Modeling Noisy Hierarchical Types in Fine-Grained Entity Typing: A Content-Based Weighting Approach

Junshuang Wu[1,2], **Richong Zhang**[1,2 *], **Yongyi Mao**[3], **Hongyu Guo**[4] and **Jinpeng Huai**[1,2]
[1]SKLSDE, School of Computer Science and Engineering, Beihang University, Beijing, China
[2]Beijing Advanced Institution on Big Data and Brain Computing, Beihang University, Beijing, China
[3]School of Electrical Engineering and Computer Science, University of Ottawa, Ottawa, Canada
[4]National Research Council Canada, Ottawa, Canada
{wujs,zhangrc,huaijp}@act.buaa.edu.cn, ymao@uottawa.ca, hongyu.guo@nrc-cnrc.gc.ca

Abstract

Fine-grained entity typing (FET), which annotates the entities in a sentence with a set of finely specified type labels, often serves as the first and critical step towards many natural language processing tasks. Despite great processes have been made, current FET methods have difficulty to cope with the noisy labels which naturally come with the data acquisition processes. Existing FET approaches either pre-process to clean the noise or simply focus on one of the noisy labels, sidestepping the fact that those noises are related and content dependent. In this paper, we directly model the structured, noisy labels with a novel content-sensitive weighting schema. Coupled with a newly devised cost function and a hierarchical type embedding strategy, our method leverages a random walk process to effectively weight out noisy labels during training. Experiments on several benchmark datasets validate the effectiveness of the proposed framework and establish it as a new state of the art strategy for noisy entity typing problem.

1 Introduction

Fine-grained entity typing (FET) [Ling and Weld, 2012; Yogatama *et al.*, 2015; Ren *et al.*, 2016a; McCallum *et al.*, 2018] refers to the task of annotating the text expressions (also known as mentions) of entities in a document or sentence, using a set of finely specified type labels. Knowledge acquired through FET is often utilized to facilitate many downstream natural language processing (NLP) tasks, such as knowledge base construction and expansion [Dong *et al.*, 2014], relation extraction [Koch *et al.*, 2014; Liu *et al.*, 2014], entity linking [Ling *et al.*, 2015] and factual question answer [Dong *et al.*, 2015], amongst many others.

Despite great processes have been made in the past years, the FET problem is however confronted by a major challenge issue: noisy labels. Such noisy labels naturally come with the data acquisition processes such as the popular distant supervision method [Mintz *et al.*, 2009]. Additionally, a mention, namely, a word or phrase, may assume different types.

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

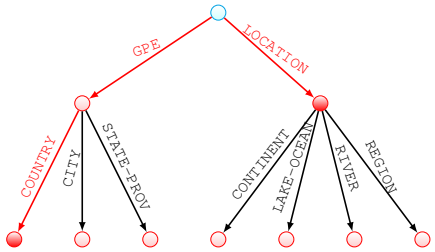

Figure 1: An example of a type hierarchy extracted from the BBN dataset, where two possible types GPE.COUNTRY and LOCATION for the word "Mexico" are shown as red paths and solid nodes.

For example, depending on its context, the word "Mexico" may refer to the location of Mexico in a general sense, or it may also refer to the country "United Mexican States" in the political sense (as shown in a small example of a type hierarchy depicted in Figure 1). Such noise can significantly degrade the predictive performance of the FET models [Ren *et al.*, 2016a]. Unfortunately, existing FET approaches either embrace a labor intensive pre-processing step to clean the noise [Gillick *et al.*, 2014] or simply bias the learning to only model one of the noisy labels [Ren *et al.*, 2016a; Abhishek *et al.*, 2017; Xu and Barbosa, 2018], sidestepping the fact that those noises are related and content dependent.

To directly model the structured noisy labels for FET, in this paper, we propose a novel content-sensitive weighting schema that is powered by a newly devised cost function and a hierarchical type embedding strategy. In detail, the hierarchically structured typing labels in the given data are first normalized to having only the leaf nodes of the tree as candidate types. Next, context dependent weights for all typing labels are then randomly initialized. Finally, through maximizing the expected reward of a random walk process, the weights of all candidate types are re-weighted such that only one of them will have large value, which corresponds to the correct type. In this way, the proposed weighting schema is able to effectively de-noise the noisy hierarchical typing labels during training, thus improving the predictive performance of the FET models.

