Approx. Inference via Sampling (wrap-up), Bayesian Deep Learning

CS772A: Probabilistic Machine Learning

Piyush Rai

Hamiltonian/Hybrid Monte Carlo (HMC)

- HMC (Neal, 1996) is an "auxiliary variable sampler" and incorporates gradient info
- Uses the idea of simulating a Hamiltonian Dynamics of a physical system
- Consider the target posterior $p(\theta|\mathcal{D}) \propto \exp(-U(\theta))$
- Think of $\boldsymbol{\theta}$ as "position" then $U(\theta) = -\log p(\mathcal{D}|\theta)p(\theta)$ is like "potential energy"
- Let's introduce an auxiliary variable - the momentum $\boldsymbol{r}$ of the system
- Can now define a joint distribution over the position and momentum as

$$p(\theta, \boldsymbol{r}) \propto \exp\left(-U(\theta) - \frac{1}{2}\boldsymbol{r}^\top M^{-1}\boldsymbol{r}\right) = p(\theta|\mathcal{D})p(\boldsymbol{r})$$

- The total energy (potential + kinetic) or the Hamiltonian of the system

Constant w.r.t. time

$$H(\theta, \boldsymbol{r}) = U(\theta) + \tfrac{1}{2}\boldsymbol{r}^\top M^{-1}\boldsymbol{r} = U(\theta) + K(\boldsymbol{r})$$

- Given a sample $(\theta, \boldsymbol{r})$ from $p(\theta, \boldsymbol{r})$, ignoring $\boldsymbol{r}$, θ will be a sample from $p(\theta|\mathcal{D})$

Generating Samples in HMC

- Given an initial $(\boldsymbol{\theta}, \boldsymbol{r})$, Hamiltonian Dynamics defines how $(\boldsymbol{\theta}, \boldsymbol{r})$ changes w.r.t. time $\boldsymbol{t}$

$$\frac{\partial \theta}{\partial t} = \frac{\partial H}{\partial \mathbf{r}} = \frac{\partial K}{\partial \mathbf{r}}$$
$$\frac{\partial \mathbf{r}}{\partial t} = -\frac{\partial H}{\partial \theta} = -\frac{\partial U}{\partial \theta}$$

$$(H(\theta, \mathbf{r}) = U(\theta) + \tfrac{1}{2}\mathbf{r}^\top M^{-1}\mathbf{r} = U(\theta) + K(\mathbf{r}))$$

- We can use these equations to update $(\boldsymbol{\theta}, \boldsymbol{r}) \rightarrow (\boldsymbol{\theta}^*, \boldsymbol{r}^*)$ by discretizing time

Reason: Getting analytical solutions for the above requires integrals which is in general intractable

- For $\boldsymbol{s} = 1{:}\boldsymbol{S}$, sample as follows
  - Initialize $\theta_0 = \theta^{(s-1)}$, $\mathbf{r}_* \sim \mathcal{N}(0, \mathbf{I})$ and $\mathbf{r}_0 = \mathbf{r}_* - \frac{\rho}{2}\frac{\partial U}{\partial \theta}|_{\theta_0}$
  - Do $\boldsymbol{L}$ "leapfrog" steps with learning rates $\boldsymbol{\rho_\ell = \rho}$ for $\boldsymbol{\ell < L}$ and $\boldsymbol{\rho_L = \rho/2}$
    - For $\boldsymbol{\ell = 1{:}L}$

$$\theta_\ell = \theta_{\ell-1} + \rho\frac{\partial K}{\partial \mathbf{r}}|_{\mathbf{r}_{\ell-1}}$$
$$\mathbf{r}_\ell = \mathbf{r}_{\ell-1} - \rho_\ell\frac{\partial U}{\partial \theta}|_{\theta_\ell}$$

$\boldsymbol{L}$ usually set to 5 and learning rate tuned to make acceptance rate around 90%

