Probabilistic Modeling for Deep Learning

CS698X: Topics in Probabilistic Modeling and Inference

Piyush Rai

(Deep) Neural Networks

- These are nonlinear function approximators
- Consists of an input layer, one or more hidden layers, and an output layer

Can think of the last hidden layer's node values being used as features in a GLM (linear/logistic/softmax, etc) modeled by the output layer

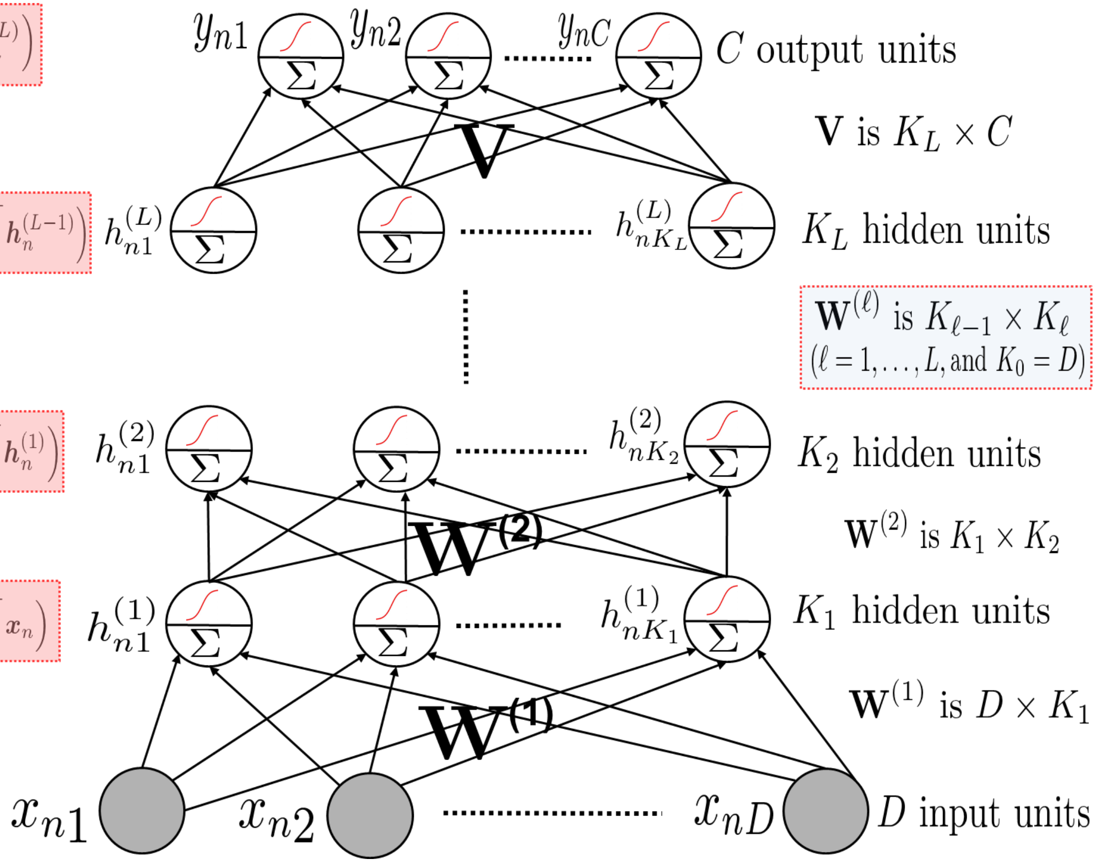
$\boldsymbol{y}_n = o\left(\mathbf{V}^\top \boldsymbol{h}_n^{(L)}\right)$

$\boldsymbol{h}_n^{(L)} = g\left({\mathbf{W}^{(L)}}^\top \boldsymbol{h}_n^{(L-1)}\right)$

Hidden layers act as feature extractors

$\boldsymbol{h}_n^{(2)} = g\left({\mathbf{W}^{(2)}}^\top \boldsymbol{h}_n^{(1)}\right)$

$\boldsymbol{h}_n^{(1)} = g\left({\mathbf{W}^{(1)}}^\top \boldsymbol{x}_n\right)$



