Neural Tangent Kernel: Convergence and Generalization in Neural Networks

Arthur Jacot
École Polytechnique Fédérale de Lausanne
arthur.jacot@netopera.net

Franck Gabriel
Imperial College London and École Polytechnique Fédérale de Lausanne
franckrgabriel@gmail.com

Clément Hongler
École Polytechnique Fédérale de Lausanne
clement.hongler@gmail.com

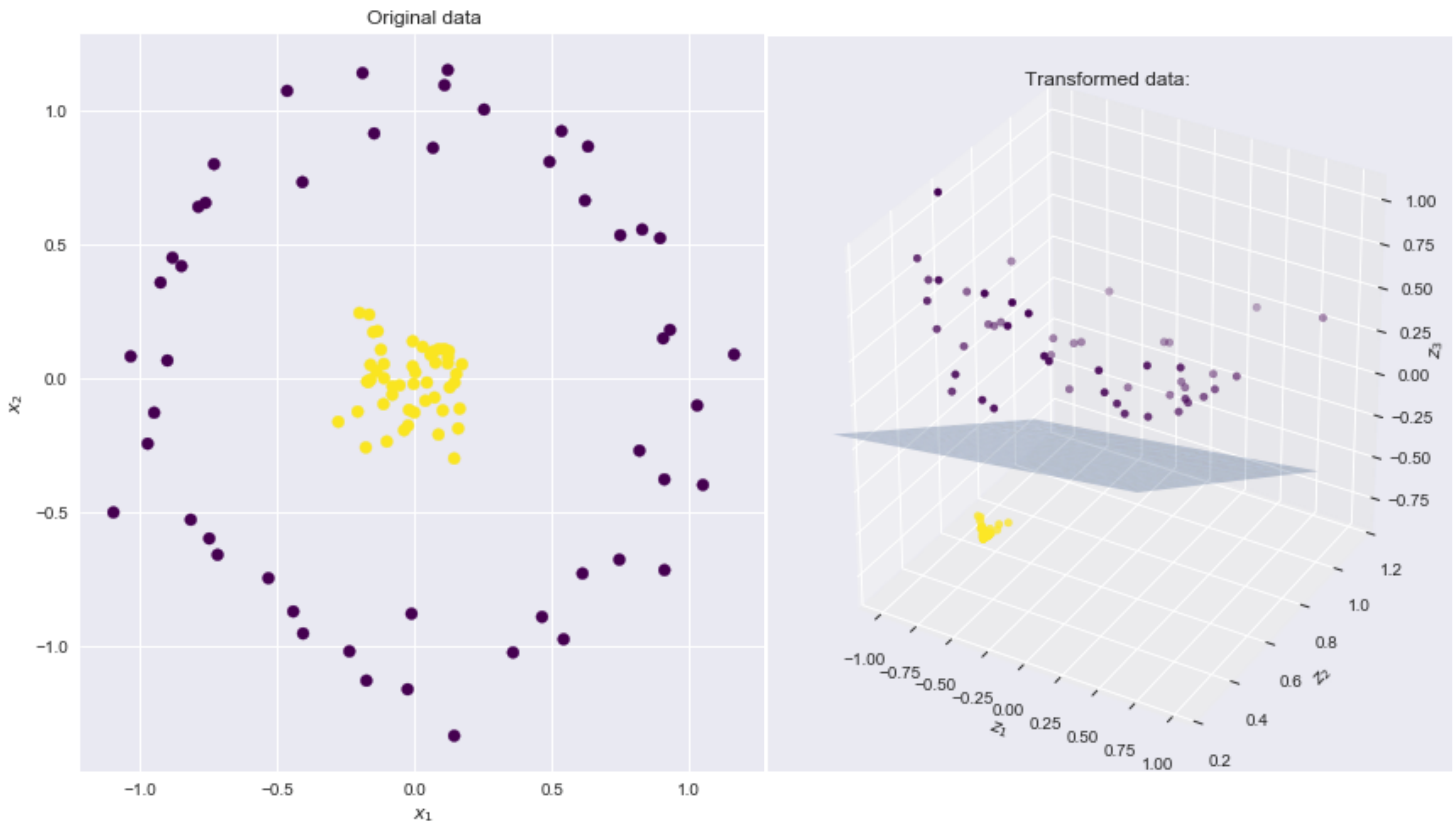
Slides by Nolan Dey

https://arxiv.org/abs/1806.07572

What is a kernel?