Experiments on several widely used datasets show that our method significantly outperforms its competitors, resulting in new state-of-the-art performance on these benchmark data

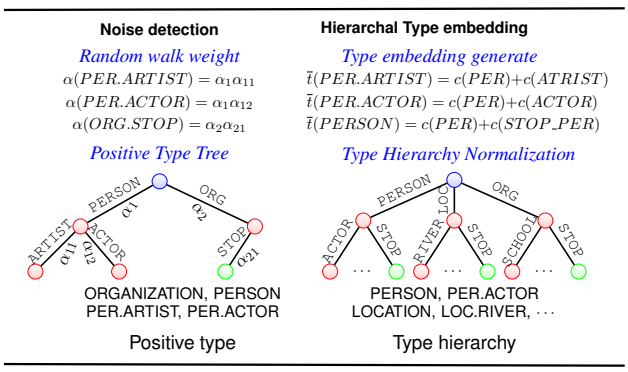

Figure 2: An example of our proposed fine-grained entity typing (FET) model Noise-Detective Prediction (NDP).

sets. We also provide an extensive ablation study to validate the effectiveness of different components of our proposed framework, highlighting the effectiveness of our newly devised cost function. To the best of our knowledge, this is the first work to jointly model the hierarchical structure of and the noise contained in the label type. We have posted the code on the GitHub (https://github.com/wujsAct/fine-grained-entity-typing-Hier-Denoise).

2 The Content-Based Weighting Model (NDP)

The overall architecture of our content-based weighting model is depicted in Figure 2. We denote our method as Noise Detective Prediction (NDP). In a nutshell, our NDP method first normalizes the hierarchically structured typing labels, so that only the leaf nodes represent valid type. Next, a loss constraint is used to regularize the training to weight out noisy labels.

2.1 Problem Statement

Let $\mathcal{T}$ denote the given type hierarchy of data set for the fine-grained entity typing (FET) task. Specifically, we interpret $\mathcal{T}$ as a tree, in which each edge is labeled by a *type token*, such as `LOCATION`, `CITY`, `REGION`, `RIVER` etc in Figure 1. Each node u in the tree $\mathcal{T}$ is a type, which can be uniquely specified by the path from the root node to the node u. For example, `LOCATION`, `LOCATION.RIVER` in Figure 1 are all types. We consider that $\mathcal{T}$ is given.

In general, an entity in the reality, such as in a KB (Knowledge base), may be categorized into multiple types in $\mathcal{T}$. For example, an entity that refers to a person that can be categorized both as type `PERSON.BUSINESSMAN` and as type `PERSON.POLITICIAN`. Thus the same entity in different context may assume different types.

A sentence s is regarded as a sequence $(w_1, w_2, \ldots, w_L)$ of *words*. In a sentence of interest, a subsequence of words $m := (w_J, w_{J+1}, w_{J+d-1})$ has been recognized as a *mention*, namely that it corresponds to some entity in a KB. Let $e(m)$ denote the entity that the mention m refers to in sentence s. The objective of a *(sentence-level) fine-grained entity typing* task is to find a type in $\mathcal{T}$, which is the *most appropriate* for the entity $e(m)$ in the context of the sentence s. We will refer to the most appropriate type for mention m in sentence s as the *correct type* for simplicity.

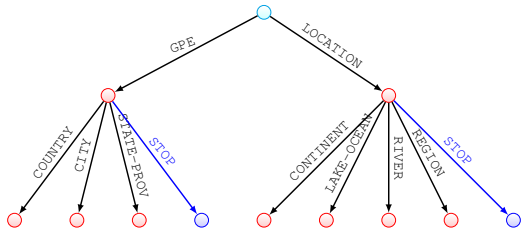

Figure 3: The normalized type hierarchy for the type tree in Figure 1. The added nodes and edges are shown in blue.

The dominant approach to solving this problem is via learning a predictive model which predicts the correct type for a given sentence s and a mention m therein. This relies on some training data $\mathcal{D}_{\text{train}}$, which typically takes the following form: $\mathcal{D}_{\text{train}} := \{(s_i, m_i, \tau_i) : i = 1, 2, \ldots, N\}$, where in each triple (s_i, m_i, τ_i), s_i is sentence, m_i is a mention in s_i, and τ_i is a subset of $\mathcal{T}$, namely a subset of types associated with the entity $e(m_i)$ corresponding to m_i. Ideally, we wish that τ_i contains only the correct type for m_i in s_i. In reality, however, it is difficult or costly to acquire the correct types for a large number of training examples and usually τ_i is obtained in a automatic manner such as via a weak multi-label classifier, e.g., distant supervision [Mintz *et al.*, 2009]. Inevitably, some τ_i's contain additional incorrect types, and we will call such sentence-mention pairs *noisy*. It has been observed that in standard datasets, rather significant fractions of mentions have noise type labels [Ren *et al.*, 2016a], and the statistics of such noise for the standard datasets can be found in Table 1.

