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

Junshuang Wu[1] , **Richong Zhang**[1*] , **Yongyi Mao**[2] , **Masoumeh Soflaei Shahrbabak**[2] , **Jinpeng Huai**[1]

[1]SKLSDE, School of Computer Science and Engineering, Beihang University, Beijing, China
[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 presented 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 relaitons (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.

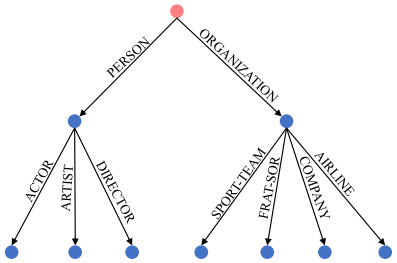


Figure 1: A type hierarchy sampled from the data set. The root node in the tree denotes mention *Demi Lovato* in the sentence *"Don't Forget" is the third and final single released from Demi Lovato's debut album.* It is labeled as PERSON, PERSON.ACTOR, PERSON.ARTIST and ORGANIZATION.

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

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 inherentent 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-backed 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 $\mathcal{D} = \{(s_i, m_i, \tau_i), i \in \{1, 2, \cdots, N\}\}$, where s_i denotes a sentence, m_i represents an entity mention, and τ_i denotes the noisy type label set. The objective of the FET task is to train a predictive model on $\mathcal{D}$ that is capable of predicting the *true type* for any sentence-mention pair (s, m), despite the fact that $\mathcal{D}$ 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:

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

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

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

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:

$$P_{Z|X}(\cdot|x) = A P_{Y|X}(\cdot|x) \tag{3}$$

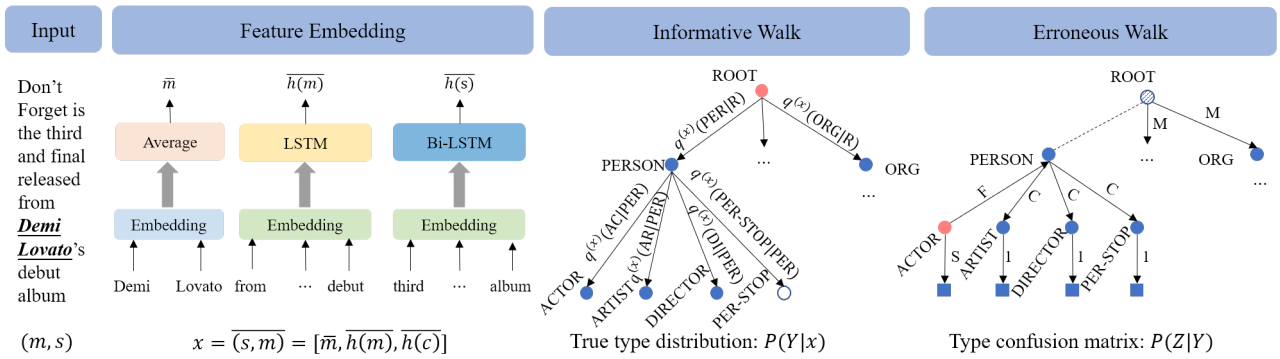


Figure 2: Left: The Feature Embedding process. Middle: An example of Forward Tree. Right: An example of Backward Tree construction

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 $\mathcal{T}$ by adding a *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 $\mathcal{T}$, 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 $u \in \mathcal{T}$, and denote a type as t. Each type t is a path from the root to a leaf node in $\mathcal{T}$. From here on, whenever we speak of a type, we refer to its leaf-node representation in $\mathcal{T}$.

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 $\overline{(s, m)}$ as described in the following.

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

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

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

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 $u \in \mathcal{T}$, we define its token embedding as $\overline{u} \in \mathbb{R}^{d_u \times 1}$. Then we measure the similarity between an input x and a node u via a score function

$$\phi(x, u) := x^{\mathsf{T}} W \overline{u} \tag{4}$$

where $W \in \mathbb{R}^{d_s \times d_u}$ denotes a trainable matrix.

