Final Presentation:

Dropout as a Bayesian Approximation

Yutaro Yamada

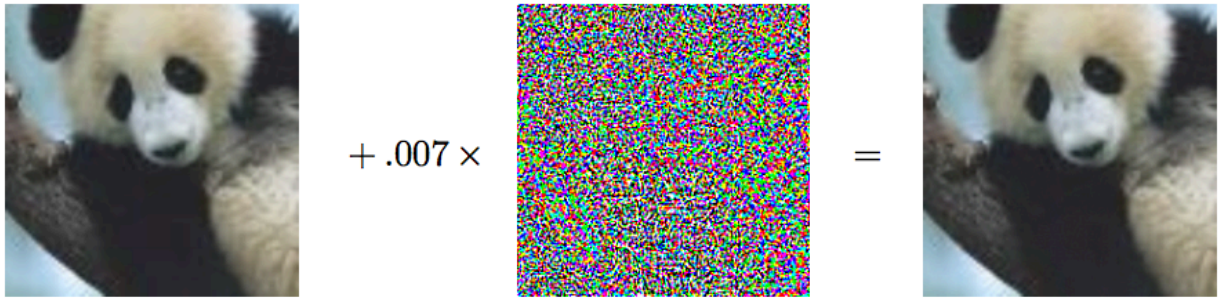
STAT 654

04 May 2017

Background

Neural Networks

–



"Representing Model Uncertainty in Deep Learning"

Dropout as a Bayesian Approximation:
Representing Model Uncertainty in Deep Learning

Yarin Gal YG279@CAM.AC.UK
Zoubin Ghahramani ZG201@CAM.AC.UK
University of Cambridge

Abstract

Deep learning tools have gained tremendous attention in applied machine learning. However such tools for regression and classification do not capture model uncertainty. In comparison, Bayesian models offer a mathematically grounded framework to reason about model uncertainty, but usually come with a prohibitive computational cost. In this paper we develop a new theoretical framework casting dropout training in deep neural networks (NNs) as approximate Bayesian inference in deep Gaussian pro-

With the recent shift in many of these fields towards the use of Bayesian uncertainty (Herzog & Ostwald, 2013; Trafimow & Marks, 2015; Nuzzo, 2014), new needs arise from deep learning tools.

Standard deep learning tools for regression and classification do not capture model uncertainty. In classification, predictive probabilities obtained at the end of the pipeline (the softmax output) are often erroneously interpreted as model confidence. A model can be uncertain in its predictions even with a high softmax output (fig. 1). Passing a point estimate of a function (solid line 1a) through a softmax (solid line 1b) results in extrapolations with unjustified

Outline

1. Background
2. Big picture
3. Gaussian Processes
4. Variational Inference
5. Dropout as a Bayesian Approximation
6. Demo

How to obtain model uncertainty?: Big picture

How to obtain model uncertainty?

- Take a neural network trained with an SRT
- Given some input x^*, Obtain a random output through stochastic forward pass
- Repeat this several times, sampling $i.i.d$ outputs $\{\widehat{y}_1^*(x^*), ..., \widehat{y}_T^*(x^*)\}$
- Calculate the predictive mean and variance:

$$\mathbf{E}[\mathbf{y}^*] \approx \frac{1}{T}\sum_{t=1}^{T} \widehat{y}_1^*(x^*)$$

$$\mathbf{V}[\mathbf{y}^*] \approx \tau^{-1}\mathbf{I}_D + \frac{1}{T}\sum_{t=1}^{T} \widehat{y}_1^*(x^*)^T \widehat{y}_1^*(x^*) - \mathbf{E}[\mathbf{y}^*]^T\mathbf{E}[\mathbf{y}^*]$$

More on uncertainty estimates

$$\{\widehat{\mathbf{y}}_1^*(\mathbf{x}^*), ..., \widehat{\mathbf{y}}_T^*(\mathbf{x}^*)\}$$

