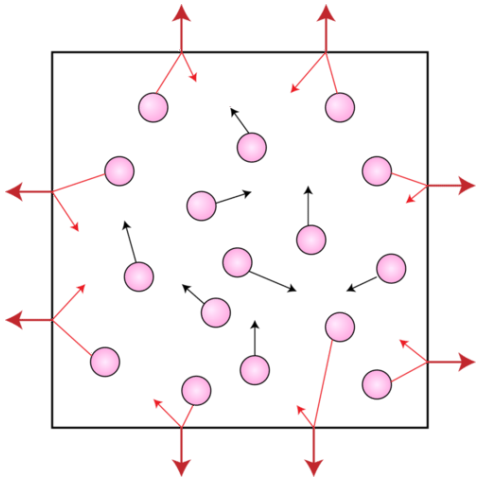
CSC 2541: Neural Net Training Dynamics

Lecture 6 - Infinite Limits and Overparameterization

Roger Grosse

University of Toronto, Winter 2021

Today



$PV = NRT$

- Many systems become much simpler at a macroscopic scale, since the behaviors of the component parts can be summarized statistically
- Neural nets are an example of this: their behavior can become much simpler when they're extremely wide

Today

The plan for today:

- Recap of kernels and Gaussian processes (Tutorial 5)
- Two ways of taking infinite width limits
  - Wide Bayesian neural nets $\rightarrow$ GPs
  - Gradient descent on wide (non-Bayesian) networks, and the Neural Tangent Kernel
- Limiting behavior with depth
- Relating this all back to why neural nets can generalize

Recap of Kernels and Gaussian Processes

Full Bayesian Inference

- **Recall:** full Bayesian inference makes predictions by averaging over all likely explanations under the posterior distribution.
- Compute posterior using Bayes' Rule:

$$p(\mathbf{w} \mid \mathcal{D}) \propto p(\mathbf{w}) p(\mathcal{D} \mid \mathbf{w})$$

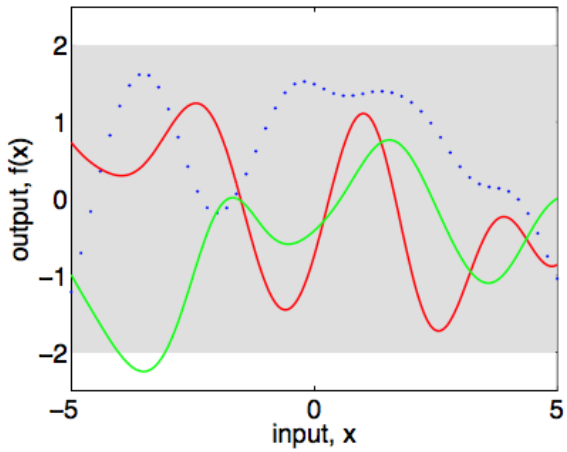
- Make predictions using the posterior predictive distribution:

$$p(t \mid \mathbf{x}, \mathcal{D}) = \int p(\mathbf{w} \mid \mathcal{D})\, p(t \mid \mathbf{x}, \mathbf{w})\, \mathrm{d}\mathbf{w}$$

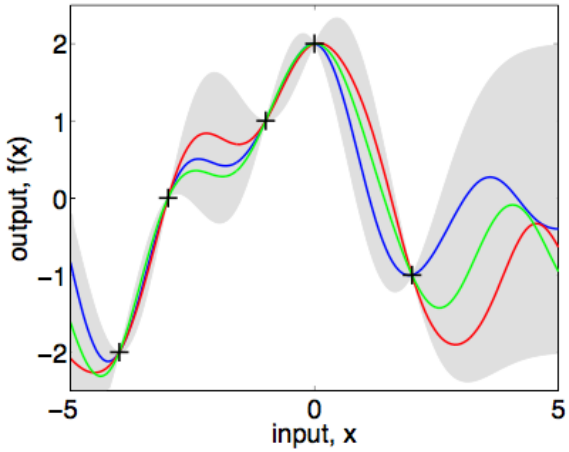
- Doing this lets us quantify our uncertainty.

Gaussian Processes

- Gaussian processes are a kind of stochastic process, or distribution over functions.
- The trick is to define multivariate Gaussian distributions over the values at any finite set of points and show that they're consistent with each other. They can then be extended to a stochastic process using the Kolmogorov Extension Theorem.
- All of the actual computations are done using multivariate Gaussians at finite sets of points.
- But the stochastic process interpretation gives a lot of useful intuitions.



(a), prior



(b), posterior

- How do you think these plots were generated?

Gaussian Processes

- Defining a GP requires specifying:
  - a mean function $\mathbb{E}[f(\mathbf{x}_i)] = \mu(\mathbf{x}_i)$
  - a covariance function called a kernel function: $\text{Cov}(f(\mathbf{x}_i), f(\mathbf{x}_j)) = k(\mathbf{x}_i, \mathbf{x}_j)$
- Let $\mathbf{K}_\mathbf{X}$ denote the kernel matrix for points $\mathbf{X}$. This is a matrix whose (i, j) entry is $k(\mathbf{x}^{(i)}, \mathbf{x}^{(j)})$, and is called the Gram matrix.
- We require that k be symmetric and $\mathbf{K}_\mathbf{X}$ be positive semidefinite for *any* $\mathbf{X}$. Other than that, μ and k can be arbitrary.

Kernel Trick

- This is an instance of a more general trick called the Kernel Trick.
- Many algorithms (e.g. linear regression, logistic regression, SVMs) can be written in terms of dot products between feature vectors, $\boldsymbol{\phi}(\mathbf{x})^\top \boldsymbol{\phi}(\mathbf{x}')$.
- A kernel implements an inner product between feature vectors, typically implicitly, and often much more efficiently than the explicit dot product.
- For instance, the following feature vector is quadratic in size:

$$\phi(\mathbf{x}) = (1, \sqrt{2}x_1, ..., \sqrt{2}x_d, \sqrt{2}x_1x_2, \sqrt{2}x_1x_3, ...\sqrt{2}x_{d-1}x_d, x_1^2, ..., x_d^2)$$

- But the quadratic kernel can compute the inner product in linear time:

$$k(\mathbf{x}, \mathbf{x}') = \phi(\mathbf{x})^\top \phi(\mathbf{x}') = 1 + \sum_{i=1}^{d} 2x_i x_i' + \sum_{i,j=1}^{d} x_i x_j x_i' x_j' = (1 + \mathbf{x}^\top \mathbf{x}')^2$$

Kernel Trick

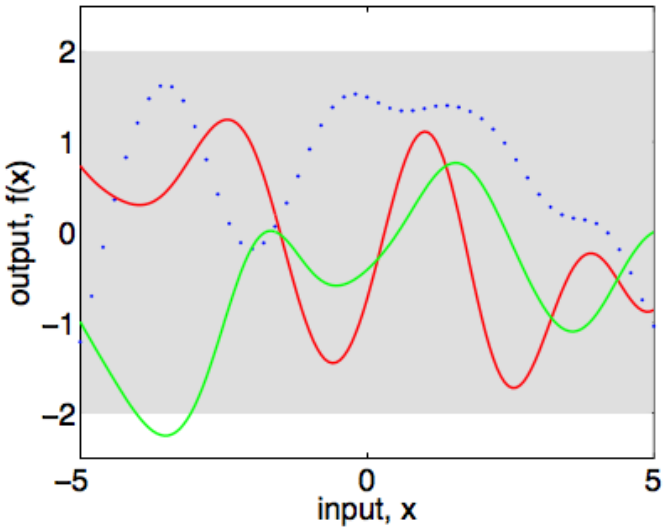
- Many algorithms can be kernelized, i.e. written in terms of kernels, rather than explicit feature representations.
- We rarely think about the underlying feature space explicitly. Instead, we build kernels directly.
- Useful composition rules for kernels:
  - A constant function $k(\mathbf{x}, \mathbf{x}') = \alpha$ is a kernel.
  - If k_1 and k_2 are kernels and $a, b \geq 0$, then $ak_1 + bk_2$ is a kernel.
  - If k_1 and k_2 are kernels, then the product $k(\mathbf{x}, \mathbf{x}') = k_1(\mathbf{x}, \mathbf{x}')k_2(\mathbf{x}, \mathbf{x}')$ is a kernel. (Interesting and surprising fact!)
- Before neural nets took over, kernel SVMs were probably the best-performing general-purpose classification algorithm.

GP Kernels

- One way to define a kernel function is to give a set of basis functions and put a Gaussian prior on $\mathbf{w}$.
- But we have lots of other options. Here's a useful one, called the squared-exp, or Gaussian, or radial basis function (RBF) kernel:

$$k_{\text{SE}}(\mathbf{x}_i, \mathbf{x}_j) = \sigma^2 \exp\left(-\frac{\|\mathbf{x}_i - \mathbf{x}_j\|^2}{2\ell^2}\right)$$

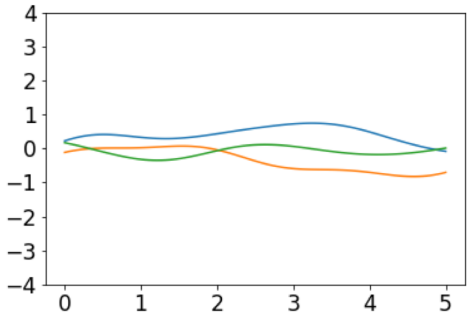
- More accurately, this is a kernel family with hyperparameters σ and ℓ.
- It gives a distribution over smooth functions:



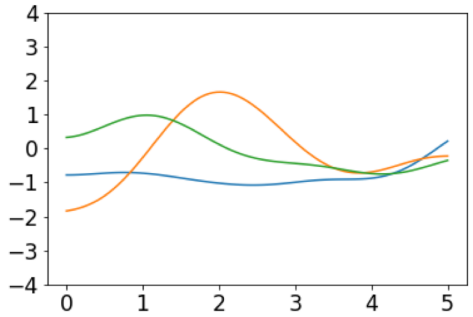
GP Kernels

$$k_{\text{SE}}(x_i, x_j) = \sigma^2 \exp\left(-\frac{(x_i - x_j)^2}{2\ell^2}\right)$$

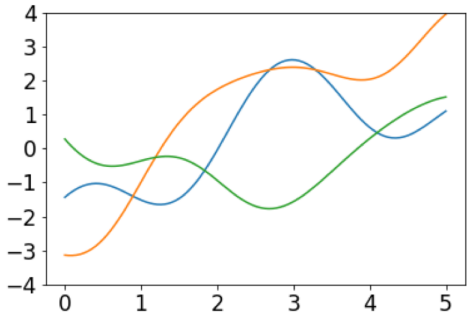
- The hyperparameters determine key properties of the function.
- Varying the output variance σ^2:



$\sigma^2 = 0.3$

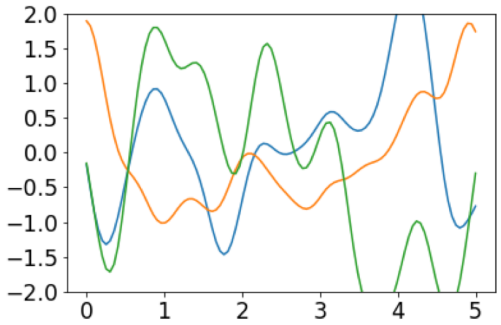


$\sigma^2 = 1$

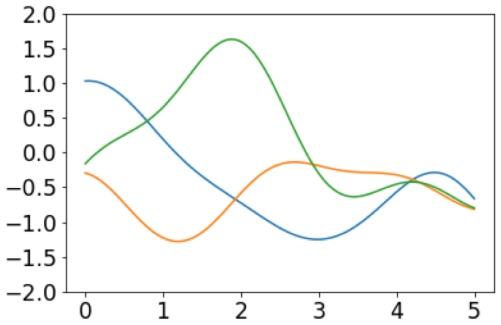


$\sigma^2 = 3$

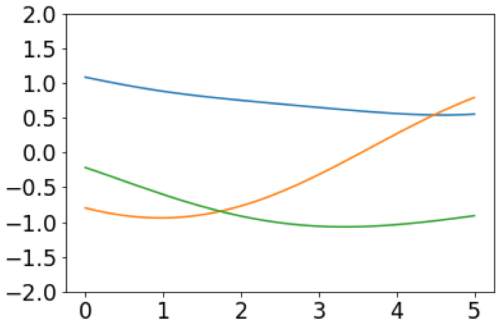
- Varying the lengthscale ℓ:



