

Lecture 14:

Language Models and Transformer



Markus Hohle

University California, Berkeley

Bayesian Data Analysis and
Machine Learning for Physical
Sciences

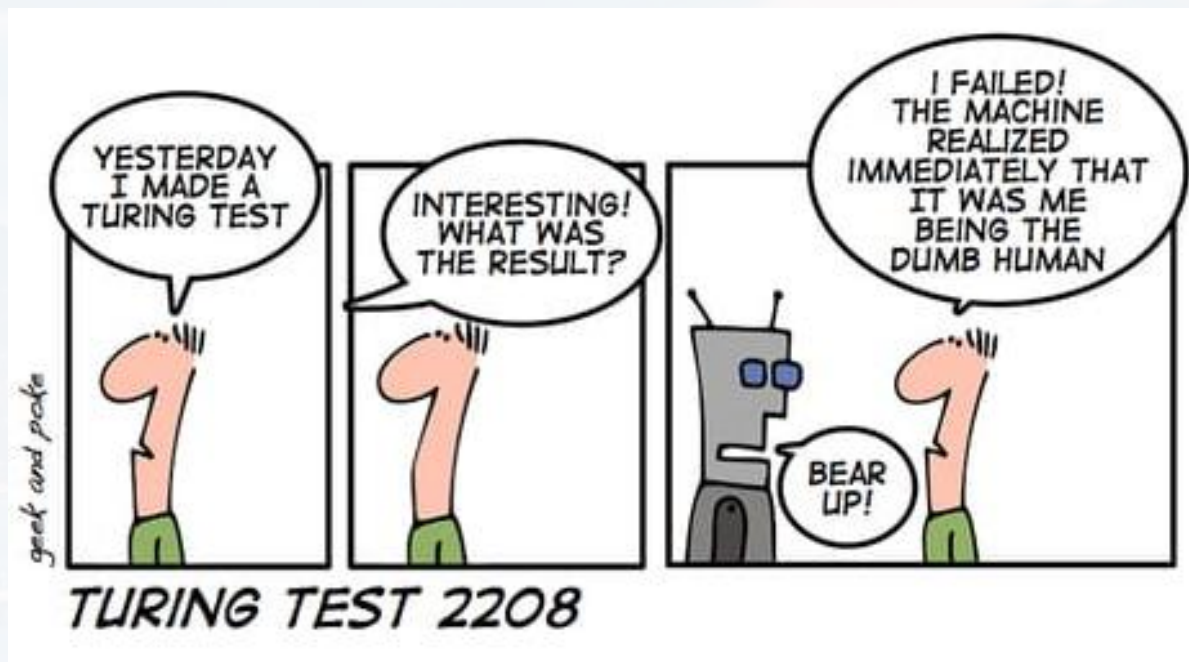


Course Map

Module 1	Maximum Entropy and Information, Bayes Theorem
Module 2	Naive Bayes, Bayesian Parameter Estimation, MAP
Module 3	MLE, Lin Regression
Module 4	Model selection I: Comparing Distributions
Module 5	Model Selection II: Bayesian Signal Detection
Module 6	Variational Bayes, Expectation Maximization
Module 7	Hidden Markov Models, Stochastic Processes
Module 8	Monte Carlo Methods
Module 9	Machine Learning Overview, Supervised Methods & Unsupervised Methods
Module 10	ANN: Perceptron, Backpropagation, SGD
Module 11	Convolution and Image Classification and Segmentation
Module 12	RNNs and LSTMs
Module 13	RNNs and LSTMs + CNNs
Module 14	Transformer and LLMs
Module 15	Graphs & GNNs



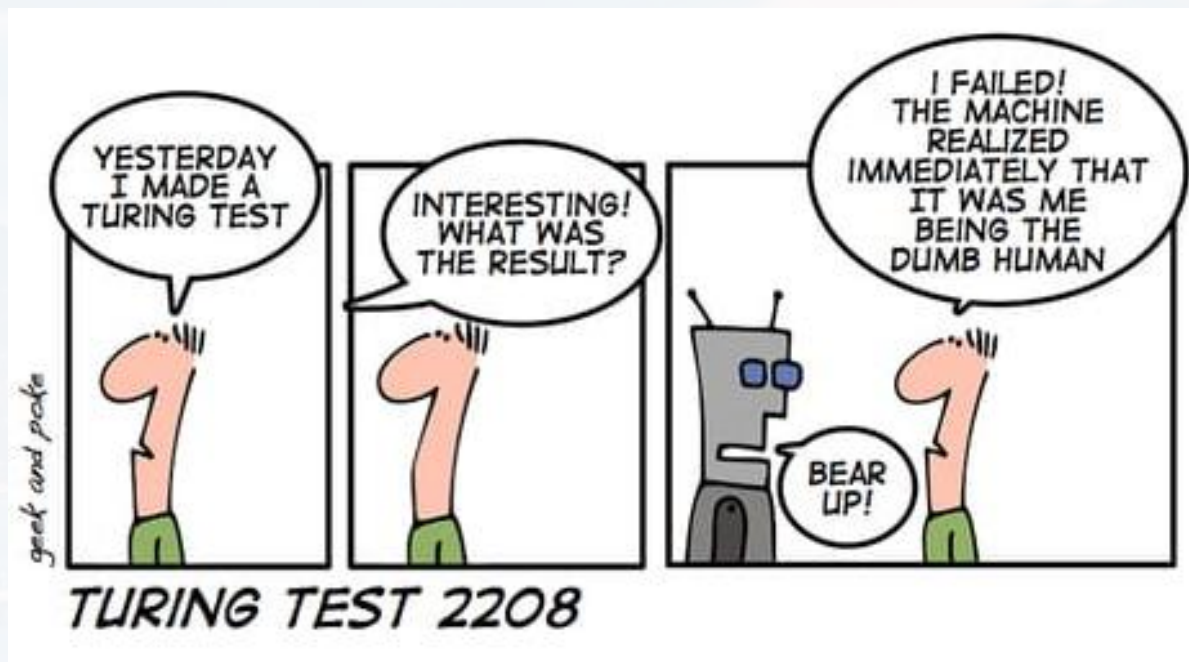
Outline



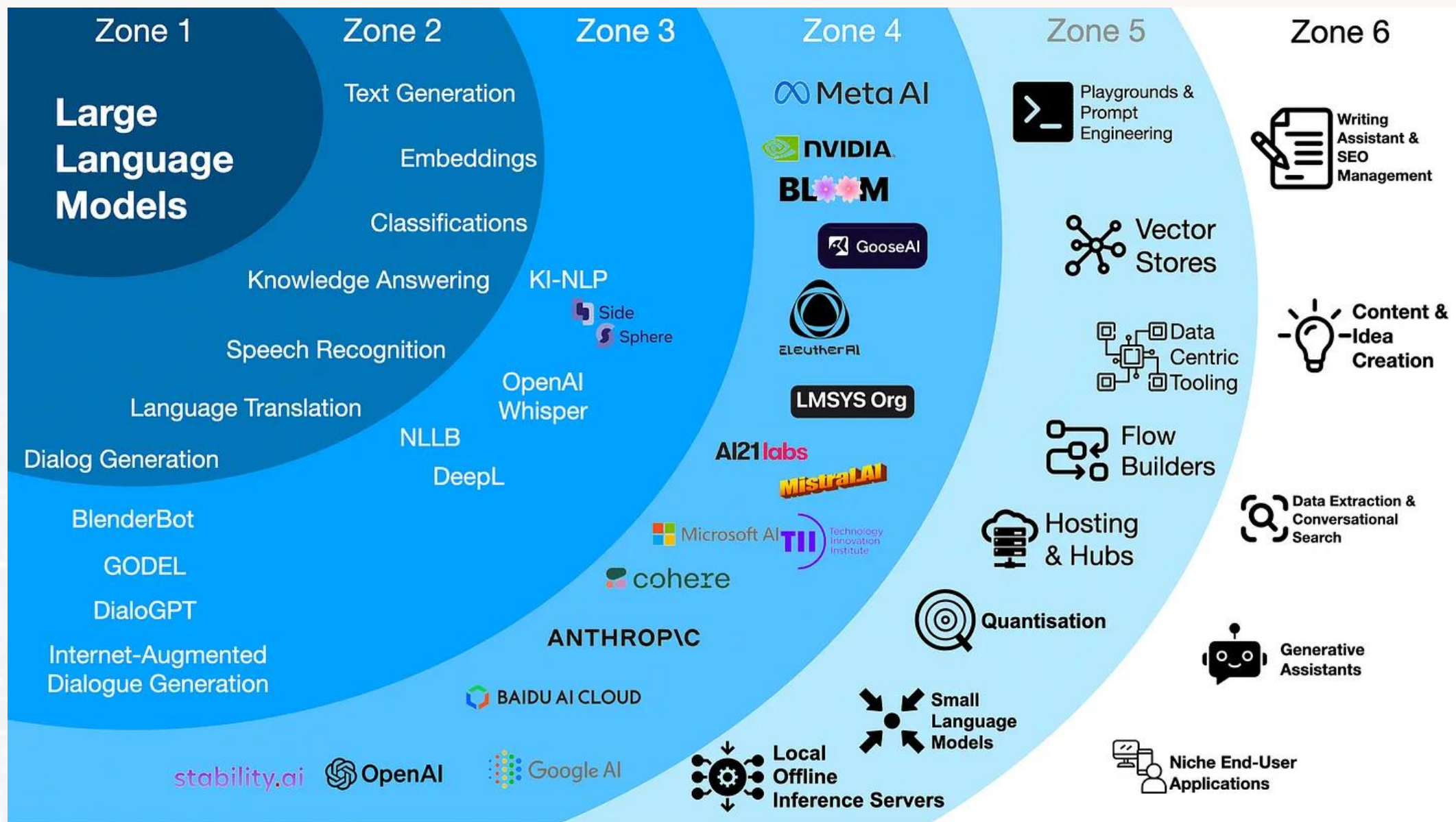
- Introduction
- Bigram and MAP
- Positional Encoding
- Word Embedding
- Attention
- Transformer Architecture



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

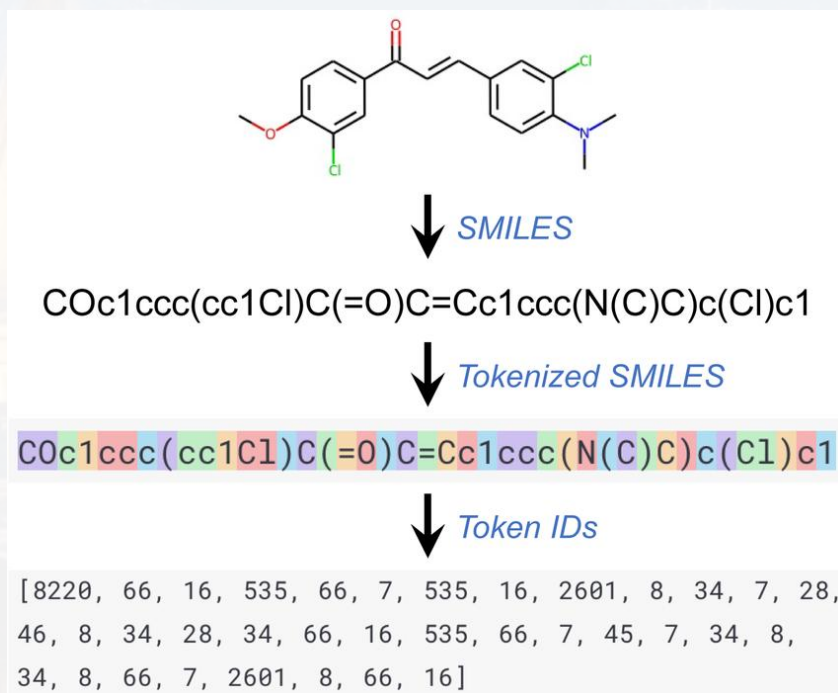


corpus: (large, representative) data set containing sequences of a language

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

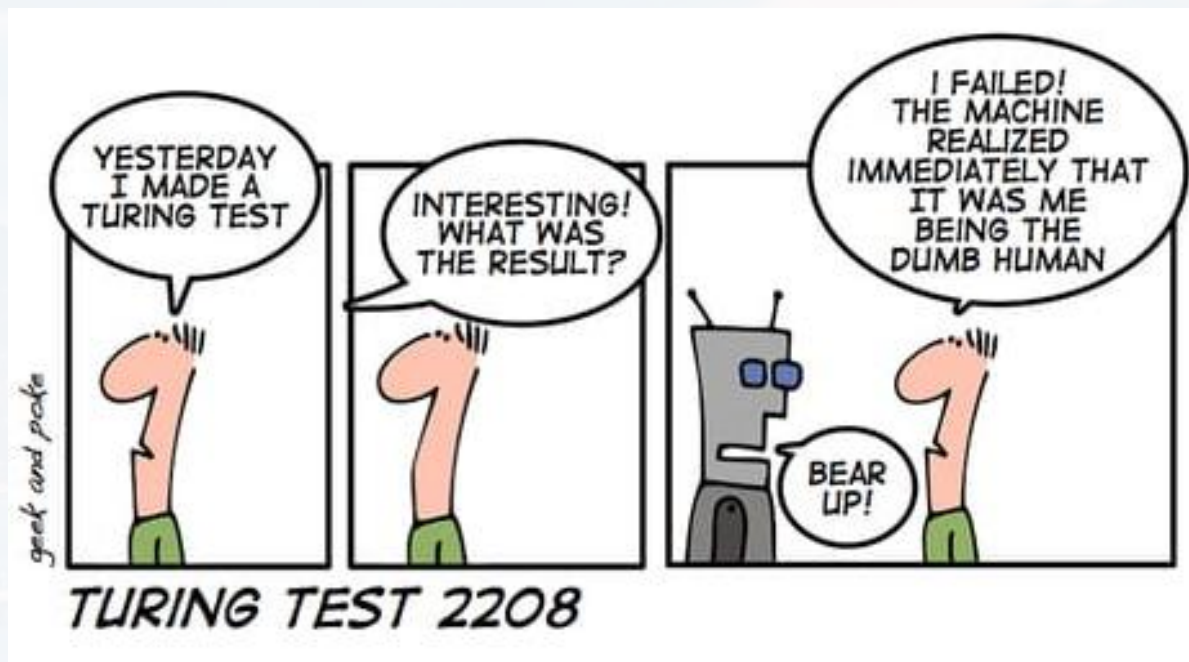
note: language models don't know grammar as we do, but they don't need to anyway...

