What is Machine Learning?

Chia-Hung Yuan & DataLab

Department of Computer Science,
National Tsing Hua University, Taiwan

Outline

1. Estimating Probability from Data
2. Gaussian Process
3. Information Propagation
4. Neural Networks as Gaussian Process
5. Neural Tangent Kernel

1

Estimating Probability from Data

Estimating Probability from Data

- Image we are playing a coin flipping game, and we want to estimate the **probability** to get a “head”

- How?

Estimating Probability from Data

- To estimate the probability, the simplest way is to do so by counting. For example,

H T T H H H T T T T

$$\Rightarrow \quad P(H) \approx \frac{n_H}{n_H + n_T}$$

- Here we assume there is a **ground truth distribution** $P(X, Y)$, otherwise it's impossible to do any machine learning

Maximum Likelihood Estimation (MLE)

$$P(D;\theta),\ \theta = \arg\max_{\theta} P(D;\theta)$$ Likelihood

H T T H H H T T T T

1. Binomial distribution $P(D;\theta) = \binom{n_H + n_T}{n_H}\theta^{n_H}(1-\theta)^{n_T}$

2. Solve θ by MLE:

   $$\theta = \arg\max_{\theta} \log\binom{n_H + n_T}{n_H}\theta^{n_H}(1-\theta)^{n_T},\ \frac{\partial}{\partial\theta} = 0$$

3. $$\theta = \frac{n_H}{n_H + n_T}$$

Maximum Likelihood Estimation (MLE)

- However, MLE estimates the probability distribution purely from data

- If we only have small amount of data, MLE might not work. For example,

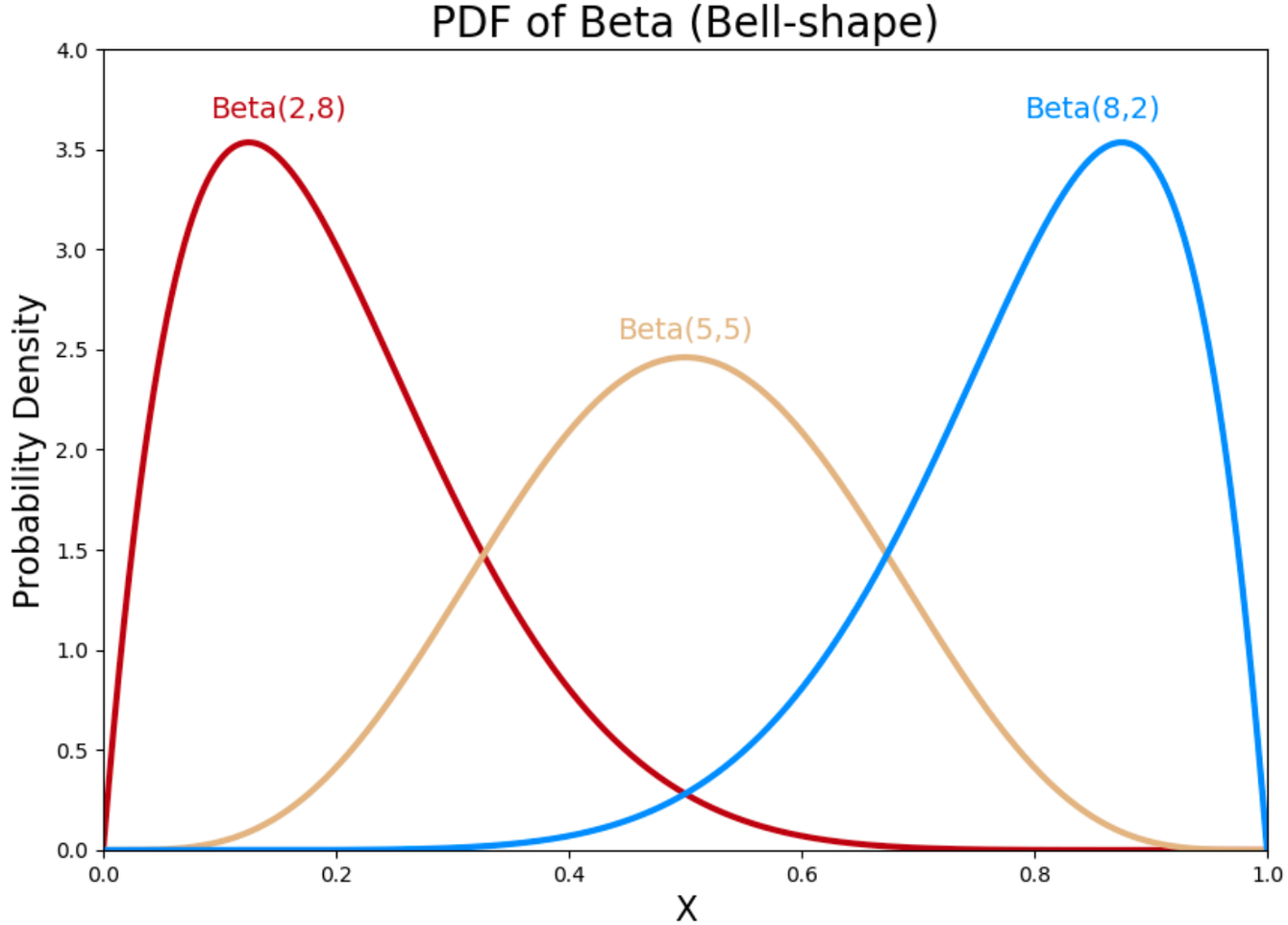
T T T T or H H H H

- Let’s introduce so called “prior knowledge”

Prior

- Based on our understanding, coin flipping can hardly always get hear or tail

- This is our “prior knowledge”, and we can model it by beta distribution



Bayes' Theorem

Likelihood Prior

Posterior $$P(\theta \mid D) = \frac{P(D \mid \theta) P(\theta)}{P(D)}$$

Maximum A Posterior Estimation (MAP)

H T T H H H T T T T

Likelihood Prior

Posterior $P(\theta|D) \propto P(D|\theta)P(\theta)$

$$P(D|\theta) \sim Bin,\ P(\theta) = \frac{\theta^{\alpha-1}(1-\theta)^{\beta-1}}{B(\alpha,\beta)}$$

$$P(\theta|D) \propto P(D|\theta)P(\theta) \propto \binom{n_H+n_T}{n_H}\theta^{n_H}(1-\theta)^{n_T}\theta^{\alpha-1}(1-\theta)^{\beta-1}$$

- Analogously, we can solve $\theta = \frac{n_H + \boxed{\alpha - 1}}{n_H + n_T + \boxed{\alpha + \beta - 2}}$

Smoothing

Maximum Likelihood Estimation (MLE)

- Proper prior causes MAP to converge faster, while wrong prior makes it require more data to correct the distribution

- Is prior reasonable?

Human do have prior

MLE, MAP and Bayesian Inference

- Given the observed data D, estimations of a probabilistic model's parameter θ by MLE and MAP are

$$\hat{\theta}_{MLE} = \arg\max_{\theta} P(D \mid \theta)$$

$$\begin{aligned}\hat{\theta}_{MAP} &= \arg\max_{\theta} P(\theta \mid D) \\ &= \arg\max_{\theta} \frac{P(D \mid \theta)P(\theta)}{P(D)} \\ &= \arg\max_{\theta} P(D \mid \theta)P(\theta)\end{aligned}$$

- Both methods give us a single fixed value, thus they're considered as **point estimators**

MLE, MAP and Bayesian Inference

- On the other hand, Bayesian inference fully calculate the posterior probability distribution

$$P(\theta \mid D) = \frac{P(D \mid \theta)P(\theta)}{P(D)}$$

- MLE and MAP returns a **single fixed value**, but Bayesian inference returns **probability density/mass function**

Why Bayesian?

- Assume we are in casino, and we heard the rumor that there's one special slot machine with 67% winning probability

Why Bayesian?

- After observing for a while, we notice that there are 2 suspicious candidates, where we collected the following data:

Machine A: 3 wins out of 4 plays
Machine B: 81 wins out of 121 plays

- By intuition, we would think *machine B* is the special one

Why Bayesian?

Machine A: 3 wins out of 4 plays
Machine B: 81 wins out of 121 plays

- However, if we estimate winning probabilities by MAP with hyperparameters $\alpha = \beta = 2$, we will get the surprising results

- Recall that $\hat{\theta}_{MAP} = \dfrac{k + \alpha - 1}{n + \alpha + \beta - 2}$

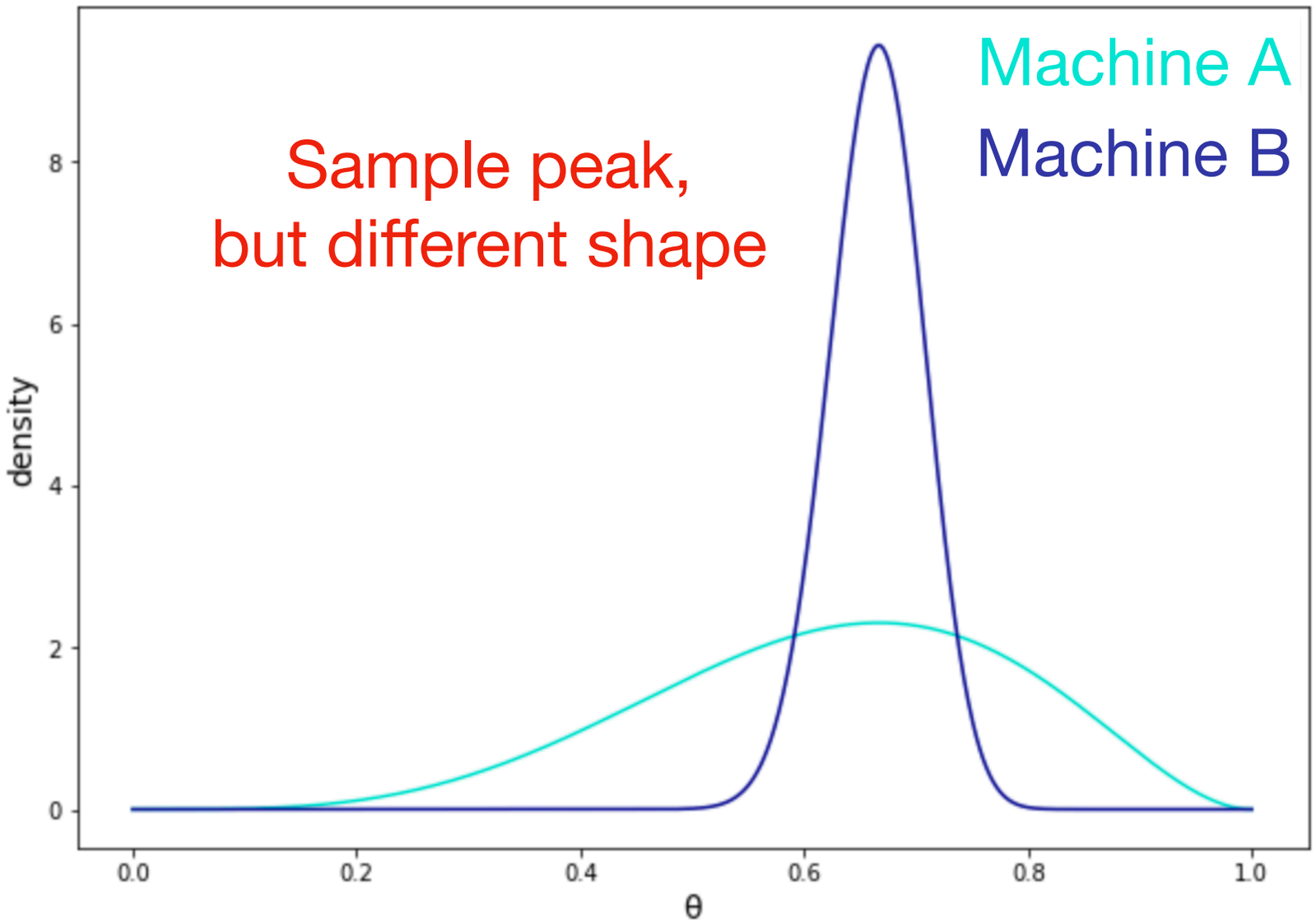
We cannot determine which one is the special slot machine

Machine A: $(3 + 2 - 1)/(4 + 2 + 2 - 2) = \mathbf{66.7\%}$
Machine B: $(81 + 2 - 1)/(121 + 2 + 2 - 2) = \mathbf{66.7\%}$

Bayesian Inference

- Let's fully calculate the posterior probability distribution

$$P(\theta \mid D) = \frac{\Gamma(n+\alpha+\beta)}{\Gamma(k+\alpha)\Gamma(n-k+\beta)}\theta^{k+\alpha-1}(1-\theta)^{n-k+\beta-1}$$



Bayesian Inference

- Both MAP and Bayesian inference are based on Bayers' theorem

- However, for Bayesian inference, we need to calculate $P(D)$ called **marginal likelihood** or **evidence**

Likelihood Prior

Posterior $$P(\theta \mid D) = \frac{P(D \mid \theta)P(\theta)}{P(D)}$$

Evidence

Bayesian Inference

- $P(D)$ is obtained by marginalization of joint probability

$$P(D) = \int_{\theta} P(D, \theta) d\theta = \int_{\theta} P(D \mid \theta) P(\theta) d\theta$$

- Now, put this into original formula of posterior probability distribution

$$P(\theta \mid D) = \frac{P(D \mid \theta) P(\theta)}{P(D)} = \frac{P(D \mid \theta) P(\theta)}{\int_{\theta} P(D \mid \theta) P(\theta) d\theta}$$

Bayesian Inference

- We have already calculated $P(D \mid \theta)$ and $P(\theta)$ earlier, let's focus on $P(D)$

$$P(D) = \int_{\theta} P(D \mid \theta) P(\theta) d\theta$$

$$= \int_0^1 \binom{n}{k} \theta^k (1-\theta)^{n-k} \frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)} \theta^{\alpha-1} (1-\theta)^{\beta-1} d\theta$$

$$= \binom{n}{k} \frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)} \int_0^1 \theta^{k+\alpha-1} (1-\theta)^{n-k+\beta-1} d\theta$$

Bayesian Inference

- With Euler integral of the first kind, the above formula can be deformed to

$$P(D) = \binom{n}{k} \frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)} \int_0^1 \theta^{k+\alpha-1}(1-\theta)^{n-k+\beta-1} d\theta$$

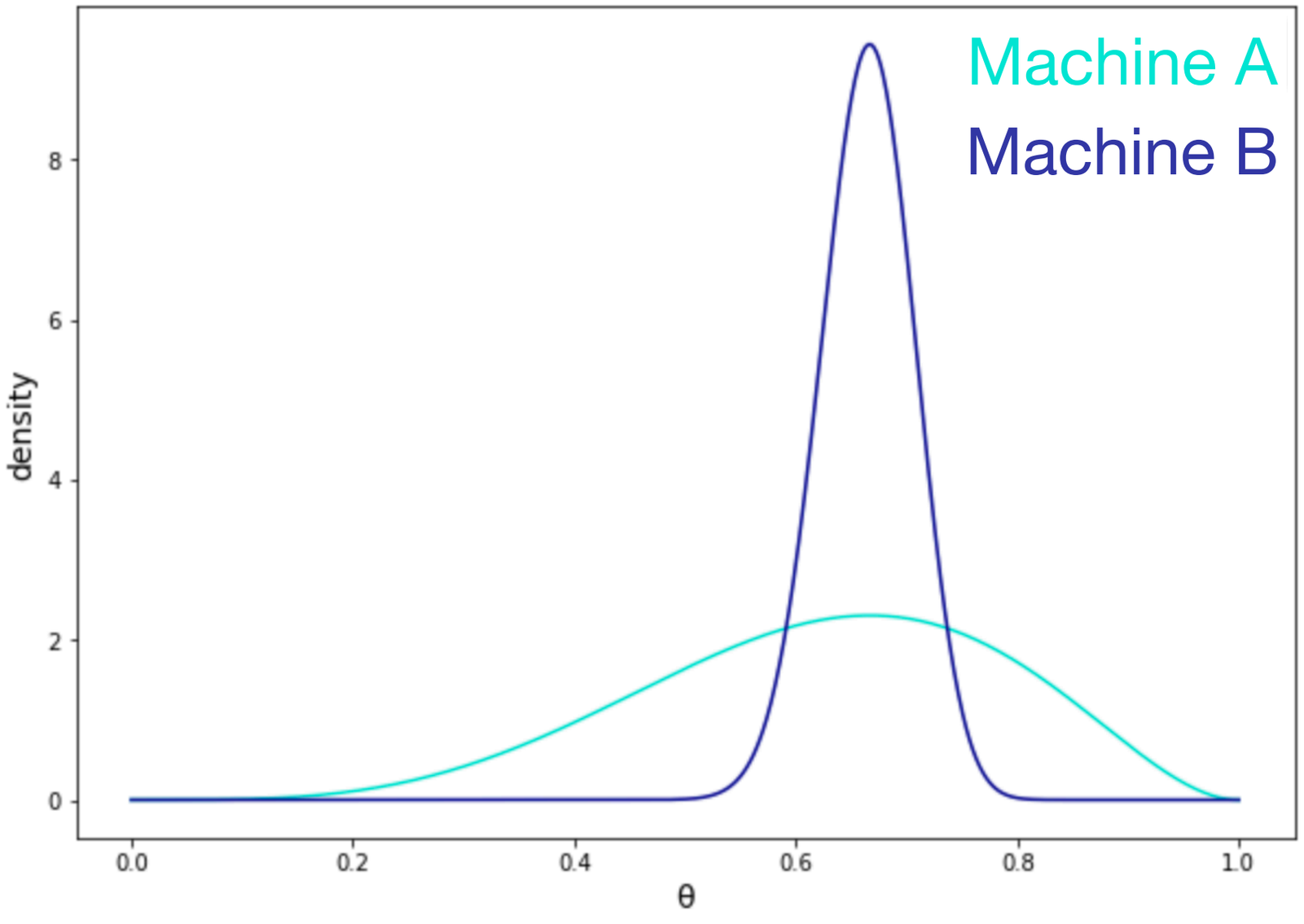
$$= \binom{n}{k} \frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)} \frac{\Gamma(k+\alpha)\Gamma(n-k+\beta)}{\Gamma(n+\alpha+\beta)}$$

- Finally, we can obtain $P(\theta \mid D)$ as below

$$P(\theta \mid D) = \frac{\Gamma(n+\alpha+\beta)}{\Gamma(k+\alpha)\Gamma(n-k+\beta)} \theta^{k+\alpha-1}(1-\theta)^{n-k+\beta-1}$$

Bayesian Inference

$$P(\theta \mid D) = \frac{\Gamma(n+\alpha+\beta)}{\Gamma(k+\alpha)\Gamma(n-k+\beta)}\theta^{k+\alpha-1}(1-\theta)^{n-k+\beta-1}$$



Bayesian Inference

- Bayesian provides us the tools to update our beliefs based on new data

Likelihood Prior

Posterior $$P(\theta \mid D) = \frac{P(D \mid \theta)P(\theta)}{P(D)}$$

- Posterior can turns into the prior of next time step

Recap on Deep Learning

- Given the observed data $D = \{(x_1, y_1), \cdots, (x_n, y_n)\}$, estimations of model's weights w by MLE, MAP and Bayesian are

Here is our beloved Deep Learning!

$$\text{MLE: } P(D \mid w) = \Pi_{i=1}^{n} P(y_i \mid x_i; w)$$

$$\text{MAP: } P(w \mid D) \propto P(D \mid w)P(w)$$

$$\text{Bayesian: } P(w \mid D) = \frac{P(D \mid w)P(w)}{P(D)}$$

- If model's weights w are deterministic, we can make a prediction by $y_t = w^{\intercal} x_t$

Bayesian Learning

- However, Bayesian inference returns the distribution on weights

- How to make a prediction based on the distribution?

$$p(y \mid x, D) = \int_w P(y \mid x, w) P(w \mid D) dw$$

Consider all possible models

- Ensemble of deep neural networks is an approximation to Bayesian networks

Bayesian Inference

- As seen above, Bayesian inference provides much more information than point estimators like MLE and MAP

- However, the complexity of its integral computation is quite challenging

  - In the real-world application, the integral computation might not be possible to solve analytically

  - We then need to use MCMC (Markov Chain Monte Carlo) or other algorithms as a substitution for the direct integral

- So far, we knew the mechanism of MLE, MAP and bayesian inference. Let’s move on to the next stage

2

Gaussian Process

Gaussian Process

- How does machine learning work?

Data D → Weights w → Prediction $y = w^\intercal x$

- Why don't we model the prediction directly?

$$\boxed{p(y \mid x, D)} = \int_w P(y \mid x, w) P(w \mid D) dw = \int_w P(y \mid x, D, w) dw$$

Gaussian Process

$$p(y_t \mid x_t, D) = \int_w P(y_t \mid x_t, w) P(w \mid D) dw = \int_w P(y \mid x, D, w) dw$$

$f(x) = w^\intercal x + \epsilon$ → $P(y \mid x; w) \sim N(w^\intercal x, \epsilon^2 I)$ → Gaussian!

$P(D \mid w) P(w)$ → $P(D \mid w) = \Pi_{i=1}^{n} P(y_i \mid x_i; w)$ → Gaussian!

- If the prior $P(w)$ we choose is also Gaussian, then $p(y_t \mid x_t, D)$ is Gaussian!

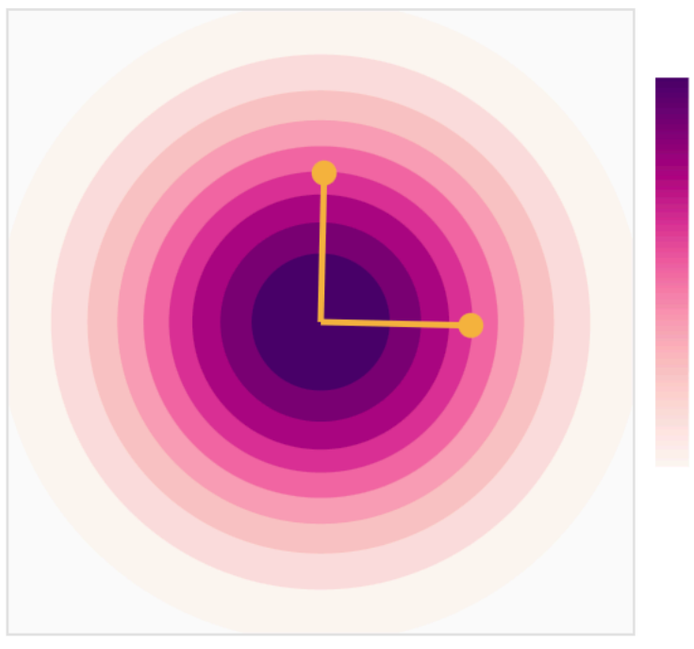
- Therefore, we can model the prediction directly, and this is so called Gaussian Process (GP)

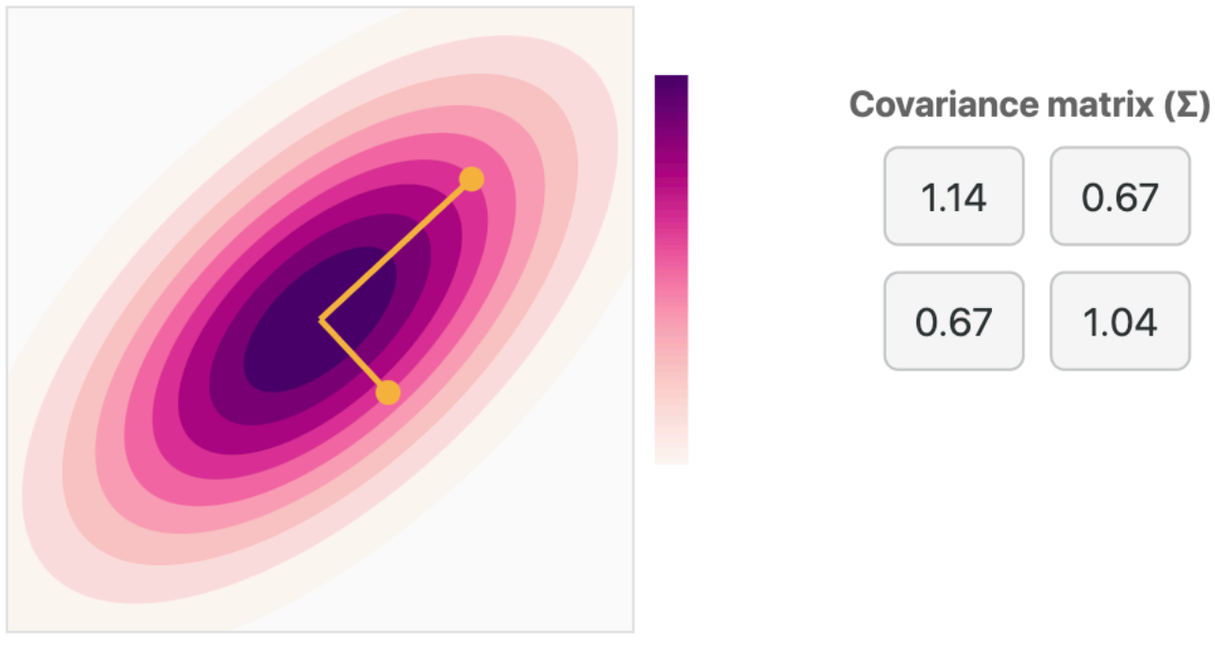
$$P(y \mid x; w) \sim N(\mu, \Sigma)$$

Gaussian Process

$$P(y \mid x, D) \sim N(\mu, \Sigma) \quad \rightarrow \quad P(\begin{bmatrix} y_1 \\ \vdots \\ y_n \\ y \end{bmatrix} \mid x, [x_1, \cdots, x_n]) \sim N(\mu, \Sigma)$$

- Σ is covariance matrix. We assume test data is highly correlated with training data





Gaussian Process

- For example, I am going to use Gaussian Process to make a prediction on house pricing

$$P(y \mid x, D) \sim N(\mu, \Sigma)$$

μ = average house price

$\Sigma = ?$

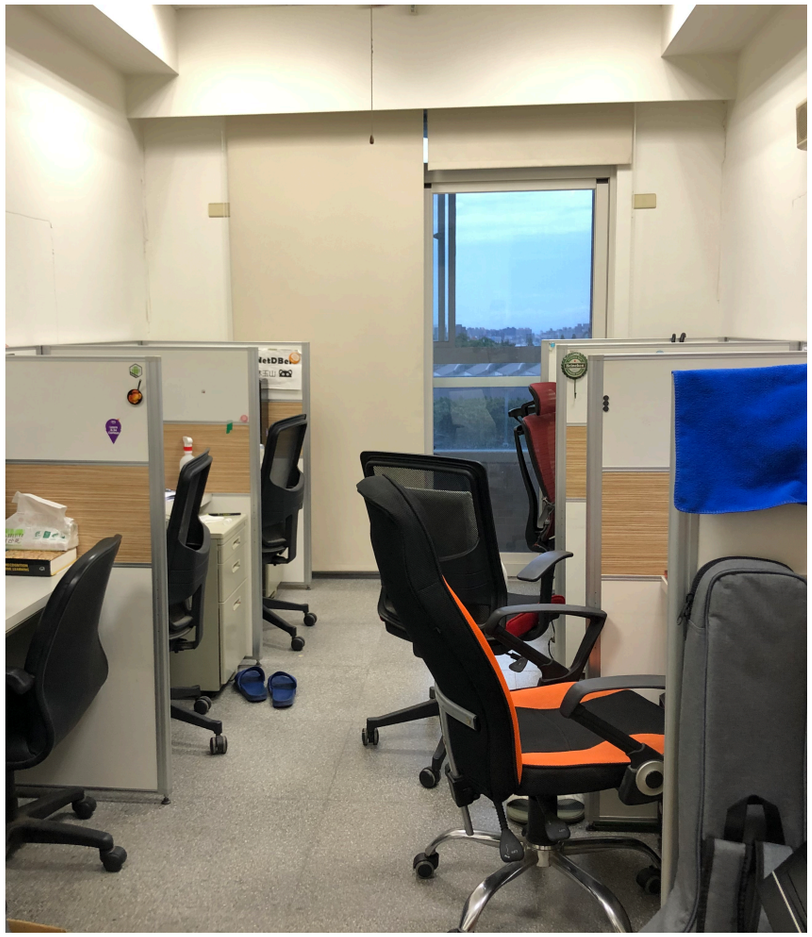
Correlation

$$\Sigma = \begin{bmatrix} 1 & 0.9 & 0 \\ 0.9 & 1 & 0 \\ 0 & 0 & 10 \end{bmatrix}$$

My house
Neighbors’
Moris Chang’s

Variance on
each training point

My house



Neighbors’

Moris Chang’s

Gaussian Process

- For example, I am going to use Gaussian Process to make a prediction on house pricing

$$P(y \mid x, D) \sim N(\mu, \Sigma)$$

μ = average house price

$$\Sigma = \begin{bmatrix} 1 & 0.9 & 0 \\ 0.9 & 1 & 0 \\ 0 & 0 & 10 \end{bmatrix}$$

Correlation

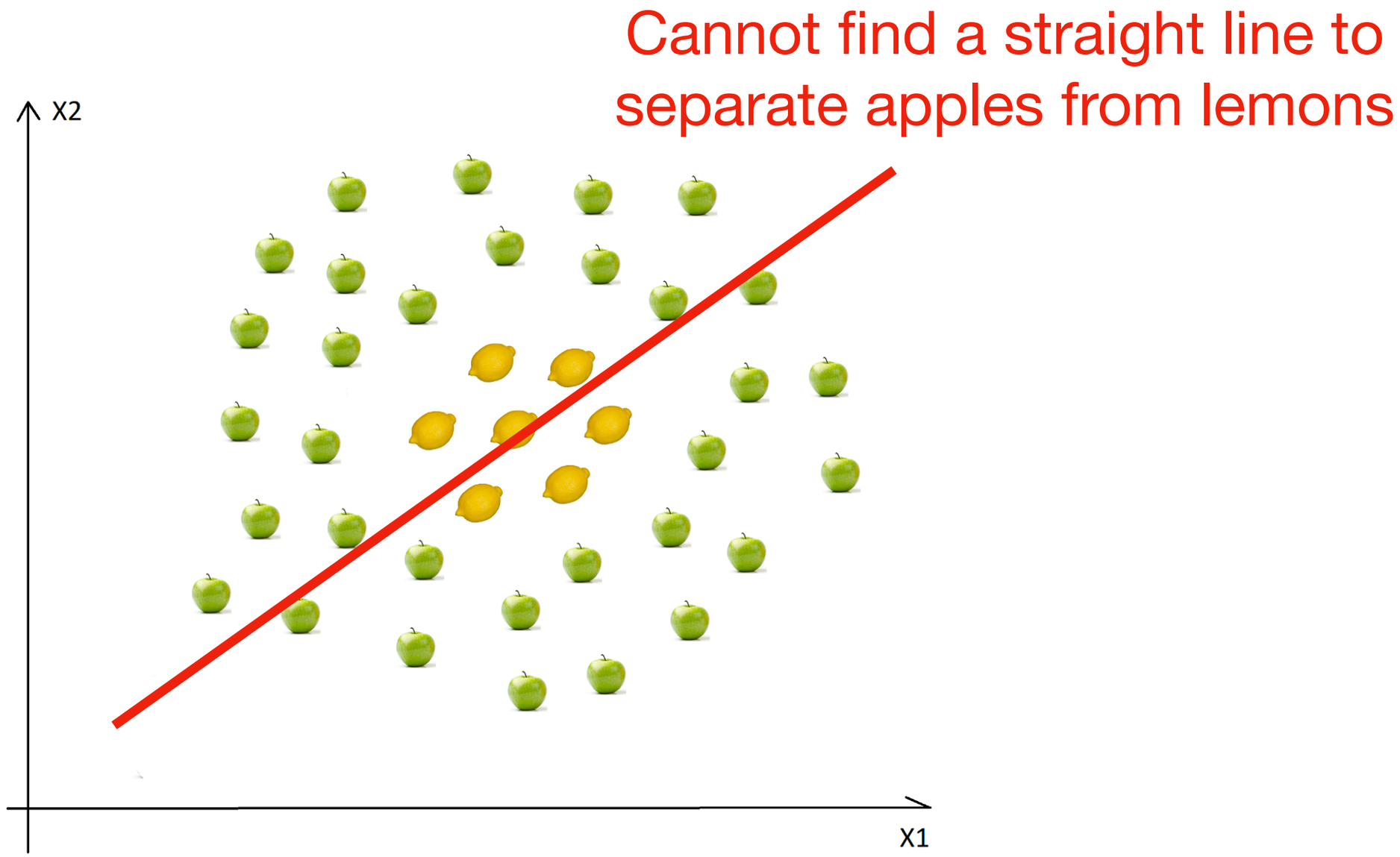
My house
Neighbors'
Moris Chang's

Variance on each training point

- Σ is positive semi-definite, thus we can represent it with **kernel function** K (measure similarity)

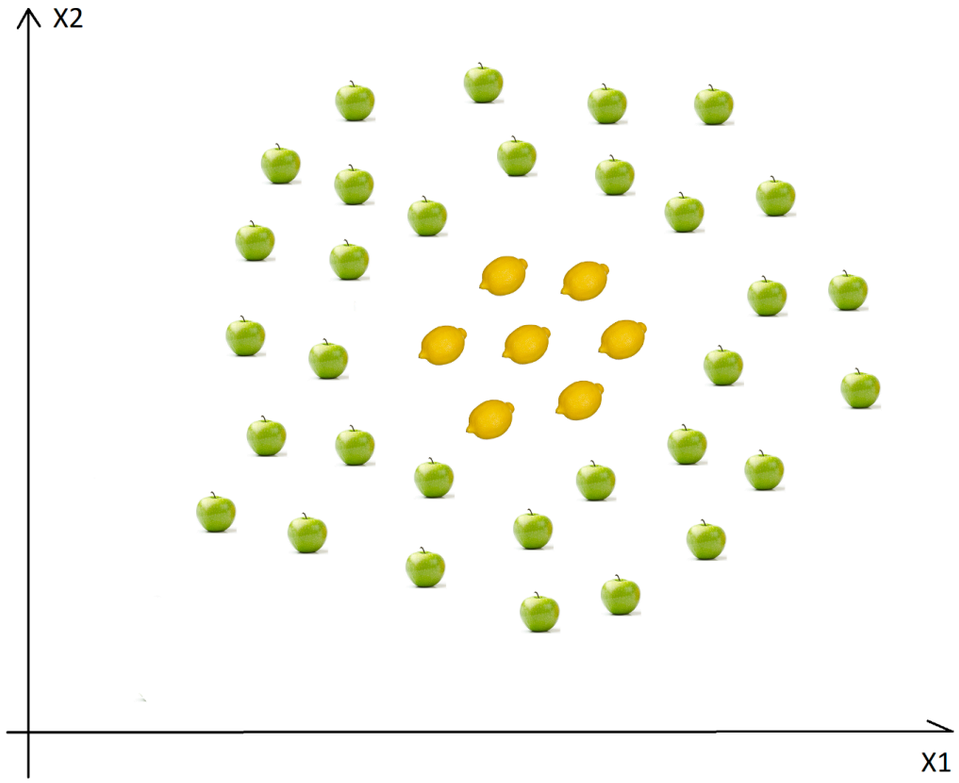
Kernel Function

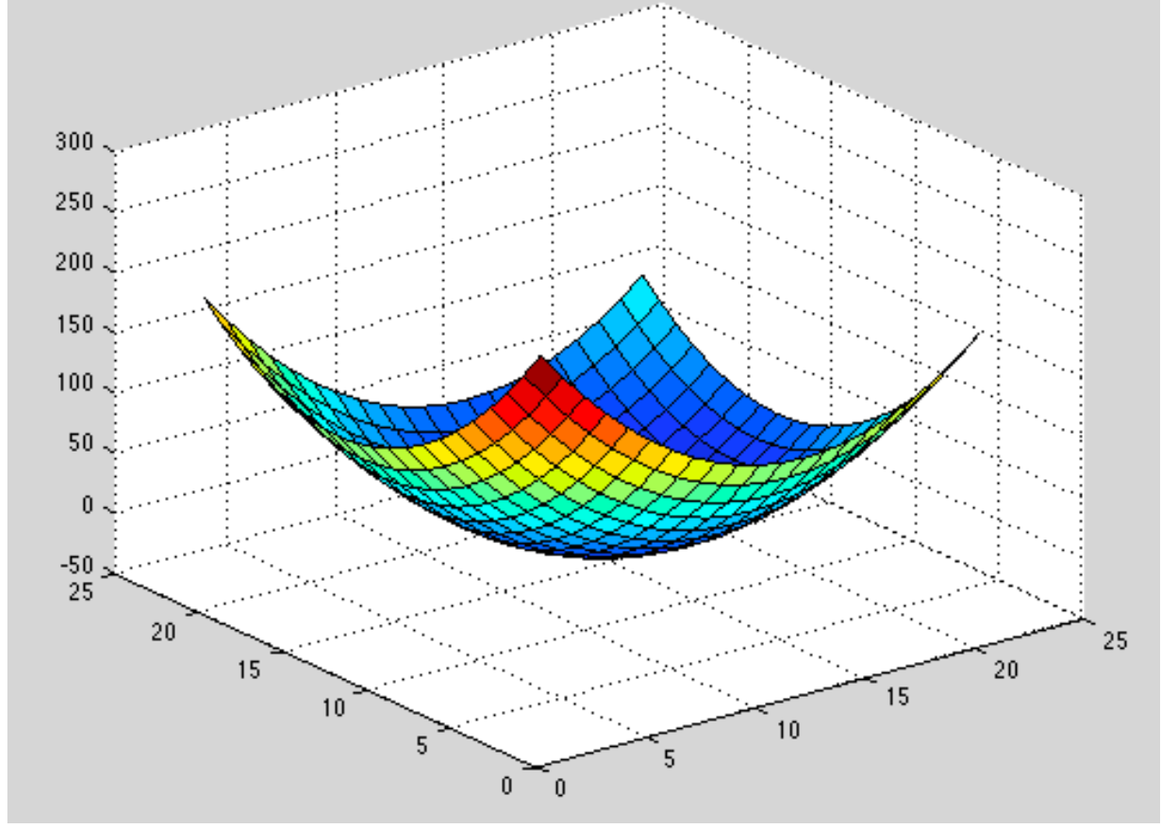
- It is a method of using linear classifier to solve a non-linear problem



