Bayesian Deep Learning

Borrowed from ICML 2020 Tutorial by Andrew Gordon Wilson, NYU

Deep Neural Networks

- Train a network for certain task, e.g., image recognition
- Find out the optimal weights using gradient descent on some error function
- Using the optimal weights predict the output for a new input

- For a given input – only one output
- Only a single set of weights obtained by optimization – point estimates

Deep Neural Networks

- Very successful for several tasks like face recognition, speech recognition, text classification
- Some questions yet to be answered for safety critical applications like autonomous driving, medical diagnosis
  - How confident are the predictions?
  - Are there gaps in learning?
  - Do we need more training data?
- Why can we train a large network with a small data set? (overparametrization)

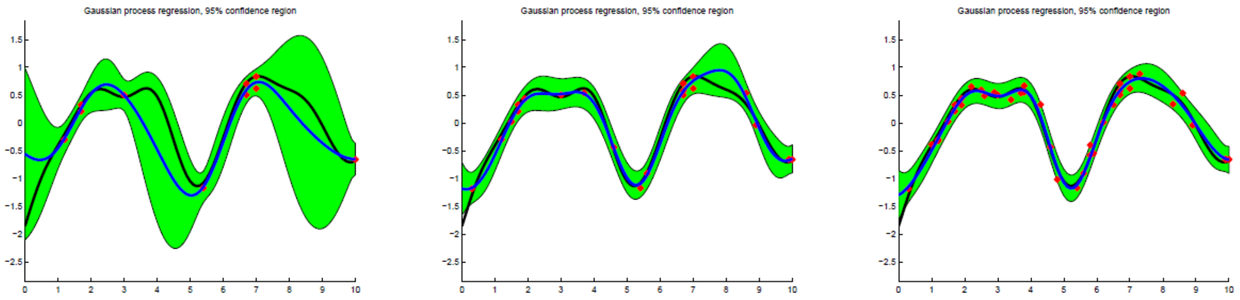
Limitations of DNN

- No quantification of uncertainty in the predictions - what is the confidence in prediction?

- Point estimate of weights by optimization of training error – might not be good for small training data

Bayesian Learning

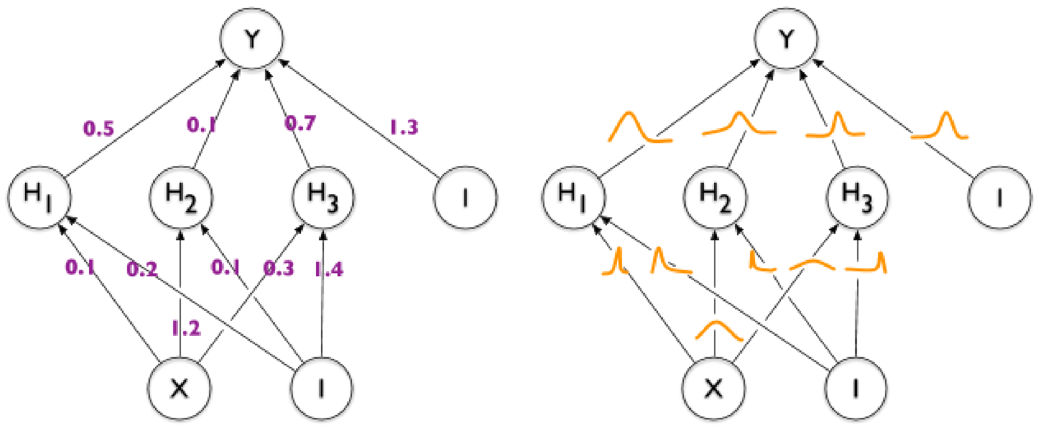
- Output is not a single prediction but a distribution over predictions
- Parameter distributions are estimated instead of point estimates
- Priors are incorporated to smooth small data estimates



(a) (b) (c)

Bayesian Deep Neural Network

- Uses Bayesian approach to handle limitations of deep neural networks
- Provides quantification of uncertainty in predicted output
- Takes into account distribution of weights rather than point estimates
- Helps better training of DNN with less training data



Bayesian Regression

- Regression 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.

- Likelihood

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

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

Bayesian Regression

- Posterior (distribution over parameter w)

$$p(\mathbf{w}|\mathbf{y}, X, \sigma^2) = \frac{p(\mathbf{y}|X, \mathbf{w}, \sigma^2)p(\mathbf{w})}{p(\mathbf{y}|X, \sigma^2)}$$

- Posterior Predictive Distribution (marginalisation over w)

