MATH 292.01

November 13, 2025

Taehun Cha
(Teaching Fellow)

AIML@K KOREA UNIVERSITY

Contents

- Recap) Gradient Descent
- Vanishing Gradient
- Functional Data Analysis
- Curse of Smoothness
  - We will observe additional difficulty in NN training

Recap) Gradient Descent

- For a function f with input variable x,
  - Gradient Descent: $x' = x - \gamma \nabla_x f$
- For a loss function L with data $\mathcal{D} = \{(x, y)\}_{i=1}^{N}$ and weight w
  - Gradient Descent: $w' = w - \gamma \nabla_w L(x, y)$
  - For multiple data points, simply: $w' = w - \gamma \frac{1}{N} \Sigma_i \nabla_w L(x_i, y_i)$
- Sampling the small portion of data works nicely and efficiently
  - Stochastic Gradient Descent: $w' = w - \gamma \frac{1}{|B|} \Sigma_{i \in B} \nabla_w L(x_i, y_i)$, B: minibatch

Recap) Gradient Descent

- Theoretically, under the following conditions:
  - f is convex, (no saddle, no local minima)
  - f is L-smooth, (it does not 'spike' suddenly)
  - and learning rate γ is small enough, ($< \frac{1}{L}$)
- GD results in $x^* = \arg_x \min f(x)$ with infinite steps
- Does this hold in neural network training?

Recap) Gradient Descent

- In neural network training:
  - f is NOT convex, (bunch of saddle and local minima)
  - we don't know whether f is L-smooth, (Loss can 'spikes' suddenly)
  - (at least) we can set γ is small enough, ($< \frac{1}{L}$)
    - But we don't know L!!
- NN loss landscape is too wild 😱
- But surprisingly, it works nicely in many cases! 🤔

Recap) Gradient Descent

모집일정

2026학년도 전기 대학원 입학전형 일정 및 지원자격

전형별일정

구분		2026학년도 전기 원서접수 일정
내국인 전형	신입학 전형	인터넷 원서접수 2025년 09월 26(금) - 10월 02일(목)
	편입학 전형	
외국인 전형	신입학 전형	인터넷 원서접수 2025년 09월 01(월) - 09월 10일(수)
학·석사연계과정 전형		2025년 12월 1일 - 12월 3일
석·박사통합과정 (진입)전형		2025년 12월 1일 - 12월 3일

Theories exist but under strict conditions…

In other word… many things to discover yet!

IEEE SIGNAL PROCESSING LETTERS, © 2025 IEEE.

Curse of Smoothness in Functional Neural Networks

Taehun Cha and Donghun Lee

Abstract—**Functional neural networks (FNNs) have emerged as powerful tools for modeling complex relationships in functional data, leveraging the flexibility of deep learning to capture non-linear patterns. However, most components of FNNs are directly**

traditional data analysis—such as dimension reduction, regression, and classification—all specifically adapted to preserve the infinite-dimensional nature of functional observations.

Vanishing Gradient

- Assume an L-layer neural network f with
  - Hidden states at l-th layer: $h_l = \sigma(W_{l-1}h_{l-1})$
  - σ: non-linear activation function
  - W_{l-1}: weight matrix of $l-1$-th layer
  - $h_0 = x$, i.e. input data
- We train the NN with a loss function $\mathcal{L}$

Vanishing Gradient

- What we know
  - Train with GD, $W_l \leftarrow W_l - \gamma \nabla_{W_l} \mathcal{L}$
  - Chain rule: $\nabla_{W_l} \mathcal{L} = (\nabla_{h_L} \mathcal{L}) \cdot (\nabla_{h_{L-1}} h_L) \cdot (\dots) \cdot (\nabla_{h_{l+1}} h_{l+2}) \cdot (\nabla_{W_l} h_{l+1})$
    - where $\nabla_{h_{l+1}} h_{l+2} = diag[\sigma'(W_{l+1} h_{l+1})] \cdot W_{l+1}$, σ': derivative of σ
    - and $\nabla_{[W_l]_{i,:}} [h_{l+1}]_i = [\sigma'(W_l h_l)]_i \cdot h_l$,
  - (Big Beautiful) Backpropagation
    - Compute the loss value $\mathcal{L}(x, y; f)$
    - Compute the gradient backward, $\nabla_{h_L} \mathcal{L}$, $\nabla_{h_{L-1}} h_L$, $\nabla_{h_{l+1}} h_{l+2}$, …
    - Compute $\nabla_{W_l} h_{l+1}$
    - Then multiply!

