Recent development of Gaussian Process Regression

Qi Wang

January, 2021

1 Deep Neural Networks as Gaussian Processes

1. Neal [1996] first proved that one-layer neural network is equivalent to Gaussian Processes,
2. and then Lee et al. [2018] proved multi-layer NN is also Gaussian Processes and comes up with the model NNGP referring to a GP which corresponds to a deep, infinitely wide neural network.
3. Then Pang et al. [2019] extends this NNGP model to a more general form and use it to solve partial differential equations (PDEs).

(proof and evaluation refer to papers)

2 Combine GPs with NN

Friedrich [2020] mapped the equations for the GP's predictive mean and variance onto a specific NN of finite size and train it using standard deep learning techniques.

3 Large scale of data

1. **Exact GPs** Wang et al. [2019] utilizes multi-GPU parallelization and conjugate gradients method to implement GP with $10^4 - 10^6$ data points (normal GPR is suitable for thousands of data points).
2. **A review paper** Liu et al. [2020] reviews recent advances for improving the scalability and capability of scalable GPs, e.g., prior approximation (DTC, FITC, etc.) and posterior approximations. He also summarized future research directions:
    - Scalable deep GP
    - Scalable multi-task GP
    - Scalable online GP
    - Scalable GP classification

4 Rates of Convergence for Sparse Variational Gaussian Process Regression

Burt et al. [2019] wins the best paper award at ICML 2019.

5 Deep Neural Networks as Gaussian Processes

5.1 Derivations of NNGP kernel

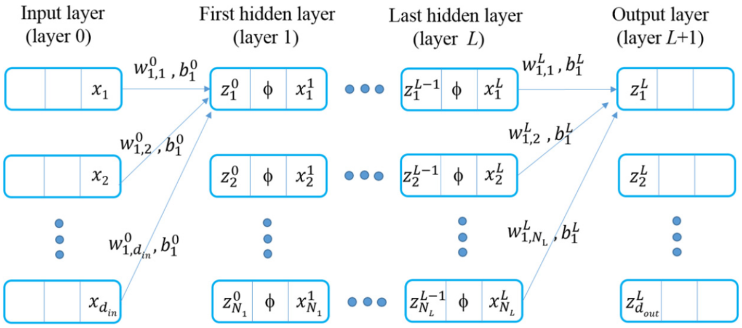

Fig. 1. A fully connected neural net with L hidden layers. For each unit or neuron in hidden layers, there exist one input z_i^{l-1} and one output x_i^l with $l = 1, 1\cdots, L$. Layer 0 is the input layer. x_i is the i-th component of the input vector $\boldsymbol{x}$, and z_i^L is the i-th component of the output vector $\boldsymbol{z}$. The dimensions of input and output spaces are d_{in} and d_{out}, respectively. The width for hidden layer l is N_l. At the center of each unit is the activation function $\phi(\cdot)$ that transforms input to the corresponding output. Between two successive layers, the weight w_{ij}^l for $l = 0, 1, \cdots, L$ denotes the contribution of unit j in layer l to unit i in layer $l+1$. Layer $L+1$ is the output layer. The bias b_i^l is attached to unit i in layer $l+1$ for $l = 0, 1, \cdots, L$. Note that for clarity most of connecting arrows between layers are omitted.

Figure 1: DNN figure from Pang et al. [2019]

Assume parameters w_{ij}^l are i.i.d.s for all i, j, l; AND b_i^l are i.i.d.s for all i, l, and

$$\begin{aligned}
\mathbb{E}[w_{ij}^l] = 0, &\quad \mathbb{E}[b_i^l] = 0, \\
Var[w_{ij}^l] = \frac{\sigma_w^2}{N_l}, &\quad Var[b_i^l] = \sigma_b^2, \\
\text{thus } \mathbb{E}[(w_{ij}^l)^2] = \frac{\sigma_w^2}{N_l}, &\quad \mathbb{E}[(b_i^l)^2] = \sigma_b^2.
\end{aligned}$$

As shown in Figure 1, we have

$$\begin{aligned}
z_j^0 &= b_j^0 + \sum_{i=1}^{\mathrm{d_{in}}} w_{ij}^0 x_i, & x_j^1 &= \phi(z_j^0), \\
z_j^{L-1} &= b_j^{L-1} + \sum_{i=1}^{N_{L-1}} w_{ij}^{L-1} x_i^{L-1}, & x_j^L &= \phi(z_j^{L-1}), \\
z_j^L &= b_j^L + \sum_{i=1}^{N_L} w_{ij}^L x_i^L.
\end{aligned}$$

