Generalization Lecture 2

Compression

Last week

From many experiments over history, we have developed a few principles for how we can extract value from limited data

1. **Regularization (statistics)**
2. Sharpness (optimization)
3. Description length (architecture)
4. **Conditioning on less related data (fine-tuning, in-context learning)**
5. Procuring more data (augmentation, filtering)

This week

From many experiments over history, we have developed a few principles for how we can extract value from limited data

1. Regularization (statistics)
2. **Sharpness (optimization)**
3. **Description length (architecture)**
4. Conditioning on less related data (fine-tuning, in-context learning)
5. **Procuring more data (augmentation, filtering)**

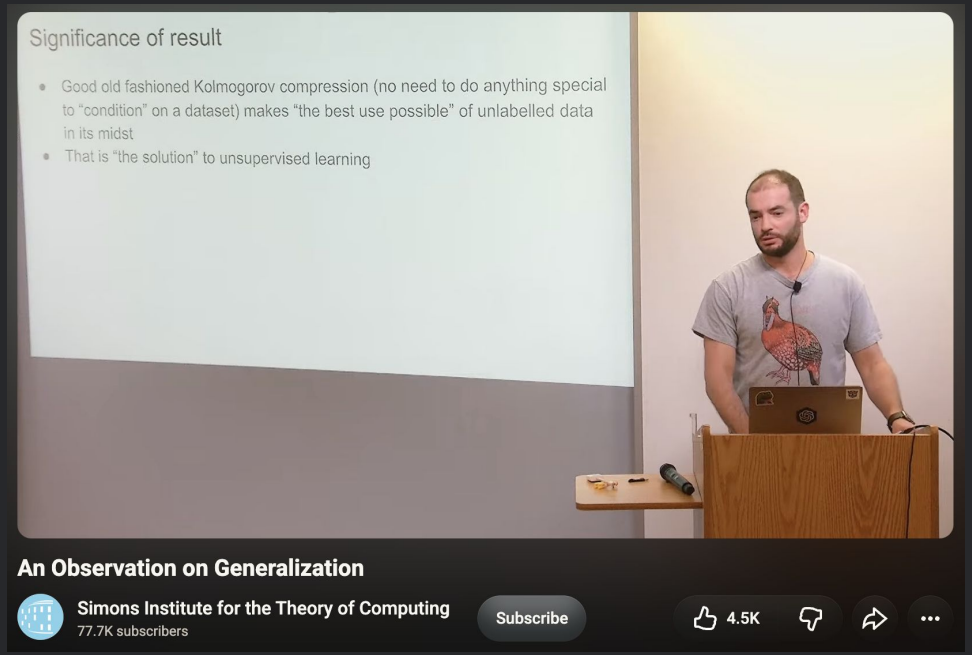
Outline

- Algorithmic Information Theory
- Application 1: Description length in architecture
- Application 2: Sharpness in optimization
- Application 3: Exploiting inefficiencies via data (filtering, augmentation)

Compression concepts

Compression

Information theory and Kolmogorov complexity serve as a strong source of inspiration for how to design methods that generalize (watch this talk for more)



Compression

The math powering these intuitions is actually quite simple. We will go over a crash course in the following topics

1. Information theory
2. Complexity theory
3. What predictors should we use?
    - Solomonoff Induction
    - Minimum Description Length principle

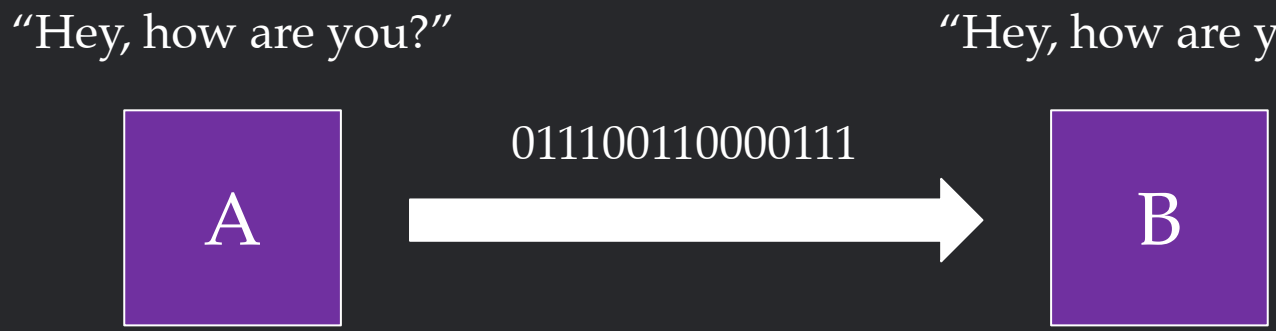
Warning: Our treatment will not be rigorous (ignore semimeasures, constants, etc)

1. Information Theory

Information Theory Primer

Information theory was developed by Claude Shannon in 1948 when he was working at Bell Labs

Original motivation: How many bits does it take to communicate symbols across as a channel?



Information Theory Primer

Simple example: Suppose we have distribution P with four symbols (apple, banana, cherry, durian) each with probability 1/2, 1/4, 1/8, 1/8.

What code f can we develop to communicate these symbols with lowest expected length $\mathbb{E}_{x\sim P}[|f(x)|]$?



Information Theory Primer

One possible code: assign length 2 string to each symbol. This gets expected code length 2. Can you come up with a shorter code?

Symbol	Probability	Codeword	Length
A	1/2	00	2
B	1/4	01	2
C	1/8	10	2
D	1/8	11	2

Information Theory Primer

A different code which gives shorter codewords to higher likelihood symbols gets 1.75 bits per symbol instead.

Symbol	Probability	Codeword	Length
A	1/2	0	1
B	1/4	10	2
C	1/8	110	3
D	1/8	111	3

Information Theory Primer

A different code which gives shorter codewords to higher likelihood symbols gets 1.75 bits per symbol instead. Safe to communicate because it is **prefix-free**

Symbol	Probability	Codeword	Length
A	1/2	0	1
B	1/4	10	2
C	1/8	110	3
D	1/8	111	3

Information Theory Primer

General strategy: Given symbols with arbitrary probabilities, assign lengths proportional to negative log likelihood of symbol

Symbol	Probability	Length
A	p_1	$-\log \lceil p_1 \rceil$
B	p_2	$-\log \lceil p_2 \rceil$
C	p_3	$-\log \lceil p_3 \rceil$
D	p_4	$-\log \lceil p_4 \rceil$

Information Theory Primer

Inverse: When one specifies a prefix-free code, it implicitly maps back to a distribution

Symbol	Probability	Length
A	$\propto \exp(-s_1)$	s_1
B	$\propto \exp(-s_2)$	s_2
C	$\propto \exp(-s_3)$	s_3
D	$\propto \exp(-s_4)$	s_4

Shannon source coding theorem

Shannon source coding theorem: Given a distribution with probabilities p_i, the optimal compressor for f has expected code length at least $\mathbb{E}_{x\sim P}[-\log P(x)]$. Furthermore, there exists f that achieves this compression up to 1 bit.

$$\mathbb{E}_{x\sim P}[-\log P(x)] \leq \mathbb{E}_{x\sim P}[|f(x)|] \leq \mathbb{E}_{x\sim P}[-\log P(x)] + 1$$

Shannon source coding theorem

Shannon source coding theorem: Given a distribution with probabilities p_i, the optimal compressor for f has expected code length at least $\mathbb{E}_{x\sim P}[-\log P(x)]$. Furthermore, there exists f that achieves this compression up to 1 bit.

$$\mathbb{E}_{x\sim P}[-\log P(x)] \leq \mathbb{E}_{x\sim P}[|f(x)|] \leq \mathbb{E}_{x\sim P}[-\log P(x)] + 1$$

entropy H(P)

Shannon source coding theorem

Shannon source coding theorem: Given a distribution with probabilities p_i, the optimal compressor for f has expected code length at least H(P). Furthermore, there exists f that achieves this compression up to 1 bit.

$$H(P) \leq \mathbb{E}_{x \sim P}[|f(x)|] \leq H(P) + 1$$

Entropy

Entropy H(P) connects two aspects

- The inherent “randomness” in a distribution (a coin flip has maximal entropy)
- The difficulty of communicating/compressing samples from a distribution

Measuring error

How do we measure the error of a suboptimal code?

As described earlier, specifying a suboptimal code f with maps the code lengths s_i to an implicit distribution $q_i \propto \exp(-s_i)$. Therefore, a suboptimal code f gets expected description length

$$\mathbb{E}_{x\sim P}[-\log Q(x)]$$

Measuring error

How do we measure the error of a suboptimal code?

As described earlier, specifying a suboptimal code f with maps the code lengths s_i to an implicit distribution $q_i \propto \exp(-s_i)$. Therefore, a suboptimal code f gets expected description length

$$\mathbb{E}_{x \sim P}[-\log Q(x)]$$

cross-entropy H(P, Q)

Connecting entropy and cross-entropy

Shannon's source coding theorem is the same as "cross-entropy > entropy"

We can connect both quantities more explicitly

$$H(P, Q) - H(P)$$

$$= \mathbb{E}_{x \sim P}[-\log Q(x)] - \mathbb{E}_{x \sim P}[-\log P(x)]$$

$$= \mathbb{E}_{x \sim P}[\log P(x) - \log Q(x)]$$

$$= KL(P \,||\, Q)$$

Therefore, minimizing cross-entropy (our standard loss) minimizes KL divergence

Summary

The difficulty of compressing a distribution to communicate samples from it is equivalent to the entropy

Minimizing cross-entropy loss is the same as minimizing the number of bits used in compressing the distribution. It is also the same minimizing the KL divergence

2. Complexity Theory

Complexity Theory Primer

So far, we've discussed how to optimally compress samples from a distribution

Now, we will take a different algorithmic perspective on compression: the complexity of a sample/distribution is the length of the program necessary to implement it

Complexity Theory Primer

Setup: Define a Turing machine T as a function that accepts valid programs h as bitstrings and outputs bitstrings x if the program halts.

- Prefix-free Turing machines have a prefix-free domain (no input program is a prefix of another valid program)
- Universal Turing machines U can simulate other Turing machines T given the description T as input



Sample level complexity

Definition: The Kolmogorov complexity of a string x is the length of the shorter program h that produces x under prefix-free, universal Turing machine U

$$K(x) = \min_h \{|h| : U(h) = x\}$$

Sample level complexity

Examples:

000000……000000 has low Kolmogorov complexity: a short for loop can emit arbitrary 0’s

010101……010101 has low complexity for similar reasons

010110……110101 (randomly generated) has high complexity w.h.p. (provably)

Sample level complexity

$$K(x) = \min_h \{|h| : U(h) = x\}$$

One concern is that this definition is specific to the choice of Turing machine U

However, if there was a different Turing machine U’ that got short description length $K_{U'}(x)$, U can encode this with $K_U(x) \leq K_{U'}(x) + |U'|$ by universality.

