OTyper : A Neural Architecture for Open Named Entity Typing

Zheng Yuan and **Doug Downey**
Department of Electrical Engineering & Computer Science
Northwestern University, Evanston IL 60208, USA
zys133@eecs.northwestern.edu, d-downey@northwestern.edu

Abstract

Named Entity Typing (NET) is valuable for many natural language processing tasks, such as relation extraction, question answering, knowledge base population, and co-reference resolution. Classical NET targeted a few coarse-grained types, but the task has expanded to sets of hundreds of types in recent years. Existing work in NET assumes that the target types are specified in advance, and that hand-labeled examples of each type are available. In this work, we introduce the task of *Open* Named Entity Typing (ONET), which is NET when the set of target types is not known in advance. We propose a neural network architecture for ONET, called OTyper , and evaluate its ability to tag entities with types not seen in training. On the benchmark FIGER(GOLD) dataset, OTyper achieves a weighted AUC-ROC score of 0.870 on unseen types, substantially outperforming pattern- and embedding-based baselines.

1 Introduction

Named Entity Typing (NET) is the task of labeling a given entity mention in text with a semantic label. NET is valuable for many natural language processing tasks, such as relation extraction, question answering, knowledge base population, and co-reference resolution.

Traditional NET focused on a small set of mutually-exclusive types, such as person, location, and organization (Tjong Kim Sang and De Meulder 2003; Carreras, Marquez, and Padró 2002; Abdul-Hamid and Darwish 2010; Chinchor and Robinson 1997). More recent work has generalized NET to much larger type systems that include fine-grained types, e.g. book, artist, city, and so on (Rabinovich and Klein 2017; Shimaoka et al. 2016b; Ling and Weld 2012; Yogatama, Gillick, and Lazic 2015). In fine-grained NET, often the target types are non-disjoint ("Paris" is both a city and a location, for example) and thus fine-grained NET is a multi-label task, i.e. each mention may be assigned multiple positive labels.

Existing fine-grained NET techniques have a limitation: they require labeled training mentions for each target type. For types that are not in the training set—referred to as *unseen* types—NET systems cannot output the labels. This is a limitation because in many cases, we want to know whether a given entity belongs to a specific type that is *not* present in our limited training dataset. Learning whether an entity belongs to an unseen type is an instance of "zero-shot learning," where no training examples exist for a given output label (Palatucci et al. 2009; Socher et al. 2013; Chang et al. 2008). While existing hypernym discovery techniques can identify whether a given phrase belongs to a type, hypernym discovery is context insensitive, whereas NET is context sensitive. For example, given the sentence "I went to Chicago last year," hypernym discovery would output all possible types of "Chicago" (city, band, movie, etc.). Whereas in NET we must only assign types that are correct for "Chicago" in the given context, e.g. city and location. Unsupervised Named Entity Recognition (Huang et al. 2017) is context-sensitive and in principle can handle entities of unseen types, but these techniques require mutually exclusive entity categories—i.e. they cannot apply to the multi-label setting encountered in fine-grained NET.

In this paper, we introduce the task of *Open* Named Entity Typing (ONET), which is fine-grained NET when the set of target types is not known in advance. We propose a novel neural architecture for ONET. The intuition behind our approach is simple: we represent mentions and types in our model using word embeddings pre-trained on unlabeled text, and leverage regularities in the embedding space to extend to unseen types. Specifically, our model projects the embeddings of mentions and types into a common space. We train the model such that in the common space, the representations of correct types are close to the mention representation, while the representations for incorrect types are far away. Our hypothesis is that regularities in the pre-trained embedding space will allow this approach to extend to unseen types. That is, if our model learns to map mentions of the type "musician" to be close to the representation for the word "musician," it will also map mentions of the type "drummer" to be close to the representation of the word "drummer," even if the latter type never appears in training. Our architecture extends previous work (Shimaoka et al. 2016b) by adding a new component, type embeddings, and a method for comparing the type embedding to a given mention embedding. In addition, our model is able to incorporate mention- and pattern-based features as optional components.

We refer to our model as OTyper[1]. We evaluate OTyper on a common benchmark NET dataset, FIGER(GOLD) (Ling and Weld 2012) and MSH-WSD dataset(Jimeno-Yepes, McInnes, and Aronson 2011). The results show that OTyper outperforms two baseline models and achieves weighted average ROC_AUC score of 0.870 and 0.780 on unseen types for FIGER(GOLD) and MSH-WSD. We also analyze which factors drive the system's performance, finding that the presence of training types that are more similar to an unseen target type improves accuracy on that type.

