The Neural Tanget Kernel

Reporter : Xun-Jian LI

Outline

Introduction

1.1 The gradient flow

- Gradient flow in **linear models** with least squares losses (gradient descent with an infinitesimal step size)
- **Evolution** of the parameters, loss, and predictions
- **Trainability** and **training converges** of a system

1.2 The neural tangent kernel

- In the **infinite width limit**, a neural network behaves as if it is **linear**
- Its training dynamics can be captured by the **neural tangent kernel** (**NTK**)
- The **NTK** can be computed in **closed–form** for infinitely wide networks

1.3 Applications

- The neural tangent kernel provides insights into the **trainability and convergence of neural networks**

- The **NTK** can be regarded as a **non-linear transformation of the input data**, which allows us to reinterpret inference in neural networks as a kernel regression problem and **make predictions in closed–form** without ever explicitly training the network.

Infinite width neural networks

2.1 Infinite width neural networks

What happens when increase the width of neural network?

Define the network with both the input x and the output y are scalar:

$$\begin{aligned}
\mathbf{h}_1 &= \mathbf{ReLU}\Big[\boldsymbol{\beta}_0 + \boldsymbol{\omega}_0 x\Big] \\
\mathbf{h}_2 &= \frac{1}{\sqrt{D}}\Big(\mathbf{ReLU}\Big[\boldsymbol{\beta}_1 + \boldsymbol{\Omega}_1 \mathbf{h}_1\Big]\Big) \\
y &= \frac{1}{\sqrt{D}}\Big(\beta_2 + \boldsymbol{\Omega}_2 \mathbf{h}_2\Big),
\end{aligned}$$

- $\mathbf{h}_1, \mathbf{h}_2 \in \mathbb{R}^D$: the D hidden units in the first and second hidden layers
- D: the D hidden units second hidden layers
- $\mathbf{ReLU}(\cdot)$: the elementwise application of the standard ReLU function.
- Weights: $\boldsymbol{\omega}_0 \in \mathbb{R}^{D\times 1}$, $\boldsymbol{\Omega}_1 \in \mathbb{R}^{D\times D}$, $\boldsymbol{\Omega}_2 \in \mathbb{R}^{1\times D}$
- Bias: $\boldsymbol{\beta}_0 \in \mathbb{R}^D$, $\boldsymbol{\beta}_1 \in \mathbb{R}^D$, $\beta_2 \in \mathbb{R}$

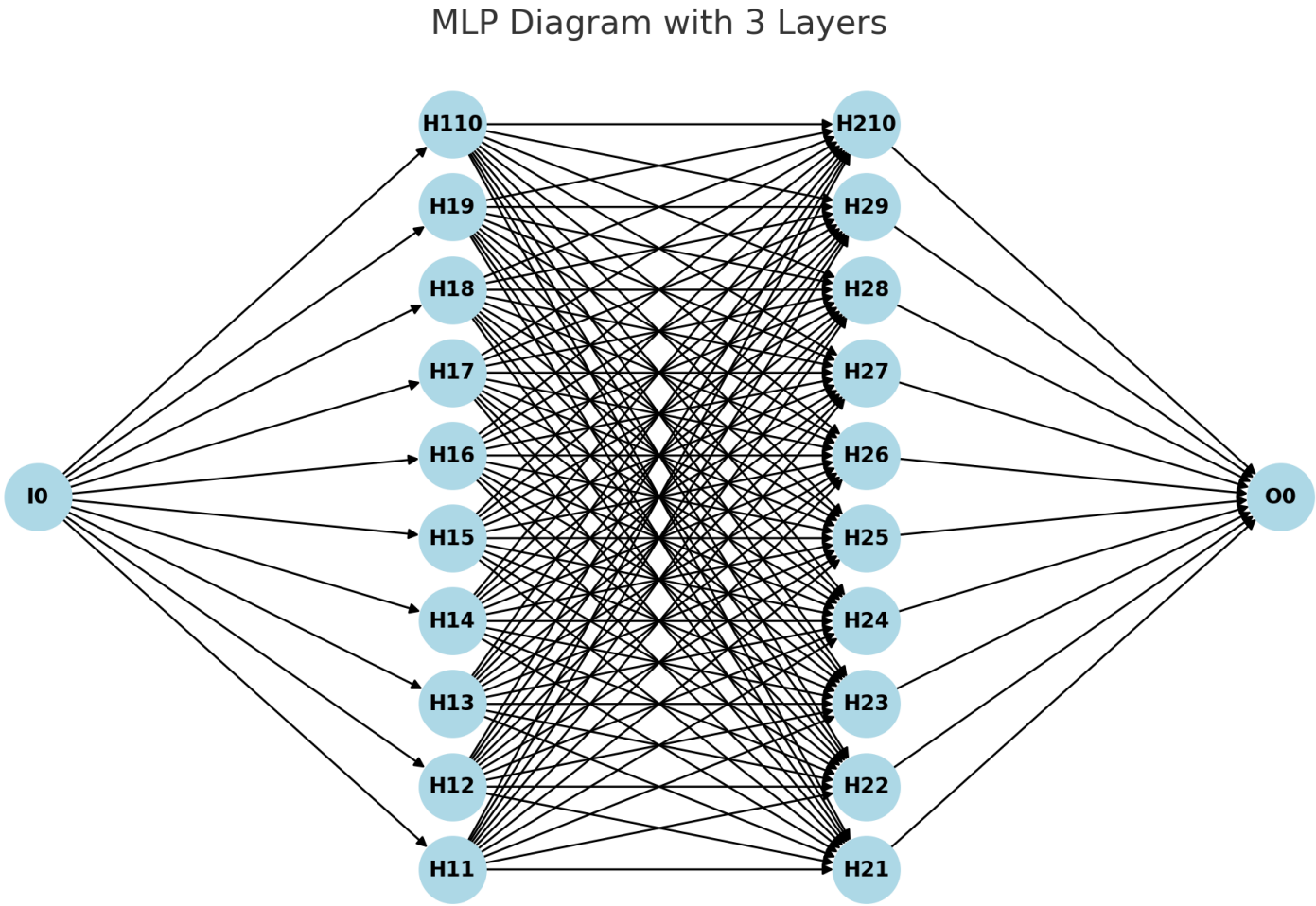


Figure 2. Neural networks (ploted by ChatGPT)

2.2 Initialization of ANN parameters

Initializing ANN parameters by drawing from a normal distribution (called Kaiming initialization, He *et al.* 2015) is crucial for several reasons:

- **Symmetry Breaking**: Ensures that different neurons learn different features by preventing them from starting with identical weights.
- **Effective Training**: Helps in achieving a balanced variance in activations, which promotes stable and efficient gradient propagation during training.
- **Theoretical Underpinning**: Underpins theoretical results such as the infinite-width neural networks converging to **Gaussian processes**, aiding in understanding and optimizing learning dynamics (Lee *et al.* 2019).

This initialization strategy lays the foundation for effective learning and convergence in ANNs.

At initialization, artificial neural networks (ANNs) are equivalent to Gaussian processes in the infinite-width limit, thus connecting them to kernel methods (Jacot *et al.* 2018).

- A Gaussian process (GP) is a non-parametric Bayesian method used for regression and classification tasks. It can be viewed as a distribution over functions, where any finite set of functions has a joint Gaussian distribution. A GP is defined by a **mean function** and a **kernel function (covariance function)**, which describes the similarity between data points.

- kernel method: Kernel methods are machine learning algorithms based on kernel functions, such as Support Vector Machines (SVM) and kernel ridge regression. Kernel functions **map input data to a high-dimensional feature space**, enabling linear methods to handle nonlinear problems in that space.

- Data on $I = 10$ data pairs (x_i, y_i) where both inputs x_i and targets y_i are drawn from a **standard normal distribution**
- Initialize the **biases** to zero and the **weights** from a **standard normal distribution**
- **Stochastic gradient descent with momentum** to train networks with different widths $D = 10,\ 50,\ 1000$

The changes of **individual parameters** is less as the growth of the width?

This is expected since the responsibility for describing the finite dataset becomes distributed between more and more parameters, so the change in any given parameter is smaller.

