

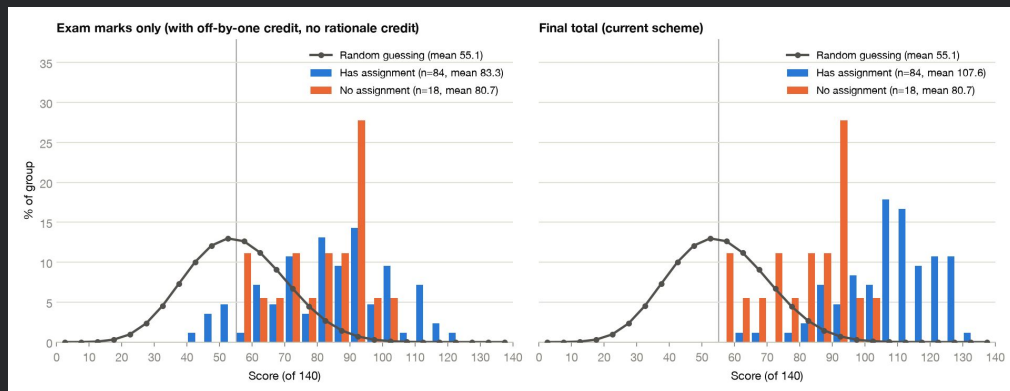
Loss Landscapes Part 1

Basins, River Valley, Critical Points

10/13

Announcements

Quiz 2 scores are out! It was hard but y'all's average was good!



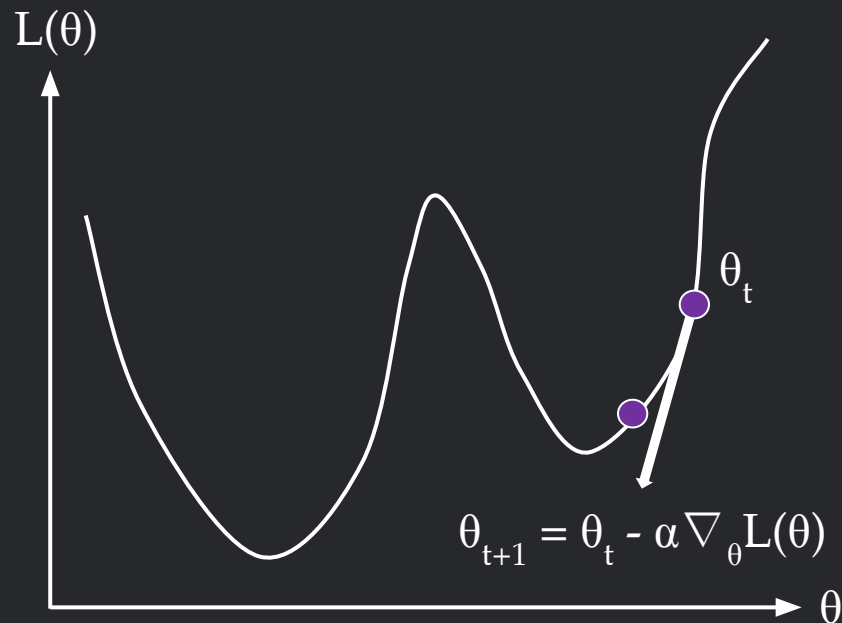
This week we will be having tutorial. You should have received an email suggesting what time to meet. Plan to ask questions about assignment/quiz 2 in addition to discussing a problem related to the last unit

Loss landscapes

This assignment will be developing an intuition for the loss landscape, or the relationship between the loss and the parameters

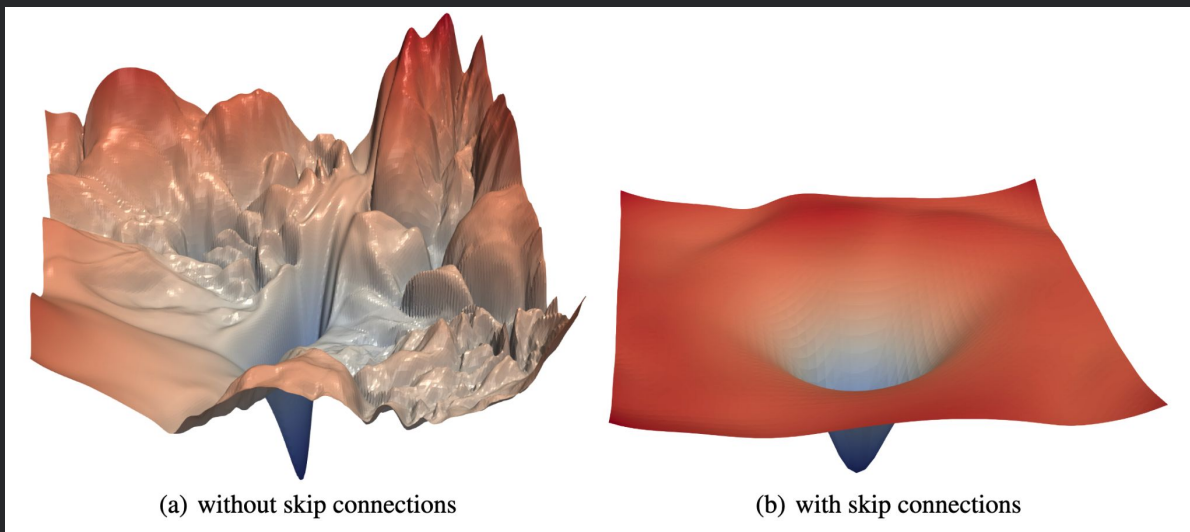
Many suspicions remain

- How non-convex is the landscape?
- Do models get stuck anywhere?
- Do models end in the same minima?
- How sharp or flat is the landscape?



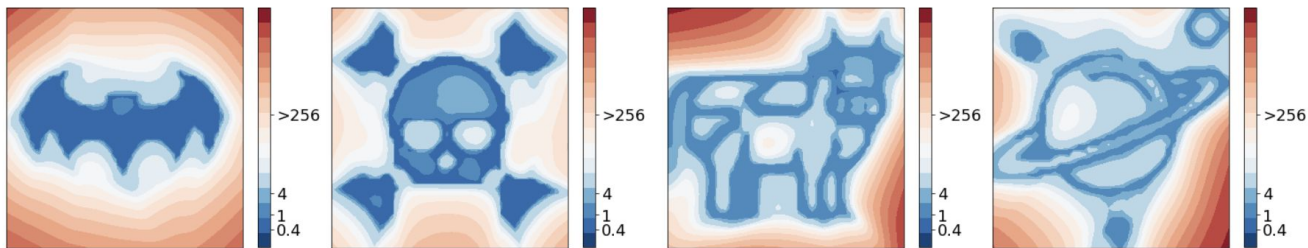
Loss landscapes

The real loss landscape is much more complicated than our cartoons. For example, the choice of architecture significantly affects the smoothness of the landscape [[Li et al, 2018](#)]



Loss landscapes

The real loss landscape is much more complicated than our cartoons. In fact, its possible to construct loss landscapes that would somewhat defy our intuitions, looking like literal cartoons [[Skorokhodov and Burtsev et al, 2025](#)]



(a) Loss surface on FashionMNIST dataset

(b) Loss surface on CIFAR10 dataset

Figure 1: Examples of a loss landscape of a typical CNN model on FashionMNIST and CIFAR10 datasets found with MPO. Loss values are color-coded according to a logarithmic scale. We used a small VGG-like model which architecture is presented in appendix B; additional visualizations are presented in appendix D. Here we used *test* sets to compute the loss values.

High-dimensional intuitions

How do we even begin to analyze the landscape?

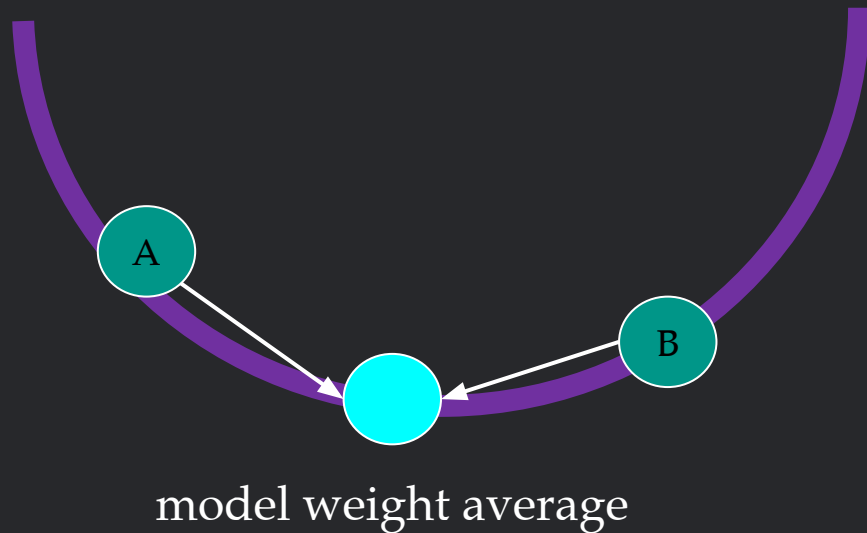
“To deal with hyper-planes in a 14-dimensional space, visualize a 3-D space and say ‘fourteen’ to yourself very loudly. Everyone does it.” - Geoffrey Hinton

Researchers have developed heuristics to visualize natural high dimensional landscapes via slices/projections of the training landscape, low-dimensional toy examples, and summary statistics of high-dimensional properties

In today's lecture, we will go over a few mental models for the loss landscape ranging from empirical to theoretical

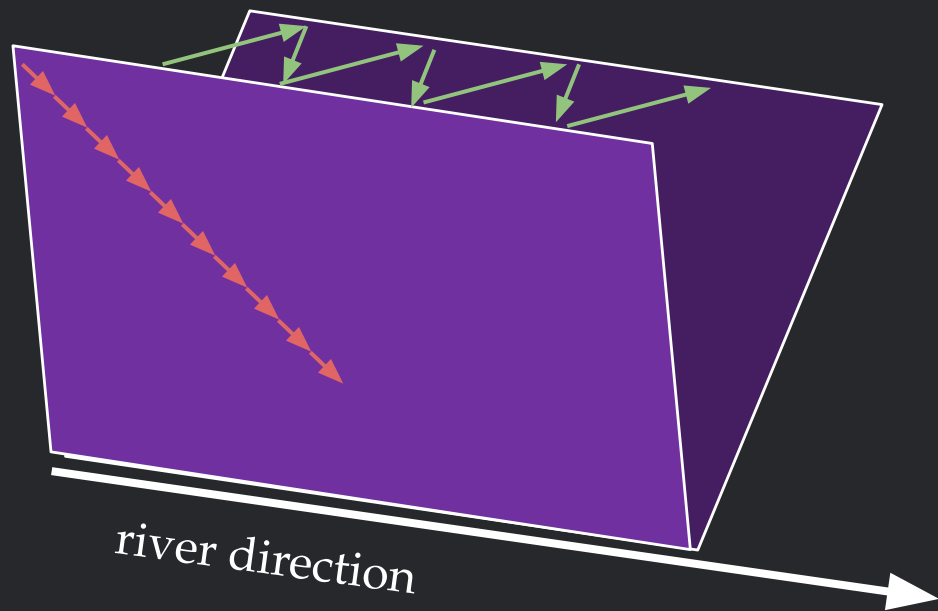
Loss landscape geometry

Model 1 (Basins): In certain training regimes, the loss landscape is actually convex. If two models end up in the same basin, averaging their weights can result in an even better set of parameters



