

Bayesian Deep Learning and a Probabilistic Perspective of Model Construction

Andrew Gordon Wilson

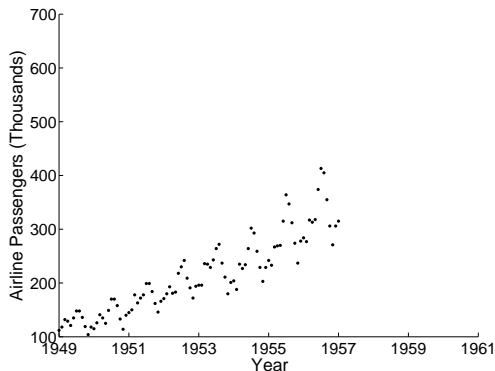
<https://cims.nyu.edu/~andrewgw>
Courant Institute of Mathematical Sciences
Center for Data Science
New York University

International Conference on Machine Learning
Tutorial
July 13, 2020

Thanks:

**Johann Brehmer, Pavel Izmailov, Rajesh Ranganath, Cristina Savin, He He
Students, Collaborators, Colleagues**

Model Selection



Which model should we choose?

(1): $f_1(x) = w_0 + w_1x$

(2): $f_2(x) = \sum_{j=0}^3 w_jx^j$

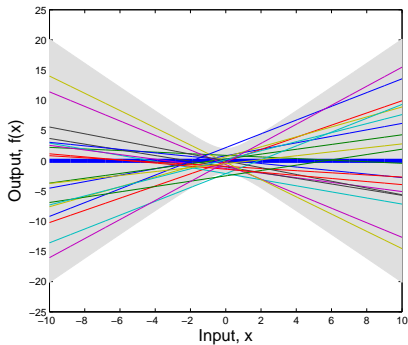
(3): $f_3(x) = \sum_{j=0}^{10^4} w_jx^j$

A Function-Space View

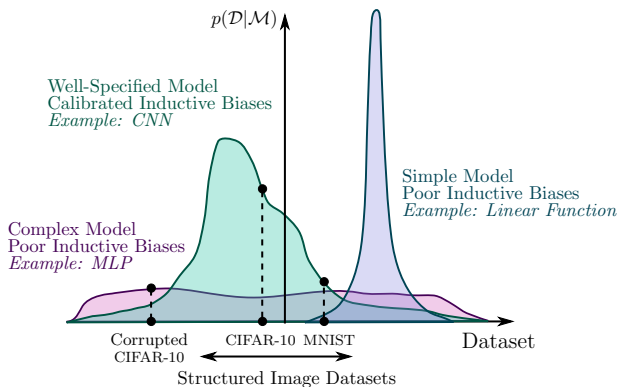
Consider the simple linear model,

$$f(x) = w_0 + w_1x, \quad (1)$$

$$w_0, w_1 \sim \mathcal{N}(0, 1). \quad (2)$$

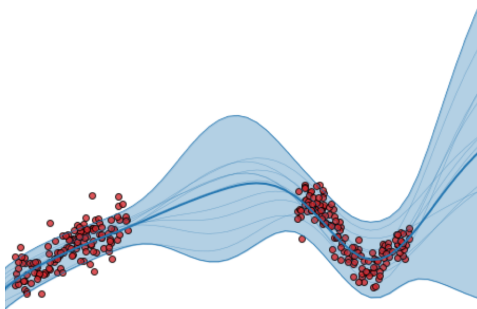


Model Construction and Generalization

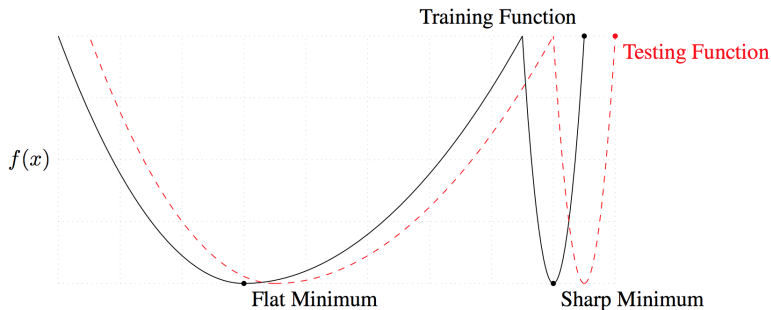


What is Bayesian learning?

- ▶ The key distinguishing property of a Bayesian approach is **marginalization** instead of optimization.
- ▶ Rather than use a single setting of parameters \mathbf{w} , use all settings weighted by their posterior probabilities in a *Bayesian model average*.



Why Bayesian Deep Learning?



Keskar et. al, ICLR 2017.

On Large-Batch Training for Deep Learning: Generalization Gap and Sharp Minima.

Bayesian integration will give very different predictions in deep learning especially!

Bayesian Deep Learning

Why?

- ▶ Neural networks represent many complementary explanations for the data.
- ▶ Better uncertainty representation (crucial for decision making).
- ▶ *Better point predictions.*
- ▶ It was the most successful approach at the end of the second wave of neural networks (Neal, 1996) [1].
- ▶ Neural nets are much less mysterious when viewed through the lens of probability theory.

Why not?

- ▶ Can be computationally intractable (but doesn't have to be).
- ▶ Can involve a lot of moving parts (but doesn't have to).

In the last year, Bayesian neural networks have been providing *better* practical results than classical training, without significant overhead. [e.g., 2, 3, 4, 5]

[1] *Bayesian Learning for Neural Networks*. Neal, R. Springer, 1996.

[2] *Subspace Inference for Bayesian Deep Learning*. Izmailov et. al, UAI 2019.

[3] *A Systematic Comparison of Bayesian Deep Learning Robustness in Diabetic Retinopathy Tasks*. Filos et. al, 2019.

[4] *Efficient and Scalable Bayesian Neural Nets with Rank-1 Factors*. Dusenberry et. al, 2020.

[5] *Bayesian Deep Learning and a Probabilistic Perspective of Generalization*. Wilson & Izmailov, 2020.

► Part 1: Introduction to Bayesian modelling

- *Foundations, overview, importance of model averaging in deep learning, epistemic uncertainty, examples*

► Part 2: The function-space view

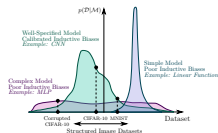
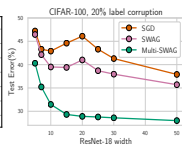
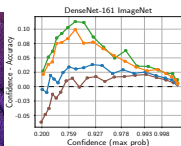
- *Gaussian processes, infinite neural networks, training a neural network is kernel learning, Bayesian non-parametric deep learning*

► Part 3: Practical methods for Bayesian deep learning

- *Loss surfaces, SWAG, Subspace Inference, K-FAC Laplace, MC Dropout*

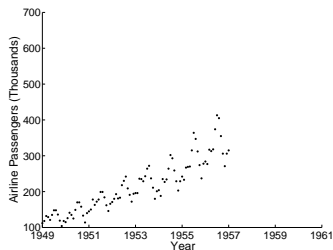
► Part 4: Bayesian model construction and generalization

- *Deep ensembles, MultiSWAG, tempering, prior-specification, re-thinking generalization, double descent, width-depth trade-offs*



- ▶ This tutorial is based on my own biased experiences and expertise. A decent portion is based on my own work.
- ▶ It is not meant to be a review of all Bayesian deep learning.
- ▶ Some important work will not be covered (but if you feel I should have included something, please send me an e-mail and I will try to include it next time).

Statistics from Scratch



Basic Regression Problem

- ▶ Training set of n targets (observations) $\mathbf{y} = (y(x_1), \dots, y(x_n))^T$.
- ▶ Observations evaluated at inputs $X = (x_1, \dots, x_n)^T$.
- ▶ Want to predict the value of $y(x_*)$ at a test input x_* .

For example: Given airline passenger numbers \mathbf{y} measured at times X , what will be the number of passengers when $x_* = 1961$?

Just knowing high school math, what might you try?

Guess the parametric form of a function that could fit the data

- ▶ $f(x, \mathbf{w}) = \mathbf{w}^T x$ [Linear function of \mathbf{w} and x]
- ▶ $f(x, \mathbf{w}) = \mathbf{w}^T \phi(x)$ [Linear function of \mathbf{w}] (Linear Basis Function Model)
- ▶ $f(x, \mathbf{w}) = g(\mathbf{w}^T \phi(x))$ [Non-linear in x and \mathbf{w}] (E.g., Neural Network)

$\phi(x)$ is a vector of basis functions. For example, if $\phi(x) = (1, x, x^2)$ and $x \in \mathbb{R}^1$ then $f(x, \mathbf{w}) = w_0 + w_1x + w_2x^2$ is a quadratic function.

Choose an error measure $E(\mathbf{w})$, minimize with respect to \mathbf{w}

- ▶ $E(\mathbf{w}) = \sum_{i=1}^n [f(x_i, \mathbf{w}) - y(x_i)]^2$

Statistics from Scratch

A probabilistic approach

We could explicitly account for noise in our model.

- ▶ $y(x) = f(x, \mathbf{w}) + \epsilon(x)$, where $\epsilon(x)$ is a noise function.

One commonly takes $\epsilon(x) = \mathcal{N}(0, \sigma^2)$ for i.i.d. additive Gaussian noise, in which case

$$p(y(x)|x, \mathbf{w}, \sigma^2) = \mathcal{N}(y(x); f(x, \mathbf{w}), \sigma^2) \quad \text{Observation Model} \quad (3)$$

$$p(\mathbf{y}|x, \mathbf{w}, \sigma^2) = \prod_{i=1}^n \mathcal{N}(y(x_i); f(x_i, \mathbf{w}), \sigma^2) \quad \text{Likelihood} \quad (4)$$

- ▶ Maximize the likelihood of the data $p(\mathbf{y}|x, \mathbf{w}, \sigma^2)$ with respect to σ^2, \mathbf{w} .

For a Gaussian noise model, this approach will make the same predictions as using a squared loss error function:

$$\log p(\mathbf{y}|X, \mathbf{w}, \sigma^2) \propto -\frac{1}{2\sigma^2} \sum_{i=1}^n [f(x_i, \mathbf{w}) - y(x_i)]^2 \quad (5)$$

Statistics from Scratch

- ▶ The probabilistic approach helps us interpret the error measure in a deterministic approach, and gives us a sense of the noise level σ^2 .
- ▶ Both approaches are prone to *over-fitting* for flexible $f(x, \mathbf{w})$: low error on the training data, high error on the test set.

Regularization

- ▶ Use a penalized log likelihood (or error function), such as

$$E(\mathbf{w}) = \underbrace{-\frac{1}{2\sigma^2} \sum_{i=1}^n (f(x_i, \mathbf{w}) - y(x_i))^2}_{\text{model fit}} \underbrace{-\lambda \mathbf{w}^T \mathbf{w}}_{\text{complexity penalty}}. \quad (6)$$

- ▶ **But how should we define and penalize complexity?**
- ▶ Can set λ using *cross-validation*.
- ▶ Same as maximizing a posterior $\log p(\mathbf{w}|\mathbf{y}, X) = \log p(\mathbf{y}|\mathbf{w}, X) + \log p(\mathbf{w}) + c$ with a Gaussian prior $p(\mathbf{w})$. **But this is not Bayesian!**

Bayesian Inference

Bayes' Rule

$$p(a|b) = p(b|a)p(a)/p(b), \quad p(a|b) \propto p(b|a)p(a). \quad (7)$$

$$\text{posterior} = \frac{\text{likelihood} \times \text{prior}}{\text{marginal likelihood}}, \quad p(\mathbf{w}|\mathbf{y}, X, \sigma^2) = \frac{p(\mathbf{y}|X, \mathbf{w}, \sigma^2)p(\mathbf{w})}{p(\mathbf{y}|X, \sigma^2)}. \quad (8)$$

Sum Rule

$$p(x) = \sum_y p(x, y) \quad (9)$$

Product Rule

$$p(x, y) = p(x|y)p(y) = p(y|x)p(x) \quad (10)$$

Bayesian Predictive Distribution

Sum rule: $p(x) = \sum_x p(x, y)$. **Product rule:** $p(x, y) = p(x|y)p(y) = p(y|x)p(x)$.

$$p(y|x_*, \mathbf{y}, X) = \int p(y|x_*, \mathbf{w})p(\mathbf{w}|\mathbf{y}, X)d\mathbf{w}. \quad (11)$$

- ▶ Think of each setting of \mathbf{w} as a different model. Eq. (32) is a *Bayesian model average*, an average of infinitely many models weighted by their posterior probabilities.
- ▶ Represents *epistemic uncertainty* over which $f(x, w)$ fits the data.
- ▶ Automatically calibrated complexity even with highly flexible models.
- ▶ Can view classical training as using an approximate posterior $q(\mathbf{w}|\mathbf{y}, X) = \delta(w = w_{\text{MAP}})$.

Bayesian Model Averaging is Not Model Combination

$$p(y|\mathcal{D}) = \sum_h p(y|h)p(h|\mathcal{D})$$

- ▶ The weights $p(h|\mathcal{D})$ only represent a statistical inability to distinguish between hypotheses.
- ▶ Assumes that combination models are not in the hypothesis space.
- ▶ In the limit of infinite data, $p(h|\mathcal{D})$ will collapse onto a point mass.

Example: Biased Coin

Suppose we flip a biased coin with probability λ of landing tails.

1. What is the likelihood of a set of data

$$\mathcal{D} = \{y_1, y_2, \dots, y_n\}?$$

2. What is the maximum likelihood solution for λ ?
3. Suppose the first flip is tails. What is the probability that the next flip will be a tails, using maximum likelihood?

Example: Biased Coin

Likelihood of the data:

$$p(\{y_i\}_{i=1}^n) = \prod_{i=1}^n \lambda^{y_i} (1 - \lambda)^{1-y_i} \quad (12)$$

where $y_i = 1$ if y_i is tails, and $y_i = 0$ if y_i is heads.

Likelihood of getting m tails is

$$p(\mathcal{D}|m, \lambda) = \binom{n}{m} \lambda^m (1 - \lambda)^{n-m} \quad (13)$$

Example: Biased Coin

Likelihood of getting m tails is

$$p(\mathcal{D}|m, \lambda) = \binom{n}{m} \lambda^m (1 - \lambda)^{n-m} \quad (14)$$

The maximum likelihood solution is

$$\hat{\lambda}_{\text{ML}} = \operatorname{argmax}_{\lambda} p(\mathcal{D}|m, \lambda) = \frac{m}{n} \quad (15)$$

Example: Biased Coin

Likelihood of getting m tails is

$$p(\mathcal{D}|m, \lambda) = \binom{n}{m} \lambda^m (1 - \lambda)^{n-m} \quad (16)$$

The maximum likelihood solution is

$$\hat{\lambda}_{\text{ML}} = \operatorname{argmax}_{\lambda} p(\mathcal{D}|m, \lambda) = \frac{m}{n} \quad (17)$$

Do you believe this solution? Why or why not?

Example: Biased Coin

Likelihood of getting m tails is

$$p(\mathcal{D}|m, \lambda) = \binom{n}{m} \lambda^m (1 - \lambda)^{n-m} \quad (18)$$

Example: Biased Coin

Likelihood of getting m tails is

$$p(\mathcal{D}|m, \lambda) = \binom{n}{m} \lambda^m (1 - \lambda)^{n-m} \quad (19)$$

If we choose a prior $p(\lambda) \propto \lambda^\alpha (1 - \lambda)^\beta$ then the posterior will have the same functional form as the prior.