$\ell = 0.3$



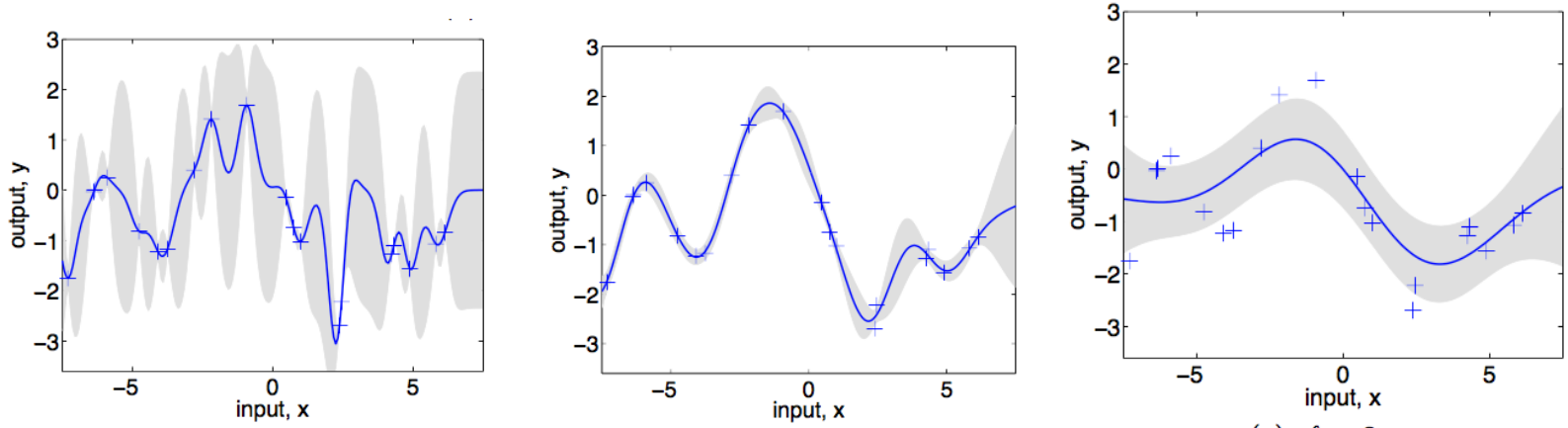
$\ell = 1$



$\ell = 3$

GP Kernels

- The choice of hyperparameters heavily influences the predictions:



(b), $\ell = 0.3$ (a), $\ell = 1$ (c), $\ell = 3$

- In practice, it's very important to tune the hyperparameters (e.g. by maximizing the marginal likelihood).

Wide BNNs $\rightarrow$ GPs

Wide Nets → GPs

Some insights from Radford Neal's visionary PhD thesis (1993):

- GPs are useful for machine learning
- (Bayesian) neural nets can generalize well despite being highly overparameterized
- Posterior sampling using HMC
- Infer the model complexity using Automatic Relevance Determination
- The infinite width limit of a Bayesian neural net is a GP

His presentation of BNNs is very close to our full modern understanding!

Wide BNNs → GPs

Two views of Bayesian regression:

Weight Space
(e.g. Bayesian linear regression)

$$p(t \mid \mathbf{x}, \mathcal{D}) = \int p(t \mid \mathbf{x}, \mathbf{w})\, p(\mathbf{w} \mid \mathcal{D})\, \mathrm{d}\mathbf{w}$$

$$p(\mathbf{w} \mid \mathcal{D}) = \frac{p(\mathbf{w})\, p(\mathcal{D} \mid \mathbf{w})}{p(\mathcal{D})}$$

Function Space
(e.g. GPs)

$$p(t \mid \mathbf{x}, \mathcal{D}) = \int p(t \mid \mathbf{x}, \mathbf{f})\, p(\mathbf{f} \mid \mathcal{D})\, \mathrm{d}\mathbf{f}$$

$$p(\mathbf{f} \mid \mathcal{D}) = \frac{p(\mathbf{f})\, p(\mathcal{D} \mid \mathbf{f})}{p(\mathcal{D})}$$

- $\mathbf{f}$ is a vector of function values (e.g. at training and query points)
- If we want to take a limit of models with different parameter spaces, we need to work in function space
- Since $p(t \mid \mathbf{x}, \mathbf{f})$ and $p(\mathcal{D} \mid \mathbf{f})$ depend only on the observation model (which we'll take as fixed), the important object to study is the prior $p(\mathbf{f})$

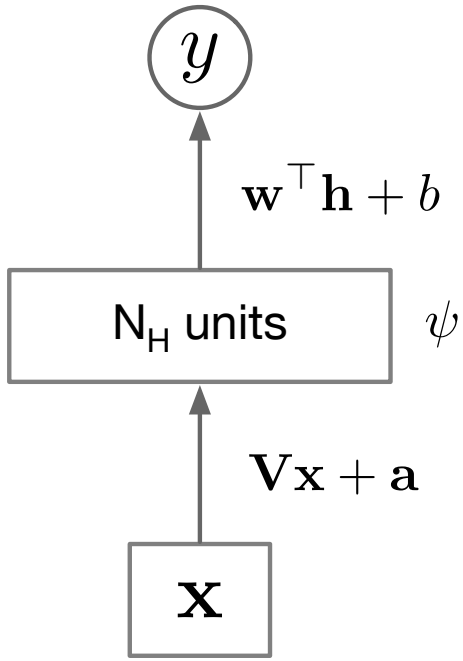
Wide BNNs → GPs

- Vanilla BNN model definition

$$y = f(\mathbf{x}) = \sum_i w_i h_i(\mathbf{x}) + b$$
$$= \sum_i w_i \psi(\mathbf{v}_i^\top \mathbf{x} + a_i) + b$$

- Priors (all independent)

$$w_i \sim \mathcal{N}(0, \sigma_w^2)$$
$$b \sim \mathcal{N}(0, \sigma_b^2)$$
$$v_{ij} \sim \mathcal{N}(0, \sigma_v^2)$$
$$a_i \sim \mathcal{N}(0, \sigma_a^2)$$



Wide BNNs → GPs

- Expectation of the function:

$$\begin{aligned}
\mathbb{E}[f(\mathbf{x})] &= \mathbb{E}\left[\sum_i w_i h_i(\mathbf{x}) + b\right] \\
&= \sum_i \mathbb{E}\left[w_i h_i(\mathbf{x})\right] + \underbrace{\mathbb{E}[b]}_{=0} \\
&= \sum_i \underbrace{\mathbb{E}[w_i]}_{=0} \mathbb{E}[h_i(\mathbf{x})] \qquad \text{(by independence)} \\
&= 0
\end{aligned}$$

Wide BNNs → GPs

- Variance of the function:

$$\begin{aligned}
\mathrm{Var}(f(\mathbf{x})) &= \mathrm{Var}\Big(\sum_i w_i h_i(\mathbf{x}) + b\Big) \\
&= N_H \,\mathrm{Var}(w_i h_i(\mathbf{x})) + \mathrm{Var}(b) && \text{(i.i.d. prior)} \\
&= N_H (\mathbb{E}[(w_i h_i(\mathbf{x}))^2] - \underbrace{\mathbb{E}[w_i h_i(\mathbf{x})]^2}_{=0}) + \mathrm{Var}(b) && \text{(prev. slide)} \\
&= N_H \,\mathbb{E}[w_i^2]\,\mathbb{E}[h_i(\mathbf{x})^2] + \mathrm{Var}(b) && \text{(independence)} \\
&= N_H \,\sigma_w^2\,\mathbb{E}[h_i(\mathbf{x})^2] + \sigma_b^2
\end{aligned}$$

- So we need to scale the variance as $\sigma_w^2 = \frac{\omega}{N_H}$ for some ω in order to have a consistent limit!

Wide BNNs → GPs

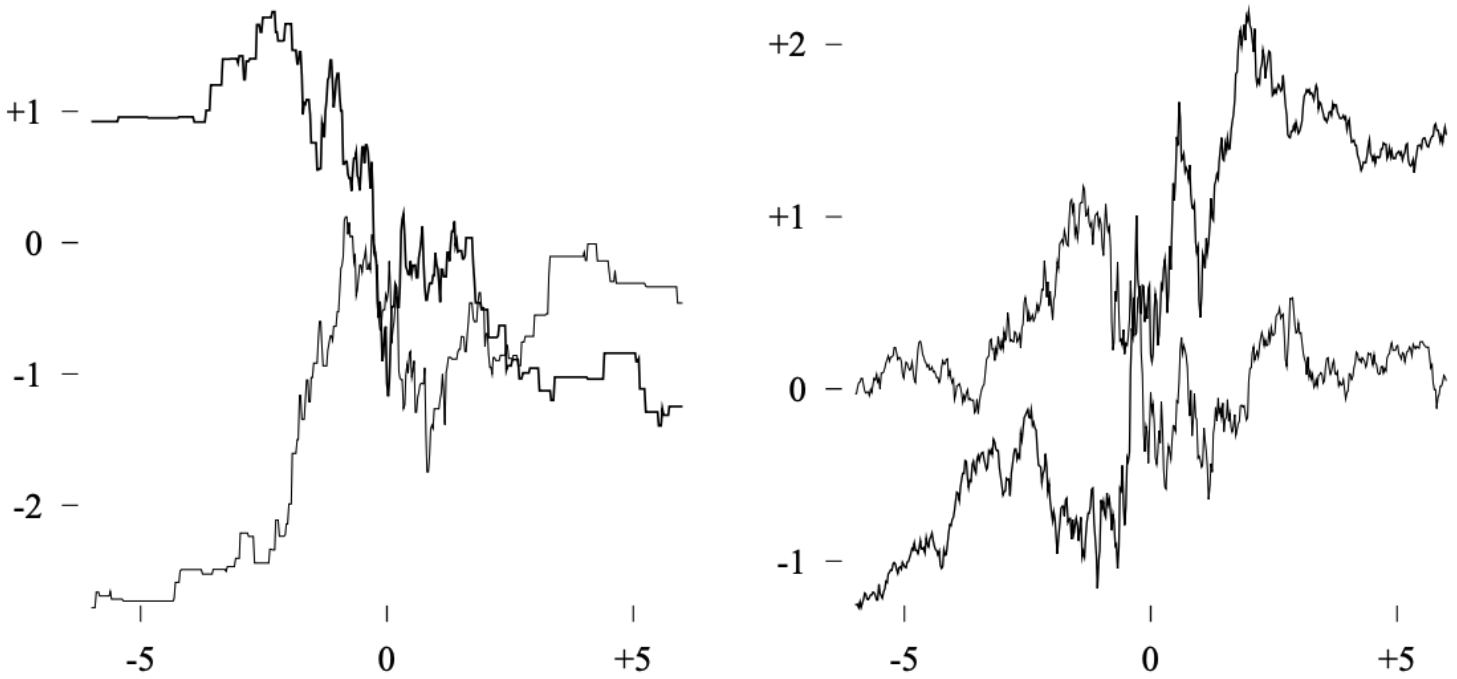
- Covariance of the function: with an analogous derivation,

$$\begin{aligned}\mathrm{Cov}(f(\mathbf{x}), f(\mathbf{x}')) &= \omega\, \mathbb{E}[h_i(\mathbf{x}) h_i(\mathbf{x}')] + \sigma_b^2 \\ &= \omega \int \psi(\mathbf{v}^\top \mathbf{x} + a)\, \psi(\mathbf{v}^\top \mathbf{x}' + a)\, p(\mathbf{v}, a)\, \mathrm{d}\{\mathbf{v}, a\} \;+\; \sigma_b^2\end{aligned}$$

