Documentation for the Tree HMM Code

Adrianna Loback

December 8, 2018

1 Description

Provided is the code to infer the parameters of the Tree hidden Markov model (HMM), which was introduced in [1], given an input set of neural population spike train data. For the mathematical details regarding the formulation of the model and fitting procedure (which is done via a modified version of the Baum-Welch algorithm), please see references [1, 2, 3]. The original code was written by Jason Prentice in C++ as multiple source files, and has been mexed for direct, callable use in Matlab (see below sections for instructions on how to run the code within Matlab). Though the code used for the results reported in [1, 2, 3] was written to be run in parallel on the clusters at Princeton University, it has recently been modified here for local use. Please note that this code is provided freely with no guarantees. For questions regarding problems in running the code, please contact Adrianna at arl64@cam.ac.uk.

Note that the specific version of the code provided here fits the full version of the Tree HMM (i.e. including the transition probabilities and tree edge weight parameters for each emission distribution), and returns the log-likelihood on the designated held-out test set. This version can thus be used for cross-validation to choose the number of latent modes.

2 Running the Code

2.1 Running with 64-bit Matlab 2017b

To run the code locally as-is, you will need to have the 64-bit version of Matlab 2017b installed, which was used to mex the C++ code into a Matlab compatible format. In this case, you simply need to copy the file `EMBasins.mexmaci64` to your local Matlab directory (i.e., the subfolder within your local Matlab directory from which you want to call the function from). Then proceed to section 2.3 of this document.

2.2 Compiling (mexing) & running with another version of Matlab

In the case that you have another version of Matlab, then you will need to first mex the code on your local computer. To do this, copy the files `EMBasins.cpp`, `BasinModel.o`, and `TreeBasin.o` into the same subdirectory (within your local Matlab directory). You will need to have BOOST (a set of C++ libraries) installed locally. If you do not have BOOST installed locally, then I would recommend using Homebrew (if you have Mac OS X) to install BOOST. To do this in Mac OS X, open the command prompt, and execute the following command:

Installing boost and boost-python on OSX with Homebrew — Shell

```
1 $ brew install boost --with-python
```

Note that this installation may take a while. Once finished, BOOST should be installed in the directory `/usr/local/Cellar/boost/<version number>`. As of writing this, the most recent version was 1.68.0.

Next, open Matlab, and ensure that you are located in the same subdirectory within which the `EMBasins.cpp`, `BasinModel.o`, and `TreeBasin.o` files are located. Then, **within Matlab**, type and enter the following command:

```
mex -largeArrayDims -I/usr/local/include -I/usr/local/Cellar/boost/1.68.0 -lgsl
-lgslcblas EMBasins.cpp BasinModel.o TreeBasin.o
```
(1)

2.3 Syntax

After successfully mexing (i.e. for 64-bit Matlab, you will have a file called `EMBasins.mexmaci64`), you can call the code to fit the Tree HMM directly within Matlab. Note that for running k-fold cross-validation, you will need to manually create the k training and test sets, and provide these as inputs.

The syntax for calling the fitting procedure within Matlab is:

```
[logli, trans, P, alpha, backward, params] =
EMBasins(spiketimes, testset, binsize, nbasins, niter)
```
(2)

where the output arguments are:

- `logli` is the log-likelihood on the *test* set
- `trans` is the inferred transition probability matrix
- `P` is the probability of each basin over time
- `alpha` is the most likely basin in each time bin (inferred via the Viterbi algorithm)
- `backward` is similar to `P`, but only uses past data
- `params` is a Matlab structure that contains the fit parameters $J_\alpha \in \mathbb{R}^{N\times N}$ and $m_\alpha \in \mathbb{R}^N$ for each latent mode α, where N denotes the number of neurons in the population for the input data.

For the input arguments:

- `spiketimes` is a cell array of length N, each entry of which contains the spike times of one neuron (i.e. entry i contains the spike times of neuron i). **Important note**: The units of the input argument `binsize` should match the units of the spike times. (E.g. 10 kHz in our case).
- `testset` is a $n_{test} \times 2$ matrix, where the first column indicates the time bin of the start of each held-out test portion, and the second column indicates the time bin of the end of each held-out test portion. (See example code).
- `binsize` is the time bin size, with units matching those of the spike time data
- `nbasins` is the user-designated number of latent modes (a hyperparameter, which can be optimized via e.g. cross-validation)
- `niter` denotes the number of iterations to run the iterative fitting algorithm (Baum-Welch) for. In practice, we generally found that 100 iterations was sufficient to reach steady-state log-likelihood.

3 Example Matlab Code

An example of a Matlab wrapper function that calls `EMBasins` is also provided, called `getHMMParams.m`.

References

[1] Prentice, J.S., Marre, O., Ioffe, M.L., Loback, A.R., Tkacik, G., and Berry, M.J. 2nd. (2016). Error-robust modes of the retinal population code. *PLoS Comput. Biol.*, 13(11): e1005855.

[2] Loback, A.R., Prentice, J.S., Ioffe, M., and Berry, M.J. 2nd. (2017). Noise-robust modes of the retinal population code have the geometry of "ridges" and correspond to neuronal communities. *Neural Computation*, 29: 3119-3180.

[3] Loback, A.R. (2018). Representational and Robust Principles of the Retinal Population Code. *Ph.D. Thesis*, Princeton University.Tre