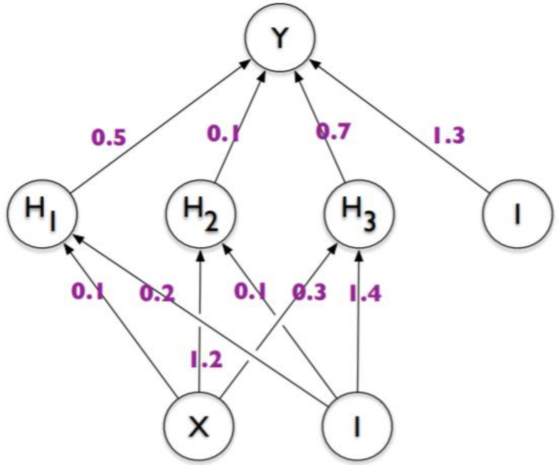
Network weights typically learned by backpropagation (basically, gradient descent + chain rule)

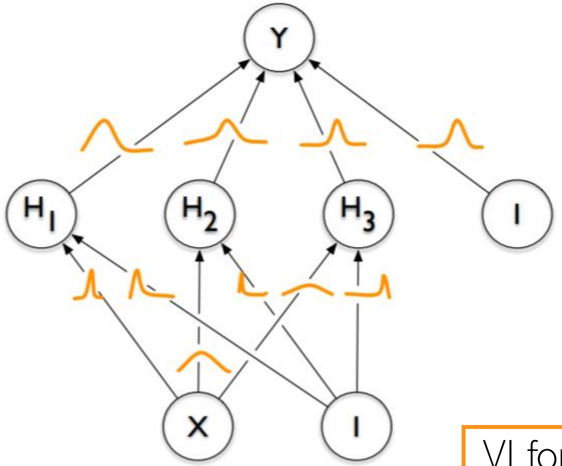
Bayesian Neural Networks

- Backprop for neural nets only gives us point estimates for the weights
- Another alternative is to be Bayesian and learn the posterior distribution over weights

Standard neural net: Each weight has a fixed value, learned by backprop

Note: Just having a likelihood and prior will still give us a standard neural net if we choose to do MLE/MAP only





Bayesian neural net: Each weight has a posterior distribution inferred by some Bayesian inference algo (VI/MCMC/Laplace approx., etc)

Also, test time will require computing PPD, not just a plug-in prediction

$$
\begin{aligned}
\mathbf{w}^{\mathrm{MLE}} &= \arg\max_{\mathbf{w}} \log P(\mathcal{D}|\mathbf{w}) \\
&= \arg\max_{\mathbf{w}} \sum_i \log P(\mathbf{y}_i|\mathbf{x}_i, \mathbf{w}) \\
\mathbf{w}^{\mathrm{MAP}} &= \arg\max_{\mathbf{w}} \log P(\mathbf{w}|\mathcal{D}) \\
&= \arg\max_{\mathbf{w}} \log P(\mathcal{D}|\mathbf{w}) + \log P(\mathbf{w})
\end{aligned}
$$

VI for Bayesian neural net

Using reparametrization trick (known as "Bayes by Backprop"* in this context), BBVI etc

$$
\begin{aligned}
\theta^\star &= \arg\min_{\theta} \mathbf{KL}[q(\mathbf{w}|\theta)||P(\mathbf{w}|\mathcal{D})] \\
&= \arg\min_{\theta} \int q(\mathbf{w}|\theta) \log \frac{q(\mathbf{w}|\theta)}{P(\mathbf{w})P(\mathcal{D}|\mathbf{w})} \mathbf{dw} \\
&= \arg\min_{\theta} \mathbf{KL}\left[q(\mathbf{w}|\theta) \mid\mid P(\mathbf{w})\right] - \mathbb{E}_{q(\mathbf{w}|\theta)}\left[\log P(\mathcal{D}|\mathbf{w})\right]
\end{aligned}
$$

Pic from: *Weight Uncertainty in Neural Networks (Blundell et al, 2015)

