A ADDITIONAL DETALS ON METHOD

Algorithm 1: Deep Probabilistic Kernel Learning

Input : Training points $\mathbf{X}$ and targets $\mathbf{y}$ along with a set of initial model parameters $\{\mathbf{w}_i^{(0)}\}_{i=1}^m$
Output : $\{\mathbf{w}_i\}_{i=1}^m \sim \hat{p}(\mathbf{w})$ where $\hat{p}(\mathbf{w})$ is obtained by functional gradient descent on (3)

Initialize model parameters $\{\mathbf{w}_i^{(0)}\}_{i=1}^m \sim p_0(\mathbf{w})$ for some known $p_0(\mathbf{w})$
for *iteration* t **do**
  $\mathbf{w}_i^{(t+1)} = \mathbf{w}_i^{(t)} - \epsilon_t \phi(\mathbf{w}_i^{(t)})$,
  where $\phi(\mathbf{w}) = \sum_{l=1}^m \kappa(\mathbf{w}, \mathbf{w}_l) \nabla_{\mathbf{w}_l} \hat{L}(\mathbf{w}_1, \ldots, \mathbf{w}_m)$
end

PROOF OF PROPOSITION 1

Recall that the entries of the kernel matrix K_{ij} under the transformation $\mathbf{u} = T(\mathbf{w}) = \mathbf{w} + s(\mathbf{w})$ are given by

$$K_{ij} = \mathbb{E}_{\mathbf{u},\mathbf{u}' \sim p_{[T]}(\mathbf{U})}[k(g_{\mathbf{u}}(\mathbf{x}_i), g_{\mathbf{u}'}(\mathbf{x}_j))] \tag{5}$$
$$= \mathbb{E}_{\mathbf{w},\mathbf{w}' \sim p(\mathbf{W})}[k(g_{T(\mathbf{w})}(\mathbf{x}_i), g_{T(\mathbf{w}')}(\mathbf{x}_j))] \tag{6}$$

Defining $k_{ij}(\mathbf{w}, \mathbf{w}') = k(g_{\mathbf{w}}(\mathbf{x}_i), g_{\mathbf{w}'}(\mathbf{x}_j))$ we have

$$K_{ij} = \mathbb{E}_{\mathbf{w},\mathbf{w}' \sim p(\mathbf{W})}[k_{ij}(\mathbf{w} + s(\mathbf{w}), \mathbf{w}' + s(\mathbf{w}'))] \tag{7}$$

Assuming that the distributions $p(\mathbf{W})$ and shifts $s(\mathbf{W})$ are functions in a RKHS $\mathcal{H}$ given by the kernel κ (κ, k, and K are all different), we have (from the definition of functional gradient $\nabla_s K_{ij}[s]$),

$$K_{ij}[s + \epsilon r] = K_{ij}[s] + \epsilon < \nabla_s K_{ij}[s], r >_{\mathcal{H}} + \mathcal{O}(\epsilon^2) \tag{8}$$

Thus we need to compute the difference $K_{ij}[s + \epsilon r] - K_{ij}[s]$ which, from Equation (7) is given by

$$\begin{aligned} K_{ij}[s + \epsilon r] - K_{ij}[s] = \mathbb{E}_p[k_{ij}(\mathbf{w} + s(\mathbf{w}) + \epsilon r(\mathbf{w}), \mathbf{w}' + s(\mathbf{w}') + \epsilon r(\mathbf{w}'))] \\ - \mathbb{E}_p[k_{ij}(\mathbf{w} + s(\mathbf{w}), \mathbf{w}' + s(\mathbf{w}'))] \end{aligned} \tag{9}$$

