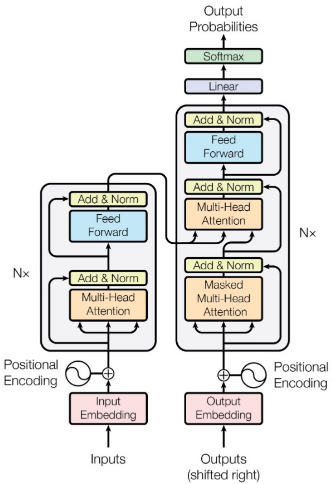
Step-by-step calculation of the transformer architecture[1] (encoder)

1. data, vocabulary, vocabulary size
2. encoding & static embedding
3. positional encoding
4. combining embedding and positional encoding
5. single-head attention (SHA) $\rightarrow$ 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 5	you 17	play 7	game 12	of 15	thrones 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

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

- let us encode the position of the token <u>of</u>, 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	PE(x | 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 <u>of</u> is given by the PE column

Step 3 — positional encoding II

- applying the same calculation to all input positions results in

d	when 5	you 17	play 7	game 12	of 15	thrones 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 5	you 17	play 7	game 12	of 15	thrones 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)

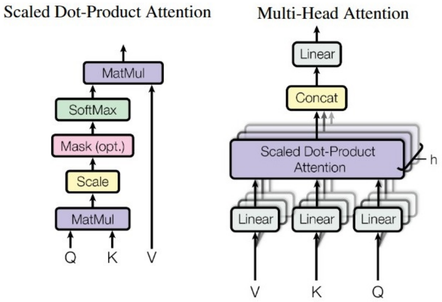
d	when 5	you 17	play 7	game 12	of 15	thrones 19
1	0.0000	0.8415	0.9093	0.1411	-0.7568	-0.9589
2	1.0000	0.5403	-0.4161	-0.9900	-0.6536	0.2837
3	0.0000	0.0464	0.0927	0.1388	0.1846	0.2300
4	1.0000	0.9989	0.9957	0.9903	0.9828	0.9732
5	0.0000	0.0022	0.0043	0.0065	0.0086	0.0108
6	1.0000	1.0000	1.0000	1.0000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

=

(input matrix for multi-head attention)

d	when 5	you 17	play 7	game 12	of 15	thrones 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)

when	0.79	1.60	0.96	1.64	0.97	1.20
you	1.22	0.17	0.06	0.79	0.90	0.74
play	0.92	1.51	0.27	1.31	0.56	1.59
game	0.26	0.74	0.66	0.22	0.07	0.37
of	0.12	0.59	0.80	0.62	0.50	0.70
thrones	-0.36	1.30	0.76	1.48	0.94	1.21

$\times$

(linear weights for query W^Q)

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

$=$

(query matrix Q)

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

(linear weights for key W^K)

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

$=$

(key matrix K)

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

(linear weights for value W^V)

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

$=$

(value matrix V)

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

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

Step 5.2 — query and key matrix multiplication & scaling

(query matrix Q)

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

$\times$

(transposed key matrix K^T)

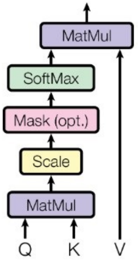
3.71	2.18	3.28	1.07	1.49	2.51
4.04	2.51	3.11	1.13	1.97	3.04
4.15	1.64	3.65	1.64	2.14	3.45
3.41	1.93	3.01	1.35	1.81	2.28

$=$

(QK^T)

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

Scaled Dot-Product Attention



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

$(QK^T/2)$

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

Step 5.3 — calculating softmax

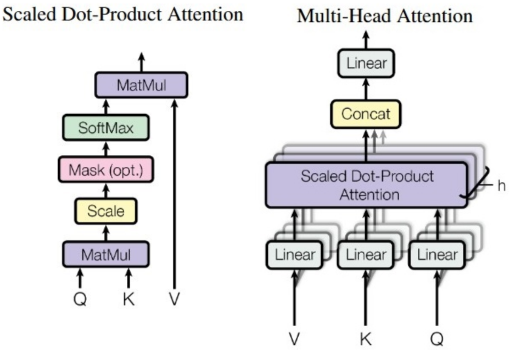
$$\text{softmax}(z_i) = \exp(z_i) / \sum_{j=1}^{d} \exp(z_j), \quad i = 1, \ldots, d$$

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

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

Step 5.3 — multiplication of softmax and value matrix

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



- 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

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

- 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

when	0.79	1.60	0.96	1.64	0.97	1.20
you	1.22	0.17	0.06	0.79	0.90	0.74
play	0.92	1.51	0.27	1.31	0.56	1.59
game	0.26	0.74	0.66	0.22	0.07	0.37
of	0.12	0.59	0.80	0.62	0.50	0.70
thrones	-0.36	1.30	0.76	1.48	0.94	1.21

(MHA input matrix) + (MHA output matrix)

when	12.33	10.87	8.07	8.71	7.12	9.16
you	12.18	10.73	7.97	8.60	7.02	9.05
play	12.32	10.85	8.06	8.70	7.10	9.14
game	11.69	10.28	7.63	8.25	6.73	8.71
of	12.03	10.59	7.86	8.49	6.93	8.95
thrones	12.27	10.81	8.03	8.67	7.08	9.11