2.2 Normalized Type Hierarchy

As discussed earlier, a type can in general correspond to any node in a type hierarchy $\mathcal{T}$. This creates difficulty for a model to directly predict the mention of a type at the right tree depth. To resolve this problem cleanly, we propose converting the type hierarchy to a different representation. Given the type hierarchy $\mathcal{T}$, we transform it to a *normalized type hierarchy* $\mathcal{T}^*$ as follows: For each intermediate (namely, non-leaf and non-root) node u in $\mathcal{T}$, add a child node v to u, and label the edge connecting u to v by a new type token, which we denote by `STOP`. This gives rise to the normalized type hierarchy $\mathcal{T}^*$. Figure 3 shows the transformation of a type hierarchy to its normalized version.

There is a natural map φ that maps all nodes in the original tree $\mathcal{T}$ to the leaf nodes in tree $\mathcal{T}^*$. Specifically, if u is the root node or any leaf node in $\mathcal{T}$, $\varphi(u)$ is the same node in $\mathcal{T}^*$; and if u is an intermediate node in $\mathcal{T}$, $\varphi(u)$ is the the newly added leaf node of u in $\mathcal{T}^*$.

Under this mapping, every type in $\mathcal{T}$ can be identified with a leaf node in $\mathcal{T}^*$, or equivalently by a path in $\mathcal{T}^*$ connecting the root node to that leaf node. In a normalized type hierarchy, only the leaf nodes represent valid types. Using such a normalized representation, the learned model will only consider the leaf nodes of the tree as candidate types. If the model predicts the type as a leaf node v that is pointed to by a

STOP edge, for example, the node LOCATION.STOP, then the parent node of v, namely, LOCATION, is declared as the predicted type. Thus using such a normalized tree allows the model to predict types at any level of the original type hierarchy, without recourse to additional heuristics. From here on, we will work with the $\mathcal{T}^*$ instead of $\mathcal{T}$, and every reference to a type will refer to its image in $\mathcal{T}^*$ under φ.

2.3 Weight Out Noise with Loss Function

For notation convenience, we will denote the set of all sentence-mention pairs in $\mathcal{D}_{\text{train}}$ by $\mathcal{D}^{\text{in}}_{\text{train}}$, that is, $\{(s_i, m_i, \tau_i) : i = 1, 2, \cdots, N\} := \mathcal{D}^{\text{in}}_{\text{train}}$.

For each training example (s_i, m_i, τ_i), we will also generate a *negative training example*, (s_i, m_i, τ_i^-), where τ_i^- is a random set of types in $\mathcal{T}^*$ but not in τ_i. We will denote by $\mathcal{D}^-_{\text{train}}$ the set of all negative training examples $\{(s_i, m_i, \tau_i^-) : (s_i, m_i) \in \mathcal{D}^{\text{in}}_{\text{train}}\}$. Our NDP model will be trained on both $\mathcal{D}^{\text{in}}_{\text{train}}$ and $\mathcal{D}^-_{\text{train}}$.

For any given sentence-mention pair (s, m), let $\overline{x}(s, m) \in \mathbb{R}^{k_{\text{F}}}$ denote the embedding of (s, m). For each type $t \in \mathcal{T}$, let $\bar{t} \in \mathbb{R}^{k_{\text{T}}}$ denote the embedding of t. We will postpone the construction of embeddings $\overline{x}(s, m)$ and $\bar{t}$ in later sections.

Our loss function relies on the following hypotheses.

Hypothesis 1 *For every sentence s of interest and the identified mention m in s, there is exactly one type for m in the context s.*

Hypothesis 2 *For each training example $(s_i, m_i, \tau_i) \in \mathcal{D}_{\text{train}}$, τ_i contains the correct type of m_i in s_i.*

Hypothesis 1 above may appear overly stringent at the first glance. We wish to note however that this is in fact consistent with our problem setting. Recall that the problem is to predict *the most appropriate* type for the mention of interest. Even if from the context two or more types may be appropriate, it is reasonable to consider that there is only one that is *the most* appropriate. On the other hand, it is remarkable that an important purpose of setting up hypotheses in machine learning is to constrain the space of learning so as to reduce the model capacity, enhance learn-ability and avoid over-fitting. Such practice usually becomes particularly important when the training data is limited. Theoretically, the hypotheses one sets up need not to be perfectly "correct", it is only required that the hypotheses *do not conflict* with the ground truth, exhibited partially from the observed data. Such a conflict, if existing, would result in under-fitting and degrade the model performance. But if such a conflict does not exist, even over-simplified hypotheses provide benefits to learning. This has been the reason underlying numerous models in which the very unrealistic Gaussian hypotheses are made.

Given that the training set $\mathcal{D}^{\text{in}}_{\text{train}}$ is usually noisy, it is critical that the FET model is capable of coping with such noise.