Kernel Function

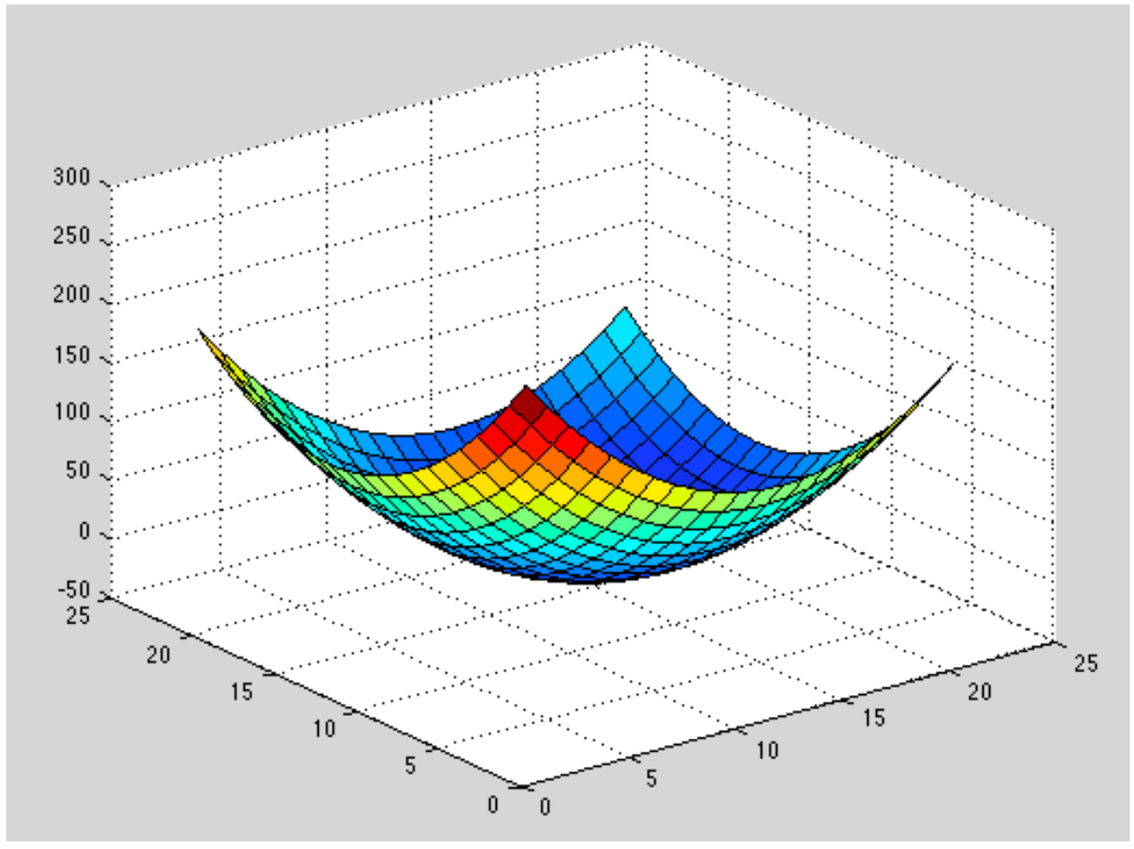
- When data are not linearly separable, we can
    1. Add more features
    2. Use linear classifier

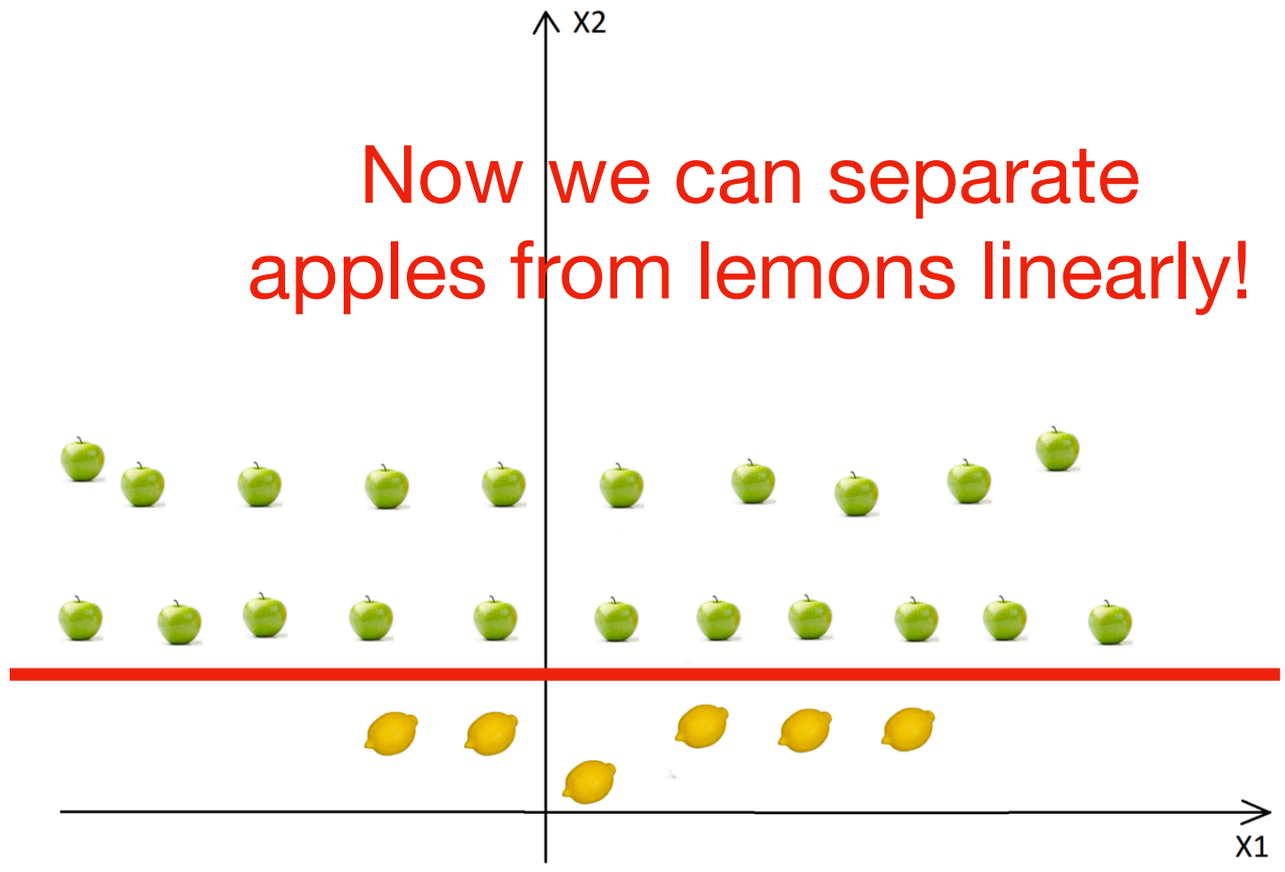




Kernel Function

- When data are not linearly separable, we can
  1. Add more features
  2. Use linear classifier





Kernel Function

- Kernel function can measure similarity between data in higher dimensional space

- Meanwhile it can save a lot of computations

Kernel Function

$$x = (x_1, x_2, x_3); y = (y_1, y_2, y_3)$$

$$f(x) = (x_1x_1, x_1x_2, x_1x_3, x_2x_1, x_2x_2, x_2x_3, x_3x_1, x_3x_2, x_3x_3)$$

- If we want to measure the similarity $< f(x), f(y) >$, we have to first calculate $f(x)$ and $f(y)$

- Suppose $x = (1,2,3); y = (4,5,6)$, then

  $f(x) = (1,2,3,2,4,6,3,6,9)$

  $f(y) = (16,20,24,20,25,30,24,30,36)$

$$< f(x), f(y) > = 16 + 40 + 72 + 40 + 100 + 180 + 72 + 180 + 324$$
$$= 1024$$

Kernel Function

- A lot of algebra, mainly because f is a mapping from 3-dimensional to 9 dimensional space

- If we use kernel $K(x, y) = (< x, y >)^2$, then

$$K(x, y) = (4 + 10 + 18)^2 = 32^2 = \boxed{1024}$$

Same results, but much faster

$$< f(x), f(y) > = 16 + 40 + 72 + 40 + 100 + 180 + 72 + 180 + 324$$
$$= \boxed{1024}$$

Kernel Function

- There are several types of kernel
  - Linear kernel: $k(x, z) = x^\intercal z$
  - Polynomial kernel: $k(x, z) = (1 + x^\intercal z)^P$
  - **RBF kernel:** $k(x, z) = e^{-\frac{\|x - z\|^2}{\sigma^2}}$

Gaussian Process

$$P(\begin{bmatrix} y_1 \\ \vdots \\ y_n \\ y \end{bmatrix} | x, [x_1, \cdots, x_n]) \sim N(0, \Sigma)$$

Kernel as covariance

$$\boxed{\Sigma = K} \qquad k_{ij} = K(x_i, x_j)$$

Train Test v.s. Train

$$\Sigma = \begin{bmatrix} K & K_* \\ K_*^\top & K_{**} \end{bmatrix}$$

Test v.s. Train Test

How to get $P(y | x, y_{1\ldots n}, x_{1\ldots n})$ from $P(\begin{bmatrix} y_1 \\ \vdots \\ y_n \\ y \end{bmatrix} | x, [x_1, \cdots, x_n])$?

Properties of Gaussian distribution

- Let Gaussian random variable $y = \begin{bmatrix} y_A \\ y_B \end{bmatrix}$, mean $\mu = \begin{bmatrix} \mu_A \\ \mu_B \end{bmatrix}$ and covariance matrix $\Sigma = \begin{bmatrix} \Sigma_{AA} & \Sigma_{AB} \\ \Sigma_{BA} & \Sigma_{BB} \end{bmatrix}$
- The conditional probability is also Gaussian and can be calculated as below

$$P(y_A \mid y_B) = \frac{P(y_A, y_B; \mu, \Sigma)}{\int_{y_A} P(y_A, y_B; \mu, \Sigma) dy_A}$$

$$\sim N(\mu_A + \Sigma_{AB}\Sigma_{BB}^{-1}(y_B - \mu_B), \Sigma_{AA} - \Sigma_{AB}\Sigma_{BB}^{-1}\Sigma_{BA})$$

Gaussian Process

- Therefore, we can calculate $P(y \mid x, y_{1\ldots n}, x_{1\ldots n})$ precisely, which is

Kernel regression (prediction) — Variance (uncertainty)

$$P(y \mid x, y_{1\ldots n}, x_{1\ldots n}) \sim N(K_*^\intercal K^{-1} Y, K_{**} - K_*^\intercal K^{-1} K_*)$$

Train — Test v.s. Train

$$\Sigma = \begin{bmatrix} K & K_* \\ K_*^\top & K_{**} \end{bmatrix}$$

Test v.s. Train — Test

- Kernel regression only makes prediction, while Gaussian Process also tells **uncertainty**

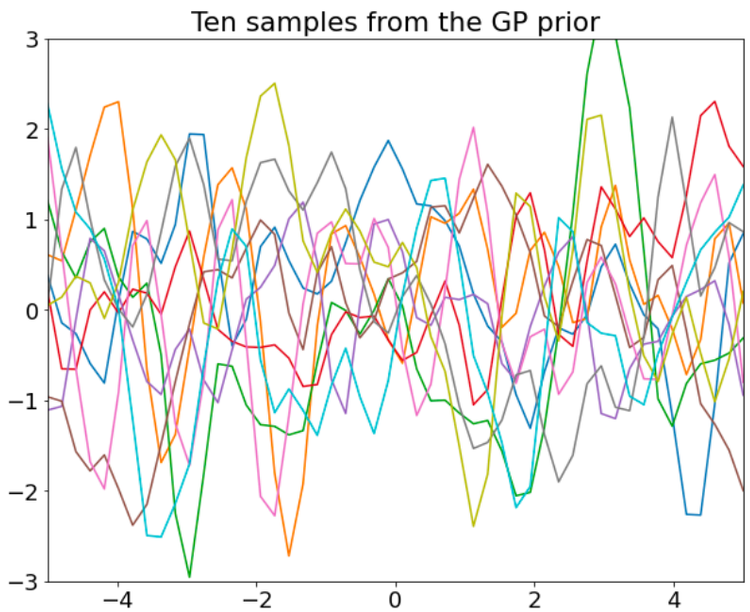
GP Prior

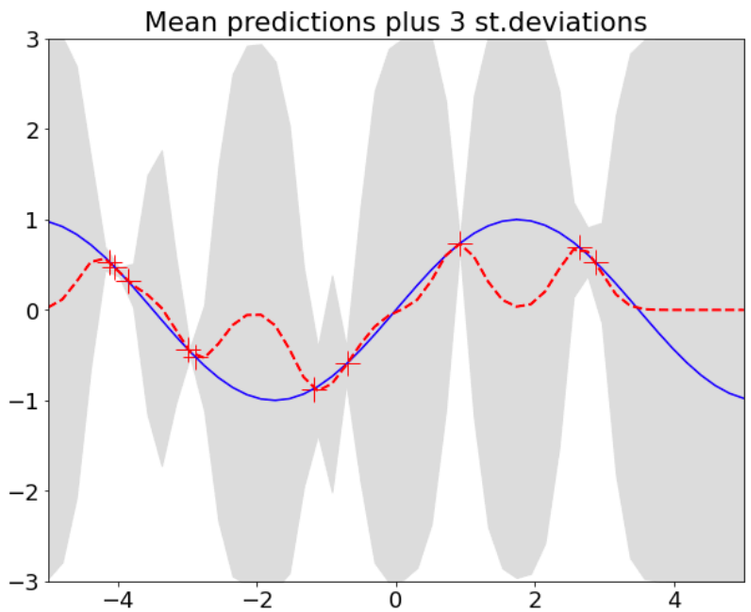
$$P(\begin{bmatrix} y_1 \\ \vdots \\ y_n \\ y \end{bmatrix} | x, [x_1, \cdots, x_n]) \sim N(0, \Sigma)$$

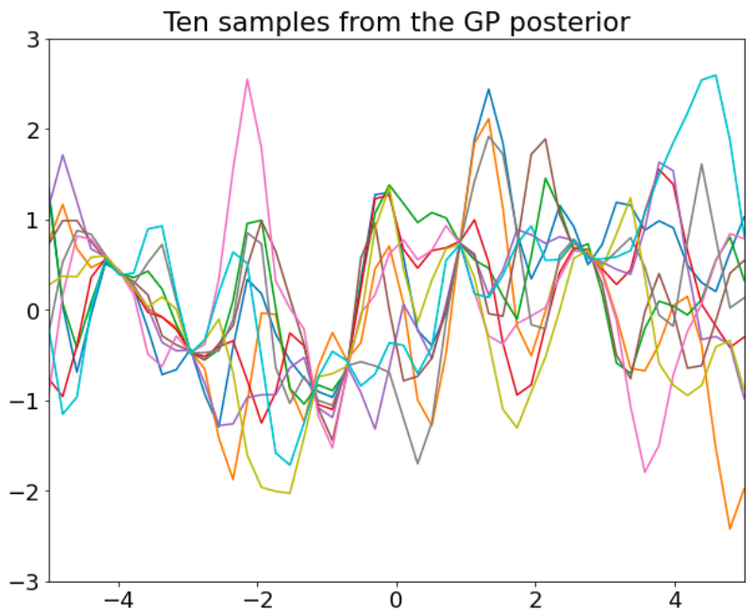
GP Posterior

$$P(y \,|\, x, y_{1\ldots n}, x_{1\ldots n}) \sim N(K_*^\intercal K^{-1} Y, K_{**} - K_*^\intercal K^{-1} K_*)$$

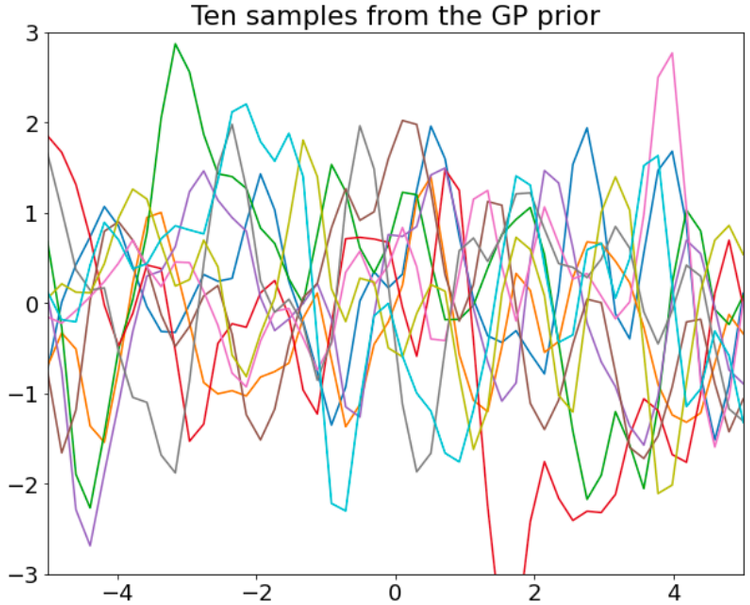
n=10

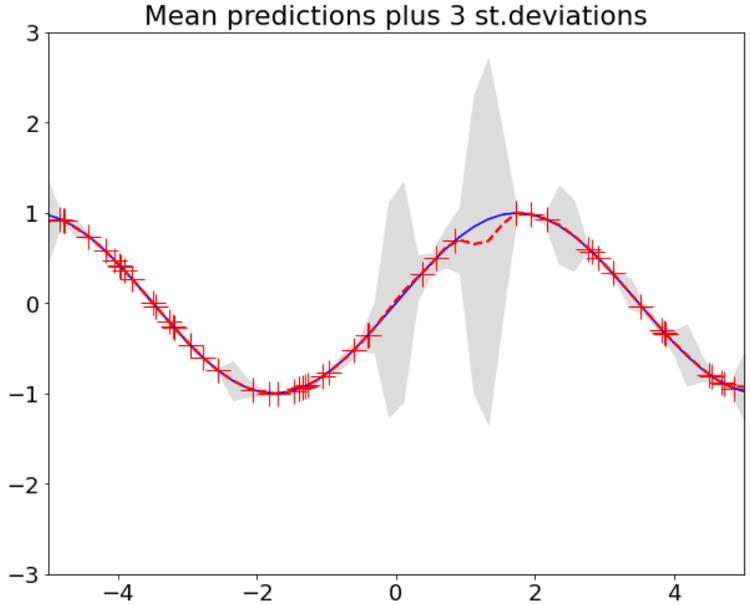


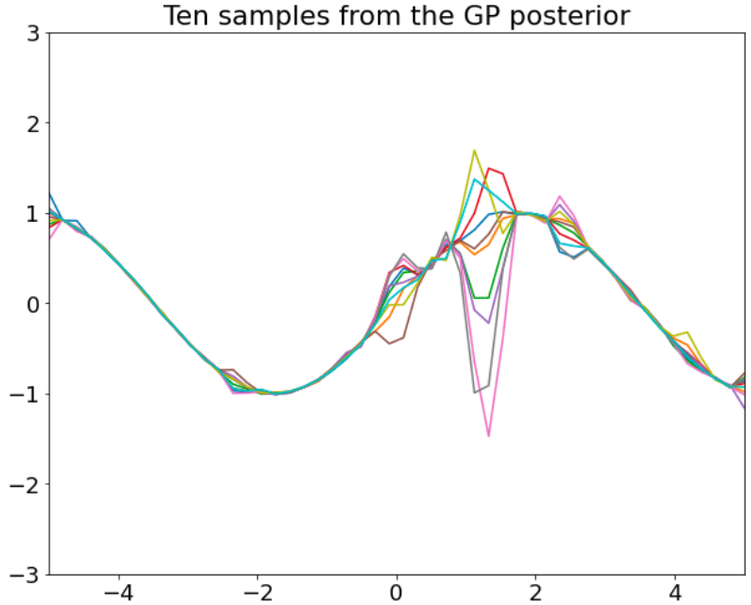




n=50







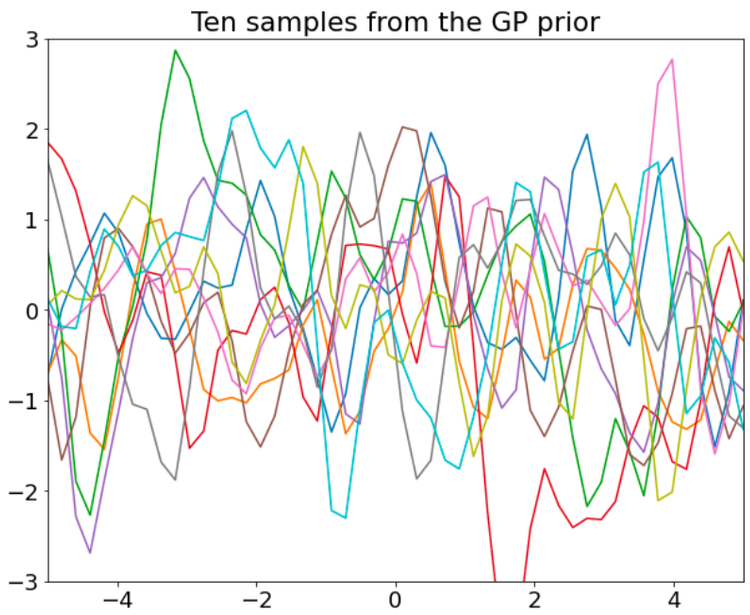
GP Prior

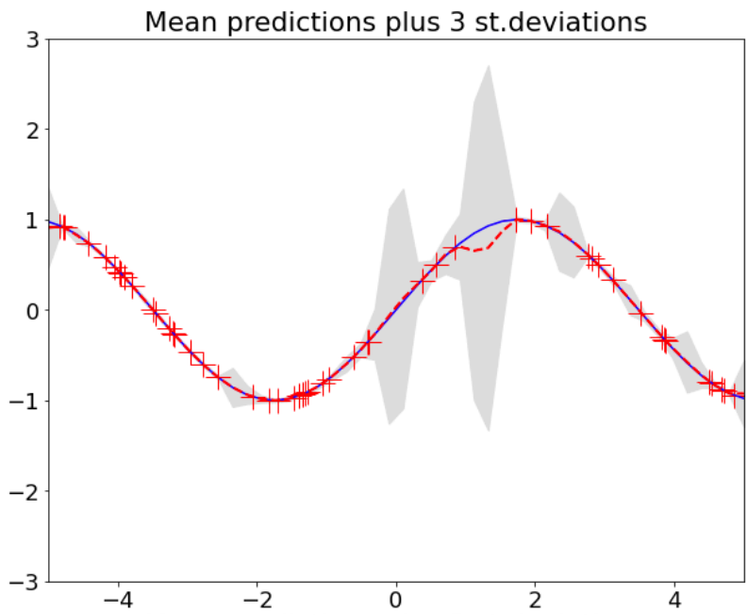
$$P(\begin{bmatrix} y_1 \\ \vdots \\ y_n \\ y \end{bmatrix} | x, [x_1, \cdots, x_n]) \sim N(0, \Sigma)$$

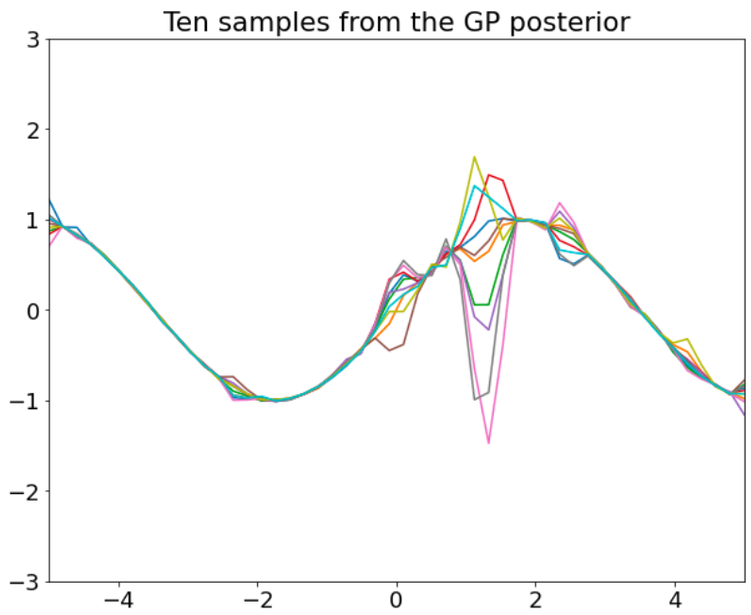
GP Posterior

$$P(y | x, y_{1\ldots n}, x_{1\ldots n}) \sim N(K_*^\intercal K^{-1} Y, K_{**} - K_*^\intercal K^{-1} K_*)$$

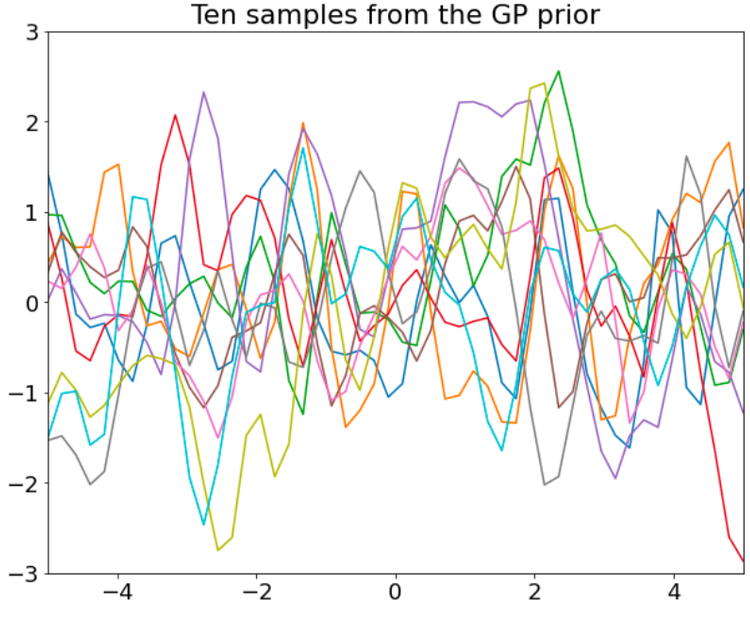
Noise -free

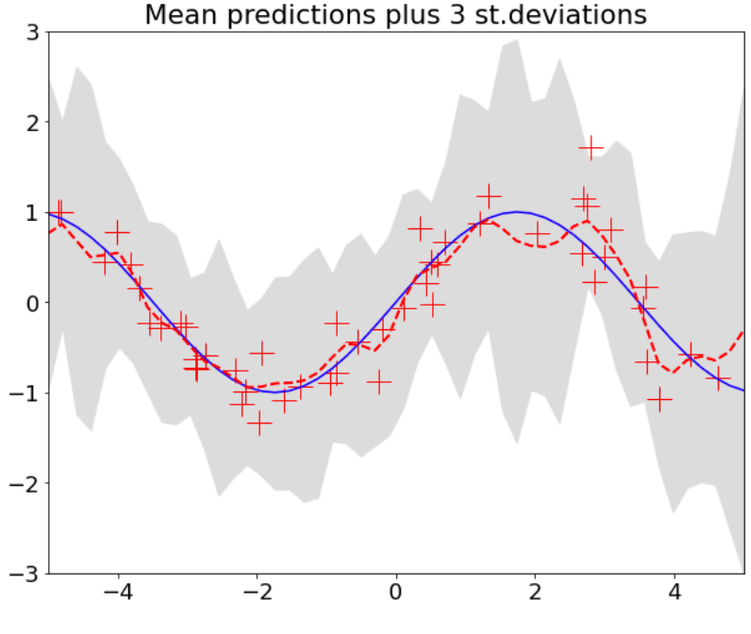


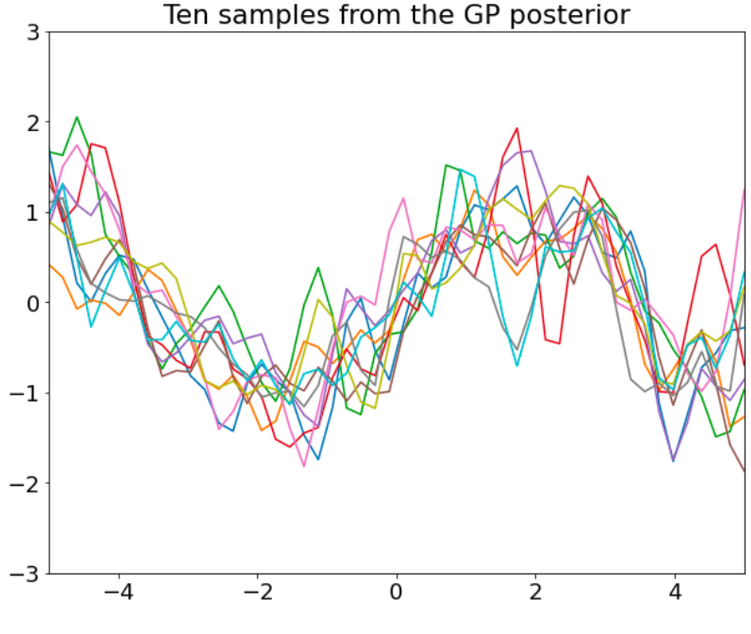




Noisy





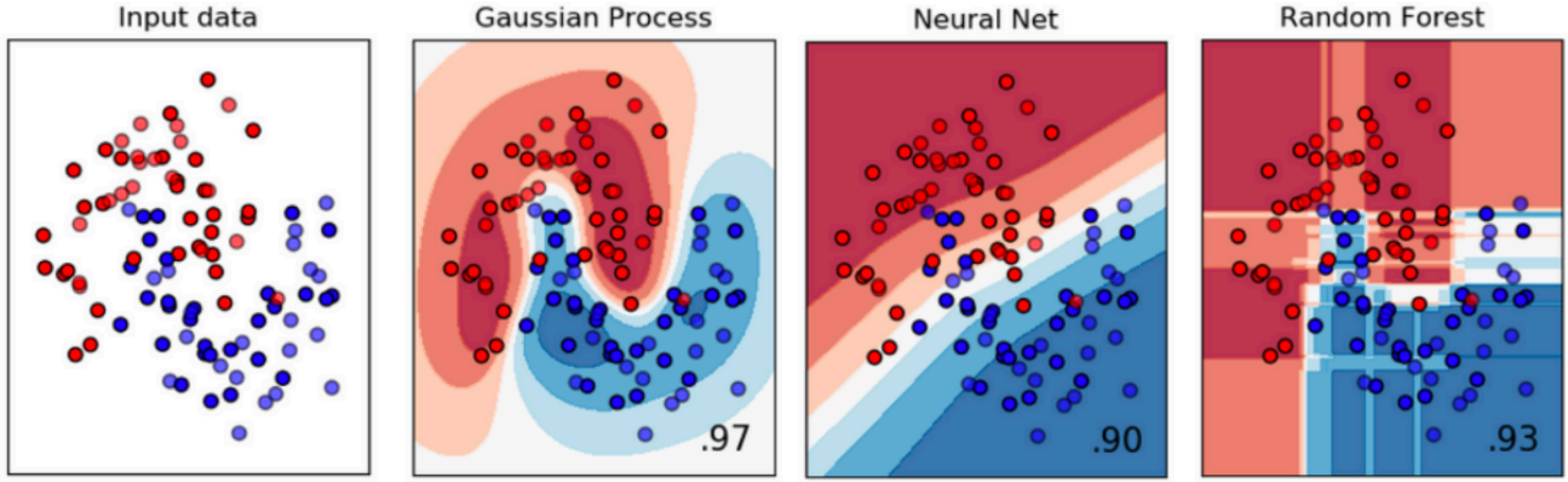


Why does uncertainty matter?

- Gaussian processes know what they don’t know
  - Many, if not most ML methods don’t share this property
  - **Uncertainty of a fitted GP increases away from the training data**, which is a consequence of GPs roots in probability and Bayesian inference

Why does uncertainty matter?

- Neural networks and random forest maintain high certainty of their predictions far from the training data
  - This could be linked to the phenomenon of **adversarial examples** where powerful classifiers give very wrong predictions for strange reasons



Pros and Cons

- Pros
  - Having uncertainty
  - Allowing us to incorporate expert knowledge
- Cons
  - Computationally expensive

Parametric v.s. Non-parametric

- Parametric: fixed-size of parameters learned during training
  - Slower training, faster testing
  - Ex. Neural networks
- Non-parametric: model size grows when getting more training data
  - Faster training, slower testing
  - Ex. k-NN, Gaussian process

3

Information Propagation

Related Works

- Exponential Expressivity in Deep Neural Networks through Transient Chaos, B. Poole et al., NeurIPS'16

- Deep Information Propagaiton, S. Schoenholz and J. Gilmer et al., ICLR'17

Related Works

- Exponential Expressivity in Deep Neural Networks through Transient Chaos, B. Poole et al., NeurIPS'16

- Deep Information Propagaiton, S. Schoenholz and J. Gilmer et al., ICLR'17

Motivation

- A key factor thought to underlie success of DNNs is their high expressivity
  - Compactly express highly complex functions over input space in a way that shallow networks cannot
  - Disentangle highly curved manifolds in input space into flattened manifolds in hidden space

Main Idea

- Combine Riemannian geometry with the mean field theory of high dimensional chaos to study the nature of **signal propagation** in deep neural networks
  - Reveal the existence of an **order to chaos transition** as a function of the statistics of weights and biases
  - For very broad classes of nonlinearities, deep networks can construct hidden representations whose global curvature grows exponentially with depth but not width

Signal Propagation

- Consider a fully-connected, untrained, feed-forward, neural network of depth L with layer width N_l and some nonlinearity ϕ
  - Since this is an untrained network, we suppose its weights and biases are respectively i.i.d. as $W_{ij}^l \sim N(0, \sigma_w^2/N_l)$ and $b_i^l \sim N(0, \sigma_b^2)$
  - Weight scaling ensures that the input contribution to each individual neuron at layer l from activation in layer $l-1$ remains $O(1)$

- The feedforward dynamics elicited is given by

$$z_i^l = \sum_i W_{ij}^l y_j^l + b_i^l; \;\; y_i^{l+1} = \phi(z_i^l)$$

Mean Field Theory

- Since the weights and biases are randomly distributed, these equations define a probability distribution on the activations over an ensemble of untrained networks

- The “mean-field” approximation is then replace z_i^l by a Gaussian whose first two moments match those of z_i^l

Signal Propagation

- Consider the evolution of single input $x_{i;\,a}$ as it evolves through the network, the first two moment will be,

$$\mathbb{E}[z^l_{i;\,a}] = 0 \qquad\qquad \mathbb{E}[z^l_{i;\,a} z^l_{j;\,a}] = q^l_{qq}\delta_{ij}$$

- For large N_l, this empirical distribution converges to a zero mean Gaussian since each $z^l_i = \sum_i W^l_{ij} y^l_j + b^l_i$ is a weighted sum of a large number of independent random variables, i.e. weights W^l_{ij} and b^l_i

- q^l_{aa} is the variance of the pre-activations

Signal Propagation

- By propagating this Gaussian distribution across one layer, we obtain the recursion relation

$$q_{aa}^{l} = \sigma_w^2 \int Dz\phi^2(\sqrt{q_{aa}^{l-1}}z) + \sigma_b^2$$

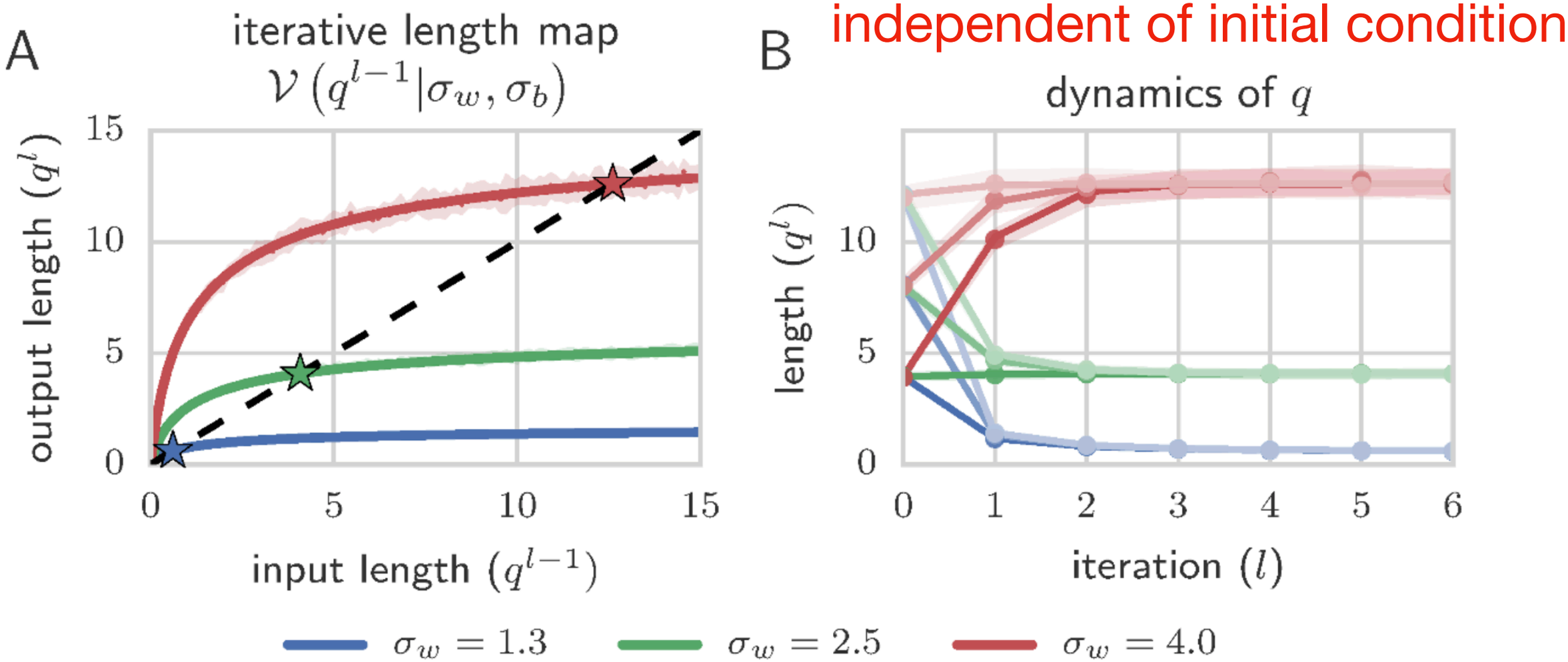
  - ,where $\int Dz = \frac{1}{\sqrt{2\pi}} \int dz e^{-\frac{1}{2}z^2}$ is the measure for a standard Gaussian distribution

