Deep learning for pedestrians: backpropagation in Transformers

Laurent Boué

Oracle [*]

Abstract

This document is a follow-up to our previous paper dedicated to a vectorized derivation of backpropagation in CNNs [1]. Following the same principles and notations already put in place there, we now focus on transformer-based next-token-prediction architectures. To this end, we apply our lightweight index-free methodology to new types of layers such as embedding, multi-headed self-attention and layer normalization. In addition, we also provide gradient expressions for LoRA layers to illustrate parameter-efficient fine-tuning. Why bother doing manual backpropagation when there are so many tools that do this automatically? Any gap in understanding of how values propagate forward will become evident when attempting to differentiate the loss function. By working through the backward pass manually, we gain a deeper intuition for how each operation influences the final output. A complete PyTorch implementation of a minimalistic GPT-like network is also provided along with analytical expressions for of all of its gradient updates.

1 Sequence modeling from 20,000 feet...

Data representation. In the following, a token is understood to be any atomic unit of information such as words in natural language, pixels in an image, amino acids in proteins, time stamps in time series forecasting... A "sample" of data is understood to be a sequence of $n_\mathcal{T}$ tokens where the relative arrangement of tokens with respect to each other encodes a meaningful higher-level structure. For instance, in natural language, the meaning of a sentence emerges from the way sequences of words are combined with each other to convey higher-level ideas. Similarly, in computer vision, while each pixel in an image holds a value (such as color or intensity), higher-level concepts like objects emerge when coherent sequences of pixels are considered together.

Each token t is identified — via a specialized "tokenizer" — as an integer $t \sim \mathbb{N} \in [1, \cdots, n_\text{vocab}]$ where n_vocab corresponds to the maximum number of tokens in our "vocabulary". We refer the reader to [2] for a review of tokenizers for different data modalities. Therefore, one single data input is denoted by a vector of integers $[t_1 \sim \mathbb{N}, \cdots, t_{n_\mathcal{T}} \sim \mathbb{N}] \sim \mathbb{N}^{n_\mathcal{T}}$ of size $n_\mathcal{T}$ corresponding to the number of tokens in the sequence.

Next-token prediction. Let us denote the model as a parametrized function $\mathcal{N}_\mathcal{P}$ that takes as input a sequence of tokens $\mathbf{a}_0 \sim \mathbb{N}^{n_\mathcal{T}}$ and returns a new sequence

$$\mathbf{y}_\text{pred} = \mathcal{N}_\mathcal{P}(\mathbf{a}_0) \sim \mathbb{R}^{n_\mathcal{T} \times n_\text{vocab}}$$

where each token $\sim \mathbb{N}$ is transformed into a normalized probability density vector $\sim \mathbb{R}^{n_\text{vocab}}$ over the vocabulary of tokens. Alongside this probabilistic prediction, each token is also associated with a ground-truth target token which, ideally, should match as best as possible the prediction vector produced by $\mathcal{N}_\mathcal{P}$. Graphically, this can be represented as

$$\mathbf{a}_0 = \begin{pmatrix} t_1 \sim \mathbb{N} \\ \vdots \\ t_{n_\mathcal{T}} \sim \mathbb{N} \end{pmatrix} \longrightarrow \mathbf{y}_\text{pred} = \begin{pmatrix} \mathbf{y}_\text{pred}(t=1) \sim \mathbb{R}^{n_\text{vocab}} \\ \vdots \\ \mathbf{y}_\text{pred}(t=n_\mathcal{T}) \sim \mathbb{R}^{n_\text{vocab}} \end{pmatrix} \quad \text{vs.} \quad \mathbf{y}_\text{gt} = \begin{pmatrix} y_\text{gt}(t=1) \sim \mathbb{N} \\ \vdots \\ y_\text{gt}(t=n_\mathcal{T}) \sim \mathbb{N} \end{pmatrix}$$

[*] (work initiated while at Microsoft)

As usual in classification settings, the mismatch between the prediction and the ground-truth for token t is quantified via the cross-entropy loss function

$$\ell_{\boldsymbol{\mathcal{P}}}\left(\mathbf{y}_{\text{gt}}\,,\mathbf{y}_{\text{pred}}\right) = \begin{pmatrix} -\,\mathbf{y}_{\text{gt}}(t=1)_{\text{OHE}}\cdot\log\,\mathbf{y}_{\text{pred}}(t=1)\sim\mathbb{R} \\ \vdots \\ -\,\mathbf{y}_{\text{gt}}(t=n_{\mathcal{T}})_{\text{OHE}}\cdot\log\,\mathbf{y}_{\text{pred}}(t=n_{\mathcal{T}})\sim\mathbb{R} \end{pmatrix} = -\mathbf{y}_{\text{gt}}\ominus\log\mathbf{y}_{\text{pred}}\sim\mathbb{R}^{n_{\mathcal{T}}}$$

applied independently to all $n_{\mathcal{T}}$ tokens in the sequence and where the ground-truth tokens have been lifted from integers into their equivalent “One Hot Encoded” (OHE) representations.

For next-token prediction tasks (relevant for model pre-training and supervised fine-tuning — SFT) the ground-truth $\mathbf{y}_{\text{gt}}$ is chosen as a “shifted-by-one” version of the input $\mathbf{a}_0$. That is to say that, considering a token $t^\star$ from the input $\mathbf{a}_0$, its target should be $y_{\text{gt}}(t=t^\star)=t^\star+1$ taken from the same $\mathbf{a}_0$. This may be understood graphically as

$$\mathbf{y}_{\text{gt}}=\text{shiftByOne}(\mathbf{a}_0)\quad\Longleftrightarrow\quad\mathbf{y}_{\text{gt}}=\begin{pmatrix} y_{\text{gt}}(t=1)=t_2\sim\mathbb{N} \\ \vdots \\ y_{\text{gt}}(t=n_{\mathcal{T}}-1)=t_{n_{\mathcal{T}}}\sim\mathbb{N} \end{pmatrix}\quad\mathbf{a}_0=\begin{pmatrix} t_1\sim\mathbb{N} \\ t_2\sim\mathbb{N} \\ \vdots \\ t_{n_{\mathcal{T}}}\sim\mathbb{N} \end{pmatrix}$$

Due to this shift-by-one property between the tokens of $\mathbf{y}_{\text{gt}}$ and $\mathbf{a}_0$ in standard next-token prediction tasks, the very first token t_1 in $\mathbf{a}_0$ is not used in the loss function and the number of elements contributing to the loss for a sequence of $n_{\mathcal{T}}$ tokens is reduced to $n_{\mathcal{T}}-1$.

2 Embedding layer

The purpose of embedding layers is to transform categorical variables from their discrete representations — where they exist as static structureless elements of a set — into continuous and “meaningful” d-dimensional vector-based representations $\sim\mathbb{R}^d$ known as “embeddings”. Here, “meaningful” implies that the learned embedding vectors are expected to capture useful and objective-dependent information (as enforced by the loss function).

While tokens are the primary categorical variables for which “token embeddings” are constructed, their relative positions within a sequence also constitute an abstract categorical variable giving rise to “positional embeddings.” In the case of token embeddings, some desirable properties would, for example, be that tokens with similar semantic meanings would be associated with vector representations that are close to each other. Conversely, one would expect positional embeddings to reflect order-sensitive information. Let us consider a specific token $t^\star$ from an input sequence $\mathbf{a}_{i-1}$ of $n_{\mathcal{T}}$ tokens [1]. As a discrete unit of information, this token can always be represented as an integer $\mathbf{a}_{i-1}(t=t^\star)\sim\mathbb{N}$ but the range of possible values $\mathbf{a}_{i-1}(t=t^\star)\in[1,\cdots,n_{\mathcal{V}}]$ depends on whether we look at it from its vocabulary-membership perspective or from its position in the sequence.

- For the purposes of token embeddings, the tokenizer assigns $t^\star$ with $\mathbf{a}_{i-1}^{\text{token}}(t=t^\star)\in[1,\cdots,n_{\text{vocab}}]$ where n_{vocab} denotes the total number of tokens in the vocabulary. The exact integer value serves only to distinguish $t^\star$ from all the other tokens in the vocabulary but carries no inherent meaning.
- On the other hand, for positional embeddings, the integer representation of $t^\star$ would correspond to the index of its position in the sequence of length $n_{\mathcal{T}}$. Therefore $\mathbf{a}_{i-1}^{\text{position}}(t=t^\star)\in[1,\cdots,n_{\text{context}}]$ where n_{context} is the context length of the model, i.e. the maximum number of tokens that the model can handle as a single sequence. Unless the sequence $\mathbf{a}_{i-1}$ has been padded to exactly match the context length, we would have $n_{\text{context}}\geq n_{\mathcal{T}}$.

[1] Typically $\mathbf{a}_{i-1}$ would be the very first data source $i=1$ so that $\mathbf{a}_{i-1}\equiv\mathbf{a}_0$. Nonetheless, we stick to the $\mathbf{a}_{i-1}$ notation for input data into a layer and $\mathbf{a}_i$ for its output to stay with a consistent terminology for all types of layers.

Without loss of generality, let us denote by $n_{\mathcal{V}}$ the number of possible integers that tokens may be associated with and introduce a database-like structure $\mathbf{w}_{\text{emb}} \sim \mathbb{R}^{n_{\mathcal{V}} \times d}$ where each row of $\mathbf{w}_{\text{emb}}$ contains the "embedding" representation for each one of the possible token values

$$\mathbf{w}_{\text{emb}} = \begin{pmatrix} — & \mathbf{w}_{\text{emb}}(t=1) \sim \mathbb{R}^d & — \\ & \vdots & \\ — & \mathbf{w}_{\text{emb}}(t=n_{\mathcal{V}}) \sim \mathbb{R}^d & — \end{pmatrix} \sim \mathbb{R}^{n_{\mathcal{V}} \times d} \quad \text{where} \quad n_{\mathcal{V}} \equiv \begin{cases} n_{\text{vocab}} & \text{"token"} \\ n_{\text{context}} & \text{"positional"} \end{cases} \tag{1}$$

Although embeddings may initially be assigned random values, it is important to realize that they must remain trainable so that, once optimized, tokens acquire representations that reflect the structure and requirements of the task. We will see in the backward pass how the embeddings receive error signals from the loss function allowing them to converge to representations that effectively encode relevant information. In transformer-based deep learning architectures, it is the responsibility of mechanisms such as self-attention to learn inter-token dependencies as we will see in Section 3 and transform them into the error signals that reach the embedding layer.

Although token embeddings are usually learned via backpropagation, there does exist non-adjustable versions of positional embeddings that are designed to manually impose some structural constraints. This is the case, for example, with RoPE (Rotary Positional Embeddings) which aims to break the permutation equivariance property of non-causal attention by injecting information about the relative position of the tokens.

Forward pass. During the forward pass, the job of the embedding layer is simply to pull out the relevant embeddings associated with the $n_{\mathcal{T}}$ tokens present in the input sequence $\mathbf{a}_{i-1} \sim \mathbb{N}^{n_{\mathcal{T}}}$ from the embedding store $\mathbf{w}_{\text{emb}}$ defined in eq.(1). In order to formulate this database lookup as a differentiable vectorized operation, let us first transform $\mathbf{a}_{i-1}$ into its "One Hot Encoded" (OHE) representation.

As an example, let us go back to token $t^\star$ and expand its integer representation $\mathbf{a}_{i-1}(t=t^\star) \sim \mathbb{N}$ into a $n_{\mathcal{V}}$-dimensional binary sparse vector

$$\mathbf{a}_{i-1}(t=t^\star) \sim \mathbb{N} \quad \longrightarrow \quad \mathbf{a}_{i-1}(t=t^\star)_{\text{OHE}} = \left[a_{t^\star 1}, \cdots, a_{t^\star n_{\mathcal{V}}}\right] \sim \mathbb{R}^{n_{\mathcal{V}}} \quad \text{with} \quad a_{t^\star t'} = \delta_{t^\star t'}$$

This way, the vector $\mathbf{a}_{i-1}(t=t^\star)_{\text{OHE}} \sim \mathbb{R}^{n_{\mathcal{V}}}$ only has a single non-null component $a_{t^\star t'} = 1$ when $t' = t^\star$ and all other components are $a_{t^\star t'} = 0$ for $t' \neq t^\star$ across the whole range $t' \in [1, \cdots, n_{\mathcal{V}}]$. Because of this OHE representation, the product

$$\mathbf{a}_{i-1}(t=t^\star)_{\text{OHE}}\, \mathbf{w}_{\text{emb}} = \mathbf{w}_{\text{emb}}(t=t^\star) \sim \mathbb{R}^d$$

immediately picks up the correct embedding vector for token $t^\star$. Applying this OHE representation to all $n_{\mathcal{T}}$ tokens, the input data $\mathbf{a}_{i-1} \sim \mathbb{N}^{n_{\mathcal{T}}}$ can therefore be represented as a sparse array

$$\text{ohe}(\mathbf{a}_{i-1}) \equiv \begin{pmatrix} — & \mathbf{a}_0(t=1)_{\text{OHE}} \sim \mathbb{R}^{n_{\mathcal{V}}} & — \\ & \vdots & \\ — & \mathbf{a}_0(t=n_{\mathcal{T}})_{\text{OHE}} \sim \mathbb{R}^{n_{\mathcal{V}}} & — \end{pmatrix} \sim \mathbb{R}^{n_{\mathcal{T}} \times n_{\mathcal{V}}}$$

which can be multiplied by $\mathbf{w}_{\text{emb}} \sim \mathbb{R}^{n_{\mathcal{V}} \times d}$ to simultaneously pull out the relevant $n_{\mathcal{T}}$ embedding vectors associated with the specific $n_{\mathcal{T}}$ tokens present in the input sequence $\mathbf{a}_{i-1} \sim \mathbb{N}^{n_{\mathcal{T}}}$ and collect them into the output $\mathbf{a}_i \sim \mathbb{R}^{n_{\mathcal{T}} \times d}$ of the embedding layer

$$\mathbf{a}_i = \text{ohe}(\mathbf{a}_{i-1})\, \mathbf{w}_{\text{emb}} = \begin{pmatrix} — & \mathbf{w}_{\text{emb}}(t=1) \sim \mathbb{R}^d & — \\ & \vdots & \\ — & \mathbf{w}_{\text{emb}}(t=t_{n_{\mathcal{T}}}) \sim \mathbb{R}^d & — \end{pmatrix} \sim \mathbb{R}^{n_{\mathcal{T}} \times d}$$

In summary, the forward pass of an embedding layer results in $\mathbf{a}_i \sim \mathbb{R}^{n_{\mathcal{T}} \times d}$ where the rows of $\mathbf{a}_i$ contain the embeddings of the $n_{\mathcal{T}}$ tokens present in $\mathbf{a}_{i-1}$ by extracting them from the complete embedding store $\mathbf{w}_{\text{emb}}$ defined in eq.(1) via OHE matrix multiplication

Embedding layer: forward pass

$$\mathbf{a}_i = \text{ohe}(\mathbf{a}_{i-1})\, \mathbf{w}_{\text{emb}} \tag{2}$$

Backward pass. As usual, we need to evaluate the recursive backward error flow described in eq.(11) of the reference paper [1] with $\boldsymbol{\Delta}_i \sim \mathbf{a}_i \sim \mathbb{R}^{n_\mathcal{T}\times d}$ so that

$$\begin{aligned}\boldsymbol{\Delta}_i \cdot \mathrm{d}\mathbf{a}_i &= \boldsymbol{\Delta}_i \cdot \mathrm{d}\left(\mathrm{ohe}(\mathbf{a}_{i-1})\,\mathbf{w}_{\mathrm{emb}}\right)\\ &= \underbrace{\mathrm{ohe}(\mathbf{a}_{i-1})^t\,\boldsymbol{\Delta}_i}_{\frac{\partial\mathcal{L}_{\mathrm{seq}}}{\partial\mathbf{w}_{\mathrm{emb}}}}\cdot \mathrm{d}\mathbf{w}_{\mathrm{emb}} + \underbrace{\boldsymbol{\Delta}_i\,\mathbf{w}_{\mathrm{emb}}^t}_{\boldsymbol{\Delta}_{i-1}}\cdot\mathrm{d}\left(\mathrm{ohe}(\mathbf{a}_{i-1})\right)\end{aligned}$$

Normally, we consider the input data sequence $\mathbf{a}_{i-1} \equiv \mathbf{a}_0$ (see footnote) so that the backward error flow stops here with $\mathrm{d}\left(\mathrm{ohe}(\mathbf{a}_{i-1})\right) \sim \mathrm{d}\mathbf{a}_{i-1} = \mathbf{0}$ and therefore there is no need to consider $\boldsymbol{\Delta}_{i-1}$. Some scenarios where one may be interested in keeping this term involve adversarial attacks, feature attribution/interpretability... In summary, the backward pass through an embedding layer is given by

Embedding layer: backward pass

$$\frac{\partial\mathcal{L}_{\mathrm{seq}}}{\partial\mathbf{w}_{\mathrm{emb}}} = \mathrm{ohe}(\mathbf{a}_{i-1})^t\,\boldsymbol{\Delta}_i \quad \sim \mathbb{R}^{n_\mathcal{V}\times d} \tag{3}$$

3 Self-attention layer: Single head

Let us start by looking at a single head of self-attention. We will see in Section 4 how a complete self-attention layer combines together multiple heads and in Section 7 how self-attention layers themselves are composed with each other into transformer blocks.

3.1 Self-attention layer: Single head — Forward pass

Let us denote the input data by $\mathbf{a}_{i-1} \sim \mathbb{R}^{n_\mathcal{T}\times d}$ consisting of a sequence of $n_\mathcal{T}$ tokens which each, individually, have a d-dimensional feature map representation

$$\mathbf{a}_{i-1} = \left[\mathbf{a}_{i-1}(t=1)\sim\mathbb{R}^d, \cdots, \mathbf{a}_{i-1}(t=n_\mathcal{T})\sim\mathbb{R}^d\right] \sim \mathbb{R}^{n_\mathcal{T}\times d}$$

These feature maps may be their initial embedding representations or the output of previous transformer blocks. The purpose an attention head h is to learn a new d_h-dimensional representation for each one of the $n_\mathcal{T}$ tokens

$$\mathbf{a}_{i-1} \quad\longrightarrow\quad \mathbf{a}_i^h = \left[\mathbf{a}_i^h(t=1)\sim\mathbb{R}^{d_h}, \cdots, \mathbf{a}_i^h(t=n_\mathcal{T})\sim\mathbb{R}^{d_h}\right] \sim \mathbb{R}^{n_\mathcal{T}\times d_h}$$

The h superscript in $\mathbf{a}_i^h$ is here to denote that each attention head produces different output feature maps.

It is expected that all the tokens in $\mathbf{a}_{i-1}$ should be treated collectively as a coherent sequence rather than in isolation from each other. In other words, we would like to think of $\mathbf{a}_{i-1}$ as a one "sample" even though it is composed of multiple elements. This way, the output feature maps should be such that tokens get information from other tokens in the sequence. Stacking the sequence of tokens together, the self-attention head h is summarized as:

$$\mathbf{a}_{i-1}\sim\mathbb{R}^{n_\mathcal{T}\times d} \equiv \begin{pmatrix} — & \mathbf{a}_{i-1}(t=1)\sim\mathbb{R}^d & — \\ & \vdots & \\ — & \mathbf{a}_{i-1}(t=n_\mathcal{T})\sim\mathbb{R}^d & — \end{pmatrix}$$

$$\downarrow$$

Sequence modeling: one head h of self-attention

$$\downarrow$$

$$\mathbf{a}_i^h\sim\mathbb{R}^{n_\mathcal{T}\times d_h} \equiv \begin{pmatrix} — & \mathbf{a}_i^h(t=1)\sim\mathbb{R}^{d_h} & — \\ & \vdots & \\ — & \mathbf{a}_i^h(t=n_\mathcal{T})\sim\mathbb{R}^{d_h} & — \end{pmatrix}$$

One head h of self-attention

Layer	Forward pass	Shape	Backward pass
Input data	$\mathbf{a}_{i-1}$	$\mathbb{R}^{n_{\mathcal{T}} \times d}$	$\mathbf{\Delta}^h_{i-1} = \mathbf{\Delta}_{v_h} + \mathbf{\Delta}_{q_h} + \mathbf{\Delta}_{k_h}$
↓ *(input feature maps for the $n_{\mathcal{T}}$ tokens)* ↓		↓ ↑	↑ *(downstream error signal for head h)* ↑
Fully connected — "queries"	$\mathbf{q}_h = \mathbf{a}_{i-1}\, \mathbf{w}_{q_h} + \widetilde{\mathbf{b}_{q_h}}$	$\mathbb{R}^{n_{\mathcal{T}} \times d_\rho}$	$\mathbf{\Delta}_{q_h} = \mathbf{\Delta}^h_{\mathrm{raw}}\, \mathbf{k}_h \mathbf{w}^t_{q_h}$ *(terminal)*
Fully connected — "keys"	$\mathbf{k}_h = \mathbf{a}_{i-1}\, \mathbf{w}_{k_h} + \widetilde{\mathbf{b}_{k_h}}$	$\mathbb{R}^{n_{\mathcal{T}} \times d_\rho}$	$\mathbf{\Delta}_{k_h} = (\mathbf{\Delta}^h_{\mathrm{raw}})^t\, \mathbf{q}_h\, \mathbf{w}^t_{k_h}$ *(terminal)*
Raw attention weights	$\boldsymbol{\rho}^{\mathrm{raw}}_{(\mathbf{a}_{i-1},\, h)} = \mathbf{q}_h\, \mathbf{k}^t_h$	$\mathbb{R}^{n_{\mathcal{T}} \times n_{\mathcal{T}}}$	*($\mathbf{\Delta}^h_{raw}$ splits into queries and keys branches)*
Scaling	$\boldsymbol{\rho}^{\mathrm{scaled}}_{(\mathbf{a}_{i-1},\, h)} = \boldsymbol{\rho}^{\mathrm{raw}}_{(\mathbf{a}_{i-1},\, h)} \big/ \sqrt{d_\rho}$	$\mathbb{R}^{n_{\mathcal{T}} \times n_{\mathcal{T}}}$	$\mathbf{\Delta}^h_{\mathrm{raw}} = \mathbf{\Delta}^h_{\mathrm{scaled}} / \sqrt{d_\rho}$
Causal mask	$\boldsymbol{\rho}^{\mathrm{causal}}_{(\mathbf{a}_{i-1},\, h)} = \mathbf{m} \circ \boldsymbol{\rho}^{\mathrm{scaled}}_{(\mathbf{a}_{i-1},\, h)}$	$\mathbb{T}_L(\mathbb{R}^{n_{\mathcal{T}} \times n_{\mathcal{T}}})$	$\mathbf{\Delta}^h_{\mathrm{scaled}} = \mathbf{\Delta}^h_{\mathrm{causal}}$
Attention weights	$\boldsymbol{\rho}_{(\mathbf{a}_{i-1},\, h)} = \mathrm{softmax}\, \boldsymbol{\rho}^{\mathrm{causal}}_{(\mathbf{a}_{i-1},\, h)}$	$\mathbb{T}_L(\mathbb{R}^{n_{\mathcal{T}} \times n_{\mathcal{T}}})$	$\mathbf{\Delta}^h_{\mathrm{causal}} = \left[\mathbf{\Delta}^h_i \mathbf{v}^t_h - \widetilde{\left(\mathbf{\Delta}^h_i \mathbf{v}^t_h \right) \ominus} \boldsymbol{\rho}_{(\mathbf{a}_{i-1},\, h)} \right] \circ \boldsymbol{\rho}_{(\mathbf{a}_{i-1},\, h)}$
Fully connected — "values"	$\mathbf{v}_h = \mathbf{a}_{i-1} \mathbf{w}_{v_h} + \widetilde{\mathbf{b}_{v_h}}$	$\mathbb{R}^{n_{\mathcal{T}} \times d_h}$	$\mathbf{\Delta}_{v_h} = \boldsymbol{\rho}^t_{(\mathbf{a}_{i-1},\, h)} \mathbf{\Delta}^h_i \mathbf{w}^t_{v_h}$ *(terminal)*
↓		↓ ↑	*($\mathbf{\Delta}^h_i$ splits into attention and values branches)*
↓ *(transformed feature maps for the $n_{\mathcal{T}}$ tokens)* ↓		↓ ↑	↑ *(upstream error signal for head h)* ↑
Output data	$\mathbf{a}^h_i = \boldsymbol{\rho}_{(\mathbf{a}_{i-1},\, h)}\, \mathbf{v}_h$	$\mathbb{R}^{n_{\mathcal{T}} \times d_h}$	$\mathbf{\Delta}^h_i$

Table 1: Illustration of the data flow through an attention head. In practice it is common to choose the dimensionality d_ρ of the queries/keys feature maps to match the dimensionality of the values output feature maps with $d_\rho \equiv d_h$. As we discuss in the main part of the text, this is not a requirement and we keep it different here for the sake of generality. In the backward pass, we denote by "terminal" the branches where the upstream error signal $\mathbf{\Delta}^h_i$ assigned to attention head h reaches the input feature maps $\mathbf{a}_{i-1}$ of the $n_{\mathcal{T}}$ tokens. Those terminal error signals converge with each other at the input data layer and accumulate to produce the downstream error signal $\mathbf{\Delta}^h_{i-1}$.

The complete data flow through an attention head (with causal mask) is detailed in Table 1. In order to understand the logic and design choices, it is instructive to start from the desired output representation of the tokens and work backwards from there.

