Neural Tangent Kernel

Comprehending Convergence & Generalization in Neural Networks[1]

J. Setpal

March 6, 2025

MACHINE LEARNING
@ PURDUE

[1] Jacot, Gabriel, Hongler. [NIPS 2018]

Outline

1. Background & Intuition

2. Neural Tangent Kernel

Outline

1. Background & Intuition

2. Neural Tangent Kernel

Problem Setting

Given network $f_\theta : \mathbb{R}^d \to \mathbb{R}$, we have two distinctive regimes:

Underparameterized Learning: $\boldsymbol{n \geq p}, \quad \theta \in \mathbb{R}^p, \quad \mathcal{D} = \{(x_i, y_i)\}_{i=1}^n$

Overparameterized Learning: $\boldsymbol{p \geq n}, \quad \theta \in \mathbb{R}^p, \quad \mathcal{D} = \{(x_i, y_i)\}_{i=1}^n$

Problem Setting

Given network $f_\theta : \mathbb{R}^d \to \mathbb{R}$, we have two distinctive regimes:
Underparameterized Learning: $\boldsymbol{n} \geq \boldsymbol{p}, \quad \theta \in \mathbb{R}^p, \quad \mathcal{D} = \{(x_i, y_i)\}_{i=1}^n$
Overparameterized Learning: $\boldsymbol{p} \geq \boldsymbol{n}, \quad \theta \in \mathbb{R}^p, \quad \mathcal{D} = \{(x_i, y_i)\}_{i=1}^n$

Either way, we optimize θ to minimize an empirical loss (here, quadratic):

$$\mathcal{L}(\theta, \mathcal{D}) = \frac{1}{n} \sum_{i=1}^{n} \ell(f_\theta(x_i), y_i) = \frac{1}{n} \| f_\theta(\boldsymbol{x}) - \boldsymbol{y} \|_2^2 \tag{1}$$

Problem Setting

Given network $f_\theta : \mathbb{R}^d \to \mathbb{R}$, we have two distinctive regimes:
Underparameterized Learning: $\boldsymbol{n \geq p}$, $\quad \theta \in \mathbb{R}^p$, $\quad \mathcal{D} = \{(x_i, y_i)\}_{i=1}^n$
Overparameterized Learning: $\boldsymbol{p \geq n}$, $\quad \theta \in \mathbb{R}^p$, $\quad \mathcal{D} = \{(x_i, y_i)\}_{i=1}^n$

Either way, we optimize θ to minimize an empirical loss (here, quadratic):

$$\mathcal{L}(\theta, \mathcal{D}) = \frac{1}{n} \sum_{i=1}^{n} \ell(f_\theta(x_i), y_i) = \frac{1}{n} \| f_\theta(\boldsymbol{x}) - \boldsymbol{y} \|_2^2 \tag{1}$$

Q: Why does overparameterized learning generalize?

Problem Setting

Given network $f_\theta : \mathbb{R}^d \to \mathbb{R}$, we have two distinctive regimes:
Underparameterized Learning: $\boldsymbol{n \geq p}$, $\quad \theta \in \mathbb{R}^p$, $\quad \mathcal{D} = \{(x_i, y_i)\}_{i=1}^n$
Overparameterized Learning: $\boldsymbol{p \geq n}$, $\quad \theta \in \mathbb{R}^p$, $\quad \mathcal{D} = \{(x_i, y_i)\}_{i=1}^n$

Either way, we optimize θ to minimize an empirical loss (here, quadratic):

$$\mathcal{L}(\theta, \mathcal{D}) = \frac{1}{n} \sum_{i=1}^{n} \ell(f_\theta(x_i), y_i) = \frac{1}{n} \| f_\theta(\boldsymbol{x}) - \boldsymbol{y} \|_2^2 \tag{1}$$

Q: Why does overparameterized learning generalize?
A: NTK's approach:

a. Construct an analogy to a simpler paradigm (kernel methods).

Problem Setting

Given network $f_\theta : \mathbb{R}^d \to \mathbb{R}$, we have two distinctive regimes:
Underparameterized Learning: $\boldsymbol{n \geq p}$, $\quad \theta \in \mathbb{R}^p$, $\quad \mathcal{D} = \{(x_i, y_i)\}_{i=1}^n$
Overparameterized Learning: $\boldsymbol{p \geq n}$, $\quad \theta \in \mathbb{R}^p$, $\quad \mathcal{D} = \{(x_i, y_i)\}_{i=1}^n$

Either way, we optimize θ to minimize an empirical loss (here, quadratic):

$$\mathcal{L}(\theta, \mathcal{D}) = \frac{1}{n} \sum_{i=1}^{n} \ell(f_\theta(x_i), y_i) = \frac{1}{n} \| f_\theta(\boldsymbol{x}) - \boldsymbol{y} \|_2^2 \tag{1}$$

Q: Why does overparameterized learning generalize?
A: NTK's approach:

a. Construct an analogy to a simpler paradigm (kernel methods).
b. Prove the analogy actually holds for sufficiently wide networks.

