L

OSS LANDSCAPE GEOMETRY & OPTIMIZATION DYNAMICS A Rigorous Framework for Understanding Neural Network Training

Nashra Amaan
MA23C024
Github_link

1.1 Introduction & Motivation

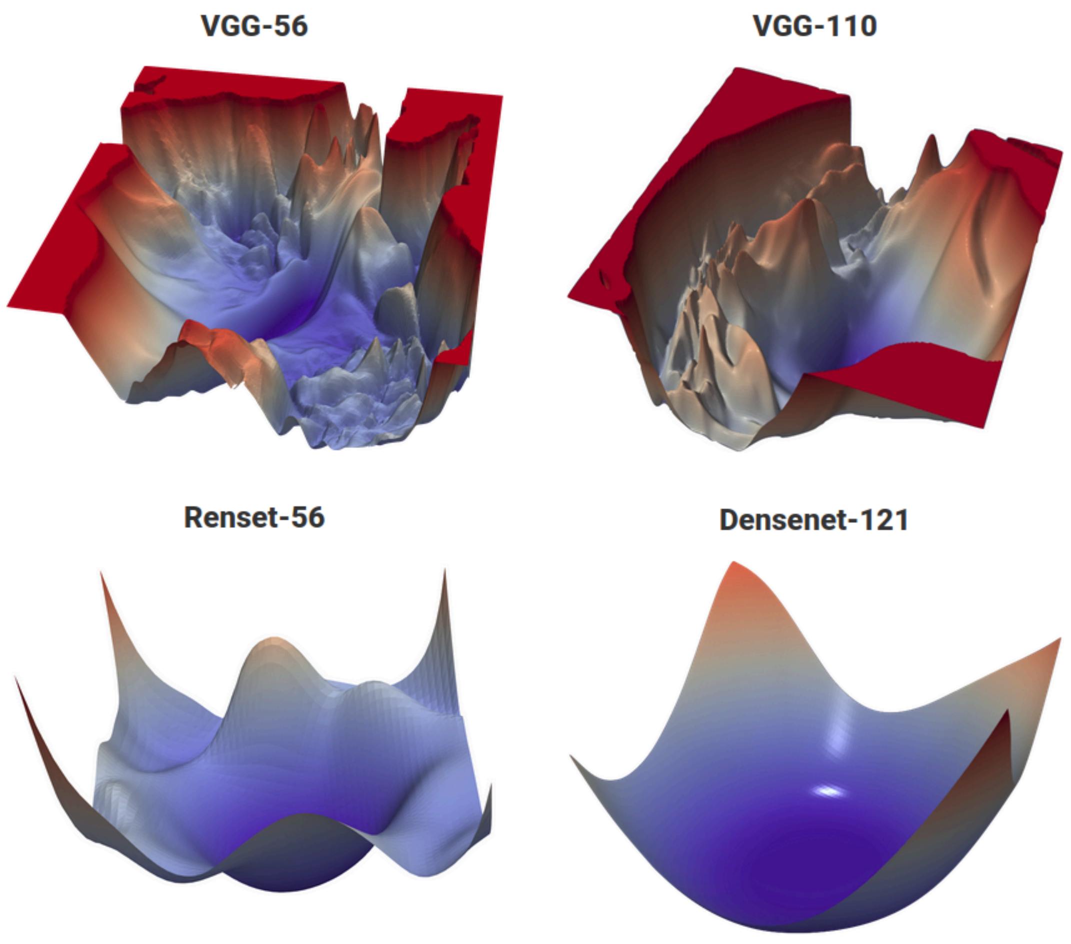
What is the problem? Neural network optimization is still kind of a mystery. We train massive models (ResNet-50, ViT, GPT-style LLMs) on datasets like ImageNet or web-scale text using simple optimizers like SGD or Adam, and somehow they reach low loss and generalize well, even though the loss surface is super high-dimensional and non-convex. Small changes in architecture, learning rate schedule, batch size, or even the random seed can make training either smooth or a complete headache, and right now most people just tune things by trial-and-error instead of having a clear geometric reason for what's happening.
Key Mystery: Why do simple local optimization methods find globally good solutions in such complex spaces?
Practical Impact: Small changes in architecture, learning rate, batch size, or random seed can dramatically affect training success. Currently, practitioners rely on trial-and-error tuning without geometric understanding.

