Stan's ADVI

Alexandre Piche

May 1, 2016

1 Introduction

Variational inference (VI) is an approximate Bayesian inference technique that gained a lot of ground in the recent years. It differs from the MCMC techniques by posing the estimation of the conditional distribution as an optimization problem. VI is usually a lot faster than conventional MCMC methods, however it can be tedious to derive the algorithm for a new model.

In this project, we will explore how the Stan's ADVI algorithm compares to exact MCMC methods in term of performance accuracy and computation time. We will explore three types of model: logistic regression, Bayesian neural network, and Gaussian process on the Dorothea dataset.

2 Variational Inference

In Bayesian statistics, we are often interested in approximating the conditional distribution of hidden variables z given the observed variables x.

$$p(z|x) \quad = \quad \frac{p(z,x)}{p(x)}$$

Unfortunately, often $p(x) = \int_z p(z,x)$ is intractable or very expensive to compute. VI is an optimization technique to approximate $p(z|x)$.

2.1 Evidence Lower Bound

In VI to approximate the conditional distribution, we minimize a notion of distance between $p(z|x)$ and a simple distribution. We first need to specify a family of distributions $\mathcal{Q}$ over the latent variables z. Then, we optimize $q(z) \in \mathcal{Q}$ to minimize the KL divergence:

$$q^*(z) \quad = \quad \arg\min_{q(z)\in\mathcal{Q}} KL(q(z)||p(z|x))$$

where $KL(q(z)|p(z|x))$ is given by:

$$\begin{aligned} KL(q(z)||p(z|x)) &= E[\log q(z)] - E[\log p(z|x)] \\ &= E[\log q(z)] - E[\log p(z,x)] + \log p(x) \end{aligned}$$

where $p(x)$ is intractable in most cases, but is a constant wrt to $q(z)$. Instead, we optimize the evidence lower bound (ELBO):

$$\begin{aligned} ELBO(q) &= E[\log p(z,x)] - E[\log q(z)] \\ &= E[\log p(x|z)] - KL(q(z)||p(z)) \end{aligned}$$

From the second equation, it is easy to see that maximizing the ELBO is the same as minimizing the KL divergence.

The ELBO also has a statistical intuitive interpretation. Specifically, writing it as:

$$ELBO(q) = E[\log p(z)] + E[\log p(x|z)] - E[\log q(z)]$$

The first term is the expected likelihood, thus it encourages the distribution to place the mass on configurations of the latent variables that explain the observed data[1]. While the second term is the negative divergence between the variational distribution and the prior, thus encouraging the distribution to be close to the prior. Those properties are aligned with the Bayesian framework.

The $ELBO$ is that it lower-bounds the log evidence, $\log p(x) \geq ELBO(q)$ for any $q(z)$ [1], since:

$$\log p(x) = KL(q(z)||p(z|x)) + ELBO(q)$$

2.2 Mean-Field Variational Inference

To ease the computation, we usually make the assumption that the latent variables are mutually independent, specifically:

$$q(z) = \prod_{j=1}^{m} q_j(z_j)$$

Note that each latent variable z_j has its own distribution $q_j \in \mathcal{Q}$. The mean-field approximation can capture any marginal distribution of the latent variables, but not the correlation between them.

2.3 Coordinate Ascent Variational Inference

We usually maximize the ELBO using coordinate ascent variational inference (CAVI). It iteratively optimizes each distribution q_j, while holding q_{-j} fix. Taking advantage of using mean-field VI, we can write the optimal $q*_j(z_j)$ as:

$$\begin{aligned} q*_j(z_j) &\propto \exp\{E_{-j}[\log p(z_j|z_{-j},x)]\} \\ &\propto \exp\{E_{-j}[\log p(z_j,z_{-j},x)]\} \end{aligned}$$

The ELBO can be written as:

$$ELBO(q_j) = E_j[E_{-j}[\log p(z_j,z_{-j},x)]] - E_j[\log q_j(z_j)] + c$$

where $\log q_{-j}(z_{-j})$ is absorbed by the constant c, since it does not depend on z_j.

2.4 Stochastic Variational Inference

Going through the entire large dataset to compute the ELBO is expensive. Instead, we can use batch of data to maximize the ELBO, a method call stochastic variational inference [3]. Specifically, we can use batch of data to obtain noisy, but unbias gradients, and to converge to a local minimum. The step-size must satisfy [7] :

$$\sum_t \epsilon_t = \infty \quad ; \quad \sum_t \epsilon_t^2 < \infty$$

3 Automatic Differentiation Variational Inference

We now have the tools to explore Stan Automatic Differentiation Variational Inference (ADVI) algorithm.

3.1 Transformation of Constrained Variables

ADVI first transforms the latent variables to the real space. Doing so removed constraints such as the variance being positive. Let T be a one-to-one differentiable mapping from the constrained space to the real space and $\zeta = T(\theta)$ [4]. The transformed joint density is given by:

$$g(X,\zeta) = p(X,T^{-1}(\zeta))|detJ_{T^{-1}}|$$

where p is the original joint density of the latent variables, and $J_{T^{-1}}$ is the jacobian of the inverse of the transformation T.

$$L(\phi) = E_{q(\zeta|\phi)}[\log p(y, T^{-1}(\zeta)) + \log |detJ_{T^{-1}}(\zeta)|] - E_{q(\zeta|phi)}[\log q(\zeta|\phi)]$$

3.2 Variational Approximation

ADVI then approximates the variational distribution $q(\zeta)$ with a mean-field Gaussian, such that:

$$q(\zeta|\phi) = N(\zeta|\mu, \sigma) = \prod_{k=1}^{K} N(\zeta_k|\mu_k, \sigma_k)$$

where the vector $\phi = (\mu_1, \ldots, \mu_K, \sigma_1, \ldots, \sigma_K)$. The ELBO is thus given by:

$$\mathcal{L}(\mu, \sigma) = E_{q(\zeta)}[\log p(X, T^{-1}(\zeta)) + \log |detJ_{T^{-1}}(\zeta)|] + \frac{K}{2}(1 + \log 2\pi) + \sum_{k=1}^{K} \log \sigma_k$$

3.3 Stochastic Optimization

We could maximize the ELBO on the real space as:

$$\begin{aligned} \mu*, \sigma* &= \text{arg max}_{\mu,\sigma} \mathcal{L}(\mu, \sigma) \\ &\quad \text{such that } \sigma > 0 \end{aligned}$$

but the expectation is intractable. Instead, we transform σ to be unconstrained as $\omega = \log \sigma$. We can use the elliptical standardization, by defining $\eta = S_{\mu,\omega}(\zeta) = \text{diag } (\exp(\omega)^{-1})(\zeta - \mu)$ The density is now given by:

$$\begin{aligned} q(\eta; 0, I) &= N(\eta; 0, I) \\ &= \prod_k N(\eta_k; 0, 1) \end{aligned}$$

and the optimization problem is now given by:

$$\begin{aligned} \mu*, \omega* &= \text{arg max}_{\mu,\omega} \mathcal{L}(\mu, \omega) \\ &= \text{arg max}_{\mu,\omega} E\left[\log p(X, T^{-1}(S^{-1}_{\mu,\omega}(\eta))) + \log |\det\ J_{T^{-1}}(S^{-1}_{\mu,\omega}(\eta))| \right] + \sum_{k=1}^{K} \omega_k \end{aligned}$$

Where the gradients are given by:

$$\begin{aligned} \nabla_\mu \mathcal{L}(\mu, \omega) &= E_{N(\eta)}[\nabla_\theta \log p(X, \theta) \nabla_\zeta T^{-1}(\zeta) + \nabla_\zeta \log |\det\ J_{T^{-1}}(\zeta)|] \\ \nabla_{\omega_k} \mathcal{L}(\mu, \omega) &= E_{\eta_k}[(\nabla_{\theta_k} \log p(X, \theta) \nabla_{\zeta_k} T^{-1}(\zeta) + \nabla_{\zeta_k} \log |\det\ J_{T^{-1}}(\zeta)|) \eta_k \exp(\omega_k)] + 1 \end{aligned}$$

We use MC integration to get an unbias estimate of the gradient that we can use in SVI.

ADVI has complexity $O(BMK)$ per iteration, where B is the batch size, M the number of MC samples while CAVI has complexity $O(NK)$. Note that the optimization of the ELBO stop when the change is under a certain threshold, 0.01 in our case.

4 Experiments on the Dorothea Dataset

In 2003, NIPS had a features selection challenge consisting of 5 datasets. The biggest of all was the Dorothea dataset with 100000 structural molecule features. The task require predicting whether a chemical compound is binding to thrombin or not. This is an important task related to drug discovery. Given that half the features were designed to be irrelevant, feature selection is at the center of the problem. We will explore three methods: logistic regression (LR), a Bayesian neural network (BNN), and a Gaussian process (GP). For every models, we experimented with a variant of automatic relevance determination (ARD). To make computations reasonable, we used PCA to reduce the dimension to 75 inputs.

We will use the balance error rate (BER) to measure the accuracy of our models:

$$BER = 0.5 \times \left(\frac{TP}{TP + FN} + \frac{TN}{TN + FP} \right)$$

where TP is true positive, FN is false negative, TN is true negative, and FP is false positive.

4.1 Logistic Regression

Logistic regression is one of the simplest classification tool, but is performing well on real world problems. It can divide correctly any linearly divisible dataset.