In summary, our contributions include:

- We introduce a new task of Open Named Entity Typing (ONET), which is fine-grained NET when the set of target types is not known in advance.
- We propose a neural network model, OTyper, for the ONET task and experimentally demonstrate that the technique outperforms baseline approaches on unseen types.

The rest of this paper is organized as follows. In section 2, we cover previous work in NET. We present our OTyper model in Section 3 and evaluate it in section 4. Section 5 concludes.

2 Related Work

Named Entity Typing is a long-standing task in Natural Language Processing (Grishman and Sundheim 1996; Collins and Singer 1999; Elsner, Charniak, and Johnson 2009; Ratinov and Roth 2009; Ritter et al. 2011; Kuru, Can, and Yuret 2016). Most work in NET is performed in the context of the *Named Entity Recognition* (NER) task, which includes NET as a subtask. In NER, systems must first find and delimit entity mentions, and then perform NET, i.e. assign each mention to a type. Using a variety of supervised approaches, existing NET methods can achieve high accuracy and recall. Most of these systems only deal with traditional NET over three categories: person, location, organization.

Many end tasks such as relation extraction and question answering can benefit from finer-grained entity typing, which has become a focus in recent years. Unlike traditional NET, fine-grained NET considers hundreds or thousands of types, and each entity mention can be assigned to more than one type. Ling and Weld (2012) introduced 113 entity types derived from Freebase and released a dataset, FIGER, now a commonly-used benchmark for fine-grained NET. They used semantic features to train a multi-instance multi-label distant supervision classifier on FIGER. In this paper, we use FIGER as our evaluation dataset. Lee et al. (2006) defined 147 fine-grained named entity types and used a CRF on fine grained NET for question answering. Yogatama et al. (2015) introduced embedding methods that use a ranking loss and learn a joint representation of features and labels, which allows for information sharing among related labels. Rabinovich et al. (2017) considered a large number of types and applied a linear model for fine-grained NET. Shimaoka et al. (2016a) proposed the first model for fine-grained entity typing that learns to recursively compose representations for the context of each entity using an attention model. Shimaoka et al. (2016b) combined the attention model with mention features and hierarchical label-sharing parameters in a NET system, and achieved the current state-of-the-art result on the FIGER(GOLD) dataset. By adopting a universal schema approach, Yao et al. (2013) operates over the union of all types from its input sources, and is able to classify over 16,000 types. However, all these methods assume a pre-defined a set of fine-grained types that is known at training time—they are not applicable to ONET, where the target types do not appear in the training dataset.

Huang et al. (2017; 2016) proposed an unsupervised entity-typing framework by combining symbolic and distributional semantics. They use domain knowledge bases as an additional data resource. Their approach creates a knowledge graph and knowledge representation based on domain knowledge base, and then clusters entity embeddings and named entities. Their system produces only one type label for each mention, while in ONET each mention can correspond to many types. Also, domain knowledge bases are not always available, which is a limitation for their system. By contrast our OTyper approach does not use a knowledge base, only pre-trained embeddings.

Similar to named entity typing, hypernym discovery (Snow, Jurafsky, and Ng 2005; Seitner et al. 2016) also labels entities with their hypernyms. Hypernym discovery is the task of, given an NP e, finding a set of NPs c_i such that each c_i is a hypernym of e (Ritter, Soderland, and Etzioni 2009). Hypernym discovery is often powered by Hearst patterns (Hearst 1992), which we also employ as features in OTyper . The primary difference between our ONET task and hypernym discovery is that ONET is context sensitive—we must not only assign types to each entity name, but also determine which of those types are relevant to the particular sense of the entity name used in a given context.

3 OTyper Architecture

We now formally define Open Named Entity Typing (ONET), and present our OTyper architecture for the task.

3.1 Problem Definition

As in traditional NET, in ONET we take as input a set of *mentions*. Each mention is an occurrence of a named entity in text, along with its surrounding context. We are also given a set of target types, and the the task is to associate each mention with its correct types. An ONET system is trained on a labeled dataset of annotated mentions. The key difference between ONET and NET is that in ONET, some of the target types (i.e. the test types) may not occur at all in the training phrase. We use the symbol t_i to denote a type in training or testing, and m_i to denote a mention. Each mention m_i contains two parts: the entity e_i and its textual context on the left and right, referred to as cl_i and cr_i. In our remaining notation, we use lowercase letters for scalars, bold lowercase for vectors and uppercase letters for matrices and constants. We define *seen types* to be the types in the training dataset. *Unseen types* are those that do not exist in training dataset but are found in test dataset. While in principle test types may include both seen types and unseen types, in our experiments we focus on unseen types.

