ENSAE

Deep learning

Unpacking Transformers, LLMs and image generation

Session 2

Syllabus

Session	Lecture	TP
1	Intro to DL Neural networks 101	Intro to micrograd (Value class)
2	DL fundamentals • Backprop • Loss functions • Residual networks • Batch norm, LayerNorm • Initialization • Intro to Bengio et al 2003	Bigram model and MLP for next-character prediction (*) add batchnorm
3	Attention Transformers	Backprop ninja (*) Fleuret practical 3 (MLP for MNIST)
4	Transformers in the real world (from scratch starting from a bigram)	mini-gpt (training on GPU) Find a good open-source LLM and build a notebook running it on a CPU (*) Fleuret practical 5 and 6
5	DL for computer vision: convnets, unets	Convnets for Fashion MNIST (*) Fleuret practical 4
6	VAE & Diffusion models	1D diffusion model 2D diffusion model (*) train on GPU **Quiz**

onfido

Summary so far

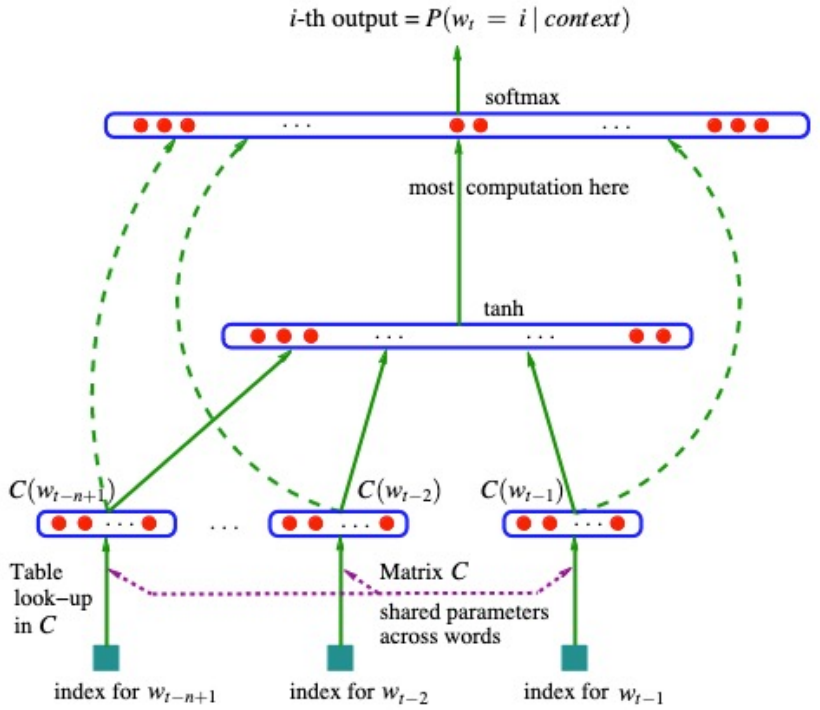
Deep learning fundamentals.

Practical use case on next-character prediction with :

- A bigram model
- A slightly more elaborate model (NLPM)

Going beyond N-gram with Recurrent Neural Networks (RNNs)

Neural Probabilistic Language Model

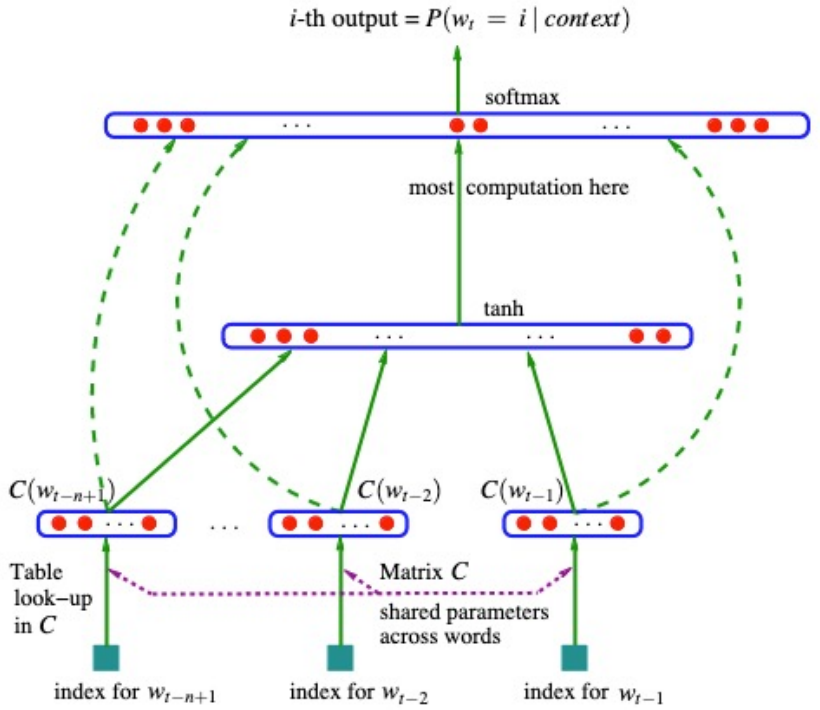


A Neural Probabilistic Language Model, Bengio et al, 2003

Going beyond N-gram with Recurrent Neural Networks (RNNs)

Neural Probabilistic Language Model

Main limitation: fixed length context



A Neural Probabilistic Language Model, Bengio et al, 2003

Going beyond N-gram with Recurrent Neural Networks (RNNs)

