CS395T: Foundations of Machine Learning for Systems Researchers

Fall 2025

Lecture 4: Attention, Transformers, and Large Language Models (LLMs)

Some material taken from Gianfranco Bilardi's talk at Boost'24 and Sebastian Raschka's blog.

Large Language Models: Applications

- Machine translation
- Program generation
- Chatbots
- Document summarization
- AI agents

Semantics (meaning) seem essential for these tasks.
Amazingly, they can be done without explicit notion of meaning!

Three Major Discoveries

1. Semantics of words and sentences can be captured by vectors of real numbers
2. Efficiently computable *statistical* models for natural languages
3. Probabilistic generation produces meaningful text

History

Exploiting statistical properties of language:

Cryptography: reconstruct a secret code (from al-Kindi, c.801-873 A.C., and Caesar's (100-46) codes, to computational cryptography (1970s-))

Compression: encode a text so that it occupies less space to save memory and bandwidth (information theory Shannon (1916-2001), source coding).

Error correction: encode a text to facilitate detection and elimination of alterations (Shannon, channel coding).

Main concepts

Statistical language models

Series of increasingly more sophisticated DNNs for translation

Embeddings: static and contextual (Zettlemoyer 2018)

Attention

Statistical Language Model

Token (word) vocabulary: $T = \{\tau_1, \ldots, \tau_{|T|}\}$

Context: sequence of tokens: T^*

$x_1 x_2 ..$ where $x_t \in T, \quad t = 1, 2, \ldots$

Language model: The conditional probabilities of the next token given a context

$$Pr[x_t | \mathbf{c}_1 x_1 x_2 \ldots x_{t-1}] = Pr[x_t | \mathbf{c}_t], \qquad t = 1, 2, \ldots$$

Text generation:

- Prompt is initial context $\boldsymbol{c}_1$ (such as <START>)
- Given context $\boldsymbol{c}_t$, use model to produce probability vector and sample to pick next word x_t
- Append x_t to $\boldsymbol{c}_t$ and repeat

Text translation:

- Prompt is sentence to be translated followed by <START>

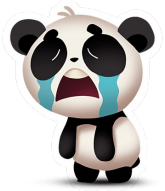
Building the language model

Analyze huge corpus of text and build table

context → probability vector ($[p1, ..., p|T|]$)

Oxford English Dictionary has > 200K words ($|T|$)

Number of probabilities for context of length t = $|T|^{t+1}$



Solution: train NN to *compute* probability vectors

Running example: English→Turkish

Source language: English, |S| words

Target language: Turkish, |U| words

Training data: English-Turkish sentence pairs

The cat ate the mouse .	Kedi fareyi yedi .

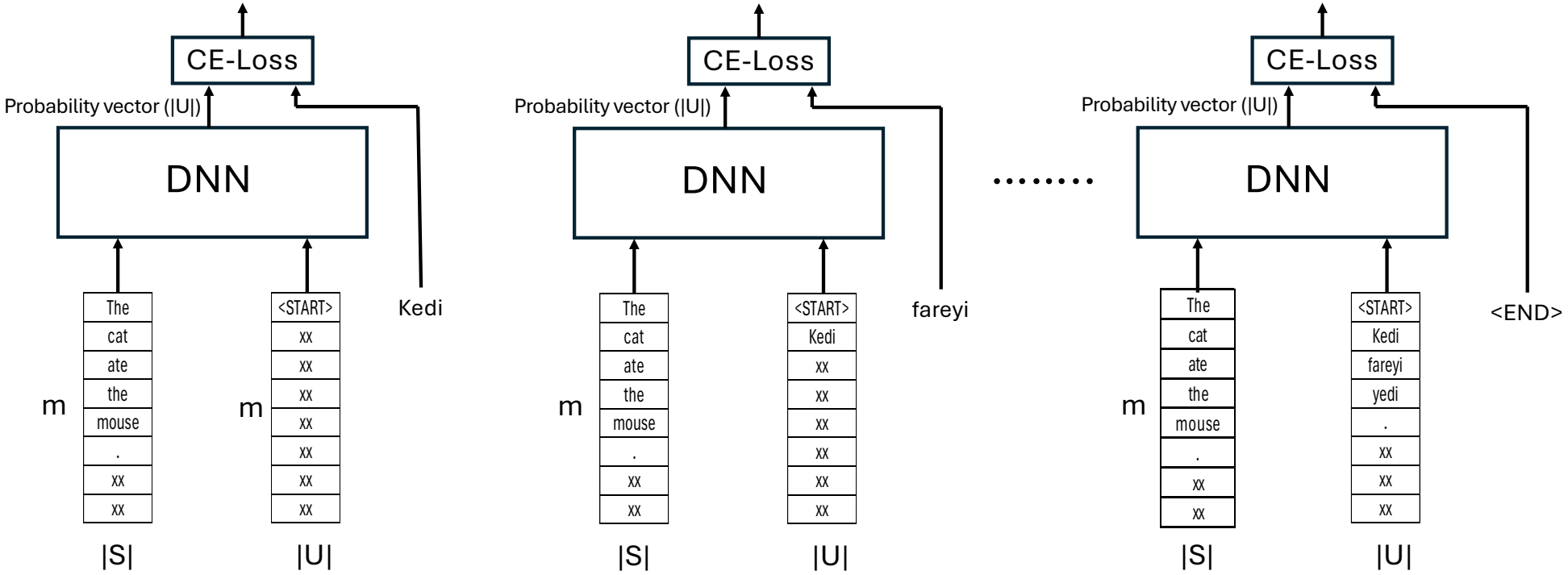
First attempt:

Encode words as one-hot vectors

Assume maximum sentence lengths in both languages is *m*.

Pad sentences (we will use xx) to extend all of them to length *m*.

DNN1 (Training)



Example: Kedi fareyi yedi .

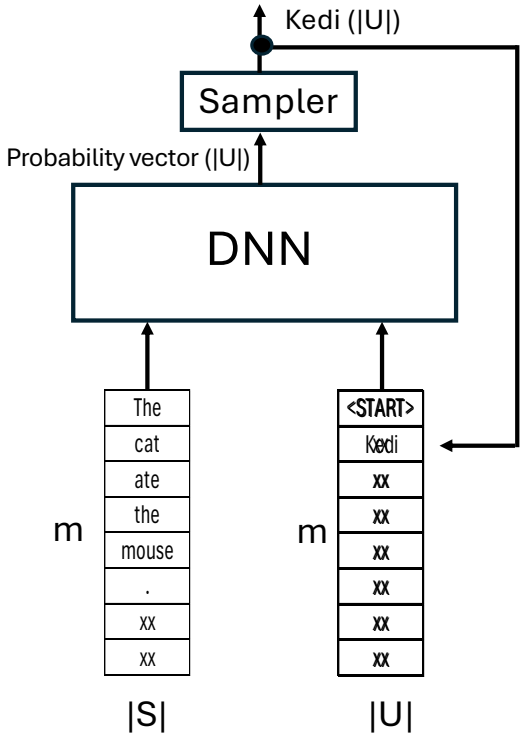
One-hot encodings
of source & target words

Teacher forcing: predicted word is thrown away

DNN1 (Inference)

Autoregressive decoding:

- translated text produced sequentially
- translated text begins with <START>
- each word produced by DNN is appended to input to predict next word



Drawbacks of one-hot encodings

Inefficient: curse of dimensionality

|S| and |U| are huge: > 200K

Does not generalize

Unclear what happens with word not seen during training

Solution: vector-space models of language (embeddings)

Embeddings

Central Idea in Modern NLP

"You shall know a *word by the company it keeps*."

