Gaussian Processes Kernels and Neural Tangent Kernels of Deep Neural Networks

Jiaxi Zhao (Stony Brook University)

8th January, 2021

Origin: Radford M. Neal's Ph.D. thesis

By defining classes of prior distributions for network parameters that reach sensible limits as the size of the network goes to infinity. In this limit, the properties of these priors can be elucidated. Some priors converge to Gaussian processes, in which functions computed by the network may be smooth, Brownian, or fractionally Brownian. Other priors converge to non-Gaussian stable processes.[1]

[1] Neal, Radford M. Bayesian learning for neural networks. Vol. 118. Springer Science & Business Media, 2012.

Multilayer neural network

We consider the following L-layer neural network with activation function given by $\phi(\cdot)$. The number of neurons in each layer is given by $N_1, N_2, \cdots, N_L$.

$$z_i^l(\mathbf{x}) = b_i^l + \sum_{j=1}^{N_l} W_{ij}^l y_j^l(\mathbf{x}), \qquad l = 1, 2, \cdots, L$$

$$y_j^l(\mathbf{x}) = \phi\left(z_i^{l-1}(\mathbf{x})\right), \qquad l = 2, \cdots, L,$$

and we further postulate that $\mathbf{y}^1(\mathbf{x}) = \mathbf{x}$. We call $\mathbf{z}^l$ the output of the l-th layer.

Multilayer neural network

The parameters are initialized according to the following prior

$$W_{ij}^{l} \overset{i.i.d.}{\sim} \mathcal{N}\left(0, \frac{\sigma_w^2}{N_l}\right), \quad i = 1, 2, \cdots, N_l, j = 1, 2, \cdots, N_{l-1},$$

$$b_i^{l} \overset{i.i.d.}{\sim} \mathcal{N}\left(0, \sigma_b^2\right), \quad i = 1, 2, \cdots, N_l.$$

Indeed, we can use many distribution with certain moment condition for weight parameter W, only the scale of variance is crucial.

First layer GP: linear model

$$m^1(\mathbf{x}) = \mathbb{E}\left[\mathbf{z}^1(\mathbf{x})\right] = \mathbb{E}\left[\mathbf{b}^1 + \mathbf{W}^1\mathbf{x}\right] = \mathbf{0},$$

$$\begin{aligned}
\left(k^1(\mathbf{x},\mathbf{x}')\right)_{ij} &= \left(\mathbb{E}\left[\mathbf{z}^1(\mathbf{x})\left(\mathbf{z}^1(\mathbf{x}')\right)^T\right]\right)_{ij} \\
&= \mathbb{E}\left[z_i^1(\mathbf{x})\, z_j^1(\mathbf{x}')\right] \\
&= \mathbb{E}\left[\left(b_i^1 + \sum_{k=1}^{N_1} W_{ik}^1 x_k\right)\left(b_j^1 + \sum_{k=1}^{N_1} W_{jk}^1 x_k'\right)\right] \\
&= \delta_{ij}\left(\sigma_b^2 + \frac{\sigma_w^2 \mathbf{x}\cdot\mathbf{x}'}{N_1}\right), \quad i,j = 1,2,\cdots,N_2.
\end{aligned}$$

First layer neural networks as GP

Since $W_{ik}^1 x_k, k = 1, 2, \cdots, N_1$ are mutually independent, by CLT, we conclude that $\mathbf{z}^1(\mathbf{x}) \sim \mathcal{GP}(\mathbf{0}, k^1)$. Each components are i.i.d.

How about the correlation?

$$z_i^1(\mathbf{x}) = b_i^1 + \sum_{k=1}^{N_1} W_{ik}^1 x_k, z_j^1(\mathbf{x}) = b_j^1 + \sum_{k=1}^{N_1} W_{jk}^1 x_k.$$

Since $b_i^1, b_j^1, W_{ik}^1, W_{jk}^1$ are all mutually independent, we conclude that z_i^1, z_j^1 are independent $\forall i, j \in N_2$. Another way to obtain the independency is by the equivalence of independent and un-correlated for Gaussian r.v.

From one layer to multi-layer

Now, we move to the second layer, which can be viewed as an one layer neural network whose input is given by the output of the first layer with elementwisely activation function, i.e.

$$\mathbf{z}^2 = \mathbf{b}^2 + \mathbf{W}^2\mathbf{y}^2.$$

The mean of the GP $\mathbf{z}^2$ is easy to calculate as

$$\mathbb{E}\left[\mathbf{z}^2\right] = \mathbb{E}\left[\mathbf{b}^2\right] + \mathbb{E}\left[\mathbf{W}^2\right]\mathbb{E}\left[\mathbf{y}^2\right] = 0.$$

We use the independency between $\mathbf{y}^2, \mathbf{W}^2$ in the first equality.

