Advanced Topic A02: *Attention Mechanisms*

Data Science with Python – BSc Advanced Lectures

Joerg Osterrieder

10 Minutes

Motivation: Why Did Transformers Replace RNNs?

In 2017 a single paper changed everything

- RNNs process sequences one token at a time: slow, hard to parallelise
- Long-range dependencies decay through many recurrent steps
- Vaswani et al. (2017): "Attention Is All You Need" removed the RNN entirely
- Within 5 years, transformers dominated NLP, vision, protein folding, and code

Transformers are not magic: they are differentiable key-value databases

The Core Idea

Every token can attend to every other token simultaneously

- Instead of passing hidden state forward one step at a time, compute all pairwise relationships at once
- Weight each token's contribution by how relevant it is to the current query
- The result: a weighted average of values, where weights reflect similarity
- This is attention: similarity-weighted aggregation

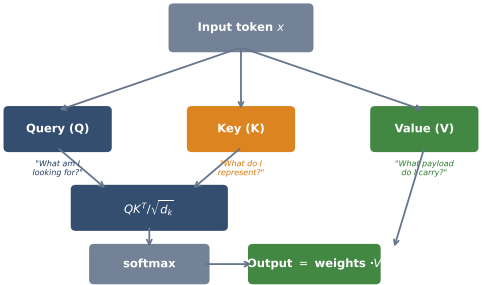
Attention replaces sequential processing with parallel similarity lookup

Query, Key, Value: The Intuition

Three learned projections do the work

- **Query (Q):** "What am I looking for?" (the current token asking a question)
- **Key (K):** "What do I represent?" (every token advertising itself)
- **Value (V):** "What payload do I carry?" (the actual information to aggregate)
- Dot product $Q \cdot K^T$ measures how well a query matches each key

Query, Key, Value: Attention as Differentiable Lookup



Q, K, V are linear projections of the same input: $Q = XW_Q$, $K = XW_K$, $V = XW_V$

The Attention Equation

Scaled dot-product attention in one formula

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

Reading it left to right:

- QK^T: compute all pairwise dot products (similarity scores)
- $/\sqrt{d_k}$: scale to prevent softmax saturation at high dimension
- softmax$(\cdot)$: convert scores to a probability distribution
- $\cdot V$: take weighted average of value vectors

The entire operation is differentiable and can be computed in one matrix multiply

Why Scale by $\sqrt{d_k}$?

Without scaling, softmax saturates and gradients vanish

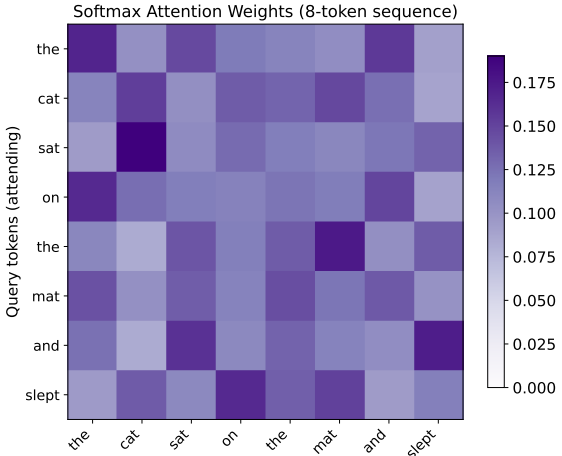
- Dot products $q \cdot k$ have variance proportional to d_k (embedding dimension)
- At $d_k = 512$: typical dot product magnitude ≈ 22; softmax pushes to near one-hot distribution
- Near one-hot softmax: gradient almost zero, learning stalls
- Dividing by $\sqrt{d_k}$ normalises variance to 1 regardless of embedding size

Scaling keeps the attention distribution spread out during early training

Softmax Attention Weights

Visualising pairwise attention scores

- Each row shows how much one token attends to every other token
- Bright cells: strong attention (high similarity between Q and K)
- Rows sum to 1 (probability distribution over positions)



Self-Attention vs Cross-Attention

Same equation, different sources for Q and K/V

Self-attention:

- Q, K, V all from the same sequence
- Used in encoder and decoder stacks
- Captures within-sequence dependencies

Cross-attention:

- Q from decoder, K and V from encoder
- Connects encoder output to decoder
- Enables sequence-to-sequence translation

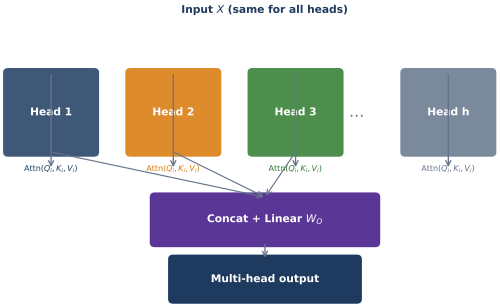
The decoder in a transformer uses both self-attention and cross-attention

Multi-Head Attention

Run h attention heads in parallel, then concatenate

- Split d_{model} into h heads of dimension $d_k = d_{model}/h$
- Each head learns a different type of relationship (syntactic, semantic, positional)
- Concatenate all head outputs, apply a final linear projection
- Typical: $d_{model} = 512$, $h = 8$, $d_k = 64$

Multi-Head Attention: h Parallel Heads



Multi-head attention is cheaper than you think: total cost is the same as one head at full dimension