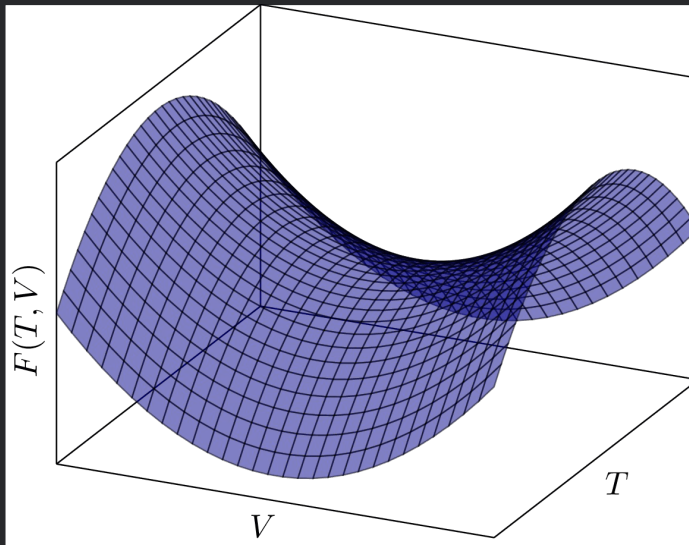
Loss landscape geometry

Model 2 (River Valley): Instead of **smoothly descending** the loss landscape, training **oscillates** between the valley while making progress along the river



Loss landscape geometry

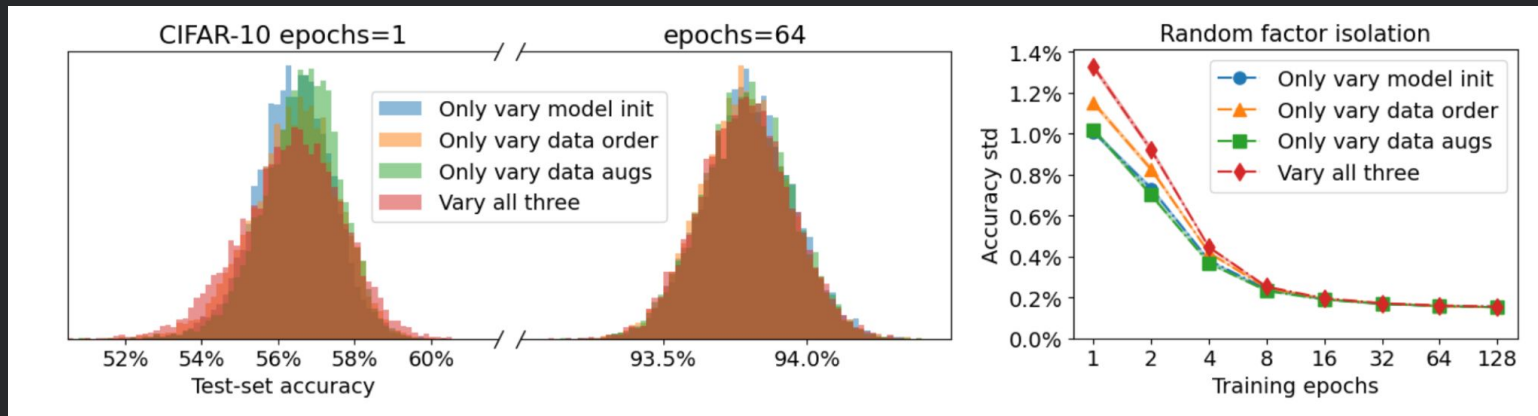
Model 3 (Critical points): Does training ever get stuck in local minima or saddle points? Are these points harmful for performance?



Basins and Mode Connectivity

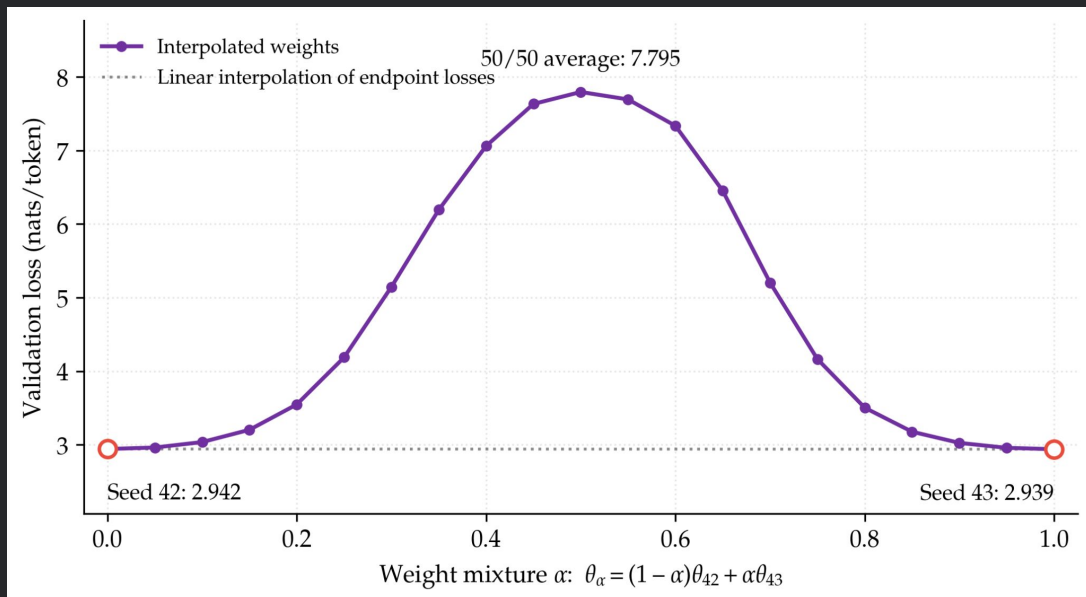
Basins

Traditionally, deep learning is thought to have a very chaotic landscape with many local minima. Therefore, it may not be expected for training runs to end up in the same minima



Most simple example fails

Initial experiment: train two default d8 models completely separately and average their weights. The average of their weights results in a totally nonsensical model. Since the intermediate models are worse, the endpoints are not in the same basin

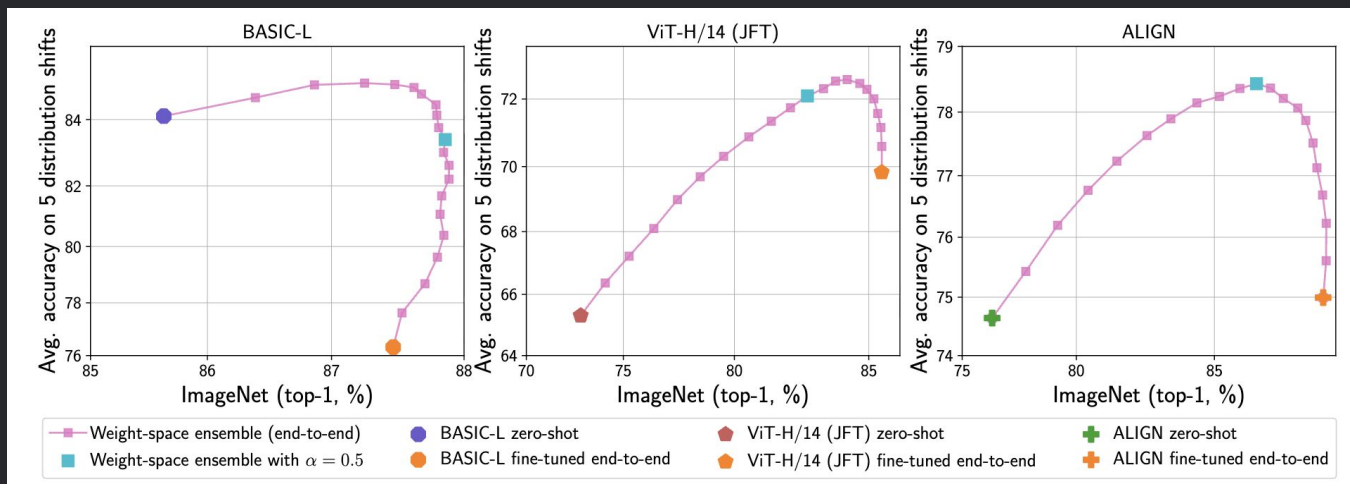


Most simple example fails

However, there are many examples where models do end up in the same minima. In addition to pre-training, we will look to the many examples in fine-tuning.

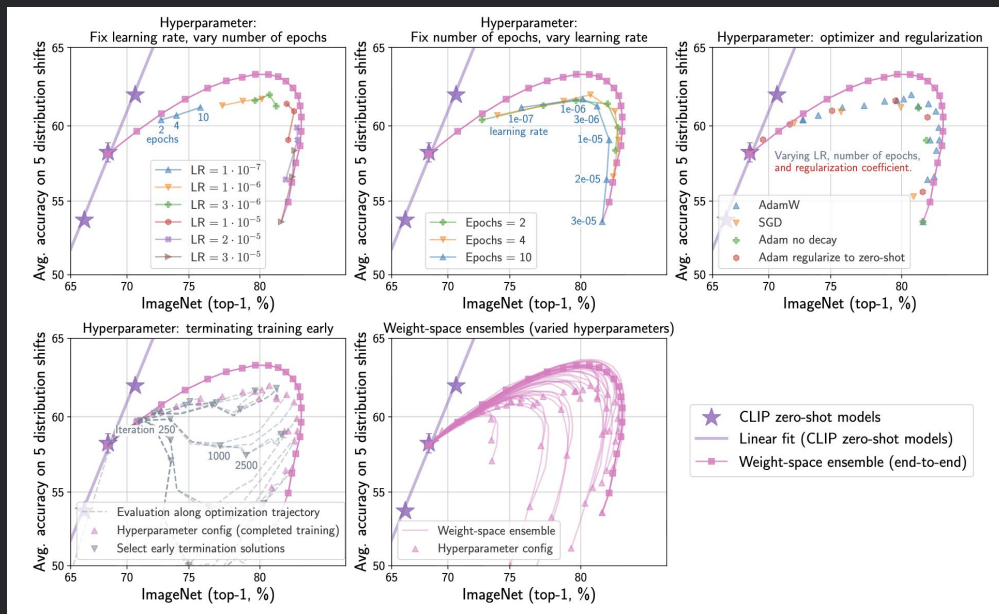
Averaging fine-tuning with initialization

We first analyze averaging a fine-tune and its initialization [[Wortsman and Ilharco et al, 2021](#)]. By averaging their weights, you can improve in-distribution (and out-of-distribution) performance.