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

Bayesian Model Averaging

- Posterior Predictive Distribution

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

- Average over infinitely many models weighted by their posterior probabilities

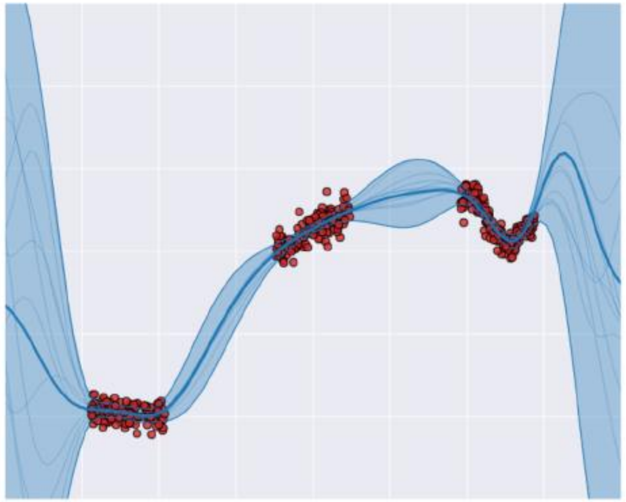
- On the other hand MAP finds w by maximizing posterior – point estimate (optimization approach)

Quantification of Model Uncertainty

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

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

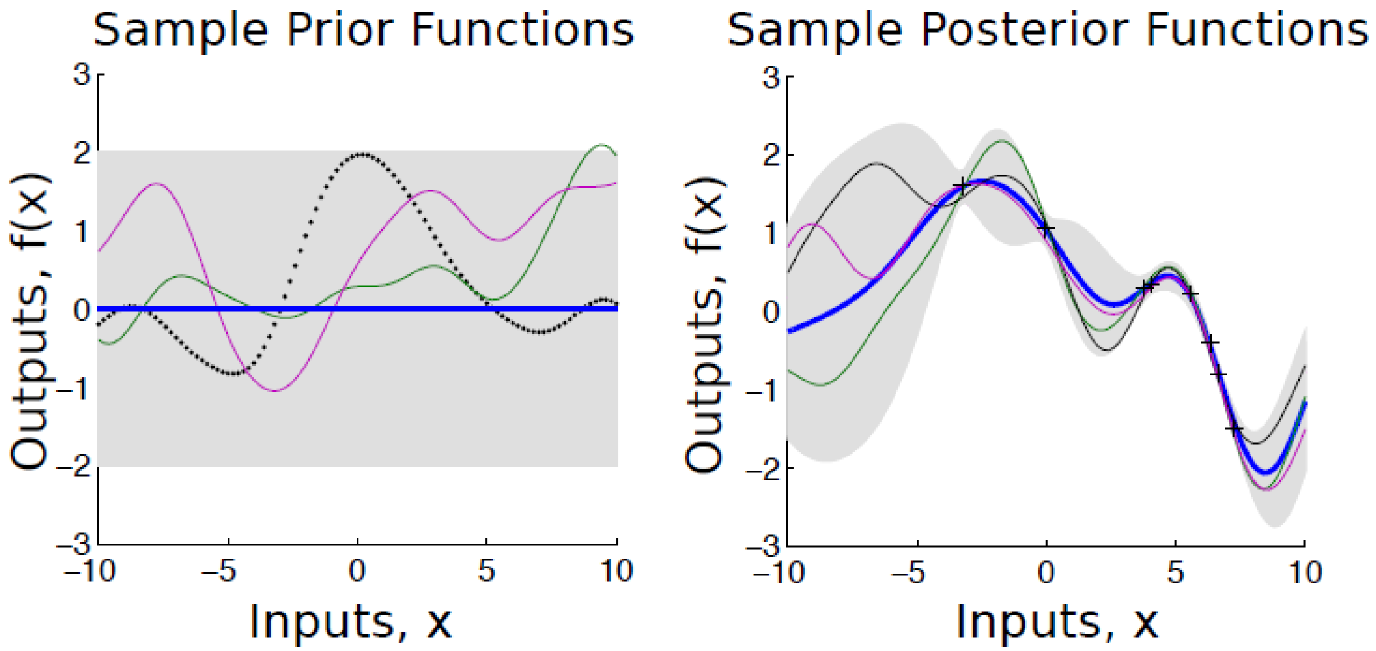
- First Term: Uncertainty in estimate of w – <u>epistemic uncertainty</u>
- Second Term: Noise – aleatoric uncertainty
- Epistemic uncertainty reduces as data increases



Function Space: Gaussian Process

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

$$\overbrace{p(f(x)|\mathcal{D})}^{\text{GP posterior}} \propto \overbrace{p(\mathcal{D}|f(x))}^{\text{Likelihood}}\,\overbrace{p(f(x))}^{\text{GP prior}}$$



