greta demo

Bruce Campbell

5/7/2018

https://rviews.rstudio.com/2018/04/23/on-first-meeting-greta/

greta lets users write TensorFlow-based Bayesian models directly in R.

We demo greta with a model used by Richard McElreath in section 8.3 of Statistical Rethinking: A Bayesian Course with Examples in R and Stan. This model seeks to explain the log of a country's GDP based on a measure of terrain ruggedness while controlling for whether or not the country is in Africa. I am going to use it just to illustrate MCMC sampling with greta. The extended example in McElreath's book, however, is a meditation on the subtleties of modeling interactions, and is well worth studying.

DiagrammeR is for plotting the TensorFlow flow diagram of the model, and bayesplot is used to plot trace diagrams of the Markov chains.

The rugged data set which provides 52 variables for 234 is fairly interesting, but we will use a trimmed-down data set with only 170 counties and three variables.

```
# install.packages(c("coda","mvtnorm","devtools","loo"))
# library(devtools)
# devtools::install_github("rmcelreath/rethinking")
```

```
library(rethinking)
```

```
## Loading required package: rstan
```

```
## Loading required package: ggplot2
```

```
## Loading required package: StanHeaders
```

```
## rstan (Version 2.17.3, GitRev: 2e1f913d3ca3)
```

```
## For execution on a local, multicore CPU with excess RAM we recommend calling
## options(mc.cores = parallel::detectCores()).
## To avoid recompilation of unchanged Stan programs, we recommend calling
## rstan_options(auto_write = TRUE)
```

```
## Loading required package: parallel
```

```
## rethinking (Version 1.59)
```

```
library(greta)
```

```
##
## Attaching package: 'greta'
```

```
## The following object is masked from 'package:rethinking':
##
##     logistic
```

```
## The following objects are masked from 'package:stats':
##
##     binomial, poisson
```

```
## The following objects are masked from 'package:base':
##
##     %*%, backsolve, beta, colMeans, colSums, diag, forwardsolve,
```

```
##     gamma, rowMeans, rowSums, sweep
```

```
#library(DiagrammeR)
library(bayesplot)
```

```
## This is bayesplot version 1.5.0
```

```
## - Plotting theme set to bayesplot::theme_default()
```

```
## - Online documentation at mc-stan.org/bayesplot
```

```
library(ggplot2)

# Example from section 8.3 Statistical Rethinking
data(rugged)
d <- rugged
d$log_gdp <- log(d$rgdppc_2000)
dd <- d[complete.cases(d$rgdppc_2000), ]
dd_trim <- dd[ , c("log_gdp","rugged","cont_africa")]
head(dd_trim)
```

```
##     log_gdp rugged cont_africa
## 3  7.492609  0.858           1
## 5  8.216929  3.427           0
## 8  9.933263  0.769           0
## 9  9.407032  0.775           0
## 10 7.792343  2.688           0
## 12 9.212541  0.006           0
```

```
set.seed(1234)
```

In this section of code, we set up the TensorFlow data structures. The first step is to move the data into greta arrays. These data structures behave similarly to R arrays in that they can be manipulated with functions. However, greta doesn't immediately calculate values for new arrays. It works out the size and shape of the result and creates a place-holder data structure.

```
#data
g_log_gdp <- as_data(dd_trim$log_gdp)
g_rugged <- as_data(dd_trim$rugged)
g_cont_africa <- as_data(dd_trim$cont_africa)
```

In this section, we set up the Bayesian model. All parameters need prior probability distributions. Note that the parameters a, bR, bA, bAR, sigma, and mu are all new greta arrays that don't contain any data. a is 1 x 1 array and mu is a 170 x 1 array with one slot for each observation.

The distribution() function sets up the likelihood function for the model.

```
# Variables and Priors

a <- normal(0, 100)
bR <- normal(0, 10)
bA <- normal(0, 10)
bAR <- normal(0,10)
sigma <- cauchy(0,2,truncation=c(0,Inf))
```

```
a
```

```
## greta array (variable following a normal distribution)
##
##      [,1]
```

```
## [1,]  ?
```

```
mu <- a + bR*g_rugged + bA*g_cont_africa + bAR*g_rugged*g_cont_africa

dim(mu)
```

```
## [1] 170   1
```

```
distribution(g_log_gdp) = normal(mu, sigma)
```

