Hierarchical Modeling of Label Dependency and Label Noise in Fine-grained Entity Typing

Junshuang Wu\textsuperscript{1}, Richong Zhang\textsuperscript{1*}, Yongyi Mao\textsuperscript{2}, Masoumeh Soflaei Shahrbabak\textsuperscript{2}, Jinpeng Huai\textsuperscript{1}

\textsuperscript{1}SKLSDE, School of Computer Science and Engineering, Beihang University, Beijing, China \\
\textsuperscript{2}School of Electrical Engineering and Computer Science, University of Ottawa, Ottawa, Canada \\
\{wujs,zhangrc,huaijp\}@act.buaa.edu.cn, ymao@uottawa.ca, msofl083@uottawa.ca

Abstract

Fine-grained entity typing (FET) aims to annotate the entity mentions in a sentence with fine-grained type labels. It brings plentiful semantic information for many natural language processing tasks. Existing FET approaches apply hard attention to learn on the noisy labels, and ignore that those noises have structured hierarchical dependency. Despite their successes, these FET models are insufficient in modeling type hierarchy dependencies and handling label noises. In this paper, we directly tackle the structured noisy labels by combining a forward tree module and a backward tree module. Specifically, the forward tree formulates the informative walk that hierarchically represents the type distributions. The backward tree models the erroneous walk that learns the noise confusion matrix. Empirical studies on several benchmark data sets confirm the effectiveness of the proposed framework.

1 Introduction

Fine-grained Entity Typing (FET) [Ling and Weld, 2012] aims at assigning proper fine-grained type labels for entity mentions in a sentence. The finely annotated mention type is often utilized to facilitate many downstream natural language processing (NLP) tasks, such as question answering [Dong et al., 2015], semantic parsing [Yavuz et al., 2016] and information extraction [Koch et al., 2014; Ling et al., 2015; Schütze et al., 2017].

Existing FET models face two challenges: modeling the type hierarchy dependency and reducing the label noise. Type hierarchy (or Taxonomy) is a natural and practical structure that used to organize the type labels (Concepts). Figure 1 illustrates a type hierarchy sampled from a FET data set. In the figure, type PERSON and its three sub-types ACTOR, AUTHOR and DIRECTOR form a hierarchical structure. Several methods have been proposed to model the type hierarchy dependency, including hierarchically embedding method [Ma et al., 2016], heuristic normalization loss [Xu and Barbosa, 2018] and hierarchical structure loss [Murty et al., 2018]. However, these models are limited by: (1) only modeling the parent-child relation, ignoring other relations (e.g. sibling). (2) using the same bilinear matrix or penalty hyper-parameter to model the correlations between different subtypes and their parent types. The AFET model [Ren et al., 2016] attempts to tackle these limitations by utilizing adaptive margins between types. The margin of two types is computed through their normalized number of shared entities in Knowledge Graph (KG). Nonetheless, the incompleteness and noisy annotations of KG may make those margins unreliable.

Label noise comes with the unreliable data acquisition processes such as distant supervision (DS) method [Mintz et al., 2009]. The DS method assigns all the possible types of the entity for its mentions in the corpus. However, an entity mention may belong to different types in different contexts. As is shown in Figure 1, the phrase Demi Lovato may refer to PERSON in a general sense, and refer to PERSON.ARTIST in a specific sense. But type ORGANIZATION can not be inferred from the given sentence, thus turns to a noisy type. Such noisy types may degrade the performance of the FET models. Although various approaches [Ren et al., 2016; Abhishek et al., 2017; Xu and Barbosa, 2018; Zhang et al., 2020] have been proposed to handle these noises, they ignore the fact that the noisy labels are structural. For instance, (PERSON.ACTOR, PERSON.SINGER) is a set of fine-grained types, where PERSON is in a higher level, and ACTOR/SINGER are in a lower level. NDP [Wu et al., 2019] considers such structural information by assigning type weights through random walking in the mention level. How-

\*Corresponding author: zhangrc@act.buaa.edu.cn

Figure 1: A type hierarchy sampled from the data set. The root node in the tree denotes mention Demi Lovato in the sentence “Don’t Forget” is the third and final single released from Demi Lovato’s debut album. It is labeled as PERSON, PERSON.ACTOR, PERSON.ARTIST and ORGANIZATION.
ever, it provides too much freedom in learning, thus may potentially risk over-fitting.