- A kernel K(x,x’) is a function that computes the dot product of two vectors x and x’ in some feature space

- You can think of K(x,x’) as the distance between x and x’ in some space



https://stats.stackexchange.com/questions/152897/how-to-intuitively-explain-what-a-kernel-is
https://xavierbourretsicotte.github.io/Kernel_feature_map.html

Double Descent Phenomenon

https://openai.com/blog/deep-double-descent/

Setup

- We have some neural network: $f(x, w)$
    - Input data: $x \in \mathbb{R}^{n \times d}$
    - Network parameters: $w \in \mathbb{R}^p$
- Loss: $L(f(x, w), y) = \frac{1}{2}(f(x, w) - y)^2$
- Optimize using full-batch gradient descent

What is the Neural Tangent Kernel (NTK)?

- Infinitely wide neural networks can fit any function
- Infinite width networks trained to convergence can be described by the NTK
- NTK describes training dynamics: $\dfrac{df(x, w)}{dt} = - NTK(w_0)(f(x, w) - y)$

https://arxiv.org/abs/1806.07572

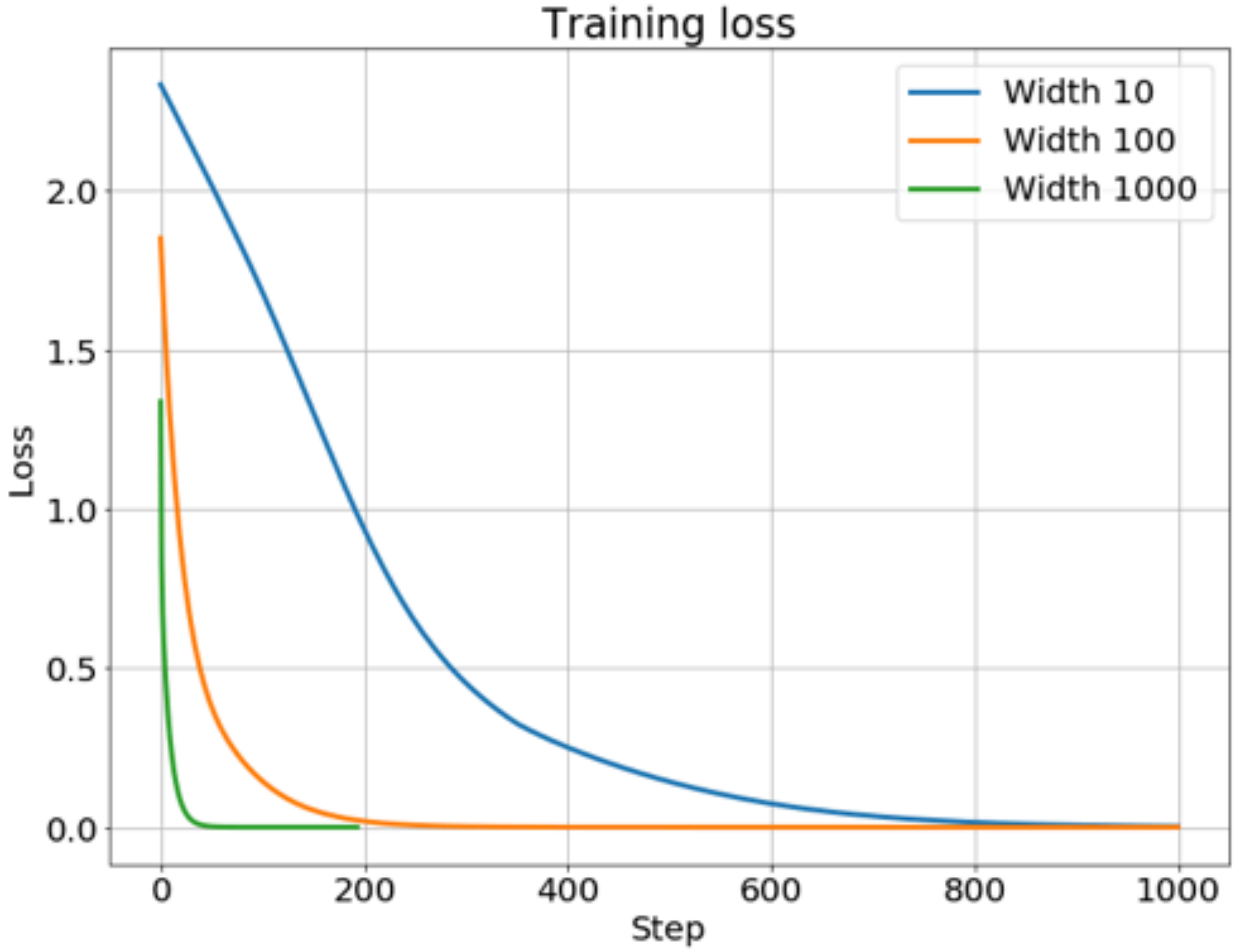
Taylor expansion of neural network

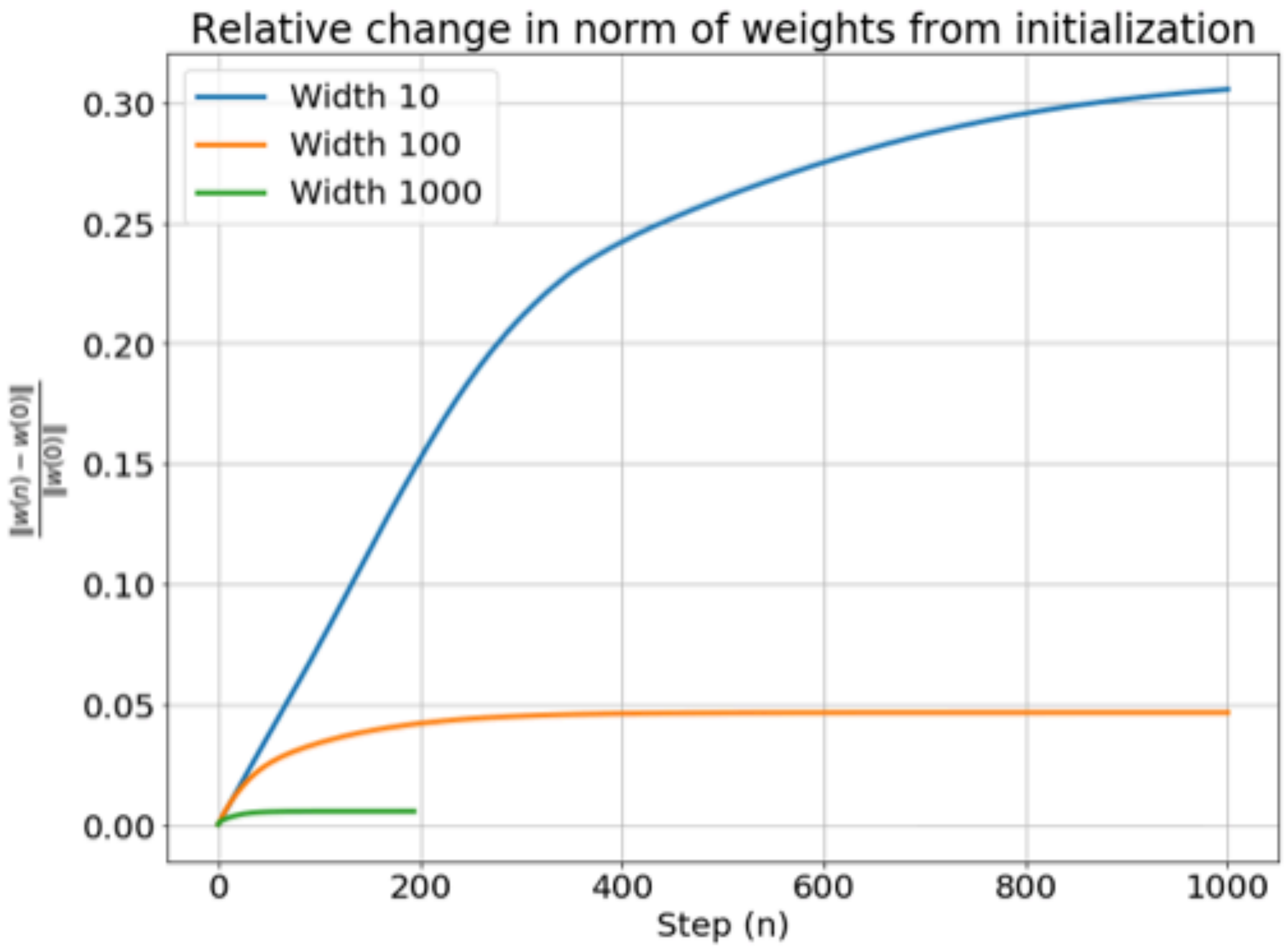
- Taylor expansion of neural network with respect to initialization weights w_0
    - $f(x, w) \approx f(x, w_0) + \nabla_w f(x, w_0)^T (w - w_0)$
- This Taylor expansion is only accurate when weights remain close to initialization

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

When is Taylor expansion accurate?

