The Neural Tangent Kernel, with Applications to Autoencoder Learning

Chinmay Hegde
April 24, 2020

NYU CSE/ECE
Joint work with Thanh Nguyen (ISU) and Raymond Wong (TAMU)
Motivation
Recent advances in machine learning

Simple ML models (such as SVMs, NN)...
Recent advances in machine learning

Simple ML models (such as SVMs, NN)...

...have given way to complicated ML models (such as deep networks)
Training neural network models

\[ \min_{\theta} \sum_{i=1}^{n} L_i(\theta) \]

Loss landscape is terrible! [Li, Xu, Taylor, Studer, Goldstein 2018]
Training neural network models

$$\min_{\theta} \sum_{i=1}^{n} L_i(\theta)$$

**Gradient descent:**

$$\theta^{t+1} \leftarrow \theta^t - \eta^t \sum_{i=1}^{n} \nabla L_i(\theta^t)$$
Training neural network models

\[ \min_\theta \sum_{i=1}^{n} L_i(\theta) \]

Gradient descent:

\[ \theta^{t+1} \leftarrow \theta^t - \eta^t \sum_{i=1}^{n} \nabla L_i(\theta^t) \]

Loss landscape is terrible! [Li, Xu, Taylor, Studer, Goldstein 2018]
Why does gradient descent work?

[Li, Xu, Taylor, Studer, Goldstein 2018]
Background: Linear Regression, Kernel Methods, Gradient Flow
Linear models

\[ u = f(\theta) = X\theta \]
Linear models

\[ u = f(\theta) = X\theta \quad L(\theta) = \frac{1}{2} \| y - X\theta \|^2 \]
Linear models

\[ u = f(\theta) = X\theta \quad L(\theta) = \frac{1}{2}||y - X\theta||_2^2 \]

\[ \theta^{t+1} \leftarrow \theta^t - \eta \nabla L(\theta^t) \]
Linear models

\[ u = f(\theta) = X\theta \quad L(\theta) = \frac{1}{2} ||y - X\theta||^2_2 \]

\[ \theta^{t+1} \leftarrow \theta^t - \eta \nabla L(\theta^t) \]

Infinitesimally small step size (gradient flow):

\[ \frac{d\theta}{dt} = -\nabla L(\theta) = -X^T(X\theta - y) \]
Linear models

\[ u = f(\theta) = X \theta \]
\[ L(\theta) = \frac{1}{2} \| y - X \theta \|_2^2 \]

\[ \theta^{t+1} \leftarrow \theta^t - \eta \nabla L(\theta^t) \]

Infinitesimally small step size (gradient flow):

\[ \frac{d\theta}{dt} = -\nabla L(\theta) = -X^T (X \theta - y) \]

Rewriting dynamics in terms of the model outputs:

\[ \frac{du}{dt} = \nabla_\theta u \frac{d\theta}{dt} \]
Linear models

\[ u = f(\theta) = X\theta \quad L(\theta) = \frac{1}{2}\|y - X\theta\|^2 \]

\[ \theta^{t+1} \leftarrow \theta^t - \eta \nabla L(\theta^t) \]

Infinitesimally small step size (gradient flow):

\[ \frac{d\theta}{dt} = -\nabla L(\theta) = -X^T (X\theta - y) \]

Rewriting dynamics in terms of the model outputs:

\[ \frac{du}{dt} = \nabla_\theta u \frac{d\theta}{dt} \]

\[ = -X X^T (u - y) \]

Gram/kernel matrix
This dynamics actually holds for arbitrary models $u = f(\theta)$:

$$\frac{du}{dt} = \nabla_{\theta} u \nabla_{\theta} u^T (y - u(\theta))$$
This dynamics actually holds for arbitrary models $u = f(\theta)$:

$$\frac{du}{dt} = \nabla_\theta u \nabla_\theta u^T (y - u(\theta))$$

$$\nabla_\theta u \nabla_\theta u^T := K(\theta)$$

is called the tangent kernel.
This dynamics actually holds for arbitrary models $u = f(\theta)$:

$$\frac{du}{dt} = \nabla_\theta u \nabla_\theta u^T (y - u(\theta))$$

$$\nabla_\theta u \nabla_\theta u^T := K(\theta)$$

is called the \textit{tangent kernel}.