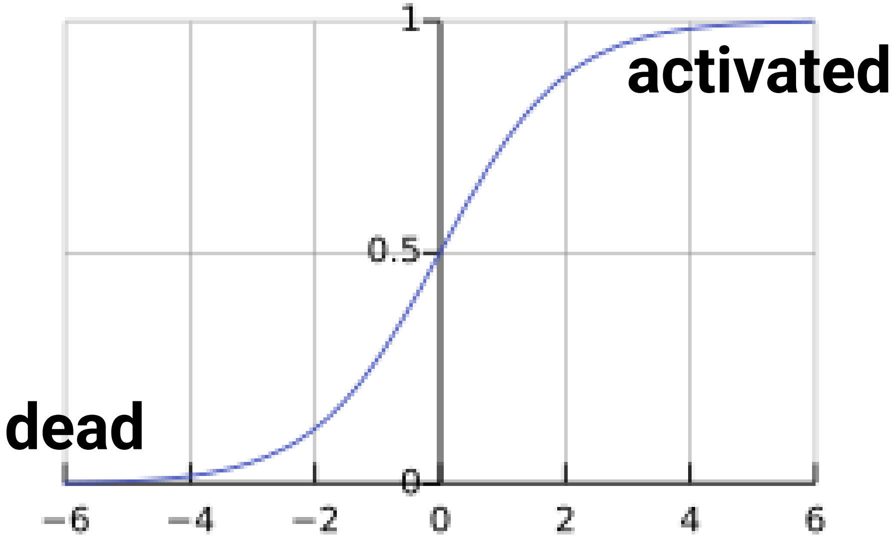
Vanishing Gradient

- Recently, deeeeep NN shows much better performance
  - GPT-4 is estimated to use 120 layers
  - ResNet-152 uses 152 layers
- But the theoretical benefit of depth is not yet established
- Moreover, deeeeep NN suffers from the **vanishing gradient**

Vanishing Gradient

- Suppose σ is Sigmoid function, i.e.

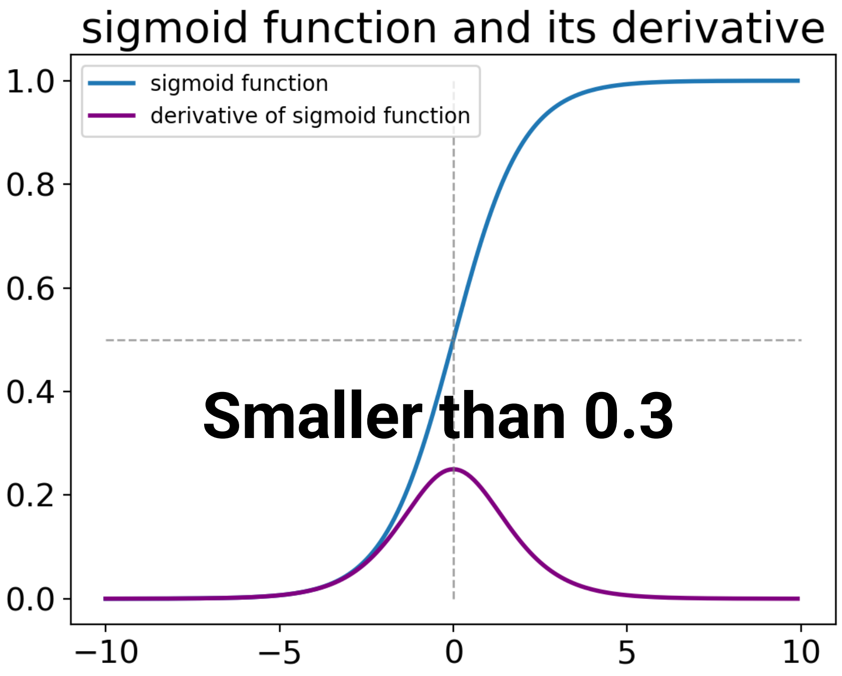
$$\sigma(x) = \frac{1}{1 + e^{-x}}$$



Vanishing Gradient

- What if we draw σ'? i.e.