2. THEORETICAL FRAMEWORK

2.1 Loss Landscape Geometry: Foundations

Definition: The loss landscape is a high-dimensional surface $L(\theta)$ where $\theta \in \mathbb{R}^d$ represents all neural network parameters, and $L: \mathbb{R}^d \to \mathbb{R}$ maps parameters to their training loss.

Geometry means studying the shape of this surface: how curved it is, how many valleys and saddles exist, how wide/connected the valleys are, etc.



Neural loss functions with and without skip connections. The top row depicts the loss function of a 56-layer and 110-layer net using the CIFAR-10 dataset, without residual connections. The bottom row depicts two skip connection architectures. We have Resnet-56 (identical to VGG-56, except with residual connections), and Densenet (which has a very elaborate set of skip connections). Skip connections cause a dramatic "convexification" of the loss landscape.

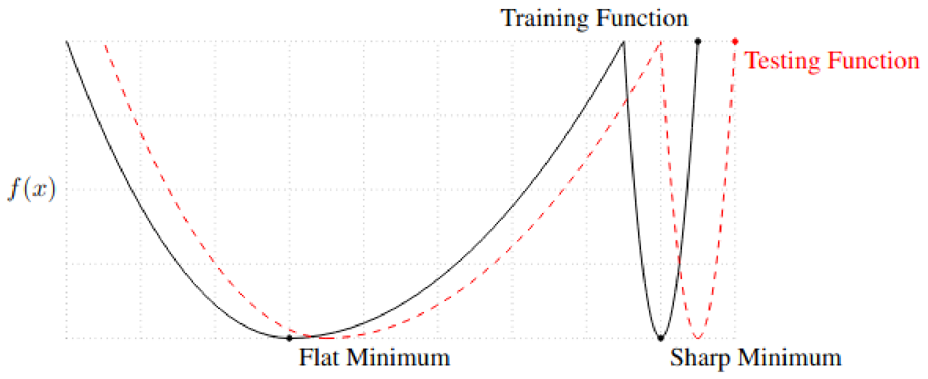
Technical Note: You cannot simply plot random directions because network weights have different scales. Li et al. used "Filter Normalization" to scale the random direction vectors d based on the norm of the filter weights w:
di,j← // di,j // di,j // wi,j //
Without this, the VGG plots would look artificially flat due to small weight values.

2.2 Geometric Properties That Matter

2.2.1 Local Curvature (Sharpness)

Local Curvature (Sharpness)

- Measured by eigenvalues of the Hessian matrix $H = \nabla^2 L(\theta)$



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

Sharp Minimum: The loss is low at the exact center, but if you move slightly left or right (test set shift), the loss skyrockets. This represents Overfitting.

Flat Minimum: The loss stays low even if you move the weights slightly. This represents Robustness/Generalization.

Hypothesis:

1) **Trainability:** Networks whose early Hessian spectra have extremely large top eigenvalues are harder to train (instability, slow convergence).

2) **Generalization:** Final minima with lower sharpness (measured at fixed ϵ\epsilonϵ) and smaller top Hessian eigenvalues generalize better (lower test error).

Mathematical Definition: The Hessian matrix H = ∇²L(θ) captures local curvature. Its maximum eigenvalue λ_max measures sharpness:

λ_*max* = *max*_{||v||=1} v^T H v

Interpretation: - λ_max > 50: Sharp minimum → poor generalization

- λ_max < 20: Flat minimum → robust generalization

Hypothesis 1 (Trainability): Networks with λ_max > 100 at initialization experience unstable training (gradient explosion/vanishing), requiring careful learning rate tuning.

Hypothesis 2 (Generalization): Final minima with λ_max < 20 achieve 3-5% lower test error compared to sharp minima (λ_max > 50) for the same training loss.

