Continual Learning for Named Entity Recognition

Natawut Monaikul[*,1] **Giuseppe Castellucci,**[2] **Simone Filice,**[2] **Oleg Rokhlenko**[2]
[1]University of Illinois at Chicago, Chicago, IL, USA
[2]Amazon, Seattle, WA, USA
monaiku1@uic.edu, {giusecas, filicesf, olegro}@amazon.com

Abstract

Named Entity Recognition (NER) is a vital task in various NLP applications. However, in many real-world scenarios (e.g., voice-enabled assistants) new named entity types are frequently introduced, entailing re-training NER models to support these new entity types. Re-annotating the original training data for the new entity types could be costly or even impossible when storage limitations or security concerns restrict access to that data, and annotating a new dataset for all of the entities becomes impractical and error-prone as the number of types increases. To tackle this problem, we introduce a novel Continual Learning approach for NER, which requires new training material to be annotated only for the new entity types. To preserve the existing knowledge previously learned by the model, we exploit the *Knowledge Distillation* (KD) framework, where the existing NER model acts as the teacher for a new NER model (i.e., the student), which learns the new entity types by using the new training material and retains knowledge of old entities by *imitating* the teacher's outputs on this new training set. Our experiments show that this approach allows the student model to "progressively" learn to identify new entity types without forgetting the previously learned ones. We also present a comparison with multiple strong baselines to demonstrate that our approach is superior for continually updating an NER model.

1 Introduction

Named Entity Recognition (NER) or Slot Filling are common NLP tasks that require extracting entities from unstructured text. In the standard setting, an NLU system is trained to recognize a fixed set of entity types (e.g., *Person*, *Organization*, *Protein*, or *Law*). However, in real scenarios, NLU systems are continually evolving to support new functionalities and this typically implies that new entity types must be recognized. For instance, voice assistants like Siri or Alexa continually introduce new intents to their capabilities and consequently new entity types are often added to their slot filling models. The simplest solution for accommodating new entity types consists of (*i*) updating the annotation guidelines to cover the new types and re-annotating the training material; (*ii*) restructuring the model to support the new types; and (*iii*) re-training the model from scratch on the modified data. This procedure is both expensive and time-consuming. Furthermore, the original training material may no longer be available due to storage constraints or privacy concerns. Finally, the original training data may not necessarily contain enough examples of the additional types.

A possible solution would be to annotate a new dataset for all the entities that the model should recognize. However, this can easily become impractical and error-prone when the number of entities is large. Another option is to annotate a new dataset *only for the new entity types* and use this data to update the model in a Continual Learning (CL) setting (Lange et al. 2019). However, this approach is vulnerable to the *catastrophic forgetting* of previous entity types, a well-documented concern in CL (Chen et al. 2018). We instead seek to have the existing NER model impart knowledge about the entities it already supports to a new model. One way to do so would be a self-training approach, in which the existing model is used to annotate the new dataset, i.e., filling in the gaps from the manual annotation of only the new entity types. This dataset can then be used to train the new model to recognize the new and old entity types.

One disadvantage of self-training is that the errors of the old model are propagated to the new model without taking into account the uncertainty of the old model's predictions. We argue that the old model uncertainty can help the new one to better learn about the old entity types. In order to leverage such information, in this paper, we investigate applying *Knowledge Distillation* (KD) (Hinton, Vinyals, and Dean 2015) to the CL problem for NER. This technique involves incorporating the predictions of a "teacher" model into the objective function of a "student" model being trained to perform a similar but slightly modified task. The student is encouraged to behave similarly to the teacher by learning about its output probability distribution, rather than just the labels. Though KD has traditionally been used for the purpose of model compression, i.e., teaching a smaller and simpler model to make similar but faster predictions than a large, complex model, more recent works have applied KD in a CL setting, such as incrementally learning multiple computer vision tasks (Li and Hoiem 2018).

For the task of NER in a CL setting, we treat a trained NER model as the teacher and a new model that supports additional entity types as the student. We conduct an extensive empirical investigation using BERT-based models on

[*]Work done during an internship in Amazon.

two NER datasets, and we show that KD allows new models to "progressively" learn to identify new entity types while retaining previous types using unseen data that have only been annotated for the new types.

