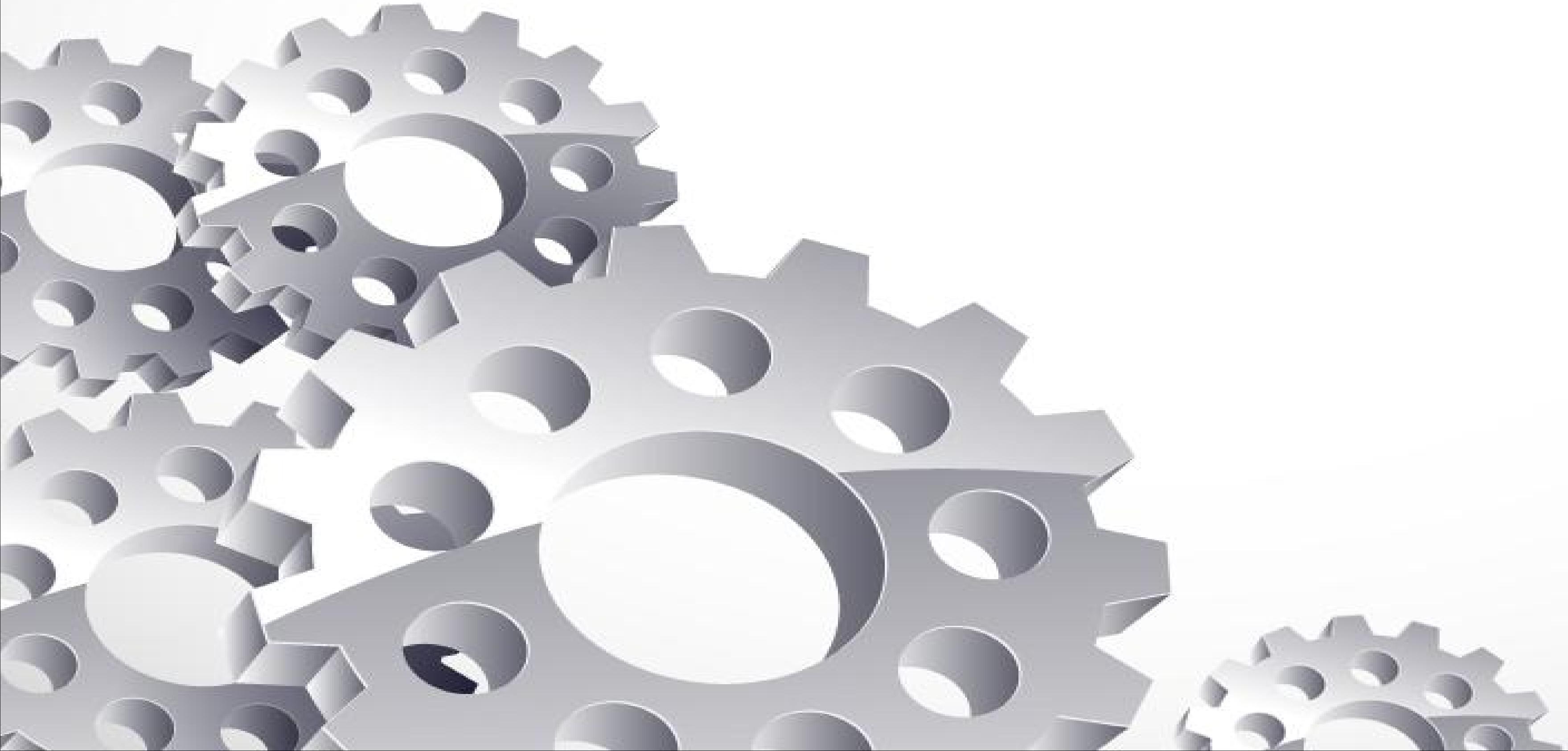
DEEP LEARNING FOR TEXT

(NLP)



NLP Applications

1. Text Classification:
Categorizing documents into predefined classes, e.g. Classifying emails as spam or not spam.

2. Content Filtering:
Identifying and blocking inappropriate content.

3. Sentiment Analysis:
Determining the emotional tone of a piece of text.

4. Translation:
Converting text from one language to another.

5. Summarization:
Generating concise summaries of longer texts.

6. Question-Answering:
Coverstaions bots, qa from given context

Evolution of NLP Models

Handcrafted Rules (Pre-1990s):

- Engineers and linguists manually crafting rule sets.

Statistical Approaches (1990s–2010s):

- Transition to statistical models, including logistic regression.

Rise of Recurrent Neural Networks (2014–2017):

- Introduction of LSTM, dominating NLP tasks.

Transformer Architecture (2017–Present):

- Transformers replacing RNNs, driving recent advancements.

Example:
Transition from RNNs to Transformers: Improving machine translation tasks from sentence-based to context-aware translation.

Text Vectorization Overview

Text Vectorization in Deep Learning:

- - Deep learning models process numeric tensors.
- - Vectorizing text transforms it into numeric tensors.

Text Vectorization Process (Figure 11.1):

- 1. **Standardization:** Convert to lowercase, remove punctuation.
- 2. **Tokenization:** Split text into units (tokens), like words or characters.
- 3. **Indexing:** Assign a numerical index to each token.
- 4. **Vector Encoding:** Convert each token to a numerical vector.
- **Example:**
- - Original Text: "Sunset came. I was staring at the Mexico sky. Isn't nature splendid?"
- - After Standardization: "sunset came i was staring at the mexico sky isnt nature splendid"
- - Tokenization: ["sunset", "came", "i", "was", "staring", "at", "the", "mexico", "sky", "isnt", "nature", "splendid"]
- - Indexing: {"sunset": 1, "came": 2, "i": 3, ..., "splendid": 12}
- - Vector Encoding: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]

Text Standardization and Tokenization

Text Standardization (11.2.1):

• - Importance of standardization for encoding consistency.

• - Example: "i" and "I" represented differently in byte strings.

Common Standardization Schemes:

• 1. Convert to lowercase and remove punctuation.

• 2. Convert special characters to standard forms.

Advanced Standardization:

• - Stemming: Convert variations of a term into a single representation.

• - Example: "was staring" and "stared" both become "[stare]".

Text Splitting (Tokenization - 11.2.2):

• - Break text into units (tokens) after standardization.

• - Three approaches: Word-level, N-gram, Character-level.

Example:

• - Original Text: "The cat sat on the mat."

• - After Standardization: "the cat sat on the mat"

• - Word-level Tokenization: ["the", "cat", "sat", "on", "the", "mat"]

• - Token Indices: [1, 2, 3, 4, 1, 5]

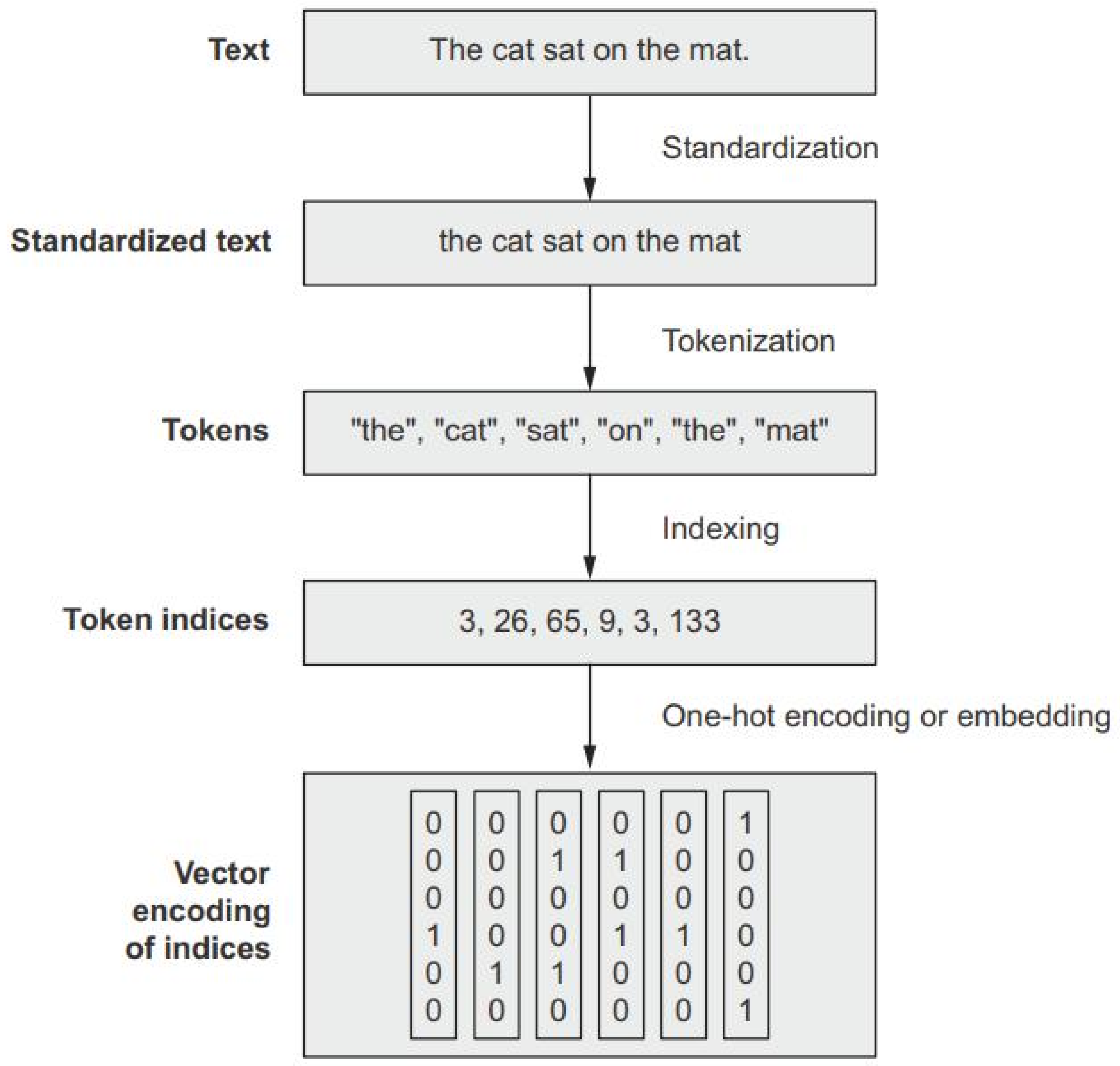


Figure 11.1 From raw text to vectors

Understanding N-grams and Bag-of-Words
(A traditional ML approach)

Introduction to N-grams and Bag-of-Words:

- - N-grams are groups of N consecutive words or characters extracted from a sentence.
- - The concept is applied for both words and characters.

