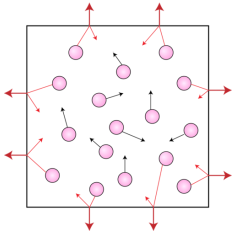
# CSC 2541: Neural Net Training Dynamics

## 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:

- Review of Bayesian regression and kernels
  - Bayesian linear regression
  - Gaussian processes
- 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

## Recap: Full Bayesian Inference

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

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

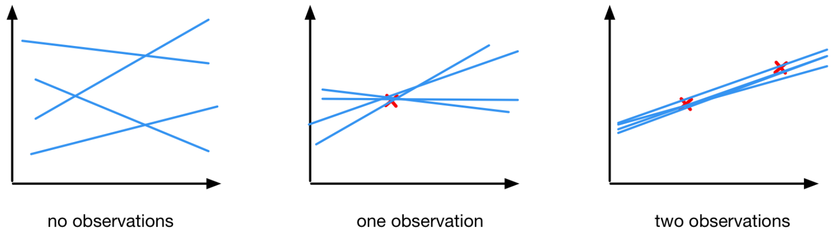
- Make predictions using the posterior predictive distribution:

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

- Doing this lets us quantify our uncertainty.

## Bayesian Linear Regression

- Bayesian linear regression considers various plausible explanations for how the data were generated.
- It makes predictions using all possible regression weights, weighted by their posterior probability.



- **Prior distribution:** $\mathbf{w} \sim \mathcal{N}(\mathbf{0}, \mathbf{S})$
- **Likelihood:** $t \,|\, \mathbf{x}, \mathbf{w} \sim \mathcal{N}(\mathbf{w}^\top \boldsymbol{\phi}(\mathbf{x}),\ \sigma^2)$
- Assuming fixed/known $\mathbf{S}$ and $\sigma^2$ is a big assumption. More on this later.

## Bayesian Linear Regression: Posterior

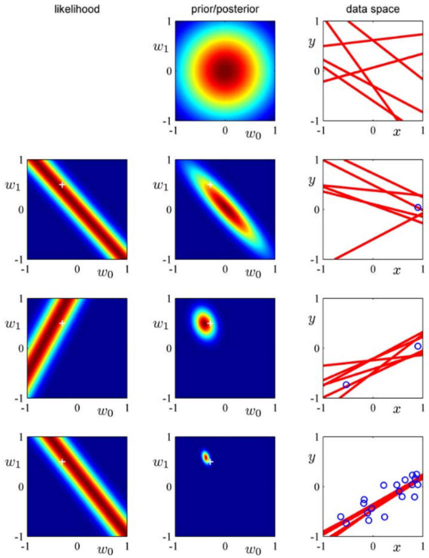
- Posterior distribution:

$$\begin{aligned}\mathbf{w} \mid \mathcal{D} &\sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma}) \\ \boldsymbol{\mu} &= \sigma^{-2}\boldsymbol{\Sigma}\boldsymbol{\Phi}^\top \mathbf{t} \\ \boldsymbol{\Sigma}^{-1} &= \sigma^{-2}\boldsymbol{\Phi}^\top\boldsymbol{\Phi} + \mathbf{S}^{-1}\end{aligned}$$

- Since a Gaussian prior leads to a Gaussian posterior, this means the Gaussian distribution is the conjugate prior for linear regression!
- Compare $\boldsymbol{\mu}$ the closed-form solution for linear regression:

$$\mathbf{w} = (\boldsymbol{\Phi}^\top\boldsymbol{\Phi} + \lambda\mathbf{I})^{-1}\boldsymbol{\Phi}^\top\mathbf{t}$$

