TLSA Manual

Samuel J. Gershman

February 4, 2013

Contents

1 Introduction

This document contains instructions for how to use the TLSA Matlab code to analyze EEG and fMRI data. (Note that it is not limited to these imaging modalities, but for concreteness we will restrict our attention to them). See `tlsa_demo.m` for a demonstration.

Any questions should be directed to: sjgershm@mit.edu.

2 Function glossary

- `tlsa_EM.m`: run variational expectation-maximization to fit TLSA to a data set.
- `tlsa_opts.m`: set default options.

- `tlsa_init.m`: initialize approximate posterior. This function is specialized for the spatial and spatiotemporal radial basis functions. If you use a customized basis function, you need to initialize the posterior yourself.
- `tlsa_decode_gaussian.m`: decode continuous covariates under a Gaussian prior.
- `tlsa_decode_gradient.m`: decode continuous covariates under non-Gaussian priors using gradient descent.
- `tlsa_decode_discrete.m`: decode discrete covariates.
- `tlsa_linear_contrast.m`: performa a linear contrast on TLSA weights.
- `map_rbf.m`: spatial radial basis function.
- `map_st_rbf.m`: spatiotemporal radial basis function.

3 Terminology

Here we lay out the basic notation of the model, to which we will refer in later sections. Let C be the number of covariates, N be the number of observations, K be the number of latent sources, S be the number of subjects, and V be the number of neural features. The model consists of the following variables:

- $\mathbf{X} \in \mathbb{R}^{N \times C}$, the *design matrix* containing each covariate's timeseries for subject s.
- $\mathbf{W} \in \mathbb{R}^{C \times K}$, the *weight matrix* encoding how each covariate loads on each source.
- $\mathbf{F} \in \mathbb{R}^{K \times V}$, the matrix of basis images for each latent source.
- $\mathbf{Y} \in \mathbb{R}^{N \times V}$, the *neural data matrix*, representing the pattern of activity at each observation.
- $\mathbf{R} \in [0, 1]^{V \times D}$, the *location matrix*, specifying, in D-dimensional coordinates, the location of each neural feature (e.g., voxel location).

Each source is parameterized at the subject level by $\omega_{sk} \in \mathbb{R}^M$ and at the group level by ω_{0k}.

The model includes the following hyperparameters:

- $\beta \geq 0$, the *coupling parameter* that controls how similar ω_{sk} is to ω_{0k}, and thus indirectly controls how strongly coupled subjects are to one another in terms of their source parameters. When $\beta = 0$, subjects are entirely uncoupled, and there is no enforced correspondence between their sources (in this case, group-level analyses of sources are meaningless).
- $\nu > 0, \rho > 0$, parameters that control the prior on the global noise parameter τ. Typically, changing these parameters has little effect on the results.

- $\Lambda_0 \in \mathbb{R}^{M \times M}$, precision matrix of the Gaussian prior on ω_{0k}. This matrix must be positive semi-definite. For most purposes, a diagonal matrix with positive reals on the diagonal will suffice.
- $\bar{\omega} \in \mathbb{R}^M$, the mean of the Gaussian prior on ω_{0k}.

4 Data structures

Data in TLSA are stored in `data`, which is a $1 \times S$ array of structures. `data(s)` contains the data for subject s, comprising the following fields (see above for more information about these variables):

- `data(s).X`: the design matrix
- `data(s).Y`: the neural data matrix
- `data(s).R`: the location matrix

In practice, we typically normalize $\mathbf{R}$ so that its maximum is 1 and its minimum is 0.

TLSA has special handling for 3 common data types. EEG data is considered to be 4-dimensional: the first three columns of $\mathbf{R}$ are spatial dimensions (in Cartesian coordinates), and the last column is time within a trial. fMRI data is typically 3-dimensional, each column of $\mathbf{R}$ corresponds to a spatial dimension. TLSA also handles 2D fMRI data (slices).

The `opts` structure stores configurable parameters of TLSA. Each of these has a default value:

- `opts.nIter`: the number of expectation-maximization iterations to run (default: 50)
- `opts.K`: number of latent sources (default: 50)
- `opts.nu`: shape parameter for τ (default: 1)
- `opts.rho`: scale parameter for τ (default: 1)
- `opts.beta`: coupling parameter (default: 0.01)

Some of the defaults depends on the basis function used:

- `opts.mapfun`: function handle for the function that maps source parameters to basis images
- `opts.Lambda0`: precision matrix (Λ_0)
- `opts.omega_bar`: prior mean ($\bar{\omega}$)
- `opts.omega_lb`: parameter lower bounds
- `opts.omega_ub`: parameter upper bounds

By calling

```
>> opts = tlsa_opts(opts,data)
```

the missing fields of `opts` to set to their defaults. You can also set `opts=[]` in which case all the fields will be set to their defaults.