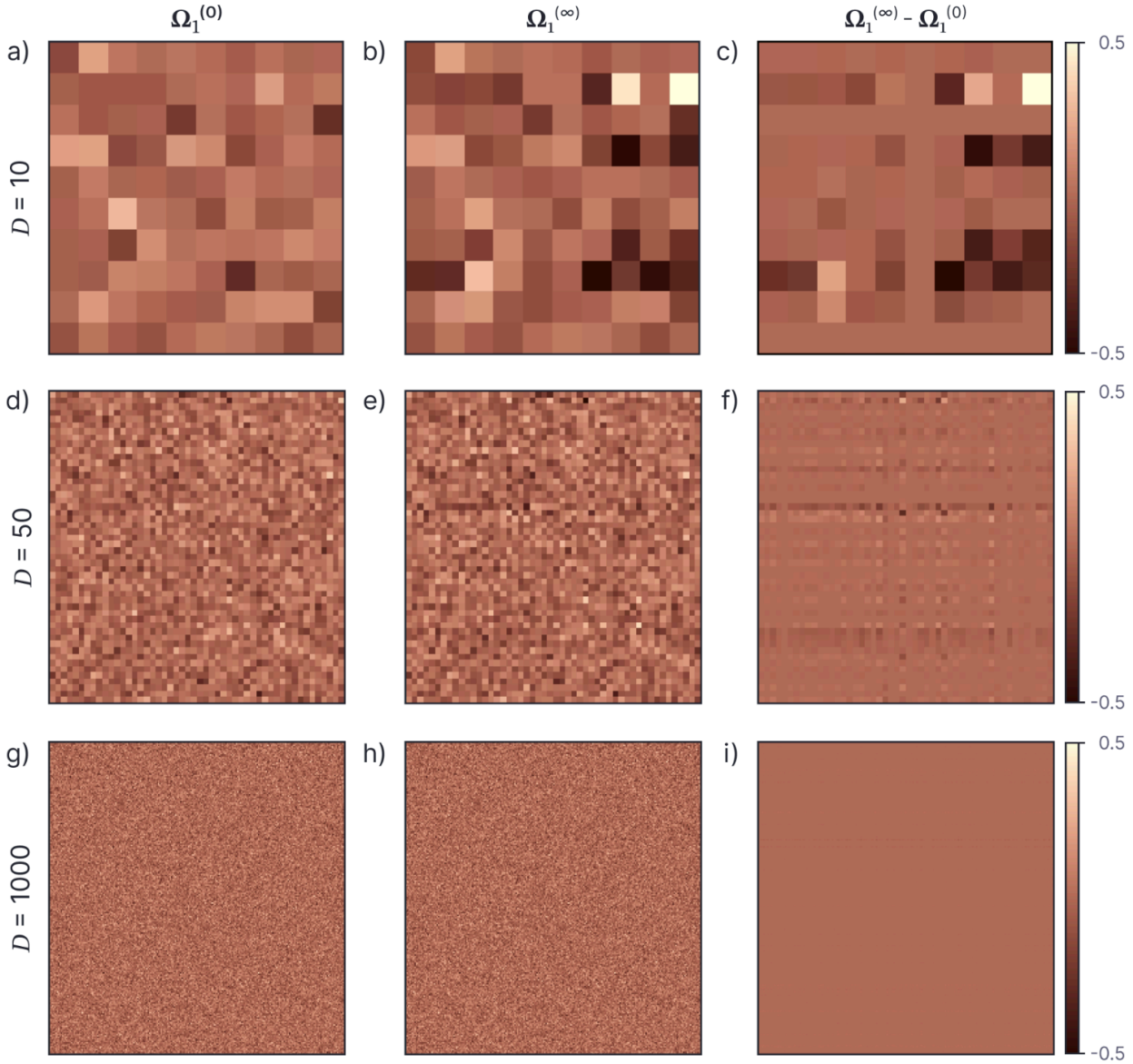


Figure 1. Change in weights in two layer neural network as the width D changes. a) Weights $\mathbf{\Omega}_1$ between first and second hidden layers before training with width $D = 10$. b) Weights $\mathbf{\Omega}_1$ after training. c) Change in weights during training. d-f) Weight matrix before training, weight matrix after training, and change in weight matrix for width $D = 50$ and g-i) width $D = 1000$. As the width increases, the individual weights change less, and at large widths are almost constant throughout training. Figure inspired by *Dwaraknath*(2019).

- Figure 2a depicts the **Frobenius norms of original weights and their changes** increasing as the width increases as the growth of weights.
- Figure 2b depicts the **norm of the change in weights** is roughly constant; but this change is distributed across more parameters as the width increases, so the individual weights change less.
- The tendency of the weights to remain nearly constant by the **ratio of the norm of the change to the norm of the original weights** (figure 2c).

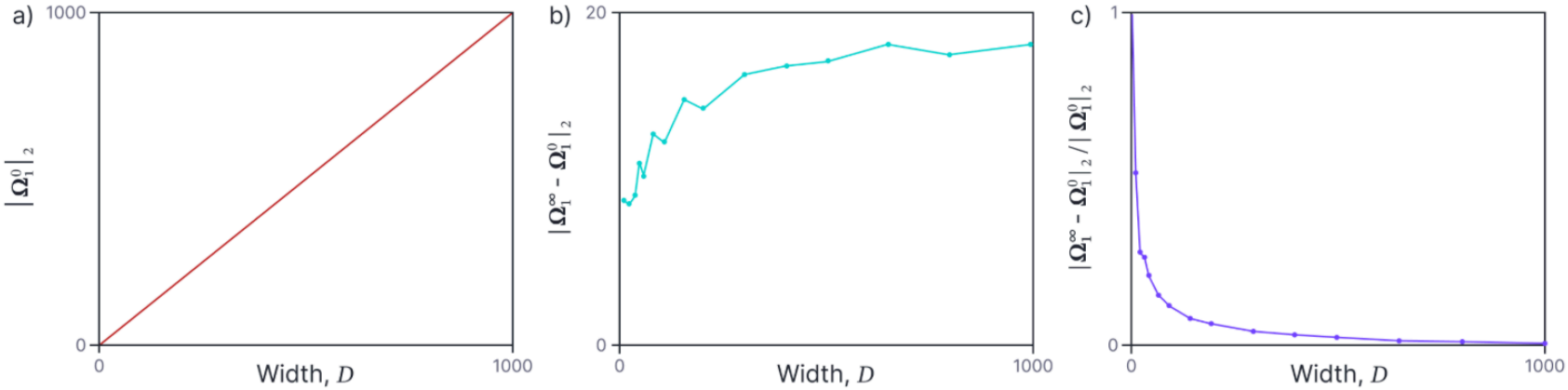


Figure 2. Change in weights in two layer neural network as the width D changes. a) The Frobenius norm $|\Omega_1^0|_2$ of the initial weight matrix increases as the width increases (there are more weights, each with the same amount of variation). b) The norm of the change in weights $|\Omega_1^\infty - \Omega_1^0|_2$ is roughly constant as the width increases (but this change is now distributed across more weights). c) Consequently, the norm of the change relative to the initial norm $|\Omega_1^\infty - \Omega_1^0|_2 / |\Omega_1^0|_2$ decreases with width.

2.3 Mathematical characterization of weight change

- Neural network with D hidden unit:

$$f(x, \boldsymbol{\phi}) = \frac{1}{\sqrt{D}} \sum_{d=1}^{D} \theta_d \cdot a[\phi_d x] \tag{2.1}$$

where $a[\cdot]$ is a Lipschitz activation function, and **the weights θ_d** that map from the hidden layer to the output are assigned randomly to ± 1 and fixed.

- Parameters are initialized from a **standard normal distribution**.

- **Loss function**:

$$L[\boldsymbol{\phi}] = \frac{1}{2} \sum_{i=1}^{I} \left(f(x, \boldsymbol{\phi}) - y_i \right)^2 . \tag{2.2}$$

2.4 Analysis of gradient flow

- With gradient flow, the **evolution of a parameters** ϕ_d is defined by:

$$\frac{\mathrm{d}\phi_d}{\mathrm{d}t} = -\frac{\partial L}{\partial \phi_d} = -\frac{1}{\sqrt{D}} \left(f(x, \boldsymbol{\phi}) - y_i\right) \theta_d x_i \cdot a'[\phi_d x]. \tag{2.3}$$

- Now consider the total change in a parameter over time:

$$\left\|\boldsymbol{\phi}_d[t] - \boldsymbol{\phi}_d[0]\right\|_2 \leqslant \int_0^T \left\| \frac{\mathrm{d}\boldsymbol{\phi}_d[t]}{\mathrm{d}t} \right\|_2 = \mathcal{O}\Big[\frac{1}{\sqrt{D}}\Big]. \tag{2.4}$$

  - $\|f(x, \boldsymbol{\phi}) - y_i\|$ decreases over time
  - $\theta_d x_i$ is constant
  - $a'[\phi_d x]$ is limited by steepest slope of the activation function

- Consequently, the change in the weight is dominated by $1/\sqrt{D}$ and becomes infinitesimal as $D \to \infty$ (Jacot et al. 2018).

