Attention-Based Architectures: Transformers

Handout Notes - Generative and Deep Learning (GDL)

Davide Bacciu - University of Pisa

Notation. A sequence of token embeddings is denoted by $\mathbf{X} = (\mathbf{x}_1, \ldots, \mathbf{x}_N)$, where each $\mathbf{x}_i \in \mathbb{R}^D$. Stacking row-wise gives a matrix $X \in \mathbb{R}^{N \times D}$. Query, key, and value matrices are denoted by Q, K, V. Their per-token vectors are $\mathbf{q}_i, \mathbf{k}_i, \mathbf{v}_i$. The key/query dimension is d_k, and the value dimension is d_v. Attention weights are denoted by α_{ij}. A dataset is denoted by $\mathcal{D}$ with size $N_{\mathcal{D}} = |\mathcal{D}|$. Model parameters are denoted by θ. For self-supervised learning, a corrupted input is denoted by $\tilde{\mathbf{x}}$. For contrastive learning, positive and negative samples relative to $\mathbf{x}$ are denoted by $\mathbf{x}^+$ and $\mathbf{x}^-$, respectively.

1 Why self-attention?

Convolutional and recurrent neural architectures introduced strong and useful inductive biases for structured data, but both also carry important limitations. Convolutional networks rely on local receptive fields; they can represent long-range interactions only indirectly, through many layers. Recurrent models, in contrast, process data sequentially and in principle maintain long-range context, but they are difficult to parallelize and can suffer from vanishing or exploding gradients over long temporal paths.

This motivates a natural question:

> *Can we build a sequence model that captures global dependencies directly, while remaining fully parallelizable across positions?*

Self-attention is one answer to this question. Its core idea is simple: each token representation looks at all the other token representations, decides which ones matter, and then builds a new contextualized embedding as a weighted combination of them. In a sentence such as

"The animal didn't cross the street because it was too tired",

the token corresponding to "it" should attend strongly to "animal", because that is the relevant contextual referent. Self-attention gives the model a mechanism for learning such context-dependent dependencies directly.

The price, however, is that the architecture loses some of the strong inductive bias built into convolution and recurrence. A convolution natively encodes locality; an RNN natively encodes order and causal progression. Self-attention does not. This is both a strength and a weakness. It is more flexible, but it must learn more structure from data.

2 Queries, keys, and values

The fundamental building block of self-attention is the triplet of projections known as *queries*, *keys*, and *values*. Given an input token embedding $\mathbf{x}_i \in \mathbb{R}^D$, we compute three vectors:

$$\mathbf{q}_i = \Omega_q \mathbf{x}_i, \qquad \mathbf{k}_i = \Omega_k \mathbf{x}_i, \qquad \mathbf{v}_i = \Omega_v \mathbf{x}_i,$$

where

$$\Omega_q \in \mathbb{R}^{d_k \times D}, \qquad \Omega_k \in \mathbb{R}^{d_k \times D}, \qquad \Omega_v \in \mathbb{R}^{d_v \times D}$$

are learned linear maps.

The role of these objects is asymmetric:

- the *query* asks what kind of contextual information the current token is looking for;
- the *key* describes what kind of information each token offers;
- the *value* is the actual content that will be aggregated once relevance has been determined.

This separation matters because relevance need not be symmetric. A token may find another token useful even if the reverse is not true. For example, in

"I swam across the river to get to the other bank."

the word "bank" should attend strongly to "river", but "river" need not attend back to "bank" with the same strength. This is why distinct query and key projections are useful.

3 Scaled dot-product self-attention

Once queries, keys, and values have been computed, self-attention proceeds in three steps: scoring, normalization, and aggregation.

3.1 Scoring relevance

For a fixed token i, the model compares its query $\mathbf{q}_i$ with every key $\mathbf{k}_j$ in the sequence. The simplest score is the dot product:

$$e_{ij} = \mathbf{q}_i^\top \mathbf{k}_j.$$

A large dot product means that token j is considered highly relevant when contextualizing token i.