$$\sigma'(x) = \sigma(x)[1 - \sigma(x)]$$



Smaller than 0.3

and (near) zero for too many part of domain

Vanishing Gradient

- What if we multiply the multiple $\sigma' \in (0, 0.3)$?
  - $0.3^{10} = 0.000006$
- Recall the chain rule,

$$\nabla_{W_l}\mathcal{L} = (\nabla_{h_L}\mathcal{L}) \cdot (\nabla_{h_{L-1}} h_L) \cdot (\dots) \cdot (\nabla_{h_{l+1}} h_{l+2}) \cdot (\nabla_{W_l} h_{l+1})$$

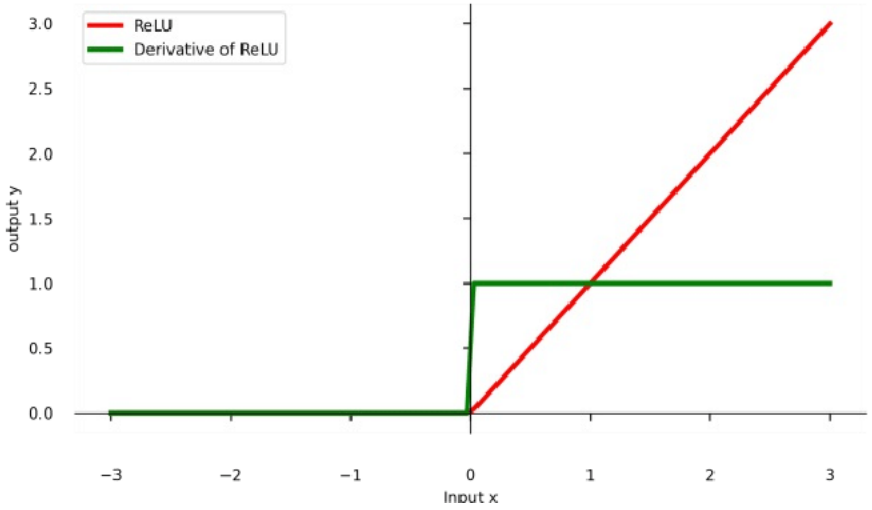
where $\nabla_{h_{l+1}} h_{l+2} = diag\boxed{[\sigma'}(W_{l+1}h_{l+1})] \cdot W_{l+1}$

σ' everywhere!

- As a result, W_l with low l would receive nearly 0 gradient

Vanishing Gradient

- How the modern NN resolved this problem?
  - Activation functions (e.g. ReLU) with higher derivative values



Vanishing Gradient

- How the modern NN resolved this problem?
  - Activation functions (e.g. ReLU) with higher derivative values
  - Residual connection:

$$h_l = \sigma(W_{l-1}h_{l-1}) + h_{l-1}$$

$$\nabla_{h_{l-1}} h_l = diag[\sigma'(W_{l-1}h_{l-1})] \cdot W_{l-1} + I$$

    - ResNet is Res(idual)Net
    - GPT (Transformers) also use this

Functional Data Analysis

- There is a "function"
  - Think it as an infinite-dimensional vector
  - Think of a vector $a = [1, 2, 3]$, where $a_i = i$.
  - Now stretch it two 5 dimension $[1, 1.5, 2, 2.5, 3]$.
  - Stretch it into 9 dimension $[1, 1.25,1.5, 1.75,2, 2.25,2.5, 2.75,3]$.

  …
  - Then we can obtain a vector looks like $f(x) = x, x \in [1,3]$

Functional Data Analysis

- There is a "function"
- We are familiar with finite-dimension data
- But there exists infinite-dimension one
  - Continuous timeseries, electric signal, voice, …
- Classic statistics preprocessed it into a low-dimensional data
- Some statisticians wanted to **preserve** the functional property

Functional Data Analysis

- Basic example: Linear regression
  - What we know: $y = w^{\top}x, x \in \boldsymbol{R}^d$
  - What FDA do: $y = \int \beta(t)x(t)dt$
- Why we need this?
  - If we represent the weight function $\beta(t) = \Sigma_i w_i \phi_i(t)$,
  - where ϕ: basis function (Fourier, Legendre, …)
  - If we use 5 basis, we do LR of inf-dim function with only 5 parameters!