Therefore, Kolmogorov complexity is universal up to constants

Distributional complexity

We can also measure complexity of distributions rather than samples

Definition: The Kolmogorov complexity of a distribution P is the length of the shortest program that can implement likelihood evaluation (i.e. given x, output P(x))

Algorithmic Coding Theorem

We will now connect our information-theoretic notions of compression with our complexity-theoretic notions

$$H(P) \leq \mathbb{E}_{x \sim P}[K(x)] \leq H(P) + K(P) + O(1)$$

Intuitively: compressing samples x by assigning them their shortest program achieves near optimal compression

Algorithmic Coding Theorem

Lower bound: Simple consequence of Shannon's source coding theorem, no code can beat entropy for compression

$$H(P) \leq \mathbb{E}_{x \sim P}[\,|f(x)|\,]$$

Algorithmic Coding Theorem

Upper bound: We need to provide one program h that achieves compression H(P) + K(P) + O(1)

Lemma: For a sample x and for any distribution P, we have

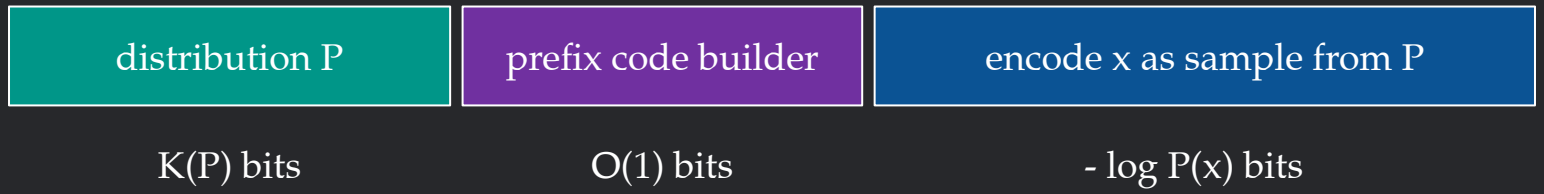
$$K(x) \leq - \log P(x) + K(P) + O(1)$$

Algorithmic Coding Theorem

$$K(x) \leq -\log P(x) + K(P) + O(1)$$

Proof: We need to show one program h that achieves this bound

1. encode the distribution P
2. construct a prefix code
3. encode x as a sample from P (using Shannon encoding)



Algorithmic Coding Theorem

By taking the expectation over P, we get the original theorem statement

$$H(P) \leq \mathbb{E}_{x \sim P}[K(x)] \leq H(P) + K(P) + O(1)$$

K(x) achieves optimal compression

K(x) as a compressor maps to $\propto 2^{-K(x)}$, the **universal distribution**

Universal distribution as a mixture

There are multiple representations of $2^{-K(x)}$ equivalent up to constants

One important representation is as a mixture of all computable hypotheses P_i weighted by distributional complexity

$$2^{-K(x)} \leq \Sigma_i 2^{-K(Pi)} P_i(x) + O(1)$$

3. Induction

Induction

So far, we've discussed how to compress samples from a distribution

However, what we're interested is in **induction**, or "given observations, how can we infer what process generated them?"

Simple formulation

Suppose there was an unknown program h producing a stream of bits

00101110101101010111110001 _

Given sequence S, how can we predict the next bit? There are many classical principles surrounding how we should proceed

Occam's razor

"Entities should not be multiplied beyond necessity"

or

"Among the theories that are consistent with the observed phenomena, one should select the simplest theory"

For the bitstrings, what is the simplest theory? What does simple even mean?

Bayes' Rule

"The probability of a hypothesis H being true is proportional to the learner's initial belief in H (the prior probability) multiplies by the conditional probability of data D given H"

This is frequently used in Bayes' rule

$$P(H_i \mid D) = \frac{P(D \mid H_i)\, P(H_i)}{\Sigma_j P(D \mid H_j)\, P(H_j)}$$

Solomonoff Induction

Ray Solomonoff outlined an approach to induction using Kolmogorov complexity (which he concurrently defined) [Solomonoff 1964]

Algorithm: Suppose you were given a sequence of bits S produced by an unknown distribution P with program h. Solomonoff induction predicts the next bit using the Solomonoff prior

$$\Sigma_i \, 2^{-K(Pi)} \, P_i(x)$$

Solomonoff Induction

Using this prediction rule is equivalent to performing Bayesian inference over all possible programs weighted by a prior on simplicity. In this sense, Solomonoff Induction combines Occam's razor with Bayes rule with a universal notion of complexity

$$P(P_i \mid S) = \frac{P_i(S)\, 2^{-K(Hi)}}{\Sigma_j P_j(S)\, 2^{-K(Hj)}}$$

$$P(0 \mid S) = \Sigma_i P_i(0 \mid S)\, P(P_i \mid S)$$

Solomonoff Induction error bound

Suppose we had to make n predictions for bits drawn from distribution P. How much error do we incur?

$$KL(P, \Sigma_i 2^{-K(Pi)} P_i(x))$$

$$= \mathbb{E}_{x\sim P}[- \log (\Sigma_i 2^{-K(Pi)} P_i(x))]$$

$$\leq \mathbb{E}_{x\sim P}[- \log (2^{-K(P)} P(x))]$$

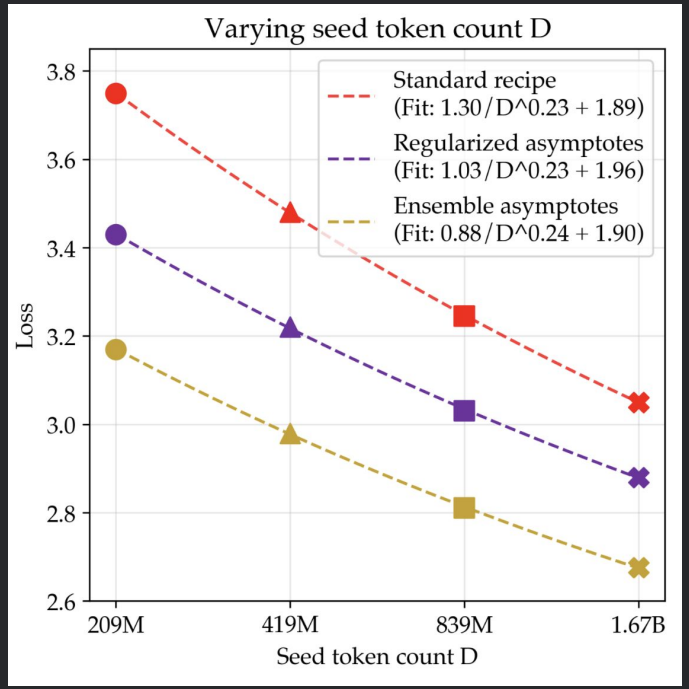
$$= K(P) + H(P) \qquad \text{(up to constants)}$$

Solomonoff Induction error bound

$$KL(P, \Sigma_i 2^{-K(Pi)} P_i(x)) \leq K(P) + H(P)$$

Though entropy likely scales in sequence length, K(P) does not, implying constant horizon-dependent error

For n samples, loss decreases at a rate K(P) / n, faster than our last measured rate of $n^{0.23}$



Minimum Description Length

Unfortunately, Solomonoff Induction is incomputable and does NOT yield feasible algorithms. However, we can reformulate the algorithm as an objective

Instead of optimizing for loss of Q_h, also minimize description length

$$MDL(h) = \min_h \{H(P, Q_h) + K(Q_h)\}$$

For example, Solomonoff Induction shows that $MDL(h) \leq H(P) + K(P)$

MDL as regularization

$$\mathrm{MDL}(h) = H(P, Q_h) + K(Q_h)$$

Our formulation of MDL is equivalent to last week's presentation on regularization: Instead of simply minimizing train loss, also explicitly minimize the description length.

Different definition of description length give different methods

MDL Application: Weight Decay

One notion of description length is parameter norm. In fact, one of the original interpretations of weight decay [Van Camp and Hinton, 1993]

$$L(\theta) = H(P, Q_\theta) + \theta^2 / 2$$

> This is just the standard "weight-decay" method. The fact that weight-decay improves generalization (Hinton, 1987) can therefore be viewed as a vindication of this crude MDL approach in which the standard deviations of the gaussians used for coding the data misfits and the weights are both fixed in advance.[2]

We will now apply MDL to architecture and optimization

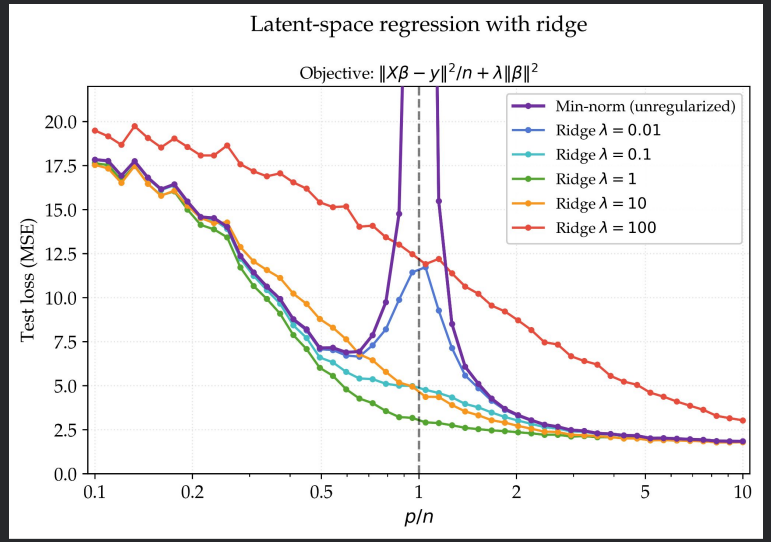
Architecture: Description Length

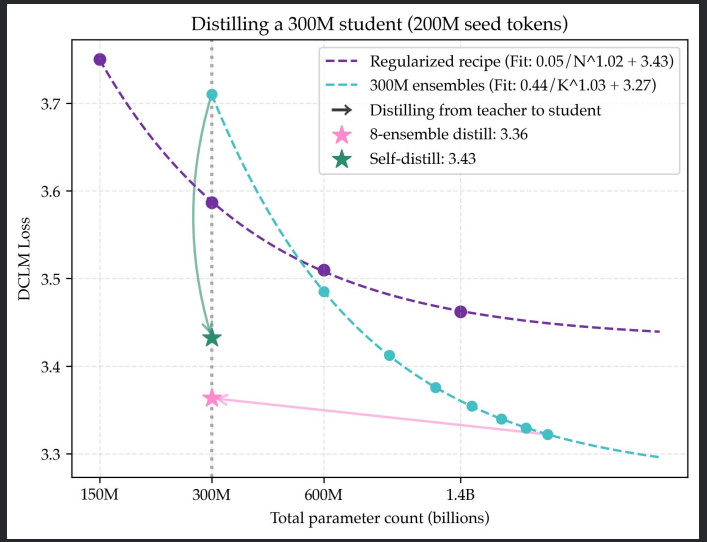
Motivating description length

Suppose we take Minimum Description Length principle seriously. How do we minimize description length of our architectures?

Motivating description length

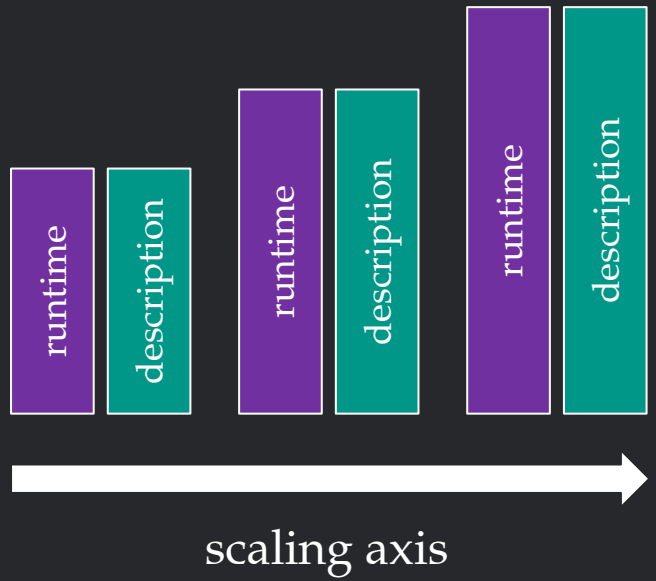
Unfortunately, parameter count is a poor notion of description length. As we saw last lecture, overparameterized models can generalize. Furthermore, large models can be distilled into small models, suggesting that capacity isn't saturated





