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

Wittawat Jitkrittum,[1] **Arthur Gretton,**[1] **Nicolas Heess,**[*] **S. M. Ali Eslami**[*]
Balaji Lakshminarayanan,[1] **Dino Sejdinovic**[2] and **Zoltán Szabó**[1]

Gatsby Unit, University College London[1]
University of Oxford[2]

{wittawatj, arthur.gretton}@gmail.com, nheess@gmail.com
ali@arkitus.com, balaji@gatsby.ucl.ac.uk,
dino.sejdinovic@gmail.com, zoltan.szabo@gatsby.ucl.ac.uk

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. This learned operator replaces the multivariate integral required in classical EP, which may not have an analytic expression. 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 it is essential to accurately assess uncertainty and to efficiently and robustly update the message operator.

1 INTRODUCTION

An increasing priority in Bayesian modelling is to make inference accessible and implementable for practitioners, without requiring specialist knowledge. This is a goal sought, for instance, in probabilistic programming languages (Wingate et al., 2011; Goodman et al., 2008), as well as in more granular, component-based 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 inverse gamma prior, which is widely used as a convenient conjugate prior for the variance, has quite pathological behaviour (Gelman, 2006). In general, more expressive, freely chosen models are more likely to require expensive sampling or quadrature approaches, 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 achievable 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 these estimates converge to the desired integrals for a sufficient number of importance samples, the sampling procedure must be run at every iteration during inference, hence 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

[*] Currently at Google DeepMind.

of the sampler. Heess et al. (2013) use neural networks to learn the mapping from incoming to outgoing messages, and the learned mappings perform well on a variety of practical problems. 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 (e.g., datasets representative of all classification problems the user proposes to solve), and it has no way of assessing the uncertainty of its prediction, or of updating the model online in the event that a prediction is uncertain.

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 provide uncertainty estimates for each prediction. This allows them to be trained 'just-in-time', during EP inference, whenever the predictor decides it is uncertain. Uncertainty estimation for random forests relies on unproven heuristics, however: we demonstrate empirically that such heuristics can become highly misleading as we move away from the initial training data. Moreover, online updating can result in unbalanced trees, resulting in a cost of prediction of $O(N)$ for training data of size N, rather than the ideal of $O(\log(N))$.

We propose a novel, kernel-based approach to learning a message operator nonparametrically for expectation propagation. The learning algorithm takes the form of a distribution regression problem, where the inputs are probability measures represented as embeddings of the distributions to a reproducing kernel Hilbert space (RKHS), and the outputs are vectors of message parameters (Szabó 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, 2010, eq. 9). To make the algorithm computationally tractable, we regress directly in the primal from random Fourier features of the data (Rahimi and Recht, 2007; Le et al., 2013; Yang et al., 2015). In particular, we establish a novel random feature representation for when inputs are distributions, via a two-level random feature approach. This gives us both fast prediction (linear in the number of random features), and fast online updates (quadratic 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, i.e., the uncertain predictions trigger just-in-time updates of the regressor. We demonstrate empirically that our uncertainty estimates are more robust and informative than those for random forests, especially as we move away from the training data.

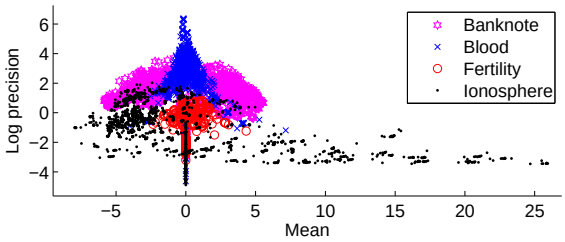


Figure 1: Distributions of incoming messages to logistic factor in four different UCI datasets.

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 training data for the message operator. We also give a brief overview of previous learning approaches to the problem, with a focus on that of Eslami et al. (2014). Next, in Section 3, we describe our kernel regression approach, and the form of an efficient kernel message operator mapping the input messages (distributions embedded in an RKHS) to outgoing messages (sets of parameters of the outgoing messages). Finally, in Section 4, we describe our experiments, which cover three topics: a benchmark of our uncertainty estimates, a demonstration of factor learning on artificial data with well-controlled statistical properties, and a logistic regression experiment on four different real-world datasets, demonstrating that our just-in-time learner can correctly evaluate its uncertainty and update the regression function as the incoming messages change (see Fig. 1). Code to implement our method is available online at `https://github.com/wittawatj/kernel-ep`.

2 BACKGROUND

We assume that distributions (or densities) p over a set of variables $\mathbf{x} = (x_1, \dots x_d)$ of interest can be represented as factor graphs, i.e. $p(\mathbf{x}) = \frac{1}{Z}\prod_{j=1}^{J} f_j(\mathbf{x}_{\mathrm{ne}(f_j)})$. The factors f_j are non-negative functions which are defined over subsets $\mathbf{x}_{\mathrm{ne}(f_j)}$ of the full set of variables $\mathbf{x}$. These variables form the neighbors of the factor node f_j in the factor graph, and we use $\mathrm{ne}(f_j)$ to denote the corresponding set of indices. Z is the normalization constant.

We deal with models in which some of the factors have a non-standard form, or may not have a known analytic expression (i.e. "black box" factors). Although our approach applies to any such factor in principle, in this paper we focus on *directed* factors $f(\mathbf{x}_{\mathrm{out}}|\mathbf{x}_{\mathrm{in}})$ which specify a conditional distribution over variables $\mathbf{x}_{\mathrm{out}}$ given $\mathbf{x}_{\mathrm{in}}$ (and thus $\mathbf{x}_{\mathrm{ne}(f)} = (\mathbf{x}_{\mathrm{out}}, \mathbf{x}_{\mathrm{in}})$). 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 $\mathbf{x}_{\text{out}} \sim f(\cdot|\mathbf{x}_{\text{in}})$. In particular, the ability to evaluate the value of $f(\mathbf{x}_{\text{out}}|\mathbf{x}_{\text{in}})$ is not assumed. A natural way to specify f is as code in a probabilistic program.