J.R.Firth, English linguist (1957)

Firth's work: basis for field of "*distributional semantics*"

– "*linguistic items with similar distributions have similar meanings.*"

Led to vector-space representations (embeddings) of words, sentences, documents, etc.

JOHN RUPERT FIRTH

BSOAS. XXIV]

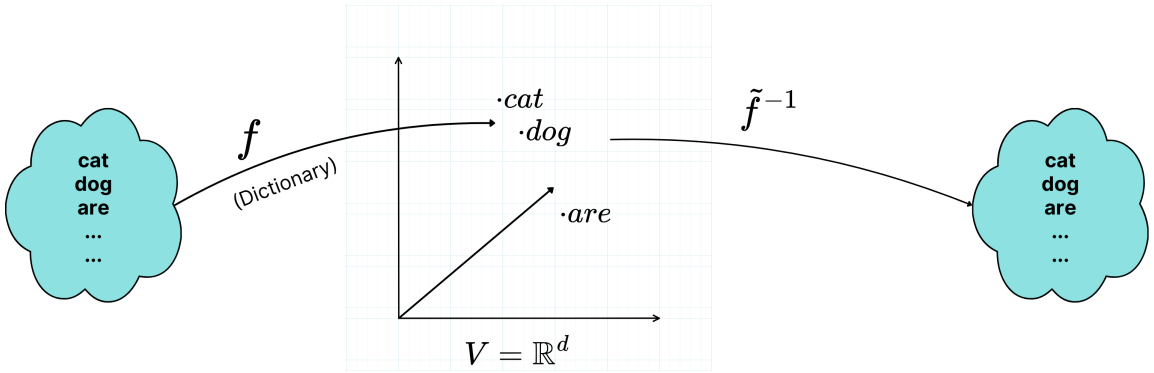
Vector-Space Embeddings

Embedding maps token (word) to *d*-dimensional *row-vector* of reals
(*e.g.*) GloVe: Manning et al. 6 billion tokens, *d* = 50,100,200,300

f : one-hot encoding $\longrightarrow$ embedding

$O \cdot W_f^{(|T| \times d)}$

$\tilde{f}^{-1} =$ $V = \mathbb{R}^d$ $W_b^{|d| \times |T|}$ $\mathbb{R}^{|T|}$ **softmax** $\pi^{|T|}$ Probability vector **Sampler** **word**



Semantic feature space

Three central assumptions about embeddings

Principle of compositionality: representation of context is function of its word embeddings

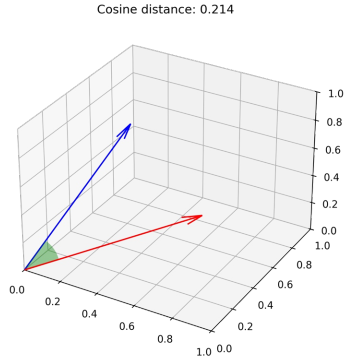
$$\boldsymbol{F}(\mathbf{c}) = \gamma(f(x_1), f(x_2), \ldots, f(x_k))$$

Probability vector for next word for given context is function of representation of context

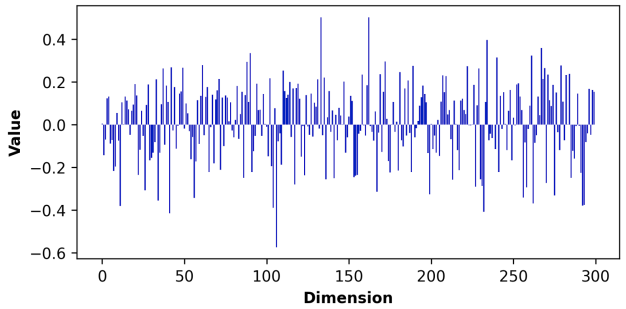
$$[p1, p2, \ldots p|T|] = \beta(F(\mathbf{c}))$$

Similarity/relevance of word *y* to word *x* is function of embeddings (cosine similarity, Euclidean distance, etc.)

$$A(x,y) = \alpha(f(x), f(y))$$

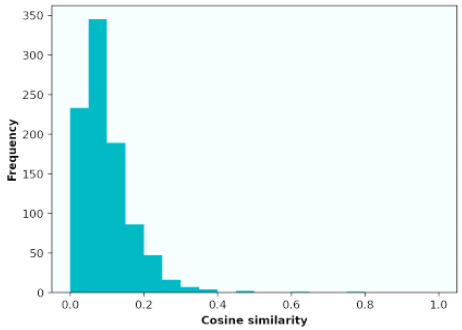


Word2Vec Embeddings



Embedding of “queen”

