Recent development of Gaussian Process Regression

Qi Wang

January, 2021

1 Current directions

1.1 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)

1.2 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.

1.3 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

1.4 Rates of Convergence for Sparse Variational Gaussian Process Regression

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

2 Deep Neural Networks as Gaussian Processes

2.1 Gaussian Processes Regression

For $(x_i, y_i) \in (X_1, Y_1)$, with noise $\epsilon_i \in \mathcal{N}(0, \sigma^2)$ being independent with $f(x_i)$, we have

$$y_i = f(x_i) + \epsilon_i.$$

The distribution of Gaussian process $(Y_1, f(X_2))$ is

$$\begin{pmatrix} Y_1 \\ f(X_2) \end{pmatrix} \sim \mathcal{GP}\left(\begin{pmatrix} \mu_{X_1} \\ \mu_{X_2} \end{pmatrix}, \begin{pmatrix} \Sigma_{X_1X_1} + \sigma^2 I_{n_1} & \Sigma_{X_1X_2} \\ \Sigma_{X_2X_1} & \Sigma_{X_2X_2} \end{pmatrix} \right).$$

And the mean and covariance of posterior $f(X_2)|Y_1$ is

$$\tilde{\mu} = \mu_{X_2} + \Sigma_{X_2X_1}(\Sigma_{X_1X_1} + \sigma^2 I_{n_1})^{-1}(Y_1 - \mu_{X_1}),$$
$$\tilde{\Sigma} = \Sigma_{X_2X_2} - \Sigma_{X_2X_1}(\Sigma_{X_1X_1} + \sigma^2 I_{n_1})^{-1}\Sigma_{X_1X_2}$$

We use $\tilde{\mu}$ as the prediction value in machine learning.

2.2 Derivations of NNGP kernel

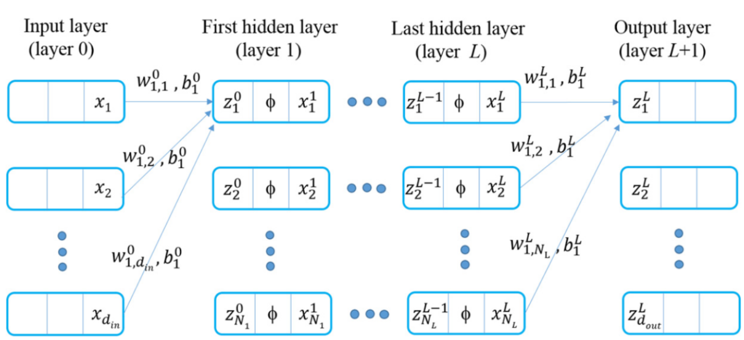

Fig. 1. A fully connected neural net with L hidden layers. For each unit or neuron in hidden layers, there exist one input z^{l-1}_i and one output x^l_i with $l = 1, 1 \cdots, L$. Layer 0 is the input layer. x_i is the i-th component of the input vector $\boldsymbol{x}$, and z^L_i is the i-th component 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^l_{ij} 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^l_i 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

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

As shown in Figure 1, we have

$$\begin{array}{lll}
\mathbf{x}, & z_j^0(\mathbf{x}) = b_j^0 + \sum_{i=1}^{\mathrm{d_{in}}} w_{ij}^0 x_i, & \mathbb{E}[z_j^0(\mathbf{x})] = 0 \\
x_j^1(\mathbf{x}) = \phi(z_j^0(\mathbf{x})), & z_j^1(\mathbf{x}) = b_j^1 + \sum_{i=1}^{\mathrm{d_{in}}} w_{ij}^1 x_i^1(\mathbf{x}), & \mathbb{E}[z_j^1(\mathbf{x})] = 0 \\
\cdots & z_j^{L-1}(\mathbf{x}) = b_j^{L-1} + \sum_{i=1}^{N_{L-1}} w_{ij}^{L-1} x_i^{L-1}(\mathbf{x}), & \mathbb{E}[z_j^{L-1}(\mathbf{x})] = 0 \\
x_j^L(\mathbf{x}) = \phi(z_j^{L-1}(\mathbf{x})), & z_j^L(\mathbf{x}) = b_j^L + \sum_{i=1}^{N_L} w_{ij}^L x_i^L(\mathbf{x}), & \mathbb{E}[z_j^L(\mathbf{x})] = 0
\end{array}$$