Functional Data Analysis

- LR to Neural Network
  - Recall the multi-target LR: $y = Wx$,
  - Recall the NN: $h_l = \sigma(W_{l-1} h_{l-1})$
  - It can be seen as a recursive LR + non-linear σ

Functional Data Analysis

- Functional LR to Functional Neural Network
  - Recall the functional LR: $y = \int \beta(t)x(t)dt$,
  - (Univariate) FNN: $h_l^k(t) = \sigma\left(\Sigma_j \beta_{l-1}^{j,k}(t) h_{l-1}^j(t)\right)$,
  - (Bivariate) FNN: $h_l^k(t) = \sigma\left(\Sigma_j \int \beta_{l-1}^{j,k}(s,t)\, h_{l-1}(s)ds\right)$,
    - j: previous dimension, k: current dimension
    - and bivariate weight function $\beta(s,t) = \Sigma_i w_i \phi_i(s) w_j \psi_j(t)$
  - It can be seen as a recursive FLR + non-linear σ

Curse of Smoothness

- FDA is proposed to overcome the *Curse of Dimensionality*
  - Input data is infinite-dimension → need infinite weight?
  - Not with FDA!
- But there was another curse… ***Curse of Smoothness***… 😱

Curse of Smoothness

- Initial observation
  - I found FNN works so poorly when it gets deeper
  - When σ: Sigmoid, deeper DNN and FNN perform worse
  - I first thought it's just vanishing gradient

Dataset	Model	Sigmoid			
		2 layer	4 layer	6 layer	Δ Acc.
EEG	DNN	55.44% (1.74)	53.63% (0.96)	46.72% (0.88)	-8.72
	U-FNN	69.98% (2.11)	70.48% (2.26)	65.97% (2.23)	-4.01
	B-FNN	69.68% (1.06)	66.09% (1.77)	29.82% (6.12)	-39.86

Curse of Smoothness

- Initial observation
  - I found FNN works so poorly when it gets deeper
  - When σ: Sigmoid, deeper DNN and FNN perform worse
  - I first thought it's just vanishing gradient
  - But it persists with ReLU!
    - Though DNN recovered!

Dataset	Model	Sigmoid				ReLU			
		2 layer	4 layer	6 layer	Δ Acc.	2 layer	4 layer	6 layer	Δ Acc.
EEG	DNN	55.44% (1.74)	53.63% (0.96)	46.72% (0.88)	-8.72	52.24% (1.18)	53.70% (1.37)	53.62% (1.56)	+1.38
	U-FNN	69.98% (2.11)	70.48% (2.26)	65.97% (2.23)	-4.01	69.99% (1.52)	65.51% (1.33)	65.58% (2.02)	-4.41
	B-FNN	69.68% (1.06)	66.09% (1.77)	29.82% (6.12)	-39.86	68.21% (1.49)	63.93% (2.35)	52.35% (4.70)	-15.86

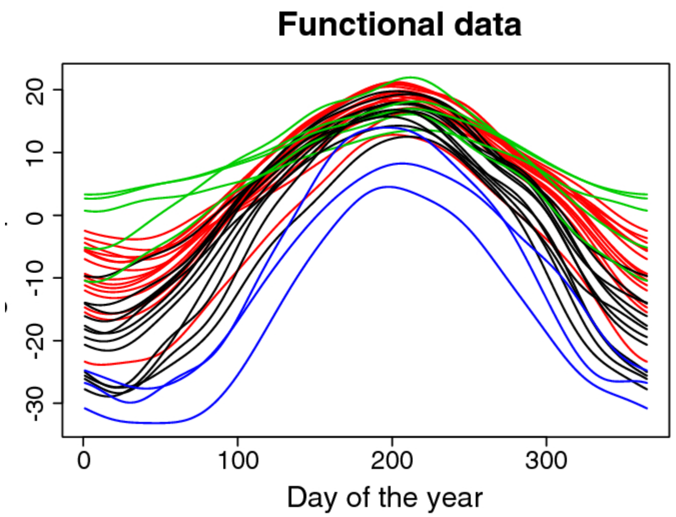
Curse of Smoothness

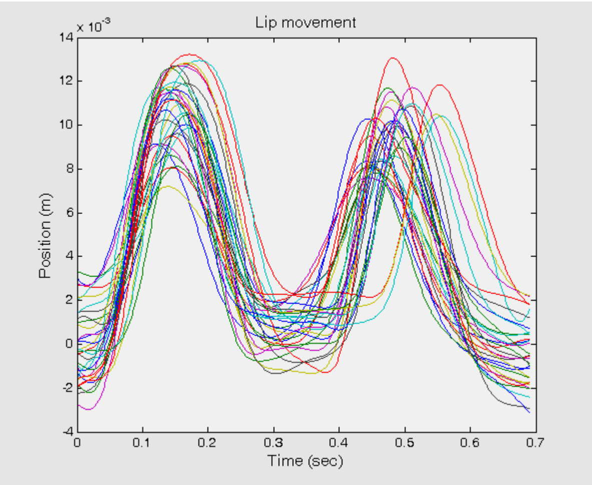
- Initial observation
  - It happens everywhere!
  - Why does it happen?
  - Even with ReLU?