## Bayesian Linear Regression

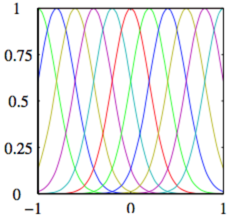


— Bishop, Pattern Recognition and Machine Learning

## Bayesian Linear Regression

- Example with radial basis function (RBF) features

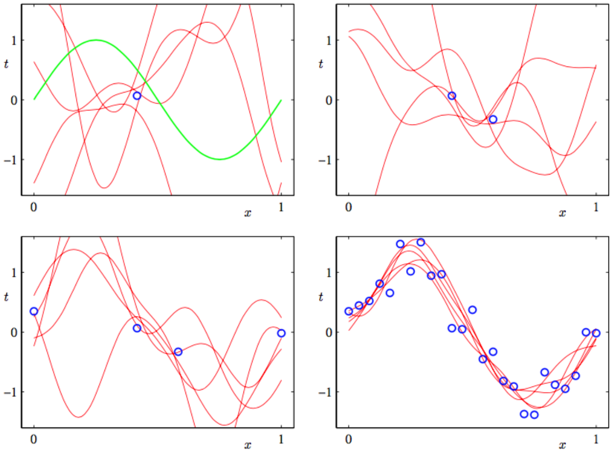
$$\phi_j(x) = \exp\left(-\frac{(x-\mu_j)^2}{2s^2}\right)$$



— Bishop, Pattern Recognition and Machine Learning

## Bayesian Linear Regression

Functions sampled from the posterior:



— Bishop, Pattern Recognition and Machine Learning

## Bayesian Linear Regression

- Posterior predictive distribution:

$$p(t \,|\, \mathbf{x}, \mathcal{D}) = \int \underbrace{p(t \,|\, \mathbf{x}, \mathbf{w})}_{\mathcal{N}(t\,;\,\mathbf{w}^\top \boldsymbol{\phi}(\mathbf{x}), \sigma)} \; \underbrace{p(\mathbf{w} \,|\, \mathcal{D})}_{\mathcal{N}(\mathbf{w}\,;\,\boldsymbol{\mu}, \boldsymbol{\Sigma})} \, \mathrm{d}\mathbf{w}$$

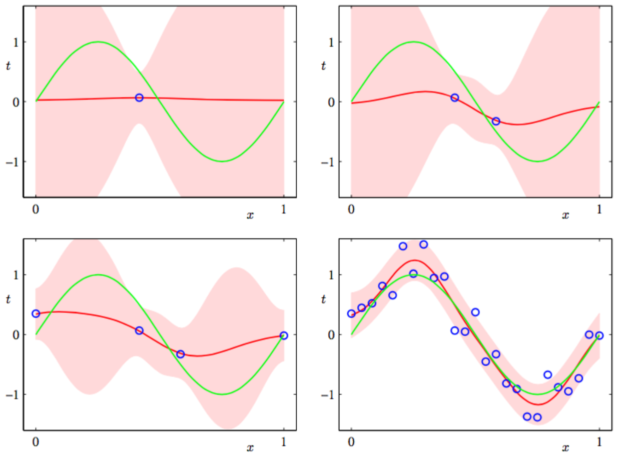
- Another interpretation: $t = \mathbf{w}^\top \boldsymbol{\phi}(\mathbf{x}) + \varepsilon$, where $\varepsilon \sim \mathcal{N}(0, \sigma)$ is independent of $\mathbf{w}$.
- By the linear combination rules for Gaussian random variables, $t$ is a Gaussian distribution with parameters

$$\mu_{\text{pred}} = \boldsymbol{\mu}^\top \boldsymbol{\phi}(\mathbf{x})$$
$$\sigma^2_{\text{pred}} = \boldsymbol{\phi}(\mathbf{x})^\top \boldsymbol{\Sigma} \boldsymbol{\phi}(\mathbf{x}) + \sigma^2$$

- Hence, the posterior predictive distribution is $\mathcal{N}(t\,;\, \mu_{\text{pred}}, \sigma^2_{\text{pred}})$.

## Bayesian Linear Regression

Here we visualize confidence intervals based on the posterior predictive mean and variance at each point:

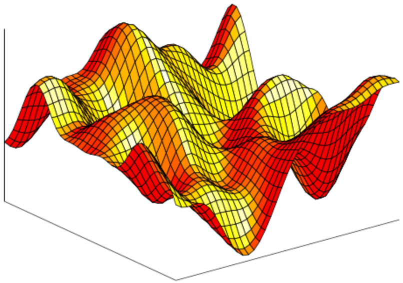


— Bishop, Pattern Recognition and Machine Learning

## Gaussian Processes

## Towards Gaussian Processes

- Gaussian Processes are distributions over functions.
- They're actually a simpler and more intuitive way to think about regression, once you're used to them.