$\rightarrow$ Empirical samples from our approximate predictive posterior $p(\mathbf{y}^*|\mathbf{x}^*, \mathbf{X}, \mathbf{Y})$

$$\begin{aligned}
\mathbf{y}_t(\mathbf{x}^*) \sim p(\mathbf{y}^*|\mathbf{x}^*, \mathbf{X}, \mathbf{Y}) &= \int p(\mathbf{y}^*|\mathbf{x}^*, \mathbf{w}) p(\mathbf{w}|\mathbf{X}, \mathbf{Y}) d\mathbf{w} \\
&\approx \int p(\mathbf{y}^*|\mathbf{x}^*, \mathbf{w}) q(\mathbf{w}) d\mathbf{w} \\
&= \frac{1}{T} \sum_{t=1}^{T} p(\mathbf{y}^*|x^*, \mathbf{w}_t), \mathbf{w}_t \sim q_\theta(\mathbf{w})
\end{aligned}$$

Claim:

If we specify $p(w)$ as a Gaussian and $q_\theta(w)$ as a mixture of Gaussians:

$$p(w) = \mathcal{N}(w; 0, \sigma^2 I)$$
$$q_\theta(w) = p_1 \mathcal{N}(w; \mu, \Sigma) + (1 - p_1)\mathcal{N}(w; 0, \sigma^2 \mathbf{I})$$

Then, training a neural network with dropout+L_2 regularization is equivalent to approximate variational inference of posterior predictive distribution obtained from Gaussian process. (= we will obtain the same parameter vector w)

Claim:

Training a neural network with dropout+L_2 regularization (1) is equivalent (2) to approximate variational inference (4) of posterior predictive distribution obtained from Gaussian process (3).

(1) Training a neural network

Goal: Given N training sample pairs $\mathbf{X} \in \mathbf{R}^{N \times Q}$ and $\mathbf{Y} \in \mathbf{R}^{N \times D}$, learn a function f that maps $\mathbf{x} \in \mathcal{X}$ to $\mathbf{y} \in \mathcal{Y}$.

Model: a single-hidden neural network with K hidden units and some non-linear activation $\sigma(\cdot)$

$$\mathbf{y} = f(\mathbf{x}) = \sigma(\mathbf{x}\mathbf{W_1} + \mathbf{b})\mathbf{W_2}$$

where

$$\mathbf{W}_1 \in \mathbf{R}^{Q \times K}, \mathbf{W}_2 \in \mathbf{R}^{K \times D}, \mathbf{b} \in \mathbf{R}^K$$

Assumption: Gaussian observation noise: $y = f(x) + noise$

(1) Training a neural network

Want to learn a function f, parameterized by W <=> find (a point estimate of) W by minimizing loss

Loss function:

$$E = \frac{1}{2N} \sum_{n=1}^{N} \|\mathbf{y}_n - \widehat{\mathbf{y}}_n\|_2^2$$

Run Gradient Descent.

$$\widetilde{\mathbf{w}}_{t+1} := \widetilde{\mathbf{w}}_t + \alpha \nabla L_{\widetilde{\mathbf{w}}_\mathbf{t}}(\hat{\mathbf{y}}, \mathbf{y})$$

Dropout

Given $\mathbf{x}_i$:

- Sample two binary vectors $\mathbf{z_1} \in \mathbf{R}^Q$ and $\mathbf{z_2} \in \mathbf{R}^K$
- Apply element-wise to the input vector $\mathbf{x}_i$ and the intermediate vector $\sigma(\mathbf{x}_i\mathbf{W_1} + \mathbf{b})$ to obtain $\mathbf{x}_i \circ \mathbf{z_1}$ and $\sigma(\mathbf{x}_i\mathbf{W_1} + \mathbf{b}) \circ \mathbf{z_2}$
- (The elements of the vectors $\mathbf{z_1}$ and $\mathbf{z_2}$ are sampled from a Bernoulli distribution with p_1 and p_2, respectively.)
- Calculate the output of the neural network:

$$\begin{aligned}\widehat{\mathbf{y}}_{dropout} &= ((\sigma((\mathbf{x} \circ \mathbf{z_1})\mathbf{W_1} + \mathbf{b})) \circ \mathbf{z_2})\,\mathbf{W_2} \\ &= \sigma((\mathbf{x}(\mathbf{z_1}\mathbf{W_1}) + \mathbf{b})(\mathbf{z_2}\mathbf{W_2}) \\ &= \mathbf{f}^{\widehat{\mathbf{W}}_1, \widehat{\mathbf{W}}_2, \mathbf{b}}(\mathbf{x}) = \sigma(\mathbf{x}\widehat{\mathbf{W}}_1 + \mathbf{b})\widehat{\mathbf{W}}_2\end{aligned}$$

with random variable realizations as weights $\widehat{\mathbf{W}}_1 = \text{diag}(\mathbf{z}_1)\mathbf{W}_1$ and $\widehat{\mathbf{W}}_2 = \text{diag}(\mathbf{z}_2)\mathbf{W}_2$.

Training a neural net with Dropout+L_2 regularization

$$\widetilde{\mathbf{w}}_{t+1} := \widetilde{\mathbf{w}}_t + \alpha \nabla L_{\widetilde{\mathbf{w}}_\mathbf{t}}(\widehat{\mathbf{y}}_{dropout}, \mathbf{y})$$

Loss: $\widehat{\mathcal{L}}_{dropout}(\mathbf{W}_1, \mathbf{W}_2, \mathbf{b}) =$

$$\frac{1}{M} \sum_{i \in S} E^{\widehat{\mathbf{W}}_1, \widehat{\mathbf{W}}_2, \widehat{\mathbf{b}}}(\mathbf{x}_i, \mathbf{y}_i) + \lambda_1 \|\mathbf{W}_1\|_2^2 + \lambda_2 \|\mathbf{W}_2\|_2^2 + \lambda_3 \|\mathbf{b}\|_2^2$$

where S is the minibatch (or a sub-sampling set).

(1) Training a neural net with Dropout+L_2 regularization

By assuming a Gaussian observation noise, we can rewrite the regression loss ($E^{\mathbf{W}_1,\mathbf{W}_2,\mathbf{b}}(\mathbf{x},\mathbf{y})$) as negative log-likelihood:

$$\begin{aligned} E^{\mathbf{W}_1,\mathbf{W}_2,\mathbf{b}}(\mathbf{x},\mathbf{y}) &= \frac{1}{2}\left\|\mathbf{y}-\mathbf{f}^{\mathbf{W}_1,\mathbf{W}_2,\mathbf{b}}(\mathbf{x})\right\|_2^2 \\ &= -\frac{1}{\tau}\log p(\mathbf{y}|\mathbf{f}^{\mathbf{W}_1,\mathbf{W}_2,\mathbf{b}}(\mathbf{x})) + \text{const} \end{aligned}$$

where $p(\mathbf{y}|\mathbf{f}^{\mathbf{W}_1,\mathbf{W}_2,\mathbf{b}}(\mathbf{x})) = \mathcal{N}(\mathbf{y};\mathbf{f}^{\mathbf{W}_1,\mathbf{W}_2,\mathbf{b}}(\mathbf{x}),\tau^{-1}\mathbf{I})$ with τ^{-1} observation noise.

Algorithm

Input: Dataset $\mathbf{X}, \mathbf{Y}$

Output: $\widehat{\theta}$

Initialize θ randomly. Set a learning schedule η.

while θ not converged **do**

Sample M random variables $\hat{\epsilon}_i \sim p(\epsilon)$, S a random subset of $\{1, \ldots, N\}$ of size M.

