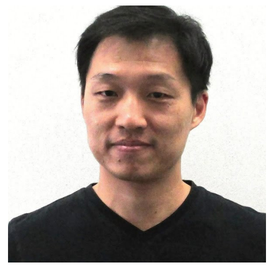
Understanding infinite width neural networks (from the perspective of statistical physics)

Jascha Sohl-Dickstein

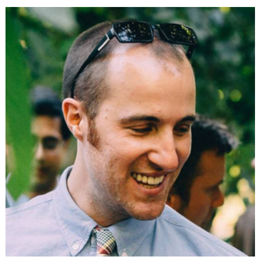
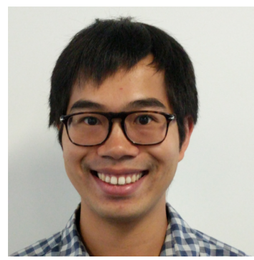
Anthropic ← we're hiring!

Collaborators







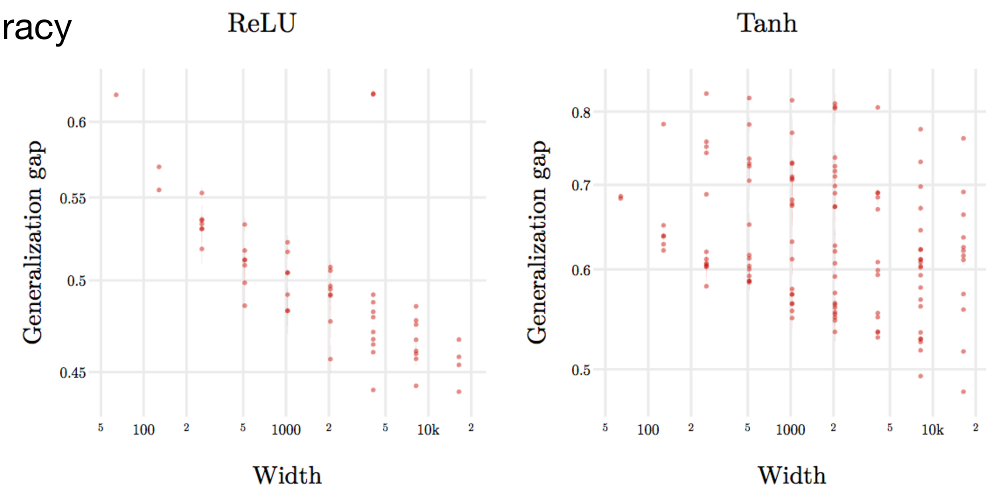
Yasaman Bahri, Roman Novak, Jaehoon Lee, Surya Ganguli, Jeffrey Pennington,
Sam Schoenholz, Jascha Sohl-Dickstein, Lechao Xiao, Greg Yang, Jiri Hron

Why study overparameterized neural networks?

Test accuracy increases with model width:

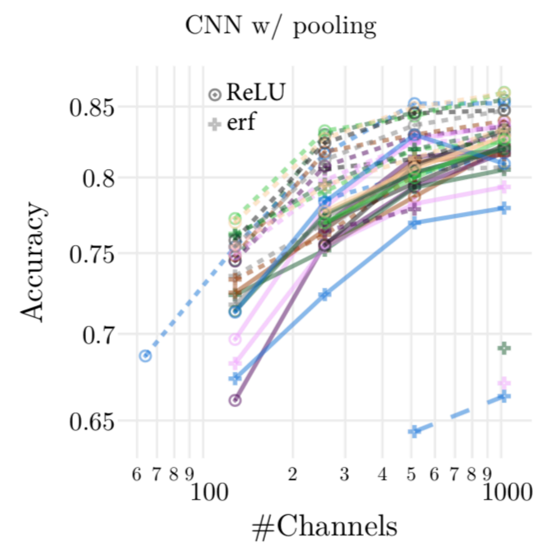
Fully-Connected, CIFAR-10

Networks with 100% training accuracy



Lee and Bahri, et al.

Convolutional, CIFAR-10



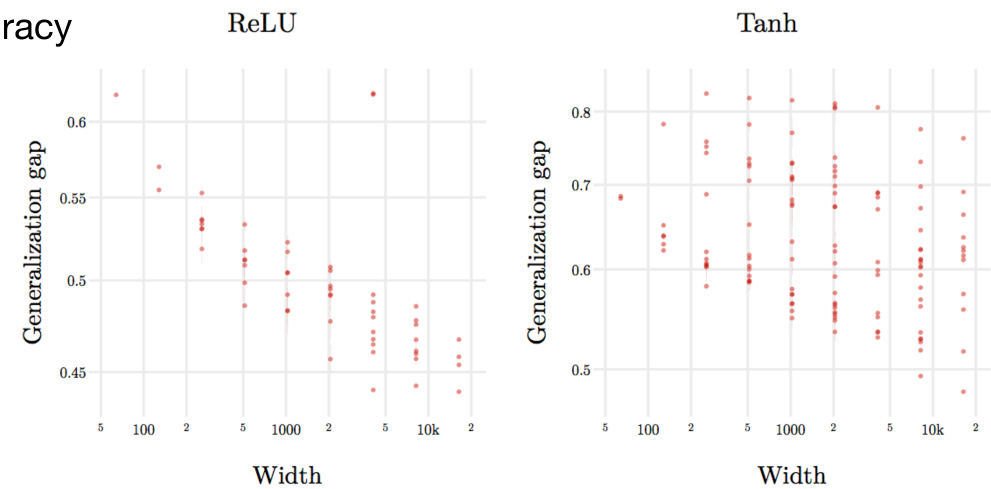
Novak and Xiao, et al.

Why study overparameterized neural networks?

Test accuracy increases with model width:

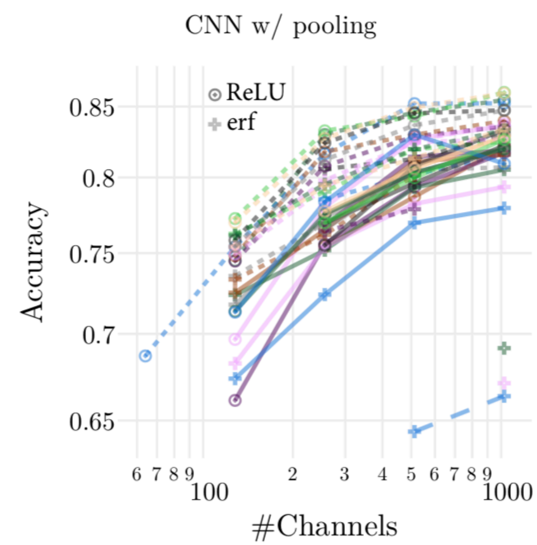
Networks with 100% training accuracy

Fully-Connected, CIFAR-10



Lee and Bahri, et al.

Convolutional, CIFAR-10



Novak and Xiao, et al.

What happens in the limit of infinite width?

As neural networks become infinitely wide, they become analytically simple

	Parameter Space	Function Space
Bayesian Inference		
Gradient Descent Training		

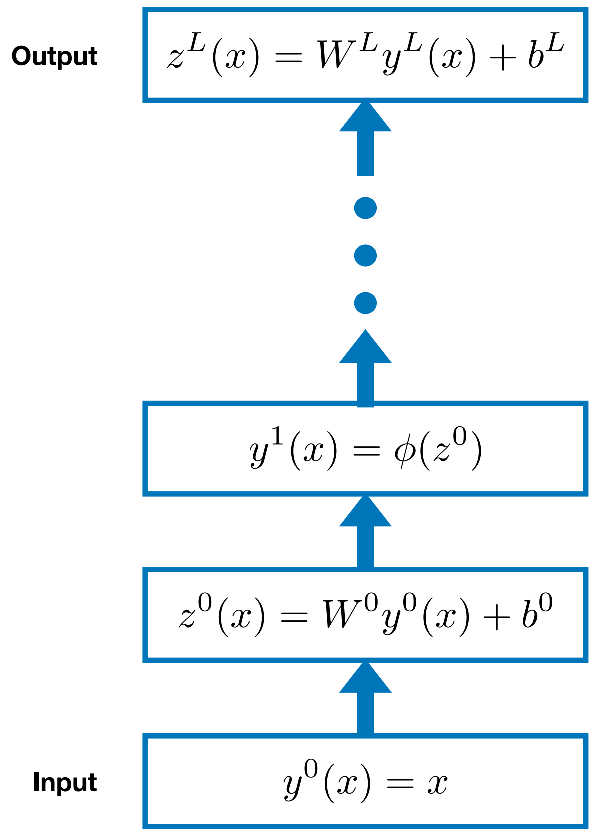
https://github.com/google/neural-tangents

As neural networks become infinitely wide, they become analytically simple

	Parameter Space	**Function Space**
Bayesian Inference		**Neural Network Gaussian Process** $z(x) \sim \mathcal{GP}(z(x))$ $z(x^*)\big\vert\mathcal{D} \sim \mathcal{GP}(z(x^*)\big\vert\mathcal{D})$
Gradient Descent Training		

https://github.com/google/neural-tangents

Distribution over functions induced by randomly initialized feedforward network



$$x \equiv \text{input}$$

$$y^l(x) = \begin{cases} x & l = 0 \\ \phi(z^{l-1}(x)) & l > 0 \end{cases}$$

$$z_i^l(x) = \sum_j W_{ij}^l y_j^l(x) + b_i^l$$

$$W_{ij}^l \sim \mathcal{N}\left(0, \frac{\sigma_w^2}{n^l}\right)$$

$$b_i^l \sim \mathcal{N}\left(0, \sigma_b^2\right)$$

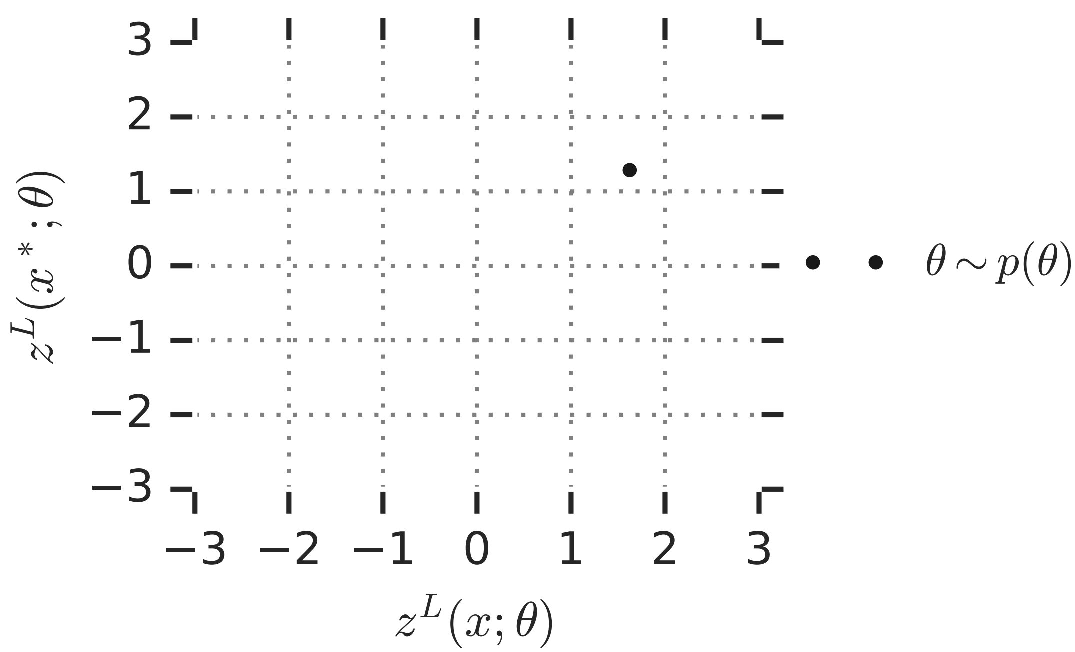
$$\phi(\cdot) \equiv \text{nonlinearity}$$

