Mathematics of Deep Learning
Lecture 13: GS Implicit Bias, ResNets/Generative Models and ODEs, More on Active Versus Lazy Training, Vanishing & Exploding Gradient

Prof. Predrag R. Jelenković

Time: Tuesday 4:10-6:40pm

Dept. of Electrical Engineering
Columbia University , NY 10027, USA
Office: 812 Schapiro Research Bldg.
Phone: (212) 854-8174
Email: predrag@ee.columbia.edu
URL: http://www.ee.columbia.edu/∼predrag

Implicit Bias of Gradient Descent

- Could it be that we are finding nice generalizable solutions due to GD optimization?
- For linear predictors with linearly separable data, Soudry, Hoffer, and Srebro (2017) show that GD on the cross-entropy loss is implicitly biased towards a maximum margin direction.
    - Bias of GD towards margin maximization means that gradient descent "prefers" a solution that is likely to generalize well, and not just achieve low empirical risk.
- The preceding work inspired many other results, e.g.: Ji and Telgarsky 2019+; Gunasekar et al. 2018; Lyu and Li 2019; Chizat and Bach 2020; Ji et al. 2020.
- In this lecture, I'll cover some of the results from Chapter 15 in Telgarsky, 2021.

Implicit Margin Maximization of Gradient Descent

- Consider n data points: $(y_i, x_i), 1 \le i \le n, y_i \in \{-1, 1\}, x_i \in \mathbb{R}^d$ (For convenience, assume $x_i \equiv \tilde{x}_i = (1, x_{i1}, \ldots, x_{i,d-1})$)
- Assume that data points are *linearly separable*, i.e., there exists w, such that

$$\min_i y_i \langle w, x_i \rangle > 0$$

- Typically, there are many such w-s. Which one is the best in terms of generalization?
- Maximum margin classifier is given by $\hat{y}(x) = \operatorname{sign}\langle w^*, x \rangle$, where

$$w^* = \operatorname*{argmax}_{\|w\|_2 = 1} \min_i y_i \langle w, x_i \rangle$$

  and the margin is $\gamma := \min_i y_i \langle w^*, x_i \rangle$.
- What is the geometric interpretation of this classifier? Why do we expect it to generalize well? (This is the basis of Support Vector Machines.)
- It turns out that under the appropriate conditions, gradient descent converges to the maximum margin classifier.

Implicit Margin Maximization of Gradient Descent

- Consider now replacing $\langle w, x_i \rangle$ with $f(x_i; w)$, and margin mapping

$$m_i(w) := y_i f(x_i; w),$$

where $f(x)$ is locally-Lipschitz and L-homogeneous, i.e., $f(cx) = c^L f(x), c \geq 0$.

- For $\ell(z) = e^{-z}$, let $\mathcal{L}$ be unnormalized loss (no division by n)

$$\mathcal{L}(w) := \sum_{i=1}^{n} \ell(m_i(w)) = \sum_{i=1}^{n} \ell(y_i f(x_i; w)).$$

- Next, we say that data is m-separable it there is a w, such that $\min_i m_i(w) > 0$.
- Now, define the (general) margin, maximum margin and smooth margin, respectively as

$$\gamma(w) := \min_i m_i(w/\|w\|) = \frac{\min_i m_i(w)}{\|w\|^L}, \ \bar{\gamma} := \max_{\|w\|=1} \gamma(w), \ \tilde{\gamma} := \frac{\ell^{-1}(\mathcal{L}(w))}{\|w\|^L}.$$

Not that the motivation for smooth margin comes from (recall ℓ is exponential):
$\ell^{-1}(\mathcal{L}(w)/n) \geq \min_i m_i(w) \geq \ell^{-1}(\max_i \ell(m_i(w))) \geq \ell^{-1}(\mathcal{L}(w))$

Implicit Margin Maximization of Gradient Descent

Note: Gradient descent is biased towards larger margins, which guarantee good generalization.

Theorem (15.1 in Telgarsky, 2021) Consider the linear case, with linearly separable data, exponential loss, $\ell(z) = e^{-z}$, and $max_i \|x_i\| \leq 1$. Then, for gradient descent path w_t with $w(0) = 0$,

$$\gamma(w_t) \geq \tilde{\gamma}(w_t) \geq \gamma - \frac{\ln n}{\ln t + \ln(2n\gamma^2) - \ln\ln(2tne\gamma^2)}$$

Note: In other words, among all possible classifiers/separating hyperplanes, GD converges to the maximum margin classifier, which generalizes well.

Implicit Margin Maximization of Gradient Descent

Few remarks:

- The rate of convergence is $1/\ln(t)$, which can be accelerated by rescaling time, Chizat and Bach (2020): see Theorem 10.2 in Telgarsky, 2021.
- Extension to nonlinear homogeneous functions can be found Theorem 10.3 in Telgarsky, 2021. The theorem is originally due to Lyu and Li (2019).
- Extensions to NNs under restrictions can be found in the aforementioned papers:
  - The Implicit Bias of Gradient Descent on Separable Data, Soudry et al., 2017.
  - Gradient Descent Maximizes the Margin of Homogeneous Neural Networks, Lyu, Kaifeng, and Jian Li, 2019.
  - Implicit Bias of Gradient Descent for Wide Two-Layer Neural Networks Trained with the Logistic Loss, Chizat and Bach, 2020.
  - For additional references check the follow up references on Google Scholar, and Chapter 10 in Telgarsky 2021.

Recent Result by Barron: L_∞ Approximation

The theorem is explicitly stated in terms of bias constants b_k.

