transformers

motivation, intuition, and models

Contextualized embedding

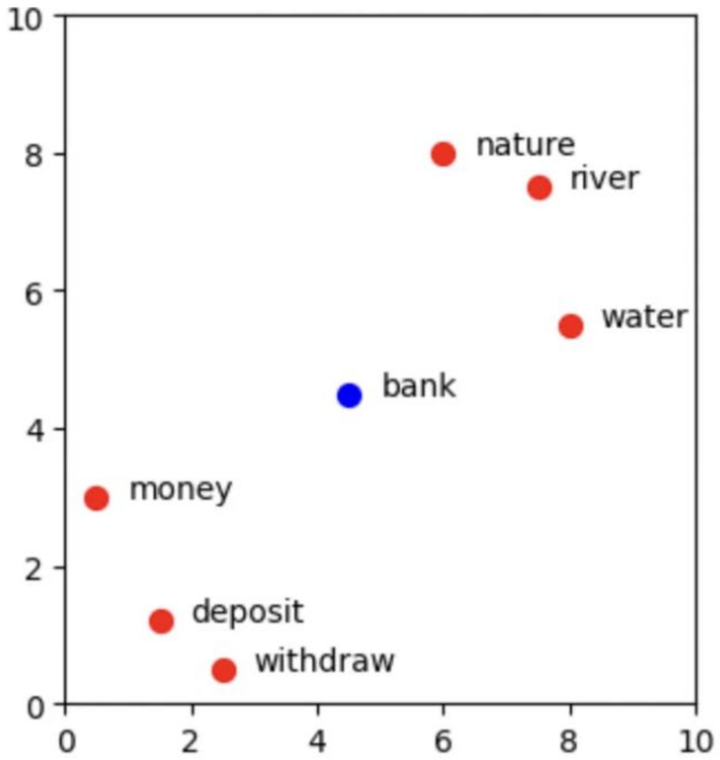
The issue with static word embeddings

Multiple word senses

- each word has just one embedding
- embeddings of words with *multiple senses* will be a average of word senses' (latent) embeddings, weighted by their relative frequency in the training corpus

Example

- “I will hike along the **bank** of a river.”
- “I will open a new account at my **bank** and deposit some money.”



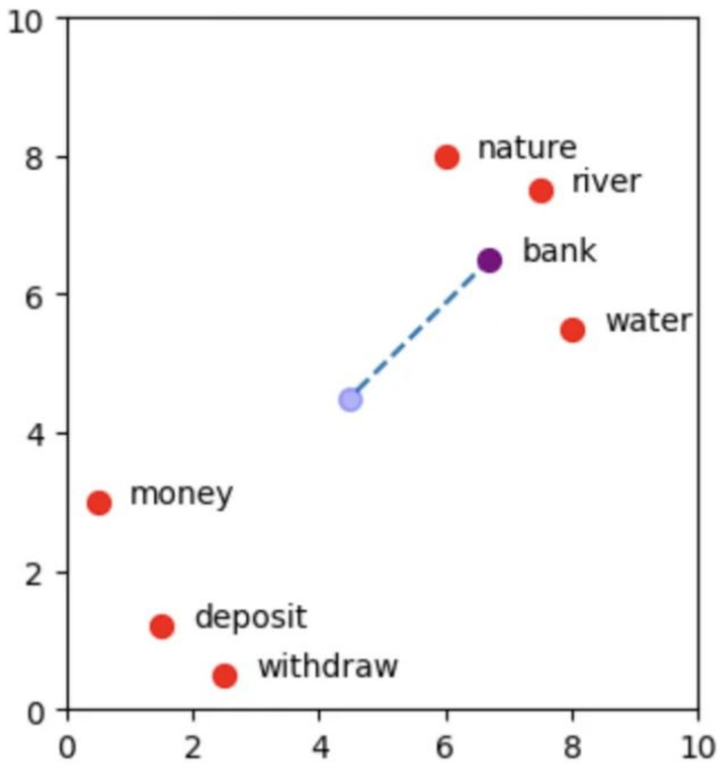
Contextualization to the rescue!

contextualization: infer the meaning of a word based on the the (local) context in which its used

Example

- "I will hike along the **bank** of a river."
- "I will open a new account at my **bank** and deposit some money."