- This describe the evolution of single input through mean field neural network

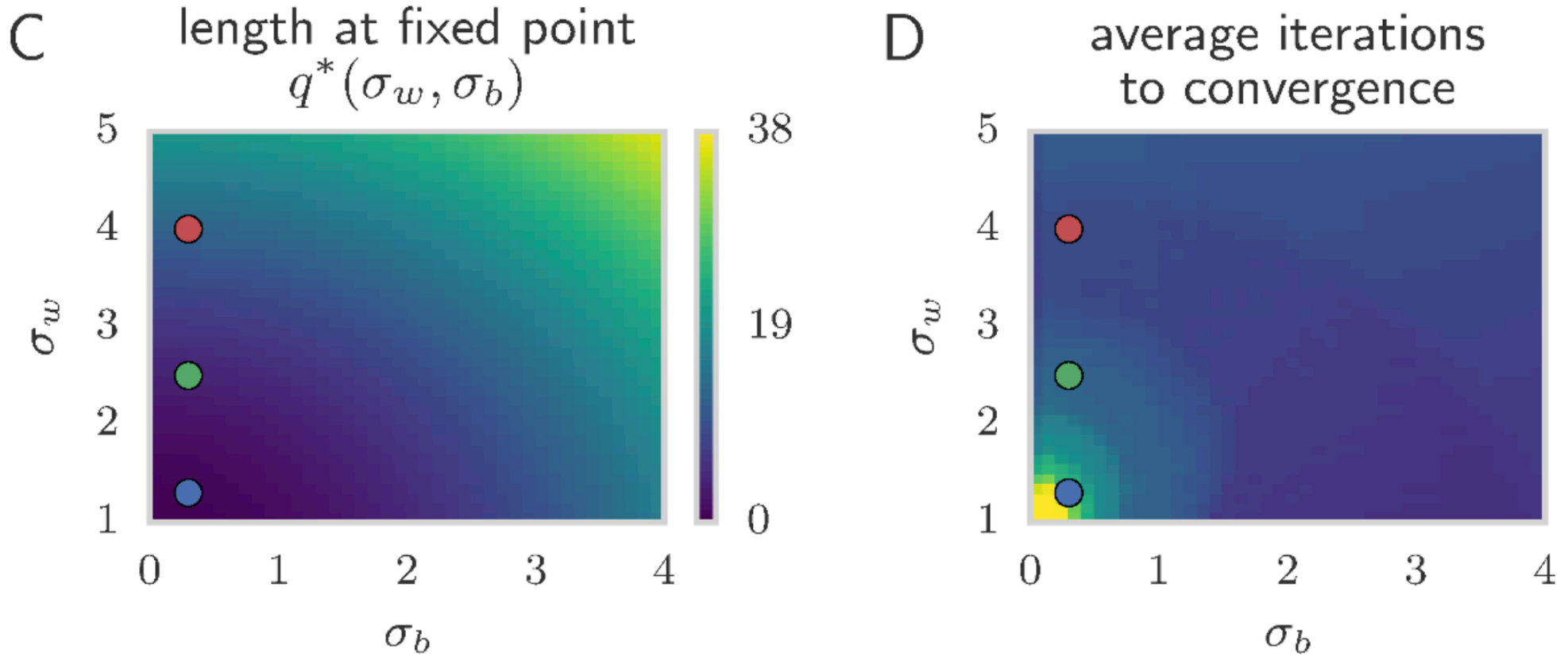
Signal Propagation

- For any choice of σ_w^2 and σ_b^2 with bounded ϕ, q_{aa}^l has a **fixed point** $q^*(\sigma_w, \sigma_b) = \lim_{l \to \infty} q_{aa}^l$

Rapid convergence, independent of initial condition



Signal Propagation



	$\sigma_b = 0$	$\sigma_b \neq 0$
$\sigma_w < 1$	Shrinks all 0	Non-zero
$\sigma_w > 1$	0 & non-zero	Non-zero

Expand small, shrinks large

Transient Chaos in Deep Networks

- Now consider two input $x^0_{i;\,a}$ and $x^0_{i;\,b}$. The geometry of these two inputs as they propagate through the network is captured by the 2 x 2 matrix of inner products

Variance

$$Q^l = \begin{bmatrix} q^l_{aa} & q^l_{ab} \\ q^l_{ba} & q^l_{bb} \end{bmatrix}$$

Covariance

$$\mathbb{E}[z^l_{i;\,a} z^l_{j;\,b}] = q^l_{ab}\delta_{ij}$$

Transient Chaos in Deep Networks

- The covariance q_{ab}^{l} will be given by the recurrence relation

$$q_{ab}^{l} = \sigma_w^2 \int Dz_1 Dz_2 \phi(u_1)\phi(u_2) + \sigma_b^2$$

- ,where $u_1 = \sqrt{q_{aa}^{l-1}} z_1$ and $u_2 = \sqrt{q_{bb}^{l-1}} \left(c_{ab}^{l-1} z_1 + \sqrt{1 - (c_{ab}^{l-1})^2} z_2 \right)$
- $c_{ab}^{l} = q_{ab}^{l} (q_{aa}^{l} q_{bb}^{l})^{-1/2}$ is the **correlation coefficient**

Transient Chaos in Deep Networks

- We can further derive the iterative correlation coefficient map, or C-map for c_{ab}^{l}:

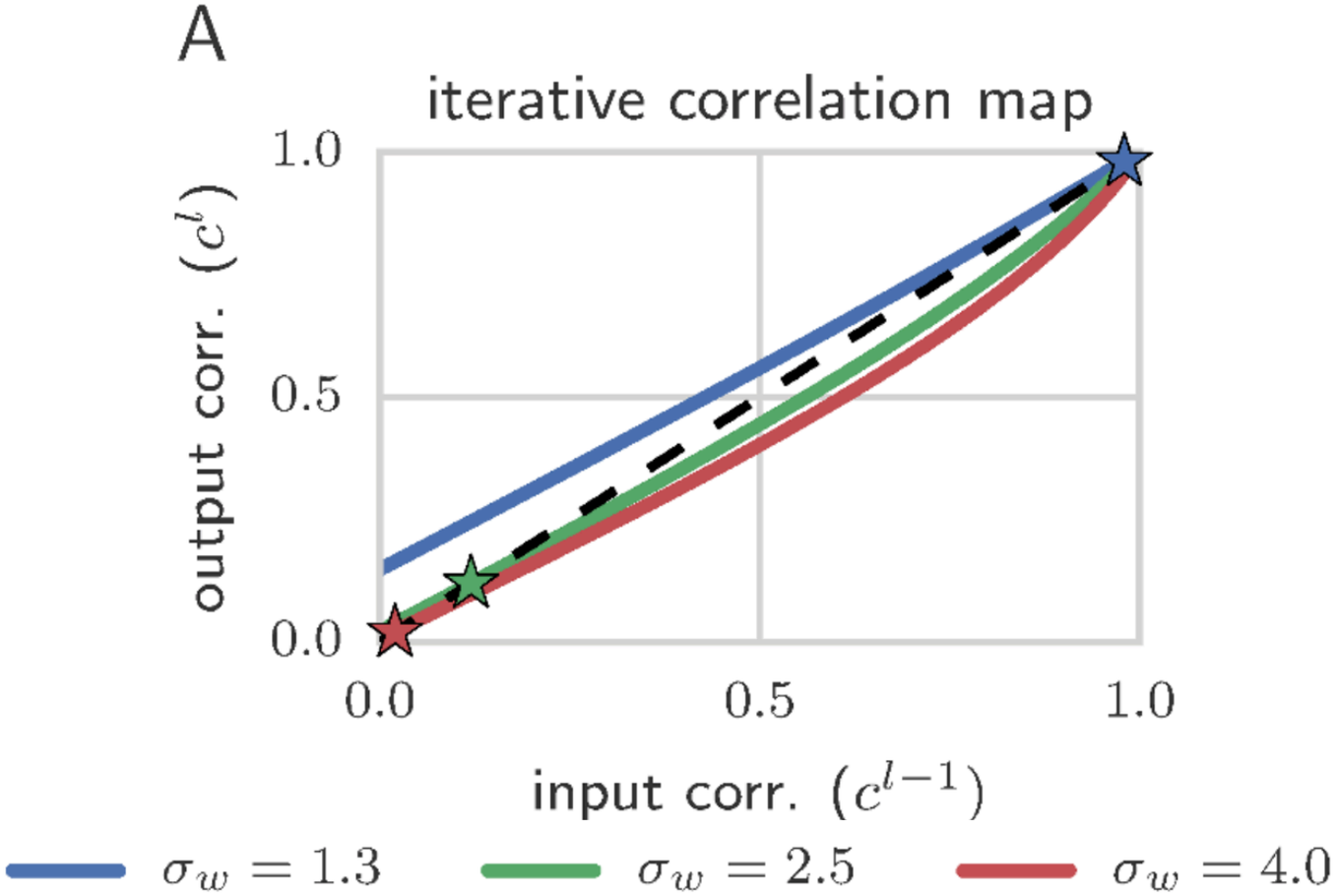
$$c_{ab}^{l} = \frac{1}{q^*} C(c_{ab}^{l-1}, q^*, q^* \,|\, \sigma_w, \sigma_b)$$

- What happens to two nearby points as they propagate through the layers?

Their relation can be tracked by the **correlation coefficient** c_{12}^{l} between two points, which approaches a fixed point $c^*(\sigma_w, \sigma_b)$ at large depth

Transient Chaos in Deep Networks

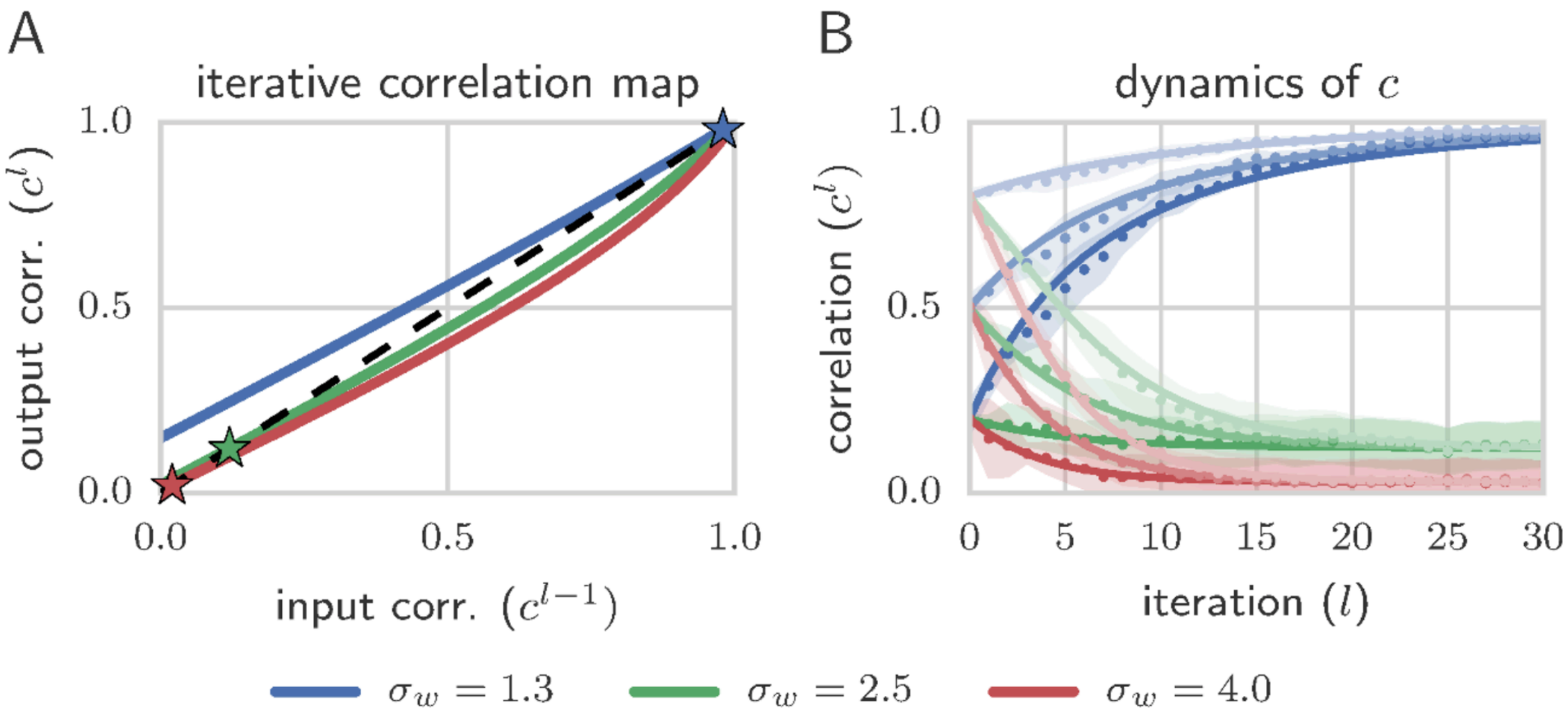
- It always has a fixed point at $c^* = 1$
- However, the stability of this fixed point depends on the slope χ_1 of map at 1



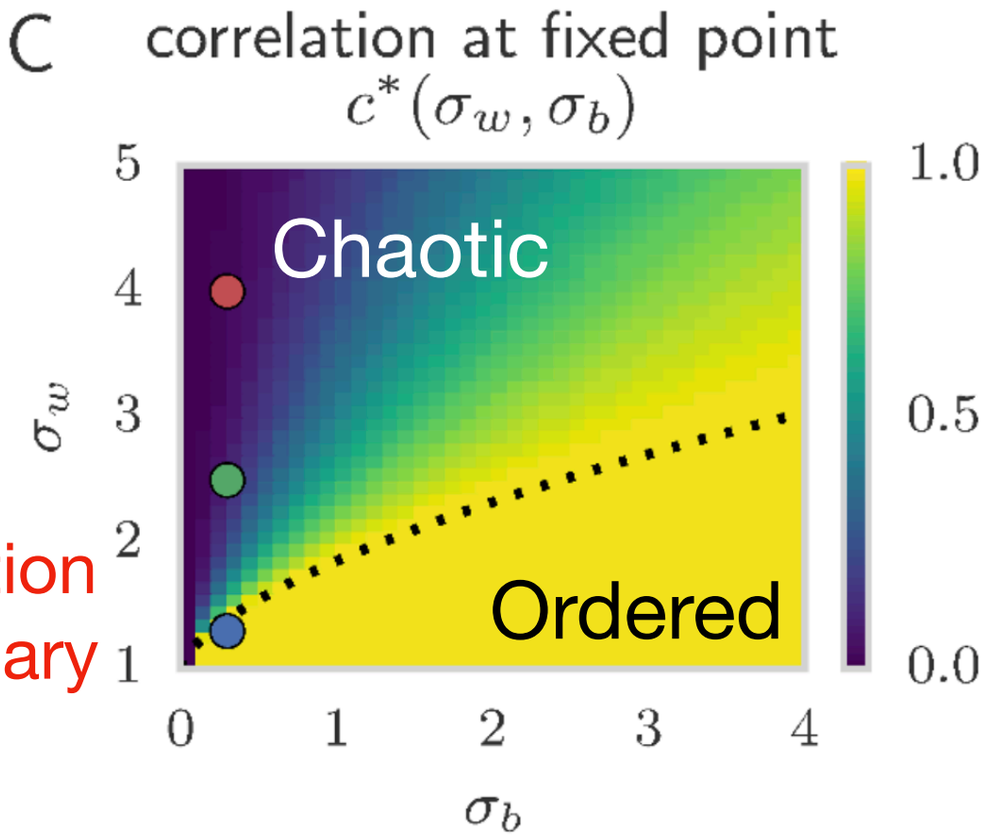
	Stability	**2 points**
$\chi_1 < 1$	Stable	Similar
$\chi_1 > 1$	Unstable	Seperate

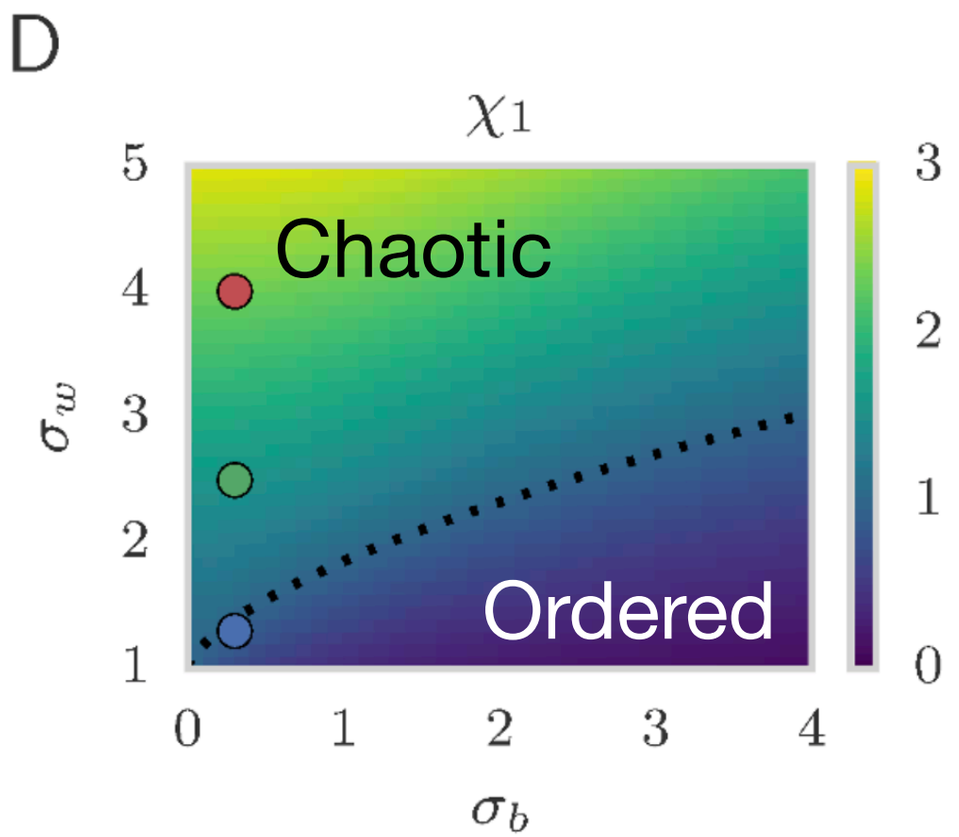
Transient Chaos in Deep Networks

- Correlation dynamics are much slower than the length dynamics, because C-map is closer to the unity line



Transient Chaos in Deep Networks





Transition Boundary

Finite σ_b	Small σ_w	Medium σ_w	Large σ_w
χ_1	< 1	~ 1	> 1
$c^*(\sigma_w, \sigma_b)$	1 Converge	1& non-one	0 De-correlate (orthogonal)

Conclusion

- Reveal the existence of an order to chaos transition as a function of the statistics of weights and biases
  - Weights and nonlinearities de-correlate inputs, while biases correlate them
- Random neural networks are exponentially expressive in their depth

Related Works

- Exponential Expressivity in Deep Neural Networks through Transient Chaos, B. Poole et al., NeurIPS'16

- Deep Information Propagaiton, S. Schoenholz and J. Gilmer et al., ICLR'17

Motivation

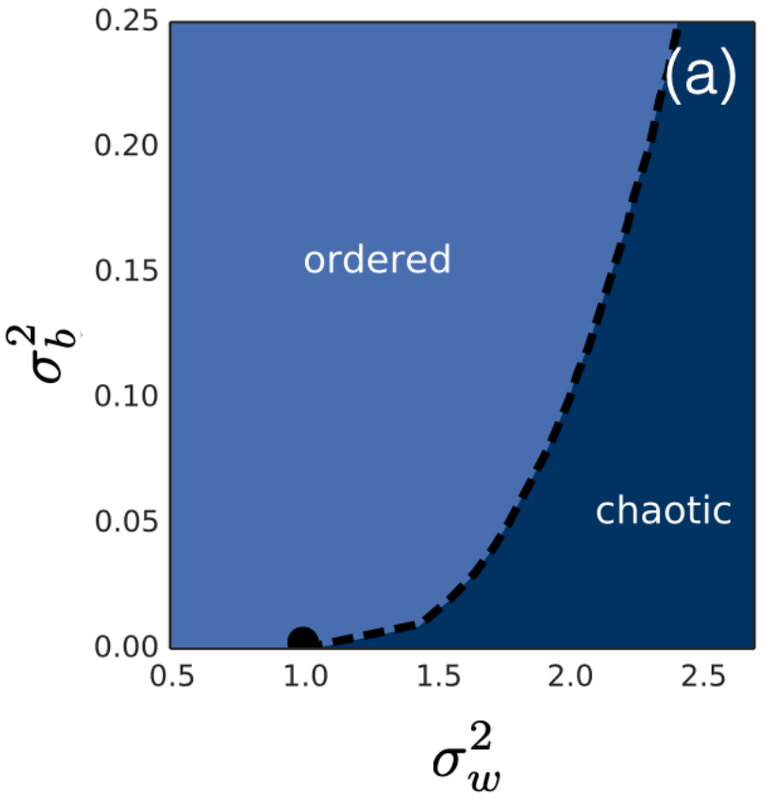
- Despite the great success of deep networks, designing novel network architectures is frequently equal parts art and science

Main Idea

- Show the existence of **depth scales** that naturally limit the maximum depth of signal propagation through these random networks
  - Random networks may be trained precisely when information can travel through them
  - The depth scales provide bounds on how deep a network may be trained for a specific choice of hyperparameters
- Develop a mean field theory for backpropagation and show that the ordered and chaotic phases correspond to regions of vanishing and exploding gradient respectively

Recap

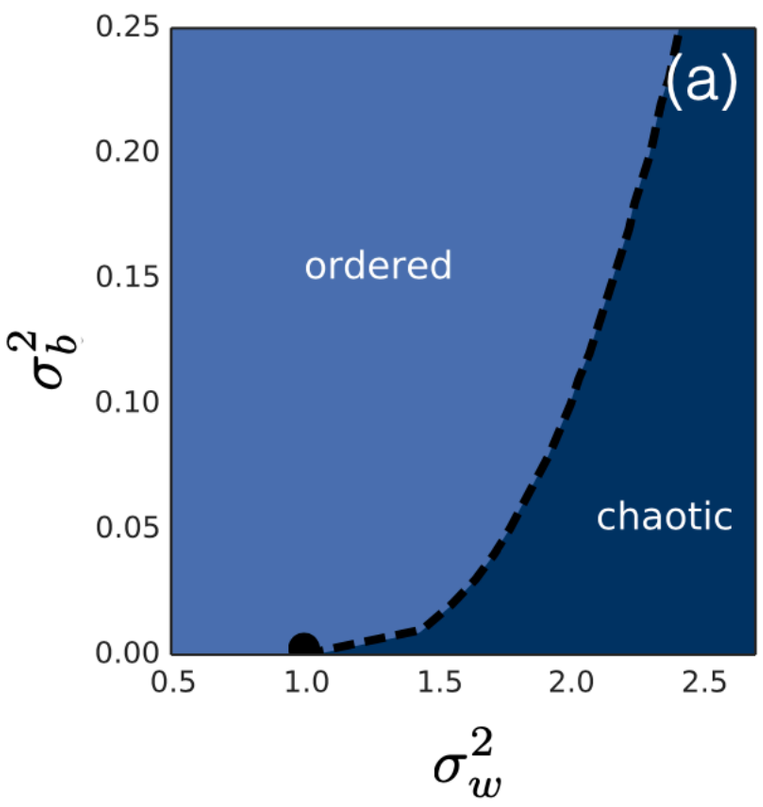
- Mean field phase diagram shows the boundary between ordered and chaotic phases as a function of σ_w^2 and σ_b^2

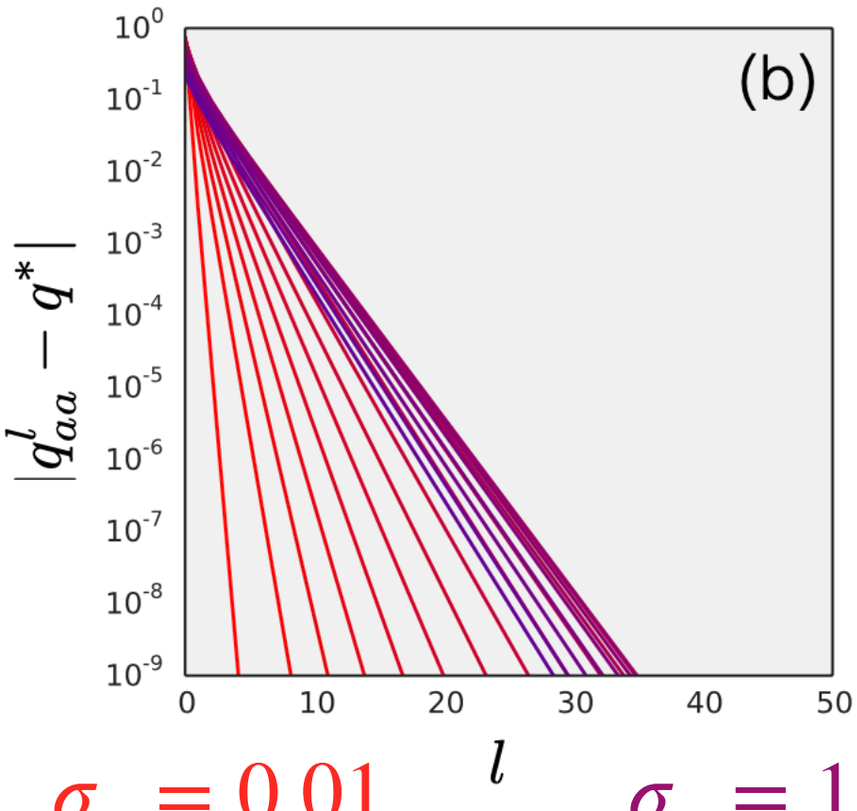


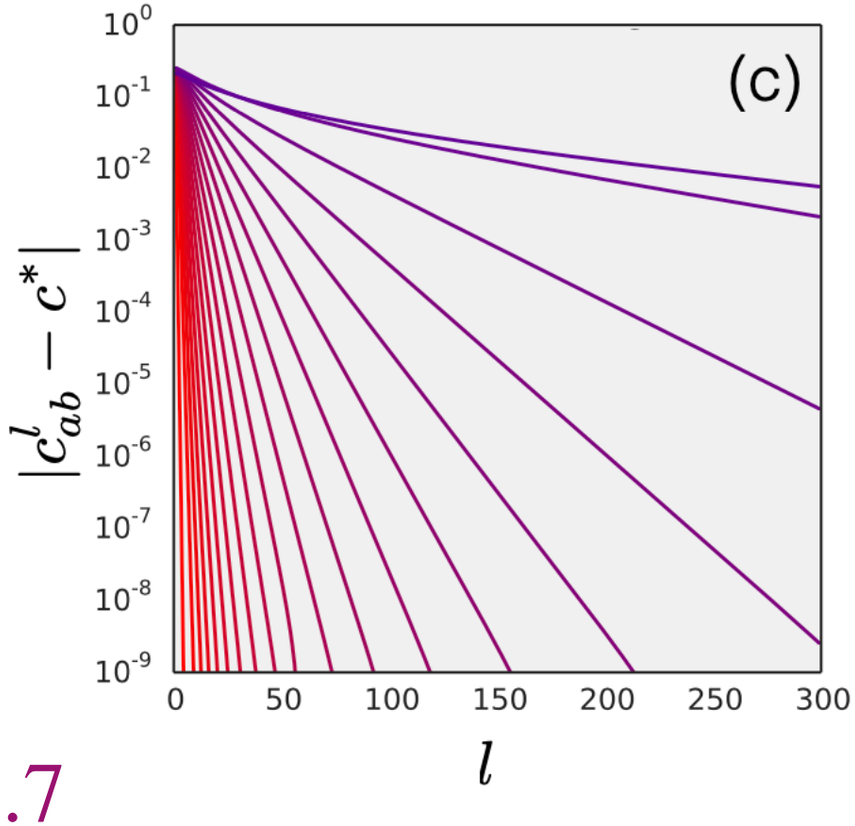
Depth Scales

- The residual as a function of depth indicates clear exponential behavior

- Anticipate asymptotically $|q_{aa}^{l} - q^{*}| \sim e^{-\frac{l}{\xi_q}}$ and $|c_{ab}^{l} - c^{*}| \sim e^{-\frac{l}{\xi_c}}$ for sufficiently large l







$\sigma_w = 0.01$ $\sigma_w = 1.7$

Depth Scales

- The residual as a function of depth indicates clear exponential behavior

- Anticipate asymptotically $|q_{aa}^{l} - q^{*}| \sim e^{-\frac{l}{\xi_q}}$ and $|c_{ab}^{l} - c^{*}| \sim e^{-\frac{l}{\xi_c}}$ for sufficiently large l
  - ξ_q and ξ_c define depth-scales over which information may propagate about the magnitude of a single input and the correlation between two inputs respectively

- This asymptotic recurrence relation in turn implies exponential decay towards the fixed point over a depth-scale ξ_x

Depth Scales

- Let $q^l_{aa} = q^* + \epsilon^l$. By construction so long as $\lim_{l\to\infty} q^l_{aa} = q^*$ exists it follows that $\epsilon^l \to 0$ as $l \to \infty$

$$\epsilon^{l-1} = \epsilon^l\left[\chi_1 + \sigma_w^2 \int Dz\phi''(\sqrt{q^*}z)\phi(\sqrt{q^*}z)\right] + O((\epsilon^l)^2)$$

- Notably, the term multiplying ϵ^l is a constant. It follows that for large l the asymptotic recurrence relation has an exponential solution $\epsilon^l \sim e^{-\frac{l}{\xi_q}}$

$$\xi_q^{-1} = -\log\left[\chi_1 + \sigma_w^2 \int Dz\phi''(\sqrt{q^*}z)\phi(\sqrt{q^*}z)\right]$$

Depth Scales

- This establishes ξ_q as depth scale that controls how deep information from a single input may penetrate into a random neural network

Depth Scales

- Similarly, we can expect $c_{ab}^{l} = c^{*} + \epsilon^{l}$ to find asymptotic recurrence relation

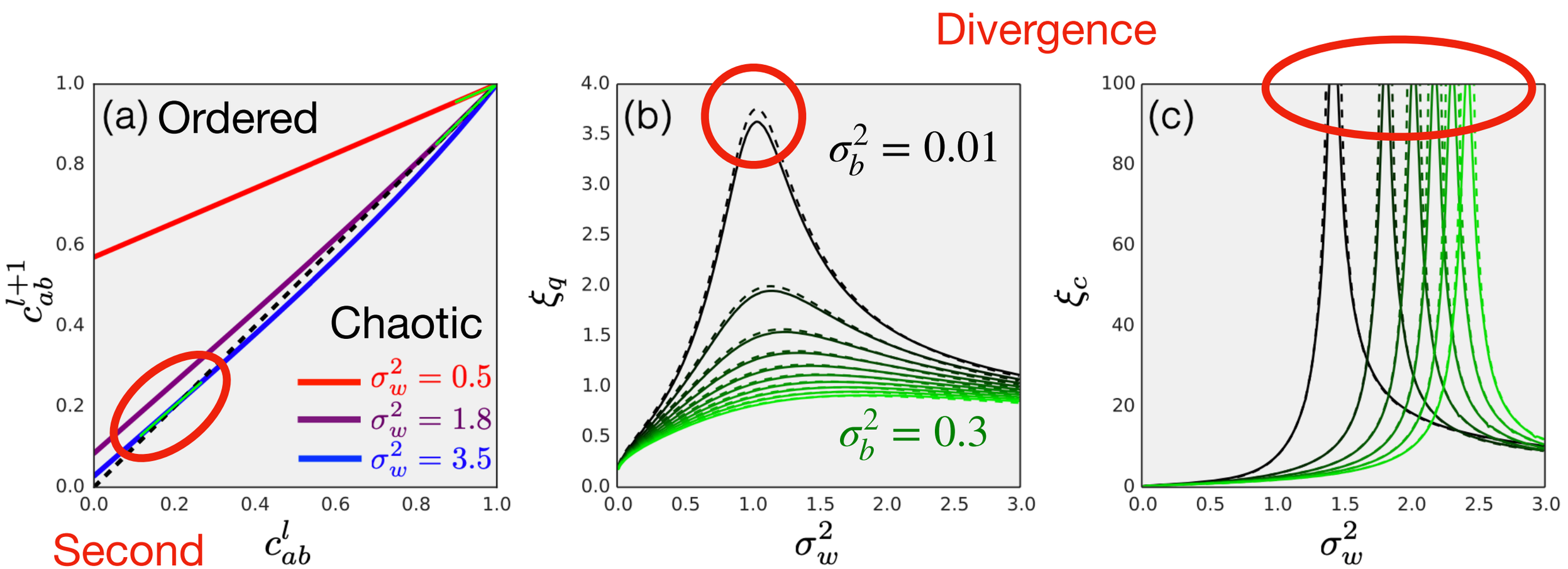
- Once again, we expect for large l this recurrence will have an exponential solution $\epsilon^{l} \sim e^{-\frac{l}{\xi_c}}$

$$\xi_c^{-1} = -\log\left[\sigma_w^2 \int Dz_1 Dz_2 \phi'(u_1^*)\phi'(u_2^*)\right]$$

Depth Scales

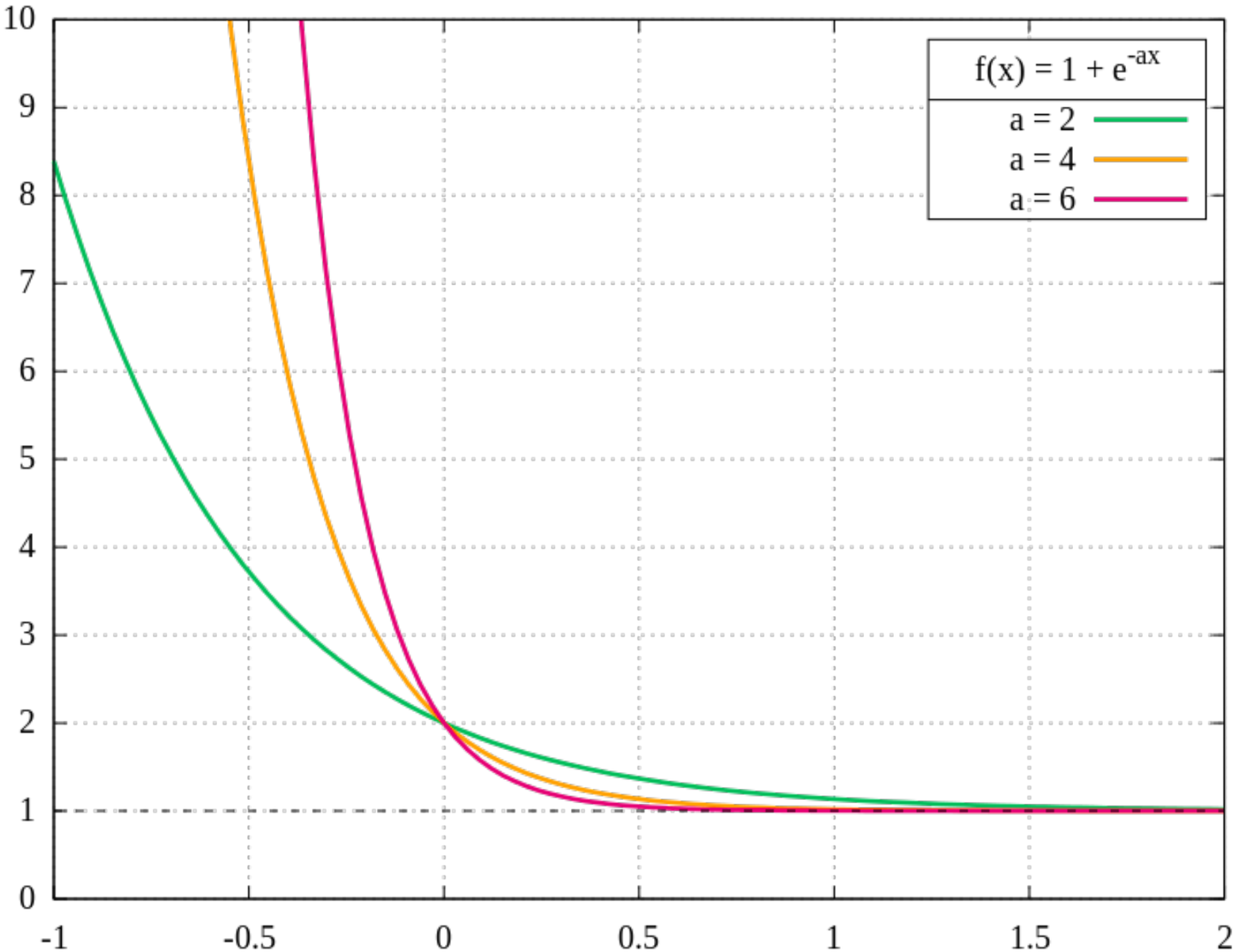
- In the ordered phase $c^* = 1$ and so $\xi_c^{-1} = -\log \chi_1$
- Since the transition between order and chaos occurs when $\chi_1 = 1$ it follow that ξ_c diverges $\epsilon^l \sim e^{-\frac{l}{\xi_c}}$

Divergence



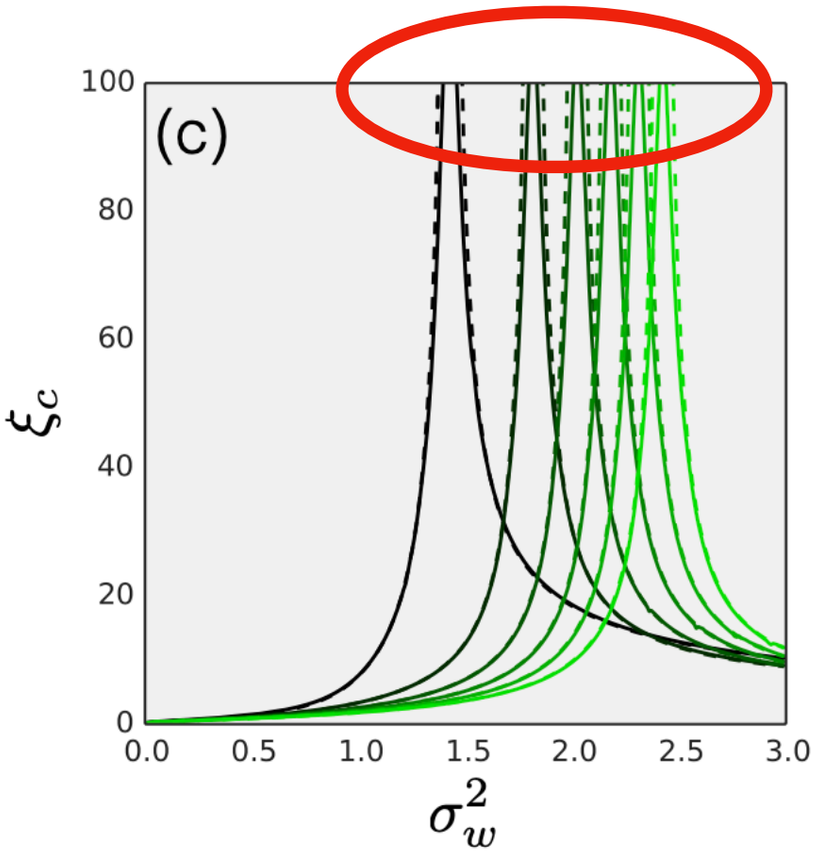
Second fixed point

Depth Scales



$$\epsilon^l \sim e^{-\frac{l}{\xi_c}}$$

Divergence



Dropout

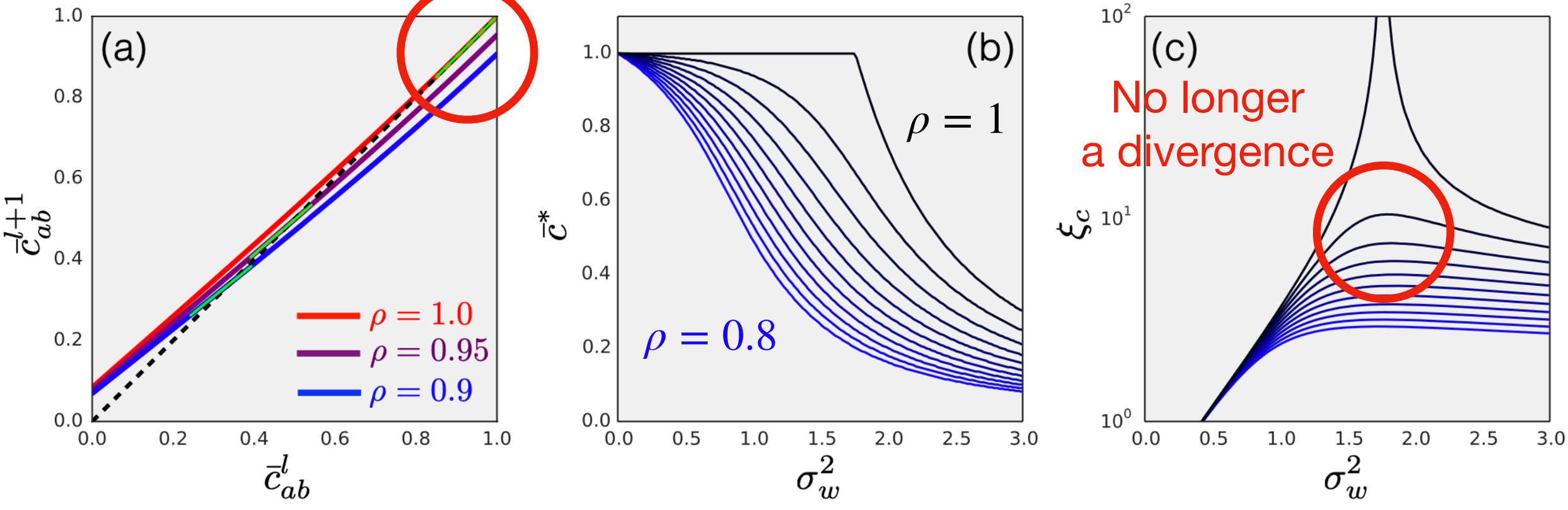
- The mean field formalism can be extended to include dropout

