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-

(a) According to the review aggregator Rotten Tomatoes , 89 % of critics gave [**the film**] positive reviews.
film, movie, show, art, entertainment, creation

(b) [**The film**] is based on a hit London and New York play , which was based on a best-selling book.
film, movie, show, art, entertainment, creation

(c) "A pretty good day all round," said [**Gascoyne** , a British veteran of stints with the original Tyrrell team] in a roller-coaster F1 career.
region
person ✔

(d) Djokovic lost to [**Rafael Nadal**] on Monday, in a rain-delayed U.S. Open final.
player, tennis player, champion, achiever, winner, contestant, person, athlete

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.

tity typing scenario and use the same two distant supervision sources as them, based on entity linking and head words. On top of an adapted model from Choi et al. (2018) incorporating ELMo (Peters et al., 2018), naïvely adding distant data actually hurts performance. However, when our learned denoising model is applied to the data, performance improves, and it improves more than heuristic denoising approaches tailored to this dataset. Our strongest denoising model gives a gain of 3 F_1 absolute over the ELMo baseline, and a 4.4 F_1 improvement over naive incorporation of distant data. This establishes a new state-of-the-art on the test set, outperforming concurrently published work (Xiong et al., 2019) and matching the performance of a BERT model (Devlin et al., 2018) on this task. Finally, we show that denoising helps even when the label set is projected onto the OntoNotes label set (Hovy et al., 2006; Gillick et al., 2014), outperforming the method of Choi et al. (2018) in that setting as well.

2 Setup

We consider the task of predicting a structured target y associated with an input x. Suppose we have high-quality labeled data of n (input, target) pairs $\mathcal{D} = \{(x^{(1)}, y^{(1)}), \ldots, (x^{(n)}, y^{(n)})\}$, and noisily labeled data of n' (input, target) pairs $\mathcal{D}' = \{(x^{(1)}, y^{(1)}_{\text{noisy}}), \ldots, (x^{(n')}, y^{(n')}_{\text{noisy}})\}$. For our tasks, $\mathcal{D}$ is collected through manual annotation and $\mathcal{D}'$ is collected by distant supervision. We use two models to denoise data from $\mathcal{D}'$: a filtering function f disposes of unusable data (e.g., mislabeled examples) and a relabeling function g transforms the noisy target labels y_{noisy} to look more like true labels. This transformation improves the noisy data so that we can use it to $\mathcal{D}$ without introducing damaging amounts of noise. In the second stage, a classification model is trained on the augmented data ($\mathcal{D}$ combined with denoised $\mathcal{D}'$) and predicts y given x in the inference phase.