From one layer to multi-layer

$$\begin{aligned}\left(k^2\left(\mathbf{x},\mathbf{x}'\right)\right)_{ij} &= \mathbb{E}\left[\left(b_i^2+\sum_{k=1}^{N_2} W_{ik}^2 y_k^2\left(\mathbf{x}\right)\right)\left(b_j^2+\sum_{k=1}^{N_2} W_{jk}^2 y_k^2\left(\mathbf{x}'\right)\right)\right] \\ &= \delta_{ij}\left(\sigma_b^2+\sigma_w^2\mathbb{E}_{z\sim\mathcal{GP}(0,k^1)}\left[\phi\left(z\left(\mathbf{x}\right)\right)\phi\left(z\left(\mathbf{x}'\right)\right)\right]\right), \\ & \quad i,j=1,2,\cdots,N_2.\end{aligned}$$

We use the following

$$\mathbb{E}\left[W_{ik}^2 y_k^2\left(\mathbf{x}\right) W_{jk}^2 y_k^2\left(\mathbf{x}'\right)\right] = \mathbb{E}\left[W_{ik}^2\right]\mathbb{E}\left[W_{jk}^2\right]\mathbb{E}\left[y_k^2\left(\mathbf{x}\right) y_k^2\left(\mathbf{x}'\right)\right] = 0, i\neq j$$

$$\begin{aligned}\mathbb{E}\left[W_{ik}^2 y_k^2\left(\mathbf{x}\right) W_{ik}^2 y_k^2\left(\mathbf{x}'\right)\right] &= \mathbb{E}\left[\left(W_{ik}^2\right)^2\right]\mathbb{E}\left[y_k^2\left(\mathbf{x}\right) y_k^2\left(\mathbf{x}'\right)\right] \\ &= \frac{\sigma_w^2}{N_2}\mathbb{E}\left[\phi\left(z_k^1\left(\mathbf{x}\right)\right)\phi\left(z_k^1\left(\mathbf{x}'\right)\right)\right].\end{aligned}$$

From one layer to multi-layer

Moreover, the GP kernel is defined for each layer inductively[2].

$$
\begin{aligned}
m^l(\mathbf{x}) &= \mathbf{0}, \quad l = 1, 2, \cdots, L, \\
\left(k^1(\mathbf{x}, \mathbf{x}')\right)_{ij} &= \delta_{ij}\left(\sigma_b^2 + \sigma_w^2 \frac{\mathbf{x} \cdot \mathbf{x}'}{N_1}\right), \\
\left(k^l(\mathbf{x}, \mathbf{x}')\right)_{ij} &= \delta_{ij}\left(\sigma_b^2 + \right. \\
&\left. \sigma_w^2 \mathbb{E}_{z_i^{l-1} \sim \mathcal{GP}(0, k^{l-1})}\left[\phi\left(z_i^{l-1}(x)\right)\phi\left(z_i^{l-1}(x')\right)\right]\right), \\
l &= 2, 3, \cdots, L.
\end{aligned} \tag{1}
$$

By CLT, each z_k^2 is Gaussian, hence vanishing correlation indicates independency.

[2] Jaehoon Lee, Yasaman Bahri, Roman Novak, Samuel S Schoenholz, Jeffrey Pennington, and Jascha Sohl-Dickstein. Deep neural networks as gaussian processes. arXiv preprint arXiv:1711.00165, 2017.

GP for PINN?

What is the difference between PINN and ordinary NN? One answer is the linear operator acts on the function, e.g. derivative, Laplacian.

Problem

Is the derivative of neural network functions also a GP?

We restrict our attention to the scalar value functions, i.e. the output layer has dimension 1.

