Experiment 2.9.4: closed-loop regularizer weights

> We replaced the timed regularizer anneal with a feedback controller, so that each weight climbs while its constraint is being violated and settles back once it is met. It kind of worked but not very well, and it adds complexity.

Ex-2.9.3 traced anchoring failures to an instability late in training. The anchored solution is metastable during the high learning-rate plateau: the regularizers hold it in place, and the timed anneal takes that support away while the optimizer is still hot. We fixed it statically, by halving the peak learning rate. That works, but it sets the timing by hand.

This experiment lets the regularizer weights respond to the training signals, so a weight climbs while its constraint is being violated and settles back down once the constraint is met.

The mechanism is a small feedback controller (dual ascent with hysteresis[1]) acting on the anchor and anti-anchor weights. It reads only signals available during training, with no privileged ground-truth probe:

- Each controlled term keeps a running average of its own raw value (an EMA[2]). The anchor term is measured on labeled samples only, so its average updates on the ~6% of batches that carry a label.
- The weight λ rises while its average sits above an engage threshold, decays (5× faster) once it falls below a release threshold, and holds steady in the band between the two. That keeps the ordinary early transient from winding the weight up, and lets λ return to zero in a healthy run.
- λ is capped at 0.15, close to the dopesheet constant 0.1 from earlier experiments.

The anti-subspace weight is left uncontrolled. Its raw value has a floor set by the red samples themselves, and a sensor that cannot see labels has no way to tell that floor apart from leakage. So we hold it at its small late value instead of annealing it. Learning rate and `separate` follow the dopesheet as before.

We ran ten conditions, 32 seeds each, scored by the ex-2.9.2 `redirect` intervention: {static timed-anneal, controller} × peak LR {0.10 risky, 0.05 safe} with the fallback term; the same pair with the fallback term removed, at 0.10, since ex-2.9.3 showed that is where catastrophic failures actually appear; and a sensitivity grid over the controller parameters (targets ×0.75, ×1.5; gains ×0.5, ×2). The experiment is `experiment.py`:

```
bin/mini run docs/m1/ex-2.9.4/experiment.py --app moda
```

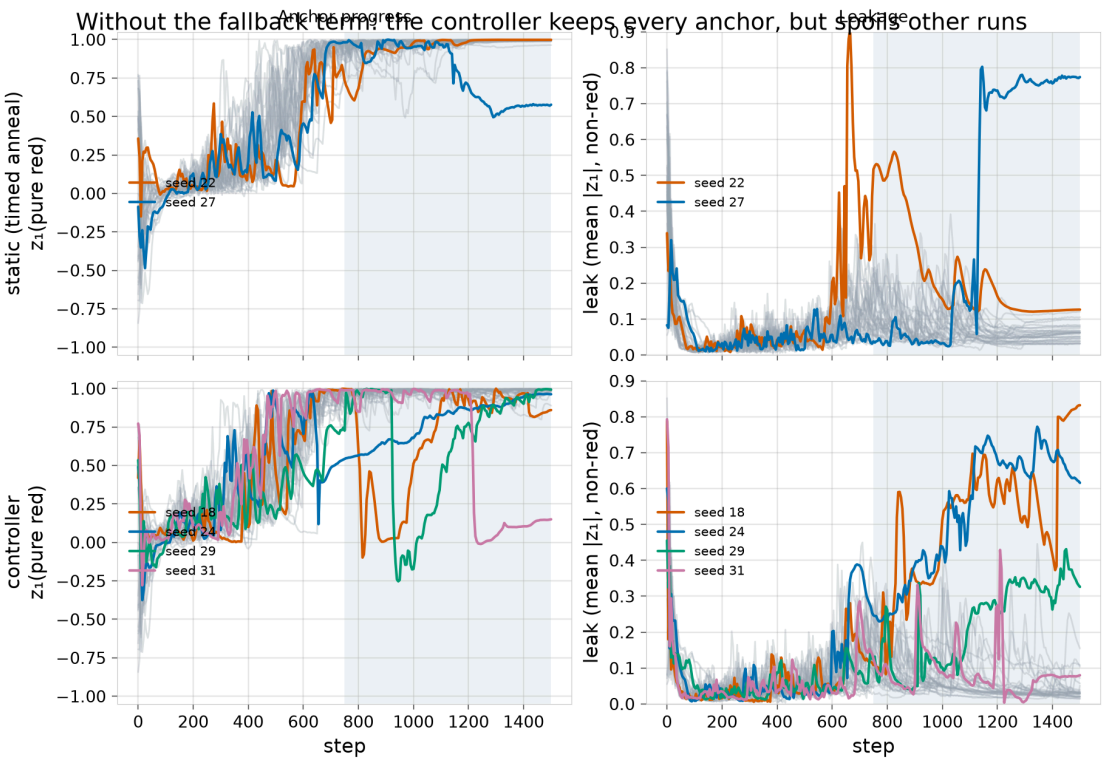
128 + 64 + 128 runs completed across ten conditions. This is a clear negative result. Without the fallback term, the feedback loop does what it was designed to do on the seeds the static schedule loses: its 2 catastrophic seeds (22, 27) train cleanly under control (redirect scores 0.94, 0.92). But the controller introduces 4 new failures on other seeds (18, 24, 29, 31), so the overall failure rate does not improve. With the fallback term on (the recipe we adopted), no condition has a single catastrophic failure, so there is nothing left for the controller to rescue. So we will probably keep the static schedule.