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

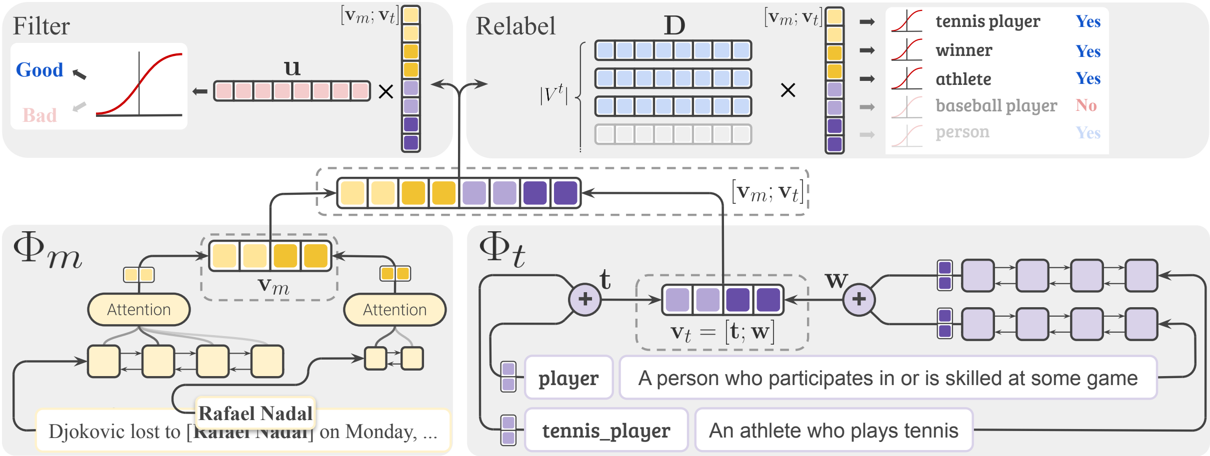

Figure 2: Denoising models. The *Filter* model predicts whether the example should be kept at all; if it is kept, the *Relabel* model attempts to automatically expand the label set. Φ_m is a mention encoder, which can be a state-of-the-art entity typing model. Φ_t encodes noisy types from distant supervision.

completely incorrect types, and (2) systematically miss certain types.

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 $\mathbf{v}_m$. 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 $\mathbf{v}_t = \Phi_t(t)$. 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 $\{\mathbf{t}_1, \ldots, \mathbf{t}_m\}$. The final embedding of the type set $\mathbf{t} = \sum_j \mathbf{t}_j$.

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.[1]

Let w_i^j denote the ith word of the jth type's most common WordNet definition. Each w_i^j is embedded using GloVe (Pennington et al., 2014). The resulting word embedding vectors $\mathbf{w}_i^j$ 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 $\mathbf{w}^j$. The final representation of the definitions is the sum over these vectors for each type: $\mathbf{w} =$

[1] 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.

$\sum_k \mathbf{w}^k$.

Our final $\mathbf{v}_t = [\mathbf{t}; \mathbf{w}]$, 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 $\mathbf{v}_m$ and $\mathbf{v}_t$ are computed using the mention and type encoders. The model prediction is defined as $P(\text{error}) = \sigma\left(\mathbf{u}^\top \text{Highway}\left([\mathbf{v}_m; \mathbf{v}_t]\right)\right)$, where σ is a sigmoid function, $\mathbf{u}$ is a parameter vector, and Highway$(\cdot)$ is a 1-layer highway network (Srivastava et al., 2015). We can apply f to each distant pair in our distant dataset $\mathcal{D}'$ 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 $\mathcal{D}_{\text{error}}$ for f based on the manually labeled data $\mathcal{D}$. For 30% of the examples in $\mathcal{D}$, 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 *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 $\mathcal{D}$ until we find t'_{error} that does not have any common types with t. We then create a positive training example $((s, m, t'_{\text{error}}), z = 1)$. We create a negative training example $((s, m, t), z = 0)$ using the remaining 70% of examples. f is trained on $\mathcal{D}_{\text{error}}$ 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 $\mathbf{v}_m$ and $\mathbf{v}_t$ by the same procedure as the filtering function f. The decoder is a linear layer with parameters $\mathbf{D} \in \mathbb{R}^{|V^{\text{t}}| \times (d_m + d_t)}$. We compute $\mathbf{e} = \sigma\left(\mathbf{D}\left[\mathbf{v}_m; \mathbf{v}_t\right]\right)$, where σ 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 $(s, m, t) \in \mathcal{D}'$ and replace t by $\tilde{t}$ to create the denoised data $\mathcal{D}'_{\text{denoise}} = \{(s, m, \tilde{t}), \dots\}$. For the final prediction, we choose all types t_ℓ where $e_\ell > 0.5$, 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 $\mathcal{D}_{\text{drop}}$ generated from the manually-labeled data $\mathcal{D}$. To mimic the type distribution of the distantly-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 $\mathcal{D}$ and create a noised training set $\mathcal{D}_{\text{drop}}$, where a single training example is $((s, m, t'), t)$. g is trained on $\mathcal{D}'_{\text{drop}}$ 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 $\mathbf{v}$ as well as t.