A Hybrid Bayesian Neural Net

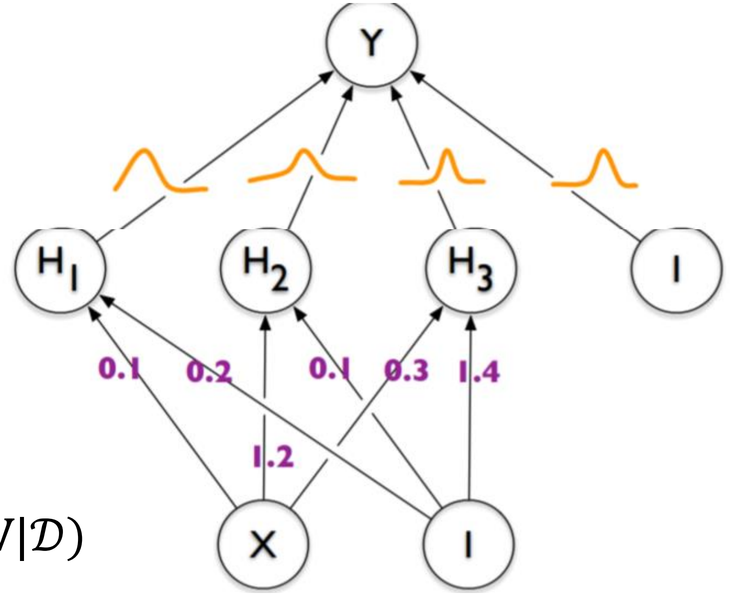
$$p(y_*|x_*, \mathcal{D}) \approx \frac{1}{S}\sum_{s=1}^{S} p(y_*|x_*, \theta^{(s)})$$

where $\theta^{(s)} \sim p(\theta|\mathcal{D})$

- Learning the posterior for all weights can be expensive
- PPD computation is also slow if using Monte Carlo approximation for PPD
- A cheaper practical alternative is
  - Do point estimation for hidden layer weights ($\mathbf{W}$)
  - Infer the full posterior for output layer weights ($\mathbf{V}$)
  - The PPD will then be

Faster because the posterior of $\mathbf{V}$ is much lower dimensional

$$p(y_*|x_*, \mathcal{D}) \approx \frac{1}{S}\sum_{s=1}^{S} p(y_*|x_*, \mathbf{V}^{(s)}, \widehat{\mathbf{W}}) \quad \text{where } \mathbf{V}^{(s)} \sim p(\mathbf{V}|\mathcal{D})$$



- A rough approximation of the above is the following
  - Use a pretrained neural net to extract feature
  - Train Bayesian linear model (e.g., Bayesian linear/logistic/softmax/GLM reg.) on these features

Approximation since in the hybrid approach, we still learn $\mathbf{W}$ and $\mathbf{V}$ together, unlike this approach where it is a two-step process

Bayesian Neural Networks: The Priors

- Zero-mean isotropic Gaussian priors are common and convenient
  - Corresponds to weight-decay or ℓ_2 regularizer

- Another alternative is to use sparsity-inducing priors, e.g.,

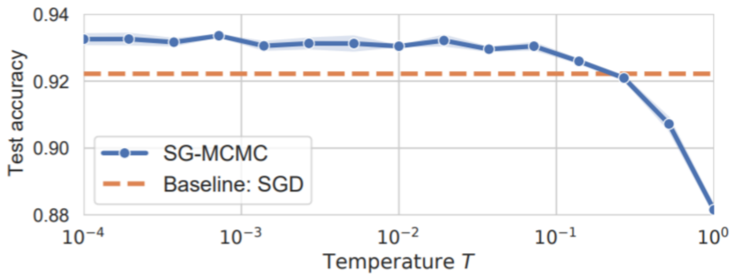
$$p(\mathbf{w}) = \prod_j \pi \mathcal{N}(w_j|0, \sigma_1^2) + (1-\pi)\mathcal{N}(w_j|0, \sigma_2^2) \qquad \sigma_1 > \sigma_2 \text{ and } \sigma_2 \ll 1$$

- Gaussian priors have been found somewhat problematic in recent work
  - Cold-posterior effect

T is like temperature

$T = 1$ is the standard Bayesian inference

$$\log p(w|x,y)^{\frac{1}{T}} = \frac{1}{T}[\log p(y|w,x) + \log p(w)] + Z(T)$$



Recent work has shown that BNNs with standard Gaussian priors work poorly for $T = 1$ but $T \ll 1$ improves performance

Maybe Gaussian priors aren't really ideal??

