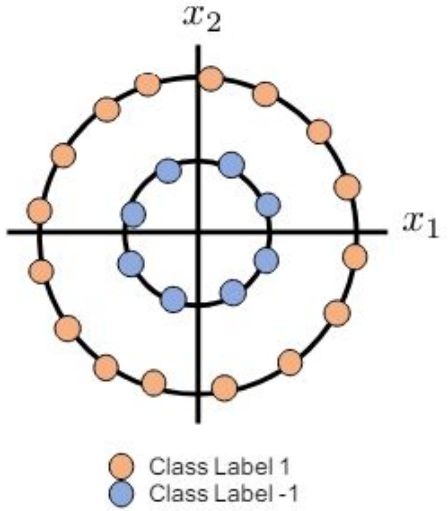
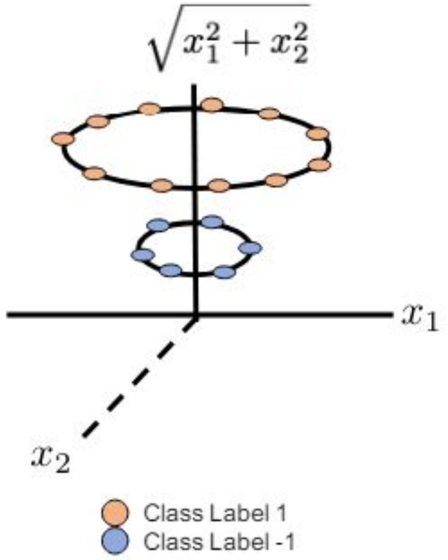
NTK Explanation and small scale Benchmark

Linear Regression vs Kernel



(a)



(b)

Polynomial Kernel

$$\begin{bmatrix} x1 \\ x2 \end{bmatrix} \quad \begin{bmatrix} y1 \\ y2 \end{bmatrix}$$

X and Y are two different data points

(x.T*y)^2 = (x1y1 + x2y2)^2

(x1)^2*(y1)^2 + (x2)^2*(y2)^2 + 2*x1*y1*x2*y2 (eqn.1)

To represent the above equation in vector form:

$$\begin{bmatrix} x1^2 \\ x2^2 \\ sqrt(2)*x1*x2 \end{bmatrix}^T \qquad \begin{bmatrix} y1^2 \\ y2^2 \\ sqrt(2)*y1*y2 \end{bmatrix}$$

phi(x) phi(y)

phi(x).T*phi(y) = eqn.1

This is essentially a dot product - the Kernel here is a dot product (in between vectors) Otherwise AAT.^n.

Polynomial Kernel

$$\psi\left(\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}\right) = \begin{bmatrix} x_1 \\ x_2 \\ \sqrt{x_1^2 + x_2^2} \end{bmatrix}.$$

$$\phi : x \rightarrow \phi(x) = \begin{bmatrix} x_1^2 \\ \sqrt{2}x_1x_2 \\ x_2^2 \end{bmatrix}$$

$$\phi(x_m)^T\phi(x_n) = (x_{m1}x_{n1} + x_{m2}x_{n2})^2$$

$$\phi(x_m)^T\phi(x_n) = (\boldsymbol{x}_{\boldsymbol{m}}^T\boldsymbol{x}_{\boldsymbol{n}})^2$$

$$\phi(x_m)^T\phi(x_n) = k(x_m, x_n)$$

$$\phi(x_m)^T\phi(x_n) = \begin{bmatrix} x_{m1}^2 \\ \sqrt{2}x_{m1}x_{m2} \\ x_{m2}^2 \end{bmatrix}^T \begin{bmatrix} x_{n1}^2 \\ \sqrt{2}x_{n1}x_{n2} \\ x_{n2}^2 \end{bmatrix}$$

$$\phi(x_m)^T\phi(x_n) = x_{m1}^2x_{n1}^2 + 2x_{m1}x_{m2}x_{n1}x_{n2} + x_{m2}^2x_{n2}^2$$

This is essentially a dot product - the Kernel here is a dot product.

(AAT + I).^n = Polynomial Kernel

Expanded to larger n’s, this can be expanded as polynomial regression.

