1 Linear Algebra

Column wise decomposition. Any matrix $\mathbf{A} \in \mathbb{R}^{m\times n}$ can be decomposed into the sum of its columns:

$$\mathbf{A} = \sum_{i=1}^{n} \mathbf{A}_{:j} e_j^\top, \tag{1}$$

where e_j are standard basis vectors of $\mathbb{R}^n$. Notice that this is a rank 1 decomposition.

Row wise decomposition. Any matrix $\mathbf{A} \in \mathbb{R}^{m\times n}$ can be decomposed into the sum of its rows:

$$\mathbf{A} = \sum_{i=1}^{m} e_i \mathbf{A}_{i:}^\top, \tag{2}$$

where e_i are standard basis vectors of $\mathbb{R}^m$. Notice that this is a rank 1 decomposition.

2 LLM Training

Scaling the logits after LLM head. We usually apply RMS norm to normalize (along the last dimension E, where it stands for model dimension, B means batch size and L means sequence length) the tensor $\mathbf{X} \in \mathbb{R}^{B\times L\times E}$ we feed into LLM head, and obtain the corresponding logits l. After RMS normalization, each tensor corresponding to the token $x_i \in \mathbb{R}^E$ will then have $\mathrm{RMS}(x_t) = 1$. Now notice that for each coordinate $x_{i,t}, t \in [E]$, treating as a random variable, its variance is given by

$$\mathrm{Var}\left(x_{i,t}\right) = \mathbb{E}\left[x_{i,t}^2\right] - \left(\mathbb{E}\left[x_{i,t}\right]\right)^2, \tag{3}$$

and if it is zero-mean (or small), then $\mathrm{Var}\left(x_{i,t}\right) \simeq \mathbb{E}\left[x_{i,t}^2\right]$, which is to say that second moment reflects the variance.

The next step is to use the empirical observation that for linear layers, hidden vectors tend to be approximatedly rotation-invariant (isotropic), i.e., each coordinate behaves like the others, so we can use the second moment over the coordinate in a token to replace the actual second moment. And the former, is given by

$$\mathrm{Var}\left(x_{i,t}\right) \simeq \frac{1}{E}\sum_{t=1}^{E} x_{i,t} = 1. \tag{4}$$

Now we start to consider the logits, which is generated by

$$l_{j,i} = w_j^\top x_i = \sum_{t=1}^{E} w_{j,t} x_{i,t}.$$

If we assume each weight entry $w_{j,t}$ are i.i.d. with variance σ^2 the logits variance is give by

$$\mathrm{Var}\left(l_{j,i}\right) = \sum_{t=1}^{E} \sigma^2 \,\mathrm{Var}\left(x_{t,i}\right) \simeq E\sigma^2.$$

So the standard deviation $\sim \sqrt{E}$. To ensure that logits do not scale with the model dimension, we scale it by $\sqrt{E}$.

3 Attention

Multihead Self Attention (MHA) Consider an input tensor $\mathbf{X} \in \mathbb{R}^{B\times L\times d_{\text{model}}}$ to an attention layer, where B is the batch size, L is the sequence length, and d_{model} is the model dimension.

1. The first step involves computing queries, keys, and values. We have three matrices, $\mathbf{W}_q$, $\mathbf{W}_k$, and $\mathbf{W}_v \in \mathbb{R}^{d_{\text{model}}\times d_{\text{model}}}$, and simultaneously perform the following operations

$$\mathbf{Q} = \mathbf{X}\mathbf{W}_q; \quad \mathbf{K} = \mathbf{X}\mathbf{W}_k; \quad \mathbf{V} = \mathbf{X}\mathbf{W}_v. \tag{5}$$

   These operations are vectorized, meaning that for each sequence b in the batch of size B, we do

$$\mathbf{Q}_b = \mathbf{X}_b\mathbf{W}_q \quad \forall b \in [B].$$

   $\mathbf{W}_q$, $\mathbf{W}_k$, and $\mathbf{W}_v$ are trainable parameters shared across the entire batch. The resulting $\mathbf{Q}$, $\mathbf{K}$, and $\mathbf{V}$ have the shape $\mathbb{R}^{B\times L\times d_{\text{model}}}$.

2. Next, for multihead attention, we reshape $\mathbf{Q}$, $\mathbf{K}$, and $\mathbf{V}$ from shape $\mathbb{R}^{B\times L\times d_{\text{model}}}$ into $\mathbb{R}^{B\times H\times L\times d_{\text{head}}}$, where H is the number of attention heads and d_{head} is the dimension of each head. To achieve this, we first divide d_{model} into H heads, resulting in shapes of $\mathbb{R}^{B\times L\times H\times d_{\text{head}}}$. Then we rearrange into $\mathbb{R}^{B\times H\times L\times d_{\text{head}}}$. Conceptually, each head uses a subset of dimensions from d_{model} to compute scores between queries and keys along the sequence dimension L. We will use the following notations $\mathbf{Q}_h, \mathbf{K}_h, \mathbf{V}_h$ to denote the per head tensor in $R^{B\times 1\times L\times d_{\text{head}}}$ for each head $h \in [H]$.

3. In the next step, we perform the attention calculation:

$$\begin{aligned}\mathbf{S}_h &:= \text{Scores}_h(\mathbf{Q}_h, \mathbf{K}_h) = \frac{\mathbf{Q}_h\mathbf{K}_h^\top}{\sqrt{d_{\text{head}}}} + \mathbf{M}\\ \mathbf{A}_h &:= \text{Attention}_h(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}\left(\mathbf{S}_h\right)\mathbf{V}_h, \quad \forall h \in [H].\end{aligned}$$

   The scaled multiplication of $\mathbf{Q}_h$ and $\mathbf{K}_h^\top$ is vectorized, resulting in $\mathbf{S}_h \in \mathbb{R}^{B\times 1\times L\times L}$ and $\mathbf{S} \in \mathbb{R}^{B\times H\times L\times L}$.[1] Optionally, we could use a mask matrix to mask out certain tokens, an example would be the causual self attention. To stabilize the gradients, we element-wise divide raw scores by $\sqrt{d_{\text{head}}}$. This scaling choice can be justified because each element of $\mathbf{Q}_h\mathbf{K}_h^\top$ represents a dot product between vectors of dimension d_{model}. The variance of this dot product scales as $\text{Var}\left(\langle q_h, k_h\rangle\right) \sim d_{\text{model}}\sigma_q^2\sigma_k^2$. Since variance scales quadratically, we divide by $\sqrt{d_{\text{head}}}$. The softmax operation turns the scores after masking into probabilities, along the last dimension.[2] Imagine $z = [z_1, \ldots, z_L] \in \mathbb{R}^L$ is a row vector, then essentially, softmax defines the operation:

$$\sigma(z)_i := \frac{e^{z_i}}{\sum_{j=1}^L e^{z_j}}. \tag{6}$$

   Sometimes we use a numerically stable version to replace it

$$\tilde{\sigma}(z) := \frac{e^{z_i - \max(z)}}{\sum_{j=1}^L e^{z_j - \max(z)}}. \tag{7}$$

   It is worth mentioning that in single head attention (scaled dot product), the complexity of computation is $\mathcal{O}\left(BL^2 d_{\text{model}}\right)$, while for multihead attention, it is the same since we do $\mathcal{O}\left(H \times BL^2 d_{\text{head}}\right) = \mathcal{O}\left(BL^2 d_{\text{model}}\right)$.

[1] Here $\mathbf{S}$ is the stack of $\mathbf{S}_h$ along dimension H.

[2] This is to say that for each $L \times L$ matrix, we softmax every row.

4. Finally, we concatenate and mix attention outputs from all heads. Concatenation involves first transposing $\mathbf{A}_h$ to $\mathbb{R}^{B\times L\times H\times d_{\text{head}}}$ and then merging the last two dimensions into $\mathbf{A}\in\mathbb{R}^{B\times L\times d_{\text{model}}}$. This concatenated result is projected using a matrix $\mathbf{W}_O$, as follows:

$$\text{MHA}(\mathbf{X}) = \mathbf{A}\mathbf{W}_O. \tag{8}$$

   The final output retains the shape $\mathbb{R}^{B\times L\times d_{\text{model}}}$.