In this paper, we present a novel FET framework that jointly model the hierarchical dependency and the label noises of the types. This framework consists of a forward tree module (FTree) and a backward tree module (BTree). For modeling type hierarchical dependency, we first transform the pre-defined type hierarchy into the normalized type hierarchy that each leaf node represents a valid type. Then, the FTree module is proposed to compute the distribution of the type labels through random walking on the type hierarchy.

To model the label noises in other research topics, previous works [Schütze et al., 2017; Luo et al., 2017; Goldberger and Ben-Reuven, 2017] learn the transformation from the true labels to the observed labels. However, these models ignore the inherent hierarchical structure. We provide an alternative that parameterizes this transformation via the random walk process on the type hierarchy tree. We formulate the perturbation process with the BTree module. It learns the transformation from the latent true labels to observed noisy labels by backward random walking on type hierarchy. It parametrizes the confusion matrix of the type labels under the assumption that a label is likely to be perturbed into its nearby label in the tree. This approach puts an additional constraint on the confusion matrix according to the structure of the type hierarchy, thereby restricting the model capacity.

The empirical studies confirm the capability of our model in modeling the type hierarchy and discovering the true labels from the noisy data. The extensive ablation studies verify the effectiveness of each component of our model.

2 Related Work

2.1 Hierarchical Type Modeling

The most earlier models ignore the type hierarchy and treat this problem as a multi-label classification task [Ling and Weld, 2012; Yogatama et al., 2015; Abhishek et al., 2017]. Recently, some studies [Yosef et al., 2012; Ren et al., 2016; Shimaoka et al., 2017; Xu and Barbosa, 2018; Murty et al., 2018; Chen et al., 2020] have made efforts to explicitly model the correlation among hierarchical types. These models, although demonstrating some successes, rely on additional loss function constraints and some heuristic procedures to walk along the type hierarchy to find the predicted type. In this paper, we propose a generic method that directly models the hierarchical structure without additional steps for inferring or loss functions for model training.

2.2 Label Noise Handling

Various methods have been proposed to deal with noisy training samples. Specifically, [Gillick et al., 2014] refined the training data by applying a set of heuristics. This method, however, reduces the size of the training set and leads to performance degradation, as seen in [Ren et al., 2016].

Recently, many methods have been proposed to avoid the deletion of training samples or pruning type hierarchy. Some works [Yogatama et al., 2015; Ma et al., 2016] treat clean and noisy entity mentions equally and utilize ranking loss to attenuate the effects of extraneous labels. However, some studies [Ren et al., 2016; Abhishek et al., 2017; Xu and Barbosa, 2018] deal with the problem by modeling noisy and clean entity mentions separately and force the model to choose the most relevant type as the true label for noisy samples during the training process. NDP [Wu et al., 2019] proposed to weight out the noisy type during the training process. NFETC-AR [Zhang et al., 2020] directly estimated the pseudo-truth type distribution from noisy type distribution by minimizing the KL-divergence.

Although existing models have shown effectiveness in handling label noise, the noise detection and the hierarchical classification are modeled separately. In addition, the cleaned data should be feed-forward to the original model to improve the model trained from the noisy data. We argue, in this paper, these facts should be considered when building FET models.

3 Model

3.1 Problem Definition

In this section, we formally describe the fine-grained entity typing (FET). Given a pre-defined type label set with a hierarchical structure $T$ and training data $D = \{(s_i, m_i, \tau_i), i \in \{1, 2, \cdots, N\}\}$, where $s_i$ denotes a sentence, $m_i$ represents an entity mention, and $\tau_i$ denotes the noisy type label set. The objective of the FET task is to train a predictive model on $D$ that is capable of predicting the true type for any sentence-mention pair $(s, m)$, despite the fact that $D$ is noisy.

For convenience, we denote a $(s, m)$ pair by random variable $X$, and denote the true label by random variable $Y$.

