Experiments with Static Analyser for Pyro Programs

1. Experiment Setup

We used eight Pyro examples taken from the Pyro webpage (http://pyro.ai/examples/, retrieved on April 1, 2019). The details of the examples are shown in the below table.

Name	Corresponding probabilistic model	LoC	Total dimension of PyTorch tensors	Total number of Pyro plates
br	Bayesian regression	27	189	1
csis	Inference compilation	31	484	0
lda	Amortised latent Dirichlet allocation	76	206,408	5
vae	Variational autoencoder (VAE)	91	579,904	2
sgdef	Deep exponential family	94	1,773,280	8
dmm	Deep Markov model	246	1,515,600	2
ssvae	Semi-supervised VAE	349	1,024,800	2
air	Attend-infer-repeat	410	6,221,595	2

The fourth column of the table represents the total dimension of only those PyTorch tensors in each example that are either trained by Pyro's inference engine, or generated by Pyro's sample statements, or used by Pyro's observe statements. Note that each example creates many more PyTorch tensors, as intermediate tensor values, than the tensors shown in the fourth column.

2. Summary of Experiment Results

We ran our static analyser for Pyro programs on the eight Pyro examples to verify whether the examples have no dimension-related errors caused by PyTorch tensors and Pyro plates. The analysis results are shown in the below table.

Name	Valid uses of PyTorch tensors?	Valid uses of Pyro plates?	Verification time (sec)
br	o	o	0.006

csis	o	o	0.007
lda	o	o	0.014
vae	o	o	0.005
sgdef	o	o	0.070
dmm	o	o	0.536
ssvae	o	o	0.013
air	o	o	4.093

The second and third columns show the outputs of our analyser and the fourth shows the computation times taken by our analyser.

The table demonstrates that our analyser successfully verified, within a few seconds, that all the eight examples are free of any dimension-related errors caused by PyTorch tensors and Pyro plates. This matches with our expected result.