4 Typing Model

In this section, we define the sentence and mention encoder Φ_m, which is use 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 Φ_m that returns a vector $\mathbf{v}_m \in \mathbb{R}^{d_\Phi}$, 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 Φ_m consists of four vectors: a sentence representation $\mathbf{s}$, a word-level mention representation $\mathbf{m}^{\text{word}}$, a character-level mention representation $\mathbf{m}^{\text{char}}$, and a head-word mention vector $\mathbf{m}^{\text{head}}$. 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

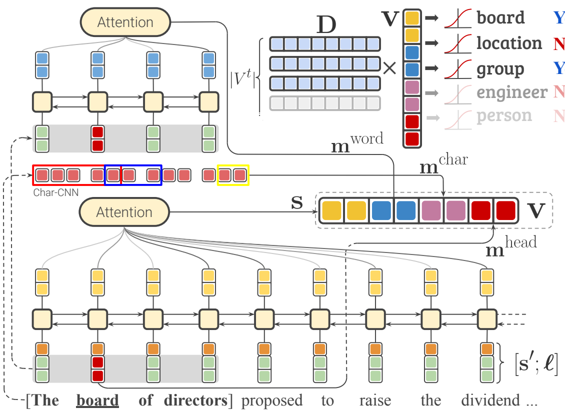

Figure 3: Sentence and mention encoder used to predict types. We compute attention over LSTM encodings of the sentence and mention, as well as using character-level and head-word representations to capture additional mention properties. These combine to form an encoding which is used to predict types.

using ELMo; let $\mathbf{s}'_i \in \mathbb{R}^{d_{ELMo}}$ denote the embedding of the ith word. As suggested in Peters et al. (2018), we learn task specific parameters $\gamma^{\text{task}} \in \mathbb{R}$ and $\mathbf{s}^{\text{task}} \in \mathbb{R}^3$ governing these embeddings. We do not fine-tune the parameters of the ELMo LSTMs themselves.

Sentence Encoder Following Choi et al. (2018), we concatenate the mth word vector $\mathbf{s}_m$ in the sentence with a corresponding location embedding $\boldsymbol{\ell}_m \in \mathbb{R}^{d_{\text{loc}}}$. Each word is assigned one of four location tokens, based on whether (1) the word is in the left context, (2) the word is the first word of the mention span, (3) the word is in the mention span (but not first), and (4) the word is in the right context. The input vectors $[\mathbf{s}'; \boldsymbol{\ell}]$ are fed into a bi-LSTM encoder, with hidden dimension is d_{hid}, followed by a span attention layer (Lee et al., 2017; Choi et al., 2018): $\mathbf{s} = \text{Attention}(\text{bi-LSTM}([\mathbf{s}'; \mathbf{l}]))$, where $\mathbf{s}$ is the final representation of the sentence s.

Mention Encoder To obtain a mention representation, we use both word and character information. For the word-level representation, the mention's contextualized word vectors $\mathbf{m}'$ 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: $\mathbf{m}^{\text{word}} = \text{Attention}(\text{bi-LSTM}(\mathbf{m}'))$.

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 $\mathbf{m}^{\text{char}}$.

Finally, we take the contextualized word vector of the headword $\mathbf{m}^{\text{head}}$ 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, $\mathbf{v} = \left[\mathbf{s}; \mathbf{m}^{\text{word}}; \mathbf{m}^{\text{char}}; \mathbf{m}^{\text{head}}\right] \in \mathbb{R}^{d_\Phi}$.

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^t. Similar to the decoder of the relabeling function g, we compute $\mathbf{e} = \sigma\left(\mathbf{E}\mathbf{v}\right)$, where $\mathbf{E} \in \mathbb{R}^{|V^{\text{t}}| \times d_\Phi}$ and $\mathbf{e} \in \mathbb{R}^{|V^{\text{t}}|}$. For the final prediction, we choose all types t_ℓ where $e_\ell > 0.5$. If none of e_ℓ is greater than 0.5, we choose $t_\ell = \arg\max \mathbf{e}$ (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:

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

where $\mathcal{L}_{\dots}$ is a loss function for a specific type class: general, fine-grained, or ultra-fine, and $\mathbb{1}_{\dots}(t)$ is an indicator function that is active when one of the types t is in the type class. Each $\mathcal{L}_{\dots}$ 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 hyperparameters in our model largely follows Choi et al. (2018) and recommendations for using the pretrained ELMo-Small model.[2] The word embedding size d_{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[3] 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, $\mathcal{D}_{\text{error}}$ and $\mathcal{D}_{\text{drop}}$, 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 $\mathcal{D}'$. Our augmented training data is a combination of the manually-annotated data $\mathcal{D}$ and $\mathcal{D}'_{\text{denoised}}$.