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_n \end{bmatrix} = \begin{bmatrix} 1 & x_1 & x_1^2 & \dots & x_1^m \\ 1 & x_2 & x_2^2 & \dots & x_2^m \\ 1 & x_3 & x_3^2 & \dots & x_3^m \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_n & x_n^2 & \dots & x_n^m \end{bmatrix} \begin{bmatrix} \beta_0 \\ \beta_1 \\ \beta_2 \\ \vdots \\ \beta_m \end{bmatrix} + \begin{bmatrix} \varepsilon_1 \\ \varepsilon_2 \\ \varepsilon_3 \\ \vdots \\ \varepsilon_n \end{bmatrix},$$

Polynomial Regression

Kernel Regression

Linear

$$y = \boldsymbol{w}^T \boldsymbol{x}$$

$$x, w \in \mathbb{R}^D$$

$$y \in \mathbb{R}$$

Polynomial

$$y = \boldsymbol{w}^T \phi(\boldsymbol{x})$$

$$\phi(x), w \in \mathbb{R}^M$$

$$y \in \mathbb{R}$$

$$\phi: \mathbb{R}^D \rightarrow \mathbb{R}^M$$

$$J(w) = \sum_{i=1}^{n} (y_n - w^T x_n)^2$$

$$w^* = \arg\min_w J(w)$$

$$w = (X^T X)^{-1} X^T Y$$

$$w = (\phi^T \phi)^{-1} \phi^T Y$$

$$w = (\phi^T \phi + \lambda I)^{-1} \phi^T Y$$

This is Ridge Regression

But this is hard to compute.

The advantage of a kernel is that you can compute it differently (explained in NNGP)

The Kernel Trick

$$J(w) = \sum_{i=1}^{n} (y_n - w^T\phi(x_n))^2 + \frac{\lambda}{2}\sum_{i=1}^{n} ||w||^2$$

$$\boldsymbol{w}^* = \frac{1}{\lambda}\sum_{i=1}^{n} (y_n - \boldsymbol{w}^T\phi(\boldsymbol{x}_n))\phi(\boldsymbol{x}_n)$$

$$\alpha_n = \frac{1}{\lambda}\sum_{i=1}^{n} (y_n - \boldsymbol{w}^T\phi(\boldsymbol{x}_n))$$

$$\boldsymbol{w}^* = \sum_{i=1}^{n} \alpha_n\phi(\boldsymbol{x}_n)$$

$$\boldsymbol{w}^* = \boldsymbol{\phi}^T\boldsymbol{\alpha}$$

$$J(w) = \sum_{i=1}^{n} (y_n - w^T\phi(x_n))^2 + \frac{\lambda}{2}\sum_{i=1}^{n} ||w||^2$$

$$J(w) = (y - \boldsymbol{\phi}w)^T(y - \boldsymbol{\phi}w) + \frac{\lambda}{2}w^Tw$$

$$J(w) = y^Ty - y^T\boldsymbol{\phi}w - w^T\boldsymbol{\phi}^Ty + w^T\boldsymbol{\phi}^T\boldsymbol{\phi}w + \frac{\lambda}{2}w^Tw$$

$$J(\alpha) = y^Ty - y^T\phi(\boldsymbol{\phi}^T\alpha) - (\boldsymbol{\phi}^T\alpha)^T\phi^Ty + (\boldsymbol{\phi}^T\alpha)^T\boldsymbol{\phi}^{\boldsymbol{T}}\boldsymbol{\phi}(\boldsymbol{\phi}^T\alpha) + \frac{\lambda}{2}(\boldsymbol{\phi}^T\alpha)^T(\boldsymbol{\phi}^T\alpha)$$

$$J(\alpha) = y^Ty - y^T\underline{\boldsymbol{\phi}\boldsymbol{\phi}^{\boldsymbol{T}}}\alpha - \alpha^T\underline{\boldsymbol{\phi}\boldsymbol{\phi}^T}y + \alpha^T\underline{\boldsymbol{\phi}\boldsymbol{\phi}^{\boldsymbol{T}}\boldsymbol{\phi}\boldsymbol{\phi}^T}\alpha + \frac{\lambda}{2}\alpha^T\underline{\boldsymbol{\phi}\boldsymbol{\phi}^T}\alpha$$

$$J(\alpha) = y^T y - y^T \boldsymbol{\phi\phi^T} \alpha - \alpha^T \boldsymbol{\phi\phi}^T y + \alpha^T \boldsymbol{\phi\phi^T\phi\phi}^T \alpha + \frac{\lambda}{2} \alpha^T \boldsymbol{\phi\phi}^T \alpha$$