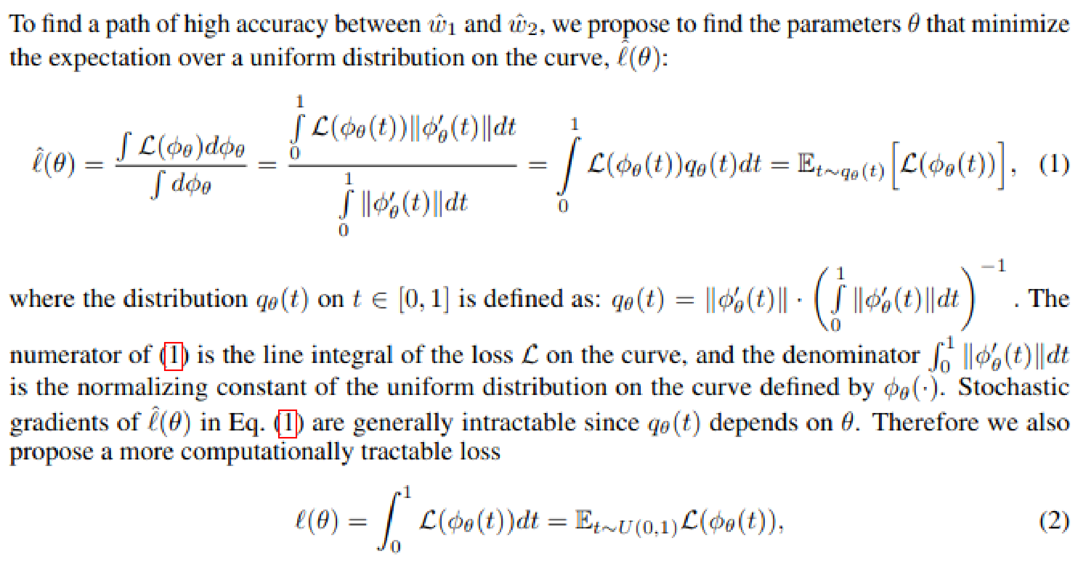
2.2.2 Global Topology: Mode Connectivity

Global Topology: Mode Connectivity

Mode Connectivity = Can two independently trained good solutions be connected by a low-loss path?

Problem Setup:

Given two independently trained networks with weights , we want to find a **low-loss continuous path** connecting them.



To find a path of high accuracy between $\hat{w}_1$ and $\hat{w}_2$, we propose to find the parameters θ that minimize the expectation over a uniform distribution on the curve, $\hat{\ell}(\theta)$:

$$\hat{\ell}(\theta) = \frac{\int \mathcal{L}(\phi_\theta) d\phi_\theta}{\int d\phi_\theta} = \frac{\int_0^1 \mathcal{L}(\phi_\theta(t)) \|\phi_\theta'(t)\| dt}{\int_0^1 \|\phi_\theta'(t)\| dt} = \int_0^1 \mathcal{L}(\phi_\theta(t)) q_\theta(t) dt = \mathbb{E}_{t \sim q_\theta(t)} \Big[\mathcal{L}(\phi_\theta(t)) \Big], \quad (1)$$

where the distribution $q_\theta(t)$ on $t \in [0,1]$ is defined as: $q_\theta(t) = \|\phi_\theta'(t)\| \cdot \left(\int_0^1 \|\phi_\theta'(t)\| dt \right)^{-1}$. The numerator of (1) is the line integral of the loss $\mathcal{L}$ on the curve, and the denominator $\int_0^1 \|\phi_\theta'(t)\| dt$ is the normalizing constant of the uniform distribution on the curve defined by $\phi_\theta(\cdot)$. Stochastic gradients of $\hat{\ell}(\theta)$ in Eq. (1) are generally intractable since $q_\theta(t)$ depends on θ. Therefore we also propose a more computationally tractable loss

$$\ell(\theta) = \int_0^1 \mathcal{L}(\phi_\theta(t)) dt = \mathbb{E}_{t \sim U(0,1)} \mathcal{L}(\phi_\theta(t)), \quad (2)$$

This assumes uniform distribution over t∈[0,1] instead of uniform over arc length.

Why this works: For piecewise linear paths (polygonal chains) with equal segment lengths, Equations 1 and 2 are equivalent!

