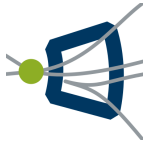


Data Analysis Block Course: Statistics and Machine learning



Teacher Answer Key: Transformers by Hand

Prof. Dr. Mikhail Mikhasenko

Assistants: Hendric Jonas, Christoph Tönnis



30 September 2026

Indico schedule

RUB Moodle

Part I - Embeddings

Exercise 1.1 - The embedding matrix

1. $E \in \mathbb{R}^{6 \times 4}$: $V = 6$ rows, $d = 4$ columns.
2. $6 \times 4 = 24$ trainable parameters.
3. A one-hot vector $e_i \in \{0, 1\}^V$ has a 1 only at position i . Multiplying $e_i^\top E$ sums each row of E weighted by the corresponding entry of e_i ; since only entry i is nonzero, the product equals row i of E exactly.
4. Feeding the raw integer index into a dense layer would impose an arbitrary numeric/ordinal relationship between unrelated words (e.g. index 5 being "closer" to index 4 than to index 0), which has no linguistic meaning. A learned embedding lets the network place semantically related words near each other in a learned space instead.

Exercise 1.2 - Embedding a sentence

1. The embedded sequence is

$$X = \begin{pmatrix} 1 & 0 & 0 & 1 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 2 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{pmatrix}$$

(rows: the, cat, sat, on, the, mat)

2. $X \in \mathbb{R}^{6 \times 4}$, i.e. $L \times d$ with $L = 6$, $d = 4$.
3. Batched: $(8, 6, 4)$, i.e. (B, L, d) .
4. Yes, both occurrences of "the" give the identical row $(1, 0, 0, 1)$, because the lookup depends only on the token identity, not its position. This alone cannot distinguish the two occurrences of "the" in the sentence - that is exactly the gap positional encoding fills (Exercise 1.3).

Exercise 1.3 - Why attention needs position information

1. If we permute the tokens (and the padding mask) with the same permutation π , every pairwise comparison $q_i \cdot k_j$ that existed before still exists after, just relabelled by π : the comparison between the *same two tokens* still happens, only its row/column index in the score matrix changes.
2. No. Plain self-attention is **permutation equivariant**: permuting the input tokens permutes the output the same way, but the *content* of each token's output (given its neighbours) is unchanged. It carries no information about the original order.
3. E.g. "the dog bit the man" vs. "the man bit the dog" - identical bag of words, opposite meaning.
4. A positional encoding is added to the embedding **before** it is compared via attention, and it depends only on the position t , not on which word is there. Two identical words at different positions now receive different vectors (word embedding + different positional vector), so the query/key comparisons - and hence attention output - become sensitive to where each token sits in the sequence.

Part II - Scaled dot-product attention by hand

Exercise 2.1 - A three-token example

Step 1:

$$s_{\text{the}} = (0, 2, 0, 0) \cdot (2, 0, 0, 0) = 0$$

$$s_{\text{cat}} = (0, 2, 0, 0) \cdot (0, 2, 0, 0) = 4$$

$$s_{\text{sat}} = (0, 2, 0, 0) \cdot (0, 0, 2, 0) = 0$$

Step 2: dividing by $\sqrt{4} = 2$:

$$\tilde{s}_{\text{the}} = 0 \quad \tilde{s}_{\text{cat}} = 2 \quad \tilde{s}_{\text{sat}} = 0$$

Step 3:

i	$\tilde{s}_i$	$e^{\tilde{s}_i}$	a_i
the	0	1.00	0.107
cat	2	7.39	0.787
sat	0	1.00	0.107

Sum: $1.00 + 7.39 + 1.00 = 9.39$; weights sum to $0.107 + 0.787 + 0.107 \approx 1.00$, as required.

Step 4:

$$\begin{aligned} z_{\text{cat}} &\approx 0.107(1, 0) + 0.787(0, 1) + 0.107(1, 1) \\ &= (0.107 + 0.107, 0.787 + 0.107) \\ &\approx (0.214, 0.894) \end{aligned}$$

Step 5:

1. "cat" attends most strongly to **itself**, since $q_{\text{cat}} \cdot k_{\text{cat}} = 4$ is the largest of the three dot products - its query and key vector happen to be identical here, so it is maximally "similar to itself".

2. Yes, the weights would change. Scaling every key by c scales every score $q \cdot k$ by c as well, and softmax is **not** invariant to a uniform rescaling of its inputs (multiplying all logits by $c > 1$ sharpens the distribution towards the largest score; $c < 1$ flattens it).
3. If entries of q and k are $\mathcal{O}(1)$ and roughly independent, the dot product $q \cdot k = \sum_{m=1}^{d_k} q_m k_m$ is a sum of d_k roughly independent terms, so its variance grows linearly with d_k . Without rescaling, scores would grow in magnitude as d_k increases, pushing softmax into a saturated, near-one-hot regime with very small gradients. Dividing by $\sqrt{d_k}$ keeps the variance of the scores approximately constant regardless of d_k .

