1 Notations

Let $f(x;\theta)$ a function of $x \in X$ with parameter $\theta \in \Theta$. The dimension of Θ is N_d. Let $y = f(x;\theta)$ an observable, thus the observed value obeys a Gaussian distribution. Let D denotes a set of observations, $D := \{(x_i, y_i, \sigma_i): i = 1, ..., N_D\}$, wherein x_i is the ith input, y_i its observed value, and σ_i the observational error of y_i. We may employ mini-batch technique, thus denote $D_m := \{(x_i, y_i, \sigma_i): i = 1, ..., N_m\} \subset D$ as a mini-batch, with batch-size $N_m \leqslant N_D$. We use $\mathbb{E}_{f(\theta)}[g(\theta)]$ represent the expectation of function g of a random variable obeys the p.d.f. f. Φ is for Gaussian p.d.f.

Later we will introduce variables a_i, $\mu_{i\alpha}$, and $\zeta_{i\alpha}$, where $i = 1, ..., N_c$ (defined later) and $\alpha = 1, ..., N_d$. Let $z := (a, \mu, \zeta)$; and for $\forall i$ given, $z_i := (a_i, \mu_{i\alpha}, \zeta_{i\alpha})$ for $\forall \alpha$. Define space $Z := \{\forall z\}$; and for $\forall i$ given, define its subspace $Z_i := \{\forall z_i\}$.

2 Neural Network for Posterior (nn4post)

2.1 The Model

Suppose we have some prior on θ, $p(\theta)$, we gain the un-normalized posterior $p(D|\theta)\, p(\theta)$. With D arbitrarily given, this un-normalized posterior is a function of θ, denoted by $p(\theta; D)$[1].[2]

We we are going to do is fit this $p(\theta; D)$ by ANN for any given D. To do so, we have to assume that $\mathrm{supp}\{p(\theta; D)\} = \mathbb{R}^d$ for some $d \in \mathbb{N}^+$ (i.e. has no compact support) but decrease exponentially fast as $\|\theta\| \to +\infty$. With this assumption, $\ln p(\theta; D)$ is well-defined. For ANN, we propose using Gaussian function as the activation-function. Thus, we have the fitting function

$$q(\theta; a, \mu, \zeta) := \sum_{i=1}^{N_c} c_i(a) \left\{ \prod_{\alpha=1}^{d} \Phi(\theta_\alpha - \mu_{i\alpha}, \sigma(\zeta_{i\alpha})) \right\}, \tag{1}$$

where

$$\begin{aligned} c_i(a) &= \frac{\exp(a_i)}{\sum_{j=1}^{N_c} \exp(a_j)} = \mathrm{softmax}(i; a); && (2)\\ \sigma(\zeta_{i\alpha}) &= \ln(1 + \exp(\zeta_{i\alpha})), && (3) \end{aligned}$$

and $a_i, \mu_{i\alpha}, \zeta_{i\alpha} \in \mathbb{R}$ for $\forall i, \forall \alpha$, and

$$\Phi(x; \mu, \sigma) := \sqrt{\frac{1}{2\pi\sigma^2}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2} \right) \tag{4}$$

being the Gaussian p.d.f.. The introduction of ζ is for numerical consideration, see below.

2.1.1 Numerical Consideration

If, in q, we regard w, μ, and σ as independent variables, then the only singularity appears at $\sigma = 0$. Indeed, σ appears in Φ (as well as the derivatives of Φ) as denominator only, while others as numerators. However, once doing numerical iterations with a finite step-length of σ, the probability of reaching or even crossing 0 point cannot be surely absent. This is how we may encounter this singularity in practice.

Introducing the ζ is our trick of avoiding this singularity. Precisely, using a singular map that pushes the singularity to infinity solves the singularity. In this case, using softplus(.) that pushes $\sigma = 0$ to $\zeta \to -\infty$, so that, with finite steps of iteration, singularity (at $-\infty$) cannot be reached.

1. This is why we use " ; " instead of " , ", indicating that D has been (arbitrarily) given and fixed.
2. The normalized posterior $p(\theta|D) = p(D|\theta)\, p(\theta)/p(D) = p(\theta; D)/p(D)$, by Bayes's rule.

This trick (i.e. pushing a singularity to infinity) is the same as in avoiding the horizon-singularity of Schwarzschild solution of black hole.

