Why training rests neurons below GELU's dip

Will Sackfield, Open Interpretability · https://open-interpretability.com/research/why-neurons-rest

Abstract

Training moves most of GPT-2's neurons to a median pre-activation h of about -1, as the paper on default-on negative poles found. That is below GELU's minimum at -0.75. That paper also found that the input weights do this. It did not find why. This paper trains small GPT-2s and changes one thing at a time. It finds a small push on each neuron's rest that is steady over many steps, and that the optimizer's step size sets how far the push takes the rest.

Status: Preprint. Work left: #6499 and #6785 (the causes of the push that are still open).

1 Introduction

A transformer's MLP neuron applies an activation function to its pre-activation h, and GPT-2 uses GELU (Hendrycks & Gimpel, 2016). GELU is not monotone. Below 0 its output is negative, falls to a minimum at -0.75 and then rises back toward 0. We call this minimum *GELU's dip*. We call the median of a neuron's h over tokens its *rest*: the level where the neuron sits on a typical token. At random initialisation a neuron rests at about 0. In trained GPT-2 small (Radford et al., 2019), most neurons rest near -1, below the dip (Table 1).

The rest matters for anyone who reads neurons one at a time. We study each sign of a neuron apart, and call one sign a *pole*. An explanation of a pole is *VERIFIED* when it passes every gate of an adversarial review, in which a second model writes probes meant to break it. A neuron that rests below 0 keeps its negative pole active on most tokens. The registry's protocol stops before the review on such a *default-on* pole, because no probe can stay silent on it. Most of GPT-2's neurons rest below GELU's dip, and that resting level is most of what a default-on pole costs. A protocol that ignores the rest must first know where it comes from.

The paper on default-on negative poles found that the input weights set the rest, by turning against the MLP's mean input. It did not find what in training makes them do so. We therefore ask what in training sets the level where a neuron rests. We train small GPT-2s from fresh weights and change one thing at a time: the activation, the width, dropout, weight decay, the optimizer and its rate, and the routes by which the loss reaches the MLP's output.

Our main measure is the *push*: the gradient on a neuron's whole h, with its sign reversed. It gives the direction in which one step of gradient descent moves the rest. The *push ratio* is the push's mean over 100 steps divided by its root mean square, so it measures how steady the push is. Under AdamW (Loshchilov & Hutter, 2019), a parameter moves by about the learning rate times such a ratio at each step. All of the results are on small models, except the figures for GPT-2 small and Pythia-70m, and each number is from one or two seeds.

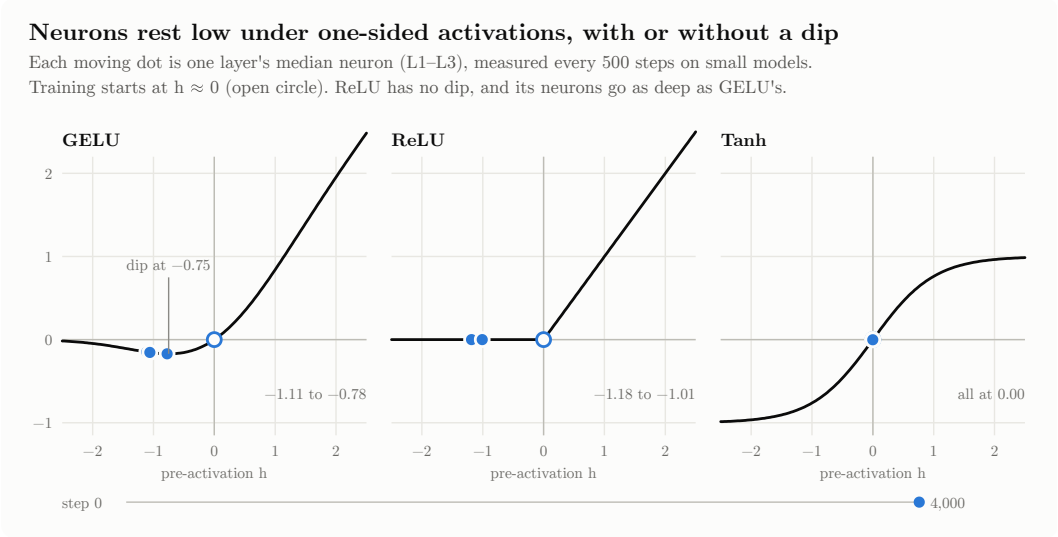

Figure 1. GELU, ReLU and tanh, each with the rest of layers L1 to L3 of a small model marked on its curve. Each moving dot is one layer's median neuron, measured every 500 steps from the start of training near $h = 0$. Under GELU and ReLU the layers rest near -1, and under tanh they stay at 0. ReLU has no dip, and its neurons rest as deep as GELU's.

The paper's contributions are these:

- Controlled training runs of small GPT-2s that show GPT-2's rest, by the same route of weights set against the MLP's mean input. They test the activation, the width, dropout, weight decay, the optimizer and its rate one at a time.
- Evidence that the rest needs a one-sided activation but not GELU's dip. ReLU rests as deep as GELU, tanh does not move, and no dip and no floor gives the order of depth of the activations that we tried.
- A record of the push on each neuron's rest. It points down on only 50%–52% of single steps, but in 85%–100% of 100-step windows for each one-sided activation. Its steadiness, with AdamW's step size, sets the depth.
- A split of the push into a slope term and a spread term. It shows that a neuron's mean output decides which term carries the push, and that the final LayerNorm and dropout carry a pull on the MLP's mean output toward 0.
- Tests on trained GPT-2 small and on Pythia-70m's public checkpoints. They find no steady push at GPT-2's trained weights, and only a weak one in Pythia.

The paper bears on research question 5, *What makes a neuron rest, stay dead or stay on?* (research questions).

2 Related work

Activation functions. GELU (Hendrycks & Gimpel, 2016) and SiLU (Elfwing et al., 2017) both dip below 0, and softplus (Dugas et al., 2001) is a smooth curve with no dip. We compare these with ReLU, leaky ReLU and tanh. The works that proposed these activations measured how well networks learn with them. We ask a different question: where each activation leaves a neuron's pre-activation after training, and why.

Interference and a negative bias. In toy models, Elhage et al. (2022) show that a negative bias can turn small positive interference into negative interference, which is free

under a ReLU in their one-sparse case. A low rest could therefore be a filter against interference. Our step is to apply this idea to the rest of real MLPs. The idea predicts that narrower MLPs, with more interference, rest lower. Our runs do not show that (Table 2). So they count against that version of the idea, and not against the toy result.

Neurons that act through the final LayerNorm. Stolfo et al. (2024) find two kinds of neuron in trained models. Entropy neurons change the final LayerNorm's scale, and token frequency neurons move the output toward or away from the unigram distribution. We find that a constant offset along the MLPs' mean output has both effects (Table 10). In our models the second effect sets the sign of the pull on the mean output.

Dead and dormant units. Dohare et al. (2024) study loss of plasticity in continual learning, and count dead units as one of its correlates. On their Online Permuted MNIST task, "up to 25% of units die after 800 tasks" under backpropagation. Under Adam, the dead units rise early and plateau "at around 60%". A dead ReLU unit is off on almost every input. A neuron that rests below 0 is a weaker form of the same state: its positive pole is mostly off. Both works thus find that the optimizer drives units toward rest. Neither study tests the other's setting, so this is agreement in direction only.

Training runs in public. Pythia publishes the checkpoints of its training runs (Biderman et al., 2023). This lets us read the push while a real model's rest forms, which GPT-2's closed training run does not allow.

3 Method

Terms. For each neuron, h_t is its pre-activation on token t, and the rest is the median of h_t over tokens. A layer's rest is the median over its neurons. The *offset* is $\mathbf{w} \cdot \bar{x}$, with $\mathbf{w}$ the neuron's input weights and $\bar{x}$ the mean of the MLP's input. A neuron's weights are *against the mean input* when its offset is below 0. At each step we recorded $-\partial L \,/\, \partial h$ summed over the batch's tokens, for each neuron:

$$p = -\sum_t \frac{\partial L}{\partial h_t}$$

Here L is the loss and h_t is the neuron's h on token t. This p is the push: the direction in which one gradient step moves the neuron's whole h. It is negative when it moves the rest down. Without centring, it is the gradient on the bias with its sign reversed. We also recorded how far each step moves each neuron's rest, $\bar{x} \cdot \Delta W + \Delta b$, at the batch's mean input $\bar{x}$. Each row of the data sums 100 steps. The push ratio is the push's mean over those 100 steps divided by its RMS, as a median over neurons:

$$\text{push ratio} = \frac{\text{mean}(p)}{\text{RMS}(p)}$$

AdamW moves a parameter by about the learning rate times this kind of ratio.

Setup. The script is `scripts/train_resting.py`. Its output is in `results.jsonl`. `scripts/paper_figures.py` draws the figures from that output, with no model.

- **Models.** The architecture of GPT-2 (Radford et al., 2019) with 4 layers, width 256 and 8 heads, trained from fresh weights for 4,000 steps. Each step is 32 windows of 128 tokens, with AdamW (Loshchilov & Hutter, 2019) at a learning rate of 3e-4 and gradient clipping at 1.0.
- **Data.** Training uses the first 10,000 documents of the Pile (Gao et al., 2020) without the documents of the `corpus.jsonl` sample, which leaves 9,482 of 10,000 documents.

Evaluation uses 128 test windows of the same split as What default-on negative poles do as a population.

- **Activation.** The baseline uses GELU (Hendrycks & Gimpel, 2016). Other conditions use ReLU, tanh, softplus (Dugas et al., 2001), leaky ReLU and SiLU (Elfwing et al., 2017). Softplus is $\log(1 + e^h)$. Leaky ReLU is h above 0 and $0.1h$ below 0, or $0.01h$ for the condition `leaky_relu_0.01`. SiLU is $h \cdot \sigma(h)$, with σ the logistic sigmoid.
- **Optimizer.** Other conditions use AdamW at 1e-4 or 1e-3, or SGD with momentum 0.9 at 0.1. We chose 0.1 from 0.01, 0.03, 0.1 and 0.3 over 500 steps. 0.1 and 0.3 reached the same loss, and 0.3 was less steady at step 250. Each of these conditions ran once, on seed 0.
- **Measures.** The median over neurons of each neuron's median h, and the median offset ($\mathbf{w} \cdot \bar{x}$). Then the median bias, the skew, and the share of neurons with weights against $\bar{x}$.
- **Push.** With `--push-every 100`, each run also records the gradient on each neuron's rest at every step, and how far the step moves the rest. It writes a summary of each 100 steps as a `push` row. Recording does not change training: a 500-step GELU run with it matched the earlier run's loss and rests exactly.
- **Centred conditions.** h is centred per neuron over each batch, then moved to a rest. That rest is fixed for `rest_0` and `rest_-1`, and is a parameter for each neuron for `rest_learned` and `rest_learned_fast`. Centring removes the weights' effect on the rest, so we report the rest as applied, not the raw h.
- **Held rests.** Conditions such as `relu_rest_0` and `softplus_rest_-2` centre h in the same way and hold it at a fixed rest, with another activation. The gradient on that rest is then the push at a rest chosen in advance (Tables 7 and 11).
- **Norm conditions.** `relu_norms_centred` centres the input of each block's LayerNorms over the batch's tokens, and `relu_all_norms_centred` also the final LayerNorm's, so that no LayerNorm sees a constant offset. `relu_final_norm_centred_dropout_0` centres only the final LayerNorm's input. A `_dropout_0` suffix, and the `dropout_0` condition for GELU, set every dropout to 0.
- **GPT-2 push.** `scripts/pretrained_push.py gpt2` records the same terms for trained GPT-2 small, over 50 batches of the corpus sample's windows, with no optimizer step. It records with dropout off (`eval`) and with GPT-2's dropout of 0.1 (`train`).
- **Output conditions.** `gelu_output_centred` and `silu_output_centred` centre each neuron's output a over the batch, so the output's gradient sums to 0 over tokens. `relu_output_shift` and `gelu_output_shift` add a constant to the output that moves its mean at init to the other activation's.
- **Pythia's setup.** Each `pythia_` condition adds one part of Pythia's training to the no-dropout model. The parts are its warm-up, its AdamW settings, parallel attention, its learning rate and a larger batch. `pythia_all` adds all of them (Table 20).
- **Offsets.** `scripts/ln_offset.py` adds an offset to the input of a trained model's LayerNorm. It measures the loss's slope by autograd (Table 10). The small models are saved with `--save-every`.
- **Terms of the push.** The recorder also splits the push into a slope term and a spread term, and keeps the push ratio of the weights along $\bar{x}$. It keeps the cosine of the MLP's mean output with the gradient on its output bias. It also keeps two correlations: the mean output against the output's push over neurons, and the output against its gradient over tokens. `--push-part` labels the push rows of a rerun so that they do not mix with the first run's rows. `--spread-bins` also splits the spread term by bins of h, with the edges in `PUSH_BINS`.

Figures. Figure 1, Figure 3 and Figure 8 move. In Figures 1 and 3, each dot follows one layer's median neuron along the activation's curve, every 500 steps. Figure 8 draws the rest over 12,000 steps as training goes on.

4 What sets the rest, and do the activations differ?

The first question is whether small models rest as GPT-2 does, and by the same route, so that they are a fair place to test causes. On GPT-2 small itself, the rest is an offset, not a skew (Table 1). On every layer but L1, the mean of h is close to its median, and skew is small. L0's skew is the largest, at 0.62. L1 is the exception, with a skew of -0.33 and a shallow rest of -0.24. In every layer, 89% to 100% of neurons have weights against the mean input, against about half at random initialisation.

Layer	**Median** h	**Offset from the mean input**	**Bias**	**Skew**	**Against mean input**	**Random init: median** h**, against**
L0	-1.16	-1.01	-0.07	0.62	93%	0.00, 50%
L1	-0.24	-0.33	-0.05	-0.33	89%	0.01, 49%
L2	-1.15	-1.04	-0.09	0.08	95%	0.00, 50%
L3	-1.25	-1.13	-0.09	0.10	99%	0.01, 49%
L4	-0.99	-0.92	-0.08	0.04	99%	0.01, 49%
L5	-1.11	-1.02	-0.07	0.10	99%	-0.01, 51%
L6	-1.05	-0.95	-0.08	0.11	99%	-0.01, 51%
L7	-1.01	-0.91	-0.08	0.11	99%	0.00, 50%
L8	-1.03	-0.93	-0.08	0.12	99%	0.00, 50%
L9	-1.04	-0.95	-0.08	0.13	100%	-0.01, 51%
L10	-0.99	-0.90	-0.08	0.15	99%	0.00, 49%
L11	-0.86	-0.78	-0.07	0.15	97%	0.00, 50%

Table 1. GPT-2 small, by layer: the median h, the median offset from the mean input, the median bias, the skew, and the share of neurons with weights against the mean input. The last column gives the median h and that share for the same architecture at random initialisation.

The small models show GPT-2's pattern (Table 2). Their rest comes from weights set against the MLP's mean input, and not from the bias or from a skewed tail. So a small model is a fair place to test causes, with the limits that *Limitations* gives. The rest does not need GELU's dip. ReLU has no dip, and its neurons rest at the same depth (Figure 1). Tanh's neurons do not move at all. So the rest goes with a one-sided activation, and "below the dip" may be a side effect.