Pic from: *How Good is the Bayes Posterior in Deep Neural Networks Really? (Wenzel et al, 2020)

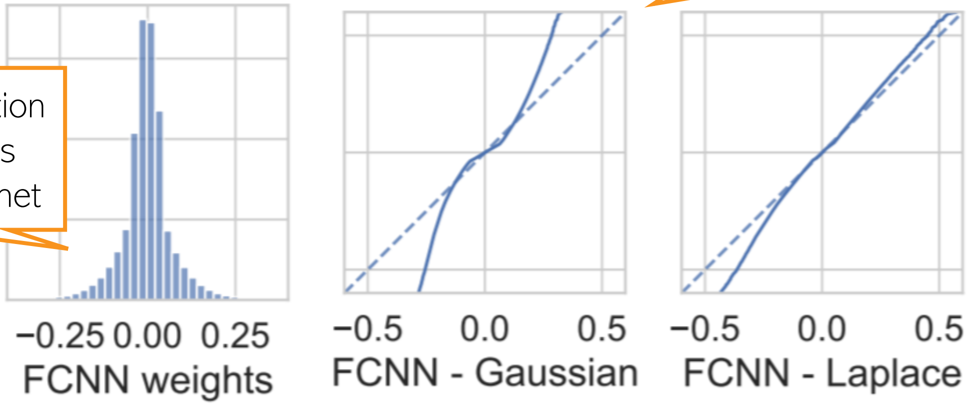
Bayesian Neural Networks: The Priors

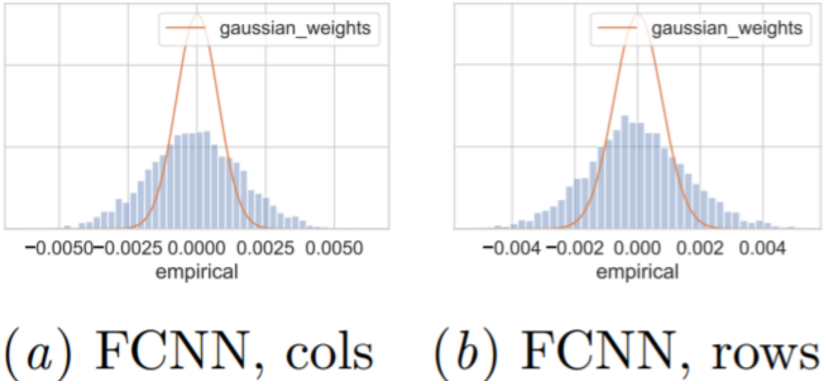
- Deep neural net weights are empirically found to have
  - Heavy-tailed distributions*
  - Correlations among weights*

Q-Q plot shows that Laplace (a heavy-tailed distribution) is closer to than Gaussian

Distribution of off-diagonal elements in the empirical covariance matrix of the weights

Empirical distribution of learned weights of a deep neural net





- Thus heavy-tailed priors (e.g., Laplace, student-t) tend to do better than iso Gaussian
- Gaussian prior with non-diagonal covariance can help model correlations
- Designing better priors for Bayesian deep neural nets is still an open problem

Pic from: *Bayesian Neural Network Priors Revisited (Fortuin et al, 2021)

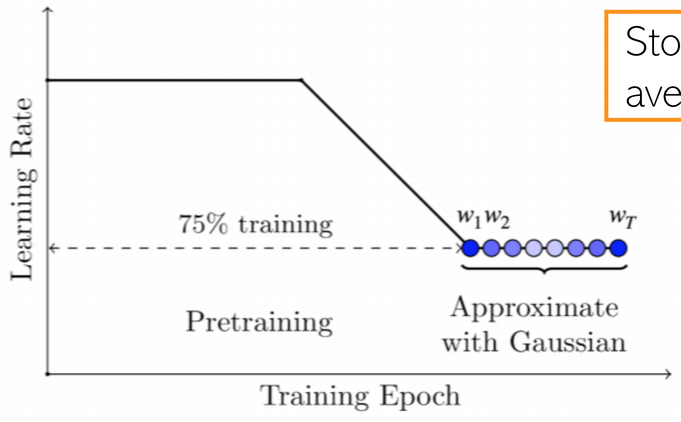
Other Inference Methods for Bayesian Neural Nets

- Laplace approximation is very common: $p(W|\mathcal{D}) \approx \mathcal{N}(W_{MAP}, \mathbf{H}^{-1})$
  - However, can be slow since the number of parameters is very large
  - One option is to use a simpler covariance matrix (e.g,, diagonal or block-diag)
  - Another option is to use the hybrid Bayesian neural net
    - Use MAP estimates for the hidden layer weights
    - Use Laplace approximation only for the output layer weights

Extension: A mixture of Gaussian approximation: Multi-SWAG – Run SGD M times and use a mixture of M such Gaussians

SWA based Gaussian approximation: SWAG

- Using SGD iterates obtained from backprop



Stochastic weight averaging (SWA)