Example: Sentence "the cat sat on the mat":

- - **2-grams (Bigrams):**
- - {"the", "the cat", "cat", "cat sat", "sat", "sat on", "on", "on the", "the mat", "mat"}
- - **3-grams (Trigrams):**
- - {"the", "the cat", "cat", "cat sat", "the cat sat", "sat", "sat on", "on", "cat sat on", "on the", "sat on the", "the mat", "mat", "on the mat"}

****Bag-of-N-grams:** [The set is also called bag of words]**

- - A set of tokens without a specific order.
- - Extracting N-grams is a manual approach for creating features (feature engineering).
- - Not Order-Preserving and No **context-based** representation.
- - Tokens are treated as a set, not a sequence, losing the sentence's contextual structure.
- **-** Bag-of-Words is foundational, commonly applied in traditional ML.

Sequence of Word approach (DL way)

The text is tokenizad as a sequence, and a whole word is grouped into a list of tokens, thus preserving sequence.

Vocabulary Indexing: (text to numeric conversion):

The tokens are converted to list of numerical values, using vocabulary indexing where tokens are mapped to unique integers.

One-hot-encoded Vocabulary:

will have huge sparse matrix, and very slow without any real advantage, example: page 328/329)

Vocabulary Size Restriction:

- Common to limit the vocabulary to the top 20,000 or 30,000 most common words.
- Reduces feature space, eliminating rare terms with minimal information.

Tokenization and Vectorization

Handling Out-of-Vocabulary (OOV):

- Unseen words during training may cause errors when indexed.
- It is customary to use an OOV token (e.g., "[UNK]") when decoding integer sequences.

Special Tokens:

- **OOV Token (Index 1):** Represents unrecognized words.
- **Mask Token (Index 0):** Used for padding sequences.

Concept of Padding [Pad]:

As the neural networ required a fixed length tensors, in order to make all sentences equal, we will use padding.

`[5, 7, 124, 4, 89]` and `[8, 34, 21]` become `[[5, 7, 124, 4, 89], [8, 34, 21, 0, 0]]`

Keras TextVectorization layer

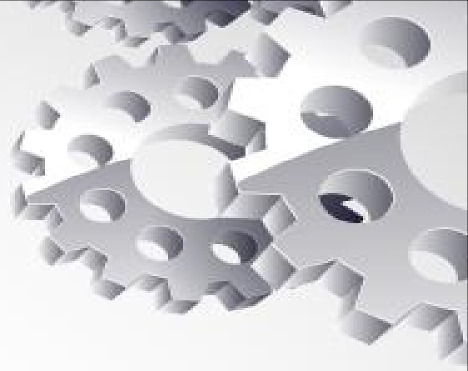
```
from tensorflow.keras.layers import TextVectorization
text_vectorization = TextVectorization(output_mode="int", )
```

Note:
By default, the TextVectorization layer will use the setting “convert to lowercase and remove punctuation” for text standardization, and “split on whitespace” for tokenization.
(important: you can provide custom functions for standardization and tokenization)

Usage:

```
dataset = [
 "I write, erase, rewrite",
 "Erase again, and then",
 "A poppy blooms.",]
text_vectorization.adapt(dataset)
>>> text_vectorization.get_vocabulary()
     ["", "[UNK]", "erase", "write", ...]
```

One Fundamental Question



How to encode the way words are woven into sentences ?

-we discussed bag of words (traditional models approach)

-we discussed sequence of words (we can implement LSTM models for this)

Challenge:

- Unlike the steps of a timeseries, words in a sentence don't have a natural, canonical order. (e.g. the sentence structure of English is quite different from that of Japanese)

- Even within a given language, you can typically say the same thing in different ways by reshuffling the words a bit.
- Even further, if you fully randomize the words in a short sentence, you may still largely figure out what it was saying.

Order is clearly important, but its relationship to meaning isn't straightforward.

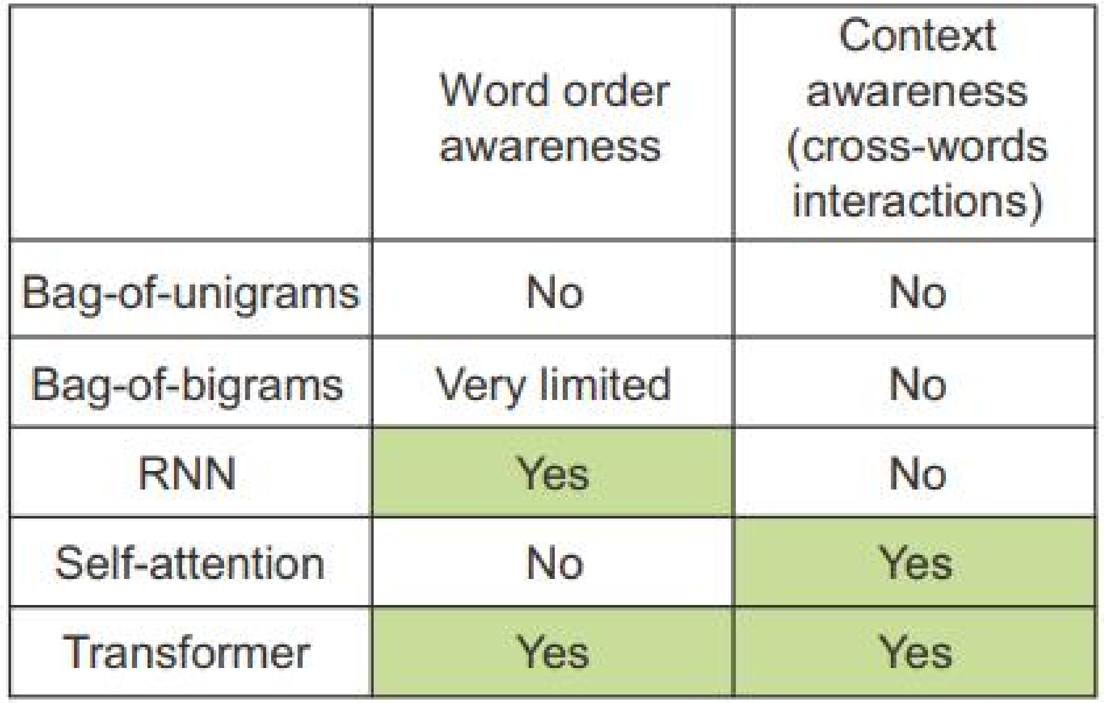
Possibilities

1. Discard word order, treat text as an unordered set.
   (Bag-of-Words Models)
2. Process words in order, like steps in a time series.
   (use Sequential Models (RNNs/LSTM)
3. Hybrid Approach: Use best of both worlds
   (Transformers)

Transformers architecture is technically order-agnostic, yet it injects word-position information into the representations it processes, which enables it to simultaneously look at different parts of a sentence (unlike RNNs) while still being order-aware.

Because they take into account word order, and operate on whole sequence both RNNs and Transformers are called sequence models

Comparioson of various approaches



	Word order awareness	Context awareness (cross-words interactions)
Bag-of-unigrams	No	No
Bag-of-bigrams	Very limited	No
RNN	Yes	No
Self-attention	No	Yes
Transformer	Yes	Yes

Figure 11.10 Features of different types of NLP models

Implementation of Bag-of-Word models

- Self study. Page 322
- Section 11.3.2 Processing words as a set: The bag-of-words approach
- The topic is just included to make comparisons with new approaches.

Implementation of Sequence Models

The process of implementing a sequence model involves representing input samples as sequences of integer indices.

Each integer is mapped to a vector to create vector sequences.

- These sequences of vectors are fed into a stack of layers. The layers are designed to capture and analyze patterns in the sequential data. Different types of layers can be used for this purpose.
- The layers are capable of cross-correlating features from adjacent vectors. This means the model can learn relationships and dependencies between words that are close to each other in the sequence.