Condition	Loss, seed 0	Loss, seed 1	L0	L1	L2	L3	Against mean input, L1–L3
Baseline: GELU, 4× width, dropout 0.1, no weight decay	5.509	5.509	−0.34	−1.11	−1.06	−0.78	100%
−GELU(−h): a check, since this is the baseline with signs reversed	5.501		0.33	1.08	0.99	0.78	0%
ReLU	5.486	5.484	−0.40	−1.13	−1.18	−1.01	100%
Tanh	5.686		0.00	0.00	0.00	0.00	48%–50%
1× width	5.546		−0.29	−1.11	−1.05	−0.76	100%
16× width	5.504		−0.37	−1.17	−1.13	−0.73	100%
No dropout	5.452		−0.16	−0.97	−1.00	−0.76	100%
Dropout 0.3	5.675		−0.57	−0.84	−0.98	−0.74	100%
Weight decay 0.1	5.523		−0.28	−0.98	−0.94	−0.67	100%

Table 2. What changes the rest, at step 4,000 of each condition. Columns L0–L3 are the median h. A blank seed 1 entry means that the condition ran once.

Width, dropout and weight decay do not set the rest (Figure 2). It forms with no dropout and no weight decay. A 16× range of width changes L1–L3 by at most 0.08. The interference explanation predicts that narrower MLPs rest lower, and we do not see that. The appendix *Reading the changes to the baseline* gives the detail of each condition.

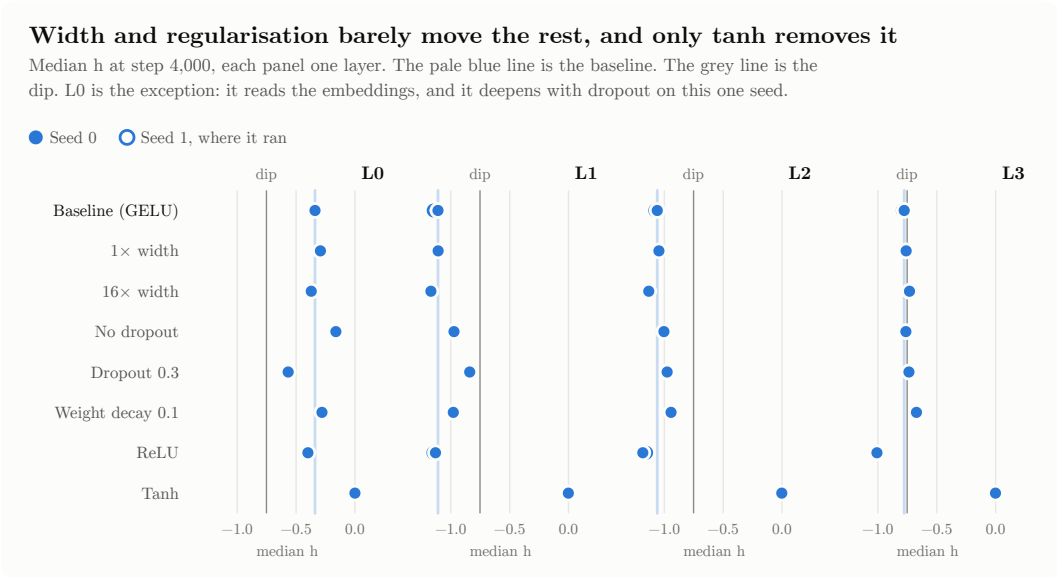

Figure 2. The median h at step 4,000 under each change to the baseline, one panel for each of layers L0 to L3. Width, dropout, weight decay and ReLU stay close to the baseline at L1 to L3, and only tanh removes the rest. L0 is the exception, and deepens with dropout on one seed.

No dip and no floor sets the depth (Table 3 and Figure 3). Every one-sided activation that we tried rests below 0, with its weights against the mean input. But softplus rests at −4.68, leaky ReLU at −0.69 and SiLU at −0.49 at L1. SiLU's dip is deeper than GELU's, and its neurons rest shallower than GELU's. So no single property of these shapes that we checked gives their order (appendix *Reading the other activations*).

Activation	Loss	L0	L1	L2	L3	SD of h, L1–L3	Against mean input, L1–L3
GELU (baseline)	5.509	−0.34	−1.11	−1.06	−0.78	0.78–0.95	100%
ReLU	5.486	−0.40	−1.13	−1.18	−1.01	0.62–0.74	100%
Softplus	5.826	−3.83	−4.68	−4.60	−3.52	1.55–2.17	99%–100%
Leaky ReLU, slope 0.1	5.518	−0.41	−0.69	−0.79	−0.68	0.55–0.68	100%
Leaky ReLU, slope 0.01	5.496	−0.40	−1.11	−1.20	−1.06	0.62–0.74	100%
SiLU	5.588	−0.30	−0.49	−0.42	−0.24	0.46–0.71	98%–99%
Tanh	5.686	0.00	0.00	0.00	0.00	0.15–0.19	48%–50%

Table 3. More activations, at step 4,000 of each condition, with AdamW at the base rate. Columns L0–L3 are the median h.

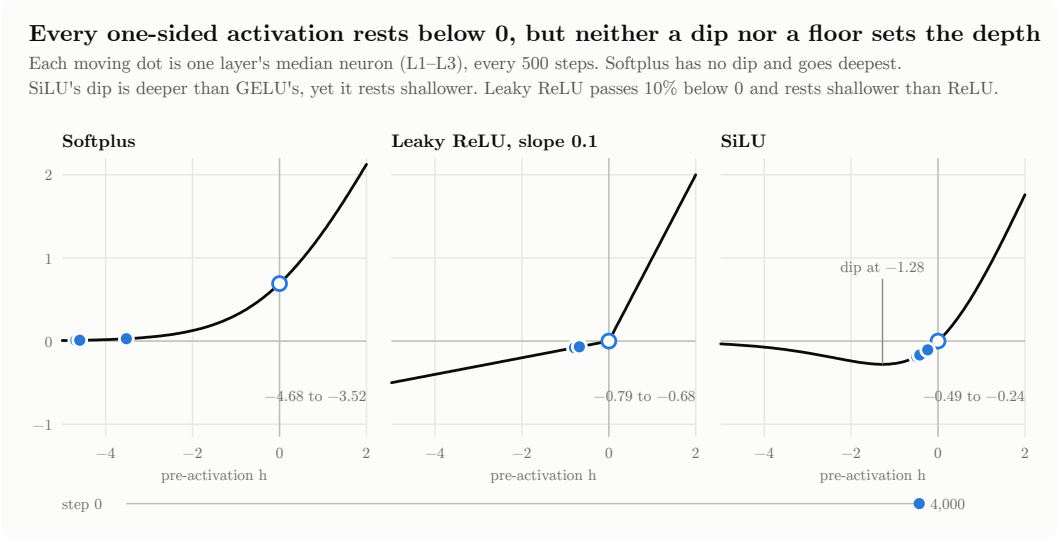

Figure 3. Softplus, leaky ReLU and SiLU, each with the rest of layers L1 to L3 marked on its curve, every 500 steps. Softplus has no dip and goes deepest. SiLU's dip is deeper than GELU's, yet it rests shallower. Leaky ReLU passes 10% below 0 and rests shallower than ReLU.

5 How do the optimizer and the learning rate set the depth?

The second question is whether the depth is a target of the loss or a product of how the optimizer moves. The optimizer's step size sets the depth. With AdamW, L1's rest at step 4,000 is −0.37, −1.11 and −3.18 at learning rates of 1e-4, 3e-4 and 1e-3 (Table 4). That is close to proportional to the rate. The two larger rates end at the same loss, so the loss does not need a particular depth (Figure 4). Under SGD the rest forms, but it is shallower than AdamW's at the same loss.

Activation	Optimizer	Loss	L0	L1	L2	L3	L1 at a loss of 5.9
GELU	AdamW 1e-4	5.877	−0.14	−0.37	−0.30	−0.24	−0.37
GELU	AdamW 3e-4	5.509	−0.34	−1.11	−1.06	−0.78	−0.68
GELU	AdamW 1e-3	5.505	−1.71	−3.18	−3.07	−2.47	−2.55
GELU	SGD 0.1	5.804	−0.62	−0.32	−0.25	−0.23	−0.28
ReLU	AdamW 1e-4	5.863	−0.17	−0.40	−0.43	−0.38	−0.39
ReLU	AdamW 3e-4	5.486	−0.40	−1.13	−1.18	−1.01	−0.87
ReLU	AdamW 1e-3	5.494	−2.74	−4.04	−4.43	−3.33	−2.95
ReLU	SGD 0.1	5.797	−0.68	−0.59	−0.59	−0.60	−0.55

Table 4. The optimizer and the learning rate, at step 4,000 of each condition, seed 0. The AdamW 3e-4 rows are the baseline and ReLU from Table 2. Columns L0–L3 are the median h. The last column interpolates linearly between the two evaluations around a loss of 5.9.

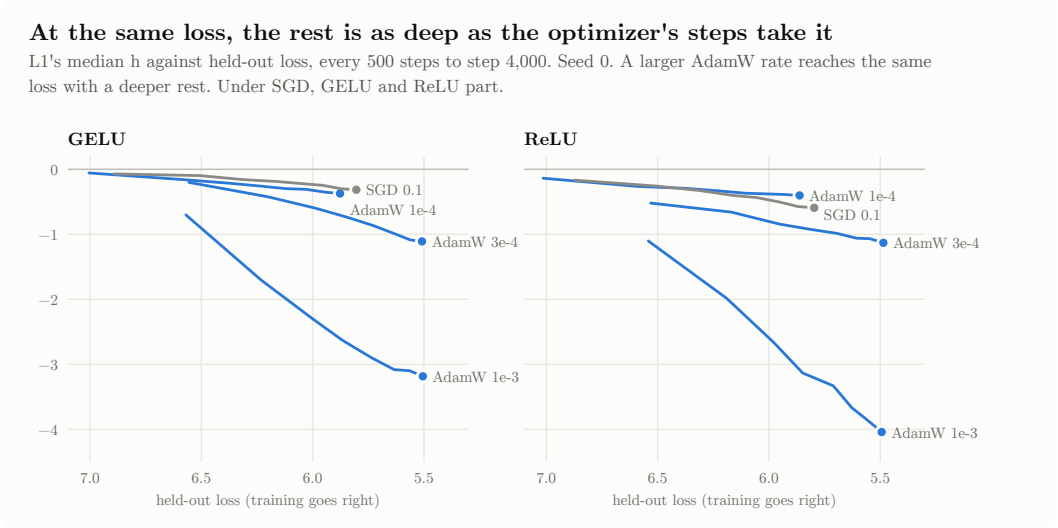

Figure 4. L1's median h against held-out loss, every 500 steps to step 4,000, on seed 0, for GELU and ReLU. A larger AdamW rate reaches the same loss with a deeper rest. Under SGD, GELU and ReLU part.

ReLU and GELU agree only under AdamW. They rest at the same depth at AdamW's two lower rates. Under SGD, ReLU rests nearly twice as deep as GELU (−0.59 against −0.32 at L1). At 1e-3, ReLU is deeper (−4.04 against −3.18). The first version of this paper used only Adam at one rate, and warned that GELU and ReLU might agree because of it. These runs show that this is so in part.

The push accounts for this pattern (Table 5). Its ratio does not depend on the rate: for ReLU it is −0.018 to −0.020 at all three AdamW rates, and for GELU −0.014 to −0.019. So AdamW takes a step in proportion to its rate on a push of about the same steadiness, which is a fixed distance per step. SGD's step instead follows the push's raw size. ReLU's raw push under SGD is 1.7× GELU's, and ReLU rests 2.3× as deep as GELU under SGD. The appendix *Reading the optimizer runs* gives the detail and seed 1.

Activation	Optimizer	Rest, L1–L3	Push ratio, steps 1–4,000	Median raw push
GELU	AdamW 1e-4	−0.29	−0.016	-5.2×10^{-6}
GELU	AdamW 3e-4	−0.98	−0.019	-3.4×10^{-6}
GELU	AdamW 1e-3	−2.95	−0.014	-1.1×10^{-6}
GELU	SGD 0.1	−0.26	−0.015	-3.0×10^{-6}
ReLU	AdamW 1e-4	−0.40	−0.018	-5.8×10^{-6}
ReLU	AdamW 3e-4	−1.10	−0.020	-4.1×10^{-6}
ReLU	AdamW 1e-3	−4.32	−0.018	-9.3×10^{-7}
ReLU	SGD 0.1	−0.59	−0.021	-5.0×10^{-6}

Table 5. The push on the rest under each optimizer and rate, from the runs that recorded it. The rows are seed 1, except AdamW 3e-4, which is the seed-0 reruns of Table 6. Rest, L1–L3 is the mean of the three layers' median h at step 4,000.

6 What pushes the rest down, and where does the push come from?

The third question is what in each step of training moves the rest. A small, steady push sets the depth (Table 6 and Figure 5). The gradient on each neuron's rest points down on only 50%–52% of single steps. But its mean over 100 steps points down in 85%–100% of windows for each one-sided activation. Its size, as mean over RMS, gives the order of depth. Softplus is first. ReLU, GELU and leaky ReLU at slope 0.01 are next, then leaky ReLU at slope 0.1, SiLU and tanh. The push moves the rest through the weights, which carry 97%–99% of each step's movement.

Activation	Seed	Loss	L1	Rest, L1–L3	Push ratio, steps 1–1,000	Push ratio, steps 1–4,000	Windows pushed down	Single steps pushed down
Softplus	0	5.826	−4.69	−4.27	−0.080	−0.043	100%	51.7%
Softplus	1	5.797	−4.96	−4.41	−0.082	−0.044	100%	51.6%
ReLU	0	5.491	−1.11	−1.10	−0.036	−0.020	100%	50.0%
Leaky ReLU, slope 0.01	0	5.496	−1.11	−1.12	−0.036	−0.020	100%	50.1%
Leaky ReLU, slope 0.01	1	5.470	−1.16	−1.12	−0.036	−0.020	100%	50.1%
GELU	0	5.509	−1.11	−0.98	−0.022	−0.019	97%	50.4%
Leaky ReLU, slope 0.1	0	5.557	−0.70	−0.70	−0.026	−0.015	93%	50.0%
Leaky ReLU, slope 0.1	1	5.517	−0.70	−0.74	−0.027	−0.017	100%	50.1%
SiLU	0	5.588	−0.49	−0.38	+0.002	−0.010	85%	50.2%
SiLU	1	5.588	−0.55	−0.40	+0.001	−0.010	87%	50.3%
Tanh	0	5.686	0.00	0.00	+0.001	+0.001	42%	50.0%

Table 6. The push on the rest, at step 4,000 of each run, with AdamW at the base rate. The push columns are means over L1–L3. Rest, L1–L3 is the mean of the three layers' median h.