Example: Biased Coin

Likelihood of getting m tails is

$$p(\mathcal{D}|m, \lambda) = \binom{n}{m} \lambda^m (1 - \lambda)^{n-m} \quad (20)$$

If we choose a prior $p(\lambda) \propto \lambda^\alpha (1 - \lambda)^\beta$ then the posterior will have the same functional form as the prior.

We can choose the beta distribution:

$$\text{Beta}(\lambda|a, b) = \frac{\Gamma(a+b)}{\Gamma(a)\Gamma(b)} \lambda^{a-1} (1 - \lambda)^{b-1} \quad (21)$$

The Gamma functions ensure that the distribution is normalized:

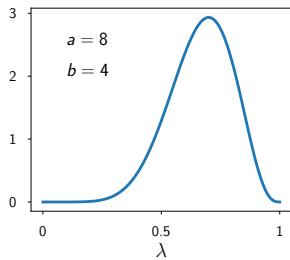
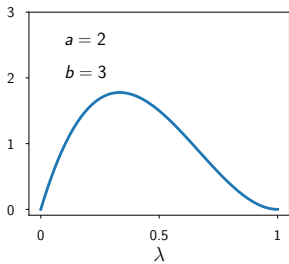
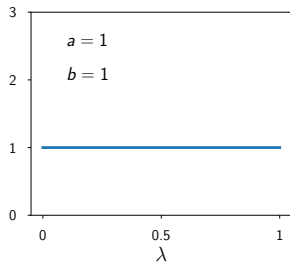
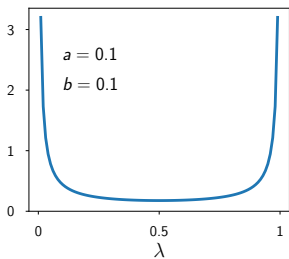
$$\int \text{Beta}(\lambda|a, b) d\lambda = 1 \quad (22)$$

Moments:

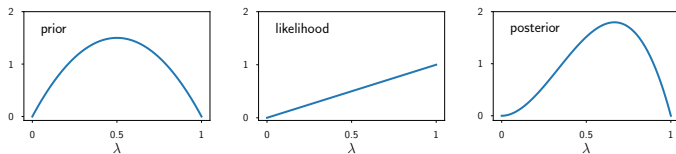
$$\mathbb{E}[\lambda] = \frac{a}{a+b} \quad (23)$$

$$\text{var}[\lambda] = \frac{ab}{(a+b)^2(a+b-1)}. \quad (24)$$

Beta Distribution



Example: Biased Coin



Applying Bayes theorem, we find:

$$p(\lambda|\mathcal{D}) \propto p(\mathcal{D}|\lambda)p(\lambda) \quad (25)$$

$$= \text{Beta}(\lambda; m + a, n - m + b) \quad (26)$$

We can view a and b as pseudo-observations!

$$\mathbb{E}[\lambda|\mathcal{D}] = \frac{m + a}{n + a + b} \quad (27)$$

1. What is the probability that the next flip is tails?
2. What happens in the limits of a, b ?
3. What happens in the limit of infinite data?

Example: Biased Coin

Applying Bayes theorem, we find:

$$p(\lambda|\mathcal{D}) \propto p(\mathcal{D}|\lambda)p(\lambda) \quad (28)$$

$$= \text{Beta}(\lambda; m + a, n - m + b) \quad (29)$$

We can view a and b as pseudo-observations!

$$\mathbb{E}[\lambda|\mathcal{D}] = \frac{m + a}{n + a + b} \quad (30)$$

1. What is the probability that the next flip is tails?
2. What happens in the limits of a, b ?
3. What happens in the limit of infinite data?
4. **Does the MAP estimate**

$$\hat{\lambda}_{MAP} = \mathbf{argmax}_{\lambda} \log p(\lambda|\mathcal{D}) = \mathbf{argmax}_{\lambda} \log p(\mathcal{D}|\lambda) + \log p(\lambda) \quad (31)$$

with a uniform prior $p(\lambda)$ give the same answer as Bayesian marginalization to find $p(\text{tails next flip}|\mathcal{D}) = \int \lambda p(\lambda|\mathcal{D})d\lambda = \mathbb{E}[\lambda|\mathcal{D}]$ with a uniform prior $p(\lambda)$?

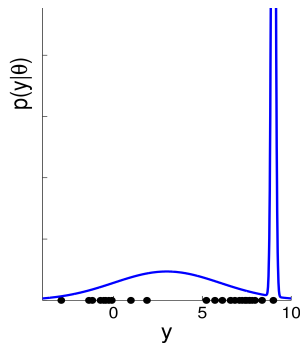
Example: Density Estimation

- ▶ Observations y_1, \dots, y_n from unknown density $p(y)$.
- ▶ Specify an observation model. For example, we can let the points be drawn from a mixture of Gaussians:

$$p(y|\theta) = w_1 \mathcal{N}(y|\mu_1, \sigma_1^2) + w_2 \mathcal{N}(y|\mu_2, \sigma_2^2),$$
$$\theta = \{w_1, w_2, \mu_1, \mu_2, \sigma_1, \sigma_2\}.$$

- ▶ Likelihood $p(\mathbf{y}|\theta) = \prod_{i=1}^n p(y_i|\theta)$.

Can learn all free parameters θ using maximum likelihood...



$$\hat{\theta}_{\text{ML}} = \operatorname{argmax}_{\theta} \prod_{i=1}^n \frac{w_1}{\sqrt{2\pi\sigma_1^2}} \exp\left(-\frac{1}{2\sigma_1^2}(y_i - \mu_1)^2\right) + \frac{w_2}{\sqrt{2\pi\sigma_2^2}} \exp\left(-\frac{1}{2\sigma_2^2}(y_i - \mu_2)^2\right)$$

Can you look at this equation and see what θ achieves maximum likelihood?

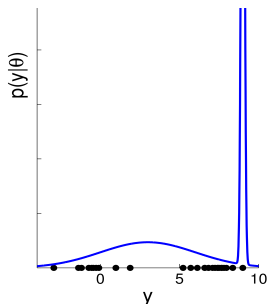
Regularization = MAP \neq Bayesian Inference

Regularization or MAP

- ▶ Find

$$\operatorname{argmax}_{\theta} \log p(\theta|\mathbf{y}) \stackrel{c}{=} \underbrace{\log p(\mathbf{y}|\theta)}_{\text{model fit}} + \underbrace{\log p(\theta)}_{\text{complexity penalty}}$$

- ▶ Choose $p(\theta)$ such that $p(\theta) \rightarrow 0$ faster than $p(\mathbf{y}|\theta) \rightarrow \infty$ as σ_1 or $\sigma_2 \rightarrow 0$.



Bayesian Inference

- ▶ Predictive Distribution: $p(y_*|\mathbf{y}) = \int p(y_*|\theta)p(\theta|\mathbf{y})d\theta$.
- ▶ Parameter Posterior: $p(\theta|\mathbf{y}) \propto p(\mathbf{y}|\theta)p(\theta)$.

Approximate Inference

Ultimately we wish to compute a Bayesian model average:

$$p(y|x_*, \mathcal{D}) = \int p(y|x_*, w)p(w|\mathcal{D})dw. \quad (32)$$

For most models, including Bayesian neural networks, this integral is *not analytic*. It is common to use a *Simple Monte Carlo* approximation:

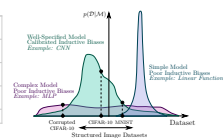
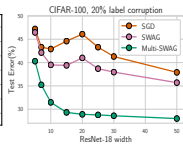
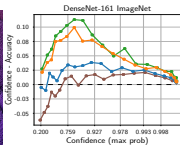
$$p(y|x_*, \mathcal{D}) \approx \frac{1}{J} \sum_j p(y|x_*, w_j), \quad w_j \sim q(w|\mathcal{D}). \quad (33)$$

w_j are samples from an approximate posterior $q(w|\mathcal{D})$ typically found by:

1. **Deterministic Methods:** Approximate $p(w|\mathcal{D})$ with convenient $q(w|\mathcal{D}, \theta)$, often Gaussian. θ (e.g. mean of q) chosen to make q close to p . Variational methods find $\operatorname{argmin}_\theta \mathcal{KL}(q||p)$. Classical training: $q(w|\mathcal{D}) = \delta(w = w_{\text{MAP}})$.
E.g.: Laplace, Expectation Propagation, Variational, Standard Training.
 2. **MCMC:** Form a Markov chain of approximate (but asymptotically exact) samples from $p(w|\mathcal{D})$.
E.g.: Metropolis-Hastings, Hamiltonian Monte Carlo, SGLD, SGHMC.
- Later we will argue we may want to avoid the simple MC perspective.

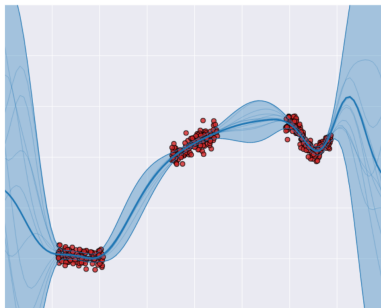
Part 2: The Function-Space View

- ▶ **Part 1: Introduction to Bayesian modelling**
 - ▶ *Importance of model averaging, epistemic uncertainty*
- ▶ **Part 2: The function-space view**
 - ▶ *Gaussian processes, infinite neural networks, Bayesian non-parametric deep learning*
- ▶ **Part 3: Practical methods for Bayesian deep learning**
 - ▶ *Loss surfaces, SWAG, Subspace Inference, K-FAC Laplace, MC Dropout*
- ▶ **Part 4: Bayesian model construction and generalization**
 - ▶ *Deep ensembles, MultiSWAG, tempering, prior-specification, re-thinking generalization, double descent, width-depth trade-offs*

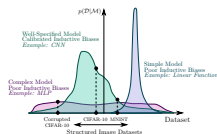
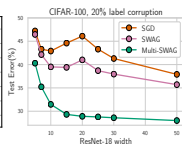
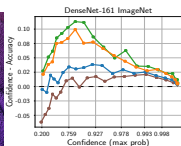


— K-FAC-Laplace — SGD — SWA-Drop — SWA-Temp — SWAG — SWAG-Diag

Part 2: The Function-Space View



- ▶ **Part 1: Introduction to Bayesian modelling**
 - ▶ *Importance of model averaging, epistemic uncertainty*
- ▶ **Part 2: The function-space view**
 - ▶ *Gaussian processes, infinite neural networks, training a neural network is learning a kernel, Bayesian non-parametric deep learning*
- ▶ **Part 3: Practical methods for Bayesian deep learning**
 - ▶ *Loss surfaces, SWAG, Subspace Inference, K-FAC Laplace, MC Dropout*
- ▶ **Part 4: Bayesian model construction and generalization**
 - ▶ *Deep ensembles, MultiSWAG, tempering, prior-specification, re-thinking generalization, double descent, width-depth trade-offs*



The Function-Space View

The parameters in isolation are completely divorced from the statistical properties of a model. Yet we focus most of our effort on learning parameters \mathbf{w} . What we really care about are how those parameters \mathbf{w} combine with a functional form $f(x, \mathbf{w})$. Ideally we want to perform inference directly in function space.

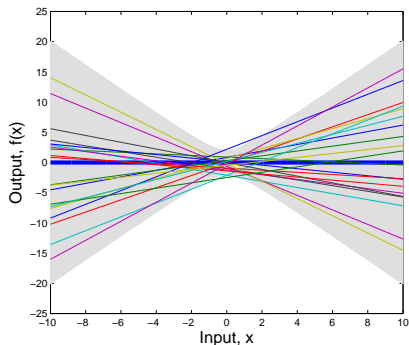
The Function-Space View

A distribution over parameters $p(\mathbf{w})$ induces a *distribution over functions* $p(f(x))$.

Consider the simple linear model,

$$f(x, w) = w_0 + w_1 x, \quad (34)$$

$$w_0, w_1 \sim \mathcal{N}(0, 1). \quad (35)$$



The Function-Space View

We can express the distribution over functions *directly*. Suppose:

$$f(x, \mathbf{w}) = \mathbf{w}^T \phi(x) \quad (36)$$

$$p(\mathbf{w}) = \mathcal{N}(0, \Sigma_w) \quad (37)$$

$\phi(x)$ is a vector of basis functions. For example, if $\phi(x) = (1, x, x^2)$ and $x \in \mathbb{R}^1$ then $f(x, \mathbf{w}) = w_0 + w_1x + w_2x^2$ is a quadratic function.

Can you derive the moments of the induced distribution over functions?

$$\mathbb{E}[f(x, \mathbf{w})] = m(x) = \mathbb{E}[\mathbf{w}^T] \phi(x) =? \quad (38)$$

$$\text{cov}(f(x_i), f(x_j)) = k(x_i, x_j) = \mathbb{E}[f(x_i)f(x_j)] - \mathbb{E}[f(x_i)]\mathbb{E}[f(x_j)] =? \quad (39)$$

The Function-Space View

We can express the distribution over functions *directly*. Suppose:

$$f(x, \mathbf{w}) = \mathbf{w}^T \boldsymbol{\phi}(x) \quad (40)$$

$$p(\mathbf{w}) = \mathcal{N}(\mathbf{0}, \Sigma_w) \quad (41)$$

Moments of Induced Distribution over Functions

$$\mathbb{E}[f(x, \mathbf{w})] = m(x) = \mathbb{E}[\mathbf{w}^T] \boldsymbol{\phi}(x) = 0 \quad (42)$$

$$\text{cov}(f(x_i), f(x_j)) = k(x_i, x_j) = \mathbb{E}[f(x_i)f(x_j)] - \mathbb{E}[f(x_i)]\mathbb{E}[f(x_j)] \quad (43)$$

$$= \boldsymbol{\phi}(x_i)^T \mathbb{E}[\mathbf{w}\mathbf{w}^T] \boldsymbol{\phi}(x_j) - 0 \quad (44)$$

$$= \boldsymbol{\phi}(x_i)^T \Sigma_w \boldsymbol{\phi}(x_j) \quad (45)$$

Are there any higher moments?

The Function-Space View

We can express the distribution over functions *directly*. Suppose:

$$f(x, \mathbf{w}) = \mathbf{w}^T \phi(x) \quad (46)$$

$$p(\mathbf{w}) = \mathcal{N}(\mathbf{0}, \Sigma_w) \quad (47)$$

Moments of Induced Distribution over Functions

$$\mathbb{E}[f(x, \mathbf{w})] = m(x) = \mathbb{E}[\mathbf{w}^T] \phi(x) = 0 \quad (48)$$

$$\text{cov}(f(x_i), f(x_j)) = k(x_i, x_j) = \mathbb{E}[f(x_i)f(x_j)] - \mathbb{E}[f(x_i)]\mathbb{E}[f(x_j)] \quad (49)$$

$$= \phi(x_i)^T \mathbb{E}[\mathbf{w}\mathbf{w}^T] \phi(x_j) - 0 \quad (50)$$

$$= \phi(x_i)^T \Sigma_w \phi(x_j) \quad (51)$$

$f(x) \sim \mathcal{GP}(m, k)$ is a *Gaussian process* with mean function $m(x)$ and covariance function (aka kernel) $k(x, x')$.

We can do inference directly over $f(x)$ instead of \mathbf{w} .

