ChatGPT

Mathematical Formulation of Transformer Attention

Transformer attention uses **queries**, **keys**, and **values** to compute weighted sums of values. Given an input sequence of N tokens represented by row vectors in a matrix $X\in\mathbb{R}^{N\times d_{\text{model}}}$, we first project X into query, key, and value spaces by learned matrices. Concretely, for one attention head we use weight matrices $W^Q,W^K\in\mathbb{R}^{d_{\text{model}}\times d_k}$ and $W^V\in\mathbb{R}^{d_{\text{model}}\times d_v}$ to form:

- $Q = XW^Q \in \mathbb{R}^{N\times d_k}$ (queries),
- $K = XW^K \in \mathbb{R}^{N\times d_k}$ (keys),
- $V = XW^V \in \mathbb{R}^{N\times d_v}$ (values).

Each row q_i of Q is the **query vector** for token i, each row k_j of K is a **key vector** for token j, and each row v_j of V is a **value vector** for token j [1]. Intuitively, each query q_i "asks" how much attention to pay to each key k_j, and these attention weights are used to form a weighted sum of the corresponding values v_j. This projection step is summarized by Jurafsky & Martin (2025):

> *"For one head we multiply X by the query, key, and value matrices W^Q, W^K, W^V to produce matrices Q, K, V containing all the key, query, and value vectors: $Q=XW^Q, \;K=XW^K,\;V=XW^V$"* [1].

Throughout, d_k is the dimensionality of the key/query vectors and d_v is the dimensionality of the value vectors. In self-attention typically N (sequence length) equals the number of queries and keys.

Scaled Dot-Product Attention

The core of the Transformer's attention mechanism is **scaled dot-product attention**. We compute raw similarity scores between every query and every key by a matrix dot-product. Let $$S = QK^\top\in\mathbb{R}^{N\times N},$$ so that $S_{ij}=q_i\cdot k_j$ is the dot product between query i and key j. To convert these scores into normalized weights, we apply the following steps [2]:

1. **Scale:** Divide the score matrix by $\sqrt{d_k}$:
   $$S' = \frac{QK^\top}{\sqrt{d_k}}\,. $$
   The factor $1/\sqrt{d_k}$ prevents the dot products from growing too large in magnitude (for high-dimensional q_i,k_j), which would make the softmax saturate with very small gradients [3]. Vaswani *et al.* (2017) note that without this scaling the variance of $q_i\cdot k_j$ grows with d_k, so dividing by $\sqrt{d_k}$ keeps values in a more moderate range [3].

2. **Softmax:** Apply the softmax function row-wise to S' to obtain attention weights:
   $$A = \text{softmax}(S')\,,\qquad A_{ij} = \frac{\exp(S'{ij})}{\sum^N \exp(S'{ij'})}\,. $$
   This yields a matrix $A\in\mathbb{R}^{N\times N}$ where each row sums to 1. Softmax converts arbitrary

*scores into a probability distribution over keys for each query. It ensures all weights $A*=1$ [4]. In effect, the largest dot-products get larger weights, but in a }\in[0,1]$ and $\sum_j A_{ij**smooth, differentiable** way (unlike a hard $\arg\max$) [5] [4].

3. **Weighted sum:** Multiply the weight matrix by V:
   $$\text{Attention}(Q,K,V) = A\,V\,,\qquad O = AV\in\mathbb{R}^{N\times d_v}\,. $$
   Here each output row o_i is a weighted sum of value vectors:
   $$o_i = \sum_{j=1}^N A_{ij}\,v_j\,. $$
   In other words, query i attends to all values v_j in proportion to the weight A_{ij}. Jurafsky & Martin (2025) describe this procedure:

   > *"Once we have the $QK^\top$ matrix, we can scale these scores, take the softmax, and then multiply the result by V resulting in a matrix of shape $N\times d$..."* [6].

Putting this together, the **matrix formula** for scaled dot-product attention is given in Vaswani *et al.* (2017) as:
$$ \text{Attention}(Q,K,V) \;=\; \text{softmax}!\Bigl(\frac{QK^\top}{\sqrt{d_k}}\Bigr)\,V\,. $$
This compact equation encapsulates the above steps [2]. Note that the softmax is applied independently to each query's row of scores, so each q_i produces its own weight vector over all keys [7].

> **Scaled Dot-Product Attention (Vaswani *et al.*, 2017)** – For query matrix Q and key matrix K, compute raw scores $QK^\top$. Scale by $1/\sqrt{d_k}$, apply softmax row-wise to get weights, then multiply by V: $$\mathrm{Attention}(Q,K,V) = \mathrm{softmax} \Bigl(\tfrac{QK^\top}{\sqrt{d_k}}\Bigr)V\,. \quad\text{ [2] [7] }$$

Calculating Attention Weights

More explicitly, for each query vector q_i (row i of Q) the attention weight α_{ij} on value v_j is given by:
$$ \alpha_{ij} \;=\; \frac{\exp!\bigl(q_i\cdot k_j/\sqrt{d_k}\bigr)}{\sum_{j'=1}^N \exp!\bigl(q_i\cdot k_{j'}/ \sqrt{d_k}\bigr)}\,. $$
Then the output for query i is $$o_i = \sum_{j=1}^N \alpha_{ij}\,v_j\,. $$ In matrix form this is exactly the ith row of AV. Thus the **attention weight matrix** $A=\mathrm{softmax}(QK^\top/\sqrt{d_k})$ has entries $A_{ij}=\alpha_{ij}$. In summary:

- Compute score vector $s_i = q_i K^\top$ (dot products with all keys).
- Scale $s_i' = s_i/\sqrt{d_k}$.
- Normalize $\alpha_i = \mathrm{softmax}(s_i')$, so $\alpha_i\in\mathbb{R}^N$ sums to 1.
- Form output $o_i = \alpha_i V$.