- The vector of values $f(\mathbf{x})$ at various points $\mathbf{x}$ is the sum of i.i.d. random variables, so (assuming finite variance) a multivariate version of the Central Limit Theorem implies their limit is Gaussian
  - (informal?) so the limiting distribution over functions is a GP
- Neal's thesis derives kernels associated with various activation functions

Wide BNNs $\rightarrow$ GPs

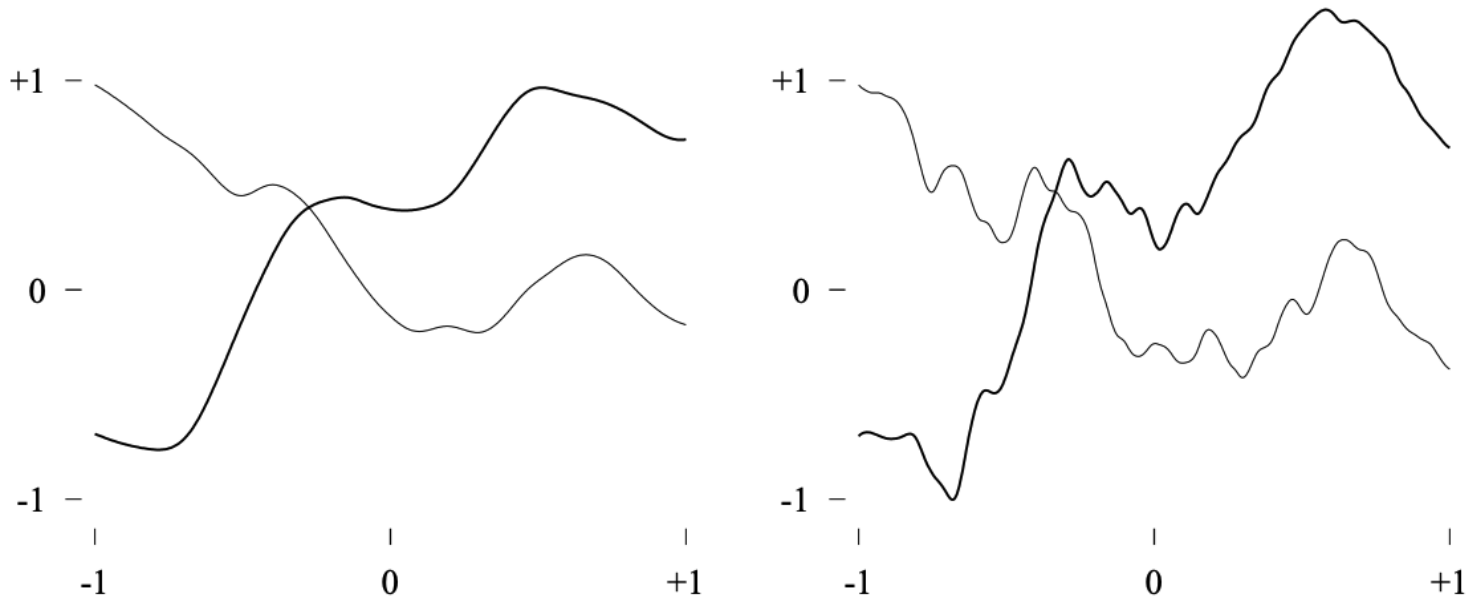
BNN with hard threshold activations $\rightarrow$ Brownian motion (Left: $N_H = 300$, Right: $N_H = 10000$)



(Neal, 1993)

Wide BNNs $\rightarrow$ GPs

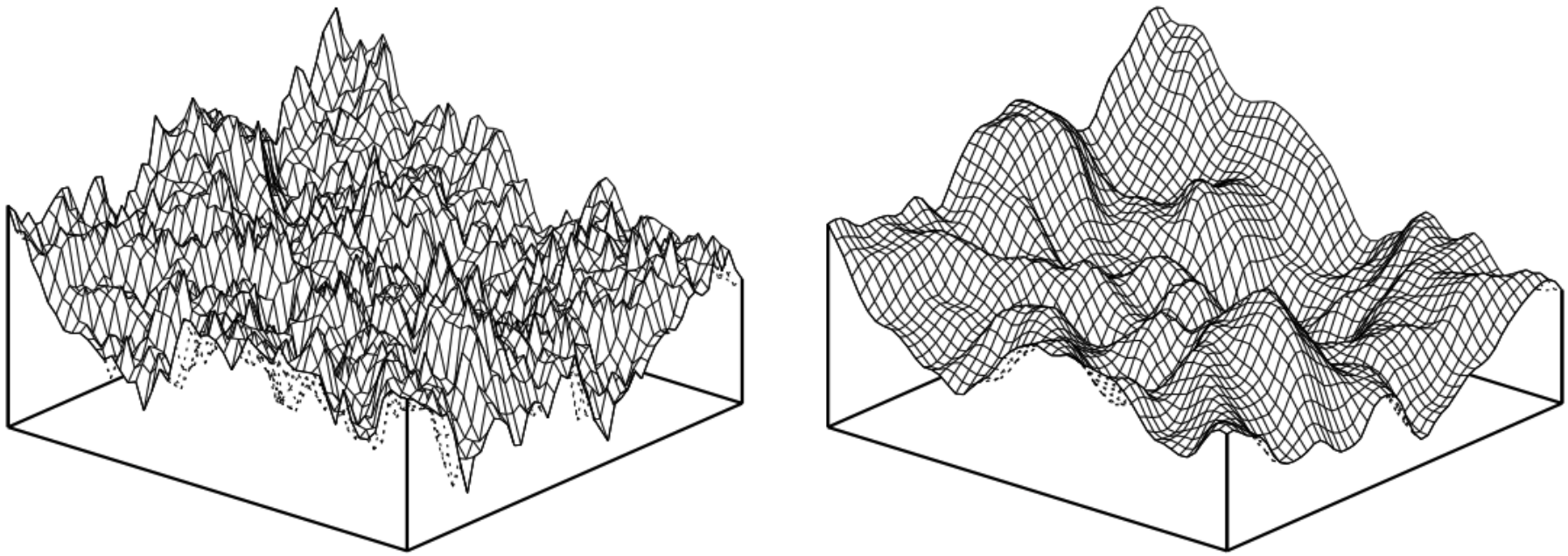
BNN with tanh activations, small $\sigma_v^2 \rightarrow$ smooth GP



(Neal, 1993)

Wide BNNs $\rightarrow$ GPs

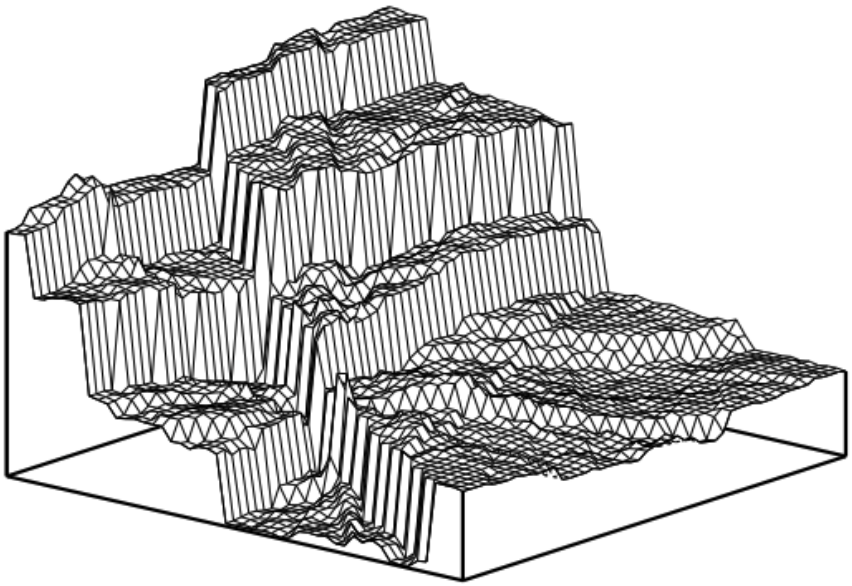
Two dimensions (Left: hard threshold, Right: tanh)

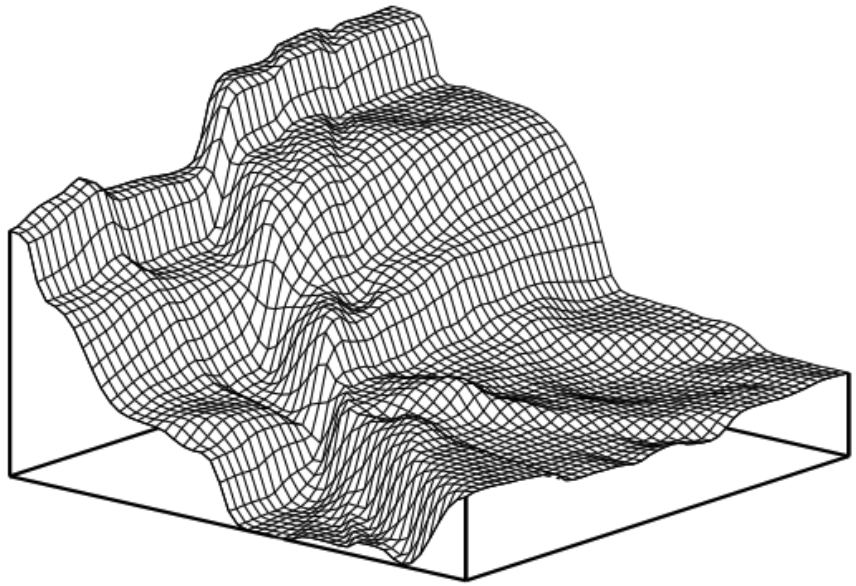


(Neal, 1993)

Wide BNNs $\rightarrow$ GPs

Going beyond GPs by sampling the weight variance for each hidden unit from a distribution rather than setting it to a fixed σ_v^2



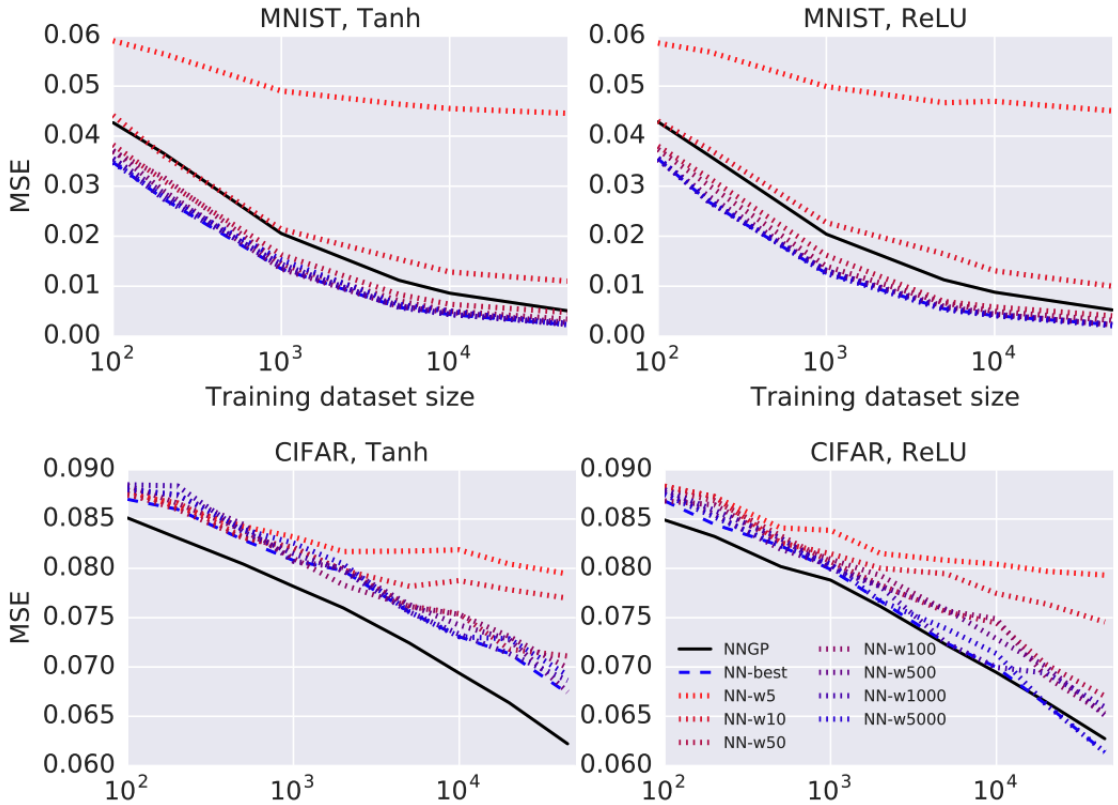


(Neal, 1993)