If $D = 2$, TLSA will by default assume 2D (slice) fMRI data and a 2D spatial radial basis function (RBF) source, in which case the defaults are:

- `opts.mapfun = @(theta,R) map_rbf(theta,R)`
- `opts.Lambda0 = diag([1e-5 1e-5 10])`
- `opts.omega_bar = [0 0 log(0.1)]`
- `opts.omega_lb = [inf inf 0]`
- `opts.omega_ub = -[inf inf 4]`

Here, the first two components of ω encode the source center, and the third component encodes the width. Note that the source and width parameters are transformed inside the mapping function to the $[0, 1]$ interval using a logistic sigmoid. Thus, here they have a real-valued parameterization.

If $D = 3$, TLSA will by default assume 3D fMRI data and a 3D spatial RBF source, in which case the defaults are:

- `opts.mapfun = @(theta,R) map_rbf(theta,R)`
- `opts.Lambda0 = diag([1e-5 1e-5 1e-5 10])`
- `opts.omega_bar = [0 0 0 log(0.1)]`
- `opts.omega_lb = [inf inf inf 0]`
- `opts.omega_ub = -[inf inf inf 4]`

Here, the first three components of ω encode the source center, and the fourth component encodes the width.

If $D = 4$, TLSA will by default assume 4D EEG data (last dimension time) and a spatiotemporal RBF source, in which case the defaults are:

- `opts.mapfun = @(theta,R) map_st_rbf(theta,R)`
- `opts.Lambda0 = diag([1e-5 1e-5 1e-5 10 1e-5 10])`
- `opts.omega_bar = [0 0 0 log(0.1) 0 log(0.1)]`

- `opts.omega_lb = [inf inf inf 0 inf 0]`
- `opts.omega_ub = -[inf inf inf 4 inf 4]`

Here, the first three components of ω encode the source center, the fourth component encodes the spatial width, the fifth component encodes the temporal center, and the sixth component encodes the temporal width.

The function `tlsa_EM.m` returns a `results` structure containing the following fields:

- `results.q`: subject-level posterior, a $1 \times S$ array of structures, with the following fields:
  - `results.q(s).omega`: $K \times M$ matrix of posterior mean parameter estimates for each source
  - `results.q(s).W`: $C \times K$ maximum likelihood weight matrix
  - `results.q(s).nu`: posterior shape parameter
  - `results.q(s).rho`: posterior scale parameter
- `results.q0`: group-level posterior structure with the following fields:
  - `results.q0.omega0`: $K \times M$ matrix of posterior mean parameter estimates for each source
  - `results.q0.beta`: $M \times 2$ posterior hyperparameters of the Gamma prior on the diagonal components of Λ (the precision matrix of the Gaussian distribution on ω_{sk}).
- `results.opts`: options structure with missing fields set to defaults

5 Decoding brain images

There are two functions provided for decoding covariates from brain images. For continuous covariates, the function `tlsa_decode_gaussian.m` assumes a Gaussian prior on the covariates (by default estimated from the training data) and recovers a Gaussian posterior distribution over covariates given some test data. If you wish to use a non-Gaussian distribution, see `tlsa_decode_gradient.m`, which allows you to supply your own prior; in this case the solution is no longer available in closed form, and therefore gradient descent is used. For discrete covariates, the function `tlsa_decode_discrete.m` takes the set of possible covariates and returns a posterior probability distribution over this set for each test point.

6 Hypothesis testing

The function `tlsa_linear_contrast.m` takes as input a $C \times 1$ contrast vector and returns the results of t-tests on the contrast for each source. Specifically, the inner product between the weights (across covariates) and the contrast vector for each source is compared to 0 across subjects. The

function also returns a thresholded contrast image (averaged across subjects) for plotting purposes. The contrast image is analogous to traditional contrasts in neuroimaging, but applied to the source weights instead of the regression coefficients for a mass-univariate general linear model. Note that β must be greater than 0 (i.e., subjects must be hierarchically coupled) for this analysis to be meaningful.