- Even infinitesimal amounts of dropout destroys the mean field critical point, and therefore limits the trainable network depth

Dropout

- Since there is no longer a sharp critical point with dropout, we don't expect a diverging depth scale
  - Identical inputs passed through two different dropout masks will become increasingly dissimilar as the dropout rate increases

$\bar{c}^{l}_{ab} = 1$ is no longer a fixed point



Gradient Backpropagation

- There is a duality between **forward propagation of signals** and **backpropagation of gradients**

- Consider the backpropagation equations given a loss E

$$\frac{\partial E}{\partial W_{ij}^l} = \frac{\partial E}{\partial z_i^l}\frac{\partial z_i^l}{\partial W_{ij}^l} = \delta_i^l \phi(z_j^{l-1})$$

$$\delta_i^l = \frac{\partial E}{\partial z_i^l} = \phi'(z_i^l)\sum_i \delta_j^{l+1} W_{ji}^{l+1}$$

Scale of fluctuations of gradients will be proportional to $\mathbb{E}[(\delta_i^l)^2]$

- We can work out a recurrence relation for the variance of the error, $\tilde{q}_{aa}^l = \mathbb{E}[(\delta_i^l)^2]$

$$\tilde{q}_{aa}^l = \tilde{q}_{aa}^{l+1}\frac{N_{l+1}}{N_l}\chi_1$$

Gradient Backpropagation

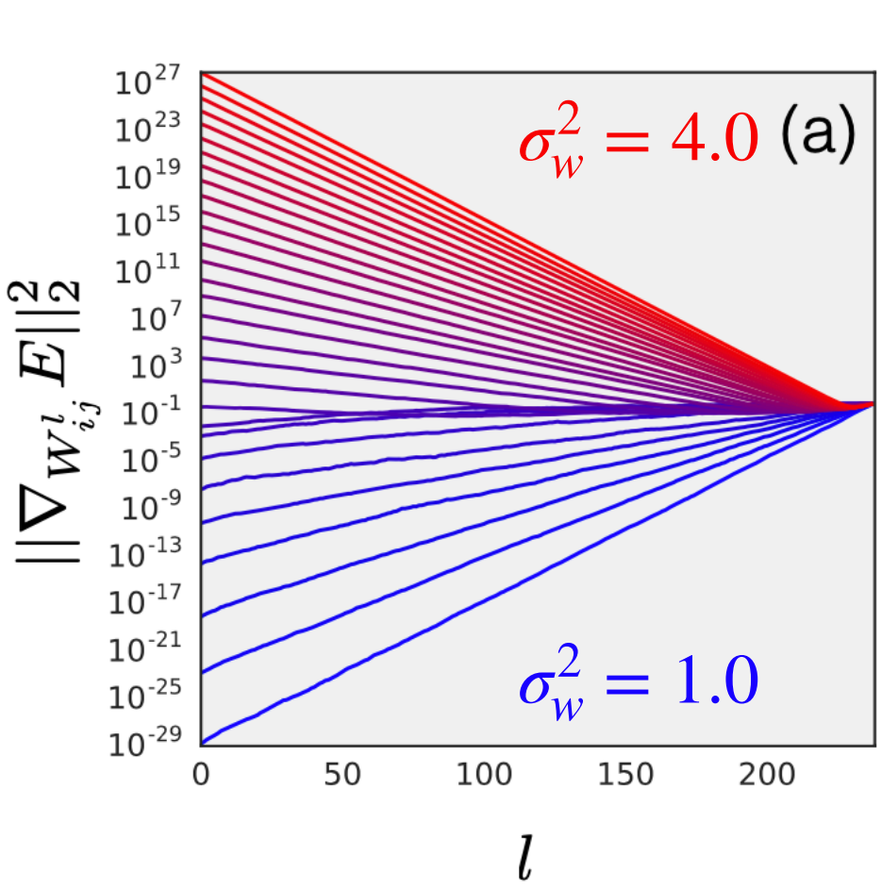
- Since χ_1 depends only on the asymptotic q^* it follows that for constant width networks we expect to again have an exponential solution with

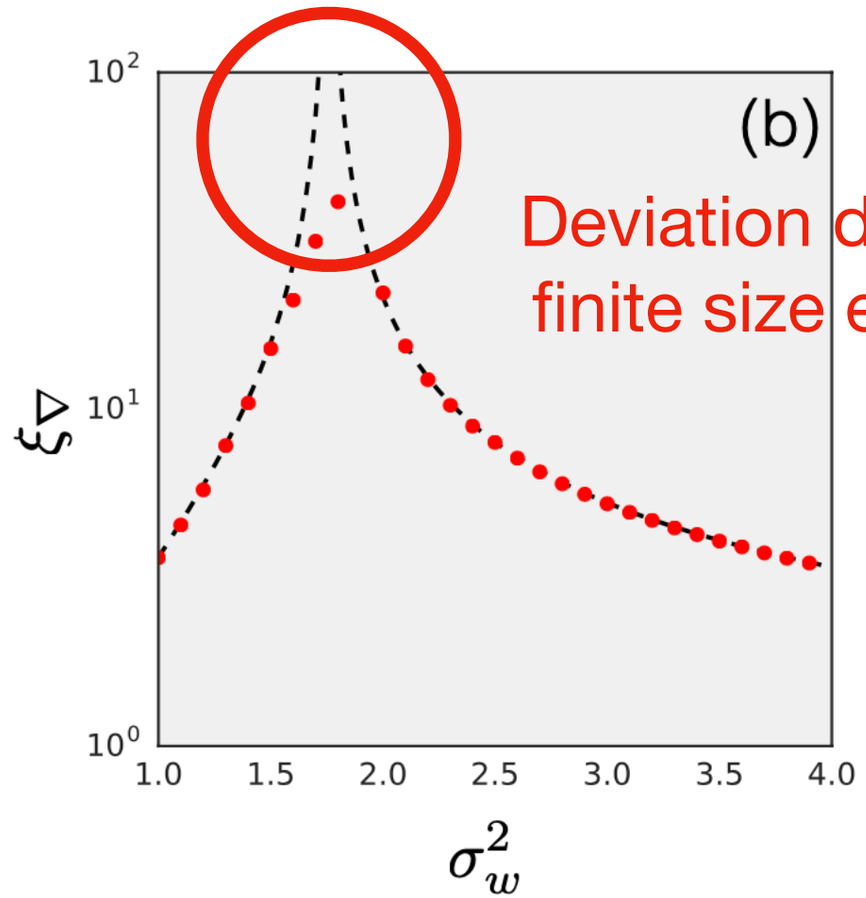
$$\tilde{q}^l_{aa} = \tilde{q}^L_{aa} e^{-(L-l)/\xi_\nabla} \qquad \xi_\nabla^{-1} = -\log \chi_1$$

Phase	χ_1	ξ_∇^{-1}	Gradient
Ordered	< 1	> 0	Vanish
Critical	$\to 1$	$\to \infty$	Stable
Chaotic	> 1	< 0	Explode

Gradient Backpropagation

- 2-norm of the gradients, $\|\nabla_{W_{ab}^l} E\|_2^2$, behaves exponentially over many orders of magnitude



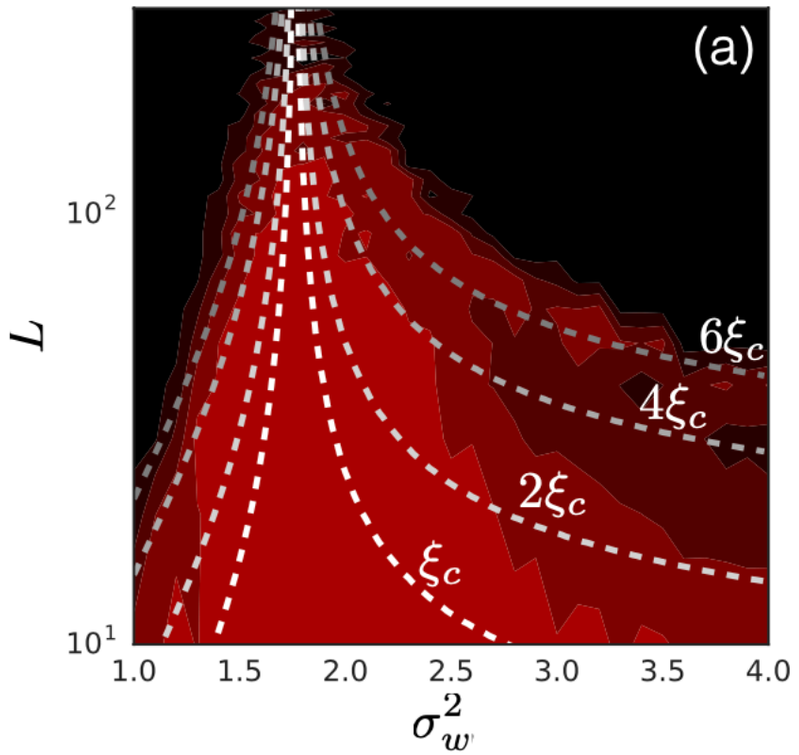


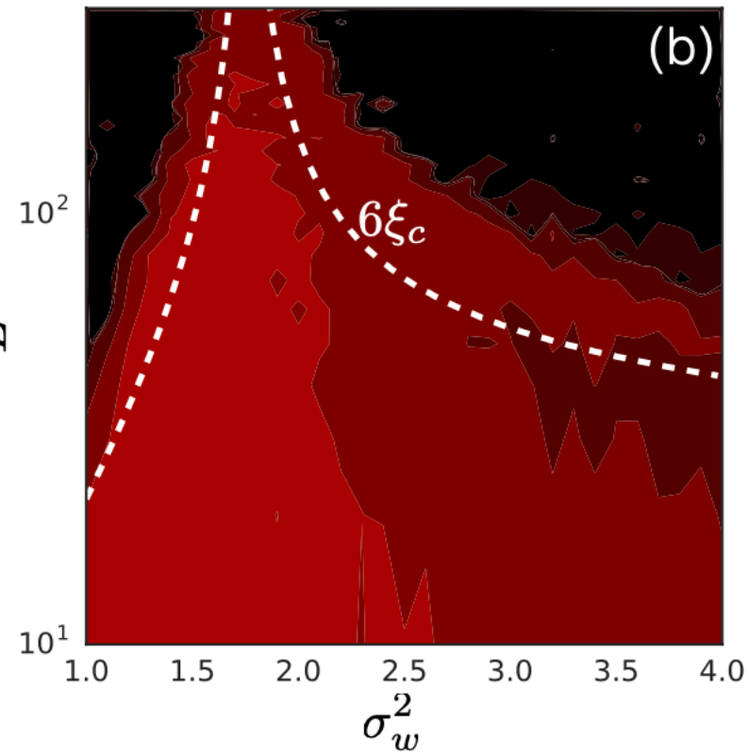
Hypothesis

- A necessary condition for a random network to be trained is that
    1. information about the inputs should be able to propagate forward
    2. information about the gradients should be able to propagate backward

Low Accuracy High

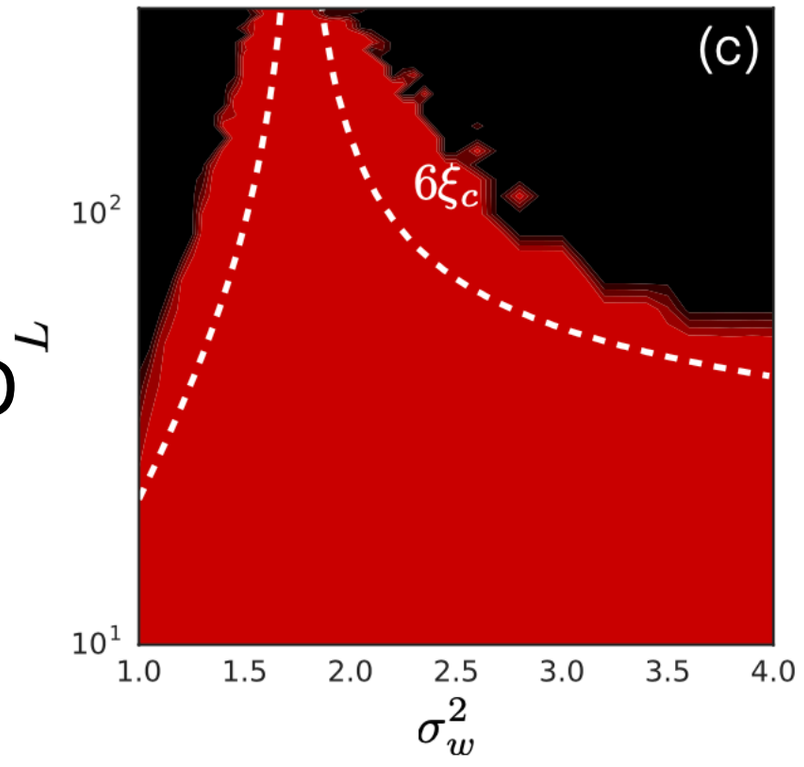
MNIST
200 / SGD

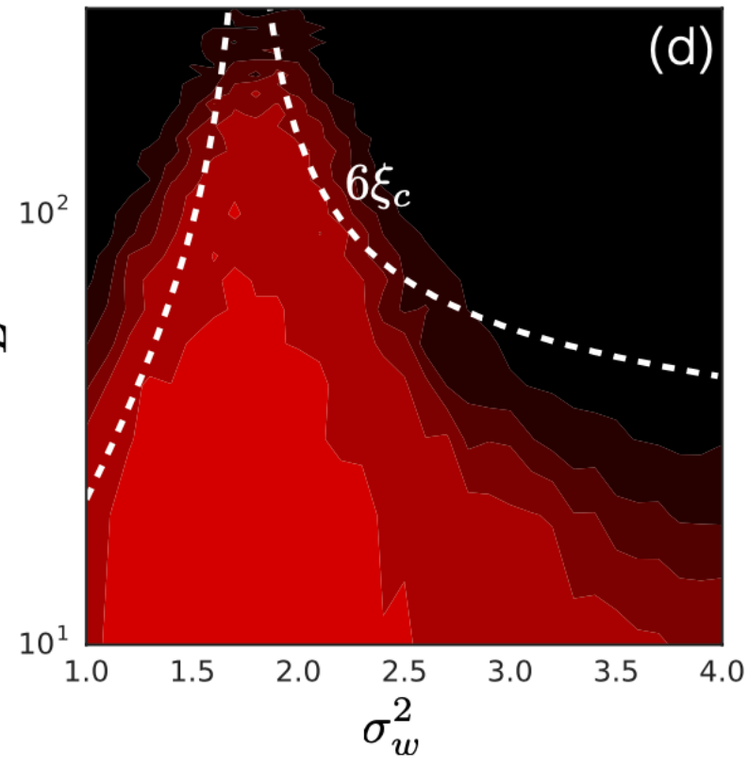




CIFAR
2000 / SGD

MNIST
14000 / SGD

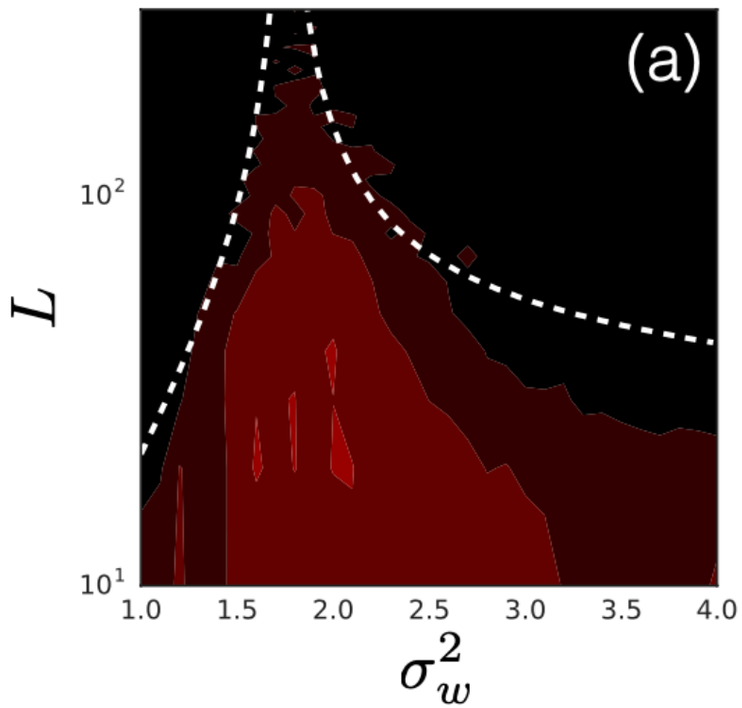


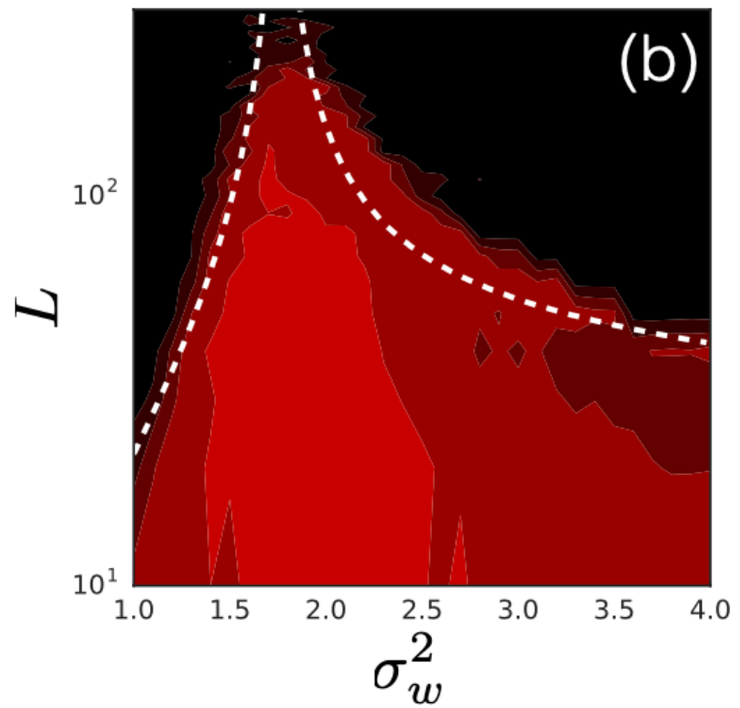


MNIST
300 / RMS

Low Accuracy High

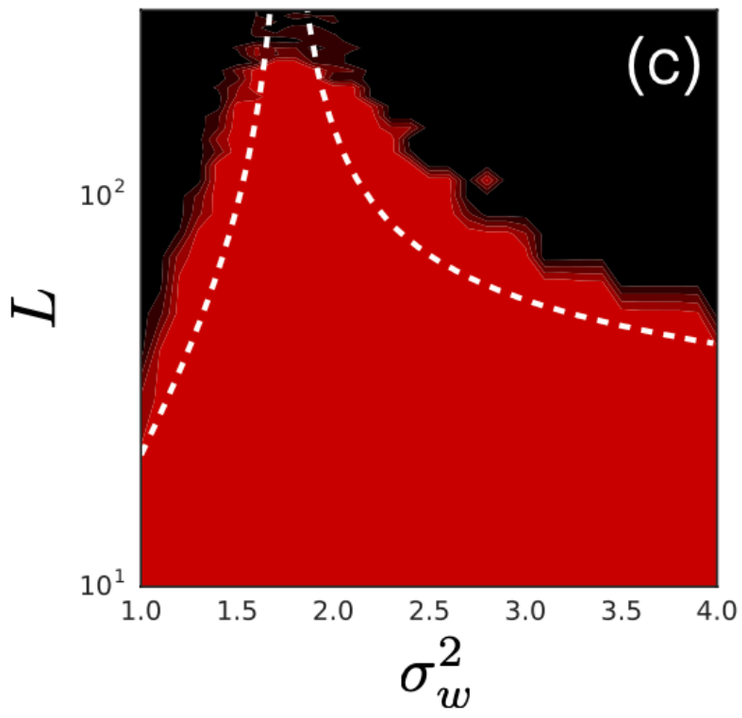
MNIST
45 / SGD

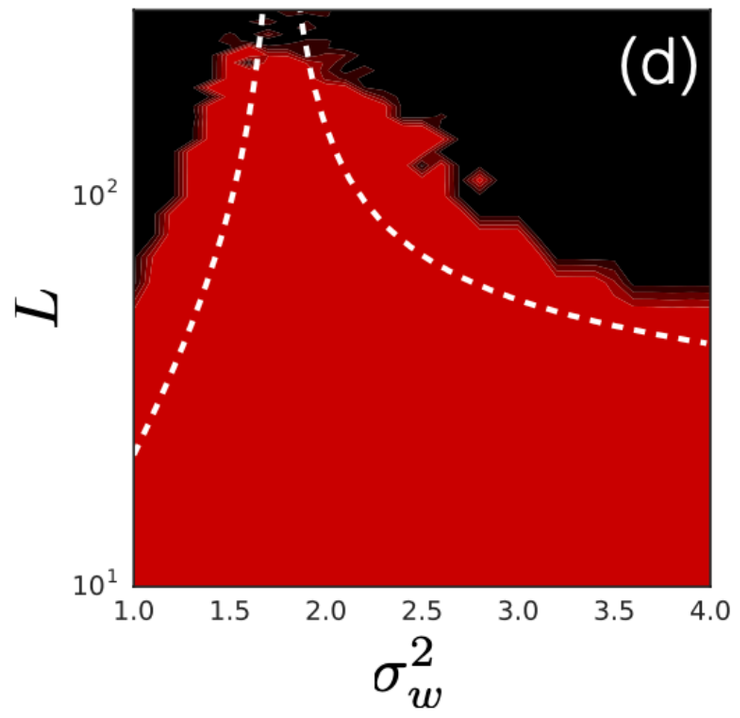




MNIST
304 / SGD

MNIST
2048 / SGD

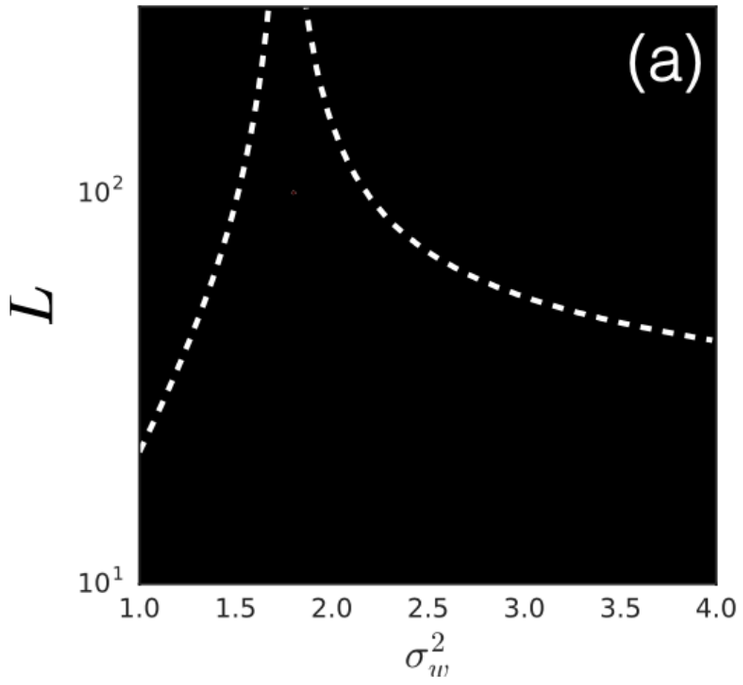


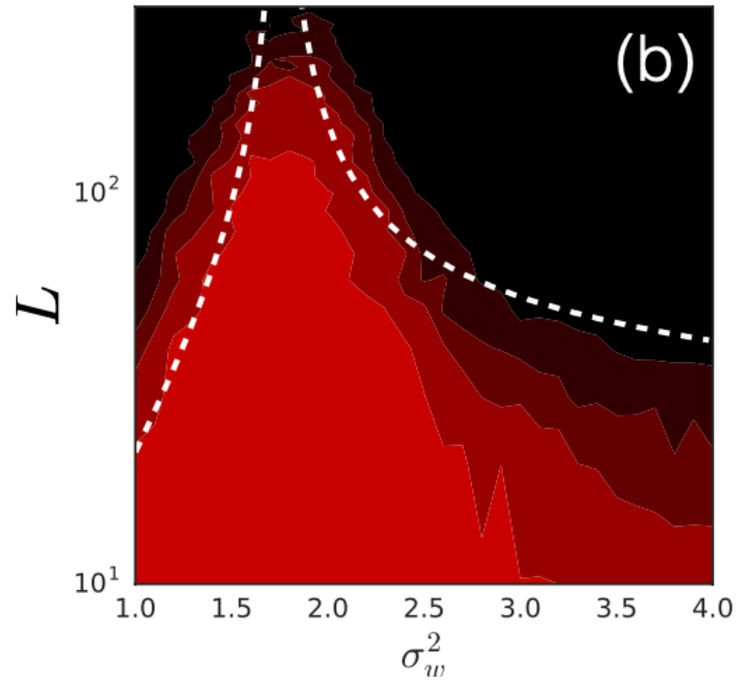


MNIST
13780 / SGD

Low Accuracy High

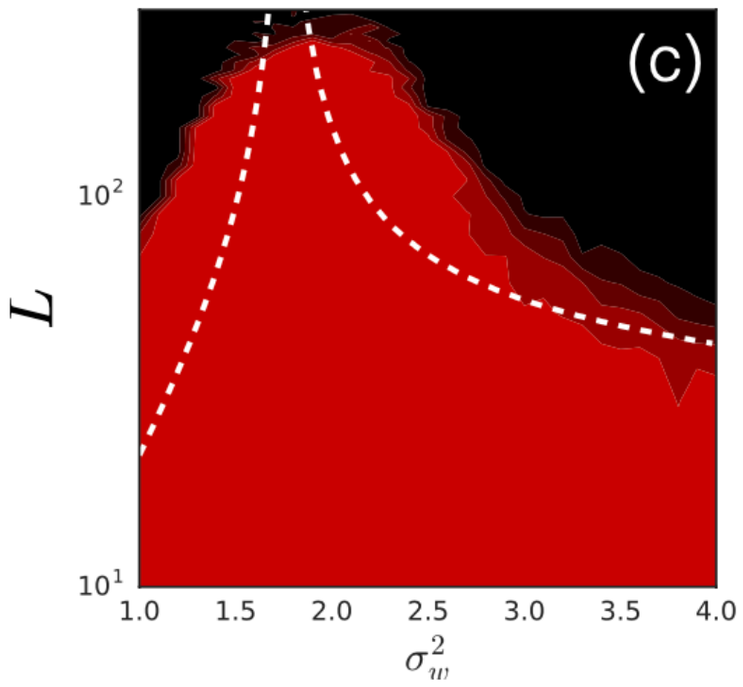
MNIST
45 / RMS

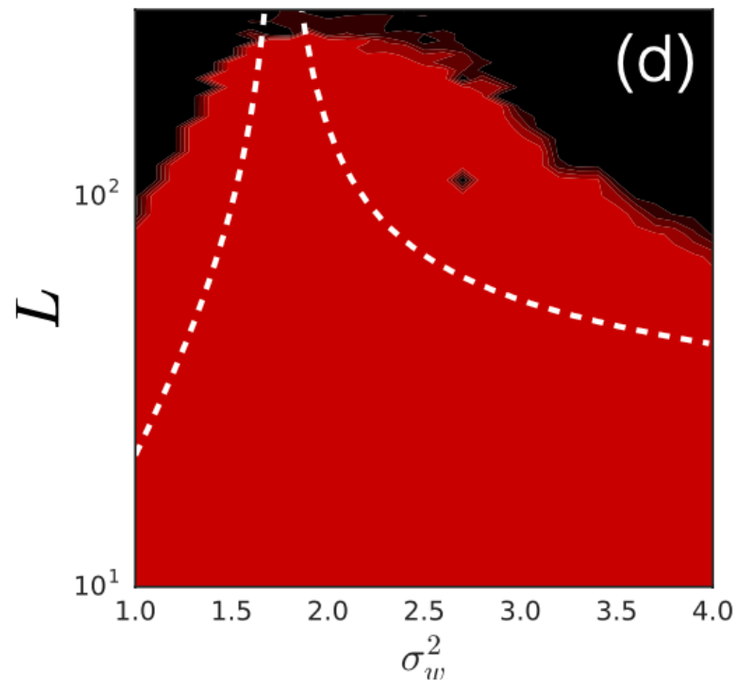




MNIST
304 / RMS

MNIST
2048 / RMS





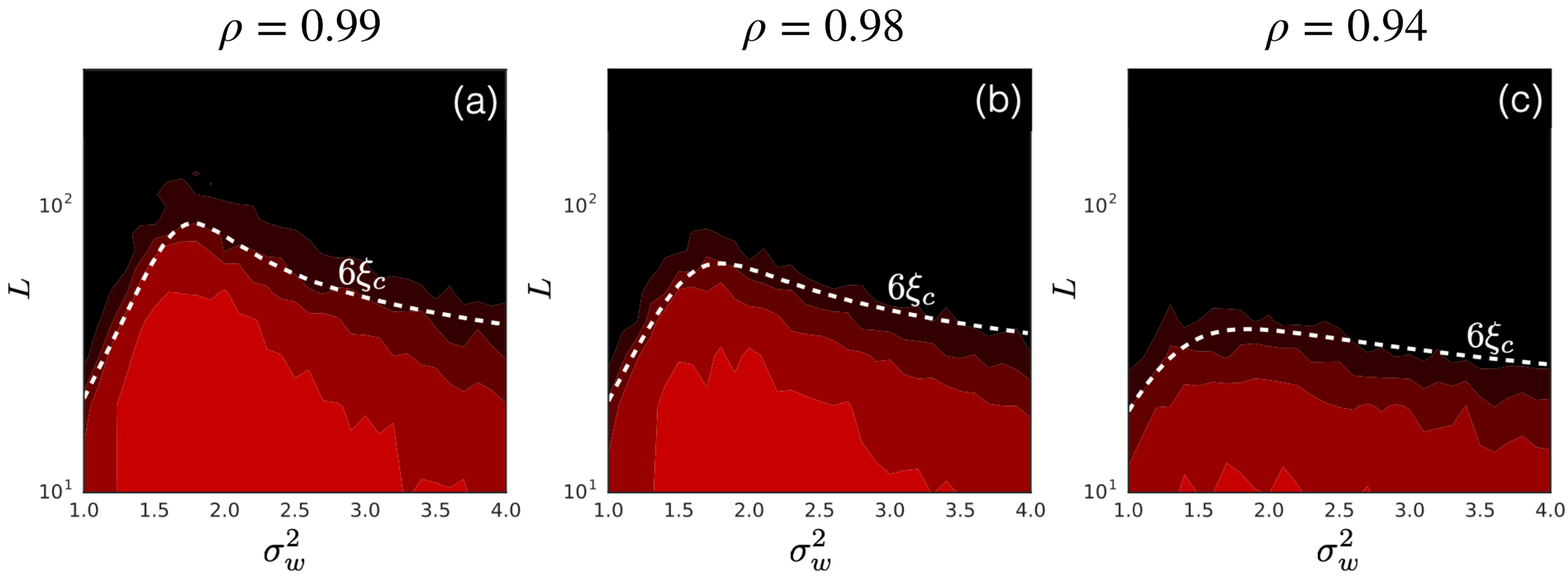
MNIST
13780 / RMS

Experiments

- Network is trainable when the network depth L is not much larger than the depth-scale ξ_c
  - This criterion is data independent and therefore offers a “universal” constraint on hyperparameters that depends on network architecture alone
- Networks closer to their critical point appear to train more quickly then those further away
- Near critical line, $c_{ab}^{l} \rightarrow c^{*}$ is significantly slower and non-exponential

Experiment (Dropout)

- Even infinitesimal amounts of dropout disrupt the order-to-chaos phase transition and cause the depth scale to become finite



Conclusion

- Same depth scales control signal propagation and gradient backpropagation in random networks
  - Ordered and chaotic phases correspond to regions of vanishing and exploding gradients, respectively
- Necessary condition for a random neural network to be trainable is information should be able to pass through it
  - Deep neural networks should be trainable provided they are initialized sufficiently close to the order-to-chaos transition
  - Depth-scales bound the set of hyperparameters that will lead to successful training
- Dropout destroys the order-to-chaos critical point and consequently removes the divergence in ξ_c

Conclusion

- At the transition between order and chaos, information stored in the correlation between inputs can propagate infinitely far through networks

- An alternative perspective as to why information stored in the covariance is crucial for training can be understood by appealing to the correspondence between infinitely wide **Bayesian neural networks** and **Gaussian Process**

  - In particular, the covariance q_{ab}^{l} is intimately related to the kernel of the induced Gaussian Process

4

Neural Networks as Gaussian Process

Related Works

- Deep Neural Networks as Gaussian Process, J. Lee and Y. Bahri et al., ICLR'18

Motivation

- Single-layer fully-connected neural network with an i.i.d. prior over its parameters is equivalent to a **Gaussian Process (GP)** in the limit of infinite width

- Enable exact Bayesian inference for infinite width neural networks on regression tasks by evaluating the corresponding GP

- The computation requires building the necessary covariance matrices over the training and test sets and straightforward linear algebra

Main Idea

- Derive the exact equivalent between infinitely wide deep networks and GPs

- Develop a computationally efficient pipeline to compute the covariance function for these GPs

GP and Single-layer Neural Networks

- Consider L-hidden-layer fully-connected neural networks with hidden layers of width N_l and pointwise nonlinearities ϕ
  - Weights and biases for the lth layer have component $W_{ij}^l \sim N(0, \frac{\sigma_w^2}{N_l})$ and $b_i^l \sim N(0, \sigma_b^2)$ respectively, which are independently and random drawn

GP and Single-layer Neural Networks

$$z_i^1(x) = b_i^1 + \sum_{j=1}^{N_1} W_{ij}^1 x_j^1(x) \qquad x_j^1(x) = \phi\Big(b_j^0 \sum_{k=1}^{d_{in}} W_{jk}^0 x_k\Big)$$

- Since $z_i^1(x)$ is a sum of i.i.d. terms, $z_i^1(x)$ will be Gaussian distributed according to Central Limit Theorem as $N_l \to \infty$
  - Weights and biases are taken to be i.i.d., thus the post-activation x_j^1 and $x_{j'}^1$ are independent for $j \neq j'$

- Likewise, any finite collection of $\{z_i^l(x^{\alpha=1}), \cdots, z_i^1(x^{\alpha=k})\}$ will have joint multivariate Gaussian distribution, which is exactly the definition of Gaussian Process

GP and Single-layer Neural Networks

- Therefore, we conclude that $z_i^l \sim GP(\mu^1, K^1)$, a GP with mean μ^1 and covariance K^1

$$\mu^1(x) = \mathbb{E}[z_i^1(x)] = 0$$

$$\begin{aligned} K^1(x, x') &\equiv \mathbb{E}[z_i^1(x) z_i^1(x')] \\ &= \sigma_b^2 + \sigma_w^2 \, \mathbb{E}[x_i^1(x) x_i^1(x')] \\ &\equiv \sigma_b^2 + \sigma_w^2 C(x, x') \end{aligned}$$

GP and Deep Neural Networks

- We proceed by taking the hidden layer widths to be infinite in succession ($N_1 \to \infty$, $N_2 \to \infty$, etc) as we continue induction, to guarantee the input to the layer under consideration is already **governed by a GP**

GP and Deep Neural Networks

- Suppose z_j^{l-1} is a GP. After $l-1$ steps, the network computes

$$z_i^l(x) = b_i^l + \sum_{j=1}^{N_l} W_{ij}^l x_j^l(x) \qquad\qquad x_j^l(x) = \phi(z_j^{l-1}(x))$$

- As before, $z_i^l(x)$ is sum of i.i.d. terms therefore it can be described as $z_i^l(x) \sim GP(0, K^l)$. The covariance is

$$\begin{aligned} K^l(x, x') &\equiv \mathbb{E}[z_i^l(x) z_i^l(x')] \\ &= \sigma_b^2 + \sigma_w^2 \, \mathbb{E}_{z_i^{l-1} \sim GP(0,\, K^{l-1})}[\phi(z_i^{l-1}(x))\phi(z_i^{l-1}(x'))] \end{aligned}$$

GP and Deep Neural Networks

- Integrating against the joint distribution of $z_i^{l-1}(x)$ and $z_i^{l-1}(x')$ can be described by a zero mean, two-dimensional Gaussian whose covariance matrix has distinct entries $K^{l-1}(x,x')$, $K^{l-1}(x,x)$ and $K^{l-1}(x',x')$

$$
\begin{aligned}
K^l(x,x') &\equiv \mathbb{E}[z_i^l(x) z_i^l(x')] \\
&= \sigma_b^2 + \sigma_w^2 \, \mathbb{E}_{z_i^{l-1} \sim GP(0, K^{l-1})}[\phi(z_i^{l-1}(x))\phi(z_i^{l-1}(x'))] \\
&= \sigma_b^2 + \sigma_w^2 F_\phi(K^{l-1}(x,x'), K^{l-1}(x,x), K^{l-1}(x',x'))
\end{aligned}
$$

