# Design Doc:
# Tensor Dimensions in Pyro 0.2

Authors @fritzo, @jankowiak

First edit 2018-02-05 - Last edit 2018-02-12

## Objective

Organize tensor dimensions in Pyro to support **nested iarange** and **parallel enum_discrete** aka BranchPoutine.

## Background

Pyro 0.1.2 release supports only a single batch dimension and a single event dimension. This limits Pyro programs to a single iarange (no nesting). Moreover Pyro 0.1.2 implements enum_discrete via sequential sampling of traces. Early attempts at parallelizing via BranchPoutine were blocked by lack of support for broadcasting: to parallelize, Pyro must introduce a new batch dimension on the left, but Pyro 0.1.2 only allows a single batch dimension.

Recent migration to PyTorch distributions has given Pyro the ability to support arbitrarily many dimensions and broadcasting on the left (samples and distribution parameters can be arbitrarily broadcast). This makes it possible in theory to support nested iarange and parallel enum_discrete, even in a single Pyro model/guide pair.

# Design of Dimension Handling

## multiple batch dimensions

Consider the pyro model for time series with outlying noisy observations:

```
1.  def model_v1(time_series):
2.      N = time_series.size(0)  # number of time series
3.      y = pyro.sample("y", Bernoulli(ng_ones(365) / 2))  # global sensor outage
4.      with pyro.iarange("series", N):
5.          z = pyro.sample("z", Bernoulli(ng_ones(N, 365) / 2)) # local sensor outage
6.          is_outlier = y * z
7.          pyro.sample("good", Normal(..., log_pdf_mask=not is_outlier), obs=time_series)
8.          pyro.sample("bad", Normal(..., log_pdf_mask=is_outlier), obs=time_series)
```

Note that the sample sites on lines 7-8 both have .log_prob() shape (N, 365). TraceGraph_ELBO should sum over the dimension of size 365 and preserve the dimension of size N. Currently TraceGraph_ELBO assumes there is only one dimension and preserves all dimensions there (both in this example). To indicate which dimensions should be summed out entirely, Pyro 0.2 will require users to locate these dimensions on the right and set extra_event_dims=n:

```
1.  def model_v2(time_series):
2.      N = time_series.size(0)  # number of time series
3.      y = pyro.sample("y", Bernoulli(ng_ones(365) / 2), extra_event_dims=1)
4.      with pyro.iarange("series", N):
5.          z = pyro.sample("z", Bernoulli(ng_ones(N, 365) / 2, extra_event_dims=1))
6.          is_outlier = y * z
7.          pyro.sample("good", Normal(..., log_pdf_mask=not is_outlier, extra_event_dims=1),
8.              obs=time_series)
9.          pyro.sample("bad", Normal(..., log_pdf_mask=is_outlier, extra_event_dims=1),
10.             obs=time_series)
```

## enum_discrete

To support enum_discrete over both "y" and "z" we could have instead introduced an additional iarange over the time dimension. After this extra_event_dims is no longer necessary since all dimensions are batched:

```
1.  def model_v3(time_series):
```

```
2.      N = time_series.size(0)  # number of time series
3.      with pyro.iarange("time", 365):
4.          y = pyro.sample("y", Bernoulli(ng_ones(365) / 2))
5.          with pyro.iarange("series", N):
6.              z = pyro.sample("z", Bernoulli(ng_ones(N, 365) / 2))
7.              is_outlier = y * z
8.              pyro.sample("good", Normal(..., log_pdf_mask=not is_outlier), obs=time_series)
9.              pyro.sample("bad", Normal(..., log_pdf_mask=is_outlier), obs=time_series)
```

This version ensures that “y” and “z” both have zero event dims; therefore we can cheaply enumerate in lock-step rather than enumerate over the cartesian product.

## parallel enum_discrete

To support parallel enum_discrete (aka BranchPoutine), Pyro 0.2 will additionally require a way to safely allocate new batch dimensions. To see the difficulty, note that Pyro 0.2 wants to allocate a new batch dimension at sample site “y” on line 4 above, but does not yet know about the upcoming batch dimension needed by iarange on line 5.