4.1.1 LR Results

I used a normal prior bounded on $[-0.5, 0.5]$ for the β to avoid some awkward convergence issues, and an unbounded normal prior for the bias term α. Both normal priors have unknown mean.

LR Comparative Results		
Methods	BER	Computation time
HMC	0.223	41.87
ADVI	0.230	4.73

The accuracy and computation time are fairly simillar in both cases, but we can note that ADVI is faster but does not reach the same accuracy level than HMC.

4.1.2 LR-ARD Results

We now let each β_j come from $N(0, \sigma_j)$, where σ_j is unknown. As the posterior for $\sigma_j \to 0$, β_j will be near 0 and the feature will have almost no impact on the prediction.

LR-ARD Comparative Results		
Methods	BER	Computation time
HMC	0.216	94.38
ADVI	0.226	5.50

In both the LR and the LR-ARD, the ADVI is a lot faster in proportion to HMC, but comparable in absolute term, and the balance error rate is comparable for the two methods.

4.2 Bayesian Neural Network

Radford Neal used NewBayes, a Bayesian neural network, in the 2003 competition to attain the best average predictive performance over the 5 datasets. He used a Dirichlet Diffusion tree, to average different predictions, thus we won't be able to replicate his results here.

We build a neural network with one hidden layer and 25 neurons. The stan code for the neural network is inspired by Herra Huu's code available at: https://groups.google.com/forum/!topic/stan-users/3QBBpo11Lus . It has been modified to use ARD and tanh activation function [5].

4.2.1 BNN Results

For every weights, I used a normal prior with mean 0 and unknown variance specific for each level.

BNN Comparative Results		
Methods	BER	Computation time
HMC	0.2704	36325.61
ADVI	0.5	359.39

The results clearly highlight the trade-off between accuracy and speed. Using the HMC sampling results in a much higher accuracy, but the computation are 10 times more expensive. For the next sections, the HMC sampling methods were not possible due to computation constraints. They are presented since they provide us with insight on the ADVI method.

4.2.2 BNN-ARD Results

In addition to the normal prior used for the previous BNN, I used a different variance σ_j for every input weight β_j [2].

BNN-ARD Comparative Results		
Methods	BER	Computation time
HMC	-	-
ADVI	0.3542	7523.39

Using ARD improved ADVI results, but is still not as accurate at the simple BNN using HMC.

4.3 Gaussian Process

Experimenting with the BNN, I noticed that as a grow the number of nodes the performance improve, however after a certain threshold it was not computationally sustainable. Given that infinitely wide NN tends to Gaussian processes (GPs), it was natural to include the later to our analysis. GPs can model any continuous function. Since the HMC computation were too expensive using Stan, I also included results using the GPy package [8].

4.3.1 GP Results

We modeled the probability $p(y|x) = 1$ using the GP, written as:

$$p(y_i|x_i) \quad = \quad \sigma(f(x_i))$$

where σ is the logistic function, and f is the GP posterior [6]. We will use the squared exponential kernel:

$$k(x, x') \quad = \quad \tau^2 \exp\left(-\sum_{d=1}^{D} \frac{(x_d - x'_d)}{2\sigma^2} \right)$$

GP Comparative Results		
Methods	BER	Computation time
HMC	-	-
ADVI	0.4242	2354.45
GPy	0.2595	34.48

With the GPy's model, we are achieving a better result that the BNN using HMC, but surprisingly we are not achieving the accuracy results from the logistic regression. The ADVI results are nowhere as good as the one attained by GPy.

4.3.2 GP-ARD Results

Finally for the GP-ARD, we will use the ARD kernel:

$$k(x, x') \quad = \quad \tau^2 \exp\left(-\sum_{d=1}^{D} \frac{(x_d - x'_d)}{2\sigma_d^2} \right)$$

Note that as $\sigma_d^2 \to \infty$ the feature d will become irrelevant.

GP-ARD Comparative Results		
Methods	BER	Computation time
HMC	-	-
ADVI	0.4835	8835.94
GPy	0.1860	41.87

Surprisingly, increasing the flexibility of the model decrease the prediction accuracy for the ADVI. It might be due to an increasing difficulty in approximating the posterior. Using GPy's classification GP, we are attaining our best results overall which is what we expected since the GP-ARD is the most flexible model we used, and can model any function.

5 Conclusion

We compared the ADVI method to the conventional MCMC method used by Stan on three bayesian models: logistic regression, bayesian neural network and Gaussian process. From our results, it is obvious that HMC is always reaching the best balance error rate on the test set, but for more complex models HMC is very expensive and might not be computationally feasible. I think the poor accuracy performance of the ADVI are due to complex posteriors. It would be interesting to tweak the learning rate or the optimization methods, to see if they actually improve the accuracy performance. On a different note, the good performance of the logistic regression, and computation speed from GPy are both impressive.

References

[1] D. M. Blei, A. Kucukelbir, and J. D. McAuliffe. Variational inference: A review for statisticians. *arXiv preprint arXiv:1601.00670*, 2016.

[2] I. Guyon, S. Gunn, M. Nikravesh, and L. A. Zadeh. *Feature extraction: foundations and applications*, volume 207. Springer, 2008.

[3] M. D. Hoffman, D. M. Blei, C. Wang, and J. Paisley. Stochastic variational inference. *The Journal of Machine Learning Research*, 14(1):1303–1347, 2013.

[4] A. Kucukelbir, R. Ranganath, A. Gelman, and D. Blei. Automatic variational inference in stan. In *Advances in Neural Information Processing Systems*, pages 568–576, 2015.

[5] R. M. Neal. *Bayesian learning for neural networks*, volume 118. Springer Science & Business Media, 2012.

[6] C. E. Rasmussen. Gaussian processes for machine learning. 2006.

[7] H. Robbins and S. Monro. A stochastic approximation method. *The annals of mathematical statistics*, pages 400–407, 1951.

[8] The GPy authors. GPy: A gaussian process framework in python. `http://github.com/SheffieldML/GPy`, 2012–2015.

ADVI experiments

April 28, 2016

```
In [1]: import rpy2
        import time
        %load_ext rpy2.ipython
```

```
In [2]: %%R
        library(rstan)
        library(loo)
        source("read_one_stan_csv.r")
```

```
/usr/local/lib/python2.7/site-packages/rpy2/robjects/functions.py:106: UserWarning: Loading required pa

  res = super(Function, self).__call__(*new_args, **new_kwargs)
/usr/local/lib/python2.7/site-packages/rpy2/robjects/functions.py:106: UserWarning: rstan (Version 2.9.

  res = super(Function, self).__call__(*new_args, **new_kwargs)
/usr/local/lib/python2.7/site-packages/rpy2/robjects/functions.py:106: UserWarning: For execution on a l
rstan_options(auto_write = TRUE)
options(mc.cores = parallel::detectCores())

  res = super(Function, self).__call__(*new_args, **new_kwargs)
/usr/local/lib/python2.7/site-packages/rpy2/robjects/functions.py:106: UserWarning: This is loo version

  res = super(Function, self).__call__(*new_args, **new_kwargs)
```

1 Logistic Regression

Abstract: DOROTHEA is a drug discovery dataset. Chemical compounds represented by structural molecular features must be classified as active (binding to thrombin) or inactive. This is one of 5 datasets of the NIPS 2003 feature selection challenge.

http://pages.cs.wisc.edu/~dpage/kddcup2001/Cheng.pdf

```
In [3]: !cat examples/lr/lr.stan
```

```
functions {
  real BER(vector y, vector yhat, int Nt){
    real TN;
    real FN;
    real TP;
    real FP;
    real BER;
    real b1;
    TN = 0.0;
    FN = 0.0;
    TP = 0.0;
```

```
    FP = 0.0;
    for(i in 1:Nt){
      if(round(yhat[i]) == 0 && y[i] == 0) {
        TN = TN + 1.0;
      }
      else if(round(yhat[i]) == 0 && y[i] == 1) {
        FN = FN + 1.0 ;
      }
      else if(round(yhat[i]) == 1 && y[i] == 1) {
        TP = TP + 1.0 ;
      }
      else if(round(yhat[i]) == 1 && y[i] == 0) {
        FP = FP + 1.0 ;
      }
    }
    BER = 0.5 * (FP/(FP+TN) + FN/(FN+TP));
    return BER;
  }
}


data {
  int N;
  int Ntest;
  int D;
  int<lower=0,upper=1> y[N];
  matrix[N,D] x;
  vector<lower=0,upper=1>[Ntest] ytest;
  matrix[Ntest,D] xtest;
  real<lower=0> lambda;
}
parameters {
  //real<lower=-0.5,upper=0.5> alpha;
  vector<lower=-0.5,upper=0.5>[D] beta;
  real alpha;
  //vector[D] beta;
  real<lower=0> sigma;
}
transformed parameters {
  vector[D] zeros;        // zeros for mean of MVN
  for (i in 1:D)
    zeros[i] = 0.0;
}
model {
  alpha ~ normal(0,5);
  sigma ~ inv_gamma(1,2);
  beta ~ normal(0, sigma);
  //increment_log_prob(- lambda * dot_self(beta)); // Ridge to perform variable selection
  //for (d in 1:D){
    //increment_log_prob(- lambda * fabs(beta[d])); // Lasso to perform variable selection
  //}

  for(n in 1:N)
    y[n] ~ bernoulli(inv_logit(alpha + x[n]*beta)); //more efficient
```

```
}
generated quantities {
  real<lower=0> errors[Ntest];
  real<lower=0> average_error;
  vector[N] log_lik;
  vector[Ntest] predictions;

  for(n in 1:Ntest){
    predictions[n] = inv_logit(alpha + xtest[n]*beta);
    errors[n] = fabs(ytest[n] - round(predictions[n]));
  }
  average_error = BER(ytest, predictions, Ntest);//sum(errors) / Nt;
  //average_error = sum(errors) / Ntest;
  for(n in 1:N)
    log_lik[n] = bernoulli_logit_log(y[n], alpha + x[n] * beta);
}
```

