Assignment 6: Generalization

CS 312 — Ungraded · Due: November 19, 2026, 1:00pm Pacific

Setting. Use d8 (eight blocks, width 512, QK normalization), the course AdamW recipe, peak LR .003, WD 3.2, and batch size 64 sequences per optimizer update. The DNA vocabulary and context are described on the following page. The default baseline trains for 32 epochs over the fixed 30k pool (983.04M tokens of exposure), unless varied below. Repeating the pool increases exposure, not unique data. Keep training and validation sequences disjoint. Use warmup followed by linear LR decay to zero, except in Problem 3(a), which also tests warmup followed by constant LR.

Instructions. Submit one PDF report (at most 20 pages) on Gradescope by the deadline, with labeled plots and a short synthesis for each problem. Starter code and reference files are in `experiments/a6_generalization/` in the assignments repository. Reuse matching reference runs. Tune central hyperparameters when requested, then confirm the main comparisons with a second seed, except for Problem 3(a); pair seeds and data order across methods. For fine-tuning, vary data order too, since the pretrained weights are fixed. Select recipes and report NLL on the same validation set; no additional test split is required. Repeated selection can make validation results optimistic. Report successful and failed comparisons.

Reference histories are in `hashimoto-group/public-alchemy-a6`. Prepared target/validation data, dense and batch checkpoints, three pretrained initializations, and no-dropout ensemble predictions are shared on Modal in `cs312-shared-data`, volume `hard-dl-dclm-v1`, mounted read-only at `/root/shared_data/a6_generalization/v1`. The release manifest lists every file. MoE/looping runs, dropout ensemble banks, and the original pretraining datasets are not supplied.

The DNA dataset.

We study next-token prediction on biological sequences from the Arc Institute's OpenGenome2, the corpus used to train Evo 2. Our course subset mixes metagenomes, eukaryotic genic windows, bacterial and archaeal genomes, mRNA, viral sequences, noncoding RNA, organelles, and promoters. We take bounded prefixes of raw FASTA sources at a pinned revision, with quotas approximating the published raw-base mixture.

Preparation and vocabulary. We discard FASTA headers, uppercase sequences, and tokenize each character separately. The **19-token vocabulary** contains A/C/G/T, eleven ambiguity symbols, U and X, and only two special tokens: BOS and EOS. We preserve ambiguous characters, append EOS to each record, and concatenate records within each source without padding. Each 1,023-token payload gets a BOS prefix, making a 1,024-token chunk that may cross record boundaries. Predict the next token causally; BOS is context only. No species tags or reverse-complement augmentation are added.

The assignment pool. After a fixed random chunk shuffle, we select **30,000 training chunks: 30.72M tokens including BOS/EOS**, plus a disjoint **1,000-chunk validation set**. Use this same validation set for hyperparameter selection and all reported NLL comparisons. Problems 1–3 reuse this pool; Problem 4 varies fine-tuning data. Chunks are disjoint, but neighboring genomic regions or homologous sequences may cross splits: this tests generalization within the mixture, not to held-out species or genomes. Use the supplied prepared splits.

Generalization motivation.

This assignment is a major departure from the typical pre-training setup that we have. Instead of limiting ourselves to one pass of a 600M dataset with d8 architecture, we will instead consider a 30M dataset that we can train on as many times as we like. The goal is to get the best possible held-out loss using these 30M tokens.

To get more value out of our limited data, the two simplest methods are (1) epoching (2) scaling parameter count. Following the investigations in Kim et al. (2025), we can establish a baseline where we tune learning rate, weight decay, and epoch count for each model size and method. We find that despite noise, there is a trend where larger models end up getting better loss. However, this trend plateaus, meaning that we can't simply scale model size for arbitrarily lower losses. Furthermore, this improvement on validation loss translates to downstream benchmarks (specifically, the DART task (Patel et al., 2024) where we measure whether a model can distinguish between a real sequence and a randomly shuffled variant of the same sequence)

In this assignment, you will assess different ways of reducing the loss in this regime where we are limited by data. To scope our investigation, we will mostly be training d8's. A good starting point is LR 0.003, WD 3.2, and 32 epochs.