Starting from the end: What should a reasonable $\mathbf{a}_i^h$ look like? Fully connected layers are the "bread and butter" of deep learning architectures. To change the dimensionality of the token feature maps from the input d to the output d_h, it is tempting to start by passing the input data $\mathbf{a}_{i-1}$ through a fully connected layer parametrized by $\{\mathbf{w}_{v_h} \sim \mathbb{R}^{d\times d_h}, \mathbf{b}_{v_h} \sim \mathbb{R}^{d_h}\}$. The index h indicates that those parameters are specific to one attention head h. Therefore, we have

$$\mathbf{a}_{i-1} = \begin{pmatrix} \mathbf{a}_{i-1}(t=1) \sim \mathbb{R}^d \\ \vdots \\ \mathbf{a}_{i-1}(t=n_{\mathcal{T}}) \sim \mathbb{R}^d \end{pmatrix} \sim \mathbb{R}^{n_{\mathcal{T}}\times d} \Rightarrow \{\mathbf{w}_{v_h}, \mathbf{b}_{v_h}\} \Rightarrow \mathbf{v}_h = \begin{pmatrix} \mathbf{v}_h(t=1) \sim \mathbb{R}^{d_h} \\ \vdots \\ \mathbf{v}_h(t=n_{\mathcal{T}}) \sim \mathbb{R}^{d_h} \end{pmatrix} \sim \mathbb{R}^{n_{\mathcal{T}}\times d_h}$$

Values: forward pass

$$\mathbf{v}_h = \mathbf{a}_{i-1}\mathbf{w}_{v_h} + \widetilde{\mathbf{b}_{v_h}} \tag{4}$$

Unfortunately, the "values" $\mathbf{v}_h \sim \mathbb{R}^{n_{\mathcal{T}}\times d_h}$ produced by this operation are all independent from each other. In other words, $\mathbf{v}_h(t=t^\star)$ depends solely on input token $\mathbf{a}_{i-1}(t=t^\star)$ without mixing in any information from any of the other tokens. This is exactly the same as considering the $n_{\mathcal{T}}$ tokens as independent samples which contradicts our goal of treating the entire sequence $\mathbf{a}_{i-1}$ itself as a single coherent sample.

Let us now design a simple way to remedy this shortfall and start treating the tokens as context-aware sequential elements rather than a disorganized group of independent elements.

For clarity, we assume a causal relationship where a token can only be "aware" of the tokens that occur before it in the sequence and not those that follow it. In this case

- the first token does not have any context and therefore it is reasonable to define its output feature map $\mathbf{a}_i^h(t=1) \sim \mathbb{R}^{d_h}$ directly equal to its value feature map

  $$\mathbf{a}_i^h(t=1) \equiv \rho_{11}\,\mathbf{v}_h(t=1) \quad \text{with} \quad \rho_{11} = 1 \tag{5.a}$$

- the second token may take information from both the first token as well as from itself. Therefore, it is reasonable to assign its output representation $\mathbf{a}_i^h(t=2) \sim \mathbb{R}^{d_h}$ as a weighted average of the two available value feature maps

  $$\mathbf{a}_i^h(t=2) \equiv \rho_{21}\,\mathbf{v}_h(t=1) + \rho_{22}\,\mathbf{v}_h(t=2) \quad \text{with} \quad \rho_{21} + \rho_{22} = 1 \tag{5.b}$$

- similarly, the third token is expressed as a linear combination of the value feature maps from the first three tokens so that $\mathbf{a}_i^h(t=3) \sim \mathbb{R}^{d_h}$ is given by

  $$\mathbf{a}_i^h(t=3) \equiv \rho_{31}\,\mathbf{v}_h(t=1) + \rho_{32}\,\mathbf{v}_h(t=2) + \rho_{33}\,\mathbf{v}_h(t=3) \quad \text{with} \quad \rho_{31} + \rho_{32} + \rho_{33} = 1 \tag{5.c}$$

- finally, we reach the last token which has access to all of the tokens in the sequence. Therefore, the representation of the final token $\mathbf{a}_i^h(t=n_{\mathcal{T}}) \sim \mathbb{R}^{d_h}$ is written as

  $$\begin{aligned} &\mathbf{a}_i^h(t=n_{\mathcal{T}}) \equiv \rho_{n_{\mathcal{T}}1}\,\mathbf{v}_h(t=1) + \rho_{n_{\mathcal{T}}2}\,\mathbf{v}_h(t=2) + \rho_{n_{\mathcal{T}}3}\,\mathbf{v}_h(t=3) + \cdots + \rho_{n_{\mathcal{T}}n_{\mathcal{T}}}\,\mathbf{v}_h(t=n_{\mathcal{T}}) \\ &\text{with} \quad \rho_{n_{\mathcal{T}}1} + \rho_{n_{\mathcal{T}}2} + \rho_{n_{\mathcal{T}}3} + \cdots + \rho_{n_{\mathcal{T}}n_{\mathcal{T}}} = 1 \end{aligned} \tag{5.d}$$

Notice how we have introduced normalized "attention weights" $\rho_{\alpha\beta}$ as coefficients that define the weighted averages. These coefficients link together different pairs of tokens according to their relative relevance to each other. We will discuss later how one may go beyond these second-order (pairwise token-to-token) interactions and consider higher-level token interactions that span the entire sequence.

Crucially, we have not yet specified how these weights are determined: This will be the topic of the next paragraph. Nonetheless, before providing their exact expressions, we should already mention that we do not intend to freeze these coefficients to any permanent values. Instead, they should depend — in a learnable way — upon the specific tokens present in $\mathbf{a}_{i-1}$ and their pairwise relationships with one another. In other words, we expect the attention weights to be a function parametrized by a set of head h specific parameters

$$\rho_{\alpha\beta} \equiv \text{att}_h \left\langle \mathbf{a}_{i-1}(t = t_\alpha), \, \mathbf{a}_{i-1}(t = t_\beta) \right\rangle$$

This means that the values of $\rho_{\alpha\beta}$ will change, not only during training, as parameters associated with the head-specific attention function att_h are learned, but also during inference by dynamically assuming new values depending on the specific tokens $t = t_\alpha$ and $t = t_\beta$ in the input sample.

In other words, the output feature maps $\mathbf{a}_i^h$ are weighted averages of the values feature maps $\mathbf{v}_h$ where the attention coefficients $\rho_{\alpha\beta}$ are data-dependent on $\mathbf{a}_{i-1}$. Note that, at this point, we are considering only a single head h of self-attention.

Ultimately, we will consider multiple heads so that the self-attention function att_h will have different parameters for each head (although the overall form of the functional dependence of $\rho_{\alpha\beta}$ on $\mathbf{a}_{i-1}$ will remain the same).

Before moving on to specifying a functional form for the attention weights, let us organize them into a matrix representation $\boldsymbol{\rho}_{(\mathbf{a}_{i-1}, \, h)}$ where the subscript makes it clear that those coefficients should depend on the input sample $\mathbf{a}_{i-1}$ and on the specific self-attention head h . Since we have considered only pairwise $\rho_{\alpha\beta}$ connections between the tokens, the full attention weight matrix is square $\sim \mathbb{R}^{n_\mathcal{T} \times n_\mathcal{T}}$. In addition, we have also enforced causality so the coefficients must respect $\rho_{\alpha\beta} = 0$ if $\beta > \alpha$. In this case, $\boldsymbol{\rho}_{(\mathbf{a}_{i-1}, \, h)} \sim \mathbb{T}_L(\mathbb{R}^{n_\mathcal{T} \times n_\mathcal{T}})$ is a square lower triangular matrix [2]. Using the matrix representation of $\boldsymbol{\rho}_{(\mathbf{a}_{i-1}, \, h)}$, the weighted averages defining the output sequence $\mathbf{a}_i^h$ of a self-attention layer can be expressed as

$$\boxed{\mathbf{a}_i^h \equiv \boldsymbol{\rho}_{(\mathbf{a}_{i-1}, \, h)} \, \mathbf{v}_h} = \begin{pmatrix} \rho_{11} & 0 & 0 & \cdots & 0 \\ \rho_{21} & \rho_{22} & 0 & \cdots & 0 \\ \rho_{31} & \rho_{32} & \rho_{33} & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \rho_{n_\mathcal{T} 1} & \rho_{n_\mathcal{T} 2} & \rho_{n_\mathcal{T} 3} & \cdots & \rho_{n_\mathcal{T} n_\mathcal{T}} \end{pmatrix} \begin{pmatrix} — & \mathbf{v}_h(t=1) & — \\ — & \mathbf{v}_h(t=2) & — \\ — & \mathbf{v}_h(t=3) & — \\ & \vdots & \\ — & \mathbf{v}_h(t=n_\mathcal{T}) & — \end{pmatrix} \quad (6)$$

One can verify via straightforward expansion of eq.(6), that we exactly recover the expected expressions $\mathbf{a}_i^h(t=1), \cdots, \mathbf{a}_i^h(t=n_\mathcal{T})$ for the causal weighted averages given by eqs.(5.a)–(5.d), see eq.(48) for an explicit derivation.

Defining the attention weights. Having worked our way backwards starting with the desired output representation $\mathbf{a}_i^h$, we are now in a position to define the attention function att_h at the heart of the attention weight matrix $\boldsymbol{\rho}_{(\mathbf{a}_{i-1}, h)}$.

Let us consider two tokens t_α and t_β and their feature maps $\mathbf{a}_{i-1}(t = t_\alpha) \sim \mathbb{R}^d$ and $\mathbf{a}_{i-1}(t = t_\beta) \sim \mathbb{R}^d$ from the input sequence $\mathbf{a}_{i-1}$. Ideally, we would like their pairwise attention weight $\rho_{\alpha\beta} \sim \mathbb{R}$ to quantify the level of "relevance" these tokens have for each other. How to define a good expression for $\rho_{\alpha\beta}$?

$$\rho_{\alpha\beta} = \text{att}_h \left\langle \mathbf{a}_{i-1}(t = t_\alpha), \, \mathbf{a}_{i-1}(t = t_\beta) \right\rangle = \, ?$$

Rather than diving directly into the complete formulation, let us explore various possible implementations, examine their limitations and gradually build a deeper understanding for the definition of proper attention weights.

[2] This is a common situation for generative language models where a strict left-to-right order needs to be enforced such as, for example, in autoregressive text generation. In contrast, architectures using encoders designed for language translation generally do not use causal masks so they are free to capture the overall relationships between words. In this case, the attention weight matrix may also have a non-zero upper triangular part $\boldsymbol{\rho}_{(\mathbf{a}_{i-1}, \, h)} \sim \mathbb{R}^{n_\mathcal{T} \times n_\mathcal{T}}$. We will see this when we provide a general definition for attention weights.

1 As a first attempt, one might be tempted to define the attention weights as a direct vector dot-product $\rho_{\alpha\beta} \overset{?}{\equiv} \mathbf{a}_{i-1}(t = t_\alpha) \cdot \mathbf{a}_{i-1}(t = t_\beta)$. Although this definition ensures that $\rho_{\alpha\beta} \sim 1$ when the input feature maps are aligned with each other (and $\rho_{\alpha\beta} \sim 0$ when they are orthogonal to each other), one issue with this choice is that the resulting attention weights would be fixed by the initial feature maps without the possibility of learning from data. (Those initial feature maps would most likely even be random or determined by pretrained embeddings.)

2 In our second attempt, we solve this problem of static attention weights by introducing a fully-connected layer with adjustable parameters $\{\mathbf{w}_{q_h} \sim \mathbb{R}^{d \times d_\rho}, \mathbf{b}_{q_h} \sim \mathbb{R}^{d_\rho}\}$. The input token feature maps are transformed into so-called "query" feature maps in d_ρ dimensions

$$\big(\mathbf{a}_{i-1}(t = t_\alpha) \sim \mathbb{R}^d \, , \, \mathbf{a}_{i-1}(t = t_\beta) \sim \mathbb{R}^d \big) \Rightarrow \{\mathbf{w}_{q_h}, \mathbf{b}_{q_h}\} \Rightarrow \big(\mathbf{q}_h(t = t_\alpha) \sim \mathbb{R}^{d_\rho} \, , \, \mathbf{q}_h(t = t_\beta) \sim \mathbb{R}^{d_\rho}\big)$$

We can now try to define the attention weights as the dot product $\rho_{\alpha\beta} \overset{?}{\equiv} \mathbf{q}_h(t = t_\alpha) \cdot \mathbf{q}_h(t = t_\beta)$. Using this revised tentative dot-product, the attention weights would be free to evolve as the parameters $\mathbf{w}_{q_h}$ and $\mathbf{b}_{q_h}$ are updated during training. However, this definition of attention weights would still be rather restrictive as it enforces an undesired symmetric relationship between the tokens since $\rho_{\alpha\beta} = \rho_{\beta\alpha}$. Ideally, we would like to define attention weights in a way that takes into account the potentially asymmetric nature of token relationships [3].

3 For our third attempt, we look for a definition of token-to-token attention that goes beyond simple symmetric similarity and that, instead, allows $\rho_{\alpha\beta} \neq \rho_{\beta\alpha}$. This can be achieved by introducing an additional fully-connected layer parametrized by $\{\mathbf{w}_{k_h} \sim \mathbb{R}^{d \times d_\rho}, \mathbf{b}_{k_h} \sim \mathbb{R}^{d_\rho}\}$ of matching dimensionality to $\{\mathbf{w}_{q_h} \sim \mathbb{R}^{d \times d_\rho}, \mathbf{b}_{q_h} \sim \mathbb{R}^{d_\rho}\}$ which was introduced in the previous attempt. Each token would now be associated with two different and independent representations, so-called "queries" and "keys":

$$\begin{tikzcd}
 & \{\mathbf{w}_{q_h}, \mathbf{b}_{q_h}\} \arrow[r, Rightarrow] & \big(\mathbf{q}_h(t = t_\alpha) \sim \mathbb{R}^{d_\rho} \, , \mathbf{q}_h(t = t_\beta) \sim \mathbb{R}^{d_\rho}\big) \\
\big(\mathbf{a}_{i-1}(t = t_\alpha) \sim \mathbb{R}^d \, , \, \mathbf{a}_{i-1}(t = t_\beta) \sim \mathbb{R}^d \big) \arrow[ur, Rightarrow] \arrow[dr, Rightarrow] & & \\
 & \{\mathbf{w}_{k_h}, \mathbf{b}_{k_h}\} \arrow[r, Rightarrow] & \big(\mathbf{k}_h(t = t_\alpha) \sim \mathbb{R}^{d_\rho} \, , \mathbf{k}_h(t = t_\beta) \sim \mathbb{R}^{d_\rho}\big)
\end{tikzcd}$$

This allows us to define the attention weights between two tokens t_α and t_β as the dot-product

$$\rho_{\alpha\beta} = \mathbf{q}_h(t = t_\alpha) \cdot \mathbf{k}_h(t = t_\beta) \sim \mathbb{R} \tag{7}$$

Since the parameters of the queries and keys are different from each other $\{\mathbf{w}_{q_h}, \mathbf{b}_{q_h}\} \neq \{\mathbf{w}_{k_h}, \mathbf{b}_{k_h}\}$, we now have broken the symmetry and attention weights are such that $\rho_{\alpha\beta} \neq \rho_{\beta\alpha}$.

It is this expression for quantifying the degree of pairwise attention $\rho_{\alpha\beta}$ between two tokens that has emerged as the preferred candidate for self-attention. Other, even more general, formulations may also be proposed (such as multiplicative bilinear attention, see [4] for an easy informal review) but the dot-product presented in eq.(7) remains a favorite among practitioners.

Now that we have agreed upon a formulation using eq.(7) for attention weights, we need to evaluate $\rho_{\alpha\beta}$ for all possible pairwise token-to-token combinations in $\mathbf{a}_{i-1}$. Therefore, going beyond just two tokens, the complete set of $n_\mathcal{T}^2$ attention weights requires first the creation of two independent linear transformations of the input tokens $\mathbf{a}_{i-1}$ into queries $\mathbf{q}_h$ and keys $\mathbf{k}_h$ using two fully-connected layers:

$$\begin{tikzcd}
 & \{\mathbf{w}_{q_h}, \mathbf{b}_{q_h}\} \arrow[r, Rightarrow] & \mathbf{q}_h = \begin{pmatrix} \mathbf{q}_h(t = 1) \sim \mathbb{R}^{d_\rho} \\ \vdots \\ \mathbf{q}_h(t = n_\mathcal{T}) \sim \mathbb{R}^{d_\rho} \end{pmatrix} \sim \mathbb{R}^{n_\mathcal{T} \times d_\rho} \quad \text{(queries)} \\
\mathbf{a}_{i-1} = \begin{pmatrix} \mathbf{a}_{i-1}(t = 1) \sim \mathbb{R}^d \\ \vdots \\ \mathbf{a}_{i-1}(t = n_\mathcal{T}) \sim \mathbb{R}^d \end{pmatrix} \sim \mathbb{R}^{n_\mathcal{T} \times d} \arrow[ur, Rightarrow] \arrow[dr, Rightarrow] & & \text{where } \mathbf{w}_{q_h} \sim \mathbf{w}_{k_h} \sim \mathbb{R}^{d \times d_\rho} \text{ and } \mathbf{b}_{q_h} \sim \mathbf{b}_{k_h} \sim \mathbb{R}^{d_\rho} \quad (8) \\
 & \{\mathbf{w}_{k_h}, \mathbf{b}_{k_h}\} \arrow[r, Rightarrow] & \mathbf{k}_h = \begin{pmatrix} \mathbf{k}_h(t = 1) \sim \mathbb{R}^{d_\rho} \\ \vdots \\ \mathbf{k}_h(t = n_\mathcal{T}) \sim \mathbb{R}^{d_\rho} \end{pmatrix} \sim \mathbb{R}^{n_\mathcal{T} \times d_\rho} \quad \text{(keys)}
\end{tikzcd}$$

Queries and Keys: forward pass

$$\mathbf{q}_h = \mathbf{a}_{i-1}\mathbf{w}_{q_h} + \widetilde{\mathbf{b}_{q_h}} \tag{9.a}$$

$$\mathbf{k}_h = \mathbf{a}_{i-1}\mathbf{w}_{k_h} + \widetilde{\mathbf{b}_{k_h}} \tag{9.b}$$

Note that, until now, each token in the input sequence has been processed independently by the fully connected layers. This means that the feature maps in $\mathbf{q}_h$ and $\mathbf{k}_h$ still do not communicate or share information with each other, even within the queries or keys themselves, as the processing remains independent across tokens. It is only once we perform the pairwise token-to-token vector dot-products $\mathbf{q}_h(t=t_\alpha)\cdot\mathbf{k}_h(t=t_\beta)$ between the tokens' query/key representations that their affinity/relationship with each other is materialized via their attention weights.

For the sake of clarity, we are now adding a "raw" label $\rho^{\text{raw}}_{\alpha\beta} \longleftarrow \rho_{\alpha\beta}$ to the attention weights defined in eq.(7) to indicate that these weights are still left as straightforward dot-products; Normalization will be the topic of the next paragraph. The complete set of raw attention weights are then collected together into a square attention weight matrix $\boldsymbol{\rho}^{\text{raw}}_{(\mathbf{a}_{i-1},\,h)} \sim \mathbb{R}^{n_\mathcal{T}\times n_\mathcal{T}}$ defined as

$$\boldsymbol{\rho}^{\text{raw}}_{(\mathbf{a}_{i-1},\,h)} \equiv \begin{pmatrix} \mathbf{q}_h(t=1)\cdot\mathbf{k}_h(t=1) & \mathbf{q}_h(t=1)\cdot\mathbf{k}_h(t=2) & \mathbf{q}_h(t=1)\cdot\mathbf{k}_h(t=3) & \cdots & \mathbf{q}_h(t=1)\cdot\mathbf{k}_h(t=n_\mathcal{T}) \\ \mathbf{q}_h(t=2)\cdot\mathbf{k}_h(t=1) & \mathbf{q}_h(t=2)\cdot\mathbf{k}_h(t=2) & \mathbf{q}_h(t=2)\cdot\mathbf{k}_h(t=3) & \cdots & \mathbf{q}_h(t=2)\cdot\mathbf{k}_h(t=n_\mathcal{T}) \\ \mathbf{q}_h(t=3)\cdot\mathbf{k}_h(t=1) & \mathbf{q}_h(t=3)\cdot\mathbf{k}_h(t=2) & \mathbf{q}_h(t=3)\cdot\mathbf{k}_h(t=3) & \cdots & \mathbf{q}_h(t=3)\cdot\mathbf{k}_h(t=n_\mathcal{T}) \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \mathbf{q}_h(t=n_\mathcal{T})\cdot\mathbf{k}_h(t=1) & \mathbf{q}_h(t=n_\mathcal{T})\cdot\mathbf{k}_h(t=2) & \mathbf{q}_h(t=n_\mathcal{T})\cdot\mathbf{k}_h(t=3) & \cdots & \mathbf{q}_h(t=n_\mathcal{T})\cdot\mathbf{k}_h(t=n_\mathcal{T}) \end{pmatrix}$$

To conclude, the raw (unnormalized) attention weights $\boldsymbol{\rho}^{\text{raw}}_{(\mathbf{a}_{i-1},\,h)}$ are expressed as a matrix product between the queries and the keys

$$\mathbf{k}^t_h \sim \mathbb{R}^{d_\rho\times n_\mathcal{T}} \equiv \begin{pmatrix} | & | & | & & | \\ \mathbf{k}_h(t=1) & \mathbf{k}_h(t=2) & \mathbf{k}_h(t=3) & \ldots & \mathbf{k}_h(t=n_\mathcal{T}) \\ | & | & | & & | \end{pmatrix}$$

$$\mathbf{q}_h \sim \mathbb{R}^{n_\mathcal{T}\times d_\rho} \equiv \begin{pmatrix} — & \mathbf{q}_h(t=1) & — \\ — & \mathbf{q}_h(t=2) & — \\ — & \mathbf{q}_h(t=3) & — \\ & \vdots & \\ — & \mathbf{q}_h(t=n_\mathcal{T}) & — \end{pmatrix} \qquad \Longrightarrow \qquad \boldsymbol{\rho}^{\text{raw}}_{(\mathbf{a}_{i-1},\,h)} = \mathbf{q}_h\,\mathbf{k}^t_h \sim \mathbb{R}^{n_\mathcal{T}\times n_\mathcal{T}}$$

Raw attention weights: forward pass

$$\boldsymbol{\rho}^{\text{raw}}_{(\mathbf{a}_{i-1},\,h)} = \mathbf{q}_h\,\mathbf{k}^t_h \tag{10}$$

A few observations before we move on the final steps

- The construction of the self-attention matrix $\boldsymbol{\rho}^{\text{raw}}_{(\mathbf{a}_{i-1},\,h)}$ is the only place (with the exception of a simpler softmax normalization that will be described the next paragraph) where the inter-token relationships is actually exploited via the pairwise attention weights. All other computations, such as queries, keys and values in the self-attention layer operate on tokens as independent entities. By definition, each row of $\boldsymbol{\rho}^{\text{raw}}_{(\mathbf{a}_{i-1},\,h)} \sim \mathbb{R}^{n_\mathcal{T}\times n_\mathcal{T}}$ consists of a vector

  $$\boldsymbol{\rho}^{\text{raw}}_{(\mathbf{a}_{i-1},\,h)}(t=t^\star) = \left[\rho^{\text{raw}}_{t^\star 1}, \cdots, \rho^{\text{raw}}_{t^\star n_\mathcal{T}}\right] \sim \mathbb{R}^{n_\mathcal{T}} \tag{11}$$

  that contains the $n_\mathcal{T}$ attention weights of a specific token $t^\star$ with all the other tokens in the sequence. Choosing rows for this definition of attention weight vectors allows us to connect back with the expressions of the desired causal output in eq.(6) and this choice is standard convention in the literature.

- Although this definition of attention weights with eq.(7) does not show an explicit dependence of $\boldsymbol{\rho}^{\text{raw}}_{(\mathbf{a}_{i-1},\, h)}$ on the input data $\mathbf{a}_{i-1}$ and specific head h, these dependencies are implicit through the way that queries $\mathbf{q}_h$ are keys $\mathbf{k}_h$ are built via fully-connected layers applied to $\mathbf{a}_{i-1}$ using head-specific parameters $\{\mathbf{w}_{q_h}, \mathbf{b}_{q_h}\}$ and $\{\mathbf{w}_{k_h}, \mathbf{b}_{k_h}\}$.

- These attention weights based on token-to-token dot-product means that a single layer of self-attention can only model up to two-token relationships. We will discuss at the end of this Section how composing together multiple layers of self-attention allows one to model higher-level relationships between the tokens.

- A common choice, for practical convenience and optimization benefits, is to choose $d_\rho = d_h$ so that the dimensionality of the feature maps for the queries $\mathbf{q}_h$, keys $\mathbf{k}_h$ and values $\mathbf{v}_h$ are all the same. This decision is widely applied in general-purpose deep learning libraries. Nonetheless, it is important to realize that this alignment of dimensions is incidental rather than a fundamental restriction. In fact, it is even possible to have different dimensionality for different attention heads, i.e. $d_\rho = d_\rho(h)$ since all heads are independent of each other. The only requirement is that queries and keys within the same head have the same dimensionality so that we can carry out their vector dot-product to define attention weights. Regardless of the choice for $d_\rho(h)$, in the end, the complete attention matrix will always be square $\boldsymbol{\rho}^{\text{raw}}_{(\mathbf{a}_{i-1},\, h)} \sim \mathbb{R}^{n_\mathcal{T} \times n_\mathcal{T}}$ with scalar components $\rho^{\text{raw}}_{\alpha\beta} \sim \mathbb{R}$.

Final enhancements. We have now gone over the heart of self-attention and how to define the attention matrix $\boldsymbol{\rho}^{\text{raw}}_{(\mathbf{a}_{i-1},\, h)}$. The only few steps left are some small enhancements to turn these raw attention weights into a more appropriate version $\boldsymbol{\rho}_{(\mathbf{a}_{i-1},\, h)}$ which we will use as our final attention weight matrix to produce the weighted averages of the value feature maps (where we started from in the first paragraph).