Problem Setting

Given network $f_\theta : \mathbb{R}^d \to \mathbb{R}$, we have two distinctive regimes:
Underparameterized Learning: $\boldsymbol{n \geq p}$, $\quad \theta \in \mathbb{R}^p$, $\quad \mathcal{D} = \{(x_i, y_i)\}_{i=1}^n$
Overparameterized Learning: $\boldsymbol{p \geq n}$, $\quad \theta \in \mathbb{R}^p$, $\quad \mathcal{D} = \{(x_i, y_i)\}_{i=1}^n$

Either way, we optimize θ to minimize an empirical loss (here, quadratic):

$$\mathcal{L}(\theta, \mathcal{D}) = \frac{1}{n} \sum_{i=1}^{n} \ell(f_\theta(x_i), y_i) = \frac{1}{n} \| f_\theta(\boldsymbol{x}) - \boldsymbol{y} \|_2^2 \tag{1}$$

Q: Why does overparameterized learning generalize?
A: NTK's approach:

a. Construct an analogy to a simpler paradigm (kernel methods).
b. Prove the analogy actually holds for sufficiently wide networks.
c. Use the analysis from the kernel method to understand training dynamics of the neural network.

Pre-Requisite 1/3 – Taylor Series

We can approximate function g using a polynomial via the Taylor Series:

$$P_a(x) = \sum_{i=0}^{\infty} \frac{g^{(n)}(a)}{n!}(x-a)^n \tag{2}$$

Pre-Requisite 1/3 – Taylor Series

We can approximate function g using a polynomial via the Taylor Series:

$$P_a(x) = \sum_{i=0}^{\infty} \frac{g^{(n)}(a)}{n!}(x-a)^n \tag{2}$$

The first-order Taylor Expansion is a linear approximation of the function:

$$P_a(x) = g(a) + g'(a)(x-a) \tag{3}$$

Pre-Requisite 1/3 – Taylor Series

We can approximate function g using a polynomial via the Taylor Series:

$$P_a(x) = \sum_{i=0}^{\infty} \frac{g^{(n)}(a)}{n!}(x-a)^n \tag{2}$$

The first-order Taylor Expansion is a linear approximation of the function:

$$P_a(x) = g(a) + g'(a)(x-a) \tag{3}$$

We will use the first-order approximation to model **training dynamics in neural networks**.

Pre-Requisite 2/3 – Operator Norm

The operator norm of matrix A is the maximum amount of stretching that A can do to arbitrary vector x:

$$A \in \mathbb{R}^{m \times n}, \quad x \in \mathbb{R}^n \tag{4}$$

$$\|A\|_{\text{op}} = \max_{x \in \mathbb{R}^n} \frac{\|Ax\|_p}{\|x\|_q} \tag{5}$$

Usually $p = q = 2$.

Pre-Requisite $^3/_3$ – Kernel Methods

$K : \mathbb{R}^m \times \mathbb{R}^m \to \mathbb{R}$ is a kernel if $\exists \phi : \mathcal{X} \to \mathcal{H}$ s.t:

$$K(x, x') = \phi(x)^T \phi(x') = \langle \phi(x), \phi(x') \rangle \tag{6}$$

Any PSD matrix defines a kernel.

Pre-Requisite $^3/_3$ – Kernel Methods

$K : \mathbb{R}^m \times \mathbb{R}^m \to \mathbb{R}$ is a kernel if $\exists \phi : \mathcal{X} \to \mathcal{H}$ s.t:

$$K(x, x') = \phi(x)^T \phi(x') = \langle \phi(x), \phi(x') \rangle \tag{6}$$

Any PSD matrix defines a kernel.

A kernel methods are **instance-based** learners, instead of learning parameters over input features, we learn parameters over data-pairs, and interpolate using the kernel function to predict for the unseen sample.

Pre-Requisite $^3/_3$ – Kernel Methods

$K : \mathbb{R}^m \times \mathbb{R}^m \to \mathbb{R}$ is a kernel if $\exists \phi : \mathcal{X} \to \mathcal{H}$ s.t:

$$K(x, x') = \phi(x)^T \phi(x') = \langle \phi(x), \phi(x') \rangle \tag{6}$$

Any PSD matrix defines a kernel.