Calculate stochastic derivative estimator w.r.t. θ:

$$\widehat{\Delta\theta} \leftarrow -\frac{1}{M\tau} \sum_{i \in S} \frac{\partial}{\partial \theta} \log p(\mathbf{y}_i | \mathbf{f}^{g(\theta,\epsilon)}(\mathbf{x}_i))$$

$$+\frac{\partial}{\partial \theta} \left(\lambda_1 \left\| \mathbf{W}_1 \right\|_2^2 + \lambda_2 \left\| \mathbf{W}_2 \right\|_2^2 + \lambda_3 \left\| \mathbf{b} \right\|_2^2 \right)$$

Update θ: $\theta \leftarrow \theta + \eta \widehat{\Delta\theta}$

end while

where $\mathbf{w} = \{\text{diag}(\epsilon_1)\mathbf{W}_1, \text{diag}(\epsilon_2)\mathbf{W}_2, \mathbf{b}\} =: g(\theta, \epsilon)$

(2) Equivalence

$$\mathcal{L}_{VI} := \int q(\omega) \log p(\mathbf{Y}|\mathbf{X}, \omega) d\omega - \mathrm{KL}(q(\omega)||p(\omega))$$

$$\begin{aligned}
\widehat{\mathcal{L}}_{VI}(\theta) &= -\frac{N}{M} \sum_{i \in S} \int q_\theta(\mathbf{w}) \log p(\mathbf{y}_i|\mathbf{f}^{\mathbf{w}}(\mathbf{x}_i)) d\mathbf{w} + \mathrm{KL}(q_\theta(\mathbf{w})||p(\mathbf{w})) \\
&= -\frac{N}{M} \sum_{i \in S} \int p(\epsilon) \log p(\mathbf{y}_i|\mathbf{f}^{g(\theta,\epsilon)}(\mathbf{x}_i)) d\epsilon + \mathrm{KL}(q_\theta(\mathbf{w})||p(\mathbf{w}))
\end{aligned}$$

Run gradient descent with this objective.

Algorithm

Input: Dataset $\mathbf{X}, \mathbf{Y}$

Output: θ which parametrizes $q_\theta(\mathbf{w})$

Initialize θ randomly. Set a learning schedule η.

while θ not converged **do**

Sample M random variables $\hat{\epsilon}_i \sim p(\epsilon)$, S a random subset of $\{1, \ldots, N\}$ of size M.

Calculate stochastic derivative estimator w.r.t. θ:

$$\widehat{\Delta\theta} \leftarrow -\frac{N}{M}\sum_{i \in S} \frac{\partial}{\partial \theta} \log p(\mathbf{y}_i | \mathbf{f}^{g(\theta,\epsilon)}(\mathbf{x}_i)) + \frac{\partial}{\partial \theta}\mathrm{KL}(q_\theta(\mathbf{w})||p(\mathbf{w}))$$

Update θ: $\theta \leftarrow \theta + \eta\widehat{\Delta\theta}$

end while

(2) Equivalence

If:

$$\frac{\partial}{\partial\theta}\mathrm{KL}(q_\theta(\mathbf{w})||p(\mathbf{w})) = \frac{\partial}{\partial\theta}\left(\lambda_1\left\|\mathbf{W}_1\right\|_2^2 + \lambda_2\left\|\mathbf{W}_2\right\|_2^2 + \lambda_3\left\|\mathbf{b}\right\|_2^2\right)$$

Then:

Training a neural network with dropout+L_2 regularization is equivalent to approximate variational inference (4) of posterior predictive distribution obtained from Gaussian process (3).

(3) Gaussian Process

- allow us to model distributions over functions
- completely specified by its mean function and covariance function
- stochastic processes = a collection of random variables; the index set $\mathcal{X}$ is the set of all possible inputs i.e. $\mathbf{R}^Q$
- Gaussian Process = a collection of random variables, any finite number of which have a joint Gaussian distribution
- random variable = the value of $f(x)$ at location $x \in \mathcal{X}$

(3) Gaussian Process

- allow us to ”model distributions over functions”??
- put some probability mass on every possible function in the function space of interest?
- How do we do that?