In [4]: `!cat examples/dorothea/datadorothea.r`

```
load("dorothea.rda")

pmatrix <- scale(x.train)
princ <- prcomp(x.train)

nComp <- 75
pca_x.train <- data.frame(predict(princ, newdata=x.train)[,1:nComp])
pca_x.valid <- data.frame(predict(princ, newdata=x.valid)[,1:nComp])

my.model <- glm(y.train~., data=pca_x.train)

yhat_valid <- predict(my.model, newdata = data.frame(pca_x.valid))


N <- 800
Ntest <- dim(pca_x.valid)[1]
D <- nComp
y <- y.train
x <- scale(pca_x.train)
ytest <- y.valid
xtest <- scale(pca_x.valid)
lambda <- 0.5

y <- plyr::mapvalues(y, -1, 0)
ytest <- plyr::mapvalues(ytest, -1, 0)

rstan::stan_rdump(c('N','Ntest','D','y','x','ytest', 'xtest', 'lambda'),
                  file="dorothea.data.R")

N <- 800
Nt <- dim(pca_x.valid)[1]
d <- nComp
num_nodes <- 25
num_middle_layers <- 1
y <- y.train
X <- scale(pca_x.train)
```

```
yt <- y.valid
Xt <- scale(pca_x.valid)
y <- plyr::mapvalues(y, -1, 0)
yt <- plyr::mapvalues(yt, -1, 0)

rstan::stan_rdump(c('N','d', 'num_nodes', 'num_middle_layers',
                    'Nt','y','X','yt', 'Xt'),file="dorotheabnn.data.R")


N1 <- 800
N2 <- dim(pca_x.valid)[1]
D <- nComp
z1 <- y.train
x1 <- scale(pca_x.train)
z2 <- y.valid
x2 <- scale(pca_x.valid)
z1 <- plyr::mapvalues(z1, -1, 0)
z2 <- plyr::mapvalues(z2, -1, 0)

rstan::stan_rdump(c('D', 'N1', 'x1', 'z1', 'N2', 'x2', 'z2'),
                  file="dorotheagp.data.R")
```

```
In [15]: !make examples/lr/lr
```

```
--- Translating Stan model to C++ code ---
bin/stanc examples/lr/lr.stan --o=examples/lr/lr.hpp
Model name=lr_model
Input file=examples/lr/lr.stan
Output file=examples/lr/lr.hpp

--- Linking C++ model ---
g++ -I src -I stan/src -isystem stan/lib/stan_math/ -isystem stan/lib/stan_math/lib/eigen_3.2.8 -isystem
```

```
In [16]: start = time.time()
         !examples/lr/lr sample data file=examples/dorothea/dorothea.data.R output file=lr_nuts.csv
         end = time.time()

         print end-start
```

```
method = sample (Default)
  sample
    num_samples = 1000 (Default)
    num_warmup = 1000 (Default)
    save_warmup = 0 (Default)
    thin = 1 (Default)
    adapt
      engaged = 1 (Default)
      gamma = 0.050000000000000003 (Default)
      delta = 0.80000000000000004 (Default)
      kappa = 0.75 (Default)
      t0 = 10 (Default)
      init_buffer = 75 (Default)
      term_buffer = 50 (Default)
      window = 25 (Default)
    algorithm = hmc (Default)
```

```
      hmc
        engine = nuts (Default)
          nuts
            max_depth = 10 (Default)
        metric = diag_e (Default)
        stepsize = 1 (Default)
        stepsize_jitter = 0 (Default)
id = 0 (Default)
data
  file = examples/dorothea/dorothea.data.R
init = 2 (Default)
random
  seed = 1549913731
output
  file = lr_nuts.csv
  diagnostic_file =  (Default)
  refresh = 100 (Default)


Gradient evaluation took 0.000862 seconds
1000 transitions using 10 leapfrog steps per transition would take 8.62 seconds.
Adjust your expectations accordingly!


Iteration:    1 / 2000 [  0%]  (Warmup)
Iteration:  100 / 2000 [  5%]  (Warmup)
Iteration:  200 / 2000 [ 10%]  (Warmup)
Iteration:  300 / 2000 [ 15%]  (Warmup)
Iteration:  400 / 2000 [ 20%]  (Warmup)
Iteration:  500 / 2000 [ 25%]  (Warmup)
Iteration:  600 / 2000 [ 30%]  (Warmup)
Iteration:  700 / 2000 [ 35%]  (Warmup)
Iteration:  800 / 2000 [ 40%]  (Warmup)
Iteration:  900 / 2000 [ 45%]  (Warmup)
Iteration: 1000 / 2000 [ 50%]  (Warmup)
Iteration: 1001 / 2000 [ 50%]  (Sampling)
Iteration: 1100 / 2000 [ 55%]  (Sampling)
Iteration: 1200 / 2000 [ 60%]  (Sampling)
Iteration: 1300 / 2000 [ 65%]  (Sampling)
Iteration: 1400 / 2000 [ 70%]  (Sampling)
Iteration: 1500 / 2000 [ 75%]  (Sampling)
Iteration: 1600 / 2000 [ 80%]  (Sampling)
Iteration: 1700 / 2000 [ 85%]  (Sampling)
Iteration: 1800 / 2000 [ 90%]  (Sampling)
Iteration: 1900 / 2000 [ 95%]  (Sampling)
Iteration: 2000 / 2000 [100%]  (Sampling)

 Elapsed Time: 13.1903 seconds (Warm-up)
               11.4771 seconds (Sampling)
               24.6674 seconds (Total)

25.9431531429

In [17]: %%R
```

```
output_nuts <- read_stan_csv("lr_nuts.csv")



log_lik <- extract_log_lik(output_nuts) # see ?extract_log_lik


print(loo(log_lik))
print(waic(log_lik))
print(summary(extract_log_lik(output_nuts, "average_error")))
```

```
Computed from 1000 by 800 log-likelihood matrix

         Estimate   SE
elpd_loo   -170.5 14.8
p_loo        34.7  4.6
looic       341.0 29.5
Computed from 1000 by 800 log-likelihood matrix

          Estimate   SE
elpd_waic   -159.6 14.3
p_waic        23.9  2.9
waic         319.2 28.5
       V1
 Min.   :0.1581
 1st Qu.:0.2054
 Median :0.2201
 Mean   :0.2238
 3rd Qu.:0.2364
 Max.   :0.3084
```

```
In [8]: start = time.time()
        !examples/lr/lr variational data file=examples/dorothea/dorothea.data.R output file=lr_advi.csv
        end = time.time()

        print end-start
```

```
method = variational
  variational
    algorithm = meanfield (Default)
      meanfield
    iter = 10000 (Default)
    grad_samples = 1 (Default)
    elbo_samples = 100 (Default)
    eta = 1 (Default)
    adapt
      engaged = 1 (Default)
      iter = 50 (Default)
    tol_rel_obj = 0.01 (Default)
    eval_elbo = 100 (Default)
    output_samples = 1000 (Default)
id = 0 (Default)
data
```

```
  file = examples/dorothea/dorothea.data.R
init = 2 (Default)
random
  seed = 1526252890
output
  file = lr_advi.csv
  diagnostic_file =  (Default)
  refresh = 100 (Default)


This is Automatic Differentiation Variational Inference.

(EXPERIMENTAL ALGORITHM: expect frequent updates to the procedure.)

Gradient evaluation took 0.001389 seconds
1000 iterations under these settings should take 1.389 seconds.
Adjust your expectations accordingly!

Begin eta adaptation.
Iteration:   1 / 250 [  0%]  (Adaptation)
Iteration:  50 / 250 [ 20%]  (Adaptation)
Iteration: 100 / 250 [ 40%]  (Adaptation)
Iteration: 150 / 250 [ 60%]  (Adaptation)
Iteration: 200 / 250 [ 80%]  (Adaptation)
Success! Found best value [eta = 1] earlier than expected.

Begin stochastic gradient ascent.
  iter       ELBO   delta_ELBO_mean   delta_ELBO_med   notes
   100     -2e+02             1.000            1.000
   200     -2e+02             0.508            1.000
   300     -2e+02             0.341            0.016
   400     -2e+02             0.258            0.016
   500     -2e+02             0.207            0.011
   600     -2e+02             0.173            0.011
   700     -2e+02             0.148            0.008   MEDIAN ELBO CONVERGED

Drawing a sample of size 1000 from the approximate posterior...
COMPLETED.
4.73134207726
```