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:

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

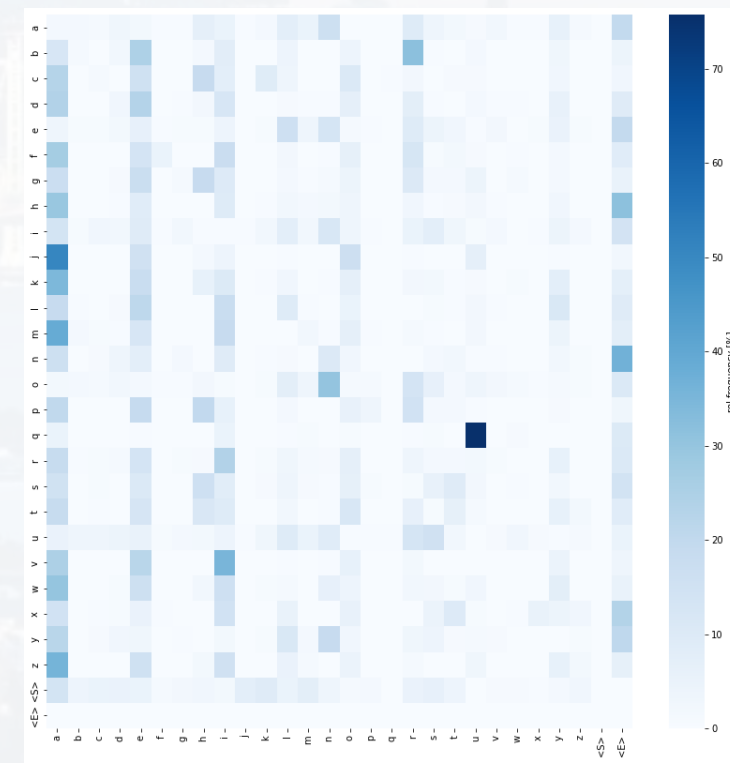
bigram (1st order Markov Chain, see e.g. first WhatsApp versions):

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

$P(i|j)$: 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 (1st 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 (1st order Markov Chain):

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

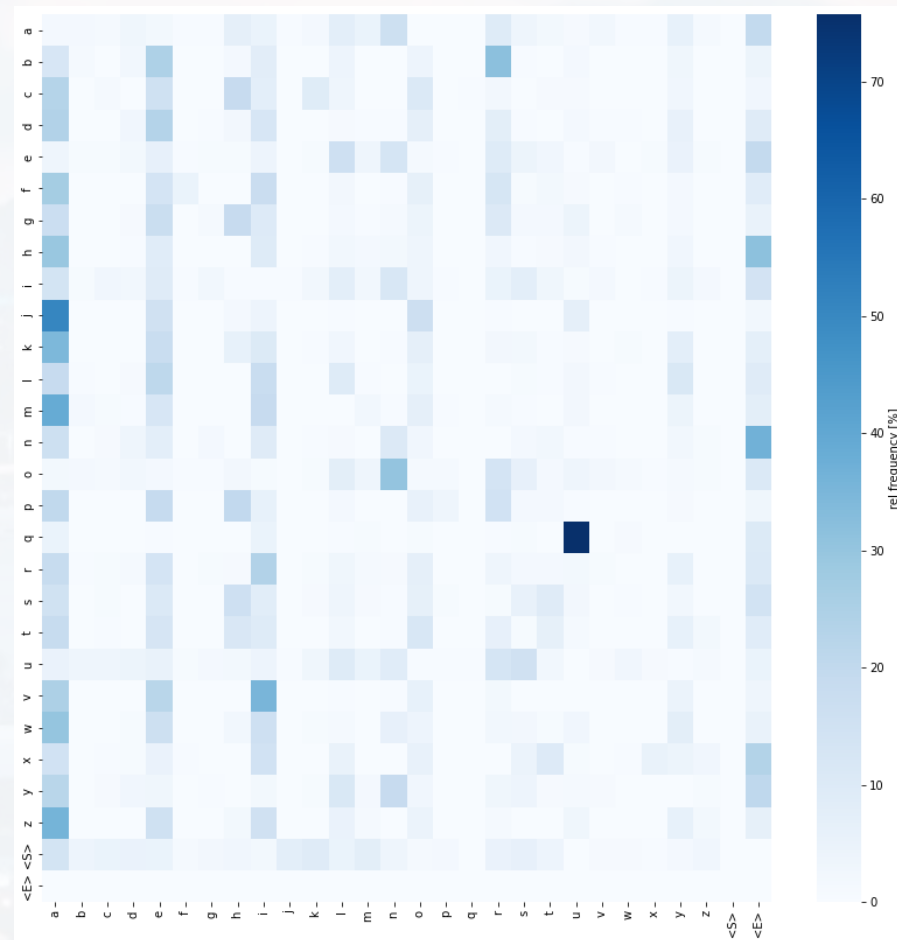
- 1) dictionary first (will help for counting)
- 2) count how often letter i is followed by letter j

→ 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 (1st 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>inzoerobzwoovx
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



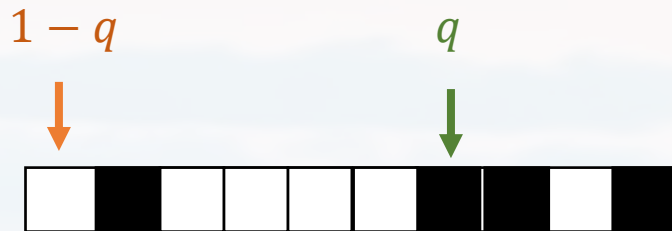
$$P(k|n, q) = \binom{n}{k} q^k (1-q)^{n-k}$$

$$q = ?$$



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

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 1st data point
(max entropy!)

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

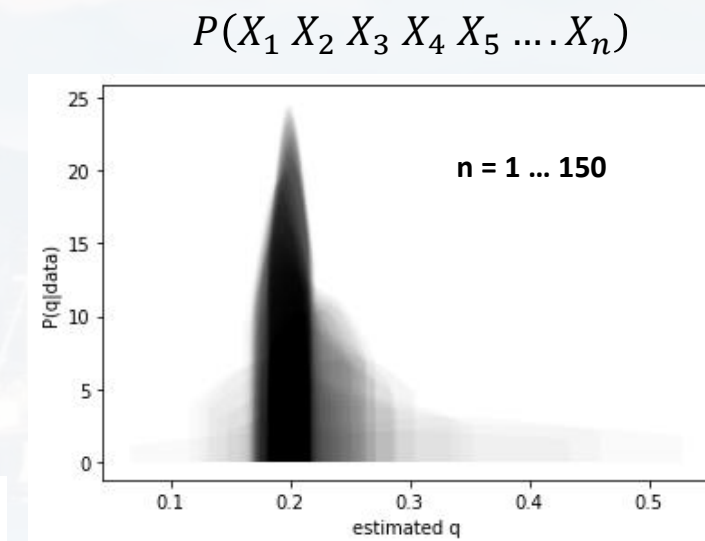
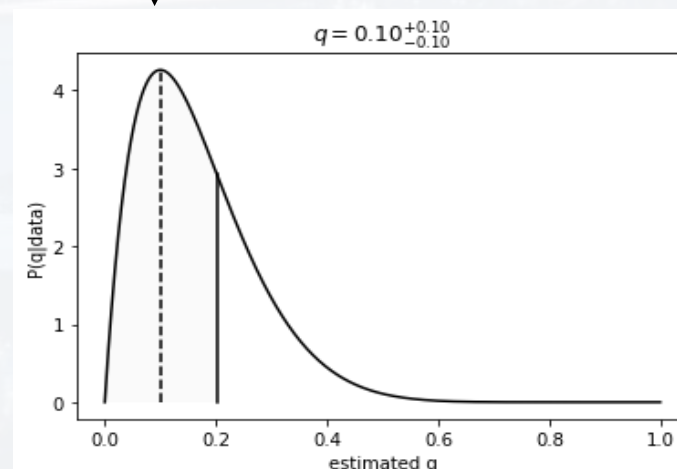
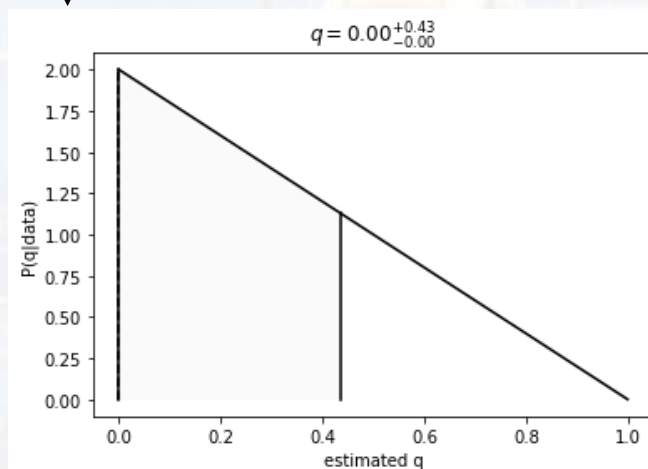
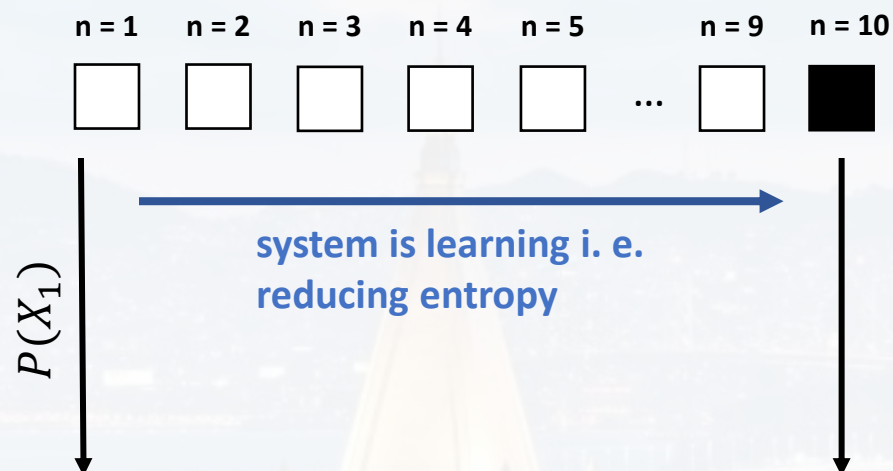
$q = \text{conjugate prior}$
after nth data point



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

Bayesian
Parameter
Estimation*)

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



*) see lecture 2



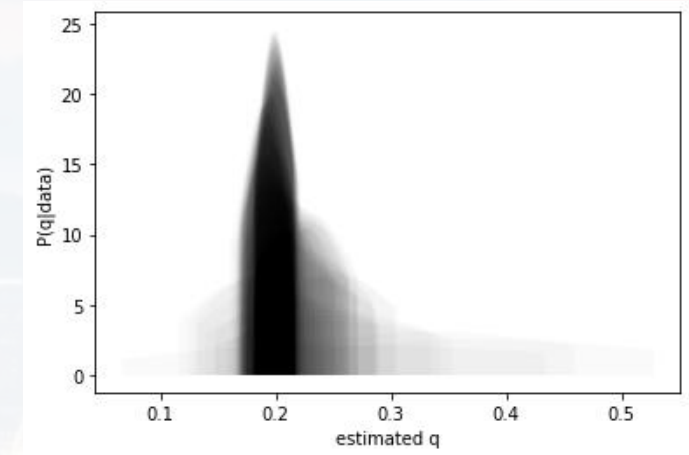
$$P(X_1 X_2 X_3 X_4 X_5 \dots X_n) = P(X_n | X_{n-1} \dots X_1) P(X_{n-1} | X_{n-2} \dots X_1) \dots P(X_1)$$

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

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

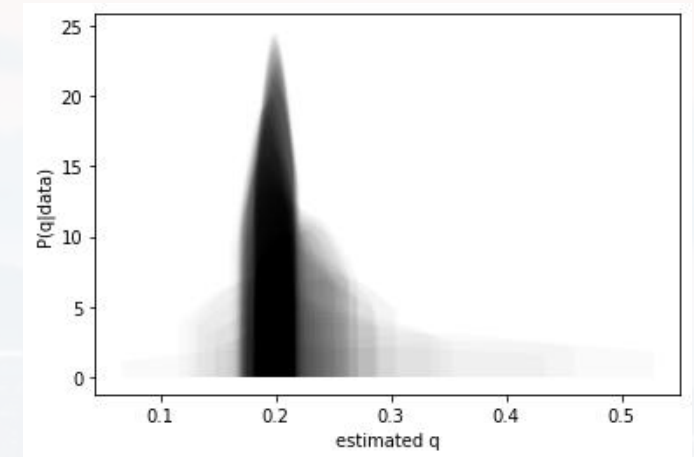


$$P(X_1 X_2 X_3 X_4 X_5 \dots X_n) = P(X_n | X_{n-1} \dots X_1) P(X_{n-1} | X_{n-2} \dots X_1) \dots P(X_1)$$

Dirichlet distribution

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

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



- 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 $P(\text{sequence})$ 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)

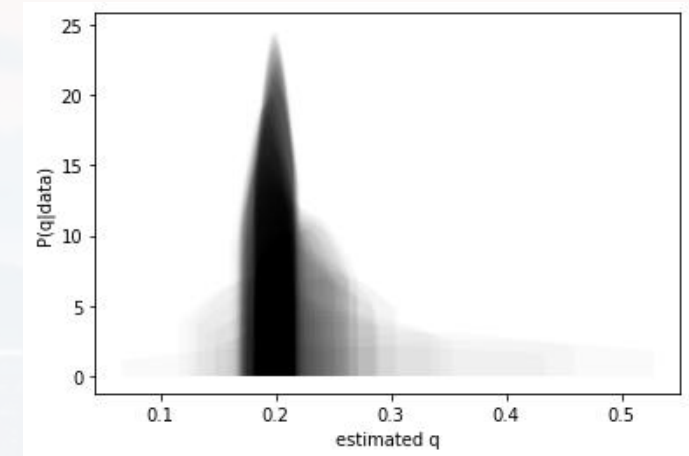


$$P(X_1 X_2 X_3 X_4 X_5 \dots X_n) = P(X_n | X_{n-1} \dots X_1) P(X_{n-1} | X_{n-2} \dots X_1) \dots P(X_1)$$

Dirichlet distribution

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

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

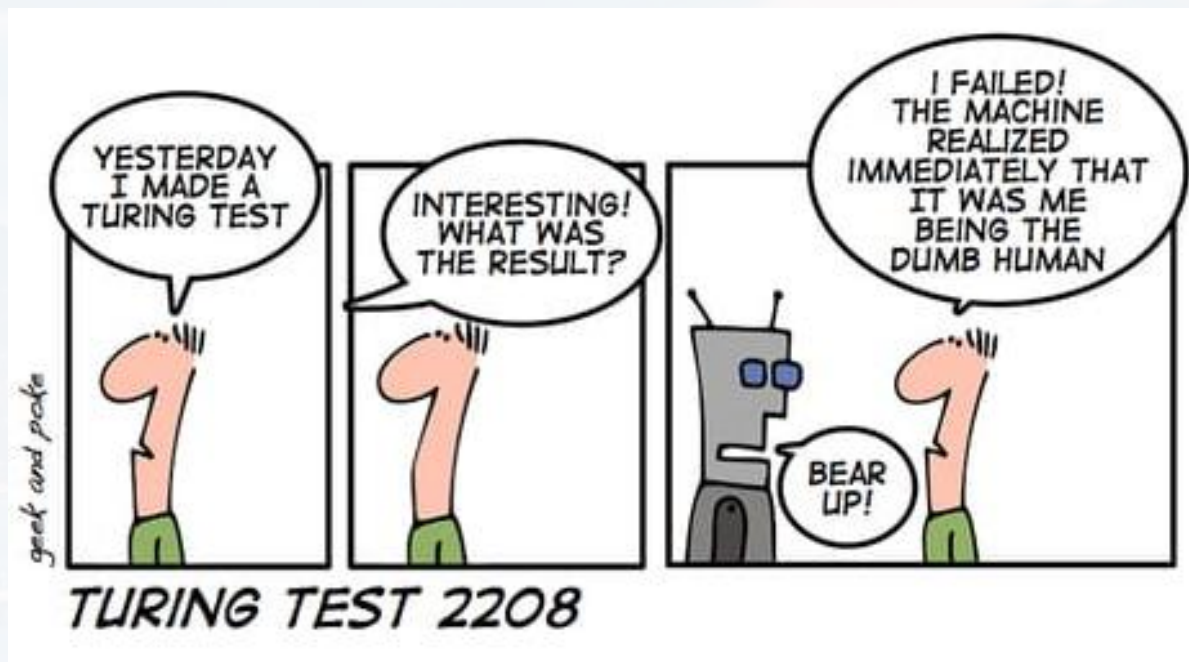


Maximum a-posteriori (MAP)

