tt-2: A Small Transformer on enwik9

Claude and MJC

2026-11-25, 10:21 UTC

tt-2 is a transformer with 115,008 learned numbers, trained to predict the next byte of enwik9. It is the model that the tiny-transformer track re-expresses as a UM. This paper says what it is, how well it predicts, and which of six behaviors expected of transformers it shows. It is the same model as in the scores paper, which gives its formulas in more detail.

The model in brief

The model reads up to 128 bytes and gives a probability distribution over the next byte at every position. Each position carries a vector $x_t \in \mathbb{R}^{64}$, starting as a column of a learned 64×256 matrix selected by the byte there. Two *layers* follow. A layer adds to x_t, first, the output of four *heads*, and second, $W_2 \max(0, W_1 x_t)$, with 256 rows in W_1 (each row, with its maximum, is a *neuron*). A head computes, at each t, a distribution $a_{t\cdot}$ over positions $0..t$, from dot products of learned linear functions of x_t and x_s, and adds a learned linear function of $\sum_s a_{ts} x_s$. The vector is divided by its root mean square before each of these. The output is the softmax of 64×256 dot products. Layers and heads are numbered from 0: `L1H2` is head 2 of layer 1.

How well it predicts

Bits per byte (bpc), the average of $-\log_2$ of the probability given to the true next byte, over 10^6 bytes of enwik9 that were not used in training (bytes 10^7 to $1.1 \cdot 10^7$), cut into windows of 128 in which nothing is carried over. Training: the first 10^7 bytes, 8 epochs (19,531 steps of 32 windows).

Model	What differs	Numbers	bpc
tt-2	two seeds (random starting values and windows)	115,008	2.2183, 2.2148
tt-1	one layer	65,728	2.4789
tt-2-swiglu	a product of two linear maps of x_t, one passed through a smooth threshold, in place of $\max(0, \cdot)$	117,056	2.1961
tt-2-nonorm	no division by the root mean square	114,688	2.3343

The two tt-2 seeds differ by 0.0035, so the 0.26 lost by one layer and the 0.116 lost without the norm are clear, and the 0.022 gained by swiglu probably is, though swiglu has one seed.

Cost falls with position in the window, since the first bytes have little to go on:

Position	0	1	2–3	4–7	8–15	16–31	32–63	64–127
tt-2, bits	4.06	3.57	2.89	2.39	2.26	2.21	2.16	2.16

It is flat after 32 bytes. The overall 2.218 includes the short-context positions; it would be near 2.16 if each byte were predicted from a full window, an estimate, since that was not run.

Where the heads look

For each head, the average of $a_{t,t-k}$ over held-out windows, by distance k ($k = 0$ is the position itself).

tt-2 head	$k = 0$	1	2	3	sum 0..3	sum 0..15	sum 0..31
L0H0	0.14	0.38	0.24	0.09	0.86	0.96	0.98
L0H1	0.12	0.38	0.20	0.06	0.76	0.92	0.97
L0H2	0.12	0.31	0.25	0.12	0.81	0.94	0.97
L0H3	0.12	0.28	0.29	0.14	0.82	0.96	0.98
L1H0	0.13	0.33	0.33	0.12	0.92	0.99	0.99
L1H1	0.09	0.12	0.15	0.12	0.48	0.83	0.94
L1H2	0.17	0.65	0.15	0.01	0.97	1.00	1.00
L1H3	0.29	0.53	0.13	0.02	0.97	0.99	1.00

tt-2 is a short-range model. Every head puts at least 0.94 of its weight within the last 32 bytes, and all but `L1H1` put at least 0.76 within the last four. The second seed is the same: at least 0.96 within 32, and all but one head at least 0.73 within the last four. This agrees with the cost being flat after 32 bytes.

The six expected behaviors

Behavior	What it would look like	What was measured	Verdict
Previous-byte head	a head in the first layer with most of its weight on the byte before	first layer: at most 0.39 (second seed 0.43, tt-1 0.42). Second layer: 0.65 and 0.53 in tt-2, 0.71 in the second seed	not in the first layer; present in the second
Copying (induction)	a second-layer head that looks at the byte after an earlier match	share of weight on the right place: 0.004, 0.002 in the two seeds; 0.8 in a model trained on copying only	absent
Better with more context	cost late in the window (positions 112–127) lower than early (8–15)	lower by 0.085 bits	present, small
Parking on position 0	much weight on the first byte of the window	at most 0.003, where an even spread gives 0.018	absent
Neurons as key and value	a neuron whose strongest positions are followed by the bytes it raises	for a neuron, *lift*: how many times more often the five bytes it raises most come next at its strongest 1% of positions than overall (1 is no effect). Median 1.07 in layer 0, 2.10 in layer 1 (second seed: 0.86, 1.63); lift at least 2 in 75 and 135 of 256	in layer 1
More features than neurons	a neuron active on several unrelated contexts	the bytes at a neuron's strongest 1% of positions are spread like 4.7 (layer 0) and 7.2 (layer 1) equally common bytes, in the median	consistent, not shown

Two of the six, parking on position 0 and copying, are absent, and one, previous-byte heads, is in the other layer from where it was expected. Lift has no control with shuffled text, and the last row counts bytes only, not contexts; both are weak evidence. The copying result is from the test in the scores paper, where a model trained only on repeated random strings does copy. How much each head matters to bpc on ordinary text was not measured.