Variational Inference for Bayesian Neural Networks

Jesse Bettencourt, Harris Chan, Ricky Chen, Elliot Creager, Wei Cui, Mohammad Firouzi, Arvid Frydenlund, Amanjit Singh Kainth, Xuechen Li, Jeff Wintersinger, Bowen Xu

October 6, 2017

University of Toronto

Overview

Variational Autoencoders

Kingma and Welling, 2014. Auto-encoding variational Bayes.

Variational Inference for BNNs

Origins of VI: MDL Interpretation

Hinton and van Camp, 1993. Keeping the neural networks simple by minimizing the description length of the weights.

Practical VI for Neural Networks

Graves, 2011. Practical variational inference for neural networks.

Weight Uncertainty in Neural Networks

Blundell et al., 2015. Weight uncertainty in neural networks.

The Local Reparameterization Trick

Kingma, Salimans, and Welling, 2015. Variational dropout and the local reparameterization trick.

Sparsification

Louizos et al., 2017. Bayesian compression for deep learning.

Variational Autoencoders (VAE)

From Autoencoders to Variational Autoencoders

- Autoencoders (AE)
  - Neural network which reconstructs its own inputs, x
  - Learns useful latent representation, z
  - Regularized by bottleneck layer – compresses latent representation
  - Encoder $f(x) \to z$ and decoder $g(z) \to x$
  - Compresses point in input space to point in latent space
- Variational autoencoders (VAE)
  - Regularized by forcing z to be close to some given distribution
  - $z \sim \mathcal{N}(\mu = 0, \sigma^2 = 1)$, with diagonal covariance
  - Learn distribution over latent space
  - Compresses point in input space to distribution in latent space

Implementing a VAE

Three implementation differences between a VAE and an AE

1. Our encoder network parameterizes a probability distribution
   - Normal distribution is parameterized by its means μ and variances σ^2
   - Encoder $f(x) \rightarrow \mu, \sigma^2$
   - Decoder $g(z) \rightarrow x$, where $z \sim \mathcal{N}(\mu, \sigma^2)$
2. Need to sample z
   - **Problem:** Can not backpropagate through sampling z
   - **Solution:** reparameterization trick
     - $z = \mu + \sigma * \epsilon$, where ϵ is a noise input variable and $\epsilon \sim \mathcal{N}(0, 1)$
3. We need to add a new term to the cost function
   - Reconstruction error (log-likelihood)
   - KL divergence between distribution of z and normal distribution
     - KL term acts as regularizer on z

Autoencoders

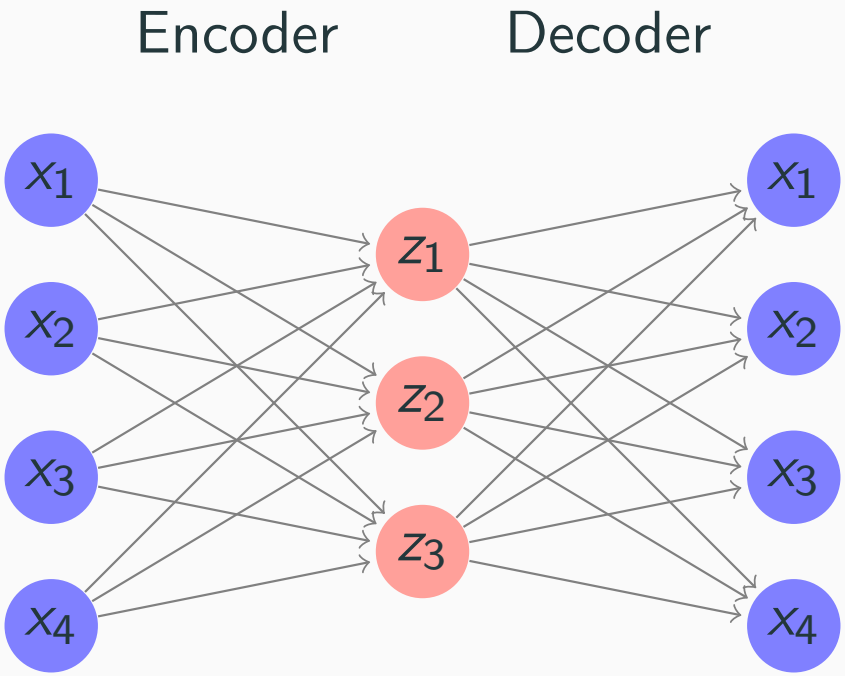


Figure 1: Inputs are shown in blue and the latent representation is shown in red.

Variational Autoencoders

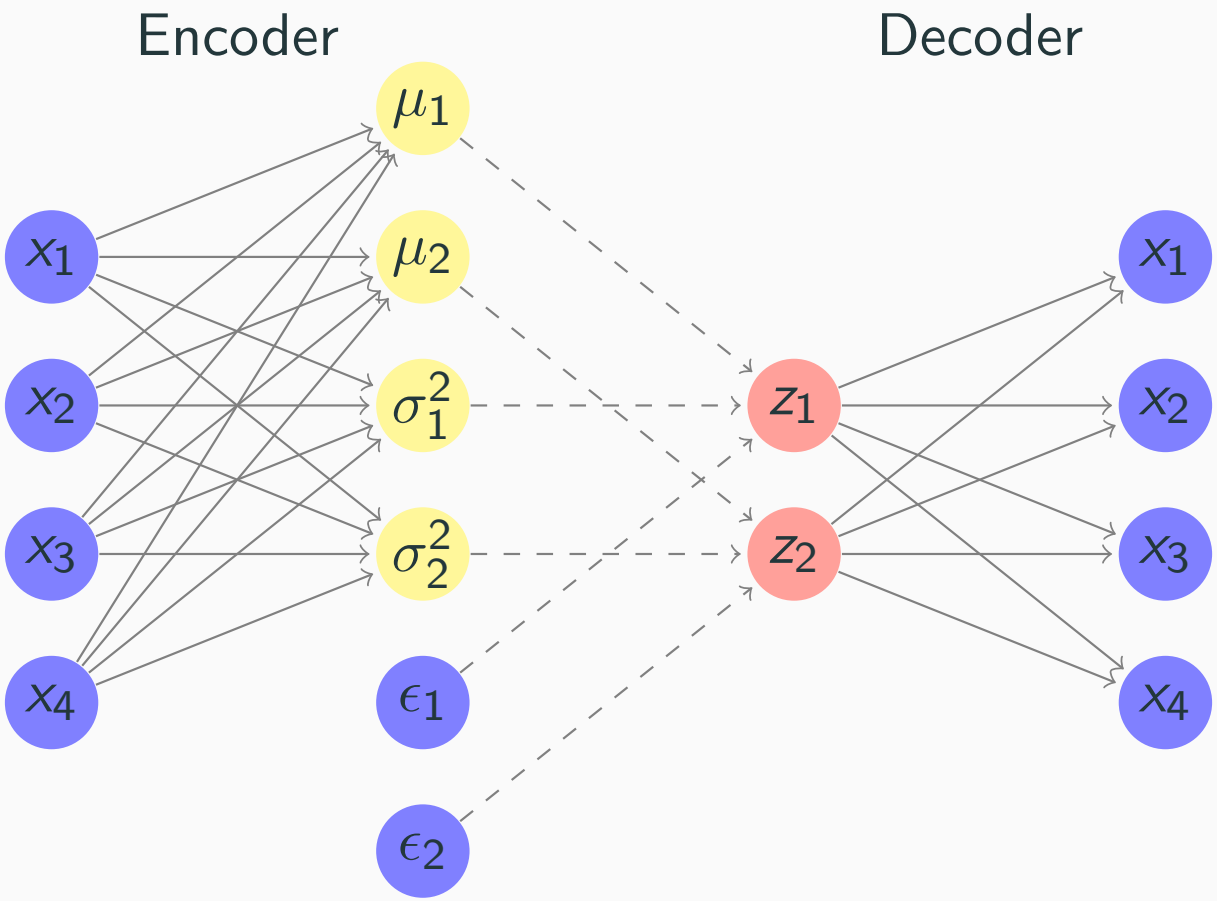


Figure 2: Inputs, x, are shown in blue. The latent representation, z, is shown in red. The parameters, μ and σ^2, of the normal distribution are shown in yellow. They are combined with the noise input, ϵ, by $z = \mu + \sigma * \epsilon$, shown in dashed lines.

Paper Results

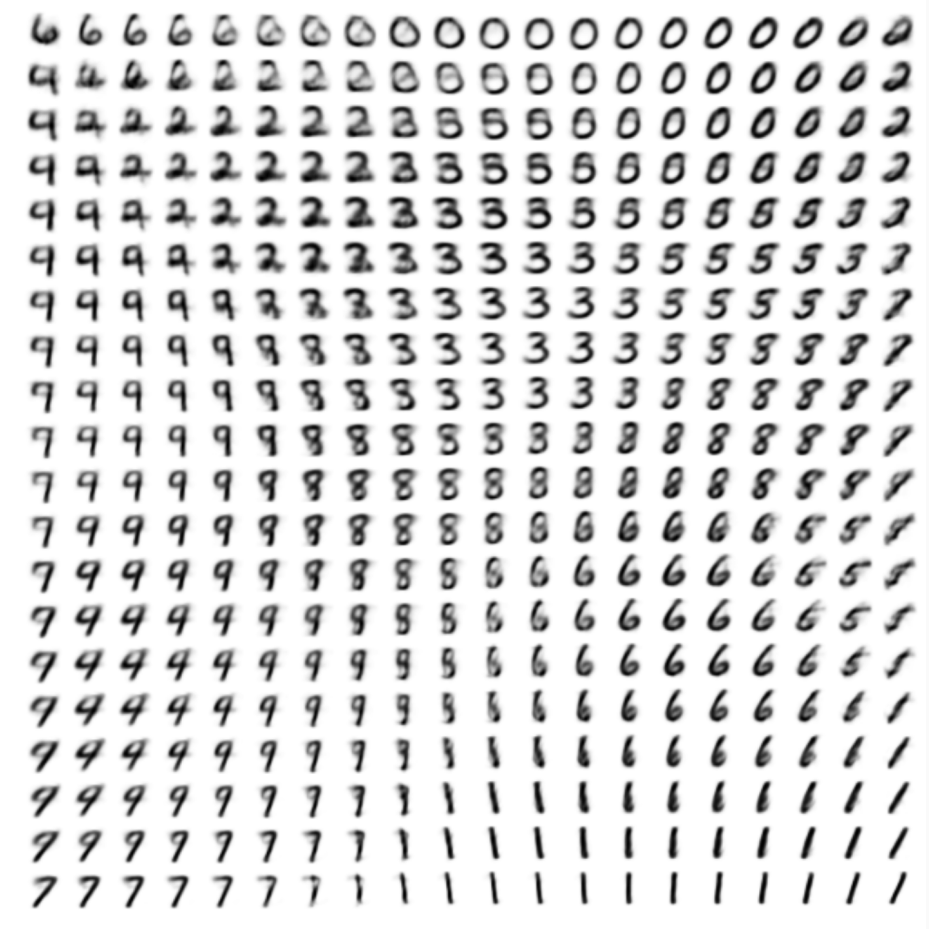
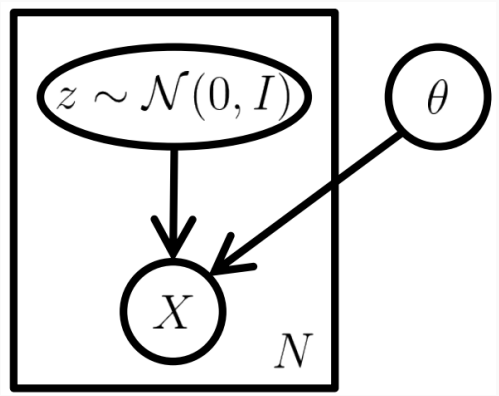


Figure 3: Sampled 2D latent space of MNIST.