[1]https://github.com/Websail-NU/OTyper/tree/AAAI-18

3.2 Overview

OTyper outputs a vector for each m_i, where each element in the vector is a probability estimate that m_i belongs to a type. Our intuition is that if mention embeddings are appropriately mapped into a common space with type embeddings, then the mention embeddings will be nearby the correct type embeddings and far away from the incorrect type embeddings—even for types that do not explicitly occur in the training set.

Figure 1 shows the neural network architecture of OTyper. The number of dimensions is listed in parentheses after the name. The arrows represent fully interconnected weight matrices with one exception—the mapping from the mention and type representation to the dot product is a dot product operation of these two representations. Also, OTyper uses a fixed identity matrix for the weights from the type embedding to the type representation. OTyper has a single logistic function output for each <mention, type> pair. Thus, OTyper can have a different number of types in training and testing phase, which is critical for ONET. OTyper maps the mention embedding into a common space with the type embedding and is trained to minimize the dot-product distance between the mention embedding and each of its correct type embeddings.

The high level mathematical formulation of OTyper is as follows:

$$\mathbf{a}_{\text{rep}} = [\mathbf{f}_{\text{m-emb}}, \mathbf{f}_{\text{m-fet}}] * W_{\text{mention}} \tag{1}$$

$$\mathbf{b}_{\text{rep}} = \mathbf{f}_{\text{t-emb}} * W_{\text{type}} \tag{2}$$

$$d = \mathbf{a}_{\text{rep}} \cdot \mathbf{b}_{\text{rep}} \tag{3}$$

$$l = [d, \mathbf{f}_{\text{p-fet}}] * W_l + b \tag{4}$$

$$\hat{y} = sigmoid(l) \tag{5}$$

OTyper concatenates the mention features ($\mathbf{f}_{\text{m-fet}}$) with the mention embedding ($\mathbf{f}_{\text{fet}}$) and projects it into the common space in Equation 1. Equation 2 projects the type embedding, $\mathbf{f}_{\text{t-emb}}$, into the common space. As mentioned above, in OTyper, the type projection is an identity transformation; exploring alternative transformations is an item of future work. Then, the dot product between the projected mention and type is computed in Equation 3. Pattern features vector ($\mathbf{f}_{\text{p-fet}}$) are concatenated with the dot product. In equation 4, the concatenation vector is then fed into a hidden layer, which outputs a scalar. The last layer transforms the hidden layer output with a logistic function, to produce a probability estimate that the given mention is of the given type. To train the model, OTyper minimizes the cross-entropy loss on the training data.

Note that in the training phase, we feed seen type embeddings into OTyper , while during the test phrase, we feed target types which can include unseen types.

In 3.3 and 3.4, we introduce mention and type embeddings. The features that OTyper uses are illustrated in 3.5.

3.3 Mention embedding

The mention embedding in OTyper contains two parts: an entity embedding and a context embedding. OTyper adopts the mention embedding approach proposed for NET in (Shimaoka et al. 2016b), which we describe in this section.

Our entity embedding simply averages the individual word embeddings for the entity, as shown in equation 6. $m_i^{(j)}$ denotes the j^{th} word of i^{th} mention. ml_i is the number of words of i^{th} mention. The function $emb(\cdot)$ returns the word embedding.

$$\mathbf{f}_{\text{e-emb}}(m_i) = \frac{1}{ml_i} \sum_{j=1}^{ml_i} emb(m_i^{(j)}) \tag{6}$$

To generate a context embedding, OTyper trains two bi-LSTMs (Graves 2012) on both sides of context. An attention model is applied to the output states of the bi-LSTMs to get weighted summations on both sides, which are concatenated for form the context embedding.

Equations 7 to 10, from (Shimaoka et al. 2016b), describe how our mention embedding is computed. Equations 7 to 9 formulate how OTyper computes its context embedding. We use $\overrightarrow{o}^l_{ij}$ and $\overleftarrow{o}^l_{ij}$ to denote the bidirectional output states of bi-LSTMs for the j^{th} word in the left side context of m_i, $j \in \{1, \ldots, C\}$, with analogous quantities for the right side. To get the attention weights in Equation 9, the attention model trains a two-layer feed-forward neural network as in Equations 7 and 8. The scalar a^l_{ij} is the weight for the left context output state for the j^{th} word. The right context scalars are analogous. The context embedding is shown in Equation 10.