To compute the weights of edges in $\mathcal{T}$, we introduce a notation $\mathcal{H}_u$ 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

$$q^{(x)}(u_j|u_i) = \frac{e^{\phi(x,u_j)}}{\sum_{u_k \in \mathcal{H}_{u_j}} e^{\phi(x,u_k)}} \tag{5}$$

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

$$p^{(x)}(t) := \prod_{i=1}^{L} q^{(x)}(u_i|u_{i-1}) \tag{6}$$

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

$$P_{Y|X}(\cdot|x) := [p^{(x)}(t_1), \cdots, p^{(x)}(t_J)] \tag{7}$$

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 $t_i = \{u_o^{(i)}, \cdots, u_L^{(i)}\}$ is the leaf token $u_L^{(i)}$. 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 $u_L^{(j)}$. 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 $\mathcal{T}_i$ 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 exactly the element A_{ij} in the confusion matrix A. To compute A_{ij}, we define the probability of the erroneous walker which starts from $u_L^{(i)}$ walking through edge $r(u_{i'}, u_{j'})$ from a father node $u_{i'}$ to a child node $u_{j'}$ is calculated as:

$$a^{(i)}_{r(u_{i'},u_{j'})} = \frac{e^{\kappa_{r(u_{i'},u_{j'})}}}{\sum_{u \in \mathcal{H}^b_{u_{i'}}} e^{\kappa_{r(u_{i'},u)}}} \tag{8}$$

where $\mathcal{H}^b_{u_{i'}}$ denotes the set of all child nodes of node $u_{i'}$ and $\kappa_{r(u_{i'},u)} \in \{\mathtt{F}^{(\mathtt{i})}, \mathtt{C}^{(\mathtt{i})}, \mathtt{S}^{(\mathtt{i})}, \mathtt{M}^{(\mathtt{i})}, \mathtt{1}\}$ denotes the weight that erroneous walker of type t_i chooses edge $r(u_{i'}, u)$. 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:

$$\hat{A}_{i,j} = \prod_{r \in R_{i \to j}} a_r^{(i)} \tag{9}$$

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

Datasets	Wiki (FIGER)	OntoNotes	BBN
#Type hierarchy depth	2	3	2
#Type tokens	177	141	63
#`STOP` tokens	49	52	16
#Leaf types	128	89	47
#Training ment	2.69M	220,398	86,078
#Test ment	563	8,963	12,845
%Noisy training ment	35.42	27.4	24.08
%Noisy test sent	11.72	6.23	0

Table 1: Statistics of datasets

The whole confusion matrix is computed as:

$$A = \hat{A} + \epsilon * I_J \tag{10}$$

where the parameter $\epsilon \in (0, 1)$ is a hyper-parameter that restricts the the training process from over-fitting.

So far, $P_{Z|X}$ is well defined through Equation 3. It is noteworthy that the confusion matrix generated by BTree is highly constrained and independent from the FTree. Apart from FTree, it can be applied to other mention-to-type compatibility models.

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:

$$\mathcal{L} = \sum_{i=1}^{N} -\log P_{Z|X}(z^*|x_i) \tag{11}$$

where $z_i^* = \arg\max_{z \in \tau_i} P_{Z|X}(z|x_i)$ 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:

$$\mathcal{L} = \sum_{i \in \mathcal{N}} -\log P_{Z|X}(z_i^*|x_i) + \sum_{i \in \mathcal{C}} -\log P_{Y|X}(y_i|x_i) \tag{12}$$

where $\mathcal{N}$ denotes the noisy training sample set, in which training example may have been labeled with more than one type, and $\mathcal{C}$ denotes the clean training sample set that means a training example is labeled only with one type, y_i is the only type label in clean type set τ_i. In this loss function, we essentially have assumed that the examples in $\mathcal{C}$ 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 $\mathcal{C}$.

3.8 Type Inference

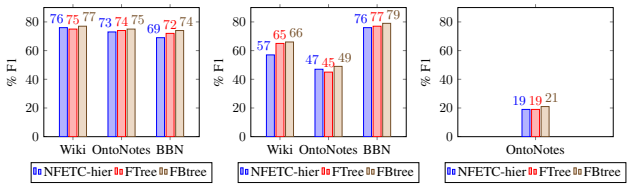
In the training process, we jointly train the FTree and BTree. We denote the joint model of FTree and BTree by FBTree. During type inference, we directly utilize the type probability distribution of FTree as the predictive probability. The type t with the highest probability is then declared as the true type.

