo (software for the board game Go)
- GoogleTranslate was based on LSTMs, besides other systems

- no benefits in one to one scenarios
- (were) popular in seq2seq scenarios

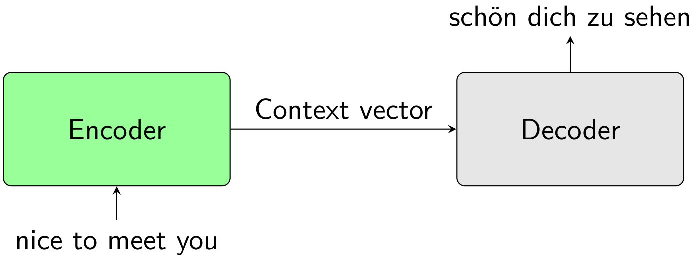
Encoder-decoder

- how to solve machine translation (MT) problem
- idea of encoder-decoder models:
  1. encode on input sequence
  2. decode to output sequence
- use, e.g., 2 LSTMs one after another
- concept used for (all) sota models: BERT, GPT-3, ...

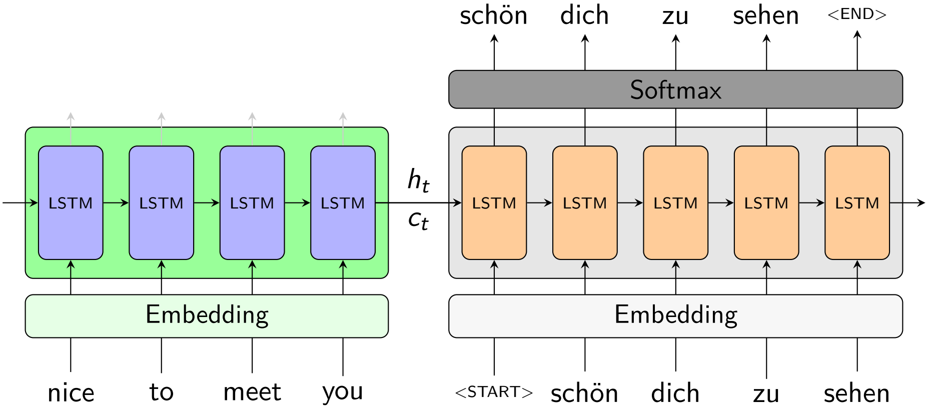


Encoder-decoder in MT

- input: nice to meet you
- task: translation to German
- (target) output: schön dich zu sehen
- encoder: collect information about input in context vector
- context vector: final state of encoder, encapsulates meaning of input
- decoder: predict output token by token using context vector



Example: encoder-decoder LSTM in MT

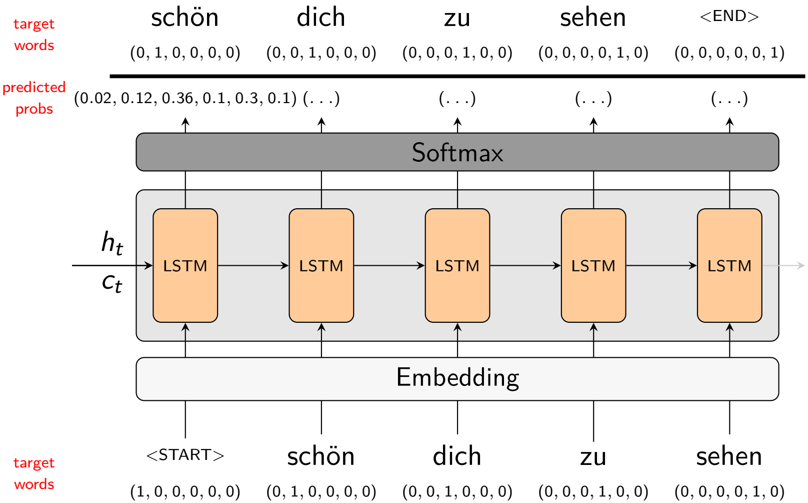


cf. `https://medium.com/analytics-vidhya/encoder-decoder-seq2seq-models-clearly-explained-c34186fbf49b`

Training and testing phases

- vectorizing the tokens: one-hot-encoding → embeddings
- <START> and <END> also get one-hot-encoding & embeddings
- training & testing of encoder identical and straightforward
- instead, for decoder two different non-trivial approaches:
  - teacher forcing while training (cf. next slide):
    use true labels as inputs, not the predicted output sequence
  - during the testing phase the predicted token are used as input for next token — as one-hot-encoded vector passed to the embedding layer

Decoder in the training phase



target words

schön (0, 1, 0, 0, 0, 0)

dich (0, 0, 1, 0, 0, 0)

zu (0, 0, 0, 1, 0, 0)

sehen (0, 0, 0, 0, 1, 0)

<END> (0, 0, 0, 0, 0, 1)

predicted probs

(0.02, 0.12, 0.36, 0.1, 0.3, 0.1) (. . .) (. . .) (. . .) (. . .)

Softmax

h_t

c_t

LSTM LSTM LSTM LSTM LSTM

Embedding

target words

<START> (1, 0, 0, 0, 0, 0)

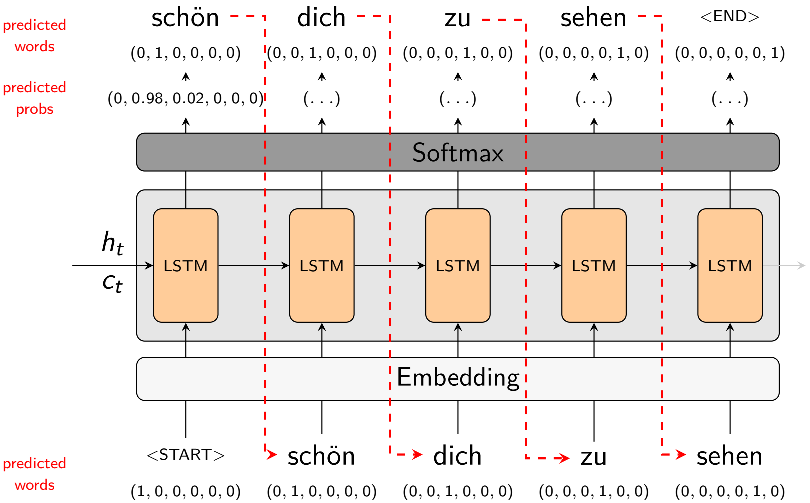
schön (0, 1, 0, 0, 0, 0)

dich (0, 0, 1, 0, 0, 0)

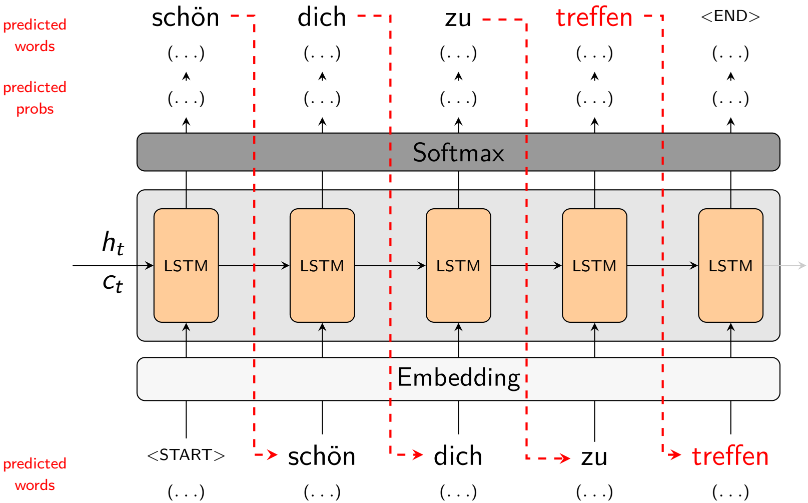
zu (0, 0, 0, 1, 0, 0)

sehen (0, 0, 0, 0, 1, 0)

Decoder in the testing phase



Wrong predictions in testing phase



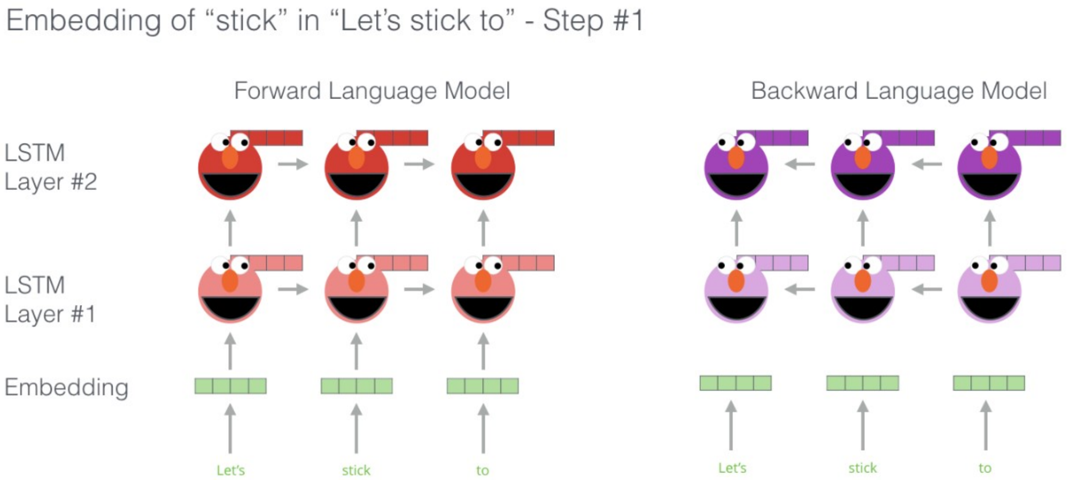
Embeddings from language models (ELMo)

Let’s stick to statistics. The dog gets the stick.