We use $\mathbb{E}_p$ to denote the expectation when $\mathbf{w}, \mathbf{w}' \sim p(\mathbf{W})$. The above equation can be rewritten as $K_{ij}[s + \epsilon r] - K_{ij}[s] = V_1 + V_2$ where

$$V_1 = \mathbb{E}_p[k_{ij}(\mathbf{w} + s(\mathbf{w}) + \epsilon r(\mathbf{w}), \mathbf{w}' + s(\mathbf{w}') + \epsilon r(\mathbf{w}'))] - \mathbb{E}_p[k_{ij}(\mathbf{w} + s(\mathbf{w}), \mathbf{w}' + s(\mathbf{w}') + \epsilon r(\mathbf{w}'))] \tag{10}$$

$$= \epsilon \mathbb{E}_p[\nabla_{\mathbf{w}} k_{ij}(\mathbf{w} + s(\mathbf{w}), \mathbf{w}' + s(\mathbf{w}') + \epsilon r(\mathbf{w}'))r(\mathbf{w}))] + \mathcal{O}(\epsilon^2) \tag{11}$$

$$\begin{aligned} &= \epsilon \mathbb{E}_p[(\nabla_{\mathbf{w}} k_{ij}(\mathbf{w} + s(\mathbf{w}), \mathbf{w}' + s(\mathbf{w}') + \epsilon r(\mathbf{w}')) \\ &- \nabla_{\mathbf{w}} k_{ij}(\mathbf{w} + s(\mathbf{w}), \mathbf{w}' + s(\mathbf{w}')) + \nabla_{\mathbf{w}} k_{ij}(\mathbf{w} + s(\mathbf{w}), \mathbf{w}' + s(\mathbf{w}')))r(\mathbf{w}))] + \mathcal{O}(\epsilon^2) \end{aligned} \tag{12}$$

$$= \epsilon \mathbb{E}_p[\nabla_{\mathbf{w}} k_{ij}(\mathbf{w} + s(\mathbf{w}), \mathbf{w}' + s(\mathbf{w}')))r(\mathbf{w}))] + \mathcal{O}(\epsilon^2) \tag{13}$$

$$= \epsilon < \mathbb{E}_p[\nabla_{\mathbf{w}} k_{ij}(\mathbf{w} + s(\mathbf{w}), \mathbf{w}' + s(\mathbf{w}')))\kappa(\mathbf{w}, .))], r >_{\mathcal{H}} + \mathcal{O}(\epsilon^2) \tag{14}$$

where the last line follows from the RKHS property. Similarly,

$$V_2 = \mathbb{E}_p[k_{ij}(\mathbf{w} + s(\mathbf{w}), \mathbf{w}' + s(\mathbf{w}') + \epsilon r(\mathbf{w}'))] - \mathbb{E}_p[k_{ij}(\mathbf{w} + s(\mathbf{w}), \mathbf{w}' + s(\mathbf{w}'))] \tag{15}$$

$$= \epsilon < \mathbb{E}_p[\nabla_{\mathbf{w}'} k_{ij}(\mathbf{w} + s(\mathbf{w}), \mathbf{w}' + s(\mathbf{w}')))\kappa(\mathbf{w}', .))], r >_{\mathcal{H}} + \mathcal{O}(\epsilon^2) \tag{16}$$

Since we transform the weights $\mathbf{w}$ after every iteration, therefore we only ever need to compute the gradient at $s(\mathbf{w}) = 0$. Thus, finally, we have the expression

$$\nabla_s K_{ij}[s] \mid_{s=0} = \mathbb{E}_p[\nabla_{\mathbf{w}} k_{ij}(\mathbf{w}, \mathbf{w}')\kappa(\mathbf{w}, .) + \nabla_{\mathbf{w}'} k_{ij}(\mathbf{w}, \mathbf{w}')\kappa(\mathbf{w}', .)] \tag{17}$$

If we draw m samples of model parameters $\mathbf{w}_1, \ldots, \mathbf{w}_m \sim p(\mathbf{w})$, the empirical estimate of $\nabla_s K_{ij}[s] \mid_{s=0}$ given by replacing expectations with sample averages is given by

$$\nabla_s K_{ij}[s] \mid_{s=0} \simeq \frac{1}{m^2} \sum_{l,l'=1}^{m} [\nabla_{\mathbf{w}_l} k_{ij}(\mathbf{w}_l, \mathbf{w}_{l'}))\kappa(\mathbf{w}_l, .)) + \nabla_{\mathbf{w}_{l'}} k_{ij}(\mathbf{w}_l, \mathbf{w}_{l'}))\kappa(\mathbf{w}_{l'}, .))] \tag{18}$$

