An Introduction to the Neural Tangent Kernel (NTK)

Stephan Rabanser

stephan@cs.toronto.edu

University of Toronto
Department of Computer Science

Vector Institute for
Artificial Intelligence

February 2, 2023

Recap: Linear Regression

- We are given a dataset $\mathcal{D} = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^n$ with $\boldsymbol{x}_i \in \mathbb{R}^D$ and $y_i \in \mathbb{R}$.
- Goal: predict a response y from features $\boldsymbol{x}$ using a linear function.
- Model the relationship using the predictive function:

$$f_{\boldsymbol{w}}(\boldsymbol{x}) = \langle \boldsymbol{w}, \boldsymbol{x} \rangle = \boldsymbol{w}^\top \boldsymbol{x}$$

- The quality of our fit is determined by a loss function:

$$\mathcal{L}_{\boldsymbol{w}} = \frac{1}{2} \sum_{i=1}^{n} (y_i - f_{\boldsymbol{w}}(\boldsymbol{x}_i))^2$$

- We want to minimize the loss: $\min_{\boldsymbol{w}} \mathcal{L}_{\boldsymbol{w}}$. We can do this using gradient descent:

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t - \eta \nabla_{\boldsymbol{w}} \mathcal{L}_{\boldsymbol{w}}$$

Recap: Kernel Methods

- Linear relations are restrictive, interesting datasets have non-linear interactions!
- Transform features into higher-dimensional space:

$$\boldsymbol{x} \in \mathbb{R}^D \to \phi(\boldsymbol{x}) \in \mathbb{R}^K, K \gg D$$

- Structure of prediction function stays the same: $f_{\boldsymbol{w}}(\boldsymbol{x}) = \boldsymbol{w}^\top \phi(\boldsymbol{x})$.
- The model is *linear* in $\boldsymbol{w}$ but *non-linear* in $\boldsymbol{x}$.
- As a result, the objective $\mathcal{L}_{\boldsymbol{w}}$ is still convex and solvable using gradient descent:

$$\mathcal{L}_{\boldsymbol{w}} = \frac{1}{2}\sum_{i=1}^{n}(y_i - f_{\boldsymbol{w}}(\boldsymbol{x}_i))^2 = \frac{1}{2}\sum_{i=1}^{n}(y_i - \boldsymbol{w}^\top \phi(\boldsymbol{x}_i))^2 \qquad \min_{\boldsymbol{w}} \mathcal{L}_{\boldsymbol{w}}$$

- This is great! We have non-linear features but still a convex objective! However:
  - The transformation function $\phi(\cdot)$ is fixed and needs to be tuned manually.
  - The computation of the feature map $\phi(\boldsymbol{x}) \in \mathbb{R}^K$ with $K \gg D$ can be expensive.

Recap: Kernel Trick

- Many ML algorithms can be reformulated to feature space inner products:

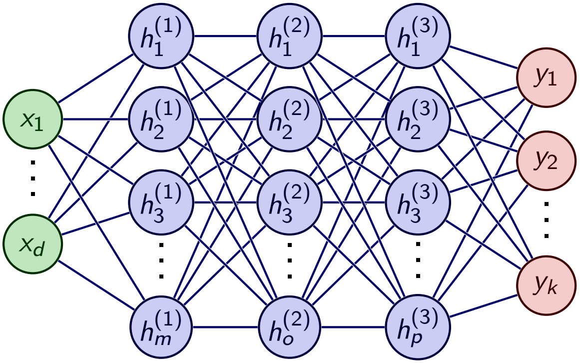
$$\kappa(\boldsymbol{x}, \boldsymbol{x}') = \langle \phi(\boldsymbol{x}), \phi(\boldsymbol{x}') \rangle$$