- Taylor expansion is only accurate when the weights don't change much during training
    - Weights don't change much when network is sufficiently wide
    - Lazy training = weights don't need to change





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

Gradient Flow

- Gradient descent: $w_{t+1} = w_t - \eta \nabla_w L(w_t)$

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

Gradient Flow

- Gradient descent: $w_{t+1} = w_t - \eta \nabla_w L(w_t)$
- Rewrite as: $\dfrac{w_{t+1} - w_t}{\eta} = - \nabla_w L(w_t)$

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

Gradient Flow

- Gradient descent: $w_{t+1} = w_t - \eta \nabla_w L(w_t)$
- Rewrite as: $\dfrac{w_{t+1} - w_t}{\eta} = -\nabla_w L(w_t)$
  - Resembles finite difference^
- Take infinitesimally small learning rate η

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

Gradient Flow

- Gradient descent: $w_{t+1} = w_t - \eta \nabla_w L(w_t)$
- Rewrite as: $\dfrac{w_{t+1} - w_t}{\eta} = -\nabla_w L(w_t)$
    - Resembles finite difference^
- Take infinitesimally small learning rate η
- Gradient flow! —> $\dfrac{dw(t)}{dt} = -\nabla_w L(w(t))$

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

Neural Tangent Kernel

- $\dfrac{dw(t)}{dt} = -\nabla_w L(w(t)) = -\nabla_w [\frac{1}{2}(f(x,w) - y)^2] = -\nabla_w f(x,w)(f(x,w) - y)$

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

Neural Tangent Kernel

- $\frac{dw(t)}{dt} = -\nabla_w L(w(t)) = -\nabla_w[\frac{1}{2}(f(x,w) - y)^2] = -\nabla_w f(x,w)(f(x,w) - y)$
- $\frac{df(x,w(t))}{dt} = \frac{df(x,w(t))}{dw}\frac{dw(t)}{dt} = \nabla_w f(x,w)\frac{dw(t)}{dt} = -\nabla_w f(x,w)^T \nabla_w f(x,w)(f(x,w) - y)$

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

Neural Tangent Kernel

- $\frac{dw(t)}{dt} = -\nabla_w L(w(t)) = -\nabla_w [\frac{1}{2}(f(x, w) - y)^2] = -\nabla_w f(x, w)(f(x, w) - y)$
- $\frac{df(x, w(t))}{dt} = \frac{df(x, w(t))}{dw}\frac{dw(t)}{dt} = \nabla_w f(x, w)\frac{dw(t)}{dt} = -\nabla_w f(x, w)^T \nabla_w f(x, w)(f(x, w) - y)$
- $NTK_t(x, x') = \nabla_w f(x, w)^T \nabla_w f(x, w)$

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

Neural Tangent Kernel

- $\frac{dw(t)}{dt} = -\nabla_w L(w(t)) = -\nabla_w [\frac{1}{2}(f(x,w) - y)^2] = -\nabla_w f(x,w)(f(x,w) - y)$
- $\frac{df(x,w(t))}{dt} = \frac{df(x,w(t))}{dw}\frac{dw(t)}{dt} = \nabla_w f(x,w)\frac{dw(t)}{dt} = -\nabla_w f(x,w)^T \nabla_w f(x,w)(f(x,w) - y)$
- $NTK_t(x,x') = \nabla_w f(x,w)^T \nabla_w f(x,w)$
- In the infinite width limit: $NTK_t(x,x') = NTK_0(x,x')$

