tt-2 With a Short Context: tt-2-span16 Is the Model for Step 3

Claude and MJC

2026-11-26

tt-2 is the small transformer of the earlier papers: 115,008 learned numbers, trained on the first 10^7 bytes of enwik9 in windows of 128 bytes, predicting each byte from those before it. Its output at position t can depend on every byte of the window up to t. The plan for step 3 proposed limiting that, so that we can say for certain that nothing more than about 30 bytes back affects a prediction, and named two ways to do it. Both have been trained, in the same way as tt-2 (same bytes, 8 passes, same number of steps, same 4,096 bytes per step):

- **tt-2-ctx32**: windows of 32 bytes in training and in use. A prediction depends on at most 32 bytes: itself and 31 before.
- **tt-2-span16**: windows of 128, but each head may read only its own position and the 15 before it. A head of the second layer reads vectors that the first layer has already filled in from 15 further back, so a prediction depends on at most 31 bytes: itself and 30 before.

Result: tt-2-span16 costs 0.022 **bits per byte (bpc) on its training text against tt-2, and tt-2-ctx32** 0.074**.** tt-2-span16 is the model to rebuild as a UM.

1 The measurement

Each byte of a sample is predicted several times, each time with only the last n bytes visible (n counts the byte at t itself). The sample is 32,512 bytes: the last 127 positions of 256 windows, from training text (the first 10^7 bytes) and from unseen text (the next 10^6); the same bytes for every model and every n. Each entry on its own has a standard error of about 0.02, but the bytes are the same in every row, so a difference between rows is far more precise than that; we have not computed its error, and use the two seeds below as the yardstick.

	training text					unseen text				
Bytes visible, n	8	16	24	32	128	8	16	24	32	128
tt-2	2.118	1.994	1.962	1.946	**1.933**	2.322	2.236	2.207	2.196	**2.186**
tt-2-s2 (second seed)	2.112	1.991	1.960	1.945	1.931	2.312	2.227	2.201	2.190	2.179
tt-2, span 16 imposed	2.118	1.994	1.980	1.978	1.978	2.322	2.236	2.222	2.221	2.221
tt-2-span16	2.104	1.980	1.957	1.955	**1.955**	2.303	2.219	2.202	2.200	2.200
tt-2-ctx32	2.084	2.018	2.008	2.008	–	2.307	2.262	2.252	2.250	–

What to read off.

- *The yardstick.* Two trainings of tt-2 that differ only in their random start (tt-2, tt-2-s2) differ by 0.002 on training text and 0.007 on unseen. A difference much larger than that is the model, not the luck of one training.
- *tt-2-span16* is 0.022 worse than tt-2 on training text and 0.014 on unseen. Forcing the same limit on tt-2 without retraining ("span 16 imposed") costs 0.045 and 0.034: retraining recovers half of it.

- *The limit is exact.* For tt-2-span16 the columns from $n = 32$ on are the same to every digit printed, here and at the eleven other values of n measured between 32 and 128: nothing before the last 31 bytes reaches the output. For tt-2 the same columns keep falling, by 0.012 on training text from 32 to 128.
- *tt-2-ctx32* is the worst from 16 bytes visible on, and 0.062 behind tt-2 cut to its last 32 bytes without any retraining (0.054 on unseen text). It cannot be given more than 32: with 64 visible it costs 6.3 bpc, because it never saw a position past 31.

2 Why tt-2-ctx32 is worse

The table also shows where tt-2-ctx32 is better: at $n = 8$ on training text it beats every other model, by 0.021 (on unseen text it is level with tt-2-span16). Training minimizes the average cost over every position of every window. With windows of 32 a quarter of the bytes it is trained to predict are among the first 8 of their window and so have at most 8 bytes to go on; with windows of 128 it is one in sixteen. tt-2-ctx32 was fitted to a task in which short contexts count four times as much, and it is better at them and worse at long ones. Cost by position in a window, on the unseen text that `tt eval` uses (31,250 windows of 32 for tt-2-ctx32, 7,812 of 128 for the others, so these are other bytes than above):

Position in window (= bytes before it)	0	1	2	4	7	15	31	mean of 0..31
tt-2	4.060	3.565	3.071	2.475	2.339	2.225	2.134	2.385
tt-2-span16	4.048	3.554	3.062	2.447	2.342	2.195	2.138	2.376
tt-2-ctx32	3.990	3.330	2.859	2.385	2.290	2.236	2.220	**2.368**

On its own task, the mean over positions 0 to 31, tt-2-ctx32 is the best of the three. Past position 8 or so it falls behind, and by 31 it is 0.086 behind tt-2. This is our reading of the two tables, not a test of it: it agrees with both, but other causes are not ruled out. tt-2-span16 has no such trade: its windows are 128 long, so only the first 30 positions of each window, as for tt-2, have less than its full reach.

3 What tt-2-span16 gives step 3

- **Every prediction is a function of the last** 31 **bytes, the same function at every position from** 30 **on.** The positions of the window enter only through how far back a byte is (the angle by which a head turns its query and key depends on position, and the score of a pair on their difference only). So an event found at one place in the text means the same thing at any other.
- **Each head is an event space with** 16 **events**, "I read from k bytes back", $k = 0..15$, at every position.
- **Its numbers have the same shapes** as tt-2's: the same 115,008 parameters; only which positions a head may read differs. The explainer and `tt terms` and `tt neuron` work on it unchanged.

Its price, 0.022 bpc on training text, is what the bytes more than 30 back were worth to tt-2. For interpreting the model it buys a window that is fixed and small.

On training text against unseen. Each model costs about 0.25 bpc more on the unseen bytes than on the training bytes (tt-2 0.253, tt-2-s2 0.249, tt-2-span16 0.245, tt-2-ctx32 0.243 with 32 visible). These are different bytes of a different part of enwik9, so the number mixes what each model fitted in particular with how the texts differ, as in the plan; it does not separate the models. The comparisons above come out the same on both texts, except tt-2-ctx32 at $n = 8$.