Neural Tangent Kernel (NTK) Made Practical

Wei Hu

Princeton University

Today’s Theme

- NTK theory [Jacot et al. ’18]: a class of infinitely wide (or ultra-wide) neural networks trained by gradient descent $\Leftrightarrow$ kernel regression with a fixed kernel (NTK)

- Can we use make this theory *practical*?
  1. to understand optimization and generalization phenomena in neural networks
  2. to evaluate the empirical performance of infinitely wide neural networks
  3. to understand network architectural components (e.g. convolution, pooling) through their NTK
  4. to design new algorithms inspired by the NTK theory

Outline

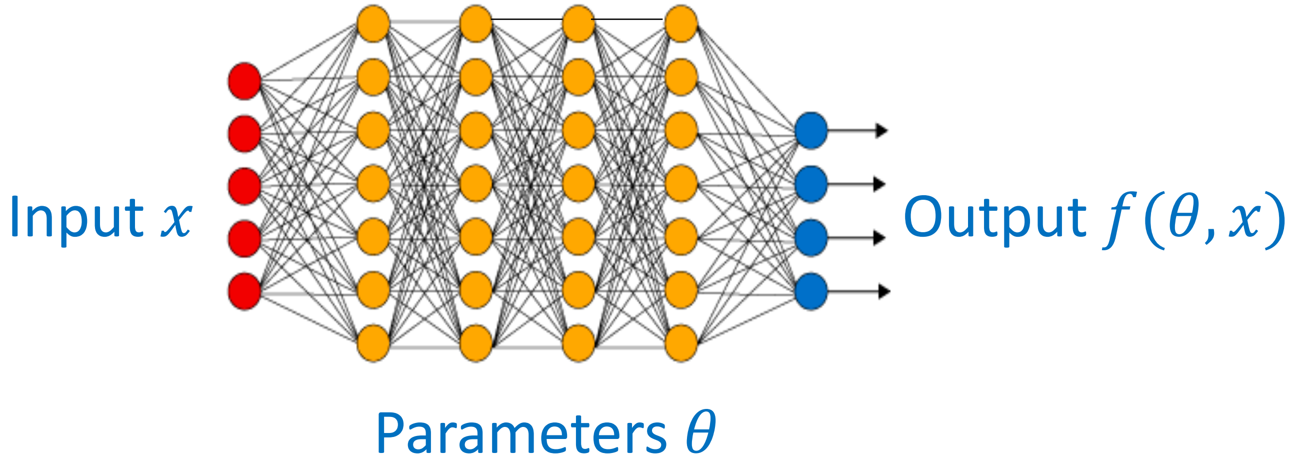
- Recap of the NTK theory
- Implications on opt and gen
- How do infinitely-wide NNs perform?
- NTK with data augmentation
- New algo to deal with noisy labels

Outline

- Recap of the NTK theory
- Implications on opt and gen
- How do infinitely-wide NNs perform?
- NTK with data augmentation
- New algo to deal with noisy labels

Setting: Supervised Learning

- Given data: $(x_1, y_1), \dots, (x_n, y_n) \sim D$
- A neural network: $f(\theta, x)$



- Goal: minimize population loss $\mathbb{E}_{(x,y)\sim D}\ell(f(\theta, x), y)$
- Algorithm: minimize training loss $\frac{1}{n}\sum_{i=1}^{n} \ell(f(\theta, x_i), y_i)$ by (S)GD

Multi-Layer NN with “NTK Parameterization”

Neural network: $f(\theta, x) = W^{(L+1)} \sqrt{\frac{c}{d_L}} \sigma\left(W^{(L)} \sqrt{\frac{c}{d_{L-1}}} \sigma\left(W^{(L-1)} \cdots \sqrt{\frac{c}{d_1}} \sigma\left(\sqrt{\frac{c}{d_0}} W^{(1)} x\right)\right)\right)$

- $W^{(j)} \in \mathbb{R}^{d_j \times d_{j-1}}$, single output ($d_{L+1} = 1$)
- Activation function $\sigma(z)$, with $c = \left(\mathbb{E}_{z \sim \mathcal{N}(0,1)} \sigma^2(z)\right)^{-1}$
- Initialization: $W_{ij}^{(h)} \sim \mathcal{N}(0,1)$
- Let hidden widths $d_1, \ldots, d_L \to \infty$

- Empirical risk minimization: $\ell(\theta) = \frac{1}{2} \sum_{i=1}^{n} (f(\theta, x_i) - y_i)^2$
- Gradient descent (GD): $\theta(t+1) = \theta(t) - \eta \nabla \ell(\theta(t))$

The Main Idea of NTK: Linearization

First-order Taylor approximation (linearization) of the network around the initialized parameters θ^{init}:

$$f(\theta, x) \approx \cancel{f\left(\theta^{\text{init}}, x\right)} + \left\langle \nabla_\theta f\left(\theta^{\text{init}}, x\right), \theta - \theta^{\text{init}} \right\rangle$$

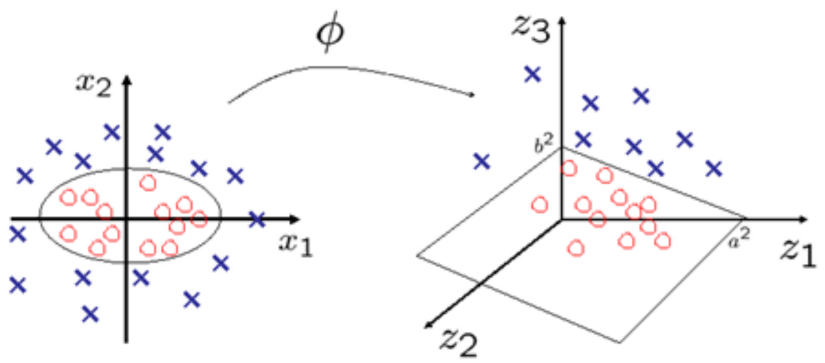
- The approximation holds if θ is close to θ^{init} (true for ultra-wide NN trained by GD)
- The network is *linear* in the gradient feature map $x \mapsto \nabla_\theta f\left(\theta^{\text{init}}, x\right)$

* Ignore $f\left(\theta^{\text{init}}, x\right)$ for today: can make sure it's 0 by setting $f(\theta, x) = g(\theta_1, x) - g(\theta_2, x)$ and initialize $\theta_1 = \theta_2$

Kernel Method

Input $\boldsymbol{x}$ → $\boldsymbol{\phi(x)}$

Possibly infinitely dimensional



Kernel regression/SVM: learn linear function of $\phi(x)$

Kernel trick: only need to compute $k(x, x') = \langle \phi(x), \phi(x') \rangle$ for pair of inputs x, x'

- Assuming $f(\theta, x) \approx \langle \nabla_\theta f(\theta^{\mathrm{init}}, x), \theta - \theta^{\mathrm{init}} \rangle$, we care about the kernel

$$k^{\mathrm{init}}(x, x') = \langle \nabla_\theta f(\theta^{\mathrm{init}}, x), \nabla_\theta f(\theta^{\mathrm{init}}, x') \rangle$$