- To emphasize the recurrence relationship between K^l and K^{l-1} via a deterministic function F whose form depends only on the nonlinearity ϕ

- For certain activations, such as tanh and ReLU, it can be computed analytically. However, when no analytics form exists, it have to be computed numerically

Bayesian Inference

- Given a dataset $D = \{(x^1, t^1), (x^2, t^2), \cdots, (x^n, t^n)\}$, we wish to make a Bayesian prediction at test point x^* using a distribution over functions $z(x)$

$$p(z^* \mid D, x^*) = \int dz\, P(z^* \mid \mathbf{z}, \mathbf{x}, x^*) p(\mathbf{z} \mid D)$$

$$= \frac{1}{P(\mathbf{t})} \int dz\, P(z^*, \mathbf{z} \mid x^*, \mathbf{x}) P(\mathbf{t} \mid \mathbf{z})$$

- , where $P(\mathbf{t} \mid z)$ corresponds to observation noise

Bayesian Inference

- If $z^1, \cdots z^n, z^*$ are drawn from a GP, then $z^*, \mathbf{z} \,|\, x^*, \mathbf{x} \sim N(0, K)$ is a multivariate Gaussian with covariance matrix

Train Train v.s. Test

$$K = \begin{bmatrix} K_{D,D} & K_{x^*,D}^\top \\ K_{x^*,D} & K_{x^*,x^*} \end{bmatrix}$$

Train v.s. Test Test

- As is standard in GPs, we can get $P(z^* \,|\, D, x^*) \sim N(\bar{\mu}, \bar{K})$ with

$$\bar{\mu} = K_{x^*,D}(K_{D,D} + \sigma_w^2 \mathbb{I}_n)^{-1}\mathbf{t}$$

$$\bar{K} = K_{x^*,x^*} - K_{x^*,D}(K_{D,D} + \sigma_w^2 \mathbb{I}_n)^{-1}K_{x^*,D}^\top$$

Bayesian Inference

- The predicted distribution for $P(z^* \mid D, x^*)$ is determined from straightforward matrix computations, corresponding to **fully Bayesian training** of deep neural network

- The form of covariance function used is determined by the choice of **GP prior**, i.e. the neural network model class, which depends on **depth**, **nonlinearity**, and **weight** and **bias variance**

GP Kernel

- Given an L-layer deep neural network with fixed hyperparameters, constructing matrix K^L for the equivalent GP involves computing the Gaussian integral in

$$K^l(x, x') = \sigma_b^2 + \sigma_w^2 \mathbb{E}_{z_i^{l-1} \sim GP(0, K^{l-1})}[\phi(z_i^{l-1}(x))\phi(z_i^{l-1}(x'))]$$

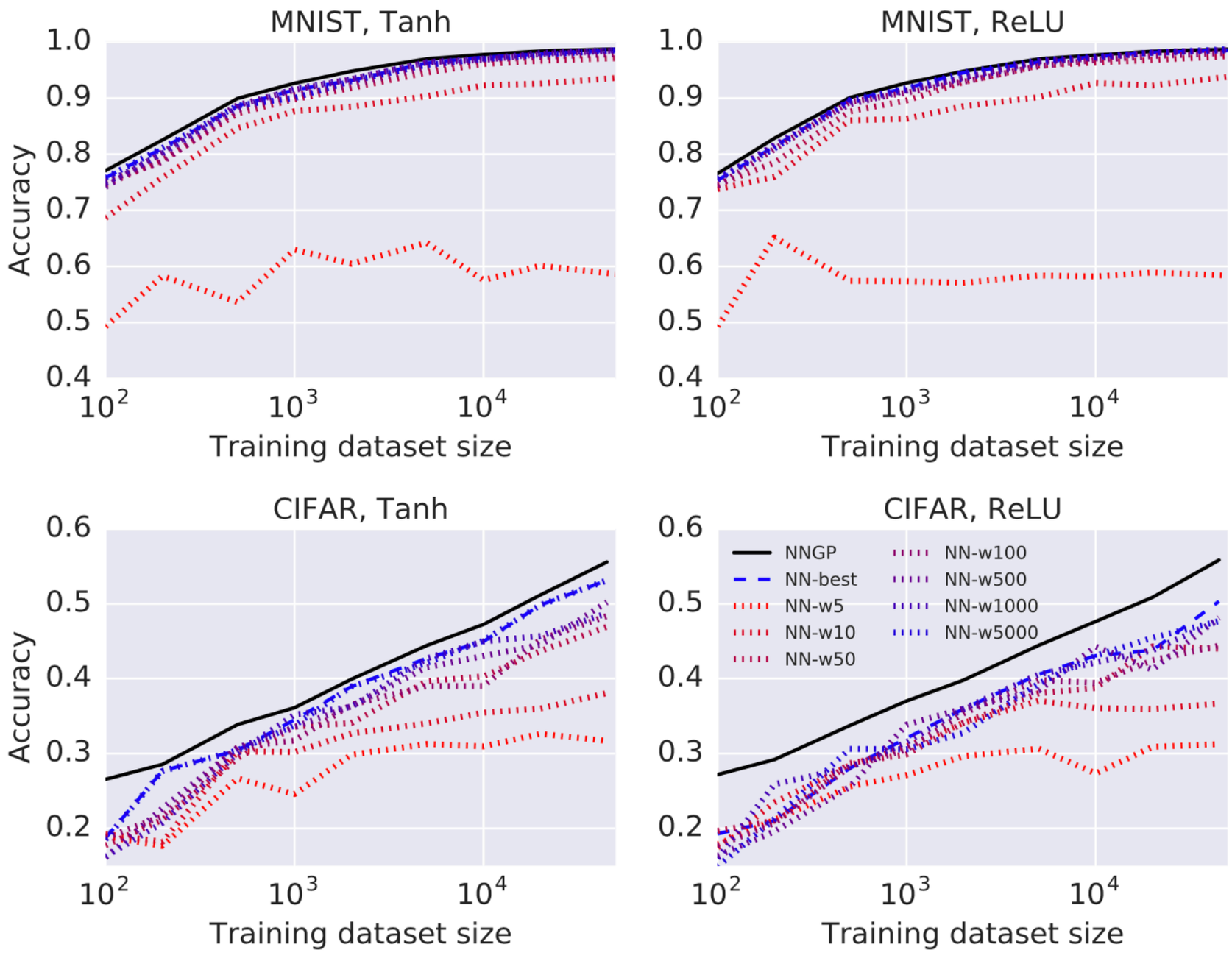
- for all pairs of training-training and training-test points, recursively for all layers
    - For some nonlinearities, such as ReLU, this integration can be done analytically. However, to compute kernel corresponding to arbitrary nonlinearities, the integral must be performed numerically

GP Kernel

- By careful pipelining and preprocessing all inputs to have identical norm, we can speed up this computation
  - Full computational pipeline is **deterministic** and **differentiable**. The shape and properties of a deep network kernel are purely determined by hyperparemeters of the deep neural network
  - Since GPs give exact marginal likelihood estimates, this kernel construction may allow principled hyperparameter selection, or nonlinearity design
  - Constructing K_{DD} for each layer took 90-140s. Solving linear equations via Cholesky decomposition took 180-220s for 1000 test point

Experiment

- NNGP (Neural Network Gaussian Process) often outperforms trained finite width networks



Experiment

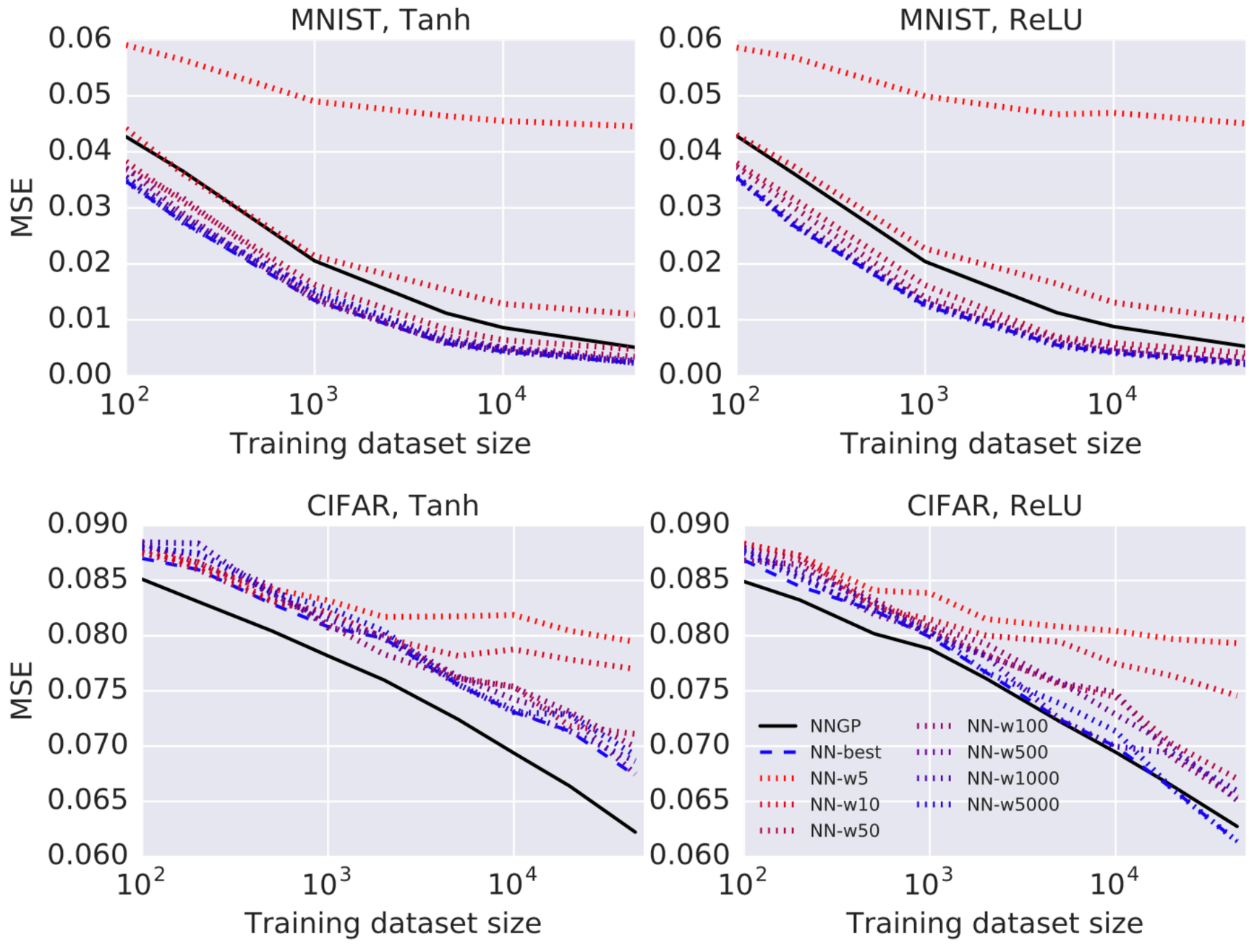
- NNGP (Neural Network Gaussian Process) often outperforms trained finite width networks

NN-depth-width-σ_w^2-σ_b^2 GP-depth-σ_w^2-σ_b^2

Num training	Model (ReLU)	Test accuracy	Model (tanh)	Test accuracy
MNIST:1k	NN-2-5000-3.19-0.00	0.9252	NN-2-1000-0.60-0.00	0.9254
	GP-20-1.45-0.28	**0.9279**	GP-20-1.96-0.62	0.9266
MNIST:10k	NN-2-2000-0.42-0.16	0.9771	NN-2-2000-2.41-1.84	0.9745
	GP-7-0.61-0.07	0.9765	GP-2-1.62-0.28	**0.9773**
MNIST:50k	NN-2-2000-0.60-0.44	0.9864	NN-2-5000-0.28-0.34	0.9857
	GP-1-0.10-0.48	0.9875	GP-1-1.28-0.00	**0.9879**
CIFAR:1k	NN-5-500-1.29-0.28	0.3225	NN-1-200-1.45-0.12	0.3378
	GP-7-1.28-0.00	0.3608	GP-50-2.97-0.97	**0.3702**
CIFAR:10k	NN-5-2000-1.60-1.07	0.4545	NN-1-500-1.48-1.59	0.4429
	GP-5-2.97-0.28	**0.4780**	GP-7-3.48-2.00	0.4766
CIFAR:45k	NN-3-5000-0.53-0.01	0.5313	NN-2-2000-1.05-2.08	0.5034
	GP-3-3.31-1.86	**0.5566**	GP-3-3.48-1.52	0.5558

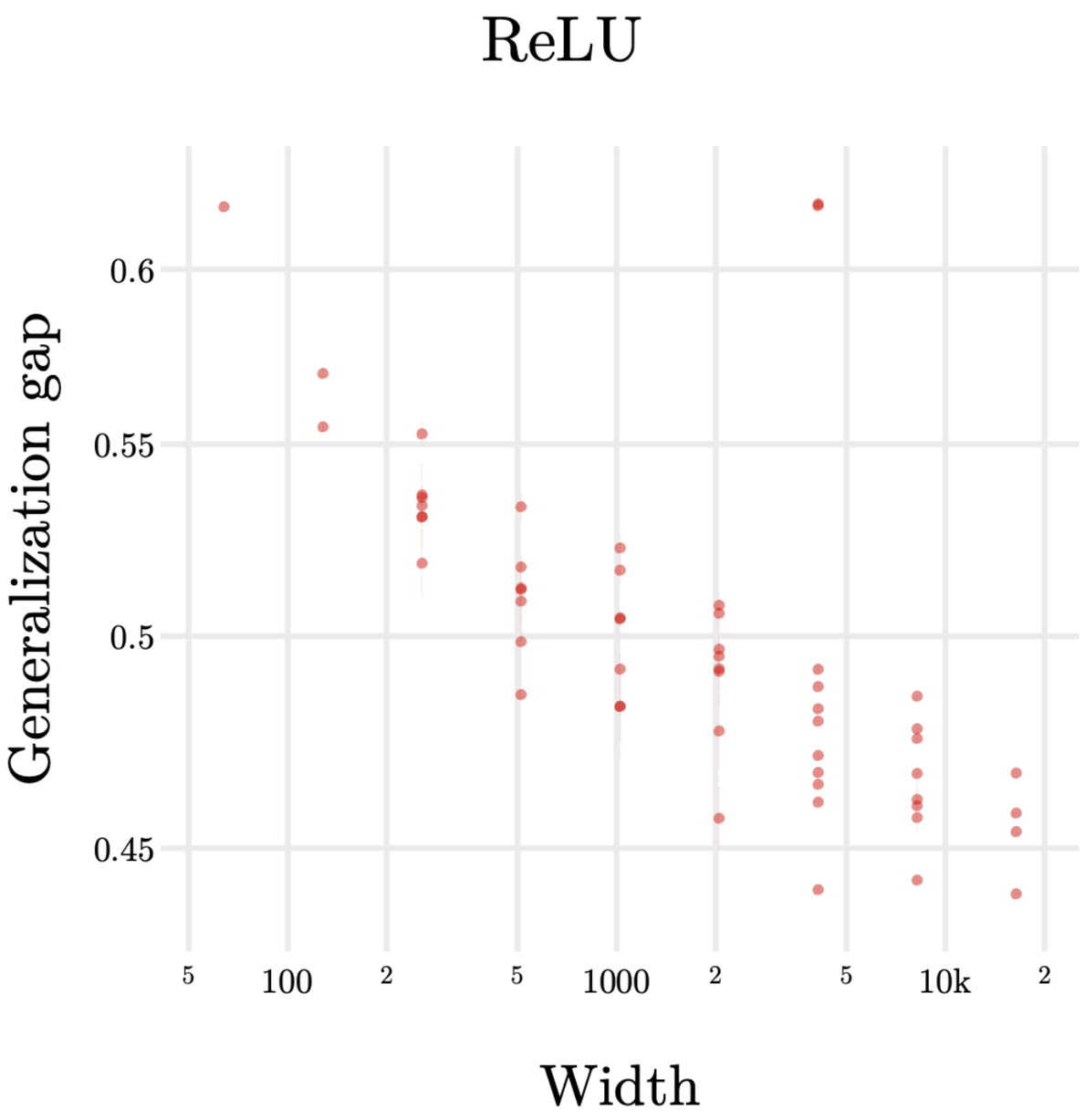
Experiment

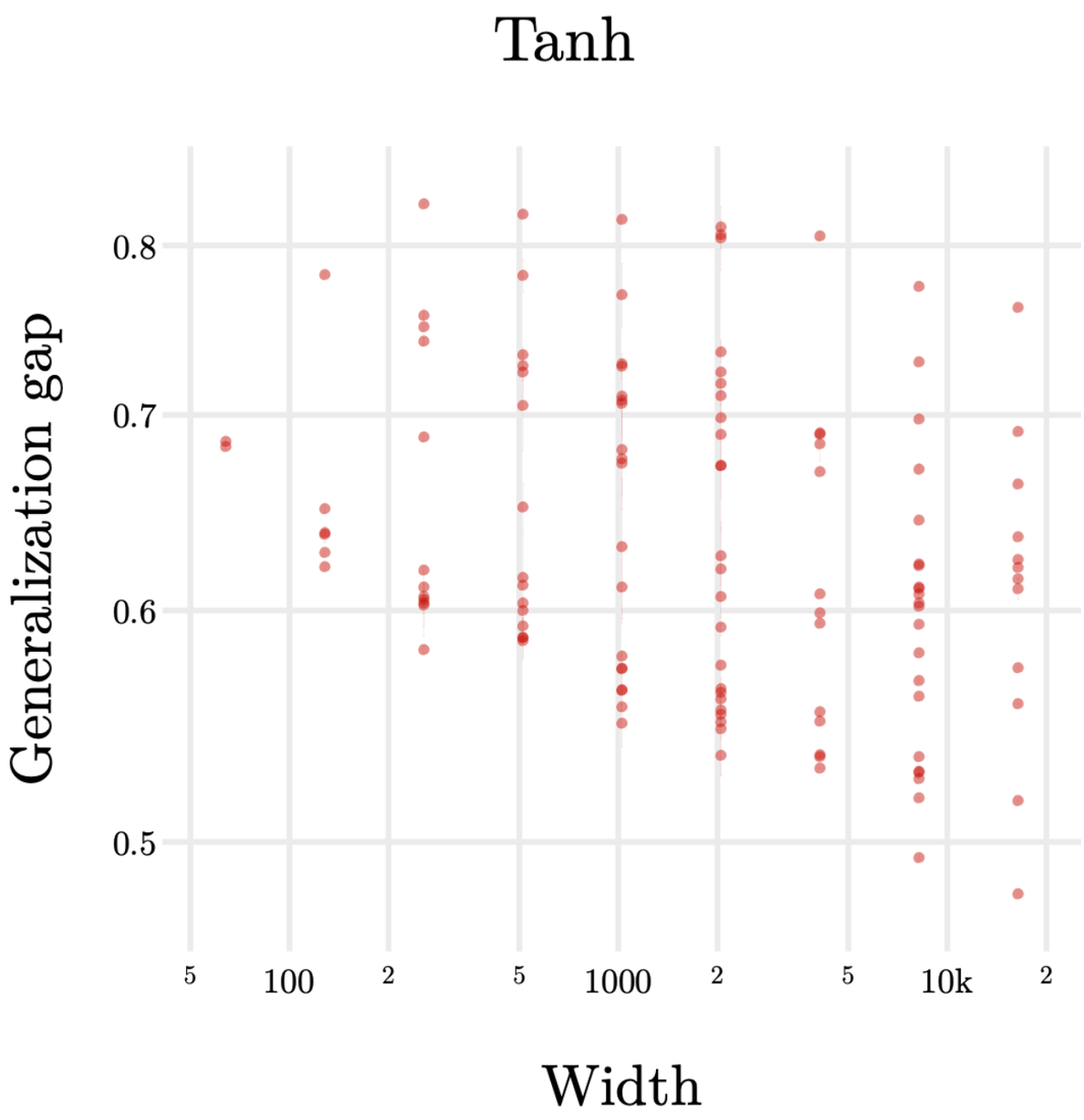
- NNGP (Neural Network Gaussian Process) often outperforms trained finite width networks



Generalization Gap

- The best generalizing networks are consistently the widest

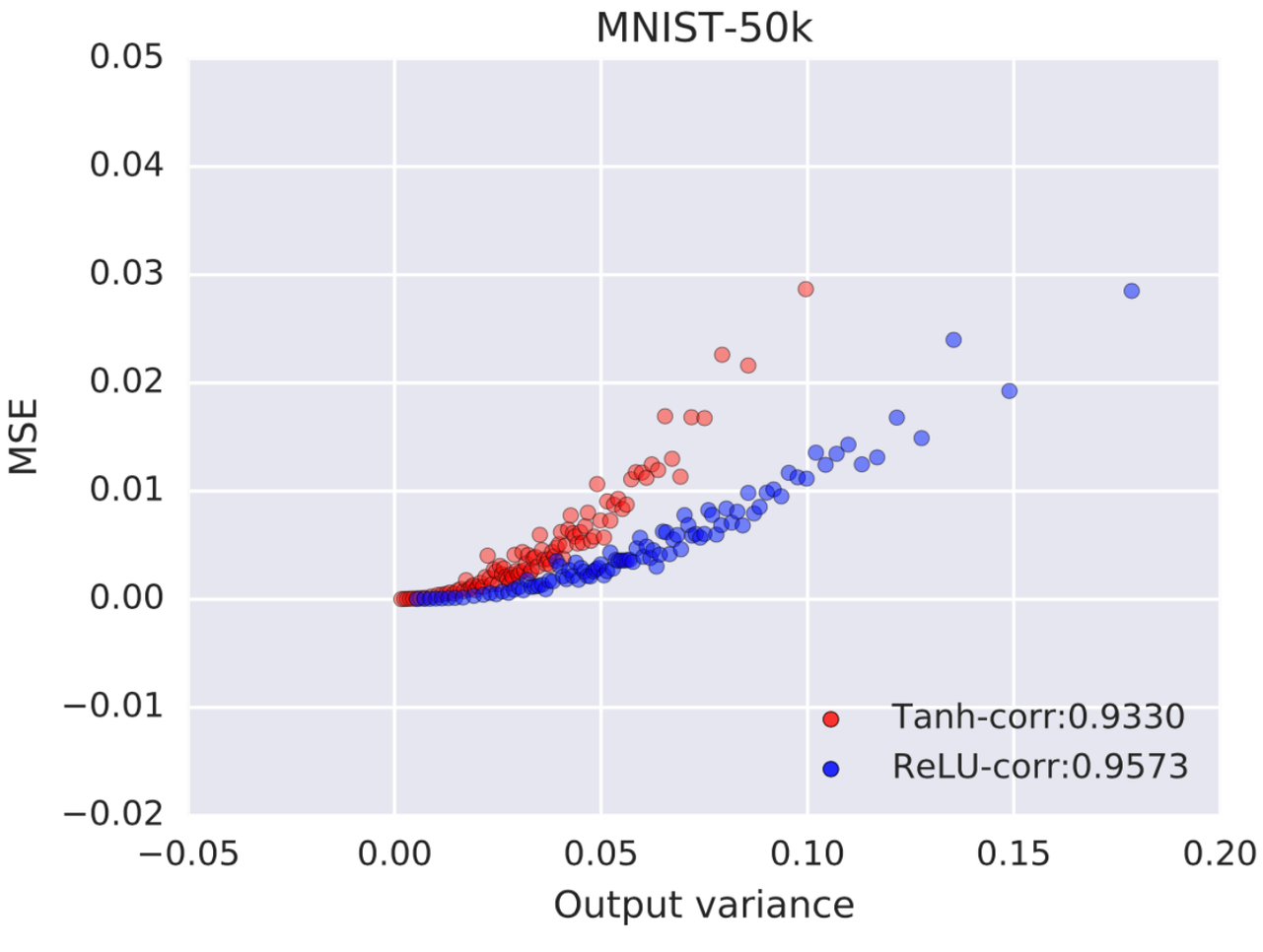


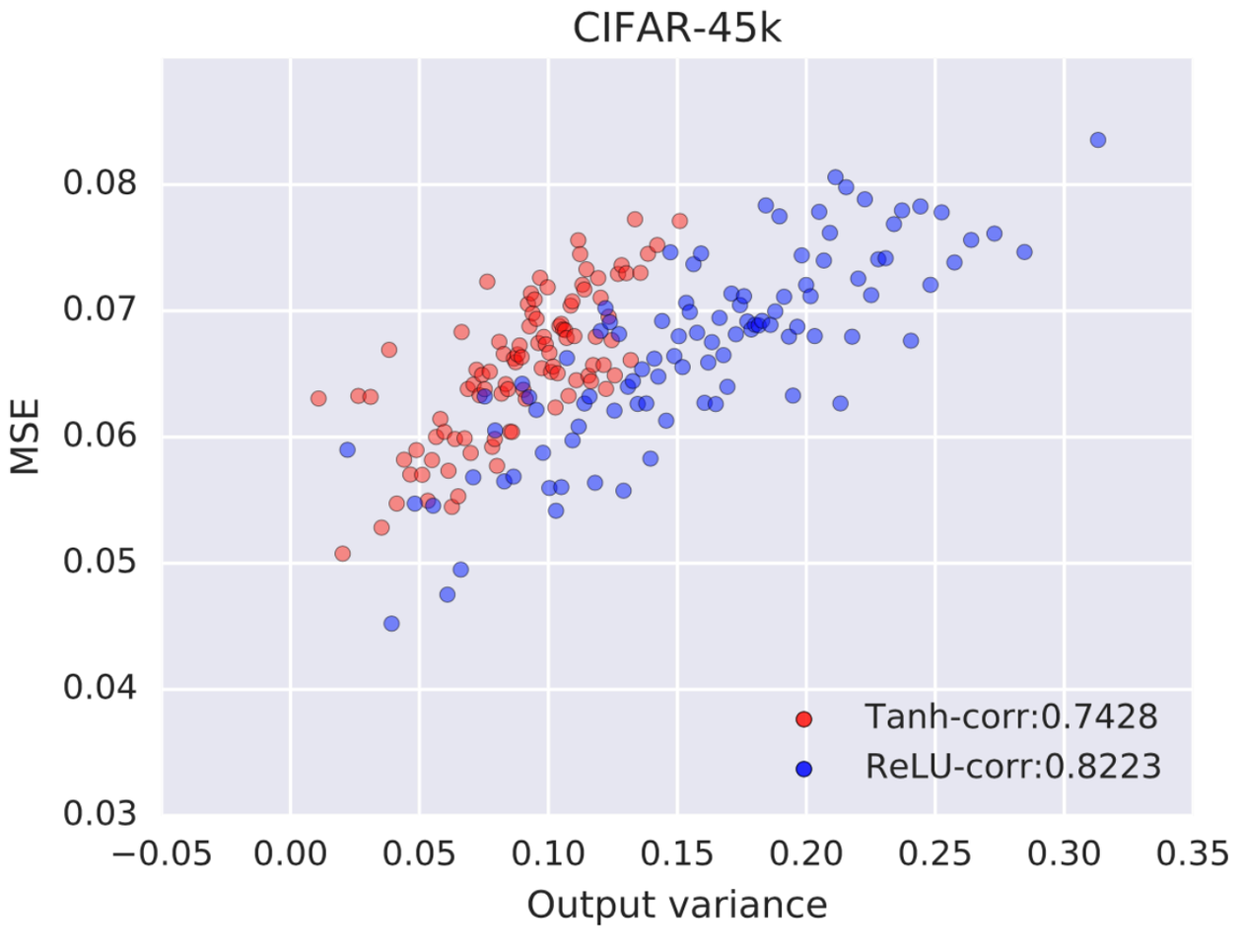


Uncertainty

- Due to Bayesian nature, all predictions have uncertainty estimates, while for neural networks, capturing the uncertainty is challenging

- NNGP uncertainty estimate is highly correlated with prediction error





Experiment

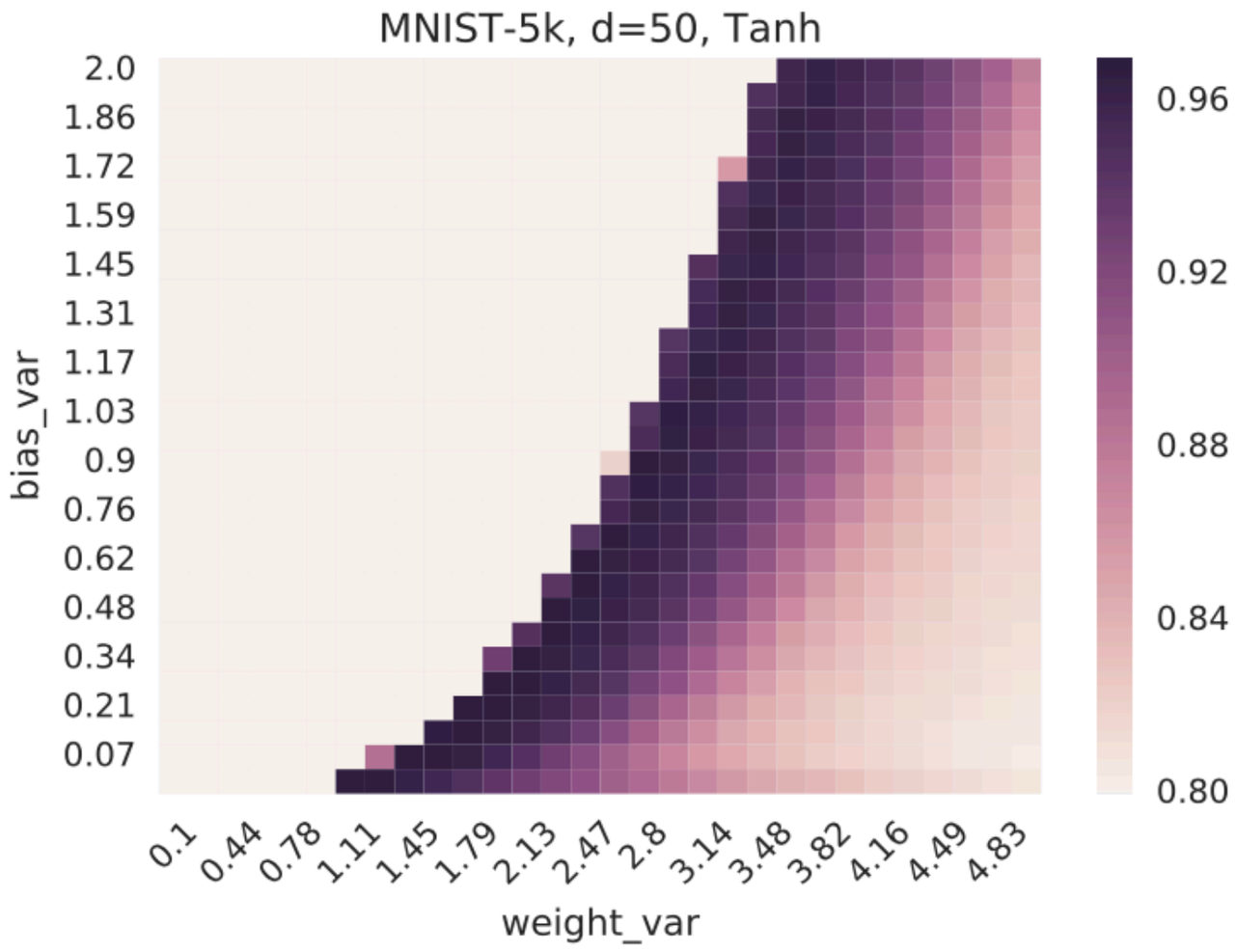
- The performance of the best finite-width NNs, trained with a variant of SGD, approaches that of the NNGP with increasing layer width
  - NNs are commonly believed to be powerful because of their ability to do flexible representation learning, while NNGP uses fixed basis function. There is no salient performance advantage to the former
  - Were the neural networks trained in fully Bayesian fashion, rather than by SGD, the approach to NNPG in the large width limit would be guaranteed?
- The widest NNs have better generalizability
  - Limit of infinite network width, which inherent to the GP, is far from disadvantage

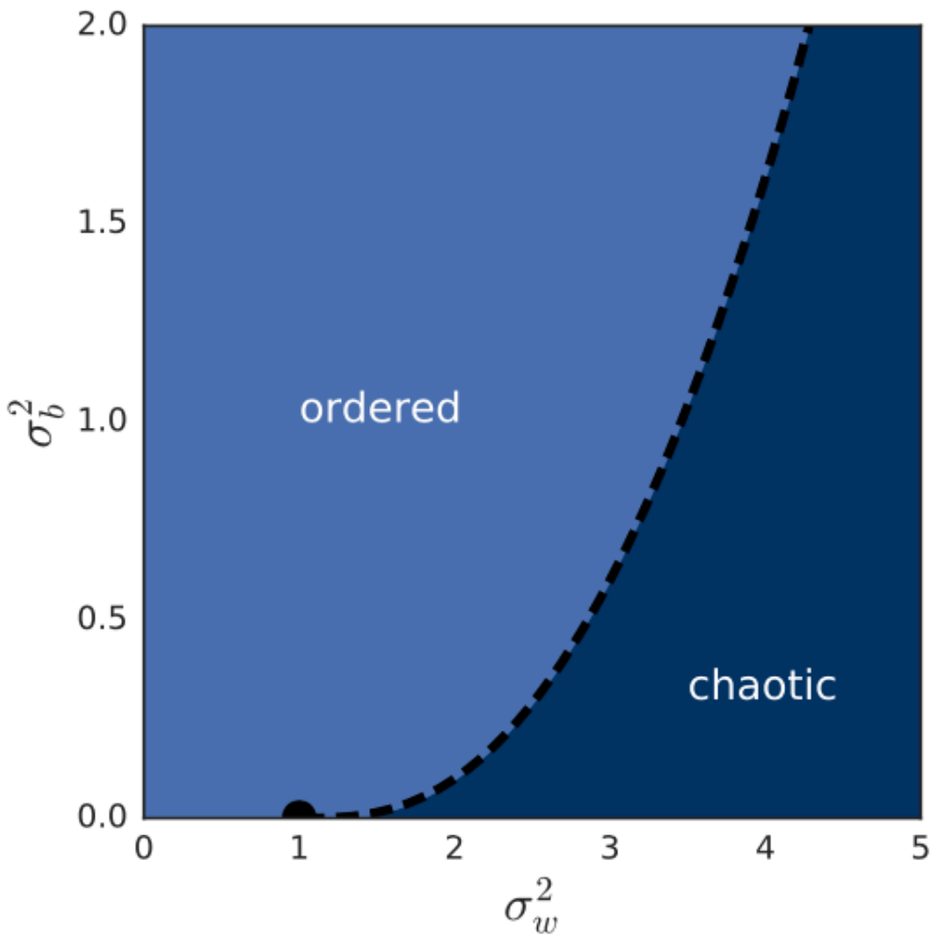
Deep Signal Propagation

- Several prior works have noted the recurrence relations commonly approach a functionally uninteresting **fixed point** with depth $l \to \infty$, in that $K^{\infty}(x, x')$ becomes a constant or piecewise constant map

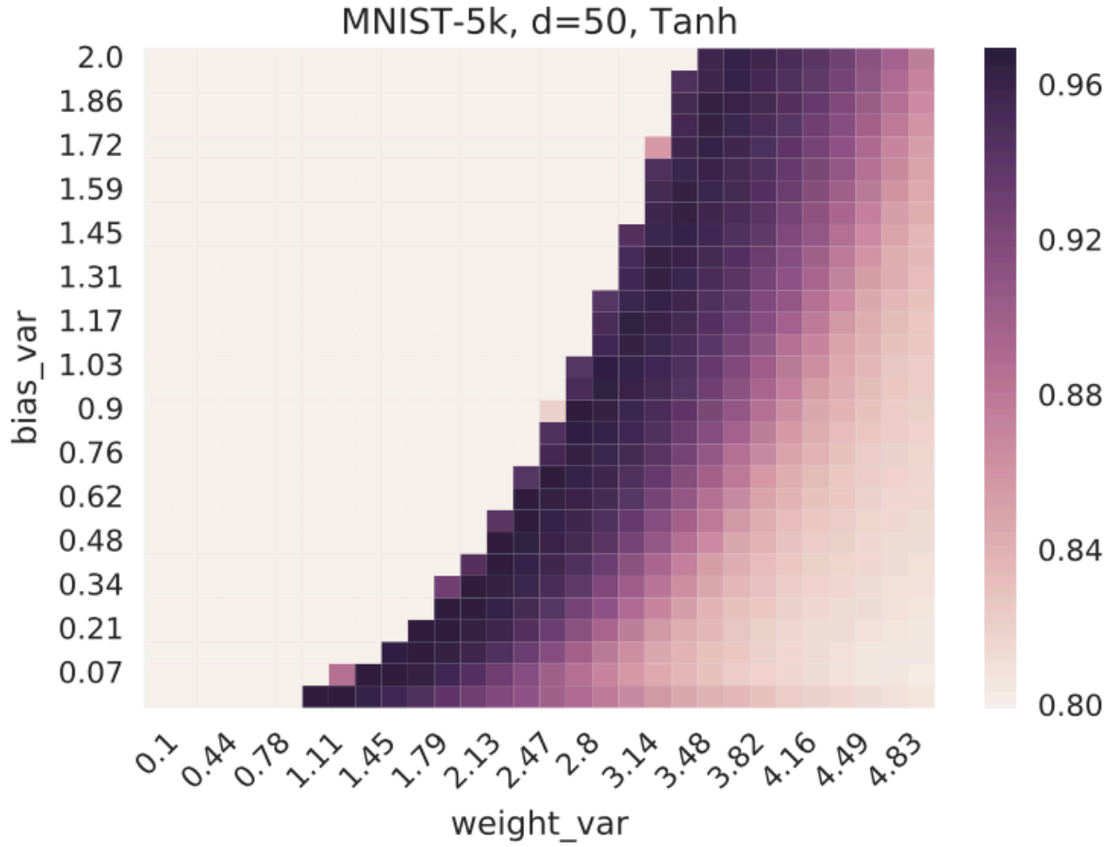
Deep Signal Propagation

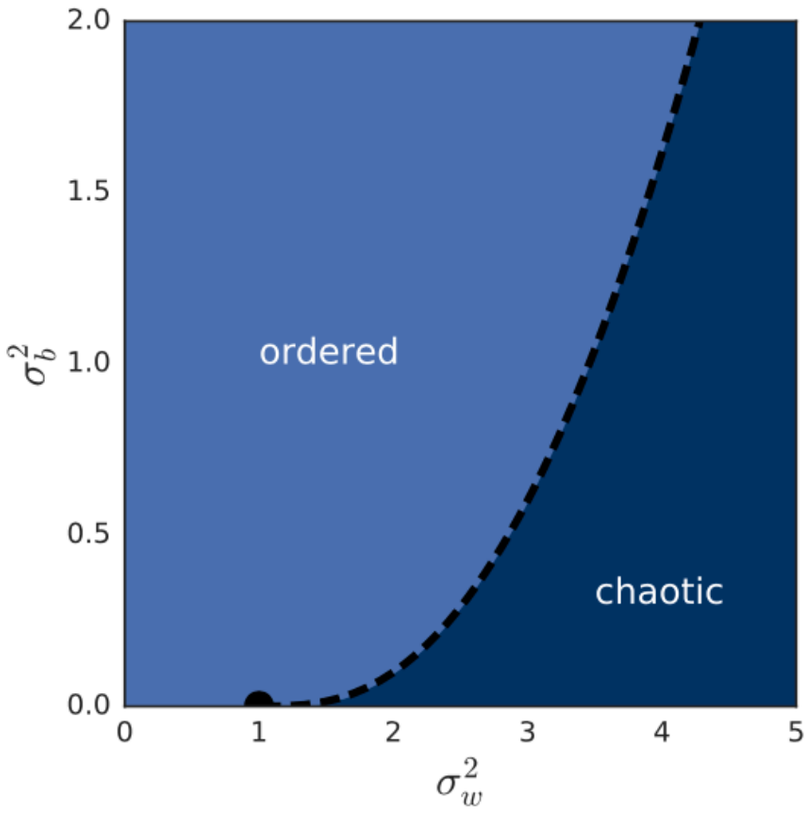
- The best performing NNGP hyperparameters agree with predictions by deep signal propagation, along critical line



