Just-In-Time Kernel Regression for Expectation Propagation

Wittawat Jitkrittum[1] WITTAWATJ@GMAIL.COM
Arthur Gretton[1] ARTHUR.GRETTON@GMAIL.COM
Nicolas Heess[*] NHEESS@GMAIL.COM
S. M. Ali Eslami[*] ALI@ARKITUS.COM
Balaji Lakshminarayanan[1] BALAJI@GATSBY.UCL.AC.UK
Dino Sejdinovic[2] DINO.SEJDINOVIC@GMAIL.COM
Zoltán Szabó[1] ZOLTAN.SZABO@GATSBY.UCL.AC.UK
[1]Gatsby Unit, University College London
[2]University of Oxford

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 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. 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.

In this work, we address the particular setting of expectation propagation (EP; Minka, 2001), a message passing algorithm wherein messages are confined to being members of a particular parametric family. Sending EP outgoing messages requires an expensive computation involving integrating incoming messages over a factor potential, and projecting the result onto the chosen family. We propose a novel, kernel-based approach to learning a message operator nonparametrically for EP. The learning algorithm takes the form of a distribution regression problem, where the inputs are probability measures (incoming messages), and the outputs are vectors of message parameters (Szabó et al., 2014).

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 to determine when to query the oracle for additional input/output pairs. 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.

[*] Currently at Google DeepMind.

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 $f : \mathbf{x}_{\mathrm{in}} \mapsto \mathbf{x}_{\mathrm{out}}$. In particular, the ability to evaluate the value of $f(\mathbf{x}_{\mathrm{out}}|\mathbf{x}_{\mathrm{in}})$ is not assumed.

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). The message $m_{f\to V}(x_V)$ from factor f to variable $V \in \mathrm{ne}(f)$ is

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

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

Computing the numerator of (1) can be challenging and often requires hand-crafted approximations, or the use of expensive numerical integration techniques; for "black-box" factors f 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 approach to the integration and projection step based on the following well known result. When the projection proj 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). The estimated expected sufficient statistics provide us with an estimate of the parameters η of the result q of the projection $\mathrm{proj}\,[p]$, from which the message is readily computed.

Learning to pass EP messages One approach to compute the expected sufficient statistics, due to Barthelmé and Chopin (2011), is 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 of the sampler. Heess et al. (2013) use a neural network to learn the mapping from incoming to outgoing messages. Specifically, they trained a neural network to directly map $(m_{V'\to f})_{V'\in\mathrm{ne}(f)}$ to $m_{f\to V}$, i.e. they learn a mapping $M^{\theta}_{f\to V} : (m_{V'\to f})_{V'\in\mathrm{ne}(f)} \mapsto m_{f\to V}$, where θ are the parameters of the approximator.

Although 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 provided uncertainty estimates for each prediction. This allowed them to be trained 'just-in-time', during EP inference, whenever the predictor decides it is uncertain. That is, if the uncertainty on the current input messages exceeds a pre-defined threshold, the required outgoing message is approximated via importance sampling and $M^{\theta}_{f\to V}$ is updated on this new data point (leading to a new set of parameters θ'). However, uncertainty estimation for random forests relies on unproven heuristics: such heuristics can become highly misleading as we move away from the training data.

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. We regress from the tuple of incoming messages, which are probability distributions, to the parameters of the outgoing message.

3.1. Kernels on tuples of distributions

In contrast to Eslami et al. (2014); Heess et al. (2013) our kernel-based approach does not need a pre-specified customized features to represent incoming messages. Rather, we use a general characteristic kernel operated directly on distributions (Christmann and Steinwart, 2010, eq. 9). The kernel is independent of the parameterization of incoming messages.

