



Data Analysis Block Course: Statistics and Machine learning



Transformers by Hand

Prof. Dr. Mikhail Mikhasenko

Assistants: Hendric Jonas, Christoph Tönnis

30 September 2026

Indico schedule

RUB Moodle

We work with a tiny vocabulary of six words:

$\{\text{"the"}, \text{"cat"}, \text{"sat"}, \text{"on"}, \text{"mat"}, \text{"dog"}\}$

Each word is represented by an integer index $0, \dots, 5$ (in the order above).

Part I - Tokens and embeddings

Exercise 1.1 - Subword tokens

Real language models don't tokenize whole words like our toy vocabulary above - they use **byte-pair encoding (BPE)**. To find tokens, BPE starts from single bytes/characters, repeatedly finds the most frequent adjacent pair in a large training texts and merges it into a new symbol. The procedure is repeated until the vocabulary reaches a target size (GPT-2: 50,257 tokens, learned once, before training the network).

1. Why not tokenize by individual character?

What would the model have to learn "for free" that it wouldn't need to with larger units?

2. Why not tokenize by whole word instead?

What happens to a typo, a rare name, or a word from another language under a word-level vocabulary?

3. Why does BPE avoid both problems from 1. and 2.?
4. Why does this make a task like "count the letters r in *strawberry*" surprisingly hard for a language model?

How does this hold up against current models? Try questions 2 and 4 against a current frontier model (e.g. Claude, GPT-5, Gemini) and compare to what you argued above.

Exercise 1.2 - The embedding matrix

In our 6 word example, an embedding layer stores one vector of length $d = 4$ per vocabulary entry, collected into a matrix $E \in \mathbb{R}^{6 \times 4}$.

1. How many trainable parameters does this embedding layer have? GPT-2 uses $V = 50,257$ and $d = 768$ - how many parameters is that, and how does it compare to GPT-2's total of 124M parameters?
2. The embedding matrix E contains a set of trainable parameters, typically randomly initialized and then updated by backpropagation together with the rest of the network. What would a "bad" set of embeddings look like at initialization? How does training change what nearby rows of E come to represent?

3. The embedding dimension d is a hyperparameter chosen by the model designer, independent of vocabulary size. What goes wrong if d is chosen too small (relative to the vocabulary and the complexity of the task)? What goes wrong if d is chosen too large?
4. After training on real text, nearest neighbours in embedding space cluster meaningfully (e.g. nearest to Monday: Tuesday, Wednesday, Thursday; nearest to electron: photon, neutron), and vector arithmetic on the embeddings captures relationships: $\text{king} - \text{man} + \text{woman} \approx \text{queen}$. What does this tell you about what the embedding space comes to represent?
5. The word "the" appears twice in the sentence "the cat sat on the mat". Are its two embedding vectors identical after the lookup? Explain why this alone is not enough for the model to distinguish the two occurrences.

Part II - Scaled dot-product attention by hand

Each attention head turns every token's (position-encoded) embedding vector - the same vectors produced in Part I, now collected row-wise as the input sequence X - into a query, key and value vector with (tunable) length d_k and d_v , via a learned linear projection. Collecting these row-wise across the sequence gives

$$A = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right) \quad Z = AV$$

A is the matrix of **attention weights**: entry a_{ij} says how much query i attends to key j , and softmax is taken **row-wise** (separately for each query, over the key dimension), so every row of A sums to 1. Z is the **attention output**: row i of Z is the weighted average of all value vectors using row i of A as weights - i.e. token i 's new representation, built by mixing in information from the tokens it attends to.

Exercise 2.1 - From embeddings to queries, keys and values

The input to one attention head is $X \in \mathbb{R}^{B \times L \times d_{\text{model}}}$: a batch of B token sequences of length L , exactly the (position-encoded) embedded sequences from Part I, so d_{model} is the same dimension as d in Part I's embedding matrix E (Exercise 1.2).

Complete the table with tensor shapes, in terms of B , L , d_k and d_v :

quantity	shape
Q	
K	
V	
$QK^\top$	
$A = \text{softmax}(QK^\top / \sqrt{d_k})$	
$Z = AV$	

1. Does the shape of Z depend on d_k ?
2. Q , K and V are each produced from X by a learned linear projection - one weight matrix per query/key/value, applied to every token's embedding vector the same way. Call these matrices W_Q , W_K , W_V . What is the shape of W_Q , in terms of d_k and d_{model} ? What about W_K and W_V , in terms of d_v and d_{model} ?