For linear models, $K(\theta) = K = XX^T$ (constant, independent of $u, y, \theta$) and only depends on the geometry of the data points.
This dynamics actually holds for arbitrary models $u = f(\theta)$:

$$\frac{du}{dt} = \nabla_\theta u \nabla_\theta u^T (y - u(\theta))$$

$$\nabla_\theta u \nabla_\theta u^T := K(\theta)$$

is called the tangent kernel.

For linear models, $K(\theta) = K = XX^T$ (constant, independent of $u, y, \theta$) and only depends on the geometry of the data points.

Replace $H_{ij} = x_i^T x_j$ by other kernel dot product $K(x_i, x_j) = g(x_i)^T g(x_j)$: Kernel trick
The Neural Tangent Kernel
This talk (results can be extended to deep, wide nets)

\[ u_i = f(x_i, \theta) = \frac{1}{\sqrt{m}} \sum_{j=1}^{m} a_j \phi(x_i^T w_j) \]
This talk (results can be extended to deep, wide nets)

\[ u_i = f(x_i, \theta) = \frac{1}{\sqrt{m}} \sum_{j=1}^{m} a_j \phi(x_i^T w_j) \quad L(\theta) = \frac{1}{2} \|u - y\|_2^2 \]

Nonconvex loss function!
Dynamics of training

Consider a two-hidden-layer network with width of hidden layers $m = 10, 100$
Dynamics of training

Consider a two-hidden-layer network with width of hidden layers $m = 10, 100$

$$\theta^{t+1} \leftarrow \theta^t - \eta \nabla L(\theta^t)$$
Dynamics of training

Consider a two-hidden-layer network with width of hidden layers $m = 10, 100$

\[ \theta^{t+1} \leftarrow \theta^t - \eta \nabla L(\theta^t) \]

Let’s visualize the evolution of the inner weights:

*Left: $m = 10$, Right: $m = 100*$

[Rajat VD, Github]
Linearizing the model

Taylor series expansion (look at tangents):

\[ u(x, \theta) \approx f(x, \theta_0) + \nabla_{\theta} f(x, \theta_0) (\theta - \theta_0) \]

Therefore, the network behaves like a local linear model with kernel feature map:

\[ g(x) = \nabla_{\theta} f(x, \theta_0) K(x, x') \]

Idea: Analyze gradient dynamics the same way as kernel methods!

Caveats: If Taylor series is no longer a good approximation, kernel \( K \) is no longer constant (depends on \( \theta \)). How to deal with this? Also, how to avoid non-degenerate kernels?
Taylor series expansion (look at **tangents**):

\[ u = f(x, \theta) \approx f(x, \theta_0) + \nabla_\theta f(x, \theta_0)^T (\theta - \theta_0) \]
Linearizing the model

Taylor series expansion (look at tangents):

\[ u = f(x, \theta) \approx f(x, \theta_0) + \nabla_\theta f(x, \theta_0)^T (\theta - \theta_0) \]

Therefore, the network behaves like a **local linear model** with kernel feature map:

\[ g(x) = \nabla_\theta f(x, \theta_0) \]
Linearizing the model

Taylor series expansion (look at tangents):

\[ u = f(x, \theta) \approx f(x, \theta_0) + \nabla_\theta f(x, \theta_0)^T (\theta - \theta_0) \]

Therefore, the network behaves like a **local linear model** with kernel feature map:

\[ g(x) = \nabla_\theta f(x, \theta_0) \]

\[ K(x, x') = [g(x)^T g(x')] \]
Linearizing the model

Taylor series expansion (look at tangents):

\[ u = f(x, \theta) \approx f(x, \theta_0) + \nabla_\theta f(x, \theta_0)^T (\theta - \theta_0) \]

Therefore, the network behaves like a local linear model with kernel feature map:

\[ g(x) = \nabla_\theta f(x, \theta_0) \]
\[ K(x, x') = [g(x)^T g(x')] \]

Idea: Analyze gradient dynamics the same way as kernel methods!
Linearizing the model

Taylor series expansion (look at tangents):

\[ u = f(x, \theta) \approx f(x, \theta_0) + \nabla_\theta f(x, \theta_0)^T (\theta - \theta_0) \]

Therefore, the network behaves like a local linear model with kernel feature map:

\[ g(x) = \nabla_\theta f(x, \theta_0) \]
\[ K(x, x') = [g(x)^T g(x')] \]

Idea: Analyze gradient dynamics the same way as kernel methods!