2.2 Interpretation

2.2.1 As a Mixture Distribution

$q(\theta;\, a,\, \mu,\, \zeta)$ has a probabilistic interpretation. $\prod_{j=1}^{d} \Phi(\theta_j - \mu_{ij},\, \sigma(\zeta_{ij}))$ corresponds to multi-dimensional Gaussian distribution (denote $\mathcal{N}$), with all dimensions independent with each other. The $\{c_i(a)\}$ is a categorical distribution, randomly choosing the Gaussian distributions. Thus $q(\theta; a, \mu, \zeta)$ is a composition: categorical → Gaussian. This is the *mixture distribution*.

2.2.2 As a Generalization

This model can also be interpreted as a direct generalization of mean-field variational inference. Indeed, let $N_c = 1$, this model reduces to mean-field variational inference. Remark that mean-field variational inference is a mature algorithm and has been successfully established on many practical applications.

2.2.3 As a Neural Network

2.3 Marginalization

This model can be marginalized easily. This then benefits the transfering of the model components. Precisely, for any dimension-index β given, we can marginalize all other dimensions directly, leaving

$$q(\theta_\beta; a, \mu, \zeta) = \prod_{\forall \gamma \neq \beta} \int \mathrm{d}\theta_\gamma \sum_{i=1}^{N_c} c_i(a) \left\{ \prod_{\alpha=1}^{d} \Phi(\theta_\alpha; \mu_{i\alpha}, \sigma(\zeta_{i\alpha})) \right\} \tag{5}$$

$$= \sum_{i=1}^{N_c} c_i(a)\, \Phi(\theta_\beta; \mu_{i\beta}, \sigma(\zeta_{i\beta})), \tag{6}$$

where employed the normalization of Φ.

2.4 Loss-Function

We employ "evidence of lower bound" (ELBO)[3]. It is ensured to have a unique global maximum, at which $p(\theta; D) = q(\theta; a, \mu, \zeta)$.

$$\mathrm{ELBO}(a, \mu, \zeta) := \mathbb{E}_{\theta \sim q(\theta; a, \mu, \zeta)}[\ln p(\theta; D) - \ln q(\theta; a, \mu, \zeta)].$$

Since there's no compact support for both $p(\theta; D)$ and $q(\theta; a, \mu, \zeta)$, ELBO is well-defined. The loss-function (or say loss-function, performance, etc) of the fitting is then defined as $\mathcal{L} := -\mathrm{ELBO}$, i.e.

$$\mathcal{L}(a, \mu, \zeta) = -\mathbb{E}_{\theta \sim q(\theta; a, \mu, \zeta)}[\ln p(\theta; D) - \ln q(\theta; a, \mu, \zeta)],$$

or, recall $\mathbb{H}[q] := -\mathbb{E}_q[\ln q]$ for any distribution q,

$$\mathcal{L}(a, \mu, \zeta) = -\mathbb{E}_{\theta \sim q(\theta; a, \mu, \zeta)}[\ln p(\theta; D)] - \mathbb{H}[q(\theta; a, \mu, \zeta)].$$

3. The relation between ELBO and KL-divergence is that $\mathrm{ELBO} = -\mathrm{KL}(q \| p) + \mathrm{Const}$.

2.5 Relation with the Traditional MAP Loss

$\theta_* = \mathrm{argmin}_\theta \{-\ln p(\theta; D)\}$.

Set $N_c = 1$ and for $\forall \alpha$ $\zeta_\alpha \rightarrow -\infty$ so that $\sigma_\alpha(\zeta) \rightarrow 0$, we get $q(\theta; z) \rightarrow \delta(\theta - z)$.

$$\begin{aligned}
\mathcal{L}(z) &= -\mathbb{E}_{\theta \sim q(\theta; z)}[\ln p(\theta; D) - \ln q(\theta; z)] \\
&= -\int \mathrm{d}\theta q(\theta; z) [\ln p(\theta; D) - \ln q(\theta; z)] \\
&= -\int \mathrm{d}\theta \delta(\theta - z) [\ln p(\theta; D) - \ln \delta(\theta - z)] \\
&= -\ln p(z; D) + \mathrm{Const} \\
&= \mathcal{L}_{\mathrm{MAP}}(z) + \mathrm{Const}
\end{aligned}$$