Neural Network Kernel

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

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

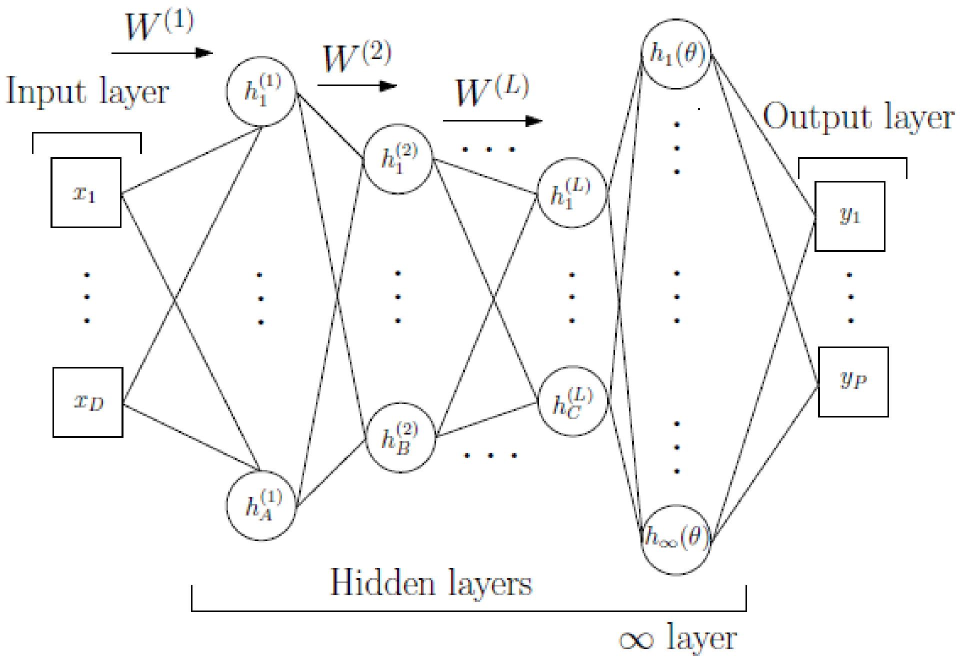
Then we obtain

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

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

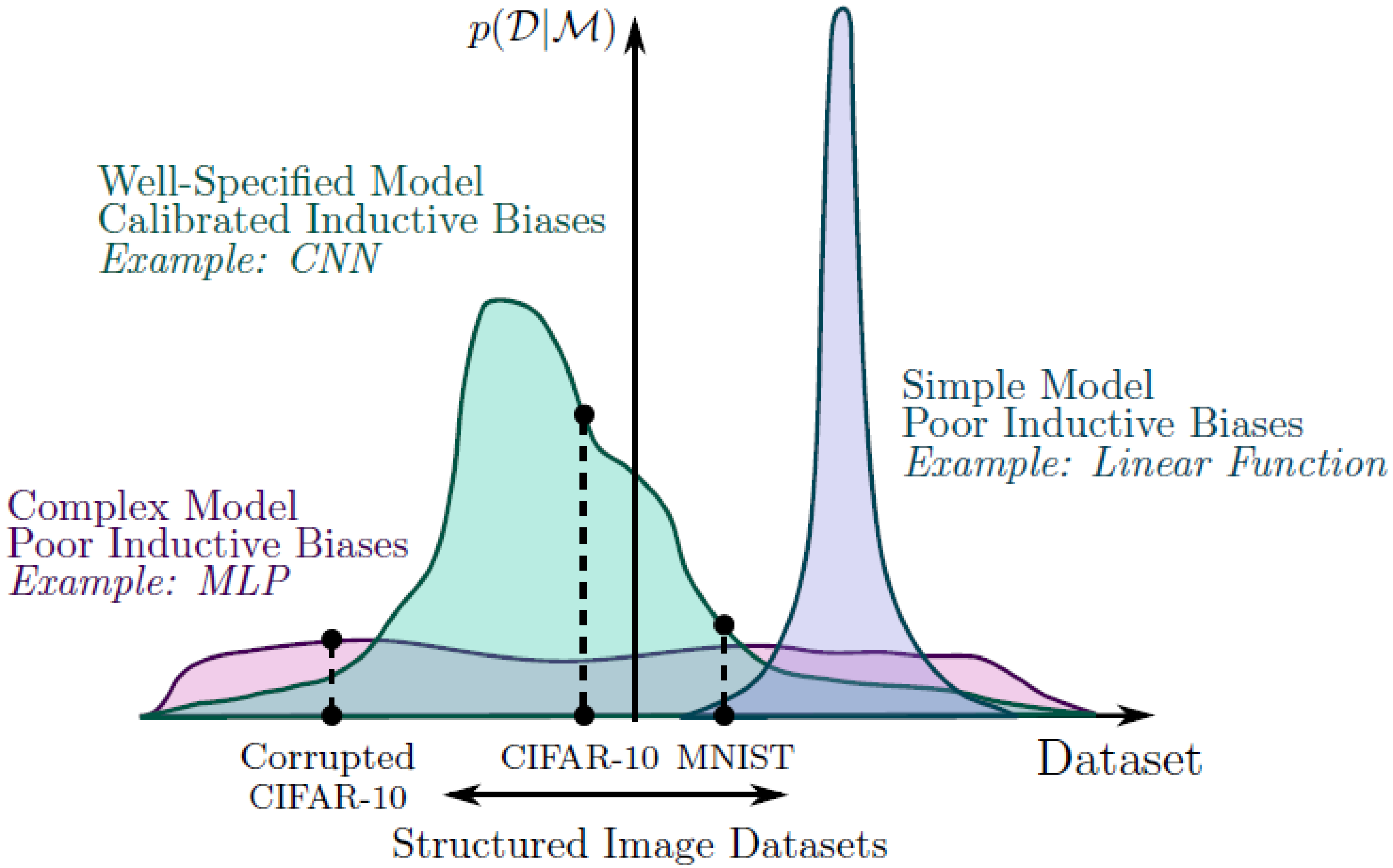
Deep Kernel Learning

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



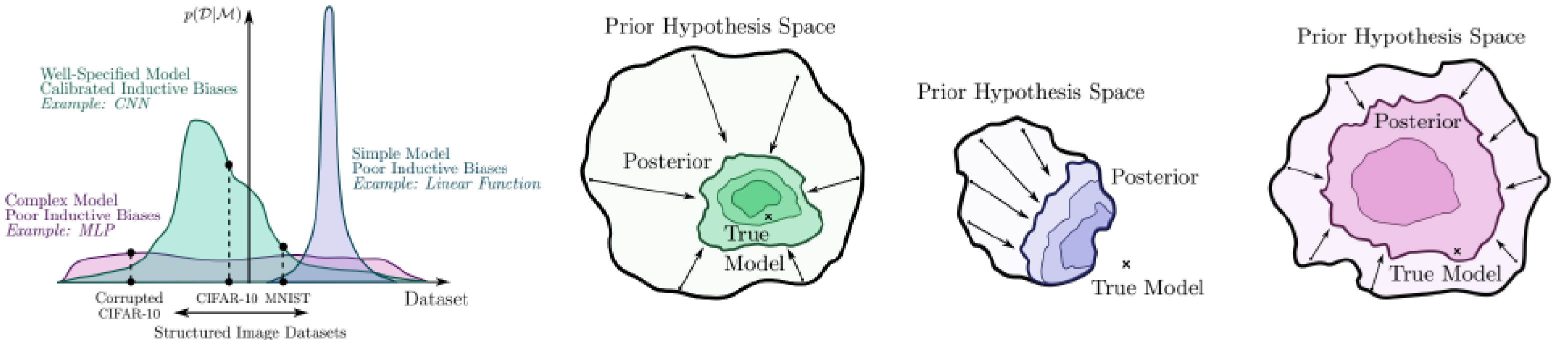
Base kernel hyperparameters θ and deep network hyperparameters w are jointly trained through the marginal likelihood objective.