The Large Width Limit

$f(\theta, x)$

$$= W^{(L+1)} \sqrt{\frac{c}{d_L}} \sigma \left(W^{(L)} \sqrt{\frac{c}{d_{L-1}}} \sigma \left(W^{(L-1)} \cdots \sqrt{\frac{c}{d_1}} \sigma \left(\sqrt{\frac{c}{d_0}} W^{(1)} x \right) \right) \right)$$

If $d_1 = d_2 = \cdots = d_L = m \to \infty$, we have

1. Convergence to a *fixed* kernel at initialization

Theorem [Arora, Du, H, Li, Salakhutdinov, Wang '19]:

If $\min\{d_1, \dots, d_L\} \geq \epsilon^{-4} \mathrm{poly}(L) \log\left(\frac{1}{\delta}\right)$, then for any inputs x, x', w.p. $1 - \delta$ over the random initialzation θ^{init}:

$$\left| k^{\mathrm{init}}(x, x') - k(x, x') \right| \leq \epsilon$$

$k: \mathbb{R}^d \times \mathbb{R}^d$ is a kernel function that depends on network architecture

The Large Width Limit

$$f(\theta, x) = W^{(L+1)} \sqrt{\frac{c}{d_L}} \sigma \left(W^{(L)} \sqrt{\frac{c}{d_{L-1}}} \sigma \left(W^{(L-1)} \cdots \sqrt{\frac{c}{d_1}} \sigma \left(\sqrt{\frac{c}{d_0}} W^{(1)} x \right) \right) \right)$$

If $d_1 = d_2 = \cdots = d_L = m \to \infty$, we have

1. Convergence to a *fixed* kernel at initialization
2. Weights *don't move* much during GD: $\frac{\|W^{(h)}(t) - W^{(h)}(0)\|_F}{\|W^{(h)}(0)\|_F} = O\left(\frac{1}{\sqrt{m}}\right)$
3. *Linearization* approximation holds: $f(\theta(t), x) = \langle \nabla_\theta f(\theta^{\text{init}}, x), \theta(t) - \theta^{\text{init}} \rangle \pm O\left(\frac{1}{\sqrt{m}}\right)$

Equivalence to Kernel Regression

For infinite-width network, we have $f(\theta, x) = \langle \nabla_\theta f(\theta^{\text{init}}, x), \theta - \theta^{\text{init}} \rangle$ and $\langle \nabla_\theta f(\theta^{\text{init}}, x), \nabla_\theta f(\theta^{\text{init}}, x') \rangle = k(x, x')$

$\Longrightarrow$ GD is doing kernel regression w.r.t. the fixed kernel $k(\cdot,\cdot)$

Kernel regression solution:

$$f_{\text{ker}}(x) = (k(x, x_1), \ldots, k(x, x_n)) \cdot K^{-1} \cdot y$$

$K \in \mathbb{R}^{n \times n}, K_{i,j} = k(x_i, x_j)$
$y \in \mathbb{R}^n$

Theorem [Arora, Du, H, Li, Salakhutdinov, Wang '19]:
If $d_1 = d_2 = \cdots = d_L = m$ is sufficiently large, then for any input x, w.h.p. the NN at the end of GD satisfies:

$$|f_{\text{nn}}(x) - f_{\text{ker}}(x)| \leq \epsilon$$

Wide NN trained by GD is equivalent to kernel regression w.r.t. the NTK!

Outline

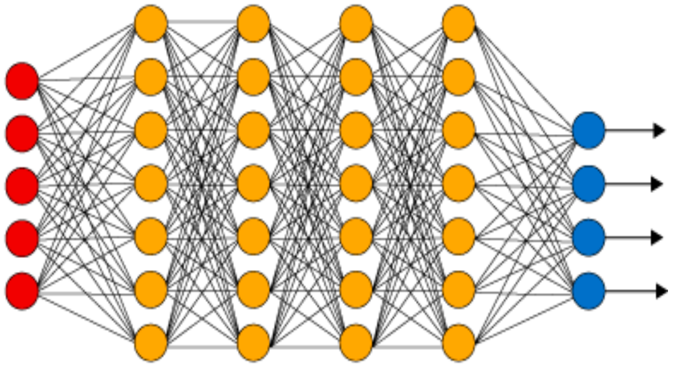
- Recap of the NTK theory
- Implications on opt and gen
- How do infinitely-wide NNs perform?
- NTK with data augmentation
- New algo to deal with noisy labels

True vs Random Labels

"Understanding deep learning requires rethinking generalization", Zhang et al. 2017