- This inner product is called a *kernel function* $\kappa(\cdot, \cdot)$ and can be thought of as a similarity measure of two input vectors $\boldsymbol{x}$ and $\boldsymbol{x}'$.
- Example kernels:

$$\kappa_{\text{Pol}}(\boldsymbol{x}, \boldsymbol{x}') = (\boldsymbol{x}^\top \mathbf{x}' + c)^d \qquad \kappa_{\text{SE}}(\boldsymbol{x}, \boldsymbol{x}') = \sigma^2 \exp\left(-\frac{(\boldsymbol{x} - \boldsymbol{x}')^2}{2\ell^2}\right)$$

- Collect all similarities in a postive semi-definite kernel matrix: $\boldsymbol{K} \in \mathbb{R}^{n \times n} \succeq 0$.
- Algorithms can be "kernelized": only rely on kernels instead of expl. feature maps.
- Are such kernels flexible enough? What about the computational concerns?

Recap: Neural Networks



- We don't want to define non-linear transformations ourselves: we want to *learn* them!
- Discover multiple (hierarchical) feature spaces:

$$\mathbb{R}^d \to \mathbb{R}^m \to \mathbb{R}^o \to \mathbb{R}^p \to \cdots \to \mathbb{R}^k$$

- The objective $\mathcal{L}_{\boldsymbol{w}}$ is not convex, yet we still use gradient descent:

$$\mathcal{L}_{\boldsymbol{w}} = \frac{1}{2}\sum_{i=1}^{n}(y_i - f_{\boldsymbol{w}}(\boldsymbol{x}_i))^2 = \frac{1}{2}\sum_{i=1}^{n}\left(y_i - \sigma(\boldsymbol{W}_l^\top \cdots \sigma(\boldsymbol{W}_2^\top \sigma(\boldsymbol{W}_1^\top \boldsymbol{x}_i)))\right)^2 \qquad \boldsymbol{w} = (\boldsymbol{W}_1, \ldots, \boldsymbol{W}_l)$$

Understanding the Performance of Neural Nets: Central Questions

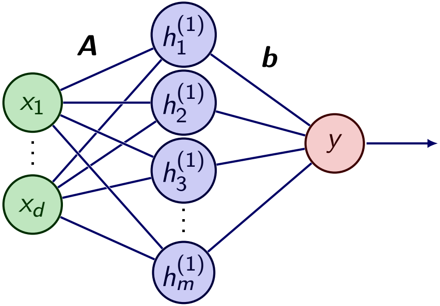
Neural networks empirically perform well, but their convergence and generalization properties are hard to analyze.

Deep learning is poorly understood. Kernel methods on the other hand are based on solid mathematical theory. *Can we distill a neural network into a kernel?*

It is a known result that at initialization time, a wide neural network is a Gaussian process. *Can we also describe the training process using a kernel?*

Deriving a Kernel from a Neural Network

- Assume a simple neural network with a single hidden layer and params $\boldsymbol{w} = (\boldsymbol{A}, \boldsymbol{b})$:



$y = f_{\boldsymbol{w}}(\boldsymbol{x}) = \frac{1}{\sqrt{m}} \sum_{j=1}^{m} b_j \sigma(\boldsymbol{a}_j^\top \boldsymbol{x})$

- As discussed, the objective $\mathcal{L}_{\boldsymbol{w}}$ is no longer convex:

$$\mathcal{L}_{\boldsymbol{w}} = \frac{1}{2} \sum_{i=1}^{n} (y_i - f_{\boldsymbol{w}}(\boldsymbol{x}_i))^2 = \frac{1}{2} \sum_{i=1}^{n} \left(y_i - \frac{1}{\sqrt{m}} \sum_{j=1}^{m} b_j \sigma(\boldsymbol{a}_j^\top \boldsymbol{x}_i) \right)^2$$

Deriving a Kernel from a Neural Network (cont'd)

$$\mathcal{L}_{\boldsymbol{w}} = \frac{1}{2}\sum_{i=1}^{n}(y_i - f_{\boldsymbol{w}}(\boldsymbol{x}_i))^2 = \frac{1}{2}\sum_{i=1}^{n}\left(y_i - \frac{1}{\sqrt{m}}\sum_{j=1}^{m} b_j\sigma(\boldsymbol{a}_j^\top \boldsymbol{x}_i)\right)^2$$