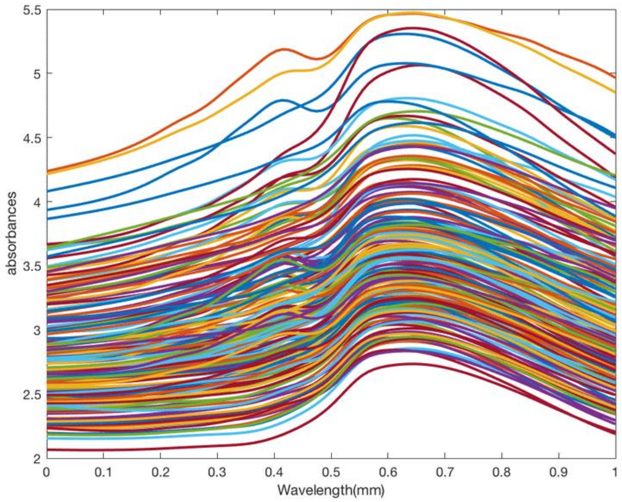
Dataset	Model	Sigmoid				ReLU			
		2 layer	4 layer	6 layer	Δ Acc.	2 layer	4 layer	6 layer	Δ Acc.
Growth	DNN	95.67% (1.63)	94.62% (1.63)	96.72% (1.33)	+1.05	96.72% (0.0)	95.67% (0.0)	94.62% (0.0)	-2.10
	U-FNN	96.72% (1.63)	81.87% (4.52)	51.52% (6.18)	-45.20	96.72% (1.63)	93.56% (5.33)	90.41% (2.50)	-6.31
	B-FNN	95.67% (1.33)	76.55% (7.48)	80.70% (6.18)	-14.97	95.67% (1.33)	86.20% (2.50)	79.71% (4.52)	-15.96
Tecator	DNN	93.95% (0.71)	94.88% (1.69)	93.95% (1.30)	0.0	93.95% (1.59)	91.16% (1.92)	92.56% (2.49)	-1.39
	U-FNN	88.37% (3.71)	66.05% (4.14)	64.19% (4.43)	-24.18	93.95% (0.92)	86.51% (6.61)	78.14% (2.98)	-15.81
	B-FNN	87.44% (2.52)	68.84% (6.28)	65.58% (5.14)	-21.86	93.02% (1.42)	83.72% (5.83)	60.93% (4.67)	-32.09
EEG	DNN	55.44% (1.74)	53.63% (0.96)	46.72% (0.88)	-8.72	52.24% (1.18)	53.70% (1.37)	53.62% (1.56)	+1.38
	U-FNN	69.98% (2.11)	70.48% (2.26)	65.97% (2.23)	-4.01	69.99% (1.52)	65.51% (1.33)	65.58% (2.02)	-4.41
	B-FNN	69.68% (1.06)	66.09% (1.77)	29.82% (6.12)	-39.86	68.21% (1.49)	63.93% (2.35)	52.35% (4.70)	-15.86

Curse of Smoothness

- Hypothesis: may be the functional smoothness is cause
  - FDA is proposed to represent the **smooth** functions