2.1 EXPECTATION PROPAGATION

Expectation Propagation (EP) is an approximate iterative procedure for computing marginal beliefs of variables by iteratively passing messages between variables and factors until convergence (Minka, 2001). It can be seen as an alternative to belief propagation, where the marginals are projected onto a member of some class of known parametric distributions. The message $m_{f\to V}(x_V)$ from factor f to variable $V \in \text{ne}(f)$ is

$$\frac{\text{proj}\left[\int f(\mathbf{x}_{\text{ne}(f)}) \prod_{V' \in \text{ne}(f)} m_{V'\to f}(x_{V'}) \text{d}\mathbf{x}_{\text{ne}(f)\setminus V}\right]}{m_{V\to f}(x_V)}, \tag{1}$$

where $m_{V'\to f}$ are the messages sent to factor f from all of its neighboring variables $x_{V'}$, $\text{proj}\,[p] = \text{argmin}_{q\in\mathcal{Q}}\text{KL}\,[p||q]$, and $\mathcal{Q}$ is typically in the exponential family, e.g. the set of Gaussian or Beta distributions.

Computing the numerator of (1) can be challenging, as it requires evaluating a high-dimensional integral as well as minimization of the Kullback-Leibler divergence to some non-standard distribution. Even for factors with known analytic form this often requires hand-crafted approximations, or the use of expensive numerical integration techniques; for "black-box" factors implemented as forward sampling functions, fully nonparametric techniques are needed.

Barthelmé and Chopin (2011); Heess et al. (2013); Eslami et al. (2014) propose an alternative, stochastic approach to the integration and projection step. When the projection is to a member $q(x|\eta) = h(x)\exp\left(\eta^\top u(x) - A(\eta)\right)$ of an exponential family, one simply computes the expectation of the sufficient statistic $u(\cdot)$ under the numerator of (1). A sample based approximation of this expectation can be obtained via Monte Carlo simulation. Given a forward-sampling function f as described above, one especially simple approach is importance sampling,

$$\mathbb{E}_{\mathbf{x}_{\text{ne}(f)}\sim b}\left[u(x_V)\right] \approx \frac{1}{M}\sum_{l=1}^{M} w(\mathbf{x}^l_{\text{ne}(f)})u(x^l_V), \tag{2}$$

where $\mathbf{x}^l_{\text{ne}(f)} \sim \tilde{b}$, for $l = 1,\ldots,M$ and on the left hand side,

$$b(\mathbf{x}_{\text{ne}(f)}) = f(\mathbf{x}_{\text{ne}(f)}) \prod_{W\in\text{ne}(f)} m_{W\to f}(x_W).$$

On the right hand side we draw samples $\mathbf{x}^l_{\text{ne}(f)}$ from some proposal distribution $\tilde{b}$ which we choose to be $\tilde{b}(\mathbf{x}_{\text{ne}(f)}) = r(\mathbf{x}_{\text{in}})f(\mathbf{x}_{\text{out}}|\mathbf{x}_{\text{in}})$ for some distribution r with appropriate support, and compute importance weights

$$w(\mathbf{x}_{\text{ne}(f)}) = \frac{\prod_{W\in\text{ne}(f)} m_{W\to f}(x_W)}{r(\mathbf{x}_{\text{in}})}.$$

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

2.2 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 when the number of samples M is small, and the importance sampling estimate is not reliable. On the other hand, for large M the computational cost of running EP with approximate messages can be very high, as importance sampling must be performed for sending each outgoing message. To obtain low-variance message approximations at lower computational cost, Heess et al. (2013) and 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 $(m_{V'\to f})_{V'\in\text{ne}(f)}$ to an approximate factor to variable message $m_{f\to V}$, i.e. they learn a mapping

$$M^\theta_{f\to V} : (m_{V'\to f})_{V'\in\text{ne}(f)} \mapsto m_{f\to V}, \tag{3}$$

where θ are the parameters of the approximator.

Heess et al. (2013) use neural networks and a large, fixed training set to learn their approximate message operator prior to running EP. By contrast, Eslami et al. (2014) employ random forests as their class of learning functions, and update their approximate message operator on the fly during inference, depending on the predictive uncertainty of the current message operator. Specifically, they endow their function approximator with an uncertainty estimate

$$\mathfrak{V}^\theta_{f\to V} : (m_{V'\to f})_{V'\in\text{ne}(f)} \mapsto \mathfrak{v}, \tag{4}$$

where $\mathfrak{v}$ indicates the expected unreliability of the predicted, approximate message $m_{f\to V}$ returned by $M^\theta_{f\to V}$. If $\mathfrak{v} = \mathfrak{V}^\theta_{f\to V}\left((m_{V'\to f})_{V'\in\text{ne}(f)}\right)$ exceeds a predefined threshold, the required message is approximated via importance sampling (cf. (2)) and $M^\theta_{f\to V}$ is updated on this new datapoint (leading to a new set of parameters θ' with $\mathfrak{V}^{\theta'}_{f\to V}\left((m_{V'\to f})_{V'\in\text{ne}(f)}\right) < \mathfrak{V}^\theta_{f\to V}\left((m_{V'\to f})_{V'\in\text{ne}(f)}\right)$.

Eslami et al. (2014) estimate the predictive uncertainty $\mathfrak{V}^\theta_{f\to V}$ via the heuristic of looking at the variability of the forest predictions for each point (Criminisi and Shotton, 2013). They implement their online updates by splitting the trees at their leaves. Both these mechanisms can be 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 4, we demonstrate that uncertainty heuristics for random forests 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 are from a very different distribution to the initial training sample (e.g. Lakshminarayanan et al., 2014, Fig. 3). For large amounts of training set drift, the leaf-splitting approach of Eslami et al. can result in a decision tree in the form of a long chain, giving a worst case cost of prediction (computational and storage) of $O(N)$ for training data of size N, vs the ideal of $O(\log(N))$ for balanced trees. Finally, note that the approach of Eslami et al. uses certain bespoke features of the factors when specifying tree traversal in the random forests, notably the value of the factor potentials at the mean and mode of the incoming messages. These features require expert knowledge of the model on the part of the practitioner, and are not available in the "forward sampling" setting. The present work does not employ such features.

In terms of computational cost, prediction for the random forest of Eslami et al. costs $O(KD_rD_t\log(N))$, and updating following a new observation costs $O(KD_r^3D_t\log(N))$, where K is the number of trees in the random forest, D_t is the number of features used in tree traversal, D_r is the number of features used in making predictions at the leaves, and N is the number of training messages. Representative values are $K = 64$, $D_t = D_r \approx 15$, and N in the range of 1,000 to 5,000.