Caveats: If Taylor series is no longer a good approximation, kernel \( K \) is no longer constant (depends on \( \theta \)). How to deal with this? Also, how to avoid non-degenerate kernels?
At \( t = 0 \), initialize all \( a, w \sim \mathcal{N}(0, 1) \), let width \( m \to \infty \).
Key Idea (1): Random initialization, large width

At $t = 0$, initialize all $a, w \sim \mathcal{N}(0, 1)$, let width $m \to \infty$

$$K_0(x, x') \to K^\infty(x, x') = \mathbb{E}_{\theta \sim \mathcal{N}} \langle \nabla f(x, \theta), \nabla f(x', \theta) \rangle$$
Key Idea (1): Random initialization, large width

At $t = 0$, initialize all $a, w \sim \mathcal{N}(0, 1)$, let width $m \to \infty$

$$K_0(x, x') \to K^\infty(x, x') = \mathbb{E}_{\theta \sim \mathcal{N}} \langle \nabla f(x, \theta), \nabla f(x', \theta) \rangle$$

The kernel asymptotically converges to a \textit{deterministic} matrix called the Neural Tangent Kernel (NTK).
Key Idea (1): Random initialization, large width

At $t = 0$, initialize all $a, w \sim \mathcal{N}(0, 1)$, let width $m \to \infty$

$$K_0(x, x') \to K^\infty(x, x') = \mathbb{E}_{\theta \sim \mathcal{N}} \langle \nabla f(x, \theta), \nabla f(x', \theta) \rangle$$

The kernel asymptotically converges to a deterministic matrix called the Neural Tangent Kernel (NTK).

Moreover, we can get a non-asymptotic bound by Matrix Hoeffding:
Key Idea (1): Random initialization, large width

At $t = 0$, initialize all $a, w \sim \mathcal{N}(0, 1)$, let width $m \to \infty$

$$K_0(x, x') \to K^\infty(x, x') = \mathbb{E}_{\theta \sim \mathcal{N}} \langle \nabla f(x, \theta), \nabla f(x', \theta) \rangle$$

The kernel asymptotically converges to a deterministic matrix called the Neural Tangent Kernel (NTK). Moreover, we can get a non-asymptotic bound by Matrix Hoeffding:

**Theorem**

*[Du-Zhai-Poczos-Singh 2019]* If $m = \Omega(n^2 \log(n/\delta)/\lambda_0^2)$, then with probability at least $1 - \delta$:

$$\|K_0 - K^\infty\|_2 \leq 0.25\lambda_0,$$

$$\lambda_{\text{min}}(K) \geq 0.75\lambda_0,$$

where $\lambda_0$ is the minimum eigenvalue of $K^\infty$. 
Key Idea 2: Stability of NTK

When does the kernel not move too much from initialization?

When is the tangent approximation accurate (this is called the NTK regime or the lazy regime)?

Bound the relative change in the Jacobian as the model evolves:

One step of GD: model parameters change by

\[ \|y - u\|_2 \]

Propagate the errors to the Jacobian:

Change in \( \nabla \theta u(0) \)

\[ \approx \|u(0) - y\|_2 \|\nabla \theta u(0)\|_2 \|\nabla^2 \theta u(0)\|_2 = \kappa(\theta_0) \]

(again, matrix concentration – somewhat hairy expressions)

Norm of Hessian (numerator) \( \sim O(1) \), norm of gradient (denominator) \( \sim O(1/\sqrt{m}) \)

So \( \kappa(\theta_0) = O(1/m)! \)
Key Idea 2: Stability of NTK

When does the kernel not move too much from initialization? When is the tangent approximation accurate (this is called the NTK regime or the lazy regime)
Key Idea 2: Stability of NTK

When does the kernel not move too much from initialization? When is the tangent approximation accurate (this is called the NTK regime or the lazy regime)