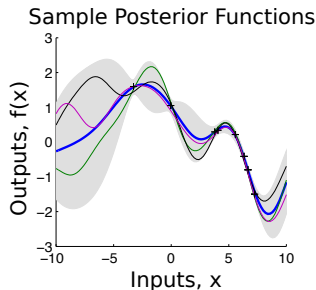
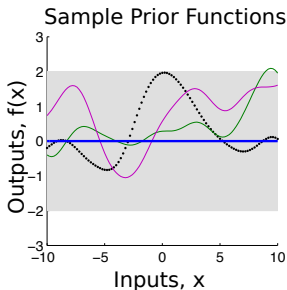
Definition

A Gaussian process (GP) is a collection of random variables, any finite number of which have a joint Gaussian distribution.

Nonparametric Regression Model

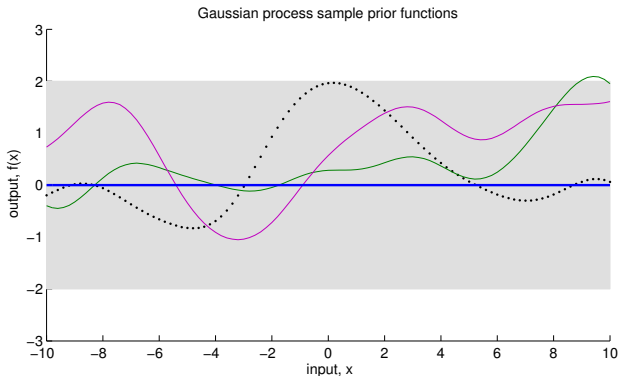
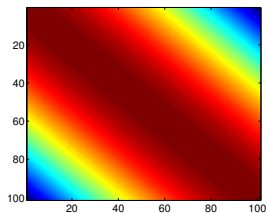
- Prior: $f(x) \sim \mathcal{GP}(m(x), k(x, x'))$, meaning $(f(x_1), \dots, f(x_N)) \sim \mathcal{N}(\boldsymbol{\mu}, K)$, with $\boldsymbol{\mu}_i = m(x_i)$ and $K_{ij} = \text{cov}(f(x_i), f(x_j)) = k(x_i, x_j)$.

$$\underbrace{p(f(x)|\mathcal{D})}_{\text{GP posterior}} \propto \underbrace{p(\mathcal{D}|f(x))}_{\text{Likelihood}} \underbrace{p(f(x))}_{\text{GP prior}}$$



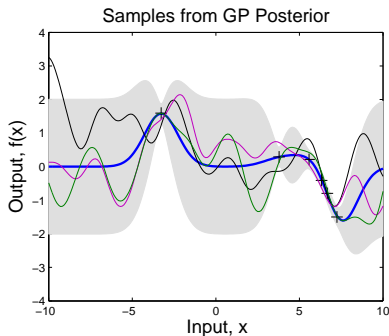
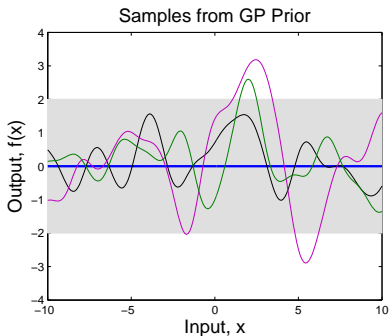
Example: RBF Kernel

$$\begin{aligned}k_{\text{RBF}}(x, x') &= \text{cov}(f(x), f(x')) \\ &= a^2 \exp\left(-\frac{\|x - x'\|^2}{2\ell^2}\right)\end{aligned}$$



Inference using an RBF kernel

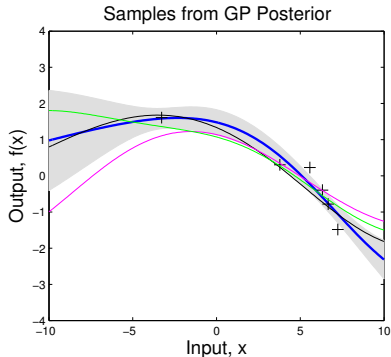
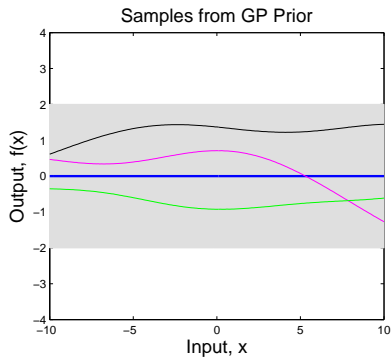
- ▶ Specify $f(x) \sim \mathcal{GP}(0, k)$.
- ▶ Choose $k_{\text{RBF}}(x, x') = a_0^2 \exp(-\frac{\|x-x'\|^2}{2\ell_0^2})$. Choose values for a_0 and ℓ_0 .
- ▶ Observe data, look at the prior and posterior over functions.



- ▶ Does something look strange about these functions?

Inference using an RBF kernel

Increase the length-scale ℓ .



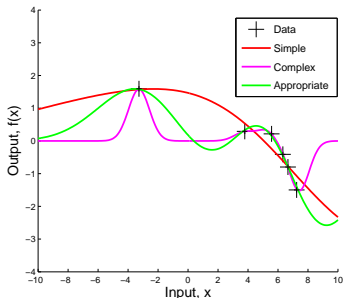
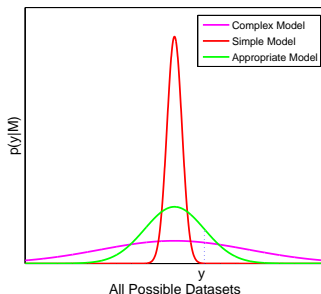
► Does something look strange about these functions?

Learning and Model Selection

$$p(\mathcal{M}_i|\mathbf{y}) = \frac{p(\mathbf{y}|\mathcal{M}_i)p(\mathcal{M}_i)}{p(\mathbf{y})} \quad (52)$$

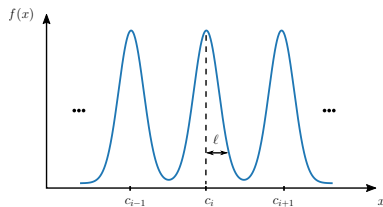
We can write the *evidence* of the model as

$$p(\mathbf{y}|\mathcal{M}_i) = \int p(\mathbf{y}|\mathbf{f}, \mathcal{M}_i)p(\mathbf{f})d\mathbf{f} \quad (53)$$



Gaussian processes for Machine Learning. Rasmussen, C.E. and Williams, C.K.I. MIT Press, 2006.
Bayesian Methods for Adaptive Models. MacKay, D.J. PhD Thesis, 1992.
Covariance Kernels for Fast Automatic Pattern Discovery and Extrapolation with Gaussian Processes.
Wilson, A.G. PhD Thesis, 2014.

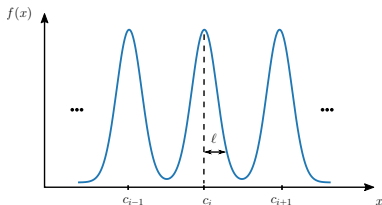
Deriving the RBF Kernel



$$f(x) = \sum_{i=1}^J w_i \phi_i(x), \quad w_i \sim \mathcal{N}\left(0, \frac{\sigma^2}{J}\right), \quad \phi_i(x) = \exp\left(-\frac{(x - c_i)^2}{2\ell^2}\right) \quad (54)$$

$$\therefore k(x, x') = \frac{\sigma^2}{J} \sum_{i=1}^J \phi_i(x) \phi_i(x') \quad (55)$$

Deriving the RBF Kernel



$$f(x) = \sum_{i=1}^J w_i \phi_i(x), \quad w_i \sim \mathcal{N}\left(0, \frac{\sigma^2}{J}\right), \quad \phi_i(x) = \exp\left(-\frac{(x - c_i)^2}{2\ell^2}\right) \quad (56)$$

$$\therefore k(x, x') = \frac{\sigma^2}{J} \sum_{i=1}^J \phi_i(x) \phi_i(x') \quad (57)$$

- ▶ Let $c_J = \log J$, $c_1 = -\log J$, and $c_{i+1} - c_i = \Delta c = 2\frac{\log J}{J}$, and $J \rightarrow \infty$, the kernel in Eq. (60) becomes a Riemann sum:

$$k(x, x') = \lim_{J \rightarrow \infty} \frac{\sigma^2}{J} \sum_{i=1}^J \phi_i(x) \phi_i(x') = \int_{c_0}^{c_\infty} \phi_c(x) \phi_c(x') dc \quad (58)$$

Deriving the RBF Kernel

$$f(x) = \sum_{i=1}^J w_i \phi_i(x), \quad w_i \sim \mathcal{N}\left(0, \frac{\sigma^2}{J}\right), \quad \phi_i(x) = \exp\left(-\frac{(x - c_i)^2}{2\ell^2}\right) \quad (59)$$

$$\therefore k(x, x') = \frac{\sigma^2}{J} \sum_{i=1}^J \phi_i(x) \phi_i(x') \quad (60)$$

- ▶ Let $c_J = \log J$, $c_1 = -\log J$, and $c_{i+1} - c_i = \Delta c = 2\frac{\log J}{J}$, and $J \rightarrow \infty$, the kernel in Eq. (60) becomes a Riemann sum:

$$k(x, x') = \lim_{J \rightarrow \infty} \frac{\sigma^2}{J} \sum_{i=1}^J \phi_i(x) \phi_i(x') = \int_{c_0}^{c_\infty} \phi_c(x) \phi_c(x') dc \quad (61)$$

- ▶ By setting $c_0 = -\infty$ and $c_\infty = \infty$, we spread the infinitely many basis functions across the whole real line, each a distance $\Delta c \rightarrow 0$ apart:

$$k(x, x') = \int_{-\infty}^{\infty} \exp\left(-\frac{(x - c)^2}{2\ell^2}\right) \exp\left(-\frac{(x' - c)^2}{2\ell^2}\right) dc \quad (62)$$

$$= \sqrt{\pi} \ell \sigma^2 \exp\left(-\frac{(x - x')^2}{2(\sqrt{2}\ell)^2}\right) \propto k_{\text{RBF}}(x, x'). \quad (63)$$

Deriving the RBF Kernel

- ▶ It is remarkable we can work with infinitely many basis functions with finite amounts of computation using the *kernel trick* – replacing inner products of basis functions with kernels.
- ▶ GPs with RBF kernels are **flexible** (universal approximators) but **simple**.
- ▶ These models have strong *inductive biases* that concentrate prior support around simple solutions, providing good generalization even on small datasets, while retaining flexibility.

A Note About The Mean Function

- ▶ Often $m(x)$ is taken as 0 for notational convenience.
- ▶ One can use any deterministic mean function without fundamentally changing the modelling procedure.
- ▶ Typically the covariance (kernel) function is the key object of interest. There are often degeneracies between the mean and covariance function, in which case typically it is preferred to do the modelling in the covariance function. For example, the kernel function shows up in the Occam factor for the marginal likelihood, but not the mean function.
- ▶ A mean function can be a good way of incorporating scientific inductive biases in a model; for example, we can concentrate the prior support around a parametric physical model, but allow a non-parametric relaxation with the Gaussian process to account for model misspecification and uncertainty.

Neural Network Kernel

- ▶ The neural network kernel (Neal, 1996) is famous for triggering research on Gaussian processes in the machine learning community.

Consider a neural network with one hidden layer:

$$f(x) = b + \sum_{i=1}^J v_i h(x; \mathbf{u}_i). \quad (64)$$

- ▶ b is a bias, v_i are the hidden to output weights, h is any bounded hidden unit transfer function, \mathbf{u}_i are the input to hidden weights, and J is the number of hidden units. Let b and v_i be independent with zero mean and variances σ_b^2 and σ_v^2/J , respectively, and let the \mathbf{u}_i have independent identical distributions.

Collecting all free parameters into the weight vector \mathbf{w} ,

$$\mathbb{E}_{\mathbf{w}}[f(x)] = 0, \quad (65)$$

$$\text{cov}[f(x), f(x')] = \mathbb{E}_{\mathbf{w}}[f(x)f(x')] = \sigma_b^2 + \frac{1}{J} \sum_{i=1}^J \sigma_v^2 \mathbb{E}_{\mathbf{u}}[h_i(x; \mathbf{u}_i)h_i(x'; \mathbf{u}_i)], \quad (66)$$

$$= \sigma_b^2 + \sigma_v^2 \mathbb{E}_{\mathbf{u}}[h(x; \mathbf{u})h(x'; \mathbf{u})]. \quad (67)$$

We can show any collection of values $f(x_1), \dots, f(x_N)$ must have a joint Gaussian distribution using the central limit theorem.

Neural Network Kernel

$$f(x) = b + \sum_{i=1}^J v_i h(x; \mathbf{u}_i). \quad (68)$$

- ▶ Let $h(x; \mathbf{u}) = \text{erf}(u_0 + \sum_{j=1}^P u_j x_j)$, where $\text{erf}(z) = \frac{2}{\sqrt{\pi}} \int_0^z e^{-t^2} dt$
- ▶ Choose $\mathbf{u} \sim \mathcal{N}(0, \Sigma)$

Then we obtain

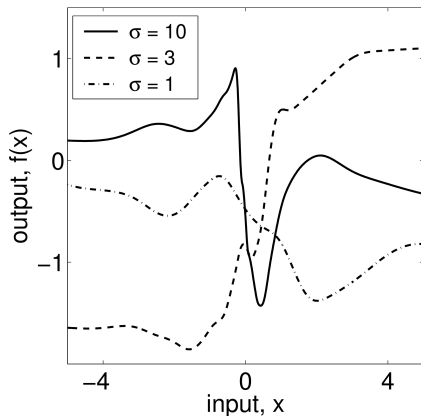
$$k_{\text{NN}}(x, x') = \frac{2}{\pi} \sin\left(\frac{2\tilde{x}^T \Sigma \tilde{x}'}{\sqrt{(1 + 2\tilde{x}^T \Sigma \tilde{x})(1 + 2\tilde{x}'^T \Sigma \tilde{x}')}}\right), \quad (69)$$

where $x \in \mathbb{R}^P$ and $\tilde{x} = (1, x^T)^T$.

Neural Network Kernel

$$k_{\text{NN}}(x, x') = \frac{2}{\pi} \sin\left(\frac{2\tilde{x}^T \Sigma \tilde{x}'}{\sqrt{(1 + 2\tilde{x}^T \Sigma \tilde{x})(1 + 2\tilde{x}'^T \Sigma \tilde{x}')}}\right) \quad (70)$$

Set $\Sigma = \text{diag}(\sigma_0, \sigma)$. Draws from a GP with a neural network kernel with varying σ :

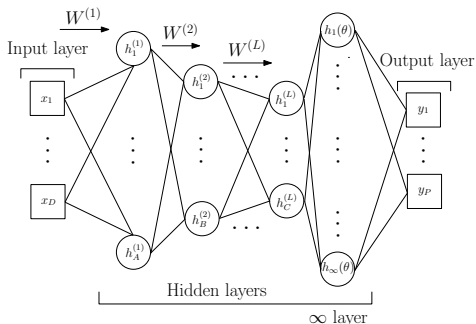


“How can Gaussian processes possibly replace neural networks? Have we thrown the baby out with the bathwater?” (MacKay, 1998)

Introduction to Gaussian processes. MacKay, D. J. In Bishop, C. M. (ed.), *Neural Networks and Machine Learning*, Chapter 11, pp. 133-165. Springer-Verlag, 1998.

Deep Kernel Learning

Deep kernel learning combines the inductive biases of deep learning architectures with the non-parametric flexibility of Gaussian processes.



Base kernel hyperparameters θ and deep network hyperparameters w are jointly trained through the marginal likelihood objective.