Complexity: The Quadratic Bottleneck

Attention scales as $O(n^2 d)$ in sequence length n

- Computing QK^T requires n^2 dot products for a sequence of length n
- Memory also scales as $O(n^2)$ for the attention matrix
- At $n = 4096$: $4096^2 \approx 16M$ entries per head per layer
- Motivates efficient variants: Flash Attention, Mamba (state-space models), linear attention approximations

Flash Attention (Dao et al. 2022) reduces memory to $O(n)$ via tiling without approximation

Positional Encodings

Attention is permutation-invariant; order must be injected

- Swapping all tokens produces the same attention output without position info
- Solution: add a positional embedding to each token before attention
- **Sinusoidal (original):** $PE_{pos,2i} = \sin(pos/10000^{2i/d})$
- **RoPE (modern):** rotate Q and K vectors; relative position enters via dot product geometry; used in LLaMA, Gemma, Qwen

Positional encoding is the reason transformers know "first" from "last"

What Attention Heads Actually Learn

Empirical analysis reveals specialisation

- **Syntax heads:** attend to subject-verb pairs, dependency arcs
- **Induction heads:** "if I saw AB before, predict B after A again" (in-context learning mechanism)
- **Previous-token heads:** attend almost entirely to the immediately preceding token
- Not all heads are interpretable; some appear redundant

Elhage et al. (2021): A mathematical framework for transformer circuits

The Transformer Layer Stack

Each layer is: multi-head attention + FFN with residual connections

- **Sub-layer 1:** Multi-head self-attention
- **Sub-layer 2:** Feed-forward network (two linear layers + ReLU or GELU)
- **Both sub-layers:** residual connection + LayerNorm
- Stack L such layers (GPT-3: $L = 96$)

```
x = LayerNorm(x + MultiHeadAttn(x))
x = LayerNorm(x + FFN(x))
```

The residual stream is the primary communication channel across layers

Limitations

Transformers are powerful but not perfect

- Quadratic memory in sequence length: expensive for long documents
- No inherent notion of recurrence or state: must re-read context every time
- Positional encodings struggle to generalise beyond training length
- Sensitive to prompt phrasing; emergent capabilities are hard to predict

Context windows of 1M+ tokens exist but at enormous compute cost

Open Problems

Active research frontiers

- **Linear attention:** approximate softmax(QK^T) in $O(n)$; quality trade-off not yet resolved
- **State-space models (Mamba):** recurrent alternative that matches transformer quality on some tasks with $O(n)$ inference
- **Long context:** RoPE interpolation, YaRN, Infini-Attention
- **Fundamental question:** is the attention mechanism necessary, or is it just a convenient differentiable lookup?

The architecture may still change significantly over the next decade

Code Pointer

Scaled dot-product attention in 10 lines of PyTorch

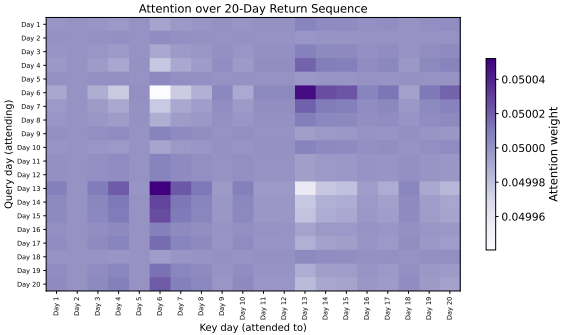
- `torch.nn.functional.scaled_dot_product_attention` (PyTorch 2.0+)
- Or implement manually: `scores = Q @ K.T / math.sqrt(d_k)`
- `weights = torch.softmax(scores, dim=-1)`
- `output = weights @ V`
- Full multi-head: `torch.nn.MultiheadAttention`

PyTorch 2.0 fuses attention into a single CUDA kernel automatically (Flash Attention)

Finance Application: Attention over Return Sequences

Which past day matters most for today's return?

- Treat a window of daily returns as a token sequence
- Self-attention assigns weights to past days based on pattern similarity
- Days with similar volatility signatures receive high attention weight
- Caveat: raw attention weights are not causal explanations (see A03)



Temporal attention is used in FinBERT, time-series transformers, and trading signal models

Interpretability: Attention is Not Explanation

A common misconception

- High attention weight on a token does not mean that token caused the prediction
- Jain & Wallace (2019): attention weights are not always faithful to model decisions
- Wiegreffe & Pinter (2019): counter-argument; context matters
- Safe claim: attention shows what the model *looked at*, not why it decided
- For real explanations, use gradient-based methods (SHAP, A04) or probing

Use attention maps for exploration, not for regulatory compliance

Practical Takeaway

Three things every practitioner should know

- Attention is similarity-weighted averaging: all the complexity is in how Q, K, V are learned
- The quadratic cost is real: benchmark sequence-length scaling before choosing a transformer for long-range tasks
- Positional encodings are a design choice: RoPE generalises better than sinusoidal for long-context applications

When in doubt, use a pretrained transformer and fine-tune; do not train from scratch

Summary and Open Questions

Three things to remember

- Attention computes $\text{softmax}(QK^T/\sqrt{d_k})V$: all tokens attend to all others in $O(n^2)$
- Multi-head attention runs h independent lookups in parallel, each learning a different relational pattern
- Finance use: attention over return sequences; interpret with care (not causal)

Open questions:

1. Is the attention mechanism fundamental or accidental?
2. When does linear attention suffice for financial time-series?
3. Can we attend directly to risk factors in a portfolio?

Further reading: Vaswani et al. 2017; Elhage et al. 2021; Dao et al. 2022