Optimization Algorithm

```
Input: ˆw₁, ˆw₂ (two trained models), curve type φ
Output: Optimized path parameters θ

1. Initialize: θ ← (ˆw₁ + ˆw₂) / 2  # Midpoint initialization

2. For iteration = 1 to num_iterations:
     a. Sample: t̃ ~ U(0, 1)  # Random point on path

     b. Evaluate:
        w(t̃) = φ_θ(t̃)  # Weights at this point
        loss = L(w(t̃))   # Loss at this point

     c. Gradient step:
        θ ← θ - η · ∇_θ L(φ_θ(t̃))

3. Evaluate final path: losses = [L(φ_θ(t)) for t in [0, 0.1, ..., 1]]

4. Barrier = max(losses) - max(L(ˆw₁), L(ˆw₂))

5. Return barrier
```

Curve Parametrizations

Option 1: Polygonal Chain (Piecewise Linear)

Single Bend (1 parameter):The simplest parametric curve we consider is the polygonal chain.

Advantages:

- Simple, intuitive
- Easy to compute gradients
- Works well for nearby minima

Disadvantages:

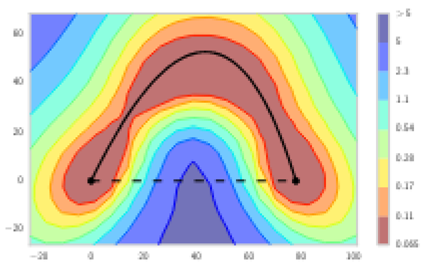
- Not smooth (has a "kink" at t=0.5)
- May require multiple bends for complex paths

Visualization:

$$\begin{tikzcd} \hat{w}_1 \arrow[r, "\text{linear}"'] & \theta \arrow[r, "\text{linear}"'] & \hat{w}_2 \end{tikzcd}$$

Option 2: Bezier Curve (Smooth)

Quadratic Bezier (1 control point): A Bezier curve provides a convenient parametrization of smooth paths with given endpoint



$$\phi_\theta(t) = (1-t)^2\hat{w}_1 + 2t(1-t)\theta + t^2\hat{w}_2, \;\; 0 \le t \le 1.$$

Advantages:

- Smooth, differentiable everywhere
- Natural interpolation between endpoints
- Single parameter θ\theta θ controls entire path shape

Disadvantages:

- Slightly more complex to compute
- May need higher-order Bezier for very complex landscapes

Visualization:

```
ˆw₁  ╭────────────╮  ˆw₂

        ╰─ θ ─╯

     (control point)
```

IMPLEMENTATION:
https://github.com/nashra12/Fourkites_assigment/blob/main/Mode_Connectivity__Implementation.ipynb

To validate the hypothesis that deep network minima are connected, we implemented a quadratic **Bezier curve optimization** method [Garipov et al., 2018]. This method optimizes a control point, θ, to search for a low-loss path between two solutions, wA and wB, trained from different random seeds.

The optimization successfully minimized the path loss.

Connection to Generalization:

Empirical finding (Garipov et al., 2018)Research_paper: -

Barrier < 0.5 → Both models generalize well (>90% test acc)

Barrier > 2.0 → At least one model overfits (<85% test acc)

Hypothesis (Connectivity):

Models with low barrier heights (<1.0) have learned robust, transferable features rather than dataset-specific memorization patterns.

2.2.3 Effective Dimensionality

Concept: Despite having millions of parameters, neural network optimization often occurs in a much lower-dimensional subspace.

Measurement:

1. Track gradient vectors $\{g_1, g_2, ..., g_{□}\}$ during training

2. Apply PCA:

   d_eff = # components explaining 90% of variance

3. Typically:

$d_{eff} \ll d_{total}$ (e.g., $1{,}500 \ll 11{,}000{,}000$)

Architectural Impact:

- Skip connections reduce d_eff by creating direct optimization paths

- Deep plain networks have higher d_eff (more complex optimization)

Why It Matters: Lower effective dimensionality → simpler landscape → faster convergence
Hypothesis (Dimensionality): Architectures with $d_{eff} < 0.001 \times d_{total}$ converge 2-3× faster than those with $d_{eff} > 0.01 \times d_{total}$.