$$y^l(x), z^{l-1}(x) \in \mathbb{R}^{n^l \times 1}$$

$$n^L = 1$$

$$\theta = \{W^0, b^0, \dots, W^L, b^L\}$$

Cartoon distribution over network outputs



$$x \equiv \text{input}$$

$$y^l(x) = \begin{cases} x & l = 0 \\ \phi(z^{l-1}(x)) & l > 0 \end{cases}$$

$$z_i^l(x) = \sum_j W_{ij}^l y_j^l(x) + b_i^l$$

$$W_{ij}^l \sim \mathcal{N}\left(0, \frac{\sigma_w^2}{n^l}\right)$$

$$b_i^l \sim \mathcal{N}\left(0, \sigma_b^2\right)$$

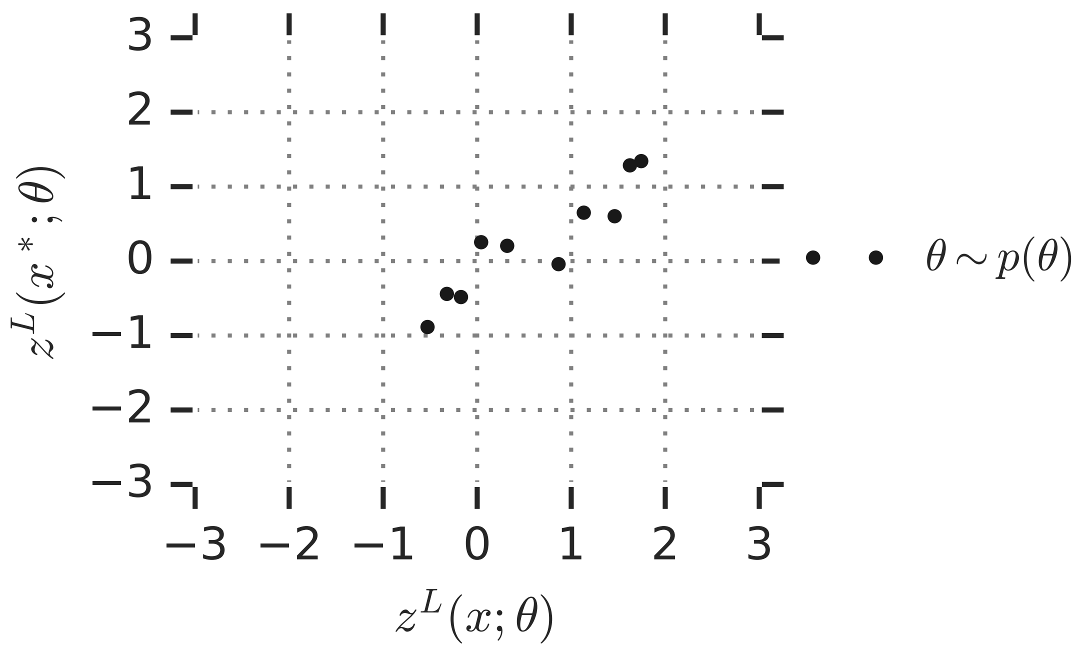
$$\phi(\cdot) \equiv \text{nonlinearity}$$

$$y^l(x), z^{l-1}(x) \in \mathbb{R}^{n^l \times 1}$$

$$n^L = 1$$

$$\theta = \{W^0, b^0, \ldots, W^L, b^L\}$$

Cartoon distribution over network outputs



$$x \equiv \text{input}$$

$$y^l(x) = \begin{cases} x & l = 0 \\ \phi(z^{l-1}(x)) & l > 0 \end{cases}$$

$$z_i^l(x) = \sum_j W_{ij}^l y_j^l(x) + b_i^l$$

$$W_{ij}^l \sim \mathcal{N}\left(0, \frac{\sigma_w^2}{n^l}\right)$$

$$b_i^l \sim \mathcal{N}\left(0, \sigma_b^2\right)$$

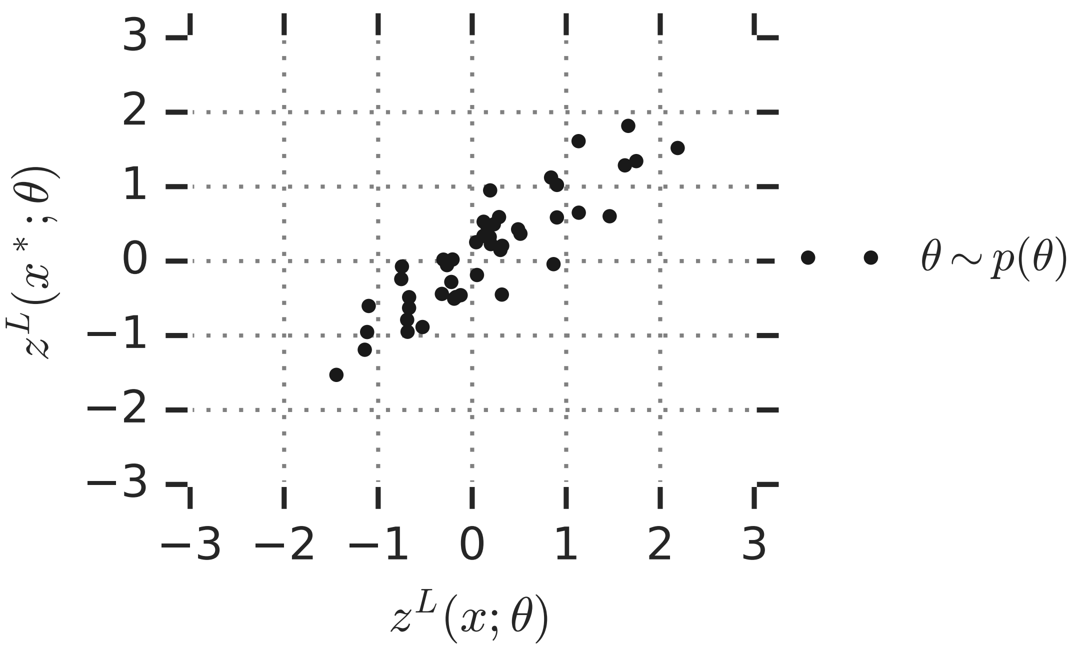
$$\phi(\cdot) \equiv \text{nonlinearity}$$

$$y^l(x), z^{l-1}(x) \in \mathbb{R}^{n^l \times 1}$$

$$n^L = 1$$

$$\theta = \{W^0, b^0, \ldots, W^L, b^L\}$$

Cartoon distribution over network outputs



$$x \equiv \text{input}$$

$$y^l(x) = \begin{cases} x & l = 0 \\ \phi(z^{l-1}(x)) & l > 0 \end{cases}$$

$$z_i^l(x) = \sum_j W_{ij}^l y_j^l(x) + b_i^l$$

$$W_{ij}^l \sim \mathcal{N}\left(0, \frac{\sigma_w^2}{n^l}\right)$$

$$b_i^l \sim \mathcal{N}\left(0, \sigma_b^2\right)$$

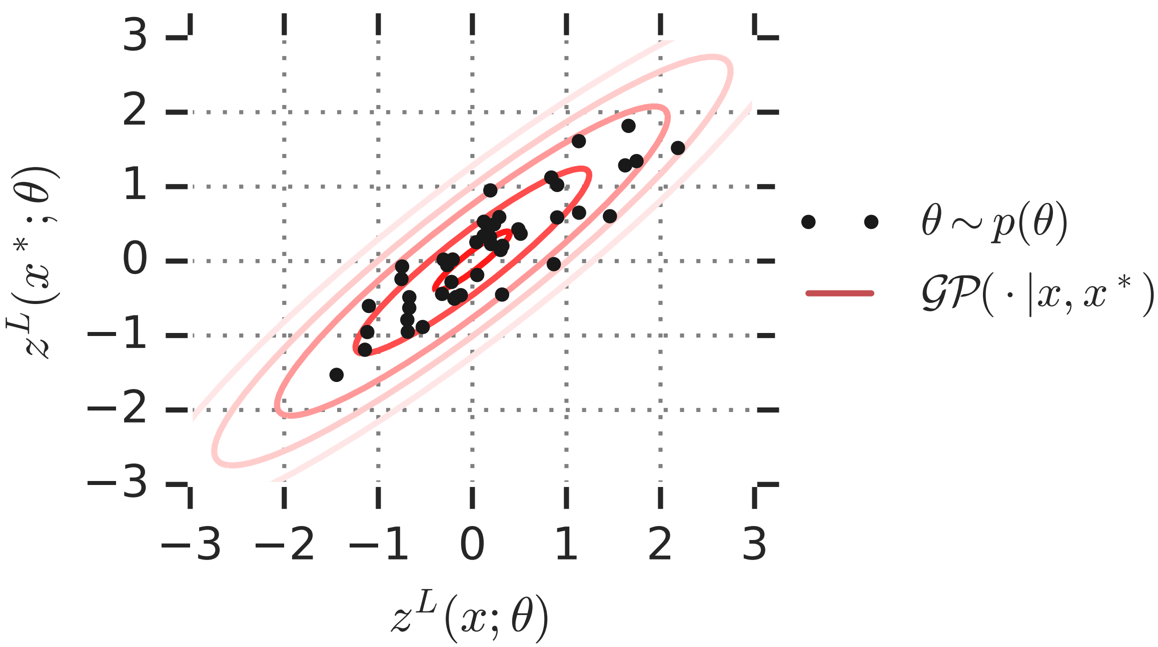
$$\phi(\cdot) \equiv \text{nonlinearity}$$

$$y^l(x), z^{l-1}(x) \in \mathbb{R}^{n^l \times 1}$$

$$n^L = 1$$

$$\theta = \{W^0, b^0, \ldots, W^L, b^L\}$$

Cartoon distribution over network outputs



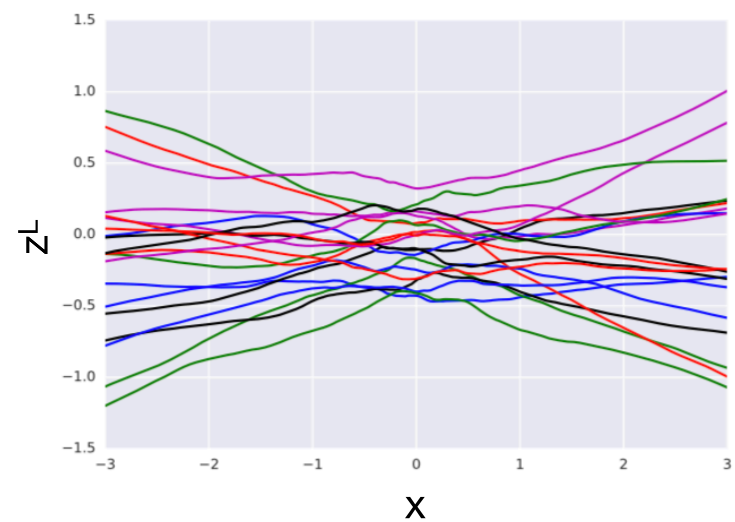
$$x \equiv \text{input}$$
$$y^l(x) = \begin{cases} x & l = 0 \\ \phi(z^{l-1}(x)) & l > 0 \end{cases}$$
$$z_i^l(x) = \sum_j W_{ij}^l y_j^l(x) + b_i^l$$
$$W_{ij}^l \sim \mathcal{N}\left(0, \frac{\sigma_w^2}{n^l}\right)$$
$$b_i^l \sim \mathcal{N}\left(0, \sigma_b^2\right)$$
$$\phi(\cdot) \equiv \text{nonlinearity}$$
$$y^l(x), z^{l-1}(x) \in \mathbb{R}^{n^l \times 1}$$
$$n^L = 1$$
$$\theta = \{W^0, b^0, \dots, W^L, b^L\}$$

Reminder: Gaussian Processes (GPs)

Definition: $z^L(x) \sim \mathcal{GP}(\mu, K)$ is a Gaussian process, with mean and covariance functions $\mu(x)$ and $K(x, x')$, if any finite set of draws $\left[z^L(x_1), \ldots, z^L(x_m)\right]^\top$ follows $\mathcal{N}(\mu, \mathbf{K})$ with

$$\mu = \begin{bmatrix} \mu(x_1) \\ \vdots \\ \mu(x_m) \end{bmatrix}, \qquad \mathbf{K} = \begin{bmatrix} K(x_1, x_1) & \cdots & K(x_1, x_m) \\ \vdots & \ddots & \vdots \\ K(x_m, x_1) & \cdots & K(x_m, x_m) \end{bmatrix}.$$

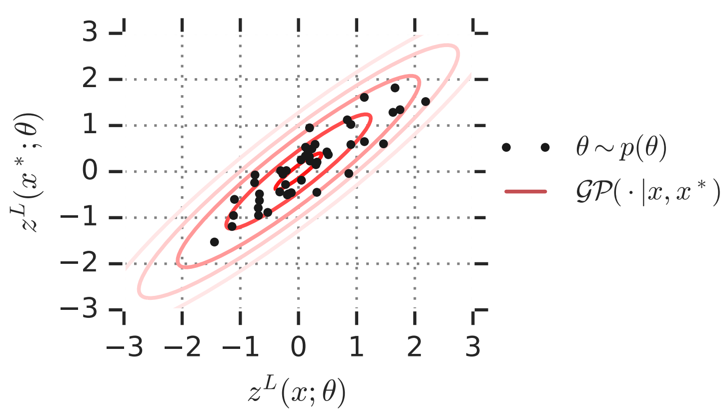
Draws from
Neural Network-equivalent GP
(NNGP)



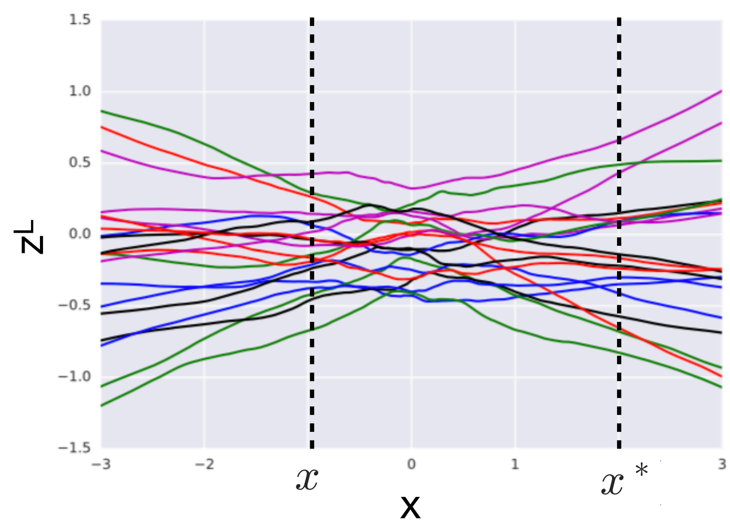
Reminder: Gaussian Processes (GPs)