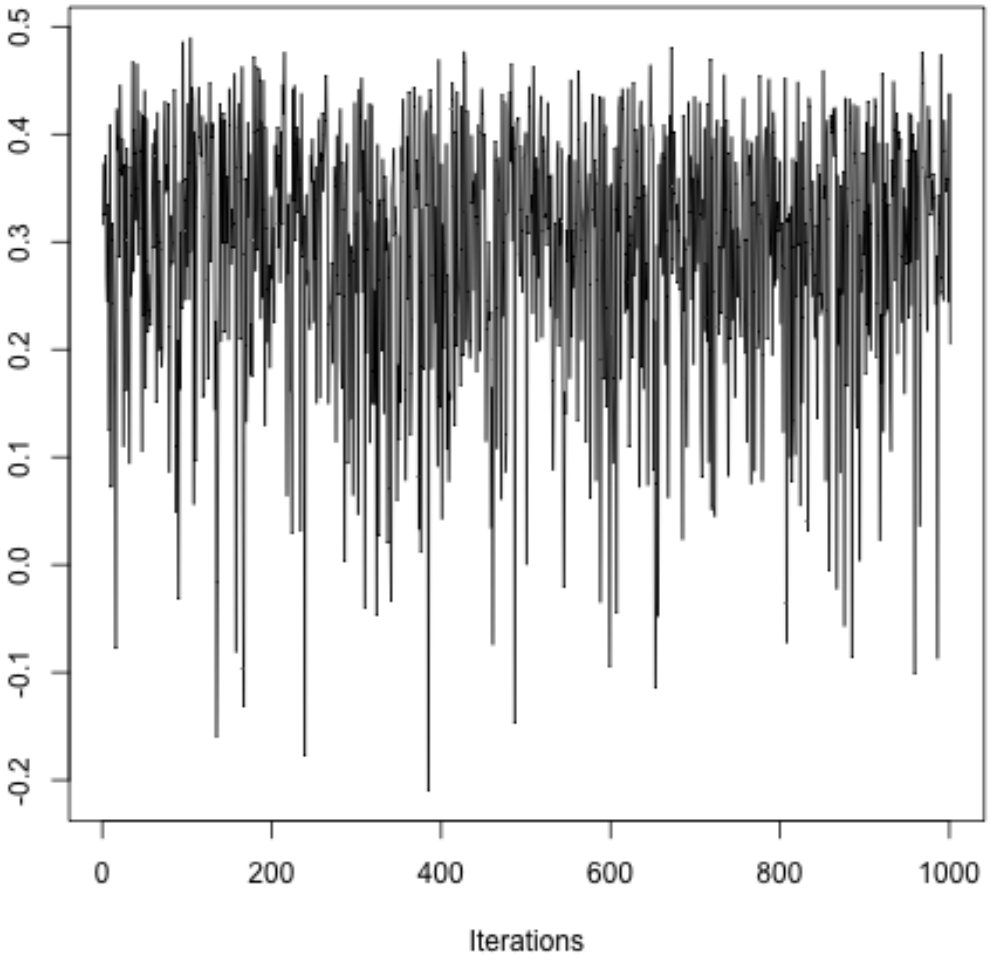
```
In [9]: %%R
        library(rstan)
        output_advi <- read_one_stan_csv("lr_advi.csv")
        #head(output_advi)
        print(summary(output_advi$average_error))
        #log_lik1 <- extract_log_lik(output_advi) # see ?extract_log_lik
        col_nb <- grep("log_lik", names(output_advi))
        log_lik <- as.matrix(output_advi[,col_nb])
        #head(as.matrix(log_lik))
        print(loo(log_lik))
        print(waic(log_lik))
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
 0.1566  0.2154  0.2301  0.2303  0.2464  0.3446
Computed from 1001 by 800 log-likelihood matrix
```

```
         Estimate   SE
elpd_loo   -224.2 19.4
p_loo        86.6 12.4
looic       448.4 38.7
Computed from 1001 by 800 log-likelihood matrix

          Estimate   SE
elpd_waic   -182.7 16.1
p_waic        45.1  6.4
waic         365.4 32.1
```

In [10]:
```
%%R
coda::traceplot(x=coda::as.mcmc(output_advi$beta.1))
```



1.1 LR-ARD

```
In [11]: !make examples/lr/lr-ard
```

```
--- Translating Stan model to C++ code ---
bin/stanc examples/lr/lr-ard.stan --o=examples/lr/lr-ard.hpp
Model name=lr_ard_model
Input file=examples/lr/lr-ard.stan
Output file=examples/lr/lr-ard.hpp

--- Linking C++ model ---
g++ -I src -I stan/src -isystem stan/lib/stan_math/ -isystem stan/lib/stan_math/lib/eigen_3.2.8 -isystem
```

```
In [12]: start = time.time()
         !examples/lr/lr-ard sample data file=examples/dorothea/dorothea.data.R output file=lr-ard_nuts
         end = time.time()

         print end-start
```

```
method = sample (Default)
  sample
    num_samples = 1000 (Default)
    num_warmup = 1000 (Default)
    save_warmup = 0 (Default)
    thin = 1 (Default)
    adapt
      engaged = 1 (Default)
      gamma = 0.050000000000000003 (Default)
      delta = 0.80000000000000004 (Default)
      kappa = 0.75 (Default)
      t0 = 10 (Default)
      init_buffer = 75 (Default)
      term_buffer = 50 (Default)
      window = 25 (Default)
    algorithm = hmc (Default)
      hmc
        engine = nuts (Default)
          nuts
            max_depth = 10 (Default)
        metric = diag_e (Default)
        stepsize = 1 (Default)
        stepsize_jitter = 0 (Default)
id = 0 (Default)
data
  file = examples/dorothea/dorothea.data.R
init = 2 (Default)
random
  seed = 1526328210
output
  file = lr-ard_nuts.csv
  diagnostic_file =  (Default)
  refresh = 100 (Default)


Gradient evaluation took 0.001034 seconds
```

```
1000 transitions using 10 leapfrog steps per transition would take 10.34 seconds.
Adjust your expectations accordingly!


Iteration:    1 / 2000 [  0%]  (Warmup)
Iteration:  100 / 2000 [  5%]  (Warmup)
Iteration:  200 / 2000 [ 10%]  (Warmup)
Iteration:  300 / 2000 [ 15%]  (Warmup)
Iteration:  400 / 2000 [ 20%]  (Warmup)
Iteration:  500 / 2000 [ 25%]  (Warmup)
Iteration:  600 / 2000 [ 30%]  (Warmup)
Iteration:  700 / 2000 [ 35%]  (Warmup)
Iteration:  800 / 2000 [ 40%]  (Warmup)
Iteration:  900 / 2000 [ 45%]  (Warmup)
Iteration: 1000 / 2000 [ 50%]  (Warmup)
Iteration: 1001 / 2000 [ 50%]  (Sampling)
Iteration: 1100 / 2000 [ 55%]  (Sampling)
Iteration: 1200 / 2000 [ 60%]  (Sampling)
Iteration: 1300 / 2000 [ 65%]  (Sampling)
Iteration: 1400 / 2000 [ 70%]  (Sampling)
Iteration: 1500 / 2000 [ 75%]  (Sampling)
Iteration: 1600 / 2000 [ 80%]  (Sampling)
Iteration: 1700 / 2000 [ 85%]  (Sampling)
Iteration: 1800 / 2000 [ 90%]  (Sampling)
Iteration: 1900 / 2000 [ 95%]  (Sampling)
Iteration: 2000 / 2000 [100%]  (Sampling)

 Elapsed Time: 42.1561 seconds (Warm-up)
               45.2249 seconds (Sampling)
               87.381 seconds (Total)

94.38083601
```

In [13]:
```
%%R
        output_nuts <- read_stan_csv("lr-ard_nuts.csv")



        log_lik <- extract_log_lik(output_nuts) # see ?extract_log_lik


        print(loo(log_lik))
        print(waic(log_lik))
        print(summary(extract_log_lik(output_nuts, "average_error")))
```

```
Computed from 1000 by 800 log-likelihood matrix

        Estimate   SE
elpd_loo   -168.1 14.8
p_loo        36.7  4.5
looic       336.3 29.7
Computed from 1000 by 800 log-likelihood matrix

          Estimate   SE
elpd_waic   -156.3 14.4
```

```
p_waic        24.9  3.0
waic         312.7 28.8
       V1
 Min.   :0.1450
 1st Qu.:0.2023
 Median :0.2170
 Mean   :0.2162
 3rd Qu.:0.2317
 Max.   :0.2921
```

```
In [14]: start = time.time()
         !examples/lr/lr-ard variational data file=examples/dorothea/dorothea.data.R output file=lr-ard_
         end = time.time()

         print end-start
```

```
method = variational
  variational
    algorithm = meanfield (Default)
      meanfield
    iter = 10000 (Default)
    grad_samples = 1 (Default)
    elbo_samples = 100 (Default)
    eta = 1 (Default)
    adapt
      engaged = 1 (Default)
      iter = 50 (Default)
    tol_rel_obj = 0.01 (Default)
    eval_elbo = 100 (Default)
    output_samples = 1000 (Default)
id = 0 (Default)
data
  file = examples/dorothea/dorothea.data.R
init = 2 (Default)
random
  seed = 1526429116
output
  file = lr-ard_advi.csv
  diagnostic_file =  (Default)
  refresh = 100 (Default)


This is Automatic Differentiation Variational Inference.

(EXPERIMENTAL ALGORITHM: expect frequent updates to the procedure.)

Gradient evaluation took 0.001776 seconds
1000 iterations under these settings should take 1.776 seconds.
Adjust your expectations accordingly!

Begin eta adaptation.
Iteration:   1 / 250 [  0%]  (Adaptation)
Iteration:  50 / 250 [ 20%]  (Adaptation)
```

```
Iteration: 100 / 250 [ 40%]  (Adaptation)
Iteration: 150 / 250 [ 60%]  (Adaptation)
Iteration: 200 / 250 [ 80%]  (Adaptation)
Success! Found best value [eta = 1] earlier than expected.

Begin stochastic gradient ascent.
  iter       ELBO   delta_ELBO_mean   delta_ELBO_med   notes
   100     -2e+02             1.000            1.000
   200     -2e+02             0.509            1.000
   300     -2e+02             0.340            0.019
   400     -2e+02             0.256            0.019
   500     -2e+02             0.207            0.011
   600     -2e+02             0.175            0.014
   700     -2e+02             0.151            0.011
   800     -2e+02             0.133            0.011
   900     -2e+02             0.119            0.009   MEDIAN ELBO CONVERGED

Drawing a sample of size 1000 from the approximate posterior...
COMPLETED.
5.50135421753
```