True Labels: 2 1 3 1 4

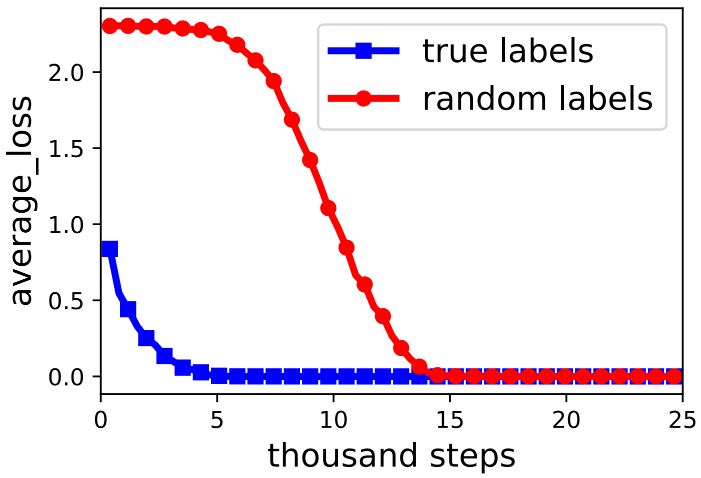
Random Labels: 5 1 7 0 8

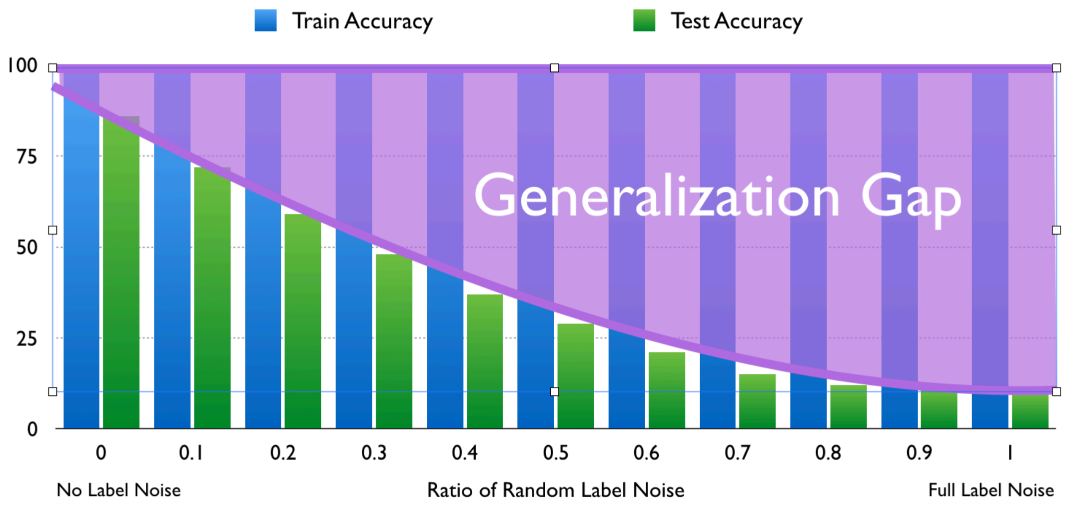


Phenomena

① Faster convergence with true labels than random labels

② Good generalization with true labels, poor generalization with random labels





Explaining Training Speed Difference

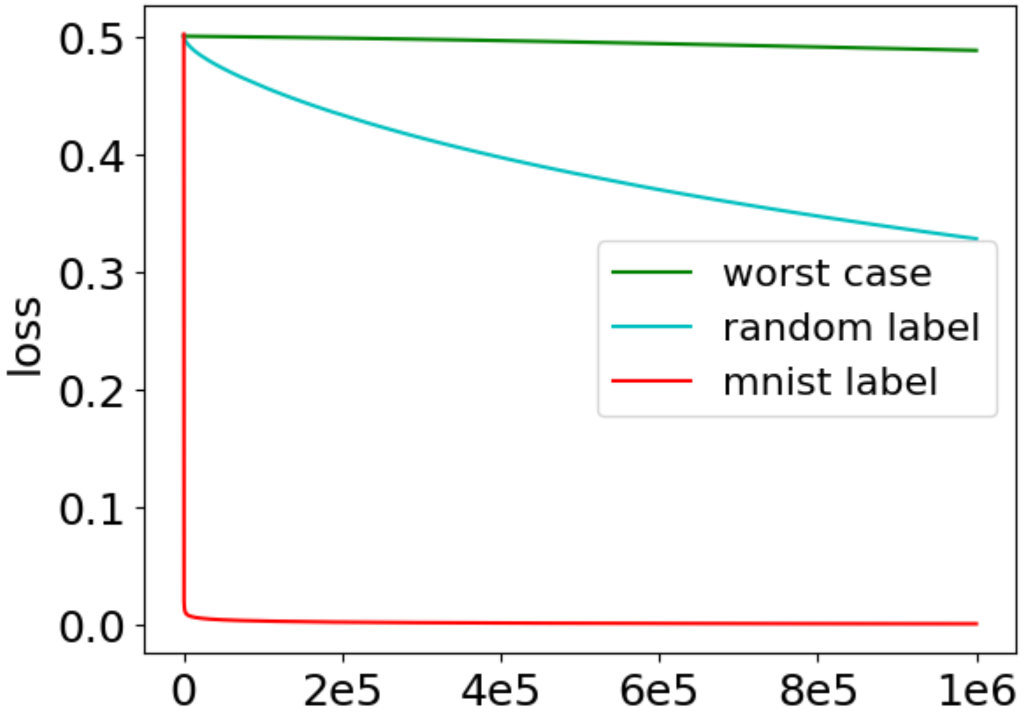
Theorem [Arora, Du, H, Li, Wang '19]: Training loss at time t is

$$\sum_{i=1}^{n}(f_t(x_i) - y_i)^2 \approx \sum_{i=1}^{n} e^{-2\lambda_i t} \cdot \langle v_i, y\rangle^2$$

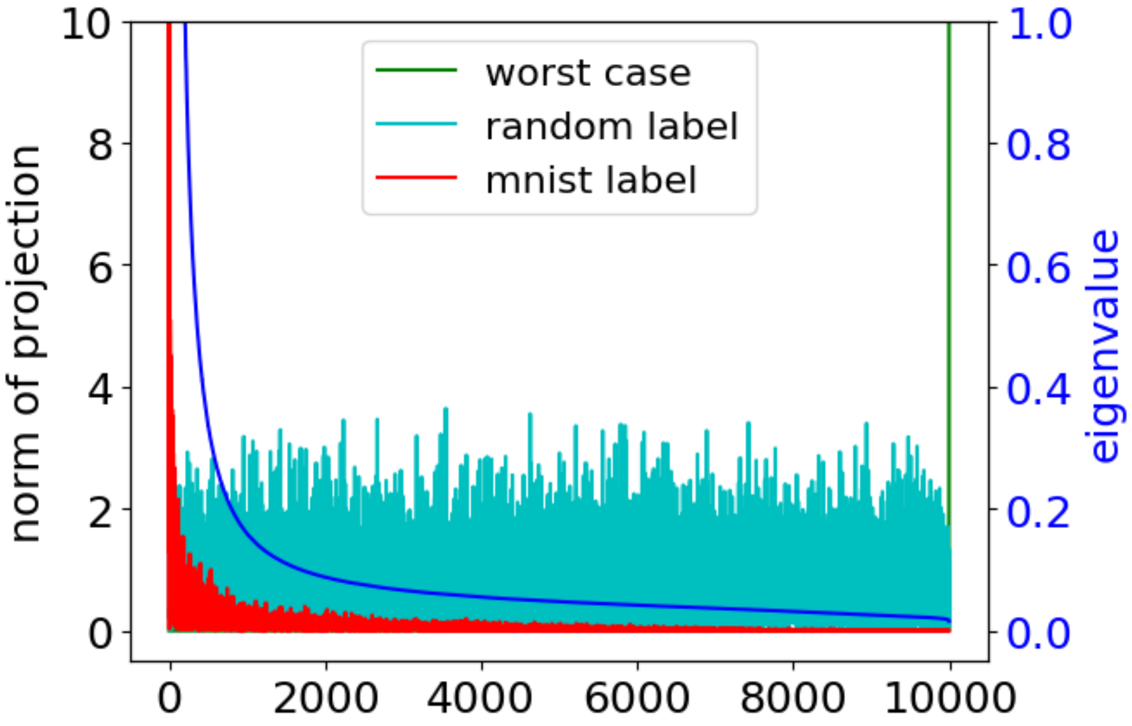
where $K = \sum_{i=1}^{n} \lambda_i v_i v_i^\top$ is the eigen-decomposition

- Components of y on larger eigenvectors of K converge faster than those on smaller eigenvectors
- If y aligns well with top eigenvectors, then GD converges quickly
- If y's projections on eigenvectors are close to being uniform, then GD converges slowly

MNIST (2 Classes)