Deep Model Construction



Deep Model Construction

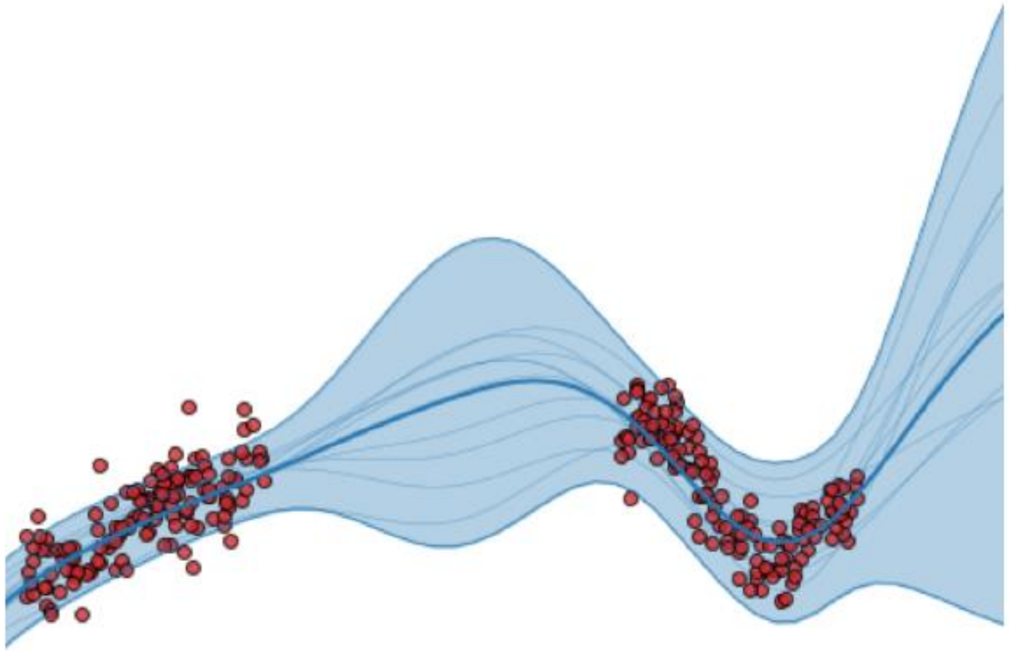
- The ability for a system to learn is determined by its *support* (which solutions are a priori possible) and *inductive biases* (which solutions are a priori likely).
- We should not conflate *flexibility* and *complexity*.
- An influx of new *massive* datasets provide great opportunities to automatically learn rich statistical structure, leading to new scientific discoveries.