The big picture of VAEs

- Goal: maximize $p_\theta(x) = \int p_\theta(x|z)p(z)dz$
  - Generative model intuition: if our model has high likelihood of reproducing the data it has seen, it also has high probability of producing samples similar to x, and low probability of producing dissimilar samples
- How to proceed? Simple: choose $p_\theta(x|z)$ st it's continuous and easy to compute—then we can optimize via SGD
- Examples from "Tutorial on Variational Autoencoders" (Doersch 2016), arXiv:1606.05908

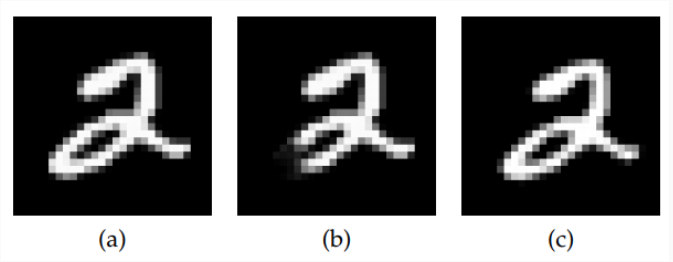


Defining a latent space

- How do we define what information the latent z carries?
- Naively, for MNIST, we might say one dimension conveys digit identity, another conveys stroke width, another stroke angle
  - But we'd rather have the network learn this
- VAE solution: say there's no simple interpretation of z
  - Instead, draw z from $\mathcal{N}(0, I)$, then map through a parameterized and sufficiently expressive function
- Let $p_\theta(x|z) \triangleq \mathcal{N}(x; \mu_\theta(z), \Sigma_\theta(z))$, with $\mu_\theta(\cdot), \Sigma_\theta(\cdot)$ as deterministic neural nets.
- Now tune the parameters θ in order to maximize $p_\theta(x)$.

Estimating $p_\theta(x)$ is hard

- To optimize $p_\theta(x)$ via SGD we will need to compute it.
- We could do Monte Carlo estimate of $p_\theta(x)$ with $z \sim \mathcal{N}(0, I)$, and $p_\theta(x) \approx \frac{1}{n} \sum_i p_\theta(x|z_i)$
- But ... in high dimensions, we likely need extremely large n



- Here, (a) is the original, (b) is a bad sample from model, and (c) is a good sample from model
  - Since $p_\theta(x|z) = \mathcal{N}(x; \mu_\theta(z), \Sigma_\theta(z))$ and with $\Sigma_\theta(z) \triangleq \sigma^2 I$, we have $\log p_\theta(x) \propto -\frac{||\mu_\theta(z) - x||_2^2}{\sigma^2}$
  - x_b is subjectively "bad" but has distance relatively close to the original: $||x_b - x_a||_2^2 = 0.0387$
  - x_c is subjectively "good" (just x_a shifted down & right by half-pixel), but scores poorly since $||x_c - x_a||_2^2 = 0.2693$

Sampling z values efficiently estimate $p_\theta(x)$

- Conclusion: to reject bad samples like x_b, we must set σ^2 to be extremely small
  - But this means that to get samples similar to x_a, we'll need to sample a huge number of z values
  - One solution: define better distance metric—but these are difficult to engineer
  - Better solution: sample only z that have non-negligible $p_\theta(z|x)$
- For most z sampled from $p(z)$, we have $p_\theta(x|z) \approx 0$, so contribute almost nothing to $p_\theta(x)$ estimate
- Idea: define function $q_\phi(z|x)$ that helps us sample z with non-negligible contribution to $p_\theta(x)$

What is Variational Inference?

Posterior inference over z often intractable:

$$p_\theta(z|x) = \frac{p_\theta(x|z)p(z)}{p_\theta(x)} = \frac{p_\theta(z,x)}{p_\theta(x)} = \frac{p_\theta(z,x)}{\int_z p_\theta(x,z)}$$

Want:

Q – tractable family of distribution

$q_\phi(z|x) \in Q$ similar to $p_\theta(z|x)$

Approximate posterior inference using q_ϕ

Idea:

Inference $\rightarrow$ Optimization $\mathcal{L}(x;\theta,\phi)$

Measuring Similarity of Distributions

Optimization objective must measure similarity between p_θ and q_ϕ.

To capture this we use the Kullback-Leibler divergence:

$$\begin{aligned} \mathrm{KL}(q_\phi || p_\theta) &= \int_z q_\phi(z|x) \log \frac{q_\phi(z|x)}{p_\theta(z|x)} \\ &= \mathbb{E}_q \log \frac{q_\phi(z|x)}{p_\theta(z|x)} \end{aligned}$$

Divergence not distance:

$\mathrm{KL}(q_\phi || p_\theta) \geq 0$

$\mathrm{KL}(q_\phi || p_\theta) = 0 \iff q_\phi = p_\theta$

~~$\mathrm{KL}(q || p_\theta) = \mathrm{KL}(p_\theta || q_\phi)$~~ KL is not symmetric!

Intuiting KL Divergence

To get a feeling for what KL Divergence is doing:

$$\mathrm{KL}(q_\phi||p_\theta) = \int_z q_\phi(z|x) \log \frac{q_\phi(z|x)}{p_\theta(z|x)} = \mathbb{E}_{q_\phi} \log \frac{q_\phi(z|x)}{p_\theta(z|x)}$$

Consider these three cases:

q is high & p is high

q is high & p is low

q is low

Isolating Intractability in KL-Divergence

We can't minimize the KL-Divergence directly:

$$
\begin{aligned}
\mathrm{KL}(q_\phi || p_\theta) &= \mathbb{E}_{q_\phi} \log \frac{q_\phi(z|x)}{p_\theta(z|x)} \\
&= \mathbb{E}_{q_\phi} \log \frac{q_\phi(z|x) p_\theta(x)}{p_\theta(z,x)} \qquad \left(p_\theta(z|x) = \tfrac{p_\theta(z,x)}{p_\theta(x)}\right) \\
&= \mathbb{E}_{q_\phi} \log \frac{q_\phi(z|x)}{p_\theta(z,x)} + \mathbb{E}_{q_\phi} \log p_\theta(x) \\
&= \mathbb{E}_{q_\phi} \log \frac{q_\phi(z|x)}{p_\theta(z,x)} + \log p_\theta(x)
\end{aligned}
$$

Isolating Intractability in KL-Divergence

We have isolated the intractable evidence term in KL-Divergence!

$$\begin{aligned} \mathrm{KL}(q_\phi || p_\theta) &= (\mathbb{E}_{q_\phi} \log \frac{q_\phi(z|x)}{p_\theta(z,x)}) + \log p_\theta(x) \\ &= -\mathcal{L}(x;\theta,\phi) + \log p_\theta(x) \end{aligned}$$

Rearrange terms to express isolated intractable evidence:

$$\log p_\theta(x) = \mathrm{KL}(q_\phi || p_\theta) + \mathcal{L}(x;\theta,\phi)$$

Deriving a Variational Lower Bound

Since KL-Divergence is non-negative:

$$\log p_\theta(x) = \mathrm{KL}(q_\phi || p_\theta) + \mathcal{L}(x; \theta, \phi)$$
$$\log p_\theta(x) \geq \mathcal{L}(x; \theta, \phi)$$

where

$$\mathcal{L}(x; \theta, \phi) = -\mathbb{E}_{q_\phi} \log \frac{q_\phi(z|x)}{p_\theta(z, x)}$$

A Variational Lower Bound on the intractable evidence term!
This is also called the Evidence Lower Bound (ELBO).

Intuiting Variational Lower Bound

Expand the derived variational lower bound:

$$
\begin{aligned}
\mathcal{L}(x;\theta,\phi) &= -\mathbb{E}_{q_\phi}[\log \frac{q_\phi(z|x)}{p_\theta(z,x)}] \\
&= \mathbb{E}_{q_\phi}[\log \frac{p_\theta(x|z)p(z)}{q_\phi(z|x)}] \\
&= \mathbb{E}_{q_\phi}[\log p_\theta(x|z) + \log p(z) - \log q_\phi(z|x)] \\
&= \mathbb{E}_{q_\phi}[\log p_\theta(x|z) + \log \frac{p(z)}{q_\phi(z|x)}] \\
&= \underbrace{\mathbb{E}_{q_\phi}[\log p_\theta(x|z)]}_{\text{Reconstruction Likelihood}} - \underbrace{\mathrm{KL}(q_\phi(z|x)||p(z))}_{\text{Divergence from Prior}}
\end{aligned}
$$

Optimizing the ELBO in VAE

To optimize the ELBO,

$$\mathcal{L}(x;\theta,\phi) = \underbrace{\mathbb{E}_{z\sim q_\phi(z|x)}[\log p_\theta(x|z)]}_{\triangleq \mathcal{R}(x;\theta,\phi)} - \underbrace{\mathrm{KL}(q_\phi(z|x)||p(z))}_{\text{Divergence from prior; analytic expression by design}},$$

$\triangleq \mathcal{R}(x;\theta,\phi)$: Reconstruction likelihood

we need to compute gradients $\nabla_\theta \mathcal{L}$ and $\nabla_\phi \mathcal{L}$.

- $\nabla_\theta \mathrm{KL}(\cdot)$ and $\nabla_\phi \mathrm{KL}(\cdot)$ by automatic differentiation
- $\nabla_\theta \mathcal{R}(x;\theta,\phi)$ by auto diff given samples $z \sim q_\phi(z|x)$
- $\nabla_\phi \mathcal{R}(x;\theta,\phi)$ by reparameterization trick or other gradient estimator

Reparameterizing: a computation graph view

With[1] $q_\phi(z|x) \triangleq g(\phi, x, \epsilon)$:

$$\begin{aligned}
&\nabla_\phi \mathbb{E}_{z\sim q_\phi(z|x)}[f(z)] \\
&= \nabla_\phi \int f(z) q_\phi(z|x) dz \\
&\overset{(rep.tr.)}{=} \nabla_\phi \int f(g(\phi, x, \epsilon)) p(\epsilon) d\epsilon \\
&= \mathbb{E}_{p(\epsilon)}[\nabla_\phi f(g(\phi, x, \epsilon))]
\end{aligned}$$

With $(rep.tr.)$ due to $|q_\phi(z|x)dz| = |p_\theta(\epsilon)d\epsilon|$. This permits a specific alteration to the computation graph without introducing bias.