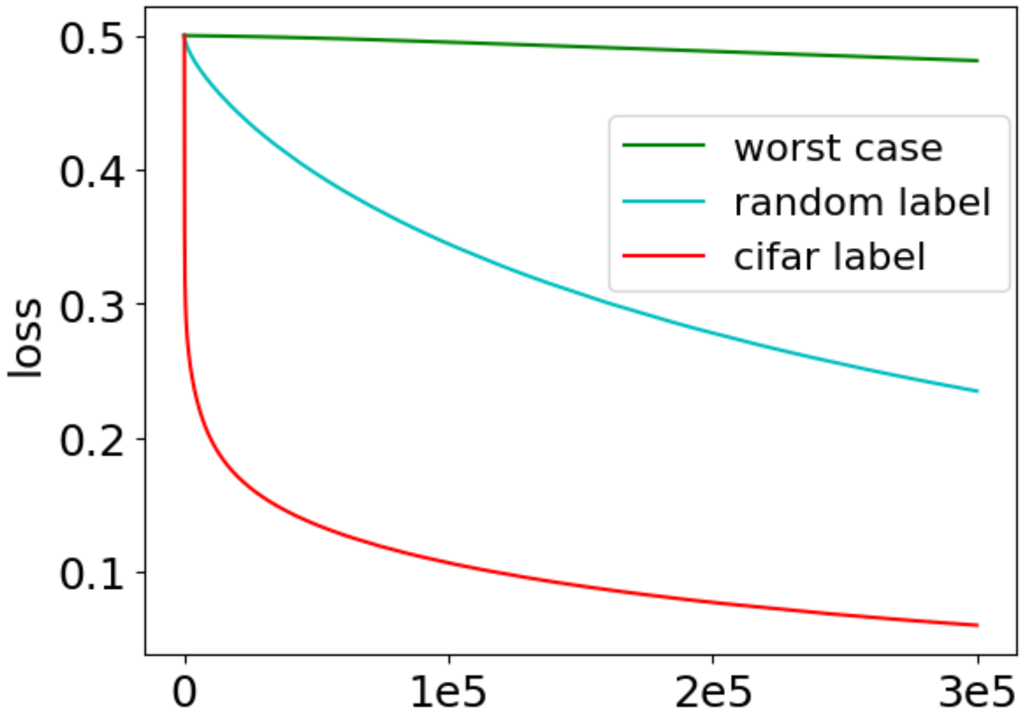


Convergence Rate

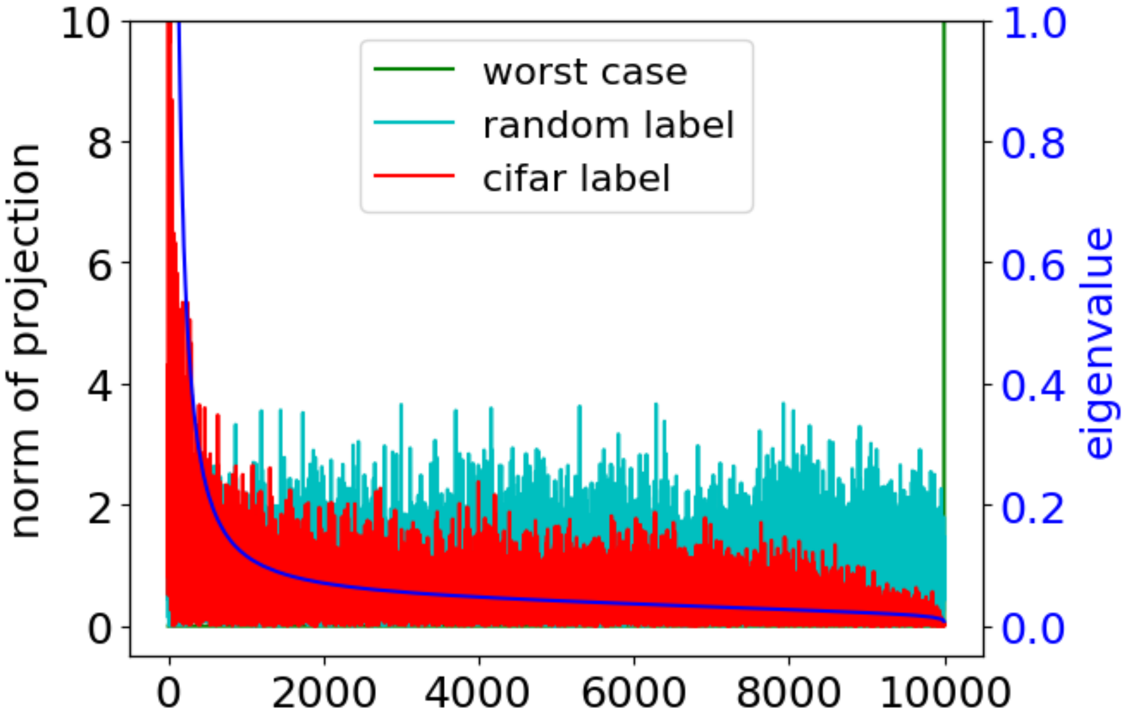


Projections

CIFAR-10 (2 Classes)



Convergence Rate



Projections

Generalization

- It suffices to analyze generalization for $f_{\mathrm{ker}}(x) = (k(x, x_1), \dots, k(x, x_n))K^{-1}y$
- For $f(x) = \sum_{i=1}^{n} \alpha_i k(x, x_i)$, its RKHS norm is $\|f\|_{\mathcal{H}} = \sqrt{\alpha^\top K \alpha}$

  $\Longrightarrow \|f_{\mathrm{ker}}\|_{\mathcal{H}} = \sqrt{y^\top K^{-1} y}$

- Then, from the classical Rademacher complexity bound [Bartlett and Mendelson '02] we obtain the population loss bound for f_{ker}:

$$\mathbb{E}_{(x,y)\sim D}|f_{\mathrm{ker}}(x) - y| \le \frac{2\sqrt{y^\top K^{-1} y}\sqrt{\mathrm{tr}[K]}}{n} + \sqrt{\frac{\log(1/\delta)}{n}}$$

- $\mathrm{tr}[K] = O(n) \quad \Longrightarrow \mathbb{E}_{(x,y)\sim D}|f_{\mathrm{ker}}(x) - y| \le O\left(\sqrt{\frac{y^\top K^{-1} y}{n}}\right)$
  - Such bound appeared in [Arora, Du, H, Li, Wang '19], [Cao and Gu '19]