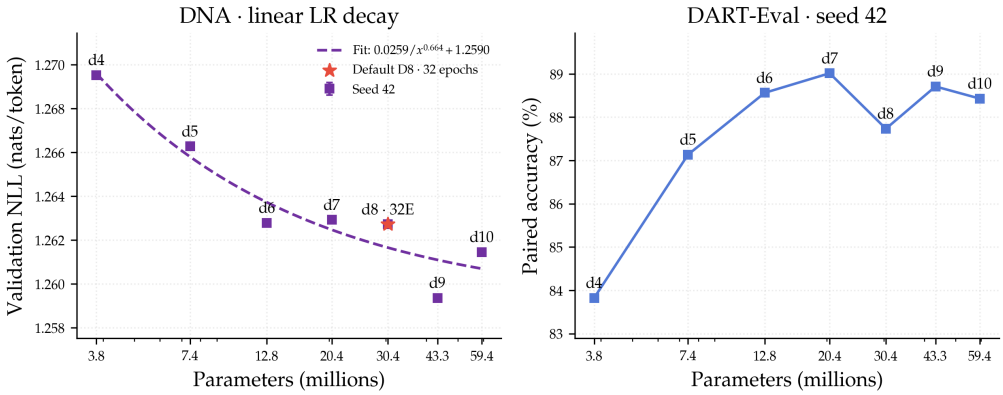


Figure 1: Parameter scaling and DART evaluation. Models trained with linear LR decay on the same 30k DNA chunks, with LR/WD/epochs selected on validation data. Left: validation NLL and a descriptive power-law fit; the star marks the default 32-epoch D8 at LR .003 and WD3.2, the same D8 selected in the tuned suite. Right: seed-42 checkpoints of the same selected recipes on 8,192 regulatory element–shuffled-control pairs, averaging both strands. DART does not select recipes; training did not explicitly exclude its test chromosomes.

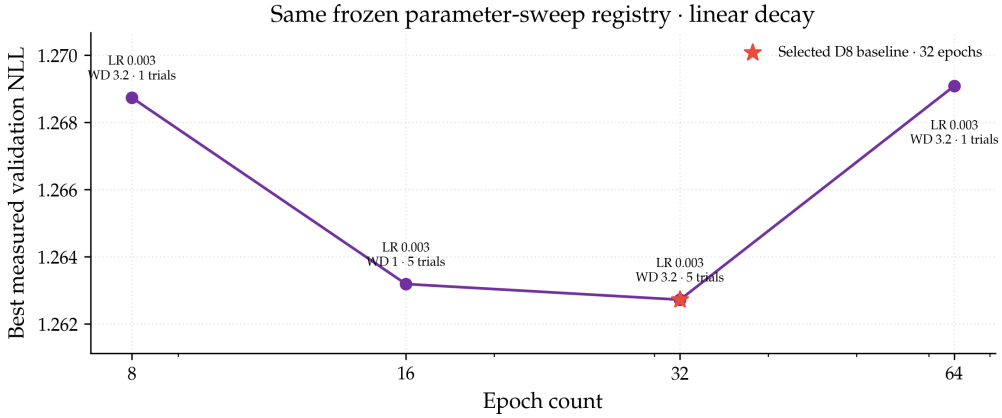


Figure 2: Epoch-count reference for Problem 2. Best measured final validation NLL at each training duration on the same 30k chunks (30.72M tokens/pass), seed 42, with linear LR decay. Annotations show the LR and WD selected on validation data within the measured candidates at each horizon; epoch ticks double from left to right. Uses the same frozen registry as Figure 1: 1/5/5/1 recipes at 8/16/32/64 epochs; later refinements are excluded. Problem 2(a)(i) separately uses a fixed 16-epoch grid and varies only LR and WD.