For an arbitrary $\overline{x} \in \mathbb{R}^{k_{\text{F}}}$ and a $\bar{t} \in \mathbb{R}^{k_{\text{T}}}$, let there be a *score* function $\phi(\overline{x}, \bar{t})$. Although one may explore a wide variety of score functions, in this paper, we specialize the score function by

$$\phi(\overline{x}, \bar{t}) = \overline{x}^T A \bar{t} \tag{1}$$

where A is the trainable matrix of size $k_{\text{F}} \times k_{\text{T}}$.

For each input (s_i, m_i) in $\mathcal{D}^{\text{in}}_{\text{train}}$, let its *positive-example score* be defined as

$$S(s_i, m_i) := \sum_{t \in \tau_i} \alpha(t; s_i, m_i) \phi(\overline{x}(s_i, m_i), \bar{t}) \tag{2}$$

where $\alpha(t; s_i, m_i)$'s are a set of non-negative weights summing to 1, namely, satisfying

$$\sum_{t \in \tau_i} \alpha(t; s_i, m_i) = 1.$$

The construction of these weights is given in a later section. The idea here is that after these weights are learned, only one of them will have large value (near 1), which corresponds to the correct type assumed in Hypotheses 1 and 2.

For each input (s_i, m_i) in $\mathcal{D}^{\text{in}}_{\text{train}}$, let its *negative-example score* with respect to a negative type $t^- \in \tau_i^-$ be defined as

$$S^-(s_i, m_i, t^-) := \phi(\overline{x}(s_i, m_i), \overline{t^-}) \tag{3}$$

We then define a *margin-based loss* $E(s_i, m_i)$ by

$$E(s_i, m_i) := \sum_{t^- \in \tau_i^-} \max\big(0, \lambda - S(s_i, m_i) + S^-(s_i, m_i, t^-)\big) \tag{4}$$

where λ is a positive hyper-parameter. Note that under this equation, when $E(s_i, m_i)$ takes the minimum value 0, for any wrong type $t^- \in \tau_i^-$, the score $S^-(s_i, m_i, t^-)$ is lower than the positive-example score $S(s_i, m_i)$ by at least λ. Thus when we minimize $E(s_i, m_i)$ for input (s_i, m_i), we implicitly maximize $S(s_i, m_i)$ and force score $S^-(s_i, m_i, t^-)$ to be sufficiently low.

The overall loss function of the model is then given by

$$\mathcal{L} := \sum_{(s_i, m_i) \in \mathcal{D}^{\text{in}}_{\text{train}}} E(s_i, m_i). \tag{5}$$

Generation of Weights

Suppose that a training input (s_i, m_i) is given. We now explain how the weights $\{\alpha(t; s_i, m_i) : t \in \tau_i\}$ are generated. Briefly these weights are essentially the probabilities that a random walk on $\mathcal{T}^*$ lands on each leaf node of $\mathcal{T}^*$.

Specifically, let Q be the random walk on the normalized type hierarchy $\mathcal{T}^*$, which we now define. The walk starts from the root of $\mathcal{T}^*$ and follows the tree structure to eventually arrives at some leaf node. For each node $u \in \mathcal{T}^*$, let $C(u)$ be the set of all children of u in $\mathcal{T}^*$. For each node u in $\mathcal{T}^*$, let $q(\cdot|u)$ be a distribution on $C(u)$, where $q(v|u)$ for a child v of u is the probability that the walk goes to v given its current location is u. Given the set of distributions $\{q(\cdot|u) : u \in \mathcal{T}^*\}$, one can express the probability $p(z)$ that the walk eventually arrives at any given leaf node z. More precisely, suppose that the path from the root node to z is the node sequence $(u_0, u_1, \ldots, u_m)$, where u_0 is the root node and u_m is z. Then

$$p(z) = \prod_{j=1}^{m} q(u_j | u_{j-1})$$

For every training input (s_i, m_i), we construct such a process $Q(s_i, m_i)$, namely, a set of conditional distributions $\{q(\cdot|u) : u \in \mathcal{T}^*\}$). Then we define for every type $t \in \tau_i$,

$$\alpha(t; s_i, m_i) := p(t) \tag{6}$$

under the random walk $Q(s_i, m_i)$. The set of all such random walks $\{Q(s_i, m_i) : (s_i, m_i) \in \mathcal{D}^{\text{in}}_{\text{train}}\}$ will be learned during training.

We also propose two testing weight methods: average weight (AW) and global data-dependent weight (DDW). For average weight, we treat all the types equally. For global DDW, we directly utilize the normalized type score as the type weight. The the type weight are computed similar to Equation 1 as follows:

$$\alpha(t; s_i, m_i) \propto \exp(\overline{s_i}^T B \bar{t}) \tag{7}$$

where B is the trainable matrix of size $k_{\text{F}} \times k_{\text{T}}$.