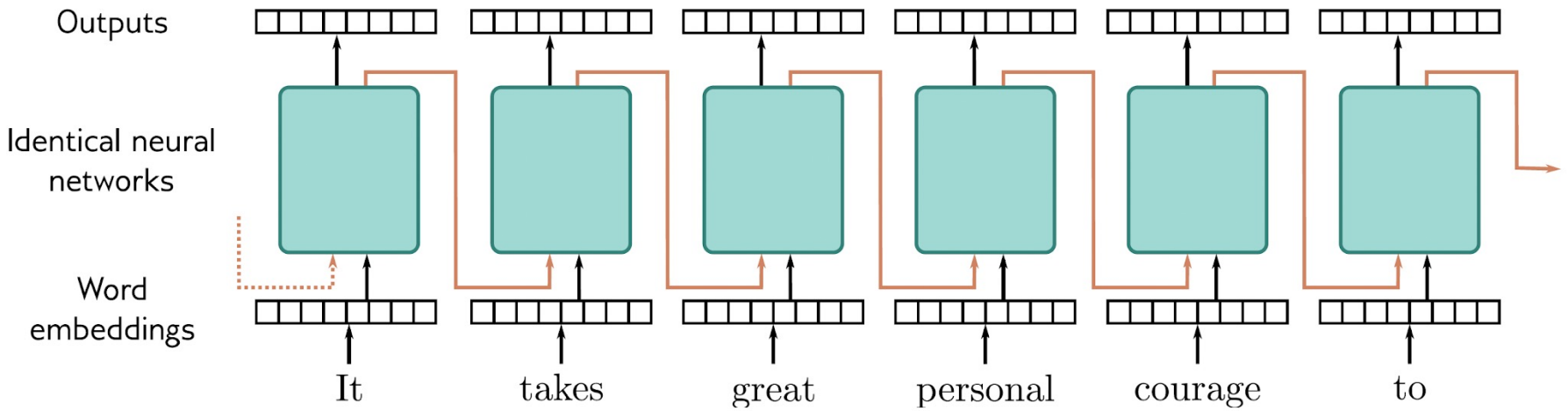


Figure 12.19 Recurrent neural networks (RNNs). The word embeddings are passed sequentially through a series of identical neural networks. Each network has two outputs; one is the output embedding, and the other (orange arrows) feeds back into the next neural network, along with the next word embedding. Each output embedding contains information about the word itself and its context in the preceding sentence fragment. In principle, the final output contains information about the entire sentence and could be used to support classification tasks similarly to the <cls> token in a transformer encoder model. However, RNNs sometimes gradually "forget" about tokens that are further back in time.

Source: Understanding Deep Learning, S. Prince, 2023

Going beyond N-gram with Recurrent Neural Networks (RNNs)

RNNs

Input layer

$$x(t) = w(t) + s(t-1)$$

Context / hidden state

$$s_j(t) = f\left(\sum_i x_i(t) u_{ji}\right)$$

Output layer

$$y_k(t) = g\left(\sum_j s_j(t) v_{kj}\right)$$

Recurrent neural network based language model, Mikolov et al, InterSpeech, 2010

Going beyond N-gram with Recurrent Neural Networks (RNNs)

Standard RNNs are unable to store information about past inputs for very long.

Learning Long-Term Dependencies with Gradient Descent is Difficult, Bengio et al, 1994.

Going beyond N-gram with Recurrent Neural Networks (RNNs)

LSTM

Long Short-term Memory is an RNN architecture designed to
be better at storing and accessing information than standard RNNs.

Generating Sequences With Recurrent Neural Networks, A. Graves, 2014

Going beyond N-gram with Recurrent Neural Networks (RNNs)

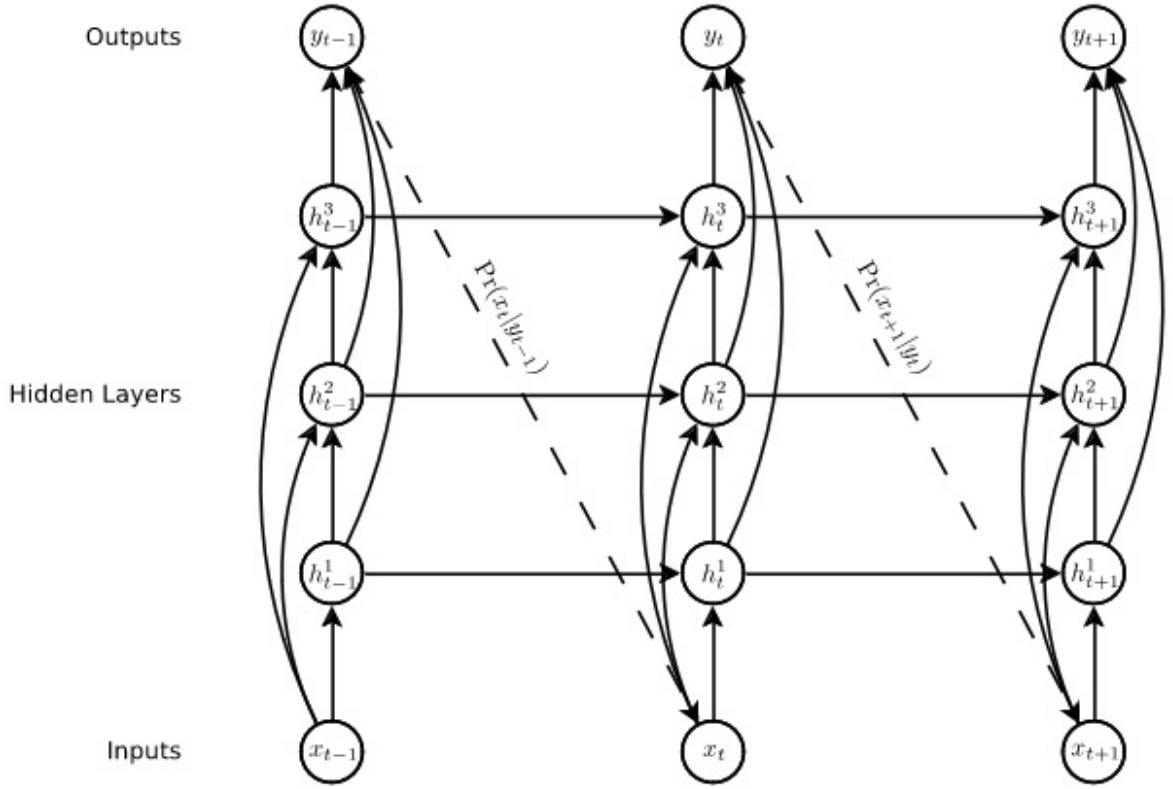


Figure 1: **Deep recurrent neural network prediction architecture.** The circles represent network layers, the solid lines represent weighted connections and the dashed lines represent predictions.

Generating Sequences With Recurrent Neural Networks, A. Graves, 2014

Going beyond N-gram with Recurrent Neural Networks (RNNs)