Contextualized *embedding*: an embedding of a word that reflects the local context of how it is used in the input sequence



Attention

But how to contextualize? Not all words in a focal word's context are equally informative

"*attention* is all you need"!

attention is a method for computing how much relevant context information each of the words surrounding a focal word in a sequence contribute to "understand" the focal word's meaning

Illustration of the idea of attention

Sentence 1

sentence: "I will hike along the **bank** of a river."

attention: I will hike along the **bank** of a river.

note: shading indicates attention weight

Sentence 2

sentence: "I will open a new account at my **bank**."

attention: I will open a new account at my **bank**.

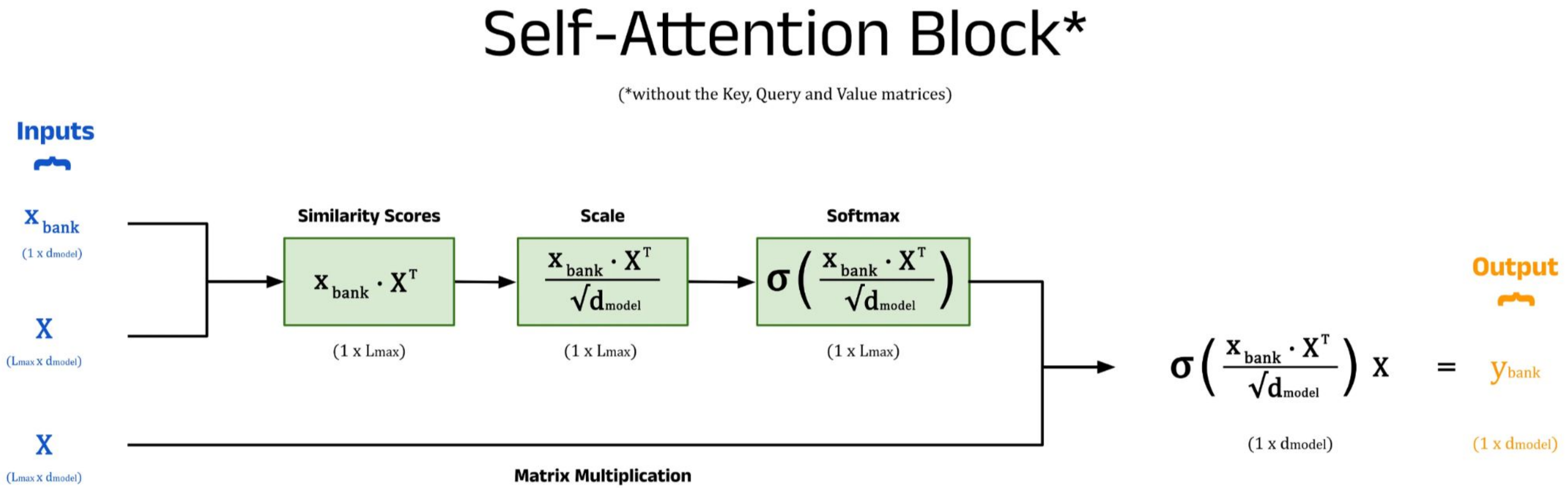
Transformer embeddings

Exercise: Understanding attention, autodidactically

- go to this article, section 3.3
- try to be able to explain how an **input embedding** gets **contextualized** through the attention mechanism in transformer models
- focus on the following question
    - Where do the **input embeddings** come from?
    - Why do we need the input embeddings of a focal word and its surrounding words to contextualize it?
    - What’s the purpose of computing the **dot product**?
    - What’s the purpose of the **softmax function** in the attention block?
    - How does the output of the softmax function in the attention block contribute to **contextualization** for generating the output embedding?