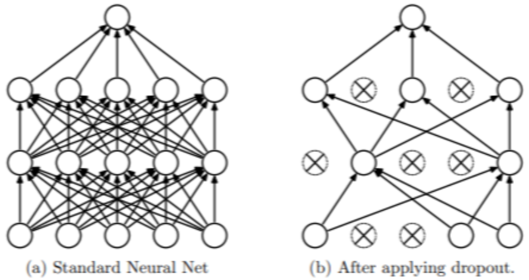
$$p(w|\mathcal{D}) \approx q(w|\mathcal{D}) = \mathcal{N}(\bar{w}, K)$$

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

Pic from: *A Simple Baseline for Bayesian Uncertainty in Deep Learning (Maddox et al, 2019)

Other Inference Methods for Bayesian Neural Nets

- Monte Carlo Dropout is another popular and efficient way



- Standard Dropout
  - Drop some weights randomly (with some "drop" probability) during training
  - At test time, multiply each weight by the "keep" probability
  - Note: Dropout applied only at training time

- Monte Carlo Dropout*

$$p(y_*|x_*,\mathcal{D}) \approx \frac{1}{S}\sum_{s=1}^{S} p(y_*|x_*,\theta^{(s)})$$

where $\theta^{(s)} \sim p(\theta|\mathcal{D})$

Can be seen as learning a variational approximation of the weights (see paper for details, if interested)

$$p(y_*|x_*,\mathcal{D}) \approx \frac{1}{S}\sum_{s=1}^{S} p(y_*|x_*,\theta^{(s)})$$

where $\theta^{(s)} = \epsilon^{(s)} \odot \hat{\theta}$

Vector of Bernoulli or Gaussian noise

Elementwise product

Point estimate

*Dropout as a Bayesian Approximation: Representing Model Uncertainty in Deep Learning (Gal and Ghahramani, 2016)

Other Inference Methods for Bayesian Neural Nets

- SGMCMC methods like SGLD and SGHMC are also used nowadays (very efficient)

$$\theta^{(t)} = \theta^{(t-1)} + \frac{\eta_t}{2}\nabla_\theta[\log p(\mathcal{D}|\theta) + \log p(\theta)]\big|_{\theta^{(t-1)}} + \epsilon_t$$

- Recently, SGMCMC with cyclic step sizes (cSGLD) was proposed (Zhang et al, 2020)
  - Use big steps to explore different modes
  - Use small steps later to sample once a mode is localized

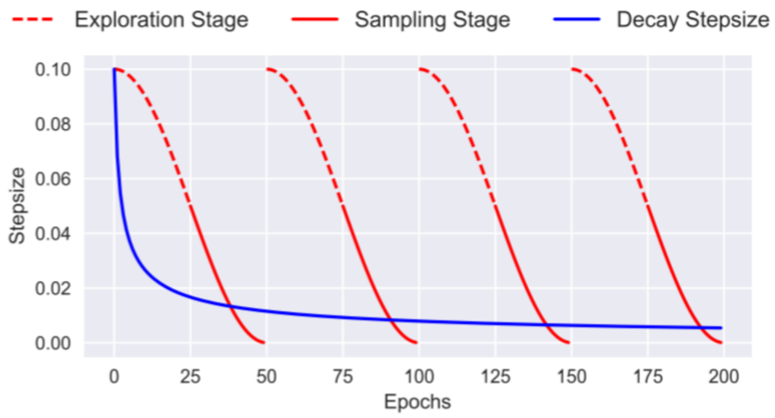
Step size in iteration k

$$\alpha_k = \frac{\alpha_0}{2}\left[\cos\left(\frac{\pi \bmod(k-1, \lceil K/M \rceil)}{\lceil K/M \rceil}\right) + 1\right]$$

K is the total number of iterations and M is the number of cycles

(a) Target (b) SGLD (c) cSGLD

A complex mixture of Gaussian distributions



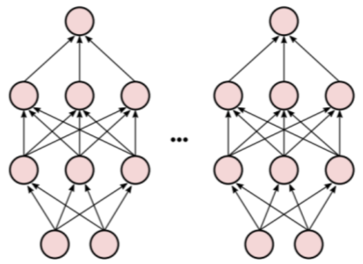
	CIFAR-10	CIFAR-100
SGD	5.29±0.15	23.61±0.09
SGDM	5.17±0.09	22.98±0.27
Snapshot-SGD	4.46±0.04	20.83±0.01
Snapshot-SGDM	4.39±0.01	20.81±0.10
SGLD	5.20±0.06	23.23±0.01
cSGLD	4.29±0.06	20.55±0.06
SGHMC	4.93±0.1	22.60±0.17
cSGHMC	**4.27**±0.03	**20.50**±0.11