Exercise 2.2 - A three-token example

Consider the three tokens "the", "cat", "sat" with $d_k = 4$ key/query vectors and $d_v = 2$ value vectors:

token	key k_i	value v_i
the	(2, 0, 0, 0)	(1, 0)
cat	(0, 2, 0, 0)	(0, 1)
sat	(0, 0, 2, 0)	(1, 1)

We compute the output for the query of token "cat", $q_{\text{cat}} = (0, 2, 0, 0)$.

Step 1 - Raw scores

Calculate the three dot products

$$s_i = q_{\text{cat}} \cdot k_i \quad i \in \{\text{the, cat, sat}\}$$

Step 2 - Scale

Divide each score by $\sqrt{d_k}$.

Step 3 - Softmax

Use $e^0 = 1$ and $e^2 \approx 7.39$. Compute

$$a_i = \frac{e^{\tilde{s}_i}}{\sum_j e^{\tilde{s}_j}}$$

Complete the table:

i	$\tilde{s}_i$	$e^{\tilde{s}_i}$	a_i
the			
cat			
sat			

Check that your three weights sum to 1.

Step 4 - Weighted output

Calculate

$$z_{\text{cat}} = a_{\text{the}} v_{\text{the}} + a_{\text{cat}} v_{\text{cat}} + a_{\text{sat}} v_{\text{sat}}$$

Step 5 - Interpretation

1. Which token does "cat" attend to most strongly? Relate this to the dot product $q_{\text{cat}} \cdot k_{\text{cat}}$.
2. Would the result change if we scaled every key vector by the same positive constant c before the scaled dot product? Would it change the *weights* a_i ?
3. Why do we divide by $\sqrt{d_k}$ rather than, say, d_k or not scaling at all? Think about what happens to the scores as d_k grows, if q and k have entries of typical size $\mathcal{O}(1)$.
4. Suppose the sentence continues: "the cat sat on the mat", adding a fourth token "on", and a fifth one, "mat", with key vector $k_{\text{mat}} = (1, 1, 1, 0)$. Compute $q_{\text{cat}} \cdot k_{\text{mat}}$ and compare it to $q_{\text{cat}} \cdot k_{\text{cat}}$ from Step 1. Would "cat" now attend more to itself or to "mat"? What could

a large query-key dot product between "cat" and "mat" represent, semantically, once these vectors are learned from real text rather than handed to you?

Exercise 2.3 - Why attention needs position information

In Exercise 2.2, the attention weights a_i came only from comparing content vectors q and k - nothing in that computation referred to *where* in the sentence a token sits.

1. Take a sentence and reorder its tokens by some permutation π (e.g. "the cat sat" $\rightarrow$ "sat the cat"). Argue informally that the resulting attention weights are exactly the old ones, just relocated.
2. Based on 1., is plain self-attention sensitive to the order of the input tokens?
3. For fun: Give one concrete example of a sentence pair where the two sentences contain the same words but the order changes the meaning.
4. To restore order-sensitivity, a **positional encoding** vector $\mathbf{p}_t$, depending only on the position $t = 0, 1, 2, \dots$, is added to each token's embedding: $\mathbf{x}_t = \mathbf{e}_{\text{token}(t)} + \mathbf{p}_t$. Why does this fix the problem identified in 2.?
5. Different models encode position differently:

model	position encoding
GPT-2 (2019)	a learned vector per position, up to a maximum length (e.g. 1024)
Llama and most models since 2023	RoPE: rotate $\mathbf{q}$ and $\mathbf{k}$ by an angle proportional to t

What is one practical limitation of GPT-2's learned-position-vector approach that a fixed or rotation-based scheme does not share? What happens if you feed the model a sequence longer than any it was trained on?

6. (Optional/advanced) RoPE is designed so that the attention score between two tokens depends on the *distance* $t_i - t_j$ between them rather than on their absolute positions. Why might that be a more useful property for a model that needs to generalize to texts of different lengths than it saw during training? Context: concretely, pair i of the head's dimensions is rotated by $t \cdot \text{base}^{-2i/d_k}$; the base was 10,000 in the original RoPE paper, but many recent long-context models use a much larger base - e.g. Llama 3 uses 500,000 - to slow down the fastest-rotating dimension pairs and support longer context windows

Exercise 2.4 - Multi-head attention dimensionality

A multi-head attention layer with h heads and model dimension d_{model} splits Q, K, V into h smaller pieces, each of dimension

$$d_k = d_v = \frac{d_{\text{model}}}{h}$$

runs scaled dot-product attention independently per head, concatenates the h outputs, and applies one more linear projection back to d_{model} via an **output projection matrix** W_O .

