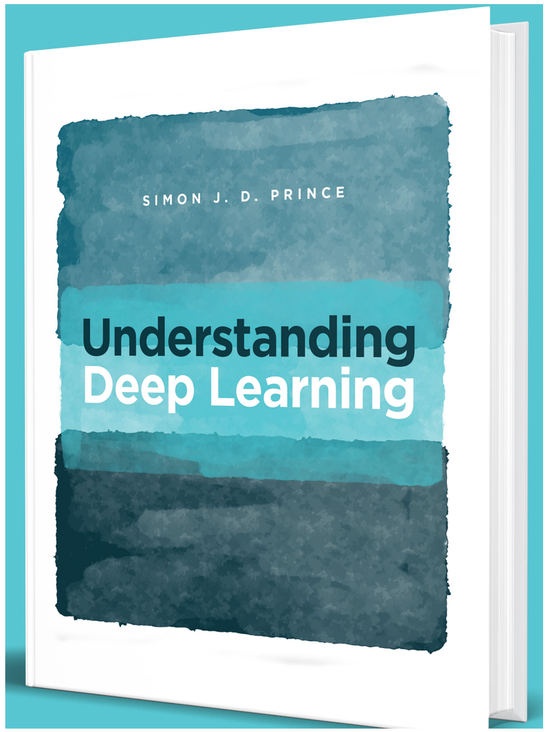
Lecture 10: Transformers

Maurizio Pierini (CERN), Thea Aarrestad (ETH)

Program for Today



Date	Topic	Tutorial
Sep 17	Intro & class description	Linear Algebra in a nutshell + prob and stat
Sep 24	Basic of machine learning + Dense NN	*Basic jupyter + DNN on mnist (give jet dnn as homework)*
Oct 1	Convolustional NN	*Convolutional NNs with MNIST*
Oct 8	Training in practice: regularization, optimization, etc	*Practical methodology*
Oct 15	*Google tutorial*	
Oct 22	Recurrent NN	*Hands-on exercise*
Oct 29	Graph NNs	*Tutorial on Graph NNs*
Nov 5	Unsupervised learning and anomaly detection	*Autoencoders with MNIST*
Nov 12	Generative models: GANs, VAEs, etc	Normalizing flows
Nov 19	Transformers	
Nov 26	Network compression (pruning, quantization, Knoweldge Distillation)	
Dec 3	Tutorial on hls4ml/qkeras	
Dec 10	Quantum Machine Learning	
Dec 17	Q/A Prior to exam	

NLP beyond RNNs

- *At the begging of this course, we discussed the two architectures that drove the Deep Learning revolution*
  - *CNNs for image processing*
  - *RNNs for Natural Language Processing (NLP)*
- *Today we discus Transformers*
  - *Initially targeting NLPs, processing data in which sentences are embedded*
- *We discussed how these two classes of problems differ*
  - *They share similar challenges in terms of large dimensionality of the data*
  - *But text data is variable in size, while images are at fixed size*

Processing text

- *A two-step process*
  - *process the text and map it to some abstract representation of its meaning*

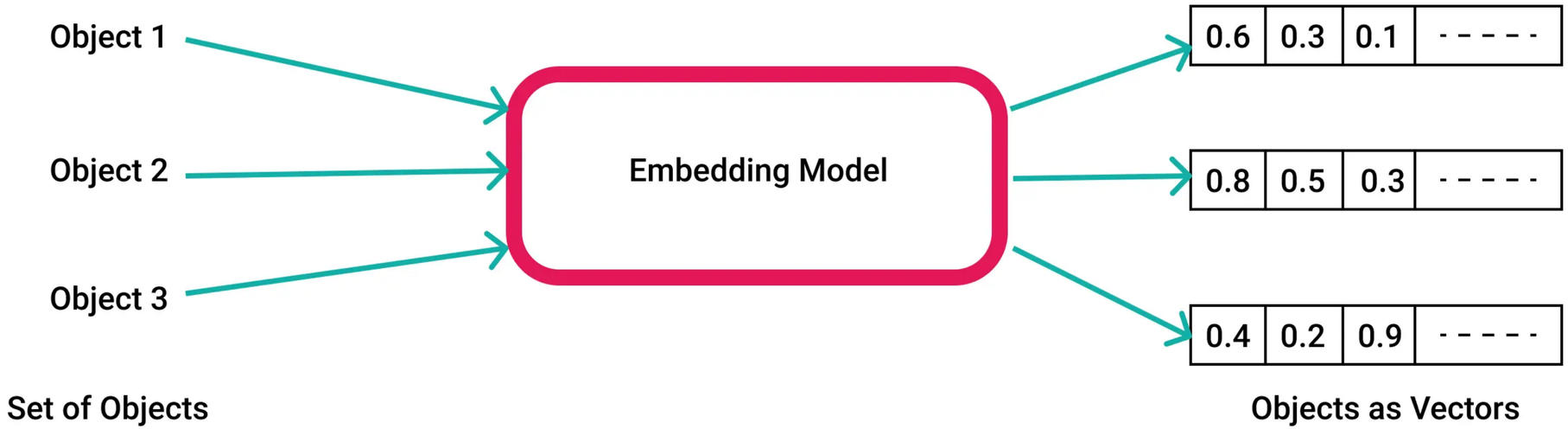
> The restaurant refused to serve me a ham sandwich because it only cooks vegetarian food. In the end, they just gave me two slices of bread. Their ambiance was just as good as the food and service.

  - *Use this abstract representation to answer specific questions*

> Does the restaurant serve steak?

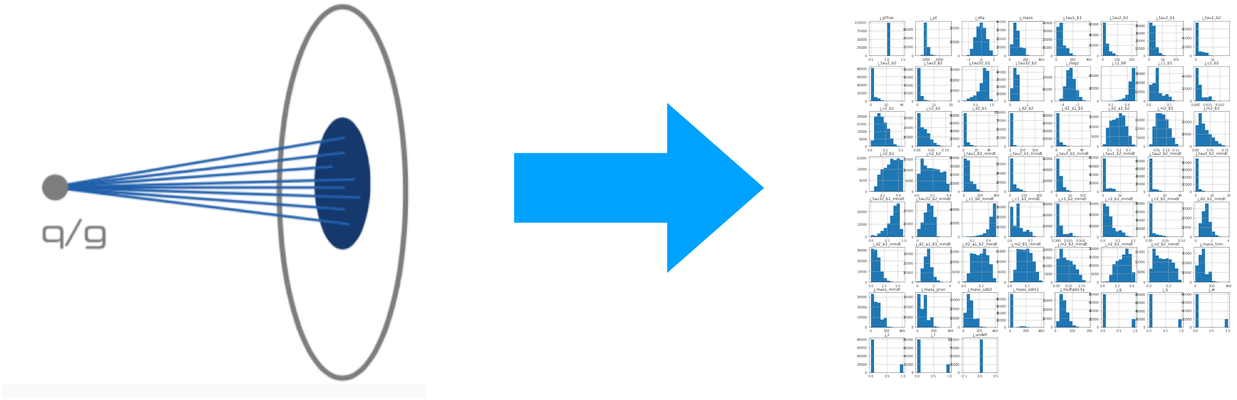
Vector Embedding

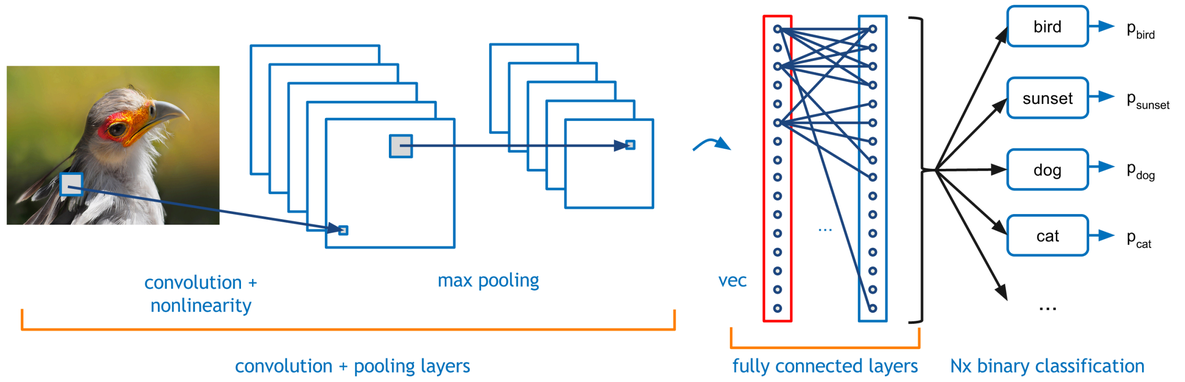
- *Vector embedding is a standard procedure to digitise a text so that it can be turned into numbers that can be processed by a ML algorithm*
- *Vector embedding*
  - *Turns "tokens" (e.g., words) into fixed-size vectors*
  - *It does so that similar text is turned into nearby vectors (semantic similarity 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: the HLFs in your jet classification exercise use physics knowledge to transform a jet into an array of numbers*
- *One could use a Deep Learning model*
  - *We saw the example of CNNs for image classification*