3 KERNEL LEARNING OF OPERATORS

We now propose a kernel regression method for jointly learning the message operator $M^\theta_{f\to V}$ and uncertainty estimate $\mathfrak{V}^\theta_{f\to V}$. We regress from the tuple of incoming messages, which are probability distributions, to the parameters of the outgoing message. To this end we apply a kernel over distributions from (Christmann and Steinwart, 2010) to the case where the input consists of more than one distribution.

We note that Song et al. (2010, 2011) propose a related regression approach for predicting outgoing messages from incoming messages, for the purpose of belief propagation. Their setting is different from ours, however, as their messages are smoothed conditional density functions rather than parametric distributions of known form.

To achieve fast predictions and factor updates, we follow Rahimi and Recht (2007); Le et al. (2013); Yang et al. (2015), and express the kernel regression in terms of random features whose expected inner product is equal to the kernel function; i.e. we perform regression directly in the primal on these random features. In Section 3.1, we define our kernel on tuples of distributions, and then derive the corresponding random feature representation in Section 3.2. Section 3.3 describes the regression algorithm, as well as our strategy for uncertainty evaluation and online updates.

3.1 KERNELS ON TUPLES OF DISTRIBUTIONS

In the following, we consider only a single factor, and therefore drop the factor identity from our notation. We write the set of c incoming messages to a factor node as a tuple of probability distributions $R := (r^{(l)})_{l=1}^c$ of random variables $X^{(l)}$ on respective domains $\mathcal{X}^{(l)}$. Our goal is to define a kernel between one such tuple, and a second one, which we will write $S := (s^{(l)})_{l=1}^c$.

We define our kernel in terms of embeddings of the tuples R, S into a reproducing kernel Hilbert space (RKHS). We first consider the embedding of a single distribution in the tuple: Let us define an RKHS $\mathcal{H}^{(l)}$ on each domain, with respective kernel $k^{(l)}(x_1^{(l)}, x_2^{(l)})$. We may embed individual probability distributions to these RKHSs, following (Smola et al., 2007). The *mean embedding* of $r^{(l)}$ is written

$$\mu_{r^{(l)}}(\cdot) := \int k^{(l)}(x^{(l)}, \cdot)\, dr^{(l)}(x^{(l)}). \tag{5}$$

Similarly, a mean embedding may be defined on the product of messages in a tuple $\mathsf{r} = \times_{l=1}^c r^{(l)}$ as

$$\mu_{\mathsf{r}} := \int k([x^{(1)}, \ldots, x^{(c)}], \cdot)\, d\mathsf{r}(x^{(1)}, \ldots, x^{(c)}), \tag{6}$$

where we have defined the joint kernel k on the product space $\mathcal{X}^{(1)} \times \cdots \times \mathcal{X}^{(c)}$. Finally, a kernel on two such embeddings $\mu_{\mathsf{r}}, \mu_{\mathsf{s}}$ of tuples R, S can be obtained as in Christmann and Steinwart (2010, eq. 9),

$$\kappa(\mathsf{r}, \mathsf{s}) = \exp\left(-\frac{\|\mu_{\mathsf{r}} - \mu_{\mathsf{s}}\|^2_{\mathcal{H}}}{2\gamma^2}\right). \tag{7}$$

This kernel has two parameters: γ^2, and the width parameter of the kernel k defining $\mu_{\mathsf{r}} = \mathbb{E}_{x\sim\mathsf{r}} k(x, \cdot)$.

We have considered several alternative kernels on tuples of messages, including kernels on the message parameters, kernels on a tensor feature space of the distribution embeddings in the tuple, and dot products of the features (6). We have found these alternatives to have worse empirical performance than the approach described above. We give details of these experiments in Section C of the supplementary material.

3.2 RANDOM FEATURE APPROXIMATIONS

One approach to learning the mapping $M^\theta_{f\to V}$ from incoming to outgoing messages would be to employ Gaussian process regression, using the kernel (7). This approach is

not suited to just-in-time (JIT) learning, however, as both prediction and storage costs grow with the size of the training set; thus, inference on even moderately sized datasets rapidly becomes computationally prohibitive. Instead, we define a finite-dimensional random feature map $\hat{\psi} \in \mathbb{R}^{D_{\text{out}}}$ such that $\kappa(\mathsf{r}, \mathsf{s}) \approx \hat{\psi}(\mathsf{r})^\top \hat{\psi}(\mathsf{s})$, and regress directly on these feature maps in the primal (see next section): storage and computation are then a function of the dimension of the feature map D_{out}, yet performance is close to that obtained using a kernel.

In Rahimi and Recht (2007), a method based on Fourier transforms was proposed for computing a vector of random features $\hat{\varphi}$ for a translation invariant kernel $k(x, y) = k(x - y)$ such that $k(x, y) \approx \hat{\varphi}(x)^\top \hat{\varphi}(y)$ where $x, y \in \mathbb{R}^d$ and $\hat{\varphi}(x), \hat{\varphi}(y) \in \mathbb{R}^{D_{\text{in}}}$. This is possible because of Bochner's theorem (Rudin, 2013), which states that a continuous, translation-invariant kernel k can be written in the form of an inverse Fourier transform:

$$k(x - y) = \int \hat{k}(\omega) e^{j\omega^\top (x-y)} \, d\omega,$$

where $j = \sqrt{-1}$ and the Fourier transform $\hat{k}$ of the kernel can be treated as a distribution. The inverse Fourier transform can thus be seen as an expectation of the complex exponential, which can be approximated with a Monte Carlo average by drawing random frequencies from the Fourier transform. We will follow a similar approach, and derive a two-stage set of random Fourier features for (7).