THE BEGINING OF MODERN NLP

Introduction to Word Embeddings

- Word embeddings are vector representations of words.
- Recognition that words share information and have a structured relationship.
- Geometric relationship between word vectors should reflect semantic relationships.
- Example: "Movie" and "film" should not be orthogonal but close in the vector space.
- Word embeddings capture semantic relationships, addressing one-hot encoding limitations.
- Visualization of word embeddings reveals meaningful geometric relationships.

Word Embeddings

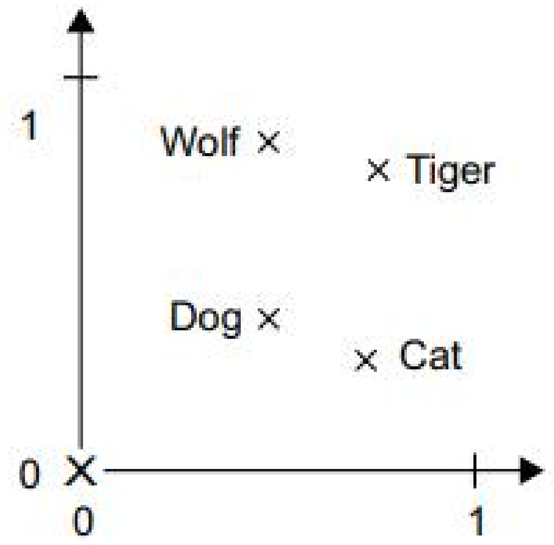


Figure 11.3 **A toy example of a word-embedding space**

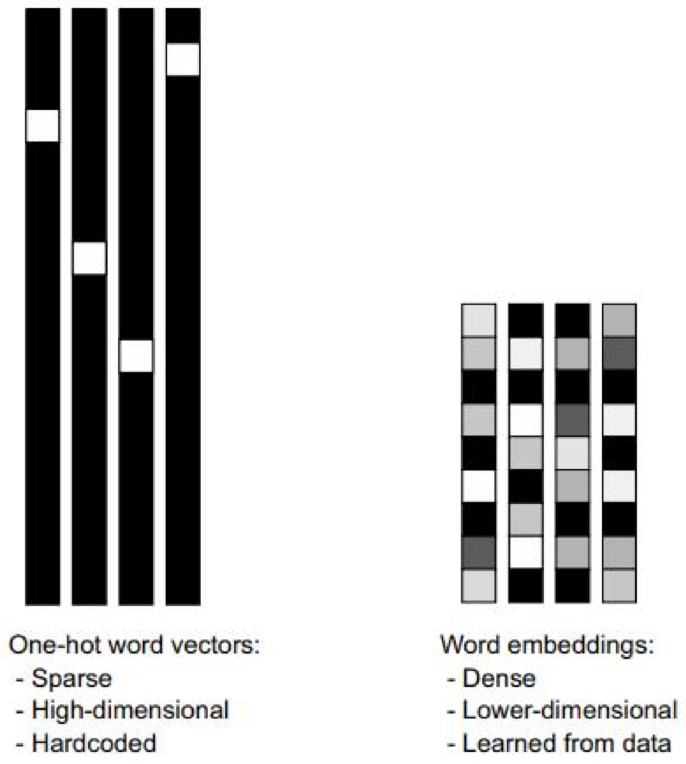
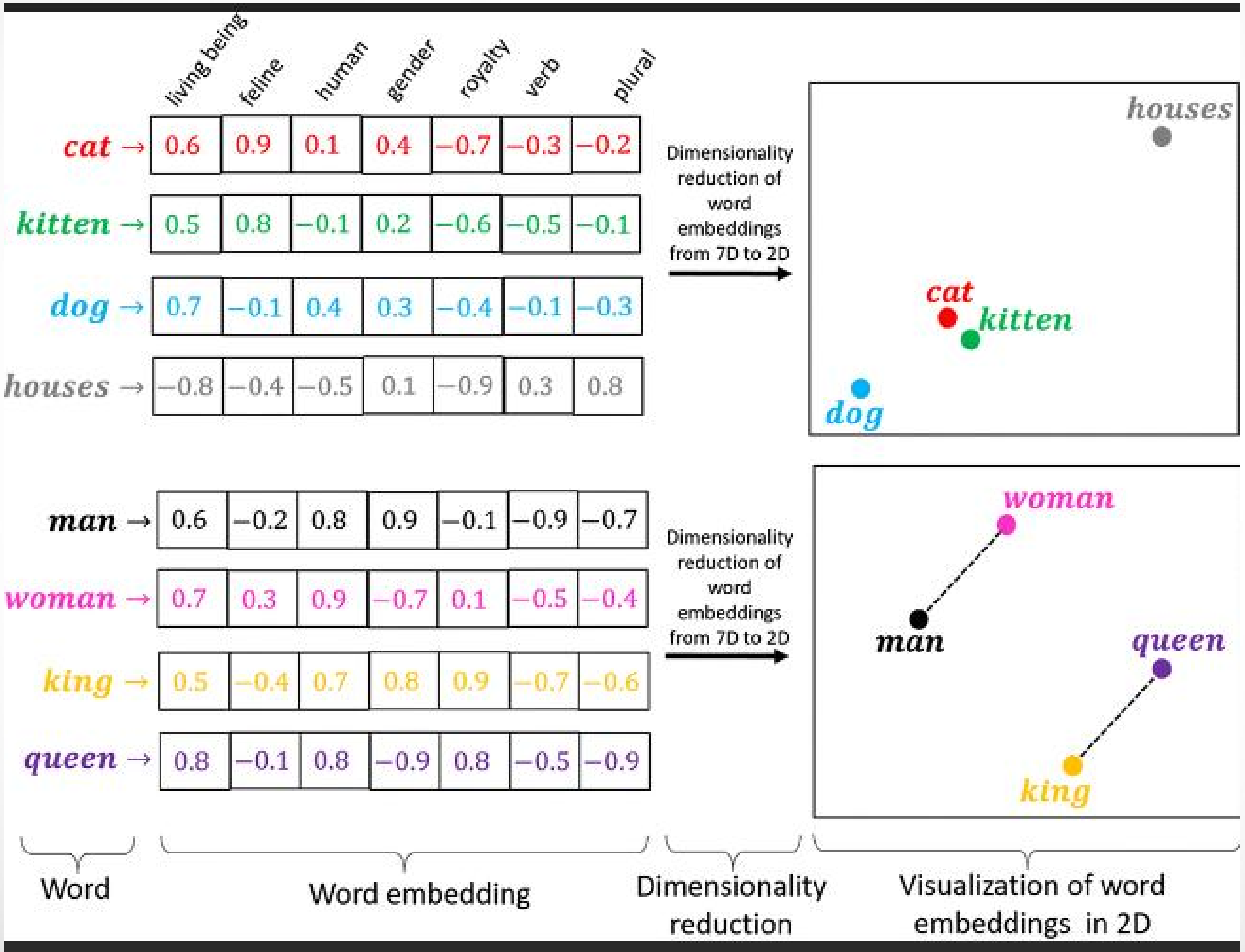


Figure 11.2 **Word representations obtained from one-hot encoding or hashing are sparse, high-dimensional, and hardcoded. Word embeddings are dense, relatively low-dimensional, and learned from data.**

Word Embeddings vs One-Hot Encodings

Embeddings example in multiple dimensions



	living being	feline	human	gender	royalty	verb	plural
cat →	0.6	0.9	0.1	0.4	−0.7	−0.3	−0.2
kitten →	0.5	0.8	−0.1	0.2	−0.6	−0.5	−0.1
dog →	0.7	−0.1	0.4	0.3	−0.4	−0.1	−0.3
houses →	−0.8	−0.4	−0.5	0.1	−0.9	0.3	0.8

man →	0.6	−0.2	0.8	0.9	−0.1	−0.9	−0.7
woman →	0.7	0.3	0.9	−0.7	0.1	−0.5	−0.4
king →	0.5	−0.4	0.7	0.8	0.9	−0.7	−0.6
queen →	0.8	−0.1	0.8	−0.9	0.8	−0.5	−0.9

Dimensionality reduction of word embeddings from 7D to 2D
houses
cat
kitten
dog
Dimensionality reduction of word embeddings from 7D to 2D
woman
man
queen
king
Word
Word embedding
Dimensionality reduction
Visualization of word embeddings in 2D

How to use such an embedding space in practice.

There are two ways to obtain word embeddings:

1. Learn word embeddings jointly with the main task you care about (such as document classification or sentiment prediction).
   In this setup, you start with random word vectors and then learn word vectors in the same way you learn the weights of a neural network.

2. Load into your model word embeddings that were precomputed using a different machine learning task than the one you’re trying to solve. These are called pretrained word embeddings.

1. Using own Embedding Layers

```
embedding_layer = layers.Embedding(input_dim=max_tokens, output_dim=256)
```

The Embedding layer is best understood as a dictionary that maps integer indices (which stand for specific words) to dense vectors. It takes integers as input, looks up these integers in an internal dictionary, and returns the associated vectors. It’s effectively a dictionary lookup.

Page 332, Listing 11.16, Page 334, Listing 11.17

Have example of Embeddings Training from scratch.

Its out of scope of our study. You can go through in your own time.

Practically speaking any viable training of word embedding model would require lot of training data and compute resource to have realistic representation power.