Definition: $z^L(x) \sim \mathcal{GP}(\mu, K)$ is a Gaussian process, with mean and covariance functions $\mu(x)$ and $K(x, x')$, if any finite set of draws $\left[z^L(x_1), \ldots, z^L(x_m)\right]^\top$ follows $\mathcal{N}(\mu, \mathbf{K})$ with

$$\mu = \begin{bmatrix} \mu(x_1) \\ \vdots \\ \mu(x_m) \end{bmatrix}, \qquad \mathbf{K} = \begin{bmatrix} K(x_1, x_1) & \cdots & K(x_1, x_m) \\ \vdots & \ddots & \vdots \\ K(x_m, x_1) & \cdots & K(x_m, x_m) \end{bmatrix}.$$



Draws from
Neural Network-equivalent GP
(NNGP)



Network output is a GP: 4 step sketch

1. $z^l \big| y^l$ is a GP

$$z_i^l(x) = \sum_j W_{ij}^l y_j^l(x) + b_i^l$$

$$x \equiv \text{input}$$

$$y^l(x) = \begin{cases} x & l = 0 \\ \phi(z^{l-1}(x)) & l > 0 \end{cases}$$

$$z_i^l(x) = \sum_j W_{ij}^l y_j^l(x) + b_i^l$$

$$W_{ij}^l \sim \mathcal{N}\left(0, \frac{\sigma_w^2}{n^l}\right)$$

$$b_i^l \sim \mathcal{N}\left(0, \sigma_b^2\right)$$

$$\phi(\cdot) \equiv \text{nonlinearity}$$

$$y^l(x), z^{l-1}(x) \in \mathbb{R}^{n^l \times 1}$$

$$\theta = \{W^0, b^0, \ldots, W^L, b^L\}$$

[Neal, 1996]
[Lee, Bahri, et al., 2018]
[Matthews, et al., 2018]
[Garriga-Alonso, et al., 2019]
[Novak, Xiao, et al., 2019]
[Yang, 2019]

Network output is a GP: 4 step sketch

1. $z^l \big| y^l$ is a GP

$$z_i^l(x) = \sum_j \boxed{W_{ij}^l} y_j^l(x) + \boxed{b_i^l}$$

Gaussian

$$x \equiv \text{input}$$

$$y^l(x) = \begin{cases} x & l = 0 \\ \phi(z^{l-1}(x)) & l > 0 \end{cases}$$

$$z_i^l(x) = \sum_j W_{ij}^l y_j^l(x) + b_i^l$$

$$W_{ij}^l \sim \mathcal{N}\left(0, \frac{\sigma_w^2}{n^l}\right)$$

$$b_i^l \sim \mathcal{N}\left(0, \sigma_b^2\right)$$

$$\phi(\cdot) \equiv \text{nonlinearity}$$

$$y^l(x), z^{l-1}(x) \in \mathbb{R}^{n^l \times 1}$$

$$\theta = \{W^0, b^0, \ldots, W^L, b^L\}$$

[Neal, 1996]
[Lee, Bahri, et al., 2018]
[Matthews, et al., 2018]
[Garriga-Alonso, et al., 2019]
[Novak, Xiao, et al., 2019]
[Yang, 2019]

Network output is a GP: 4 step sketch

1. $z^l \big| y^l$ is a GP

$$z_i^l(x) = \sum_j \boxed{W_{ij}^l} y_j^l(x) + \boxed{b_i^l}$$

Gaussian

weighted sum of
Gaussian random variables
(with weights y_j^l)

$$x \equiv \text{input}$$

$$y^l(x) = \begin{cases} x & l = 0 \\ \phi(z^{l-1}(x)) & l > 0 \end{cases}$$

$$z_i^l(x) = \sum_j W_{ij}^l y_j^l(x) + b_i^l$$

$$W_{ij}^l \sim \mathcal{N}\left(0, \frac{\sigma_w^2}{n^l}\right)$$

$$b_i^l \sim \mathcal{N}\left(0, \sigma_b^2\right)$$

$$\phi(\cdot) \equiv \text{nonlinearity}$$

$$y^l(x), z^{l-1}(x) \in \mathbb{R}^{n^l \times 1}$$

$$\theta = \{W^0, b^0, \ldots, W^L, b^L\}$$

[Neal, 1996]
[Lee, Bahri, et al., 2018]
[Matthews, et al., 2018]
[Garriga-Alonso, et al., 2019]
[Novak, Xiao, et al., 2019]
[Yang, 2019]

Network output is a GP: 4 step sketch

1. $z^l \big| y^l$ is a GP

$$z_i^l(x) = \sum_j W_{ij}^l y_j^l(x) + b_i^l$$

weighted sum of
Gaussian random variables
(with weights y_j^l)

Gaussian

$$z_i^l \mid y^l \sim \mathcal{GP}\left(0, \sigma_w^2 K^l + \sigma_b^2\right)$$

$$K^l(x, x') = \frac{1}{n^l} \sum_i y_i^l(x) y_i^l(x')$$

$$x \equiv \text{input}$$

$$y^l(x) = \begin{cases} x & l = 0 \\ \phi(z^{l-1}(x)) & l > 0 \end{cases}$$

$$z_i^l(x) = \sum_j W_{ij}^l y_j^l(x) + b_i^l$$

$$W_{ij}^l \sim \mathcal{N}\left(0, \frac{\sigma_w^2}{n^l}\right)$$

$$b_i^l \sim \mathcal{N}\left(0, \sigma_b^2\right)$$

$$\phi(\cdot) \equiv \text{nonlinearity}$$

$$y^l(x), z^{l-1}(x) \in \mathbb{R}^{n^l \times 1}$$

$$\theta = \{W^0, b^0, \ldots, W^L, b^L\}$$

[Neal, 1996]
[Lee, Bahri, et al., 2018]
[Matthews, et al., 2018]
[Garriga-Alonso, et al., 2019]
[Novak, Xiao, et al., 2019]
[Yang, 2019]

Network output is a GP: 4 step sketch

1. $z^l \big| y^l$ is a GP
2. $z^l \big| K^l$ is a GP

$$z_i^l \mid y^l \sim \mathcal{GP}\left(0, \sigma_w^2 K^l + \sigma_b^2\right)$$

only depends on y^l via K^l

$$x \equiv \text{input}$$
$$y^l(x) = \begin{cases} x & l = 0 \\ \phi(z^{l-1}(x)) & l > 0 \end{cases}$$
$$z_i^l(x) = \sum_j W_{ij}^l y_j^l(x) + b_i^l$$
$$W_{ij}^l \sim \mathcal{N}\left(0, \frac{\sigma_w^2}{n^l}\right)$$
$$b_i^l \sim \mathcal{N}\left(0, \sigma_b^2\right)$$
$$\phi(\cdot) \equiv \text{nonlinearity}$$
$$y^l(x), z^{l-1}(x) \in \mathbb{R}^{n^l \times 1}$$
$$\theta = \{W^0, b^0, \ldots, W^L, b^L\}$$
$$K^l(x, x') = \frac{1}{n^l} \sum_i y_i^l(x) y_i^l(x')$$

[Neal, 1996]
[Lee, Bahri, et al., 2018]
[Matthews, et al., 2018]
[Garriga-Alonso, et al., 2019]
[Novak, Xiao, et al., 2019]
[Yang, 2019]

Network output is a GP: 4 step sketch

1. $z^l \big| y^l$ is a GP
2. $z^l \big| K^l$ is a GP

$$z_i^l \mid y^l \sim \mathcal{GP}\left(0, \sigma_w^2 K^l + \sigma_b^2\right)$$

only depends on
y^l via K^l

$$z_i^l \mid K^l \sim \mathcal{GP}\left(0, \sigma_w^2 K^l + \sigma_b^2\right)$$

$$x \equiv \text{input}$$
$$y^l(x) = \begin{cases} x & l = 0 \\ \phi(z^{l-1}(x)) & l > 0 \end{cases}$$
$$z_i^l(x) = \sum_j W_{ij}^l y_j^l(x) + b_i^l$$
$$W_{ij}^l \sim \mathcal{N}\left(0, \frac{\sigma_w^2}{n^l}\right)$$
$$b_i^l \sim \mathcal{N}\left(0, \sigma_b^2\right)$$
$$\phi(\cdot) \equiv \text{nonlinearity}$$
$$y^l(x), z^{l-1}(x) \in \mathbb{R}^{n^l \times 1}$$
$$\theta = \{W^0, b^0, \dots, W^L, b^L\}$$
$$K^l(x, x') = \frac{1}{n^l} \sum_i y_i^l(x) y_i^l(x')$$

[Neal, 1996]
[Lee, Bahri, et al., 2018]
[Matthews, et al., 2018]
[Garriga-Alonso, et al., 2019]
[Novak, Xiao, et al., 2019]
[Yang, 2019]

Network output is a GP: 4 step sketch

1. $z^l \big| y^l$ is a GP
2. $z^l \big| K^l$ is a GP
3. As layer width $n^l \to \infty$, $K^l \big| K^{l-1}$ becomes deterministic

$$K^0(x, x') = \frac{1}{n^0} \sum_i x_i x_i',$$

$$\lim_{n^l \to \infty} K^l = F\left(K^{l-1}\right), \quad \text{for } l > 0$$

By concentration of measure argument

$$x \equiv \text{input}$$

$$y^l(x) = \begin{cases} x & l = 0 \\ \phi(z^{l-1}(x)) & l > 0 \end{cases}$$

$$z_i^l(x) = \sum_j W_{ij}^l y_j^l(x) + b_i^l$$

$$W_{ij}^l \sim \mathcal{N}\left(0, \frac{\sigma_w^2}{n^l}\right)$$

$$b_i^l \sim \mathcal{N}\left(0, \sigma_b^2\right)$$

$$\phi(\cdot) \equiv \text{nonlinearity}$$

$$y^l(x), z^{l-1}(x) \in \mathbb{R}^{n^l \times 1}$$

$$\theta = \{W^0, b^0, \dots, W^L, b^L\}$$

$$K^l(x, x') = \frac{1}{n^l} \sum_i y_i^l(x) y_i^l(x')$$

[Neal, 1996]
[Lee, Bahri, et al., 2018]
[Matthews, et al., 2018]
[Garriga-Alonso, et al., 2019]
[Novak, Xiao, et al., 2019]
[Yang, 2019]

Network output is a GP: 4 step sketch

1. $z^l \big| y^l$ is a GP
2. $z^l \big| K^l$ is a GP
3. As layer width $n^l \to \infty,\ K^l \big| K^{l-1}$ becomes deterministic
4. $z^l \big| x$ is a Neural Network-equivalent GP (NNGP)

$$z_i^L \mid K^L \sim \mathcal{GP}\left(0, \sigma_w^2 K^L + \sigma_b^2\right)$$
$$\lim_{n^l \to \infty} K^l = F\left(K^{l-1}\right)$$

$$x \equiv \text{input}$$
$$y^l(x) = \begin{cases} x & l = 0 \\ \phi(z^{l-1}(x)) & l > 0 \end{cases}$$
$$z_i^l(x) = \sum_j W_{ij}^l y_j^l(x) + b_i^l$$
$$W_{ij}^l \sim \mathcal{N}\left(0, \frac{\sigma_w^2}{n^l}\right)$$
$$b_i^l \sim \mathcal{N}\left(0, \sigma_b^2\right)$$
$$\phi(\cdot) \equiv \text{nonlinearity}$$
$$y^l(x), z^{l-1}(x) \in \mathbb{R}^{n^l \times 1}$$
$$\theta = \{W^0, b^0, \ldots, W^L, b^L\}$$
$$K^l(x, x') = \frac{1}{n^l} \sum_i y_i^l(x) y_i^l(x')$$
$$\lim_{n^l \to \infty} K^l = F(K^{l-1})$$

[Neal, 1996]
[Lee, Bahri, et al., 2018]
[Matthews, et al., 2018]
[Garriga-Alonso, et al., 2019]
[Novak, Xiao, et al., 2019]
[Yang, 2019]

Network output is a GP: 4 step sketch

1. $z^l \big| y^l$ is a GP
2. $z^l \big| K^l$ is a GP
3. As layer width $n^l \to \infty$, $K^l \big| K^{l-1}$ becomes deterministic
4. $z^l \big| x$ is a Neural Network-equivalent GP (NNGP)

$$z_i^L \mid K^L \sim \mathcal{GP}\left(0, \sigma_w^2 K^L + \sigma_b^2\right)$$

$$\lim_{n^l \to \infty} K^l = F\left(K^{l-1}\right)$$

$$\lim_{\min(n^1,\ldots,n^L)\to\infty} K^L = F \circ F \cdots \left(K^0\right) = F^L\left(K^0\right)$$

$$z_i^L(x) \sim \mathcal{GP}\left(0, \sigma_w^2 F^L\left(K^0\right) + \sigma_b^2\right)$$

$$x \equiv \text{input}$$

$$y^l(x) = \begin{cases} x & l = 0 \\ \phi(z^{l-1}(x)) & l > 0 \end{cases}$$

$$z_i^l(x) = \sum_j W_{ij}^l y_j^l(x) + b_i^l$$

$$W_{ij}^l \sim \mathcal{N}\left(0, \frac{\sigma_w^2}{n^l}\right)$$

$$b_i^l \sim \mathcal{N}\left(0, \sigma_b^2\right)$$

$$\phi(\cdot) \equiv \text{nonlinearity}$$

$$y^l(x), z^{l-1}(x) \in \mathbb{R}^{n^l \times 1}$$

$$\theta = \{W^0, b^0, \ldots, W^L, b^L\}$$

$$K^l(x, x') = \frac{1}{n^l} \sum_i y_i^l(x) y_i^l(x')$$

$$\lim_{n^l \to \infty} K^l = F(K^{l-1})$$

[Neal, 1996]
[Lee, Bahri, et al., 2018]
[Matthews, et al., 2018]
[Garriga-Alonso, et al., 2019]
[Novak, Xiao, et al., 2019]
[Yang, 2019]

Distribution over functions induced by random initialization of wide neural network is a Gaussian process

- What can we do with this understanding?