Interpretation of Loss Function

We now pause and revisit the loss function (5) to give some insight on the design of this loss function. Given a training input (s_i, m_i), one can imagine that a random walker walks down the type hierarchy $\mathcal{T}^*$, according to an unknown process $Q(s_i, m_i)$. According to process, he may arrive at any leaf node in τ_i with some probability. At leaf node $t \in \tau_i$, he would collect a "reward" whose value equals the score $\phi(\overline{x}(s_i, m_i), \bar{t})$. Then $S(s_i, m_i)$ defined in (2) may then be regarded as the expected reward the walker will get under $Q(s_i, m_i)$. Thus, maximizing $S(s_i, m_i)$ over its set of weights in (2) is equivalent to maximizing the walker's expected reward over the random walk process $Q(s_i, m_i)$.

It is easy to see that under such a random walk process, the probabilities that the walker arrives at two sibling leaf nodes in the tree $\mathcal{T}^*$ are related by the probability the walker goes through their common parent. Thus, the random walk process naturally looks after the correlation between the siblings.

We also like to stress that such a random walk is not global to all training examples. Rather, for each training example, there is a separate such process, and its parameter $q(\cdot|\cdot)$ is learned during training. Since incorrect type labels tend to give high loss value, for each training example, the optimization process will result in a function q which puts a low probability on the wrong types and a high probability to the correct type. That is, minimizing the loss function *squeezes out* noise and enhances signal.

2.4 Type and Feature Embedding

For type embedding, we construct a embedding matrix D_{T} for the set of all type tokens in $\mathcal{T}^*$, from each token is assigned a vector in $\mathbb{R}^{k_{\text{T}}}$ as its embedding vector. We note that all "STOP" tokens in the $\mathcal{T}^*$ are consider different and hence they all have different embedding vectors. For every type $t \in \mathcal{T}^*$, if the path from the root to node t is the sequence $(c_1, c_2, \ldots, c_l)$, the embedding $\bar{t}$ of type t is defined as

$$\bar{t} := \sum_{j=1}^{l} \overline{c_j}, \tag{8}$$

where $\overline{c_j}$ is the embedding vector for token c_j.

Feature embedding in our model largely follows the previous approaches in [Abhishek *et al.*, 2017] and [Xin *et al.*, 2018]. Specifically, for any sentence-mention pair (s, m), its feature embedding $\overline{x}(s, m)$ is the concatenation of vectors $\overline{s} \in \mathbb{R}^{k_{\text{s}}}$ and vector $\overline{m} \in \mathbb{R}^{k_{\text{m}}}$, which we now define.

First note that we adopt a pre-trained word embedding dictionary D_{W} from Glove [Pennington *et al.*, 2014]. Using this embedding dictionary D_{W}, the vector $\overline{m}$ is simply the average of the embedding vectors of all surface words in m.

To obtain vector $\overline{s}$, first note that the mention m partitions s into two word subsequences, the sequences $(w^{\text{L}}_1, w^{\text{L}}_2, \ldots, w^{\text{L}}_a)$ left to the mention and the sequences $(w^{\text{L}}_1, w^{\text{L}}_2, \ldots, w^{\text{L}}_a)$ right to the mention. The left and right sequences are each passed to Bidirectional LSTM network, and correspondingly generate output sequences $(h^{\text{L}}_1, h^{\text{L}}_2, \ldots, h^{\text{L}}_a)$ and $(h^{\text{L}}_1, h^{\text{L}}_2, \ldots, h^{\text{L}}_a)$ respectively. The outputs of the two networks are respectively attentively combined and then summed up to give rise the embedding $\overline{s}$. More precisely,

$$\overline{s} := \sum_{i=1}^{a} \beta^{\text{L}}_i h^{\text{L}}_i + \sum_{i=1}^{b} \beta^{\text{R}}_i h^{\text{R}}_i$$

Here β^{L}_i's and β^{R}_i's are attention weights, computed according to

$$\begin{aligned} \beta^{\text{L}}_i &\propto \exp(\theta \tanh(U h^{\text{L}}_i + V \overline{m})) \\ \beta^{\text{R}}_i &\propto \exp(\theta \tanh(U h^{\text{R}}_i + V \overline{m})) \end{aligned}$$

where U and W are trainable parameter matrices, serving to map a vector into $\mathbb{R}^{k_{\text{A}}}$, and θ is a vector of dimension k_{A}.

At this point, we have completely defined the proposed NDP model. When using the trained model for predicting the type of a mention m in a sentence s, we simply compute the score $\phi(\overline{x}(s, m), t)$ for every $t \in \mathcal{T}^*$ under the model. The type t with the highest score is then declared as the correct type. Note that if the declared type is one pointed to by the STOP edge, we regard the declared type as its parent type.

