TensorFlow Distributions

Joshua V. Dillon, Ian Langmore, Dustin Tran, Eugene Brevdo, Srinivas Vasudevan,
Dave Moore, Brian Patton, Alex Alemi, Matt Hoffman, Rif A. Saurous

Why TensorFlow?

- automatic differentiation: numeric eval of derivative
- vectorized: exploits single instruction, multiple data instructions
- open source: www.tensorflow.org
- deferred execution model
- leverage custom hardware: TPU, CPU, GPU
- distributed computation
- Google-quality engineering

Why TensorFlow Distributions?

- first-class citizen in TensorFlow (see above)
- fast: API makes it hard to write slow code
- flexible: random variable transformations, derived distributions
- numerically precise (@ 16, 32, 64 bits)
- Bijectors
  - volume-tracking transformations
  - automatic caching
- workhorse behind Edward probabilistic programming (`edwardlib.org`)
- well suited for black-box variational inference and HMC

Shape Semantics

Distributions exploits vector computation by combining multiple draws and parameterizations in a single `Tensor`.

[$\underbrace{\text{$n$ draws per parameterization}}$, $\underbrace{\text{$b$ different parameterizations}}$, $\underbrace{\text{$d$ dimensions per draw}}$]

`sample_shape` (indep, identically distributed) | `batch_shape` (indep, *not* identical) | `event_shape` (can be dependent)

The shape of sample `Tensor`s is partitioned into three components:

1. Sample shape represents independent, identically distributed draws, generated by running `dist.sample(n)`.
2. Batch shape indexes different parameterizations to the same distribution family; this enables the common use case in machine learning of a "batch" of input/output pairs where each input parameterizes a different distribution.
3. Event shape represents the shape of a single draw (random event) from the distribution. E.g., scalar distributions (Normal, Gamma, etc.) have event shape `[]`, while a distribution over images might have event shape `[640, 480]`.

Black-Box Variational Inference

```
e = make_encoder(x)
z = e.sample(n)
d = make_decoder(z)
r = make_prior()

avg_elbo_loss = tf.reduce_mean(
 e.log_prob(z) - d.log_prob(x) - r.log_prob(z))

train = tf.train.AdamOptimizer().minimize(
 avg_elbo_loss)
```

Example 1: Variational Autoencoder

```
def make_encoder(x, z_size=8):
  net = make_nn(x, z_size*2)
  return tfd.MultivariateNormalDiag(
    loc=net[..., :z_size],
    scale=tf.nn.softplus(net[..., z_size:]))

def make_decoder(z, x_shape=(28, 28, 1)):
  net = make_nn(z, tf.reduce_prod(x_shape))
  logits = tf.reshape(
    net, tf.concat([[-1], x_shape], axis=0))
  return tfd.Independent(tfd.Bernoulli(logits))

def make_prior(z_size=8, dtype=tf.float32):
  return tfd.MultivariateNormalDiag(
    loc=tf.zeros(z_size, dtype)))
```

Example 2: Laplace-Normal compound

$$p(x \mid \sigma, \mu_0, \sigma_0) = \int_{\mathbb{R}} \mathsf{Normal}(x \mid \mu, \sigma)\, \mathsf{Laplace}(\mu \mid \mu_0, \sigma_0)\, d\mu.$$

```
# Draw n iid samples from a Laplace.
mu = tfd.Laplace(
  loc=mu0, scale=sigma0).sample(n)
# ==> shape: ([n], [], [])

# Compute n different Normal pdfs at
# scalar x, one for each Laplace draw.
pr_x_given_mu = tfd.Normal(
  loc=mu, scale=sigma).prob(x)
# ==> shape: ([], [n], [])

# Average across each Normal's pdf.
pr_x = tf.reduce_mean(pr_x_given_mu, axis=0)
# ==> pr_estimate.shape=x.shape=[]
```

Bijectors

A Bijector implements a bijective, differentiable function, its inverse, and the log of its Jacobian determinant. A new random variable Y can be defined in terms of another random variable a bijector, e.g.,

$$p_Y(y) = p_X(F^{-1}(y))\ |DF^{-1}(y)|,$$

where DF^{-1} is the inverse of the Jacobian of F.

Using Bijectors

TransformedDistribution is a distribution $p(y)$ consisting of a base distribution $p(x)$ and invertible, differentiable transform $Y = g(X)$.

```
standard_gumbel = tfd.TransformedDistribution(
  distribution=tfd.Exponential(rate=1.),
  bijector=tfb.Chain([
    tfb.Affine(
      scale_identity_multiplier=-1.,
      event_ndims=0),
    tfb.Invert(tfb.Exp()),
  ]))
```

Bijector Caching

- Bijectors automatically cache input/output pairs of operations, including the log Jacobian determinant.
- Cache hits occur when computing probabilities of sampled values (as in variational inference); this allows us to elide the inverse calculation.
- Advantageous when inverse calculation is slow, numerically unstable, or not easily implementable.
- Can improve asymptotic complexity, e.g., takes InverseAutoregressiveFlows from quadratic to linear time.

Smooth Coverings

- Bijector framework extends to non-injective transformations where the domain can be partitioned as a finite union of D_k's such that each $F : D_k \to F(D)$ is a diffeomorphism (i.e., smooth coverings).
- Example:

```
half_cauchy = tfd.TransformedDistribution(
  bijector=tfb.AbsoluteValue(),
  distribution=tfd.Cauchy(loc=0., scale=1.))
```

Mountain View, CA

{rif,trandustin}@google.com