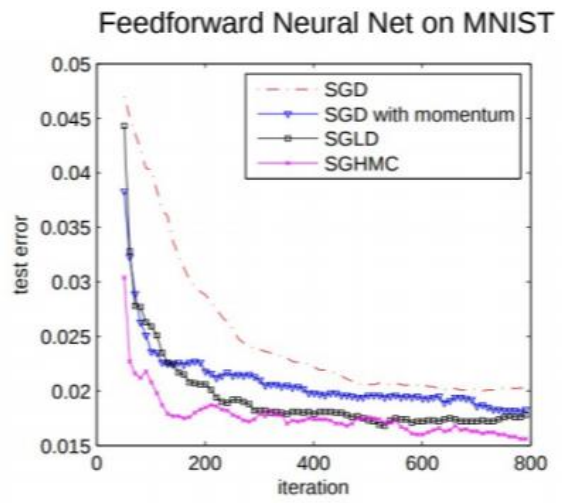
  - Perform MH accept/reject test on $(\boldsymbol{\theta_L}, \boldsymbol{r_L})$. If accepted $\boldsymbol{\theta^{(s)} = \theta_L}$

A single sample generated by taking $\boldsymbol{L}$ steps

- The momentum forces exploring different regions instead of getting driven to regions where the MAP solution is

HMC in Practice

- HMC typically has very low rejection rate (that too, primarily due to discretization error)
- Performance can be sensitive to L (no. of leapfrog steps) and step-sizes, tuning hard
- A lot of renewed interest in HMC (you may check out NUTS - No U-turn Sampler – doesn't require setting L)
  - Prob. Prog. packages e.g., Tensorflow Prob., Stan, etc, contain implementations of HMC
- Can also do HMC on minibatches (Stochastic Gradient HMC - Chen et al, 2014)
- An illustration: SGHMC vs other methods on MNIST classification (Bayesian neural net)



(Figure: Stochastic Gradient Hamiltonian Monte Carlo (Chen et al, 2014))

Parallel/Distributed MCMC

- Suppose our goal is to compute the posterior of $\theta \in \mathbb{R}^D$ (assuming N is very large)

$$p(\theta|\mathbf{X}) \propto p(\theta)p(\mathbf{X}|\theta) = p(\theta)\prod_{n=1}^{N} p(\mathbf{x}_n|\theta)$$

- Suppose we have J machines with data partitioned as $\mathbf{X} = \{\mathbf{X}^{(j)}\}_{j=1}^{J}$
- Let's assume that the posterior $p(\theta|\mathbf{X})$ factorizes as

$$p(\theta|\mathbf{X}) = \prod_{i=1}^{J} p^{(j)}(\theta|\mathbf{X}^{(j)})$$

- Here $p^{(j)}(\theta|\mathbf{X}^{(j)}) \propto p(\theta)^{1/J} \prod_{\mathbf{x}_n \in \mathbf{X}^{(j)}} p(\mathbf{x}_n|\theta)$ is known as the "subset posterior"
- Assume the j^{th} machine generates T MCMC samples $\{\theta_{j,t}\}_{t=1}^{T}$
- We need a way to combine these subset posteriors using a "consensus"

$$\hat{\theta}_1, \ldots, \hat{\theta}_T = \texttt{CONSENSUSSAMPLES}(\{\theta_{j,1}, \ldots, \theta_{j,T}\}_{j=1}^{J})$$

Parallel/Distributed MCMC

- Many ways to compute the consensus samples. Let's look at two of them
- Approach 1: Weighted Average: $\hat{\theta}_t = \sum_{j=1}^{J} W_j \theta_{j,t}$ where W_j can be learned as follows
  - Assuming Gaussian likelihood and Gaussian prior

$$\begin{aligned} \bar{\Sigma}_j &= \text{sample covariance of } \{\theta_{j,1}, \ldots, \theta_{j,T}\} \\ \Sigma &= (\Sigma_0^{-1} + \sum_{j=1}^{J} \bar{\Sigma}_j^{-1})^{-1} \quad (\Sigma_0 \text{ is the prior's covariance}) \\ W_j &= \Sigma(\Sigma_0^{-1}/J + \bar{\Sigma}_j^{-1}) \end{aligned}$$

