Learning to Denoise Distantly-Labeled Data for Entity Typing

Yasumasa Onoe and Greg Durrett
Department of Computer Science
The University of Texas at Austin
{yasumasa, gdurrett}@cs.utexas.edu

Abstract

Distantly-labeled data can be used to scale up training of statistical models, but it is typically noisy and that noise can vary with the distant labeling technique. In this work, we propose a two-stage procedure for handling this type of data: denoise it with a learned model, then train our final model on clean and denoised distant data with standard supervised training. Our denoising approach consists of two parts. First, a filtering function discards examples from the distantly labeled data that are wholly unusable. Second, a relabeling function repairs noisy labels for the retained examples. Each of these components is a model trained on synthetically-noised examples generated from a small manually-labeled set. We investigate this approach on the ultra-fine entity typing task of Choi et al. (2018). Our baseline model is an extension of their model with pre-trained ELMo representations, which already achieves state-of-the-art performance. Adding distant data that has been denoised with our learned models gives further performance gains over this base model, outperforming models trained on raw distant data or heuristically-denoised distant data.

1 Introduction

With the rise of data-hungry neural network models, system designers have turned increasingly to unlabeled and weakly-labeled data in order to scale up model training. For information extraction tasks such as relation extraction and entity typing, distant supervision (Mintz et al., 2009) is a powerful approach for adding more data, using a knowledge base (Del Corro et al., 2015; Rabinovich and Klein, 2017) or heuristics (Ratner et al., 2016; Hancock et al., 2018) to automatically label instances. One can treat this data just like any other supervised data, but it is noisy; more effective approaches employ specialized probabilistic models (Riedel et al., 2010; Ratner et al., 2018a), capturing its interaction with other supervision (Wang and Poon, 2018) or breaking down aspects of a task on which it is reliable (Ratner et al., 2018b). However, these approaches often require sophisticated probabilistic inference for training of the final model. Ideally, we want a technique that handles distant data just like supervised data, so we can treat our final model and its training procedure as black boxes.

This paper tackles the problem of exploiting weakly-labeled data in a structured setting with a two-stage denoising approach. We can view a distant instance’s label as a noisy version of a true underlying label. We therefore learn a model to turn a noisy label into a more accurate label, then apply it to each distant example and add the resulting denoised examples to the supervised training set. Critically, the denoising model can condition on both the example and its noisy label, allowing it to fully leverage the noisy labels, the structure of the label space, and easily learnable correspondences between the instance and the label.

Concretely, we implement our approach for the task of fine-grained entity typing, where a single entity may be assigned many labels. We learn two denoising functions: a relabeling function takes an entity mention with a noisy set of types and returns a cleaner set of types, closer to what manually labeled data has. A filtering function discards examples which are deemed too noisy to be useful. These functions are learned by taking manually-labeled training data, synthetically adding noise to it, and learning to denoise, similar to a conditional variant of a denoising autoencoder (Vincent et al., 2008). Our denoising models embed both entities and labels to make their predictions, mirroring the structure of the final entity typing model itself.

We evaluate our model following Choi et al. (2018). We chiefly focus on their ultra-fine en-
According to the review aggregator Rotten Tomatoes, 89% of critics gave [the film] positive reviews.

The film is based on a hit London and New York play, which was based on a best-selling book.

“A pretty good day all round,” said [Gascoyne, a British veteran of stints with the original Tyrrell team] in a roller-coaster F1 career.

Djokovic lost to [Rafael Nadal] on Monday, in a rain-delayed U.S. Open final.

Figure 1: Examples selected from the Ultra-Fine Entity Typing dataset of Choi et al. (2018). (a) A manually-annotated example. (b) The head word heuristic functioning correctly but missing types in (a). (c) Entity linking providing the wrong types. (d) Entity linking providing correct but incomplete types.

2.1 Case Study: Ultra-Fine Entity Typing

The primary task we address here is the fine-grained entity typing task of Choi et al. (2018). Instances in the corpus are assigned types from a vocabulary of more than 10,000 types, which are divided into three classes: 9 general types, 121 fine-grained types, and 10, 201 ultra-fine types. This dataset consists of 6K manually annotated examples and approximately 25M distantly-labeled examples. 5M examples are collected using entity linking (EL) to link mentions to Wikipedia and gather types from information on the linked pages. 20M examples (HEAD) are generated by extracting nominal head words from raw text and treating these as singular type labels.