Without loss of generality consider all the terms in the above expression that contain the gradient with respect to $\mathbf{w}_1$ and let us call that part of the summation T_1. Therefore

$$T_1 = \frac{1}{m^2} \sum_{l'=1}^{m} \nabla_{\mathbf{w}_1} k_{ij}(\mathbf{w}_1, \mathbf{w}_{l'}))\kappa(\mathbf{w}_1, .)) + \frac{1}{m^2} \sum_{l=1}^{m} \nabla_{\mathbf{w}_1} k_{ij}(\mathbf{w}_l, \mathbf{w}_1))\kappa(\mathbf{w}_1, .)) \tag{19}$$

Recall the expression for the empirical estimate of the entries of the kernel matrix K_{ij}

$$\hat{K}_{ij} \simeq \frac{1}{m^2} \sum_{l,l'=1}^{m} k_{ij}(\mathbf{w}_l, \mathbf{w}_{l'}) \tag{20}$$

Differentiating both sides with respect to $\mathbf{w}_1$,

$$\nabla_{\mathbf{w}_1} \hat{K}_{ij} \simeq \frac{1}{m^2} \sum_{l'=1}^{m} \nabla_{\mathbf{w}_1} k_{ij}(\mathbf{w}_1, \mathbf{w}_{l'}) + \frac{1}{m^2} \sum_{l=1}^{m} \nabla_{\mathbf{w}_1} k_{ij}(\mathbf{w}_l, \mathbf{w}_1) \tag{21}$$

Note that the term $\nabla_{\mathbf{w}_1} k_{ij}(\mathbf{w}_1, \mathbf{w}_1)$ occurs in both summmations. This is because $\nabla_{\mathbf{w}_1} k_{ij}(\mathbf{u}, \mathbf{v}) = \nabla_{\mathbf{u}} k_{ij}(\mathbf{u}, \mathbf{v}) \nabla_{\mathbf{w}_1} \mathbf{u} + \nabla_{\mathbf{v}} k_{ij}(\mathbf{u}, \mathbf{v}) \nabla_{\mathbf{w}_1} \mathbf{v} = \nabla_{\mathbf{w}_1} k_{ij}(\mathbf{w}_1, \mathbf{w}_1) + \nabla_{\mathbf{w}_1} k_{ij}(\mathbf{w}_1, \mathbf{w}_1)$ when $\mathbf{u} = \mathbf{v} = \mathbf{w_1}$ ($\mathbf{u}, \mathbf{v}, \mathbf{w_1}$ are all variable) .

Substituting Equation (21) in Equation (19)

$$T_1 = \kappa(\mathbf{w}_1, .) \nabla_{\mathbf{w}_1} \hat{K}_{ij} \tag{22}$$

We can apply the same argument to simplify the terms in Equation (18) that contain gradients with respect to other weights $\mathbf{w}_2, \ldots, \mathbf{w}_m$ in the same fashion. Therefore,

$$\nabla_s K_{ij}[s] \mid_{s=0} \simeq \frac{1}{m^2} \sum_{l,l'=1}^{m} [\nabla_{\mathbf{w}_l} k_{ij}(\mathbf{w}_l, \mathbf{w}_{l'}))\kappa(\mathbf{w}_l, .)) + \nabla_{\mathbf{w}_{l'}} k_{ij}(\mathbf{w}_l, \mathbf{w}_{l'}))\kappa(\mathbf{w}_{l'}, .))] \tag{23}$$

$$= \sum_{l=1}^{m} \kappa(\mathbf{w}_l, .) \nabla_{\mathbf{w}_l} \hat{K}_{ij} \tag{24}$$

From the chain rule for functional gradient descent we have $\nabla_s L = \sum_{i,j} \frac{\partial L}{\partial K_{ij}} \nabla_s K_{ij}[s]$ and the corresponding empirical estimate $\nabla_s L \mid_{s=0} \simeq \sum_{i,j} \frac{\partial \hat{L}}{\partial \hat{K}_{ij}} \nabla_s (\sum_{l=1}^{m} \kappa(\mathbf{w}_l, .) \nabla_{\mathbf{w}_l} \hat{K}_{ij})$. Switching the order of the summations gives

$$\nabla_s L \mid_{s=0} \simeq \sum_{l=1}^{m} \kappa(\mathbf{w}_l, .) \nabla_{\mathbf{w}_l} \hat{L}(\mathbf{w}_1, \ldots, \mathbf{w}_m) \tag{25}$$