2.6 Relation between Relative Error of Inference and Loss

For the arbitrary model $y = f(x; \theta)$, for $\forall x$, Bayesian inference gives prediction, $\langle f \rangle(x)$, as

$$\langle f \rangle(x) := \mathbb{E}_{\theta \sim p(\theta | D)}[f(x; \theta)].$$

Since $q(.; z)$ (z as the parameter of q) is an approximation to $p(.|D)$, let $\langle f \rangle_q(x) := \mathbb{E}_{\theta \sim q(\theta; z)}[f(x; \theta)]$, then the difference between them is $\delta \langle f \rangle(x) := \langle f \rangle(x) - \langle f \rangle_q(x)$.

Theorem 1. *We have the relation of order between the relative error of inference and loss*

$$\frac{\delta \langle f \rangle}{\langle f \rangle}(x) \sim \mathcal{L}.$$

Proof. By definition, $\delta \langle f \rangle(x) = \int \mathrm{d}\theta f(x; \theta) [p(\theta | D) - q(\theta)]$. Thus

$$\begin{aligned}
\delta \langle f \rangle(x) &= \int \mathrm{d}\theta q(\theta)\, f(x; \theta) \frac{p(\theta | D) - q(\theta)}{q(\theta)} \\
&= \int \mathrm{d}\theta q(\theta) \left(\frac{p(\theta | D)}{q(\theta)} - 1 \right) \\
&= \mathbb{E}_{\theta \sim q(\theta)} \left[f(x; \theta) \left(\frac{p(\theta | D)}{q(\theta)} - 1 \right) \right].
\end{aligned}$$

Then, we have the relation of order

$$\frac{\delta \langle f \rangle}{\langle f \rangle}(x) \sim \mathbb{E}_{\theta \sim q(\theta)} \left[\frac{p(\theta | D)}{q(\theta)} - 1 \right]$$

On the other hand, if $p(.|D) \approx q(.)$ as we expect for $q(.)$, then we have

$$\ln \left(\frac{p(\theta | D)}{q(\theta)} \right) = \ln \left(\left[\frac{p(\theta | D)}{q(\theta)} - 1 \right] + 1 \right) \approx \frac{p(\theta | D)}{q(\theta)} - 1.$$

Thus,

$$\frac{\delta \langle f \rangle}{\langle f \rangle}(x) \sim \mathbb{E}_{\theta \sim q(\theta)} \left[\frac{p(\theta | D)}{q(\theta)} - 1 \right] \approx \mathbb{E}_{\theta \sim q(\theta)} \left[\ln \left(\frac{p(\theta | D)}{q(\theta)} \right) \right] = \mathcal{L}. \qquad \square$$

3 Optimization

3.1 ADVI

Automatic differentation variational inference (ADVI)[4] has the advantage that the variance of its Monte Carlo integral is orderly smaller than that of black box variational inference (i.e. optimization directly using ELBO without further reparameterization).

3.1.1 Derivation

Precisely, recall $\mathbb{E}$ for mean value, Φ for Gaussian p.d.f., $\sigma(.)$ for softplus function, $c(.)$ for softmax function, and

$$q(\theta; a, \mu, \zeta) = \sum_{i=1}^{N_c} c_i(a)\, \Phi(\theta; \mu_i, \sigma(\zeta_i)), \tag{7}$$

we have, for any function f,

$$\begin{aligned}
\mathbb{E}_{q(\theta; a, \mu, \zeta)}[f(\theta)] &= \int \mathrm{d}\theta \sum_{i=1}^{N_c} c_i(a)\, \Phi(\theta; \mu_i, \sigma(\zeta_i))\, f(\theta) && (8)\\
&= \sum_{i=1}^{N_c} c_i(a) \int \mathrm{d}\theta\, \Phi(\theta; \mu_i, \sigma(\zeta_i))\, f(\theta) && (9)\\
&= \sum_{i=1}^{N_c} c_i(a)\, \mathbb{E}_{\Phi(\theta; \mu_i, \sigma(\zeta_i))}[f(\theta)]. && (10)
\end{aligned}$$