Theorem 1 (Klusowski & Barron, 2018) Suppose that f admits an integral representation, i.e., what we call Barron function,

$$f(x) = \nu \int_{[0,1]\times\{w:\|w\|_1=1\}} a(w,b)(w^\top x - b)^+ dP(b,w)$$

where $x \in [-1,1]^d$, $a(w,b) \in \{\pm 1\}$, and P is the probability measure on $[0,1] \times \{w \in \mathbb{R}^d : \|w\|_1 = 1\}$. Then, there exists a neural network function

$$f_m(x) = \frac{\nu}{m} \sum_{k=1}^{m} a_k (w_k^\top x - b_k)^+$$

with $a_k \in [-1,1], \|w_k\| = 1, 0 \le b_k \le 1$ such that for some universal c

$$\sup_{x\in[-1,1]^d} |f(x) - f_m(x)| \le c\sqrt{d + \log(m)} m^{-1/2-1/d}$$

Note: no curse of dimensionality since it assumes a smaller class of functions.

Deep Residual Networks

Proposed by: Deep Residual Learning for Image Recognition, He et al., 2015.

- Easier to optimize at larger depth: scale to 100+ depth
- Reduces the problem of vanishing/exploding gradients
- Hidden state transformations

$$\boldsymbol{h}_{t+1} = \boldsymbol{h}_t + f(\boldsymbol{h}_t, \theta_t), \tag{1}$$

$t \in \{0, \ldots, T\}$, $\boldsymbol{h}_t \in \mathbb{R}^d$

- $\boldsymbol{h}_0$ is the input layer and $\boldsymbol{h}_T$ is the output layer

$$\boldsymbol{h}_T = \boldsymbol{h}_0 + \sum_{t=0}^{T-1} f(\boldsymbol{h}_t, \theta_t)$$

The function is additive and the gradient should behave better

Continuous Approximation to ResNets

Neural Ordinary Differential Equations, by Chen et al., 2018.

- Approximate Equation (1) by a differential equation

$$\frac{d\boldsymbol{h}(t)}{dt} = f(\boldsymbol{h}(t), t, \theta)$$

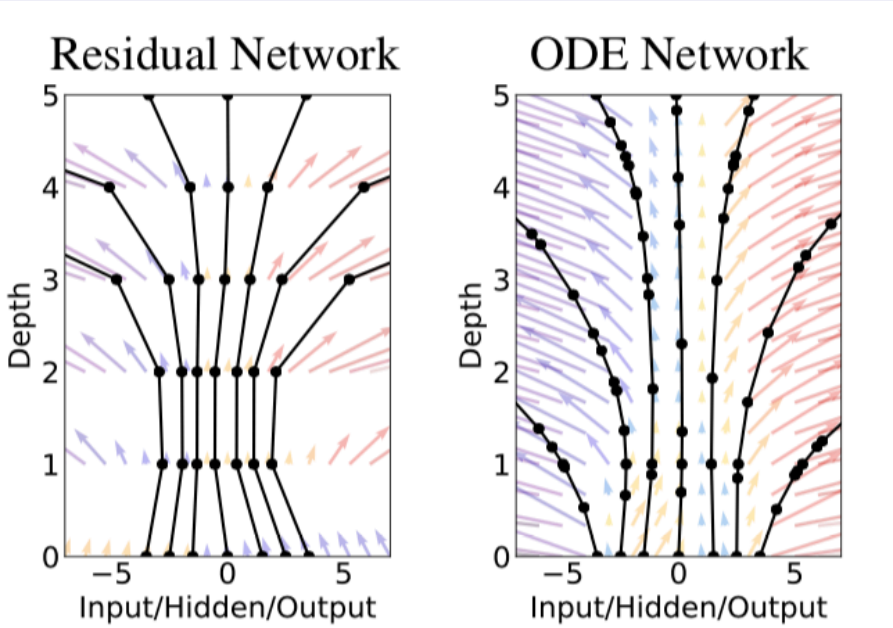


Figure 1: *Left:* A Residual network defines a discrete sequence of finite transformations. *Right:* A ODE network defines a vector field, which continuously transforms the state. *Both:* Circles represent evaluation locations.

Continuous Approximation

Claimed advantages

- Can use standard ODE sovers
- Memory efficiency
- Solving ODEs well understood: over 120 years of experience
- Parameter efficiency
- Continuous transformations and change of variables
- Continuous time series models: can incorporate data that arrives at arbitrary times

Computing the Gradient

- Instead of backpropagation, compute the gradient by solving another (adjoint) ODE backward in time
- Consider optimizing a scalar loss $L(\cdot)$, whose input is the result of an ODE solver

$$L(\boldsymbol{z}(t_1)) = L\left(\boldsymbol{z}(t_0) + \int_{t_0}^{t_1} f(\boldsymbol{z}(t), t, \theta)dt\right)$$

- To optimize L, we need a gradient with respect to θ
- So, we first compute the *adjoint*

$$\boldsymbol{a}(t) = \frac{\partial L}{\partial z(t)}$$

- $\boldsymbol{a}(t)$ dynamics is given by another ODE (an instantaneous analog of the chain rule)

$$\frac{d\boldsymbol{a}(t)}{dt} = -\boldsymbol{a}(t)^\top \frac{\partial f(z(t), t, \theta)}{\partial z(t)}$$

Computing the Gradient

- $\boldsymbol{a}(t)$ dynamics is given by another ODE (an instantaneous analog of the chain rule)

$$\frac{d\boldsymbol{a}(t)}{dt} = -\boldsymbol{a}(t)^\top \frac{\partial f(z(t), t, \theta)}{\partial z(t)}$$

- We can compute $\boldsymbol{a}(t) = \partial L / \partial z(t)$ by solving the preceding equation backward in time starting with the initial value $\boldsymbol{a}(t_1) = \partial L / \partial z(t_1)$
- Complication: need to know $z(t)$ along its entire trajectory
  - Recompute $z(t)$ backward in time, together with the adjoint
- Finally, compute the gradient by evaluating the integral

$$\frac{dL}{d\theta} = -\int_{t_1}^{t_0} \boldsymbol{a}(t)^\top \frac{\partial f(z(t), t, \theta)}{\partial z(t)} dt$$

Generative Models and ODE

FFJORD: FREE-FORM CONTINUOUS DYNAMICS FOR SCALABLE REVERSIBLE GENERATIVE MODELS, by Grathwohl et al., 2019.