We start by expanding the exponent of (7) as

$$\exp\left(-\frac{1}{2\gamma^2}\langle \mu_\mathsf{r}, \mu_\mathsf{r}\rangle + \frac{1}{\gamma^2}\langle \mu_\mathsf{r}, \mu_\mathsf{s}\rangle - \frac{1}{2\gamma^2}\langle \mu_\mathsf{s}, \mu_\mathsf{s}\rangle\right).$$

Assume that the embedding kernel k used to define the embeddings μ_r and μ_s is translation invariant. Since $\langle \mu_\mathsf{r}, \mu_\mathsf{s}\rangle = \mathbb{E}_{x\sim\mathsf{r}}\mathbb{E}_{y\sim\mathsf{s}} k(x - y)$, one can use the result of Rahimi and Recht (2007) to write

$$\begin{aligned}\langle \mu_\mathsf{r}, \mu_\mathsf{s}\rangle &\approx \mathbb{E}_{x\sim\mathsf{r}}\mathbb{E}_{y\sim\mathsf{s}} \hat{\varphi}(x)^\top \hat{\varphi}(y) \\ &= \mathbb{E}_{x\sim\mathsf{r}} \hat{\varphi}(x)^\top \mathbb{E}_{y\sim\mathsf{s}} \hat{\varphi}(y) := \hat{\phi}(\mathsf{r})^\top \hat{\phi}(\mathsf{s}),\end{aligned}$$

where the mappings $\hat{\phi}$ are D_{in} standard Rahimi-Recht random features, shown in Steps 1-3 of Algorithm 1.

With the approximation of $\langle \mu_\mathsf{r}, \mu_\mathsf{s}\rangle$, we have

$$\kappa(\mathsf{r}, \mathsf{s}) \approx \exp\left(-\frac{\|\hat{\phi}(\mathsf{r}) - \hat{\phi}(\mathsf{s})\|^2_{D_{\text{in}}}}{2\gamma^2}\right), \tag{8}$$

which is a standard Gaussian kernel on $\mathbb{R}^{D_{\text{in}}}$. We can thus further approximate this Gaussian kernel by the random Fourier features of Rahimi and Recht, to obtain a vector of random features $\hat{\psi}$ such that $\kappa(\mathsf{r}, \mathsf{s}) \approx \hat{\psi}(\mathsf{r})^\top \hat{\psi}(\mathsf{s})$ where $\hat{\psi}(\mathsf{r}), \hat{\psi}(\mathsf{s}) \in \mathbb{R}^{D_{\text{out}}}$. Pseudocode for generating the random features $\hat{\psi}$ is given in Algorithm 1. Note that the sine component in the complex exponential vanishes due to the translation invariance property (analogous to an even function), i.e., only the cosine term remains. We refer to Section B.3 in the supplementary material for more details.

Algorithm 1 Construction of two-stage random features for κ

Input: Input distribution r, Fourier transform $\hat{k}$ of the embedding translation-invariant kernel k, number of inner features D_{in}, number of outer features D_{out}, outer Gaussian width γ^2.

Output: Random features $\hat{\psi}(\mathsf{r}) \in \mathbb{R}^{D_{\text{out}}}$.

1: Sample $\{\omega_i\}_{i=1}^{D_{\text{in}}} \overset{i.i.d}{\sim} \hat{k}$.

2: Sample $\{b_i\}_{i=1}^{D_{\text{in}}} \overset{i.i.d}{\sim} \text{Uniform}[0, 2\pi]$.

3: $\hat{\phi}(\mathsf{r}) = \sqrt{\frac{2}{D_{\text{in}}}}\left(\mathbb{E}_{x\sim\mathsf{r}} \cos(\omega_i^\top x + b_i)\right)_{i=1}^{D_{\text{in}}} \in \mathbb{R}^{D_{\text{in}}}$
If $\mathsf{r}(x) = \mathcal{N}(x; m, \Sigma)$,

$$\hat{\phi}(\mathsf{r}) = \sqrt{\frac{2}{D_{\text{in}}}}\left(\cos(\omega_i^\top m + b_i)\exp\left(-\frac{1}{2}\omega_i^\top \Sigma \omega_i\right)\right)_{i=1}^{D_{\text{in}}}.$$

4: Sample $\{\nu_i\}_{i=1}^{D_{\text{out}}} \overset{i.i.d}{\sim} \hat{k}_{\text{gauss}}(\gamma^2)$ i.e., Fourier transform of a Gaussian kernel with width γ^2.

5: Sample $\{c_i\}_{i=1}^{D_{\text{out}}} \overset{i.i.d}{\sim} \text{Uniform}[0, 2\pi]$.

6: $\hat{\psi}(\mathsf{r}) = \sqrt{\frac{2}{D_{\text{out}}}}\left(\cos(\nu_i^\top \hat{\phi}(\mathsf{r}) + c_i)\right)_{i=1}^{D_{\text{out}}} \in \mathbb{R}^{D_{\text{out}}}$

For the implementation, we need to pre-compute $\{\omega_i\}_{i=1}^{D_{\text{in}}}, \{b_i\}_{i=1}^{D_{\text{in}}}, \{\nu_i\}_{i=1}^{D_{\text{out}}}$ and $\{c_i\}_{i=1}^{D_{\text{out}}}$, where D_{in} and D_{out} are the number of random features used. A more efficient way to support a large number of random features is to store only the random seed used to generate the features, and to generate the coefficients on-the-fly as needed (Dai et al., 2014). In our implementation, we use a Gaussian kernel for k.

3.3 REGRESSION FOR OPERATOR PREDICTION

Let $\mathsf{X} = (\mathsf{x}_1|\cdots|\mathsf{x}_N)$ be the N training samples of incoming messages to a factor node, and let $\mathsf{Y} = \left(\mathbb{E}_{x_V\sim q^1_{f\to V}} u(x_V)|\cdots|\mathbb{E}_{x_V\sim q^N_{f\to V}} u(x_V)\right) \in \mathbb{R}^{D_y \times N}$ be the expected sufficient statistics of the corresponding output messages, where $q^i_{f\to V}$ is the numerator of (1). We write $\mathsf{x}_i = \hat{\psi}(\mathsf{r}_i)$ as a more compact notation for the random feature vector representing the i^{th} training tuple of incoming messages, as computed via Algorithm 1.

