# *revrand*: Technical Report

**Daniel Steinberg** daniel.steinberg@data61.csiro.au
**Louis Tiao** louis.tiao@data61.csiro.au
**Lachlan McCalman** lachlan.mccalman@data61.csiro.au
**Alistair Reid** alistair.reid@data61.csiro.au
**Simon O'Callaghan** simon.ocallaghan@data61.csiro.au
*DATA61, CSIRO*
*Sydney, Australia*

## Abstract

This is a technical report on the *revrand* software library. This library implements Bayesian linear models (Bayesian linear regression), generalized linear models and approximate Gaussian processes. These algorithms have been implemented such that they can be used for large-scale learning by using stochastic variational inference. All of the algorithms in *revrand* use a unified feature composition framework that allows for easy concatenation and selective application of regression basis functions.

## Contents



## 1. Core Algorithms

Recent developments in stochastic gradient optimisation have simplified the application of machine learning to massive datasets, in particular, modern stochastic optimisation algorithms are far more robust to initial learning rate settings (see Kingma and Ba (2014) for instance). Furthermore, Bayesian machine learning algorithms have a number of well defined methods for learning model parameters and *hyperparameters* from training data, that do not involve cross validation. When used in combination, stochastically optimized Bayesian machine learning algorithms allow practitioners to learn probabilistic predictors from large data sets with minimal tuning and retraining.

We make use of some of these recent developments in stochastic gradient methods and stochastic variational inference in *revrand* for supervised regression tasks. We outline the core algorithms implemented in *revrand* in this section of the report, beginning with how

we use stochastic optimization with variational objectives, then onto the two core regression algorithms in *revrand*, and how to use them to appropriate Gaussian Processes (GPs).

### 1.1 Stochastic Gradients and Variational Objective Functions

When a machine learning objective function factorises over data,

$$f(\mathbf{X}, \theta) = \sum_{\mathbf{x} \in \mathbf{X}} f(\mathbf{x}, \theta)\,, \tag{1}$$

a regular gradient descent algorithm would perform the following iterations to minimise the function w.r.t. the parameters $\theta$,

$$\theta_k := \theta_{k-1} - \eta_k \sum_{\mathbf{x} \in \mathbf{X}} \nabla_\theta f(\mathbf{x}, \theta)|_{\theta=\theta_{k-1}}, \tag{2}$$

where $\eta_k$ is the learning rate (step size) at iteration $k$. Stochastic gradient algorithms use the following update,

$$\theta_k := \theta_{k-1} - \eta_k \sum_{\mathbf{x} \in \mathbf{B}} \nabla_\theta f(\mathbf{x}, \theta)|_{\theta=\theta_{k-1}}, \tag{3}$$

where $\mathbf{B} \subset \mathbf{X}$ is a mini-batch of the original dataset, where $|\mathbf{B}| \ll |\mathbf{X}|$. Frequently objective functions to not entirely decompose over the data, i.e.,

$$f(\mathbf{X}, \theta) = \sum_{\mathbf{x} \in \mathbf{X}} f(\mathbf{x}, \theta) + g(\theta)\,. \tag{4}$$

However, it is trivial to make these objectives work in a stochastic gradient setting. Let $M = |\mathbf{B}|$ and $N = |\mathbf{X}|$, then we can divide the contribution of the constant term amongst the mini-batches in stochastic gradients,

$$\theta_k := \theta_{k-1} - \eta_k \sum_{\mathbf{x} \in \mathbf{B}} \nabla_\theta f(\mathbf{x}, \theta)|_{\theta=\theta_{k-1}} - \frac{M}{N} \eta_k \nabla_\theta g(\theta)|_{\theta=\theta_{k-1}}. \tag{5}$$

or, equivalently, boost the contribution of the mini-batch,

$$\theta_k := \theta_{k-1} - \frac{N}{M} \eta_k \sum_{\mathbf{x} \in \mathbf{B}} \nabla_\theta f(\mathbf{x}, \theta)|_{\theta=\theta_{k-1}} - \eta_k \nabla_\theta g(\theta)|_{\theta=\theta_{k-1}}. \tag{6}$$

This is particularly relevant for variational inference where the evidence lower bound objective has a component independent of the data. For example, lets consider the model,

$$\text{Likelihood:} \quad \prod_{n=1}^{N} p(y_n|\theta)\,, \tag{7}$$

$$\text{prior:} \quad p(\theta|\alpha)\,, \tag{8}$$

where we want to learn the values of the hyper-parameters, $\alpha$. Minimising negative log-marginal likelihood is a reasonable objective in this instance[1], since we don't care about the value(s) of $\theta$,

$$\underset{\alpha}{\text{argmin}} - \log \int \prod_{n=1}^{N} p(y_n|\theta)\, p(\theta|\alpha)\, d\theta. \tag{9}$$

1. see Bishop (2006, Chapter 3.4) and Rasmussen and Williams (2006, Chapter 5) for a discussion on this.