word	similarity
dog	1.000000
cat	0.760946
animal	0.643801
horse	0.482581

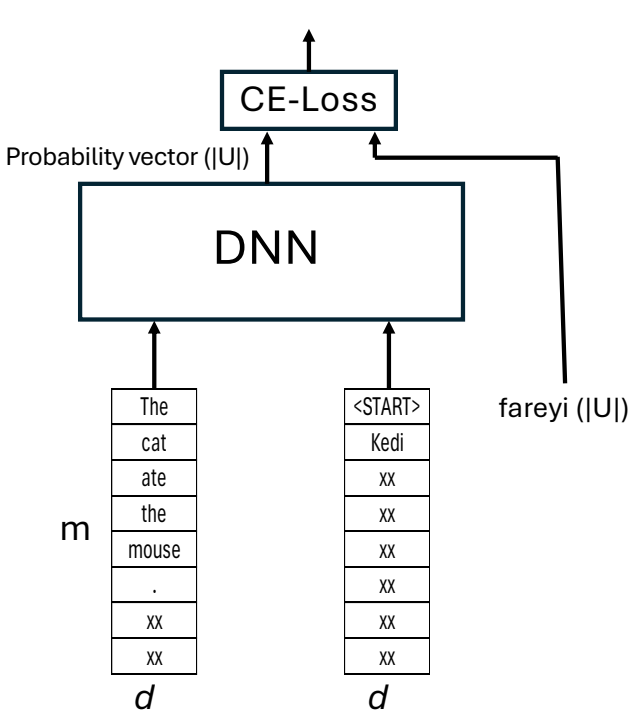


Pamela Fox: An introduction to Vector Embeddings

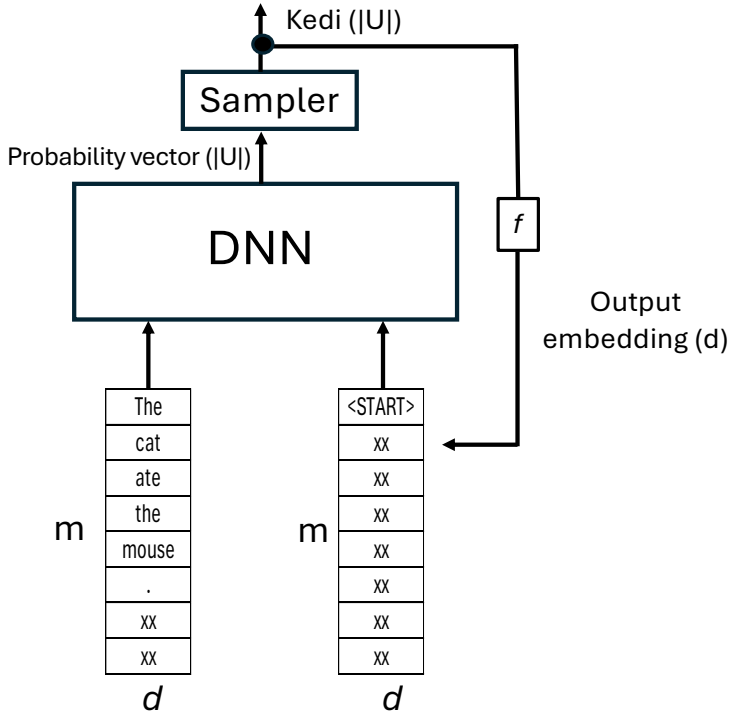
DNN2: Same as DNN1 w/embeddings

β: representation of context → probabilities

γ: representation of context is concatenation of embeddings of its words



Training



Inference

Contextual Embeddings

Why contextual embeddings

Word embeddings from GloVe, Word2Vec etc. are *static*

- do not depend on task
- do not change during training/inference

Embeddings in transformers are *contextual* (Peters et al. 2018) and *learned*

- handle polysemy (*e.g.* "I *bank* only at my local *bank*.")
- contextualized using *attention*
- initial embeddings for training can be random since they are learned

Easier to understand if initial embeddings come from GloVe etc.

- "Teaching is the art of telling fewer and fewer lies."

How to embed context in **V?**

Design γ so that it returns an embedding in **V**

One idea: some linear combination of GloVe word embeddings

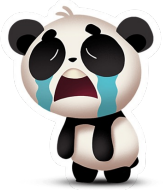
$$\boldsymbol{F(c)} = \sum_{j=1}^{k} a(j) * f(xj)$$

Embeddings squished together so you lose most information

To retain information, give higher weights (*a(j)*'s) to important words

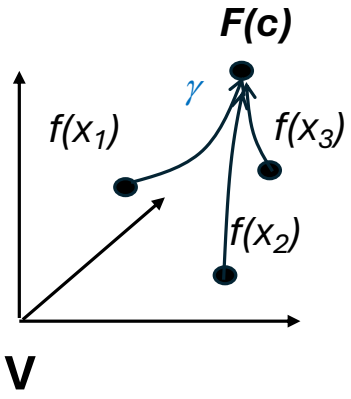
What is important in sentence (*e.g.* "I bank at the local bank") for one word may not be important for other words

First "bank" needs "I" to resolve its part of speech
Second "bank" need "local"



Context: $x_1\ x_2\ x_3 \ldots x_k$

$\boldsymbol{F(c)} = \gamma(f(x_1), f(x_2), \ldots, f(x_k))$



Refinement of idea

Embed context *from perspective of each* x_i

$\boldsymbol{F(c;x_i)} = \sum_{j=1}^{k} a(i,j) * f(xj)$ I bank only at my local bank

Putting all $\boldsymbol{F(c;x_i)}$ vectors together

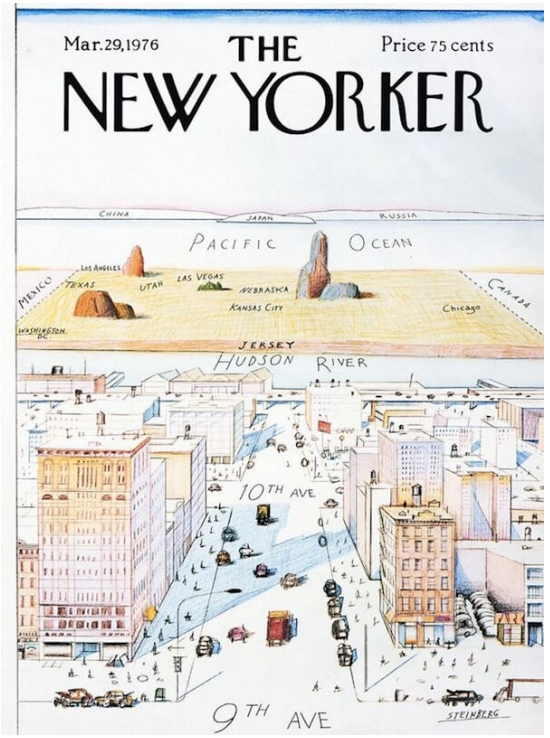
$Z^{mxd} = A^{mxm} X^{mxd}$ where X is matrix of word embeddings

Intuitively, combines strengths of

DNN2 (one embedding per word)

RNN (one embedding for entire context)

What should the A matrix be?



