

## Lecture 15:

# Language Models and Transformer



Markus Hohle

University California, Berkeley

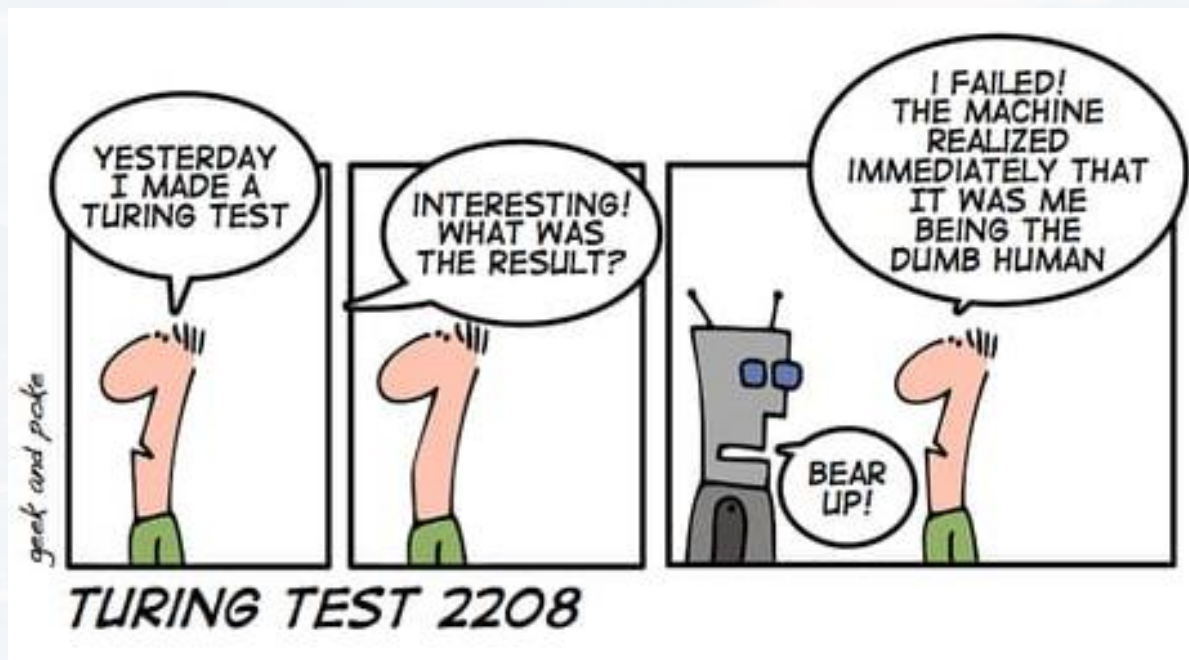
**Machine Learning Algorithms**

MSSE 277B, 3 Units

Fall 2024



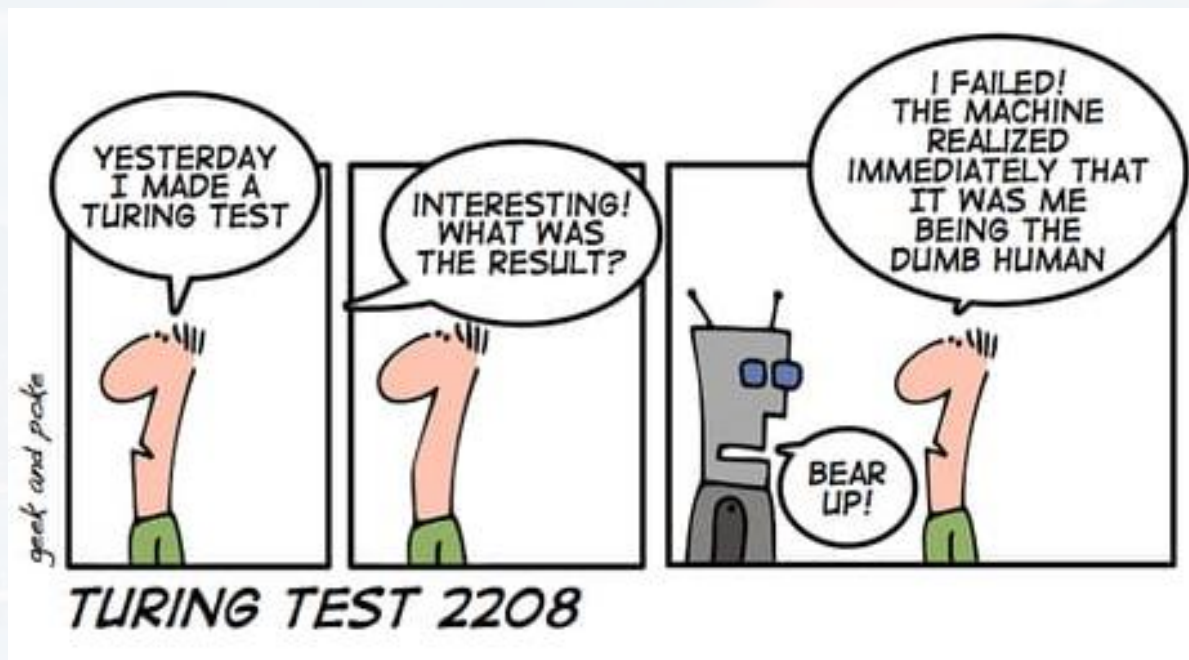
### Outline



- Introduction
- Bigram and MAP
- Positional Encoding
- Word Embedding
- Attention
- Transformer Architecture

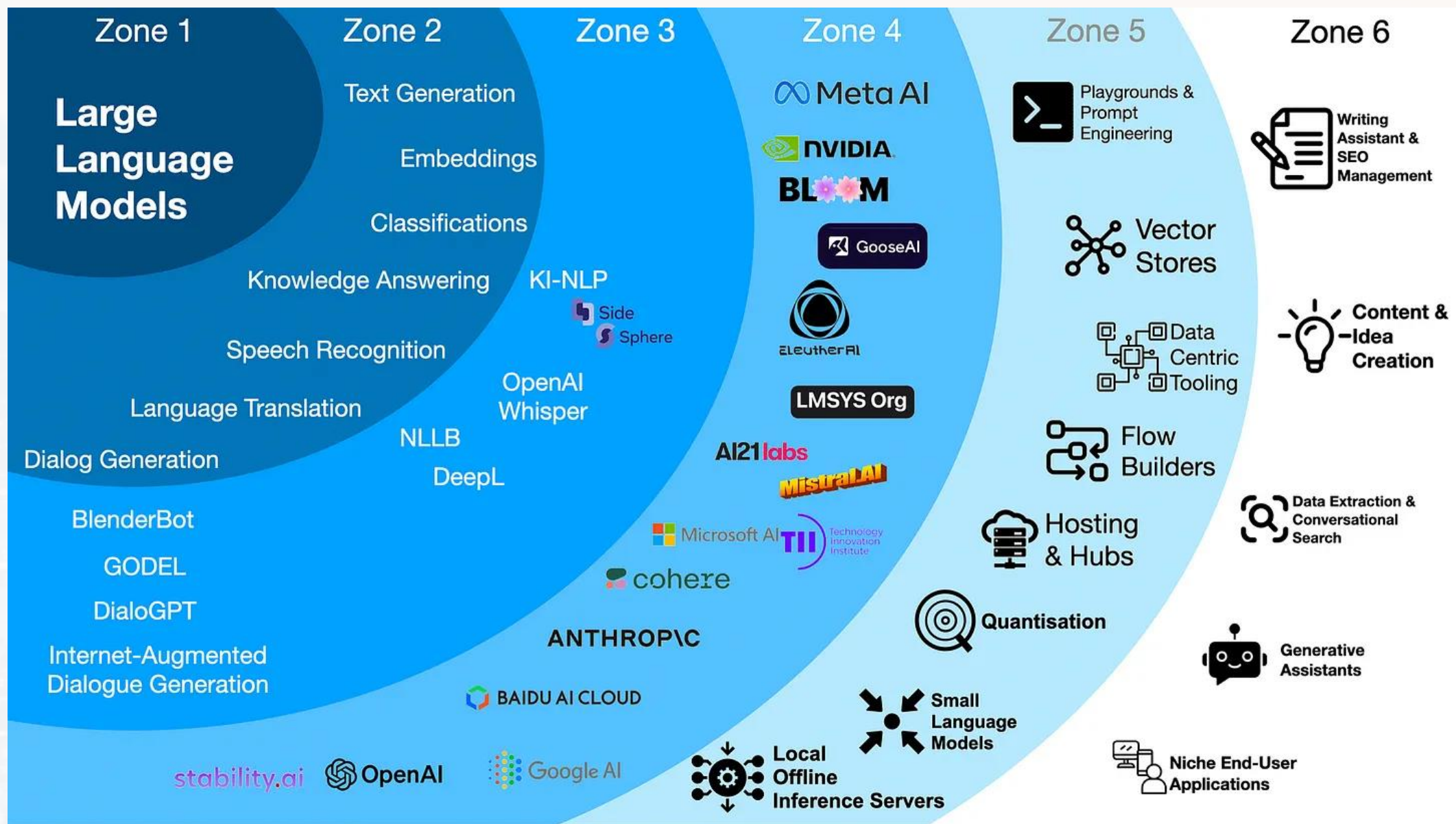


### Outline



- Introduction
- Bigram and MAP
- Positional Encoding
- Word Embedding
- Attention
- Transformer Architecture







**corpus:** (large, representative) data set containing sequences of a language

**token:** individual, independent entity of a language

**alphabet/vocabulary:** set of tokens

token	size of <i>alphabet</i>
- letters in a word	- $10^2$
- words in a sentence (upper/lower case, cases, gender, tenses, conjugations)	- $10^4 \dots 10^6$
- amino acids in a protein sequence	- 21
- nucleotides in a DNA/RNA sequence	- 4
- motifs in a DNA/RNA sequence	- $10^4$

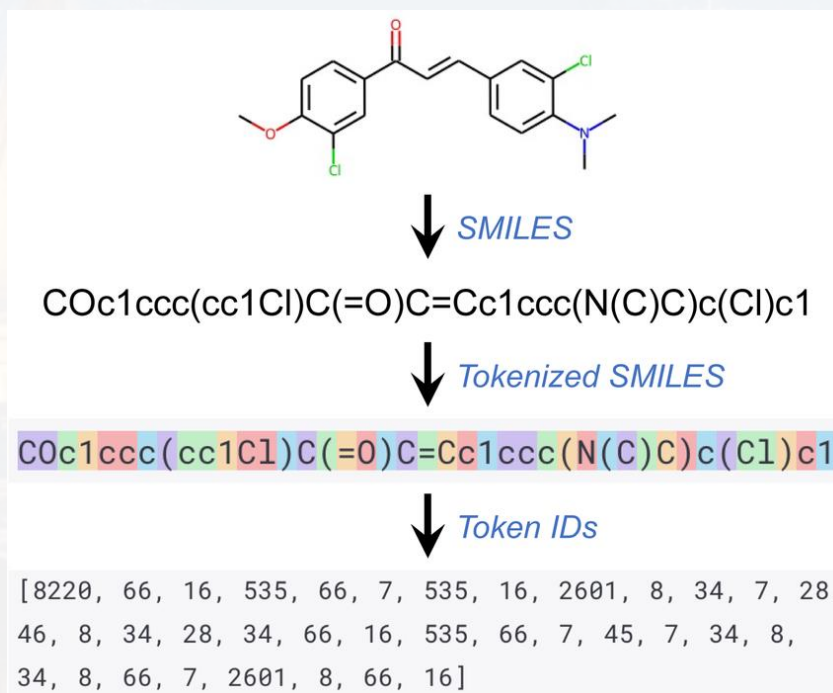


**corpus:** (large, representative) data set containing sequences of a language

**token:** individual, independent entity of a language

**alphabet/vocabulary:** set of tokens

### tokenization



**token:** - single atom vs...  
- ...functional group





**corpus:** (large, representative) data set containing sequences of a language

**token:** individual, independent entity of a language

**alphabet/vocabulary:** set of tokens

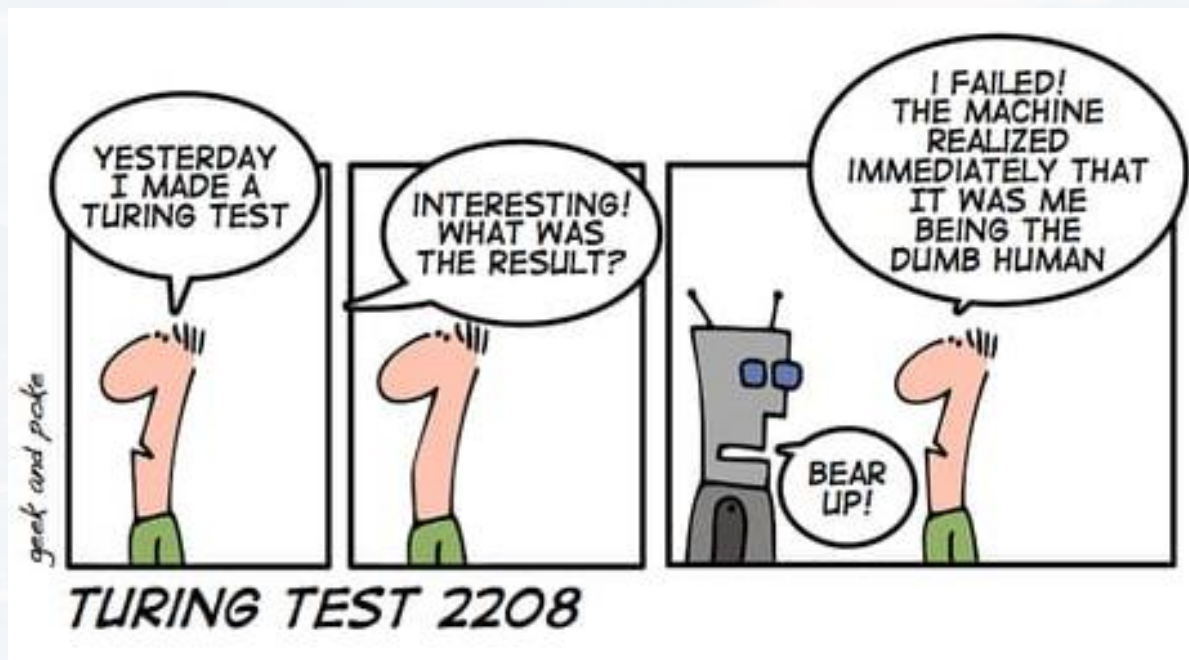
**note:** language models don't know grammar as we do, but they don't need to anyway...