Since we require uncertainty estimates on our predictions, we perform Bayesian linear regression from the random features to the output messages, which yields predictions close to those obtained by Gaussian process regression with the kernel in (7). The uncertainty estimate in this case will

be the predictive variance. We assume prior and likelihood

$$w \sim \mathcal{N}\left(w; 0, I_{D_{\text{out}}}\sigma_0^2\right), \tag{9}$$

$$\mathsf{Y} \mid \mathsf{X}, w \sim \mathcal{N}\left(\mathsf{Y}; w^\top \mathsf{X}, \sigma_y^2 I_N\right), \tag{10}$$

where the output noise variance σ_y^2 captures the intrinsic stochasticity of the importance sampler used to generate Y. It follows that the posterior of w is given by (Bishop, 2006)

$$p(w|\mathsf{Y}) = \mathcal{N}(w; \mu_w, \Sigma_w), \tag{11}$$

$$\Sigma_w = \left(\mathsf{X}\mathsf{X}^\top \sigma_y^{-2} + \sigma_0^{-2} I\right)^{-1}, \tag{12}$$

$$\mu_w = \Sigma_w \mathsf{X}\mathsf{Y}^\top \sigma_y^{-2}. \tag{13}$$

The predictive distribution on the output y^* given an observation x^* is

$$p(\mathsf{y}^*|\mathsf{x}^*, \mathsf{Y}) = \int p(\mathsf{y}^*|w, \mathsf{x}^*, \mathsf{Y}) p(w|\mathsf{Y})\, dw \tag{14}$$

$$= \mathcal{N}\left(\mathsf{y}^*; \mathsf{x}^{*\top}\mu_w, \mathsf{x}^{*\top}\Sigma_w \mathsf{x}^* + \sigma_y^2\right). \tag{15}$$

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).

Online Update We describe an online update for Σ_w and μ_w when observations (i.e., random features representing incoming messages) x_i arrive sequentially. We use $\cdot^{(N)}$ to denote a quantity constructed from N samples. Recall that $\Sigma_w^{-1(N)} = \mathsf{X}\mathsf{X}^\top \sigma_y^{-2} + \sigma_0^{-2} I$. The posterior covariance matrix at time $N+1$ is

$$\Sigma_w^{(N+1)} = \Sigma_w^{(N)} - \frac{\Sigma_w^{(N)} \mathsf{x}_{N+1} \mathsf{x}_{N+1}^\top \Sigma_w^{(N)} \sigma_y^{-2}}{1 + \mathsf{x}_{N+1}^\top \Sigma_w^{(N)} \mathsf{x}_{N+1} \sigma_y^{-2}}, \tag{16}$$

meaning it can be expressed as an inexpensive update of the covariance at time N. Updating Σ_w for all the D_y outputs costs $O((D_{\text{in}} D_{\text{out}} + D_{\text{out}}^2) D_y)$ per new observation. For $\mu_w = \Sigma_w \mathsf{X}\mathsf{Y}^\top \sigma_y^{-2}$, we maintain $\mathsf{X}\mathsf{Y}^\top \in \mathbb{R}^{D_{\text{out}} \times D_{\text{y}}}$, and update it at cost $O(D_{\text{in}} D_{\text{out}} D_y)$ as

$$\left(\mathsf{X}\mathsf{Y}^\top\right)^{(N+1)} = \left(\mathsf{X}\mathsf{Y}^\top + \mathsf{x}_{N+1}\mathsf{y}_{N+1}^\top\right). \tag{17}$$

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 prespecified predictive variance thresholds. Given a new input x^*, if $v_1^* > \tau_1$ or $\cdots$ or $v_{D_y}^* > \tau_{D_y}$ (the operator is uncertain), a query is made to the oracle to obtain a ground truth y^*. The pair $(\mathsf{x}^*, \mathsf{y}^*)$ is then used to update Σ_w and μ_w.

4 EXPERIMENTS

We evaluate our learned message operator using two different factors: the logistic factor, and the compound gamma factor. In the first and second experiment we demonstrate that the proposed operator is capable of learning high-quality mappings from incoming to outgoing messages, and that the associated uncertainty estimates are reliable. The third and fourth experiments assess the performance of the operator as part of the full EP inference loop in two different models: approximating the logistic, and the compound gamma factor. Our final experiment demonstrates the ability of our learning process to reliably and quickly adapt to large shifts in the message distribution, as encountered during inference in a sequence of several real-world regression problems.

For all experiments we used Infer.NET (Minka et al., 2014) with its extensible factor interface for our own operator. We used the default settings of Infer.NET unless stated otherwise. The regression target is the marginal belief (numerator of (1)) in experiment 1,2,3 and 5. We set the regression target to the outgoing message in experiment 4. Given a marginal belief, the outgoing message can be calculated straightforwardly.

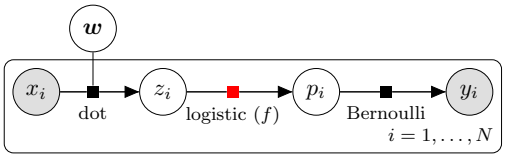
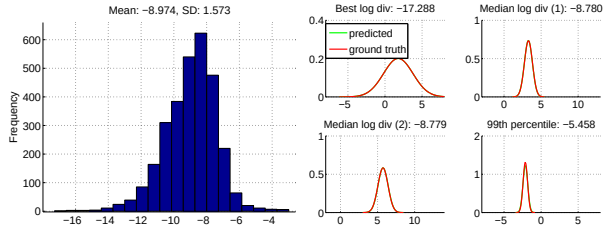


Figure 2: Factor graph for binary logistic regression. The kernel-based message operator learns to approximate the logistic factor highlighted in red. The two incoming messages are $m_{z_i \to f} = \mathcal{N}(z_i; \mu, \sigma^2)$ and $m_{p_i \to f} = \text{Beta}(p_i; \alpha, \beta)$.

Experiment 1: Batch Learning As in (Heess et al., 2013; Eslami et al., 2014), we study the logistic factor $f(p|z) = \delta\left(p - \frac{1}{1+\exp(-z)}\right)$, where δ is the Dirac delta function, in the context of a binary logistic regression model (Fig. 2). The factor is deterministic and there are two incoming messages: $m_{p_i \to f} = \text{Beta}(p_i; \alpha, \beta)$ and $m_{z_i \to f} = \mathcal{N}(z_i; \mu, \sigma^2)$, where $z_i = \boldsymbol{w}^\top x_i$ represents the dot product between an observation $x_i \in \mathbb{R}^d$ and the coefficient vector $\boldsymbol{w}$ whose posterior is to be inferred.