- learn different embeddings for words in different contexts
- word2vec, fastText, GloVe would all result in the same embeddings for both appearances

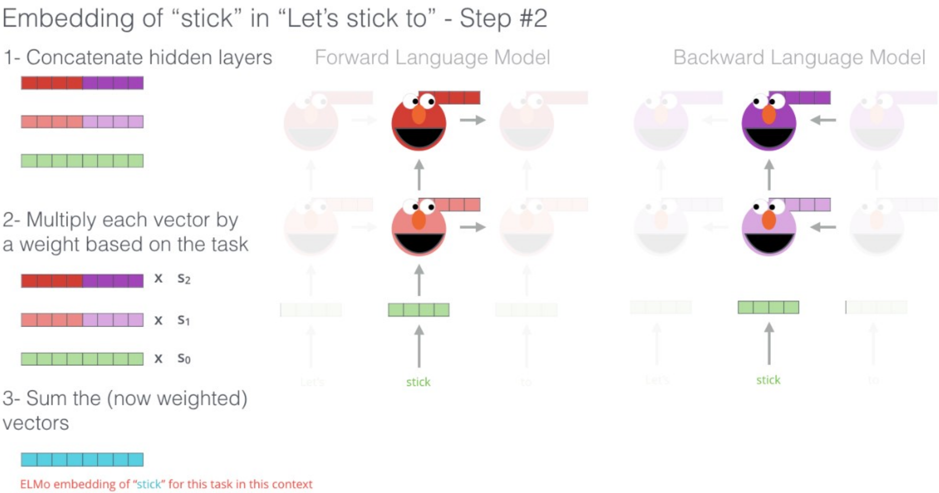
- ELMo: based on LSTM layers
- idea: encode each word including its context
- first model using contextual embeddings
- bidirectional LSTM with LM objective (predict word)

ELMo — step 1



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

ELMo — step 2



Embedding of "stick" in "Let's stick to" - Step #2

1- Concatenate hidden layers

2- Multiply each vector by a weight based on the task

3- Sum the (now weighted) vectors

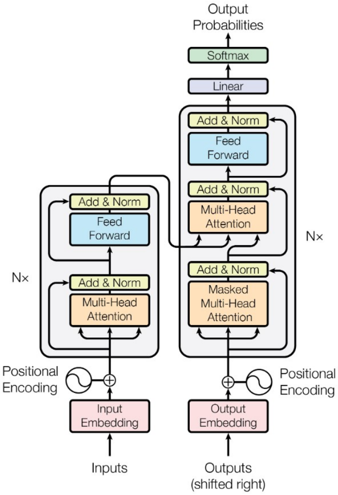
ELMo embedding of "stick" for this task in this context

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

Transformer idea

- RNN cannot be computed in parallel
- transformer enables parallel computation by using
  - attention
  - positional encoding
  - FF layers

- replacement of LSTMs
- better long-term memory



(Vaswani et al., 2017)

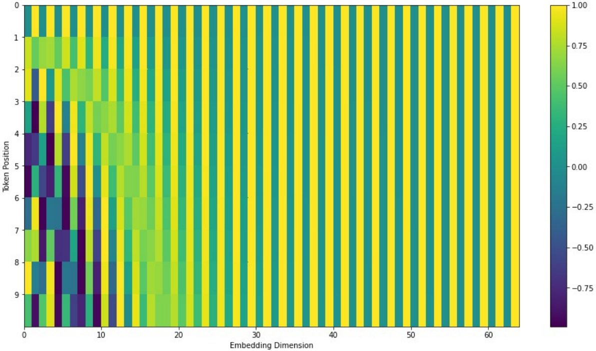
Positional encoding

$$\mathrm{PE}(x \mid \mathrm{pos}) = \begin{cases} \sin(\mathrm{pos}/10000^{x/d}) & \text{if } x \text{ is even} \\ \cos(\mathrm{pos}/10000^{(x-1)/d}) & \text{if } x \text{ is odd} \end{cases}$$

- enables parallel calculation
- is simply added to the word embedding
- pos is the token position $(0, \dots ,)$
- x is the embedding dimension $(0, \dots , d)$

Positional encoding (visualization)

$$\mathrm{PE}(x \mid \mathrm{pos}) = \begin{cases} \sin(\mathrm{pos}/10000^{x/d}) & \text{if } x \text{ is even} \\ \cos(\mathrm{pos}/10000^{(x-1)/d}) & \text{if } x \text{ is odd} \end{cases}$$



https://jalammar.github.io/illustrated-transformer/ — here $d = 64$

Attention is all you need[1]

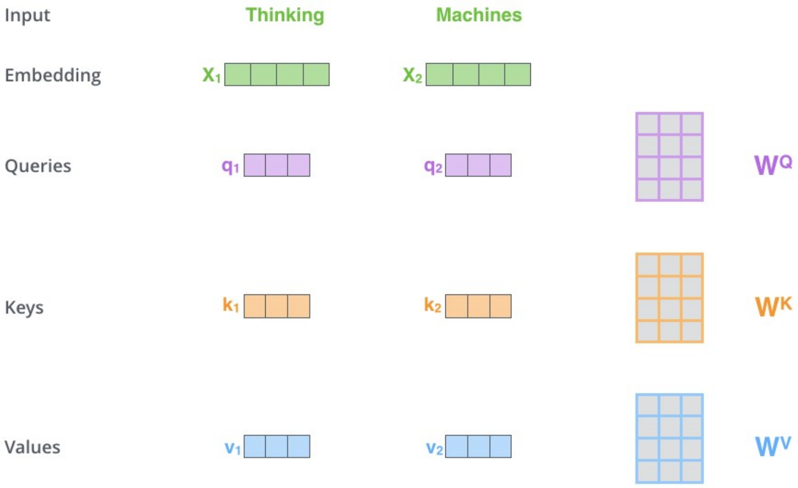
Self-attention

The trophy does not fit into the suitcase because it is too big/small. (cf. slide 29)

- question: what does "it" refers to in the sentence above
- human: if big → trophy, if small → suitcase
- not that easy for LM
- self-attention allows to look at other tokens' embeddings
- similar approach as in RNNs

[1] Attention Is All You Need (Vaswani et al., 2017) `https://doi.org/10.48550/arXiv.1706.03762`

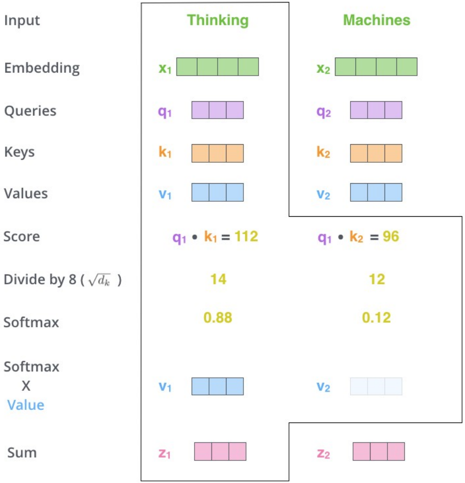
Self-attention elements



- $x_i \in \mathbb{R}^{512}$
- $q_i, k_i, v_i \in \mathbb{R}^{64}$
- $W^Q \in \mathbb{R}^{512 \times 64}$ $(W^K, W^V$ same)
- $V \neq$ vocab size

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

Self-attention calculation



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

1. get q_1, k_1, v_1 by multiplying x_1 with W^Q, W^K, W^V
2. dot-product q_1 and k_i for all i in the same sequence
3. divide by 8
4. calculate softmax
5. weighted sum of v_i
6. z_1 resulting attention vector

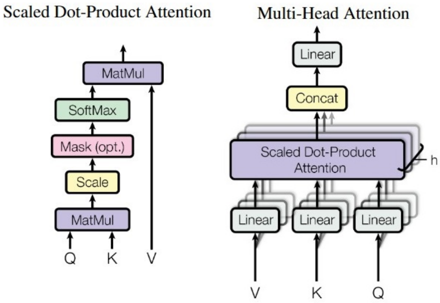
Self-attention matrix representation

- $Q = XW^Q$
- $K = XW^K$
- $V = XW^V$

- scaled dot-product attention
- $Z = \text{softmax}(QK^T/\sqrt{64})V$
- 64 because it is the respective dimension d_k
  $\Rightarrow$ to prevent from extremely small gradients for the softmax function

- in general:
  - $d \times d_k$ dimension of W^Q and W^K
  - $d \times d_v$ dimension of W^V
  - $d_k = d_v = d/h = 64$ often selected $(d = 512)$
  - $h = 8$ number of (multi-head) attention layers (cf. following slides)

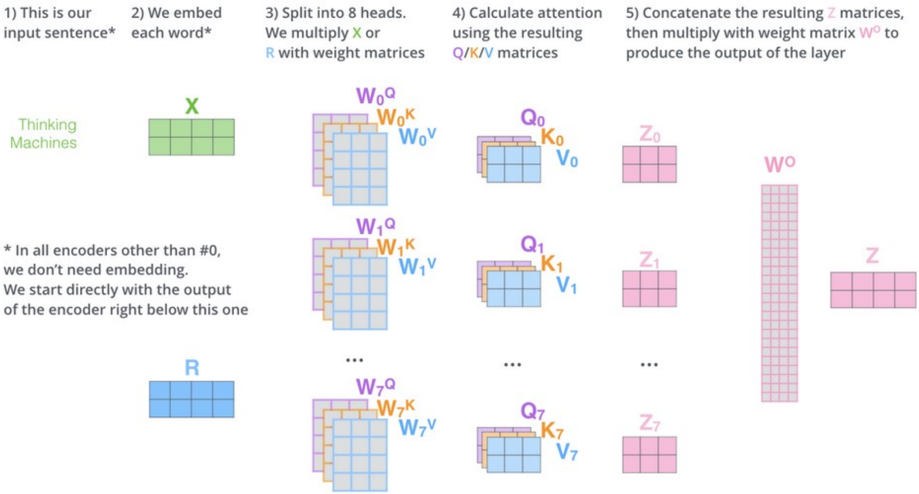
(Simple) self-attention and multi-head attention (MHA)



(Vaswani et al., 2017)

- idea: several (typically $h = 8$) parallel self-attention layers
- each layer returns an attention matrix $Z_i, i = 1, \dots, h$ (cf. previous slide)
- $Z = (Z_1, \dots, Z_8)W^0$
- $W^0 \in \mathbb{R}^{hd_v \times d} (= \mathbb{R}^{512 \times 512})$

MHA in one figure



1) This is our input sentence*

2) We embed each word*

3) Split into 8 heads. We multiply X or R with weight matrices

4) Calculate attention using the resulting Q/K/V matrices

5) Concatenate the resulting Z matrices, then multiply with weight matrix W^O to produce the output of the layer

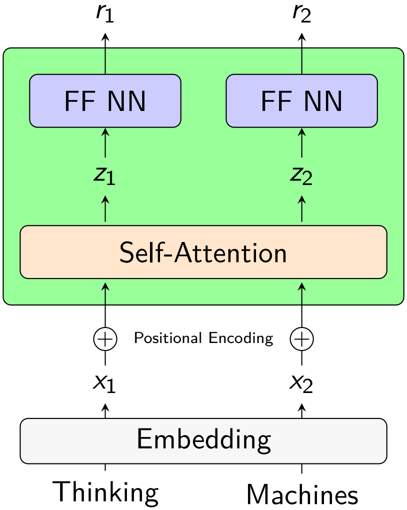
* In all encoders other than #0, we don't need embedding. We start directly with the output of the encoder right below this one

