Kernel-Based Just-In-Time Learning for Passing Expectation Propagation Messages

Abstract

We propose an efficient nonparametric strategy for learning a message operator in expectation propagation (EP), which takes as input the set of incoming messages to a factor node, and produces an outgoing message as output. We use kernel-based regression, which is trained on a set of probability distributions representing the incoming messages, and the associated outgoing messages. The kernel approach has two main advantages: first, it is fast, as it is implemented using a novel two-layer random feature representation of the input message distributions; second, it has principled uncertainty estimates, and can be cheaply updated online, meaning it can request and incorporate new training data when it encounters inputs on which it is uncertain. In experiments, our approach is able to solve learning problems where a single message operator is required for multiple, substantially different data sets (logistic regression for a variety of classification problems), where the ability to accurately assess uncertainty and to efficiently and robustly update the message operator are essential.

1 Introduction

An increasing priority in Bayesian modelling is to make inference accessible and implementable for practitioners, without their requiring specialist expertise in Bayesian inference. This is a goal sought in probabistic programming languages [Wingate et al., 2011, Goodman et al., 2008], as well as in more granular, component-based procedures [Duvenaud et al., 2013, Grosse et al., 2012] and systems [Stan Development Team, 2014, Minka et al., 2014]. In all cases, the user should be able to freely specify what they wish their model to express, without having to deal with the complexities of sampling, variational approximation, or distribution conjugacy. In reality, however, model convenience and simplicity can limit or undermine intended models, sometimes in ways the users might not expect. To take one example, the inverted Gamma prior, which is a convenient conjugate prior for standard deviation, has quite pathological behaviour [Gelman, 2006]. In general, more expressive, freely chosen models are more likely to require expensive sampling or quadrature approaches in their implementation, which can make them challenging to implement or impractical to run.

We address the particular setting of expectation propagation [Minka, 2001], a message passing algorithm wherein messages are confined to being members of a particular parametric family. The process of integrating incoming messages over a factor potential, and projecting the result onto the required output family, can be difficult, and in some cases not achieveable in closed form. Thus, a number of approaches have been proposed to implement EP updates numerically, independent of the details of the factor potential being used. One approach, due to Barthelmé and Chopin [2011], is to compute the message update via importance sampling. While this is exact for sufficient samples, this sampling procedure must be run at every iteration during inference, hence — while flexible — it is not viable for large-scale problems.

An improvement on this approach is to use importance sampled instances of input/output message pairs to train a regression algorithm, which can then be used in place of the sampler. The approach of Heess et al. [2013] uses neural networks to learn the mapping from incoming to outgoing messages. The mappings learned performed well on a variety of practical problems, however this approach comes with a disadvantage: it requires training data that cover the entire set of possible input messages for a given type of problem, and it has no way of assessing the uncertainty of its predictions (or of updating the model online in the event a prediction is uncertain). This can be a problem if it is desired to learn a message operator for a broad class of problems: to illustrate, if the goal is to do logistic regression, we would need to see during training a set of representative messages for all classification problems the user proposes to solve.

The disadvantages of the neural network approach were the basis for work by Eslami et al. [2014], who replaced the neural networks with random forests. The random forests were updated online when messages were observed on which the predictor was uncertain, where uncertainty was estimated via the heuristic of looking at the variability of the forest predictions for each point [Criminisi and Shotton, 2013]. The online updates were made by splitting the trees at their leaves. Both of these aspects are problematic, however. First, the heuristic used in computing uncertainty has no guarantees: indeed, uncertainty estimation for random forests remains a challenging topic of current research Hutter [2009]. This is not merely a theoretical consideration: in our experiments in Section 6, we demonstrate that such uncertainty estimates become unstable and inaccurate as we move away from the initial training data. Second, online updates of random forests may not work well when the newly observed data is from a very different distribution to the initial training sample [e.g. Lakshminarayanan et al., 2014, Fig. 3]. Indeed, for large amounts of training set drift, the leaf-splitting approach of Elsami et al. can result in a decision tree which is a long chain, giving a worst case cost of prediction of $O(n)$ for training data of size n, vs the ideal of $O(\log(n))$ for balanced trees.

We propose a novel, kernel-based approach to learning a message operator noparametrically for expectation propagation. The learning algorithm takes the form of a distribution regression problem, where the inputs are probabilty measures (represented as embeddings of the distributions to an RKHS), and the outputs are vectors of message parameters Szabo et al. [2014]. A first advantage of this approach is that one does not need to pre-specify customized features of the distributions, as in [Eslami et al., 2014, Heess et al., 2013]. Rather, we use a general characteristic kernel on input distributions [Christmann and Steinwart, 2010b, eq. 9], which in our experiments gives better performance than customized features. A potential downside of the kernel approach is that it can be computationally costly, with training time of $O(n^3)$ and a cost of $O(n^2)$ to make a prediction. One approach proposed to address this problem is a random feature approach [Rahimi and Recht, 2007, Le et al., 2013], where random features of the inputs are computed, however this approach does not apply when the inputs are distributions. We introduce a novel two-level random feature approach for kernels on distributions, which gives us both fast prediction (linear in the number of random features), and fast online updates (square in the number of random features).

A second advantage of our approach is that, being an instance of Gaussian process regression, there are well established estimates of predictive uncertainty [Rasmussen and Williams, 2006, Ch. 2]. We use these uncertainty estimates so as to determine when to query the importance sampler for additional input/output pairs. This is especially crucial if the message operator is being trained on multiple different datasets, where the typical input messages observed can be very different, and new training data are required. We demonstrate empirically that when the input messge distributions are significantly different from the initial training data, the uncertainty estimates returned for the kernel regression remain robust and informative, whereas those for random forests are highly unstable.

Our paper proceeds as follows. In Section 2, we introduce the notation for expectation propagation, and indicate how an importance sampling procedure can be used as an oracle to provide "gold standard" training data for the message operator (albeit at significant computational cost). We also give a brief overview of previous learning approaches to the problem, with a focus on that of Eslami et al. [2014], with brief details on the random forest regression, the uncertainty computation procedure, and the overall computational cost. Next, in Section **??**, we describe our kernel regression approach, and the form of the kernel message operator mapping the input messages (probability distributions in an RKHS) to output messages (sets of parameters of the outgoing messages), along with their uncertainty estimates. For the purposes of fast and efficient representation, we provide a random feature-based implementation of the regression, which may be of independent interest.Finally, in Section 6, we provide experiments, which cover three topics. First, we show that our uncertainty esimates are much more reliable than those obtained for random forests, hence more useful in determining when to request additional training message pairs from the oracle. Second, we demonstrate our method on artificial data with well-controlled statistical properties: we quickly learn the EP factor for logistic regression with high confidence, at a cost far lower than the importance sampling; and for the compound Gamma prior, again much faster than the quadrature implementation in Infer.net. Finally, we establish that our method is genuinely adaptive: we perform logistic regression on four different real-world datasets, updating our message operator as new datasets are made available, and the uncertainty over the resulting input messages increases.

2 Background

As is common in the message passing community we assume that distributions (or densities) over a set of variables $\mathbf{x} = (x_1, \dots x_D)$ of interest can be represented as factor graphs, i.e.

$$p(\mathbf{x}) = \frac{1}{Z} \prod_{j=1}^{J} \psi_j(\mathbf{x}_{\mathrm{ne}(\psi_j)}). \tag{1}$$

The factors ψ_j are non-negative functions which are defined over subsets $\mathbf{x}_{\mathrm{ne}(\psi_j)}$ of the full set of variables $\mathbf{x}$. These variables form the neighbors of the factor node ψ_j in the factor graph, and we use $\mathrm{ne}(\psi_j)$ to denote the corresponding set of indices. Z is the normalization constant.

In this work we are specifically interested in dealing with models in which some of the factors have a non-standard form, or may not have a known analytic expression be known (i.e. "black box" factors). Although our approach applies to any such factor in principle, in this paper we focus on *directed* factors $\psi(\mathbf{x}_{\mathrm{out}}|\mathbf{x}_{\mathrm{in}})$ which specify a conditional distribution over variables $\mathbf{x}_{\mathrm{out}}$ given $\mathbf{x}_{\mathrm{in}}$ (and thus $\mathbf{x}_{\mathrm{ne}(\psi)} = (\mathbf{x}_{\mathrm{out}}, \mathbf{x}_{\mathrm{in}})$). The only assumption we make is that we are provided with a forward sampling function $f : \mathbf{x}_{\mathrm{in}} \mapsto \mathbf{x}_{\mathrm{out}}$, i.e. a function that maps (stochastically or deterministically) a setting of the input variables $\mathbf{x}_{\mathrm{in}}$ to a sample from the conditional distribution over $\mathbf{x}_{\mathrm{out}} \sim \psi(\cdot|\mathbf{x}_{\mathrm{in}})$. One very natural way of specifying f is a short piece of code in a probabilistic program.

2.1 Message passing EP

The belief propagation - or sum-product algorithm - computes marginal distributions over subsets of variables by iteratively passing messages between variables and factors, ensuring consistency of the obtained marginals at convergence. Specifically, the messages sent from a factor ψ to variable x_i (where $i \in \mathrm{ne}(\psi)$) are computed as

$$m_{\psi \to i}(x_i) = \int \psi(\mathbf{x}_{\mathrm{ne}(\psi)}) \prod_{i' \in \mathrm{ne}(\psi)\backslash i} m_{i' \to \psi}(x_{i'}) \mathrm{d}\mathbf{x}_{\mathrm{ne}(\psi)\backslash i}, \tag{2}$$

where $m_{i' \to \psi}(\cdot)$ are the messages sent to factor ψ from its neighboring variables $x_{i'}$ other than x_i.

EP introduces an approximation in the case when the message $m_{\psi \to i}(\cdot)$ does not have a simple parametric form, by projecting the exact marginal $m_{i \to \psi}(x_i) m_{\psi \to i}(x_i)$ onto a member of some class of known parametric distributions. That is $m_{\psi \to i}(x_i)$ is given by

$$\frac{\mathrm{proj}\left[m_{i \to \psi}(x_i) \int \psi(\mathbf{x}_{\mathrm{ne}(\psi)}) \prod_{i' \in \mathrm{ne}(\psi)\backslash i} m_{i' \to \psi}(x_{i'}) \mathrm{d}\mathbf{x}_{\mathrm{ne}(\psi)\backslash i} \right]}{m_{i \to \psi}(x_i)} \tag{3}$$

where $\mathrm{proj}\left[p\right] = \mathrm{argmin}_{q \in \mathcal{Q}} \mathrm{KL}\left[p||q\right]$, and $\mathcal{Q}$ is typically in the exponential family, e.g. the set of Gaussian or Beta distributions.