These approaches can also be used to make VI parallel/distributed

- Approach 2: Fit J Gaussians, one for each $\{\theta_{j,t}\}_{t=1}^{T}$ and take their product

$$\begin{aligned} \bar{\mu}_j &= \text{sample mean of } \{\theta_{j,1}, \ldots, \theta_{j,T}\}, \quad \bar{\Sigma}_j = \text{sample covariance of } \{\theta_{j,1}, \ldots, \theta_{j,T}\} \\ \hat{\Sigma}_J &= (\sum_{j=1}^{J} \bar{\Sigma}_j^{-1})^{-1}, \quad \hat{\mu}_J = \hat{\Sigma}_J(\sum_{j=1}^{J} \bar{\Sigma}_j^{-1} \bar{\mu}_j) \quad (\text{cov and mean of prod. of Gaussians}) \\ \hat{\theta}_t &\sim \mathcal{N}(\hat{\mu}_J, \hat{\Sigma}_J), t = 1, \ldots, T \quad (\text{the final consensus samples}) \end{aligned}$$

- For detailed proofs and other approaches, may refer to the reference below

Patterns of Scalable Bayesian Inference (Angelino et al, 2016)

Approximate Inference: VI vs Sampling

- VI approximates a posterior distribution $\boldsymbol{p(Z|X)}$ by another distribution $\boldsymbol{q(Z|\phi)}$
- Sampling uses $\boldsymbol{S}$ samples $\boldsymbol{Z^{(1)}, Z^{(2)}, \dots, Z^{(S)}}$ to approximate $\boldsymbol{p(Z|X)}$
- Sampling can be used within VI (ELBO approx using Monte-Carlo)
- In terms of "comparison" between VI and sampling, a few things to be noted
  - Convergence: VI only has local convergence, sampling (in theory) can give exact posterior
  - Storage: Sampling based approx needs to storage all samples, VI only needs var. params $\boldsymbol{\phi}$
  - Prediction Cost: Sampling always requires Monte-Carlo avging for posterior predictive; with VI, sometimes we can get closed form posterior predictive

PPD if using sampling: $p(x_*|X) = \int p(x_*|Z)p(Z|X)dZ \approx \frac{1}{S}\sum_{s=1}^{S} p(x_*|Z^{(s)})$

PPD if using VI: $p(x_*|X) = \int p(x_*|Z)p(Z|X)dZ \approx \int p(x_*|Z)q(Z|\phi)dZ$

Compressing the S samples into something more compact

- There is some work on "compressing" sampling-based approximations*

*"Compact approximations to Bayesian predictive distributions" by Snelson and Ghaharamani, 2005; and "Bayesian Dark Knowledge" by Korattikara et al, 2015

Inference Methods: Summary

- MLE/MAP: Straightforward for differentiable models (can even use automatic diff.)
- Conjugate models with one “main” parameter: Straightforward posterior updates
- MLE-II/MAP-II: Often useful for estimating the hyperparameters
- EM: If we want to do MLE/MAP for models with latent variables
  - Very general algorithm, can also be made online
  - Used when we want point estimates for some unknowns and posterior over others
  - Can use it for hyperparameter estimation as well
  - Often better than using direct gradient methods
- VI and sampling methods can be used to get full posterior for complex models
  - Quite easy if we have local conjugacy (VI has closed form updates, Gibbs sampler is easy to derive)
  - In other cases, we have general VI with Monte-Carlo gradients, MH sampling
  - MCMC can also make use of gradient info (LD/SGLD)
- For large-scale problems, online/distributed VI/MCMC, or SGD based posterior approx