Deep Kernel Learning. Wilson, A.G., Hu, Z., Salakhutdinov, R., Xing, E.P. AISTATS, 2016

Face Orientation Extraction

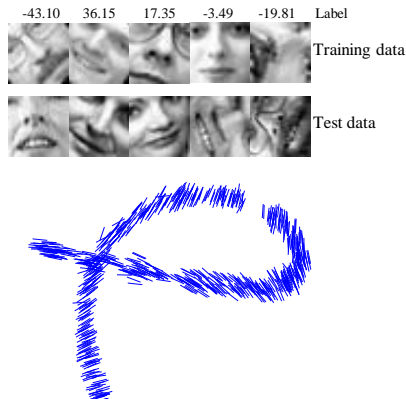


Figure: **Top:** Randomly sampled examples of the training and test data. **Bottom:** The two dimensional outputs of the convolutional network on a set of test cases. Each point is shown using a line segment that has the same orientation as the input face.

Learning Flexible Non-Euclidean Similarity Metrics

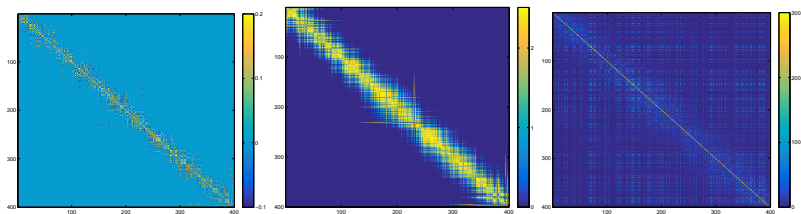


Figure: **Left:** The induced covariance matrix using DKL-SM (spectral mixture) kernel on a set of test cases, where the test samples are ordered according to the *orientations* of the input faces. **Middle:** The respective covariance matrix using DKL-RBF kernel. **Right:** The respective covariance matrix using regular RBF kernel. The models are trained with $n = 12,000$.

Step Function

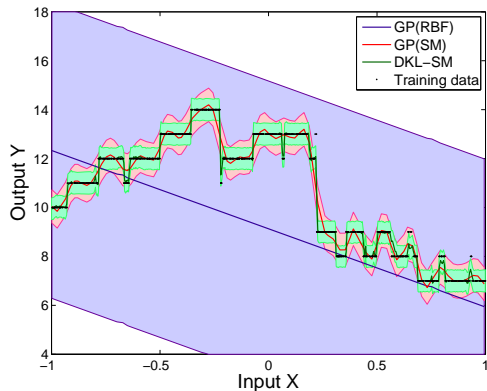
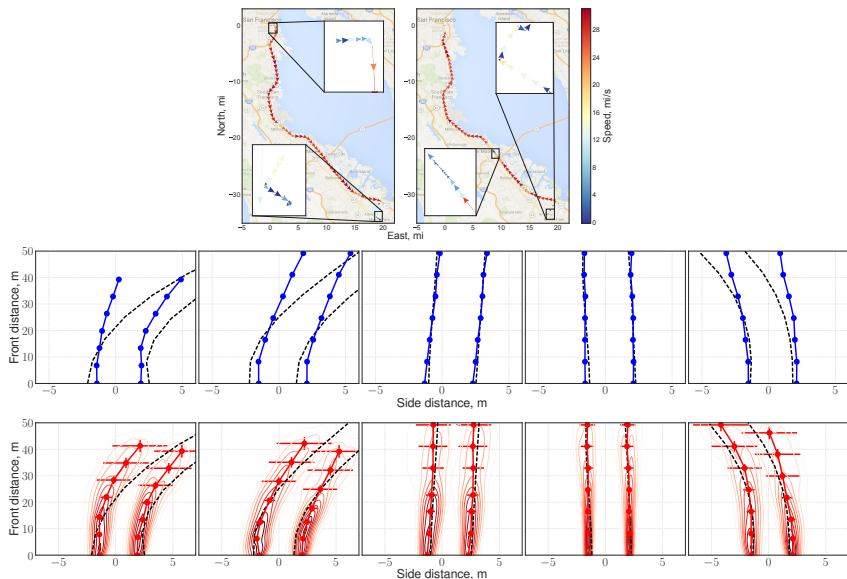


Figure: Recovering a step function. We show the predictive mean and 95% of the predictive probability mass for regular GPs with RBF and SM kernels, and DKL with SM base kernel.

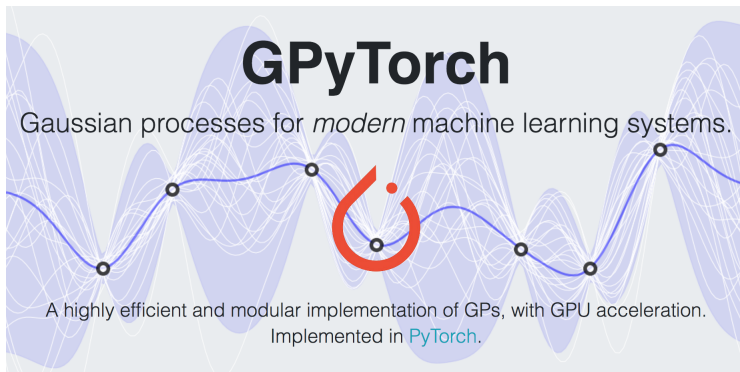
Deep Kernel Learning for Autonomous Driving



Learning scalable deep kernels with recurrent structure.
Al-Shedivat, M., Wilson, A.G., Saatchi, Y., Hu, Z., Xing, E.P. JMLR, 2017.

Scalable Gaussian Processes

- ▶ Run **exact** GPs on millions of points in minutes.
- ▶ Based on Krylov subspace methods that permit significant GPU acceleration.
- ▶ Outperforms stand-alone deep neural networks by learning **deep kernels**.
- ▶ **Implemented in the new library GPyTorch:** `gpytorch.ai`

The image shows a promotional graphic for GPyTorch. At the top, the word "GPyTorch" is written in a large, bold, black sans-serif font. Below it, the text "Gaussian processes for modern machine learning systems." is displayed in a smaller, black sans-serif font. In the center, there is a stylized logo consisting of a red circle with a white dot inside, and a blue line that forms a partial circle around the dot. Below the logo, the text "A highly efficient and modular implementation of GPs, with GPU acceleration." is written in a black sans-serif font, followed by "Implemented in PyTorch." in a smaller, blue sans-serif font. The background of the graphic is light blue with several overlapping, semi-transparent blue wave-like shapes that resemble Gaussian process realizations. The overall design is clean and modern.

GPyTorch

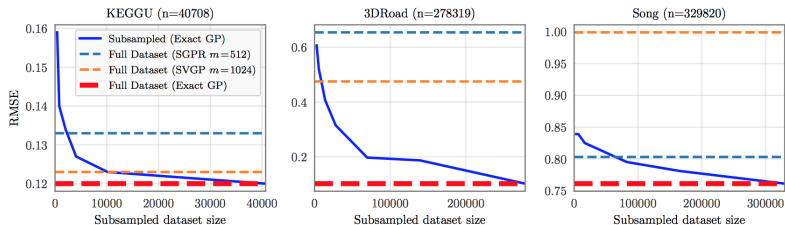
Gaussian processes for *modern* machine learning systems.

A highly efficient and modular implementation of GPs, with GPU acceleration.
Implemented in [PyTorch](#).

GPyTorch: Blackbox Matrix-Matrix Gaussian Process Inference with GPU Acceleration.
Gardner, J. R., Pleiss, G., Bindel, D., Weinberger, K. Q., Wilson, A. G. NeurIPS, 2018.

Exact Gaussian Processes on a Million Data Points

- ▶ Exact GPs previously intractable for more than about 10,000 points.
- ▶ By developing stochastic Krylov methods that exploit parallel cores in GPUs we can now run exact GPs on millions of points.
- ▶ Results show the benefit of retaining a *non-parametric* representation.



Exact Gaussian processes on a million data points.

Wang, K.A., Pleiss, G., Gardner, J., Tyree, S., Weinberger, K., Wilson, A.G. NeurIPS, 2019.

Additional GP Libraries

- ▶ *GPFlow*, in Tensorflow.

The logo for GPFlow, featuring the text "GPflow" in a green, stylized font. The letters "G" and "P" are larger and more prominent, with "flow" in a smaller font size.

- ▶ *GPpy*, in Python.

The logo for GPpy, consisting of the letters "GPpy" in a white, serif font, set against a solid black rectangular background.

- ▶ *BoTorch*, for Bayesian optimization, in PyTorch.

The logo for BoTorch, featuring a stylized flame icon in shades of orange, red, and purple above the text "BoTorch" in a white, sans-serif font, all on a dark blue background.

Different algorithmic foundations and use-cases.

Additional Work on GP+NN

- ▶ *Gaussian process regression networks* [1] and *Deep Gaussian processes* [2] build hierarchical models replacing neurons in neural networks with Gaussian processes.
- ▶ Several recent works [3, 4, 5, 6, 7] have extended Radford Neal's limits to multilayer nets and other architectures.
- ▶ Many more!

[1] *Gaussian process regression networks*. Wilson et. al, ICML 2012.

[2] *Deep Gaussian processes*. Damianou and Lawrence, AISTATS 2013.

[3] *Deep Convolutional Networks as Shallow Gaussian Processes*. Garriga-Alonso et. al, NeurIPS 2018.

[4] *Gaussian Process Behaviour in Wide Deep Neural Networks*. Matthews et. al, ICLR 2018.

[5] *Deep neural networks as Gaussian processes*. Lee et. al, ICLR 2018.

[6] *Bayesian Deep CNNs with Many Channels are Gaussian Processes*. Novak et. al, ICLR 2019.

[7] *Scaling limits of wide neural networks with weight sharing*. Yang, G. arXiv 2019.

Neural Tangent Kernels

- ▶ Recent work [e.g., 1, 2, 3] deriving *neural tangent kernels* (Jacot et. al, 2018) from infinite neural network limits, with promising results.
- ▶ Note that most kernels from infinite neural network limits have a *fixed structure*. On the other hand, standard neural networks essentially *learn* a similarity metric (kernel) for the data. Learning a kernel amounts to *representation learning*. Bridging this gap is interesting future work.

[1] *Neural tangent kernel: convergence and generalization in neural networks*. Jacot et. al, NeurIPS 2018.

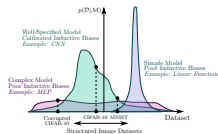
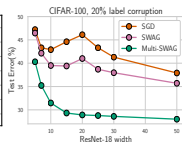
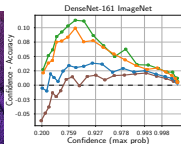
[2] *On exact computation with an infinitely wide neural net*. Arora et. al, NeurIPS 2019.

[3] *Harnessing the Power of Infinitely Wide Deep Nets on Small-data Tasks*. Arora et. al, arXiv 2019.

Bayesian Non-Parametric Deep Learning

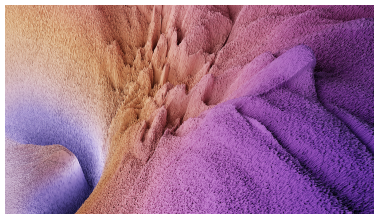
- ▶ There are many ways to realize Bayesian principles in deep learning, in addition to the standard approach of marginalizing distributions over parameters in a neural network.
- ▶ Here we have considered how we can use neural networks to provide inductive biases in Gaussian processes, for a *Bayesian non-parametric approach to deep learning*.
- ▶ In general, combining principles of Bayesian nonparametrics with deep learning is an exciting area for future work.

- ▶ **Part 1: Introduction to Bayesian modelling**
 - ▶ *Importance of model averaging, epistemic uncertainty*
- ▶ **Part 2: The function-space view**
 - ▶ *Gaussian processes, infinite neural networks, training a neural network is learning a kernel, Bayesian non-parametric deep learning*
- ▶ **Part 3: Practical methods for Bayesian deep learning**
 - ▶ *Loss surfaces, SWAG, Subspace Inference, K-FAC Laplace, MC Dropout*
- ▶ **Part 4: Bayesian model construction and generalization**
 - ▶ *Deep ensembles, MultiSWAG, tempering, prior-specification, re-thinking generalization, double descent, width-depth trade-offs*

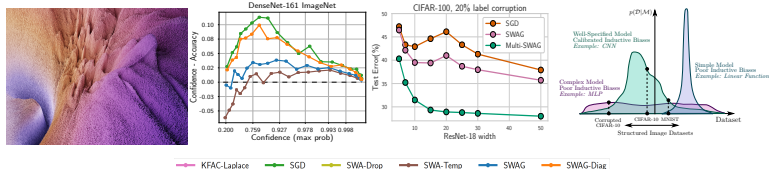


— K-FAC-Laplace
 — SGD
 — SWA-Drop
 — SWA-Temp
 — SWAG
 — SWAG-Diag

Part 3: Methods for Bayesian Deep Learning

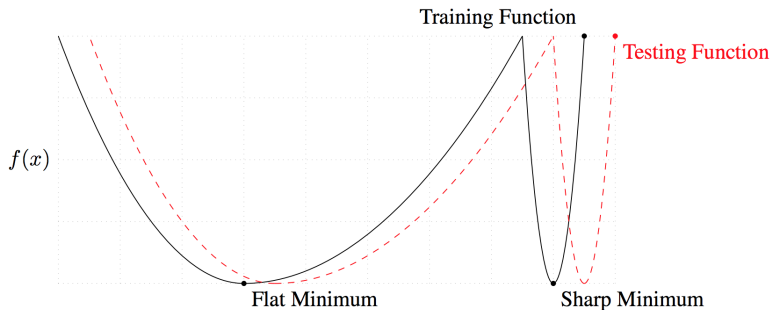


- ▶ **Part 1: Introduction to Bayesian modelling**
 - ▶ *Importance of model averaging, epistemic uncertainty*
- ▶ **Part 2: The function-space view**
 - ▶ *Gaussian processes, infinite neural networks, Bayesian non-parametric deep learning*
- ▶ **Part 3: Practical methods for Bayesian deep learning**
 - ▶ *Loss surfaces, SWAG, Subspace Inference, K-FAC Laplace, MC Dropout*
- ▶ **Part 4: Bayesian model construction and generalization**
 - ▶ *Deep ensembles, MultiSWAG, tempering, prior-specification, re-thinking generalization, double descent, width-depth trade-offs*



— KFAC-Laplace — SGD — SWA-Drop — SWA-Temp — SWAG — SWAG-Diag

Wide Optima Generalize Better



Keskar et. al, ICLR 2017.

On Large-Batch Training for Deep Learning: Generalization Gap and Sharp Minima.

Bayesian integration will give very different predictions in deep learning especially!

Understanding Loss Surfaces for BMA

Recall the *Bayesian model average* (BMA):

$$p(y|x_*, \mathcal{D}) = \int p(y|x_*, w)p(w|\mathcal{D})dw. \quad (71)$$

- ▶ The posterior $p(w|\mathcal{D})$ (or loss $\mathcal{L} = -\log p(w|\mathcal{D})$) for neural networks is extraordinarily complex, containing many complementary solutions, which is why BMA is *especially* significant in deep learning.
- ▶ Understanding the structure of neural network loss landscapes is crucial for better estimating the BMA.