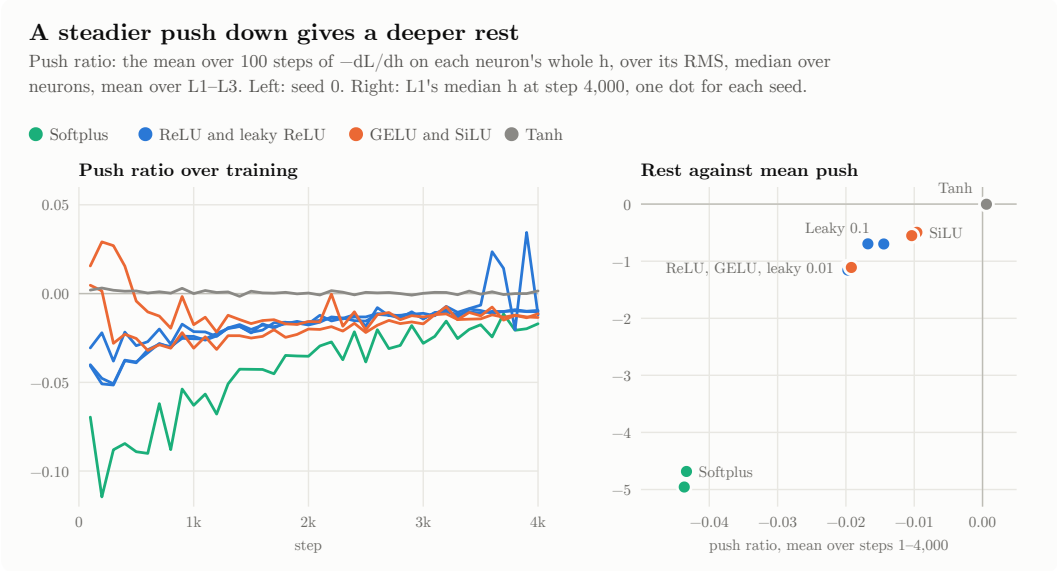

Figure 5. Left: the push ratio over 4,000 steps for each activation, as a median over neurons and a mean over L1–L3, on seed 0. Right: L1's median h at step 4,000 against the mean push ratio, one dot for each seed. A steadier push down gives a deeper rest.

Table 6 is a correlation along each run, because each push is measured where that run's neurons rest. Runs that hold every activation at the same rest separate the activation's effect from the depth's (Table 7 and Figure 6). At an equal rest, the activation sets only a coarse order. Held at a rest of 0, softplus, ReLU and leaky ReLU at slope 0.1 get the same push, to 0.001, on two seeds. GELU's push is a little weaker, SiLU's is weaker again, and tanh gets none. So the depth that a neuron reaches changes its push, and that change sets most of the order (appendix *Reading the push*).

Activation	**Push ratio held at 0, seed 0**	**Seed 1**	**Free push ratio**	**Free rest, L1–L3**	**Loss held at 0, seed 0**
Softplus	−0.069	−0.070	−0.043	−4.27	6.019
ReLU	−0.070	−0.070	−0.020	−1.10	5.659
Leaky ReLU, slope 0.1	−0.070	−0.070	−0.015	−0.70	5.674
GELU	−0.064	−0.065	−0.019	−0.98	5.645
SiLU	−0.041	−0.043	−0.010	−0.38	5.686
Tanh	0.000	+0.002	+0.001	0.00	5.757

Table 7. The push at an equal rest. h is centred per neuron over each batch and held at 0. Push ratios are means over steps 1–4,000 and over L1–L3. The free columns are from Table 6.

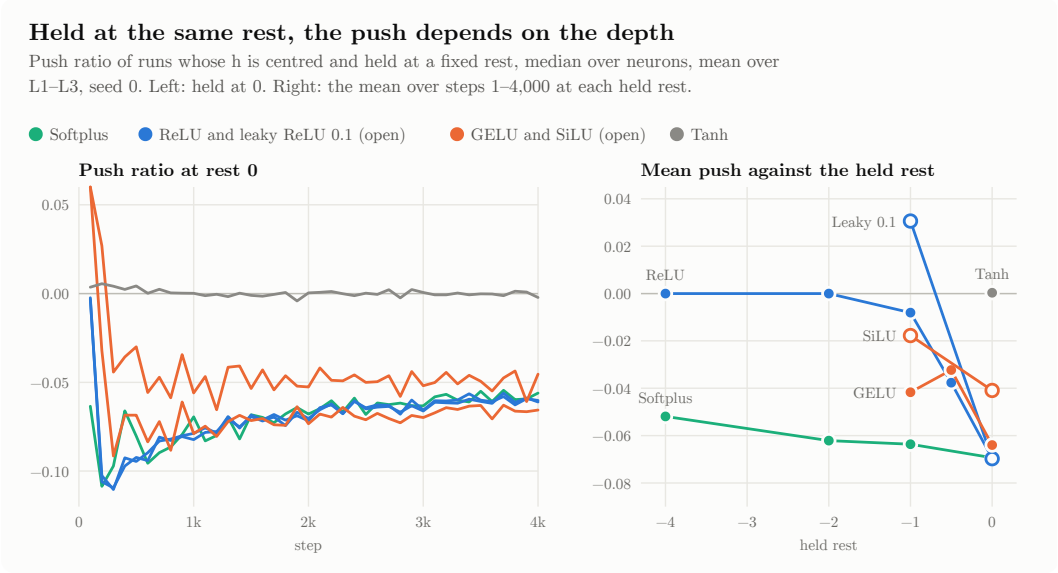

Figure 6. Left: the push ratio over 4,000 steps with every activation held at a rest of 0, on seed 0. Right: the mean push ratio over steps 1–4,000 at each held rest. Held at the same rest, the push depends on the depth.

To find where the push comes from, we split it. On each token t, the push on h is the output's push, $g_t = -\,\partial L \,/\, \partial a_t$, times the slope $a'(h_t)$. So the push over the batch is the sum of two terms:

$$p = \sum_t g_t\, a'(h_t) = \Big(\sum_t g_t\Big)\,\overline{a'} + \sum_t g_t\,\Big(a'(h_t) - \overline{a'}\Big)$$

Here $\overline{a'}$ is the mean slope over tokens. The *slope term* is the output's push summed over tokens, times the mean slope over tokens. The *spread term* is the remainder. It comes from tokens whose slope differs from the mean.

The push has two terms, and each one-sided activation has at least one that points down (Table 8). For ReLU and softplus, the output's push points down all through training, and the slope term carries the push. For GELU and SiLU, the output's push points up, and the spread term carries the push down. For tanh, both terms are 0.

Activation	Output's push, steps 1–200	Slope term	Spread term	Output's push, steps 1,001–2,000	Slope term	Spread term
ReLU	−12.3	−5.9	+1.6	−30.5	−3.4	−0.7
Softplus	−3.1	−1.4	+0.1	−7.3	−1.1	+0.3
GELU	+8.9	+1.6	−1.5	+16.3	+2.6	−6.9
SiLU	+11.4	+3.9	−1.1	+8.2	+2.9	−4.5
ReLU, shifted down	+7.4	−1.1	−1.6	+15.3	+1.5	−6.0
GELU, shifted up	+0.2	−2.6	−0.1	+2.4	−0.5	−3.6
Tanh	+0.6	+0.5	0.0	0.0	0.0	0.0

Table 8. The terms of the push, from runs that repeat seven activations for 2,000 steps. Units are 10^{-6}, as medians over neurons and means over L1–L3 and over the steps. Medians of the terms do not add up to the median of the push.

A neuron's mean output decides which term carries the push. ReLU and softplus have large mean outputs at init, and GELU and SiLU small ones. Adding a constant to the output leaves the activation's shape the same. ReLU shifted to GELU's mean output then gets GELU's pattern: its output's push points up, and it rests about a third shallower. GELU shifted to ReLU's mean output loses its up phase and rests about twice as deep. One cause is a weak pull of the loss on the MLP's mean output toward 0, which holds in every window of every run.

The spread term comes from tokens with a low slope, not from the steepest tokens (Table 16, in the appendices). For GELU and SiLU, the output's push points up mostly on tokens below the rest (Table 23), also when no norm and no dropout can see a constant. Their slope is below the mean, so they make the spread term point down. In GELU's dip their slope is negative, so they push h down directly. The steepest tokens, from 0.75 to 1.5, make the spread term point up. Two seeds agree to 0.13 (10^{-6}) in each bin. The appendix *The terms of the push* gives the detail.

7 How do the final LayerNorm and depth shape the push?

The fourth question is how the loss reaches the MLP's mean output, and how the push changes as the rest deepens. A constant offset in the residual stream can reach the loss in two ways. It adds to each token's input to the LayerNorms that read the stream. Dropout on the MLP's output also turns it into noise on each token. The norm conditions close the LayerNorm routes, and the `_dropout_0` conditions close the dropout route, each for 2,000 steps on ReLU (Table 9).

Run	Routes open	Cosine, seed 0	Cosine, seed 1	Windows above 0, seed 0
ReLU	block norms, final norm, dropout	+0.072	+0.072	20 of 20
ReLU, no dropout	block norms, final norm	+0.043	+0.044	20 of 20
Block norms centred	final norm, dropout	+0.051	+0.052	20 of 20
Block norms centred, no dropout	final norm	+0.046	+0.048	20 of 20
All norms centred	dropout	+0.059	+0.060	20 of 20
Final norm centred, no dropout	block norms	−0.017	−0.019	1 of 20
All norms centred, no dropout	none	−0.001	0.000	6 of 20
GELU	block norms, final norm, dropout	+0.063	+0.066	20 of 20
GELU, no dropout	block norms, final norm	+0.049	+0.053	20 of 20

Table 9. The routes of the pull. The cosine is between the MLP's mean output and the gradient on its output bias, as a mean over L1–L3 and over the 20 windows. A positive cosine means that gradient descent shrinks the mean output.

The pull on the mean output goes through the final LayerNorm and dropout. With dropout at 0, the pull stays, but it is weaker: ReLU's cosine falls from +0.072 to +0.043, on two seeds. When the final LayerNorm cannot see a constant and nothing drops out, the pull stops. The block LayerNorms alone do not make it. With nothing to pull it back, the MLPs' mean output then grows until a shared constant fills the input of the later blocks' LayerNorms, and L2 and L3 nearly shut (appendix *Why L2 and L3 shut*). Dropout's own

pull goes through the block LayerNorms and the final LayerNorm alike (appendix *Dropout's routes*). With both routes closed, the push on ReLU's rest keeps its pace, so the pull is not needed for the rest. The appendix *The pull on the mean output* gives the detail.

The final LayerNorm's pull comes from the unigram prior. One reading was that a constant takes part of each token's norm. Then the LayerNorm divides by a larger norm, shrinks the parts that differ between tokens, and flattens the logits. `scripts/ln_offset.py` tests this on trained models. It adds an offset c along a direction to the input x of a LayerNorm, in three ways:

- **Full** is what the model computes: $\ln(x + c)$.
- **Shrink** keeps only the larger divisor. It normalises x by the standard deviation of $x + c$.
- **Shift** keeps only the moved input. It normalises $x + c$ by the standard deviation of x.

To first order, the full effect is the sum of the other two. So autograd gives the loss's slope as c grows from 0, for each way. The unit of c is the mean norm of $x - \bar{x}$, the part of x that differs between tokens. The models are the base GELU on seeds 0 and 1, and ReLU on seed 0, saved every 1,000 steps. Retrained, GELU on seed 0 matches the earlier run exactly, and the others are within 0.003 in loss. Each slope is over the 128 test windows, in eval mode, so dropout is off.

Model, step 4,000	Layer	Full	Shrink	Shift
GELU, seed 0	L1	+0.053	−0.057	+0.110
GELU, seed 0	L2	+0.058	−0.035	+0.093
GELU, seed 1	L1	+0.093	−0.052	+0.145
ReLU, seed 0	L1	+0.234	−0.053	+0.287
ReLU, seed 0	L2	+0.074	−0.035	+0.109

Table 10. Slopes of the loss per unit of offset at the final LayerNorm, along the layer's MLP mean output.

An offset along the MLPs' mean output costs loss through the final LayerNorm only by moving its input (Table 10). That move adds a fixed logit vector that follows token frequency ($r = +0.75$ to $+0.91$). The larger divisor flattens the logits and lowers the loss. The later blocks' LayerNorms use the same offset, and it lowers their loss. So they do not make the pull. The two ways match two kinds of neuron that Stolfo et al. (2024) found in trained models. Entropy neurons change the final LayerNorm's scale, which is the shrink. Token frequency neurons move the output toward or away from the unigram distribution, which is the shift. Here a constant in the MLPs' output does both, and the shift sets the sign. The appendix *What the final LayerNorm does with a constant* gives the detail.

The push survives at depth only if the slope does (Table 11). Held at -1, ReLU keeps a ninth of its push at 0, and at -2 none. Softplus keeps three quarters of its push even at -4, because every token still has a slope. So softplus keeps sinking long after ReLU slows, and rests about twice as deep as its mean push predicts. Leaky ReLU at slope 0.1 is pushed back up at -1, where its mean output turns negative. So it settles between 0 and -1.

Held rest	ReLU	Softplus	GELU	Leaky ReLU, slope 0.1	SiLU
0	−0.070	−0.069	−0.064	−0.070	−0.041
−0.5	−0.038		−0.032		
−1	−0.008	−0.064	−0.042	+0.031	−0.018
−2	0	−0.062			
−4	0	−0.052			

Table 11. Push against depth: the push ratio of runs held at rests below 0, on seed 0. Push ratios are means over steps 1–4,000 and over L1–L3. The rows at 0 are the equal-rest runs of Table 7. Seed 1 gives the same push ratios to 0.003.

8 Does the model need the rest?

The fifth question is whether the rest helps the model, or whether training only drifts there. Forcing the rest to 0 costs 0.136 of loss on seed 0 and 0.155 on seed 1. But centring is part of that change, so we added controls (Table 12 and Figure 7).

Condition	Loss, seed 0	Loss, seed 1	Rest set by	L0	L1	L2	L3	Against mean input, L1–L3
Baseline	5.509	5.509	the weights	−0.34	−1.11	−1.06	−0.78	100%
Centred, learned rest at 30× the rate	5.574	5.581	the learned value	−0.44	−0.92	−0.85	−0.71	47%–58%
Centred, learned rest at the base rate	5.635		the learned value	−0.05	−0.09	−0.08	−0.08	29%–52%
Centred, rest 0	5.645	5.664	forced	0.00	0.00	0.00	0.00	26%–49%
Centred, rest −1	5.657	5.664	forced	−1.00	−1.00	−1.00	−1.00	38%–56%

Table 12. The centred controls, at step 4,000. Columns L0–L3 are the rest.

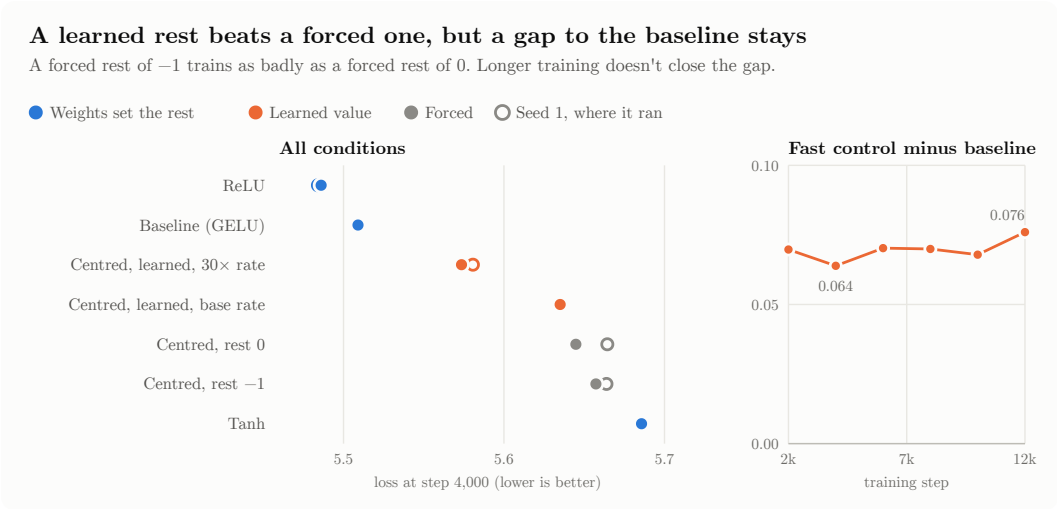

Figure 7. Left: the loss at step 4,000 for each condition. ReLU and GELU are lowest, the learned rest at 30 times the rate is next, and the forced rests and tanh are worst. Right: the fast control's gap to the baseline over steps. A forced rest of −1 trains as badly as a forced rest of 0, and longer training does not close the gap.

