tfprobability: R interface to TensorFlow Probability

The multiverse team at RStudio, Inc.

What is `tfprobability`?

- R interface to TensorFlow Probability, the library for probabilistic programming and statistical estimation on top of TensorFlow
- Provides:
  - A vast range of **distributions** for use in upper layers
  - An equally large range of **bijectors** (invertible transformations)
  - **Distribution layers**: Keras layers that wrap **distributions**, not tensors
  - Frameworks for fitting multi-level models with **Hamiltonian Monte Carlo** or **Variational Inference**
  - Dynamic linear models (Kálmán filter, decomposition)
  - Extensions to TensorFlow functionality as regards optimizers, linear algebra, and statistics
- Fully integrated with TensorFlow Core

`tfprobability` and deep learning with Keras

- Learn *distributions*, not values
- Using *distribution layers* directly in Keras networks

Example: Uncertainty estimates for neural networks

- Have the network learn the actual spread in the data (a.k.a. "aleatoric uncertainty")
- Model uncertainty in the weights ("epistemic uncertainty"), learning an *approximate posterior* by minimizing the *evidence lower bound* (ELBO)
- See also: blogs.rstudio.com/tensorflow/posts/2019-06-05-uncertainty-estimates-tfprobability/

```
library(tensorflow)
library(tfprobability)
library(keras)

# just a keras model
model <- keras_model_sequential() %>%
  # a dense layer, but with uncertainty in the weights
  # one unit for the mean and scale each of the normal distribution
  layer_dense_variational(units = 2,
                          make_posterior_fn = posterior_mean_field,
                          make_prior_fn = prior_trainable,
                          kl_weight = 1/n
                          ) %>%
  # layer wrapping a normal distribution with learned mean and scale
  layer_distribution_lambda(function(x) tfd_normal(
    loc = x[, 1, drop = FALSE],
    scale = 1e-3 + tf$math$softplus(0.01 * x[, 2, drop = FALSE])))

negloglik <- function(y, model) - (model %>% tfd_log_prob(y))
model %>% compile(optimizer = "adam", loss = negloglik)
model %>% fit(x, y, epochs = 1000)
```

We visualize the posterior predictive as an ensemble of lines, each representing a draw from the weight posterior. The learned uncertainty due to the data is indicated by the shades that frame each line.

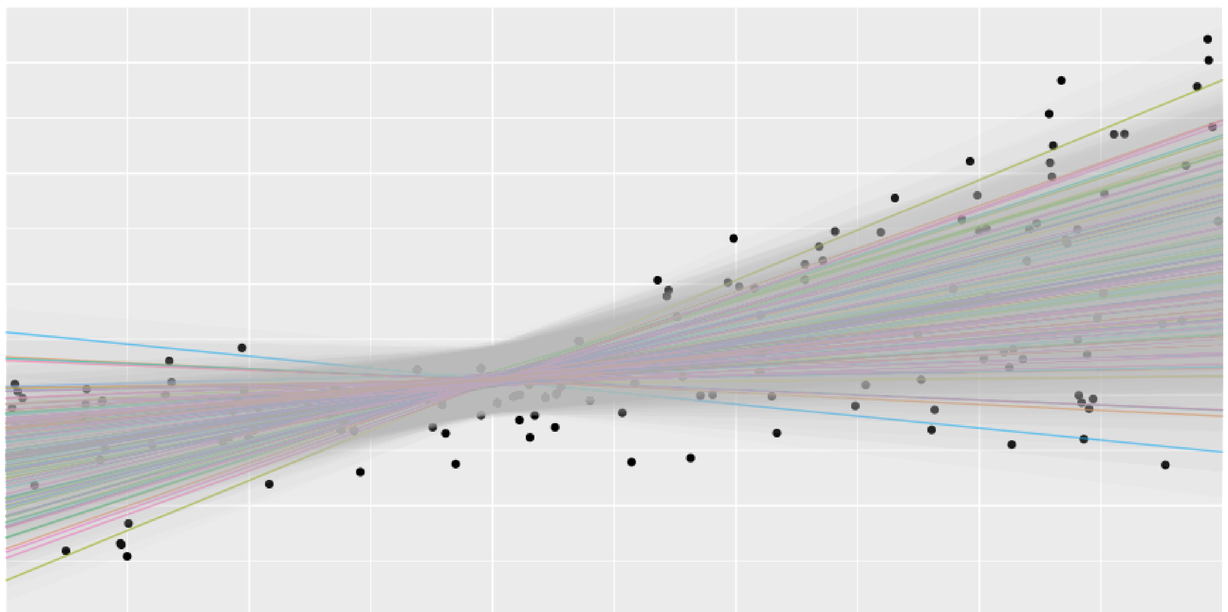

Figure 1: Posterior predictive distribution. Each line is produced by sampling from the posterior on the weights, the shading indicating the respective scales (= the learned spread in the data).