- Predict trainability
  - Random initialization corresponds to the start of training

- Perform inference with infinitely wide Bayesian neural networks
  - (without ever instantiating a neural network)

Distribution over functions induced by random initialization of wide neural network is a Gaussian process

- What can we do with this understanding?

- **Predict trainability**
  - **Random initialization corresponds to the start of training**

- Perform inference with infinitely wide Bayesian neural networks
  - (without ever instantiating a neural network)

Trainability determined by phase diagram

Large depth behavior of kernel

$$K^L = K^* + \mathcal{O}\left(\exp(-L \,/\, \xi_c)\right)$$

$$x \equiv \text{input}$$
$$y^l(x) = \begin{cases} x & l = 0 \\ \phi(z^{l-1}(x)) & l > 0 \end{cases}$$
$$z_i^l(x) = \sum_j W_{ij}^l y_j^l(x) + b_i^l$$
$$W_{ij}^l \sim \mathcal{N}\left(0, \frac{\sigma_w^2}{n^l}\right)$$
$$b_i^l \sim \mathcal{N}\left(0, \sigma_b^2\right)$$
$$\phi(\cdot) \equiv \text{nonlinearity}$$
$$y^l(x), z^{l-1}(x) \in \mathbb{R}^{n^l \times 1}$$
$$\theta = \{W^0, b^0, \ldots, W^L, b^L\}$$
$$K^l(x, x') = \frac{1}{n^l} \sum_i y_i^l(x) y_i^l(x')$$
$$\lim_{n^l \to \infty} K^l = F(K^{l-1})$$

[Poole, et al., 2016]
[Schoenholz, et al., 2016]

Trainability determined by phase diagram

Large depth behavior of kernel

$$K^L = K^* + \mathcal{O}\left(\exp(-L \,/\, \xi_c)\right)$$

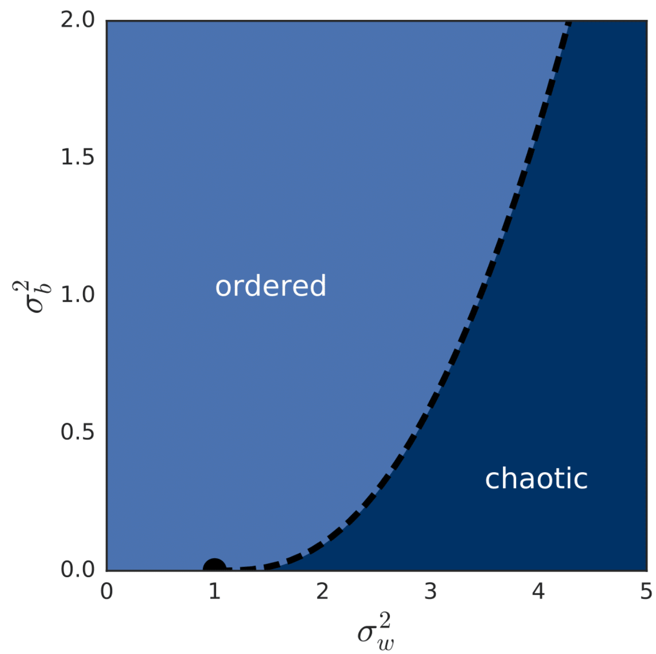
[Poole, et al., 2016]
[Schoenholz, et al., 2016]

Trainability determined by phase diagram

Large depth behavior of kernel

$$K^L = K^* + \mathcal{O}\left(\exp(-L \,/\, \xi_c)\right)$$

tanh fully connected network



[Poole, et al., 2016]
[Schoenholz, et al., 2016]

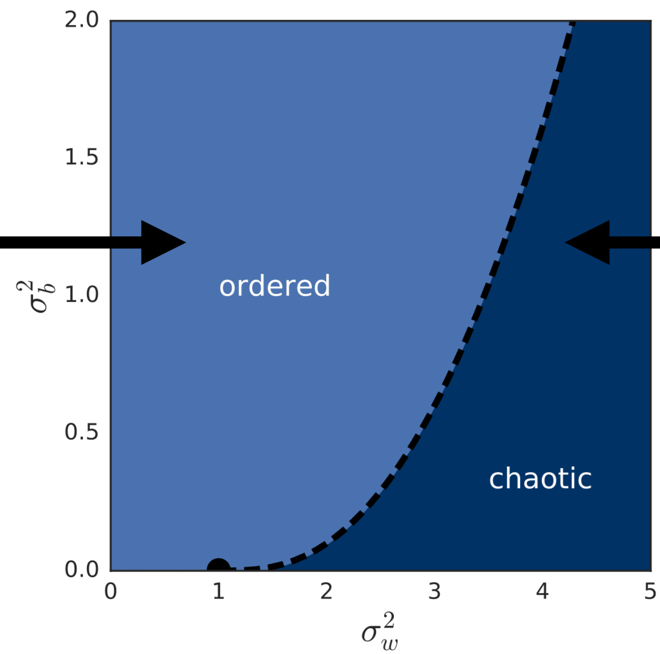
Trainability determined by phase diagram

Large depth behavior of kernel

$$K^L = K^* + \mathcal{O}\left(\exp(-L \,/\, \xi_c)\right)$$

- Dissimilar inputs converge
- $K^* = \beta \mathbf{1}\mathbf{1}^T$

tanh fully connected network



- Similar inputs diverge
- $K^* = \alpha \mathbf{I} + \beta \mathbf{1}\mathbf{1}^T$

[Poole, et al., 2016]
[Schoenholz, et al., 2016]

Trainability determined by phase diagram

Large depth behavior of kernel

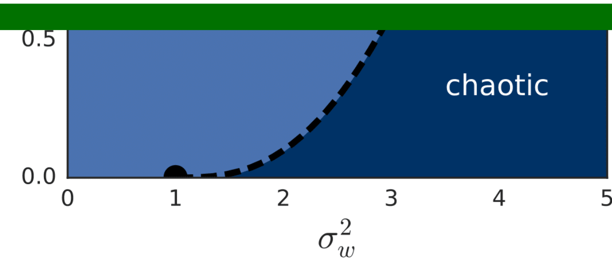
$$K^L = K^* + \mathcal{O}\left(\exp(-L\,/\,\xi_c)\right)$$

- Gradient is a linear operator
- Gradient of (sufficiently smooth) GP is still a GP
- Jacobian *J* of infinite width network is a GP

- Dissi
  conv
- $K^* =$

uts

$\beta \mathbf{1}\mathbf{1}^T$



[Poole, et al., 2016]
[Schoenholz, et al., 2016]

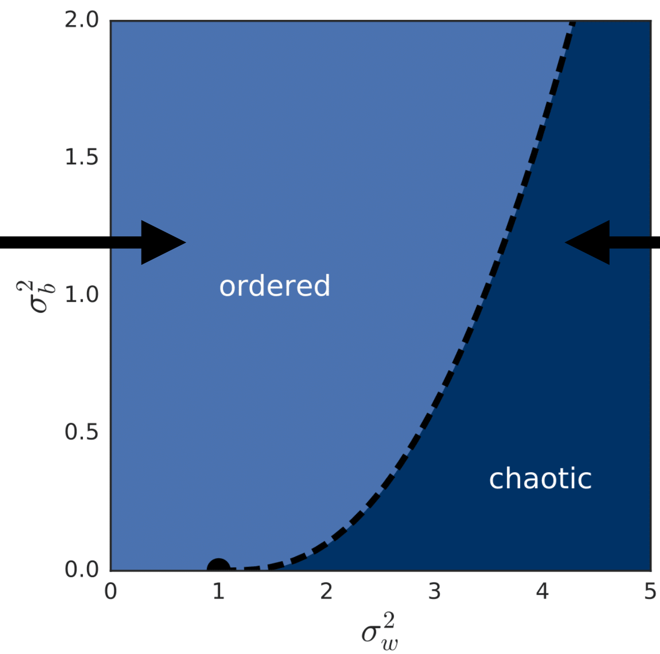
Trainability determined by phase diagram

Large depth behavior of kernel

$$K^L = K^* + \mathcal{O}\left(\exp(-L \,/\, \xi_c)\right)$$

- Dissimilar inputs converge
- $K^* = \beta \mathbf{1}\mathbf{1}^T$

tanh fully connected network



- Similar inputs diverge
- $K^* = \alpha \mathbf{I} + \beta \mathbf{1}\mathbf{1}^T$

[Poole, et al., 2016]
[Schoenholz, et al., 2016]

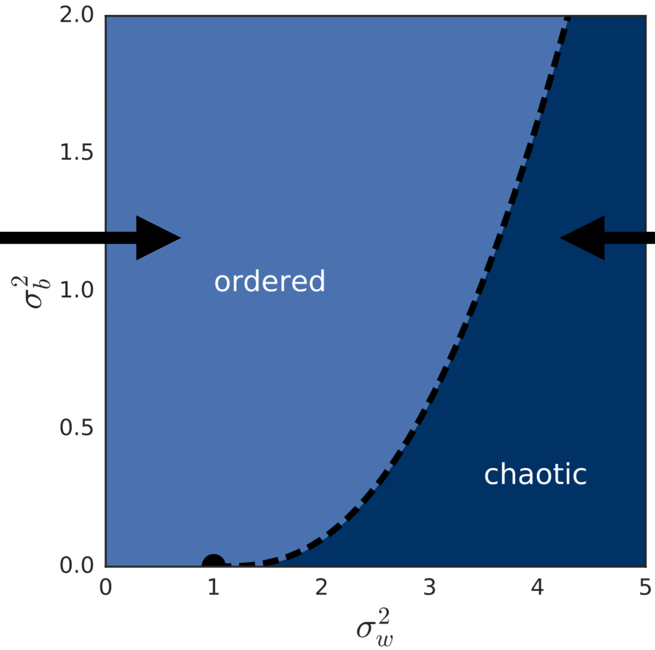
Trainability determined by phase diagram

Large depth behavior of kernel

$$K^L = K^* + \mathcal{O}\left(\exp(-L \,/\, \xi_c)\right)$$

- Dissimilar inputs converge
- $K^* = \beta \mathbf{1}\mathbf{1}^T$
- Gradients vanish $||J|| \to 0$

tanh fully connected network



- Similar inputs diverge
- $K^* = \alpha \mathbf{I} + \beta \mathbf{1}\mathbf{1}^T$
- Gradients explode $||J|| \to \infty$

[Poole, et al., 2016]
[Schoenholz, et al., 2016]

Trainability determined by phase diagram

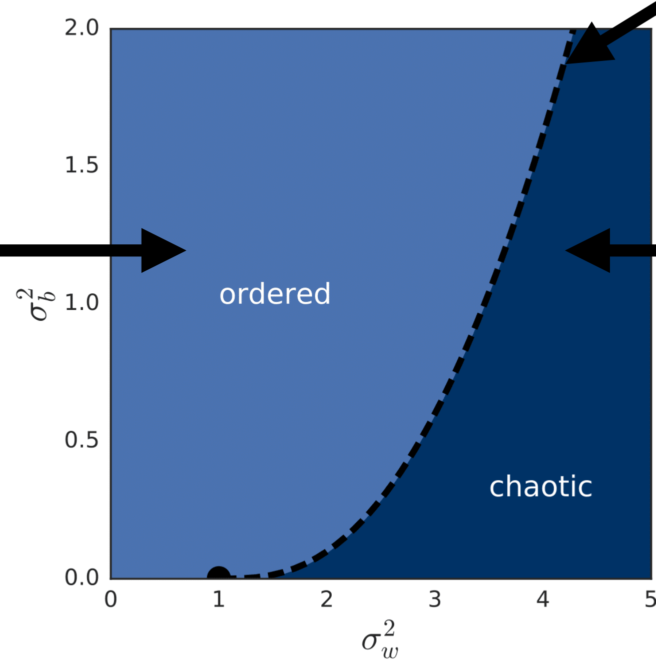
Large depth behavior of kernel

$$K^L = K^* + \mathcal{O}\left(\exp(-L \,/\, \xi_c)\right)$$

- Critical line
- $\xi_c \to \infty$
- Gradient does not exponentially grow or shrink