Bound the relative change in the Jacobian as the model evolves:

\[ \| \frac{\partial y}{\partial u}(0) - y \|_2 \approx \frac{1}{\| \nabla_{\theta} u(0) \|_2} \| \nabla^2_{\theta} u(0) \|_2 = \kappa(\theta_0) \]

(norm of Hessian (numerator) \(\sim O(1)\), norm of gradient (denominator) \(\sim O(1/\sqrt{m})\))

So \(\kappa(\theta_0) = O(1/m)\)!
Key Idea 2: Stability of NTK

When does the kernel not move too much from initialization? When is the tangent approximation accurate (this is called the NTK regime or the lazy regime)

Bound the relative change in the Jacobian as the model evolves:
One step of GD: model parameters change by $\frac{\|y-u\|_2}{\|\nabla_\theta u(0)\|}$
Key Idea 2: Stability of NTK

When does the kernel not move too much from initialization? When is the tangent approximation accurate (this is called the NTK regime or the lazy regime)

Bound the relative change in the Jacobian as the model evolves:

One step of GD: model parameters change by \( \frac{\|y-u\|_2}{\|\nabla_\theta u(0)\|} \)

Propagate the errors to the Jacobian:

\[
\text{Change in } \nabla_\theta u \approx \frac{\|u(0) - y\|_2 \|\nabla^2_\theta u(0)\|_2}{\|\nabla_\theta u(0)\|_2 \|\nabla_\theta u(0)\|_2}
\]
Key Idea 2: Stability of NTK

When does the kernel not move too much from initialization? When is the tangent approximation accurate (this is called the NTK regime or the lazy regime)

Bound the relative change in the Jacobian as the model evolves:
One step of GD: model parameters change by
Propagate the errors to the Jacobian:

\[
\text{Change in } \nabla_\theta u \approx \frac{\| u(0) - y \|_2}{\| \nabla_\theta u(0) \|_2} \frac{\| \nabla^2_\theta u(0) \|_2}{\| \nabla_\theta u(0) \|_2} = \kappa(\theta_0)
\]
Key Idea 2: Stability of NTK

When does the kernel not move too much from initialization? When is the tangent approximation accurate (this is called the NTK regime or the lazy regime)

Bound the relative change in the Jacobian as the model evolves:
One step of GD: model parameters change by
\[
\frac{\|\mathbf{y} - \mathbf{u}\|_2}{\|\nabla_\theta \mathbf{u}(0)\|}
\]

Propagate the errors to the Jacobian:

\[
\text{Change in } \nabla_\theta \mathbf{u} \approx \frac{\|\mathbf{u}(0) - \mathbf{y}\|_2}{\|\nabla_\theta \mathbf{u}(0)\|} \frac{\|\nabla^2_\theta \mathbf{u}(0)\|_2}{\|\nabla_\theta \mathbf{u}(0)\|_2} = \kappa(\theta_0)
\]

(again, matrix concentration – somewhat hairy expressions)

norm of Hessian (numerator) \(\sim O(1)\), norm of gradient (denominator) \(\sim O\left(\frac{1}{\sqrt{m}}\right)\)
Key Idea 2: Stability of NTK

When does the kernel not move too much from initialization? When is the tangent approximation accurate (this is called the NTK regime or the lazy regime)

Bound the relative change in the Jacobian as the model evolves:
One step of GD: model parameters change by $\frac{\|y-u\|_2}{\|\nabla_\theta u(0)\|}$

Propagate the errors to the Jacobian:

\[
\text{Change in } \nabla_\theta u \approx \frac{\|u(0) - y\|_2}{\|\nabla_\theta u(0)\|_2} \frac{\|\nabla^2_\theta u(0)\|_2}{\|\nabla_\theta u(0)\|_2} = \kappa(\theta_0)
\]

(again, matrix concentration – somewhat hairy expressions)

norm of Hessian (numerator) $\sim O(1)$,
norm of gradient (denominator) $\sim O\left(\frac{1}{\sqrt{m}}\right)$

So $\kappa(\theta_0) = O\left(\frac{1}{m}\right)$!
Key quantity: **NTK**

\[ K^\infty(x, x') = \mathbb{E}_{\theta \sim \mathcal{N}} \langle \nabla f(x, \theta), \nabla f(x', \theta) \rangle \]

