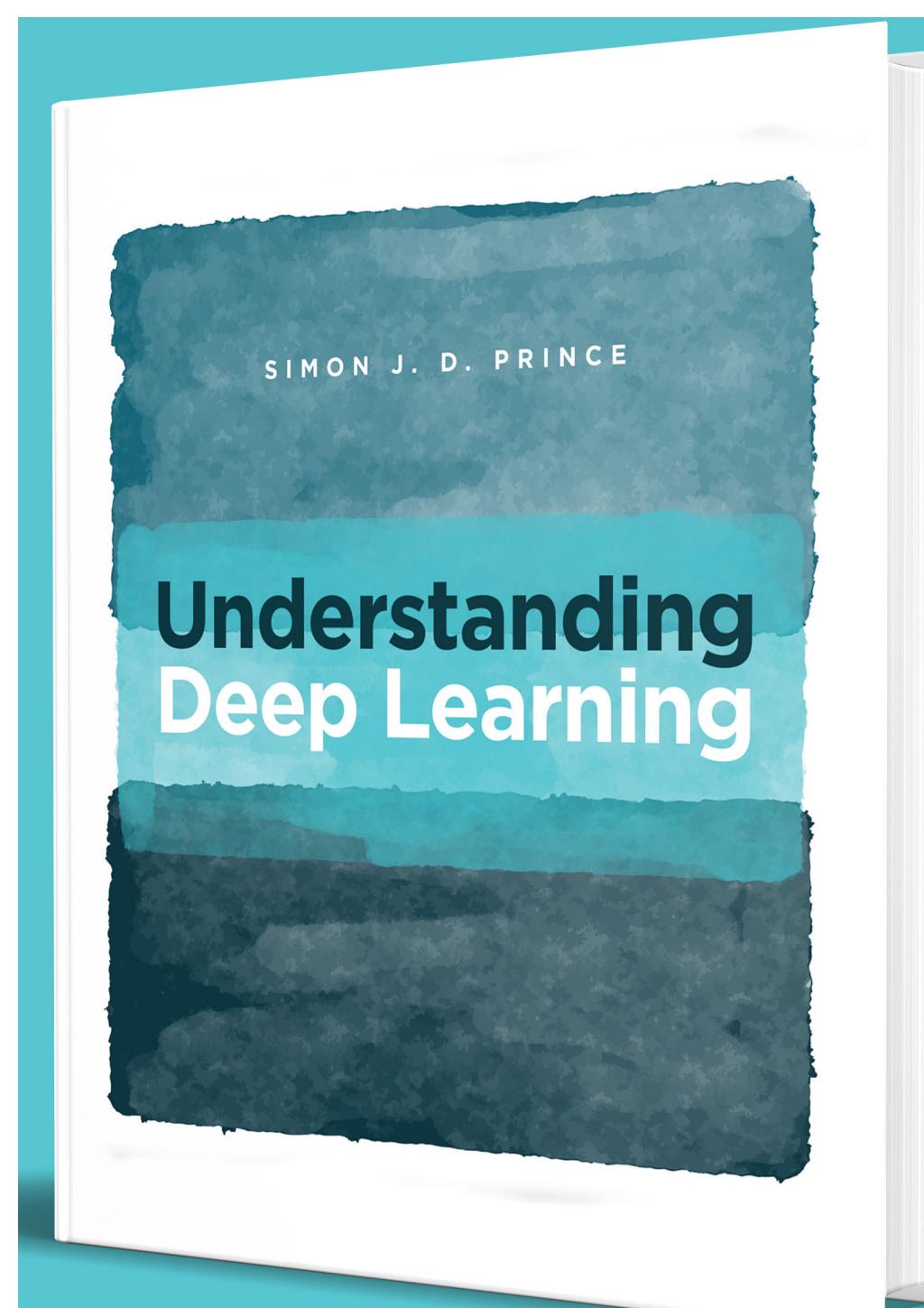




Lecture 7: Transformers

Program for Today

12 Transformers	207
12.1 Processing text data	207
12.2 Dot-product self-attention	208
12.3 Extensions to dot-product self-attention	213
12.4 Transformer layers	215
12.5 Transformers for natural language processing	216
12.6 Encoder model example: BERT	219
12.7 Decoder model example: GPT3	222
12.8 Encoder-decoder model example: machine translation	226
12.9 Transformers for long sequences	227
12.10 Transformers for images	228
12.11 Summary	232



Date	Topic	Tutorial
Sep 16		Intro & class description
Sep 23	probability Theory + Basic of machine learning + Dense NN	<i>Basic jupyter + DNN on mnist (give jet dnn as homework)</i>
Sep 30	Statistics + Convolutional NN	<i>Convolutional NNs with MNIST</i>
Oct 7	Training in practice: regularization, optimization, etc	<i>Free Practice</i>
Oct 14	Unsupervised learning and anomaly detection	<i>Autoencoders with MNIST</i>
Oct 21	Graph NNs	<i>Tutorial on Graph NNs</i>
Oct 28		Transformers
Nov 4		Network compression (pruning, quantization, Knowledge Distillation)
Nov 11		<i>Tutorial on hls4ml</i>
Nov 18	Generative models: GANs, VAEs, etc	<i>TBD</i>
Nov 25	Normalizing flows	<i>TBD</i>
Dec 2		To Be Decided
Dec 9		Quantum Machine Learning
Dec 16		Exams To Be Confirmed

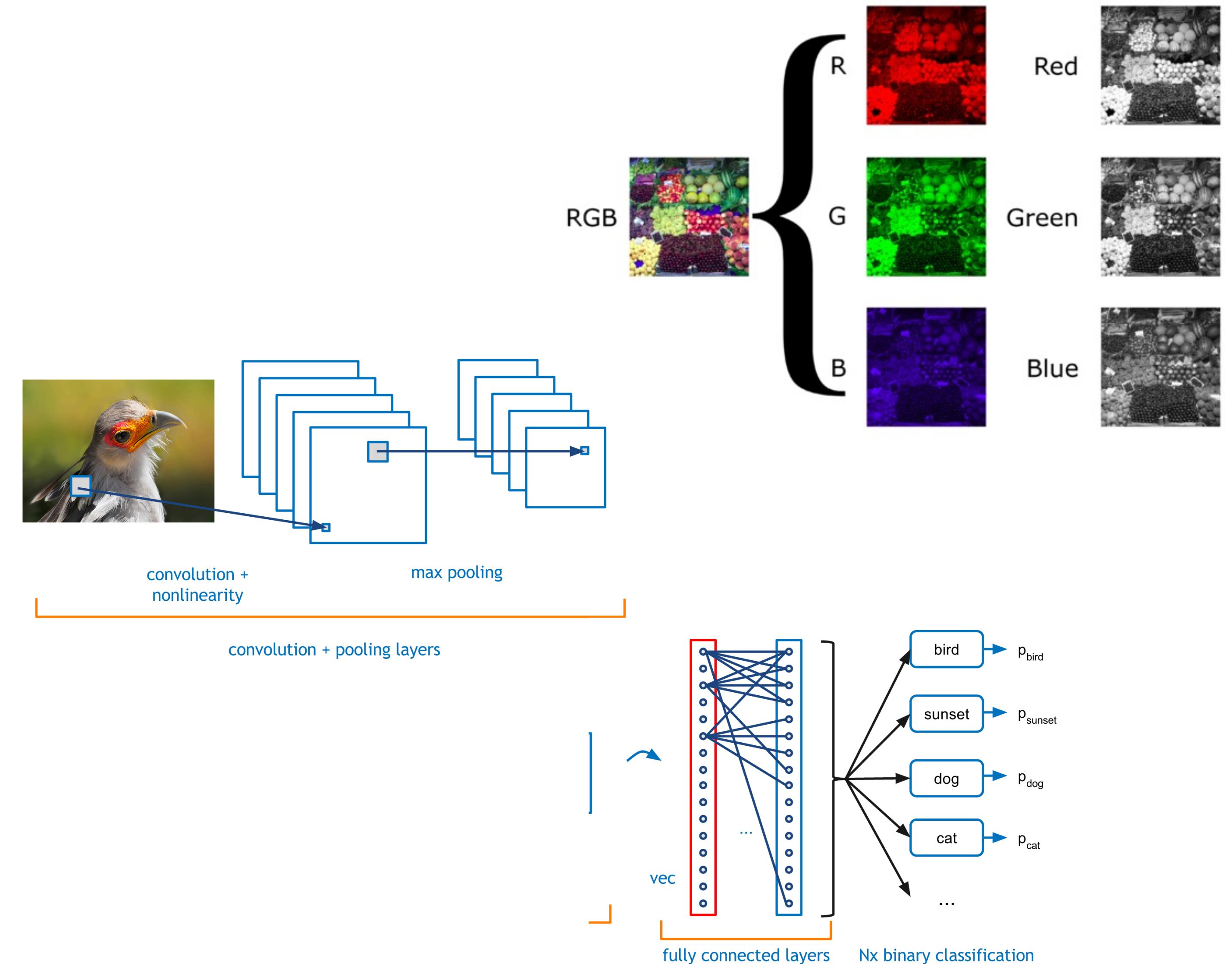


NLP and Transformers

- *The success of Deep Learning (DL) has been mainly driven by*
- *CNNs for image processing*
- *RNNs for Natural Language Processing (NLP)*
- *Other architectures have found successful applications in specific domains*
- *Graph NNs in Physics Chemistry etc.*
- *Transformers gave a huge impact to DL-driven revolution in AI*
- *Marked a breakthrough in NLP and beyond*

Processing Images

- Turn input data into a numerical representation
- Build an abstract representation of the data
- Exploit this abstract representation for task solving





Processing text

- Similarly, with text, we need to
 - Convert text into numbers
 - Map these numerical inputs to some abstract representation of its meaning

The restaurant refused to serve me a ham sandwich because it only cooks vegetarian food. In the end, they just gave me two slices of bread. Their ambiance was just as good as the food and service.

- Use this abstract representation to answer specific questions

Does the restaurant serve steak?

One-hot encoding

- We encountered one-hot encoding to turn ground truth classes to numbers in classification problems

- One can use the same approach for input text

- ⦿ This assumes that one has a dictionary with all possible inputs (bag of words)

- Could be words, could be word fragments

Human-Readable

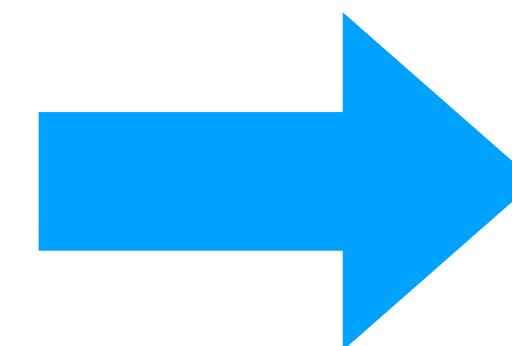
Machine-Readable

Pet	Cat	Dog	Turtle	Fish
Cat	1	0	0	0
Dog	0	1	0	0
Turtle	0	0	1	0
Fish	0	0	0	1
Cat	1	0	0	0

I [1,0,0,0,0,0,0]

like [0,1,0,0,0,0,0]

the [0,0,1,0,0,0,0]