- Generative models:
  model distribution/density
- Difficult in high dimensions
- Same authors as Neural ODE
  propose ODEs for generative modeling
- Use continuous dynamics
  to transform simple distribution
  e.g., Gaussian, into a complicated one that generates data

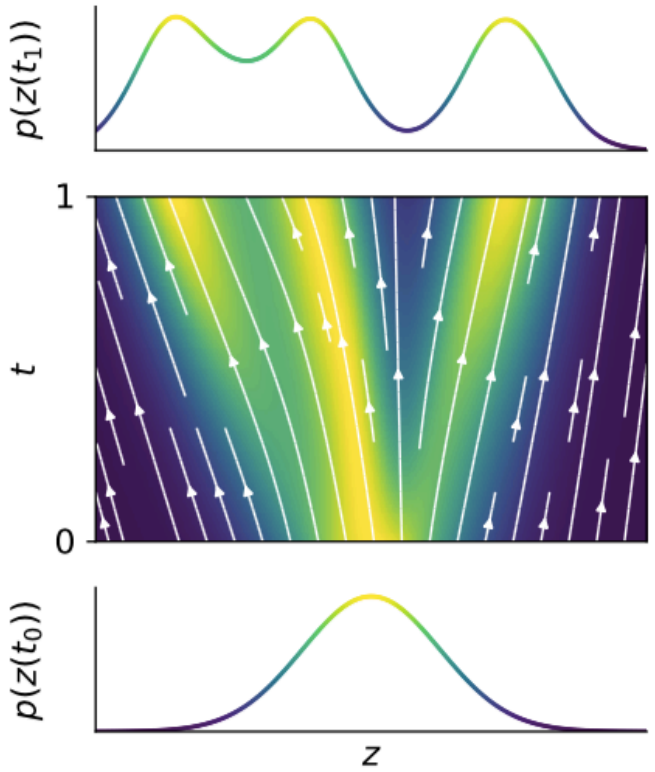


Figure 1: FFJORD transforms a simple base distribution at t_0 into the target distribution at t_1 by integrating over learned continuous dynamics.

Normalizing Flow

Variational Inference with Normalizing Flows, by Resende & Mohamed, 2015.

- General idea: start with an initial random variable with a relatively simple distribution with known (and computationally cheap) probability density function, say Gaussian
- Then, apply a chain of invertible parameterized transformations $\boldsymbol{f}_t$,such that the last iterate z_T has a more flexible distribution:

$$z_0 \sim q(z_0|x), \quad z_t = f_t(z_{t-1}, x), \quad t = 1, \ldots, T$$

- As long as the Jacobian of each transformation can be computed, we can compute the probability density function of the last iterate:

$$\log q(z_T|x) = \log q(z_0|x) - \sum_{t=1}^{T} \log \det \left| \frac{dz_t}{dz_{t-1}} \right|$$

- Resende & Mohamed, 2015, use

$$z_t = f(z_{t-1}) = z_{t-1} + uh(w^\top z_{t-1} + b)$$

where u, w are vectors, b is scalar, and h is a nonlinearity.

Continuous Normalizing Flow

- Chen et al. (2018) and Grathwohl et al. (2019) propose continuous normalizing flow which satisfies

$$\frac{\partial \log p(\boldsymbol{z}(t))}{\partial t} = -\text{Trace}\left(\frac{\partial f}{\partial \boldsymbol{z}(t)}\right)$$

- After integration, the total change in log-density is

$$\log p(\boldsymbol{z}(t_1)) = \log p(\boldsymbol{z}(t_0)) - \int_{t_0}^{t_1} \text{Trace}\left(\frac{\partial f}{\partial \boldsymbol{z}(t)}\right) dt$$

- The main advantage is that computing the Trace is computationally cheaper than evaluating the determinant of the Jacobian $|\partial f/\partial z|$

Active/Feature Versus Lazy Training

- A considerable number of recent papers investigate the differences between "feature (active) training", e.g., mean field regime where weights move considerably, versus "lazy training", e.g., NTK, where weights stay close to the initial value
- Recent numerical paper:
  - Disentangling feature and lazy training in deep neural networks, M. Geiger, S. Spigler, A. Jacot and M. Wyart, Oct 2020.
- Investigates numerically the transition from "lazy" to "feature" training for **deep** networks
- Please take a close look at this paper, especially if your project is on this issue.

