A Small (Probabilistic) Program Can Be A Big Challenge

David Tolpin

Pyro

```
# Works with importance sampling,
# not differentiable
def _update_beliefs(beliefs, i, j, bandwidth):
    beliefs[i, j] += 1
    evidence = beliefs[i, :].sum()
    if evidence > bandwidth:
        beliefs[i, :] *= bandwidth / evidence
    return beliefs


# Differentiable, works with variational inference
def update_beliefs(beliefs, i, j, bandwidth):
    update = torch.zeros(beliefs.shape)
    update[i, j] = 1
    beliefs = beliefs + update
    evidence = beliefs[i, :].sum()
    if evidence > bandwidth:
        scale = torch.ones(beliefs.shape)
        scale[i, :] = bandwidth / evidence
        beliefs = beliefs * scale
    return beliefs


def model(total_page_count, data):
    bandwidth = pyro.sample("bandwidth",
        Exponential(torch.tensor(0.05)))
    churn_probability = 1 / PAGES_PER_SESSION_PRIOR
    churn_beliefs = torch.stack(
        [2 * churn_probability
           * torch.ones(total_page_count - 1),
         2 * (1 - churn_probability)
           [* torch.ones(total_page_count - 1)],
        dim=1)

    for (id, pps) in enumerate(data):
        for ip in range(total_page_count - 1):
            a, b = churn_beliefs[ip]
            dist = Bernoulli(probs=a/(a + b))
            if ip == int(pps) - 1:
                # churned out
                sample("obs_{}_{}".format(id, ip),
                    dist, obs=torch.tensor(1.))
                churn_beliefs = update_beliefs(
                    churn_beliefs, ip, 0, bandwidth)
                break
            else:
                # stayed
                sample("obs_{}_{}".format(id, ip),
                    dist, obs=torch.tensor(0.))
                churn_beliefs = update_beliefs(
                    churn_beliefs, ip, 1, bandwidth)
```

Page–Per–Visit Forecasting

- An internet article is split into pages.
- Advertisements are shown on every page.
- The ith visitor 'churns' after K_ith page.
- We want to forecast the number of pages.

Model

- A vector of *Beta–Bernoulli* distributions.
- Bandwidth C accounts for changes.

Inferring C

$C \sim Prior$
for $i = 1$ **to** N
  for $j = 1$ **to** K_i
    if $j = K_i$
      $\alpha_j \leftarrow \alpha_j + 1$
      $1 \sim Beta\text{-}Bernoulli(\alpha_j, \beta_j)$
    else
      $\beta_j \leftarrow \beta_j + 1$
      $0 \sim Beta\text{-}Bernoulli(\alpha_j, \beta_j)$
    if $\alpha_j + \beta_j > C$
      $\alpha_j, \beta_j \leftarrow \alpha_j \cdot \frac{C}{\alpha_j+\beta_j}, \beta_j \cdot \frac{C}{\alpha_j+\beta_j}$

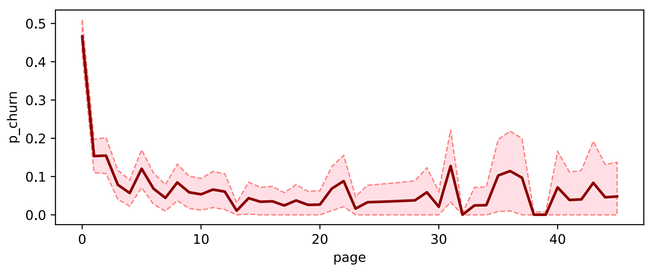
Stan

```
// initialize beliefs
real beliefs[npages, 2];
real churn_probability = 2. / npages;
int churned;

for(i in 1:npages) {
   beliefs[i][1] = 2. * churn_probability;
   beliefs[i][2] = 2. * (1 - churn_probability);
}
// put a prior on the bandwidth
target += -bandwidth / prior_bandwidth;

for (i in 1:nsessions)
   for(j in 1:npages) {
      real evidence = beliefs[j, 1] + beliefs[j, 2];
      churned = j < pps[1] ? 0 : 1;

      if(churned) {
         target += log(beliefs[j, 1] / evidence);
         beliefs[j, 1] += 1;
      } else {
         target += log(beliefs[j, 2] / evidence);
         beliefs[j, 2] += 1;
      }

      // discount the beliefs based on the bandwidth
      if(evidence >= bandwidth) {
         real discount = bandwidth / evidence;
         beliefs[j, 1] *= discount;
         beliefs[j, 2] *= discount;
      }
      if(churned) break;
   }
```

Edward

```
def model(bandwidth,
    page_count, number_of_sessions, data):
    churn_probability = 1 / PAGES_PER_SESSION_PRIOR
    beliefs = tf.stack([
        2 * churn_probability
          * tf.ones(page_count),
        2 * (1 - churn_probability)
          * tf.ones(page_count)],
       axis=1)

    def over_sessions(state, isession):

        def over_pages(beliefs, ipage, last_page):
            last_page = tf.logical_or(
                tf.equal(ipage, data[isession] - 1),
                tf.equal(ipage, page_count - 1))
            beliefs = update_beliefs(
                beliefs, ipage,
                tf.cond(last_page,
                    lambda: 0, lambda: 1),
                bandwidth)
            return (beliefs, ipage + 1, last_page)

        def continues(lefts, ipage, last_page):
            return tf.logical_not(last_page)

        beliefs, _ = state
        beliefs, _, _ = tf.while_loop(continues,
            over_pages,
            (beliefs, 0, tf.constant(False)))

        return beliefs, beliefs

    _, beliefs = tf.scan(over_sessions,
        tf.range(number_of_sessions),
        (beliefs, beliefs))

    scattered_lefts = \
        Bernoulli(probs=beliefs[:, :, 0] / \
        (beliefs[:, :, 0] + beliefs[:, :, 1]))
    scattered_lefts = tf.concat([
        # never left after 0 pages
        tf.zeros((scattered_lefts.shape[0], 1),
            dtype=tf.int32),
        scattered_lefts[:, :-1],
        # always left  by reaching the end
        tf.ones((scattered_lefts.shape[0], 1),
            dtype=tf.int32)],
        axis=1)
    lefts = tf.argmax(scattered_lefts, axis=1)

    return Normal(tf.cast(lefts, dtype=tf.float32),
                  scale=0.5)
```

Posterior churn probabilities



Checklist

- ☐ Updatable data structures: ~~Pyro~~, ~~Edward~~.
- ☐ Robust automatic differentiation: ~~Pyro~~, ~~Edward~~.
- ☐ Model representation in 'host' language: ~~Edward~~, ~~Stan~~.

$\frac{\text{msec}}{\text{sample}}$	Stan	Anglican	Pyro	Edward
	0.3	0.8	70	40