**three things make context** (details: see later):

- **word embedding** (relation between similar/different token)
- **positional encoding** (location of token in a sequence)
- **attention** (relation between token within a sequence)



### Outline



- Introduction
- **Bigram and MAP**
- Positional Encoding
- Word Embedding
- Attention
- Transformer Architecture





$X_1 X_2 X_3 X_4 X_5 \dots X_n$

sequence of  $n$  token  $X$

actually:

$$P(X_1 X_2 X_3 X_4 X_5 \dots X_n) = P(X_n | X_{n-1} \dots X_1) P(X_{n-1} | X_{n-2} \dots X_1) \dots P(X_1)$$

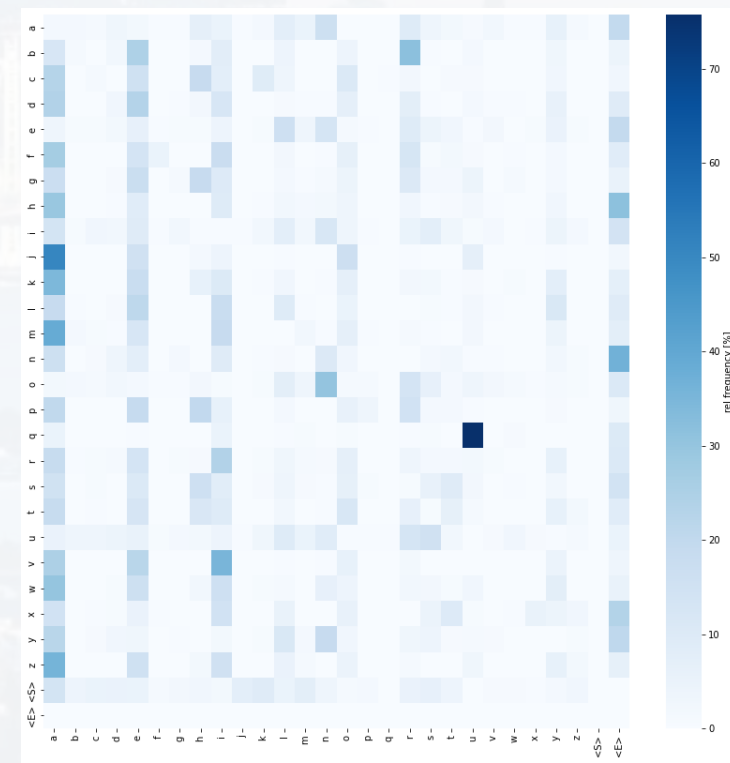
bigram (1<sup>st</sup> order Markov Chain, see e.g. first WhatsApp versions):

$$P(X_1 X_2 X_3 X_4 X_5 \dots X_n) = P(X_n | X_{n-1}) P(X_{n-1} | X_{n-2}) \dots P(X_1)$$

$P(i|j)$ : that token  $i$  is generated after token  $j$

→ N x N transition matrix from frequencies

→ "bigram" = "two words"



frequency matrix of letters in common names



bigram (1<sup>st</sup> order Markov Chain):

let's build our own bigram model: **generate new names** based on a corpus of names

```
In [15]: words[0:12]
Out[15]:
['emma',
 'olivia',
 'ava',
 'isabella',
 'sophia',
 'charlotte',
 'mia',
 'amelia',
 'harper',
 'evelyn',
 'abigail',
 'emily']
```

see **Andrej Karpathy's GitHub** repository

$$P(X_1 X_2 X_3 X_4 X_5 \dots X_n) = P(X_n|X_{n-1})P(X_{n-1}|X_{n-2}) \dots P(X_1)$$

we only need to count **how often** a letter is followed by another

we also need to indicate when a name has **started** and **ended**

[ '**<S>**' ] + [ '*olivia*' ] + [ '**<E>**' ]

→ **alphabet**: 26 letters + the two special “letters”

let's create a dictionary first (will help for counting):



### bigram (1<sup>st</sup> order Markov Chain):

let's build our own bigram model: **generate new names** based on a corpus of names

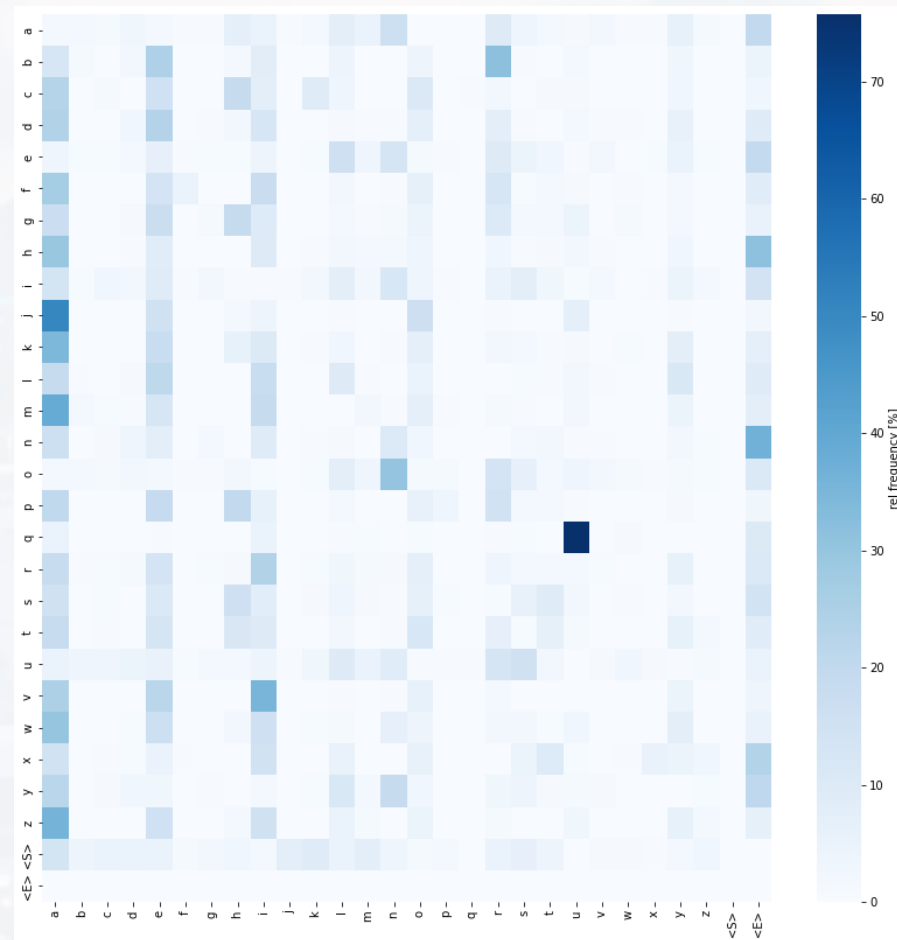
- 1) dictionary first (will help for counting)
- 2) count how often letter  $i$  is followed by letter  $j$

→ bigram matrix  $\mathbf{N}$

- 3) normalize  $\mathbf{N}$  accordingly
- 4) begin with a start token
- 5) draw a letter randomly based on  $\mathbf{N}$ , using

`np.argmax(np.random.multinomial(1,p))`

- 6) if next token is stop token → stop







bigram (1<sup>st</sup> order Markov Chain):

let's build our own bigram model: **generate new names** based on a corpus of names

check out **Bigram.ipynb**

**B.SampleNames(15)** vs totally random **B.SampleNames(15, False)**

some names  
are gibberish

some names  
sound real

some names  
are real

```
In [295]: B.SampleNames(15)
```

```
keesa  
ann  
ja  
jon  
nma  
malynojana  
sall  
daha  
drvah  
lzaxi  
tyunusthun  
jorrwro  
ja  
asoow  
s
```

```
In [296]: B.SampleNames(15, False)
```

```
mtkgy  
yufexhviovmorhqvikbbbjxebpxwurejaqlzzuwuanxmmomhr<S>uhb  
xlmusadjfdzxadaotd  
ik<S>vdtvdvxevtaselkykcfbamceprtvl  
zyr<S>inzoerobzwuovx  
eg<S>pbdvikf<S>tomcnkfsjay<S>rikatnaykizszcivpds<S>zj  
kh<S>y<S>ualzugqgakakeubjbasc  
bblupnibtqmyl<S>vyobf  
kybs  
rznjgpmlo  
tnhoxuckkjjzbwmj<S>vshkycicf<S>kowskphy  
rxodh  
jvswmzw  
jzpcfnpcg
```



Note, there is no conceptual difference between applying our model to *letters in a word* vs *words in a sentence*

caveats:

- the bigram model derives  $P(X_n)$  from **observed** frequencies  
 → essentially **MLE** (problematic if a letter hasn't appeared in the sequence yet  
 →  $P(X_n)$  assumed to be zero!)

```
Nsam = N/np.sum(N+0.0001, axis = 1, keepdims = True)
S_bi += np.sum(-N[:,i]*np.log(N[:,i]+1e-16))
```

- can we implement something that is closer to:

$$P(X_1 X_2 X_3 X_4 X_5 \dots X_n) = P(X_n | X_{n-1} \dots X_1) P(X_{n-1} | X_{n-2} \dots X_1) \dots P(X_1) \quad ?$$

binomial process

$$P(k|n, q) = \binom{n}{k} q^k (1 - q)^{n-k}$$

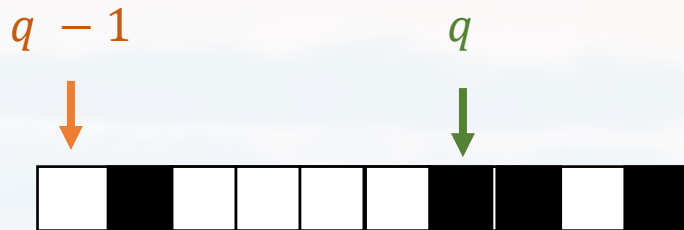


$q = ?$



$$P(X_1 X_2 X_3 X_4 X_5 \dots X_n) = P(X_n | X_{n-1} \dots X_1) P(X_{n-1} | X_{n-2} \dots X_1) \dots P(X_1)$$

Bayesian  
Parameter  
Estimation\*)



$$P(k|n, q) = \binom{n}{k} q^k (1 - q)^{n-k} \quad q = ?$$

likelihood function (here: binomial)

$$P(q|data\ set) = \frac{\overbrace{P(data\ set|q)}^{\text{likelihood function (here: binomial)}} \overbrace{P(q)}^{\text{prior } (\sim P(X_n|X_{n-1} \dots X_1))}}{\underbrace{P(data\ set)}_{\text{evidence (const wrt } q\text{)}}}$$

$$= \frac{1}{\int_0^1 P(q|data\ set) dq} (1 - q)^{n-k} q^k$$

$q = \text{const}$   
before 1<sup>st</sup> data point  
(max entropy!)

$$= \frac{q^{k+\alpha-1} (1-q)^{n-k+\beta-1}}{\int_0^1 q^{k+\alpha-1} (1-q)^{n-k+\beta-1} dq}$$

$q = \text{conjugate prior}$   
after  $n^{\text{th}}$  data point

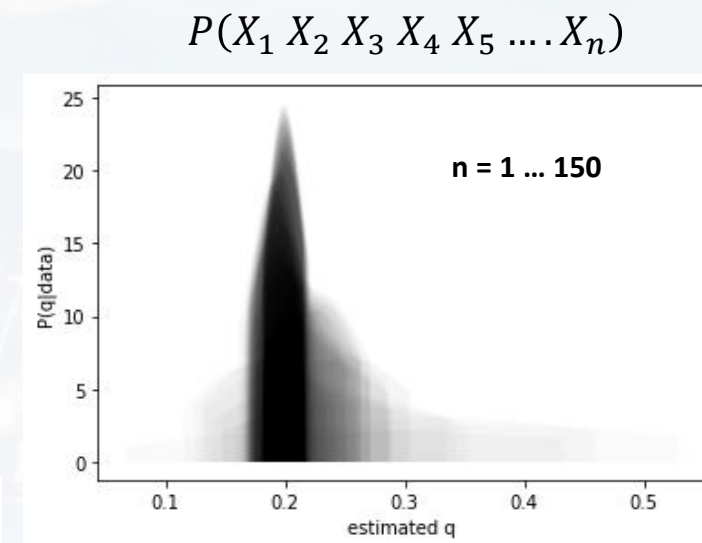
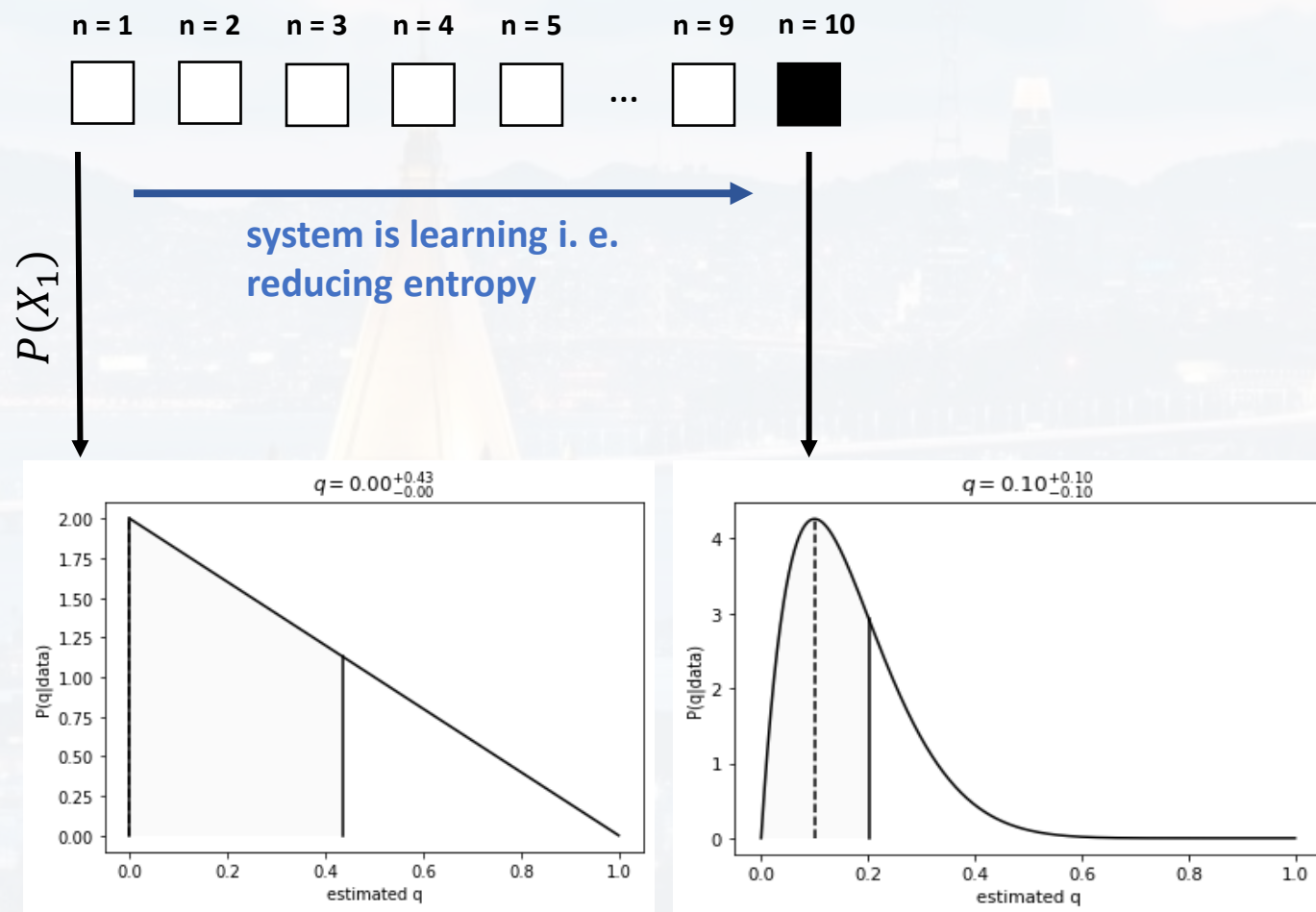




$$P(X_1 X_2 X_3 X_4 X_5 \dots X_n) = P(X_n | X_{n-1} \dots X_1) P(X_{n-1} | X_{n-2} \dots X_1) \dots P(X_1)$$