2.2 Monte-Carlo message approximation

By projecting messages onto simple parametric forms, EP introduces an approximation that allows message passing to proceed when the true messages are not easily representable in closed form. Computing *the numerator* of (3) remains, however, a challenging problem in itself as it requires evaluating a high-dimensional integral as well as minimization of the Kullback-Leibler divergence to some non-standard distribution. For non-trivial factors with known analytic form this often requires hand-crafted approximations **?** or, for instance, the use of expensive numerical integration techniques; for "black-box" factors as described above (sec. 2) it is entirely unclear how to proceed.

This significantly limits the ease with which EP can be applied and its scope in practice. Motivated by these shortcomings Barthelmé and Chopin [2011], Heess et al. [2013], Eslami et al. [2014] exploit an alternative, stochastic approximation: In general, the projection $\mathrm{proj}\left[\cdot\right]$ of p onto some member of the exponential family corresponds to matching the relevant moments of p, e.g. its mean and variance, if q is required to be a Gaussian distribution. Thus, projecting p onto a general member of the exponential family, $q(x|\eta) = h(\theta) \exp\left(\eta^\top u(x) - A(\eta)\right)$ with a vector of sufficient statistics u and natural parameters η requires us to compute the expectation of $u(\cdot)$ under the numerator of 3).

A sample based approximation of this expectation can, in general, obtained via MCMC. Given a forward-sampling function f as described above, one especially simple approach is importance sampling:

$$\mathbb{E}_{\mathbf{x}_{\mathrm{ne}(\psi)} \sim b}\left[u(x)\right] \approx \frac{1}{N} \sum_{l=1}^{N} w(\mathbf{x}^l_{\mathrm{ne}(\psi)}) u(x^l_i), \qquad \mathbf{x}^s_{\mathrm{ne}(\psi)} \sim \tilde{b} \tag{4}$$

where, on the left hand side,

$$b(\mathbf{x}_{\mathrm{ne}(\psi)}) = \psi(\mathbf{x}_{\mathrm{ne}(\psi)}) \prod_{i \in \mathrm{ne}(\psi)} m_{i \to \psi}(x_i). \tag{5}$$

On the right hand side we draw samples $\mathbf{x}^l_{\mathrm{ne}(\psi)}$ from some proposal distribution $\tilde{b}$ which we choose to be

$$\tilde{b}(\mathbf{x}_{\mathrm{ne}(\psi)}) = r(\mathbf{x}_{\mathrm{in}}) \psi(\mathbf{x}_{\mathrm{out}}|\mathbf{x}_{\mathrm{in}}) \tag{6}$$

for some r with appropriate support, and compute importance weights

$$w(\mathbf{x}_{\mathrm{ne}(\psi)}) = \frac{\prod_{i\in\mathrm{ne}(\psi)} m_{i\to\psi}(x_i)}{r(\mathbf{x}_{\mathrm{in}})}. \tag{7}$$

The thus estimated expected sufficient statistics provide us with an estimate of the parameters η of the result q of the projection proj $[p]$, from which the message is readily computed.

2.3 Just-in-time learning of messages

Message approximations as in the previous section could be used directly when running the EP algorithm as in Barthelmé and Chopin [2011], but this approach can suffer from the unreliability of the importance sampling estimates when the number of samples N is not very large. On the other hand, for large N the computational cost of running EP with approximate messages can be very high, as the message approximations have to be re-computed in each iteration of EP. To obtain low-variance message approximations at lower computational cost Heess et al. [2013], Eslami et al. [2014] both amortize previously computed approximate messages by training a function approximator to directly map a tuple of incoming variable-to-factor messages $(m_{i'\to\psi})_{i'\in\mathrm{ne}(\psi)}$ to an approximate factor to variable message $m_{\psi\to i}$, i.e. they learn a mapping

$$\hat{m}^{\theta}_{\psi\to i} : (m_{i'\to\psi})_{i'\in\mathrm{ne}(\psi)} \mapsto m_{\psi\to i}, \tag{8}$$

where θ are the parameters of the function approximator. Note that for exponential family distribution this effectively reduces to a multi-variate regression problem as each message can be represented by a finite-dimensional parameter vector, and the message update (3) can thus be understood as a vector valued function.

While Heess et al. [2013] use neural networks and a very large, fixed training set to learn their approximate message operator prior to running EP with the intractable factor, Eslami et al. [2014] investigate an online learning approach in which the approximate message operator is updated on the fly, during inference, depending on whether the function approximator can be expected to produce a reliable message approximation. To this end, they endow their function approximator with an uncertainty estimate which

$$\hat{u}^{\theta}_{\psi\to i} : (m_{i'\to\psi})_{i'\in\mathrm{ne}(\psi)} \mapsto u, \tag{9}$$

where u indicates the expected unreliability of the predicted, approximate message $m_{\psi\to i}$ returned by $\hat{m}^{\theta}_{\psi\to i}$. If $u = \hat{u}^{\theta}_{\psi\to i}\left((m_{i'\to\psi})_{i'\in\mathrm{ne}(\psi)}\right)$ exceeds a predefined threshold, the required message is approximated via importance sampling (cf. eq. 4) and $\hat{m}^{\theta}_{\psi\to i}$ is re-trained on this new datapoint (leading to a new set of parameters θ' with $\hat{u}^{\theta'}_{\psi\to i}\left((m_{i'\to\psi})_{i'\in\mathrm{ne}(\psi)}\right) < \hat{u}^{\theta}_{\psi\to i}\left((m_{i'\to\psi})_{i'\in\mathrm{ne}(\psi)}\right)$.

The just-in-time approach of Eslami et al. [2014] critically relies on two properties of the class of function approximators used to represent $\hat{m}^{\theta}_{\psi\to i}$: they need to provide (un)reliability estimates $\hat{u}^{\theta}_{\psi\to i}$ over their domain, and they need to allow for online learning. Eslami et al. [2014] use decision trees. [These do fulfill both criteria, however, only when resorting to unprincipled heuristics. In the next section we present a framework based on kernel mean embeddings which possesses the desired properties and achieves this in a principled manner.] **Ali: if you could fill in some sentences describing, at a high level, online learning and uncertainty estimates in trees to justify the claim that these are heuristics/can be improved upon, that would be great. Also, we may want to move the computational complexity of tree operations here.**

3 Proposed Method

In principle, any distribution-to-vector regression function can be applied to learn a message operator $C_{f\to V'}$, once the training set $S_{V'}$ is obtained. Given incoming messages, the operator outputs $q_{f\to V'}$ from which the outgoing EP message is given by $m_{f\to V'} = q_{f\to V'}/m_{V'\to f}$ which can be computed analytically. We opt for kernel ridge regression [Schölkopf and Smola, 2002] as our message operator for its simplicity, its ease of use in an online setting during inference (see Sec. 3.3), and rich supporting theory.

3.1 Kernel Ridge Regression

WJ: We need to unify Kernel ridge regression and Bayesian linear regression sections. Posterior mean estimate of the Bayesian regression is the same as the solution to primal kernel ridge regression. Bayesian regression assumes a Gaussian noise output allowing us to compute predictive variance.

We consider here the problem of regressing smoothly from distribution-valued inputs to feature-valued outputs. We follow the regression framework of Micchelli and Pontil [2005], with convergence guarantees provided by Caponnetto and De Vito [2007]. Under smoothness constraints, this regression can be interpreted as computing the conditional expectation of the output features given the inputs [Grünewälder et al., 2012].

Let $\mathsf{X} = (\mathsf{x}_1|\cdots|\mathsf{x}_N)$ be the training regression inputs and $\mathsf{Y} = \left(\mathbb{E}_{q^1_{f\to V'}} u(v') | \cdots | \mathbb{E}_{q^N_{f\to V'}} u(v')\right) \in \mathbb{R}^{D_y \times N}$ be the regression outputs. The ridge regression in the primal form seeks $\mathsf{W} \in \mathbb{R}^{D_y \times D}$ for the regression function $g(\mathsf{x}) = \mathsf{W}\mathsf{x}$ which minimizes the squared-loss function $J(\mathsf{W}) = \sum_{i=1}^N \|\mathsf{y}_i - \mathsf{W}\mathsf{x}_i\|_2^2 + \lambda \operatorname{tr}\left(\mathsf{W}\mathsf{W}^\top\right)$ where λ is a regularization parameter and tr denotes a matrix trace.

It is well known that the solution is given by $\mathsf{W} = \mathsf{Y}\mathsf{X}^\top \left(\mathsf{X}\mathsf{X}^\top + \lambda I\right)^{-1}$ which has an equivalent dual solution $\mathsf{W} = \mathsf{Y}\left(K + \lambda I\right)^{-1} \mathsf{X}^\top = A\mathsf{X}^\top$. The dual formulation allows one to regress from any type of input objects if a kernel can be defined between them. All the inputs enter to the regression function through the gram matrix $K \in \mathbb{R}^{N\times N}$ where $(K)_{ij} = \kappa(\mathsf{x}_i, \mathsf{x}_j)$ yielding the regression function of the form $g(\mathsf{x}) = \sum_{i=1}^N a_i \kappa(\mathsf{x}_i, \mathsf{x})$ where $A := (a_1|\cdots|a_N)$. The dual formulation therefore allows one to straightforwardly regress from incoming messages to vectors of mean parameters. Although this property is appealing, the training size N in our setting can be chosen to be arbitrarily large, making computation of $g(\mathsf{x})$ expensive for a new unseen point x.

To eliminate the dependency on N, we propose to apply random Fourier features [Rahimi and Recht, 2007] $\hat{\phi}(\mathsf{x}) \in \mathbb{R}^D$ for $\mathsf{x} := [m_{V\to f}]_{V\in\mathcal{V}(f)}$ such that $\kappa(\mathsf{x}, \mathsf{x}') \approx \hat{\phi}(\mathsf{x})^\top \hat{\phi}(\mathsf{x}')$ where D is the number of random features. The use of the random features allows us to go back to the primal form of which the regression function $g(\hat{\phi}(\mathsf{x})) = \mathsf{W}\hat{\phi}(\mathsf{x})$ can be computed efficiently. In effect, computing an EP outgoing message requires nothing more than a multiplication of a matrix W ($D_y \times D$) with the D-dimensional feature vector generated from the incoming messages.

3.2 Bayesian Linear Regression

Denote $X = (x_1|\cdots|x_N) \in \mathbb{R}^{D\times N}$ as (finite-dimensional) input and $Y = (y_1|\cdots|y_N) \in \mathbb{R}^{1\times N}$ (one coordinate of all expected sufficient statistics) as regression output. In our application, x_n will denote a random feature vector for $[m^n_{V\to f}]_{V\in\mathcal{V}(f)}$. Assume

$$\begin{aligned} w &\sim \mathcal{N}\left(w; 0, I_D\sigma_0^2\right) \\ Y \mid X, w &\sim \mathcal{N}\left(Y; w^\top X, \sigma_y^2 I_N\right). \end{aligned}$$

The output noise variance σ_y^2 should capture the intrinsic stochasticity of the importance sampler used to generate Y. It follows that the posterior of w is given by [Bishop, 2006]

$$\begin{aligned} p(w \mid Y) &= \mathcal{N}(w; \mu_w, \Sigma_w) \\ \Sigma_w &= \left(XX^\top\sigma_y^{-2} + \sigma_0^{-2} I\right)^{-1} \\ \mu_w &= \Sigma_w XY^\top \sigma_y^{-2}. \end{aligned}$$

The noise variance σ_y^2 is proportional to the regularization parameter in linear regression. The predictive distribution on the output y^* given an observation x^* is

$$\begin{aligned} p(y^*|x^*, Y) &= \int dw\, p(y^*|w, x^*, Y) p(w|Y) \\ &= \mathcal{N}\left(y^*; x^{*\top}\mu_w, x^{*\top}\Sigma_w x^* + \sigma_y^2\right) \end{aligned}$$

For simplicity, we treat each output (expected sufficient statistic) as a separate regression problem. Treating all outputs jointly can be achieved with a multi-output kernel [Alvarez et al., 2011].