Figure 1 shows examples from these datasets which illustrate the challenges in automatic annotation using distant supervision. The manually-annotated example in (a) shows how numerous the gold-standard labeled types are. By contrast, the HEAD example (b) shows that simply treating the head word as the type label, while correct in this case, misses many valid types, including more general types. The EL example (c) is incorrectly annotated as region, whereas the correct coarse type is actually person. This error is characteristic of entity linking-based distant supervision since identifying the correct link is a challenging problem in and of itself (Milne and Witten, 2008): in this case, Gascoyne is also the name of a region in Western Australia. The EL example in (d) has reasonable types; however, human annotators could choose more types (grayed out) to describe the mention more precisely. The average number of types annotated by humans is 5.4 per example while the two distant supervision techniques combined yields 1.5 types per example on average.

In summary, distant supervision can (1) produce

predicts y given x in the inference phase.
Djokovic lost to Rafael Nadal on Monday, ...

An athlete who plays tennis

Filter model predicts whether the example should be kept at all; if it is kept, the Relabel model attempts to automatically expand the label set. \( \Phi_m \) is a mention encoder, which can be a state-of-the-art entity typing model. \( \Phi_t \) encodes noisy types from distant supervision.

3 Denoising Model

To handle the noisy data, we propose to learn a denoising model as shown in Figure 2. This denoising model consists of filtering and relabeling functions to discard and relabel examples, respectively; these rely on a shared mention encoder and type encoder, which we describe in the following sections. The filtering function is a binary classifier that takes these encoded representations and predicts whether the example is good or bad. The relabeling function predicts a new set of labels for the given example.

We learn these functions in a supervised fashion. Training data for each is created through synthetic noising processes applied to the manually-labeled data, as described in Sections 3.3 and 3.4.

For the entity typing task, each example \((x, y)\) takes the form \((s, m, t)\), where \(s\) is the sentence, \(m\) is the mention span, and \(t\) is the set of types (either clean or noisy).

3.1 Mention Encoder

This encoder is a function \( \Phi_m(s, m) \) which maps a sentence \( s \) and mention \( m \) to a real-valued vector \( v_m \). This allows the filtering and relabeling function to recognize inconsistencies between the given example and the provided types. Note that these inputs \( s \) and \( m \) are the same as the inputs for the supervised version of this task; we can therefore share an encoder architecture between our denoising model and our final typing model. We use an encoder following Choi et al. (2018) with a few key differences, which are described in Section 4.

3.2 Type Encoder

The second component of our model is a module which produces a vector \( v_t = \Phi_t(t) \). This is an encoder of an unordered bag of types. Our basic type encoder uses trainable vectors as embeddings for each type and combines these with summing. That is, the noisy types \( t_1, \ldots, t_m \) are embedded into type vectors \( \{t_1, \ldots, t_m\} \). The final embedding of the type set \( t = \sum_j t_j \).

Type Definition Encoder Using trainable type embeddings exposes the denoising model to potential data sparsity issues, as some types appear only a few or zero times in the training data. Therefore, we also assign each type a vector based on its definition in WordNet (Miller, 1995). Even low-frequent types are therefore assigned a plausible embedding.

Let \( w_{jl} \) denote the \( i \)th word of the \( j \)th type’s most common WordNet definition. Each \( w_{jl} \) is embedded using GloVe (Pennington et al., 2014). The resulting word embedding vectors \( w_{jl} \) are fed into a bi-LSTM (Hochreiter and Schmidhuber, 1997; Graves and Schmidhuber, 2005), and a concatenation of the last hidden states in both directions is used as the definition representation \( w_j \). The final representation of the definitions is the sum over these vectors for each type: \( w = \sum_j w_j \).

We found this technique to be more effective than using pretrained vectors from GloVe or ELMo. It gave small improvements on an intrinsic evaluation over not incorporating it; results are omitted due to space constraints.
\[ \sum_k w^k. \]

Our final \( v_t = [t; w] \), the concatenation of the type and definition embedding vectors.