$\Rightarrow$ Take some input points $X_* \in R^{N \times Q}$ and write out the corresponding covariance matrix using pre-defined covariance function element-wise. Then we generate a random Gaussian vector with this covariance matrix:

$$f_* \sim N(0, K(X_*, K_*))$$

(3) Gaussian Process: Problem Setting

Goal: Given N training sample pairs $\mathbf{X} \in \mathbf{R}^{N \times Q}$ and $\mathbf{Y} \in \mathbf{R}^{N \times D}$, learn a function f that maps $\mathbf{x} \in \mathcal{X}$ to $\mathbf{y} \in \mathcal{Y}$.

Model: Gaussian Process

$$\mathbf{y} = f(\mathbf{x}) \quad (\text{previously:} f(\mathbf{x}) = \sigma(\mathbf{x}\mathbf{W_1} + \mathbf{b})\mathbf{W_2})$$

where $f \sim p(f|X, Y)$. To be precise:

$$p(\mathbf{y}|\mathbf{x}, \mathbf{X}, \mathbf{Y}) = \int_f p(\mathbf{y}|\mathbf{x}, \mathbf{f}) p(\mathbf{f}|\mathbf{X}, \mathbf{Y}) d\mathbf{f}$$

Some issue: How do we compute this?

$$p(f|\mathbf{X}, \mathbf{Y}) \propto p(\mathbf{Y}|\mathbf{X}, f) p(\mathbf{f})$$

(3) Gaussian Process

Do we really need to evaluate the integral $\int_f p(\mathbf{y}|\mathbf{x}, \mathbf{f}) p(\mathbf{f}|\mathbf{X}, \mathbf{Y}) d\mathbf{f}$?
Recall that in GP, any output follows multivariate Gaussians:

$$\begin{bmatrix} \mathbf{Y} \\ \mathbf{y} \end{bmatrix} \sim N\left(0, \begin{bmatrix} K(\mathbf{X}, \mathbf{X}) & K(\mathbf{X}, x) \\ K(x, \mathbf{X}) & K(x, x) \end{bmatrix}\right)$$

So using the properties of multivariate Gaussians: $p(\mathbf{y}|\mathbf{x}, \mathbf{X}, \mathbf{Y}) =$

$$\mathcal{N}\left(K(\mathbf{x}, \mathbf{X})K(\mathbf{X}, \mathbf{X})^{-1}\mathbf{y}, K(\mathbf{x}, \mathbf{x}) - K(\mathbf{x}, \mathbf{X})K(\mathbf{X}, \mathbf{X})^{-1}K(\mathbf{X}, \mathbf{x})\right)$$

$\Rightarrow$ requires the inversion of $N \times N$ matrix $K(\mathbf{X}, \mathbf{X})$. So we do need to evaluate the integral. How can we avoid the computation of $p(f|\mathbf{X}, \mathbf{Y})$? $\Rightarrow$ Variational Inference

(3) Approximate neural net with Gaussian Process

(4) Variational Inference

A method of approximating model posterior which would otherwise be difficult to work with directly.
Steps:

- Define a manageable distirbution $q_\theta(w)$
- minimize $\mathrm{KL}(q_\theta(w)||p(w|\mathbf{X},\mathbf{Y}))$
- Use $q_\theta(w)$ instead of $p(w|\mathbf{X},\mathbf{Y})$ to calculate predictive distribution

$$p(\mathbf{y}_*|\mathbf{X}_*,\mathbf{X},\mathbf{y}) = \int_\omega p(\mathbf{y}_*|\mathbf{X}_*,\omega)p(\omega|\mathbf{X},\mathbf{Y})d\omega$$
$$\Rightarrow q(\mathbf{y}^*|\mathbf{x}^*) = \int_\omega p(\mathbf{y}^*|\mathbf{x}^*,\omega)q(\omega)d\omega$$