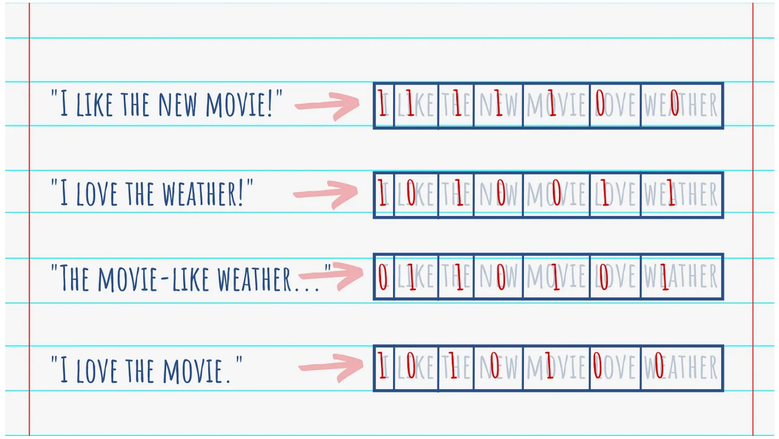
Rule-based vs Learned

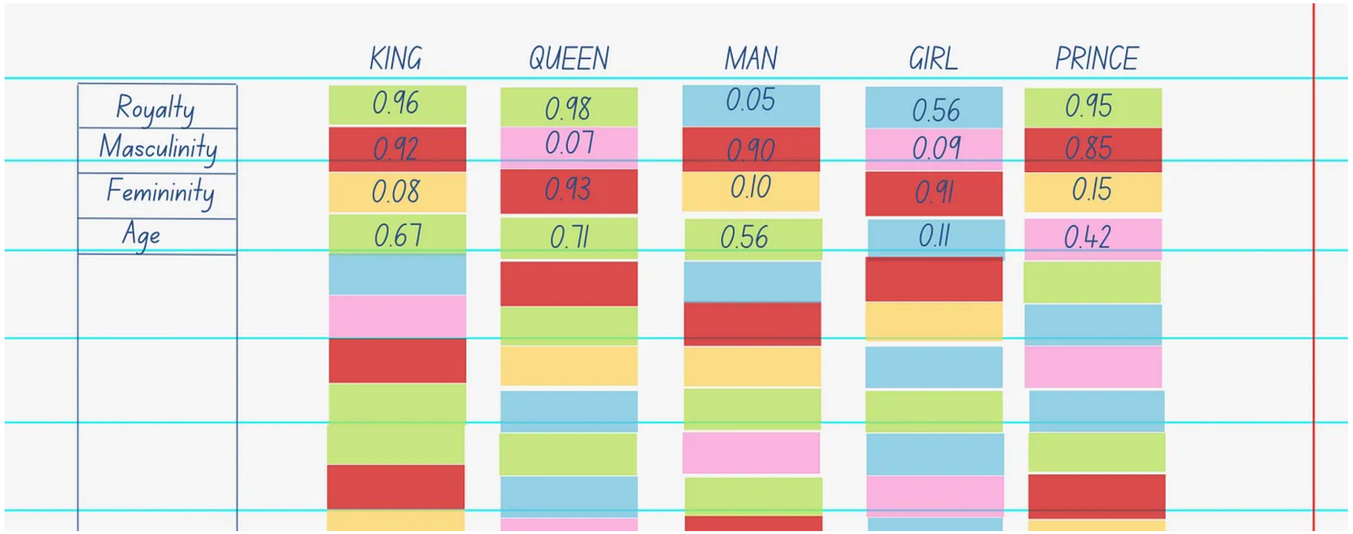
- To run a MLP model, one has to turn text into some numerical form
  - Bag Of Words:
    - use one-hot encoding for words
    - a sentence becomes a vector
- One can go beyond this, using NNs to learn a vectorial representation of the sentence, rather than using BoW rules (Word2Vec, GLoVE, BERT)

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

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

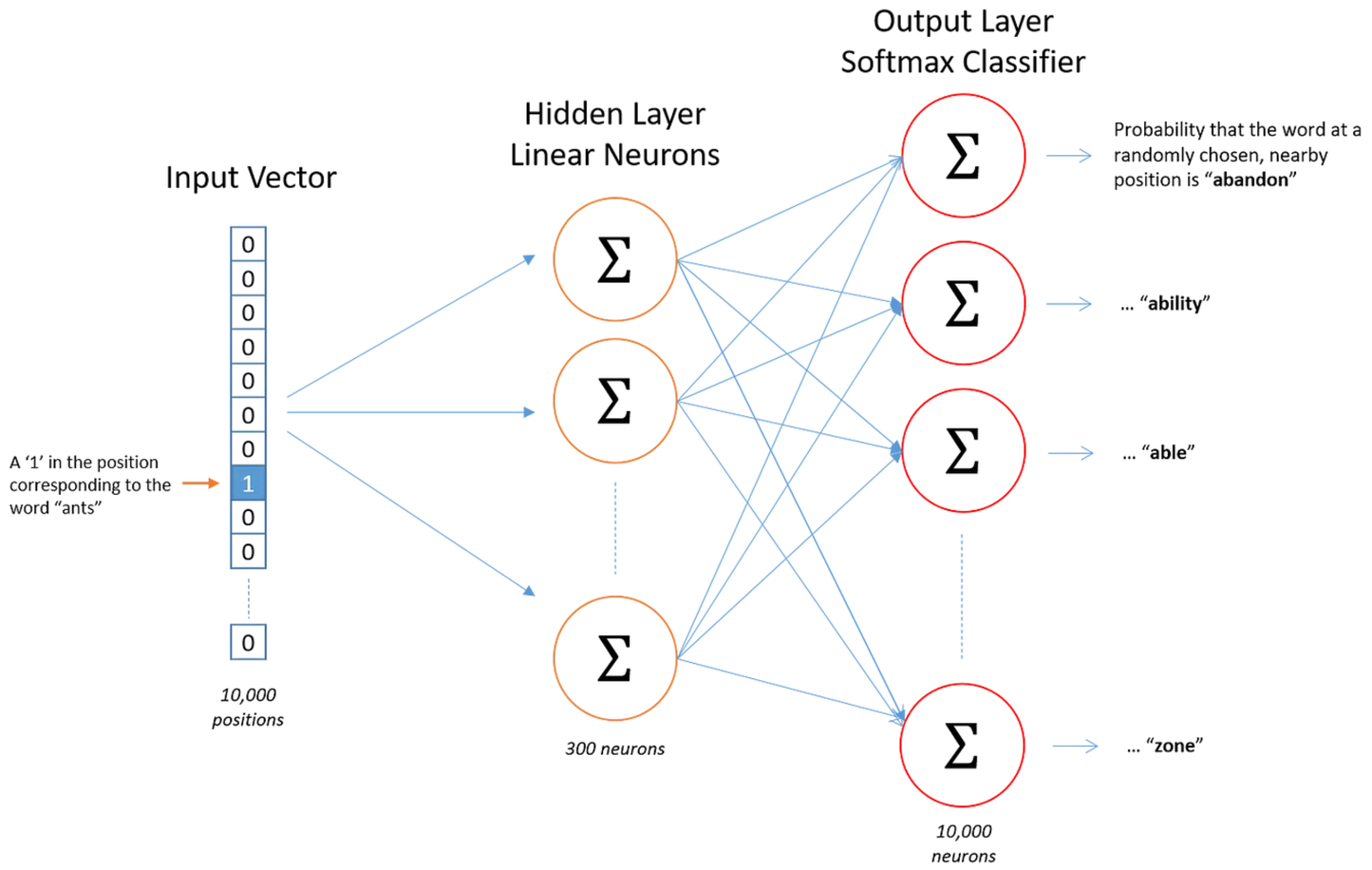
the [0,0,1,0,0,0,0]





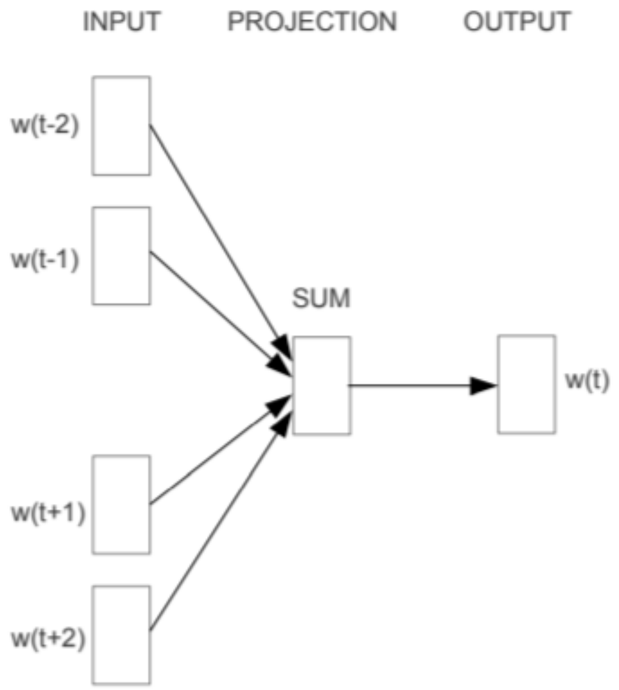
Word2Vec

- *Shallow neural network with number of neurons = dimensionality that we want for words*
- *Starts with a Bag-Of-Words representation and replace each cell with the probability of each word*
  - *Continuous Bag of Words (CBOW) predicts the target word taking as input the nearby words in the sentence*
  - *Skip-gram: predict them other words in the sentence taking as input the target word*