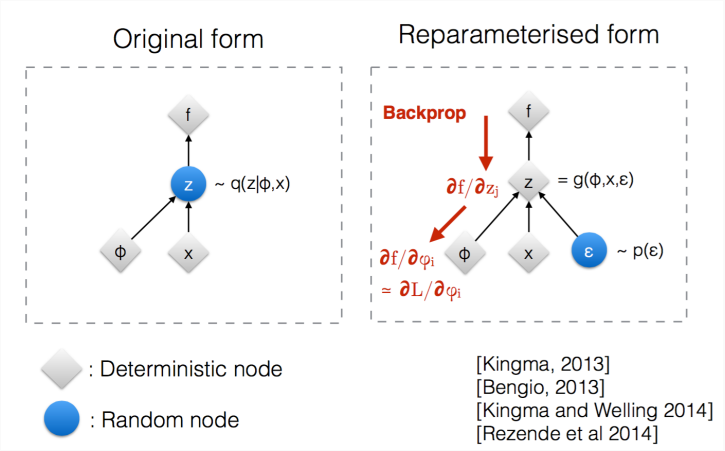


Figure 4: from Kingma's slides at NIPS 2015 Workshop on Approx. Inference

Other gradient estimators

Yes, the reparameterization trick *makes back-prop work* for estimating gradients like $\nabla_\phi \mathbb{E}_{q_\phi(z)}[f_\theta(z)]$, but there are other options. In general, we want unbiased gradient estimators with low variance.

- score function estimator (i.e., REINFORCE):
  $\nabla_\phi \mathbb{E}_{z\sim q_\phi(z)}[f_\theta(z)] = \mathbb{E}_{z\sim q_\phi(z)}[f_\theta(z)\nabla_\phi \log q_\phi(z)]$
  - unbiased, high variance
- reparameterization trick:
  $z = g(\epsilon, \phi) \rightarrow \nabla_\phi \mathbb{E}_{z\sim q_\phi(z)}[f_\theta(z)] = \mathbb{E}_{\epsilon\sim p_\theta(\epsilon)}[\nabla_\phi f_\theta(g(\epsilon, \phi))]$
  - unbiased, reasonably low variance
- straight-through estimator: pretend the stochastic node acts like an identity function on the backward pass
  - biased
- etc.

Approximating full Bayes

Approximate MAP

- Recast $\int_z(\cdot)dz$ as an optimization.
- Variational dist'ns $q_\phi(z|x)$; prior $p_\theta(z)$.

$$\log p_\theta(X) \geq \mathcal{L}(X;\theta,\phi)$$
$$= \mathbb{E}_{z\sim q_\phi(z|x)}[\log p_\theta(x|z)] - \mathrm{KL}(q_\phi(z|x)||p_\theta(z))$$

- Estimate ∇_ϕ by reparameterizing $q_\phi(z|x)$ then back-prop.
- Estimate ∇_θ by sampling $q_\phi(z|x)$ and pathwise derivative estimator.

Approximate full Bayes

- Recast $\int_\theta(\cdot)d\theta$ as an optimization.
- Variational dist'ns $q_\phi(\theta)$, $q_\phi(z|x)$; hyperprior $p_\alpha(\theta)$.

$$\log p_\alpha(X) \geq \mathcal{L}(\phi;X)$$
$$= \mathbb{E}_{\theta\sim q_\phi(\theta)}[\log p_\theta(X) \log p_\alpha(\theta) - \log q_\phi(\theta)]$$

- Estimate ∇_ϕ by reparameterizing $q_\phi(\theta)$ and $q_\phi(z|x)$ then back-prop.

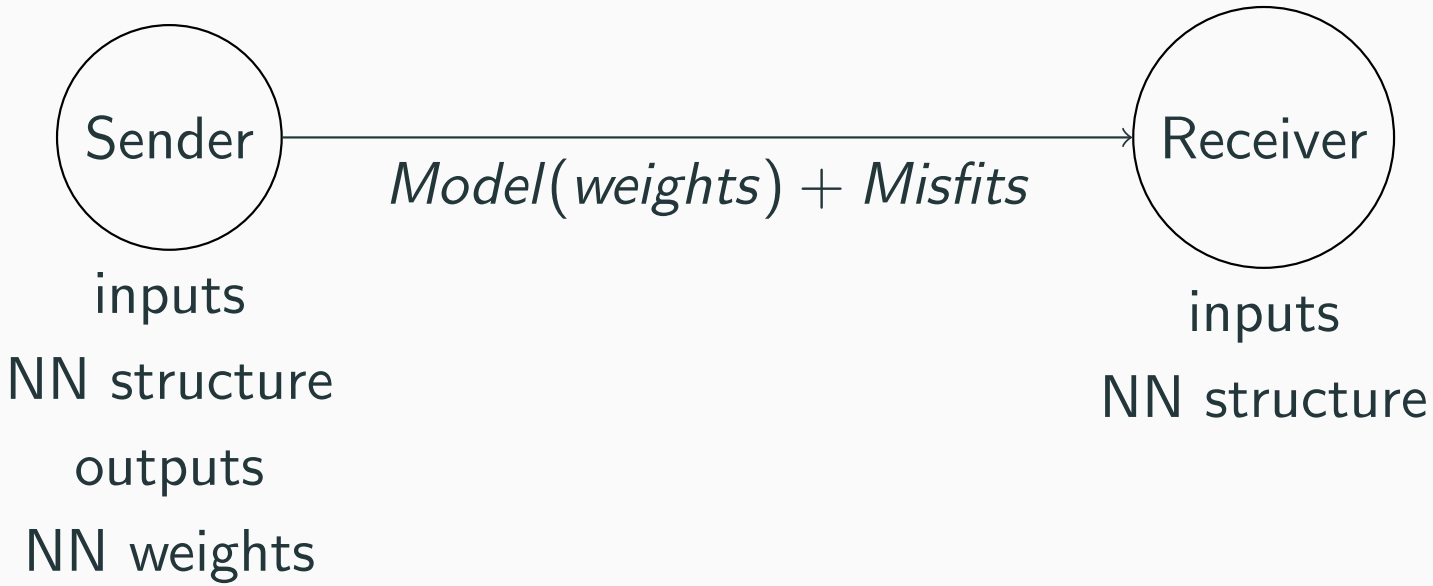
Variational Inference for BNNs

Variational Inference for BNNs (Originations)

- Originally started with Hinton and Camp work.
- They had information theoretic view to the supervised learning problem.
- Used minimum description length (MDL) principle to improve generalization on new data
- Introduced bits-back argument (**KL divergence** showed himself here!)

Minimum Description Length

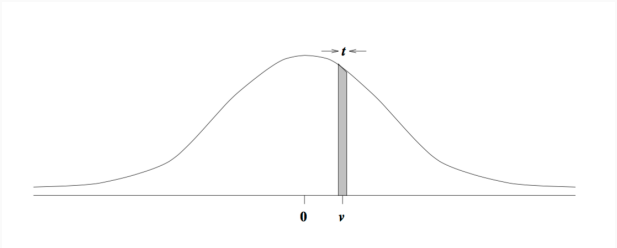
- Which model is the best?
- According to MDL principle, A model is best that minimizes the combined cost of
  - **Describing the model**
  - **Describing the misfit between the model and the data.**



Shannon's Coding Theorem

- Entropy definition: $H(X) = \sum_x P(x)(-\log P(x))$
- **Shannon's Coding Theorem:**
  - N i.i.d. random variables each with entropy $H(X)$ can be compressed into more than $NH(X)$ bits with negligible risk of information loss, as $N \to \infty$.
  - Conversely, if they are compressed into fewer than $NH(X)$ bits it is virtually certain that information will be lost.
- According to this theorem, if a sender and a receiver have agreed on a distribution $P(x)$, then we can code the x using $-\log P(x)$ bits.

Coding the Data Misfits and the Weights



- Coding Misfits
  - Assuming data misfits are coming from a Gaussian distribution: $P(d_j^c - y_j^c) = \boldsymbol{t} \frac{1}{\sqrt{2\pi}\sigma_j} \exp(\frac{-(d_j^c - y_j^c)^2}{2\sigma_j^2})$
  - So, description length would be:
    $-\log P(d_j^c - y_j^c) = -\log t + log\sqrt{2\pi} + \log \sigma_j + \frac{-(d_j^c - y_j^c)^2}{2\sigma_j^2}$
- Coding Weights
  - Assuming a weight $w_{i,j}$ is coming from a zero-mean Gaussian distribution with a fixed variance σ_w^2, we can get a similar description length.
- Total Cost
  - By removing constants total cost will become:
    $C = \sum_j \frac{1}{2\sigma_j^2} \sum_c ((d_j^c - y_j^c)^2) + \frac{1}{2\sigma_w^2} \sum_{i,j} w_{i,j}^2$
  - This is just the classic standard "weight-decay" method.

Adding Noise to Weights

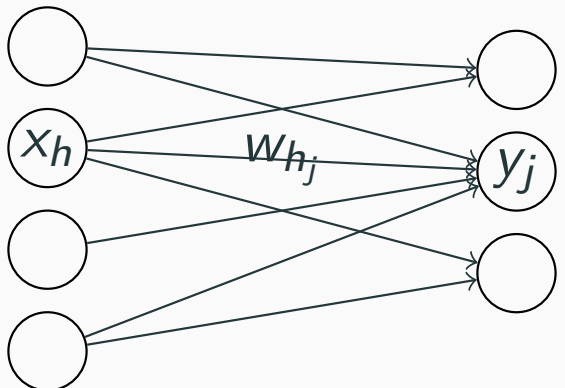
- More complicated problem can be obtained by adding Gaussian noise to weights.
- Suppose sender and receiver have agreed on a Gaussian prior P, for a given weight. After learning, the sender has a Gaussian distribution, Q, for the weight.
  $P(w) : Normal$
  $Q(w|D) = N(\mu_w, \sigma_w^2) \Rightarrow w = \mu_w + \epsilon, \epsilon \sim N(0, \sigma_w^2)$
- Now, lets send the a noisy weight (model description) that comes from posterior distribution by **"bits-back"** coding scheme.

Bits-back Argument

- Before the beginning choose a very fine precision value t.
- Sender **collapses the posterior by using a source of random bits**
- Sender then **picks a precise w from $Q(w|D)$** and **encode it using $P(w)$**. So, expected cost of sending w is: $C = -\int Q(w|D)\log(tP(w))dw$
- But wait! Suppose sender has sent the misfits too. So, by having the precise weights and the misfits, **the receiver has whatever is needed to run the learning algorithm (whatever it was) to obtain the posterior.** Thus, he can recover the random bits used to encode posterior into that weight. The expected value of the number of random bit used to collapse posterior is: $R = -\int Q(w|D)\log(tQ(w|D))dw$
- Total cost will be: $C - R = D_{KL}[P||Q]$

Data Misfits Cost in the Noisy Weights Case

- In the noisy weights case, for general feedforward neural networks, it's hard to calculate the cost of data misfits.
- We needed to compute expected value of $(d_j - y_j)^2$. It could be written as follows: $\mathbb{E}[(d_j - y_j)^2] = (d_j - \mu_{y_j})^2 + V_{y_j}$
- for a feedforward neural network hidden layer without non-linearities, assuming: $mean[x_h] = \mu_{x_h}, var[x_h] = V_{x_h}, mean[w_{h_j}] = \mu_{w_{h_j}}, var[w_{h_j}] = V_{w_{h_j}}, y_j = \sum_h w_{h_j} x_h$
- mean and variance of y_j could be computed as follows: $\mu_{y_j} = \sum_h \mu_{w_{h_j}} \mu_{x_h}$, $V_{y_j} = \sum_h \mu^2_{w_{h_j}} V_{x_h} + \mu^2_{x_h} + V_{x_h} V_{w_{h_j}}$
- Then we can do backpropagation on $E = \sum_j \mathbb{E}[(d_j - y_j)^2]$ to obtain mean and variance updates.



