Scalable Training of Inference Networks for Gaussian-Process Models Appendix

A. Experiment Details and Additional Results

A.1. Synthetic Data

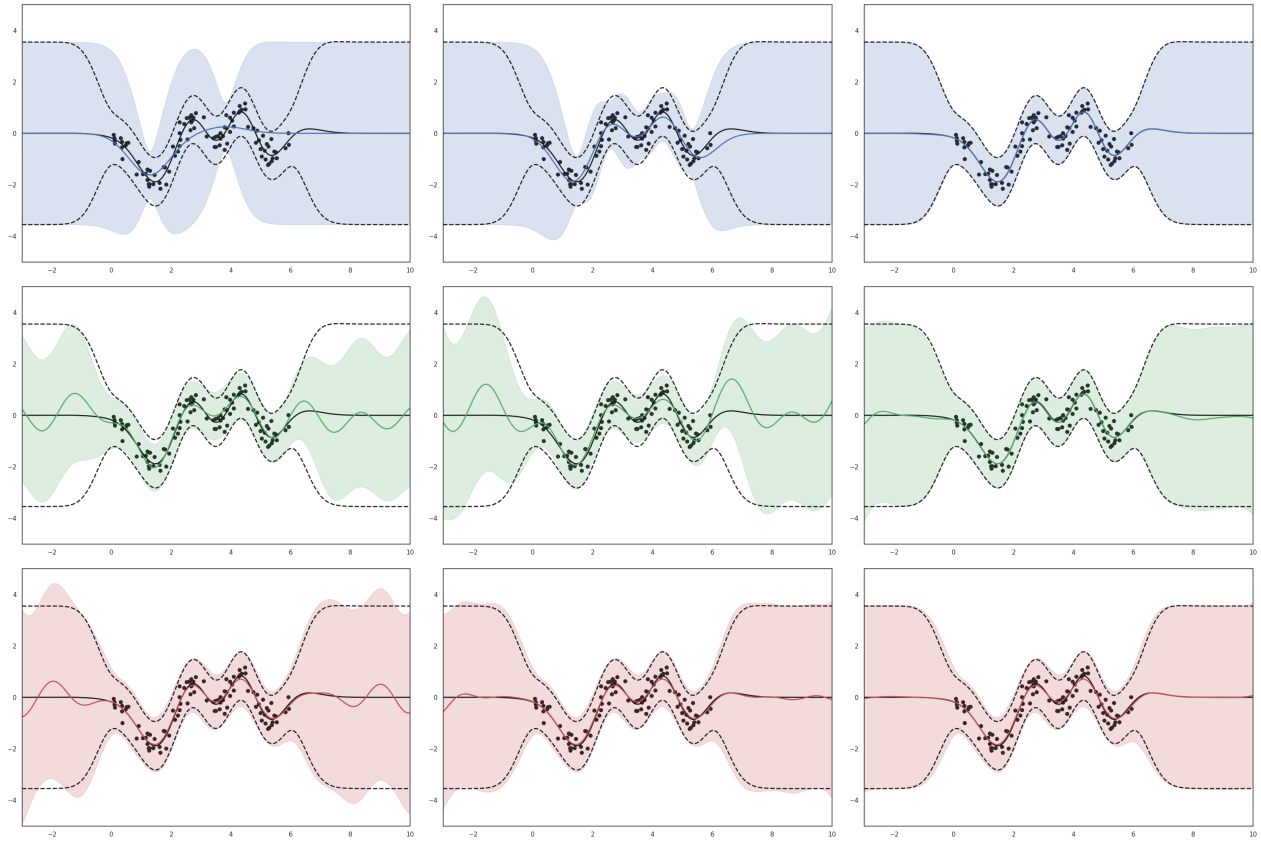

Figure 4. Posterior process on the Snelson dataset. (a) From top to bottom: SVGP with $M \in \{2, 5, 20\}$ (left to right) inducing points; GPNet with $M \in \{2, 5, 20\}$ measurement points; FBNN with $M \in \{2, 5, 20\}$ measurement points

The full figures including FBNN with $M = 2, 5, 20$ are shown in Fig. 4.

A.2. Regression

For all regression experiments, we set the measurement points to be sampled from the empirical distribution of training data convolved with the prior RBF kernel if the prior hyperparameters are initialized by optimizing GP marginal likelihood on a random subset, otherwise we set $c(\mathbf{x})$ simply to be the training data distribution. We fix $c(\mathbf{x})$ and do not adapt it together with the prior kernel parameters when the prior hyperparameters are updated during training.