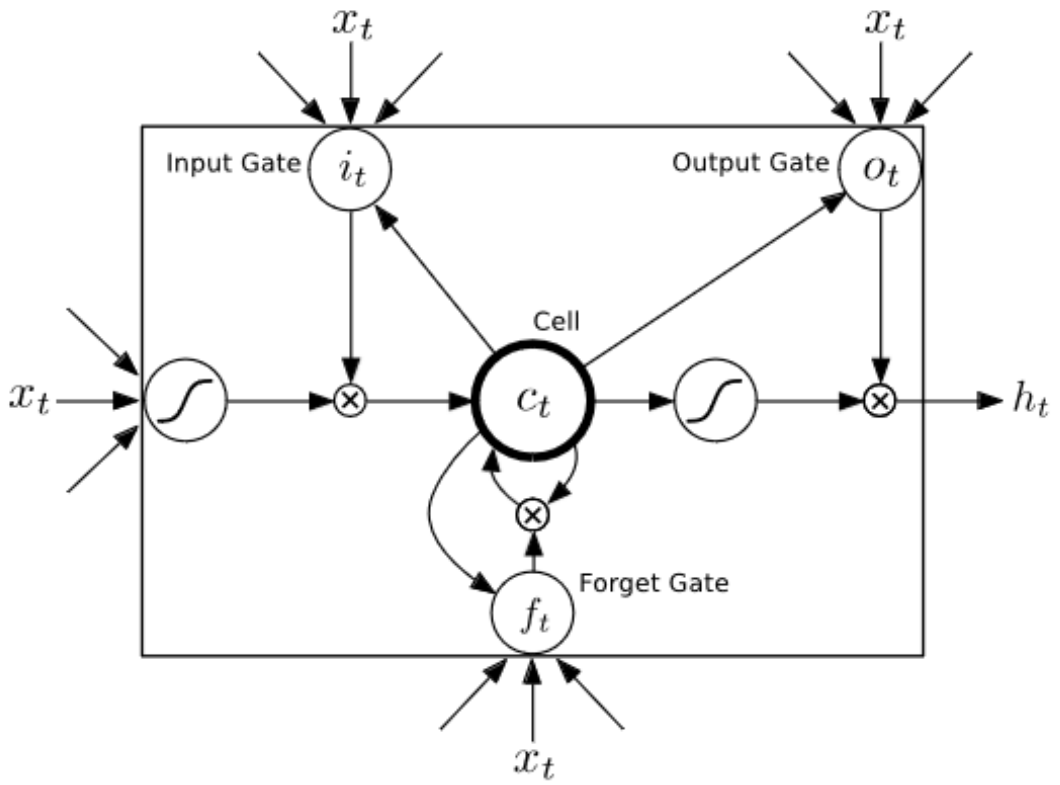


Figure 2: **Long Short-term Memory Cell**

Generating Sequences With Recurrent Neural Networks, A. Graves, 2014

Going beyond N-gram with Recurrent Neural Networks (RNNs)

Better LSTMs

Sequence to sequence learning with neural networks, Sutskever, I., Vinyals, O., and Le, Q. , NIPS 2014.

Neural Machine Translation by Jointly Learning to Align and Translate, D. Bahdanau, K. Cho, Y. Bengio, 2015

GRUs (Gated Recurrent Units)

On the Properties of Neural Machine Translation: Encoder-Decoder Approaches, K. Cho et al, 2014

Going beyond N-gram with Recurrent Neural Networks (RNNs)

GRU and LSTM seem comparable in performance.

They suffer from the same limitations:

Compressing the input sequence into a single vector.

Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling, J. Chung, C. Gulcehre, K. Cho, Y. Bengio, 2014.

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 the service.

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 the service.

Main constraints of NLP

1. NLP tasks often induce variable-length inputs.

2. Language is ambiguous: we need words/tokens to pay attention to each other.

3. Brute-force encoding the input generates large vectors: 37 words represented by 1024 embeddings -> 37,888 dimensions.

We need an efficient communication mechanism between words/tokens.

Attention and Transformers

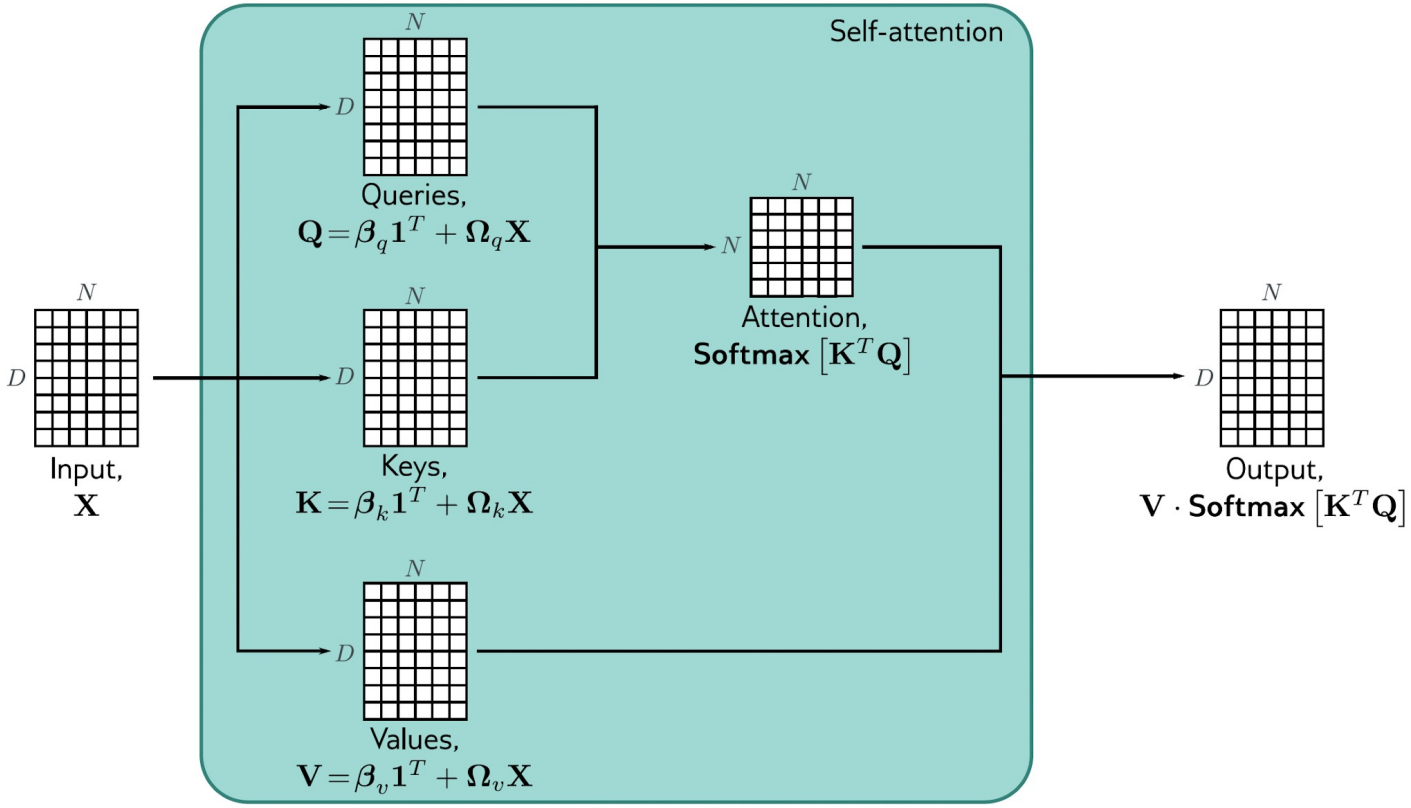
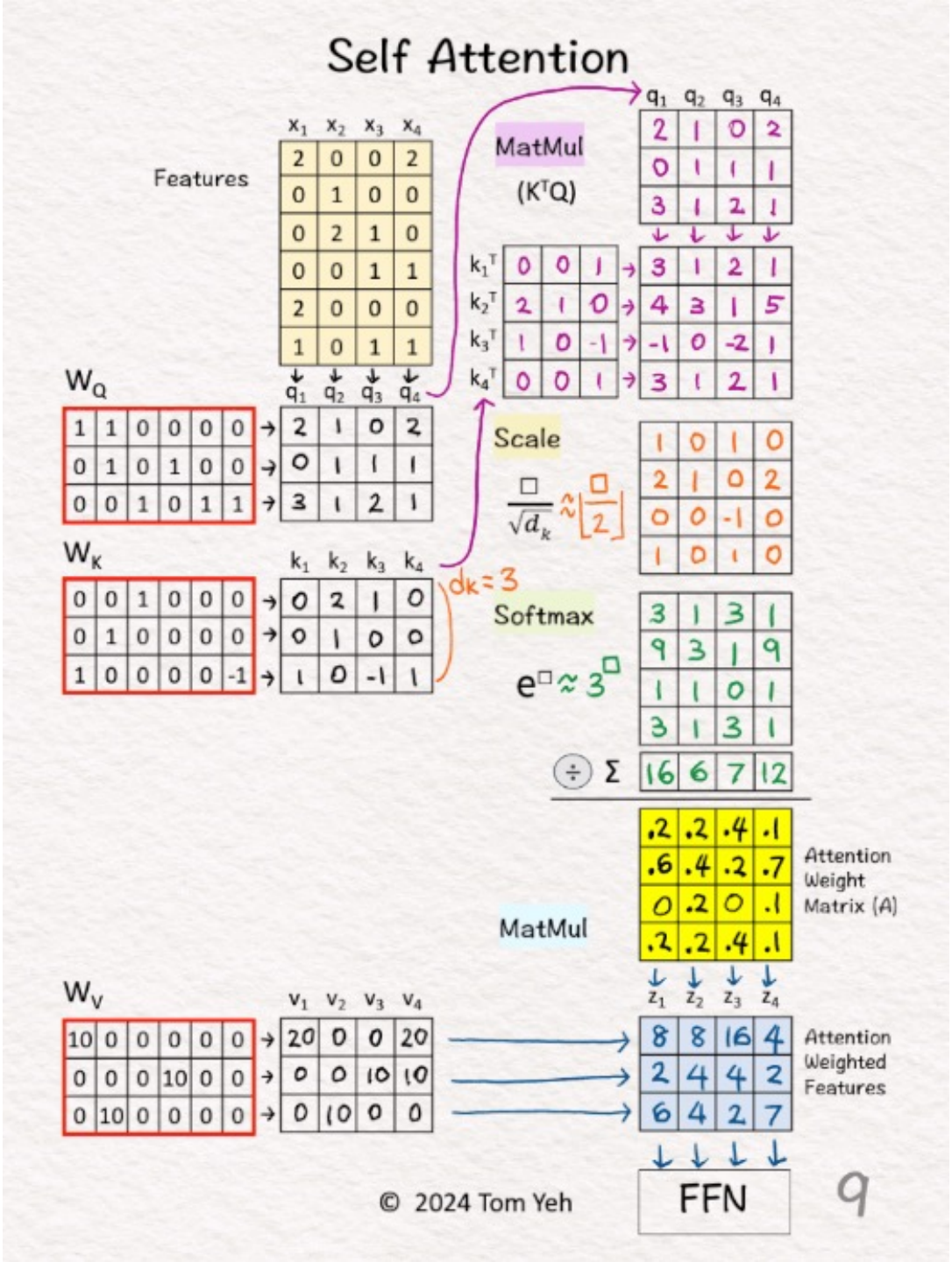