A kernel methods are **instance-based** learners, instead of learning parameters over input features, we learn parameters over data-pairs, and interpolate using the kernel function to predict for the unseen sample.

As an example:

$$\hat{y} = \textbf{sgn} \sum_{i=1}^{n} w_i y_i k(x, x') \tag{7}$$

Outline

1 Background & Intuition

2 Neural Tangent Kernel

Neural Network Setup

We have fully-connected network f_θ with $n_0, \ldots, n_L$ neurons in each layer, activation function $\sigma : \mathbb{R} \to \mathbb{R}$ with pre-activations $\tilde{\alpha}$ and activations α defined recursively as follows:

$$\alpha^{(0)}(x;\ \theta) = x \tag{8}$$

$$\tilde{\alpha}^{(\ell+1)}(x;\ \theta) = \frac{1}{\sqrt{n_\ell}} W^{(\ell)} \alpha^{(\ell)}(x;\theta) + \beta b^{(\ell)} \tag{9}$$

$$\alpha^{(\ell)}(x;\ \theta) = \sigma(\tilde{\alpha}^{(\ell)}(x;\theta)) \tag{10}$$

A **realization function** $F^{(L)} : \mathbb{R}^p \to \mathcal{F}$ maps parameters to a function.

Neural Network Setup

We have fully-connected network f_θ with $n_0, \ldots, n_L$ neurons in each layer, activation function $\sigma : \mathbb{R} \to \mathbb{R}$ with pre-activations $\tilde{\alpha}$ and activations α defined recursively as follows:

$$\alpha^{(0)}(x;\ \theta) = x \tag{8}$$

$$\tilde{\alpha}^{(\ell+1)}(x;\ \theta) = \frac{1}{\sqrt{n_\ell}} W^{(\ell)} \alpha^{(\ell)}(x;\theta) + \beta b^{(\ell)} \tag{9}$$

$$\alpha^{(\ell)}(x;\ \theta) = \sigma(\tilde{\alpha}^{(\ell)}(x;\theta)) \tag{10}$$

A **realization function** $F^{(L)} : \mathbb{R}^p \to \mathcal{F}$ maps parameters to a function.

At initialization, $\theta \sim \mathcal{N}(0, I_p)$.

Neural Network Setup

We have fully-connected network f_θ with $n_0, \ldots, n_L$ neurons in each layer, activation function $\sigma : \mathbb{R} \to \mathbb{R}$ with pre-activations $\tilde{\alpha}$ and activations α defined recursively as follows:

$$\alpha^{(0)}(x;\ \theta) = x \tag{8}$$

$$\tilde{\alpha}^{(\ell+1)}(x;\ \theta) = \frac{1}{\sqrt{n_\ell}} W^{(\ell)} \alpha^{(\ell)}(x;\theta) + \beta b^{(\ell)} \tag{9}$$

$$\alpha^{(\ell)}(x;\ \theta) = \sigma(\tilde{\alpha}^{(\ell)}(x;\theta)) \tag{10}$$

A **realization function** $F^{(L)} : \mathbb{R}^p \to \mathcal{F}$ maps parameters to a function.

At initialization, $\theta \sim \mathcal{N}(0, I_p)$. As a result at initialization, our network is equivalent to a gaussian process.

Neural Network Setup

We have fully-connected network f_θ with $n_0, \ldots, n_L$ neurons in each layer, activation function $\sigma : \mathbb{R} \to \mathbb{R}$ with pre-activations $\tilde{\alpha}$ and activations α defined recursively as follows:

$$\alpha^{(0)}(x;\ \theta) = x \tag{8}$$

$$\tilde{\alpha}^{(\ell+1)}(x;\ \theta) = \frac{1}{\sqrt{n_\ell}} W^{(\ell)} \alpha^{(\ell)}(x;\theta) + \beta b^{(\ell)} \tag{9}$$

$$\alpha^{(\ell)}(x;\ \theta) = \sigma(\tilde{\alpha}^{(\ell)}(x;\theta)) \tag{10}$$

A **realization function** $F^{(L)} : \mathbb{R}^p \to \mathcal{F}$ maps parameters to a function.

At initialization, $\theta \sim \mathcal{N}(0, I_p)$. As a result at initialization, our network is equivalent to a gaussian process.

Aside

Factors $1/\sqrt{n_\ell}$, β scale gradients & enable consistent aymptotic behavior.