(4) Variational Inference

Minimizing $\mathrm{KL}(q_\theta(\omega)||p(\omega|\mathbf{X},\mathbf{Y}))$ is equivalent to maximizing the log evidence lower bound:

$$\mathcal{L}_{VI}(\theta) := \int q_\theta(\omega) \log p(\mathbf{Y}|\mathbf{X},\omega)\, d\omega - \mathrm{KL}(q_\theta(\omega)||p(\omega))$$

Note that evaluating $\int q_\theta(\omega) \log p(\omega|\mathbf{X},\mathbf{Y})\, d\omega$ requires the calculation over the entire dataset. To see this more concretely, we can rewrite it as

$$\int q_\theta(\omega) \log p(\mathbf{Y}|\mathbf{X},\omega)\, d\omega = \sum_{n=1}^{N} \int q_\theta(\omega) \log p(\mathbf{y}_n|\mathbf{f}_\omega(\mathbf{x_n}))\, d\omega$$

(4) Variational Inference

Computational issue:

- huge N? $\rightarrow$ sub-samplining approximation (a.k.a. mini-batch optimization)
- integral? $\rightarrow$ Use some nice Monte-Carlo estimates

MC estimates for integral:

$$\int f(x; w)p(w)dw \approx \sum_{t=1}^{T} f(x; w_t), \text{ where } w_t \sim p(w)$$

(4) Variational Inference

$$\sum_{n=1}^{N} \int q_\theta(\omega) \log p(\mathbf{y}_n | \mathbf{f}_\omega(\mathbf{x_n})) d\omega$$

$$\approx \frac{N}{M} \sum_{i \in S} \int q_\theta(\omega) \log p(\mathbf{y}_i | \mathbf{f}^{g(\theta,\epsilon)}(\mathbf{x}_i))$$

$$\approx \frac{N}{M} \sum_{i \in S} \sum_{k=1}^{K} \log p(\mathbf{y}_i | \mathbf{f}_\omega(\mathbf{x}_i)), \omega \sim q_\theta(\omega)$$

$$\approx \frac{N}{M} \sum_{i \in S} \log p(\mathbf{y}_i | \mathbf{f}_\omega(\mathbf{x}_i)), (\text{ just one sample})$$

Looks good. Now, can we differentiate this w.r.t. θ? (Recall: what we want to do at the end is to optimize $L_{VI}(\theta)$ by gradient descent)
$\mathcal{L}_{VI}(\theta) := \int q_\theta(\omega) \log p(\mathbf{Y}|\mathbf{X}, \omega) d\omega - \mathrm{KL}(q_\theta(\omega)||p(\omega))$

(4) Variational Inference

We can't take the derivative because ω_t is already drawn!
Workaround: let $\omega := g(\theta, \epsilon)$:

$$\begin{aligned}
\frac{N}{M}\sum_{i\in S}\frac{\partial}{\partial\theta}\log p(\mathbf{y}_i|\mathbf{f}^{\omega}(\mathbf{x}_i)) &= \frac{N}{M}\sum_{i\in S}\frac{\partial}{\partial\theta}\log p(\mathbf{y}_i|\mathbf{f}^{g(\theta,\epsilon)}(\mathbf{x}_i)) \\
&= \frac{N}{M}\sum_{i\in S}(\log p(\mathbf{y}_i|\mathbf{f}^{g(\theta,\epsilon)}(\mathbf{x}_i)))'\frac{\partial}{\partial\theta}g(\theta,\epsilon) \\
&= \frac{N}{M}\sum_{i\in S}f'(g(\theta,\epsilon))\frac{\partial}{\partial\theta}g(\theta,\epsilon)
\end{aligned}$$

So, if $g(\theta, \epsilon)$ a deterministic transformation of ϵ s.t. $\omega := g(\theta, \epsilon)$, then we can take the derivative.