Hyper-priors, Other Priors

- **Hyper-priors**
  - So far, we have assumed the prior that is used for coding the weights is a single Gaussian.
  - In a Bayesian approach we set some hyper-parameters for the parameters of the coding-prior. This would take into account the cost of communicating coding-prior given hyper-priors. In practice, we just ignore the cost of communicating the two parameters of the coding-prior. This is in some sense similar to type 2 maximum likelihood (marginalizing out the parameters): $\arg\max_\alpha P(y|\boldsymbol{x}, \alpha) = \int P(y|\boldsymbol{x}, w)P(w|\alpha)dw$
- **More flexible prior**
  - Gaussian prior is too limited to model many distributions on weights in a feedforward neural network. Mixture of Gaussians could be a good substitute. Why?
    - Can model different structures.
    - Could be useful when we want different coding-priors in different subsets.

Practical VI for Neural Networks

Graves 2011

- Stochastic Variational Inference for Neural Networks
  - Minimum description length (MDL)
  - Approximate inference as Compression
  - Optimisation
- Bayesian Formulation (VI) vs Coding Theory (MDL)
  - Predictive Accuracy
  - Generalization
  - Model selection
- Occam's Razor in Minimum message length (MML)
  - Regularisation

Bayesian Formulation

- Variational free energy

$$\mathcal{F}(\boldsymbol{\alpha}, \boldsymbol{\beta}; \mathcal{D}) = \left\langle \log \left[\frac{q_{\boldsymbol{\beta}}(\mathbf{w}|\mathcal{D})}{p(\mathcal{D}|\mathbf{w}) p_{\boldsymbol{\alpha}}(\mathbf{w})} \right] \right\rangle_{\mathbf{w} \sim q_{\boldsymbol{\beta}}(\mathbf{w}|\mathcal{D})}$$

$$L^N(\mathbf{w}, \mathcal{D}) = -\log p(\mathcal{D}|\mathbf{w})$$

- Evidence lower bound (ELBO)

$$\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\phi}; \mathbf{x}) = \mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})} \left[\log p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z}) \right] - \mathbb{D}_{\mathrm{KL}}(q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}) \| p_{\boldsymbol{\theta}}(\mathbf{z}))$$

- Equivalent formulations

$$\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\phi}; \mathbf{x}) = -\mathcal{F}(\boldsymbol{\theta}, \boldsymbol{\phi}; \mathbf{x})$$

Minimum description length (mdl)

- Transmission cost

$$\mathcal{F}(\boldsymbol{\alpha}, \boldsymbol{\beta}; \mathcal{D}) = \langle L^N(\mathbf{w}, \mathcal{D}) \rangle_{\mathbf{w} \sim q_{\boldsymbol{\beta}}(\mathbf{w})} + \mathcal{D}_{\mathsf{KL}}(q_{\boldsymbol{\beta}}(\mathbf{w}) \| p_{\boldsymbol{\alpha}}(\mathbf{w}))$$

$$L^E(\boldsymbol{\beta}, \mathcal{D}) = \langle L^N(\mathbf{w}, \mathcal{D}) \rangle_{\mathbf{w} \sim q_{\boldsymbol{\beta}}(\mathbf{w})}$$

$$L^C(\boldsymbol{\alpha}, \boldsymbol{\beta}) = \mathcal{D}_{\mathsf{KL}}(q_{\boldsymbol{\beta}}(\mathbf{w}) \| p_{\boldsymbol{\alpha}}(\mathbf{w}))$$

$$\mathcal{F}(\boldsymbol{\alpha}, \boldsymbol{\beta}; \mathcal{D}) = L^E(\boldsymbol{\beta}, \mathcal{D}) + L^C(\boldsymbol{\alpha}, \boldsymbol{\beta})$$

- MDL principle for learning

$$\begin{aligned} \mathcal{L}(\mathcal{D}) &= \mathcal{L}(\boldsymbol{\theta}) + \mathcal{L}(\mathcal{D}|\boldsymbol{\theta}) \\ &= \underbrace{-\log(p(\boldsymbol{\theta}|\mathcal{H})\epsilon_{\boldsymbol{\theta}}^{|\boldsymbol{\theta}|})}_{\text{Complexity cost}} \underbrace{-\log(p(\mathcal{D}|\boldsymbol{\theta}, \mathcal{H})\epsilon_{\mathcal{D}}^{|\mathcal{D}|})}_{\text{Error cost}} \end{aligned}$$

Bits-back coding

- Expected code length

$$\mathbb{E}_{q(\boldsymbol{\theta})}[\mathcal{L}(\mathcal{D})] = \mathbb{E}_{q(\boldsymbol{\theta})}[\mathcal{L}(\boldsymbol{\theta})] + \mathbb{E}_{q(\boldsymbol{\theta})}[\mathcal{L}(\mathcal{D}|\boldsymbol{\theta})]$$

- Expected bits-back coding length

$$\begin{aligned}
\mathcal{L}_{q(\boldsymbol{\theta})}(\mathcal{D}) &= \mathbb{E}_{q(\boldsymbol{\theta})}[\mathcal{L}(\mathcal{D})] - \mathbb{H}[q(\boldsymbol{\theta})] \\
&= \left\langle \log\left[\frac{q(\boldsymbol{\theta})}{p(\mathcal{D}|\boldsymbol{\theta},\mathcal{H})p(\boldsymbol{\theta}|\mathcal{H})}\right] \right\rangle_{\boldsymbol{\theta}\sim q(\boldsymbol{\theta})} \\
&= \mathcal{D}_{\mathsf{KL}}(q(\boldsymbol{\theta}) \| p(\boldsymbol{\theta}|\mathcal{D},\mathcal{H})) - \log\left(p(\mathcal{D}|\mathcal{H})\right) \\
\mathcal{L}_{\text{optimal}}(\mathcal{D}) &= -\log\left(p(\mathcal{D}|\mathcal{H})\right)
\end{aligned}$$

- Optimisation = Compression

Mean field approximation

- $q(\boldsymbol{\beta}) = \prod_{i=1}^{W} q_i(\boldsymbol{\beta}_i) \implies L^C(\boldsymbol{\alpha}, \boldsymbol{\beta}) = \sum_{i}^{W} \mathbb{D}_{KL}(q_i(\boldsymbol{\beta}_i)|p(\boldsymbol{\alpha}))$
- SGD affected by choice of posterior $q(\boldsymbol{\beta})$ and prior $p(\boldsymbol{\alpha})$
- Delta Posterior
  - $L^C(\boldsymbol{\alpha}, \boldsymbol{\beta}) = -\log\left(p(\mathbf{w}|\boldsymbol{\alpha})\right) + C$
  - Uniform prior $\implies$ MLE
  - Laplace prior $\implies$ L1 regularisation
  - Gaussian prior $\implies$ L2 regularisation
- Diagonal Gaussian Posterior
  - Uniform prior $\implies$ weight noise
  - Gaussian prior $\implies$ adaptive weight noise

Variational Regularisation

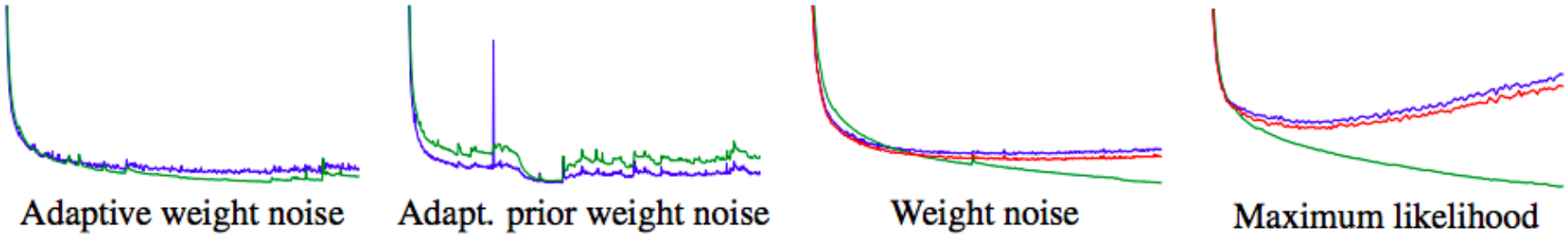


Figure 2: **Error curves for four networks during training.** The green, blue and red curves correspond to the average per-sequence error loss $L^E(\boldsymbol{\beta}, \mathcal{D})$ on the training, test and validation sets respectively. Adaptive weight noise does not overfit, and normal weight noise overfits much more slowly than maximum likelihood. Adaptive weight noise led to longer training times and noisier error curves.

Figure 5: Preventing overfitting

Model Selection and Pruning

- Pruning
  - high $q(\mathbf{w}|\boldsymbol{\beta}) \Longrightarrow$ low $L^N(\mathbf{w}, \mathcal{D})$ and pruning $w_k \Leftrightarrow w_k = 0$
  - Remove w if $q(\mathbf{w} = 0|\boldsymbol{\beta})$ is high
  - $\exp(-\frac{\mu_i^2}{2\sigma_i^2}) \geq \gamma \Longrightarrow |\frac{\mu_i}{\sigma_i}| \leq \lambda = \sqrt{-2\log 2}$
- Bayes Factor

$$\frac{p(\mathcal{H}_1|\mathcal{D})}{p(\mathcal{H}_2|\mathcal{D})} = \frac{p(\mathcal{H}_1)}{p(\mathcal{H}_2)} \frac{p(\mathcal{D}|\mathcal{H}_1)}{p(\mathcal{D}|\mathcal{H}_2)}$$

- Occam's factor and the prior
- MML principle : Shortest overall message more probable
- Uncertainty aids compression and prevents overfitting

Model Generalisation

Table 2: **Effect of Network Pruning.** 'λ' is the threshold used for pruning. 'Weights' is the number of weights left after pruning and 'Percent' is the same figure expressed as a percentage of the original weights. 'Initial Error' is the test error immediately after pruning and 'Retrain Error' is the test error following 'Retrain Epochs' of subsequent retraining. 'Bits/weight' is the average bit cost (as defined in Eq. (13)) of the *unpruned* weights.