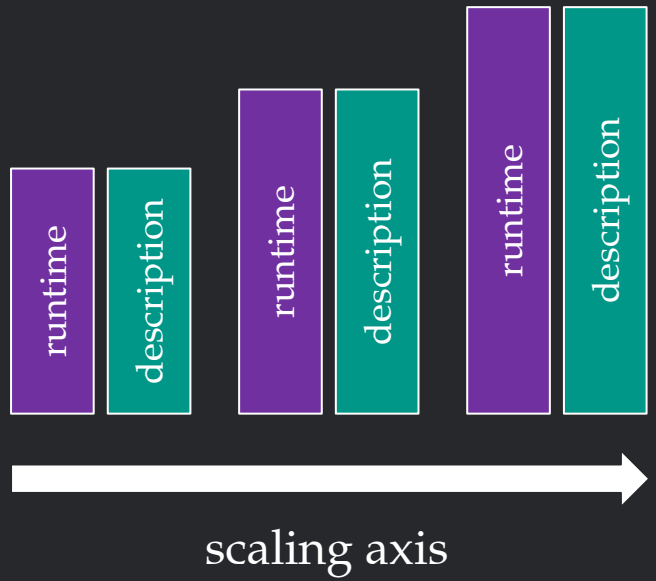
Motivating description length

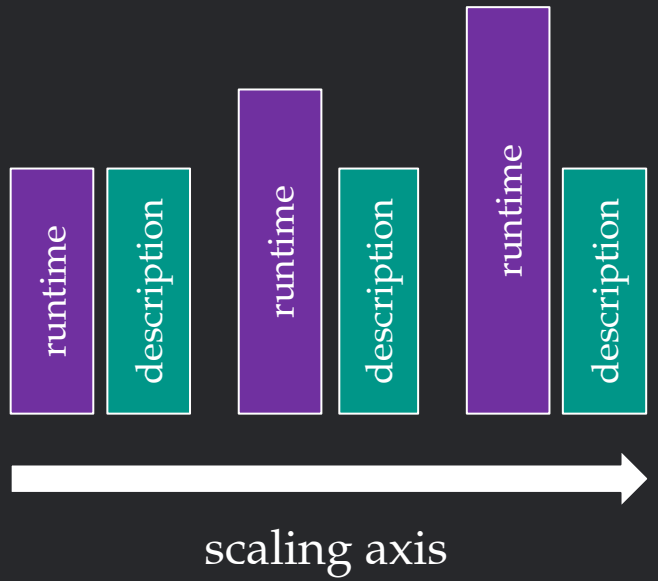
Important intuition (credit to Minhak): Specifying a parameter count corresponds to a description length *and* a program runtime.



Motivating description length

Instead of minimizing description length, we should maximize runtime subject to description length.





Motivating description length

Architectures we will consider

1. Depth: Looping
2. Width: MoE
3. Sequence: Tokenization and CoT

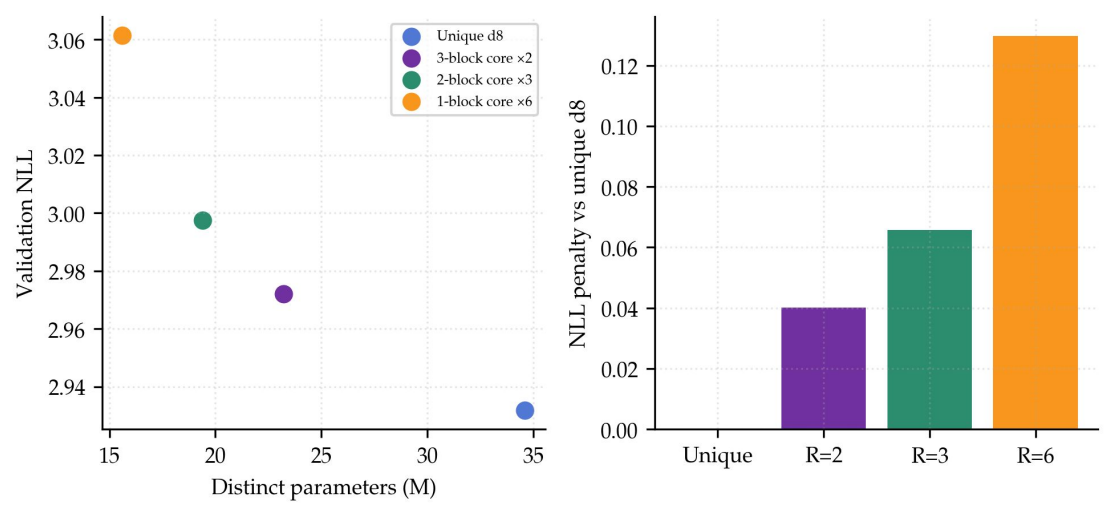
Looping

Conceptually, looping allows us to express models with high program runtime while restricting the description length



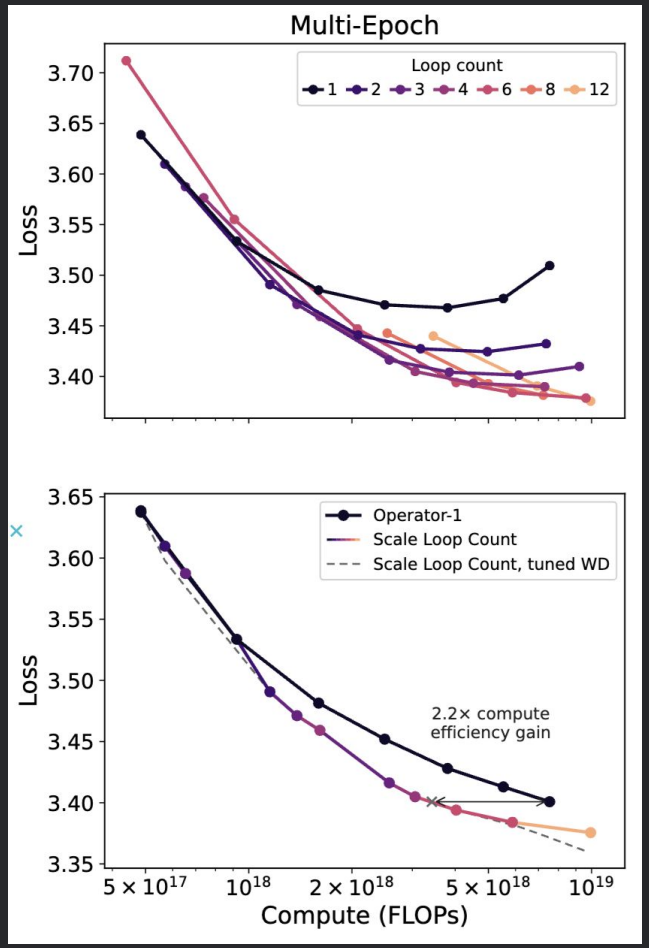
Looping

In Capacity, you identified that choosing looped layers over unique layers always hurt performance in our 1 pass regime. Consistent with prior literature



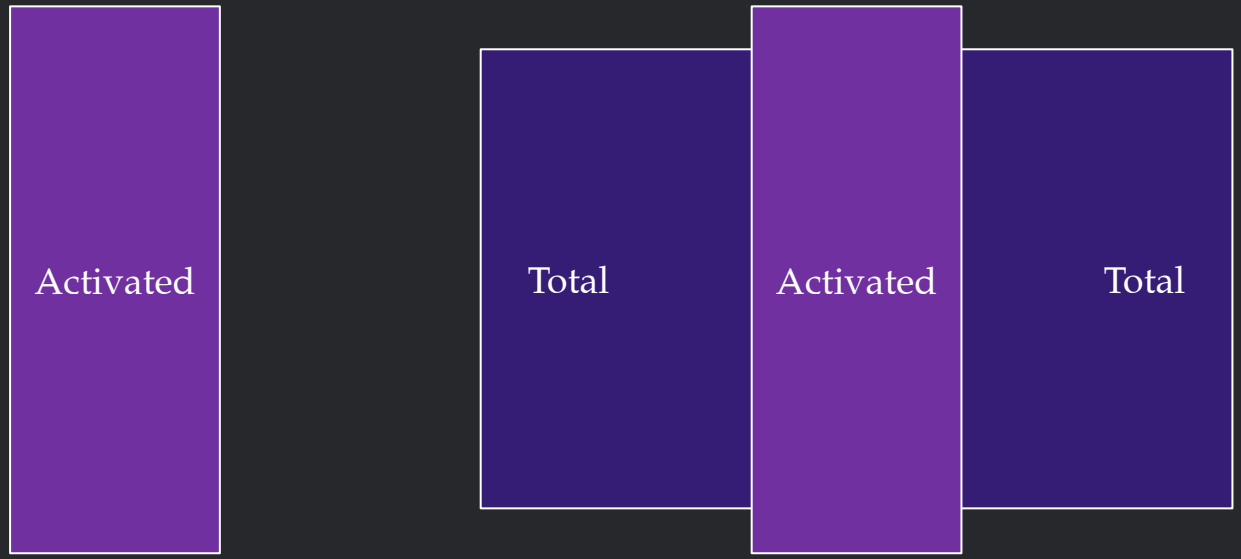
Looping

Interestingly, in the data-efficient regime, the conclusion changes and looping can be more effective than scaling unique parameters [Chen et al, 2026]



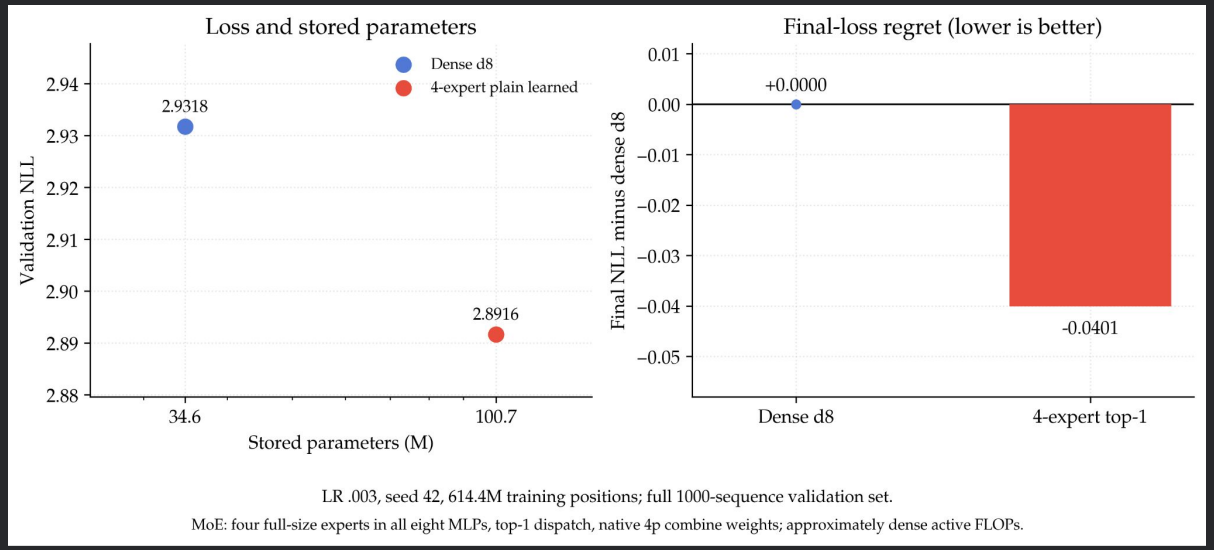
Mixture of Experts

On the other hand, Mixture of Experts follows the opposite idea: For a fixed program runtime, increase the description length.



Mixture of Experts

In Capacity, you identified that using multiple experts while only activating one improved performance in our 1 pass regime. Consistent with prior literature



Mixture of Experts

On the other hand, (limited) evidence that MoE's tolerate less epochs and generalize worse than dense models [Jha et al, 2026].

