1 Included Files

1. submit-script.sh Bash submission script currently set for running on debug queue with two nodes and two workers.

2. calling_script.py: Calling script for ensemble. No user generator function is used, loads arguments passed to simulator function from *test_training_args.npz*.

3. simulator.py User simulation function for ensemble. Because only one application can be submitted per node and my application did not benefit from additional CPUs past four, I spawned multiple processes per node (10) and trained a single network per process. This allowed me to submit a single application per worker that trained 10 networks simultaneously per node. Each worker launches an application with:

```
aprun -L 3834 -n 10 -N 10 -d 4 -j 1 -cc depth -e OMP_NUM_THREADS=1 <path to app>/train.py <app args>
```

4. train.py User application for training recurrent networks. If execution time permits, application will run to completion, otherwise network will checkpoint training history and weights in */grand/rnn-robustness/test-checkpoints/*.

5. training_dependencies.py Training application helper functions.

6. load_data.py Helper functions for loading experiment training data.

7. test_training_args.npz: Python dictionary containing pre-generated training arguments (used in place of libE generator). Access dictionary with

```
import numpy as np
args = np.load('test_training_args.npz', allow_pickle=True)['training_args'].item()
```

2 Training Output

Depending on model and experiment configuration, training can exceed allowed ensemble execution time. When network training completes in allotted execution time, training history is saved to *ensemble/(worker id)/(network specifier).npz*. If training is not scheduled to complete prior to exceeding ensemble execution time, its training history, training arguments and model/optimizer weights are stored in */grand/rnn-robustness/test-checkpoints/(model specifier)/*. A secondary (disjoint) ensemble script is called later to complete these checkpointed models.