Acitve Versus Lazy Training

Some notation:

- Output function

$$f(w,x) = \frac{1}{m}\sum_{i=1}^{m} a_i \sigma(w_i \cdot x + b_i)$$

$w = \cup_i (a_i, w_i, b_i)$; activation soft-plus: $\sigma(x) = (a/\beta)\ln(1+e^{\beta x})$, $(a \approx 1.404, \beta = 5)$. (In the paper $m = h$)

- Corresponding NTK

$$\Theta(w, x_1, x_2) = \nabla_w f(w, x_1) \cdot \nabla_w f(w, x_2)$$

- Empirically learn the following model:

$$F(w,x) := \alpha(f(w,x) - f(w_0,x));$$

- Loss: soft-hinge ($\ell(f,y) = sp_\beta(1-fy)$, $sp_\beta(x)$ is soft-plus)

$$L(w) = \frac{1}{\alpha^2 n}\sum_{i=1}^{n} \ell(F(w,x_i),y_i) \approx \frac{1}{n}\sum_{i=1}^{n} \ell(F(w,x_i)/\alpha^2, y_i)$$

$\alpha = O(m^{-1/2})$ - mean-field, and $\alpha = O(1)$ - NTK, where

$$F(w,x) \approx \alpha \nabla_w f(w_0,x) \cdot (w - w_0)$$

Active Versus Lazy Training

Architecture:

- $x \in \mathbb{R}^d$, with$= m$, depth$= L$
- Preactivations $\tilde{z}^\ell$ and activations z^ℓ with $\tilde{z}^1 = W^0 x/\sqrt{d}$, $z^1 = \sigma(\tilde{z}^1)$, and for $1 \leq \ell < L$

$$\tilde{z}^{\ell+1} = m^{-1/2} W^\ell z^\ell$$
$$z^{\ell+1} = \sigma(\tilde{z}^{\ell+1})$$
$$f(w, x) = m^{-1/2} W^L z^L,$$

  where $L = 3$ is fixed and w is the union of all parameters W^ℓ
- Gaussian initialization $W^\ell_{ij} \sim \mathcal{N}(0, 1)$ and m is varied in simulation

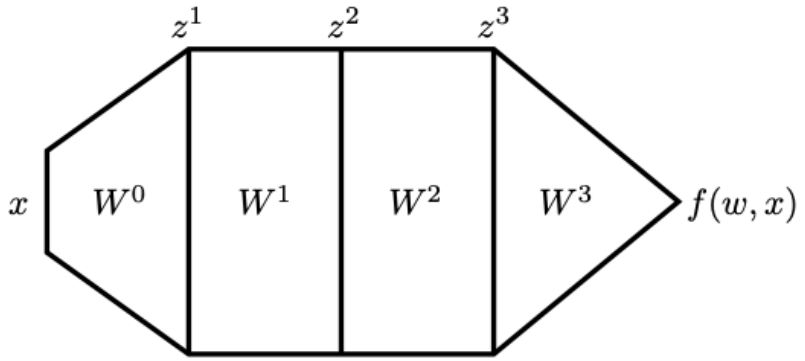


Figure 2: Fully-connected architecture with $L = 3$ hidden layers.

Acitve/Feature Versus Lazy Training

Study the performance in the (α, m) plane

Key observations:

- In the (α, m) plane, the two regimes, feature versus lazy, are separated by
$$\alpha^* = O(m^{-1/2})$$
- The fluctuations of the output, δF, decay with width as (recall $F(w, x) := \alpha(f(w, x) - f(w_0, x))$)
$$\mathrm{Var}(F) \sim \frac{O(1)}{m} \Rightarrow \delta F = O(m^{-1/2})$$
- In the feature/active training there exists a characteristic time scale $t_c \sim \alpha\sqrt{m}$, such that
  - For $t \ll t_c$, the dynamics is linear
  - At $t \sim t_c$, the output grows by a factor of $\sqrt{m}$ and the tangent kernel $\|\Delta\Theta\|$ varies significantly: $\|\Delta\Theta\| \sim (\alpha\sqrt{m})^{-a}, a < 2$.