Bayesian  
Parameter  
Estimation\*)

$$P(k | n, q) = \binom{n}{k} q^k (1 - q)^{n-k} \quad q = ?$$



\*) see lecture 2



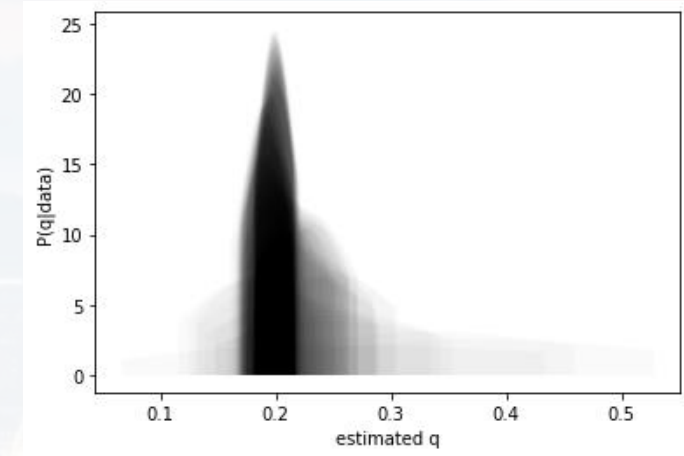
$$P(X_1 X_2 X_3 X_4 X_5 \dots X_n) = P(X_n | X_{n-1} \dots X_1) P(X_{n-1} | X_{n-2} \dots X_1) \dots P(X_1)$$

$$P(k | n, q) = \binom{n}{k} q^k (1 - q)^{n-k}$$

Bayesian  
Parameter  
Estimation\*)

$$P(q | \text{data set}) = \frac{q^{k+\alpha-1} (1-q)^{n-k+\beta-1}}{\int_0^1 q^{k+\alpha-1} (1-q)^{n-k+\beta-1} dq}$$

Beta function



more general, we want to learn the probability  $P_j(a)$  of letter  $a$  at position  $j$   $q \rightarrow P_j(a)$

→ multinomial problem

→ conjugate prior is the **Dirichlet distribution**

$$P(\text{sequence}) \sim \prod_j \prod_a P(a)_j^{\alpha(a)-1}$$

equivalent to what was  $P(q | \text{data set})$  earlier

\*) see lecture 2

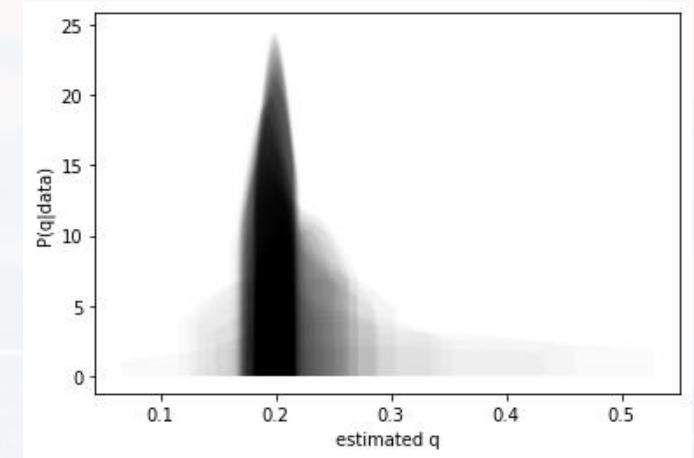


$$P(X_1 X_2 X_3 X_4 X_5 \dots X_n) = P(X_n | X_{n-1} \dots X_1) P(X_{n-1} | X_{n-2} \dots X_1) \dots P(X_1)$$

### Dirichlet distribution

$$P(sequence) \sim \prod_j \prod_a P(a)_j^{\alpha(a)-1}$$

note:  $\sum_{\text{over all } a} P(a)_j = 1 \quad \rightarrow \text{N-dim simplex}$



- note:
- we don't need to extract  $P(a)$  from the maximum of the pdf given by the BPE posterior
  - we can directly derive the maximum of  $P(a)$  from  $P(sequence)$  given the constrain  $\sum_{\text{over all } a} P(a)_j = 1$  (**Lagrangian multipliers**)
  - **Maximum a-posteriori (MAP)** approach  $\rightarrow$  see XXmotif (Siebert & Soeding, 2016)



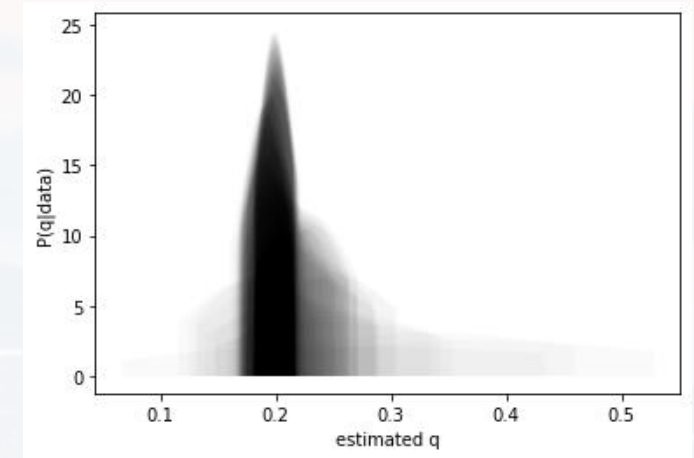


$$P(X_1 X_2 X_3 X_4 X_5 \dots X_n) = P(X_n | X_{n-1} \dots X_1) P(X_{n-1} | X_{n-2} \dots X_1) \dots P(X_1)$$

### Dirichlet distribution

$$P(\text{sequence}) \sim \prod_j \prod_a P(a)_j^{\alpha(a)-1}$$

note:  $\sum_{\text{over all } a} P(a)_j = 1 \quad \rightarrow \text{N-dim simplex}$

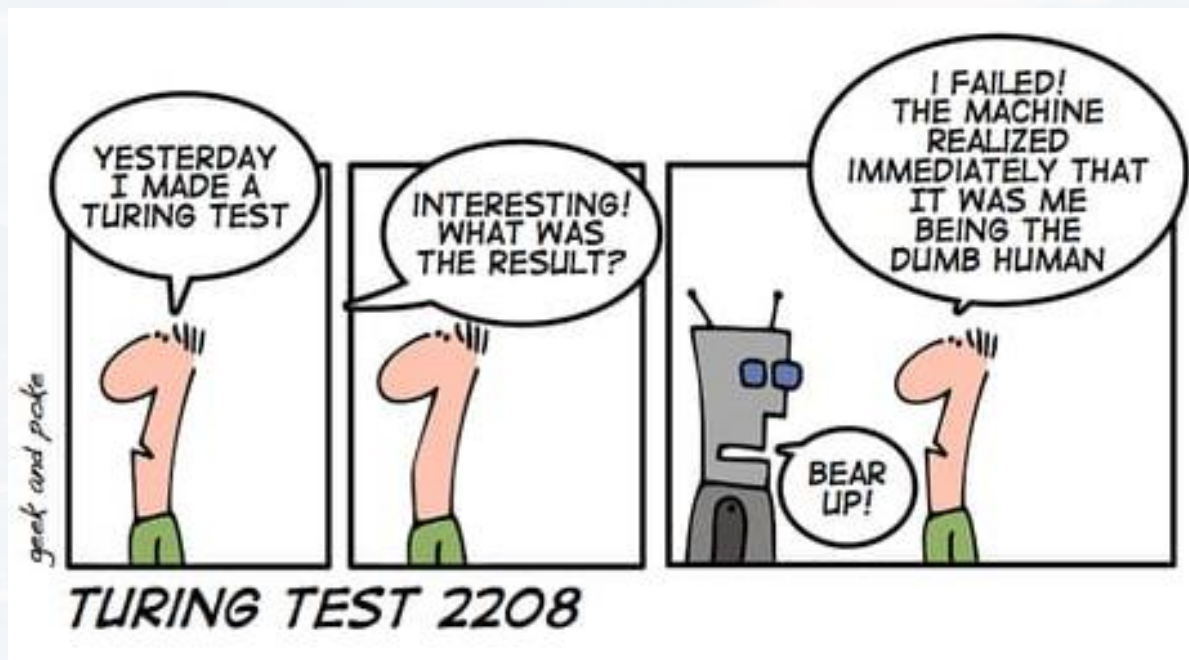


### Maximum a-posteriori (MAP)

- XXmotif (Siebert & Soeding, 2016) significantly outperformed PWMs
- it struggled however with related motifs which were **physically located far apart** from each other
- solution see later: attention
- older solutions: LSTMs



### Outline



- Introduction
- Bigram and MAP
- **Positional Encoding**
- Word Embedding
- Attention
- Transformer Architecture



three things make context:

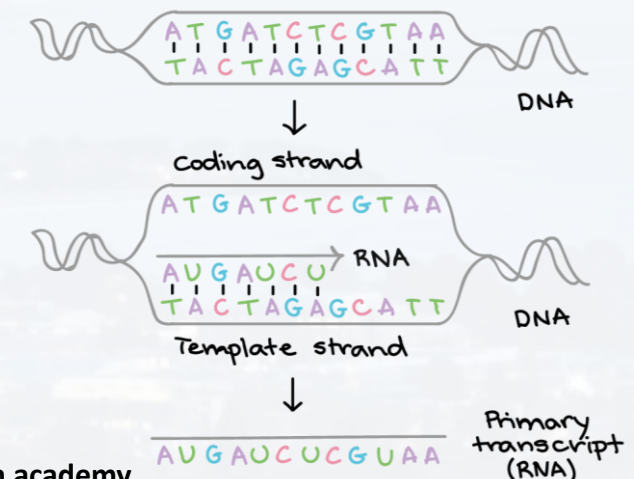
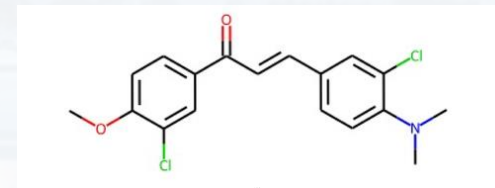
- **positional encoding** (location of token in a sequence)
- **word embedding** (relation between similar/different token)
- **attention** (relation between token within a sequence)

*“The cat jumped on the roof.”*

order matters!:

- 1<sup>st</sup>: article
- 2<sup>nd</sup>: noun/subject
- 3<sup>rd</sup>: verb
- 4<sup>th</sup>: noun/object (in English)

→ **positional encoding**







**goal:** find a positional encoding that is

- reasonably simple
- independent from the length of the sequence
- somehow normalized

one idea: n-bit binary encoding

position code

76543210 ← 8bit i.e. eight dimensions

1	0000000 <b>1</b>
2	000000 <b>1</b> 0
3	000000 <b>1</b> 1
4	00000 <b>1</b> 00
5	00000 <b>1</b> 0 <b>1</b>
6	00000 <b>1</b> 10
7	00000 <b>1</b> 11
8	0000 <b>1</b> 000
9	0000 <b>1</b> 00 <b>1</b>
10	0000 <b>1</b> 0 <b>1</b> 0
11	0000 <b>1</b> 0 <b>1</b> 1
12	0000 <b>1</b> 100
13	0000 <b>1</b> 10 <b>1</b>
14	0000 <b>1</b> 110
15	0000 <b>1</b> 111
16	000 <b>1</b> 0000

Does that look familiar?  
→ *like* Fourier Series

depending on dimensions (bit)

→ different frequencies

bit	frequency
0	$1/2$
1	$1/4$
2	$1/8$
...	



even dimensions:  $E(p, 2k) = \sin\left(\frac{p}{10,000^{\frac{2k}{d}}}\right)$

odd dimensions:  $E(p, 2k + 1) = \cos\left(\frac{p}{10,000^{\frac{2k}{d}}}\right)$

$p$ : position in sequence

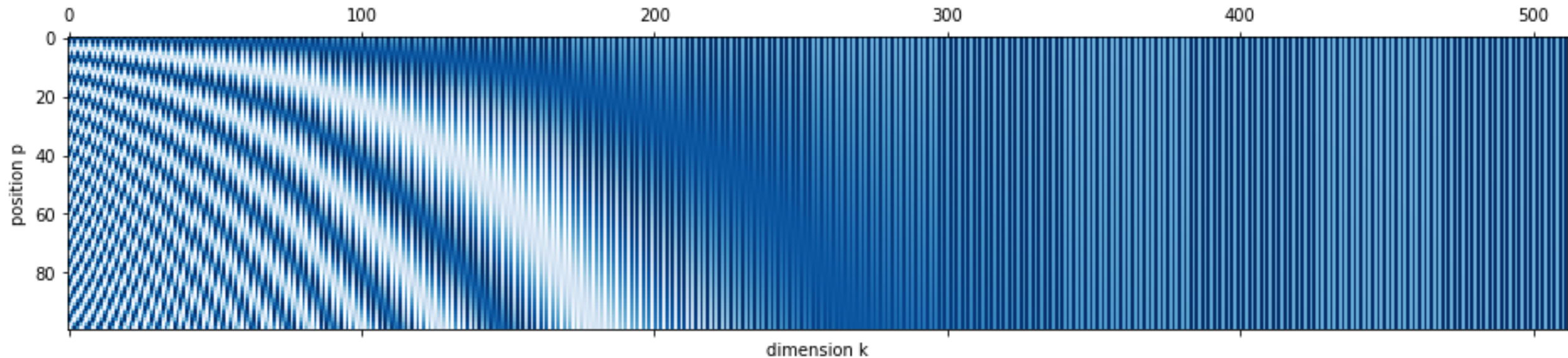
$k$ : dimension index

$d$ : number of dimensions

10,000: an arbitrary number Vaswani et al., 2017

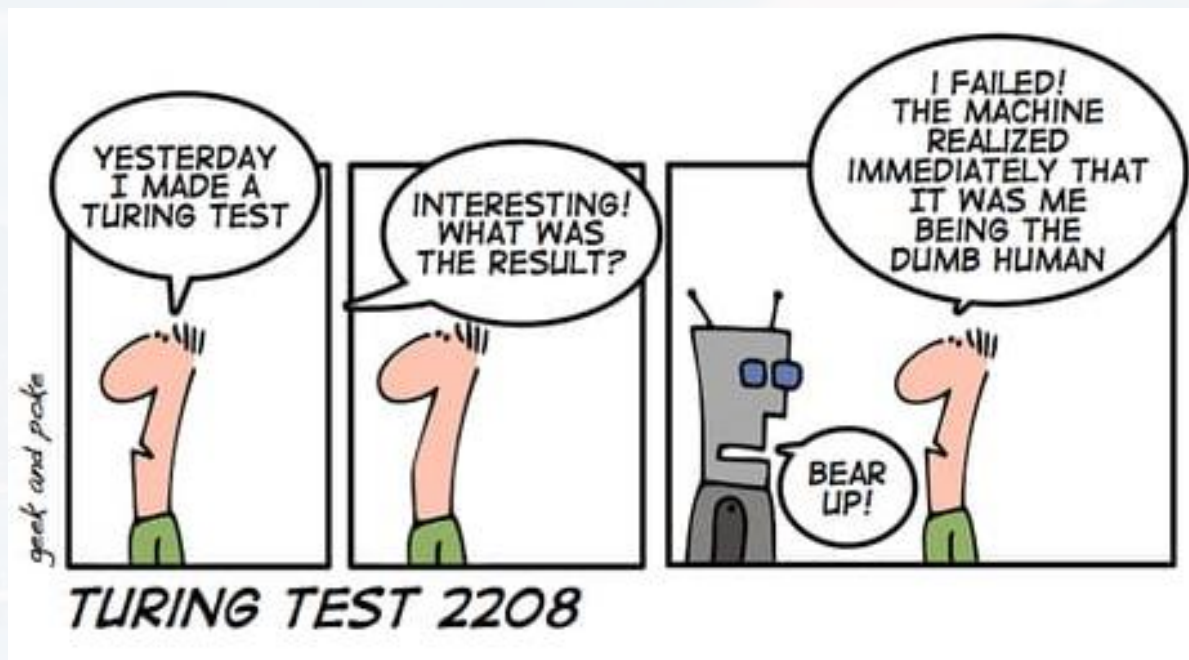
run **PlotPositionEncoding.py**

more info [here](#)





### Outline



- Introduction
- Bigram and MAP
- Positional Encoding
- **Word Embedding**
- Attention
- Transformer Architecture





three things make context:

- **positional encoding** (location of token in a sequence)
- **word embedding** (relation between similar/different token)
- **attention** (relation between token within a sequence)

problem: turning token (words/letters) into numbers

single letters:

ACGT

- one – hot works perfectly (four different token)

abcd...