3.2 Scaling the dot product

If the entries of $\mathbf{q}_i$ and $\mathbf{k}_j$ have variance σ^2 and are roughly independent, then the dot product

$$\mathbf{q}_i^\top \mathbf{k}_j = \sum_{\ell=1}^{d_k} q_{i\ell} k_{j\ell}$$

has variance proportional to d_k. More precisely,

$$\mathrm{Var}(\mathbf{q}_i^\top \mathbf{k}_j) \approx d_k \sigma^4.$$

Therefore, as d_k grows, dot products tend to become large in magnitude. Feeding such large values directly into a softmax makes it saturate, which in turn harms gradient propagation.

To compensate, we scale by $\sqrt{d_k}$:

$$e_{ij}^{\text{scaled}} = \frac{\mathbf{q}_i^\top \mathbf{k}_j}{\sqrt{d_k}}.$$

Worked example — Why the $\sqrt{d_k}$ scaling stabilizes softmax

Suppose $\mathbf{q}_i$ and $\mathbf{k}_j$ have i.i.d. zero-mean components with variance σ^2. Then each product $q_{i\ell} k_{j\ell}$ has variance σ^4, and summing over d_k dimensions gives

$$\mathrm{Var}(\mathbf{q}_i^\top \mathbf{k}_j) = d_k \sigma^4.$$

Hence the standard deviation grows like $\sqrt{d_k}\sigma^2$. If we divide by $\sqrt{d_k}$, then

$$\mathrm{Var}\left(\frac{\mathbf{q}_i^\top \mathbf{k}_j}{\sqrt{d_k}}\right) = \sigma^4,$$

which is independent of d_k. Therefore the scores entering the softmax remain on a stable scale as the representation dimension changes.

3.3 Softmax normalization

The scaled scores are turned into attention weights:

$$\alpha_{ij} = \frac{\exp\left(\mathbf{q}_i^\top \mathbf{k}_j / \sqrt{d_k}\right)}{\sum_{\ell=1}^{N} \exp\left(\mathbf{q}_i^\top \mathbf{k}_\ell / \sqrt{d_k}\right)}.$$

For each fixed i, the weights satisfy

$$\alpha_{ij} \geq 0, \qquad \sum_{j=1}^{N} \alpha_{ij} = 1.$$

Thus the weights define a probability distribution over all positions j in the sequence.

3.4 Weighted aggregation

The contextualized representation for token i is then

$$\mathbf{c}_i = \sum_{j=1}^{N} \alpha_{ij} \mathbf{v}_j.$$

This output depends on all tokens, but not equally: it is shaped by the learned attention distribution. Tokens judged more relevant contribute more strongly.

Putting everything together,

$$\mathbf{c}_i = \sum_{j=1}^{N} \frac{\exp\left(\mathbf{q}_i^\top \mathbf{k}_j / \sqrt{d_k}\right)}{\sum_{\ell=1}^{N} \exp\left(\mathbf{q}_i^\top \mathbf{k}_\ell / \sqrt{d_k}\right)} \mathbf{v}_j.$$

Worked example — Self-attention for one token

Suppose token i attends over three tokens and the scaled scores are

$$e_{i1} = 0, \qquad e_{i2} = 2, \qquad e_{i3} = 1.$$

Then

$$\alpha_{i1} = \frac{e^0}{e^0 + e^2 + e^1}, \qquad \alpha_{i2} = \frac{e^2}{e^0 + e^2 + e^1}, \qquad \alpha_{i3} = \frac{e^1}{e^0 + e^2 + e^1}.$$

Numerically,

$$\alpha_{i1} \approx 0.09, \qquad \alpha_{i2} \approx 0.67, \qquad \alpha_{i3} \approx 0.24.$$

So the contextual embedding becomes

$$\mathbf{c}_i \approx 0.09\,\mathbf{v}_1 + 0.67\,\mathbf{v}_2 + 0.24\,\mathbf{v}_3.$$

Hence token i mostly uses information from token 2, while still incorporating some content from the other positions.