Feature Versus Lazy Training

In the paper $h \equiv m =$ network width

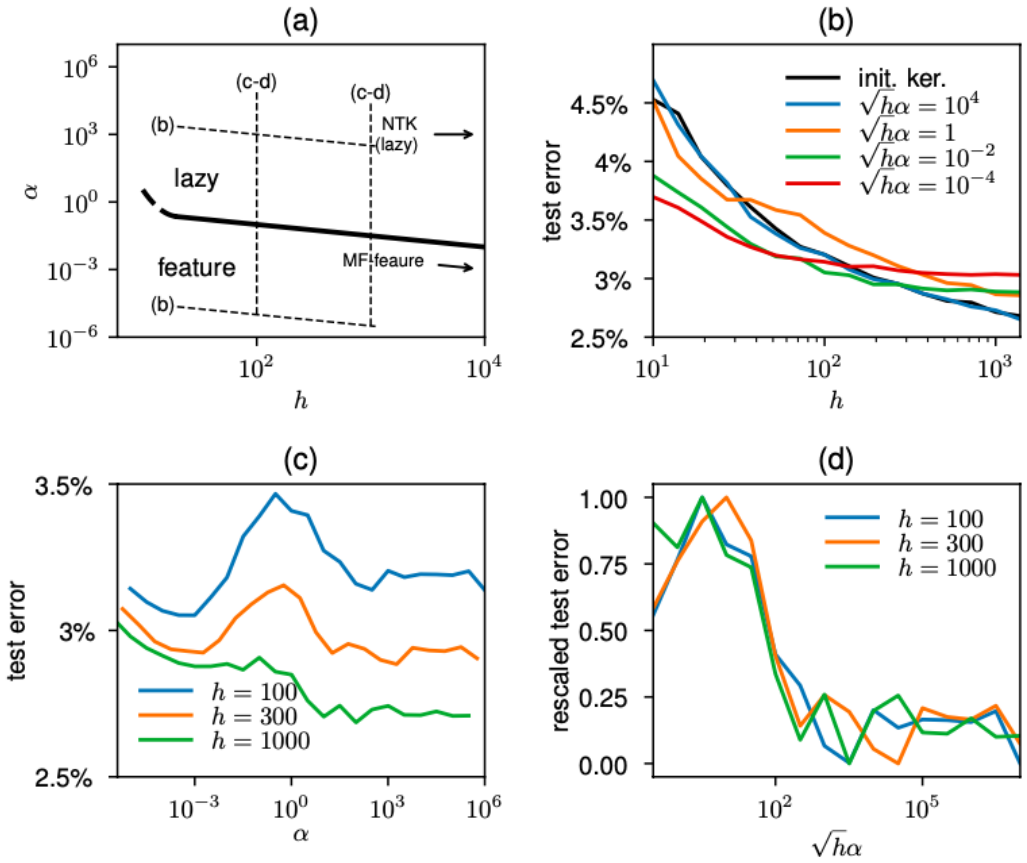
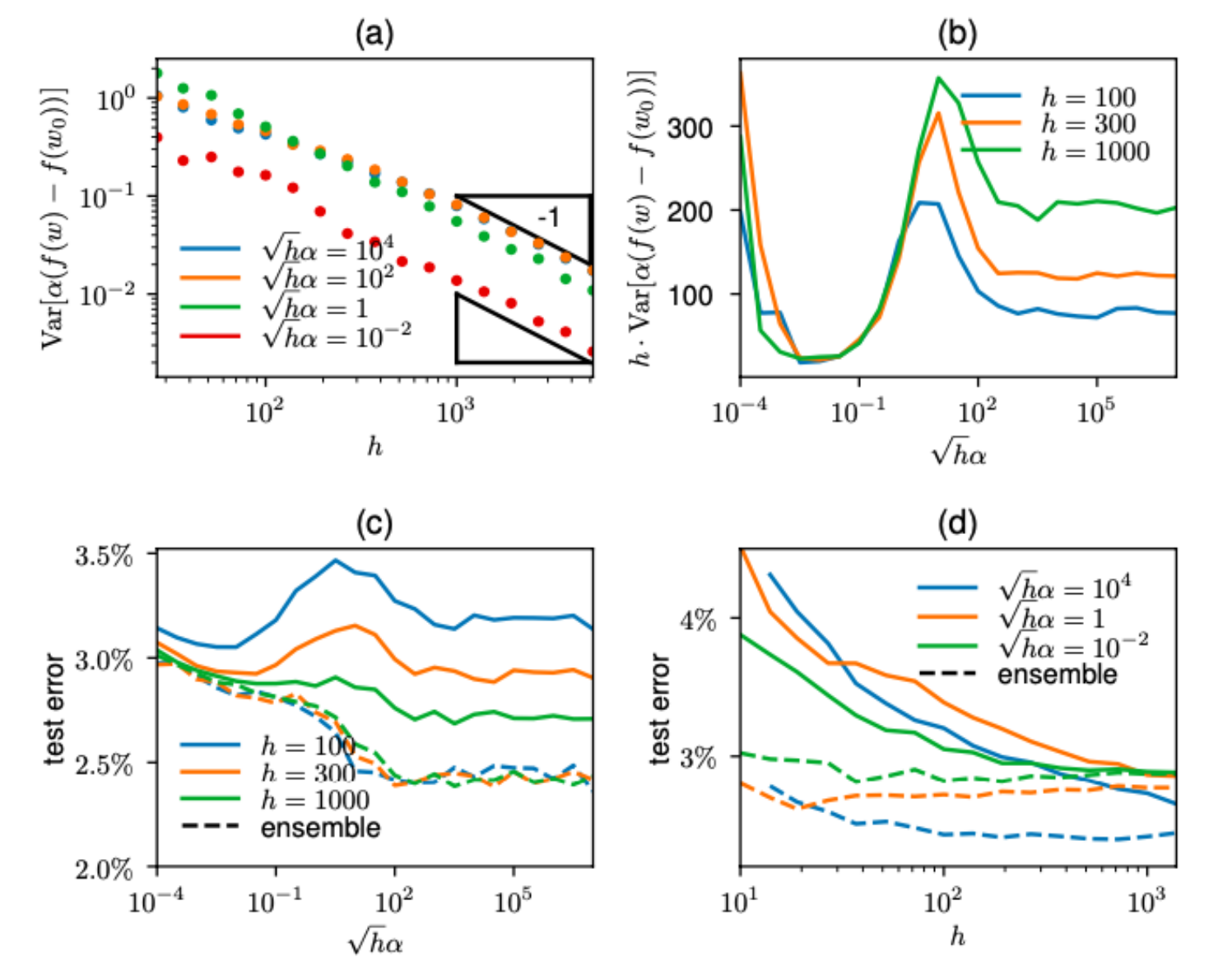


Figure 3: *(a)* Schematic representation of the parameters that we probe: either we fix $\alpha\sqrt{h}$ or we keep the width h constant and we vary α. The location of the cross-over between the lazy and feature-training regimes is also indicated. *(b)* Test error *v.s.* network's width h for different values of $\sqrt{h}\alpha$ as indicated in legend. The black solid line is the test error of the frozen NTK at initialization, a limit that is recovered as $\alpha\sqrt{h} \to \infty$. *(c)* Test error *v.s.* α, for different widths h. (averaged over 20 initializations) *(d)* Same data as in *(c)*: after an arbitrary affine transformation $(ax + b)$ of the test error, the curves collapse when plotted against $\sqrt{h}\alpha$.