- lower/upper case, special characters (50 different token), one – hot is fine too

words:

actual words

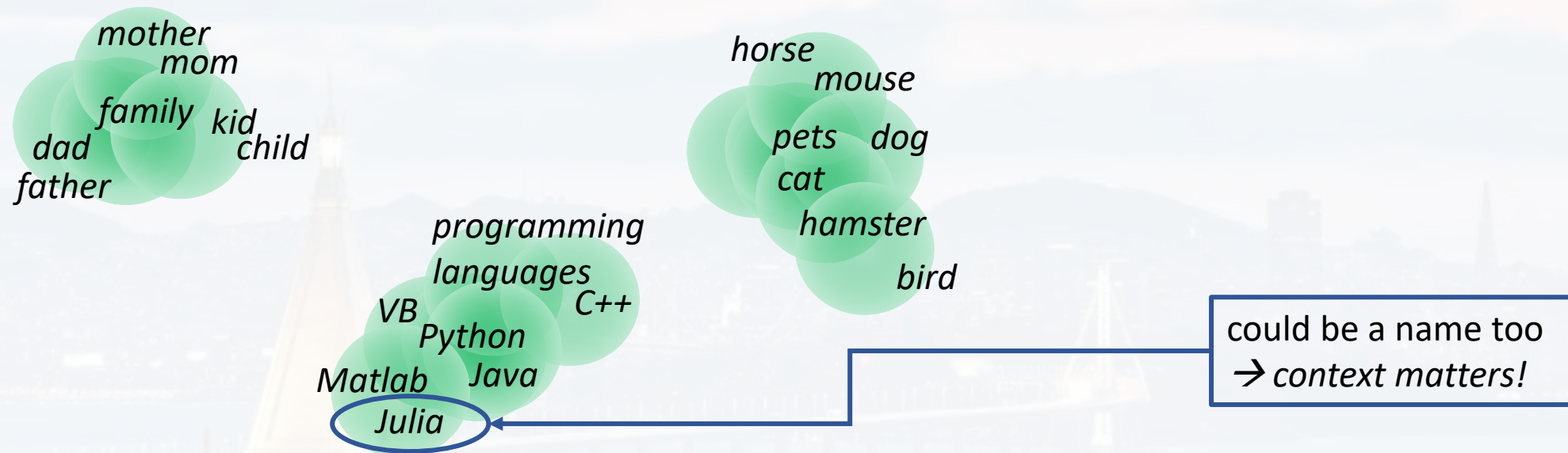
-  $10^4 \dots 10^6$  (upper/lower case, cases, gender, tenses, conjugations)

→ one – hot doesn't work (matrices would be too large)

→ some words have a **similar meaning**, should be **close** in parameter space (**cluster**)



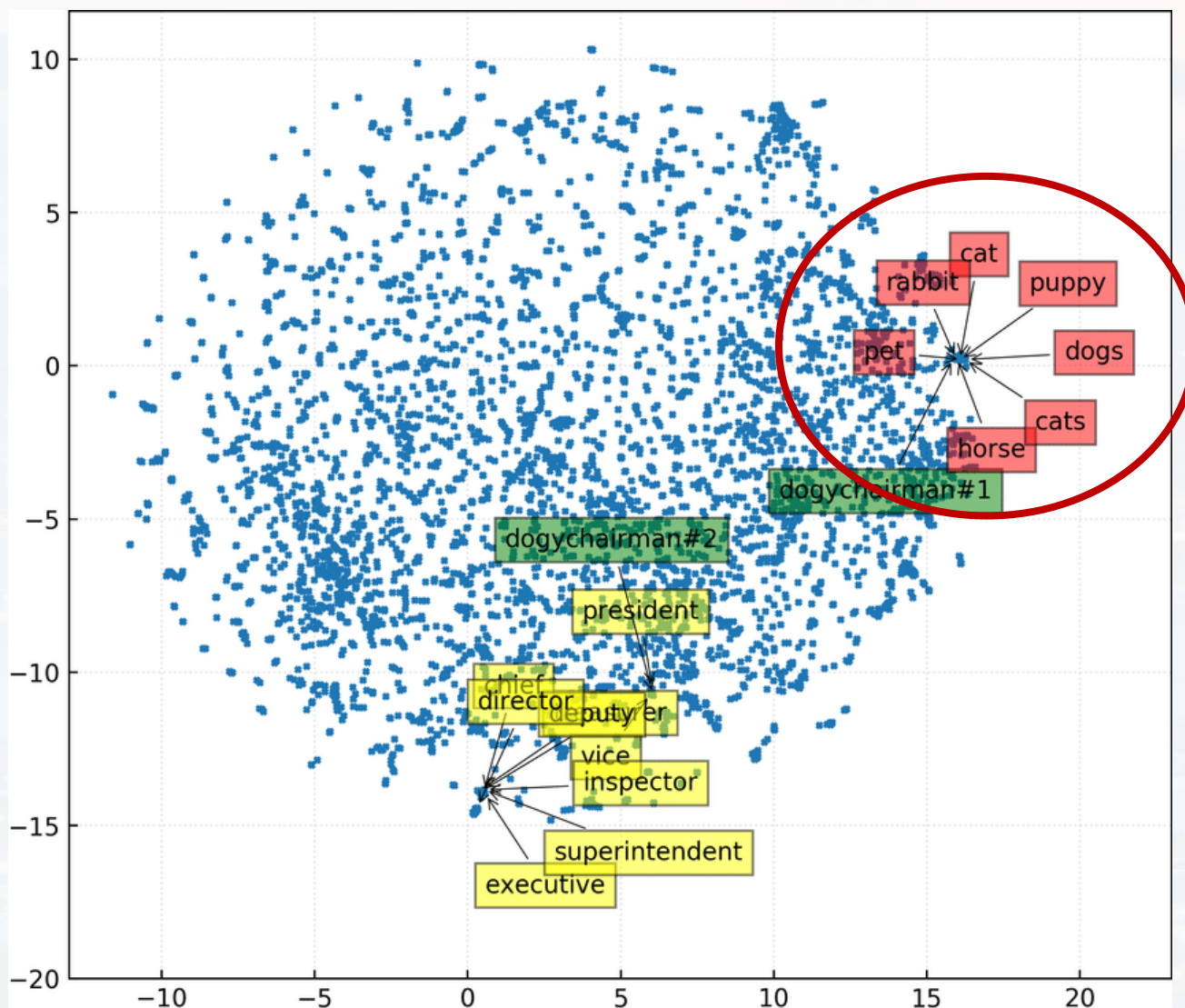
words with a **similar meaning** should form **cluster**



- embedding, instead of one – hot encoding
- from experience: **N = 30 – 300** dim vector for each token (**which is a lot less than  $10^4$  ...  $10^6$** ) is sufficient
- as a result: token with **similar meaning are close** in the vector space!



words with a **similar meaning** should form **cluster**



*"Joint Learning of Sense and Word Embeddings"*  
M Alsuhaibani & D Bollegala





common training set: recorded speeches from the European Parliament:

*...It seems absolutely disgraceful that we pass legislation and do not adhere to it ourselves. Mrs Lynne, you are quite right and I shall check whether this has actually not been done. I shall also refer the matter to the College of Quaestors, and I am certain that they will be keen to ensure that we comply with the regulations we ourselves vote on.*

*Madam President, Mrs Díez González and I had tabled questions on certain opinions of the Vice-President, Mrs de Palacio, which appeared in a Spanish newspaper.*

*The competent services have not included them in the agenda on the grounds that they had been answered in a previous part-session.*

*I would ask that they reconsider, since this is not the case....*

words of similar meaning should appear in similar environment

→ target token within a window

Two common algorithms are **C**ontinuous **B**ag **O**f **W**ords and **s**kip **g**ram



*Continuous **B**ag **O**f **W**ords*

$n$ : number of unique token from corpus  
 $m$ : desired number of dimensions for embedding

The competent services **have** not included them in the agenda on the grounds that they...

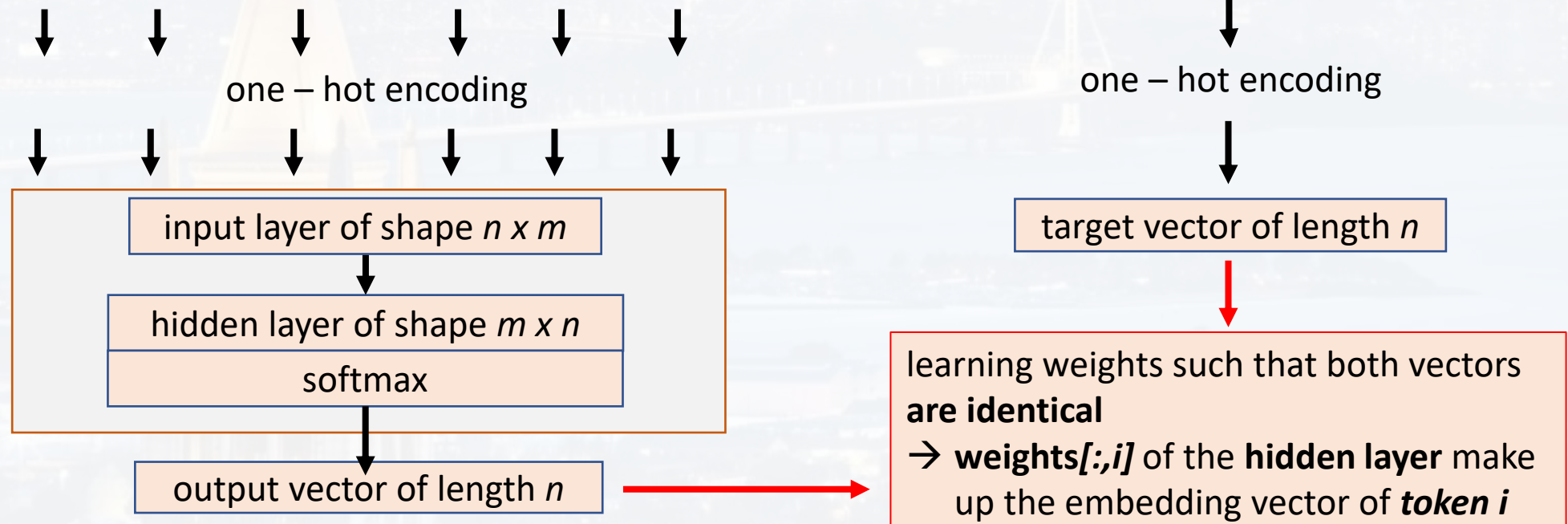
target

have

context window

The competent services not included them

shallow ANN





*Continuous **B**ag **O**f **W**ords*

$n$ : number of unique token from corpus  
 $m$ : desired number of dimensions for embedding

The competent services have not included them in the agenda on the grounds that they...