λ	Weights	Percent	Initial error	Retrain error	Retrain Epochs	Bits/weight
0	139,536	100%	23.8	23.8	0	0.53
0.01	107,974	77.4%	23.8	24.0	972	0.72
0.05	63,079	45.2%	23.9	23.5	35	1.15
0.1	52,984	37.9%	23.9	23.3	351	1.40
0.2	43,182	30.9%	23.9	23.7	740	1.82
0.5	31,120	22.3%	24.0	23.3	125	2.21
1	22,806	16.3%	24.5	24.1	403	3.19
2	16,029	11.5%	28.0	24.5	335	3.55

Figure 6: Improving generalisation

Weight Uncertainty in Neural Networks

by Charles Blundell, Julien Cornebise, Koray Kavukcuoglu, and Daan Wierstra

Problem Focused in the Work

- Utilize re-parametrization trick for limited amount of parameters introduced through variational inference
- Formulate objective function to relax restrictions on prior and variational posterior for requiring closed-form expression; allowing more prior/variational posterior combinations
- Develop optimization algorithm for obtaining unbiased gradient estimates, with small variances in gradient signals

Recap of Variational Inference

- Let $P(\mathbf{w})|\mathfrak{D})$ denote the actual posterior distribution on weights provided with prior and data; Let $q(\mathbf{w})|\theta)$ denote "variational posterior": the distribution used to approximate the actual posterior
- The essence of Variational Inference is to use Kullback-Leibler divergence as the metric to obtain quality variational posterior

Mathematical Formulation for Optimization Problem

- Optimization on variational posterior parameters is minimization on KL divergence written as following:

$$\theta^* = argmin_\theta KL[q(\mathbf{w}|\theta)||P(\mathbf{w}|\mathfrak{D})] \tag{1}$$

$$= argmin_\theta \int q(\mathbf{w}|\theta) \log \frac{q(\mathbf{w}|\theta)}{P(\mathbf{w}P(\mathfrak{D}|\mathbf{w}))} \tag{2}$$

$$= argmin_\theta \mathbf{KL}[q(\mathbf{w}|\theta)||P(\mathbf{w})] - \mathbb{E}_{q(\mathbf{w}|\theta)}[\log P(\mathfrak{D}|\mathbf{w})] \tag{3}$$

- The paper proposes gradient descent based optimization on above expression through the methods shown in following slides, without need for computing closed formed KL terms.
- This relaxes restriction on prior and posterior forms of selection.

Defining the Objective Function for Optimization

- With optimization addressing minimizing KL divergence as defined previously, would like to reformulate it into a convenient choice of objective function that's easy to optimize
- Recap the optimization as parameters minimization on the KL divergence between variational posterior and actual posterior as our cost function:

$$\theta^* = argmin_\theta \int q(\mathbf{w}|\theta) \log \frac{q(\mathbf{w}|\theta)}{P(\mathbf{w})P(\mathfrak{D}|\mathbf{w})} d\mathbf{w} \tag{4}$$

- Define the objective function $f((w), \theta)$ as the component being taken expectation of:

$$f(\mathbf{w}, \theta) = \log q(\mathbf{w}|\theta) - \log P(\mathbf{w})P(\mathfrak{D}|\mathbf{w}) \tag{5}$$

- With substituting in this cost function notation, the KL divergence minimization problem becomes:

$$\theta^* = argmin_\theta \mathbb{E}_{q(\mathbf{w}|\theta)} f(\mathbf{w}, \theta) \tag{6}$$

Benefits for the Objective Function Choice

- With the objective function defined previously, a Monte Carlo estimation is as following:

$$f(\mathbf{w}, \theta) \approx \sum_{i=1}^{n} \log q(\mathbf{w}^{(i)}|\theta) - \log P(\mathbf{w}^{(i)}) - \log P(\mathfrak{D}|\mathbf{w}^{(i)}) \quad (7)$$

  With weights samples $\mathbf{w}^{(i)}$ drawn according to our variational posterior
- This formulation of objective function provides two computational benefits:
  1. Every term depends upon $\mathbf{w}^{(i)}$ drawn from variational posterior, thus utilizing a variance reduction technique common random numbers (Owen, 2013) for the approximation.
  2. Note that unlike original Variational Inference formulations (maximizing ELBO):

$$ELBO = \mathbb{E}_{w\ q(w|\theta)}[\log p_\theta(x|z)] - \underbrace{KL(q(w|\theta)||p(z))}_{\substack{\text{complexity cost of the model} \\ \text{analytically computed for closed form term}}} \quad (8)$$

     This objective function doesn't collect terms for getting this KL term, and thus not requiring closed form solution to be computed. Thus this allows richer prior/posterior combinations.

Optimization by Gradient Descent

- With objective function defined, learning focuses on minimizing objective function (thus the KL-divergence for high quality variational posterior) by learning variational posterior parameters (to be defined later):

$$\frac{\partial}{\partial \theta}\mathbb{E}_{q(\mathbf{w}|\theta)}[f(\mathbf{w}|\theta)] \tag{9}$$

- It is attempting to directly use Monte Carlo estimates sampling from variational posterior, given the expectation form.
- However by implementing a reparametrization trick, the gradient signals could be obtained through standard back-propagation; while reducing the gradient signal variance (as introduced in one of the gradient estimators in presentation Part I).
- To illustrate the reparametrization, a mathematical proposition is needed first.

An Important Mathematical Proposition

- A proposition is introduced to utilize the above reparametrization in estimation of gradient expectation
- **Proposition 1.** *Let ϵ be a random variable having a probability density given by $q(\epsilon)$ and let $\mathbf{w} = t(\theta, \epsilon)$ where $t(\theta, \epsilon)$ is a deterministic function. Suppose further that the marginal probability density of $\mathbf{w}, q(\mathbf{w}|\theta)$, is such that $q(\epsilon)d\epsilon = q(\mathbf{w}|\theta)d\mathbf{w}$. Then for a function f with derivatives in $\mathbf{w}$:*

$$\frac{\partial}{\partial\theta}\mathbb{E}_{q\mathbf{w}|\theta}[f(\mathbf{w}|\theta)] = \mathbb{E}_{q(\epsilon)}[\frac{\partial f(\mathbf{w},\theta)}{\partial\mathbf{w}}\frac{\partial\mathbf{w}}{\partial\theta} + \frac{\partial f(\mathbf{w},\theta)}{\partial\theta}] \quad (10)$$

Reparametrization Trick on Variational Posterior

- Need to define parameters θ for variational posterior $q(\mathbf{w}|\theta)$
- Desire to be both easy for gradient expectation computation, and efficient in amount of parameters introduced
- With assuming **Gaussian variational posterior**, the paper proposes the following reparametrization trick:
  Start by sampling a unit Gaussian vector, denoted by ϵ Define variational posterior parameters "θ" to be: $\theta = (\mu, \rho)$ denoting element-wise mean and variance
  The Gaussian variational posterior is then defined as:

$$\mathbf{w} = \mu + \rho\epsilon \tag{11}$$

- Note: to ensure the variance is always positive during training, the following parameterization is actually use to denote the variance:
  variance: $\log(1 + exp(\rho))$
  Thus, the final reparametrization for variational posterior is:

$$\mathbf{w} = \mu + \log(1 + exp(\rho))\epsilon \tag{12}$$

Optimize Network by Using Unbiased Monte Carlo Gradients

- With gradient decent optimization, computation is to be conducted for gradients of above cost function expectation with respect to parameters.
- According to the previously mentioned proposition, along with the reparametrization trick on variational posterior, the gradient expression then could be reformulated as:

$$\frac{\partial}{\partial \theta}\mathbb{E}_{q(\mathbf{w}|\theta)}f(\mathbf{w},\theta) = \mathbb{E}_{q(\epsilon)}[\frac{\partial f(\mathbf{w},\theta)}{\partial \mathbf{w}}\frac{\partial \mathbf{w}}{\partial \theta} + \frac{\partial f(\mathbf{w},\theta)}{\partial \theta}] \quad (13)$$

- Thus, with the above reformulation, Monte Carlo estimates could be formed by taking samples from unit Gaussian ϵ directly rather than from variational posterior $q(\mathbf{w}|\theta)$

Algorithm Steps for Optimization with Variational Inference on Weights Posteriors

- With the previous problem reformulation, utilizing Monte Carlo estimates, the detailed algorithm steps for optimizing variational posterior parameters are as following:

1. Sample ϵ $\mathcal{N}(0, I)$.
2. Let $\mathbf{w} = \mu + \log(1 + exp(\rho)) \odot \epsilon$
   (with $\odot$ denoting element-wise multiplication)
3. Let $\theta = (\mu, \rho)$
4. Let $f(\mathbf{w}, \theta) = \log q(\mathbf{w}|\theta) - \log P(\mathbf{w})P(\mathfrak{D}|\mathbf{w})$.

Continued: Algorithm Steps for Optimization with Variational Inference on Weights Posteriors

5. Calculate the gradient with respect to the mean

$$\delta_\mu = \frac{\partial f(\mathbf{w}, \theta)}{\partial \mathbf{w}} + \frac{\partial f(\mathbf{w}, \theta)}{\partial \mu} \tag{14}$$

6. Calculate the gradient with respect to the standard deviation parameter ρ

$$\delta_\rho = \frac{\partial f(\mathbf{w}, \theta)}{\partial \mathbf{w}} \frac{\epsilon}{1 + exp(-\rho)} + \frac{\partial f(\mathbf{w}, \theta)}{\partial \rho} \tag{15}$$

7. Update the variational parameters:

$$\mu \leftarrow \mu - \alpha\delta_\mu \rho \leftarrow \rho - \alpha\delta_\rho \tag{16}$$

- Observation: Note for the above differentiation terms, the term $\frac{\partial f(\mathbf{w},\theta)}{\partial \mathbf{w}}$ is shared among both mean and standard deviation gradients.
- Also notice this term could be found through starting with normal backpropagation through the network, then scaled and shifted based on other components within the derivative trivially computed.

Some Details, Variations for the Algorithm

- Scale mixture prior:
  - As no closed form complexity cost and entropy term is required. Design constraint on prior could be relaxed.
  - In the paper, the prior is used with a mixture of two Gaussians: one with small variance and another with large variance, which resembles "spike-and-slab" prior (to be covered more in later "Sparsification" section).
- Minibatches and KL re-weighting
  - Recall the KL divergence cost being:

$$f(\mathfrak{D}, \theta) = KL[q(\mathbf{w}|\theta)||P(\mathbf{w})] - \mathbb{E}_{q(\mathbf{w}|\theta)}[\log P(\mathfrak{D}|\mathbf{w})] \tag{17}$$

  - This cost function could be optimized by breaking down into components corresponding to minibatches:

$$f_i^{\pi}(\mathfrak{D}_i, \theta) = \pi_i KL[q(\mathbf{w}|\theta)||P(\mathbf{w})] - \mathbb{E}_{q(\mathbf{w}|\theta)}[\log P(\mathfrak{D}_i|\mathbf{w})] \tag{18}$$

    With $\pi_i = \frac{2^{M-i}}{2^M - 1}$ ("M" being amount of minibatches).
  - This partition weight coefficients π_i ensures first few minibatches focus heavily on complexity cost; while in later minibatches with more and more data observed, data likelihood gradually becomes the focus for the cost function.

Local Reparameterization Trick **Variational Dropout and the Local Reparameterization Trick**

By Diederik P. Kingma, Tim Salimans, and Max Welling

Motivation

- If the **variance** in the gradients is **too large**, the stochastic gradient ascent may **not perform well**.
- What's the variance of the Stochastic Gradient Variational Bayes (SGVB) estimator and how can we reduce it?

Variational Inference

- Given N i.i.d. observation tuples $(\mathbf{x}, \mathbf{y}) \in \mathcal{D}$, we want to learn a model with parameters $\mathbf{w}$ of the conditional probability $p(\mathbf{y}|\mathbf{x}, \mathbf{w})$
- Optimize parameters ϕ of parameterized model $q_\phi(\mathbf{w})$ such that $q_\phi(\mathbf{w})$ closely approximates $p(\mathbf{w}|\mathcal{D})$ as measured by KL-divergence.
- Done by maximizing Evidence Lower Bound $\mathcal{L}(\phi)$ of the marginal likelihood of the data:

$$\mathcal{L}(\phi) = -D_{KL}(q_\phi(\mathbf{w})||p(\mathbf{w})) + \underbrace{\sum_{(x,y)\in\mathcal{D}} (\mathbb{E}_{q_\phi(\mathbf{w})}[\log p(\mathbf{y}|\mathbf{x}, \mathbf{w})])}_{\text{Expected Log-Likelihood } L_{\mathcal{D}(\phi)}} \quad (19)$$

Stochastic Gradient Variational Bayes (SGVB)

- The SGVB parameterize the random parameters $\mathbf{w} \sim q_\phi(\mathbf{w})$ as $\mathbf{w} = f(\epsilon, \phi)$ with $f(.)$ differentiable and $\epsilon \sim p(\epsilon)$ a random noise variable.
- The unbiased differentiable minibatched-based Monte Carlo estimator of the expected log-likelihood:

$$L_{\mathcal{D}} \simeq L_{\mathcal{D}}^{SGVB}(\phi) = \frac{N}{M} \sum_{i=1}^{M} \log p(\mathbf{y}^i | \mathbf{x}^i, \mathbf{w} = f(\epsilon, \phi)) \qquad (20)$$

where $(\mathbf{x}^i, \mathbf{y}^i)_{i=1}^M$ is a minibatch of data with M random datapoints from data $\mathcal{D}$

Variance of SGVB

- Let $L_i = \log p(\mathbf{y}^i|\mathbf{x}^i, \mathbf{w} = f(\epsilon, \phi))$ as a shorthand
- The variance of the estimator:

$$Var[L_{\mathcal{D}}^{SGVB}(\phi)] = \frac{N^2}{M^2}\left(\sum_{i=1}^{M} Var[L_i] + 2\sum_{i=1}^{M}\sum_{j=i+1}^{M} Cov[L_i, L_j]\right) \tag{21}$$

$$= N^2\left(\frac{1}{M} Var[L_i] + \frac{M-1}{M} Cov[L_i, L_j]\right) \tag{22}$$

- The first term is inversely proportional to minibatch size M, but the second term (off diagonal covariances) does not scale by M.
- If we can make the $Cov[L_i, L_j] = 0$, then the variance will be inversely proportional to the minibatch size ($\frac{1}{M}$), leading to better performance.

Naïve Approach

Consider a simple neural network:

- The input to the neural network is a $Mx1000$ matrix $\mathbf{A}$ with M minibatch size and 1000 input feature dimension.
- A single layer of 1000 hidden units. A $1000x1000$ weight matrix $\mathbf{W}$ multiplies the input matrix: $\mathbf{B} = \mathbf{AW}$.
- Approx. posterior on $\mathbf{W}$ is Gaussian: $q_\phi(w_{i,j}) = \mathcal{N}(\mu_{i,j}, \sigma^2_{i,j})$, parameterized as $w_{i,j} = \mu_{i,j} + \sigma_{i,j}\epsilon_{i,j}$ with $\epsilon \sim \mathcal{N}(0, 1)$

Naïve approach to ensure $Cov[L_i, L_j] = 0$:

- Sample a separate weight matrix $\mathbf{W}$ for each training example in the minibatch
- But it's computationally inefficient: Need to sample $Mx1000x1000$ numbers in each minibatch!

Local Reparameterization Trick: Computationally Efficient

The paper proposes the *local reparameterization trick*: Reparameterize from global noise to local noise to sample from an intermediate computation state ($\epsilon \rightarrow f(\epsilon)$

- In the simple neural network example, the weights **W** influence the log likelihood through the pre-activation neurons **B**.
- Instead, sample directly from **B**, requiring only *Mx*1000 numbers
- Example: for a factorized Gaussian posterior on the weights, the posterior for the activations (conditional on the input A) is also factorized Gaussian:

$$q_\phi(w_{i,j}) = \mathcal{N}(\mu_{i,j}, \sigma^2_{i,j})\ \forall w_{i,j}\ \in \mathbf{W} \Longrightarrow q_\phi(b_{m,j}|\mathbf{A}) = \mathcal{N}(\gamma_{m,j}, \delta_{m,j})$$

$$\gamma_{m,j} = \sum_{i=1}^{1000} a_{m,i}\mu_{i,j}, \quad \delta_{m,j} = \sum_{i=1}^{1000} a^2_{m,i}\sigma^2_{i,j}$$

- We parameterize $b_{m,j}$ using: $b_{m,j} = \gamma_{m,j} + \sqrt{\delta_{m,j}}\zeta_{m,j}$ with $\zeta_{m,j} \sim \mathcal{N}(0,1)$, where ζ is a *Mx*1000 matrix.

Local Reparameterization Trick: Even Lower Variance

- The local reparameterization trick also leads to lower variance than naively sampling weight matrices per training example in the minibatch
- Consider the stochastic gradient estimate w.r.t. posterior parameter $\sigma_{i,j}^2$ for minibatch of size $M = 1$. If we draw separate weight matrices $\mathbf{W}$:

$$\frac{\partial L_{\mathcal{D}}^{SGVB}}{\partial \sigma_{i,j}^2} = \frac{\partial L_{\mathcal{D}}^{SGVB}}{\partial b_{m,j}} \frac{\epsilon_{i,j} a_{m,i}}{2\sigma_{i,j}} \tag{23}$$

- If we use the local reparameterization trick:

$$\frac{\partial L_{\mathcal{D}}^{SGVB}}{\partial \sigma_{i,j}^2} = \frac{\partial L_{\mathcal{D}}^{SGVB}}{\partial b_{m,j}} \frac{\zeta_{m,j} a_{m,i}^2}{2\sqrt{\delta_{m,j}}} \tag{24}$$

Experiments

- Comparing variance of gradients on MNIST

stochastic gradient estimator	top layer 10 epochs	top layer 100 epochs	bottom layer 10 epochs	bottom layer 100 epochs
local reparameterization (ours)	7.8×10^3	1.2×10^3	1.9×10^2	1.1×10^2
weight sample per data point (slow)	1.4×10^4	2.6×10^3	4.3×10^2	2.5×10^2
weight sample per minibatch (standard)	4.9×10^4	4.3×10^3	8.5×10^2	3.3×10^2
no dropout noise (minimal var.)	2.8×10^3	5.9×10^1	1.3×10^2	9.0×10^0

Table 1: Average empirical variance of minibatch stochastic gradient estimates (1000 examples) for a fully connected neural network, regularized by variational dropout with independent weight noise.

- Comparing the speed:
  - Drawing separate weight samples per datapoint: 1635 seconds
  - Using local reparameterization trick: 7.4 seconds

  $\Longrightarrow$ an over 200 fold speedup

Sparsification

Sparsification: Overview Sparse parameterization & representation

- Saves memory and improves computational efficiency
- Improves learned representation by better ignoring noise in data

We present:

- Sparsity-inducing priors
- Group sparsity, convolutional neural nets
- Better approximation via non-centered parameterization

Sparsity-inducing priors

- $p(w|\mathcal{D}) \propto p(\mathcal{D}|w)p(w) \implies$ structure of prior could affect posterior
- What kind of priors could encourage sparsity:
  - Having a mean/mode at zero?
  - Having a lot of density near zero?

Sparsity-inducing priors

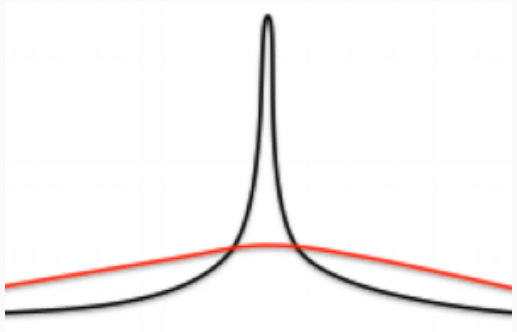
- $p(w|\mathcal{D}) \propto p(\mathcal{D}|w)p(w) \implies$ structure of prior could affect posterior
- What kind of priors could encourage sparsity:
  - Having a mean/mode at zero?
  - Having a lot of density near zero?
- Neither are sufficient as eg. a $\mathcal{N}(0, \sigma^2)$ prior only squashes weights.

Spike and slab priors

Let's imagine we use a very complex model and we believe *a priori* that a fraction of the weights should be zero.

$$w \sim (1 - \beta)\delta_0(w) + \beta\pi(w) \tag{25}$$

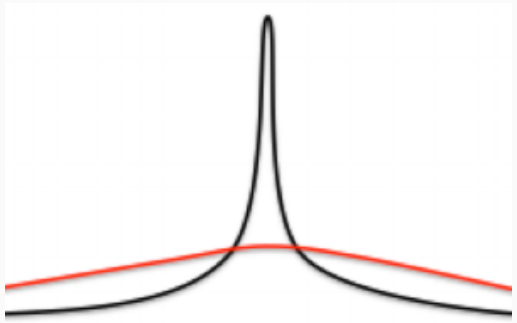
where δ_0 is a peaky distribution at 0 and π a flat distribution.



The slab π in this mixture is important: it allows large values to be accommodated.

Spike and slab priors

- In the extreme case, we use a *dirac delta* and a *uniform* distribution.
- This combination seems to be perceived as a "gold standard", but *neither actually give useful gradient information*, so let's use normals.



Maybe use $\mathcal{N}(0, 0.000001)$ as the spike and $\mathcal{N}(0, 1000000)$ as the slab?

Scale mixture of normals

Instead of a finite mixture, we can take an infinite one. Define:

$$(w|\lambda) \sim \mathcal{N}(w|0, \lambda^2), \qquad \lambda \sim p(\lambda) \tag{26}$$

The marginal distribution of w (integrating λ out) is the mixture of various normal distributions that are centered at zero with different scales

$$p(w) = \int p(\lambda)\mathcal{N}(w|0, \lambda^2)d\lambda \tag{27}$$

Scale mixture of normals defined by different $p(\lambda)$

$$p(w) = \int p(\lambda)\mathcal{N}(w|0, \lambda^2)d\lambda \tag{28}$$

$p(\lambda)$	$p(w)$	Regularization
Dirac delta	Normal	Ridge Regression (L2)
Exponential	Laplacian	LASSO (L1)
Inverse Gamma	Student-t	RVM
Log-normal ($\propto \frac{1}{\lvert\lambda\rvert}$)	Log-normal	
Half-Cauchy	**Horseshoe**	

Table 2: Correspondence of distributions of λ, marginal distributions of w, and regularization schemes.

The horseshoe prior

The horseshoe distribution $p(w)$ has no closed-form equation but behaves essentially like $\log(1 + 2/w^2)$.

- $p(w = 0) = \infty$
- Heavy tail.