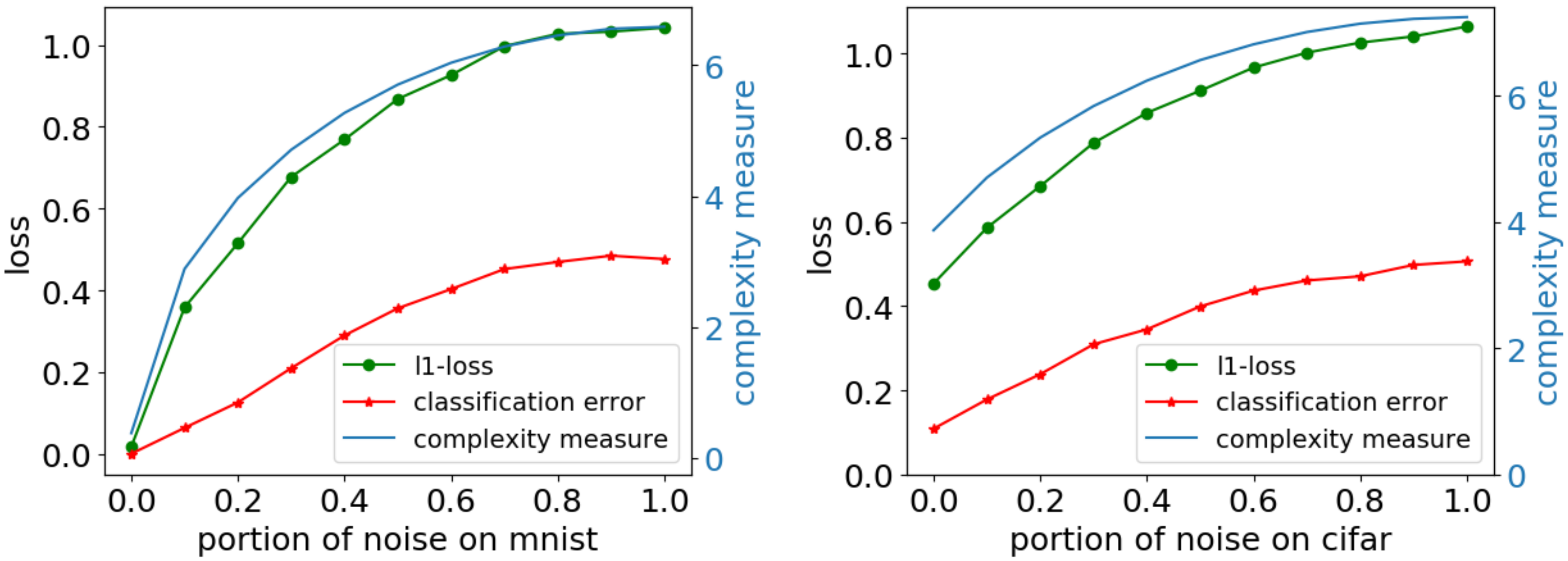
Explaining Generalization Difference

- Consider binary classification ($y_i = \pm 1$)
- We have the bound on classification error:

$$\Pr_{(x,y)\sim D}\left[\operatorname{sign}\left(f_{\mathrm{ker}}(x)\right) \neq y\right] \leq \sqrt{\frac{2y^\top K^{-1} y}{n}}$$

- This bound is *a priori* (can be evaluated before training)
- Can this bound distinguish true and random labels?

Explaining Generalization Difference



MNIST

CIFAR-10

Outline

- Recap of the NTK theory
- Implications on opt and gen
- How do infinitely-wide NNs perform?
- NTK with data augmentation
- New algo to deal with noisy labels

NTK Formula

$$f(\theta, x) = W^{(L+1)} \sqrt{\frac{c}{d_L}} \sigma \left(W^{(L)} \sqrt{\frac{c}{d_{L-1}}} \sigma \left(W^{(L-1)} \cdots \sqrt{\frac{c}{d_1}} \sigma \left(\sqrt{\frac{c}{d_0}} W^{(1)} x \right) \right) \right)$$

At random init, what does the value $\langle \nabla_\theta f(\theta^{\text{init}}, x), \nabla_\theta f(\theta^{\text{init}}, x') \rangle$ converge to?

NTK Formula

$$f(\theta,x) = W^{(L+1)}\sqrt{\frac{c}{d_L}}\sigma\left(W^{(L)}\sqrt{\frac{c}{d_{L-1}}}\sigma\left(W^{(L-1)}\cdots\sqrt{\frac{c}{d_1}}\sigma\left(\sqrt{\frac{c}{d_0}}W^{(1)}x\right)\right)\right)$$

At random init, what does the value $\langle\nabla_\theta f(\theta^{\text{init}},x),\nabla_\theta f(\theta^{\text{init}},x')\rangle$ converge to?

[Jacot et al. '18]:

$$\Sigma^{(0)}(\boldsymbol{x},\boldsymbol{x}') = \boldsymbol{x}^\top\boldsymbol{x}',$$

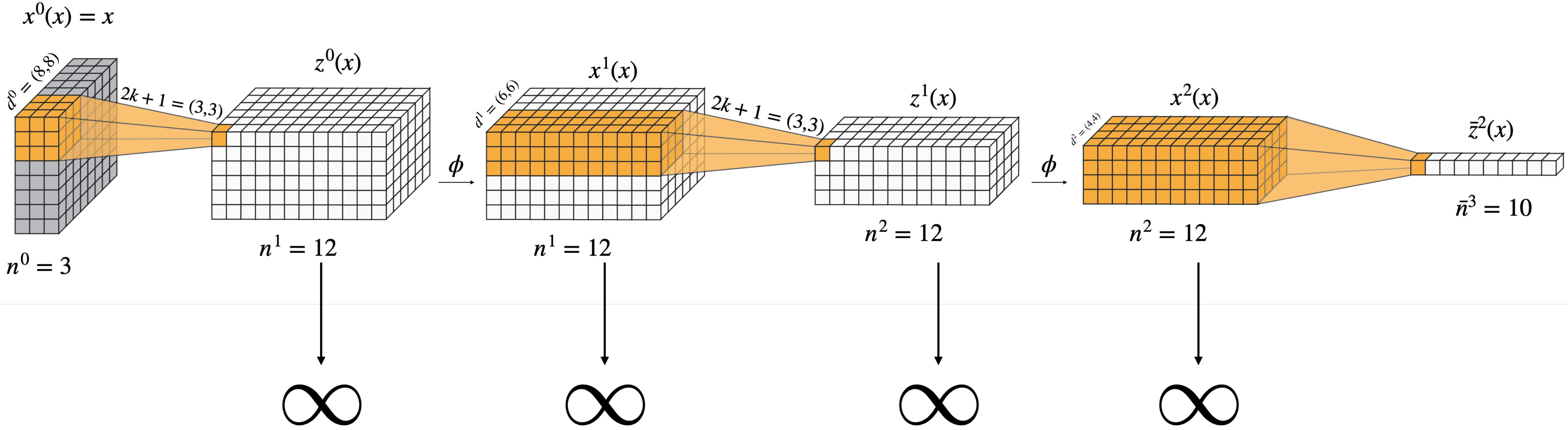
$$\boldsymbol{\Lambda}^{(h)}(\boldsymbol{x},\boldsymbol{x}') = \begin{pmatrix}\Sigma^{(h-1)}(\boldsymbol{x},\boldsymbol{x}) & \Sigma^{(h-1)}(\boldsymbol{x},\boldsymbol{x}')\\ \Sigma^{(h-1)}(\boldsymbol{x}',\boldsymbol{x}) & \Sigma^{(h-1)}(\boldsymbol{x}',\boldsymbol{x}')\end{pmatrix}\in\mathbb{R}^{2\times 2},$$

$$\Sigma^{(h)}(\boldsymbol{x},\boldsymbol{x}') = c_\sigma \mathop{\mathbb{E}}_{(u,v)\sim\mathcal{N}\left(\boldsymbol{0},\boldsymbol{\Lambda}^{(h)}\right)}\left[\sigma(u)\,\sigma(v)\right].$$

$$\dot{\Sigma}^{(h)}(\boldsymbol{x},\boldsymbol{x}') = c_\sigma \mathop{\mathbb{E}}_{(u,v)\sim\mathcal{N}\left(\boldsymbol{0},\boldsymbol{\Lambda}^{(h)}\right)}\left[\dot{\sigma}(u)\dot{\sigma}(v)\right]$$

$$k(x,x') = \sum_{h=1}^{L+1}\left(\Sigma^{(h-1)}(x,x')\cdot\prod_{h'=h}^{L+1}\dot{\Sigma}^{(h')}(x,x')\right)$$

