(tiny and tidy)

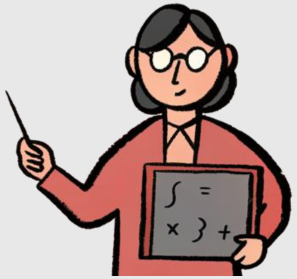
Language Models

Leonardo Cotta, 2025

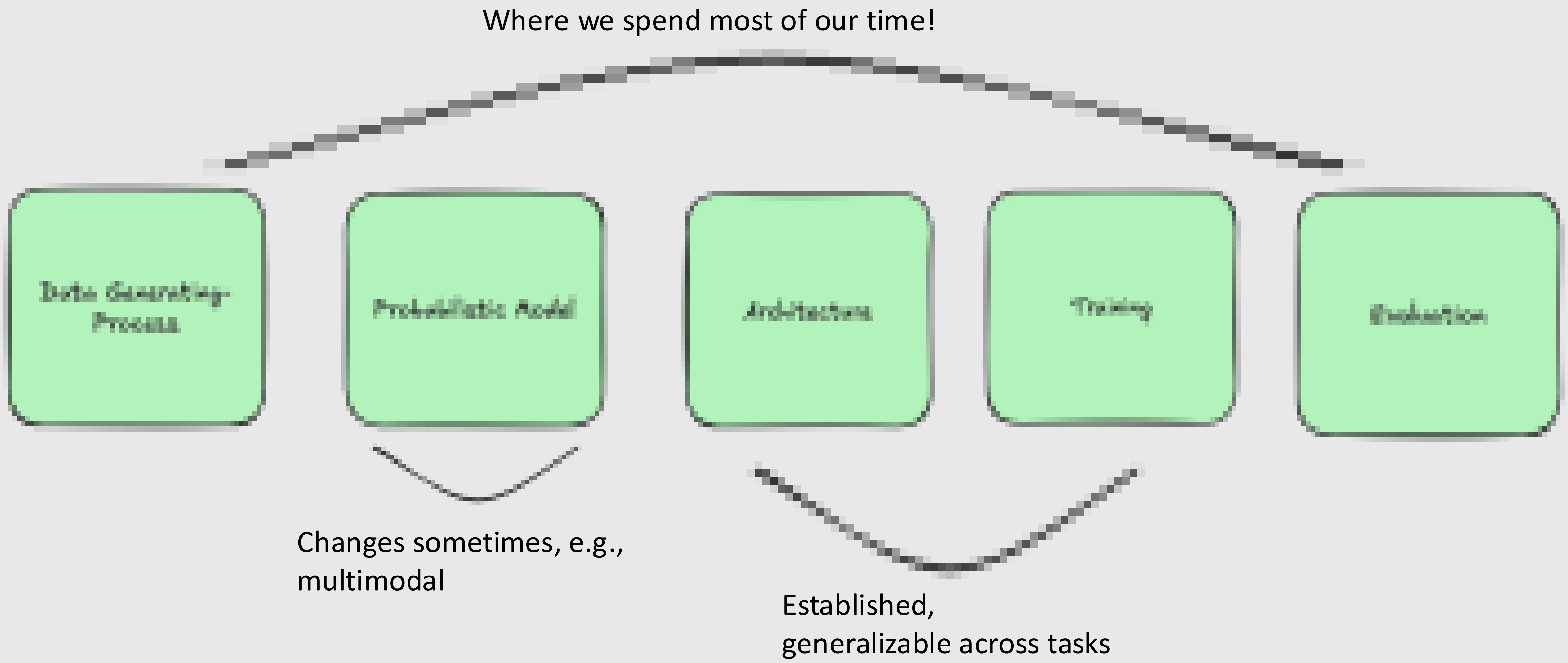
A modern problem

- For most of us, it feels overwhelming to keep up with LLM progress… How can we do it?
- Our take in this two-day course:

Fundamentals of AI + Just do it!



How to think about modern AI projects



Where we spend most of our time!

Changes sometimes, e.g., multimodal

Established, generalizable across tasks

Our focus: Pretraining

Due to time restrictions, we will focus on the pretraining part of modern language models and discuss how to learn and think about post-training, RL, etc. in the end!

PART I:

- Data-generating process;
- Probabilistic Model.