2.5 Linear approximation

Taking a Taylor expansion around the initial parameters $\boldsymbol{\phi}_0$, we could approximate the network output as:

$$f(x, \boldsymbol{\phi}) \approx f(x, \boldsymbol{\phi}) + \left[\frac{\partial f(x, \boldsymbol{\phi}_0)}{\partial \boldsymbol{\phi}_0}\right]^\top (\boldsymbol{\phi} - \boldsymbol{\phi}_0),$$

where $\boldsymbol{\phi}$ contains the current parameters.

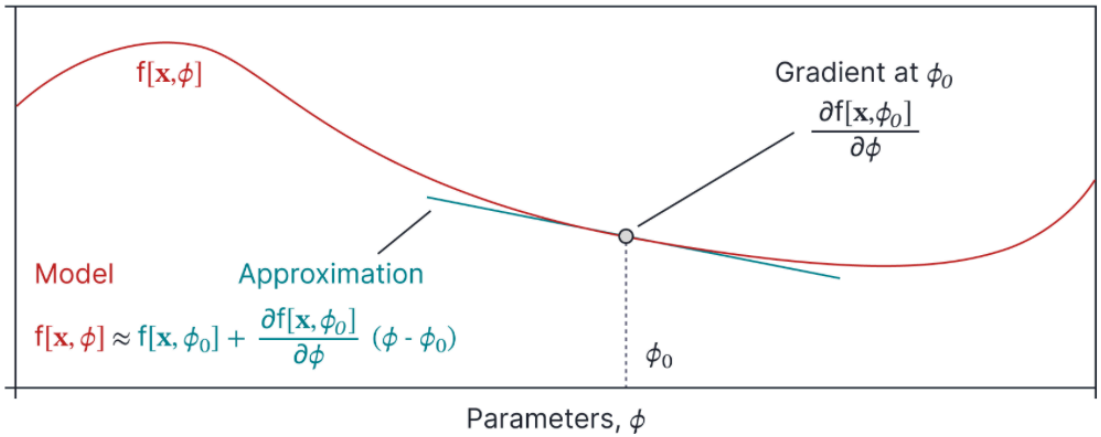


Figure 3. Taylor approximation. For a fixed input $\mathbf{x}$, the output $\mathrm{f}[\mathbf{x}, \boldsymbol{\phi}]$ of the model changes as a function of the choice of parameters $\boldsymbol{\phi}$ (orange curve). This relationship can be locally approximated by a linear function around initial parameters $\boldsymbol{\phi}_0$ (blue line), when the deviation from $\boldsymbol{\phi}_0$ is small.

When this linear approximation is valid, then

- We can write **closed-form expressions for the training evolution** of the loss and the parameters.
- We can also find **closed-form solutions for the final predictions**.
- It follows that if the network is approximately linear at large widths, there would be analogous results for neural networks, and these could provide valuable **insights about trainability, convergence, and generalization**.

2.6 Validity of linear approximation

- The linear approximation will be valid if **the curvature of with respect to the parameters is small** in the limited region;
- More formally, Liu et al. (2020) identified that the **spectral norm of the Hessian matrix** containing the second derivatives must be small compared to the magnitude of the gradient in a ball of a certain radius.

$$f(\mathbf{x}) \approx f(\mathbf{x}_0) + \nabla f(\mathbf{x}_0)(\mathbf{x} - \mathbf{x}_0) + \frac{1}{2}(\mathbf{x} - \mathbf{x}_0)^T \mathbf{H}(\mathbf{x}_0)(\mathbf{x} - \mathbf{x}_0)$$

- The Hessian matrix $\mathbf{H} \in \mathbb{R}^{D \times D}$ of the toy neural network is **diagonal** with values:

$$\mathbf{H}_{dd} = \frac{\partial^2 f(x, \boldsymbol{\phi})}{\partial \phi_d^2} = \frac{1}{\sqrt{D}} \theta_d a''[\phi_d x] x^2.$$

- If $\|x\| \leqslant C$, then:

$$\|\mathbf{H}\| = \max_d \left[\|\mathbf{H}\|_{dd}\right] = \frac{x^2}{\sqrt{D}} \max_d \left\{\theta_d a''[\phi_d x]\right\} \leqslant \frac{C^2}{\sqrt{D}} \max_d \left\{\theta_d a''[\phi_d x]\right\} \leqslant \frac{C^2 A}{\sqrt{D}}$$

where A is the maximum value of the second derivative $a''[\cdot]$.

- It's clear that as $D \to \infty$, the **Hessian norm converges to zero**.

- The magnitude of the **derivative vector**:

$$\sum_{d=1}^{D}\left(\frac{\partial f(x,\boldsymbol{\phi})}{\partial \phi_d}\right)^2 = \frac{1}{D}\sum_{d=1}^{D} x^2 a'[\phi_d x],$$

where the θ_d^2 terms have disappeared as they equal one regardless of whether they were ± 1.

- It's easy to see that the **magnitude of the gradient stays roughly constant** as $D \to \infty$ because the factor $1/D$ cancels with the D terms in the sum.

- It follows that as the width D increases, the curvature (Hessian) decreases, but the gradient (derivative vector) stays constant, and so **the linear approximation is reasonable** for this simple network when the width becomes very large.

- A more general proof for multi-layer networks with multiple inputs is provided by Liu et al. (2020).

2.7 Derivative vector as non-linear transformation

As the hidden layers of neural networks become infinitely large:

- The change in any **individual parameter** from their initialized values during training **decreases**
- The magnitude of the **gradient** of the function with respect to the parameter vector at initialization stays approximately **constant**
- The **second derivatives** of the function with respect to the parameter vector at initialization **decrease**

We can write:

$$f(\mathbf{x}, \boldsymbol{\phi}) \approx f^{lin}(\mathbf{x}, \boldsymbol{\phi}) = f(\mathbf{x}, \boldsymbol{\phi}_0) + \left[\frac{\partial f(\mathbf{x}, \boldsymbol{\phi}_0)}{\partial \boldsymbol{\phi}_0}\right]^\top (\boldsymbol{\phi} - \boldsymbol{\phi}_0),$$

where $\mathbf{x}$ is an input example, and $\boldsymbol{\phi}_0$ is a vector containing all of the initial weights and biases.

2.8 Interpretation

- A linear function of a fixed **non-linear transformation** $g(\mathbf{x})$ of the input $\mathbf{x}$:

$$f^{lin}(\mathbf{x}, \boldsymbol{\phi}) = f(\mathbf{x}, \boldsymbol{\phi}_0) + g(\mathbf{x})^\top (\boldsymbol{\phi} - \boldsymbol{\phi}_0),$$

where

$$g(\mathbf{x}) = \frac{\partial f(\mathbf{x}, \boldsymbol{\phi}_0)}{\partial \boldsymbol{\phi}_0}.$$

- Note that the **kernel methods** which rely on dot products between non-linear transformations $g(\mathbf{x}_i)$ and $g(\mathbf{x}_j)$ of input data examples $\mathbf{x}_i$, $\mathbf{x}_j$.

- The term **neural** derives from neural networks, the term **tangent** from the first term in the Taylor expansion (see figure 3), and the term **kernel** from the interpretation of the gradient vector as a non-linear transformation of the input data.