Benchmarks The GP we use has a RBF kernel with dimension-wise lengthscales, also known as *Automatic Relevance Determination* (ARD) (MacKay, 1996). The RFE inference network we use has 1000 features (hidden units). We initialize the lengthscales in the network using the lengthscales of the prior kernel, and initialize the frequencies (first-layer weights)

Table 3. Regression: RMSE.

DATA SET	N	D	SVGP, 100	GPNET, 100	SVGP, 500	GPNET, 500	FBNN, 500
BOSTON	506	12	2.897±0.132	2.786±0.142	3.023±0.187	2.754±0.143	2.717±0.143
CONCRETE	1030	8	5.768±0.094	5.301±0.127	5.075±0.119	5.050±0.132	5.130±0.127
ENERGY	768	8	0.469±0.014	0.493±0.022	0.439±0.015	0.461±0.014	0.465±0.013
KIN8NM	8192	8	0.086±0.001	0.080±0.001	0.074±0.000	0.067±0.000	0.073±0.000
POWER	9568	4	3.941±0.033	3.942±0.032	3.791±0.034	3.898±0.032	4.140±0.028
PROTEIN	45730	9	4.536±0.010	4.540±0.014	4.154±0.010	4.329±0.013	4.082±0.052
WINE	1599	11	0.625±0.009	0.614±0.010	0.626±0.009	0.627±0.009	0.633±0.008

Table 4. Regression: Test log likelihood.

DATA SET	N	D	SVGP, 100	GPNET, 100	SVGP, 500	GPNET, 500	FBNN, 500
BOSTON	506	12	-2.465±0.054	-2.421±0.049	-2.458±0.072	-2.429±0.055	-2.389±0.030
CONCRETE	1030	8	-3.166±0.015	-3.115±0.024	-3.027±0.023	-3.066±0.022	-3.048±0.028
ENERGY	768	8	-0.675±0.024	-1.060±0.008	-0.600±0.033	-0.847±0.013	-0.808±0.016
KIN8NM	8192	8	1.006±0.004	1.095±0.011	1.183±0.004	1.283±0.005	1.186±0.005
POWER	9568	4	-2.793±0.008	-2.794±0.007	-2.755±0.008	-2.783±0.008	-2.848±0.006
PROTEIN	45730	9	-2.932±0.002	-3.057±0.032	-2.841±0.002	-2.986±0.029	-2.818±0.015
WINE	1599	11	-0.949±0.014	-0.917±0.014	-0.949±0.015	-0.948±0.014	-0.961±0.013

with random samples from a standard Gaussian. We then train these frequencies and lengthscales as inference network parameters. For FBNN, we keep all settings (including the inference network) the same as GPNet except the training objective used. We use 20 random splits for each dataset, where we keep 90% of the dataset as the training set and use the remaining 10% for test. The inputs and outputs of all data points are scaled to have nearly zero mean and unit variance using the mean and standard deviation calculated from training data. We use minibatch size 500, learning rate $\eta = 0.003$ for all the experiments. With each random seed we ran 10K iterations. For Boston, Concrete, Kin8nm and Protein we initialize the prior hyperparameters by maximizing GP marginal likelihood for 1K iterations on a randomly chosen subset of 1000 points. For Power and Wine we optimize the prior hyperparameters during training using the minibatch lower bound and the same learning rate as η, as described in section 3.3. We set $\beta_0 = 1$ and $\xi = 1$ except for Power and Protein we use $\beta_0 = 0.01, \xi = 0.1$ and $\beta_0 = 0.1, \xi = 0.1$, respectively. In Table 3 and Table 4 we list the mean and standard errors of all experiments.

Airline Delay We use the same type of GP prior and inference networks as in benchmark datasets above. We set $\beta_0 = 0.1, \xi = 0.1$. For all methods we train for 10K iterations with minibatch size 500 and learning rate $\eta = 0.003$. For GPNet we optimize the prior hyperparameters during training with the same learning rate as η, for which the objective is described in section 3.3. We also initialize the prior hyperparameters by maximizing GP marginal likelihood for 1K iterations on a randomly chosen subset of 1000 points. We do this for both GPNet and FBNN, though we found that for GPNet this does not improve the performance.