Take $d_{\text{model}} = 8$, $h = 2$, $L = 5$, $B = 1$. For scale: GPT-2 small has $d_{\text{model}} = 768$, $h = 12$; GPT-3 has $d_{\text{model}} = 12,288$, $h = 96$. Many recent large models, e.g. Llama 3 70B ($d_{\text{model}} = 8192$, $h = 64$ query heads), additionally use *grouped-query attention*: several query heads share one key/value head, keeping Q at full size while shrinking K and V - this is done specifically to reduce the KV cache from Exercise 3.3, Question 6.

1. What is d_k per head?
2. What is the shape of Q, K, V **per head**?
3. What is the shape of the attention output Z_{head} per head?
4. What is the shape after concatenating all h heads?
5. What is the shape after the final output projection, and what is the shape of W_O itself?
6. Why must d_{model} be divisible by h ?
7. If we increased h from 2 to 4 while keeping $d_{\text{model}} = 8$ fixed, what happens to d_k per head? Does the total number of attended values change?

Exercise 2.5 - Induction heads (Optional/advanced)

GPT-2 was shown a passage of random, unrelated words, then the *same* words again, in the same order. Its per-token prediction loss dropped from 11.8 nats (first time - unpredictable) to 0.73 nats (second time - essentially copied), despite never seeing these specific words during training. A nat is a unit of information: the cross-entropy loss $-\ln p(\text{token})$ is measured in nats because it uses the natural log $\ln$ rather than $\log_2$; 1 nat $\approx$ 1.443 bits. Lower loss means the model assigned higher probability to the token that actually appeared.

This is explained by a two-head circuit found inside the model: 1. a **previous-token head**, which writes into each token's representation "the word before me was A"; 2. an **induction head**, which, at the *current* token B, searches earlier positions whose *previous* token was also B, and copies whatever followed it there.

Pattern: "... A B ... A $\rightarrow$ " predicts B.

1. Which sub-layer of the transformer block (attention or MLP) is responsible for each of these two steps, and why does it have to be that one? *Hint: which sub-layer moves information between token positions?*
2. Why does this circuit require *two* attention layers working in sequence - one head's output feeding into a head in a later layer - rather than a single attention layer?
3. Why is this a form of "learning inside the prompt" (in-context learning), rather than a form of learning that requires updating any weights?

Part III - Transformer blocks, decoding, and scale

Exercise 3.1 - The transformer block and causal masking

A transformer block consists of, in order:

1. layer normalisation, followed by multi-head self-attention over the (normalised) input sequence *tokens exchange information*,
2. a residual (skip) connection: the block's original input (from *before* step 1) is added to the attention output,
3. layer normalisation, followed by a position-wise MLP *two linear layers with a nonlinearity in between, applied to each token independently*,
4. another residual connection: the input to step 3 is added to the MLP output.

A **decoder** block (used for left-to-right text generation, e.g. GPT) additionally uses a **causal mask** in step 1: token i is only allowed to attend to tokens $j \leq i$.

1. Draw this as a block diagram with boxes and arrows, labelling the input X , the intermediate tensors, and the two "Add & Norm" steps.
2. What does "residual connection" mean here, concretely, in terms of a tensor addition?

3. Why is a residual connection useful for training deep stacks of these blocks? Connect this to what you learned about backpropagation through many layers.
4. In practice, the causal mask is implemented by setting the disallowed scores to $-\infty$ before the softmax, rather than by zeroing weights afterwards. Why does this matter? Think about what softmax does to a row of otherwise-unmasked scores if some scores are simply removed vs. set to $-\infty$.
5. Self-attention in an **encoder** (e.g. BERT) is bidirectional: a token can attend to tokens both before and after it. Why is this appropriate for, e.g., understanding the meaning of a complete sentence, but not appropriate for a model that must generate text left to right?

Exercise 3.2 - The residual stream and the logit lens

Each block does not replace its input - it *adds* its output to a running vector per token, called the **residual stream**:

$$\mathbf{x} \leftarrow \mathbf{x} + \text{Attn}(\text{LN}(\mathbf{x})), \quad \mathbf{x} \leftarrow \mathbf{x} + \text{MLP}(\text{LN}(\mathbf{x})).$$

Because every block only *adds* to the stream, we can peek inside it: multiplying the residual stream at *any* layer by the model's final unembedding matrix (a trick called the **logit lens**) gives the "guess" the model would make if it stopped right there. For the prompt "The electron has a negative...", GPT-2's guess for the next token evolves layer by layer: *pressure* (layer 3) $\rightarrow$ *impact* (layer 7) $\rightarrow$ *value* (layer 9) $\rightarrow$ *spin* (layer 11) $\rightarrow$ *charge* (layer 12, correct, $p = 0.47$).