### 3.3 Filtering Function

The filtering function \( f \) is a binary classifier designed to detect examples that are completely mislabeled. Formally, \( f \) is a function mapping a labeled example \((s, m, t)\) to a binary indicator \( z \) of whether this example should be discarded or not.

In the forward computation, the feature vectors \( v_s, v_m \) and \( v_t \) are computed using the mention and type encoders. The model prediction is defined as \( P(\text{error}) = \sigma (u^T \text{Highway}([v_m; v_t])) \), where \( \sigma \) is a sigmoid function, \( u \) is a parameter vector, and \text{Highway}(\cdot) is a 1-layer highway network (Srivastava et al., 2015). We can apply \( f \) to each distant pair in our distant dataset \( D' \) and discard any example predicted to be erroneous \( (P(\text{error}) > 0.5) \).

**Training data** We do not know a priori which examples in the distant data should be discarded, and labeling these is expensive. We therefore construct synthetic training data \( D_{\text{error}} \) for \( f \) based on the manually labeled data \( D \). For 30% of the examples in \( D \), we replace the gold types for that example with non-overlapping types taken from another example. The intuition for this procedure follows Figure 1: we want to learn to detect examples in the distant data like \textit{Gascoyne} where heuristics like entity resolution have misfired and given a totally wrong label set.

Formally, for each selected example \((s, m, t)\), we repeatedly draw another example \((s', m', t')\) from \( D \) until we find \( t'_{\text{err}} \) that does not have any common types with \( t \). We then create a positive training example \((s, m, t', z = 1)\). We create a negative training example \((s, m, t', z = 0)\) using the remaining 70% of examples. \( f \) is trained on \( D_{\text{error}} \) using binary cross-entropy loss.

### 3.4 Relabeling Function

The relabeling function \( g \) is designed to repair examples that make it through the filter but which still have errors in their type sets, such as missing types as shown in Figure 1b and 1d. \( g \) is a function from a labeled example \((s, m, t)\) to an improved type set \( \tilde{t} \) for the example.

Our model computes feature vectors \( v_s, v_m \) and \( v_t \) by the same procedure as the filtering function \( f \). The decoder is a linear layer with parameters \( D \in \mathbb{R}^{|V^d| \times (d_m + d_t)} \). We compute \( e = \sigma (D [v_m; v_t]) \), where \( \sigma \) is an element-wise sigmoid operation designed to give binary probabilities for each type.

Once \( g \) is trained, we make a prediction \( \tilde{t} \) for each \((s, m, t) \in D' \) and replace \( t \) by \( \tilde{t} \) to create the denoised data \( \tilde{D}_{\text{denoise}} = \{(s, m, \tilde{t}), \ldots \} \). For the final prediction, we choose all types \( t' \) where \( e_{t'} > 0.5 \), requiring at least two types to be present or else we discard the example.

**Training data** We train the relabeling function \( g \) on another synthetically-noised dataset \( D_{\text{drop}} \) generated from the manually-labeled data \( D \). To mimic the type distribution of the distant-labeled examples, we take each example \((s, m, t)\) and randomly drop each type with a fixed rate 0.7 independent of other types to produce a new type set \( t' \). We perform this process for all examples in \( D \) and create a noised training set \( D_{\text{drop}} \) where a single training example is \((s, m, t', t), g \) is trained on \( D_{\text{drop}} \) with a binary classification loss function over types used in Choi et al. (2018), described in the next section.

One can think of \( g \) as a type of denoising autoencoder (Vincent et al., 2008) whose reconstructed types \( \tilde{t} \) are conditioned on \( v \) as well as \( t \).

### 4 Typing Model

In this section, we define the sentence and mention encoder \( \Phi_m \), which is used both in the denoising model as well as in the final prediction task. We extend previous attention-based models for this task (Shimaoka et al., 2017; Choi et al., 2018). At a high level, we have an instance encoder \( \Phi_m \) that returns a vector \( v_m \in \mathbb{R}^{d_m} \), then multiply the output of this encoding by a matrix and apply a sigmoid to get a binary prediction for each type as a probability of that type applying.

Figure 3 outlines the overall architecture of our typing model. The encoder \( \Phi_m \) consists of four vectors: a sentence representation \( s \), a word-level mention representation \( m_{\text{word}} \), a character-level mention representation \( m_{\text{char}} \), and a headword mention vector \( m_{\text{head}} \). The first three of these were employed by Choi et al. (2018). We have modified the mention encoder with an additional bi-LSTM to better encode long mentions, and additionally used the headword embedding directly in order to focus on the most critical word. These pieces use pretrained contextualized word embeddings (ELMo) (Peters et al., 2018) as input.

**Pretrained Embeddings** Tokens in the sentence \( s \) are converted into contextualized word vectors
The Char-CNN board attention mechanism for the word-level mention representation: The concatenated hidden states of both directions into a bi-LSTM with hidden dimension is mention’s contextualized word vectors information. For the word-level representation, we use both word and character information to obtain a mention representation. 

**Mention Encoder** We obtain a mention representation, we use both word and character information. For the word-level representation, the mention’s contextualized word vectors $m'$ are fed into a bi-LSTM with hidden dimension is $d_{hid}$. The concatenated hidden states of both directions are summed by a span attention layer to form the word-level mention representation: $m_{\text{word}} = \text{Attention}([\text{bi-LSTM}(m')])$.

Second, a character-level representation is computed for the mention. Each character is embedded and then a 1-D convolution (Collobert et al., 2011) is applied over the characters of the mention. This gives a character vector $m_{\text{char}}$.

Finally, we take the contextualized word vector of the headword $m_{\text{head}}$ as a third component of our representation. This can be seen as a residual connection (He et al., 2016) specific to the mention head word. We find the headwords in the mention spans by parsing those spans in isolation using the spaCy dependency parser (Honnibal and Johnson, 2015). Empirically, we found this to be useful on long spans, when the span attention would often focus on incorrect tokens.

The final representation of the input $x$ is a concatenation of the sentence, the word- & character-level mention, and the mention headword representations, $v = [s; m_{\text{word}}; m_{\text{char}}; m_{\text{head}}] \in \mathbb{R}^{d_{v}}$.

**Decoder** We treat each label prediction as an independent binary classification problem. Thus, we compute a score for each type in the type vocabulary $V$. Similar to the decoder of the relabeling function, we compute $e = \sigma(\mathbf{E}v)$, where $\mathbf{E} \in \mathbb{R}^{|V| \times d_{v}}$ and $e \in \mathbb{R}^{|V|}$. For the final prediction, we choose all types $t_{\ell}$ where $e_{\ell} > 0.5$. If none of $e_{\ell}$ is greater than 0.5, we choose $t_{\ell} = \arg\max e$ (the single most probable type).

**Loss Function** We use the same loss function as Choi et al. (2018) for training. This loss partitions the labels in general, fine, and ultra-fine classes, and only treats an instance as an example for types of the class in question if it contains a label for that class. More precisely:

$$
\mathcal{L} = \mathcal{L}_{\text{general}} \mathbb{1}_{\text{general}}(t) + \mathcal{L}_{\text{fine}} \mathbb{1}_{\text{fine}}(t) + \mathcal{L}_{\text{ultra-fine}} \mathbb{1}_{\text{ultra-fine}}(t),
$$

where $\mathcal{L}_{\text{...}}$ is a loss function for a specific type class: general, fine-grained, or ultra-fine, and $\mathbb{1}_{\text{...}}(t)$ is an indicator function that is active when one of the types $t$ is in the type class. Each $\mathcal{L}_{\text{...}}$ is a sum of binary cross-entropy losses over all types in that category. That is, the typing problem is viewed as independent classification for each type.

Note that this loss function already partially repairs the noise in distant examples from missing labels: for example, it means that examples from HEAD do not count as negative examples for general types when these are not present. However, we show in the next section that this is not sufficient for denoising.