Under such neural network transformation, for node j of layer $L+1$, we have the expectation and covariance of data points $\mathbf{x}, \mathbf{x}'$ are

$$\mathbb{E}[z_j^L(\mathbf{x})] = 0 \tag{1}$$

$$\begin{aligned}
k^L(\mathbf{x}, \mathbf{x}') &= \mathbb{E}[z_j^L(\mathbf{x}) z_j^L(\mathbf{x}')] = \mathbb{E}[(b_j^L + \sum_{i=1}^{N_L} w_{ij}^L x_i^L(\mathbf{x}))(b_j^L + \sum_{i=1}^{N_L} w_{ij}^L x_i^L(\mathbf{x}'))] \\
&= \mathbb{E}[(b_j^L)^2 + b_j^L(\sum_{i=1}^{N_L} w_{ij}^L x_i^L(\mathbf{x}) + \sum_{i=1}^{N_L} w_{ij}^L x_i^L(\mathbf{x}')) + \sum_{i=1}^{N_L} w_{ij}^L x_i^L(\mathbf{x}) \sum_{i=1}^{N_L} w_{ij}^L x_i^L(\mathbf{x}')] \\
&= \sigma_b^2 + 0 + \mathbb{E}[\sum_{i=1}^{N_L} \sum_{k=1}^{N_L} w_{ij}^L x_i^L(\mathbf{x}) w_{kj}^L x_k^L(\mathbf{x}')] \\
&= \sigma_b^2 + \sum_{i=1}^{N_L} \mathbb{E}[(w_{ij}^L)^2] \mathbb{E}[x_i^L(\mathbf{x}) x_i^L(\mathbf{x}')] \\
&= \sigma_b^2 + \frac{\sigma_w^2}{N_L} \sum_{i=1}^{N_L} \mathbb{E}[x_i^L(\mathbf{x}) x_i^L(\mathbf{x}')] \\
&= \sigma_b^2 + \sigma_w^2 \mathbb{E}[x_i^L(\mathbf{x}) x_i^L(\mathbf{x}')] \\
&= \sigma_b^2 + \sigma_w^2 \mathbb{E}[\phi(z_i^{L-1}(\mathbf{x})) \phi(z_i^{L-1}(\mathbf{x}'))] \\
&= \sigma_b^2 + \sigma_w^2 \int\int \phi(z_i^{L-1}(\mathbf{x})) \phi(z_i^{L-1}(\mathbf{x}')) p(z_i^{L-1}(\mathbf{x}), z_i^{L-1}(\mathbf{x}')) d(z_i^{L-1}(\mathbf{x})) d(z_i^{L-1}(\mathbf{x}'))
\end{aligned} \tag{2}$$

As we know $(z_i^{L-1}(\mathbf{x}), z_i^{L-1}(\mathbf{x}'))$ is a Gaussian Processes with distribution of $\mathcal{GP}(\mathbf{0}, \mathbf{\Sigma})$, where

$$\mathbf{\Sigma} = \begin{pmatrix} k^{L-1}(\mathbf{x}, \mathbf{x}) & k^{L-1}(\mathbf{x}, \mathbf{x}') \\ k^{L-1}(\mathbf{x}', \mathbf{x}) & k^{L-1}(\mathbf{x}', \mathbf{x}') \end{pmatrix} = \begin{pmatrix} k^{L-1}(\mathbf{x}, \mathbf{x}) & k^{L-1}(\mathbf{x}, \mathbf{x}') \\ k^{L-1}(\mathbf{x}, \mathbf{x}') & k^{L-1}(\mathbf{x}', \mathbf{x}') \end{pmatrix}.$$

Thus the bivariate pdf is

$$p(z_i^{L-1}(\mathbf{x}), z_i^{L-1}(\mathbf{x}')) = (2\pi)^{-1} \det(\mathbf{\Sigma})^{-\frac{1}{2}} \exp\left(-\frac{1}{2} \begin{pmatrix} z_i^{L-1}(\mathbf{x}) \\ z_i^{L-1}(\mathbf{x}') \end{pmatrix}^T \mathbf{\Sigma}^{-1} \begin{pmatrix} z_i^{L-1}(\mathbf{x}) \\ z_i^{L-1}(\mathbf{x}') \end{pmatrix}\right) \tag{3}$$

Plug 3 into 2 and calculate the integral, we can write 2 as a recursive function

$$k^L(\mathbf{x}, \mathbf{x}') = \sigma_b^2 + \sigma_w^2 F_\phi\left(k^{L-1}(\mathbf{x}, \mathbf{x}), k^{L-1}(\mathbf{x}, \mathbf{x}'), k^{L-1}(\mathbf{x}', \mathbf{x}')\right) \tag{4}$$

