Supplementary Material: Learning Structured Weight Uncertainty in Bayesian Neural Networks

Shengyang Sun
Tsinghua University

Changyou Chen
Duke University

Lawrence Carin
Duke University

A When $\tilde{f}$ is from the Prior

If $\tilde{f}$ is a prior term, and $\mathcal{N}(\mathbf{w};\mathbf{m},\mathbf{V})$ is our current multivariate normal posterior approximation. Then according to Bayes Formula, we get the updated posterior $\tilde{q}(\mathbf{w})$:

$$s(\mathbf{w}) = Z^{-1}\tilde{f}(\mathbf{w})\mathcal{N}(\mathbf{w};\mathbf{m},\mathbf{V})$$

As $f(\mathbf{w})$ can be very complicated, getting the exact result of $s(\mathbf{w})$ can be hard or even impossible. So we choose to approximate it with another multivariate Gaussian Distribution. As the multivariate Gaussian distribution has two parameter $\mathbf{m}^{new}$ and $\mathbf{V}^{new}$, we only need to compute the mean and covariance matrix of the new posterior distribution $s(\mathbf{w})$.
According to (Minka, 2001), we get

$$\begin{aligned}\mathbf{m}^{new} &= \mathbf{m} + \mathbf{V}\nabla_{\mathbf{w}}\log Z \\ \mathbf{V}^{new} &= \mathbf{V} - \mathbf{V}\left(\nabla_{\mathbf{w}}\nabla_{\mathbf{w}}^T\log Z - 2\nabla_{\mathbf{V}}\log Z\right)\mathbf{V}\end{aligned} \tag{14}$$

Therefore, if we can compute the normalizer Z, we can update the multivariate Gaussian distribution analytically.
Take $\{\mathbf{m}_\ell^{(1)}, \mathbf{\Sigma}_\ell^{(1)}\}$ for example, the prior is $\mathcal{N}\left(\mathbf{v}_\ell^{(1)};\mathbf{0},\phi^{-1}\mathbf{I}_{V_\ell}\right)$

$$\begin{aligned}Z &= \int \mathcal{N}\left(\mathbf{v}_\ell^{(1)};\mathbf{0},\phi^{-1}\mathbf{I}_{V_\ell}\right) q(\mathbf{R})\,d\mathbf{R} \\ &= \int \mathcal{N}\left(\mathbf{v}_\ell^{(1)};\mathbf{0},\phi^{-1}\mathbf{I}_{V_\ell}\right) Gam\left(\phi;\alpha^\phi,\beta^\phi\right) \\ &\quad \mathcal{N}\left(\mathbf{m}_\ell^{(1)},\mathbf{\Sigma}_\ell^{(1)}\right) d\mathbf{v}_\ell^{(1)} d\phi \\ &= \int St\left(\mathbf{v}_\ell^{(1)};\mathbf{0},\frac{\beta^\phi}{\alpha^\phi}\mathbf{I}_{V_\ell},2\alpha^\phi\right)\mathcal{N}\left(\mathbf{v}_\ell^{(1)};\mathbf{m}_\ell^{(1)},\mathbf{\Sigma}_\ell^{(1)}\right)d\mathbf{v}_\ell^{(1)} \\ &\approx \int \mathcal{N}\left(\mathbf{v}_\ell^{(1)};\mathbf{0},\frac{\alpha^\phi}{\alpha^\phi-1}\frac{\beta^\phi}{\alpha^\phi}\mathbf{I}_{V_\ell}\right)\mathcal{N}\left(\mathbf{v}_\ell^{(1)};\mathbf{m}_\ell^{(1)},\mathbf{\Sigma}_\ell^{(1)}\right)d\mathbf{v}_\ell^{(1)} \\ &= \mathcal{N}\left(\mathbf{m}_\ell^{(1)};\mathbf{0},\frac{\beta^\phi}{\alpha^\phi-1}\mathbf{I}_{V_\ell}+\mathbf{\Sigma}_\ell^{(1)}\right)\end{aligned} \tag{15}$$

The distribution $St(\mathbf{x};\mu,\mathbf{\Sigma},v)$ is the multivariate Student Distribution. We approximate it using a multivariate Gaussian Distribution with the same mean and covariance in the fourth equation. The third equation takes advantage of the fact that

$$\int Gam(\lambda;\alpha,\beta)\,\mathcal{N}\left(\mathbf{x};\mu,\frac{\mathbf{\Sigma}}{\lambda}\right)d\lambda = St\left(\mathbf{x};\mu,\frac{\beta}{\alpha}\mathbf{\Sigma},2\alpha\right)$$

