Controlling the Structure of Inference and Learning in Neural Networks

Yisong Yue

Caltech

https://arxiv.org/abs/2210.10101
Machine learning is transforming science

Halicin: structurally new antibiotic


Personalized Exoskeletons

http://roams.caltech.edu/

https://www.microsoft.com/en-us/research/project/crispr/

AlphaFold
...and autonomous decision-making

 Automated Speech Animation
https://la.disneyresearch.com/publication/deep-learning-speech-animation/

Microsoft Azure Personalizer

AlphaGo
...and creativity

...and common-sense reasoning

**Query:** How many muffins can each kid have for it to be fair?

**Execution**

```python
muffin_patches = image_patch.find("muffin")
len(muffin_patches) = 8
len(kid_patches) = 2
8 // 2 = 4
Result: 4
```
Operationally: What is Machine Learning?

Data: $x$ → Learning Signal: $L_x(\cdot)$

Optimization Problem:

$$\text{argmin}_\theta L(\theta) = \sum_x L_x(f(x; \theta))$$

Profit!
Tuning Neural Networks is Messy and Hard

Initialization, activation, loss, architecture type, depth & width, dropout rate, optimizer, learning rate, momentum, batch size, ...

(Brock et al, 2019)

Heuristic Tuning

HARK Side of Deep Learning - From Grad Student Descent to Automated Machine Learning

Tuning Hyperparameters without Grad Students: Scalable and Robust Bayesian Optimisation with Dragonfly

(Brock et al, 2019)
Canonical Foundational View of ML Optimization

• Typical update rule:

\[ \theta \leftarrow \theta - \eta \nabla L(\theta) \]

(ignoring stochastic aspect, i.e., full batch optimization)
Canonical Foundational View of ML Optimization

Typical update rule:

\[ \theta \leftarrow \theta - \eta \nabla L(\theta) \]

Parameters

- Gradient of Loss w.r.t. parameters
- Learning Rate (tunable hyperparameter)

In Theory:

- Set \( \eta \) via perturbation analysis
- How much \( f \) can change w.r.t. \( \theta \)
- (e.g., global Lipschitz constant of \( f \))

In Practice:

- Re-tune \( \eta \) if we change anything about learning setup!

(ignoring stochastic aspect, i.e., full batch optimization)
Looking Inside a Neural Network

Example:
Linear w/ ReLU activation

\[ f(h; \theta) = [\theta^T h]_+ \]

(ignoring bias/offset)
Intuition (for binary classification)

Sequence of transformations
- Each dimension is a half-space mapping
- Goal: last layer is a separable space with perfect classification
Idea #1: Per-Layer Perturbation Analysis

• How does the layer’s function change under parameter perturbation?

\[
\theta_n \, \text{vs} \, \theta_n + \Delta \theta_n \quad \text{perturbation}
\]

Potential Application: depth- & width-invariant learning rate $\eta$
Idea #2: Control Dynamics of Hidden Layers

**Recall:** we want each layer to push representation towards good answer

**Goal:** control sequence of hidden layers $h_1, \ldots, h_N$
- quickly and robustly converge to low loss
How to exploit the structure of NNs to develop a more nuanced theory?

- **Majorize-Minimize Framework**
  - Per-Layer Perturbation: “Deep Relative Trust”

- **Control-Theoretic Shaping of Neural ODEs**
  - “Lyapunov Loss”
Warm-Up:
Local Perturbation Analysis

- Linear approximation breaks down as $\Delta \theta$ increases!
- Understand rate of break-down via perturbation analysis.

Taylor Expansion:

\[
L(\theta + \Delta \theta) = L(\theta) + \nabla_\theta L(\theta)^T \Delta \theta + \frac{1}{2} \Delta \theta^T \nabla^2_\theta L(\theta) \Delta \theta + \ldots
\]

Linear Approximation

(ignoring stochastic aspect, i.e., full batch optimization)
Majorize-Minimize Framework


Typical Form: \( L(\theta + \Delta \theta) \leq L^{(k)}(\theta + \Delta \theta) + \psi_\theta(\Delta \theta) \)

Order-k Taylor approximation (k=1 for linear)

Upper bound of the rest

Majorization: upper bound on error that lies tangent.

Error \( L(\theta) \)

Minimize majorization \( \Rightarrow \) reduces error.