- 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



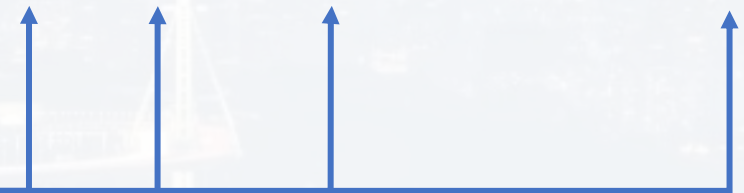
- Introduction
- Bigram and MAP
- **Positional Encoding**
- Word Embedding
- Attention
- Transformer Architecture



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)

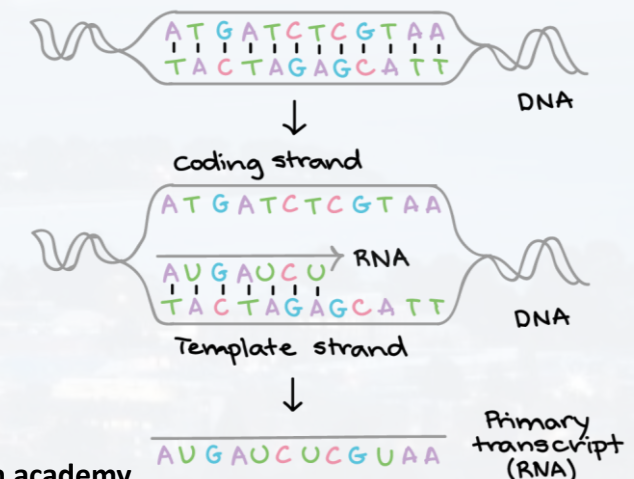
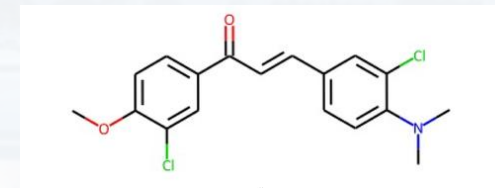
"The cat jumped on the roof."



order matters!:

- 1st: article
- 2nd: noun/subject
- 3rd: verb
- 4th: 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 1
2	000000 10
3	000000 11
4	00000 100
5	00000 101
6	00000 110
7	00000 111
8	0000 1000
9	0000 1001
10	0000 1010
11	0000 1011
12	0000 1100
13	0000 1101
14	0000 1110
15	0000 1111
16	000 10000

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

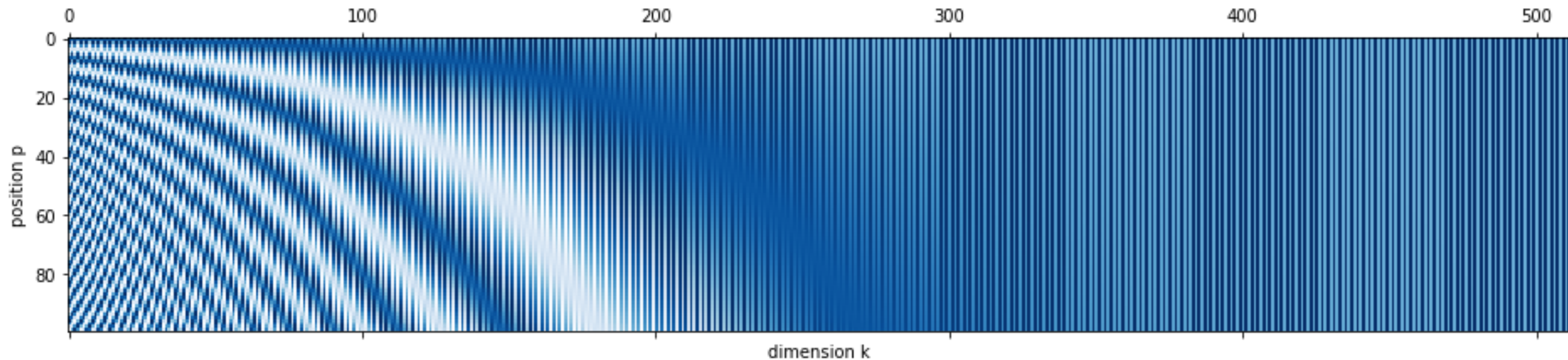
k : dimension index

d : number of dimensions

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

run **PlotPositionEncoding.py**

more info [here](#)





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

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

p : position in sequence

k : dimension index

d : number of dimensions

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

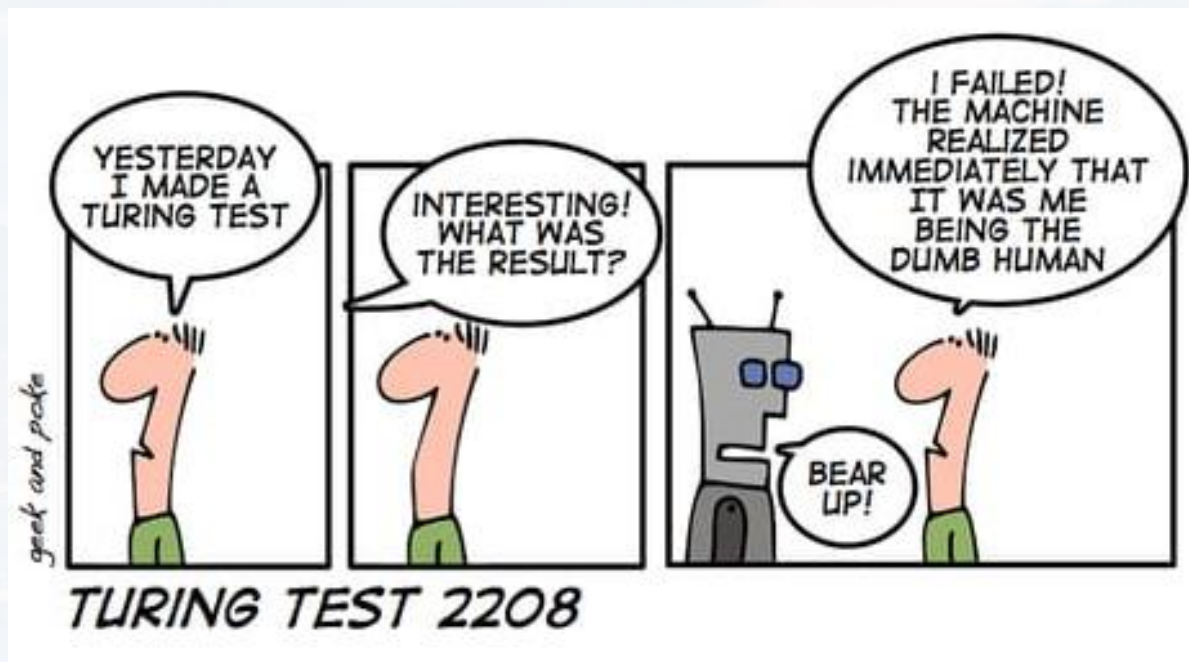
run **PlotPositionEncoding.py**

note:

- easier to handle numerically vs discrete encoding
- $\cos(x + 2\pi k) = \cos(x)$ and $\sin(x + 2\pi k) = \sin(x)$
→ absolute position not relevant but **relative** position



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

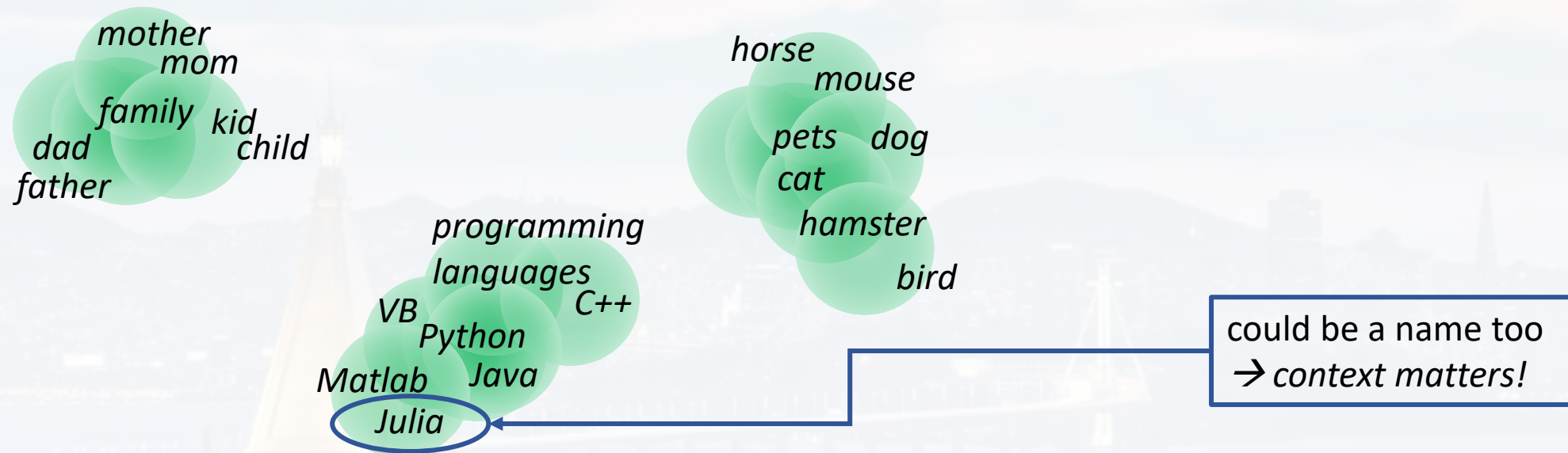
- $10^4 \dots 10^6$ (upper/lower case, cases, gender, tenses, conjugations)

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

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



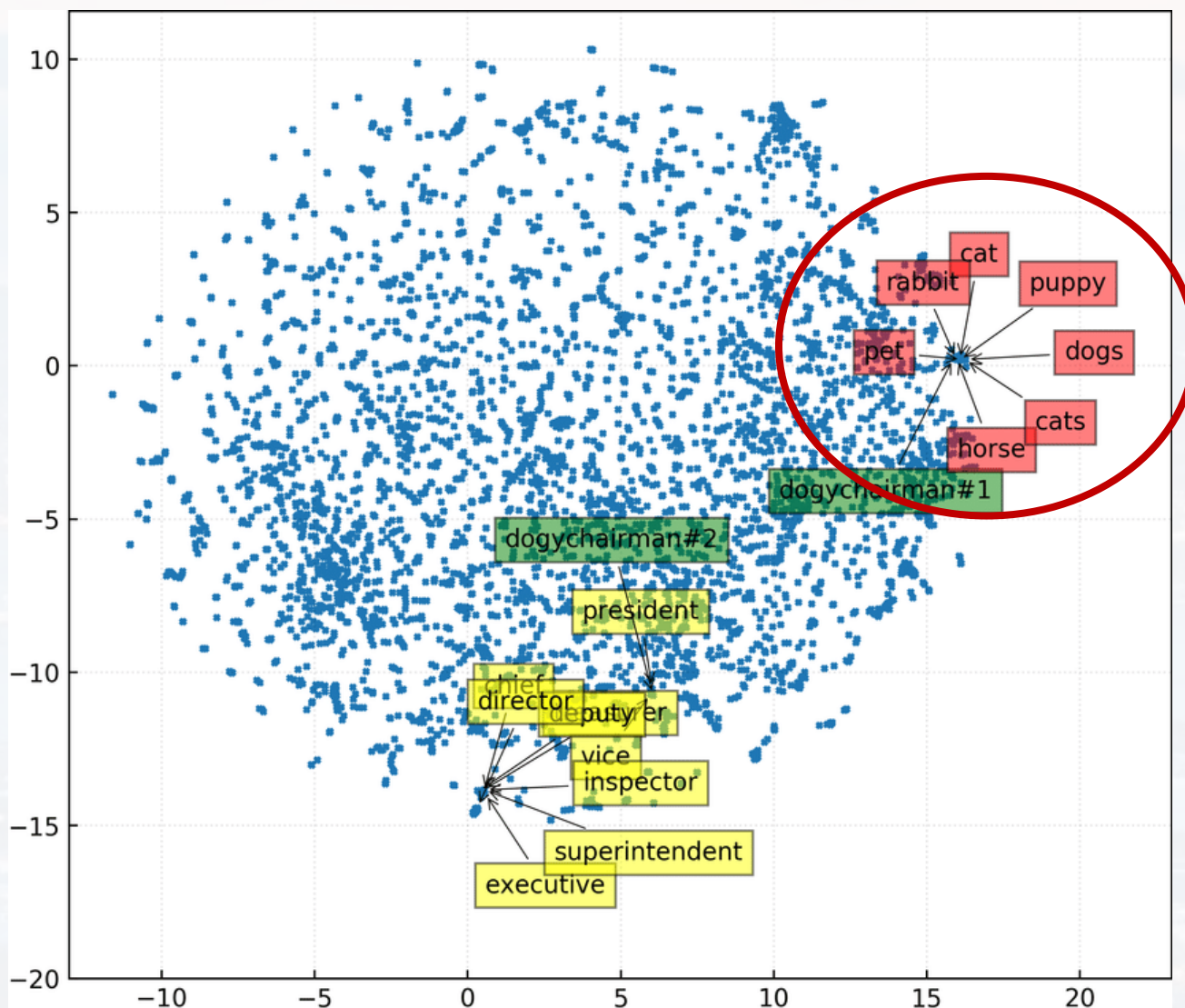
words with a **similar meaning** should form **cluster**



- 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...

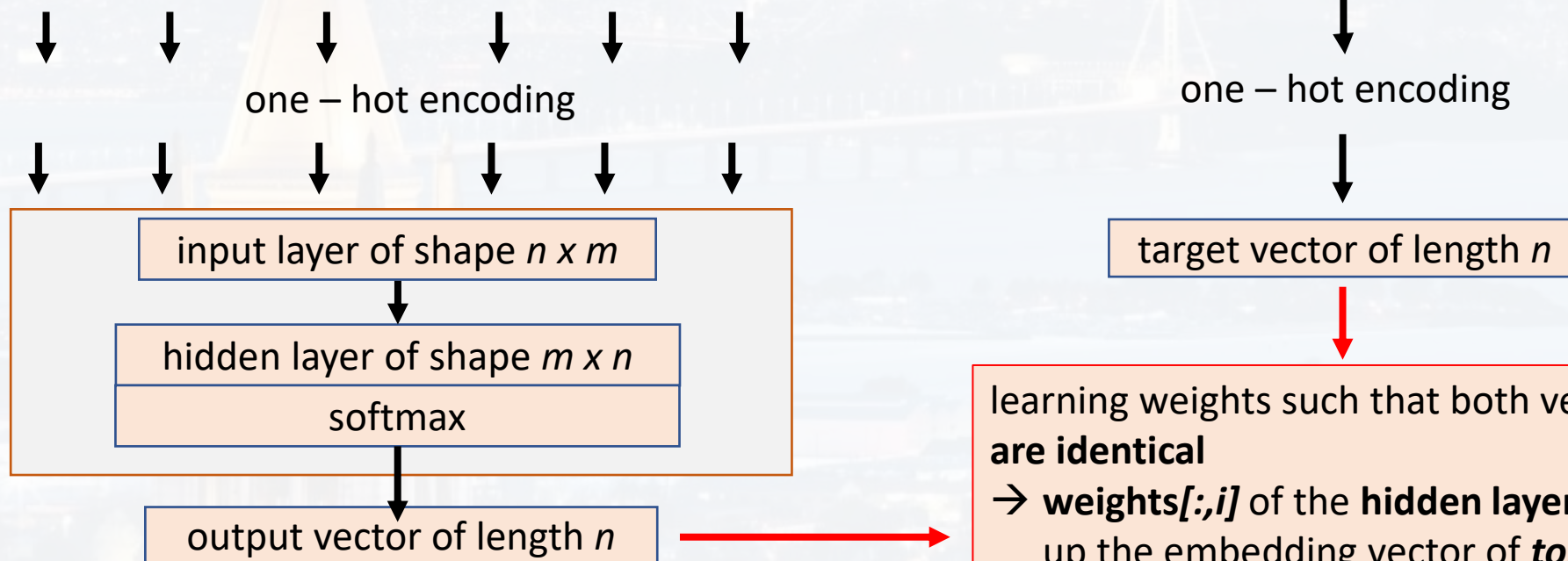
target

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...