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

Now we're finally encoding

- on the right: first encoder layer
- all other layers get r vectors
- $\text{FFNN}(z) = \max(0, zW_1 + b_1)W_2 + b_2$ (two linear transformations + ReLU)

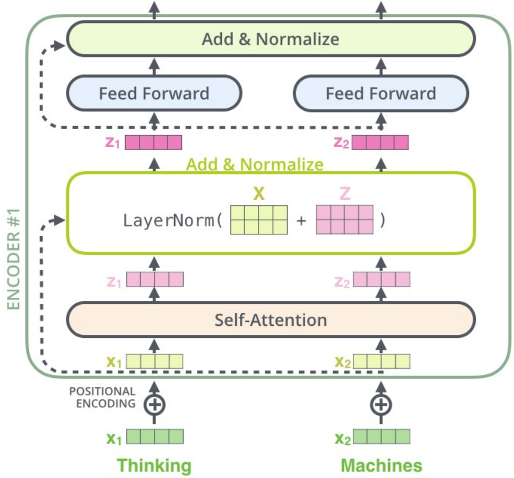
- typically:
  - $x_i \in \mathbb{R}^{512}$
  - $z_i \in \mathbb{R}^{512}$
  - $r_i \in \mathbb{R}^{512}$



Residuals and layer normalization

- skip/residual connection
- allow direct information flow

- (layer) normalization
- across the features
- to avoid covariate shift
- to avoid slow down training



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

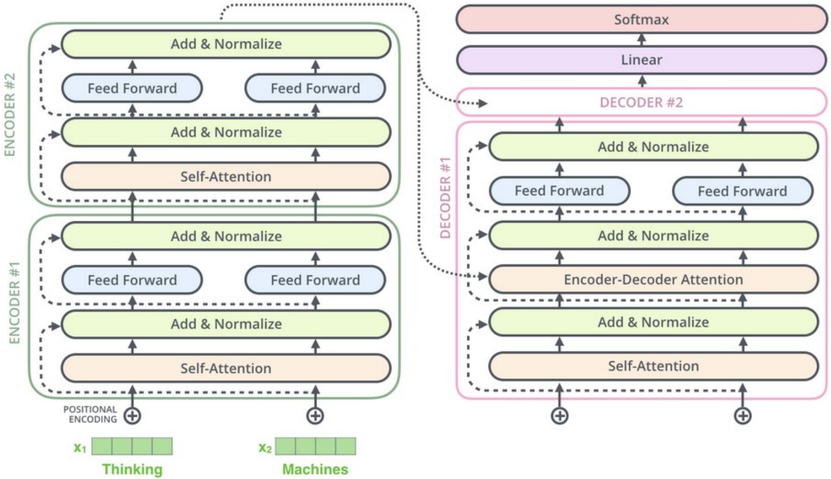
Decoder layers

- decoder is very similar to encoder layer
- masked MHA at the bottom
- padding mask: too long or too short sequences
    - pad 0 after shorter sequences
    - pad $\approx -\infty$ for pieces of longer sequences
- sequence mask:
    - set future positions to $\approx -\infty$
- input to decoder is shifted right
    - prevent from just learning the copy/paste task

Output layer

- after all decoder layers:
  1. linear layer
  2. softmax
- linear layer
  - fully connected NN (up-projection)
  - vocabulary size
  - results in logits
- softmax layer
  - transforms logits to pseudo-probabilities
  - then, arg max a typical prediction

A transformer architecture for $N = 2$



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

cf. also https://medium.com/@yulemoon/detailed-explanations-of-transformer-step-by-step-dc32d90b3a98

Complexity comparison

Layer type	Complexity	Number of sequential operations
Attention	$O(n^2 d)$	$O(1)$
RNN	$O(nd^2)$	$O(n)$
CNN	$O(knd^2)$	$O(1)$

- n is the sequence length
- d is the embedding size
- k is the kernel size
- often $n < d$
- more important: Attention better parallelizable due to less sequential operations

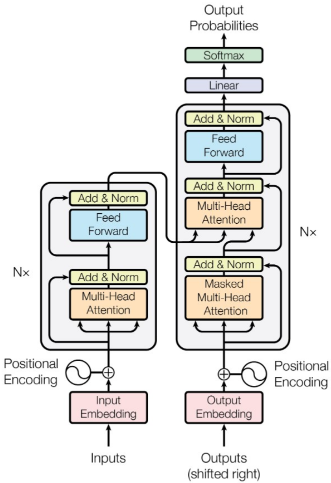
Step-by-step calculation of the transformer architecture[1] (encoder)

1. data, vocabulary, vocabulary size
2. encoding & static embedding
3. positional encoding
4. combining embedding and positional encoding
5. single-head attention (SHA) → multi-head attention (MHA)
    1. calculating query, key, and value matrices
    2. query and key matrix multiplication & scaling
    3. calculating softmax & multiplication with value matrix
    4. repeat, concatenate, linear transformation for final SHA/MHA output matrix
6. add & normalize
7. FFNN

[1] inspired by `https://levelup.gitconnected.com/understanding-transformers-from-start-to-end-a-step-by-step-math-example-16d4e64e6eb1`

Step-by-step calculation of the transformer architecture (decoder)

8. masked MHA
9. decoder MHA
10. predicting tokens



(Vaswani et al., 2017)

Step 1 — data, vocabulary, vocabulary size

- assume a dataset containing (for simplicity: only) three sentences
  1. I drink and I know things.
  2. When you play the game of thrones, you win or you die.
  3. The true enemy won't wait out the storm, he brings the storm.
- tokenizing (for simplicity: without subword information) results in
  1. I, drink, and, I, know, things
  2. when, you, play, the, game, of, thrones, you, win, or, you, die
  3. the, true, enemy, won't, wait, out, the, storm, he, brings, the, storm
- the resulting set of vocabularies is given by

  I, drink, and, know, things, when, you, play, the, game, of, thrones, win, or, die, true, enemy, won't, wait, out, storm, he, brings

- $V = 23$

Step 2 — encoding

encoding (just mixing up the vocabularies a little bit for demonstration):

1	2	3	4	5	6	7	8	9	10	11	12
I	drink	things	know	when	won't	play	out	true	storm	brings	game
13	14	15	16	17	18	19	20	21	22	23	
the	win	of	enemy	you	wait	thrones	and	or	die	he	

we select the second sentence as an example and input the first part into the encoder of the transformer, i.e.

when	you	play	game	of	thrones
5	17	7	12	15	19

Step 2 — static embedding

d	when	you	play	game	of	thrones
	5	17	7	12	15	19
1	0.79	0.38	0.01	0.12	0.88	0.60
2	0.60	-0.37	1.93	1.73	1.24	1.02
3	0.96	0.01	0.18	0.52	0.62	0.53
4	0.64	-0.21	0.31	-0.77	-0.36	0.51
5	0.97	0.90	0.56	0.06	0.49	0.93
6	0.20	-0.26	0.59	-0.63	-0.30	0.21

- the attention paper uses $d = 512$, we select $d = 6$ for demonstration
- the input embedding layer (somewhat a *lookup* table) is initialized randomly and updated during training
- the (current) static embedding for *of* is $(0.88, 1.25, 0.04, -0.03, 0.32, -0.28)$

Step 3 — positional encoding I

$$\mathrm{PE}(x \mid \mathrm{pos}) = \begin{cases} \sin(\mathrm{pos}/10000^{x/d}) & \text{if } x \text{ is even} \\ \cos(\mathrm{pos}/10000^{(x-1)/d}) & \text{if } x \text{ is odd} \end{cases}$$

- let us encode the position of the token *of*, which is the fifth token in our example sentence
- since programming languages usually start counting at 0, this refers to token position 4

pos	x	even/odd	formula	$\mathrm{PE}(x \mid \mathrm{pos})$
4	0	even	$\sin(4/10000^{0/6})$	-0.7568
4	1	odd	$\cos(4/10000^{0/6})$	-0.6536
4	2	even	$\sin(4/10000^{2/6})$	0.1846
4	3	odd	$\cos(4/10000^{2/6})$	0.9828
4	4	even	$\sin(4/10000^{4/6})$	0.0086
4	5	odd	$\cos(4/10000^{4/6})$	1.0000

- the positional encoding for *of* is given by the PE column

Step 3 — positional encoding II

- applying the same calculation to all input positions results in

d	when	you	play	game	of	thrones
	5	17	7	12	15	19
1	0.0000	0.8415	0.9093	0.1411	-0.7568	-0.9589
2	1.0000	0.5403	-0.4161	-0.9900	-0.6536	0.2837
3	0.0000	0.0464	0.0927	0.1388	0.1846	0.2300
4	1.0000	0.9989	0.9957	0.9903	0.9828	0.9732
5	0.0000	0.0022	0.0043	0.0065	0.0086	0.0108
6	1.0000	1.0000	1.0000	1.0000	1.0000	0.9999

Step 4 — combining embedding and positional encoding

d	when	you	play	game	of	thrones
	5	17	7	12	15	19
1	0.79	0.38	0.01	0.12	0.88	0.60
2	0.60	-0.37	1.93	1.73	1.24	1.02
3	0.96	0.01	0.18	0.52	0.62	0.53
4	0.64	-0.21	0.31	-0.77	-0.36	0.51
5	0.97	0.90	0.56	0.06	0.49	0.93
6	0.20	-0.26	0.59	-0.63	-0.30	0.21

(input embedding) + (positional encoding)

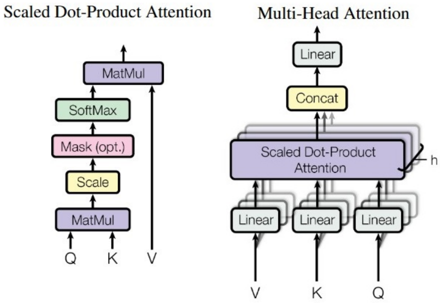
d	when	you	play	game	of	thrones
	5	17	7	12	15	19
1	0.0000	0.8415	0.9093	0.1411	-0.7568	-0.9589
2	1.0000	0.5403	-0.4161	-0.9900	-0.6536	0.2837
3	0.0000	0.0464	0.0927	0.1388	0.1846	0.2300
4	1.0000	0.9989	0.9957	0.9903	0.9828	0.9732
5	0.0000	0.0022	0.0043	0.0065	0.0086	0.0108
6	1.0000	1.0000	1.0000	1.0000	1.0000	0.9999