4 Matrix form of self-attention

Self-attention is especially elegant in matrix notation. Let

$$X \in \mathbb{R}^{N \times D}$$

be the input sequence matrix. Then

$$Q = XW_Q, \qquad K = XW_K, \qquad V = XW_V,$$

where now

$$W_Q \in \mathbb{R}^{D\times d_k}, \qquad W_K \in \mathbb{R}^{D\times d_k}, \qquad W_V \in \mathbb{R}^{D\times d_v}.$$

The full attention matrix is

$$A = \mathrm{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right),$$

where the softmax is applied row-wise. The output matrix is

$$C = AV.$$

This formula is central:

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

It shows immediately why self-attention is highly parallelizable: all pairwise scores can be computed at once through matrix multiplication.

5 Self-attention as contextual embedding

It is helpful to interpret self-attention as a general contextual-embedding mechanism. Each input token begins with a local representation $\mathbf{x}_i$. After self-attention, it becomes a new representation $\mathbf{c}_i$ that incorporates information from the entire sequence.

Thus self-attention can be seen as a learned context operator:

$$\mathbf{x}_i \mapsto \mathbf{c}_i.$$

Unlike recurrence, which propagates context step by step, or convolution, which propagates context locally across layers, self-attention allows every token to directly interact with every other token in one layer.

This is why it is so effective at modeling long-range dependencies: the distance between two tokens in the attention graph is one, regardless of how far apart they are in the sequence.

6 Multi-head attention

A single attention map may not be rich enough to capture all relevant relationships. For example, in language one head might need to focus on syntactic structure while another focuses on long-range semantic agreement. Multi-head attention addresses this by running several attention mechanisms in parallel.

For head h, define separate projections

$$Q^{(h)} = XW_Q^{(h)}, \qquad K^{(h)} = XW_K^{(h)}, \qquad V^{(h)} = XW_V^{(h)}.$$

Each head computes

$$\mathrm{head}_h = \mathrm{Attention}(Q^{(h)}, K^{(h)}, V^{(h)}).$$

The heads are then concatenated and linearly projected:

$$\mathrm{MultiHead}(X) = \mathrm{Concat}(\mathrm{head}_1, \ldots, \mathrm{head}_H)W_O.$$

The interpretation is that each head learns a different subspace projection and can therefore model a different family of interactions. This makes multi-head attention more expressive than a single large attention computation.

Worked example — Dimension bookkeeping in multi-head attention

Suppose the model dimension is $D = 512$ and we use $H = 8$ heads. A common choice is

$$d_k = d_v = \frac{D}{H} = 64.$$

Then each head maps

$$X \in \mathbb{R}^{N\times 512} \quad \mapsto \quad \text{head}_h \in \mathbb{R}^{N\times 64}.$$

Concatenating all heads yields

$$\text{Concat}(\text{head}_1, \ldots, \text{head}_8) \in \mathbb{R}^{N\times 512},$$

which can then be projected back into the model dimension by W_O. Thus multi-head attention preserves the external dimension while internally splitting computation into several lower-dimensional attentional subspaces.

7 Efficiency variants: grouped-query and multi-query attention

Standard multi-head attention is expressive, but storing separate keys and values for every head is computationally and memory intensive. This has motivated more efficient variants.

- **Multi-head attention (MHA):** each head has its own queries, keys, and values.
- **Multi-query attention (MQA):** many query heads share the same keys and values.
- **Grouped-query attention (GQA):** an intermediate compromise in which groups of query heads share keys and values.

The trade-off is straightforward: full multi-head attention is more expressive, while sharing keys and values is more efficient. In many large models, grouped-query attention offers a good balance between these two objectives.

8 Self-attention is permutation-equivariant

A key conceptual property of self-attention is that, by itself, it does not encode sequence order. If the input tokens are permuted, the attention mechanism permutes accordingly. It treats the input fundamentally as a set or multiset of tokens, not as an ordered sequence.

This is a major difference from recurrence and convolution:

- RNNs are inherently order-sensitive because they process tokens sequentially;
- convolutions are order-sensitive because neighborhood is defined spatially or temporally;
- self-attention alone has no native notion of order.

This is both a strength and a weakness. It makes self-attention generic and fully parallel, but it means that explicit positional information must be injected into the model.