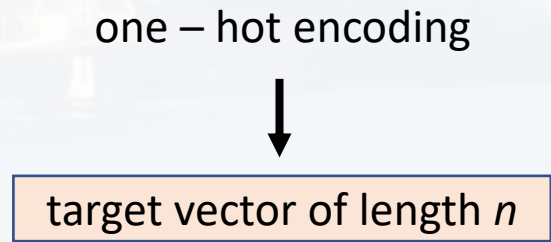
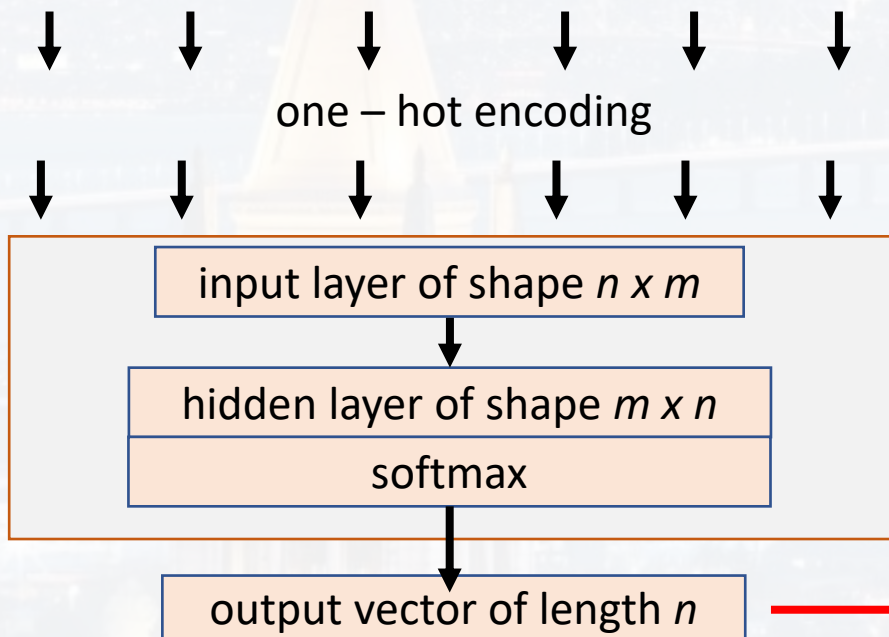
target

not

context window

competent services have included them in

shallow ANN



learning weights such that both vectors are identical  
→  $\text{weights[:,i]}$  of the hidden layer make up the embedding vector of *token i*





### Continuous **B**ag **O**f **W**ords

$n$ : number of unique token from corpus  
 $m$ : desired number of dimensions for embedding

The competent **services have not** **included** **them in the** agenda on the grounds that they...

target

**included**

context window

**services have not** **them in the**

one – hot encoding

one – hot encoding

shallow ANN

input layer of shape  $n \times m$

hidden layer of shape  $m \times n$

softmax

output vector of length  $n$

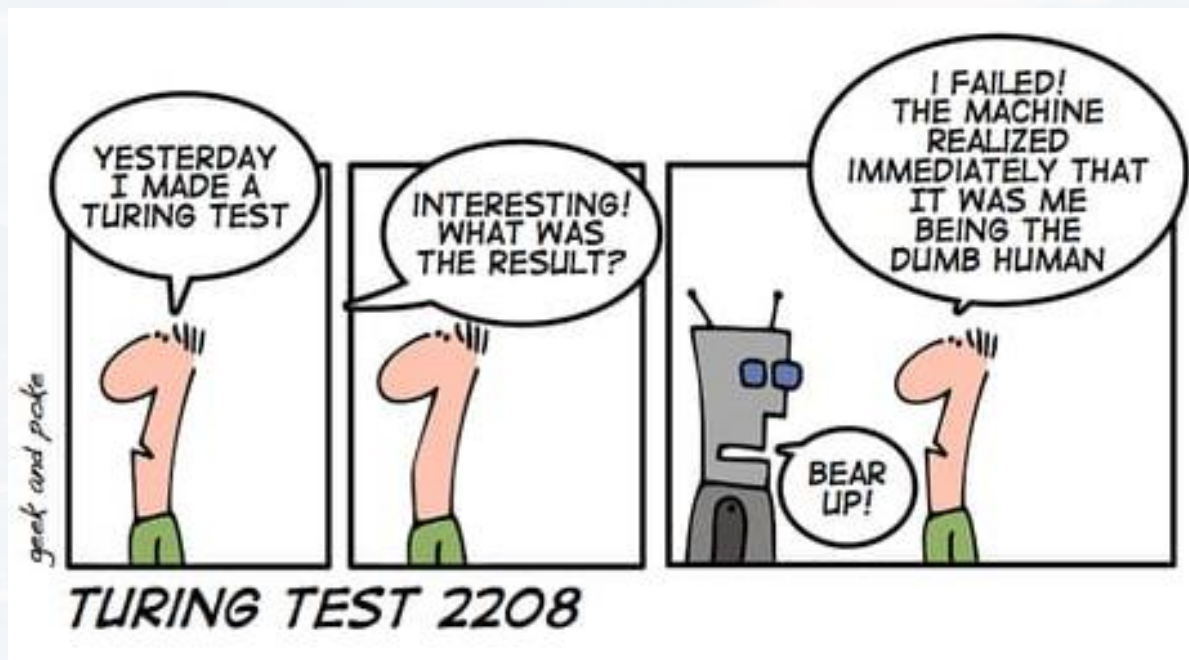
target vector of length  $n$

learning weights such that both vectors  
**are identical**  
→ **weights[:, $i$ ]** of the **hidden layer** make  
up the embedding vector of **token  $i$**

... and so on....



### Outline



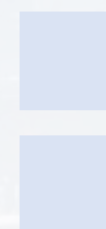
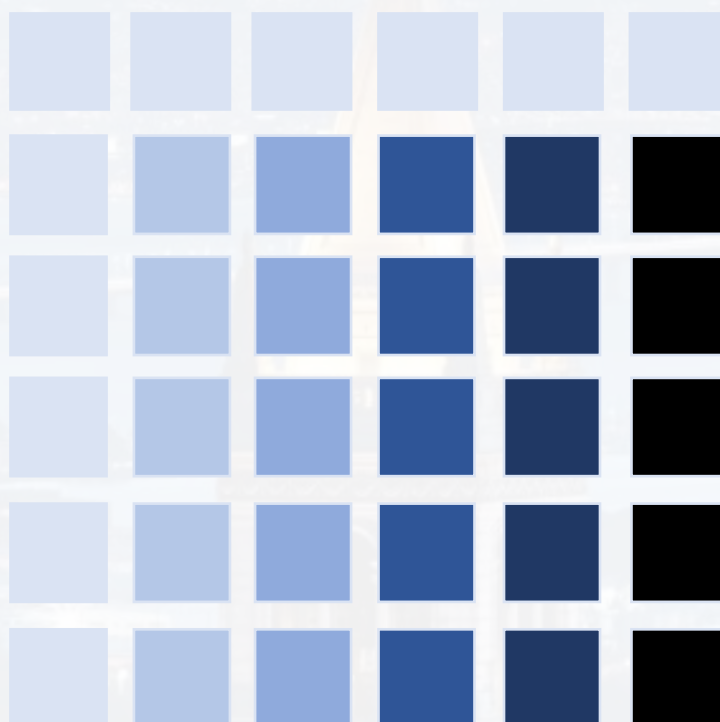
- Introduction
- Bigram and MAP
- Positional Encoding
- Word Embedding
- **Attention**
- Transformer Architecture



three things make context:

- **positional encoding** (location of token in a sequence)
- **word embedding** (relation between similar/different token)
- **attention** (relation between token within a sequence)

*“The cat jumped on the roof.”*

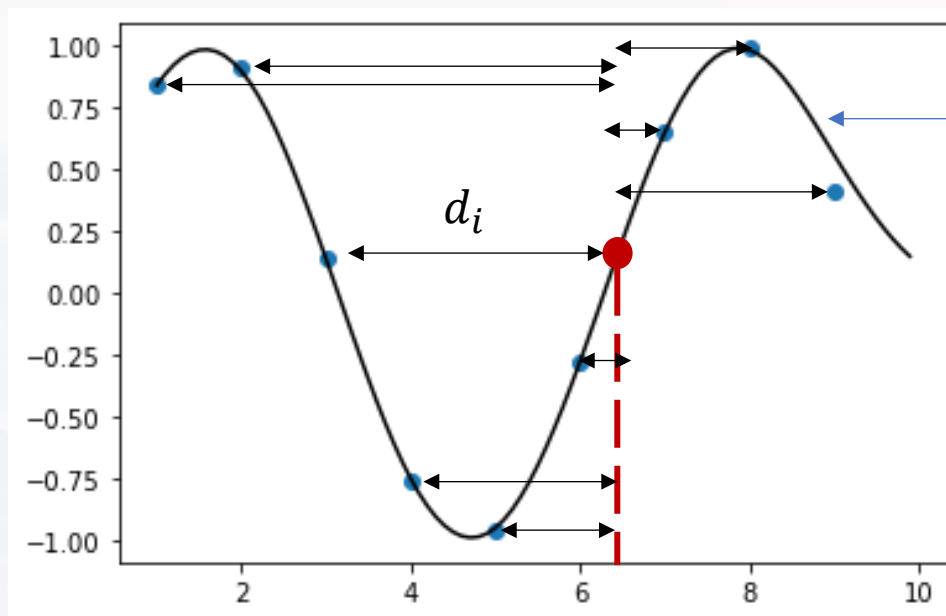


how the first token influences all other token

how the second token influences all other token

.... and so on





We want to interpolate between the blue dots  
→ generating the black line  
→ **no curve fitting!**

- idea:
- select a point for which we want the interpolation for
  - calculate distance  $d_i$  to every other point
  - each data point should influence the value of the interpolated point
  - the closer, the stronger the influence → weighted mean

$$y_{int} \sim \sum_{i=1}^I w_i y_i$$

$$w_i \sim \frac{1}{d_i}$$



calculating distance



```
D = np.tile(V, (1, len(V))) - np.tile(L.transpose(), (len(V), 1))
```

- each data point should influence the value of the interpolated point
- the closer, the stronger the influence → weighted mean

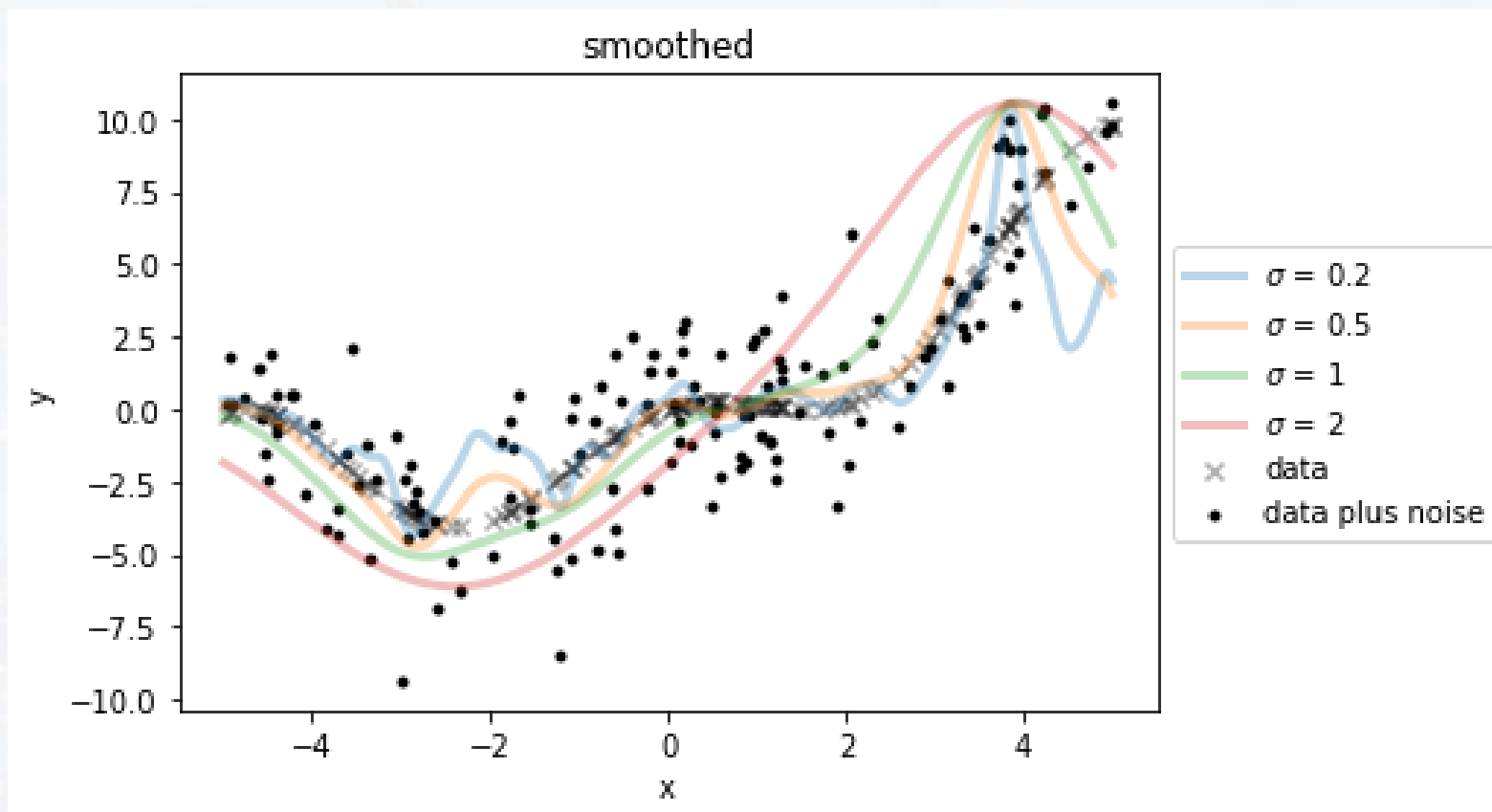
$$y_{int} \sim \sum_{i=1}^I w_i y_i$$

```
Gaussian kernel    W      = np.exp(-(D**2)/(sigma))
                   W      = W/np.sum(W)
                   yint   = np.dot(W.transpose(), y)
```



```
D = np.tile(V, (1, len(V))) - np.tile(L.transpose(), (len(V), 1))
```

```
Gaussian kernel    W      = np.exp(-(D**2)/(sigma))  
                  W      = W/np.sum(W)  
                  yint   = np.dot(W.transpose(), y)
```



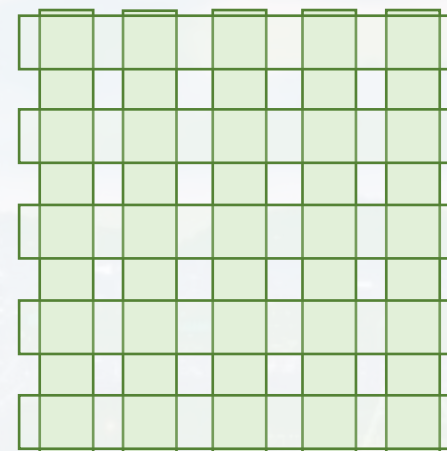
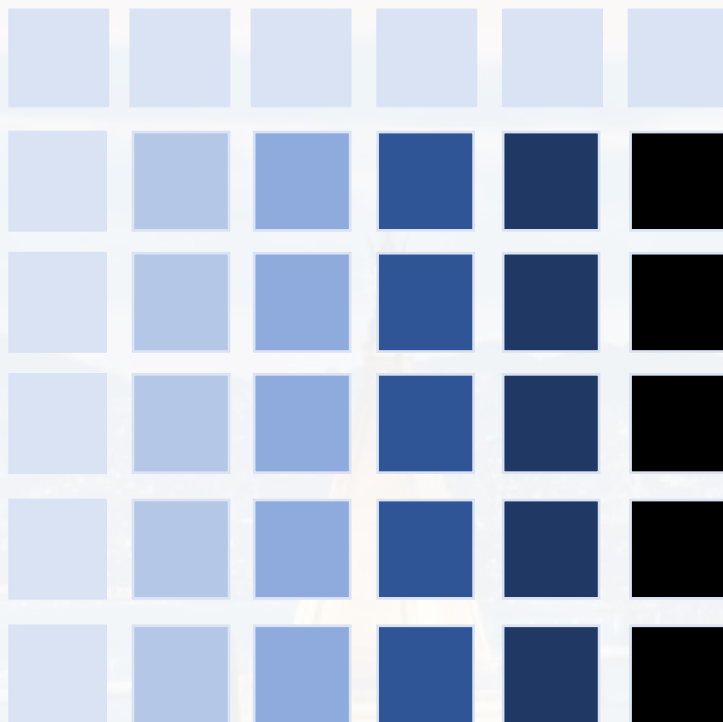
check out:

SmoothGaussKernel.py  
SmoothExamples.py





*“The cat jumped on the roof.”*

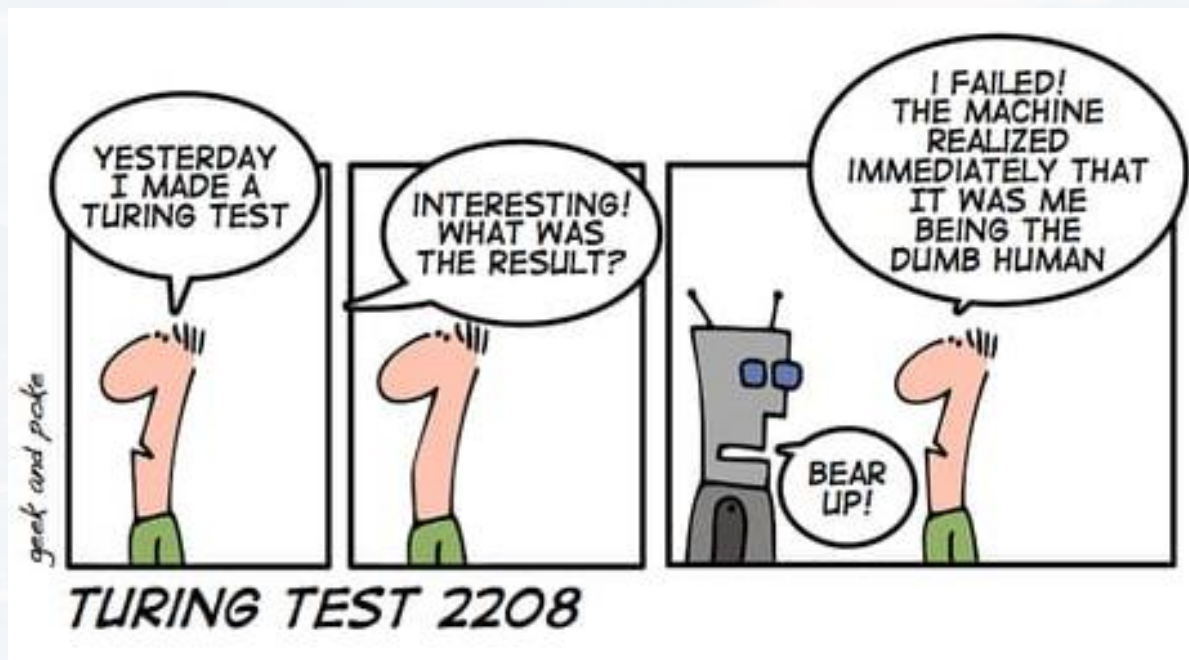


```
Gaussian kernel    W      = np.exp(-(D**2)/(sigma))  
                   W      = W/np.sum(W)  
                   yint   = np.dot(W.transpose(), y)
```

**actual attention:**  
**these weights are learnable,**  
**no kernel assumed!**



### Outline

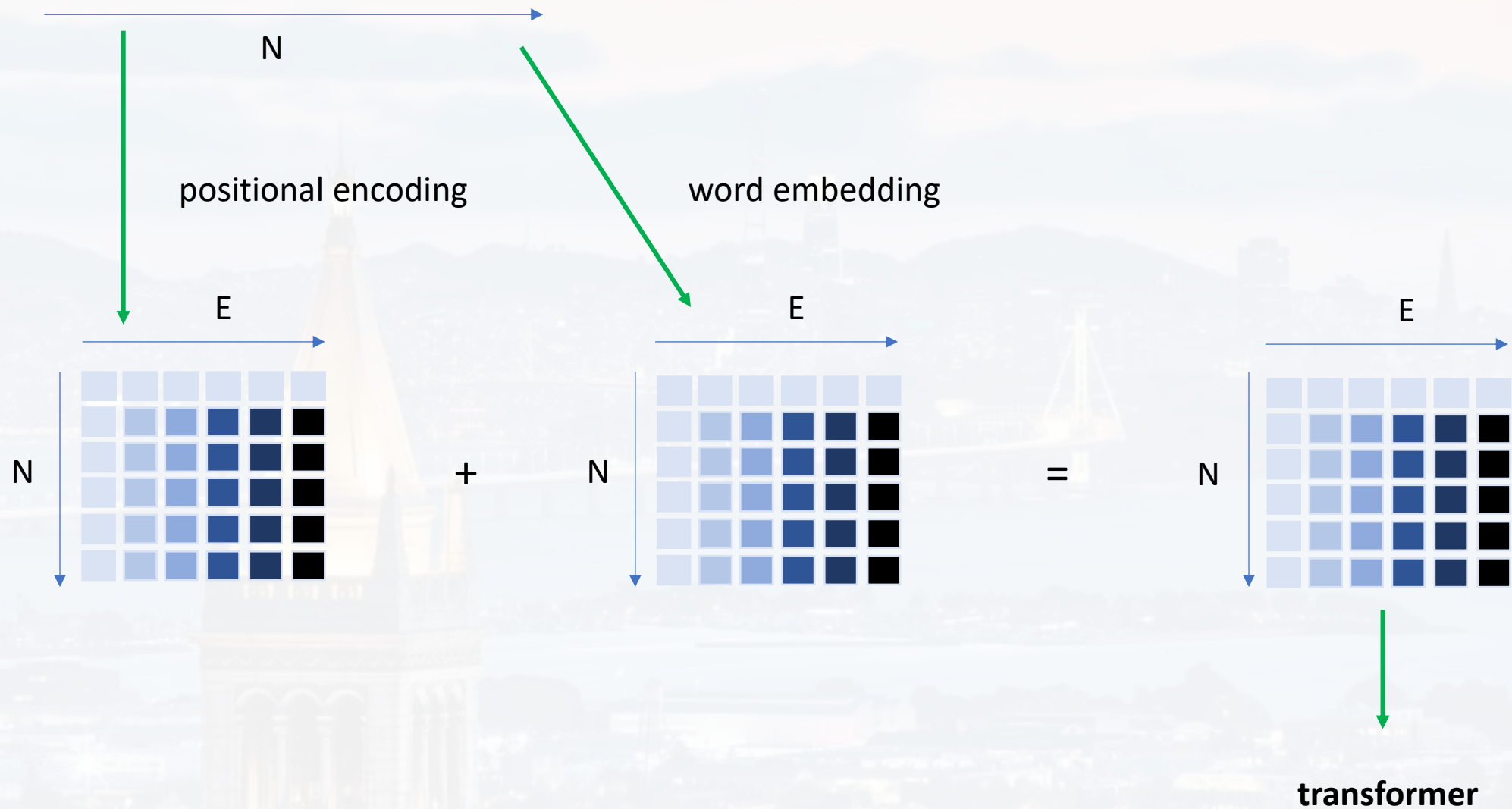


- Introduction
- Bigram and MAP
- Positional Encoding
- Word Embedding
- Attention
- **Transformer Architecture**



**N:** number of token  
**E:** number of embedding dimensions

*"The cat jumped on the roof."*

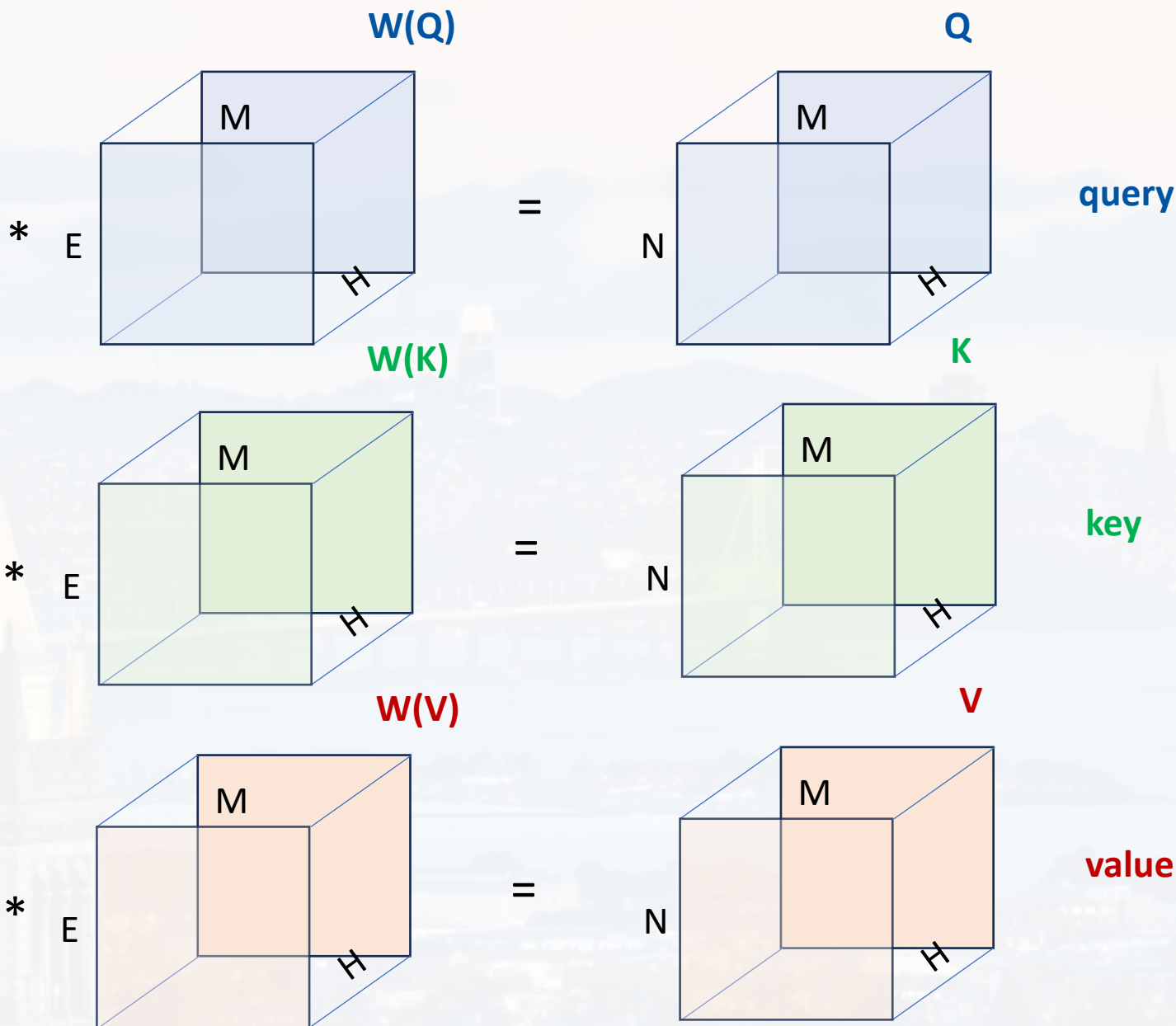
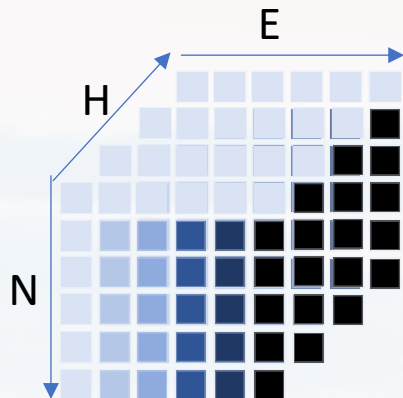






attention:

within the transformer:



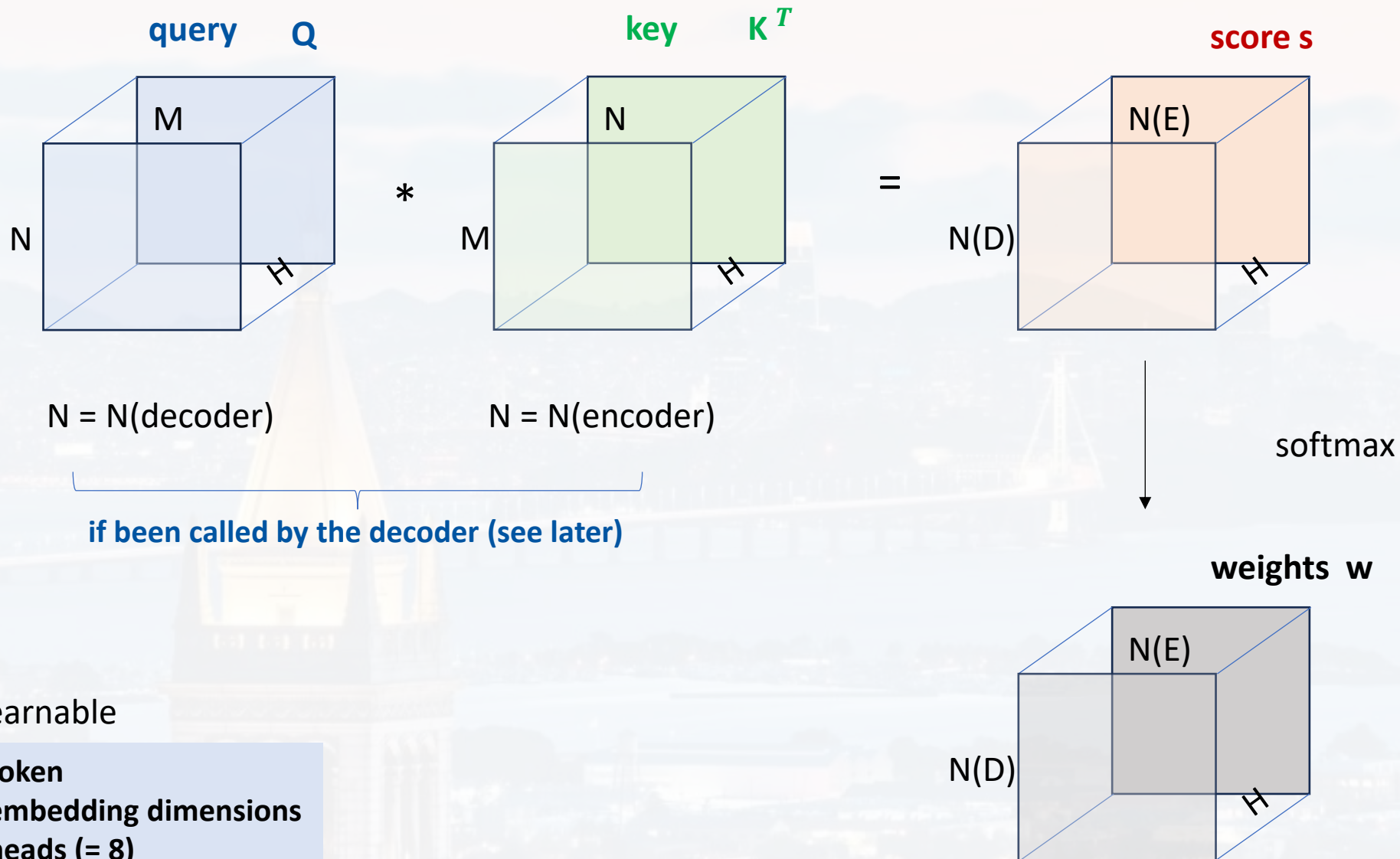
$W(Q)$ ,  $W(K)$ ,  $W(V)$ : learnable

**N:** number of token  
**E:** number of embedding dimensions  
**H:** number of heads (= 8)  
**M:** head size (= 64)



within the transformer:

attention:



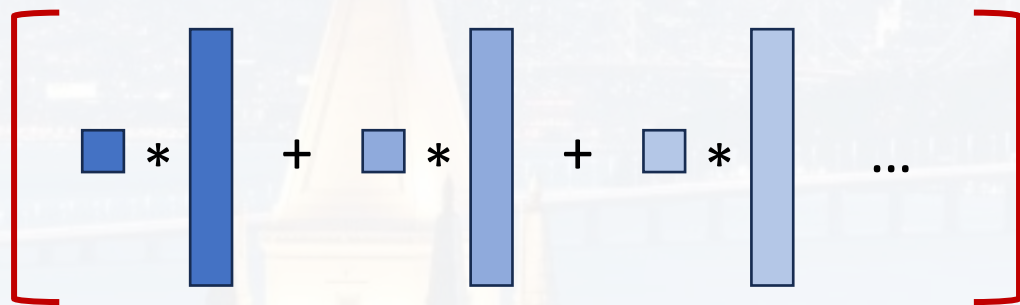
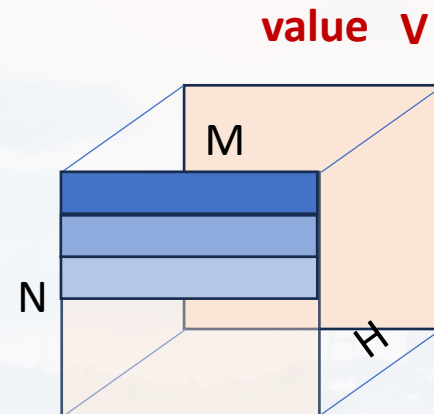
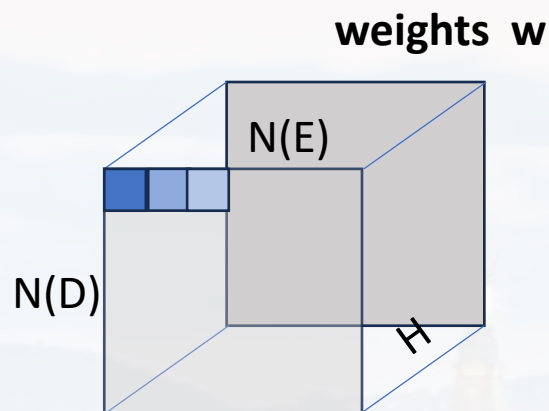
$W(Q)$ ,  $W(K)$ ,  $W(V)$ : learnable

**N:** number of token  
**E:** number of embedding dimensions  
**H:** number of heads (= 8)  
**M:** head size (= 64)



within the transformer:

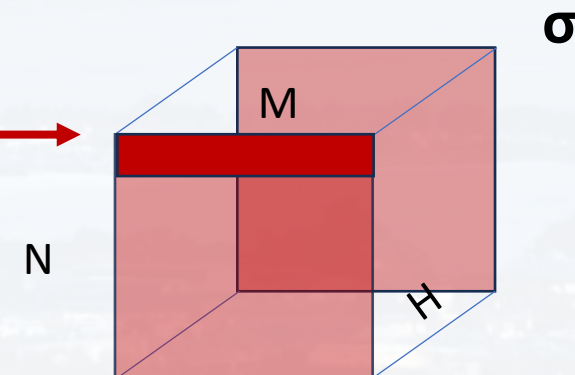
attention:



$$\sigma[i, :, k] = \sum_j w[i, j, k] * v[j, :, k]$$

N:	number of token
E:	number of embedding dimensions
H:	number of heads (= 8)
M:	head size (= 64)

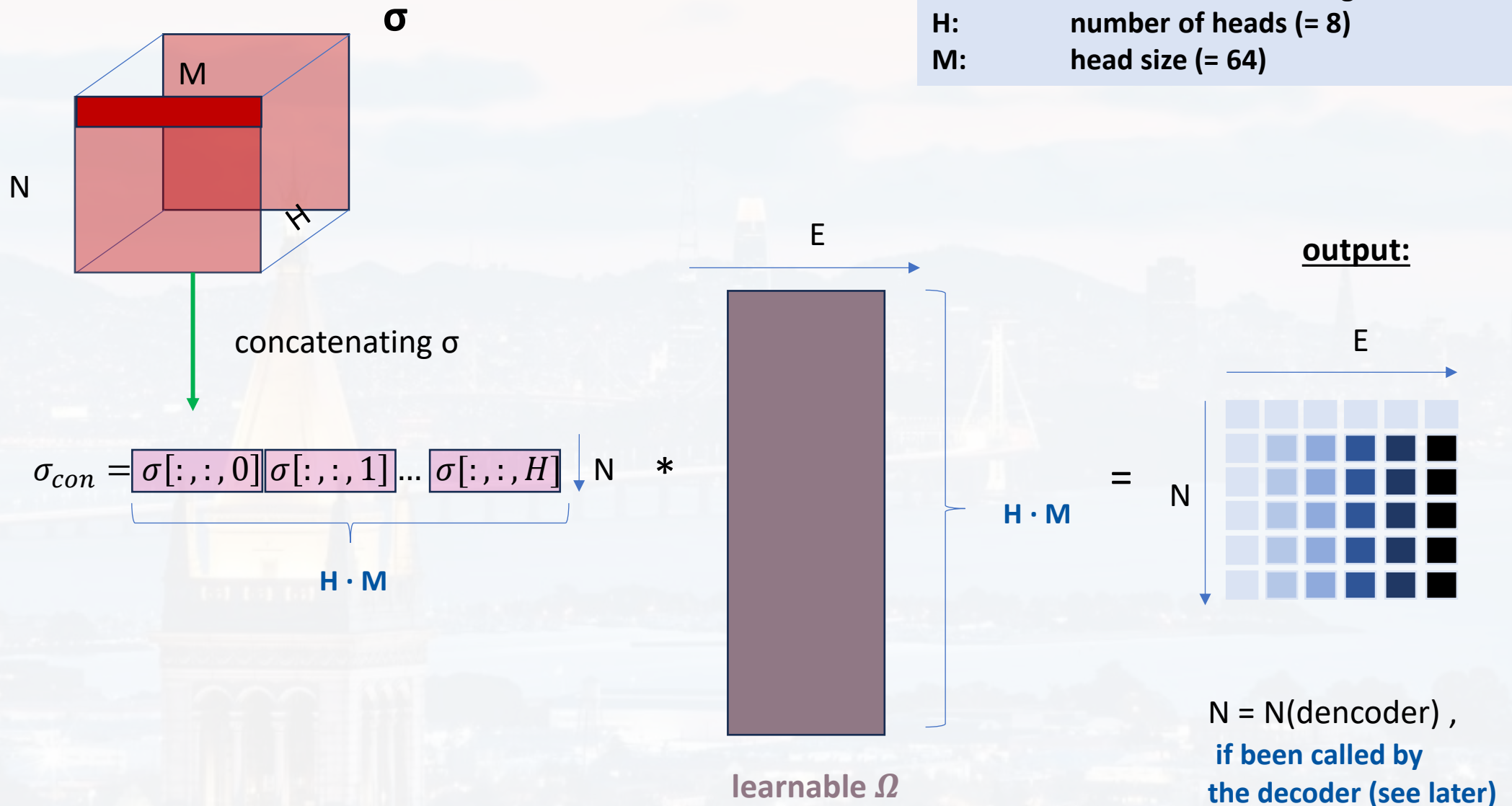
$N = N(\text{encoder})$  , **if been called by the decoder**  
(see later)







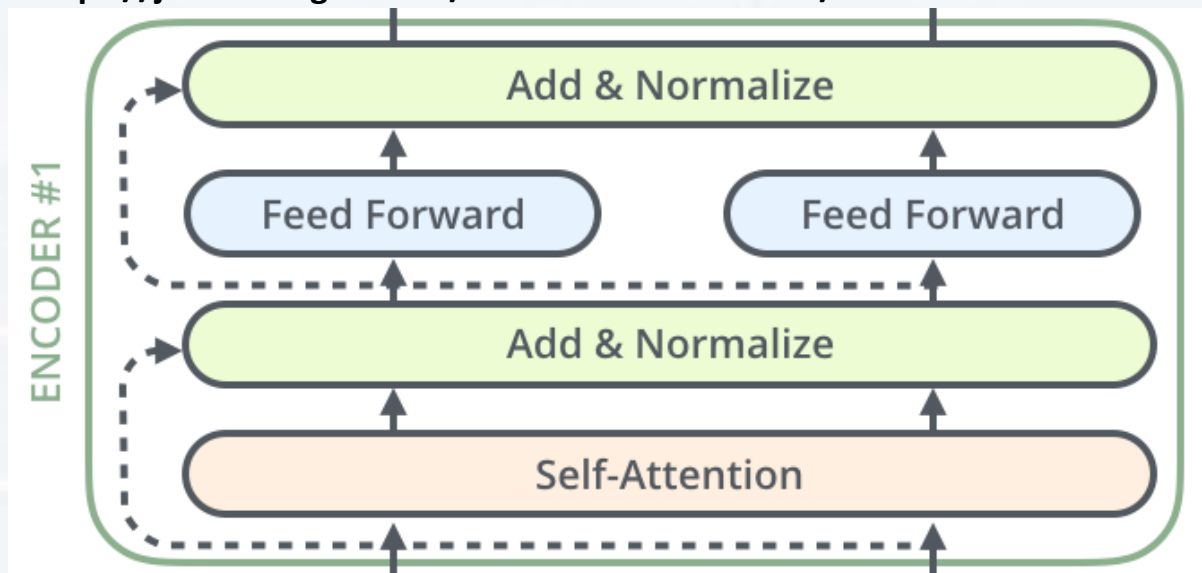
within the transformer:





that was attention → now: encoder

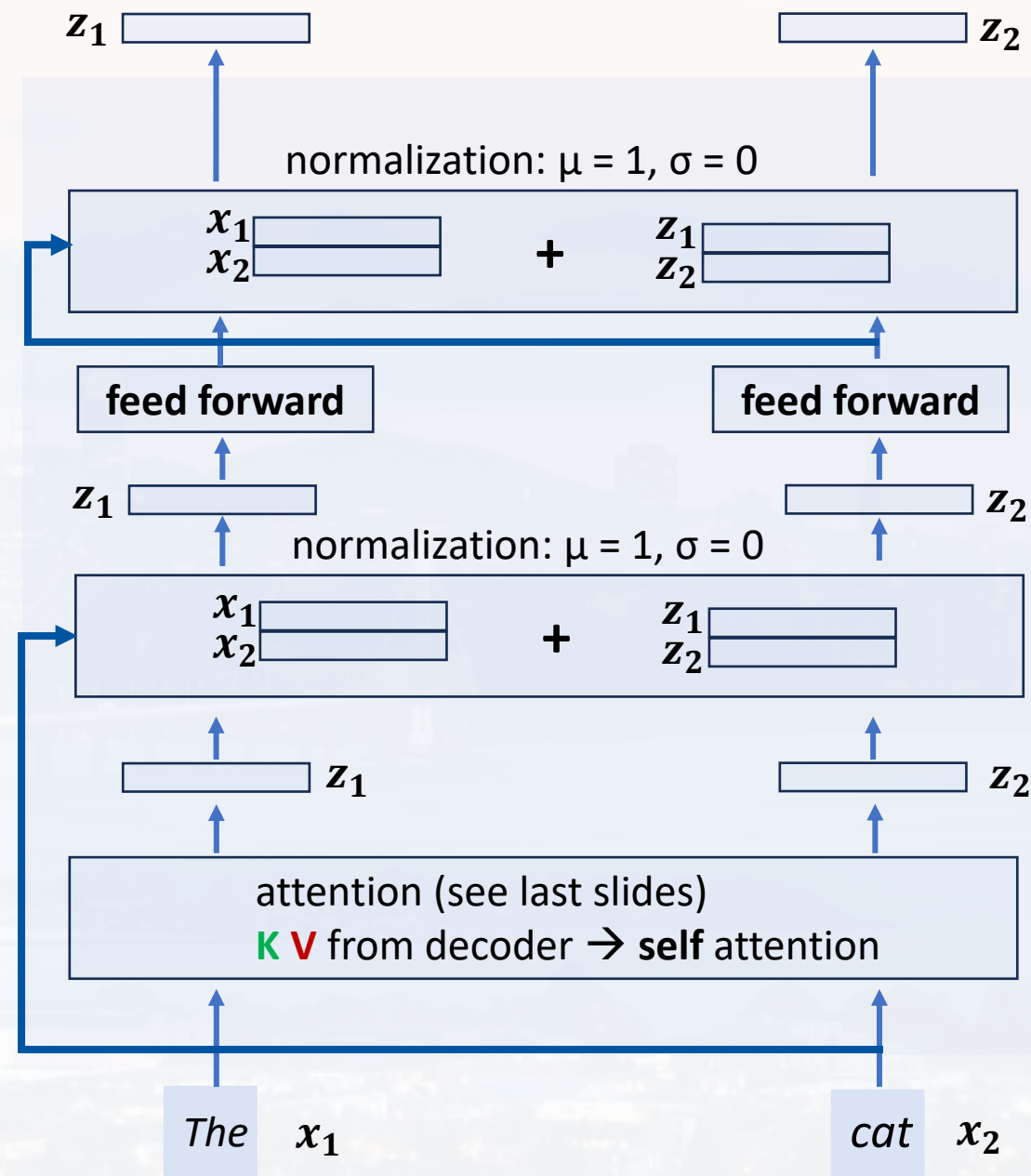
<https://jalammr.github.io/illustrated-transformer/>

