1. Tracing Multiple Subskills in DBNs

Knowledge tracing models the student’s knowledge of a skill at step $n$ as a hidden state $K(n)$ that is true iff the student knows the skill. The model’s learning parameters already know, learn, and forget respectively estimate the probabilities of knowing the skill at step 0, of a transition from not knowing the skill at step $n-1$ to knowing the skill at step $n$, and of a transition (typically omitted) from knowing to not knowing. The model’s performance parameters guess and slip respectively estimate the probabilities of performing the step correctly without knowing the skill, and of getting the step wrong despite knowing the skill. We can then use these parameters to infer the probability of knowing the skill at each step from a student’s observed performance $P(n)$ (correct or incorrect) on a sequence of steps requiring the skill.

To model steps that require multiple subskills, LR-DBN models each knowledge transition as a logistic regression over all of the required subskills. Figure 1 shows the architecture of LR-DBN, where $S_j^{(n)}$ is true iff step $n$ requires subskill $j$. LR-DBN computes the learning parameters for knowledge tracing as follows:

\[ \text{already know} = P_r(K^{(0)} = T) = 1 - \text{sigmoid}(\sum \beta_j^{(0)}) \]  
\[ \text{“learn”} = P_r(K^{(n)} = T|K^{(n-1)} = F) = 1 - \text{sigmoid}(\sum \beta_j s_j^{(n)}) \]  
\[ \text{“forget”} = P_r(K^{(n)} = F|K^{(n-1)} = T) = \text{sigmoid}(\sum \gamma_j s_j^{(n)}) \]  

Here $T$ means true, $F$ means false, and sigmoid is the sigmoid function in logistic regression:

\[ \text{sigmoid}(x) = \frac{1}{1 + e^{-x}} \]

LR-DBN uses the junction tree algorithm for exact inference and the Expectation Maximization (EM) algorithm to estimate parameters. EM can return a local optimal solution to maximize the likelihood of the model given the fitting data. It also uses the iteratively reweighted least squares (IRLS) algorithm to fit a logistic regression in a DBN.

2. Tracing Multiple Subskills with BNT-SM

We implemented LR-DBN as an extension of BNT-SM. Using BNT-SM requires just four steps:

1. Specify the data source in an XML specification.
2. Specify and initialize parameters in XML.
3. Call RunBnet.m in Matlab.

The main task in using BNT-SM is to configure the XML specification, which includes sections for input, output, and structure. The first two sections specify the input and output files to use. The structure section has a nodes subsection that specifies the DBN topology, and an eclasses subsection that describes the parameters.

2.1 Specifying a LR-DBN in XML

The new BNT-SM uses the same XML as before, but adds an optional directive to use LR-DBN instead of a traditional DBN:

\[ <\text{multi_subskill}>yes </\text{multi_subskill}> \]

As shown in Figure 2, the nodes section specifies the network topology: node 1 ($kc$, short for “knowledge component”) denotes the required multiple subskills, node 2 (knowledge) denotes the student’s overall hidden knowledge, and the fluent node denotes the student’s performance.

The type multi identifies node 1 as the multiple-subskills node; its values field shows how many subskills are enumerated in the input data; and its prefix_field identifies which columns of input data to read for node $kc$. Non-multi type nodes, such as knowledge and fluent, have type discrete, values 2 (i.e. true or false), and field to identify which column to read in the input data. Latent is yes for knowledge node since it is a hidden state in LR-DBN.

Note that the topology, as well as the parameters, should repeat as the DBN unrolled further. Thus we only need to specify them for the first two time slices, i.e. the within and between transition fields indicating which nodes the current node has arcs to within the time slice and between time slices, respectively.

![Figure 1. Structure of an example DBN input to LR-DBN](image)

Therefore, besides the parameters guess and slip, LR-DBN must fit the coefficients $\beta_j^{(0)}$, $\beta_j$, and $\gamma_j$, for $j = 1, 2, \ldots, m$, instead of the traditional parameters already know, learn, and forget.
So far we have explained how to specify LR-DBN's topology. Now we will show how to specify its Conditional Probabilities Distributions (CPDs), i.e. its parameters, under section eclasses in XML. There are four eclasses: the first three describe the CPDs within the first time slice; while the last one describes the CPDs transitioned from the first time slice to the second.

As shown in Figure 3, The first eclass with formula $P1(kc)$ has a type root. It is because $kc$ has no parents and all of its values are observed, and no parameters need to be estimated. Thus we also don’t need to specify any cpd for this particular node.

The type of formula $P2(knowledge)$ is softmax (we quote this notation from BNT, indicating the same meaning of fitting a logistic regression on the current discrete node over its parents). Equation $P2(T)$ represents the parameter already know in formula (1), and we gives it an abbreviated name as $L0$. 