Pic from: *Cyclical Stochastic Gradient MCMC for Bayesian Deep Learning (Zhang et al, 2020)

Deep Ensembles

Both VI and Sampling may be prone to capturing only a single "Basin of attraction"

- Most inference methods tend to produce local approximations only
  - VI methods typically learn an approximation around one of the modes
  - Sampling methods may give most samples near one of the modes (though in principle they may explore other modes as well)
  - Thus the uncertainties may be underestimated in general



- Deep Ensembles* is a method that tries to address this issue
  - Train the network M times with different seeds and permutations of training data
  - Denote the learned weights by $\theta_1, \theta_2, \ldots, \theta_M$ (assuming these are M modes)
  - Approximate the posterior by the following

$$p(\theta|\mathcal{D}) = \frac{1}{M}\sum_{m=1}^{M} \delta_{\theta_m}(\theta)$$

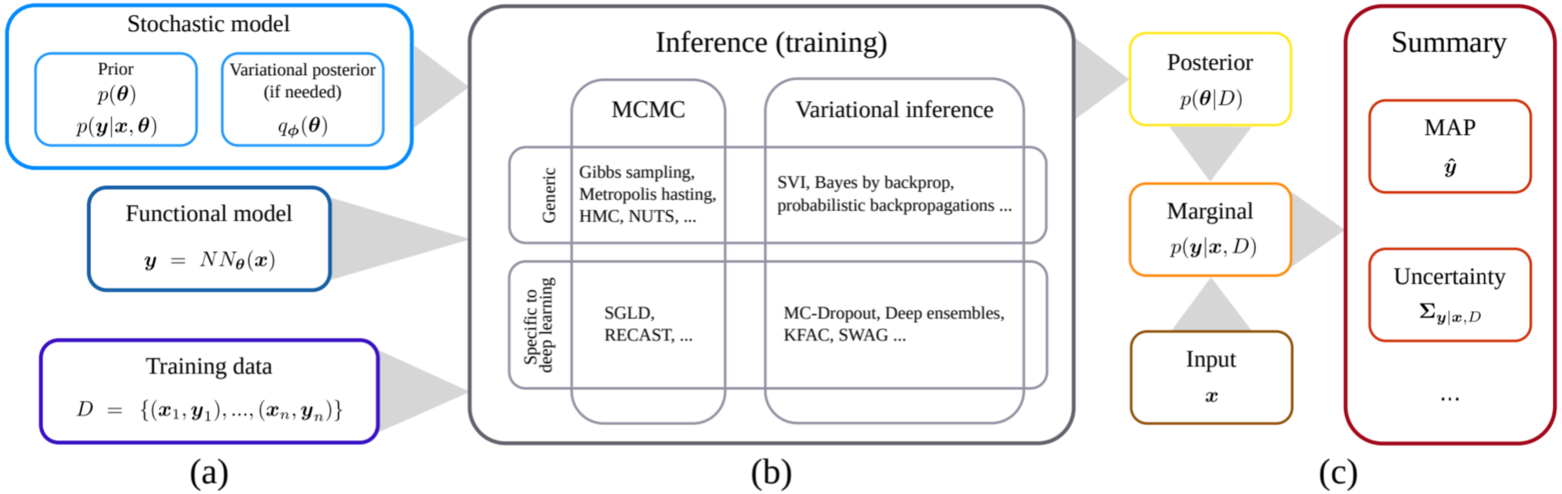
Akin to Bayesian Model Averaging using M models

  - This approach is considered non-Bayesian but often performs better (in terms of more diversity in the set of parameters learned) than other inference methods

*Simple and Scalable Predictive Uncertainty Estimation using Deep Ensembles (Lakshminarayanan, 2017)

Some Comments/Summary

- Bayesian neural networks can be useful for getting uncertainty estimates in deep learning models
- A summary of the pipeline/building blocks



- A lot of recent progress in this area
  - Better architectures and priors
  - Better inference methods
  - Connections with other methods (e.g., kernel methods and Gaussian Processes)

Pic source: Hands-on Bayesian Neural Networks - a Tutorial for Deep Learning Users (Jospin et al, 2020)

Coming Up Next

- Deep Generative Models for Unsupervised Learning