The last equation is the application of the formula that

$$\int \mathcal{N}(\mathbf{x};\mu_{\mathbf{f}},\mathbf{\Sigma}_{\mathbf{f}})\,\mathcal{N}(\mathbf{x};\mu_{\mathbf{g}},\mathbf{\Sigma})d\mathbf{x} = \mathcal{N}(\mu_{\mathbf{f}};\mu_{\mathbf{g}},\mathbf{\Sigma}_{\mathbf{f}}+\mathbf{\Sigma}_{\mathbf{g}})$$

The above multivariate Gaussian formula can also be extended to handle the 1-d Gaussian case. Inserting the above result into (14), we can update the mean and variance of any Gaussian distributed variables in the model.

B When $\tilde{f}$ is from the Likelihood

When $\tilde{f}$ is from the likelihood, for a Gaussian variational distribution, we need to compute the normalizer:

$$Z = \int \mathcal{N}\left(y_n; f(x_n,\mathbf{W},\mathbf{P},\mathbf{Q}),\gamma^{-1}\right) q(\Theta)\,d\Theta$$

Assume $\mathbf{z}_{nL} = f(x_n,\mathbf{W},\mathbf{P},\mathbf{Q})$ has an approximate Gaussian distribution with mean $\mathbf{m}_{\mathbf{z}_{nL}}$ and variance vector $\mathbf{v}_{\mathbf{z}_{nL}}$. Then the normalizer Z can be computed as

$$\begin{aligned}Z &= \int \mathcal{N}\left(\mathbf{y}_n; f\left(x_n,\mathbf{P}_L\,\mathbf{B}_L\,\mathbf{Q}_L^T,\right),\gamma^{-1}\,\mathbf{I}_{V_L}\right) q(\mathbf{R})\,d\mathbf{R} \\ &\approx \int \mathcal{N}\left(\mathbf{y}_n;\mathbf{z}_{nL},\gamma^{-1}\,\mathbf{I}_{V_L}\right)\mathcal{N}\left(\mathbf{z}_{nL};\mathbf{m}_{\mathbf{z}_{nL}},\operatorname{diag}\mathbf{v}_{\mathbf{z}_{nL}}\right) \\ &\quad Gam\left(\gamma;\alpha^\gamma,\beta^\gamma\right)d\,\mathbf{z}_{nL}\,d\gamma \\ &= \int St\left(\mathbf{y}_n;\mathbf{z}_{nL},\frac{\beta^\gamma}{\alpha^\gamma},2\alpha^\gamma\right)\mathcal{N}\left(\mathbf{z}_{nL};\mathbf{m}_{\mathbf{z}_{nL}},\mathbf{v}_{\mathbf{z}_{nL}}\right)d\,\mathbf{z}_{nL} \\ &= \mathcal{N}\left(\mathbf{y}_n;\mathbf{m}_{\mathbf{z}_{nL}},\frac{\beta^\gamma}{\alpha^\gamma-1}+\mathbf{v}_{\mathbf{z}_{nL}}\right)\end{aligned} \tag{16}$$

Here, we still do not know the value of mean $\mathbf{m}^{\mathbf{z}_{nL}}$ and variance vector $\mathbf{v}^{\mathbf{z}_{nL}}$. This can be resolved by using the feed forward network structure. We propagate the distributions forward through the neural network and if necessary we approximate the distribution with a Gaussian distribution.