3.2 Sketch of Overall Model

Overall our proposed model assumes the following generative process from $X$ to $Z$. Assume that a random walker is to walk on the type hierarchy tree $T$ from the root. It assumes a probabilistic walk strategy generated from $X$. Following the strategy, at each node of $T$, it may keep going down along a downstream branch or may choose to stop. Whenever it stops, the node it lands on is the true type $Y$. We will call this walk “informative walk”.

But the true type $Y$ can not be directly observed. It then assumes another random walk strategy. At each node of $T$, it may walk along any direction (even going back) or choose to stop following this strategy. The node it stops at is then the observed noisy type $Z$. We will call this walk “erroneous walk”. The process of informative walk and erroneous walk is formulated as follows:

$$P_{Z|X}(z|x) = \sum_{y \in T} P_{Z|Y,X}(z|y, x)P_{Y|X}(y|x) \quad (1)$$

In this paper, we assume the erroneous walk process is independent with input sentence $x$, we have

$$P_{Z|X}(z|x) = \sum_{y \in T} P_{Z|Y}(z|y)P_{Y|X}(y|x) \quad (2)$$

Note that the move from the true type $Y$ to the noisy label $Z$ can be aggregated to a transition matrix (also called confusion matrix) $A$. Choosing a strategy for the erroneous walk is equivalent to parameterizing the matrix $A$. Then we can get:

$$P_{Z|X}(z|x) = AP_{Y|X}(z|x) \quad (3)$$


where \( P_{Y|X}(\cdot|x) \) is the conditional distribution of \( Y \) given \( X = x \) written as a vector, and \( P_{Z|X}(\cdot|x) \) is defined as similarly. We postpone defining the loss function later.

To define and implement this model, we first introduce a "normalized type hierarchy" from previous work [Wu et al., 2019] on which informative walk and erroneous walk can be interpreted as walking forward and walking backward.

### 3.3 Normalized Type Hierarchy

Let \( T \) denotes the type hierarchy of the given entity mention. Each node in \( T \) denotes a token from a type name. For example, an entity mention could belong to PERSON or PERSON.ACTOR, where these types are in the different levels of the hierarchy \( T \). Thus, models must take necessary measures to predict the proper grained type. In this paper, we use the same normalized type hierarchy as the previous work [Wu et al., 2019]. Specifically, we transform a hierarchical structure \( T \) into the normalization type hierarchy \( \tilde{T} \) by adding a \textit{stop} node as the child of each non-leaf node. For example, after type hierarchy normalization, node PERSON \( T \) is followed by a new node PERSON.PER-STOP, as shown in the middle picture of Figure 2. Normalizing type hierarchy allows every type being mapped to a unique leaf node in \( \tilde{T} \), thus maintains the information of the original type in its hierarchy level.

For later use, we denote a node in the normalized type hierarchy as \( u \in \tilde{T} \), and denote a type as \( t \). Each type \( t \) is a path from the root to a leaf node in \( \tilde{T} \). From here on, whenever we speak of a type, we refer to its leaf-node representation in \( \tilde{T} \).

### 3.4 Feature Embedding

Give the \((s, m)\) pair, we utilize the neural network proposed in NFETC [Xu and Barbosa, 2018] to extract the feature embedding \((s, m)\) as described in the following.

Given the entity mention \( m = \{w_p, \ldots, w_l\} \), the average feature \( \bar{m} \) is the average word embedding of the word in \( m \). Moreover, the LSTM is applied to the extended mention \( m^* = \{w_{p-1}, \ldots, w_{l+1}\} \) and generates the outputs \( \{h_{p-1}, \ldots, h_{l+1}\} \). The last output \( h_{l+1} \) is regarded as the LSTM representation \( \bar{h}(m) \) of \( m \).

The Bidirectional LSTM combined with a word-level attention module is applied to the context sequence \( c = \{w_{p-k}, \ldots, w_{l+k}\} \) to generate the context feature \( \bar{h}(c) \), where \( k \) is the context window size.