In this first experiment we simply learn a kernel-based operator to send the message $m_{f \to z_i}$. Following Eslami et al. (2014), we set d to 20, and generated 20 different datasets, sampling a different $\boldsymbol{w} \sim \mathcal{N}(0, I)$ and then a set of $\{(x_i, y_i)\}_{i=1}^n$ ($n = 300$) observations according to the model. For each dataset we ran EP for 10 iterations, and collected incoming-outgoing message pairs in the first five iterations of EP from Infer.NET's implementation of the logistic factor. We partitioned the messages randomly into 5,000 training and 3,000 test messages, and learned a message operator to predict $m_{f \to z_i}$ as described in Section 3. Regularization and kernel parameters were chosen

by leave-one-out cross validation. We set the number of random features to $D_{in} = 500$ and $D_{out} = 1,000$; empirically, we observed no significant improvements beyond 1,000 random features.



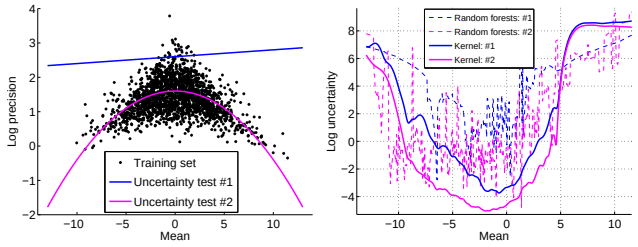
(a) KL errors (b) Examples of predictions

Figure 3: Prediction errors for predicting the projected beliefs to z_i, and examples of predicted messages at different error levels.

We report $\log \mathrm{KL}[q_{f\to z_i} \| \hat{q}_{f\to z_i}]$ where $q_{f\to z_i}$ is the ground truth projected belief (numerator of (1)) and $\hat{q}_{f\to z_i}$ is the prediction. The histogram of the log KL errors is shown in Fig. 3a; Fig. 3b shows examples of predicted messages for different log KL errors. It is evident that the kernel-based operator does well in capturing the relationship between incoming and outgoing messages. The discrepancy with respect to the ground truth is barely visible even at the 99th percentile. See Section C in the supplementary material for a comparison with other methods.

Experiment 2: Uncertainty Estimates For the approximate message operator to perform well in a JIT learning setting, it is crucial to have reliable estimates of operator's predictive uncertainty in different parts of the space of incoming messages. To assess this property we compute the predictive variance using the same learned operator as used in Fig. 3. The forward incoming messages $m_{z_i\to f}$ in the previously used training set are shown in Fig. 4a. The backward incoming messages $m_{p_i\to f}$ are not displayed. Shown in the same plot are two curves (a blue line, and a pink parabola) representing two "uncertainty test sets": these are the sets of parameter pairs on which we wish to evaluate the uncertainty of the predictor, and pass through regions with both high and low densities of training samples. Fig. 4b shows uncertainty estimates of our kernel-based operator and of random forests, where we fix $m_{p_i\to f} := \mathrm{Beta}(p_i; 1, 2)$ for testing. The implementation of the random forests closely follows Eslami et al. (2014).

From the figure, as the mean of the test message moves away from the region densely sampled by the training data, the predictive variance reported by the kernel method increases much more smoothly than that of the random forests. Further, our method clearly exhibits a higher uncertainty on the test set #1 than on the test set #2. This



(a) Parameters of $m_{z_i\to f}$ (b) Uncertainty estimates

Figure 4: (a) Incoming messages from z to f from 20 EP runs of binary logistic regression, as shown in Fig. 2. (b) Uncertainty estimates of the proposed kernel-based method (predictive variance) and Eslami et al.'s random forests (KL-based agreement of predictions of different trees) on the two uncertainty test sets shown. For testing, we fix the other incoming message $m_{p_i\to f}$ to $\mathrm{Beta}(p_i; 1, 2)$.

behaviour is desirable, as most of the points in test set #1 are either in a low density region or an unexplored region. These results suggest that the predictive variance is a robust criterion for querying the importance sampling oracle. One key observation is that the uncertainty estimates of the random forests are highly non-smooth; i.e., uncertainty on nearby points may vary wildly. As a result, a random forest-based JIT learner may still query the importance sampler oracle when presented with incoming messages similar to those in the training set, thereby wasting computation.

We have further checked that the predictive uncertainty of the regression function is a reliable indication of the error in KL divergence of the predicted outgoing messages. These results are given in Figure 10 of Appendix C.

Experiment 3: Just-In-Time Learning In this experiment we test the approximate operator in the logistic regression model as part of the full EP inference loop in a just-in-time learning setting (KJIT). We now learn two kernel-based message operators, one for each outgoing direction from the logistic factor. The data generation is the same as in the batch learning experiment. We sequentially presented the operator with 30 related problems, where a new set of observations $\{(x_i, y_i)\}_{i=1}^n$ was generated at the beginning of each problem from the model, while keeping $\boldsymbol{w}$ fixed. This scenario is common in practice: one is often given several sets of observations which share the same model parameter (Eslami et al., 2014). As before, the inference target was $p(\boldsymbol{w}|\{(x_i, y_i)\}_{i=1}^n)$. We set the maximum number of EP iterations to 10 in each problem.

We employed a "mini-batch" learning approach in which the operator always consults the oracle in the first few hundred factor invocations for initial batch training. In principle, during the initial batch training, the operator can perform cross validation or type-II maximum likelihood esti-

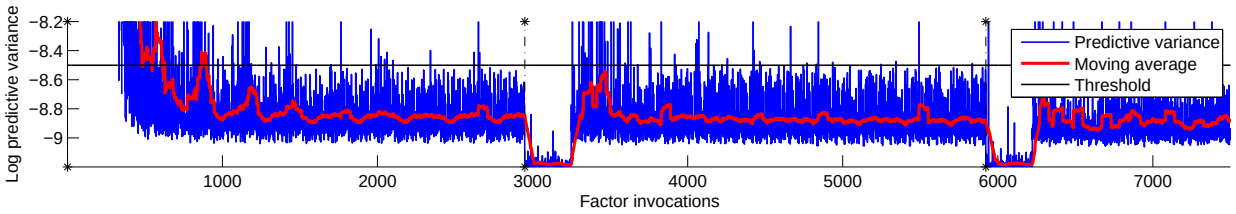
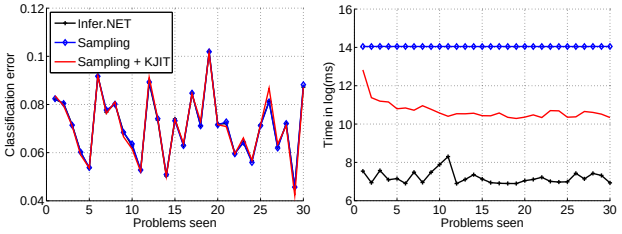