9 Positional encoding

To make self-attention useful for sequences, we modify each input embedding by adding a positional embedding:

$$\tilde{\mathbf{x}}_n = \mathbf{x}_n + \mathbf{r}_n,$$

where $\mathbf{r}_n \in \mathbb{R}^D$ encodes the position n.

A good positional encoding should:

- give distinct representations to different positions,
- remain on a reasonable numerical scale,
- make relative distances expressible,
- generalize to arbitrary sequence lengths.

9.1 Sinusoidal positional encoding

A classical choice is the sinusoidal encoding introduced in the original Transformer:

$$\mathrm{PE}(p, 2i) = \sin\Big(\frac{p}{10000^{2i/D}}\Big),$$

$$\mathrm{PE}(p, 2i+1) = \cos\Big(\frac{p}{10000^{2i/D}}\Big),$$

where:

- p is the position,
- i indexes the embedding dimension pair,
- D is the model dimension.

Low-frequency components vary slowly across positions, while high-frequency components vary more rapidly. This creates a continuous, bounded encoding that supports relative-position reasoning.

Worked example — Why sinusoidal embeddings support relative position reasoning

Consider one sinusoidal pair for frequency ω:

$$r_p = \begin{bmatrix} \sin(\omega p) \\ \cos(\omega p) \end{bmatrix}.$$

Using angle-addition identities,

$$\sin(\omega(p+k)) = \sin(\omega p)\cos(\omega k) + \cos(\omega p)\sin(\omega k),$$

$$\cos(\omega(p+k)) = \cos(\omega p)\cos(\omega k) - \sin(\omega p)\sin(\omega k).$$

Thus the representation at position $p+k$ can be expressed as a linear transformation of the representation at position p, with coefficients depending only on the offset k. This is one reason sinusoidal encodings make relative position information accessible to the model.

10 The Transformer architecture

The Transformer is the first major sequence architecture based entirely on attention, with no recurrence and no convolution in its core sequence-processing blocks. In its original form, it has an encoder–decoder structure, but each component is built from attention and feedforward layers.

10.1 Encoder layer

Given input

$$X \in \mathbb{R}^{N \times D},$$

one encoder block performs:

Step 1: Multi-head self-attention

$$H = \text{MultiHead}(X).$$

Step 2: Residual connection and layer normalization

$$H' = \text{LayerNorm}(X + H).$$

Step 3: Position-wise feedforward network

$$Z = \text{FFN}(H'),$$

where a typical feedforward block is

$$\text{FFN}(\mathbf{x}) = W_2\,\phi(W_1\mathbf{x} + b_1) + b_2,$$

applied independently at every position. Usually W_1 expands the dimension from D to about $4D$, and W_2 projects it back to D.

Step 4: Another residual connection and layer normalization

$$Y = \text{LayerNorm}(H' + Z).$$

Thus one Transformer encoder block computes

$$H' = \text{LN}(X + \text{Attention}(X)), \qquad Y = \text{LN}(H' + \text{FFN}(H')).$$

11 Layer normalization and residual learning

Residual connections are crucial because they create short gradient paths through deep architectures. Layer normalization is used to stabilize internal activations.

Given an input vector $\mathbf{x} \in \mathbb{R}^D$, layer normalization computes

$$\mu = \frac{1}{D}\sum_{i=1}^{D} x_i, \qquad \sigma^2 = \frac{1}{D}\sum_{i=1}^{D}(x_i - \mu)^2,$$

then normalizes and rescales:

$$\text{LayerNorm}(\mathbf{x}) = \gamma \odot \frac{\mathbf{x} - \mu}{\sqrt{\sigma^2 + \varepsilon}} + \beta,$$

where $\gamma, \beta \in \mathbb{R}^D$ are learned parameters.

Unlike batch normalization, layer normalization computes statistics per token representation rather than across a mini-batch. This makes it naturally compatible with variable-length sequence modeling and autoregressive decoding.

12 The Transformer decoder and masked attention

The decoder resembles the encoder, but with one essential difference: when generating token t, the model must not look at future output tokens. This is enforced by *masked self-attention*.