3 Experimental Studies

3.1 Settings

We evaluate the NDP approach using three popular datasets, Wiki [1] [Ling and Weld, 2012], OntoNotes [Weischedel *et al.*, 2011] and BBN [Weischedel and Brunstein, 2015]. We utilize the pre-processed data provided by [Ren *et al.*, 2016a], where 10% of each testing data is used as a validation set, following the standard protocol widely adopted in the previous works [Abhishek *et al.*, 2017; Shimaoka *et al.*, 2017; Ren *et al.*, 2016b; Ren *et al.*, 2016a]. Table 1 shows the statistics of the three datasets.

We note that the fraction of training sentence-mention pairs ("mentions" in Table 1) that are noisy (namely, containing more than one type) is 35.42% in Wiki, 27.4% in OntoNotes, and 27.4% in BBN. This suggests that the training set contains significant amount of noise and that learning approaches robust to such noise are highly desirable. On the other hand, noise in the testing data is much less severe. In fact, the testing set of BBN contains no noisy mention, and the testing

[1] In the Wiki dataset, there exist types that have no parent. The original structure of types in Wiki thus does not qualify as a tree. Ren et al. [Ren *et al.*, 2016a] fixed this problem by adding 15 types. As we do not consider those added types during testing, adding those types have no effect on the test performance

Data sets	Wiki	OntoNotes	BBN
#Orginial types	113	89	47
#Type hierarchy level	2	3	2
#Type tokens	177	141	63
#STOP tokens	49	52	16
#Types	128	89	47
#Training mentions	2.69M	220,398	86,078
#Training sentences	1.51M	88,284	32,739
#Test mentions	563	9,603	13,282
#Test sentences	434	1,312	6,431
%Noisy training mentions	**35.42**	**27.4**	**24.08**
%Noisy test mentions	**11.72**	**6.23**	**0**

Table 1: Statistics of datasets

Hyper-parameter	Wiki	OntoNotes	BBN
Learning rate	0.001	0.001	0.001
Batch size	2000	1000	1000
Loss margin	0.2	0.4	0.3
#Negative types	100	70	30
LSTM hidden-size	300	200	100
k_{s}	600	400	200
k_{m}	300	300	300
k_{F}	900	700	500
k_{T}	200	100	50
k_{A}	50	40	40
L2 loss weight	0.001	0.001	0.001

Table 2: Hyper-parameters for three datasets

data of OntoNotes is also quite clean (with 6.23% noisy mentions). The fraction of noisy mentions in the testing data of Wiki is 11.72%, higher than the other two datasets, although mild relative to its training data.

Similar to previous work [Abhishek *et al.*, 2017], in our implementation of the proposed NDP model, we utilize the word embedding provided by Glove [Pennington *et al.*, 2014]. And, We do not fine tune them during training. The Adam optimizer [Kingma and Ba, 2014] is utilized to train the model. An early-stop strategy is employed to terminate training when the loss on validation data does not decrease. We incorporate dropout and L2 regularization to avoid over-fitting. The left and right sequence lengths are truncated to 10. We utilize random search to explore the proper hyper-parameters. The chosen hyper-parameters are listed in Table 2.

We compare NDP with several state-of-the-art neural network models: FIGER [Ling and Weld, 2012], HYENA [Yosef *et al.*, 2012], WSABIE [Yogatama *et al.*, 2015], ProtoLE and Proto-HLE [Ma *et al.*, 2016], AFET [Ren *et al.*, 2016a], ATTN [Shimaoka *et al.*, 2017], ABH [Abhishek *et al.*, 2017], and NFETC [Xu and Barbosa, 2018]. For comparison, we use the reported performances of these models in their respective original papers, except for FIGER and ATTN, the performances of which are taken from [Ma *et al.*, 2016] and [Abhishek *et al.*, 2017]. The evaluation metrics used are Accuracy (Acc), Micro-averaged F1 (Mi-F1) score and Macro-F1 (Ma-F1) score. These metrics are the most commonly used for evaluating the FET models [Ling and Weld, 2012; Ren *et al.*, 2016a].

3.2 Predictive Performance

The comparison results are presented in Table 3, where best performance is highlighted in **bold**. The performances of the compared models are all taken from the literature. We also run the published code of ATTN, ABH, ABH-AllC and NFETC and the re-produced results are labelled by $*$ symbol. Note that original NFETC paper using a prepossessed dataset. So their original results are not included for comparison. We re-implement the NFETC model by using the original type hierarchy from Wiki dataset.