**Implementation Details** The settings of hyper-parameters in our model largely follows Choi et al. (2018) and recommendations for using the pre-
trained ELMo-Small model. The word embedding size $d_{\text{ELMo}}$ is 1024. The type embedding size and the type definition embedding size are set to 1024. For most of other model hyperparameters, we use the same settings as Choi et al. (2018): $d_{\text{loc}} = 50$, $d_{\text{hid}} = 100$, $d_{\text{char}} = 100$. The number of filters in the 1-d convolutional layer is 50. Dropout is applied with $p = 0.2$ for the pretrained embeddings, and $p = 0.5$ for the mention representations. We limit sentences to 50 words and mention spans to 20 words for computational reasons. The character CNN input is limited to 25 characters; most mentions are short, so this still captures subword information in most cases. The batch size is set to 100. For all experiments, we use the Adam optimizer (Kingma and Ba, 2014). The initial learning rate is set to 2e-03. We implement all models using PyTorch. To use ELMo, we consult the AllenNLP source code.

5 Experiments

Ultra-Fine Entity Typing We evaluate our approach on the ultra-fine entity typing dataset from Choi et al. (2018). The 6K manually-annotated English examples are equally split into the training, development, and test examples by the authors of the dataset. We generate synthetically-noised data, $D_{\text{error}}$ and $D_{\text{drop}}$, using the 2K training set to train the filtering and relabeling functions, $f$ and $h$. We randomly select 1M EL and 1M HEAD examples and use them as the noisy data $D'$. Our augmented training data is a combination of the manually-annotated data $D$ and $D'_{\text{denoised}}$.

OntoNotes In addition, we investigate if denoising leads to better performance on another dataset. We use the English OntoNotes dataset (Gillick et al., 2014), which is a widely used benchmark for fine-grained entity typing systems. The original training, development, and test splits contain 250K, 2K, and 9K examples respectively. Choi et al. (2018) created an augmented training set that has 3.4M examples. We also construct our own augmented training sets with/without denoising using our noisy data $D'$, using the same label mapping from ultra-fine types to OntoNotes types described in Choi et al. (2018).

5.1 Ultra-Fine Typing Results

We first compare the performance of our approach to several benchmark systems, then break down the improvements in more detail. We use the model architecture described in Section 4 and train it on the different amounts of data: manually labeled only, naive augmentation (adding in the raw distant data), and denoised augmentation. We compare our model to Choi et al. (2018) as well as to BERT (Devlin et al., 2018), which we fine-tuned for this task. We adapt our task to BERT by forming an input sequence "[CLS] sentence [SEP] mention [SEP]" and assign the segment embedding A to the sentence and B to the mention span. Then, we take the output vector at the position of the [CLS] token (i.e., the first token) as the feature vector v, analogous to the usage for sentence pair classification tasks. The BERT model is fine-tuned on the 2K manually annotated examples. We use the pretrained BERT-Base, un-cased model with a step size of 2e-05 and batch size 32.

Results Table 1 compares the performance of these systems on the development set. Our model with no augmentation already matches the system of Choi et al. (2018) with augmentation, and incorporating ELMo gives further gains on both precision and recall. On top of this model, adding the distantly-annotated data lowers the performance; the loss function-based approach of (Choi et al., 2018) does not sufficiently mitigate the noise in this data. However, denoising makes the distantly-annotated data useful, improving recall by a substantial margin especially in the general class. A possible reason for this is that the relabeling function tends to add more general types given finer types. BERT performs similarly to ELMo with denoised distant data. As can be seen in the performance breakdown, BERT gains from improvements in recall in the fine class.

Table 2 shows the performance of all settings on the test set, with the same trend as the performance on the development set. Our approach outperforms the concurrently-published Xiong et al. (2019); however, that work does not use ELMo. Their improved model could be used for both denoised and non-denoised distant data.

---

2https://allennlp.org/elmo
3The code for experiments is available at https://github.com/yasumassonoe/DenoiseET
4https://github.com/google-research/bert
Table 1: Macro-averaged P/R/F1 on the dev set for the entity typing task of Choi et al. (2018) comparing various systems. ELMo gives a substantial improvement over baselines. Over an ELMo-equipped model, data augmentation using the method of Choi et al. (2018) gives no benefit. However, our denoising technique allows us to effectively incorporate distant data, matching the results of a BERT model on this task (Devlin et al., 2018).

Table 2: Macro-averaged P/R/F1 on the test set for the entity typing task of Choi et al. (2018). Our denoising approach gives substantial gains over naive augmentation and matches the performance of a BERT model.

<table>
<thead>
<tr>
<th>Model</th>
<th>Total</th>
<th>General</th>
<th>Fine</th>
<th>Ultra-Fine</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>P</td>
<td>R</td>
<td>F1</td>
<td>P</td>
</tr>
<tr>
<td>Ours + GloVe w/o augmentation</td>
<td>47.6</td>
<td>23.3</td>
<td>31.0</td>
<td></td>
</tr>
<tr>
<td>Ours + ELMo w/o augmentation</td>
<td>55.8</td>
<td>27.7</td>
<td>37.0</td>
<td></td>
</tr>
<tr>
<td>Ours + ELMo w augmentation</td>
<td>51.5</td>
<td>33.0</td>
<td>40.2</td>
<td></td>
</tr>
<tr>
<td>+ filter &amp; relabel</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BERT-Base, Uncased</td>
<td>51.6</td>
<td>32.8</td>
<td><strong>40.1</strong></td>
<td></td>
</tr>
<tr>
<td>Choi et al. (2018) w augmentation</td>
<td>48.1</td>
<td>23.2</td>
<td>31.3</td>
<td>67.4</td>
</tr>
</tbody>
</table>

5.1.1 Comparing Denoising Models

We now explicitly compare our denoising approach to several baselines. For each denoising method, we create the denoised EL, HEAD, and EL & HEAD dataset and investigate performance on these datasets. Any denoised dataset is combined with the 2K manually-annotated examples and used to train the final model.

<table>
<thead>
<tr>
<th>Model</th>
<th>P</th>
<th>R</th>
<th>F1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ours + GloVe w/o augmentation</td>
<td>47.6</td>
<td>23.3</td>
<td>31.3</td>
</tr>
<tr>
<td>Ours + ELMo w/o augmentation</td>
<td><strong>55.8</strong></td>
<td>27.7</td>
<td>37.0</td>
</tr>
<tr>
<td>Ours + ELMo w augmentation</td>
<td>51.5</td>
<td>33.0</td>
<td><strong>40.2</strong></td>
</tr>
<tr>
<td>+ filter &amp; relabel</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BERT-Base, Uncased</td>
<td>51.6</td>
<td>33.0</td>
<td><strong>40.2</strong></td>
</tr>
<tr>
<td>Choi et al. (2018) w augmentation</td>
<td>47.1</td>
<td>24.2</td>
<td>32.0</td>
</tr>
</tbody>
</table>

Results

Table 3 compares the results on the development set. We report the performance on each of the EL & HEAD, EL, and HEAD dataset. On top of the baseline ORIGINAL, adding synonyms and hypernyms by consulting external knowledge does not improve the performance. Expanding labels with the PAIR technique results in small gains over ORIGINAL. OVERLAP is the most ef-
Table 3: Macro-averaged P/R/F1 on the dev set for the entity typing task of Choi et al. (2018) with various types of augmentation added. The customized loss from Choi et al. (2018) actually causes a decrease in performance from adding any of the datasets. Heuristics can improve incorporation of this data: a relabeling heuristic (Pair) helps on HEAD and a filtering heuristic (Overlap) is helpful in both settings. However, our trainable filtering and relabeling models outperform both of these techniques.

Table 4: Test results on OntoNotes. Denoising helps substantially even in this reduced setting. Using fewer distant examples, we nearly match the performance using the data from Choi et al. (2018) (see text).

5.2 OntoNotes Results

We compare our different augmentation schemes for deriving data for the OntoNotes standard as well. Table 4 lists the results on the OntoNotes test set following the adaptation setting of Choi et al. (2018). Even on this dataset, denoising significantly improves over naive incorporation of distant data, showing that the denoising approach is not just learning quirks of the ultra-fine dataset. Our augmented set is constructed from 2M seed examples while Choi et al. (2018) have a more complex procedure for deriving augmented data from 25M examples. Ours (total size of 2.1M) is on par with their larger data (total size of 3.4M), despite having 40% fewer examples. In this setting, BERT still performs well but not as well as our model with augmented training data.