Fluctuations of the Output

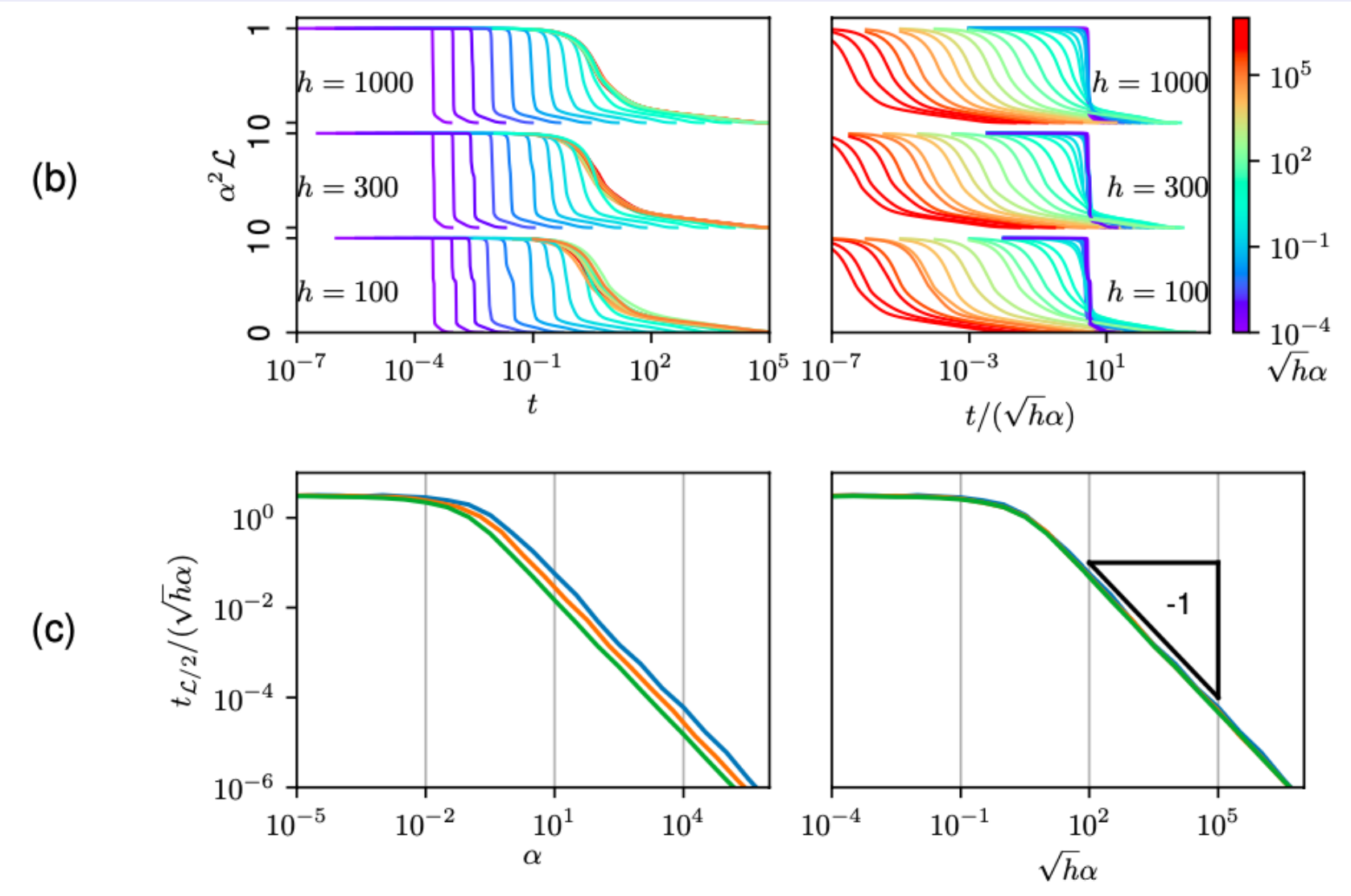
In the paper $h \equiv m =$ network width



Training Dynamics

$t_{\mathcal{L}/2}$: time when loss is reduced by half.

Feature training regime: $t_{\mathcal{L}/2} \sim \sqrt{m}\alpha$ $(m = h)$



Feature Versus Lazy Training

Heuristic explanations:

- Ballpark evolution of partial derivatives ($\sigma'(\tilde{z}) = O(1)$)

$$\frac{\partial f}{\partial W^0} = O\left(\frac{1}{\sqrt{m}}\right), \quad \frac{\partial f}{\partial W^\ell} = O\left(\frac{1}{m}\right), \quad \frac{\partial f}{\partial W^L} = O\left(\frac{1}{\sqrt{m}}\right).$$

  From which, using GD $\dot{w} = -\nabla L$,

$$\dot{W}^0 = O\left(\frac{1}{\alpha\sqrt{m}}\right), \quad \dot{W}^\ell = O\left(\frac{1}{\alpha m}\right), \quad \dot{W}^L = O\left(\frac{1}{\alpha\sqrt{m}}\right).$$

- Moreover, the preceding implies $m^{-1/2}\dot{W}^\ell z^\ell = O(1/(\alpha\sqrt{m}))$, resulting in

$$\dot{\tilde{z}}^{\ell+1} = \frac{1}{\sqrt{m}}(\dot{W}^\ell z^\ell + W^\ell \dot{\tilde{z}}^\ell) = \frac{1}{\sqrt{m}} W^\ell \dot{\tilde{z}}^\ell + O\left(\frac{1}{\alpha\sqrt{m}}\right)$$