First, the raw attention weights $\rho^{\text{raw}}_{\alpha\beta}$ are defined as a dot-product between two d_ρ-dimensional vectors. If we assume that the components of these keys and queries feature map values are distributed according to a normal distribution (i.e. $\mathbf{q}_h(t = t_\alpha) \sim [\mathcal{N}(0,1), \cdots, \mathcal{N}(0,1)]$ and $\mathbf{k}_h(t = t_\beta)) \sim [\mathcal{N}(0,1), \cdots, \mathcal{N}(0,1)]$) then, the expected mean of their product $\rho^{\text{raw}}_{\alpha\beta}$ should be $\langle \mu(\rho^{\text{raw}}_{\alpha\beta}) \rangle = 0$ and their expected variance is $\langle \sigma^2(\rho^{\text{raw}}_{\alpha\beta}) \rangle = d_\rho$. To remove the dependence of the statistics of attention weights on the dimensionality d_ρ of the queries/keys feature maps (since d_ρ is an internal detail of the self-attention mechanism which does not appear in either $\mathbf{a}_{i-1}$ or $\mathbf{a}^h_i$), we rescale the attention weights $\rho^{\text{scaled}}_{\alpha\beta} = \rho^{\text{raw}}_{\alpha\beta} / \sqrt{d_\rho}$ so that we now have $\langle \sigma^2(\rho^{\text{scaled}}_{\alpha\beta}) \rangle = 1$. In summary, we introduce a scaled version of the attention weights

$$\boldsymbol{\rho}^{\text{scaled}}_{(\mathbf{a}_{i-1},\, h)} = \boldsymbol{\rho}^{\text{raw}}_{(\mathbf{a}_{i-1},\, h)} / \sqrt{d_\rho} \tag{12}$$

Second, for the sake of this paper, we decided to focus on causal models where a strict left-to-right order must be enforced. This is implemented by introducing a masking matrix $\mathbf{m} \sim \mathbb{R}^{n_\mathcal{T} \times n_\mathcal{T}}$ of the same dimensionality as the attention matrix and populated by binary 1/0 components in the lower triangular part so that $\mathbf{m} \sim \mathbb{T}_L(\mathbb{R}^{n_\mathcal{T} \times n_\mathcal{T}})$ and $\rho^{\text{scaled}}_{\alpha\beta} = 0$ if $\beta > \alpha$. The causal attention weights are therefore given by

$$\boldsymbol{\rho}^{\text{causal}}_{(\mathbf{a}_{i-1},\, h)} = \mathbf{m} \circ \boldsymbol{\rho}^{\text{scaled}}_{(\mathbf{a}_{i-1},\, h)} \tag{13}$$

which is expressed more explicitly as

$$\begin{pmatrix} \rho_{11} & 0 & 0 & \cdots & 0 \\ \rho_{21} & \rho_{22} & 0 & \cdots & 0 \\ \rho_{31} & \rho_{32} & \rho_{33} & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \rho_{n_\mathcal{T}1} & \rho_{n_\mathcal{T}2} & \rho_{n_\mathcal{T}3} & \cdots & \rho_{n_\mathcal{T}n_\mathcal{T}} \end{pmatrix}_{\text{causal}} = \begin{pmatrix} 1 & 0 & 0 & \cdots & 0 \\ 1 & 1 & 0 & \cdots & 0 \\ 1 & 1 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & 1 & 1 & \cdots & 1 \end{pmatrix} \circ \begin{pmatrix} \rho_{11} & \rho_{12} & \rho_{13} & \cdots & \rho_{1n_\mathcal{T}} \\ \rho_{21} & \rho_{22} & \rho_{23} & \cdots & \rho_{2n_\mathcal{T}} \\ \rho_{31} & \rho_{32} & \rho_{33} & \cdots & \rho_{3n_\mathcal{T}} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \rho_{n_\mathcal{T}1} & \rho_{n_\mathcal{T}2} & \rho_{n_\mathcal{T}3} & \cdots & \rho_{n_\mathcal{T}n_\mathcal{T}} \end{pmatrix}_{\text{scaled}}$$

where $\circ$ stands for the Hadamard product and we see that we recover an attention matrix with the same causal shape the one as described in eq.(6).

Finally, we ensure a proper normalization of the attention weights to close the loop and completely recover the desired output weighted averages initially discussed in eqs.(5.a)–(5.d) (i.e., not just the same causal shape but also the normalization constraint discussed there). As discussed previously, each row of the attention matrix $\sim \mathcal{R}^{n_\mathcal{T} \times n_\mathcal{T}}$ contains the $n_\mathcal{T}$ attention vectors $\sim \mathcal{R}^{n_\mathcal{T}}$ for all the tokens in the sequence. By analogy with eq.(11), let us consider a specific token $t^\star$ and its causal attention weight vector

$$\boldsymbol{\rho}^{\text{causal}}_{(\mathbf{a}_{i-1},\, h)}(t = t^\star) = \left[\rho^{\text{causal}}_{t^\star 1}, \cdots, \rho^{\text{causal}}_{t^\star n_\mathcal{T}}\right] \sim \mathbb{R}^{n_\mathcal{T}}$$

whose components quantify the attention scores between token $t^\star$ and all the other $n_\mathcal{T}$ tokens in the sequence. Causality means that, depending on the position of $t^\star \in [1, \cdots, n_\mathcal{T}]$ in the sequence, some of its attention scores would be identically null as prescribed by the mask $\mathbf{m}$ above. Applying a softmax function to this causal attention weight vector produces a probability distribution

$$\boldsymbol{\rho}_{(\mathbf{a}_{i-1},\, h)}(t = t^\star) = \left[\rho_{t^\star 1}, \cdots, \rho_{t^\star n_\mathcal{T}}\right] \equiv \text{softmax}\, \boldsymbol{\rho}^{\text{causal}}_{(\mathbf{a}_{i-1},\, h)}(t = t^\star) \sim \mathbb{R}^{n_\mathcal{T}}$$

where the components are normalized such that

$$\sum \boldsymbol{\rho}_{(\mathbf{a}_{i-1},\, h)}(t = t^\star) = \sum_{t'=1}^{n_\mathcal{T}} \rho_{t^\star t'} = 1 \tag{14}$$

Repeating the same softmax normalization to all the causal attention weight vectors, i.e. the rows of $\boldsymbol{\rho}^{\text{causal}}_{(\mathbf{a}_{i-1},\, h)}$, leads to the final expression for the self-attention matrix $\boldsymbol{\rho}_{(\mathbf{a}_{i-1},\, h)} \sim \mathbb{R}^{n_\mathcal{T} \times n_\mathcal{T}}$

$$\begin{pmatrix} \text{—}\ \boldsymbol{\rho}_{(\mathbf{a}_{i-1},\, h)}(t = 1)\ \text{—} \\ \vdots \\ \text{—}\ \boldsymbol{\rho}_{(\mathbf{a}_{i-1},\, h)}(t = n_\mathcal{T})\ \text{—} \end{pmatrix} \equiv \begin{pmatrix} \text{softmax}\, \boldsymbol{\rho}^{\text{causal}}_{(\mathbf{a}_{i-1},\, h)}(t = 1) \\ \vdots \\ \text{softmax}\, \boldsymbol{\rho}^{\text{causal}}_{(\mathbf{a}_{i-1},\, h)}(t = n_\mathcal{T}) \end{pmatrix} \Longrightarrow \boxed{\boldsymbol{\rho}_{(\mathbf{a}_{i-1},\, h)} = \text{softmax}\, \boldsymbol{\rho}^{\text{causal}}_{(\mathbf{a}_{i-1},\, h)}} \tag{15}$$

In addition to producing normalized probability distributions (offering direct interpretable attention allocation), softmax normalization is known to also be associated with beneficial implicit regularization mechanisms [5].

Summary and some remarks. We can now present the output representation

$$\mathbf{a}^h_i = \boldsymbol{\rho}_{(\mathbf{a}_{i-1},\, h)}\, \mathbf{v}_h \sim \mathbb{R}^{n_\mathcal{T} \times d_h} \tag{16}$$

initially proposed in eq.(6) where each step in the construction has been explained. (As a special case, if we restrict the attention weight matrix to be the identity matrix, then the output of the self-attention head reduces to the values feature maps with $\mathbf{a}^h_i \equiv \mathbf{v}^h$; This makes sense as tokens only attend to themselves with such attention weights.)

In summary, an attention head h can be seen as a function parametrized by $\mathcal{P}_h$ that takes in as input arguments the d-dimensional feature maps of the $n_\mathcal{T}$ tokens $\mathbf{a}_{i-1} \sim \mathbb{R}^{n_\mathcal{T} \times d}$ and returns their transformed d_h-dimensional representations $\mathbf{a}^h_i \sim \mathbb{R}^{n_\mathcal{T} \times d_h}$ with the following signature

$$\text{Att}_{\mathcal{P}_h} : \mathbf{a}_{i-1} \sim \mathbb{R}^{n_\mathcal{T} \times d} \longrightarrow \mathbf{a}^h_i \sim \mathbb{R}^{n_\mathcal{T} \times d_h} \quad \text{with } \mathcal{P}_h \equiv \begin{cases} \mathbf{w}_{q_h} \sim \mathcal{R}^{d \times d_\rho} & ; \ \mathbf{b}_{q_h} \sim \mathcal{R}^{d_\rho} \\ \mathbf{w}_{k_h} \sim \mathcal{R}^{d \times d_\rho} & \\ \mathbf{w}_{v_h} \sim \mathcal{R}^{d_h} & ; \ \mathbf{b}_{v_h} \sim \mathcal{R}^{d_h} \end{cases} \tag{17}$$

where we intentionally ignored the biases $\mathbf{b}_{k_h}$ of the keys (we will see below that self-attention does not depend on those) and where the exact expression for $\mathbf{a}^h_i = \text{Att}_{\mathcal{P}_h}(\mathbf{a}_{i-1})$ is given by

Self-attention: forward pass

$$\mathbf{a}^h_i = \text{softmax}\left(\mathbf{m} \circ \frac{\mathbf{q}_h\, \mathbf{k}^t_h}{\sqrt{d_\rho}}\right) \mathbf{v}_h \tag{18}$$

where the queries and keys $\mathbf{q}_h \sim \mathbf{k}_h \sim \mathbb{R}^{n_\mathcal{T} \times d_\rho}$ depend on $\mathbf{a}_{i-1}$ via the fully-connected layers expressed in eqs.(9.a)-(9.b) and the values $\mathbf{v}_h \sim \mathbb{R}^{n_\mathcal{T} \times d_h}$ also depend on $\mathbf{a}_{i-1}$ via another fully-connected layer given by eq.(4).

Before moving on to the backward pass, let us make a few general observations about self-attention:

Computational complexity. By virtue of its own definition as pairwise dot-products between tokens, the attention matrix $\boldsymbol{\rho}_{(\mathbf{a}_{i-1},\, h)} \sim \mathbf{q}_h\, \mathbf{k}_h^t \sim \mathbb{R}^{n_\mathcal{T} \times n_\mathcal{T}}$ requires quadratic complexity with respect to the number $n_\mathcal{T}$ of input tokens. This can be seen by looking at the computational complexity of the matrix product $\mathcal{O}(\mathbf{q}_h\, \mathbf{k}_h^t) \sim d_\rho n_\mathcal{T}^2$. This quadratic scaling with the number of tokens appears also when evaluating the linear mixing of values feature maps with $\mathcal{O}(\boldsymbol{\rho}_{(\mathbf{a}_{i-1},\, h)}\, \mathbf{v}_h) \sim d_h n_\mathcal{T}^2$. Overall, this confirms that self-attention has a computational complexity with grows quadratically with the number of tokens $\mathcal{O}(\mathbf{a}_i^h) \sim n_\mathcal{T}^2$. This potential bottleneck has been discussed extensively in the literature and we refer the readers to external references and their citations for reviews of computational complexity [6], approximation methods [7, 8] and low-level optimization [9]. We also discuss in a small side-note here the technique of KV cache optimization.

Adjustable bias parameters. Practitioners rarely include bias terms such as $\mathbf{b}_{q_h} \sim \mathbf{b}_{k_h} \sim \mathbb{R}^{d_\rho}$ and $\mathbf{b}_{v_h} \sim \mathbb{R}^{d_h}$ in self-attention layers: The original paper ignored them altogether [10]. In fact, one can show that, because of the shift-invariance property of the softmax normalization, the self-attention layer as defined in eq.(18) does not even depend on the bias term $\mathbf{b}_{k_h}$ of the keys [3]. This means that the d_ρ parameters of $\mathbf{b}_{k_h}$ are "impotent", in the sense that have no influence over the output of self-attention and therefore over the loss function itself (we will even confirm in the backward pass section that the derivative of the loss with respect to $\mathbf{b}_{k_h}$ is indeed identically null showing that those parameters cannot ever learn anything). Therefore, the bias term $\mathbf{b}_{k_h}$ can be removed without any loss of generality. Even though, this is not the case for the other bias terms $\mathbf{b}_{q_h}$ and $\mathbf{b}_{v_h}$ which do have valid contributions to self-attention, they are still frequently ignored by practitioners.

Composition of multiple layers of self-attention. Finally, let us discuss how one may go beyond pairwise token interactions by composing together multiple layers of self-attention. For the sake of simplicity, let us take $d_h = d$ so that the input $\mathbf{a}_{i-1}$ and output $\mathbf{a}_i^h$ of a self-attention head have the same dimensionality $\mathbf{a}_i^h \equiv \mathbf{a}_i \sim \mathbf{a}_{i-1} \sim \mathbb{R}^{n_\mathcal{T} \times d}$ (dropping the superscript h from $\mathbf{a}_i^h$ and from $\mathcal{P}_h \equiv \mathcal{P}$ to lighten the notation). Thanks to this dimensionality matching, one may compose together multiple attention heads, i.e. use the output representation as a new input. We will see in Section 4 how the same dimensionality matching may be achieved even when $d_h \neq d$ by combining together multiple attention heads into a multi-headed attention layer. In any case, the observation we make here remains conceptually identical for multi-headed attention albeit at the cost of tedious but superficial modifications (for example the superscript h would need to be kept to distinguish between different heads). Starting with $\mathbf{a}_{i-1}$ and applying the attention head twice takes us through a series of token feature maps going from $\mathbf{a}_{i-1}$ on to $\mathbf{a}_i = \text{Att}_\mathcal{P}(\mathbf{a}_{i-1})$ finishing with $\mathbf{a}_{i+1} = \text{Att}_\mathcal{P}(\mathbf{a}_i)$ summarized as

$$\mathbf{a}_{i+1} \sim \mathbb{R}^{n_\mathcal{T} \times d} = \text{Att}_\mathcal{P}\left(\text{Att}_\mathcal{P}\left(\mathbf{a}_{i-1} \sim \mathbb{R}^{n_\mathcal{T} \times d}\right) \sim \mathbb{R}^{n_\mathcal{T} \times d}\right) \sim \mathbb{R}^{n_\mathcal{T} \times d}$$

Let us consider the final $\mathbf{a}_{i+1} \sim \mathbb{R}^{n_\mathcal{T} \times d}$ and focus on the feature vector $\mathbf{a}_{i+1}(t = t^\star) \sim \mathbb{R}^d$ of a specific token $t^\star$. Its representation as a weighted sum over all of the $n_\mathcal{T}$ values' feature vectors in the sequence

[3] This can be seen by explicitly expanding out the expressions of the queries $\mathbf{q}_h$ and keys $\mathbf{k}_h$ which define the normalized attention weights $\boldsymbol{\rho}_{(\mathbf{a}_{i-1},\, h)}$ in eq.(15)

$$\begin{aligned}
\boldsymbol{\rho}_{(\mathbf{a}_{i-1},\, h)} \sim \text{softmax}\left(\mathbf{q}_h\, \mathbf{k}_h^t\right) &= \text{softmax}\left(\left(\mathbf{a}_{i-1}\mathbf{w}_{q_h} + \widetilde{\mathbf{b}}_{q_h}\right)\left(\mathbf{a}_{i-1}\mathbf{w}_{k_h} + \widetilde{\mathbf{b}}_{k_h}\right)^t\right) \\
&= \text{softmax}\left(\left(\mathbf{a}_{i-1}\mathbf{w}_{q_h} + \widetilde{\mathbf{b}}_{q_h}\right)\left(\mathbf{a}_{i-1}\mathbf{w}_{k_h}\right)^t + \left(\mathbf{a}_{i-1}\mathbf{w}_{q_h} + \mathbf{b}_{q_h}\right)\widetilde{\mathbf{b}_{k_h}}^t\right) \\
&= \text{softmax}\left(\left(\mathbf{a}_{i-1}\mathbf{w}_{q_h} + \widetilde{\mathbf{b}}_{q_h}\right)\left(\mathbf{a}_{i-1}\mathbf{w}_{k_h}\right)^t\right) \qquad (19)
\end{aligned}$$

where the last equality shows that the dependence of $\boldsymbol{\rho}_{(\mathbf{a}_{i-1},\, h)}$ on $\mathbf{b}_{k_h}$ completely disappears. Essentially, it stems from the fact that any matrix $\mathbf{H}$ multiplied by the transpose of a **broadcast** vector $\mathbf{b}_{k_h}$ produces a matrix $\mathbf{H}\widetilde{\mathbf{b}}_{k_h}^t$ where the rows are all constant. Since the softmax normalization is shift-invariant, this constant shift of the rows cancels out with $\text{softmax}(\mathbf{G} + \mathbf{H}\widetilde{\mathbf{b}}_{k_h}^t) = \text{softmax}\, \mathbf{G}$ so that the dependence on $\mathbf{b}_{k_h}$ drops out. This identity is proven in detail in eq.(54) of the appendix section. Note that the apparent asymmetry between queries and keys is incidental. In order to produce a square self-attention matrix, one of either keys or queries needs to be transposed and it is for this one that the dependence of $\boldsymbol{\rho}_{(\mathbf{a}_{i-1},\, h)}$ on their bias term will drop out. If one were to swap the notation but continue to define attention weight vectors as the rows of $\boldsymbol{\rho}_{(\mathbf{a}_{i-1},\, h)} \sim \text{softmax}(\mathbf{k}_h\, \mathbf{q}_h^t)$, in this case it would be the $\mathbf{b}_{q_h}$ that would become redundant. Other than this, the self-attention layer would behave in exactly the same manner confirming that queries and keys are internal parameters used to define attention weights and remain completely interchangeable.

is defined by eq.(16), which is specialized to $t^\star$ following eq.(47), yielding

$$\mathbf{a}_{i+1}(t=t^\star) = \sum_{t_\alpha} \rho^{(i)}_{t^\star t_\alpha} \mathbf{v}_{(i)}(t=t_\alpha) = \sum_{t_\alpha} \rho^{(i)}_{t^\star t_\alpha} \mathbf{a}_i(t=t_\alpha)\mathbf{w}_v$$

where we ignored the biases in eq.(4) (this does not affect the conclusions and just helps simplify the notation) and introduced an index (i) in the attention weights and values vectors to indicate that those are specific to $\mathbf{a}_i$. Moving on, we now need an expression for $\mathbf{a}_i(t=t_\alpha)$, which following the same logic is expressed as a similar weighted sum

$$\mathbf{a}_i(t=t_\alpha) = \sum_{t_\beta} \rho^{(i-1)}_{t_\alpha t_\beta} \mathbf{v}_{(i-1)}(t=t_\beta) = \sum_{t_\beta} \rho^{(i-1)}_{t_\alpha t_\beta} \mathbf{a}_{i-1}(t=t_\beta)\mathbf{w}_v$$

Putting all the pieces back together, we get

$$\mathbf{a}_{i+1}(t=t^\star) = \sum_{t_\alpha}\sum_{t_\beta} \rho^{(i)}_{t^\star t_\alpha} \rho^{(i-1)}_{t_\alpha t_\beta} \mathbf{a}_{i-1}(t=t_\beta)\mathbf{w}_v^2 \tag{20}$$

thereby showing explicitly how the weighted sum that defines $\mathbf{a}_{i+1}$ after two layers of self-attention now involves third-order $(t^\star, t_\alpha, t_\beta)$ token interactions instead of just pairwise relationships when we considered only a single layer of self-attention. Composing even more layers together generates higher-order interactions eventually spanning the entire sequence. [4]

3.2 Self-attention layer: Single head — Backward pass

Just like any other layer, the backward pass through a self-attention layer starts by evaluating the recursive backward error flow $\boldsymbol{\Delta}_i^h \cdot \mathrm{d}\mathbf{a}_i^h$ and gradient extraction described in eq.(11) of the reference paper [1]. Here $\boldsymbol{\Delta}_i^h \sim \mathbb{R}^{n_\mathcal{T} \times d_h}$ represents the upstream error flow going into attention head h that was produced by layers closer to the loss function and $\mathbf{a}_i^h \sim \mathbb{R}^{n_\mathcal{T} \times d_h}$ represents the $n_\mathcal{T}$ feature maps, each of dimensionality $\sim \mathbb{R}^{d_h}$, produced by the same attention head h. We defer the discussion of how the complete error signal $\boldsymbol{\Delta}_i \sim \mathbb{R}^{n_\mathcal{T} \times d}$ is split for each attention head h to Section 4.

Given an attention head h, its output data representation $\mathbf{a}_i^h = \boldsymbol{\rho}_{(\mathbf{a}_{i-1},\,h)}\,\mathbf{v}_h$ is given by a weighted average of the value feature maps $\mathbf{v}_h$ with the attention matrix $\boldsymbol{\rho}_{(\mathbf{a}_{i-1},\,h)}$, see eq.(16). Writing out the recursive backward error flow explicitly, we have

$$\begin{aligned}
\boldsymbol{\Delta}_i^h \cdot \mathrm{d}\mathbf{a}_i^h &= \boldsymbol{\Delta}_i^h \cdot \left[\left(\mathrm{d}\boldsymbol{\rho}_{(\mathbf{a}_{i-1},\,h)}\right)\mathbf{v}_h + \boldsymbol{\rho}_{(\mathbf{a}_{i-1},\,h)}\mathrm{d}\mathbf{v}_h\right] \\
&= \boldsymbol{\Delta}_i^h \cdot \left(\mathrm{d}\boldsymbol{\rho}_{(\mathbf{a}_{i-1},\,h)}\right)\mathbf{v}_h + \boldsymbol{\Delta}_i^h \cdot \left(\boldsymbol{\rho}_{(\mathbf{a}_{i-1},\,h)}\mathrm{d}\mathbf{v}_h\right) \\
&= \left(\boldsymbol{\Delta}_i^h \mathbf{v}_h^t\right) \cdot \mathrm{d}\boldsymbol{\rho}_{(\mathbf{a}_{i-1},\,h)} + \left(\boldsymbol{\rho}^t_{(\mathbf{a}_{i-1},\,h)}\boldsymbol{\Delta}_i^h\right) \cdot \mathrm{d}\mathbf{v}_h
\end{aligned}$$

At this point, the error flow splits into **two different branches** due to the definition of $\mathbf{a}_i^h$ as a product between the self-attention weights $\boldsymbol{\rho}_{(\mathbf{a}_{i-1},\,h)}$ and the feature maps of the values $\mathbf{v}_h$.

Since $\mathbf{v}_h$ is produced by a fully-connected layer with the input sequence $\mathbf{a}_{i-1}$, this branch is terminal as it already comes back to the source data of the self-attention layer. Applying directly the formulas already established in Section 5 of the reference paper [1] for error signal propagation and gradient extraction through fully-connected layers, we immediately get

$$\begin{aligned}
\left(\boldsymbol{\rho}^t_{(\mathbf{a}_{i-1},\,h)}\boldsymbol{\Delta}_i^h\right) \cdot \mathrm{d}\mathbf{v}_h &= \left(\boldsymbol{\rho}^t_{(\mathbf{a}_{i-1},\,h)}\boldsymbol{\Delta}_i^h\right) \cdot \mathrm{d}\left(\mathbf{a}_{i-1}\mathbf{w}_{v_h} + \widetilde{\mathbf{b}_{v_h}}\right) \\
&= \underbrace{\mathbf{a}_{i-1}^t\,\boldsymbol{\rho}^t_{(\mathbf{a}_{i-1},\,h)}\boldsymbol{\Delta}_i^h}_{\frac{\partial \mathcal{L}_{\text{seq}}}{\partial \mathbf{w}_{v_h}}} \cdot \mathrm{d}\mathbf{w}_{v_h} + \underbrace{\sum_{\text{tokens}} \boldsymbol{\rho}^t_{(\mathbf{a}_{i-1},\,h)}\boldsymbol{\Delta}_i^h}_{\frac{\partial \mathcal{L}_{\text{seq}}}{\partial \mathbf{b}_{v_h}}} \cdot \mathrm{d}\mathbf{b}_{v_h} + \underbrace{\boldsymbol{\rho}^t_{(\mathbf{a}_{i-1},\,h)}\boldsymbol{\Delta}_i^h\mathbf{w}_{v_h}^t}_{\boldsymbol{\Delta}^h_{v_h}} \cdot \mathrm{d}\mathbf{a}_{i-1}
\end{aligned}$$

[4] While on the topic of composition of deep learning layers, this is an occasion to observe, in practice, the importance of normalization (such as the softmax and scaling methods discussed above). Indeed, as a high-level approximation, one can model the matrix products that appear in eq.(20) as a product of random matrices. Under this assumption, it may be shown that the variance of $\mathbf{a}_{i+1}$ is unbounded and grows exponentially with the number of products [11]. Therefore, normalization layers are required to rescale the data representations and stabilize the training of the model.