In masked attention, the score matrix is modified by a mask M:

$$\text{softmax}\left(\frac{QK^\top + M}{\sqrt{d_k}}\right),$$

where

$$M_{ij} = \begin{cases} 0 & \text{if } j \le i, \\ -\infty & \text{if } j > i. \end{cases}$$

Thus future positions receive zero probability after the softmax.

The decoder also contains *cross-attention* over encoder outputs. So the original Transformer decoder uses:

- masked self-attention over previous output tokens,
- cross-attention to the encoder representations,
- a position-wise feedforward block.

Worked example — Why the causal mask works

Suppose at decoder position i the raw score vector over positions is

$$(s_{i1}, s_{i2}, \ldots, s_{iN}).$$

To prevent access to future positions, define

$$M_{ij} = -\infty \quad \text{for } j > i.$$

Then the masked softmax is

$$\alpha_{ij} = \frac{\exp((s_{ij} + M_{ij})/\sqrt{d_k})}{\sum_{\ell=1}^{N} \exp((s_{i\ell} + M_{i\ell})/\sqrt{d_k})}.$$

If $j > i$, then

$$\exp((s_{ij} - \infty)/\sqrt{d_k}) = 0,$$

so

$$\alpha_{ij} = 0.$$

Therefore the decoder can attend only to the current and previous output positions, which enforces autoregressive causality.

13 Complexity and gradient paths

One of the most important conceptual advantages of self-attention is that the path length between any two tokens in one layer is constant:

$$\text{maximum path length} = O(1).$$

By contrast:

- in an RNN, the path between distant positions grows linearly with sequence length,
- in a CNN, it grows with depth relative to receptive-field expansion.

This has important implications for gradient propagation. Because signal paths are short, self-attention alleviates the kind of long sequential gradient chains that trouble recurrent models.

However, this comes at a computational cost. Full self-attention computes all pairwise token interactions, which requires:

$$O(N^2 d)$$

operations per layer (up to implementation details and constant factors), and

$$O(N^2)$$

memory for the attention matrix. Thus self-attention trades favorable gradient paths and full parallelism for quadratic dependence on sequence length.

14 Transformers as low-inductive-bias models

Transformers are often described as low-inductive-bias architectures. This means they impose less structural prior knowledge than CNNs or RNNs.

- CNNs assume locality and translation structure.
- RNNs assume sequential dependence and causal propagation.
- Transformers assume mainly that token-token interactions can be learned through attention.

This weaker bias increases flexibility, but it also means the model often needs more data to learn robust generalizations. This is one reason why Transformers achieved their strongest impact in regimes with very large datasets and compute budgets.

15 Vision Transformers

Self-attention is not restricted to text. The Vision Transformer (ViT) applies the Transformer encoder to images by turning them into sequences of patches.

Let an image be split into patches, each flattened and linearly projected into an embedding vector. These patch embeddings are then treated as tokens:

$$(\mathbf{x}_1, \dots, \mathbf{x}_N).$$

Positional embeddings are added, and the resulting sequence is fed through a Transformer encoder.

For classification, a common trick is to prepend a learned special token, often written `[CLS]`. After the encoder, the final representation of this token is used as the image-level summary for prediction.

Worked example — Patch embedding in a Vision Transformer

Suppose an image has size

$$224 \times 224 \times 3,$$

and is split into patches of size

$$16 \times 16.$$

Then the number of patches is

$$\frac{224}{16} \cdot \frac{224}{16} = 14 \cdot 14 = 196.$$

Each patch contains

$$16 \cdot 16 \cdot 3 = 768$$

numbers. If each flattened patch is projected into a D-dimensional embedding, then the image becomes a sequence

$$X \in \mathbb{R}^{196 \times D}.$$

If a special classification token is added, the sequence length becomes

$$197.$$

Thus ViT converts image classification into sequence processing over patch tokens.

This highlights a general theme: the Transformer is a generic architecture for sets or sequences of embeddings, and different modalities can often be recast into that form.

16 Self-supervised learning

A major reason Transformers became so powerful is not only their architecture, but also the training paradigm that often accompanies them: self-supervised learning.