We assume the output $\mathbf{z}_{n(\ell-1)}$ of layer $\ell-1$ is a diagonal Gaussian distribution with mean $\mathbf{m}_{\mathbf{z}_{n(l-1)}}$ and variance vector $\mathbf{v}_{\mathbf{z}_{n(l-1)}}$. What needs to be explained is that the independence of neurons does not interfere with the dependence of parameters. In each layer of our model, the information goes through three stages, multiplying $\mathbf{Q}_\ell^T$, multiplying $\mathbf{B}_\ell$ and then multiplying $\mathbf{P}_\ell$. After the first stage, $\mathbf{z}_{n\ell}^{(1)} = \mathbf{Q}_\ell^T \mathbf{z}_{n(l-1)}$. As $\mathbf{Q}_\ell = \mathbf{I} - 2\frac{\mathbf{v}_\ell^{(1)}\mathbf{v}_\ell^{(1)T}}{\mathbf{v}_\ell^{(1)T}\mathbf{v}_\ell^{(1)}}$ and $\mathbf{v}_\ell^{(1)} \sim \mathcal{N}\left(\mathbf{m}_\ell^{(1)}, \mathbf{\Sigma}_\ell^{(1)}\right)$, also $\mathbf{Q}_\ell$ is an orthogonal matrix, the covariance matrix of $\mathbf{z}_{n\ell}^{(1)}$ is the same as the covariance matrix of $\mathbf{z}_{n(l-1)}$. As we assume $\mathbf{z}_{n(l-1)}$ is a multivariate Gaussian random variable with diagonal covariance a matrix, $\mathbf{z}_{n\ell}^{(1)}$ is still diagonal multivariate Gaussian distributed with variance vector $\mathbf{v}_{\mathbf{z}_{n(l-1)}}$. Thus we can get the mean and variance vector of $\mathbf{z}_{n\ell}^{(1)}$ as

$$\mathbf{m}_{\mathbf{z}_{n\ell}}^{(1)} = \left(\mathbf{I} - 2\mathbf{E}\frac{\mathbf{v}_\ell^{(1)}\mathbf{v}_\ell^{(1)T}}{\mathbf{v}_\ell^{(1)T}\mathbf{v}_\ell^{(1)}}\right)\mathbf{m}_{\mathbf{z}_{n(l-1)}},$$
$$\mathbf{v}_{\mathbf{z}_{n\ell}}^{(1)} = \mathbf{v}_{\mathbf{z}_{n(l-1)}} ,$$

where for the expectation, we use Mento Carolo sampling to approximate it. As $\mathbf{\Sigma}_\ell^{(1)}$ is a positive definite matrix, we perform cholesky decomposition and get $\mathbf{\Sigma}_\ell^{(1)} = \mathbf{K}\mathbf{K}^T$. Therefore if we set $\mathbf{k} = \mathbf{K}\left(\mathbf{v}_\ell^{(1)} - \mathbf{m}_\ell^{(1)}\right)$, $\mathbf{k} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$, the expectation becomes

$$E\frac{\left(\mathbf{K}^{-1}\mathbf{k} + \mathbf{m}_{\mathbf{z}_{n\ell}}^{(1)}\right)\left(\mathbf{K}^{-1}\mathbf{k} + \mathbf{m}_{\mathbf{z}_{n\ell}}^{(1)}\right)^T}{\left(\mathbf{K}^{-1}\mathbf{k} + \mathbf{m}_{\mathbf{z}_{n\ell}}^{(1)}\right)^T\left(\mathbf{K}^{-1}\mathbf{k} + \mathbf{m}_{\mathbf{z}_{n\ell}}^{(1)}\right)} \approx \frac{1}{M}\sum_{i=1}^{M}\frac{\left(\mathbf{K}^{-1}\mathbf{k}_i + \mathbf{m}_{\mathbf{z}_{n\ell}}^{(1)}\right)\left(\mathbf{K}^{-1}\mathbf{k}_i + \mathbf{m}_{\mathbf{z}_{n\ell}}^{(1)}\right)^T}{\left(\mathbf{K}^{-1}\mathbf{k}_i + \mathbf{m}_{\mathbf{z}_{n\ell}}^{(1)}\right)^T\left(\mathbf{K}^{-1}\mathbf{k}_i + \mathbf{m}_{\mathbf{z}_{n\ell}}^{(1)}\right)} \tag{17}$$

Based on this, it is convenient to calculate the derivative with respect to $\mathbf{m}_{\mathbf{z}_{n\ell}}^{(1)}$ and $\mathbf{K}$. Note that we need to compute the derivative of Z with respect to $\mathbf{m}_\ell^{(1)}$ and $\mathbf{\Sigma}_\ell^{(1)}$ in order to get the updated mean and covariance matrix. The reparameterization trick provides a efficient way to do this. Specifically, because $\mathbf{\Sigma}_\ell^{(1)} = \mathbf{K}\mathbf{K}^T$, we get the derivative with respect to $\mathbf{\Sigma}_\ell^{(1)}$ as

$$\nabla_{\mathbf{\Sigma}_\ell^{(1)}} \log Z = 0.5 * \nabla_{\mathbf{K}} \log Z * \mathbf{K}^{-1} .$$