Because of the softmax normalization of $\boldsymbol{\rho}_{(\mathbf{a}_{i-1},\,h)}$, the expression $\partial\mathcal{L}_{\text{seq}}/\partial\mathbf{b}_{v_h}$ above for the gradient for the biases can be simplified further. Writing it out explicitly using the graphical representation of the transpose of the attention weight matrix, see.(15), we have

$$\begin{aligned}
\frac{\partial\mathcal{L}_{\text{seq}}}{\partial\mathbf{b}_{v_h}} &= \sum_{\text{tokens}} \begin{pmatrix} | & & | \\ \boldsymbol{\rho}_{(\mathbf{a}_{i-1},\,h)}(t=1) & \cdots & \boldsymbol{\rho}_{(\mathbf{a}_{i-1},\,h)}(t=n_{\mathcal{T}}) \\ | & & | \end{pmatrix} \boldsymbol{\Delta}_i^h \\
& \downarrow \boxed{\text{using eq.(52)}} \\
&= \left(\sum_{t'=1}^{n_{\mathcal{T}}} \rho_{1t'}\,, \cdots, \sum_{t'=1}^{n_{\mathcal{T}}} \rho_{n_{\mathcal{T}}t'} \right) \boldsymbol{\Delta}_i^h \\
& \downarrow \boxed{\text{because of the softmax normalization of attention weight vectors, see eq.(14)}}\,. \\
&= (1, \cdots, 1)\, \boldsymbol{\Delta}_i^h \\
\frac{\partial\mathcal{L}_{\text{seq}}}{\partial\mathbf{b}_{v_h}} &= \sum_{\text{tokens}} \boldsymbol{\Delta}_i^h \sim \mathbb{R}^{d_h} \qquad (21)
\end{aligned}$$

In other words, the gradients of the biases of the fully-connected layer that determines $\mathbf{v}_h$ do not depend on the attention weight matrix $\boldsymbol{\rho}_{(\mathbf{a}_{i-1},\,h)}$ of attention head h but only on its (head-specific) allocated upstream error signal $\boldsymbol{\Delta}_i^h$. This is a direct consequence of the softmax normalization of the attention weight matrix. We will see later how the same constraint also impacts other bias gradients.

In summary, the backward pass through the value feature maps $\mathbf{v}_h$ branch of self-attention leads to error propagation and gradients given by the following expressions

Values: backward pass

$$\boldsymbol{\Delta}_{v_h}^h = \boldsymbol{\rho}_{(\mathbf{a}_{i-1},\,h)}^t \boldsymbol{\Delta}_i^h \mathbf{w}_{v_h}^t \qquad \sim \mathbb{R}^{n_{\mathcal{T}} \times d} \tag{22}$$

$$\frac{\partial\mathcal{L}_{\text{seq}}}{\partial\mathbf{w}_{v_h}} = \left(\boldsymbol{\rho}_{(\mathbf{a}_{i-1},\,h)}\, \mathbf{a}_{i-1}\right)^t \boldsymbol{\Delta}_i^h \qquad \sim \mathbb{R}^{d \times d_h} \tag{23}$$

$$\frac{\partial\mathcal{L}_{\text{seq}}}{\partial\mathbf{b}_{v_h}} = \sum_{\text{tokens}} \boldsymbol{\Delta}_i^h \qquad \sim \mathbb{R}^{d_h} \tag{24}$$

We can now move on to the second branch related to backpropagation through the self-attention weight matrix $\boldsymbol{\rho}_{(\mathbf{a}_{i-1},\,h)}$. Repeating the expression already derived above and replacing $\boldsymbol{\rho}_{(\mathbf{a}_{i-1},\,h)}$ by its definition as the softmax normalized version of $\boldsymbol{\rho}^{\text{causal}}_{(\mathbf{a}_{i-1},\,h)}$, see eq.(15), we have

$$\begin{aligned}
\left(\boldsymbol{\Delta}_i^h \mathbf{v}_h^t\right) \cdot \mathrm{d}\boldsymbol{\rho}_{(\mathbf{a}_{i-1},\,h)} &= \left(\boldsymbol{\Delta}_i^h \mathbf{v}_h^t\right) \cdot \mathrm{d}\big(\text{softmax}\, \boldsymbol{\rho}^{\text{causal}}_{(\mathbf{a}_{i-1},\,h)}\big) \\
& \downarrow \boxed{\text{using eq.(13) of the reference paper [1]}} \\
&= \left(\boldsymbol{\Delta}_i^h \mathbf{v}_h^t\right) \cdot \left[\boldsymbol{\rho}_{(\mathbf{a}_{i-1},\,h)} \circ \left(\mathrm{d}\boldsymbol{\rho}^{\text{causal}}_{(\mathbf{a}_{i-1},\,h)} - \boldsymbol{\rho}_{(\mathbf{a}_{i-1},\,h)} \widetilde{\ominus}\, \mathrm{d}\boldsymbol{\rho}^{\text{causal}}_{(\mathbf{a}_{i-1},\,h)} \right) \right] \\
&= \left(\boldsymbol{\Delta}_i^h \mathbf{v}_h^t\right) \cdot \left[\boldsymbol{\rho}_{(\mathbf{a}_{i-1},\,h)} \circ \mathrm{d}\boldsymbol{\rho}^{\text{causal}}_{(\mathbf{a}_{i-1},\,h)} - \boldsymbol{\rho}_{(\mathbf{a}_{i-1},\,h)} \circ \left(\boldsymbol{\rho}_{(\mathbf{a}_{i-1},\,h)} \widetilde{\ominus}\, \mathrm{d}\boldsymbol{\rho}^{\text{causal}}_{(\mathbf{a}_{i-1},\,h)} \right) \right] \\
&= \left(\boldsymbol{\Delta}_i^h \mathbf{v}_h^t\right) \cdot \left[\boldsymbol{\rho}_{(\mathbf{a}_{i-1},\,h)} \circ \mathrm{d}\boldsymbol{\rho}^{\text{causal}}_{(\mathbf{a}_{i-1},\,h)} \right] - \left(\boldsymbol{\Delta}_i^h \mathbf{v}_h^t\right) \cdot \left[\boldsymbol{\rho}_{(\mathbf{a}_{i-1},\,h)} \circ \left(\boldsymbol{\rho}_{(\mathbf{a}_{i-1},\,h)} \widetilde{\ominus}\, \mathrm{d}\boldsymbol{\rho}^{\text{causal}}_{(\mathbf{a}_{i-1},\,h)} \right) \right] \\
& \downarrow \boxed{\text{using eq.(53) of the reference paper [1] on the left-most term}} \\
&= \left(\boldsymbol{\Delta}_i^h \mathbf{v}_h^t\right) \cdot \left[\boldsymbol{\rho}_{(\mathbf{a}_{i-1},\,h)} \circ \mathrm{d}\boldsymbol{\rho}^{\text{causal}}_{(\mathbf{a}_{i-1},\,h)} \right] - \left[\left(\boldsymbol{\Delta}_i^h \mathbf{v}_h^t\right) \circ \boldsymbol{\rho}_{(\mathbf{a}_{i-1},\,h)} \right] \cdot \left(\boldsymbol{\rho}_{(\mathbf{a}_{i-1},\,h)} \widetilde{\ominus}\, \mathrm{d}\boldsymbol{\rho}^{\text{causal}}_{(\mathbf{a}_{i-1},\,h)} \right) \\
& \downarrow \boxed{\text{using eqs.(50)-(51) on the left-most term}} \\
&= \left[\left(\boldsymbol{\Delta}_i^h \mathbf{v}_h^t\right) \circ \boldsymbol{\rho}_{(\mathbf{a}_{i-1},\,h)} \right] \cdot \mathrm{d}\boldsymbol{\rho}^{\text{causal}}_{(\mathbf{a}_{i-1},\,h)} - \left(\left[\left(\boldsymbol{\Delta}_i^h \mathbf{v}_h^t\right) \widetilde{\ominus}\, \boldsymbol{\rho}_{(\mathbf{a}_{i-1},\,h)} \right] \circ \boldsymbol{\rho}_{(\mathbf{a}_{i-1},\,h)} \right) \cdot \mathrm{d}\boldsymbol{\rho}^{\text{causal}}_{(\mathbf{a}_{i-1},\,h)}
\end{aligned}$$

$$= \underbrace{\left[\boldsymbol{\Delta}_i^h \mathbf{v}_h^t - \widetilde{\left(\boldsymbol{\Delta}_i^h \mathbf{v}_h^t \right) \ominus} \, \boldsymbol{\rho}_{(\mathbf{a}_{i-1},\, h)} \right] \circ \boldsymbol{\rho}_{(\mathbf{a}_{i-1},\, h)}}_{\boldsymbol{\Delta}^h_{\text{causal}} \sim \mathbb{R}^{n_\mathcal{T} \times n_\mathcal{T}}} \cdot \mathrm{d}\boldsymbol{\rho}^{\text{causal}}_{(\mathbf{a}_{i-1},\, h)} \tag{25}$$

At this point, it is interesting to pause and take a deeper look at the structure of $\boldsymbol{\Delta}^h_{\text{causal}}$. We saw in the forward pass that the rows of $\boldsymbol{\rho}_{(\mathbf{a}_{i-1},\, h)}$ are normalized into a probability distribution. Now that we have propagated the error flow back through the softmax function responsible for this normalization, we should expect $\boldsymbol{\Delta}^h_{\text{causal}}$ to also reflect this constraint on the self-attention weights. Generally, $\boldsymbol{\Delta}^h_{\text{causal}}$ can be written as a row stack of error flow vectors $\sim \mathbb{R}^{n_\mathcal{T}}$ associated with the $n_\mathcal{T}$ tokens in the sequence just as we did for the rows of $\boldsymbol{\rho}_{(\mathbf{a}_{i-1},\, h)}$, see eq.(15). Therefore, we have

$$\boldsymbol{\Delta}^h_{\text{causal}} = \begin{pmatrix} - \, \boldsymbol{\delta}^h_{\text{causal}}(t=1) \sim \mathbb{R}^{n_\mathcal{T}} \, - \\ \vdots \\ - \, \boldsymbol{\delta}^h_{\text{causal}}(t=n_\mathcal{T}) \sim \mathbb{R}^{n_\mathcal{T}} \, - \end{pmatrix} = \begin{pmatrix} \delta^{\text{causal}}_{11} & \cdots & \delta^{\text{causal}}_{1 n_\mathcal{T}} \\ \vdots & \ddots & \vdots \\ \delta^{\text{causal}}_{n_\mathcal{T} 1} & \cdots & \delta^{\text{causal}}_{n_\mathcal{T} n_\mathcal{T}} \end{pmatrix}_h \sim \mathbb{R}^{n_\mathcal{T} \times n_\mathcal{T}}$$

Now, in complete analogy with eq. (14) where we considered one row $\boldsymbol{\rho}_{(\mathbf{a}_{i-1},\, h)}(t = t^\star) \sim \mathbb{R}^{n_\mathcal{T}}$ of the self-attention matrix and verified the normalization $\sum \boldsymbol{\rho}_{(\mathbf{a}_{i-1},\, h)}(t = t^\star) = 1$ of the pairwise attention weights $\rho_{t^\star t'}$ between token $t^\star$ and all the other tokens $t' \in [1, \cdots, n_\mathcal{T}]$, let us now consider one row $\boldsymbol{\delta}_{\text{causal}}(t = t^\star) \sim \mathbb{R}^{n_\mathcal{T}}$ of $\boldsymbol{\Delta}^h_{\text{causal}}$ and evaluate the sum of its $n_\mathcal{T}$ components

$$\sum_{t'=1}^{n_\mathcal{T}} \boldsymbol{\delta}_{\text{causal}}(t = t^\star) = \sum_{t'=1}^{n_\mathcal{T}} \left[\left(\boldsymbol{\Delta}_i^h \mathbf{v}_h^t \right)_{t^\star} - \widetilde{\left(\boldsymbol{\Delta}_i^h \mathbf{v}_h^t \right)_{t^\star} \cdot \left(\boldsymbol{\rho}_{(\mathbf{a}_{i-1},\, h)} \right)_{t^\star}} \right] \circ \left(\boldsymbol{\rho}_{(\mathbf{a}_{i-1},\, h)} \right)_{t^\star}$$

since we consider one row of $\boldsymbol{\Delta}^h_{\text{causal}}$, the operator $\ominus$ reduces to a dot-product

we use the notation $(\cdots)_{t^\star} \sim \mathbb{R}^{n_\mathcal{T}}$ as shorthand for $(\cdots)(t = t^\star)$, i.e. the $t^\star$ row of $(\cdots)$

$$= \sum_{t'=1}^{n_\mathcal{T}} \left[\left(\boldsymbol{\Delta}_i^h \mathbf{v}_h^t \right)_{t^\star} \circ \left(\boldsymbol{\rho}_{(\mathbf{a}_{i-1},\, h)} \right)_{t^\star} \right] - \sum_{t'=1}^{n_\mathcal{T}} \left[\left[\widetilde{\left(\boldsymbol{\Delta}_i^h \mathbf{v}_h^t \right)_{t^\star} \cdot \left(\boldsymbol{\rho}_{(\mathbf{a}_{i-1},\, h)} \right)_{t^\star}} \right] \circ \left(\boldsymbol{\rho}_{(\mathbf{a}_{i-1},\, h)} \right)_{t^\star} \right]$$

$\downarrow$ because $\sum \mathbf{a} \circ \mathbf{b} = \mathbf{a} \cdot \mathbf{b} \sim \mathbb{R}$ and the broadcast is $\widetilde{\mathbf{a} \cdot \mathbf{b}} = (\mathbf{a} \cdot \mathbf{b})[1, \cdots, 1] \sim \mathbb{R}^{n_\mathcal{T}}$

and $(\mathbf{a} \cdot \mathbf{b})$ can be taken out of the sum in the left-most expression

$$= \left(\boldsymbol{\Delta}_i^h \mathbf{v}_h^t \right)_{t^\star} \cdot \left(\boldsymbol{\rho}_{(\mathbf{a}_{i-1},\, h)} \right)_{t^\star} - \left[\left(\boldsymbol{\Delta}_i^h \mathbf{v}_h^t \right)_{t^\star} \cdot \left(\boldsymbol{\rho}_{(\mathbf{a}_{i-1},\, h)} \right)_{t^\star} \right] \sum_{t'=1}^{n_\mathcal{T}} \boldsymbol{\rho}_{(\mathbf{a}_{i-1},\, h)}(t = t^\star)$$

$\downarrow$ because of the softmax normalization defined in eq.(14) with $\sum_{t'=1}^{n_\mathcal{T}} \boldsymbol{\rho}_{(\mathbf{a}_{i-1},\, h)}(t = t^\star) = 1$

$$= \left(\boldsymbol{\Delta}_i^h \mathbf{v}_h^t \right)_{t^\star} \cdot \left(\boldsymbol{\rho}_{(\mathbf{a}_{i-1},\, h)} \right)_{t^\star} - \left(\boldsymbol{\Delta}_i^h \mathbf{v}_h^t \right)_{t^\star} \cdot \left(\boldsymbol{\rho}_{(\mathbf{a}_{i-1},\, h)} \right)_{t^\star}$$

$$= 0$$

This confirms that, just like the rows of $\boldsymbol{\rho}_{(\mathbf{a}_{i-1},\, h)}$ which are constrained by eq.(14), also the rows of $\boldsymbol{\Delta}^h_{\text{causal}}$ reflect this conservation law on probability mass with another constraint on the gradients

$$\sum_{t'=1}^{n_\mathcal{T}} \boldsymbol{\delta}^h_{\text{causal}}(t = t^\star) = \sum_{t'=1}^{n_\mathcal{T}} \left(\delta^{\text{causal}}_{t^\star t'} \right)_h = 0 \quad ; \text{for each } t^\star \in \left[1, \cdots, n_\mathcal{T} \right] \tag{26}$$

This observation is a general property due to the softmax normalization and will manifest itself later when we inspect the gradients with respect to the biases of the keys.

Now that we have finished this pause on analyzing the structure of $\boldsymbol{\Delta}^h_{\text{causal}}$, we can move back to propagating the error signal one more step back through the causal mask prescribed by eq.(13). Since this operation is parameter-free, there are no gradients to extract and

$$\begin{aligned}
\boldsymbol{\Delta}^h_{\text{causal}} \cdot \mathrm{d}\boldsymbol{\rho}^{\text{causal}}_{(\mathbf{a}_{i-1},\,h)} &= \boldsymbol{\Delta}^h_{\text{causal}} \cdot \mathrm{d}\left(\mathbf{m}\circ\boldsymbol{\rho}^{\text{scaled}}_{(\mathbf{a}_{i-1},\,h)}\right) \\
&= \boldsymbol{\Delta}^h_{\text{causal}} \cdot \left(\mathbf{m}\circ \mathrm{d}\boldsymbol{\rho}^{\text{scaled}}_{(\mathbf{a}_{i-1},\,h)}\right) \\
&\downarrow \ \text{eq.(53) in the reference paper [1]} \\
&= \mathbf{m}\circ\boldsymbol{\Delta}^h_{\text{causal}} \cdot \mathrm{d}\boldsymbol{\rho}^{\text{scaled}}_{(\mathbf{a}_{i-1},\,h)} \\
&\downarrow \ \text{because } \boldsymbol{\Delta}^h_{\text{causal}} \text{ is proportional to } \boldsymbol{\rho}_{(\mathbf{a}_{i-1},\,h)} \text{ which already contains the mask } \mathbf{m} \\
&= \underbrace{\boldsymbol{\Delta}^h_{\text{causal}}}_{\boldsymbol{\Delta}^h_{\text{scaled}}} \cdot \mathrm{d}\boldsymbol{\rho}^{\text{scaled}}_{(\mathbf{a}_{i-1},\,h)}
\end{aligned} \tag{27}$$

Since $\boldsymbol{\Delta}^h_{\text{causal}}$ is proportional to $\boldsymbol{\rho}_{(\mathbf{a}_{i-1},\,h)}$ which already contains the mask $\mathbf{m}$, the error flow stays unchanged with $\boldsymbol{\Delta}^h_{\text{scaled}} = \boldsymbol{\Delta}^h_{\text{causal}}$.

The next step is another parameter-free scaling given by eq.(12). Applying the usual recursive backward error propagation, we have

$$\begin{aligned}
\boldsymbol{\Delta}^h_{\text{scaled}} \cdot \mathrm{d}\boldsymbol{\rho}^{\text{scaled}}_{(\mathbf{a}_{i-1},\,h)} &= \boldsymbol{\Delta}^h_{\text{scaled}} \cdot \mathrm{d}\left(\boldsymbol{\rho}^{\text{raw}}_{(\mathbf{a}_{i-1},\,h)} \,/\, \sqrt{d_\rho}\right) \\
&= \underbrace{\boldsymbol{\Delta}^h_{\text{scaled}}/\sqrt{d_\rho}}_{\boldsymbol{\Delta}^h_{\text{raw}}} \cdot \mathrm{d}\boldsymbol{\rho}^{\text{raw}}_{(\mathbf{a}_{i-1},\,h)}
\end{aligned} \tag{28}$$

We have now reached the point where the raw self-attention weights $\boldsymbol{\rho}^{\text{raw}}_{(\mathbf{a}_{i-1},\,h)} \sim \mathbb{R}^{n_\mathcal{T}\times n_\mathcal{T}}$ are determined by eq.(10) as the dot-product between the queries $\mathbf{q}_h \sim \mathbb{R}^{n_\mathcal{T}\times d_\rho}$ and the keys $\mathbf{k}_h \sim \mathbb{R}^{n_\mathcal{T}\times d_\rho}$ feature maps of the attention head h. Propagating $\boldsymbol{\Delta}^h_{\text{raw}} \sim \mathbb{R}^{n_\mathcal{T}\times n_\mathcal{T}}$ back through this product, we have

$$\begin{aligned}
\boldsymbol{\Delta}^h_{\text{raw}} \cdot \mathrm{d}\boldsymbol{\rho}^{\text{raw}}_{(\mathbf{a}_{i-1},\,h)} &= \boldsymbol{\Delta}^h_{\text{raw}} \cdot \mathrm{d}\left(\mathbf{q}_h\,\mathbf{k}^t_h\right) \\
&= \boldsymbol{\Delta}^h_{\text{raw}} \cdot \left[\, (\mathrm{d}\mathbf{q}_h)\,\mathbf{k}^t_h + \mathbf{q}_h\,(\mathrm{d}\mathbf{k}^t_h)\,\right] \\
&\downarrow \ \text{eq.(52) in the reference paper [1] and eq.(49) of this paper} \\
&= \left(\boldsymbol{\Delta}^h_{\text{raw}}\,\mathbf{k}_h\right)\cdot \mathrm{d}\mathbf{q}_h + \left((\boldsymbol{\Delta}^h_{\text{raw}})^t\,\mathbf{q}_h\right)\cdot \mathrm{d}\mathbf{k}_h
\end{aligned}$$

which splits into two different branches due to the queries/keys product.

- Let us first focus on the queries $\mathbf{q}_h$ and use their definition from eq.(9.a) to evaluate the first branch

$$\begin{aligned}
\left(\boldsymbol{\Delta}^h_{\text{raw}}\,\mathbf{k}_h\right)\cdot \mathrm{d}\mathbf{q}_h &= \left(\boldsymbol{\Delta}^h_{\text{raw}}\,\mathbf{k}_h\right)\cdot \mathrm{d}\left(\mathbf{a}_{i-1}\mathbf{w}_{q_h} + \widetilde{\mathbf{b}_{q_h}}\right) \\
&= \underbrace{\mathbf{a}^t_{i-1}\,\boldsymbol{\Delta}^h_{\text{raw}}\,\mathbf{k}_h}_{\frac{\partial\mathcal{L}_{\text{seq}}}{\partial\mathbf{w}_{q_h}}} \cdot \mathrm{d}\mathbf{w}_{q_h} + \underbrace{\sum_{\text{tokens}} \boldsymbol{\Delta}^h_{\text{raw}}\,\mathbf{k}_h}_{\frac{\partial\mathcal{L}_{\text{seq}}}{\partial\mathbf{b}_{q_h}}} \cdot \mathrm{d}\mathbf{b}_{q_h} + \underbrace{\boldsymbol{\Delta}^h_{\text{raw}}\,\mathbf{k}_h\mathbf{w}^t_{q_h}}_{\boldsymbol{\Delta}_{q_h}} \cdot \mathrm{d}\mathbf{a}_{i-1}
\end{aligned}$$

As a simple fully-connected layer, those expressions for the gradients and error signal propagation through the queries can be directly adapted from Section 5 of the reference paper [1].

- Next, we look at the keys $\mathbf{k}_h$ and use their definition from eq.(9.b) to evaluate the second branch

$$\begin{aligned}\boxed{\left((\boldsymbol{\Delta}^h_{\text{raw}})^t \, \mathbf{q}_h \right) \cdot \mathrm{d}\mathbf{k}_h} &= \left((\boldsymbol{\Delta}^h_{\text{raw}})^t \, \mathbf{q}_h \right) \cdot \mathrm{d} \left(\mathbf{a}_{i-1} \mathbf{w}_{k_h} + \widetilde{\mathbf{b}_{k_h}} \right) \\ &= \underbrace{\mathbf{a}^t_{i-1} \, (\boldsymbol{\Delta}^h_{\text{raw}})^t \, \mathbf{q}_h}_{\frac{\partial \mathcal{L}_{\text{seq}}}{\partial \mathbf{w}_{k_h}}} \cdot \mathrm{d}\mathbf{w}_{k_h} + \underbrace{\sum_{\text{tokens}} (\boldsymbol{\Delta}^h_{\text{raw}})^t \, \mathbf{q}_h}_{\frac{\partial \mathcal{L}_{\text{seq}}}{\partial \mathbf{b}_{k_h}} = \mathbf{0}} \cdot \mathrm{d}\mathbf{b}_{k_h} + \underbrace{(\boldsymbol{\Delta}^h_{\text{raw}})^t \, \mathbf{q}_h \, \mathbf{w}^t_{k_h}}_{\boldsymbol{\Delta}_{k_h}} \cdot \mathrm{d}\mathbf{a}_{i-1}\end{aligned}$$

Since this is another fully-connected layer, we get similar expressions as those we saw for the queries. The only difference is that, now, the gradient of the loss with respect to the biases is identically null. Let us see why this is the case. We have seen previously that the softmax normalization imposes a mirror constraint on the rows of $\boldsymbol{\Delta}^h_{\text{raw}} = \boldsymbol{\Delta}^h_{\text{causal}} / \sqrt{d_\rho} \sim \mathbb{R}^{n_\mathcal{T} \times n_\mathcal{T}}$, see eq.(26). Since we are interested in its transposed version $(\boldsymbol{\Delta}^h_{\text{raw}})^t$, rows become columns and we represent this graphically as

$$\boldsymbol{\Delta}^h_{\text{raw}} = \begin{pmatrix} — \, \boldsymbol{\delta}^h_{\text{raw}}(t=1) \sim \mathbb{R}^{n_\mathcal{T}} \, — \\ \vdots \\ — \, \boldsymbol{\delta}^h_{\text{raw}}(t=n_\mathcal{T}) \sim \mathbb{R}^{n_\mathcal{T}} \, — \end{pmatrix} \implies (\boldsymbol{\Delta}^h_{\text{raw}})^t = \begin{pmatrix} | & & | \\ \boldsymbol{\delta}^h_{\text{raw}}(t=1) & \cdots & \boldsymbol{\delta}^h_{\text{raw}}(t=n_\mathcal{T}) \\ | & & | \end{pmatrix}$$

where $\boldsymbol{\delta}^h_{\text{raw}}(t = t^\star) = \left[\delta^{\text{raw}}_{t^\star 1}, \cdots, \delta^{\text{raw}}_{t^\star n} \right]_h$ for each $t^\star \in \left[1, \cdots, n_\mathcal{T} \right]$. Let us now evaluate the expression derived above for the gradient of the loss with respect to $\mathbf{b}_{k_h}$ as

$$\begin{aligned}\frac{\partial \mathcal{L}_{\text{seq}}}{\partial \mathbf{b}_{k_h}} = \sum_{\text{tokens}} (\boldsymbol{\Delta}^h_{\text{raw}})^t \, \mathbf{q}_h &= \frac{1}{\sqrt{d_\rho}} \sum_{\text{tokens}} \begin{pmatrix} \delta^{\text{causal}}_{11} & \cdots & \delta^{\text{causal}}_{n_\mathcal{T} 1} \\ | & \cdots & | \\ \delta^{\text{causal}}_{1 n_\mathcal{T}} & \cdots & \delta^{\text{causal}}_{n_\mathcal{T} n_\mathcal{T}} \end{pmatrix}_h \mathbf{q}_h \\ &\downarrow \; \text{using eq.(52)} \\ &= \frac{1}{\sqrt{d_\rho}} \left(\sum_{t'=1}^{n_\mathcal{T}} (\delta^{\text{causal}}_{1t'})_h \, , \cdots , \sum_{t'=1}^{n_\mathcal{T}} (\delta^{\text{causal}}_{n_\mathcal{T} t'})_h \right) \mathbf{q}_h \\ &\downarrow \; \text{because of the constraint } \textstyle\sum_{t'=1}^{n_\mathcal{T}} (\delta^{\text{causal}}_{t^\star t'})_h = 0 \\ & \; \text{due to softmax normalization, see eq.(26)}\end{aligned}$$

$$\frac{\partial \mathcal{L}_{\text{seq}}}{\partial \mathbf{b}_{k_h}} = \mathbf{0} \sim \mathbb{R}^{d_\rho} \tag{29}$$

This shows that the gradients with respect to the biases of the keys are identically null. This is a consequence of our previous observation on the gradient constraint. It is also consistent with our discussion about the independence of the self-attention weights $\boldsymbol{\rho}_{(\mathbf{a}_{i-1}, h)}$ on the biases $\mathbf{b}_{k_h}$ of the keys during the forward pass.