Studying Convergence in Function Space

Notice that quadratic loss[2] is **convex w.r.t. function space** but often **non-convex w.r.t. parameter space**.

$$C : \mathcal{F} \to \mathbb{R} \qquad \text{Convex :D} \tag{11}$$

$$C \circ F^{(L)} : \mathbb{R}^p \to \mathbb{R} \qquad \text{Highly Non-Convex D:} \tag{12}$$

[2] also most others

Studying Convergence in Function Space

Notice that quadratic loss[2] is **convex w.r.t. function space** but often **non-convex w.r.t. parameter space**.

$$C : \mathcal{F} \to \mathbb{R} \qquad \text{Convex :D} \tag{11}$$

$$C \circ F^{(L)} : \mathbb{R}^p \to \mathbb{R} \qquad \text{Highly Non-Convex D:} \tag{12}$$

Now, if we could analyze training in the convex setting, we can obtain meaningful insight about convergence of these networks.

[2] also most others

Studying Convergence in Function Space

Notice that quadratic loss[2] is **convex w.r.t. function space** but often **non-convex w.r.t. parameter space**.

$$C : \mathcal{F} \to \mathbb{R} \qquad \text{Convex :D} \tag{11}$$

$$C \circ F^{(L)} : \mathbb{R}^p \to \mathbb{R} \qquad \text{Highly Non-Convex D:} \tag{12}$$

Now, if we could analyze training in the convex setting, we can obtain meaningful insight about convergence of these networks.
Q_1: How can we do that?
A_1: By constructing a first-order approximation of training dynamics.

[2] also most others

Studying Convergence in Function Space

Notice that quadratic loss[2] is **convex w.r.t. function space** but often **non-convex w.r.t. parameter space**.

$$C : \mathcal{F} \to \mathbb{R} \qquad \text{Convex :D} \tag{11}$$

$$C \circ F^{(L)} : \mathbb{R}^p \to \mathbb{R} \qquad \text{Highly Non-Convex D:} \tag{12}$$

Now, if we could analyze training in the convex setting, we can obtain meaningful insight about convergence of these networks.
Q_1: How can we do that?
A_1: By constructing a first-order approximation of training dynamics.

Q_2: Why can we do that?

[2] also most others

Studying Convergence in Function Space

Notice that quadratic loss[2] is **convex w.r.t. function space** but often **non-convex w.r.t. parameter space**.

$$C : \mathcal{F} \to \mathbb{R} \qquad \text{Convex :D} \tag{11}$$

$$C \circ F^{(L)} : \mathbb{R}^p \to \mathbb{R} \qquad \text{Highly Non-Convex D:} \tag{12}$$

Now, if we could analyze training in the convex setting, we can obtain meaningful insight about convergence of these networks.
Q_1: How can we do that?
A_1: By constructing a first-order approximation of training dynamics.

Q_2: Why can we do that?
A_2: In the overparameterized regime, learning is *lazy*.

[2] also most others

Studying Convergence in Function Space

Notice that quadratic loss[2] is **convex w.r.t. function space** but often **non-convex w.r.t. parameter space**.

$$C : \mathcal{F} \to \mathbb{R} \qquad \text{Convex :D} \tag{11}$$

$$C \circ F^{(L)} : \mathbb{R}^p \to \mathbb{R} \qquad \text{Highly Non-Convex D:} \tag{12}$$

Now, if we could analyze training in the convex setting, we can obtain meaningful insight about convergence of these networks.
Q_1: How can we do that?
A_1: By constructing a first-order approximation of training dynamics.

Q_2: Why can we do that?
A_2: In the overparameterized regime, learning is *lazy*.

We'll answer the questions in detail, but in reverse order.

[2] also most others

Lazy Learning

For L-smooth f, gradient descent using small η guarantees convergent loss.

$$\mathcal{L}(\theta_{t+1}, \mathcal{D}) \leq \mathcal{L}(\theta_t, \mathcal{D}), \quad \eta \leq 1/L \tag{13}$$

$$\|\theta - \theta_0\| \approx \frac{\|f_{\theta_0}(x) - y\|}{\|\nabla_\theta f_{\theta_0}(x)\|_{\text{op}}} \tag{14}$$

Lazy Learning

For L-smooth f, gradient descent using small η guarantees convergent loss.

$$\mathcal{L}(\theta_{t+1}, \mathcal{D}) \leq \mathcal{L}(\theta_t, \mathcal{D}), \quad \eta \leq 1/L \tag{13}$$

$$\|\theta - \theta_0\| \approx \frac{\|f_{\theta_0}(x) - y\|}{\|\nabla_\theta f_{\theta_0}(x)\|_{\text{op}}} \tag{14}$$