Mode Connectivity

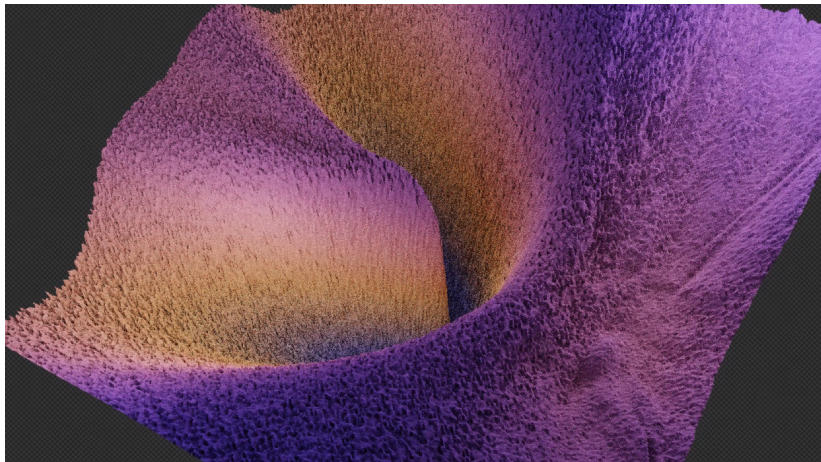


Loss Surfaces, Mode Connectivity, and Fast Ensembling of DNNs.

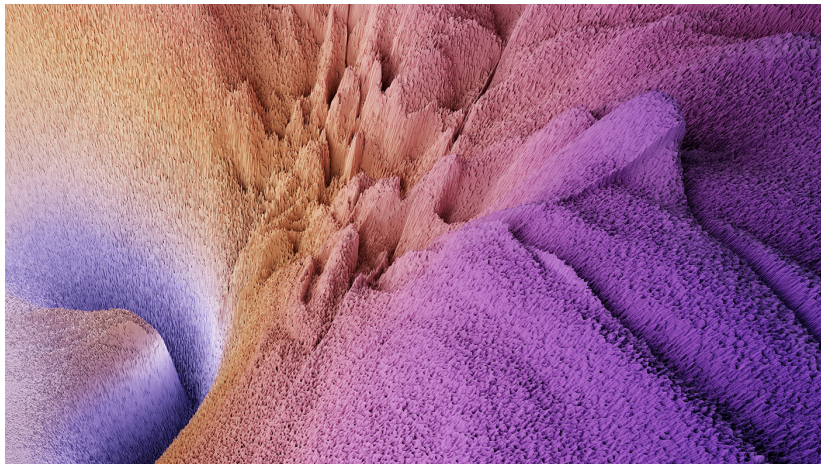
T. Garipov, P. Izmailov, D. Podoprikin, D. Vetrov, A.G. Wilson. NeurIPS 2018.

Loss landscape figures in collaboration with Javier Ideami (losslandscape.com).

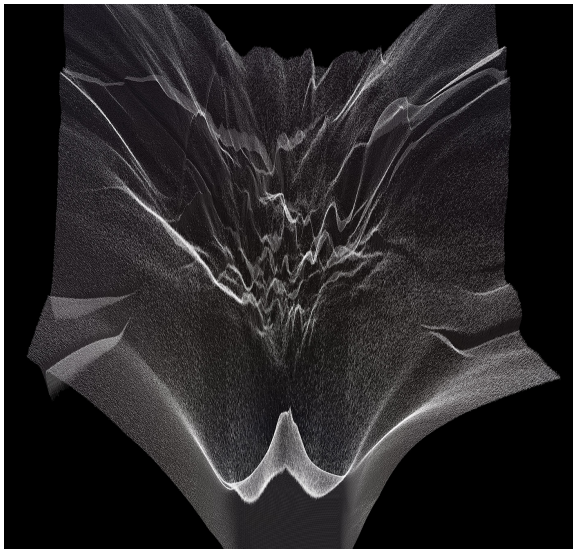
Mode Connectivity



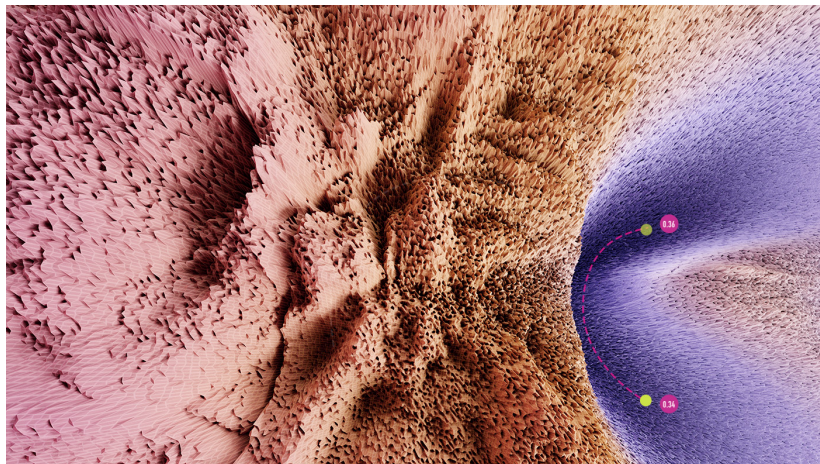
Mode Connectivity



Mode Connectivity



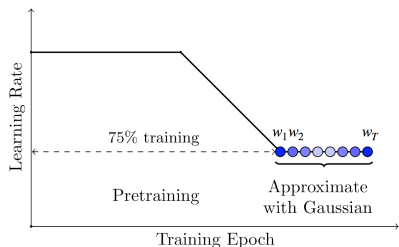
Mode Connectivity



Can we recycle geometric information in the SGD trajectory for scalable posterior approximations, centred on flat regions of the loss?

Uncertainty Representation with SWAG

1. Leverage theory that shows SGD with a constant learning rate is approximately sampling from a Gaussian distribution.
2. Compute first *two* moments of SGD trajectory (SWA computes just the first).
3. Use these moments to construct a Gaussian approximation in weight space.
4. Sample from this Gaussian distribution, pass samples through predictive distribution, and form a Bayesian model average.

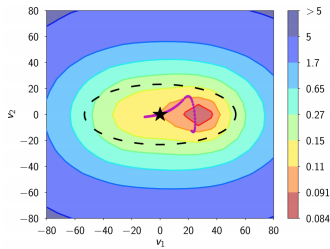


$$p(y_* | \mathcal{D}) \approx \frac{1}{J} \sum_{j=1}^J p(y_* | w_j), \quad w_j \sim q(w | \mathcal{D}), \quad q(w | \mathcal{D}) = \mathcal{N}(\bar{w}, K)$$

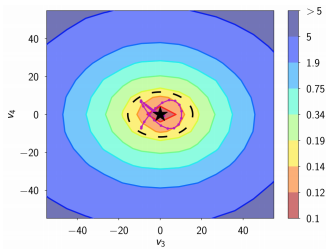
$$\bar{w} = \frac{1}{T} \sum_i w_i, \quad K = \frac{1}{2} \left(\frac{1}{T-1} \sum_i (w_i - \bar{w})(w_i - \bar{w})^T + \frac{1}{T-1} \sum_i \text{diag}(w_i - \bar{w})^2 \right)$$

SWAG: *A Simple Baseline for Bayesian Uncertainty in Deep Learning*. Maddox et. al, NeurIPS 2019.
SWA: *Averaging Weights Leads to Wider Optima and Better Generalization*. Izmailov et. al, UAI 2018.

Trajectory in PCA Subspace

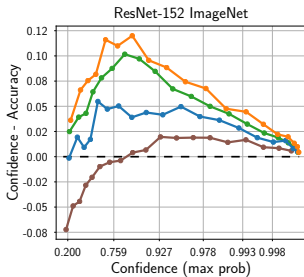
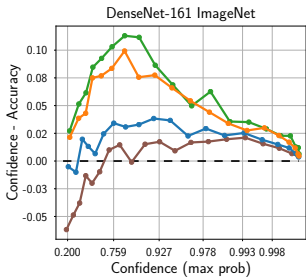
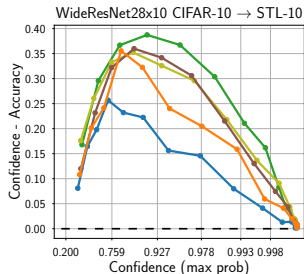
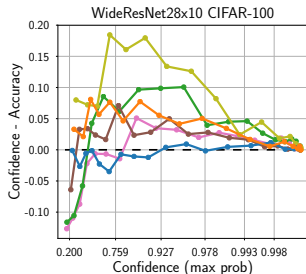


\star SWA — Trajectory (proj)
- - SWAG 3σ region



\star SWA — Trajectory (proj)
- - SWAG 3σ region

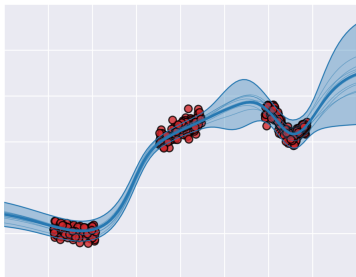
Uncertainty Calibration



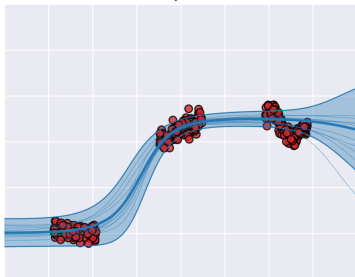
— KFAC-Laplace — SGD — SWA-Drop — SWA-Temp — SWAG — SWAG-Diag

SWAG Regression Uncertainty

SWAG



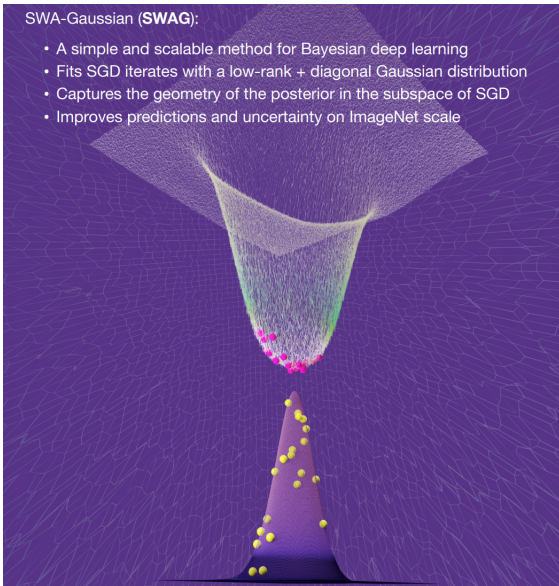
Full Space VI



SWAG Visualization

SWA-Gaussian (SWAG):

- A simple and scalable method for Bayesian deep learning
- Fits SGD iterates with a low-rank + diagonal Gaussian distribution
- Captures the geometry of the posterior in the subspace of SGD
- Improves predictions and uncertainty on ImageNet scale



Subspace Inference for Bayesian Deep Learning

A modular approach:

- ▶ Construct a subspace of a network with a high dimensional parameter space
- ▶ Perform inference directly in the subspace
- ▶ Sample from approximate posterior for Bayesian model averaging

We can approximate the posterior of a WideResNet with 36 million parameters in a 5D subspace and achieve state-of-the-art results!

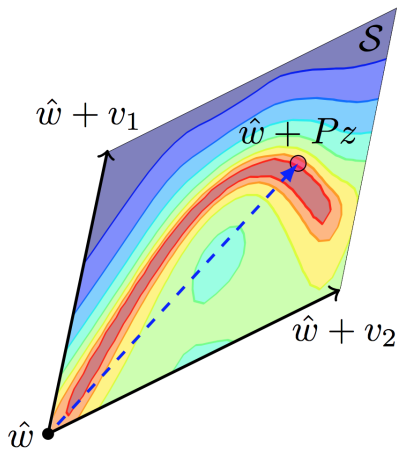
Subspace Inference for Bayesian Deep Learning.

P. Izmailov, W. Maddox, P. Kirichenko, T. Garipov, D. Vetrov, A.G. Wilson

UAI 2018

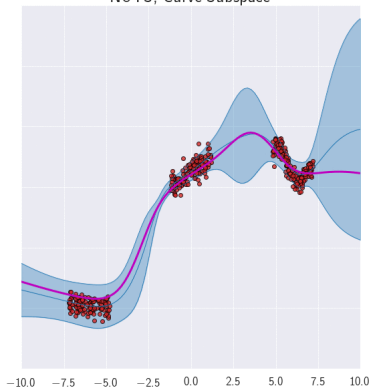
Subspace Construction

- ▶ Choose shift \hat{w} and basis vectors $\{d_1, \dots, d_k\}$.
- ▶ Define subspace $S = \{w | w = \hat{w} + z_1 d_1 + z_k d_k\}$.
- ▶ Likelihood $p(\mathcal{D}|z) = p_M(\mathcal{D}|w = \hat{w} + Pz)$.
- ▶ Posterior inference $p(z|\mathcal{D}) \propto p(\mathcal{D}|z)p(z)$.