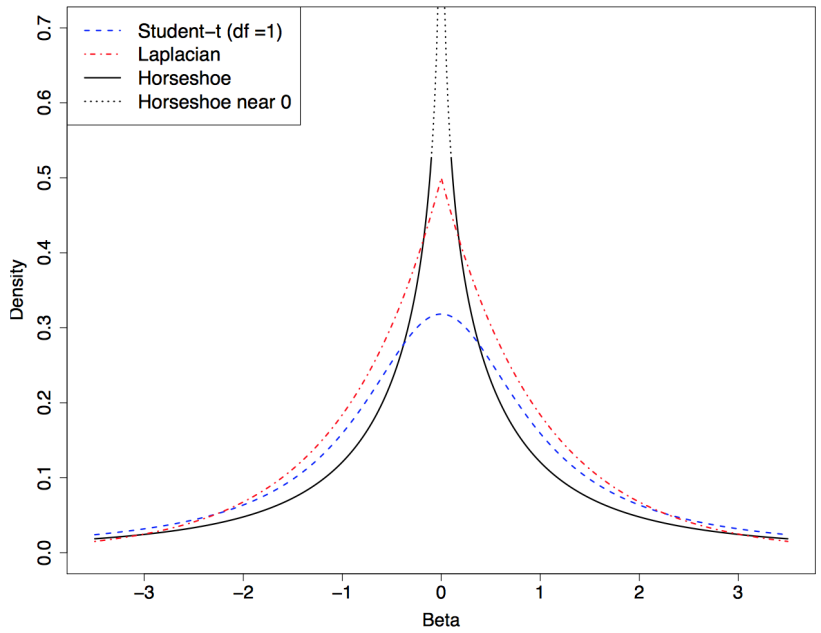
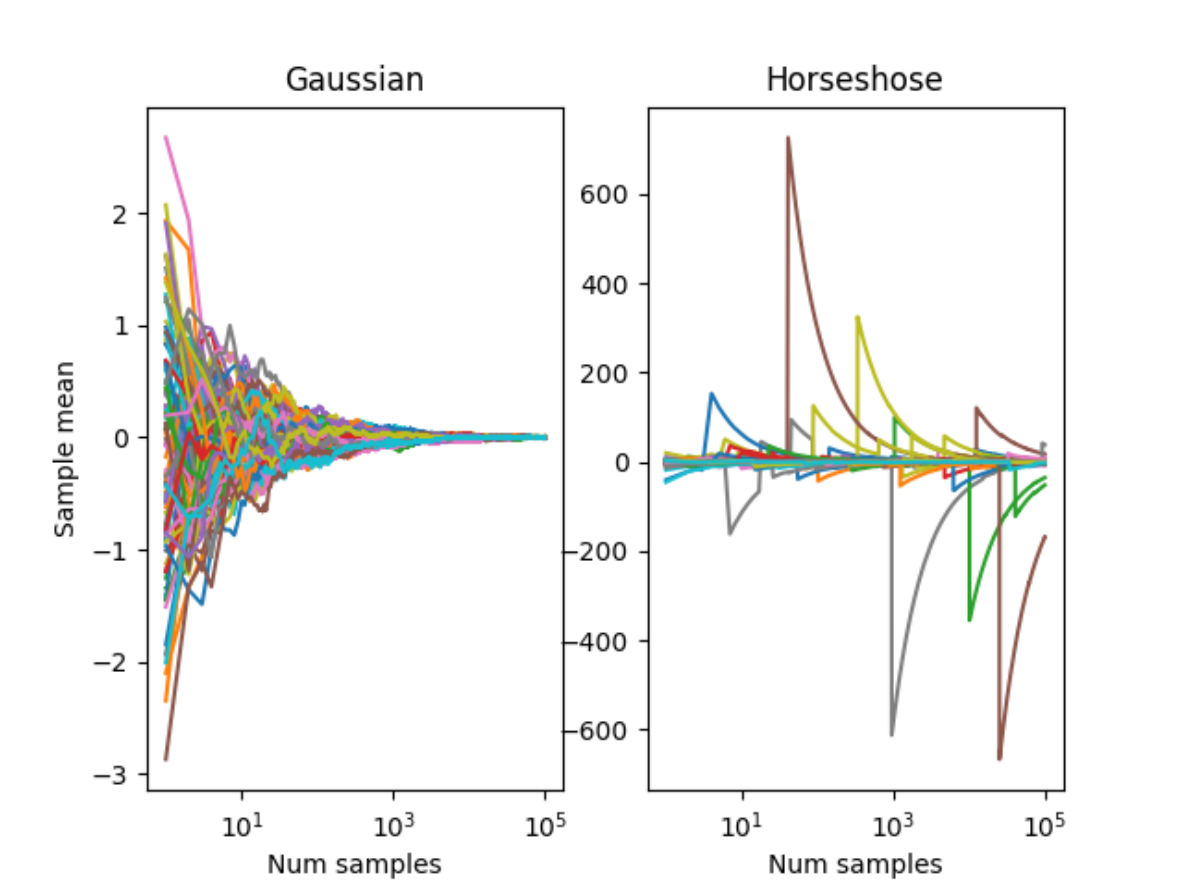


Figure 1: The horseshoe prior and two close cousins: Laplacian and Student-t.

A heavy tail vs. the Law of Large Numbers

Heavy tail: high probability of sampling a large value.



Advantages of Horseshoe

- Horseshoe has both an **infinite spike at zero** and a **heavy tail**.
- Recall $w|\lambda \sim \mathcal{N}(w|0, \lambda^2)$
- Define $\kappa = 1/(1+\lambda)$
- $\lambda \to \infty \implies \kappa \to 0 \implies w \to w^*$
- $\lambda \to 0 \implies \kappa \to 1 \implies w \to 0$
- Horseshoe gives less incentive to interpolate between w^* and 0 compared to Laplacian.

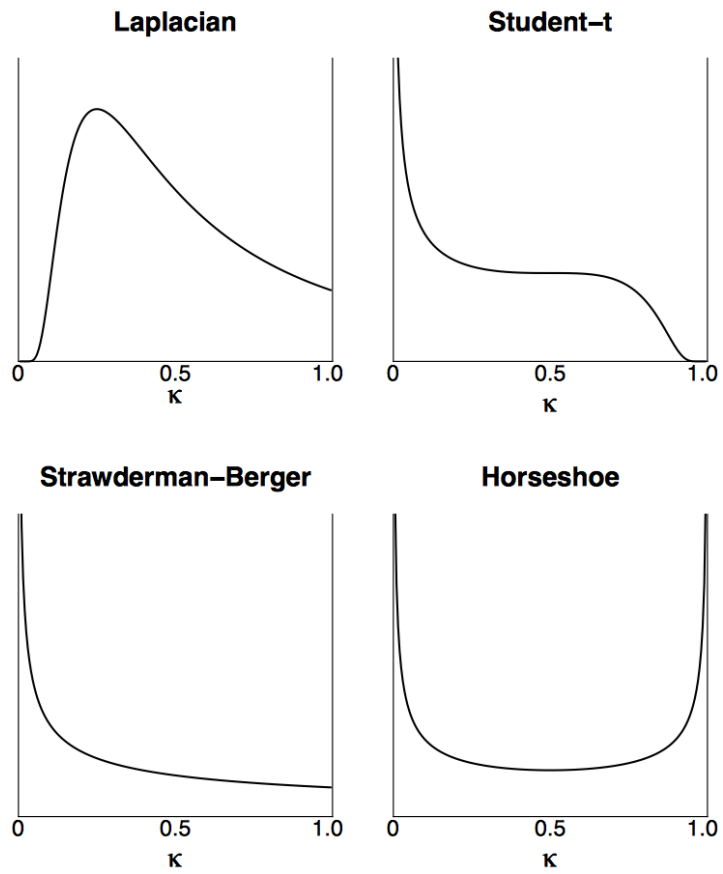
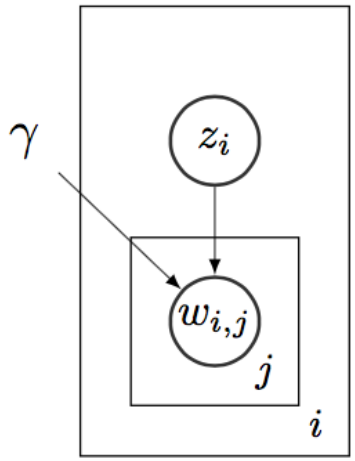


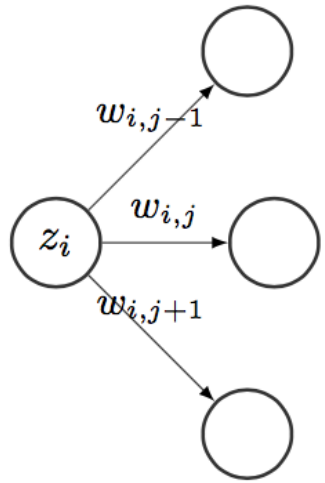
Figure 2: Densities for the shrinkage weights $\kappa_i \in [0, 1]$. $\kappa_i = 0$ means no shrinkage and $\kappa_i = 1$ means total shrinkage to zero.

Group Sparsity in Neural Nets

- Grouping outgoing weights by unit: inducing dependence between outgoing weights from the same hidden unit
- For the i-th hidden, define scale variable z_i (log-normal/half-Cauchy)
- Outgoing weight $w_{i,j}$ has scale-mixture prior with scale z_i
- Define approximate posterior to also factorize in this manner



Graphical Model of Prior



Group Sparsity in Neural Nets

What we need for VI to work:

- Efficient Sampling from approximate posterior
  - Achieved by ancestral sampling. Sample z_i from $q(z_i)$, then sample $w_{i,j}$ from $q(w_{i,j}|z_i)$
- Evaluating the $KL(q(W,Z)||p(W,Z))$
  - $\mathbb{E}_{q(Z)}[KL(q(W|Z)||p(W|Z))] + KL(q(Z)||p(Z))$
  - $KL(q(W|Z)||p(W|Z))$ can be computed in closed form when $q(W|Z)$ and $p(W|Z)$ are Gaussians
  - $KL(q(Z)||p(Z))$ can be computed in closed form when both $q(Z)$ and $p(Z)$ are Gaussian OR $q(Z)$ is half-Cauchy and $p(Z)$ is log-normal
  - Differentiability w.r.t. variational parameters is guaranteed

Group Sparsity in Neural Nets

Other Details:

- Group pruning is determined by simple thresholding using certain statistics of the approximate posterior of z
- Local & global pruning: local to unit, global to layer
- Decomposing half-Cauchy R.V. into product of Inverse Gamma and Gamma R.V.s
- Inferring bit-precision

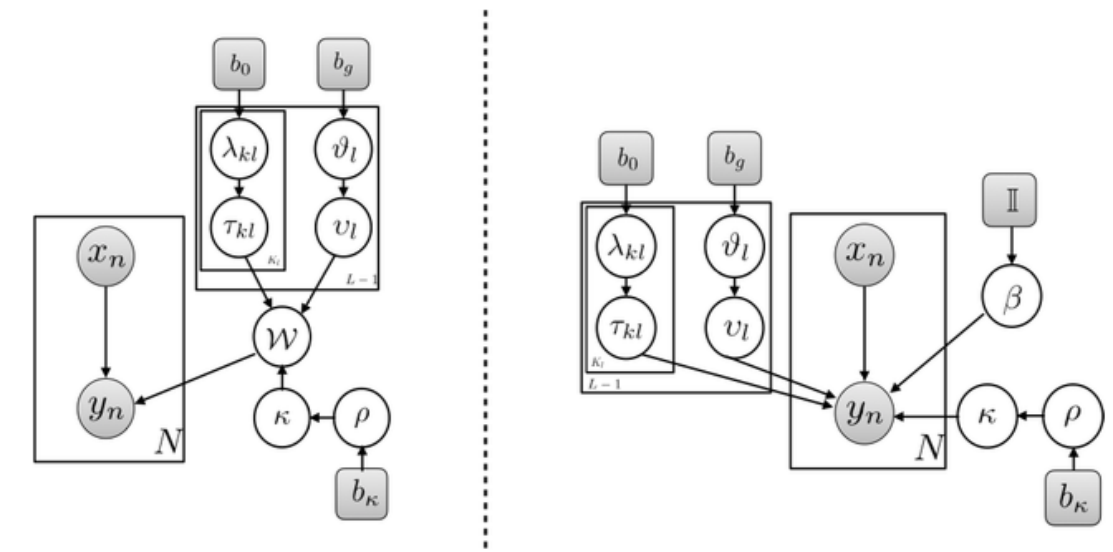
Inferring bit-precision

Using the average marginal variance $\mathcal{V}(w_{i,j})$ across layer, we can infer the unit round off precision.