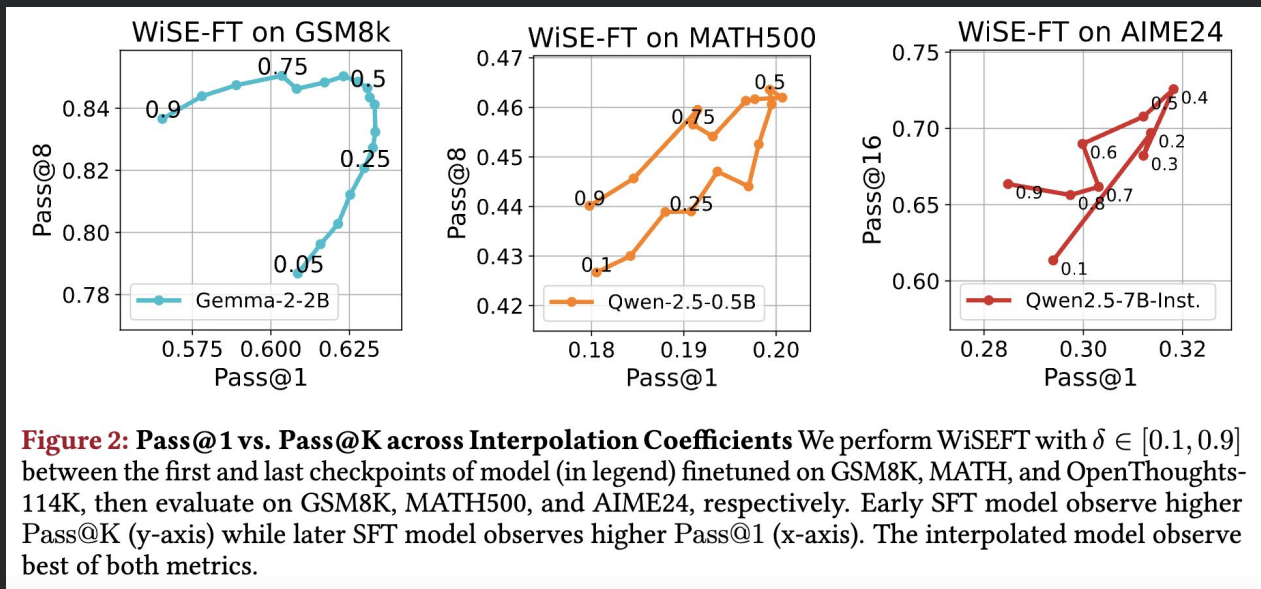
Averaging fine-tuning with initialization

The hull induced by averaging can even outperform tuning various hyperparameters such as learning rate, epoch count, weight decay, etc.



Averaging fine-tuning with initialization

This technique can be used with other performance tradeoffs such as accuracy and coverage for reasoning models [[Dang et al, 2025](#)]



Averaging fine-tuning runs

For fine-tuned models, you can average completely separate runs! One representative paper is Model Soups [[Wortsman et al, 2022](#)]

	Method	Cost
Best on val. set	$f(x, \arg \max_i \text{ValAcc}(\theta_i))$	$\mathcal{O}(1)$
Ensemble	$\frac{1}{k} \sum_{i=1}^k f(x, \theta_i)$	$\mathcal{O}(k)$
Uniform soup	$f(x, \frac{1}{k} \sum_{i=1}^k \theta_i)$	$\mathcal{O}(1)$
Greedy soup	Recipe 1	$\mathcal{O}(1)$
Learned soup	Appendix I	$\mathcal{O}(1)$

Recipe 1 GreedySoup

Input: Potential soup ingredients $\{\theta_1, \dots, \theta_k\}$ (sorted in decreasing order of $\text{ValAcc}(\theta_i)$).

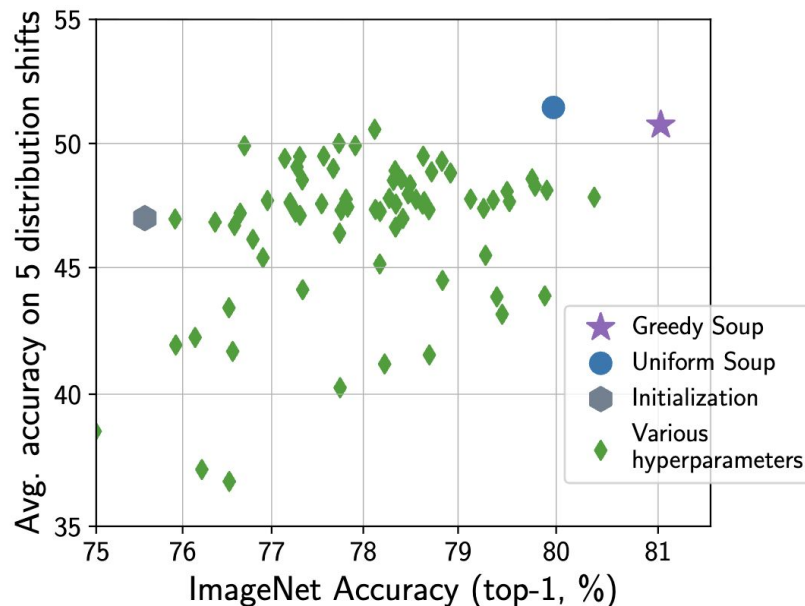
ingredients $\leftarrow \{\}$

for $i = 1$ **to** k **do**

if $\text{ValAcc}(\text{average}(\text{ingredients} \cup \{\theta_i\})) \geq$
 $\text{ValAcc}(\text{average}(\text{ingredients}))$ **then**

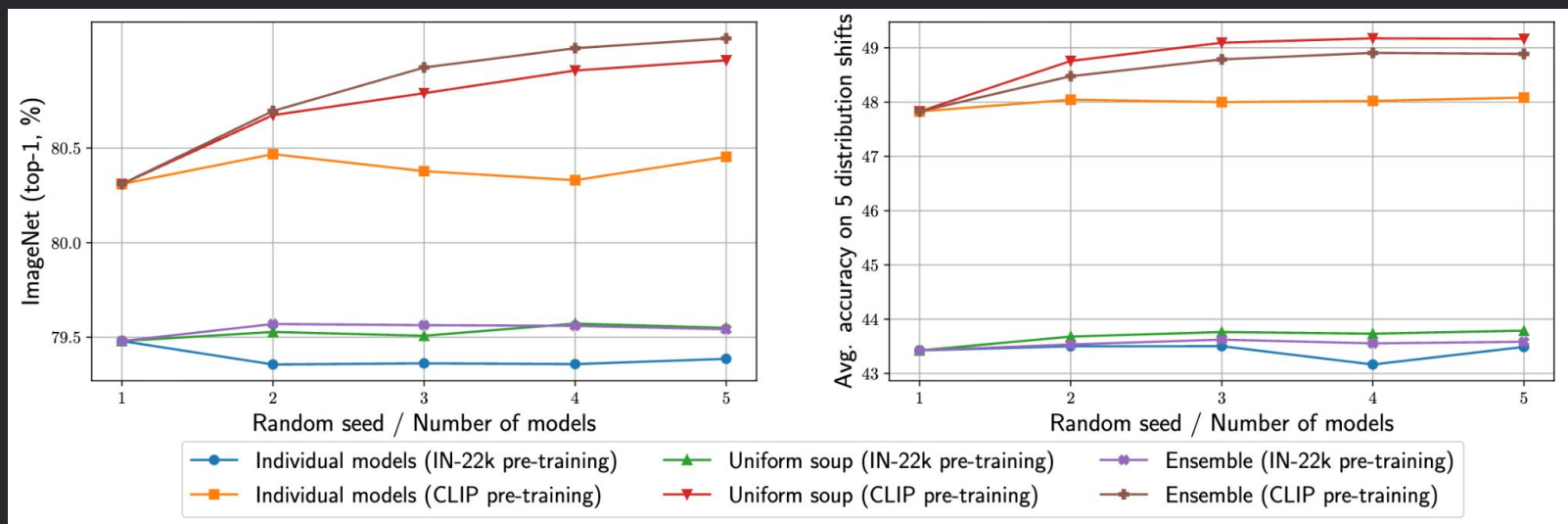
 ingredients $\leftarrow$ ingredients $\cup \{\theta_i\}$

return average(ingredients)



Averaging fine-tuning runs

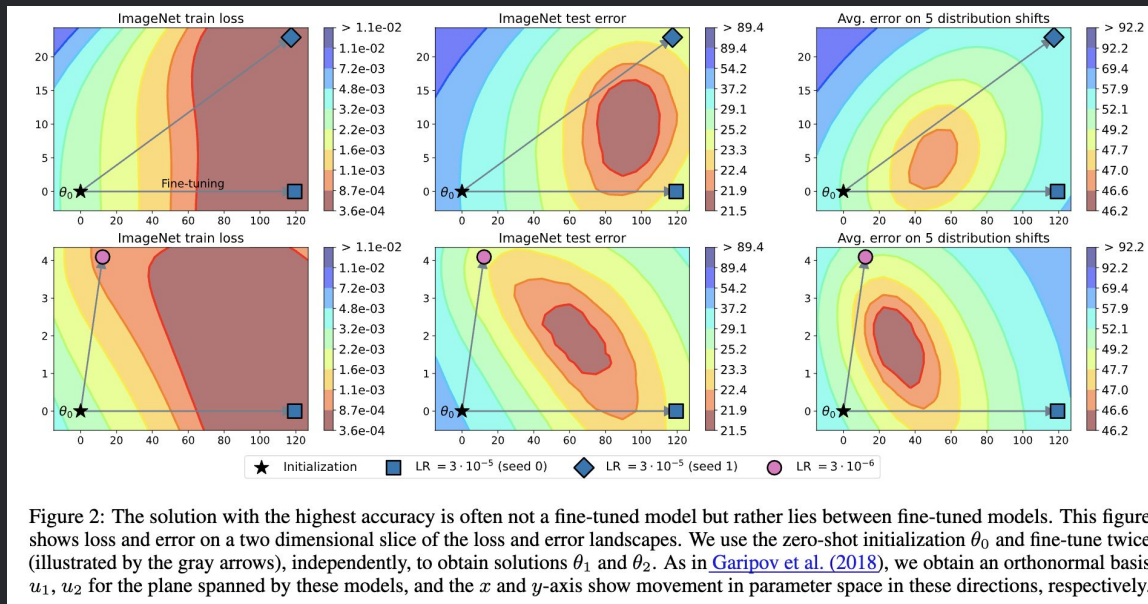
Most controlled experiment: fine-tunes with different seeds can be averaged for better performance.