Wide BNNs → GPs

Lee et al., 2017, "Deep neural networks as Gaussian processes"

- Extended Neal's analysis to multilayer BNNs
- The infinite width limit is still a GP
- This limiting regime (distinct from the Neural Tangent Kernel regime, discussed next) is now called neural net Gaussian process (NNGP)



Mean Field Approximation

Poole et al., 2016, "Exponential expressivity in neural networks through transient chaos"

- Introduced the mean field approximation for studying deep, wide networks with random weights
  - Extension of Neal's analysis
- In Neal's analysis, the hidden units are all i.i.d. random variables, so (by the Central Limit Theorem) the sum of their contributions to the next layer is approximately Gaussian
  - Only important information about the distribution of h is the covariance function $\text{Cov}(h_i(\mathbf{x}), h_i(\mathbf{x}'))$
- Poole et al. apply this insight recursively and analyze how the covariance evolves as a function of depth

Mean Field Approximation

- Fully connected layer:

$$\mathbf{s}_\ell = \mathbf{W}_\ell \mathbf{a}_{\ell-1} + \mathbf{b}_\ell$$
$$\mathbf{a}_\ell = \phi(\mathbf{s}_\ell)$$

- By assumption, the weights are i.i.d. normal with variance σ_w^2 and the biases are i.i.d. normal with variance σ_b^2 (shared between layers for simplicity).
- Let $a_\ell^i(\mathbf{x})$ denote the ith hidden unit in layer ℓ evaluated for input $\mathbf{x}$. Similarly for $s_\ell^i(\mathbf{x})$.
- Neal's argument can be applied in each layer to show that the activations $a_\ell^i(\mathbf{x})$ are i.i.d. random variables and the pre-activations $s_\ell^i(\mathbf{x})$ are i.i.d. Gaussian random variables.
- We are interested in tracking the covariances $\text{Cov}(s_\ell^i(\mathbf{x}), s_\ell^i(\mathbf{x}'))$ for any given $\mathbf{x}$ and $\mathbf{x}'$. Note that by the i.i.d. property, these are the same for all i.

Mean Field Approximation

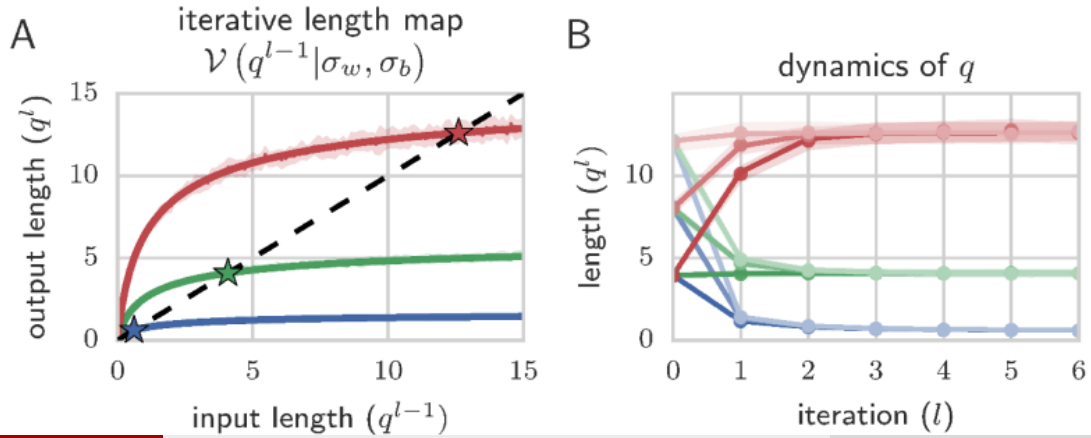
- We can track the normalized squared length of the hidden vector for each layer:

$$q^\ell = \frac{1}{N_\ell} \sum_{i=1}^{N_\ell} s_\ell^i(\mathbf{x})$$

- The values can be computed recursively using the V-map:

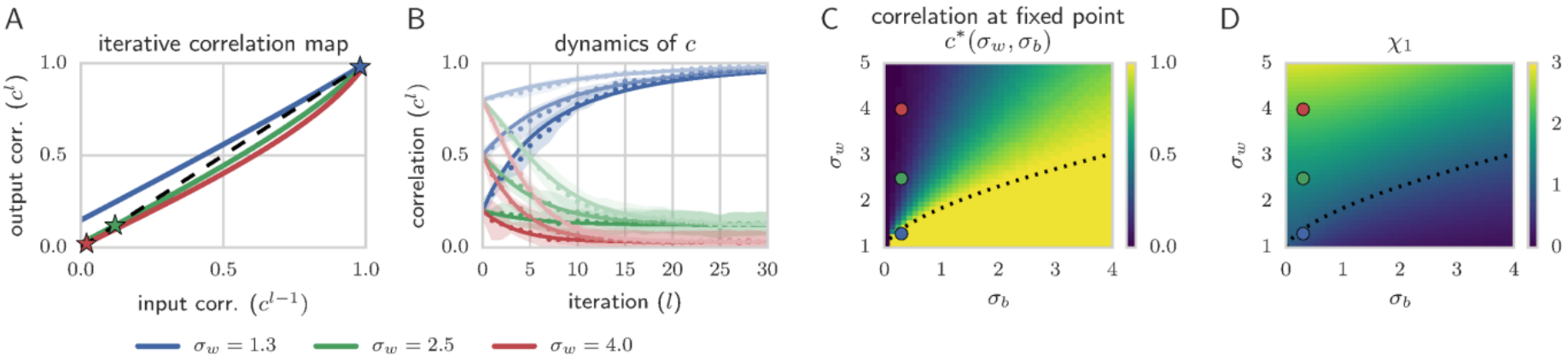
$$\begin{aligned} q^\ell &= V(q^{\ell-1}|\sigma_w, \sigma_b) \\ &\triangleq \sigma_w^2 \int \mathcal{N}(z; 0, 1)\phi(\sqrt{q^{\ell-1}}z)\,\mathrm{d}z \;+\; \sigma_b^2 \end{aligned}$$

- Examples of iterating this map with three different values of σ_w for a tanh network:



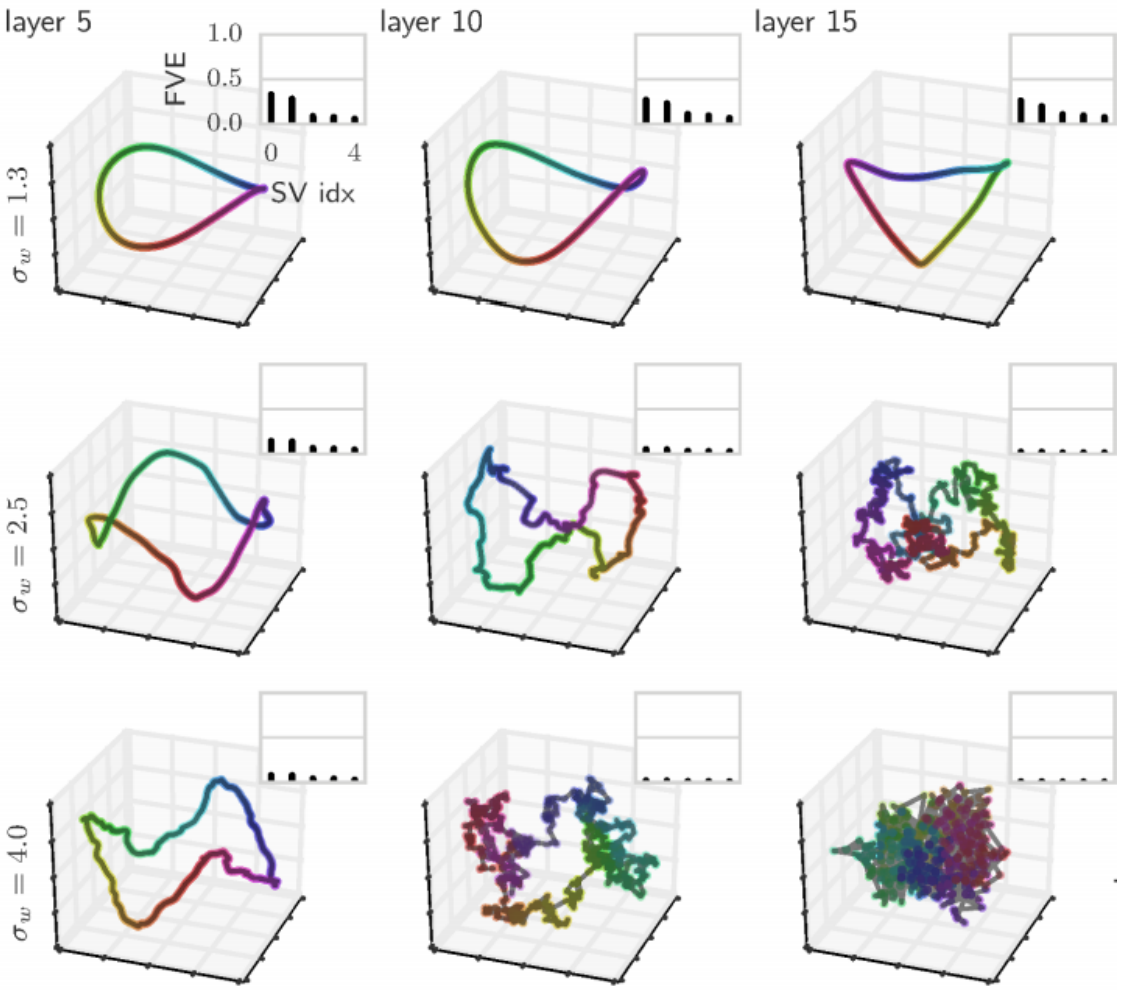
Mean Field Approximation

- The correlations can be tracked using an analogous map called the C-map.
- There are two regimes: an ordered regime (correlations $\to 1$, degenerate function) and a chaotic regime (correlations $\to 0$, exponential expressivity)
- You want to be on the boundary between them. This gives a way to choose σ_w and σ_b.



(Poole et al., 2016)

Mean Field Approximation



(Poole et al., 2016)

Deep Kernel Shaping

Martens et al., 2021, "Rapid training of deep neural networks without skip connections or normalization layers using Deep Kernel Shaping"

- The above mean field analysis was specific to multilayer perceptrons.
- In a 100-page tour-de-force, Martens et al. extended this basic analysis to a much wider range of architectures (e.g. ImageNet CNNs).
- The theory is accurate enough to suggest good initializations, transformations of the activation functions, etc.
- Impressively, they are able to eliminate batch norm and skip connections, and also use activation functions that are traditionally hard to train, like the logistic sigmoid.
- While they design the network based on the prior distribution of function values, the depth still creates ill-conditioning. Optimizing these architectures to near-SOTA accuracy required K-FAC.