There a bunch of reasons why we are using multi heads instead of scaled dot product attention.

- **Diversity of learned attention patterns**: Each head learns different attention patterns in parallel. A single attention head computes only one set of attention scores.
- **Subspace specialization**: Instead of operating in d_{model}, each head projects to a lower dimension subspace d_{head}. This suggests that each head operates in a distinct feature subspace.
- **Improved gradient flow and representation mixing**: Independent paths improve gradient flow and richness of learned representations.

Notice that the compuatational cost are the **SAME**!

Multi Query Attention (MQA) In MQA, different heads have its own query, but share the same key and value. Specifically, for a head h, we have

$$\begin{aligned}\mathbf{S}_h &:= \text{Scores}(\mathbf{Q}_h,\mathbf{K}) = \frac{\mathbf{Q}_h\mathbf{K}^\top}{\sqrt{d_{\text{head}}}} + \mathbf{M}\\ \mathbf{A}_h &:= \text{Attention}_h(\mathbf{Q}_h,\mathbf{K},\mathbf{V}) = \text{softmax}(\mathbf{S}_h)\mathbf{V}, \quad \forall h\in[H].\end{aligned}$$

This means that for each head h, we have a separate $\mathbf{Q}_h\in\mathbb{R}^{B\times 1\times L\times d_{\text{head}}}$ and shared $\mathbf{K},\mathbf{V}\in\mathbb{R}^{B\times 1\times L\times d_{\text{head}}}$.

Compared to standrad MHA, MQA has the following features:

- **Reduced parameter count**: Each head has its own query only, shared key and value.
- **Smaller activation size (memory usage)**: Now $\mathbf{K},\mathbf{V}\in\mathbb{R}^{B\times 1\times L\times d_{\text{head}}}$, so the activation size is smaller.
- **Reduced KV cache (fast inferencing)**: For transformer based models such as GPT, we generate text one token at a time. To avoid recomputing attention over all previous tokens on every step, we cache k, v (key and value vectors) for all previously seen tokens. Specifically, in standard MHA, for each layer and token generated, we neeed $2BHLd_{\text{head}}$ for cached $\mathbf{K},\mathbf{V}$. In MQA, we share $\mathbf{K}$ and $\mathbf{V}$ so that the cost becomes $2BLd_{\text{head}}$.
- **Minimal accuracy loss.** Used in GPT-3.5, PaLM, LLaMA, etc.

KV cache The motivation for KV caching is to enable efficient inference — both in terms of compute time and memory bandwidth. At inference time only, autoregressive models input a sequence of tokens $\{x_t,\ldots,x_{t+L-1}\}$ to generate the next token x_{t+L}. To avoid recomputing key and value vectors for all previous tokens every time, we cache the k, v pairs corresponding to the tokens $x_t\ldots,x_{t+L-1}$ in the forward pass. Then, when generating x_{t+L+1}, we can reuse the vectors for cached $x_{t+1},\ldots,x_{t+L-1}$ and we only need to compute k, v for x_{t+L}. This mechanism is known as the **KV cache**.

Grouped Query Attention (GQA) GQA is like an interpolation between MQA and MHA, where we ask groups of heads to share $\mathbf{K}, \mathbf{V}$. Specifically, let $g(h)$ be a function that maps a head h to its corrsponding group index, then we have

$$\mathbf{S}_h := \text{Scores}(\mathbf{Q}_h, \mathbf{K}_{g(h)}) = \frac{\mathbf{Q}_h \mathbf{K}_{g(h)}^\top}{\sqrt{d_{\text{head}}}} + \mathbf{M}$$
$$\mathbf{A}_h := \text{Attention}_h(\mathbf{Q}_h, \mathbf{K}_{g(h)}, \mathbf{V}_{g(h)}) = \text{softmax}(\mathbf{S}_h)\mathbf{V}_{g(h)}, \quad \forall h \in [H].$$

Benefits:

1. Less memory than MHA.
2. Flexible compute/memory tradeoff by controlling the number of k, v heads.

It is used in LLaMA 2 and Mistral.

Multihead Local Attention (MLA)