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 $D_{\text{in}}$, number of outer features $D_{\text{out}}$, outer Gaussian width $\gamma^2$.
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 $\{b_i\}_{i=1}^{D_{\text{in}}} \overset{i.i.d}{\sim}$ 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 $\gamma^2$.
5: Sample $\{c_i\}_{i=1}^{D_{\text{out}}} \overset{i.i.d}{\sim}$ Uniform$[0, 2\pi]$.
6: $\hat{\psi}(\mathsf{r}) = \sqrt{\frac{2}{D_{\text{out}}}} \left(\cos(\nu_i^\top \hat{\phi}(\mathsf{r}) + c_i)\right)_{i=1}^{D_{\text{out}}} \in \mathbb{R}^{D_{\text{out}}}$
```

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 $\mu_{r^{(l)}}(\cdot) := \int k^{(l)}(x^{(l)}, \cdot)\,\mathrm{d}r^{(l)}(x^{(l)})$. Similarly, a mean embedding may be defined on the product of messages in a tuple $\mathsf{r} = \times_{l=1}^c r^{(l)}$ as $\mu_{\mathsf{r}} := \int k([x^{(1)}, \ldots, x^{(c)}], \cdot)\,\mathrm{d}r(x^{(1)}, \ldots, x^{(c)})$, 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),

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

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

3.2. Random feature approximations

One approach to learning the mapping $M_{f\to V}^\theta$ from incoming to outgoing messages would be to employ Gaussian process regression, using the kernel (2). 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. Instead, we define a finite-dimensional random feature map $\hat{\psi}(\mathsf{r}), \hat{\psi}(\mathsf{s}) \in \mathbb{R}^{D_{\text{out}}}$ 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: 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 $\hat{\varphi}(x), \hat{\varphi}(y) \in \mathbb{R}^{D_{\text{in}}}$. We will follow a similar approach, and derive a two-stage set of random Fourier features for (2).

We start by expanding the exponent of (2) 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 $\hat{\phi}(\mathsf{r}), \hat{\phi}(\mathsf{s})$ 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 $\kappa(\mathsf{r}, \mathsf{s}) \approx \exp\left(-\frac{\|\hat{\phi}(\mathsf{r}) - \hat{\phi}(\mathsf{s})\|_{D_{\text{in}}}^2}{2\gamma^2}\right)$, 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 $\hat{\psi}(\mathsf{r}), \hat{\psi}(\mathsf{s}) \in \mathbb{R}^{D_{\text{out}}}$ such that $\kappa(\mathsf{r}, \mathsf{s}) \approx \hat{\psi}(\mathsf{r})^\top\hat{\psi}(\mathsf{s})$. Pseudocode for generating the random features $\hat{\psi}$ is given in Algorithm 1. For the implementation, $\{\omega_i\}_{i=1}^{D_{\text{in}}}, \{b_i\}_{i=1}^{D_{\text{in}}}, \{\nu_i\}_{i=1}^{D_{\text{out}}}$ and $\{c_i\}_{i=1}^{D_{\text{out}}}$ need to be sampled only once, where D_{in} and D_{out} are the number of random features used. In the experiments, 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 $\mathsf{Y} = \left(\mathbb{E}_{x_V\sim q_{f\to V}^1} u(x_V)|\cdots|\mathbb{E}_{x_V\sim q_{f\to V}^N} u(x_V)\right) \in \mathbb{R}^{D_y\times N}$ be the expected sufficient statistics of the corresponding output messages, where $q_{f\to V}^i$ 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 (2). The uncertainty estimate in this case will be the predictive variance. We assume priors

$$w \sim \mathcal{N}\left(w; 0, I_{D_{\text{out}}}\sigma_0^2\right), \quad (3)$$

$$\mathsf{Y} \mid \mathsf{X}, w \sim \mathcal{N}\left(\mathsf{Y}; w^\top \mathsf{X}, \sigma_y^2 I_N\right), \quad (4)$$

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)

$$p(w|\mathsf{Y}) = \mathcal{N}(w; \mu_w, \Sigma_w), \quad (5)$$

$$\Sigma_w = \left(\mathsf{X}\mathsf{X}^\top \sigma_y^{-2} + \sigma_0^{-2} I\right)^{-1}, \quad (6)$$

$$\mu_w = \Sigma_w \mathsf{X}\mathsf{Y}^\top \sigma_y^{-2}. \quad (7)$$

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

$$p(\mathsf{y}^*|\mathsf{x}^*, \mathsf{Y}) = \mathcal{N}\left(\mathsf{y}^*; \mathsf{x}^{*\top}\mu_w, \mathsf{x}^{*\top}\Sigma_w \mathsf{x}^* + \sigma_y^2\right). \quad (8)$$