Convolutional Neural Networks (CNNs)



Convolutional NTK?

CNTK Formula

- For $\alpha = 1, \ldots, C^{(0)}$, $(i, j, i', j') \in [P] \times [Q] \times [P] \times [Q]$, define

$$\boldsymbol{K}^{(0)}_{(\alpha)}\left(\boldsymbol{x}, \boldsymbol{x}'\right) = \boldsymbol{x}_{(\alpha)} \otimes \boldsymbol{x}'_{(\alpha)} \text{ and } \left[\boldsymbol{\Sigma}^{(0)}(\boldsymbol{x}, \boldsymbol{x}')\right]_{ij,i'j'} = \sum_{\alpha=1}^{C^{(0)}} \operatorname{tr}\left(\left[\boldsymbol{K}^{(0)}_{(\alpha)}(\boldsymbol{x}, \boldsymbol{x}')\right]_{\mathcal{D}_{ij,i'j'}}\right).$$

- For $h \in [L]$,
  - For $(i, j, i', j') \in [P] \times [Q] \times [P] \times [Q]$, define

$$\boldsymbol{\Lambda}^{(h)}_{ij,i'j'}(\boldsymbol{x}, \boldsymbol{x}') = \begin{pmatrix} \left[\boldsymbol{\Sigma}^{(h-1)}(\boldsymbol{x}, \boldsymbol{x})\right]_{ij,ij} & \left[\boldsymbol{\Sigma}^{(h-1)}(\boldsymbol{x}, \boldsymbol{x}')\right]_{ij,i'j'} \\ \left[\boldsymbol{\Sigma}^{(h-1)}\left(\boldsymbol{x}', \boldsymbol{x}\right)\right]_{i'j',ij} & \left[\boldsymbol{\Sigma}^{(h-1)}\left(\boldsymbol{x}', \boldsymbol{x}'\right)\right]_{i'j',i'j'} \end{pmatrix} \in \mathbb{R}^{2\times 2}.$$

  - Define $\boldsymbol{K}^{(h)}(\boldsymbol{x}, \boldsymbol{x}'), \dot{\boldsymbol{K}}^{(h)}(\boldsymbol{x}, \boldsymbol{x}') \in \mathbb{R}^{P\times Q\times P\times Q}$, for $(i, j, i', j') \in [P] \times [Q] \times [P] \times [Q]$

$$\left[\boldsymbol{K}^{(h)}(\boldsymbol{x}, \boldsymbol{x}')\right]_{ij,i'j'} = \underset{(u,v)\sim\mathcal{N}\left(\mathbf{0}, \boldsymbol{\Lambda}^{(h)}_{ij,i'j'}(\boldsymbol{x}, \boldsymbol{x}')\right)}{\mathbb{E}} \left[\sigma\left(u\right)\sigma\left(v\right)\right],$$

$$\left[\dot{\boldsymbol{K}}^{(h)}(\boldsymbol{x}, \boldsymbol{x}')\right]_{ij,i'j'} = \underset{(u,v)\sim\mathcal{N}\left(\mathbf{0}, \boldsymbol{\Lambda}^{(h)}_{ij,i'j'}(\boldsymbol{x}, \boldsymbol{x}')\right)}{\mathbb{E}} \left[\dot{\sigma}\left(u\right)\dot{\sigma}\left(v\right)\right].$$

  - Define $\boldsymbol{\Sigma}^{(h)}(\boldsymbol{x}, \boldsymbol{x}') \in \mathbb{R}^{P\times Q\times P\times Q}$, for $(i, j, i', j') \in [P] \times [Q] \times [P] \times [Q]$

$$\left[\boldsymbol{\Sigma}^{(h)}(\boldsymbol{x}, \boldsymbol{x}')\right]_{ij,i'j'} = \frac{c_\sigma}{q^2} \operatorname{tr}\left(\left[\boldsymbol{K}^{(h)}(\boldsymbol{x}, \boldsymbol{x}')\right]_{D_{ij,i'j'}}\right).$$

CNTK Formula (Cont'd)

1. First, we define $\boldsymbol{\Theta}^{(0)}(\boldsymbol{x}, \boldsymbol{x}') = \boldsymbol{\Sigma}^{(0)}(\boldsymbol{x}, \boldsymbol{x}')$.
2. For $h = 1, \ldots, L$ and $(i, j, i', j') \in [P] \times [Q] \times [P] \times [Q]$, we define

$$\left[\boldsymbol{\Theta}^{(h)}(\boldsymbol{x}, \boldsymbol{x}')\right]_{ij,i'j'} = \frac{c_\sigma}{q^2} \operatorname{tr}\left(\left[\dot{\boldsymbol{K}}^{(h)}(\boldsymbol{x}, \boldsymbol{x}') \odot \boldsymbol{\Theta}^{(h-1)}(\boldsymbol{x}, \boldsymbol{x}') + \boldsymbol{K}^{(h)}(\boldsymbol{x}, \boldsymbol{x}')\right]_{D_{ij,i'j'}}\right)$$

3. Lastly, the final kernel value is defined as

$$\operatorname{tr}\left(\boldsymbol{\Theta}^{(L)}(\boldsymbol{x}, \boldsymbol{x}')\right).$$

CNTK on CIFAR-10 [Arora, Du, H, Li, Salakhutdinov, Wang ’19]

Depth	CNN-V	CNTK-V	CNTK-V-2K	CNN-GAP	CNTK-GAP	CNTK-GAP-2K
3	59.97%	64.47%	40.94%	63.81%	70.47%	49.71%
4	60.20%	65.52%	42.54%	80.93%	75.93%	51.06%
6	64.11%	66.03%	43.43%	83.75%	76.73%	51.73%
11	69.48%	65.90%	43.42%	82.92%	**77.43%**	51.92%
21	75.57%	64.09%	42.53%	83.30%	77.08%	52.22%

- CNTKs/infinitely wide CNNs achieve reasonably good performance
- but are 5%-7% worse than the corresponding finite-width CNNs trained with SGD (w/o batch norm, data aug)