=

(input matrix for multi-head attention)

d	when	you	play	game	of	thrones
	5	17	7	12	15	19
1	0.79	1.22	0.92	0.26	0.12	-0.36
2	1.60	0.17	1.51	0.74	0.59	1.30
3	0.96	0.06	0.27	0.66	0.80	0.76
4	1.64	0.79	1.31	0.22	0.62	1.48
5	0.97	0.90	0.56	0.07	0.50	0.94
6	1.20	0.74	1.59	0.37	0.70	1.21

Step 5 — single-head attention (SHA) → multi-head attention (MHA)



1. calculating query, key, and value matrices
2. query and key matrix multiplication & scaling
3. calculating softmax & multiplication with value matrix
4. repeat, concatenate & linear transformation for final SHA/MHA output matrix

Step 5.1 — calculating query, key, and value matrices

(input matrix for multi-head attention)

$$\begin{array}{l|rrrrrr} \text{when} & 0.79 & 1.60 & 0.96 & 1.64 & 0.97 & 1.20 \\ \text{you} & 1.22 & 0.17 & 0.06 & 0.79 & 0.90 & 0.74 \\ \text{play} & 0.92 & 1.51 & 0.27 & 1.31 & 0.56 & 1.59 \\ \text{game} & 0.26 & 0.74 & 0.66 & 0.22 & 0.07 & 0.37 \\ \text{of} & 0.12 & 0.59 & 0.80 & 0.62 & 0.50 & 0.70 \\ \text{thrones} & -0.36 & 1.30 & 0.76 & 1.48 & 0.94 & 1.21 \end{array} \quad \times$$

(linear weights for query W^Q) (query matrix Q)

$$\begin{array}{rrrr} 0.52 & 0.45 & 0.91 & 0.69 \\ 0.05 & 0.85 & 0.37 & 0.83 \\ 0.49 & 0.10 & 0.56 & 0.61 \\ 0.71 & 0.64 & 0.40 & 0.14 \\ 0.76 & 0.27 & 0.92 & 0.67 \\ 0.85 & 0.56 & 0.57 & 0.07 \end{array} = \begin{array}{rrrr} 3.88 & 3.80 & 4.08 & 3.42 \\ 2.55 & 1.86 & 2.77 & 1.78 \\ 3.39 & 3.60 & 3.49 & 2.72 \\ 1.02 & 1.18 & 1.24 & 1.30 \\ 1.90 & 1.56 & 1.88 & 1.53 \\ 3.04 & 2.90 & 2.73 & 2.22 \end{array}$$

(linear weights for key W^K) (key matrix K)

$$\begin{array}{rrrr} 0.74 & 0.57 & 0.21 & 0.73 \\ 0.55 & 0.16 & 0.90 & 0.17 \\ 0.25 & 0.74 & 0.80 & 0.98 \\ 0.80 & 0.73 & 0.20 & 0.31 \\ 0.37 & 0.96 & 0.42 & 0.08 \\ 0.28 & 0.41 & 0.87 & 0.86 \end{array} = \begin{array}{rrrr} 3.71 & 4.04 & 4.15 & 3.41 \\ 2.18 & 2.51 & 1.64 & 1.93 \\ 3.28 & 3.11 & 3.65 & 3.01 \\ 1.07 & 1.13 & 1.64 & 1.35 \\ 1.49 & 1.97 & 2.14 & 1.81 \\ 2.51 & 3.04 & 3.45 & 2.28 \end{array}$$

(linear weights for value W^V) (value matrix V)

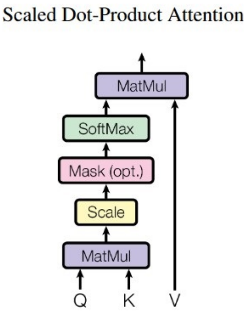
$$\begin{array}{rrrr} 0.62 & 0.07 & 0.70 & 0.95 \\ 0.20 & 0.97 & 0.61 & 0.35 \\ 0.57 & 0.80 & 0.61 & 0.50 \\ 0.67 & 0.35 & 0.98 & 0.54 \\ 0.47 & 0.83 & 0.34 & 0.94 \\ 0.60 & 0.69 & 0.13 & 0.98 \end{array} = \begin{array}{rrrr} 3.63 & 4.58 & 4.21 & 4.76 \\ 2.22 & 1.83 & 2.17 & 3.25 \\ 3.12 & 3.77 & 3.41 & 4.33 \\ 1.09 & 1.65 & 1.32 & 1.38 \\ 1.72 & 2.34 & 1.80 & 2.21 \\ 2.63 & 3.98 & 2.93 & 3.36 \end{array}$$

- we select $d_k = d_v = 4$ for simplicity

Step 5.2 — query and key matrix multiplication & scaling

(query matrix Q) (transposed key matrix K^T) (QK^T)

$$\begin{matrix} 3.88 & 3.80 & 4.08 & 3.42 \\ 2.55 & 1.86 & 2.77 & 1.78 \\ 3.39 & 3.60 & 3.49 & 2.72 \\ 1.02 & 1.18 & 1.24 & 1.30 \\ 1.90 & 1.56 & 1.88 & 1.53 \\ 3.04 & 2.90 & 2.73 & 2.22 \end{matrix} \times \begin{matrix} 3.71 & 2.18 & 3.28 & 1.07 & 1.49 & 2.51 \\ 4.04 & 2.51 & 3.11 & 1.13 & 1.97 & 3.04 \\ 4.15 & 1.64 & 3.65 & 1.64 & 2.14 & 3.45 \\ 3.41 & 1.93 & 3.01 & 1.35 & 1.81 & 2.28 \end{matrix} = \begin{matrix} 58.34 & 31.29 & 49.73 & 19.75 & 28.19 & 43.16 \\ 34.54 & 18.21 & 29.62 & 11.78 & 16.61 & 25.67 \\ 50.88 & 27.40 & 43.24 & 17.09 & 24.53 & 37.70 \\ 18.13 & 9.73 & 15.45 & 6.21 & 8.85 & 13.39 \\ 26.37 & 14.09 & 22.55 & 8.94 & 12.70 & 19.49 \\ 41.89 & 22.67 & 35.64 & 14.00 & 20.10 & 30.93 \end{matrix}$$



- calculating QK^T
- scaling with $\sqrt{d_k} = \sqrt{4} = 2$

($QK^T/2$)

$$\begin{matrix} 29.17 & 15.64 & 24.86 & 9.88 & 14.10 & 21.58 \\ 17.27 & 9.11 & 14.81 & 5.89 & 8.30 & 12.84 \\ 25.44 & 13.70 & 21.62 & 8.54 & 12.27 & 18.85 \\ 9.06 & 4.87 & 7.72 & 3.10 & 4.42 & 6.70 \\ 13.19 & 7.04 & 11.28 & 4.47 & 6.35 & 9.74 \\ 20.95 & 11.34 & 17.82 & 7.00 & 10.05 & 15.46 \end{matrix}$$

Step 5.3 — calculating softmax

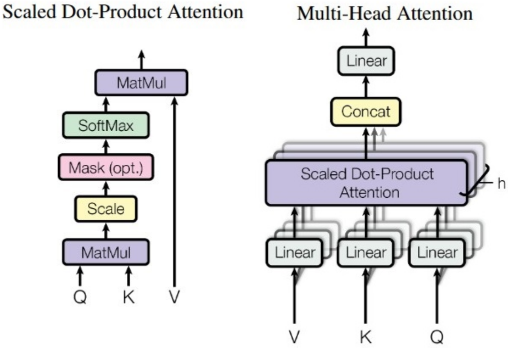
$$\text{softmax}(z_i) = \exp(z_i) / \sum_{j=1}^{d} \exp(z_j), \quad i = 1, \ldots, d$$

$$\begin{aligned}\text{softmax}(29.17) = \exp(29.17)/(&\exp(29.17) + \exp(15.64) + \exp(24.86) + \\ &\exp(9.88) + \exp(14.10) + \exp(21.58)) = 0.9862\end{aligned}$$

$$\text{softmax}(QK^T/2) = \begin{array}{cccccc|l} 0.9862 & 0.0000 & 0.0133 & 0.0000 & 0.0000 & 0.0005 & \text{when} \\ 0.9111 & 0.0003 & 0.0777 & 0.0000 & 0.0001 & 0.0108 & \text{you} \\ 0.9772 & 0.0000 & 0.0214 & 0.0000 & 0.0000 & 0.0013 & \text{play} \\ 0.7231 & 0.0108 & 0.1897 & 0.0019 & 0.0070 & 0.0676 & \text{game} \\ 0.8450 & 0.0018 & 0.1251 & 0.0001 & 0.0009 & 0.0270 & \text{of} \\ 0.9542 & 0.0001 & 0.0418 & 0.0000 & 0.0000 & 0.0040 & \text{thrones} \end{array}$$

Step 5.3 — multiplication of softmax and value matrix

$$\overset{\text{softmax}(QK^T/2)}{\begin{matrix} 0.9862 & 0.0000 & 0.0133 & 0.0000 & 0.0000 & 0.0005 \\ 0.9111 & 0.0003 & 0.0777 & 0.0000 & 0.0001 & 0.0108 \\ 0.9772 & 0.0000 & 0.0214 & 0.0000 & 0.0000 & 0.0013 \\ 0.7231 & 0.0108 & 0.1897 & 0.0019 & 0.0070 & 0.0676 \\ 0.8450 & 0.0018 & 0.1251 & 0.0001 & 0.0009 & 0.0270 \\ 0.9542 & 0.0001 & 0.0418 & 0.0000 & 0.0000 & 0.0040 \end{matrix}} \times \overset{(\text{value matrix } V)}{\begin{matrix} 3.63 & 4.58 & 4.21 & 4.76 \\ 2.22 & 1.83 & 2.17 & 3.25 \\ 3.12 & 3.77 & 3.41 & 4.33 \\ 1.09 & 1.65 & 1.32 & 1.38 \\ 1.72 & 2.34 & 1.80 & 2.21 \\ 2.63 & 3.98 & 2.93 & 3.36 \end{matrix}} = \overset{(Z)}{\begin{matrix} 3.6227 & 4.5689 & 4.1987 & 4.7536 \\ 3.5790 & 4.5095 & 4.1332 & 4.7108 \\ 3.6174 & 4.5614 & 4.1908 & 4.7485 \\ 3.4326 & 4.3353 & 3.9277 & 4.5437 \\ 3.5343 & 4.4548 & 4.0688 & 4.6626 \\ 3.6049 & 4.5439 & 4.1717 & 4.7368 \end{matrix}}$$