- Dissimilar inputs converge
- $K^* = \beta \mathbf{1}\mathbf{1}^T$
- Gradients vanish $||J|| \to 0$

tanh fully connected network

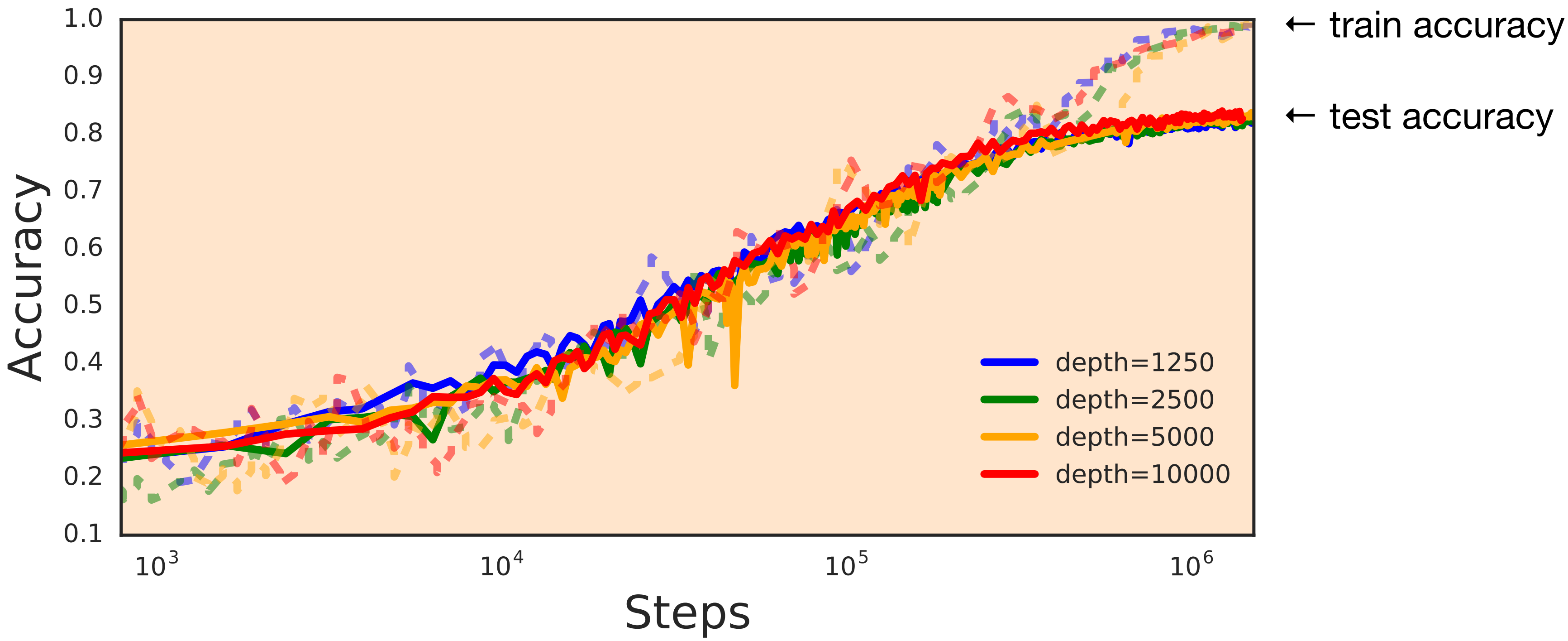


- Similar inputs diverge
- $K^* = \alpha \mathbf{I} + \beta \mathbf{1}\mathbf{1}^T$
- Gradients explode $||J|| \to \infty$

[Poole, et al., 2016]
[Schoenholz, et al., 2016]

Phase diagram and depth scale ξ_c predict trainability

Critically initialized* vanilla CNN with tanh nonlinearity on CIFAR-10



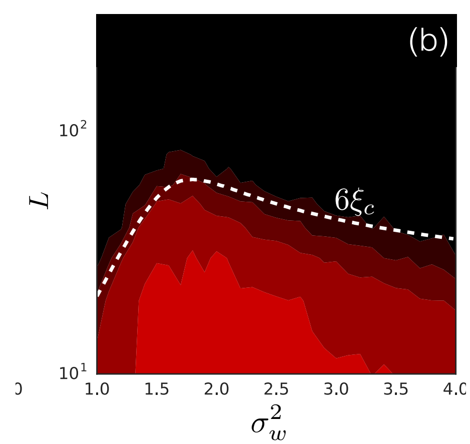
* also initialized with well conditioned Jacobian (aka *dynamical isometry* [Pennington, et al., 2017])

[Xiao, et al., 2018]

Similar GP / phase diagram analyses apply to many architectures

- CNNs
- Fully connected
- **Dropout**
- **ReLUs**
- **ResNets**
- **RNNs**
- **Batch norm**
- **Transformers**
- **Hypernetworks**
- **Quantized networks**
- …

- Infinitesimal amounts of dropout destroys order-to-chaos transition
- Significantly limits maximum trainable depth
- Other noise regularization techniques likely to behave similarly

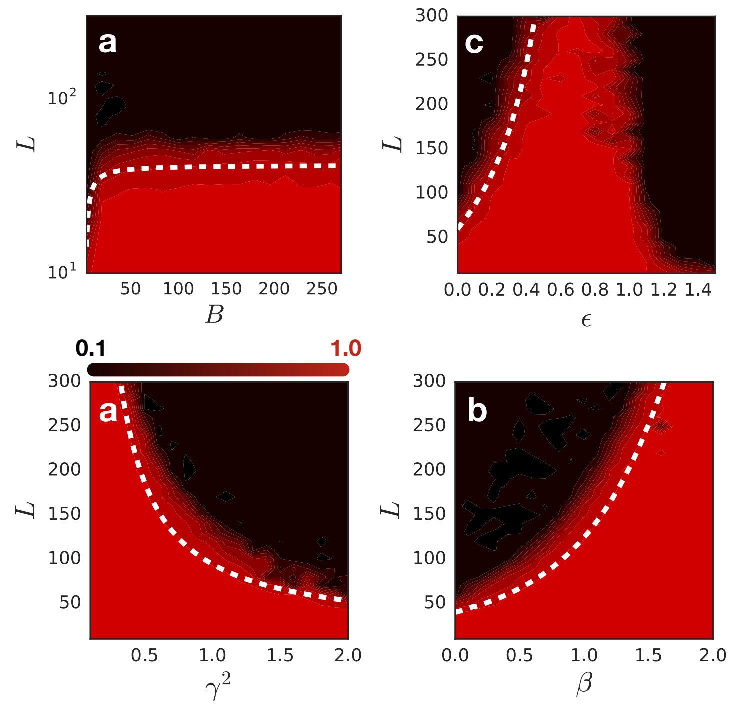


[Schoenholz, et al., 2016]

Similar GP / phase diagram analyses apply to many architectures

- CNNs
- Fully connected
- Dropout
- ReLUs
- ResNets
- RNNs
- **Batch norm**
- **Transformers**
- **Hypernetworks**
- **Quantized networks**
- ...

- Gradients for fully-connected networks with batch normalization explode with depth

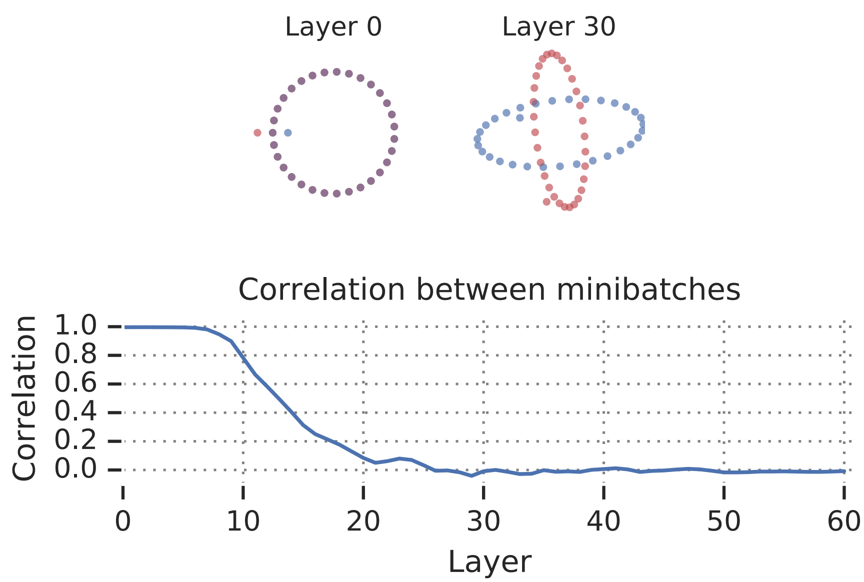


[Yang, et al., 2019]

Similar GP / phase diagram analyses apply to many architectures

- CNNs
- Fully connected
- Dropout
- ReLUs
- ResNets
- RNNs
- **Batch norm**
- **Transformers**
- **Hypernetworks**
- **Quantized networks**
- ...

- Batch norm causes chaos for all hyperparameters
- Two similar minibatches decorrelate when passed through a deep linear network with batch norm



[Yang, et al., 2019]

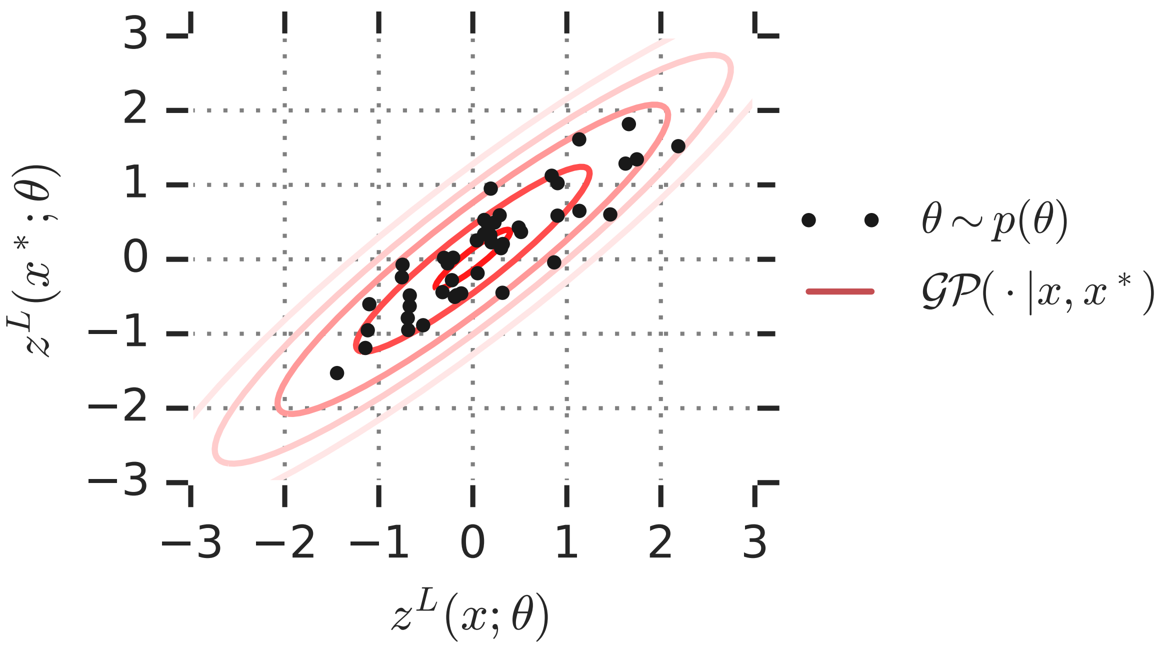
Distribution over functions induced by random initialization of wide neural network is a Gaussian process

- What can we do with this understanding?

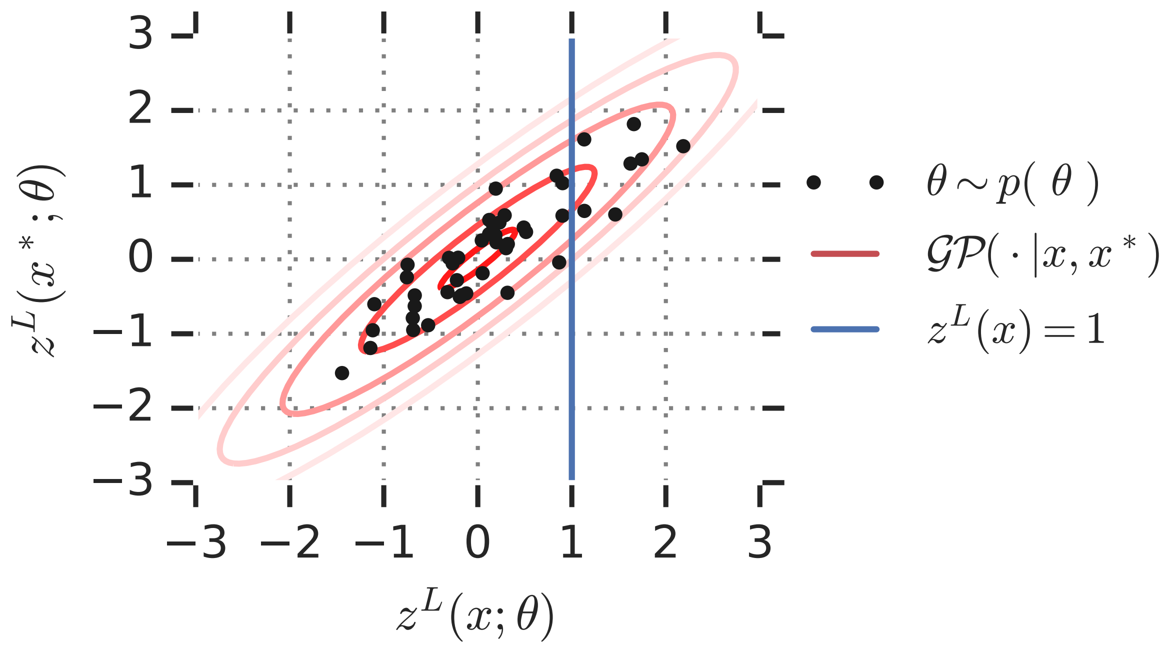
- Predict trainability
  - Random initialization corresponds to the start of training