Takeaway: With gaussian initialization, as $\theta \in \mathbb{R}^p$ s.t. $p \to \infty$

$$\exists \theta^* \quad \text{s.t.} \quad \mathcal{L}(f_{\theta^*}, \mathcal{D}) = \min_\theta \mathcal{L}(f_\theta, \mathcal{D}) \quad \text{and} \quad \|\theta^* - \theta_0\| < B \ll \infty \tag{15}$$

Lazy Learning

For L-smooth f, gradient descent using small η guarantees convergent loss.

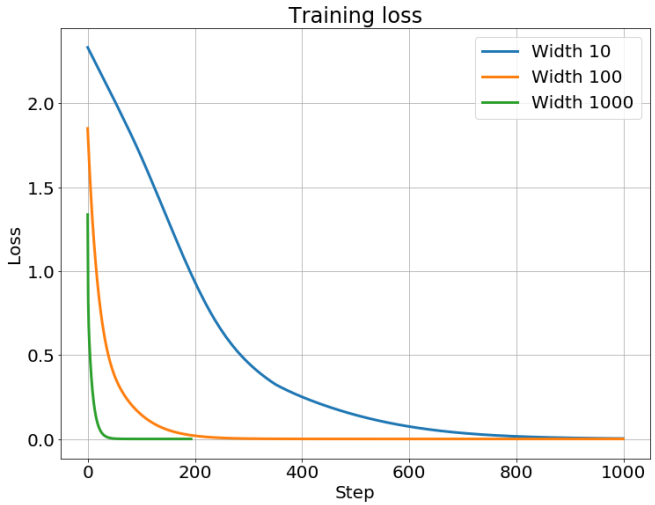
$$\mathcal{L}(\theta_{t+1}, \mathcal{D}) \leq \mathcal{L}(\theta_t, \mathcal{D}), \quad \eta \leq 1/L \tag{13}$$

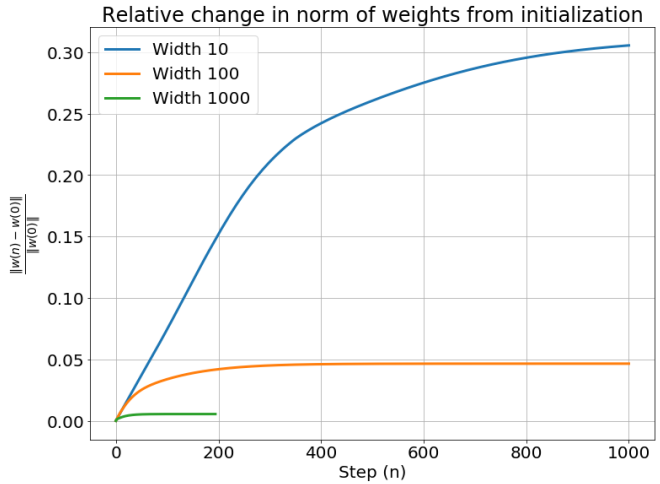
$$\|\theta - \theta_0\| \approx \frac{\|f_{\theta_0}(x) - y\|}{\|\nabla_\theta f_{\theta_0}(x)\|_{\text{op}}} \tag{14}$$

Takeaway: With gaussian initialization, as $\theta \in \mathbb{R}^p$ s.t. $p \to \infty$

$$\exists \theta^* \quad \text{s.t.} \quad \mathcal{L}(f_{\theta^*}, \mathcal{D}) = \min_\theta \mathcal{L}(f_\theta, \mathcal{D}) \quad \text{and} \quad \|\theta^* - \theta_0\| < B \ll \infty \tag{15}$$

We can see this by the change of $\|\theta\|$ as $\theta \in \mathbb{R}^{10}, \mathbb{R}^{100}, \mathbb{R}^{1000}$:





Linear Approximation of f_θ

Applying first-order taylor expansion of f with respect to the evolution of parameters θ, we have:

$$g_\theta(x) := f_{\theta_0}(x) + \langle \nabla_\theta f_{\theta_0}(x), \underbrace{\theta - \theta_0}_{\Delta\theta} \rangle \tag{16}$$

Linear Approximation of f_θ

Applying first-order taylor expansion of f with respect to the evolution of parameters θ, we have:

$$g_\theta(x) := f_{\theta_0}(x) + \langle \nabla_\theta f_{\theta_0}(x), \underbrace{\theta - \theta_0}_{\Delta\theta} \rangle \tag{16}$$

For convenience of analysis, we can set θ_0 s.t. $f_{\theta_0}(x) = 0 \ \forall x$.