Neural Tangent Kernel

Neural Tangent Kernel

- Bayesian inference and the GP limit give a lot of insight into how overparameterized neural networks can generalize well
- But explicitly training BNNs would be a radical departure from current practice
- Can we apply a similar interpretation to ordinary gradient descent on the sorts of networks we use routinely?
  - **Goal:** analyze the dynamics of gradient descent on an extremely wide neural net
  - Just like with BNNs, everything simplifies in the infinite limit
- In order to take the infinite limit, we need to work in function space

Output Space View of Linear Regression

- Consider linear regression (assume bias absorbed into $\boldsymbol{\phi}$):

$$y = \mathbf{w}^\top \boldsymbol{\phi}(\mathbf{x}) \qquad \mathbf{y} = \boldsymbol{\Phi}\mathbf{w}$$

- The output space view of gradient descent:

$$\begin{aligned}
\Delta \mathbf{y} &= \boldsymbol{\Phi} \Delta \mathbf{w} \\
&= \boldsymbol{\Phi}[-\alpha \nabla \mathcal{J}(\mathbf{w})] \\
&= -\frac{\alpha}{N} \underbrace{\boldsymbol{\Phi}\boldsymbol{\Phi}^\top}_{=\mathbf{K}} (\mathbf{y} - \mathbf{t})
\end{aligned}$$

- The matrix $\mathbf{K} = \boldsymbol{\Phi}\boldsymbol{\Phi}^\top$ is the Gram matrix
  - Same as the Kernel matrix $\mathbf{K}$ for a GP, if we put a spherical Gaussian prior on $\mathbf{w}$
- Solving the recurrence:

$$\mathbf{y}^{(k)} = \mathbf{t} + (\mathbf{I} - \frac{\alpha}{N}\mathbf{K})^k (\mathbf{y}^{(0)} - \mathbf{t})$$

Output Space View of Linear Regression

$$\mathbf{K} = \mathbf{\Phi}\mathbf{\Phi}^\top$$

- **Recall:** the Hessian for linear regression is $\mathbf{H} = \mathbf{\Phi}^\top\mathbf{\Phi}$ (if we remove the typical $1/N$ scaling from the cost function)
- **Observe:** $\mathbf{H}$ and $\mathbf{K}$ are symmetric matrices which share the same nonzero eigenvalues, which are the squared singular values of $\mathbf{\Phi}$
- If $\mathbf{\Phi} = \mathbf{U}\mathbf{D}\mathbf{V}^\top$ is the SVD of $\mathbf{\Phi}$, then

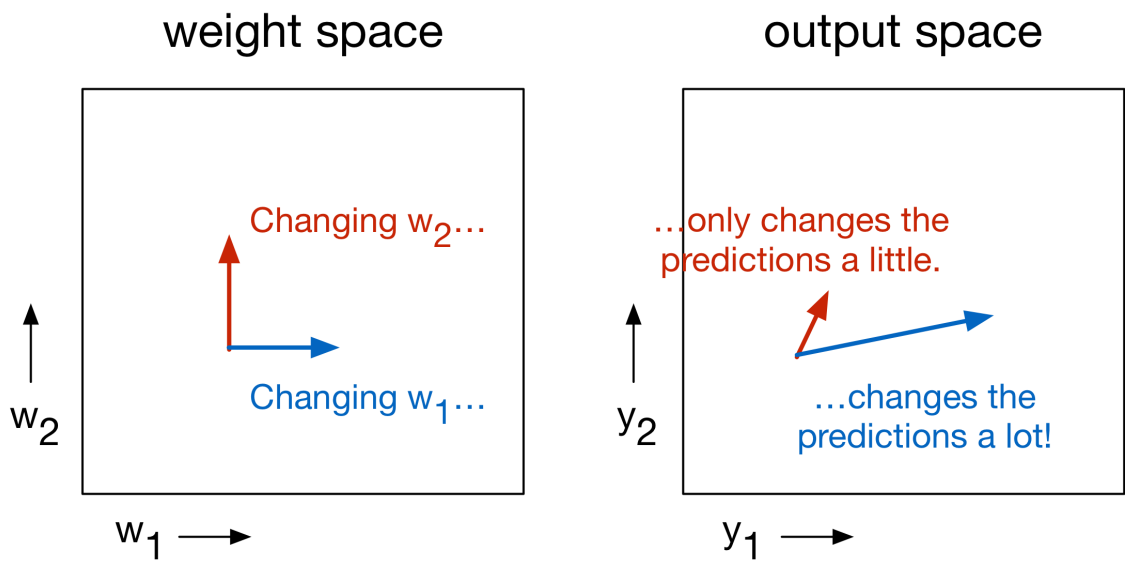
$$\mathbf{H} = \mathbf{V}\mathbf{D}^2\mathbf{V}^\top \qquad \text{(spectral decomposition of } \mathbf{H}\text{)}$$
$$\mathbf{K} = \mathbf{U}\mathbf{D}^2\mathbf{U}^\top \qquad \text{(spectral decomposition of } \mathbf{K}\text{)}$$

- **Interpretation:** the directions of high curvature are the directions of high sensitivity, i.e. the directions in weight space that have the largest effect on the predictions

Output Space View of Linear Regression

Recall from Lecture 1

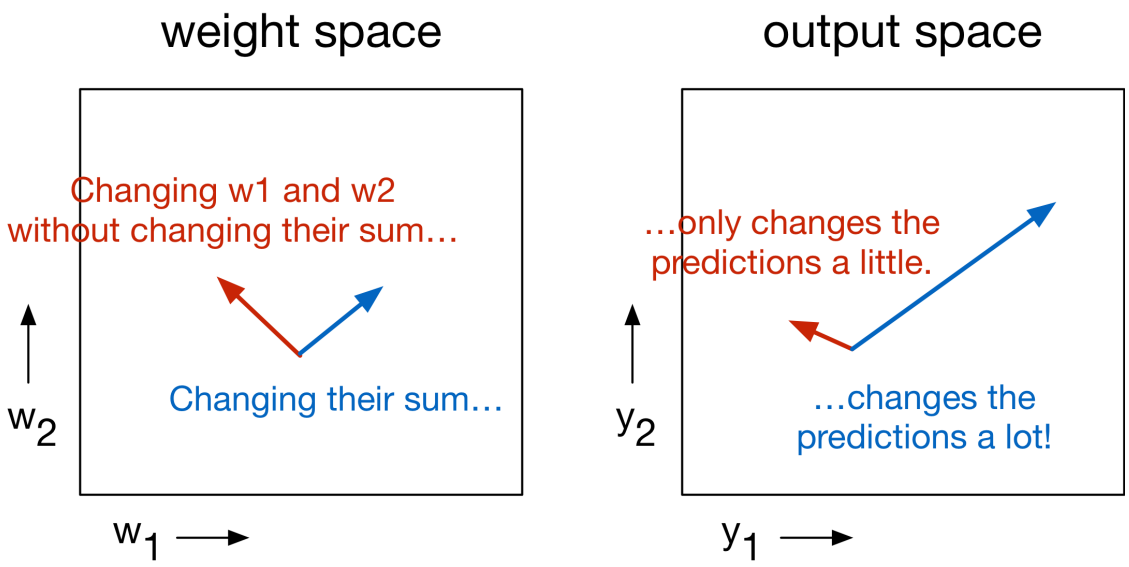
x_1	x_2	t
114.8	0.00323	5.1
338.1	0.00183	3.2
98.8	0.00279	4.1
⋮	⋮	⋮



Output Space View of Linear Regression

Recall from Lecture 1

x_1	x_2	t
1003.2	1005.1	3.3
1001.1	1008.2	4.8
998.3	1003.4	2.9
⋮	⋮	⋮



Neural Tangent Kernel

- We can apply a similar analysis to neural networks
- Consider full batch gradient descent (no straightforward way to apply this to SGD!)
- Let $\bar{\mathbf{z}}$ denote all of the outputs (e.g. logits) on the entire dataset, stacked into a vector, and $\bar{\mathbf{J}}$ be the Jacobian of $\bar{\mathbf{z}}$ with respect to $\mathbf{w}$

$$\begin{aligned}\Delta\bar{\mathbf{z}} &\approx \bar{\mathbf{J}}\Delta\mathbf{w} \\ &= \bar{\mathbf{J}}[-\alpha\nabla\mathcal{J}(\mathbf{w})] \\ &= \bar{\mathbf{J}}[-\frac{\alpha}{N}\bar{\mathbf{J}}^\top\nabla\mathcal{L}(\bar{\mathbf{z}})] \\ &= -\frac{\alpha}{N}\underbrace{\bar{\mathbf{J}}\bar{\mathbf{J}}^\top}_{=\mathbf{K}}\nabla\mathcal{L}(\bar{\mathbf{z}})\end{aligned}$$

- The matrix $\mathbf{K} = \bar{\mathbf{J}}\bar{\mathbf{J}}^\top$ is the neural tangent kernel (NTK)
- Unlike for regression, the above model is only approximate because
  - $\Delta\bar{\mathbf{z}}$ is a nonlinear function of $\Delta\mathbf{w}$
  - $\mathbf{K}$ changes over time

Neural Tangent Kernel

$$\mathbf{K} = \bar{\mathbf{J}}\bar{\mathbf{J}}^\top$$

- Consider the finite sample approximation to the Gauss-Newton matrix ($\mathbf{J}_i$ is the Jacobian for training example i)

$$\mathbf{G} = \frac{1}{N}\sum_i \mathbf{J}_i^\top \mathbf{J}_i = \frac{1}{N}\bar{\mathbf{J}}^\top \bar{\mathbf{J}}$$

  - Gauss-Newton Hessian for squared error loss
  - pullback metric for Euclidean distance
- $\mathbf{G}$ and $\mathbf{K}$ have the same eigenvalues (up to scaling), which are the squared singular vectors of $\bar{\mathbf{J}}$
  - $\bar{\mathbf{J}}$ measures the sensitivity of the predictions to a direction in weight space

Neural Tangent Kernel

- This interpretation becomes exact when we consider the gradient flow, the continuous time limit of gradient descent (i.e. lots of steps with tiny learning rate)

$$\frac{\mathrm{d}\mathbf{w}}{\mathrm{d}t} = -\alpha \nabla \mathcal{J}(\mathbf{w})$$

- The flow in output space is:

$$\frac{\mathrm{d}\bar{\mathbf{z}}}{\mathrm{d}t} = -\frac{\alpha}{N}\mathbf{K}(t)\nabla \mathcal{L}(\bar{\mathbf{z}})$$

- I wrote $\mathbf{K}(t)$ to remind us that $\mathbf{K}$ is time dependent (which makes this ODE difficult to solve in general)

Neural Tangent Kernel

Jacot et al., 2018. "Neural tangent kernel: Convergence and generalization in neural networks"

- Considers a wide neural net limit distinct from the NNGP one (main difference is the effective learning rates, in the sense of Lecture 5)
- As the width goes to infinity, $\mathbf{K}$ approaches a well-defined limit
- As the width increases, the distance in weight space required to fit the training set goes to 0
- In the limit, $\bar{\mathbf{J}}$, and therefore, $\mathbf{K}$, are constant
- The flow for a regression problem (just a linear ODE!)

$$\frac{\mathrm{d}\bar{\mathbf{z}}}{\mathrm{d}t} = -\frac{\alpha}{N}\mathbf{K}(\mathbf{y} - \mathbf{t})$$