$$\mathbf{e}^l_{ij}(cl_{ij}) = \tanh(W_e * \begin{bmatrix} \overrightarrow{\mathbf{o}}^l_{ij} \\ \overleftarrow{\mathbf{o}}^l_{ij} \end{bmatrix}) \tag{7}$$

$$\tilde{a}^l_{ij} = \exp(W_a * \mathbf{e}^l_{ij}) \tag{8}$$

$$a^l_{ij} = \frac{\tilde{a}^l_{ij}}{\sum_{j=1}^{C}(\tilde{a}^l_{ij} + \tilde{a}^r_{ij})} \tag{9}$$

$$\mathbf{f}_{\text{c-emb}}(cl_i, cr_i) = \sum_{j=1}^{C}(a^l_{ij} * \begin{bmatrix} \overrightarrow{\mathbf{o}}^l_{ij} \\ \overleftarrow{\mathbf{o}}^l_{ij} \end{bmatrix} + a^r_{ij} * \begin{bmatrix} \overrightarrow{\mathbf{o}}^r_{ij} \\ \overleftarrow{\mathbf{o}}^r_{ij} \end{bmatrix}) \tag{10}$$

The mention embedding (Equation 11) is the concatenation of the entity and context embedding.

$$\mathbf{f}_{\text{m-emb}}(m_i, cl_i, cr_i) = [\mathbf{f}_{\text{e-emb}}(m_i), \mathbf{f}_{\text{c-emb}}(cl_i, cr_i)] \tag{11}$$

We also tried the two alternative context models proposed in (Shimaoka et al. 2016b), one based on averaging the context embeddings and the other based on LSTMs without attention. We found the attention embedding performed the best, so we use it in OTyper.

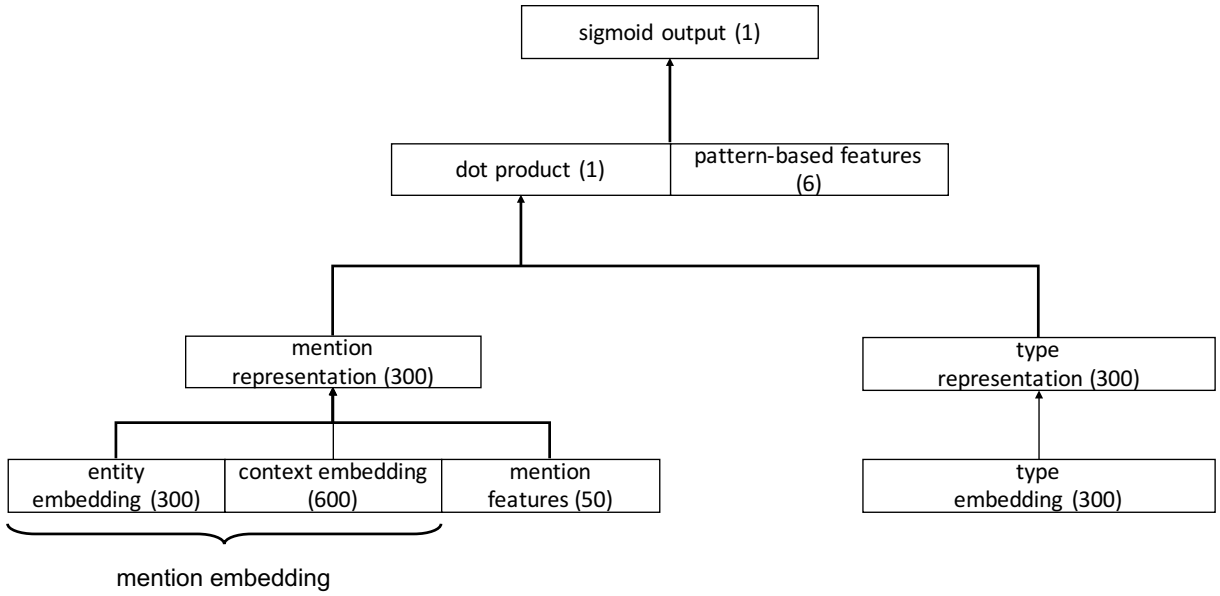

Figure 1: Neural architecture for OTyper. The number of neuronal units is provided for each component in parentheses

3.4 Type embeddings

In equation 12, OTyper computes type embeddings by simply averaging the word embeddings of the words comprising the type name:

$$\mathbf{f}_{\text{t-emb}}(t_i) = \frac{1}{tl_i}\sum_{j=1}^{tl_i} emb(t_i^{(j)}) \tag{12}$$

where $t_i^{(j)}$ denotes the j^{th} word of type t_i. tl_i is the number of words in t_i.

3.5 Features

Shimaoka et al. (2016b) showed substantial F1 score improvement in NET by incorporating mention features such as syntactic features, word shape features and topic features. We also use these mention features in OTyper. Following (Shimaoka et al. 2016b), we represent mention features as binary vectors and use a trainable linear projection to map the binary vector to lower dimension. We use $f_{\text{m-fet}}(m_i)$ to represent mention features for mention i.