Linear Approximation of f_θ

Applying first-order taylor expansion of f with respect to the evolution of parameters θ, we have:

$$g_\theta(x) := f_{\theta_0}(x) + \langle \nabla_\theta f_{\theta_0}(x), \underbrace{\theta - \theta_0}_{\Delta\theta} \rangle \tag{16}$$

For convenience of analysis, we can set θ_0 s.t. $f_{\theta_0}(x) = 0 \ \forall x$. Now,

$$g_\theta(x) := \langle \underbrace{\nabla_\theta f_{\theta_0}(x)}_{\phi(x)}, \Delta\theta \rangle \tag{17}$$

Linear Approximation of f_θ

Applying first-order taylor expansion of f with respect to the evolution of parameters θ, we have:

$$g_\theta(x) := f_{\theta_0}(x) + \langle \nabla_\theta f_{\theta_0}(x), \underbrace{\theta - \theta_0}_{\Delta\theta} \rangle \tag{16}$$

For convenience of analysis, we can set θ_0 s.t. $f_{\theta_0}(x) = 0 \ \forall x$. Now,

$$g_\theta(x) := \langle \underbrace{\nabla_\theta f_{\theta_0}(x)}_{\phi(x)}, \Delta\theta \rangle \tag{17}$$

Since we are in the overparameterized lazy learning regime, we have $\theta \approx \theta_0$, and $g_\theta \approx f_\theta$, which means:

$$\mathcal{L}(f_\theta, \mathcal{D}) \approx \mathcal{L}(g_\theta, \mathcal{D}) \tag{18}$$

Linear Approximation of f_θ

Applying first-order taylor expansion of f with respect to the evolution of parameters θ, we have:

$$g_\theta(x) := f_{\theta_0}(x) + \langle \nabla_\theta f_{\theta_0}(x), \underbrace{\theta - \theta_0}_{\Delta\theta} \rangle \tag{16}$$

For convenience of analysis, we can set θ_0 s.t. $f_{\theta_0}(x) = 0 \ \forall x$. Now,

$$g_\theta(x) := \langle \underbrace{\nabla_\theta f_{\theta_0}(x)}_{\phi(x)}, \Delta\theta \rangle \tag{17}$$

Since we are in the overparameterized lazy learning regime, we have $\theta \approx \theta_0$, and $g_\theta \approx f_\theta$, which means:

$$\mathcal{L}(f_\theta, \mathcal{D}) \approx \mathcal{L}(g_\theta, \mathcal{D}) \tag{18}$$

Now for $G^{(L)} : \mathbb{R}^p \to \mathcal{G}$ cost-composition $C \circ G^L : \mathbb{R}^p \to \mathbb{R}$ is convex!!

Connection to Convergence

Next, notice $\phi(x)$ **does not** depend on post training parameters.

Connection to Convergence

Next, notice $\phi(x)$ **does not** depend on post training parameters.

We can use this to define feature map Φ over the entire dataset:

$$\Phi = \left[\phi(x_i)^T\right]_n = \left[\nabla f_{\theta_0}(x_i)^T\right]_n \in \mathbb{R}^{n \times p} \tag{19}$$

Connection to Convergence

Next, notice $\phi(x)$ **does not** depend on post training parameters.

We can use this to define feature map Φ over the entire dataset:

$$\Phi = \left[\phi(x_i)^T\right]_n = \left[\nabla f_{\theta_0}(x_i)^T\right]_n \in \mathbb{R}^{n \times p} \tag{19}$$

Optimizing $\mathcal{L}(f_\theta) \approx$ optimizing $\mathcal{L}(g_\theta)$ which is convex in parameter space. So this is basically just linear regression:

$$\min_{g_\theta} \|y - g_\theta(x)\|_2^2 = \min_{\Delta\theta} \|y - \Phi \cdot \Delta\theta\|_2^2 \tag{20}$$

Comparing in output space, we can show that decay of error is exponential.

Connection to Convergence

Next, notice $\phi(x)$ **does not** depend on post training parameters.