3.3 Online Update

In this section, we consider an online update for Σ_w and μ_w when observations X comes in stream. We will use $\cdot^{(n)}$ to denote a quantity constructed from n samples. Recall that $\Sigma_w^{-1(N)} = XX^\top\sigma_y^{-2} + \sigma_0^{-2}I$. The posterior covariance matrix at time $N+1$ can be written as

$$\begin{aligned} \Sigma_w^{(N+1)} &= \left(XX^\top\sigma_y^{-2} + \sigma_0^{-2}I + x_{N+1}x_{N+1}^\top\sigma_y^{-2}\right)^{-1} \\ &= \left(\Sigma_w^{-1(N)} + x_{N+1}x_{N+1}^\top\sigma_y^{-2}\right)^{-1} \\ &= \Sigma_w^{(N)} - \frac{\Sigma_w^{(N)} x_{N+1}x_{N+1}^\top \Sigma_w^{(N)} \sigma_y^{-2}}{1 + x_{N+1}^\top \Sigma_w^{(N)} x_{N+1}\sigma_y^{-2}} \end{aligned}$$

where we used the Sherman-Morrison formula. In this form, the posterior covariance at time $N+1$ can be expressed in terms of the covariance at time N. Updating Σ_w costs $O(D^2)$ per one new observation. For $\mu_w = \Sigma_w XY^\top\sigma_y^{-2}$, we maintain $XY^\top \in \mathbb{R}^D$ and update it with

$$\left(XY^\top\right)^{(N+1)} = \left(XY^\top + x_{N+1}y_{N+1}\right),$$

costing $O(D)$ per update.

WJ: Mention initial minibatch training here.

3.4 Online-Active Learning

Since we have D_y regression functions, for each tuple of incoming messages x^*, there are D_y predictive variances, $v_1^*, \ldots, v_{D_y}^*$, one for each output. Let $\{\tau_i\}_{i=1}^{D_y}$ be pre-specified predictive variance thresholds. **WJ: How should we choose these thresholds ?** Given a new input x^*, if $v_1^* > \tau_1$ or $\cdots$ or $v_{D_y}^* > \tau_{D_y}$ i.e., the operator is uncertain, a query is made to the oracle to obtain a ground truth y^*. The pair (x^*, y^*) is then used to update Σ_w and μ_w as described previously.

3.5 Computational Complexity

WJ: Compare ridge regression cost with Ali's forest here ?

Let K be the number of trees in the random forest, D be the dimensionality of message representation, and N be the number of oracle consultations so far. Let M be number of training points in each leaf. Assuming that the depth of trees is $O(\log(N))$, one prediction from a random forest costs $O(KD\log(N))$, and one update costs $O(KD^2\log(N))$. This is because for each tree in forest $O(K)$, tree traversal takes $O(\log(N))$, and one prediction costs $O(D)$. One full training of a regression function costs $O(D^2(D+M))$ which depends on M. However typically M is negligible at a lower depth, a full training costs $O(D^3)$. The total updating cost across all trees is $O(KD^3 log(N))$. **maybe some of the tree complexity could be discussed in section 2.3** .

4 Kernels on Distributions

A number of kernels on distributions have been studied in the literature [Jebara and Kondor, 2003, Jebara et al., 2004]. Relevant to us are kernels whose random features can be efficiently computed. Due to the space constraint, we only give a few examples here.

4.1 Expected Product Kernel

An expected product kernel is also known as a set kernel. Let $\mu_{r^{(l)}} := \mathbb{E}_{r^{(l)}(a)}k(\cdot, a)$ be the mean embedding [Smola et al., 2007] of the distribution $r^{(l)}$ into RKHS $\mathcal{H}^{(l)}$ induced by the kernel k. Assume $k = k_{\text{gauss}}$ (Gaussian kernel) and assume there are c incoming messages $\mathsf{x} := (r^{(i)}(a^{(i)}))_{i=1}^c$ and $\mathsf{y} := (s^{(i)}(b^{(i)}))_{i=1}^c$. The expected product kernel κ_{pro} is defined as

$$\kappa_{\text{pro}}(\mathsf{x}, \mathsf{y}) := \left\langle \bigotimes_{l=1}^{c} \mu_{r^{(l)}}, \bigotimes_{l=1}^{c} \mu_{s^{(l)}} \right\rangle_{\otimes_l \mathcal{H}^{(l)}}$$
$$= \prod_{l=1}^{c} \mathbb{E}_{r^{(l)}(a)}\mathbb{E}_{s^{(l)}(b)} k_{\text{gauss}}^{(l)}(a, b) \approx \hat{\phi}(\mathsf{x})^\top \hat{\phi}(\mathsf{y})$$

where $\hat{\phi}(\mathsf{x})^\top\hat{\phi}(\mathsf{y}) = \prod_{l=1}^c \hat{\phi}^{(l)}(r^{(l)})^\top \hat{\phi}^{(l)}(s^{(l)})$. The feature map $\hat{\phi}^{(l)}(r^{(l)})$ can be estimated by applying the random Fourier features to $k_{\text{gauss}}^{(l)}$ and taking the expectations $\mathbb{E}_{r^{(l)}(a)}\mathbb{E}_{s^{(l)}(b)}$. The final feature map is $\hat{\phi}(\mathsf{x}) = \hat{\phi}^{(1)}(r^{(1)}) \circledast \hat{\phi}^{(2)}(r^{(2)}) \circledast \cdots \circledast \hat{\phi}^{(c)}(r^{(c)}) \in \mathbb{R}^{d^c}$ where $\circledast$ denotes a Kronecker product and we assume that $\hat{\phi}^{(l)} \in \mathbb{R}^d$ for $l \in \{1, \ldots, c\}$.

4.2 Expected Product Kernel on Joint Embeddings

Another way to define a kernel on x, y is to consider joint distributions $\mathsf{r} = \prod_{i=1}^c r^{(i)}$ and $\mathsf{s} = \prod_{i=1}^c s^{(i)}$ given by the product of all incoming messages, and define a kernel on the joint distributions. One can consider the expected product kernel on the joint distributions as

$$\kappa_{\text{joint}}(\mathsf{x}, \mathsf{y}) := \langle \mu_{\mathsf{r}}, \mu_{\mathsf{s}} \rangle_{\mathcal{G}} = \mathbb{E}_{\mathsf{r}(x)}\mathbb{E}_{\mathsf{s}(y)} k(x, y)$$

where $\mathcal{G}$ is an RKHS consisting of functions $g : \mathcal{X}^{(1)} \times \cdots \times \mathcal{X}^{(c)} \to \mathbb{R}$ and $\mathcal{X}^{(l)}$ denotes the domain of $r^{(l)}$ and $s^{(l)}$.

4.3 Gaussian Kernel on Joint Embeddings

Given two distributions r and s, a Gaussian kernel on mean embeddings Christmann and Steinwart [2010a] is defined as

$$\kappa_{\text{gauss}}(\mathsf{r}, \mathsf{s}) = \exp\left(-\frac{\|\mu_{\mathsf{r}} - \mu_{\mathsf{s}}\|_{\mathcal{H}}^2}{2\gamma^2}\right).$$

This kernel has two parameters, one for k in $\mu_{\mathsf{r}} = \mathbb{E}_{\mathsf{r}} k(x, \cdot)$ and γ^2. Empirically, this kernel performs significantly better than other kernels on predicting outgoing messages from incoming messages.

WJ: Since the norm squared inside exp is nothing but MMD^2 in a unit-ball RKHS (Gaussian kernel), Arthur may have some good intuition why it works well in practice ? Would be nice to have a paragraph on theoretical explanation if any.

Random features for Gaussian kernel on mean embeddings One way to get random features for the kernel is given by a two-staged approximation procedure as follows. By expanding the square in κ_{gauss}, we have

WJ: Cha
p, q to r,

$$\exp\left(-\frac{1}{2\gamma^2}\langle\mu_p, \mu_p\rangle + \frac{1}{\gamma^2}\langle\mu_p, \mu_q\rangle - \frac{1}{2\gamma^2}\langle\mu_q, \mu_q\rangle\right).$$

Assume k is translation invariant. The inner product of the mean embeddings $\langle\mu_p, \mu_q\rangle = \mathbb{E}_{p(x)}\mathbb{E}_{q(y)}k(x - y) \approx \hat{\phi}(p)^\top\hat{\phi}(q)$ can be approximated as in Sec. B.3. Assume $\hat{\phi}(p), \hat{\phi}(q) \in \mathbb{R}^{D_{in}}$. With the approximation of $\langle\mu_p, \mu_q\rangle$, we have

$$\kappa_{\text{gauss}}(p, q) \approx \exp\left(-\frac{\|\hat{\phi}(p) - \hat{\phi}(q)\|_{D_{in}}^2}{2\gamma^2}\right)$$

which can be thought of as a standard Gaussian kernel on $\mathbb{R}^{D_{in}}$. We can thus further approximate the Gaussian kernel with $k_{\text{gauss}}\left(\hat{\phi}(p), \hat{\phi}(q); \gamma^2\right) \approx \hat{\psi}(p)^\top\hat{\psi}(q)$ by standard random Fourier features [Rahimi and Recht, 2007] where $\hat{\psi}(p) \in \mathbb{R}^{D_{out}}$. A pseudocode for generating the random features is summarized in Algorithm 1. We need to pre-compute $\{\omega_i\}_{i=1}^{D_{in}}, \{b_i\}_{i=1}^{D_{in}}, \{\nu_i\}_{i=1}^{D_{out}}$ and $\{c_i\}_{i=1}^{D_{out}}$ where D_{in} and D_{out} are the number of random features to be

Algorithm 1 Construction of two-staged random features for a Gaussian kernel on mean embeddings

```
Input: Input distribution r, Fourier transform $\hat{k}$ of the embedding kernel $k$, number of inner features $D_{in}$, number of outer features $D_{out}$, outer Gaussian width $\gamma^2$.
Output: Random features $\hat{\psi}(\mathsf{r}) \in \mathbb{R}^{D_{out}}$.
1: Sample $\{\omega_i\}_{i=1}^{D_{in}} \overset{i.i.d}{\sim} \hat{k}$.
2: Sample $\{b_i\}_{i=1}^{D_{in}} \overset{i.i.d}{\sim}$ Uniform$[0, 2\pi]$.
3: $\hat{\phi}(\mathsf{r}) = \sqrt{\frac{2}{D_{in}}} \left(\mathbb{E}_{\mathsf{r}(x)} \cos(\omega_i^\top x + b_i)\right)_{i=1}^{D_{in}} \in \mathbb{R}^{D_{in}}$
   If $\mathsf{r}(x) = \mathcal{N}(x; m, V)$,
   $\hat{\phi}(\mathsf{r}) = \sqrt{\frac{2}{D_{in}}} \left(\cos(w_i^\top m + b_i) \exp\left(-\frac{1}{2} w_i^\top V w_i\right)\right)_{i=1}^{D_{in}}$.
4: Sample $\{\nu_i\}_{i=1}^{D_{out}} \overset{i.i.d}{\sim} \hat{k}_{\text{gauss}}(\gamma^2)$. i.e., Fourier transform of a Gaussian kernel with width $\gamma^2$.
5: Sample $\{c_i\}_{i=1}^{D_{out}} \overset{i.i.d}{\sim}$ Uniform$[0, 2\pi]$.
6: $\hat{\psi}(\mathsf{r}) = \sqrt{\frac{2}{D_{out}}} \left(\cos(\nu_i^\top x + c_i)\right)_{i=1}^{D_{out}} \in \mathbb{R}^{D_{out}}$
```

specified. A more efficient way to support a large number of random features is to store only the random seed used to generate them. The coefficients are generated on the fly when needed [Dai et al., 2014].