- **Perform inference with infinitely wide Bayesian neural networks**
  - **(without ever instantiating a neural network)**

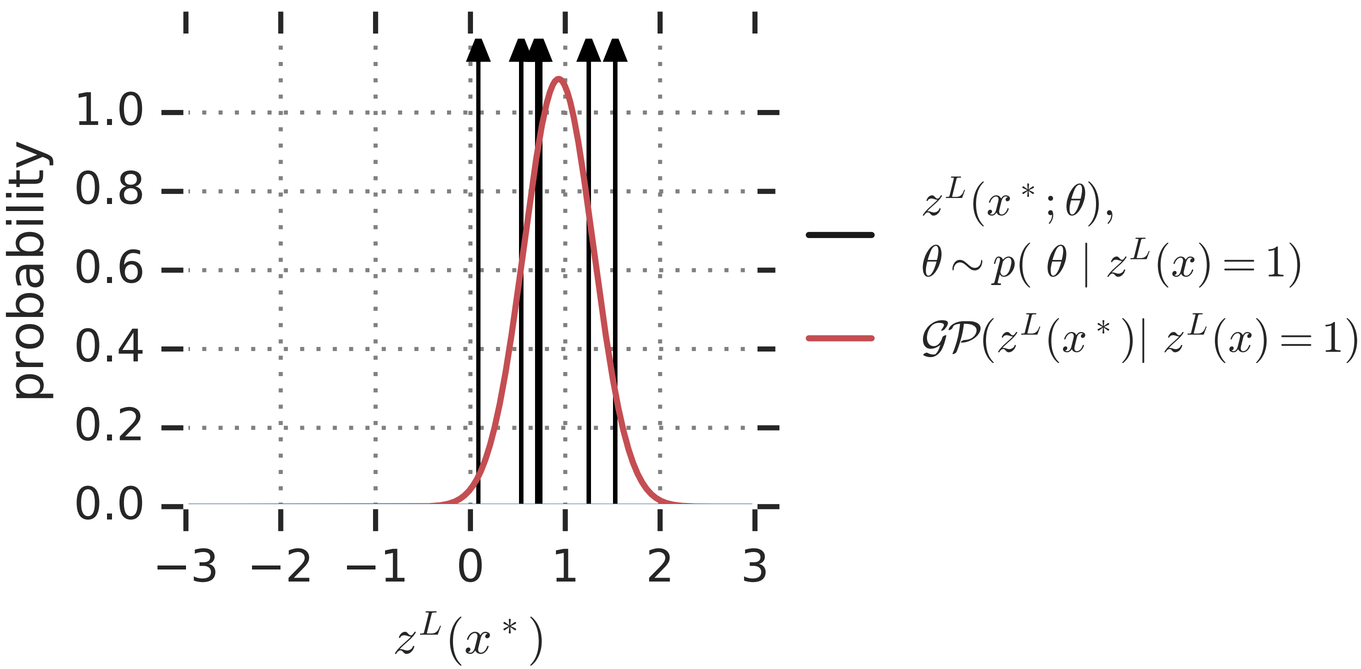
Cartoon illustration of fully Bayesian training



Cartoon illustration of fully Bayesian training



Cartoon illustration of fully Bayesian training



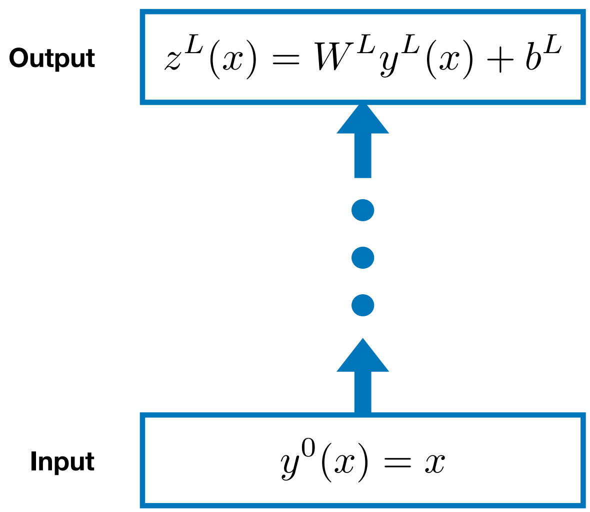
Predictions of fully Bayesian wide NNs correspond to predictions from NNGP posterior

$$\int d\theta \; \underbrace{p\left(\theta \mid X_{\text{train}}, Z^L_{\text{train}}\right)}_{\text{Bayesian posterior over parameters}} p\left(Z^L_{\text{test}} \mid X_{\text{test}}, \theta\right) = \underbrace{p\left(Z^L_{\text{test}} \mid X_{\text{test}}, X_{\text{train}}, Z^L_{\text{train}}\right)}_{\text{Bayesian posterior over test points}} = \underbrace{\mathcal{GP}\left(Z^L_{\text{test}}(X_{\text{test}}) \mid Z^L_{\text{train}}(X_{\text{train}})\right)}_{\text{GP posterior over test points}}$$

[Hron, et al., 2020]

Predictions of fully Bayesian wide NNs correspond to predictions from NNGP posterior

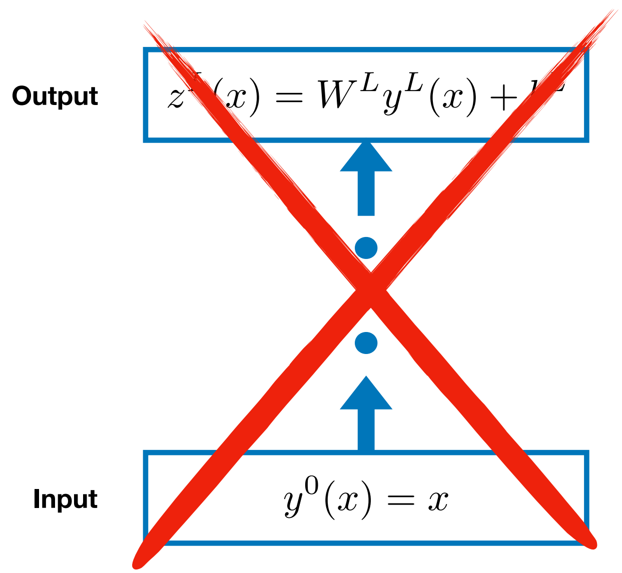
$$\int d\theta \; \underbrace{p\left(\theta \mid X_{\text{train}}, Z^L_{\text{train}}\right)}_{\text{Bayesian posterior over parameters}} p\left(Z^L_{\text{test}} \mid X_{\text{test}}, \theta\right) = \underbrace{p\left(Z^L_{\text{test}} \mid X_{\text{test}}, X_{\text{train}}, Z^L_{\text{train}}\right)}_{\text{Bayesian posterior over test points}} = \underbrace{\mathcal{GP}\left(Z^L_{\text{test}}(X_{\text{test}}) \mid Z^L_{\text{train}}(X_{\text{train}})\right)}_{\text{GP posterior over test points}}$$



[Hron, et al., 2020]

Predictions of fully Bayesian wide NNs correspond to predictions from NNGP posterior

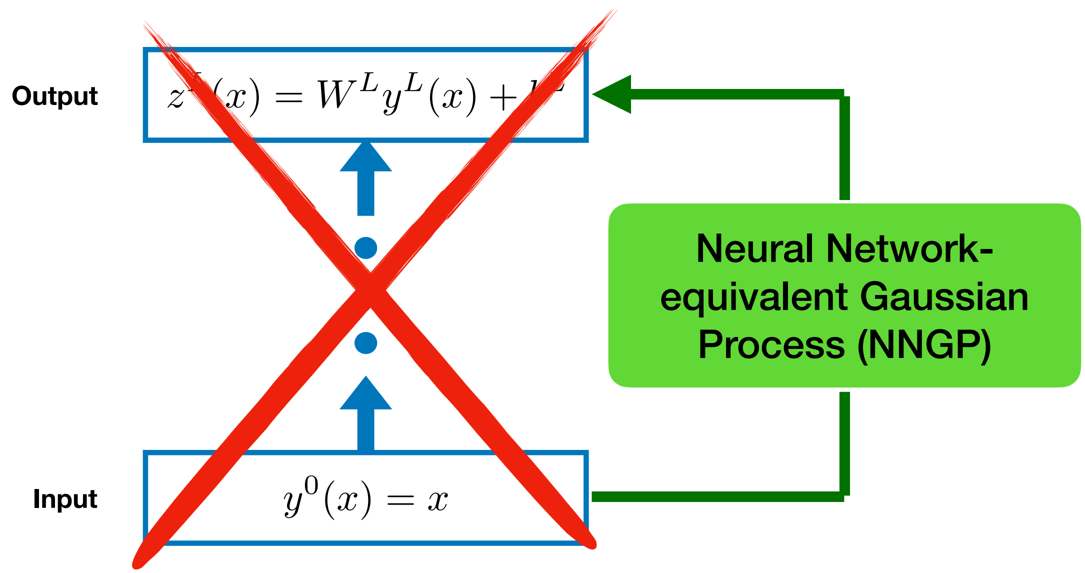
$$\int d\theta\ \underbrace{p\left(\theta \mid X_{\text{train}}, Z^L_{\text{train}}\right)}_{\text{Bayesian posterior over parameters}} p\left(Z^L_{\text{test}} \mid X_{\text{test}}, \theta\right) = \underbrace{p\left(Z^L_{\text{test}} \mid X_{\text{test}}, X_{\text{train}}, Z^L_{\text{train}}\right)}_{\text{Bayesian posterior over test points}} = \underbrace{\mathcal{GP}\left(Z^L_{\text{test}}(X_{\text{test}}) \mid Z^L_{\text{train}}(X_{\text{train}})\right)}_{\text{GP posterior over test points}}$$



[Hron, et al., 2020]

Predictions of fully Bayesian wide NNs correspond to predictions from NNGP posterior

$$\int d\theta\; \underbrace{p\left(\theta \mid X_{\text{train}}, Z^L_{\text{train}}\right)}_{\text{Bayesian posterior over parameters}}\, p\left(Z^L_{\text{test}} \mid X_{\text{test}}, \theta\right) = \underbrace{p\left(Z^L_{\text{test}} \mid X_{\text{test}}, X_{\text{train}}, Z^L_{\text{train}}\right)}_{\text{Bayesian posterior over test points}} = \underbrace{\mathcal{GP}\left(Z^L_{\text{test}}(X_{\text{test}}) \mid Z^L_{\text{train}}(X_{\text{train}})\right)}_{\text{GP posterior over test points}}$$



[Hron, et al., 2020]

Large width behavior of neural networks

	Parameter Space	Function Space
Bayesian Inference		**Neural Network Gaussian Process** $z(x) \sim \mathcal{GP}(z(x))$ $z(x^*)\big\vert\mathcal{D} \sim \mathcal{GP}(z(x^*)\big\vert\mathcal{D})$
Gradient Descent Training		

https://github.com/google/neural-tangents

Large width behavior of neural networks

	Parameter Space	Function Space
Bayesian Inference		**Neural Network Gaussian Process** $z(x) \sim \mathcal{GP}(z(x))$ $z(x^*)\big\vert\mathcal{D} \sim \mathcal{GP}(z(x^*)\big\vert\mathcal{D})$
Gradient Descent Training	$z(x;\theta_t) = z(x;\theta_0) + \nabla_\theta z(x;\theta_0)(\theta_t - \theta_0) + \mathcal{O}\left(\frac{1}{\sqrt{n}}\right)$ **Linearization**	

https://github.com/google/neural-tangents

Large width behavior of neural networks

	Parameter Space	Function Space
Bayesian Inference		**Neural Network Gaussian Process** $z(x) \sim \mathcal{GP}(z(x))$ $z(x^*)\big\vert\mathcal{D} \sim \mathcal{GP}(z(x^*)\big\vert\mathcal{D})$
Gradient Descent Training	$z(x;\theta_t) = z(x;\theta_0) + \nabla_\theta z(x;\theta_0)(\theta_t - \theta_0) + \mathcal{O}\left(\frac{1}{\sqrt{n}}\right)$ **Linearization**	$\partial_t z_t(x) = -\eta\hat{\Theta}\nabla_{z_t}\mathcal{L}(z_t(x))$ $z_t(x^*)\big\vert\mathcal{D} \sim \mathcal{GP}\left(z_t(x^*); \mu_t(\mathcal{D}), \Sigma_t(\mathcal{D})\right)$ **Neural Tangent Kernel**

https://github.com/google/neural-tangents

Wide neural networks are linear in their parameters throughout training

As width n^l goes to infinity, the change due to gradient descent training in any single intermediate layer receptive field or pre-activation goes to 0:

$$\left\|\frac{\partial W^l_{i\bullet}}{\partial t}\right\| \to 0, \quad \left|\frac{\partial z^l_i(x)}{\partial t}\right| \to 0 \quad \text{as } n^l \to \infty, \quad \text{but } \left|\frac{\partial z^L_i(x)}{\partial t}\right| = \mathcal{O}(1)$$

See Neural Tangent Kernel (NTK) [Jacot, et al., 2018] for analogous ideas in function space.