Exercise 2.2 - Shapes of scaled dot-product attention

quantity	shape
Q	(B, L, d_k)
K	(B, L, d_k)
V	(B, L, d_v)
$QK^\top$	(B, L, L)
A	(B, L, L)
$Z = AV$	(B, L, d_v)

1. No - the shape of Z is (B, L, d_v) ; it depends on d_v , not d_k . d_k only affects the (square) shape of the intermediate score matrix.
2. $Q = XW_Q$ with $X \in \mathbb{R}^{B \times L \times d_{\text{model}}}$ and $W_Q \in \mathbb{R}^{d_{\text{model}} \times d_k}$ (applied identically to every token/batch element).

Exercise 2.3 - Multi-head attention dimensionality

1. $d_k = 8/2 = 4$.
2. $(B, L, d_k) = (1, 5, 4)$ per head.
3. $(1, 5, 4)$ per head.
4. Concatenated: $(1, 5, 8)$.
5. After output projection: $(1, 5, 8)$ - the output projection maps $d_{\text{model}} \rightarrow d_{\text{model}}$, so the model dimension is preserved.
6. Concatenating h heads of width d_k must reconstruct exactly d_{model} ; if d_{model} were not divisible by h , there would be no way to split it into h equal-width heads (or one head would need a different width, breaking the uniform per-head computation).
7. With $h = 4$, $d_k = 8/4 = 2$ per head - it halves. Each head now attends over the same L keys as before (the *number* of values attended per head is unchanged), but each head represents its query/key/value comparisons in a narrower (2-dimensional instead of 4-dimensional) subspace; the total representational budget $d_{\text{model}} = 8$ is split more finely across more, narrower heads.

Part III - Encoder and decoder blocks

Exercise 3.1 - Labelling a transformer encoder layer

Diagram: $X \rightarrow \text{Multi-Head Self-Attention} \rightarrow (+X) \rightarrow \text{Add \& Norm} \rightarrow \text{Feed-Forward} \rightarrow (+\text{previous output}) \rightarrow \text{Add \& Norm} \rightarrow \text{output}$.

1. A residual connection means the sub-layer's *input* is added elementwise to its *output* before normalisation: $X' = X + \text{Sublayer}(X)$.
2. It gives gradients a direct, unimpeded path (an identity shortcut) back to earlier layers, so they do not have to pass through every nonlinear transformation at every stacked layer. This is the same reasoning as for skip connections in deep ReLU networks: it mitigates vanishing gradients and makes very deep stacks trainable.
3. Bidirectional attention lets every token use both left and right context, which is exactly what is needed to understand the meaning of a whole, already-written sentence. For left-to-right generation, however, a token must be predicted using only what came before it; allowing it to see future tokens during training would let the model trivially "cheat" by copying the answer, and that information would not exist yet at inference time.

Exercise 3.2 - A decoder layer and causal masking

query i key j	$j = 0$	$j = 1$	$j = 2$	$j = 3$
$i = 0$	True	False	False	False
$i = 1$	True	True	False	False
$i = 2$	True	True	True	False
$i = 3$	True	True	True	True

1. Setting disallowed scores to $-\infty$ before softmax guarantees that, after softmax, the corresponding weight is exactly 0 **while the remaining allowed weights still form a valid probability distribution that sums to 1** over exactly the allowed keys. If instead the weights were zeroed *after* an unmasked softmax, they would no longer sum to 1 (an invalid distribution) and the masked positions would still have influenced the softmax normalisation and gradients during the forward/backward pass.
2. In an encoder-decoder model, the entire source sequence is already available and fully encoded before generation starts. Cross-attention lets every decoder step look at the *complete* encoder output at once - there is no notion of "the future part of the source hasn't been generated yet", so no causal restriction is needed on that step (only the decoder's own self-attention over its own, still-being-generated output needs the causal mask).
3. The **decoder-only** model (GPT-style) uses the causal mask, because it is trained to predict each token from only the tokens before it (next-token prediction), matching how it must generate text at inference time. The **encoder-only** model (BERT-style) does not use a causal mask, because it is trained with a bidirectional objective (e.g. predicting masked tokens using both left and right context) and is not used to generate text autoregressively.

Exercise 3.3 - Full dimensionality and parameter count

1. Attention projections: $4 \times 512^2 = 4 \times 262,144 = 1,048,576$.
2. Feed-forward: $2 \times 512 \times 2048 = 2,097,152$.
3. Total per layer: $1,048,576 + 2,097,152 = 3,145,728 \approx 3.1\text{M}$.