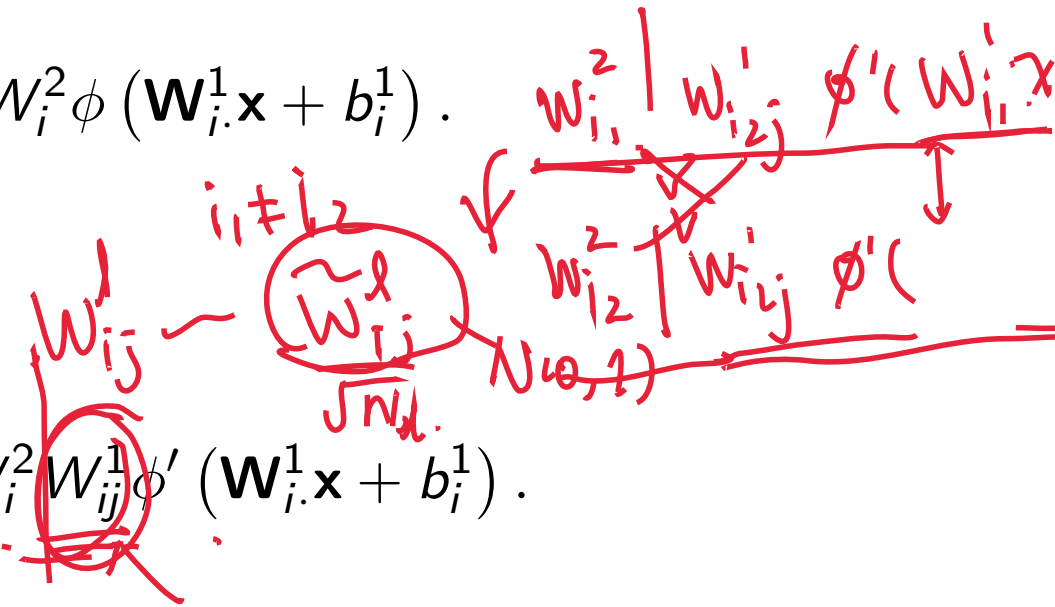
GP for PINN: Two layer NN

If the network has only two layers, i.e.

$$z^2(\mathbf{x}) = \sum_{i=1}^{N_2} W_i^2 \phi\left(\mathbf{W}_{i\cdot}^1 \mathbf{x} + b_i^1\right).$$

Taking derivative, we obtain

$$\frac{dz^2(\mathbf{x})}{dx_j} = \sum_{i=1}^{N_2} W_i^2 W_{ij}^1 \phi'\left(\mathbf{W}_{i\cdot}^1 \mathbf{x} + b_i^1\right).$$



Again, this is the sum of many independent r.v.s. It is still a GP[3].

[3] Wang, Sifan, Xinling Yu, and Paris Perdikaris. "When and why PINNs fail to train: A neural tangent kernel perspective." arXiv preprint arXiv:2007.14527 (2020).

GP for PINN: Two layer NN

$$\frac{dz^2(\mathbf{x})}{dx_j} = \sum_{i=1}^{N_2} W_i^2 W_{ij}^1 \phi'\left(\mathbf{W}_{i\cdot}^1 \mathbf{x} + b_i^1\right).$$

The mean and covariance is given by

$$\widetilde{m}^1(\mathbf{x}) = \mathbf{0},$$

$$\begin{aligned}
\widetilde{k}^1(\mathbf{x}, \mathbf{x}') &= \mathbb{E}\left[\sum_{i=1}^{N_2} W_i^2 W_{ij}^1 \phi'\left(\mathbf{W}_{i\cdot}^1 \mathbf{x} + b_i^1\right) \cdot \sum_{i=1}^{N_2} W_i^2 W_{ij}^1 \phi'\left(\mathbf{W}_{i\cdot}^1 \mathbf{x}' + b_i^1\right)\right] \\
&= \mathbb{E}\left[\sum_{i=1}^{N_2} \left(W_i^2\right)^2 \left(W_{ij}^1\right)^2 \phi'\left(\mathbf{W}_{i\cdot}^1 \mathbf{x} + b_i^1\right) \phi'\left(\mathbf{W}_{i\cdot}^1 \mathbf{x}' + b_i^1\right)\right] \\
&= \sigma_w^2 \mathbb{E}\left[\left(W_{ij}^1\right)^2 \phi'\left(\mathbf{W}_{i\cdot}^1 \mathbf{x} + b_i^1\right) \phi'\left(\mathbf{W}_{i\cdot}^1 \mathbf{x}' + b_i^1\right)\right].
\end{aligned}$$

GP for PINN: Three layer NN

If the network has three layers, i.e.

$$z^3(\mathbf{x}) = \sum_{i=1}^{N_3} W_i^3 \phi\left(\mathbf{W}_{i\cdot}^2 \phi\left(\mathbf{W}^1 \mathbf{x} + \mathbf{b}^1\right) + b_i^2\right).$$

Taking derivative, we obtain

$$\frac{dz^3(\mathbf{x})}{dx_j} = \sum_{i=1}^{N_3} W_i^3 \phi'\left(\mathbf{W}_{i\cdot}^2 \phi\left(\mathbf{W}^1 \mathbf{x} + \mathbf{b}^1\right) + b_i^2\right) \mathbf{W}_{i\cdot}^2 \left[\mathbf{W}_{\cdot j}^1 \odot \phi'\left(\mathbf{W}^1 \mathbf{x} + \mathbf{b}^1\right)\right],$$

where the last $\odot$ refers to the elemently product for vectors. Notice that in the above derivation, the summation can be restricted to the first row.