Training dynamics

- In the previous section, we saw that in the infinite width limit, the network predictions for an input x become a **linear model** with parameters $\boldsymbol{\phi}$
- In part I of this series, we found **closed–form expressions for the training dynamics** for linear models
- Now, we combine these observations to find **closed-form expressions for the training dynamics of neural networks** in the infinite limit

For convenience,

- store the I training data vectors $\{\mathbf{x}_i\}$ in the columns of a matrix $\mathbf{X} \in \mathbb{R}^{D \times I}$
- store the targets $\{y_i\}$ in a column vector $\mathbf{y} \in \mathbb{R}^{I \times 1}$
- A neural network model $f(\mathbf{X}, \boldsymbol{\phi})$ is applied to all of the data simultaneously and produces an $I \times 1$ column vector whose i^{th} entry is $f(\mathbf{x}_i, \boldsymbol{\phi})$

- The linear model can now be written as:

$$f^{lin}(\mathbf{X}, \boldsymbol{\phi}) = f(\mathbf{X}, \boldsymbol{\phi}) + \left[\frac{\partial f(\mathbf{X}, \boldsymbol{\phi}_0)}{\partial \boldsymbol{\phi}_0}\right]^\top (\boldsymbol{\phi} - \boldsymbol{\phi}_0).$$

- In part I of this series, we showed that the ODE that governs the evolution of the **residual differences** between model predictions $f(\mathbf{X}, \boldsymbol{\phi})$ and the ground truth targets $\mathbf{y}$ is:

$$\frac{\mathrm{d}}{\mathrm{d}t}\left[f(\mathbf{X}, \boldsymbol{\phi}) - \mathbf{y}\right] = -\left(\left[\frac{\partial f(\mathbf{X}, \boldsymbol{\phi}_0)}{\partial \boldsymbol{\phi}} \frac{\partial f(\mathbf{X}, \boldsymbol{\phi}_0)}{\partial \boldsymbol{\phi}}\right]^\top\right)\left[f(\mathbf{X}, \boldsymbol{\phi}) - \mathbf{y}\right].$$

- This ODE has a closed form solution:

$$f(\mathbf{X}, \boldsymbol{\phi}_t) = \exp\left\{-\left(\left[\frac{\partial \mathbf{f}(\mathbf{X}, \boldsymbol{\phi}_\mathbf{0})}{\partial \boldsymbol{\phi}} \frac{\partial \mathbf{f}(\mathbf{X}, \boldsymbol{\phi}_\mathbf{0})}{\partial \boldsymbol{\phi}}\right]^\top t\right)\right\}\left[f(\mathbf{X}, \boldsymbol{\phi}) - \mathbf{y}\right] + \mathbf{y}. \quad (3.1)$$

3.1 The neural tangent kernel

- We now refer to the $I \times I$ matrix from equation (3.1) as the *neural tangent kernel* (**NTK**):

$$\mathbf{NTK}[\mathbf{X}, \mathbf{X}] = \frac{\partial f(\mathbf{X}, \boldsymbol{\phi}_0)}{\partial \boldsymbol{\phi}} \frac{\partial f(\mathbf{X}, \boldsymbol{\phi}_0)}{\partial \boldsymbol{\phi}}^\top$$

- If we could compute this term, then we would have a **closed–form solution for the evolution of the residuals**, and the contents of this matrix would shine a light on the trainability and convergence speed of the system.

- In principle could compute the **NTK** for a neural network by simply computing the derivatives for every training example $\mathbf{x}_i$ using the backpropagation algorithm and taking the appropriate dot products.

- However, it turns out that for **some models**, we can get a closed form solution for the **NTK**.

- The **NTK** measures the change in $f(x, \theta)$ when using SGD to optimize parameters, corresponding to a random sample x, after a very small step size η in parameter updates.

- In particular:

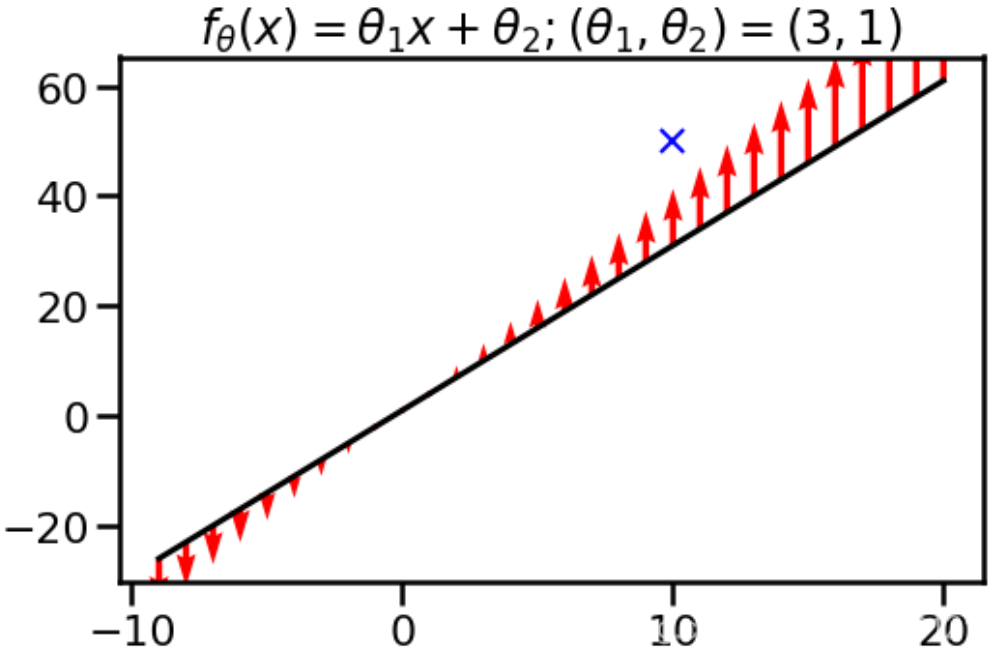
$$\mathbf{NTK}(x, x') = \lim_{\eta \to 0} \frac{f\left(x, \theta + \eta \frac{df(x', \theta)}{d\theta}\right) - f(x, \theta)}{\eta}$$

- Using a 1st order Taylor expansion of $f_\theta(x)$, it is possible to show that

$$\mathbf{NTK}_\theta(x, x') = \left\langle \frac{df(x, \theta)}{d\theta}, \frac{df(x', \theta)}{d\theta} \right\rangle$$

3.2 An intuitive understanding of the NTK

- Consider an example of a linear function, defined as $f(x, \theta) = \theta_1 x + \theta_2$
- Initialize the parameters as $\theta_1 = 3$ and $\theta_2 = 1$.
- Consider a sample point $(x, y) = (10, 50)$. Based on this sample, perform a gradient descent update on the parameters.



Empirical NTK for a shallow network

- Consider a **shallow neural network** with D hidden units:

$$f(\mathbf{x}, \boldsymbol{\phi}) = \frac{1}{\sqrt{D}} \sum_{d=1}^{D} w_{1d} \cdot a[\boldsymbol{w}_{0d}\mathbf{x}] \tag{4.1}$$

- The derivatives with regards to the two sets of parameters are:

$$\frac{\partial f(\mathbf{x}, \boldsymbol{\phi})}{\partial \boldsymbol{w}_{0d}} = \frac{1}{\sqrt{D}} \sum_{d=1}^{D} a'[\boldsymbol{w}_{0d}\mathbf{x}] \cdot w_{1d}\mathbf{x},$$

$$\frac{\partial f(\mathbf{x}, \boldsymbol{\phi})}{\partial w_{1d}} = \frac{1}{\sqrt{D}} a[\boldsymbol{w}_{0d}\mathbf{x}].$$

- Since the **inner product** of the concatenated derivatives just the sum of the inner products of the individual derivatives, we can write a closed form expression for the kernel:

$$\begin{aligned}
\mathbf{NTK}[\mathbf{x}_i, \mathbf{x}_j] &= \sum_{d=1}^{D} \frac{\partial f(\mathbf{x}_i, \boldsymbol{\phi})}{\partial \boldsymbol{w}_{0d}} \frac{\partial f(\mathbf{x}_j, \boldsymbol{\phi})}{\partial \boldsymbol{w}_{0d}} + \frac{\partial f(\mathbf{x}_i, \boldsymbol{\phi})}{\partial w_{1d}} \frac{\partial f(\mathbf{x}_j, \boldsymbol{\phi})}{\partial w_{1d}} \\
&= \frac{1}{\sqrt{D}} \sum_{d=1}^{D} w_{1d}^2 a'[\boldsymbol{w}_{0d}\mathbf{x}_i] a'[\boldsymbol{w}_{0d}\mathbf{x}_j] \mathbf{x}_j^\top \mathbf{x}_j + a[\boldsymbol{w}_{0d}\mathbf{x}_i] a[\boldsymbol{w}_{0d}\mathbf{x}_j].
\end{aligned}$$

The ReLU

To make this concrete, we consider using the rectified linear unit (ReLU):

$$a[z] = \mathbf{ReLU}[z] = \begin{cases} 0\,, & z < 0 \\ z\,, & z \geqslant 0 \end{cases}$$

which has the derivative (figure 4):

$$a'[z] = \frac{\mathrm{d}\mathbf{ReLU}[z]}{\mathrm{d}z} = \begin{cases} 0\,, & z < 0 \\ 1\,, & z \geqslant 0 \end{cases}$$

or $\mathbb{I}[z > 0]$ for short.