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 $\mathcal{D}'$, using the same label mapping from ultra-fine types to OntoNotes types described in Choi et al. (2018).

[2] https://allennlp.org/elmo

[3] The code for experiments is available at https://github.com/yasumasaonoe/DenoiseET

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.[4] 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, uncased model[5] 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 de-

[4] We investigated several approaches, including taking the head word piece from the last layer and using that for classification (more closely analogous to what Devlin et al. (2018) did for NER), but found this one to work best.

[5] https://github.com/google-research/bert

Model	Total			General			Fine			Ultra-Fine		
	P	R	F1	P	R	F1	P	R	F1	P	R	F1
Ours + GloVe w/o augmentation	46.4	23.3	31.0	57.7	65.5	61.4	41.3	31.3	35.6	42.4	9.2	15.1
Ours + ELMo w/o augmentation	**55.6**	28.1	37.3	**69.3**	77.3	73.0	**47.9**	35.4	40.7	**48.9**	12.6	20.0
Ours + ELMo w augmentation	55.2	26.4	35.7	69.4	72.0	70.7	46.6	38.5	42.2	48.7	10.3	17.1
Ours + ELMo w augmentation + filter & relabel	50.7	**33.1**	**40.1**	66.9	**80.7**	73.2	41.7	46.2	43.8	45.6	**17.4**	**25.2**
BERT-Base, Uncased	51.6	32.8	**40.1**	67.4	80.6	**73.4**	41.6	**54.7**	**47.3**	46.3	15.6	23.4
Choi et al. (2018) w augmentation	48.1	23.2	31.3	60.3	61.6	61.0	40.4	38.4	39.4	42.8	8.8	14.6

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 allow us to effectively incorporate distant data, matching the results of a BERT model on this task (Devlin et al., 2018).

noising as well as prediction in our setting, and we believe this would stack with our approach.

Usage of Pretrained Representations Our model with ELMo trained on denoised data matches the performance of the BERT model. We experimented with incorporating distant data (raw and denoised) in BERT, but the fragility of BERT made it hard to incorporate: training for longer generally caused performance to go down after a while, so the model cannot exploit large external data as effectively. Devlin et al. (2018) prescribe training with a small batch size and very specific step sizes, and we found the model very sensitive to these hyperparameters, with only 2e-05 giving strong results. The ELMo paradigm of incorporating these as features is much more flexible and modular in this setting. Finally, we note that our approach could use BERT for denoising as well, but this did not work better than our current approach. Adapting BERT to leverage distant data effectively is left for future work.

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.

Heuristic Baselines These heuristics target the same factors as our filtering and relabeling functions in a non-learned way.

SYNONYMS AND HYPERNYMS For each type observed in the distant data, we add its synonyms and hypernyms using WordNet (Miller, 1995). This is motivated by the data construction process in Choi et al. (2018).

Model	P	R	F1
Ours + GloVe w/o augmentation	47.6	23.3	31.3
Ours + ELMo w/o augmentation	**55.8**	27.7	37.0
Ours + ELMo w augmentation	55.5	26.3	35.7
Ours + ELMo w augmentation + filter & relabel	51.5	**33.0**	**40.2**
BERT-Base, Uncased	51.6	**33.0**	**40.2**
Choi et al. (2018) w augmentation	47.1	24.2	32.0
LABELGCN (Xiong et al., 2019)	50.3	29.2	36.9

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.