Then, the final feature embedding is the concatenation of the above three features: \( x = (s, m) = [\bar{m}, \bar{h}(m), \bar{h}(c)] \).

### 3.5 Modelling Informative Walk

During the informative walk, the random walker starts walking from the root node and is allowed to only stop on a leaf node. For each node \( u \in \tilde{T} \), we define its token embedding as \( \bar{u} \in \mathbb{R}^{d_u \times 1} \). Then we measure the similarity between an input \( x \) and a node \( u \) via a score function

\[
\phi(x, u) := x^T W \bar{u}
\]

where \( W \in \mathbb{R}^{d_x \times d_u} \) denotes a trainable matrix.

To compute the weights of edges in \( \tilde{T} \), we introduce a notation \( H_u \) to denote the set of all child nodes of node \( t \) together. Then the probability of the informative walker walking from a father node \( u_i \) to a child node \( u_j \) is calculated as

\[
q^{(x)}(u_j|u_i) = \frac{e^{\phi(x, u_j)}}{\sum_{u_k \in H_{u_j}} e^{\phi(x, u_k)}}
\]

Given a type path \( t = \{u_0, u_1, \ldots, u_L\} \), we then define the the probability of the walker arriving at \( t \) given \( x \) as

\[
p^{(x)}(t) := \prod_{l=1}^{L} q^{(x)}(u_l|u_{l-1})
\]

where \( L \) is the depth of \( \tilde{T} \) and \( u_0 \) is the root node. Then the true probability of type distribution given the input \( x \) is defined as:

\[
P_{Y|X}(\cdot|x) := [p^{(x)}(t_1), \ldots, p^{(x)}(t_J)]
\]

where \( J \) denotes the number of all types.

The informative-walk component is referred to as “Forward Tree” or FTree.

### 3.6 Modelling Erroneous Walk

We next describe the “backward tree” or BTree, which models the perturbation process using the erroneous walk. It approximates the confusion matrix \( A \) that transforms the true labels \( Y \) to the noise label \( Z \).
Backward Tree Construction
The backward tree is constructed for every type to explicitly modeling backtracking trace from a leaf node to another leaf node. Figure 2 shows a backward tree for a random walker starting from the leaf token of type path “PERSON.ACTOR”.

The root of backward tree for type \( t_i = \{ u^{(i)}_0, \cdots, u^{(i)}_L \} \) is the leaf token \( u^{(i)}_L \). For convenience, we virtually add a blue square token to denote keeping staying in this leaf token. When the random walker lands on a leaf token, it has only two choices, staying there or back-walk to its parent token. When it goes back to its parent, in the next step, it can continue to go back to the parent token or a child token of the current token until reaching the leaf token \( u^{(i)}_0 \). This process models the type perturbation from type \( t_i \) to type \( t_j \).

There are five types of backtracking edges: (1) The edge from the starting token to a square token; (2) The edge from a token to its father token; (3) The edge from a token to its child token, (4) the edge from the level 1 token (such as PERSON) to another level 1 token (such as ORGANIZATION) through dummy token and (5) The edge from a non-starting leaf token to a square token. For each type of backtracking edge, we use a learnable parameter to describe the tendency that the random walker walking through this edge. Specifically, as shown in the right picture of Figure 2, the parameter of the first four types of backtracking edges are \( S, F, C, \) and \( M \), and we fix this probability for the fifth edge as one to restrict a random walker from backtracking again. Note that for the level 1 type token (such as PERSON, ORGANIZATION), its father token is a dummy token which has no corresponding type token. It forces the random walker not to stop and continue walking to one of its children tokens.