To understand training dynamics of simple two layer network:

1. Show that NTK is PSD: \( \lambda_{\text{min}}(K^\infty) > 0 \)

Similar arguments can be applied to deep (but wide) networks [Arora-Du-Hu-Li-Salakhutdinov-Wang, 2019]
To recap

Key quantity: NTK

\[ K^\infty(x, x') = \mathbb{E}_{\theta \sim \mathcal{N}} \langle \nabla f(x, \theta), \nabla f(x', \theta) \rangle \]

To understand training dynamics of simple two layer network:

1. Show that NTK is PSD: \( \lambda_{\text{min}}(K^\infty) > 0 \)

2. Show that kernel at initialization is close to NTK:
   \[ K_0(x, x') \approx K^\infty(x, x') \]
To recap

Key quantity: NTK

\[ K^\infty(x, x') = \mathbb{E}_{\theta \sim \mathcal{N}} \langle \nabla f(x, \theta), \nabla f(x', \theta) \rangle \]

To understand training dynamics of simple two layer network:

1. Show that NTK is PSD: \( \lambda_{\min}(K^\infty) > 0 \)
2. Show that kernel at initialization is close to NTK:
   \[ K_0(x, x') \approx K^\infty(x, x') \]
3. Show that kernel does not move too much from initialization:
   \[ K_t(x, x') \approx K_0(x, x') \]

Similar arguments can be applied to deep (but wide) networks [Arora-Du-Hu-Li-Salakhutdinov-Wang, 2019]
To recap

Key quantity: NTK

$$K^\infty(x, x') = \mathbb{E}_{\theta \sim \mathcal{N}} \langle \nabla f(x, \theta), \nabla f(x', \theta) \rangle$$

To understand training dynamics of simple two layer network:

1. Show that NTK is PSD: $\lambda_{\text{min}}(K^\infty) > 0$

2. Show that kernel at initialization is close to NTK:
   $$K_0(x, x') \approx K^\infty(x, x')$$

3. Show that kernel does not move too much from initialization:
   $$K_t(x, x') \approx K_0(x, x')$$

$1 + 2 + 3$: gradient flow for two layer networks converges to zero loss

Similar arguments can be applied to deep (but wide) networks

[Arora-Du-Hu-Li-Salakhutdinov-Wang, 2019]
Application to Autoencoders
This talk: Unsupervised models

- Autoencoders
This talk: Unsupervised models

- **Autoencoders**
  
  popular building blocks of deep networks
This talk: Unsupervised models

- **Autoencoders**
  
  popular building blocks of deep networks

Often used when very few labeled data points are available.
Autoencoders in practice

Application: Representation learning in images
Autoencoders in practice

Application: Representation learning in images

Goal: Learn a basis in patch space
Autoencoders in practice

Application: Representation learning in images

Goal: Learn a basis in patch space
Autoencoders in practice

Application: Representation learning in images

Goal: Learn a basis in patch space
Autoencoders in practice

Autoencoder weights (visualized as images)
Autoencoders in practice

Autoencoder weights (visualized as images)

Interpretable features (learned features are edges)
Two-layer autoencoders

Given \( n \) samples \( \{x_i\}_{i=1}^n \in \mathbb{R}^d \), the reconstruction loss is

\[
\min_{W, A} L(W, A) = \frac{1}{2} \sum_{i=1}^n \|x_i - \hat{x}_i\|_2^2 = \frac{1}{2} \sum_{i=1}^n \|x_i - A\phi(W^\top x_i)\|_2^2
\]

ReLU activation \( \phi \)

\( m \gg d, n \)
Two-layer autoencoders

Given $n$ samples $\{x_i\}_{i=1}^n \in \mathbb{R}^d$, the reconstruction loss is

$$\min_{W, A \in \mathbb{R}^{d \times m}} L(W, A) = \frac{1}{2} \sum_{i=1}^n \|x_i - \hat{x}_i\|^2 = \frac{1}{2} \sum_{i=1}^n \left\| x_i - \frac{1}{\sqrt{md}} A\phi(W^\top x_i) \right\|^2$$
Learning algorithm

Gradient descent:

- initialize \( \theta^0 = \{ w^0_{ij}, a^0_{ij} \} \sim \mathcal{N}(0, 1) \)
- perform gradient descent over \( \theta = (W, A) \)

\[
\theta^{t+1} = \theta^t - \eta \nabla_\theta L(\theta^t), \ t = 0, 1, \ldots
\]

Two scenarios

1. Weakly trained: only optimize over the first layer
2. Jointly trained: optimize over both layers
Assumptions

- $\|x_i\|_2 = 1$ for $i = 1, \ldots, n$

- Take $w \sim \mathcal{N}(0, I)$ and compute feature maps for $i \in [n]$:
  
  - $z_i = \phi(w^\top x_i)$, assume $\lambda_{\text{min}}(\mathbb{E}_w[\phi(X^\top w)\phi(w^\top X)]) \geq \lambda_0$
  
  - $\tilde{x}_i = [w^\top x_i \geq 0] x_i$, assume $\lambda_{\text{min}}(\mathbb{E}_w[\tilde{X}^\top \tilde{X}]) \geq \lambda_0$

then the autoencoder NTK:

$$K^\infty = \mathbb{E}_w[\phi(X^\top w)\phi(w^\top X)] + \mathbb{E}_w[\tilde{X}^\top \tilde{X}]$$ is p.d.
Assumptions

- \( \|x_i\|_2 = 1 \) for \( i = 1, \ldots, n \)

- Take \( w \sim \mathcal{N}(0, I) \) and compute feature maps for \( i \in [n] \):
  - \( z_i = \phi(w^\top x_i) \), assume \( \lambda_{\text{min}}(\mathbb{E}_w[\phi(X^\top w)\phi(w^\top X)]) \geq \lambda_0 \)
  - \( \tilde{x}_i = [w^\top x_i \geq 0]x_i \), assume \( \lambda_{\text{min}}(\mathbb{E}_w[\tilde{X}^\top \tilde{X}]) \geq \lambda_0 \)

then the autoencoder NTK:

\[
K^\infty = \mathbb{E}_w[\phi(X^\top w)\phi(w^\top X)] + \mathbb{E}_w[\tilde{X}^\top \tilde{X}] \text{ is p.d.}
\]

- Oymak et al., 2019: \( \Delta = \min_{i \neq j} \|x_i - x_j\| > 0 \) implies \( \lambda_0 > 0 \)
Result 1: Weakly-trained setting

\[
\min_{W,A \in \mathbb{R}^{d \times m}} L(W, A) = \frac{1}{2} \sum_{i=1}^{n} \left\| x_i - \frac{1}{\sqrt{md}} A \phi(W^\top x_i) \right\|^2,
\]

- \( n \): number of samples, \( d \): input dimension
- \( m \): size of hidden layer
- \( \lambda_0 = \lambda_{\min}(K^\infty) \), and \( \lambda_n = \|X^\top X\| \)

**Theorem**

*For any \( \delta \in (0,1) \) and for a sufficiently large constant \( C > 0 \). If \( m \geq C \frac{n^5 d^4 \lambda_n}{\lambda_0^4 \delta^3} \), then gradient descent over \( W \) linearly converges to a global minimum with probability at least \( 1 - \delta \).*

\[
\left\| X - \hat{X}(t) \right\|_F^2 \leq (1 - \frac{\eta \lambda_0}{2d})^t \left\| X - \hat{X}(0) \right\|_F^2
\]

*for \( k = 0, 1, \ldots \)
Result 2: Jointly trained setting

\[
\min_{W,A \in \mathbb{R}^{d \times m}} L(W, A) = \frac{1}{2} \sum_{i=1}^{n} \left\| x_i - \frac{1}{\sqrt{md}} A \phi(W^\top x_i) \right\|^2,
\]

- \( n \): number of samples, \( d \): input dimension
- \( m \): size of hidden layer
- \( \lambda_0 = \lambda_{\text{min}}(K^\infty) \), and \( \lambda_n = \|X^\top X\| \)