— GPML

## Towards Gaussian Processes

- A Bayesian linear regression model defines a distribution over functions:
$$f(\mathbf{x}) = \mathbf{w}^\top \boldsymbol{\phi}(\mathbf{x})$$
Here, $\mathbf{w}$ is sampled from the prior $\mathcal{N}(\boldsymbol{\mu}_\mathbf{w}, \boldsymbol{\Sigma}_\mathbf{w})$.
- Let $\mathbf{f} = (f_1, \ldots, f_N)$ denote the vector of function values at $(\mathbf{x}_1, \ldots, \mathbf{x}_N)$.
- By the linear transformation rules for Gaussian random variables, the distribution of $\mathbf{f}$ is a Gaussian with
$$\mathbb{E}[f_i] = \boldsymbol{\mu}_\mathbf{w}^\top \boldsymbol{\phi}(\mathbf{x})$$
$$\mathrm{Cov}(f_i, f_j) = \boldsymbol{\phi}(\mathbf{x}_i)^\top \boldsymbol{\Sigma}_\mathbf{w} \boldsymbol{\phi}(\mathbf{x}_j)$$
- In vectorized form, $\mathbf{f} \sim \mathcal{N}(\boldsymbol{\mu}_\mathbf{f}, \boldsymbol{\Sigma}_\mathbf{f})$ with
$$\boldsymbol{\mu}_\mathbf{f} = \mathbb{E}[\mathbf{f}] = \boldsymbol{\Phi} \boldsymbol{\mu}_\mathbf{w}$$
$$\boldsymbol{\Sigma}_\mathbf{f} = \mathrm{Cov}(\mathbf{f}) = \boldsymbol{\Phi} \boldsymbol{\Sigma}_\mathbf{w} \boldsymbol{\Phi}^\top$$

## Towards Gaussian Processes

- Recall that in Bayesian linear regression, we assume noisy Gaussian observations of the underlying function.

$$y_i \sim \mathcal{N}(f_i, \sigma^2) = \mathcal{N}(\mathbf{w}^\top \boldsymbol{\phi}(\mathbf{x}_i), \sigma^2).$$

- The observations $\mathbf{y}$ are jointly Gaussian, just like $\mathbf{f}$.

$$\begin{aligned} \mathbb{E}[y_i] &= \mathbb{E}[f(\mathbf{x}_i)] \\ \mathrm{Cov}(y_i, y_j) &= \begin{cases} \mathrm{Var}(f(\mathbf{x}_i)) + \sigma^2 & \text{if } i = j \\ \mathrm{Cov}(f(\mathbf{x}_i), f(\mathbf{x}_j)) & \text{if } i \neq j \end{cases} \end{aligned}$$

- In vectorized form, $\mathbf{y} \sim \mathcal{N}(\boldsymbol{\mu}_\mathbf{y}, \boldsymbol{\Sigma}_\mathbf{y})$, with

$$\begin{aligned} \boldsymbol{\mu}_\mathbf{y} &= \boldsymbol{\mu}_\mathbf{f} \\ \boldsymbol{\Sigma}_\mathbf{y} &= \boldsymbol{\Sigma}_\mathbf{f} + \sigma^2 \mathbf{I} \end{aligned}$$

## Towards Gaussian Processes

- Bayesian linear regression is just computing the conditional distribution in a multivariate Gaussian!
- Let $\mathbf{y}$ and $\mathbf{y}'$ denote the observables at the training and test data.
- They are jointly Gaussian:

$$\begin{pmatrix} \mathbf{y} \\ \mathbf{y}' \end{pmatrix} \sim \mathcal{N}\left( \begin{pmatrix} \boldsymbol{\mu}_{\mathbf{y}} \\ \boldsymbol{\mu}_{\mathbf{y}'} \end{pmatrix}, \begin{pmatrix} \boldsymbol{\Sigma}_{\mathbf{y}\mathbf{y}} & \boldsymbol{\Sigma}_{\mathbf{y}\mathbf{y}'} \\ \boldsymbol{\Sigma}_{\mathbf{y}'\mathbf{y}} & \boldsymbol{\Sigma}_{\mathbf{y}'\mathbf{y}'} \end{pmatrix} \right).$$