Pattern-based features are helpful for ONET. For example, given an entity-type pair, $<e, t>$, if the number of textual occurrences of "e is a t" is low in a large corpus, it is less likely that e is of type t. We use two kinds of pattern based features in OTyper: entity-type features and type-only features. We use a set of hypernym patterns applied to a large corpus taken from the web-is-a database (Seitner et al. 2016). We compute features based on the number of pattern matches for different entities and types. Entity-type features capture the pattern matching information of a specific $<e, t>$ pair. Specifically, entity-type features include: the number of matches, the number of distinct matched patterns, and the number of matched URL domains. Type-only features marginalize entity-type features over types, and can be viewed as a prior over types — i.e. types that appear in more patterns may be the type labels that OTyper should be output more often, all else being equal. Equations 13, 14 are the formulas for entity-type features ($\mathbf{f}_{\text{e-t-fet}}$) and type-only features($\mathbf{f}_{\text{t-o-fet}}$). n_m, n_p and n_u are the number of matches, the number of distinct matched patterns and the number of URL domains for $<e, t>$ separately. Pattern based features (in Equation 15) combines entity-type features with type-only features.

$$\mathbf{f}_{\text{e-t-fet}}(e_i, t_i) = [n_m(e_i, t_i), n_p(e_i, t_i), n_u(e_i, t_i)] \tag{13}$$

$$\mathbf{f}_{\text{t-o-fet}}(t_i) = \sum_i \mathbf{f}_{\text{e-t-fet}}(e_i, t_i) \tag{14}$$

$$\mathbf{f}_{\text{p-fet}}(e_i, t_i) = [\mathbf{f}_{\text{e-t-fet}}(e_i, t_i), \mathbf{f}_{\text{t-o-fet}}(t_i)] \tag{15}$$

4 Experiments

This section evaluates OTyper and answers three questions:

- How well can OTyper label unseen types?
- How much do similar training types help—does unseen type accuracy correlate with how similar the training type embeddings are to the unseen type embeddings?
- How much does each feature impact the accuracy of OTyper?

4.1 Experimental setup

We evaluate OTyper on two datasets: FIGER(GOLD) and MSH-WSD.

- **FIGER(GOLD)** is a benchmark dataset for fine-grained NET. The training and development FIGER(GOLD) data were generated from Wikipedia text (Ling and Weld

2012). The FIGER(GOLD) test dataset consists of manually annotated newspaper articles. Each mention has 113 binary type labels, where the types were derived from Freebase.

- **MSH-WSD** is a word sense disambiguation dataset that was automatically collected from the Unified Medical Language System (UMLS) Metathesaurus and the manual MeSH indexing of MEDLINE (Jimeno-Yepes, McInnes, and Aronson 2011). We type each MSH-WSD mention using UMLS tags. Specifically, we define the correct types for a mention to be all of its ancestors in the UMLS taxonomy tree. We randomly select 80% of the mentions for training, 10% for development and 10% for test. There are 1387 different types in MSH-WSD.

We use published mention features for the FIGER(GOLD) dataset from (Shimaoka et al. 2016b). There are no mention features available for MSH-WSD. The pattern features used in our experiments are extracted from the web-is-a database (Seitner et al. 2016). Mention and type embeddings are taken from pre-trained GloVe-840B (Pennington, Socher, and Manning 2014) word embeddings.

For all experiments, the dimension of the common space is 300. As mentioned above, we set W_{type} to be the identity matrix. The rest of hyper-parameters are the same as in (Shimaoka et al. 2016b). The projection of the mention features is 50 dimensional. The Adam optimizer (Kingma and Ba 2014) with a learning rate 0.001 is applied to minimize the loss. The model is trained for 5 epochs with batch size 1,000. Dropout with keep probability of 0.5 is applied to the mention embeddings and mention features. The context window size is 10. We present the test performance of the model that performed best of the development set.

Our evaluation metric is weighted AUC-ROC score, which computes an AUC-ROC score for each type and then averages the scores weighted by the type frequency.

4.2 Unseen Type labeling

To evaluate how well OTyper labels unseen types, we split test types into 10 parts and do 10-fold cross-validation on the types. For each fold, only type labels *other than* the test types are utilized in the training and development set.

We compare OTyper with two baseline models. Pattern-based methods are a canonical approach for identifying hypernym relations. So, our first baseline is a pattern-based model. It uses the number of matches in web-is-a for <e,t> as its score (Sang 2007), and forms a relatively strong baseline. The second baseline is based on word embeddings. It trains a logistic regression model to classify the vector difference between the entity embedding for e and the type embedding for t. Vector differences between embeddings have been shown to reflect relation information (Mikolov, Yih, and Zweig 2013). In addition, to provide an upper bound on accuracy, we also evaluate OTyper in a fully supervised setting in which all types are available in training.