With this general relation, we get

$$\begin{aligned}
\mathcal{L}(a, \mu, \zeta) &= -\{\mathbb{E}_{q(\theta; a, \mu, \zeta)}[\ln p(\theta; D)] - \mathbb{E}_{q(\theta; a, \mu, \zeta)}[\ln q(\theta; a, \mu, \zeta)]\} && (11)\\
&= -\sum_{i}^{N_c} c_i(a)\, \mathbb{E}_{\Phi_i(\theta; \mu_i, \sigma(\zeta_i))}[\ln p(\theta; D) - \ln q(\theta; a, \mu, \zeta)] && (12)
\end{aligned}$$

Then, for $\forall i = 1, ..., N_c$, $\forall \alpha = 1, ..., N_d$, let

$$\eta_\alpha := \frac{\theta_\alpha - \mu_{i\alpha}}{\sigma(\zeta_{i\alpha})}, \tag{13}$$

we have

$$\theta_\alpha = \sigma(\zeta_{i\alpha})\, \eta_\alpha + \mu_{i\alpha} \tag{14}$$

(or $\theta = \sigma(\zeta_i)\, \eta + \mu_i$ if hide the α index). So, for any i-component, we transform

$$\begin{aligned}
\theta &\rightarrow \sigma(\zeta_i)\, \eta + \mu_i; && (15)\\
\mathbb{E}_{\Phi(\theta; \mu_i, \sigma(\zeta_i))}[f(\theta)] &\rightarrow \mathbb{E}_{\Phi(\eta; 0, 1)}[f(\sigma(\zeta_i)\, \eta + \mu_i)], && (16)
\end{aligned}$$

where function f is arbitrary, thus holds for both $\ln p(.; D)$ and $\ln q(.; a, \mu, \zeta)$.

With this setting, the derivatives to μ and to ζ are completely independent of $\mathbb{E}[.]$. And now, the loss function becomes

$$\mathcal{L}(a, \mu, \zeta) = -\sum_{i}^{N_c} c_i(a)\, \mathbb{E}_{\Phi(\eta; 0, 1)}[\ln p(\sigma(\zeta_i)\, \eta + \mu_i; D) - \ln q(\sigma(\zeta_i)\, \eta + \mu_i; a, \mu, \zeta)].$$

4. See, Kucukelbir, et al, 2016.

3.2 Redefination of Gradients

3.2.1 Gauge Fixing

Let Δt the learning-rate. Then the updation of a_i at one iteration by gradient decent method is

$$\Delta a_i = -\frac{\partial \mathcal{L}}{\partial a_i}(a, \mu, \zeta)\, \Delta t.$$

Notice that redefining the $\partial \mathcal{L} / \partial a$ by

$$\frac{\partial \mathcal{L}}{\partial a_i}(a, \mu, \zeta) \rightarrow \frac{\partial \mathcal{L}}{\partial a_i}(a, \mu, \zeta) + C,$$

where C can be any constant, leaves the updation of $c_i(a)$ invariant, since it makes

$$\Delta a_i \rightarrow -\frac{\partial \mathcal{L}}{\partial a_i}(a, \mu, \zeta)\, \Delta t - C\, \Delta t,$$

thus

$$c_i(a + \Delta a) \rightarrow \frac{\exp\,(a_i + \Delta a_i - C\,\Delta t)}{\sum_j \exp\,(a_j + \Delta a_j - C\,\Delta t)} = \frac{\exp\,(a_i + \Delta a_i)}{\sum_j \exp\,(a_j + \Delta a_j)} = c_i(a + \Delta a).$$

This C thus provides an additional dof.[5] We can tune the value of C so that the updation of a_i is numerically stable. Indeed, let C be the average of $\{\partial \mathcal{L} / \partial a_i : i = 1, ..., N_c\}$, we find a pretty stability of a as well as a pretty accuracy of c in the iteration process of optimization, as the experiment on Gaussian mixture model shows.

This motives us to, instead of modifying gradients, re-write $\mathcal{L}$ by replacing the c in it by

$$c_i(a) \rightarrow c_i\left(a - \frac{\sum_j^{N_c} a_j}{N_c}\right).$$

Thus

$$\frac{\partial c_i}{\partial a_k}(a) \rightarrow \partial_k c_i\left(a - \frac{\sum_j^{N_c} a_j}{N_c}\right) - \frac{1}{N_c} \sum_k^{N_c} \partial_k c_i\left(a - \frac{\sum_j^{N_c} a_j}{N_c}\right).$$

These two approaches are almost the same. But when N_d is great enough, the difference between them raises. Indeed, experiments on Gaussian mixture distribution (as target) shows that the later converges apperately faster than the first.[6] Additionally, the second approach provides stability for softmax function, since the input of softmax is regular no matter how great the a is. So, we will use the later approach, i.e. modify the relation $c_i(a)$ in loss directly.