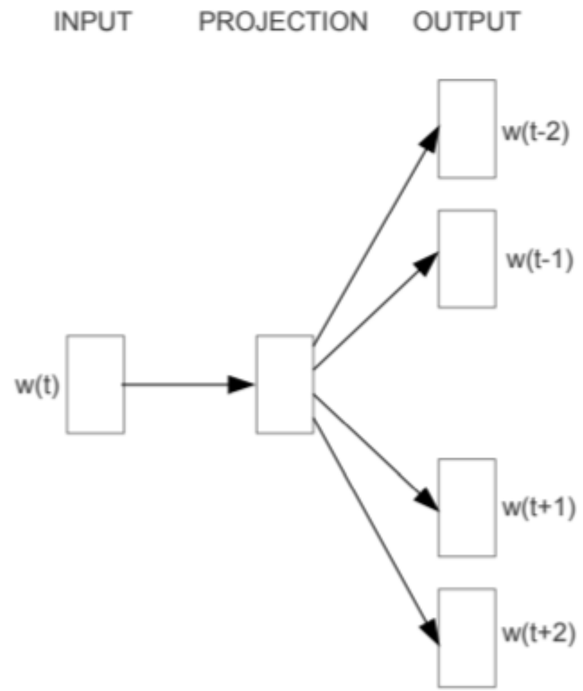


Word2Vec

- *Shallow neural network with number of neurons = dimensionality that we want for words*
- *Starts with a Bag-Of-Words representation and replace each cell with the probability of each word*
  - *Continuous Bag of Words (CBOW) predicts the target word taking as input the nearby words in the sentence*
  - *Skip-gram: predict them other words in the sentence taking as input the target word*



CBOW



Skip-gram

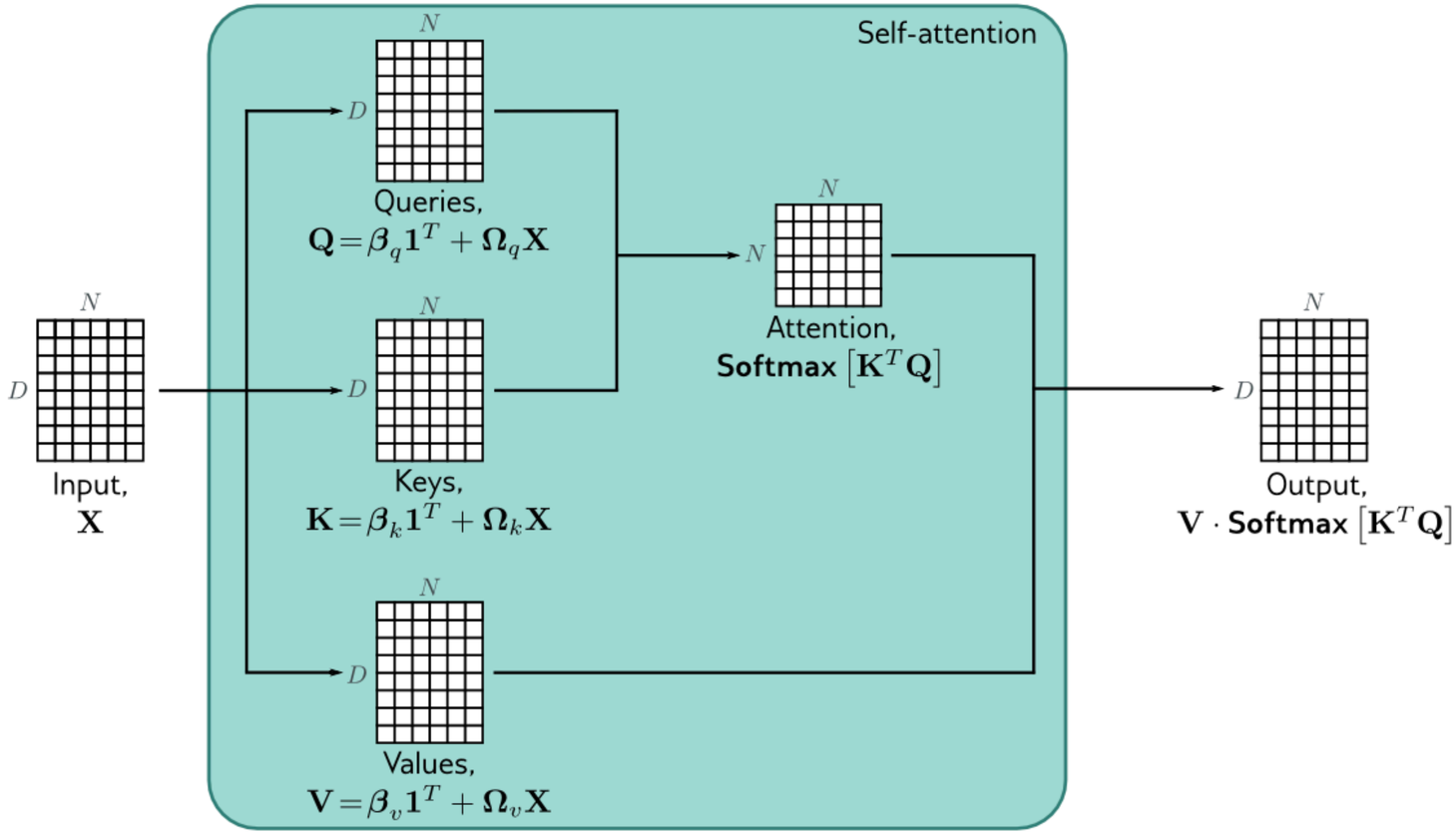
Issues with Word Encoding

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, using parameter sharing to process text*
- *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 an mechanism to learn the connections between words*

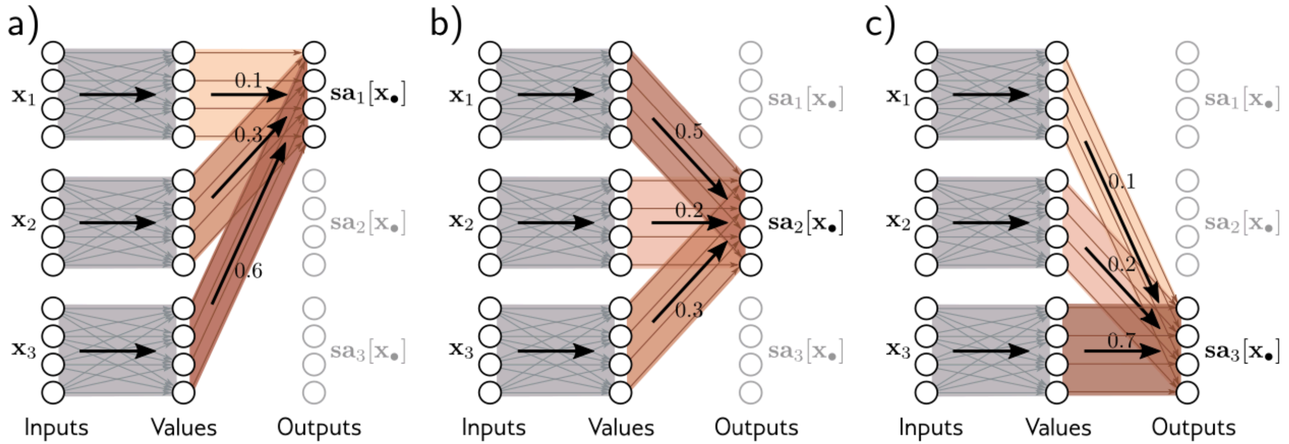


Dot-product self attention

- A standard Neural network layer takes an input vector x of dimension D and returns $f(\beta + \Omega x)$ where f is the activation function
- A self-attention block $sa[]$ takes as input N D-dim vectors $x_1, x_2, \ldots, x_N$ (words) and returns N D-dim outputs
  - Compute values $v_m = \beta + \Omega x_m$
  - Return a weighted average

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

  - The matrix of weighs is the attention matrix $a[x_m, x_n]$
    - $a[x_m, x_n] \geq 0$
    - $\sum n, m a[x_m, x_n] = 1$