There are two problems with this objective however, (1) it may not in general factor over data and (2) the integral may be intractable, for instance, if the prior and likelihood are not conjugate. In variational inference we use Jensen's inequality to lower-bound log-marginal likelihood with a tractable objective function called the evidence lower bound (ELBO),

$$\begin{aligned}\log p(\mathbf{y}|\alpha) &= \log \int \prod_{n=1}^{N} p(y_n|\theta)\, p(\theta|\alpha)\, d\theta \\ &= \log \int \frac{\prod_n p(y_n|\theta)\, p(\theta|\alpha)}{q(\theta)} q(\theta)\, d\theta \\ &\geq \int q(\theta) \log \left[ \frac{\prod_n p(y_n|\theta)\, p(\theta|\alpha)}{q(\theta)} \right] d\theta \end{aligned} \tag{10}$$

where $q(\theta)$ is an approximation of the true posterior $p(\theta|\{y_1, \ldots, y_N\}, \alpha)$, chosen to make inference easier. This can be re-written as,

$$\mathcal{L} = \sum_{n=1}^{N} \langle \log p(y_n|\theta) \rangle_q - \mathrm{KL}[q(\theta) \| p(\theta|\alpha)], \tag{11}$$

which takes the form of Equation (4), and so if we use stochastic gradients optimisation we can weight the Kullback-Leibler term like the constant term, $g(\cdot)$, from Equation (5), or boost the expected log likelihood term like in Equation (6).

### 1.2 Bayesian Linear Regression – `StandardLinearModel`

The first machine learning algorithm in *revrand* is a simple Bayesian linear regressor of the following form,

$$\text{Likelihood:} \quad \prod_{n=1}^{N} \mathcal{N}\left(y_n \middle| \boldsymbol{\phi}_n^\top \mathbf{w}, \sigma^2\right), \tag{12}$$

$$\text{prior:} \quad \mathcal{N}(\mathbf{w}|\mathbf{0}, \boldsymbol{\Lambda}), \tag{13}$$

where $\boldsymbol{\phi}_n := \phi(\mathbf{x}_n, \theta)$ is a feature, or basis, function that maps $\phi : \mathbb{R}^d \to \mathbb{R}^D$, and $\boldsymbol{\Lambda} \in \mathbb{R}^{D \times D}$ is a diagonal matrix (i.e. it could be $\lambda \mathbf{I}_D$) that has the effect of regularising the magnitude of the weights. This is the same algorithm described in Rasmussen and Williams (2006, Chapter 2). We then:

- Optimise $\sigma^2, \boldsymbol{\Lambda}$ and $\theta$ w.r.t. log-marginal likelihood,

$$\log p\left(\mathbf{y} \middle| \sigma^2, \boldsymbol{\Lambda}, \theta\right) = \log \mathcal{N}\left(\mathbf{y} \middle| \mathbf{0}, \sigma^2 \mathbf{I}_N + \boldsymbol{\Phi}^\top \boldsymbol{\Lambda} \boldsymbol{\Phi}\right), \tag{14}$$

  where $\boldsymbol{\Phi} \in \mathbb{R}^{N \times D}$ is the concatenation of all the features, $\boldsymbol{\phi}_n$. Note this results in the covariance of the log-marginal likelihood being $N \times N$, though we can use the Woodbury identity to simplify the corresponding matrix inversion.

- Solve analytically for the posterior over weights, $\mathbf{w}|\mathbf{y} \sim \mathcal{N}(\mathbf{m}, \mathbf{C})$ given the above hyperparameters, where,

$$\mathbf{C} = \left[\mathbf{\Lambda}^{-1} + \frac{1}{\sigma^2}\mathbf{\Phi}^\top\mathbf{\Phi}\right]^{-1},$$
$$\mathbf{m} = \frac{1}{\sigma^2}\mathbf{C}\mathbf{\Phi}^\top\mathbf{y}.$$

- Use the predictive distribution

$$\begin{aligned} p(y^*|\mathbf{y}, \mathbf{X}, \mathbf{x}^*) &= \int \mathcal{N}\left(y^*\middle|\boldsymbol{\phi}^{*\top}\mathbf{w}, \sigma^2\right)\mathcal{N}(\mathbf{w}|\mathbf{m}, \mathbf{C})\, d\mathbf{w}, \\ &= \mathcal{N}\left(y^*\middle|\boldsymbol{\phi}^{*\top}\mathbf{m}, \sigma^2 + \boldsymbol{\phi}^{*\top}\mathbf{C}\boldsymbol{\phi}^*\right) \end{aligned} \tag{15}$$

  for query inputs, $\mathbf{x}^*$. This gives us the useful expectations,

$$\mathbb{E}[y^*] = \boldsymbol{\phi}^{*\top}\mathbf{m}, \tag{16}$$
$$\mathbb{V}[y^*] = \sigma^2 + \boldsymbol{\phi}^{*\top}\mathbf{C}\boldsymbol{\phi}^{*\top}. \tag{17}$$

It is actually easier to use the ELBO form with stochastic gradients for learning the parameters of this algorithm, rather than log-marginal likelihood recast using the Woodbury identity. This is because it is plainly in the same form as Equation (4), though it would give the same result as log-marginal likelihood because the "approximate" posterior is the same form as the true posterior, i.e. $q(\mathbf{w}) = \mathcal{N}(\mathbf{w}|\mathbf{m}, \mathbf{C})$. The ELBO for this model is,

$$\mathcal{L} = \sum_{n=1}^{N}\left\langle \log\mathcal{N}\left(y_n\middle|\boldsymbol{\phi}_n^\top\mathbf{w}, \sigma^2\right)\right\rangle_q - \mathrm{KL}[\mathcal{N}(\mathbf{w}|\mathbf{m}, \mathbf{C})\|\mathcal{N}(\mathbf{w}|\mathbf{0}, \mathbf{\Lambda})]\,. \tag{18}$$