Scaled Dot-Product Attention

Multi-Head Attention

- calculating $Z = \text{softmax}(QK^T/\sqrt{d_k})V$
- this is the last step in the SHA setting
- in this example, we won't repeat this procedure h times to get MHA setting
  - if so: concatenate Z matrices

Step 5.4 — linear transformation for final SHA/MHA output matrix

$$\overset{(Z)}{\begin{matrix} 3.6227 & 4.5689 & 4.1987 & 4.7536 \\ 3.5790 & 4.5095 & 4.1332 & 4.7108 \\ 3.6174 & 4.5614 & 4.1908 & 4.7485 \\ 3.4326 & 4.3353 & 3.9277 & 4.5437 \\ 3.5343 & 4.4548 & 4.0688 & 4.6626 \\ 3.6049 & 4.5439 & 4.1717 & 4.7368 \end{matrix}} \times \overset{(\text{linear weight matrix } W^0)}{\begin{matrix} 0.80 & 0.34 & 0.45 & 0.54 & 0.07 & 0.53 \\ 0.85 & 0.74 & 0.78 & 0.50 & 0.75 & 0.55 \\ 0.53 & 0.81 & 0.55 & 0.59 & 0.49 & 0.14 \\ 0.70 & 0.60 & 0.12 & 0.42 & 0.29 & 0.87 \end{matrix}} = \overset{(\text{output matrix of multi-head attention})}{\begin{matrix} 12.33 & 10.87 & 8.07 & 8.71 & 7.12 & 9.16 \\ 12.18 & 10.73 & 7.97 & 8.60 & 7.02 & 9.05 \\ 12.32 & 10.85 & 8.06 & 8.70 & 7.10 & 9.14 \\ 11.69 & 10.28 & 7.63 & 8.25 & 6.73 & 8.71 \\ 12.03 & 10.59 & 7.86 & 8.49 & 6.93 & 8.95 \\ 12.27 & 10.81 & 8.03 & 8.67 & 7.08 & 9.11 \end{matrix}}$$

- once again we calculate a linear transformation, here: ZW^0
- dimensions of W^0 have to be set in order to get an output matrix that matches the dimensions of the input

Step 6 — add

$$\begin{array}{l|rrrrrr}
\text{when} & 0.79 & 1.60 & 0.96 & 1.64 & 0.97 & 1.20 \\
\text{you} & 1.22 & 0.17 & 0.06 & 0.79 & 0.90 & 0.74 \\
\text{play} & 0.92 & 1.51 & 0.27 & 1.31 & 0.56 & 1.59 \\
\text{game} & 0.26 & 0.74 & 0.66 & 0.22 & 0.07 & 0.37 \\
\text{of} & 0.12 & 0.59 & 0.80 & 0.62 & 0.50 & 0.70 \\
\text{thrones} & -0.36 & 1.30 & 0.76 & 1.48 & 0.94 & 1.21
\end{array}$$

(MHA input matrix) + (MHA output matrix)

$$\begin{array}{l|rrrrrr}
\text{when} & 12.33 & 10.87 & 8.07 & 8.71 & 7.12 & 9.16 \\
\text{you} & 12.18 & 10.73 & 7.97 & 8.60 & 7.02 & 9.05 \\
\text{play} & 12.32 & 10.85 & 8.06 & 8.70 & 7.10 & 9.14 \\
\text{game} & 11.69 & 10.28 & 7.63 & 8.25 & 6.73 & 8.71 \\
\text{of} & 12.03 & 10.59 & 7.86 & 8.49 & 6.93 & 8.95 \\
\text{thrones} & 12.27 & 10.81 & 8.03 & 8.67 & 7.08 & 9.11
\end{array}$$

$$=$$

(matrix to normalize)

$$\begin{array}{l|rrrrrr}
\text{when} & 13.12 & 12.47 & 9.03 & 10.35 & 8.09 & 10.36 \\
\text{you} & 13.40 & 10.90 & 8.03 & 9.39 & 7.92 & 9.79 \\
\text{play} & 13.24 & 12.36 & 8.33 & 10.01 & 7.66 & 10.73 \\
\text{game} & 11.95 & 11.02 & 8.29 & 8.47 & 6.80 & 9.08 \\
\text{of} & 12.15 & 11.18 & 8.66 & 9.11 & 7.43 & 9.65 \\
\text{thrones} & 11.91 & 12.11 & 8.79 & 10.15 & 8.02 & 10.32
\end{array}$$

- the input and output matrices are just added together

Step 6 — normalize

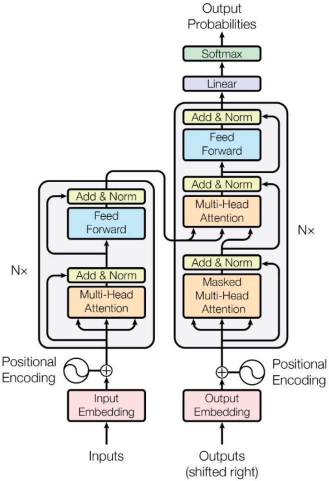
(matrix to normalize) (matrix after normalization)

$$
\begin{array}{l|rrrrrr}
\text{when} & 13.12 & 12.47 & 9.03 & 10.35 & 8.09 & 10.36 \\
\text{you} & 13.40 & 10.90 & 8.03 & 9.39 & 7.92 & 9.79 \\
\text{play} & 13.24 & 12.36 & 8.33 & 10.01 & 7.66 & 10.73 \\
\text{game} & 11.95 & 11.02 & 8.29 & 8.47 & 6.80 & 9.08 \\
\text{of} & 12.15 & 11.18 & 8.66 & 9.11 & 7.43 & 9.65 \\
\text{thrones} & 11.91 & 12.11 & 8.79 & 10.15 & 8.02 & 10.32
\end{array}
\quad \rightarrow \quad
\begin{array}{l|rrrrrr}
\text{when} & 1.3176 & 0.9817 & -0.7957 & -0.1137 & -1.2814 & -0.1085 \\
\text{you} & 1.7078 & 0.4862 & -0.9162 & -0.2516 & -0.9699 & -0.0562 \\
\text{play} & 1.3026 & 0.9007 & -0.9402 & -0.1728 & -1.2463 & 0.1561 \\
\text{game} & 1.4140 & 0.9236 & -0.5159 & -0.4209 & -1.3015 & -0.0993 \\
\text{of} & 1.4270 & 0.8628 & -0.6030 & -0.3412 & -1.3184 & -0.0271 \\
\text{thrones} & 1.0371 & 1.1596 & -0.8738 & -0.0408 & -1.3454 & 0.0633
\end{array}
$$

- row-wise (token-wise) we determine mean $\bar{x}$ and standard deviation s_x
- we calculate a classical normalization $\frac{x-\bar{x}}{s_x+\epsilon}$
- we here select $\epsilon = 0.0001$

Step 7 — FFNN

- after *add & norm*, a classical FFNN is applied to the resulting matrix
- for the example, we assume this to be very simplistic
    - here: one linear layer + ReLU activation function $\max\{0, XW + b\}$
        - max is applied element-wise, b is added per row
    - realistic: multiple linear layers with activation functions



(Vaswani et al., 2017)

Step 7 — FFNN linear layer

(matrix after normalization)

$$\begin{array}{l|rrrrrr}
\text{when} & 1.3176 & 0.9817 & -0.7957 & -0.1137 & -1.2814 & -0.1085 \\
\text{you} & 1.7078 & 0.4862 & -0.9162 & -0.2516 & -0.9699 & -0.0562 \\
\text{play} & 1.3026 & 0.9007 & -0.9402 & -0.1728 & -1.2463 & 0.1561 \\
\text{game} & 1.4140 & 0.9236 & -0.5159 & -0.4209 & -1.3015 & -0.0993 \\
\text{of} & 1.4270 & 0.8628 & -0.6030 & -0.3412 & -1.3184 & -0.0271 \\
\text{thrones} & 1.0371 & 1.1596 & -0.8738 & -0.0408 & -1.3454 & 0.0633
\end{array}
\times$$

(weight matrix W)

$$\begin{array}{rrrrrr}
0.50 & 0.05 & 0.97 & 0.22 & 0.56 & 0.02 \\
0.17 & 0.52 & 0.63 & 0.48 & 0.06 & 0.60 \\
0.53 & 0.87 & 0.47 & 0.10 & 0.31 & 0.79 \\
0.83 & 0.58 & 0.38 & 0.09 & 0.64 & 0.25 \\
0.81 & 0.85 & 0.74 & 0.35 & 0.31 & 0.53 \\
0.25 & 0.31 & 0.22 & 0.77 & 0.57 & 0.85
\end{array}$$

(XW)

$$= \begin{array}{l|rrrrrr}
\text{when} & -0.7555 & -1.3047 & 0.5073 & 0.1392 & 0.0182 & -0.8130 \\
\text{you} & -0.5575 & -1.4466 & 0.7066 & 0.1121 & 0.2078 & -1.0226 \\
\text{play} & -0.8078 & -1.3957 & 0.4355 & 0.2933 & 0.0841 & -0.7473 \\
\text{game} & -0.8378 & -1.2790 & 0.5661 & 0.1330 & -0.0421 & -0.7045 \\
\text{of} & -0.8173 & -1.3315 & 0.5331 & 0.1548 & 0.0214 & -0.7372 \\
\text{thrones} & -0.8552 & -1.2530 & 0.3287 & 0.2716 & -0.0276 & -0.6433
\end{array}$$

$$\begin{array}{l|rrrrrr}
\text{when} & -0.7555 & -1.3047 & 0.5073 & 0.1392 & 0.0182 & -0.8130 \\
\text{you} & -0.5575 & -1.4466 & 0.7066 & 0.1121 & 0.2078 & -1.0226 \\
\text{play} & -0.8078 & -1.3957 & 0.4355 & 0.2933 & 0.0841 & -0.7473 \\
\text{game} & -0.8378 & -1.2790 & 0.5661 & 0.1330 & -0.0421 & -0.7045 \\
\text{of} & -0.8173 & -1.3315 & 0.5331 & 0.1548 & 0.0214 & -0.7372 \\
\text{thrones} & -0.8552 & -1.2530 & 0.3287 & 0.2716 & -0.0276 & -0.6433
\end{array}$$