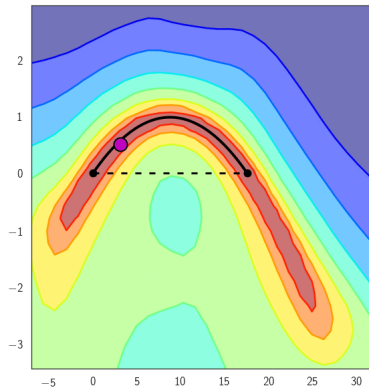


Curve Subspace Traversal

Predictive Distribution
NUTS, Curve Subspace

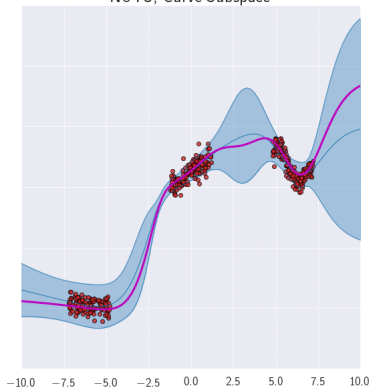


Posterior Log-Density
NUTS, Curve Subspace

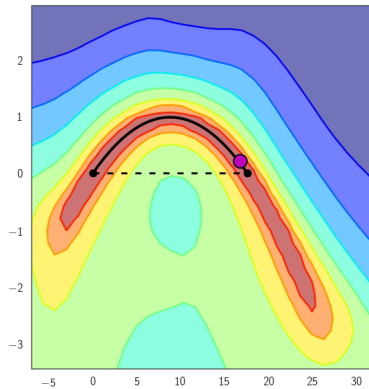


Curve Subspace Traversal

Predictive Distribution
NUTS, Curve Subspace



Posterior Log-Density
NUTS, Curve Subspace

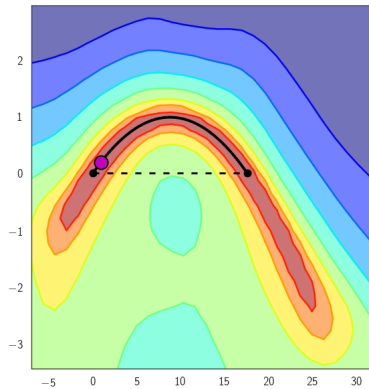


Curve Subspace Traversal

Predictive Distribution
NUTS, Curve Subspace

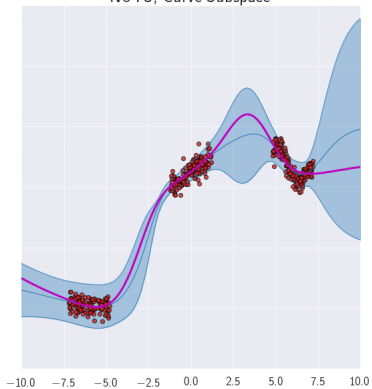


Posterior Log-Density
NUTS, Curve Subspace

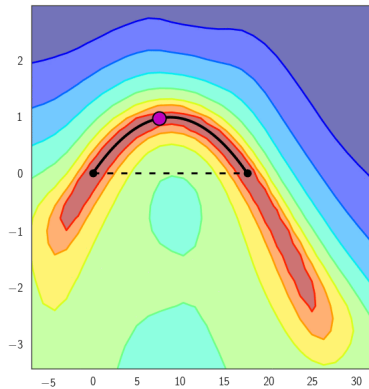


Curve Subspace Traversal

Predictive Distribution
NUTS, Curve Subspace

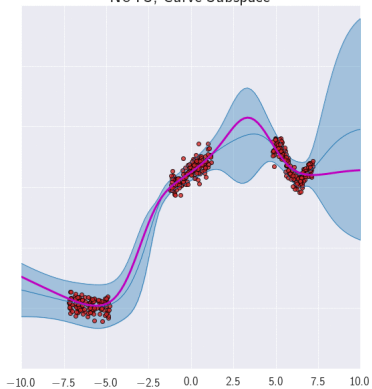


Posterior Log-Density
NUTS, Curve Subspace

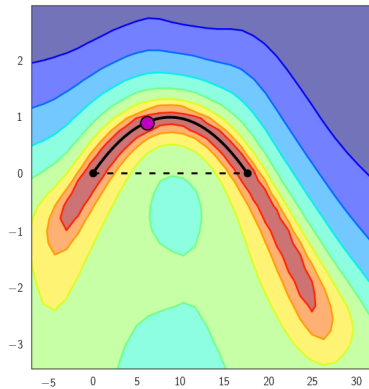


Curve Subspace Traversal

Predictive Distribution
NUTS, Curve Subspace

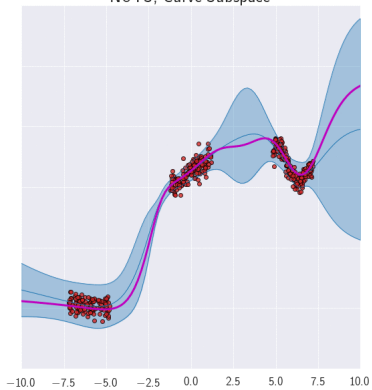


Posterior Log-Density
NUTS, Curve Subspace

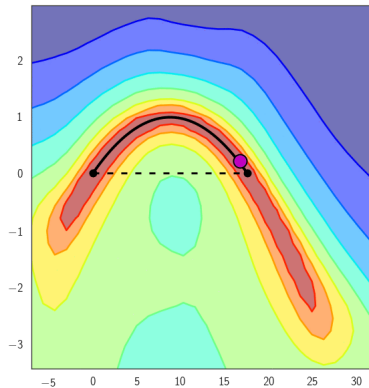


Curve Subspace Traversal

Predictive Distribution
NUTS, Curve Subspace

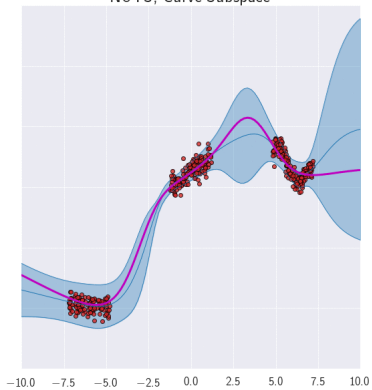


Posterior Log-Density
NUTS, Curve Subspace

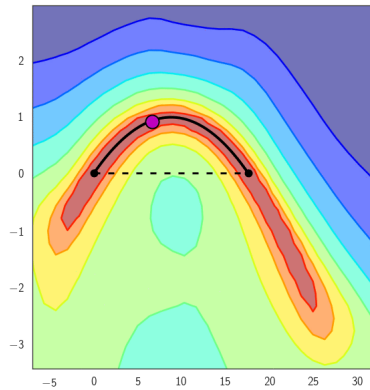


Curve Subspace Traversal

Predictive Distribution
NUTS, Curve Subspace

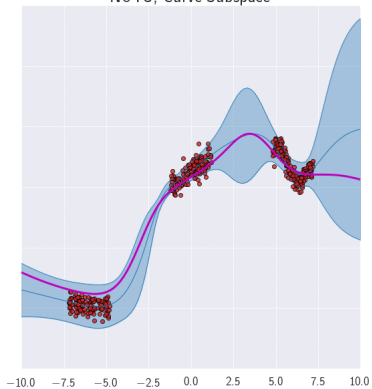


Posterior Log-Density
NUTS, Curve Subspace

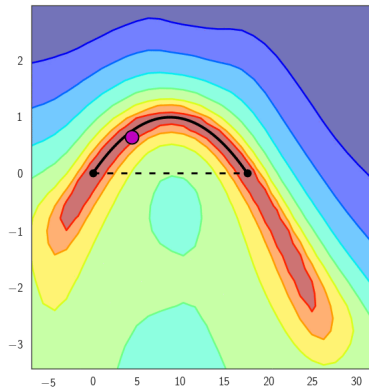


Curve Subspace Traversal

Predictive Distribution
NUTS, Curve Subspace

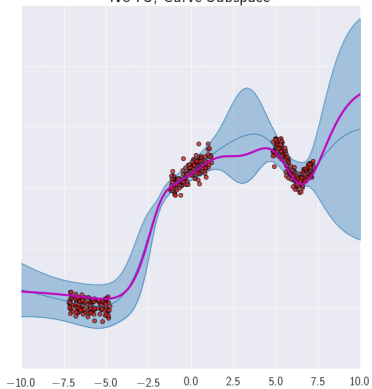


Posterior Log-Density
NUTS, Curve Subspace

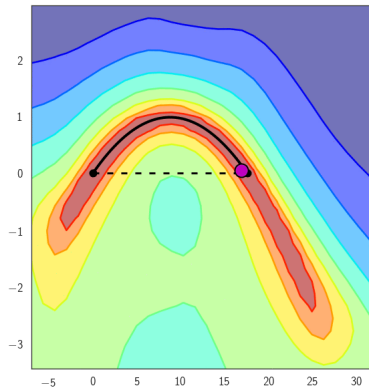


Curve Subspace Traversal

Predictive Distribution
NUTS, Curve Subspace

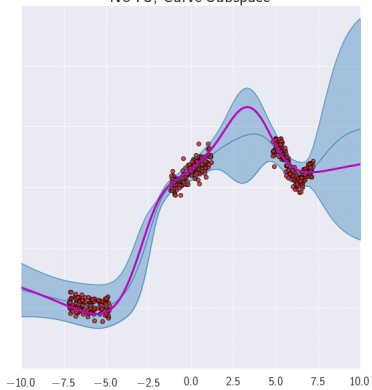


Posterior Log-Density
NUTS, Curve Subspace

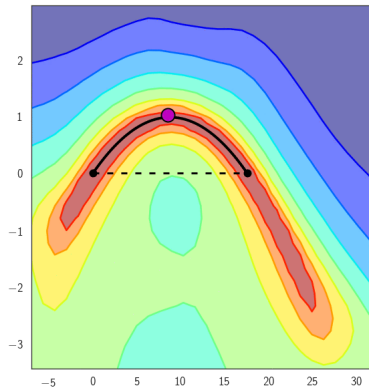


Curve Subspace Traversal

Predictive Distribution
NUTS, Curve Subspace

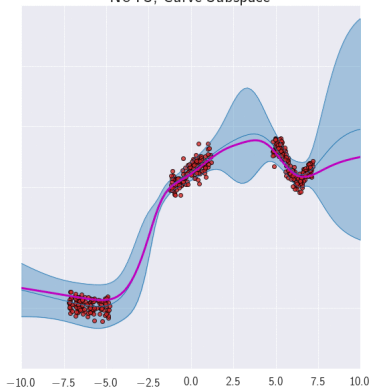


Posterior Log-Density
NUTS, Curve Subspace

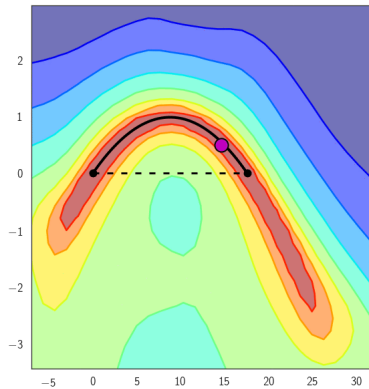


Curve Subspace Traversal

Predictive Distribution
NUTS, Curve Subspace



Posterior Log-Density
NUTS, Curve Subspace

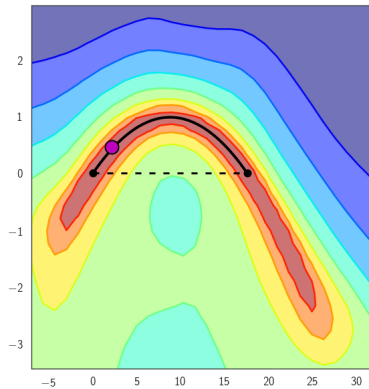


Curve Subspace Traversal

Predictive Distribution
NUTS, Curve Subspace

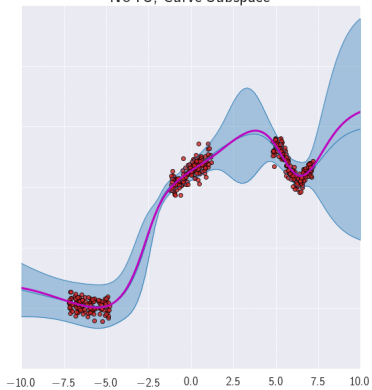


Posterior Log-Density
NUTS, Curve Subspace

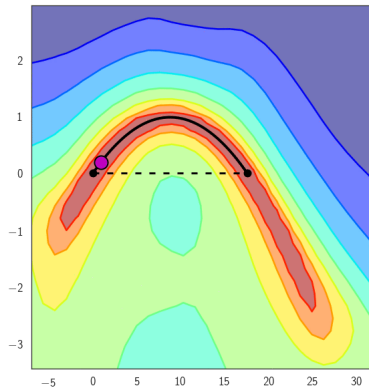


Curve Subspace Traversal

Predictive Distribution
NUTS, Curve Subspace

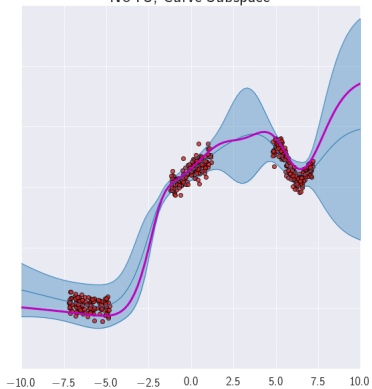


Posterior Log-Density
NUTS, Curve Subspace

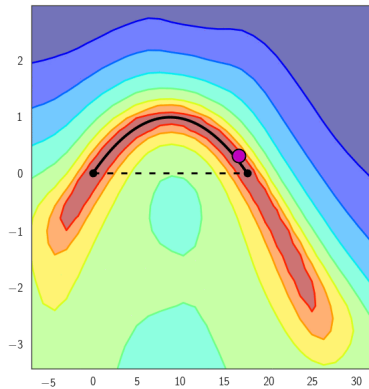


Curve Subspace Traversal

Predictive Distribution
NUTS, Curve Subspace

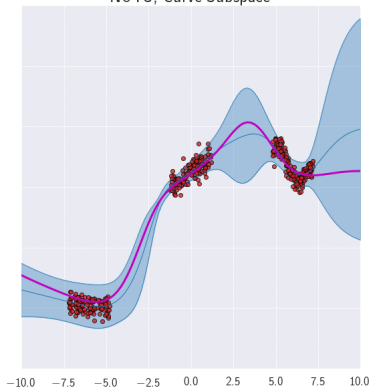


Posterior Log-Density
NUTS, Curve Subspace

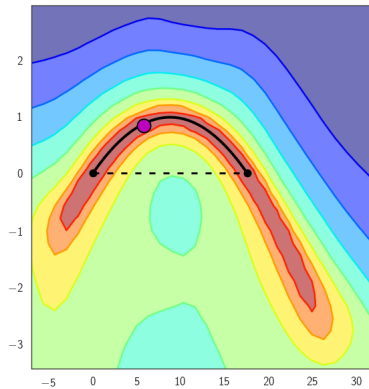


Curve Subspace Traversal

Predictive Distribution
NUTS, Curve Subspace

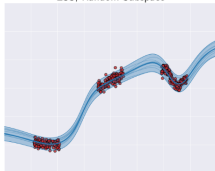


Posterior Log-Density
NUTS, Curve Subspace

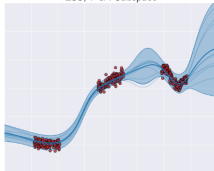


Subspace Comparison (Regression)

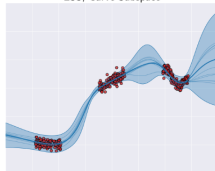
Predictive Distribution
ESS, Random Subspace



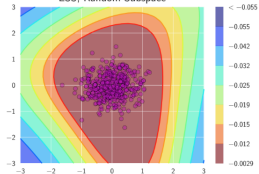
Predictive Distribution
ESS, PCA Subspace



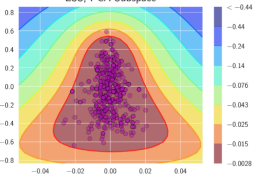
Predictive Distribution
ESS, Curve Subspace



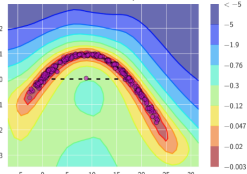
Posterior log-density
ESS, Random Subspace



Posterior log-density
ESS, PCA Subspace

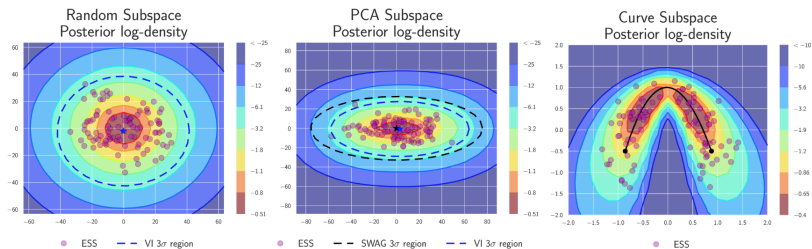


Posterior log-density
ESS, Curve Subspace



Subspace Comparison (Classification)

Accuracy and NLL on CIFAR-100



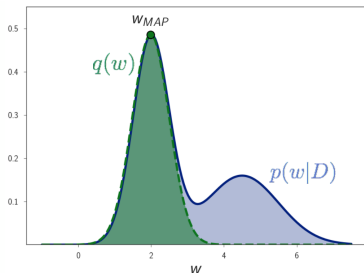
	SGD	Random	PCA	Curve
NLL	0.946 ± 0.001	0.686 ± 0.005	0.665 ± 0.004	0.646
Accuracy (%)	78.50 ± 0.32	80.17 ± 0.03	80.54 ± 0.13	81.28

Bayesian methods also lead to better point predictions in deep learning!

K-FAC Laplace Approximation

Approximate posterior with a Gaussian $q(w|\mathcal{D}, \theta) = \mathcal{N}(w; \mu, A^{-1})$ where $\theta = \{\mu, A^{-1}\}$ are found by a second order Taylor approximation around the log unnormalized true posterior $p(w|\mathcal{D})$:

- ▶ Set $\mu = w_{\text{MAP}} = \operatorname{argmax}_w p(w|\mathcal{D})$.
- ▶ Set $A = -\nabla\nabla \log p(w|\mathcal{D})|_{w=w_{\text{MAP}}}$, the Hessian of the log posterior.
- ▶ For high dimensional w the full A is too large to store. Typically A is taken to be diagonal. K-FAC Laplace models A as a Kronecker factorization.
- ▶ Form approximate BMA: $p(y|x_*) \approx \frac{1}{J} \sum_j p(y|x_*, w_j)$, $w_j \sim q(w|\mathcal{D}, \theta)$.