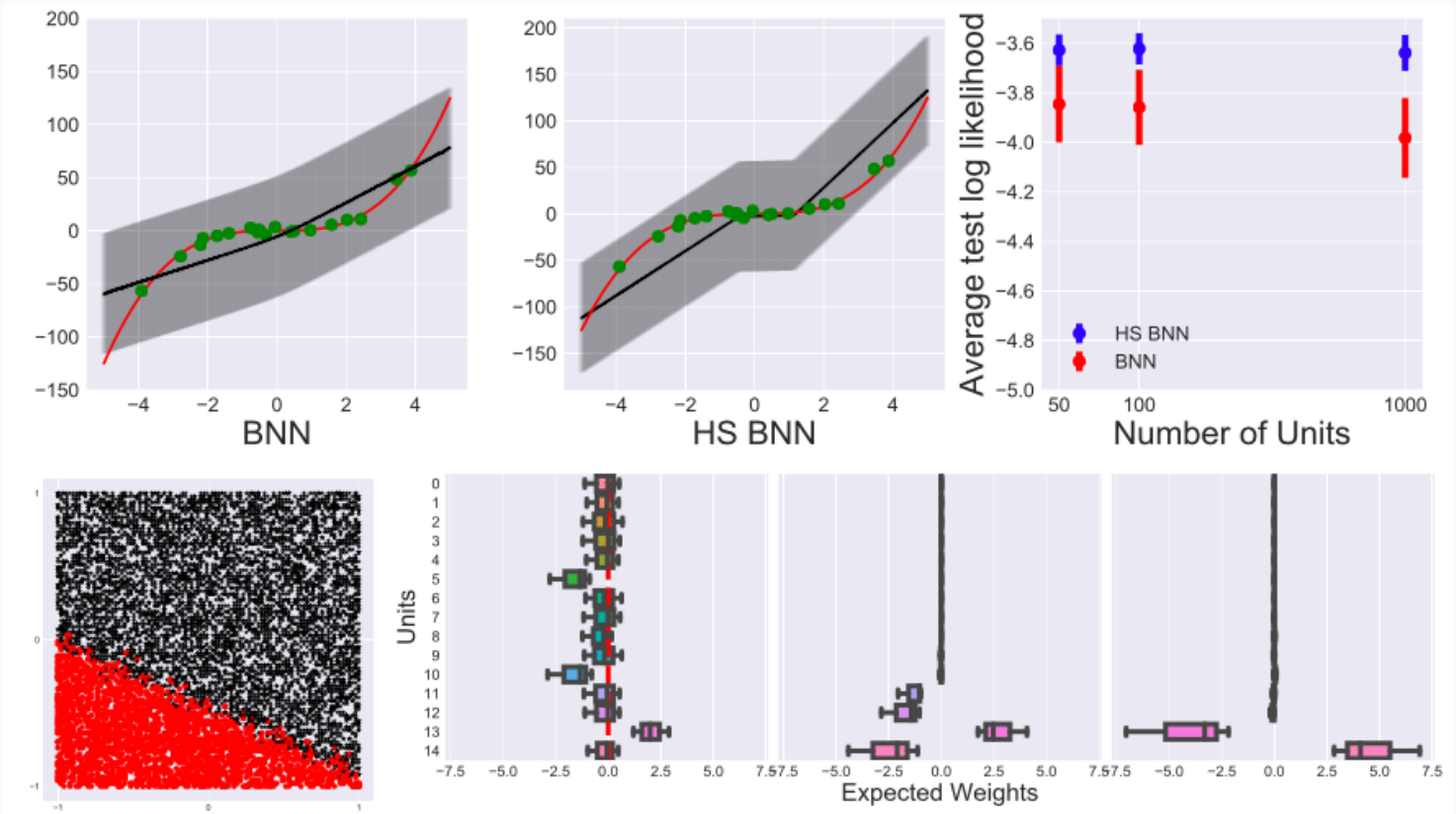
Network & size	Method	Pruned architecture	Bit-precision
LeNet-300-100	Sparse VD	512-114-72	8-11-14
784-300-100	BC-GNJ	278-98-13	8-9-14
	BC-GHS	311-86-14	13-11-10
LeNet-5-Caffe	Sparse VD	14-19-242-131	13-10-8-12
	GD	7-13-208-16	-
20-50-800-500	GL	3-12-192-500	-
	BC-GNJ	8-13-88-13	18-10-7-9
	BC-GHS	5-10-76-16	10-10–14-13
VGG	BC-GNJ	63-64-128-128-245-155-63- -26-24-20-14-12-11-11-15	10-10-10-10-8-8-8- -5-5-5-5-5-6-7-11
(2× 64)-(2× 128)- -(3×256)-(8× 512)	BC-GHS	51-62-125-128-228-129-38- -13-9-6-5-6-6-6-20	11-12-9-14-10-8-5- -5-6-6-6-8-11-17-10

Related Work: Model Selection in Bayesian Neural Networks via Horseshoes Priors

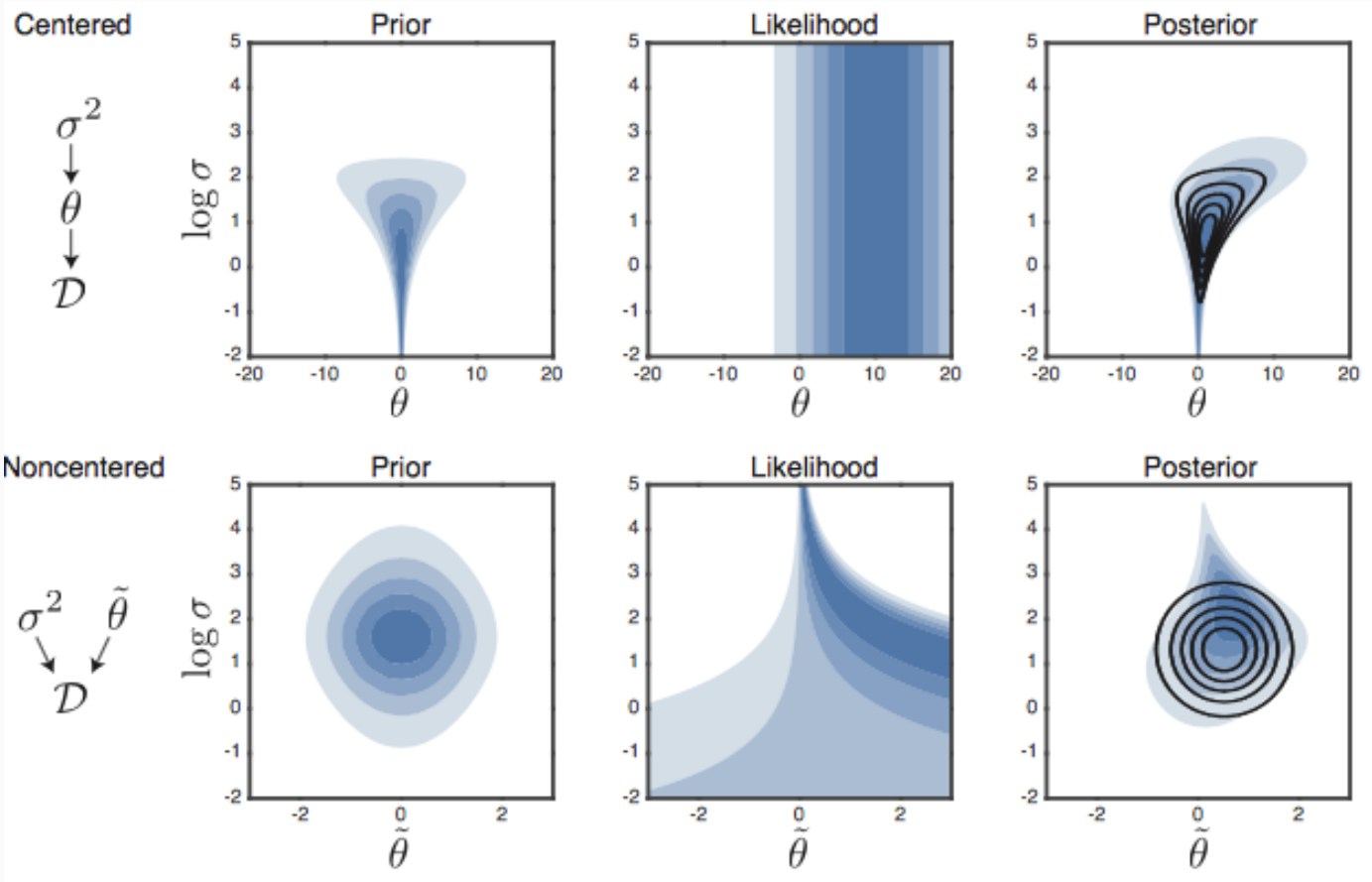
- Induces heavy-tailed priors over network weights using scale mixture of Gaussians
- Induces unit level sparsity by sharing a common prior for all weights incident to same unit



Experimental Results: Centered VS Non-Centered Parameterizations



Centered Vs Non-Centered Parameterization



J. Ingraham and D. Marks, "Variational inference for sparse and undirected models," in *Proceedings of the 34th International Conference on Machine Learning* (D. Precup and Y. W. Teh, eds.), vol. 70 of *Proceedings of Machine Learning Research*, (International Convention Centre, Sydney, Australia), pp. 1607–1616, PMLR, 06–11 Aug 2017.

S. Ghosh and F. Doshi-Velez, "Model Selection in Bayesian Neural Networks via Horseshoe Priors," *ArXiv e-prints*, May 2017.

C. Louizos, K. Ullrich, and M. Welling, "Bayesian Compression for Deep Learning," *ArXiv e-prints*, May 2017.

C. M. Carvalho, N. G. Polson, and J. G. Scott, "Handling sparsity via the horseshoe," in *Proceedings of the Twelth International Conference on Artificial Intelligence and*

Statistics (D. van Dyk and M. Welling, eds.), vol. 5 of *Proceedings of Machine Learning Research*, (Hilton Clearwater Beach Resort, Clearwater Beach, Florida USA), pp. 73–80, PMLR, 16–18 Apr 2009.