Figure 12.4 Self-attention in matrix form. Self-attention can be implemented efficiently if we store the N input vectors $\mathbf{x}_n$ in the columns of the $D\times N$ matrix $\mathbf{X}$. The input $\mathbf{X}$ is operated on separately by the query matrix $\mathbf{Q}$, key matrix $\mathbf{K}$, and value matrix $\mathbf{V}$. The dot products are then computed using matrix multiplication, and a softmax operation is applied independently to each column of the resulting matrix to calculate the attentions. Finally, the values are post-multiplied by the attentions to create an output of the same size as the input.

Source: Understanding Deep Learning, S. Prince, 2023



Source: Tom Yeh, 2024

Attention and Transformers

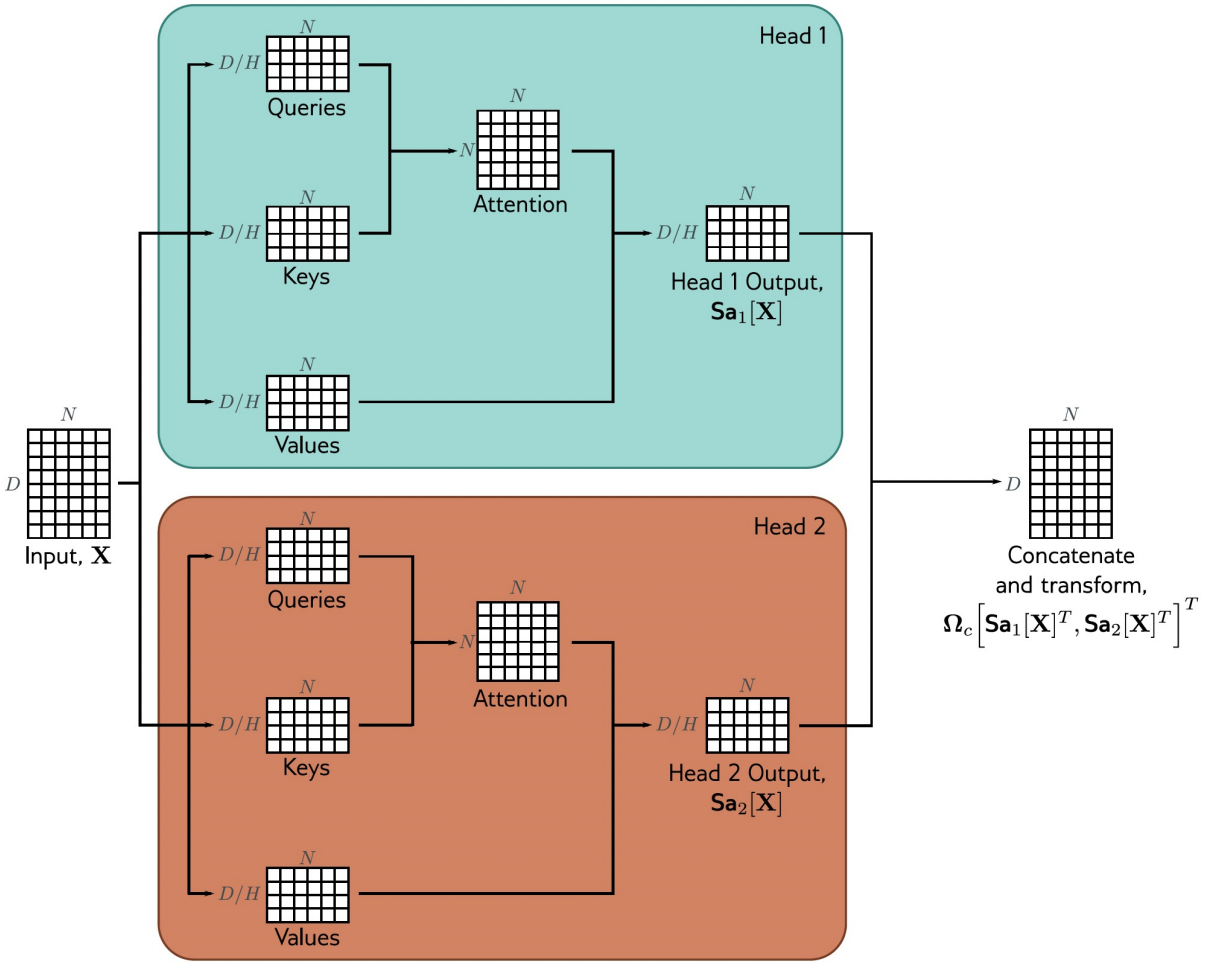


Figure 12.6 Multi-head self-attention. Self-attention occurs in parallel across multiple "heads." Each has its own queries, keys, and values. Here two heads are depicted, in the cyan and orange boxes, respectively. The outputs are vertically concatenated, and another linear transformation $\boldsymbol{\Omega}_c$ is used to recombine them.

Source: Understanding Deep Learning, S. Prince, 2023

Attention and Transformers

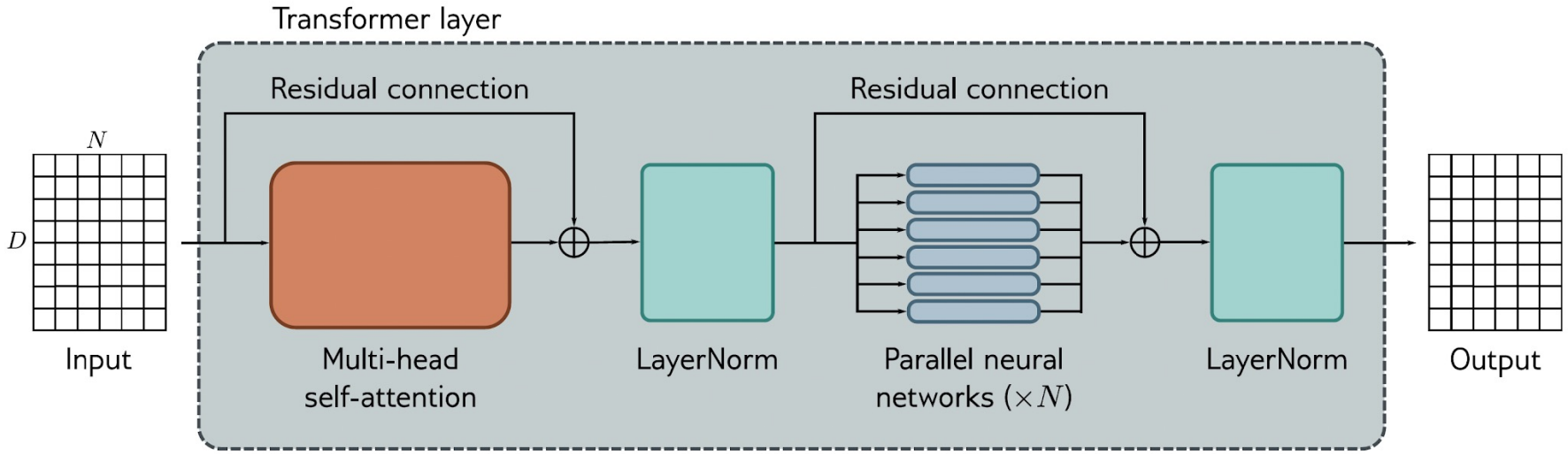


Figure 12.7 The transformer. The input consists of a $D \times N$ matrix containing the D-dimensional word embeddings for each of the N input tokens. The output is a matrix of the same size. The transformer consists of a series of operations. First, there is a multi-head attention block, allowing the word embeddings to interact with one another. This forms the processing of a residual block, so the inputs are added back to the output. Second, a LayerNorm operation is applied. Third, there is a second residual layer where the same fully connected neural network is applied separately to each of the N word representations (columns). Finally, LayerNorm is applied again.