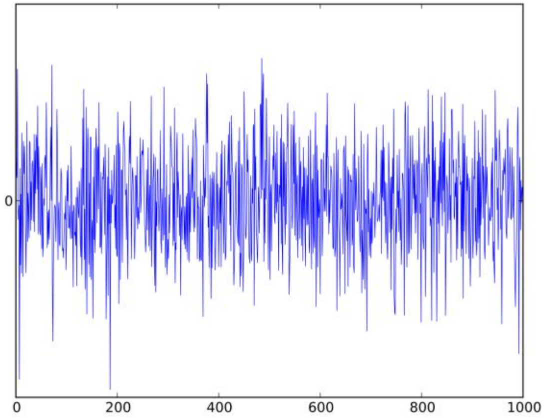


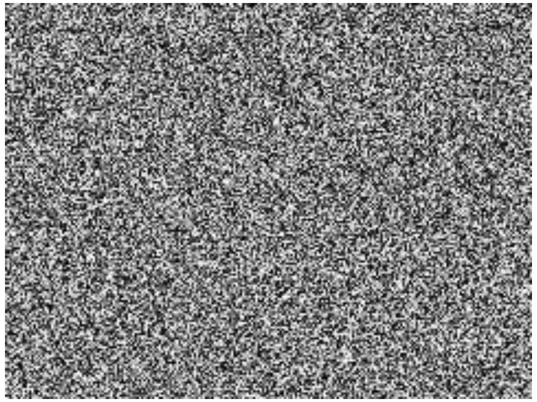


Functional Data Examples

Curse of Smoothness

- Hypothesis: may be the functional smoothness is cause
  - FDA is proposed to represent the **smooth** functions
  - Usually don't say white noise is functional data

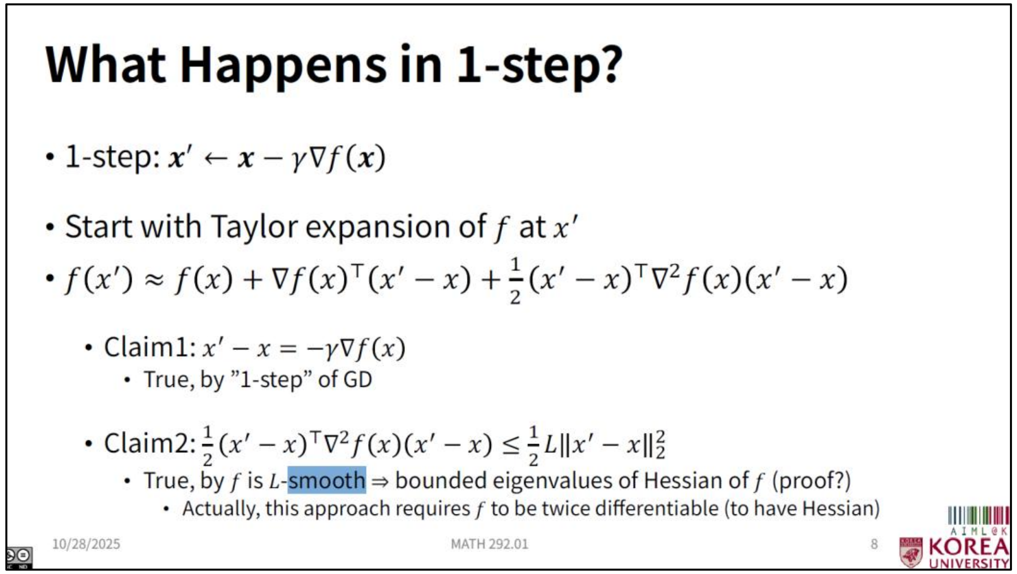




Non-Functional Data Examples

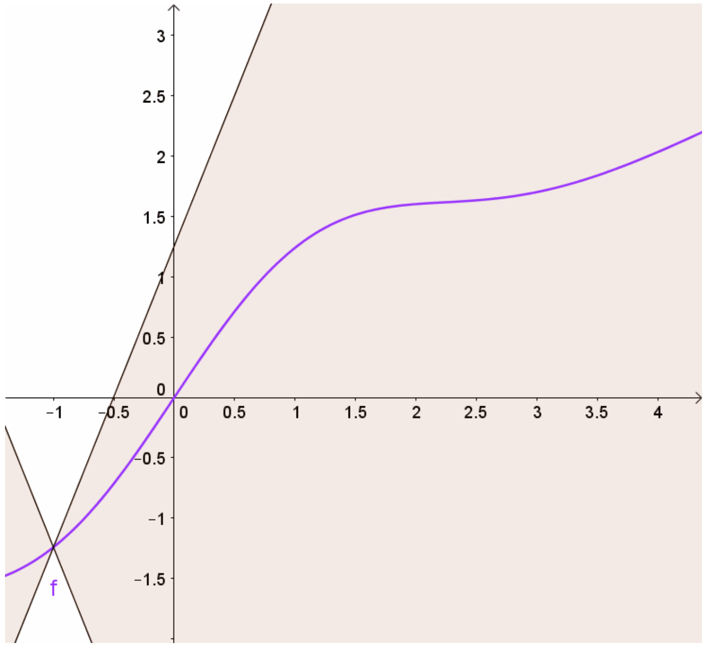
Curse of Smoothness

- Recall, the definition of L-smoothness
  - "f is L-smooth" is equiv. to "∇f is L-Lipschitz"
  - Function $f: \mathcal{X} \to \mathcal{Y}$ is L-Lipschitz if $\forall x', x$: $||f(x') - f(x)|| \leq L||x' - x||$
- Where we used it?