Model soups: averaging weights of multiple fine-tuned models improves accuracy without increasing inference time

Averaging fine-tuning runs

Geometric intuition: the weights are in a loss basin and averaging two weights gets closer to the bottom even though the models achieve similar loss



Model soups: averaging weights of multiple fine-tuned models improves accuracy without increasing inference time

Averaging fine-tuning runs

Paper suggests that models that are more different (higher angle between weights) increase gain from weight averaging. To get more variation, they aggregate over different hyperparameters (learning rates, augmentation strengths)

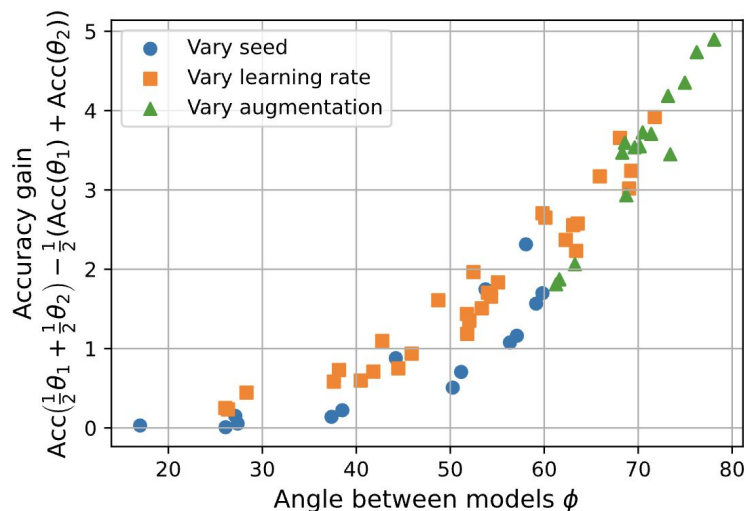
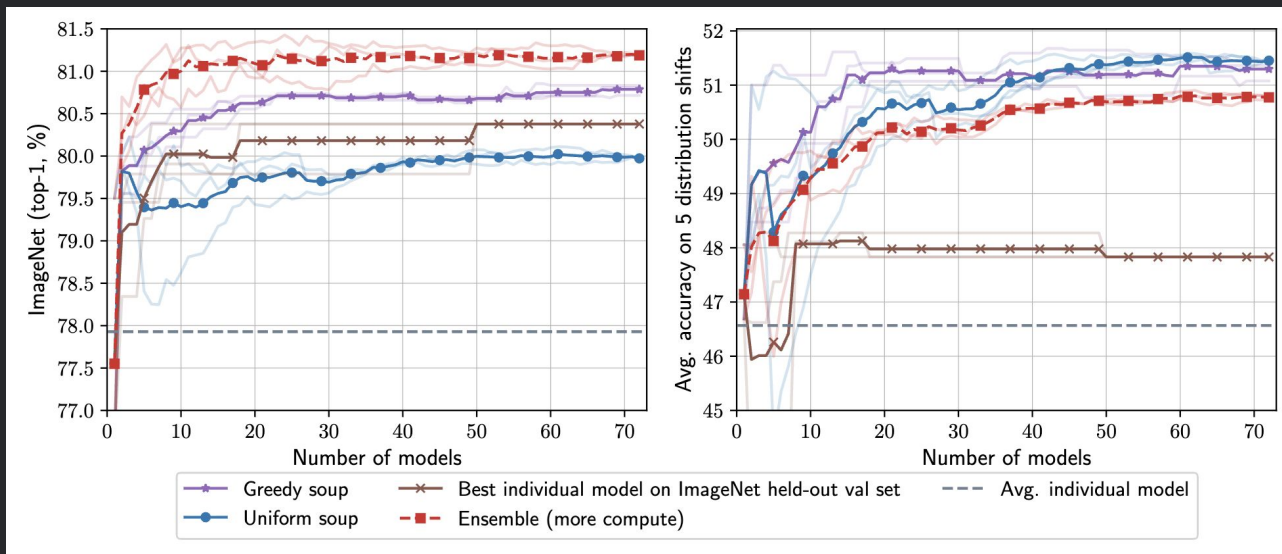


Figure 3: The advantage of averaging solutions (y -axis) is correlated with the angle ϕ between solutions, while varying hyperparameter configurations between pairs enables a larger ϕ . Each point corresponds to a pair of models θ_1, θ_2 that are fine-tuned independently from a shared initialization θ_0 with different hyperparameter configurations. The angle ϕ between solutions refers to the angle between $\theta_1 - \theta_0$ and $\theta_2 - \theta_0$ (i.e., the initialization is treated as the origin). Accuracy is averaged over ImageNet and the five distribution shifts described in Section 3.1.

Averaging fine-tuning runs

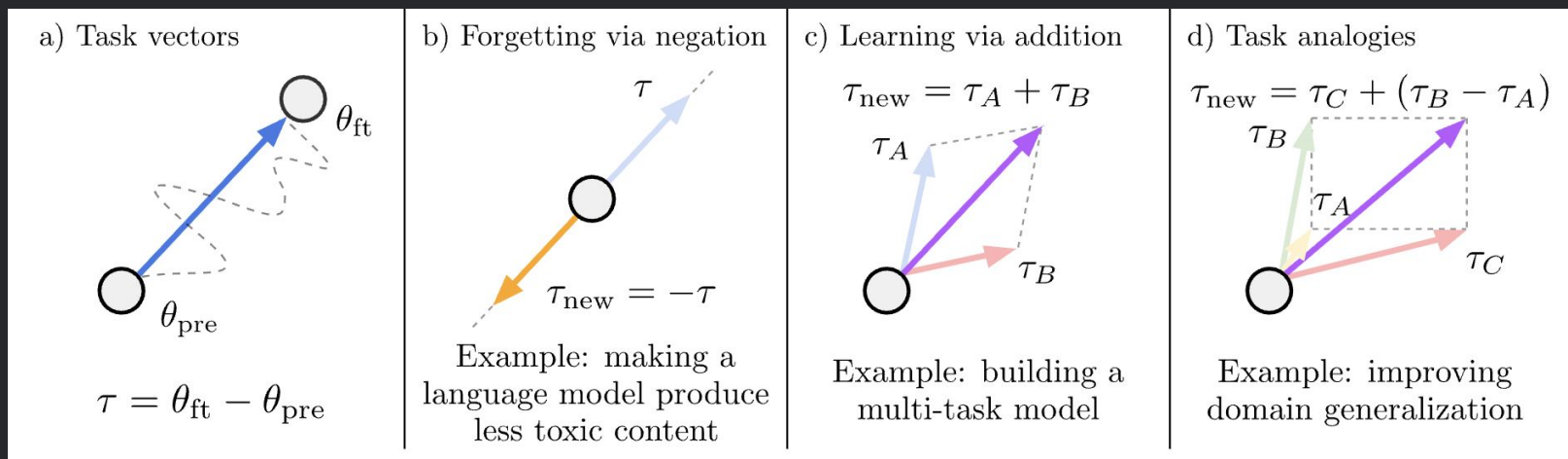
Similar scaling trends when aggregating across different hyperparameters. Greedily keeping the best hyperparameters gets better performance may even be essential



Model soups: averaging weights of multiple fine-tuned models improves accuracy without increasing inference time

Averaging fine-tuning tasks

The weight space around fine-tuning is so well-behaved that separate fine-tuned can be applied via addition/subtraction [[Ilharco et al, 2022](#)]



Averaging fine-tuning tasks

By fine-tuning the model on the task of interest and negating the task vector, we can remove the capability that we introduced into the model

Table 1: Forgetting image classification tasks via negation. Results are shown for CLIP models, reporting average accuracy (%) on the eight target tasks we wish to forget (Cars, DTD, EuroSAT, GTSRB, MNIST, RESISC45, SUN397 and SVHN), and the control task (ImageNet). Negating task vectors reduce the accuracy of a pre-trained ViT-L/14 by 45.8 percentage points on the target tasks, with little loss on the control task. Additional details and results are shown in [Appendix B](#).

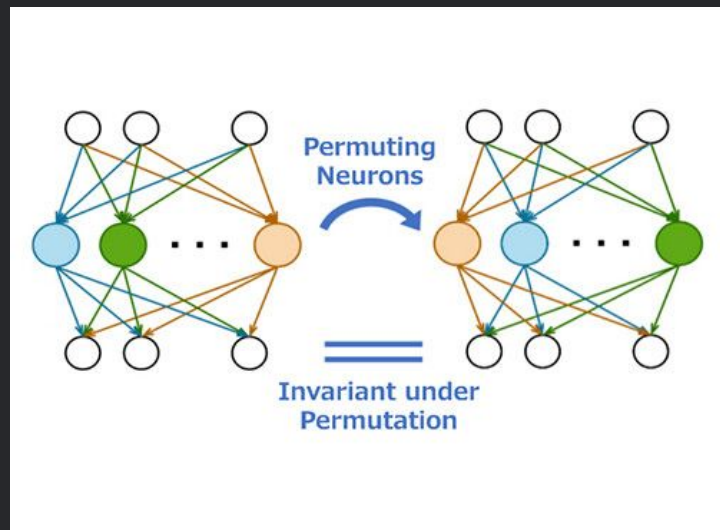
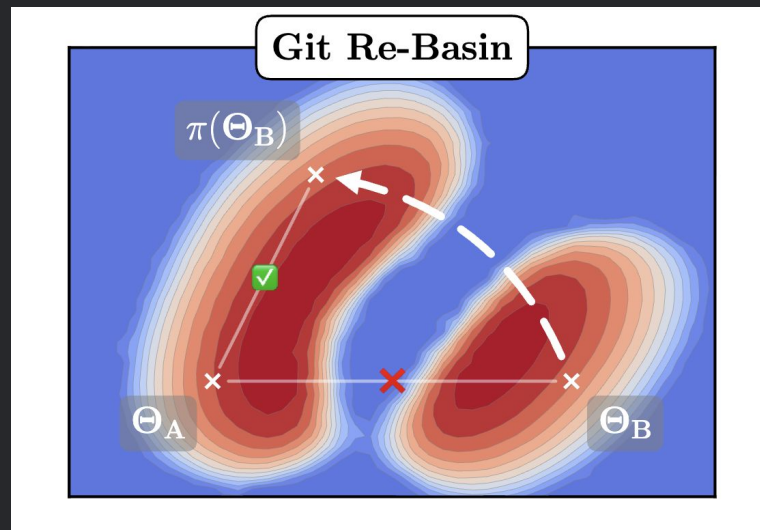
Method	ViT-B/32		ViT-B/16		ViT-L/14	
	Target (↓)	Control (↑)	Target (↓)	Control (↑)	Target (↓)	Control (↑)
Pre-trained	48.3	63.4	55.2	68.3	64.8	75.5
Fine-tuned	90.2	48.2	92.5	58.3	94.0	72.6
Gradient ascent	2.73	0.25	1.93	0.68	3.93	16.3
Random vector	45.7	61.5	53.1	66.0	60.9	72.9
Negative task vector	24.0	60.9	21.3	65.4	19.0	72.9

Averaging two separate pre-training runs

If weight averaging works so well for fine-tuning, why doesn't it work for pre-training?

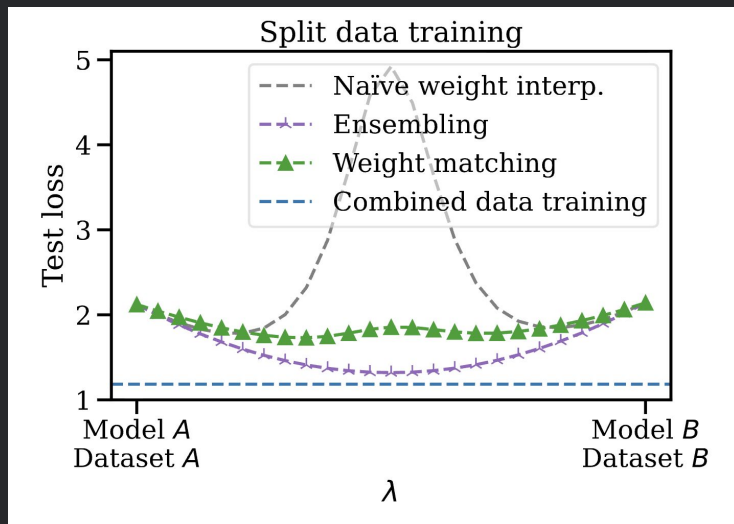
Averaging two separate pre-training runs

Git Re-Basin [[Ainsworth et al, 2022](#)] and Model Fusion via OT [[Singh and Jaggi 2019](#)] believe that models belong to the same basin under permutation symmetry.