Table 1 presents the results on FIGER(GOLD). The results show that using only pattern-based features or embeddings cannot solve the ONET task, because ONET is context-sensitive whereas the pattern features and embeddings are context-insensitive. OTyper achieves an AUC-ROC of 0.870 and outperforms the two baselines by substantial margins. The pattern baseline is better than the embedding baseline in these experiments. The supervised upper bound model gets an AUC-ROC score of 0.943. It is notable that OTyper can score relatively close to the performance of the supervised model, given that OTyper must solve the much more challenging ONET task in which the test types do not occur in the labeled training examples.

Model	AUC-ROC
Pattern baseline	0.639
Embedding baseline	0.413
OTyper	0.870
Supervised upper bound	0.943

Table 1: Unseen type weighted AUC-ROC comparison on FIGER(GOLD). OTyper outperforms the baselines.

Model	AUC-ROC
Pattern baseline	0.005
Embedding baseline	0.350
OTyper	0.780
Supervised upper bound	0.891

Table 2: Unseen type-weighted AUC-ROC comparison on the MSH-WSD dataset. OTyper outperforms the baselines.

Table 2 shows the results on MSH-WSD. Again, OTyper achieves much higher weighted type AUC-ROC score compared to the baselines. However, the AUC-ROC is not as high as in FIGER(GOLD) because MSH-WSD has a much higher number of types than FIGER(GOLD), and also lacks mention features, which makes the ONET task harder than in FIGER(GOLD). Further, since MSH-WSD type comes from biomedical domain, they rarely show up in the Web corpus used to form the web-is-a database. Thus, pattern-based features are sparse. In fact, out of 51 million <e, t> pairs in MSH-WSD only 817 of them have non-zero pattern matches. This implies that the pattern-based features are not informative here, and is the reason that the pattern-based baseline gets almost zero AUC-ROC score on MSH-WSD data. Due to the feature limitations of MSH-WSD, we only use FIGER(GOLD) in the following experiments.

Note that although OTyper achieves high weighted type AUC scores, if we evaluate on an F1 metric over unseen types, OTyper achieves a score of 0.26. This indicates that OTyper works well for ranking the unseen types, but not that well for classification.

4.3 Influence of training types

This subsection attempts to explain the performance of OTyper by analyzing how much the unseen type AUC-ROC correlates with how similar the training type embeddings are to the target unseen type embedding. We use cosine similarity in Equation 16 to measure the distance between types:

$$cos\text{-}similarity(\mathbf{a}, \mathbf{b}) = \frac{\mathbf{a} \cdot \mathbf{b}}{\|\mathbf{a}\|_2 \|\mathbf{b}\|_2} \quad (16)$$

For these experiments, we focus on the 11 of our 41 test types that occur at least ten times in our test dataset. For each of the 11 types, we hold the type out from training and development sets and train three different OTyper models. In the first model, the training and development types contain all other types other than the test type. We name this model *All*. In the second and third models, the three most and least similar types from the test type are removed from the training and development datasets. We call them *top-3* and *bot-3* model. If OTyper is utilizing similar types to the test type in the training set as a source of information, we would expect *top-3*, which removes the most similar types, to substantially underperform *bot-3*, which removes dissimilar types.

Type name	All	Top-3	Bot-3
location	0.739	0.692	0.742
person	0.885	0.82	0.897
organization	0.880	0.819	0.811
city	0.917	0.856	0.921
country	0.857	0.787	0.893
company	0.849	0.611	0.842
sports team	0.862	0.827	0.873
athlete	0.957	0.897	0.953
building	0.929	0.885	0.894
educational institution	0.888	0.720	0.904
time	0.793	0.863	0.801
Average	**0.869**	**0.798**	**0.866**

Table 3: Performance of OTyper when holding out types that are similar (*Top-3*) or dissimilar (*Bot-3*) to the target type, compared to keeping all training types (*All*). We report AUC-ROC on the FIGER(GOLD) dataset. On average, removing similar types hurts performance, whereas removing dissimilar types has negligible impact.

Table 3 shows the type AUC-ROCs of all three models. On average, *bot-3* achieves 0.068 higher AUC-ROC score than *top-3*. The result suggests that to predict unseen types, the similar types in the training data are more informative than dissimilar types. We also observed that average AUC-ROC of *All* is almost the same with bot-3. So, dissimilar types do not affect unseen type prediction.

4.4 Feature analysis

This section investigates whether having mention- and pattern-based features is helpful in predicting unseen types. We remove each feature from our model separately and evaluate our model using 10 fold cross validation as in 4.2. The results are summarized in Table 4.