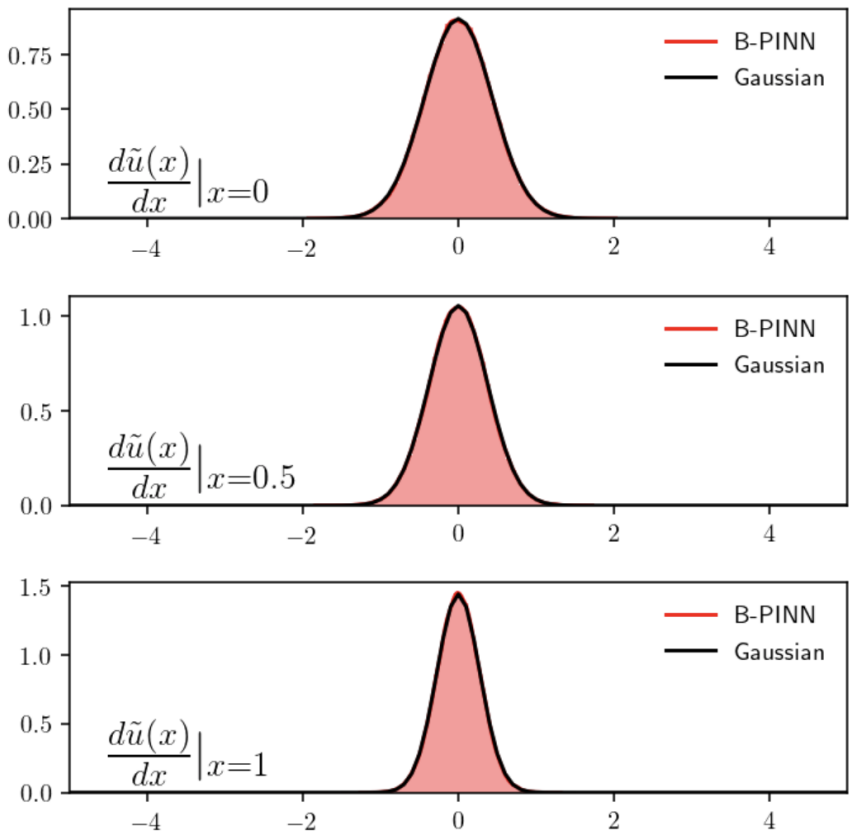
GP for PINN: Three layer NN

For different i, the terms $\phi'\left(\mathbf{W}^2_{i\cdot}\phi\left(\mathbf{W}^1\mathbf{x}+\mathbf{b}^1\right)+b^2_i\right)$, W^3_i are all mutually independent by the prior and inductive assumption on the previous layer.

The term $\mathbf{W}^2_{i\cdot}\left[\mathbf{W}^1_{\cdot j}\odot\phi'\left(\mathbf{W}^1\mathbf{x}+\mathbf{b}^1\right)\right]$ is exactly the gradient of the two layer NN for each i. Their independency is proved by non-correlation.

Hence their vector product is again a sum of independent small random variables, which should be Gaussian. We leave the calculation of kernel to further study.

GP for PINN: Three layer NN



500

Figure: GP of PINN. Three layers with each layer 50 neurons.

Neural tangent kernel: definition

Now, we move onto the optimization step, denote the loss function by $F : \theta \to \mathbb{R}$. The evolution of the function is given by

$$\frac{\partial \theta}{\partial t} = \frac{dF(\theta)}{d\theta}$$

$$\begin{aligned}
\left(\partial_t f(x)\right)_l &= \frac{\partial \left(f(x)\right)_l}{\partial \theta} \frac{\partial \theta}{\partial t} \\
&= \frac{\partial \left(f(x)\right)_l}{\partial \theta} \frac{dF(\theta)}{d\theta} \\
&= \frac{\partial \left(f(x)\right)_l}{\partial \theta} \otimes \frac{\partial \left(f(x)\right)_l}{\partial \theta} \frac{dF(f)}{df} \\
&= \frac{1}{N} \sum_{i=1}^{N} \sum_{l'=1}^{N_L} K_{l',l}(x_i, x) \left(\frac{dF(f)}{df}(x_i)\right)_{l'} \\
&= \frac{1}{N} K(x_D, x) \frac{dF(f)}{df}(x_D)
\end{aligned}$$

NTK

$K(x, x') = \sum_{\theta \text{ para}} \frac{\partial f}{\partial \theta}(x) \frac{\partial f}{\partial \theta}(x')$

Neural tangent kernel: convergence

We again prove the convergence of NTK by induction. We decompose the set of parameters to two sets: parameters in the outmost layer, parameters remain, i.e.

$$z^L(\mathbf{x}) = \sum_{i=1}^{N_L} W_i^L y_i^L(\mathbf{x}),$$

$$\frac{1}{N_L}\sum_{i=1}^{N_L} \frac{\partial z^L(\mathbf{x})}{\partial W_i^L}\frac{\partial z^L(\mathbf{x}')}{\partial W_i^L} = \frac{1}{N_L}\sum_{i=1}^{N_L} y_i^L(\mathbf{x})\, y_i^L(\mathbf{x}') = k^L(\mathbf{x},\mathbf{x}').$$

Neural tangent kernel: convergence

For the other parameters, we use chain rule, i.e.