(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

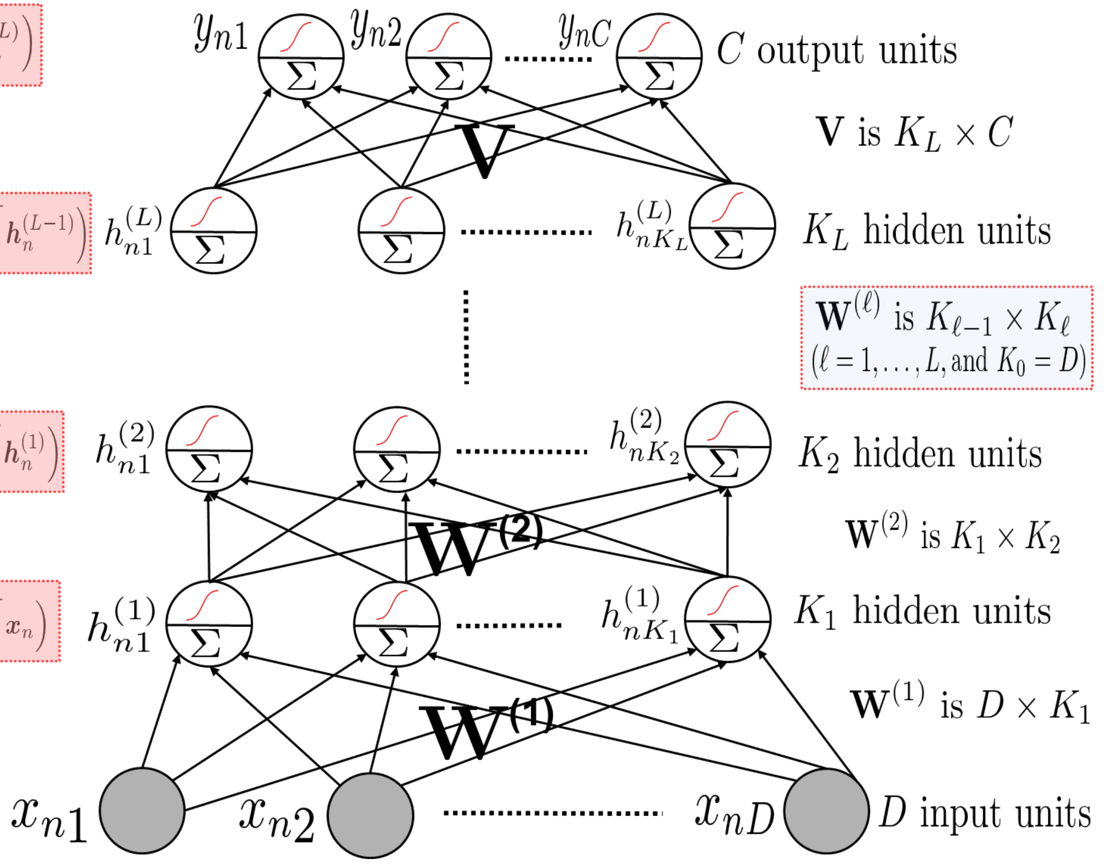
$y_n = o\left(\mathbf{V}^\top h_n^{(L)}\right)$

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

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

$h_n^{(1)} = g\left({\mathbf{W}^{(1)}}^\top x_n\right)$

Hidden layers act as feature extractors



$\mathbf{W}^{(\ell)}$ is $K_{\ell-1} \times K_\ell$
$(\ell = 1, \ldots, L$, and $K_0 = D)$

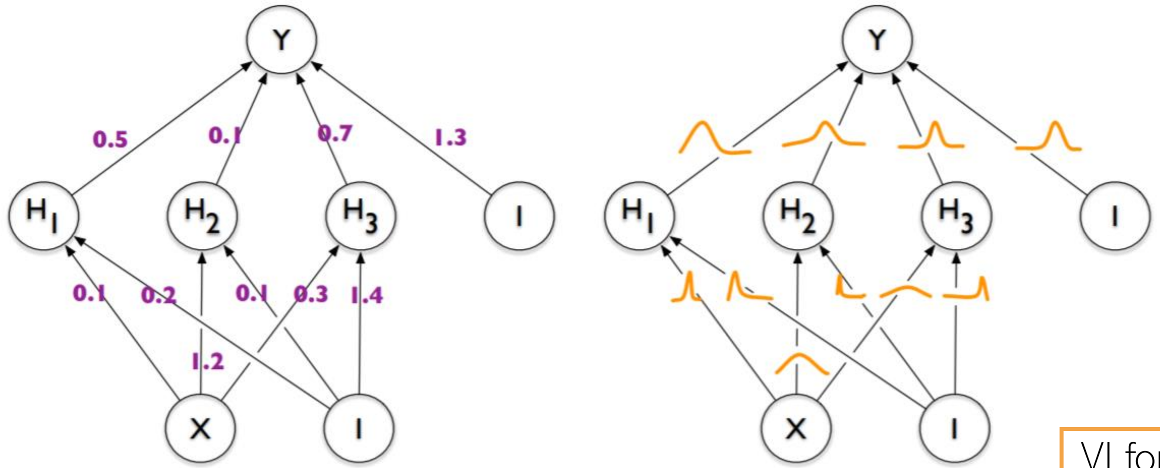
Network weights typically learned by backpropagation (basically, gradient descent + chain rule)