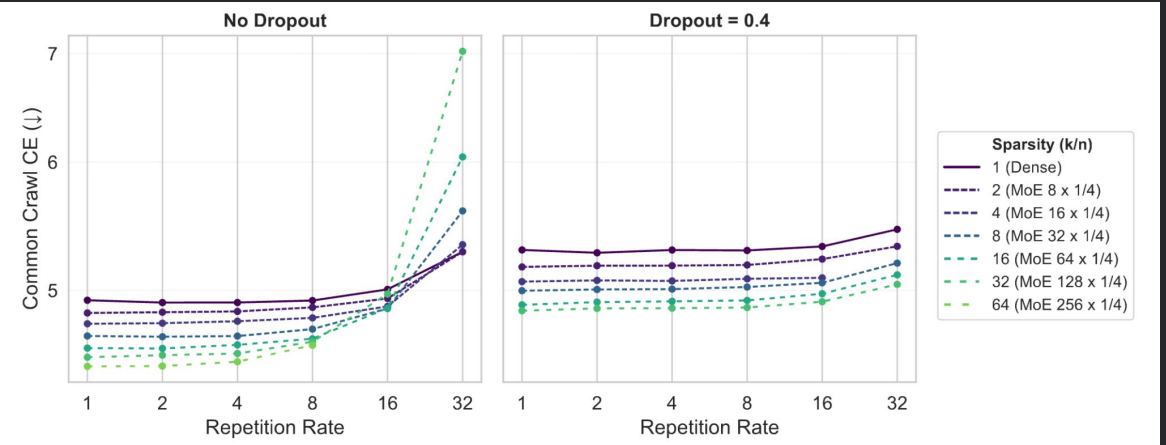


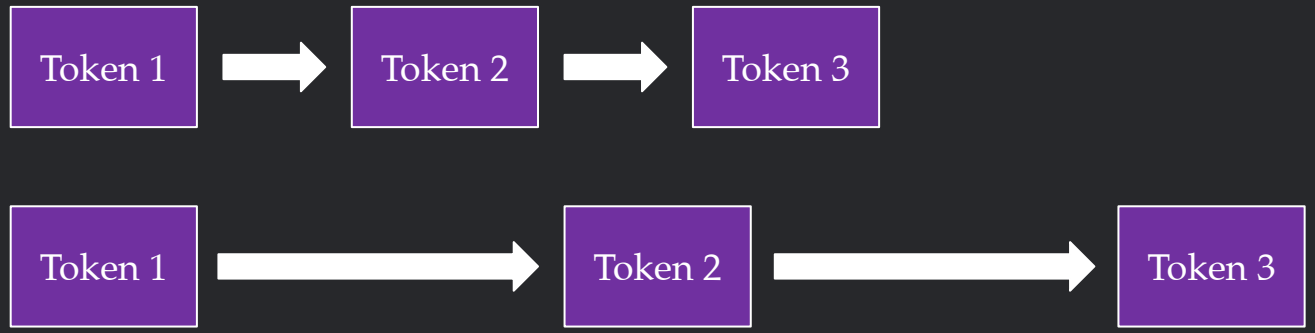
Figure 1: **Sparser MoEs degrade more with data repetition (left), but regularization may ameliorate the effect (right).** Matching compute via total data and active parameters, we find that higher repetition of fewer unique tokens results in performance degradation. This phenomenon increases in severity with MoE sparsity, and thus depends on *total* rather than active parameters (§3.1). Patterns are consistent across data mixes, total tokens, and other configuration details (§3.1-3.4). However, sufficiently strong regularization may substantially decrease overfitting (§4).

Assignment

In the assignment, you will revisit looping and MoE's with a fresh eye for data efficiency. How do conclusions for MoE and looping change?

Sequence dimension

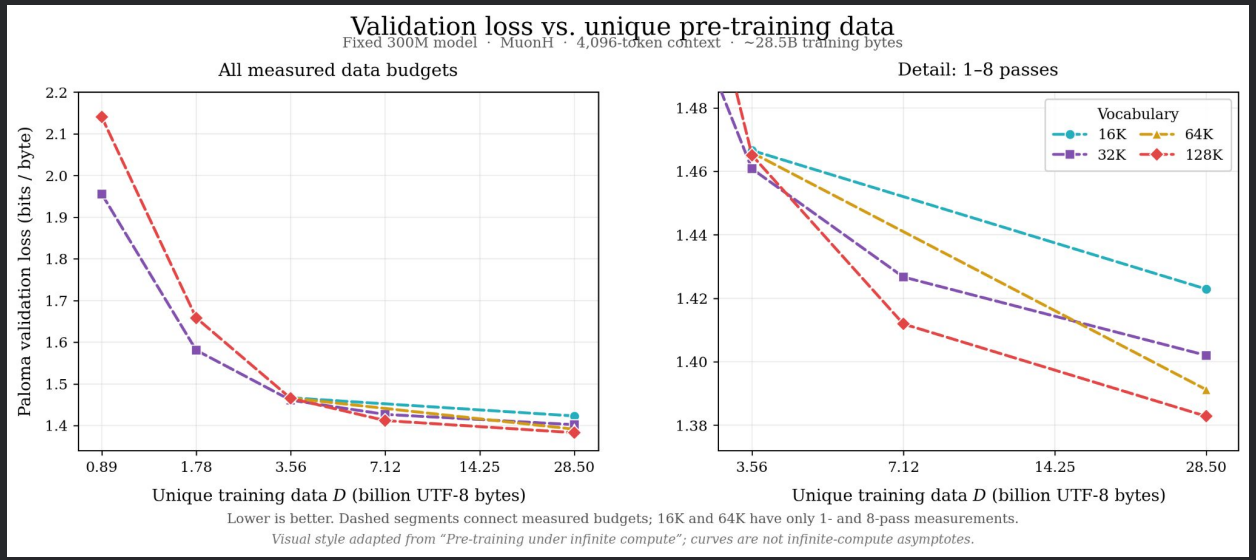
Instead of increasing program runtime in the depth and width dimensions, we can also increase it along the sequence dimension.



Method #1: Tokenization

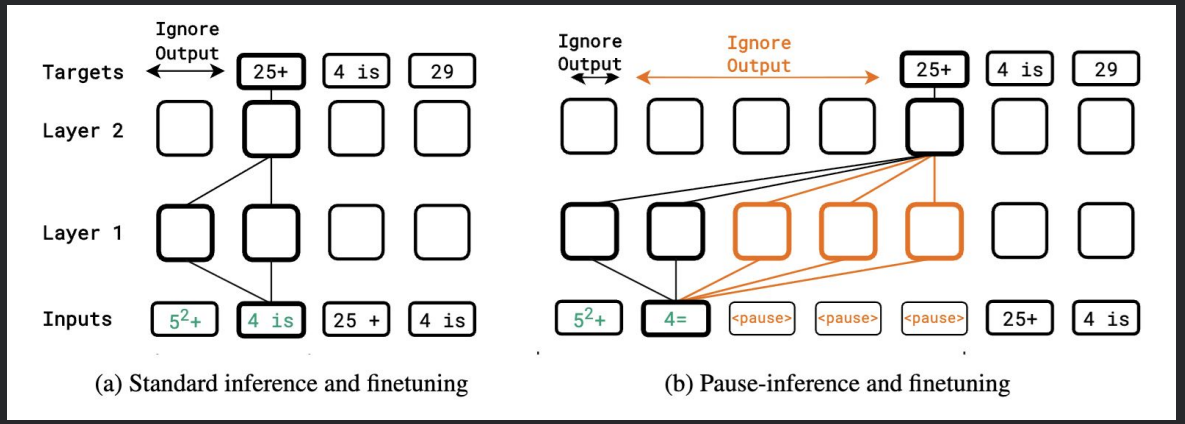
Prediction: Use smaller vocab size with less data, supported by Chai, 2026

Contrasts literature in compute-efficient tokenization which suggests growing vocab size with compute [Tao et al, 2024]



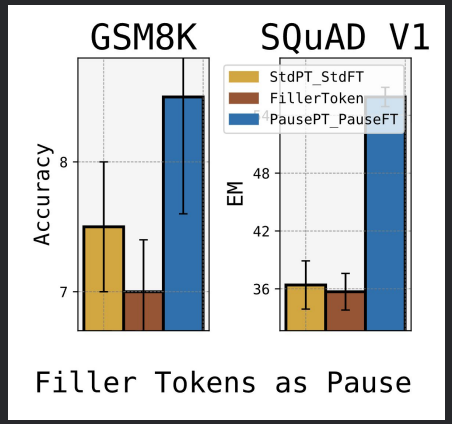
Method #2: Pause tokens

Can increase program runtime by insert placeholder pause tokens. Qualitatively similar to looping, initial experiments in Goyal et al, 2024. Open research problem for pre-training!



(a) Standard inference and finetuning

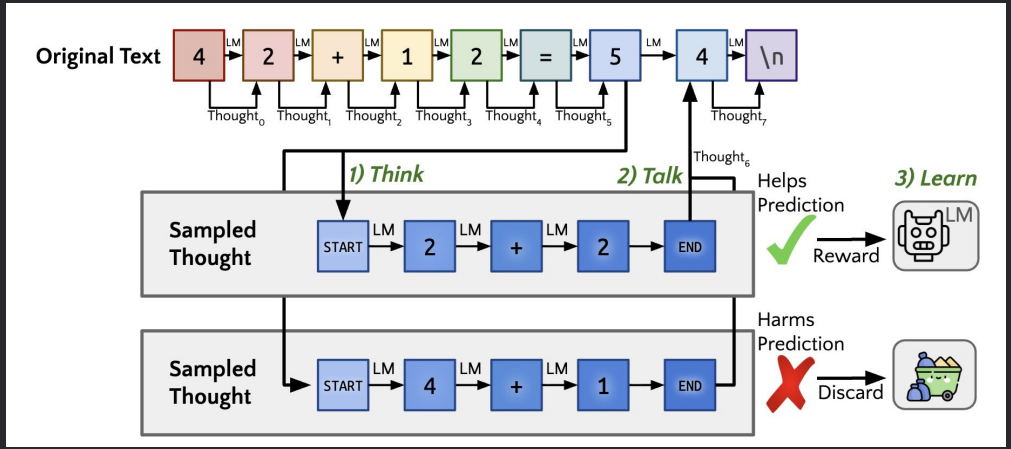
(b) Pause-inference and finetuning



Method #3: Chain of Thought

More sophisticated variant: allow the model to decide which tokens to use. Similar to Chain of Thought (wildly successful for RL)

Quiet-STaR [Zelikman et al, 2024] offers initial explorations along this direction. Open research problem for pre-training!



Optimization: Sharpness

Motivating sharpness

There is a folklore intuition that solutions with flatter minima correspond to higher generalization [Keskar et al, 2016]

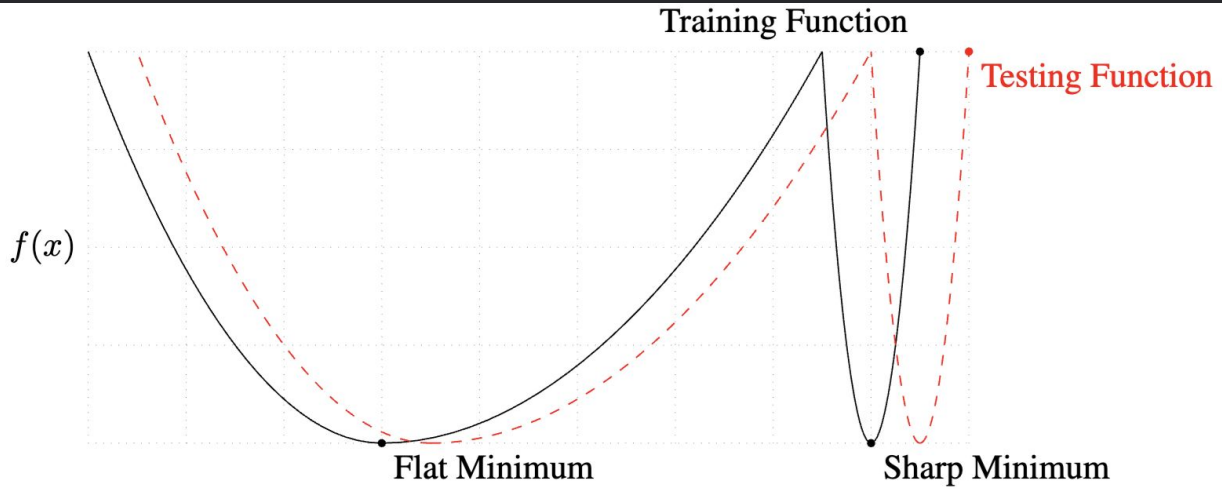


Figure 1: A Conceptual Sketch of Flat and Sharp Minima. The Y-axis indicates value of the loss function and the X-axis the variables (parameters)

Motivating sharpness

As discussed during optimization, Jiang et al, 2019 studies how well various generalization measures correlate with performance. Most measures correlate poorly, with sharpness being a promising contender