Training pushes the rest down by whichever route it has. Centring can stop the weights from setting the rest. Then a learned value for each neuron goes down at nearly the same pace, with the same order of layers. So the weights' turn against the mean input is one route to the rest, not its cause.

The rest seems to help, but this is not proven. Letting the learned value reach that depth lowers the loss by 0.061 on seed 0, with seeds differing by about 0.02. A centred model still does not match the baseline, and a forced rest of −1 trains as badly as a forced rest of 0.

Step 4,000 is early, because the rest is still moving. So seed 0 of the baseline and of the fast control ran to step 12,000, and ReLU ran to step 12,000 later (Table 13). The value −1 is not a target. Over 12,000 steps the rest keeps deepening, to −1.46 at L1 with the weights and −1.32 with the learned value (Figure 8 and Figure 9). ReLU keeps pace, at −1.49. Its growth slows, but it passes GPT-2's −1.04. The closeness to −1 at step 4,000 is partly the moment that we measured. The appendix *Reading the runs on the rest's value* gives the detail.

Step	Baseline loss	Fast control loss	Gap	Baseline rest: L0	L1	L2	L3	Fast control rest: L0	L1	L2	L3
2,000	5.841	5.910	0.070	−0.21	−0.74	−0.68	−0.50	−0.30	−0.60	−0.53	−0.41
4,000	5.510	5.574	0.064	−0.34	−1.10	−1.06	−0.78	−0.44	−0.92	−0.85	−0.71
6,000	5.348	5.418	0.070	−0.42	−1.28	−1.24	−0.95	−0.51	−1.08	−1.02	−0.86
8,000	5.241	5.311	0.070	−0.47	−1.35	−1.37	−1.04	−0.55	−1.19	−1.12	−0.96
10,000	5.163	5.231	0.068	−0.50	−1.43	−1.44	−1.13	−0.58	−1.26	−1.20	−1.02
12,000	5.137	5.213	0.076	−0.54	−1.46	−1.49	−1.15	−0.60	−1.32	−1.23	−1.07

Table 13. Longer training, seed 0: the loss and the rest of the baseline and of the fast control, every 2,000 steps to step 12,000.

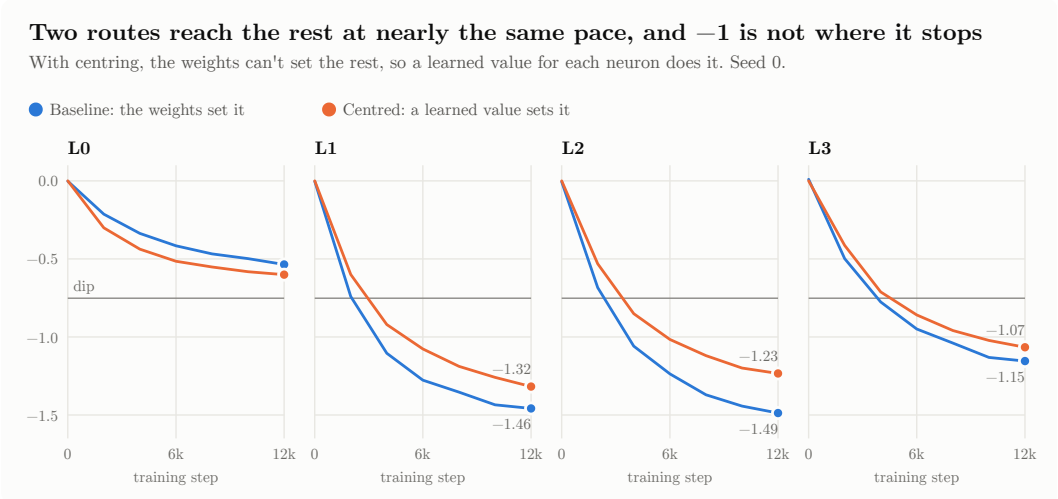

Figure 8. The rest over 12,000 steps on layers L0 to L3, for the baseline, where the weights set the rest, and for the fast control, where a learned value sets it. The lines draw themselves as training goes on. The two routes reach the rest at nearly the same pace, and −1 is not where the rest stops.

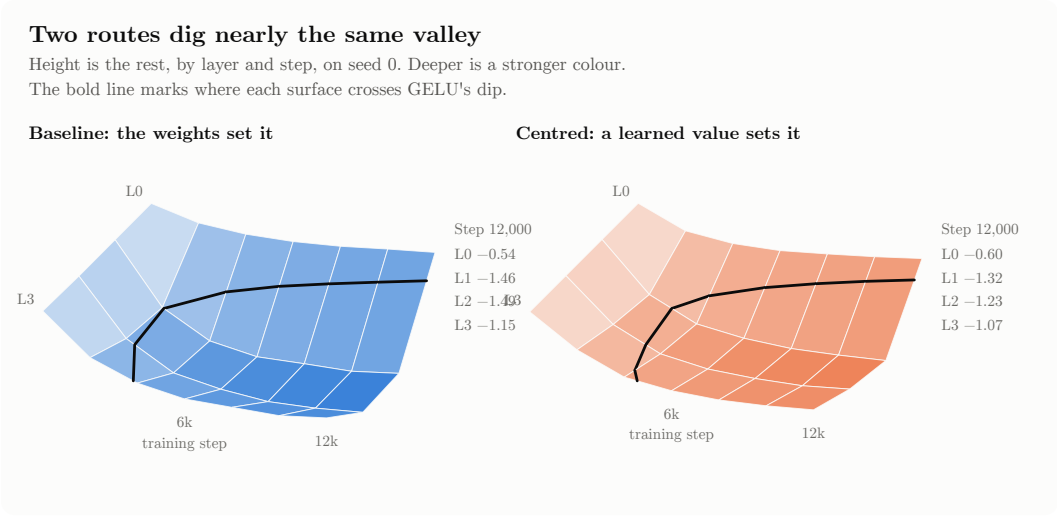

Figure 9. The rest as a surface over layer and step, on seed 0, for the baseline and the fast control. Height is the rest, and a deeper rest has a stronger colour. A bold line marks where each surface crosses GELU's dip. Both fall steeply on L1 and L2, and less on L0 and L3.

9 Is the push there in GPT-2 small and Pythia-70m?

The last question is whether trained models show the push that the small models show. At its trained weights, GPT-2 small has no steady push and no pull (Table 14). The slope term points up and the spread term down, as for the small GELU models, but they cancel. Half of its neurons are pushed down, and the cosine of the mean output with its gradient is negative in all 12 layers.

Measure, over the 12 layers	No dropout	Dropout 0.1
Push ratio, median over neurons	−0.041 to +0.043	−0.116 to +0.234
Share of neurons pushed down	48%–52%	39%–58%
Slope term above 0	12 layers	8 layers
Spread term below 0	11 layers	2 layers
Cosine, mean output to its gradient	−0.075 to −0.006	−0.046 to +0.042
Output against its gradient, over tokens	0.000 to +0.001	−0.003 to 0.000

Table 14. The push in trained GPT-2 small, over 50 batches of 32 windows from the corpus sample, with no optimizer step.

GPT-2's training run is not public, but Pythia's checkpoints are (Biderman et al., 2023). Pythia-70m's checkpoints do not show the push clearly (Table 15). Its rest forms by step 4,000, through its weights, as in the small models. But the push at each checkpoint points down in only a few layers, and its sign matches each neuron's next move at chance. The same test finds the push at the checkpoints of a small model (Table 19). So Pythia's push is weaker or less steady, or its far-apart checkpoints hide it. No part of Pythia's training setup weakens the push in a small model, alone or together (Table 20). Its batch was tested only at 4 times ours.

Step	Loss	Rest, L0–L5	Layers with the push down	Push against the next move
0	11.10	0.00 to −0.02	4 of 6	−0.00 to +0.15
512	5.19	−0.03 to −0.22	2 of 6	−0.06 to +0.16
1,000	4.28	−0.25 to −0.39	1 of 6	+0.04 to +0.24
2,000	3.79	−0.39 to −0.64	2 of 6	−0.00 to +0.12
4,000	3.56	−0.42 to −0.72	4 of 6	+0.03 to +0.14
16,000	3.39	−0.47 to −0.78	2 of 6	−0.01 to +0.07
143,000	3.35	−0.46 to −0.87	2 of 6	

Table 15. The push in Pythia-70m at its checkpoints. "Layers with the push down" counts layers whose median push ratio is below 0. The last column is the correlation over neurons between a neuron's push and how far its rest moves by the next checkpoint. The rows at steps 8,000, 32,000 and 64,000 are in `results.jsonl`.

The appendices *The push in GPT-2 small* and *The push in Pythia-70m* give the detail, with the calibration of the checkpoint test and Pythia's setup one part at a time.

10 Discussion

Why neurons rest low. Training keeps up a weak push on each neuron's rest that is steady over many steps. Under AdamW, a steady sign is what moves a parameter. So at each step the rest drifts by about the learning rate times the push's steadiness. Two terms make up the push. When a neuron's mean output is large, the output's own gradient points down, partly because the loss pulls the MLP's mean output toward 0. When the mean output is small, the output's push points up instead. It points up most on tokens below the rest, where the slope is low, so it moves h less than the mean slope predicts. In GELU's dip the slope is negative, and there a push up on the output is a push down on h. Neither term needs a dip or a floor, and tanh, with a symmetric shape and a mean output of 0, gets neither.

Where the rest slows. The rest slows when few tokens keep a slope to carry the push. So ReLU slows near 1.6 SDs below its mean, and softplus keeps going. Leaky ReLU at slope 0.1 can make its mean output negative, so it stops where its mean output is near 0. The loss pulls the MLP's mean output toward 0 through the final LayerNorm and through dropout, and the block LayerNorms do not take part. A constant offset at the final LayerNorm raises the loss because it adds the model's unigram prior to every token's logits (Table 10). One cause stays open. We do not know why the output's push points up most on tokens below the rest.

Other explanations. Three explanations that the results count against are worth stating. The first is that neurons seek GELU's dip. ReLU has no dip and rests as deep, and SiLU's deeper dip gives a shallower rest (Table 3). The second is that neurons rest low to filter interference, after Elhage et al. (2022). Width does not move the rest (Table 2), although these widths may not change interference enough. The third is that the loss needs a particular depth. Two AdamW rates end at the same loss with rests of −1.11 and −3.18 (Table 4). The word "seeks" in this paper means only that the gradient pushes that way. It does not show a designed function.

Links to other work. Dohare et al. (2024) find that dead units rise under Adam in continual learning, and we find that AdamW's step size sets how deep a neuron rests. The two results point the same way, but their units are off on almost every input, and ours rest at a level that keeps the negative pole on. Tigges et al. (2026) report, of a large mixture-of-

experts model, that "neurons are never actually off". A neuron that rests below GELU's dip fits that description: its positive pole is mostly off, and its negative pole is on for most tokens. Our step from their model to GPT-2's dense GELU MLPs is untested.

Why this matters for reading negative poles. Most of the cost of the negative side in GPT-2 is its resting level (What default-on negative poles do as a population). Clamping it at its mean adds 0.19 nats, and zeroing it adds 3.72. These runs fit that. Training pushes the level down, while variation below the dip has less effect. This supports the protocol's stop on auditing a default-on pole, but it does not show that a pole has no content.

11 Limitations

- **These are small models.** They have 4 layers and short training. Their pattern matches GPT-2 on offset, bias and weights, but not everywhere. GPT-2's L1 rests shallow (-0.24) and its L0 deep (-1.16), and small models show the opposite order. Results on small models suggest causes for GPT-2, and do not prove them.
- **The optimizer shapes the depth.** The first version of this analysis used only Adam at one rate, and warned that GELU and ReLU might agree because of it. Table 4 shows this in part. We tried one SGD rate, with momentum, and three AdamW rates. Other schedules, warm-up, and Adam's β values were not tried, except in the Pythia conditions.
- **Runs can go wrong with no error.** One 12,000-step run and two push runs went wrong, and we discarded them. Same-seed reruns match to about 0.001, so a mismatch is easy to see, but most conditions here were not rerun. The appendix *Failed, slow and drifting runs* gives each case.
- **Few seeds.** The baseline, ReLU, the fixed and fast centred conditions, softplus, both leaky ReLUs, SiLU, SGD and each AdamW rate have two seeds. The rest have one. The push has two seeds for softplus, both leaky ReLUs and SiLU, and one for GELU, ReLU and tanh.
- **Some of the newest runs have one seed.** The equal-rest runs at 0 have two seeds, which agree to 0.002 in push ratio. The offset, norm and no-dropout runs have two, which agree to 0.008 in cosine. The output and split runs have two, and every term has the same sign on both seeds. The depth runs and the calibration runs now have two. The depth runs agree to 0.003 in push ratio. The calibration runs agree in direction, but GELU's push at the checkpoints is weaker on seed 1 (Table 19).
- **Two leaky slopes.** At slope 0.01, the usual default, leaky ReLU rests like ReLU. At 0.1 it rests shallower. We did not try slopes between them.
- **Centring is a blunt tool.** It takes its mean from the batch, also at evaluation, and the gradient passes through that mean.
- **A held rest changes the model.** The held runs centre h, so the weights can no longer turn against $\bar{x}$. They train worse than free runs, at a loss of 5.56–6.02 against 5.49–5.83. So the push at an equal rest is the push in a centred model. A free model at that rest may get a different push.
- **Medians do not add.** The slope term, the spread term and the push are each a median over neurons, so the first two do not sum to the third.
- **GPT-2 is seen only at its end point.** Its training run is not public, so a match at the end point is a weak test.

12 Conclusion

Training sets where a neuron rests through a weak push on its rest that is steady over many steps, and the optimizer's step size sets how far that push takes it. The push needs a one-sided activation, but not GELU's dip. A neuron's mean output decides whether the slope term or the spread term carries it, and the final LayerNorm and dropout carry a pull on the MLP's mean output toward 0. Open questions are issues in the Linear project *Why neurons rest* (open issues). Three tests would sharpen the result:

- Find why the output's push of GELU and SiLU points up most on tokens below the rest. This is the one cause that stays open.
- Split dropout's part in the pull in the same way as the final LayerNorm's. Through the final LayerNorm, a constant offset costs loss because it adds the unigram prior. Dropout's part in the pull is not split up in the same way.
- Test Pythia's scale, its full batch and its gaps between checkpoints as causes of its weak push. At Pythia's checkpoints the push is weak and mixed, though the same test finds it in a small model. No part of Pythia's training setup weakens the push in a small model, alone or together. Its batch was tested only at 4 times ours. So Pythia's scale, its full batch or its gaps between checkpoints remain as causes.