In summary, we have

Queries and Keys: backward pass

$$\boldsymbol{\Delta}_{q_h} = \boldsymbol{\Delta}^h_{\text{raw}} \, \mathbf{k}_h \mathbf{w}^t_{q_h} \quad ; \quad \boldsymbol{\Delta}_{k_h} = (\boldsymbol{\Delta}^h_{\text{raw}})^t \, \mathbf{q}_h \, \mathbf{w}^t_{k_h} \quad \sim \mathbb{R}^{n_\mathcal{T} \times d} \tag{30.a-b}$$

$$\frac{\partial \mathcal{L}_{\text{seq}}}{\partial \mathbf{w}_{q_h}} = \mathbf{a}^t_{i-1} \, \boldsymbol{\Delta}^h_{\text{raw}} \, \mathbf{k}_h \quad ; \quad \frac{\partial \mathcal{L}_{\text{seq}}}{\partial \mathbf{w}_{k_h}} = (\boldsymbol{\Delta}^h_{\text{raw}} \, \mathbf{a}_{i-1})^t \, \mathbf{q}_h \quad \sim \mathbb{R}^{d \times d_\rho} \tag{31.a-b}$$

$$\frac{\partial \mathcal{L}_{\text{seq}}}{\partial \mathbf{b}_{q_h}} = \sum_{\text{tokens}} \boldsymbol{\Delta}^h_{\text{raw}} \, \mathbf{k}_h \quad ; \quad \frac{\partial \mathcal{L}_{\text{seq}}}{\partial \mathbf{b}_{k_h}} = \mathbf{0} \quad \sim \mathbb{R}^{d_\rho} \tag{32.a-b}$$

At this point, we have extracted the gradients of the loss function for a single sequence of $n_\mathcal{T}$ tokens with respect to all of the parameters of the self-attention head h that transformed the d-dimensional token feature maps of the input sequence $\mathbf{a}_{i-1} \sim \mathbb{R}^{n_\mathcal{T} \times d}$ into new d_h-dimensional feature maps of the output sequence $\mathbf{a}_i \sim \mathbb{R}^{n_\mathcal{T} \times d_h}$. Those gradients comprise of three fully-connected layers associated with the values $\partial \mathcal{L}_{\text{seq}} / \partial \{ \mathbf{w}_{v_h}, \mathbf{b}_{v_h} \}$, see eqs.(23)-(24), queries $\partial \mathcal{L}_{\text{seq}} / \partial \{ \mathbf{w}_{q_h}, \mathbf{b}_{q_h} \}$ and keys $\partial \mathcal{L}_{\text{seq}} / \partial \{ \mathbf{w}_{k_h}, \mathbf{b}_{k_h} \}$, see eqs.(31.a-b)-(32.a-b).

Along the way, we have also propagated the upstream error signal $\mathbf{\Delta}_i^h \sim \mathbb{R}^{n_\mathcal{T} \times d_h}$ allocated to attention head h from the $\mathbf{a}_i^h \sim \mathbb{R}^{n_\mathcal{T} \times d_h}$ output sequence level downstream back to the level of the input sequence $\mathbf{a}_{i-1} \sim \mathbb{R}^{n_\mathcal{T} \times d}$. As per usual terminology, let us denote by $\mathbf{\Delta}_{i-1}^h \sim \mathbb{R}^{n_\mathcal{T} \times d}$ this downstream error signal. Since there are three trainable layers (values, queries and keys) in the attention head, $\mathbf{\Delta}_{i-1}^h$ should also be made up of three contributions.

We saw at the very first stage of backpropagation how the upstream error signal $\mathbf{\Delta}_i^h$ splits into two different branches. The branch associated with the values feature maps $\mathbf{v}_h$ is directly connected to the input sequence $\mathbf{a}_{i-1}$ and therefore its error term $\mathbf{\Delta}_{v_h}$, see eq.(22), is the first contributor to the downstream signal $\mathbf{\Delta}_{i-1}^h$. The other branch goes through a series of steps (all internal attributes of the attention head h not exposed elsewhere) following $\mathbf{\Delta}_{i-1}^h \to \mathbf{\Delta}_{\text{causal}}^h \to \mathbf{\Delta}_{\text{scaled}}^h \to \mathbf{\Delta}_{\text{raw}}^h$, see eqs.(25), (27), (28) before reaching another split due to the query/key product. Since those feature maps $\mathbf{q}_h$ and $\mathbf{k}_h$ are themselves directly connected to the input sequence $\mathbf{a}_{i-1}$, the process ends here with their respective two contributions $\mathbf{\Delta}_{q_h}$ and $\mathbf{\Delta}_{k_h}$, see eqs.(30.a-b), adding into the downstream error signal $\mathbf{\Delta}_{i-1}^h$.

In summary, the downstream error signal for attention head h is given by

Error signal: backward pass

$$\mathbf{\Delta}_{i-1}^h = \mathbf{\Delta}_{v_h} + \mathbf{\Delta}_{q_h} + \mathbf{\Delta}_{k_h} \quad \sim \mathbb{R}^{n_\mathcal{T} \times d} \tag{33}$$

Notice how (as it should be) the dimensionality of the downstream error signal matches that of the input sequence $\mathbf{\Delta}_{i-1}^h \sim \mathbf{a}_{i-1} \sim \mathbb{R}^{n_\mathcal{T} \times d}$ even though it is produced by a single attention head h. We will see in Section 4 how multiple error signals coming from different attention heads are combined together into a complete $\mathbf{\Delta}_{i-1} \sim \mathbb{R}^{n_\mathcal{T} \times d}$ of the same dimensionality.

Some other noteworthy properties of self-attention

Permutation equivariance in non-causal attention. Here, we focus on non-causal attention where the mask $\mathbf{m}$ in eq.(18) is removed (restriction-free attention range with a matrix of ones $\mathbf{m} = \mathbf{J}_{n_\mathcal{T}}$). Let us consider a permutation matrix $\mathbf{P}_\pi \sim \mathbb{R}^{n_\mathcal{T} \times n_\mathcal{T}}$ and apply it to the tokens (i.e. the rows) of the input sequence (i.e. from the left [12]). Passing this token-order permutated $\pi(n_\mathcal{T})$ input sequence $\mathbf{P}_\pi \, \mathbf{a}_{i-1} \sim \mathbb{R}^{\pi(n_\mathcal{T}) \times d}$ through a non-causal self-attention head h, we get

$$\begin{aligned}
\text{Att}_{\mathcal{P}_h}\big(\mathbf{P}_\pi \, \mathbf{a}_{i-1}\big) &= \text{softmax}\big[\mathbf{P}_\pi \mathbf{q}_h \, (\mathbf{P}_\pi \mathbf{k}_h)^t \, / \sqrt{d_\rho}\,\big] \, \mathbf{P}_\pi \mathbf{v}_h \\
&= \text{softmax}\big[\mathbf{P}_\pi \big(\mathbf{q}_h \mathbf{k}_h^t / \sqrt{d_\rho}\big) \mathbf{P}_\pi^t\big] \, \mathbf{P}_\pi \mathbf{v}_h \\
&\downarrow \ \text{see the commutative properties in eqs.(55.a)-(55.b)} \\
&= \mathbf{P}_\pi \, \text{softmax}\big(\mathbf{q}_h \mathbf{k}_h^t / \sqrt{d_\rho}\big) \mathbf{P}_\pi^t \, \mathbf{P}_\pi \mathbf{v}_h \\
&\downarrow \ \text{since } \mathbf{P}_\pi^t = \mathbf{P}_\pi^{-1} \text{ for permutation matrices [12].} \\
&= \mathbf{P}_\pi \, \text{softmax}\big(\mathbf{q}_h \mathbf{k}_h^t / \sqrt{d_\rho}\big) \, \mathbf{v}_h
\end{aligned}$$

leading to $\text{Att}_{\mathcal{P}_h}\big(\mathbf{P}_\pi \, \mathbf{a}_{i-1}\big) = \mathbf{P}_\pi \, \text{Att}_{\mathcal{P}_h}\big(\mathbf{a}_{i-1}\big)$ demonstrating that non-causal self-attention is equivariant under permutation: Any permutation in the order of the input token feature maps is straightforwardly inherited by the output feature maps which end up permutated in exactly the same manner as the input was. This symmetry ensures that the output feature map $\mathbf{a}_i^h(t = t^\star) \sim \mathbb{R}^{n_\mathcal{T} \times d_h}$ of a token $t^\star \in [1, \cdots, t_{n_\mathcal{T}}]$ stays with the same feature values regardless of its position in the sequence. Permutation equivariance can be understood as a structural constraint enforcing that each token preserves a distinct "identity" (where this "identity" is determined by a token's pairwise attention relationships with every other $n_\mathcal{T}$ tokens in context) which is order-agnostic. For some tasks (such as language) where order is crucial, positional encoding is added into the architecture along with shortcut connections to enforce token-order sensitivity.

Note that this equivariance property should be distinguished from permutation invariance which would require the output representation $\mathbf{a}_i^h$ to always be the same for all possible permutations $\mathbf{P}_\pi \, \mathbf{a}_{i-1}$ of the input effectively reducing $\mathbf{a}_i^h$ to a global "pooling" summary that eliminates individual token identities [13]. Finally, we also point out that permutation equivariance does not hold for causal attention heads where the mask implicitly injects positional information. This raises the question of whether explicit positional encodings remain necessary in architectures employing causal masking [14, 15].

KV cache in autoregressive next-token generation. Let us consider a trained model designed for causal next-token sampling: Given an input sequence $\mathbf{a}_{i-1} \sim \mathbb{N}^{n_\mathcal{T}}$ consisting of $n_\mathcal{T}$ tokens, the forward pass returns a probability distribution over the vocabulary and picks a new token $t_\text{next} \sim \mathbb{N}$. This token is appended to the original input tokens to form a new sequence $[\mathbf{a}_{i-1} \oplus t_\text{next}] \sim \mathbb{N}^{n_\mathcal{T}+1}$ now composed of $n_\mathcal{T}+1$ tokens. This updated sequence can then be fed back as a new input to the model to generate yet one more token. Repeating this process multiple times allows the iterative generation of one token per forward pass resulting in an ever increasing length of input sequence (as each newly predicted token is appended back into a new input sequence for the model).

As the computational complexity of a self-attention head grows quadratically with the number of tokens, this process becomes computationally prohibitive for long sequences. However, due to the autoregressive nature of next-token prediction, self-attention involves a lot of redundant computations across generation steps which can be eliminated via KV (Key Value) caching for optimized inference efficiency.

To see how KV caching works, let us assume that we have already evaluated the self-attention head for a sequence $\mathbf{a}_{i-1}$ of $n_\mathcal{T}$ tokens and kept in memory the tokens' keys feature maps $\mathbf{k}_h^\text{cache} \leftarrow \mathbf{k}_h \sim \mathbb{R}^{n_\mathcal{T} \times d_\rho}$ as well as their values feature maps $\mathbf{v}_h^\text{cache} \leftarrow \mathbf{v}_h \sim \mathbb{R}^{n_\mathcal{T} \times d_h}$. When a new token t_next is appended $[\mathbf{a}_{i-1} \oplus t_\text{next}]$, we only need to evaluate three new feature maps $\mathbf{q}_h(t = t_\text{next}) \sim \mathbf{k}_h(t = t_\text{next}) \sim \mathbb{R}^{d_\rho}$ and $\mathbf{v}_h(t = t_\text{next}) \sim \mathbb{R}^{d_h}$ associated with this token only. Indeed, there is no need to recalculate the other feature maps for any of the other tokens since those features are functions of the individual tokens which are processed independently by fully connected layers (and therefore would end up with again the same features regardless of how many and/or which other tokens are present in the input sequence). Moreover, because of causality in autoregressive generation, we only need to compute the last row $\sim \mathbb{R}^{n_\mathcal{T}+1}$ of the new attention weight matrix $\boldsymbol{\rho}_{([\mathbf{a}_{i-1} \oplus t_\text{next}],\, h)} \sim \mathbb{R}^{(n_\mathcal{T}+1) \times (n_\mathcal{T}+1)}$. The causal mask ensures that the last column (right-most) is always null except for the last row (bottom-most associated with the new token t_next) which is the only one with a complete row of non-zero values, see eq.(13). This means the presence of this new token t_next adds a new row to $\boldsymbol{\rho}_{([\mathbf{a}_{i-1} \oplus t_\text{next}],\, h)}$ but leaves all the other components completely unchanged.

To evaluate this last row of attention weights for t_next, we carry out a vector-matrix product between the new query feature map $\mathbf{q}_h(t = t_\text{next})$ and the (transposed) keys feature maps

$$\boldsymbol{\rho}_{([\mathbf{a}_{i-1} \oplus t_\text{next}],\, h)}(t = t_\text{next}) = \text{softmax}\left(\mathbf{q}_h(t = t_\text{next}) \Big[\mathbf{k}_h^\text{cache} \oplus \mathbf{k}_h(t = t_\text{next}) \Big]^t \right) \sim \mathbb{R}^{n_\mathcal{T}+1}$$

where the new key feature map vector has been appended to the cached feature maps of the other tokens $[\mathbf{k}_h^\text{cache} \oplus \mathbf{k}_h(t = t_\text{next})] \sim \mathbb{R}^{(n_\mathcal{T}+1) \times d_\rho}$ (see above eq.(10) and consider only the last row of queries for a vector-matrix product).

Multi-headed self-attention

Layer	Forward pass	Shape	Backward pass
Input data	$\mathbf{a}_{i-1}$	$\mathbb{R}^{n_\mathcal{T}\times d}$	$\boldsymbol{\Delta}_{i-1} = \sum_{h=1}^{n_h} \boldsymbol{\Delta}_{i-1}^{h}$
Head #1	$\mathbf{a}_i^{(h=1)} = \text{Att}_{(h=1)}(\mathbf{a}_{i-1})$	$\mathbb{R}^{n_\mathcal{T}\times d_h}$	$\boldsymbol{\Delta}_i^{(h=1)} \rightarrow \boldsymbol{\Delta}_{i-1}^{(h=1)} \sim \mathbb{R}^{n_\mathcal{T}\times d}$
⋮		⋮	⋮
Head $\#n_h$	$\mathbf{a}_i^{(h=n_h)} = \text{Att}_{(h=n_h)}(\mathbf{a}_{i-1})$	$\mathbb{R}^{n_\mathcal{T}\times d_h}$	$\boldsymbol{\Delta}_i^{(h=n_h)} \rightarrow \boldsymbol{\Delta}_{i-1}^{(h=n_h)} \sim \mathbb{R}^{n_\mathcal{T}\times d}$
↓		↓ ↑	*each head h is allocated its own* $\boldsymbol{\Delta}_i^h \sim \mathbb{R}^{n_\mathcal{T}\times d_h}$ *sliced out of* $\boldsymbol{\Delta}_i$
Output data	$\mathbf{a}_i = \text{concat}\left[\mathbf{a}_i^{(h=1)}, \cdots\cdots, \mathbf{a}_i^{(h=n_h)}\right]$	$\mathbb{R}^{n_\mathcal{T}\times d}$	$\boldsymbol{\Delta}_i$

The ouptut representation for t_next is then obtained as a linear combination of the cached and new values feature maps $[\mathbf{v}_h^\text{cache} \oplus \mathbf{v}_h(t = t_\text{next})] \sim \mathbb{R}^{(n_\mathcal{T}+1)\times d_h}$ weighted by the attention vector computed above using another vector-matrix product

$$\mathbf{a}_i^h(t = t_\text{next}) = \boldsymbol{\rho}_{([\mathbf{a}_{i-1} \oplus t_\text{next}],\, h)}(t = t_\text{next})\left[\mathbf{v}_h^\text{cache} \oplus \mathbf{v}_h(t = t_\text{next})\right] \sim \mathbb{R}^{(n_\mathcal{T}+1)\times d_h}$$

In most use-cases, $\mathbf{a}_i^h(t = t_\text{next})$ is, by itself, (after downstream processing by more fully-connected layers), enough to predict the probability distribution of token coming after t_next and the KV caching process is henceforth repeated iteratively to keep on generating more tokens.

Crucially, by expressing autoregressive next-token generation as a process that is limited to evaluating vector-matrix expressions of complexity $\mathcal{O}(n_\mathcal{T})$, KV cache makes inference grow linearly with respect to sequence length instead of quadratically as it would be with a naïve recalculation of all attention weights via matrix-matrix products of complexity $\mathcal{O}(n_\mathcal{T}^2)$ (ignoring the complexity scaling on the dimensionality of the feature maps). Obviously, this accelerated inference comes at the cost of significant memory requirements to cache the keys $\mathbf{k}_h^\text{cache} \sim \mathbb{R}^{n_\mathcal{T}\times d_\rho}$ and values feature maps $\mathbf{v}_h^\text{cache}$ whose size is unbounded as they grow linearly with the (potentially unknown in advance) number of tokens thereby creating memory management complications and restricted context windows [16].

(Finally, as long as we define the rows of the attention matrix to represent the attention weights of the each token, we can swap the notation from queries to keys and, in this case, we would instead talk about a QV cache...)

4 Multi-headed attention layer

In Section 3, we saw how a self-attention head h can be seen as a parametrized function $\text{Att}_{\mathcal{P}_h}$ that transforms the d-dimensional input feature maps $\mathbf{a}_{i-1} \sim \mathcal{R}^{n_\mathcal{T}\times d}$ of the $n_\mathcal{T}$ tokens into new d_h-dimensional representations $\mathbf{a}_i^h \sim \mathbb{R}^{n_\mathcal{T}\times d_h}$ (by treating the sequence itself as a collective unit). Each attention head h is associated with its own set of parameters $\mathcal{P}_h$.

In this Section, we focus on multi-headed attention in which, instead of a single attention head, we consider a layer composed of multiple independent attention heads. In a manner somewhat similar to the different filters in the convolution layers of CNNs [1], it may be argued that having multiple parallel heads in attention layers allows the network to learn different aspects of the data simultaneously; see [17] and citations for more discussion. As far as practitioners are concerned, multi-headed attention has become an overwhelming standard.

Forward pass. Let us denote by n_h the number of attention heads in a multi-headed layer of self-attention which takes $\mathbf{a}_{i-1} \sim \mathbb{R}^{n_\mathcal{T} \times d}$ as its input. In order to have an integer number of heads, we choose $n_h = d/d_h \in \mathbb{N}$ enforcing that the dimensionality d_h of the tokens' feature vectors produced by the attention heads be an exact divisor of their input dimensionality d.

Since all attention heads are parametrized by their own set of parameters $\mathcal{P}_1 \neq \cdots \neq \mathcal{P}_{n_h}$, each head operates independently of all the other ones and a multi-headed attention layer is defined as a list of functions

$$\text{MultiAtt} = \left[\text{Att}_{\mathcal{P}_1}, \cdots, \text{Att}_{\mathcal{P}_{n_h}} \right]$$

Applying these functions to the same input sequence $\mathbf{a}_{i-1}$ produces n_h output representations

$$\text{MultiAtt}(\mathbf{a}_{i-1}) = \left[\text{Att}_{\mathcal{P}_1}(\mathbf{a}_{i-1}), \cdots, \text{Att}_{\mathcal{P}_{n_h}}(\mathbf{a}_{i-1}) \right] = \left[\mathbf{a}_i^{(h=1)} \sim \mathbb{R}^{n_\mathcal{T} \times d_h}, \cdots, \mathbf{a}_i^{(h=n_h)} \sim \mathbb{R}^{n_\mathcal{T} \times d_h} \right]$$

where each head h contributes its own $\mathbf{a}_i^h = \text{Att}_{\mathcal{P}_h}(\mathbf{a}_{i-1}) \sim \mathbb{R}^{n_\mathcal{T} \times d_h}$ like in eq.(17). Finally, one concatenates together the feature maps produced by all the heads (i.e. column-wise concatenation) to yield the consolidated output $\mathbf{a}_i \sim \mathbb{R}^{n_\mathcal{T} \times d}$ where we recover the expected $d = n_h \times d_h$. In summary

Multi-headed self-attention: forward pass

$$\mathbf{a}_i = \text{concat} \left[\mathbf{a}_i^{(h=1)}, \cdots, \mathbf{a}_i^{(h=n_h)} \right] \tag{34}$$

Because of the choice $n_h = d/d_h \in \mathbb{N}$, after passing through a multi-headed attention layer each token is represented by a new d-dimensional vector which is made up of n_h different d_h-dimensional feature maps produced by all the self-attention heads. This constraint on the values of the (n_h, d_h, d) tuple ensures that the dimensionality of the output tokens' feature maps is the same as that of the input feature maps, i.e. $\mathbf{a}_{i-1} \sim \mathbf{a}_i \sim \mathbb{R}^{n_\mathcal{T} \times d}$. This way, one may easily compose multiple multi-headed attention layers together. One benefit of stacking multiple attention layers is that, even though an individual self-attention head involves only pairwise token-to-token interactions, composing multiple such layers effectively introduces higher-level interactions which eventually span the entire sequence of length $n_\mathcal{T}$. This point was discussed in detail for a single head of self-attention and remains equally valid for multi-headed attention.

Backward pass. The first step consists in reversing the concatenation operation carried out in the last step of the forward pass by slicing out the upstream error signal $\mathbf{\Delta}_i \sim \mathbb{R}^{n_\mathcal{T} \times d}$ column-wise into n_h sub-components

$$\mathbf{\Delta}_i \sim \mathbb{R}^{n_\mathcal{T} \times d} \longrightarrow \left[\mathbf{\Delta}_i^{(h=1)} \sim \mathbb{R}^{n_\mathcal{T} \times d_h}, \cdots, \mathbf{\Delta}_i^{(h=n_h)} \sim \mathbb{R}^{n_\mathcal{T} \times d_h} \right]$$

where each $\mathbf{\Delta}_i^h \sim \mathbb{R}^{n_\mathcal{T} \times d_h}$ is allocated to a specific head h. At this point, we can use simply eq.(33) to propagate this error signal back through h to get the downstream error signal $\mathbf{\Delta}_{i-1}^h \sim \mathbb{R}^{n_\mathcal{T} \times d}$ which, as required, recovers the dimensionality of the input sequence. Doing the same to all n_h attention heads, leads the complete downstream error signal through a multi-headed attention layer as

Multi-headed self-attention: backward pass

$$\mathbf{\Delta}_{i-1} = \sum_{h=1}^{n_h} \mathbf{\Delta}_{i-1}^h \quad \sim \mathbb{R}^{n_\mathcal{T} \times d} \tag{35}$$

5 Layer normalization

Typically, neural network architectures designed for datasets with an inherent sequential nature favor layer normalization [18] – `LN` over batch normalization [19] – `BN` for the purpose of training stabilization. While the original motivation for layer normalization came from its observed empirical superiority in recurrent architectures, it remains preferred even in transformer-based models. As layer normalization treats all tokens (referred to as samples in `BN`) independently, it is able to gracefully handle variable-length sequences without being affected by cross-token/sample statistics.