More specifically,

$$\left\langle \log\mathcal{N}\left(y_n\middle|\boldsymbol{\phi}_n^\top\mathbf{w}, \sigma^2\right)\right\rangle_q = \log\mathcal{N}\left(y_n\middle|\boldsymbol{\phi}_n^\top\mathbf{m}, \sigma^2\right) - \frac{1}{2\sigma^2}\mathrm{tr}\left(\boldsymbol{\phi}_n^\top\boldsymbol{\phi}_n\mathbf{C}\right),$$
$$\mathrm{KL}[\mathcal{N}(\mathbf{w}|\mathbf{m}, \mathbf{C})\|\mathcal{N}(\mathbf{w}|\mathbf{0}, \mathbf{\Lambda})] = \frac{1}{2}\left[\mathrm{tr}\left(\mathbf{\Lambda}^{-1}\mathbf{C}\right) + \mathbf{m}^\top\mathbf{\Lambda}^{-1}\mathbf{m} - \log|\mathbf{C}| + \log|\mathbf{\Lambda}| - D\right]$$

We have not implemented a stochastic gradient version of this algorithm since it still requires a matrix solve of a $D \times D$ matrix, and so is $\mathcal{O}\left(D^3\right)$ in complexity, per iteration. This is true even if we optimise the posterior covariance directly (or a triangular parameterisation). The GLM presented in the next section circumvents this issue, and is more suited to really large $N$ and $D$ problems.

### 1.3 Bayesian Generalized Linear Models – `GeneralizedLinearModel`

The algorithm of primary interest in *revrand* is the Bayesian generalized linear model (GLM). The general form of the model is,

$$\text{Likelihood:} \quad \prod_{n=1}^{N} p\left(y_n\middle|g\left(\boldsymbol{\phi}_n^\top\mathbf{w}\right), \gamma\right), \tag{19}$$
$$\text{prior:} \quad \mathcal{N}(\mathbf{w}|\mathbf{0}, \mathbf{\Lambda}), \tag{20}$$

for an arbitrary univariate likelihood, $p(\cdot)$, with an appropriate transformation (inverse link) function, $g(\cdot)$, and parameter(s), $\gamma$.

Naturally, both calculating the exact posterior over the weights, $p(\mathbf{w}|\mathbf{y}, \mathbf{X})$, and the log-marginal likelihood, $p(\mathbf{y})$, for hyperparameter learning are intractable since we may have a non-conjugate relationship between the likelihood and prior. Therefore we must resort to approximating the true posterior and the log-marginal likelihood.

Firstly, we approximate the true posterior over weights with a mixture of $K$ diagonal Gaussians,

$$\begin{aligned} p(\mathbf{w}|\mathbf{y}, \mathbf{X}) &\approx q(\mathbf{w})\,, \\ &= \frac{1}{K}\sum_{k=1}^{K}\mathcal{N}(\mathbf{w}|\mathbf{m}_k, \boldsymbol{\Psi}_k)\,, \end{aligned} \tag{21}$$

where $\boldsymbol{\Psi}_k = \mathrm{diag}\big([\Psi_{k,1}, \ldots, \Psi_{k,D}]^\top\big)$, which is inspired by similar approximations made in Gershman et al. (2012) and Nguyen and Bonilla (2014). This is a very flexible form for the approximate posterior, and has the nice consequence that our algorithm will longer have a $\mathcal{O}\big(D^3\big)$ cost associated with that of a full Gaussian covariance.

Then we approximate the log marginal likelihood using variational Bayes with the re-parameterisation trick (Kingma and Welling, 2014). The exact lower bound on log marginal likelihood is,

$$\mathcal{L} = \sum_{n=1}^{N}\left\langle \log p\Big(y_n\Big|g\Big(\boldsymbol{\phi}_n^\top\mathbf{w}\Big)\,,\gamma\Big)\right\rangle_q - \mathrm{KL}\big[\tfrac{1}{K}\textstyle\sum_k \mathcal{N}(\mathbf{w}|\mathbf{m}_k, \boldsymbol{\Psi}_k)\big\|\mathcal{N}(\mathbf{w}|\mathbf{0}, \boldsymbol{\Lambda})\big]\,. \tag{22}$$

This can be expanded,

$$\begin{aligned} \mathcal{L} = \frac{1}{K}\sum_{k=1}^{K}\sum_{n=1}^{N}\left\langle \log p\Big(y_n\Big|g\Big(\boldsymbol{\phi}_n^\top\mathbf{w}\Big)\,,\gamma\Big)\right\rangle_{q_k} + \frac{1}{K}\sum_{k=1}^{K}\langle\log\mathcal{N}(\mathbf{w}|\mathbf{0}, \boldsymbol{\Lambda})\rangle_{q_k} \\ + \mathbb{H}\left[\tfrac{1}{K}\textstyle\sum_k \mathcal{N}(\mathbf{w}|\mathbf{m}_k, \boldsymbol{\Psi}_k)\right]\,, \end{aligned} \tag{23}$$

but unfortunately there are two intractable integrals here: the expected log likelihood, and the entropy of the Gaussian mixture. We can use the lower bound on the entropy term also used in Gershman et al. (2012); Nguyen and Bonilla (2014),

$$\mathbb{H}\left[\tfrac{1}{K}\textstyle\sum_k \mathcal{N}(\mathbf{w}|\mathbf{m}_k, \boldsymbol{\Psi}_k)\right] \geq -\frac{1}{K}\sum_{k=1}^{K}\log\sum_{j=1}^{K}\frac{1}{K}\mathcal{N}(\mathbf{m}_k|\mathbf{m}_j, \boldsymbol{\Psi}_k + \boldsymbol{\Psi}_j)\,, \tag{24}$$

and then we use the re-parameterisation trick in Kingma and Welling (2014) to obtain samples of the expected log likelihood,