The model() function does all of the work. It fits the model and produces a fairly complicated object organized as three lists that contain, respectively, the R6 class, TensorFlow structures, and the various greta data arrays.

```
# defining the model
mod <- model(a,bR,bA,bAR,sigma)

str(mod,give.attr=FALSE,max.level=1)
```

```
## List of 3
##  $ dag                 :Classes 'dag_class', 'R6' <dag_class>
##   Public:
##     adjacency_matrix: function ()
##     build_dag: function (greta_array_list)
##     clone: function (deep = FALSE)
##     compile: TRUE
##     define_gradients: function ()
##     define_joint_density: function ()
##     define_tf: function ()
##     example_parameters: function (flat = TRUE)
##     find_node_neighbours: function ()
##     get_tf_names: function (types = NULL)
##     gradients: function (adjusted = TRUE)
##     initialize: function (target_greta_arrays, tf_float = tf$float32, n_cores = 2L,
##     log_density: function (adjusted = TRUE)
##     make_names: function ()
##     n_cores: 16
##     node_list: list
##     node_tf_names: variable_1 distribution_1 data_1 data_2 operation_1 oper ...
##     node_types: variable distribution data data operation operation oper ...
##     parameters_example: list
##     send_parameters: function (parameters, flat = TRUE)
##     subgraph_membership: function ()
##     target_nodes: list
##     tf_environment: environment
##     tf_float: tensorflow.python.framework.dtypes.DType, python.builtin.object
##     tf_name: function (node)
##     trace_values: function ()
##  $ target_greta_arrays :List of 5
##  $ visible_greta_arrays:List of 9
```

Plotting mod produces the TensorFlow flow diagram that shows the structure of the underlying TensorFlow model, which is simple for this model and easily interpretable.

```
# plotting
#plot(mod)
```

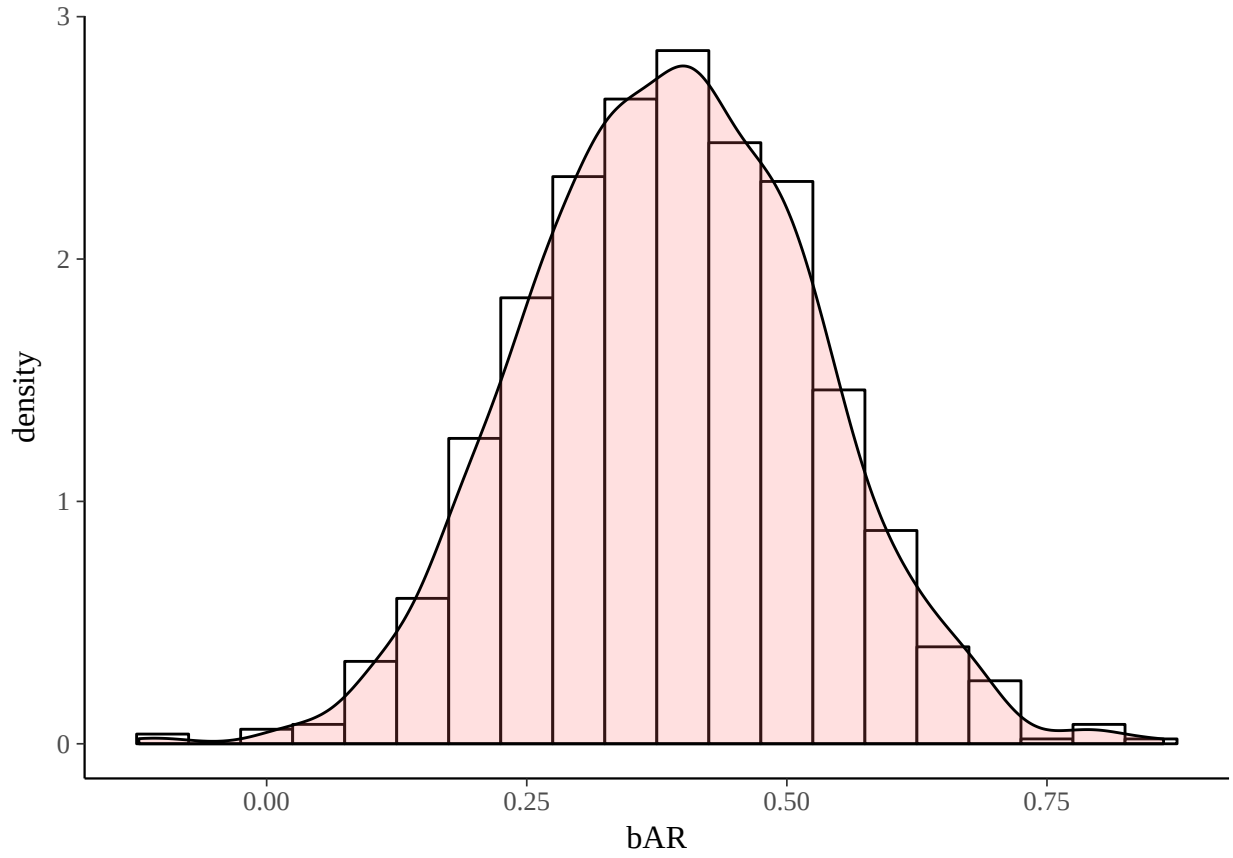
Next, we use the greta function mcmc() to sample from the posterior distributions defined in the model.

```r
# sampling
draws <- mcmc(mod, n_samples = 1000)
summary(draws)
```

```
##
## Iterations = 1:1000
## Thinning interval = 1
## Number of chains = 1
## Sample size per chain = 1000
##
## 1. Empirical mean and standard deviation for each variable,
##    plus standard error of the mean:
##
##          Mean      SD Naive SE Time-series SE
## a      9.2157 0.14588 0.004613       0.004722
## bR    -0.2006 0.07837 0.002478       0.002745
## bA    -1.9399 0.25014 0.007910       0.004723
## bAR    0.3884 0.13638 0.004313       0.003697
## sigma  0.9497 0.05191 0.001642       0.001725
##
## 2. Quantiles for each variable:
##
##          2.5%     25%     50%     75%    97.5%
## a      8.9269  9.1148  9.2180  9.3064  9.51972
## bR    -0.3559 -0.2537 -0.2009 -0.1482 -0.04605
## bA    -2.4314 -2.1060 -1.9384 -1.7689 -1.44838
## bAR    0.1227  0.2960  0.3920  0.4822  0.65547
## sigma  0.8594  0.9121  0.9480  0.9814  1.05268
```