Model	Wiki			OntoNotes			BBN		
	Acc	Ma-F1	Mi-F1	Acc	Ma-F1	Mi-F1	Acc	Ma-F1	Mi-F1
AFET	53.3	69.3	66.4	55.1	71.1	64.7	67.0	72.7	73.5
ABH	65.8	81.2	77.4	52.2	68.5	63.3	60.4	74.1	75.7
NFETC	56.2 ± 1.0	77.2 ± 0.9	74.3 ± 1.1	54.8 ± 0.4	71.8 ± 0.4	65.0 ± 0.4	73.8 ± 0.6	78.4 ± 0.6	78.9 ± 0.6
NFETC-hier	68.9 ± 0.6	81.9 ± 0.7	79.0 ± 0.7	60.2 ± 0.2	76.4 ± 0.1	70.2 ± 0.2	73.9 ± 1.2	78.8 ± 1.2	79.4 ± 1.1
NDP	67.7	81.8	78	58	71.2	64.8	72.7	76.4	77.7
MLL2R (exclusive)	69.1	82.6	80.8	58.3	72.4	67.2	48.2	63.2	61.0
MLL2R (undefined)	65.5	80.5	78.1	58.7	73.0	68.1	75.2	79.7	80.5
NFETC-AR	58.1 ± 1.1	79.0 ± 0.4	76.1 ± 0.4	62.8 ± 0.4	77.8 ± 0.4	71.8 ± 0.5	76.7 ± 0.2	81.4 ± 0.3	81.5 ± 0.3
NFETC-AR-hier	70.1 ± 0.9	83.2 ± 0.7	80.1 ± 0.6	64.0 ± 0.3	$\mathbf{78.8 \pm 0.3}$	$\mathbf{73.0 \pm 0.3}$	74.9 ± 0.6	80.4 ± 0.6	80.3 ± 0.6
FTree	68.5 ± 0.9	82.3 ± 0.4	79.4 ± 0.3	62.0 ± 0.4	77.5 ± 0.3	71.4 ± 0.3	75.5 ± 0.2	80.4 ± 0.2	80.7 ± 0.2
FBTree	$\mathbf{70.7 \pm 1.2}$	$\mathbf{83.7 \pm 0.9}$	$\mathbf{80.9 \pm 0.9}$	$\mathbf{64.0 \pm 0.1}$	78.4 ± 0.2	72.5 ± 0.2	$\mathbf{77.3 \pm 0.4}$	$\mathbf{82.0 \pm 0.3}$	$\mathbf{82.2 \pm 0.3}$

Table 2: Performance of FET models on Wiki, OntoNotes and BBN datasets

Hyper-parameter	Wiki	OntoNotes	BBN
Learning rate	0.0002	0.0006	0.0007
Batch size	512	512	512
Hidden size	180	440	180
L2-loss weight	{1e-4,2e-4,3e-4}		
ϵ	{0.1,0.2,0.3,0.4,0.5}		
Pos emb size	85	70	20

Table 3: Hyper-parameters for our proposed model



(a) Level 1 (b) Level 2 (c) Level 3

Figure 3: Performance of different level types on three test data sets

4 Experiment

4.1 Datasets and Settings

Three benchmark datasets, Wiki, OntoNotes, and BBN are used to evaluate our proposed models. We utilize the pre-processed 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 over-fitting. 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-AllC [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 noiseless, 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 model 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 that utilizes complex bilinear mappings to capture hierarchical information [Murty *et al.*, 2018]. The results in