- We can still minimize the objective $\mathcal{L}_{\boldsymbol{w}}$ using *full-batch* gradient descent:

$$\begin{aligned}\boldsymbol{w}_{t+1} &= \boldsymbol{w}_t - \eta\nabla_{\boldsymbol{w}}\mathcal{L}_{\boldsymbol{w}_t} \\ &= \boldsymbol{w}_t - \eta\sum_{i=1}^{n}(y_i - f_{\boldsymbol{w}_t}(\boldsymbol{x}_i)) \underbrace{\nabla_{\boldsymbol{w}} f_{\boldsymbol{w}_t}(\boldsymbol{x}_i)}_{\substack{\text{variable during training} \\ \text{changes depending on } \boldsymbol{w}_t}}\end{aligned}$$

Under what circumstances does $\nabla_{\boldsymbol{w}} f_{\boldsymbol{w}_t}(\boldsymbol{x}_i)$ not change much during training?

Lazy Training

- Assume we initialize the weights randomly using a standard Gaussian: $\mathcal{N}(0,1)$.
- We can observe the trajectory of the weights during training:

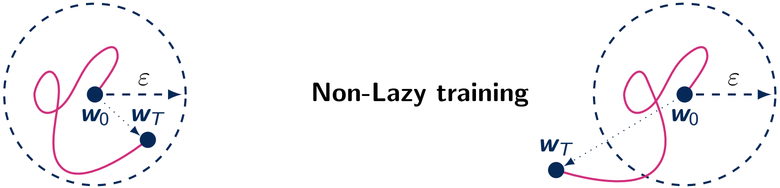
$$\boldsymbol{w}_0 \longrightarrow \boldsymbol{w}_1 \longrightarrow \boldsymbol{w}_2 \longrightarrow \dots \longrightarrow \boldsymbol{w}_T$$

Empirical Observation

When m is large ($m \to \infty$), parameters show stable evolution patterns ($\varepsilon \to 0$).

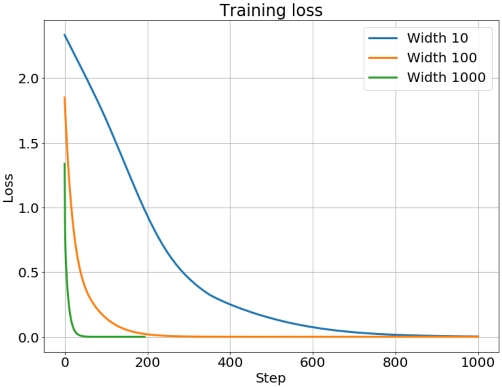
Lazy training

Non-Lazy training

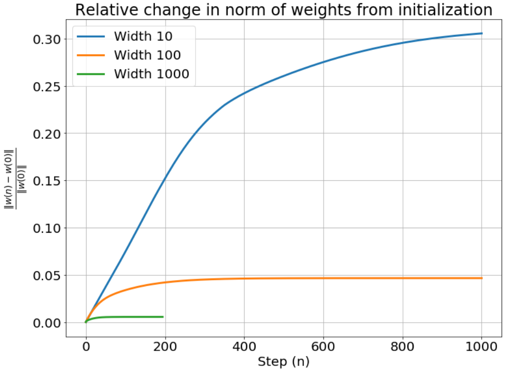


UNIVERSITY OF TORONTO VECTOR INSTITUTE

Depiction of Lazy Training for Wide Neural Nets



(a) Loss evolution



(b) Weight distance from origin

`https://rajatvd.github.io/NTK/`

Neural Tangent Kernel: Approximating a Lazy Trajectory

- If lazy training holds, then a first-order Taylor approximation of the function around its initialization $\boldsymbol{w}_0$ might be helpful:

$$f_{\boldsymbol{w}}(\boldsymbol{x}) \approx f_{\boldsymbol{w}_0}(\boldsymbol{x}) + \underbrace{\nabla_{\boldsymbol{w}} f_{\boldsymbol{w}_0}(\boldsymbol{x})^\top}_{\phi(\boldsymbol{x})^\top}(\boldsymbol{w} - \boldsymbol{w}_0) \underbrace{+ \ \ldots}_{\substack{\text{higher order} \\ \text{Taylor terms}}}$$

$$\approx \underbrace{c + \phi(\boldsymbol{x})^\top(\boldsymbol{w} - \boldsymbol{w}_0)}_{\text{model is an affine func in } \boldsymbol{w}}$$