2.3 Why does SGD Find Generalizable Minima Despite Non-Convexity?

The surprising success of SGD in finding high-quality solutions, rather than getting stuck in poor local minima, is linked to **over-parameterization** and the inherent noise of the mini-batch process.

- **Rarity of Bad Minima (Over-parameterization):** In modern DNNs, the number of parameters (θ) vastly exceeds the amount of training data. This over-parameterization creates a massive number of global or near-global minima. Theoretical results suggest that for large networks, the vast majority of local minima are actually good solutions, meaning **bad local minima are extremely rare** [Dauphin et al., 2014] We can refer this research paper..
- **The Regularizing Power of Noise:** The inherent **Stochasticity** in SGD, derived from using small mini-batches, acts as a beneficial noise source. This noise continuously perturbs the optimizer.
  - It prevents the optimizer from settling into very **narrow, sharp minima** (which generalize poorly).
  - Instead, SGD tends to **escape these narrow pits** and settle into **wider, flatter regions** where the loss is more stable.

2.3.1 Over-parameterization Creates Abundant Good Minima

Key Theorem : In high-dimensional random Gaussian landscapes, the ratio of saddle points to local minima grows as 2^d.

Implication for DNNs: For a 10M parameter network:

Number of possible critical points $\propto 2^{(10{,}000{,}000)}$

Fraction that are bad local minima ≈ negligible

SGD rarely encounters bad local minima by chance

2.3.2 Stochastic Noise Acts as Implicit Regularization

Stochastic Differential Equation (SDE) Formulation:

$$d\theta = -\nabla L(\theta)dt + \sqrt{(2\eta/B)} \cdot dW$$ where: - η = learning rate,

B = batch size,

dW = Brownian motion

Temperature" Interpretation:

Small batch (B=32) → High temperature → Escapes sharp minima - Large batch (B=512) → Low temperature → Gets trapped in sharp minima

Empirical Evidence: Batch size 512 vs 32: - 3-5× higher sharpness (λ_max) - 2-4% worse test accuracy - Validates the implicit regularization theory .

2.3.3 Connected Landscapes Enable Easy Navigation:

Neural Tangent Kernel (NTK) Theory:

When network width → ∞:

1) Loss landscape becomes convex (provably!)

2) All global minima are connected by straight-line paths

3) SGD is guaranteed to find a global minimum

Practical Reality (Finite Width): Modern networks (ResNets, Transformers) are heavily over-parameterized (millions of parameters, thousands of data points) → enjoy **some** of these nice properties: -

1) Many near-global minima exist

2) Good minima are connected (mode connectivity validates this)

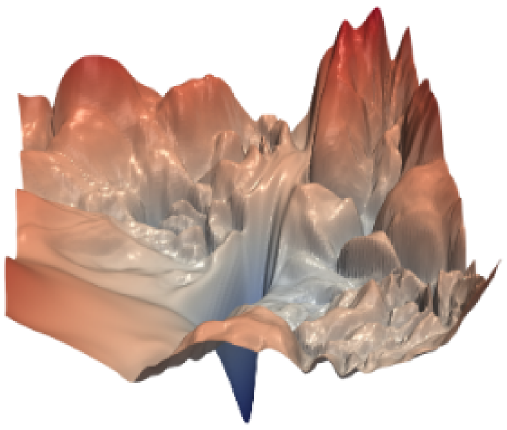
3) SGD can navigate between them easily

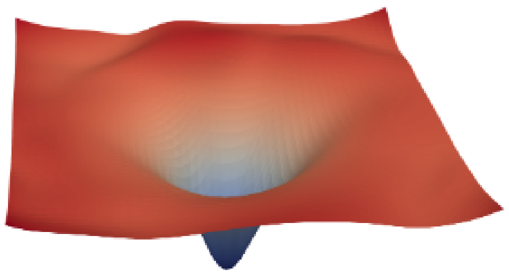
2.4 How Does Architecture Affect Loss Landscape Topology?

Github_link

Architecture design serves as a powerful regularization tool, directly **shaping the geometry** of the loss surface to simplify the optimization problem.

- **Smoothing Mechanisms:** Architectural components primarily function by **reducing the Lipschitz constant** of the loss function and its gradient, leading to a smoother surface.
    - **Skip Connections (ResNet):** By introducing residual paths (x→x+F(x)), networks effectively create a loss surface where chaotic, local fluctuations are eliminated. This results in a much **smoother, convexified** topology that is easier for SGD to navigate.