B Additional Details on Experiments

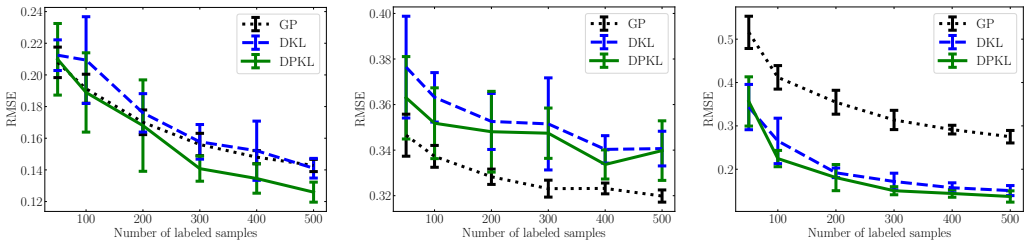
Infrastructure. All models were implemented in TensorFlow (Abadi et al., 2016). The experiments were run on machines with a 16-core Intel Xeon processor, 64 GiB RAM, TitanX GPU, 400GB NVMe SSD and a 2x4TB HDD. All experiments were run on CPU (GPU was not used).

Model Details: The GP model uses a Squared Exponential (SE) kernel directly on the data i.e. $k(\mathbf{x}_i, \mathbf{x}_j) = \exp(-\frac{1}{2} \sum_l \frac{1}{h_l^2} (\mathbf{x}_{il} - \mathbf{x}_{jl})^2)$. The DKL model embeds the data into a low-dimensional latent space before using an SE kernel i.e. $k(\mathbf{x}_i, \mathbf{x}_j) = \exp(-\frac{1}{2} \left\| g_{\mathbf{w}}(\mathbf{x}_i) - g_{\mathbf{w}}(\mathbf{x}_j) \right\|^2)$. The DPKL model embeds the data into a *probability distribution* in the latent space (as described in Section 3) and then use the SE kernel between distributions as

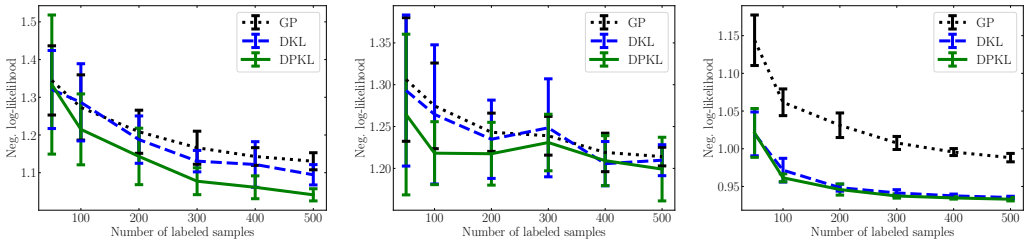
$$k(\mathbf{x}_i, \mathbf{x}_j) = \mathbb{E}_{\mathbf{z} \sim \Pr(\mathbf{Z}|\mathbf{x}_i),\, \mathbf{z}' \sim \Pr(\mathbf{Z}|\mathbf{x}_j)} [\exp(-\frac{1}{2} ||\mathbf{z} - \mathbf{z}'||^2)] \tag{26}$$

Dataset	Number of samples	Dimensionality
Buzz	$583,250$	77
CTSlice	$53,500$	384
Electric	$2,049,280$	6
Elevators	$16,599$	18
Parkinsons	$5,875$	20
Skillcraft	$33,285$	18

Table 1: Details (total number of samples and dimensionality) of each dataset from the UCI repository, used for regression



(a) RMSE for Elevators ($D = 18$) (b) RMSE for Skillcraft ($D = 18$) (c) RMSE for Electric ($D = 6$)



(d) Negll for Elevators ($D = 18$) (e) Negll for Skillcraft ($D = 18$) (f) Negll for Electric ($D = 6$)

Figure 3: Results for regression on 3 additional UCI Datasets with $n = \{50, 100, 200, 300, 400, 500\}$ labeled samples. Plots (a) - (c) show that DPKL has lower RMSE than DKL(Wilson et al., 2016) on all datasets and GP in 5 (3+2) out of 6 (3+3) datasets. Plots (d) - (f) show that DPKL quantifies uncertainty better (lower average negative log-likelihood (Negll) of test data)