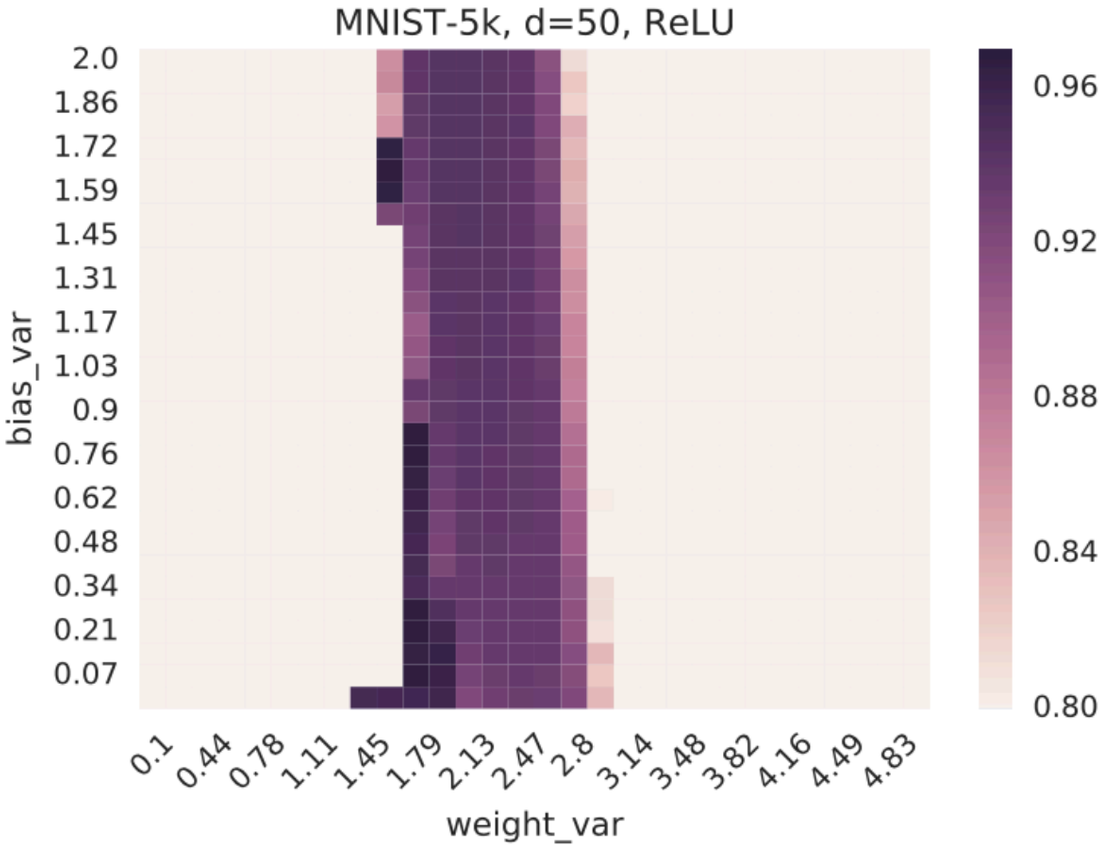


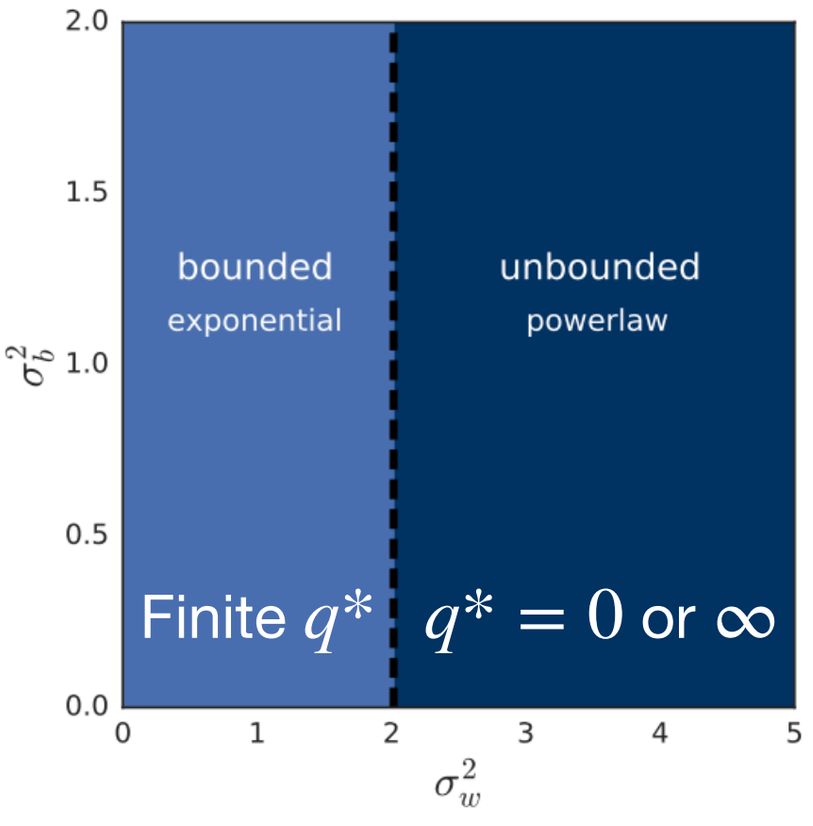
Tanh


MNIST-5k, d=50, Tanh
bias_var
2.0
1.86
1.72
1.59
1.45
1.31
1.17
1.03
0.9
0.76
0.62
0.48
0.34
0.21
0.07
0.1
0.44
0.78
1.11
1.45
1.79
2.13
2.47
2.8
3.14
3.48
3.82
4.16
4.49
4.83
weight_var
0.96
0.92
0.88
0.84
0.80


ordered
chaotic
σ_b^2
σ_w^2
0.0
0.5
1.0
1.5
2.0
0
1
2
3
4
5

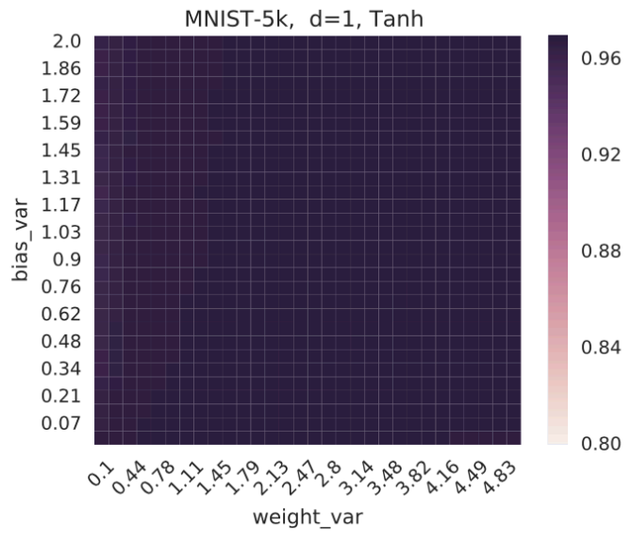
ReLU

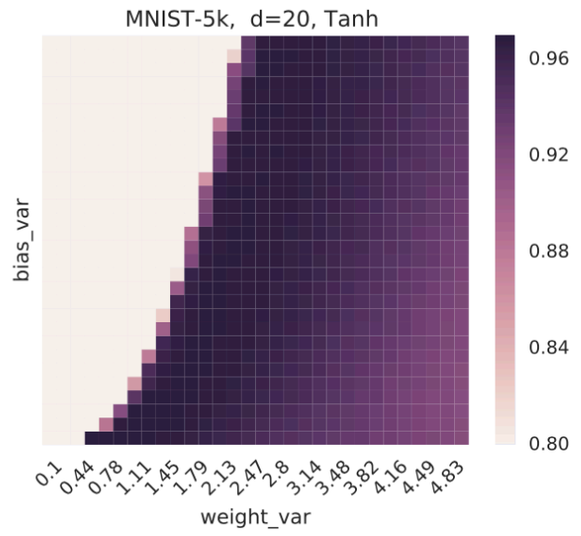

MNIST-5k, d=50, ReLU
bias_var
2.0
1.86
1.72
1.59
1.45
1.31
1.17
1.03
0.9
0.76
0.62
0.48
0.34
0.21
0.07
0.1
0.44
0.78
1.11
1.45
1.79
2.13
2.47
2.8
3.14
3.48
3.82
4.16
4.49
4.83
weight_var
0.96
0.92
0.88
0.84
0.80

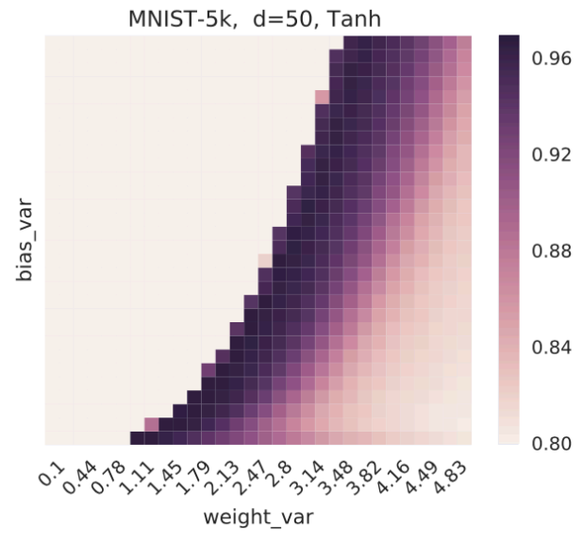

bounded
exponential
unbounded
powerlaw
Finite q^*
$q^* = 0$ or ∞
σ_b^2
σ_w^2
0.0
0.5
1.0
1.5
2.0
0
1
2
3
4
5

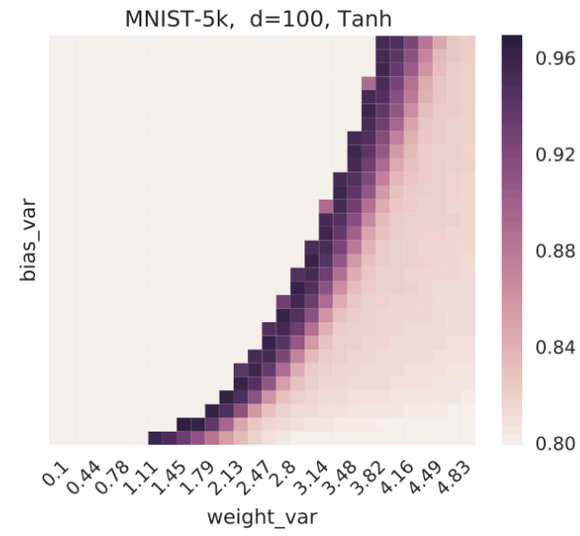
Deep Signal Propagation

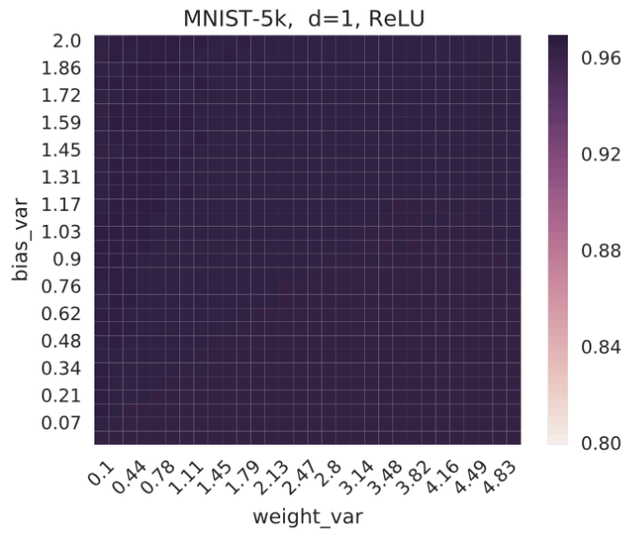
- Accuracy appears to drop more quickly away from phase boundary with increase in depth L of the GP kernel K^L
  - Information will be available through the difference $K^L(x, x') - K^\infty(x, x')$. As the depth gets larger, this difference becomes smaller

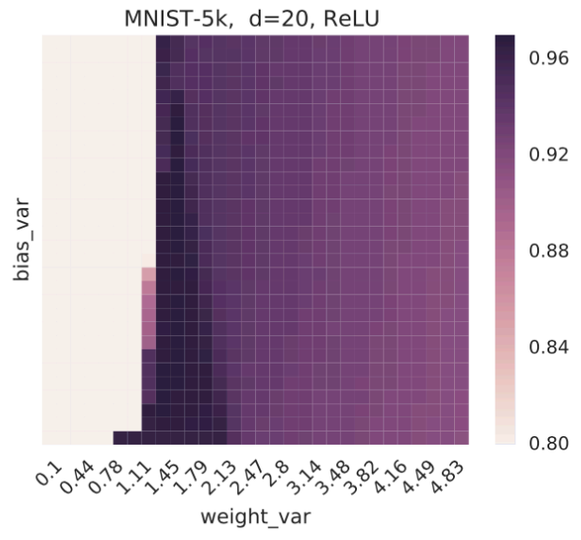


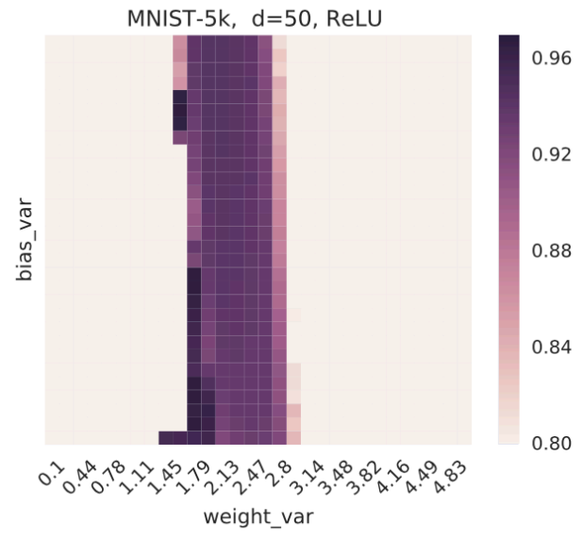


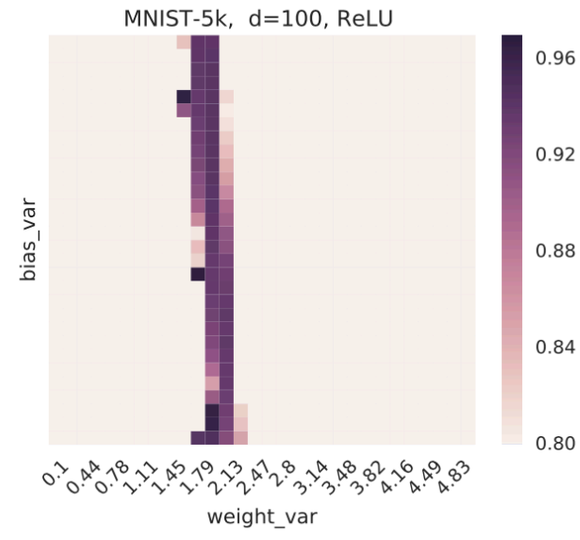












Conclusion

- By harnessing the limit of infinite width, we have specified a correspondence between priors on deep neural networks and Gaussian Process
  - GP kernel is constructed in a compositional, but fully deterministic and differentiable
- Trained neural network accuracy approaches that of corresponding GP with increasing layer width
  - GP predictions typically outperform finite-width networks
- GP uncertainty is strongly correlated with trained neural network prediction error

5

Neural Tangent Kernel

Related Works

- Neural Tangent Kernel: Convergence and Generalization in Neural Networks, A. Jacot et al., NeurIPS'18

- Disentangling Trainability and Generalization in Deep Neural Networks, L. Xiao et al., ICML'20

Related Works

- Neural Tangent Kernel: Convergence and Generalization in Neural Networks, A. Jacot et al., NeurIPS'18

- Disentangling Trainability and Generalization in Deep Neural Networks, L. Xiao et al., ICML'20

Motivation

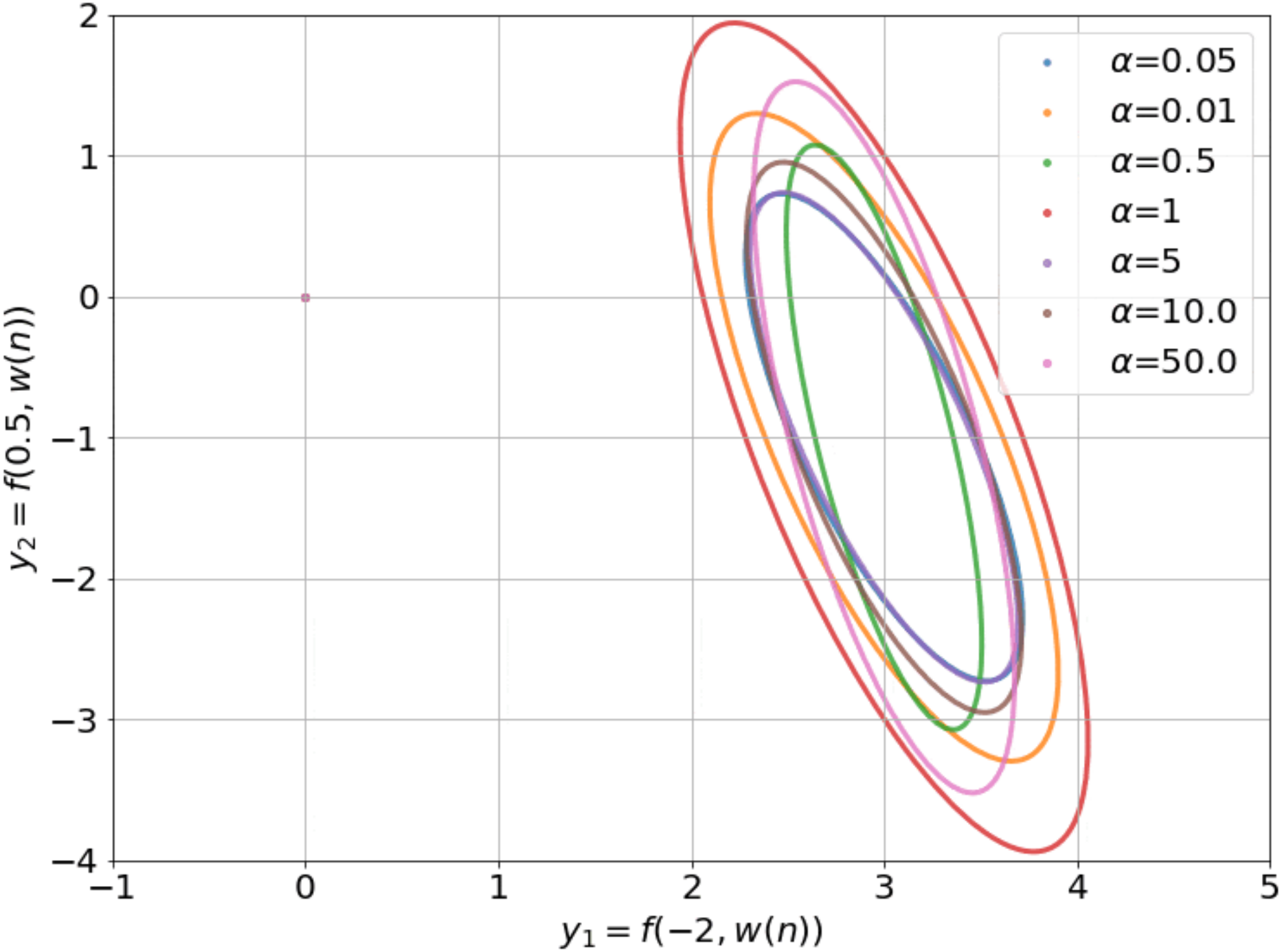
- At initialization, Artificial Neural Network (ANNs) are equivalent to Gaussian Process in the infinite-width limit, thus connecting to kernel methods

- How about during training? Is it possible to model the training dynamics of neural network?

Main Idea

- Evolution of an ANN during training can also be described by a kernel
  - During gradient descent on the parameters of an ANN, the network function f_θ follows the kernel gradient of the function cost w.r.t. a new kernel, the **Neural Tangent Kernel (NTK)**
- NTK is random at initialization and varies during training, in the infinite-width limit it **converges** to an explicit limiting kernel and **stay constant** during training

Training Dynamics of Neural Network



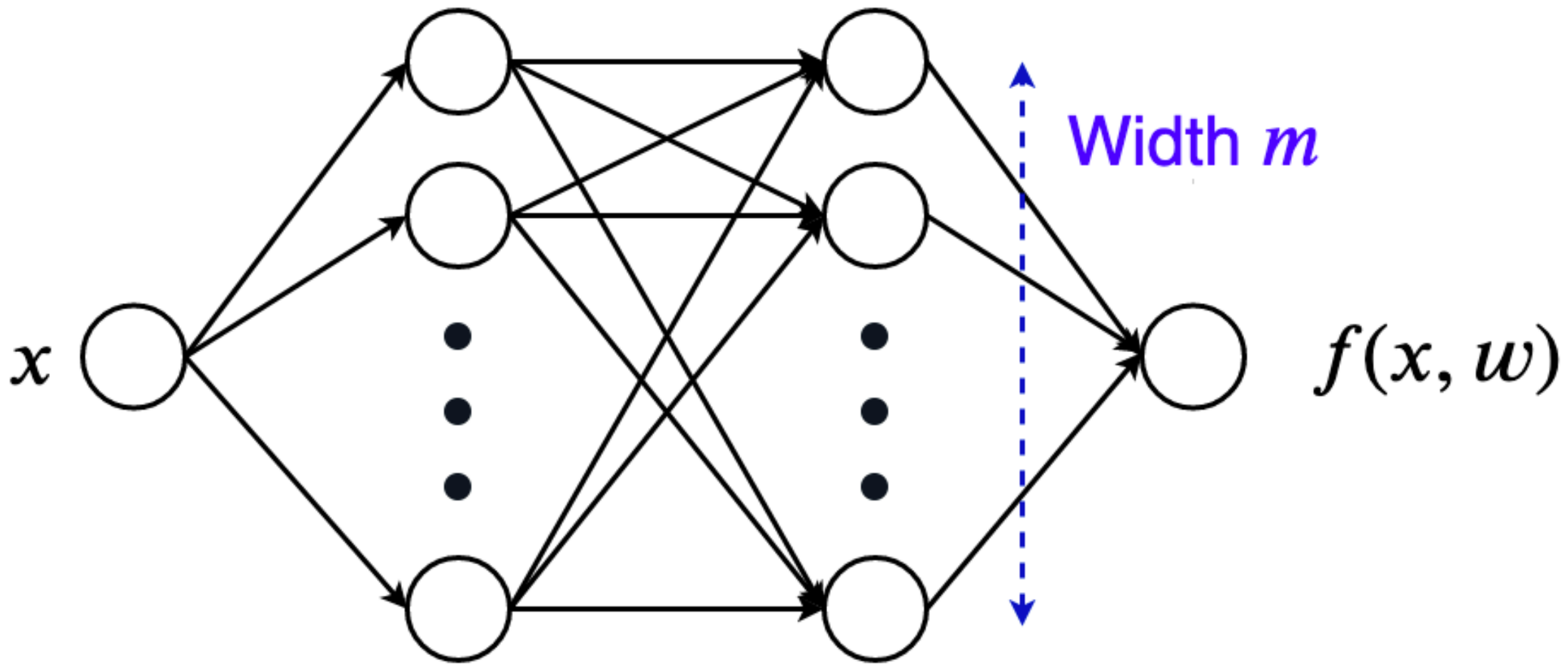
High-level Overview

- In the limit of infinite-width, neural networks simplify to **linear models** with a kernel called **Neural Tangent Kernel**

- Gradient descent is therefore simple to study

Setup

- Start with a 1-D input and 1-D output network, which is a simple 2-hidden layer ReLU network with width m



Setup

- Let's call this neural network $f(x, w)$ where x and w are input and weight vectors respectively. $D = \{\bar{x}_i, \bar{y}_i\}_{i=1}^{N}$ represents dataset

- For learning the network, we perform full-batch gradient descent on the least squares loss

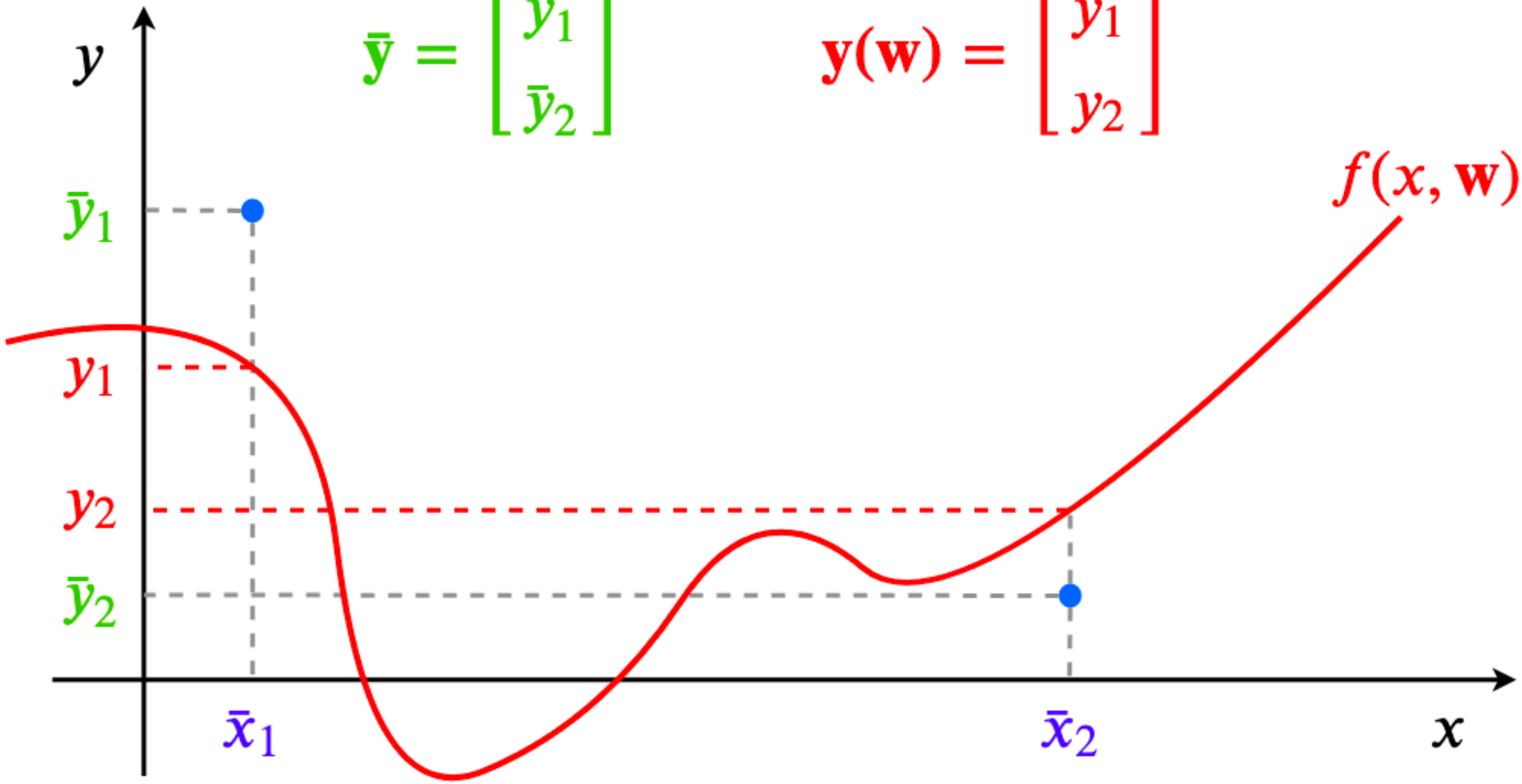
$$L(w) = \frac{1}{N} \sum_{i=1}^{N} \frac{1}{2} (f(\bar{x}_i, w) - \bar{y}_i)^2$$

- We can simplify this using some vector notation

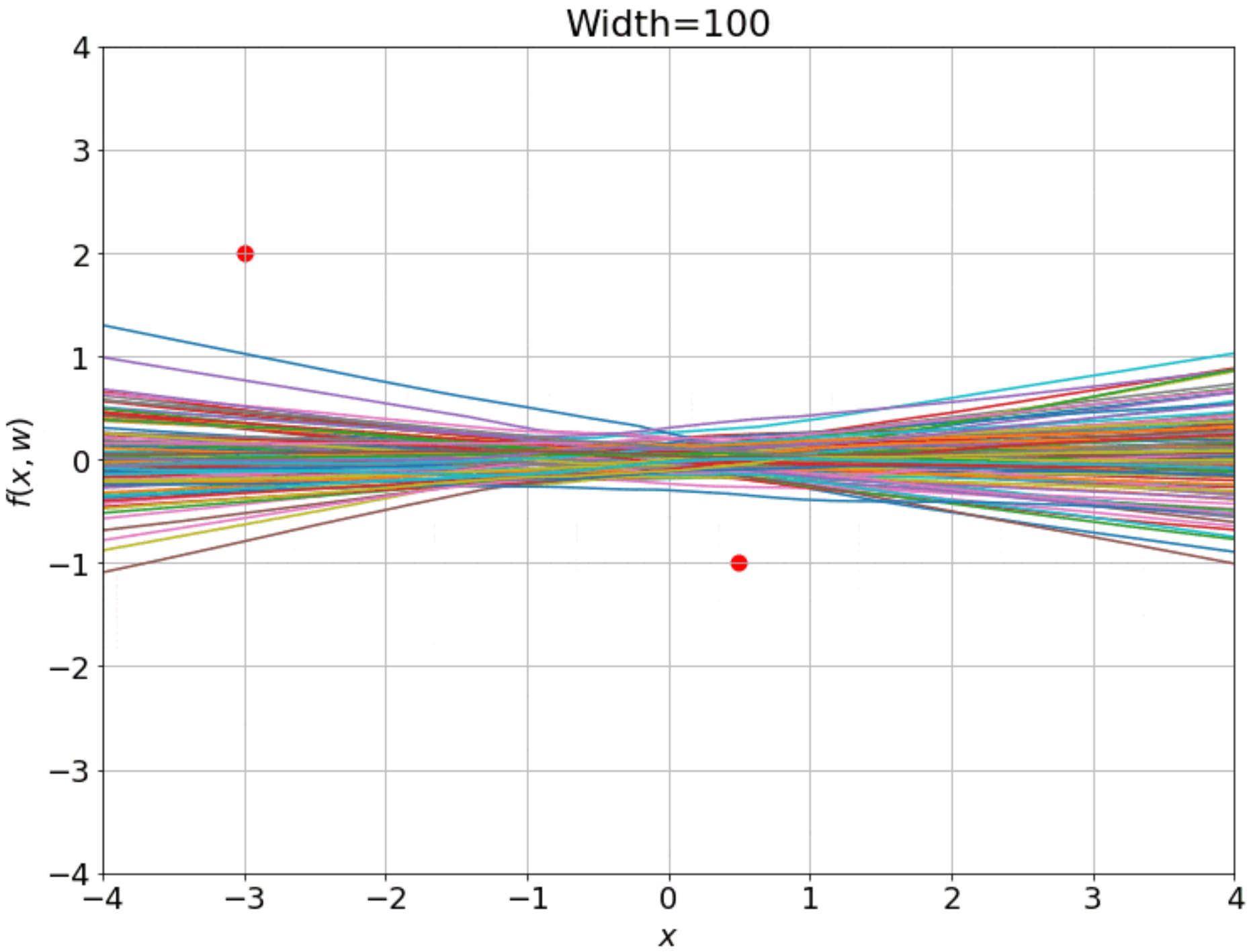
$$L(w) = \frac{1}{2} \|y(w) - \bar{y}\|_2^2 \qquad y(w)_i = f(\bar{x}_i, w)$$

Setup

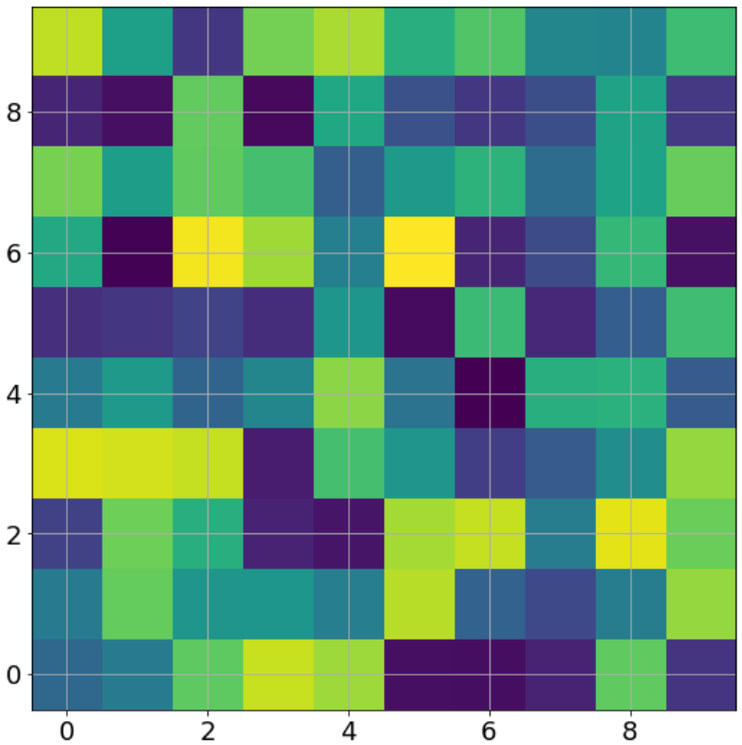
$$\bar{\mathbf{y}} = \begin{bmatrix} \bar{y}_1 \\ \bar{y}_2 \end{bmatrix} \qquad \mathbf{y}(\mathbf{w}) = \begin{bmatrix} y_1 \\ y_2 \end{bmatrix}$$