Our contributions are (*i*) we show how to adapt CL techniques to the NLU domain for progressively learning new entity types for NER; (*ii*) our results on two English NER datasets demonstrate that our CL approach enables the model to continuously learn new entity types without losing the capability to recognize previously acquired types; and (*iii*) we show that our semi-supervised strategy achieves results comparable to a fully supervised setting.

The rest of the paper is organized as follows. Section 2 discusses related work. Section 3 introduces our approach, and in Section 4 we discuss the experimental results. Finally, Section 5 discusses the conclusions.

2 Related Work

Continual Learning (also referred to as lifelong learning (Chen et al. 2018)) studies the problem of learning from a stream of data. This stream can change over time in terms of an evolution in the input distribution or by incorporating new tasks. The goal of CL is to gradually extend the knowledge acquired in a model (Lange et al. 2019), without incurring any Catastrophic Forgetting (CF) (Goodfellow et al. 2013), mainly to account for changes in the data distribution. Previous work mostly focused on Computer Vision (Shmelkov, Schmid, and Alahari 2017; Li and Hoiem 2018; Rannen et al. 2017) by using Knowledge Distillation (Hinton, Vinyals, and Dean 2015) as the base framework.

In the context of NLU, some works focus on the Online Learning aspect of the CL (Filice et al. 2014). Liu, Ungar, and Sedoc (2019) have approached the task of CL by proposing an unsupervised method to train sentence encoders that can continually learn features from new corpora. Hu et al. (2019) introduce the Parameter Generation and Model Adaptation framework for image and text classifiers. A neural network with two sets of parameters is proposed, where the first set is shared across different tasks while the second set is a placeholder, which is filled by a generator network for each test instance.

In terms of CL for NER specifically, the work which is most related is by Chen and Moschitti (2019), who propose an approach for transferring the knowledge of a neural network for sequence labeling trained on one (source) dataset to a new model trained on another (target) dataset in which new label categories appear. Their approach is to add a "neural adapter" to learn the difference between the source and the target label distribution. The main difference from our work is that in each step, we have only the new entity types annotated in the new dataset, as opposed to having all the types annotated. Greenberg et al. (2018) present a method for training a single CRF extractor from multiple datasets. They use marginal likelihood training to strengthen the knowledge acquired on labels that are present in the data, while filling in "missing labels" for each dataset. Nguyen et al. (2019) propose a model for solving the Lifelong Learning problem for Vietnamese NER. The model is a deep neural network, where at each step the network extracts a set of "prefix-features" into a knowledge base, which are then used to solve a new NER task.

Self-training (Triguero, García, and Herrera 2015) is the semi-supervised iterative procedure in which a model is first trained on a small labeled dataset, and then it is used to predict labels of unlabeled examples. Then, confidently labeled examples are selected and included in the training set that is adopted to re-train the model. Our setting is similar, since we assume to have partially-annotated data where only new entity types are labeled, and we use a previously trained model to predict the presence of old entity types. However, our approach differs from self-training because instead of using the labels predicted by the model directly, we use the output probability distribution of the model via distillation loss. This allows for taking into account model uncertainty and mitigating the propagation of the errors of the old model.

3 Continually Learning Named Entities

In this section, we present our CL approach to add new entity types to an already trained NER model. Section 3.1 briefly discusses the KD approach (Hinton, Vinyals, and Dean 2015). Then, Section 3.2 introduces our model update strategy.

3.1 Knowledge Distillation

Knowledge Distillation (Hinton, Vinyals, and Dean 2015) is a technique to transfer the knowledge between models with a process called "distillation". KD was proposed to transfer knowledge from a cumbersome model into a smaller and more efficient one. This technique has been used to compress huge language models, such as BERT (Devlin et al. 2019), into smaller ones, e.g., DistilBERT (Sanh et al. 2019).

KD works by first training a teacher model on some task. After training, the teacher will be able to assign a probability distribution p^T over the categories of the task. p^T is typically computed by the softmax over the logits z_j of the last layer of a neural network architecture, for each category j. The softmax is defined as $softmax(z_j) = \frac{exp(z_j/T_m)}{\sum_l exp(z_l/T_m)}$, where T_m is a temperature hyper-parameter[1], which is typically set to 1. The teacher is trained by minimizing the cross-entropy between the one-hot distribution and its predicted output distribution. Then, a student model is trained by imitating the teacher output distribution over a transfer dataset. That is, the student is trained by computing the KL divergence between p^T (also referred to as the "soft targets") and the student output probability distribution p^S.