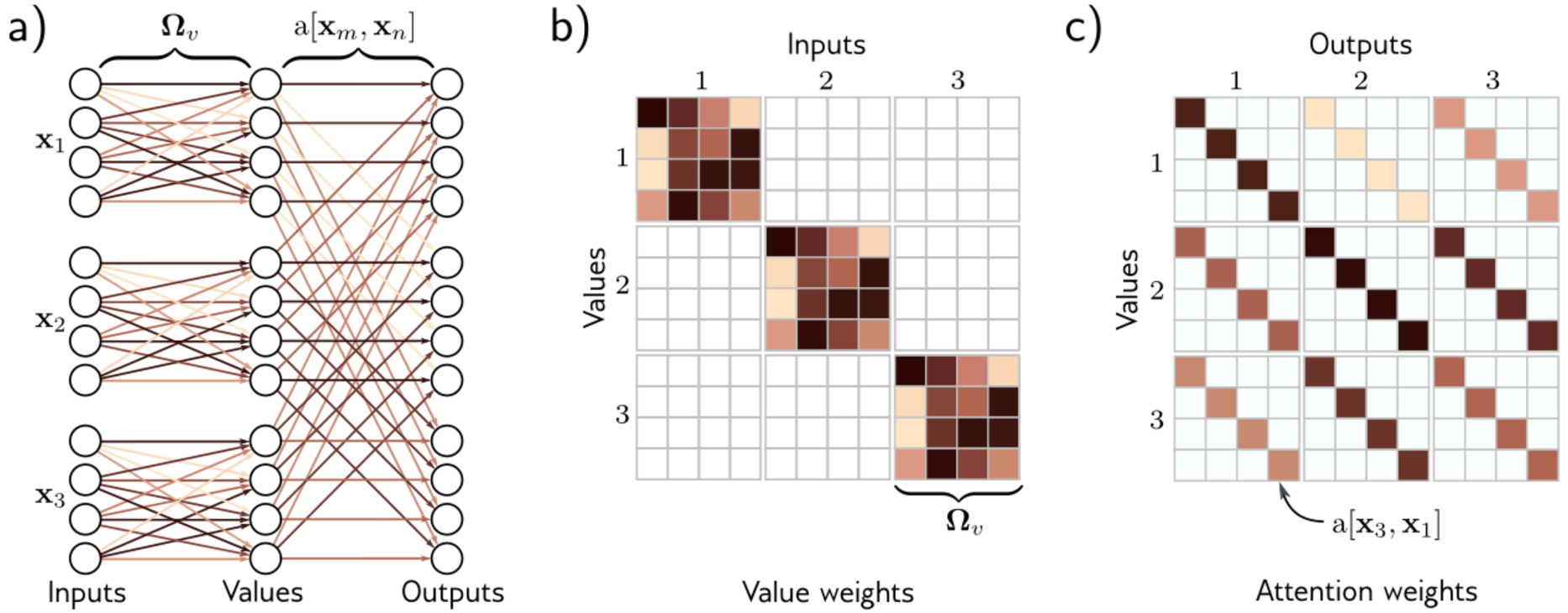


Attention acts as an input rerouting, based on a learned matrix of cross-relevance

Computational cost under control

- *Parameter Sharing: Same β and Ω for all inout words (computation scales linearly with sentence length vs quadratic scaling of standard networks)*

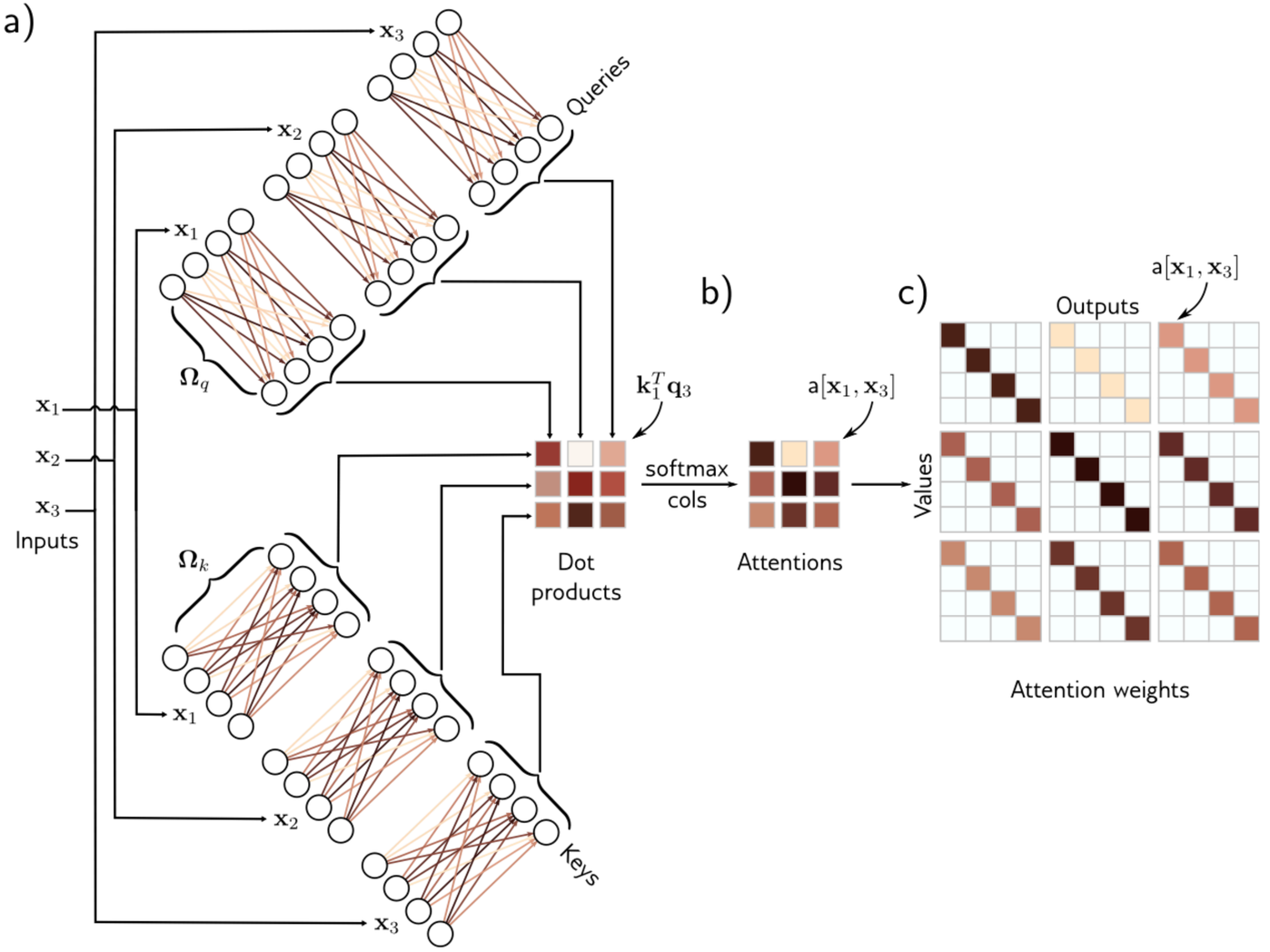
- *Sparsity in DxN space: Attentions's computation is quadratic in N, but independent on D*



Learning Attention

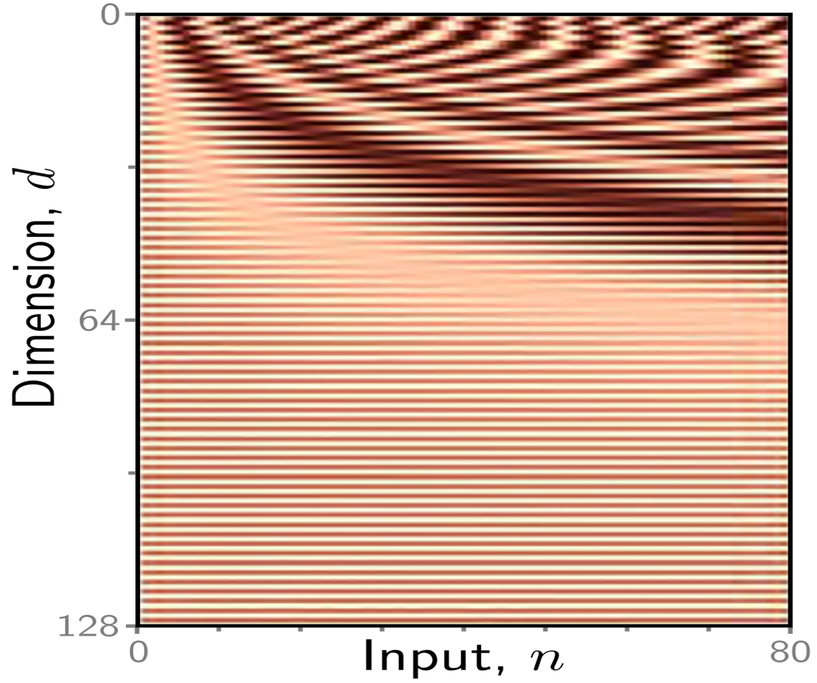
- *Attention is learned through two transformations of the inputs*
  - *queries:* $q_n = \beta_q + \Omega_q x_n$
  - *keys:* $q_n = \beta_k + \Omega_k x_n$
- *One then compute the attention as*