Curse of Smoothness

- We will use the L-Lipschitz on f, not ∇f
  - It's not "L-Smooth", but it also implies some smoothness



Curse of Smoothness

- Settings
  - Let a L-layer functional NN with hidden $h_l(t) = \sigma(\beta_{l-1}\,(t)h_{l-1}(t))$,
  - where $\beta_{l-1}\,(t) = \Sigma_b w_b \phi_b(t)$.
  - It's univariate, single hidden dimension (but can be extended).
- Assume,
  - $h_{l-1}(t)$ and $\phi_i(t)$ are l_1 and l_2-Lipschitz (Smooth),
  - $h_{l-1}(t)$ and $\phi_i(t)$ are zero at some point (Normalization),
  - All functions are in $L^1(0,1)$, $L^2(0,1)$, and $L^4(0,1)$ (Another smooth).

AIML@K KOREA UNIVERSITY

Curse of Smoothness

- Assume,
  - $h_{l-1}(t)$ and $\phi_i(t)$ are l_1 and l_2-Lipschitz (Smooth),
  - $h_{l-1}(t)$ and $\phi_i(t)$ are zero at some point (Normalization),
  - All functions are in $L^1(0,1)$, $L^2(0,1)$, and $L^4(0,1)$ (Another smooth).
- Statement:
  - Then $\left\|\left|\frac{dh_l}{dw_b}\right|\right\|_1 \leq \sqrt{0.2} \cdot \left\|\sigma'\big(\beta_{l-1}(t)h_{l-1}(t)\big)\right\|_2 \cdot l_1 \cdot l_2$

Curse of Smoothness

- Recall the Vanishing Gradient

Vanishing Gradient

- What if we multiply the multiple $\sigma' \in (0, 0.3)$?
  - $0.3^{10} = 0.000006$
- Recall the chain rule,

$$\nabla_{W_l}\mathcal{L} = (\nabla_{h_L}\mathcal{L}) \cdot (\nabla_{h_{L-1}} h_L) \cdot (\dots) \cdot (\nabla_{h_{l+1}} h_{l+2}) \cdot (\nabla_{W_l} h_{l+1})$$

where $\nabla_{h_{l+1}} h_{l+2} = diag[\boxed{\sigma'}(W_{l+1}h_{l+1})] \cdot W_{l+1}$

σ' everywhere!

- As a result, W_l with low l would receive nearly 0 gradient

11/13/2025 MATH 292.01 (TF Session) 13

- Now check the statement again:

If $\sqrt{0.2} \cdot l_1 \cdot l_2 < 1$ …

- Then $\boxed{\left\|\frac{dh_l}{dw_b}\right\|_1} \leq \boxed{\sqrt{0.2}} \cdot \boxed{\left\|\sigma'(\beta_{l-1}(t)h_{l-1}(t))\right\|_2} \cdot \boxed{l_1 \cdot l_2}$

Multiplied L times **Resolved with ReLU**

Curse of Smoothness

- So, the statement,
  - $\left\|\left\|\frac{dh_l}{dw_b}\right\|\right\|_1 \leq \sqrt{0.2} \cdot \left\|\sigma'\left(\beta_{l-1}(t)h_{l-1}(t)\right)\right\|_2 \cdot l_1 \cdot l_2$
- implies gradient norm is bounded by the function smoothness
  - Lower Lipschitz Constant => Smoother
- And it make gradient vanish at deeper layer
- **Curse of Smoothness!!**

Curse of Smoothness

- Cf) Short Proof
  - $\left\|\frac{dh_l}{dw_b}\right\| = \left\|\sigma'\big(\beta_{l-1}(t)h_{l-1}(t)\big) \cdot h_{l-1}(t) \cdot \phi_b(\mathrm{t})\right\|$