1. Why does the logit lens trick - reading out an intermediate layer using the *final* unembedding matrix - only make sense because of the residual connection? (Why would it fail if each block fully replaced $\mathbf{x}$ instead of adding to it?)
2. What does the progression *pressure* $\rightarrow$ *impact* $\rightarrow$ *value* $\rightarrow$ *spin* $\rightarrow$ *charge* across layers suggest about *where* in the network the correct prediction is formed - is it already "known" right after the embedding layer, or does it emerge gradually?
3. The MLP sub-layer makes up roughly two-thirds of a block's weights and is thought to be where much of a model's factual "knowledge" lives. Given that the MLP is applied identically and independently to each token position, what kind of "knowledge" can it plausibly store, and what kind of information must instead come from attention?

Exercise 3.3 - Counting parameters and compute

GPT-2 has $d = 768$ (d_{model}), $h = 12$ heads, $d_{\text{ff}} = 3072$, and $N_{\text{layers}} = 12$ blocks.

1. The attention sub-layer has four weight matrices of shape (d, d) (W_Q, W_K, W_V, W_O ; ignore biases). How many parameters is that, in terms of d ?
2. The MLP sub-layer has two weight matrices, of shape (d, d_{ff}) and (d_{ff}, d) (ignore biases). How many parameters is that, in terms of d and d_{ff} ? Using $d_{\text{ff}} = 4d$ (as in GPT-2), express this purely in terms of d .
3. Combine 1. and 2. to get the parameter count for one block.
4. For $N = 12$ blocks, estimate the total number of block parameters. Compare it to GPT-2's reported total of 124M parameters - what is missing from your estimate? Look back at Part I
5. Does the parameter count from 3.-4. depend on the sequence length L ? Does the amount of *computation* (and the memory needed for the attention matrix A , of shape (B, h, L, L)) depend on L ? Explain the difference.
6. (**Optional/advanced**) When generating text token by token, the keys and values of earlier tokens never change once computed, so they can be stored instead of recomputed (a **KV cache**). Why does this mean the cost of generating one *new* token scales with the number of tokens seen so far, rather than with its square?

Exercise 3.4 - From logits to text (Optional/advanced)

The final layer produces logits $\mathbf{z} \in \mathbb{R}^{50257}$ for the next token, turned into probabilities via a softmax with **temperature** T :

$$p_i = \frac{e^{z_i/T}}{\sum_j e^{z_j/T}}.$$

1. What happens to the distribution as $T \rightarrow 0$? What is this decoding strategy called, and why can it lead to repetitive loops (e.g. "the lunch menu today ... is the lunch menu today...")?
2. What happens as T increases well above 1? What is the tradeoff against $T \rightarrow 0$?
3. **top-k** sampling keeps only the k most likely tokens before sampling; **top-p** sampling keeps the smallest set of tokens whose probabilities sum to at least p . Give one advantage top-p has over a fixed top-k in a case where the model is very confident about the next token (probability mass concentrated on 1-2 tokens), compared to a case where it is very unsure (mass spread over hundreds of tokens).

Exercise 3.5 - Matching architectures to tasks (Optional/advanced)

Match each task to the most natural architecture choice: **encoder-only**, **decoder-only**, or **encoder-decoder**. Justify each in one sentence.

task	architecture	justification
Sentiment classification of a movie review		
Autoregressive next-word text generation		
Machine translation (English $\rightarrow$ German)		
Producing a single embedding vector to search a document database		

Short conceptual questions

1. Why is self-attention, on its own, permutation equivariant, and what two mechanisms fix that in a real transformer one for the encoder input, one for the decoder's generation order?
2. Why do queries, keys and values need separate learned projections rather than using the raw input embeddings directly for all three?
3. If the embedding dimension d_{model} is increased while keeping the number of heads h fixed, what happens to d_k per head, and what does that imply for the granularity of what each head can represent?
4. Why does layer normalisation typically operate over the feature dimension d_{model} rather than over the batch dimension, unlike batch normalisation?
5. Explain, without equations, what would go wrong if a decoder used bidirectional (unmasked) self-attention during training for next-token prediction.
6. The MLP sub-layer of a transformer block is applied identically and independently to every token position. What is doing the "mixing" of information *between* token positions in a transformer, if not the MLP?
7. **(Optional/advanced)** Instead of training the embedding matrix E from scratch, one could initialize it with **pretrained** embeddings (e.g. word2vec/GloVe) and either freeze them or fine-tune them further on the downstream task. When would freezing be preferable, and when would fine-tuning be preferable?
8. **(Optional/advanced)** Sampling with temperature T uses $p_i \propto e^{z_i/T}$. Show that this is equivalent to a softmax computed in a different base $a = e^{1/T}$ applied directly to the untouched logits z_i . In these terms, what does a low temperature ($T < 1$) correspond to, and what does a high temperature ($T > 1$) correspond to?