$$\frac{1}{N_{L-1}} \sum_{i=1}^{N_{L-1}} \frac{\partial z^L(\mathbf{x})}{\partial W_{ji}^{L-1}} \frac{\partial z^L(\mathbf{x}')}{\partial W_{ji}^{L-1}}$$

$$= \frac{1}{N_{L-1}} \sum_{i=1}^{N_{L-1}} \frac{\partial z^{L-1}(\mathbf{x})}{\partial W_{ji}^{L-1}} \frac{\partial z^{L-1}(\mathbf{x}')}{\partial W_{ji}^{L-1}} \phi'\left(z^{L-1}(\mathbf{x})\right) \phi'\left(z^{L-1}(\mathbf{x}')\right) \left(W_j^L\right)^2$$

$$= \frac{1}{N_L} K^{L-1}(\mathbf{x}, \mathbf{x}') \, \mathbb{E}_{z \sim \mathcal{GP}\left(\mathbf{0}, k^{L-1}\right)} \left[\phi'\left(z^{L-1}(\mathbf{x})\right) \phi'\left(z^{L-1}(\mathbf{x}')\right)\right].$$

Neural tangent kernel: convergence

In the infinite-width limit, the NTK also converges to

Theorem 1. *For a network of depth L at initialization, with a Lipschitz nonlinearity σ, and in the limit as the layers width $n_1, \ldots, n_{L-1} \to \infty$, the NTK $\Theta^{(L)}$ converges in probability to a deterministic limiting kernel:*

$$\Theta^{(L)} \to \Theta_\infty^{(L)} \otimes Id_{n_L}.$$

The scalar kernel $\Theta_\infty^{(L)} : \mathbb{R}^{n_0} \times \mathbb{R}^{n_0} \to \mathbb{R}$ is defined recursively by

$$\Theta_\infty^{(1)}(x, x') = \Sigma^{(1)}(x, x')$$

$$\Theta_\infty^{(L+1)}(x, x') = \Theta_\infty^{(L)}(x, x')\dot{\Sigma}^{(L+1)}(x, x') + \Sigma^{(L+1)}(x, x'),$$

where

$$\dot{\Sigma}^{(L+1)}(x, x') = \mathbb{E}_{f \sim \mathcal{N}\left(0, \Sigma^{(L)}\right)}\left[\dot{\sigma}\left(f\left(x\right)\right)\dot{\sigma}\left(f\left(x'\right)\right)\right],$$

taking the expectation with respect to a centered Gaussian process f of covariance $\Sigma^{(L)}$, and where $\dot{\sigma}$ denotes the derivative of σ.

Figure: Theoretic result on NTK.

[3] Arthur Jacot, Franck Gabriel, and Clement Hongler. Neural tangent kernel: Convergence and general- ization in neural networks, 2018.

GP for PINN: Three layer NN

Theorem 2. *Assume that σ is a Lipschitz, twice differentiable nonlinearity function, with bounded second derivative. For any T such that the integral $\int_0^T \|d_t\|_{p^{in}} dt$ stays stochastically bounded, as $n_1, ..., n_{L-1} \to \infty$, we have, uniformly for $t \in [0, T]$,*

$$\Theta^{(L)}(t) \to \Theta_\infty^{(L)} \otimes Id_{n_L}.$$

As a consequence, in this limit, the dynamics of f_θ is described by the differential equation

$$\partial_t f_{\theta(t)} = \Phi_{\Theta_\infty^{(L)} \otimes Id_{n_L}} \left(\langle d_t, \cdot \rangle_{p^{in}} \right).$$

Figure: Theoretic result on NTK.

Proposition 2. *For a non-polynomial Lipschitz nonlinearity σ, for any input dimension n_0, the restriction of the limiting NTK $\Theta_\infty^{(L)}$ to the unit sphere $\mathbb{S}^{n_0-1} = \{x \in \mathbb{R}^{n_0} : x^T x = 1\}$ is positive-definite if $L \geq 2$.*

Figure: Positivity of NTK.

[3] Arthur Jacot, Franck Gabriel, and Clement Hongler. Neural tangent kernel: Convergence and general- ization in neural networks, 2018.

Comparison of NNGP and NT kernel

In linear model where one does not apply non-linearity to the input, it is easy to conclude that two kernels coincide. While in general, they differ from each other in the sense that NNGP kernel describes the behavior of neural network under Bayesian computation, i.e. calculate the Bayesian posterior and NTK describes that under the setting of gradient descent dynamics. Moreover, computing the Bayesian posterior is equivalent to gradient descent with all but the last-layer weights frozen.

Infinite-width neural network as GP: inference

Viewing infinite-width neural network as GP, we can do GP regression using this GP kernel. Given data x_D, y_D, we want to estimate a function s.t. $f(x_D) = y_D$ and calculate its value at some points x.