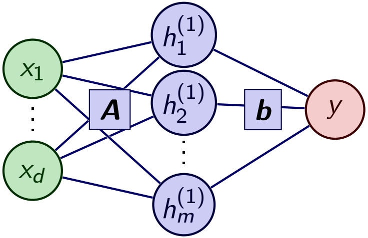
Neural Tangent Kernel (NTK)

For the standard kernel definition $\kappa(\boldsymbol{x}, \boldsymbol{x}') = \langle \phi(\boldsymbol{x}), \phi(\boldsymbol{x}') \rangle$, use the gradient of the neural network's output evaluated at $\boldsymbol{w}_0$ as the kernel function:

$$\phi(\boldsymbol{x}) = \nabla_{\boldsymbol{w}} f_{\boldsymbol{w}_0}(\boldsymbol{x})$$

Computing the NTK for Our Toy Example

- Recall our simple neural network from before:



$$y = f_{\boldsymbol{w}}(\boldsymbol{x}) = \frac{1}{\sqrt{m}} \sum_{j=1}^{m} b_j \sigma(\boldsymbol{a}_j^\top \boldsymbol{x})$$

$$\begin{cases} \nabla_{\boldsymbol{a}_j} f_{\boldsymbol{w}}(\boldsymbol{x}) = \frac{1}{\sqrt{m}} b_j \sigma'(\boldsymbol{a}_j^\top \boldsymbol{x})\boldsymbol{x} \\ \nabla_{b_j} f_{\boldsymbol{w}}(\boldsymbol{x}) = \frac{1}{\sqrt{m}} \sigma(\boldsymbol{a}_j^\top \boldsymbol{x}) \end{cases}$$

- Computing the Neural Tangent Kernel for this network:

$$\begin{aligned} \kappa_{\text{NTK}}(\boldsymbol{x}, \boldsymbol{x}') &= \kappa_{\text{NTK}}^{(\boldsymbol{A})}(\boldsymbol{x}, \boldsymbol{x}') + \kappa_{\text{NTK}}^{(\boldsymbol{b})}(\boldsymbol{x}, \boldsymbol{x}') \\ &= \frac{1}{m} \sum_{i=1}^{m} b_i^2 \sigma'(\boldsymbol{a}_i^\top \boldsymbol{x}) \sigma'(\boldsymbol{a}_j^\top \boldsymbol{x}') \boldsymbol{x}\boldsymbol{x}' + \frac{1}{m} \sum_{j=1}^{m} \sigma(\boldsymbol{a}_j^\top \boldsymbol{x}) \sigma(\boldsymbol{a}_j^\top \boldsymbol{x}') \\ &\overset{m \to \infty}{=} \mathbb{E}[\boldsymbol{b}^2 \sigma'(\boldsymbol{A}^\top x) \sigma'(\boldsymbol{A}^\top \boldsymbol{x}') \boldsymbol{x}\boldsymbol{x}'] + \mathbb{E}[\sigma(\boldsymbol{A}^\top x) \sigma(\boldsymbol{A}^\top \boldsymbol{x}')] \end{aligned}$$

Analyzing Gradient Descent

- This kernel can directly be used in a kernel machine!
- We can analyze further properties of neural network training using flow dynamics.
- Example: parameter gradient flow dynamics ($\eta \to 0$):

$$\begin{aligned} \boldsymbol{w}_{t+1} &= \boldsymbol{w}_t - \eta \nabla_{\boldsymbol{w}} \mathcal{L}_{\boldsymbol{w}_t} \\ \frac{\boldsymbol{w}_{t+1} - \boldsymbol{w}_t}{\eta} &= -\nabla_{\boldsymbol{w}} \mathcal{L}_{\boldsymbol{w}_t} \\ \frac{d\boldsymbol{w}_t}{dt} &\overset{\eta \to 0}{=} -\nabla_{\boldsymbol{w}} \mathcal{L}_{\boldsymbol{w}_t} \end{aligned}$$

Can we use differential equations to analyze the evolution of various properties of neural networks (weights, predictions, losses, etc. over time)?