[Lee, Xiao, et al., 2019]

Wide neural networks are linear in their parameters throughout training

As width n^l goes to infinity, the change due to gradient descent training in any single intermediate layer receptive field or pre-activation goes to 0:

$$\left\|\frac{\partial W^l_{i\bullet}}{\partial t}\right\| \to 0, \quad \left|\frac{\partial z^l_i(x)}{\partial t}\right| \to 0 \quad \text{as } n^l \to \infty, \quad \text{but } \left|\frac{\partial z^L_i(x)}{\partial t}\right| = \mathcal{O}(1)$$

This infinitesimal change in intermediate weights and pre-activations allows the network to be replaced by its linearization:

$$z^L(x;\theta) = z^L(x;\theta_0) + \nabla_\theta z^L(x;\theta_0)(\theta_t - \theta_0) + \mathcal{O}\left(\min\left(n^1, \ldots, n^L\right)^{-\frac{1}{2}}\right)$$

See Neural Tangent Kernel (NTK) [Jacot, et al., 2018] for analogous ideas in function space. **[Lee, Xiao, et al., 2019]**

Wide neural networks are linear in their parameters throughout training

As width n^l goes to infinity, the change due to gradient descent training

in any … oes to 0:

$$\left|\left|\frac{\partial}{\quad} \right.\right. \cdots \mathcal{O}(1)$$

- But also see feature learning infinite width limits
  - Mean field networks: **[Mei, Misiakiewicz, Montanari, 2019]**
  - Maximal Update Parameterization: **[Yang, et al., 2022]**
  - Dynamical field theory of NTK: **[Bordelon, Pehlevan, 2023]**

This i… ons

allows …

$$z^L(x;\theta) = z^L(x;\theta_0) + \nabla_\theta z^L(x;\theta_0)(\theta_t - \theta_0) + \mathcal{O}\left(\min\left(n^1,\ldots,n^L\right)^{-\frac{1}{2}}\right)$$

See Neural Tangent Kernel (NTK) [Jacot, et al., 2018] for analogous ideas in function space. **[Lee, Xiao, et al., 2019]**

Wide neural networks are linear in their parameters throughout training

As width n^l goes to infinity, the change due to gradient descent training in any single intermediate layer receptive field or pre-activation goes to 0:

$$\left\|\frac{\partial W^l_{i\bullet}}{\partial t}\right\| \to 0, \quad \left|\frac{\partial z^l_i(x)}{\partial t}\right| \to 0 \quad \text{as } n^l \to \infty, \quad \text{but } \left|\frac{\partial z^L_i(x)}{\partial t}\right| = \mathcal{O}(1)$$

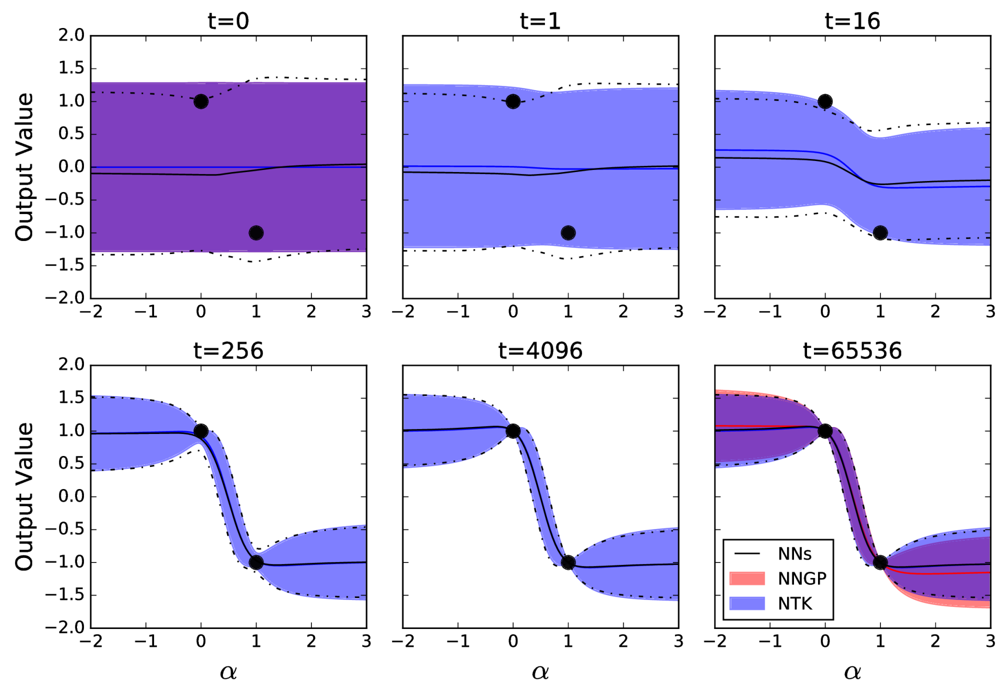
This infinitesimal change in intermediate weights and pre-activations allows the network to be replaced by its linearization:

$$z^L(x;\theta) = z^L(x;\theta_0) + \nabla_\theta z^L(x;\theta_0)(\theta_t - \theta_0) + \mathcal{O}\left(\min\left(n^1, \ldots, n^L\right)^{-\frac{1}{2}}\right)$$

See Neural Tangent Kernel (NTK) [Jacot, et al., 2018] for analogous ideas in function space. **[Lee, Xiao, et al., 2019]**

The functions computed by gradient descent trained wide neural networks are also described by a Gaussian process

- For MSE loss, can solve for distribution over learned functions analytically.
- Corresponds to a GP throughout training.
- Does *not* correspond to Bayesian posterior distribution (ie, there is no prior for which GP is a posterior conditioned on training data)



tanh FC, 3 layer, width 8192, Binary CIFAR, MSE loss, ensemble of 100 networks

[Lee, Xiao, et al., 2019]

Large width behavior of neural networks

	Parameter Space	Function Space
Bayesian Inference		**Neural Network Gaussian Process** $z(x) \sim \mathcal{GP}(z(x))$ $z(x^*)\big\vert\mathcal{D} \sim \mathcal{GP}(z(x^*)\big\vert\mathcal{D})$
Gradient Descent Training	$z(x;\theta_t) = z(x;\theta_0) + \nabla_\theta z(x;\theta_0)(\theta_t - \theta_0) + \mathcal{O}\left(\frac{1}{\sqrt{n}}\right)$ **Linearization**	$\partial_t z_t(x) = -\eta\hat{\Theta}\nabla_{z_t}\mathcal{L}(z_t(x))$ $z_t(x^*)\big\vert\mathcal{D} \sim \mathcal{GP}\left(z_t(x^*); \mu_t(\mathcal{D}), \Sigma_t(\mathcal{D})\right)$ **Neural Tangent Kernel**

https://github.com/google/neural-tangents

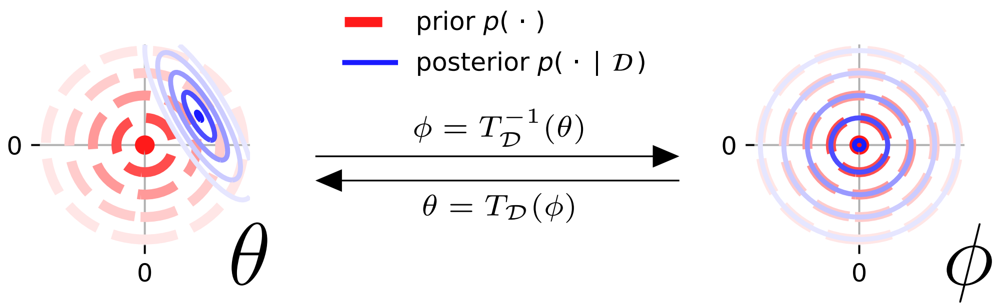
Large width behavior of neural networks

	Parameter Space	Function Space
Bayesian Inference	**Reprioritisation** $$\phi^l := \begin{cases} \Sigma^{-1/2}(\theta^l - \mu) & l = L+1, \\ \theta^l & \text{otherwise.} \end{cases}$$	**Neural Network Gaussian Process** $$z(x) \sim \mathcal{GP}(z(x))$$ $$z(x^*)\big\vert\mathcal{D} \sim \mathcal{GP}(z(x^*)\big\vert\mathcal{D})$$
Gradient Descent Training	$$z(x;\theta_t) = z(x;\theta_0) + \nabla_\theta z(x;\theta_0)(\theta_t - \theta_0) + \mathcal{O}\left(\frac{1}{\sqrt{n}}\right)$$ **Linearization**	$$\partial_t z_t(x) = -\eta\hat{\Theta}\nabla_{z_t}\mathcal{L}(z_t(x))$$ $$z_t(x^*)\big\vert\mathcal{D} \sim \mathcal{GP}\left(z_t(x^*); \mu_t(\mathcal{D}), \Sigma_t(\mathcal{D})\right)$$ **Neural Tangent Kernel**

https://github.com/google/neural-tangents

Repriorisation: Summary

- A data-dependent reparameterization converges the *posterior* over parameters onto the *prior* over parameters, with increasing width



$$\lim_{\min(n^1 \ldots n^L) \to \infty} \mathrm{KL}\left(p_\theta \mid p_{\phi|\mathcal{D}}\right) = 0$$

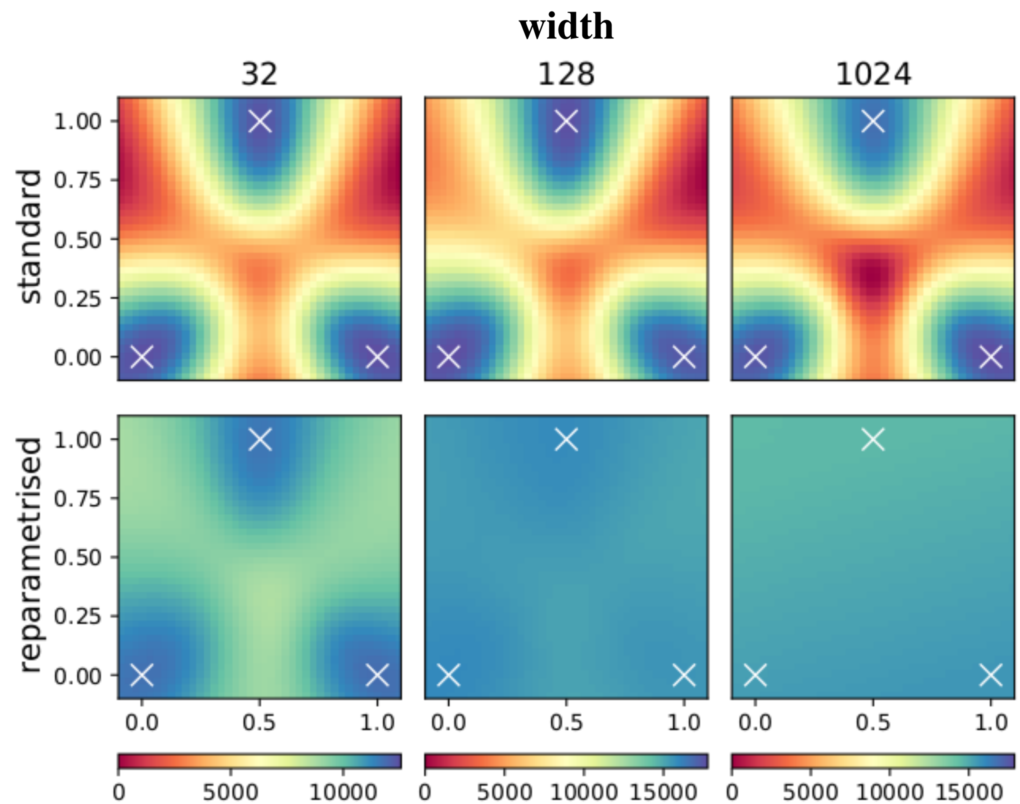
- Theory: Analytic form for Bayesian posterior of wide networks

- Practice: MCMC sampling after reparameterization ("repriorisation") *much faster*, and gets *better with increasing width*

[Hron, et al., 2022]

Repriorisation: Energy landscape far smoother, and easier to sample from, after repriorisation

Log posterior, interpolated between three parameter samples:



[Hron, et al., 2022]

Summary: a powerful framework for understanding neural networks

	Parameter Space	Function Space
Bayesian Inference	**Repriorisation** $\phi^l := \begin{cases} \Sigma^{-1/2}(\theta^l - \mu) & l = L+1, \\ \theta^l & \text{otherwise.} \end{cases}$	**Neural Network Gaussian Process** $z(x) \sim \mathcal{GP}(z(x))$ $z(x^*)\big\vert\mathcal{D} \sim \mathcal{GP}(z(x^*)\big\vert\mathcal{D})$
Gradient Descent Training	$z(x;\theta_t) = z(x;\theta_0) + \nabla_\theta z(x;\theta_0)(\theta_t - \theta_0) + \mathcal{O}\left(\frac{1}{\sqrt{n}}\right)$ **Linearization**	$\partial_t z_t(x) = -\eta\hat{\Theta}\nabla_{z_t}\mathcal{L}(z_t(x))$ $z_t(x^*)\big\vert\mathcal{D} \sim \mathcal{GP}\left(z_t(x^*);\mu_t(\mathcal{D}),\Sigma_t(\mathcal{D})\right)$ **Neural Tangent Kernel**

https://github.com/google/neural-tangents

Summary: a powerful framework for understanding neural networks

	Parameter Space	Function Space
Bayesian Inference	**Repriorisation** $\phi^l :=$	**Neural Network Gaussian Process**
Gradient Descent Training	$z(x;\theta_t) = z$ **Linearization**	$\Sigma_t(\mathcal{D}))$ **Neural Tangent Kernel**

- Predict how trainability depends on architecture and hyperparameters.
- Compute test set predictions without instantiating network.
- Insights into network behavior.
- Better posterior sampling for finite width Bayesian networks.