Noisy case:

$$\mu_b(x) = k(x, x_D)(k(x_D, x_D) + \sigma^2 I)^{-1} y_D$$
$$k_b(x, x') = k(x, x') - k(x, x_D)(k(x_D, x_D) + \sigma^2 I)^{-1} k(x_D, x')$$

Unnoisy case:

$$\mu_b(x) = k(x, x_D) k(x_D, x_D)^{-1} y_D$$
$$k_b(x, x') = k(x, x') - k(x, x_D) k(x_D, x_D)^{-1} k(x_D, x')$$

Infinite-width neural network training as kernel gradient flow

$$\partial_t f(x) \frac{1}{N} K(x_D, x) \frac{dF(f)}{df}(x_D), \partial_t f(x_D) = \frac{1}{N} K(x_D, x_D) d|_{f_t}(x_D),$$

$$\partial_t f(x) = K(x, x_D) K^{-1}(x_D, x_D) \partial_t f(x_D)$$

Consequently, we have the following characteristic for training trajectory.

$$\begin{aligned} f_\infty(x) &= f_0(x) + \int_0^\infty \partial_t f(x) dt \\ &= f_0(x) + K(x, x_D) K^{-1}(x_D, x_D) \int_0^\infty \partial_t f(x_D) dt \\ &= f_0(x) + K(x, x_D) K^{-1}(x_D, x_D)(f_\infty(x_D) - f_0(x_D)) \\ &= K(x, x_D) K^{-1}(x_D, x_D) f_\infty(x_D) + \\ &\quad [f_0(x) - K(x, x_D) K^{-1}(x_D, x_D) f_0(x_D)] \end{aligned} \tag{2}$$

Neural network training as kernel gradient flow

We can further calculate the mean and variance for the neural tangent kernel.

$$\begin{aligned}
\mu_e(x) &= K(x, x_D)K(x_D, x_D)^{-1} y_D \\
K_e(x, x') &= k(x, x') - K(x, x_D)K(x_D, x_D)^{-1} k(x_D, x') \\
&\quad - K(x', x_D)K(x_D, x_D)^{-1} k(x_D, x) \\
&\quad + K(x, x_D)K(x_D, x_D)^{-1} k(x_D, x_D) K(x_D, x_D)^{-1} K(x_D, x'),
\end{aligned}$$

This coincides with the Bayesian posterior for NNGP kernel if we have $k = K$. NNGP regression can be viewed as training only the last layer.

Numerical experiment of GPK and NTK: initialization

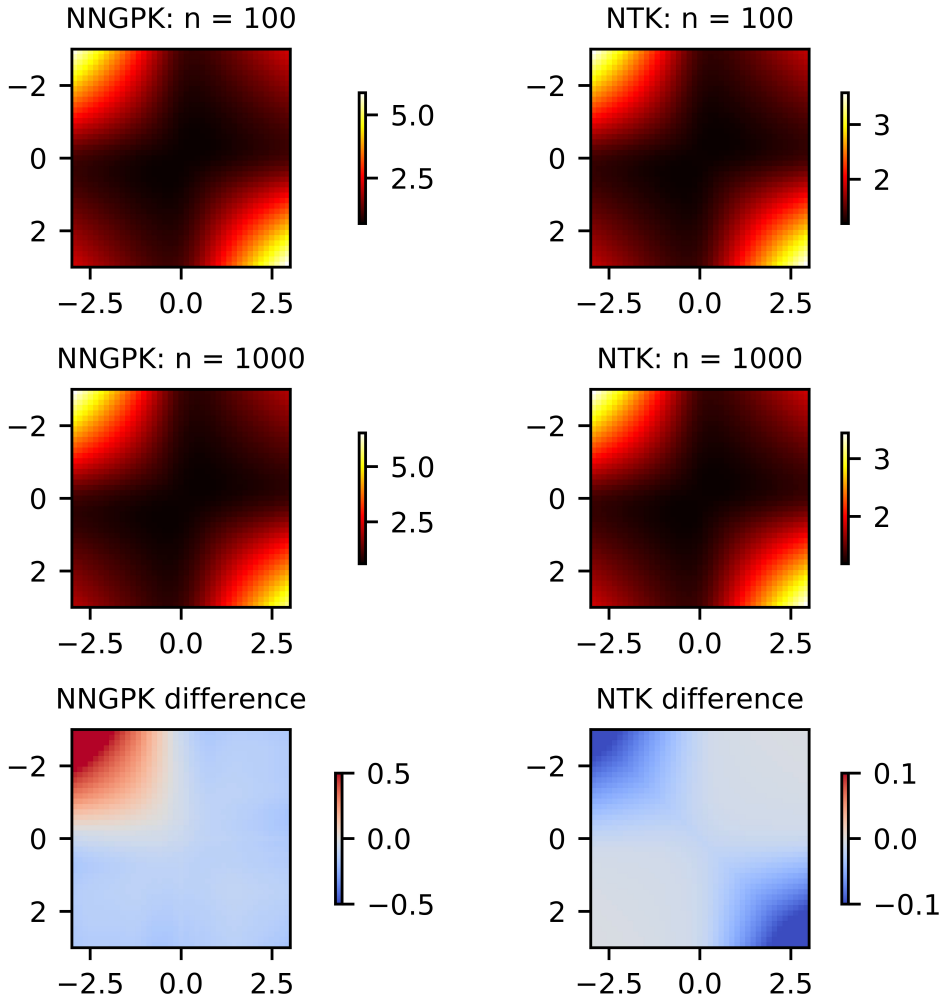


Figure: First row, 2000 samples of a network with $L = 2, N_1 = N_2 = 100$ and a sigmoid linearity. Second row, 200 samples with $N_1 = N_2 = 1000$.