Figure 5: Uncertainty estimate of KJIT in its prediction of outgoing messages at each factor invocation, for the binary logistic regression problem. The black dashed lines indicate the start of a new inference problem.

mation for parameter selection; however for computational simplicity we set the kernel parameters according to the median heuristic (Schölkopf and Smola, 2002). Full detail of the heuristic is given in Section A in the supplementary material. The numbers of random features were $D_{\text{in}} = 300$ and $D_{\text{out}} = 500$. The output noise variance σ_y^2 was fixed to 10^{-4} and the uncertainty threshold on the log predictive variance was set to -8.5. To simulate a black-box setup, we used an importance sampler as the oracle rather than Infer.NET's factor implementation, where the proposal distribution was fixed to $\mathcal{N}(z; 0, 200)$ with 5×10^5 particles.

Fig. 5 shows a trace of the predictive variance of KJIT in predicting the mean of each $m_{f \to z_i}$ upon each factor invocation. The black dashed lines indicate the start of a new inference problem. Since the first 300 factor invocations are for the initial training, no uncertainty estimate is shown. From the trace, we observe that the uncertainty rapidly drops down to a stable point at roughly -8.8 and levels off after the operator sees about 1,000 incoming-outgoing message pairs, which is relatively low compared to approximately 3,000 message passings (i.e., 10 iterations $\times$ 300 observations) required for one problem. The uncertainty trace displays a periodic structure, repeating itself in every 300 factor invocations, corresponding to a full sweep over all 300 observations to collect incoming messages $m_{z_i \to f}$. The abrupt drop in uncertainty in the first EP iteration of each new problem is due to the fact that Infer.NET's inference engine initializes the message from $\boldsymbol{w}$ to have zero mean, leading to $m_{z_i \to f}$ also having a zero mean. Repeated encounters of such a zero mean incoming message reinforce the operator's confidence; hence the drop in uncertainty.

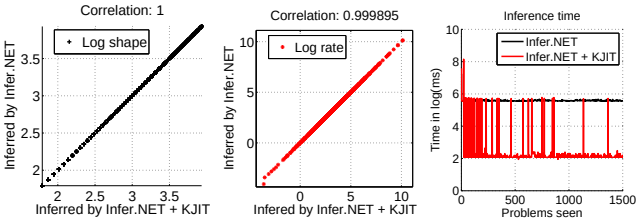
Fig. 6a shows binary classification errors obtained by using the inferred posterior mean of $\boldsymbol{w}$ on a test set of size 10000 generated from the true underlying parameter. Included in the plot are the errors obtained by using only the importance sampler for inference ("Sampling"), and using the Infer.NET's hand-crafted logistic factor. The loss of KJIT matches well with that of the importance sampler and Infer.NET, suggesting that the inference accuracy is as good as these alternatives. Fig. 6b shows the inference time required by all methods in each problem. While the inference quality is equally good, KJIT is orders of magnitude faster



(a) Binary classification error (b) Inference time

Figure 6: Classification performance and inference times of all methods in the binary logistic regression problem.

than the importance sampler.



(a) Inferred shape (b) Inferred rate (c) Inference time

Figure 7: Shape (a) and rate (b) parameters of the inferred posteriors in the compound gamma problem. (c) KJIT is able to infer equally good posterior parameters compared to Infer.NET, while requiring a runtime several orders of magnitude lower.

Experiment 4: Compound Gamma Factor We next simulate the compound gamma factor, a heavy-tailed prior distribution on the precision of a Gaussian random variable. A variable τ is said to follow the compound gamma distribution if $\tau \sim \text{Gamma}(\tau; s_2, r_2)$ (shape-rate parameterization) and $r_2 \sim \text{Gamma}(r_2; s_1, r_1)$ where (s_1, r_1, s_2) are parameters. The task we consider is to infer the posterior of the precision τ of a normally distributed variable $x \sim \mathcal{N}(x; 0, \tau)$ given realizations $\{x_i\}_{i=1}^n$. We consider the setting $(s_1, r_1, s_2) = (1, 1, 1)$ which was used in Heess et al. (2013). Infer.NET's implementation requires two gamma factors to specify the compound gamma. Here, we collapse them into one factor and let the operator learn to

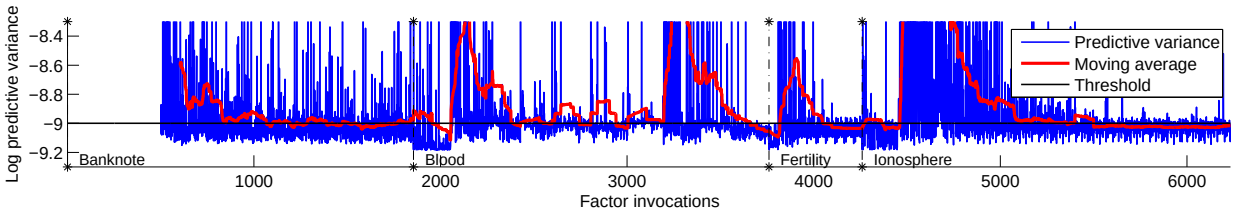


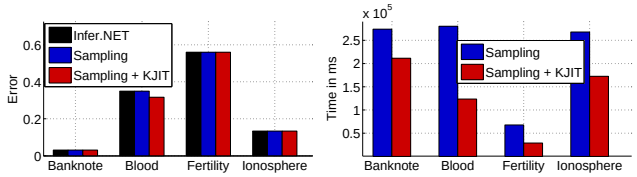
Figure 8: Uncertainty estimate of KJIT for outgoing messages on the four UCI datasets.

directly send an outgoing message $m_{f\to\tau}$ given $m_{\tau\to f}$, using Infer.NET as the oracle. The default implementation of Infer.NET relies on a quadrature method. As in Eslami et al. (2014), we sequentially presented a number of problems to our algorithm, where at the beginning of each problem, a random number of observations n from 10 to 100, and the parameter τ, were drawn from the model.