We can use this to define feature map Φ over the entire dataset:

$$\Phi = \left[\phi(x_i)^T\right]_n = \left[\nabla f_{\theta_0}(x_i)^T\right]_n \in \mathbb{R}^{n \times p} \tag{19}$$

Optimizing $\mathcal{L}(f_\theta) \approx$ optimizing $\mathcal{L}(g_\theta)$ which is convex in parameter space. So this is basically just linear regression:

$$\min_{g_\theta} \|y - g_\theta(x)\|_2^2 = \min_{\Delta\theta} \|y - \Phi \cdot \Delta\theta\|_2^2 \tag{20}$$

Comparing in output space, we can show that decay of error is exponential.

Caveat: We're discussing gradient descent, not *stochastic* gradient descent. In practice, we see performance dropoffs when optimizing over g_θ.

Defining NTK

We can define a kernel using our feature map ϕ:

$$\kappa(x, x') = \phi(x)^T \phi(x') = \langle \nabla_\theta f_{\theta_0}(x), \nabla_\theta f_{\theta_0}(x') \rangle \tag{21}$$

This kernel is the **neural tangent kernel**.

Defining NTK

We can define a kernel using our feature map ϕ:

$$\kappa(x, x') = \phi(x)^T \phi(x') = \langle \nabla_\theta f_{\theta_0}(x), \nabla_\theta f_{\theta_0}(x') \rangle \tag{21}$$

This kernel is the **neural tangent kernel**.

The NTK effectively exists as part of gradient descent:

$$\theta_{t+1} = \theta_t - \nabla_\theta \mathcal{L}(f_{\theta_t}, \mathcal{D}) \tag{22}$$

$$\frac{\partial \theta}{\partial t} = -\eta \nabla_\theta \mathcal{L}(\theta, \mathcal{D}) = -\frac{\eta}{n} \sum_{i=1}^{n} \nabla_\theta f_\theta(x_i) \nabla_{f_\theta} \ell(f_\theta, y_i) \tag{23}$$

$$\frac{\partial f_\theta(x)}{\partial t} = \frac{\partial f_\theta(x)}{\partial \theta} \frac{\partial \theta}{\partial t} = -\frac{\eta}{n} \sum_{i=1}^{n} \underbrace{\nabla_\theta f_\theta(x) \nabla_\theta f_\theta(x_i)}_{NTK} \nabla_{f_\theta} \ell(f_\theta, y_i) \tag{24}$$

Defining NTK

We can define a kernel using our feature map ϕ:

$$\kappa(x, x') = \phi(x)^T \phi(x') = \langle \nabla_\theta f_{\theta_0}(x), \nabla_\theta f_{\theta_0}(x') \rangle \tag{21}$$

This kernel is the **neural tangent kernel**.

The NTK effectively exists as part of gradient descent:

$$\theta_{t+1} = \theta_t - \nabla_\theta \mathcal{L}(f_{\theta_t}, \mathcal{D}) \tag{22}$$

$$\frac{\partial \theta}{\partial t} = -\eta \nabla_\theta \mathcal{L}(\theta, \mathcal{D}) = -\frac{\eta}{n} \sum_{i=1}^{n} \nabla_\theta f_\theta(x_i) \nabla_{f_\theta} \ell(f_\theta, y_i) \tag{23}$$

$$\frac{\partial f_\theta(x)}{\partial t} = \frac{\partial f_\theta(x)}{\partial \theta} \frac{\partial \theta}{\partial t} = -\frac{\eta}{n} \sum_{i=1}^{n} \underbrace{\nabla_\theta f_\theta(x) \nabla_\theta f_\theta(x_i)}_{NTK} \nabla_{f_\theta} \ell(f_\theta, y_i) \tag{24}$$

The NTK depends on θ_0, is random at initialization and varies during training, but in the limit, this changes.

Limiting Behavior of NTK

When the width tends to infinity, the NTK is deterministic at intitialization and doesn't change through training:

$$k \to \infty \implies f_{\theta,k} \to \mathcal{N}(0, \Sigma_k) \; \forall k \in \{1, \ldots, n_L\} \tag{25}$$

Limiting Behavior of NTK

When the width tends to infinity, the NTK is deterministic at intitialization and doesn't change through training:

$$k \to \infty \implies f_{\theta,k} \to \mathcal{N}(0, \Sigma_k) \; \forall k \in \{1, \ldots, n_L\} \tag{25}$$

Where covariance matrices Σ_k are defined recursively as follows:

$$\Sigma_1(x, x') = \frac{1}{n_0} x^T x' + \beta^2 \tag{26}$$

$$\Sigma_{k+1}(x, x') = \mathbb{E}_{f \sim \mathcal{N}(0, \Sigma_k)}[\sigma(f(x))\sigma(f(x'))] + \beta^2 \tag{27}$$