One source of our improvements from data augmentation comes from additional data that is able to be used because some OntoNotes type can be derived. This is due to denoising doing a better job of providing correct general types. In the EL setting, this yields 730k usable examples out of 1M (vs 540K for no denoising), and in HEAD, 640K out of 1M (vs. 73K).

5.3 Analysis of Denoised Labels

To understand what our denoising approach does to the distant data, we analyze the behavior of our filtering and relabeling functions. Table 5 reports the average numbers of types added/deleted by the relabeling function and the ratio of examples discarded by the filtering function.

Overall, the relabeling function tends to add more and delete fewer number of types. The HEAD examples have more general types added than the EL examples since the noisy HEAD labels are typically finer. Fine-grained types are added to both EL and HEAD examples less frequently. Ultra-fine examples are frequently added to both datasets, with more added to EL; the noisy EL labels are mostly extracted from Wikipedia defini-
6 Related Work

Past work on denoising data for entity typing has used multi-instance multi-label learning (Yaghoobzadeh and Schütze, 2015, 2017; Murty et al., 2018). One view of these approaches is that they delete noisily-introduced labels, but they cannot add them, or filter bad examples. Other work focuses on learning type embeddings (Yogatama et al., 2015; Ren et al., 2016a,b); our approach goes beyond this in treating the label set in a structured way. The label set of Choi et al. (2018) is distinct in not being explicitly hierarchical, making past hierarchical approaches difficult to apply.

Denoising techniques for distant supervision have been applied extensively to relation extraction. Here, multi-instance learning and probabilistic graphical modeling approaches have been used (Riedel et al., 2010; Hoffmann et al., 2011; Surdeanu et al., 2012; Takamatsu et al., 2012) as well as deep models (Lin et al., 2016; Feng et al., 2017; Luo et al., 2017; Lei et al., 2018; Han et al., 2018), though these often focus on incorporating signals from other sources as opposed to manually labeled data.

7 Conclusion

In this work, we investigated the problem of denoising distant data for entity typing tasks. We trained a filtering function that discards examples from the distantly labeled data that are wholly unusable and a relabeling function that repairs noisy labels for the retained examples. When distant data is processed with our best denoising model, our final trained model achieves state-of-the-art performance on an ultra-fine entity typing task.

Acknowledgments

This work was partially supported by NSF Grant IIS-1814522, NSF Grant SHF-1762299, a Bloomberg Data Science Grant, and an equipment grant from NVIDIA. The authors acknowledge the Texas Advanced Computing Center (TACC) at The University of Texas at Austin for providing HPC resources used to conduct this research. Results presented in this paper were obtained using the Chameleon testbed supported by the National Science Foundation. Thanks as well to the anonymous reviewers for their thoughtful comments, members of the UT TAUR lab and Pengxiang Cheng for helpful discussion, and Eunsol Choi for providing the full datasets and useful resources.

Table 5: The average number of types added or deleted by the relabeling function per example. The right-most column shows that the rate of examples discarded by the filtering function.

```
<table>
<thead>
<tr>
<th>Data</th>
<th>General</th>
<th>Fine</th>
<th>Ultra-Fine</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Add</td>
<td>Del</td>
<td>Add</td>
</tr>
<tr>
<td>EL</td>
<td>0.87</td>
<td>0.01</td>
<td>0.36</td>
</tr>
<tr>
<td>HEAD</td>
<td>1.18</td>
<td>0.00</td>
<td>0.51</td>
</tr>
</tbody>
</table>
```

Figure 4: Examples of the noisy labels (left) and the denoised labels (right) for mentions (bold). The colors correspond to type classes: general (purple), fine-grained (green), and ultra-fine (yellow).
References


Mike Mintz, Steven Bills, Rion Snow, and Daniel Jurafsky. 2009. 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.


Xiang Ren, Wenqi He, Meng Qu, Clare R. Voss, Heng Ji, and Jiawei Han. 2016b. Label Noise Reduction in Entity Typing by Heterogeneous Partial-Label Embedding. In Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining.

Sebastian Riedel, Limin Yao, and Andrew McCallum. 2010. Modeling Relations and Their Mentions without Labeled Text. In Machine Learning and Knowledge Discovery in Databases.