COMMON TYPE PAIRS We use type pair statistics in the manually labeled training data. For each base type that we observe in a distant example, we add any type which is seen more than 90% of the time the base type occurs. For instance, the type `art` is given at least 90% of the times the `film` type is present, so we automatically add `art` whenever `film` is observed.

OVERLAP We train a model on the manually-labeled data only, then run it on the distantly-labeled data. If there is an intersection between the noisy types t and the predicted type $\hat{t}$, we combine them and use as the expanded type $\tilde{t}$. Inspired by tri-training (Zhou and Li, 2005), this approach adds "obvious" types but avoids doing so in cases where the model has likely made an error.

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-

Type	Denoising Method	EL & HEAD			EL			HEAD		
		P	R	F1	P	R	F1	P	R	F1
Heuristic Baselines	RAW DATA	**55.2**	26.4	35.7	52.3	26.1	34.8	**52.8**	28.4	36.9
	SYNONYMS & HYPERNYMS	43.0	30.0	35.3	47.5	26.3	33.9	44.8	31.7	37.1
	PAIR	50.2	29.0	36.8	49.6	27.0	35.0	50.6	31.2	38.6
	OVERLAP	50.0	32.3	39.2	49.5	**30.8**	38.0	50.6	31.4	38.7
Proposed Approach	FILTER	53.1	28.2	36.8	51.9	26.5	35.1	51.2	31.2	38.7
	RELABEL	52.1	32.2	39.8	50.2	31.4	38.6	50.2	31.8	38.9
	FILTER & RELABEL	50.7	**33.1**	**40.1**	**52.7**	30.5	**38.7**	50.7	**32.1**	**39.3**
	Choi et al. (2018)	48.1	23.2	31.3	50.3	19.6	28.2	48.4	22.3	30.6

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.

fective heuristic technique. This simple filtering and expansion heuristic improves recall on EL. FILTER, our model-based example selector, gives similar improvements to PAIR and OVERLAP on the HEAD setting, where filtering noisy data appears to be somewhat important.[6] RELABEL and OVERLAP both improve performance on both EL and HEAD while other methods do poorly on EL. Combining the two model-based denoising techniques, FILTER & RELABEL outperforms all the baselines.

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

Model	Acc.	Ma-F1	Mi-F1
Ours + ELMo w/o augmentation	42.7	72.7	66.7
Ours + ELMo w augmentation	59.3	76.5	70.7
Ours + ELMo w augmentation + filter & relabel	63.9	**84.5**	78.9
Ours + ELMo w augmentation by Choi et al. (2018)	**64.9**	**84.5**	**79.2**
BERT-Base, Uncased	51.8	76.6	69.1
Shimaoka et al. (2017)	51.7	70.9	64.9
AFET (Ren et al., 2016a)	55.1	71.1	64.7
PLE (Ren et al., 2016b)	57.2	71.5	66.1
Choi et al. (2018)	59.5	76.8	71.8
LABELGCN (Xiong et al., 2019)	59.6	77.8	72.2

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).

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] One possible reason for this is identifying stray word senses; *film* can refer to the physical photosensitive object, among other things.

	General		Fine		Ultra-Fine		
Data	Add	Del	Add	Del	Add	Del	Filter (%)
EL	0.87	0.01	0.36	0.17	2.03	0.12	9.4
HEAD	1.18	0.00	0.51	0.01	1.15	0.16	10.0

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.

tions, so those labels often do not include ultra-fine types. The filtering function discards similar numbers of examples for the EL and HEAD data: 9.4% and 10% respectively.

Figure 4 shows examples of the original noisy labels and the denoised labels produced by the relabeling function. In example (a), taken from the EL data, the original labels, {location, city}, are correct, but human annotators might choose more types for the mention span, **Minneapolis**. The relabeling function retains the original types about the geography and adds ultra-fine types about administrative units such as {township, municipality}. In example (b), from the HEAD data, the original label, {dollar}, is not so expressive by itself since it is a name of a currency. The labeling function adds coarse types, {object, currency}, as well as specific types such as {medium of exchange, monetary unit}. In another EL example (c), the relabeling function tries to add coarse and fine types but struggles to assign multiple diverse *ultra-fine* types to the mention span **Michelangelo**, possibly because some of these types rarely cooccur (painter and poet).