2. Using Pre-trained Embedding Layers

In most cases you would be using pretrained embeddings models (atleast for 1+ year of practising of NLP)

The book (this is published in 2021) speaks of Precomputed embedding databases named Word2Vec, GloVe, downloadable from keras website, but I see they have little relevance these days. Therefore will not waste much time on these (see page 334/335)

Let's concentrate on Transformers, and we will see newer embedding models in lecture 14,15 in more details.

TRANSFORMERS

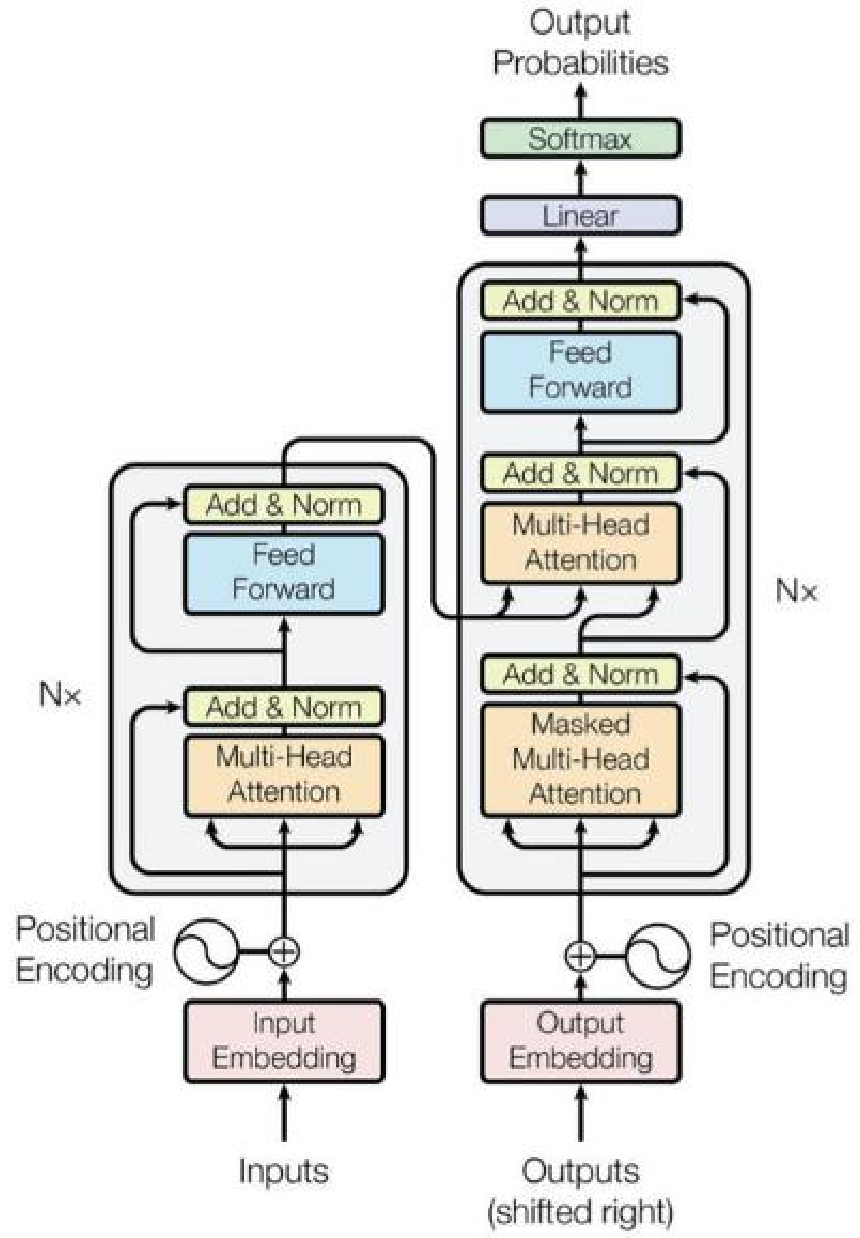


Figure 1: The Transformer - model architecture.

Attention is All You Need (2017)

****Introduction:****

• - Emergence in 2017 as a dominant architecture in NLP tasks.

• - Introduced in "Attention is all you need" by Vaswani et al.

• - Central idea: "neural attention" for powerful sequence models.

• - Mechanism without needing recurrent or convolutional layers.

****Impact:****

• - Revolutionary shift in NLP and beyond.

• - Neural attention becomes a leading concept in deep learning.

• - Fundamental shift in sequence modeling.

- **Self-Attention:**

• - Core concept in the Transformer.

• - Effective for processing sequential data.

- **Application:**

• - Leveraging self-attention for a Transformer encoder

Self Attention - Central to Transformers

Concept:

- Models should focus differently on input features, assigning varying importance. (Analogous to human reading habits - skimming and focusing on different parts)

Comparison to Previous Concepts:

Max Pooling (Convnets): Selects one most important feature in a spatial region.

TF-IDF Normalization: Assigns importance scores to tokens based on information relevance.

Importance Scores:

- Computed for a set of features.
- Higher scores for more relevant features; lower scores for less relevant ones.

Purpose of Attention Mechanism:

- Goes beyond highlighting or erasing features.
- Makes features context-aware
- Context-specific representations needed due to the context-specific nature of language.

Self-Attention in Word Embeddings:

- Embedding spaces capture semantic relationships but lack context specificity.
- Introduction of self-attention to make token representations context-aware.

General Concept of Attention in DL

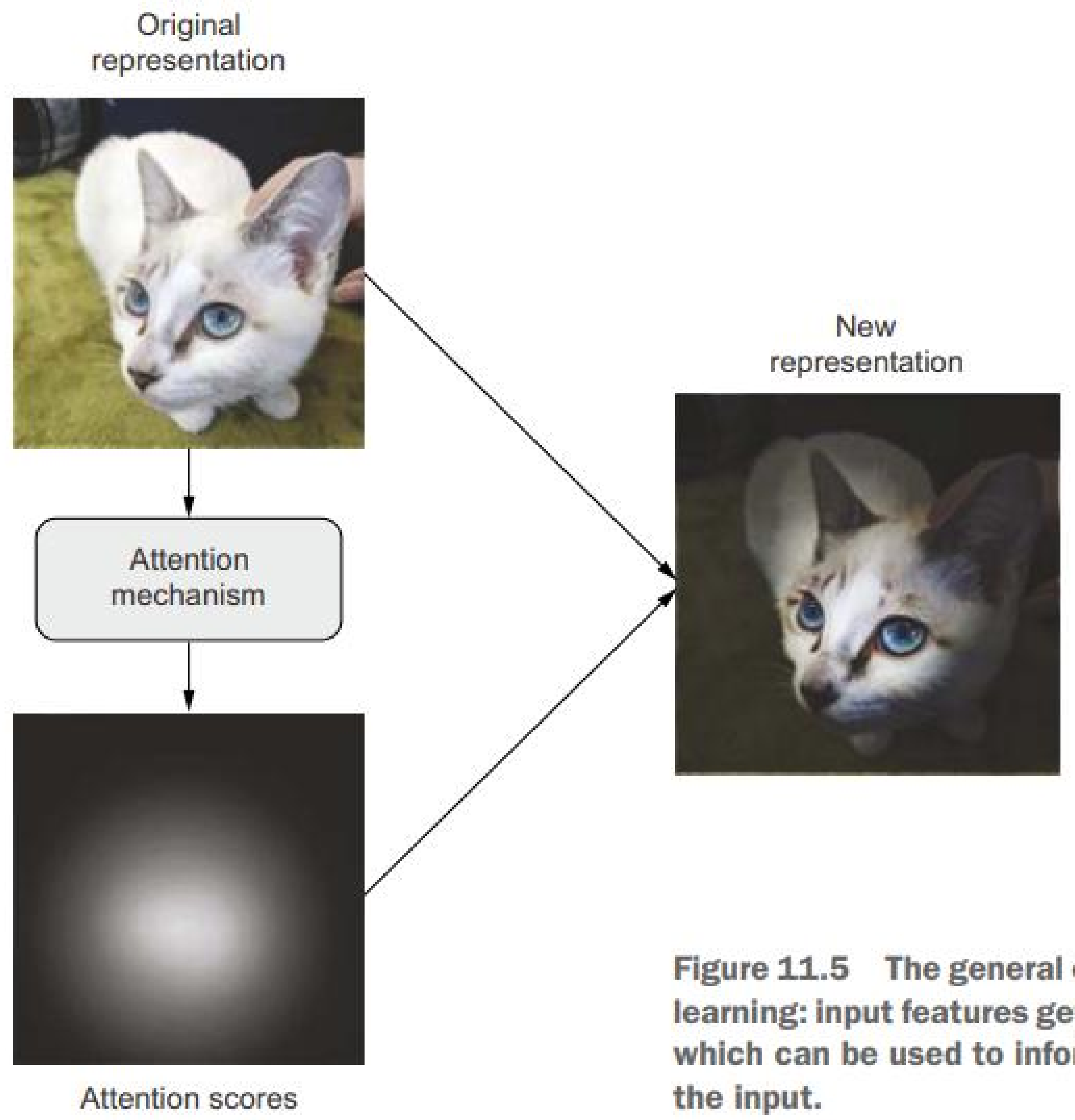


Figure 11.5 The general concept of "attention" in deep learning: input features get assigned "attention scores," which can be used to inform the next representation of the input.