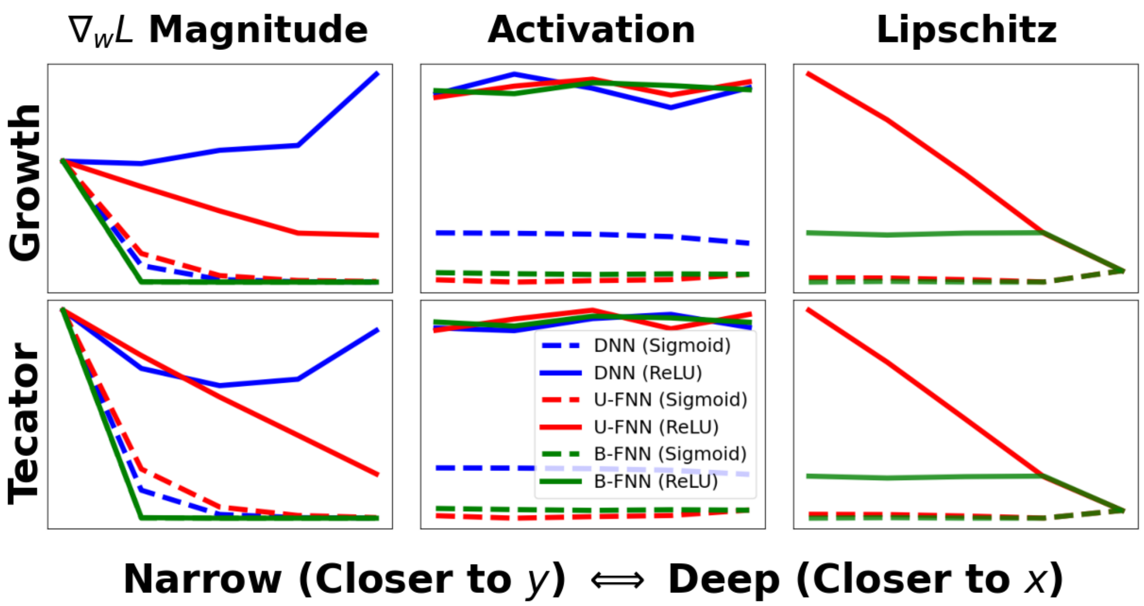
    $\leq \left\|\sigma'^{(\beta_{l-1}(t)h_{l-1}(t))}\right\|_2 \cdot \left\|h_{l-1}(t)\right\|_4 \cdot \left\|\phi_b(t)\right\|_4$ by generalized Holder
  - Note, if a function $g \in L^4(0,1)$ is L-Lipschitz and zero at t', then

    $$|g(t')| = |g(t') - g(t)| \leq L|t' - t|$$
  - and $\left\|\mathrm{g}(t)\right\|_4 = \left[\int_{(0,1)} |g(t)|^4 \, dt\right]^{1/4} \leq L\left[\int_{(0,1)} |t' - t|^4 \, dt\right]^{1/4} \leq L\left[\frac{1}{5}\right]^{1/4}$ (check!)
  - So $\left\|h_{l-1}(t)\right\|_4 \leq l_1 \left[\frac{1}{5}\right]^{1/4}$ and $\left\|\phi_b(t)\right\|_4 \leq l_2 \left[\frac{1}{5}\right]^{1/4}$

■

Curse of Smoothness

- Empirical Results
  - With Sigmoid (dashed line)
    - All models suffer from vanishing grad
    - with low activation gradient σ'
    - + In FNN, smoothness gets severe
  - With ReLU (real line)
    - DNN recovered but FNN couldn't
    - Despite high activation gradient σ',
    - Smoothness hinder it to obtain high grad.



Conclusion

- Optimizing NN with GD is so difficult
  - Suffer from non-convex, non-smooth loss landscape
  - Suffer from **vanishing gradient** (maybe solved)
- Statisticians wanted to overcome the **Curse of Dimensionality**
  - in infinite-dimensional functional data
  - with functional data analysis framework
- However, functional data falls into the **Curse of Smoothness**
  - Which is functional version of **vanishing gradient**

Conclusion

- Optimizing NN with GD is so difficult
  - Suffer from non-convex, non-smooth loss landscape
  - Suffer from **vanishing gradient** (maybe solved) **] We thought it's resolved**
- Statisticians wanted to overcome the **Curse of Dimensionality** **] Different problem appeared**
  - in infinite-dimensional functional data
  - with functional data analysis framework **] Solution proposed**
- However, functional data falls into the **Curse of Smoothness** **] Another Problem showed up**
  - Which is functional version of **vanishing gradient** **] Related to previously resolved issue**