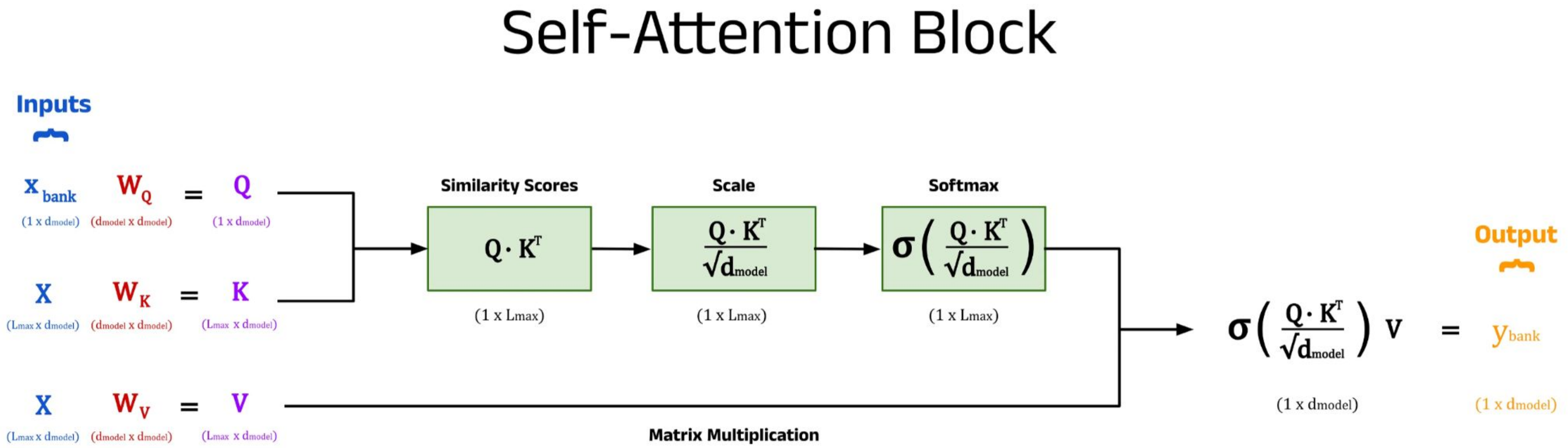
Take 20 minutes, then discuss your notes with your neighbor for 5-10 minutes

Computing attention scores



- Step 1: Take (i) focal word's **input embedding** and (ii) input embeddings of all words in sequence
    - Step 2.1: Compute similarities between focal and other words' embeddings
    - Step 2.2: scale (to reduce impact of high similarity scores)
    - Step 2.3: apply softmax (normalize while rewarding stronger signals) ⇒ **attention weights** that sum to 1
- Step 3: calculate weighted sum of all word embeddings, using attention weights ⇒ **output embedding**

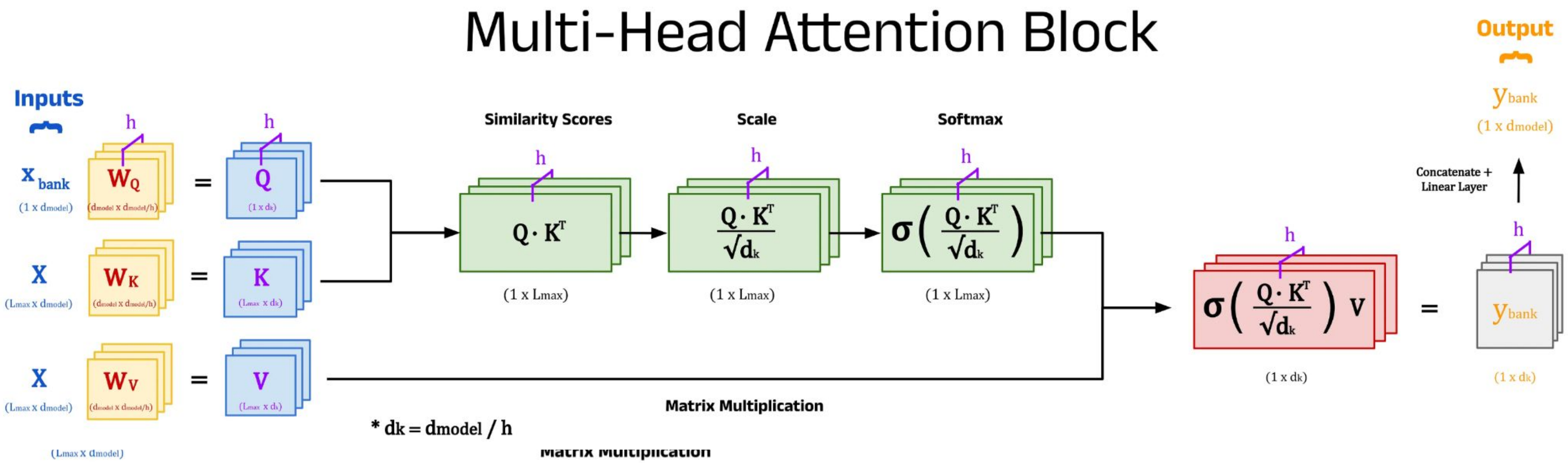
Computing attention scores



in **transformer**

- instead of taking input embeddings **X** as-it, we apply **three distinct matrices** to them: $\mathbf{W}_{Query}$, $\mathbf{W}_{Key}$, $\boldsymbol{W}_{Value}$
- these matrices are ***learnable* parameters** ⇒ makes transformers very capable representation learners