The expectation of pre-activation variable $\mathbb{E}[z_j^l(\mathbf{x})]$ is $\mathbf{0}$ because the parameters w_{ij}^l and data points $\mathbf{x} \in \mathbb{R}^{d_{in}}$ are all independent for any i, j, l. Thus the covariance of data points $\mathbf{x}$ and $\mathbf{x}'$ after neural network transformation at Layer L node j is (note we shorten $k^L(z_j^L(\mathbf{x}), z_j^L(\mathbf{x}'))$ to $k^L(\mathbf{x}, \mathbf{x}')$)

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

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

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

Plug 2 into 1 and calculate the integral, we can write 1 as a recursive function

$$k^L(\mathbf{x}, \mathbf{x}') = \sigma_b^2 + \sigma_w^2 F_\phi \left(k^{L-1}(\mathbf{x}, \mathbf{x}), k^{L-1}(\mathbf{x}, \mathbf{x}'), k^{L-1}(\mathbf{x}', \mathbf{x}') \right) \tag{3}$$

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.

2.3 Analytical recursive kernel function

1. When ϕ is **Relu**, Lee et al. [2018] utilized the result of Cho and Saul [2009], writing the analytical form of recursive kernel function for $l \geq 1$, where

$$k^l(z_j^l(\mathbf{x}), z_j^l(\mathbf{x}'))) \tag{4}$$

$$= \sigma_b^2 + \sigma_w^2 \int_{\mathbb{R}^2} \phi(z_i^{l-1}(\mathbf{x})) \phi(z_i^{l-1}(\mathbf{x}')) p(z_i^{l-1}(\mathbf{x}), z_i^{l-1}(\mathbf{x}')) d(z_i^{l-1}(\mathbf{x})) d(z_i^{l-1}(\mathbf{x}')) \tag{5}$$

$$= \sigma_b^2 + \sigma_w^2 \int_0^\infty \int_0^\infty z_i^{l-1}(\mathbf{x}) z_i^{l-1}(\mathbf{x}') \frac{1}{2\pi\sqrt{\det(\Sigma)}} \exp\left(-\frac{1}{2} \begin{pmatrix} z_i^{l-1}(\mathbf{x}) \\ z_i^{l-1}(\mathbf{x}') \end{pmatrix}^T \mathbf{\Sigma}^{-1} \begin{pmatrix} z_i^{l-1}(\mathbf{x}) \\ z_i^{l-1}(\mathbf{x}') \end{pmatrix} \right) d(z_i^{l-1}(\mathbf{x})) d(z_i^{l-1}(\mathbf{x}')) \tag{6}$$

$$= \sigma_b^2 + \sigma_w^2 \frac{1}{2\pi} \sqrt{k_{11}^{l-1} k_{22}^{l-1}} \left(\sin\theta + (\pi - \theta)\cos\theta \right) \tag{7}$$

$$= \sigma_b^2 + \sigma_w^2 \frac{1}{2\pi} \left(\sqrt{\det(\mathbf{\Sigma})} + (\pi - \theta) k_{12}^{l-1} \right), \tag{8}$$

$$\text{where } \theta = \arccos\left(\frac{k_{12}^{l-1}}{\sqrt{k_{11}^{l-1} k_{22}^{L-1}}} \right), \quad \text{and } \mathbf{\Sigma} = \begin{pmatrix} k_{11}^{l-1} & k_{12}^{l-1} \\ k_{12}^{l-1} & k_{22}^{l-1} \end{pmatrix} \tag{9}$$

TODO, derive equation (6) to (7) by hand.