Example: Variational autoencoder

- Encoder and decoder both are sequential models, joined via the functional API
- The last layer of the encoder adds a KL loss so we can simply fit with maximum likelihood

```
encoder_model <- keras_model_sequential() %>%
  [...] %>%
  layer_multivariate_normal_tri_l(event_size = encoded_size) %>%
  layer_kl_divergence_add_loss([...])

decoder_model <- keras_model_sequential() %>%
  [...] %>%
 layer_independent_bernoulli([...])

vae_model <- keras_model(inputs = encoder_model$inputs,
                         outputs = decoder_model(encoder_model$outputs[1]))
vae_loss <- function (x, rv_x) - (rv_x %>% tfd_log_prob(x))
```

Fitting multi-level models with `tfprobability`

- Varying intercepts example from R. McElreath's "Statistical rethinking"
- Define model as a sequence of conditional distributions and fit with Hamiltonian Monte Carlo
- Employs *partial pooling* to make use of common features between tanks
- See also: blogs.rstudio.com/tensorflow/posts/2019-05-06-tadpoles-on-tensorflow

```
model <- tfd_joint_distribution_sequential(
  list(
    tfd_normal(loc = 0, scale = 1.5),
    tfd_exponential(rate = 1),
    function(sigma, a_bar)
      tfd_sample_distribution(
        tfd_normal(loc = a_bar, scale = sigma),
        sample_shape = list(n_tadpoles)
      ),
    function(l)
      tfd_independent(
        tfd_binomial(total_count = n_start, logits = l),
        reinterpreted_batch_ndims = 1
      )
  )
)

hmc <- mcmc_hamiltonian_monte_carlo([...])
res <- hmc %>% mcmc_sample_chain([...])
```

After fitting we see the expected shrinkage in the mean survival estimates per tank:

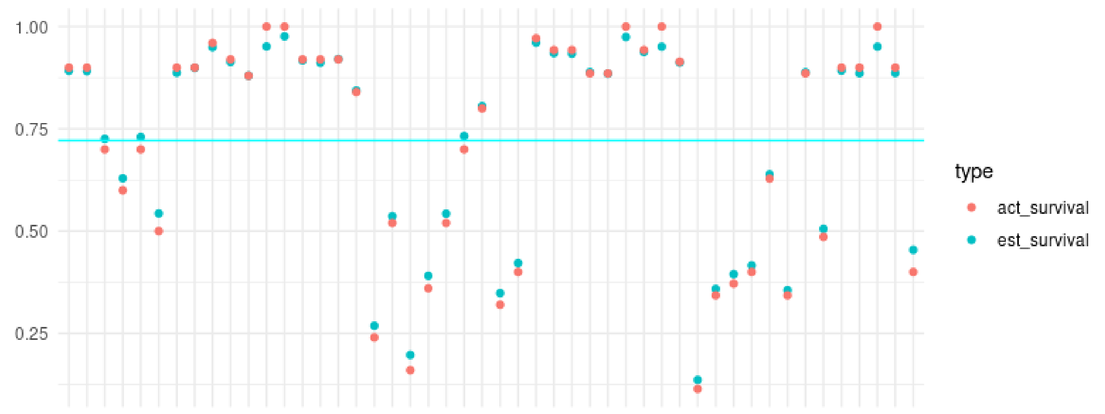

Figure 2: Shrinkage in mean survival estimates due to partial pooling.

Where to from here?

- Follow the TensorFlow for R blog: blogs.rstudio.com/tensorflow/
- Documentation: rstudio.github.io/tfprobability/
- Github: github.com/rstudio/tfprobability
- The *multiverse team* on Youtube: www.youtube.com/channel/UCAwJMtPx4HgmMX

And… stay skeptic!

Figure 3: The skeptical hamster, popularized by Richard McElreath in his 2019 Statistical Rethinking lectures.

"1 skeptical hamster for sale. He keeps looking at you skeptically like you're doing it all wrong. It's driving me crazy, I can't stand this look of reproach anymore. His name is Olaf."