Problem 1: Architecture

For (b–c), train the architectures from Assignment 5 on the same 30k-sequence pool for 32 passes and WD 3.2; these runs are not supplied. Tune each architecture's LR using three candidates: .001, .003, and .01. Compare with the supplied baseline dense d8 at LR .003. Pair seeds and data order, keeping unique data and exposure matched.

(a) **Recall: parameter scaling.** We provide a **jointly tuned suite of dense d4–d10 models**, with recipes, histories, and checkpoints. Each size is tuned locally over LR, WD, and epoch count on the same 30k-chunk pool, using one fixed model seed. Analyze the supplied runs and Figure 1 without repeating the sweeps. Plot validation NLL against parameter count and fit $L(N) = L_\infty + AN^{-\alpha}$. Is the trend monotonic, and do returns diminish? Show how selected epoch count and training compute vary with size.

(b) **Recall: MoE from Assignment 5.** Reuse the four-expert, learned top-1 MoE with quantile balancing from the previous assignment. After LR tuning, compare its validation NLL with dense d8 and report $\Delta L = L_{\text{MoE}} - L_{\text{d8}}$. Does extra stored capacity help at the same active compute?

(c) **Recall: looping from Assignment 5.** Reuse the RMS+recall looping architecture: one prelude, six passes through a shared core block, and one coda. After LR tuning, compare its validation NLL with dense d8 and report $\Delta L = L_{\text{loop}} - L_{\text{d8}}$. How much stored capacity is saved, and what is the effect on loss at the same active compute?

(d) **Exploration.** Explore architectures that may improve data efficiency at a fixed parameter budget. Compare your best design with a parameter-matched dense model.

Since our models can now overfit to the training data, it is important to experiment with various forms of regularization to prevent the model from generalizing poorly. In this problem, we will experiment with two forms of regularization: norm control via weight decay and hyperball and stochastic regularization via dropout and ensembling.

Problem 2: Regularization

(a) **Weight decay.**

(i) **Recall: LR $\times$ WD.** Use the A6 starter `quadratic_fit.py` with `references/lr_wd_runs.json`; no new training is required. At the fixed epoch count $E = 16$, fit a quadratic loss surface in log LR and log positive WD, as in Assignment 2, and plot its contours. Also plot validation loss on the y-axis against LR $\times$ WD on the x-axis. How well does the product organize the losses? How much larger is the best WD than the earlier WD .1 reference?

(ii) **Hyperball.** In Assignment 2, we showed how we can eliminate weight decay by instead utilizing Hyperball. Does the benefit of Hyperball also translate to this setting? Compare the loss of LR-tuned Hyperball versus our baseline without WD tuning (WD 0.1) and with WD tuning (WD 3.2). Which does best, and what fraction of performance do the other methods achieve? The AdamW controls are supplied; train and LR-tune Hyperball.

(b) **Ensembling and dropout.** Test the intuition that dropout helps by approximating an ensemble. Use the 32-epoch baseline for all three investigations.

(i) **Dropout alone.** Use the baseline d8 recipe (LR .003, WD 3.2, 32 epochs) and compare no dropout with attention-probability dropout at rates .05, .15, and .25. Train the dropout models yourself, keeping LR and WD fixed, and disable dropout at evaluation. Does it help a single model?

(ii) **Independent ensembles.** Use the supplied validation predictions from five no-dropout models with different initialization seeds on the same data. Plot loss against ensemble size $K \in \{1, 2, 3, 4, 5\}$, averaging predicted probabilities. Average over all member subsets at each K; $K = 1$ is mean single-model loss. Build a scaling law $L(K) = L_\infty + AK^{-\alpha}$ to describe how loss changes as a function of ensemble member count. Use paired resampling of validation chunks and discuss sensitivity to fit range and functional form; five members may not identify the infinite-ensemble limit. Account for training and inference cost.

(iii) **Composition.** How much do ensembling and dropout compose? Train additional dropout members as needed, form ensembles, and see how the scaling law for ensembling changes. Dropout ensemble banks are not supplied.