The fallback-free test: rescues and new failures

Ex-2.9.3 showed that catastrophic failures only appear when there's no fallback term and a high LR. That makes it the fair place to test a rescue mechanism. The charts below show anchor progress and leakage for all 32 seeds under each variant.

The static schedule failures are in the anchor panel: the anchor forms, then falls away late in training. With the controller, the same panel shows the feedback at work, with runs dipping hard in the middle of the plateau and getting pulled back up to 1.

The response has a cost, though, and it doesn't always succeed. The leak panel shows runs where a sustained penalty drags pinkish labeled colors onto the axis until the geometry is spoiled, and one run loses the anchor outright with its weight pinned at the cap.

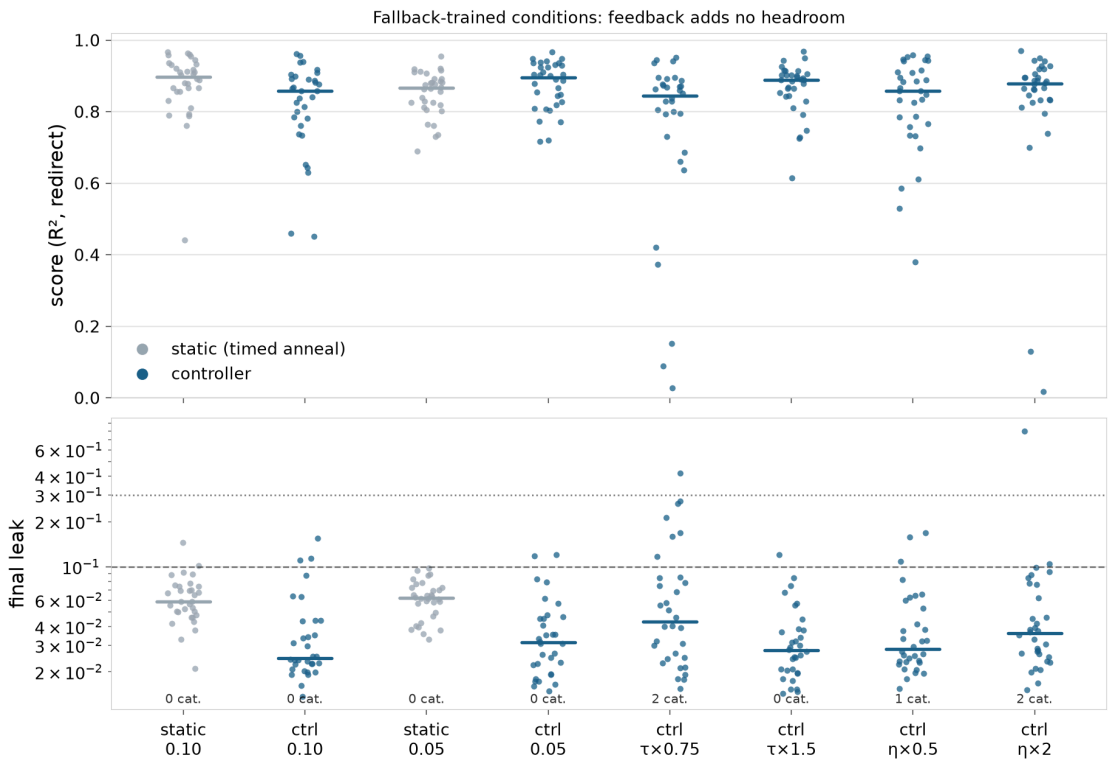


The weight trajectory separates the two outcomes. In the rescues, λ engages while the anchor is forming, then releases once the constraint is met. On seed 27 (the worst under the static schedule, still below $z_1 = 0.7$ at step 1500), the controlled run holds z_1 near 1 through the plateau, with λ decaying to 0.03. In the failures the labeled anchor EMA never reaches its target, so λ climbs to the 0.15 cap and stays there: 4 of 4 catastrophic runs had mean λ > 0.13, versus 1 of 25 clean ones.

This is a problem with the sensor. The anchor term is measured on noisy labels, and a pinkish color placed off the axis on purpose looks the same, to that measurement, as a red that has drifted off it. On most seeds the average settles below target anyway; on some it can't, and overcorrects. One run lost its anchor even with its weight pinned at maximum.

With the fallback term

The fallback term from ex-2.9.2 turns out to prevent every catastrophic failure on its own: none in 448 fallback-trained runs across this experiment and ex-2.9.3, against 7 of 224 without it.