For input layer $L = 0$, we have

$$
\begin{aligned}
k^0(\mathbf{x}, \mathbf{x}') = \mathbb{E}[z_j^0(\mathbf{x})z_j^0(\mathbf{x}')] &= \mathbb{E}[(b_j^0 + \sum_{i=1}^{d_{in}} w_{ij}^0 x_i(\mathbf{x}))(b_j^0 + \sum_{i=1}^{d_{in}} w_{ij}^0 x_i(\mathbf{x}'))] \\
&= \mathbb{E}[(b_j^0)^2 + b_j^0(\sum_{i=1}^{d_{in}} w_{ij}^0 x_i(\mathbf{x}) + \sum_{i=1}^{d_{in}} w_{ij}^0 x_i(\mathbf{x}')) + \sum_{i=1}^{d_{in}} w_{ij}^0 x_i(\mathbf{x}) \sum_{i=1}^{d_{in}} w_{ij}^0 x_i(\mathbf{x}')] \\
&= \sigma_b^2 + \sum_{i=1}^{d_{in}} \mathbb{E}[(w_{ij}^0)^2] x_i(\mathbf{x}) x_i(\mathbf{x}') \\
&= \sigma_b^2 + \frac{\sigma_w^2}{d_{in}} \mathbf{x}^T \mathbf{x}'.
\end{aligned}
$$

Thus we can calculate $k^L(\mathbf{x}, \mathbf{x}')$ recursively from layer 0 to layer L.

When ϕ is **Relu**, Lee et al. [2018] utilized the result of Cho and Saul [2009], writing the analytical form of F_ϕ, where

$$
\begin{aligned}
&F_\phi\left(k^{L-1}(\mathbf{x}, \mathbf{x}), k^{L-1}(\mathbf{x}, \mathbf{x}'), k^{L-1}(\mathbf{x}', \mathbf{x}')\right) \\
=&\frac{1}{2\pi}\sqrt{k^{L-1}(\mathbf{x}, \mathbf{x}) k^{L-1}(\mathbf{x}', \mathbf{x}')}\left(\sin\theta_{\mathbf{x},\mathbf{x}'}^{L-1} + (\pi - \theta_{\mathbf{x},\mathbf{x}'}^{L-1})\cos\theta_{\mathbf{x},\mathbf{x}'}^{L-1}\right), \\
\text{where } \theta_{\mathbf{x},\mathbf{x}'}^{L-1} =&\cos^{-1}\left(\frac{k^{L-1}(\mathbf{x}, \mathbf{x}')}{\sqrt{k^{L-1}(\mathbf{x}, \mathbf{x}) k^{L-1}(\mathbf{x}', \mathbf{x}')}}\right).
\end{aligned}
$$

For other nonlinearity activation function ϕ, Lee et al. [2018] proposed a method to approximate the double integration in equation 2, i.e, a approximation of function $F_\phi(.)$, (by populating a table $F_\phi(i, j)$ in advance and calculate $F_\phi(.)$ by bilinear interpolation). However, the result is very bad when I test paper's open code (`https://github.com/brain-research/nngp`) using my dataset to do a regression task.

5.2 A new infinite width neural network open library

Lee and Novak et al. [2019] continued working on infinite wide neural networks and are developing an open library for it (`https://github.com/google/neural-tangents`) with a documentation paper Novak et al. [2019]. It provides efficient APIs to apply NNGP kernel and NTK kernel. NTK kernel represents the infinite-width neural networks trained by gradient descent. Several papers have shown that randomly initialized neural networks trained with gradient descent are characterized by a distribution that is related to the NNGP, and is described by the so-called Neural Tangent Kernel (NTK).

I also tested my dataset using this library, the result is shown as in Table 1 I am not sure about the architecture of this NNGP in this library (how to deal with the intractable integration.)

5.3 Gradient descent trained infinite-width neural network (NTK)

Lee et al. [2019] proved that gradient-based training of wide neural networks with a squared loss produces test set predictions drawn from a Gaussian process with a particular compositional kernel.