- The predictive distribution is a special case of the conditioning formula for a multivariate Gaussian:

$$\begin{aligned} \mathbf{y}' \,|\, \mathbf{y} &\sim \mathcal{N}(\boldsymbol{\mu}_{\mathbf{y}'|\mathbf{y}}, \boldsymbol{\Sigma}_{\mathbf{y}'|\mathbf{y}}) \\ \boldsymbol{\mu}_{\mathbf{y}'|\mathbf{y}} &= \boldsymbol{\mu}_{\mathbf{y}'} + \boldsymbol{\Sigma}_{\mathbf{y}'\mathbf{y}} \boldsymbol{\Sigma}_{\mathbf{y}\mathbf{y}}^{-1} (\mathbf{y} - \boldsymbol{\mu}_{\mathbf{y}}) \\ \boldsymbol{\Sigma}_{\mathbf{y}'|\mathbf{y}} &= \boldsymbol{\Sigma}_{\mathbf{y}'\mathbf{y}'} - \boldsymbol{\Sigma}_{\mathbf{y}'\mathbf{y}} \boldsymbol{\Sigma}_{\mathbf{y}\mathbf{y}}^{-1} \boldsymbol{\Sigma}_{\mathbf{y}\mathbf{y}'} \end{aligned}$$

- We're implicitly marginalizing out $\mathbf{w}$!

## Towards Gaussian Processes

- To summarize:

$$
\begin{aligned}
\boldsymbol{\mu}_{\mathbf{f}} &= \boldsymbol{\Phi}\boldsymbol{\mu}_{\mathbf{w}} \\
\boldsymbol{\Sigma}_{\mathbf{f}} &= \boldsymbol{\Phi}\boldsymbol{\Sigma}_{\mathbf{w}}\boldsymbol{\Phi}^{\top} \\
\boldsymbol{\mu}_{\mathbf{y}} &= \boldsymbol{\mu}_{\mathbf{f}} \\
\boldsymbol{\Sigma}_{\mathbf{y}} &= \boldsymbol{\Sigma}_{\mathbf{f}} + \sigma^2\mathbf{I} \\
\boldsymbol{\mu}_{\mathbf{y}'|\mathbf{y}} &= \boldsymbol{\mu}_{\mathbf{y}'} + \boldsymbol{\Sigma}_{\mathbf{y}'\mathbf{y}}\boldsymbol{\Sigma}_{\mathbf{y}\mathbf{y}}^{-1}(\mathbf{y} - \boldsymbol{\mu}_{\mathbf{y}}) \\
\boldsymbol{\Sigma}_{\mathbf{y}'|\mathbf{y}} &= \boldsymbol{\Sigma}_{\mathbf{y}'\mathbf{y}'} - \boldsymbol{\Sigma}_{\mathbf{y}'\mathbf{y}}\boldsymbol{\Sigma}_{\mathbf{y}\mathbf{y}}^{-1}\boldsymbol{\Sigma}_{\mathbf{y}\mathbf{y}'} \\
p(\mathbf{y} \mid \mathbf{X}) &= \mathcal{N}(\mathbf{y}; \boldsymbol{\mu}_{\mathbf{y}}, \boldsymbol{\Sigma}_{\mathbf{y}})
\end{aligned}
$$

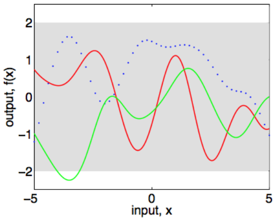
- After defining $\boldsymbol{\mu}_{\mathbf{f}}$ and $\boldsymbol{\Sigma}_{\mathbf{f}}$, we can forget about $\mathbf{w}$!
- What if we just let $\boldsymbol{\mu}_{\mathbf{f}}$ and $\boldsymbol{\Sigma}_{\mathbf{f}}$ be anything?

## Gaussian Processes

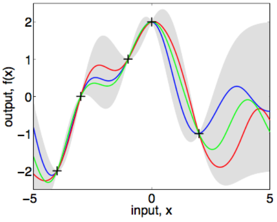
- When I say let $\boldsymbol{\mu}_{\mathbf{f}}$ and $\boldsymbol{\Sigma}_{\mathbf{f}}$ be anything, I mean let them have an arbitrary functional dependence on the inputs.
- We need to specify
  - a mean function $\mathbb{E}[f(\mathbf{x}_i)] = \mu(\mathbf{x}_i)$
  - a covariance function called a kernel function: $\mathrm{Cov}(f(\mathbf{x}_i), f(\mathbf{x}_j)) = k(\mathbf{x}_i, \mathbf{x}_j)$
- 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 $\mathbf{K}_{\mathbf{X}}$ be positive semidefinite for *any* $\mathbf{X}$. Other than that, $\mu$ and $k$ can be arbitrary.