$1/\alpha'$ sharpness mag	0.516
sum-of-fro/margin	0.381
pacbayes-orig	0.360
path-norm/margin	0.305
grad-noise-final	0.292
fro/spec	0.243
cross-entropy	0.147
# params	−0.15

Motivating sharpness - MDL

There are many arguments for why lower sharpness is good for generalization. We will start with Hochreiter and Schmidhuber, 1997 which argues that flatter minima are easier to compress

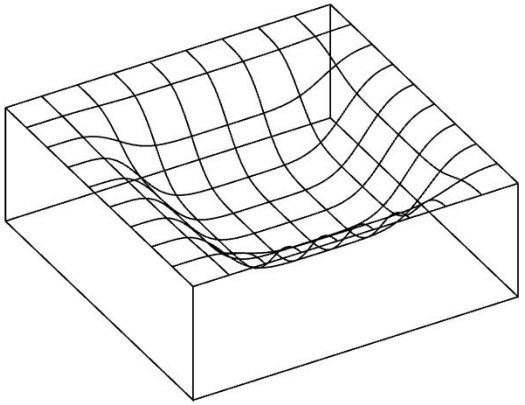


Figure 1: *Example of a "flat" minimum.*

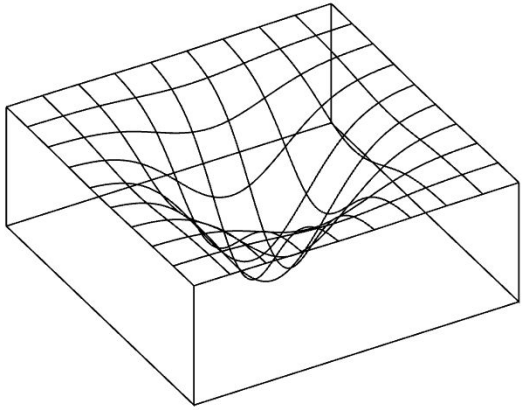
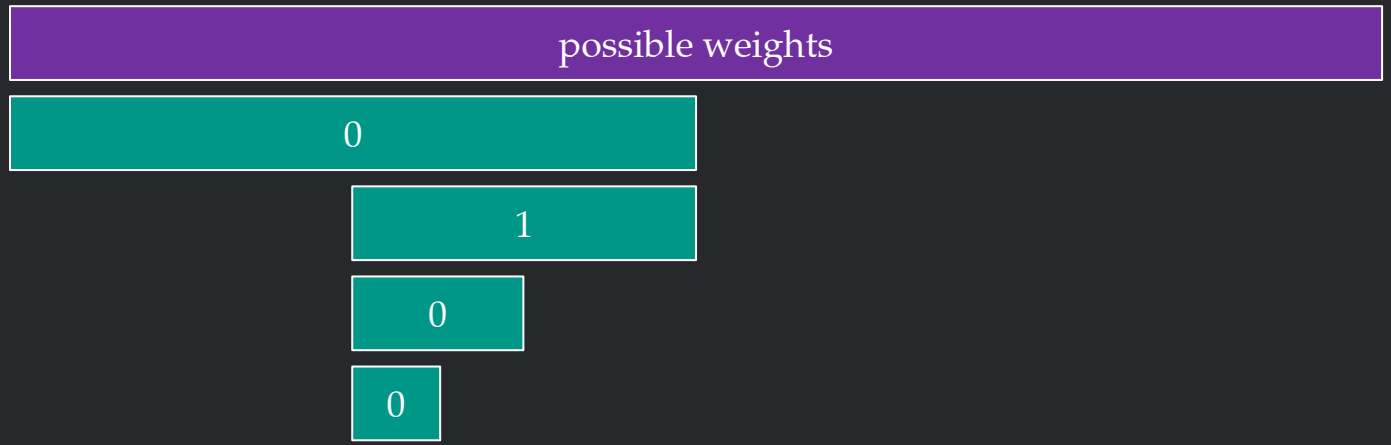


Figure 2: *Example of a "sharp" minimum.*

Motivating sharpness - MDL

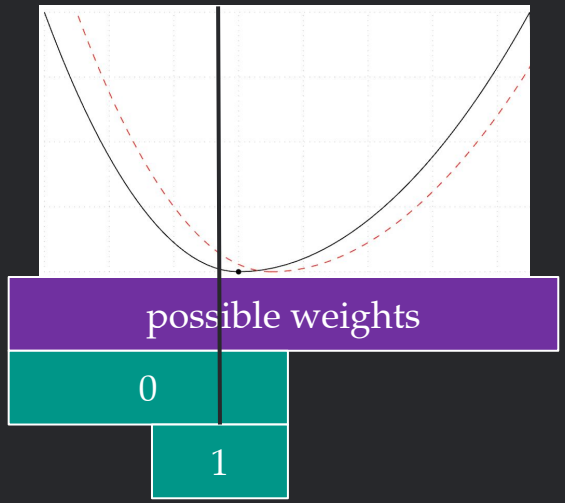
High-level argument: Flatter solutions have lower description length than sharper solutions

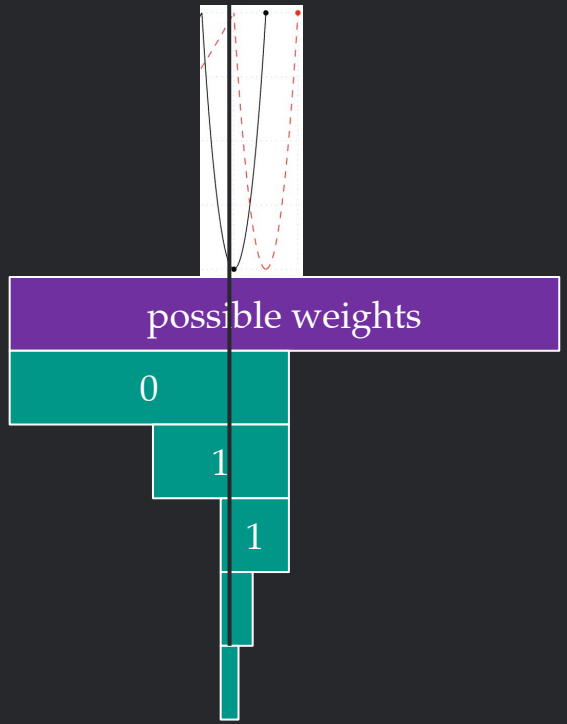
Suppose have to specify the exact weights. We can represent the value in each dimension by iteratively cutting up the hypothesis class



Motivating sharpness - MDL

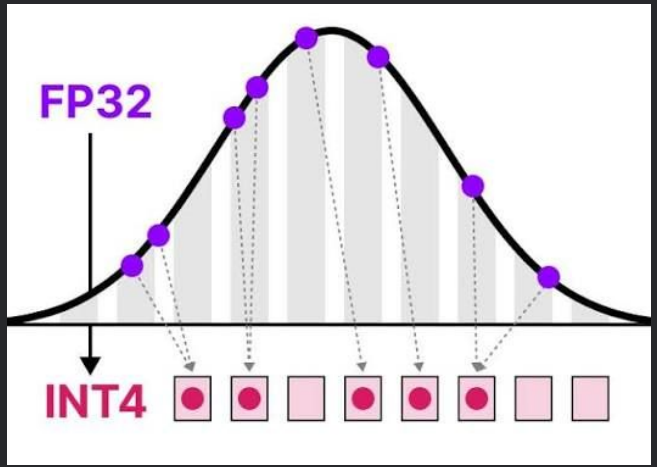
Flatter solutions use less bits to specify the same solution up to an error threshold





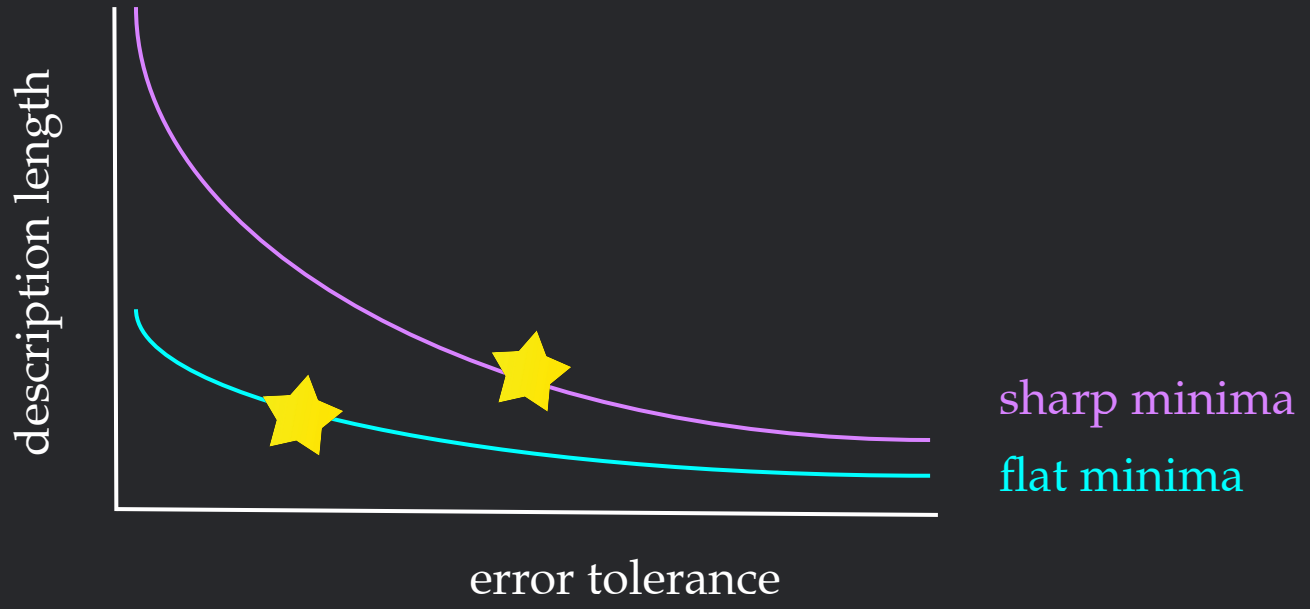
Motivating sharpness - MDL

Conceptual counterpart to quantization: functions intuitively have shorter description length if their loss is less sensitive to quantization to lower precision



Motivating sharpness - MDL

Since flatter solutions require less bits at every error threshold, they have lower minimum description length



Empirically reducing sharpness

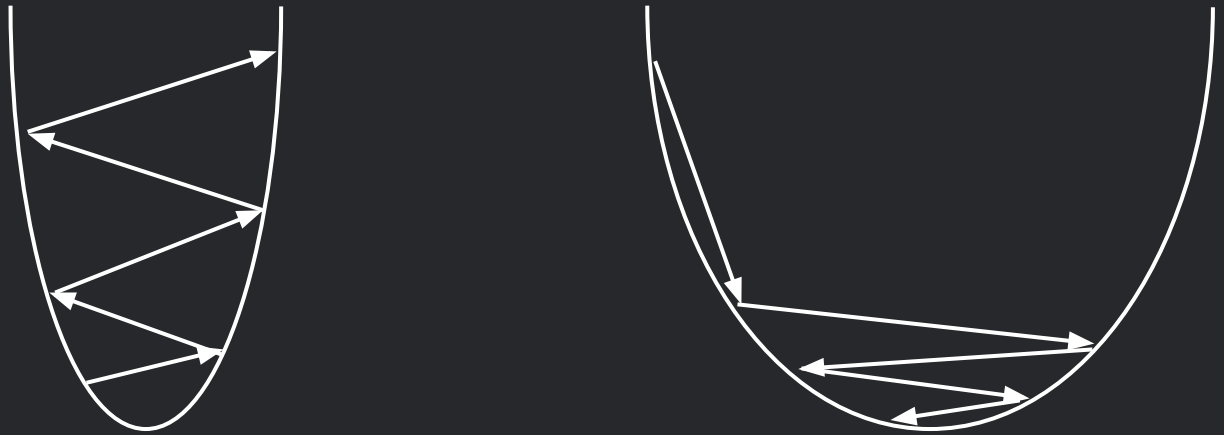
If we believe in the sharpness mental model, we can think about how we can control it via optimization