"I LIKE THE NEW MOVIE!" → I L I K E T H E N E W M O V I E D O V E W E O T H E R

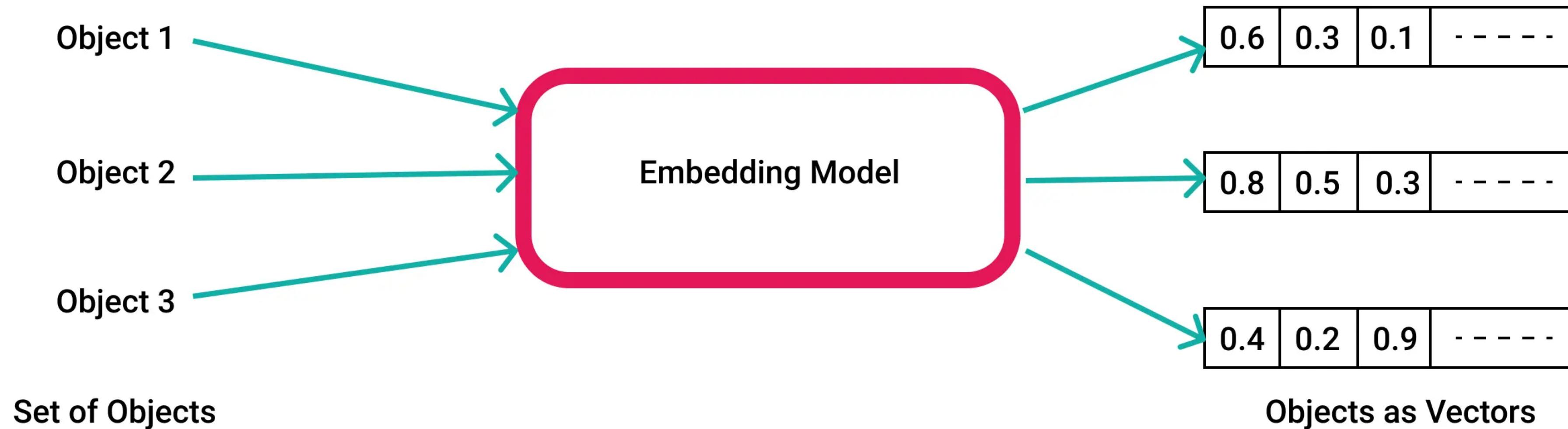
"I LOVE THE WEATHER!" → I LOVE THE NEW MOVIE LOVE WEATHER

"THE MOVIE-LIKE WEATHER..." → 0 1 1 E T H N O M V I E 0 V E W 1 A T H E R

"I LOVE THE MOVIE." → 1 0 K E T H E N W M I O V I E 0 O V E W 0 E H T R

Vector Embedding

- This is a (rule-based) example of **Vector Embedding**
- Turns “tokens” (e.g., words or word fragments) into fixed-size vectors
- Ideally, similar text should be turned into nearby vectors (semantic similarity retained as proximity)



Examples of Embeddings

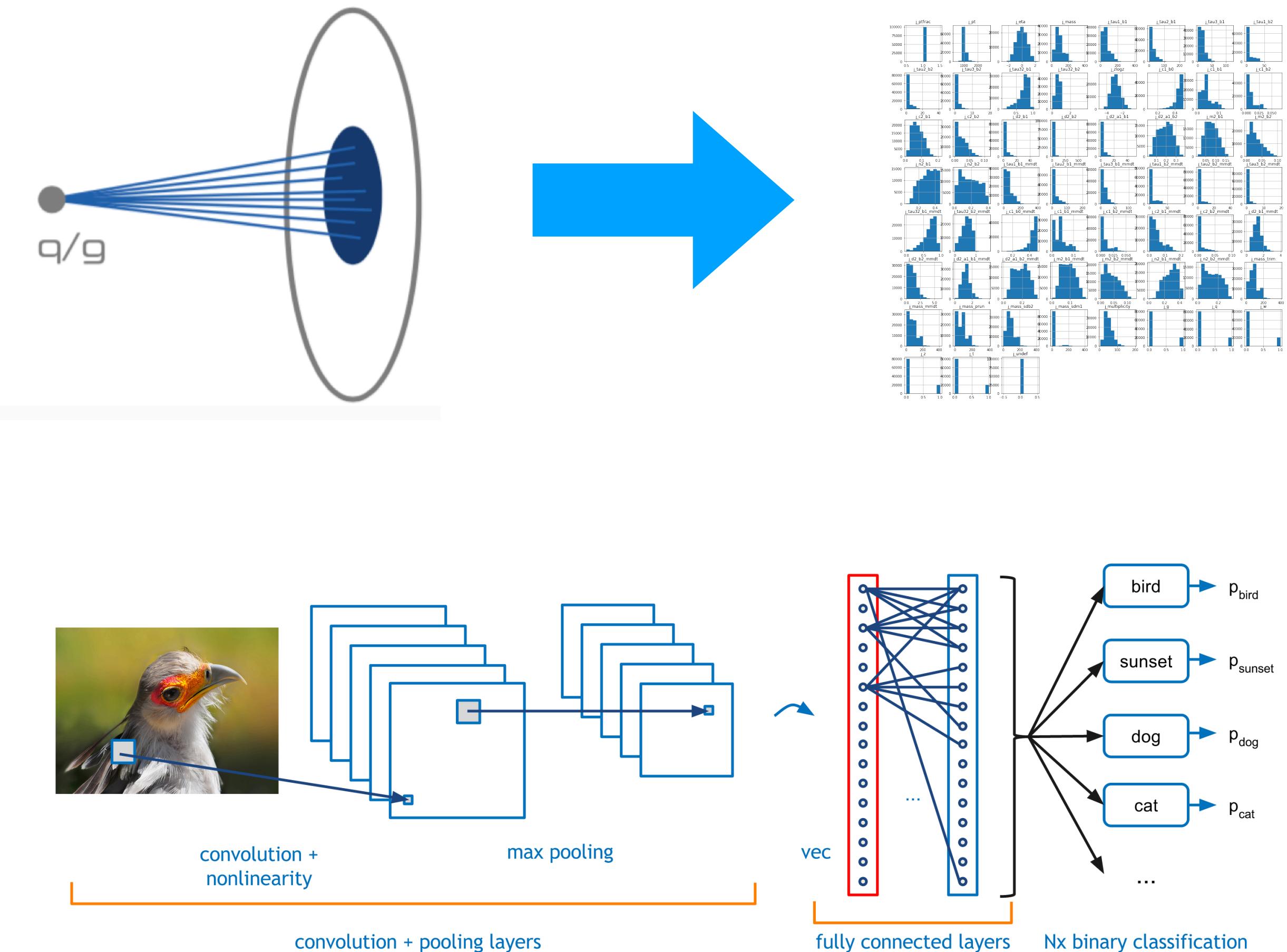
- *Embedding is a broader topic than NLP applications*

- *One could use domain knowledge and engineer high-level features by hand*

- *Example: one can use physics knowledge to transform a jet (shower of particles at LHC) into an array of numbers*

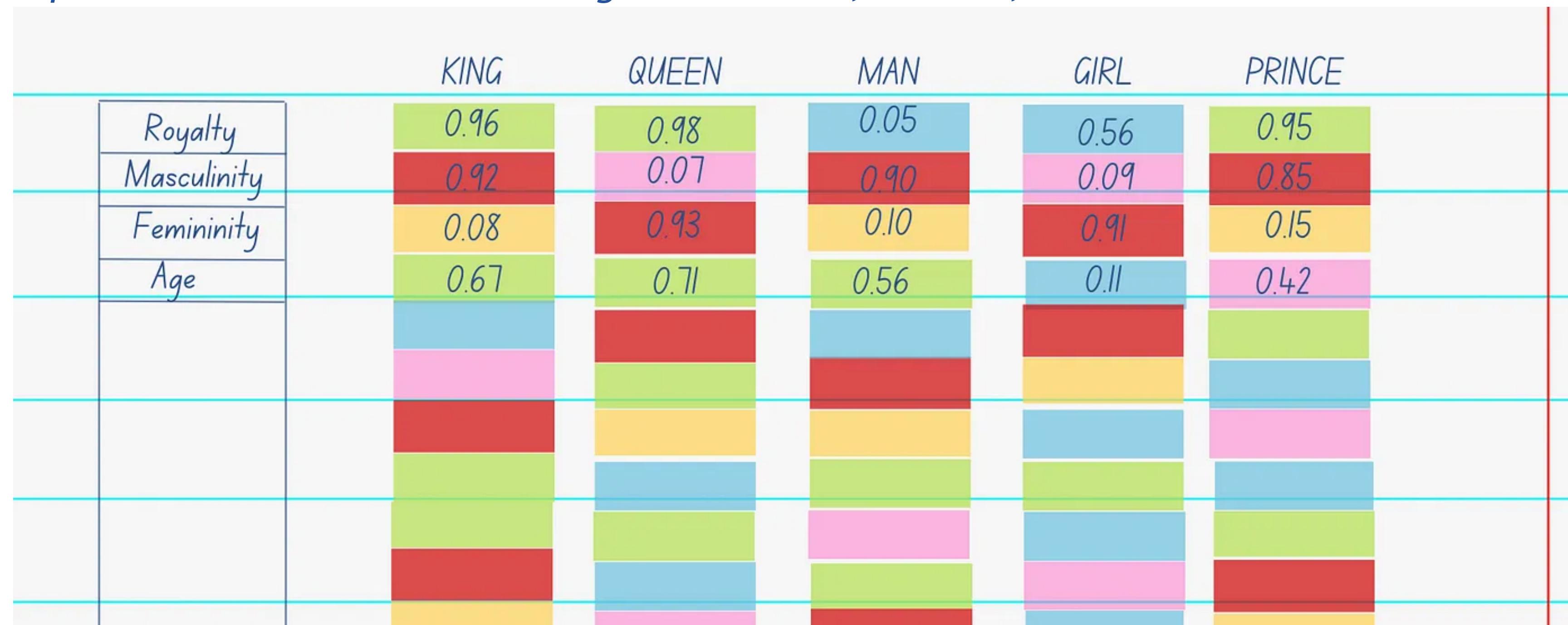
- **LEARNING EMBEDDING:** One could use a Deep Learning model

- *We saw the example of CNNs for image classification*



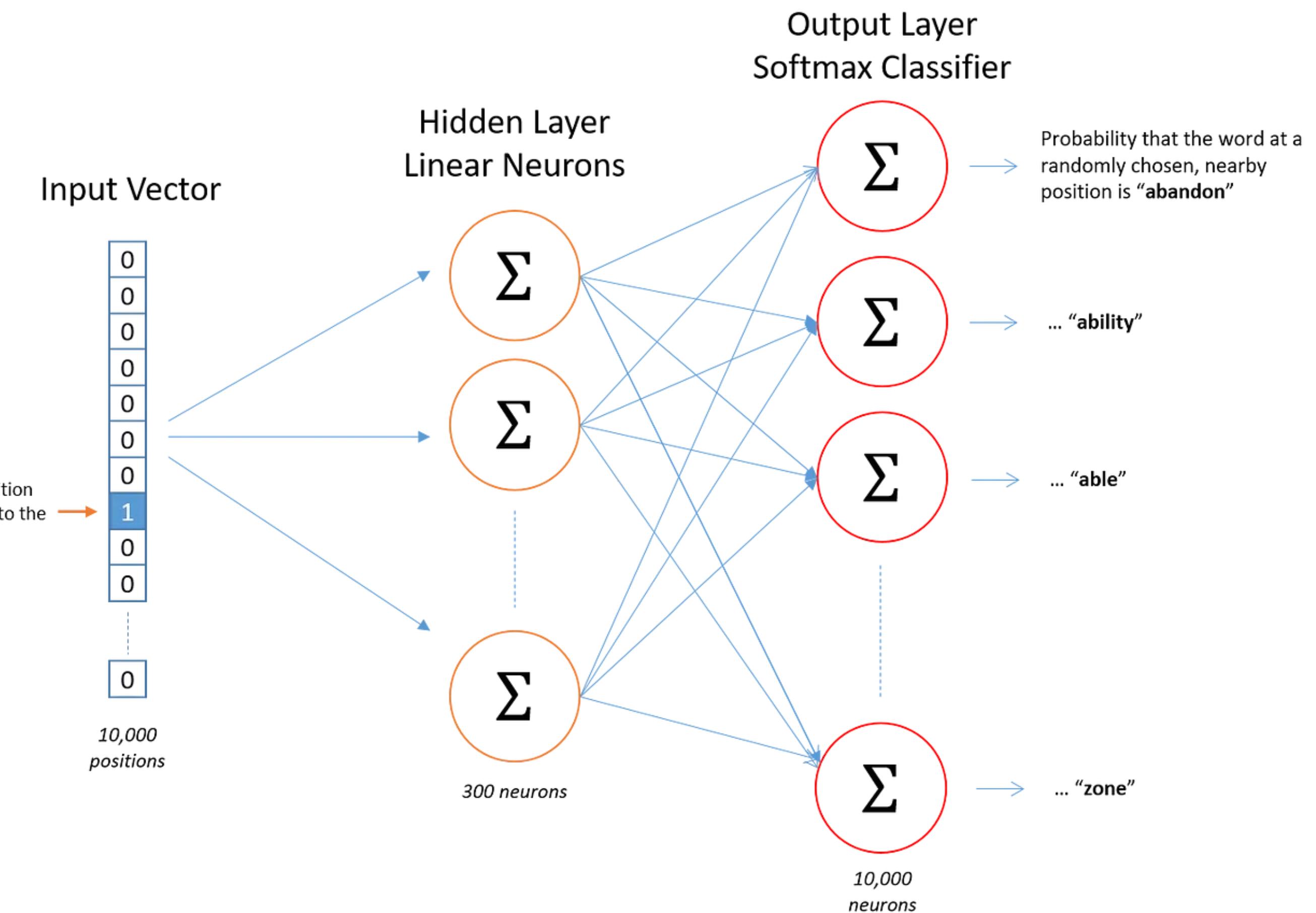
Learned Embedding

- One can go beyond this, using NNs to learn a vectorial representation of the sentence
 - A given input sentence is projected into a vector of [0,1] numbers: the larger the number, the closer the input word is to the dictionary word
 - No need to have a complete dictionary as long as the dictionary is reach enough (unlike bad of words)
 - Several examples of learned embedding: Word2Vec, GLoVE, BERT