The rectified linear unit (ReLU) activation function.

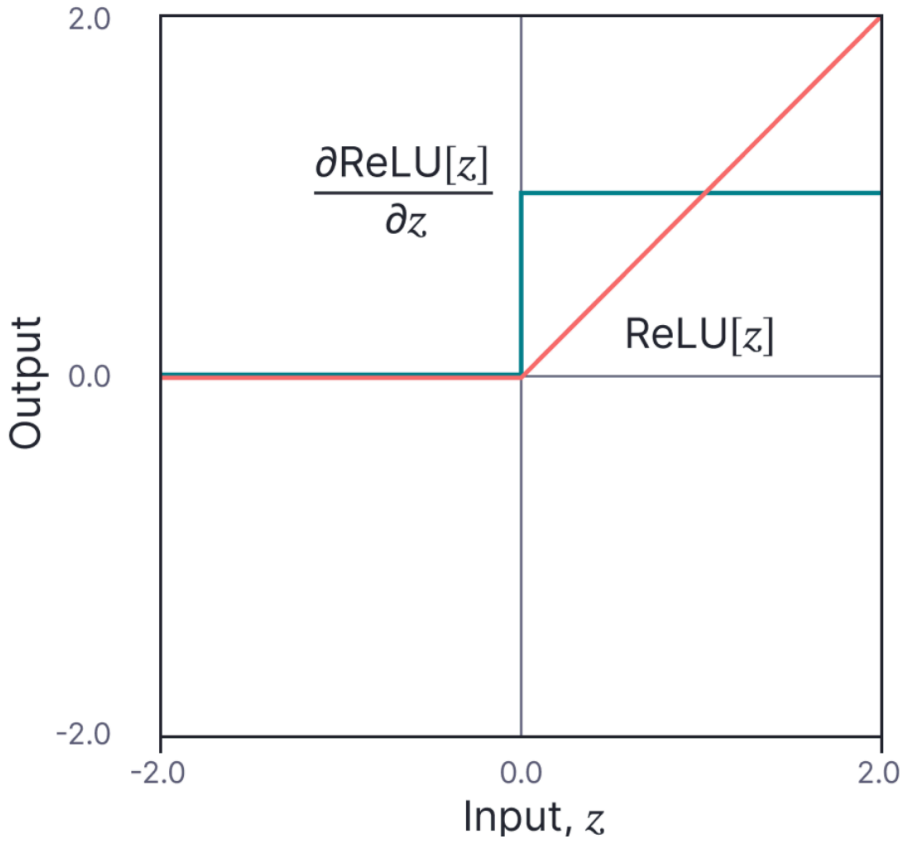


Figure 4. The rectified linear unit (ReLU) activation function.

The neural tangent kernel

The neural tangent kernel is given by:

$$\mathbf{NTK}[\mathbf{x}_i, \mathbf{x}_j] = \frac{1}{\sqrt{D}} \sum_{d=1}^{D} w_{1d}^2 \mathbb{I}[\boldsymbol{w}_{0d}\mathbf{x}_i > 0]\mathbb{I}[\boldsymbol{w}_{0d}\mathbf{x}_j > 0]\mathbf{x}_j^\top \mathbf{x}_j + \mathbf{ReLU}[\boldsymbol{w}_{0d}\mathbf{x}_i]\mathbf{ReLU}[\boldsymbol{w}_{0d}\mathbf{x}_j]. \tag{4.2}$$

- The first term is the component due to the derivatives in $\boldsymbol{w}_{0d}$
- The second term is the component due to the derivatives in w_{1d}
- For finite width networks, the neural tangent kernel (however it was calculated) depends on the particular random draw of the parameters and here it's referred to as the empirical tangent kernel.

Analytical NTK for shallow network

Let $D \to \infty$, then the expectation in equation (4.2) becomes:

$$\mathbf{NTK}[\mathbf{x}_i, \mathbf{x}_j] = E_{\boldsymbol{w}_0, \boldsymbol{w}_1} \left\{ \boldsymbol{w}_1^2 \mathbb{I}[\boldsymbol{w}_0 \mathbf{x}_i > 0] \mathbb{I}[\boldsymbol{w}_0 \mathbf{x}_j > 0] \mathbf{x}_i^\top \mathbf{x}_j \right.$$
$$\left. + \mathbf{ReLU}[\boldsymbol{w}_0 \mathbf{x}_i] \mathbf{ReLU}[\boldsymbol{w}_0 \mathbf{x}_j] \right\}.$$

For variables $\mathbf{z} = [z_i, z_j]^\top$ with mean $\mathbf{0}$ and covariance:

$$\boldsymbol{\Sigma} = \begin{bmatrix} \sigma_i^2 & \sigma_{ij}^2 \\ \sigma_{ij}^2 & \sigma_j^2 \end{bmatrix},$$

it can (non-obviously) be shown (Cho & Saul 2009; Golikov *et al.* 2022) that

$$E\left[\mathbf{ReLU}[z_i] \cdot \mathbf{ReLU}[z_j]\right] = \frac{\sigma_i \sigma_j \left[\cos(\theta) \cdot (\pi - \theta) + \sin(\theta)\right]}{2\pi}$$

$$E\left[\mathbb{I}[z_i > 0] \cdot \mathbb{I}[z_j > 0]\right] = (\pi - \theta)/2\pi$$

where

$$\theta = \arccos\left[\frac{\sigma_{ij}^2}{\sigma_i \sigma_j}\right].$$

- Assuming that both sets of weights are **initialized from a standard normal distribution**, we have $\sigma_i = |\mathbf{x}_i|$, $\sigma_j = |\mathbf{x}_j|$ and $\sigma_{ij} = \mathbf{x}_i^\top \mathbf{x}_j$ which gives the final result:

$$\mathbf{NTK}[\mathbf{x}_i, \mathbf{x}_j] = \frac{\mathbf{x}_i^\top \mathbf{x}_j}{2\pi}(\pi - \theta) + \frac{|\mathbf{x}_i| \cdot |\mathbf{x}_j|}{2\pi}\left[(\pi - \theta)\cos\theta + \sin\theta\right]. \quad (5.1)$$

- Note that in the **infinite limit, the kernel is constant**; it does not depend on the initial choices for the parameters. We term this the analytical kernel.

- This result can be extended to **deep neural networks** where the kernel at **each layer** is computed as a function of the kernel at the previous layer.

- The analytical NTK also been calculated for **convolutional networks** (Arora et al., 2019, Li et al. 2019) and other common architectures.

Empirical NTK for a deep network

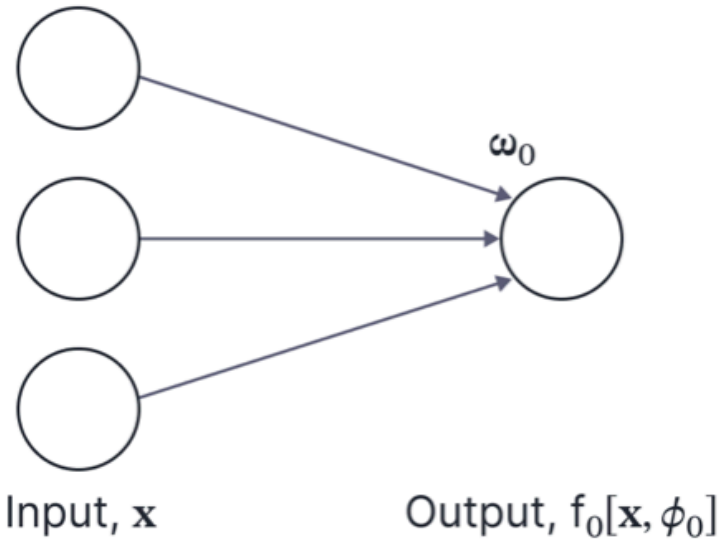
- We can similarly compute the NTK for a **deep neural network**.
- The NTK is computed **iteratively**, by expressing the NTK for a depth L network in terms of the NTK for a depth $L-1$ network.
- It is rather mathematically involved and can be skipped on the first reading.