Example : If $\omega \sim \mathcal{N}(\mu, \sigma^2)$, then $\omega = g(\theta, \epsilon) := \mu + \sigma\epsilon$ where $\epsilon \sim \mathcal{N}(0, 1)$ and $\theta = \{\mu, \sigma^2\}$ (parameter-free)

(4) Variational Inference

Our approximation of $\mathcal{L}_{VI}$ is now:

$$\widehat{\mathcal{L}}_{MC}(\theta) = -\frac{N}{M}\sum_{i \in S} \log p(\mathbf{y}_i | \mathbf{f}^{g(\theta,\epsilon)}(\mathbf{x}_i)) + \mathrm{KL}(q_\theta(\mathbf{w}) || p(\mathbf{w}))$$

s.t. $\mathbf{E}_{S,\epsilon}[\widehat{\mathcal{L}}_{MC}(\theta)] = \mathcal{L}_{VI}(\theta)$.

If we set $g(\theta, \epsilon)$:

$$\mathbf{w} = \{\mathrm{diag}(\epsilon_1)\mathbf{W}_1, \mathrm{diag}(\epsilon_2)\mathbf{W}_2, \mathbf{b}\} =: g(\theta, \epsilon)$$

where $\theta = \{\mathbf{W}_1, \mathbf{W}_2, \mathbf{b}\}$ and $\epsilon_1 \sim Bern(p_1), \epsilon_2 \sim Bern(p_2)$
Then, we recover the objective function for Dropout neural network (for the first term).

(4) Variational Inference

For VI to result in identical optimisation procedure to that of a dropout NN, the second term also has to match:

$$\frac{\partial}{\partial \theta} \mathrm{KL}(q_\theta(\mathbf{w}) || p(\mathbf{w})) \approx \frac{\partial}{\partial \theta} N\tau(\lambda_1 \|\mathbf{W}_1\|_2^2 + \lambda_2 \|\mathbf{W}_2\|_2^2 + \lambda_3 \|\mathbf{b}\|_2^2)$$

Under what constraint does this hold? This depends on the model specification (the choice of prior $p(\omega)$) and selection of variational distribution $q_\theta(\omega)$.

In our case, we set $p(\omega) = \prod_{i=1}^{L} p(\mathbf{W}_i) = \prod_{i=1}^{L} \mathcal{N}(0, \mathbf{I}/l_i^2)$ (independent normal prior for each layer). What about $q_\theta(\epsilon)$?

(4) Variational Inference

First, by law of total probability, the following holds for any $q_\theta(\omega|\epsilon)$:

$$q_\theta(\omega) = \int_\epsilon q_\theta(\omega|\epsilon)p(\epsilon)d\epsilon$$

In particular, it holds for $q_\theta(\omega|\epsilon) = \delta(\omega - g(\theta, \epsilon))$. We approximate the delta function by a narrow Gaussian. That is,
$\delta(\omega - g(\theta, \epsilon)) \sim \mathcal{N}(\omega; g(\theta, \epsilon), \sigma^2 I)$. Then,

$$\begin{aligned} q_\theta(\omega) &= \int_\epsilon q_\theta(\omega|\epsilon)p(\epsilon)d\epsilon \\ &\approx p_1\mathcal{N}(\omega; g(\theta, \epsilon), \sigma^2 I) + (1 - p_1)\mathcal{N}(\omega; 0, \sigma^2 I) \end{aligned}$$

(Recall: $g(\theta, \epsilon) = \{\text{diag}(\epsilon_1)\mathbf{W}_1, \text{diag}(\epsilon_2)\mathbf{W}_2, \mathbf{b}\}$)