  Also, for wide networks, by CLT, $W^\ell \dot{\tilde{z}}^\ell \approx \sqrt{m}\dot{\tilde{z}}^\ell$, producing

$$\dot{\tilde{z}}^\ell = O\left(\frac{1}{\alpha\sqrt{m}}\right)$$

Feature Versus Lazy Training

Heuristic explanations:

- Hence, for $t \ll t_c := \alpha\sqrt{m}$,

$$W^L(t) - W^L(0) = o(1), \quad \tilde{z}^\ell(t) - \tilde{z}^\ell(0) = o(1)$$

  we are in the lazy/linear training regime where preactivations and weights did not have time to evolve

- Difference between the lazy and active training
  - Lazy training: finds zero loss in $O(1)$ time, and if $t_c = \alpha\sqrt{m} >> 1$, the network never leaves the linear regime
  - Active training: when $\alpha\sqrt{m} \ll 1$, the training does not stop at $t \sim t_c = \alpha\sqrt{m}$, for which we have significant change in parameters, namely

$$W^L(t_c) - W^L(0) = O(1), \quad \tilde{z}^\ell(t_c) - \tilde{z}^\ell(0) = O(1)$$

Random Initialization: Vanishing & Exploding Gradient

Matrix Recursion

- Let $\boldsymbol{F}^{(1)} = (F_1^{(1)}, \ldots, F_{n_1}^{(1)})$ be a column vector of the outputs of the input layer with n_1 neurons.
- ReLU activations: $F_i^{(1)} = (\boldsymbol{w}_i^{(0)} \cdot \boldsymbol{x} + b_i^{(0)})_+$, $\boldsymbol{x} \in \mathbb{R}^d, \boldsymbol{w}_i^{(0)} \in \mathbb{R}^d, b_i^{(0)} \in \mathbb{R}$.
- Let $\boldsymbol{W}^{(0)}$ be a $n_1 \times d$ matrix and $\boldsymbol{B}^{(0)} = (b_1^{(0)}, \ldots b_{n_1}^{(0)})$ a column vector, both having i.i.d. components and independent of each other.
- Assume ℓ layers with each layer having $n_i, 1 \leq i \leq \ell$ neurons. Then, the successive layers satisfy the following recursion for $1 \leq i \leq \ell$

$$\boldsymbol{F}^{(i)} = \left(\boldsymbol{W}^{(i-1)} \boldsymbol{F}^{(i-1)} + \boldsymbol{B}^{(i-1)} \right)_+,$$

  where $\boldsymbol{F}^{(i)}, \boldsymbol{B}^{(i-1)}$ are column vectors with n_i components and $\boldsymbol{W}^{(i-1)}$ is a $n_i \times n_{i-1}$ matrix.
- Assume that all $\boldsymbol{B}^{(i)}, \boldsymbol{W}^{(i)}$, are mutually independent with i.i.d. components and let a typical component from $\boldsymbol{B}^{(i)}, \boldsymbol{W}^{(i)}$ be equal in distribution to $B^{(i)}, W^{(i)}$, respectively.

Random Initialization: Vanishing & Exploding Gradient

Computation of variance assuming $B^{(i)}, W^{(i)}$ being symmetric.

- By symmetry and independence, for $\ell \geq 2$,

$$\begin{aligned}
\mathbb{E}(F_j^{(\ell)})^2 &= \frac{1}{2}\mathbb{E}\left(\sum_{k=1}^{n_{\ell-1}} W_{jk}^{(\ell-1)} F_k^{(\ell-1)} + B_j^{(\ell-1)}\right)^2 \\
&= \frac{n_{\ell-1}}{2}\mathbb{E}(W^{(\ell-1)})^2\mathbb{E}(F_1^{(\ell-1)})^2 + \frac{1}{2}\mathbb{E}(B^{(\ell-1)})^2 \\
&=: \rho_{\ell-1}\mathbb{E}(F_1^{(\ell-1)})^2 + \frac{1}{2}\mathbb{E}(B^{(\ell-1)})^2 \\
&= \rho_{\ell-1}\cdots\rho_1\mathbb{E}(F_1^{(1)})^2 + \frac{1}{2}\sum_{k=0}^{\ell-2}\pi_k\mathbb{E}(B^{(\ell-k-1)})^2,
\end{aligned}$$

  where $\pi_0 := 1$ and $\pi_k := \rho_{\ell-1}\cdots\rho_{\ell-k}, 1 \leq k < \ell$.

- If $\rho_i > 1$, variance grows exponentially
  If $\rho_i < 1$, variance decreases exponentially
  critical case: typical initialization in NNs (lazy regime)

$$\mathbb{E}(W^{(i)})^2 = \frac{2}{n_i} \quad \Leftrightarrow \quad \sigma(W^{(i)}) = \frac{\sqrt{2}}{\sqrt{n_i}}$$

Random Initialization: Variance of Gradient

Assume critical initialization

$$\sigma(W^{(i)}) = \frac{\sqrt{2}}{\sqrt{n_i}}$$

Then, gradient on layer back from the last

$$F_j^{(\ell)} = \left(\sum_{k=1}^{n_{\ell-1}} W_{jk}^{(\ell-1)} F_k^{(\ell-1)} + B_j^{(\ell-1)} \right)_+$$

$$\Rightarrow \frac{\partial F_j^{(\ell)}}{\partial W_{jk}^{(\ell-1)}} = 1_{\{\sum_{k=1}^{n_{\ell-1}} W_{jk}^{(\ell-1)} F_k^{(\ell-1)} + B_j^{(\ell-1)} \geq 0\}} F_k^{(\ell-1)}$$

$$\Rightarrow \frac{\partial F_j^{(\ell)}}{\partial B_j^{(\ell-1)}} = 1_{\{\sum_{k=1}^{n_{\ell-1}} W_{jk}^{(\ell-1)} F_k^{(\ell-1)} + B_j^{(\ell-1)} \geq 0\}}$$

implying stable variance of the gradient

$$\mathbb{E}\left(\frac{\partial F_j^{(\ell)}}{\partial W_{jk}^{(\ell-1)}} \right)^2 = \frac{1}{2}\mathbb{E}(F_k^{(\ell-1)})^2, \quad \mathbb{E}\left(\frac{\partial F_j^{(\ell)}}{\partial B_j^{(\ell-1)}} \right)^2 = \frac{1}{2}.$$