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

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

VI for Bayesian neural net

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

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

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

A Hybrid Bayesian Neural Net

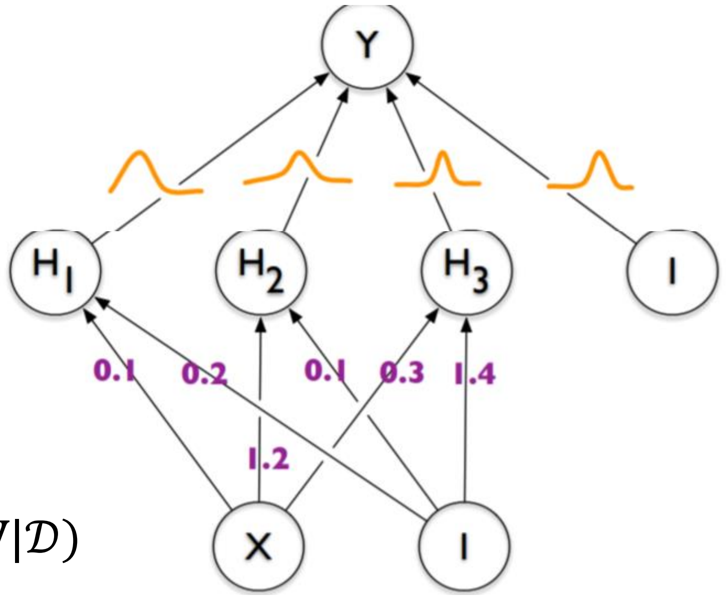
$$p(y_*|x_*,\mathcal{D}) \approx \frac{1}{S}\sum_{s=1}^{S} p(y_*|x_*,\theta^{(s)})$$

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

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



- 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

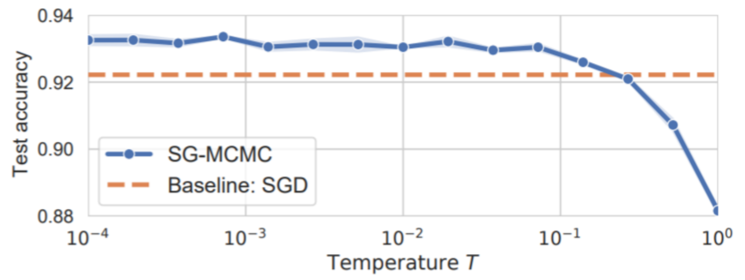
T is like temperature

$T = 1$ 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)

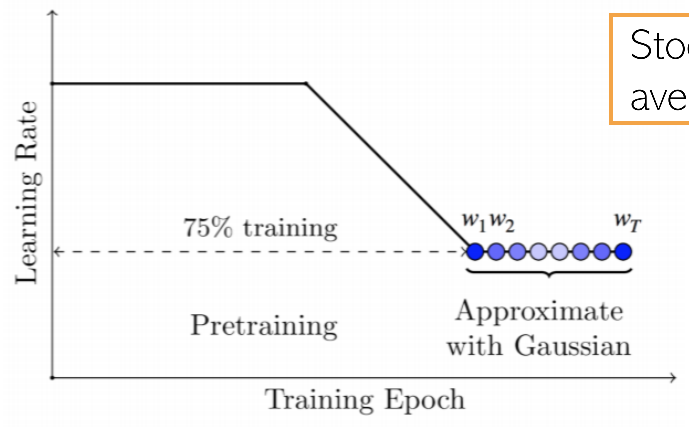
Other Inference Methods for Bayesian Neural Nets