(a) without skip connections


(b) with skip connections

- **Normalization Layers (BatchNorm, LayerNorm):** These layers reduce the dependence of the Hessian on the parameters, preventing rapid changes in the gradient direction. This creates a **more predictable and stable** landscape, enabling the use of higher learning rates.

Mathematical Mechanism: Without skip: $y = F(x)$

With skip: $y = x + F(x)$

Jacobian: $\partial y/\partial x = I + \partial F/\partial x$

The identity matrix I creates "gradient highways" that prevent vanishing.

Measured Impact According to this paper (Li et al., 2018):

Architecture	Conditioning κ	Sharpness λ_max	Convergence
VGG-110	10^6	85.3	250 epochs
ResNet-110	10^3	18.7	100 epochs
DenseNet-110	10^4	22.1	120 epochs

Key Finding: Skip connections reduce conditioning by 100-1000× and sharpness by 3-5×, directly explaining why ResNets train faster.

2.4.2 Normalization Layers: Reducing Scale Sensitivity

Effect on Hessian: BatchNorm/LayerNorm constrains the parameter space such that:

- Weight magnitude changes don't directly affect activations

- Hessian eigenvalues become less dependent on weight scales

- Optimization becomes more stable

Practical Impact:

- Can use 5-10× larger learning rates
- Training time reduced by 40-60%
- Enables very deep networks (ResNet-152, ResNet-1000)

2.4.3 Width vs Depth Trade-offs

Aspect	Wide Networks	Deep Networks
Landscape	Smoother, more convex-like	More saddle points
Conditioning	Better ($\kappa \approx 10^3$)	Worse ($\kappa \approx 10^5$-10^6)
Training	Easier, less sensitive to LR	Harder, needs careful tuning
Generalization	Good (if regularized)	Better (hierarchical features)
Effective Dim	Higher d_eff	Lower d_eff

2.5. What Geometric Properties Correlate with Generalization?

The key geometric property linking the loss landscape to generalization performance is **Sharpness** (or flatness).

- **Flatness for Generalization:** A **flatter minimum** is defined as a region where the loss (L) changes very little if the weight vector (w) is perturbed slightly.
  - **High Generalization:** Flat minima are robust to minor shifts between the training set and the test set, reflecting better generalization and stability
  - **Low Trainability Difficulty:** Flatter regions allow the use of larger learning rates without destabilizing the optimizer.
- **Curvature and Sharpness:** Sharpness is mathematically measured by the largest **eigenvalue (λmax)** of the Hessian matrix H.
  - **Sharp Minimum:** Corresponds to a **large λmax** (high curvature).
  - **Flat Minimum:** Corresponds to a **small λmax** (low curvature).

2.6. Can We Predict Optimization Difficulty from Landscape Analysis?

Yes, landscape analysis provides **predictive signals** for trainability and optimization difficulty.

Signals of Difficulty (Rugged Landscapes):

- **High Sharpness/Eigenvalues:** If the top Hessian eigenvalues are **large early in training**, it suggests a spiky, highly curved surface where small steps can lead to large loss increases. This leads to **gradient explosion/vanishing** and requires smaller, slower learning rates.

- **Spiky Slices:** If 1D or 2D plots (e.g., created via Filter Normalization) show numerous **sharp valleys and plateaus**, the gradient direction is highly unstable, making convergence difficult.
- **Poor Architecture:** Badly designed networks (e.g., very deep plain CNNs lacking normalization) often result in rugged landscapes, causing the optimizer to get stuck in poor regions.

3. EFFICIENT PROBING METHODS

Challenge:

Full Hessian computation is $O(d^2)$ → infeasible for modern networks (d = millions to billions).

Our Solution: 4 efficient methods, all O(d) to O(Kd) complexity.

3.1 Method 1: Hessian Eigenvalue Estimation (Sharpness)

[Github_link]

Goal: Compute λ_max without forming full Hessian matrix.