Self-Attention - an Algorithmic Illustration

** Context is what matters**

- When you say, “I’ll see you soon,” the meaning of the word “see” is subtly different from the “see” in “I’ll see this project to its end” or “I see what you mean.”
- And, of course, the meaning of pronouns like “he,” “it,” “in,” etc.,

** Smart Embedding Space**

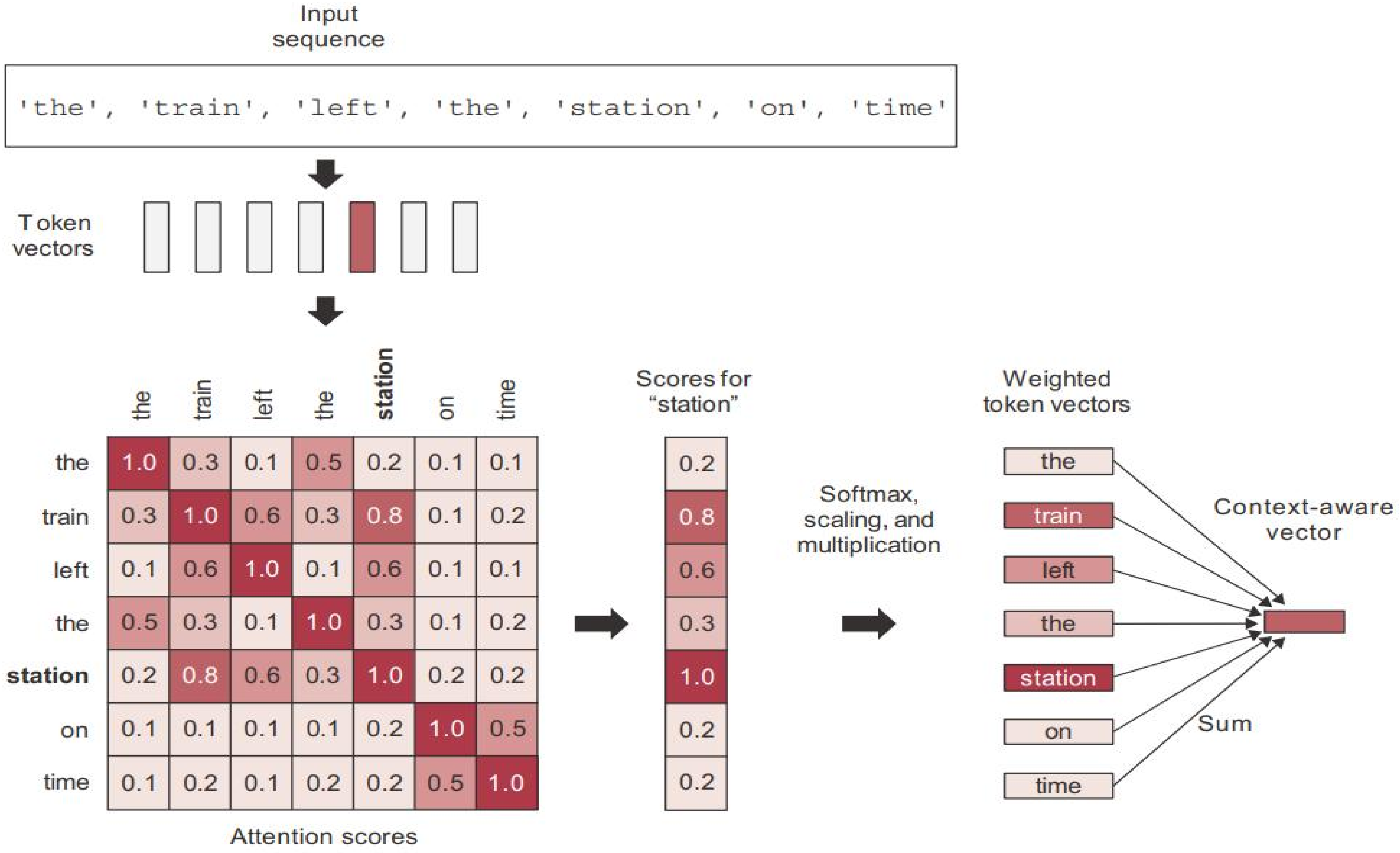
- A smart embedding space would provide a different vector representation for a word depending on the other words surrounding it. That’s where self-attention comes in.
- The purpose of self-attention is to modulate the representation of a token by using the representations of related tokens in the sequence.

Example with the word "station" in the sentence.

- Computation of relevancy scores between "station" and other words in the sentence.
- Summing word vectors weighted by relevancy scores to obtain a new representation for "station."

Implementation Details:

- Utilizes dot product as a measure of the relationship strength between word vectors.
- Scaling function and softmax used in practice for relevancy scores.

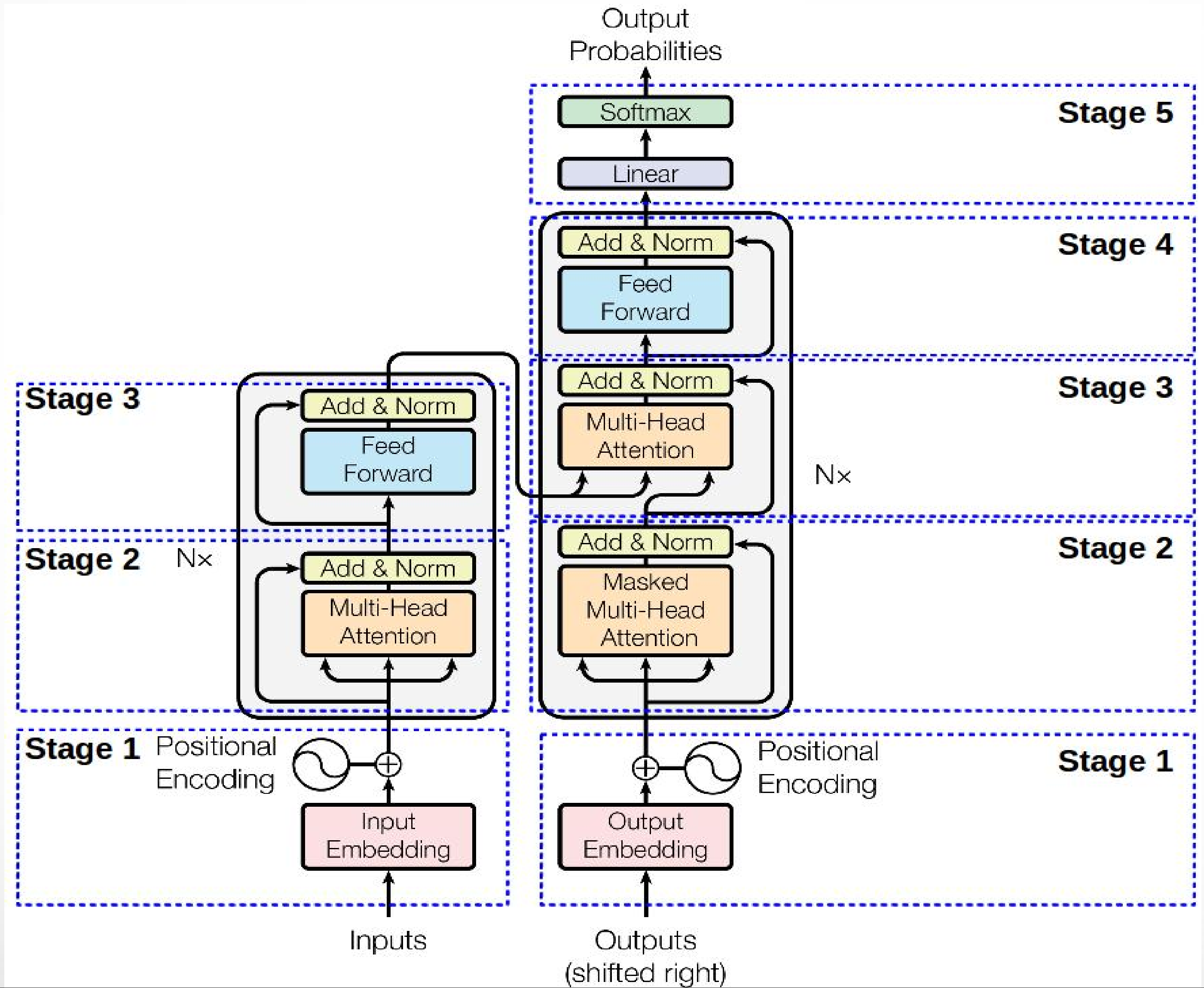


	the	train	left	the	**station**	on	time
the	1.0	0.3	0.1	0.5	0.2	0.1	0.1
train	0.3	1.0	0.6	0.3	0.8	0.1	0.2
left	0.1	0.6	1.0	0.1	0.6	0.1	0.1
the	0.5	0.3	0.1	1.0	0.3	0.1	0.2
station	0.2	0.8	0.6	0.3	1.0	0.2	0.2
on	0.1	0.1	0.1	0.1	0.2	1.0	0.5
time	0.1	0.2	0.1	0.2	0.2	0.5	1.0

Figure 11.6 Self-attention: attention scores are computed between "station" and every other word in the sequence, and they are then used to weight a sum of word vectors that becomes the new "station" vector.