References

Biderman, S., Schoelkopf, H., Anthony, Q., Bradley, H., O'Brien, K., Hallahan, E., Khan, M. A., Purohit, S., Prashanth, U. S. S., Raff, E., Skowron, A., Sutawika, L., & van der Wal, O. (2023). *Pythia: A suite for analyzing large language models across training and scaling.* ICML 2023. arXiv:2304.01373.

Dohare, S., Hernandez-Garcia, J. F., Lan, Q., Rahman, P., Mahmood, A. R., & Sutton, R. S. (2024). *Loss of plasticity in deep continual learning.* Nature, 632, 768–774. doi:10.1038/s41586-024-07711-7.

Dugas, C., Bengio, Y., Bélisle, F., Nadeau, C., & Garcia, R. (2001). *Incorporating second-order functional knowledge for better option pricing.* Advances in Neural Information Processing Systems 13 (NIPS 2000), 472–478. NeurIPS proceedings.

Elfwing, S., Uchibe, E., & Doya, K. (2017). *Sigmoid-weighted linear units for neural network function approximation in reinforcement learning.* arXiv:1702.03118.

Elhage, N., Hume, T., Olsson, C., Schiefer, N., Henighan, T., Kravec, S., Hatfield-Dodds, Z., Lasenby, R., Drain, D., Chen, C., Grosse, R., McCandlish, S., Kaplan, J., Amodei, D., Wattenberg, M., & Olah, C. (2022). *Toy models of superposition.* Transformer Circuits Thread. arXiv:2209.10652.

Gao, L., Biderman, S., Black, S., Golding, L., Hoppe, T., Foster, C., Phang, J., He, H., Thite, A., Nabeshima, N., Presser, S., & Leahy, C. (2020). *The Pile: An 800GB dataset of diverse text for language modeling.* arXiv:2101.00027.

Hendrycks, D., & Gimpel, K. (2016). *Gaussian error linear units (GELUs).* arXiv:1606.08415.

Loshchilov, I., & Hutter, F. (2019). *Decoupled weight decay regularization.* ICLR 2019. arXiv:1711.05101.

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., & Sutskever, I. (2019). *Language models are unsupervised multitask learners.* OpenAI.

Stolfo, A., Wu, B., Gurnee, W., Belinkov, Y., Song, X., Sachan, M., & Nanda, N. (2024). *Confidence regulation neurons in language models.* NeurIPS 2024. arXiv:2406.16254.

Tigges, C., Kowal, M., McGrath, T., Lewis, O., & Cammarata, N. (2026). *Neurons as an interpretable basis.* Goodfire Research. goodfire.com.

Data

- `NeelNanda/pile-10k`, revision `127bfedc`: the first 10,000 documents of the Pile. Its card gives the licence as `bigscience-bloom-rail-1.0`.

Appendices

Reproduce

Each command appends to its `--out`. The runs took about 25–40 minutes each on an Apple GPU. Run nothing else on the machine at the same time. Parallel runs on MPS were slower and failed once, and one run that overlapped other jobs went wrong with no error.

```
OUT=papers/why-neurons-rest/results.jsonl
uv run python scripts/train_resting.py gpt2 --out $OUT
uv run python scripts/train_resting.py train --seed 0 --steps 4000 --eval-every 500 --
cache tokens.pt --out $OUT
uv run python scripts/train_resting.py train --seed 1 --steps 4000 --eval-every 500 --
cache tokens.pt \
    --condition gelu --condition rest_0 --condition rest_learned_fast --condition relu \
    --condition rest_-1 --out $OUT
uv run python scripts/train_resting.py train --seed 0 --steps 12000 --eval-every 2000 --
cache tokens.pt \
    --condition gelu --condition rest_learned_fast --out $OUT
uv run python scripts/train_resting.py train --seed 0 --steps 4000 --eval-every 500 --
cache tokens.pt \
     --condition sgd --condition relu_sgd --condition softplus --condition leaky_relu --
condition silu \
       --condition lr_1e-4 --condition lr_1e-3 --condition relu_lr_1e-4 --condition
relu_lr_1e-3 --out $OUT
uv run python scripts/train_resting.py train --seed 0 --steps 12000 --eval-every 2000 --
cache tokens.pt \
    --part training_long --condition relu --out $OUT
PUSH="--steps 4000 --eval-every 500 --push-every 100 --cache tokens.pt --out $OUT"
uv run python scripts/train_resting.py train --seed 0 --condition leaky_relu_0.01 $PUSH
uv run python scripts/train_resting.py train --seed 0 --part training_push_rerun --
condition gelu \
     --condition relu --condition softplus --condition leaky_relu --condition silu --
condition tanh $PUSH
uv run python scripts/train_resting.py train --seed 1 --condition softplus --condition
leaky_relu \
    --condition silu --condition leaky_relu_0.01 --condition sgd --condition relu_sgd \
       --condition lr_1e-4 --condition lr_1e-3 --condition relu_lr_1e-4 --condition
relu_lr_1e-3 $PUSH
T="--eval-every 500 --push-every 100 --cache tokens.pt --out $OUT"
uv run python scripts/train_resting.py train --seed 0 --seed 1 --steps 4000 --part
training_equal_rest \
    --condition rest_0 --condition relu_rest_0 --condition softplus_rest_0 \
    --condition leaky_relu_rest_0 --condition silu_rest_0 --condition tanh_rest_0 $T
uv run python scripts/train_resting.py train --seed 0 --steps 1000 --part
training_push_split \
    --push-part push_split --condition gelu --condition silu $T
uv run python scripts/train_resting.py train --seed 0 --steps 4000 --part
training_push_split \
    --push-part push_split --condition softplus --condition relu $T
uv run python scripts/train_resting.py train --seed 0 --steps 1000 --part
training_output \
            --push-part push_output --condition gelu_output_centred --condition
silu_output_centred \
    --condition relu_output_shift --condition gelu_output_shift $T
uv run python scripts/train_resting.py train --seed 0 --steps 4000 --part training_depth
\
    --push-part push_depth --condition relu_rest_-1 --condition relu_rest_-2 --condition
relu_rest_-4 \
          --condition softplus_rest_-1 --condition softplus_rest_-2 --condition
softplus_rest_-4 \
    --condition rest_-1 --condition rest_-0.5 --condition relu_rest_-0.5 \
    --condition leaky_relu_rest_-1 --condition silu_rest_-1 $T
uv run python scripts/train_resting.py train --seed 0 --steps 2000 --part
training_offset \
         --push-part push_offset --condition relu --condition gelu --condition
```

```
relu_output_shift \
    --condition gelu_output_shift --condition softplus --condition silu --condition tanh
$T
uv run python scripts/train_resting.py train --seed 0 --steps 2000 --part
training_offset \
            --push-part push_offset --condition relu_norms_centred --condition
relu_all_norms_centred $T
O="--steps 2000 --part training_offset --push-part push_offset $T"
uv run python scripts/train_resting.py train --seed 0 --condition relu_dropout_0 --
condition dropout_0 $O
uv run python scripts/train_resting.py train --seed 0 --condition
relu_norms_centred_dropout_0 \
    --condition relu_all_norms_centred_dropout_0 $O
uv run python scripts/train_resting.py train --seed 1 --condition relu_dropout_0 --
condition dropout_0 $O
uv run python scripts/train_resting.py train --seed 1 --condition relu --condition gelu
\
    --condition relu_norms_centred --condition relu_all_norms_centred $O
uv run python scripts/train_resting.py train --seed 0 --condition
relu_final_norm_centred_dropout_0 $O
uv run python scripts/pretrained_push.py gpt2 --out $OUT
uv run python scripts/pretrained_push.py pythia --out $OUT
uv run python scripts/train_resting.py train --seed 1 --steps 2000 --part
training_offset \
     --push-part push_offset --condition relu_output_shift --condition gelu_output_shift
\
    --condition softplus --condition silu --condition tanh $T
uv run python scripts/train_resting.py train --seed 1 --steps 1000 --part
training_output \
            --push-part push_output --condition gelu_output_centred --condition
silu_output_centred \
    --condition relu_output_shift --condition gelu_output_shift $T
uv run python scripts/train_resting.py train --seed 1 --steps 1000 --part
training_push_split \
    --push-part push_split --condition gelu --condition silu $T
uv run python scripts/train_resting.py train --seed 1 --steps 4000 --part
training_push_split \
    --push-part push_split --condition softplus --condition relu $T
for SEED in 0 1; do
   uv run python scripts/train_resting.py train --seed $SEED --steps 1000 --part
training_spread_bins \
         --push-part push_spread_bins --spread-bins --device cpu --condition gelu --
condition silu \
      --condition relu $T
done
for SEED in 0 1; do  # Table 23, on an NVIDIA GPU (#3193)
   uv run python scripts/train_resting.py train --seed $SEED --steps 1000 --part
training_output_bins \
         --push-part push_output_bins --spread-bins --device cuda --condition gelu --
condition silu \
                        --condition relu --condition dropout_0 --condition
gelu_all_norms_centred_dropout_0 \
      --condition gelu_final_norm_centred_dropout_0 $T
done
C="--condition gelu_final_norm_centred_dropout_0 --condition
gelu_all_norms_centred_dropout_0"
for L in 0 1 2 3; do C="$C --condition gelu_final_L${L}_ln1_dropout_0 --condition
gelu_final_L${L}_ln2_dropout_0"; done
for SEED in 0 1; do  # Table 25 and Figure 10, on an Apple GPU (#6465)
```

```
    uv run python scripts/train_resting.py train --seed $SEED --steps 1000 --part
training_norm_pairs \
      --push-part push_norm_pairs --spread-bins --device mps $C $T
done
for C in dropout_0 relu_dropout_0; do
  uv run python scripts/train_resting.py train --seed 0 --condition $C --steps 2000 \
      --part training_calib --push-part push_calib --save-every 250 --save-dir ckpt $T
   uv run python scripts/pretrained_push.py small --condition $C --save-dir ckpt --out
$OUT
done
uv    run    python    scripts/train_resting.py    train    --seed    1    --condition
relu_norms_centred_dropout_0 \
                       --condition      relu_all_norms_centred_dropout_0      --condition
relu_final_norm_centred_dropout_0 $O
for C in gelu relu; do
  uv run python scripts/train_resting.py train --seed 0 --condition $C --steps 4000 --
eval-every 1000 \
       --cache tokens.pt --part training_ln_offset --save-every 1000 --save-dir ckpt --
out $OUT
done
uv run python scripts/train_resting.py train --seed 1 --condition gelu --steps 4000 --
eval-every 1000 \
     --cache tokens.pt --part training_ln_offset --save-every 1000 --save-dir ckpt --out
$OUT
for C in pythia_warmup pythia_adam pythia_parallel pythia_lr pythia_batch pythia_all; do
  uv run python scripts/train_resting.py train --seed 0 --condition $C --steps 2000 \
      --part training_pythia --push-part push_pythia --save-every 250 --save-dir ckpt $T
   uv run python scripts/pretrained_push.py small --condition $C --save-dir ckpt --out
$OUT
done
for SEED in 0 1; do  # Table 24, on an NVIDIA GPU (#4093)
   uv run python scripts/train_resting.py train --seed $SEED --condition dropout_0 --
steps 2000 \
       --part training_thinned --push-part push_thinned --save-every 50 --save-dir ckpt
$T
  uv run python scripts/pretrained_push.py small --condition dropout_0 --seed $SEED --
save-dir ckpt \
      --step 0 --step 250 --step 500 --step 1000 --step 2000 --part thinned_pythia --out
$OUT
  uv run python scripts/pretrained_push.py small --condition dropout_0 --seed $SEED --
save-dir ckpt \
      $(for S in 0 250 500 750 1000 1250 1500 1750 2000; do echo --step $S; done) \
      --part thinned_every_250 --out $OUT
  uv run python scripts/pretrained_push.py small --condition dropout_0 --seed $SEED --
save-dir ckpt \
      --part thinned_every_50 --out $OUT
done
uv run python scripts/ln_offset.py gpt2 --out $OUT
uv run python scripts/ln_offset.py small --checkpoint ckpt/gelu_seed0_step4000.pt --out
$OUT
for F in ckpt/gelu_seed1_step*.pt ckpt/relu_seed0_step*.pt ckpt/gelu_seed0_step0.pt; do
  C=$(basename $F | cut -d_ -f1)
  uv run python scripts/ln_offset.py small --slopes-only --condition $C --checkpoint $F
--out $OUT
done
uv run python scripts/ln_offset.py frequency --condition gelu --cache tokens.pt --out
$OUT \
    $(for F in ckpt/gelu_*.pt; do echo --checkpoint $F; done)
uv run python scripts/ln_offset.py frequency --condition relu --cache tokens.pt --out
```

```
$OUT \
    $(for F in ckpt/relu_*.pt; do echo --checkpoint $F; done)
# Why L2 and L3 shut, and dropout's routes (#3192), on an NVIDIA GPU
for S in 0 1; do
            for      C      in      relu_final_norm_centred_dropout_0      relu_dropout_0
relu_norms_centred_dropout_0; do
    uv run python scripts/train_resting.py train --seed $S --condition $C --steps 2000 -
-eval-every 100 \
        --cache tokens.pt --device cuda --part training_shut --out $OUT
  done
  for C in relu gelu; do
    uv run python scripts/train_resting.py train --seed $S --condition $C --steps 2000 -
-eval-every 500 \
        --cache tokens.pt --device cuda --part training_dropout_routes --save-every 2000
--save-dir ckpt --out $OUT
  done
done
for C in relu gelu; do
  uv run python scripts/dropout_routes.py --condition $C --device cuda --out $OUT \
      --checkpoint ckpt/${C}_seed0_step2000.pt --checkpoint ckpt/${C}_seed1_step2000.pt
done
uv run python scripts/paper_figures.py
```

The seed-1 equal-rest runs ran after the seed-0 ones, in a run of their own. The split and output runs ran before the recorder kept the offset and token correlations, so their rows lack those fields. The binned runs ran on the CPU, while the GPU ran other runs.

The 12,000-step runs of GELU and the fast control were labelled `training_long` by hand. The `--part` option does this now.

Reading the changes to the baseline

The seeds agree on Table 2. The baseline's depth differs by at most 0.04 between seeds, and ReLU's by at most 0.03. Conditions with no seed 1 entry ran once. Mirrored GELU is only a check. It is the baseline with h and the weights negated, so its rest must flip, and it does.

ReLU rests as deep as GELU. It has no dip and no flat region below zero that differs by depth, so the dip cannot be its target. ReLU is also indifferent to scale, so a rest near -1 has no special meaning for it. That similarity may have a simple cause in the training (see *Limitations*). Tanh does not shift. This is weak evidence, because tanh is symmetric and no explanation gives it a direction. Its spread of h falls to 0.15–0.19 on L1–L3, so these neurons mostly stay in its near-linear region. It has the worst loss here.

Width does not move the rest. If neurons rest low to filter interference, as the toy models of *Related work* suggest, narrower MLPs have more interference and should rest lower. They do not. These widths may not change interference enough, so this counts against that version of the idea and does not remove it.

Regularisation is not needed. The no-dropout run also has no weight decay, and its rest forms at L1–L3. Dropout changes L1–L3 without a clear trend. L0 is the exception, and deepens with dropout (-0.16, -0.34, -0.57). L0 reads the embeddings, so this may be real, but it is three points on one seed.