It can be seen that NDP outperforms all compared models resulting in a state-of-the-art result under all metrics. This confirms the advantage of the proposed NDP model. In particular, on BBN, NDP outperforms other testing methods with a large margin. We believe that this is the most accurate and truthful demonstration of the advantage of NDP. As mentioned earlier, the testing set of BBN has signficantly less noise than the other datasets. As such, evaluation on this dataset is the closest to evaluation against the ground truth. Thus relative to other used datasets, the evaluation results obtained on BBN are the most reliable.

It is worth noting that NDP exhibits superior performance consistently on all three datasets, whereas other models tend to achieve good performance, if at all, only on one or two datasets. In this sense, NDP is more robust against the varying intrinsic structures of the datasets.

3.3 Ablation Studies

Performances at Different Levels of the Type Hierarcy
We further investigate the performances of the compared models at each level of the type hierarchy. We use the BBN dataset for this purpose and examine the level-1 and level-2 performances of these models. Specifically, for level-2 performances, the testing examples without having type at level 2 are disregarded from evaluation. For level-1 performances, the testing examples which only have type at level 1 are considered. The results of this evaluation are given in Table 5. At both levels NDP outperforms significantly the other models in F1.

Benefit of the Hierarchical Type Normalization
We also conduct experiments to evaluate the impact of the hierarchical normalization type embedding on the NDP framework. We simple replace our hierarchical type embedding with the Complex Bilinear hierarchical structure model (denoted ComplEx) recently introduced in [McCallum *et al.*, 2018]. Results are presented in Table 4. Table 4 indicates that the normalization component we adopted significantly benefit the predictive performance of our NDP model (the last three rows of Table 4), compared to that of the ComplEx strategy (the first three rows of Table 4).

Impact of the Random Walk Weights
We further exams the effects of our random walk strategy on our framework. We compare with average weighting (AW) and a learnable data-dependent weighting schema (DDW). The comparison results are depicted in Table 4. From Table 4, we can see that the random walk method brings significant gain, in terms of predictive performance, over the other two testing weighting schemes.

Model	Wiki			OntoNotes			BBN		
	Acc	Ma-F1	Mi-F1	Acc	Ma-F1	Mi-F1	Acc	Ma-F1	Mi-F1
FIGER	0.474	0.692	0.655	0.369	0.578	0.516	0.467	0.672	0.612
HYENA	0.288	0.528	0.506	0.249	0.497	0.446	0.523	0.576	0.587
WSABIE	0.480	0.679	0.657	0.404	0.580	0.527	0.619	0.670	0.680
ProtoLE	0.535	0.681	0.665	0.469	0.659	0.591	0.704	0.758	0.765
Proto-HLE	0.505	0.666	0.653	0.493	0.682	0.613	0.695	0.745	0.74.5
AFET	0.533	0.693	0.664	0.551	0.711	0.647	0.670	0.727	0.735
ATTN	0.581	0.780	0.744	0.473	0.655	0.586	0.484	0.732	0.724
ATTN*	0.57	0.775	0.736	0.497	0.636	0.558	0.512	0.700	0.728
ABH-AllC	0.662	0.805	0.77	0.514	0.672	0.626	0.655	0.736	0.752
ABH-AllC*	0.650	0.791	0.766	0.530	0.673	0.611	0.650	0.752	0.767
ABH	0.658	0.812	0.774	0.522	0.685	0.633	0.604	0.741	0.757
ABH*	0.627	0.766	0.743	0.537	0.675	0.621	0.667	0.761	0.772
NFETC*	0.645	0.794	0.751	0.556	0.703	0.641	0.680	0.719	0.734
NDP	**0.677**	**0.818**	**0.780**	**0.580**	**0.712**	**0.648**	**0.727**	**0.764**	**0.777**

Table 3: Performance of FET models on Wiki, OntoNotes and BBN datasets. Reproducing results for these models are labelled with $*$.

Model	Wiki			OntoNotes			BBN		
	Acc	Ma-F1	Mi-F1	Acc	Ma-F1	Mi-F1	Acc	Ma-F1	Mi-F1
NDP (ComplEx AW)	0.602	0.724	0.687	0.565	0.689	0.623	0.706	0.737	0.748
NDP (ComplEx DDW)	0.609	0.734	0.685	0.561	0.689	0.613	0.707	0.745	0.755
NDP (ComplEx RWW)	0.627	0.748	0.706	0.571	0.696	0.634	0.712	0.748	0.756
NDP (AW)	0.632	0.766	0.733	0.585	0.720	0.656	0.710	0.751	0.758
NDP (DDW)	0.652	0.785	0.747	0.573	0.703	0.637	0.713	0.752	0.765
NDP (RWW)	0.677	0.818	0.780	0.580	0.712	0.648	0.727	0.764	0.777

Table 4: Performance of variant NDP models. AW: average weight; DDW: data-dependent weight; RWW: random walk weight

Model	Level 1			Level 2		
	P	R	F1	P	R	F1
ATTN	0.379	0.675	0.485	0.808	0.636	0.712
ABH-AllC	0.433	0.748	0.549	0.886	0.608	0.721
ABH	0.466	0.752	0.575	0.858	0.631	0.727
NFETC	0.540	0.681	0.603	0.775	0.688	0.729
NDP	0.580	0.731	0.646	0.817	0.725	0.769

Table 5: Performance of different level types on BBN test data.