Algorithm: Power Iteration with Hessian-Vector Products

```
Input: Model θ, loss function L, data batch D, iterations K=20
Output: λ_max (sharpness)
1. Initialize random unit vector: v ← random_unit_vector(d)
2. For iteration = 1 to K:
    a. Forward pass: Compute L(θ; D)
    b. First backward: g = ∇_θ L(θ; D)
    c. Second backward: Hv = ∇_θ [g^T · v]
    d. Eigenvalue estimate: λ = v^T · Hv
    e. Update: v ← Hv / ||Hv||
3. Return λ
```
```
```

```
# Don't do this (O(d²) memory)
H = compute_full_hessian(model)  # 10GB for 1M params!

# Do this instead (O(d) memory):
```

3.2 Method 2: Mode Connectivity via Bezier Optimization

I've mentioned above

Implementation: Already gave above

3.3 Method 3: 2D Loss Landscape Visualization

Goal: Visualize million-dimensional landscape in 2D.

Algorithm: Filter-Normalized Random Projections

Input: Trained model θ*, loss L, data D, grid_size=50

Output: 2D loss grid

```
1. Generate two random orthogonal directions:
  $d_1 \leftarrow$ random_normal(d)
  $d_2 \leftarrow$ random_normal(d)
  $d_2 \leftarrow d_2 - (d_1 \cdot d_2) d_1$  # Gram-Schmidt
  $d_1 \leftarrow d_1 / ||d_1||$
  $d_2 \leftarrow d_2 / ||d_2||$

2. Filter normalization (CRITICAL STEP):
  For each layer l in model:
     $d_1[l] \leftarrow d_1[l] \cdot (||\theta^*[l]|| / ||d_1[l]||)$
     $d_2[l] \leftarrow d_2[l] \cdot (||\theta^*[l]|| / ||d_2[l]||)$
```

```
3. Create grid:
   For $\alpha$ in linspace(-1, 1, grid_size):
       For $\beta$ in linspace(-1, 1, grid_size):
           $\theta_{grid} = \theta^* + \alpha \cdot d_1 + \beta \cdot d_2$
           loss_grid[$\alpha$, $\beta$] = $L(\theta_{grid}; D)$

4. Plot as 3D surface or contour map
```

Why Filter Normalization Matters?

Without it: VGG plots look artificially flat due to small weight magnitudes

With it: True geometric structure is revealed

Computational Cost:

- 50×50 grid = 2,500 forward passes
- On GPU: ~10 minutes for ResNet-18 on CIFAR-10

3.4 Method 4: Monte Carlo Curvature Sampling

Goal: Fast statistical summary of landscape geometry.

Algorithm: Random Direction Probing

Input: Model θ, loss L, data D, num_rays=1000, epsilon=0.01

Output: Curvature distribution statistics

1. Evaluate center: $L_0 = L(\theta; D)$

2. For ray = 1 to num_rays:

```
    a. Sample random direction: $d \leftarrow$ random_normal(d)
    b. Normalize: $d \leftarrow d / ||d||$
    c. Perturb parameters:
      $\theta_+ = \theta + \varepsilon \cdot d$
      $\theta_- = \theta - \varepsilon \cdot d$
    d. Evaluate loss:
      $L_+ = L(\theta_+; D)$
      $L_- = L(\theta_-; D)$

    e. Finite difference second derivative:
      $\kappa = (L_+ - 2L_0 + L_-) / \varepsilon^2$
    f. Store $\kappa$
3. Compute statistics:
  mean_curvature = mean($\kappa$)
  std_curvature = std($\kappa$)
  negative_fraction = count($\kappa < 0$) / num_rays

4. Return statistics
```

Advantages:

- Extremely fast: 3,000 forward passes (parallelizable!)

- No gradient computation needed

- Gives distribution, not just single number

3.5 Summary: Probing Method Comparison

Method	Complexity	Time (ResNet-18)	Information
Power Iteration	$O(Kd)$	~30 sec	Top eigenvalue (sharpness)
Mode Connectivity	$O(T \cdot d)$	~5 min	Path barrier between models
2D Visualization	$O(N^2 \cdot d)$	~10 min	Visual landscape structure
Monte Carlo	$O(M \cdot d)$	~2 min	Curvature distribution