We used the same neural network architecture as (Jean et al., 2018) ($D - 100 - 50 - 50 - d$) for mapping data points to the latent space in the DKL and DPKL models. Here D is the dimensionality of the datasets and $d = 2$ is the dimensionality of the latent embedding $\mathbf{z} \in \mathcal{Z}$. For DPKL we used $m = 10$ samples from this architecture to represent the distribution over model parameters. Following (Liu & Wang, 2016) we use the RBF kernel as the kernel κ between model parameters $\mathbf{w}$ (for functional gradient descent) with bandwidth chosen according to the median heuristic described in their work since it causes $\sum_j \kappa(\mathbf{w}, \mathbf{w}_j) \simeq 1$ for all $\mathbf{w}$, leading κ to behave like a probability distribution. The lengthscales for the GP SE kernel, and the neural network weights in DKL are optimized using gradient descent on the negative log-likelihood while the *distribution* over neural network weights in DPKL is optimized using functional gradient descent (Algorithm 1). We used the Random Fourier Features Approximation (Rahimi & Recht, 2008) to speed up kernel computation in DPKL with $R = 100$ samples from the Fourier Transform of k. The hyperparameter GP noise variance is set to $\sigma_\eta^2 = 1$.

Optimizer: We used the Adam Optimizer (Kingma & Ba, 2014), with a learning rate of 10^{-3} and Nesterov Momentum as implemented in Tensorflow (Abadi et al., 2016) under the name Nadam, to estimate model parameters in DPKL, DKL, as well as the GP kernel lengthscales, by maximum likelihood.

Data Processing: Following (Jean et al., 2018), we evaluate our models on 6 datasets of the UCI repository. The details of the datasets are given in Table 1. For each dataset and each model, we set aside 1000 examples as the test set. From the remaining examples we vary the number of labeled examples as $n = \{50, 100, 200, 300, 400, 500\}$. Labels (in $\mathbf{y}$) are normalized to have zero mean and unit variance. All models are trained for 50 epochs. For models, we create a validation set of 10% of the n training examples for early stopping (checked after every 10 epochs). For every value of n, models are trained on a different set of n examples in each trial (keeping the test set fixed). The process is repeated for 10 trials and models are evaluated in terms of prediction accuracy and uncertainty quantification.

Prediction Accuracy: We use Root Mean Squared Error (RMSE) as a measure of prediction accuracy. RMSE is computed between the labels of the test data set, and the predicted mean values of the test labels under the corresponding model. Low RMSE implies predicted labels are close to true labels. We plot average test RMSE v/s number of labeled examples for all datasets and all models. We also include error bars corresponding to one standard deviation. The results are distributed between Figure 2 in the main paper, and Figure 3 here. From the results it is clear that DPKL improves over DKL across *all* datasets and over GP in 5 out of the 6 datasets. While GP does a bit better than DPKL on one of the lower dimensional datasets (Skillcraft), we note that DPKL is significantly better than GP on all other datasets, including the high dimensional datasets CTSlice ($D = 384$) and Buzz ($D = 77$), thus clearly illustrating the advantages of our approach in high dimensional settings.

Uncertainty quantification: Following (Lakshminarayanan et al., 2017), we use test data negative log-likelihood as a measure of uncertainty quantification. For a test data point $\mathbf{x}_*$ with normalized target value y_*, if the Gaussian Process predicted mean is $\mu(\mathbf{x}_*)$ and predicted variance is $\sigma^2(\mathbf{x}_*)$, then the negative log-likelihood is given by

$$\text{Negll} = \frac{(y_* - \mu(\mathbf{x}_*)^2}{2\sigma^2(\mathbf{x}_*)} + \frac{1}{2}\log\sigma^2(\mathbf{x}_*) + \frac{1}{2}\log(2\pi)$$

Low negative log-likelihood implies that the model explains the data distribution (i.e quantifies uncertainty) well. The results are distributed between Figure 2 in the main paper, and Figure 3 here. We plot average test negative log-likelihood for DPKL, DKL, and GP with error bars corresponding to one standard deviation. The plots show that by projecting high dimensional data to low dimensional probability distributions before making predictions, DPKL consistently outperforms DKL and GP in this metric across all datasets.