Reading the other activations

Each one-sided activation in Table 3 rests below 0, by the same route. The weights turn against $\bar{x}$, and the bias stays between -0.06 and 0.

The dip does not set the depth. SiLU is GELU's shape stretched along h by about 1.7, so its dip is at -1.28 and not -0.75. If neurons sought the dip, or the activation's scale, SiLU would rest deeper than GELU. It rests at about half GELU's depth.

A floor does not set the depth either. One idea was that a neuron goes down until its output is near 0, its "off" state. Softplus fits: its output at -4.68 is 0.009. Leaky ReLU fits too: it passes 10% below 0, so going deeper never turns it off, and it rests shallower than ReLU. SiLU does not fit. Its output is within 0.1 of 0 only below -3.54, against -2.25 for softplus, so this idea puts SiLU deeper than softplus. It rests at -0.49, where its output is -0.19.

Softplus widens as well as shifts. Its SD of h reaches 2.17, more than twice the baseline's, and it trains the worst of the one-sided activations here.

Reading the optimizer runs

At the same step, the depth in Table 4 is close to proportional to the rate. For GELU, L1's depth grows 3.0× from 1e-4 to 3e-4, and 2.9× from 3e-4 to 1e-3. Adam moves each weight by about the learning rate at each step, whatever the gradient's size. So this is what a push of the same sign at every step gives, with nothing that pushes back.

At the same loss, a larger rate still rests deeper. The 3e-4 and 1e-3 runs end at nearly the same loss (5.509 and 5.505 for GELU), with rests of -1.11 and -3.18. So the loss does not need a particular depth, and the depth is not only a measure of how far training has progressed.

SGD gives a shallower rest. At a loss of 5.9, GELU's L1 is at -0.28 under SGD and -0.68 under AdamW at the base rate. The weights still turn against $\bar{x}$ for 99%–100% of neurons. So the rest does not need Adam, but its depth depends on the optimizer.

ReLU and GELU part outside AdamW's lower rates. Under SGD, ReLU rests about twice as deep as GELU at every layer from L1 to L3. So their agreement at the base rate comes partly from AdamW, as the caveats of the first version of this analysis suspected. L0 deepens fastest under SGD, where it rests deepest of the four layers. That is the order of GPT-2 small, whose L0 rests at -1.16. With one seed and one rate, this is only a hint.

Seed 1 agrees. At step 4,000 on seed 1, GELU's L1 rests at -0.37, -3.22 and -0.30 at AdamW 1e-4, AdamW 1e-3 and SGD. ReLU's rests at -0.42, -4.75 and -0.60. The largest difference from seed 0 is ReLU at 1e-3, at 0.71. The two larger rates still end at nearly the same loss.

The push ratio in Table 5 does not depend on the rate. For ReLU it is -0.018 to -0.020 at all three AdamW rates, and for GELU -0.014 to -0.019. The depth grows about 3× with each 3× step in rate. So AdamW takes a step in proportion to its rate on a push of about the same steadiness. That is the fixed distance per step that the long runs and the rates suggest.

SGD has a push of the same steadiness but a shallower rest. SGD's step follows the push's raw size, not its ratio to its RMS. ReLU's raw push under SGD is 1.7× GELU's, and ReLU rests 2.3× as deep as GELU under SGD. Under AdamW the two have about the same ratio and rest at about the same depth. This may explain why ReLU and GELU agree only under AdamW, but it is two pairs of runs.

The raw push gets smaller as the rest gets deeper. At 1e-3 the raw push is 3–6× smaller than at 1e-4, while its ratio stays the same. So the push's size cannot set the depth under AdamW. Its steadiness can.

Reading the push

The push in Table 6 keeps its sign over many steps, not over one. On a single step it is close to a coin toss, at 50.0%–51.7%. Its mean over 100 steps points down in 85%–100% of windows for each one-sided activation, and in 42% for tanh. So it is a small bias in a noisy gradient. The gradient on the weights along $\bar{x}$ behaves the same way. This fits the rest's steady deepening and its dependence on the learning rate (Table 4). It does not fit a large push that stops at a target.

Its size gives the order of depth. Softplus has the largest ratio and the deepest rest. Then come ReLU, leaky ReLU at slope 0.01 and GELU, at -0.019 to -0.020 and about -1.1. Then come leaky ReLU at slope 0.1, SiLU and tanh. That is the order in which they rest. The relation is not linear. Softplus rests about twice as deep as the others' ratio of depth to push would give.

The order forms early. Over steps 3,500–4,000, ReLU, leaky ReLU at slope 0.01, GELU and SiLU all have push ratios of -0.010 to -0.013, and softplus has -0.019. So late in the run, the push no longer gives the order. The differences come from the first 1,000 steps. There, the order is softplus, ReLU and leaky ReLU at slope 0.01, leaky ReLU at slope 0.1, GELU, then SiLU and tanh. That is the order of depth, except that GELU's early push is weaker than that of leaky ReLU at slope 0.1, while GELU rests deeper.

GELU and SiLU are first pushed up. Their push points up until step 200 for GELU and step 400 for SiLU. The push on ReLU, leaky ReLU and softplus points down from the first step. The gradient on the output of GELU and SiLU points up all through training. Their push on h turns down only because the activation's slope weights the tokens, and where those tokens lie changes as the rest moves. For ReLU, both leaky ReLUs early on, and softplus, the gradient on the output points down from the start. So a single property of the loss, such as "make each neuron's output smaller", does not explain all six.

The weights carry the push. 97%–99% of each step's movement of the rest is $\bar{x} \cdot \Delta W$, and 1%–3% is the bias. This matches GPT-2 small (Table 1), where the offset and not the bias sets the rest.

Table 6 is a correlation along each run. We measured each push where that run's neurons rest at that step. So the activation may set the push, or the depth that a run reaches may set its push. The equal-rest runs of Table 7 separate the two: the depth sets most of the order.

Leaky ReLU at slope 0.01 rests like ReLU. It rests at -1.11 and -1.16 at L1, against -1.13 for ReLU. So the shallow rest of leaky ReLU at slope 0.1 comes from its large slope below 0. One run has gradient spikes. Leaky ReLU at slope 0.1, seed 0, has a few windows near step 3,600–3,900 where the push ratio jumps above 0. Its seed 1 has none. We report medians and ratios, because such spikes dominate the means of the raw push.

The runs of Table 7 hold every activation at the same rest. h is centred per neuron over each batch and moved to 0, as in `rest_0`. Then no parameter can move the rest, and the gradient on that rest is the push at an equal state for all 4,000 steps. Held at 0, softplus, ReLU and leaky ReLU at slope 0.1 get the same push, to 0.001. GELU is a little weaker, SiLU is weaker again, and tanh gets none. The two seeds agree to 0.002.

Softplus and leaky ReLU differ from ReLU only by depth. At the same rest they get ReLU's push. In free runs, softplus rests 4 times as deep as ReLU and leaky ReLU 0.6 times as deep. So where a neuron rests changes the push it gets, and the order of depth comes from that change.

A held rest gets a stronger push than a free one. Each held run's push is 1.6–4.7 times its free run's. A free rest moves away from the push, and the push weakens as it does (Table 11). GELU and SiLU are pushed up for the first 100 steps. Held at 0, as in free runs, their first window points up. ReLU, leaky ReLU and softplus point down from the start.

The terms of the push

Runs labelled `training_offset` repeat seven activations for 2,000 steps, seed 0, and record both terms of Table 8. Runs labelled `training_push_split` record them for GELU and SiLU over 1,000 steps, in finer detail. The losses of both sets match the earlier runs exactly, except ReLU's, which differ by up to 0.004.

Table 8 shows two groups of activations. For ReLU and softplus, the output's push points down all through training, and the slope term carries the push. Their spread term is small, and points up early. For GELU and SiLU, the output's push points up, and the spread term carries the push down. For tanh, both terms are 0.

GELU and SiLU start up because their output's push points up. Near $h = 0$, GELU's and SiLU's slopes are close to 0.5 on every token. So the slope term follows the output's push, which points up for both. The spread term points down from the first steps and grows as h spreads. It passes the slope term near step 200 for GELU, and between steps 400 and 500 for SiLU. SiLU turns later because its slope varies less. Near 0, GELU's slope changes by 0.80 for each unit of h, and SiLU's by 0.50. So the same spread of h gives GELU a larger spread term. SiLU also has a larger slope term.

The spread term comes from tokens with a low slope. Runs labelled `training_spread_bins` repeat GELU, SiLU and ReLU for 1,000 steps on seeds 0 and 1. They split each neuron's spread term by bins of h, and the bins add up to the same spread term as `training_push_split`. Table 16 gives each bin's part as a median over neurons. It is a mean over L1–L3, over steps 201–1,000 and over the two seeds. The seeds agree to 0.13 in each bin.

Activation	Below −1.5	−1.5 to −0.75	−0.75 to −0.25	−0.25 to 0.25	0.25 to 0.75	0.75 to 1.5	Above 1.5
GELU	−0.10	−3.09	−1.62	−1.58	+0.24	+0.53	0.00
SiLU	0.00	−1.20	−1.73	−0.66	+0.14	+0.31	0.00
ReLU	−0.01	−1.61	−0.18	−1.93	+3.41	+0.65	0.00

Table 16. The spread term by bins of h, as a median over neurons and a mean over L1–L3, steps 201–1,000 and two seeds. Units are 10^{-6}.

The steepest tokens push up, not down. From 0.75 to 1.5, each activation's part is above 0, and only 28%–32% of neurons get a part below 0. So the spread term does not come from the steep part of the activation.

The push down comes from below the rest. A token's part is its output's push times the amount by which its slope differs from the mean. Below the rest the slope is below the mean, so a negative part means that the output's push points up there. For GELU, the bin from −1.5 to −0.75 gives the largest part, and 90% of neurons get a part below 0 there. Most of that bin lies in GELU's dip, where the slope is negative. There, a push up on the output is a push down on h. SiLU's slope is still above 0 down to −1.28, so its largest part comes from −0.75 to −0.25.

In the first 200 steps, the push down comes from just below 0. Then the rests are shallow, and few tokens reach the dip. GELU's largest part is then −1.3 to −1.4, from −0.75 to

−0.25, on each seed.

ReLU has the same pattern, but its two sides cancel. Below 0 its slope is 0, and its part is below 0. Above 0 its slope is 1, and its part is above 0, at +3.4 from 0.25 to 0.75. So ReLU's spread term is small, and its slope term carries its push. We do not know why the output's push points up below the rest. Over tokens, the output and its gradient do not correlate (Table 17), so it is not a pull on each token's output.

The output's push itself points up below the rest, and it still does when no LayerNorm and no dropout can see a constant (#3193). Runs labelled `training_output_bins` repeat Table 16's runs and add three conditions. For each neuron and each bin, they record the output's push per token, not only its part of the spread term. These runs are on an NVIDIA GPU, and they reproduce Table 16's spread-term shares to 0.01 in every bin. Table 23 gives the share of neurons whose output's push points up in each bin.

Condition	**Below −1.5**	**−1.5 to −0.75**	**−0.75 to −0.25**	**−0.25 to 0.25**	**0.25 to 0.75**	**0.75 to 1.5**	**Above 1.5**
GELU	0.73	0.90	0.68	0.18	0.53	0.67	0.54
SiLU	0.73	0.86	0.83	0.16	0.53	0.69	0.61
ReLU	0.49	0.62	0.23	0.05	0.62	0.62	0.50
GELU, no dropout	0.73	0.90	0.67	0.18	0.52	0.66	0.53
GELU, every norm centred, no dropout	0.73	0.88	0.71	0.12	0.26	0.51	0.64
GELU, final norm centred, no dropout	0.56	0.60	0.63	0.36	0.43	0.52	0.56

Table 23. The share of neurons whose output's push per token points up, in each bin of h. It is a mean over L1–L3, steps 201–1,000 and two seeds. The seeds agree to 0.04 in each bin.

For GELU and SiLU, the output's push points up on 68%–90% of neurons below −0.25, and on 16%–18% near 0. So the up push comes from the tokens below the rest, as Table 16 inferred from the spread term. ReLU has no such pattern below −0.75. Without dropout, GELU's shares change by at most 0.01. With every norm centred and no dropout, no LayerNorm and no dropout mask can see a constant offset. The shares below −0.25 then stay within 0.03 of GELU's. So the pattern does not reach the output's push through a constant offset.

Centring only the final norm, with no dropout, weakens the pattern: 56%–63% of neurons below −0.25 have an up push. Centring every norm does not weaken it. At step 1,000 both conditions have a higher loss than GELU, 6.37–6.45 against 6.20–6.21, by about the same amount. So the loss does not explain the difference.

No single block norm brings the pattern back (#6465). Runs labelled `training_norm_pairs` centre the final norm and one block norm, for each of the eight, with no dropout. They also repeat the final-only and every-norm conditions on the same machine, an Apple GPU, and match Table 23 to 0.01. Adding a block norm before attention (`ln_1`) leaves the shares at or below the final-only level. Adding one before the MLP (`ln_2`) lifts the −1.5 to −0.75 bin most: to 0.72 at L1, 0.71 at L2 and 0.67 at L3, from 0.60. Every norm centred gives 0.88. The seeds differ by up to 0.11 in a bin, so even that lift is small. So the restoration needs several block norms, mostly the MLP's. The losses lie between 6.32 and 6.54, near the reference conditions', so they do not explain the shares. #6785 tests the final norm with all four `ln_2` against all four `ln_1`.

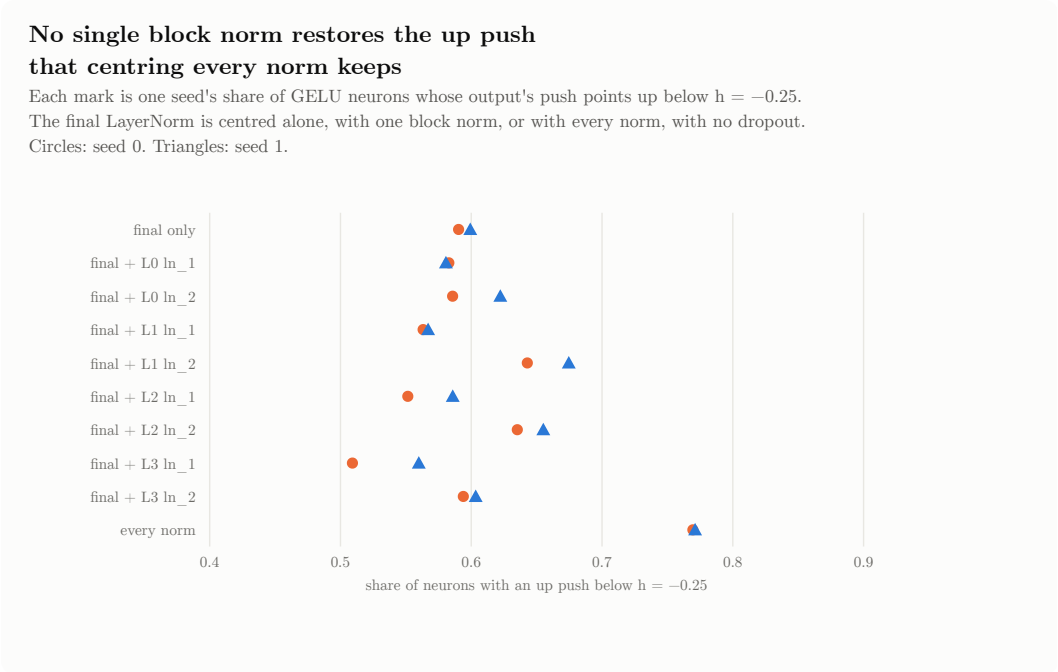

Figure 10. The share of GELU neurons whose output's push points up below h = −0.25, a mean over its three bins, L1–L3 and steps 201–1,000. One row per condition, one mark per seed.