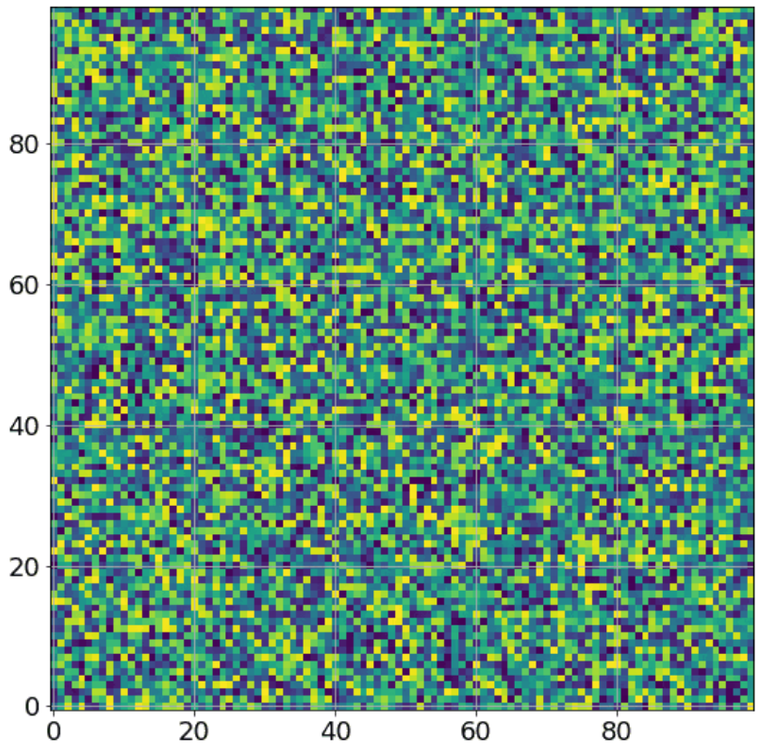


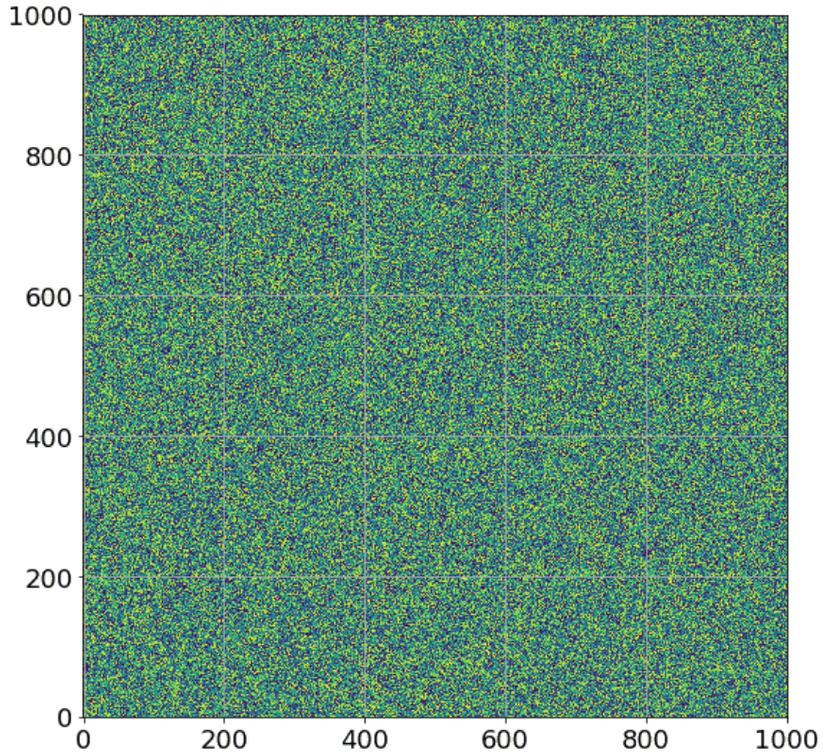
Training with Gradient Descent



Weight Matrices during Training





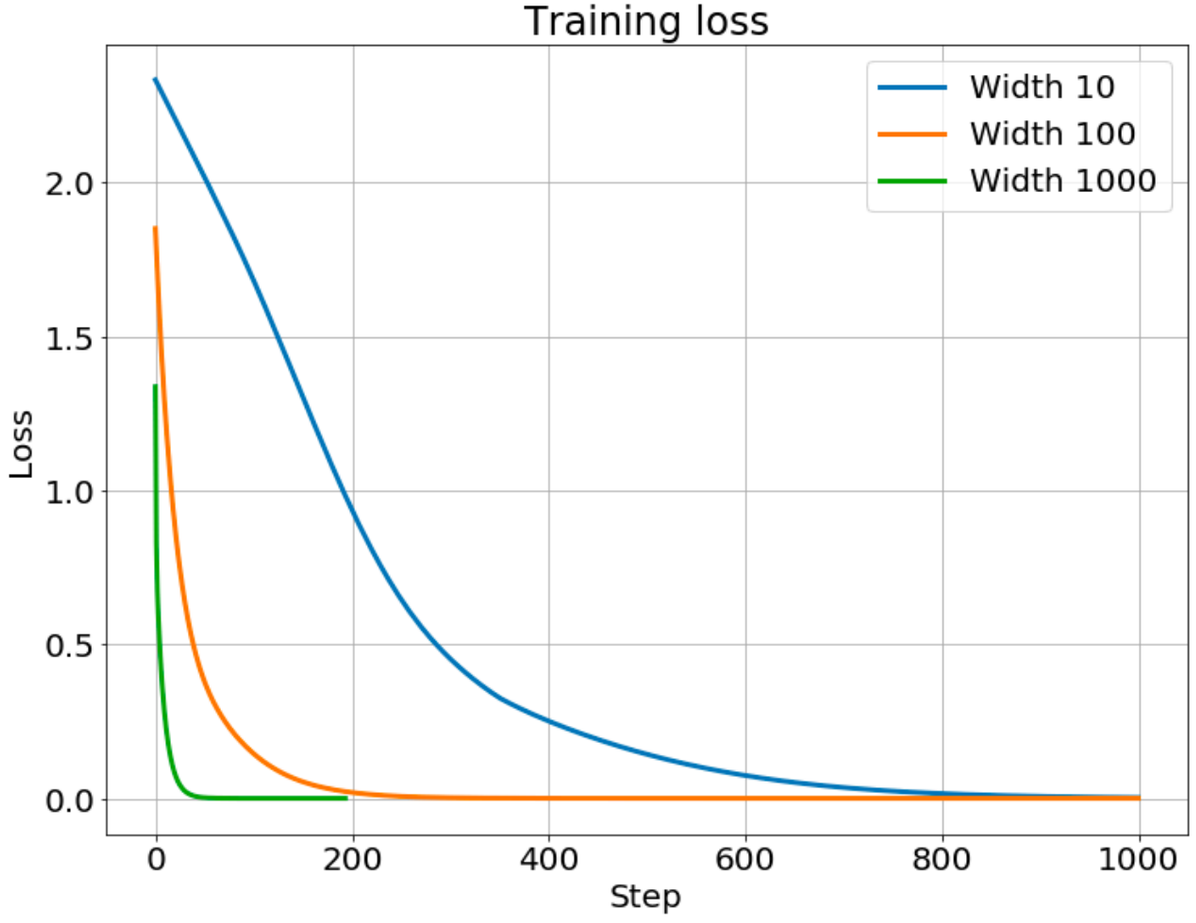


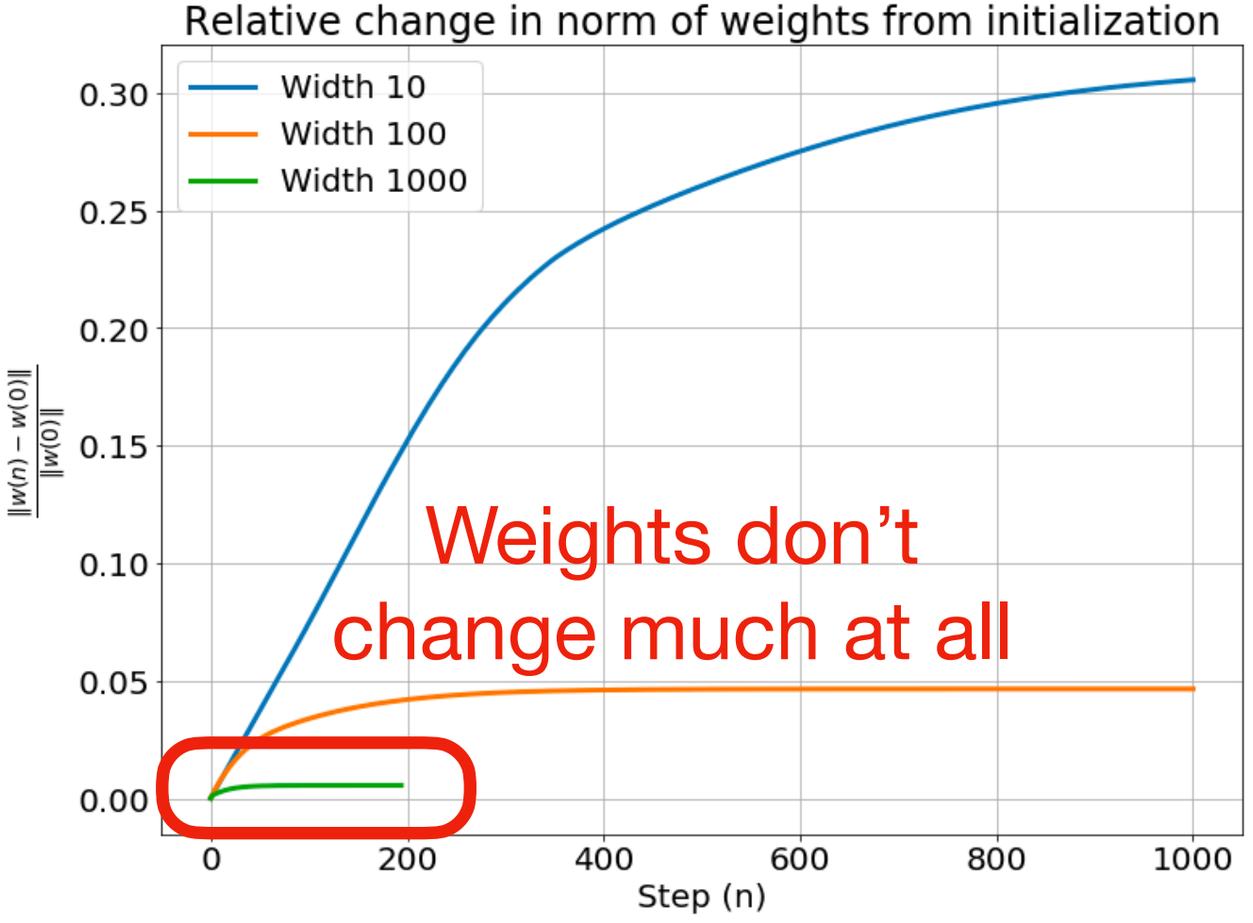
Weights don’t even budge for larger widths!

Weight Matrices during Training

- We can analyze this phenomenon by looking at the relative change of weights

$$\frac{\|w(n) - w_0\|_2}{\|w_o\|_2}$$





Weight Matrices during Training

- The weights don’t change much at all for larger hidden widths

- What’s good thing to do if some variable doesn’t change much?

Taylor Expansion

- Use Taylor expansion to approximate the network function **with respect to the weights** around its **initialization**

$$f(x, w) \approx f(x, w_0) + \nabla_w f(x, w_0)^T (w - w_0)$$

$$\rightarrow \quad y(w) \approx y(w_0) + \nabla_w y(w_0)^T (w - w_0)$$

  - Initial output $y(w_0)$ and the model Jacobian $\nabla_w y(w_0)$ are just constant!

- This approximation is a linear model, thus minimizing the least squares loss reduces to just doing **linear regression!**

$$f(x, w) = b + x^T w$$

Taylor Expansion

$$y(w) \approx y(w_0) + \nabla_w y(w_0)^T (w - w_0)$$

- Notice that the model function is still **non-linear**, because finding the gradient of the model is not a linear operation

- It is a linear model using a feature map $\phi(x)$, which is the gradient vector at initialization

$$\phi(x) = \nabla_w f(x, w_0) = \nabla_w y(w_0)$$

Gradient Flow

- Training dynamics of neural networks under gradient descent can be expressed as

$$w_{k+1} = w_k - \eta \nabla_w L(w_k)$$

$$\rightarrow \quad \frac{w_{k+1} - w_k}{\eta} = -\nabla_w L(w_k)$$

Finite-difference approximation to a derivative

- If we take the learning rate to be infinitesimally small, we can look at the evolution of the weight over time

$$\frac{dw(t)}{dt} = -\nabla_w L(w(t))$$ Gradient flow

Gradient Flow

- The trajectory of gradient descent in parameter space closely approximates the trajectory of the solution of this differential equation if the learning rate is small enough

$$\dot{w}(t) = \frac{dw(t)}{dt} = -\nabla_w L(w(t)) \qquad L(w) = \frac{1}{2}\|y(w) - \bar{y}\|_2^2$$

$$\rightarrow \qquad \dot{w} = -\nabla y(w)(y(w) - \bar{y})$$

- We can now derive the dynamics of the model outputs $y(w)$ induced by this gradient flow using the chain rule

$$\dot{y}(w) = \nabla y(w)^T \dot{w} = -\nabla y(w)^T \nabla y(w)(y(w) - \bar{y})$$

Neural Tangent Kernel (NTK)

Neural Tangent Kernel (NTK)

- In earlier Taylor expansion, linearized model has a feature map $\phi(x) = \nabla_w(x, w_0) = \nabla_w y(w_0)$

$$y(w) \approx y(w_0) + \nabla_w y(w_0)^T (w - w_0)$$

- The kernel matrix corresponding to this feature map is obtained by taking pairwise inner products $\nabla_w y(w_0)^T \nabla_w y(w_0)$, which is exactly NTK!

- If the model is close to its linear approximation, the Jacobian of the model output does not change as training progresses

$$\nabla y(w(t)) \approx \nabla y(w_0)$$

$$\rightarrow \nabla y(w(t))^T \nabla y(w(t)) \approx \nabla y(w_0)^T \nabla y(w_0)$$

$$n\left[\mathbf{H}(\mathbf{w}_0)\right] = \left[\nabla_w \mathbf{y}(\mathbf{w}_0)^T\right] \left[\nabla_w \mathbf{y}(\mathbf{w}_0)\right] p$$

n n p n

NTK

$$= \begin{bmatrix} — \ \phi(\bar{x}_1)^T \ — \\ \vdots \\ — \ \phi(\bar{x}_n)^T \ — \end{bmatrix} \begin{bmatrix} | & & | \\ \phi(\bar{x}_1) & \cdots & \phi(\bar{x}_n) \\ | & & | \end{bmatrix}$$

Neural Tangent Kernel (NTK)

- This is referred to as kernel regime, because the tangent kernel stays constant during training

- The training dynamics now reduces to a very simple **linear ordinary differential equation (ODE)**

$$\dot{y}(w) = \nabla y(w)^T \dot{w} = -\nabla y(w)^T \nabla y(w)(y(w) - \bar{y})$$
$$= -H(w_0)(y(w) - \bar{y})$$

$$\rightarrow \quad \dot{u} = -H(w_0)u; \;\; u = y(w) - \bar{y}$$

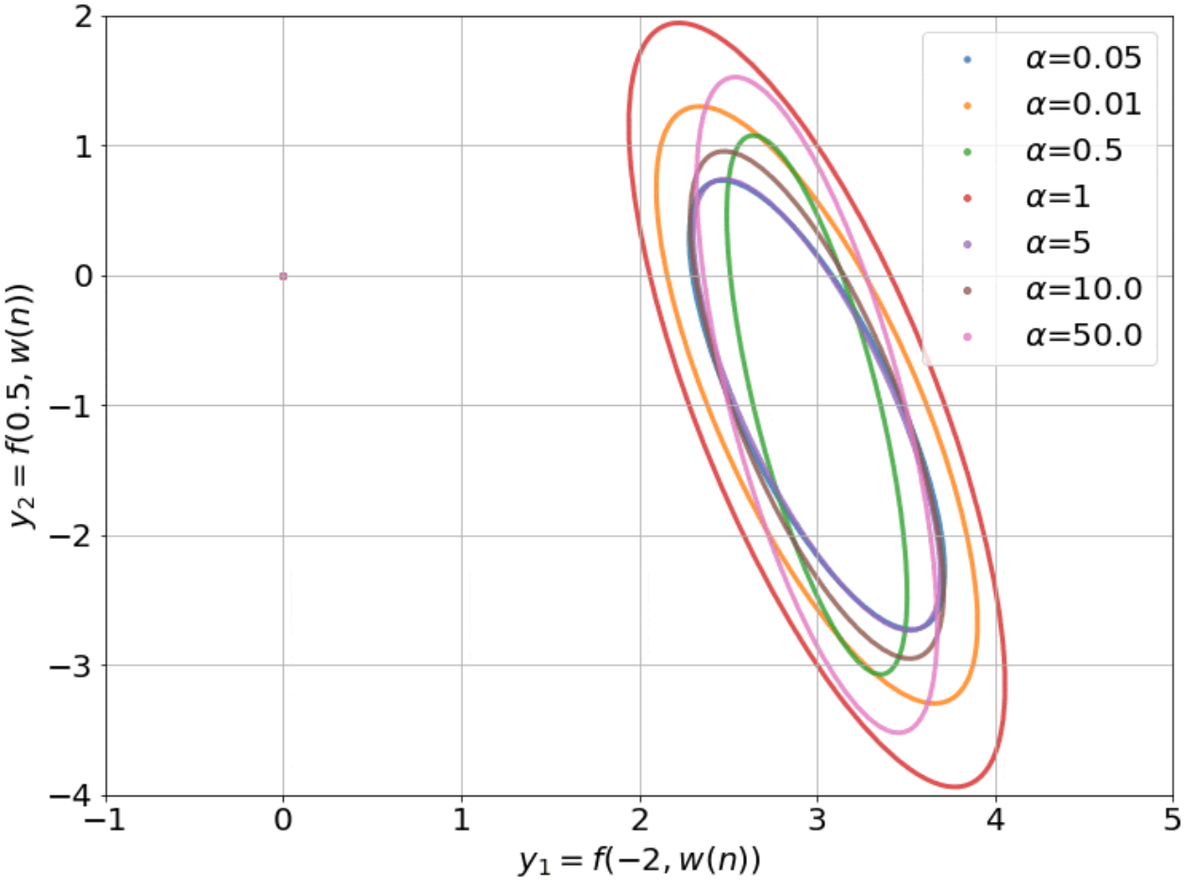
- The solution of this ODE is given by a matrix exponential

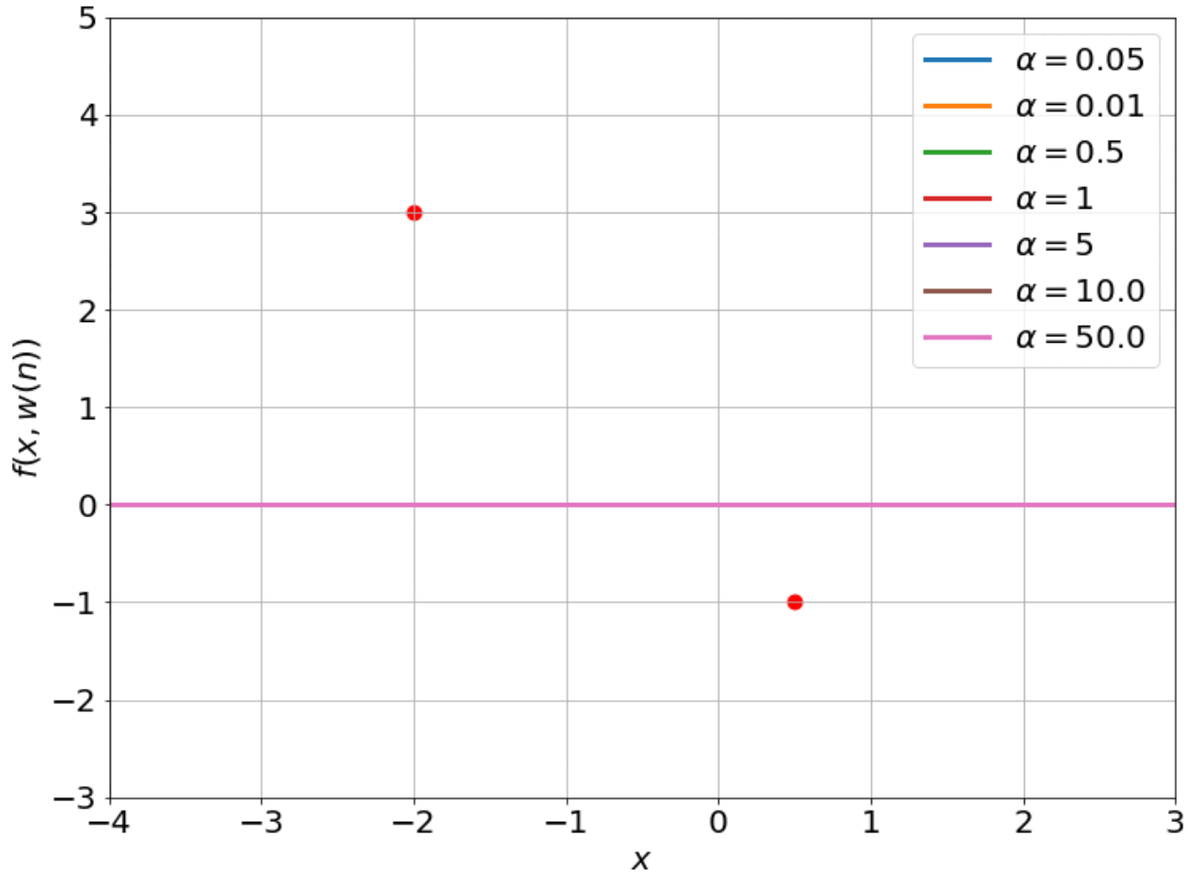
$$u(t) = u(0)e^{-H(w_0)t}$$

Neural Tangent Kernel (NTK)

- Let’s consider 2-dataset example. For 2 data point, the NTK is just a 2x2 positive definite matrix

- We can visualize it as ellipses in a 2-D plane, where axes are eigenvectors, and the length are inversely proportional to the square root of the eigenvalues





Neural Tangent Kernel (NTK)

- The final learned neural network at time $t \to \infty$ is equivalent to the **kernel regression** solution with respect to the NTK

$$f_N N(x) \approx f_{NTK}(x) = \Theta(x, X)^T \Theta(X, X)^{-1} Y$$

  - , where $\Theta(x, x') = \nabla f(x, w)^T \nabla f(x', w)$

Before training

Gaussian Process: $\Theta(x, x') = \mathbb{E}_{w \sim W}\left[\langle f(x, w), f(x', w) \rangle\right]$

After training

Neural Tangent: $\Theta(x, x') = \mathbb{E}_{w \sim W}\left[\langle \frac{\partial f(x, w)}{\partial w}, \frac{\partial f(x', w)}{\partial w} \rangle\right]$

Experiment (CIFAR-10)

- This results in a significant new benchmark based on a pure kernel-based method on CIFAR-10, being 10% higher than methods reported in 2019

Depth	CNN-V	CNTK-V	CNN-GAP	CNTK-GAP
3	61.97%	64.67%	57.96%	70.47%
4	62.12%	65.52%	80.58%	75.93%
6	64.03%	66.03%	80.97%	76.73%
11	70.97%	65.90%	75.45%	77.43%
21	80.56%	64.09%	81.23%	77.08%

Assumption in NTK

1. Infinite-width
2. Over-parametrization
3. Proper initialization
4. Small learning rate

Conclusion

- Training neural network is same as solving kernel regression over feature map $\phi(x) = \nabla_w f(x, w_0)$ if assumptions hold

- NTK is random at initialization and varies during training, in the infinite-width limit it **converges** to an explicit limiting kernel and **stay constant** during training

- Gradient descent converges to 0 training loss for any non-linear model as long as it is close to its linearization

Related Works

- Neural Tangent Kernel: Convergence and Generalization in Neural Networks, A. Jacot et al., NeurIPS'18

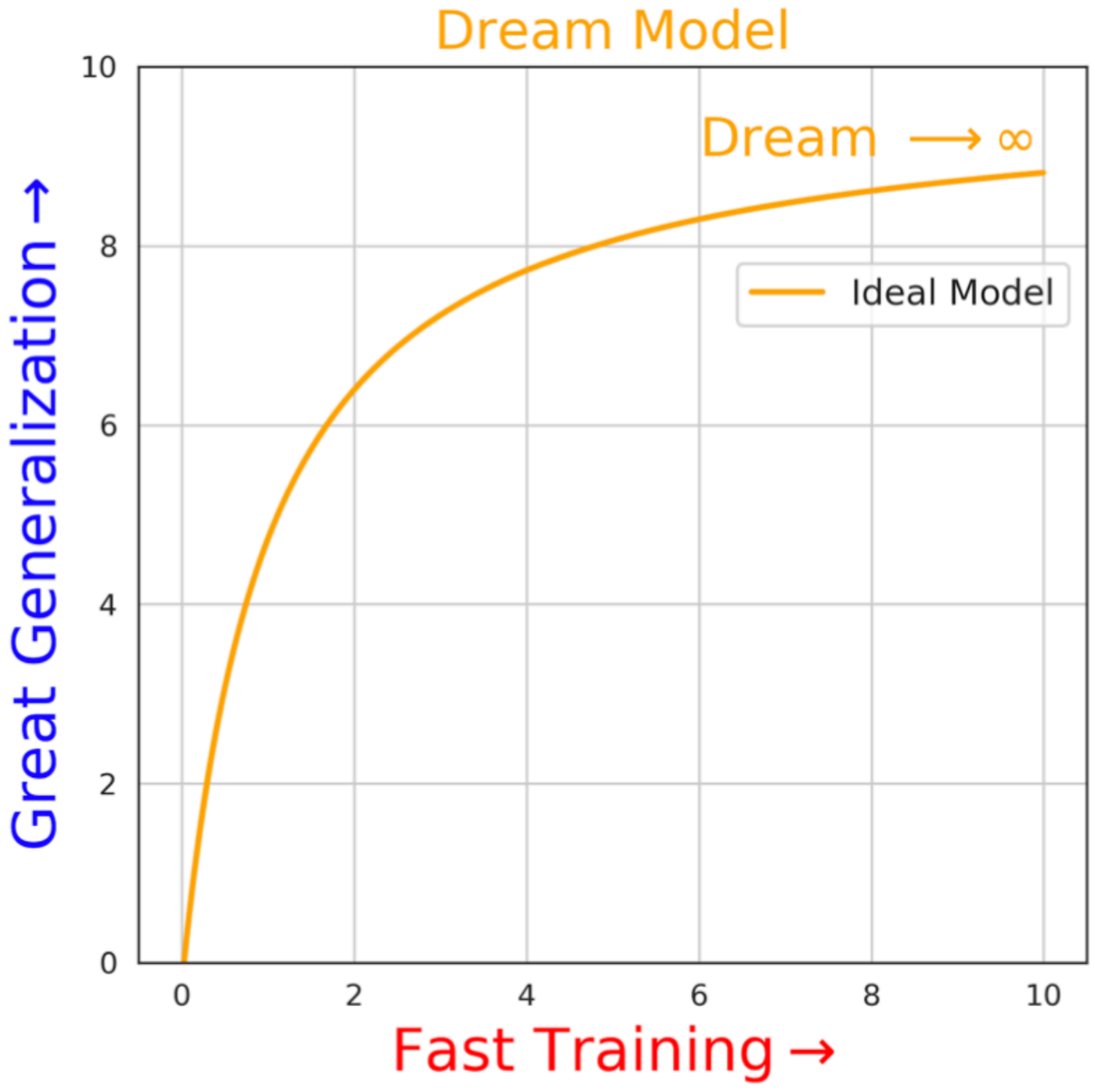
- Disentangling Trainability and Generalization in Deep Neural Networks, L. Xiao et al., ICML'20

Main Idea

- Characterize the conditions under which a given neural network architecture will be **trainable**, and if so, how well it might **generalize** to unseen data

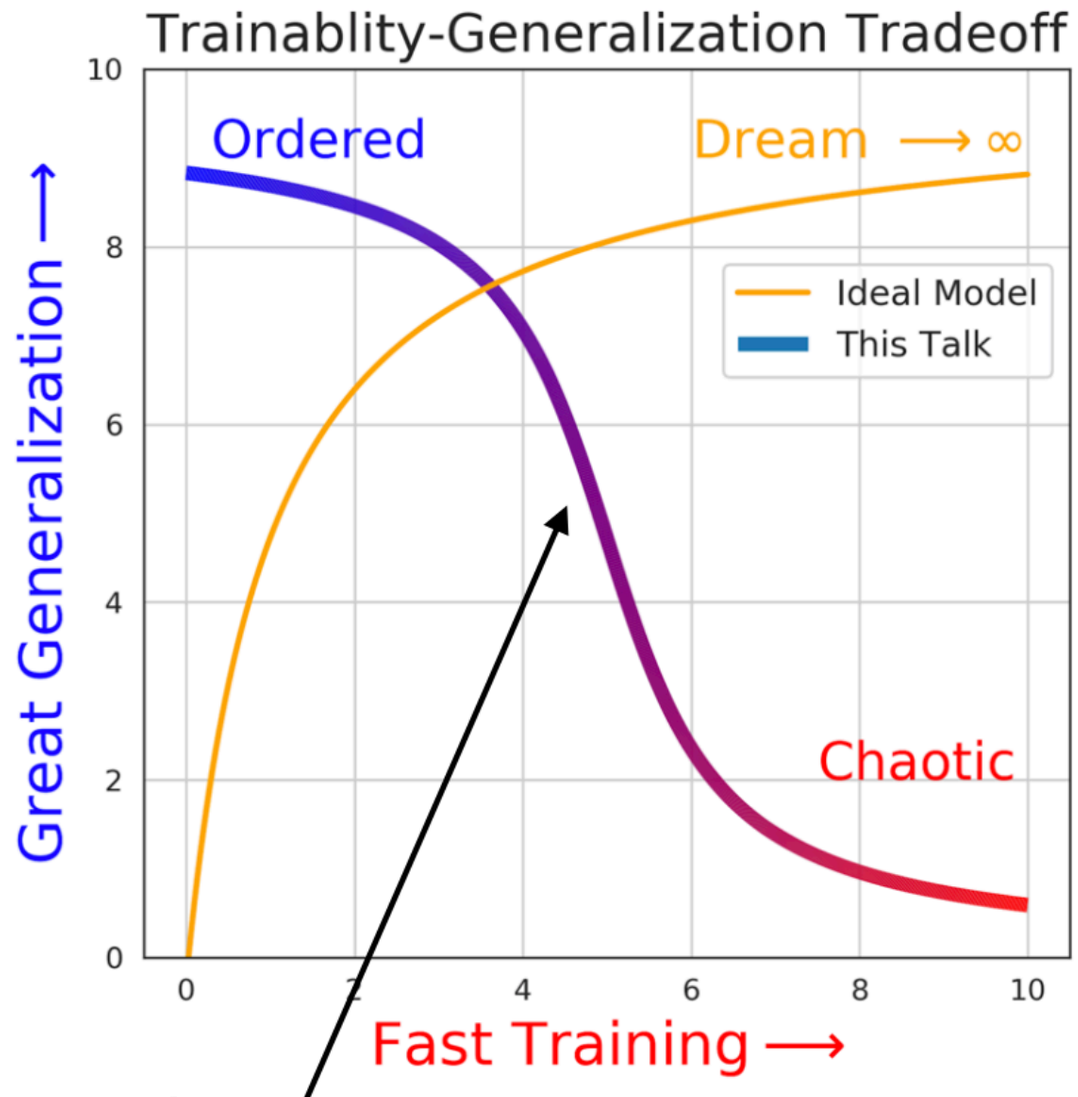
Two Fundamental Theoretical Questions

- Trainability (optimization)
  - Efficient algorithm to reach global minimum
- Generalizability
  - Performance on unseen data



A Trade-off between Trainability and Generalization

- Chaotic phase
  - Easy to train, but not able to generalize
- Ordered phase
  - Hard to train, but able to generalize



Deep neural networks

Gradient Descent Dynamics

- We have already shown that gradient descent dynamics is

$$\frac{df_\theta(X_{\text{train}})}{dt} = -\eta\Theta_{\text{train,train}}(f_\theta(X_{\text{train}} - Y_{\text{train}}))$$

$$\Theta_{\text{train,train}} = \nabla f_\theta(X_{\text{train}})^T \nabla f_\theta(X_{\text{train}})$$

- Training dynamics

$$\mu_t(X_{\text{train}}) = (\mathbf{I} - e^{-\eta\Theta\text{train,train}^t})Y_{\text{train}}$$

- Learning dynamics

$$\mu_t(X_{\text{test}}) = \Theta_{\text{test,train}}\Theta^{-1}_{\text{train,train}}(\mathbf{I} - e^{-\eta\Theta\text{train,train}^t})Y_{\text{train}}$$

Metric for Trainability: Condition Number

- Eigendecomposition of NTK

$$\mu_t(X_{\text{train}}) = (\mathbf{I} - e^{-\eta\Theta_{\text{train,train}}{}^t})Y_{\text{train}}$$

$$\Theta_{\text{train,train}} = U^T D U \qquad D = \text{diag}(\lambda_1, \lambda_2, \cdots, \lambda_m)$$

  - The smallest eigenvector converges at rate $1/\kappa$

- Trainability metric $\kappa = \dfrac{\lambda_0}{\lambda_m}$ (condition number)

If condition number of NTK diverges,
then it will become untrainable

Metric for Generalization: Mean Prediction

- Generalization metric $P(\Theta)Y_{\text{train}}$ (mean predictor)

$$P(\Theta) \equiv \Theta_{\text{test,train}}\Theta^{-1}_{\text{train,train}}$$

$$\mu_t(X_{\text{test}}) = \Theta_{\text{test,train}}\Theta^{-1}_{\text{train,train}}(\mathbf{I} - e^{-\eta\Theta_{\text{train,train}}{}^{t}})Y_{\text{train}}$$

Turn into $\mathbf{I}$ as $t \to \infty$

Generalization performance can be quantified
by consider the rate at which $P(\Theta^{(l)})Y_{\text{train}} \to 0$
(independent from data)

Evolution of the Metrics with Depth

Neural networks $\cdots \to f_\theta^{(l)}(X) \to f_\theta^{l+1}(X) \to \cdots$

NTK $\cdots \to \Theta^{(l)} \to \Theta^{l+1} \to \cdots \to \Theta^*$

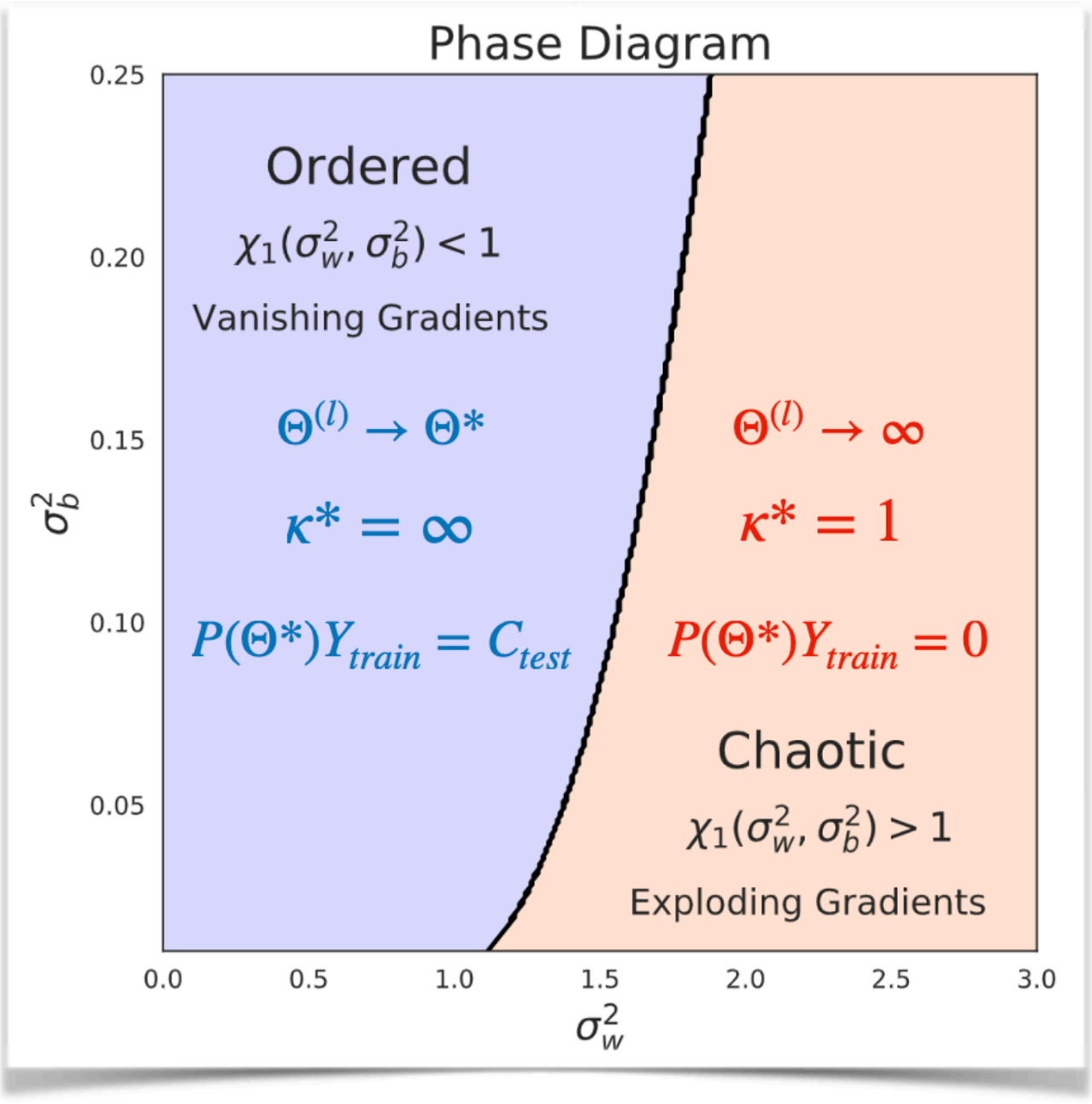
Condition number $\cdots \to \kappa^{(l)} \to \kappa^{l+1} \to \cdots \to \kappa^*$

Mean prediction

$$\cdots \to P(\Theta^{(l)})Y_{\text{train}} \to P(\Theta^{l+1})Y_{\text{train}} \to \cdots \to P(\Theta^*)Y_{\text{train}}$$

Convergence of NTK

- Convergence of $\Theta^{(l)}$ is determined by a bivariate function χ_1 defined on (σ_w^2, σ_b^2) plane



Chaotic Phase $\chi_1 > 1$

- Entries dynamics of NTK
  - $\Theta^{(l)}(x, x) \propto \chi_1^l \to \infty$ Diagonal: diverge
  - $\Theta^{(l)}(x, x') \propto p_{ab}^* < \infty$ Off-diagonal: converge
  - $\Theta^{(l)} \approx \chi_1^l \mathbf{I}$ Diagonal metric

- Corresponding trainability and generalization metrics
  - $\kappa^{(l)} = 1 + O(\chi_1^{-l}) \to 1$ The smaller, the better
  - $P(\Theta^{(l)}) Y_{\text{train}} = O\left(l\left(\frac{\chi_c^*}{\chi_1}\right)^l\right) \to 0$ The larger, the better

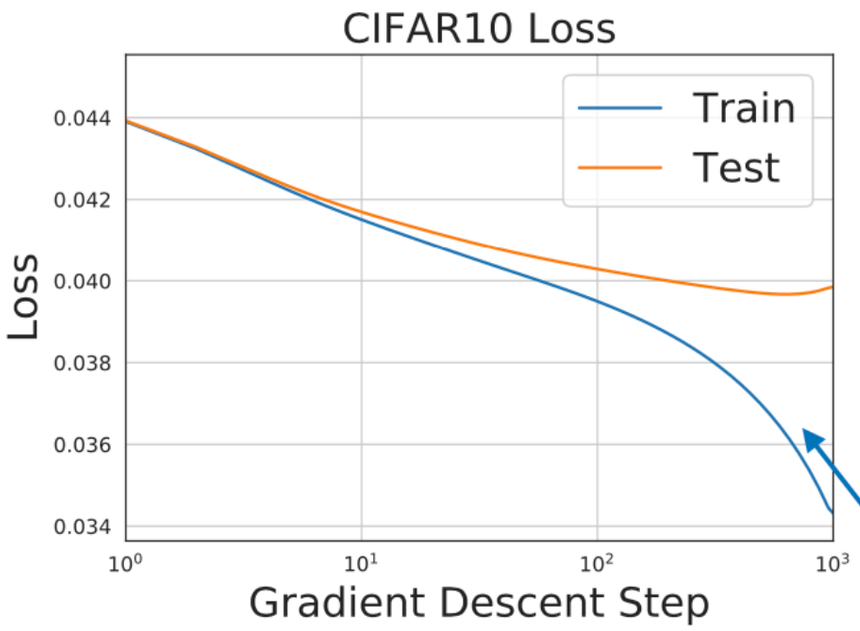
- **Easy to train, but hard to generalize**

Ordered Phase $\chi_1 < 1$

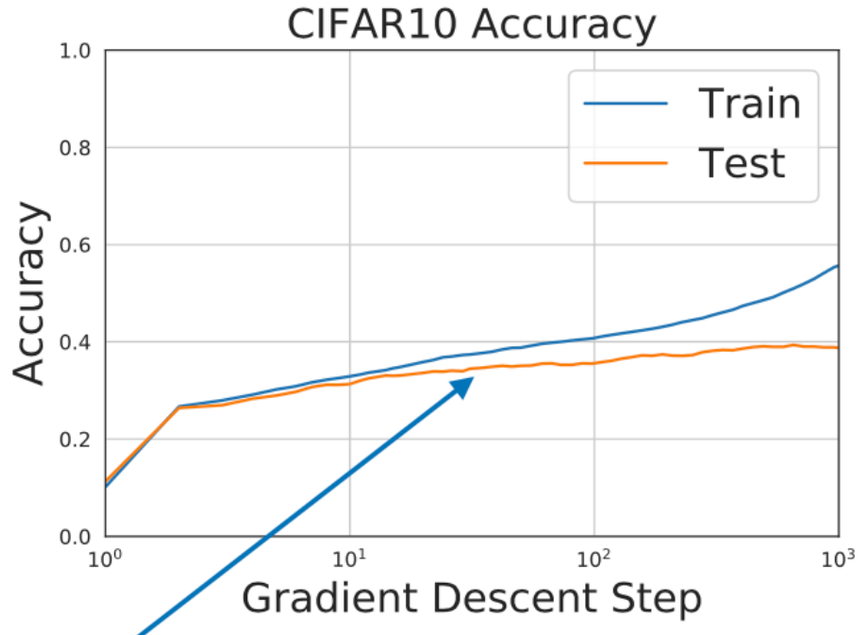
- Entries dynamics of NTK
  - $\Theta^{(l)}(x, x) \propto p^* + O(\chi_1^l) \to p^*$ Diagonal: converge
  - $\Theta^{(l)}(x, x') \propto p^* + O(l\chi_1^l) \to p^*$ Off-diagonal: converge
  - $\Theta^{(l)} = p^* \mathbf{1}^T \mathbf{1} + O(l\chi_1^l)$ Matrix of p^*
- Corresponding trainability and generalization metrics
  - $\kappa^{(l)} \gtrsim \chi_1^{-l}/l$ The smaller, the better
  - $P(\Theta^{(l)}) Y_{\text{train}} \to C_{\text{test}}$ The larger, the better