Choices for A matrix

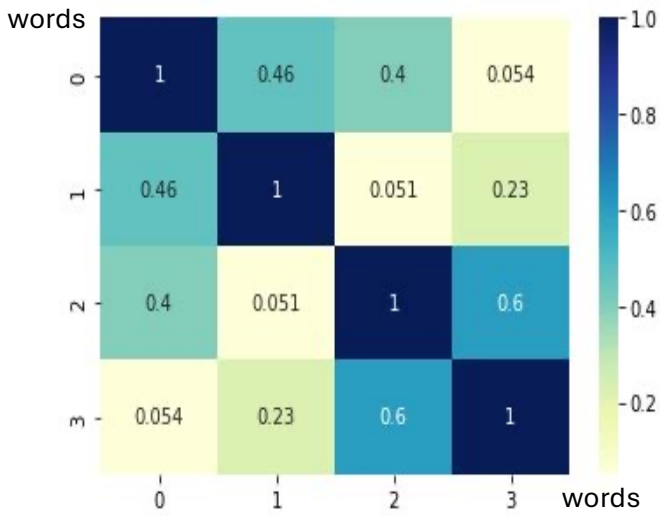
Find cosine similarity between embeddings of each pair of words in sentence

Useful to visualize as matrix

Normalize values in each row using softmax

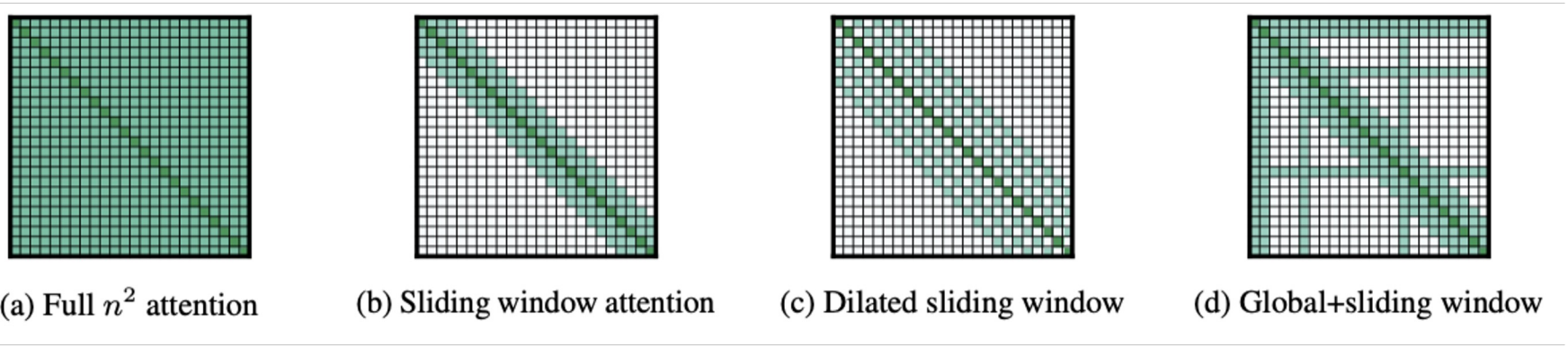
Attention matrix A^{mxm}

Detail: transformers compute cosine similarity (or dot products) using new vectors called Query and Key generated by linear transforms on embeddings (ignore for now)



Cosine similarity matrix for four word sentence

Attention Variants



(a) Full n^2 attention (b) Sliding window attention (c) Dilated sliding window (d) Global+sliding window

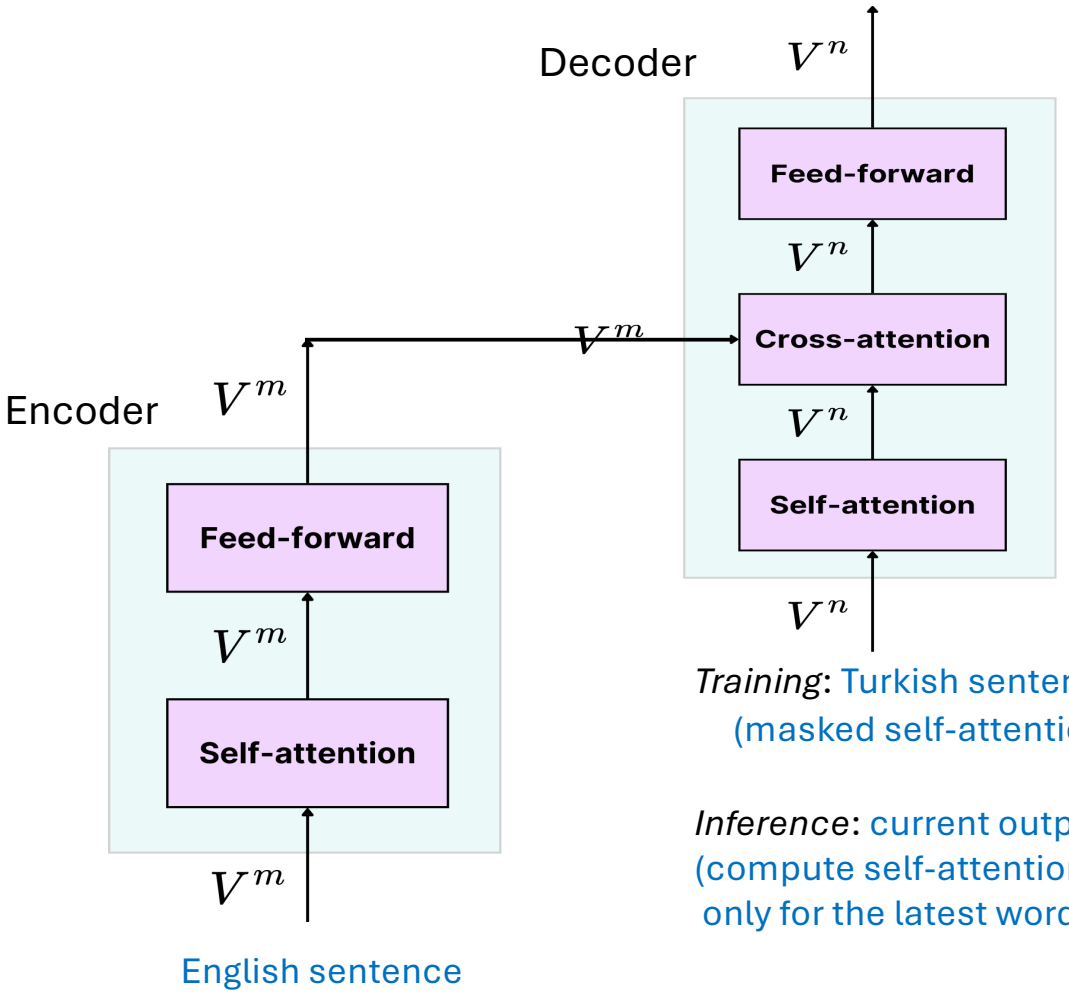
Attention terminology

Self-attention: between words in same context

Cross-attention: between words in different contexts

Masked self-attention: word attends only to words before it in sequence (attention matrix is lower triangular)

Implementation of Attention in Transformers (Vaswani et al.)