At the second stage, $\mathbf{z}_{n\ell}^{(2)} = \mathbf{W}_\ell \mathbf{z}_{n\ell}^{(1)}/\sqrt{V_{l-1}+1}$. The mean and variance vector of $\mathbf{z}_{n\ell}^{(2)}$ can be computed by

$$\mathbf{m}_{\mathbf{z}_{n\ell}}^{(2)} = \mathbf{M}_\ell \mathbf{m}_{\mathbf{z}_{n\ell}}^{(1)}/\sqrt{V_{l-1}+1}$$
$$\mathbf{v}_{\mathbf{z}_{n\ell}}^{(2)} = \Big[\left(\mathbf{M}_\ell \circ \mathbf{M}_\ell\right)\mathbf{v}_{\mathbf{z}_{n\ell}}^{(1)} + \mathbf{\Sigma}_\ell^{(1)}\left(\mathbf{m}_{\mathbf{z}_{n\ell}}^{(1)} \circ \mathbf{m}_{\mathbf{z}_{n\ell}}^{(1)}\right) + \mathbf{\Sigma}_\ell^{(1)}\mathbf{v}_{\mathbf{z}_{n\ell}}^{(1)}\Big]/\sqrt{V_{l-1}+1} ,$$

where $\mathbf{M}_\ell$ and $\mathbf{\Sigma}_\ell^{(1)}$ are $V_l \times (V_{l-1}+1)$ matrices whose entries are given by $m_{ij\ell}$ and $\lambda_{ij\ell}$, respectively. And $\circ$ denotes the Hadamard elementwise product.

The last state is similar to the first stage as $\mathbf{P}_\ell$ and $\mathbf{Q}_\ell$ have the same kind of distribution. Let $\mathbf{a}_{n\ell} = \mathbf{P}_\ell\, \mathbf{z}_{n\ell}^{(2)}$. As $\mathbf{P}_\ell = \mathbf{I} - 2\frac{\mathbf{v}_\ell^{(2)}\mathbf{v}_\ell^{(2)T}}{\mathbf{v}_\ell^{(2)T}\mathbf{v}_\ell^{(2)}}$ and $\mathbf{v}_\ell^{(2)} \sim \mathcal{N}\left(\mathbf{m}_\ell^{(2)}, \mathbf{\Sigma}_\ell^{(2)}\right)$, we can get the mean and variance vector of $\mathbf{a}_{n\ell}$ as

$$\mathbf{m}_{\mathbf{z}_{n\ell}}^{(2)} = \left(\mathbf{I} - 2\mathbf{E}\frac{\mathbf{v}_\ell^{(2)}\mathbf{v}_\ell^{(2)T}}{\mathbf{v}_\ell^{(2)T}\mathbf{v}_\ell^{(2)}}\right)\mathbf{m}_{\mathbf{z}_{nl}}^{(2)},$$
$$\mathbf{v}_{\mathbf{z}_{n\ell}}^{(2)} = \mathbf{v}_{\mathbf{z}_{nl}^{(2)}} .$$

After multiplying the parameter matrices, the information of $\mathbf{a}_{n\ell}$ go thorough the neurons in layer ℓ. Let $\mathbf{b}_{n\ell} = \text{ReLU}\left(\mathbf{a}_{n\ell}\right)$, we can compute the mean and variance of the i-th entry in $\mathbf{b}_{n\ell}$ as:

$$m_i^{\mathbf{b}_{n\ell}} = \Phi\left(\alpha_i\right) v_i^{'}$$
$$v_i^{\mathbf{b}_{n\ell}} = m_i^{\mathbf{b}_{n\ell}} v_i^{'} \Phi\left(-\alpha_i\right) + \Phi\left(\alpha_i\right) v_i^{\mathbf{a}_{n\ell}}\left(1 - \gamma_i\left(\gamma_i + \alpha_i\right)\right)$$

where

$$v_i^{'} = m_i^{\mathbf{a}_l} + \sqrt{v_i^{\mathbf{a}_l}}\gamma_i, \quad \alpha_i = \frac{m_i^{\mathbf{a}_l}}{\sqrt{v_i^{\mathbf{a}_l}}}, \quad \gamma_i = \frac{\phi\left(-\alpha_i\right)}{\Phi\left(\alpha_i\right)}$$