(XW) + (bias vector b)

$$\begin{array}{l|rrrrrr}
\text{bias} & 0.4200 & 0.1800 & 0.2500 & 0.4200 & 0.3500 & 0.4500
\end{array}$$

$(XW + b)$

$$= \begin{array}{l|rrrrrr}
\text{when} & -0.3355 & -1.1247 & 0.7573 & 0.5592 & 0.3682 & -0.3630 \\
\text{you} & -0.1375 & -1.2666 & 0.9566 & 0.5321 & 0.5578 & -0.5726 \\
\text{play} & -0.3878 & -1.2157 & 0.6855 & 0.7133 & 0.4341 & -0.2973 \\
\text{game} & -0.4178 & -1.0990 & 0.8161 & 0.5530 & 0.3079 & -0.2545 \\
\text{of} & -0.3973 & -1.1515 & 0.7831 & 0.5748 & 0.3714 & -0.2872 \\
\text{thrones} & -0.4352 & -1.0730 & 0.5787 & 0.6916 & 0.3224 & -0.1933
\end{array}$$

Step 7 — FFNN activation

$$\max\{0, XW + b\} = \begin{array}{l|cccccc}
\text{when} & 0.0000 & 0.0000 & 0.7573 & 0.5592 & 0.3682 & 0.0000 \\
\text{you} & 0.0000 & 0.0000 & 0.9566 & 0.5321 & 0.5578 & 0.0000 \\
\text{play} & 0.0000 & 0.0000 & 0.6855 & 0.7133 & 0.4341 & 0.0000 \\
\text{game} & 0.0000 & 0.0000 & 0.8161 & 0.5530 & 0.3079 & 0.0000 \\
\text{of} & 0.0000 & 0.0000 & 0.7831 & 0.5748 & 0.3714 & 0.0000 \\
\text{thrones} & 0.0000 & 0.0000 & 0.5787 & 0.6916 & 0.3224 & 0.0000
\end{array}$$

- after the FFNN there is an additional *add & norm* layer applied
- this completes the *first* encoder layer
- we won't calculate additional encoder layers here
- in practice there are N encoder layers, where the output of the *add & norm* layer (mentioned above) would serve as input for the second encoder layer

Decoder elements

- encoder input: when you play game of thrones
- decoder input: <start> you win or you die <end>
- most of the calculation in the decoder is the same
- three elements are new:
  8. decoder MHA
  9. masked MHA
  10. predicting tokens

Small detail: padding

- assume n_sequence = 10 as sequence length of the model
- our input sentence is of size 6
- we have to *pad* the encoder input tokens 7, 8, 9, and 10 with 0s

d	when	you	play	game	of	thrones	PAD	PAD	PAD	PAD
	5	17	7	12	15	19				
1	0.79	0.38	0.01	0.12	0.88	0.60	0	0	0	0
2	0.60	-0.37	1.92	1.73	1.25	1.02	0	0	0	0
3	0.96	-0.15	-0.14	0.05	0.04	-0.12	0	0	0	0
4	0.64	-0.19	0.40	-0.58	-0.03	1.01	0	0	0	0
5	0.97	0.85	0.47	-0.07	0.32	0.71	0	0	0	0
6	0.20	-0.26	0.59	-0.62	-0.28	0.24	0	0	0	0

Padding in MHA

$$Z = \text{softmax}\left(\frac{QK^T + \text{MASK}}{\sqrt{d_k}}\right)V$$

- MASK is just a padding matrix in the encoder
- here $-\infty$ is the padding token, since it results in 0s after softmax
- in our example:

$$\text{MASK} = \begin{matrix} 0 & 0 & 0 & 0 & 0 & 0 & -\infty & -\infty & -\infty & -\infty \\ 0 & 0 & 0 & 0 & 0 & 0 & -\infty & -\infty & -\infty & -\infty \\ 0 & 0 & 0 & 0 & 0 & 0 & -\infty & -\infty & -\infty & -\infty \\ 0 & 0 & 0 & 0 & 0 & 0 & -\infty & -\infty & -\infty & -\infty \\ 0 & 0 & 0 & 0 & 0 & 0 & -\infty & -\infty & -\infty & -\infty \\ 0 & 0 & 0 & 0 & 0 & 0 & -\infty & -\infty & -\infty & -\infty \\ -\infty & -\infty & -\infty & -\infty & -\infty & -\infty & -\infty & -\infty & -\infty & -\infty \\ -\infty & -\infty & -\infty & -\infty & -\infty & -\infty & -\infty & -\infty & -\infty & -\infty \\ -\infty & -\infty & -\infty & -\infty & -\infty & -\infty & -\infty & -\infty & -\infty & -\infty \\ -\infty & -\infty & -\infty & -\infty & -\infty & -\infty & -\infty & -\infty & -\infty & -\infty \end{matrix}$$

Step 8 — masked MHA

$$Z = \text{softmax}\left(\frac{QK^T + \text{MASK}}{\sqrt{d_k}}\right)V$$

- masking in in the decoder MHA works quite similar to padding
- we again use $-\infty$ as masking token
- we mask all future tokens, i.e.

$$\text{MASK} = \begin{matrix}
0 & -\infty & -\infty & -\infty & -\infty & -\infty & -\infty & -\infty & -\infty & -\infty \\
0 & 0 & -\infty & -\infty & -\infty & -\infty & -\infty & -\infty & -\infty & -\infty \\
0 & 0 & 0 & -\infty & -\infty & -\infty & -\infty & -\infty & -\infty & -\infty \\
0 & 0 & 0 & 0 & -\infty & -\infty & -\infty & -\infty & -\infty & -\infty \\
0 & 0 & 0 & 0 & 0 & -\infty & -\infty & -\infty & -\infty & -\infty \\
0 & 0 & 0 & 0 & 0 & 0 & -\infty & -\infty & -\infty & -\infty \\
-\infty & -\infty & -\infty & -\infty & -\infty & -\infty & -\infty & -\infty & -\infty & -\infty \\
-\infty & -\infty & -\infty & -\infty & -\infty & -\infty & -\infty & -\infty & -\infty & -\infty \\
-\infty & -\infty & -\infty & -\infty & -\infty & -\infty & -\infty & -\infty & -\infty & -\infty \\
-\infty & -\infty & -\infty & -\infty & -\infty & -\infty & -\infty & -\infty & -\infty & -\infty
\end{matrix}$$

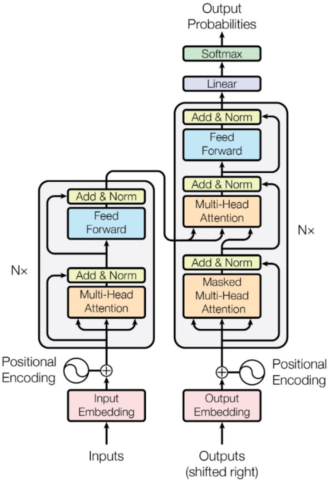
Input for the decoder

- <start> and <end>, as well as other special tokens (e.g., <sep>), are part of the vocabulary set
- the decoder input matrix looks like this:

d	<start>	you	win	or	you	die	<end>	PAD	PAD	PAD
	<start>	17	14	21	17	22	<end>			
1	0.51	0.38	0.91	0.12	0.38	0.60	0.11	0	0	0
2	0.83	-0.37	1.92	0.03	-0.37	1.22	0.12	0	0	0
3	0.22	-0.15	-0.14	0.05	-0.15	-0.12	-0.50	0	0	0
4	0.04	-0.55	0.20	-0.58	-0.55	-1.01	0.01	0	0	0
5	-0.11	0.85	0.77	-0.57	0.85	0.31	1.30	0	0	0
6	0.20	-1.80	0.59	-0.62	-1.80	0.24	0.012	0	0	0

Step 9 — decoder MHA

- $Z = \text{softmax}((QK^T + \text{MASK})/\sqrt{d_k})V$
  - Q is calculated from the output of the first *add & norm* layer from the decoder
    - $Q = X^D W^Q$
    - where X^D is the output from the *add & norm* layer
  - K, V are calculated using the output of the last encoder layer
    - $K = X^E W^K$
    - $V = X^E W^V$
    - where X^E is the output from the last encoder layer
- in the following *add & norm* layer, we calculate the normalization of $X^D + ZW^0$



(Vaswani et al., 2017)

Step 9 — decoder MHA MASK matrix

$$Z = \text{softmax}\left(\frac{QK^T + \text{MASK}}{\sqrt{d_k}}\right)V$$

- here we have a quite similar structure to encoder MHA
- Q comes from decoder with one more input token → one row less to pad with $-\infty$

$$\text{MASK} = \begin{matrix} 0 & 0 & 0 & 0 & 0 & 0 & -\infty & -\infty & -\infty & -\infty \\ 0 & 0 & 0 & 0 & 0 & 0 & -\infty & -\infty & -\infty & -\infty \\ 0 & 0 & 0 & 0 & 0 & 0 & -\infty & -\infty & -\infty & -\infty \\ 0 & 0 & 0 & 0 & 0 & 0 & -\infty & -\infty & -\infty & -\infty \\ 0 & 0 & 0 & 0 & 0 & 0 & -\infty & -\infty & -\infty & -\infty \\ 0 & 0 & 0 & 0 & 0 & 0 & -\infty & -\infty & -\infty & -\infty \\ 0 & 0 & 0 & 0 & 0 & 0 & -\infty & -\infty & -\infty & -\infty \\ -\infty & -\infty & -\infty & -\infty & -\infty & -\infty & -\infty & -\infty & -\infty & -\infty \\ -\infty & -\infty & -\infty & -\infty & -\infty & -\infty & -\infty & -\infty & -\infty & -\infty \\ -\infty & -\infty & -\infty & -\infty & -\infty & -\infty & -\infty & -\infty & -\infty & -\infty \end{matrix}$$

Step 10 — predicting tokens

- we learn a final linear layer that projects the output of the last *add & norm* layer to logits for each token position
- output of last *add & norm* layer: n_sequence$\times d$ (here: 10×6)
- output of final linear layer: n_sequence$\times V$ (here: $10 \times 23 +$ number of special tokens)
- we need a weight matrix W of size $d \times V$