5 Related Work

TODO/TO add:

- To connect to Ali's work, can mention that online learning of random forests is difficult. Online learning is straightforward and exact for Bayesian linear regression. May ask Balaji more on relevant citations.
- It was argued in Hutter [2009] (Section 11.2) that a random forest may have problems with exploring previously-unseen areas of the input domain.

6 Experiments

To ensure that the operator is capable of learning the relationship of incoming and outgoing messages, we first consider a batch learning setting.

6.1 Predictive Power

Logistic As in Heess et al. [2013], Eslami et al. [2014], we consider the logistic factor **WJ: This x is not x in the factor graph. Fix this.**

$$f(z|x) = \delta\left(z - \frac{1}{1 + \exp(-x)}\right)$$

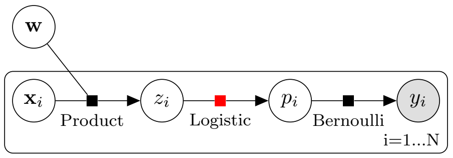


Figure 1: Factor graph for binary logistic regression.

where δ is the Dirac delta function, in the context of a binary logistic regression model as shown in Fig. 1. The two incoming messages are $m_{X\to f}(x) = \mathcal{N}(x; \mu, \sigma^2)$ and $m_{Z\to f}(z) = \text{Beta}(z; \alpha, \beta)$. Following Eslami et al. [2014], we draw $x, w \sim \mathcal{N}(0, I_{20})$, and set the number of observations to 400. We repeat the experiment 20 times where in each trial a new w and a new set of observations are generated. In each trial we collect messages in only the first five iterations of EP from Infer.NET's default implementation of the logistic factor. All collected messages are randomly partitioned into 5000 training messages and 2000 testing messages. We learn a message operator using a Gaussian kernel on joint embeddings as described in Eq. .... for sending $m_{f\to X}$. Regularization and kernel parameters are chosen by leave-one-out cross validation. The number of features are set to $D_{in} = 500$ and $D_{out} = 1000$. **AE: Why 1000?** . We report $\log KL[q\|\hat{q}]$ where $q = q_{f\to X}$ is the ground truth output message obtained from Infer.NET's and $\hat{q}$ is the message output from the operator. For better numerical scaling, regression outputs are set to $(\mathbb{E}_q[x], \log \mathbb{V}_q[x])$ instead of the expectations of the first two moments. **AE: No need to mention log-scaling?** The histogram of log KL errors is shown on the left of Fig. 2. The right figure shows sample output messages at different log KL errors. It can be seen that the operator is able to capture the relationship of incoming and outgoing messages. **WJ: some more explanation here.**

Compound Gamma **WJ: Hopefully I will have enough time to do this.** Andrew Gelman [Gelman, 2006]: "Gamma prior on precision is not good".

6.2 Uncertainty Estimate

In this experiment, we aim to verify that the predictive variance of the kernel-based message operator is a reliable quantity to be used as a criterion for deciding whether or not to query the oracle. We compute the predictive variance using the same operator as used in the logistic factor experiment on the training and test sets. A plots of KL-divergence errors versus predictive variances for predicting the mean of $m_{f\to X}$ is shown in Fig. 10

WJ: Feel free to fill in more explanation. The

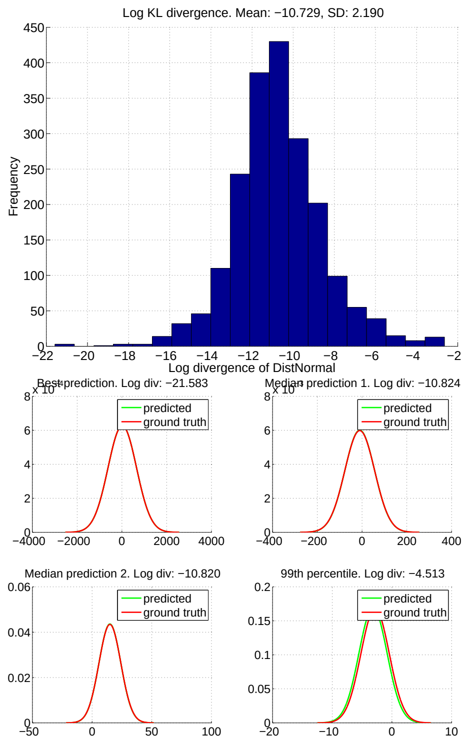


Figure 2: KL-divergence on a logistic factor test set using kernel on joint embeddings.

uncertainty estimates show a positive correlation with the KL-divergence errors. No points lie at the bottom right i.e., high confidence when it makes a big error. No points lie at the top left i.e., not confident when it should not be. The fact that the errors on the training set are roughly the same as the errors on the test set indicates that the operator does not overfit.

Unexplored Region One of the most important tasks in JIT learning is to correctly estimate the predictive uncertainty in unexplored regions in the space of incoming messages. In this section, we show that the operator reports a high uncertainty when encountered with incoming messages from a region far away from the messages in the training set. The first figure in Fig. 3 shows the training set as used in the logistic factor experiment. Each point represents one incoming message from X i.e., $m_{X\to f}$. The incoming messages from Z are not shown. We consider two test sets represented as two curves in the first figure. The second figure in Fig. 3 shows predict variance of each point in the two test sets, where we fix $m_{Z\to f} := \text{Beta}(z; 1, 2)$.

WJ: Finish this explanation. Two key observations. 1. As a test point lies further away from the training set, the predictive variance grows exponentially. 2. In both test sets, the operator is most certain around zeros mean. However, the first test set passes through a lower density region, the operator reports a higher uncertainty compared to the second test set.

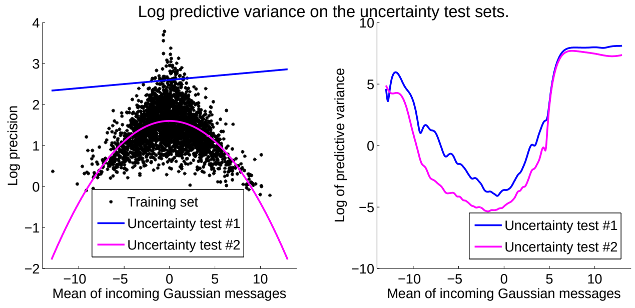
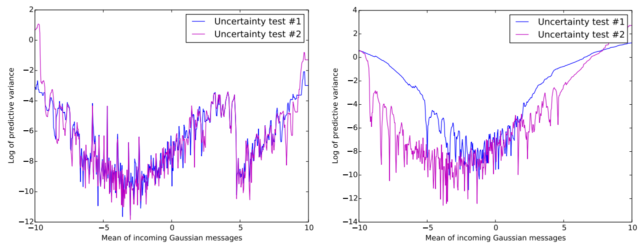


Figure 3: Collected training messages for the logistic factor and uncertainty estimates of the proposed method on the two uncertainty test sets.



(a) Breiman's random forests (b) Extremely randomized trees

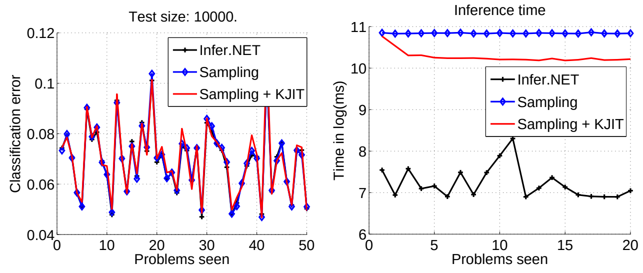
Figure 4: Uncertainty estimates of random forests on two uncertainty test sets shown in Fig. 3.

Uncertainty estimates from two commonly used random forests on the same problem are shown in Fig. 4 where Fig. 4a shows uncertainty estimates of Breiman's random forests [Breiman, 2001] and Fig. 4b shows uncertainty estimates of extremely randomized trees [Geurts et al., 2006]. We set the number of trees to 100 in both cases and other parameters to the default settings as set in Scikit-learn package [Pedregosa et al., 2011].

WJ: Need to add how they compute the uncertainty estimates. ? WJ: Add comments on uncertainty estimates of the trees.

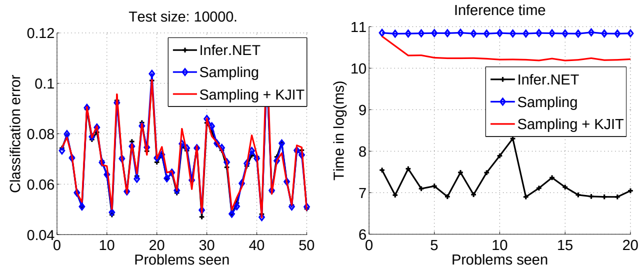
6.3 Online Learning

Fig. 5 shows predictive variance of KJIT on predicting the mean of each $m_{f\to X}$. The black dotted lines mark the start of a new inference problem. We set the number of observations in the logistic regression problem to 300. It can be seen that the uncertainty estimate exhibits a periodic structure, repeating itself for every 300 time steps. The predictive variance levels off after the operator sees roughly 1000 incoming messages (roughly one-third of the total messages in the first



(a) Binary classification error (b) Inference time

Figure 6: Classification performance and inference times of all methods.