Summary: a powerful framework for understanding neural networks

	Parameter Space	Function Space
Bayesian Inference	**Repriorisation** $\phi^l := \begin{cases} \Sigma^{-1/2}(\theta^l - \mu) & l = L+1, \\ \theta^l & \text{otherwise.} \end{cases}$	**Neural Network Gaussian Process** $z(x) \sim \mathcal{GP}(z(x))$ $z(x^*)\big\vert\mathcal{D} \sim \mathcal{GP}(z(x^*)\big\vert\mathcal{D})$
Gradient Descent Training	$z(x;\theta_t) = z(x;\theta_0) + \nabla_\theta z(x;\theta_0)(\theta_t - \theta_0) + \mathcal{O}\left(\frac{1}{\sqrt{n}}\right)$ **Linearization**	$\partial_t z_t(x) = -\eta\hat{\Theta}\nabla_{z_t}\mathcal{L}(z_t(x))$ $z_t(x^*)\big\vert\mathcal{D} \sim \mathcal{GP}\left(z_t(x^*); \mu_t(\mathcal{D}), \Sigma_t(\mathcal{D})\right)$ **Neural Tangent Kernel**

https://github.com/google/neural-tangents

SCRAP SLIDES

Postscript: The power of the Neural Tangents library!
https://github.com/google/neural-tangents

- Prior approaches to infinite width experiments
  - Do a lot of algebra by hand
  - Write a lot of bespoke code
  - (think neural networks before automatic differentiation)
- We can do much better!

Postscript: The power of the Neural Tangents library!
https://github.com/google/neural-tangents

Define an infinite width network:

```
from neural_tangents import stax
init_fn, apply_fn, kernel_fn = stax.serial(stax.Dense(2048, W_std=1.5, b_std=0.05), stax.Erf(),
                                           stax.Dense(2048, W_std=1.5, b_std=0.05), stax.Erf(),
                                           stax.Dense(1, W_std=1.5, b_std=0.05))
```

Postscript: The power of the Neural Tangents library!
https://github.com/google/neural-tangents

Define an infinite width network:

```
from neural_tangents import stax
init_fn, apply_fn, kernel_fn = stax.serial(stax.Dense(2048, W_std=1.5, b_std=0.05), stax.Erf(),
                                           stax.Dense(2048, W_std=1.5, b_std=0.05), stax.Erf(),
                                           stax.Dense(1, W_std=1.5, b_std=0.05))
```

Init and forward-pass functions of the finite model

Postscript: The power of the Neural Tangents library!
https://github.com/google/neural-tangents

Define an infinite width network:

```
from neural_tangents import stax
init_fn, apply_fn, kernel_fn = stax.serial(stax.Dense(2048, W_std=1.5, b_std=0.05), stax.Erf(),
                                            stax.Dense(2048, W_std=1.5, b_std=0.05), stax.Erf(),
                                            stax.Dense(1, W_std=1.5, b_std=0.05))
```

Kernel function(s) of the infinite model

$$\mathcal{K}(\cdot,\cdot), \Theta(\cdot,\cdot)$$

Postscript: The power of the Neural Tangents library!
https://github.com/google/neural-tangents

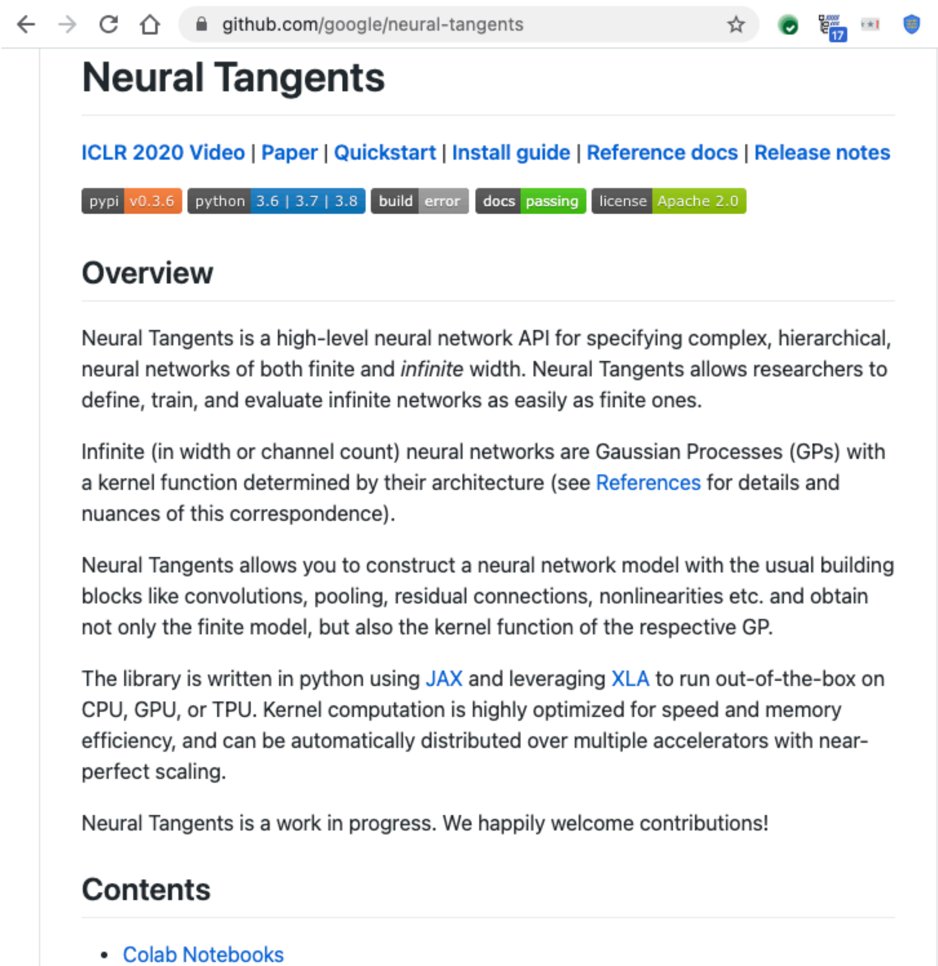
Build a more complex architecture:

An infinitely WideResNet

```
from neural_tangents import stax

def WideResNetBlock(channels, strides=(1, 1), channel_mismatch=False):
  Main = stax.serial(stax.Relu(), stax.Conv(channels, (3, 3), strides, padding='SAME'),
                     stax.Relu(), stax.Conv(channels, (3, 3), padding='SAME'))
  Shortcut = (stax.Identity() if not channel_mismatch else
              stax.Conv(channels, (3, 3), strides, padding='SAME'))
  return stax.serial(stax.FanOut(2), stax.parallel(Main, Shortcut), stax.FanInSum())

def WideResNetGroup(n, channels, strides=(1, 1)):
  blocks = [WideResNetBlock(channels, strides, channel_mismatch=True)]
  for _ in range(n - 1):
    blocks += [WideResNetBlock(channels, (1, 1))]
  return stax.serial(*blocks)

def WideResNet(block_size, k, num_classes):
  return stax.serial(stax.Conv(16, (3, 3), padding='SAME'),
                     WideResNetGroup(block_size, int(16 * k)),
                     WideResNetGroup(block_size, int(32 * k), (2, 2)),
                     WideResNetGroup(block_size, int(64 * k), (2, 2)),
                     stax.GlobalAvgPool(), stax.Dense(num_classes))

init_fn, apply_fn, kernel_fn = WideResNet(block_size=4, k=1, num_classes=10)
```

Postscript: The power of the Neural Tangents library!

https://github.com/google/neural-tangents



github.com/google/neural-tangents

Neural Tangents

ICLR 2020 Video | Paper | Quickstart | Install guide | Reference docs | Release notes

pypi v0.3.6 | python 3.6 | 3.7 | 3.8 | build error | docs passing | license Apache 2.0

Overview

Neural Tangents is a high-level neural network API for specifying complex, hierarchical, neural networks of both finite and *infinite* width. Neural Tangents allows researchers to define, train, and evaluate infinite networks as easily as finite ones.

Infinite (in width or channel count) neural networks are Gaussian Processes (GPs) with a kernel function determined by their architecture (see References for details and nuances of this correspondence).

Neural Tangents allows you to construct a neural network model with the usual building blocks like convolutions, pooling, residual connections, nonlinearities etc. and obtain not only the finite model, but also the kernel function of the respective GP.

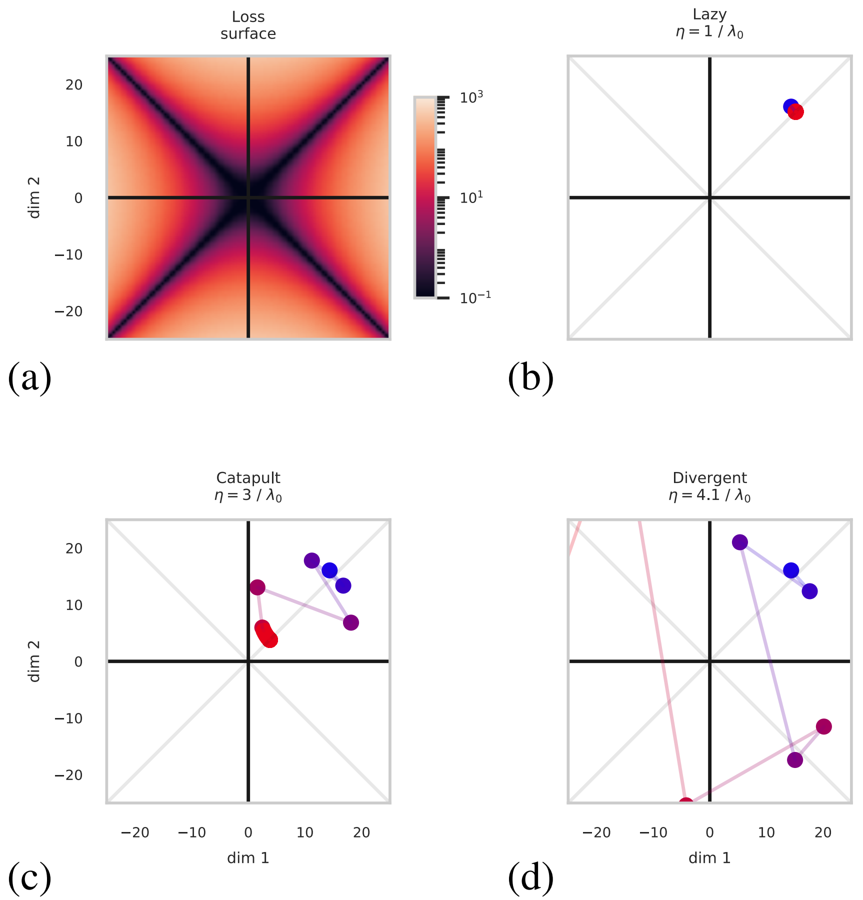
The library is written in python using JAX and leveraging XLA to run out-of-the-box on CPU, GPU, or TPU. Kernel computation is highly optimized for speed and memory efficiency, and can be automatically distributed over multiple accelerators with near-perfect scaling.

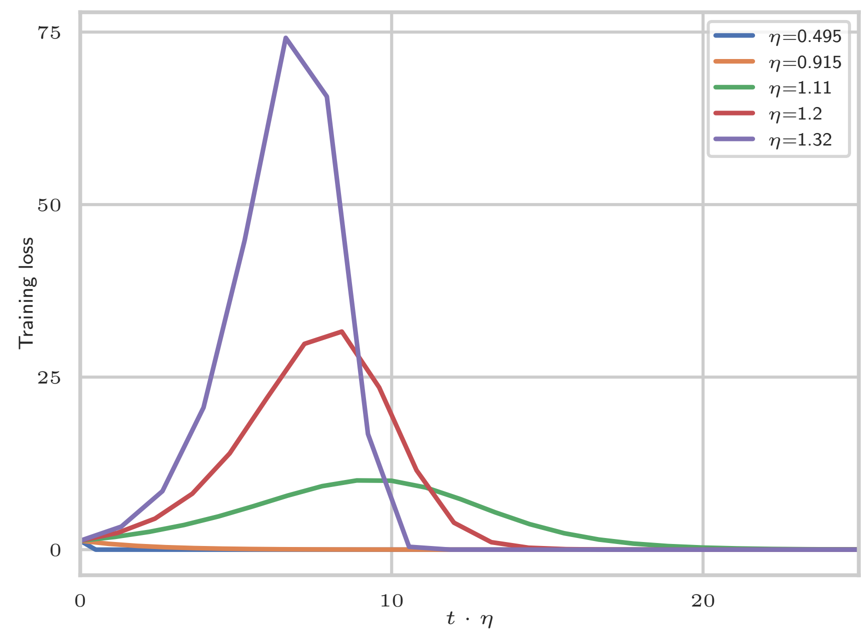
Neural Tangents is a work in progress. We happily welcome contributions!

Contents

- Colab Notebooks

Catapult dynamics — large width without linearization





[Lewkowycz, et al., 2020]