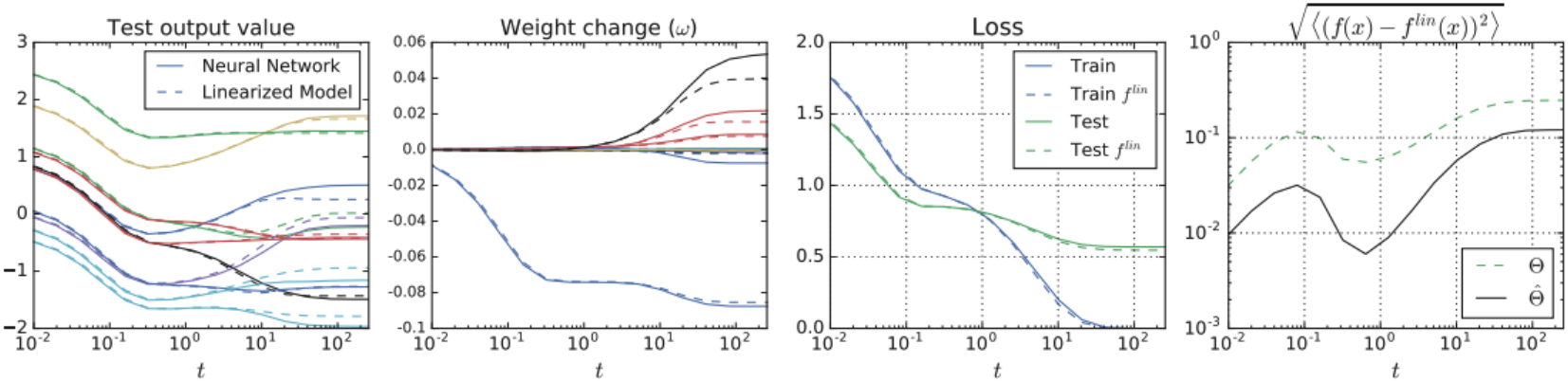
$$\mathbf{y}(t) = \mathbf{t} + \exp\left(-\frac{\alpha t}{N}\mathbf{K}\right)(\mathbf{y}(0) - \mathbf{t})$$

- More details in the student presentation next week

Neural Tangent Kernel

Lee et al., 2019. "Wide networks of any depth evolve as linear models under gradient descent"

- For wide (but finite) networks, $\bar{\mathbf{J}}$ changes slowly enough over training that the network is well approximated by its first-order Taylor approximation around $\mathbf{w}_0$
  - I.e., it behaves like a linear model, where the features are the columns of $\bar{\mathbf{J}}$
- This requires wider networks than we normally use (but not ridiculously so), a smaller learning rate, and full batch training
- There's still a gap between linearized training and SOTA, so probably neural nets are more than just linear random feature models



Neural Tangent Kernel

- These ideas lead to provable bounds for wide but finite networks. A big challenge is proving that the Jacobian changes slowly enough with high probability.
- Du et al., 2019. "Gradient descent provably optimizes over-parameterized neural networks"
  - **Optimization:** gradient descent on a randomly initialized wide network provably converges linearly to a global optimum (despite non-convexity!)
- Arora et al., 2019. "Fine-grained analysis of optimization and generalization for overparameterized two-layer neural networks"
  - **Generalization:** if the training and validation labels are well-aligned with the large eigenvalues of $\mathbf{K}$, then a network trained with gradient descent will generalize well (despite overparameterization!)
  - Sort of like a function space view of the min-norm analyses from Lecture 1

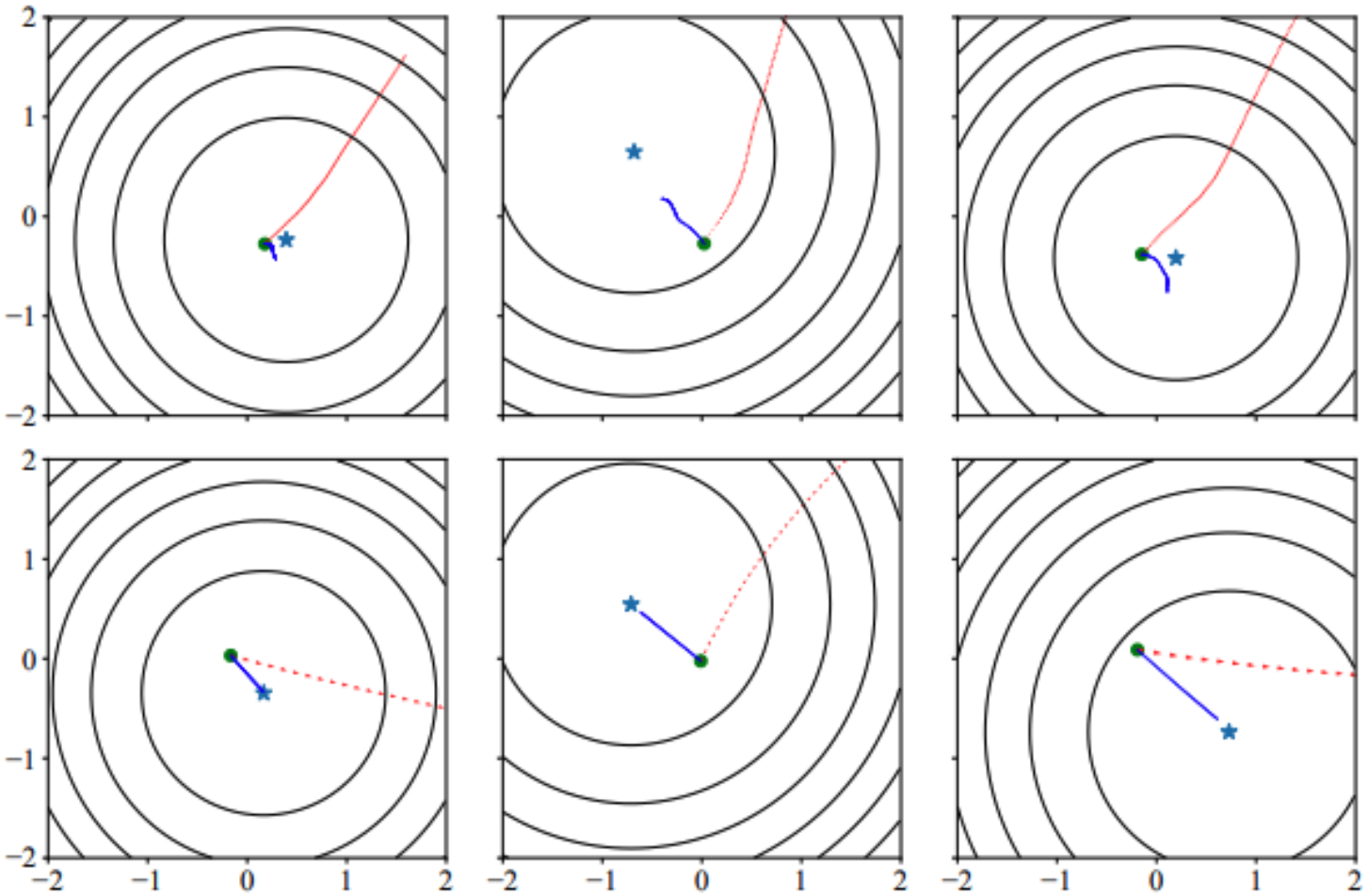
Neural Tangent Kernel

Zhang et al., 2019. "Fast convergence of natural gradient descent for overparameterized neural networks"

- In Lecture 3, we motivated natural gradient descent as an approximation to "gradient descent on the outputs"
- In the infinite width limit, because the network becomes more linear, this interpretation becomes more accurate.
- As a result can prove faster convergence rates for (exact) NGD than the analogous wide network results for GD.

Neural Tangent Kernel

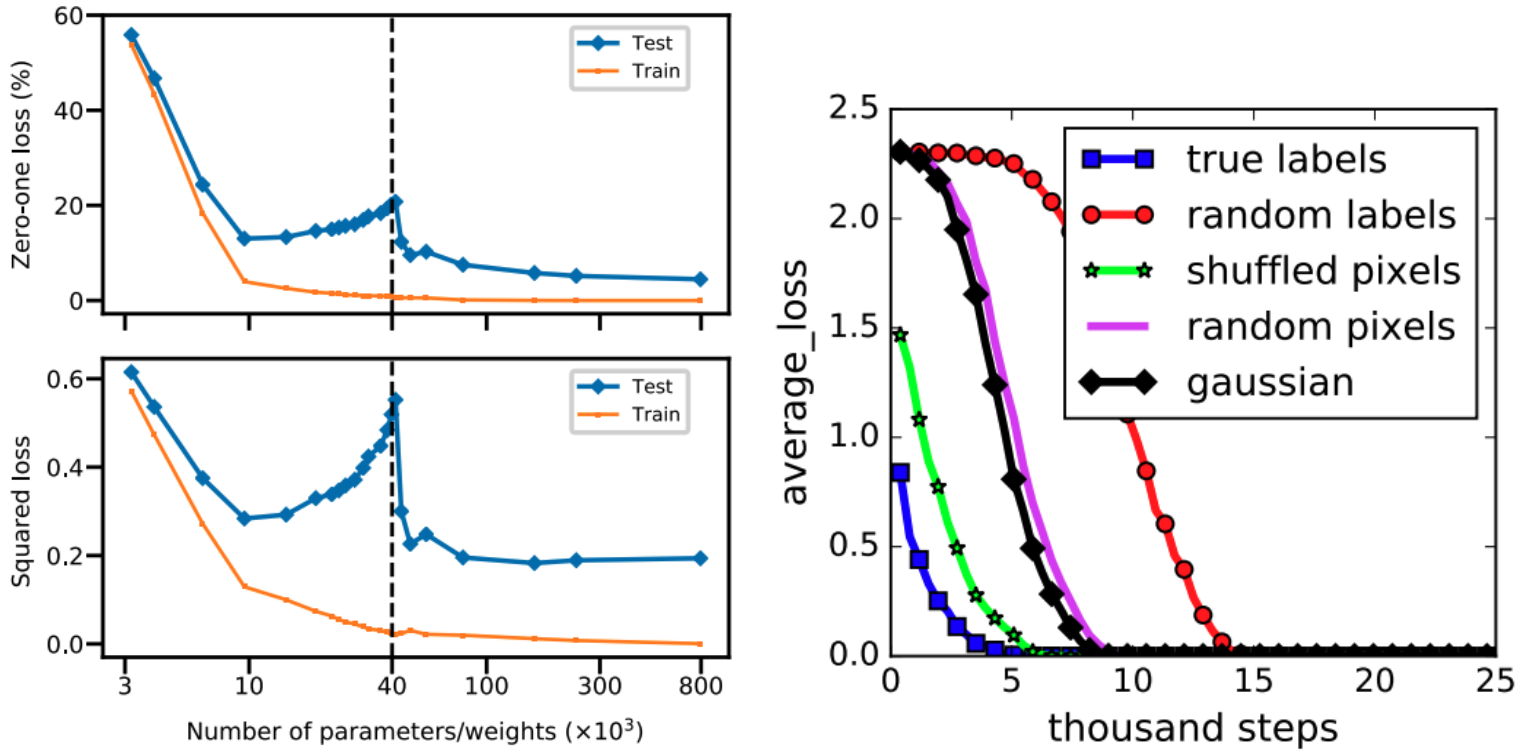
Output space trajectories for GD and NGD updates. **Top:** 100 units/layer. **Bottom:** 6000 units/layer.



Min-Norm Bias and Generalization

Min-Norm Bias

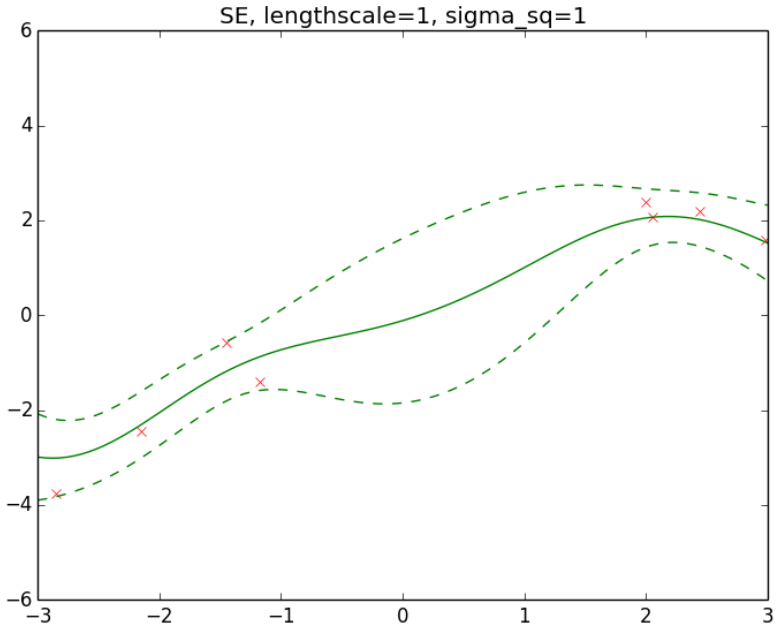
Recall from Lecture 1:



The regime of overparameterized models, where the model can fit the training data exactly but still generalize well, is the interpolation regime (as distinguished from the classical regime of underparameterized models).