4 Related Work

Type Hierarchy Modeling The most earlier models ignore the type hierarchy and simply treat it as a flat structure (see, e.g., [Ling and Weld, 2012; Yogatama *et al.*, 2015; Shimaoka *et al.*, 2017; Abhishek *et al.*, 2017]). Some works, e.g., [Yosef *et al.*, 2012; Ren *et al.*, 2016a; Xu and Barbosa, 2018; McCallum *et al.*, 2018] have made efforts explicitly modeling the correlation among hierarchical types. These models, although demonstrating some successes, rely on some heuristic procedure to walk down the type hierarchy, stop at a node, and then declare the node as the predicted type. The recent work [McCallum *et al.*, 2018] proposed the structure loss to solve this problem. Unlike these works, we propose a generic framework that directly model the hierarchical typing without additional structure loss function to train or heuristic method to infer through the type tree.

Label Noise Handling Various approaches have been proposed to deal with *noisy* entity mentions in the training set. Gillick et al. [Gillick *et al.*, 2014] refined the training data by applying a set of heuristics to prune types. [Yogatama *et al.*, 2015] proposed a man-made function to transform the positive type rank to a weight. [Ren *et al.*, 2016a; Abhishek *et al.*, 2017; Xu and Barbosa, 2018] model *noisy* and *clean* entity mentions separately. Nevertheless, these methods enforce the model to choose the most relevant type as the golden positive type label for *noisy* entity mention during training. Instead, we explicitly model all the noisy labels and their structures during training.

5 Conclusion and Future Work

We introduce a novel approach for Fine-grained Entity Typing modeling. Our method leverages a novel cost function to jointly model the correlation among hierarchical types and label noises. Our experimental results demonstrate the power of this approach convincingly. We prefer call this approach a "framework" rather than a single model, since under this approach, one can freely explore other constructions of feature embedding module and other choices of score function ϕ. More research is underway to explore this rich space of strategies.

Acknowledgments

This work is supported partly by China 973 program (No. 2015CB358700), by the National Natural Science Foundation of China (No. 61772059, 61421003), and by the Beijing Advanced Innovation Center for Big Data and Brain Computing (BDBC) and State Key Laboratory of Software Development Environment (No. SKLSDE-2018ZX-17).

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.

[Dong *et al.*, 2014] Xin Dong, Evgeniy Gabrilovich, Geremy Heitz, Wilko Horn, Ni Lao, Kevin Murphy, Thomas Strohmann, Shaohua Sun, and Wei Zhang. Knowledge vault: a web-scale approach to probabilistic knowledge fusion. In *KDD*, pages 601–610, 2014.

[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. *arXiv preprint arXiv:1412.1820*, 2014.

[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.

[Liu *et al.*, 2014] Yang Liu, Kang Liu, Liheng Xu, and Jun Zhao. Exploring fine-grained entity type constraints for distantly supervised relation extraction. In *Proceedings of COLING*, pages 2107–2116, 2014.

[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.

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

[Mintz *et al.*, 2009] Mike Mintz, Steven Bills, Rion Snow, and D aniel Jurafsky. Distant supervision for relation extraction without labeled data. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP*, pages 1003–1011, 2009.

[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.*, 2016a] 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.

[Ren *et al.*, 2016b] Xiang Ren, Wenqi He, Meng Qu, Clare R. Voss, Heng Ji, and Jiawei Han. Label noise reduction in entity typing by heterogeneous partial-label embedding. In *Proceedings of KDD*, pages 1825–1834, 2016.

[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.

[Weischedel and Brunstein, 2015] Ralph Weischedel and Ada Brunstein. Bbn pronoun coreference and entity type corpus. page 112, 2015.

[Weischedel *et al.*, 2011] Ralph Weischedel, Eduard Hovy, Mitchell Marcus, Martha Palmer, Robert Belvin, Sameer Pradhan, Lance Ramshaw, and Nianwen Xue. Ontonotes: A large training corpus for enhanced processing. 2011.

[Xin *et al.*, 2018] Ji Xin, Yankai Lin, Zhiyuan Liu, and Maosong Sun. Improving neural fine-grained entity typing with knowledge attention. In *Proceedings of AAAI*, 2018.

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

[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. ACL, 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 2012: Posters*, pages 1361–1370, 2012.