Numerical experiment of GPK and NTK: Over-parametrized training

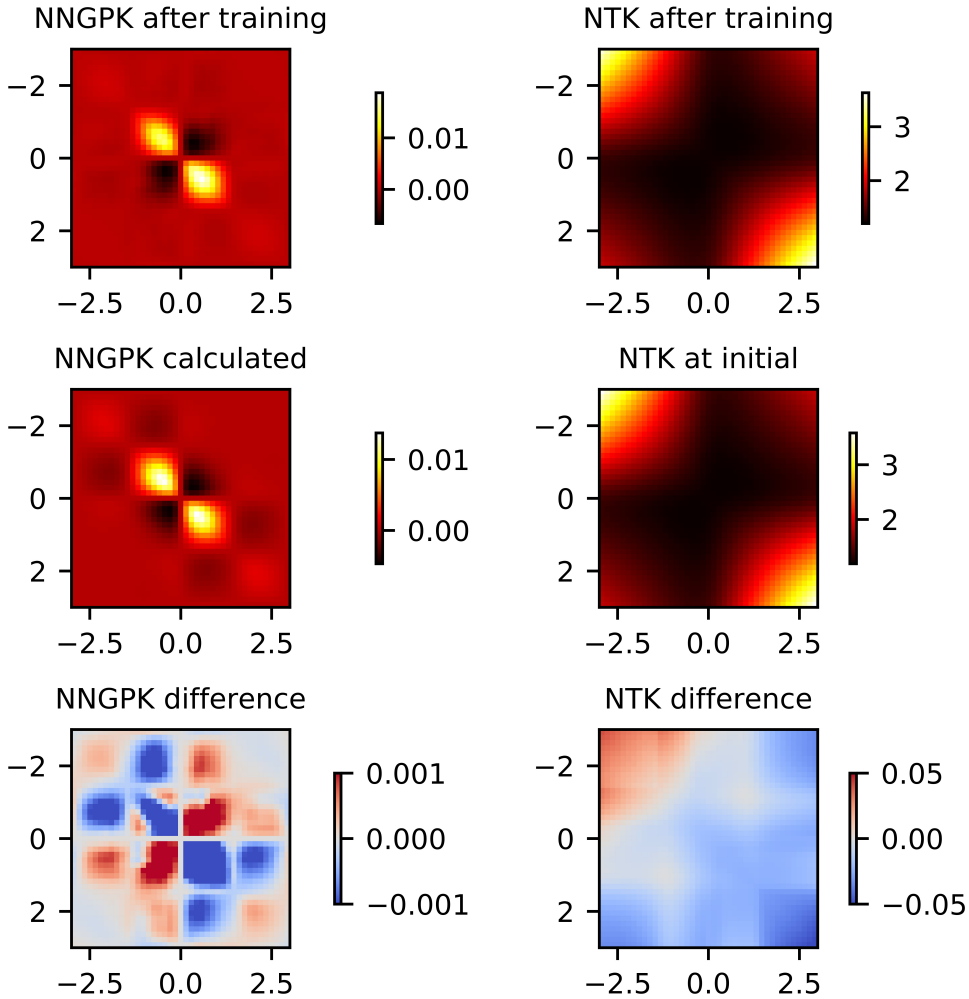


Figure: 2000 samples of a network with $L = 2, N_1 = N_2 = 30$ and a sigmoid linearity, training for 200 epochs on 5 data points.