## Gaussian Processes

- We've just defined a distribution over *function values* at an arbitrary finite set of points.
- This can be extended to a distribution over *functions* using a kind of black magic called the Kolmogorov Extension Theorem. This distribution over functions is called a Gaussian process (GP).
- We only ever need to compute with distributions over function values. The formulas from a few slides ago are all you need to do regression with GPs.
- But distributions over functions are conceptually cleaner.



(a), prior



(b), posterior

- How do you think these plots were generated?

## 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 (to be proved in Homework 7):
    - 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.

## Kernel Trick: Computational Cost

- The kernel trick lets us implicitly use very high-dimensional (even infinite-dimensional) feature spaces, but this comes at a cost.
- **Bayesian linear regression:**

$$\boldsymbol{\mu} = \sigma^{-2}\boldsymbol{\Sigma}\boldsymbol{\Phi}^\top \mathbf{t}$$
$$\boldsymbol{\Sigma}^{-1} = \sigma^{-2}\boldsymbol{\Phi}^\top\boldsymbol{\Phi} + \mathbf{S}^{-1}$$

  - Need to compute the inverse of a $D \times D$ matrix, which is an $\mathcal{O}(D^3)$ operation. ($D$ is the number of features.)
- **GP regression:**

$$\boldsymbol{\mu}_{\mathbf{y}'|\mathbf{y}} = \boldsymbol{\mu}_{\mathbf{y}'} + \boldsymbol{\Sigma}_{\mathbf{y}'\mathbf{y}}\boldsymbol{\Sigma}_{\mathbf{y}\mathbf{y}}^{-1}(\mathbf{y} - \boldsymbol{\mu}_{\mathbf{y}})$$
$$\boldsymbol{\Sigma}_{\mathbf{y}'|\mathbf{y}} = \boldsymbol{\Sigma}_{\mathbf{y}'\mathbf{y}'} - \boldsymbol{\Sigma}_{\mathbf{y}'\mathbf{y}}\boldsymbol{\Sigma}_{\mathbf{y}\mathbf{y}}^{-1}\boldsymbol{\Sigma}_{\mathbf{y}\mathbf{y}'}$$

  - Need to invert an $N \times N$ matrix! ($N$ is the number of training examples.)

## Kernel Trick: Computational Cost

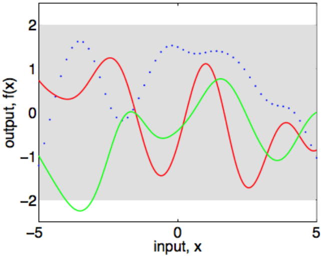
- This $\mathcal{O}(N^3)$ cost is typical of kernel methods. Most exact kernel methods don't scale to more than a few thousand data points.
- Kernel SVMs can be scaled further, since you can show you only need to consider the kernel over the support vectors, not the entire training set. (This is part of why they were so useful.)
- Scaling GP methods to large datasets is an active (and fascinating) research area.

## 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:

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

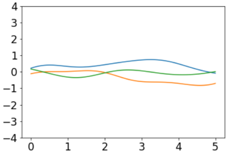
- More accurately, this is a kernel family with hyperparameters $\sigma$ and $\ell$.
- It gives a distribution over smooth functions:

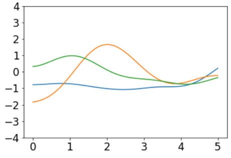


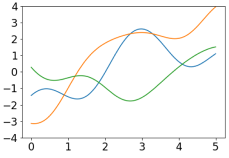
## GP Kernels

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

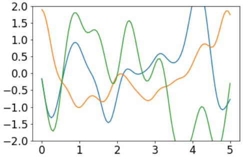
- The hyperparameters determine key properties of the function.
- Varying the output variance $\sigma^2$:

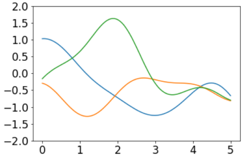

$\sigma^2 = 0.3$

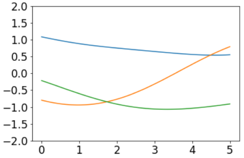

$\sigma^2 = 1$


$\sigma^2 = 3$

- Varying the lengthscale $\ell$:

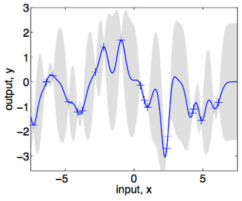

$\ell = 0.3$


$\ell = 1$

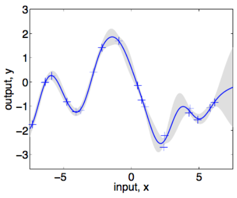

$\ell = 3$

## GP Kernels

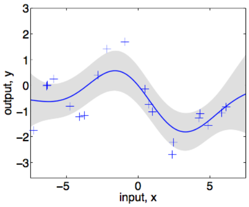
- The choice of hyperparameters heavily influences the predictions:



(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 $\rightarrow$ 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)

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

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

**Function Space**
(e.g. GPs)

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

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

- $\mathbf{f}$ 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 $p(t \,|\, \mathbf{x}, \mathbf{f})$ and $p(\mathcal{D} \,|\, \mathbf{f})$ depend only on the observation model (which we'll take as fixed), the important object to study is the prior $p(\mathbf{f})$

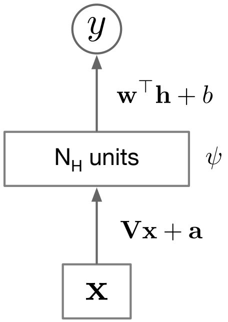
## Wide BNNs → GPs

- Vanilla BNN model definition

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

- Priors (all independent)

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


$y$
$\mathbf{w}^\top \mathbf{h} + b$
$N_H$ units
$\psi$
$\mathbf{V}\mathbf{x} + \mathbf{a}$
$\mathbf{x}$

## Wide BNNs → GPs

- Expectation of the function:

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

## Wide BNNs → GPs

- Variance of the function:

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

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

## Wide BNNs → GPs

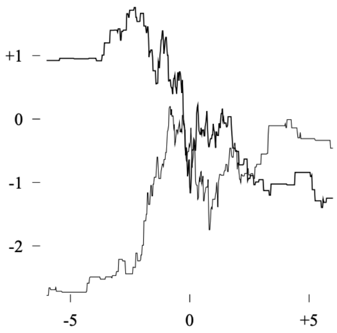
- Covariance of the function: with an analogous derivation,

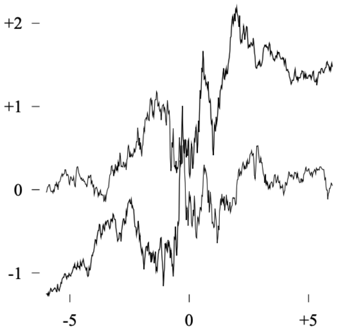
$$\begin{aligned}\mathrm{Cov}(f(\mathbf{x}), f(\mathbf{x}')) &= \omega\, \mathbb{E}[h_i(\mathbf{x}) h_i(\mathbf{x}')] + \sigma_b^2 \\ &= \omega \int \psi(\mathbf{v}^\top \mathbf{x} + a)\, \psi(\mathbf{v}^\top \mathbf{x}' + a)\, p(\mathbf{v}, a)\, \mathrm{d}\{\mathbf{v}, a\} \,+\, \sigma_b^2\end{aligned}$$

- 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 → GPs

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

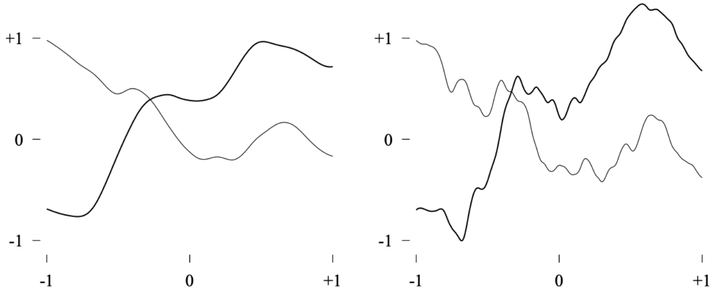




(Neal, 1993)