Source: Understanding Deep Learning, S. Prince, 2023

Attention Is All You Need, A. Vaswani et al, NeurIPS 2017

Attention and Transformers

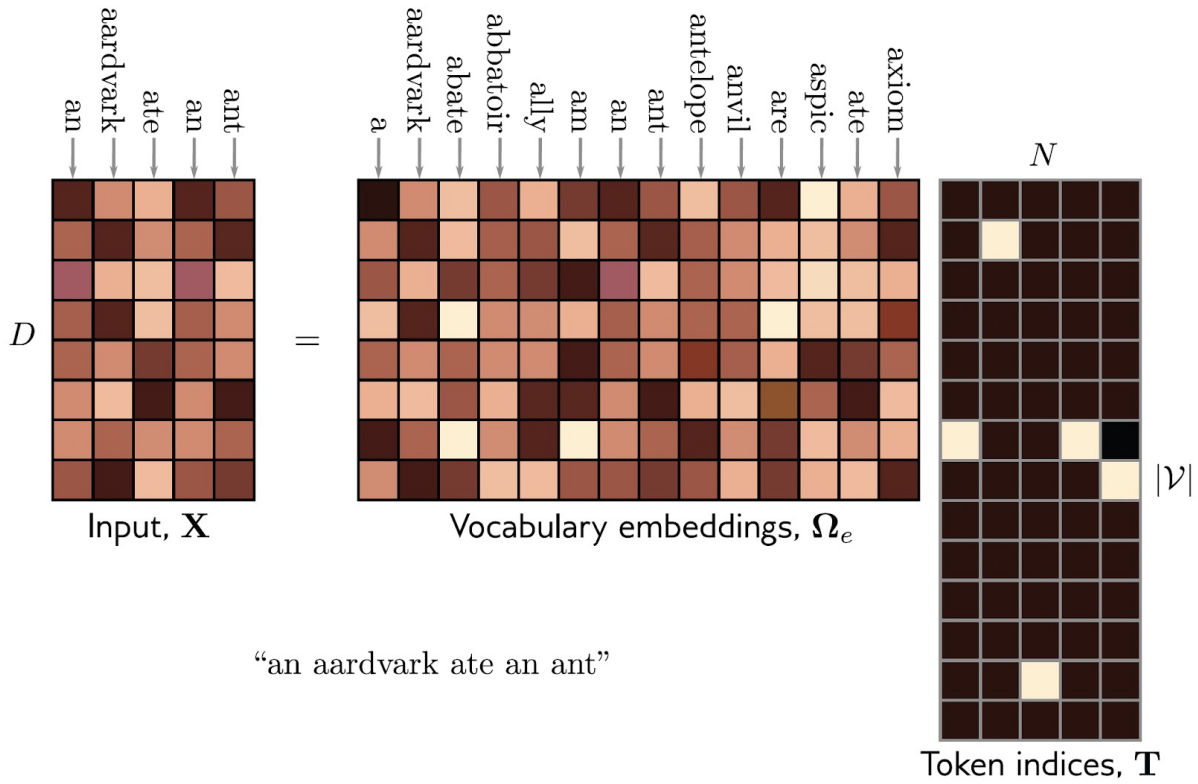


Figure 12.9 The input embedding matrix $\mathbf{X} \in \mathbb{R}^{D \times N}$ contains N embeddings of length D and is created by multiplying a matrix $\mathbf{\Omega}_e$ containing the embeddings for the entire vocabulary with a matrix containing one-hot vectors in its columns that correspond to the word or sub-word indices. The vocabulary matrix $\mathbf{\Omega}_e$ is considered a parameter of the model and is learned along with the other parameters. Note that the two embeddings for the word an in $\mathbf{X}$ are the same.

Source: Understanding Deep Learning, S. Prince, 2023

Attention and Transformers

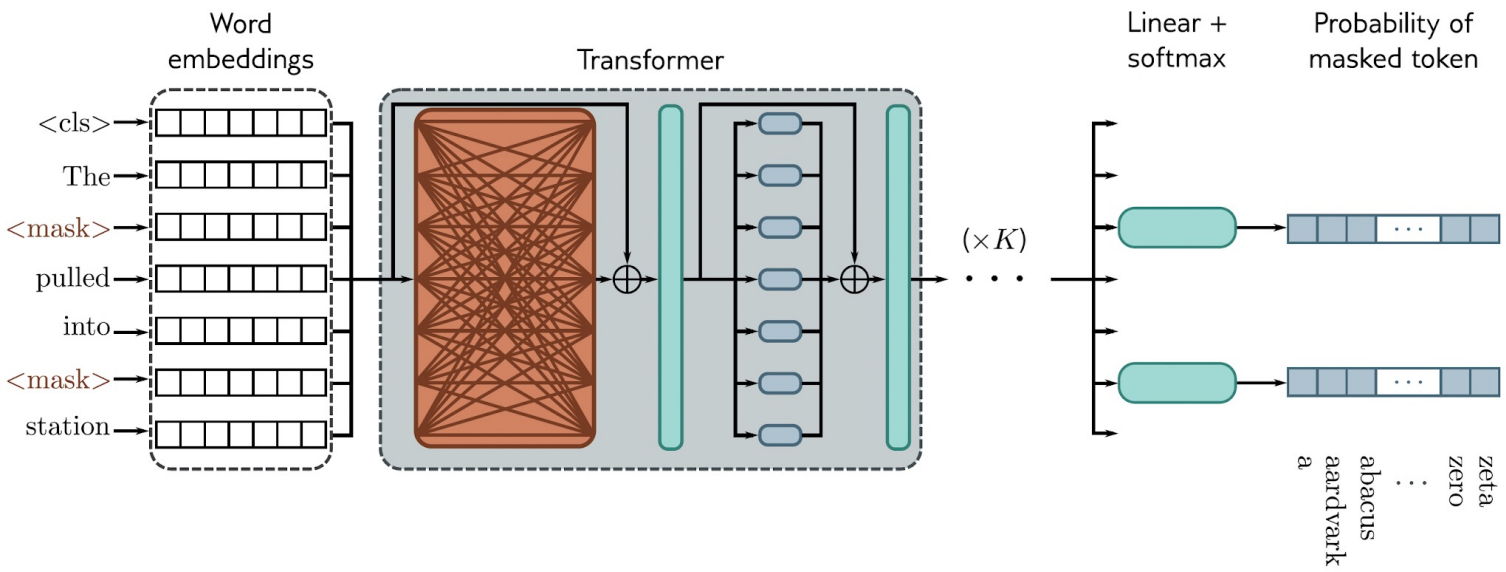


Figure 12.10 Pre-training for BERT-like encoder. The input tokens (and a special <cls> token denoting the start of the sequence) are converted to word embeddings. Here, these are represented as rows rather than columns, so the box labeled "word embeddings" is $\mathbf{X}^T$. These embeddings are passed through a series of transformers (orange connections indicate that every token attends to every other token in these layers) to create a set of output embeddings. A small fraction of the input tokens is randomly replaced with a generic <mask> token. In pre-training, the goal is to predict the missing word from the associated output embedding. As such, the output embeddings are passed through a softmax function, and the multiclass classification loss (section 5.24) is used. This task has the advantage that it uses both the left and right context to predict the missing word but has the disadvantage that it does not make efficient use of data; here, seven tokens need to be processed to add two terms to the loss function.