Type Confusion Matrix Generation
The backtracking process can be expressed as a type confusion matrix \( A \in \mathbb{R}^{J \times J} \), where \( J \) is the number of all types. To model the confusion between type \( t_i \) and \( t_j \), we imagine an erroneous walker on the backward tree \( T^*_i \) starts from type \( t_i \)'s leaf node and stop on \( t_j \)'s leaf node. The probability of the erroneous walker lands on \( t_j \) leaf token is the exact element \( A_{i,j} \) in the confusion matrix \( A \). To compute \( A_{i,j} \), we define the probability of the erroneous walker which starts from \( u^{(i)}_L \) walking through edge \( r(u^{(i)}_t, u^{(i)}_j) \) from a father node \( u^{(i)}_t \) to a child node \( u^{(i)}_j \) is calculated as:

\[
a_{r(u^{(i)}_t, u^{(i)}_j)}^{(i)} = \frac{e^{r(u^{(i)}_t, u^{(i)}_j)}}{\sum_{u^{(i)}_0 \in \mathcal{H}_{u^{(i)}_t}} e^{r(u^{(i)}_t, u^{(i)}_0)}}
\]

where \( \mathcal{H}_{u^{(i)}_t} \) denotes the set of all child nodes of node \( u^{(i)}_t \) and \( r(u^{(i)}_t, u^{(i)}_0) \in \{F^{(i)}_t, C^{(i)}_t, S^{(i)}_t, M^{(i)}_t, 1\} \) denotes the weight that erroneous walker of type \( t_i \) chooses edge \( r(u^{(i)}_t, u^{(i)}_0) \). These weight parameters are randomly initialized and updated during training. The confusion probability between type \( t_i \) and type \( t_j \) is then defined as:

\[
A_{i,j} = \prod_{r \in R_{i \rightarrow j}} a_{r(u^{(i)}_t, u^{(i)}_j)}^{(i)}
\]

where \( R_{i \rightarrow j} \) denotes the edge set from type \( t_i \) to type \( t_j \). Similarly, we can get the confusion probabilities among all types.

### 3.7 Loss Function
With \( P_{Y|X} \) defined via the FTree and \( P_{Z|Y} \) or (matrix \( A \)) defined by BTree, the model is naturally optimized by minimizing the negative log-likelihood of the training set. We define the loss function as follows:

\[
\mathcal{L} = \sum_{i=1}^{N} -\log P_{Z|X}(z^*_i | x_i)
\]

where \( z^*_i = \arg \max_{z \in \mathcal{Y}} P_{Z|X}(z | x_i) \) and \( N \) is the total training sample numbers. When \( Y \) is completely hidden the model, such a loss is non-identifiable, and there are uncountable solutions for \( P_{Y|X} \) that will equally minimize the loss. To fix this, we proposed following revision of loss function:

\[
\mathcal{L} = \sum_{i \in \mathcal{N}} -\log P_{Z|X}(z_i^* | x_i) + \sum_{i \in \mathcal{C}} -\log P_{Y|X}(y_i | x_i)
\]

where \( \mathcal{N} \) denotes the noisy training sample set, in which training example may have been labeled with more than one type, and \( C \) denotes the clean training sample set that means a training example is labeled only with one type, and \( y_i \) is the only type label in clean type set \( \tau_i \). In this loss function, we essentially have assumed that the examples in \( C \) have clean labels. In the other words, we assume that the sample with only one type path is labeled with the true type. This will allow the model to anchor the solution of \( P_{Y|X} \) around those compatible with the examples in \( C \).

### 3.8 Type Inference
In the training process, we jointly train the FTree and BTree. We denote the joint model of FTree and BTree by FBTree. During type inference, we directly utilize the type probability distribution of FTree as the predictive probability. The type \( t \) with the highest probability is then declared as the true type.
Three benchmark datasets, Wiki, OntoNotes, and BBN are used to evaluate our proposed models. We utilize the preprocessed data set provided by [Ren et al., 2016]. Following the standard protocol adopted in the previous works [Ren et al., 2016; Abhishek et al., 2017; Shimaoka et al., 2017], we split 10% instances from each test set as the validation set. Table 1 shows the statistics of the three datasets. As shown in Table 1, nearly 30% of the training samples contain noisy labels among all datasets.

In our implementation, Glove embedding [Pennington et al., 2014] is exploited to initialize the word embeddings and they are fixed during the training process. We utilize Adam [Kingma and Ba, 2014] as the model optimizer. An early-stop strategy is employed to terminate training when the loss on validation set is stable. Dropout [Srivastava et al., 2014] and L2 regularization are used to alleviate overfitting. We determine the optimal hyper-parameters by random search on the valid dataset. The main hyper-parameters are shown in Table 3. Different datasets may set different hyper-parameters because they vary in structures and scales.