Model	Wiki			OntoNotes			BBN		
	Acc	Ma-F1	Mi-F1	Acc	Ma-F1	Mi-F1	Acc	Ma-F1	Mi-F1
ComplEx	68.3 ±1.3	81.7±0.8	79.1±0.7	57.6±0.2	71.9±0.3	67.2±0.3	73.6±1.1	78.3±1.1	78.8±1.0
ComplEx+FLAT	67.9±0.5	81.8±0.2	79.3±0.3	58.7±0.4	74.0±0.4	67.6±0.4	74.8±1.1	79.2±1.1	79.8±1.1
ComplEx+TypeCorr	69.1±1.0	82.3±0.8	79.5±0.8	59.3±0.5	74.5±0.5	68.2±0.5	75.2±0.5	79.1±0.5	79.6±0.5
ComplEx+BTree	69.1±0.3	82.0±0.5	79.9±0.6	59.7±0.2	74.5±0.1	67.9±0.1	75.9±0.6	80.9±0.5	81.1±0.4
FTree	68.5±0.9	82.3±0.4	79.4±0.3	62.0±0.4	77.5±0.3	71.4±0.3	75.5±0.2	80.4±0.2	80.7±0.2
FTree+FLAT	69.2±0.7	81.3±0.8	78.1±0.8	62.0±0.1	77.4±0.2	71.3±0.2	75.4±0.2	80.2±0.2	80.5±0.2
FTree+TypeCorr	67.2±0.5	80.0±0.7	77.0±0.5	61.7±0.4	77.0±0.4	70.9±0.5	75.6±0.1	80.5±0.1	80.7±0.2
FBTree	**70.7±1.2**	**83.7±0.9**	**80.9±0.9**	**64.0±0.1**	**78.4±0.2**	**72.5±0.2**	**77.3±0.4**	**82.0±0.3**	**82.2±0.3**

Table 4: Performance of variant FET models

Model	Dataset	AW	PW	C2F	F2C
FTree	Wiki	12.5	1.8	5.3	11.9
	OntoNotes	16.1	1.4	3.3	17.3
	BBN	16.5	3.7	2.5	1.8
FBTree	Wiki	11.1	2.0	3.8	12.4
	OntoNotes	15.0	1.5	3.4	16.0
	BBN	15.1	3.6	3.5	0.5

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

Table 4 indicates that FTree is more effective than ComplEx 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 leaned 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 C2F 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 indicate that FBTree tries to reduce the confidence of the classifier on noisy samples, thus has superior generalization.

I: Coarse		
We've told Senator Pryor isn't yet a co-sponsor, but if $\cdots$		
Annotation Label	$P_{Y\|X}$	$P_{Z\|X}$
PERSON	0.774	0.066
ORG.GOVERNMENT	0.126	0.079
II: Fine		
Now a new law in Texas seems to be providing the proof.		
Annotation Label	$P_{Y\|X}$	$P_{Z\|X}$
LOCATION	0.004	0.023
GPE.STATE_PROVINCE	0.923	0.513
III: Siblings		
... propose new generation of jet trainers for the U.S. Air Force		
Annotation Label	$P_{Y\|X}$	$P_{Z\|X}$
ORG.GOVERNMENT	0.902	0.507
ORG.CORPORATION	0.097	0.068

Table 6: The predicted type scores of FTree and FBTree-LA. The words with a underline represents the mentions to be labeled

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

[Abhishek *et al.*, 2017] Abhishek, Ashish Anand, and Amit Awekar. Fine-grained entity type classification by jointly learning representations and label embeddings. In *Proceedings of EACL*, pages 797–807, 2017.

[Chen *et al.*, 2020] Tongfei Chen, Yunmo Chen, and Benjamin Van Durme. Hierarchical entity typing via multi-level learning to rank. In *Proceedings of ACL*, pages 8465–8475. ACL, 2020.

[Dong *et al.*, 2015] Li Dong, Furu Wei, Hong Sun, Ming Zhou, and Ke Xu. A hybrid neural model for type classification of entity mentions. In *Proceedings of IJCAI*, pages 1243–1249, 2015.

[Gillick *et al.*, 2014] Dan Gillick, Nevena Lazic, Kuzman Ganchev, Jesse Kirchner, and David Huynh. Context-dependent fine-grained entity type tagging. *CoRR*, abs/1412.1820, 2014.

[Goldberger and Ben-Reuven, 2017] Jacob Goldberger and Ehud Ben-Reuven. Training deep neural-networks using a noise adaptation layer. In *Proceedings of ICLR*. OpenReview.net, 2017.