$$\sum_{n=1}^{N}\left\langle \log p\Big(y_n\Big|g\Big(\boldsymbol{\phi}_n^\top\mathbf{w}\Big)\,,\gamma\Big)\right\rangle_{q_k} \approx \frac{1}{L}\sum_{l=1}^{L}\sum_{n=1}^{N}\log p\Big(y_n\Big|g\Big(\boldsymbol{\phi}_n^\top\hat{\mathbf{w}}_k^{(l)}\Big)\,,\gamma\Big) \tag{25}$$

where,

$$\hat{\mathbf{w}}_k^{(l)} = f_k\Big(\mathbf{m}_k, \boldsymbol{\Psi}_k, \boldsymbol{\epsilon}^{(l)}\Big) = \mathbf{m}_k + \sqrt{\boldsymbol{\Psi}_k}\odot\boldsymbol{\epsilon}^{(l)}, \qquad \boldsymbol{\epsilon}^{(l)} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}_D)\,. \tag{26}$$

Here $\odot$ is the element-wise product. We can also use this trick to obtain low variance samples of the derivatives for the GLM's parameters and hyperparameters, $\frac{\partial}{\partial \alpha} \langle \log p(y|\alpha) \rangle_{q(\alpha)} \approx \frac{1}{L} \sum_{l=1}^{L} \frac{\partial}{\partial \alpha} \log p\big(y \big| f\big(\alpha, \boldsymbol{\epsilon}^{(l)}\big)\big)$, which simplifies the implementation greatly! The final auto-encoding variational Bayes objective for our GLM is,

$$\mathcal{L} \approx \frac{1}{K} \sum_{k=1}^{K} \Bigg[ \frac{1}{L} \sum_{l=1}^{L} \sum_{n=1}^{N} \log p\Big(y_n \Big| g\Big(\boldsymbol{\phi}_n^{\top} \hat{\mathbf{w}}_k^{(l)}\Big), \gamma\Big) + \log \mathcal{N}(\mathbf{m}_k | \mathbf{0}, \boldsymbol{\Lambda}) - \frac{1}{2} \mathrm{tr}\big(\boldsymbol{\Lambda}^{-1} \boldsymbol{\Psi}_k\big) - \log \sum_{j=1}^{K} \frac{1}{K} \mathcal{N}(\mathbf{m}_k | \mathbf{m}_j, \boldsymbol{\Psi}_k + \boldsymbol{\Psi}_j) \Bigg]. \quad (27)$$

We can straight-forwardly use this objective with a stochastic optimization and Equation (5) or (6).

The most simple and accurate method for approximating the predictive distribution, $p(y^*|\mathbf{y}, \mathbf{X}, \mathbf{x}^*)$ is to Monte-Carlo sample the integral,

$$p(y^*|\mathbf{y}, \mathbf{X}, \mathbf{x}^*) \approx \int p\Big(\mathbf{y} \Big| g\Big(\boldsymbol{\phi}^{*\top} \mathbf{w}\Big), \gamma\Big) \frac{1}{K} \sum_{k=1}^{K} \mathcal{N}(\mathbf{w}|\mathbf{m}_k, \boldsymbol{\Psi}_k)\, d\mathbf{w}. \quad (28)$$

However, this integral is not particularly useful unless we wish to evaluate known $\mathbf{y}^*$ under the model. For prediction, it is more useful to compute (using Monte-Carlo integration) the predictive expectation,

$$\begin{aligned} \mathbb{E}[y^*] &\approx \int \frac{1}{K} \sum_{k=1}^{K} \mathcal{N}(\mathbf{w}|\mathbf{m}_k, \boldsymbol{\Psi}_k) \int y^* p\Big(y^* \Big| g\Big(\boldsymbol{\phi}^{*\top} \mathbf{w}\Big), \gamma\Big)\, dy^* d\mathbf{w} \\ &= \int \mathbb{E}\Big[p\Big(y^* \Big| g\Big(\boldsymbol{\phi}^{*\top} \mathbf{w}\Big), \gamma\Big)\Big] \frac{1}{K} \sum_{k=1}^{K} \mathcal{N}(\mathbf{w}|\mathbf{m}_k, \boldsymbol{\Psi}_k)\, d\mathbf{w}. \end{aligned} \quad (29)$$

Often we find $\mathbb{E}\big[p\big(y^* \big| g\big(\boldsymbol{\phi}^{*\top} \mathbf{w}\big), \gamma\big)\big] = g\big(\boldsymbol{\phi}^{*\top} \mathbf{w}\big)$, however this is only true with with right choice of activation function[2]. Furthermore, it is useful to compute quantiles of the predictive density in order to ascertain the predictive uncertainty. We start by sampling the predictive cumulative density function, $P(\cdot)$,

$$\begin{aligned} &P(y^* \leq \alpha | \mathbf{y}, \mathbf{X}, \mathbf{x}^*) \\ &\quad \approx \int \frac{1}{K} \sum_{k=1}^{K} \mathcal{N}(\mathbf{w}|\mathbf{m}_k, \boldsymbol{\Psi}_k) \int_{-\infty}^{\alpha} p\Big(y^* \Big| g\Big(\boldsymbol{\phi}^{*\top} \mathbf{w}\Big), \gamma\Big)\, dy^* d\mathbf{w} \\ &\quad = \int P\Big(y^* \leq \alpha \Big| g\Big(\boldsymbol{\phi}^{*\top} \mathbf{w}\Big), \gamma\Big) \frac{1}{K} \sum_{k=1}^{K} \mathcal{N}(\mathbf{w}|\mathbf{m}_k, \boldsymbol{\Psi}_k)\, d\mathbf{w}. \end{aligned} \quad (30)$$