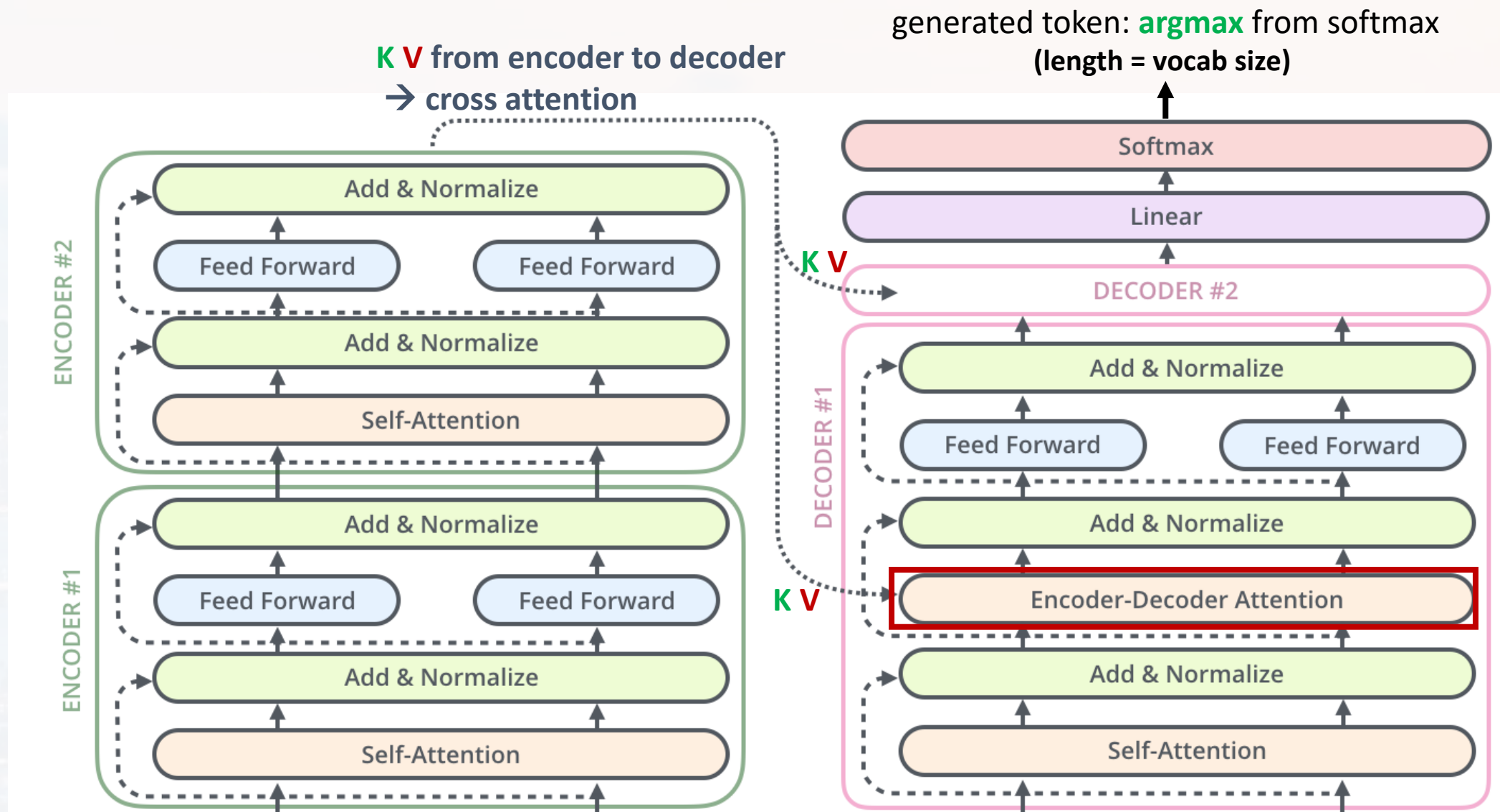


positional encoding

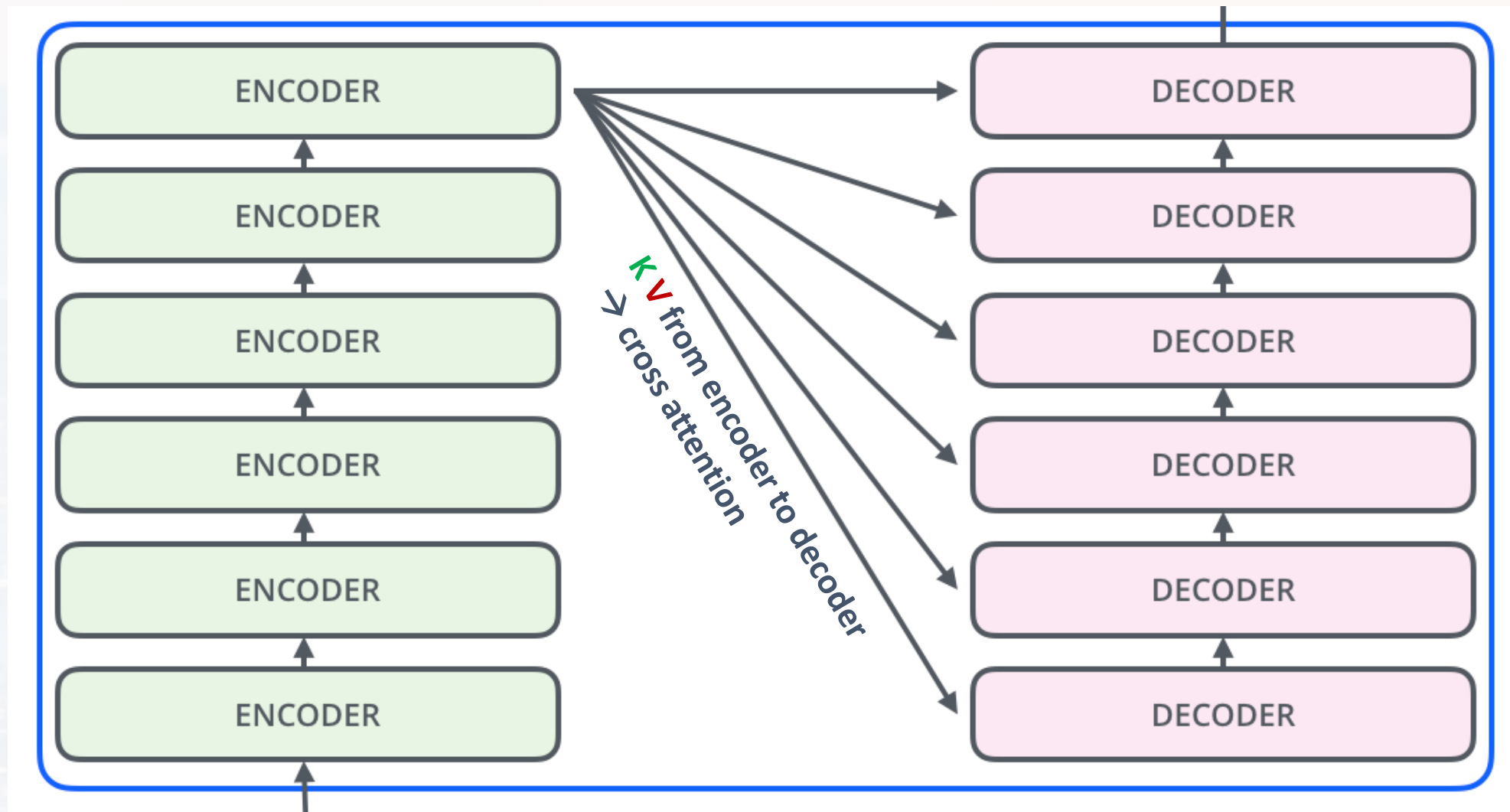
+

word embedding



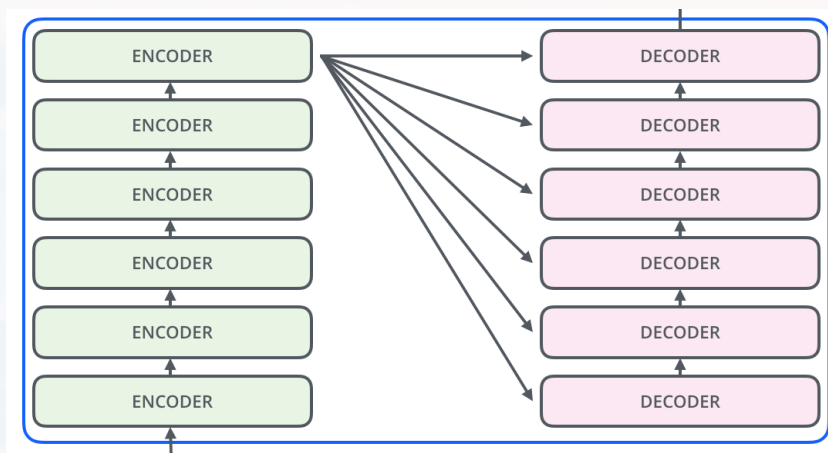








**K V** from encoder to decoder  
→ cross attention



**general:**  $N(E) \neq N(D)$

English: Let there be light.

$N(E) = 4$

German: Es werde Licht.

$N(D) = 3$

Latin: Fiat lux.

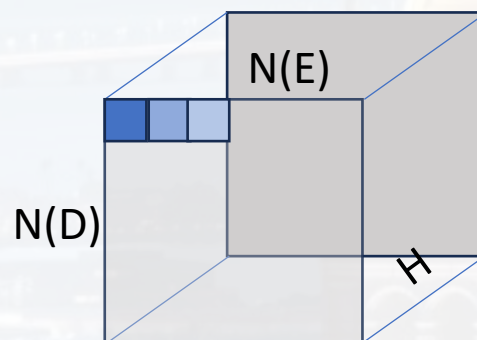
$N(D) = 2$

Hebrew: yehi ,or

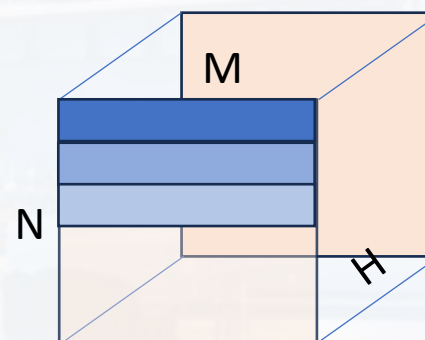
$N(D) = 2$

attention:

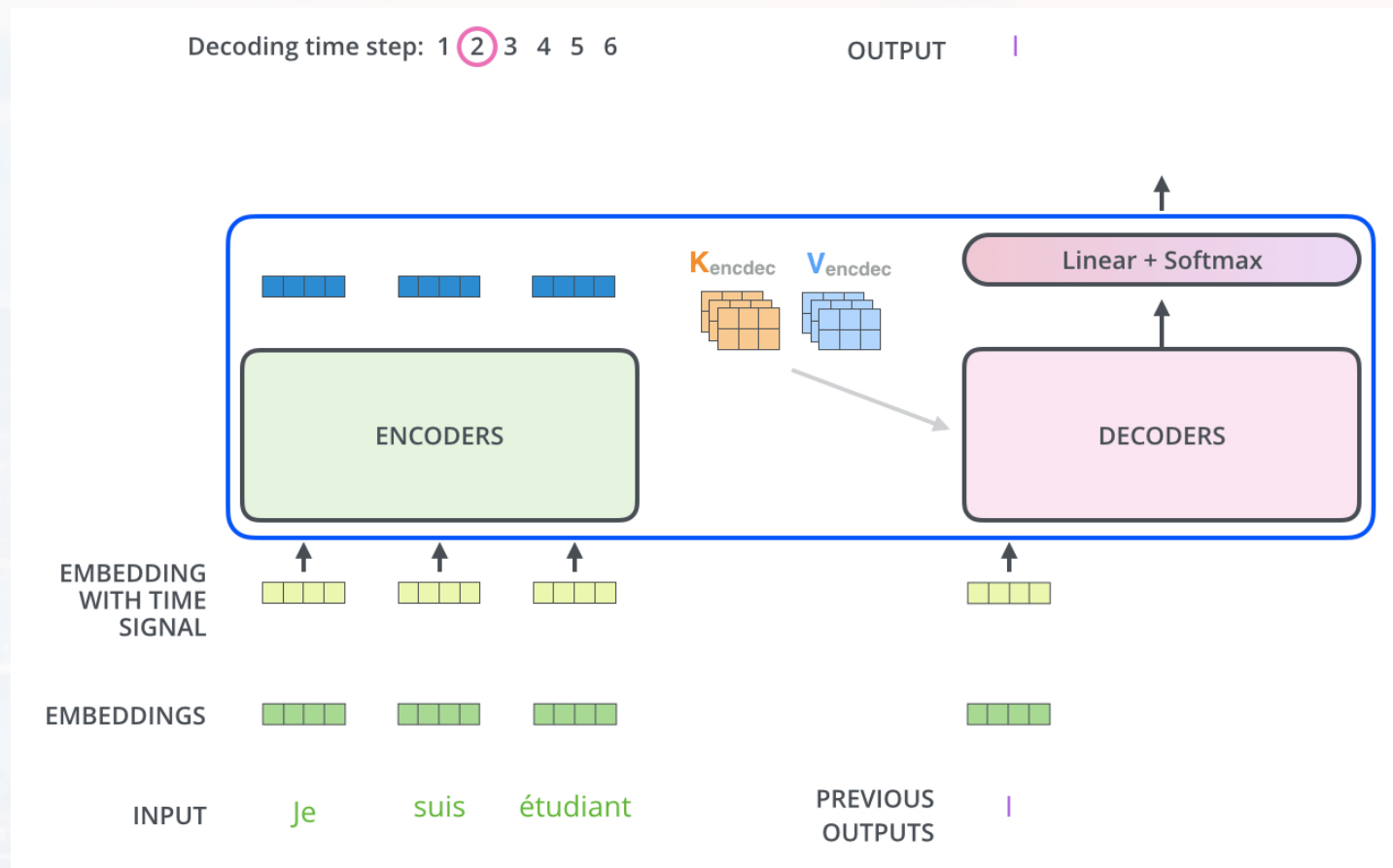
weights **w**



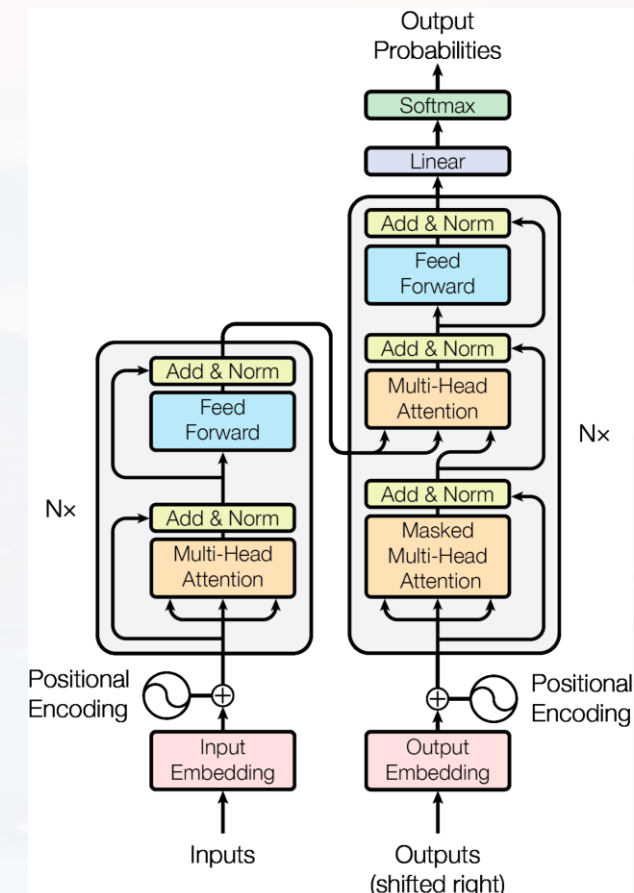
value **V**



$N = N(\text{encoder})$  , **if been called by the decoder**



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



Attention Is All You Need (Vaswani et al, 2017)





more about transformers:

Jay Alammar

Interactive Visualization

transformers intro



**Misra Turp**

@misratulp · 40.3K subscribers · 163 videos

Here is where we learn! This is a space to take it

[misratulp.com/roadmap](https://misratulp.com/roadmap) and 3 more links

Subscribe

building GPT from scratch!



**Andrej Karpathy**

@AndrejKarpathy · 451K subscribers · 14 videos

FAQ ...more

[karpathy.ai](https://karpathy.ai) and 2 more links

Subscribe

Thank you very much for your