Figure 2. A Specification of the LR-DBN Network Structure

Figure 3. A Specification of the LR-DBN Parameters
Since LR-DBN uses EM algorithm to learn parameters. So empirically we can set an initial value as a starting point for EM, or we can set init as rand to randomly choose a starting point. Due to the complementary property of probabilities, we have to initialize P2(F) as 1-P2(T). We also give a null name to P2(F) to avoid outputting redundant parameters since it can be easily calculated as the complement of L0.

Formula P3(fluent| knowledge) has a 2-by-2 discrete CPD table. Equations P3(T|F) and P3(F|T) respectively represent the parameters guess and slip. The last formula P4(knowledge| knowledge) is type of softmax as well, since P4(F|T) and P4(T|F) represent the logistic regressions in formulas (2) and (3) respectively. Note that we do not initialize “forget” with zero as we usually do in traditional knowledge tracing because of the different meaning of “forget” in LR-DBN.

2.2 Input/Output Data Format
LR-DBN uses BNT-SM’s existing input and output data formats, adding columns for the multiple subskills. The input data sources, such as evidence.train.xls and evidence.test.xls for our example in Figure 4, are tabulated files with Tab separated columns.

```xml
<input>
  <evidence_train> evidence.train.xls </evidence_train>
  <evidence_test> evidence.test.xls </evidence_test>
</input>

<output>
  <param_table> param_table.xls </param_table>
  <inference_result> inference_result.xls </inference_result>
  <inference_result_header> inference_result.xls </inference_result_header>
  <log> log.txt </log>
</output>
```

Figure 4. Specification of LR-DBN input and output in BNT-SM

Each file has a header line with the (prefixed) names that were specified in the XML. A row represents one step of a user’s attempt on a skill. Table 1 shows a partial input data of children’s oral reading fluency.

<table>
<thead>
<tr>
<th>user</th>
<th>skill</th>
<th>kc_e</th>
<th>kc_a</th>
<th>kc_f</th>
<th>kc_h</th>
<th>knowledge</th>
<th>fluent</th>
</tr>
</thead>
<tbody>
<tr>
<td>mTS1</td>
<td>CAT</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>NULL</td>
<td>2</td>
</tr>
<tr>
<td>mTS1</td>
<td>HAT</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>NULL</td>
<td>1</td>
</tr>
<tr>
<td>mTS1</td>
<td>HAT</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>NULL</td>
<td>2</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Table 1. An Example of Input Data Sources for LR-DBN

The multiple subskills headers start with a prefix “kc_” as specified in Figure 2, and their values are observed as 1 for not required in current step or 2 for required. Values of fluent are observed as 1 for false and 2 for true. This follows the tradition that a Matlab cell’s indices starts from 1. Values for knowledge are set as NULL since they are not observed in the input evidences. Although we omit the time stamp in input data, it should be sorted by the latest attempting time for each user in advance.

There are three output files as specified in Figure 4. The param_table.xls outputs each user’s estimated parameters that were specified by a non-null name in the XML. Table 2 gives an example, where the #cases denotes how many cases have been read for each user in the input training data, and the ll denotes the maximized likelihood value by the EM algorithm.

<table>
<thead>
<tr>
<th>user</th>
<th>#cases</th>
<th>ll</th>
<th>L0_c</th>
<th>learn_c</th>
<th>forget_c</th>
<th>guess</th>
<th>slip</th>
</tr>
</thead>
<tbody>
<tr>
<td>mTS1</td>
<td>430</td>
<td>-7.38</td>
<td>3.66</td>
<td>4.32</td>
<td>0.32</td>
<td>0.20</td>
<td>0.04</td>
</tr>
<tr>
<td>mTS2</td>
<td>1348</td>
<td>-5.75</td>
<td>-2.98</td>
<td>4.46</td>
<td>0.03</td>
<td>0.21</td>
<td>0.08</td>
</tr>
<tr>
<td>mTU1</td>
<td>1100</td>
<td>-9.91</td>
<td>1.22</td>
<td>-0.24</td>
<td>1.23</td>
<td>0.30</td>
<td>0.05</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

The other two files are related to output the inference results. If we set the value of inference_result_header as the same as of the inference_result (as in Figure 4), it will generate a copy of the input test file but replace NULL knowledge with its inferred probability, as Table 3 shows.

<table>
<thead>
<tr>
<th>user</th>
<th>skill</th>
<th>kc_e</th>
<th>kc_a</th>
<th>kc_f</th>
<th>kc_h</th>
<th>knowledge</th>
<th>fluent</th>
</tr>
</thead>
<tbody>
<tr>
<td>mTS1</td>
<td>CAT</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>NULL</td>
<td>0.563</td>
</tr>
<tr>
<td>mTS1</td>
<td>CAT</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>NULL</td>
<td>0.994</td>
</tr>
<tr>
<td>mTS1</td>
<td>HAT</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>NULL</td>
<td>0.996</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Table 3. An Example of Output Inference Result for LR-DBN

Finally, a Matlab command invokes LR-DBN:
```
[property evidence hash .bnet] = RunBnet('property.xml');
```
Here property.xml is the XML specification file. BNT-SM’s RunBnet.m function then trains and tests the DBN.