KD has been used for CL in the image domain (Shmelkov, Schmid, and Alahari 2017; Li and Hoiem 2018; Rannen et al. 2017) – for example, by updating an image classifier to support new categories while preventing catastrophic forgetting (Li and Hoiem 2018).

3.2 Continual Learning for Sequence Tagging

In order to update an NER model to support new entity types, we adopt KD to prevent catastrophic forgetting of

[1] As discussed in (Hinton, Vinyals, and Dean 2015), the T_m parameter can be tuned to obtain a softer distribution.

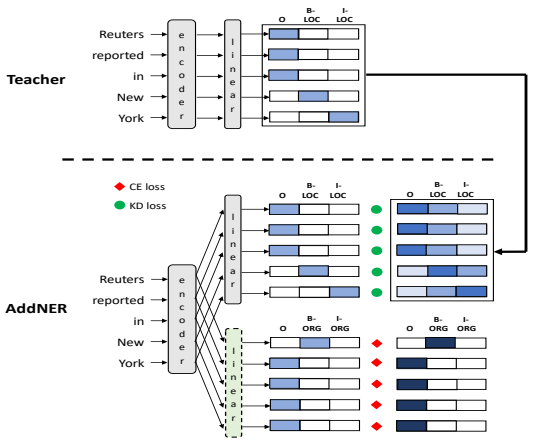


Figure 1: AddNER model: the encoder and the linear layer are copied for the old entities (grey, solid border). A new output layer (green, dashed border) is meant to recognize a new entity type. The teacher provides the soft targets for KD, while the new training data provides the one-hot encoding to train for the new entity type.

the knowledge already acquired in the model. Let us consider that we have trained an NER model M_i on the set of entity types $E_i = \{e_1, \ldots, e_n\}$, and we aim to update the model in order to also recognize the set of new entity types $E^{new} = \{e_{n+1}, e_{n+2}, \ldots, e_{n+m}\}$. We model this problem as acquiring a new student named-entity recognizer M_{i+1} able to tag sentences with respect to the full set $E_{i+1} = E_i \cup E^{new} = \{e_1, ..., e_{n+m}\}$. We are given a new dataset D^{new} consisting of sentences annotated only with respect to E^{new}. Note that the sentences in D^{new} potentially also contain tokens of old types in E_i, but this information is not annotated in this dataset. We would like M_{i+1} to perform well on the new entity types in E^{new} while preserving the performances on the old entity types in E_i, i.e., we aim to prevent catastrophic forgetting.

We propose two different student models: the AddNER and ExtendNER models depicted in Figures 1 and 2, respectively. In both cases, M_i acts as the teacher for KD. Both M_i and M_{i+1} are realized as neural networks composed of encoder layers, which produce an h-dimensional contextual representation for each token of a sentence, and tagger layers, which output probability distributions for each token with respect to the target labels. In the following, we assume that the tags are provided in the IOB format: for each token, its tag can indicate the beginning of an entity (B−), a token inside an entity (I−), or neither, i.e., the "other" tag (O), which indicates that the token is not an entity.

3.3 AddNER Model

In this schema, the student M_{i+1} is a clone of the teacher M_i, and a new output layer is added[2] to recognize the entity types in E^{new} (Figure 1). During training, M_{i+1} observes examples from D^{new}. Given that this dataset contains instances annotated only for E^{new}, we adopt KD to prevent catastrophic forgetting on the already acquired entity types. This means that each sentence is also run through the teacher M_i, which provides the soft targets for training the student on the old entity types. At the same time, the annotated data is used to learn how to recognize the new entity types. M_{i+1} is thus trained with two losses: one penalizing forgetting previous entity types (the green circles in Figure 1) and one that penalizes errors on the new entity types (the red diamonds).

More formally, for each token of a sentence in D^{new} with associated category y, we obtain the probability distribution of the teacher $p_{E_i}^{M_i}$ with respect to the old entity types in E_i. At the same time, we tag each token of the sentence with M_{i+1} obtaining $p_{E_i}^{M_{i+1}}$ for the same entities. The probability distribution $p_{E_i}^{M_i}$ will be used as the soft target to train M_{i+1}. That is, M_{i+1} is trained by minimizing the KL divergence between the teacher and the student output distributions, i.e., $L_{KL}^{Add} = KL(p_{E_i}^{M_i}, p_{E_i}^{M_{i+1}})$. The student model will also produce the output probability distribution with respect to the new entity types with the new output layer, i.e., $p_{E^{new}}^{M_{i+1}}$. This is used to compute the cross-entropy loss, i.e., $L_{CE}^{Add} = CE(y, p_{E^{new}}^{M_{i+1}})$, with the one-hot encoding derived from the annotations in D^{new}. The model is trained on the weighted sum of the two losses for each token, i.e., $L^{Add} = \alpha L_{KL}^{Add} + \beta L_{CE}^{Add}$, where α and β are two hyperparameters weighting the contribution of the two losses.