(4) Variational Inference

Proposition 1. *Fix $K, L \in \mathbf{N}$, a probability vector $\mathbf{p} = (p_1, ..., p(L)$ and $\Sigma_i \in \mathbf{R}^{K \times K}$ positive definite for $i = 1, ...L$, with the elements of each Σ_i not dependent on K. Let*

$$q(\mathbf{x}) = \sum_{i=1}^{L} p_i \mathcal{N}(\mathbf{x}; \mu_i, \Sigma_i)$$

be a mixture of Gaussians with L components and $\mu_i \in \mathbf{R}^K$ normally distributed, and let $p(\mathbf{x}) = \mathcal{N}(0, \mathbf{I}_K)$. The KL divergence between $q(\mathbf{x})$ and $p(\mathbf{x})$ can be approximated as:

$$\mathrm{KL}(q(\mathbf{x})||p(\mathbf{x})) \approx \sum_{i=1}^{L} \frac{p_i}{2}(\mathbf{u}_i^T \mathbf{u}_i + tr(\Sigma_i) - K(1 + \log(2\pi)) - \log|\Sigma_i|)$$

plus a constant for large enough K.

(4) Variational Inference

Following this proposition, for large enough K we can approximate the KL divergence term as

$$\begin{aligned}
\mathrm{KL}(q(\mathbf{W}_1)||p(\mathbf{W}_1)) &= Q \cdot \mathrm{KL}(q(\mathbf{w}_q)||p(\mathbf{w}_q)) \text{ (row-wise independence)} \\
&\approx Q \cdot (\frac{p_1}{2}(\mathbf{m}_q^T \mathbf{m}_q + tr(\sigma^2 \mathbf{I}_K) \\
&- K(1 + \log(2\pi)) - \log|\sigma^2 \mathbf{I}_K|) \\
&+ (\frac{1-p_1}{2}(tr(\sigma^2 \mathbf{I}_K) - K(1 + \log(2\pi)) - \log|\sigma^2 \mathbf{I}_K|)) \\
&= QK(\sigma^2 - \log(\sigma^2) - 1) + \frac{p_1}{2}\sum_{q=1}^{Q} \mathbf{m}_q^T \mathbf{m}_q + C
\end{aligned}$$

where C is some constant.

(4) Variational Inference

Note that $\sum_{q=1}^{Q} \mathbf{m}_q^T \mathbf{m}_q = \|\mathbf{M}_1\|_2^2$. So if we ignore the constant terms τ, σ, we obtain the following log-evidecne lower bound:

$$\mathcal{L}_{GP} \propto -\frac{\tau}{2} \sum_{n=1}^{N} \|\mathbf{y}_n - \hat{\mathbf{y}}_n\|_2^2 - \frac{p_1}{2} \|\mathbf{M}_1\|_2^2 - \frac{p_2}{2} \|\mathbf{M}_2\|_2^2 - \frac{1}{2} \|\mathbf{m}\|_2^2$$

After some scaling, we recover the objective function for dropout neural network.

Claim: (revisited)

Training a neural network with dropout+L_2 regularization (1) is equivalent (2) to approximate variational inference (4) of posterior predictive distribution obtained from Gaussian process (3).

Demo: Uncertainty estimates

For each test point x_i, we draw a set of parameters from the dropout approximate posterior $\widehat{\omega}_i \sim q_\theta(\omega)$, and conditioned on these parameters, we drew a prediction from the likelihood $\mathbf{y}_i \sim p(\mathbf{y}|\mathbf{x}_i, \omega)$. Since the predictive distribution has

$$\begin{aligned} p(\mathbf{y}_i|\mathbf{x}_i, \mathbf{X}, \mathbf{Y}) &= \int p(\mathbf{y}_i|\mathbf{x}_i, \omega) p(\omega|\mathbf{X}, \mathbf{Y}) d\omega \\ &\approx \int p(\mathbf{y}_i|\mathbf{x}_i, \omega) q_\theta(\omega) d\omega \\ &=: q_\theta(\mathbf{y}_i|\mathbf{x}_i) \end{aligned}$$

we have that $\mathbf{y}_i$ is a draw from an approximation to the predictive distribution.