Source: Understanding Deep Learning, S. Prince, 2023

Attention and Transformers

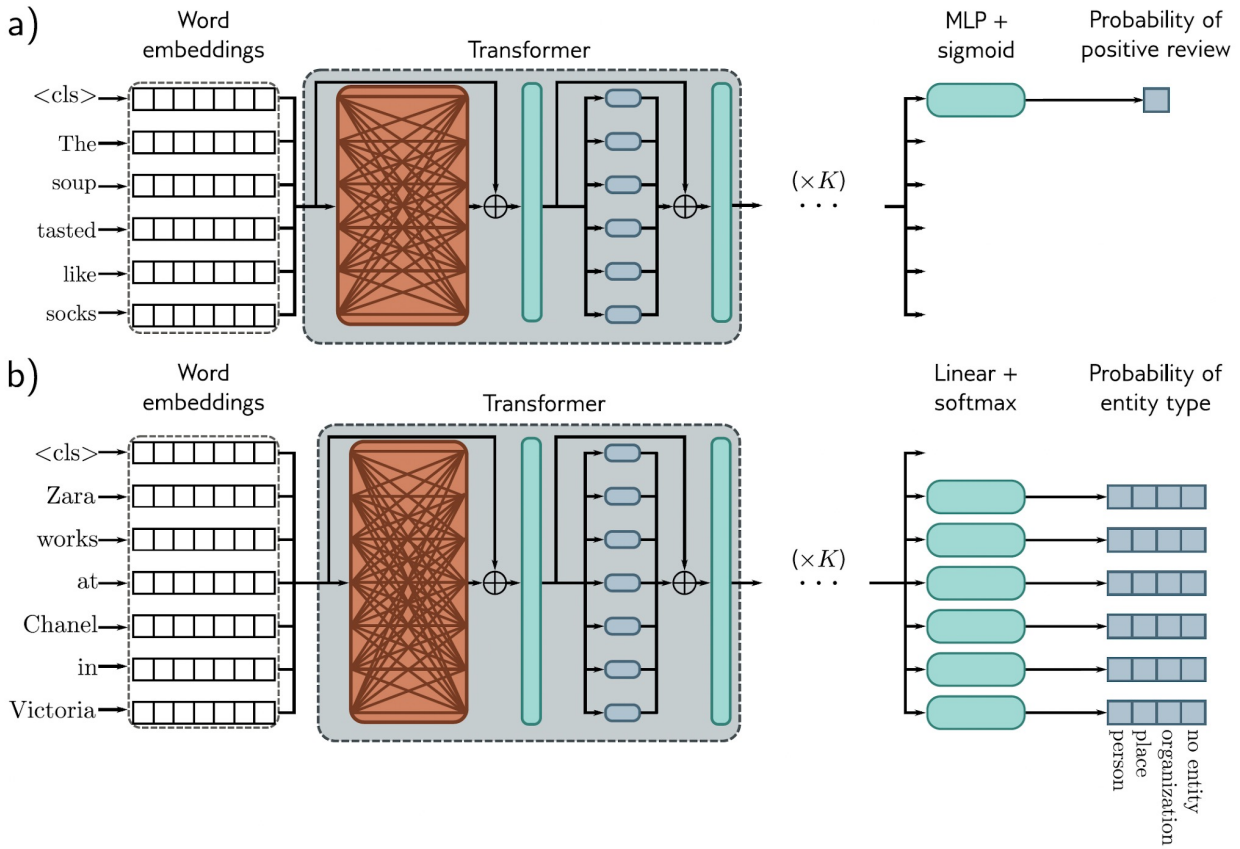


Figure 12.11 After pre-training, the encoder is fine-tuned using manually labeled data to solve a particular task. Usually, a linear transformation or a multi-layer perceptron (MLP) is appended to the encoder to produce whatever output is required. a) Example text classification task. In this sentiment classification task, the <cls> token embedding is used to predict the probability that the review is positive. b) Example word classification task. In this named entity recognition problem, the embedding for each word is used to predict whether the word corresponds to a person, place, or organization, or is not an entity.

Source: Understanding Deep Learning, S. Prince, 2023

Attention and Transformers

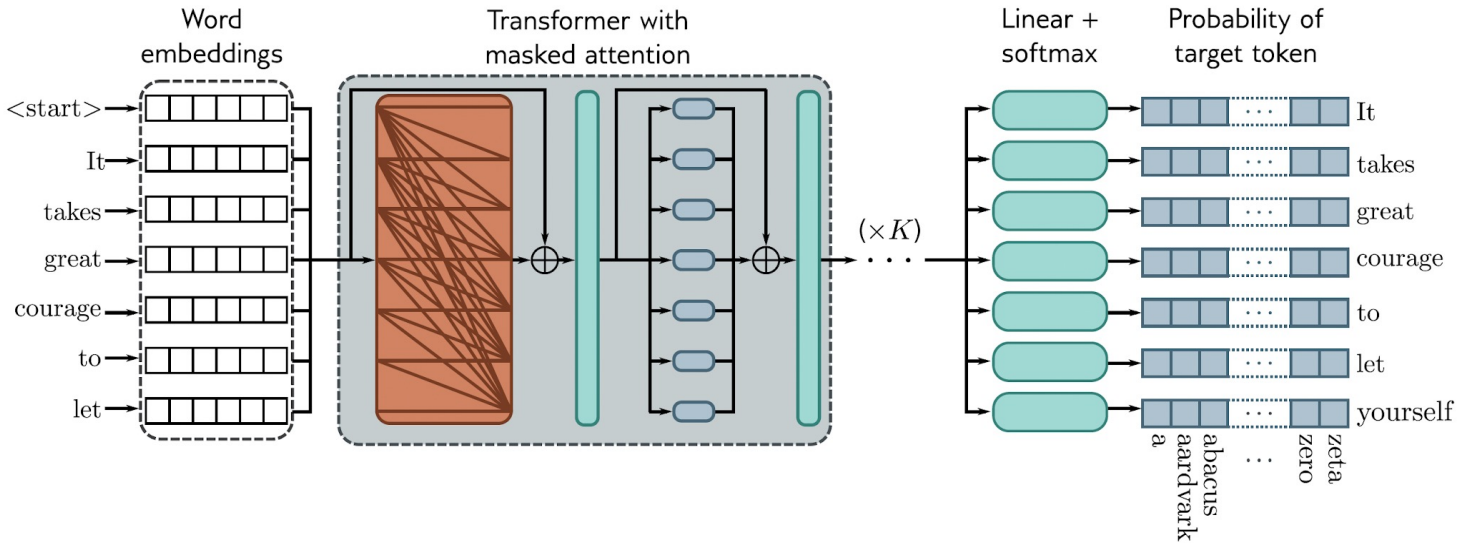


Figure 12.12 Training GPT3-type decoder network. The tokens are mapped to word embeddings with a special <start> token at the beginning of the sequence. The embeddings are passed through a series of transformers that use masked self-attention. Here, each position in the sentence can only attend to its own embedding and the embeddings of tokens earlier in the sequence (orange connections). The goal at each position is to maximize the probability of the following ground truth token in the sequence. In other words, at position one, we want to maximize the probability of the token It; at position two, we want to maximize the probability of the token takes; and so on. Masked self-attention ensures the system cannot cheat by looking at subsequent inputs. The autoregressive task has the advantage of making efficient use of the data since every word contributes a term to the loss function. However, it only exploits the left context of each word.