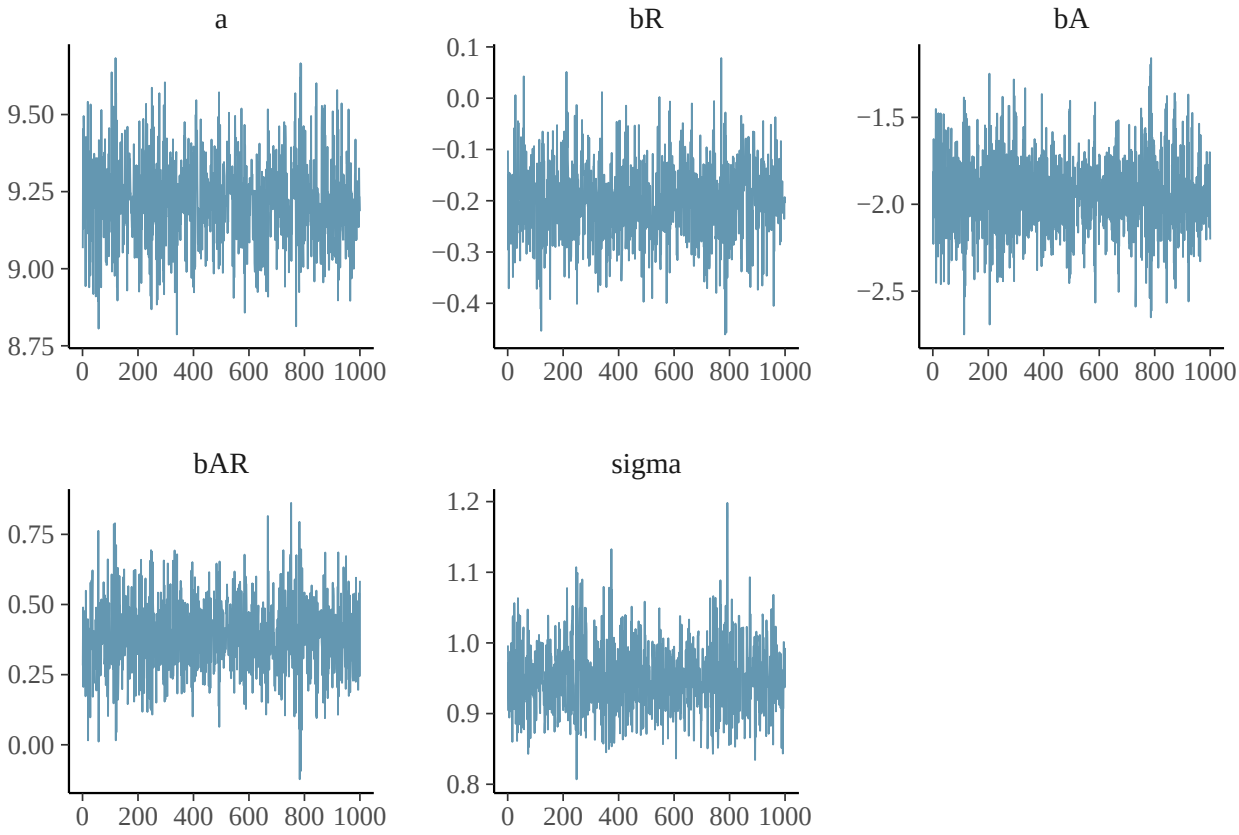
Now that we have the samples of the posterior distributions of the parameters in the model, it is straightforward to examine them. Here, we plot the posterior distribution of the interaction term.

```r
mat <- data.frame(matrix(draws[[1]],ncol=5))
names(mat) <- c("a","bR","bA","bAR","sigma")
#head(mat)
# http://www.cookbook-r.com/Graphs/Plotting_distributions_(ggplot2)/
ggplot(mat, aes(x=bAR)) +
  geom_histogram(aes(y=..density..), binwidth=.05, colour="black", fill="white") +
  geom_density(alpha=.2, fill="#FF6666")
```