In [15]:
```
%%R
library(rstan)
output_advi <- read_one_stan_csv("lr-ard_advi.csv")
#head(output_advi)
print(summary(output_advi$average_error))
#log_lik1 <- extract_log_lik(output_advi) # see ?extract_log_lik
col_nb <- grep("log_lik", names(output_advi))
log_lik <- as.matrix(output_advi[,col_nb])
#head(as.matrix(log_lik))
print(loo(log_lik))
print(waic(log_lik))
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
 0.1581  0.2023  0.2185  0.2206  0.2348  0.3246
Computed from 1001 by 800 log-likelihood matrix

         Estimate   SE
elpd_loo   -259.9 23.0
p_loo       127.3 17.6
looic       519.8 46.1
Computed from 1001 by 800 log-likelihood matrix

          Estimate   SE
elpd_waic   -188.8 16.5
p_waic        56.2  7.6
waic         377.6 33.0
```

2 BNN

In [16]:
```
!cat examples/bnn/bnn.stan
```

```
In [17]: !make examples/bnn/bnn

--- Translating Stan model to C++ code ---
bin/stanc examples/bnn/bnn.stan --o=examples/bnn/bnn.hpp
Model name=bnn_model
Input file=examples/bnn/bnn.stan
Output file=examples/bnn/bnn.hpp

--- Linking C++ model ---
g++ -I src -I stan/src -isystem stan/lib/stan_math/ -isystem stan/lib/stan_math/lib/eigen_3.2.8 -isystem
```

```
In [28]: start = time.time()
         !examples/bnn/bnn sample data file=examples/dorothea/dorotheabnn.data.R output file=bnn_dorothe
         end = time.time()

         print end-start
```

```
method = sample (Default)
  sample
    num_samples = 1000 (Default)
    num_warmup = 1000 (Default)
    save_warmup = 0 (Default)
    thin = 1 (Default)
    adapt
      engaged = 1 (Default)
      gamma = 0.050000000000000003 (Default)
      delta = 0.80000000000000004 (Default)
      kappa = 0.75 (Default)
      t0 = 10 (Default)
      init_buffer = 75 (Default)
      term_buffer = 50 (Default)
      window = 25 (Default)
    algorithm = hmc (Default)
      hmc
        engine = nuts (Default)
          nuts
            max_depth = 10 (Default)
        metric = diag_e (Default)
        stepsize = 1 (Default)
        stepsize_jitter = 0 (Default)
id = 0 (Default)
data
  file = examples/dorothea/dorotheabnn.data.R
init = 2 (Default)
random
  seed = 1572715375
output
  file = bnn_dorothea_nuts.csv
  diagnostic_file =  (Default)
  refresh = 100 (Default)


Gradient evaluation took 0.021938 seconds
1000 transitions using 10 leapfrog steps per transition would take 219.38 seconds.
Adjust your expectations accordingly!
```

```
Iteration:    1 / 2000 [  0%]  (Warmup)
Iteration:  100 / 2000 [  5%]  (Warmup)
Iteration:  200 / 2000 [ 10%]  (Warmup)
Iteration:  300 / 2000 [ 15%]  (Warmup)
Iteration:  400 / 2000 [ 20%]  (Warmup)
Iteration:  500 / 2000 [ 25%]  (Warmup)
Iteration:  600 / 2000 [ 30%]  (Warmup)
Iteration:  700 / 2000 [ 35%]  (Warmup)
Iteration:  800 / 2000 [ 40%]  (Warmup)
Iteration:  900 / 2000 [ 45%]  (Warmup)
Iteration: 1000 / 2000 [ 50%]  (Warmup)
Iteration: 1001 / 2000 [ 50%]  (Sampling)
Iteration: 1100 / 2000 [ 55%]  (Sampling)
Iteration: 1200 / 2000 [ 60%]  (Sampling)
Iteration: 1300 / 2000 [ 65%]  (Sampling)
Iteration: 1400 / 2000 [ 70%]  (Sampling)
Iteration: 1500 / 2000 [ 75%]  (Sampling)
Iteration: 1600 / 2000 [ 80%]  (Sampling)
Iteration: 1700 / 2000 [ 85%]  (Sampling)
Iteration: 1800 / 2000 [ 90%]  (Sampling)
Iteration: 1900 / 2000 [ 95%]  (Sampling)
Iteration: 2000 / 2000 [100%]  (Sampling)

 Elapsed Time: 16313.4 seconds (Warm-up)
               19877.8 seconds (Sampling)
               36191.2 seconds (Total)

36325.6097789
```

In [29]:
```
%%R
output_nuts <- read_stan_csv("bnn_dorothea_nuts.csv")



log_lik <- extract_log_lik(output_nuts) # see ?extract_log_lik


print(loo(log_lik))
print(waic(log_lik))

print(summary(extract_log_lik(output_nuts, "average_error")))
```

```
Computed from 1000 by 800 log-likelihood matrix

         Estimate   SE
elpd_loo   -971.6 26.3
p_loo       961.9 26.4
looic      1943.3 52.7
Computed from 1000 by 800 log-likelihood matrix

          Estimate  SE
elpd_waic    -32.5 3.1
p_waic        22.7 2.5
```

```
waic          65.0 6.3
       V1
 Min.   :0.1540
 1st Qu.:0.2390
 Median :0.2670
 Mean   :0.2704
 3rd Qu.:0.2991
 Max.   :0.4203
```

```
In [3]: start = time.time()
        !examples/bnn/bnn variational data file=examples/dorothea/dorotheabnn.data.R output file=bnn_do
        end = time.time()

        print end-start
```

```
method = variational
  variational
    algorithm = meanfield (Default)
      meanfield
    iter = 10000 (Default)
    grad_samples = 1 (Default)
    elbo_samples = 100 (Default)
    eta = 1 (Default)
    adapt
      engaged = 1 (Default)
      iter = 50 (Default)
    tol_rel_obj = 0.01 (Default)
    eval_elbo = 100 (Default)
    output_samples = 1000 (Default)
id = 0 (Default)
data
  file = examples/dorothea/dorotheabnn.data.R
init = 2 (Default)
random
  seed = 1528744223
output
  file = bnn_dorothea_advi.csv
  diagnostic_file =  (Default)
  refresh = 100 (Default)


This is Automatic Differentiation Variational Inference.

(EXPERIMENTAL ALGORITHM: expect frequent updates to the procedure.)

Gradient evaluation took 0.039648 seconds
1000 iterations under these settings should take 39.648 seconds.
Adjust your expectations accordingly!

Begin eta adaptation.
Iteration:   1 / 250 [  0%]  (Adaptation)
Iteration:  50 / 250 [ 20%]  (Adaptation)
Iteration: 100 / 250 [ 40%]  (Adaptation)
```

```
Iteration: 150 / 250 [ 60%]  (Adaptation)
Iteration: 200 / 250 [ 80%]  (Adaptation)
Success! Found best value [eta = 1] earlier than expected.

Begin stochastic gradient ascent.
  iter       ELBO   delta_ELBO_mean   delta_ELBO_med   notes
   100     -6e+02             1.000            1.000
   200     -4e+02             0.749            1.000
   300     -4e+02             0.526            0.498
   400     -3e+02             0.412            0.498
   500     -3e+02             0.331            0.081
   600     -3e+02             0.281            0.081
   700     -3e+02             0.247            0.071
   800     -3e+02             0.218            0.071
   900     -3e+02             0.196            0.041
  1000     -3e+02             0.178            0.041
  1100     -3e+02             0.080            0.033
  1200     -3e+02             0.033            0.023
  1300     -3e+02             0.025            0.023
  1400     -3e+02             0.019            0.019
  1500     -3e+02             0.019            0.019
  1600     -3e+02             0.018            0.019
  1700     -3e+02             0.017            0.019
  1800     -3e+02             0.016            0.019
  1900     -3e+02             0.015            0.016
  2000     -3e+02             0.015            0.017
  2100     -3e+02             0.016            0.017
  2200     -3e+02             0.015            0.012
  2300     -3e+02             0.018            0.017
  2400     -3e+02             0.018            0.017
  2500     -3e+02             0.019            0.020
  2600     -3e+02             0.019            0.017
  2700     -3e+02             0.018            0.017
  2800     -3e+02             0.021            0.019
  2900     -3e+02             0.021            0.021
  3000     -3e+02             0.021            0.021
  3100     -3e+02             0.018            0.019
  3200     -3e+02             0.020            0.021
  3300     -3e+02             0.017            0.019
  3400     -3e+02             0.017            0.019
  3500     -3e+02             0.016            0.012
  3600     -3e+02             0.018            0.019
  3700     -3e+02             0.018            0.021
  3800     -3e+02             0.016            0.012
  3900     -3e+02             0.015            0.012
  4000     -3e+02             0.015            0.012
  4100     -3e+02             0.017            0.015
  4200     -3e+02             0.016            0.012
  4300     -3e+02             0.016            0.012
  4400     -3e+02             0.016            0.014
  4500     -3e+02             0.017            0.015
  4600     -3e+02             0.018            0.015
  4700     -3e+02             0.016            0.014
  4800     -3e+02             0.016            0.014
```

```
  4900     -3e+02             0.017            0.014
  5000     -3e+02             0.018            0.018
  5100     -3e+02             0.020            0.018
  5200     -3e+02             0.024            0.020
  5300     -3e+02             0.027            0.024
  5400     -3e+02             0.027            0.024
  5500     -3e+02             0.026            0.024
  5600     -3e+02             0.023            0.020
  5700     -3e+02             0.023            0.020
  5800     -3e+02             0.023            0.020
  5900     -3e+02             0.021            0.018
  6000     -3e+02             0.022            0.020
  6100     -3e+02             0.019            0.016
  6200     -3e+02             0.015            0.012
  6300     -3e+02             0.013            0.012
  6400     -3e+02             0.012            0.012
  6500     -3e+02             0.016            0.012
  6600     -3e+02             0.018            0.016
  6700     -3e+02             0.019            0.019
  6800     -3e+02             0.018            0.019
  6900     -3e+02             0.020            0.019
  7000     -3e+02             0.018            0.018
  7100     -3e+02             0.018            0.018
  7200     -3e+02             0.018            0.018
  7300     -3e+02             0.018            0.018
  7400     -3e+02             0.020            0.019
  7500     -3e+02             0.020            0.019
  7600     -3e+02             0.020            0.019
  7700     -3e+02             0.020            0.018
  7800     -3e+02             0.019            0.018
  7900     -3e+02             0.018            0.014
  8000     -3e+02             0.016            0.014
  8100     -3e+02             0.016            0.014
  8200     -3e+02             0.015            0.013
  8300     -3e+02             0.014            0.009   MEDIAN ELBO CONVERGED

Drawing a sample of size 1000 from the approximate posterior...
COMPLETED.
359.390032053
```