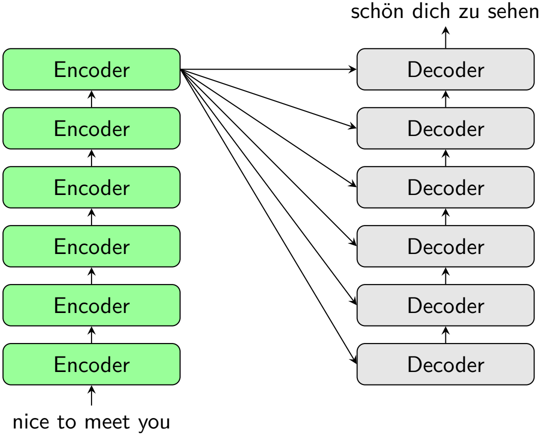
$$XW = L, \quad X \in \mathbb{R}^{\text{n\_seq} \times d}, W \in \mathbb{R}^{d \times V}, L \in \mathbb{R}^{\text{n\_seq} \times V}$$

- output of the final softmax layer is $\text{softmax}(L) \in [0, 1]^{\text{n\_seq} \times V}$
- with $\sum_{i=1}^{V} \text{softmax}(L)_{i,j} = 1 \quad \forall j = 1, \ldots, \text{n\_seq}$
- these can be seen as pseudo-probabilities
- to get the encoding index for the prediction of token j simply apply $\arg\max_{i=1,\ldots,V} L_{i,j}$

Further resources

see also `https://www.youtube.com/watch?v=EixI6t5oif0`

Breaking BERT[1] down[2,3]

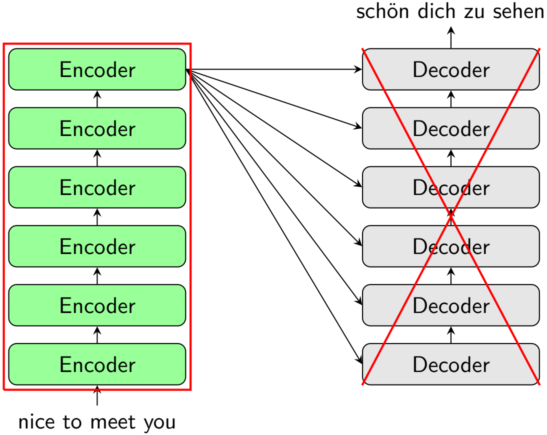


[1] Bidirectional Encoder Representations from Transformers

BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding (Devlin et al., NAACL 2019) http://dx.doi.org/10.18653/v1/N19-1423

[2] https://jalammar.github.io/illustrated-transformer/ [3] https://towardsdatascience.com/breaking-bert-down-430461f60efb

Breaking BERT[1] down[2,3]



[1] Bidirectional Encoder Representations from Transformers

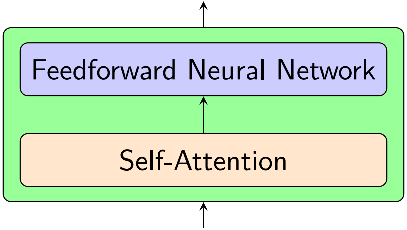
BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding (Devlin et al., NAACL 2019) http://dx.doi.org/10.18653/v1/N19-1423

[2] https://jalammar.github.io/illustrated-transformer/ [3] https://towardsdatascience.com/breaking-bert-down-430461f60efb

Encoder vs. decoder

- originally BERT is an **encoder only**
  - bidirectional approach contradicts traditional LM
  - predicting masked tokens (inside a sentence)
  - output: embeddings
  - extractive tasks (classification)
- there are **decoder only** models as well, e.g., GPT
  - unidirectional
  - predicting all tokens (to the right of a sentence)
  - output: (sequence of) words
  - language generation
- **encoder-decoder** models useful, e.g., for MT

Encoder layer in BERT



- every encoder layer contains of
  1. self-attention layer
  2. FF layer
- input of first encoder: list of embeddings (typically $d = 512$)
- size of list: hyperparameter ($\approx$ longest sequence in training data)
- input of all other encoders: output of previous encoder (same d)

BERT's innovation

- traditional LM: left-to-right
- ELMo: left-to-right and right-to-left

- BERT: bidirectional training
- using masked language modeling (see following slides)
- also uses a subword tokenizer (WordPiece), cf. fastText

- Google released two (pre-trained) versions:
  - pre-trained encoder
  - base: 12 layers, 768 output size, 12 MHA layers, 30 522 vocab
    number of parameters: 110 million
  - large: 24 layers, 1024 output size, 16 MHA layers, 30 522 vocab
    number of parameters: 340 million

How to train BERT

- pre-training was done on
  - BooksCorpus (800 million words)
  - English Wikipedia (2500 million words)
- took 4 days on 64 TPUs (tensor processing unit)
- TPUs are (even) faster in matrix operations than GPUs
- fine-tuning typically on single GPU

how to pre-train encoders?

Masked language modeling (MLM)

- BERT can see all the words in the sentences
- force BERT to learn embeddings without seeing the answer

- idea:
  1. replace a fraction of words in the input with a special [MASK] token
  2. predict these words

Masked language modeling (MLM)

- BERT can see all the words in the sentences
- force BERT to learn embeddings without seeing the answer

- idea:
  1. replace a fraction of words in the input with a special [MASK] token
  2. predict these words
- in BERT:
  - 15% of (sub)words are sampled to predict
    - 80%: replace with [MASK]
    - 10%: replace with random token
    - 10%: replace with self
  - why these values?
    - 100% [MASK] → only learning masked words' embeddings
    - 0% self → BERT would know:
      predict non-masked → always wrong word
    - 0% random: analogously to 0% self

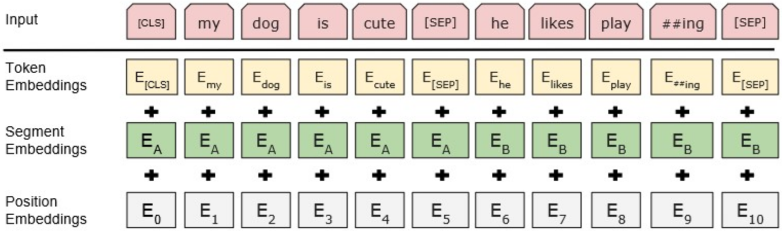
Next-sentence prediction

- pre-training includes a second objective
- given two sentences A and B
  - is B likely a sentence that follows A?
  - binary classification task
  - balanced, i.e., 50%/50% positive/negative examples
- idea: learn relationship between sentences
  - e.g., for question answering (QA) tasks

Next-sentence prediction

- pre-training includes a second objective
- given two sentences A and B
  - is B likely a sentence that follows A?
  - binary classification task
  - balanced, i.e., 50%/50% positive/negative examples
- idea: learn relationship between sentences
  - e.g., for question answering (QA) tasks
- technically (cf. next slide):
  - add [CLS] token at the beginning of first sentence
  - add [SEP] at the end of each sentence
  - add segment (sentence) embedding
  - predict [CLS] token
- turns out that next-sentence prediction is not that important for training

BERT input

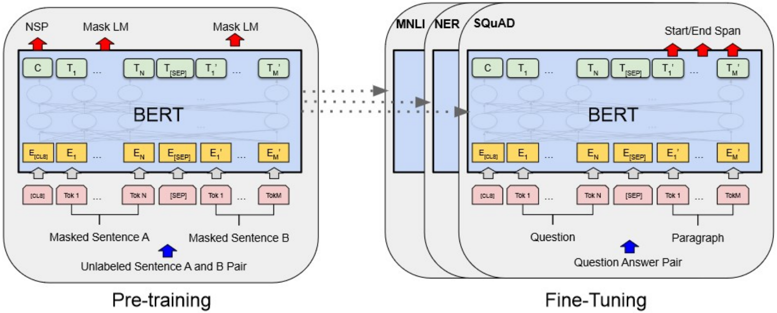


Input	[CLS]	my	dog	is	cute	[SEP]	he	likes	play	##ing	[SEP]
Token Embeddings	$E_{[CLS]}$	E_{my}	E_{dog}	E_{is}	E_{cute}	$E_{[SEP]}$	E_{he}	E_{likes}	E_{play}	$E_{\#\#ing}$	$E_{[SEP]}$
	+	+	+	+	+	+	+	+	+	+	+
Segment Embeddings	E_A	E_A	E_A	E_A	E_A	E_A	E_B	E_B	E_B	E_B	E_B
	+	+	+	+	+	+	+	+	+	+	+
Position Embeddings	E_0	E_1	E_2	E_3	E_4	E_5	E_6	E_7	E_8	E_9	E_{10}

(Devlin et al., 2019)

- [CLS] useful for subsequent classification tasks
- [sentence A, sentence B] may be, e.g., [question, answer] for QA fine-tuning (cf. next slide)

Fine-tuning (FT) BERT

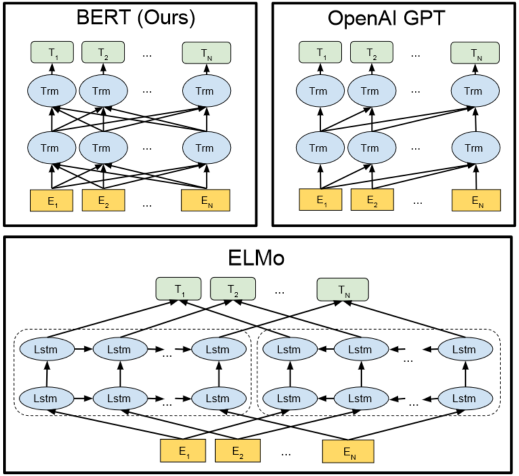


(Devlin et al., 2019)

- several different downstream NLP tasks, e.g., NER, sentence classification, QA
- output layer for token level tasks + [CLS] classification task
- fine-tuning all parameters

Differences in pre-training

- BERT: bidirectional
- GPT: left-to-right
- ELMo: left-to-right + right-to-left
- BERT & GPT: FT
- ELMo: feature-based



(Devlin et al., 2019)

Transformer-based models I

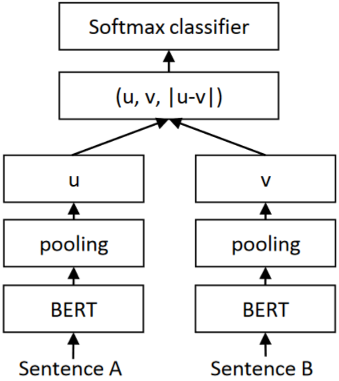
- a lot of modifications/improvements and similar architectures as BERT
- RoBERTa (English)
  - mainly just train BERT for longer and remove next sentence prediction
  - base: 12 layers, 768 output size, 50 265 vocab
  - large: 24 layers, 1024 output size, 50 265 vocab
- XLM-RoBERTa (multilingual)
  - trained on multilingual data
  - base: 12 layers, 768 output size, 250 002 vocab
  - large: 24 layers, 1024 output size, 250 002 vocab
- (m)DeBERTa (English, multilingual)
  - disentangled attention mechanism → 2 embedding vectors: content & position
  - base: 12 layers, 768 output size, 128 100 vocab (250 002 vocab)
  - large: 24 layers, 1024 output size, 128 100 vocab (250 002 vocab)

Transformer-based models II

- DistilBERT (English)
  - less parameters, compression technique, tries to mimic BERT (base)
  - 60% parameters of BERT base, 60% faster
  - preserving over 95% of performances (GLUE benchmark)
- SBERT
  - dedicated sentence embedding training objective
  - enhanced version of predicting the [CLS] token (see following slide)
- ALBERT, CamemBERT, ELECTRA, . . .