Fig. 7a and Fig. 7b summarize the inferred posterior parameters obtained from running only Infer.NET and Infer.NET + KJIT, i.e., KJIT with Infer.NET as the oracle. Fig. 7c shows the inference time of both methods. The plots collectively show that KJIT can deliver posteriors in good agreement with those obtained from Infer.NET, at a much lower cost. Note that in this task only one message is passed to the factor in each problem. Fig. 7c also indicates that KJIT requires fewer oracle consultations as more problems are seen.

Experiment 5: Classification Benchmarks In the final experiment, we demonstrate that our method for learning the message operator is able to detect changes in the distribution of incoming messages via its uncertainty estimate, and to subsequently update its prediction through additional oracle queries. The different distributions of incoming messages are achieved by presenting a sequence of different classification problems to our learner. We used four binary classification datasets from the UCI repository (Lichman, 2013): banknote authentication, blood transfusion, fertility and ionosphere, in the same binary logistic regression setting as before. The operator was required to learn just-in-time to send outgoing messages $m_{f\to z_i}$ and $m_{f\to p_i}$ on the four problems presented in sequence. The training observations consisted of 200 data points subsampled from each dataset by stratified sampling. For the fertility dataset, which contains only 100 data points, we subsampled half the points. The remaining data were used as test sets. The uncertainty threshold was set to -9, and the minibatch size was 500. All other parameters were the same as in the earlier JIT learning experiment.

Classification errors on the test sets and inference times are shown in Fig. 9a and Fig. 9b, respectively. The results demonstrate that KJIT improves the inference time on all the problems without sacrificing inference accuracy. The predictive variance of each outgoing message is shown in Fig. 8. An essential feature to notice is the rapid increase of the uncertainty after the first EP iteration of each problem. As shown in Fig. 1, the distributions of incoming messages of the four problems are diverse. The sharp rise followed by a steady decrease of the uncertainty is a good indicator that the operator is able to promptly detect a change in input message distribution, and robustly adapt to this new distribution by querying the oracle.



(a) Binary classification error (b) Inference time

Figure 9: Classification performance and inference times on the four UCI datasets.

5 CONCLUSIONS AND FUTURE WORK

We have proposed a method for learning the mapping between incoming and outgoing messages to a factor in expectation propagation, which can be used in place of computationally demanding Monte Carlo estimates of these updates. Our operator has two main advantages: it can reliably evaluate the uncertainty of its prediction, so that it only consults a more expensive oracle when it is uncertain, and it can efficiently update its mapping online, so that it learns from these additional consultations. Once trained, the learned mapping performs as well as the oracle mapping, but at a far lower computational cost. This is in large part due to a novel two-stage random feature representation of the input messages. One topic of current research is hyperparameter selection: at present, these are learned on an initial mini-batch of data, however a better option would be to adapt them online as more data are seen.

ACKNOWLEDGEMENT

We thank the anonymous reviewers for their constructive comments. WJ, AG, BL, and ZSz thank the Gatsby Charitable Foundation for the financial support.

References

M. A. Alvarez, L. Rosasco, and N. D. Lawrence. Kernels for vector-valued functions: a review. 2011. URL http://arxiv.org/abs/1106.6251.

S. Barthelmé and N. Chopin. ABC-EP: Expectation propagation for likelihood-free Bayesian computation. In *ICML*, pages 289–296, 2011.

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

A. Christmann and I. Steinwart. Universal kernels on non-standard input spaces. In *NIPS*, pages 406–414, 2010.

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 *NIPS*, pages 3041–3049, 2014.

S. M. A. Eslami, D. Tarlow, P. Kohli, and J. Winn. Just-In-Time Learning for Fast and Flexible Inference. In *NIPS*, pages 154–162, 2014.

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

N. Goodman, V. Mansinghka, D. Roy, K. Bonawitz, and J. Tenenbaum. Church: A language for generative models. In *UAI*, pages 220–229, 2008.

N. Heess, D. Tarlow, and J. Winn. Learning to pass expectation propagation messages. In *NIPS*, 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. http://www.cs.ubc.ca/~hutter/papers/Hutter09PhD.pdf.

B. Lakshminarayanan, D. Roy, and Y.-W. Teh. Mondrian forests: Efficient online random forests. In *NIPS*, pages 3140–3148, 2014.

Q. Le, T. Sarlós, and A. Smola. Fastfood - approximating kernel expansions in loglinear time. *ICML, JMLR W&CP*, 28:244–252, 2013.

M. Lichman. UCI machine learning repository, 2013. URL http://archive.ics.uci.edu/ml.

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. http://research.microsoft.com/en-us/um/people/minka/papers/ep/minka-thesis.pdf.

A. Rahimi and B. Recht. Random features for large-scale kernel machines. In *NIPS*, pages 1177–1184, 2007.

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

W. Rudin. *Fourier Analysis on Groups: Interscience Tracts in Pure and Applied Mathematics, No. 12*. Literary Licensing, LLC, 2013.

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 *ALT*, pages 13–31, 2007.

L. Song, A. Gretton, and C. Guestrin. Nonparametric tree graphical models. *AISTATS, JMLR W&CP*, 9:765–772, 2010.

L. Song, A. Gretton, D. Bickson, Y. Low, and C. Guestrin. Kernel belief propagation. *AISTATS, JMLR W&CP*, 10: 707–715, 2011.

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

Z. Szabó, B. Sriperumbudur, B. Póczos, and A. Gretton. Learning theory for distribution regression. Technical report, Gatsby Unit, University College London, 2014. (http://arxiv.org/abs/1411.2066).

D. Wingate, N. Goodman, A. Stuhlmueller, and J. Siskind. Nonstandard interpretations of probabilistic programs for efficient inference. In *NIPS*, pages 1152–1160, 2011.

Z. Yang, A. J. Smola, L. Song, and A. G. Wilson. Á la carte - learning fast kernels. In *AISTATS*, 2015. http://arxiv.org/abs/1412.6493.