Random Initialization: Variance of Gradient

Similarly, going two layers up

$$\frac{\partial F_j^{(\ell)}}{\partial W_{km}^{(\ell-2)}} = 1_{\{\sum_{k=1}^{n_{\ell-1}} W_{jk}^{(\ell-1)} F_k^{(\ell-1)} + B_j^{(\ell-1)} \geq 0\}} W_{jk}^{(\ell-1)} \times 1_{\{\sum_{m=1}^{n_{\ell-1}} W_{km}^{(\ell-2)} F_m^{(\ell-2)} + B_k^{(\ell-2)} \geq 0\}} F_m^{(\ell-2)}$$

$$\frac{\partial F_j^{(\ell)}}{\partial B_k^{(\ell-2)}} = 1_{\{\sum_{k=1}^{n_{\ell-1}} W_{jk}^{(\ell-1)} F_k^{(\ell-1)} + B_j^{(\ell-1)} \geq 0\}} \times W_{jk}^{(\ell-1)} 1_{\{\sum_{m=1}^{n_{\ell-1}} W_{km}^{(\ell-2)} F_m^{(\ell-2)} + B_k^{(\ell-2)} \geq 0\}}$$

one can compute explicitly the variance of the gradient for each layer.

One can show that the variance of the gradient remains stable when the NN is initialized as

$$\sigma(W^{(i)}) = \frac{\sqrt{2}}{\sqrt{n_i}}$$

Reading

- Implicit bias of gradient descent
  - Chapter 15 in Telgarsky, 2021.
  - The Implicit Bias of Gradient Descent on Separable Data, Soudry et al., 2017.
  - Gradient Descent Maximizes the Margin of Homogeneous Neural Networks, Lyu, Kaifeng, and Jian Li, 2019.
  - Implicit Bias of Gradient Descent for Wide Two-Layer Neural Networks Trained with the Logistic Loss, Chizat and bach, 2020.
  - For additional references check the follow up references on Google Scholar, and Chapter 15 in Telgarsky 2021.
- Recent Barron's work
  - Approximation by combinations of ReLU and squared ReLU ridge functions with ℓ_1 and ℓ_0 controls, by Klusowski & Barron, 2018.

Reading

- Residual Networks
  - Deep Residual Learning for Image Recognition, He et al., 2015.
  - Neural Ordinary Differential Equations, by Chen et al., 2018.
- Generative Models and ODEs
  - FFJORD: FREE-FORM CONTINUOUS DYNAMICS FOR SCALABLE REVERSIBLE GENERATIVE MODELS, by Grathwohl et al., 2019.
  - Variational Inference with Normalizing Flows, by Resende & Mohamed, 2015.
  - Improved Variational Inference with Inverse Autoregressive Flow, by Kingma et al., 2017.
- Recent numerical paper on "active/feature" versus "lazy" training:
  - Disentangling feature and lazy training in deep neural networks, M. Geiger, S. Spigler, A. Jacot and M. Wyart, Oct 2020.

Final Project

The key difference from other courses and guiding questions:

- What did you learn about a neural network?
  - The focus should be on NN properties instead of applications.
- How do the changes in NN impact its performance?
  - The changes could be: architecture (e.g., width/depth), activation function, training method, normalization, dropout...
  - In class, we focused on plain vanilla feed-forward networks, but you could choose other types, e.g., ResNets.
- You could center your questions on one or more of the general themes we focused on in class:
  1. Approximation and interpolation theory and the impact of depth.
  2. Optimization landscape and global convergence.
  3. Generalization theory: conditions for small/bounded testing errors.
- Many of the problems we formulated in the context of wide/over-parametrized networks with two types of scaling: NTK/lazy training or mean-field/active training.

Final Project

Rough Paper Outline, about 15 pages:

1. Introduction: e.g., describe the general problem area, DL and specific subtopic(s). Brief literature review, etc.
2. Detailed Problem Description: More detailed literature review for a selected problem(s), detailed description of the known results, theoretical or experimental, etc.
3. Some Reproduction: Theoretical or Experimental partial or full reproduction of the results. For example, run some simulations that illustrate main results.
4. New Results: Theoretical or Experimental Describe in detail your results. If experimental, describe the experiments and results. Explain clearly the graphs and tables from experiments, etc.
5. Discussion and conclusion: e.g., try to draw general inferences from your results. Compare to the known results from the literature, etc.

Final Project

- **Deliverables**:
  - **Paper**: about 15 pages - the most important part.
  - **Presentations**: about 10min each, 10 slides
  - **Software**: Document your code well
- **Second set of presentations**: April 28, Fri, 11am-2pm, 1024 Mudd
  2% EC for those presenting on April 28
- Last presentation slot (likely): Fri, May 5th, between 4:20-7:20pm in 1024 Mudd.
- **Project due**: During the exam week of May 5-12: TBA
- Academic Honesty - do not plagiarize; Turnitin will be used to check for originality

Have Fun and GOOD LUCK!