(ignoring stochastic aspect, i.e., full batch optimization)
Majorize-Minimize in Action

**Example 1:**
\[
\min_{\Delta \theta} \quad \mathcal{L}(\theta) + \frac{1}{2} \lambda |\Delta \theta|^2 \\
\Rightarrow \text{gradient descent.}
\]

**Example 2:**
\[
\min_{\Delta \theta} \quad \mathcal{L}(\theta) + \lambda \mathcal{D}(\theta + \Delta \theta, \theta) \\
\Rightarrow \text{mirror descent.}
\]

**Example 3:**
\[
\min_{\Delta \theta} \quad \mathcal{L}(\theta) + \frac{1}{2} \Delta \theta^T H \Delta \theta + \frac{1}{2} \lambda |\Delta \theta|^3 \\
\Rightarrow \text{cubic regularized Newton.}
\]

(ignoring stochastic aspect, i.e., full batch optimization)
Aside: Duality of Majorization & Trust Regions

\[
\min_{\Delta \theta} \nabla_{\theta} L(\theta)^T \Delta \theta + \lambda |\Delta \theta|^2
\]

- Closed-form solution: \( \Delta \theta = -\left(\frac{2}{\lambda}\right) \nabla_{\theta} L(\theta) \)

- Implies GD update rule: \( \theta \leftarrow \theta - \left(\frac{2}{\lambda}\right) \nabla_{\theta} L(\theta) \)

- Analogous to: \( \min_{\Delta \theta} \nabla_{\theta} L(\theta)^T \Delta \theta \; \text{s.t.} \; |\Delta \theta|^2 \leq C \)

Theoretical guidance via perturbation sensitivity, e.g., Lipschitz constant

Learning Rate

“Trust Region”
Architecture-Naive Majorizations

\[
\min_{\Delta \theta} \nabla_\theta L(\theta)^T \Delta \theta + \lambda |\Delta \theta|^2
\]

\[
\theta \leftarrow \theta - \left( \frac{2}{\lambda} \right) \nabla_\theta L(\theta)
\]

- Can use smaller \( \lambda \) → shallower majorization
- Must use larger \( \lambda \) → steeper majorization

Error

\( L(\theta) \)
Architecture-Naive Majorizations

- Deep networks have complicated optimization landscapes
- Using a single isotropic majorization can be very inefficient!

https://arxiv.org/abs/1712.09913
How to define a **majorization** that **exploits NN structure**?

• **(First) Key Idea**: per-layer perturbation analysis
  
  • Perturb entire layer’s parameters => perturbation of final output

[Image of a neural network with nodes and connections, highlighting the perturbation process through red nodes.]
Thought Experiment

• Vary only n-th layer: $\theta_n + \Delta \theta_n$

• What is the Lipchitz constant of entire function $f$?

• **Depends on parameters $\theta$ of other layers!**
  • If other parameters are larger => perturbation is larger!
Architecture-Aware Perturbation Bounds

Lemmas 5.1 & 6.2 in https://arxiv.org/abs/2210.10101

For $L_2$ loss:

$$L(\theta + \Delta \theta) \leq L^{(1)} + \psi_\theta(\Delta \theta)$$

1st order Taylor

Upper Bound of Rest
Architecture-Aware Perturbation Bounds

Lemmas 5.1 & 6.2 in https://arxiv.org/abs/2210.10101

Overall perturbation sensitivity

\[ \Delta f(\theta) := f(\theta + \Delta \theta) - f(\theta) \]

For \( L_2 \) loss:

\[ L(\theta + \Delta \theta) \leq L^{(1)} + O(|\Delta f(\theta)|_2) + O(|\Delta f(\theta) - \Delta \theta^T \nabla_\theta f(\theta)|_2) \]

\[ \leq C_1 \left[ \prod_{n=1}^N \left( 1 - \frac{|\Delta \theta_n|_*}{|\theta_n|_*} \right) - 1 \right] \]

\[ \leq C_2 \left[ \prod_{n=1}^N \left( 1 - \frac{|\Delta \theta_n|_*}{|\theta_n|_*} \right) - 1 - \sum_{n=1}^N \frac{|\Delta \theta_n|_*}{|\theta_n|_*} \right] \]

“Deep Relative Trust”: \[ \frac{|\Delta \theta_n|_*}{|\theta_n|_*} \]
Majorize-Minimize for Neural Networks