1. Low batch size
2. High learning rate (peak or schedule)
3. Sharpness Aware Minimization

Batch size and learning rate

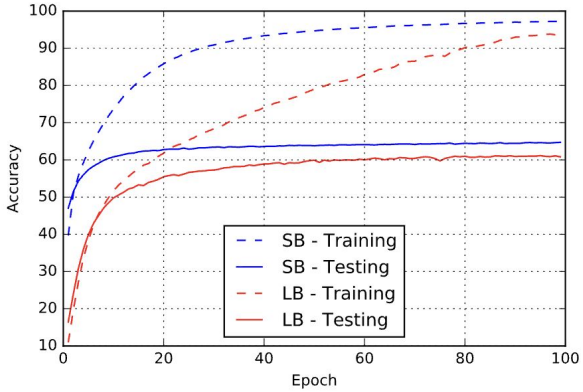
Recall from optimization that lower batch size and higher learning rates correspond to flatter solutions

Intuition: when training is more stochastic, gradient descent will oscillate out of sharp local minima and instead seek flat local minima

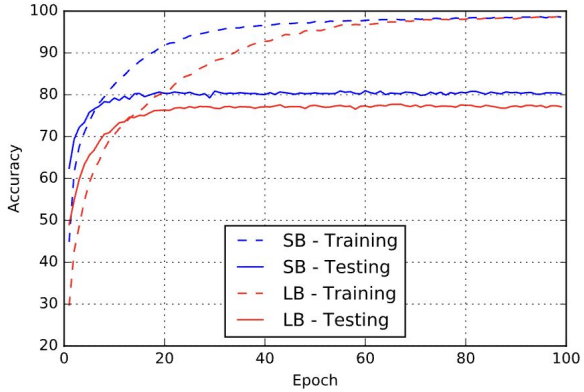


Batch size empirics

Well-known in earlier experiments that higher batch size did worse with limited data [LeCun et al, 1998]. Keskar et al, 2016 provide a cleaner analysis of this phenomenon



(a) Network F_2

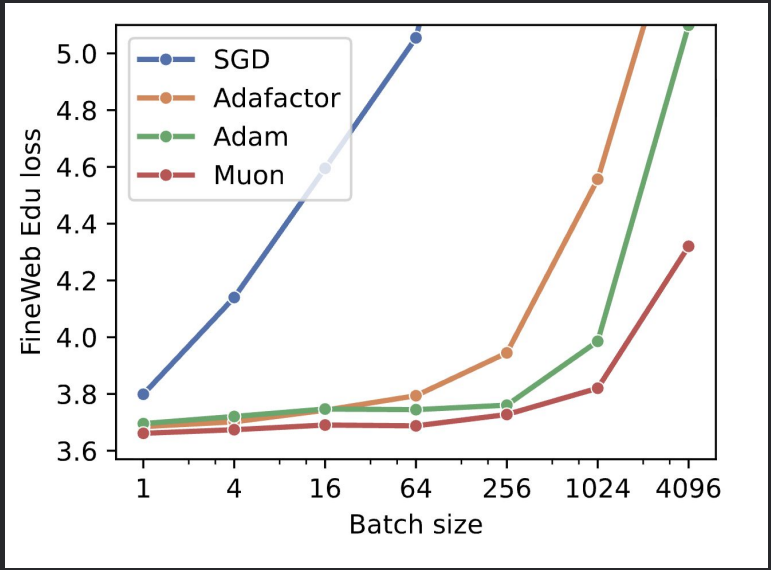


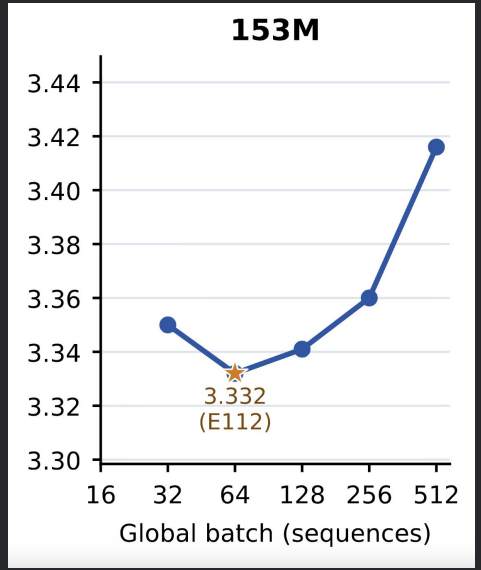
(b) Network C_1

Figure 2: Training and testing accuracy for SB and LB methods as a function of epochs.

Batch size empirics

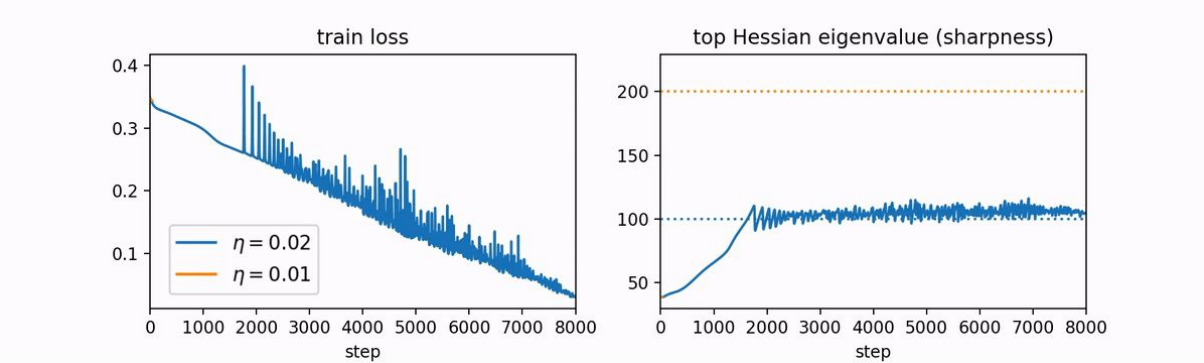
More recently, the same phenomenon has been shown for 1 pass training [Marek et al, 2025] and infinite compute pre-training [Min, 2026]. Important to tune β_2!





Learning rate empirics

We know that other ways of inducing high sharpness involve higher learning rates (either higher peak learning rate or learning rate schedules that don't decay).



In the assignment, you will explore whether such interventions improve generalization

Sharpness Aware Minimization

Recall from Sharpness lecture that explicitly reducing sharpness via SAM improves generalization (especially for label noise)

Note that it might be minimizing a different notion of sharpness (average-case instead of worst-case perturbation) [Wen et al, 2022]

Model	Augmentation	CIFAR-10 SAM	CIFAR-10 SGD	CIFAR-100 SAM	CIFAR-100 SGD
WRN-28-10 (200 epochs)	Basic	**2.7**$_{\pm 0.1}$	3.5$_{\pm 0.1}$	**16.5**$_{\pm 0.2}$	18.8$_{\pm 0.2}$
WRN-28-10 (200 epochs)	Cutout	**2.3**$_{\pm 0.1}$	2.6$_{\pm 0.1}$	**14.9**$_{\pm 0.2}$	16.9$_{\pm 0.1}$
WRN-28-10 (200 epochs)	AA	**2.1**$_{\pm <0.1}$	2.3$_{\pm 0.1}$	**13.6**$_{\pm 0.2}$	15.8$_{\pm 0.2}$

Method	Noise rate (%) 20	40	60	80
Sanchez et al. (2019)	94.0	92.8	90.3	74.1
Zhang & Sabuncu (2018)	89.7	87.6	82.7	67.9
Lee et al. (2019)	87.1	81.8	75.4	-
Chen et al. (2019)	89.7	-	-	52.3
Huang et al. (2019)	92.6	90.3	43.4	-
MentorNet (2017)	92.0	91.2	74.2	60.0
Mixup (2017)	94.0	91.5	86.8	76.9
MentorMix (2019)	**95.6**	**94.2**	91.3	**81.0**
SGD	84.8	68.8	48.2	26.2
Mixup	93.0	90.0	83.8	70.2
Bootstrap + Mixup	93.3	92.0	87.6	72.0
SAM	95.1	93.4	90.5	77.9
Bootstrap + SAM	95.4	**94.2**	**91.8**	79.9

Table 4: Test accuracy on the clean test set for models trained on CIFAR-10 with noisy labels. Lower block is our implementation, upper block gives scores from the literature, per Jiang et al. (2019).

Summary

We can reason about sharpness resulting in lower description length solutions

Empirically, reducing sharpness (implicitly via batch size or explicitly via SAM) may improve generalization

Note: not every intervention that decreases sharpness results in improved generalization. For example, Dinh et al, 2017 show parameterization dependence and Andriuschchenko et al, 2023 find negative correlations

Data

Overview

How should we process our data when we are limited? Intuitively, we want more data points

1. Filter less
2. Generate more

Filtering

In standard pre-training pipeline, due to costs, it is typical to filter data to keep only high quality data. Does this make sense when constrained by data?

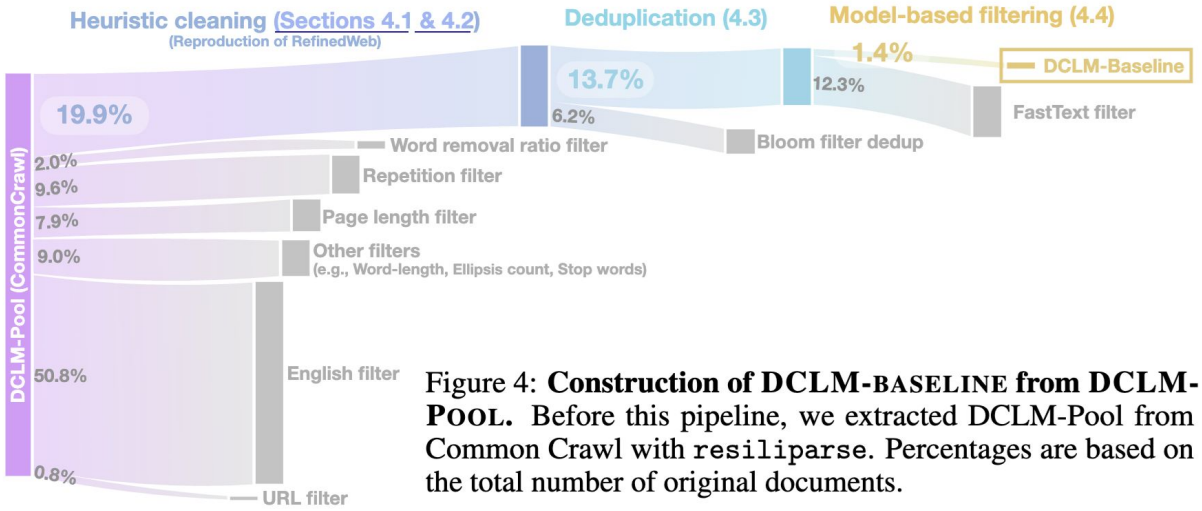


Figure 4: **Construction of DCLM-BASELINE from DCLM-POOL.** Before this pipeline, we extracted DCLM-Pool from Common Crawl with `resiliparse`. Percentages are based on the total number of original documents.

Filtering

When constrained by data, better to compress everything. Goyal et al, 2025 shows this for multimodal CLIP models, Mohri et al, 2026 show this for language models

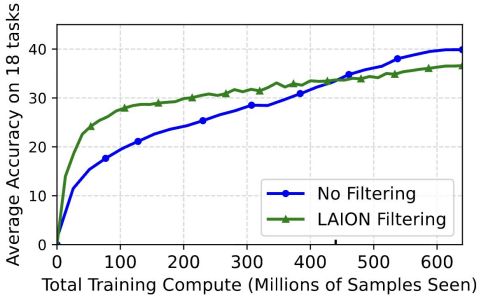


Figure 2. Given an initial data pool of 128M samples, we train ViT-B/32 CLIP models for a total of 640M samples. As we train for longer, the accuracy gains using the LAION-filtering subset that filters the common crawl to 10% of its initial size plateau. Surprisingly, even no-filtering of the common crawl is better than the popular LAION-filtering after seeing more than 450M samples.