Once we have obtained sufficient samples from the (mixture) posterior we can obtain quantiles, $\alpha$, for some chosen level of probability, $p$, using root finding techniques. Specifically, we use root finding techniques to solve the following for $\alpha$,

$$P(y^* \leq \alpha | \mathbf{y}, \mathbf{X}, \mathbf{x}^*) - p = 0. \quad (31)$$

2. That is, if we choose the activation function corresponding to the canonical link function.

### 1.4 Large Scale Gaussian Process Approximation

In *revrand* we approximate Gaussian Processes (Rasmussen and Williams, 2006) with our standard and generalized linear models by using random feature functions such as those of Rahimi and Recht (2007; 2008) and Le et al. (2013). They use Bochner's theorem regarding the relationship between a kernel and the Fourier transform of a non-negative measure (e.g. a probability measure) that establishes the duality of the covariance function of a stationary process and its spectral density,

$$k(\boldsymbol{\tau}) = \int p_{\boldsymbol{\omega}}(\boldsymbol{\omega}) e^{i\boldsymbol{\omega}^\top \boldsymbol{\tau}} d\boldsymbol{\omega}, \tag{32}$$

$$p_{\boldsymbol{\omega}}(\boldsymbol{\omega}) = \int k(\boldsymbol{\tau}) e^{-i\boldsymbol{\omega}^\top \boldsymbol{\tau}} d\boldsymbol{\tau}. \tag{33}$$

where $k(\cdot)$ is a kernel function, and $p_{\boldsymbol{\omega}}(\cdot)$ its spectral density. Rahimi and Recht's main insight (2007) is that we can approximate the kernel by constructing 'suitable' random features and Monte Carlo averaging over samples from $p_{\boldsymbol{\omega}}(\boldsymbol{\omega})$ for *shift invariant* kernels,

$$k(\mathbf{x} - \mathbf{x}') = k(\boldsymbol{\tau}) \approx \frac{1}{D} \sum_{i=1}^{D} \phi_i(\mathbf{x})^\top \phi_i(\mathbf{x}') \,, \tag{34}$$

where $\phi_i(\mathbf{x})$ corresponds to the $i$th sample from the feature map. An example of a radial basis kernel feature vector construction using the above approximation is,

$$\begin{aligned} \left[\phi_i(\mathbf{x}) \,, \phi_{D+i}(\mathbf{x})\right] &= \frac{1}{\sqrt{D}} \left[\cos(\boldsymbol{\omega}_i^T \mathbf{x}), \sin(\boldsymbol{\omega}_i^T \mathbf{x})\right] \,, \\ \text{with } \boldsymbol{\omega}_i &\sim \mathcal{N}\left(\boldsymbol{\omega}_i \middle| \mathbf{0}, l_\phi^{\text{-}1} \mathbf{I}_d\right) \,, \end{aligned} \tag{35}$$

for $i = 1, \ldots, D$, which in fact is a mapping into a $2D$-dimensional feature space. See Table 1 for some of the random kernel approximations we use in *revrand*.

Table 1: Kernels and the corresponding Fourier weight ($\boldsymbol{\omega}_i$) sampling distributions for the Rahimi and Recht (2007)-style random bases in *revrand*. Here GAL refers to a multivariate Laplace distribution Kozubowski et al. (2013).

| **Kernel** | $k(\mathbf{x} - \mathbf{x}')$ | $p_{\boldsymbol{\omega}}(\boldsymbol{\omega})$ |
|---|---|---|
| RBF | $\exp\left(-\frac{\|\mathbf{x}-\mathbf{x}'\|^2}{2l_\phi^2}\right)$ | $\boldsymbol{\omega}_i \sim \mathcal{N}\left(\mathbf{0}, l_\phi^{\text{-}1}\mathbf{I}_d\right)$, |
| Laplace | $\exp\left(-\frac{\|\mathbf{x}-\mathbf{x}'\|}{l_\phi}\right)$ | $\boldsymbol{\omega}_i \sim \prod_d \text{Cauchy}\left(l_\phi^{\text{-}1}\right)$ |
| Cauchy | $\frac{1}{1+\left(\|\mathbf{x}-\mathbf{x}'\|/l_\phi\right)^2}$ | $\boldsymbol{\omega}_i \sim \text{GAL}\left(1, \mathbf{0}, l_\phi^{\text{-}1}\mathbf{I}_d\right)$ |
| Matern 3/2 | $\left(1 + \frac{\sqrt{3}\|\mathbf{x}-\mathbf{x}'\|}{l_\phi}\right)\exp\left(-\frac{\sqrt{3}\|\mathbf{x}-\mathbf{x}'\|}{l_\phi}\right)$ | $\boldsymbol{\omega}_i \sim \text{t}_{\nu=3}\left(\mathbf{0}, l_\phi^{\text{-}1}\mathbf{I}_d\right)$ |
| Matern 5/2 | $\left(1 + \frac{\sqrt{5}\|\mathbf{x}-\mathbf{x}'\|}{l_\phi} + \frac{5\|\mathbf{x}-\mathbf{x}'\|^2}{3l_\phi^2}\right)\exp\left(-\frac{\sqrt{5}\|\mathbf{x}-\mathbf{x}'\|}{l_\phi}\right)$ | $\boldsymbol{\omega}_i \sim \text{t}_{\nu=5}\left(\mathbf{0}, l_\phi^{\text{-}1}\mathbf{I}_d\right)$ |

All of the kernel functions in Table 1 have length scale hyperparameters, and in *revrand* these can be isotropic or anisotropic length scales that are learned alongside the other hyperparameters. We have also implemented a few other variants of these random basis functions,

namely the Fastfood (Le et al., 2013), A la Carte (Yang et al., 2014) and orthogonal random features (Yu et al., 2016) that improve either scalability, representational flexibility respectively or both. In Figure 1 we compare the basis approximations with their corresponding kernels. We refer the reader to our demo notebooks for further discussion and experiments comparing these approximate basis functions and the kernels they approximate.