Averaging two separate pre-training runs

Git Re-Basin [[Ainsworth et al, 2022](#)] and Model Fusion via OT [[Singh and Jaggi 2019](#)] believe that models belong to the same basin under permutation symmetry. However, feature matching still underperforms other methods like ensembling



Averaging pre-training with initialization

Within a run, it might be the case that the final checkpoints is mode-connected to the initialization [[Lucas et al, 2021](#)]

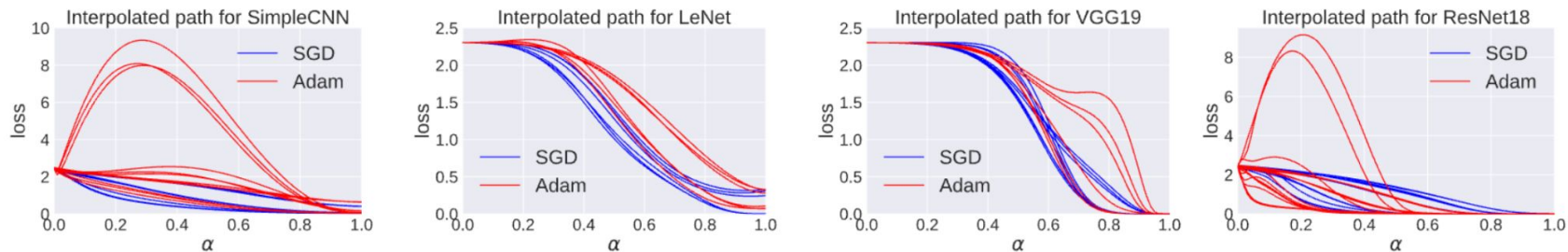
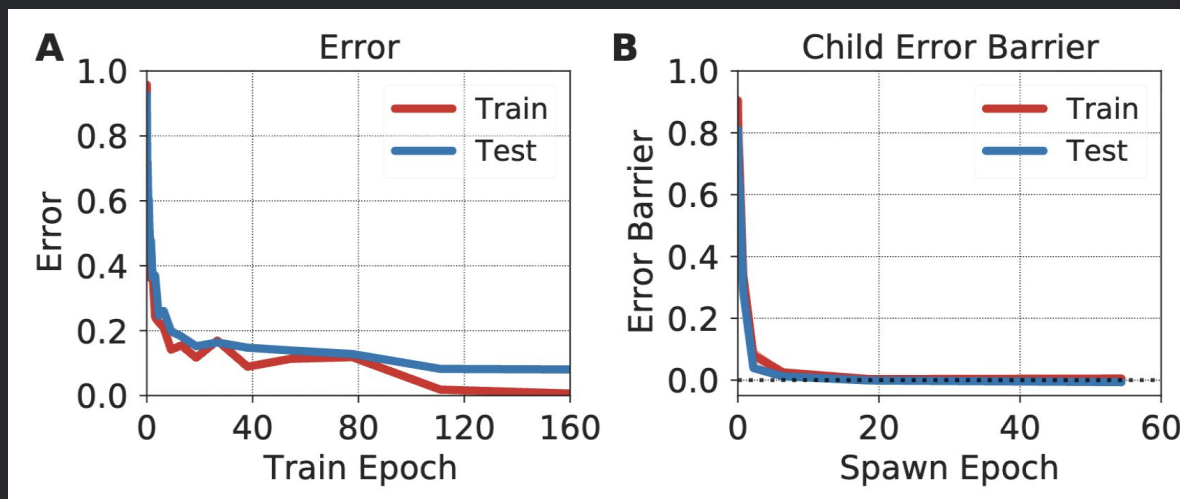


Figure 2. Training loss over the linear interpolation connecting initial and final parameters. Each curve represents a network trained on CIFAR-10 with different hyperparameter configurations (achieving at least 1.0 training loss). The MLI property holds for networks trained with SGD, but often fails for networks trained with Adam.

Averaging two forked pre-training runs

Another way to get models to end up in the same basin is share the start of training and diverge runs later as explored in [Fort et al, 2020](#), similar to fine-tuning. You will be doing a lot of this in your assignment!



Summary

Fine-tuning often ends up in the same basin, enabling interesting properties from simply averaging model weights

Pre-training maintains mode connectivity within a training run but does not naturally permit averaging across different runs

However, pre-training can be made to enter the same run under modifications

Hessians

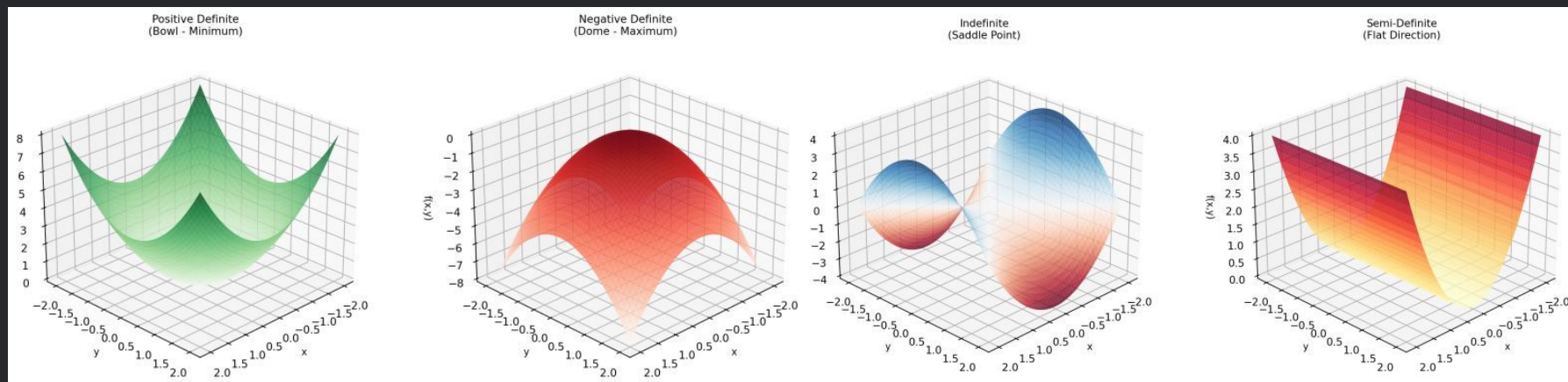
Hessian

For the next two mental models (river valley and critical points), we will want to reason about the curvature of the loss landscape. One core tool for this is the Hessian (second derivative) of the loss landscape

$$f : R^n \rightarrow R$$
$$H_f = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2^2} & \cdots & \frac{\partial^2 f}{\partial x_2 \partial x_n} \\ \vdots & & & \\ \frac{\partial^2 f}{\partial x_n \partial x_1} & \frac{\partial^2 f}{\partial x_n \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_n^2} \end{bmatrix}$$

Hessian

It is most natural to reason about the eigenvalues of the Hessian. The absolute value of the eigenvalue determines sharpness/flatness while the sign of the eigenvalue indicates whether the landscape faces up or down (under the eigenbasis)



We typically estimate the spectrum of the Hessian using Lancsoz iterations

Second-order optimizers using the Hessian

If we knew the full Hessian, we could exploit it to perform Newton's method, equivalent to gradient descent preconditioned by the Hessian:

$$\theta_{t+1} = \theta_t - H^{-1} \nabla_{\theta} L(\theta)$$

In convex optimization theory, gradient descent achieves $O(1/T)$ error over steps whereas Newton's method achieves $O(1/T^2)$

Second-order optimizers using the Hessian

In the Noisy Quadratic Model, we can model pre-conditioning as $p=1$ below for Hessian eigenvalues h_i

Optimal preconditioning results in much faster convergence (at the cost of potentially increasing steady state risk)

$$\mathbb{E} [\mathcal{L}(\boldsymbol{\theta}(t))] \leq \sum_{i=1}^d (1 - \alpha h_i^{(1-p)})^{2t} \mathbb{E} [\ell(\theta_i(0))] + \sum_{i=1}^d \frac{\alpha c_i h_i^{-p}}{2B(2 - \alpha h_i^{1-p})}.$$

Second-order optimizers using the Hessian

The Hessian may have negative eigenvalues which is a problem when using it for an update step. Instead of considering the Hessian, we can instead use the Gauss-Newton matrix

The Gauss-Newton matrix is defined to be the first term of the following decomposition of the Hessian, where $z := f(x)$ denotes the pre-softmax outputs of the model f and a goes over the output dimensions of f :

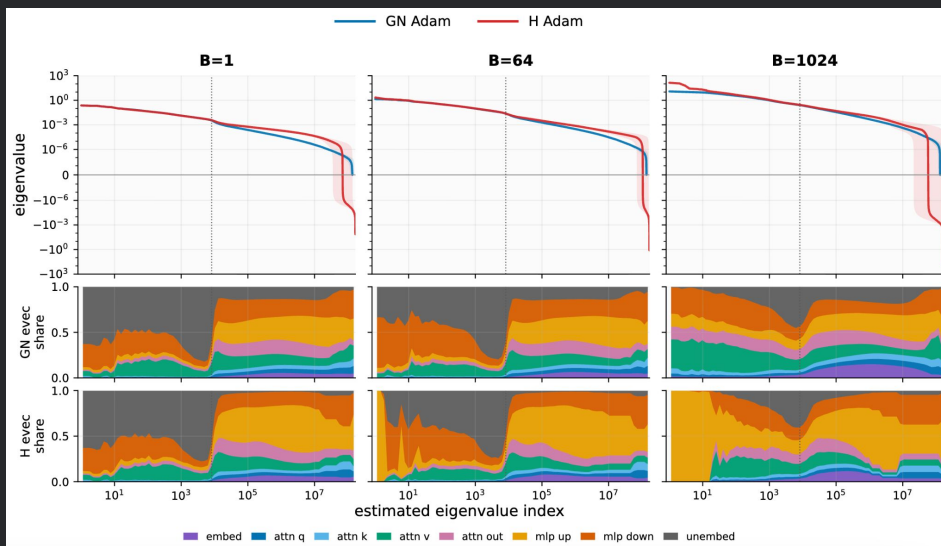
$$\nabla_{\theta}^2 \mathcal{L}(\theta) = \underbrace{\nabla_{\theta} f(\theta)^{\top} \nabla_z^2 \mathcal{L}(\theta) \nabla_{\theta} f(\theta)}_{\text{Gauss-Newton matrix}} + \sum_a \frac{\delta \mathcal{L}}{\delta z_a} \nabla_{\theta}^2 [f(\theta)]_a$$

Intuitively, this keeps the Hessian with respect to the model outputs instead of keeping the Hessian with respect to model parameters

Gauss-Newton is PSD (nonnegative eigenvalues). Though it differs in eigenvalues from the Hessian, it agrees at the minima.