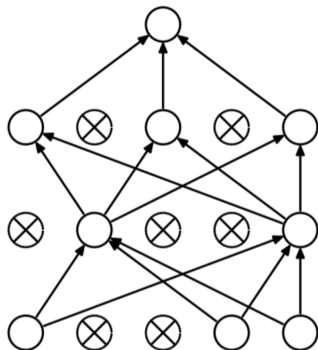


[1] *Bayesian methods for adaptive models*. MacKay, D.J. PhD Thesis, 1992

[2] *A scalable Laplace approximation for neural networks*. Ritter et. al. ICLR, 2018

MC Dropout

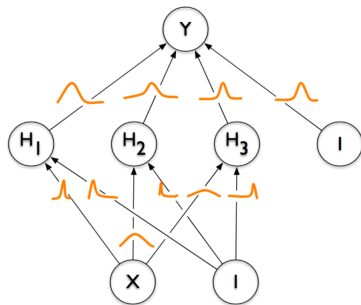
- ▶ Run *drop-out* during train and test (randomly drop out each hidden unit with probability r at each input).
- ▶ In regression, each network can be trained to output a mean μ and variance σ^2 by maximizing a Gaussian likelihood.
- ▶ Create an equally weighted ensemble of the corresponding subnetworks: $f(x) = \frac{1}{J} \sum_j f_j(x, w_j)$.
- ▶ Note that the ensemble doesn't collapse as we get more data (unlike a standard Bayesian model average).



Dropout as a Bayesian approximation: representing model uncertainty in deep learning.
Gal, Y, Ghahramani, Z. ICML 2016

Bayes by Backprop

- ▶ Introduce a (typically Gaussian) approximate posterior $q(w|\theta, \mathcal{D})$. Learn parameters using a variational method: $\theta = \operatorname{argmin}_{\theta} \mathcal{KL}(q||p)$.
- ▶ Manipulation of the KL divergence gives the *evidence lower bound* (ELBO), which can be optimized with SGD (backprop).

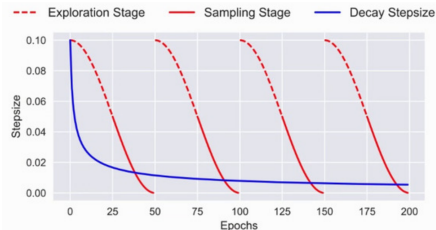
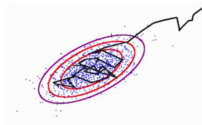


Weight uncertainty in neural networks.

Blundell, C., Cornebise, J., Kavukcuoglu, K., Wierstra, D. ICML 2015

Stochastic MCMC

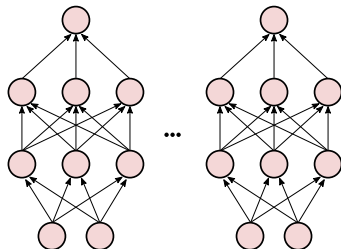
- ▶ Stochastic gradient Langevin dynamics [1, 2]:
$$w_{k+1} = w_k - \alpha_k \nabla U(w) + \sqrt{2\alpha_k} \epsilon, \quad \epsilon \sim \mathcal{N}(0, I)$$
- ▶ $U(w)$ is log posterior.
- ▶ Algorithmically similar to SGD — scalable and generally applicable!
- ▶ Cyclical learning rate schedules α_k greatly enhance the ability to explore complex multimodal loss surfaces [3].



- [1] *Bayesian learning via stochastic gradient Langevin dynamics*. Welling, M., Teh, Y. ICML 2011
[2] *Stochastic gradient Hamiltonian Monte Carlo*. Chen, T., Fox, E., Guestin, C. ICML 2014
[3] *Cyclical stochastic gradient MCMC for Bayesian deep learning*. Zhang et. al, ICLR 2020

Deep Ensembles

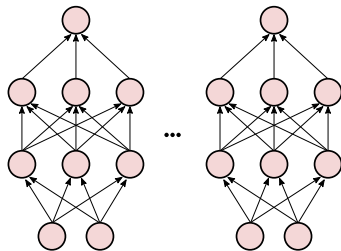
- ▶ Specify a neural network architecture.
- ▶ Retrain the model multiple times to find different SGD solutions w_1, \dots, w_J .
- ▶ In regression, each model specified to output a mean μ and variance σ^2 .
- ▶ Take an equal average of the corresponding models
$$f(x) = \frac{1}{J} \sum_j f(x, w_j).$$



Simple and scalable predictive uncertainty estimation using deep ensembles.
Lakshminarayanan et. al, NeurIPS 2017.

Deep Ensembles

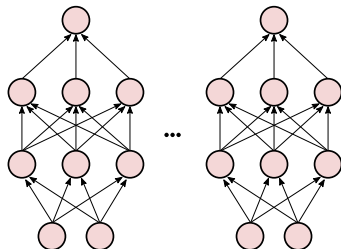
- ▶ Specify a neural network architecture.
- ▶ Retrain the model multiple times to find different SGD solutions w_1, \dots, w_J .
- ▶ In regression, each model specified to output a mean μ and variance σ^2 .
- ▶ Take an equal average of the corresponding models
$$f(x) = \frac{1}{J} \sum_j f(x, w_j).$$



... But is deep ensembles *Bayesian*? Isn't it often explicitly treated as a competing approach to Bayesian methods?

Deep Ensembles

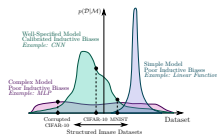
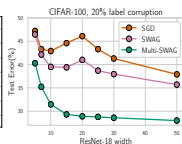
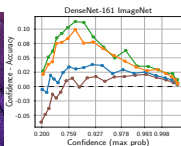
- ▶ Specify a neural network architecture.
- ▶ Retrain the model multiple times to find different SGD solutions w_1, \dots, w_J .
- ▶ In regression, each model specified to output a mean μ and variance σ^2 .
- ▶ Take an equal average of the corresponding models
$$f(x) = \frac{1}{J} \sum_j f(x, w_j).$$



... But is deep ensembles *Bayesian*? Isn't it often explicitly treated as a competing approach to Bayesian methods?

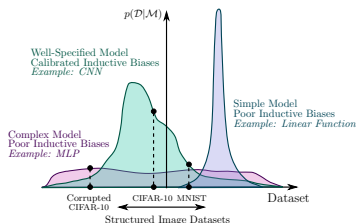
In the next part we will argue that deep ensembles in fact provides a *better* approximation to Bayesian model averaging than many of the above methods! We will also introduce MultiSWAG which generalizes deep ensembles for a more accurate BMA.

- ▶ **Part 1: Introduction to Bayesian modelling**
 - ▶ *Importance of model averaging, epistemic uncertainty*
- ▶ **Part 2: The function-space view**
 - ▶ *Gaussian processes, infinite neural networks, Bayesian non-parametric deep learning*
- ▶ **Part 3: Practical methods for Bayesian deep learning**
 - ▶ *Loss surfaces, SWAG, Subspace Inference, K-FAC Laplace, MC Dropout*
- ▶ **Part 4: Bayesian model construction and generalization**
 - ▶ *Deep ensembles, MultiSWAG, tempering, prior-specification, re-thinking generalization, double descent, width-depth trade-offs*

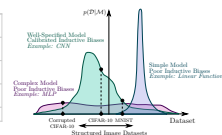
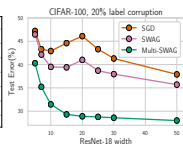
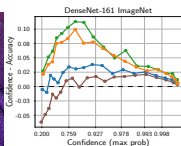


—●— KFAC-Laplace
 —●— SGD
 —●— SWA-Drop
 —●— SWA-Temp
 —●— SWAG
 —●— SWAG-Diag

Part 4: Bayesian Model Construction and Generalization



- ▶ **Part 1: Introduction to Bayesian modelling**
 - ▶ *Importance of model averaging, epistemic uncertainty*
- ▶ **Part 2: The function-space view**
 - ▶ *Gaussian processes, infinite neural networks, Bayesian non-parametric deep learning*
- ▶ **Part 3: Practical methods for Bayesian deep learning**
 - ▶ *Loss surfaces, SWAG, Subspace Inference, K-FAC Laplace, MC Dropout*
- ▶ **Part 4: Bayesian model construction and generalization**
 - ▶ *Deep ensembles, MultiSWAG, tempering, prior-specification, re-thinking generalization, double descent, width-depth trade-offs*



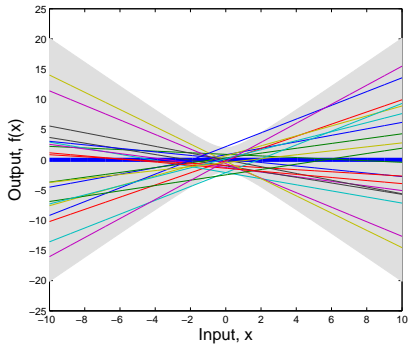
—●— KFAC-Laplace
 —●— SGD
 —●— SWA-Drop
 —●— SWA-Temp
 —●— SWAG
 —●— SWAG-Diag

A Function-Space View

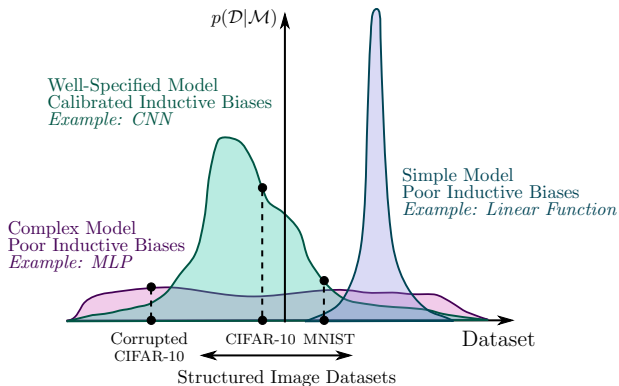
Consider the simple linear model,

$$f(x) = w_0 + w_1x, \quad (72)$$

$$w_0, w_1 \sim \mathcal{N}(0, 1). \quad (73)$$



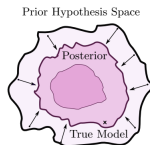
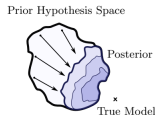
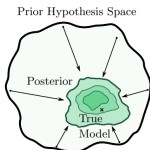
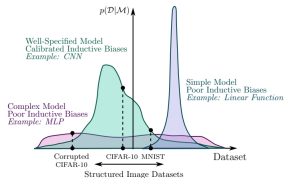
Model Construction and Generalization



$$p(\mathcal{D}|\mathcal{M}) = \int p(\mathcal{D}|\mathcal{M}, w)p(w)dw \quad (74)$$

How do we learn?

- ▶ The ability for a system to learn is determined by its *support* (which solutions are a priori possible) and *inductive biases* (which solutions are a priori likely).
- ▶ We should not conflate *flexibility* and *complexity*.



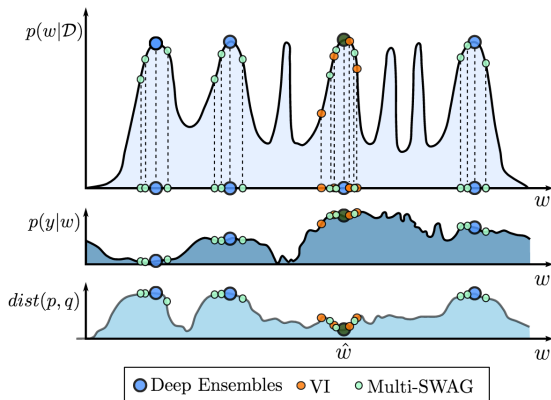
Bayesian Deep Learning and a Probabilistic Perspective of Generalization

Wilson and Izmailov, 2020

arXiv 2002.08791

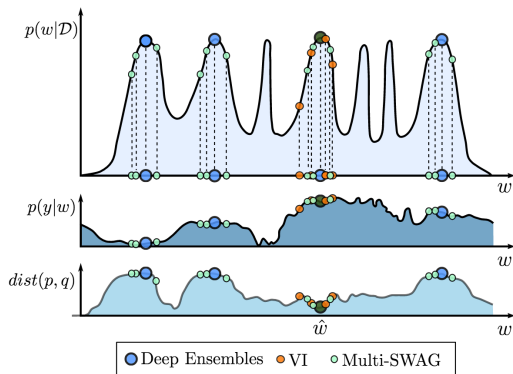
Better Marginalization

$$p(y|x_*, \mathcal{D}) = \int p(y|x_*, w)p(w|\mathcal{D})dw. \quad (75)$$



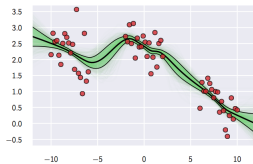
Better Marginalization

$$p(y|x_*, \mathcal{D}) = \int p(y|x_*, w)p(w|\mathcal{D})dw. \quad (76)$$

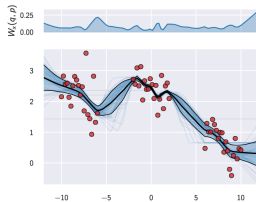


- ▶ Be wary about viewing BMA through the prism of simple MC.
- ▶ We want to best estimate the BMA integral given computational constraints.
- ▶ View BMA estimation as active learning, rather than posterior sampling.

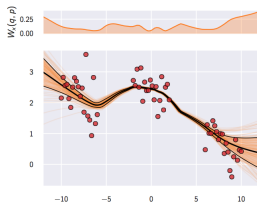
Better Marginalization: Deep Ensembles



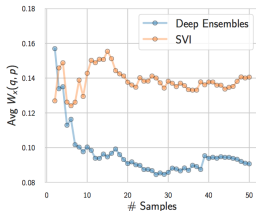
(a) Exact



(b) Deep Ensembles

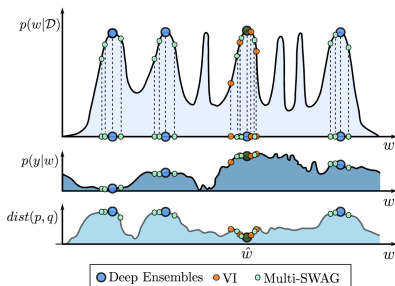


(c) Variational Inference



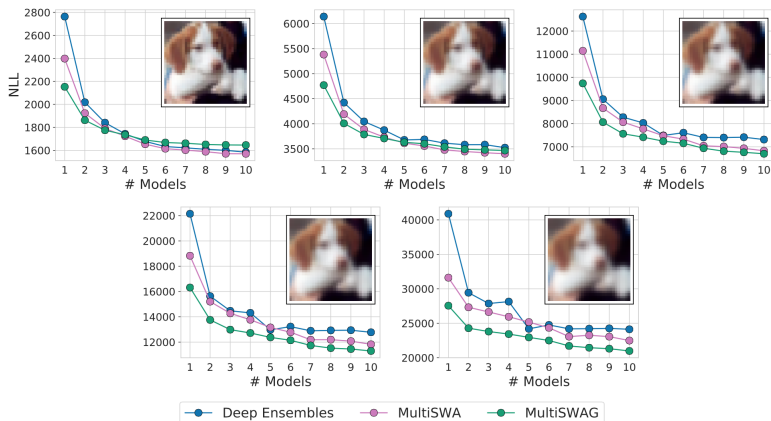
(d) Discrepancy with True BMA

$$p(y|x_*, \mathcal{D}) = \int p(y|x_*, w)p(w|\mathcal{D})dw. \quad (77)$$



- ▶ MultiSWAG forms a Gaussian mixture posterior from multiple independent SWAG solutions.
- ▶ Like deep ensembles, MultiSWAG incorporates multiple basins of attraction in the model average, but it additionally marginalizes within basins of attraction for a better approximation to the BMA.