and Φ are ϕ the CDF and the density function of a standard Gaussian Distribution.When α_i is very large and negative the previous definition of γ_i is not numerically stable. Instead, when $\alpha_i \leq -30$, we use the approximation $\gamma_i = -\alpha - \alpha^{-1} + 2{\alpha_i}^{-3}$. The output of the l-th layer $\mathbf{z}_l$ is obtained by concatenating $\mathbf{b}_l$ with constance 1 for the bias. We can therefore approximate the distribution of $\mathbf{z}_l$ to be Gaussian with marginal mean and variance

$$\mathbf{m}_{\mathbf{z}_{n\ell}} = \left[\mathbf{m}^{\mathbf{b}_{n\ell}}; 1\right]$$

$$\mathbf{v}_{\mathbf{z}_{n\ell}} = \left[\mathbf{v}^{\mathbf{b}_{n\ell}}; 0\right]$$

Therefore we get the approximating distribution of $\mathbf{z}_{n\ell}$. Performing sequential update along the neural network we can get the final distribution parameter $\mathbf{m}_{z_{nL}}$ and $\mathbf{v}_{z_{nL}}$.

C Extra Experimental Results

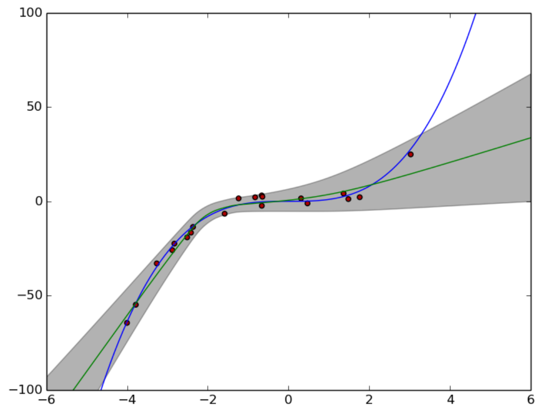
C.1 Extra results for nonlinear regression

We shown an extra toy regression experiment, which follows the setup in (Louizos and Welling, 2016) to randomly generate 20 data points x in the region $[-4, 4]$. The output y_n for each data point x_n is modeled as $y_n = x_n^3 + \epsilon_n$, where $\epsilon_n \sim \mathcal{N}(0, 9)$. We fit an FNN with one layer and 100 hidden units to this data. The number of epoch is set to 70. In our method, 100 samples is taken to approximate the expectation in (13). An extra toy regression experiment is shown in Figure 5.

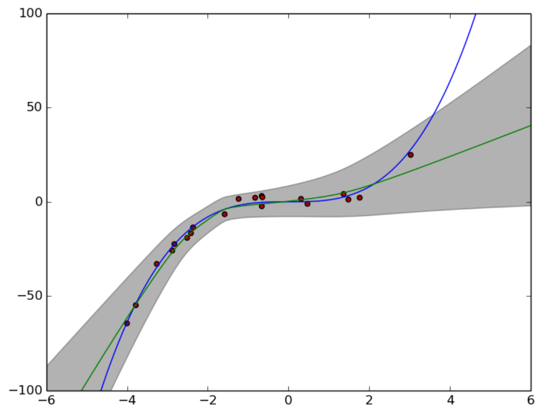
For real datasets, we plot all the learning curves for the nonlinear regression task on the 10 datasets in Figure 6 and 7, in terms of RMSE and log-likelihood, respectively.

C.2 Bayesian DNN for classification

Extending the original PBP framework (Hernández-Lobato and Adams, 2015), we provide some preliminary results to use PBP_MV for classification. The extension of PBP_MV for classification is described in Section 4.2. We train a three layer DNN with 150 neurons in each layer on the MNIST dataset. When training, we use the PBP to update the weights with MVG priors in the lower layers of the DNN, and use Adam to optimize the weight for the top layer (softmax layer). The algorithm is run for 10 epochs. We obtain a final error rate of 2.15%, which is better than purely training an DNN with stochastic optimization algorithms (Su et al., 2016), but worse than the state-of-the-art Bayesian DNN model (VMG) (Louizos and Welling, 2016). Note the comparison with VMG is not quite fair because VMG uses minnibatches to update parameters in each iteration, whereas in PBP_MV the minibatch size is actually equal to one. This brings the need for extending PBP_MV with minibatch-updating, an interesting future work to pursue.