(a) Binary classification error (b) Inference time

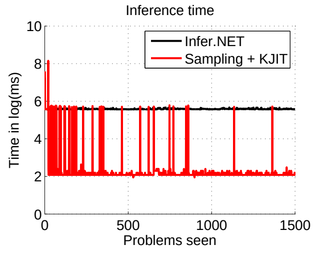
Figure 8: Classification performance and inference times in the logistic regression problem.

problem).

Fig. 8a shows binary classification errors obtained by using the inferred posterior mean parameter on a test set of size 10000 generated from the true parameter vector. The loss of KJIT matches that of the importance sampler and Infer.NET. Fig. 8b shows the inference time spent by all methods in each problem.

Experiments to have

- Use the operator for logistic in a binary logistic regression problem with a real dataset.
- A table comparing performance of different kernels. Put in the main text if there is enough space. If not, put it in the appendix. This provides an explanation for why we chose the Gaussian kernel on joint mean embeddgins.
- To determine thresholds, compute the median of log predictive variance on the test set split from the initial minibatch.



(a) KL divergence (b) Inference time

Figure 9: KL divergence of the posteriors and inference times in the compound Gamma problem.

7 Conclusions and Future Work

We propose to learn to send EP messages with kernel ridge regression by casting the KL minimization problem as a supervised learning problem. With random features, incoming messages to a learned operator are converted to a finite-dimensional vector. Computing an outgoing message amounts to computing the moment parameters by multiplying the vector with a matrix given by the solution of the primal ridge regression.

References

M. A. Alvarez, L. Rosasco, and N. D. Lawrence. Kernels for vector-valued functions: a review. *arXiv:1106.6251 [cs, math, stat]*, June 2011. URL http://arxiv.org/abs/1106.6251. arXiv: 1106.6251.

S. Barthelmé and N. Chopin. Abc-ep: Expectation propagation for likelihood-free bayesian computation. In L. Getoor and T. Scheffer, editors, *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, ICML '11, pages 289–296, New York, NY, USA, June 2011. ACM. ISBN 978-1-4503-0619-5.

C. M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006. ISBN 0387310738.

L. Breiman. Random Forests. *Mach. Learn.*, 45(1): 5–32, Oct. 2001. ISSN 0885-6125. doi: 10.1023/A:1010933404324. URL http://dx.doi.org/10.1023/A:1010933404324.

A. Caponnetto and E. De Vito. Optimal rates for the regularized least-squares algorithm. *Found. Comput. Math.*, 7(3):331–368, July 2007. ISSN 1615-3375. doi: 10.1007/s10208-006-0196-8. URL http://dx.doi.org/10.1007/s10208-006-0196-8.

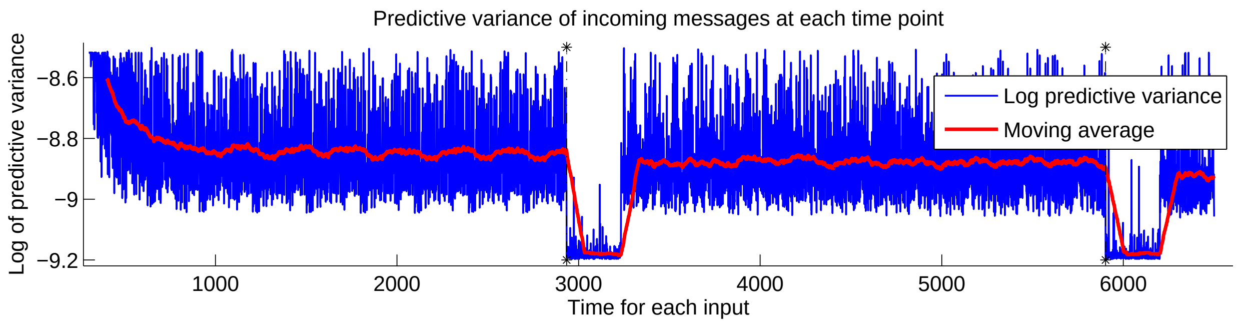


Figure 5: Uncertainty estimate of KJIT for all incoming messages at each time point in the binary logistic regression problem.

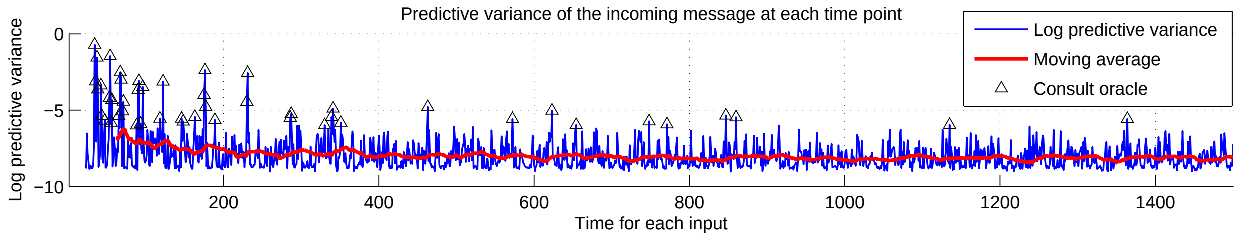


Figure 7: Uncertainty estimate of KJIT for all incoming messages at each time point in the compound Gamma problem.

A. Christmann and I. Steinwart. Universal kernels on non-standard input spaces. In *Advances in Neural Information Processing Systems*, pages 406–414, 2010a.

A. Christmann and I. Steinwart. Universal kernels on non-standard input spaces. In *Advances in neural information processing systems*, pages 406–414, 2010b.

A. Criminisi and J. Shotton. *Decision Forests for Computer Vision and Medical Image Analysis*. Springer Publishing Company, Incorporated, 2013.

B. Dai, B. Xie, N. He, Y. Liang, A. Raj, M. Balcan, and L. Song. Scalable kernel methods via doubly stochastic gradients. In *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada*, pages 3041–3049, 2014.

D. Duvenaud, J. Lloyd, R. Grosse, J. Tenenbaum, and Z. Ghahramani. Structure discovery in nonparametric regression through compositional kernel search. In *Proceedings of the 30th International Conference on Machine Learning*, 2013.

S. M. A. Eslami, D. Tarlow, P. Kohli, and J. Winn. Just-In-Time Learning for Fast and Flexible Inference. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 154–162, 2014.

A. Gelman. Prior distributions for variance parameters in hierarchical models. *Bayesian Analysis*, 1:1–19, 2006.

P. Geurts, D. Ernst, and L. Wehenkel. Extremely Randomized Trees. *Mach. Learn.*, 63(1):3–42, Apr. 2006. ISSN 0885-6125. doi: 10.1007/s10994-006-6226-1. URL http://dx.doi.org/10.1007/s10994-006-6226-1.

N. Goodman, V. Mansinghka, D. Roy, K. Bonawitz, , and J. Tenenbaum. Church: A language for generative models. In *Proc. of Uncertainty in Artificial Intelligence (UAI)*, 2008.

R. Grosse, R. Salakhutdinov, W. Freeman, and J. Tenenbaum. Exploiting compositionality to explore a large space of model structures. In *Conference on Uncertainty in Artificial Intelligence*, 2012.

S. Grünewälder, G. Lever, L. Baldassarre, S. Patterson, A. Gretton, and M. Pontil. Conditional mean embeddings as regressors. In J. Langford and J. Pineau, editors, *Proceedings of the 29th International Conference on Machine Learning*, pages 1823–1830, New York, NY, USA, 2012. Omnipress. URL http://icml.cc/2012/papers/898.pdf.

N. Heess, D. Tarlow, and J. Winn. Learning to pass expectation propagation messages. In C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Weinberger, editors, *Advances in Neural Information Processing Systems 26*, pages 3219–3227. 2013.

F. Hutter. *Automated Configuration of Algorithms for Solving Hard Computational Problems*. PhD thesis, University of British Columbia, Department of Computer Science, Vancouver, Canada, October 2009.

T. Jebara and R. Kondor. Bhattacharyya and expected likelihood kernels. In *Conference on Learning Theory*. press, 2003.

T. Jebara, R. Kondor, and A. Howard. Probability product kernels. *J. Mach. Learn. Res.*, 5:819–844, Dec. 2004. ISSN 1532-4435. URL http://dl.acm.org/citation.cfm?id=1005332.1016786.

B. Lakshminarayanan, D. Roy, and Y.-W. Teh. Mondrian forests: Efficient online random forests. In *Advances in neural information processing systems*, 2014.

Q. Le, T. Sarlos, and A. Smola. Fastfood - approximating kernel expansions in loglinear time. In *30th International Conference on Machine Learning (ICML)*, 2013. URL http://jmlr.org/proceedings/papers/v28/le13.html.

C. A. Micchelli and M. A. Pontil. On learning vector-valued functions. *Neural Comput.*, 17(1):177–204, Jan. 2005. ISSN 0899-7667. doi: 10.1162/0899766052530802. URL http://dx.doi.org/10.1162/0899766052530802.

T. Minka, J. Winn, J. Guiver, S. Webster, Y. Zaykov, B. Yangel, A. Spengler, and J. Bronskill. Infer.NET 2.6, 2014. Microsoft Research Cambridge. http://research.microsoft.com/infernet.

T. P. Minka. *A Family of Algorithms for Approximate Bayesian Inference*. PhD thesis, Massachusetts Institute of Technology, 2001. AAI0803033.

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12: 2825–2830, 2011.

A. Rahimi and B. Recht. Random features for large-scale kernel machines. In *Neural Information Processing Systems*, 2007.

C. E. Rasmussen and C. K. I. Williams. *Gaussian Processes for Machine Learning*. MIT Press, Cambridge, MA, 2006.

B. Schölkopf and A. J. Smola. *Learning with kernels : support vector machines, regularization, optimization, and beyond*. Adaptive computation and machine learning. MIT Press, 2002.

A. Smola, A. Gretton, L. Song, and B. Schölkopf. A hilbert space embedding for distributions. In *In Algorithmic Learning Theory: 18th International Conference*, pages 13–31. Springer-Verlag, 2007.

Stan Development Team. Stan: A c++ library for probability and sampling, version 2.4, 2014. URL http://mc-stan.org/.

Z. Szabo, A. Gretton, B. Poczos, and B. Sriperumbudur. Consistent, two-stage sampled distribution regression via mean embedding. Feb. 2014. URL http://arxiv-web3.library.cornell.edu/abs/1402.1754v2.

D. Wingate, N. Goodman, A. Stuhlmueller, and J. Siskind. Nonstandard interpretations of probabilistic programs for efficient inference. In *Advances in Neural Information Processing Systems*, 2011.