Centred, no dropout	Below −1.5	−1.5 to −0.75	−0.75 to −0.25	−0.25 to 0.25	Loss at step 1,000
final only	0.55	0.60	0.63	0.36	6.38
final + L0 `ln_1`	0.53	0.58	0.63	0.36	6.38
final + L0 `ln_2`	0.58	0.61	0.63	0.37	6.40
final + L1 `ln_1`	0.55	0.59	0.56	0.37	6.40
final + L1 `ln_2`	0.60	0.72	0.65	0.27	6.54
final + L2 `ln_1`	0.53	0.59	0.59	0.38	6.37
final + L2 `ln_2`	0.61	0.71	0.62	0.32	6.39
final + L3 `ln_1`	0.47	0.55	0.58	0.40	6.35
final + L3 `ln_2`	0.59	0.67	0.54	0.34	6.32
every norm	0.73	0.88	0.70	0.13	6.44

Table 25. The share of neurons whose output's push points up, by bin of h, for each norm condition, as in Table 23. A mean over L1–L3, steps 201–1,000 and two seeds. The loss is the mean of the two seeds.

Removing the output's push removes the up phase. Runs labelled `training_output` centre each neuron's output a over the batch. Then $-\partial L / \partial a$ sums to 0 over tokens, and only the spread term remains. GELU's push ratio is then −0.048 over steps 1–100 and SiLU's −0.045, against +0.005 and +0.016 in free runs. At step 1,000, GELU's L1 rests at −0.60 against −0.42, and SiLU's at −0.40 against −0.15.

A neuron's mean output decides the group. At init, with h's SD near 0.31, ReLU's mean output is 0.124 and GELU's is 0.038. `relu_output_shift` adds −0.086 to ReLU's output, and `gelu_output_shift` adds +0.086 to GELU's. A constant changes neither the slope nor the shape. It moves only the mean output. Shifted ReLU then falls in GELU's group: its

output's push points up, and the spread term carries its push. Shifted GELU's output's push falls to about 0, and its push points down from the first step.

Swapping the mean outputs swaps the depths. At step 1,000, shifted ReLU rests at −0.44 to −0.48 at L1–L3, against −0.64 to −0.69 for ReLU. Shifted GELU rests at −0.51 to −0.85, against −0.28 to −0.42 for GELU. So a larger mean output gives a deeper rest, with the same activation shape.

The pull on the mean output

The loss pulls the MLP's mean output toward 0. Runs labelled `training_offset` repeat seven activations for 2,000 steps. At every step they record the cosine between the MLP's mean output, $\bar{a} \cdot W_{\text{out}} + b_{\text{out}}$, and the gradient on b_{out}. A positive cosine means that gradient descent shrinks the mean output. It is positive in all 140 windows of the seven runs, at +0.007 to +0.10. The correlation over neurons between each neuron's mean output $\bar{a}$ and its output's push is negative in all 140 windows, at −0.002 to −0.10. So a neuron with a larger mean output is pushed down more (Table 17). Tanh shows the same pull, but its neurons' mean outputs have both signs, so the pull has no net direction.

It is a pull on the mean, not on each token. If the loss pulled each token's output toward 0, the output's gradient on each token would follow the output. Over tokens, the two correlate at −0.003 to +0.002, a median over neurons. So the pull acts on the mean output only.

Activation	Cosine, mean output to its gradient	Mean output against output push, over neurons	Output against its gradient, over tokens
ReLU	+0.07	−0.03	−0.002
GELU	+0.06	−0.04	0.000
ReLU, shifted down	+0.06	−0.04	−0.001
GELU, shifted up	+0.05	−0.05	−0.001
Softplus	+0.07	−0.02	−0.003
SiLU	+0.04	−0.03	+0.001
Tanh	+0.02	−0.03	+0.002

Table 17. The pull on the mean output by activation, as means over steps 1–2,000 and over L1–L3. The last column is a median over neurons.

The final LayerNorm and dropout carry the pull (Table 9). Each of the two routes makes the pull alone. Dropout alone gives +0.059, and the final LayerNorm alone gives +0.046. Removing dropout from a model with every route open lowers the cosine by 0.03 for ReLU and 0.013–0.014 for GELU, on each seed. The earlier test centred every norm but kept dropout, so it could not rule LayerNorm out.

The block LayerNorms do not make the pull. With only them open, the cosine is −0.017 and −0.019 on the two seeds, and is above 0 in 1 window of 20 on each. But in these runs L2 and L3 nearly shut: h's SD there fell to 0.07–0.10, against about 0.5 in the other runs. Their loss is 5.937 and 5.950, against 5.758 and 5.753 with no dropout. At L1, where h's SD is 0.42, the cosine is −0.02 to −0.04 over the last 5 windows. The appendix *Why L2 and L3 shut* gives the cause.

With no route open, the pull is 0. Then every reader of the residual stream centres it over tokens. So the gradient on b_{out} is exactly 0, and so is the output's push summed over tokens. The cosine is noise, at −0.008 to +0.007 over the two seeds.

The rest does not need the pull. In that run, the slope term is 0, because the output's push is. But ReLU's push ratio at L1 over steps 1,001–2,000 is −0.075 and −0.074 on the two seeds, against −0.077 with dropout. The spread term carries all of it, at −17.3 against −15.0 (10^{-6}). So for ReLU the pull adds to the slope term, but the spread term alone gives the same push. These runs shrink $\bar{x}$, so their depths say nothing here.

What the final LayerNorm does with a constant

The cost comes from the shift, and the shrink lowers the loss. The direction is each MLP's mean output, at the final LayerNorm. Over the 9 trained checkpoints and 4 layers, the shrink lowers the loss in 36 of 36. The shift raises it in 34 of 36, and the full slope is a pull in 32 of 36. So a constant does flatten the logits, but flatter logits help these models.

The shift adds the model's unigram prior. Through the final LayerNorm, the MLPs' mean output changes the logits by one fixed vector, the same on every token. In every trained checkpoint, that vector correlates with the log frequency of each token in the training text, at +0.75 to +0.91. At initialisation the correlation is 0.00. On GELU seed 0 at step 4,000, the tokens it raises most are `, the \n in a` and `..` Those it lowers are rare. So more mean output makes the model more sure of frequent tokens, and the loss rises.

The direction matters, not the sharing. A random direction of the same norm, shared by all tokens, has a full slope of −0.023 on GELU seed 0. GPT-2 small gives the same answer: a random direction costs as much when each token gets its own as when all tokens share one. This is in the size sweep, which also has GPT-2.

The block LayerNorms push the other way. The same offset, seen only by the LayerNorms of the later blocks, lowers the loss. At L0 and L1 it does so in 9 of 9 trained checkpoints each, and the shift carries it. At L2, which only one block reads, the sign is mixed. So the later blocks use a constant in the residual stream. This is why the block LayerNorms do not make the pull. With only them open, the earlier runs measured a cosine of −0.017 and −0.019.

With every route open, the sign depends on the layer. L0's total is a push up in 9 of 9 checkpoints, and L1's in 8 of 9, because the block route is larger. L2's and L3's totals are pulls, in 8 and 6 of 9. In training, dropout adds a pull of its own, and these slopes leave it out.

GPT-2 small has no pull here either. Along its MLP mean output, the full slope is −0.099 at the final LayerNorm, with shrink at −0.032 and shift at −0.067. The shrink still flattens its logits, with an entropy slope of +1.82, and the shift sharpens them, at −1.13.

Why L2 and L3 shut

With only the block LayerNorms open, nothing pulls the MLPs' mean output toward 0, and the later blocks even use a constant in the residual stream (*What the final LayerNorm does with a constant*). So the mean output grows until it fills the input of those LayerNorms. A LayerNorm centres each token over its features, not over tokens, so a constant that all tokens share passes through it. When the shared part is several times the part that differs between tokens, the LayerNorm's output barely differs between tokens, and neither does the next MLP's h.

To test this, the evaluation of `train_resting.py` records two more measures for each block. The *offset ratio* r at a LayerNorm's input is the norm of the residual stream's mean over tokens, over the mean norm of each token's deviation from that mean. The other is the norm of the MLP's mean output. Three ReLU conditions with no dropout ran for 2,000 steps on

two seeds, evaluated every 100 steps (Table 21). The runs were on an NVIDIA GPU. They reproduce the earlier losses to 0.016 or better.

Run	Seed	Loss	MLP mean output, L0–L3	r at L2's `ln_2`	r at L3's `ln_2`	h SD, L2	h SD, L3
Only block norms open	0	5.937	4.1, 6.0, 5.6, 2.5	2.93	3.72	0.10	0.08
Only block norms open	1	5.934	4.2, 7.5, 8.1, 3.6	3.73	4.41	0.09	0.07
Block and final norms open	0	5.758	2.3, 0.7, 0.8, 1.5	0.46	0.38	0.58	0.57
Block and final norms open	1	5.753	2.4, 0.6, 0.7, 1.5	0.49	0.38	0.57	0.58
Only the final norm open	0	5.898	2.3, 1.9, 2.1, 4.1	0.31	0.30	0.57	0.58
Only the final norm open	1	5.914	2.5, 2.1, 1.9, 4.2	0.34	0.29	0.57	0.59

Table 21. The three ReLU runs with no dropout at step 2,000. r is the offset ratio at the input of the block's second LayerNorm, which feeds its MLP.

The hypothesis holds on both seeds. With only the block norms open, the shared constant at the input of L2's and L3's MLP LayerNorms is 2.9 to 4.4 times the part that differs between tokens. With the final norm open as well, it is 0.38 to 0.49. The mean outputs show why: with no pull, L1 and L2 reach norms of 5.6 to 8.1, against 0.6 to 0.8 with the final norm open. When the block norms are centred and only the final norm is open, the mean outputs are also large, at up to 4.2. But no block LayerNorm sees them, and nothing shuts. L1 has a ratio of about 0.8 in the shut runs, and its h keeps an SD of 0.44–0.45.

The constant comes first, and the shut follows. By step 100, the ratio at L3's MLP LayerNorm is already 3.9 and 5.6 on the two seeds, while h's SD there is still 0.24 and 0.17. The ratio then settles at about 3 to 4.4, and h's SD keeps falling to 0.07–0.08 by step 2,000. A LayerNorm alone would scale the varying part by about $1 / \sqrt{1 + r^2}$, which is 0.26 at $r = 3.72$. h's SD falls further, to about 0.14 of the open run's. So the weights also turn away from what is left. These runs saved no checkpoints, so the paper does not test this by removing the constant from a trained model.

Dropout's routes

Dropout makes a pull of its own (Table 9). It drops each element of the MLP's output on each token, so a mean output c that every token shares becomes per-token noise:

$$n_t = c \odot \left(\frac{m_t}{1 - p} - 1 \right),$$

where m_t keeps each element with probability $1 - p$, and $p = 0.1$. The noise grows with c. If it costs loss, gradient descent shrinks c. `scripts/dropout_routes.py` splits this as `ln_offset.py` splits the final LayerNorm's part. On checkpoints of ReLU and GELU at step 2,000, it adds the noise of one MLP, times a scale α, to one set of readers at a time. The sets are the later blocks' LayerNorms only, the final LayerNorm only, and the residual stream itself, as in training. It gives the slope of the expected loss at $\alpha = 1$ by autograd, over 8 masks for each batch of the 128 test windows. The model is otherwise in eval mode. A positive slope is a pull.

Checkpoint	Later blocks' LayerNorms	Final LayerNorm	Residual stream	Block share
ReLU, seed 0	+3.5	+1.8	+4.4	0.67
ReLU, seed 1	+3.1	+2.0	+4.3	0.60
GELU, seed 0	+2.1	+1.6	+2.7	0.57
GELU, seed 1	+2.5	+1.5	+3.1	0.62

Table 22. Dropout's pull on the MLPs' mean output by route, as the slope of the loss summed over L0–L3, in units of 10^{-3} nats. The block share is the block route's part of the two routes' sum.

Every route pulls. The slope is positive for every MLP on each route that it can reach, in all four checkpoints. The noise that dropout makes from the constant costs loss through the later blocks' LayerNorms too, although the same constant without noise lowers the loss there. The later blocks carry 57%–67% of the two routes' sum, mostly from L0, whose mean output is the largest (1.3–1.7). For L2 and L3 the final LayerNorm carries most of it. The residual stream's slope is less than the two routes' sum, by 0.7–0.8 (10^{-3}) at L0. The noise has mean 0, so its cost is of second order in its size, and a cross term between the two routes is negative.

Push against depth

Runs labelled `training_depth` hold activations at rests below 0, on seeds 0 and 1. Tables 11 and 18 give seed 0. Seed 1 gives the same push ratios to 0.003 and the same losses to 0.025. The rows at 0 are the equal-rest runs.

Held rest	ReLU	Softplus	GELU	Leaky ReLU, slope 0.1	SiLU
0	5.659	6.019	5.645	5.674	5.686
−0.5	5.526		5.613		
−1	5.562	5.919	5.657	5.717	5.749
−2	5.748	5.820			
−4	5.748	5.645			

Table 18. The loss at step 4,000 of the runs held at rests below 0, on seed 0.

ReLU's push fades as its neurons sink. At −1 it keeps a ninth of its push at 0. At −2 and −4 no token reaches 0, because h's SD stays near 0.3. So no gradient passes ReLU, its MLPs never train, and both runs end at the same loss.

Leaky ReLU at slope 0.1 is pushed back up from −1. So it has a rest between 0 and −1 where the push is 0, and its free rest of −0.70 lies there. Below 0, it outputs $0.1h$, so a deep rest makes its mean output negative. The pull on the mean output toward 0 then points up (Table 9). At its free rest, with an SD of about 0.6, its mean output is about −0.04. ReLU and softplus cannot have a negative mean output, so the pull never turns them back.

GELU's push is steadier at −1, not larger. Its push ratio is −0.032 at −0.5 and −0.042 at −1. But its push, as a median over neurons and a mean over L1–L3 and the steps, is smaller at −1: −5.2 against −7.3 (10^{-6}). The slope term falls from +2.1 to +0.7, because few tokens near −1 have a large slope. The push's size from step to step falls more, from about 230 to about 125 (10^{-6}), estimated as the push over its ratio. The push ratio is the mean over that size, so it grows. Its h also widens as the rest deepens, to an SD of 0.95 at −1. SiLU keeps less than half its push at −1. It falls from −0.041 to −0.018, and its h widens to an SD of 1.26.

Softplus keeps its push at every depth. Its slope $\sigma(h)$ is above 0 on every token. The push ratio is a mean over an RMS, so a small slope does not weaken it. Only fewer tokens, or noisier ones, weaken it.

This is why softplus rests about twice as deep as its push predicts. In free runs, ReLU's push ratio falls from -0.044 in steps 1–500 to -0.011 in steps 3,001–4,000. Over that time its rest goes from 1.2 to 1.6 SDs below the mean. Softplus's falls from -0.089 to -0.020, and its rest goes down to 2.2 SDs. Under AdamW, depth adds up the push over the run. So an activation whose push survives at depth keeps going.