4.2 Predictive Comparison

We compare our proposed model FTree, FBTree with various recent FET models: ATTN [Shimaoka et al., 2017], ABH, ABH-AIC [Abhishek et al., 2017], NFETC [Xu and Barbosa, 2018], NDP [Wu et al., 2019], NFETC-AR [Zhang et al., 2020] and MLL2R [Chen et al., 2020]. We take the evaluation results of these baselines from their original papers.

Three standard metrics [Ling and Weld, 2012], Accuracy (Acc), Micro-averaged F1 (Mi-F1) score and Macro-F1 (Ma-F1) score, are used to evaluate the performance of models.

The overall performances are presented in Table 2, where the best value is highlighted in bold. We observed that our FBTree model outperforms all compared models on Wiki and BBN and achieves comparable performance with the SOTA model on OntoNotes. These results are desirable as we note the following. The testing set of BBN is considered noisier, or as the true type probability. But the testing sets of Wiki and OntoNotes are in fact noisy, somewhat inconsistent from the ground truth. This makes our model perform better on BBN.

It is worth noting that adding BTree module can significantly improve the performance of the FTree module. It also achieves more stable performance across all datasets than all other models.

4.3 Performances at Different Levels of Type Hierarchy

We further investigate the performances of the compared models at each level of the type hierarchy. The depth of type hierarchy for Wiki, OntoNotes, and BBN are 2, 3, and 2. The results are given in Figure 3. We find that with the increasing of the label level of the tested entities, the performance on Wiki and OntoNotes is decreasing. However, the performance of all compared models on BBN does not varies significantly. This phenomenon might be caused by the level distribution in the training set. It can also be observed that FBTree achieves the best performances at all levels of all datasets. This study confirms the superior of FBTree.

4.4 Ablation Study

Effectiveness of informative walk. We conduct experiments to evaluate the effect of our mention-to-type compatibility model FTree. We compare our FTree model with Complex utilize complex bilinear mappings to capture hierarchical information [Murty et al., 2018]. The results in

Table 3: Hyper-parameters for our proposed model
Table 4: Performance of variant FET models

<table>
<thead>
<tr>
<th>Model</th>
<th>Wiki</th>
<th>OntoNotes</th>
<th>BBN</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Acc</td>
<td>Ma-F1</td>
<td>Mi-F1</td>
</tr>
<tr>
<td>CompIEx</td>
<td>68.3±1.3</td>
<td>81.7±0.8</td>
<td>79.1±0.7</td>
</tr>
<tr>
<td>CompIEx+FLAT</td>
<td>67.9±0.5</td>
<td>81.8±0.2</td>
<td>79.3±0.3</td>
</tr>
<tr>
<td>CompIEx+TypeCorr</td>
<td>69.1±1.0</td>
<td>82.3±0.8</td>
<td>79.5±0.8</td>
</tr>
<tr>
<td>CompIEx+BTTree</td>
<td>69.1±0.3</td>
<td>82.0±0.5</td>
<td>79.9±0.6</td>
</tr>
<tr>
<td>FTree</td>
<td>68.5±0.9</td>
<td>82.3±0.4</td>
<td>79.4±0.3</td>
</tr>
<tr>
<td>FTree+FLAT</td>
<td>69.2±0.7</td>
<td>81.3±0.8</td>
<td>78.1±0.8</td>
</tr>
<tr>
<td>FTree+TypeCorr</td>
<td>67.2±0.5</td>
<td>80.0±0.7</td>
<td>77.0±0.5</td>
</tr>
<tr>
<td>FBTree</td>
<td>70.7±1.2</td>
<td>83.7±0.9</td>
<td>80.9±0.9</td>
</tr>
</tbody>
</table>