Bayesian Deep Learning and a Probabilistic Perspective of Generalization
Wilson and Izmailov, 2020
arXiv 2002.08791

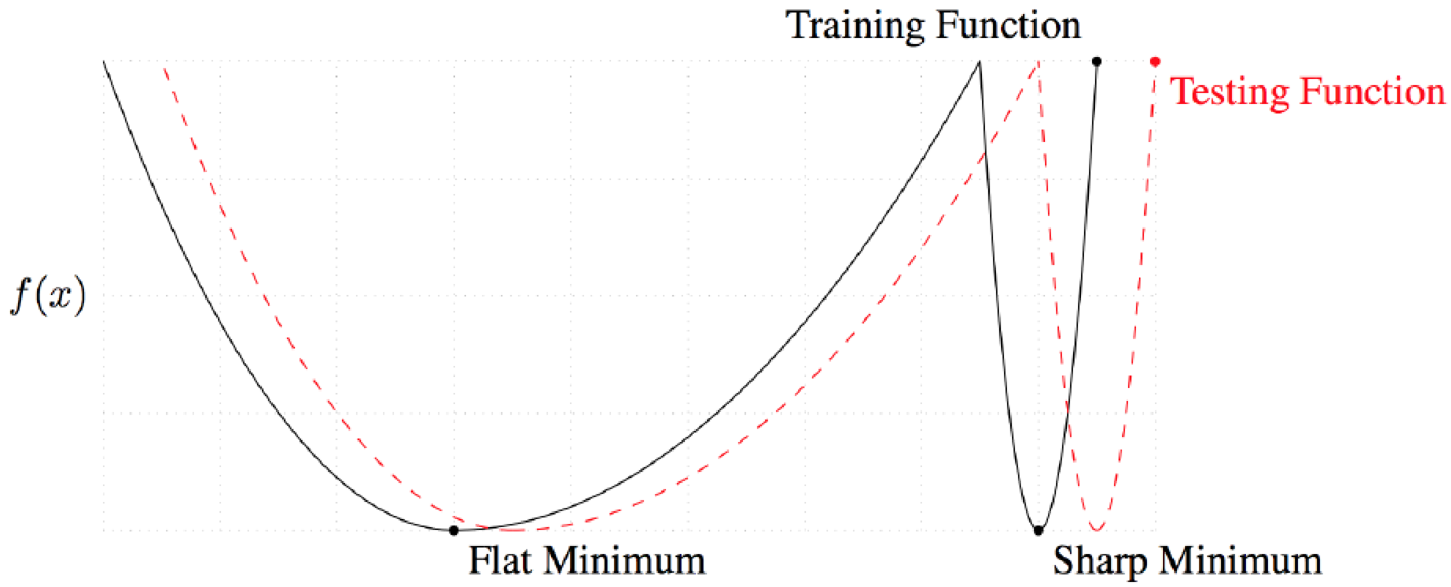
Bayesian Model Construction (Averaging)

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



Bayesian Model Averaging (BMA) in DNN

- Gradient Descent Weight Optimization



Keskar et. al, ICLR 2017.
On Large-Batch Training for Deep Learning: Generalization Gap and Sharp Minima.