An important feature of *revrand* is that we can combine these random features (and non-random features) to build even more expressive features. In particular, we support basis *concatenation* while still allowing basis function hyperparameter learning,

$$\mathbf{\Phi}_{\text{cat}} = \left[\phi_1(\mathbf{X}, \theta_1), \phi_2(\mathbf{X}, \theta_2), \ldots, \phi_P(\mathbf{X}, \theta_P)\right]. \tag{36}$$

In the following manner this approximates kernel addition,

$$\lambda_1 k_1(\mathbf{X}, \mathbf{X}, \theta_1) + \lambda_2 k_2(\mathbf{X}, \mathbf{X}, \theta_2) + \ldots + \lambda_P k_P(\mathbf{X}, \mathbf{X}, \theta_P) \approx \mathbf{\Phi}_{\text{cat}} \mathbf{\Lambda} \mathbf{\Phi}_{\text{cat}}^\top, \tag{37}$$

where $\mathbf{\Lambda} = \text{diag}([\lambda_1 \mathbf{1}, \lambda_2 \mathbf{1}, \ldots, \lambda_P \mathbf{1}])$ and each vector of $\mathbf{1}$ is the same dimension as the corresponding basis. Here the regression regularizer, or weight prior variance in Equations (13) and (20), acts as a kernel mixing/amplitude parameter. Kernel products also have an equivalent representation with basis functions (outer product of bases), however we have not yet implemented this.

We also support *partial application* of basis functions to certain dimensions of the inputs, $\mathbf{X}$, while also allowing concatentation, e.g,

$$\mathbf{\Phi}_{\text{cat,partial}} = \left[\phi_1(\mathbf{X}_{1:10}, \theta_1), \phi_2(\mathbf{X}_{5:D}, \theta_2), \ldots, \phi_P(\mathbf{X}, \theta_P)\right] \tag{38}$$

Where the subscript of $\mathbf{X}$ denotes (arbitrary) column slices. See *revrand*'s documentation on how to use this features.

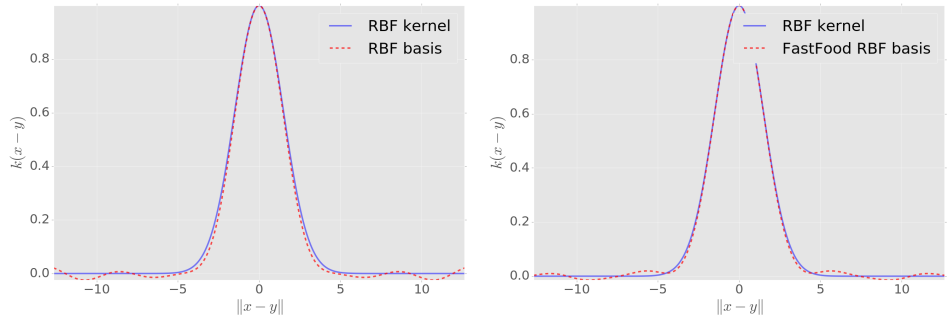
## 2. Experiments

In this section we apply *revrand*'s standard linear model (SLM) and generalized linear model (GLM) to a few standard regression and classification datasets to ascertain how they perform relative to other standard machine learning algorithms for these prediction tasks.

### 2.1 Boston Housing Regression

In Table 2 we compare *revrand*'s SLM and GLM on the Boston Housing dataset. This is a regression prediction problem, where the aim is to estimate house prices in suburbs in Boston based on attributes of the buildings, the region and the surrounding environment. There are 506 samples and 13 attributes in total.

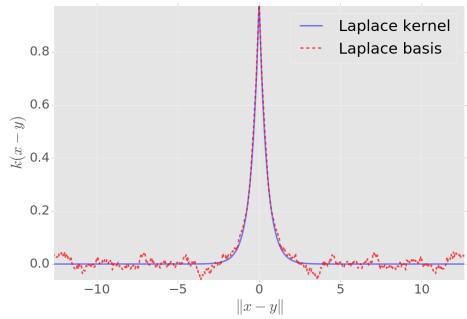
For this experiment we use 5 fold cross validation, and use R-square and mean standardised log-loss (MSLL) to evaluate the predictions on the held-out folds. The SLM and GLM use random radial basis fourier features (RBF kernel approximations) with 400 components (800 bases in total) for the SLM, and 100 components (200 bases) for the GLM, both concatenated with a linear basis. We found the GLM performed better with fewer iterations with fewer bases. We evaluate 100 random starts for the SLM, 500 for the GLM, of the length-scales (one for each dimension of the inputs, $\mathbf{X}$), regularizer and likelihood