Forward pass. As a reminder, the input data $\mathbf{a}_{i-1} \sim \mathbb{R}^{n_\mathcal{T} \times d}$ represents the d-dimensional feature vectors associated with each one of the $n_\mathcal{T}$ tokens in a sequence. We separate the forward pass into two distinct steps:

1. Normalization. Considering a specific token $t^\star$, the statistical distribution of its feature vector $\mathbf{a}_{i-1}(t = t^\star) = [a^1_{i-1}(t^\star), \cdots, a^d_{i-1}(t^\star)] \sim \mathbb{R}^d$ can be summarised by its first two moments

$$\mu_{t^\star} = \frac{1}{d} \sum_{f=1}^{d} a^f_{i-1}(t = t^\star) \sim \mathbb{R} \quad \text{and} \quad \sigma_{t^\star} = \sqrt{\frac{1}{d} \sum_{f=1}^{d} \left(a^f_{i-1}(t = t^\star) - \mu_{t^\star} \right)^2} \sim \mathbb{R}$$

Once the mean $\mu_{t^\star}$ and standard deviation $\sigma_{t^\star}$ have been evaluated, those summary statistics are used to produce a normalized feature vector $\overline{\mathbf{a}}_{i-1}(t = t^\star)$ which is specific to this token via

$$\overline{\mathbf{a}}_{i-1}(t = t^\star) = \frac{\mathbf{a}_{i-1}(t = t^\star) - \mu_{t^\star}}{\sigma_{t^\star}} \sim \mathbb{R}^d$$

where $\mu_{t^\star}$ and $\sigma_{t^\star}$ are both broadcast vector-wise such that $\overline{\mathbf{a}}_{i-1}(t = t^\star)$ is well-defined and normalized with its own token-specific values. Obviously, the same feature-wise normalization may be applied independently to all tokens yielding vectors $(\boldsymbol{\mu}, \boldsymbol{\sigma})_{\texttt{LN}} \sim \mathbb{R}^{n_\mathcal{T}}$ of mean values and standard deviations which are used to normalize the feature vectors of each token from $\mathbf{a}_{i-1}$ to

$$\boxed{\overline{\mathbf{a}}_{i-1} = \text{diag}\left(1/\boldsymbol{\sigma}\right) \left(\mathbf{a}_{i-1} - \widetilde{\boldsymbol{\mu}}\right) \sim \mathbb{R}^{n_\mathcal{T} \times d}} \tag{36}$$

where the vector of mean values is column-wise broadcast $\boldsymbol{\mu} \sim \mathbb{R}^{n_\mathcal{T}} \to \widetilde{\boldsymbol{\mu}} \sim \mathbb{R}^{n_\mathcal{T} \times d}$ and the vector of standard deviations is lifted into a diagonal representation $\text{diag}(1/\boldsymbol{\sigma}) \sim \mathbb{R}^{n_\mathcal{T} \times n_\mathcal{T}}$ to reproduce the proper token normalization shown above.

2. Learnable affine transformation. Next we apply an affine transformation by introducing two vectors $\{\mathbf{w}_{i-1} \sim \mathbb{R}^d, \mathbf{b}_{i-1} \sim \mathbb{R}^d\}$. Taking token $t^\star$ as an example, we wish for its normalized feature vector $\overline{\mathbf{a}}_{i-1}(t = t^\star)$ to be transformed into

$$\mathbf{a}_i(t = t^\star) = \overline{\mathbf{a}}_{i-1}(t = t^\star) \circ \mathbf{w}_{i-1} + \mathbf{b}_{i-1} \sim \mathbb{R}^d$$

where the components of the weights $\mathbf{w}_{i-1}$ and biases $\mathbf{b}_{i-1}$ are learned during training. Applying the same transformation to all tokens may be achieved by

$$\boxed{\mathbf{a}_i = \overline{\mathbf{a}}_{i-1} \, \text{diag}\left(\mathbf{w}_{i-1}\right) + \widetilde{\mathbf{b}}_{i-1}} \tag{37}$$

where the bias vector is broadcast row-wise $\mathbf{b}_{i-1} \sim \mathbb{R}^d \to \widetilde{\mathbf{b}}_{i-1} \sim \mathbb{R}^{n_\mathcal{T} \times d}$. Lifting the components of $\mathbf{w}_{i-1} \sim \mathbb{R}^d$ into a diagonal $\text{diag}(\mathbf{w}_{i-1}) \sim \mathbb{R}^{d \times d}$ ensures that each feature $f \in [1, \cdots, d]$ of the normalized $\overline{\mathbf{a}}_{i-1}$ is associated with its designated weight value from $\mathbf{w}_{i-1}$.

In summary, the forward pass of the layer normalization can be expressed as

Layer normalization: forward pass

$$\boldsymbol{\mathbf{a}_i = \overline{\mathbf{a}}_{i-1} \, \widetilde{\mathbf{w}}_{i-1} + \widetilde{\mathbf{b}}_{i-1}} \quad \text{with} \quad \boldsymbol{\overline{\mathbf{a}}_{i-1} = \frac{\mathbf{a}_{i-1} - \widetilde{\boldsymbol{\mu}}}{\widetilde{\boldsymbol{\sigma}}}} \tag{38}$$

where the broadcasting rules (using diagonal matrices) of the normalization step with $(\boldsymbol{\mu}, \boldsymbol{\sigma})_{\texttt{LN}}$ and those of the learnable affine transformation step with $(\mathbf{w}_{i-1}, \mathbf{b}_{i-1})$ can be understood by identification with eq.(36) and eq.(37).

At this point, it is instructive to refer to Section 9 of the reference paper [1] dedicated to batch normalization. Indeed, although we have made the current eq.(38) for layer normalization look identical to eq.(39) of the reference paper for batch normalization, there is a subtle but important difference in the way that the normalized feature vectors $\overline{\mathbf{a}}_{i-1}$ are defined

- In the case of batch normalization, the mean and standard deviation used for the normalization step are evaluated across the different samples (i.e. tokens in the current context) leading to summary statistics vectors $(\boldsymbol{\mu}\,,\boldsymbol{\sigma})_{\texttt{BN}} \sim \mathbb{R}^d$ that have the same dimensionality as the feature space (i.e. the number $n_{\mathcal{T}}$ of samples/tokens is contracted out).
- On the contrary, in the case of layer normalization, these vectors are evaluated across the feature dimension so that each token has its own summary statistics leading to $(\boldsymbol{\mu}\,,\boldsymbol{\sigma})_{\texttt{LN}} \sim \mathbb{R}^{n_{\mathcal{T}}}$ (i.e. the dimensionality d of the feature vectors is contracted out).

This difference in the way that the normalization vectors $(\boldsymbol{\mu}\,,\boldsymbol{\sigma})$ are evaluated carries over to the broadcasting rules with the row-wise broadcast of $\boldsymbol{\mu}$ for BN being replaced by column-wise broadcast for LN. Similarly the broadcasted division $1/\widetilde{\boldsymbol{\sigma}}$ is evaluated via marix multiplication from the right for BN whereas it is from the left for LN. Therefore, the crucial observation is that one can go from LN to BN and recover all these shape/statistics differences simply by applying the normalization part of BN to the transpose of our current input data $\mathbf{a}_{i-1} \sim \mathbb{R}^{n_{\mathcal{T}} \times d}$ with

$$\texttt{LN}\left(\mathbf{a}_{i-1}\right) \sim \mathbb{R}^{n_{\mathcal{T}} \times d} \;\cong\; \left[\texttt{BN}\left(\mathbf{a}_{i-1}^t \sim \mathbb{R}^{d \times n_{\mathcal{T}}}\right) \sim \mathbb{R}^{d \times n_{\mathcal{T}}}\right]^t \tag{39}$$

where we use the $\cong$ symbol to reflect the fact that the number of parameters in the learnable affine transformation step is different since BN needs to be applied to the transpose of $\mathbf{a}_{i-1}$ and that, generally, $n_{\mathcal{T}} \neq d$.

In other words, the LN and BN layers are both composed of two steps i) a normalization for which both layers are exact mirrors of each other **up to a transpose operation** followed by ii) a mechanically identical learnable affine transformation.

Backward pass. Thanks to this "transposed duality" between LN and BN, we can immediately adapt the results of the backward pass derived in eqs.(41-43) of the reference paper [1] for batch normalization (there) to layer normalization (here) by applying the appropriate transpose operations.

In particular, since the second step 2) relating to the learnable affine transformation does not depend upon the details of how $\overline{\mathbf{a}}_{i-1}$ is evaluated, the gradients of the loss with respect to the weights and biases $\partial\mathcal{L}_{\text{seq}}/\{\mathbf{w}_{i-1}, \mathbf{b}_{i-1}\} \sim \mathbb{R}^d$ remain unchanged for both BN and LN layers and we simply rename "samples" to "tokens" to better match the current context of sequence models.

On the other hand, the backpropagation of the error signal from its upstream value $\boldsymbol{\Delta}_i \sim \mathbb{R}^{n_{\mathcal{T}} \times d}$ to the downstream $\boldsymbol{\Delta}_{i-1} \sim \mathbb{R}^{n_{\mathcal{T}} \times d}$ requires to handle the different normalizations of $\overline{\mathbf{a}}_{i-1}$ of step 1) which, as explained in (39), are related to each other via a simple transposition. Carrying over this mapping from BN to LN is done by

1. copy/pasting the expression of $\boldsymbol{\Delta}_{i-1}$ as it appears in the backward pass of LN in eq.(41) of the reference paper [1]
2. replacing both $\overline{\mathbf{a}}_{i-1}$ and $\boldsymbol{\Delta}_i$ by their transpose while ensuring consistent dimensionality of the matrix multiplications. In other words $\left(\boldsymbol{\Delta}_i \widetilde{\mathbf{w}}_{i-1}\right)_{\texttt{BN}} \rightarrow \left(\widetilde{\mathbf{w}}_{i-1} \boldsymbol{\Delta}_i^t\right)_{\texttt{LN}} \sim \mathbb{R}^{d \times n_{\mathcal{T}}}$
3. replacing sums over samples in BN by sums over features for LN
4. performing the outer transpose as shown in (39) to recover the expected dimensionality of the downstream error signal $\boldsymbol{\Delta}_{i-1} \sim \mathbb{R}^{n_{\mathcal{T}} \times d}$
5. bringing the broadcasted division $1/\widetilde{\boldsymbol{\sigma}}$ out of the outer transpose so this term continues to appear as applied from the left. (Since the transpose of a diagonal matrix is equal to itself, there is no need for additional transpose symbols here).
6. finally, modifying the scaling to $1/d$ to correctly reflect the fact that normalization is carried out feature-wise in LN as opposed to sample-wise in BN.

In summary, we have

Layer normalization: backward pass

$$\mathbf{\Delta}_{i-1} = \frac{1}{d\,\widetilde{\boldsymbol{\sigma}}} \left(d\,\widetilde{\mathbf{w}}_{i-1} \mathbf{\Delta}_i^t - \sum_{\text{features}} \widetilde{\mathbf{w}}_{i-1} \mathbf{\Delta}_i^t - \overline{\mathbf{a}}_{i-1}^t \circ \sum_{\text{features}} \overline{\mathbf{a}}_{i-1}^t \circ \widetilde{\mathbf{w}}_{i-1} \mathbf{\Delta}_i^t \right)^t \sim \mathbb{R}^{n_{\mathcal{T}} \times d} \tag{40}$$

$$\frac{\partial \mathcal{L}_{\text{seq}}}{\partial \mathbf{w}_{i-1}} = \text{diag}\left(\overline{\mathbf{a}}_{i-1}^t \mathbf{\Delta}_i \right) \sim \mathbb{R}^d \tag{41}$$

$$\frac{\partial \mathcal{L}_{\text{seq}}}{\partial \mathbf{b}_{i-1}} = \sum_{\text{tokens}} \mathbf{\Delta}_i \sim \mathbb{R}^d \tag{42}$$

6 LoRA Layer

Forward pass. Typically, neural networks are composed of numerous fully-connected layers whose purpose is to modify the dimensionality of the feature maps. Normally, going from an input feature map $\mathbf{a}_{i-1} \sim \mathbb{R}^{n \times f_{i-1}}$ to an output representation $\mathbf{a}_i \sim \mathbb{R}^{n \times f_i}$ would require $(f_{i-1} \times f_i)$ parameters encoded into a weight matrix $\mathbf{w}_{i-1} \sim \mathbb{R}^{f_{i-1} \times f_i}$ (and maybe even another f_i parameters if one considers non-null biases $\mathbf{b}_{i-1} \sim \mathbb{R}^{f_i}$ in addition to the weight matrix). In the context of this paper $n \equiv n_{\mathcal{T}}$ refers to the number of tokens in the sequence whereas in the reference paper [1] it was referring to the number of samples in a mini-batch. (Regardless of the tokens/samples interpretation, all components are processed independently of each other so there is no distinction to be made as far as fully-connected layers are concerned.)

Dimensionality-wise, the same mapping from f_{i-1} to f_i may be achieved by decomposing the weight matrix into the product of two new matrices $\mathbf{d}_{i-1} \sim \mathbb{R}^{f_{i-1} \times r}$ (mapping from f_{i-1} "down" to r) and $\mathbf{u}_{i-1} \sim \mathbb{R}^{r \times f_i}$ (mapping from r back "up" to f_i). The product $\mathbf{d}_{i-1}\,\mathbf{u}_{i-1} \sim \mathbb{R}^{f_{i-1} \times f_i}$ that composes these two mappings is of the same dimensionality as that of the original weight matrix $\mathbf{w}_{i-1}$ in fully-connected layers. The trick is to choose a rank r such that $r \ll \min(f_{i-1}, f_i)$. In this case, the number of parameters associated with this low-rank decomposition $\mathbf{d}_{i-1}\,\mathbf{u}_{i-1}$ is therefore

$$r \times (f_{i-1} + f_i) \ll (f_{i-1} \times f_i)$$

Normally, LoRA layers would not be used as a substitute to linear layers but more as companions for parameter-efficient fine-tuning [20]. In practice, this means that the full linear layers are first trained to produce a large pre-trained model. Then, during fine-tuning, those weights are kept frozen and LoRA layers are introduced to receive gradient updates specific to the fine-tuning task. Since the LoRA layers and the original dense linear layers both have the same dimensionalities, the data representations are simply added together at inference time.

In summary, the parameters associated with this LoRA layer are

$$\mathcal{P}_{i-1} \begin{cases} \mathbf{d}_{i-1} \sim \mathbb{R}^{f_{i-1} \times r} \\ \mathbf{u}_{i-1} \sim \mathbb{R}^{r \times f_i} \end{cases}$$

and the forward pass can be summarized as

LoRA: forward pass

$$\mathbf{a}_i = \alpha\, \mathbf{a}_{i-1}\, \mathbf{d}_{i-1}\, \mathbf{u}_{i-1} \tag{43}$$

where $\alpha \sim \mathbb{R}$ controls the relative importance of LoRA layers during backpropagation (somewhat analogously to a layer-specific learning rate) and is (usually) chosen such that $\alpha = r$.

Backward pass. The backward pass is evaluated via the usual recursive expression and here it is useful to leverage the cyclic property of Frobenius products to expand

$$
\begin{aligned}
\mathbf{\Delta}_i \cdot \mathrm{d}\mathbf{a}_i &= \mathbf{\Delta}_i \cdot \mathrm{d}\big(\alpha\, \mathbf{a}_{i-1}\, \mathbf{d}_{i-1}\, \mathbf{u}_{i-1}\big) \\
&= \alpha\, \mathbf{\Delta}_i \cdot \mathrm{d}\big(\mathbf{a}_{i-1}\, \mathbf{d}_{i-1}\, \mathbf{u}_{i-1}\big) \\
&= \alpha\, \mathbf{\Delta}_i \cdot \Big[\big(\mathrm{d}\mathbf{a}_{i-1}\big)\, \mathbf{d}_{i-1}\, \mathbf{u}_{i-1} + \mathbf{a}_{i-1}\big(\mathrm{d}\mathbf{d}_{i-1}\big)\mathbf{u}_{i-1} + \mathbf{a}_{i-1}\, \mathbf{d}_{i-1}\big(\mathrm{d}\mathbf{u}_{i-1}\big)\Big] \\
&= \underbrace{\alpha\big[\mathbf{\Delta}_i\big(\mathbf{d}_{i-1}\mathbf{u}_{i-1}\big)^t\big]}_{\mathbf{\Delta}_{i-1}} \cdot \mathrm{d}\mathbf{a}_{i-1} + \underbrace{\alpha\big(\mathbf{a}^t_{i-1}\mathbf{\Delta}_i\mathbf{u}^t_{i-1}\big)}_{\frac{\partial \mathcal{L}_{\text{seq}}}{\partial \mathbf{d}_{i-1}}} \cdot \mathrm{d}\mathbf{d}_{i-1} + \underbrace{\alpha\big[\big(\mathbf{a}_{i-1}\mathbf{d}_{i-1}\big)^t\mathbf{\Delta}_i\big]}_{\frac{\partial \mathcal{L}_{\text{seq}}}{\partial \mathbf{u}_{i-1}}} \cdot \mathrm{d}\mathbf{u}_{i-1}
\end{aligned}
$$

In summary, the backward pass through a LoRA layer is given by:

LoRA: backward pass

$$\mathbf{\Delta}_{i-1} = \alpha\, \mathbf{\Delta}_i\big(\mathbf{d}_{i-1}\, \mathbf{u}_{i-1}\big)^t \qquad \sim \mathbb{R}^{n\times f_{i-1}} \tag{44}$$

$$\frac{\partial \mathcal{L}_{\text{seq}}}{\partial \mathbf{d}_{i-1}} = \alpha\, \mathbf{a}^t_{i-1}\mathbf{\Delta}_i\mathbf{u}^t_{i-1} \qquad \sim \mathbb{R}^{f_{i-1}\times r} \tag{45}$$

$$\frac{\partial \mathcal{L}_{\text{seq}}}{\partial \mathbf{u}_{i-1}} = \alpha\, \big(\mathbf{a}_{i-1}\mathbf{d}_{i-1}\big)^t\mathbf{\Delta}_i \qquad \sim \mathbb{R}^{r\times f_i} \tag{46}$$

We can see that α acts as a multiplicative scaling factor to influence the gradient updates in a way similar to learning rate scaling (although acting specifically on LoRA layers only).

7 Conclusion: A minimalistic transformer-based architecture

Minimalistic architecture. Released by OpenAI in 2019, GPT-2 may still be used as a reference to illustrate transformer-based networks. In this note we reproduce a smaller version of this architecture as specified in Table 2. Complete expressions for all parameter gradients are provided in Table 3.

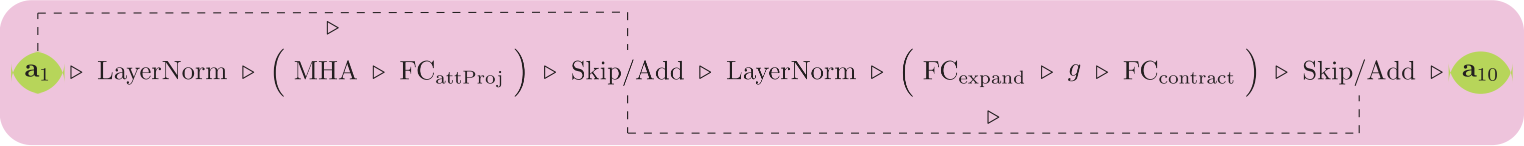
After a tokenizer has already processed the input sequence, the resulting input tokens $\mathbf{a}_0 \sim \mathbb{N}^{n_\mathcal{T}}$ are transformed into token and position embeddings $\mathbf{a}^{\text{tok}}_1 \sim \mathbf{a}^{\text{pos}}_1 \sim \mathbb{R}^{n_\mathcal{T}\times d}$ of the same dimensionality via weight matrices $\mathbf{w}_{\text{tok}} \sim \mathbb{R}^{n_{\text{vocab}}\times d}$ and $\mathbf{w}_{\text{pos}} \sim \mathbb{R}^{n_{\text{context}}\times d}$. In keeping with the "pedestrian" spirit of this note, we follow the "small" version of GPT-2 with

$$\Big(d = 768 \;\; ; n_{\text{context}} = 1,024 \;\; ; n_{\text{vocab}} = 50,257\Big)$$

Other versions of GPT-2 differ only in the values of these parameters without any modification to the architecture itself. Both embedding representations are added to each other $\mathbf{a}_1 = \mathbf{a}^{\text{tok}}_1 + \mathbf{a}^{\text{pos}}_1$ and serve as input to the transformer block. Generically, a transformer block is composed of two functional sublayers each wrapped in a $\big($LayerNorm $\triangleright$ Sublayer $\triangleright$ Skip/Add $\big)$ pattern. Those sublayers consist of

- "Self-attention" sublayer $\equiv \big(\text{MHA} \triangleright \text{FC}_{\text{attProj}}\big)$. For the sake of clarity we separate the pure MHA part described in Section 4 from its final output projection $\text{FC}_{\text{attProj}}$. (Standard implementations of self-attention typically keep both steps as a single integrated layer.)
- "Expand-and-contract" sublayer $\equiv \big(\text{FC}_{\text{expand}} \triangleright g \triangleright \text{FC}_{\text{contract}}\big)$. The first fully-connected layer expands the dimensionality of the feature maps from d to $4d$ and the second one contracts it back to d after having gone through a non-linear activation function g such as ReLU, GELU...

Denoting by $\triangleright$ the "left-to-right" (forward) function composition operator, the architecture of a complete transformer block is summarized visually in the diagram below with $\mathbf{a}_1 \sim \mathbb{R}^{n_\mathcal{T}\times d}$ as the input data and $\mathbf{a}_{10} \sim \mathbb{R}^{n_\mathcal{T}\times d}$ as the output data representation after processing by the transformer block.



$\mathbf{a}_1$ ▷ LayerNorm ▷ (MHA ▷ FC$_{\text{attProj}}$) ▷ Skip/Add ▷ LayerNorm ▷ (FC$_{\text{expand}}$ ▷ g ▷ FC$_{\text{contract}}$) ▷ Skip/Add ▷ $\mathbf{a}_{10}$
▷
▷

The step-by-step breakdown from $\mathbf{a}_1$ to $\mathbf{a}_{10}$ is presented in Table 2 where the layers belonging to the transformer block are color highlighted . This construction of transformer blocks as two functional sublayers may also be visualized as

$$\mathbf{a}_5 = \mathbf{a}_1 + \mathrm{FC}_{\mathrm{attProj}} \left[\mathrm{MHA} \left(\mathrm{LayerNorm}\, \mathbf{a}_1 \right) \right]$$

$$\mathbf{a}_{10} = \mathbf{a}_5 + \mathrm{FC}_{\mathrm{contract}} \left(g \left[\mathrm{FC}_{\mathrm{expand}} \left(\mathrm{LayerNorm}\, \mathbf{a}_5 \right) \right] \right)$$

Instead of feeding the ouptut $\mathbf{a}_{10}$ of the transformer block back as an input into another transformer block (with $n_{\mathrm{blocks}} = 12$ back-to-back blocks in "small" GPT-2), we pass $\mathbf{a}_{10}$ directly into a final set of layer normalization and fully-connected layer $\mathrm{FC}_{\mathrm{logits}}$ to produce the "logits" $\mathbf{a} \sim \mathbb{R}^{n_{\mathcal{T}} \times n_{\mathrm{vocab}}}$ which are ultimately converted, via a Softmax function, into $n_{\mathcal{T}}$ probability distributions $\mathbf{y}_{\mathrm{pred}} \sim \mathbb{R}^{n_{\mathcal{T}} \times n_{\mathrm{vocab}}}$ over the vocabulary for all tokens in the sequence. These last steps are summarized as

$$\mathbf{y}_{\mathrm{pred}} = \mathbf{a}_{10} \;\triangleright\; \mathrm{LayerNorm} \;\triangleright\; \mathrm{FC}_{\mathrm{logits}} \;\triangleright\; \mathrm{Softmax}$$

The complete architecture of our minimalistic GPT-2 version with a single $n_{\mathrm{blocks}} = 1$ transformer block is presented in Table 2.

How many parameters does the model have? Overall, the number of parameters in a transformer block is given by

$$n_{\mathrm{params}}^{(\mathrm{block})} = \Bigg[\underbrace{(2 \times d)}_{\mathrm{LN}(1)} + \underbrace{(3+1) \times \big[(d \times d) + d \big]}_{\mathrm{MHA} + \mathrm{FC}_{\mathrm{attProj}}} + \underbrace{(2 \times d)}_{\mathrm{LN}(2)} + \underbrace{\big[(d \times 4d) + 4d \big]}_{\mathrm{FC}_{\mathrm{expand}}} + \underbrace{\big[(4d \times d) + d \big]}_{\mathrm{FC}_{\mathrm{contract}}} \Bigg]$$

Using the standard GPT-2 values, each transformer block therefore contains $n_{\mathrm{params}}^{(\mathrm{block})} = 7{,}087{,}872$ parameters. Adding on the parameters associated with token/position embeddings, final layer normalization and fully-connected layer (to produce the logits), we end up with a total number of parameters given by

$$n_{\mathrm{params}} = \Bigg[\underbrace{(n_{\mathrm{vobab}} \times d)}_{\text{token emdedding}} + \underbrace{(n_{\mathrm{context}} \times d)}_{\text{position emdedding}} + \underbrace{(n_{\mathrm{blocks}} \times n_{\mathrm{params}}^{(\mathrm{block})})}_{\text{transformer blocks}} + \underbrace{(2 \times d)}_{\mathrm{LN(final)}} + \underbrace{\big[(d \times n_{\mathrm{vocab}}) + n_{\mathrm{vocab}} \big]}_{\mathrm{FC}_{\mathrm{logits}}} \Bigg]$$

where n_{blocks} denotes the number of transformer blocks.

In the minimalistic network specified in Table 2, we have a single transformer block $n_{\mathrm{blocks}} = 1$ for a total of $n_{\mathrm{params}} = 85{,}120{,}849$ parameters. With $n_{\mathrm{blocks}} = 12$, the complete GPT-2 network has a total of $n_{\mathrm{params}}^{(\mathrm{gpt2})} = 163{,}087{,}441$ parameters.

Note that this is only about twice the number of parameters compared to our minimalistic network even though there are 12 tranformer blocks instead of a single one.

In practice, it is common to tie the weights of the token embedding layer $\mathbf{w}_{\mathrm{tok}} \sim \mathbb{R}^{n_{\mathrm{vocab}} \times d}$ together with those of the final fully-connected layer $\mathrm{FC}_{\mathrm{logits}} \sim \mathbb{R}^{d \times n_{\mathrm{vocab}}}$ since those have the same dimensionality (up to a transpose) and account for a large number of parameters $n_{\mathrm{vocab}} \times d \approx 39{,}000{,}000$. In this "weight-tying" scenario, one simply ignores the biases from $\mathrm{FC}_{\mathrm{logits}}$ and, instead of learning two independent weight matrices, the model learns only one matrix. This optimization reduces the number of parameters from $\approx 163{,}000{,}000$ down to $\approx 124{,}000{,}000$ leading not only to $\approx 24\%$ savings in parameter count but may also act as a mild regularizer that enforces consistency between input and output representations.