$$\boldsymbol{\alpha}^* = (\boldsymbol{K} + \lambda' I)^{-1} y$$

$$J(\alpha) = y^T y - y^T \boldsymbol{K\alpha} - \boldsymbol{\alpha}^T \boldsymbol{K} y + \boldsymbol{\alpha}^T \boldsymbol{K^2 \alpha} + \frac{\lambda}{2} \boldsymbol{\alpha}^T \boldsymbol{K\alpha}$$

$$w = \phi^T \alpha = \phi^T (\boldsymbol{K} + \lambda' I)^{-1} y$$

$$J(\alpha) = y^T y - y^T \boldsymbol{K\alpha} - y^T \boldsymbol{K\alpha} + \boldsymbol{\alpha}^T \boldsymbol{K^2 \alpha} + \frac{\lambda}{2} \boldsymbol{\alpha}^T \boldsymbol{K\alpha} \qquad \because K = K^T$$

$$J(\alpha) = y^T y - 2 y^T \boldsymbol{K\alpha} + \boldsymbol{\alpha}^T \boldsymbol{K^2 \alpha} + \frac{\lambda}{2} \boldsymbol{\alpha}^T \boldsymbol{K\alpha}$$

Not Kernelized: $w = (\phi^T \phi + \lambda I)^{-1} \phi^T y$

Kernelized: $w = \phi^T (\boldsymbol{\phi\phi^T} + \lambda' I)^{-1} y$

$$\frac{\delta J(\alpha)}{\delta \alpha} = 0 - 2 y^T \boldsymbol{K} + 2 \boldsymbol{\alpha}^T \boldsymbol{K^2} + \lambda \boldsymbol{\alpha}^T \boldsymbol{K} = 0$$

$$-2 y^T + 2 \boldsymbol{\alpha}^T \boldsymbol{K} + \lambda \boldsymbol{\alpha}^T = 0$$

$$\boldsymbol{\alpha}^T \boldsymbol{K} + \frac{\lambda}{2} \boldsymbol{\alpha}^T = y^T$$

$$\boldsymbol{\alpha}^T = y^T \left(K + \frac{\lambda}{2} I \right)^{-1}$$

$$\boldsymbol{\alpha} = \left(\boldsymbol{K} + \frac{\lambda}{2} I \right)^{-1} y$$

Mercer’s Theorm

$$K(x, y) = \langle \phi(x), \phi(y) \rangle$$

The Kernel Matrix

$$\boldsymbol{\phi}\boldsymbol{\phi}^{\boldsymbol{T}} = K = \begin{bmatrix} \phi(x_1)^T\phi(x_1) & \phi(x_1)^T\phi(x_2) & . & \phi(x_1)^T\phi(x_n) \\ \phi(x_2)^T\phi(x_1) & \phi(x_2)^T\phi(x_2) & . & . \\ . & . & . & . \\ \phi(x_m)^T\phi(x_1) & \phi(x_m)^T\phi(x_2) & . & \phi(x_m)^T\phi(x_n) \end{bmatrix} = \begin{bmatrix} k(x_1, x_1) & k(x_1, x_2) & . & k(x_1, x_n) \\ k(x_2, x_1) & k(x_2, x_2) & . & . \\ . & . & . & . \\ k(x_m, x_1) & k(x_m, x_2) & . & k(x_m, x_n) \end{bmatrix}$$

Kernel Regression Formula Summary

$$w^* = \sum_{i=1}^{n} \alpha_i \psi(x^{(i)}) \quad ;$$

This is proved using Representer Theorem (theorem 1)

Proof. Following the result of Theorem 1, we need only show that the vector $\alpha = [\alpha_1, \alpha_2, \ldots, \alpha_n] \in \mathbb{R}^{1\times n}$ equals $y\hat{K}^\dagger$. In particular, writing Eq. 2 in matrix form, we find:

$$\mathcal{L}(\alpha) = \|y - \alpha\hat{K}\|_2$$

Now minimizing $\mathcal{L}(\alpha)$ is equivalent to solving the linear system $y = \alpha\hat{K}$, which consists of n equations with n variables.[2] In particular the solution is given by $\alpha = y\hat{K}^\dagger$ as was proved in Lecture 2 Theorem 1. Hence we conclude by Theorem 1:

$$w^* = \sum_{i=1}^{n} [y\hat{K}^\dagger]_i \psi(x^{(i)}) \quad ;$$

where $\hat{K}(X, x) \in \mathbb{R}^n$ *with entries* $\hat{K}(X, x)_i = K(x^{(i)}, x)$.