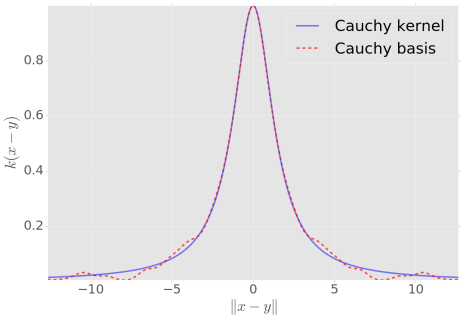

(a) RBF



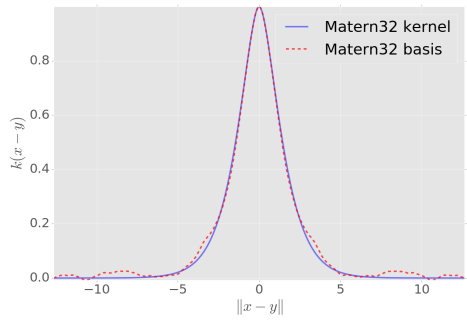

(b) RBF Fastfood



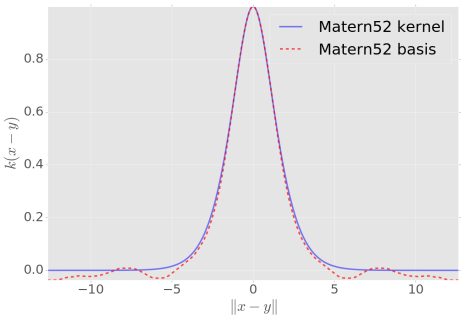

(c) Laplace



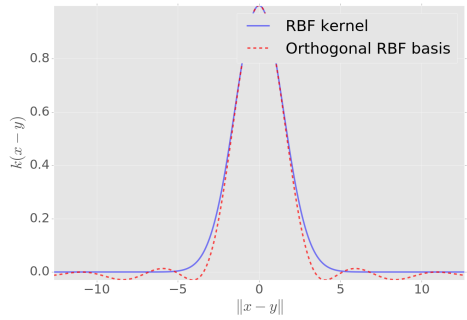

(d) Cauchy



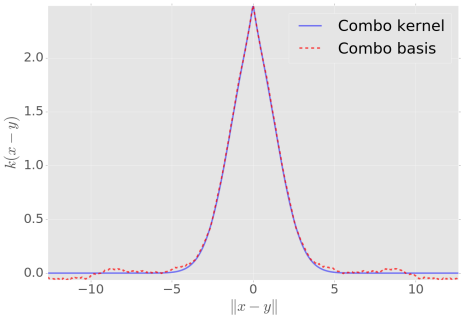

(e) Matern 3/2




(f) Matern 5/2




(g) RBF Orthogonal




(h) RBF + Laplace

Figure 1: Comparison of various kernels and their basis approximations. In each of these figures 1500 random bases have been used, and the inputs to the kernels are 10 dimensional, $\mathbf{x} \in \mathbb{R}^{10}$.

Table 2: Regression performance on the Boston Housing dataset.

| Algorithm | R-square | MSLL |
|---|---|---|
| *SLM* | 0.9018 (0.0134) | -1.504 (0.1191) |
| *GLM* | 0.8411 (0.0491) | -0.9209 (0.1530) |
| GP | **0.9027 (0.0137)** | **-1.1792 (0.1581)** |
| RF | 0.8467 (0.0709) | N/A |
| SVM | 0.6295 (0.0863) | N/A |

Table 3: Binary classification performance on the USPS digits dataset for digits '3' and '5'.

| Algorithm | Log-loss | Error (%) |
|---|---|---|
| *GLM* | 0.1138 | **2.07** |
| Logistic | 0.1734 | 3.62 |
| SVM | **0.1003** | 6.99 |
| GP | 0.1405 | 2.59 |
| RF | 0.1368 | 2.72 |

variance, parameters, drawing them from Gamma distributions, and then optimize from the best evaluation.

We compare against Scikit Learn (Pedregosa et al., 2011) implementations of Gaussian processes, support vector machines (SVM), and random forests (RF). The GP is given the same kernel as the SLM and GLM, that is an additive RBF and dot-product kernel. The GP's kernel amplitudes and it's RBF length-scales are initialised at 1, the RF uses 40 decision trees, and the SVM uses a RBF kernel, where the length-scale is chosen by a grid search with cross-validation. The exact experiment can be replicated in the `boston_housing.ipynb` jupyter notebook.

As we would expect the GP yeilds the best performance, closely followed by the SLM. The GLM and RF perform similarly, and the SVM the worst.

### 2.2 Handwritten Digits Classification

We next test *revrand*'s GLM on a classification datasets. We chose the USPS handwritten digits binary classification task, also presented in Rasmussen and Williams (2006). Here the aim is to classify the digits '3' and '5' from $16 \times 16$ pixel black and white images. There are 767 training instances, and 773 testing instances. We use log-loss and percent error as validation scores.

The GLM uses a Bernoulli likelihood with a logistic activation function, it also uses a RBF random fourier features with 800 components (1600 bases), and again 100 random initialisations of the length scale (isotropic) and regularizer with optimization proceeding from the best.

We compare to a SVM, RF with 40 estimators, logistic regressor with our same random fourier features (un-optimized), and a GP classifier with the same likelihood as our GLM, and using a Laplace posterior approximation. Again, all of these are implementations in Scikit Learn (Pedregosa et al., 2011). The GP and SVM also use RBF kernels, and the

Table 4: Regression performance on the larger SARCOS robotic arm dataset. This table (apart from the GLM) has been reproduced from Rasmussen and Williams (2006).