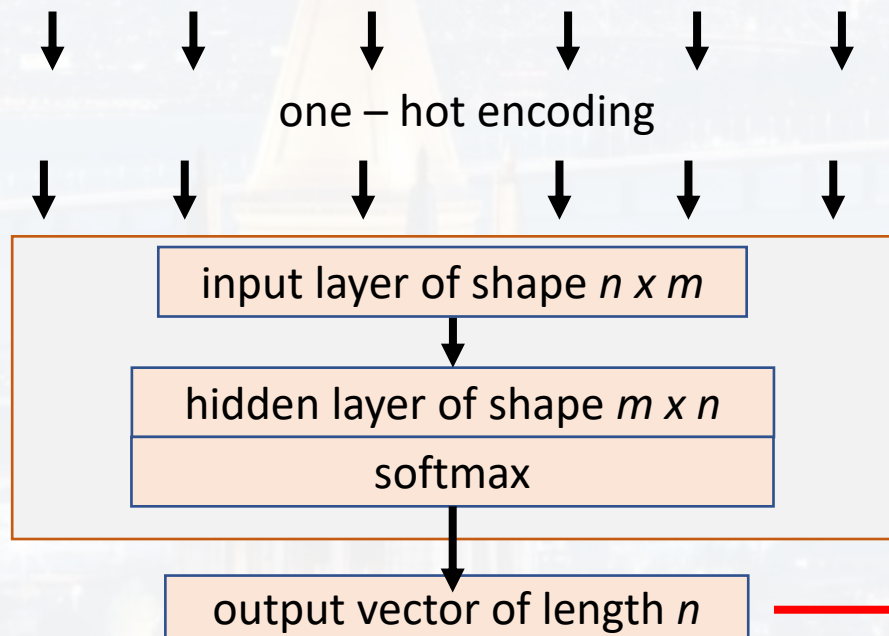
target

not

context window

competent services have *included them in*

shallow ANN



one – hot encoding

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**



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

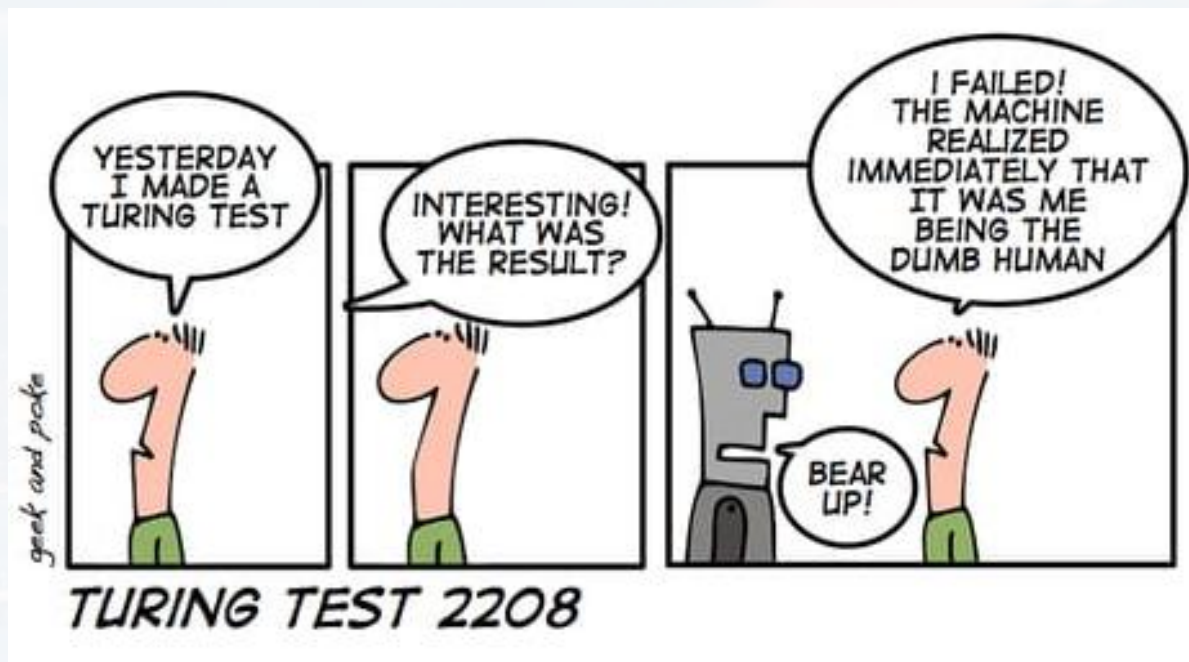
target vector of length n

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



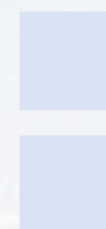
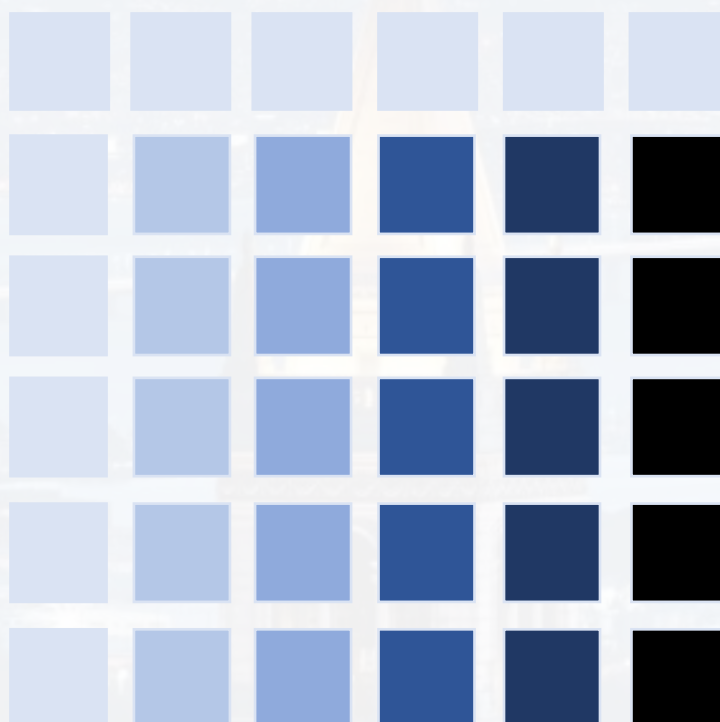
- Introduction
- Bigram and MAP
- Positional Encoding
- Word Embedding
- **Attention**
- Transformer Architecture



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)

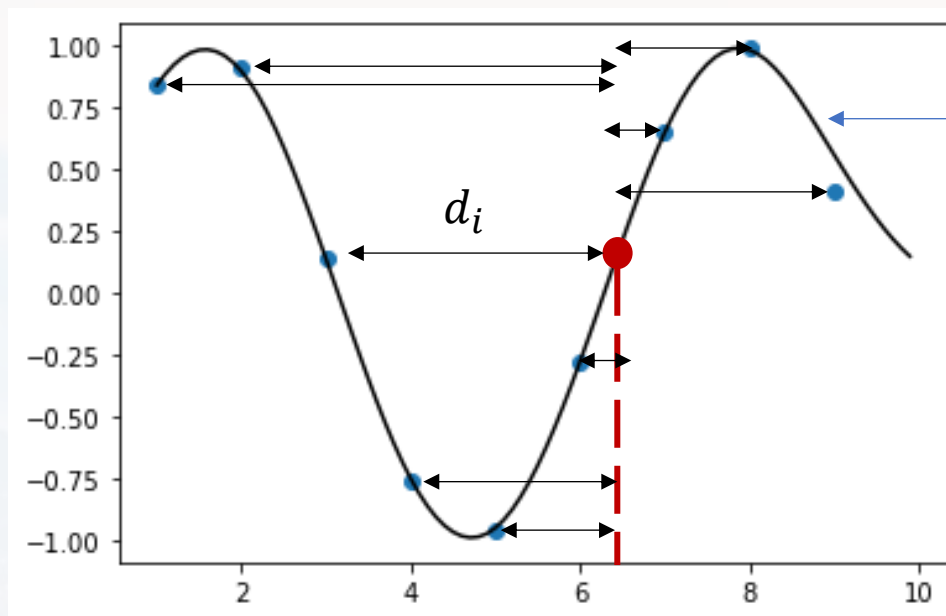
“The cat jumped on the roof.”



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(V.transpose(), (len(V), 1))
```

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

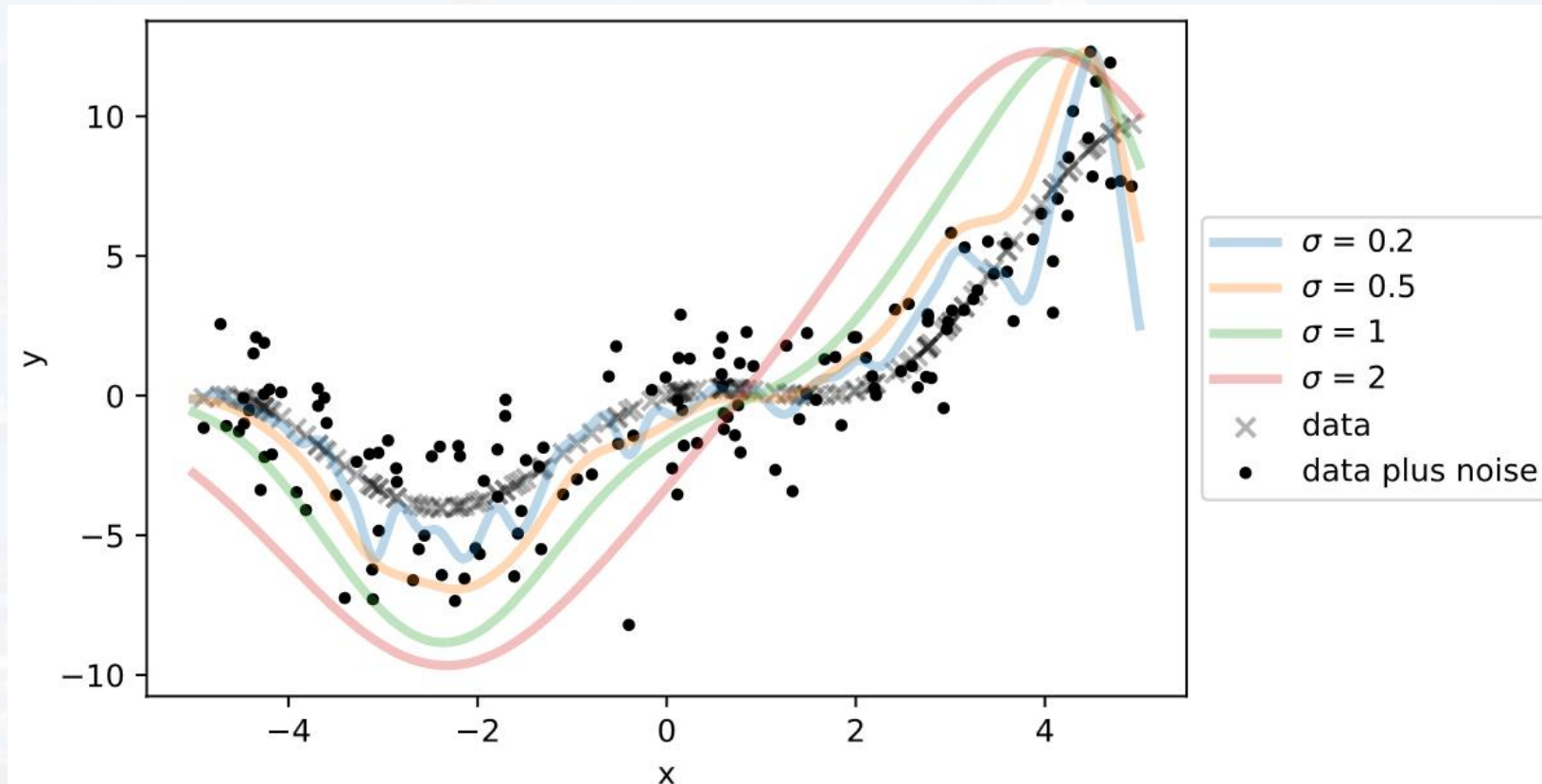
$$y_{int} \sim \sum_{i=1}^I w_i y_i$$

```
Gaussian kernel    W      = np.exp(-(D**2)/(sigma))
                   W      = W/np.sum(W + 1e-16, axis = 0)
                   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 + 1e-16, axis = 0)  
                  yint   = np.dot(W.transpose(), y)
```

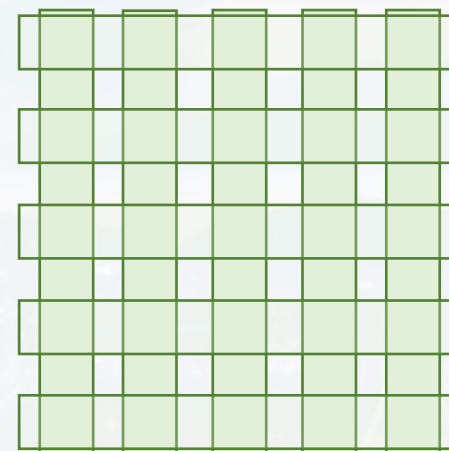
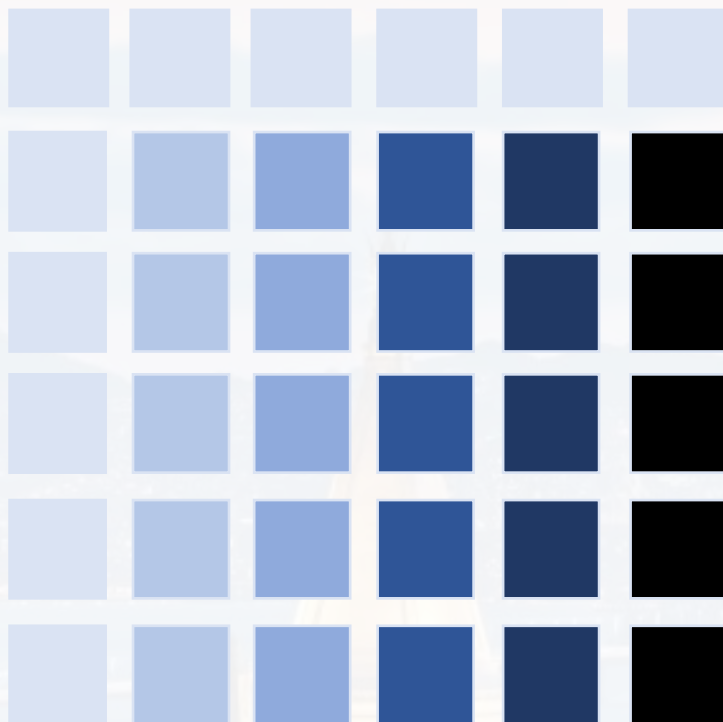


check out:

SmoothGaussKernel.py
SmoothExamples.py



"The cat jumped on the roof."



```
Gaussian kernel    W      = np.exp(-(D**2)/(sigma))  
                   W      = W/np.sum(W + 1e-16, axis = 0)  
yint               np.dot(W.transpose(), y)
```

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



self attention

imagine you want to built & train a movie GenAI **that creates movies based on queries.**

training data

“Thrilling horror science fiction movie, plays in space in a distant future”

$$X = X_1, X_2, \dots, X_N$$

each token is a vector X_n of length E

“Entertaining science fiction movie, plays in space in a distant past”

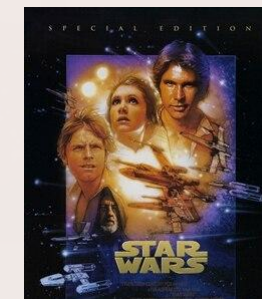
“Boring love story, plays on a ship in the past”

key

Alien



Star Wars



Titanic



value



self attention

imagine you want to built & train a movie GenAI **that creates movies based on queries.**

training data

each token is a vector X_n of length E $X = X_1, X_2, \dots, X_N$

key

Alien



value

"Boring love story, plays in space."

query





self attention

key

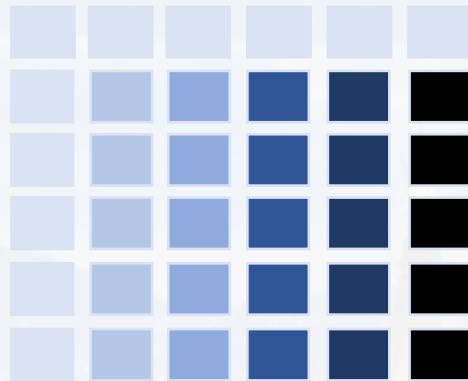
value

query

each token is a vector X_n of length E

$X = X_1, X_2, \dots, X_N$

"Boring love story, plays on a ship in the past"



weights w_{nm}

output:

$$Y_n = \sum_{m=1}^N w_{nm} X_m$$

- weights should be trainable
- weight = 0, X_m has no influence on output
- weights **should be positive** so that neg weights don't counteract positive weights
- normalization: **$\sum_{m=1}^N w_{nm} = 1$**

for returning the best suggestion:

comparing key vector X_n to query vector X_m via **dot product**

$$w_{nm} = \frac{\exp(X_n \circ X_m)}{\sum_{\mu=1}^N \exp(X_n \circ X_{\mu})} \quad \text{softmax}$$



self attention

key

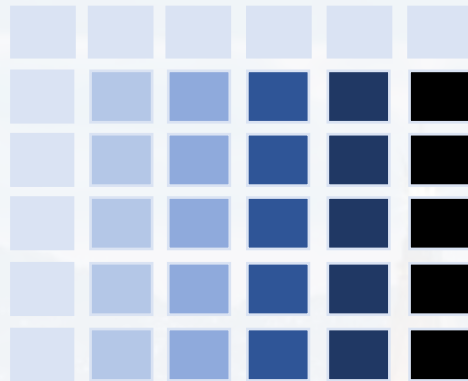
value

query

each token is a vector X_n of length E

$X = X_1, X_2, \dots, X_N$

"Boring love story, plays on a ship in the past"



weights w_{nm}

output:

$$Y_n = \sum_{m=1}^N w_{nm} X_m$$

- weights should be trainable
- weight = 0, X_m has no influence on output
- weights should be positive so that neg weights don't counteract positive weights
- normalization: $\sum_{m=1}^N w_{nm} = 1$

for returning the best suggestion:

comparing key vector X_n to query vector X_m via dot product

$$w_{nm} = \frac{\exp(X_n \mathbf{W}(\mathbf{K}) \circ X_m \mathbf{W}(\mathbf{Q}))}{\sum_{\mu=1}^N \exp(X_n \mathbf{W}(\mathbf{K}) \circ X_{\mu} \mathbf{W}(\mathbf{Q}))}$$

softmax



self attention

key

value

query

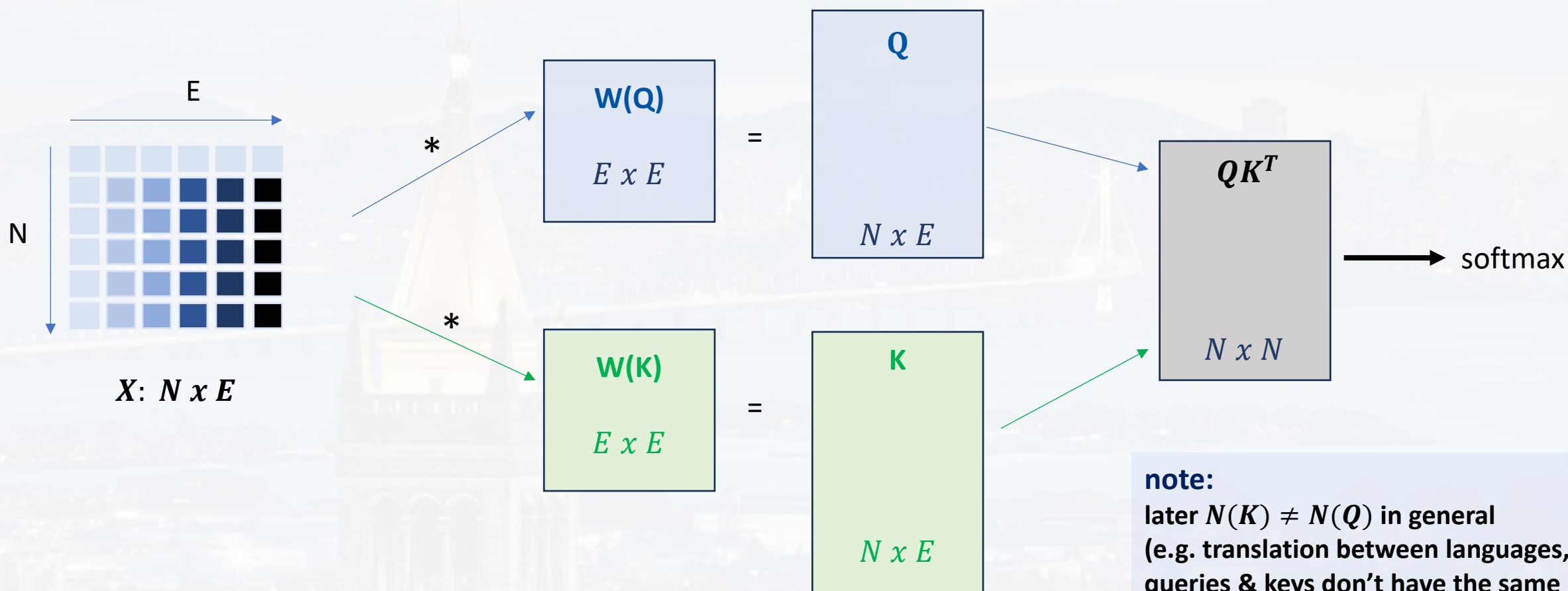
$$w_{nm} = \frac{\exp(X_n W(K) \circ X_m W(Q))}{\sum_{\mu=1}^N \exp(X_n W(K) \circ X_{\mu} W(Q))}$$

N: number of token

E: number of embedding dimensions

output: $Y_n = \sum_{m=1}^N w_{nm} X_m$

"Boring love story, plays on a ship in the past"



note:

later $N(K) \neq N(Q)$ in general
(e.g. translation between languages,
queries & keys don't have the same
number of tokens in general etc)



self attention

key value query

$$w_{nm} = \frac{\exp(\mathbf{X}_n \mathbf{W}(\mathbf{K}) \circ \mathbf{X}_m \mathbf{W}(\mathbf{Q}))}{\sum_{\mu=1}^N \exp(\mathbf{X}_n \mathbf{W}(\mathbf{K}) \circ \mathbf{X}_{\mu} \mathbf{W}(\mathbf{Q}))}$$

N: number of token

E: number of embedding dimensions

output:
$$Y_n = \sum_{m=1}^N w_{nm} X_m = \sum_{m=1}^N \frac{\exp(\mathbf{X}_n \mathbf{W}(\mathbf{K}) \circ \mathbf{X}_m \mathbf{W}(\mathbf{Q}))}{\sum_{\mu=1}^N \exp(\mathbf{X}_n \mathbf{W}(\mathbf{K}) \circ \mathbf{X}_{\mu} \mathbf{W}(\mathbf{Q}))} X_m$$

$$= \sum_{m=1}^N \text{softmax}(\mathbf{Q}_n \mathbf{K}_m^T) X_m$$

$$\rightarrow \sum_{m=1}^N \text{softmax}(\mathbf{Q}_n \mathbf{K}_m^T) X_m \mathbf{W}(\mathbf{V}) \rightarrow \mathbf{Y} = \text{softmax}(\mathbf{Q} \mathbf{K}^T) \mathbf{V} \text{ value}$$

summarizing the characteristics of the movie by using the movie itself

The output would be a movie, generated by weighted contributions of those movies (values), where the keys match well with the query



self attention

key

value

query

$$w_{nm} = \frac{\exp(X_n \mathbf{W(K)} \circ X_m \mathbf{W(Q)})}{\sum_{\mu=1}^N \exp(X_n \mathbf{W(K)} \circ X_{\mu} \mathbf{W(Q)})}$$

N: number of token

E: number of embedding dimensions

output: $Y_n = \sum_{m=1}^N w_{nm} X_m$

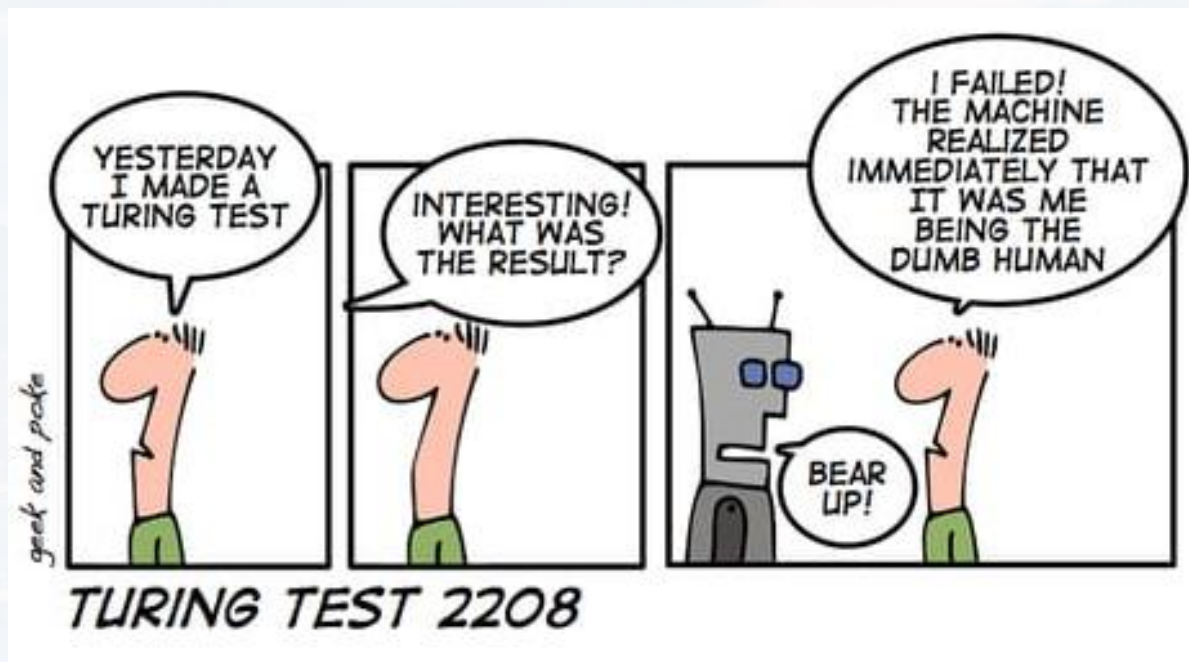
$$\rightarrow Y = \text{softmax}(QK^T)V$$

note:

- dot product scales with E , therefore normalization: $\text{softmax}\left(\frac{QK^T}{\sqrt{E}}\right)$
- cross attention:
 - eg. key a phrase in language A, query in language B
 - **encoder/decoder** structure, see next slides
- we want to recognize many underlying pattern \rightarrow **multiple attention** layers in parallel \rightarrow transformer



Outline

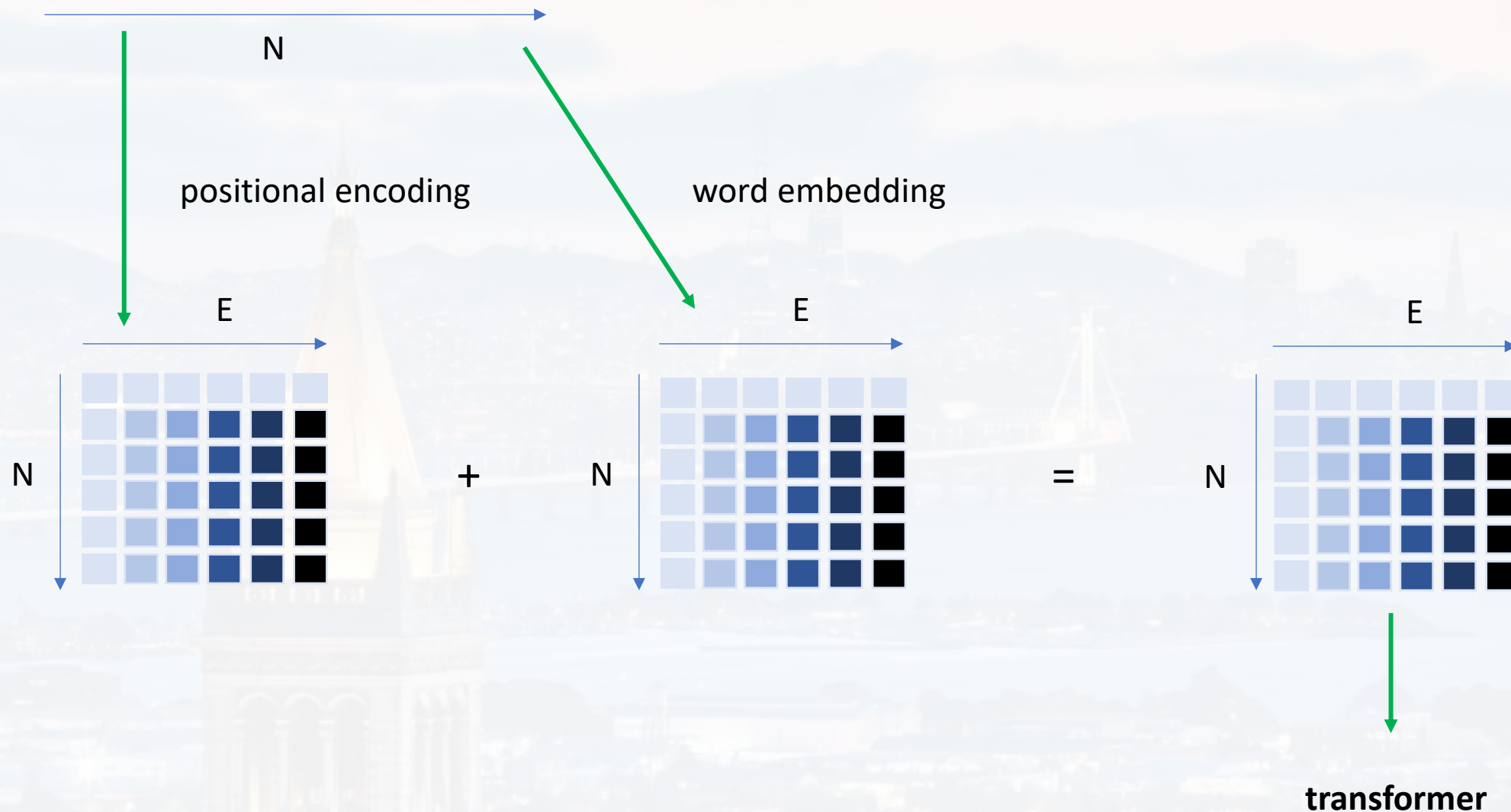


- Introduction
- Bigram and MAP
- Positional Encoding
- Word Embedding
- Attention
- **Transformer Architecture**



N: number of token
E: number of embedding dimensions

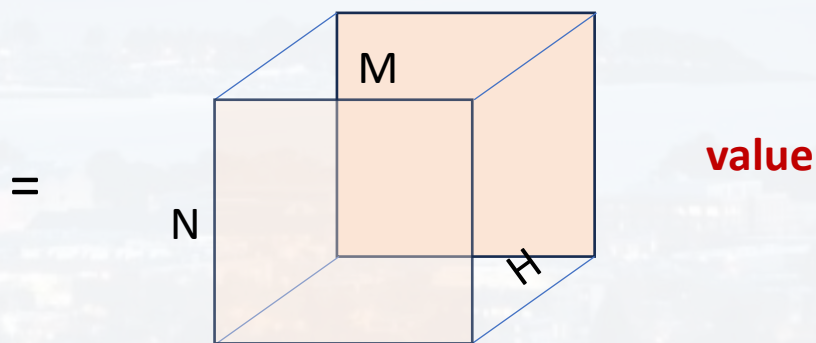
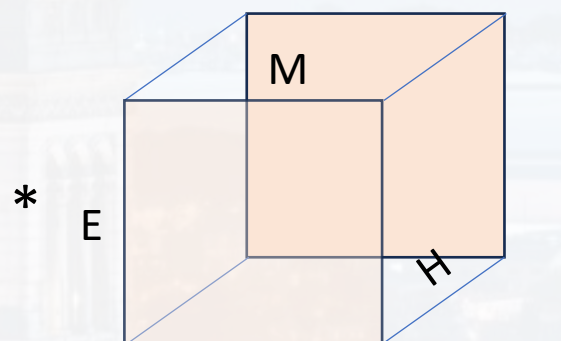
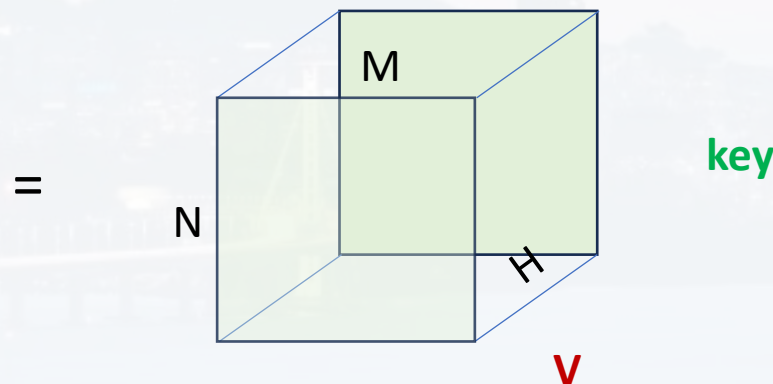
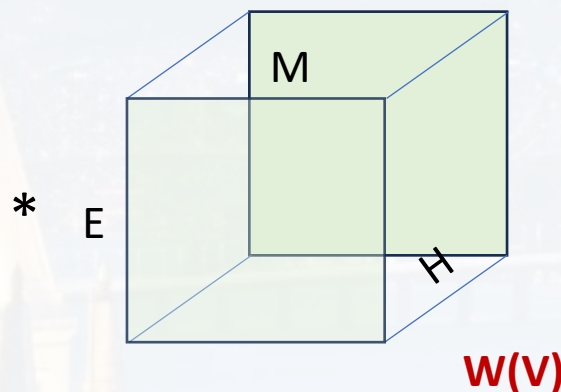
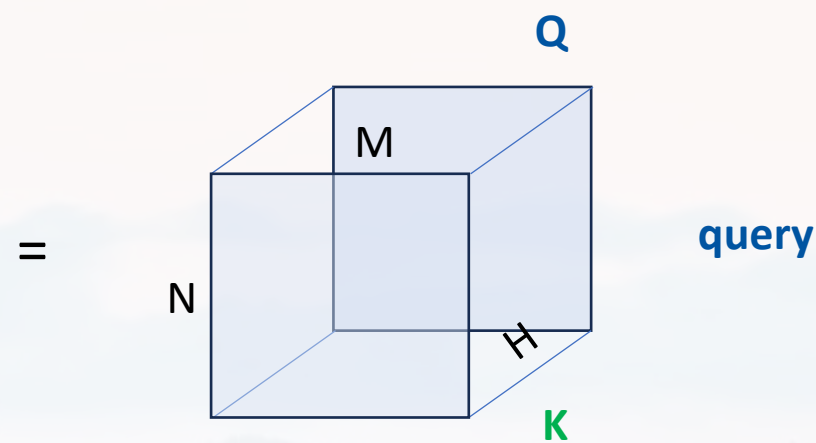
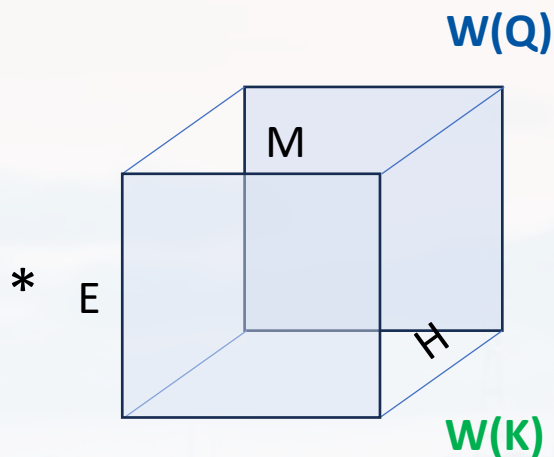
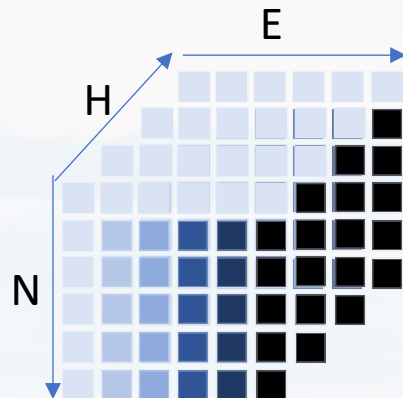
"The cat jumped on the roof."





within the transformer:

attention:



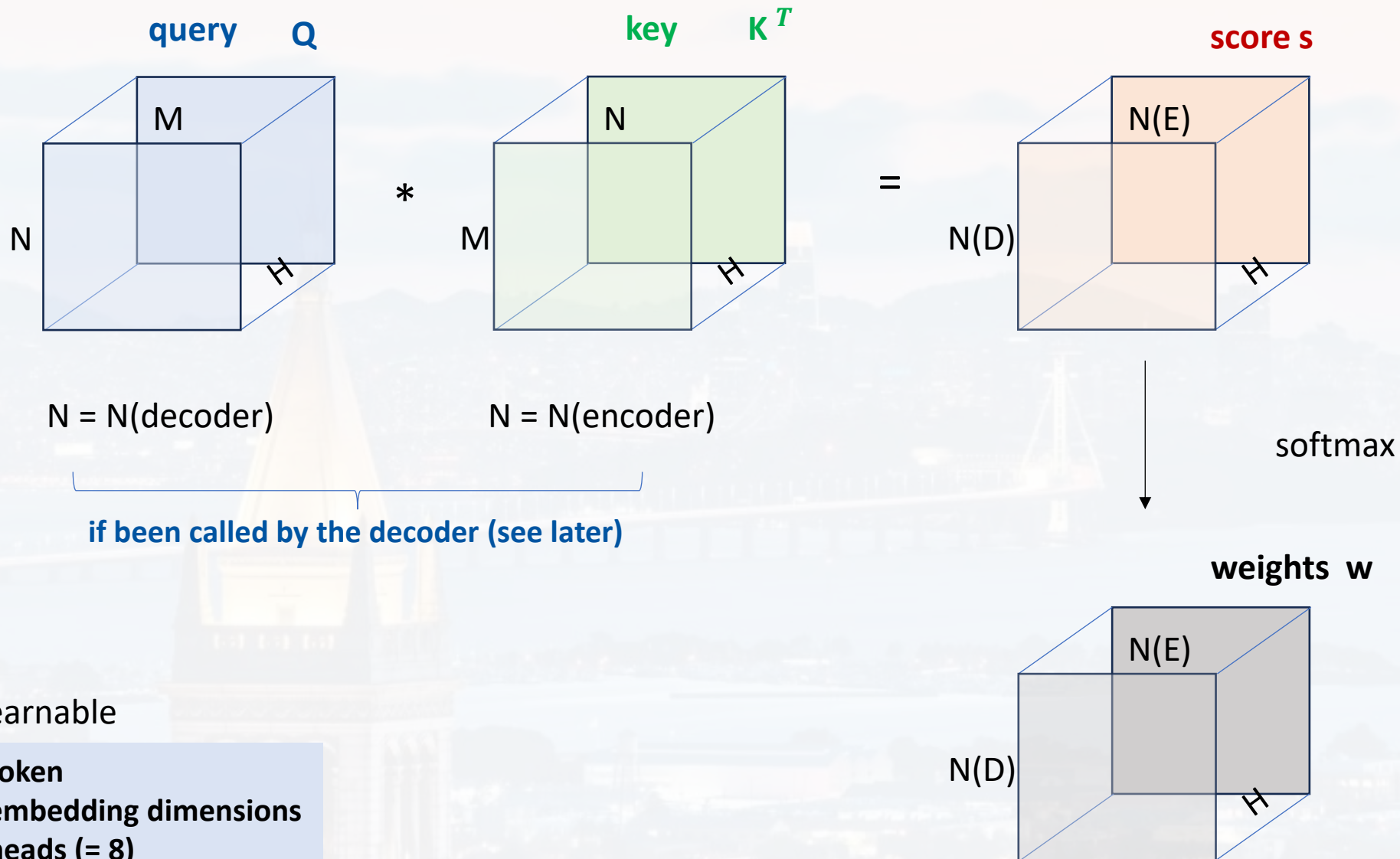
$W(Q)$, $W(K)$, $W(V)$: learnable

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



within the transformer:

attention:



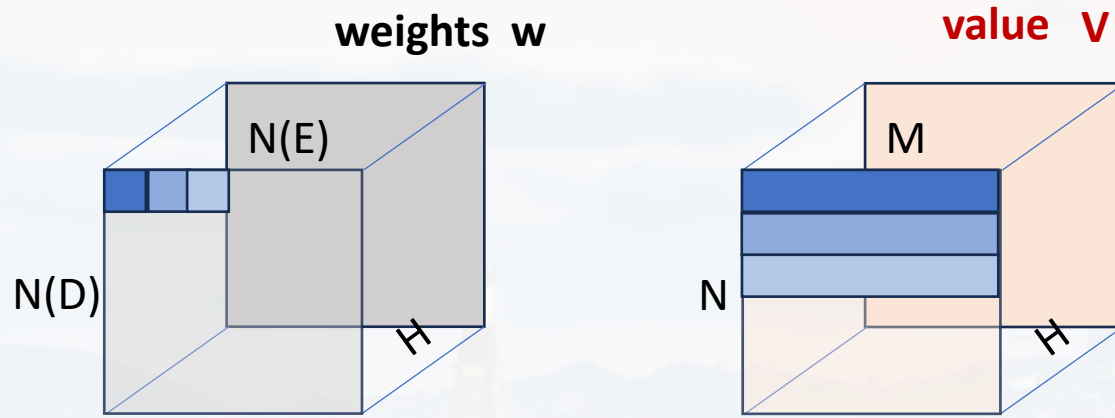
$W(Q)$, $W(K)$, $W(V)$: learnable

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



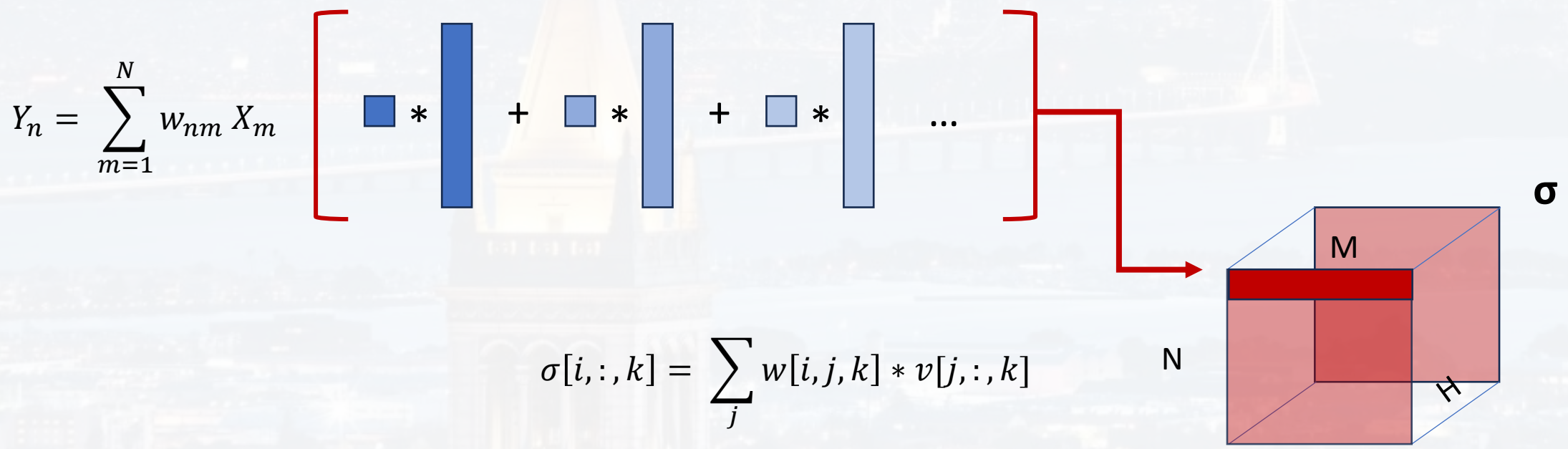
within the transformer:

attention:



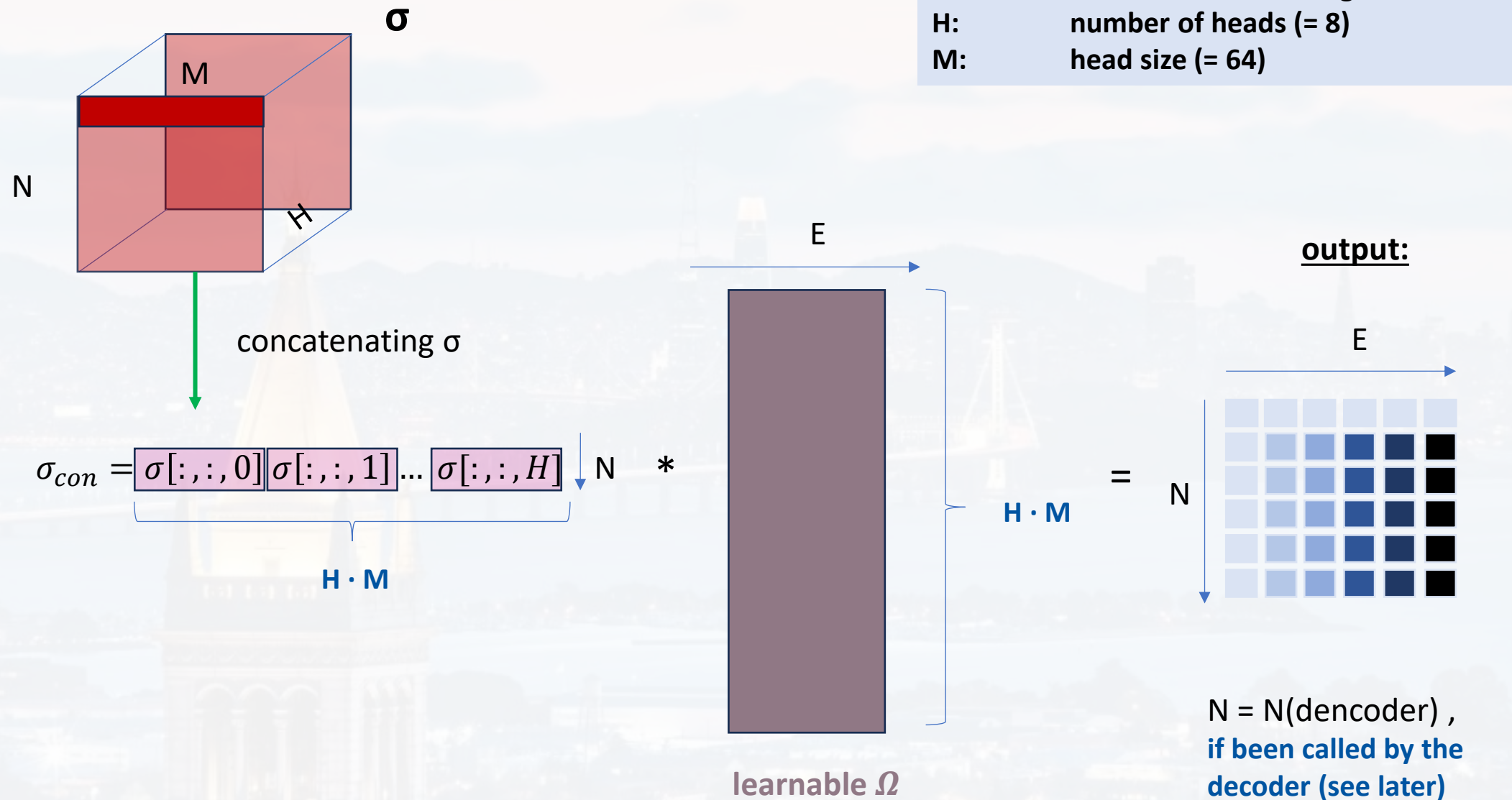
N:	number of token
E:	number of embedding dimensions
H:	number of heads (= 8)
M:	head size (= 64)

N(E) = N(encoder)
N(D) = N(decoder) , [see later](#)





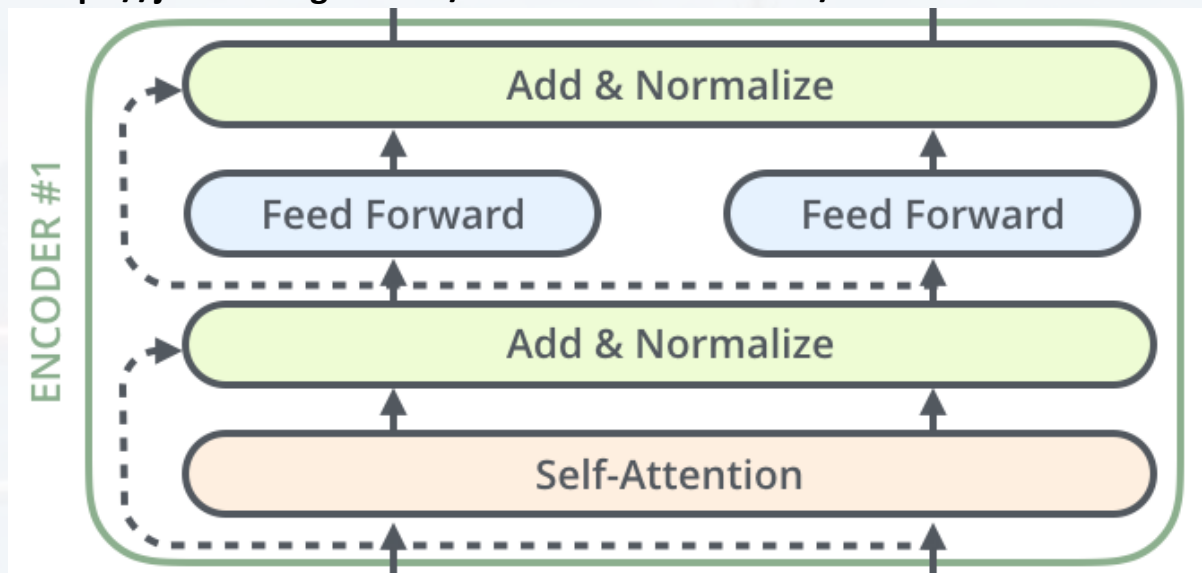
within the transformer:





that was attention → now: encoder

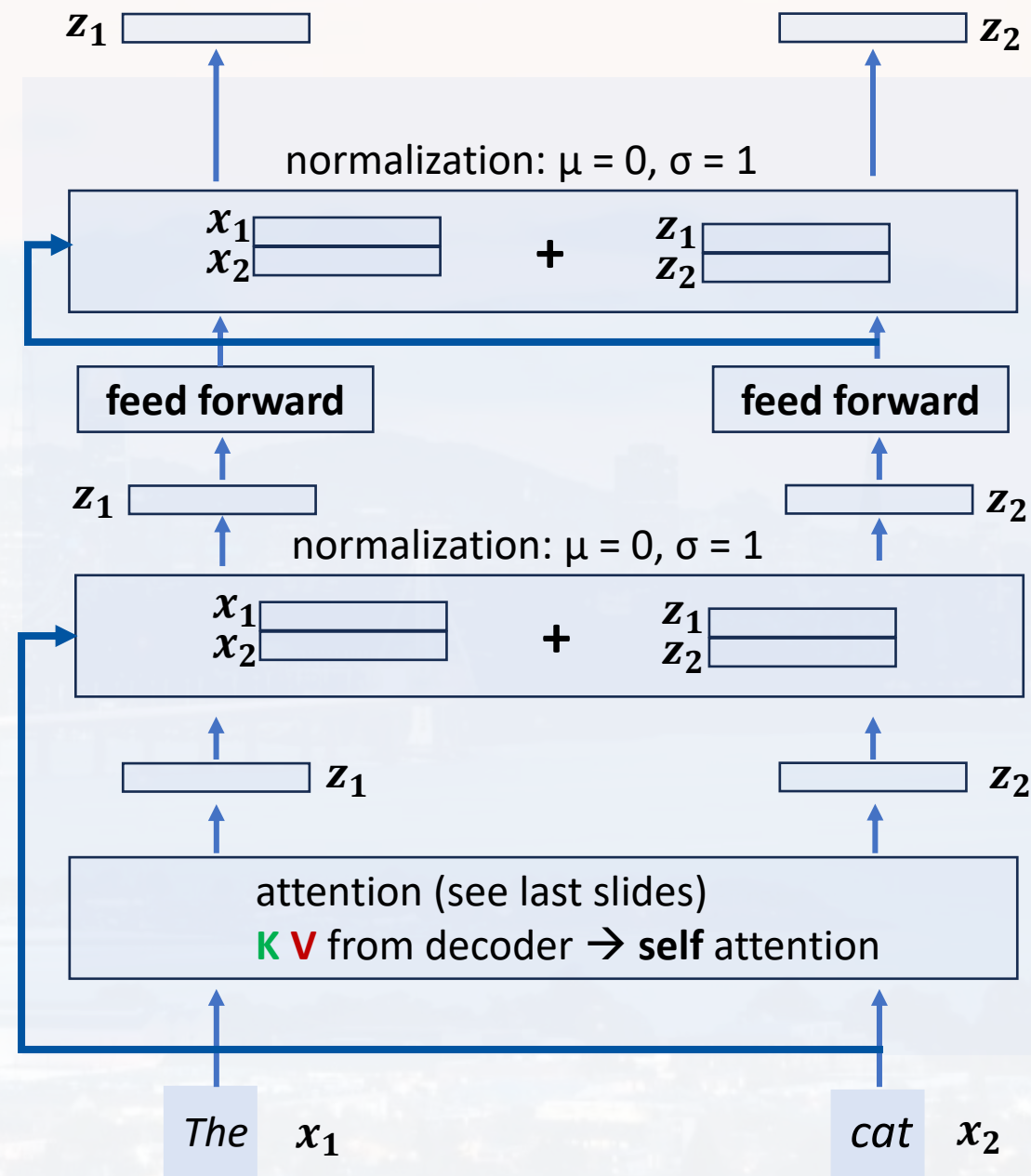
<https://jalammr.github.io/illustrated-transformer/>



positional encoding

+

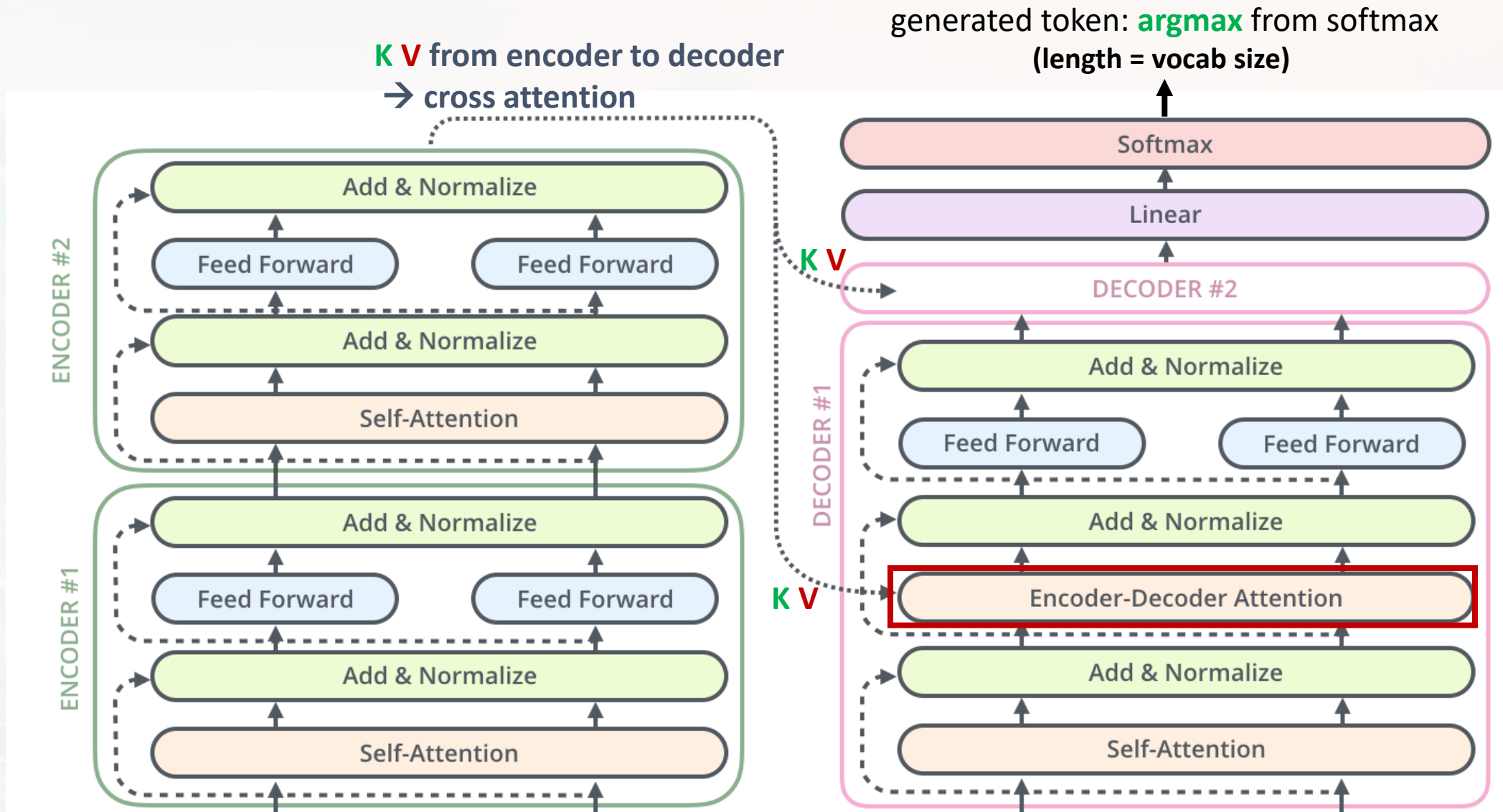
word embedding

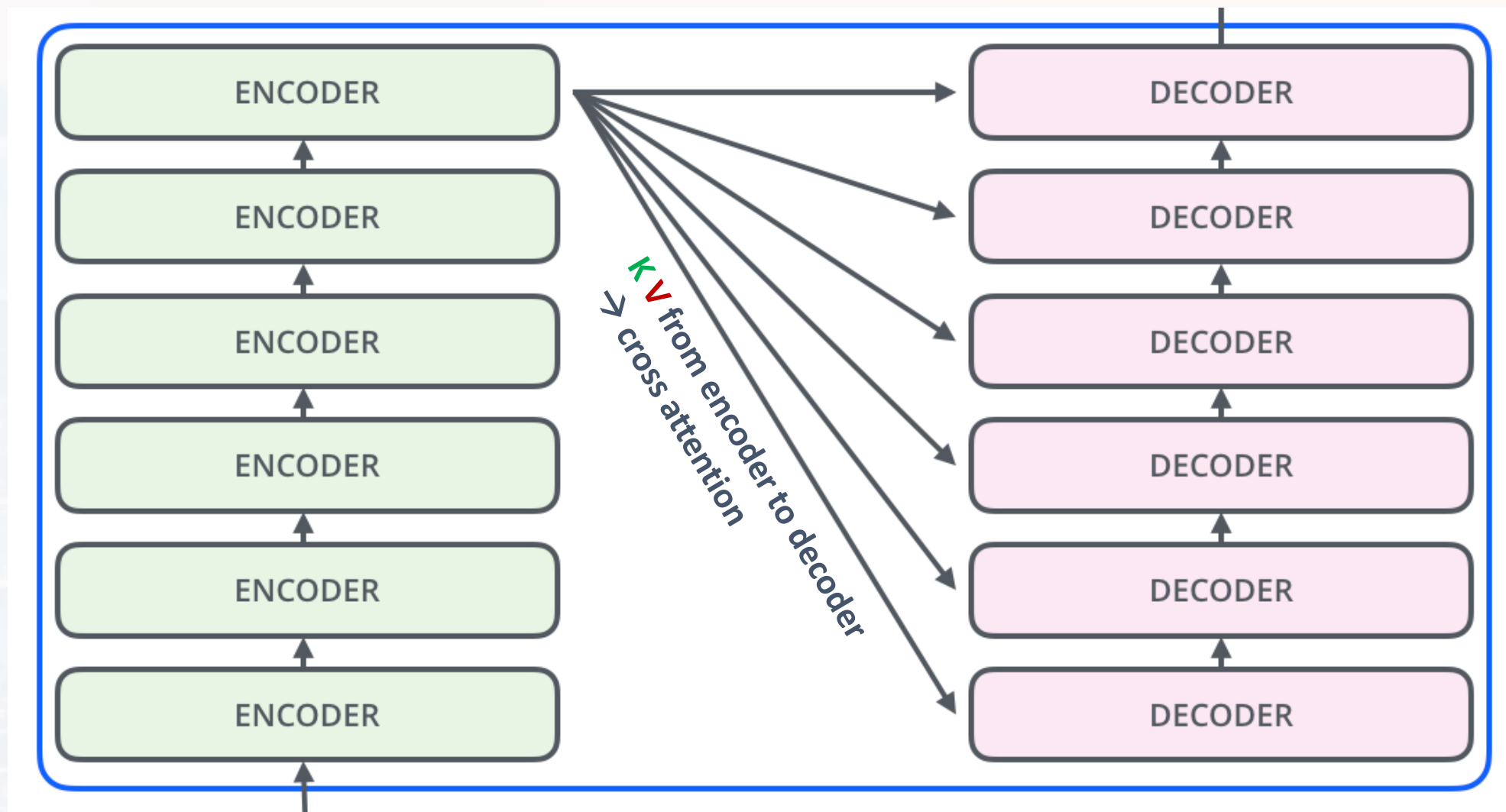




LLM & Transformer:

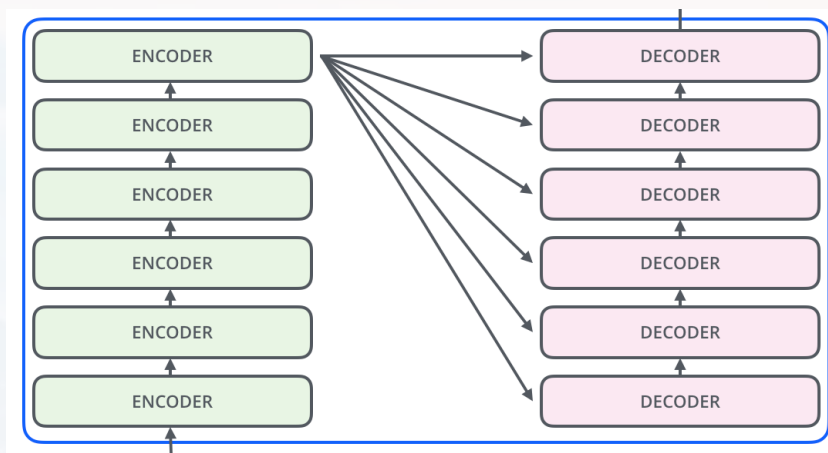
Transformer Architecture







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.

$N(D) = 3$

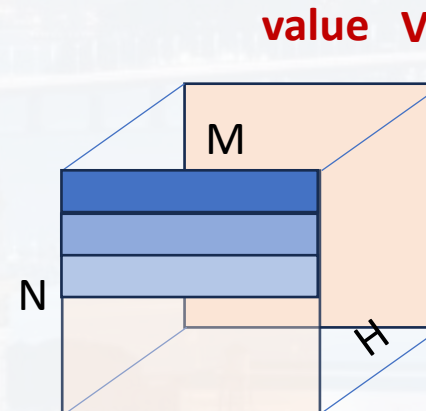
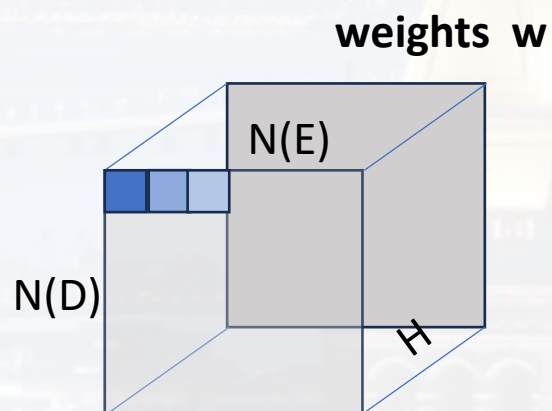
Latin: Fiat lux.

$N(D) = 2$

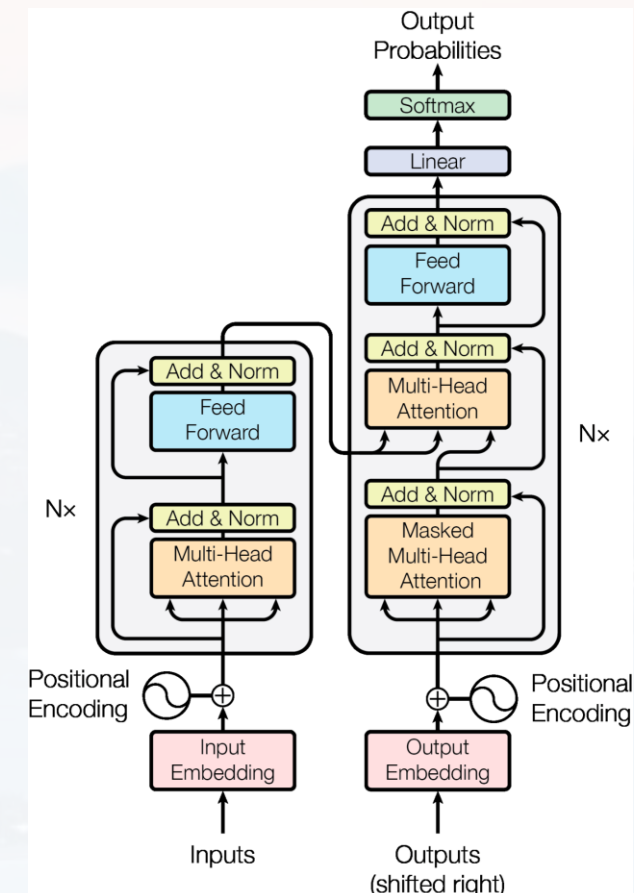
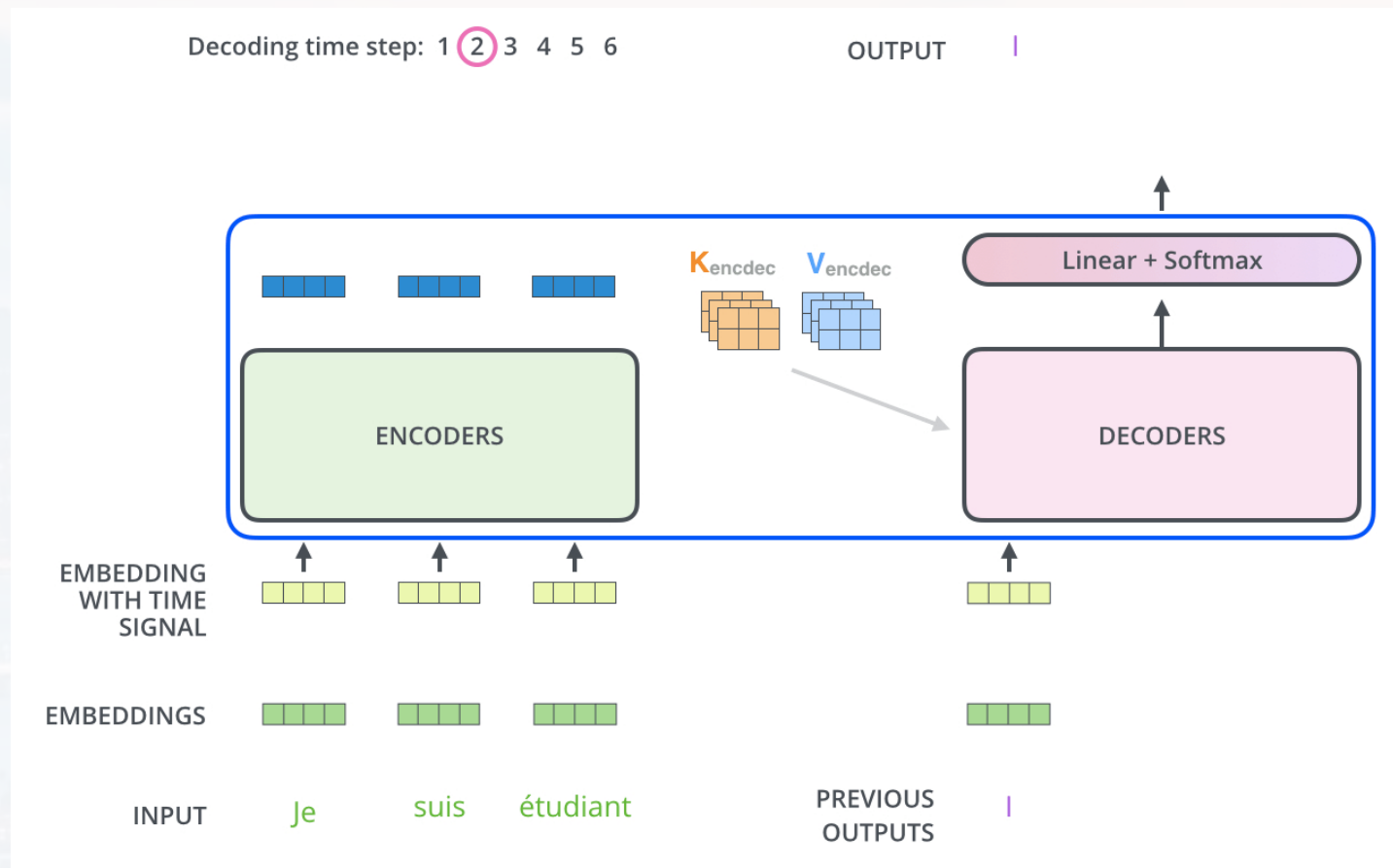
Hebrew: yehi, or