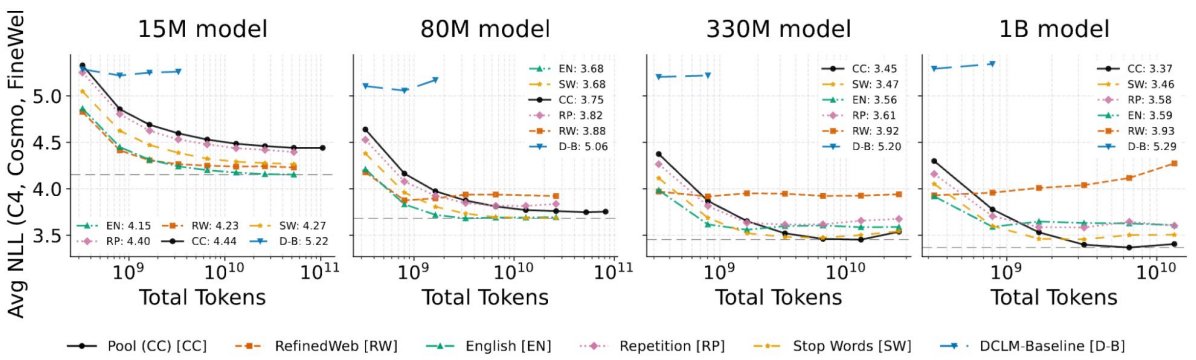


Figure 1: Comparison of models on 670M token CC pool and five filtered subsets. For sufficiently large models (330M+), the unfiltered pool (black) outperforms all five filters (colors) after sufficiently many optimization steps (x-axis, tokens under multiple epochs).

Compressing everything

Now that we're compressing all the data points, are we done?



Compressing everything

Common intuition from Data Processing Inequality: No deterministic transformation of a random variable can increase mutual information

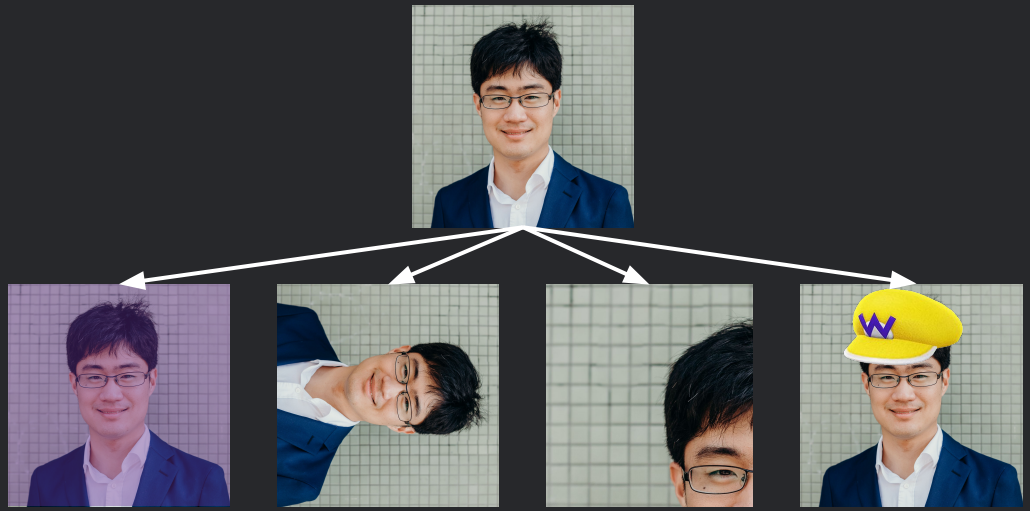
$$\text{If } X' = f(X), \text{ then } I(Y, X') \leq I(Y, X)$$

where

$$I(Y, X) = H(Y) - H(Y \mid X)$$

Augmentation

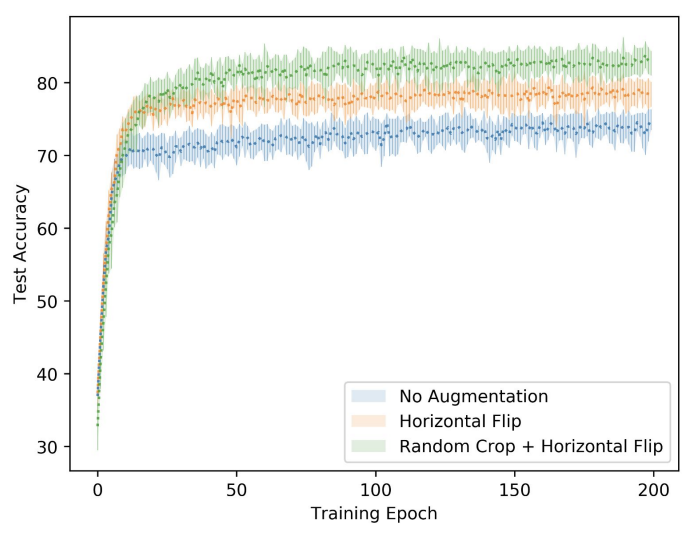
Interestingly, data augmentation (perturbation of the images) can further improve performance. Suggests that models are not optimal compressors of real data

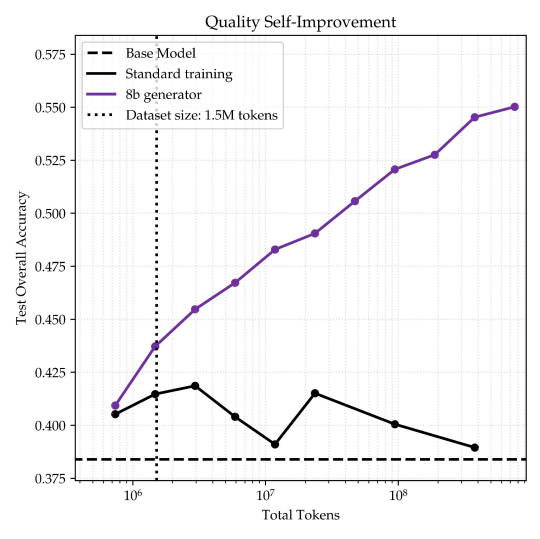


Augmentation

Data augmentation has been absolutely critical for modern ML performance.

When training with data augmentations, we can continue training for much longer than typical training





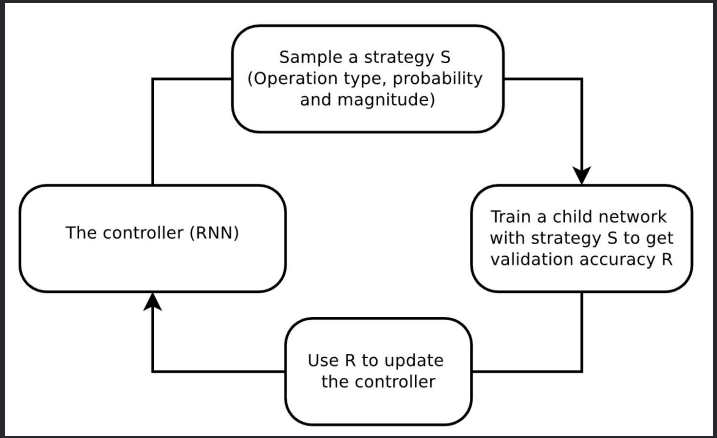
Augmentation

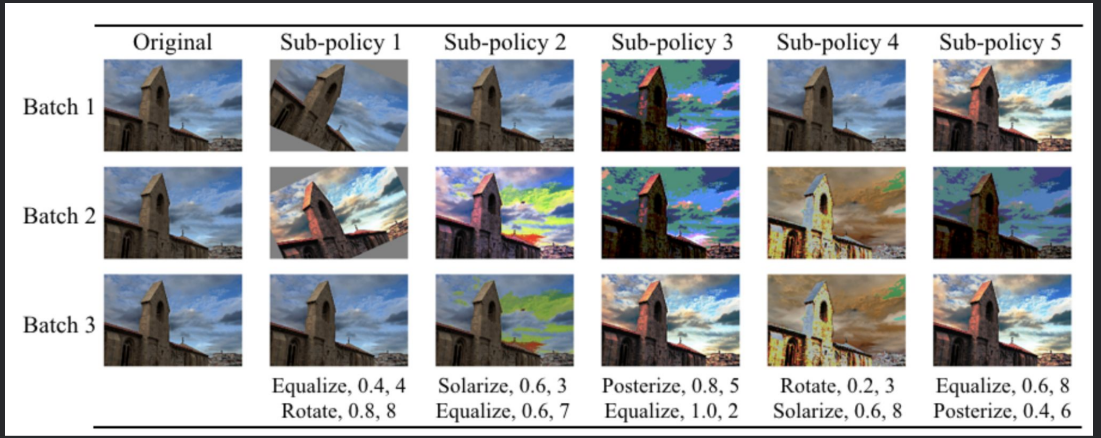
Some augmentations are quite cursed. For example, one of the most effective is mixup [Zhang et al, 2018] which interpolates multiple input output pairs



Augmentation

Since augmentations are domain-specific, some work try to automatically discover these augmentations (e.g. AutoAugment [Cubuk et al, 2018]). Open research question to remove domain dependence in a more satisfactory way





Augmentation

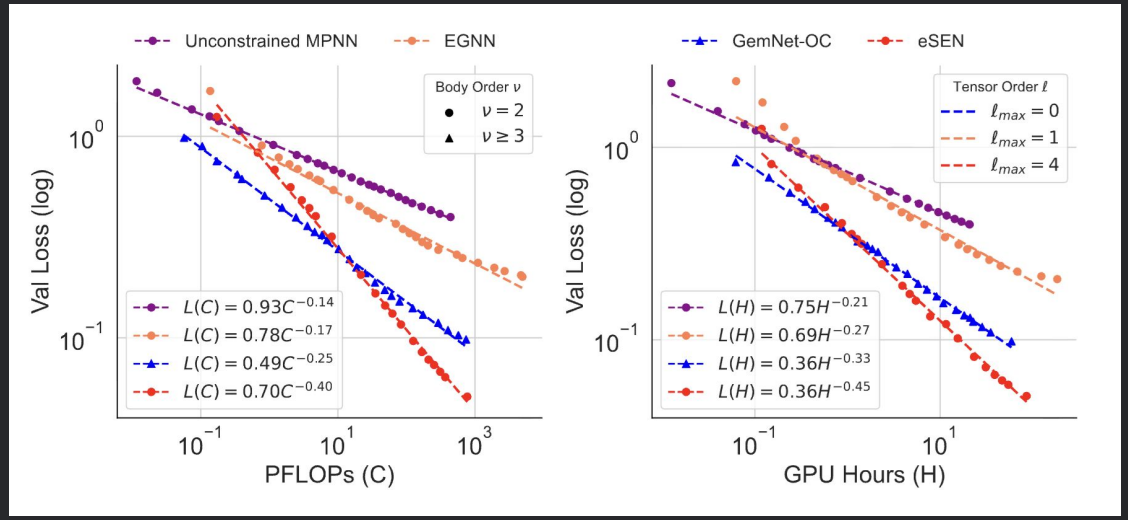
If our models are compressing the original data, why does augmentation help?

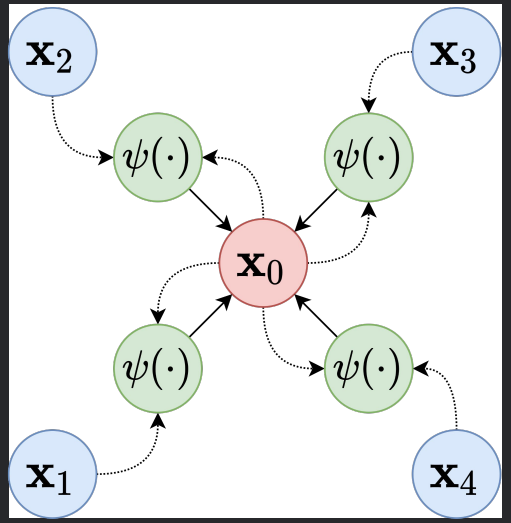
One perspective is that augmentations give additional invariances that the compressor hasn't inferred from the data

(for example, the class does not change when the label flips)

Augmentation

Architectures can be designed to explicitly enforce this invariance. For example, common protein/molecule model architectures enforce invariance to 3D translation, 3D rotation, and permutation [Ngo et al, 2025].