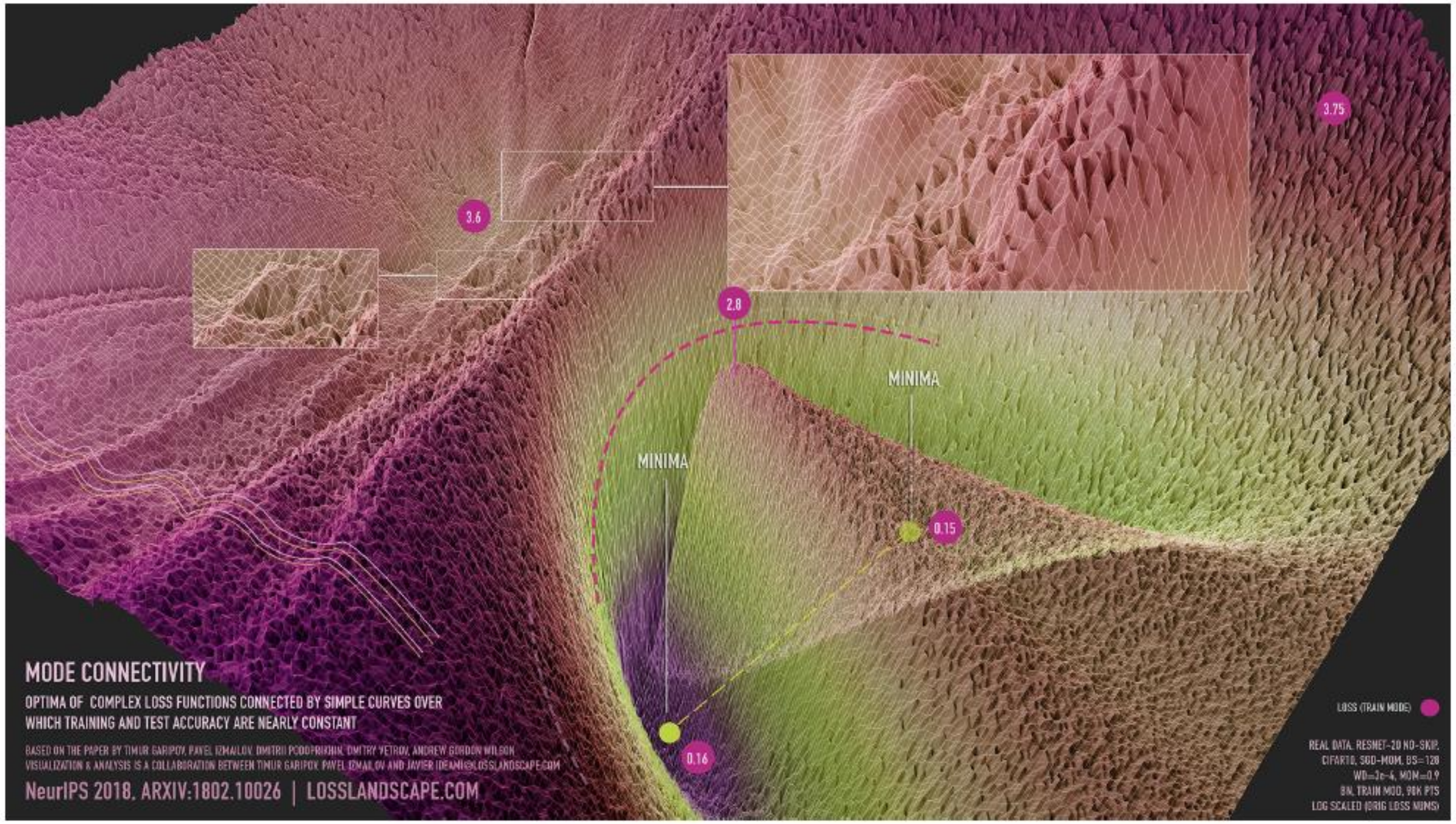
Understanding Loss Surfaces for BMA

Recall the *Bayesian model average* (BMA):