• Derive *majorization* of error

• Plug in *architecture perturbation bound*

• *Minimize* to obtain optimization algorithm
Desiderata & Caveats

\[ L(\theta + \Delta \theta) \leq L^{(1)} + O(|\Delta f(\theta)|_2) + O(|\Delta f(\theta) - \Delta \theta^T \nabla_\theta f(\theta)|_2) \]

\[ \leq C_1 \left( \prod_{n=1}^{N} \left( 1 - \frac{|\Delta \theta_n|_*}{|\theta_n|_*} \right) - 1 \right) \]

\[ \leq C_2 \left( \prod_{n=1}^{N} \left( 1 - \frac{|\Delta \theta_n|_*}{|\theta_n|_*} \right) - 1 - \sum_{n=1}^{N} \frac{|\Delta \theta_n|_*}{|\theta_n|_*} \right) \]

- “Clean” bound only for deep linear networks
  - Formula more complicated with non-linearities
  - First ever analysis even for deep linear networks

- Majorization has no (known) closed-form solution
  - Solving for the optimal \( \Delta \theta \) is itself an optimization problem
First Result

Theorem 6.2 in https://arxiv.org/abs/2210.10101

• **Key restriction:** enforce relative update of each layer to be the same
  • Suboptimal but closed-form update rule

\[
\theta_n \leftarrow \theta_n - \eta \left( \frac{1}{N} \sqrt{\min(\text{dim}_n, \text{dim}_{n-1})} \cdot \frac{|\nabla_{\theta_n} L(\theta)|}{|\nabla_{\theta_n} L(\theta)|} \right)
\]

• **Learning rate** $\eta$ transfers to wider & deeper networks!
  • (Related to mu-Parameterization by Greg Yang et al.)
Initialize Weights:
- for layer $n$ in $\{1, ..., N\}$:
  - $\theta_n \sim \text{unif}(\text{orthogonal}(\text{dim}_n, \text{dim}_{n-1}))$
  - $\theta_n \sim \theta_n \cdot \sqrt{\frac{\text{dim}_n}{\text{dim}_{n-1}}}$

Update Weights:
- $G \leftarrow \frac{1}{N} \sum_{n=1}^{N} |\nabla_{\theta_n} L|_F \cdot \sqrt{\frac{\text{dim}_n}{\text{dim}_{n-1}}}$
- $\eta \leftarrow \log \frac{1+\sqrt{1+4G}}{2}$
- for layer $n$ in $\{1, ..., N\}$:
  - $\theta_n \leftarrow \theta_n - \frac{\eta}{N} \cdot \frac{\nabla_{\theta_n} L}{|\nabla_{\theta_n} L|_F} \cdot \sqrt{\frac{\text{dim}_n}{\text{dim}_{n-1}}}$

$|\theta_n| = \sqrt{\frac{\text{dim}_n}{\text{dim}_{n-1}}}$

$|\Delta \theta_n| = \frac{\eta}{N} \cdot \sqrt{\frac{\text{dim}_n}{\text{dim}_{n-1}}}$

Latest Result

Automatic Gradient Descent

Jeremy Bernstein
Chris Mingard

https://arxiv.org/abs/2304.05187
AGD trains without hyperparameters

https://arxiv.org/abs/2304.05187
AGD trains across width and depth

https://arxiv.org/abs/2304.05187
Different Ways to Design Optimizers

\[ \theta_n \]

<table>
<thead>
<tr>
<th></th>
<th>1.2</th>
<th>-0.5</th>
<th>2.3</th>
<th>0.4</th>
<th>1.3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>0.5</td>
<td>-2.4</td>
<td>0.3</td>
<td>1.0</td>
<td></td>
</tr>
<tr>
<td>0.4</td>
<td>-2.1</td>
<td>-0.2</td>
<td>0.1</td>
<td>0.5</td>
<td></td>
</tr>
<tr>
<td>1.4</td>
<td>-0.2</td>
<td>-1.1</td>
<td>0.1</td>
<td>1.4</td>
<td></td>
</tr>
</tbody>
</table>

Per-layer

Per-neuron

Per-synapse
Different Ways to Design Optimizers

• **Fromage:** Per-layer was our first foray

• **Nero:** Per-neuron
  - Also constrain per-neuron weight norm
  - Connections to batch-norm
  - Connections to generalization

• **Madam:** Per-synapse
  - Also sign-constrain weights
  - Leads to multiplicative update rule
  - Connections to biological synapses
Summary: Architecture-Aware Perturbation Bounds

Architectural perturbation bound

$$\|\Delta f(x)\| \leq C \cdot \left[ \prod_i \left( 1 + \frac{\|\Delta W_i\|}{\|W_i\|} \right) - 1 \right]$$
Optimisation & Generalisation in Networks of Neurons

Thesis by
Jeremy Bernstein

In Partial Fulfillment of the Requirements for the Degree of Doctor of Philosophy

https://arxiv.org/abs/2210.10101
How to exploit the structure of NNs to develop a more nuanced theory? 