(a) PBP



(b) PBP_MV

Figure 5: The toy regression experiment for PBP and our model. The observation are shown as black dots. The blue line represents the true data generating function and the mean predictions are shown as green line. The light gray shaded area is the ±3 standard deriva-tion confidence interval.

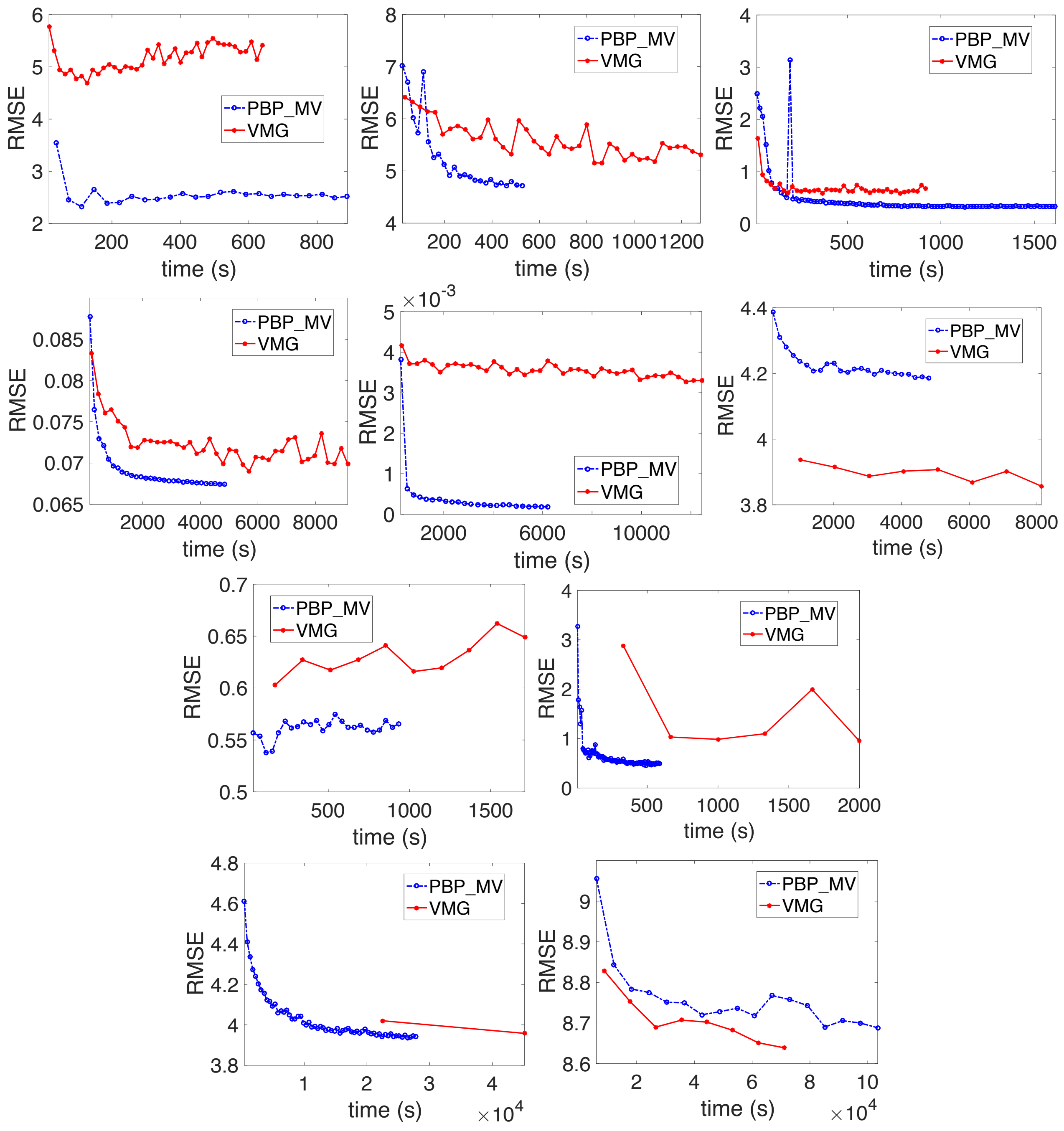


Figure 6: Learning curves in terms of *RMSE vs running time* on 10 datasets. From top-down and left to right, the datasets are: *boston_housing*, *concrete*, *energy*, *kin8nm*, *Naval*, *CCPP*, *winequality*, *yacht*, *protein* and *YearPrediction*.