Early experiments by Keskar et al. (2017) linked sharp minima to poorer generalization in large-batch training. Jiang et al. (2020), *Fantastic Generalization Measures and Where to Find Them*, tested generalization measures across a much broader collection of models and hyperparameters. We will investigate how accurately sharpness predicts generalization in our DNA setting by varying batch size and comparing SAM with AdamW, using the same supplied sharpness probe throughout.

Problem 3: Optimization

Use dense d8 on the same 30k-sequence pool. Vary the number of passes in (a); use 32 epochs in (b–d).

(a) **LR schedules.** Compare different learning rate schedules in the data-constrained regime. At a minimum, compare a linear learning rate schedule and a constant learning rate schedule at different training horizons and track the loss over training. Use the supplied histories; train additional schedules if helpful. Reason about how the learning rate schedule would affect the sharpness and how that might affect the loss.

(b) **Recall: batch size.** Smaller batches introduce more gradient noise, which intuitively favors flatter minima and lower sharpness. Analyze the supplied $B \in \{16, 64, 256, 1024, 4096\}$ runs at fixed data and epoch count. These start from the baseline $B = 64$ recipe and scale both LR and WD by $\sqrt{B/64}$, using microbatches as needed. Plot validation NLL against batch size. Which batch size is best?

(c) **SAM versus AdamW.** Though learning rate schedule and batch size can implicitly control sharpness, we can explicitly control sharpness through Sharpness-Aware Minimization (SAM). SAM operates by making a small, loss-increasing weight perturbation, then updating the original weights using the gradient at the perturbed weights. Train SAM and compare it to AdamW to see which optimizer reaches the lower loss.

(d) **Exploration.** Explore other optimizers that might improve data efficiency. Compare their validation loss with baseline AdamW and explain your findings.

Problem 4: Fine-tuning

One of the most effective ways to improve data efficiency is to first train on related data, then fine-tune on the target dataset. Hernandez, Kaplan et al. (2021), *Scaling Laws for Transfer*, study this benefit in terms of the effective amount of data transferred by pretraining. Here we compare four starting points:

Related DNA: pretraining on eukaryotic genic windows from OpenGenome2, excluding the target and validation chunks; **DCLM text:** pretraining on natural-language text; **nucleotide noise:** pretraining on IID uniform A/C/G/T, with BOS at the start of each sequence; and **scratch:** random initialization.

We release the three pretrained checkpoints with the assignment; the original pretraining datasets are not supplied. Each checkpoint was trained from scratch with the same d8 transformer body and next-token objective for approximately 10B tokens in 1,024-token sequences. Related-DNA pretraining repeats its approximately 2.57M eligible chunks about 3.8 times. DCLM uses a text tokenizer, so we transfer its transformer body and randomly initialize the DNA embedding and output head; the two nucleotide checkpoints retain their DNA embedding and head. Pretraining is already complete; your experiments start from these checkpoints or from scratch. Fine-tuning runs are not supplied. In (a) and (b), use full fine-tuning, updating *all* parameters.

(a) **Full fine-tuning.** On the 30k-sequence target pool, compare all four initializations after 1, 4, and 16 passes over the fine-tuning data. Tune the fine-tuning LR for each initialization and pass count. Which starting point achieves the lowest validation loss, and how does its advantage change with additional passes?

(b) **Scaling and effective samples.** Use fine-tuning pools of $n \in \{1\text{k}, 3\text{k}, 10\text{k}, 30\text{k}\}$ sequences, training for 16 passes over each pool. Build a scaling law for validation loss as a function of fine-tuning sample count, with the four initializations on one plot. How do their scaling laws compare? Can you map the benefit of pretraining to an effective sample count on the target task?

(c) **Adapting related DNA.** Focus on the related-DNA initialization. Investigate tuning weight decay, regularizing weights toward their pretrained initialization instead of zero, freezing only the output head, and freezing the transformer body (updating only the embedding and output head). Compare these variants with your full-fine-tuning baseline. Which adaptations help?

(d) **Exploration.** Can strategies from Problems 1–3 or previous assignments also improve data efficiency when fine-tuning? Investigate a strategy of your choice and compare it with your full-fine-tuning baseline.