**Theorem**

For any \( \delta \in (0, 1) \) and some large constant \( C > 0 \). If \( m \geq C^{nd\lambda_0^3 \frac{\log(1/\delta)}{\lambda_0^4 \delta^2}} \), then gradient descent with \( \eta = \Theta(\frac{\lambda_0}{n\lambda_n}) \) linearly converges to a global minimum with probability at least \( 1 - \delta \):

\[
\left\| X - \hat{X}(t) \right\|_F^2 \leq (1 - \frac{\eta \lambda_0}{2d})^t \left\| X - \hat{X}(0) \right\|_F^2, \; t = 0, 1, \ldots
\]
<table>
<thead>
<tr>
<th>Regime</th>
<th>Reference</th>
<th>Single output</th>
<th>Multiple output</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Du et al., 2019</td>
<td>$C \frac{n^6}{\lambda_0^4 \delta^3}$</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>This work</td>
<td>$C \frac{n^3 \lambda}{\lambda_0^4 \delta^3}$</td>
<td>$C \frac{n^5 d^4 \lambda}{\lambda_0^4 \delta^3}$</td>
</tr>
<tr>
<td></td>
<td>Oymak et al., 2019</td>
<td>$C \frac{n \lambda^3}{\lambda_0^4}$</td>
<td>X</td>
</tr>
<tr>
<td>Jointly-trained</td>
<td>Du et al., 2019</td>
<td>$C \frac{n^6 \log(m/\delta)}{\lambda_0^4 \delta^3}$</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>Zou et al., 2019</td>
<td>$C \frac{n \lambda^3}{\Delta^4}$</td>
<td>$C \frac{n^8 d}{\Delta^4}$</td>
</tr>
<tr>
<td></td>
<td>This work</td>
<td>$C \frac{n \lambda^3 \log(1/\delta)}{\lambda_0^4 \delta^2}$</td>
<td>$C \frac{n d \lambda^3 \log(1/\delta)}{\lambda_0^4 \delta^2}$</td>
</tr>
</tbody>
</table>

- $n$: number of samples, $d$: input dimension
- $\lambda_0 = \lambda_{\min}(K^{\infty})$, and $\lambda_n = \|X^\top X\|$
- $\Delta = \min_{i \neq j} \|x_i - x_j\|$
Inductive bias of autoencoders

Are they actually learning any useful features when over-parameterized?

If yes, then great! They will likely generalize to unseen inputs.

If not, then are they merely memorizing the training samples?
Are they actually learning any useful features when over-parameterized?

If yes, then great! They will likely generalize to unseen inputs.
Inductive bias of autoencoders

Are they actually learning any useful features when over-parameterized?

If yes, then great! They will likely generalize to unseen inputs.

If not, then are they merely memorizing the training samples?
Inductive bias (cont.)

- autoencoders \((m = 2048)\) trained on a single digit 7 by GD
- first row shows the test samples
- deeper networks exhibit higher memorization
Inductive bias: Theory
Inductive bias: Theory

Theorem

Let $\eta_{\text{critical}} = (\lambda_n + \lambda_0)^{-1}$. Under gradient descent with $\eta < \eta_{\text{critical}},$ as the width $m \to \infty$, the autoencoder output $u_t(x)$ for any test sample $x \in \mathbb{R}^d$ at step $t$ converges to $\mu_t(x) + \gamma_t(x)$:

$$
\mu_t(x) = \sum_{i=1}^n k_{x,x_i}^\infty (K^\infty)^{-1}(I - e^{-K^\infty t})x_i,
$$

$$
\gamma_t(x) = f_0(x) - \sum_{i=1}^n k_{x,x_i}^\infty (K^\infty)^{-1}(I - e^{-K^\infty t})f_0(x_i)
$$

- $K_{x,x_i} = \mathbb{E}_w[\phi(x^\top w)\phi(w^\top x_i) + \tilde{x}^\top \tilde{x}_i], \tilde{x} = [w^\top x \geq 0]x$

- $f_0(x)$ is the reconstruction of $x$ at initialization
Conclusions

• We prove the linear convergence of gradient descent for over-parameterized autoencoders
• We show a significant improvement in the bound for over-parameterization by joint training
• Our analysis can be extended to deep autoencoders

Open questions:

• convergence of GD for deep over-parameterized autoencoders with bottleneck layer, width $k < n, d$
• generalization bounds for autoencoders