Better Marginalization: MultiSWAG

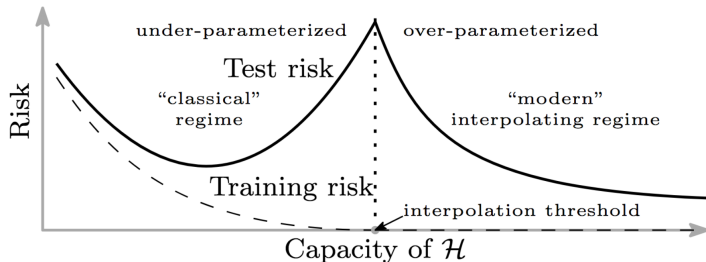


[1] *Can you trust your model's uncertainty? Evaluating predictive uncertainty under dataset shift.*

Ovadia et. al, 2019

[2] *Bayesian Deep Learning and a Probabilistic Perspective of Generalization.* Wilson and Izmailov, 2020

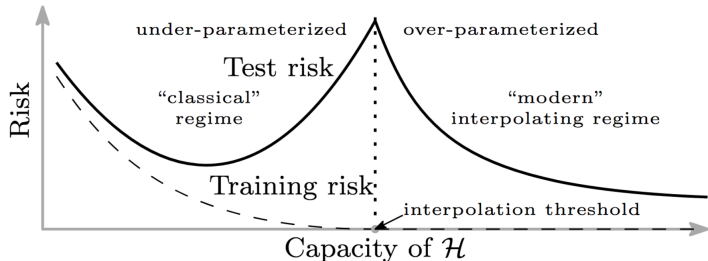
Double Descent



Belkin et. al (2018)

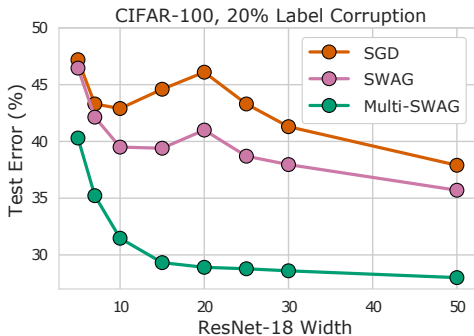
Reconciling modern machine learning practice and the bias-variance trade-off. Belkin et. al, 2018

Double Descent



Should a Bayesian model experience double descent?

Bayesian Model Averaging Alleviates Double Descent



Bayesian Deep Learning and a Probabilistic Perspective of Generalization. Wilson & Izmailov, 2020

Neural Network Priors

A parameter prior $p(w) = \mathcal{N}(0, \alpha^2)$ with a neural network architecture $f(x, w)$ induces a structured distribution over functions $p(f(x))$.

Deep Image Prior

- ▶ *Randomly initialized CNNs without training* provide excellent performance for image denoising, super-resolution, and inpainting: a sample function from $p(f(x))$ captures low-level image statistics, before any training.

Random Network Features

- ▶ Pre-processing CIFAR-10 with a *randomly initialized untrained CNN* dramatically improves the test performance of a Gaussian kernel on pixels from 54% accuracy to 71%, with an additional 2% from ℓ_2 regularization.

[1] *Deep Image Prior*. Ulyanov, D., Vedaldi, A., Lempitsky, V. CVPR 2018.

[2] *Understanding Deep Learning Requires Rethinking Generalization*. Zhang et. al, ICLR 2016.

[3] *Bayesian Deep Learning and a Probabilistic Perspective of Generalization*. Wilson & Izmailov, 2020.

Tempered Posteriors

In Bayesian deep learning it is typical to consider the *tempered* posterior:

$$p_T(w|\mathcal{D}) = \frac{1}{Z(T)} p(\mathcal{D}|w)^{1/T} p(w), \quad (78)$$

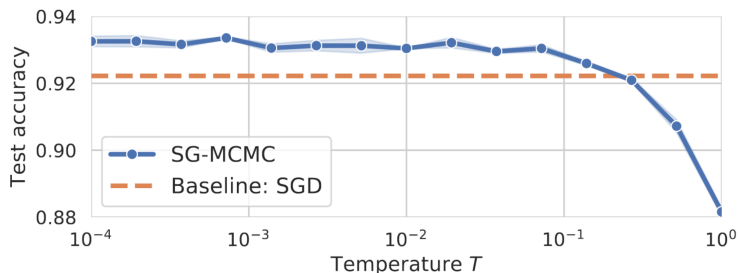
where T is a *temperature* parameter, and $Z(T)$ is the normalizing constant corresponding to temperature T . The temperature parameter controls how the prior and likelihood interact in the posterior:

- ▶ $T < 1$ corresponds to *cold posteriors*, where the posterior distribution is more concentrated around solutions with high likelihood.
- ▶ $T = 1$ corresponds to the standard Bayesian posterior distribution.
- ▶ $T > 1$ corresponds to *warm posteriors*, where the prior effect is stronger and the posterior collapse is slower.

E.g.: *The safe Bayesian*. Grunwald, P. COLT 2012.

Cold Posteriors

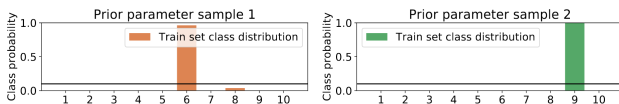
Wenzel et. al (2020) (at this ICML!) highlight the result that for $p(w) = N(0, I)$ cold posteriors with $T < 1$ often provide improved performance.



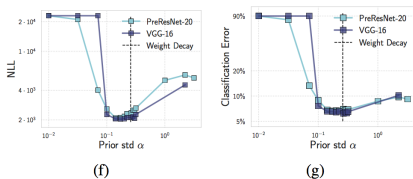
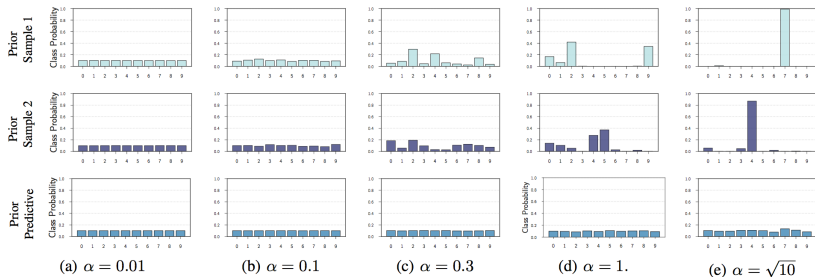
How good is the Bayes posterior in deep neural networks really? Wenzel et. al, ICML 2020.

Prior Misspecification?

They suggest the result is due to prior misspecification, showing that sample functions $p(f(x))$ seem to assign one label to most classes on CIFAR-10.

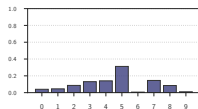
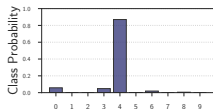
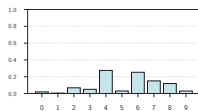
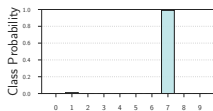


Changing the prior variance scale α



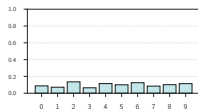
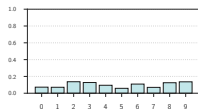
Bayesian Deep Learning and a Probabilistic Perspective of Generalization. Wilson & Izmailov, 2020.

The effect of data on the posterior



(a) Prior ($\alpha = \sqrt{10}$)

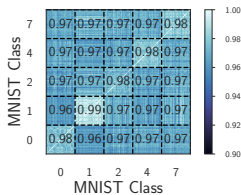
(b) 10 datapoints



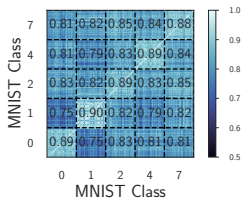
(c) 100 datapoints

(d) 10000 datapoints

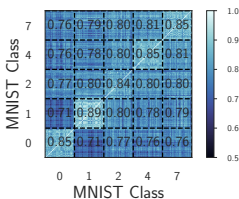
Prior Class Correlations



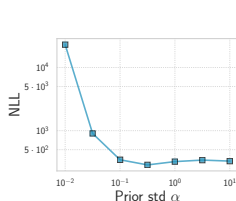
(e) $\alpha = 0.02$



(f) $\alpha = 0.1$



(g) $\alpha = 1$



(h)

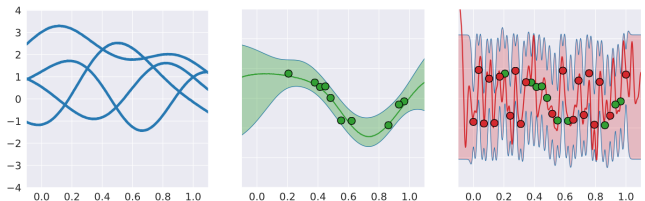
Thoughts on Tempering (Part 1)

- ▶ It would be surprising if $T = 1$ was the best setting of this hyperparameter.
- ▶ Our models are certainly misspecified, and we should acknowledge that misspecification in our estimation procedure by learning T . Learning T is not too different from learning other properties of the likelihood, such as noise.
- ▶ A tempered posterior is a more honest reflection of our prior beliefs than the untempered posterior. Bayesian inference is about honestly reflecting our beliefs in the modelling process.

Thoughts on Tempering (Part 2)

- ▶ While certainly the prior $p(f(x))$ is misspecified, the result of assigning one class to most data is a *soft* prior bias, which (1) doesn't hurt the predictive distribution, (2) is easily corrected by appropriately setting the prior parameter variance α^2 , and (3) is quickly modulated by data.
- ▶ More important is the induced *covariance function* (kernel) over images, which appears reasonable. The deep image prior and random network feature results also suggest this prior is largely reasonable.
- ▶ In addition to not tuning α , the result in Wenzel et. al (2020) could have been exacerbated due to lack of multimodal marginalization.
- ▶ There are cases when $T < 1$ will help given a finite number of samples, even if the untempered model is correctly specified. Imagine estimating the mean of $\mathcal{N}(0, I)$ from samples where $d \gg 1$. The samples will have norm close to \sqrt{d} .

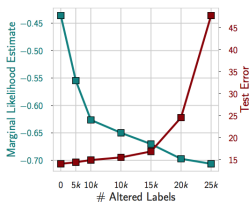
Rethinking Generalization



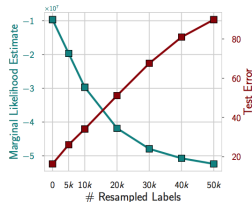
(a) Prior Draws

(b) True Labels

(c) Corrupted Labels



(d) Gaussian Process



(e) PreResNet-20

- [1] *Understanding Deep Learning Requires Rethinking Generalization*. Zhang et. al, ICLR 2016.
[2] *Bayesian Deep Learning and a Probabilistic Perspective of Generalization*. Wilson & Izmailov, 2020.

Function Space Priors

We should embrace the function space perspective in constructing priors.

- ▶ However, if we contrive priors over parameters $p(w)$ to induce distributions over functions $p(f)$ that resemble familiar models such as Gaussian processes with RBF kernels, we could be throwing the baby out with the bathwater.
- ▶ Indeed, neural networks are useful as their own model class precisely because they have different inductive biases from other models.
- ▶ We should try to gain insights by thinking in *function space*, but note that architecture design itself is thinking in function space: properties such as equivariance to translations in convolutional architectures imbue the associated distribution over functions with these properties.

PAC-Bayes provides explicit generalization error bounds for stochastic networks with posterior Q , prior P , training points n , probability $1 - \delta$ based on

$$\sqrt{\frac{\mathcal{KL}(Q||P) + \log(\frac{n}{\delta})}{2(n-1)}}. \quad (79)$$

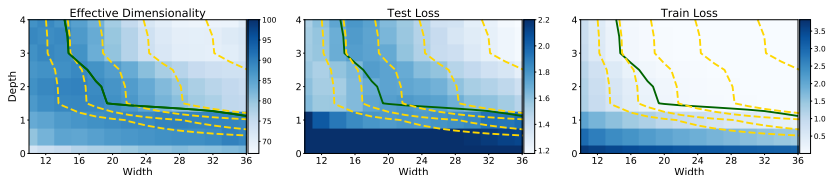
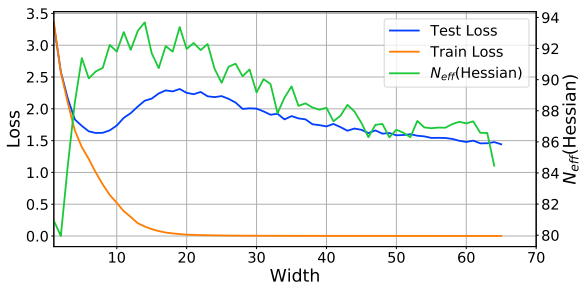
- ▶ Non-vacuous bounds derived from exploiting flatness in Q (e.g., at least 80% generalization accuracy predicted on binary MNIST).
- ▶ Very promising framework but tends not to be *prescriptive* about model construction, or informative for understanding *why* a model generalizes.
- ▶ Bounds are improved by compact P and a low dimensional parameter space. We suggest a P with large support and many parameters.
- ▶ Generalization significantly improved by multimodal Q , but not PAC-Bayes generalization bounds.

Fantastic generalization measures and where to find them. Jiang et. al, 2019.

A primer on PAC-Bayesian learning. Guedj, 2019.

Computing nonvacuous generalization bounds for deep (stochastic) neural networks. Dziugaite & Roy, 2017.

Rethinking Parameter Counting: Effective Dimension



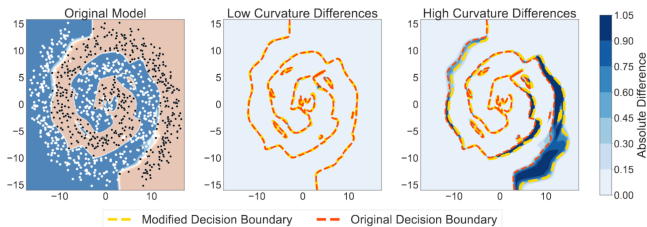
$$N_{\text{eff}}(H) = \sum_i \frac{\lambda_i}{\lambda_i + \alpha}$$

Rethinking Parameter Counting in Deep Models: Effective Dimensionality Revisited.

W. Maddox, G. Benton, A.G. Wilson, 2020.

Properties in Degenerate Directions

Decision boundaries do not change in directions of little posterior contraction, suggesting a mechanism for subspace inference!



Conclusions

- ▶ The key defining feature of Bayesian methods is marginalization, aka Bayesian model averaging.
- ▶ Bayesian model averaging is especially relevant in deep learning, because the loss landscapes contain a rich variety of high performing solutions.
- ▶ We shouldn't think of marginalization purely through the lens of simple Monte Carlo integration.
- ▶ Bayesian methods are now often providing better results than classical training, in accuracy and uncertainty representation, without significant overhead.
- ▶ Don't conflate flexibility and complexity.
- ▶ Don't parameter count as a proxy for complexity.
- ▶ We can resolve several mysterious results in deep learning by thinking about model construction and generalization from a probabilistic perspective.