Because the AddNER model contains multiple output layers, the different outputs for a single token need to be consolidated. In particular, it may be the case that the prediction of one layer conflicts with that of another. To resolve these conflicts, we developed a set of heuristics that merges the outputs of the different layers for each token:

- If all layers predict the O tag, then output=O.
- If exactly one layer predicts a B− tag and the other layers predict O, then output=B−.
- If multiple layers predict B− tags and the remaining layers predict O, then output=B− with the highest probability.
- If a layer predicts an I− tag, output=I− only if it matches the preceding tag in the sequence, i.e., the tag of the previous token must be a B− or I− of the same entity type. Otherwise, the output of that layer is treated as O, and the heuristics are applied again to determine the final output.

3.4 ExtendNER Model

In this setting, M_{i+1} is again a clone of M_i, but the tagger output layer is *extended*, i.e., new dimensions are added to the layer to recognize the m new entity types in E^{new} (Figure 2). Assuming that M_i was able to recognize n entity types, its tagger layer can be considered as a matrix with dimension $h \times (2n+1)$. The output layer of M_{i+1} will then be

[2]Notice that (*i*) if the model is iteratively extended, a new output layer is added at each iteration; (*ii*) if m is the number of new entity types added in a given iteration, then the matrix of the corresponding output layer will have a dimensionality of $h \times (2m+1)$ to accommodate the new labels B−X, I−X for each new entity type X, and O, according to the IOB tagging schema.

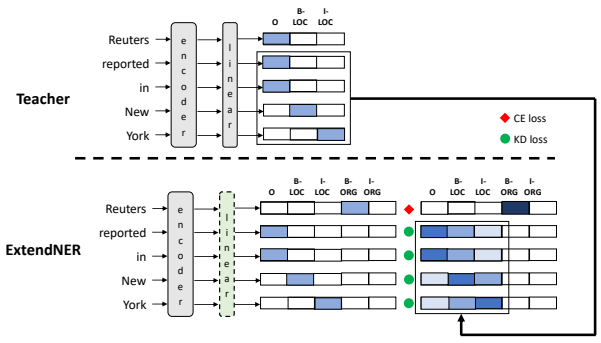


Figure 2: `ExtendNER` model: the teacher encoder layers (grey, solid border) are copied to build the new model. A new output layer (green, dashed border) substitutes the old one with the additional dimensions for the new entity type. If a token is annotated in the new training data, it will contribute to the cross entropy loss. Otherwise, the teacher soft targets will be used for the KD loss.

extended to be a matrix with dimension $h \times (2n + 2m + 1)$ in order to accommodate the new entity types[3].

Again, we adopt KD to prevent catastrophic forgetting with two loss functions: (*i*) the KL divergence between the teacher and the student outputs for the entity types in E_i; (*ii*) the cross-entropy between the student output and the gold annotation in D^{new} for the entity types in E^{new}.

More formally, for each token in a sentence with associated category y, the model will compute either the KL divergence or the cross-entropy loss, depending on the value of y. In particular, when $y = \texttt{O}$, the KL divergence is computed between the teacher output distribution and the student output distribution (the green circles in Figure 2): $L^{Ex}_{KL} = KL(p^{M_i}_{E_i}, p^{M_{i+1}}_{E_i})$. Instead, if $y \neq O$ (i.e., the token was labeled as one of the entity types in E^{new}) the model will compute the cross-entropy loss (the red diamonds in Figure 2), i.e., $L^{Ex}_{CE} = CE(y, p^{M_{i+1}}_{E^{new}})$. Again, the model is trained on the weighted sum of the two losses, i.e., $L^{Ex} = \alpha L^{Ex}_{KL} + \beta L^{Ex}_{CE}$.