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

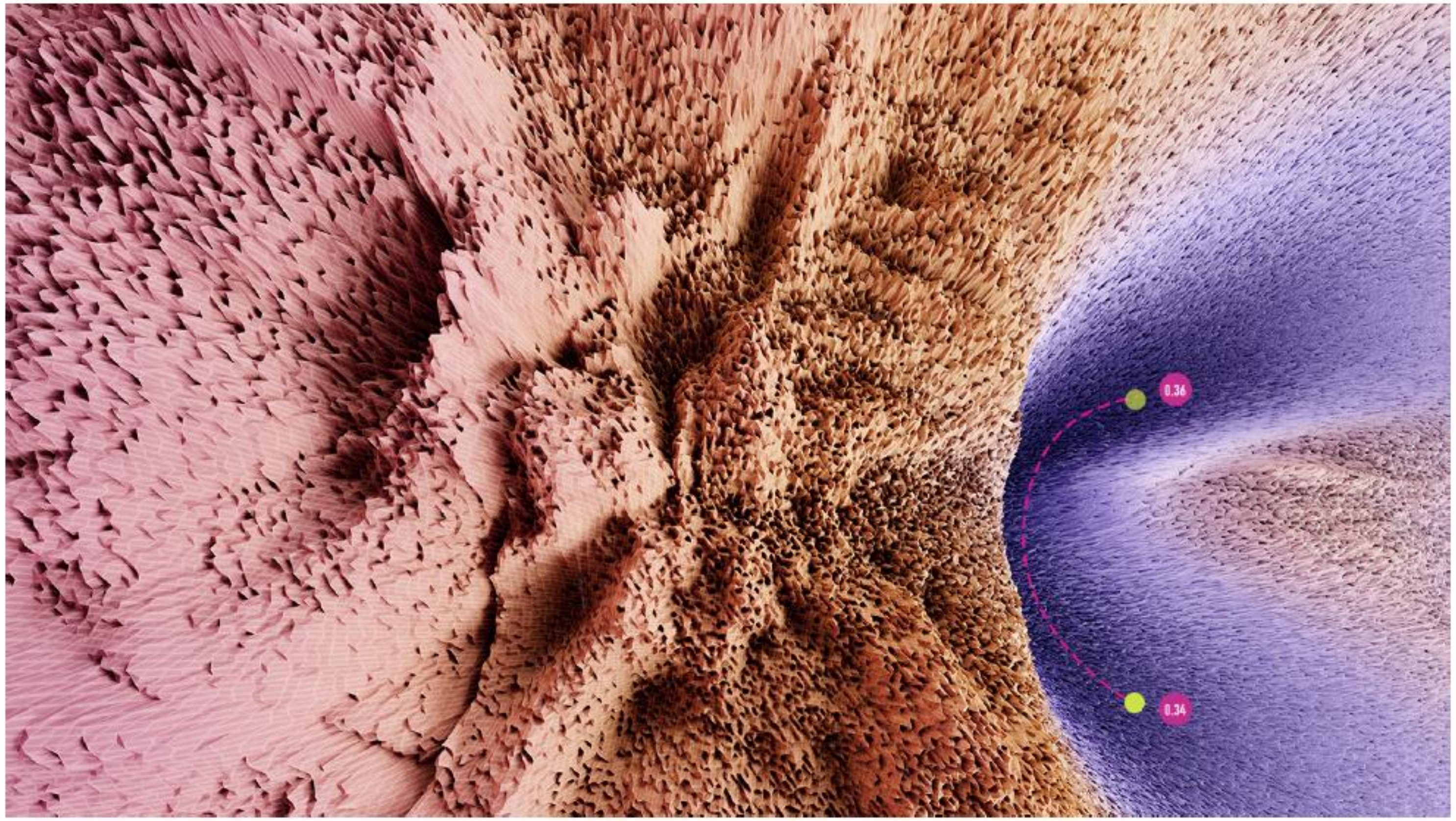
- The posterior $p(w|\mathcal{D})$ (or loss $\mathcal{L} = -\log p(w|\mathcal{D})$) for neural networks is extraordinarily complex, containing many complementary solutions, which is why BMA is *especially* significant in deep learning.
- Understanding the structure of neural network loss landscapes is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

Mode Connectivity



Loss Surfaces, Mode Connectivity, and Fast Ensembling of DNNs.
T. Garipov, P. Izmailov, D. Podoprikhin, D. Vetrov, A.G. Wilson. NeurIPS 2018.

Mode Connectivity



Stochastic Gradient Descent Trajectories

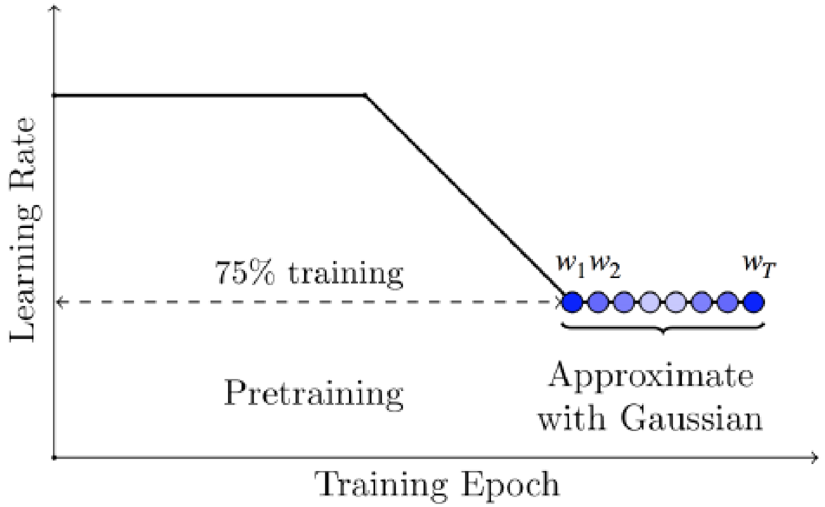
- Consider every point along the SGD trajectory as a candidate DNN model randomly sampled from a posterior distribution over the possible models

- Points on the trajectory in a flat region of the error landscape can be averaged (SWA)

- Bayesian marginalization might replace simple averaging assuming a Gaussian prior over the weights (SWAG)

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.