# Wide BNNs → GPs

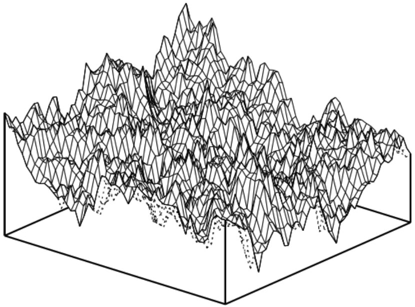
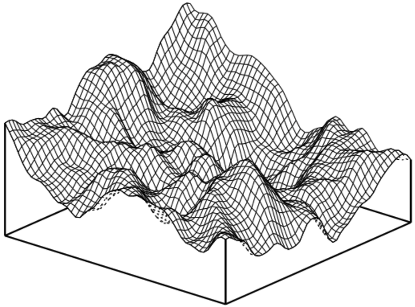
BNN with tanh activations, small $\sigma_v^2$ → smooth GP



(Neal, 1993)

## Wide BNNs → GPs

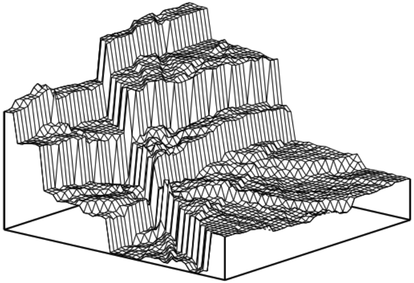
Two dimensions (Left: hard threshold, Right: tanh)

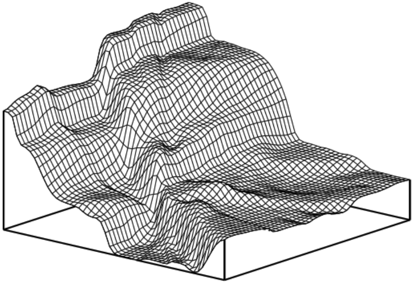



(Neal, 1993)

## 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 $\sigma_v^2$



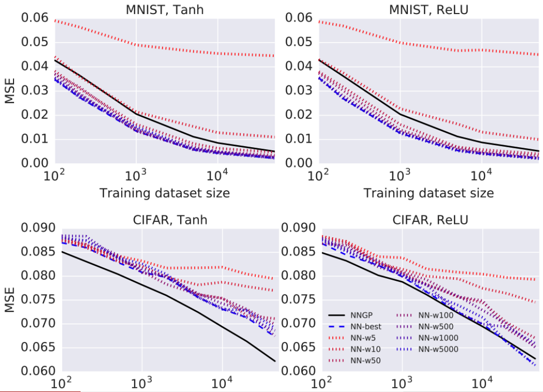


(Neal, 1993)

## 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)



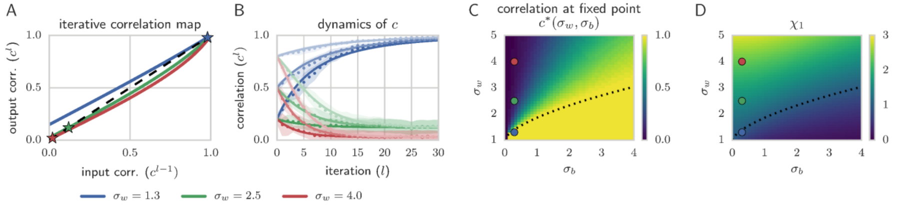
## Mean Field Approximation

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 $\mathrm{Cov}(h_i(\mathbf{x}), h_i(\mathbf{x}'))$
- Poole et al. apply this insight recursively and analyze how the covariance evolves as a function of depth

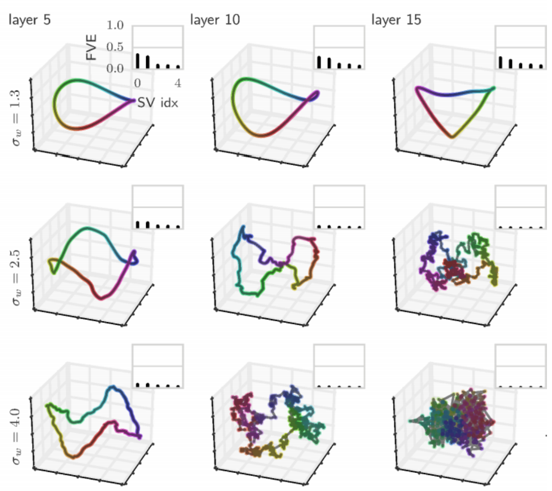
# Mean Field Approximation

- 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 $\sigma_w$ and $\sigma_b$.