A.3. Classification

CNN-GP Prior The prior is defined as follows. Let $\mathbf{Z}^{(\ell)}(\mathbf{x})$ denote the pre-activation output of the ℓ-th layer of the ConvNet. The shape of $\mathbf{Z}^{(\ell)}(\mathbf{x})$ is $C^{(\ell)} \times (H^{(\ell)} D^{(\ell)})$. Each row of it represents the flattened feature map in a channel. A hidden layer in the network makes the transformation:

$$Z^{(\ell+1)}_{j,g}(\mathbf{x}) = b^{(\ell)}_j + \sum_{i=1}^{C^{(\ell)}} \sum_{h=1}^{H^{(\ell)} D^{(\ell)}} W^{(\ell)}_{j,i,g,h} a(Z^{(\ell)}_{i,h}(\mathbf{x})), \tag{15}$$

where $\mathbf{W}_{ji}$ is the pseudo weight matrix that corresponds to the convolutional filter $\mathbf{U}_{ji}$. The elements of each row in $\mathbf{W}_{ji}$ are zero except where $\mathbf{U}_{ji}$ applies. b_j denotes the bias in the j-th channel. a is the ReLU activation function. Let x, y denote

the positions within a filter, independent Gaussian priors are placed over $u^{(\ell)}_{j,i,x,y}$ and $b^{(\ell)}_j$ to form a Bayesian ConvNet:

$$u^{(\ell)}_{j,i,x,y} \sim \mathcal{N}(0, \sigma_w^2/C^{(\ell)}), \quad b_j \sim \mathcal{N}(0, \sigma_b^2).$$

By carefully taking the limit of hidden-layer widths, one can prove that each row in $\mathbf{Z}^{(\ell)}(\mathbf{x})$ form a multivariate Gaussian, and different rows (channels) are independent and identically distributed (i.i.d.), thus showing that the Bayesian ConvNet defines a GP (Garriga-Alonso et al., 2019). It is easy to show the prior mean function is zero: $\mathbb{E}[Z^{(\ell+1)}_{j,g}(\mathbf{x})] = 0$. To determine the prior covariance kernel of the output, we can follow a recursive procedure (For simplicity, we use $v^{(\ell)}_g(\mathbf{x}, \mathbf{x}')$ to denote the covariance between $Z^{(\ell)}_{j,g}(\mathbf{x})$ and $Z^{(\ell)}_{j,g}(\mathbf{x}')$):

$$v^{(\ell+1)}_g(\mathbf{x}, \mathbf{x}') = \sigma_b^2 + \sigma_w^2 \sum_{h \in g\text{-th patch}} s^{(\ell)}_h(\mathbf{x}, \mathbf{x}'),$$

$$s^{(\ell)}_h(\mathbf{x}, \mathbf{x}') = 1/2\pi \sqrt{v^{(\ell)}_g(\mathbf{x}, \mathbf{x}) v^{(\ell)}_g(\mathbf{x}', \mathbf{x}')} J_1(\theta^{(\ell)}_g),$$

where $J_1(\theta^{(\ell)}_g) = \sin\theta^{(\ell)}_g + (\pi - \theta^{(\ell)}_g)\cos\theta^{(\ell)}_g$ and $\theta^{(\ell)}_g = \arccos\left(v^{(\ell)}_g(\mathbf{x}, \mathbf{x}')/\sqrt{v^{(\ell)}_g(\mathbf{x}, \mathbf{x}) v^{(\ell)}_g(\mathbf{x}', \mathbf{x}')}\right)$. The prior convnet we used is a deep convolutional neural network with 6 residual blocks, each two of them operates on a different size of feature maps, with the first two on feature maps with the same size as the original image. There are strided convolution (stride=2) between the three groups.

Inference Networks The inference network we used has the same structure as the prior ConvNet, except the number of convolutional filters are [64, 64, 128, 128, 256, 256]. On top of it we have a fully-connected layer of size 512 and neural tangent kernels defined by a MLP with 100 hidden units for the output of each class.

We use batch size 64 and $M = 64$ measurement points in this experiment. We set $c(\mathbf{x})$ to be the empirical distribution of the training data. In implementation this simply means that we use two different shuffles of the training dataset, and pick a minibatch from each of them. Then we use one of the two minibatches as training points, and the other as measurement points. The learning rate is $\eta = 0.0003$. We set $\beta_0 = 0.01, \xi = 0.1$, and ran for 10K iterations. We did not update prior hyperparameters in this experiment.