At the safe peak the two approaches are interchangeable on the score (medians 0.90 for the controller vs 0.87 static; floors 0.72 vs 0.69). The controller halves the typical leak (0.031 vs 0.062), since a steady low-level penalty does keep the axis cleaner on average, but it worsens the tail past the degraded threshold and costs about 2–3× in reconstruction (0.000029 vs 0.000012, both still tiny). At the risky peak the pattern is the same, with 0 catastrophes between the two conditions, thanks to the fallback term.\ The sensitivity grid is fairly clear: Tightening the targets by 25% (τ×0.75) or doubling the gain (η×2), with the fallback term still on, produces 2 and 2 catastrophic runs respectively, and halving the gain leaves 1. Loosening the targets (τ×1.5) is safe (0).

Lessons

The idea seemed reasonable: protection on demand instead of on a timer. The controller response is fine; the measurement it relies on is the problem. The training-time signal for anchor health was the anchor loss on noisy labels, and in this experiment, that signal couldn't tell a drifting red from a pink that should be off-axis.

So the stack we keep is the plain one: the fallback term (which, across two experiments, has removed every catastrophic failure), peak LR 0.05, the original timed anneal, and cheap endpoint screening.

One engineering note. `DynamicProp.set()` in `mini.temporal` can retarget a value mid-flight, which is what a controller needs, but experiments consume schedules through `realize_timeline`, which bakes the dopesheet into a static array before training. Dopesheet keyframes and runtime `set()` calls would then compete over the same prop. So the controller in this experiment lives inside the training loop instead, carrying its dual state in the `lax.scan` carry, while the dopesheet drives the props it doesn't control. That split worked well and is worth keeping, even though the controller itself is not adopted.

---

1. Dual ascent enforces a constraint by putting a price on it. While the constraint is being violated the price (here the weight λ) rises, which pushes the optimizer to satisfy it; once it is met, the price relaxes. Hysteresis means the price uses two thresholds with a gap between them rather than one, the way a thermostat lets the room drift a little before switching, so the weight doesn't chatter on and off. ↩

2. Exponential moving average: a running average that weights recent samples more than old ones, so it follows a changing signal without keeping its whole history. ↩