With LoRA: How many parameters now? As an illustration of LoRA fine-tuning, let us replace the last fully-connected layer in Table 2 by a LoRA layer. In this case, the forward pass is given by

$$\mathbf{a} \equiv \mathbf{a}_{12} = \mathtt{FC}_{\mathtt{frozen}}(\mathbf{a}_{11}) + \mathtt{LoRA}(\mathbf{a}_{11}) = \mathtt{FC}_{\mathtt{frozen}}(\mathbf{a}_{11}) + \alpha\, \mathbf{a}_{11}\, \mathbf{d}_{11}\, \mathbf{u}_{11}$$

where $\texttt{FC}_{\text{frozen}}$ indicates that the weights of the fully-connected layer are frozen and will not be updated during the backward pass

$$\boldsymbol{\Delta}_{11} = \alpha\, \boldsymbol{\Delta}_{12} \big(\mathbf{d}_{11}\, \mathbf{u}_{11}\big)^t \quad ; \quad \frac{\partial \mathcal{L}_{\text{seq}}}{\partial \mathbf{d}_{11}} = \alpha\, \mathbf{a}_{11}^t \boldsymbol{\Delta}_{12} \mathbf{u}_{11}^t \quad ; \quad \frac{\partial \mathcal{L}_{\text{seq}}}{\partial \mathbf{u}_{11}} = \alpha \left(\mathbf{a}_{11} \mathbf{d}_{11}\right)^t \boldsymbol{\Delta}_{12}$$

Instead of having a fully-connected layer with $(d \times n_{\text{vocab}}) + n_{\text{vocab}} = 38,647,633$ parameters that need to be optimized in the backward pass, using the LoRA layer reduces the number of trainable parameters down to $r \times (d + n_{\text{vocab}}) = 816,400$ which is about $\approx 2\%$ of the original amount (using a standard rank of $r = 16$).

$$n_{\text{params}}^{(\text{lora})} = n_{\text{blocks}} \times \Big[\underbrace{(3+1) \times \big[r \times (d+d)\big]}_{\text{MHA}_{(\text{LoRA})} + \text{LoRA}_{\text{attProj}}} + \underbrace{\big[r \times (d + 4d)\big]}_{\text{LoRA}_{\text{expand}}} + \underbrace{\big[r \times (4d + d)\big]}_{\text{LoRA}_{\text{contract}}} \Big] + \underbrace{\big[r \times (d + n_{\text{vocab}})\big]}_{\text{LoRA}_{\text{logits}}}$$

With $n_{\text{blocks}} = 12$, we get $n_{\text{params}}^{(\text{lora})} = 3,470,608$ to be compared with $n_{\text{params}}^{(\text{gpt2})} = 163,087,441$ for the complete GPT-2 (without weight-tying) which is consistent with a $\approx$ 98% reduction in number of parameters.

Layer	Forward pass	Shape	Backward pass
Input data	$\mathbf{a}_0$	$\mathbb{N}^{n_\mathcal{T}}$	Sequence of $n_\mathcal{T}$ tokens encoded as integers $t \sim \mathbb{N}$
Token embedding	$\mathbf{a}_1^{\text{tok}} = \text{ohe}(\mathbf{a}_0)\,\mathbf{w}_{\text{tok}}$	$\mathbb{R}^{n_\mathcal{T} \times d}$	
Position embedding	$\mathbf{a}_1^{\text{pos}} = \text{ohe}(1\!:\!n_\mathcal{T})\,\mathbf{w}_{\text{pos}}$	$\mathbb{R}^{n_\mathcal{T} \times d}$	
Input embedding	$\mathbf{a}_1 = \mathbf{a}_1^{\text{tok}} + \mathbf{a}_1^{\text{pos}}$	$\mathbb{R}^{n_\mathcal{T} \times d}$	$\boldsymbol{\Delta}_1 = \boldsymbol{\Delta}_5 + \frac{1}{d\,\widetilde{\boldsymbol{\sigma}_1}} \left[\left(d\,\widetilde{\mathbf{w}}_1 \boldsymbol{\Delta}_2^t - \sum \widetilde{\mathbf{w}}_1 \boldsymbol{\Delta}_2^t - \overline{\mathbf{a}}_1^t \circ \sum \overline{\mathbf{a}}_1^t \circ \widetilde{\mathbf{w}}_1 \boldsymbol{\Delta}_2^t \right)^t \right]$
⋆ 1st LayerNorm	$\mathbf{a}_2 = \overline{\mathbf{a}}_1 \,\text{diag}\,(\mathbf{w}_1) + \widetilde{\mathbf{b}}_1$	$\mathbb{R}^{n_\mathcal{T} \times d}$	$\boldsymbol{\Delta}_2 = \sum_h \left(\boldsymbol{\rho}^t_{(\mathbf{a}_2,\,h)} \boldsymbol{\Delta}_3^h \mathbf{w}^t_{v_h} + \boldsymbol{\Delta}^h_{3\,(\text{raw})}\, \mathbf{k}_h \mathbf{w}^t_{q_h} + (\boldsymbol{\Delta}^h_{3\,(\text{raw})})^t\, \mathbf{q}_h\, \mathbf{w}^t_{k_h} \right)$
⋆ Multi-headed attention	$\mathbf{a}_3 = \texttt{MHA}\,\mathbf{a}_2$	$\mathbb{R}^{n_\mathcal{T} \times d_h}$ $\mathbb{R}^{n_\mathcal{T} \times d}$	$\boldsymbol{\Delta}_3$ split into different heads → $\left[\boldsymbol{\Delta}_3^{(h=1)}, \cdots, \boldsymbol{\Delta}_3^{(h=n_h)} \right]$ $[(\mathbf{w}_v, \mathbf{w}_q, \mathbf{w}_k)_h\,,\ (\boldsymbol{\rho}_{\mathbf{a}_2}, \mathbf{k}, \mathbf{q})_h] \sim$ internal to $\mathbf{a}_3$ in MHA forward $\boldsymbol{\Delta}_3 = \boldsymbol{\Delta}_5\, \mathbf{w}_3^t$
⋆ $\text{FC}_{\text{attProj}}$	$\mathbf{a}_4 = \mathbf{a}_3 \mathbf{w}_3 + \widetilde{\mathbf{b}}_3$	$\mathbb{R}^{n_\mathcal{T} \times d}$	$\boldsymbol{\Delta}_5$
⋆ 1st Skip/Add	$\mathbf{a}_5 = \mathbf{a}_1 + \mathbf{a}_4$	$\mathbb{R}^{n_\mathcal{T} \times d}$	$\boldsymbol{\Delta}_5 = \boldsymbol{\Delta}_{10} + \frac{1}{d\,\widetilde{\boldsymbol{\sigma}_5}} \left[\left(d\,\widetilde{\mathbf{w}}_5 \boldsymbol{\Delta}_6^t - \sum \widetilde{\mathbf{w}}_5 \boldsymbol{\Delta}_6^t - \overline{\mathbf{a}}_5^t \circ \sum \overline{\mathbf{a}}_5^t \circ \widetilde{\mathbf{w}}_5 \boldsymbol{\Delta}_6^t \right)^t \right]$ $\boldsymbol{\Delta}_5$ → "dispatched" to $\mathbf{a}_1$ and $\mathbf{a}_4$
⋆ 2nd LayerNorm	$\mathbf{a}_6 = \overline{\mathbf{a}}_5 \,\text{diag}\,(\mathbf{w}_5) + \widetilde{\mathbf{b}}_5$	$\mathbb{R}^{n_\mathcal{T} \times d}$	$\boldsymbol{\Delta}_6 = \boldsymbol{\Delta}_7\, \mathbf{w}_6^t$
⋆ $\text{FC}_{\text{expand}}$	$\mathbf{a}_7 = \mathbf{a}_6 \mathbf{w}_6 + \widetilde{\mathbf{b}}_6$	$\mathbb{R}^{n_\mathcal{T} \times 4d}$	$\boldsymbol{\Delta}_7 = \boldsymbol{\Delta}_8 \circ g'(\mathbf{a}_7)$
⋆ Activation	$\mathbf{a}_8 = g(\mathbf{a}_7)$	$\mathbb{R}^{n_\mathcal{T} \times 4d}$	$\boldsymbol{\Delta}_8 = \boldsymbol{\Delta}_{10}\, \mathbf{w}_8^t$
⋆ $\text{FC}_{\text{contract}}$	$\mathbf{a}_9 = \mathbf{a}_8 \mathbf{w}_8 + \widetilde{\mathbf{b}}_8$	$\mathbb{R}^{n_\mathcal{T} \times d}$	$\boldsymbol{\Delta}_{10}$
⋆ 2nd Skip/Add	$\mathbf{a}_{10} = \mathbf{a}_5 + \mathbf{a}_9$	$\mathbb{R}^{n_\mathcal{T} \times d}$	$\boldsymbol{\Delta}_{10} = \frac{1}{d\,\widetilde{\boldsymbol{\sigma}_{10}}} \left[\left(d\,\widetilde{\mathbf{w}}_{10} \boldsymbol{\Delta}_{11}^t - \sum \widetilde{\mathbf{w}}_{10} \boldsymbol{\Delta}_{11}^t - \overline{\mathbf{a}}_{10}^t \circ \sum \overline{\mathbf{a}}_{10}^t \circ \widetilde{\mathbf{w}}_{10} \boldsymbol{\Delta}_{11}^t \right)^t \right]$ $\boldsymbol{\Delta}_{10}$ → "dispatched" to $\mathbf{a}_5$ and $\mathbf{a}_9$
LayerNorm(final)	$\mathbf{a}_{11} = \overline{\mathbf{a}}_{10} \text{diag}\,(\mathbf{w}_{10}) + \widetilde{\mathbf{b}}_{10}$	$\mathbb{R}^{n_\mathcal{T} \times d}$	$\boldsymbol{\Delta}_{11} = \boldsymbol{\Delta}_{12}\, \mathbf{w}_{11}^t$
$\text{FC}_{\text{logits}}$	$\mathbf{a} \equiv \mathbf{a}_{12} = \mathbf{a}_{11} \mathbf{w}_{11} + \widetilde{\mathbf{b}}_{11}$	$\mathbb{R}^{n_\mathcal{T} \times n_{\text{vocab}}}$	$\boldsymbol{\Delta}_{12} = \mathbf{y}_{\text{pred}} - \mathbf{y}_{\text{gt}}$
Softmax	$\mathbf{y}_{\text{pred}} = \text{softmax}\,\mathbf{a}$	$\mathbb{R}^{n_\mathcal{T} \times n_{\text{vocab}}}$	probability distributions over n_{vocab} classes for all $n_\mathcal{T}$ tokens

Table 2: **Minimalistic transformer architecture for next-token prediction.** Other than the fact that this network has only a single transformer block (as opposed to $n_{\text{blocks}} = 12$ back-to-back in GPT-2), its structure is conceptually identical in all other aspects to the GPT-like family of models. The fully connected layer immediately after the multi-headed attention layer allows to mix information across different heads. *(Instead of a simple concatenation as is done in our bare-bone MHA, deep learning frameworks usually incorporate this fully-connected layer directly as part of their multi-headed attention APIs as an "output projection" but we keep them separate here for the sake of clarity. In case that the attention heads are not returning $d_h = d/n_h$ dimensional feature maps, this output projection can also be used to bring the dimensionality of the feature maps back to d.)* Thanks to the softmax function, $\mathbf{y}_{\text{pred}}$ represent the next-token predicted probability distributions. For example, consider a specific token $t^\star$, its prediction vector $\mathbf{y}_{\text{pred}}(t^\star) \sim \mathbb{R}^{n_{\text{vocab}}}$ is such that $\sum \mathbf{y}_{\text{pred}}(t^\star) = 1$. Different sampling strategies may be applied to select tokens from these probability distributions.

Parameters			Dimensionality		Loss derivative
Token embedding	$\mathcal{P}_{\text{emb}}$	$\mathbf{w}_{\text{tok}}$	$\mathbb{R}^{n_{\text{vocab}} \times d}$	$38,597,376$	$\partial \mathcal{L}_{\text{batch}} / \partial \mathbf{w}_{\text{tok}} = \text{ohe}(\mathbf{a}_0)^t \, \mathbf{\Delta}_1$
Position embedding		$\mathbf{w}_{\text{pos}}$	$\mathbb{R}^{n_{\text{context}} \times d}$	$786,432$	$\partial \mathcal{L}_{\text{batch}} / \partial \mathbf{w}_{\text{pos}} = \text{ohe}(1\!:\!n_{\mathcal{T}})^t \, \mathbf{\Delta}_1$
1st LayerNorm	$\mathcal{P}_1$	$\mathbf{w}_1$	$\mathbb{R}^d$	768	$\partial \mathcal{L}_{\text{seq}} / \partial \mathbf{w}_1 = \text{diag}\left(\overline{\mathbf{a}}_1^t \mathbf{\Delta}_2\right)$
		$\mathbf{b}_1$	$\mathbb{R}^d$	768	$\partial \mathcal{L}_{\text{seq}} / \partial \mathbf{b}_1 = \sum_{\text{tokens}} \mathbf{\Delta}_2$
Multi-headed attention	$\mathcal{P}_{Q_h}$	$\mathbf{w}_{q_h}$	$\mathbb{R}^{d \times d_\rho}$	$589,824/n_h$	$\partial \mathcal{L}_{\text{seq}} / \partial \mathbf{w}_{q_h} = \mathbf{a}_2^t \, \mathbf{\Delta}_{3(\text{raw})}^h \, \mathbf{k}_h$
		$\mathbf{b}_{q_h}$	$\mathbb{R}^{d_\rho}$	$768/n_h$	$\partial \mathcal{L}_{\text{seq}} / \partial \mathbf{b}_{q_h} = \sum_{\text{tokens}} \mathbf{\Delta}_{3(\text{raw})}^h \, \mathbf{k}_h$
(Queries, Keys, Values)	$\mathcal{P}_{K_h}$	$\mathbf{w}_{k_h}$	$\mathbb{R}^{d \times d_\rho}$	$589,824/n_h$	$\partial \mathcal{L}_{\text{seq}} / \partial \mathbf{w}_{k_h} = (\mathbf{\Delta}_{3(\text{raw})}^h \, \mathbf{a}_2)^t \, \mathbf{q}_h$
		$\mathbf{b}_{k_h}$	$\mathbb{R}^{d_\rho}$	$768/n_h$	$\partial \mathcal{L}_{\text{seq}} / \partial \mathbf{b}_{k_h} \equiv \mathbf{0}$
for all attention heads $h \in [1, \cdots, n_h]$	$\mathcal{P}_{V_h}$	$\mathbf{w}_{v_h}$	$\mathbb{R}^{d \times d_h}$	$589,824/n_h$	$\partial \mathcal{L}_{\text{batch}} / \partial \mathbf{w}_{v_h} = \left(\boldsymbol{\rho}_{(\mathbf{a}_2,\, h)} \, \mathbf{a}_2\right)^t \mathbf{\Delta}_3^h$
		$\mathbf{b}_{v_h}$	$\mathbb{R}^{d_h}$	$768/n_h$	$\partial \mathcal{L}_{\text{batch}} / \partial \mathbf{b}_{v_h} = \sum_{\text{tokens}} \mathbf{\Delta}_3^h$
$\text{FC}_{\text{attProj}}$	$\mathcal{P}_3$	$\mathbf{w}_3$	$\mathbb{R}^{d \times d}$	$589,824$	$\partial \mathcal{L}_{\text{seq}} / \partial \mathbf{w}_3 = \mathbf{a}_3^t \, \mathbf{\Delta}_5$
		$\mathbf{b}_3$	$\mathbb{R}^d$	768	$\partial \mathcal{L}_{\text{seq}} / \partial \mathbf{b}_3 = \sum_{\text{tokens}} \mathbf{\Delta}_5$
2nd LayerNorm	$\mathcal{P}_5$	$\mathbf{w}_5$	$\mathbb{R}^d$	768	$\partial \mathcal{L}_{\text{seq}} / \partial \mathbf{w}_5 = \text{diag}\left(\overline{\mathbf{a}}_5^t \mathbf{\Delta}_6\right)$
		$\mathbf{b}_5$	$\mathbb{R}^d$	768	$\partial \mathcal{L}_{\text{seq}} / \partial \mathbf{b}_5 = \sum_{\text{tokens}} \mathbf{\Delta}_6$
$\text{FC}_{\text{expand}}$	$\mathcal{P}_6$	$\mathbf{w}_6$	$\mathbb{R}^{d \times 4d}$	$2,359,296$	$\partial \mathcal{L}_{\text{seq}} / \partial \mathbf{w}_6 = \mathbf{a}_6^t \, \mathbf{\Delta}_7$
		$\mathbf{b}_6$	$\mathbb{R}^{4d}$	3072	$\partial \mathcal{L}_{\text{seq}} / \partial \mathbf{b}_6 = \sum_{\text{tokens}} \mathbf{\Delta}_7$
$\text{FC}_{\text{contract}}$	$\mathcal{P}_8$	$\mathbf{w}_8$	$\mathbb{R}^{4d \times d}$	$2,359,296$	$\partial \mathcal{L}_{\text{seq}} / \partial \mathbf{w}_8 = \mathbf{a}_8^t \, \mathbf{\Delta}_{10}$
		$\mathbf{b}_8$	$\mathbb{R}^d$	768	$\partial \mathcal{L}_{\text{seq}} / \partial \mathbf{b}_8 = \sum_{\text{tokens}} \mathbf{\Delta}_{10}$
LayerNorm(final)	$\mathcal{P}_{10}$	$\mathbf{w}_{10}$	$\mathbb{R}^d$	768	$\partial \mathcal{L}_{\text{seq}} / \partial \mathbf{w}_{10} = \text{diag}\left(\overline{\mathbf{a}}_{10}^t \mathbf{\Delta}_{11}\right)$
		$\mathbf{b}_{10}$	$\mathbb{R}^d$	768	$\partial \mathcal{L}_{\text{seq}} / \partial \mathbf{b}_{10} = \sum_{\text{tokens}} \mathbf{\Delta}_{11}$
$\text{FC}_{\text{logits}}$	$\mathcal{P}_{11}^{(\text{fc})}$	$\mathbf{w}_{11}$	$\mathbb{R}^{d \times n_{\text{vocab}}}$	$38,597,376$	$\partial \mathcal{L}_{\text{seq}} / \partial \mathbf{w}_{11} = \mathbf{a}_{11}^t \, \mathbf{\Delta}_{12}$
		$\mathbf{b}_{11}$	$\mathbb{R}^{n_{\text{vocab}}}$	$50,257$	$\partial \mathcal{L}_{\text{seq}} / \partial \mathbf{b}_{11} = \sum_{\text{tokens}} \mathbf{\Delta}_{12}$
$\text{LoRA}_{\text{logits}}$	$\mathcal{P}_{11}^{(\text{lora})}$	$\mathbf{d}_{11}$	$\mathbb{R}^{d \times r}$	$12,288$	$\partial \mathcal{L}_{\text{seq}} / \partial \mathbf{d}_{11} = \alpha \, \mathbf{a}_{11}^t \, \mathbf{\Delta}_{12} \, \mathbf{u}_{11}^t$
		$\mathbf{u}_{11}$	$\mathbb{R}^{r \times n_{\text{vocab}}}$	$804,112$	$\partial \mathcal{L}_{\text{seq}} / \partial \mathbf{u}_{11} = \alpha \left(\mathbf{a}_{11} \, \mathbf{d}_{11}\right)^t \mathbf{\Delta}_{12}$

Table 3: **Parameter gradients should be read from bottom to top following the order in which they are updated during backpropagation.** As per common practice, the dimensionality d_ρ of the queries/keys feature maps in $\mathcal{P}_{Q_h/K_h}$ is set to match $d_\rho \equiv d_h$ the dimensionality of the values output feature maps in $\mathcal{P}_{V_h}$ and $n_h = d/d_h \in \mathbb{N}$ denotes the number of head in the multi-headed attention layer.

"The Queen propped her up against a tree, and said kindly, You may rest a little now.

Alice looked round her in great surprise. Why, I do believe we've been under this tree the whole time! Everything's just as it was!

Of course it is, said the Queen, what would you have it?

Well, in our country, said Alice, still panting a little, you'd generally get to somewhere else — if you ran very fast for a long time, as we've been doing.

A slow sort of country! said the Queen. Now, here, you see, it takes all the running you can do, to keep in the same place.

If you want to get somewhere else, you must run at least twice as fast as that!"

(Lewis Carroll, Through the Looking-Glass, 1871)

"If you're thinking without writing, you only think you're thinking."

(Leslie Lamport)

References

[1] Laurent Boué "Deep learning for pedestrians: backpropagation in CNNs". arXiv:1811.11987 (2018)

[2] Jian Jia, Jingtong Gao, Ben Xue, Junhao Wang, Qingpeng Cai, Quan Chen, Xiangyu Zhao, Peng Jiang & Kun Gai "From Principles to Applications: A Comprehensive Survey of Discrete Tokenizers in Generation, Comprehension, Recommendation, and Information Retrieval". arXiv:2502.12448 (2025)

[3] Ryan Xu "Motivating Self-Attention". Towards Data Science

[4] Matt Sarmiento "Making Sense of Attention". Blog post

[5] Hemanth Saratchandran, Jianqiao Zheng, Yiping Ji, Wenbo Zhang & Simon Lucey "Rethinking Softmax: Self-Attention with Polynomial Activations". arXiv:2410.18613

[6] Feyza Duman Keles, Pruthuvi Mahesakya Wijewardena & Chinmay Hegde "On The Computational Complexity of Self-Attention". PMLR (2023)

[7] Nikita Kitaev, Łukasz Kaiser & Anselm Levskaya "Reformer: The Efficient Transformer". ICML (2020)

[8] Krzysztof Choromanski, Valerii Likhosherstov, David Dohan, Xingyou Song, Andreea Gane, Tamas Sarlos, Peter Hawkins, Jared Davis, Afroz Mohiuddin, Lukasz Kaiser, David Belanger, Lucy Colwell & Adrian Weller "Rethinking Attention with Performers". ICLR (2020)

[9] Tri Dao, Dan Fu, Stefano Ermon, Atri Rudra & Christopher Ré "FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness". NeurIPS (2022)

[10] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser & Illia Polosukhin "Attention Is All You Need". NeurIPS (2017)

[11] Giacomo Livan, Marcel Novaes & Pierpaolo Vivo "Introduction to Random Matrices — Theory and Practice". arXiv:1712.07903 (2017)

[12] Permutation matrix (Wikipedia)

[13] Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Ruslan Salakhutdinov & Alexander Smola "Deep Sets". NeurIPS (2017)

[14] Yao-Hung Hubert Tsai, Shaojie Bai, Makoto Yamada, Louis-Philippe Morency & Ruslan Salakhutdinov "Transformer Dissection: An Unified Understanding for Transformer's Attention via the Lens of Kernel". EMNLP (2019)

[15] Adi Haviv, Ori Ram, Ofir Press, Peter Izsak & Omer Levy "Transformer Language Models without Positional Encodings Still Learn Positional Information". EMNLP (2022)

[16] Pierre Lienhart (Medium)

[17] Paul Michel, Omer Levy & Graham Neubig "Are Sixteen Heads Really Better than One?". NeurIPS (2019)

[18] Jimmy Lei Ba, Jamie Ryan Kiros & Geoffrey E. Hinton "Layer Normalization". arXiv:1607.06450 (2016)

[19] Sergey Ioffe & Christian Szegedy. "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift". ICML (2015)

[20] Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang & Weizhu Chen "LoRA: Low-Rank Adaptation of Large Language Models". arXiv:2106.09685 (2021)

Appendix A Matrices: some more potpourri...

Linear mixing. For the sake of generality, let us consider a mirror version of eq.(6) where the components $\rho_{\alpha\beta}$ of the weight matrix $\boldsymbol{\rho} \sim \mathbb{R}^{n\times n}$ are not limited by causality and where we simplify the notation by denoting with $\mathbf{a} \sim \mathbb{R}^{n\times d}$ the stack of d-dimensional vectors representing n tokens.