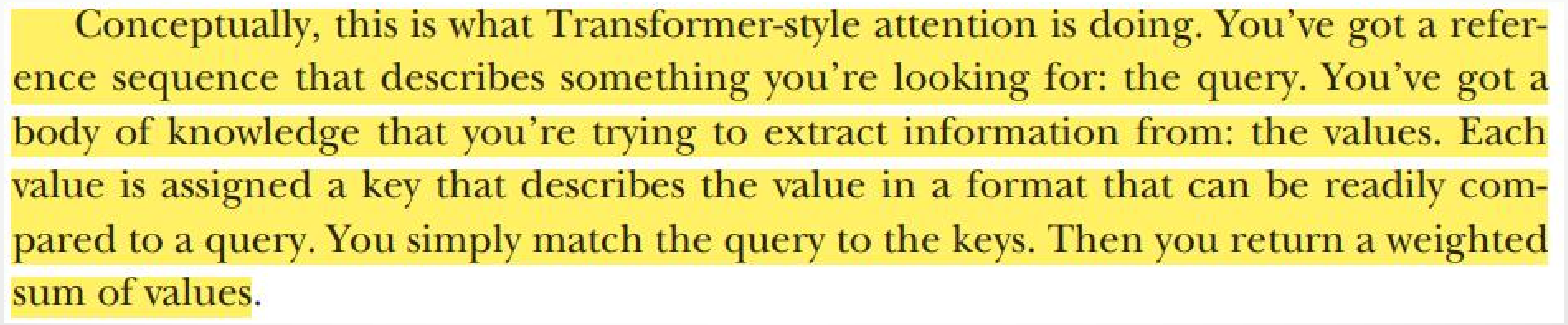
Best Resource to understand Transformer Details

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



keras Implementation of Attention **MultiHeadAttention() layer**

```
num_heads = 4
embed_dim = 256
mha_layer = MultiHeadAttention(num_heads=num_heads, key_dim=embed_dim)
outputs = mha_layer(inputs=query, inputs=key, inputs=values)
```



Conceptually, this is what Transformer-style attention is doing. You've got a reference sequence that describes something you're looking for: the query. You've got a body of knowledge that you're trying to extract information from: the values. Each value is assigned a key that describes the value in a format that can be readily compared to a query. You simply match the query to the keys. Then you return a weighted sum of values.

Positional Encoding in Transformers

1. **Objective of Positional Encoding:**

   - Give the model access to word order information.

   - Enhance word embeddings with positional details.

2. **Components of Input Word Embeddings:**

   - Word vector: Represents the word independently.

   - Position vector: Represents the word's position in the sentence.

3. **Concatenation Scheme:**

   - Simplest approach: Concatenate word's position to its embedding vector.

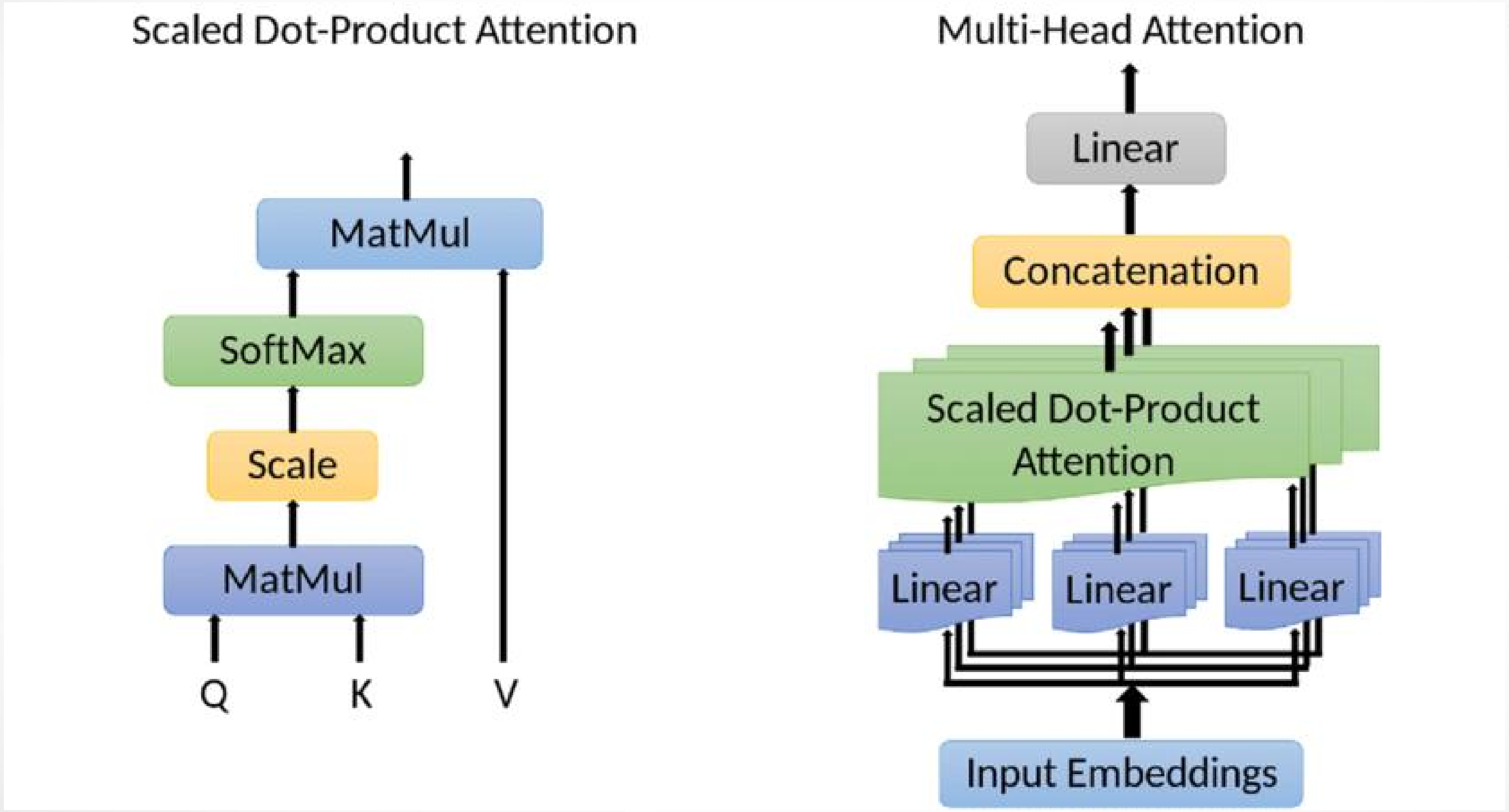
   - Add a "position" axis filled with 0 for first word, 1 for the second, and so on.

4. **Advantages of Positional Embedding:**

   - Simplicity and effectiveness in capturing word positions.

   - Enhances the model's ability to understand sequential information.