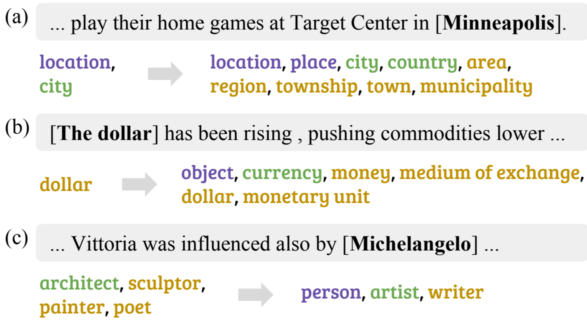

Figure 4: Examples of the noisy labels (left) and the denoised labels (right) for mentions (bold). The colors correspond to type classes: *general* (purple), *fine-grained* (green), and *ultra-fine* (yellow).

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.

References

Eunsol Choi, Omer Levy, Yejin Choi, and Luke Zettlemoyer. 2018. Ultra-Fine Entity Typing. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics*.

Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel P. Kuksa. 2011. Natural Language Processing (Almost) from Scratch. *Journal of Machine Learning Research*, 12:2493–2537.

Luciano Del Corro, Abdalghani Abujabal, Rainer Gemulla, and Gerhard Weikum. 2015. FINET: Context-Aware Fine-Grained Named Entity Typing. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *CoRR*, abs/1810.04805.

Xiaocheng Feng, Jiang Guo, Bing Qin, Ting Liu, and Yongjie Liu. 2017. Effective Deep Memory Networks for Distant Supervised Relation Extraction. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI-17*.

Dan Gillick, Nevena Lazic, Kuzman Ganchev, Jesse Kirchner, and David Huynh. 2014. Context-Dependent Fine-Grained Entity Type Tagging. *CoRR*, abs/1412.1820.

Alex Graves and Jürgen Schmidhuber. 2005. Framewise phoneme classification with bidirectional LSTM and other neural network architectures. *Neural Networks*, 18(5-6):602–610.

Xu Han, Zhiyuan Liu, and Maosong Sun. 2018. Denoising Distant Supervision for Relation Extraction via Instance-Level Adversarial Training. *CoRR*, abs/1805.10959.

Braden Hancock, Paroma Varma, Stephanie Wang, Martin Bringmann, Percy Liang, and Christopher Ré. 2018. Training Classifiers with Natural Language Explanations. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics*.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep Residual Learning for Image Recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR*.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation*, 9(8):1735–1780.

Raphael Hoffmann, Congle Zhang, Xiao Ling, Luke S. Zettlemoyer, and Daniel S. Weld. 2011. Knowledge-Based Weak Supervision for Information Extraction of Overlapping Relations. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*.

Matthew Honnibal and Mark Johnson. 2015. An Improved Non-monotonic Transition System for Dependency Parsing. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*.

Eduard Hovy, Mitchell Marcus, Martha Palmer, Lance Ramshaw, and Ralph Weischedel. 2006. OntoNotes: The 90% Solution. In *Proceedings of the 2006 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*.

Diederik P. Kingma and Jimmy Ba. 2014. Adam: A Method for Stochastic Optimization. *CoRR*, abs/1412.6980.

Kenton Lee, Luheng He, Mike Lewis, and Luke Zettlemoyer. 2017. End-to-end Neural Coreference Resolution. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*.

Kai Lei, Daoyuan Chen, Yaliang Li, Nan Du, Min Yang, Wei Fan, and Ying Shen. 2018. Cooperative Denoising for Distantly Supervised Relation Extraction. In *Proceedings of the 27th International Conference on Computational Linguistics*.