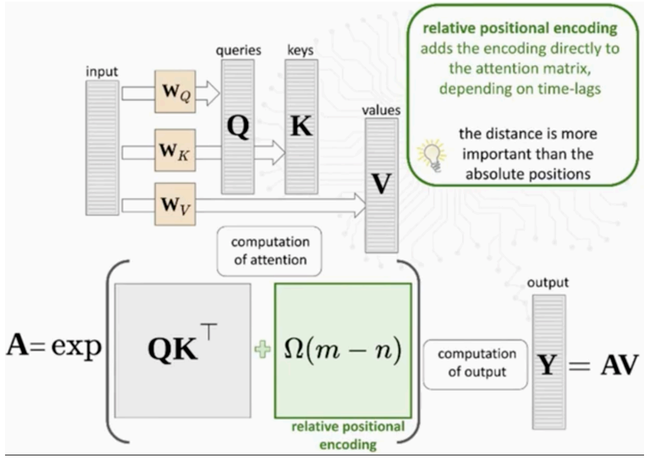
$$a[x_m, x_n] = \frac{e^{k_m^T q_n}}{\sum_j k_j^T q_n}$$



Extensions of Self Attention

- *Positional Encoding: 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 → 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*



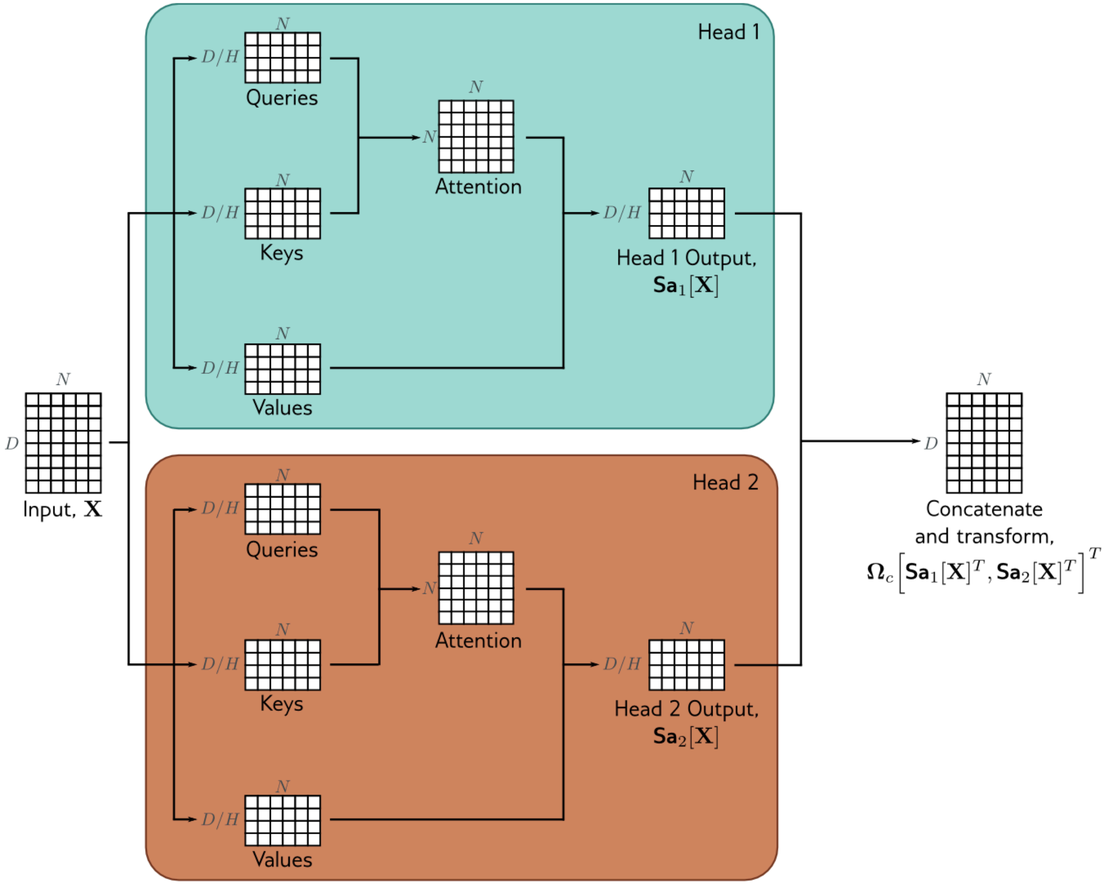


Extensions of Self Attention

- *Scaled dot product: large dot products can drive the logic of the softmax in the plateaux region (we encountered this problem already).*
  - *One scales the logic by* $\sqrt{D_q}$ *(dimension of queries and keys)*

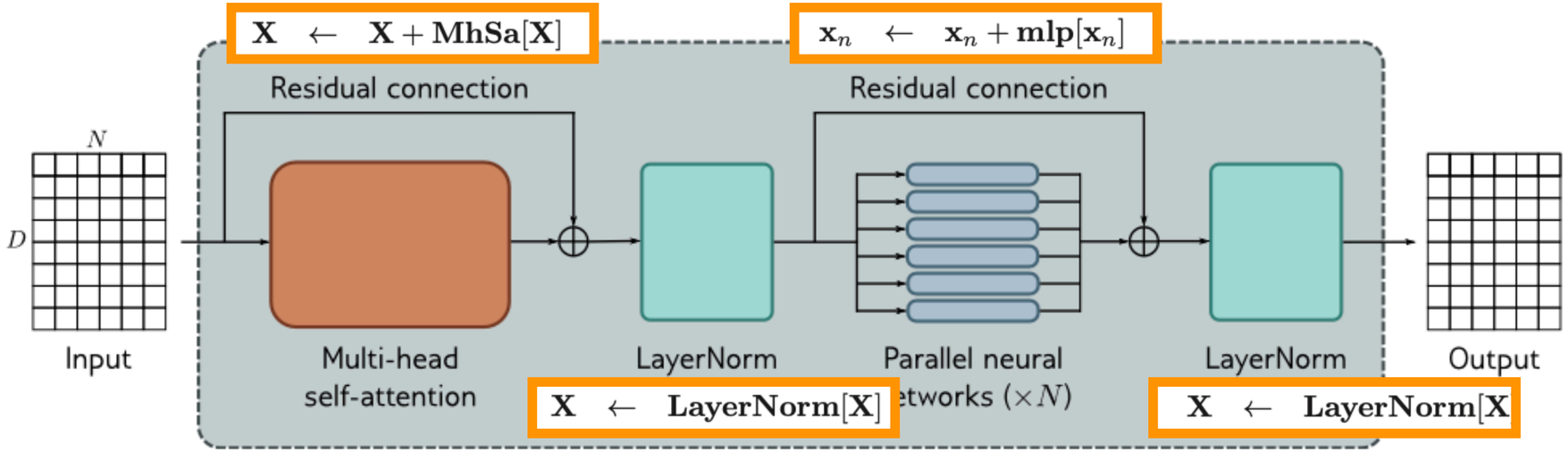
$$\mathbf{Sa}_h[\mathbf{X}] = \mathbf{V}_h \cdot \mathbf{Softmax}\left[\frac{\mathbf{K}_h^T \mathbf{Q}_h}{\sqrt{D_q}}\right]$$

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



Transformer Layers

- *Self-attention is one ingredient of the transformer layer*
- *A typical transformer layer*
  - *Uses residual connections for each operation*
  - *Starts with multi-head self attention*
  - *Process the output applying the same DNN to each word*
  - *Typically, one adds a Layer Normalization operation, 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 tokenizer splits the text into words or word fragments*
  - *Each token is embedded to a vector*
  - *the embedded vectors are processed by a chain of transformer layers*

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

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) loosing 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_