In [4]:
```
%%R
        output_advi <- read_one_stan_csv("bnn_dorothea_advi.csv")
        #head(output_advi)
        col_nb <- grep("log_lik", names(output_advi))
        log_lik <- as.matrix(output_advi[,col_nb])
        #head(as.matrix(log_lik))
        print(loo(log_lik))
        print(waic(log_lik))
        print(summary(output_advi$average_error))
```

```
Computed from 1001 by 800 log-likelihood matrix

         Estimate   SE
elpd_loo   -264.3 19.6
p_loo         9.1  0.9
```

```
looic       528.7 39.1

All Pareto k estimates OK (k < 0.5)
Computed from 1001 by 800 log-likelihood matrix

          Estimate   SE
elpd_waic   -264.4 19.6
p_waic         9.1  0.9
waic         528.7 39.1
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
    0.5     0.5     0.5     0.5     0.5     0.5
```

2.1 BNN-ARD

```
In [ ]: !make examples/bnn/bnn-ard
```

```
In [30]: start = time.time()
         !examples/bnn/bnn-ard sample data file=examples/dorothea/dorotheabnn.data.R output file=bnn-ar
         end = time.time()

         print end-start
```

```
method = sample (Default)
  sample
    num_samples = 1000 (Default)
    num_warmup = 1000 (Default)
    save_warmup = 0 (Default)
    thin = 1 (Default)
    adapt
      engaged = 1 (Default)
      gamma = 0.050000000000000003 (Default)
      delta = 0.80000000000000004 (Default)
      kappa = 0.75 (Default)
      t0 = 10 (Default)
      init_buffer = 75 (Default)
      term_buffer = 50 (Default)
      window = 25 (Default)
    algorithm = hmc (Default)
      hmc
        engine = nuts (Default)
          nuts
            max_depth = 10 (Default)
        metric = diag_e (Default)
        stepsize = 1 (Default)
        stepsize_jitter = 0 (Default)
id = 0 (Default)
data
  file = examples/dorothea/dorotheabnn.data.R
init = 2 (Default)
random
  seed = 1609051876
output
  file = bnn-ard_dorothea_nuts.csv
```

```
  diagnostic_file =  (Default)
  refresh = 100 (Default)


Gradient evaluation took 2.06366 seconds
1000 transitions using 10 leapfrog steps per transition would take 20636.6 seconds.
Adjust your expectations accordingly!



Informational Message: The current Metropolis proposal is about to be rejected because of the following
Exception thrown at line 97: stan::math::multi_normal_log: LDLT_Factor of covariance parameter is not po
If this warning occurs sporadically, such as for highly constrained variable types like covariance matri

but if this warning occurs often then your model may be either severely ill-conditioned or misspecified

Informational Message: The current Metropolis proposal is about to be rejected because of the following
Exception thrown at line 97: stan::math::multi_normal_log: LDLT_Factor of covariance parameter is not po
If this warning occurs sporadically, such as for highly constrained variable types like covariance matri

but if this warning occurs often then your model may be either severely ill-conditioned or misspecified
Iteration:    1 / 2000 [  0%]  (Warmup)

Informational Message: The current Metropolis proposal is about to be rejected because of the following
Exception thrown at line 97: stan::math::multi_normal_log: LDLT_Factor of covariance parameter is not po
If this warning occurs sporadically, such as for highly constrained variable types like covariance matri

but if this warning occurs often then your model may be either severely ill-conditioned or misspecified

Informational Message: The current Metropolis proposal is about to be rejected because of the following
Exception thrown at line 97: stan::math::multi_normal_log: LDLT_Factor of covariance parameter is not po
If this warning occurs sporadically, such as for highly constrained variable types like covariance matri

but if this warning occurs often then your model may be either severely ill-conditioned or misspecified

Informational Message: The current Metropolis proposal is about to be rejected because of the following
Exception thrown at line 97: stan::math::multi_normal_log: LDLT_Factor of covariance parameter is not po
If this warning occurs sporadically, such as for highly constrained variable types like covariance matri

but if this warning occurs often then your model may be either severely ill-conditioned or misspecified

Informational Message: The current Metropolis proposal is about to be rejected because of the following
Exception thrown at line 97: stan::math::multi_normal_log: LDLT_Factor of covariance parameter is not po
If this warning occurs sporadically, such as for highly constrained variable types like covariance matri

but if this warning occurs often then your model may be either severely ill-conditioned or misspecified
^C
17964.8243148
```

In [31]:
```
%%R
output_nuts <- read_stan_csv("bnn-ard_dorothea_nuts.csv")
```

```
        log_lik <- extract_log_lik(output_nuts) # see ?extract_log_lik


        print(loo(log_lik))
        print(waic(log_lik))

        print(summary(extract_log_lik(output_nuts, "average_error")))
```

```
Error in extract_log_lik(output_nuts) : Please load the 'rstan' package.

/usr/local/lib/python2.7/site-packages/rpy2/robjects/functions.py:106: UserWarning: Error in extract_log

  res = super(Function, self).__call__(*new_args, **new_kwargs)
/usr/local/lib/python2.7/site-packages/rpy2/robjects/functions.py:106: UserWarning: There were 11 warnin
  res = super(Function, self).__call__(*new_args, **new_kwargs)
/usr/local/lib/python2.7/site-packages/rpy2/robjects/functions.py:106: UserWarning:

  res = super(Function, self).__call__(*new_args, **new_kwargs)
```

```
In [26]: start = time.time()
         !examples/bnn/bnn-ard variational data file=examples/dorothea/dorotheabnn.data.R output file=bn
         end = time.time()

         print end-start
```

```
method = variational
  variational
    algorithm = meanfield (Default)
      meanfield
    iter = 10000 (Default)
    grad_samples = 1 (Default)
    elbo_samples = 100 (Default)
    eta = 1 (Default)
    adapt
      engaged = 1 (Default)
      iter = 50 (Default)
    tol_rel_obj = 0.01 (Default)
    eval_elbo = 100 (Default)
    output_samples = 1000 (Default)
id = 0 (Default)
data
  file = examples/dorothea/dorotheabnn.data.R
init = 2 (Default)
random
  seed = 1565181953
output
  file = bnn-ard_dorothea_advi.csv
  diagnostic_file =  (Default)
  refresh = 100 (Default)


This is Automatic Differentiation Variational Inference.

(EXPERIMENTAL ALGORITHM: expect frequent updates to the procedure.)
```

```
Gradient evaluation took 2.04371 seconds
1000 iterations under these settings should take 2043.71 seconds.
Adjust your expectations accordingly!

Begin eta adaptation.
Iteration:   1 / 250 [  0%]  (Adaptation)
Iteration:  50 / 250 [ 20%]  (Adaptation)
Iteration: 100 / 250 [ 40%]  (Adaptation)
Iteration: 150 / 250 [ 60%]  (Adaptation)
Iteration: 200 / 250 [ 80%]  (Adaptation)
Iteration: 250 / 250 [100%]  (Adaptation)
Success! Found best value [eta = 0.1].

Begin stochastic gradient ascent.
  iter       ELBO   delta_ELBO_mean   delta_ELBO_med   notes
   100     -3e+03             1.000            1.000
   200     -2e+03             0.948            1.000
   300     -1e+03             0.759            0.897
   400     -1e+03             0.618            0.897
   500     -9e+02             0.528            0.380
   600     -8e+02             0.457            0.380
   700     -8e+02             0.403            0.194
   800     -7e+02             0.361            0.194
   900     -7e+02             0.326            0.170
  1000     -7e+02             0.297            0.170
  1100     -7e+02             0.199            0.099
  1200     -6e+02             0.112            0.081
  1300     -6e+02             0.077            0.065
  1400     -6e+02             0.058            0.048
  1500     -6e+02             0.042            0.038
  1600     -6e+02             0.033            0.030
  1700     -6e+02             0.025            0.022
  1800     -6e+02             0.020            0.021
  1900     -6e+02             0.017            0.016
  2000     -6e+02             0.013            0.016
  2100     -6e+02             0.013            0.016
  2200     -6e+02             0.012            0.012
  2300     -6e+02             0.009            0.010   MEAN ELBO CONVERGED

Drawing a sample of size 1000 from the approximate posterior...
COMPLETED.
7523.39317584
```