Table 4 shows that removing any of the features results in some AUC-ROC decrease. AUC-ROC does not decease very much when mention features are removed, which is somewhat surprising. As additional analysis, we find that when mention features are removed, the AUC-ROCs increases for 18 unseen types, decreases for 21 unseen types, and remains unchanged for the other two. Thus, there appears to be no clear advantage in using mention features. The pattern-based features are found to be more informative for FIGER(GOLD) dataset.

Model	Weighted AUC-ROC
OTyper	0.870
OTyper (- mention features)	0.863
OTyper (- entity-type features)	0.842
OTyper (- type-only features)	0.848

Table 4: Impact of features on the FIGER(GOLD) dataset. The pattern-based features are most valuable for this dataset.

4.5 Supervised setting

Our focus is on ONET in which types can be unseen. In a supervised setting, where all types are seen, we would expect OTyperperforms similar to that of the recent state-of-the-art model in (Shimaoka et al. 2016b). To verify this, we evaluate OTyper against the model from (Shimaoka et al. 2016b) (NFGEC for short) on FIGER(GOLD) using the same evaluation set-up in that work, i.e. strict accuracy, loose macro F1, and loose micro F1 scores. For this experiment, all 113 types are seen types.

For NFGEC, we use the attentive context encoder with mention features, which is the setting that achieves the highest FIGER(GOLD) accuracy and F1 scores in (Shimaoka et al. 2016b). We use two feature settings for OTyper. The first setting includes only mention features. This setting uses exactly the same input data as NFGEC, i.e. mention embeddings, context embeddings and mention features. The second setting includes all features in OTyper, i.e. mention, entity-type and type-only features. Table 5 shows the results. When OTyper uses the same input data with NFGEC, OTyper gets slightly lower accuracy and F1 scores than NFGEC. When all features are included, OTyper gets similar accuracy and F1 scores with NFGEC. Overall, OTyper performs comparably to NFGEC in the supervised setting.

Model	Acc.	F1 Macro	F1 Micro
NFGEC	0.597	0.790	0.754
OTyper (mention features)	0.584	0.776	0.752
OTyper (all features)	0.595	0.779	0.759

Table 5: Comparison in the supervised setting. OTyper achieves comparable F1 to the state-of-the-art NFGEC.

5 Conclusion

In this paper, we introduced the task of *Open* Named Entity Typing (ONET), which is NET when the set of target types is not known in advance. We proposed OTyper, a neural network architecture for ONET. OTyper relies on type embeddings in order to extend to unseen types. The experimental results demonstrate that on unseen types OTyper outperforms two baseline models and achieves a weighted AUC-

ROC of 0.870 on the benchmark FIGER(GOLD) dataset, and a score of 0.78 on the MSH-WSD dataset. Our results can serve as a baseline for future ONET systems. Finally, our analysis revealed that similar training types provide more information for unseen type prediction than dissimilar training types do.

Acknowledgments

This work was supported in part by NSF Grant IIS-1351029 and the Allen Institute for Artificial Intelligence. We thank Oren Etzioni for suggesting we look at the ONET task. We also thank Mohammed Alam, David Demeter, Jared Fernandez, Thanapon Noraset, and Yiben Yang for helpful discussion.

References

Abdul-Hamid, A., and Darwish, K. 2010. Simplified feature set for arabic named entity recognition. In *Proceedings of the 2010 Named Entities Workshop*, 110–115. Association for Computational Linguistics.

Carreras, X.; Marquez, L.; and Padró, L. 2002. Named entity extraction using adaboost. In *proceedings of the 6th conference on Natural language learning-Volume 20*, 1–4. Association for Computational Linguistics.

Chang, M.-W.; Ratinov, L.-A.; Roth, D.; and Srikumar, V. 2008. Importance of semantic representation: Dataless classification. In *AAAI*, volume 2, 830–835.

Chinchor, N., and Robinson, P. 1997. Muc-7 named entity task definition. In *Proceedings of the 7th Conference on Message Understanding*, volume 29.

Collins, M., and Singer, Y. 1999. Unsupervised models for named entity classification. In *Proceedings of the joint SIGDAT conference on empirical methods in natural language processing and very large corpora*, 100–110.

Elsner, M.; Charniak, E.; and Johnson, M. 2009. Structured generative models for unsupervised named-entity clustering. In *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, 164–172. Association for Computational Linguistics.

Graves, A. 2012. Supervised sequence labelling. *Supervised sequence labelling with recurrent neural networks* 5–13.

Grishman, R., and Sundheim, B. 1996. Message understanding conference-6: A brief history. In *COLING 1996 Volume 1: The 16th International Conference on Computational Linguistics*, volume 1.