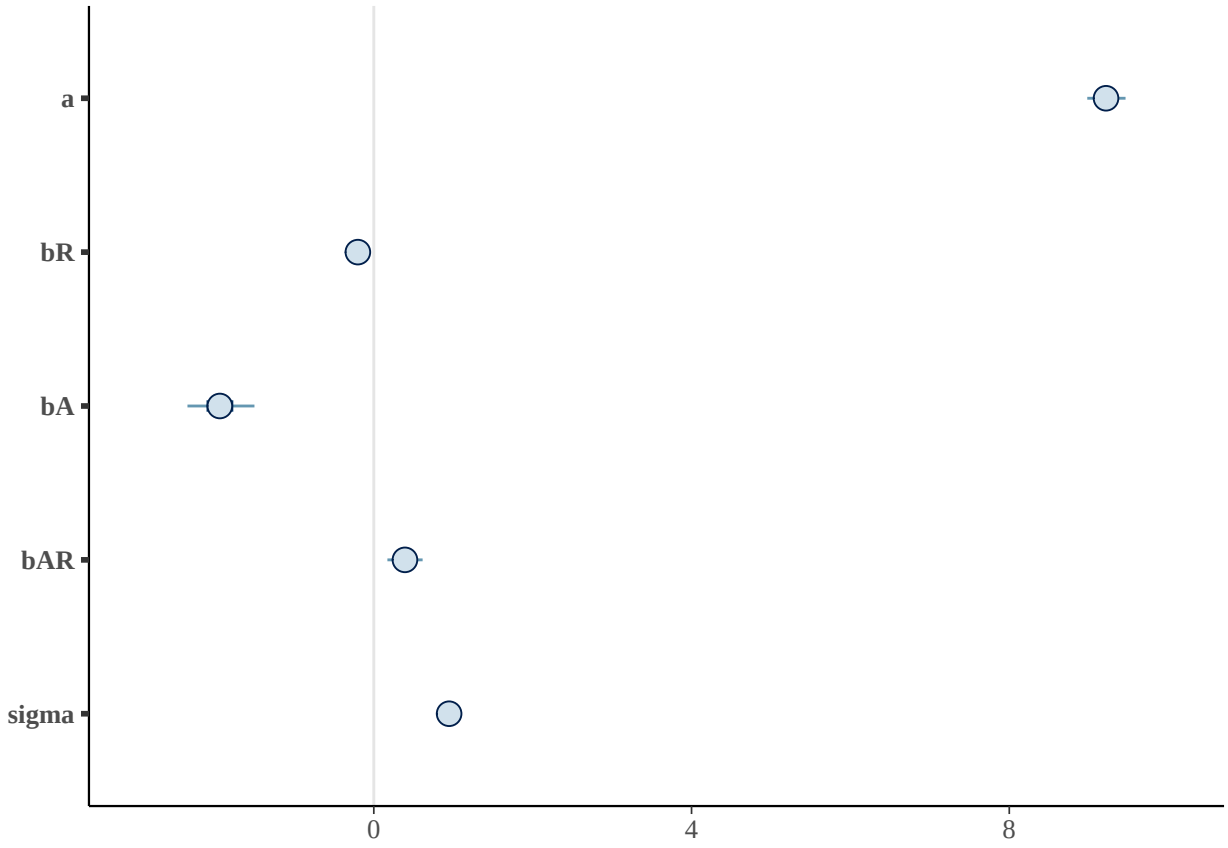


Finally, we examine the trace plots for the MCMC samples using the greta function mcmc_trace(). The plots for each parameter appear to be stationary (flat, i.e., centered on a constant value) and well-mixed (there is no obvious correlation between points). mcmc_intervals() plots the uncertainty intervals for each parameter computed from posterior draws with all chains merged.

```
mcmc_trace(draws)
```




```
mcmc_intervals(draws)
```


a
bR
bA
bAR
sigma
0
4
8