In [27]:
```
%%R
output_advi <- read_one_stan_csv("bnn-ard_dorothea_advi.csv")
#head(output_advi)
col_nb <- grep("log_lik", names(output_advi))
log_lik <- as.matrix(output_advi[,col_nb])
#head(as.matrix(log_lik))
print(loo(log_lik))
print(waic(log_lik))
print(summary(output_advi$average_error))
```

```
Computed from 1001 by 800 log-likelihood matrix
```

```
         Estimate   SE
elpd_loo   -251.2 19.9
p_loo        82.5  7.2
looic       502.3 39.8
Computed from 1001 by 800 log-likelihood matrix

          Estimate   SE
elpd_waic   -249.4 20.0
p_waic        80.8  7.3
waic         498.9 40.1
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
 0.1581  0.2995  0.3493  0.3542  0.4093  0.5201
```

3 GP

```
In [23]: !make examples/gp/gp
```

```
--- Translating Stan model to C++ code ---
bin/stanc examples/gp/gp.stan --o=examples/gp/gp.hpp
Model name=gp_model
Input file=examples/gp/gp.stan
Output file=examples/gp/gp.hpp
DIAGNOSTIC(S) FROM PARSER:
Warning (non-fatal):
Left-hand side of sampling statement (~) may contain a non-linear transform of a parameter or local vari
If so, you need to call increment_log_prob() with the log absolute determinant of the Jacobian of the tr
Left-hand-side of sampling statement:
    y ~ multi_normal(...)


--- Linking C++ model ---
g++ -I src -I stan/src -isystem stan/lib/stan_math/ -isystem stan/lib/stan_math/lib/eigen_3.2.8 -isystem
```

```
In [36]: start = time.time()
         !examples/gp/gp sample data file=examples/dorothea/dorotheagp.data.R output file=gp_dorothea_nu
         end = time.time()

         print end-start
```

```
method = sample (Default)
  sample
    num_samples = 1000 (Default)
    num_warmup = 1000 (Default)
    save_warmup = 0 (Default)
    thin = 1 (Default)
    adapt
      engaged = 1 (Default)
      gamma = 0.050000000000000003 (Default)
      delta = 0.80000000000000004 (Default)
      kappa = 0.75 (Default)
      t0 = 10 (Default)
      init_buffer = 75 (Default)
```

```
      term_buffer = 50 (Default)
      window = 25 (Default)
    algorithm = hmc (Default)
      hmc
        engine = nuts (Default)
          nuts
            max_depth = 10 (Default)
        metric = diag_e (Default)
        stepsize = 1 (Default)
        stepsize_jitter = 0 (Default)
id = 0 (Default)
data
  file = examples/dorothea/dorotheagp.data.R
init = 2 (Default)
random
  seed = 1627080563
output
  file = gp_dorothea_nuts.csv
  diagnostic_file =  (Default)
  refresh = 100 (Default)


Gradient evaluation took 1.50259 seconds
1000 transitions using 10 leapfrog steps per transition would take 15025.9 seconds.
Adjust your expectations accordingly!


Iteration:    1 / 2000 [  0%]  (Warmup)

Informational Message: The current Metropolis proposal is about to be rejected because of the following
Exception thrown at line 79: stan::math::multi_normal_log: Covariance matrix is not symmetric. Covarianc
If this warning occurs sporadically, such as for highly constrained variable types like covariance matri

but if this warning occurs often then your model may be either severely ill-conditioned or misspecified

Informational Message: The current Metropolis proposal is about to be rejected because of the following
Exception thrown at line 79: stan::math::multi_normal_log: LDLT_Factor of covariance parameter is not po
If this warning occurs sporadically, such as for highly constrained variable types like covariance matri

but if this warning occurs often then your model may be either severely ill-conditioned or misspecified
^C
16914.7633269
```

In [33]:
```
%%R
output_nuts <- read_stan_csv("gp_dorothea_nuts.csv")



log_lik <- extract_log_lik(output_nuts) # see ?extract_log_lik


print(loo(log_lik))
print(waic(log_lik))
```

```
        print(summary(extract_log_lik(output_nuts, "average_error")))
```

```
Error in extract_log_lik(output_nuts) : Please load the 'rstan' package.
```

```
/usr/local/lib/python2.7/site-packages/rpy2/robjects/functions.py:106: UserWarning: 1:
  res = super(Function, self).__call__(*new_args, **new_kwargs)
/usr/local/lib/python2.7/site-packages/rpy2/robjects/functions.py:106: UserWarning: In FUN(X[[i]], ...)
  res = super(Function, self).__call__(*new_args, **new_kwargs)
/usr/local/lib/python2.7/site-packages/rpy2/robjects/functions.py:106: UserWarning:  line with "Elapsed

  res = super(Function, self).__call__(*new_args, **new_kwargs)
/usr/local/lib/python2.7/site-packages/rpy2/robjects/functions.py:106: UserWarning: 2:
  res = super(Function, self).__call__(*new_args, **new_kwargs)
/usr/local/lib/python2.7/site-packages/rpy2/robjects/functions.py:106: UserWarning: In read_stan_csv("gp
  res = super(Function, self).__call__(*new_args, **new_kwargs)
/usr/local/lib/python2.7/site-packages/rpy2/robjects/functions.py:106: UserWarning:

  res = super(Function, self).__call__(*new_args, **new_kwargs)
/usr/local/lib/python2.7/site-packages/rpy2/robjects/functions.py:106: UserWarning:  the number of itera

  res = super(Function, self).__call__(*new_args, **new_kwargs)
```

```
In [24]: start = time.time()
         !examples/gp/gp variational data file=examples/dorothea/dorotheagp.data.R output file=gp_dorotl
         end = time.time()

         print end-start
```

```
method = variational
  variational
    algorithm = meanfield (Default)
      meanfield
    iter = 10000 (Default)
    grad_samples = 1 (Default)
    elbo_samples = 100 (Default)
    eta = 1 (Default)
    adapt
      engaged = 1 (Default)
      iter = 50 (Default)
    tol_rel_obj = 0.01 (Default)
    eval_elbo = 100 (Default)
    output_samples = 1000 (Default)
id = 0 (Default)
data
  file = examples/dorothea/dorotheagp.data.R
init = 2 (Default)
random
  seed = 1562823704
output
  file = gp_dorothea_advi.csv
  diagnostic_file =  (Default)
  refresh = 100 (Default)


This is Automatic Differentiation Variational Inference.
```

```
(EXPERIMENTAL ALGORITHM: expect frequent updates to the procedure.)

Gradient evaluation took 1.62711 seconds
1000 iterations under these settings should take 1627.11 seconds.
Adjust your expectations accordingly!

Begin eta adaptation.
Iteration:   1 / 250 [  0%]  (Adaptation)
Iteration:  50 / 250 [ 20%]  (Adaptation)
Iteration: 100 / 250 [ 40%]  (Adaptation)
Iteration: 150 / 250 [ 60%]  (Adaptation)
Iteration: 200 / 250 [ 80%]  (Adaptation)
Success! Found best value [eta = 1] earlier than expected.

Begin stochastic gradient ascent.
  iter       ELBO   delta_ELBO_mean   delta_ELBO_med   notes
   100     -4e+02             1.000            1.000
   200     -4e+02             0.563            1.000
   300     -3e+02             0.399            0.127
   400     -3e+02             0.300            0.127
   500     -3e+02             0.244            0.070
   600     -3e+02             0.211            0.070
   700     -3e+02             0.182            0.044
   800     -3e+02             0.162            0.044
   900     -3e+02             0.145            0.022
  1000     -3e+02             0.131            0.022
  1100     -3e+02             0.031            0.020
  1200     -3e+02             0.019            0.014
  1300     -3e+02             0.012            0.009   MEDIAN ELBO CONVERGED

Drawing a sample of size 1000 from the approximate posterior...
COMPLETED.
2354.44530821
```