This procedure ensures each output is a convex combination of the rows of V, weighted by how "relevant" each key is to the query.

Softmax and Scaling Insights

The **softmax** function is crucial because it converts raw dot-product scores into a differentiable probability distribution. By exponentiating and normalizing, softmax emphasizes the largest scores while keeping all weights positive and summing to 1 [4]. The Transformer authors note that softmax is a *continuous, differentiable* alternative to a hard max operation [5]. This smoothness is essential for gradient-based optimization. Moreover, applying softmax row-wise means each query independently attends to keys.

The **scaling factor** $1/\sqrt{d_k}$ arises from variance considerations. Vaswani *et al.* show that if the components of query and key vectors are independent with variance 1, then the dot-product $q_i\cdot k_j$ has variance d_k [3]. Without scaling, large d_k would push $q_i\cdot k_j$ to large magnitudes, driving the softmax into regions with extremely small gradients (saturating the softmax). To avoid this, we divide by $\sqrt{d_k}$ so that the dot products have unit variance on average. In practice, this normalization keeps the softmax inputs at a scale where the exponential function is well-behaved and gradients are stable [3].

In summary, **softmax** ensures a proper probability weighting over keys, and the **$\sqrt{d_k}$ scaling** prevents very large or very small softmax inputs for high-dimensional vectors [3] [4].

Multi-Head Attention

The Transformer improves representational power by using **multi-head attention** [8]. Instead of one attention, we run h parallel "heads," each with its own projection of queries, keys, and values. Concretely:

- We choose h attention heads. For head $i=1,\dots,h$, we have separate learned projections $W_i^Q,W_i^K\in\mathbb{R}^{d_{\text{model}}\times d_k}$ and $W_i^V\in\mathbb{R}^{d_{\text{model}}\times d_v}$.
- Compute head-specific queries/keys/values: $Q_i = XW_i^Q,\;K_i = XW_i^K,\;V_i = XW_i^V$, each of size $N\times d_k$ (for Q_i,K_i) and $N\times d_v$ (for V_i).
- Each head i independently performs scaled dot-product attention: $$\text{head}_i = \text{Attention}(Q_i,K_i,V_i) = \text{softmax}\Bigl(\tfrac{Q_iK_i^\top}{\sqrt{d_k}}\Bigr)V_i.$$
- This yields h output matrices $\text{head}_i\in\mathbb{R}^{N\times d_v}$ [8].
- Concatenate the heads along the feature dimension: $$H = [\text{head}_1;\,\dots;\,\text{head}_h]\in\mathbb{R}^{N\times (h\,d_v)}.$$
- Apply a final linear projection $W^O\in\mathbb{R}^{(h\,d_v)\times d_{\text{model}}}$ to combine heads:
  $$\text{MultiHead}(X) = HW^O \in \mathbb{R}^{N\times d_{\text{model}}}\,. $$

Jurafsky & Martin summarize this as: "we linearly project the queries, keys and values h times with different learned projections… perform the attention function in parallel… concatenate and once again project, resulting in the final values" [9]. In formula form, Vaswani *et al.* give:

$$\begin{aligned} \text{head}_i &= \text{Attention}(QW_i^Q,\ KW_i^K,\ VW_i^V) \quad (i=1,\dots,h),\\ \text{MultiHead}(X) &= [\text{head}_1;\dots;\text{head}_h]\;W^O\,. \end{aligned}$$

More compactly, with $\oplus$ denoting concatenation:
$$ \text{MultiHead}(X) = (\text{head}*1 \oplus \text{head}_2 \oplus \cdots \oplus \text{head}_h)\,W^O\,. $$
This formula is given in Jurafsky & Martin (2025) as Equation (8.37) [10]. Multi-head attention allows the model to attend to information from different representation subspaces; each head can focus on different patterns or*

relations in the input [9] *. After concatenation and projection, the output has the same shape as a single-head output ($N\times d*$), ready to be fed into the next layer of the transformer. }

References: The above formulations follow Vaswani *et al.* (2017) [2] [3] and Jurafsky & Martin (2025) [1] [10]. The use of softmax and scaling is discussed in both sources, with [26] explaining the scaling rationale and [38] providing the standard softmax formulation.

---

[1] [6] [10] web.stanford.edu
https://web.stanford.edu/~jurafsky/slp3/8.pdf

[2] [3] [8] [9] Attention is All you Need
https://papers.neurips.cc/paper/7181-attention-is-all-you-need.pdf

[4] [7] An Intuition for Attention | Jay Mody
https://jaykmody.com/blog/attention-intuition/

[5] Transformer: Attention Is All You Need | Learning-Deep-Learning
https://patrick-llgc.github.io/Learning-Deep-Learning/paper_notes/transformer.html