Yankai Lin, Shiqi Shen, Zhiyuan Liu, Huanbo Luan, and Maosong Sun. 2016. Neural Relation Extraction with Selective Attention over Instances. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*.

Bingfeng Luo, Yansong Feng, Zheng Wang, Zhanxing Zhu, Songfang Huang, Rui Yan, and Dongyan Zhao. 2017. Learning with Noise: Enhance Distantly Supervised Relation Extraction with Dynamic Transition Matrix. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics*.

George A. Miller. 1995. WordNet: A Lexical Database for English. *Commun. ACM*, 38(11):39–41.

David Milne and Ian H. Witten. 2008. Learning to Link with Wikipedia. In *Proceedings of the 17th ACM Conference on Information and Knowledge Management*.

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*.

Shikhar Murty, Patrick Verga, Luke Vilnis, Irena Radovanovic, and Andrew McCallum. 2018. Hierarchical Losses and New Resources for Fine-grained Entity Typing and Linking. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics*.

Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. Glove: Global Vectors for Word Representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing*.

Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep Contextualized Word Representations. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*.

Maxim Rabinovich and Dan Klein. 2017. Fine-Grained Entity Typing with High-Multiplicity Assignments. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics*.

Alexander Ratner, Stephen H. Bach, Henry Ehrenberg, Jason Fries, Sen Wu, and Christopher Ré. 2018a. Snorkel: Rapid Training Data Creation with Weak Supervision. In *Proceedings of VLDB*.

Alexander Ratner, Braden Hancock, Jared Dunnmon, Frederic Sala, Shreyash Pandey, and Christopher Ré. 2018b. Training Complex Models with Multi-Task Weak Supervision. *CoRR*, abs/1810.02840.

Alexander Ratner, Chris De Sa, Sen Wu, Daniel Selsam, , and Christopher Ré. 2016. Data Programming: Creating Large Training Sets, Quickly. In *Proceedings of NeurIPS*.

Xiang Ren, Wenqi He, Meng Qu, Lifu Huang, Heng Ji, and Jiawei Han. 2016a. AFET: Automatic Fine-Grained Entity Typing by Hierarchical Partial-Label Embedding. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*.

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*.

Sonse Shimaoka, Pontus Stenetorp, Kentaro Inui, and Sebastian Riedel. 2017. Neural Architectures for Fine-grained Entity Type Classification. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics*.

Rupesh Kumar Srivastava, Klaus Greff, and Jürgen Schmidhuber. 2015. Highway Networks. *CoRR*, abs/1505.00387.

Mihai Surdeanu, Julie Tibshirani, Ramesh Nallapati, and Christopher D. Manning. 2012. Multi-instance Multi-label Learning for Relation Extraction. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*.

Shingo Takamatsu, Issei Sato, and Hiroshi Nakagawa. 2012. Reducing Wrong Labels in Distant Supervision for Relation Extraction. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics*.

Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. 2008. Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th International Conference on Machine Learning*.

Hai Wang and Hoifung Poon. 2018. Deep Probabilistic Logic: A Unifying Framework for Indirect Supervision. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*.

Wenhan Xiong, Jiawei Wu, Deren Lei, Mo Yu, Shiyu Chang, Xiaoxiao Guo, and William Yang Wang. 2019. Imposing Label-Relational Inductive Bias for Extremely Fine-Grained Entity Typing. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*.

Yadollah Yaghoobzadeh and Hinrich Schütze. 2015. Corpus-level Fine-grained Entity Typing Using Contextual Information. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*.

Yadollah Yaghoobzadeh and Hinrich Schütze. 2017. Multi-level Representations for Fine-Grained Typing of Knowledge Base Entities. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics*.

Dani Yogatama, Daniel Gillick, and Nevena Lazic. 2015. Embedding Methods for Fine Grained Entity Type Classification. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing*.

Zhi-Hua Zhou and Ming Li. 2005. Tri-Training: Exploiting Unlabeled Data Using Three Classifiers. *IEEE Trans. Knowl. Data Eng.*, 17(11):1529–1541.