Control-Theoretic Shaping of Neural ODEs
“Lyapunov Loss”

Majorize-Minimize Framework
Per-Layer Perturbation: “Deep Relative Trust”
Recall Idea #2: Control Dynamics of Hidden Layers

We want each layer to push representation towards good answer

**Goal:** control sequence of hidden layers $h_1, \ldots, h_N$

- quickly and robustly converge to low loss
Warm Up: Deep Networks

\[
\begin{align*}
\text{input} & \rightarrow f_{in} \rightarrow h_0 \rightarrow f_1 \rightarrow h_1 \rightarrow \cdots \rightarrow f_N \rightarrow h_N \rightarrow f_{out} \rightarrow y \\
\text{hidden states} & \\
\text{output} & \rightarrow y
\end{align*}
\]
Warm Up: ResNets

Functional Form:

\[
\begin{align*}
h_0 &= f_{\text{in}}(x) \\
h_n &= h_{n-1} + f_n(h_{n-1}) \\
y &= f_{\text{out}}(h_N)
\end{align*}
\]
ResNets $\Rightarrow$ Continuous-in-Depth $\Rightarrow$ Neural ODEs

\[
\begin{align*}
\text{"Euler Integration Step Size"} \\
\lim_{{\delta \to 0}} \frac{\partial h}{{\partial t}} &= f(h) \\
\quad &
\end{align*}
\]
Neural Ordinary Differential Equations (NODEs)

$h_0 = f_{in}(x)$  
**Input Layer**

$\frac{\partial h}{\partial t} = f(h, x)$  
**Continuum of hidden layers**  
**“Dynamics”**

$y(t) = f_{out}(h(t))$  
**Output Layer**

WLOG: $t \in [0,1]$  

Comments:

- Forward pass requires ODE solve
- Can evaluate output at any time  
  - E.g., $y(0.5)$
- Includes continuous normalizing flows & other generative models

(“augmented” NODE since $f$ depends on $x$  -- [https://arxiv.org/abs/1904.01681](https://arxiv.org/abs/1904.01681))
Using Control Theory to Shape Learning

- Control can shape dynamical systems (most commonly ODEs)
Why Shape Dynamics of NODEs?

2D state space, **Red Class** is correct
Showing inference trajectories under perturbations

![Diagram showing trajectories and dynamics](image)

- **Dynamics:**
  - Trajectories

- **Unstable NODE**
  - Standard BackProp

- **Our Approach**
  - Lyapunov Loss

Ivan
Jimenez Rodriguez
What dynamics can we shape?

\[ h_0 = f_{in}(x) \]

\[ \frac{\partial h}{\partial t} = f(h, x) \]

\[ y(t) = f_{out}(h(t)) \]

- Full state space: \( h(t) \)?
  - (too unwieldy)

- **Projection**: \( V(h(t)) \equiv L(y(t)) \)
  - Dynamics of the training loss
  - 1-D projection of state space
Exponential Stability

Goal: \( V(h(t)) \leq V(h(0)) e^{-\kappa t} \)

Key invariant: \( \frac{\partial V^T}{\partial h} f(h) \leq -\kappa V(h) \)

Benefits: Fast convergence & Robustness

\( V(t) = ce^{-\kappa t} \)

\( slope = -\kappa ce^{-\kappa t} = -\kappa V(t) \)
Exponential Convergence in Action

LyaNet is trained to optimize for exponential convergence

Measuring Progress via Contraction Condition

Violation: \( \frac{\partial V}{\partial h} f(h) > -\kappa V(h) \)

Satisfaction: \( \frac{\partial V^T}{\partial h} f(h) \leq -\kappa V(h) \)

Contraction Satisfied Everywhere \( \Rightarrow \) Exponential Stability!
Lyapunov Loss

• Point-wise Lyapunov Loss

\[ L_V(x, y, h) \equiv \max \left\{ 0, \frac{\partial V_y^T}{\partial h} f(h) + \kappa V_y(h) \right\} \]

• Lyapunov Loss:

\[ L_V(\theta) \equiv E_{x,y} \left[ \int_0^1 L_V(x, y, h(t)) dt \right] \]

Achieving zero Lyapunov Loss (almost) everywhere implies exponential stability!