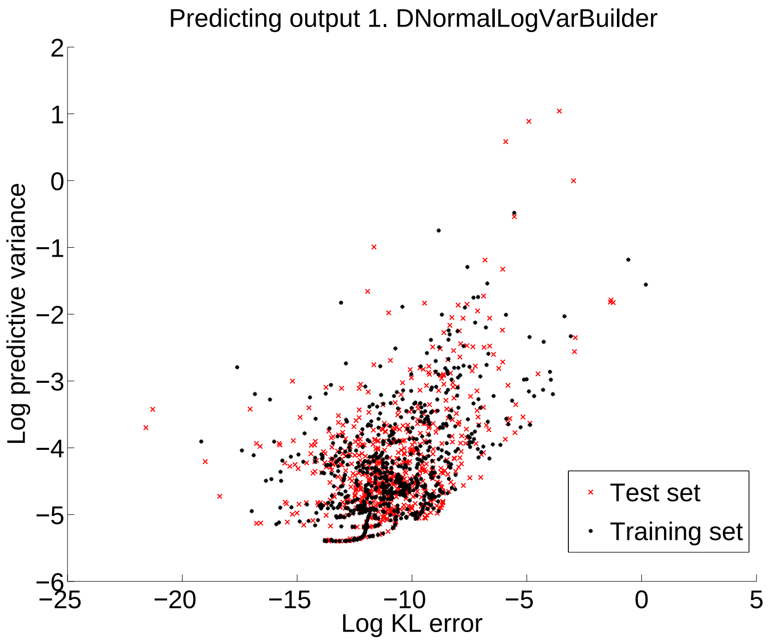


Figure 10: KL-divergence errors versus predictive variances for predicting the mean of $m_{f\to X}$ (normal distribution) in the logistic factor problem.

Supplementary Materials

The following incoherent content needs to be rewritten. I just copied it from my notes. Some part will be moved to the main text.

A Kernel Ridge Regression

A.1 Kernel Ridge Regression in Primal Form

In our work, we can apply random features to the kernel in the conditional mean embedding operator. With $\hat{\phi}$, an estimate of an operator in primal solution is

$$\begin{aligned}\hat{C}_{x'|w} &= X'W^\top \left(WW^\top + \lambda I\right)^{-1} \\ &\approx \underbrace{X'}_{d_x\times N} \underbrace{\hat{\Phi}^\top}_{N\times D} \underbrace{\left(\hat{\Phi}\hat{\Phi}^\top + \lambda I\right)^{-1}}_{D\times D}\end{aligned}$$

where $\hat{\Phi} = \left(\hat{\phi}(w_1)|\cdots|\hat{\phi}(w_N)\right) \in \mathbb{R}^{D\times N}$ and $W = X \otimes Y$. So an estimated operator can be represented with a $d_x \times D$ matrix where d_x is the dimension of the target sufficient statistic. Applying the operator to a test message $m^*_{w\to f}$ with random features given by $\phi^* \in \mathbb{R}^D$ requires just a matrix vector multiplication whose complexity does not depend on N (unlike in the dual form).

A.2 Leave-One-Out Cross Validation with Random Features

The solution of the kernel ridge regression is given by

$$\hat{C}_{Y|X} = YX^\top \left(XX^\top + \lambda I\right)^{-1}$$

where $Y \in \mathbb{R}^{d_y\times n}$ and d_y is the number of sufficient statistic outputs i.e., two for a one-dimensional Gaussian. With random features, we have $K_{ij} \approx \hat{\phi}_i^\top \hat{\phi}_j := \hat{\phi}(x_i)^\top \hat{\phi}(x_j)$ where $\hat{\phi}_i \in \mathbb{R}^d$ and d is the number of random features. In our work, x_i will be an instance in the tensor product space of the form $x_i = m_{x_1 \to f} \otimes m_{x_2 \to f} \otimes \cdots$. So the estimator can be rewritten as

$$\hat{C}_{Y|X} \approx \underbrace{Y}_{d_y \times n} \underbrace{\hat{\Phi}^\top}_{n \times d} \underbrace{\left(\hat{\Phi}\hat{\Phi}^\top + \lambda I\right)^{-1}}_{d \times d}$$

where $\hat{\Phi} := \left(\hat{\phi}_1 | \cdots | \hat{\phi}_n\right) \in \mathbb{R}^{d \times n}$. The LOOCV error can be written as

$$E_{LOOCV} := \frac{1}{n} \sum_{j=1}^{n} \| \hat{C}^{(j)}_{Y|X} \hat{\phi}_j - y_j \|^2_{d_y}$$

where $\hat{C}^{(j)}_{Y|X}$ is the estimator obtained with j^{th} instance removed and $y_j \in \mathbb{R}^{d_y}$. Following the same idea as in the ordinary ridge regression, we express $\hat{C}^{(j)}_{Y|X}$ in terms of $\hat{C}_{Y|X}$.

$$\begin{aligned}
\hat{C}^{(j)}_{Y|X} &= \left(Y\hat{\Phi} - y_j \phi_j^\top\right) \left(\hat{\Phi}\hat{\Phi}^\top + \lambda I - \hat{\phi}_j \hat{\phi}_j^\top\right)^{-1} \\
&= \left(Y\hat{\Phi} - y_j \phi_j^\top\right) \left(A^{-1} + \frac{A^{-1}\phi_j \phi_j^\top A^{-1}}{1 - \phi_j^\top A^{-1} \phi_j}\right) \\
&= \hat{C}_{Y|X} + h_{jj}^{-1} Y L \phi_j \phi_j^\top A^{-1} - h_{jj}^{-1} y_j \phi_j^\top A^{-1}
\end{aligned}$$

where $h_{ii} := (H)_{ii} = 1 - \phi_i^\top A^{-1} \phi_i$, $A := \hat{\Phi}\hat{\Phi}^\top + \lambda I$ and $L := \hat{\Phi}^\top A^{-1}$. Note that with these definitions, we have $\hat{C}_{Y|X} = YL$. By plugging $\hat{C}^{(j)}_{Y|X}$ back to E_{LOOCV}, we have

$$\begin{aligned}
E_{LOOCV} &= \frac{1}{n} \sum_{j=1}^{n} \left\| \hat{C}_{Y|X} \hat{\phi}_j + h_{jj}^{-1} Y L \hat{\phi}_j \underbrace{\hat{\phi}_j^\top A^{-1} \hat{\phi}_j}_{1 - h_{jj}} - h_{jj}^{-1} y_j \underbrace{\hat{\phi}_j A^{-1} \hat{\phi}_j}_{1 - h_{jj}} - y_j \right\|^2 \\
&= \frac{1}{n} \sum_{j=1}^{n} \left\| \hat{C}_{Y|X} \hat{\phi}_j + \left(h_{jj}^{-1} - 1\right) Y L \hat{\phi}_j - \left(h_{jj}^{-1} - 1\right) y_j - y_j \right\|^2_{d_y} \\
&= \frac{1}{n} \left\| \underbrace{\hat{C}_{Y|X}}_{YL} \hat{\Phi} + Y L \hat{\Phi} \left(\tilde{H}^{-1} - I\right) - Y \tilde{H}^{-1} \right\|^2_F \\
&= \frac{1}{n} \left\| Y L \hat{\Phi} \tilde{H}^{-1} - Y \tilde{H}^{-1} \right\|^2_Y = \frac{1}{n} \left\| Y \left(L\hat{\Phi} - I\right) \tilde{H}^{-1} \right\|^2_F \\
&= \frac{1}{n} \left\| Y H \tilde{H}^{-1} \right\|^2_F
\end{aligned}$$

where $H := I - L\hat{\Phi} = I - \hat{\Phi}^\top \left(\hat{\Phi}\hat{\Phi}^\top + \lambda I\right)^{-1} \hat{\Phi} \in \mathbb{R}^{n \times n}$, $\tilde{H}$ is a diagonal matrix with the same diagonal as H and $\|A\|_F^2 := \operatorname{tr}\left(A^\top A\right)$ is the squared Frobenius norm.

Computing E_{LOOCV} In computing E_{LOOCV}, one should not form $H \in \mathbb{R}^{n \times n}$ explicitly because n is assumed to be huge.

$$\begin{aligned}
\left\| Y H \tilde{H}^{-1} \right\|^2_F &= \operatorname{tr}\left(Y H \tilde{H}^{-2} H^\top Y^\top\right) \\
&= \operatorname{tr}\left(Y \tilde{H}^{-2} Y^\top - Y \tilde{H}^{-2} \hat{\Phi}^\top A^{-1} \hat{\Phi} Y^\top - Y \hat{\Phi}^\top A^{-1} \hat{\Phi} \tilde{H}^{-2} Y^\top + Y \hat{\Phi}^\top A^{-1} \hat{\Phi} \tilde{H}^{-2} \hat{\Phi}^\top A^{-1} \hat{\Phi} Y^\top\right)
\end{aligned}$$

Let $E := Y \hat{\Phi}^\top A^{-1} \hat{\Phi} \in \mathbb{R}^{d_y \times n}, B = Y \tilde{H}^{-1} \in \mathbb{R}^{d_y \times n}$ then the last line becomes

$$\begin{aligned}
&\operatorname{tr}\left(BB^\top - B\tilde{H}^{-1}E^\top - E\tilde{H}^{-1}B^\top + E\tilde{H}^{-2}E^\top\right) \\
&= \operatorname{tr}\left(BB^\top\right) - 2\operatorname{tr}\left(E\tilde{H}^{-1}B^\top\right) + \operatorname{tr}\left(E\tilde{H}^{-2}E^\top\right).
\end{aligned}$$

Each term in the trace is of size $d_y \times d_y$. This does not involve forming an $n \times n$ matrix. $\tilde{H}$ is the diagonal of H. So, $\tilde{H} = \mathrm{diag}\left(I - \hat{\Phi}^\top A^{-1}\hat{\Phi}\right)$. The cost of $O(d^3)$ seems to be inevitable where d is expected to be in the order of thousands. The number is large enough that d^3 is expensive. In computing E_{LOOCV} for selecting the best parameter combination, we may reduce d. The number of features d is set back to the desired value during testing.

Operator representation Once the best parameter combination is choosen, we increase d to the desired value and precompute $\hat{C}_{Y|X} \approx \underbrace{Y}_{d_y \times n} \underbrace{\hat{\Phi}^\top}_{n \times d} \underbrace{\left(\hat{\Phi}\hat{\Phi}^\top + \lambda I\right)^{-1}}_{d \times d}$ for test time. One obvious improvement is to avoid forming $\hat{\Phi}$ explicitly since the memory cost will be $O(dn)$ which can already be huge. $Y\hat{\Phi}$ can be computed incrementally without the need to form $\hat{\Phi}$. Memory cost for computing $\hat{\Phi}\hat{\Phi}^\top$ can be made lower than $O(dn)$ by computing incrementally even though the computational complexity remains $O(dn^2)$ which is still huge. The total cost for computing the operator is $O(d^2 n + d^3)$. Linear dependency on n is unavoidable.

B Kernels and Random Features

This section reviews relevant kernels and their random feature representations.

B.1 Random Features

This section contains a summary of Rahimi and Recht [2007]'s random Fourier features for a translation invariant kernel.