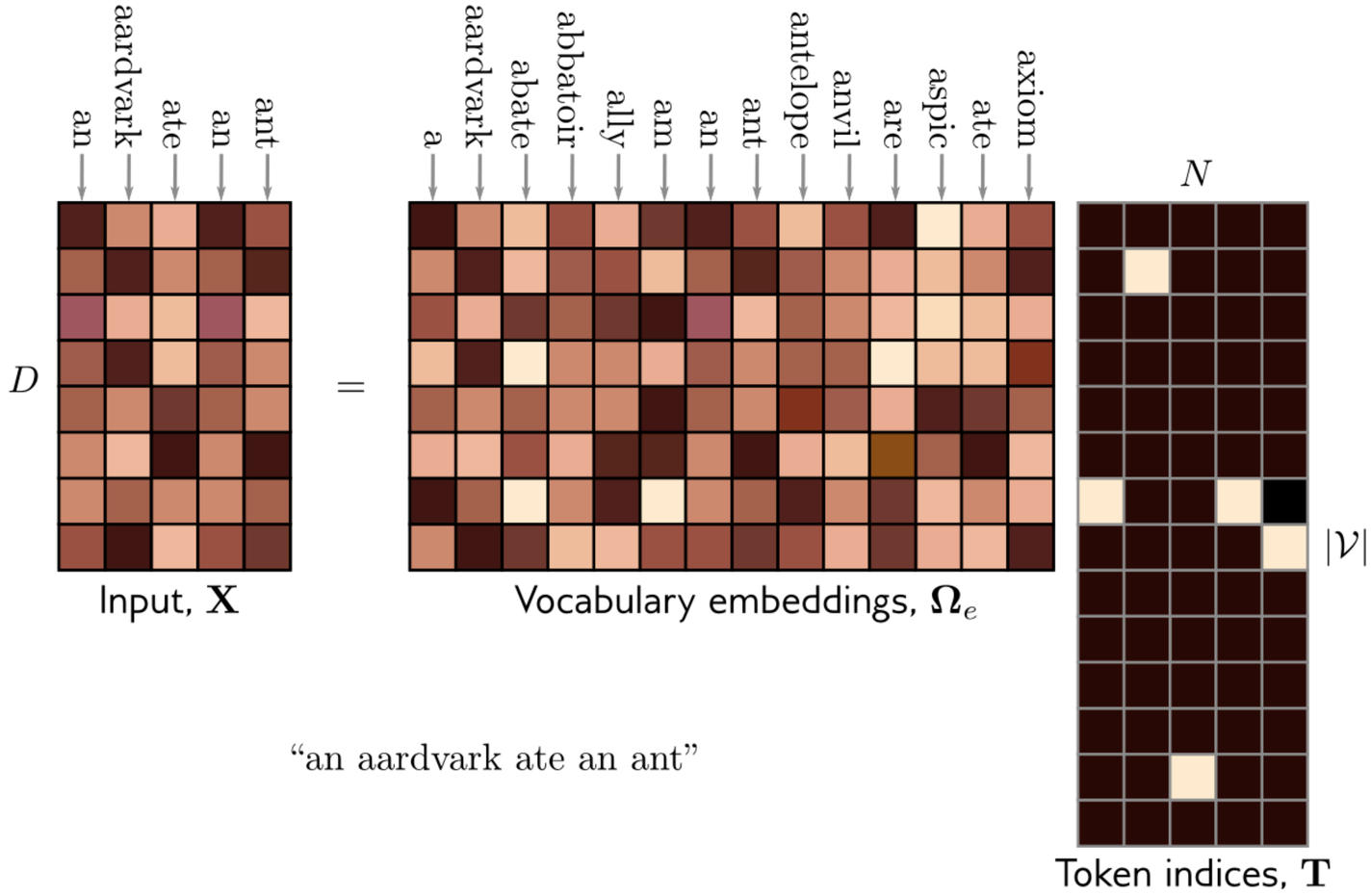
_	e	se	a	t	o	h	l	u	b	d	w	c	s	f	i	m	n	p	r
33	15	13	12	11	8	6	6	4	3	3	3	2	2	1	1	1	1	1	1

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	1

Embedding

- *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 NXD matrix*

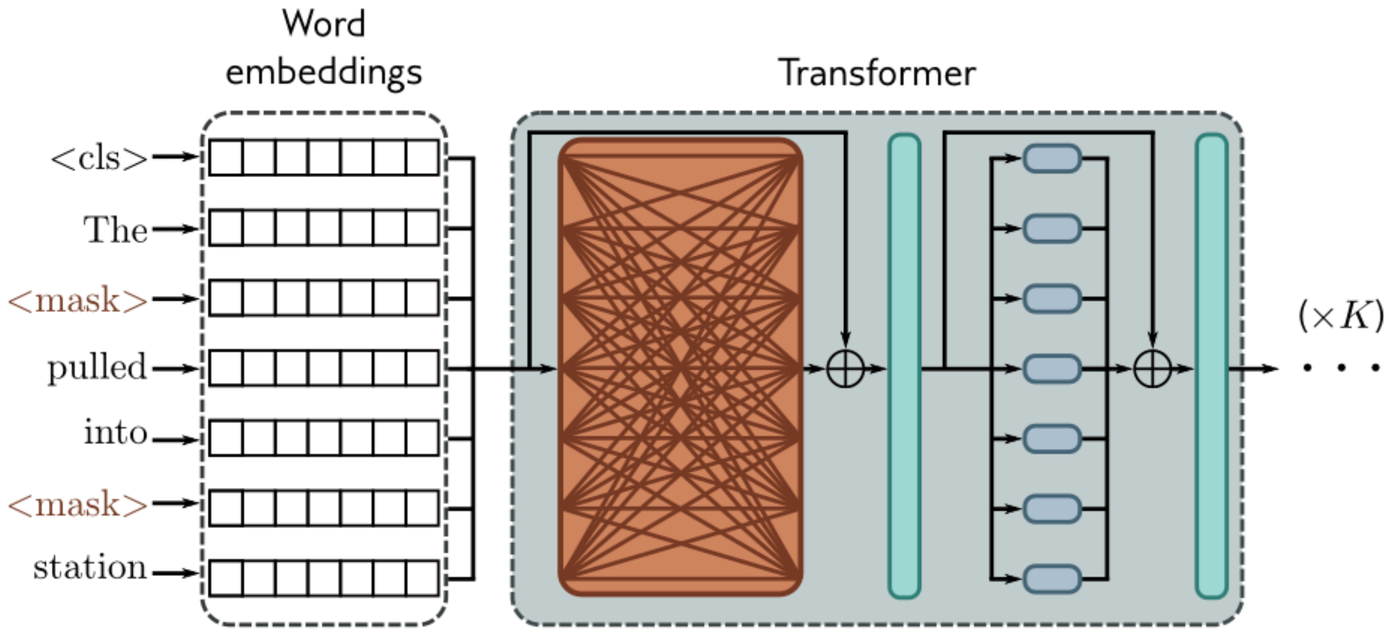


Transformer models

- *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*

BERT

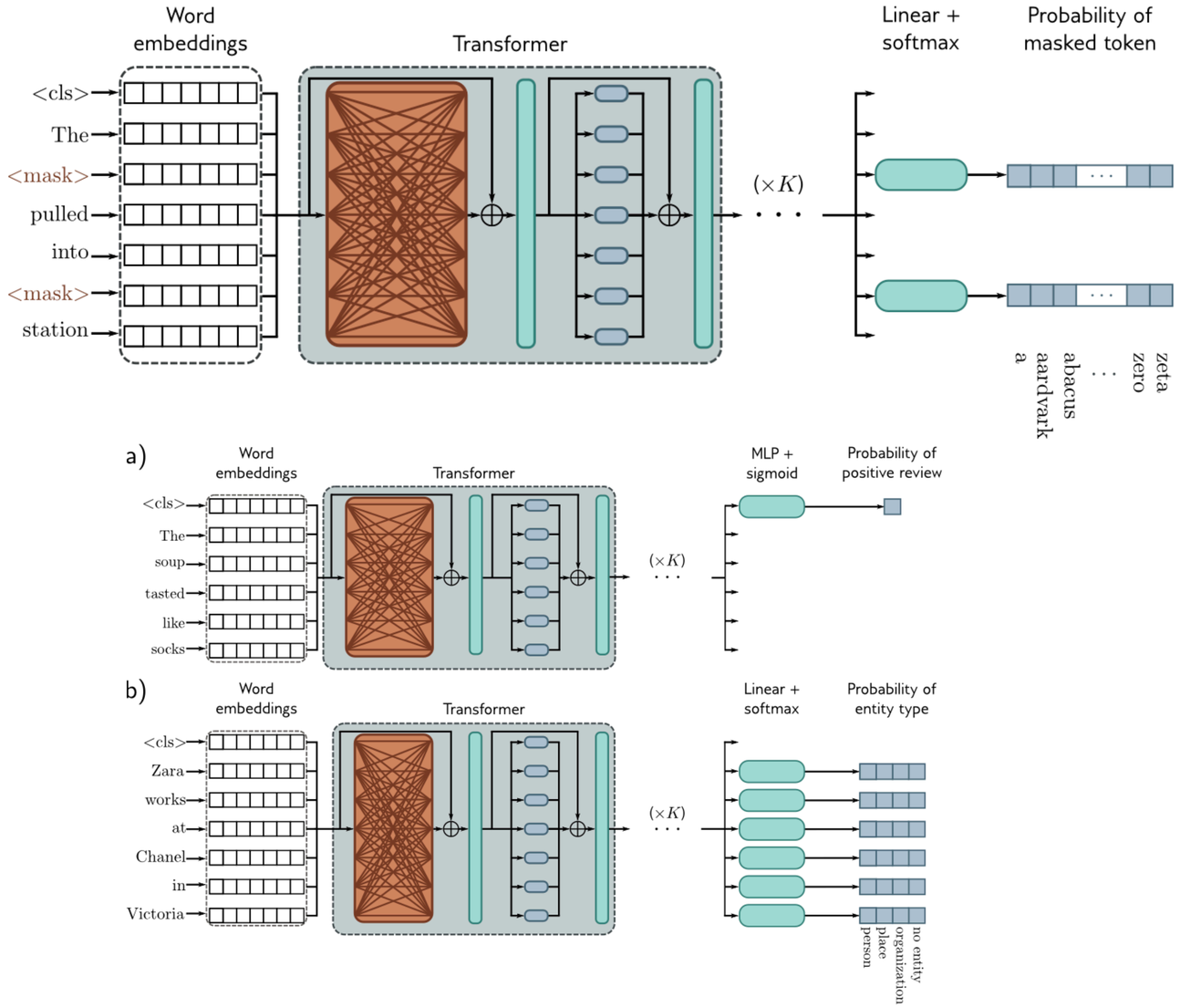
- *Vocabulary of 30K tokens*
- *Inputs converted to 1024 one-hot vectors*
- *The embedded input is processed by a chain of 24 transformers*
  - *16 heads of attentions*
  - *values, keys and queries have dimension 64 (i.e, the matrices in the transformer layer are 64x1024)*
  - *the hidden layer in the fully connected network has 4096 parameters*