Source: Understanding Deep Learning, S. Prince, 2023

Attention and Transformers

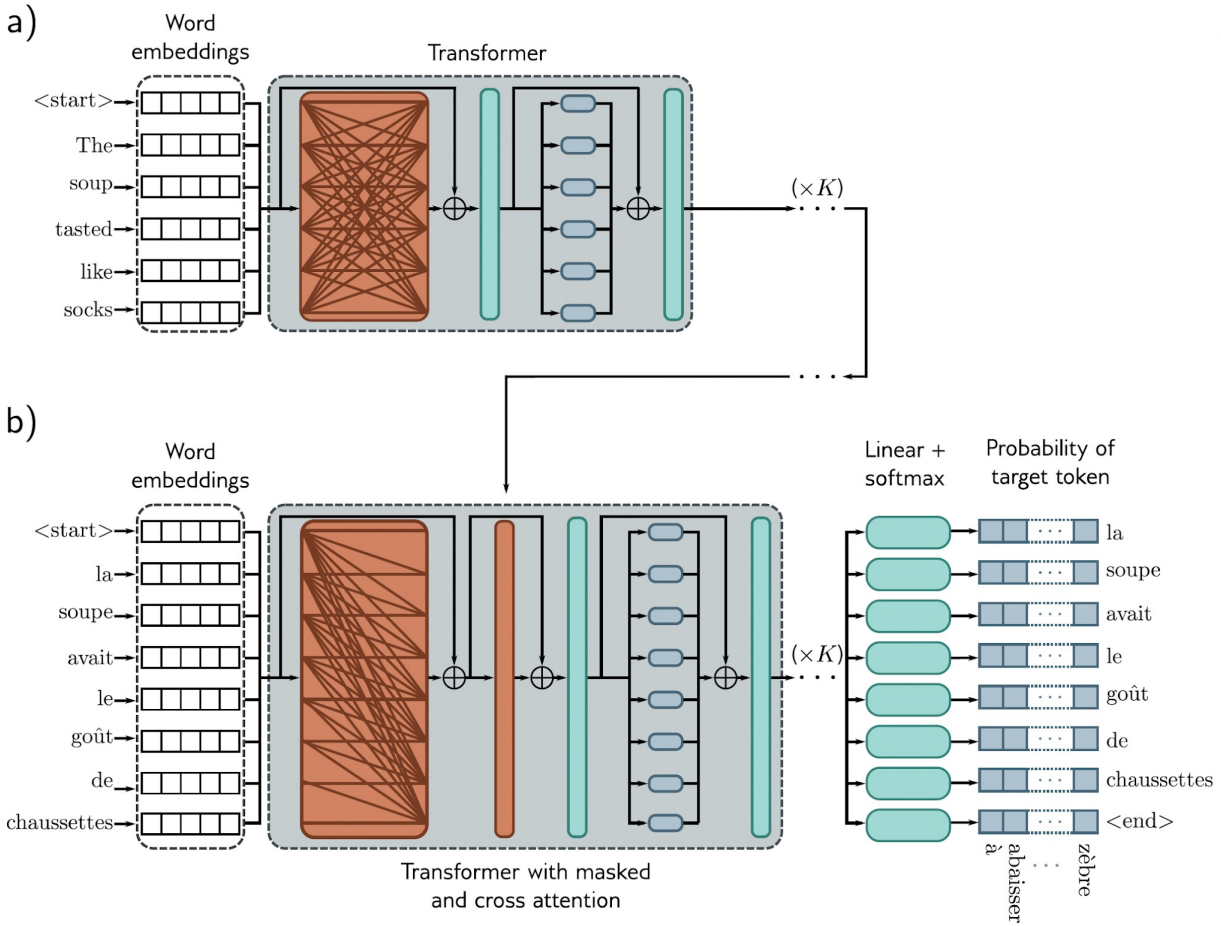


Figure 12.13 Encoder-decoder architecture. Two sentences are passed to the system with the goal of translating the first into the second. a) The first sentence is passed through a standard encoder. b) The second sentence is passed through a decoder that uses masked self-attention but also attends to the output embeddings of the encoder using cross-attention (orange rectangle). The loss function is the same as for the decoder model; we want to maximize the probability of the next word in the output sequence.

Source: Understanding Deep Learning, S. Prince, 2023

Attention and Transformers

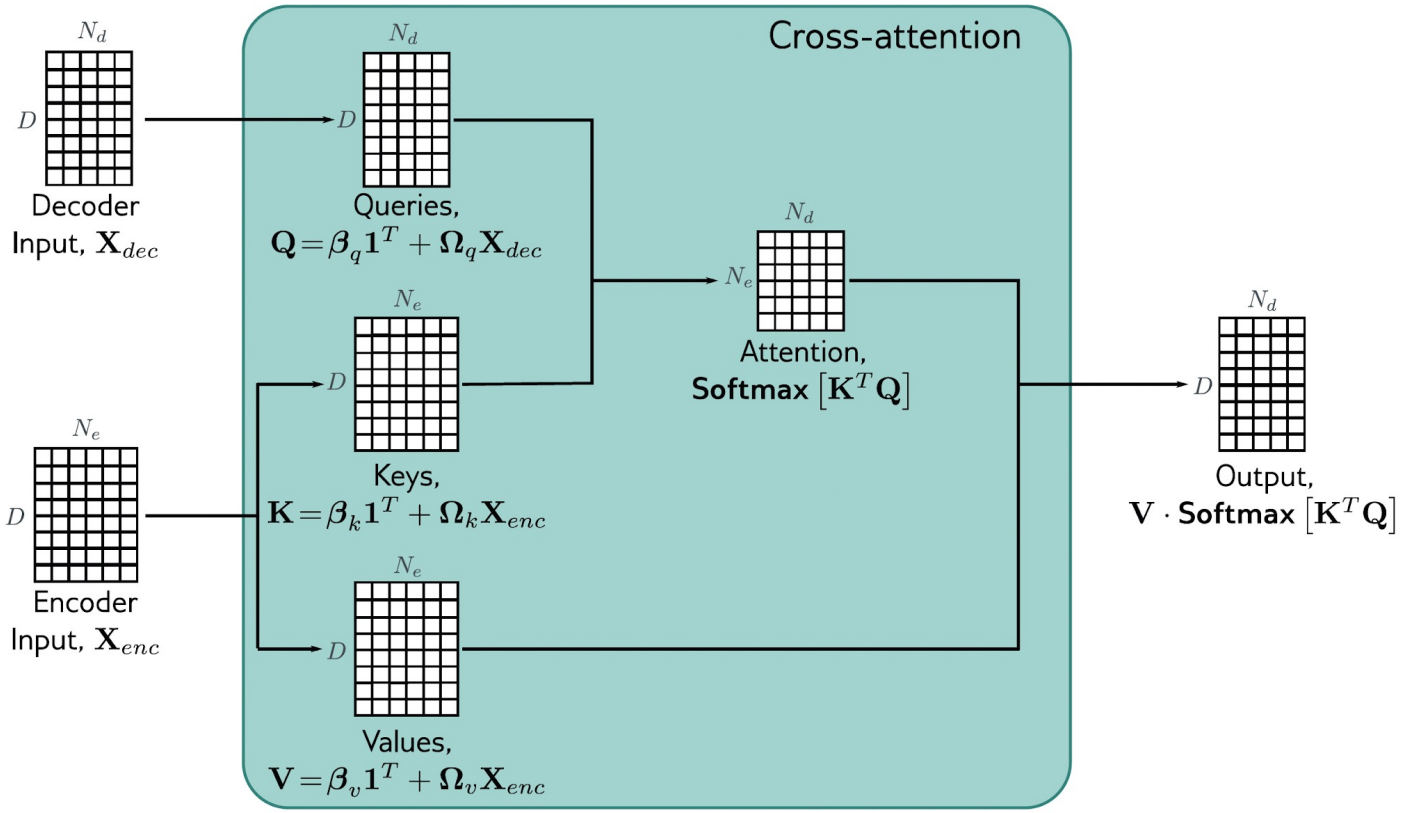


Figure 12.14 Cross-attention. The flow of computation is the same as in standard self-attention. However, the queries are now calculated from the decoder embeddings $\mathbf{X}_{dec}$, and the keys and values from the encoder embeddings $\mathbf{X}_{enc}$.

Source: Understanding Deep Learning, S. Prince, 2023