3.2.2 Re-scaling of a

In the optimization process, the scales of searching region of a and of μ and ζ may be different in order. So, there shall be an additional hyper-parameter for the re-scaling of a. The re-scaling factor, constant r, redefines

$$c_i(a) := \mathrm{softmax}(i, r a).$$

5. As CL explained, the cs have less dofs as they look, since $\sum_i^{N_c} c_i = 1$. This restriction can provides an additional gauge. And the new dof C fixes this gauge.

6. Why so?

Tuning this additional hyper-parameter can "normalize" a to the same order of scale as μ and ζ, thus may improve the optimization.

This rescaling, if dynamically (e.g. set r as `tf.placeholder` in TensorFlow), also helps fasten the speed of convergence. Indeed, especially with a large N_d, the searching of targets μ and ζ lasts longer, so that a, as nearly random moving at this epoch, can be greatly dispersed, i.e. $\max(a) \gg \min(a)$. As a result, when the targets μ and ζ have been reached, it needs extremely large number of iterations for a so that the target value (generally not so dispersed) can be reached. However, if r is inserted and tuned dynamically, setting $r \to 0$ at the early searching (of targets μ and ζ) epoch, and then setting $r \to 1$ after variables μ and ζ becoming slowly varying, meaning that the their targets have been reached. This thus largely speed up the convergence.

3.2.3 Frozen-out Problem

Generally we hope that the gradients diminish when and only when the optimization converges. However, even far from convergence, a tiny c_i will diminish all the derivatives in the i-component, e.g. derivatives of a_i, $\mu_{i\alpha}$, $\zeta_{i\alpha}$, since all these derivatives are proportional to c_i.

This problem can be solved by replacing, in the gradients, that

$$\frac{\partial \mathcal{L}}{\partial z_i} \to \frac{\partial \mathcal{L}}{\partial z_i} \frac{1}{c_i(a) + \epsilon},$$

where ϵ is a tiny number for numerial stability as usual[7]. This is valid since $c_i(a)$ are all positive. This modifies the direction of gradients in the space Z, but holds the same diection in each i-subspace Z_i individually. And if $(\partial \mathcal{L}/\partial z_i)/(c_i(a) + \epsilon) = 0$, we will have $\partial \mathcal{L}/\partial z_i = 0$, meaning that both gradients leads to the same converge-point on the space Z. So, this modification speeds up the convergence without changing the converge-point.

Generally, we can set

$$\frac{\partial \mathcal{L}}{\partial z_i} \to \frac{\partial \mathcal{L}}{\partial z_i} \left(\frac{1}{c_i(a) + \epsilon} \right)^{\beta},$$

where $\beta \in [0, 1]$. If $\beta = 1$, then back to the previous case; and if $\beta = 0$, then $\partial \mathcal{L}/\partial z_i$ transforms nothing. Running β in range $[0, 1]$ then smoothly waving the transformation of $\partial \mathcal{L}/\partial z_i$.

In TensorFlow, ϵ is usually set as `1e-08`[8]. However, the $c(a)$ can reach the order `1e-24` in practice. (The reason why ϵ cannot be vanished is in footnote.) So the frozen-out problem can still remain, since even though transform as $\partial \mathcal{L}/\partial z \propto c(a) \sim 10^{-24} \to \partial \mathcal{L}/\partial z \propto c(a)/(c(a) + \epsilon) \approx c(a)/\epsilon \sim 10^{-16}$, $\partial \mathcal{L}/\partial z$ is extremely tiny still. This can be solved by additionally clipping $c(a)$ by ϵ as the minimal value. Explicitly, after

```
a_mean = tf.reduce_mean(a, name='a_mean')  # for gauge fixing.
c = tf.softmax(r * (a - a_mean), name='c')  # rescaling of 'a'.
```

additionally set (notice $c(a) < 1$ always)

```
c = tf.clip_by_value(c, _EPSILON, 1, name='c_clipped')
```