Let us denote by $\tilde{\mathbf{a}}$ the weighted average of $\mathbf{a}$ such that

$$\tilde{\mathbf{a}} = \boldsymbol{\rho}\,\mathbf{a} \quad \Longrightarrow \quad \begin{pmatrix} \tilde{a}_{11} & — & \tilde{a}_{1d} \\ & \vdots & \\ \tilde{a}_{n1} & — & \tilde{a}_{nd} \end{pmatrix} = \begin{pmatrix} \rho_{11} & \cdots & \rho_{1n} \\ \vdots & \ddots & \vdots \\ \rho_{n1} & \cdots & \rho_{nn} \end{pmatrix} \begin{pmatrix} a_{11} & — & a_{1d} \\ & \vdots & \\ a_{n1} & — & a_{nd} \end{pmatrix}$$

and focus on a specific token $\tilde{\mathbf{a}}(t = t^\star) = \begin{bmatrix} \tilde{a}_{t^\star 1} & \cdots & \tilde{a}_{t^\star d} \end{bmatrix} \sim \mathbb{R}^d$ of $\tilde{\mathbf{a}} \sim \mathbb{R}^{n\times d}$. The components of $\tilde{\mathbf{a}}(t = t^\star)$ are given by

$$\begin{aligned} \begin{bmatrix} \tilde{a}_{t^\star 1} & \cdots & \tilde{a}_{t^\star d} \end{bmatrix} &= \begin{bmatrix} \rho_{t^\star 1} & \cdots & \rho_{t^\star n} \end{bmatrix} \begin{pmatrix} a_{11} & \cdots & a_{1d} \\ \vdots & \ddots & \vdots \\ a_{n1} & \cdots & a_{nd} \end{pmatrix} \\ &= \begin{bmatrix} (\rho_{t^\star 1} a_{11} + \cdots + \rho_{t^\star n} a_{n1}) & \cdots & (\rho_{t^\star 1} a_{1d} + \cdots + \rho_{t^\star n} a_{nd}) \end{bmatrix} \\ &= \rho_{t^\star 1} \begin{bmatrix} a_{11} & \cdots & a_{1d} \end{bmatrix} + \cdots + \rho_{t^\star n} \begin{bmatrix} a_{n1} & \cdots & a_{nd} \end{bmatrix} \\ &\downarrow \ \text{in vectorized notation} \end{aligned}$$

$$\tilde{\mathbf{a}}(t = t^\star) = \rho_{t^\star 1}\,\mathbf{a}(t = 1) + \cdots + \rho_{t^\star n}\,\mathbf{a}(t = n) \sim \mathbb{R}^d \tag{47}$$

This shows that $\tilde{\mathbf{a}}(t = t^\star)$ is a linear combination of all the token feature vectors (i.e. all rows) of $\mathbf{a}$. Applying this to all tokens, we get the expected linear mixing

$$\tilde{\mathbf{a}} = \begin{pmatrix} — \ \tilde{\mathbf{a}}(t = 1) \sim \mathbb{R}^d \ — \\ \vdots \\ — \ \tilde{\mathbf{a}}(t = n) \sim \mathbb{R}^d \ — \end{pmatrix} = \begin{pmatrix} \rho_{11}\,\mathbf{a}(t = 1) + \cdots + \rho_{1n}\,\mathbf{a}(t = n) \\ \vdots \\ \rho_{n1}\,\mathbf{a}(t = 1) + \cdots + \rho_{nn}\,\mathbf{a}(t = n) \end{pmatrix}$$

In the special case where the weight components $\rho_{\alpha\beta}$ are causal with $\rho_{\alpha\beta} = 0$ if $\beta > \alpha$, we recover the expected system of equations presented in Section 3 in the main part of the text

$$\text{Eqs.(5.a)–(5.d)} \equiv \begin{cases} \tilde{\mathbf{a}}(t = 1) = \rho_{11}\,\mathbf{a}(t = 1) \\ \tilde{\mathbf{a}}(t = 2) = \rho_{21}\,\mathbf{a}(t = 1) + \rho_{22}\,\mathbf{a}(t = 2) \\ \tilde{\mathbf{a}}(t = 3) = \rho_{31}\,\mathbf{a}(t = 1) + \rho_{32}\,\mathbf{a}(t = 2) + \rho_{33}\,\mathbf{a}(t = 3) \\ \qquad \vdots \\ \tilde{\mathbf{a}}(t = n) = \rho_{n1}\,\mathbf{a}(t = 1) + \rho_{n2}\,\mathbf{a}(t = 2) + \rho_{n3}\,\mathbf{a}(t = 3) + \cdots + \rho_{nn}\,\mathbf{a}(t = n) \end{cases} \tag{48}$$

simply by identifying $\tilde{\mathbf{a}} \equiv \mathbf{a}_i^h$, $\mathbf{a} \equiv \mathbf{v}_h$ and $n \equiv n_{\mathcal{T}}$.

Cycles of Frobenius products. Generalizing the Frobenius product identity shown in Eq. (52) of reference [1], we consider 4 matrices $\mathbf{A}, \mathbf{B}, \mathbf{C}$ and $\mathbf{D}$ subject to dimensionality constraints such that $\mathbf{A} \sim \mathbf{BCD} \sim \mathbb{R}^{n\times f}$. By definition of the Frobenius product and simple rewriting of the expressions, we have

$$\begin{aligned} \mathbf{A} \cdot (\mathbf{BCD}) &= \mathrm{Tr}\big(\mathbf{A}^t\,\mathbf{BCD}\big) \\ &= \mathrm{Tr}\big[\big((\mathbf{A}^t\,\mathbf{BC})^t\big)^t\,\mathbf{D}\big] \\ &= \mathrm{Tr}\big[\big(\mathbf{C}^t\mathbf{B}^t\mathbf{A}\big)^t\,\mathbf{D}\big] \\ &= \mathbf{C}^t\mathbf{B}^t\mathbf{A} \cdot \mathbf{D} \\ &= \big[(\mathbf{BC})^t\mathbf{A}\big] \cdot \mathbf{D} \end{aligned}$$

Other useful identities may be obtained by using the invariance of the trace under circular shifts and the same manipulations as above

$$\mathbf{A}\cdot(\mathbf{BCD}) = \mathrm{Tr}\big(\mathbf{A}^t\,\mathbf{BCD}\big) = \begin{cases} \mathrm{Tr}\big(\mathbf{DA}^t\,\mathbf{BC}\big) = \big(\mathbf{B}^t\mathbf{AD}^t\big)\cdot\mathbf{C} \\ \mathrm{Tr}\big(\mathbf{CDA}^t\,\mathbf{B}\big) = \big[\mathbf{A}\big(\mathbf{CD}\big)^t\,\big]\cdot\mathbf{B} \end{cases}$$

Another useful identity which can be derived from the same cyclic property and transpose invariance of the trace operator

$$\mathbf{A}\cdot\mathbf{B}^t = \mathbf{A}^t\cdot\mathbf{B} \tag{49}$$

More broadcasting semantics. Let us start by considering two matrices $\mathbf{A}\sim\mathbf{B}\sim\mathbb{R}^{n\times n}$ to mirror part of the expression for $\boldsymbol{\Delta}^h_{\text{causal}}$ in eq. (25) (where $\mathbf{A}=\boldsymbol{\Delta}^h_i\mathbf{v}^t_h$ and $\mathbf{B}=\boldsymbol{\rho}_{(\mathbf{a}_{i-1},\,h)}$). Their feature dot-product $\mathbf{A}\ominus\mathbf{B}\sim\mathbb{R}^n$ results in a column vector which is broadcast column-wise as

$$\widetilde{\mathbf{A}\ominus\mathbf{B}} = \begin{pmatrix} \mathbf{a}_1\cdot\mathbf{b}_1 & \cdots & \mathbf{a}_1\cdot\mathbf{b}_1 \\ \vdots & \ddots & \vdots \\ \mathbf{a}_n\cdot\mathbf{b}_n & \cdots & \mathbf{a}_n\cdot\mathbf{b}_n \end{pmatrix} \tag{50}$$

See eq.(48) of the reference paper [1] for a reminder of the $\ominus$ operator. The same broadcast can be expressed in a more convenient vectorized manner for practical implementations with

$$\begin{aligned}
\widetilde{\mathbf{A}\ominus\mathbf{B}} &= \begin{pmatrix} (a_{11}b_{11}+a_{1n}b_{1n}) & \cdots & (a_{11}b_{11}+a_{1n}b_{1n}) \\ \vdots & \ddots & \vdots \\ (a_{n1}b_{n1}+a_{nn}b_{nn}) & \cdots & (a_{n1}b_{n1}+a_{nn}b_{nn}) \end{pmatrix} \\
&= \left[\begin{pmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{n1} & \cdots & a_{nn} \end{pmatrix} \circ \begin{pmatrix} b_{11} & \cdots & b_{1n} \\ \vdots & \ddots & \vdots \\ b_{n1} & \cdots & b_{nn} \end{pmatrix}\right] \begin{pmatrix} 1 & \cdots & 1 \\ \vdots & \ddots & \vdots \\ 1 & \cdots & 1 \end{pmatrix} \\
&= (\mathbf{A}\circ\mathbf{B})\,\mathbf{J}_{n,n}
\end{aligned}$$

Next, let us consider three square matrices $\mathbf{A}\sim\mathbf{B}\sim\mathbf{C}\sim\mathbb{R}^{n\times n}$ and simplify $\big(\mathbf{A}\circ\mathbf{B}\big)\cdot\big(\widetilde{\mathbf{B}\ominus\mathbf{C}}\big)$ consisting of the Frobenius product between $(\mathbf{A}\circ\mathbf{B})\sim\mathbb{R}^{n\times n}$ and the column-wise broadcast of the feature vector $\mathbb{R}^n$ given by the feature dot-product $\mathbf{B}\ominus\mathbf{C}\sim\mathbb{R}^n$. Since the result of the feature dot-product is a column vector (one dot-product per row), the broadcast has to be column-wise. Note that in [1], the sum over tokens is denoted as a sum over "samples". In this paper, we use the terminology "token" as the unit instead of sample since we prefer to think of a sample as a sequence of tokens. Regardless, tokens have their own feature maps just like samples do in [1]. Therefore, the symbols $\sum_{\text{tokens}}$ and $\sum_{\text{samples}}$ are functionally equivalent to each other and one should just consider them as a vertical sum along columns (i.e. that cuts across rows). With this in mind, we can now evaluate the desired expression

$$\begin{aligned}
\big(\mathbf{A}\circ\mathbf{B}\big)\cdot\big(\widetilde{\mathbf{B}\ominus\mathbf{C}}\big) &= \sum_{\text{tokens}} \big(\mathbf{A}\circ\mathbf{B}\big)\ominus\big(\widetilde{\mathbf{B}\ominus\mathbf{C}}\big) \\
&= \sum_{\text{tokens}} \left[\begin{pmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{n1} & \cdots & a_{nn} \end{pmatrix} \begin{pmatrix} b_{11} & \cdots & b_{1n} \\ \vdots & \ddots & \vdots \\ b_{n1} & \cdots & b_{nn} \end{pmatrix} \ominus \left(\widetilde{\begin{pmatrix} b_{11} & \cdots & b_{1n} \\ \vdots & \ddots & \vdots \\ b_{n1} & \cdots & b_{nn} \end{pmatrix} \ominus \begin{pmatrix} c_{11} & \cdots & c_{1n} \\ \vdots & \ddots & \vdots \\ c_{n1} & \cdots & c_{nn} \end{pmatrix}}\right)\right] \\
&= \sum_{\text{tokens}} \left[\begin{pmatrix} a_{11}b_{11} & \cdots & a_{1n}b_{1n} \\ \vdots & \vdots & \vdots \\ a_{n1}b_{n1} & \cdots & a_{nn}b_{nn} \end{pmatrix} \ominus \widetilde{\begin{pmatrix} \mathbf{b}_1\cdot\mathbf{c}_1 \\ \vdots \\ \mathbf{b}_n\cdot\mathbf{c}_n \end{pmatrix}}\right] \\
&\downarrow \ \text{using the same column-wise broadcast as in eq.(50)} \\
&= \sum_{\text{tokens}} \left[\begin{pmatrix} a_{11}b_{11} & \cdots & a_{1n}b_{1n} \\ \vdots & \ddots & \vdots \\ a_{n1}b_{n1} & \cdots & a_{nn}b_{nn} \end{pmatrix} \ominus \begin{pmatrix} \mathbf{b}_1\cdot\mathbf{c}_1 & — & \mathbf{b}_1\cdot\mathbf{c}_1 \\ \cdots & \cdots & \cdots \\ \mathbf{b}_n\cdot\mathbf{c}_n & — & \mathbf{b}_n\cdot\mathbf{c}_n \end{pmatrix}\right]
\end{aligned}$$

$$
\begin{aligned}
&= \sum_{\text{tokens}} \begin{pmatrix} \big(a_{11}b_{11} + \cdots + a_{1n}b_{1n}\big)\,\mathbf{b}_1 \cdot \mathbf{c}_1 \\ \vdots \\ \big(a_{n1}b_{n1} + \cdots + a_{nn}b_{nn}\big)\,\mathbf{b}_n \cdot \mathbf{c}_n \end{pmatrix} \\
&= \sum_{\text{tokens}} \begin{pmatrix} \big(\mathbf{a}_1 \cdot \mathbf{b}_1\big)\,\mathbf{b}_1 \cdot \mathbf{c}_1 \\ \vdots \\ \big(\mathbf{a}_n \cdot \mathbf{b}_n\big)\,\mathbf{b}_n \cdot \mathbf{c}_n \end{pmatrix} \\
&= \sum_{\text{tokens}} \begin{pmatrix} \big(\mathbf{a}_1 \cdot \mathbf{b}_1\big)\,b_{11} & \cdots & \big(\mathbf{a}_1 \cdot \mathbf{b}_1\big)\,b_{1n} \\ \vdots & \ddots & \vdots \\ \big(\mathbf{a}_n \cdot \mathbf{b}_n\big)\,b_{n1} & \cdots & \big(\mathbf{a}_1 \cdot \mathbf{b}_1\big)\,b_{nn} \end{pmatrix} \ominus \begin{pmatrix} \mathbf{c}_1 \\ \vdots \\ \mathbf{c}_n \end{pmatrix} \\
&= \sum_{\text{tokens}} \left[\begin{pmatrix} \mathbf{a}_1 \cdot \mathbf{b}_1 & \cdots & \mathbf{a}_1 \cdot \mathbf{b}_1 \\ \vdots & \ddots & \vdots \\ \mathbf{a}_n \cdot \mathbf{b}_n & \cdots & \mathbf{a}_1 \cdot \mathbf{b}_1 \end{pmatrix} \circ \begin{pmatrix} b_{11} & \cdots & b_{1n} \\ \vdots & \ddots & \vdots \\ b_{n1} & \cdots & b_{nn} \end{pmatrix} \right] \ominus \begin{pmatrix} c_{11} & \cdots & c_{1n} \\ \vdots & \ddots & \vdots \\ c_{n1} & \cdots & c_{nn} \end{pmatrix} \\
&= \sum_{\text{tokens}} \big[\big(\widetilde{\mathbf{A} \ominus \mathbf{B}}\big) \circ \mathbf{B}\big] \ominus \mathbf{C} \\
&= \big[\big(\widetilde{\mathbf{A} \ominus \mathbf{B}}\big) \circ \mathbf{B}\big] \cdot \mathbf{C}
\end{aligned}
$$

Summarizing the result, we have

$$
\big(\mathbf{A} \circ \mathbf{B}\big) \cdot \big(\widetilde{\mathbf{B} \ominus \mathbf{C}}\big) = \big[\big(\widetilde{\mathbf{A} \ominus \mathbf{B}}\big) \circ \mathbf{B}\big] \cdot \mathbf{C} \tag{51}
$$

Sum over rows over matrix product. Let us consider two matrices $\mathbf{A} \sim \mathbb{R}^{n \times n}$ and $\mathbf{B} \sim \mathbb{R}^{n \times f}$ to mirror the situation of eqs.(21) and (29). We are interested in simplifying the row-wise sum (i.e. vertical) of their matrix product $\sum_{\text{rows}} \mathbf{A}\,\mathbf{B}$ resulting in a vector $\sim \mathbb{R}^f$ (one component per column). Indexing rows by i and columns by j, the row-wise sum associated with column j of the product $\mathbf{A}\,\mathbf{B}$ is given by

$$
\sum_{i=1}^{n} \left(\mathbf{A}\,\mathbf{B}\right)_{ij} = \sum_{i=1}^{n} \left(\sum_{k=1}^{n} a_{ik} b_{kj} \right) = \sum_{k=1}^{n} \sum_{i=1}^{n} a_{ik} b_{kj} = \sum_{k=1}^{n} \left(\sum_{i=1}^{n} a_{ik} \right) b_{kj} = \sum_{k=1}^{n} a_k^{\updownarrow}\, b_{kj}
$$

where we defined $a_k^{\updownarrow} = \sum_{i=1}^{n} a_{ik}$ as the row-wise (vertical) sum of the k^{th} column of $\mathbf{A}$. Collecting together the n different column sums $a_k^{\updownarrow}$, we define a vector

$$
\mathbf{a}^{\updownarrow} = \big[a_1^{\updownarrow} = \sum_{i=1}^{n} a_{i1}\,, \cdots, a_n^{\updownarrow} = \sum_{i=1}^{n} a_{in}\big] \sim \mathbb{R}^n
$$

with which the all the components of the desired expression can now be evaluated as the vector-matrix product

$$
\sum_{\text{rows}} \mathbf{A}\,\mathbf{B} = \mathbf{a}^{\updownarrow}\,\mathbf{B} \sim \mathbb{R}^f \tag{52}
$$

Appendix B Some properties of the softmax function

Shift invariance. Let us consider a square matrix $\mathbf{G} \sim \mathbb{R}^{n \times n}$ (analogous to an attention weight matrix), a feature map matrix $\mathbf{H} \sim \mathbb{R}^{n \times f}$ and a feature bias vector $\mathbf{b} \sim \mathbb{R}^f$. In this case, we can reproduce the pattern of eq.(19) that we wish to expand as

$$
\text{softmax}(\underbrace{\Big(\mathbf{a}_{i-1}\mathbf{w}_{q_h} + \widetilde{\mathbf{b}}_{q_h}\Big) \left(\mathbf{a}_{i-1}\mathbf{w}_{k_h}\right)^t}_{\mathbf{G}} + \underbrace{\left(\mathbf{a}_{i-1}\mathbf{w}_{q_h} + \mathbf{b}_{q_h}\right)}_{\mathbf{H}} \underbrace{\widetilde{\mathbf{b}_{k_h}^t}}_{\widetilde{\mathbf{b}^{\mathbf{t}}}}) \quad \Longrightarrow \quad \text{softmax}\Big(\mathbf{G} + \mathbf{H}\,\widetilde{\mathbf{b}^{\mathbf{t}}}\Big) \tag{53}
$$

The first step consists in broadcasting the vector $\mathbf{b} \sim \mathbb{R}^f$ row-wise into a matrix $\widetilde{\mathbf{b}} \sim \mathbb{R}^{n\times f}$ with dimensions compatible with $\mathbf{H} \sim \mathbb{R}^{n\times f}$.

$$\mathbf{b} = [b_1, \cdots, b_f] \sim \mathbb{R}^f \quad \Longrightarrow \begin{cases} \widetilde{\mathbf{b}} \leftarrow \begin{pmatrix} \text{—}\,\mathbf{b} \sim \mathbb{R}^f\,\text{—} \\ \vdots \\ \text{—}\,\mathbf{b} \sim \mathbb{R}^f\,\text{—} \end{pmatrix} = \begin{pmatrix} b_1 & \text{—} & b_f \\ \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots \\ b_1 & \text{—} & b_f \end{pmatrix} \sim \mathbb{R}^{n\times f} \\ \widetilde{\mathbf{b}^\mathbf{t}} \leftarrow \begin{pmatrix} | & & | \\ \mathbf{b} & \ldots & \mathbf{b} \\ | & & | \end{pmatrix} = \begin{pmatrix} b_1 & \vdots & \vdots & b_1 \\ | & \vdots & \vdots & | \\ b_f & \vdots & \vdots & b_f \end{pmatrix} \sim \mathbb{R}^{f\times n} \end{cases}$$

This way, the matrix product $\mathbf{H}\,\widetilde{\mathbf{b}^\mathbf{t}} \sim \mathbb{R}^{n\times n}$ produces a square matrix which can be added to $\mathbf{G}$. This is written explicitly as

$$\begin{aligned} \operatorname{softmax}\left(\mathbf{G} + \mathbf{H}\,\widetilde{\mathbf{b}^\mathbf{t}}\right) &= \operatorname{softmax}\left[\begin{pmatrix} g_{11} & \cdots & g_{1n} \\ \vdots & \ddots & \vdots \\ g_{n1} & \cdots & g_{nn} \end{pmatrix} + \begin{pmatrix} h_{11} & \cdots & h_{1f} \\ \vdots & \ddots & \vdots \\ h_{n1} & \cdots & h_{nf} \end{pmatrix} \begin{pmatrix} b_1 & \vdots & \vdots & b_1 \\ | & \vdots & \vdots & | \\ b_f & \vdots & \vdots & b_f \end{pmatrix}\right] \\ &= \operatorname{softmax}\left[\begin{pmatrix} g_{11} & \cdots & g_{1n} \\ \vdots & \ddots & \vdots \\ g_{n1} & \cdots & g_{nn} \end{pmatrix} + \begin{pmatrix} \mathbf{h}_1\cdot\mathbf{b} & \text{—} & \mathbf{h}_1\cdot\mathbf{b} \\ \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots \\ \mathbf{h}_n\cdot\mathbf{b} & \text{—} & \mathbf{h}_n\cdot\mathbf{b} \end{pmatrix}\right] \end{aligned}$$

$\downarrow$ the product $\mathbf{H}\,\widetilde{\mathbf{b}^\mathbf{t}}$ creates a shift-matrix where all rows are constant

define $\lambda_i \equiv \mathbf{v}_i \cdot \mathbf{b} \sim \mathbb{R}$ as a short-hand notation for the row-specific constant

$$= \operatorname{softmax}\begin{pmatrix} g_{11}+\lambda_1 & \cdots & g_{1n}+\lambda_1 \\ \cdots & \ddots & \cdots \\ g_{n1}+\lambda_n & \cdots & g_{nn}+\lambda_n \end{pmatrix}$$

$\downarrow$ showing that the components of each row i are all shifted by a constant value λ_i

$$= \begin{pmatrix} \operatorname{softmax}\left(g_{11}+\lambda_1, \cdots, g_{1n}+\lambda_1\right) \\ \vdots \\ \operatorname{softmax}\left(g_{n1}+\lambda_n, \cdots, g_{nn}+\lambda_n\right) \end{pmatrix}$$

Using one specific row i as an example, the shift-invariance property of the softmax normalization can be understood as

$$\begin{aligned} \operatorname{softmax}\left(g_{i1}+\lambda_i, \cdots, g_{in}+\lambda_i\right) &= \left[\frac{e^{(g_{i1}+\lambda_i)}}{e^{(g_{i1}+\lambda_i)} + \cdots + e^{(g_{in}+\lambda_i)}}, \cdots, \frac{e^{(g_{in}+\lambda_i)}}{e^{(g_{i1}+\lambda_i)} + \cdots + e^{(g_{in}+\lambda_i)}}\right] \\ &= \left[\frac{e^{g_{i1}}\,e^{\lambda_i}}{\left(e^{g_{i1}} + \cdots + e^{g_{in}}\right)\,e^{\lambda_i}}, \cdots, \frac{e^{g_{in}}\,e^{\lambda_i}}{\left(e^{g_{i1}} + \cdots + e^{g_{in}}\right)\,e^{\lambda_i}}\right] \\ &= \left[\frac{e^{g_{i1}}}{e^{g_{i1}} + \cdots + e^{g_{in}}}, \cdots, \frac{e^{g_{in}}}{e^{g_{i1}} + \cdots + e^{g_{in}}}\right] \\ &= \operatorname{softmax}\left(g_{i1}, \cdots, g_{in}\right) \end{aligned}$$

where the dependence on λ_i cancels out. Applying this shift-invariance to all rows leads us to the result

$$\operatorname{softmax}\left(\mathbf{G} + \mathbf{H}\,\widetilde{\mathbf{b}}^t\right) = \operatorname{softmax}\mathbf{G} \tag{54}$$

showing that the dependence on $\widetilde{\mathbf{b}^\mathbf{t}}$ completely drops out. Notice that this was possible due to the transpose of the **broadcast** of $\mathbf{b}$ which created a shift-matrix $\mathbf{H}\,\widetilde{\mathbf{b}^\mathbf{t}}$.

Going back to the original notation, one should note that $\mathbf{G}$ contains the parameters $\{\mathbf{w}_{q_h}, \mathbf{b}_{q_h}, \mathbf{w}_{k_h}\}$. So it is really just the bias parameter $\mathbf{b}_{k_h}$ that is redundant in evaluating eq.(53), copy of (19).

Permutations. We investigate the behavior of the softmax function for

- row-wise permutations. Let us consider a matrix $\mathbf{A} \sim \mathbb{R}^{n\times d}$ with pre-multiplication by the permutation matrix $\mathcal{P}_1 \sim \mathbb{R}^{n\times n}$ so that $\mathcal{P}_1\,\mathbf{A}$ is a row-permutated version of $\mathbf{A}$ [12]. What is the effect of $\mathcal{P}_1$ on $\text{softmax}\big(\mathcal{P}_1\,\mathbf{A}\big)$? Since the softmax function is applied independently to each row and does not mix data across the rows, there is no difference between i) first permutating the rows and next applying softmax to the permutated rows, i.e. $\text{softmax}\big(\mathcal{P}_1\,\mathbf{A}\big)$ or ii) first applying the softmax to each row and next permutating the rows, i.e. $\mathcal{P}_1\,\text{softmax}\,\mathbf{A}$. Therefore we have

$$\text{softmax}\big(\mathcal{P}_1\,\mathbf{A}\big) = \mathcal{P}_1\,\text{softmax}\,\mathbf{A} \tag{55.a}$$

- column-wise permutations. Let us consider another matrix of transposed dimensions $\mathbf{B} \sim \mathbb{R}^{d\times n}$ with post-multiplication by another permutation matrix $\mathcal{P}_2 \sim \mathbb{R}^{n\times n}$ so that $\mathbf{B}\,\mathcal{P}_2$ is a column-permutated version of $\mathbf{B}$ [12]. What is the effect of $\mathcal{P}_2$ on $\text{softmax}\big(\mathbf{B}\,\mathcal{P}_2\big)$? Just like before, the softmax function does not mix computations across different rows. This means that, for a given row, the normalization of the denominator is constant across all the columns. Since here we are considering only a re-ordering of the columns, it does not matter whether the exponential of the numerator is applied before or after the re-ordering is performed. Therefore, we have

$$\text{softmax}\big(\mathbf{B}\,\mathcal{P}_2\big) = \text{softmax}(\mathbf{B})\,\mathcal{P}_2 \tag{55.b}$$

In the main part of the text, we are considering a special case where $\mathbf{B} = \mathbf{A}^t$ and the same permutation matrix with $\mathcal{P}_2 = \mathcal{P}_1^t$.

Contents