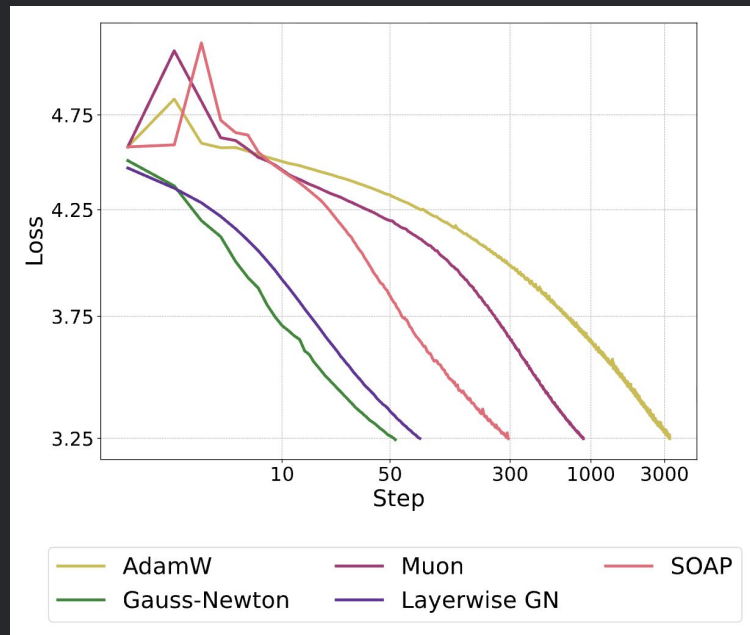
Empirical spectra

[Meterez et al, 2026](#) estimates eigenvalues of the Hessian and Gauss-Newton matrix for different batch sizes. There is a smooth decay in the eigenvalues and most (but not all) are positive. The dominant part mostly agrees



Second-order optimizers using the Hessian

Training using second-order information from the Gauss-Newton matrix can significantly accelerate convergence across steps at the cost of extra compute per step [[Abreu et al, 2025](#)]



Interaction with our mental models

In the River Valley mental model, we will argue that there is high oscillation in high curvature directions even though there is progress in low curvature directions

For critical points, we will reason about whether we are at saddle points (Hessian with positive and negative eigenvalues) or local minima (all negative)

For sharpness (Thursday lecture) we will discuss how the max eigenvalue of the Hessian behaves over training

River Valley

Theoretical intuition for gradient descent

If you recall the Noisy Quadratic Model, our loss obeys the following form for learning rate α , batch size B , Hessian eigenvalues h_i

$$\mathbb{E} [\ell(\theta_i(t))] = \underbrace{(1 - \alpha h_i)^{2t}}_{\text{convergence rate}} \mathbb{E} [\ell(\theta_i(0))] + (1 - (1 - \alpha h_i)^{2t}) \underbrace{\frac{\alpha c_i}{2B(2 - \alpha h_i)}}_{\text{steady state risk}},$$

This implies that if we train at a constant learning rate schedule we would converge to an irreducible steady state risk as $t \Rightarrow \infty$, seems undesirable

Theoretical intuition for gradient descent

Theory offers two common paths for dealing with the steady state risk

- Reduce the step size over the course of training [[Polyak, 1963](#)]
- Average weights over training to cancel out noise [[Polyak, 1990](#)]

Analysis for a single direction with $B = 1, h_i = 1$:

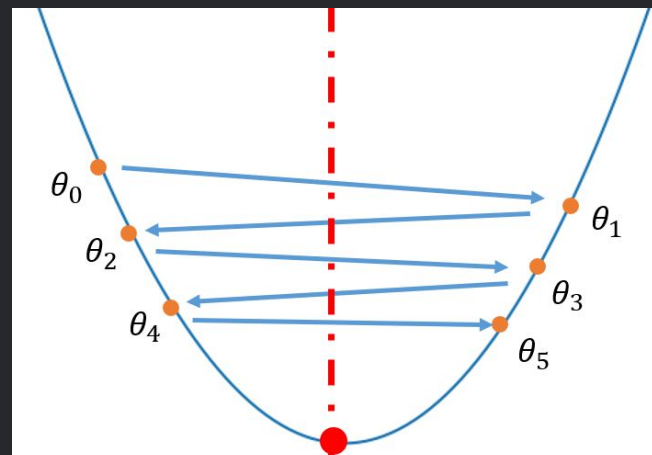
SGD

$$= \frac{\alpha\sigma^2}{2(2-\alpha)} \left(1 - (1 - 2\alpha + \alpha^2)^T\right) + \frac{1}{2}(1 - \alpha)^{2T} w_0^2.$$

Polyak
averaging

$$\bar{w}_T = \frac{1}{T} \sum_{t=1}^T w_t$$

$$\mathbf{E}[f(\bar{w}_T)] - f^* \leq \frac{f(w_0) - f^*}{\alpha^2 T^2} + \frac{\sigma^2}{2T}$$



Theoretical intuition for gradient descent

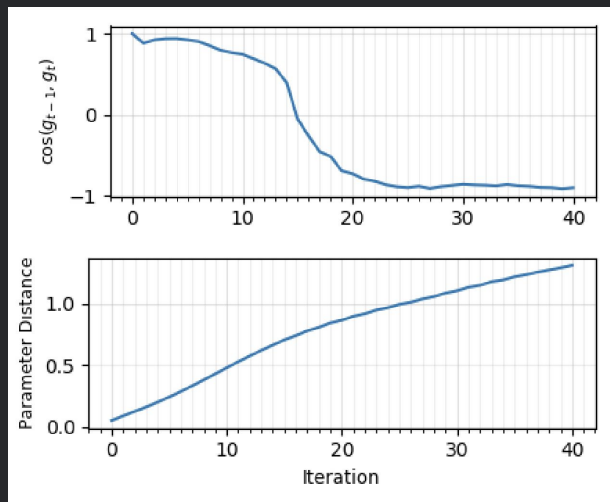
How well does the theoretical intuition line up with practice?

We will cover deep-learning specific experiments to understand how reflective the Polyak mental model is and what other components we need to add

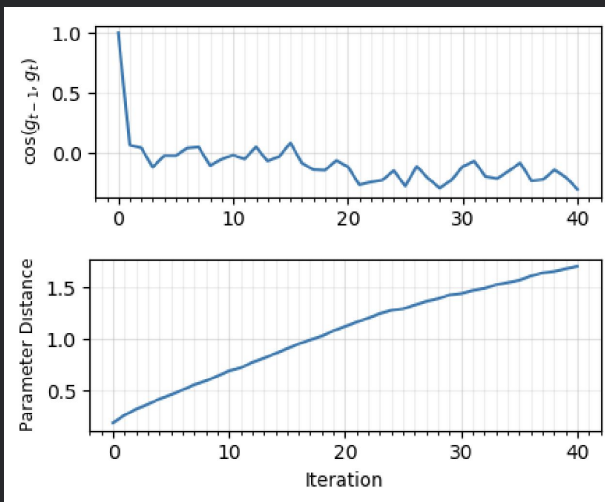
Averaging within a pre-training run

To measure oscillation, we can measure the correlation between successive updates. [[Xing et al, 2018](#)] shows negative similarity for GD, suggesting oscillation rather than consistent progress in the same direction. (SGD may have weaker sign because of noise)

GD

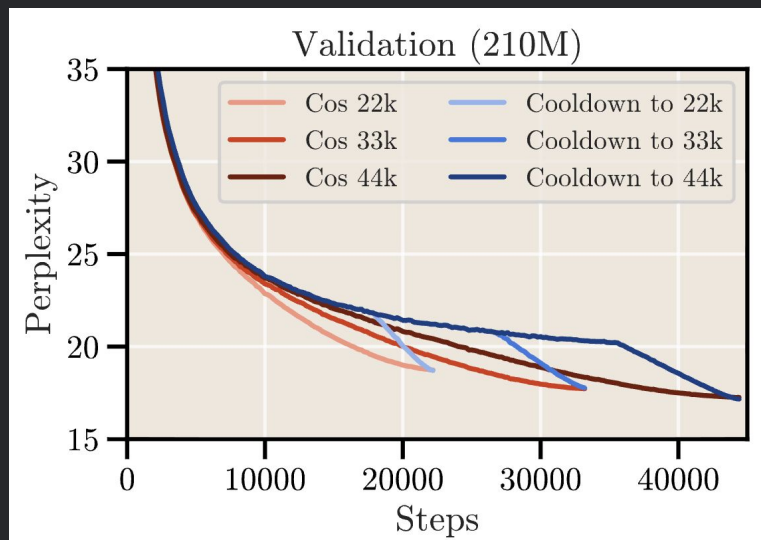


SGD



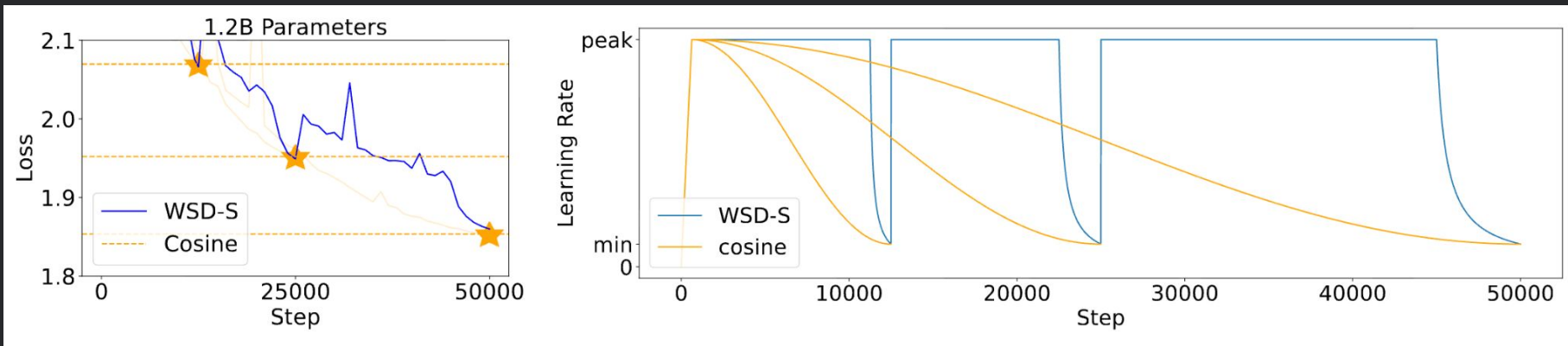
Implications for learning rate schedule

Under this perspective, different learning rate schedules get different shapes and how high learning rates may still be making hidden progress [[Hagele et al, 2024](#)]. Decreasing the learning rate reveals this hidden progress by entering the river



Implications for learning rate schedule

If the river valley perspective was true, we could rewarmup our learning rate to make more progress [[Wen et al, 2024](#)]. In your homework, you will analyze whether rewarming up does better or worse than a single learning rate schedule

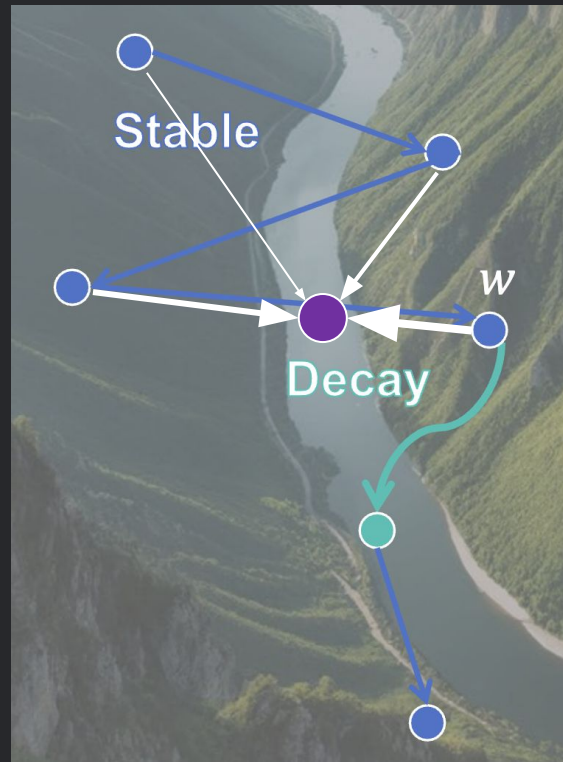


Implications for model averaging

Do the predictions of Polyak averaging also hold up in our mental model?