Or instead directly clipping on a? Indeed we can, but by clipping the gradient of a, instead of a itself. What we hope is that

$$c_i(a) \equiv \frac{\exp\left(a_i - \sum_k^{N_c} a_k/N_c\right)}{\sum_j^{N_c} \exp\left(a_j - \sum_k^{N_c} a_k/N_c\right)} = \frac{\exp(a_i)}{\sum_j^{N_c} \exp(a_j)} \geqslant \epsilon$$

7. You may wonder why not set $\epsilon = 0$, since $c(a)$ is always non-vanishing. This concerns with the numerical instability in practice. Indeed, in TensorFlow, letting $\epsilon = 0$ causes `NaN` after about 1000 iterations.

8. C.f. https://www.tensorflow.org/api_docs/python/tf/keras/backend/epsilon.

for some ϵ as the "accuracy of c" (thus named as `_C_ACCURACY` in code), which may different from the previous ϵ (i.e. the `_EPSILON`) for numerical stability in dividing, but shall have `_C_ACCURACY > _EPSILON`.

gives

$$a_i \geqslant \ln(\epsilon) + \ln\left(\sum_j^{N_c} \exp(a_j)\right).$$

To ensure this, for some ϵ and some a given, define

$$a_{\min} := \ln(\epsilon) + \ln\left(\sum_j^{N_c} \exp(a_j)\right),$$

if $a_i < a_{\min}$ and $\partial\mathcal{L}/\partial a_i > 0$ (i.e. wants to decrease itself[9]) at some iteration (with a small $|a_i - a_{\min}|$), then in the next iteration, clip $\partial\mathcal{L}/\partial a_i \to 0$. Then the a_i will be "frozen" in the next iteration, until it wants to increase itself (i.e. when $\partial\mathcal{L}/\partial a_i < 0$).

Problem 1. But the un-frozen a_js can increase themselves, thus increases the $a_{\min}$. So, if the frozen a_i keeps $\partial\mathcal{L}/\partial a_i > 0$, then the minimal value of $c_i(a)$ cannot be bounded lowerly.

Comparing with clipping of $c(a)$, clipping of a additionally benefits that it naturally avoids the problem mentioned in the section "Re-scaling of a": early random searching makes a dispersed, thus enlarges the elapsed time of convergence after reached the targets μ and ζ. Indeed, by clipping, a becomes centered, even in the early random seaching epoch.

3.3 Approximations

Comparing to the traditional MAP approach, using multi-peak mixture model makes the $\mathbb{H}(q)$ complicated, especially in the optimization process.

3.3.1 Entropy Lower Bound

Consider any mixture distribution with p.d.f. $\sum_i^{N_c} c_i\, q_i$ where cs are the categorical probabilities and q_i the p.d.f. of the component distributions of the mixture.

$$\mathbb{H}\left[\sum_i^{N_c} c_i\, q_i\right] \geqslant \sum_i^{N_c} c_i\, \mathbb{H}[q_i].$$

So, if define

$$\mathcal{L}'(a, \mu, \zeta) := -\sum_i^{N_c} c_i(a)\,\{\mathbb{E}_{\Phi(\eta;0,1)}[\ln p(\sigma(\zeta_i)\,\eta + \mu_i; D)] + \mathbb{H}[\Phi(\theta, \mu_i, \sigma(\zeta_i))]\}, \tag{17}$$

then we have

$$\mathcal{L}' \geqslant \mathcal{L} \geqslant \min(\mathcal{L}) > -\infty,$$

thus $\mathcal{L}'$ has an global minimum. In this way, the entropy part becomes completely analytic (and simple).

However, as experiment on Gaussian mixture model shows, using entropy lower bound cannot get the enough accuracy as using entropy does. We will not use this approximation.