In [25]:
```
%%R
        output_advi <- read_one_stan_csv("gp_dorothea_advi.csv")
        #head(output_advi)
        col_nb <- grep("log_lik", names(output_advi))
        log_lik <- as.matrix(output_advi[,col_nb])
        #head(as.matrix(log_lik))
        print(loo(log_lik))
        print(waic(log_lik))
        print(summary(output_advi$average_error))
```

```
Computed from 1001 by 800 log-likelihood matrix

         Estimate  SE
elpd_loo   -585.3 3.2
p_loo         8.0 0.2
looic      1170.7 6.3
Computed from 1001 by 800 log-likelihood matrix

          Estimate  SE
elpd_waic   -585.2 3.2
```

```
p_waic         7.9 0.2
waic        1170.4 6.3
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
 0.2597  0.3951  0.4241  0.4242  0.4539  0.5670
```

3.1 GP-ARD

```
In [13]: !make examples/gp/gp-ard
```

```
--- Translating Stan model to C++ code ---
bin/stanc examples/gp/gp-ard.stan --o=examples/gp/gp-ard.hpp
Model name=gp_ard_model
Input file=examples/gp/gp-ard.stan
Output file=examples/gp/gp-ard.hpp
DIAGNOSTIC(S) FROM PARSER:
Warning (non-fatal):
Left-hand side of sampling statement (~) may contain a non-linear transform of a parameter or local vari
If so, you need to call increment_log_prob() with the log absolute determinant of the Jacobian of the tr
Left-hand-side of sampling statement:
    y ~ multi_normal(...)


--- Linking C++ model ---
g++ -I src -I stan/src -isystem stan/lib/stan_math/ -isystem stan/lib/stan_math/lib/eigen_3.2.8 -isystem
```

```
In [ ]: start = time.time()
        !examples/gp/gp-ard sample data file=examples/dorothea/dorotheagp.data.R output file=gp-ard_doro
        end = time.time()

        print end-start
```

```
method = sample (Default)
  sample
    num_samples = 1000 (Default)
    num_warmup = 1000 (Default)
    save_warmup = 0 (Default)
    thin = 1 (Default)
    adapt
      engaged = 1 (Default)
      gamma = 0.050000000000000003 (Default)
      delta = 0.80000000000000004 (Default)
      kappa = 0.75 (Default)
      t0 = 10 (Default)
      init_buffer = 75 (Default)
      term_buffer = 50 (Default)
      window = 25 (Default)
    algorithm = hmc (Default)
      hmc
        engine = nuts (Default)
          nuts
            max_depth = 10 (Default)
        metric = diag_e (Default)
        stepsize = 1 (Default)
```

```
        stepsize_jitter = 0 (Default)
id = 0 (Default)
data
  file = examples/dorothea/dorotheagp.data.R
init = 2 (Default)
random
  seed = 1643995651
output
  file = gp-ard_dorothea_nuts.csv
  diagnostic_file =  (Default)
  refresh = 100 (Default)


Gradient evaluation took 3.77519 seconds
1000 transitions using 10 leapfrog steps per transition would take 37751.8 seconds.
Adjust your expectations accordingly!


Iteration:    1 / 2000 [  0%]  (Warmup)

Informational Message: The current Metropolis proposal is about to be rejected because of the following
Exception thrown at line 91: stan::math::multi_normal_log: Covariance matrix is not symmetric. Covarianc
If this warning occurs sporadically, such as for highly constrained variable types like covariance matri

but if this warning occurs often then your model may be either severely ill-conditioned or misspecified
```

In [35]:
```
%%R
        output_nuts <- read_stan_csv("gp-ard_dorothea_nuts.csv")



        log_lik <- extract_log_lik(output_nuts) # see ?extract_log_lik


        print(loo(log_lik))
        print(waic(log_lik))

        print(summary(extract_log_lik(output_nuts, "average_error")))
```

```
Error in strsplit(header, ",")[[1]] : subscript out of bounds

/usr/local/lib/python2.7/site-packages/rpy2/robjects/functions.py:106: UserWarning: Error in strsplit(he

  res = super(Function, self).__call__(*new_args, **new_kwargs)
/usr/local/lib/python2.7/site-packages/rpy2/robjects/functions.py:106: UserWarning: Warning message:

  res = super(Function, self).__call__(*new_args, **new_kwargs)
```

In [18]:
```
start = time.time()
!examples/gp/gp-ard variational data file=examples/dorothea/dorotheagp.data.R output file=gp-ar
end = time.time()

print end-start
```

```
method = variational
  variational
```

```
    algorithm = meanfield (Default)
      meanfield
    iter = 10000 (Default)
    grad_samples = 1 (Default)
    elbo_samples = 100 (Default)
    eta = 1 (Default)
    adapt
      engaged = 1 (Default)
      iter = 50 (Default)
    tol_rel_obj = 0.01 (Default)
    eval_elbo = 100 (Default)
    output_samples = 1000 (Default)
id = 0 (Default)
data
  file = examples/dorothea/dorotheagp.data.R
init = 2 (Default)
random
  seed = 1550027225
output
  file = gp_dorothea_advi.csv
  diagnostic_file =  (Default)
  refresh = 100 (Default)


This is Automatic Differentiation Variational Inference.

(EXPERIMENTAL ALGORITHM: expect frequent updates to the procedure.)

Gradient evaluation took 4.16308 seconds
1000 iterations under these settings should take 4163.08 seconds.
Adjust your expectations accordingly!

Begin eta adaptation.
Iteration:   1 / 250 [  0%]  (Adaptation)
Iteration:  50 / 250 [ 20%]  (Adaptation)
Iteration: 100 / 250 [ 40%]  (Adaptation)
Iteration: 150 / 250 [ 60%]  (Adaptation)
Iteration: 200 / 250 [ 80%]  (Adaptation)
Success! Found best value [eta = 1] earlier than expected.

Begin stochastic gradient ascent.
  iter       ELBO   delta_ELBO_mean   delta_ELBO_med   notes
   100     -4e+02             1.000            1.000
   200     -4e+02             0.541            1.000
   300     -4e+02             0.371            0.081
   400     -4e+02             0.288            0.081
   500     -4e+02             0.232            0.037
   600     -4e+02             0.195            0.037
   700     -4e+02             0.168            0.033
   800     -4e+02             0.147            0.033
   900     -4e+02             0.132            0.012
  1000     -4e+02             0.119            0.012
  1100     -4e+02             0.020            0.010
  1200     -4e+02             0.013            0.010
```

```
  1300     -4e+02             0.011            0.008   MEDIAN ELBO CONVERGED

Drawing a sample of size 1000 from the approximate posterior...
COMPLETED.
8835.93932295
```

In [20]:
```
%%R
output_advi <- read_one_stan_csv("gp-ard_dorothea_advi.csv")
#head(output_advi)
col_nb <- grep("log_lik", names(output_advi))
log_lik <- as.matrix(output_advi[,col_nb])
#head(as.matrix(log_lik))
print(loo(log_lik))
print(waic(log_lik))
print(summary(output_advi$average_error))
```

```
Computed from 1001 by 800 log-likelihood matrix

         Estimate  SE
elpd_loo   -592.2 3.3
p_loo         9.9 0.2
looic      1184.3 6.6
Computed from 1001 by 800 log-likelihood matrix

          Estimate  SE
elpd_waic   -592.0 3.3
p_waic         9.7 0.2
waic        1184.1 6.6
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
 0.3066  0.4536  0.4826  0.4835  0.5152  0.6430
```

In [ ]:

GPy

May 1, 2016

In [68]:
```
import GPy
from sklearn.decomposition import PCA
import numpy as np
import pandas as pd
import time
import pdb
```

In [103]:
```
y = pd.read_csv('dorothea/y.csv')
y = y.as_matrix()
y = y.reshape((y.shape[0],1))
yt = pd.read_csv('dorothea/ytest.csv')
yt = yt.as_matrix()
yt = yt.reshape((yt.shape[0],1))
x = pd.read_csv('dorothea/x.csv')
xt = pd.read_csv('dorothea/xtest.csv')
```

In [104]:
```
def BER(y, yhat):
    TN=0
    FN=0
    TP=0
    FP=0

    for i in range(0, y.shape[0]):
        if(np.round(yhat[0][i]) == 0 and y[i] == 0):
            TN = TN + 1.0
        elif(np.round(yhat[0][i]) == 0 and y[i] == 1):
            FN = FN + 1.0
        elif(np.round(yhat[0][i]) == 1 and y[i] == 1):
            TP = TP + 1.0
        elif(np.round(yhat[0][i]) == 1 and y[i] == 0):
            FP = FP + 1.0
        else:
            pdb.set_trace()
    BER = 0.5 * (FP/(FP+TN) + FN/(FN+TP))
    print BER
```

In [105]:
```
k = GPy.kern.RBF(X_train_pca.shape[1], ARD=False) + GPy.kern.White(X_train_pca.shape[1])
m = GPy.models.GPClassification(x, y, kernel=k)

start = time.time()
m.optimize()
end = time.time()
print(end - start)
y_test_pred = m.predict(xt.as_matrix())
BER(yt, y_test_pred)
```

```
34.4837048054
0.259493670886
```

In [106]:
```
k_ard = GPy.kern.RBF(X_train_pca.shape[1], ARD=True) + GPy.kern.White(X_train_pca.shape[1])
m_ard = GPy.models.GPClassification(x, y, kernel=k_ard)

start = time.time()
m_ard.optimize()
end = time.time()
print(end - start)
y_test_pred_ard = m_ard.predict(xt.as_matrix())
BER(yt, y_test_pred_ard)
```

```
41.8705608845
0.185964259121
```

In [ ]: