Learn and Review: Enhancing Continual Named Entity Recognition via Reviewing Synthetic Samples

Yu Xia[1,*], Quan Wang[2], Yajuan Lyu[2], Yong Zhu[2], Wenhao Wu[1], Sujian Li[1,†], Dai Dai[2]
[1]Key Laboratory of Computational Linguistics, Peking University, MOE, China
[2]Baidu Inc., Beijing, China
{yuxia, waynewu, lisujian}@pku.edu.cn
{wangquan05, lvyajuan, zhuyong, daidai}@baidu.com

Abstract

Traditional methods for named entity recognition (NER) classify mentions into a fixed set of pre-defined entity types. However, in many real-world scenarios, new entity types are incrementally involved. To investigate this problem, continual learning is introduced for NER. However, the existing method depends on the relevance between tasks and is prone to inter-type confusion. In this paper, we propose a novel two-stage framework Learn-and-Review (L&R) for continual NER under the type-incremental setting to alleviate the above issues. Specifically, for the learning stage, we distill the old knowledge from teacher to a student on the current dataset. For the reviewing stage, we first generate synthetic samples of old types to augment the dataset. Then, we further distill new knowledge from the above student and old knowledge from the teacher to get an enhanced student on the augmented dataset. This stage has the following advantages: (1) The synthetic samples mitigate the gap between the old and new task and thus enhance the further distillation; (2) Different types of entities are jointly seen during training which alleviates the inter-type confusion. Experimental results show that L&R outperforms the state-of-the-art method on CoNLL-03 and OntoNotes-5.0.

1 Introduction

Traditional Named Entity Recognition (NER) aims at extracting mentions from a given text and classifying them into a fixed set of pre-defined entity types such as *Person, Location, Organization, etc* (Ma and Hovy, 2016). However, in many real-world scenarios, new entity types emerge periodically by demand and the models are required to recognize new types of entities without forgetting the old ones, which can formulate into the paradigm of *continual learning* (a.k.a. *lifelong learning* or *incremental learning*) (Thrun, 1998; Parisi et al., 2019). For example, voice assistants such as Siri are often expected to grasp new intents (*e.g. GetMovie*) and thus new entity types (*e.g. Actor, Genre*) are continually involved. The ability to learn from continuous streams of data after deployment is important for modern NER models in specific scenarios.

However, continual learning, as it has long been recognized, suffers severely from *catastrophic forgetting*, *i.e.*, the loss or disruption of previously learned knowledge when new patterns are learned (McCloskey and Cohen, 1989; Robins, 1995; Goodfellow et al., 2013; Kirkpatrick et al., 2017). Different from human beings, an NER model (particularly that based on deep neural networks) which stores knowledge by its parameters is vulnerable to catastrophic forgetting of old knowledge while updating parameters to learn new entity types.

In order to avoid forgetting old types of entities while learning the new ones, a naive solution is to annotate a dataset for both old and new types and retrain the model from scratch. However, this method is computational-inefficient and labor-extensive, especially when the number of entity types is large. To reduce the cost, Monaikul et al. (2021) advocate annotating a training set only for new entity types and retaining previously learned knowledge via *knowledge distillation* (KD) (Hinton et al., 2015). In their approach, the current NER model acts as the teacher and the target new NER model the student. The student then learns new entity types by using the new training material and retains knowledge of old entities by imitating the teacher's output on this new training set. Despite the initial success, this KD-based approach relies on the co-occurrence of unlabeled old types in the current training data of new types. If the new training set (*e.g.* annotated only for *Restaurant*) contains little information related to the old entity types (*e.g. Sport*), the knowledge of these old types will be hard to be retained

[*] This work was done during internship at Baidu Inc.
[†] Corresponding author.

Findings of the Association for Computational Linguistics: ACL 2022, pages 2291 - 2300

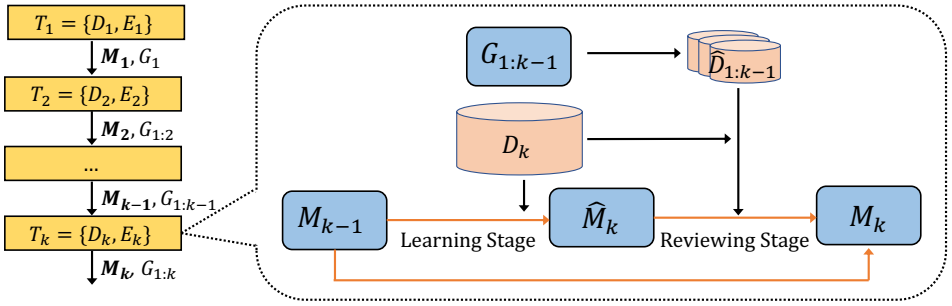

Figure 1: An overview of L&R. At the k-th step, with the new training data D_k and the old models $M_{k-1}, G_{1:k-1}$ available. We firstly distill the teacher model M_{k-1} into the student model $\hat{M}_k$ by minimizing $\alpha\mathcal{L}_{\text{CE}} + \beta\mathcal{L}_{\text{KD}}$ on D_k. Then, we use the generators $G_{1:k-1}$ to generate some unlabeled contexts $\hat{D}_{1:k-1}$ which contain old types of entities to augment the current dataset D_k. We further distill $\hat{M}_k$ and M_{k-1} into M_k by minimizing $\alpha\mathcal{L}_{\text{CE}} + \beta\mathcal{L}_{\text{KD}}$ on the augmented dataset $\bigcup_{i=1}^{k-1} \hat{D}_i \cup D_k$.

simply by distillation. Furthermore, the model will also have difficulty discriminating the old and new entity types since they rarely jointly seen. This issue is typically referred to as *inter-type confusion* (Masana et al., 2020).

In this paper, to alleviate the above issues, inspired by the reviewing behavior of human students, we propose Learn-and-Review (L&R), a two-stage framework that introduces a reviewing stage after the common learning stage. To be specific, during the learning stage, we train the student to recognize new types of entities and retain knowledge of old types under the teacher's supervision by knowledge distillation. Then, during the reviewing stage, we first generate synthetic samples containing old types of entities to augment the current training set. With the augmented data obtained, we further distill new knowledge from the above student and old knowledge from the teacher to get an enhanced student. By augmenting the current dataset with the synthetic samples of old types, we mitigate the gap between the old and the new task and thus enhance the further distillation. Moreover, since different types of entities are jointly seen during training, the model will discriminate better between types and thus alleviate the inter-type confusion. Besides, L&R improves the performance at each step and thus mitigates the error propagation caused by the distillation.

We evaluate our proposed framework on CoNLL-03 (Sang and De Meulder, 2003) and OntoNotes-5.0 (Hovy et al., 2006). Experimental results show that L&R outperforms the state-of-the-art method. We also conduct extensive analysis to discuss the effectiveness of the reviewing stage in enhancing the distillation and alleviating inter-type confusion.

Our contributions can be summarized as follows:

- To the best of our knowledge, we are the first to point out the type co-occurrence requirement, which is one particular shortcoming of the existing KD methods for class-incremental learning.
- We propose a novel augmentation strategy in the reviewing stage to reduce the type co-occurrence requirement.
- Extensive experimental results show that our method outperforms the state-of-the-art baseline. We also conduct experiments to explain the reasons of the improvement.

2 Related Work

2.1 Named Entity Recognition

The traditinal NER work focuses on extracting predifined types of entities from text (Lample et al., 2016; Zhang and Yang, 2018; Yan et al., 2021). Yet in many real-world scenarios, new entity types emerge periodically by demand and the models are required to recognize new types of entities without forgetting the old ones. It is inefficient and sometimes practically impossible to re-train a NER model from scratch every time new types added. Hence, some researchers pay their attention to updating the model by the continual learning approaches. (Monaikul et al., 2021) re-constructed the original setting into the type-incremental setting based on several well-known NER datasets in order to study how to continually train the model with the addition of new types. In this paper, we

follow (Monaikul et al., 2021) to study continual NER in a type-incremental setting.

2.2 Class-incremental Learning

In the field of machine learning, most early methods for continual learning considered the task-incremental setting in which a task-ID is available at inference time (Masana et al., 2020). More recently, methods have started addressing the more difficult setting of type/class-incremental learning, where the algorithm does not have access to the task-ID at inference time, and therefore must be able to distinguish between all types/classes from all tasks. Since types are never jointly trained, the network has difficulty discriminating all classes. This problem is referred to as inter-type/task confusion (Masana et al., 2020). To prevent inter-type confusion and learn representations which are optimal to discriminate between all classes, rehearsal based methods are commonly used. These methods keep a small number of exemplars (Rebuffi et al., 2017; Wu et al., 2019) (exemplar rehearsal), or generate synthetic samples (Shin et al., 2017; Sun et al., 2019) or features (Xiang et al., 2019) (pseudo-rehearsal). They prevent the forgetting of previous tasks by replaying the stored or generated data from previous tasks. Inspired by the pseudo rehearsal-based methods, we generate some data containing old types of entities by a language model to augment the current data. However, it is very common for entities introduced in different steps to co-occur in the same context in NER which makes the existing rehearsal approaches fail to be applied. Different from the existing rehearsal methods, we utilize the teacher and the student obtained from the learning stage to provide soft labels (*i.e. output probability*) for the unlabeled synthetic data to mitigate the type co-occurrence problem.

3 Preliminary

3.1 Problem Formulation

We adopt the type-incremental setting for NER as (Monaikul et al., 2021). We train the model on a sequence of tasks $T_1, T_2, ...T_k$, where the k-th task has its own training set D_k only annotated for the new entity types E_k. Suppose that entity types in different tasks are non-overlapping (i.e., $E_i \cap E_j = \emptyset$ if $i \neq j$). Note that the sentences in D_k potentially also contain tokens of types in the past or future step but this information is not annotated. At the k-th incremental step $(k > 1)$, with D_k and the previous model M_{k-1} available, our goal is to get a model M_k which can recognize entities of all seen types $\bigcup_{i=1}^{k} E_i$.

3.2 NER Model

NER models are usually treated as the sequence labeling task which classifies every token in a sequence into a set of entity types or non-entity. The NER model we use consists of an encoder $\mathbf{E}$ and a linear softmax classifier $\mathbf{C}$. Given a sequence of tokens and their labels $\{x_{i=1}^{L}, y_{i=1}^{L}\}$, the encoder $\mathbf{E}$ maps the inputs into the hidden vectors $\{\boldsymbol{h}_{i=1}^{L}\}$. With each $\mathbf{h}_i$ derived, the linear softmax classifier $\mathbf{C}$ maps it into the label space and calculates the probability distribution of its labels:

$$\boldsymbol{z}_i = \boldsymbol{W}\boldsymbol{h}_i + \boldsymbol{b} \tag{1}$$

$$\boldsymbol{P}(x_i; \boldsymbol{\theta}) = \text{softmax}(\boldsymbol{z}_i) = \frac{\exp(\boldsymbol{z}_i)}{\sum_j \exp(\boldsymbol{z}_j)} \tag{2}$$

where $\boldsymbol{P}(x_i; \boldsymbol{\theta}) \in \mathbb{R}^n$ with n being the size of the label space and $\boldsymbol{\theta}$ denotes the learnable model parameters. The size of the label space depends on the tagging scheme used. For example, the BIO format distinguishes begin/inside/outside of named entities under which the label space have a dimensionality of $h \times (2m + 1)$, where h is the size of hidden vector and m is the size of entity types. In the type-incremental setting, the size of the label space incrementally expands in each step. We minimize the cross entropy loss to encourage the model to correctly predict the true labels:

$$\mathcal{L}_{\text{CE}}(\boldsymbol{x}; \boldsymbol{\theta}) = -\sum_{i=1}^{L} \log P_{y_i}(x_i; \boldsymbol{\theta}) \tag{3}$$

where $P_{y_i}(x_i; \boldsymbol{\theta})$ is the model's output probability of token x_i belonging to class y_i.

4 Method

In this section, we first introduce the whole training procedure of our framework which consists of a learning and a reviewing stage. Then, we describe the two stages in detail.

4.1 Training Procedure

The training procedure of our proposed L&R is illustrated in Fig. 1 and detailed in Algorithm 1. Assuming that we are at the k-th incremental step $(k > 1)$, with the new training data D_k and the old models $M_{k-1}, G_{1:k-1}$ at our disposal. L&R includes two stages to learn new types of entities

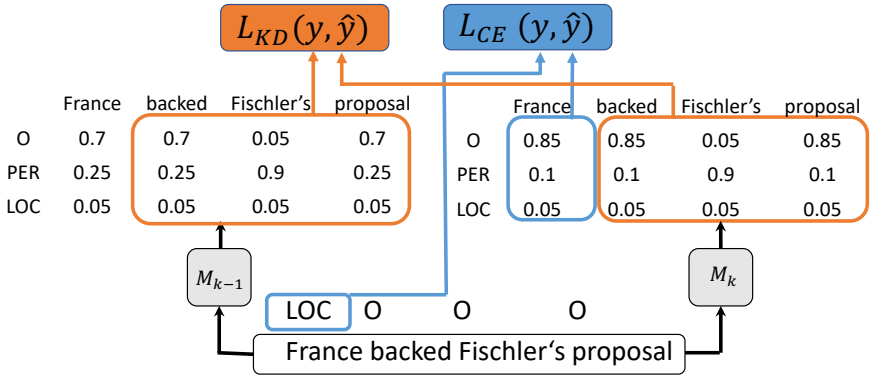

Figure 2: The distillation process. For a sentence with its labels "France backed Fischler's proposal", "LOC O O O" (Note that the gold label for *Fischler's* is *PER* but this information is not annotated at this step). If $y = LOC$, we compute the cross-entropy between the output of M_k and y (blue). Otherwise, we compute the KL divergence between the output of M_{k-1} and M_k (orange).

while avoiding forgetting the old ones: (1) At the learning stage (line 6), we distill old knowledge from the teacher M_{k-1} into the student $\hat{M}_k$ by minimizing the weighted sum of the cross-entropy loss and the knowledge distillation loss on D_k. (2) At the reviewing stage (line $8 \sim 12$), we firstly use the generators $G_{1:k-1}$ to generate some unlabeled contexts $\hat{D}_{1:k-1}$ which contain old types of entities to augment the current dataset D_k. Then, we further distill new knowledge from $\hat{M}_k$ and old knowledge from $_{k-1}$ into M_k by minimizing the above weighted sum on the augmented dataset $\bigcup_{i=1}^{k-1} \hat{D}_i \cup D_k$. Besides, we train G_k by minimizing the language modeling loss on D_k.

4.2 Learning Stage

For the k-th incremental step (k>1), with the training data D_k and the models from the last step $M_{k-1}, G_{1:k-1}$ available, the goal of this stage is to get a model capable of recognizing all previously seen types. Firstly, We initialize the student $\hat{M}_k$ with the parameters of M_{k-1} and expand its linear layer to accommodate the new entity types. To be more specific, suppose we use the BIO tagging schema (introduced in Sec. 3.2), then the original weight matrix with dimension $h \times (2n+1)$ should be expanded to $h \times (2n+2m+1)$, where $n = |\cup_{i=1}^{k} E_i|$ and $m = |E_k|$. After initializing the student, we distill the old knowledge from the teacher M_{k-1} to the student $\hat{M}_{k-1}$. Given that the training dataset D_k is only annotated for E_k, directly training $\hat{M}_{k-1}$ on it will cause catastrophic forgetting. Therefore, we utilize M_{k-1} to provide soft labels (*i.e.* output probability distribution) for old types of entities in D_k. At the same time, the gold annotation for E_k is used to train $\hat{M}_k$ to recognize entities of new types. With all previously seen types of labels obtained, $\hat{M}_k$ is trained on D_k with the weighted sum of the following two losses (Eq. 6): the cross entropy loss (Eq. 3) that penalizes errors of recognizing new entity types and the knowledge distillation loss (Eq. 5) that penalizes forgetting of old entity types.

Formally, for each token with its gold label y, we compute either the cross-entropy loss or the KL divergence for that token according to its label y. When $y \in E_k$, we compute the cross-entropy between the output distribution of $\hat{M}_k$ and y. Otherwise (*e.g.* y is non-entity), we compute the KL divergence between the output distribution of M_{k-1} and $\hat{M}_k$. The process is illustrated in Fig.2.

$$\boldsymbol{P}(x_i; \boldsymbol{\theta}, T) = \frac{\exp(z_i/T)}{\sum_j \exp(z_j/T)} \tag{4}$$

where $\boldsymbol{P}(x_i; \boldsymbol{\theta}, T) \in \mathbb{R}^n$ with n being the size of the model's label space. $\boldsymbol{\theta}$ denotes the learnable model parameters. T denotes the temperature hyper-parameter that can be tuned to obtain a softer distribution (Hinton et al., 2015).

$$\mathcal{L}_{\text{KD}} = -\sum_{i=1}^{L} \sum_{j=1}^{|\cup_{i=1}^{k} E_i|} P_j(x_i; \boldsymbol{\theta}_{k-1}, T) \log P_j(x_i; \hat{\boldsymbol{\theta}}_k, T) \tag{5}$$

where $\boldsymbol{P}(x_i; \boldsymbol{\theta}_{k-1}, T) \in \mathbb{R}^{|\cup_{i=1}^{k-1} E_i|}$ denotes the teacher's output probability and $\boldsymbol{P}(x_i; \hat{\boldsymbol{\theta}}_k, T) \in \mathbb{R}^{|\cup_{i=1}^{k} E_i|}$ denotes the student's. In order to make the teacher's output the same size as the student's, we fill the teacher's outputs of the new labels with a small constant.

$$\mathcal{L} = \alpha \mathcal{L}_{\text{CE}} + \beta \mathcal{L}_{\text{KD}} \tag{6}$$

where α, β denote the weights of the loss.

4.3 Reviewing Stage

In order to mitigate the gap between tasks and alleviate the problem of inter-task confusion, we introduce a novel reviewing stage after the common learning stage. Firstly, for each old task $i \in \{1, 2, ..., k-1\}$, we use the generator G_i to generate some unlabeled contexts related to types E_i. Then, we concatenate the output probability of old types from M_{k-1} and the probability of new types from $\hat{M}_k$ to get the probability of all seen types for the unlabeled contexts according to Eq. 7. We calculate the KL divergence between the above probability on all seen types and the output of M_k on the generated data using Eq. 8. We calculate the cross-entropy loss on the current data according to Eq. 3. Finally, we initialize M_k with $\hat{M}_k$ and train M_k using the above weighted losses Eq. 6. The process is similar to Fig. 2 except that the probability of old types is given by $\hat{M}_{k-1}$ instead of a small constant.

$$\begin{aligned} &\boldsymbol{P}(x_i; \boldsymbol{\theta}_{k-1}, \hat{\boldsymbol{\theta}}_k, T) = \\ &concat([\boldsymbol{P}_{E_{1:k-1}}(x_i; \boldsymbol{\theta}_{k-1}, T); \boldsymbol{P}_{E_k}(x_i; \hat{\boldsymbol{\theta}}_k, T)]) \end{aligned} \tag{7}$$

$$\mathcal{L}_{\text{KD}} = -\sum_{i=1}^{L} \sum_{j=1}^{|\cup_{i=1}^{k} E_i|} P_j(x_i; \boldsymbol{\theta}_{k-1}, \hat{\boldsymbol{\theta}}_k, T) \log P_j(x_i; \boldsymbol{\theta}_k, T) \tag{8}$$

Besides, we train a generator G_k using the unlabeled contexts in D_k by minimizing Eq. 11

Generator The model we use for generating contexts is a one-layer LSTM language model. We train a separate generator for each task and only use it for inference in the later steps. Specifically, given a sequence of L tokens $\{x_{i=1}^{L}\}$, we feed them into an embedding layer and a LSTM layer to get the contextualized representation for each token $\{\boldsymbol{h}_{i=1}^{L}\}$. Then, we use a linear softmax classifier to get the probability of the next token:

$$\boldsymbol{z}_i = \boldsymbol{W}\boldsymbol{h}_i + \boldsymbol{b} \tag{9}$$

$$P(x_i | x_{<i}; \boldsymbol{\theta}) = \frac{\exp(z_{i,index(x_i)})}{\sum_j \exp(z_{i,j})} \tag{10}$$

where $\boldsymbol{z}_i \in \mathbb{R}^V$ with V being the vocabulary size and $index(*)$ denotes the index of x_i in the vocabulary. We train the language model by minimizing the negative log-likelihood in predicting the next word:

$$\mathcal{L}_{\text{LM}}(\boldsymbol{x}; \boldsymbol{\theta}) = \sum_{i=1}^{L} -\log P(x_i | x_{<i}; \boldsymbol{\theta}) \tag{11}$$

For inference, *i.e.* generating synthetic samples, given the *[BOS]* token as the input, the model decodes the sentence autoregressively by sampling on the probability calculated by Eq. 10. By language modeling the contexts of a specific entity type, we extract its common patterns for the student to review and refresh its old knowledge. Owning to the randomness introduced by the sampling process, the generator tends to provide more diverse sentences rather than merely recovering old samples.

Algorithm 1 Procedure of our framework

Require: A stream of incoming tasks $T_1, T_2, \cdots, T_k, \cdots$, where each task T_k is associated with a dataset D_k consisting of sentences annotated only w.r.t. previously unseen entity types E_k.

Ensure: The latest NER model M_k at each step k which can recognize entities of all seen entity types $\cup_{i=1}^{k} E_i$.

train M_1 by minimizing $\mathcal{L}_{\text{CE}}$ on D_1;
train generator G_1 by minimizing $\mathcal{L}_{\text{LM}}$ on D_1;
$k \leftarrow 2$;
while there are still tasks left **do**
    // Learning Stage
    distill M_{k-1} into $\hat{M}_k$ by minimizing $\alpha\mathcal{L}_{\text{CE}} + \beta\mathcal{L}_{\text{KD}}$ on D_k;
    // Reviewing Stage
    for $i = 1$ to $k-1$ **do**
        generate synthetic sentences $\hat{D}_i$ from previous step i by using G_i;
    end for
    distill $M_{k-1}, \hat{M}_k$ into M_k by minimizing $\alpha\mathcal{L}_{\text{CE}} + \beta\mathcal{L}_{\text{KD}}$ on $\bigcup_{i=1}^{k-1} \hat{D}_i \cup D_k$;
    train G_k by minimizing $\mathcal{L}_{\text{LM}}$ on D_k;
    k=k+1;
end while

5 Experiment Setup

5.1 Datasets

To evaluate our framework, we re-construct the original setting into the type-incremental setting based on several well-known NER datasets including CoNLL-03 English (Sang and De Meulder,

	CoNLL-03				OntoNotes-5.0					
	PER	LOC	ORG	MISC	PERSON	GPE	ORG	DATE	CARD	NORP
Train	4373	5127	4587	2698	12195	10643	9537	8921	5788	5297
Dev	1120	1329	962	695	1553	1592	1262	1264	736	686
Test	1025	1266	1229	563	1573	1573	1230	1281	772	671

Table 1: The sentence distribution of each entity type in CoNLL-03 and OntoNotes-5.0.

2003) and OntoNotes-5.0 English (Hovy et al., 2006). For OntoNotes-5.0, we select the following types to ensure enough examples for training: *Organization*, *Person*, *Geo-Political Entity*, *Date*, *Cardinal*, *Nationalities and Religious Political Group*.

5.2 Settings

We adopt the following setup to simulate the real-world data collection. When constructing the training/dev sets for the k-th task, for a sample with L tokens $[x_1, x_2, \ldots, x_L]$ and its corresponding labels $[y_1, y_2, \ldots, y_L]$ in the original training/dev sets, we replace the label y_i with O if $y_i \notin \mathrm{E}_k$ to get $\hat{y}_i$. Then, we add $[x_1, x_2, \ldots, x_L]$ and its modified labels $[\hat{y}_1, \hat{y}_2, \ldots, \hat{y}_L]$ into the training/dev sets of the k-th task if $\exists y_i \in E_k, 1 \leq i \leq L$. When constructing the test sets for the k-th task, we replace the above E_k with $\cup_{i=1}^{k} E_k$ (all seen types up to the current step). Without loss of generality, we consider adding one type at each step. After re-constructing the datasets based on the above rules, the sentence distribution of each entity type across the official training, development, test sets are listed in Table 1.

5.3 Implementation Details

We follow the previous work (Monaikul et al., 2021) for implementation. The details can be found in Appendix A.

5.4 Compared Methods

We compare our framework to *ExtendNER* and select *non-CL complete* as the upper bound. We reimplement them according to (Monaikul et al., 2021). For *non-CL complete*, we train the model from scratch on those samples which contain the entity of all seen types up to the current step.

5.5 Metrics

Following (Monaikul et al., 2021), we compute the precision, recall and F1 scores for each entity type at each step. We report the macro-average F1 score w.r.t. all types seen up to the k-th step, averaged over all sampled permutations:

$$F_{avg}^{k,r} = \frac{1}{k \times r} \sum_{e \in \bigcup_{i=1}^{k} E_i^r} F_e^{k,r} \tag{12}$$

where $\bigcup_{i=1}^{k} E_i^r$ denotes all types seen up to the k-th step in the task order r. F_e^k denotes the F1 score of entity e at the k-th step in the order r.

We also evaluate the model's overall performance regarding order-sensitivity to have a more thorough understanding. The metric we use is Error Bound (Wu et al., 2021) which is defined as:

$$EB = Z_{\frac{\alpha}{2}} \times \frac{\sigma}{\sqrt{n}} \tag{13}$$

where $Z_{\frac{\alpha}{2}}$ is the confidence coefficient of confidence level α, and σ is the standard deviation of average F1 obtained from n different task orders. A model with a lower error bound indicates less order-sensitivity.

6 Results

6.1 Main Results

We conduct extensive experiments on CoNLL-03 and OntoNotes-5.0 and make the following observations:

(1) Table 2 shows that L&R outperforms the baseline among all the steps on the two datasets. For example, L&R achieves 4.01, 6.22, 7.83 average F1 improvement at step 2, 3, 4 on CoNLL-03. Noting that L&R achieves more improvement against ExtendNER on later steps. The reason is that we improve the performance at each step and thus alleviate the error propagation caused by the distillation.

(2) In additional to the above accumulated improvement of L&R, we also report the instant improvement of the reviewing stage at each step in Table 2. For example, L&R gets 4.01, 4.02, 4.11 improvement at step 2, 3, 4 after

Method	CoNLL-03				OntoNotes-5.0					
	Step 1	Step 2	Step 3	Step 4	Step 1	Step 2	Step 3	Step 4	Step 5	Step 6
ExtendNER	92.08	82.93	78.90	77.91	92.06	87.60	83.72	81.41	80.63	79.56
	-	±4.51	±3.82	±1.41	-	±2.12	**±1.54**	±1.70	±1.68	±0.94
L&R	92.08	**86.93**	**85.12**	**85.74**	92.06	**88.09**	**85.69**	**83.79**	**83.38**	**83.02**
	-	**±3.43**	**±2.38**	**±0.44**	-	**±1.82**	±2.02	**±1.13**	**±0.93**	**±0.63**
before reviewing	92.08	82.93	81.10	81.63	92.06	87.60	84.53	82.67	82.31	82.03
non-CL complete	92.08	89.86	88.99	88.90	92.06	91.16	90.50	89.69	89.57	89.30

Table 2: The average F1 over seen entity types on the test set of NER datasets at each step. Scores at each step are averaged over all sampled permutations. Error Bound is indicated after the ± symbol. We set the confidence as 0.95.

	CoNLL-03				OntoNotes-5.0					
	PER	LOC	ORG	MISC	PERSON	GPE	ORG	DATE	CARD	NORP
Before	90.53	85.45	77.89	70.37	89.67	89.86	73.06	76.94	76.94	80.55
After	95.19	90.46	83.30	71.67	90.21	90.32	73.40	76.99	78.26	82.93
Δ	+4.66	+5.00	+5.41	+1.30	+0.54	+0.46	+0.35	+0.05	+1.31	+2.39

Table 3: The instant improvement of the reviewing stage on different entity types in CoNLL-03 and OntoNotes-5.0

the reviewing stage, demonstrating the effectiveness of our proposed reviewing stage.

(3) Table 2 shows that L&R obtains tight error bounds among all the steps, demonstrating better stability against the task order. For example, L&R lowers the error bound by 24%, 38%, 69% at step 2, 3, 4 on CoNLL-03.

(4) Figure 3 shows that the values on the diagonal line of the confusion matrix of L&R are higher compared to those of ExtendNER. This indicates that L&R discriminates more correctly between different entity types which is one of the reasons of its improvement.

6.2 Improvement of the Reviewing Stage

In order to further understand the improvement of the reviewing stage, we break down its source into two parts. The first part comes from the instant improvement after conducting the reviewing stage at each step. We report the average F1 before/after reviewing on the fifth/third line of Table 2. The second part comes from the improvement of the previous steps which alleviates the error propagation caused by the distillation. This accumulated improvement is reported on the third line of Table 2. From the first and the third line of the table, we can observe that L&R achieves more improvement against ExtendNER on later steps. From the third and the fifth line of the table, we can see that L&R achieves an average of 4 and 1 improvement on CoNLL-03 and OnteNotes-5.0 at each step.

We also report the instant improvement of the reviewing stage on different entity types in Table 3. From the table we can see that different entity types obtain different gain from the reviewing stage. This is rational because different types have different intrinsic difficulty.

6.3 Inter-type Confusion

To verify our hypothesis that L&R alleviates the inter-type confusion and thus brings improvement, we plot the normalized confusion matrix between different types based on the predictions at the final step (Figure 3). Concretely, we use the 'B-X' (X denotes a specific entity type) label in the ground truth as the true labels, and use the 'B-X' label in the model's predictions as the predicted labels. From the figures we can see that, the values on the diagonal line of the confusion matrix of L&R are higher compared to those of ExtendNER. This indicates that L&R discriminates more correctly between different entity types compared to ExtendNER. These results are in consistent with the improvements in Table 3.

6.4 Influence of Task Order

In order to explore the effect of task orders, we plot the performance of L&R and ExtendNER at

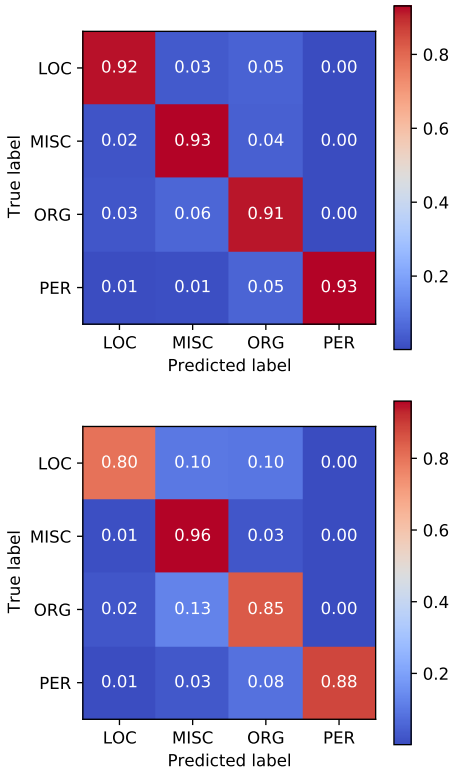

Figure 3: The normalized confusion matrices based on the predictions of L&R (up) and Extend (down).

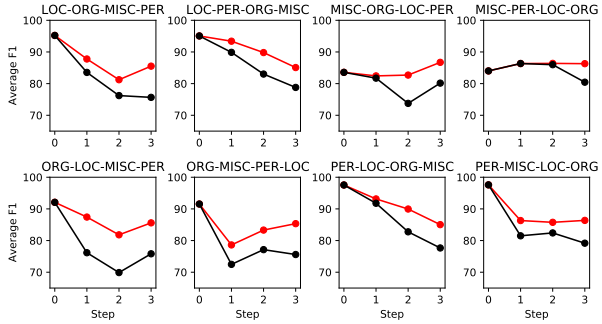

Figure 4: The performance of L&R (red) and ExtendNER (black) at each step under 8 sampled task orders.

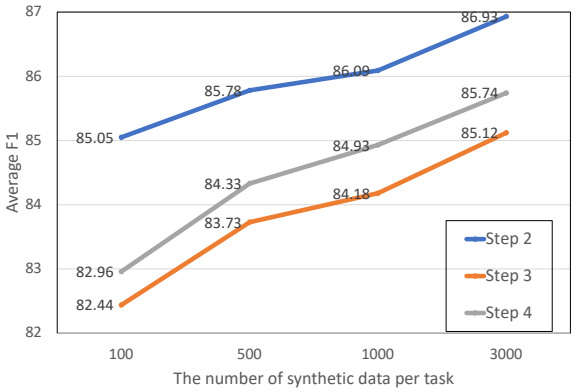

Figure 5: The performance of L&R at each step using different number of synthetic data per task.

each step under 8 sampled task orders on CoNLL-03 in Figure 4. From the figure, we can observe that: (1) Under all task orders, the performance of the methods drops with the step increases. This is in line with our expectation because the test sets and the type sets are incrementally expanding, indicating more difficult tasks. (2) Different methods under the same order show the similar trends where L&R shows a higher average F1 at each step. (3) Although the performance fluctuate at the middle steps, they converge at the final step. L&R gets a more converged result between 0.85 and 0.86 which demonstrates its robustness to the task orders. Besides, we calculate the error bounds to get a quantitative understanding. From Table 2 we can see that, the error bounds of L&R are lower than that of ExtendNER which also demonstrates the performance of L&R is less sensitive to the task orders.

6.5 Quantity of Synthetic Samples

To explore how much does the number of synthetic samples influences our performance, we conduct the experiments on CoNLL-03 with 100, 500, 1000, 3000 synthetic samples per task. From the Figure 5 we can see that, generating 100 samples per task is enough for an improvement of 5.05 against ExtendNER at the final step. Besides, the model performance conforms to the general rule of better performance with more data.

7 Conclusion

In this paper, we propose a novel framework introducing the reviewing stage to alleviate the catastrophic forgetting and intra-task confusion issues for NER under the type-incremental setting. After the learning step, we further distill the student and the teacher on the synthetic sample augmented dataset to get an enhanced student. Our experiments on the two benchmarks CoNLL-03 and OntoNotes-5.0 demonstrate that L&R is less prone to the intra-task confusion and outperforms the state-of-the-art method.

Acknowledgements

This work is supported by the National Key Research and Development Program of China (No.2020AAA0109400) and the National Natural Science Foundation of China (No. 61876009). We thank Yongwei Zhao for his valuable suggestions.

References

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Ian J Goodfellow, Mehdi Mirza, Da Xiao, Aaron Courville, and Yoshua Bengio. 2013. An empirical investigation of catastrophic forgetting in gradient-based neural networks. *arXiv preprint arXiv:1312.6211*.

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. 2015. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*.

Eduard Hovy, Mitch Marcus, Martha Palmer, Lance Ramshaw, and Ralph Weischedel. 2006. Ontonotes: the 90% solution. In *Proceedings of the human language technology conference of the NAACL, Companion Volume: Short Papers*, pages 57–60.

Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, et al. 2017. Overcoming catastrophic forgetting in neural networks. *Proceedings of the national academy of sciences*, 114(13):3521–3526.

Guillaume Lample, Miguel Ballesteros, Sandeep Subramanian, Kazuya Kawakami, and Chris Dyer. 2016. Neural architectures for named entity recognition. *arXiv preprint arXiv:1603.01360*.

Xuezhe Ma and Eduard Hovy. 2016. End-to-end sequence labeling via bi-directional lstm-cnns-crf. *arXiv preprint arXiv:1603.01354*.

Marc Masana, Xialei Liu, Bartlomiej Twardowski, Mikel Menta, Andrew D Bagdanov, and Joost van de Weijer. 2020. Class-incremental learning: survey and performance evaluation on image classification. *arXiv preprint arXiv:2010.15277*.

