1 Changes to Handle $E \times K \times K$ Stuff

These are in issue 32 (use `tensordot`, allow the AllocModel to tell the ObsModel what the sufficient statistics dimensions are, etc.). Assume this is done.

2 GraphXData

Implemented as suggested by Mike: maintains a list of only observed edges in an $E \times D$ matrix `Data.X`, and 1-D arrays `sourceID`, and `destID` give the node IDs that these edges run between. Assuming the changes in Section **??** are done, obsmodels should run their calculations totally normally on this data.

2.1 Issues

While the bnpy code will run with these changes, there are a couple of issues.

Interpretation of Missing Data. One big problem with this as-is is that there's no way to distinguish between data that was truly un-observed and data that isn't there because we're using sparse storage. For example, if we're placing a gaussian on each edge of a graph, lack of an entry for x_{ij} probably means we didn't observe anything between nodes i and j. On the other hand, the MMSB and SMSB assume that all edges are "observed," but the majority are marked with a 0. We need some extra flag for how to interpret the data, probably given to `GraphXData` by the data module that creates it.

ObsModel's `SoftEV` Matrix With Sparse Binary Data. (I'm talking about `calc_local_params` for a relational model). For the `BernObsModel`, the `SoftEv` matrix is given by:

$$\texttt{SoftEv}[n, \ell, m] = x_n \mathbb{E}_q[\log w_{\ell m}] + (1 - x_n)\mathbb{E}_q[\log 1 - w_{\ell m}]$$

If `GraphXData` stores only edges that exist, `Data.X` will be a big vector of 1's (which is a waste of space, but ignore that optimization for now), and the corresponding `SoftEv` matrix will be an $E \times K \times K$ matrix, where each $K \times K$ slice contains $\mathbb{E}_q[\log w_{\ell m}]$. Worse than this being a waste of space, it won't work, as we're completely missing $\mathbb{E}_q[\log 1 - w_{\ell m}]$, which is needed by the AllocModel to compute $\hat{\phi}_{ij\ell m}$ for $(i, j) \notin E$.

3 Solutions

Interpretation of Missing Data. For now, I think this should be a flag in the `DataObj`, `missingEntriesUnobserved` (probably needs a better name), since this is a property that entirely depends on what the data is and how it was collected. AllocModels can then determine if and how they want to distinguish between the two cases. Ultimately, we might want to allow the `DataObj` to make the distinction on a more fine grained level ("these 500 datapoints are missing because of sparse storage, these 10 are missing because they're unobserved").

This shouldn't change any existing behavior, as all current code corresponds to the case of missing entries being unobserved. I don't think that the ObsModel should ever account for this distinction; I imagine that the AllocModel will read the flag and then tell the ObsModel how to behave.

ObsModel's `SoftEV` Matrix With Sparse Binary Data. In `obsmodel.setupWithAllocModel`, the obsmodel should ask the allocmodel what the correct way to handle this problem is. The default behavior should be what is currently done; the obsmodel computes a $E \times K \times K$ matrix of $\mathbb{E}_q[p(x_e \mid \phi_{\ell m})]$ ($N \times K$ for all current models). A second option is to pass back a $V \times K \times K$ matrix `ev`, where $\texttt{ev}[v, \ell, m] = \mathbb{E}_q[p(x = v \mid \phi_{\ell m})]$. That is, V is the number of values that each observation can take on. This only makes sense for discrete likelihoods (and maybe only useful for Bernouli). This both saves space and allows the obsmodel to communicate all the needed information.

Again, this shouldn't require changing all obsmodels, only ones that we want to pair with allocmodels that use this option.