Limiting Behavior of NTK

When the width tends to infinity, the NTK is deterministic at intitialization and doesn't change through training:

$$k \to \infty \implies f_{\theta,k} \to \mathcal{N}(0, \Sigma_k) \; \forall k \in \{1, \ldots, n_L\} \tag{25}$$

Where covariance matrices Σ_k are defined recursively as follows:

$$\Sigma_1(x, x') = \frac{1}{n_0} x^T x' + \beta^2 \tag{26}$$

$$\Sigma_{k+1}(x, x') = \mathbb{E}_{f \sim \mathcal{N}(0, \Sigma_k)}[\sigma(f(x))\sigma(f(x'))] + \beta^2 \tag{27}$$

Which we further defines the limiting NTK:

$$\kappa_1(x, x') = \Sigma_1(x, x') \tag{28}$$

$$\kappa_{k+1}(x, x') = \kappa_k(x, x')\Sigma'_{k+1}(x, x') + \Sigma_k(x, x') \tag{29}$$

$$\Sigma'_{k+1}(x, x') = \mathbb{E}_{f \sim \mathcal{N}(0, \Sigma_k)}[\sigma'(f(x))\sigma'(f(x'))] + \beta^2 \tag{30}$$

Which is independent of initialization.

Training within the NTK Regime

During training f_θ follows a descent along the negative kernel gradient:

$$\partial_t f_{\theta_t} = -\nabla_{\Phi_{(L)}} C_{|f_{\theta_t}} \tag{31}$$

The Limiting Kernel is always *positive definite*. By performing an eigendecomposition, we can decouple the gradient flow into eigenvectors, that decay at the rate of their eigenvalues.

Training within the NTK Regime

During training f_θ follows a descent along the negative kernel gradient:

$$\partial_t f_{\theta_t} = -\nabla_{\Phi_{(L)}} C_{|f_{\theta_t}} \tag{31}$$

The Limiting Kernel is always *positive definite*. By performing an eigendecomposition, we can decouple the gradient flow into eigenvectors, that decay at the rate of their eigenvalues.

Since they all decay, NTK guaranteees that infinite width neural networks converge to a global minimum when trained to minimize empirical loss.

Training within the NTK Regime

During training f_θ follows a descent along the negative kernel gradient:

$$\partial_t f_{\theta_t} = -\nabla_{\Phi_{(L)}} C_{|f_{\theta_t}} \tag{31}$$

The Limiting Kernel is always *positive definite*. By performing an eigendecomposition, we can decouple the gradient flow into eigenvectors, that decay at the rate of their eigenvalues.

Since they all decay, NTK guaranteees that infinite width neural networks converge to a global minimum when trained to minimize empirical loss.

Bonus: We have theoretical motivations for early-stopping; once the k largest eigenvectors decay beyond a set threshold, stop training.

Training within the NTK Regime

During training f_θ follows a descent along the negative kernel gradient:

$$\partial_t f_{\theta_t} = -\nabla_{\Phi_{(L)}} C_{|f_{\theta_t}} \tag{31}$$

The Limiting Kernel is always *positive definite*. By performing an eigendecomposition, we can decouple the gradient flow into eigenvectors, that decay at the rate of their eigenvalues.

Since they all decay, NTK guaranteees that infinite width neural networks converge to a global minimum when trained to minimize empirical loss.

Bonus: We have theoretical motivations for early-stopping; once the k largest eigenvectors decay beyond a set threshold, stop training.

Caveat: Computing NTK is expensive $\Omega(n^2)$ while regular GD uses $\mathcal{O}(d)$ samples.

Thank you!

Have an awesome rest of your day!

Slides: https://jinen.setpal.net/slides/ntk.pdf

References:

1. Dwaraknath, Rajat. (Nov 2019). Understanding the Neural Tangent Kernel. https://www.eigentales.com/NTK/
2. Ma, Tengyu. (Nov 2022). Stanford CS229M - Lecture 13: Neural Tangent Kernel. https://youtu.be/btphvvnad0A
3. Ma, Tengyu. (Nov 2022). Stanford CS229M - Lecture 14: Neural Tangent Kernel, Implicit regularization of gradient descent. https://youtu.be/xpT1ymwCk9w
4. Weng, Lilian. (Sep 2022). Some math behind neural tangent kernel. https://lilianweng.github.io/posts/2022-09-08-ntk/
5. Pietraho, Thomas. Math 2805: Mathematical principles of machine learning. https://web.bowdoin.edu/~tpietrah/hugo/math2805/docs/unsupervised_learning/dim_reduction/hw_operator_norms/