(Poole et al., 2016)

# Mean Field Approximation



(Poole et al., 2016)

## 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

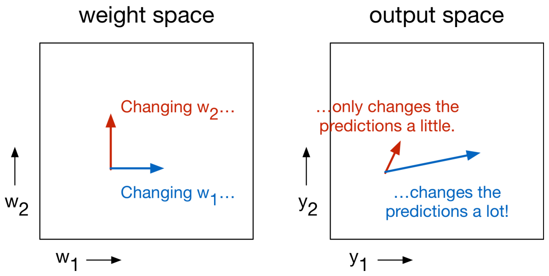
$$\mathbf{H} = \mathbf{V}\mathbf{D}^2\mathbf{V}^\top \qquad \text{(spectral decomposition of } \mathbf{H}\text{)}$$
$$\mathbf{K} = \mathbf{U}\mathbf{D}^2\mathbf{U}^\top \qquad \text{(spectral decomposition of } \mathbf{K}\text{)}$$

- **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

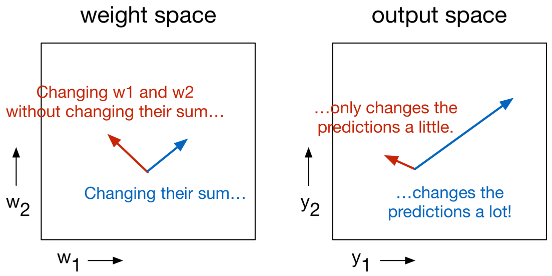
| $x_1$ | $x_2$ | $t$ |
|---|---|---|
| 114.8 | 0.00323 | 5.1 |
| 338.1 | 0.00183 | 3.2 |
| 98.8 | 0.00279 | 4.1 |
| ⋮ | ⋮ | ⋮ |



## 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 |
| $\vdots$ | $\vdots$ | $\vdots$ |


weight space
Changing w1 and w2 without changing their sum…
Changing their sum…
w2
w1
output space
…only changes the predictions a little.
…changes the predictions a lot!
y2
y1

## 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!)

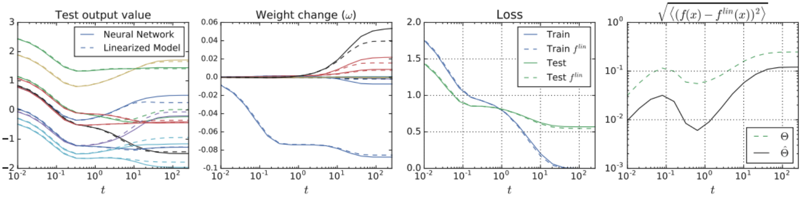
$$\frac{\mathrm{d}\bar{\mathbf{z}}}{\mathrm{d}t} = -\frac{\alpha}{N}\mathbf{K}(\mathbf{y} - \mathbf{t})$$
$$\mathbf{y}(t) = \mathbf{t} + \exp\left(-\frac{\alpha t}{N}\mathbf{K}\right)(\mathbf{y}(0) - \mathbf{t})$$

- 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

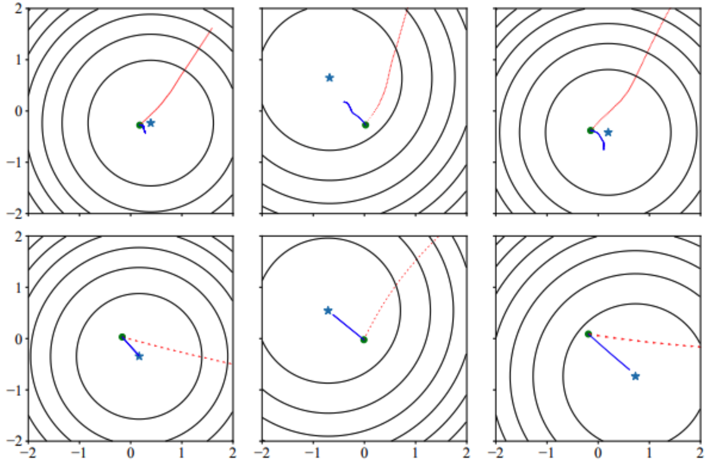
## Neural Tangent Kernel

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.


2
1
0
−1
−2
2
1
0
−1
−2
−2
−1
0
1
2
−2
−1
0
1
2
−2
−1
0
1
2