- 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)

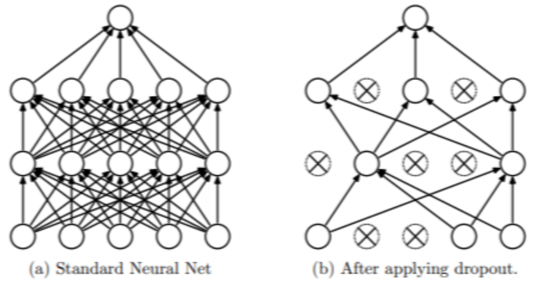
$$p(w|\mathcal{D}) \approx q(w|\mathcal{D}) = \mathcal{N}(\bar{w}, K)$$

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

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)

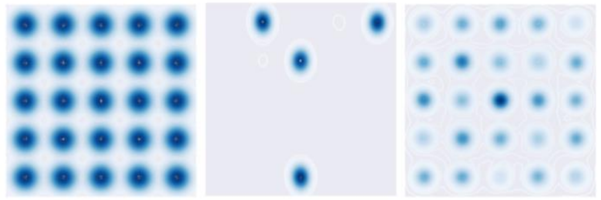
$$\boxed{\theta^{(t)} = \theta^{(t-1)} + \frac{\eta_t}{2}\nabla_\theta[\log p(\mathcal{D}|\theta) + \log p(\theta)]\big|_{\theta^{(t-1)}} + \epsilon_t}$$

- 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

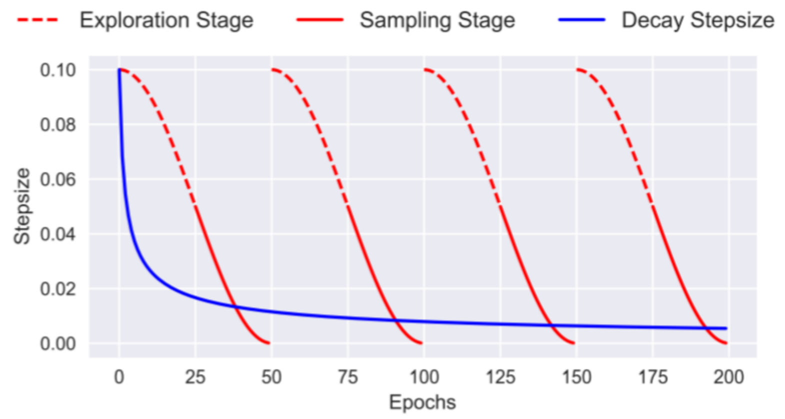
Step size in iteration k

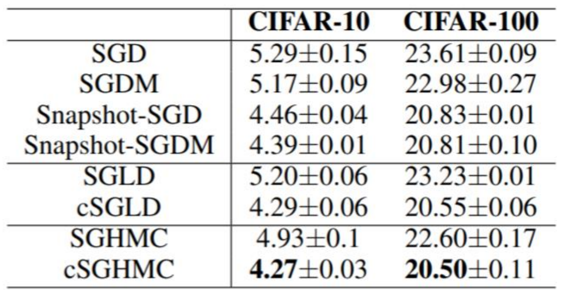
$$\alpha_k = \frac{\alpha_0}{2}\left[\cos\left(\frac{\pi \operatorname{mod}(k-1, \lceil K/M \rceil)}{\lceil K/M \rceil}\right) + 1\right]$$

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

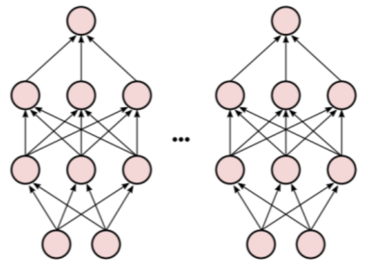
Pic from: *Cyclical Stochastic Gradient MCMC for Bayesian Deep Learning (Zhang et al, 2020)

Deep Ensembles

- 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

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

- 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)