Numerical experiment of GPK and NTK: Not over-parametrized training

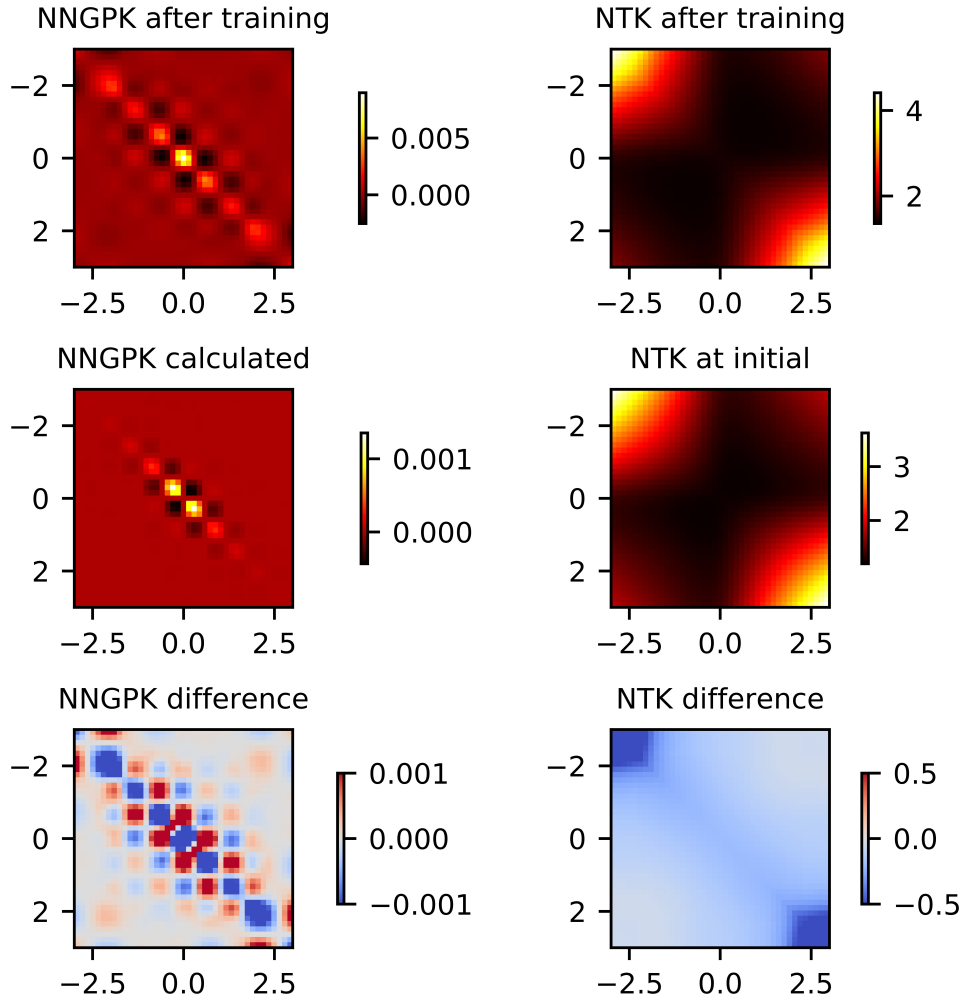


Figure: 2000 samples of a network with $L = 2, N_1 = N_2 = 30$ and a sigmoid linearity, training for 200 epochs on 10 data points.

Comparison between NNGP regression and neural network training

How does this NNGP regression relates to the network training using gradient descent, here is some experiments.

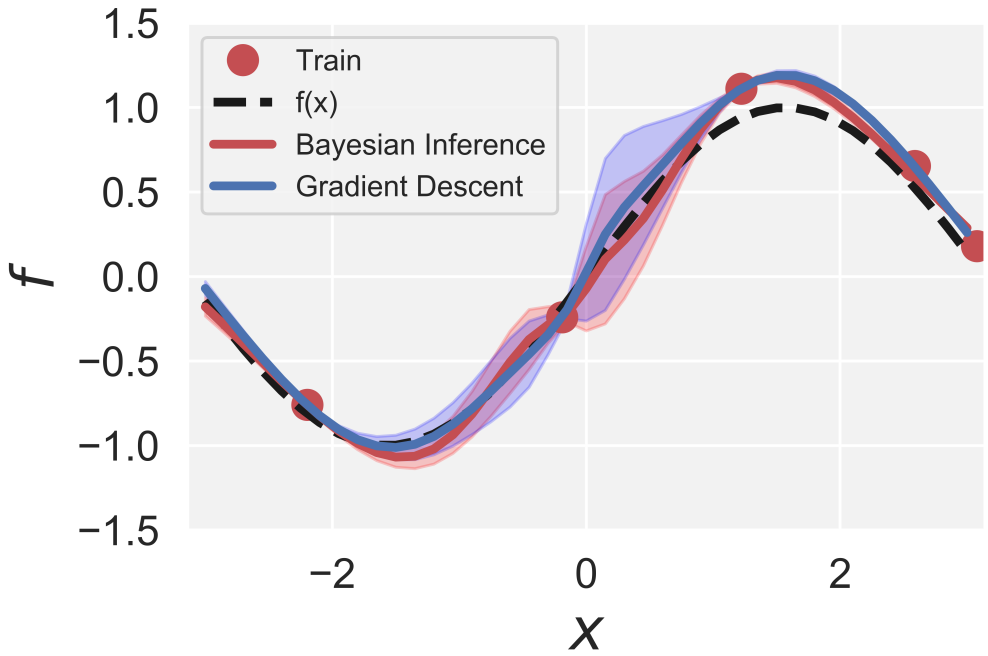


Figure: A comparison between NNGP and NTK.

NNGP for NAS

What are the advantages of NNGP regression comparing to training of NN?

- NNGP regression is more efficient, especially when the number of samples is small.
- The result of NNGP regression is correlated to network's actual performance after training, as illustrated in the infinite width.

Thanks for listening.