Linear model, one output

- Consider a linear model $f_0(\mathbf{x}, \boldsymbol{\phi}_0)$ with D outputs

$$f_0(\mathbf{x}, \boldsymbol{\phi}_0) = \beta_0 + \frac{1}{\sqrt{D_{in}}}\mathbf{w}_0\mathbf{x}$$

where D_{in} is the **input data dimension** and the parameters $\boldsymbol{\phi}_0 = \{\beta_0, \boldsymbol{w}_0\}$ comprise the bias $\beta_0 \in \mathbb{R}$ and the weights $\boldsymbol{w}_0 \in \mathbb{R}^{1\times D_{in}}$.



- The value of the **NTK** at position (i, j) is:

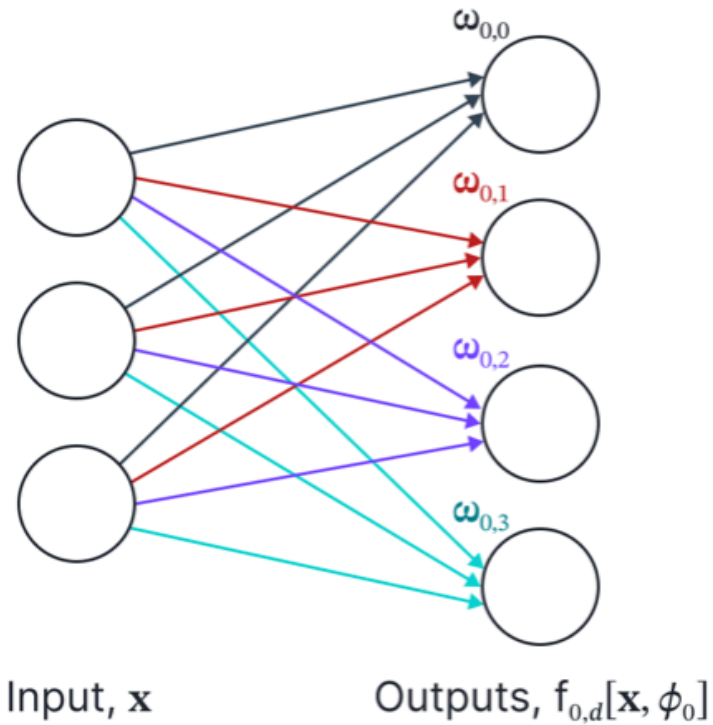
$$\begin{aligned}\mathbf{NTK}[\mathbf{x}_i, \mathbf{x}_j] &= \frac{\partial f(\mathbf{x}_i, \boldsymbol{\phi}_0)}{\partial \boldsymbol{\phi}_0}^\top \frac{\partial f(\mathbf{x}_j, \boldsymbol{\phi}_0)}{\partial \boldsymbol{\phi}} \\ &= \frac{\partial f(\mathbf{x}_i, \boldsymbol{\phi}_0)}{\partial \boldsymbol{w}_0}^\top \frac{\partial f(\mathbf{x}_j, \boldsymbol{\phi}_0)}{\partial \boldsymbol{w}_0} + \frac{\partial f(\mathbf{x}_i, \boldsymbol{\phi}_0)}{\partial \beta_0}^\top \frac{\partial f(\mathbf{x}_j, \boldsymbol{\phi}_0)}{\partial \beta_0} \\ &= \frac{1}{D_{in}} \mathbf{x}_i^\top \mathbf{x}_j + 1.\end{aligned}$$

Linear model, D output

- We start with a linear model $f_0(\mathbf{x}, \boldsymbol{\phi}_0)$:

$$f_{0,d}(\mathbf{x}, \boldsymbol{\phi}_0) = \beta_{0,d} + \frac{1}{\sqrt{D_{in}}} \boldsymbol{w}_{0,d}\mathbf{x} \tag{6.1}$$

where $\beta_{0,d} \in \mathbb{R}$ and $\boldsymbol{w}_{0,d} \in \mathbb{R}^{1\times D_{in}}$.



- The neural tangent kernel now has $ID \times ID$ elements which correspond to the dot product of the gradients for every pair of the I input vectors and D outputs. It is arranged to form $I \times I$ sub-kernels $\mathbf{NTK}_{d,d'}[\mathbf{x}_i, \mathbf{x}_j]$ each of size $D \times D$ and containing sub-elements.

- By the same logic as the previous section each has the same value:

$$\mathbf{NTK}_{d,d'}[\mathbf{x}_i, \mathbf{x}_j] = \frac{1}{D_{in}} \mathbf{x}_i^\top \mathbf{x}_j + 1.$$

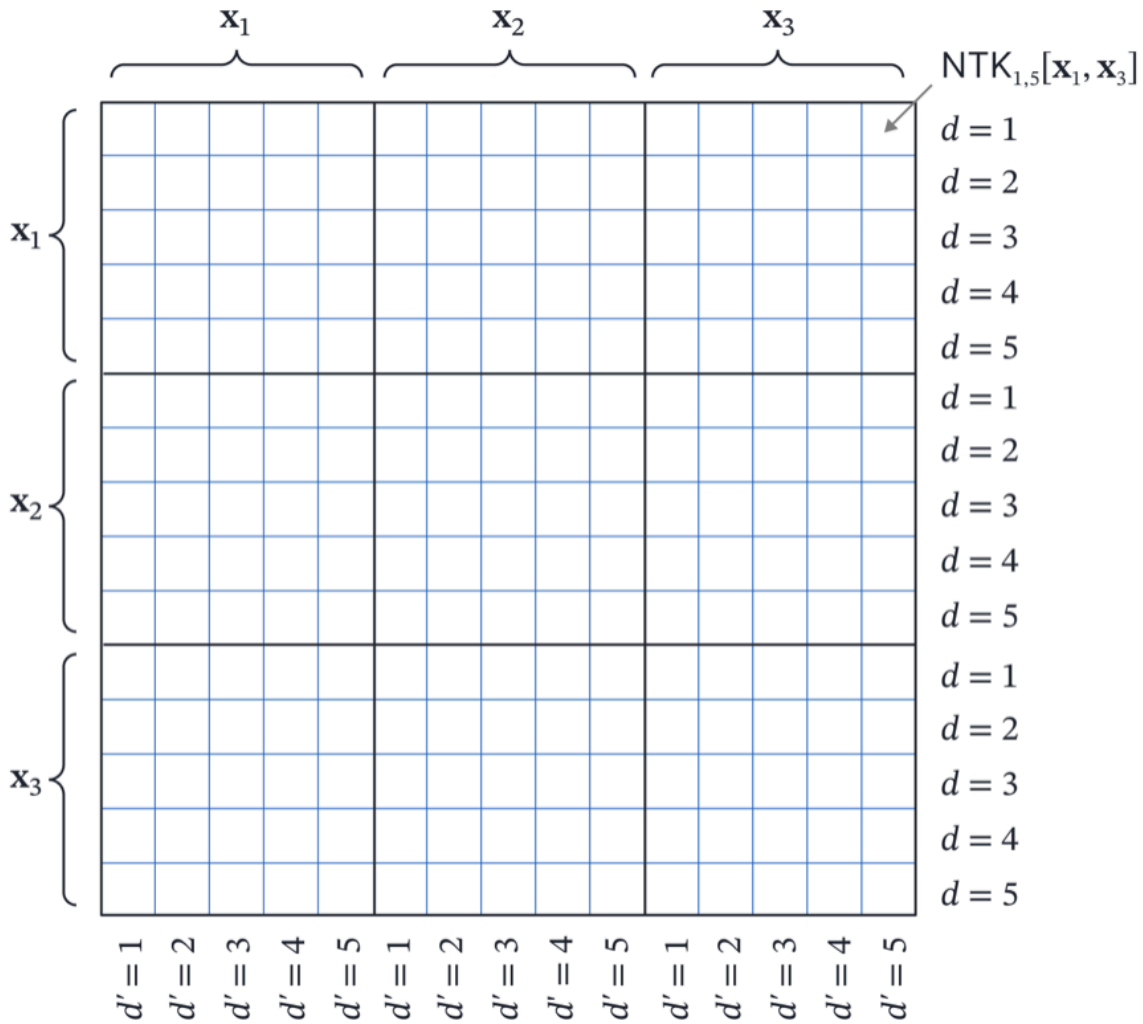


Figure 7. Kernel structure for linear model with D outputs. For I inputs, the kernel matrix now consists of $I \times I$ subkernels, each of size $D \times D$.