Sentence-BERT (SBERT)

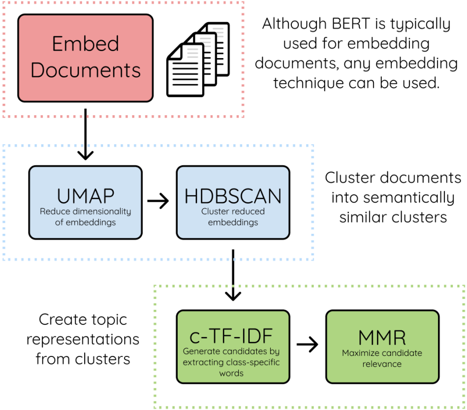
- aim: sentence embeddings
  - similar embeddings for similar sentences
- technique: siamese BERT-networks
- similarities between sentence-pairs
- classificatsberion objective: $\text{softmax}(W_t(u, v, |u - v|))$
- triplet objective: $\max(d(a, p) - d(a, n) + \epsilon, 0)$
  - a anchor sentence
  - p positive, n negative sentence
  - d euclidean distance, $\epsilon = 1$
- trained on STSb dataset
  https://huggingface.co/datasets/stsb_multi_mt



https://arxiv.org/abs/1908.10084

Further resource: https://www.pinecone.io/learn/sentence-embeddings/

How BERTopic works


Embed Documents
UMAP
Reduce dimensionality of embeddings
HDBSCAN
Cluster reduced embeddings
c-TF-IDF
Generate candidates by extracting class-specific words
MMR
Maximize candidate relevance

Although BERT is typically used for embedding documents, any embedding technique can be used.

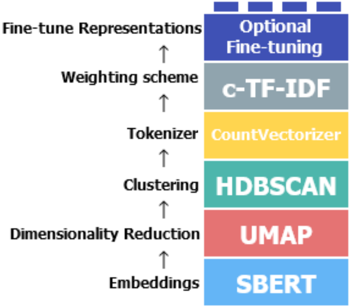
Cluster documents into semantically similar clusters

Create topic representations from clusters

https://github.com/MaartenGr/BERTopic/blob/master/docs/img/algorithm.png

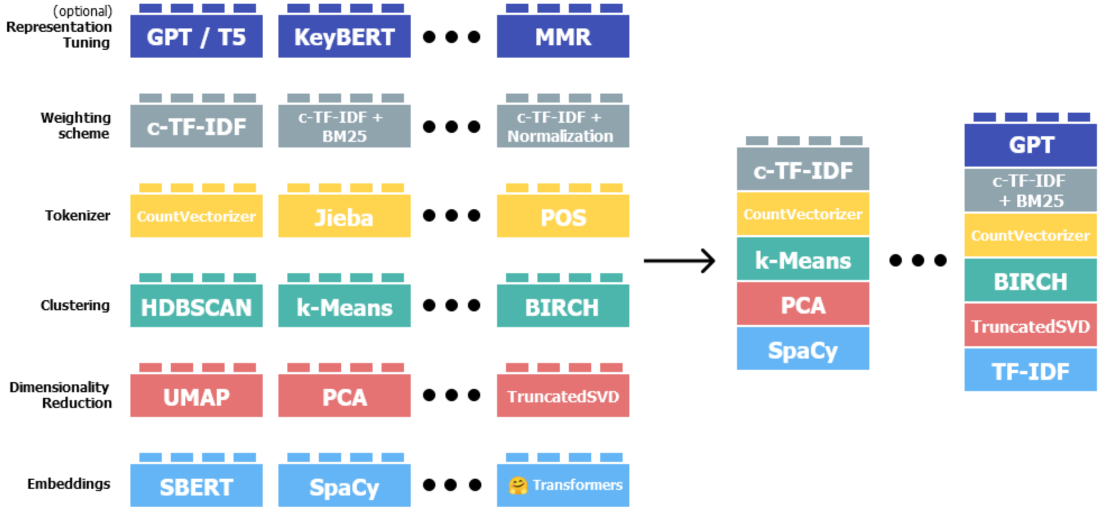
Default workflow of BERTopic

- SBERT as embedding technique
  - https://github.com/UKPLab/sentence-transformers
  - default: "all-MiniLM-L6-v2"
  - non-english: "paraphrase-multilingual-MiniLM-L12-v2"
- UMAP as dimension reduction technique
  - cf., PCA, truncated SVD
- HDBSCAN as clustering technique
  - cf., k-means
- CountVectorizer for tokenization
  - comparable to the presented approaches
- c-tf-idf as weighting technique
  - just a cluster-based tf-idf



https://maartengr.github.io/BERTopic/algorithm/algorithm.html

Modularity of BERTopic



https://maartengr.github.io/BERTopic/algorithm/algorithm.html

c-tf-idf

$$\text{c-tf-idf}_{w,c} = \frac{\text{tf}_{w,c}}{\sum_{v=1}^{V} \text{tf}_{v,c}} \log\left(1 + \frac{\frac{1}{C}\sum_{k=1}^{C}\sum_{v=1}^{V} \text{tf}_{v,k}}{\text{tf}_w}\right)$$

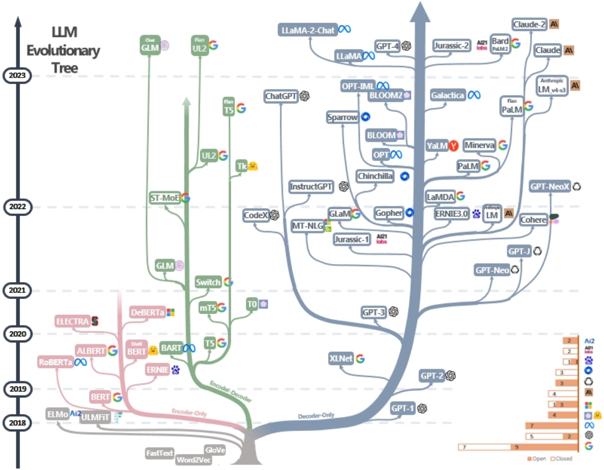
- instead of calculating idf based on documents, here based on clusters
- term-frequencies per cluster $\text{tf}_{w,c}$ are normalized
- nominator in logarithm: average number of words per class
- actually it should be named tf-icf: term-frequency – inverse-cluster-frequency
- it is also possible to use, e.g., the importance score from slide 164

Further resources on BERTopic

- `https://maartengr.github.io/BERTopic/`
- `https://youtu.be/uZxQz87lb84`
  - Maarten Grootendorst (author) explains BERTopic
  - YouTube Video (53 min.)
- `https://youtu.be/fb7LENb9eag`
  - BERTopic explained
  - YouTube Video (45 min.)
- `https://python.plainenglish.io/topic-modeling-for-beginners-using-bertopic-and-python-aaf1b421afeb`
  - Topic Modeling For Beginners Using BERTopic and Python
  - py/medium story (member-only)
- `https://towardsdatascience.com/let-us-extract-some-topics-from-text-data-part-iv-bertopic-46ddf3c91622`
  - Let us Extract some Topics from Text Data — Part IV: BERTopic
  - tds/medium story (member-only)
- `https://towardsdatascience.com/advanced-topic-modeling-with-bertopic-85fb8a90369e`
  - Advanced Topic Modeling with BERTopic
  - tds/medium story (member-only)

TBA

LLM evolutionary tree



https://github.com/Mooler0410/LLMsPracticalGuide

- it's not all ChatGPT
- but the LLM "market" is nowadays dominated by companies
- *Part of the problem is that no university in the world today can afford to develop its own model like ChatGPT. [...] Not even us.*[1]
- the only remedy: stop focusing on ever larger and more powerful (and resource-hungry) models

[1] translated: "Teil des Problems ist schon, dass es sich keine Universität der Welt heute leisten kann, ein eigenes Modell wie ChatGPT zu entwickeln. [...] Nicht einmal wir.", Prof. Fei-Fei Li (Stanford) in *Der Spiegel* Nr. 51, 16.12.2023.

Examples for pre-trained decoder language models

- GPT-3 (generative pretrained transformer)
  - successor of GPT and GPT-2 (12 layers, 768 output size, 50 257 vocab)
  - decoder only
  - 800GB storage for model parameters
- BLOOM (BigScience large open-science open-access multilingual language model)
  - decoder only
  - open access
  - developed as (free) alternative to GPT based models
- ChatGPT
  - at the moment (open) live study
  - no (formal) publication on it; 2023: but tech reports
  - open version an update of GPT-3 (GPT-3.5); 2023: payed access to GPT-4
- Gemini, PaLM, Claude, Falcon, LLaMA, Grok, OLMo, Llama 2, LLaMA 3.1, Mistral, Mixtral, T5, . . .

Research priorities for universities (in my opinion; not exhaustive)

What?	Why?	How?
efficiency	saving resources (time, energy, money)	parameter-efficient fine-tuning (PEFT), few-shot learning, effective learning (e.g., Adam), short-cutting via early-exit strategies
robustness	(out-of-domain/near-domain) application in real world scenarios	regularization (e.g., Adam, dropout layer), (transfer learning)
flexibility	efficient near-domain/out-of-domain application in real world scenarios	transfer learning, PEFT
reutilization	democratization of research for faster progress	transfer learning, modularity (e.g., PEFT)