Min-Norm Bias

- This isn't specific to neural nets. We can see it in Gaussian processes as well.
- Suppose we are trying to fit a regression dataset with some label noise. If we fit a GP with a calibrated noise estimate, we can fit it pretty well:

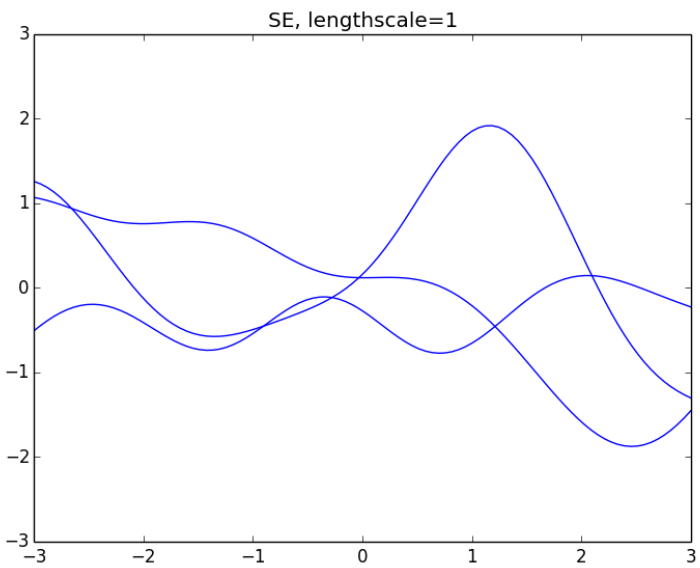


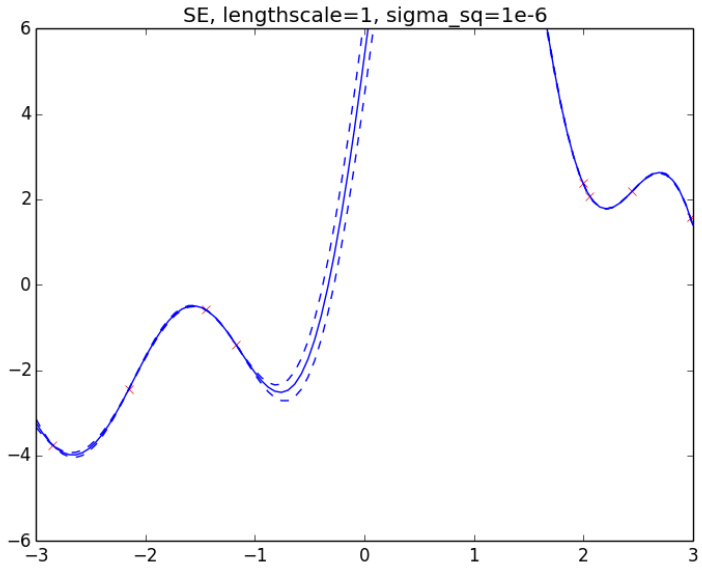
Min-Norm Bias

- But suppose we assume zero noise, i.e. force the GP to fit the data exactly. This may or may not work, depending on the kernel.
- Squared-exp kernel

$$k_{\text{SE}}(\mathbf{x}_i, \mathbf{x}_j) = \sigma^2 \exp\left(-\frac{\|\mathbf{x}_i - \mathbf{x}_j\|^2}{2\ell^2}\right)$$

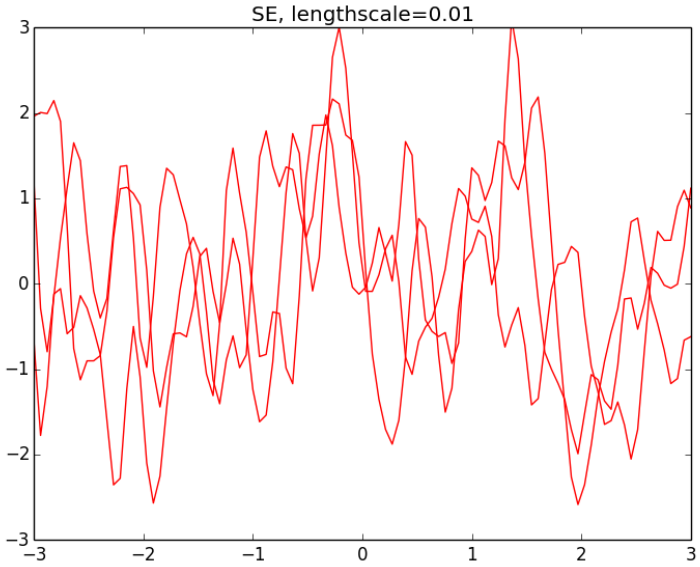
- Assume a long lengthscale, which produces smooth functions. **Left:** prior samples, **Right:** posterior predictive distribution

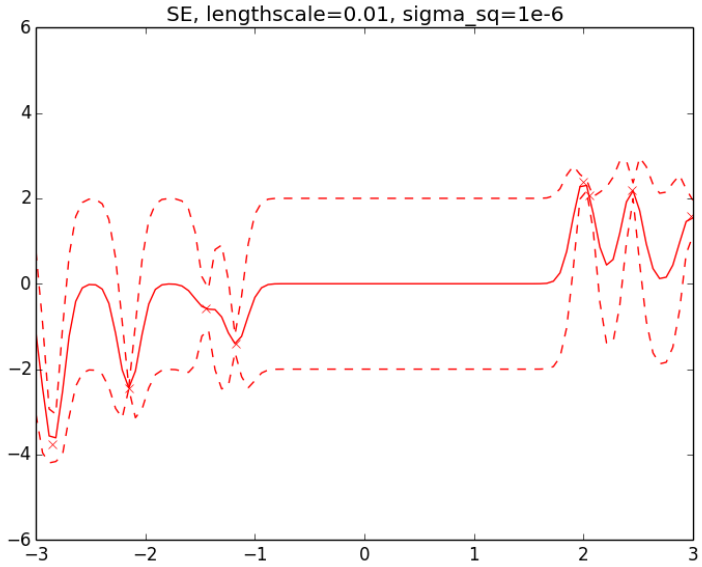




Min-Norm Bias

- With a shorter lengthscale, it produces very wiggly functions:



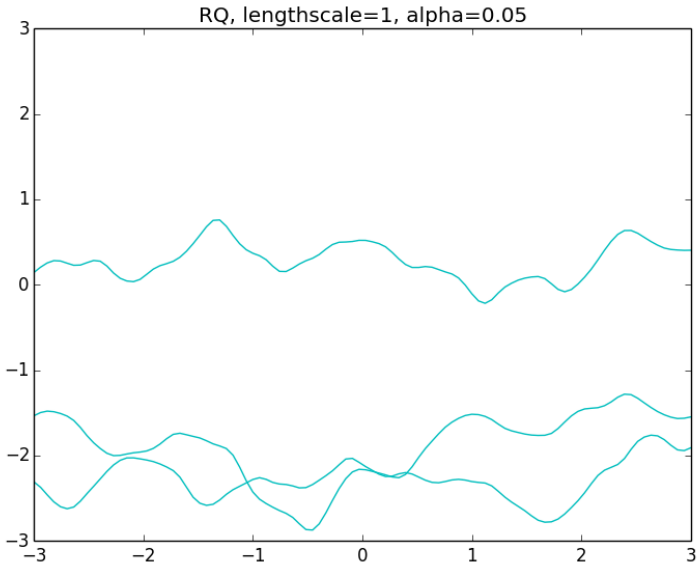


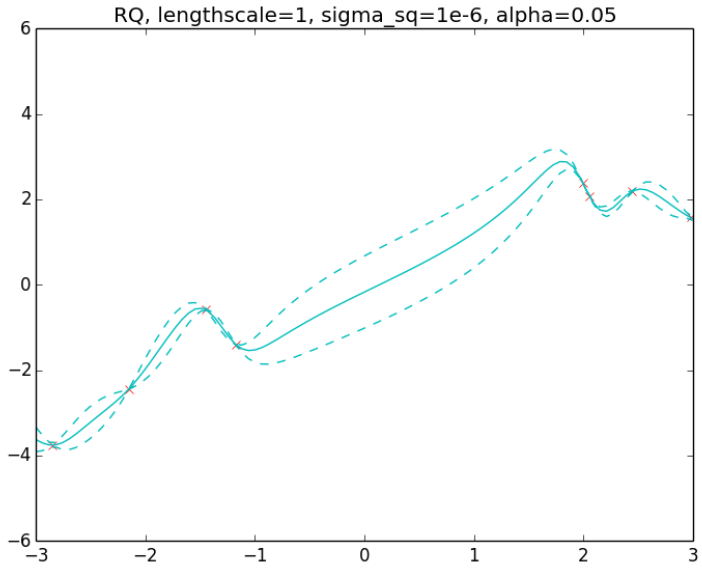
Min-Norm Bias

- Now consier the rational-quadratic kernel:

$$k_{\text{RQ}}(x_i, x_j) = \sigma^2 \left(1 + \frac{(x - x')^2}{2\alpha\ell^2}\right)^{-\alpha}$$

- This one seems to work better:





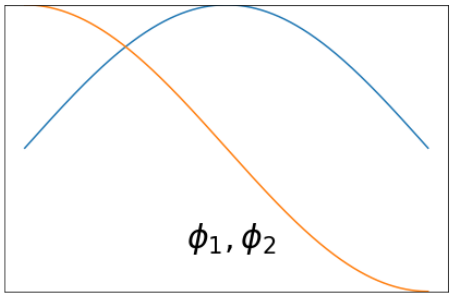
- Why does this behave differently from the previous two?

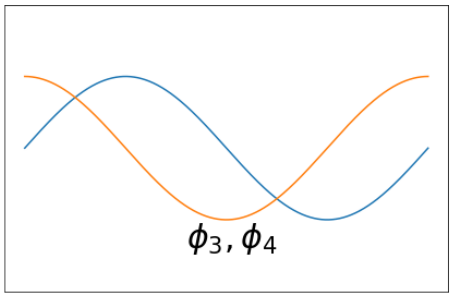
Min-Norm Bias

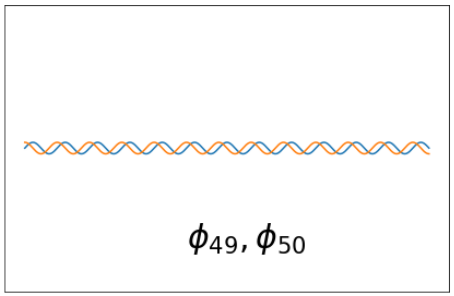
- Consider a linear regression model on top of a Fourier basis:

$$\begin{aligned} \phi_0(x) &= 1 \\ \phi_{2k-1}(x) &= k^{-\alpha} \sin k\pi x && \text{for } k = 1, \ldots, 25 \\ \phi_{2k}(x) &= k^{-\alpha} \cos k\pi x && \text{for } k = 1, \ldots, 25 \end{aligned}$$

- Here, α is a hyperparameter that controls the strength of high vs. low frequencies. For instance, if $\alpha = 1$:

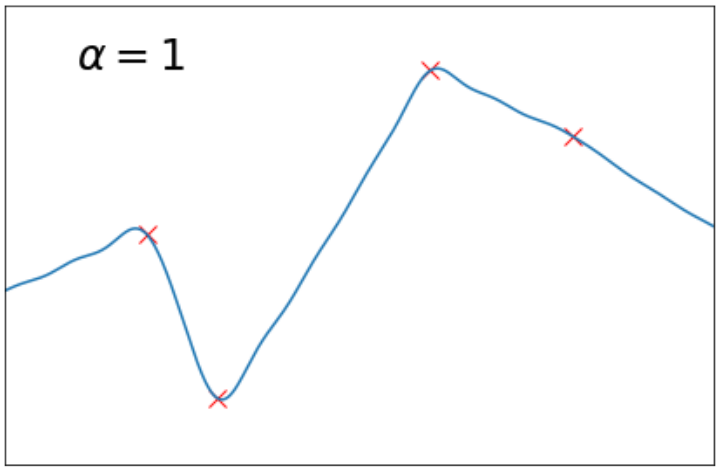






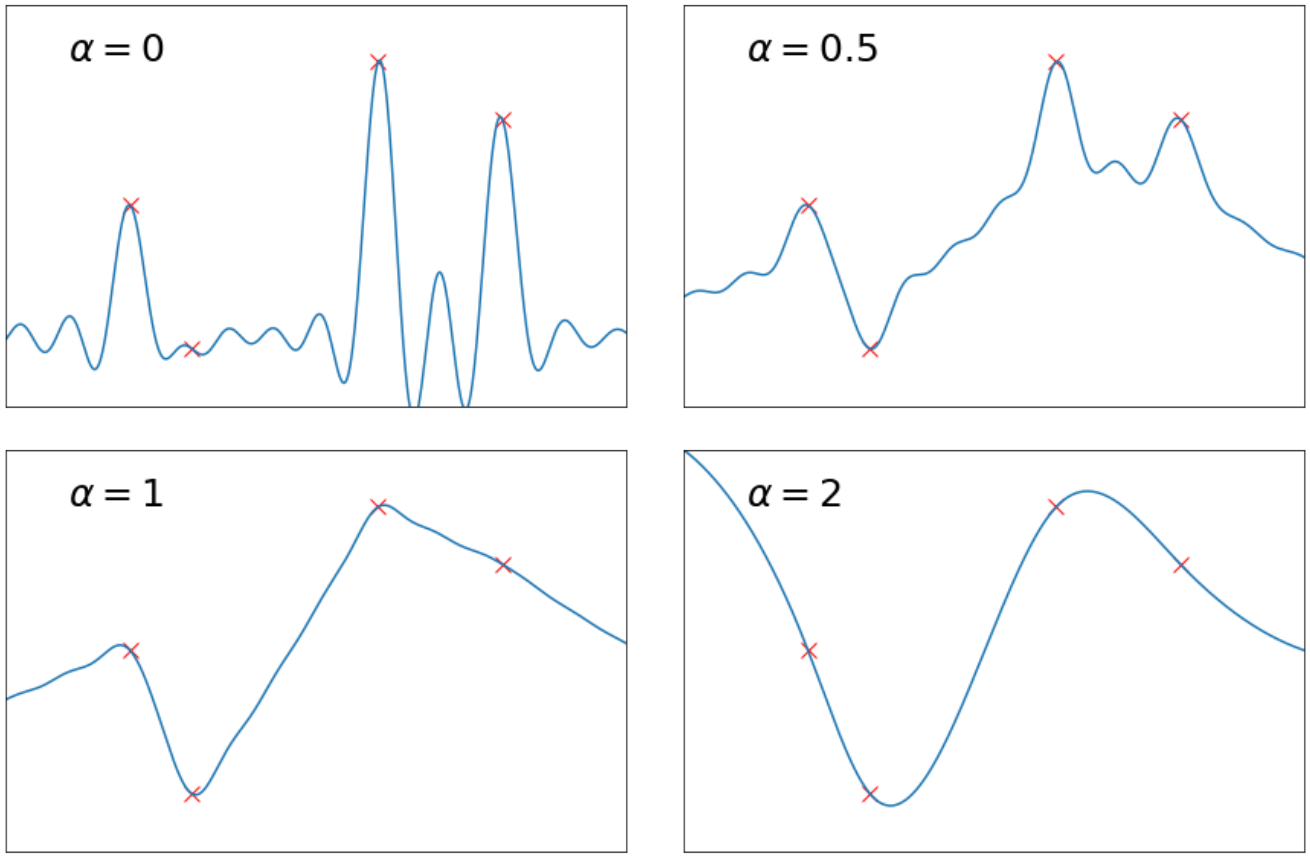
Min-Norm Bias

- With 51 features, it can fit a training set of 4 data points exactly.
- In Lecture 1, we noted that gradient descent on a quadratic objective converges to the min-norm solution, i.e. the one closest to the initialization in Euclidean distance.
- So what is the min-norm solution?
  - Since the lower frequency features are larger in magnitude, it "costs" less to use them.
  - Therefore, it will explain as much as possible using low frequency information. I.e., it has an inductive bias towards smooth functions.
- Here's the min-norm solution for $\alpha = 1$:



Min-Norm Bias

- The hyperparameter α controls how the amplitude decays with frequency.
- Larger α implies faster decay, and hence a stronger inductive bias towards smoothness.
- Here are the fits with different values of α:



Min-Norm Bias

How does this relate to GPs?

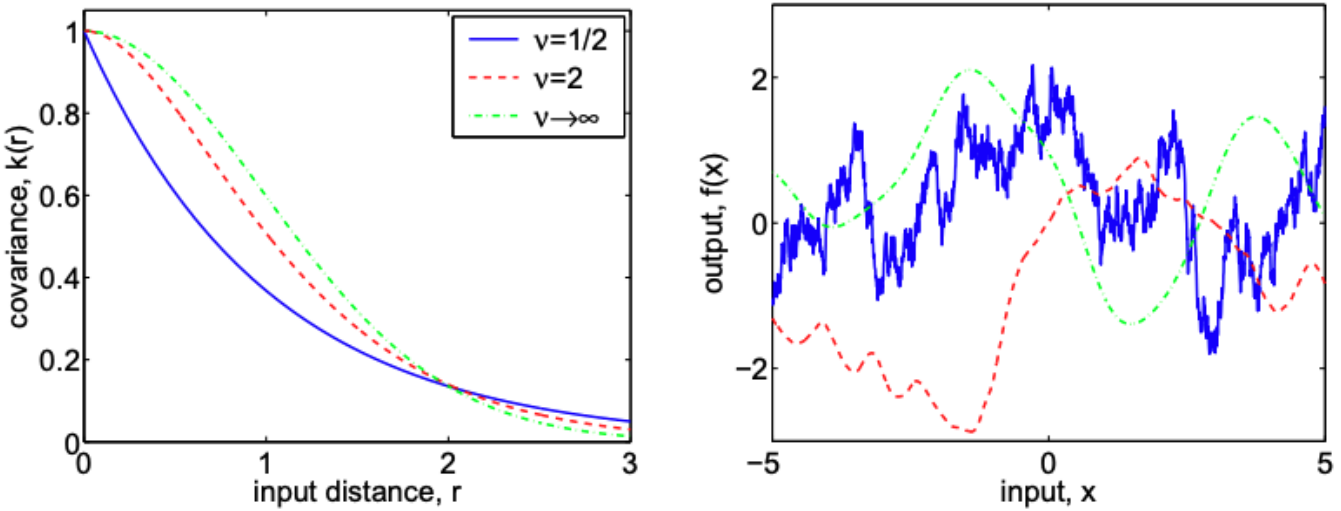
- Rasmussen and Ghahramani, "Occam's Razor." NeurIPS, 2000.
  - Analyzes Bayesian Fourier regression models like the ones we just discussed, and shows that they become equivalent to GPs in the limit as the number of features goes to ∞.
- Rahimi and Recht, "Random features for large-scale kernel machines." NeurIPS, 2007.
  - Showed that inference in kernelized models (e.g. GPs) with stationary kernels can be approximated by sampling random Fourier features.
  - The frequencies are sampled from a distribution rather than spaced equally.
  - Ironically, this is the same paper for which the authors gave the Test of Time talk criticizing batch norm and comparing deep learning to alchemy.

Min-Norm Bias

- A kernel is stationary if it's translation invariant. Examples include squared-exp and rational-quadratic (discussed earlier).
- A stationary kernel in 1-D can be written as a function of the distance $r = |x - x'|$ between two points:

$$k(x, x') = k(r)$$

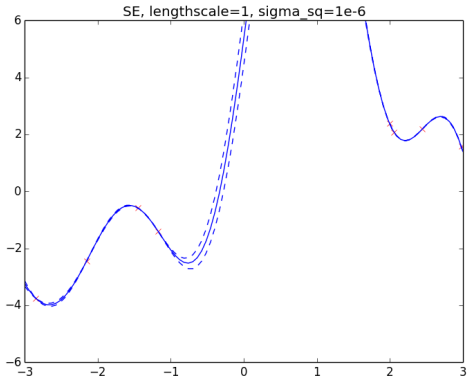
- The behavior of k near $r = 0$ determines the roughness of the prior samples:

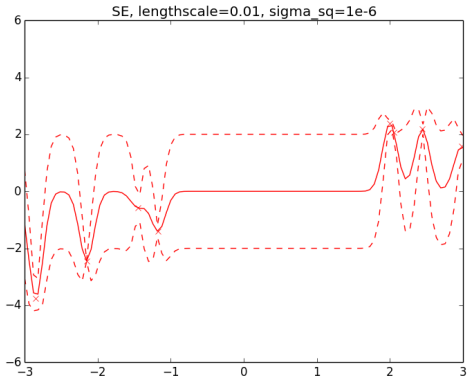


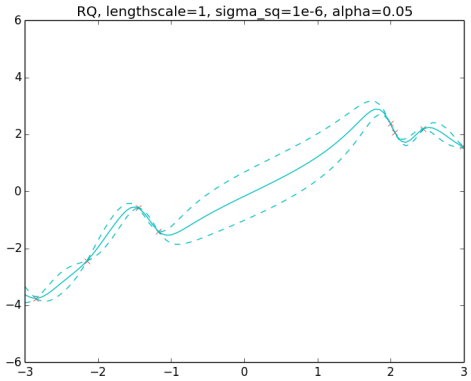
- k is closely related to the C-maps discussed earlier.

Min-Norm Bias

- Bochner's Theorem gives a way to characterize GP kernels in terms of their power spectral density, which is essentially the Fourier transform of the kernel.
- Like in the finite Fourier regression examples, the rate at which the amplitude decays with frequency determines the roughness of the samples. More power in the high frequencies implies rougher functions.
- The SE kernel has a very fast decay in the PSD. The RQ kernel has a much heavier tail.
- See Chapter 4 of *Gaussian Processes for Machine Learning*







Min-Norm Bias

- Because the rational quadratic GP has a lot of power in the high frequencies, the high frequencies can be used to represent noise in the labels.
- This allows it to fit the training labels exactly without significantly harming the rest of the predictions. This is known as benign overfitting.
- Bartlett et al., "Benign overfitting in linear regression." PNAS, 2020.
  - They analyze when benign overfitting occurs in terms of the covariance $\boldsymbol{\Sigma}$ of the features.
  - The relevant features need to be concentrated in the high variance directions, and $\boldsymbol{\Sigma}$ needs to have a heavy tailed eigenspectrum, so that the model can implicitly represent noise.
  - Analyzing the effect of label noise comes down to bounding the norm of $\boldsymbol{\Phi}^{\dagger}$, as alluded to in Lecture 1.

Min-Norm Bias

Amari et al., "When does preconditioning help or hurt generalization?" ICLR, 2021

- The previous analysis all focused on gradient descent. For quadratic objectives (e.g. linear regression), GD minimizes the *Euclidean* distance to the initialization.
- With a fixed preconditioner (e.g. natural gradient for linear regression), a different norm is implicitly regularized.
- **Intuition:** by compensating for the feature covariance, NGD tends to flatten the power spectrum. Therefore, it has less of an inductive bias towards smoother functions.
- This paper decomposes the generalization error for both GD and NGD into bias and variance terms and shows how these depend on various problem features (dimension, number of training examples, etc.). Neither method dominates the other.