Fully connected network with one layer

- Consider a **fully connected network** $f_1(\mathbf{x}, \boldsymbol{\phi}_1)$ with a single output and D hidden layer containing D neurons, associated parameters

$$\boldsymbol{\phi}_1 = \{\beta_0, \Omega_0, \beta_1, \boldsymbol{w}_1\}$$

and activation function $\mathbf{a}[\cdot]$:

$$f_1(\mathbf{x}, \boldsymbol{\phi}_1) = \beta_1 + \frac{1}{\sqrt{D}} \boldsymbol{w}_1 \mathbf{a}\left[f_0(\mathbf{x}, \boldsymbol{\phi}_0)\right] \tag{6.2}$$

where the preactivation $f_{0,d}[\mathbf{x}, \boldsymbol{\phi}]$ at each hidden unit is the linear function discussed above:

$$f_{0,d}(\mathbf{x}, \boldsymbol{\phi}_0) = \beta_d + \frac{1}{\sqrt{D_{in}}} \boldsymbol{w}_{0d} \mathbf{x} \tag{6.3}$$

and $\boldsymbol{\phi}_0 = \{\beta_{0,1}, \ldots, \beta_{0,D_{in}}, \boldsymbol{w}_{0,1}, \ldots, \boldsymbol{w}_{0,D_{in}}\}$

Plot of fully connected network with one layer:

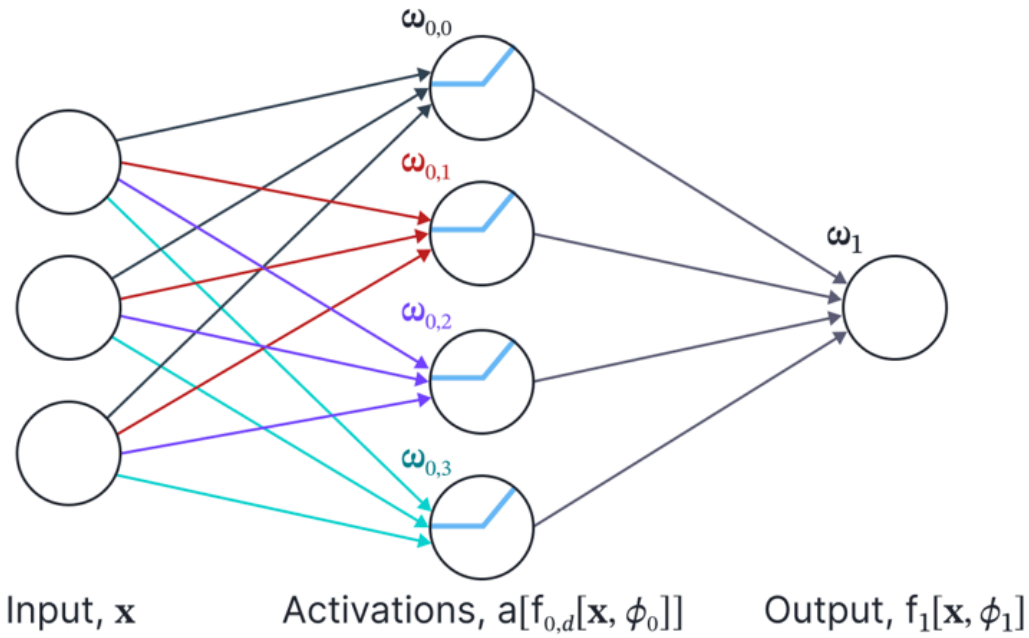


Figure 8. Building up the NTK computation for a deep neural network. We pass each of the outputs from figure 6 through an activation function $\mathbf{a}[\bullet]$ and recombine using bias $\beta_{1,1}$ (not shown) and weights $\boldsymbol{\omega}_1$ to create a neural network $\mathbf{f}[\mathbf{x}, \boldsymbol{\phi}_1]$ with a single hidden layer.

- The derivatives of $f_1(\mathbf{x}, \boldsymbol{\phi}_1)$ with respect to the unknown quantities are:

$$\frac{\partial f_1(\mathbf{x}, \boldsymbol{\phi}_1)}{\partial \boldsymbol{w}_1} = \frac{1}{\sqrt{D}} \mathbf{a}'[f_0[\mathbf{x}, \boldsymbol{\phi}_0]]$$

$$\frac{\partial f_1(\mathbf{x}, \boldsymbol{\phi}_1)}{\partial \beta_1} = 1$$

$$\frac{\partial f(\mathbf{x}, \boldsymbol{\phi})}{\partial \boldsymbol{\phi}_0} = \frac{1}{\sqrt{D}} \left(\mathbf{1}\mathbf{a}'[f_0[\mathbf{x}, \boldsymbol{\phi}_0]] \bigodot \frac{\partial f_0(\mathbf{x}, \boldsymbol{\phi}_0)}{\partial \boldsymbol{\phi}_0} \right) \boldsymbol{w}_1^\top, \tag{6.4}$$

where $\partial f_0(\mathbf{x}, \boldsymbol{\phi}_0)/\partial \boldsymbol{\phi}_0$ is a $|\boldsymbol{\phi}_0| \times D$ matrix which is block diagonal since only D_{in} weights and one bias in $\boldsymbol{\phi}_0$ modify any particular element of the activation vector. The term $\mathbf{1}$ is a $|\boldsymbol{\phi}| \times 1$ column vector containing only ones, and $\bigodot$ represents pointwise multiplication.

- Now we compute the elements $\mathbf{NTK}^{(1)}$ of the kernel for this shallow network by taking the dot products of the derivatives with respect to the parameters:

$$\mathbf{NTK}^{(1)}[\mathbf{x}_i, \mathbf{x}_j] = \frac{\partial f_1(\mathbf{x}_i, \boldsymbol{\phi}_1)}{\partial \boldsymbol{w}_1}^\top \frac{\partial f_1(\mathbf{x}_j, \boldsymbol{\phi}_1)}{\partial \boldsymbol{w}_1} + \frac{\partial f_1(\mathbf{x}_i, \boldsymbol{\phi}_1)}{\partial \beta_1}^\top \frac{\partial f_1(\mathbf{x}_j, \boldsymbol{\phi}_1)}{\partial \beta_1} + \frac{\partial f(\mathbf{x}_i, \boldsymbol{\phi}_1)}{\partial \boldsymbol{\phi}_0}^\top \frac{\partial f(\mathbf{x}_j, \boldsymbol{\phi}_1)}{\partial \boldsymbol{\phi}_0}.$$

- Substituting in the derivatives from equation (6.4), we see that:

$$\mathbf{NTK}^{(1)}[\mathbf{x}_i, \mathbf{x}_j] = \frac{1}{D}\Bigg(\mathbf{a}[f_0(\mathbf{x}_i, \boldsymbol{\phi}_0)]^\top \mathbf{a}[f_0(\mathbf{x}_j, \boldsymbol{\phi}_0)] + D + \boldsymbol{w}_1 \left(\mathbf{a}'[f_0(\mathbf{x}_i, \boldsymbol{\phi}_0)]\mathbf{a}'[f_0(\mathbf{x}_j, \boldsymbol{\phi}_0)]^\top \odot \frac{\partial f_0(\mathbf{x}_i, \boldsymbol{\phi}_0)}{\partial \boldsymbol{\phi}_0}^\top \frac{\partial f_0(\mathbf{x}_j, \boldsymbol{\phi}_0)}{\partial \boldsymbol{\phi}_0}\right) \boldsymbol{w}_1^\top\Bigg).$$

- Noting that the product of the derivative terms is just the neural tangent kernel from the single layer network, we see that:

$$\mathbf{NTK}^{(1)}[\mathbf{x}_i, \mathbf{x}_j] = \frac{1}{D}\Bigg(\mathbf{a}[f_0(\mathbf{x}_i, \boldsymbol{\phi}_0)]^\top \mathbf{a}[f_0(\mathbf{x}_j, \boldsymbol{\phi}_0)] + D + \boldsymbol{w}_1\left(\mathbf{a}'[f_0(\mathbf{x}_i, \boldsymbol{\phi}_0)]\mathbf{a}'[f_0(\mathbf{x}_j, \boldsymbol{\phi}_0)]^\top \bigodot \mathbf{NTK}^{(0)}[\mathbf{x}_i, \mathbf{x}_j]\right)\boldsymbol{w}_1^\top\Bigg).$$

- Hence, the **NTK** for the network with one hidden layer can be written in terms of the **NTK** for a network with no hidden layers.