Softplus also turns a push into more movement late in the run. After step 1,000, each unit of its push ratio moves the rest about twice as far as ReLU's, GELU's and leaky ReLU's. Its mean input is also larger, at 0.47 of the input's norm against 0.39 for ReLU. We did not find the rest of the cause.

Some depth helps. Softplus's loss falls from 6.019 at 0 to 5.645 at -4, better than free softplus at 5.826. ReLU trains best of its held rests at -0.5, and GELU at -0.5. Leaky ReLU and SiLU train worse at -1 than at 0, which fits their shallower free rests.

Reading the runs on the rest's value

A forced rest does not separate the causes in Table 12. `rest_-1` is near the baseline's depth and trains as badly as `rest_0`. It is also fixed for every neuron. It starts at -1, where the fresh model's narrow h (SD about 0.31) sits on GELU's flat part. Its loss at step 1,000 is 6.40–6.42, against 6.27–6.28 for `rest_0`, and it has almost caught up by step 4,000. So its cost looks like a slow start, but we did not test this directly.

At the base rate, a learned rest stays near 0. It reaches only -0.09 at L1, so this control mostly repeats `rest_0`. Adam at this rate can move one parameter by about 1.2 in 4,000 steps, and only if every step has the same sign. The baseline's bias moves only about 0.03. The weights can shift the rest faster, because many weights multiply a large mean input.

At 30× the rate, a learned rest reaches nearly the same depth by step 4,000. It reaches -0.92, -0.85 and -0.71 at L1–L3 on seed 0 and -0.92, -0.86 and -0.69 on seed 1. That keeps the baseline's order: L1, then L2, then L3, then L0. The weights no longer turn against $\bar{x}$. Reaching that depth lowers the loss to 5.574 and 5.581. A gap remains. The fast control's loss stays 0.065–0.072 above the baseline at step 4,000, and longer training does not close it.

The rest does not settle at -1 in Table 13. The baseline's L1 passes -1 by step 4,000 and reaches -1.46. Growth slows, with only 0.03 added over the last 2,000 steps at L1, so it may settle later. We cannot say where.

ReLU keeps pace with GELU. Its L1 is at -0.92, -1.12, -1.27, -1.35, -1.40 and -1.49 at the six steps of the table. Its loss is within 0.04 of the baseline's at each. Its L3 goes deeper, to -1.43 against -1.15.

The two routes keep pace. At every step the learned value is at 81–90% of the baseline's depth at L1. The learned value has a much higher limit on its rate, so a limit on one parameter's rate does not explain the pace of this control.

The loss gap does not close. It stays between 0.064 and 0.076 at every step in the table. So the gap is not only unfinished training. Centring itself, or the difference in depth, may cost this much. So the runs do not measure the rest's full value.

Failed, slow and drifting runs

One run was discarded. The first 12,000-step baseline gave a loss of 5.908 at step 4,000 and a much wider h. The earlier run with the same seed gave 5.509. It overlapped other jobs on the machine, and it showed no error. A rerun with nothing else running gave 5.510. The fast control ran after that overlap, and it matched its earlier values at steps 2,000 and 4,000. We did not find the cause, so we kept the discarded rows in the data as `training_long_discarded`.

Same-seed reruns match to about 0.001, so a mismatch is easy to see. The leaky ReLU run took 42 minutes against about 25 for the others, while a browser process used much of the CPU. A rerun of its first 500 steps matched to 0.0005 in loss. At step 4,000, ReLU's 12,000-step run differs from the earlier ReLU run by 0.006 in loss and at most 0.01 in rest. That is more than the usual 0.001.

The push runs found two more failures. The seed-1 run of AdamW at 1e-4 moved L1's rest to +3.92 by step 500. It ended at a loss of 7.37, against 5.877 for seed 0, whose L1 rests at −0.37. The next run, ReLU at the same rate, stopped with a NaN loss at step 112. We did not find the cause. We kept the first as `training_discarded` and `push_discarded`, and reran both. The seed-0 reruns that recorded the push matched the earlier rows exactly for GELU, softplus, SiLU and tanh. ReLU's differed by 0.005 in loss, and leaky ReLU at slope 0.1 by 0.039 in loss and 0.05 in rest. The push table uses the reruns, labelled `training_push_rerun`, and the other tables keep the earlier rows.

One norm run partly failed. With only the final LayerNorm centred and no dropout, L2 and L3 nearly shut on seed 0 (*The pull on the mean output*). Seed 1 shows the same, with h's SD there at 0.10 and 0.08. So the condition causes it, and the run did not fail. We do not know why.

ReLU reruns drift a little. Same-seed reruns of GELU, SiLU, softplus and tanh match the earlier rows exactly, also with the new recording. ReLU's differ by up to 0.004 in loss at step 2,000 and 0.002 at step 4,000, as the earlier ReLU rerun did. The step-0 evaluation of one GELU rerun differed by 0.007, while every later step matched exactly.

Some runs were slow. Leaky ReLU held at 0 and ReLU held at −1 took 37–39 minutes against about 25, while other jobs used the machine. The same-seed reruns before and after them matched.

The push in GPT-2 small

Rows labelled `gpt2_push` record the push and its terms for trained GPT-2 small, as `scripts/resting_push.py` does for the small models. Each is over 50 batches of 32 windows from the corpus sample, with no optimizer step. So they show the push at the end of training, and not while the rest formed.

At its trained weights, GPT-2's rest is not pushed (Table 14). Half of its neurons are pushed down and half up, and the median push ratio is near 0 in every layer. The small models' push points down in 85%–100% of windows while their rest forms. So GPT-2's rest looks settled. With no dropout, the terms have the small GELU models' signs. The slope term points up in all 12 layers and the spread term down in 11, as for GELU in Table 8. Here they cancel.

There is no pull toward 0. With no dropout, the cosine of the mean output with its gradient is below 0 in all 12 layers. So at these weights the loss would grow the mean output a little. With dropout it has both signs. For 768 dimensions, a random cosine has an SD of about 0.036, so the mean over 50 batches has an SD of about 0.005. We cannot see GPT-2 while it

trains. Its training run is not public, so these rows cannot show whether the two terms carried its rest down.

The push in Pythia-70m

Rows labelled `pythia_push` record Pythia-70m at 10 checkpoints from step 0 to step 143,000, with `scripts/pretrained_push.py pythia`. Each row reads the rest and the push as the GPT-2 rows do, over 50 batches of the corpus sample's windows. Pythia has GELU, parallel attention and no dropout. Each row also gives, for each layer, the correlation over neurons between a neuron's push and how far its rest moves by the next checkpoint (Table 15).

Pythia's rest forms early and by the same route. Its neurons rest at -0.25 to -0.39 by step 1,000 and at -0.42 to -0.72 by step 4,000. Then they change little. From step 512 on, the median neuron's weights point against the mean input, as in GPT-2 and the small models.

The push at a checkpoint does not point down steadily. While the rest forms, at steps 512–2,000, only 1 or 2 of the 6 layers have a median push below 0. Half the neurons are pushed down at every checkpoint (43%–56%). A neuron's push predicts its move only weakly. Up to step 4,000, the correlation is above 0 in 27 of 30 layers and intervals, at up to $+0.24$. But the push's sign matches the move's for only 45%–54% of neurons. After step 8,000 the correlation is near 0 (-0.11 to $+0.07$).

The pull on the mean output is there late. From step 16,000 on, the cosine of the mean output with its gradient is above 0 in 5 or 6 of the 6 layers. Early on it has both signs. Pythia has no dropout. So by the small models' result, its final LayerNorm is the likely route.

The checkpoints are far apart. Between checkpoints are 488–79,000 steps, each with 2 million tokens of the Pile, during warm-up for the first 1,430. Our push is from 50 batches of 4,000 tokens of another corpus, at fixed weights. The calibration below asks whether such a test can find a push that is known to be there.

Calibrating the checkpoint test. Rows labelled `small_push` apply the same test to small models whose push during training is known. GELU and ReLU, both with no dropout as in Pythia, trained for 2,000 steps and saved a checkpoint every 250 steps. `scripts/pretrained_push.py small` then read each checkpoint as the `pythia` command reads Pythia's. The same runs recorded their push during training, in rows labelled `push_calib`. During training, the push over each 250 steps points down in most layers and intervals. It does so in 47 of 48 on seed 0, and in 45 of 48 on seed 1.

Test, L1–L3 for the small models, L0–L5 for Pythia	GELU, seed 0	GELU, seed 1	ReLU, seed 0	ReLU, seed 1	Pythia-70m, steps 0–4,000
Checkpoints and layers with the push down	24 of 27	18 of 27	26 of 27	23 of 27	13 of 30
Push against the next move, median	+0.16	+0.14	+0.12	+0.15	+0.07
Push against the next move, above 0	23 of 24	24 of 24	24 of 24	24 of 24	27 of 30
Neurons whose push has the sign of their move, median	58%	55%	55%	56%	51%

Table 19. The checkpoint test on small models whose push is known, against Pythia-70m.

The test finds a steady push where there is one (Table 19). In the small models, the push at a checkpoint points down in nearly every layer. Its sign matches the neuron's next move for

more than half the neurons. That holds in 47 of 48 layers and intervals on seed 0, and 46 of 48 on seed 1. GELU's seed 1 is weaker at the checkpoints. Its push points down in 18 of 27 checkpoints and layers, against 24 on seed 0. That is still more than Pythia's 13 of 30. On both seeds, the correlation with the next move and the share of neurons that match it are above Pythia's.

Pythia's push is weaker or less steady. Its push points down in fewer than half its layers while its rest forms. Its sign matches the move at chance, and the correlation is about half the small models'. So the weak result is a property of Pythia or of its gaps between checkpoints, and not of the test alone. The gaps can still hide a push. Pythia's checkpoints are 2 to 300 times further apart than ours, and each step sees 500 times the tokens. A run with Pythia's batch size and schedule would tell the two apart.

Pythia's training setup, one part at a time. Pythia differs from the small models in its schedule, its optimizer, its parallel attention and its batch. The `pythia_` conditions add one of these at a time to the no-dropout GELU model. Each trains for 2,000 steps on seed 0 and saves a checkpoint every 250 steps, and `pretrained_push.py small` reads the checkpoints. Their rows are labelled `training_pythia`, `push_pythia` and `small_push`.

- `pythia_warmup` uses Pythia's learning rate of 1e-3. It grows the rate linearly over the first 715 steps. Pythia warms up over 1,430 of the 4,000 steps that the test reads, and 715 keeps that share.
- `pythia_adam` uses Pythia's AdamW betas of (0.9, 0.95) and weight decay of 0.01.
- `pythia_parallel` lets the attention and the MLP read the same input, as in GPT-NeoX, Pythia's architecture.
- `pythia_lr` uses Pythia's learning rate of 1e-3 with no warm-up.
- `pythia_batch` uses 4 times our batch. Pythia's batch is 500 times ours, and 4 times fits the time we have.
- `pythia_all` adds all of these parts at once.

Test, L1–L3	No dropout, seed 0	No dropout, seed 1	Warm-up at 1e-3	Pythia's AdamW	Parallel attention	Rate 1e-3, no warm-up	4× batch	All of Pythia's parts	Pythia-70m
Checkpoints and layers with the push down	24 of 27	18 of 27	22 of 27	19 of 27	24 of 27	22 of 27	21 of 27	21 of 27	13 of 30
Push against the next move, median	+0.16	+0.14	+0.12	+0.10	+0.15	+0.13	+0.13	+0.12	+0.07
Push against the next move, above 0	23 of 24	24 of 24	22 of 24	22 of 24	23 of 24	20 of 24	23 of 24	22 of 24	27 of 30
Neurons whose push has the sign of their move, median	58%	55%	59%	56%	56%	55%	55%	55%	51%
Intervals of 250 steps with the push down in training	23 of 24	21 of 24	21 of 24	24 of 24	22 of 24	22 of 24	24 of 24	21 of 24	

Table 20. The checkpoint test on small models with one part of Pythia's training setup at a time, and with all of them.

No part of Pythia's setup makes the push look like Pythia's, and neither do all of them together (Table 20). Each condition stays inside or near the range of the two no-dropout seeds. The push points down in 19 to 24 of 27 checkpoints and layers, against 13 of 30 in Pythia-70m. Its correlation with the next move is +0.10 to +0.15, against +0.07. Pythia's AdamW comes closest on that correlation. With all the parts, the push in training still points down in 21 of 24 intervals.

So the setup, at the sizes tested here, does not explain Pythia's weak push. What is left is Pythia's scale: its size, its batch of 500 times ours, and its far-apart checkpoints. These runs cannot tell those apart. Each condition has one seed. The two no-dropout seeds differ by 6 of 27 in the first row, so a difference that size between conditions is noise.

The gaps between checkpoints don't explain it either (#4093). Runs labelled `training_thinned` train the no-dropout GELU model on seeds 0 and 1 for 2,000 steps, and save a checkpoint every 50 steps. `pretrained_push.py small --step` then reads each run at three spacings, in rows labelled `thinned_every_250`, `thinned_pythia` and `thinned_every_50`. These runs were on an NVIDIA GPU.

- **Every 250 steps** is Table 19's spacing, and it reproduces Table 19 exactly on both seeds.
- **Pythia's spacing** reads steps 0, 250, 500, 1,000 and 2,000. Those are Pythia's checkpoints 0, 512, 1,000, 2,000 and 4,000, scaled from its first 4,000 steps onto our 2,000, with 256 rounded to the nearest saved step.

- **Every 50 steps** is the densest that the runs allow.

Test, L1–L3	**Every 250, seed 0**	**Every 250, seed 1**	**Pythia's spacing, seed 0**	**Pythia's spacing, seed 1**	**Every 50, seed 0**	**Every 50, seed 1**	**Pythia-70m**
Checkpoints and layers with the push down	24 of 27	18 of 27	13 of 15	10 of 15	104 of 123	93 of 123	13 of 30
Push against the next move, median	+0.16	+0.14	+0.15	+0.13	+0.20	+0.20	+0.07
Push against the next move, above 0	23 of 24	24 of 24	10 of 12	12 of 12	119 of 120	114 of 120	27 of 30
Neurons whose push has the sign of their move, median	58%	55%	58%	56%	58%	57%	51%

Table 24. The checkpoint test on one densely saved run per seed, read at three spacings, against Pythia-70m.

Thinning the checkpoints to Pythia's spacing leaves the test where it was (Table 24). The push points down in 67%–87% of checkpoints and layers, against 43% in Pythia-70m. Its correlation with the next move is +0.13 to +0.15, against +0.07. Denser checkpoints make the correlation stronger, at +0.20, but thinner ones don't bring it near Pythia's. So, at this relative spacing, the gaps don't explain Pythia's weak push. The scaling is a choice: Pythia's gaps hold far more tokens, at 2 million a step. The full batch and the size are left, and #6499 holds them.

The warm-up deepens the rest, as a higher rate does. At step 2,000, L1 rests at −1.48, against −0.66 for the no-dropout run at 3e-4 (Table 4). Pythia's rate without the warm-up deepens the rest further. At step 2,000, L1 rests at −2.63, against −1.48 with the warm-up. With all of Pythia's parts, it rests at −0.93.