Augmentation

In vision, we can design creative architectures that enforce invariances of interest such as ignoring image angle [Tai et al, 2019]

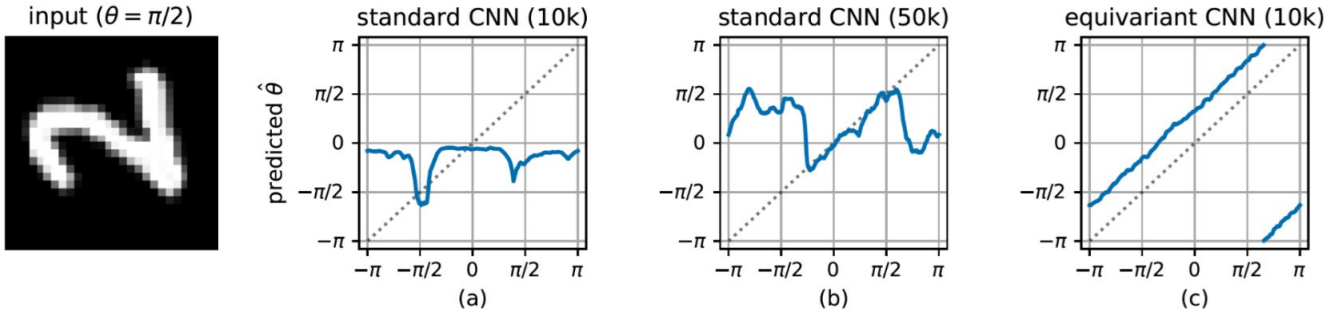


Figure 1. **Sample complexity for predicting rotations.** Predicted rotation angles vs. true angles for a rotated MNIST digit (left). The predictions of a self-consistent pose predictor will be parallel to the diagonal (dotted line). **(a)** After training with 10k rotated examples, a pose prediction CNN is not self-consistent; **(b)** with 50k rotated examples, it is only self-consistent over a limited range of angles. In contrast, **(c)** a rotationally-equivariant CNN outputs self-consistent predictions after 10k examples (with small error due to interpolation and boundary effects). There is a nonzero bias in $\hat{\theta}$ since the pose labels are latent and there is no preferred image orientation.

Augmentation as spectral clustering

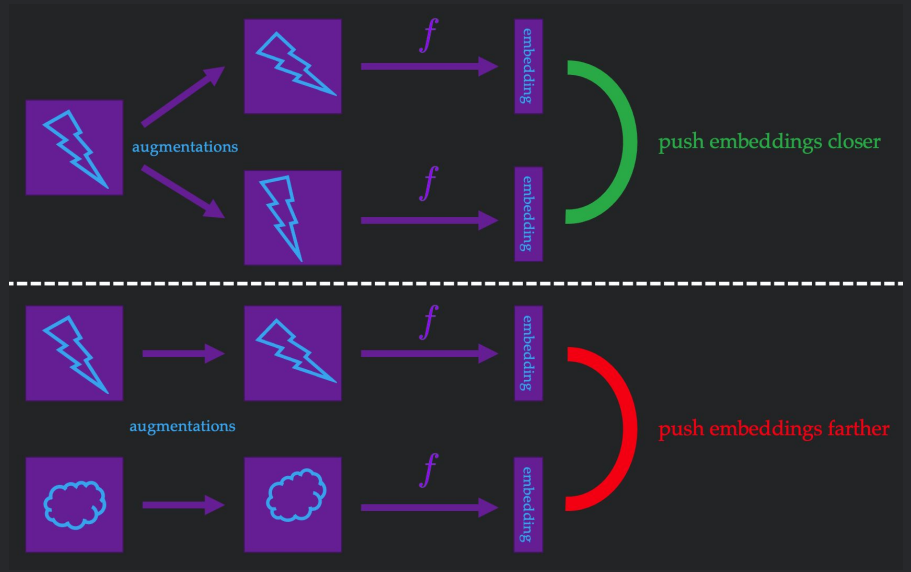
Data augmentation can induce these invariances/equivariances more conveniently without modifying the architecture, objective, or optimizer

When we use a data augmentation, what structure is the model learning?

One perspective from contrastive learning: representations are implicitly performing spectral clustering [HaoChen et al, 2022]

Detour: contrastive learning

Contrastive learning is an unsupervised algorithm to learn meaningful representations for downstream tasks. These embeddings can then be used for downstream tasks with relatively limited labels (self-supervised learning)



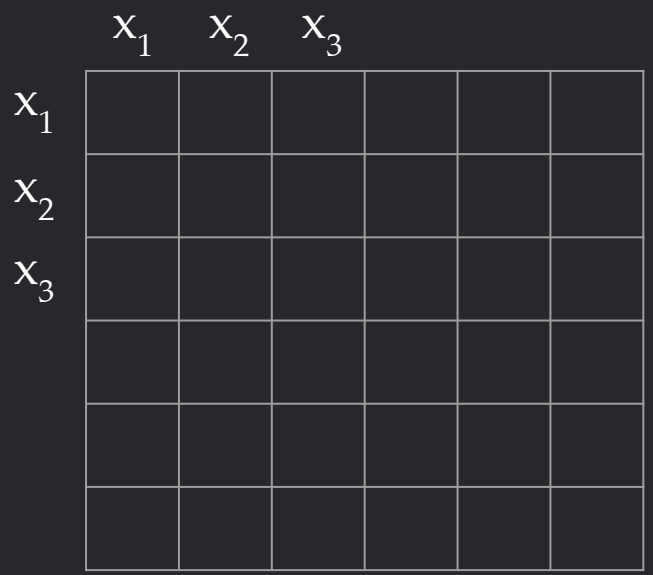
Detour: contrastive learning

Dominant algorithm is SimCLR [Chen et al, 2020]. A modified version of SimCLR minimizes the following spectral contrastive loss: upweight similarity from same class, downweight similarity from different classes

$$\mathcal{L}(f) = -2 \cdot \mathbb{E}_{x,x^+} \left[f(x)^\top f(x^+) \right] + \mathbb{E}_{x,x^-} \left[\left(f(x)^\top f(x^-) \right)^2 \right],$$

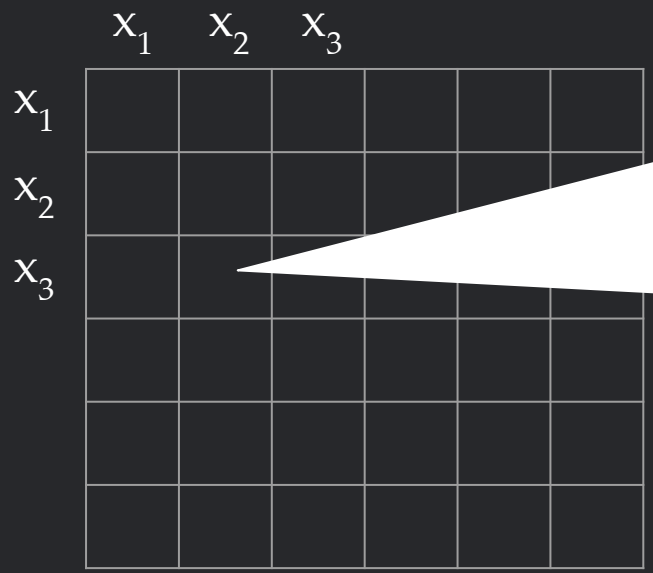
Spectral clustering

What embeddings does self-supervised learning find? To understand this, we first build a similarity graph between all possible data points



Spectral clustering

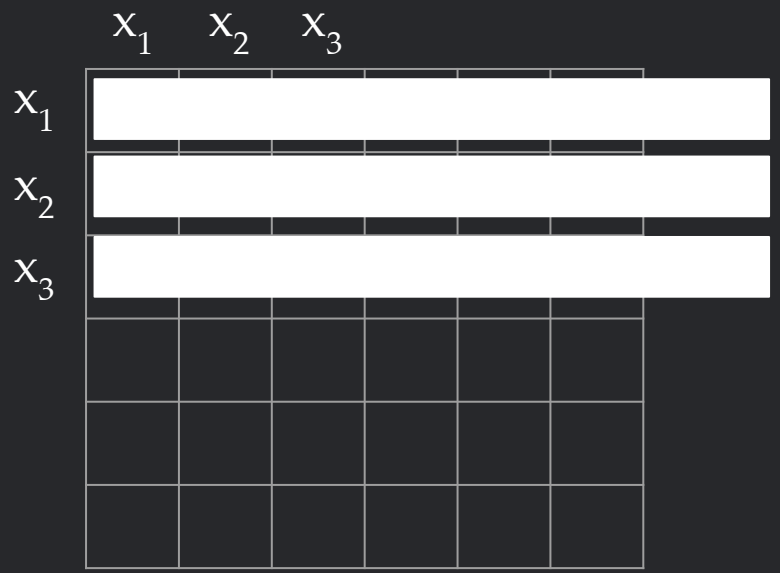
We will define a similarity metric between points as a function of the data augmentation: what is the likelihood the augmentation generated both points?



$$w_{xx'} := \mathbb{E}_{\bar{x}\sim\mathcal{P}_{\overline{\mathcal{X}}}}\left[\mathcal{A}(x|\bar{x})\mathcal{A}(x'|\bar{x})\right]$$

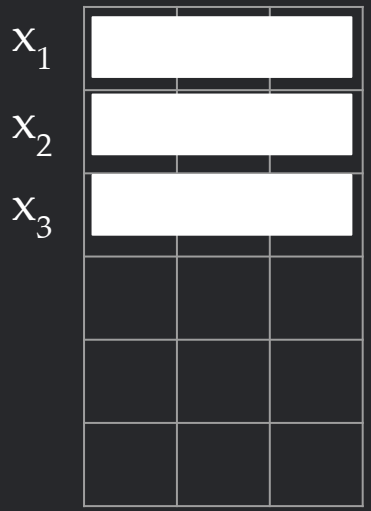
Spectral clustering

After we define this relationship, we get an embedding for each point. However, this embedding can be very high dimensional (even infinite-dimensional)



Spectral clustering

Therefore, for each point we would like to perform a low-rank approximation to keep low-dimensional embeddings per point.



Spectral clustering

Core theorem: Up to normalization, the loss minimizer of the spectral contrastive loss outputs embeddings equivalent to the low-rank approximation of the similarity graph [HaoChen et al, 2022]



$$\mathcal{L}(f) = -2 \cdot \mathbb{E}_{x,x^+}\left[f(x)^\top f(x^+)\right] + \mathbb{E}_{x,x^-}\left[\left(f(x)^\top f(x^-)\right)^2\right],$$

$$w_{xx'} := \mathbb{E}_{\bar{x}\sim\mathcal{P}_{\bar{\mathcal{X}}}}\left[\mathcal{A}(x|\bar{x})\mathcal{A}(x'|\bar{x})\right]$$

Spectral clustering

Therefore, we can reason about data augmentation implicitly defining a new relationship structure over the data points.



$$\mathcal{L}(f) = -2 \cdot \mathbb{E}_{x,x^+}\left[f(x)^\top f(x^+)\right] + \mathbb{E}_{x,x^-}\left[\left(f(x)^\top f(x^-)\right)^2\right],$$

$$w_{xx'} := \mathbb{E}_{\bar{x}\sim\mathcal{P}_{\mathcal{X}}}\left[\mathcal{A}(x|\bar{x})\mathcal{A}(x'|\bar{x})\right]$$

Summary

When constrained by data, we should keep around as much data as we can

Since models are not optimal compressors, we can generate meaningful data with domain-specific data augmentations that introduce additional information

Overall summary

Algorithmic Information Theory gives us useful intuitions for designing algorithms (even if the actual suggestions are not practical)

We can (sometimes) improve optimization through minimizing sharpness

We can (sometimes) improve architecture through maximizing expressivity with fixed description length

We can (sometimes) improve data through filtering less and generating more