-->

Why do a downwards then upwards projection in Transformer attention?

Short answer (intuition)

1. **Multiple subspaces / multiple views:** projecting into P smaller subspaces (heads) lets the model learn *different* attention patterns in parallel (e.g. syntax, coreference, locality).
2. **Efficiency & numerical stability:** per-head inner products use dimension d_k (usually smaller), so scaling by $\sqrt{d_k}$ keeps softmax gradients stable and the matmuls fit hardware/cache better.
3. **Mixing:** the upward projection W_O learns how to recombine the P heads into a single d_{model} representation suited for the next layer (and for the residual connection which requires dimension d_{model}).

Notation and typical conventions

- B : batch size
- T : sequence length (queries)
- T_k : sequence length for keys/values (source length for cross-attention)
- d_{model} : model embedding dimension (e.g. 512)
- P : number of heads (a.k.a. n_{heads}, e.g. 8)
- d_k : key / query dimension per head (typ. $d_k = d_{\text{model}}/P$)
- d_v : value dimension per head (often $= d_k$)

Usually $P \cdot d_k = d_{\text{model}}$ and $P \cdot d_v = d_{\text{model}}$.

The math (down → attention → up)

1) Downwards projections (linear)

We start with input

$$X \in \mathbb{R}^{B \times T \times d_{\text{model}}}.$$

For each head p we have learned matrices

$$W_Q^p \in \mathbb{R}^{d_{\text{model}} \times d_k}, \quad W_K^p \in \mathbb{R}^{d_{\text{model}} \times d_k}, \quad W_V^p \in \mathbb{R}^{d_{\text{model}} \times d_v}.$$

Project:

$$\begin{aligned} Q^p &= X W_Q^p \in \mathbb{R}^{B \times T \times d_k}, \\ K^p &= X W_K^p \in \mathbb{R}^{B \times T_k \times d_k}, \\ V^p &= X W_V^p \in \mathbb{R}^{B \times T_k \times d_v}. \end{aligned}$$

(Implementation note: frameworks usually compute a single big linear to get Q, K, V simultaneously, then split into heads.)

2) Attention per head

Reshape/transpose to put head dimension before sequence:

$$Q^p \to \text{shape } (B, P, T, d_k), \quad K^p \to (B, P, T_k, d_k), \quad V^p \to (B, P, T_k, d_v).$$

Compute scaled dot-product scores and values:

$$\text{scores}^p = \frac{Q^p (K^p)^\top}{\sqrt{d_k}} \quad \text{shape } (B, P, T, T_k)$$

$$\text{weights}^p = \text{softmax}(\text{scores}^p + M) \quad \text{shape } (B, P, T, T_k)$$

$$\text{head}_p = \text{weights}^p \; V^p \quad \text{shape } (B, P, T, d_v)$$

3) Upwards projection (concatenate + mix)

Concatenate heads along the feature axis:

$$\text{concat} = \text{Concat}_{\text{heads}}(\text{head}_1, \ldots, \text{head}_P) \quad \text{shape } (B, T, P \cdot d_v)$$

Apply final output projection:

$$W_O \in \mathbb{R}^{(P \cdot d_v) \times d_{\text{model}}}$$

$$\text{Output} = \text{concat } W_O \in \mathbb{R}^{B \times T \times d_{\text{model}}}$$

This output has the same dimension as X, enabling the residual connection:

$$\text{LayerOut} = \text{LayerNorm}(X + \text{Output}).$$

PyTorch-style implementation sketch (shapes)

```
# x: (B, T, d_model)
qkv = qkv_proj(x)                                                # (B, T, 3*
q, k, v = qkv.chunk(3, dim=-1)                                   # each (B,

# reshape into heads
q = q.view(B, T, P, d_k).transpose(1, 2)    # (B, P, T, d_k)
k = k.view(B, T_k, P, d_k).transpose(1, 2)  # (B, P, T_k, d_k)
v = v.view(B, T_k, P, d_v).transpose(1, 2)  # (B, P, T_k, d_v)

scores = torch.matmul(q, k.transpose(-2, -1)) / math.sqrt(d_k)  # (B, P
weights = torch.softmax(scores + mask, dim=-1)                  # (B, P
head_out = torch.matmul(weights, v)                             # (B, P

# combine heads
head_out = head_out.transpose(1, 2).contiguous().view(B, T, P * d_v)  #
out = out_proj(head_out)  # (B, T, d_model)
```

Why this design (detailed reasons)

1. **Representational diversity**
   Each head p has its own W_Q^p, W_K^p, W_V^p so it can learn *a different similarity metric* / focus pattern. Concatenating heads gives the model the ability to represent multiple relations simultaneously.

2. **Numerical stability & scaling**
   Using d_k smaller and scaling by $\sqrt{d_k}$ stabilizes softmax and gradients. If d_k were large, dot products would be large in magnitude causing softmax to be too peaky or gradients to vanish.

3. **Hardware efficiency and locality**

Smaller matmuls per head fit caches better, can map well to parallel compute (multi-GPU / multi-core), and often result in better throughput than a single huge matmul — even though aggregate FLOPs are comparable when $P \cdot d_k = d_{\text{model}}$.

4. **Parameter & capacity trade-off**
   If $P \cdot d_k = d_{\text{model}}$, total parameters for W_Q, W_K, W_V are similar to a single big projection. But splitting into heads gives more flexible parameter structure (separate projections per head) which increases representational capacity for the same overall dimension.

5. **Learned mixing (output projection)**
   W_O is crucial: it learns to combine the different head outputs into a single coherent representation. Without it, you'd have separate head channels that can't interact.

6. **Residual connections require d_{model}**
   The layer output must match d_{model} to add back to the input X in the residual connection. The upward projection ensures that.

Example numeric intuition

- $d_{\text{model}} = 512$, $P = 8$ → $d_k = d_v = 64$.
- Per-head matmul for scores: each head does $\sim T \times T \times 64$ multiplications; across 8 heads this sums to $\sim T \times T \times 512$ — same total FLOPs as one monolithic head of size 512.
  But the split allows diverse attention patterns and better numerical behavior plus practical implementation and parallelism advantages.

Final compact formulas

$$\text{head}_p = \text{Attention}(XW_Q^p,; XW_K^p,; XW_V^p)$$

$$\text{MultiHead}(X) = \text{Concat}(\text{head}_1, \ldots, \text{head}_P); W_O$$

where

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

TL;DR

- **Downwards**: linear projections map d_{model} into multiple small head subspaces (d_k) so each head can learn a different attention pattern and numerical stability is improved.
- **Upwards**: W_O recombines the head outputs into d_{model} so the residual connection and subsequent layers keep a consistent dimension and can use a learned mixture of what each head produced.