A kernel $k(x, y) = \langle \phi(x), \phi(y) \rangle$ in general may correspond to an inner product in an infinite-dimensional space whose feature map ϕ cannot be explicitly computed. In Rahimi and Recht [2007], methods of computing an approximate feature maps $\hat{\phi}$ were proposed. The approximate feature maps are such that $k(x, y) \approx \hat{\phi}(x)^\top \hat{\phi}(y)$ (with equality in expectation) where $\hat{\phi} \in \mathbb{R}^D$ and D is the number of random features. High D yields a better approximation with higher computational cost. Assume $k(x, y) = k(x - y)$ and $x, y \in \mathbb{R}^d$. Random Fourier features $\hat{\phi}(x) \in \mathbb{R}^D$ such that $k(x, y) \approx \hat{\phi}(x)^\top \hat{\phi}(y)$ are generated as follows.

1. Compute the Fourier transform $\hat{k}$ of the kernel k: $\hat{k}(\omega) = \frac{1}{2\pi} \int e^{-j\omega^\top \delta} k(\delta)\, d\delta$. For a Gaussian kernel with unit width, $\hat{k}(\omega) = (2\pi)^{-d/2}\, e^{-\|\omega\|^2/2}$.
2. Draw D i.i.d. samples $\omega_1, \ldots, \omega_D \in \mathbb{R}^d$ from $\hat{k}$.
3. Draw D i.i.d samples $b_1, \ldots, b_D \in \mathbb{R}$ from $U[0, 2\pi]$ (uniform distribution).
4. $\hat{\phi}(x) = \sqrt{\frac{2}{D}} \left(\cos\left(\omega_1^\top x + b_1\right), \ldots, \cos\left(\omega_D^\top x + b_D\right)\right)^\top \in \mathbb{R}^D$

Why it works ?

Theorem 1. *Bochner's theorem. A continuous kernel $k(x, y) = k(x - y)$ on $\mathbb{R}^m$ is positive definite iff $k(\delta)$ is the Fourier transform of a non-negative measure.*

Furthermore, if a translation invariant kernel $k(\delta)$ is properly scaled, Bochner's theorem guarantees that its Fourier transform $p(\omega)$ is a proper probability distribution. From this fact, we have

$$k(x - y) = \int \hat{k}(\omega) e^{j\omega^\top (x-y)}\, d\omega = \mathbb{E}_\omega\left[\eta_\omega(x)\eta_\omega(y)^*\right]$$

where $\eta_\omega(x) = e^{j\omega^\top x}$ and $\cdot^*$ denotes the complex conjugate. Since both p and k are real, the complex exponential contains only the cosine terms. Drawing d samples is for lowering the variance of the approximation.

Theorem 2. *Separation of variables. Let $\hat{f}$ be the Fourier transform of f. If $f(x_1, \ldots, x_n) = f_1(x_1) \cdots f_n(x_n)$, then $\hat{f}(\omega_1, \ldots, \omega_n) = \prod_{i=1}^n \hat{f}_i(\omega_i)$.*

Theorem 2 suggests that the random Fourier features can be extended to product kernel by drawing ω independently for each kernel.

B.2 MV (Mean-Variance) Kernel

Assume there are c incoming messages $\left(p^{(i)}\right)_{i=1}^{c}$. Assume that

$$
\begin{aligned}
\mathbb{E}_{p^{(l)}}[x] &= m_l \\
\mathbb{V}_{p^{(l)}}[x] &= v_l \\
\mathbb{E}_{q^{(l)}}[y] &= \mu_l \\
\mathbb{V}_{q^{(l)}}[y] &= \sigma_l^2.
\end{aligned}
$$

We use $p^{(l)}(x)$ and $m_{x_i \to f}(x)$ interchageably. Incoming messages are not necessarily Gaussian. They do have to be in the exponential family. MV (mean-variance) kernel is a a product kernel on means and variances.

$$
\begin{aligned}
\kappa\left(\left(p^{(i)}\right)_{i=1}^{c}, \left(q^{(i)}\right)_{i=1}^{c}\right) &= \prod_{i=1}^{c} \kappa^{(i)}\left(p^{(i)}, q^{(i)}\right) \\
&= \prod_{i=1}^{c} k\left(\left(m_i - \mu_i\right)/w_i^m\right) \prod_{i=1}^{c} k\left(\left(v_i - \sigma_i^2\right)/w_i^v\right)
\end{aligned}
$$

where k is a Gassian kernel with unit width. The kernel κ has $P := (w_1^m, \ldots, w_c^m, w_1^v, \ldots, w_c^v)$ as its parameters. With this kernel, we treat messages as finite dimensional vector. All incoming messages $(q^{(i)})_{i=1}^{c}$ are essentially represented as $\left(\mu_1, \ldots, \mu_c, \sigma_1^2, \ldots, \sigma_c^2\right)^\top$. This treatment reduces the problem of having distributions as inputs to the familiar problem of having input points from a Euclidean space. The random features of Rahimi and Recht [2007] can be applied straightforwardly.

B.3 Expected Product Kernel

Let $\mu_{r^{(l)}} := \mathbb{E}_{r^{(l)}(a)} k(\cdot, a)$ be the mean embedding [Smola et al., 2007] of the distribution $r^{(l)}$ into RKHS $\mathcal{H}^{(l)}$ induced by the kernel k. Assume $k = k_{\text{gauss}}$ (Gaussian kernel) and assume there are c incoming messages $\mathsf{x} := (r^{(i)}(a^{(i)}))_{i=1}^{c}$ and $\mathsf{y} := (s^{(i)}(b^{(i)}))_{i=1}^{c}$. An expected product kernel κ_{pro} is defined as

$$
\kappa_{\text{pro}}(\mathsf{x}, \mathsf{y}) := \left\langle \bigotimes_{l=1}^{c} \mu_{r^{(l)}}, \bigotimes_{l=1}^{c} \mu_{s^{(l)}} \right\rangle_{\otimes_l \mathcal{H}^{(l)}} = \prod_{l=1}^{c} \mathbb{E}_{r^{(l)}(a)} \mathbb{E}_{s^{(l)}(b)} k_{\text{gauss}}^{(l)}(a, b) \approx \hat{\phi}(\mathsf{x})^\top \hat{\phi}(\mathsf{y})
$$

where $\hat{\phi}(\mathsf{x})^\top \hat{\phi}(\mathsf{y}) = \prod_{l=1}^{c} \hat{\phi}^{(l)}(r^{(l)})^\top \hat{\phi}^{(l)}(s^{(l)})$. The feature map $\hat{\phi}^{(l)}(r^{(l)})$ can be estimated by applying the random Fourier features to $k_{\text{gauss}}^{(l)}$ and taking the expectations $\mathbb{E}_{r^{(l)}(a)} \mathbb{E}_{s^{(l)}(b)}$. The final feature map is $\hat{\phi}(\mathsf{x}) = \hat{\phi}^{(1)}(r^{(1)}) \circledast \hat{\phi}^{(2)}(r^{(2)}) \circledast \cdots \circledast \hat{\phi}^{(c)}(r^{(c)}) \in \mathbb{R}^{d^c}$ where $\circledast$ denotes a Kronecker product and we assume that $\hat{\phi}^{(l)} \in \mathbb{R}^d$ for $l \in \{1, \ldots, c\}$.

We first give some results which will be used to derive the Fourier features for inner product of mean embeddings.

Lemma 3. *If $b \sim \mathcal{N}(b; 0, \sigma^2)$, then $\mathbb{E}[\cos(b)] = \exp\left(-\frac{1}{2}\sigma^2\right)$.*

Proof. We can see this by considering the characteristic function of $x \sim \mathcal{N}(x; \mu, \sigma^2)$ which is given by $c_x(t)$.

$$
c_x(t) = \mathbb{E}_x\left[\exp\left(itb\right)\right] = \exp\left(itm - \frac{1}{2}\sigma^2 t^2\right)
$$

For $m = 0, t = 1$, we have

$$
c_b(1) = \mathbb{E}_b\left[\exp(ib)\right] = \exp\left(-\frac{1}{2}\sigma^2\right) = \mathbb{E}_b\left[\cos(b)\right]
$$

where we have cos() because the imaginary part is set to 0 i.e., $i\sin(tb)$ vanishes. □

Given two messages $p(x) = \mathcal{N}(x; m_p, V_p)$ and $q(y) = \mathcal{N}(y; m_q, V_q)$ (d-dimensional Gaussian), the expected product kernel is defined as

$$
\kappa(p, q) = \langle \mu_p, \mu_q \rangle_{\mathcal{H}} = \mathbb{E}_p \mathbb{E}_q k(x - y)
$$

where $\mu_p := \mathbb{E}_p k(x, \cdot)$ is the mean embedding of p and we assume that the kernel k associated with $\mathcal{H}$ is translation invariant i.e.g, $k(x, y) = k(x - y)$. The goal here is to derive random Fourier features for the expected product kernel. That is, we aim to find $\hat{\phi}$ such that $\kappa(p, q) \approx \hat{\phi}(p)^\top \hat{\phi}(q)$ and $\hat{\phi} \in \mathbb{R}^D$.

From Rahimi and Recht [2007] which provides random features for $k(x - y)$, we immediately have

$$\begin{aligned}
\mathbb{E}_p \mathbb{E}_q k(x - y) &\approx \mathbb{E}_p \mathbb{E}_q \frac{2}{D} \sum_{i=1}^{D} \cos\left(w_i^\top x + b_i\right) \cos\left(w_i^\top y + b_i\right) \\
&= \frac{2}{D} \sum_{i=1}^{D} \mathbb{E}_{p(x)} \cos\left(w_i^\top x + b_i\right) \mathbb{E}_{q(y)} \cos\left(w_i^\top y + b_i\right)
\end{aligned}$$

where $\{w_i\}_{i=1}^D \sim \hat{k}(w)$ (Fourier transform of k) and $\{b_i\}_{i=1}^D \sim U[0, 2\pi]$.