NNGP

Hence for any fixed finite width k, we can construct the kernel by computing the product hφ(Bx), φ(Bx˜)i. Interestingly, under certain conditions on the initialization, we can tractably compute the inner product even in the limit as k → ∞, and the resulting kernel is called the Neural Network Gaussian Process (NNGP).

The last layer of infinitely wide neural networks is given by the Neural Network Gaussian Process (NNGP)

Derivation of NNGP for one layer (last layer)

Neural Network Represented in terms of

$$\boxed{f(x) = A\phi}$$

$$(B \cdot x) \quad \mathbb{R}^{1\times k} \quad \mathbb{R}^{k\times d} \quad \mathbb{R}^{d}$$

If one layer in the NN is taken as a fixed random matrix and only the last layer is trained (A here); the network can be thought of as taking the first layer as applying a fixed feature map (kernel) to the data and then doing linear regression for the second layer (which is A here).

$$\mathcal{L}(A) = \frac{1}{2}\sum_{i=1}^{n}(y^{(i)} - f(x^{(i)}))^2 = \frac{1}{2}\sum_{i=1}^{n}(y^{(i)} - A\phi(Bx^{(i)}))^2$$

This is equivalent to using kernel regression with the kernel Σ(x, x˜) = hφ(Bx), φ (Bx˜)i to map from X to y

$$\psi(x) = \phi(Bx)$$

For a finite width k (k: neurons in layer of NN), kernel can be computed by computing the product.

$$\Sigma(x, \tilde{x}) = \langle \phi(Bx), \phi(B\tilde{x}) \rangle$$

As k → ∞ the kernel can still be computed by this dot product given some conditions: B ~ N(0,1) and this is called the NNGP.

$$\Sigma(x, \tilde{x}) = \lim_{k \to \infty} \left[\tfrac{1}{k} \langle \phi(Bx), \phi(B\tilde{x}) \rangle \right].$$

Simplifying the first equation we can get this where u = w T x and v = w T x˜ where u, v are now jointly Gaussian random variables.

$$\Sigma(x, \tilde{x}) = \mathbb{E}_{(u,v)\sim\mathcal{N}(\mathbf{0},\Lambda)}[\phi(u)\phi(v)]$$

$$\Lambda = \begin{bmatrix} \|x\|_2^2 & x^T\tilde{x} \\ x^T\tilde{x} & \|\tilde{x}\|_2^2 \end{bmatrix}.$$

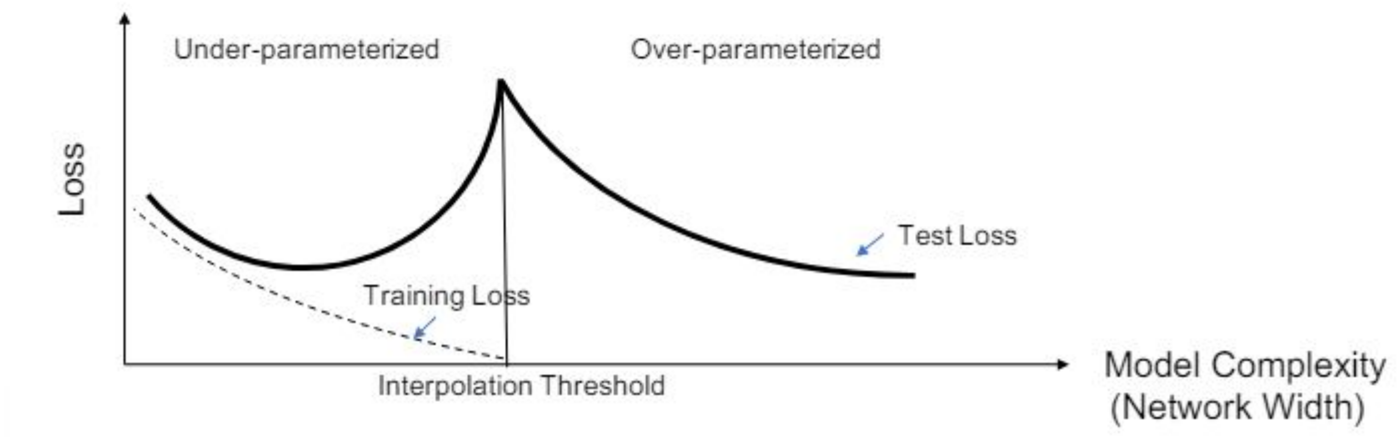
This can be computed as a kernel for different transformation functions including activation functions like RELU.