Language Data

- What’s a language sample x?
  - Something that somebody wrote down, i.e., “a document”.
- What’s the data-generating process of these docs?
  - People writing on Wikipedia, Shakespeare writing plays, or everyone writing on the Internet.
  - Each data-generating process induces a probability density of docs p(x).
- How can we model p(x)?

Language Model

- Before we make our modeling choices, we need to specify more things about the random variable X.
- What’s the support of X?
  - Sequence of characters? [I][ ][a][m][ ][a][ ][r][o][b][o][t].
  - Supp(x):= V* (product space).
- Important: What do we really want to do?
  - Sample from some approximation x ;$\sim P_{\theta}(x)$
  - Sample from an arbitrary conditional y. $\sim P_{\theta}(y|x)$
    - Note: y is a completion of x.

Language Model

- The magic of Bayes Theorem:
  - Auto-regressive models solve both problems!

$$p(x) = p(x_0) \prod_i p(x_i | x_0, \dots, x_{i-1})$$

[p(xi| x-i)]

- Mhm… what about p(x0)?
  - Let’s just add a special symbol <bos> to the vocab, and always start with it. Then, we don’t need to learn the marginal, since we know p(x)= 1, if x = <bos>, 0 otherwise.
- Our goal is now to learn p(x) by approximating its conditionals p(xi | x-i)!

Learning

- A model for next-token (letter) prediction:

$$p_{\vartheta}(x) : V^* \to [0,1]^{|V|}$$

- How do we define the loss of the model over a single document x?
  - Let’s do it together, think of Maximum-Likelihood Estimation.

$$L(x,\theta) = -\log p_{\theta}(x)$$

  - Now, with i.i.d. samples, we just average their losses.

Learning

- We are ready to apply SGD and learn the model, as you learned in a previous NN course ;)
- Before we discuss what's the actual functional form of the model, let's discuss the following:
- Does our vocab need to contain characters?
  - Why are words and characters not used?
  - What tokenizers do you know?

PART II:

- Tokenization.

Why do we need tokenizers?

- Why do we think words are better than characters?
  - Or is it the other way around?
  - Both?
  - Why not bytes?
- Characters:
  - Documents become huge sequences;
  - Lot of redundancy.
- The distribution of words is usually quite skewed;
  - Some words are very rare, hard to learn on their own;
  - They can share information with others via sub-words!

Tokenization and Compression

- Words decrease sequence size but reserve symbols in the vocab that we barely use.
    - We can have a unique symbol per example in the corpus, and not learn anything.
    - Larger vocab -> larger sample complexity.
- What do we want?
    - Balance sequence and vocab size.
- Idea:
    - Given a fixed vocab size, that we define;
    - Find the vocab that minimizes the average sequence length and appear often!

BPE: The current winner

- Most modern LLMs use a very simple tokenization algorithm, which is also a compression algorithm from 1994: Byte-pair encoding!

For an N-size vocab:

1. Start with characters;
2. Repeatedly merge most frequent adjacent pair;
3. Stop after N merges.

PART III:

- Transformers go brrr.

Functional form

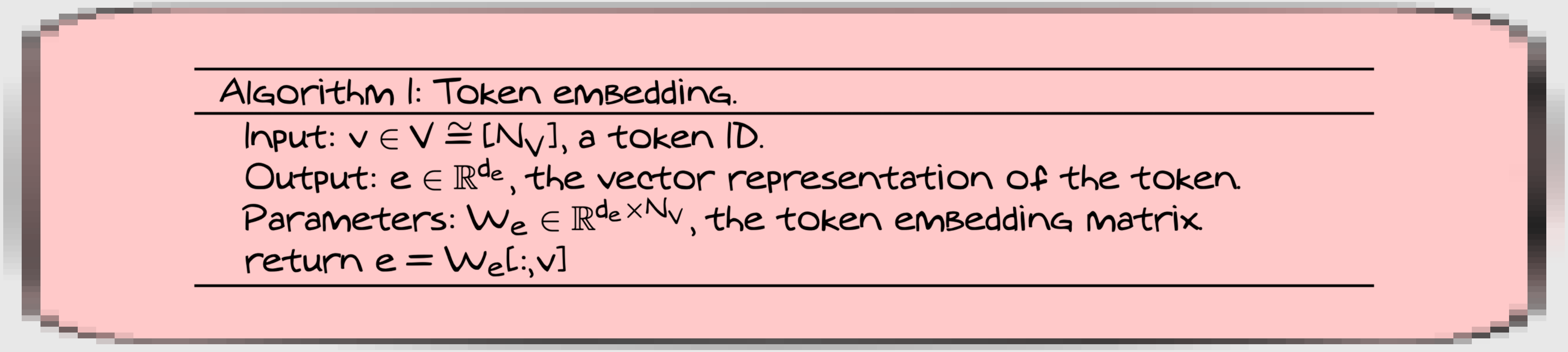
- Recall that we are now left with the task of designing a parametric function (prob density)