Hearst, M. A. 1992. Automatic acquisition of hyponyms from large text corpora. In *Proceedings of the 14th conference on Computational linguistics-Volume 2*, 539–545. Association for Computational Linguistics.

Huang, L.; May, J.; Pan, X.; and Ji, H. 2016. Building a fine-grained entity typing system overnight for a new x (x= language, domain, genre). *arXiv preprint arXiv:1603.03112*.

Huang, L.; May, J.; Pan, X.; Ji, H.; Ren, X.; Han, J.; Zhao, L.; and Hendler, J. A. 2017. Liberal entity extraction: Rapid construction of fine-grained entity typing systems. *Big Data* 5(1):19–31.

Jimeno-Yepes, A. J.; McInnes, B. T.; and Aronson, A. R. 2011. Exploiting mesh indexing in medline to generate a data set for word sense disambiguation. *BMC bioinformatics* 12(1):223.

Kingma, D., and Ba, J. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Kuru, O.; Can, O. A.; and Yuret, D. 2016. Charner: Character-level named entity recognition. In *COLING*, 911–921.

Lee, C.; Hwang, Y.-G.; Oh, H.-J.; Lim, S.; Heo, J.; Lee, C.-H.; Kim, H.-J.; Wang, J.-H.; and Jang, M.-G. 2006. Fine-grained named entity recognition using conditional random fields for question answering. In *AIRS*, 581–587. Springer.

Ling, X., and Weld, D. S. 2012. Fine-grained entity recognition. In *AAAI*.

Mikolov, T.; Yih, W.-t.; and Zweig, G. 2013. Linguistic regularities in continuous space word representations. In *hlt-Naacl*, volume 13, 746–751.

Palatucci, M.; Pomerleau, D.; Hinton, G. E.; and Mitchell, T. M. 2009. Zero-shot learning with semantic output codes. In *Advances in neural information processing systems*, 1410–1418.

Pennington, J.; Socher, R.; and Manning, C. D. 2014. Glove: Global vectors for word representation. In *EMNLP*, volume 14, 1532–1543.

Rabinovich, M., and Klein, D. 2017. Fine-grained entity typing with high-multiplicity assignments. *arXiv preprint arXiv:1704.07751*.

Ratinov, L., and Roth, D. 2009. Design challenges and misconceptions in named entity recognition. In *Proceedings of the Thirteenth Conference on Computational Natural Language Learning*, 147–155. for Computational Linguistics.

Ritter, A.; Clark, S.; Etzioni, O.; et al. 2011. Named entity recognition in tweets: an experimental study. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, 1524–1534. Association for Computational Linguistics.

Ritter, A.; Soderland, S.; and Etzioni, O. 2009. What is this, anyway: Automatic hypernym discovery. In *AAAI Spring Symposium: Learning by Reading and Learning to Read*, 88–93.

Sang, E. T. K. 2007. Extracting hypernym pairs from the web. In *Proceedings of the 45th annual meeting of the ACL on interactive poster and demonstration sessions*, 165–168. Association for Computational Linguistics.

Seitner, J.; Bizer, C.; Eckert, K.; Faralli, S.; Meusel, R.; Paulheim, H.; and Ponzetto, S. P. 2016. A large database of hypernymy relations extracted from the web. In *LREC*.

Shimaoka, S.; Stenetorp, P.; Inui, K.; and Riedel, S. 2016a. An attentive neural architecture for fine-grained entity type classification. *arXiv preprint arXiv:1604.05525*.

Shimaoka, S.; Stenetorp, P.; Inui, K.; and Riedel, S. 2016b.

Neural architectures for fine-grained entity type classification. *arXiv preprint arXiv:1606.01341*.

Snow, R.; Jurafsky, D.; and Ng, A. Y. 2005. Learning syntactic patterns for automatic hypernym discovery. In *Advances in neural information processing systems*, 1297–1304.

Socher, R.; Ganjoo, M.; Manning, C. D.; and Ng, A. 2013. Zero-shot learning through cross-modal transfer. In *Advances in neural information processing systems*, 935–943.

Tjong Kim Sang, E. F., and De Meulder, F. 2003. Introduction to the conll-2003 shared task: Language-independent named entity recognition. In *Proceedings of the seventh conference on Natural language learning at HLT-NAACL 2003-Volume 4*, 142–147. Association for Computational Linguistics.

Yao, L.; Riedel, S.; and McCallum, A. 2013. Universal schema for entity type prediction. In *Proceedings of the 2013 workshop on Automated knowledge base construction*, 79–84. ACM.

Yogatama, D.; Gillick, D.; and Lazic, N. 2015. Embedding methods for fine grained entity type classification. In *ACL (2)*, 291–296.