Training: Turkish sentence (masked self-attention)

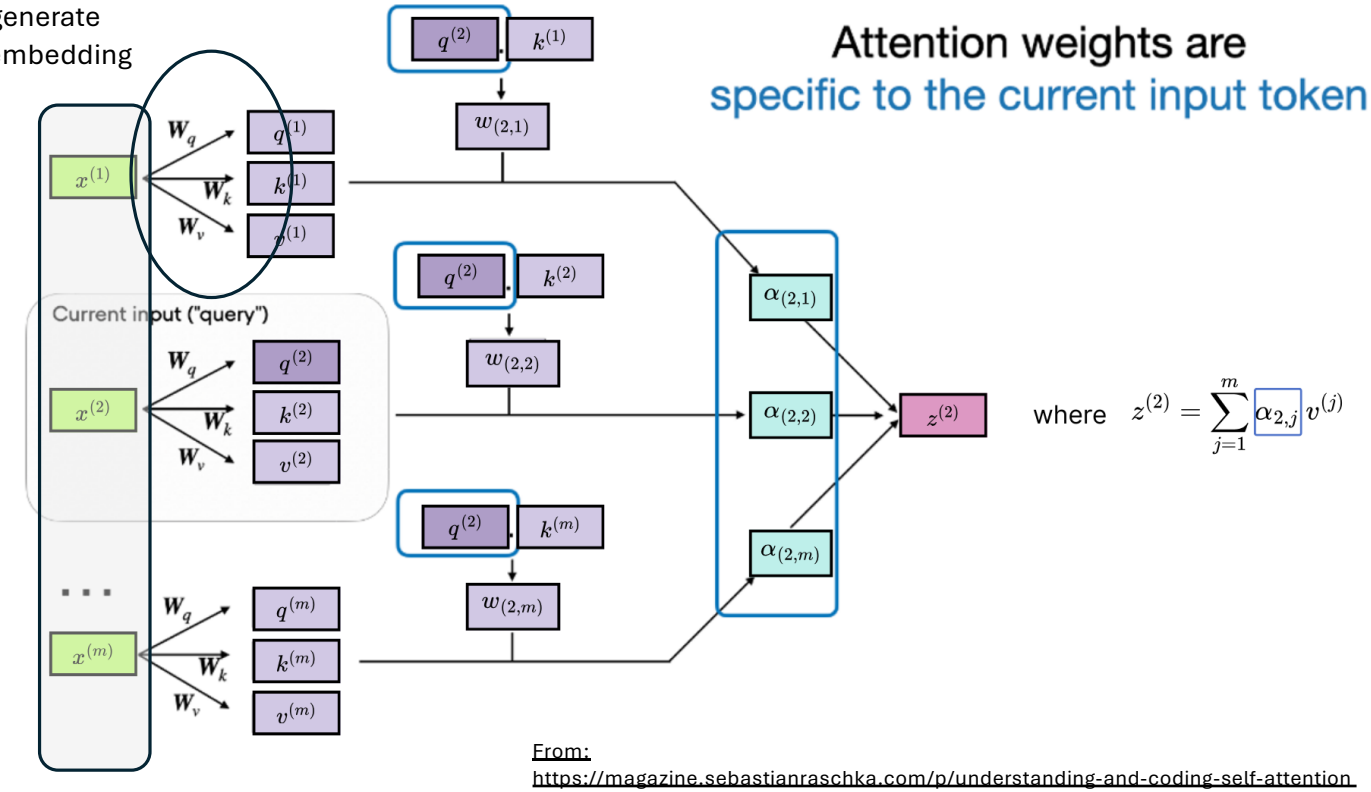
Inference: current output (compute self-attention only for the latest word)

English sentence

Encoders and decoders can be stacked (blocks/layers)

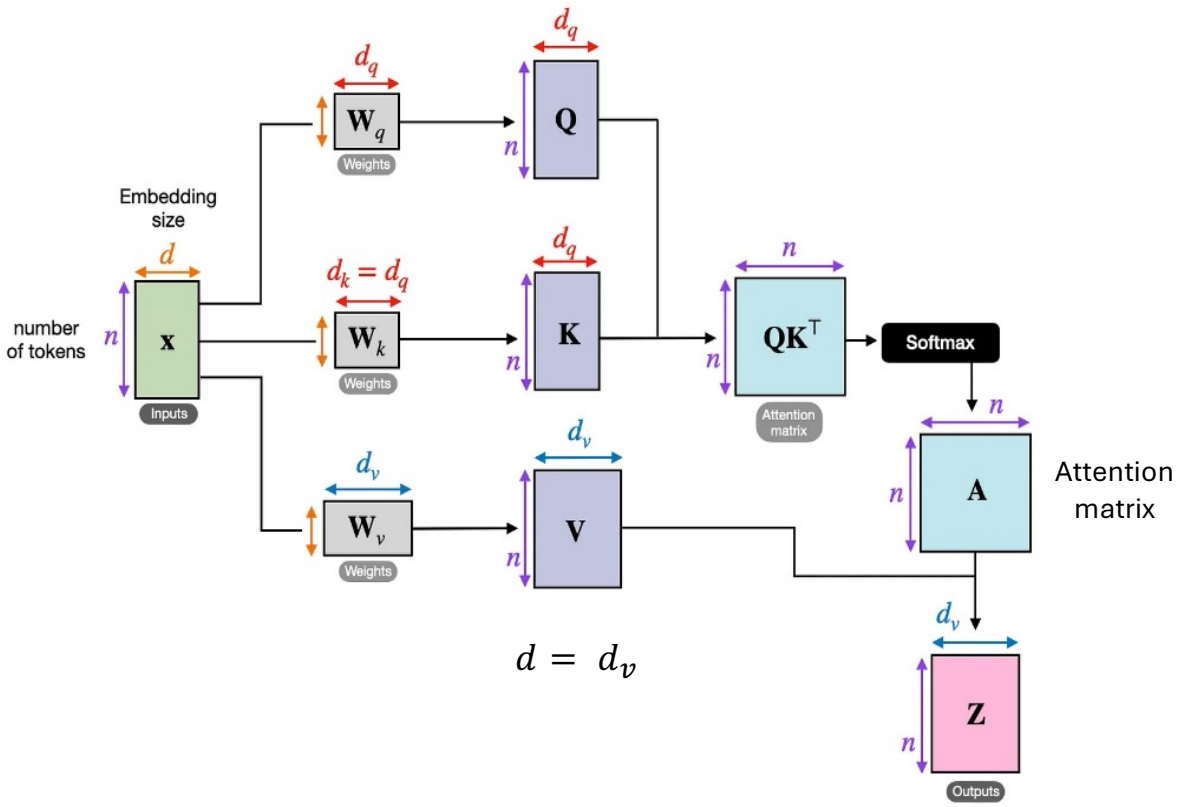
Encoder Self-attention

Linear transforms to generate Query, Key, Value from embedding



From:
https://magazine.sebastianraschka.com/p/understanding-and-coding-self-attention