- **Difficult to train, but easy to generalize**

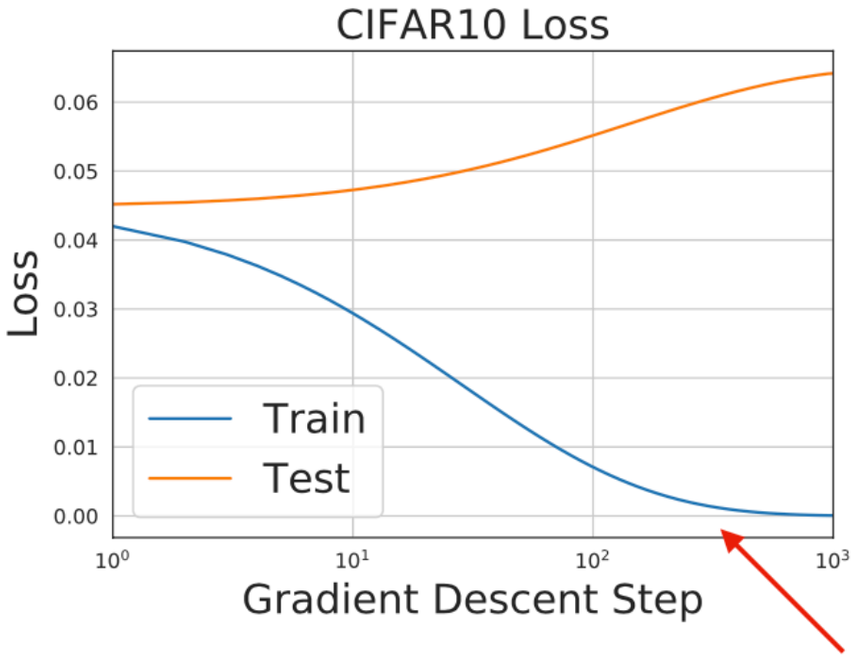
Experiment (CIFAR-10)



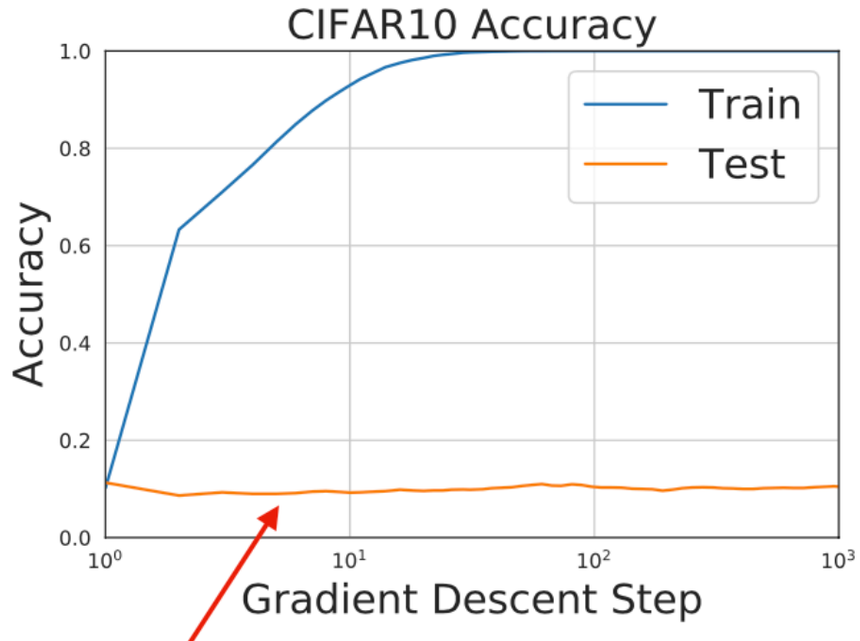
• $\sigma_w^2 = 0.5$



Difficult to Train, Generalizable

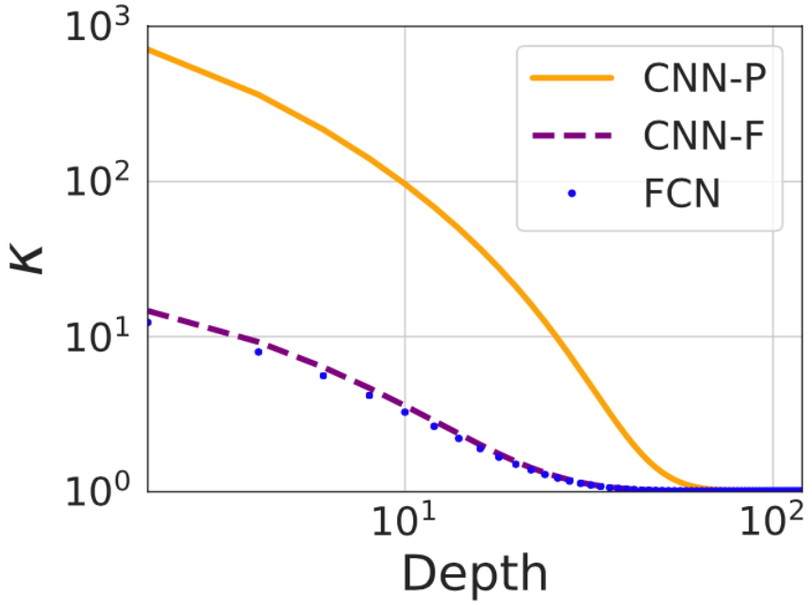


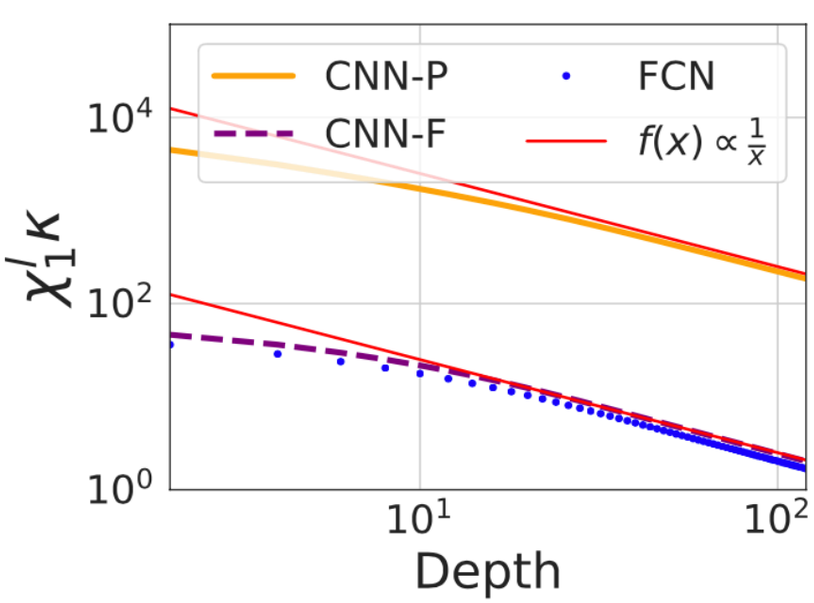
• $\sigma_w^2 = 25$

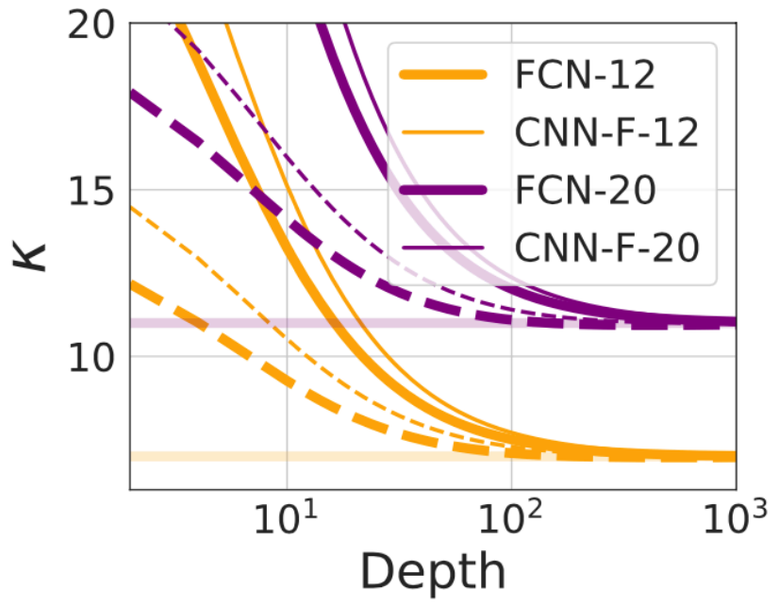


Easy to Train, but Not Generalize

Experiment



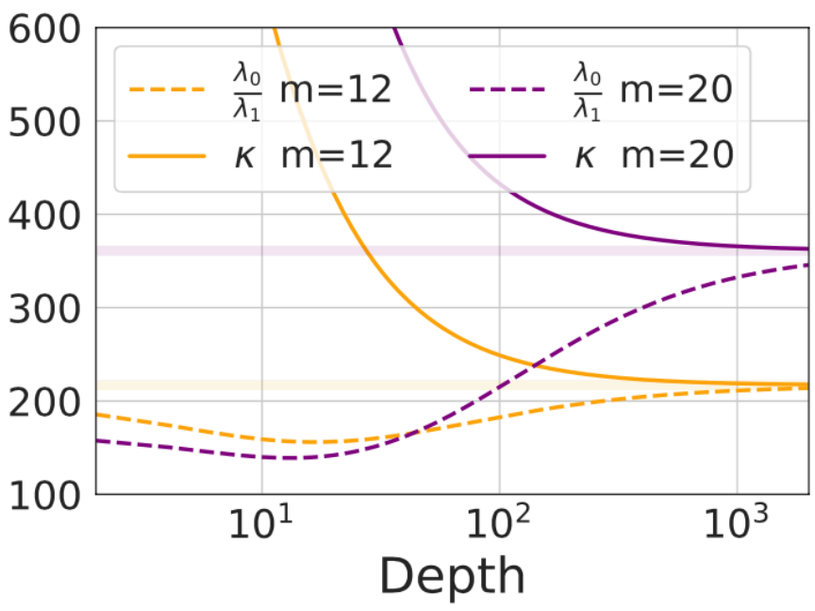


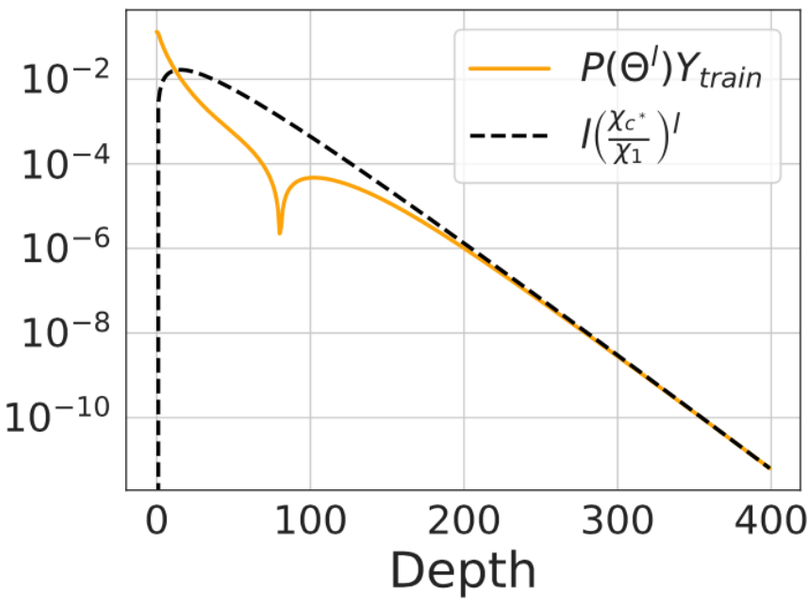


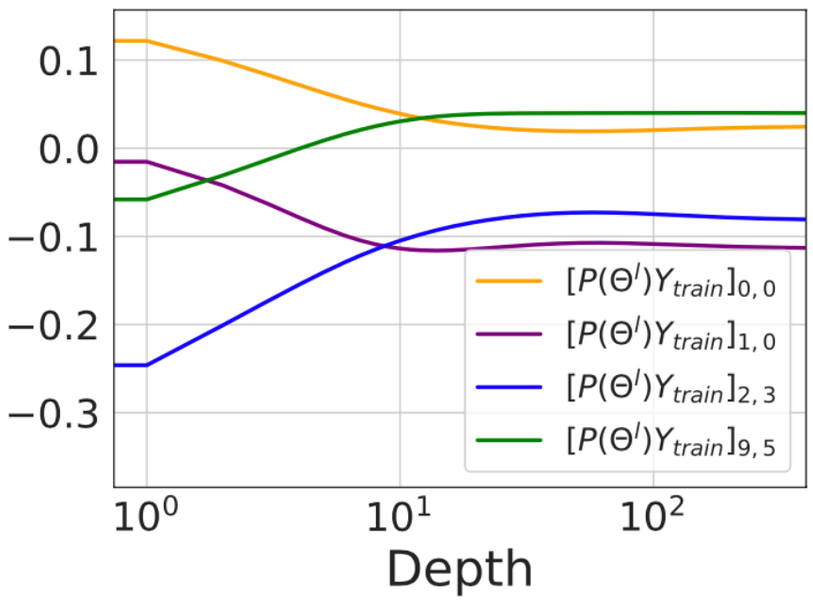
(a) Chaotic

(b) Ordered

(c) Critical: FCN and CNN-F







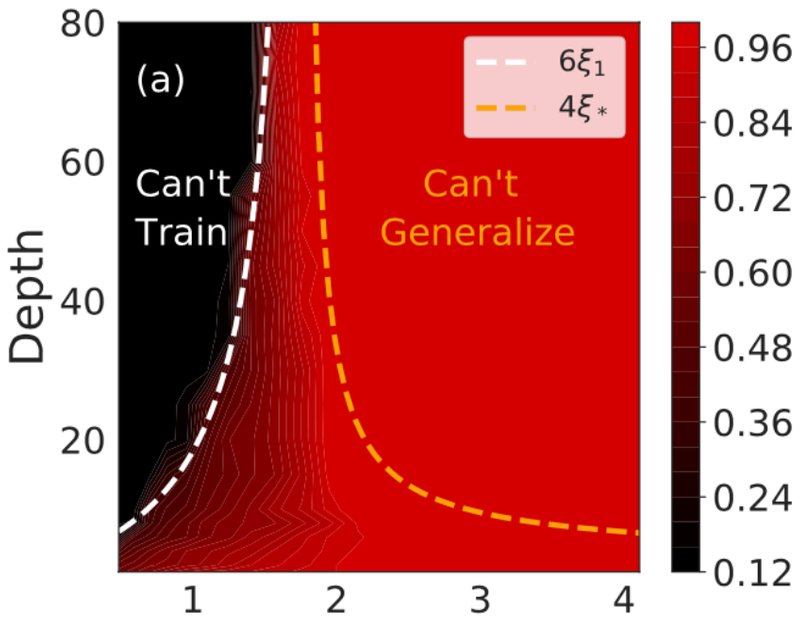
(d) Critical: CNN-P

(e) Chaotic: $P(\Theta^{(l)})Y_{\text{train}}$

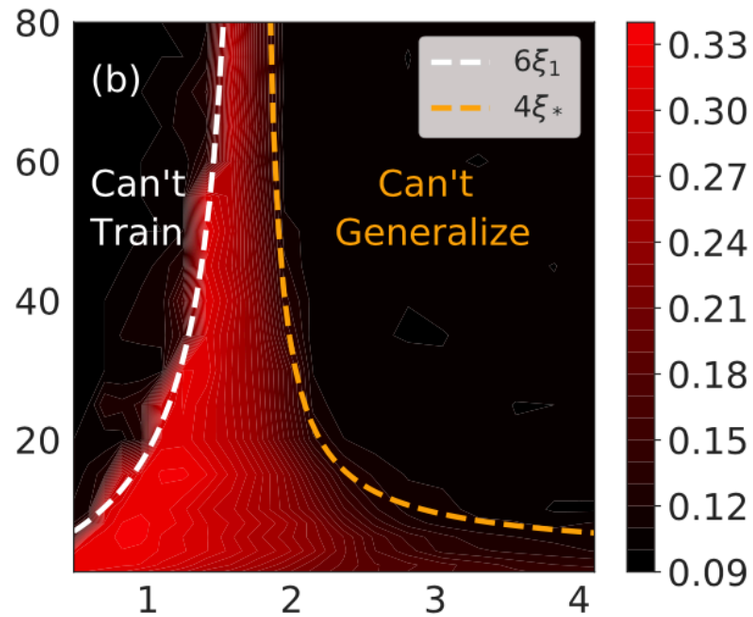
(f) Ordered: $P(\Theta^{(l)})Y_{\text{train}}$

Experiment (CIFAR-10)

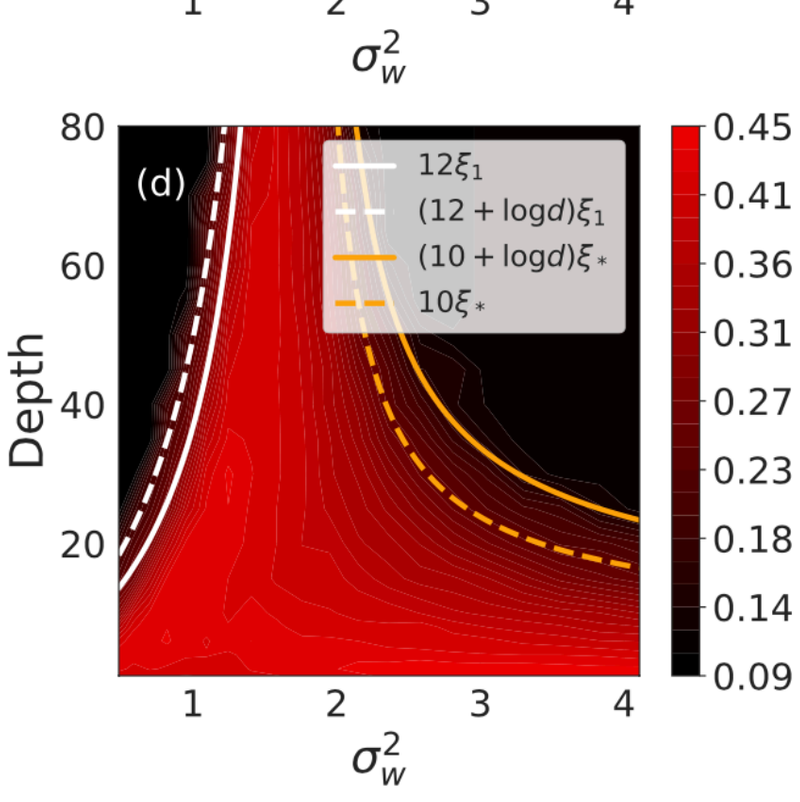
Train acc
CNN-F/SGD



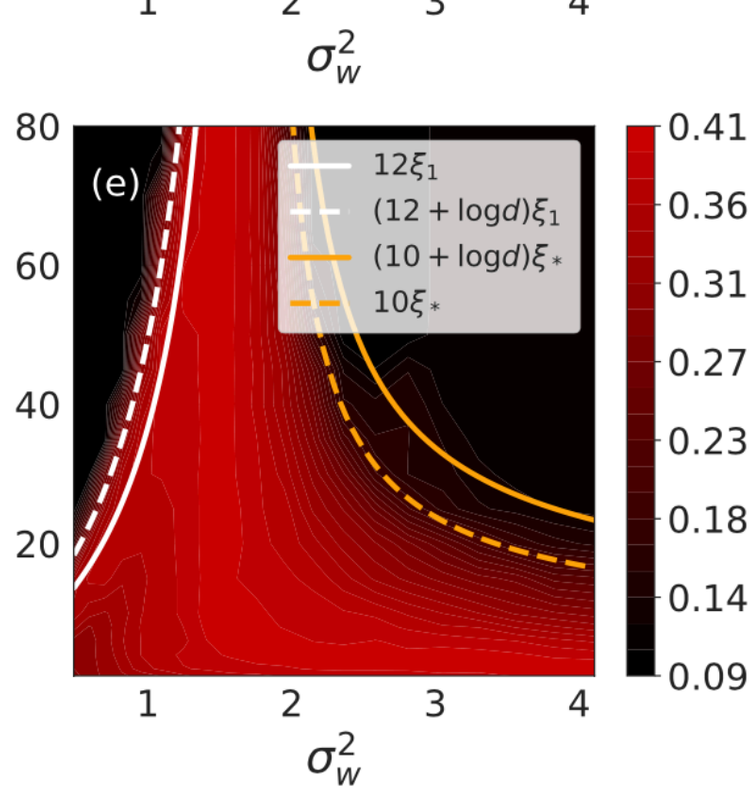
Test acc
CNN-F/SGD



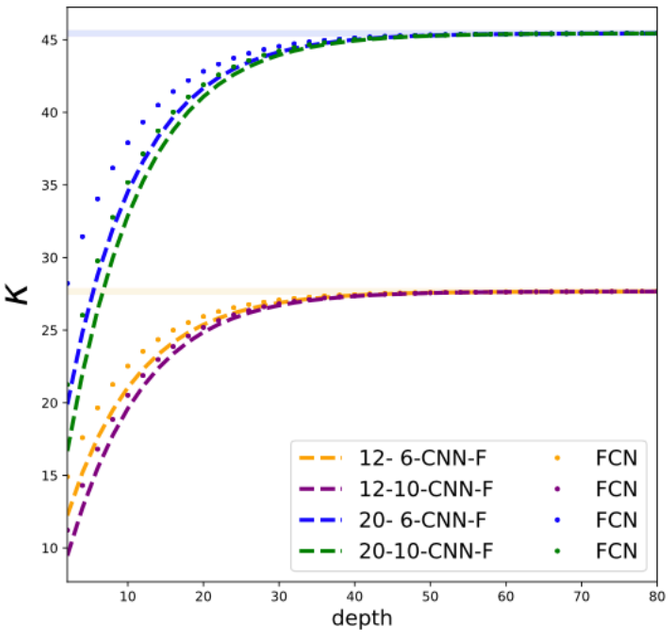
Test acc
CNN-P/NTK



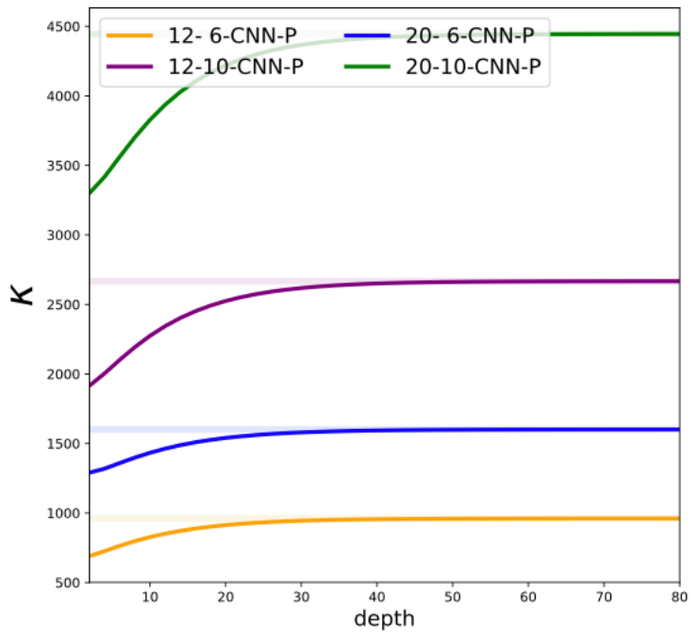
Test acc
CNN-F/NTK



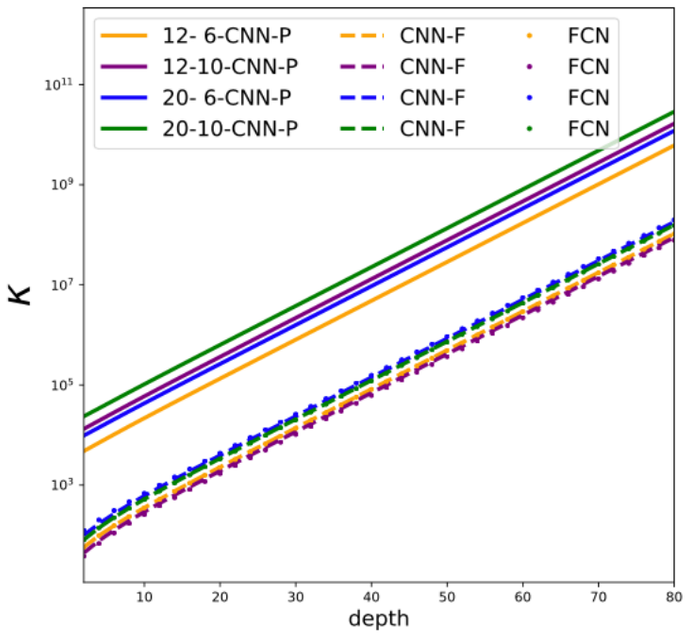
Experiment (NTK v.s. NNGP)



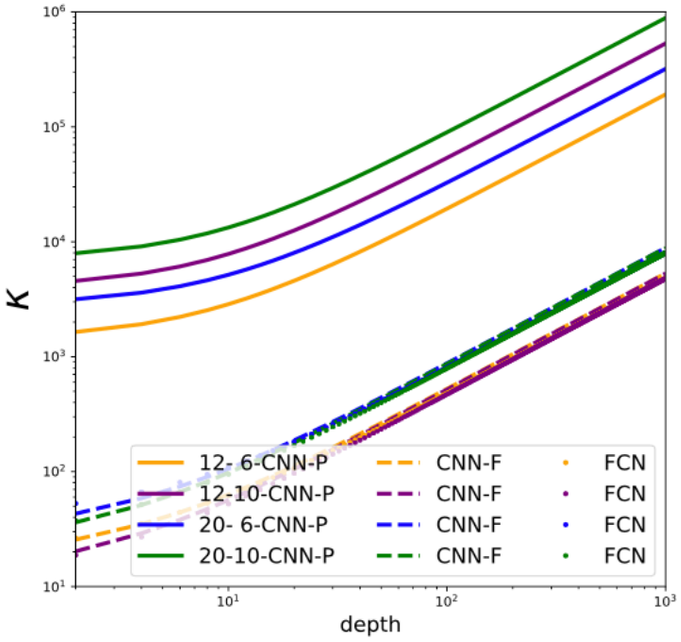
(a) NNGP Chaotic



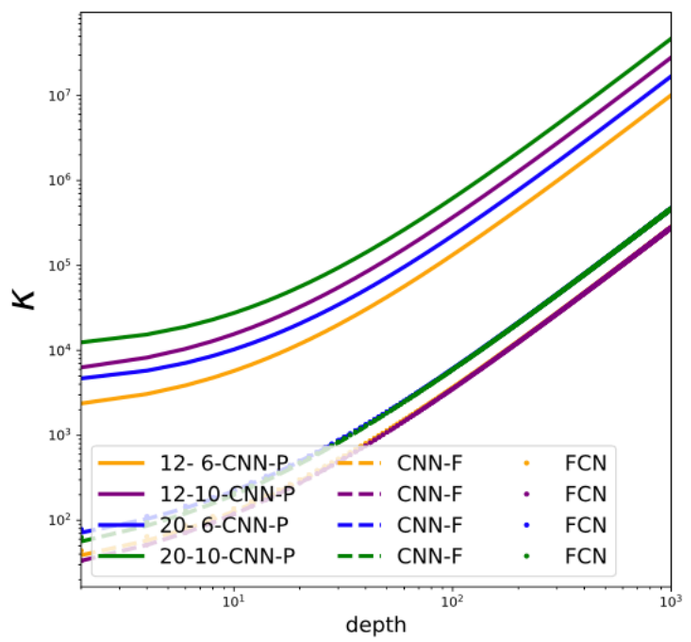
(b) NNGP Chaotic



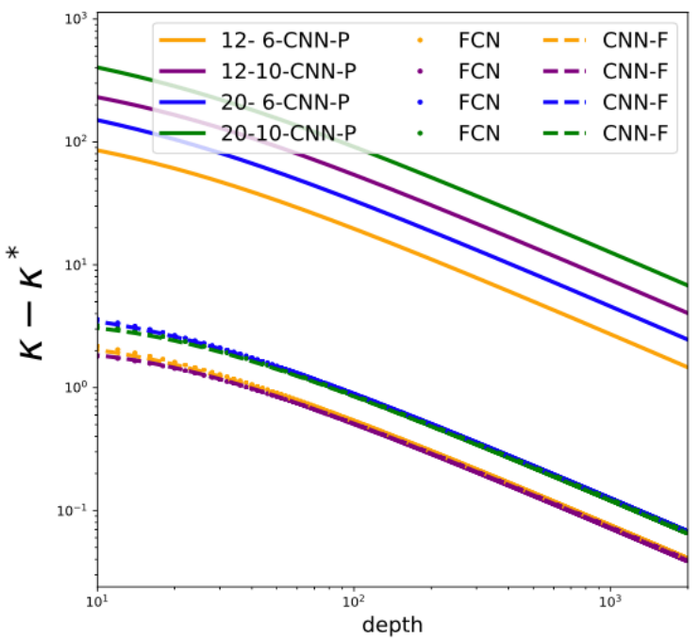
(c) NNGP Ordered



(d) NNGP Critical



(e) NNGP RELU



(f) NTK RELU

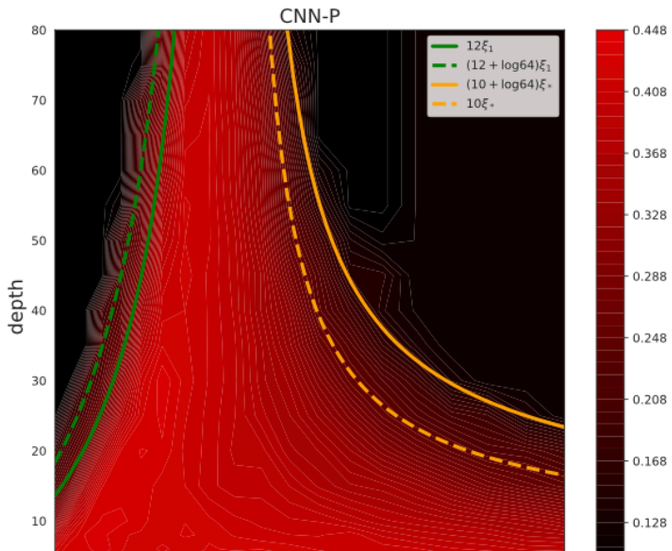
Experiment (NTK v.s. NNGP)

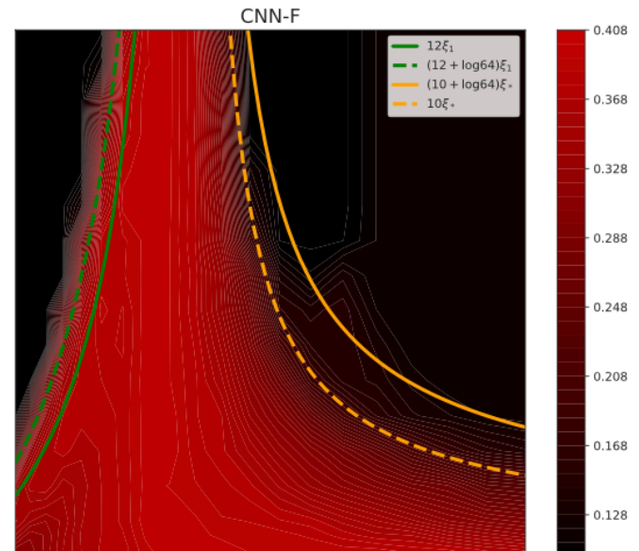
CNN-P

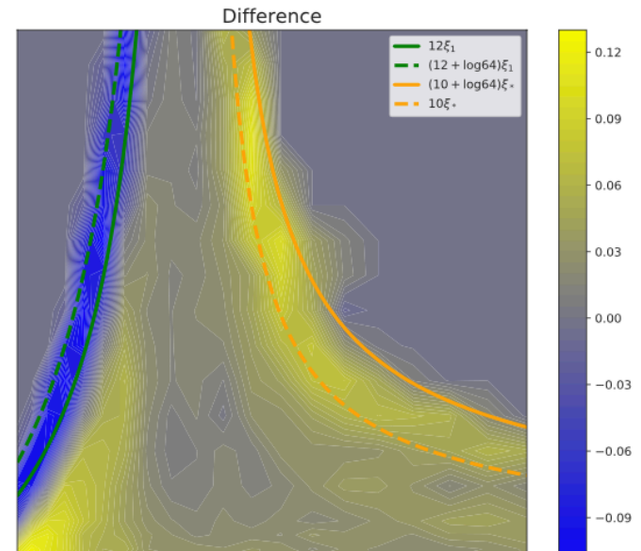
CNN-F

Difference

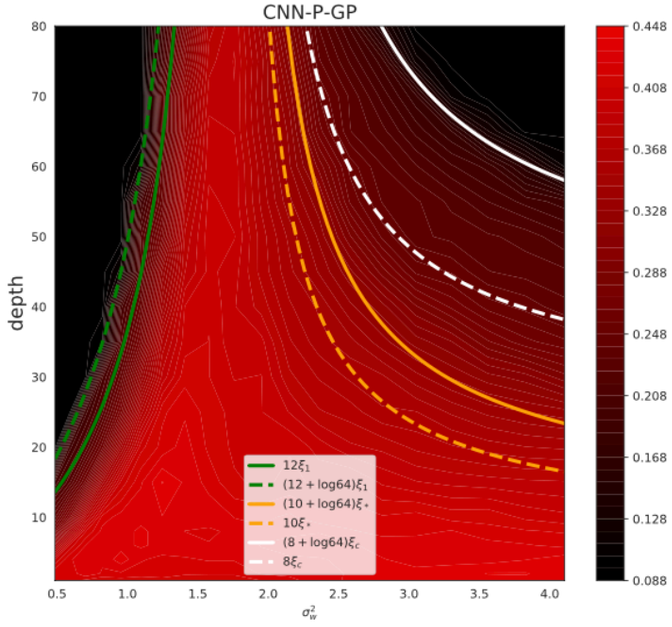
NTK

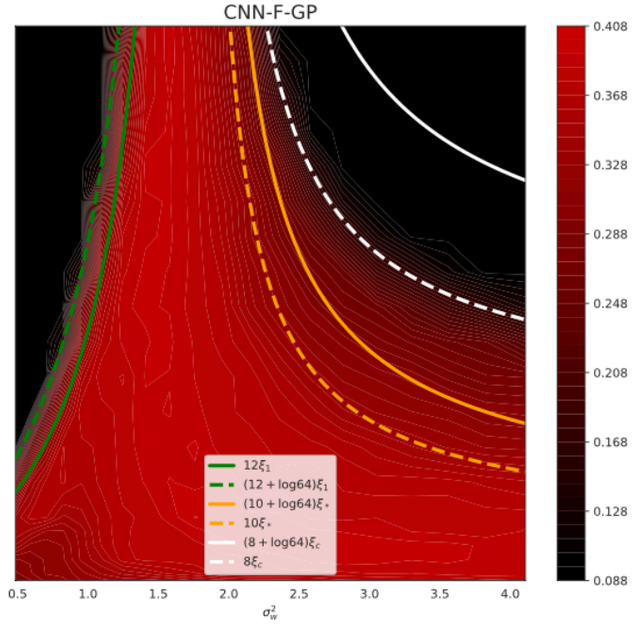


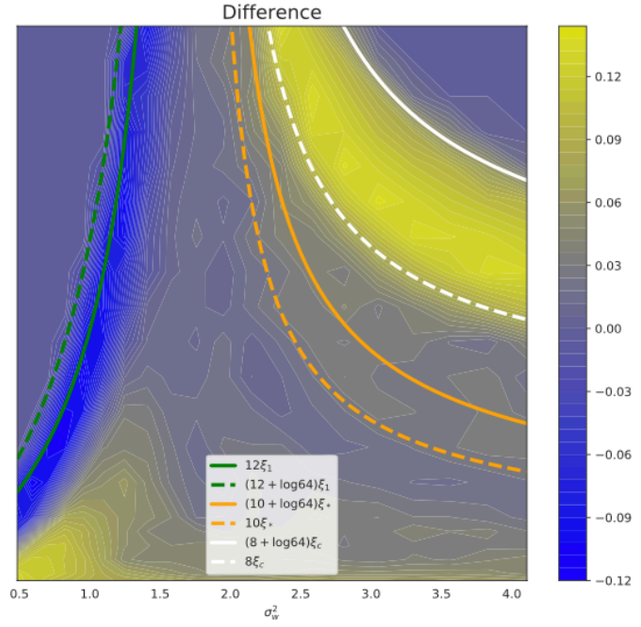




NNGP







Conclusion

- There is a trade-off between trainability and generalizability for deep and wide networks
  - In chaotic phase, it is easy to train, but hard to generalize
  - In ordered phase, it is hard to train, but easy to generalize
- The relation between NTK and NNGP is interesting to figure out

Outline

1. Estimating Probability from Data
2. Gaussian Process
3. Information Propagation
4. Neural Networks as Gaussian Process
5. Neural Tangent Kernel

Summary

- Neural networks turns into kernelized linear regression in limit if infinite width
  - Before training: **Neural Network Gaussian Process (NNGP)**
  - After training: **Neural Tangent Kernel (NTK)**
- The relation between NTK and NNGP is fascinating to figure out
- Research direction
  - Create a dataset which has poor trainability/generalizability kernel
  - Not sure whether it still holds if the width changes

Reference

- Exponential Expressivity in Deep Neural Networks through Transient Chaos, B. Poole et al., NeurIPS'16

- Deep Information Propagaiton, S. Schoenholz and J. Gilmer et al., ICLR'17

- Deep Neural Networks as Gaussian Process, J. Lee and Y. Bahri et al., ICLR'18

- Neural Tangent Kernel: Convergence and Generalization in Neural Networks, A. Jacot et al., NeurIPS'18

- Disentangling Trainability and Generalization in Deep Neural Networks, L. Xiao et al., ICML'20

Reference

- Cornell CS4780 "Machine Learning for Intelligent Systems"
- A Gentle Introduction to Maximum Likelihood Estimation and Maximum A Posteriori Estimation
- MLE, MAP and Bayesian Inference
- SVM and Kernel SVM
- Kernel Functions
- Gaussian processes
- An intuitive guide to Gaussian processes
- Understanding the Neural Tangent Kernel
- Ultra-Wide Deep Nets and the Neural Tangent Kernel (NTK)