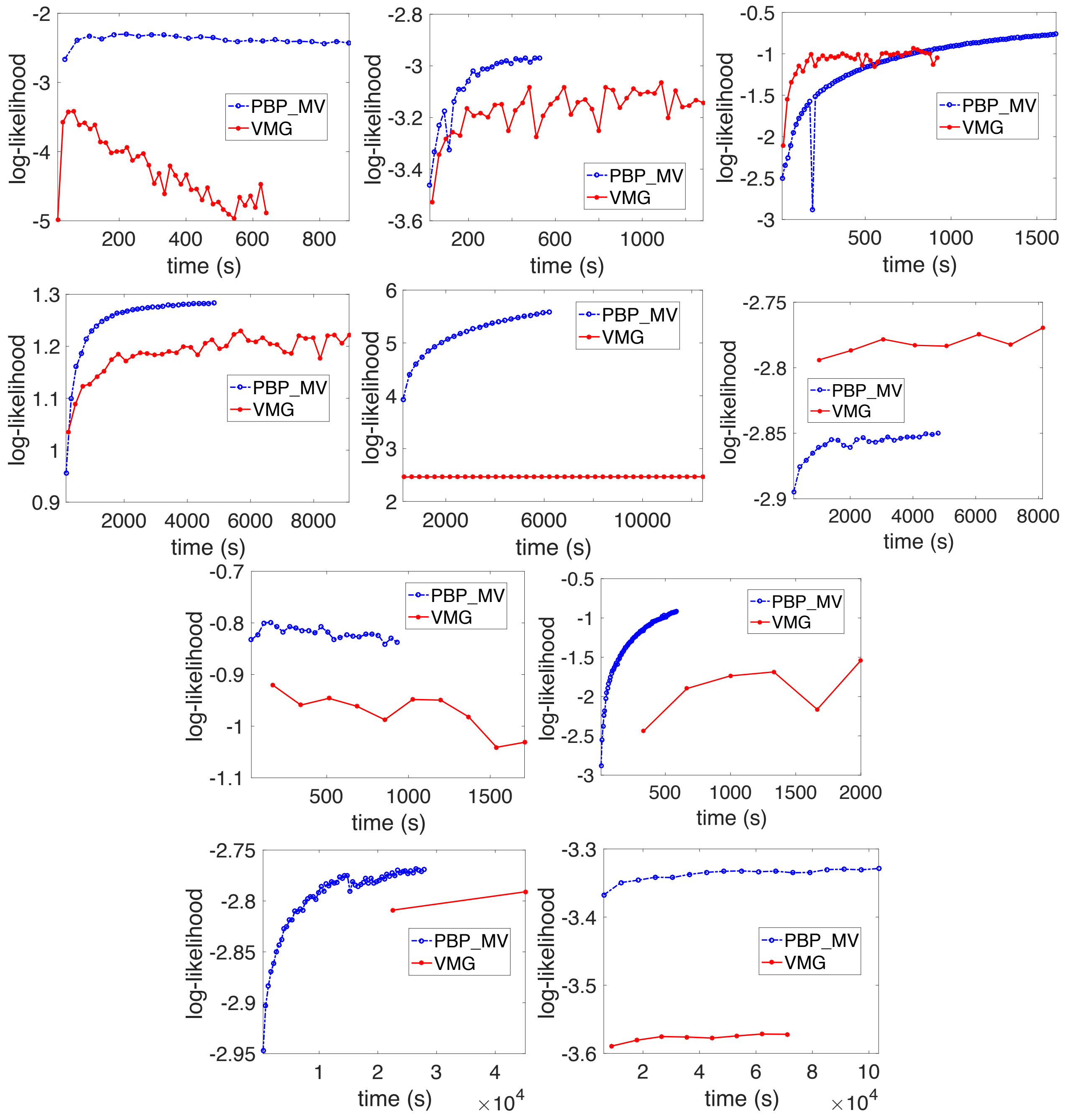


Figure 7: Learning curves in terms of *log-likelihood vs running time* on 10 datasets. From top-down and left to right, the datasets are: *boston_housing*, *concrete*, *energy*, *kin8nm*, *Naval*, *CCPP*, *winequality*, *yacht*, *protein* and *YearPrediction*.