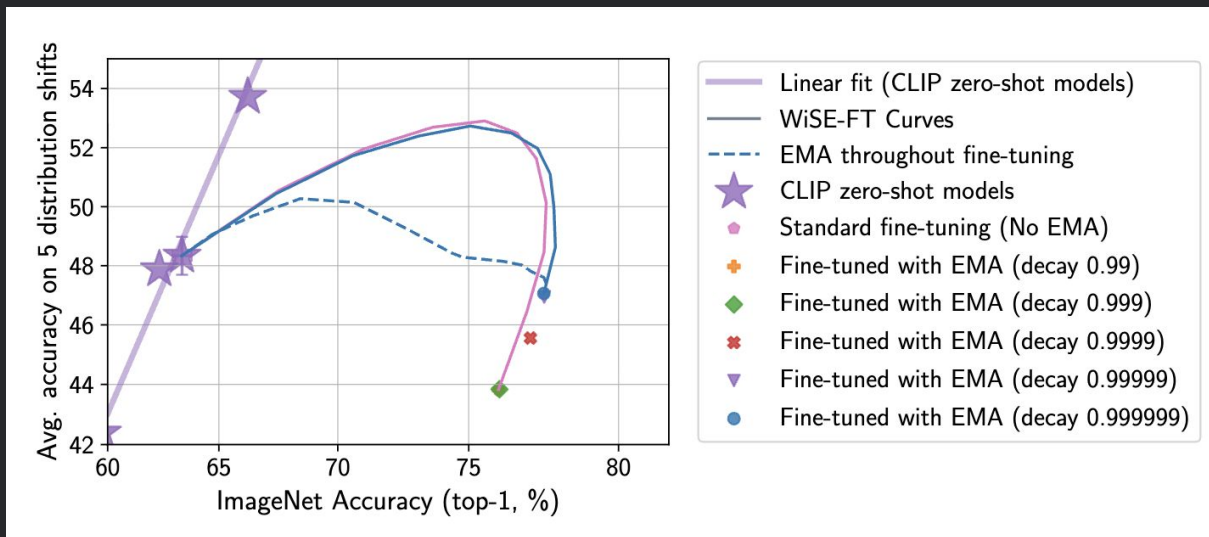
If the model is oscillating above the valley, we can estimate performance along the river by averaging the iterates (with EMA or Polyak)

In your assignment, you will assess what performance this achieves



Disagreement with basins

Interestingly, WiSE-FT suggests that the benefit of EMA is different from the benefit of averaging initialization and endpoint is different from the benefit of EMA.



Case study: multiple token horizons

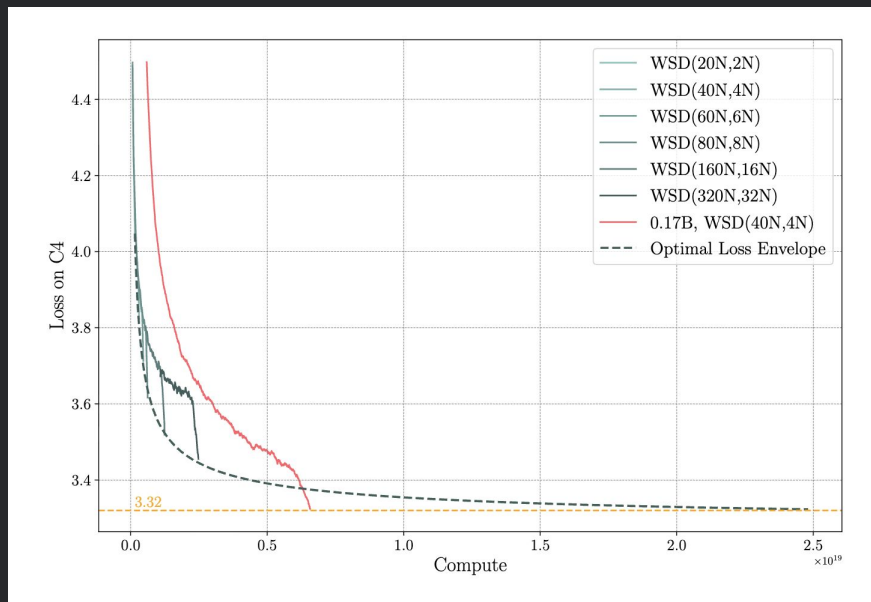
In practice, one common concern is estimating the post-annealing performance of training across multiple token horizons. This is complicated because the naive approach would require a separate learning rate schedule per experiment.

We will use this practical desiderata as a case study for the predictivity of Polyak

Case study: multiple token horizons

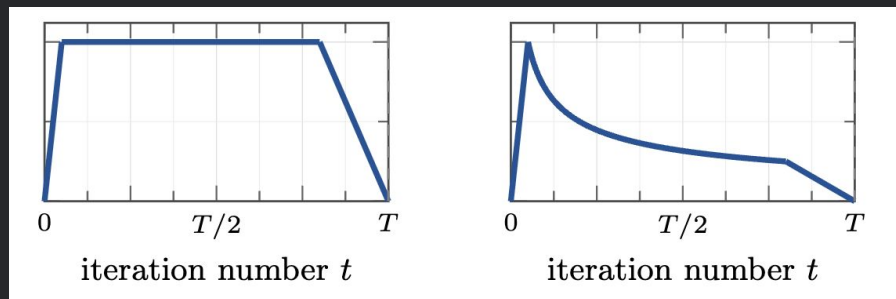
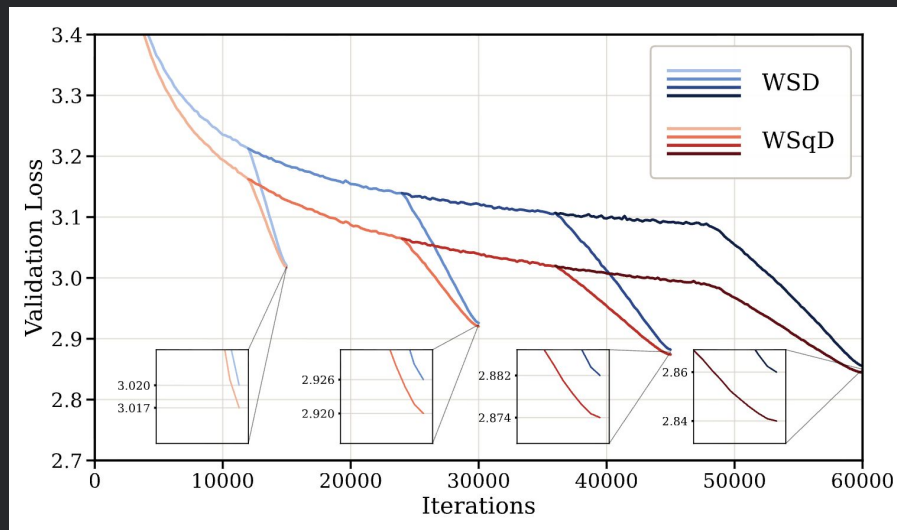
Option 1: MiniCPM keeps one long stable phase with multiple decay phases at different horizons [[Hu et al, 2024](#)]

Complication: as you find in assignment 2, optimal learning rate depends on horizon, incurring some regret from have a shared peak learning rate



Case study: multiple token horizons

Option 2: Horizon free learning rate schedules attempt to address the decreasing optimal learning rate. For example, decreasing peak learning rate as inverse square root of step count to scale the optimal peak learning rate. You will assess these in your assignment



Case study: multiple token horizons

Option 3: Polyak averaging to estimate the performance at the bottom of the river

Summary

The necessity of learning rate schedules or weight averaging schemes is predicted by classical convex optimization

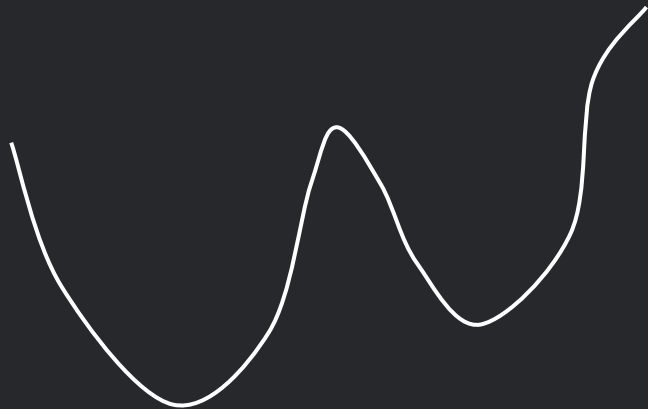
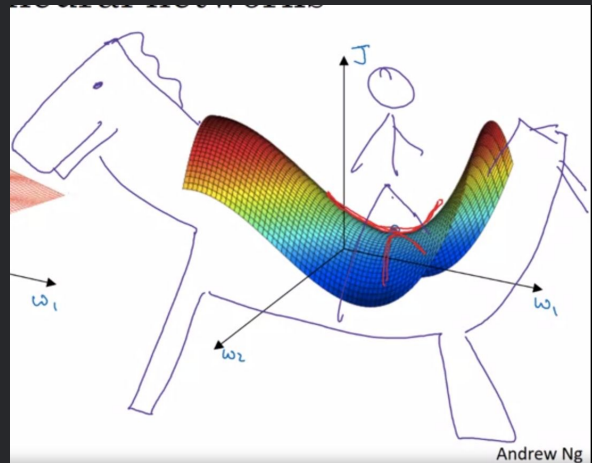
We can geometrically visualize the classical theory as a river valley landscape where training oscillates

These mental models directly impact how we carry out training runs in practice

Critical points

Fear of local minima and saddle points

Deep learning researchers were/are constantly stressed about ending up in critical points (i.e. gradient is zero) where the objective is not fully optimized. These are either local minima (positive Hessian eigenvalues) or saddle points (some positive, some negative)



Overview

In practice, people don't really utilize methods to avoid local minima and saddle points. We'll go over a series of folklore arguments suggesting that

- There are many more saddle points than local minima
- Local minima, if they existed, would get similar loss to global minima
- Gradient descent almost always avoids saddle points

This is an example of our low-dimensional intuitions being a poor proxy for what happens in high dimensions.