For example, the NNGP of the Random Fourier Feature Model (higher feature map transformation function) is recognizable as the Gaussian kernel.

$$= e^{-\frac{1}{2}\|x-\tilde{x}\|_2^2} \;;$$

Sometimes it's difficult to compute the formula for a given φ. Dual Activations when data lies on the unit sphere.

The benefit of over parameterization



Lim ->Infinite width; error is 0

NTK - different type of kernel

Solving Kernel Regression with the NNGP was for training the last layer of an infinitely wide 1 layer NN. Training all layers of a NN is equivalent to solving kernel regression with a kernel known as NTK.

NN is a map on parameters A,B and samples x. A NN can be thought of as a function of both data and parameters.

$$f(w\,;x) = A\phi(Bx)\ ;$$

$$w = \begin{bmatrix} A_{11} & A_{12} & \dots & A_{1k} & B_{11} & B_{12} & \dots & B_{kd} \end{bmatrix}^T \in \mathbb{R}^{k+kd}$$

We can take the linearization of a Neural Network around initial weights by taking the first order Taylor Series expansion

$$\tilde{f}_x(w) = f(w^{(0)}) + \nabla f_x(w^{(0)})^T (w - w^{(0)}).$$

This is equivalent to applying a linear model on top of transformed features same as how we proceeded before.

Now this linearization can be trained using the MSE loss which would make it a linear/kernel regression problem after applying the feature map. This can be solved efficiently using the Kernel Trick.

The Neural Tangent Kernel is defined as:

Definition 2 (Neural Tangent Kernel). *Let $f_x(w) : \mathbb{R}^p \to \mathbb{R}$ denote a neural network with initial parameters $w^{(0)}$. The* ***neural tangent kernel****, $K : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$, is given by:*

$$K(x, \tilde{x}) = \langle \nabla f_x(w^{(0)}), \nabla f_{\tilde{x}}(w^{(0)}) \rangle$$

$$\Theta^{\overset{\text{depth}}{(L)}}(\underbrace{x, y}_{\text{two samples}}) = \underbrace{\sum_{p=1}^{P}}_{\text{all parameters}} \frac{d}{d\theta_p} f_\theta(x) \, \frac{d}{d\theta_p} f_\theta(y)$$

For a finite width neural network

Example 1. *For $x \in \mathbb{R}$, let $f_x(w) : \mathbb{R}^4 \to \mathbb{R}$ denote a neural network parameterized as follows:*

$$f_x(w) = \begin{bmatrix} A_{11} & A_{12} \end{bmatrix} \phi\left(\begin{bmatrix} B_{11} \\ B_{21} \end{bmatrix} x \right) = A_{11}\phi(B_{11}x) + A_{12}\phi(B_{21}x) \ ;$$

where $w = \begin{bmatrix} A_{11} & A_{12} & B_{11} & B_{21} \end{bmatrix}^T$. *Then, for* $x, \tilde{x} \in \mathbb{R}$:

$$K(x,\tilde{x}) = \left\langle \begin{bmatrix} \phi(B_{11}x) \\ \phi(B_{21}x) \\ A_{11}x\phi'(B_{11}x) \\ A_{12}x\phi'(B_{21}x) \end{bmatrix}, \begin{bmatrix} \phi(B_{11}\tilde{x}) \\ \phi(B_{21}\tilde{x}) \\ A_{11}\tilde{x}\phi'(B_{11}\tilde{x}) \\ A_{12}\tilde{x}\phi'(B_{21}\tilde{x}) \end{bmatrix} \right\rangle = \sum_{i=1}^{2} \phi(B_{i1}x)\phi(B_{i1}\tilde{x}) + \sum_{i=1}^{2} A_{1i}^2 \phi'(B_{i1}x)\phi'(B_{i1}\tilde{x})x\tilde{x}$$

D wrt x

This is an example of a 4 layer network.

The Blue term accounts for training more than just the last layer.

$$w = w^{(0)} + \tilde{w},$$

$$\mathcal{L}(w) = \frac{1}{2}\sum_{i=1}^{n}(y^{(i)} - \tilde{f}_{x^{(i)}}(w))^2 \iff \mathcal{L}(\tilde{w}) = \frac{1}{2}\sum_{i=1}^{n}(y^{(i)} - f_{x^{(i)}}(w^{(0)}) - \nabla f_{x^{(i)}}(w^{(0)})^T\tilde{w})^2$$

it is easier to ignore the correction term and just set fx(i) (w (0)) = 0

Then we can solve using Kernel Regression.

$$\hat{f}(x) = y\hat{K}^{\dagger}\hat{K}(X, x) \; ;$$

So the difference between other kernels and this is perhaps that NTK finds a more accurate relationship after linearization and also that we do it for an infinite width case.