### *Solution 1:*

Require users to declare a maximum iarange nesting depth before inference. This appears reasonable since iarange is generally used per-data-type (e.g. users, cities, weeks) and there are only a few (usually 1) data types per pyro program. This could be an additional argument to SVI along with enum_discrete=”parallel”, e.g.

```
inference = SVI(model, guide, optim, ‘ELBO’,
                enum_discrete=”parallel”, max_iarange_nesting=2)
```

This dim allocation scheme would give Pyro freedom to use all batch dims to the left of dim -2, and give the user ownership over the two rightmost batch dimensions (-2 and -1). Of course the user also has ownership over all event dimensions, but they are summed out inside `.log_prob()`.

**Pros:** more composable, almost no interface change, easier to detect errors
**Cons:** not universal (no arbitrarily nested iarange)

### *Solution 2:*

Allow Pyro to decide all batch dimension usage. This would require expanding the interface of iarange to output a dimension. Let’s calls this islice to distinguish the two:

```
def model_v4(time_series):
    N = time_series.size(0)  # number of time series
    with pyro.islice(“time”, 365) as (ind1, dim1):
        y = pyro.sample(“y”, pyro.reshape(Bernoulli(ng_ones(365) / 2), dim1))
        with pyro.islice(“series”, N) as (ind2, dim2):
            z = pyro.sample(“z”, pyro.reshape(Bernoulli(ng_ones(N, 365) / 2), dim1, dim2))
            is_outlier = y * z
            time_series = pyro.reshape(time_series, dim1, dim2)
            pyro.sample(“good”, Normal(..., log_pdf_mask=not is_outlier), obs=time_series)
            pyro.sample(“bad”, Normal(..., log_pdf_mask=is_outlier), obs=time_series)
```

**Pros:** universal, somewhat composable
**Cons:** requires tensor reshaping, requires dim plumbing, difficult to detect errors

## An extended example: AIR

Consider parallelizing the num_particles enumeration in Trace_ELBO and TraceGraph_ELBO.

```
with pyro.iarange('data', data.size(0)) as index:
    for t in range(num_max_objects):
        pyro.sample('z_pres_{}' …, Bernoulli(0.5))  #  BS
        pyro.sample('z_what_{}' ...)  # D_1 x BS
        pyro.sample('z_where_{}' ...)  # D_2 x BS
    pyro.sample('obs', ..., obs=data)  # D_3 x BS
```

Note that to use parallel enumeration efficiently requires some tensor gymnastics including a Geometric distribution

```
def model():
    with pyro.iarange('data', data.size(0)) as index:
        n = pyro.sample(“num_objects”, TruncatedGeometric(...))
        mask = torch.eye(n, n).tril()
        for t in range(n):
            pyro.sample('z_pres_{}' …, Bernoulli(0.5))  # 1 x BS
            pyro.sample('z_what_{}' ...)  # D_1 x BS
            pyro.sample('z_where_{}' ...)  # D_2 x BS
        pyro.sample('obs', ..., obs=data)  # D_3 x BS

def guide():
    with pyro.iarange('data', data.size(0)) as index:
        n = pyro.sample(“num_objects”, UniformInt(max_num_objects),
                                  infer={‘enumerate’: ‘parallel’})
        mask = torch.eye(n, n).tril()
        for t in range(n):
```

```
        z_pres = guide_step(t)
```

## Work plan

1. #638 Broadcasting
    a. document good practice of broadcasting, e.g. replace:
       my_tensor[i,j] → my_tensor[..., i, j] # this uses python ellipsis slicing notation
2. #370 Nested iarange
    a. document correct usage of extra_event_dims
    b. add checks in TraceGraph_ELBO to ensure extra_event_dims is being used
3. #742 Parallel enum_discrete
    a. add max_iarange_nesting flag to TraceGraph_ELBO and error if it is violated
    b. implement infer={‘enumerate’: ‘parallel’} via BranchPoutine or similar