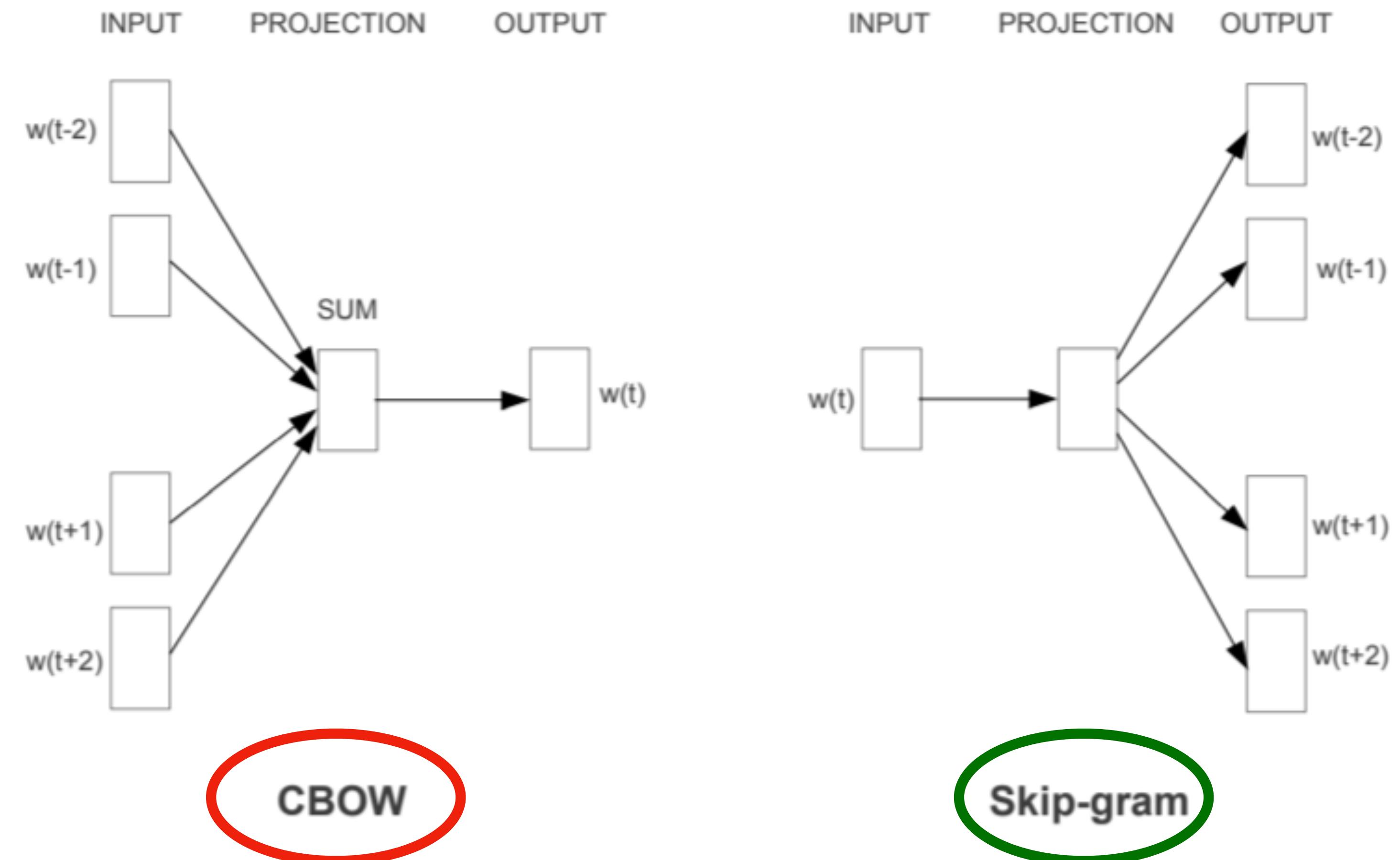
Word2Vec

- Starts with a Bag-Of-Words representation of a word as a N -vector of $N-1$ 0s and one 1
- Shallow neural network with N output neurons



Word2Vec

- Two training strategies:
- Continuous Bag of Words (**CBOW**) predicts a target word taking as input the nearby words in the sentence
- Skip-gram: predicts the other words in the sentence taking as input the target word
- In both cases, one matches a given word to the semantic affinity of the words in a dictionary
- Strategies designed so that the training ground truth can be defined without need of human intervention





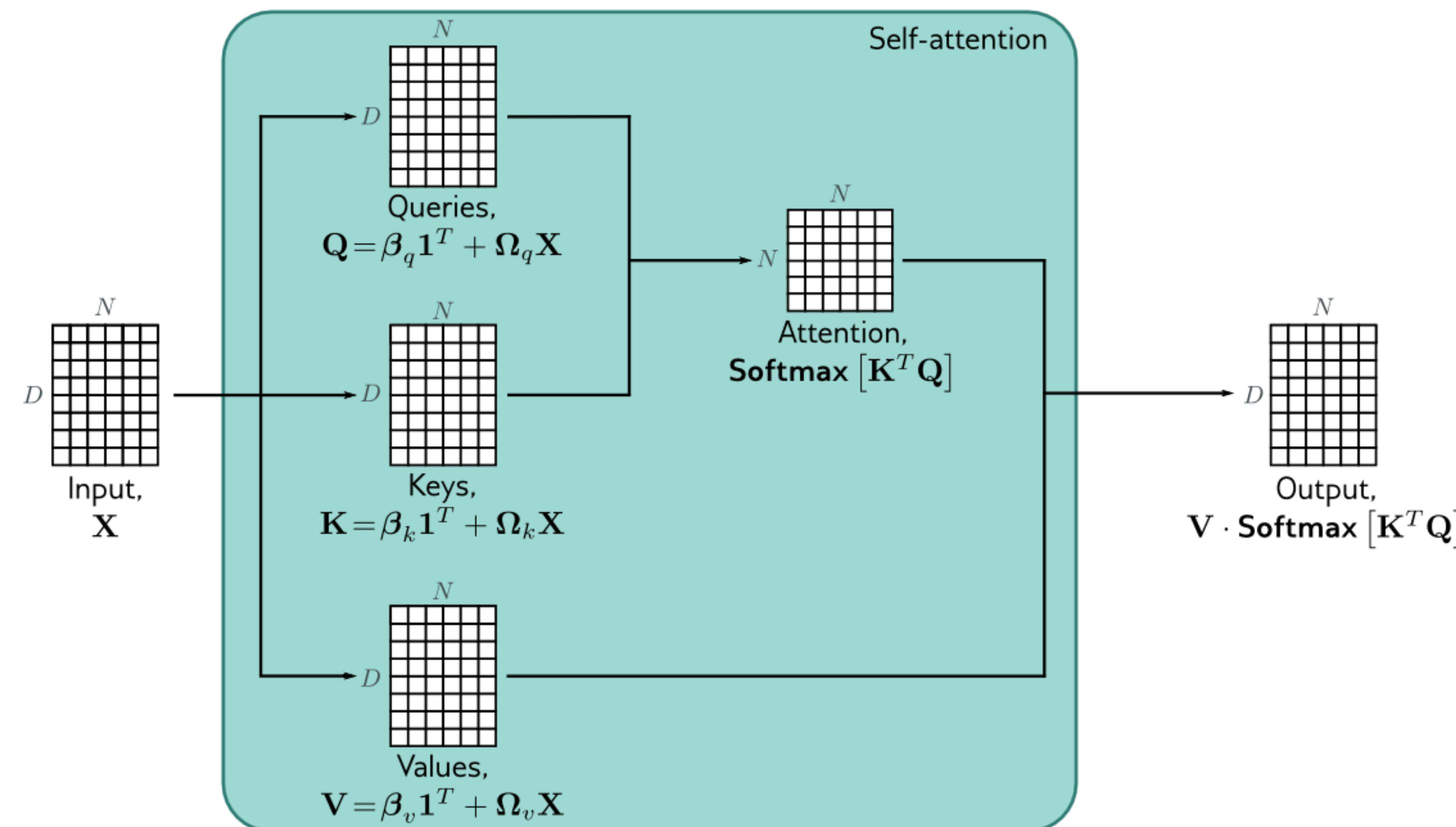
Issues with Word Encoding

The restaurant refused to serve me a ham sandwich because it only cooks vegetarian food. In the end, they just gave me two slices of bread. Their ambiance was just as good as the food and service.

- *The dimensionality of the problem grows quickly*
- *In our example there are 37 words*
- *If we want to turn each word into a 1024-size vector (one-hot, Word2Vec, ...) we end up with 37888 values*
- *The input length is variable (even at fixed representation, number of words change per sentence)*
- *We need to have some per-word processing, enforcing parameter sharing to process text without making the problem complexity explode (remember DNN vs CNN)*
- *Text is ambiguous looking at syntax alone*
- *There must be some connection between words (**attention**)*

Dot-product self attention

- Dot-product self attention is a construction that addresses these issues exploiting a mechanism to learn the connections between words



Dot-product self attention

- A layer in a dense neural networks takes an input vector x of dimension D and returns $f(\beta + \Omega x)$ where f is the activation function, Ω is the matrix of vectors, and β the vector of biases

- A self-attention block $sa[]$ takes as input N D -dim vectors x_1, x_2, \dots, x_N (words) and returns N D -dim outputs

- Compute **VALUES** $v_m = \beta + \Omega x_m$

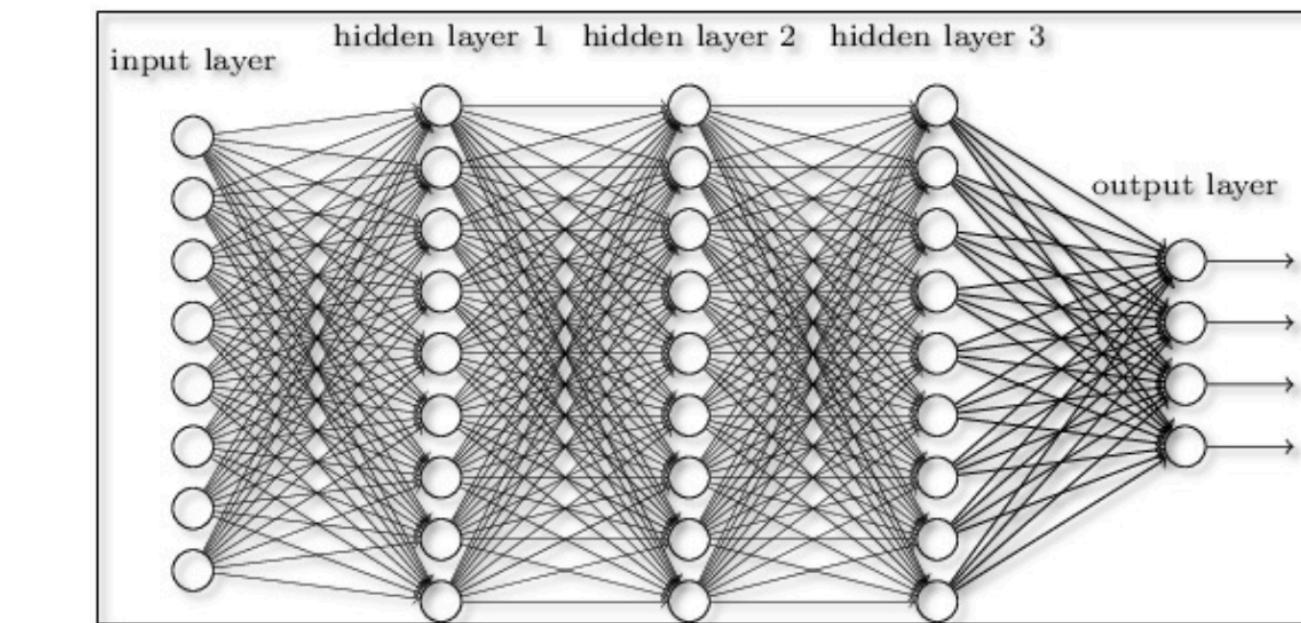
- Return a weighted average

$$sa_n[x_1, \dots, x_N] = \sum_{m=1}^N a[x_m, x_n] v_m$$

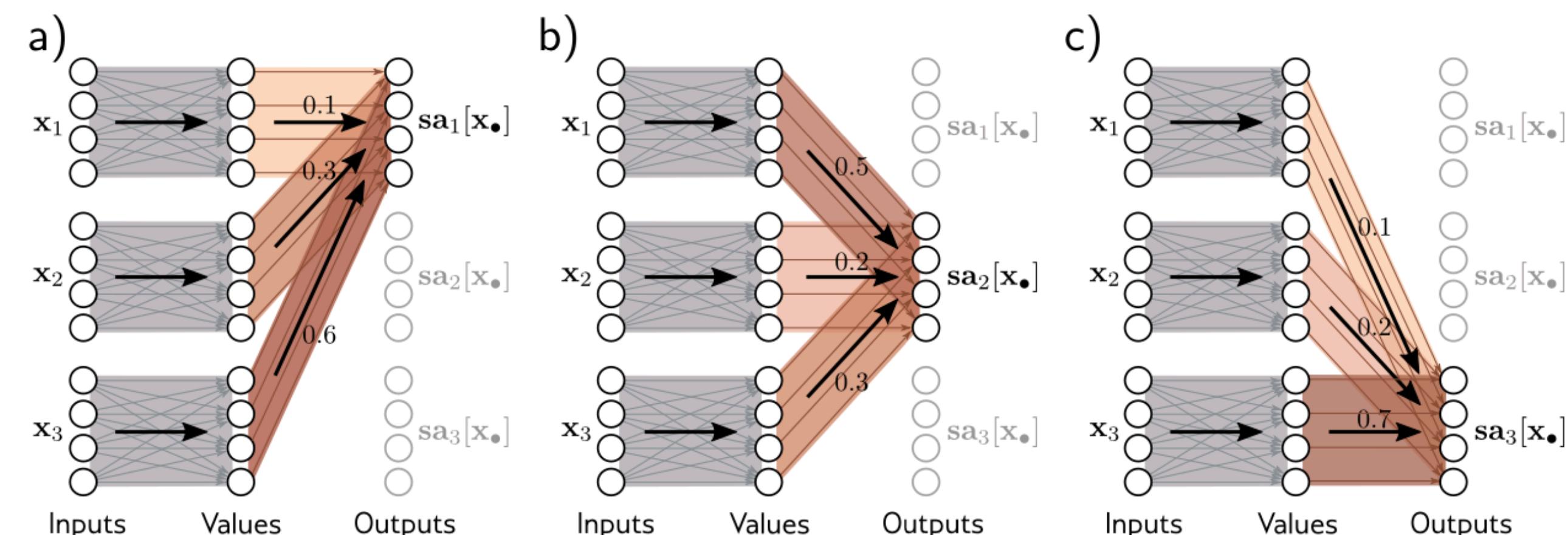
- The matrix of weighs is the attention matrix
 $a[x_m, x_n]$

- $a[x_m, x_n] \geq 0$

- $\sum_{n,m} a[x_m, x_n] = 1$



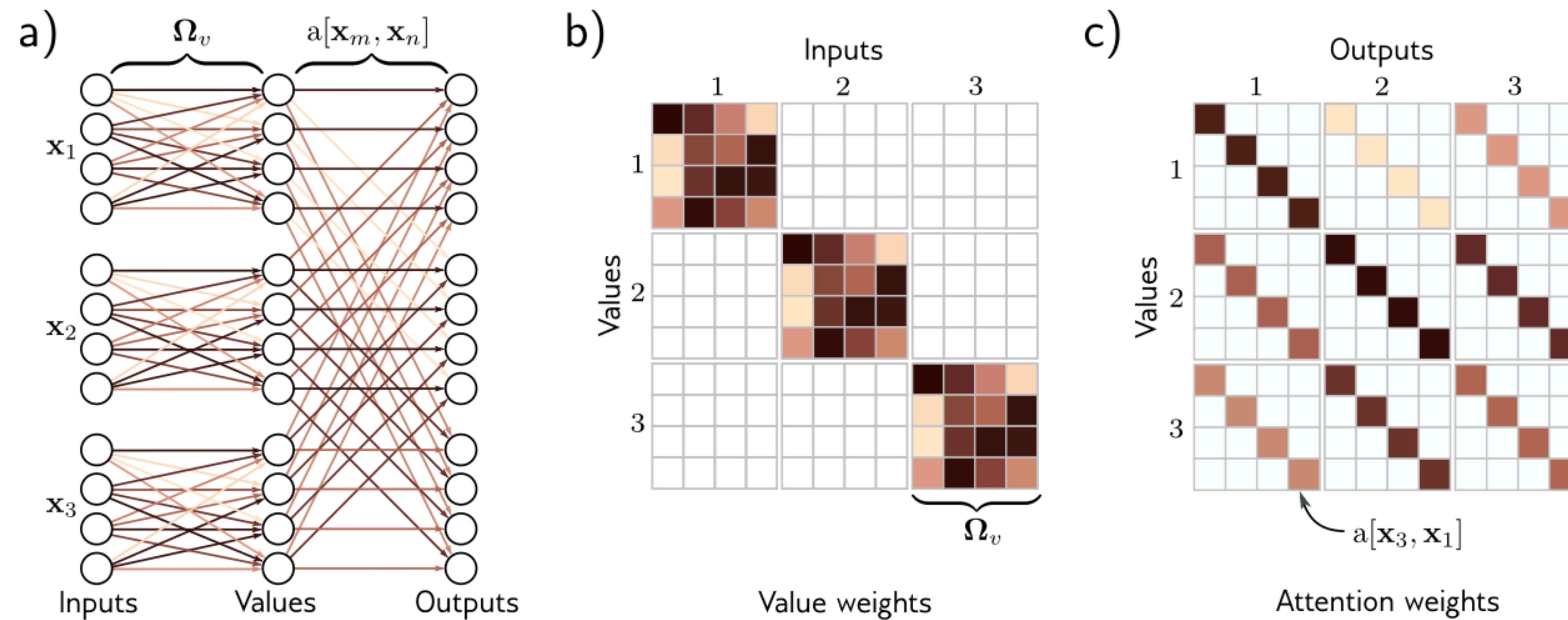
$$\hat{y}_i = f(\sum_j w_{ij} x_j + b_i)$$



Attention acts as an input rerouting, based on a learned matrix of cross-relevance

Computational cost under control

- Parameter Sharing: Same β and Ω for all input words (computation scales linearly with sentence length vs quadratic scaling of standard networks)
- Sparsity in $D \times N$ space: Attentions's computation is quadratic in N , but independent on D



Learning Attention

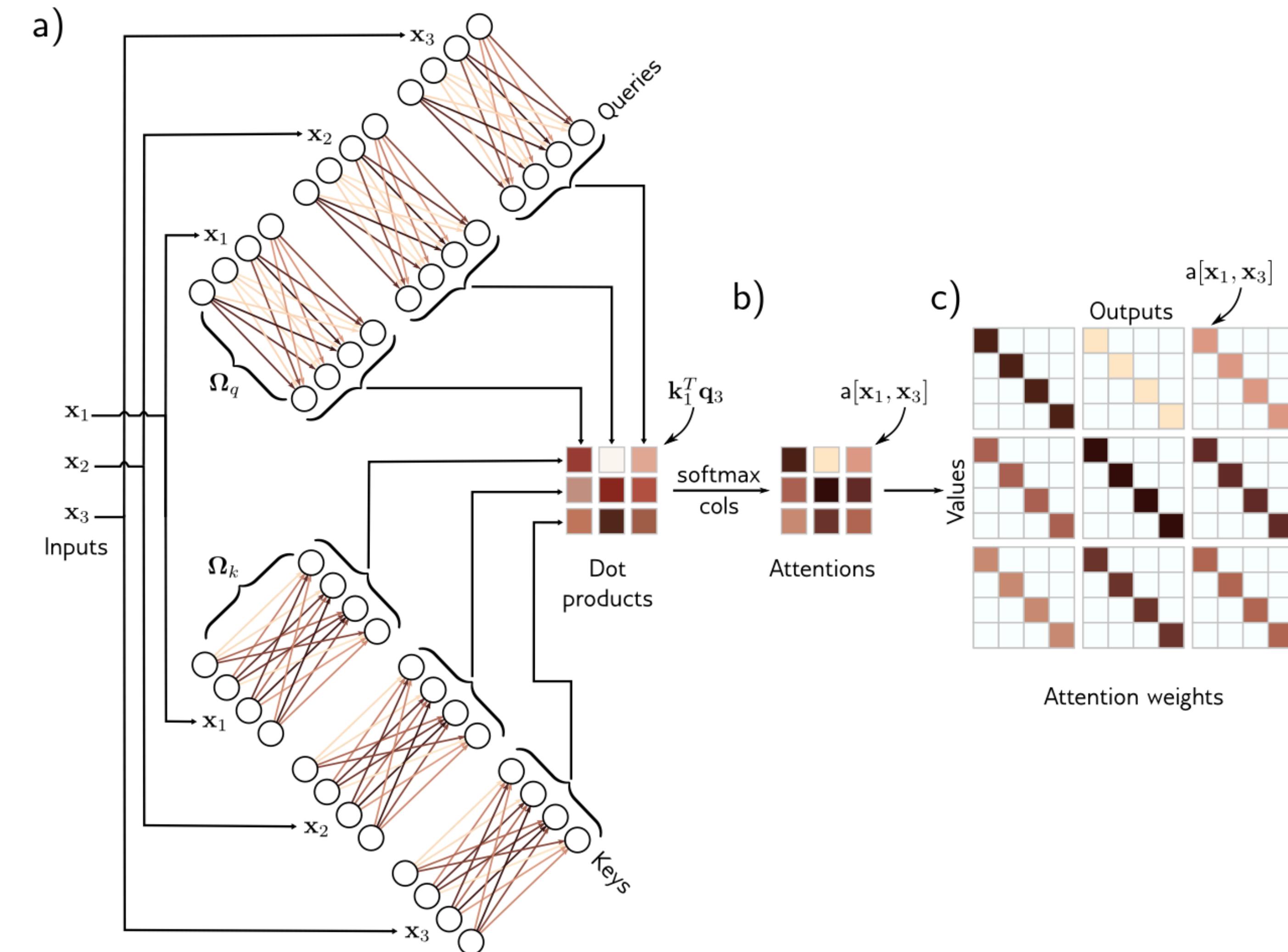
- *Attention is learned through two transformations of the inputs*

- **queries**: $q_n = \beta_q + \Omega_q x_n$

- **keys**: $k_n = \beta_k + \Omega_k x_n$

- *One then compute the attention as*

$$a[x_m, x_n] = \text{Softmax}(QK^T) = \frac{e^{k_m^T q_n}}{\sum_j e^{k_j^T q_n}}$$



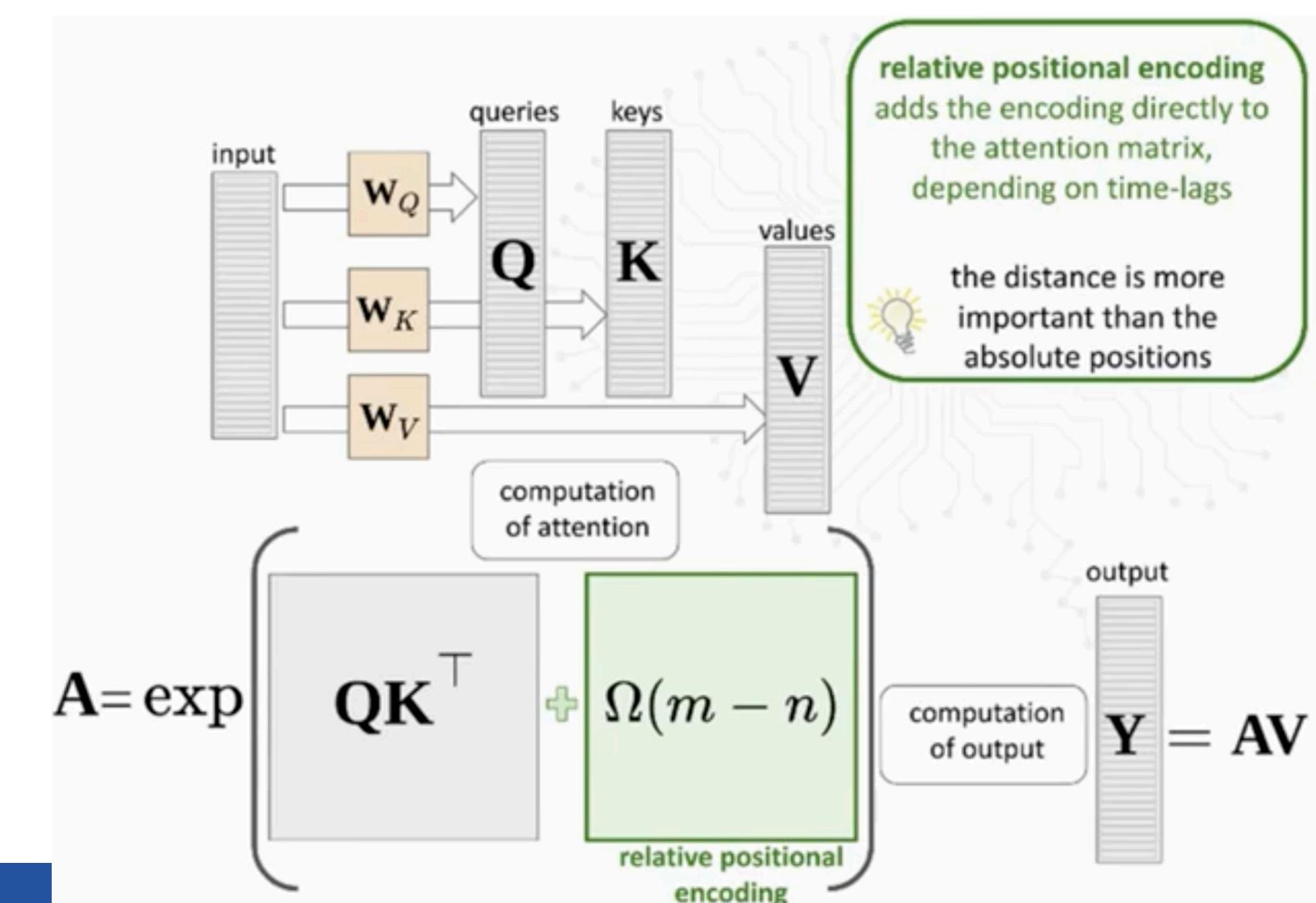
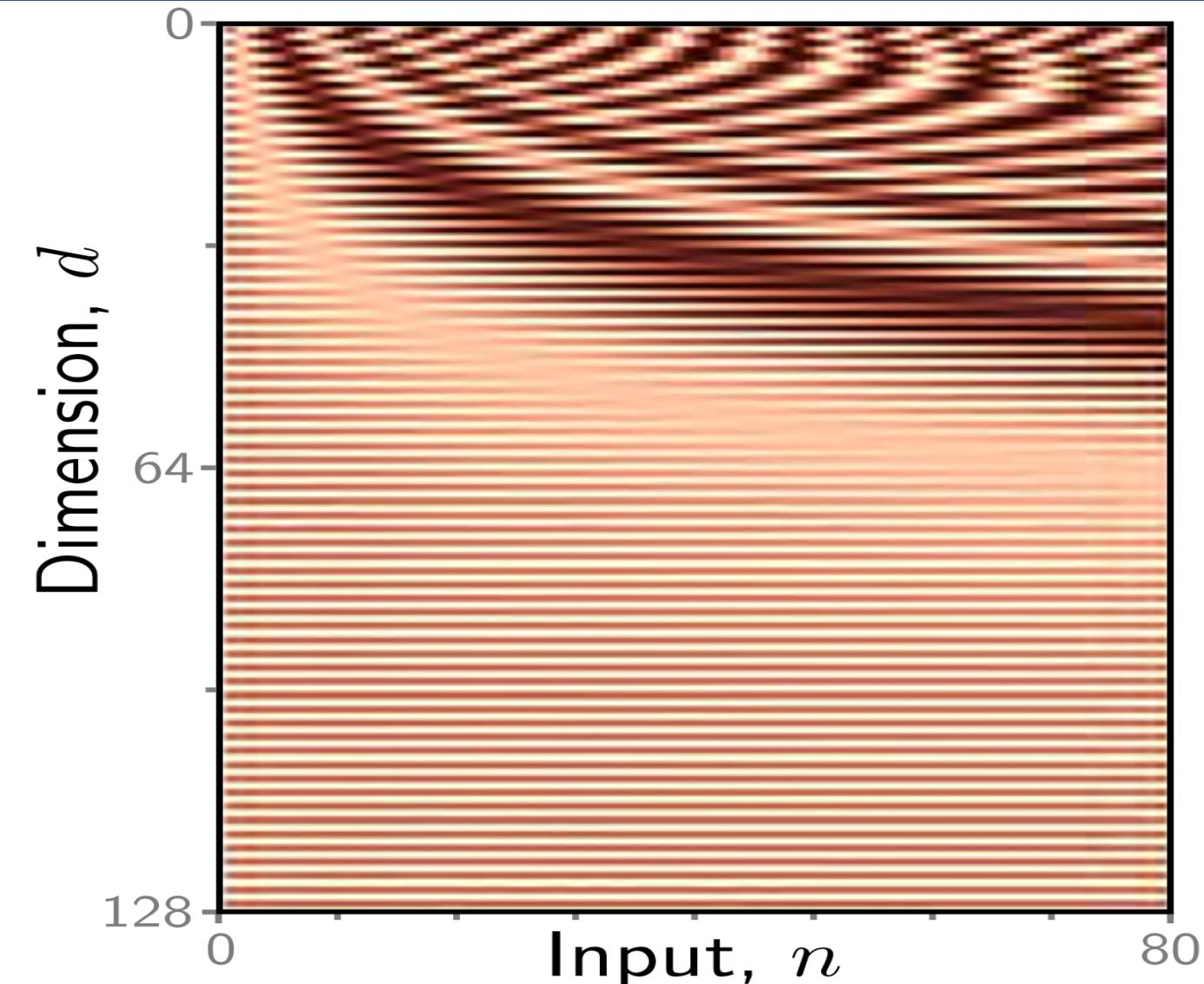


What is going on

- One starts computing the similarity between a given key and all the queries
- The Softmax is used to turn this into a normalized score (as for the output of a multiclass classifier trying to match a dictionary of categories)
- This is then used on an input value to identify the most relevant information to pass along (more in a few slides)

Extensions of Self Attention

- Notice that the order of the words in the dictionary has no role so far. Sometimes this is useful, but not necessarily with text processing
- **Positional Encoding (PE):** One could gain further insight taking into account the order of the inputs, breaking the permutation invariance of self attention.
- **Absolute PE:** One adds a matrix Π to the input, with columns that are unique \rightarrow they keep record of the absolute position. Π could be imposed (with some analytical pattern) or learned
- **Relative PE:** one learns a new matrix Ω_{mn} which depends on the distance between m and n . This matrix carries information on the relative position and it is used to modify the attention



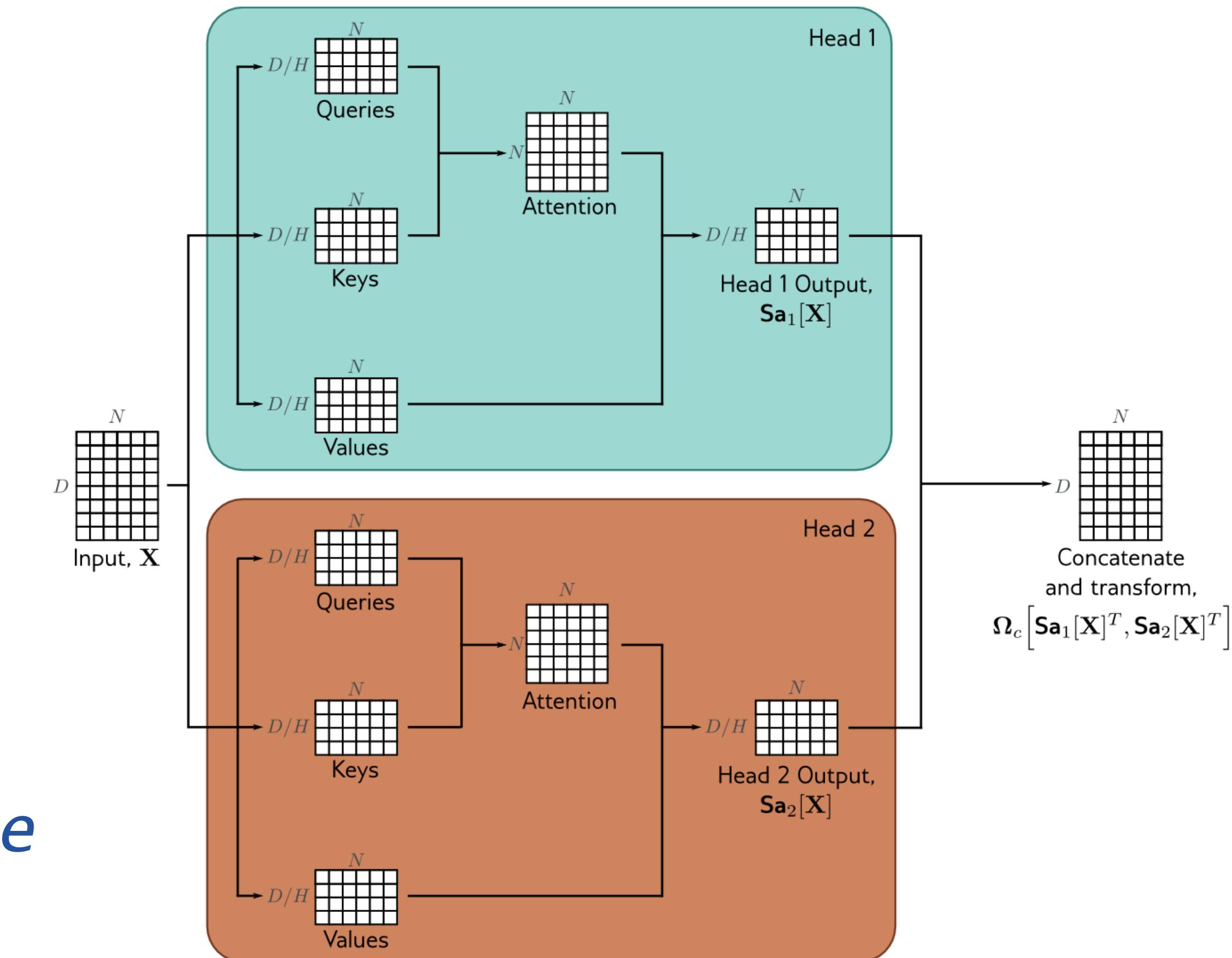
Extensions of Self Attention

- **Scaled dot product:** large dot products can drive the logit of the softmax in the plateaux region (we encountered this problem already)

- One scales the logit by $\sqrt{D_q}$ (dimension of queries and keys)

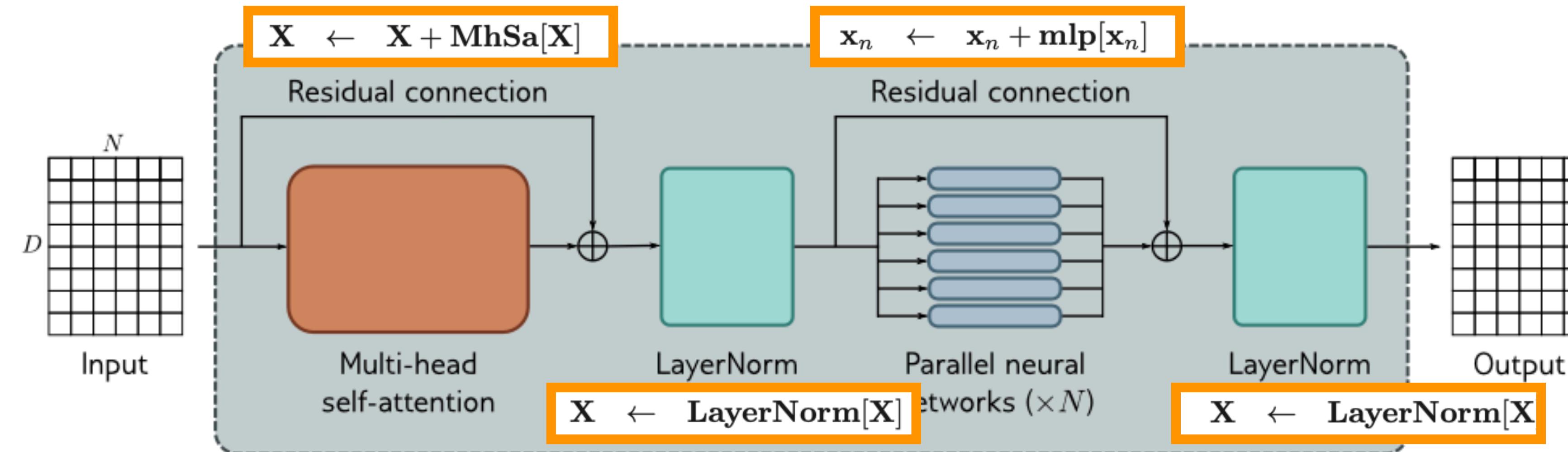
$$\mathbf{Sa}_h[\mathbf{X}] = \mathbf{V}_h \cdot \text{Softmax} \left[\frac{\mathbf{K}_h^T \mathbf{Q}_h}{\sqrt{D_q}} \right]$$

- **Multiple Heads:** One applies multiple attention mechanisms in parallel (like multiple kernels in CNN, multiple RNN units etc.). The outputs are concatenated to a single output



Transformer Layers

- Self-attention is the first step of the transformer layer
- A typical transformer layer
 - Uses residual connections for each operation
 - Starts with multi-head self attention
 - Process the self-attention output applying the same DNN to each word
 - Typically, one adds a Layer Normalization operation in between, similar to BatchNorm but wrt the layer values for a given input (and not across inputs in a batch)



Transformers for NLP

- A typical NLP pipeline exploits transformers through a pipeline of operations

a_sailor_went_to_sea_sea_sea_
to_see_what_he_could_see_see_see_
but_all_that_he_could_see_see_see_
was_the_bottom_of_the_deep_blue_sea_sea_sea_

- A tokenizer splits the text into words or word fragments

- Each token is embedded to a vector

-	e	s	a	t	o	h	l	u	b	d	w	c	f	i	m	n	p	r
33	28	15	12	11	8	6	6	4	3	3	3	2	1	1	1	1	1	1

- the embedded vectors are processed by a chain of transformer layers

Tokeneizer

- Splits text in smaller constituents from a pre-defined vocabulary. Some care needed with details
- Not all fragments will be in the vocabulary
- punctuation has to be handled somehow since it carries important information
- same word will have to enter multiple times depending on suffixes (walk, walks, walked, walking) but the relation between them will not be obvious
- One could use minimal fragments (letters and punctuation) losing information in the process that the network will have to re-learn
- Instead, one usually builds a mix of words and word fragments, starting with letters and merging frequent combinations

a_sailor_went_to_sea_sea_sea_
to_see_what_he_could_see_see_see_
but_all_that_he_could_see_see_see_
was_the_bottom_of_the_deep_blue_sea_sea_sea_

_	e	s	a	t	o	h	l	u	b	d	w	c	f	i	m	n	p	r
33	28	15	12	11	8	6	6	4	3	3	3	2	1	1	1	1	1	1

a_sailor_went_to_sea_sea_sea_
to_see_what_he_could_see_see_see_
but_all_that_he_could_see_see_see_
was_the_bottom_of_the_deep_blue_sea_sea_sea_

_	e	se	a	t	o	h	l	u	b	d	w	c	s	f	i	m	n	p	r
33	15	13	12	11	8	6	6	4	3	3	3	2	2	1	1	1	1	1	1

a_sailor_went_to_sea_sea_sea_
to_see_what_he_could_see_see_see_
but_all_that_he_could_see_see_see_
was_the_bottom_of_the_deep_blue_sea_sea_sea_

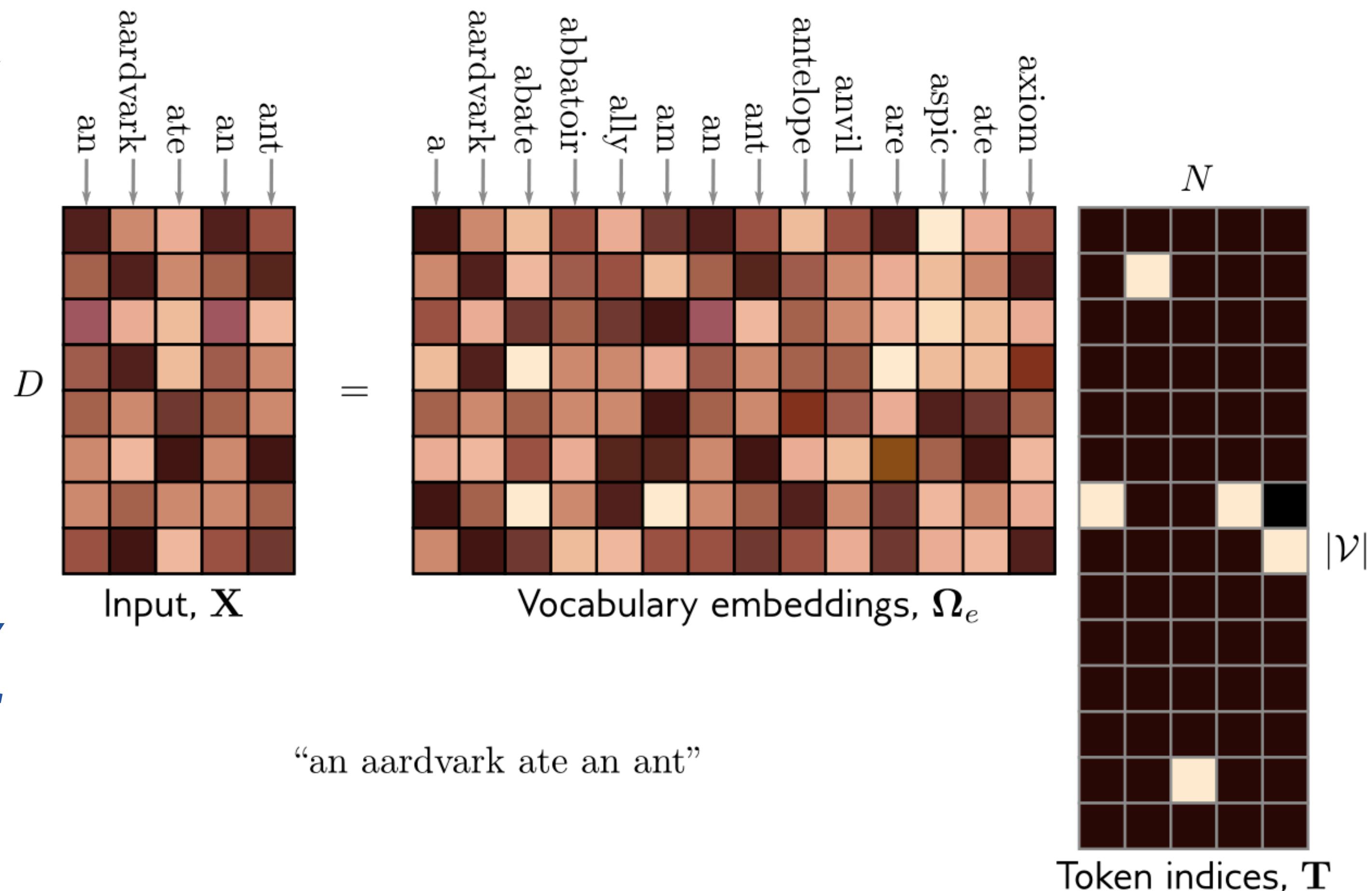
_	se	a	e_	t	o	h	l	u	b	d	e	w	c	s	f	i	m	n	p	r
21	13	12	12	11	8	6	6	4	3	3	3	3	2	2	1	1	1	1	1	

Learned Embedding (again)

- *N words (or word fragments) are stored in a $N \times |\mathcal{V}|$ one-hot encoding matrix, depending on some given vocabulary \mathcal{V}*

- *This matrix is multiplied by a learned matrix Ω_e of size $D \times |\mathcal{V}|$, which maps each word of the vocabulary into a D -dimensional vector (typically D is 1024)*

- *The result is a $N \times D$ matrix*

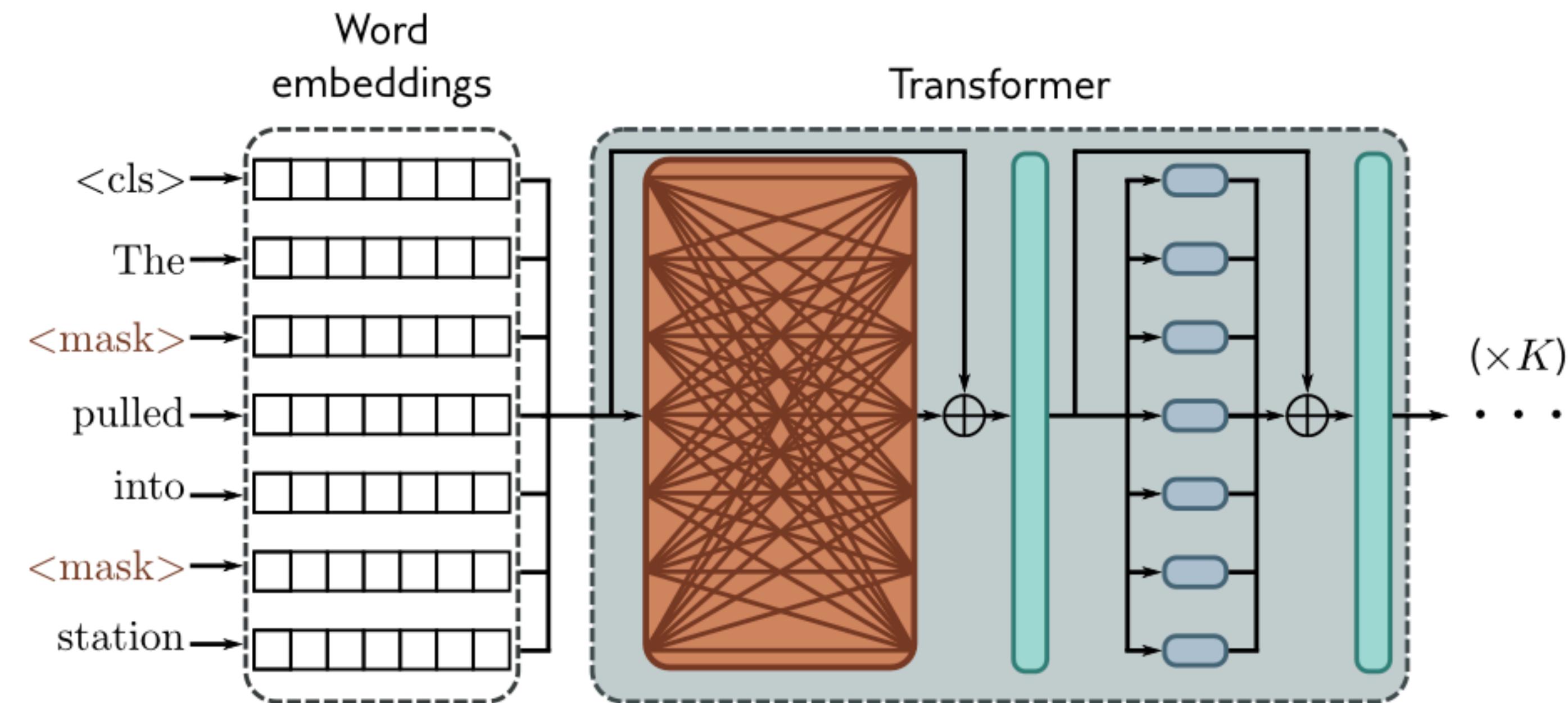




Transformer models

- *Encoders: turns text into an abstract representation from which various tasks can be accomplished*
- *Example: BERT*
- *Decoders: Predicts the next token to complete a text*
- *Example GPT*
- *Encoder-decoder: used in sequence-to-sequence tasks, to convert one string into another (e.g., for translations)*
- *Example: Machine Translation*

- Vocabulary of $|\mathcal{V}| = 30K$ tokens
- Inputs converted to vectors of dimension $D=1024$
- The embedded input is processed by a chain of 24 transformers
- 16 heads of attentions
- values, keys and queries have dimension 64 (i.e., the matrices in the transformer layer are 64×1024)
- the hidden layer in the fully connected network has 4096 parameters



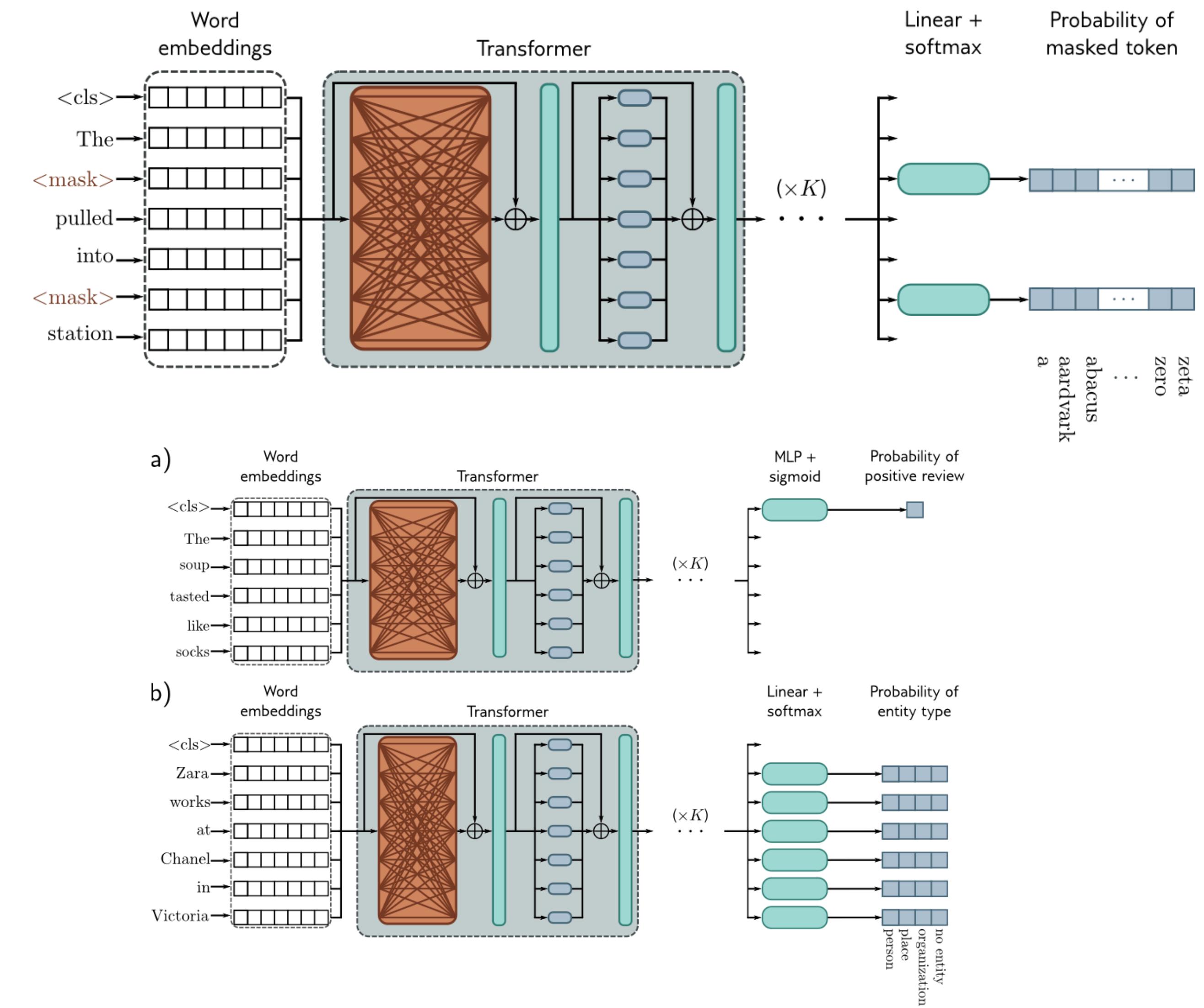
~340 Million parameters (a relatively small transformer by today standards)

- *BERT is trained in two steps:*

- *Self-supervised learning stage, where generic language structure is learned from large set of text*

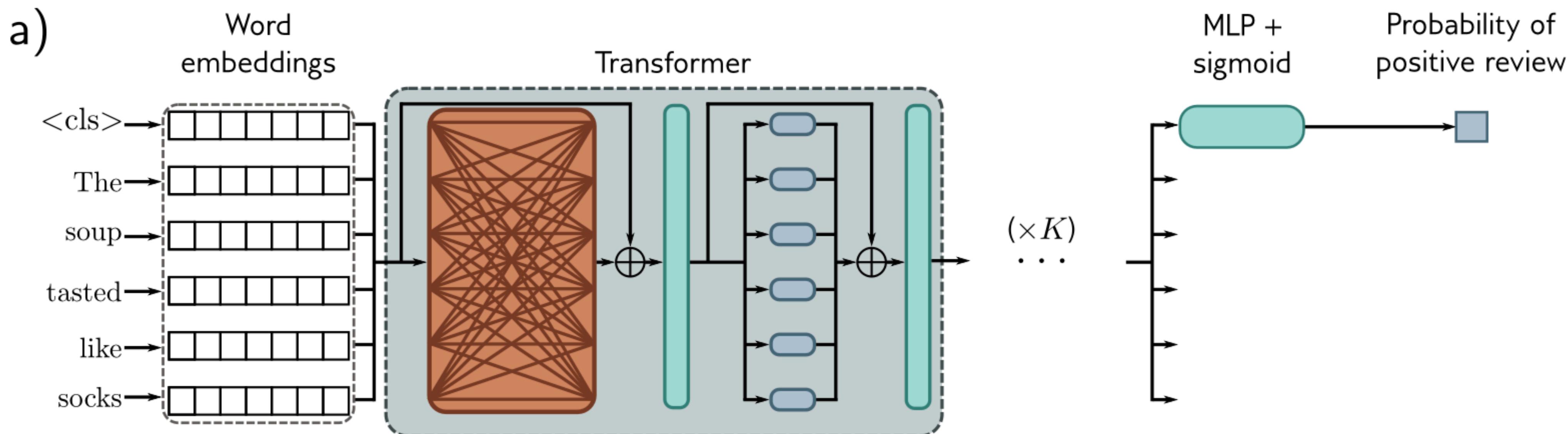
- *Predict words in text, no need for human labels*

- *Fine-tuning stage: focusing on a smaller dataset, the model is trained to accomplish a specific task*



Special Tokens

- In BERT, special tokens are inserted in the text to facilitate certain tasks
- <cls> classification tokens at the beginning of a sentence
- e.g., In sentiment analysis a sentence is labelled as positive or negative. The MLP processing of the <cls> token is passed to a sigmoid, and used to evaluate a classification loss vs the label

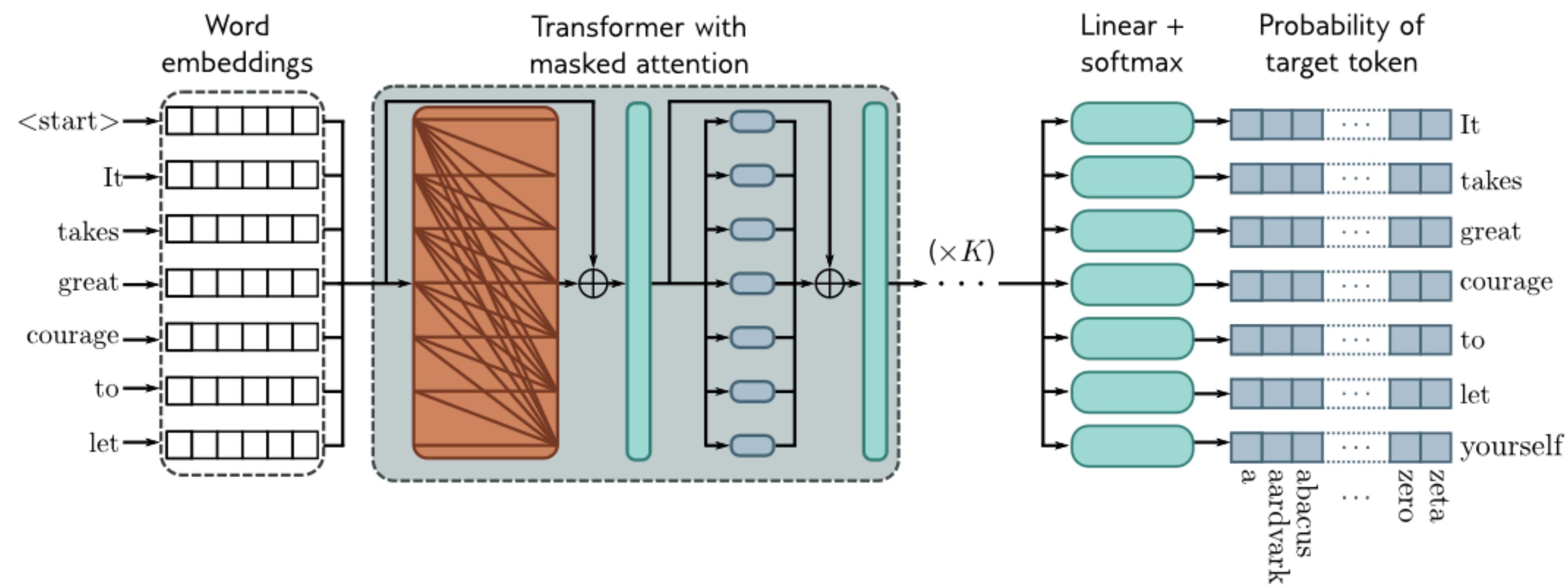


- Generative Predictive Transformers (GPTs) are the models behind ChatGPT
- Structure similar to encoders, but opposite purpose
- Start from embedded text to generate coherent text
- It uses an autoregressive approach:
 - Write probability of a sentence as product of token probabilities
 - Learn to predict $P(t_n)$ from $\prod P(t_1) \dots P(t_{n-1})$ progressively for all the tokens

$Pr(\text{It takes great courage to let yourself appear weak}) =$
 $Pr(\text{It}) \times Pr(\text{takes}|\text{It}) \times Pr(\text{great}|\text{It takes}) \times Pr(\text{courage}|\text{It takes great}) \times$
 $Pr(\text{to}|\text{It takes great courage}) \times Pr(\text{let}|\text{It takes great courage to}) \times$
 $Pr(\text{yourself}|\text{It takes great courage to let}) \times$
 $Pr(\text{appear}|\text{It takes great courage to let yourself}) \times$
 $Pr(\text{weak}|\text{It takes great courage to let yourself appear}).$

$$Pr(t_1, t_2, \dots, t_N) = Pr(t_1) \prod_{n=2}^N Pr(t_n|t_1, \dots, t_{n-1})$$

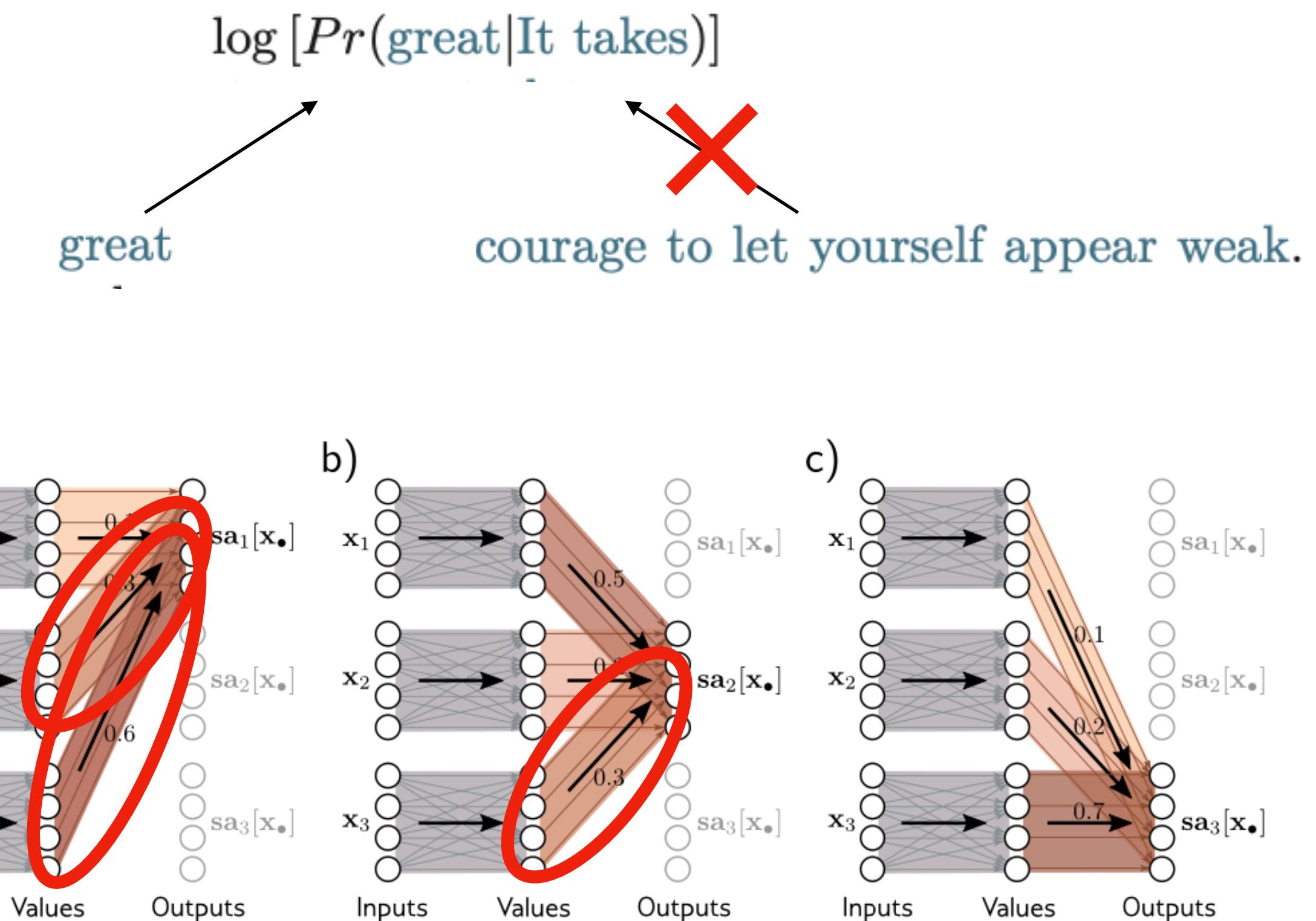
- In the example, we assumed for simplicity that the inputs are plain words. In real life, a decoder would start from embedded words
- Further steps are similar to those of an encoder
- The output is a set of embeddings, each representing a partial embedding. One has to predict the next in sequence
- A linear layer maps outputs to a vocabulary and softmax converts the score into a probability



Masked Self Attention

- Training a decoder is tricky
- Doing it sequentially (maximising next-word probability one by one) would take forever
- Parallelising naively will lead to network cheating (the answer is provided by the rest of the sentence as part of the training dataset)
- An efficient training can be performed in parallel, zeroing the element of the attention matrix that connects a word to its context
- In practice, this is done setting the corresponding dot product element to a very large negative number and passing it to the softmax

It takes courage to let yourself appear weak



Masked self attention forces all upward pointing rerouting to be zero

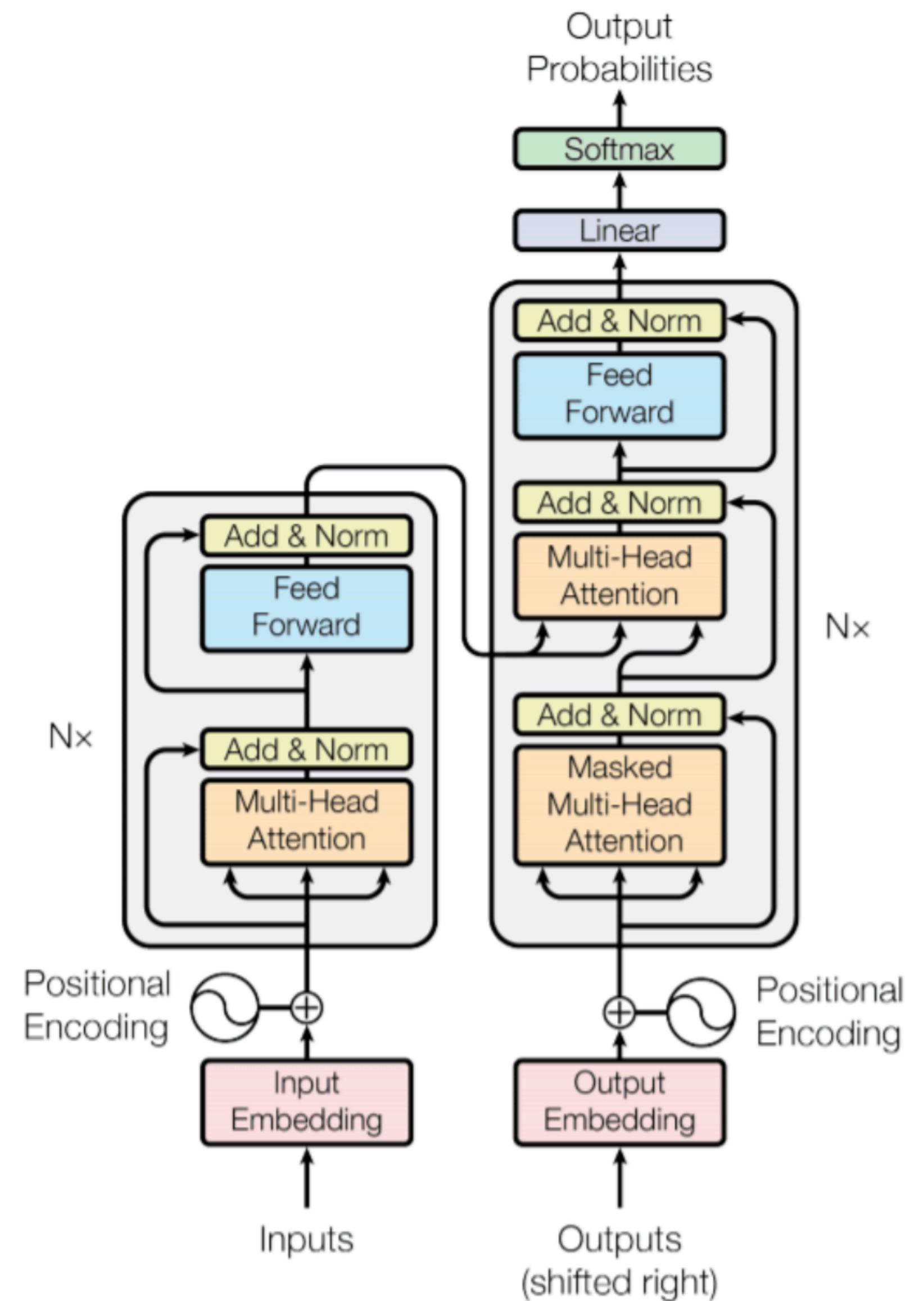


Generating Text

- Once trained, the decoder learns a probability model for language
- It can predict the next word → word by word, it can generate text from nothing
- A decoder can be used as a text generator, passing it a special <start> token, signalling the beginning of a sentence
- Efficient computation
 - thanks to masked self attention, no dependence of prior embedding to subsequent evolution of the sentence generation → early computation of embedding can be recycled later on
 - tricks developed to make the generation process more pertinent
 - generating multiple options and picking the most likely in a classification task
 - generating only from top-k word options
 - balance between the two approaches is an exploration/exploitation problem (similar to what is found in other AI applications for Deep Learning, e.g., Reinforcement Learning)

How big is GPT

- GPT models scale up the concept of transformer decoders to unprecedented levels.
- Example: ChatGPT3
- Sequences are 2048 tokens long. Total batch size is 3.2M tokens
- 96 transformers, each processing a word embedding of size 12288
- 96 attention heads for each transformer, with values, queries and keys of dimension 128
- It total 175 Billion parameters trained on 300 billion tokens
- See https://dugas.ch/artificial_curiosity/GPT_architecture.html for a detailed walk through





Few-shot learning

- *The most impressive aspect of GPT models is their capability to learn with a few examples (much closer to human experience)*
- *They are said to be capable of “few shot learning”*
- *Be careful though that high likelihood answers are not always wrong*
- *This is a generative model, so it will generate a reasonable answer to its best capability but not always this will result in the right answer*

Poor English input: I eated the purple berries.

Good English output: I ate the purple berries.

Poor English input: Thank you for picking me as your designer. I'd appreciate it.

Good English output: Thank you for choosing me as your designer. I appreciate it.

Poor English input: The mentioned changes have done. or I did the alteration that you requested. or I changed things you wanted and did the modifications.

Good English output: The requested changes have been made. or I made the alteration that you requested. or I changed things you wanted and made the modifications.

Poor English input: I'd be more than happy to work with you in another project.

Good English output: I'd be more than happy to work with you on another project.

Text given to the model

Text generated by the model

Contents

Preface

Acknowledgements

1 Introduction

1.1	Supervised learning
1.2	Unsupervised learning
1.3	Reinforcement learning
1.4	Ethics
1.5	Structure of book
1.6	Other books
1.7	How to read this book

2 Supervised learning

2.1	Supervised learning overview
2.2	Linear regression example
2.3	Summary

3 Shallow neural networks

3.1	Neural network example
3.2	Universal approximation theorem
3.3	Multivariate inputs and outputs
3.4	Shallow neural networks: general case
3.5	Terminology
3.6	Summary

Understanding Deep Learning is a new textbook from MIT Press by Simon Prince that's designed to offer an accessible, broad introduction to the field. Deep learning is a branch of machine learning that is concerned with algorithms that learn from data that is unstructured or unlabeled. The book is divided into four sections:

1. Introduction to deep learning
2. Deep learning architecture
3. Deep learning algorithms
4. Applications of deep learning

The first section offers an introduction to deep learning, including its history and origins. The second section covers deep learning architecture, discussing various types of neural networks and their applications. The third section dives into deep learning algorithms, including supervised and unsupervised learning, reinforcement learning, and more. The fourth section applies deep learning to various domains, such as computer vision, natural language processing, and robotics.

Machine Translation

- Translation is an encoder-decoder sequence-to-sequence problem
- A text sequence comes in, in Language A
- A text sequence goes out, in Language B
- Seq-to-seq models used before Transformers were based on RNNs
- Since Transformers were introduced, they are the best option for this task
- Same construction as before, but replacing self attention with **cross attention** in the decoder: queries come from the decoder embedding (Language B), while the keys and values are taken from encoder embedding (Language A)