The notion of Multihead



Example of an Image Search Algorithm

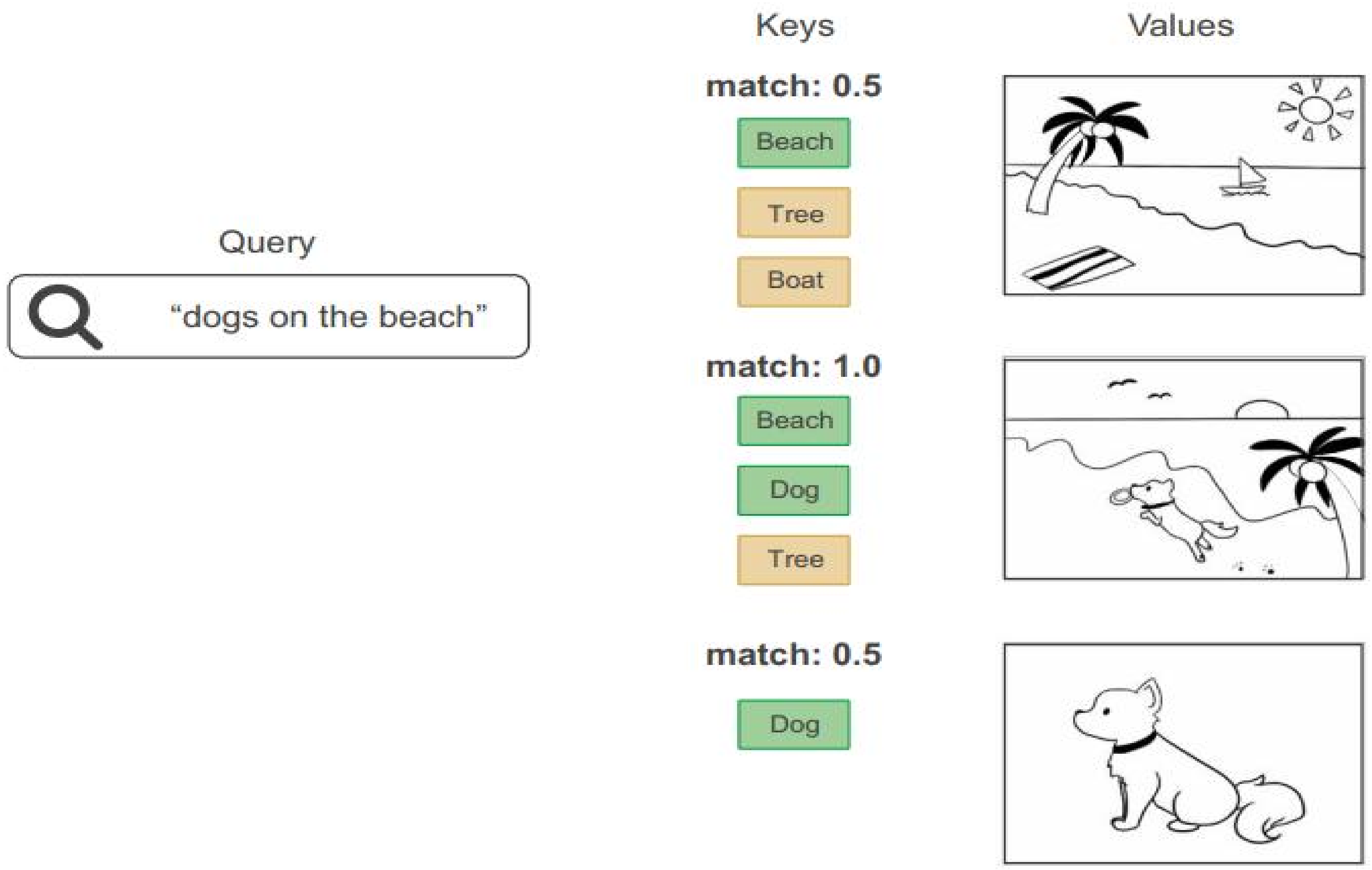
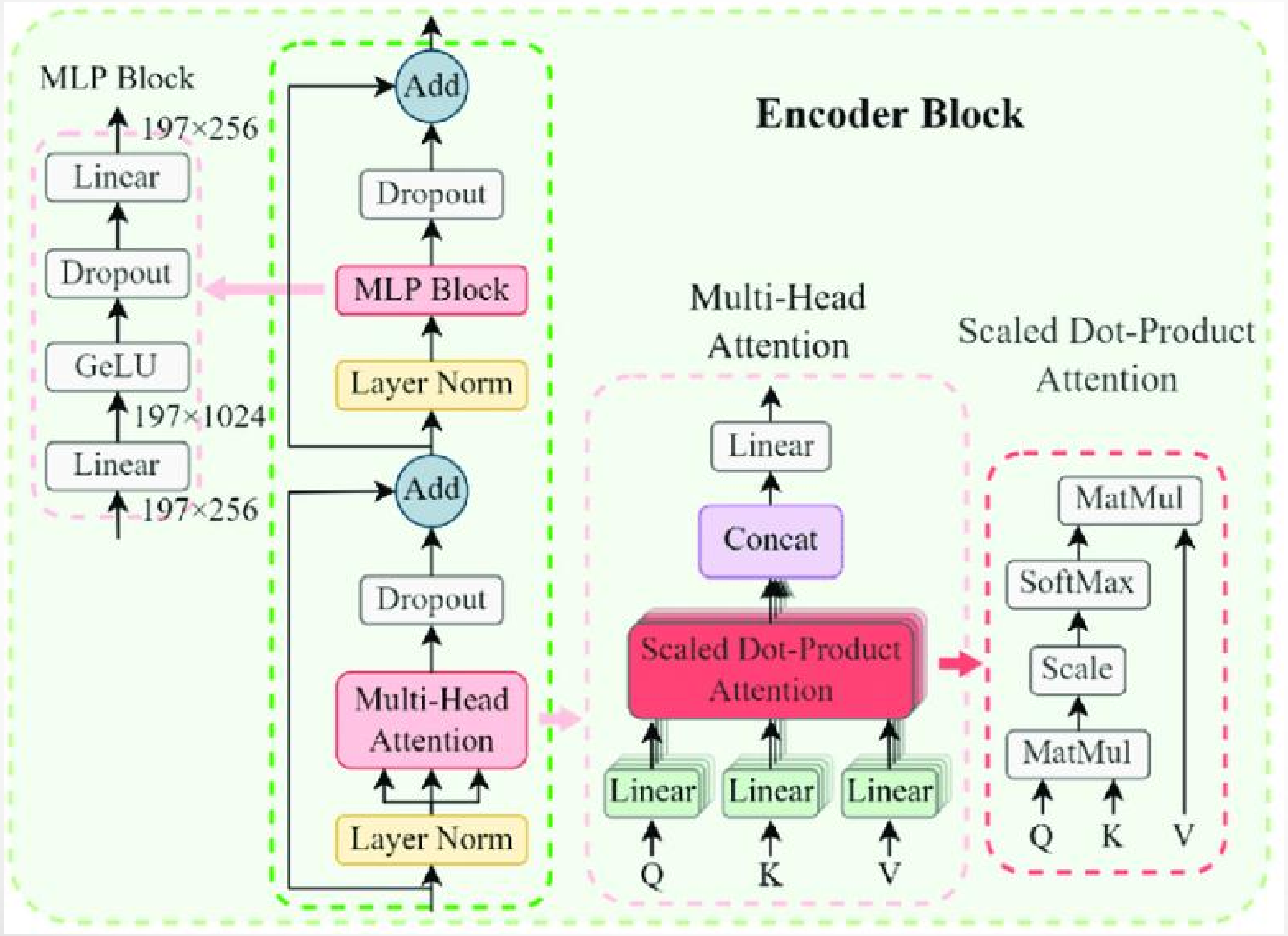
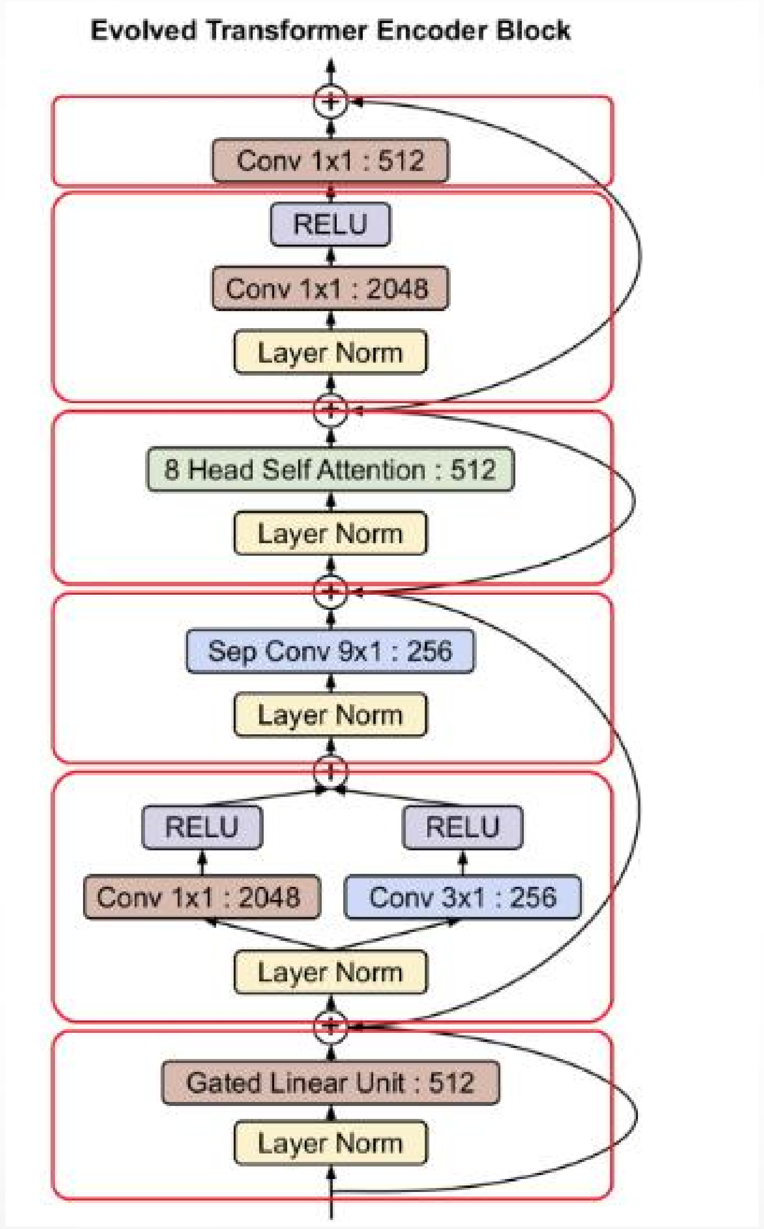


Figure 11.7 Retrieving images from a database: the "query" is compared to a set of "keys," and the match scores are used to rank "values" (images).