Matrix view of self-attention

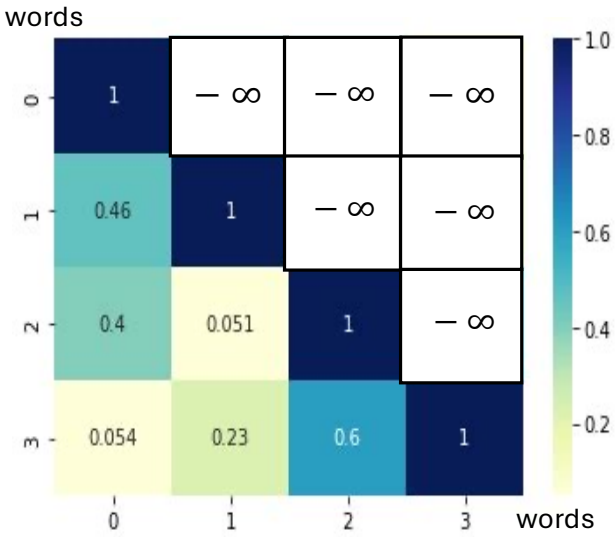


Decoder Self-Attention (training): Masked Self-attention

Entire output sentence is available but compute attention for a word using only words *before* it in sequence. Mimics what happens during inference.

- Find cosine similarity matrix
- Set values in upper triangle to -∞
- Normalize values in each row with softmax to get attention matrix A^{mxm}
- Compute contextual embeddings (Z) for all words

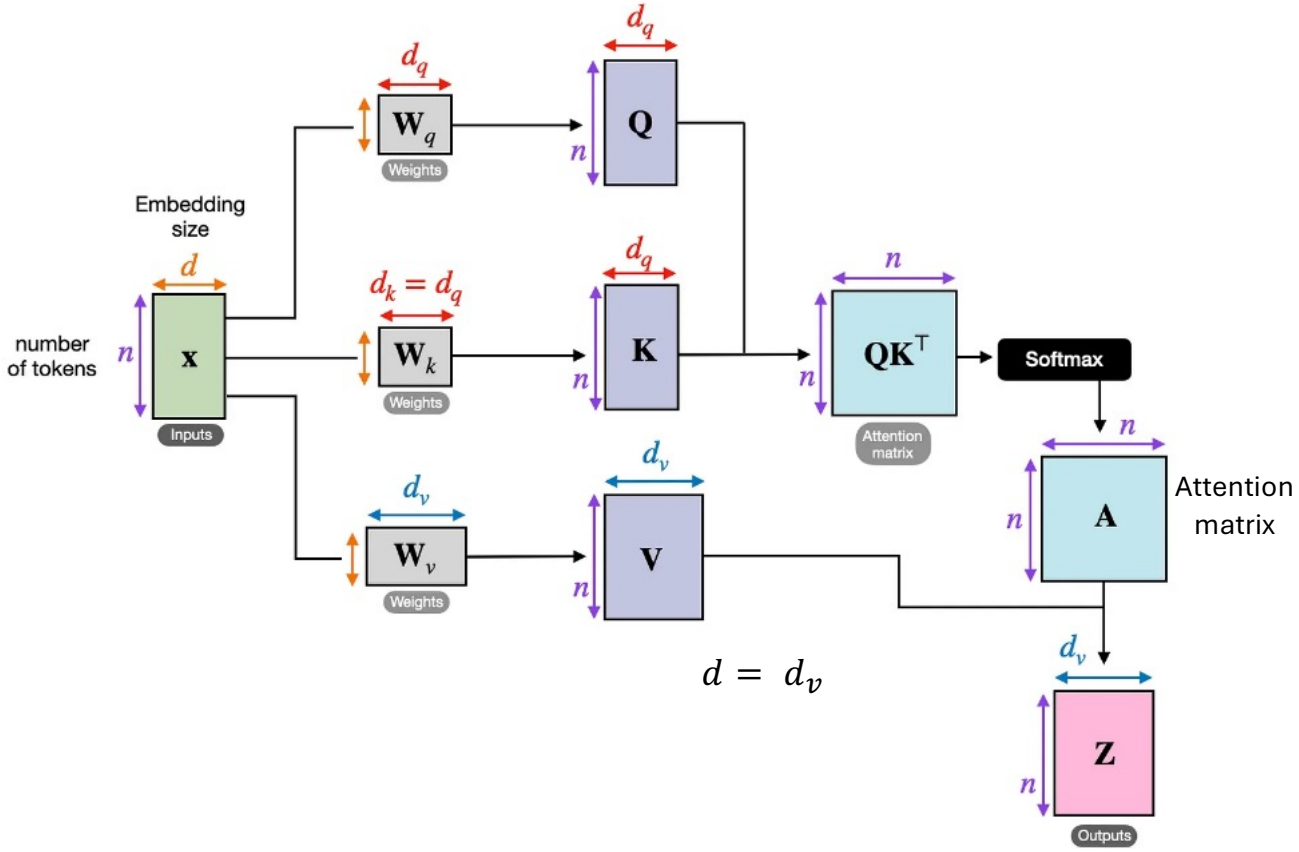
  $Z^{mxd} = A^{mxm} X^{mxd}$



Masked cosine similarity matrix

Cosine similarity matrix

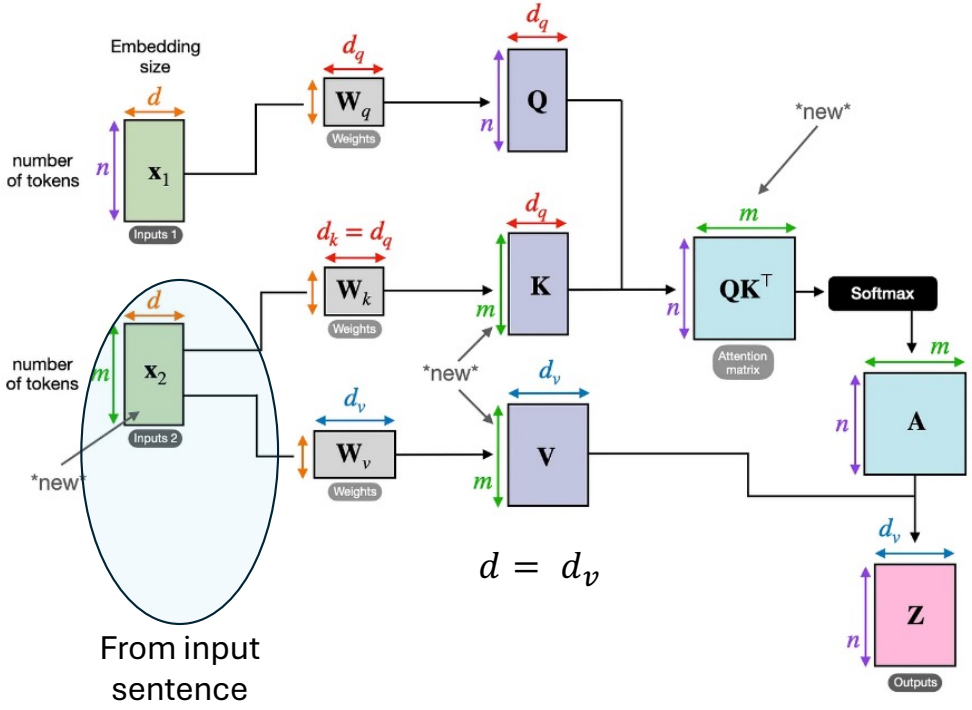
Decoder Self-Attention (Inference): KV-cache