The key idea is to create supervision from the data itself, without manual labels. Given unlabeled data $\mathbf{x} \sim p(\mathbf{x})$, one defines a pretext objective such as

$$L(\theta) = \mathbb{E}_{\mathbf{x}}[\ell(f_\theta(\tilde{\mathbf{x}}), \mathbf{x})],$$

where $\tilde{\mathbf{x}}$ is a corrupted version of $\mathbf{x}$. The model is trained to reconstruct or predict the missing/corrupted information.

A common approach is masking. For example, one can mask some positions of the input and train the model to infer them from the unmasked part. This is especially natural for attention-based models, because attention gives each token direct access to the rest of the context.

17 Autoregressive self-supervision

Another important self-supervised paradigm is autoregressive prediction. Instead of reconstructing masked tokens anywhere in the input, the model predicts the next token from all previous ones:

$$p(x_t \mid x_1, \ldots, x_{t-1}).$$

This uses the same causal masking principle as the decoder of the original Transformer.

Autoregressive training is especially prominent in generative language modeling. It is self-supervised because the target token is already present in the raw sequence; no external label is required.

18 Contrastive learning

A second major family of self-generated supervision is contrastive learning. Instead of reconstructing corrupted data directly, contrastive learning trains an encoder to bring similar samples closer and push dissimilar ones apart in representation space.

Given an encoder f_θ, let

$$\mathbf{z} = f_\theta(\mathbf{x}), \qquad \mathbf{z}^+ = f_\theta(\mathbf{x}^+), \qquad \mathbf{z}^- = f_\theta(\mathbf{x}^-),$$

where $\mathbf{x}^+$ is a positive sample related to $\mathbf{x}$ and $\mathbf{x}^-$ is a negative sample. The objective is to enforce

$$\mathrm{sim}(\mathbf{z}, \mathbf{z}^+) \gg \mathrm{sim}(\mathbf{z}, \mathbf{z}^-),$$

for a chosen similarity function sim, often the dot product or cosine similarity.

18.1 InfoNCE objective

A standard contrastive loss is InfoNCE:

$$L_{\mathrm{InfoNCE}}(\mathbf{x}) = -\log \frac{\exp(\mathrm{sim}(\mathbf{z}, \mathbf{z}^+)/\tau)}{\exp(\mathrm{sim}(\mathbf{z}, \mathbf{z}^+)/\tau) + \sum_{\mathbf{x}^- \in \mathrm{Neg}(\mathbf{x})} \exp(\mathrm{sim}(\mathbf{z}, \mathbf{z}^-)/\tau)},$$

where:

- τ is a temperature parameter,
- $\mathrm{Neg}(\mathbf{x})$ is a set of negative samples.

This can be interpreted as a softmax classification objective in which the positive pair should win against the negatives.

Worked example — InfoNCE as "classify the positive among negatives"

Suppose one anchor representation $\mathbf{z}$ is compared with one positive $\mathbf{z}^+$ and two negatives $\mathbf{z}_1^-, \mathbf{z}_2^-$. Then

$$L_{\text{InfoNCE}} = -\log \frac{\exp(\text{sim}(\mathbf{z}, \mathbf{z}^+)/\tau)}{\exp(\text{sim}(\mathbf{z}, \mathbf{z}^+)/\tau) + \exp(\text{sim}(\mathbf{z}, \mathbf{z}_1^-)/\tau) + \exp(\text{sim}(\mathbf{z}, \mathbf{z}_2^-)/\tau)}.$$

If the similarity with the positive is much larger than with the negatives, then the numerator dominates and the loss is small. If a negative has high similarity, then it competes strongly in the denominator and the loss increases. Thus minimizing InfoNCE simultaneously attracts positives and repels negatives.

19 Why self-supervision and Transformers fit well together

Transformers and self-supervised learning reinforce each other. Transformers are expressive but low-bias models that benefit from large-scale pretraining. Self-supervised objectives provide exactly the kind of large-scale training signal needed to shape such expressive models before task-specific fine-tuning.

This is true across modalities:

- in language, masked-token prediction and autoregressive next-token prediction,
- in vision, masked patch prediction and contrastive representation learning,
- in multimodal systems, alignment and cross-modal matching objectives.

Thus the Transformer revolution is as much about training strategy as it is about architectural design.