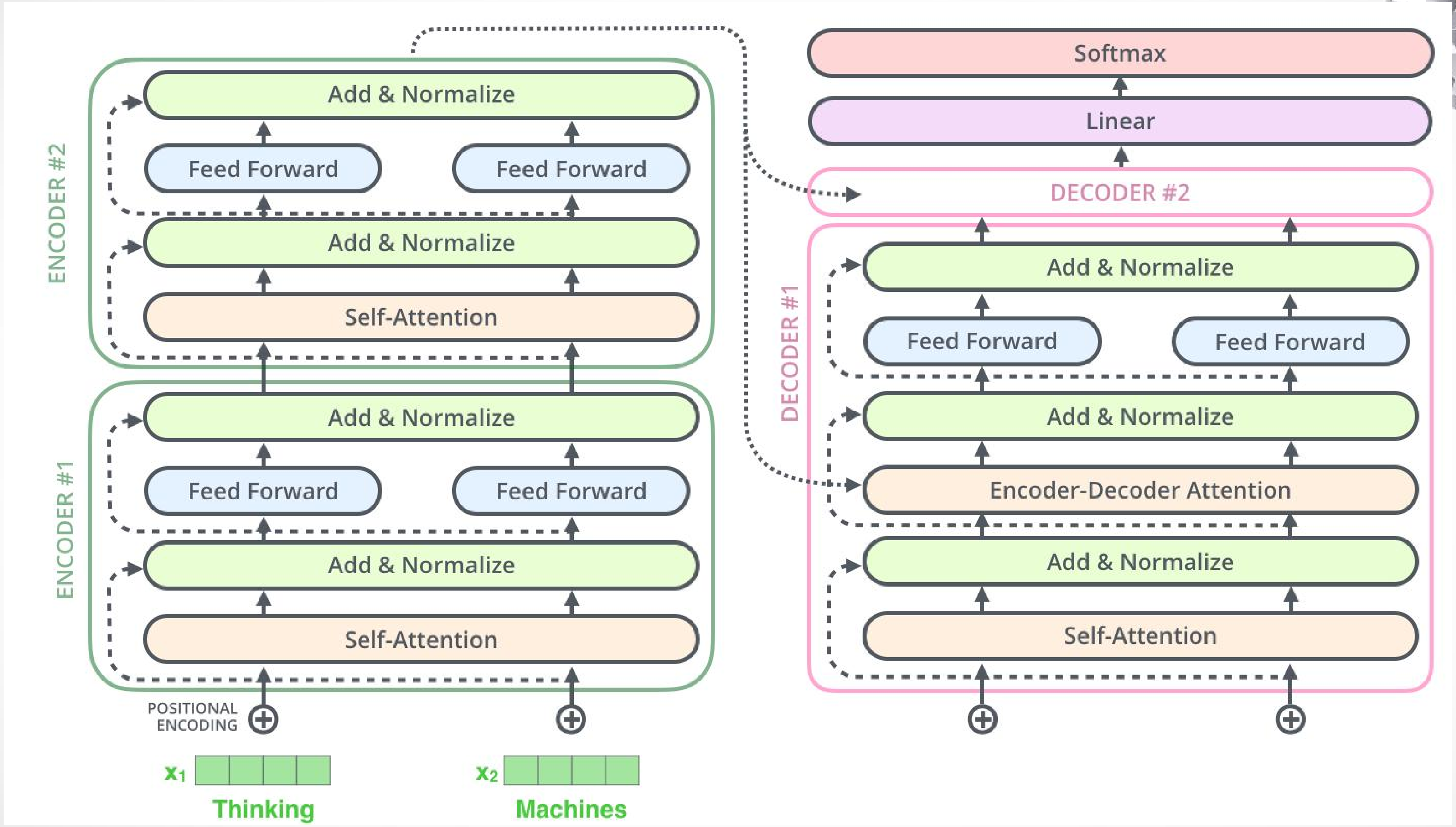
Encoder Block

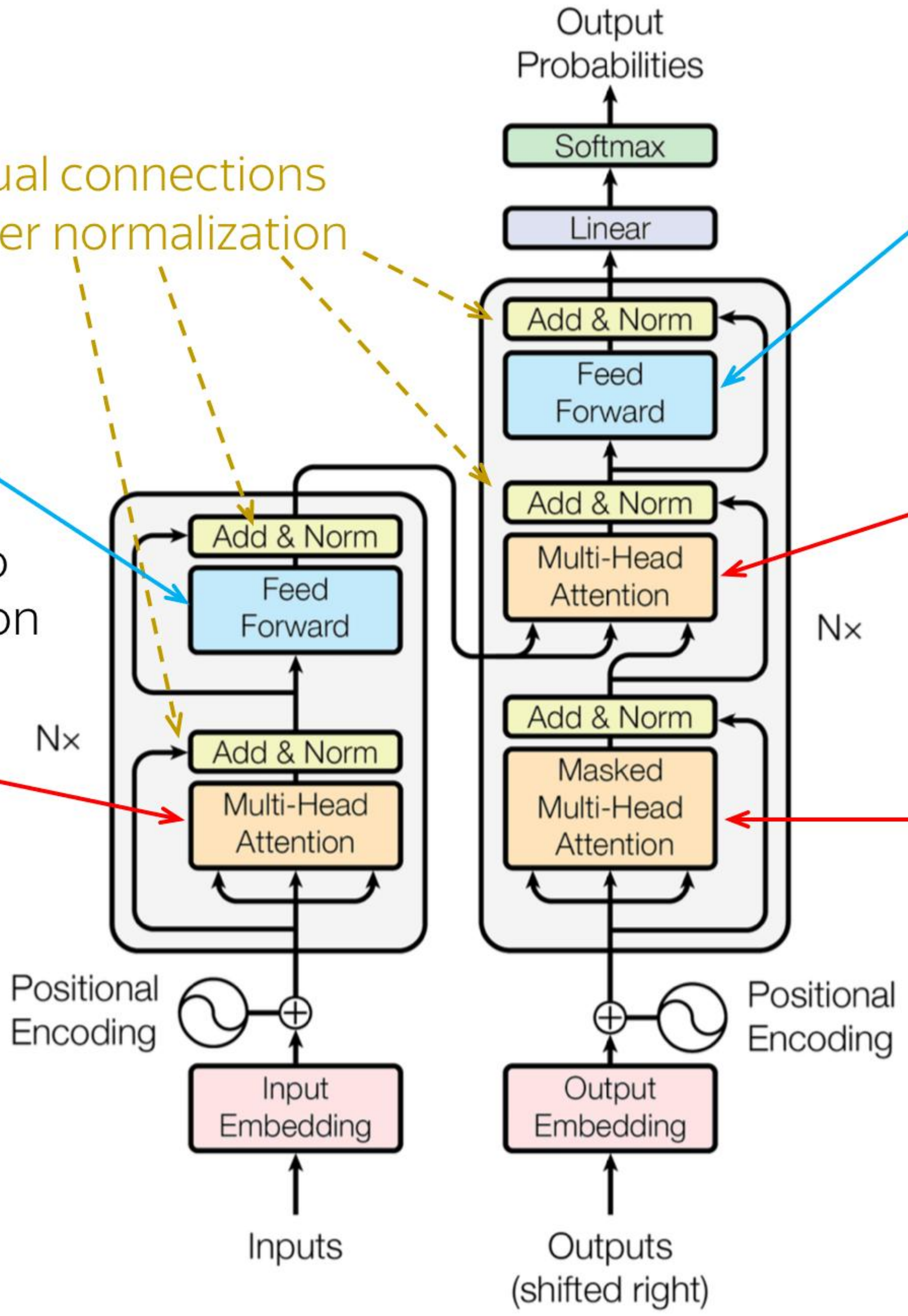


A typical layered structure



Decoder Block





Residual connections and layer normalization

Feed-forward network: after taking information from other tokens, take a moment to think and process this information

Encoder self-attention: tokens look at each other

queries, keys, values are computed from encoder states

Feed-forward network: after taking information from other tokens, take a moment to think and process this information

Decoder-encoder attention: target token looks at the source

queries – from decoder states; keys and values from encoder states

Decoder self-attention (masked): tokens look at the previous tokens

queries, keys, values are computed from decoder states

USE CASES - Transformer Encoder

The transformer encoder architecture is used for tasks like text classification, sentiment analysis, topic classification, or spam detection.

The encoder takes in a sequence of tokens and produces a fixed-size vector representation of the entire sequence, which can then be used for classification.

One of the most popular transformer encoder models is BERT (Bidirectional Encoder Representations from Transformers), which was introduced by Google in 2018.

USE CASES - Transformer Decoder

- The transformer decoder architecture is used for tasks like language generation, where the model must generate a sequence of words based on an input prompt or context. The decoder takes in a fixed-size vector representation of the context and uses it to generate a sequence of words one at a time, with each word being conditioned on the previously generated words.
- One of the most popular transformer decoder models is GPT-3 (Generative Pre-trained Transformer 3), which was introduced by OpenAI in 2020.

USE CASES - Transformer Encoder-Decoder

The transformer encoder-decoder architecture is used for tasks like language translation, where the model must take in a sentence in one language and output a sentence in another language.

The encoder takes in the input sentence and produces a fixed-size vector representation of it, which is then fed into the decoder to generate the output sentence.

One of the most popular transformer encoder-decoder models is the T5 (Text-to-Text Transfer Transformer), which was introduced by Google in 2019.

Popular Models

- BERT: Bidirectional Encoder Representations from Transformers, used for tasks like sentiment analysis, topic classification, or spam detection [3].
- GPT-3: Generative Pre-trained Transformer 3, a massive language model that can generate human-like text in a wide range of styles and genres
- T5: Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer, used for a wide range of NLP tasks, including language translation, question answering, summarization, and more.
- GPT models: Generative Pre-Training models such as GPT-3, ChatGPT and BLOOM: BigScience Large Open-science Open-access Multilingual Language Model, used for tasks like text generation, chatbots, and more.
- Llama-2 Models

Recommended Links

https://www.tensorflow.org/tutorials

https://keras.io/examples/

https://pyimagesearch.com/category/deep-learning/

https://store.augmentedstartups.com/

https://omdena.com/projects/