Statistical physics intuition

One way to build intuition is to analyze the curvature of random matrices as a proxy for real models. This style of research is popular in statistical physics and related to our μP discussion in Optimization 1.

To start, [Bray and Dean, 2006](#) analyze error ε and index α (fraction of negative eigenvalues) at critical points for Gaussian random fields and find a monotonic relationship between both quantities.

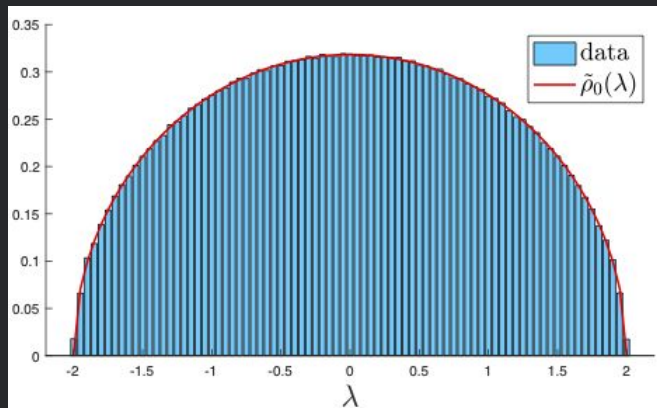
Statistical physics intuition

A tight correspondence between ε and α suggests

- 1) Critical points with high error relative to global minimum are exponentially likely to be saddle points
- 2) Local minima with index 0 are guaranteed to have error close to global minimum

Statistical physics intuition

Intuition from random matrix theory:



- A random Gaussian matrix has eigenvalue distribution following Wigner's semicircle law with mean 0. Each eigenvalue has probability 1/2 of pos/neg
- [Bray and Dean, 2006](#) show that Hessian of a critical point follows same distribution but shifted by an amount determined by ε .
 - Global minimum shifts spectrum far right enough such that all eigenvalues are positive
 - High error shifts spectrum far left enough such that most eigenvalues are negative
- In this model with 1 dimension, an exact saddle is zero probability. As the number of parameters grows, local minima become exponentially less likely

Statistical physics intuition

[Dauphin et al, 2014](#) performs an empirical analysis of this intuition, finding a strong correlation between ϵ and α for deep neural network training

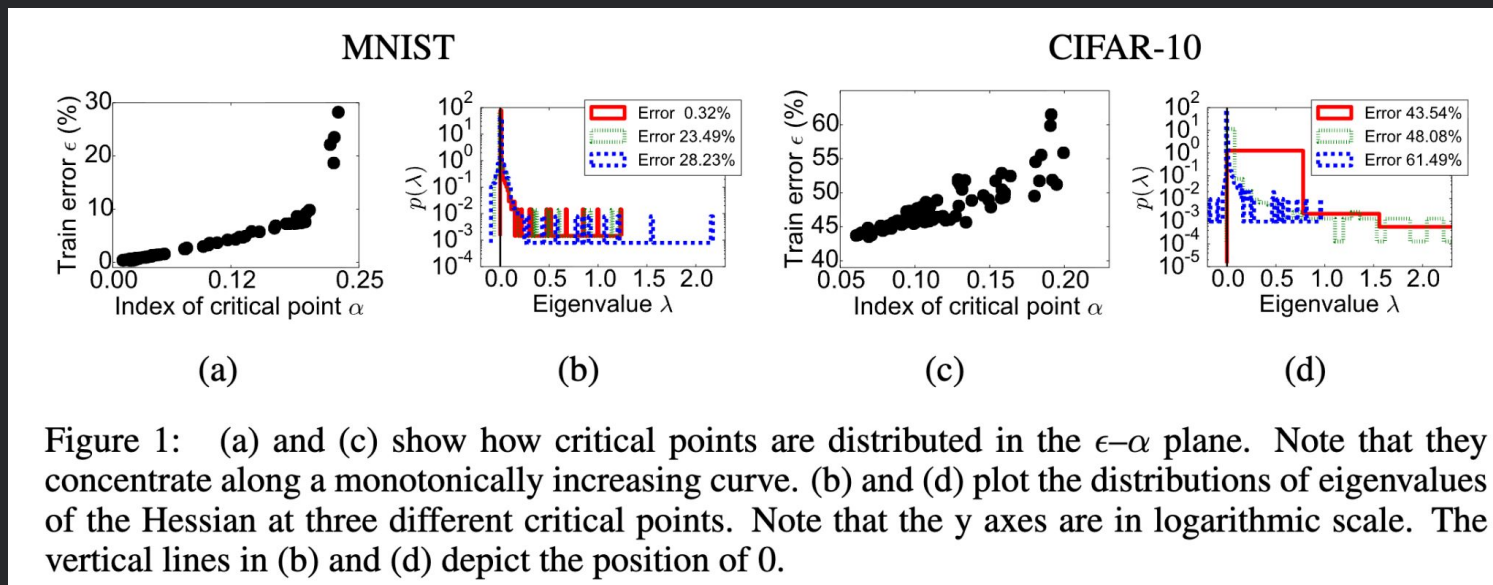


Figure 1: (a) and (c) show how critical points are distributed in the ϵ - α plane. Note that they concentrate along a monotonically increasing curve. (b) and (d) plot the distributions of eigenvalues of the Hessian at three different critical points. Note that the y axes are in logarithmic scale. The vertical lines in (b) and (d) depict the position of 0.

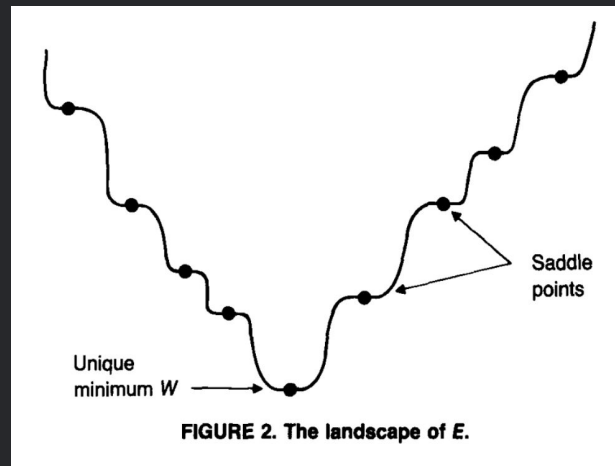
No local minima in theory

In fact, there are nonconvex learning problems where you can show there are *no* local minima and there is only one global minimum

In nonconvex matrix multiplication, [\[Ge et al, 2016\]](#) shows no local minima

For 1-layer MLP, [Baldi and Hornik 1989](#) show that there are only saddle points and no local minima

On the other hand, there are many examples where local minima do exist [\[Yun et al, 2018\]](#)



Fear of saddle points

Now that we are not concerned about local minima, how should we handle saddle points? People have come up with many algorithms to escape saddle points. One general theme is perturbing the model with some noise to escape the saddle region [[Jin et al, 2017](#)]

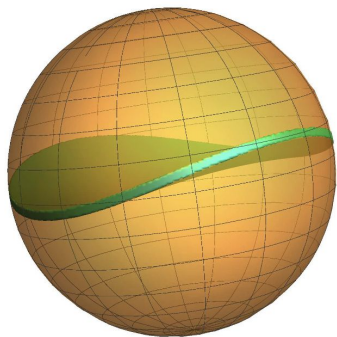


Figure 1: Perturbation ball in 3D and “thin pancake” shape stuck region

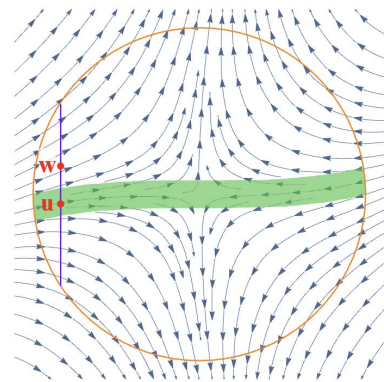


Figure 2: Perturbation ball in 2D and “narrow band” stuck region under gradient flow

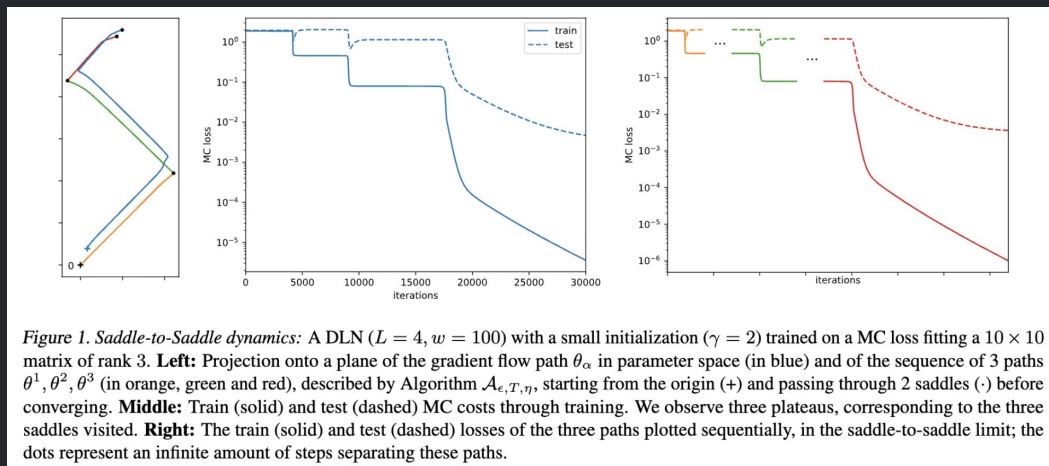
Gradient Descent Converges to Minimizers

Interestingly, there are theoretical statements that show that gradient descent will always converge to a global minima when all saddles are “strict” (have nonzero eigenvalues) [[Lee et al, 2016](#), [Lee et al, 2017](#)]

This suggests that deep learning may not get stuck in saddle points during training though it does not resolve rate of convergence

Saddle-to-Saddle

Another theoretical perspective comes from analyzing how deep *linear* networks evolve over the course of training. In this setting, you can prove that training visits the neighborhoods of a sequence of saddles until finally converging to a global minimizer



Synthesis

Putting all of these mental models together

- There are many more saddle points than local minima (if there are any local minima)
- If local minima exist, they likely have loss close to the global minima
- Gradient descent converges to one of these local minima, unblocked by saddle points

Overall, this set of mental models roughly tells us to chill out when doing gradient descent

Summary of today

Basins suggest that under the right conditions, model can be averaged to achieve stronger models than each of the endpoints

River Valley suggests that optimization proceeds by noisy oscillation, benefitting from eventual learning rate annealing or weight averaging within a trajectory

Critical point analysis suggests that local minima and saddle points are not as scary as naive intuition might suggest

Next week: How does sharpness (maximum eigenvalue of the Hessian) evolve over the course of training and how does neural network training stay stable?