Table 1: Test result

nonlinearity	depth	noise_var	nngp			ntk		
			rmse_train	rmse_test	model_time(s)	rmse_train	rmse_test	model_time(s)
relu	1	0	nan	nan	8	nan	nan	12
relu	2	0	nan	nan	12	nan	nan	16
relu	5	0	nan	nan	25	nan	nan	29
relu	10	0	nan	nan	48	nan	nan	53
tanh	1	0	nan	nan	6	nan	nan	8
tanh	2	0	nan	nan	8	nan	nan	13
tanh	5	0	nan	nan	17	nan	nan	28
tanh	10	0	nan	nan	31	nan	nan	53
relu	1	0.001	4.3699527	4.4619255	8	3.2955	4.0827436	9
relu	2	0.001	4.3921933	4.45558	13	nan	nan	14
relu	5	0.001	4.613788	4.538349	27	nan	nan	29
relu	10	0.001	nan	nan	49	nan	nan	53
tanh	1	0.001	3.5829792	4.1464314	7	nan	nan	8
tanh	2	0.001	3.5964766	4.18663	9	nan	nan	13
tanh	5	0.001	3.6501112	4.2361836	18	nan	nan	28
tanh	10	0.001	3.7202923	4.2845206	35	nan	nan	53
relu	1	0.005	4.441606	4.544863	9	3.785048	4.134365	9
relu	2	0.005	4.3993587	4.505067	13	nan	nan	14
relu	5	0.005	4.3559175	4.4584227	27	nan	nan	29
relu	10	0.005	4.3458934	4.4383006	47	nan	nan	53
tanh	1	0.005	3.7566981	4.1049905	7	nan	nan	8
tanh	2	0.005	3.7273495	4.1020055	9	nan	nan	13
tanh	5	0.005	3.70035	4.100665	18	nan	nan	30
tanh	10	0.005	3.6812818	4.0999956	32	nan	nan	53
relu	1	0.01	4.5197663	4.6207676	8	3.91866	4.1767473	9
relu	2	0.01	4.4816885	4.5841255	13	nan	nan	14
relu	5	0.01	4.4295106	4.531488	25	nan	nan	28
relu	10	0.01	4.401551	4.5011983	46	nan	nan	54
tanh	1	0.01	3.8502622	4.124656	6	nan	nan	8
tanh	2	0.01	3.823687	4.117968	9	nan	nan	13
tanh	5	0.01	3.7982786	4.1122155	18	nan	nan	27
tanh	10	0.01	3.7807302	4.109836	32	nan	nan	52
relu	1	0.05	5.1122756	5.178588	8	4.1414957	4.300323	9
relu	2	0.05	5.0991826	5.165244	13	4.0919614	4.262389	14
relu	5	0.05	5.0740542	5.1386285	25	nan	nan	29
relu	10	0.05	5.058492	5.12151	47	nan	nan	54
tanh	1	0.05	4.029341	4.2054377	6	nan	nan	8
tanh	2	0.05	4.0091906	4.193424	9	nan	nan	14
tanh	5	0.05	3.990197	4.183032	17	nan	nan	29
tanh	10	0.05	3.97672	4.176146	32	nan	nan	53

5.4 Next

1. Understand the architecture of NNGP and NTK in library **Neural Tangents**.
2. Understand the proof of NTK.

References

Radford M Neal. Priors for infinite networks. In *Bayesian Learning for Neural Networks*, pages 29–53. Springer, 1996.

Jaehoon Lee, Yasaman Bahri, Roman Novak, Samuel S Schoenholz, Jeffrey Pennington, and Jascha Sohl-Dickstein. Deep neural networks as gaussian processes. In *International Conference on Learning Representations*, 2018.

Guofei Pang, Liu Yang, and George Em Karniadakis. Neural-net-induced gaussian process regression for function approximation and pde solution. *Journal of Computational Physics*, 384:270–288, 2019.

Johannes Friedrich. Neuronal gaussian process regression. *Advances in Neural Information Processing Systems*, 33, 2020.

Ke Wang, Geoff Pleiss, Jacob Gardner, Stephen Tyree, Kilian Q Weinberger, and Andrew Gordon Wilson. Exact gaussian processes on a million data points. In *Advances in Neural Information Processing Systems*, pages 14648–14659, 2019.

Haitao Liu, Yew-Soon Ong, Xiaobo Shen, and Jianfei Cai. When gaussian process meets big data: A review of scalable gps. *IEEE Transactions on Neural Networks and Learning Systems*, 2020.

David Burt, Carl Edward Rasmussen, and Mark Van Der Wilk. Rates of convergence for sparse variational gaussian process regression. In *International Conference on Machine Learning*, pages 862–871, 2019.

Youngmin Cho and Lawrence Saul. Kernel methods for deep learning. *Advances in neural information processing systems*, 22:342–350, 2009.

Roman Novak, Lechao Xiao, Jiri Hron, Jaehoon Lee, Alexander A Alemi, Jascha Sohl-Dickstein, and Samuel S Schoenholz. Neural tangents: Fast and easy infinite neural networks in python. *arXiv preprint arXiv:1912.02803*, 2019.

Jaehoon Lee, Lechao Xiao, Samuel Schoenholz, Yasaman Bahri, Roman Novak, Jascha Sohl-Dickstein, and Jeffrey Pennington. Wide neural networks of any depth evolve as linear models under gradient descent. In *Advances in neural information processing systems*, pages 8572–8583, 2019.