Dynamics of Parameters and Predictions

Parameter Dynamics

How do the parameters change over the course of training?

Assume $\mathcal{L}_{\boldsymbol{w}_t} = \frac{1}{2}||\boldsymbol{f}_{\boldsymbol{w}_t} - \boldsymbol{y}||_2^2$. Then:

$$\nabla_{\boldsymbol{w}} \mathcal{L}_{\boldsymbol{w}_t} = \nabla_{\boldsymbol{w}} \boldsymbol{f}_{\boldsymbol{w}_t} (\boldsymbol{f}_{\boldsymbol{w}_t} - \boldsymbol{y})$$

$$\begin{aligned} \frac{d\boldsymbol{w}_t}{dt} &= -\nabla_{\boldsymbol{w}} \mathcal{L}_{\boldsymbol{w}_t} \\ &= -\nabla_{\boldsymbol{w}} \boldsymbol{f}_{\boldsymbol{w}_t} (\boldsymbol{f}_{\boldsymbol{w}_t} - \boldsymbol{y}) \end{aligned}$$

Predictions Dynamics

How do the predictions change over the course of training?

Approximate w/ NTK matrix $\boldsymbol{K}_{\text{NTK}}$:

$$\begin{aligned} \frac{d\boldsymbol{f}_{\boldsymbol{w}_t}}{dt} &= -\nabla_{\boldsymbol{w}} \boldsymbol{f}_{\boldsymbol{w}_t}^\top \frac{d\boldsymbol{w}_t}{dt} \\ &= -\underbrace{\nabla_{\boldsymbol{w}} \boldsymbol{f}_{\boldsymbol{w}_t}^\top \nabla_{\boldsymbol{w}} \boldsymbol{f}_{\boldsymbol{w}_t}}_{\text{NTK evaluated at } \boldsymbol{w}_t} (\boldsymbol{f}_{\boldsymbol{w}_t} - \boldsymbol{y}) \\ &\approx -\boldsymbol{K}_{\text{NTK}} (\boldsymbol{f}_{\boldsymbol{w}_t} - \boldsymbol{y}) \end{aligned}$$

ODE Solution for Residual Loss

- Under a simple residual loss, we can model the dynamics w/ linear ODE:

$$\boldsymbol{u} = \boldsymbol{f}_{\boldsymbol{w}_t} - \boldsymbol{y} \quad \Longrightarrow \quad \frac{d\boldsymbol{u}}{dt} \approx -\boldsymbol{K}_{\text{NTK}}\boldsymbol{u} \quad \Longrightarrow \quad \boldsymbol{u}_t = \boldsymbol{u}_0 \exp(-\boldsymbol{K}_{\text{NTK}}\, t)$$

- As $t \to \infty$, $\boldsymbol{u} \to 0$ and $f_{\boldsymbol{w}_t} \to \boldsymbol{y}$.
- In over-parameterized networks: $\boldsymbol{K}_{\text{NTK}} \succ 0$, i.e. smallest eigenvalue larger than 0!
- We can factorize the kernel matrix $\boldsymbol{K}_{\text{NTK}} = \sum_{i=1}^{k} \lambda_i \boldsymbol{v}_i \boldsymbol{v}_i^\top$ with $0 < \lambda_1 < \ldots < \lambda_k$.
- Substituting this factorization back into the ODE:

$$\boldsymbol{u}_t = \boldsymbol{u}_0 \prod_{i=1}^{k} \exp(-\lambda_i \boldsymbol{v}_i \boldsymbol{v}_i^\top t)$$

- λ_1 governs the rate of convergence.