This is for a finite width network. For the case when k approaches infinity NTK is given by:

Theorem 1. *Let $A \in \mathbb{R}^{1\times k}, B \in \mathbb{R}^{k\times d}$ and $\phi : \mathbb{R} \to \mathbb{R}$ an element-wise activation function. For $x \in \mathcal{S}^{d-1}$, let $f_x(w) : \mathbb{R}^{kd+k} \to \mathbb{R}$ denote 1 hidden layer fully connected network with $f_x(w) = \frac{1}{\sqrt{k}} A\phi(Bx)$. If $A_{1i}, B_{ij} \overset{i.i.d}{\sim} \mathcal{N}(0,1)$, then as $k \to \infty$, the NTK is given by:*

$$K(x, \tilde{x}) = \check{\phi}(x^T \tilde{x}) + \check{\phi}'(x^T \tilde{x}) x^T \tilde{x} \ ;$$

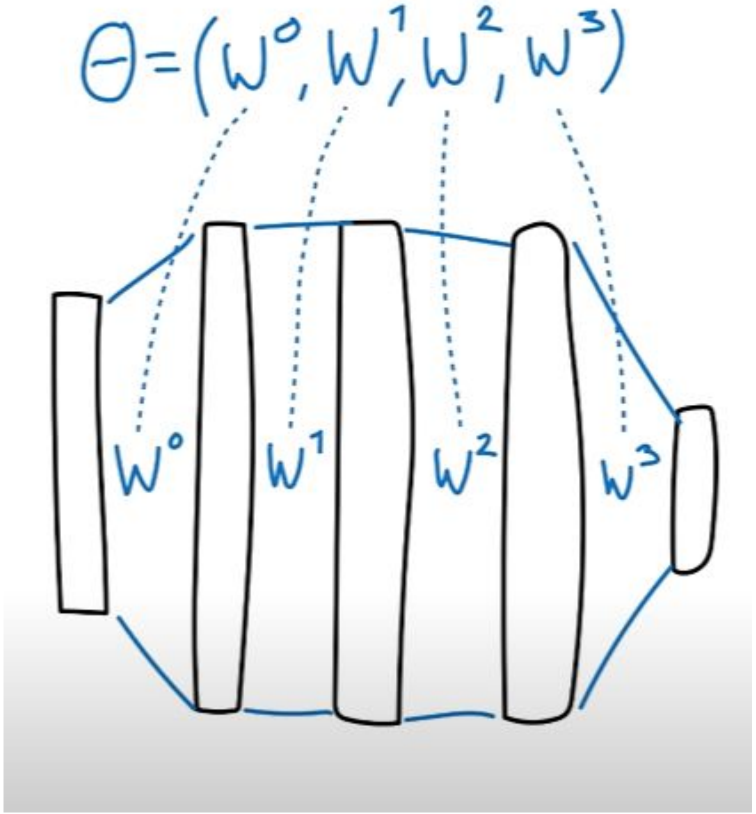
where $\check{\phi} : [-1, 1] \to \mathbb{R}$ is the dual activation for ϕ.

And then Kernel Regression as normal can be performed.