=

(matrix to normalize)

when	13.12	12.47	9.03	10.35	8.09	10.36
you	13.40	10.90	8.03	9.39	7.92	9.79
play	13.24	12.36	8.33	10.01	7.66	10.73
game	11.95	11.02	8.29	8.47	6.80	9.08
of	12.15	11.18	8.66	9.11	7.43	9.65
thrones	11.91	12.11	8.79	10.15	8.02	10.32

- the input and output matrices are just added together

Step 6 — normalize

(matrix to normalize)

when	13.12	12.47	9.03	10.35	8.09	10.36
you	13.40	10.90	8.03	9.39	7.92	9.79
play	13.24	12.36	8.33	10.01	7.66	10.73
game	11.95	11.02	8.29	8.47	6.80	9.08
of	12.15	11.18	8.66	9.11	7.43	9.65
thrones	11.91	12.11	8.79	10.15	8.02	10.32

$\rightarrow$

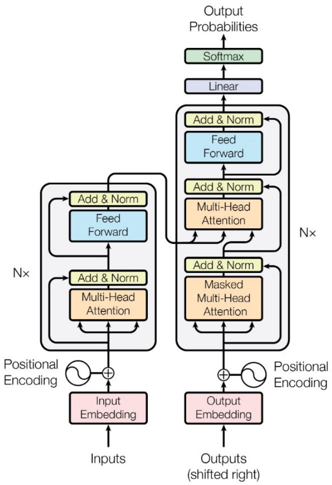
(matrix after normalization)

when	1.3176	0.9817	-0.7957	-0.1137	-1.2814	-0.1085
you	1.7078	0.4862	-0.9162	-0.2516	-0.9699	-0.0562
play	1.3026	0.9007	-0.9402	-0.1728	-1.2463	0.1561
game	1.4140	0.9236	-0.5159	-0.4209	-1.3015	-0.0993
of	1.4270	0.8628	-0.6030	-0.3412	-1.3184	-0.0271
thrones	1.0371	1.1596	-0.8738	-0.0408	-1.3454	0.0633

- 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)

when	1.3176	0.9817	-0.7957	-0.1137	-1.2814	-0.1085
you	1.7078	0.4862	-0.9162	-0.2516	-0.9699	-0.0562
play	1.3026	0.9007	-0.9402	-0.1728	-1.2463	0.1561
game	1.4140	0.9236	-0.5159	-0.4209	-1.3015	-0.0993
of	1.4270	0.8628	-0.6030	-0.3412	-1.3184	-0.0271
thrones	1.0371	1.1596	-0.8738	-0.0408	-1.3454	0.0633

$\times$

(weight matrix W)

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

$=$

(XW)

when	-0.7555	-1.3047	0.5073	0.1392	0.0182	-0.8130
you	-0.5575	-1.4466	0.7066	0.1121	0.2078	-1.0226
play	-0.8078	-1.3957	0.4355	0.2933	0.0841	-0.7473
game	-0.8378	-1.2790	0.5661	0.1330	-0.0421	-0.7045
of	-0.8173	-1.3315	0.5331	0.1548	0.0214	-0.7372
thrones	-0.8552	-1.2530	0.3287	0.2716	-0.0276	-0.6433

when	-0.7555	-1.3047	0.5073	0.1392	0.0182	-0.8130
you	-0.5575	-1.4466	0.7066	0.1121	0.2078	-1.0226
play	-0.8078	-1.3957	0.4355	0.2933	0.0841	-0.7473
game	-0.8378	-1.2790	0.5661	0.1330	-0.0421	-0.7045
of	-0.8173	-1.3315	0.5331	0.1548	0.0214	-0.7372
thrones	-0.8552	-1.2530	0.3287	0.2716	-0.0276	-0.6433

(XW) + (bias vector b)

bias	0.4200	0.1800	0.2500	0.4200	0.3500	0.4500

$=$

($XW + b$)

when	-0.3355	-1.1247	0.7573	0.5592	0.3682	-0.3630
you	-0.1375	-1.2666	0.9566	0.5321	0.5578	-0.5726
play	-0.3878	-1.2157	0.6855	0.7133	0.4341	-0.2973
game	-0.4178	-1.0990	0.8161	0.5530	0.3079	-0.2545
of	-0.3973	-1.1515	0.7831	0.5748	0.3714	-0.2872
thrones	-0.4352	-1.0730	0.5787	0.6916	0.3224	-0.1933

Step 7 — FFNN activation

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

- 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 5	you 17	play 7	game 12	of 15	thrones 19	PAD	PAD	PAD	PAD
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

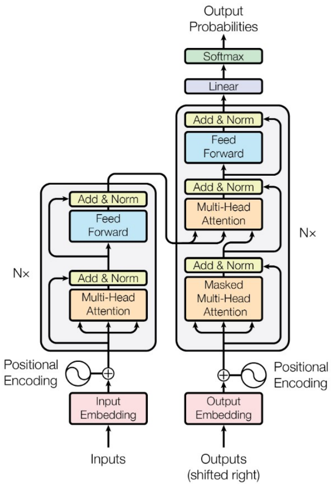
- <start> and <end>, as well as other special tokens (e.g., <sep>), are part of the vocabulary set
- the decoder input matrix looks like this:

d	<start> <start>	you 17	win 14	or 21	you 17	die 22	<end> <end>	PAD	PAD	PAD
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 $\rightarrow$ 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 $\arg\max_{i=1,\ldots,V} L_{i,j}$