Recursive calculation of kernel

- In the previous equation, we the kernel $\mathbf{NTK}^{(1)}[\mathbf{x}_i, \mathbf{x}_j]$ associated with the single output of a shallow network in terms of the $D \times D$ subkernel $\mathbf{NTK}^{(0)}[\mathbf{x}_i, \mathbf{x}_j]$ of the preactivations. As might be expected, there is a general formula that relates the kernel values at layer l to those at layer $l-1$:

$$\begin{aligned}
\mathbf{NTK}^{(0)}_{d,d'}[\mathbf{x}_i, \mathbf{x}_j] &= \frac{1}{D_{in}} \mathbf{x}_i^\top \mathbf{x}_j + 1 \\
\mathbf{NTK}^{(l)}_{d,d'}[\mathbf{x}_i, \mathbf{x}_j] &= \frac{1}{D} \Big\{ \mathbf{a}[f_{l-1}(\mathbf{x}_i, \boldsymbol{\phi}_{l-1})]^\top \mathbf{a}[f_{l-1}(\mathbf{x}_j, \boldsymbol{\phi}_{l-1})] + D \\
&+ \boldsymbol{w}_{ld} \Big(\mathbf{a}'[f_{l-1}(\mathbf{x}_i, \boldsymbol{\phi}_{l-1})] \mathbf{a}'[f_{l-1}(\mathbf{x}_j, \boldsymbol{\phi}_{l-1})]^\top \bigodot \mathbf{NTK}^{(l-1)}[\mathbf{x}_i, \mathbf{x}_j] \Big) \boldsymbol{w}_{ld'}^\top \Big\}.
\end{aligned}$$

We can use this recursive formulation to calculate the kernel for networks of arbitrary depth

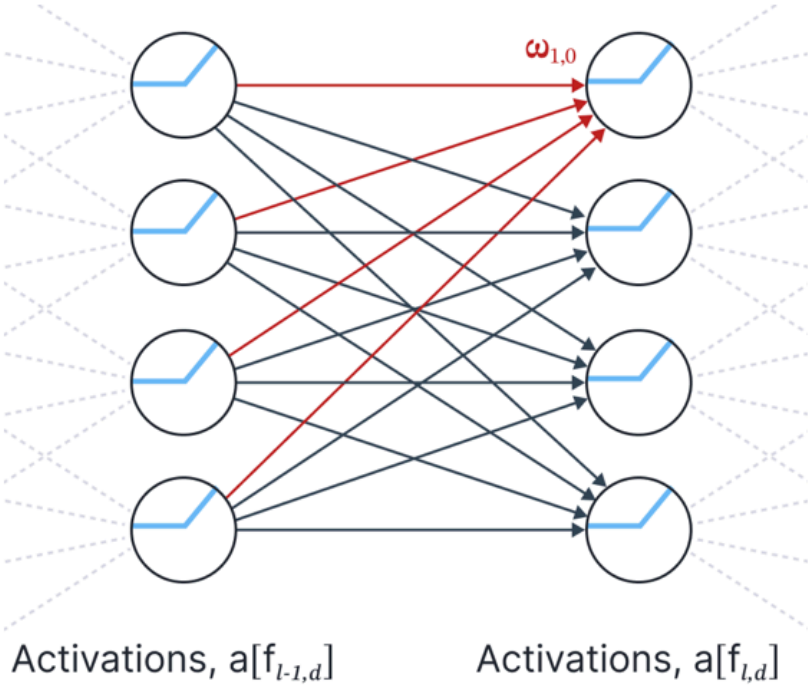


Figure 9. Building up the NTK computation for a deep neural network. Finally, we consider the general relationship between the NTK at layer l and layer $l-1$ of a deep network.

Analytical NTK for deep network

To compute the analytical **NTK**, we let the number of hidden units in each dimension become infinite. The expectations are

$$\begin{aligned}
\mathbf{NTK}^{(0)}_{d,d'}[\mathbf{x}_i, \mathbf{x}_j] &= \frac{1}{D_{in}} \mathbf{x}_i^\top \mathbf{x}_j + 1 \\
\mathbf{NTK}^{(l)}_{d,d'}[\mathbf{x}_i, \mathbf{x}_j] &= K_l[\mathbf{x}, \mathbf{x}'] + 1 + K'_l[\mathbf{x}_i, \mathbf{x}_j] \bigodot \mathbf{NTK}^{(l-1)}[\mathbf{x}_i, \mathbf{x}_j]
\end{aligned} \tag{7.1}$$

where:

$$\begin{aligned}
K_0[\mathbf{x}_i, \mathbf{x}_j] &= \mathbf{x}_i \mathbf{x}_j^\top + 1 \\
K'_0[\mathbf{x}_i, \mathbf{x}_j] &= \mathbf{x}_i \mathbf{x}_j^\top + 1 \\
K_l[\mathbf{x}_i, \mathbf{x}_j] &= E_{f_{l-1} \sim N[0, K_{l-1}]} \left[\mathbf{a}[f_{l-1}(\mathbf{x}_i, \boldsymbol{\phi}_{l-1})]^\top \mathbf{a}[f_{l-1}(\mathbf{x}_j, \boldsymbol{\phi}_{l-1})] \right] \\
K'_l[\mathbf{x}_i, \mathbf{x}_j] &= E_{f_{l-1} \sim N[0, K_{l-1}]} \left[\mathbf{a}'[f_{l-1}(\mathbf{x}_i, \boldsymbol{\phi}_{l-1})]^\top \mathbf{a}'[f_{l-1}(\mathbf{x}_j, \boldsymbol{\phi}_{l-1})] \right]
\end{aligned} \tag{7.2}$$

Analytical NTK for deep ReLU network

Assume that we use a ReLU for the activation functions, then the derivative is $a'[z] = I[z > 0]$, and we can again use the results that for expectations taken with respect to a normal distribution with mean zero and covariance:

$$\mathbf{\Sigma} = \begin{bmatrix} \sigma_i^2 & \sigma_{ij}^2 \\ \sigma_{ij}^2 & \sigma_j^2 \end{bmatrix},$$

we have

$$E_{f_{l-1}\sim N[0,K_{l-1}]}\left[\mathbf{ReLU}[z_i]\cdot\mathbf{ReLU}[z_j]\right] = \sigma_i\sigma_j\left(\cos[\theta]\cdot(\pi-\theta)+\sin[\theta]\right)/2\pi$$
$$E_{f_{l-1}\sim N[0,K_{l-1}]}\left[\mathbb{I}[z_i > 0]\cdot\mathbb{I}[z_j > 0]\right] = \pi - \theta/2\pi$$

where

$$\theta = \arccos\left[\frac{\sigma_{ij}^2}{\sigma_i\sigma_j}\right].$$

A cursory inspection of these equations reveals that substituting these results into the recursive formulae in equations (7.1)–(7.2) with $L = 1$ yields the **NTK** for a shallow ReLU network that we derived in equation (5.1).

Conclusion

Conclusion

- When neural networks become **very wide**, their **parameters do not change much** during training and they can be considered as approximately linear.
- This linearity means that we can write a **closed–form solution for the training dynamics**, and this closed–form solution depends critically on the **neural tangent kernel**.
- Each element of the neural tangent kernel consists of the inner product of the vectors of derivatives for **a pair of training data examples**.
- This can be calculated for any network and we call this the **empirical NTK**.
- If we let the width become infinite, then we can get closed–form solutions, which are referred to as **analytical NTKs**.
- We derived these solutions for both **shallow** and **deep ReLU networks**.

Q & A

THANK YOU For Your Attention!

References

Cho, Y., & Saul, L. (2009). Kernel methods for deep learning. Advances in neural information processing systems, 22.

Golikov, E., Pokonechnyy, E., & Korviakov, V. (2022). Neural tangent kernel: A survey. arXiv preprint arXiv:2208.13614.

He, K., Zhang, X., Ren, S., & Sun, J. (2015). Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In Proceedings of the IEEE international conference on computer vision (pp. 1026-1034).

Jacot, A., Gabriel, F., & Hongler, C. (2018). Neural tangent kernel: Convergence and generalization in neural networks. Advances in neural information processing systems, 31.

Lee, J., Xiao, L., Schoenholz, S., Bahri, Y., Novak, R., Sohl-Dickstein, J., & Pennington, J. (2019). Wide neural networks of any depth evolve as linear models under gradient descent. Advances in neural information processing systems, 32.

Liu, C., Zhu, L., & Belkin, M. (2020). On the linearity of large non-linear models: when and why the tangent kernel is constant. Advances in Neural Information Processing Systems, 33, 15954–15964.

Neal, R. M., & Neal, R. M. (1996). Priors for infinite networks. Bayesian learning for neural networks, 29-53.