| **Algorithm** | **Approximation size** | **SMSE** | **MSLL** |
|---|---|---|---|
| *GLM* | 256 | 0.0358 | -1.6644 |
| | 512 | 0.0252 | -1.8397 |
| | 1024 | 0.0207 | -1.9341 |
| | 2048 | 0.0171 | -2.0243 |
| | 8192 | 0.0152 | -1.9804 |
| *SD* | 256 | 0.0813 (.0198) | -1.4291 (.0558) |
| | 512 | 0.0532 (.0046) | -1.5834 (.0319) |
| | 1024 | 0.0398 (.0036) | -1.7149 (.0293) |
| | 2048 | 0.0290 (.0013) | -1.8611 (.0204) |
| | 4096 | 0.0200 (.0008) | -2.0241 (.0151) |
| *SR* | 256 | 0.0351 (.0036) | -1.6088 (.0984) |
| | 512 | 0.0259 (.0014) | -1.8185 (.0357) |
| | 1024 | 0.0193 (.0008) | -1.9728 (.0207) |
| | 2048 | 0.0150 (.0005) | -2.1126 (.0185) |
| | 4096 | 0.0110 (.0004) | -2.2474 (.0204) |
| *PP* | 256 | 0.0351 (.0036) | -1.6940 (.0528) |
| | 512 | 0.0259 (.0014) | -1.8423 (.0286) |
| | 1024 | 0.0193 (.0008) | -1.9823 (.0233) |
| | 2048 | 0.0150 (.0005) | -2.1125 (.0202) |
| | 4096 | 0.0110 (.0004) | -2.2399 (.0160) |
| *BCM* | 256 | 0.0314 (.0046) | -1.7066 (.0550) |
| | 512 | 0.0281 (.0055) | -1.7807 (.0820) |
| | 1024 | 0.0180 (.0010) | -2.0081 (.0321) |
| | 2048 | 0.0136 (.0007) | -2.1364 (.0266) |

SVM uses a grid search and cross validation to select the length scale. The results are summarized in Table 3.

Interestingly, the SVM obtains the best log-loss, but has the worst error. The GLM obtains the best error rate, and the second best log-loss, with the GP and RF also performing well.

### 2.3 SARCOS Regression

The last experiment is a large-scale regression task presented in Rasmussen and Williams (2006, Chapter 8). The task is to predict the joint torque in a SARCOS robot arm from all of the 7 joint states (position, angular velocity and acceleration), 21 dimensions in all. There are 44,484 training examples, and 4449 test points. This dataset is too large for a typical GP, and so is a good test for *revrand*'s GLM using stochastic gradients.

We summarise the results in Table 4, where we have reproduced the results of the other algorithms from Rasmussen and Williams (2006). SD is a regular GP using a random

subset of data, SR is the subset of regressors method, PP projected processes and BCM the Bayesian committee machine.

We ran the GLM with 1 million iterations of stochastic gradients (with a batch size of 10), with 500 preliminary iterations with random values for the algorithm parameters and hyperparameters. A RBF with automatic relevance determination was used, and the inputs, $\mathbf{X}$ were standardised to be consistent with the kernels used in the original experiment.

In Table 4 the 'Approximation size' column refers the number of unique components (half the number of bases) for the GLM, the number of data points for SD, partitions for the BCM, and the number of inducing points for SR and PP. We have only run the GLM once for each setting of the number of components at this stage (because of the time required to run 1 million iterations on a single machine), we will present more comprehensive results in the future.

The preliminary results in Table 4 show that the GLM is competitive with the other methods (not quite as good as SR, PP and BCM for large approximation sizes), and is better than SD. It is worth noting though that the GLM has a computational complexity of $\mathcal{O}(ND)$ per iteration, whereas SD is $\mathcal{O}\big(D^3\big)$ and SR, PP are $\mathcal{O}\big(D^2N\big)$ and the BCM is the same cost as running $D$ independent GPs (each taking a portion of N). Hence, we are able to use larger approximation sizes than the corresponding methods.

## References

Christopher M. Bishop. Pattern recognition. *Machine Learning*, 128:1–58, 2006.

Samuel Gershman, Matt Hoffman, and David Blei. Nonparametric variational inference. In *Proceedings of the 29th International Conference on Machine Learning (ICML)*, July 2012.

Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Diederik P Kingma and Max Welling. Auto-encoding variational bayes. In *Proceedings of the 2nd International Conference on Learning Representations (ICLR)*, 2014.

Tomasz J. Kozubowski, Krzysztof Podgórski, and Igor Rychlik. Multivariate generalized laplace distribution and related random fields. *Journal of Multivariate Analysis*, 113: 59–72, 2013.

Quoc Le, Tamás Sarlós, and Alex Smola. Fastfood-approximating kernel expansions in log-linear time. In *Proceedings of the international conference on machine learning (ICML)*, 2013.

Trung V. Nguyen and Edwin V. Bonilla. Automated variational inference for Gaussian process models. In *Advances in Neural Information Processing Systems (NIPS)*, 2014.

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.

Ali Rahimi and Benjamin Recht. Random features for large-scale kernel machines. In *Advances in Neural Information Processing Systems (NIPS)*. 2007.

Ali Rahimi and Benjamin Recht. Weighted sums of random kitchen sinks: Replacing minimization with randomization in learning. In *Advances in Neural Information Processing Systems (NIPS)*. 2008.

Carl Edward Rasmussen and Christopher K. I. Williams. *Gaussian processes for machine learning*. The MIT Press, Cambridge, Massachusetts, 2006.

Zichao Yang, Alexander J. Smola, Le Song, and Andrew Gordon Wilson. A la carte-learning fast kernels. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, pages 1098–1106, 2014.

Felix X. Yu, Ananda Theertha Suresh, Krzysztof M. Choromanski, Daniel N. Holtmann-Rice, and Sanjiv Kumar. Orthogonal random features. In *Advances in Neural Information Processing Systems*, pages 1975–1983, 2016.