In each round, compute self-attention only for latest token

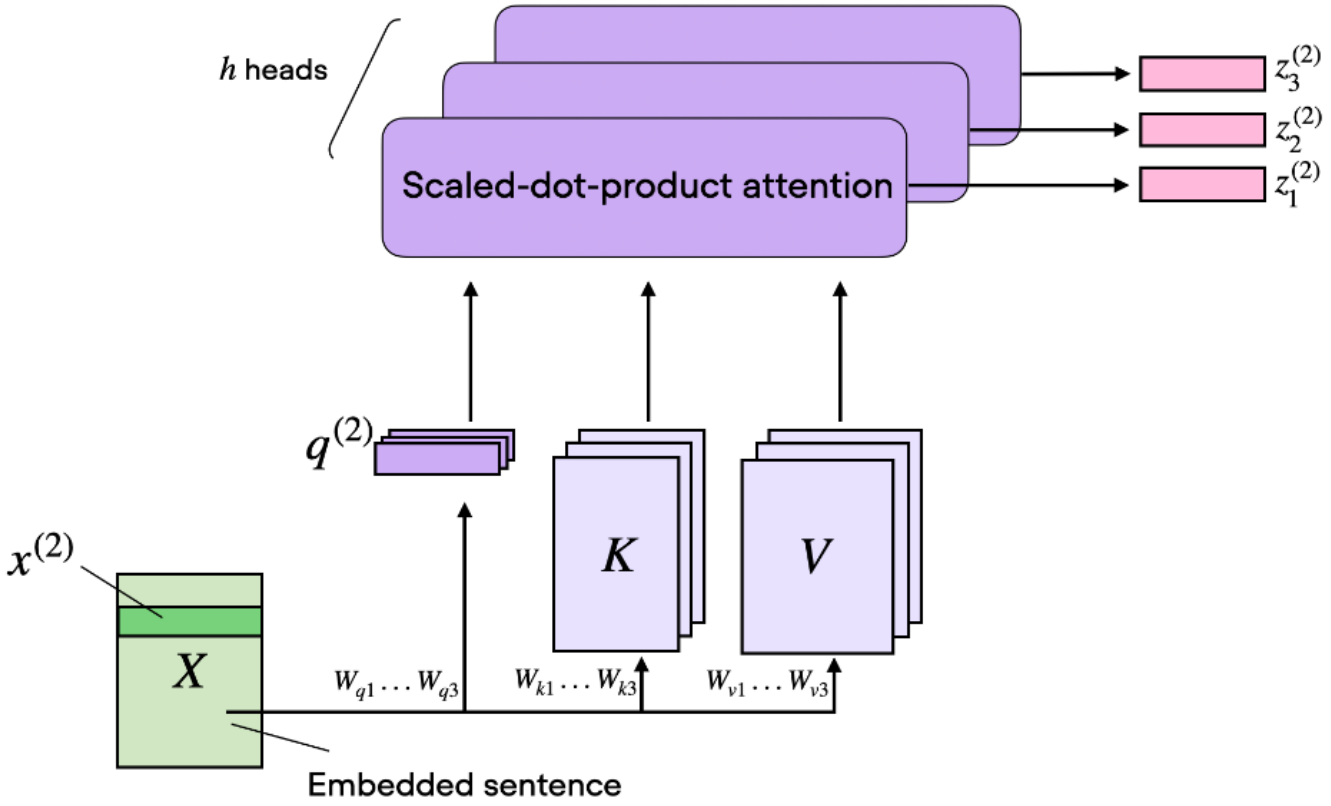
- KV-cache: keep K,V matrices and contextual embeddings for all previous output tokens
- Compute key, query, value vectors for latest token & update K,Q,V
- Compute contextual embedding for token and update Z

Cross-attention



Another design: send only $\mathbf{W}_k$ and $\mathbf{W}_v$ matrices from encoder to each decoder block/layer
Design shown in picture permits $\mathbf{W}_k$ and $\mathbf{W}_v$ matrices to be different in different layers