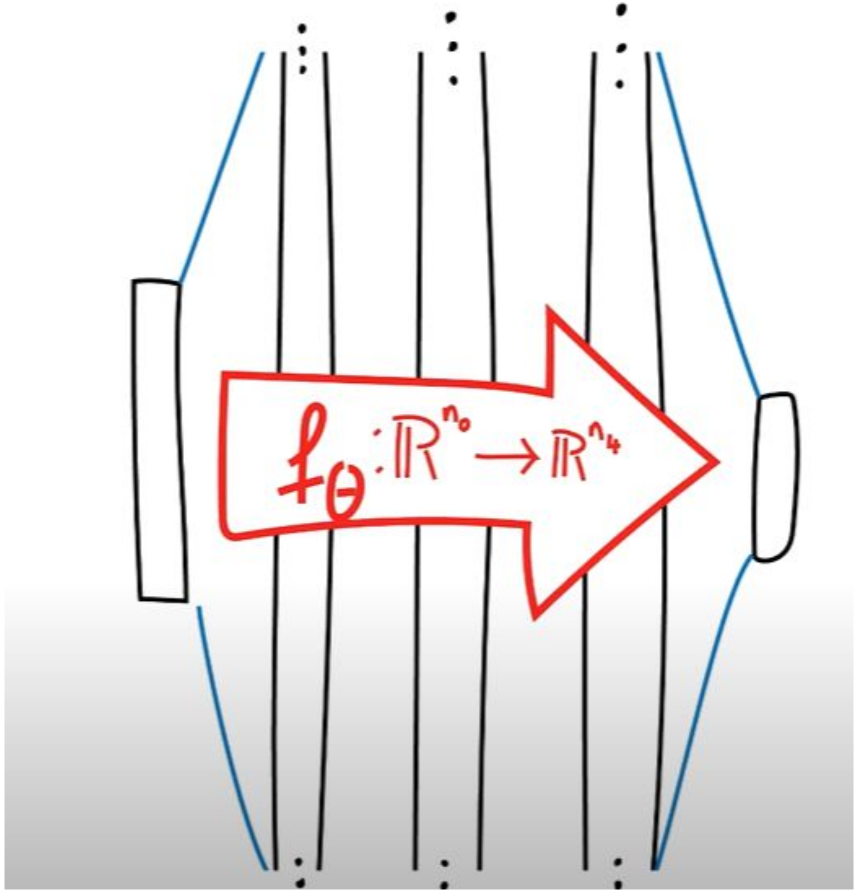
$$\hat{f}(x) = y\hat{K}^{\dagger}\hat{K}(X, x) \ ;$$

Data

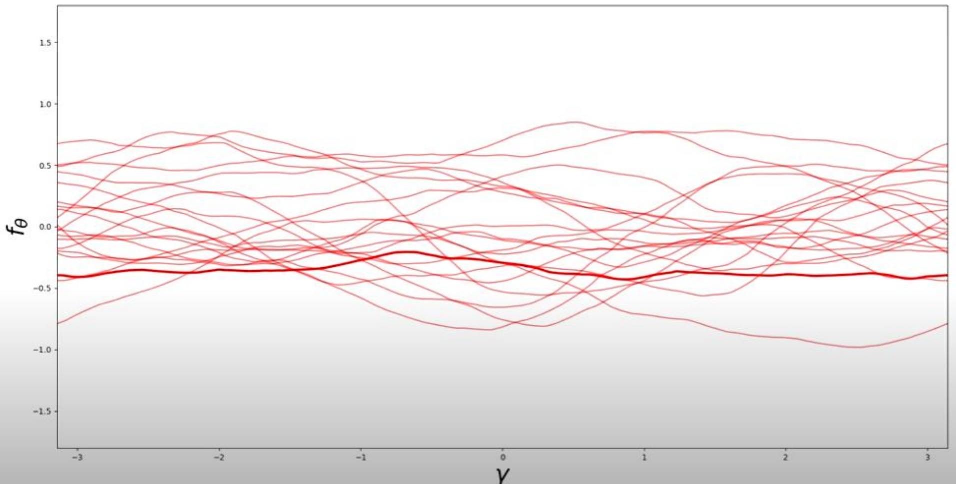
Data for NTK can be written as any number of parameters and a y output (can be multidimensional) and has to be numerical.