NTKs on Small Datasets

- On 90 UCI datasets (<5k samples)

Classifier	Avg Acc	P95	PMA
FC NTK	82%	72%	96%
FC NN	81%	60%	95%
Random Forest	82%	68%	95%
RBF Kernel	81%	72%	94%

[Arora et al. 2020]

- On graph classification tasks

	Method	COLLAB	IMDB-B	IMDB-M	PTC
GNN	GCN	79%	74%	51%	64%
GNN	GIN	80%	75%	52%	65%
GK	WL	79%	74%	51%	60%
GK	GNTK	84%	77%	53%	68%

[Du et al. 2019]

NTK is a strong off-the-shelf classifier on small datasets

Outline

- Recap of the NTK theory
- Implications on opt and gen
- How do infinitely-wide NNs perform?
- NTK with data augmentation
- New algo to deal with noisy labels

Data Augmentation

- Idea: create new training images from existing images using pixel translation and flips, assuming that these operations do not change the label

- An important technique to improve generalization, and easy to implement for SGD in deep learning

- Infeasible to incorporate in kernel methods (quadratic running time in # samples)

Global Average Pooling (GAP)

- GAP: average the pixel values of each channel in the last conv layer

  - Without GAP: the final output is defined as

    $$f(\boldsymbol{\theta}, \boldsymbol{x}) = \sum_{\alpha=1}^{C^{(L)}} \left\langle \boldsymbol{W}_{(\alpha)}^{(L+1)}, \boldsymbol{x}_{(\alpha)}^{(L)} \right\rangle$$

    where $\boldsymbol{x}_{(\alpha)}^{(L)} \in \mathbb{R}^{P \times Q}$, and $\boldsymbol{W}_{(\alpha)}^{(L+1)} \in \mathbb{R}^{P \times Q}$ is the weight of the last fully-connected layer.

  - With GAP: the final output is defined as

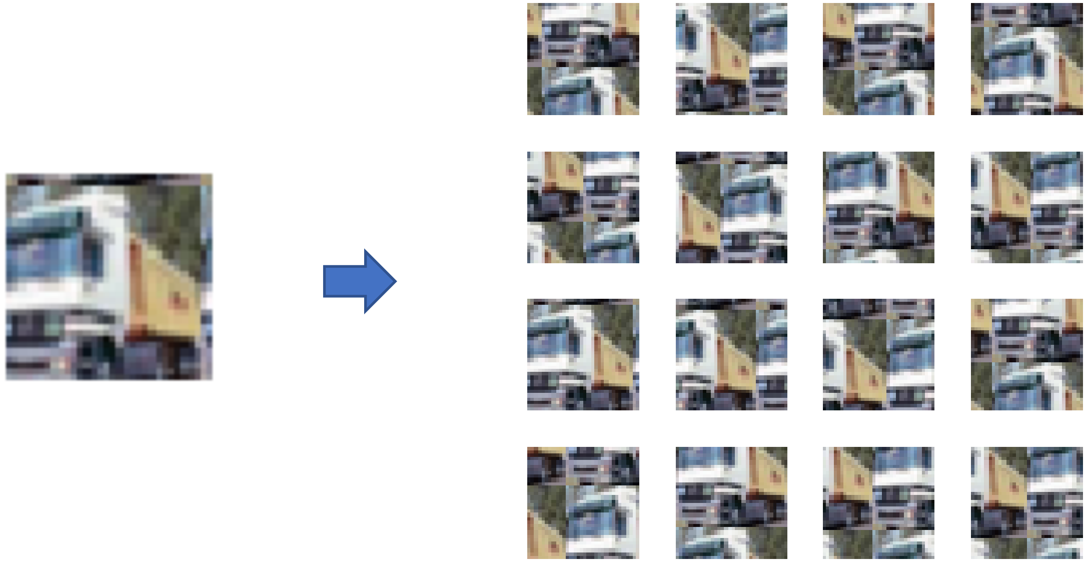
    $$f(\boldsymbol{\theta}, \boldsymbol{x}) = \frac{1}{PQ} \sum_{\alpha=1}^{C^{(L)}} \boldsymbol{W}_{(\alpha)}^{(L+1)} \cdot \sum_{(i,j) \in [P] \times [Q]} \left[\boldsymbol{x}_{(\alpha)}^{(L)}\right]_{i,j}$$

    where $\boldsymbol{W}_{(\alpha)}^{(L+1)} \in \mathbb{R}$ is the weight of the last fully-connected layer.

- We've seen that GAP significantly improves accuracy for both CNNs and CNTKs

GAP ⟺ Data Augmentation [Li, Wang, Yu, Du, H, Salakhutdinov, Arora ’19]

- Augmentation operation: full translation + circular padding



- **Theorem**: For CNTK with circular padding:

{GAP, on original dataset} ⟺ {no GAP, on augmented dataset}

Proof Sketch

- Let G be the group of operations

- Key properties of CNTKs:
  - $\mathrm{cntk}_{GAP}(x, x') = \mathrm{const} \cdot \mathbb{E}_{g \sim G}[\mathrm{cntk}(g(x), x')]$
  - $\mathrm{cntk}(x, x') = \mathrm{cntk}(g(x), g(x'))$

- Show that two kernel regression solutions give the same prediction for all x:

$$\mathrm{cntk}_{\mathrm{GAP}}(x, X) \cdot \mathrm{cntk}_{GAP}(X, X)^{-1} \cdot y = \mathrm{cntk}\left(x, X_{\mathrm{aug}}\right) \cdot \mathrm{cntk}\left(X_{\mathrm{aug}}, X_{\mathrm{aug}}\right)^{-1} \cdot y_{\mathrm{aug}}$$

Enhanced CNTK

- Full translation data augmentation seems a little weird

Full translation

Local translation

- CNTK + "local average pooling" + pre-processing [Coates et al. '11] rivals AlexNet on CIFAR-10 (89%)

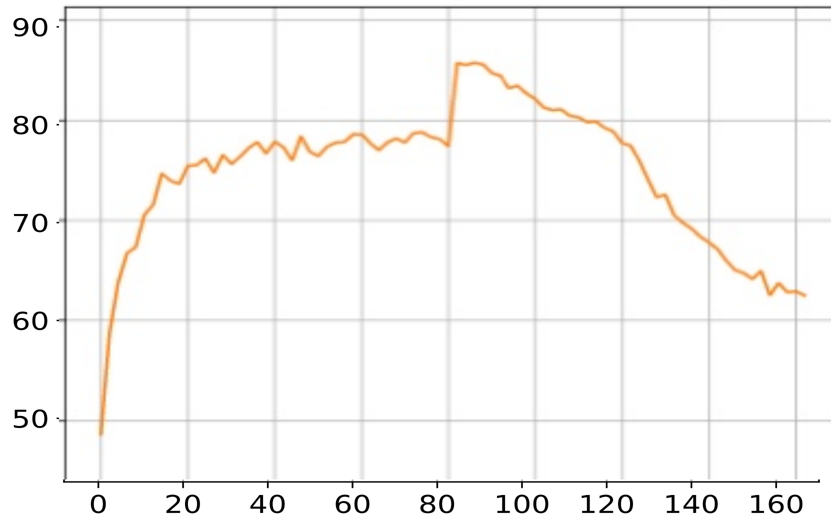
Outline

- Recap of the NTK theory
- Implications on opt and gen
- How do infinitely-wide NNs perform?
- NTK with data augmentation
- New algo to deal with noisy labels