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 Let $\cdot^{(N)}$ denote a quantity constructed from N samples. The posterior covariance matrix at time $N+1$ can be written as

$$\Sigma_w^{(N+1)} = \Sigma_w^{(N)} - \frac{\Sigma_w^{(N)} \mathsf{x}_{N+1} \mathsf{x}_{N+1}^\top \Sigma_w^{(N)} \sigma_y^{-2}}{1 + \mathsf{x}_{N+1}^\top \Sigma_w^{(N)} \mathsf{x}_{N+1} \sigma_y^{-2}}, \quad (9)$$

meaning it can be expressed as an inexpensive update of the covariance at time N. Updating μ_w can be easily achieved by maintaining $\mathsf{X}\mathsf{Y}^\top$.

4. Experiments

We evaluate our learned message operator using two different factors: the logistic factor, and the compound gamma factor. 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. Code is available at `https://github.com/wittawatj/kernel-ep`.

Experiment 1: Logistic factor 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. 1). 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.

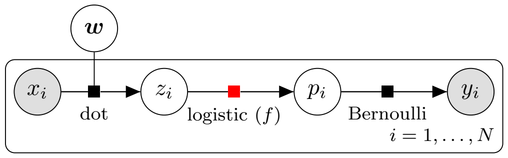


Figure 1: Factor graph for binary logistic regression. The kernel-based message operator learns to approximate the logistic factor highlighted in red.

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 used four binary classification datasets from the UCI repository (Lichman, 2013): banknote authentication, blood transfusion, fertility and ionosphere, in binary logistic regression setting. 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.

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 optimize the marginal likelihood for parameter selection; however for computational simplicity we set the kernel parameters according to the median heuristic (Schölkopf and Smola, 2002). The numbers of random features were $D_{\text{in}} = 300$ and $D_{\text{out}} = 500$; empirically, we observed no significant improvements beyond 1,000 random features. The output noise variance σ_y^2 was fixed to 10^{-4} and the uncertainty threshold on the log predictive variance (over which the oracle is queried) was set to -9. 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. We set the maximum number of EP iterations to 10 in each problem.

Classification errors on the test sets and inference times are shown in Fig. 3a and Fig. 3b, 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

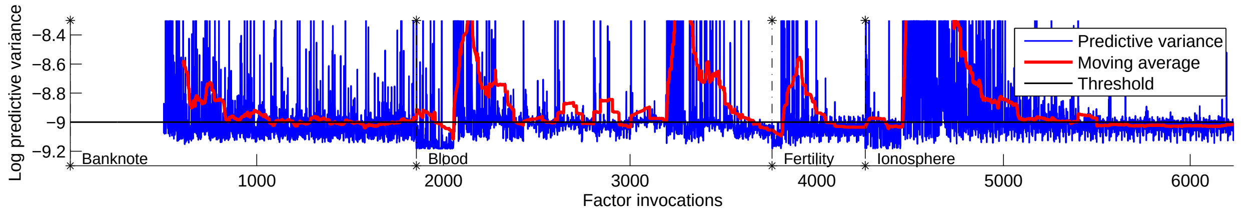
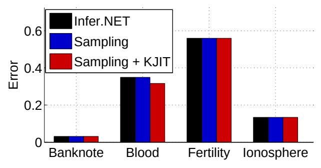
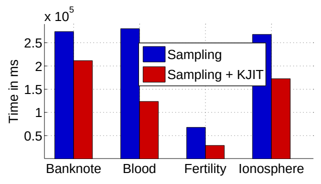


Figure 2: Uncertainty estimate of KJIT for outgoing messages on the four UCI datasets.

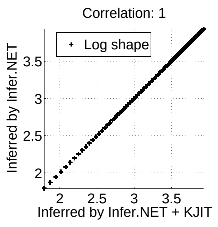


(a) Binary classification error

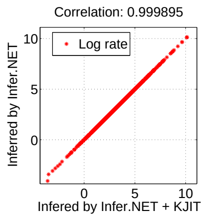


(b) Inference time

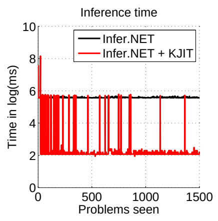
Figure 3: Classification performance and inference times on the four UCI datasets.



(a) Inferred shape



(b) Inferred rate



(c) Inference time

Figure 4: 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.

Fig. 2. An essential feature to notice is the rapid increase of the uncertainty after the first EP iteration of each problem. 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.

Experiment 2: 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 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 from 10 to 100, and the parameter τ, were drawn from the model.

Fig. 4a and Fig. 4b 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. 4c 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. 4c also indicates that KJIT requires fewer oracle consultations as more problems are seen.

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

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.

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.

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.

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.

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.

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`.