Multi-head attention

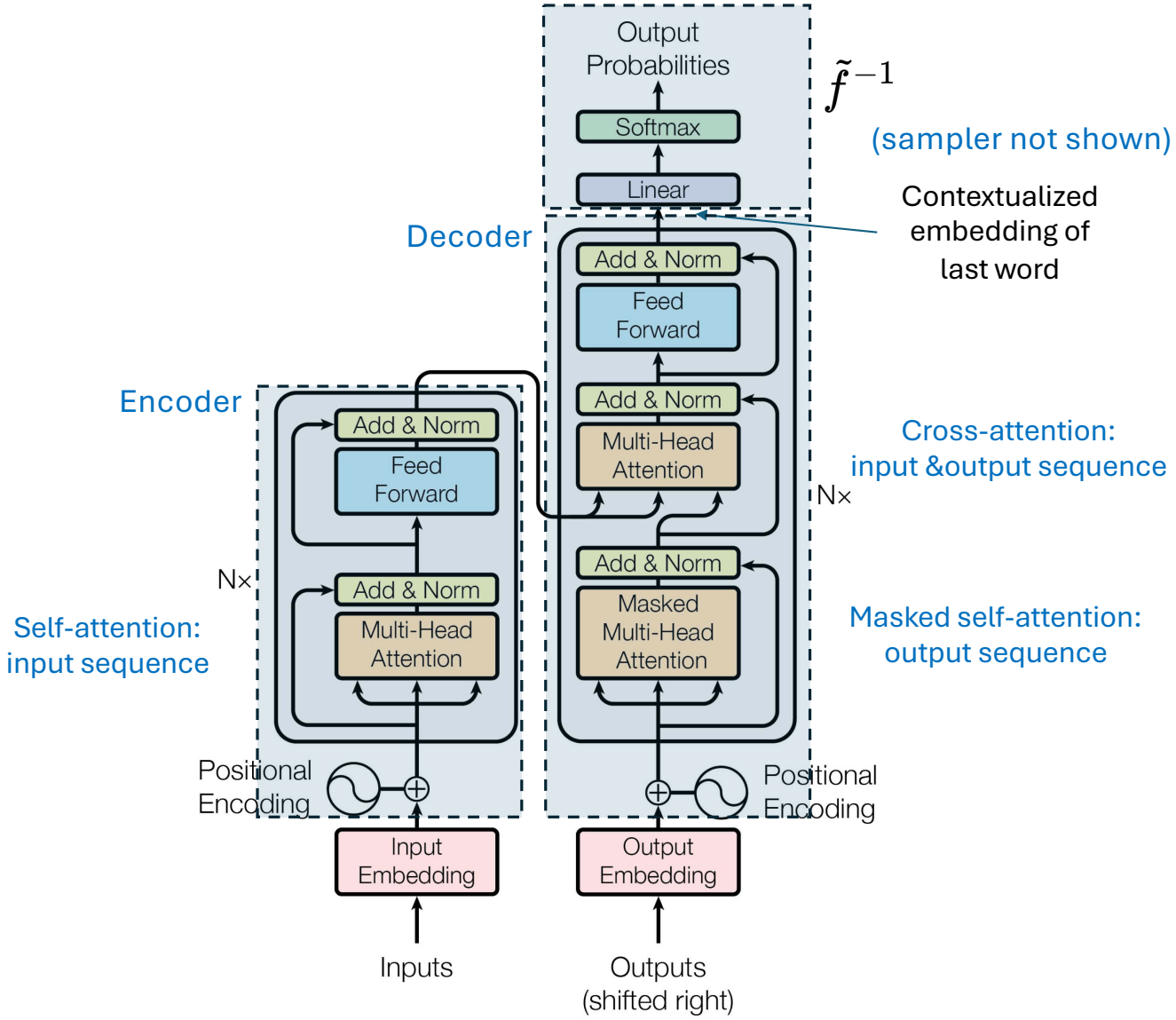


Divide embedding dimension *d* among *h* heads

Each *Q,K,V* etc. specializes on *d/h* dimensions

Concatenate outputs from heads to produce final contextual embeddings

Transformer for Translation (Vaswani et al.)



Computational Issues: Problem Size and Complexity

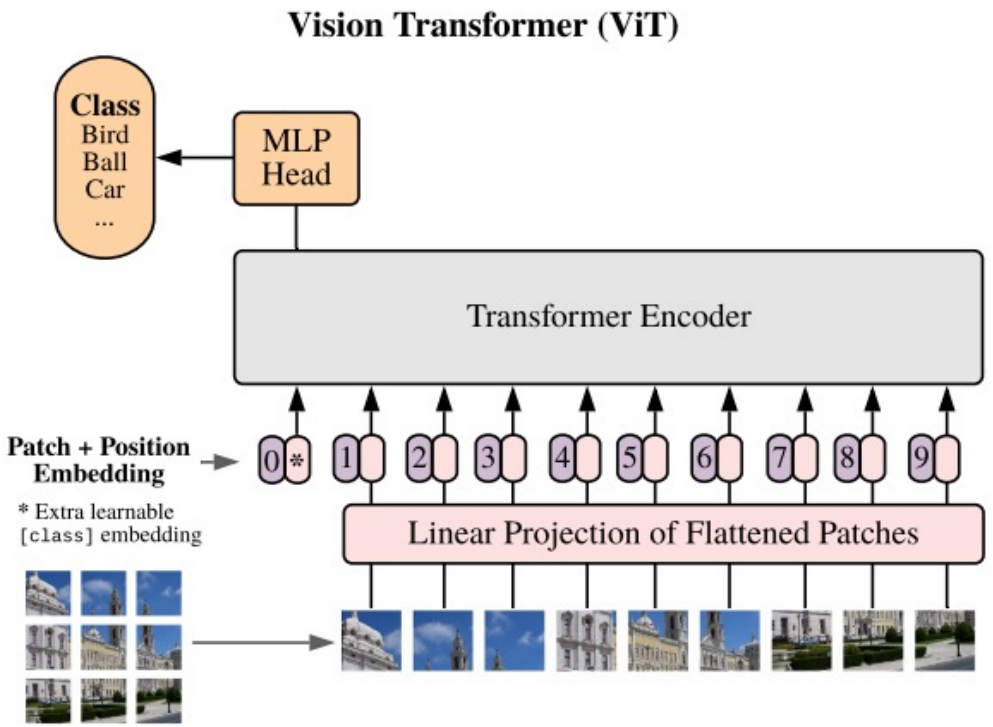
Hyper Parameters of (V)LLMs: representative values

- size of token dictionary $|T|$: $O(2^{17}) = O(100,000)$.
- embedding dimension d: $2^{13} + 2^{12} = 12,288$.
- size of input window n: $2^{15} = 32,768$.
- number of blocks N: $\geq 96 = 2^6 + 2^5$.
- number of attention heads H: $12,288 : 128 = 96$.

Learnable Parameters: $\mathbf{W}_{|T|}$ and, for each of the N levels, $\mathbf{W}_Q$, $\mathbf{W}_K$, $\mathbf{W}_V$, $\mathbf{W}_1$, $\mathbf{b}_1$, $\mathbf{W}_2$, $\mathbf{b}_2$.

- number of learnable parameters: $O(|T|d + 5Nd^2 + l.o.t.) \approx O(10^{11}–10^{12})$ *floating point* numbers (typically, 1FP = 2 bytes).
- size of training set: $O(10^{11}–10^{12})$ tokens (1 token ≈ 2 bytes).
- computational effort for training: ??
- computational effort for inference: ??
- training cost: > 10^8 USD for ChatGPT-4, according to Sam Altman (CEO of OpenAI).

Vision Transformer (ViT)



https://docs.nvidia.com/nemo-framework/user-guide/latest/nemotoolkit/vision/vit.html

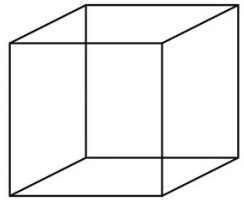
Final remarks

Omitted details

- Inner workings of Word2Vec, GloVe etc.
- Tokenization
- Positional encodings for tokens in transformer

Key insight: the point of attention

- Standard explanation: embedding (**V**) for word relative to context
- Better explanation: embedding (**V**) for context relative to word



Philosophical question

What does attention buy us over DNN2? Space-time complexity? Smaller amount of training data? More accurate inference?

Sutton's "Bitter Lesson" argues domain-specific knowledge (*e.g.*, human-generated heuristics for chess-playing programs) is not useful for ML in the long run. However, attention seems to be NLP-specific concept.

Does it come down to domain-specific *structure* (good) vs. human-designed *heuristics* (not needed in long run)? If so, is there other domain-specific structure to be discovered for making ML more "efficient" in some dimension?