9. Remind that generally a variable z decreases iff $\partial\mathcal{L}/\partial z > 0$.

3.4 Stochastic Optimization

3.4.1 Difference between Bayesian and Traditional Methods

Suppose, instead of use the whole dataset, we employ mini-batch technique. Since all data are independent, if suppose that D_m is unbiased in D, then we have,

$$\ln p(D|\theta) = \sum_D p((x_i, y_i, \sigma_i)|\theta) \approx \frac{N_D}{N_m} \sum_{D_m} p((x_i, y_i, \sigma_i)|\theta) = \frac{N_D}{N_m} \ln p(D_m|\theta). \tag{18}$$

Then,

$$\ln p(\theta; D) = \ln p(D|\theta) + \ln p(\theta) = \frac{N_D}{N_m} \ln p(D_m|\theta) + \ln p(\theta), \tag{19}$$

thus as previous

$$\ln p(\theta; D) = \frac{N_D}{N_m} \sum_{(x_i, y_i, \sigma_i) \in D_m} \left\{ -\frac{1}{2} \ln (2\pi\sigma_i^2) - \frac{1}{2} \left(\frac{y_i - f(x_i; \theta)}{\sigma_i} \right)^2 \right\} + \ln p(\theta). \tag{20}$$

In this we meet one of the main differences between the Bayesian and the traditional. In the traditional method, N_D does not matters in training, being absent in the optimizer. However, in Bayesian, the number of data that are employed is encoded into Bayesian model, and has to, since the greater number of data gives more confidence. So, while using stochastic optimization in Bayesian mode, the factor N_D/N_m of likelihood has to be taken into account. We have to know how many data we actually have, thus how confident we are.

4 Prediction

Theorem 2. *Let $p(\theta|D)$ the posterior, and $\tilde{p}(\theta|D) = Cp(\theta|D)$ for arbitrary contant C, but unknown. For the arbitrary "observable" $y = g(x;\, \theta)$, for $\forall x$, Bayesian inference gives prediction $\langle g \rangle(x)$ defined as*

$$\langle g \rangle(x) := \int \mathrm{d}\theta p(\theta|D)\, g(x; \theta)$$

can be computed by the trained distribution $q(\theta; z_*)$ ($z := (a, \mu, \zeta)$ for short, and star notation for representing the trained) by weighted Monte-Carlo integral

$$\langle g \rangle(x) \approx \sum_{s=1}^{N_s} [w(\theta_s; z_*, D) \times g(x; \theta_s)],$$

with

$$w(\theta_s; z_*, D) := \mathrm{softmax}_s(\eta)$$

where

$$\eta_s := \ln \tilde{p}(\theta_s|D) - \ln q(\theta_s; z_*),$$

and $\{\theta_s\colon s = 1, ..., N_s\}$ sampled from $q(\theta; z_*)$. The error of the approximation can be estimated as $\mathcal{O}(1/\sqrt{N_s})$ and is independent of the dimension of the space of θ.

Proof. Since

$$\int \mathrm{d}\theta\, \tilde{p}(\theta|D) = C \times \int \mathrm{d}\theta p(\theta|D) = C \times 1$$

and

$$\int \mathrm{d}\theta\, \tilde{p}(\theta|D) = \int \mathrm{d}\theta q(\theta; z_*) \frac{\tilde{p}(\theta|D)}{q(\theta; z_*)} \approx \sum_{s=1}^{N_s} \frac{\tilde{p}(\theta_s|D)}{q(\theta_s; z_*)}$$

where $\{\theta_s : s = 1, ..., N_s\}$ sampled from $q(\theta; z_*)$, we get

$$C = \sum_{s=1}^{N_s} \frac{\tilde{p}(\theta_s|D)}{q(\theta_s; z_*)}.$$

Then,

$$\begin{aligned}\int \mathrm{d}\theta p(\theta|D)\, g(x;\theta) &= \frac{1}{C} \int \mathrm{d}\theta\, \tilde{p}(\theta|D)\, g(x;\theta) \\ &= \frac{1}{C} \int \mathrm{d}\theta q(\theta; z_*) \frac{\tilde{p}(\theta|D)}{q(\theta; z_*)}\, g(x;\theta) \\ &\approx \frac{1}{C} \times \sum_{i=1}^{N_s} \left[\frac{\tilde{p}(\theta_s|D)}{q(\theta_s; z_*)}\, g(x;\theta_i) \right].\end{aligned}$$

Denoting

$$\eta_s := \ln \frac{\tilde{p}(\theta_s|D)}{q(\theta_s; z_*)},$$

and with some direct re-arrangements, we get

$$\int \mathrm{d}\theta p(\theta|D)\, g(x;\theta) \approx \sum_{s=1}^{N_s} \left[\frac{\exp(\eta_s)}{\sum_{s'=1}^{N_s} \exp(\eta_{s'})} \times g(x;\theta_s) \right]$$

which is what we want. In this proof, all the approximations come from Monte-Carlo integrals, whose error can thus be estimated as $\mathcal{O}(1/\sqrt{N_s})$ which is independent of the dimension of the space of θ. □