Computing attention scores



in **transformer**

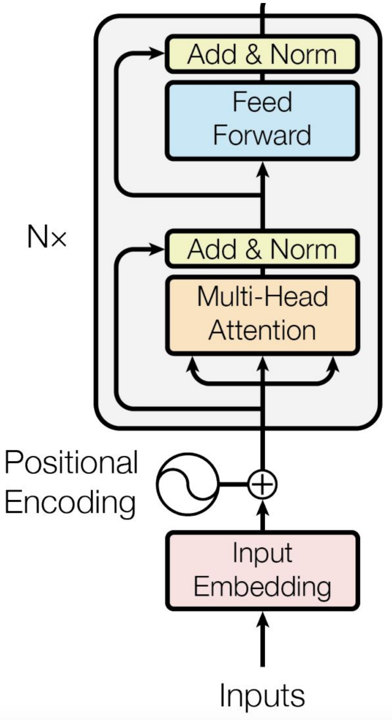
- we chunk the **input embedding** into *h* slices
- each slice is feed to a separate attention block (“head”) with their own $\mathbf{W}_{Query}$, $\mathbf{W}_{Key}$, $\mathbf{W}_{Value}$
- the outputs from each head are concatenated afterwards ⇒ **output embedding**

Stacking layers

The grey-shaded block is a **transformer layer**

Transformer models stack multiple such layers on top of each other

- starts with word embeddings (+ positional encoding)
- processed through first layer yields output embeddings
- the output embeddings from layer *k*-1 are the input embeddings for layer *k*
- repeat until final layer



Hands-on programming & exercises

Let’s code

Open the notebook contextualized_embedding_transformers_explained.ipynb and follow along

Transformer embedding basics …

1. How to **load a pre-trained model** with the python transformers library
2. How to **embed** text(s) with the pre-trained model
3. How to **access a word’s final embeddings**
4. How to **compare** the embeddings of words used in different senses

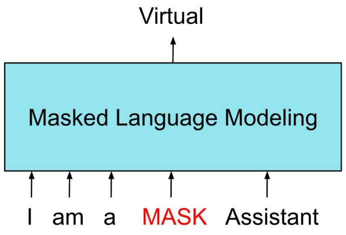
… and advanced stuff

1. How to see to which tokens in the input a transformer model pays **attention** when generating embeddings
2. How to **disambiguate** word meaning with transformer embeddings

language model (LM) pre-training

LM pre-training

- **training** in machine learning: use labeled data (input–output pairs) to optimize a prediction model
- language modeling (LM): different tasks
  - “causal”/autoregressive LM: predict next word from prior sequence of words (e.g., GPT)
  - masked LM: predict masked-out words in sequence of words (e.g., BERT)
- language modeling means *self*-supervised learning (i.e., construct labels from the data)
  - “causal”/autoregressive LM: the observed next word is the label
  - masked LM : the masked-out word is the label



(a)

BERT & Co. use this

LM pre-training

Motivation

- training a model to perform language modeling on large text datasets allows it to mimic humans' text generation behavior
- model learns
  - how words co-occur
  - how sentences, paragraphs, and documents are composed
  - what words, sentences, etc. mean in context
- can be used for *transfer learning* ⇒ that's why it's commonly referred to as *pre*-trained

Transfer learning

machine learning approach to reuse general-purpose model for a specific task

intuition knowledge gained while learning to perform a general task (e.g., language modeling) can be applied to quickly solve related task (e.g., sentiment classification)

premise general features learned during pre-training can be relevant for more specific tasks

Let’s code again

Go back to notebook contextualized_embedding_transformers_explained.ipynb and follow along

Transformer embedding basics …

1. How to **load a pre-trained model** with the python transformers library
2. How to **embed** text(s) with the pre-trained model
3. How to **access a word’s final embeddings**
4. How to **compare** the embeddings of words used in different senses