$N(D) = 2$

attention:



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



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

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



mask: \mathcal{M}

improvements:

masked attention

$$Y = \text{softmax}(QK^T)V \rightarrow Y = \text{softmax}(QK^T + \mathcal{M})V$$

causal masking: $w_{nn+k} = 0 \forall k > 0$

1	0	0	0	0	0
1	1	0	0	0	0
1	1	1	0	0	0
1	1	1	1	0	0
1	1	1	1	1	0
1	1	1	1	1	1

see left-right HMM
module 7

token at position t should only consider **previous token**
for predicting token at $t + 1$ (natural languages)

padding masking: batches of sequences might have different lengths,
→ shorter sequences are padded with special tokens.
→ model learns to ignore padding tokens
→ for inference

individual masking: we often know that some token can't appear after each other
(natural languages)



improvements:

sampling strategies

vanilla:

returning most **probable token**, from which we calculate the probabilities for the next token and return the most likely one etc

beam search:

we store b (= **beam width**) sequences of length n and then return the **most likely sequence**

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

top K-sampling:

consider **K most probable token**

→ renormalize their probabilities

→ draw randomly from these K token

$$p_k = \frac{\exp(\pi_k/T)}{\sum_{k=1}^K \exp(\pi_k/T)}$$

π_k :

probability

p_k :

renormalized probability

T :

“softening” parameter

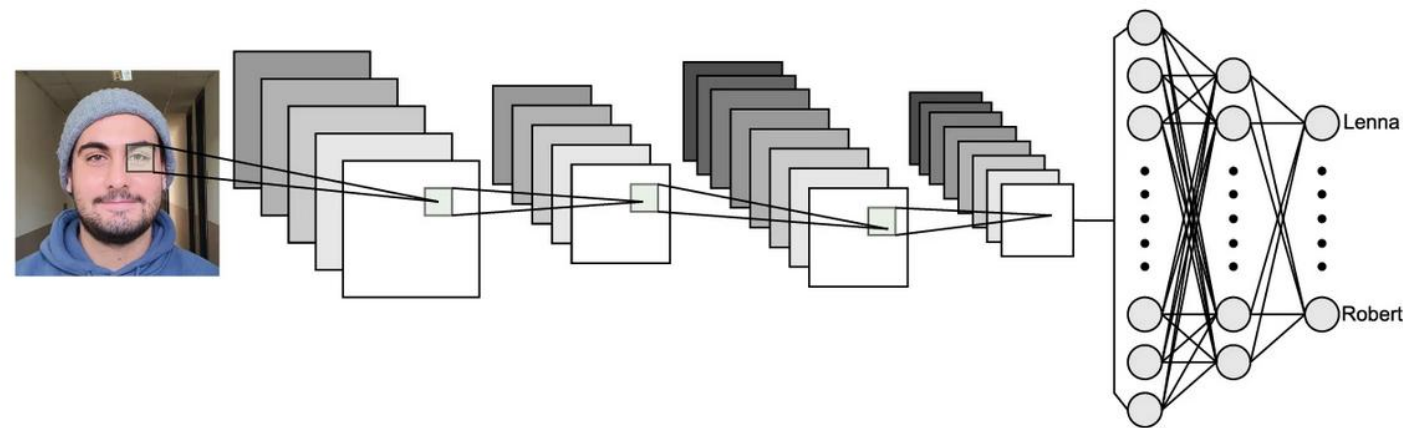
top P-sampling:

like top K , but for sequences (see beam search)

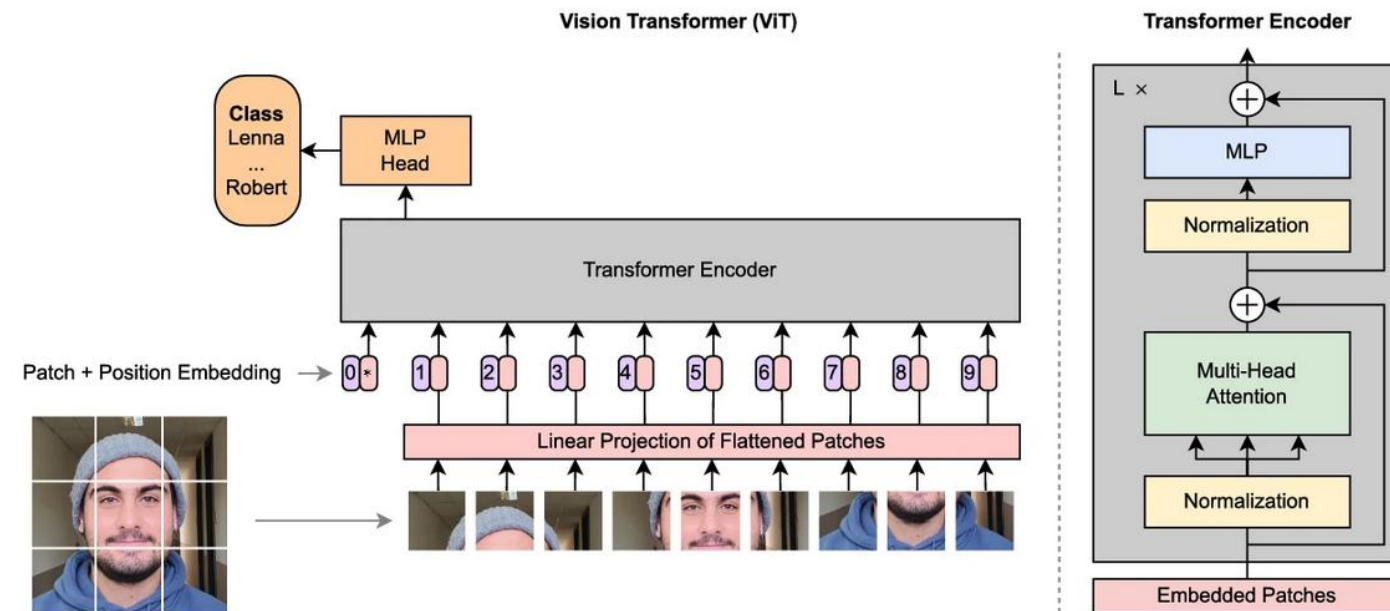


improvements:

vision transformer



(a) Common CNN architecture





more about transformers:

[Jay Alammar](#)

[Interactive Visualization](#)

[transformers intro](#)



Misra Turp

@misraturp · 40.3K subscribers · 163 videos

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

misraturp.com/roadmap and 3 more links

Subscribe

[building GPT from scratch!](#)



Andrej Karpathy

@AndrejKarpathy · 451K subscribers · 14 videos

FAQ ...more

karpathy.ai and 2 more links

Subscribe

Thank you very much for your