Consider $\mathbb{E}_{p(y)} \cos\left(w_i^\top x + b_i\right)$. Define $z_i = w_i^\top x + b_i$. So $z_i \sim \mathcal{N}(z_i; w_i^\top m_p + b_i, w_i^\top V_p w_i)$. Let $d_i \sim \mathcal{N}(0, w_i^\top V_p w_i)$. Then, $p(d_i + w_i^\top m_p + b_i) = \mathcal{N}(w_i^\top m_p + b_i, w_i^\top V_p w_i)$ which is the same distribution as that of z_i. From these definitions we have,

$$\begin{aligned}
\mathbb{E}_{p(x)} \cos\left(w_i^\top x + b_i\right) &= \mathbb{E}_{p(z_i)} \cos(z_i) \\
&= \mathbb{E}_{p(d_i)} \cos\left(d_i + w_i^\top m_p + b_i\right) \\
&\overset{(a)}{=} \mathbb{E}_{p(d_i)} \cos(d_i) \cos(w_i^\top m_p + b_i) - \mathbb{E}_{p(d_i)} \sin(d_i) \sin(w_i^\top m_p + b_i) \\
&\overset{(b)}{=} \cos(w_i^\top m_p + b_i) \mathbb{E}_{p(d_i)} \cos(d_i) \\
&\overset{(c)}{=} \cos(w_i^\top m_p + b_i) \exp\left(-\frac{1}{2} w_i^\top V_p w_i\right)
\end{aligned}$$

where at (a) we use $\cos(\alpha + \beta) = \cos(\alpha)\cos(\beta) - \sin(\alpha)\sin(\beta)$. We have (b) because sin() is an odd function and $\mathbb{E}_{p(d_i)} \sin(d_i) = 0$. The last equality (c) follows from Lemma 3. It follows that the random features $\hat{\phi}(p) \in \mathbb{R}^D$ are given by

$$\hat{\phi}(p) = \sqrt{\frac{2}{D}} \begin{pmatrix} \cos(w_1^\top m_p + b_1) \exp\left(-\frac{1}{2} w_1^\top V_p w_1\right) \\ \vdots \\ \cos(w_D^\top m_p + b_D) \exp\left(-\frac{1}{2} w_D^\top V_p w_D\right) \end{pmatrix}.$$

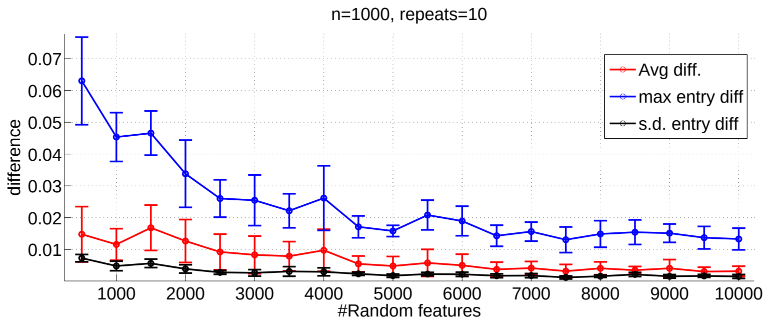
Notice that the translation invariant kernel k (i.e., Laplacian) plays a role by providing $\hat{k}$ from which $\{w_i\}_i$ are drawn. For different types of distributions p, q, we only need to be able to compute $\mathbb{E}_{p(x)} \cos\left(w_i^\top x + b_i\right)$. With $\hat{\phi}(p)$, we have $\kappa(p, q) \approx \hat{\phi}(p)^\top \hat{\phi}(q)$ with equality in expectation.

Analytic expression for Gaussian case For reference, if p, q are Gaussians defined as before and k is also Gaussian kernel, an analytic expression is available. Assume $k(x - y) = \exp\left(-\frac{1}{2}(x - y)^\top \Sigma^{-1} (x - y)\right)$ where Σ is the kernel parameter.

$$\begin{aligned}
\mathbb{E}_p \mathbb{E}_q k(x - y) &= \sqrt{\frac{\det(D_{pq})}{\det(\Sigma^{-1})}} \exp\left(-\frac{1}{2} (m_p - m_q)^\top D_{pq} (m_p - m_q)\right) \\
D_{pq} &:= (V_p + V_q + \Sigma)^{-1}
\end{aligned}$$

This is useful in checking the approximation accuracy of $\hat{\phi}$.

Approximation Quality The following result compares the randomly generated features to the true kernel matrix using various number of D. For each D, we repeat 10 times. Sample size is 1000. The derivation seems to be correct. Gaussian kernel width $\sigma^2 := 3$. "max entry diff" refers to the maximum entry-wise difference between the true kernel matrix and the approxmated kernel matrix. Messages are one-dimensional Gaussians with randomly drawn means and variances.



B.4 Product Kernel on Mean Embeddings

The kernel κ is defined as a product of kernels on mean embeddings. For each incoming message l, we need one set of random weights $\{w_i^{(l)}\}_{i=1}^D \sim \hat{k}^{(l)}$ and $\{b_i^{(l)}\}_{i=1}^D$.

$$\begin{aligned}\kappa(p,q) &= \left\langle \bigotimes_{l=1}^{c} \mu_{p^{(l)}}, \bigotimes_{l=1}^{c} \mu_{q^{(l)}} \right\rangle_{\otimes_l \mathcal{H}^{(l)}} \\ &= \prod_{l=1}^{c} \left\langle \mu_{p^{(l)}}, \mu_{q^{(l)}} \right\rangle_{\mathcal{H}^{(l)}} \\ &= \prod_{l=1}^{c} \mathbb{E}_{p^{(l)}(x)} \mathbb{E}_{q^{(l)}(y)} k^{(l)}(x-y) \\ &\approx \prod_{l=1}^{c} \hat{\phi}^{(l)}(p^{(l)})^\top \hat{\phi}^{(l)}(q^{(l)}) \\ &= \hat{\varphi}(p)^\top \hat{\varphi}(q)\end{aligned}$$

where $\hat{\varphi}(p) := \left(\hat{\phi}_{i_1}^{(1)} \times \cdots \hat{\phi}_{i_c}^{(c)}\right)_{i_1 \cdots i_c = 1}^{c} = \hat{\phi}^{(1)} \circledast \hat{\phi}^{(2)} \circledast \cdots \circledast \hat{\phi}^{(c)} \in \mathbb{R}^{D^c}$. The symbol $\circledast$ denotes a Kronecker product and we assume that each $\hat{\phi}^{(l)} \in \mathbb{R}^D$. Without a special solver which can take advantage of the Kronecker product in $\hat{\varphi}$, this random feature map might be impractical due to large memory requirement unless D is very small.

B.5 Sum Kernel on Mean Embeddings

If we instead define κ as the sum of c kernels, we have

$$\begin{aligned}\kappa(p,q) &= \sum_{l=1}^{c} \left\langle \mu_{p^{(l)}}, \mu_{q^{(l)}} \right\rangle_{\mathcal{H}^{(l)}} \\ &\approx \sum_{l=1}^{c} \hat{\phi}^{(l)}(p^{(l)})^\top \hat{\phi}^{(l)}(q^{(l)}) \\ &= \hat{\varphi}(p)^\top \hat{\varphi}(q)\end{aligned}$$

where $\hat{\varphi}(p) := \left(\hat{\phi}^{(1)}(p^{(1)})^\top, \ldots, \hat{\phi}^{(c)}(p^{(c)})^\top\right)^\top \in \mathbb{R}^{cD}$ whose memory requirement is much lower.

B.5.1 Kernel on tuples of messages

As our prediction is from $M_j = [m^j_{V_i \to f}(v_i)]_{i=1}^c$ (c is the number of variables connected to the factor f) to y_j, in the end, the kernel we need is $\kappa(M_i, M_j)$ which is defined on message tuples.

$$\kappa(M_i, M_j) = \prod_{l=1}^{c} \kappa_l \left(m^i_{V_l \to f}, m^j_{V_l \to f} \right)$$

which is a product of Gaussian kernels on mean embedding of each message. With random features, we have

$$\kappa(M_i, M_j) \approx \prod_{l=1}^{c} \hat{\psi}_l(p_l^i)^\top \hat{\psi}_l(p_l^j) = \hat{\Psi}(M_i)^\top \hat{\Psi}(M_j)$$

where $\hat{\Psi}(M_i) := \left(\left(\hat{\psi}_1(p_1^i)_{i_1} \right) \times \cdots \times \left(\hat{\psi}_c(p_c^i)_{i_c} \right) \right)_{i_1 \cdots i_c = 1}^{c} = \hat{\psi}_1(p_1^i) \circledast \hat{\psi}_2(p_2^i) \circledast \cdots \circledast \hat{\psi}_c(p_c^i) \in \mathbb{R}^{D_{out}^c}$. The symbol $\circledast$ denotes a Kronecker product. The total number of random features is given by $D = (D_{out})^c$. It can be seen that the total number of features grows exponentially in the number of connected variables. We can also use

$$\kappa\left(M_i, M_j\right) = \sum_{l=1}^{c} \kappa_l \left(m_{V_l \to f}^i, m_{V_l \to f}^j \right)$$

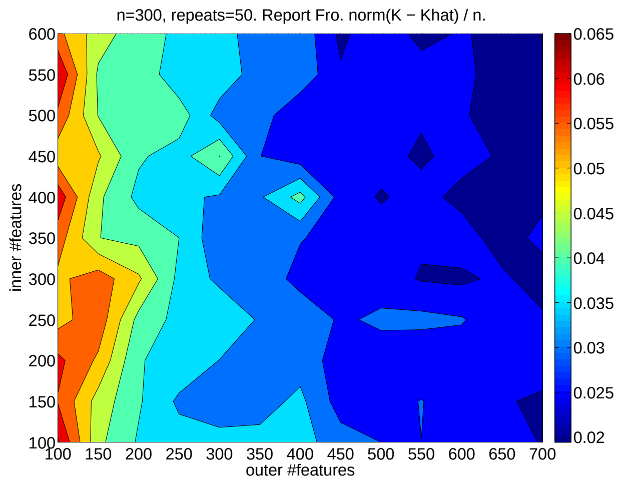
which gives $D = cD_{out}$. We expect the product kernel to be able to better capture the interaction among incoming messages.

Yet another way to form a kernel on tuples of messages is to consider a joint embedding. Let $\mathbb{M}_i := \prod_{l=1}^{c} m_{V_l \to f}^i(v_l)$ e.g., a product of all incoming messages. We can treat $\mathbb{M}_i$ as a joint distribution of one variable $V := (v_1, \ldots, v_c)$. A joint embedding kernel is straightforwardly given by defining a Gaussian kernel on mean embeddings of these joint distributions:

$$\kappa(M_i, M_j) = \kappa_{\text{gauss}}(\mathbb{M}_i, \mathbb{M}_j).$$

An advantage of this kernel is that the final number of random features is D_{out} as compared to D_{out}^c (product kernel) and cD_{out} (sum kernel). One potential disadvantage is that we treat all neighbouring variables as independent.

Experiments on Random Features for Gaussian Kernel on Mean Embeddings It is unclear how D_{in} and D_{out} affect the quality of the random feature approximation. We quantify the effect empirically as follows. We generate 300 Gaussian messages, compute the true gram matrix and the approximate gram matrix given by the random features, and report the Frobenius norm of the difference of the two matrices on a grid of D_{in} and D_{out}. For each (D_{in}, D_{out}), we repeat 20 times with a different set of random features and report the averaged Frobenius norm.



The result suggests that D_{out} has more effect in improving the approximation.

Supplementary

[sup1] M. A. Alvarez, L. Rosasco, and N. D. Lawrence. Kernels for vector-valued functions: a review. *arXiv:1106.6251 [cs, math, stat]*, June 2011. URL http://arxiv.org/abs/1106.6251. arXiv: 1106.6251.

[sup27] A. Rahimi and B. Recht. Random features for large-scale kernel machines. In *Neural Information Processing Systems*, 2007.

[sup30] A. Smola, A. Gretton, L. Song, and B. Schölkopf. A hilbert space embedding for distributions. In *In Algorithmic Learning Theory: 18th International Conference*, pages 13–31. Springer-Verlag, 2007.