[Kingma and Ba, 2014] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014.

[Koch *et al.*, 2014] Mitchell Koch, John Gilmer, Stephen Soderland, and Daniel S. Weld. Type-aware distantly supervised relation extraction with linked arguments. In *Proceedings of EMNLP*, pages 1891–1901, 2014.

[Ling and Weld, 2012] Xiao Ling and Daniel S. Weld. Fine-grained entity recognition. In *Proceedings of AAAI*, 2012.

[Ling *et al.*, 2015] Xiao Ling, Sameer Singh, and Daniel S. Weld. Design challenges for entity linking. *TACL*, 3:315–328, 2015.

[Luo *et al.*, 2017] Bingfeng Luo, Yansong Feng, Zheng Wang, Zhanxing Zhu, Songfang Huang, Rui Yan, and Dongyan Zhao. Learning with noise: Enhance distantly supervised relation extraction with dynamic transition matrix. In *Proceedings of ACL*, pages 430–439, 2017.

[Ma *et al.*, 2016] Yukun Ma, Erik Cambria, and Sa Gao. Label embedding for zero-shot fine-grained named entity typing. In *Proceedings of COLING*, pages 171–180, 2016.

[Mintz *et al.*, 2009] Mike Mintz, Steven Bills, Rion Snow, and Daniel Jurafsky. Distant supervision for relation extraction without labeled data. In *Proceedings of ACL*, pages 1003–1011, 2009.

[Murty *et al.*, 2018] Shikhar Murty, Patrick Verga, Luke Vilnis, Irena Radovanovic, and Andrew McCallum. Hierarchical losses and new resources for fine-grained entity typing and linking. In *Proceedings of ACL*, pages 97–109, 2018.

[Pennington *et al.*, 2014] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global vectors for word representation. In *Proceedings of EMNLP*, pages 1532–1543, 2014.

[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.

[Schütze *et al.*, 2017] Hinrich Schütze, Yadollah Yaghoobzadeh, and Heike Adel. Noise mitigation for neural entity typing and relation extraction. In *Proceedings of EACL*, pages 1183–1194, 2017.

[Shimaoka *et al.*, 2017] Sonse Shimaoka, Pontus Stenetorp, Kentaro Inui, and Sebastian Riedel. Neural architectures for fine-grained entity type classification. In *Proceedings of EACL*, pages 1271–1280, 2017.

[Srivastava *et al.*, 2014] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958, 2014.

[Wu *et al.*, 2019] Junshuang Wu, Richong Zhang, Yongyi Mao, Hongyu Guo, and Jinpeng Huai. Modeling noisy hierarchical types in fine-grained entity typing: A content-based weighting approach. In *Proceedings of IJCAI*, pages 5264–5270. ijcai.org, 2019.

[Xu and Barbosa, 2018] Peng Xu and Denilson Barbosa. Neural fine-grained entity type classification with hierarchy-aware loss. In *Proceedings of NAACL*, pages 1369–1378, 2018.

[Yavuz *et al.*, 2016] Semih Yavuz, Izzeddin Gur, Yu Su, Mudhakar Srivatsa, and Xifeng Yan. Improving semantic parsing via answer type inference. In *Proceedings of EMNLP*, pages 149–159, 2016.

[Yogatama *et al.*, 2015] Dani Yogatama, Daniel Gillick, and Nevena Lazic. Embedding methods for fine grained entity type classification. In *Proceedings of ACL*, pages 291–296, 2015.

[Yosef *et al.*, 2012] Mohamed Amir Yosef, Sandro Bauer, Johannes Hoffart, Marc Spaniol, and Gerhard Weikum. HYENA: hierarchical type classification for entity names. In *Proceedings of COLING*, pages 1361–1370, 2012.

[Zhang *et al.*, 2020] Haoyu Zhang, Dingkun Long, Guangwei Xu, Muhua Zhu, Pengjun Xie, Fei Huang, and Ji Wang. Learning with noise: Improving distantly-supervised fine-grained entity typing via automatic relabeling. In *Proceedings of IJCAI*, pages 3808–3815. ijcai.org, 2020.