Training with Noisy Labels

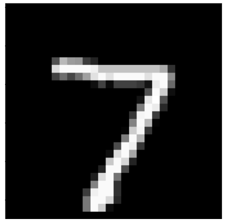
- *Noisy labels* lead to degradation of generalization

- *Early stopping* is useful empirically
- But when to stop? (need access to *clean* validation set)



Test accuracy curve on CIFAR-10 trained with 40% label noise

Setting

$x =$ 

$y = 7$

w.p. $1-p$ → $\tilde{y} = 7$

w.p. p → $\tilde{y} \in \{0, 1, \dots, 9\} \setminus \{7\}$

- In general, there is a transition matrix P such that $P_{i,j} = \Pr[\text{class } j \rightarrow \text{class } i]$
- **<u>Goal</u>**: Given a noisily labeled dataset $S = \{(x_i, \tilde{y}_i)\}_{i=1}^{n}$, train a neural network $f(\theta, \cdot)$ to get small loss on the clean data distribution $\mathcal{D}$

$$L_{\mathcal{D}}(\theta) = \mathbb{E}_{(x,y)\sim\mathcal{D}} \ell(f(\theta, x), y)$$

NTK Viewpoint

$$\ell(\theta) = \frac{1}{2}\sum_{i=1}^{n}(f(\theta, x_i) - y_i)^2 \quad \Longleftrightarrow \quad f_{\text{ker}}(x) = (k(x, x_1), \ldots, k(x, x_n)) \cdot K^{-1} \cdot y$$

Kernel ridge regression ("soft" early stopping)

$$?? \quad \Longleftrightarrow \quad f_{\text{ridge}}(x) = (k(x, x_1), \ldots, k(x, x_n)) \cdot (K + \lambda I)^{-1} \cdot y$$

NTK Viewpoint

$$\ell(\theta) = \frac{1}{2}\sum_{i=1}^{n}(f(\theta, x_i) - y_i)^2 \quad \Longleftrightarrow \quad f_{\text{ker}}(x) = (k(x, x_1), \dots, k(x, x_n)) \cdot K^{-1} \cdot y$$

$$\ell(\theta) = \frac{1}{2}\sum_{i=1}^{n}(f(\theta, x_i) - y_i)^2 + \frac{\lambda}{2}\left\|\theta - \theta^{\text{init}}\right\|_2^2$$

or

$$\ell(\theta, b) = \frac{1}{2}\sum_{i=1}^{n}\left(f(\theta, x_i) - y_i + \sqrt{\lambda} b_i\right)^2$$

$\Longleftrightarrow$

Kernel ridge regression ("soft" early stopping)

$$f_{\text{ridge}}(x) = (k(x, x_1), \dots, k(x, x_n)) \cdot (K + \lambda I)^{-1} \cdot y$$

Theorem [H, Li, Yu '20]:
For infinitely wide NN, both methods lead to kernel ridge regression

Generalization Bound

- $y \in \{\pm 1\}^n$: true labels
- $\tilde{y} \in \{\pm 1\}^n$: observed labels, each label being flipped w.p. p ($p < 1/2$)
- Goal: analyze generalization of $f_{\text{ridge}}(x) = (k(x, x_1), \dots, k(x, x_n)) \cdot (K + \lambda I)^{-1} \tilde{y}$ on the clean distribution $\mathcal{D}$

Theorem:

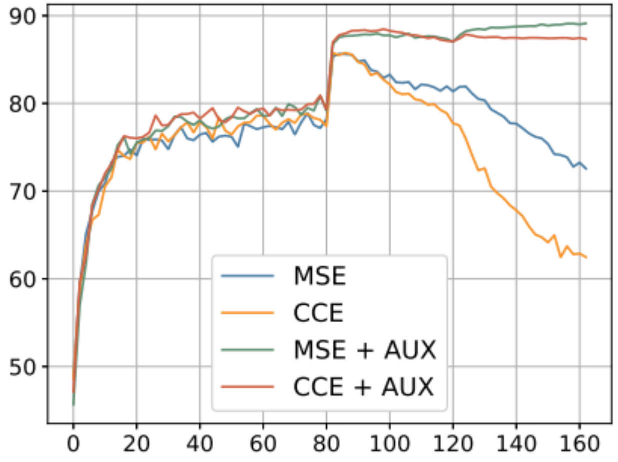
$$\Pr_{(x,y)\sim D}[\operatorname{sign}\left(f_{\text{ridge}}(x)\right) \neq y] \leq \frac{1}{1-2p} O\left(\sqrt{\lambda}\sqrt{\frac{y^\top K^{-1} y}{n}} + \frac{1}{\sqrt{\lambda}}\right)$$

- Set $\lambda = n^\alpha$

Experiments

Noise level	0	0.2	0.4	0.6
CCE normal (early stop)	94.05	89.73	86.35	79.13
MSE normal (early stop)	93.88	89.96	85.92	78.68
CCE+AUX	94.22	92.07	87.81	82.60
MSE+AUX	**94.25**	**92.31**	**88.92**	**83.90**
[Zhang and Sabuncu. 2018]	-	89.83	87.62	82.70

Test accuracy of ResNet-34 using different methods at various noise levels



Test accuracy curve for 40% label noise

- Training with AUX achieves very good accuracy, better than normal training with early stopping and the recent method of Zhang and Sabuncu using the same architecture
- Training with AUX doesn't over-fit (no need to stop early)

Concluding Remarks

- Understanding neural networks by understanding kernels, and vice versa
- NTK/infinite-width NN as a powerful classifier
- Designing principled and practical algorithms inspired by the NTK theory
- Beyond NTK?

References

- Arora, Du, Hu, Li, Salakhutdinov, Wang. *On Exact Computation with an Infinitely Wide Neural Net*. NeurIPS 2019
- Arora, Du, Hu, Li, Wang. *Fine-Grained Analysis of Optimization and Generalization for Overparameterized Two-Layer Neural Networks*. ICML 2019
- Li, Wang, Yu, Du, Hu, Salakhutdinov, Arora. Enhanced Convolutional Neural Tangent Kernels. arXiv 2019
- Hu, Li, Yu. *Understanding Generalization of Neural Networks Trained with Noisy Labels*. ICLR 2020
- Jacot, Gabriel, Hongler. *Neural Tangent Kernel: Convergence and Generalization in Neural Networks*. NeurIPS 2018

Thanks!