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

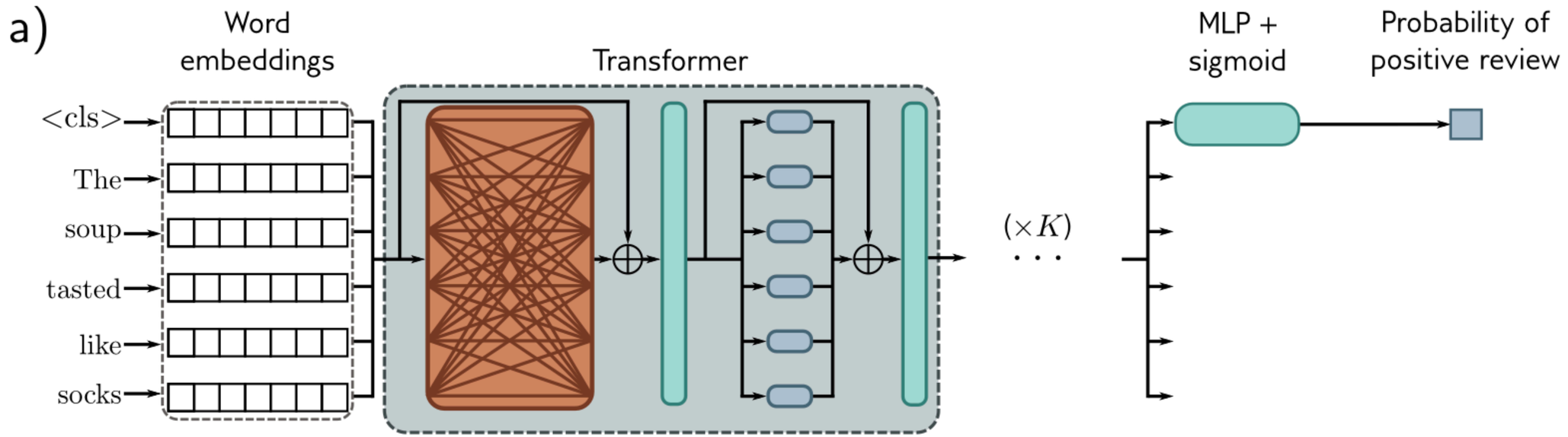
BERT

- *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*



GPT

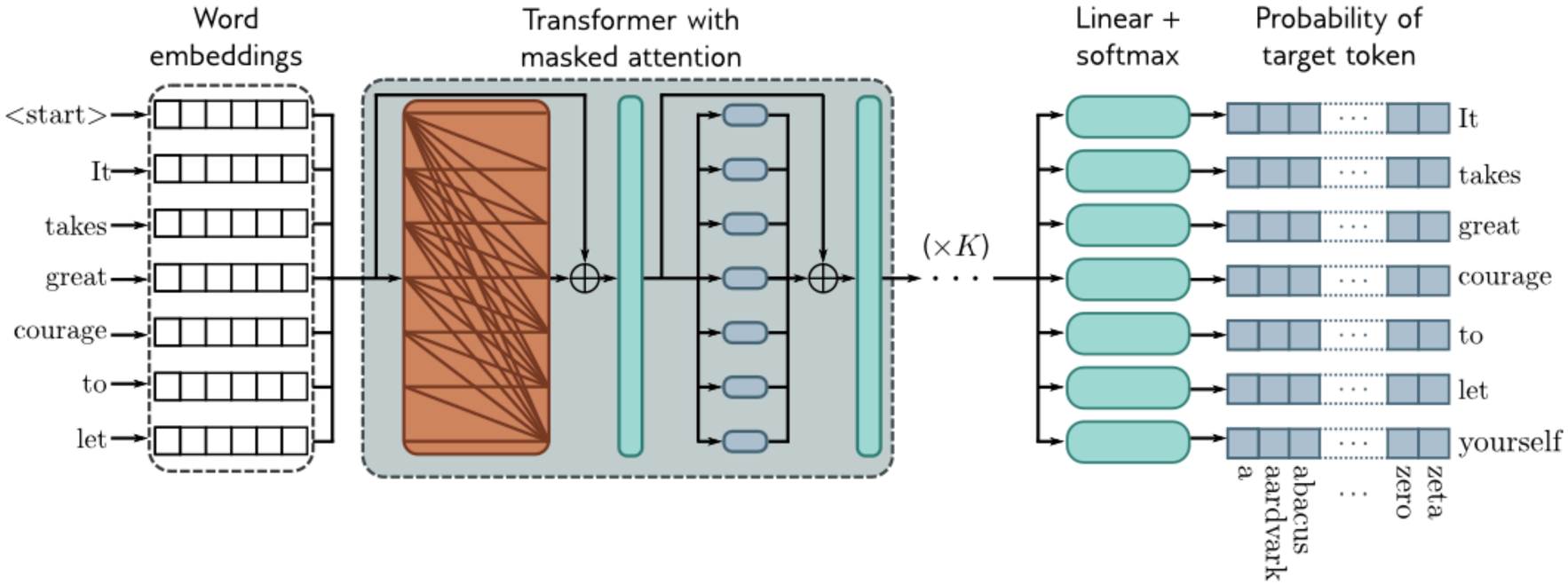
- *Generative Predictive Transformers (GPTs) are the models behind ChatGPT*
- *Structure similar to encoders, but opposite purpose*
  - *Start from embedded text to generate coherent text*
  - *It uses an autoregressive approach:*
    - *Write probability of a sentence as product of token probabilities*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

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

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

GPT

- *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

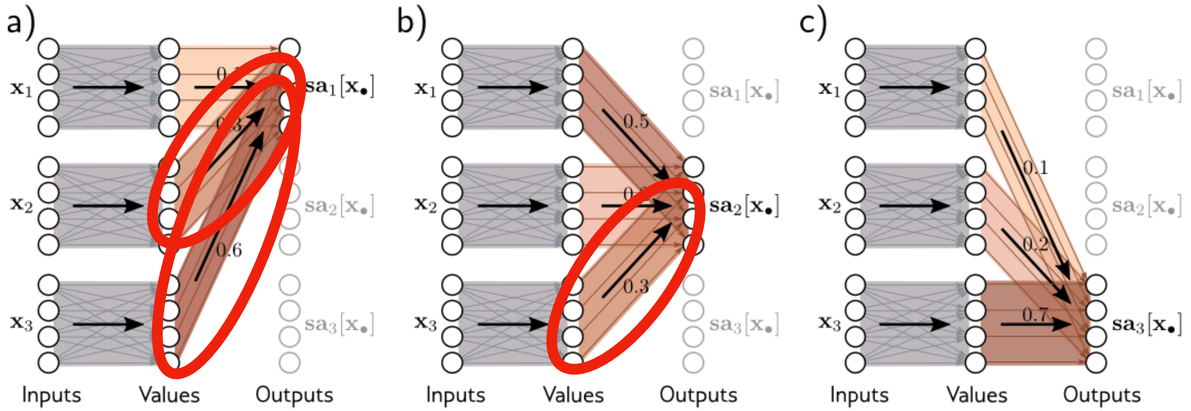
- *Training a decoder is tricky*
  - *Doing it sequentially (maximising next-word probability one by one) would take forever*
  - *Parallelising naively will lead to network cheating (the answer is provided by the rest of the sentence as part of the training dataset)*
- *An efficient training can be 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

$$\log\left[Pr(\text{great}|\text{It takes})\right]$$

great

courage to let yourself appear weak.