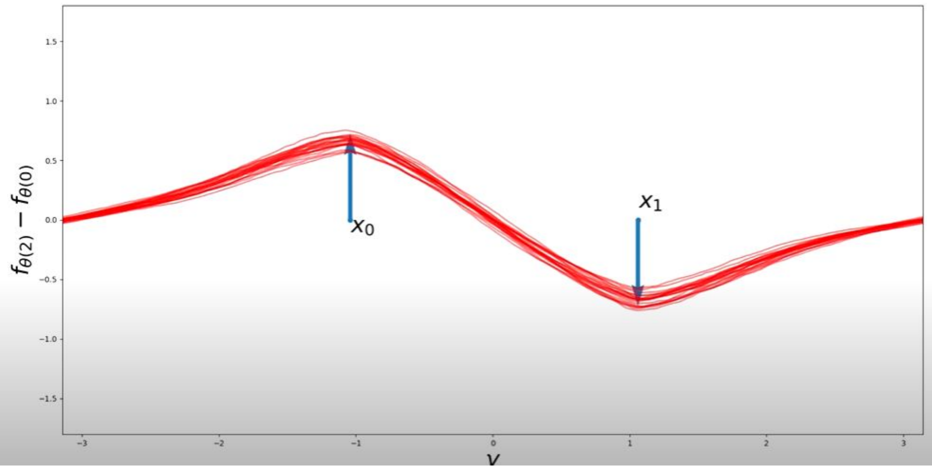


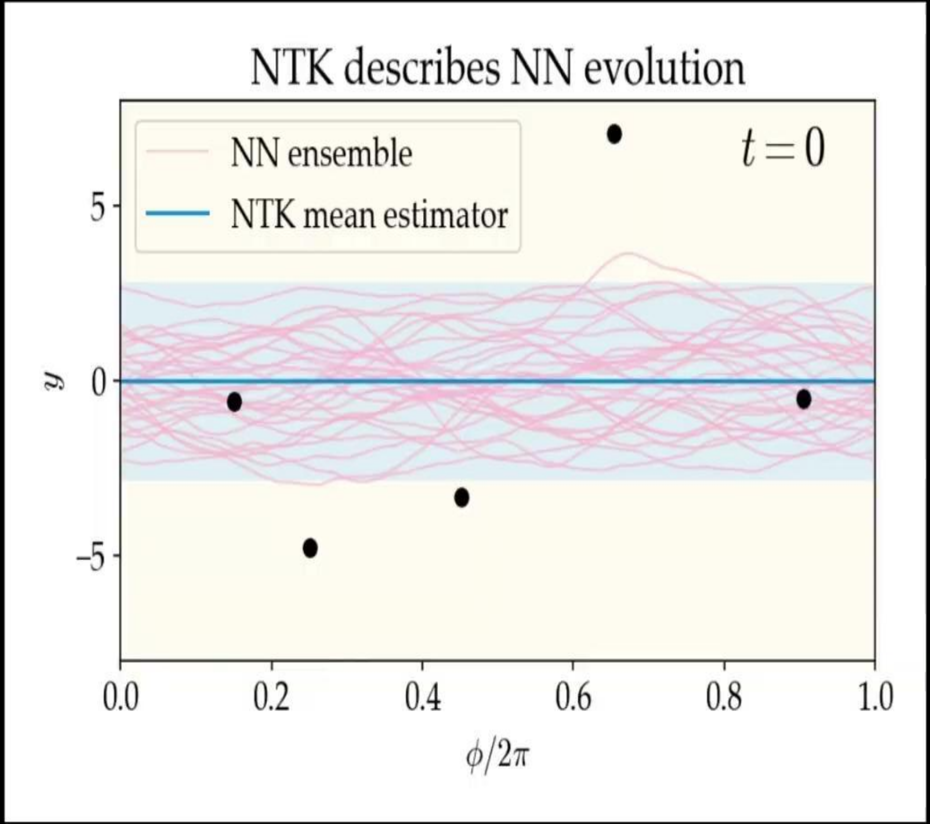
$\Theta = (W^0, W^1, W^2, W^3)$
W^0
W^1
W^2
W^3



$f_\Theta : \mathbb{R}^{n_0} \to \mathbb{R}^{n_4}$


f_θ
γ


$f_{\theta(2)} - f_{\theta(0)}$
x_0
x_1
γ



$$\Theta^{\overset{\text{depth}}{(L)}}\underbrace{(x,y)}_{\text{two samples}} = \underbrace{\sum_{p=1}^{P}}_{\text{all parameters}} \frac{d}{d\theta_p} f_\theta(x)\,\frac{d}{d\theta_p} f_\theta(y)$$

References

YouTube. (2018, November 12). *Neural tangent kernel: Convergence and generalization in Neural Networks*. YouTube. https://www.youtube.com/watch?v=raT2ECrvbag&pp=ygUXbmV1cmFsIHRhbmdlbnQga2VucmVrbCA%3D

(2022) 6.S088 modern machine learning: Simple methods that work. Available at: https://web.mit.edu/modernml/course/

The kernel trick in support Vector Machine (SVM) (2022) *YouTube*. Available at: https://www.youtube.com/watch?v=Q7vT0--5VII&pp=ygURdGhlIGtlcm5lbCB0cmljayA%3D

YouTube. (2018a, October 5). *The kernel trick - the math you should know!*. YouTube. https://www.youtube.com/watch?v=wBVSbVktLIY&pp=ygURdGhlIGtlcm5lbCB0cmljayA%3D