Table 5: Error analysis for FTree and FBTree on Acc metric

<table>
<thead>
<tr>
<th>Model</th>
<th>Dataset</th>
<th>AW</th>
<th>PW</th>
<th>C2F</th>
<th>F2C</th>
</tr>
</thead>
<tbody>
<tr>
<td>FTree</td>
<td>Wiki</td>
<td>12.5</td>
<td>1.8</td>
<td>5.3</td>
<td>11.9</td>
</tr>
<tr>
<td></td>
<td>OntoNotes</td>
<td>16.1</td>
<td>1.4</td>
<td>3.3</td>
<td>17.3</td>
</tr>
<tr>
<td></td>
<td>BBN</td>
<td>16.5</td>
<td>3.7</td>
<td>2.5</td>
<td>1.8</td>
</tr>
<tr>
<td>FBTree</td>
<td>Wiki</td>
<td>11.1</td>
<td>2.0</td>
<td>3.8</td>
<td>12.4</td>
</tr>
<tr>
<td></td>
<td>OntoNotes</td>
<td>15.0</td>
<td>1.5</td>
<td>3.4</td>
<td>16.0</td>
</tr>
<tr>
<td></td>
<td>BBN</td>
<td>15.1</td>
<td>3.6</td>
<td>3.5</td>
<td>0.5</td>
</tr>
</tbody>
</table>

Table 4 indicates that FTree is more effective than CompIEx in modeling label dependencies.

Effectiveness of erroneous walk. We further analyze the results of our BTree model. As shown in Table 4, we compare the type confusion matrix learned by the highly constrained model BTree with FLAT and TypeCorr. FLAT denotes that confusion matrix is random initialized and TypeCorr regard the normalized number of shared entities in KG between types as the type confusion weight [Ren et al., 2016]. From the table, we observe that the type confusion matrices learned by the two baselines may not bring improvements on all data sets. But our BTree model brings significant performance gains on all data sets. This experiment result confirms that BTree is efficient in noise detection.

4.5 Analysis

We make an error analysis on the FTree and FBTree model. As shown in Table 5, we classify errors into four types: all wrong (AW), partial wrong (PW), coarse to fine (C2F), and fine to coarse (F2C). The results show that FBTree decreases AW and C2F errors by about 1%, 0.7%, with a slight increasing in PW and F2C on the Wiki dataset. It also decreases AW and F2C by about 1% with a slight increasing in PW and F2C on OntoNotes and BBN. This error analysis confirms that BTree can effectively detect noises and confirms the motivation of our proposed model.

Three noisy examples in the BBN data set are used to demonstrate the effectiveness of FBTree. As shown in Table 6, BTree tends to produce low label scores in all samples. For sample II and III, the classifier can detect the two noisy types by their lower score than the most appropriate type. We also find that it is possible for the distributions $P_{Z|X}$ and $P_{Y|X}$ to have different polarity e.g. sample I. This study indicates that FBTree tries to reduce the confidence of the classifier on noisy samples, thus has superior generalization.

5 Conclusion

In this work, we identify that the entanglement of type hierarchy and noisy labels is a key obstacle for developing a high-performance predictive FET models. We tackle this problem by separately modelling the two aspects using a forward random walk on the type hierarchy and a backward random walk on a reversed tree. The two components are then used to jointly model the noisy hierarchical types that are observed in the datasets. Interestingly, we found that the random walk process is able to estimate the true entity type more than we expected, by evaluating the importance of the noisy entity types through the walk process. The empirical study confirms the effectiveness of our proposed model, outperforming recent models on benchmark datasets.

It is also worth noting that the modelling techniques presented in this work can be extended to model labels with hierarchical structure and observation noise. We hope this work may inspire more exploitation of such ideas to much broader scope of applications.

Acknowledgments

This work is supported partly by the National Natural Science Foundation of China (No. 61772059), by the Fundamental Research Funds for the Central Universities, by the Beijing S&T Committee (No. Z191100008619007) and by the State Key Laboratory of Software Development Environment (No. SKLSDE-2020ZX-14).
References


[Ren et al., 2016] Xiang Ren, Wenqi He, Meng Qu Lifu Huang, Heng Ji, and Jiawei Han. AFET: automatic fine-grained entity typing by hierarchical partial-label embedding. In Proceedings of EMNLP, pages 1369–1378, 2016.