… and advanced stuff

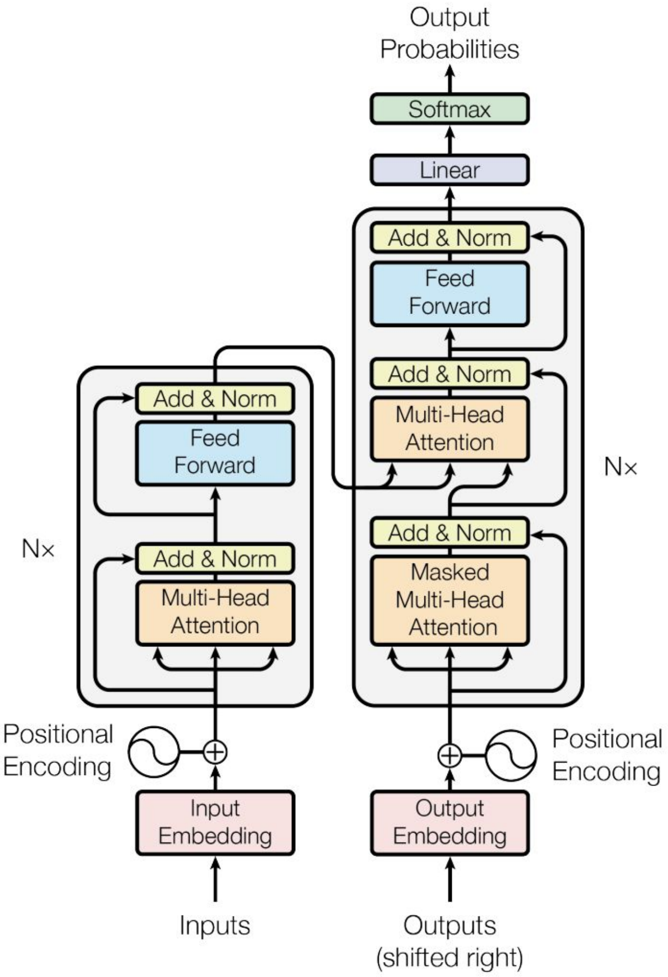
1. How to see to which tokens in the input a transformer model pays **attention** when generating embeddings
2. How to **disambiguate** word meaning with transformer embeddings

encoder vs. decoder models

Encoder vs. decoder models

transformer introduced originally introduced for tasks like **machine translation**

- works best with a "encoder-decoder" architecture
- ***encoder***: takes text in source language and **generates embedding**
- ***decoder***: takes source text embedding and tries to **generate** matching target-language text ("translation")



(Vaswani et al. 2017)

Encoder vs. decoder models

Feature	Encoder-Only Models	Decoder-Only Models
Architecture	Only uses the encoder part of the Transformer	Only uses the decoder part of the Transformer
Attention Mechanism	*Bi*directional attention: can attend to all tokens in both directions	*Uni*directional attention: attends only to previous tokens
Pre-training Objective	Masked Language Modeling (MLM) where some tokens are masked and the model predicts them using context	Auto-regressive language modeling, predicting the next token in a sequence given previous tokens
Intended Applications	Natural language understanding tasks such as text classification or named entity recognition	Text generation tasks such as language modeling, story generation, and dialogue systems
Context Understanding	Can understand the full context by attending to all tokens at once	Generates text by building context token by token, focusing on past context
Output Generation	Not designed for generating new sequences; focuses on “understanding” the input	Generates output sequences token by token, useful for creating coherent text
Examples of Models	BERT, RoBERTa, DistilBERT, DeBERTa	GPTs, LLaMA
in our workshop	Wednesday: transformer encoder fine-tuning	Thursday: few- and zero-shot LLM prompting

Sentence and document embedding

Word vs. sequence embedding with transformers

- obtaining contextualized embeddings for words is nice (thanks BERT!)
- but often our unit of measurement (or even unit of analysis) are **sequences of text** like sentences or paragraphs
- enter stage **sentence BERT**
  - re-uses pre-trained BERT
  - but fine-tunes it on labeled data sets
  - the goal is to get similar embeddings for pairs of texts that have been assigned into similar categories , marked as similar, or etc. by human annotators
- BUT on thursday we'll use generative (decoder) LLM to generate embeddings for longer seqs

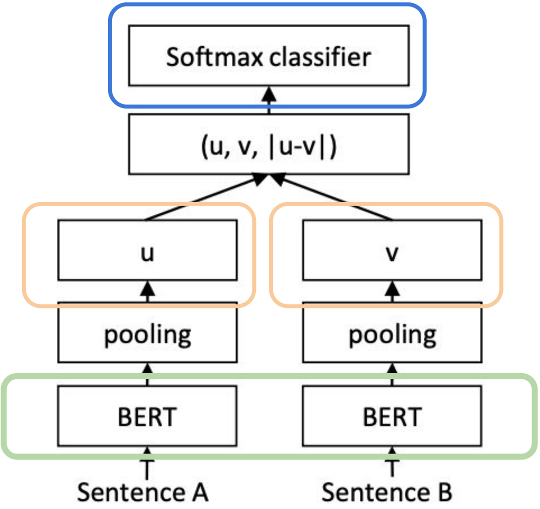


Figure 1: SBERT architecture with classification objective function, e.g., for fine-tuning on SNLI dataset. The two BERT networks have tied weights (siamese network structure).