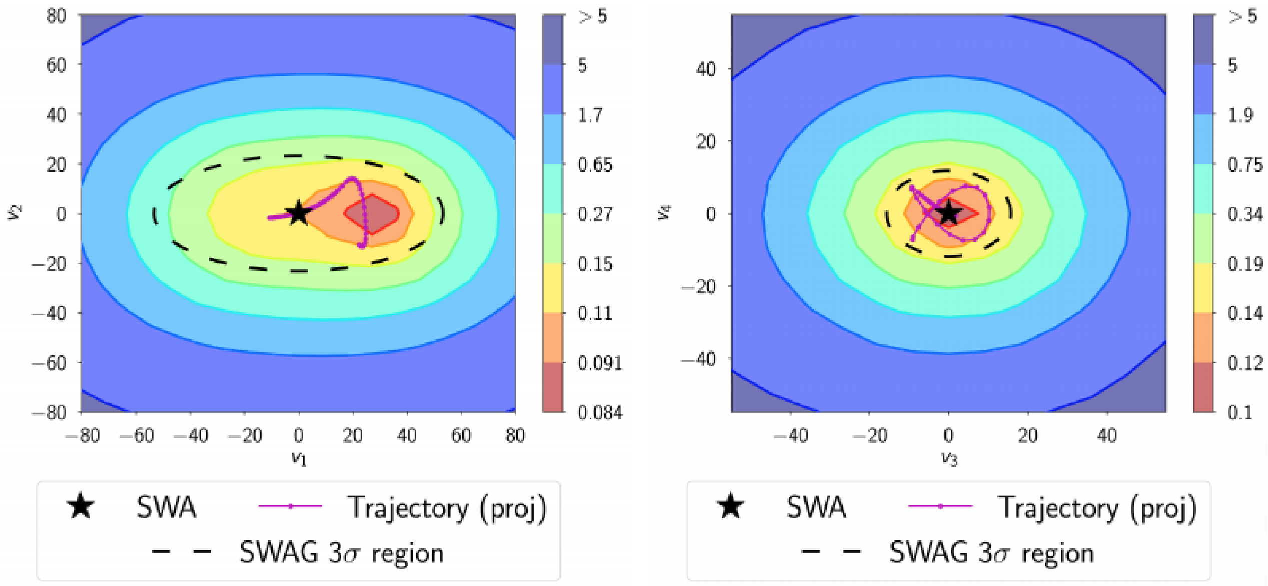


$$p(y_*|\mathcal{D}) \approx \frac{1}{J}\sum_{j=1}^{J} p(y_*|w_j)\,, \quad w_j \sim q(w|\mathcal{D})\,, \quad q(w|\mathcal{D}) = \mathcal{N}(\bar{w}, K)$$

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

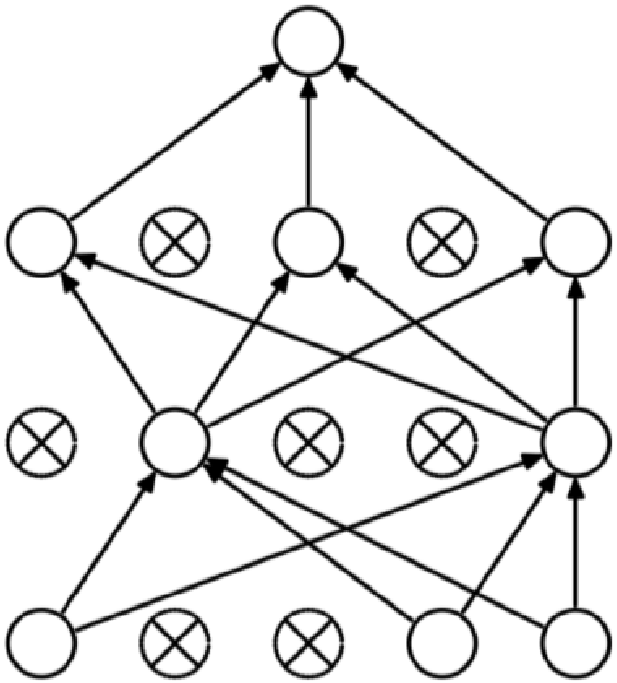
SWAG: *A Simple Baseline for Bayesian Uncertainty in Deep Learning*. Maddox et. al, NeurIPS 2019.
SWA: *Averaging Weights Leads to Wider Optima and Better Generalization*. Izmailov et. al, UAI 2018.

Trajectory in PCA Subspace



Markov Chain 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

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 Generalzation*. Zhang et. al, ICLR 2016.
[3] *Bayesian Deep Learning and a Probabilistic Perspective of Generalization*. Wilson & Izmailov, 2020.

Summary

- 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.
- Bayesian methods are now often providing better results than classical training, in accuracy and uncertainty representation, without significant overhead.
- We can resolve several mysterious results in deep learning by thinking about model construction and generalization from a probabilistic perspective.

Programming Bayesian Learning

- PyMC3 is a Python package for Bayesian statistical modeling and Probabilistic Machine Learning focusing on advanced MCMC and VI.
- ArviZ is a Python package for exploratory analysis of Bayesian models. Includes functions for posterior analysis, data storage, model checking, comparison and diagnostics.

- TensorFlow Probability is a library for probabilistic reasoning and statistical analysis. As part of the TensorFlow ecosystem, TensorFlow Probability provides integration of probabilistic methods with deep networks, gradient-based inference using automatic differentiation, and scalability to large datasets and models with hardware acceleration (GPUs).