1. Interpret training loss as potential function: $V(h(t)) \equiv L(y(t))$

2. Instantiate (point-wise) Lyapunov Loss:

   $$L_V(x, y, h) \equiv \max \left\{ 0, \frac{\partial V_y}{\partial h} f(h) - \kappa V_y(h) \right\}$$

3. Optimize Lyapunov Loss everywhere
Optimization Considerations

\[ L_V(\theta) \equiv E_{x,y} \left[ \int_0^1 L_V(x, y, h(t)) dt \right] \]  

Lyapunov Loss

• Evaluating integral exactly is hard

• Approximate by sampling (simplest is Monte Carlo)
  • Sample \((x,y,h)\) uniformly at random
  • Backprop on point-wise Lyapunov Loss

\[ L_V(x, y, h) \equiv \max \left\{ 0, \frac{\partial V_y^T}{\partial h} f(h) - \kappa V_y(h) \right\} \]  

Point-wise Lyapunov Loss

Benefits of Sampling

• Avoids expensive ODE solve

• Goal is to minimize Lyapunov Loss everywhere

\[ L_V(\theta) \equiv E_{x,y} \left[ \int_0^1 L_V(x, y, h(t)) dt \right] \]

Achieving \( L_V(\theta) = 0 \) under uniform measure implies \( L_V(\theta) = 0 \) in original measure

• Similar idea used in Score-Based Generative Models & Moser Flows

Connection to Control Theory


• V is an **Exponentially-Stabilizing Control Lyapunov Function (ES-CLF)** for a controllable ODE if for all states $h \in H$:

$$\min_{\theta} \left[ \frac{\partial V}{\partial h}^T f(h; \theta) + \kappa V(h) \right] \leq 0$$

“controller”

• Can we find a controller $\theta$ that makes V an ES-CLF?
  • In control, $f$ has uncontrolled dynamics and $\theta$ is low-dim

• Connection between controllability & learnability
  • Can we find a $\theta$ that achieves zero Lyapunov Loss?
  • For NODEs, $f$ is fully controlled and over-parameterized

Equivalent to Lyapunov Loss
Comments & Extensions

- Stabilize to sets rather than points
  - E.g., under adversarial perturbations, can still stabilize to a region of low loss
  - Sets need not be convex

- Combinations of conditions (multiple invariances)

- Other invariances:
  
  Forward-Invariance (never leaves “safe” set)
Motivating Application: Continuous Control

Note: Dynamics of Control System included in Neural ODE

\[
\min_{\theta} \left[ \frac{\partial V^T}{\partial h} f(h; \theta) + \kappa V(h) \right] \leq 0
\]
Neural Gaits

Learn NODE policy to satisfy composition of Barrier conditions
Implies indefinite walking (forward-invariance)

Example Barriers

Torso Angle

Swing Foot

Final Policy
Trained on Refined Model
2 Episodes of Data

https://arxiv.org/abs/2204.08120
Certified Forward-Invariance in NODEs

Certified Robust Forward Invariance (First Ever Result)

Certification
Deterministic Sampling on the level set
+ Formal Verification

Summary:
Control-Theoretic Shaping of Neural ODEs
Towards Structure-Aware Theory of Deep Learning

- Neural Nets are not arbitrary black-box functions
  - Analyzing structure can lead to more nuanced theory
  - Can unlock new connections

Per-Layer Perturbation Analysis

Contraction Analysis
<table>
<thead>
<tr>
<th>Research Area</th>
<th>Title</th>
<th>Authors</th>
<th>Conference/Journal</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>On the distance between two neural networks and the stability of learning</td>
<td>Jeremy Bernstein et al., NeurIPS 2020</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Learning compositional functions via multiplicative weight updates</td>
<td>Jeremy Bernstein et al., NeurIPS 2020</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Learning by Turning: Neural Architecture Aware Optimisation</td>
<td>Yang Liu*, Jeremy Bernstein*, et al., ICML 2021</td>
<td></td>
</tr>
<tr>
<td></td>
<td>LyaNet: A Lyapunov Framework for Training Neural ODEs</td>
<td>Ivan Jimenez Rodriguez et al., ICML 2022</td>
<td></td>
</tr>
<tr>
<td></td>
<td>FI-ODE: Certified and Robust Forward Invariance in Neural ODEs</td>
<td>Yujia Huang*, Ivan Jimenez Rodriguez*, et al., arxiv</td>
<td></td>
</tr>
</tbody>
</table>