4. For $N = 6$ layers: $6 \times 3,145,728 = 18,874,368 \approx 18.9\text{M}$.
- 5.
- (a) embedding output: $(32, 50, 512)$
 - (b) after the multi-head attention sub-layer: $(32, 50, 512)$
 - (c) feed-forward hidden representation: $(32, 50, 2048)$
 - (d) encoder layer output: $(32, 50, 512)$
6. No, the parameter count does **not** depend on L : all weight matrices (W_Q, W_K, W_V, W_O and the two feed-forward matrices) have fixed shapes independent of sequence length, and are applied/shared across all positions. The **computation and memory** do depend on L : the attention score matrix A has shape (B, h, L, L) , so both its memory footprint and the cost of computing it scale **quadratically** in L ; the feed-forward network's cost scales linearly in L (applied independently per position).

Exercise 3.4 - Matching architectures to tasks

task	architecture	justification
Sentiment classification of a movie review	encoder-only	Needs one representation summarising the whole (already complete) input using bidirectional context; no generation is required.
Autoregressive next-word text generation	decoder-only	Must produce tokens one at a time, each conditioned only on the tokens generated so far - exactly what causal self-attention provides.
Machine translation (English $\rightarrow$ German)	encoder-decoder	The source sentence benefits from full bidirectional encoding, while the target must be generated left-to-right, conditioned on the source via cross-attention.
Producing a single embedding vector to search a document database	encoder-only	Needs one fixed-size, context-aware representation of the full (complete) input; no autoregressive generation involved.

Summary answers

Statement	Answer
An embedding matrix has shape	vocabulary size $\times$ embedding dimension
Plain self-attention, without extra input, is	permutation equivariant / order-blind
Scores are divided by	$\sqrt{d_k}$, to keep score variance roughly constant as d_k grows
Per-head key/query dimension	$d_k = d_{\text{model}}/h$
What is added to token embeddings to restore order information	positional encodings
What masking technique makes a decoder generate strictly left-to-right	causal (look-ahead) masking , applied by setting disallowed scores to $-\infty$ before softmax
Cross-attention is used in	encoder-decoder architectures, letting the decoder attend to the full encoder output
Transformer weight-matrix parameter counts scale with	d_{model} and d_{ff} , not with sequence length L
Attention compute/memory scales with	L^2 (quadratically in sequence length)

Central teaching point

Attention mixes information across positions and is order-blind by itself.

Positional information and causal masking restore order information and, for decoders, the generation direction.

Every shape in a transformer layer is fixed by three numbers - d_{model} , h (hence $d_k = d_{\text{model}}/h$) and d_{ff} - and is otherwise independent of sequence length; only the *cost* of attention grows with L , quadratically. This is the central bookkeeping students should be able to reproduce for any given configuration.

Question answers (short conceptual questions)

1. Self-attention compares queries and keys symmetrically with no notion of "where" a token sits, so permuting the input tokens simply permutes the output the same way - the computation itself carries no order information. This is fixed by (i) adding position-dependent positional encodings to the input embeddings (so token identity and position both matter to the encoder), and (ii) causal masking in the decoder, which enforces that generation happens strictly in left-to-right order.
2. Queries, keys and values play three different functional roles - "what this token is looking for" (query), "what this token offers to be matched against" (key), and "what this token hands over once matched" (value). A single shared representation could not simultaneously be optimised for all three roles; separate learned linear projections let the model shape the same underlying embedding differently for each purpose.
3. $d_k = d_{\text{model}}/h$ grows if d_{model} increases with h fixed. A larger d_k gives each head a higher-dimensional subspace to represent its query/key/value comparisons in, i.e. more capacity per head to encode finer-grained or more complex relationships.

4. Layer normalisation computes its mean/variance over the feature dimension of a single token, independent of the batch and of other tokens, so it behaves identically regardless of batch size or sequence length and remains well-defined even for variable-length sequences or a batch size of 1. Batch normalisation's statistics are computed across the batch, which is unstable for small batches and does not transfer sensibly across positions in a sequence of varying length.
5. If a decoder used bidirectional self-attention during training, position i could directly see the token it is being trained to predict (since that token is present in the full, unmasked sequence), so the model could trivially copy it and the training loss would collapse without the model ever learning the actual predictive relationship. At inference time, however, future tokens do not exist yet, so a model trained this way would fail catastrophically compared to training.
6. Only the self-attention sub-layer mixes information *between* different token positions. The feed-forward network is applied identically and independently to each position's own vector (point-wise), so it can transform a token's representation but cannot exchange information with any other token.