Notice that we did not compute the mean value $g(x;\theta)$ directly, as $\mathbb{E}_{\theta \sim q(\theta; a_*, \mu_*, \zeta_*)}[g(x;\theta)]$, which is not accurate enough since $q(\theta; z_*)$ is just an approximation to $p(\theta|D)$. Instead, our implementation does help gain the accurate enough result of the Monte-Carlo integral, at the same time avoids the non-convergence of MCMC by "sampling by importance".

5 Deep Learning

It cannot solve the vanishing gradient problem of deep neural network, since this problem is intrinsic to the posterior of deep neural network. Indeed, the posterior has the shape like $\exp(-x^2/\sigma^2)$ with $\sigma \to 0$, where x is the variable (argument) of the posterior. It has a sharp peak, located at a tiny area, with all other region extremely flat. The problem of find this peak, or equivalently, finding its tiny area, is intrinsically intractable.

So, even for Bayesian neural network, a layer by layer abstraction along depth cannot be absent.

6 Transfer Learning

Transfer learning demands that the model can be separated so that, say, some lower level layers can be extracted out and directly transfered to another model as its lower level layers without any modification on these layers. To do so, we have to demand that the marginalization of the $q(\theta; a, \mu, \zeta)$ on some θ_is shall be easy to take. Indeed, the marginalization of our model is straight forward.

7 Why not MCMC?

Instead, the MCMC approximation to posterior cannot be marginalized easily, and even intractable. So, MCMC approximation cannot provide transfer learning as we eager. This is the most important reason that we do not prefer MCMC. Furthermore, MCMC is not greedy enough so that it converges quite slowly, especially in high-dimensional parameter-space.

8 Problems

8.1 The Curse of Dimensinality

8.1.1 Range of Sampling

Usually, the curse of dimensionality raises in the grid searching or numerial integral. And gradient based optimization and Monte Carlo integral deal the curse. However, the curse of dimensionality emerges from another aspect: the range of sampling of initial values of z in the iteration process of optimization increases as $\sqrt{N_d}$.

The large range of sampling then calls for more elapsed time of convergence.

Example 3. Consider two vector y_1 and y_2 in N_d-dimension Euclidean space $X(N_d)$. Let $Y_1(N_d) := \{x \in X(N_d) : \|x - y_1\| < \|x - y_2\|\}$. Let $S(N_d) \subset X(N_d)$ as the range of sampling. Consider the ratio

$$R_1(N_d) := \frac{\|S(N_d) \cap Y_1(N_d)\|}{\|S(N_d)\|}.$$

We find that, e.g. let $y_1 = (-1, -1, ..., -1)$, $y_2 = (3, 3, ..., 3)$, and $S(N_d) = (-r, r) \times (-r, r) \times \cdots \times (-r, r)$ wherein $r = 10$, R_1 becomes unit after $N_d \geqslant ?$, and that R_1 will be around 0.5 if let $r \sim \sqrt{N_d}$.[10]

8.1.2 Relative Error of Inference

Theorem 1 hints that the relative error of inference positively related with dimension, as the loss does so.

9 Drafts

$f(x; \theta)$, $D = \{(x_i, y_i, z_i) : i = 1, 2, ..., N_D\}$. $y_i = f(x_i; \theta) + \epsilon$, $\epsilon \sim P(0, z_i)$.

For instance, suppose $P = \mathcal{N}$, then we have

$$\ln p(y_{1:N_D} | x_{1:N_D}; \theta) = \sum_i^{N_D} \ln p(y_i | x_i; \theta).$$

From $y_i = f(x_i; \theta) + \epsilon$ gets $y_i - f(x_i; \theta) \sim P(0, z_i) = \mathcal{N}(0, z_i)$.

$$\sum_i^{N_D} \ln p(y_i | x_i; \theta) = -\sum_i^{N_D} \left\{ \ln (2\sqrt{z_i}) + \frac{(y_i - f(x_i; \theta))^2}{2 z_i^2} \right\}.$$

10. C.f. the code "`/docs/curse_of_dimensionality/curse_of_dimensionality.py`".