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

Neural Tangent Kernel

- $\frac{dw(t)}{dt} = -\nabla_w L(w(t)) = -\nabla_w[\frac{1}{2}(f(x,w) - y)^2] = -\nabla_w f(x,w)(f(x,w) - y)$
- $\frac{df(x,w(t))}{dt} = \frac{df(x,w(t))}{dw}\frac{dw(t)}{dt} = \nabla_w f(x,w)\frac{dw(t)}{dt} = -\nabla_w f(x,w)^T \nabla_w f(x,w)(f(x,w) - y)$
- $NTK_t(x,x') = \nabla_w f(x,w)^T \nabla_w f(x,w)$
- In the infinite width limit: $NTK_t(x,x') = NTK_0(x,x')$
- $\frac{df(x,w)}{dt} = -NTK_0(x,x')(f(x,w) - y)$

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

Neural Tangent Kernel ODE

- ODE: $\dfrac{df(x,w)}{dt} = -NTK(w_0)(f(x,w) - y)$
- Substitute $u = f(x,w) - y$
- $\dfrac{du}{dt} = -NTK(w_0)u$
- Solution: $u(t) = u(0)e^{-NTK(w_0)t}$

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

On Exact Computation with an Infinitely Wide Neural Net

Recall from [Lee et al., 2018] that in the infinite width limit, the pre-activations $\boldsymbol{f}^{(h)}(\boldsymbol{x})$ at every hidden layer $h \in [L]$ has all its coordinates tending to i.i.d. centered Gaussian processes of covariance $\Sigma^{(h-1)} : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ defined recursively as: for $h \in [L]$,

$$\begin{aligned}
\Sigma^{(0)}(\boldsymbol{x}, \boldsymbol{x}') &= \boldsymbol{x}^\top \boldsymbol{x}', \\
\boldsymbol{\Lambda}^{(h)}(\boldsymbol{x}, \boldsymbol{x}') &= \begin{pmatrix} \Sigma^{(h-1)}(\boldsymbol{x}, \boldsymbol{x}) & \Sigma^{(h-1)}(\boldsymbol{x}, \boldsymbol{x}') \\ \Sigma^{(h-1)}(\boldsymbol{x}', \boldsymbol{x}) & \Sigma^{(h-1)}(\boldsymbol{x}', \boldsymbol{x}') \end{pmatrix} \in \mathbb{R}^{2\times 2}, \\
\Sigma^{(h)}(\boldsymbol{x}, \boldsymbol{x}') &= c_\sigma \mathop{\mathbb{E}}_{(u,v)\sim\mathcal{N}\left(\boldsymbol{0}, \boldsymbol{\Lambda}^{(h)}\right)} \left[\sigma(u)\,\sigma(v)\right].
\end{aligned} \tag{7}$$

To give the formula of NTK, we also need to define a derivative covariance:

$$\dot{\Sigma}^{(h)}(\boldsymbol{x}, \boldsymbol{x}') = c_\sigma \mathop{\mathbb{E}}_{(u,v)\sim\mathcal{N}\left(\boldsymbol{0}, \boldsymbol{\Lambda}^{(h)}\right)} \left[\dot{\sigma}(u)\dot{\sigma}(v)\right]. \tag{8}$$

The final NTK expression for the fully-connected neural network is

$$\Theta^{(L)}(\boldsymbol{x}, \boldsymbol{x}') = \sum_{h=1}^{L+1} \left(\Sigma^{(h-1)}(\boldsymbol{x}, \boldsymbol{x}') \cdot \prod_{h'=h}^{L+1} \dot{\Sigma}^{(h')}(\boldsymbol{x}, \boldsymbol{x}') \right), \tag{9}$$

https://arxiv.org/abs/1904.11955

What does this mean?

- NTK is a useful tool for studying the dynamics of infinitely wide neural networks
- Successful nets in practice DO NOT OPERATE IN THE NTK REGIME
  - Finite nets still outperform their exactly computed infinite width counterparts
  - SGD vs full-batch gradient descent
  - NTK kernel matrix has $O(n^2)$ memory complexity

Further reading

- Convolutional NTK (https://arxiv.org/abs/1904.11955)
- Tensor Programs II: Neural Tangent Kernel for Any Architecture (https://arxiv.org/abs/2006.14548)
- Google neural_tangents library (https://arxiv.org/abs/1912.02803)
- Harnessing the Power of Infinitely Wide Deep Nets on Small-data Tasks (https://arxiv.org/abs/1910.01663)
    - SOTA on UCI datasets (structured data) (Beats random forest)

Thanks for listening!