Michael McCloskey and Neal J Cohen. 1989. Catastrophic interference in connectionist networks: The sequential learning problem. In *Psychology of learning and motivation*, volume 24, pages 109–165. Elsevier.

Natawut Monaikul, Giuseppe Castellucci, Simone Filice, and Oleg Rokhlenko. 2021. Continual learning for named entity recognition. In *Proceedings of the Thirty-Fifth AAAI Conference on Artificial Intelligence*.

German I Parisi, Ronald Kemker, Jose L Part, Christopher Kanan, and Stefan Wermter. 2019. Continual lifelong learning with neural networks: A review. *Neural Networks*, 113:54–71.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. 2019. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32:8026–8037.

Sylvestre-Alvise Rebuffi, Alexander Kolesnikov, Georg Sperl, and Christoph H Lampert. 2017. icarl: Incremental classifier and representation learning. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pages 2001–2010.

Anthony Robins. 1995. Catastrophic forgetting, rehearsal and pseudorehearsal. *Connection Science*, 7(2):123–146.

Erik F Sang and Fien De Meulder. 2003. Introduction to the conll-2003 shared task: Language-independent named entity recognition. *arXiv preprint cs/0306050*.

Hanul Shin, Jung Kwon Lee, Jaehong Kim, and Jiwon Kim. 2017. Continual learning with deep generative replay. *arXiv preprint arXiv:1705.08690*.

Fan-Keng Sun, Cheng-Hao Ho, and Hung-Yi Lee. 2019. Lamol: Language modeling for lifelong language learning. *arXiv preprint arXiv:1909.03329*.

Sebastian Thrun. 1998. Lifelong learning algorithms. In *Learning to learn*, pages 181–209. Springer.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, et al. 2019. Huggingface's transformers: State-of-the-art natural language processing. *arXiv preprint arXiv:1910.03771*.

Tongtong Wu, Xuekai Li, Yuan-Fang Li, Reza Haffari, Guilin Qi, Yujin Zhu, and Guoqiang Xu. 2021. Curriculum-meta learning for order-robust continual relation extraction. *CoRR, abs/2101.01926*.

Yue Wu, Yinpeng Chen, Lijuan Wang, Yuancheng Ye, Zicheng Liu, Yandong Guo, and Yun Fu. 2019. Large scale incremental learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 374–382.

Ye Xiang, Ying Fu, Pan Ji, and Hua Huang. 2019. Incremental learning using conditional adversarial networks. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 6619–6628.

Hang Yan, Tao Gui, Junqi Dai, Qipeng Guo, Zheng Zhang, and Xipeng Qiu. 2021. A unified generative framework for various ner subtasks. *arXiv preprint arXiv:2106.01223*.

Yue Zhang and Jie Yang. 2018. Chinese ner using lattice lstm. *arXiv preprint arXiv:1805.02023*.

Order	CoNLL-03	OntoNotes-5.0
1	$LOC \rightarrow ORG \rightarrow MISC \rightarrow PER$	$ORG \rightarrow PER \rightarrow GPE \rightarrow DATE \rightarrow CARD \rightarrow NORP$
2	$LOC \rightarrow PER \rightarrow ORG \rightarrow MISC$	$DATE \rightarrow NORP \rightarrow PER \rightarrow CARD \rightarrow ORG \rightarrow GPE$
3	$MISC \rightarrow ORG \rightarrow LOC \rightarrow PER$	$GPE \rightarrow CARD \rightarrow ORG \rightarrow NORP \rightarrow DATE \rightarrow PER$
4	$MISC \rightarrow PER \rightarrow LOC \rightarrow ORG$	$NORP \rightarrow ORG \rightarrow DATE \rightarrow PER \rightarrow GPE \rightarrow CARD$
5	$ORG \rightarrow LOC \rightarrow MISC \rightarrow PER$	$CARD \rightarrow GPE \rightarrow NORP \rightarrow ORG \rightarrow PER \rightarrow DATE$
6	$ORG \rightarrow MISC \rightarrow PER \rightarrow LOC$	$PER \rightarrow DATE \rightarrow CARD \rightarrow GPE \rightarrow NORP \rightarrow ORG$
7	$PER \rightarrow LOC \rightarrow ORG \rightarrow MISC$	
8	$PER \rightarrow MISC \rightarrow LOC \rightarrow ORG$	

Table 4: The sampled task orders of CoNLL-03 and OntoNotes-5.0.

A Implementation Details

We use uncased BERT-base as our encoder (Devlin et al., 2018). The models are implemented in Pytorch (Paszke et al., 2019) on top of the BERT Huggingface implementation (Wolf et al., 2019), and are trained on a single GeForce RTX 3090 GPU. We set the batch size as 32, the max sentence length as 128, the max training epoch number as 20 with early stopping (patience=3). We use Adam (Kingma and Ba, 2014) as our optimizer with the learning rate 5e-5 for all modules. For all student models, we set the temperature as 2 and $\alpha = \beta = 1$ for the weighted sum of the losses. For L&R, we generate 3000 samples for each previous task by default. We sample 8 and 6 task orders for CoNLL-03 and OntoNotes-5.0 respectively (listed in Table 4). For efficiency, we use a one-layer LSTM model as our generator and find it enough to achieve encouraging performance. The average runtime (training and inference) time is 10 min/task and the size is 50 MB for CoNLL-03.