2. When ϕ is **Erf**:
$$Erf(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt$$
which is similar to $\tanh(x)$ as shown in Figure 2, the analytical form of kernel function is

$$k^l(z_j^l(\mathbf{x}), z_j^l(\mathbf{x}'))) \tag{10}$$
$$= \sigma_b^2 + \sigma_w^2 \int_{\mathbb{R}^2} \phi(z_i^{l-1}(\mathbf{x}))\phi(z_i^{l-1}(\mathbf{x}'))p(z_i^{l-1}(\mathbf{x}), z_i^{l-1}(\mathbf{x}'))d(z_i^{l-1}(\mathbf{x}))d(z_i^{l-1}(\mathbf{x}')) \tag{11}$$
$$= \sigma_b^2 + \sigma_w^2 \frac{2}{\pi} \arctan(\frac{2k_{12}^{l-1}}{\sqrt{k_{11}^{l-1}k_{22}^{l-1} - 4(k_{12}^{l-1})^2}}) \tag{12}$$

Equation 12 is implemented in package (`https://github.com/brain-research/nngp`). TODO, derive equation (11) to (12) by hand.

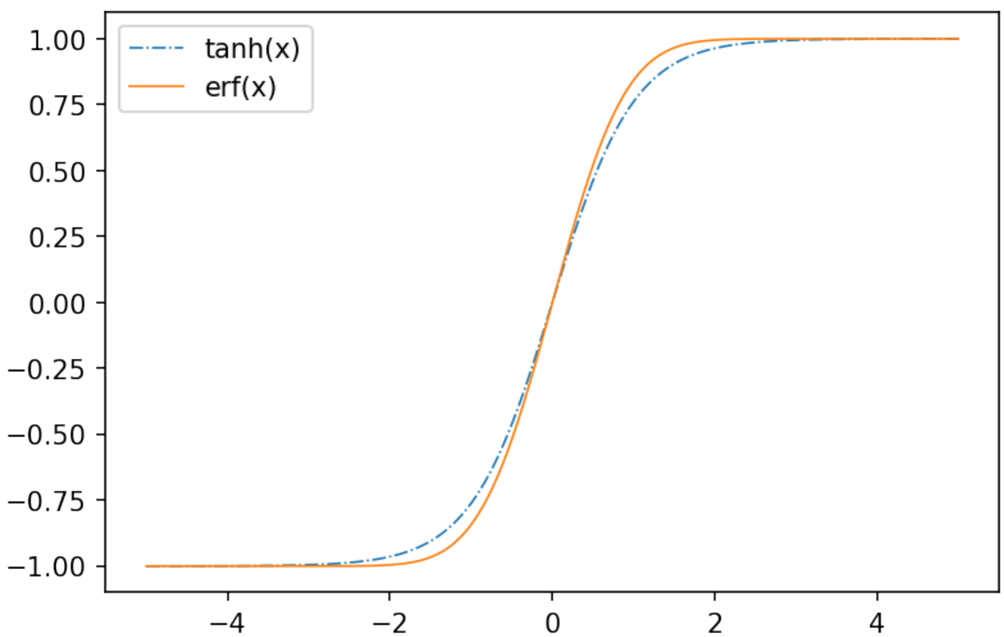

Figure 2: function $erf(x)$ versus $tanh(x)$

3. Package (`https://github.com/brain-research/nngp`) also implements analytical kernels for other nonlinearity activation functions, such as sin, LeakyRelu, Abs, etc.

2.4 Nonanalytical integration

For some nonlinearity activation function ϕ, such as tanh, Lee et al. [2018] proposed an approximation of the double gaussian integration in equation 1 (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.

2.5 NNGP 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.)

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

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.

4 Next

1. Derive the double Gaussian integration for those analytical nonlinearity case.
2. Understand the proof of NTK and the architecture of NTK in package **Neural Tangents**.
3. Consider one direction: tune parameter by optimization on log marginal likelihood.

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.