$$p_\vartheta(x) : V^* \to [0,1]^{|V|}$$

- Input: context (prefix).
- Output: next token.

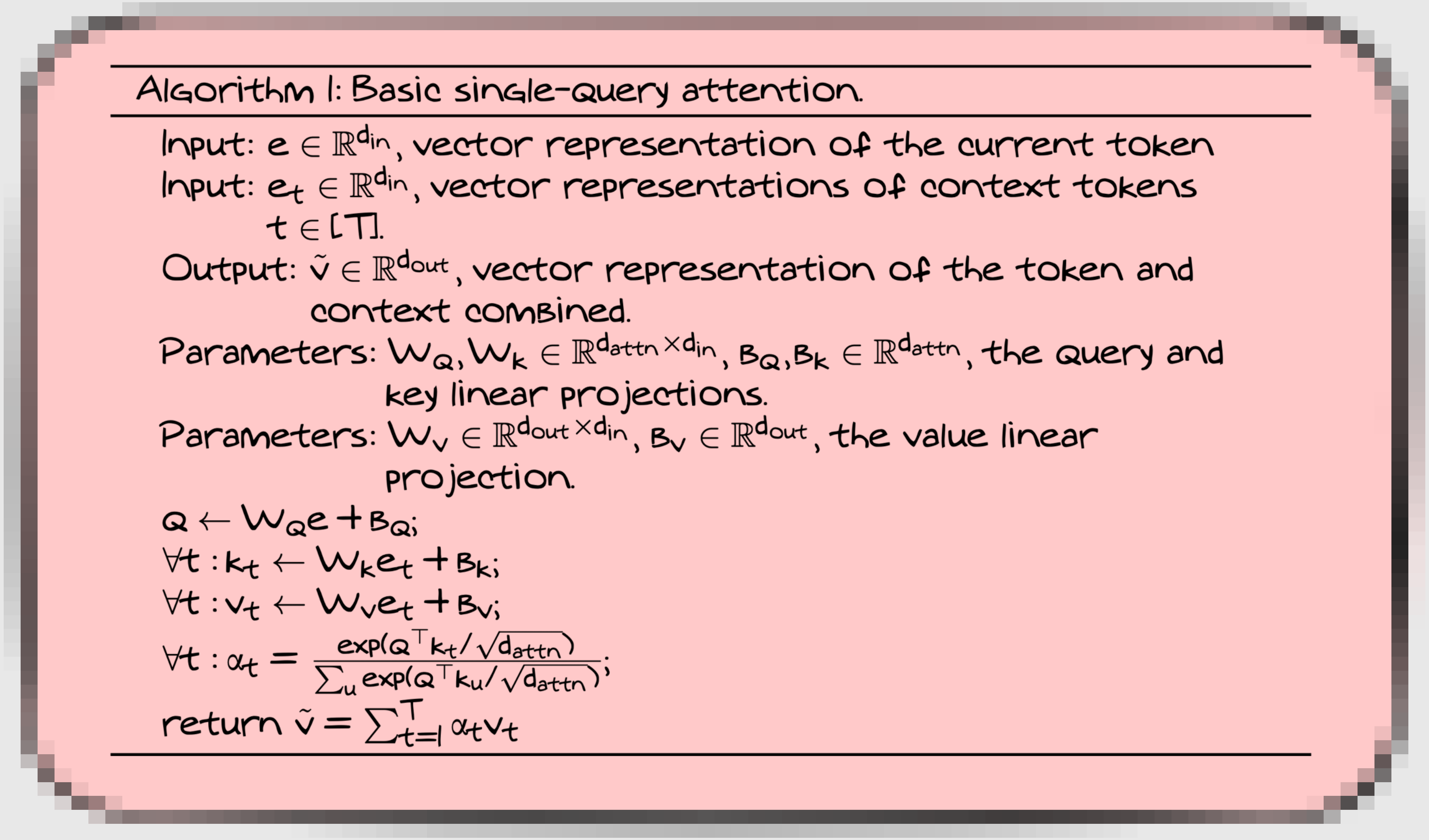
Attention in a nutshell

- Attention is a function mapping a sequence of embeddings to another.



```
Algorithm 1: Token embedding.
  Input: $v \in V \cong [N_V]$, a token ID.
  Output: $e \in \mathbb{R}^{d_e}$, the vector representation of the token.
  Parameters: $W_e \in \mathbb{R}^{d_e \times N_V}$, the token embedding matrix.
  return $e = W_e[:, v]$
```

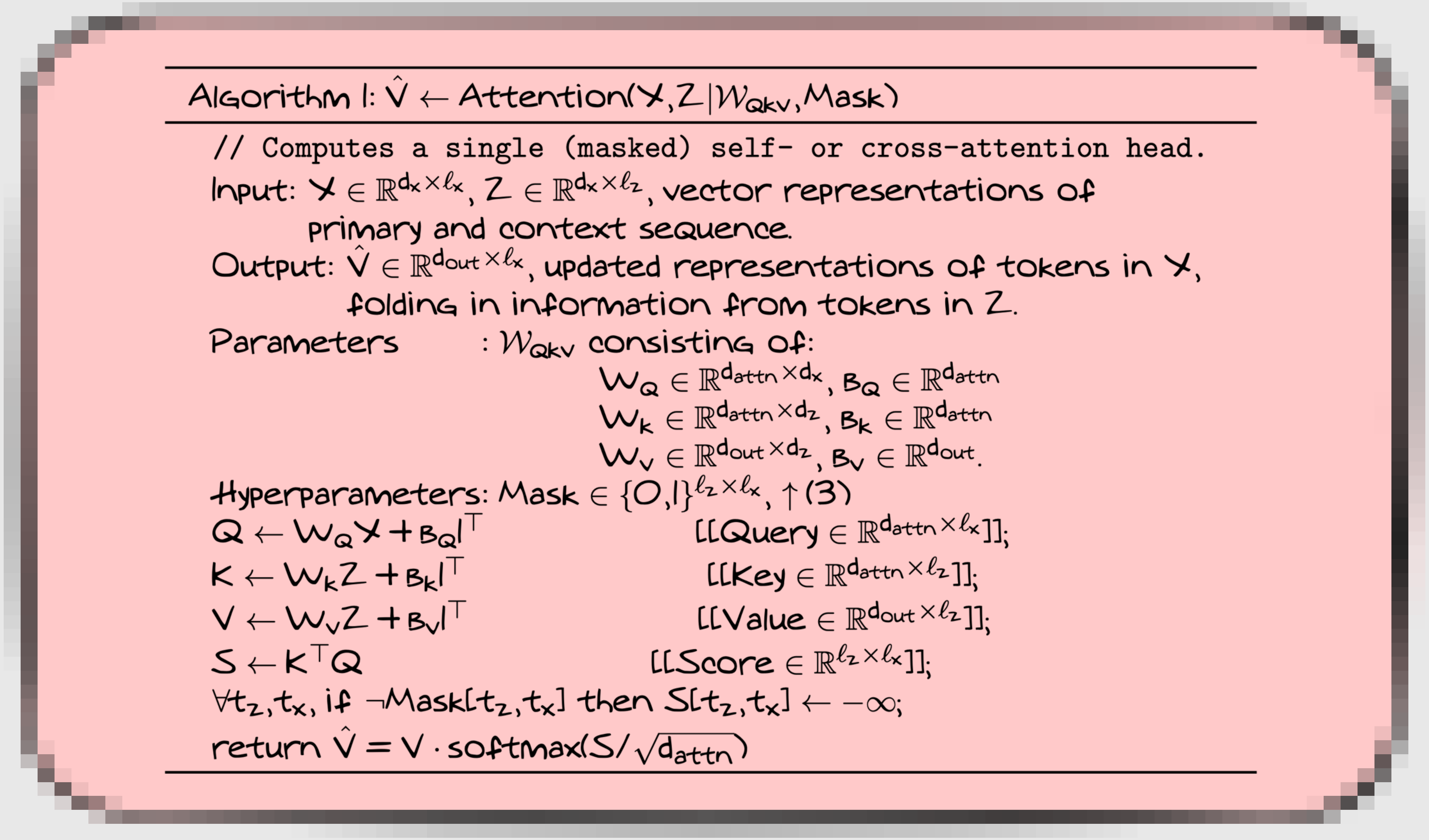
Attention in a nutshell



Algorithm 1: Basic single-query attention.

```
Input: $e \in \mathbb{R}^{d_{in}}$, vector representation of the current token
Input: $e_t \in \mathbb{R}^{d_{in}}$, vector representations of context tokens
        $t \in [T]$.
Output: $\tilde{v} \in \mathbb{R}^{d_{out}}$, vector representation of the token and
        context combined.
Parameters: $W_Q, W_k \in \mathbb{R}^{d_{attn} \times d_{in}}$, $b_Q, b_k \in \mathbb{R}^{d_{attn}}$, the query and
        key linear projections.
Parameters: $W_v \in \mathbb{R}^{d_{out} \times d_{in}}$, $b_v \in \mathbb{R}^{d_{out}}$, the value linear
        projection.
$Q \leftarrow W_Q e + b_Q$;
$\forall t : k_t \leftarrow W_k e_t + b_k$;
$\forall t : v_t \leftarrow W_v e_t + b_v$;
$\forall t : \alpha_t = \frac{\exp(Q^\top k_t / \sqrt{d_{attn}})}{\sum_u \exp(Q^\top k_u / \sqrt{d_{attn}})}$;
return $\tilde{v} = \sum_{t=1}^{T} \alpha_t v_t$
```

Attention in a nutshell



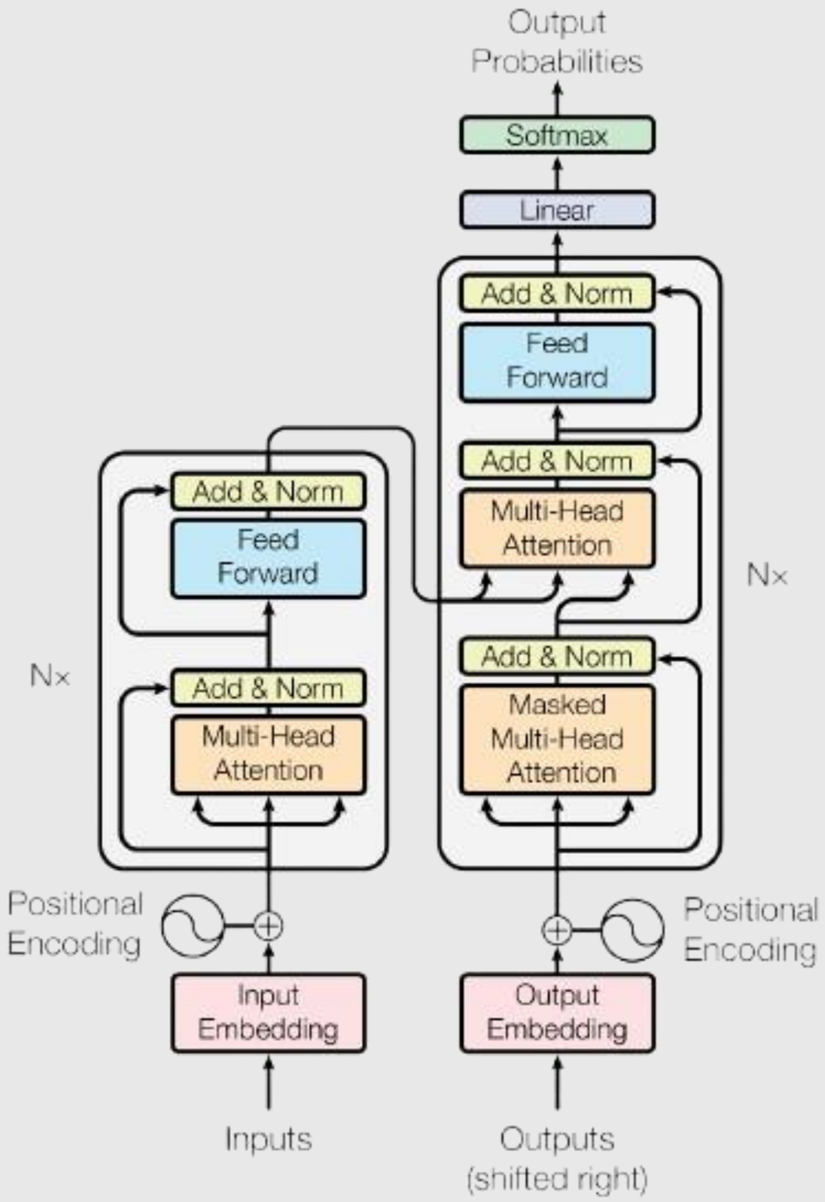
```
Algorithm 1: $\hat{V} \leftarrow \mathrm{Attention}(X, Z \mid \mathcal{W}_{\mathrm{qkv}}, \mathrm{Mask})$
    // Computes a single (masked) self- or cross-attention head.
    Input: $X \in \mathbb{R}^{d_x \times \ell_x}$, $Z \in \mathbb{R}^{d_x \times \ell_z}$, vector representations of
            primary and context sequence.
    Output: $\hat{V} \in \mathbb{R}^{d_{\mathrm{out}} \times \ell_x}$, updated representations of tokens in $X$,
            folding in information from tokens in $Z$.
    Parameters : $\mathcal{W}_{\mathrm{qkv}}$ consisting of:
            $W_q \in \mathbb{R}^{d_{\mathrm{attn}} \times d_x}$, $b_q \in \mathbb{R}^{d_{\mathrm{attn}}}$
            $W_k \in \mathbb{R}^{d_{\mathrm{attn}} \times d_z}$, $b_k \in \mathbb{R}^{d_{\mathrm{attn}}}$
            $W_v \in \mathbb{R}^{d_{\mathrm{out}} \times d_z}$, $b_v \in \mathbb{R}^{d_{\mathrm{out}}}$.
    Hyperparameters: Mask $\in \{0,1\}^{\ell_z \times \ell_x}$, ↑(3)
    $Q \leftarrow W_q X + b_q 1^{\top}$    [[Query $\in \mathbb{R}^{d_{\mathrm{attn}} \times \ell_x}$]];
    $K \leftarrow W_k Z + b_k 1^{\top}$    [[Key $\in \mathbb{R}^{d_{\mathrm{attn}} \times \ell_z}$]];
    $V \leftarrow W_v Z + b_v 1^{\top}$    [[Value $\in \mathbb{R}^{d_{\mathrm{out}} \times \ell_z}$]];
    $S \leftarrow K^{\top} Q$    [[Score $\in \mathbb{R}^{\ell_z \times \ell_x}$]];
    $\forall t_z, t_x$, if $\neg \mathrm{Mask}[t_z, t_x]$ then $S[t_z, t_x] \leftarrow -\infty$;
    return $\hat{V} = V \cdot \mathrm{softmax}(S / \sqrt{d_{\mathrm{attn}}})$
```

Attention is almost all you need

- Attention is a linear mechanism that propagates information across a set of embeddings.
    - What’s missing?
- Non-linearity?
- Positional information?
- How do we go to next-token distribution?
- We need the transformer!
    - Attention =/= transformer

All you need (or not)

- How do we do these operations for a batch of sequences?
  - Let’s do it together on the board!



Positional Encodings

- In theory we could just have independent position embeddings, i.e. embedding 1, 2, 3, etc and combine them with the token embeddings.
    - Why is that a bad idea?
    - Sharing information is a deep learning secret.

Sinusoidal encodings

For each position pos $\in \{1, 2, \ldots, n\}$ and dimension i $\in \{0, 1, \ldots, d-1\}$, the positional encoding is defined as:

$$PE_{(pos,2i)} = \sin\left(\frac{pos}{10000^{2i/d}}\right)$$

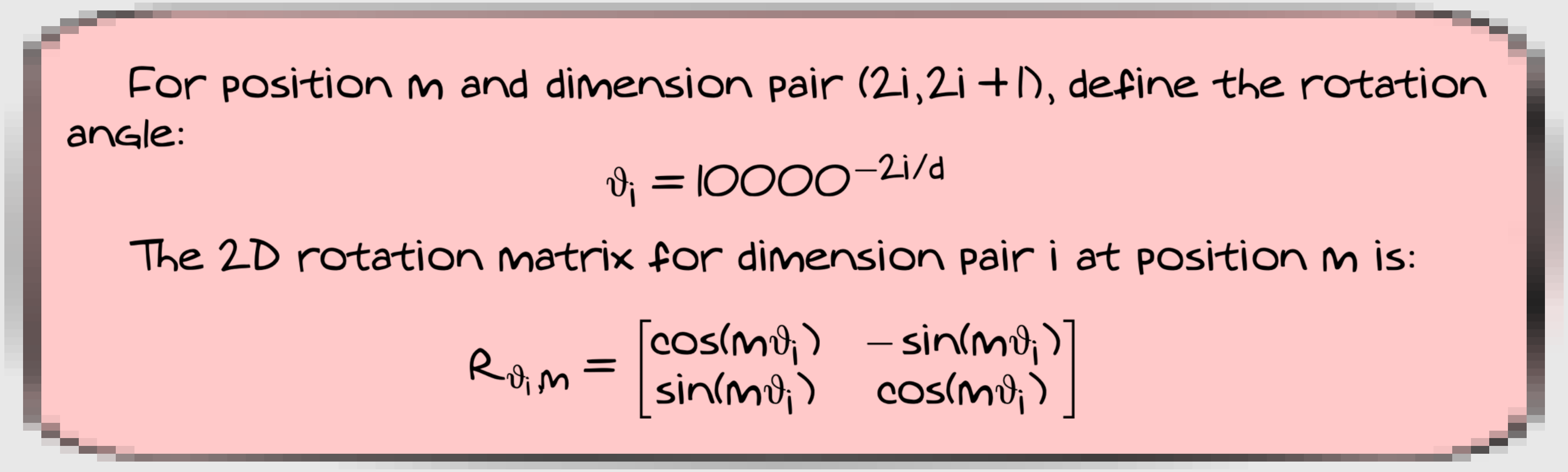
$$PE_{(pos,2i+1)} = \cos\left(\frac{pos}{10000^{2i/d}}\right)$$

where:

- Even dimensions ($2i$) use sine function
- Odd dimensions ($2i+1$) use cosine function
- The wavelength forms a geometric progression from 2π to $10000 \cdot 2\pi$

RoPE

- Rotation Matrix:



For position m and dimension pair $(2i, 2i+1)$, define the rotation angle:

$$\vartheta_i = 10000^{-2i/d}$$

The 2D rotation matrix for dimension pair i at position m is:

$$R_{\vartheta_i, m} = \begin{bmatrix} \cos(m\vartheta_i) & -\sin(m\vartheta_i) \\ \sin(m\vartheta_i) & \cos(m\vartheta_i) \end{bmatrix}$$

RoPE

- Transformation:

For a query or key vector $x_m \in \mathbb{R}^d$ at position m, partition it into pairs:

$$x_m = [x_m^{(1)}, x_m^{(2)}, \ldots, x_m^{(d/2)}]^T$$

where each $x_m^{(i)} = [x_{m,2i}, x_{m,2i+1}]^T \in \mathbb{R}^2$.

Apply rotation to each pair:

$$\tilde{x}_m^{(i)} = R_{\vartheta_i, m} \cdot x_m^{(i)}$$

The full RoPE transformation is:

$$\mathrm{RoPE}(x_m, m) = [\tilde{x}_m^{(1)}, \tilde{x}_m^{(2)}, \ldots, \tilde{x}_m^{(d/2)}]^T$$

RoPE

For each dimension pair $(2i, 2i+1)$:

$$\tilde{x}_{m,2i} = x_{m,2i} \cos(m\vartheta_i) - x_{m,2i+1} \sin(m\vartheta_i)$$

$$\tilde{x}_{m,2i+1} = x_{m,2i} \sin(m\vartheta_i) + x_{m,2i+1} \cos(m\vartheta_i)$$

RoPE

Apply RoPE to queries and keys before computing attention:

$$Q = \mathrm{RoPE}(XW_Q)$$
$$K = \mathrm{RoPE}(XW_K)$$
$$V = XW_V$$

Then compute attention as usual:

$$\mathrm{Attention}(Q,K,V) = \mathrm{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

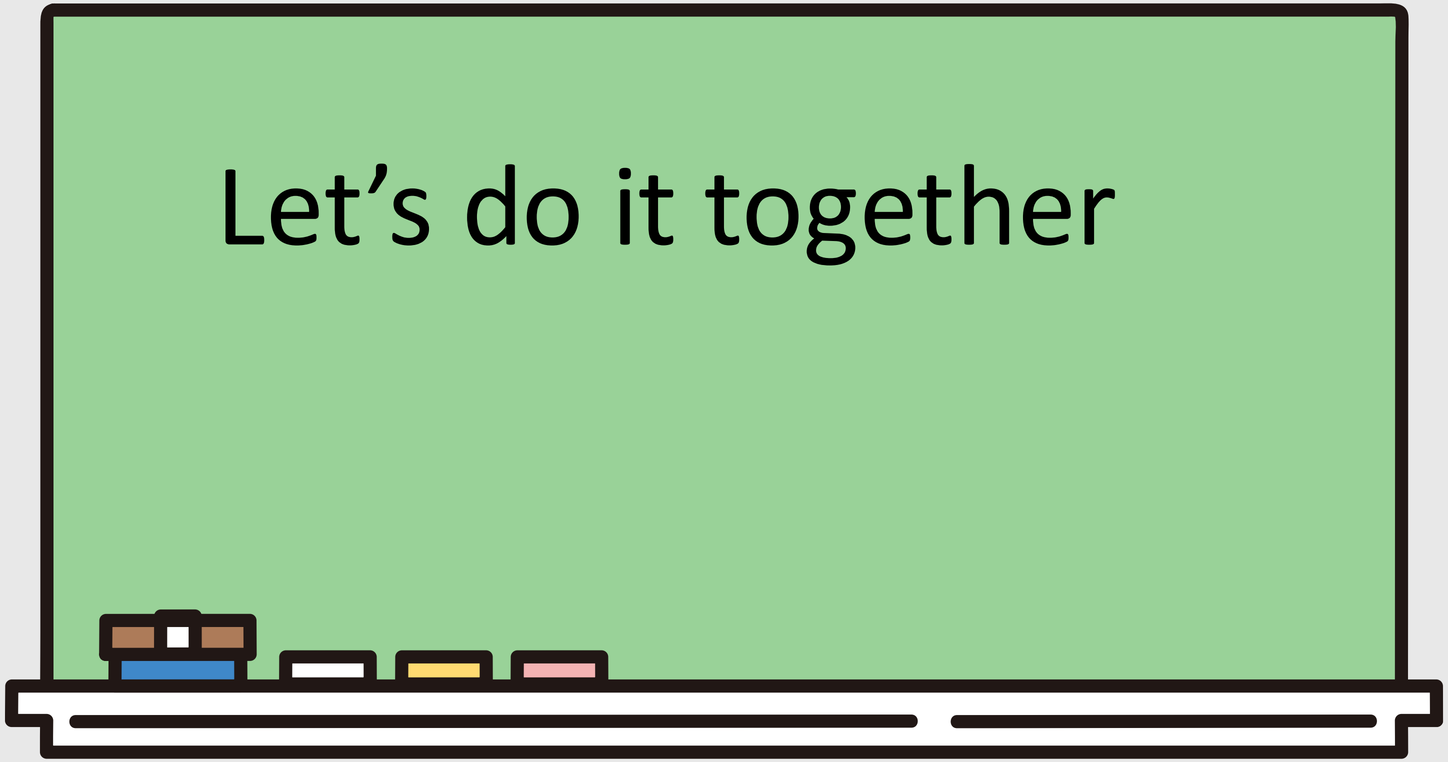
PART IV:

- Inference engine;
- Chat about advanced topics.

How do we sample next tokens?

Let’s look at engine.py

RLHF is just Bayesian Inference



Let’s do it together