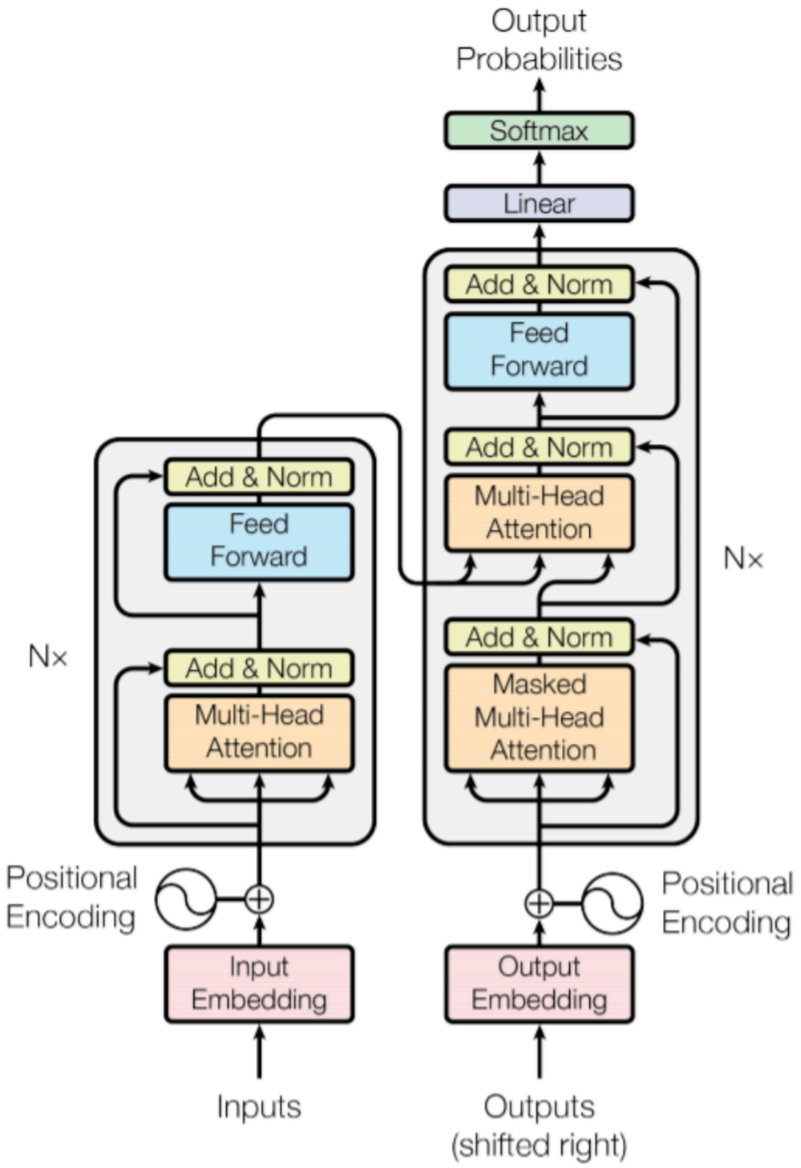
Masked self attention forces all upward pointing rerouting to be zero

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*

Text given to the model
Text generated by the model

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.

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:

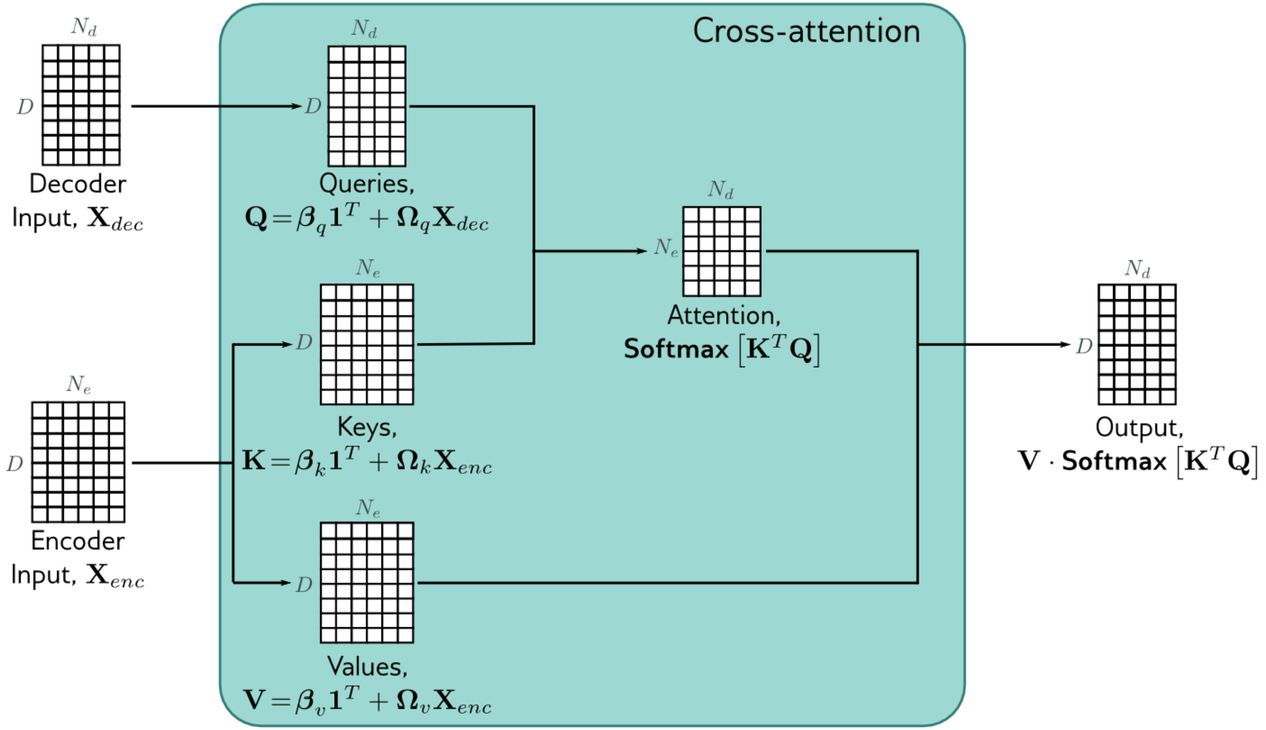
1. Introduction to deep learning
2. Deep learning architecture
3. Deep learning algorithms
4. Applications of deep learning

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.

Contents

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)*

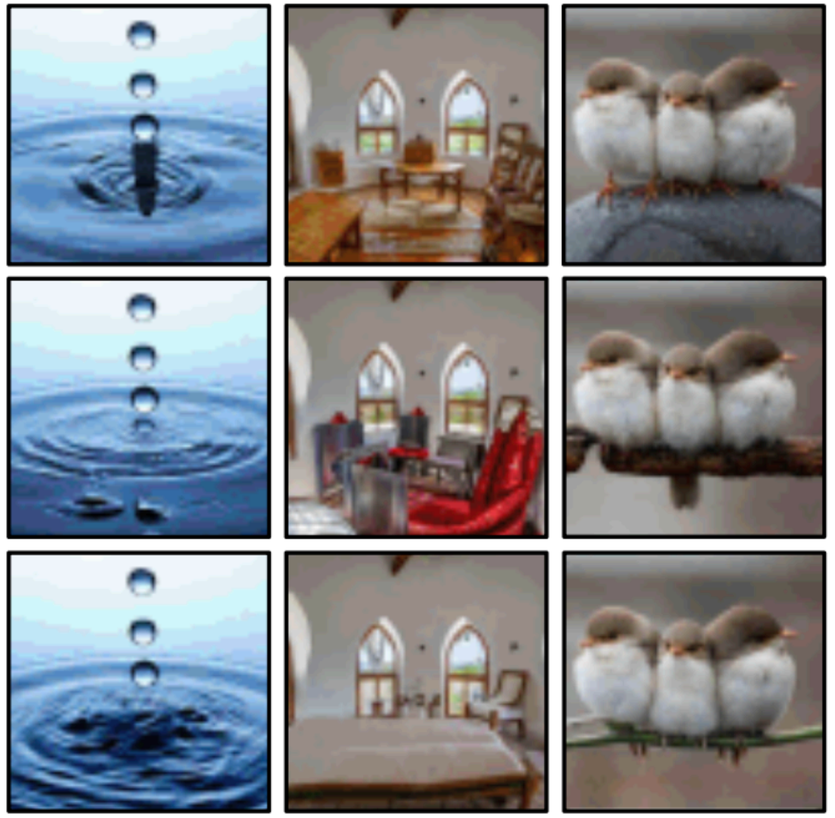


Processing Images

- *Success with text pushed people to try Transformers with images*
- *Not obviously a good idea a priori*
  - *No inductive bias in the architecture (i.e., architecture does not have the translation-invariant nature of the task enforced in the architecture)*
  - *Heavier computation (more pixels in images than tokens in a sentence) vs transformer quadratic scaling with sequence length*
- *Still, results were encouraging*

Partial input

Three different image completions by ImageGPT



Processing Images

- *ImageGPT has scaling limitations, due to quadratic dependence of Transformer computation on sequence length*
  - *It learns to predict pixels from previous ones*
  - *It needs to learn dependence from all neighbours, not just the previous one*
- *To work, it had to be limited to low resolution*
  - *64x64 pixel images*
  - *9-bit colors instead of 24-bit*
- *Did not improve over CNNs*
- *ViT solved the ImageGPT resolution issue partitioning the image in 16x16 patches*
  - *Embedding pre-training in this case is supervised (image classification on a large dataset)*
  - *Still not better than CNNs*