Summary & Open Questions

Summary of Neural Tangent Kernel

- Parameters hardly move from their initialization for $m \to \infty$.
- NTK is defined as the gradient of the NN evaluated at init: $\nabla_{\boldsymbol{w}} \boldsymbol{f}_{\boldsymbol{w}_0}^{\top} \nabla_{\boldsymbol{w}} \boldsymbol{f}_{\boldsymbol{w}_0}$.
- NTK is deterministic at initialization and constant during training.
- Can be used in ODEs to study evolution of various quantities.

Open Questions

- Results from the NTK limit are not SOTA. Where does the gap come from?
- Results only hold for full-batch GD. What is the role of SGD?
- NTK results talk about convergence? What about performance?
- How can we use these training dynamics insights for trust?

Thanks! :)

References

- https://rajatvd.github.io/NTK/
- https://lilianweng.github.io/posts/2022-09-08-ntk/
- https://www.youtube.com/watch?v=DObobAnELkU

Sanjeev Arora, Simon S Du, Wei Hu, Zhiyuan Li, Russ R Salakhutdinov, and Ruosong Wang, *On exact computation with an infinitely wide neural net*, NeurIPS **32** (2019).

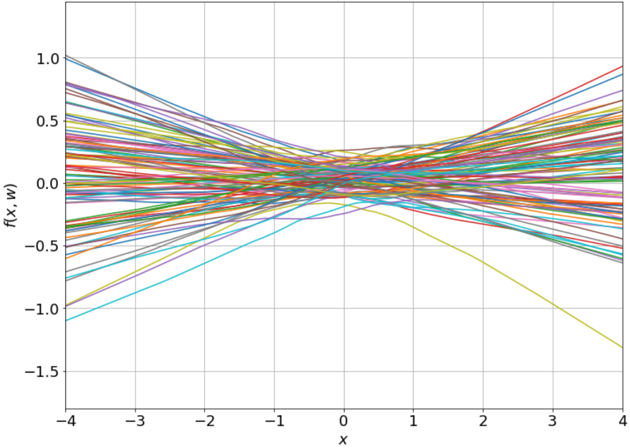
Lenaic Chizat and Francis Bach, *On the global convergence of gradient descent for over-parameterized models using optimal transport*, NeurIPS **31** (2018).

Lenaic Chizat, Edouard Oyallon, and Francis Bach, *On lazy training in differentiable programming*, NeurIPS **32** (2019).

Arthur Jacot, Franck Gabriel, and Clément Hongler, *Neural tangent kernel: Convergence and generalization in neural networks*, NeurIPS **31** (2018).

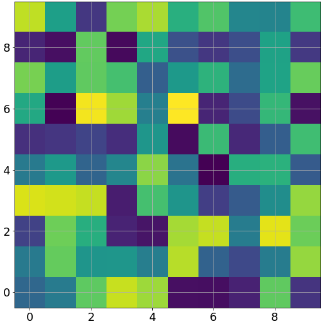
Backup

Over-Parameterization At Initialization Time → GP

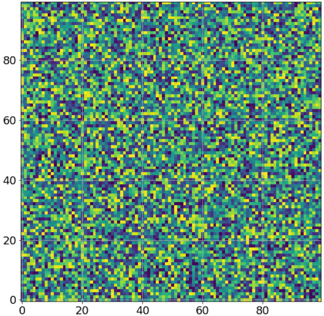


https://rajatvd.github.io/NTK/

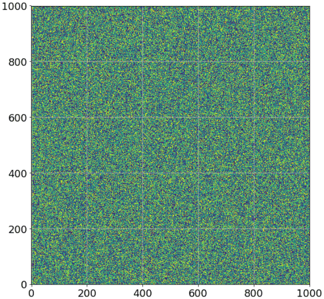
Depiction of Lazy Training for Wide Neural Nets



(a) 10 layer



(b) 100 layer



(c) 1000 layer

https://rajatvd.github.io/NTK/

UNIVERSITY OF TORONTO VECTOR INSTITUTE