The final tags are obtained by the Viterbi algorithm (Vintsyuk 1968) applied as a post-process for the $n + m$ entity types. The emission probabilities are given by the M_{i+1} outputs, i.e., $p^{M_{i+1}}_{E_{i+1}}$, while the transition probabilities have been hard-coded to prevent impossible transitions. The transition probabilities are represented as a matrix T_r of dimensions $(2n + 2m + 1) \times (2n + 2m + 1)$, where $T_r(p, r) = 0$ if $p \rightarrow r$ is an impossible transition, e.g., an `I-` tag following an `O` tag or an `I-X` tag following an `I-Y` tag, where `X` and `Y` are two different entity types. Otherwise, $T_r(p, r) = \frac{1}{N(p)}$, where $N(p)$ is the number of allowed transitions starting from p, i.e., the probability mass is equally distributed among the admissible transitions starting from the tag p.

[3]Notice that the additional $2m$ is for accommodating the `B-X` and `I-X` categories for each new entity type `X` in E^{new}.

	CoNLL-03		
	Training	**Validation**	**Test**
PER	6,532	1,829	1,597
LOC	7,125	1,832	1,664
ORG	6,271	1,325	1,654
MISC	3,398	916	698
Total	23,326	5,902	5,613

Table 1: Distribution of entity labels in CoNLL-03.

	OntoNotes		
	Training	**Validation**	**Test**
ORG	24,163	3,798	2,002
PER	22,035	3,163	2,134
GPE	21,938	3,649	2,546
DATE	18,791	3,208	1,787
CARD	10,901	1,720	1,005
NORP	9,341	1,277	990
Total	107,169	16,815	10,464

Table 2: Distribution of entity labels in OntoNotes.

4 Experimental Evaluation

Without loss of generality, we consider the case where the dataset D^{new} is annotated only for one new entity type e_{i+1}, i.e., $E^{new} = \{e_{i+1}\}$. This allows us to test the capability of our approach to learn one entity type at a time. Thus, at step $i + 1$, we train the model M_{i+1} to recognize the entity types $\{e_1, ..., e_{i+1}\}$, given the teacher M_i.

4.1 Datasets

To evaluate our approach, we used two well-known NER datasets: CoNLL-03 English NER (Tjong Kim Sang and De Meulder 2003) and OntoNotes (Hovy et al. 2006).

CoNLL-03. The English version of this dataset contains roughly 22k sentences pulled from news stories that have been annotated w.r.t. 4 named entity types: *Person* (PER), *Location* (LOC), *Organization* (ORG), and *Miscellaneous* (MISC). The distribution of the entity types across the official training, validation, and test sets are given in Table 1.

OntoNotes. The English version of this dataset includes about 144k sentences drawn from a variety of texts, such as news, transcribed telephone conversations, and weblogs. Although the dataset has been annotated for 18 entity types, we restrict our investigation to the 6 most represented types to ensure a sufficient number of examples for training. The 6 types we considered are: *Organization* (ORG), *Person* (PER), *Geo-Political Entity* (GPE), *Date* (DATE), *Cardinal* (CARD), *Nationalities and Religious Political Group* (NORP). Dataset statistics are shown in Table 2.

4.2 Experimental Setup

To test our approach in the CL setting, we divided the official training and validation sets of CoNLL-03 and OntoNotes into four and six disjoint subsets, $D^1, D^2, \ldots$, respectively:

CoNLL-03 Permutations
PER → LOC → ORG → MISC
PER → MISC → LOC → ORG
LOC → PER → ORG → MSC
LOC → ORG → MISC → PER
ORG → LOC → MISC → PER
ORG → MISC → PER → LOC
MISC → PER → LOC → ORG
MISC → ORG → PER → LOC
OntoNotes Permutations
ORG → PER → GPE → DATE → CARD → NORP
DATE → NORP → PER → CARD → ORG → GPE
GPE → CARD → ORG → NORP → DATE → PER
NORP → ORG → DATE → PER → GPE → CARD
CARD → GPE → NORP → ORG → PER → DATE
PER → DATE → CARD → GPE → NORP → ORG

Table 3: The different permutations in which entity types are added to our models for each dataset.

each D^i is annotated only for the entity type e_i. We first train an initial model M_1 on D^1 for e_1. This model becomes the teacher for e_1 with which we train a student model M_2 on the second slice D^2, which is labeled for e_2 only: M_2 thus learns to tag both e_1 and e_2. We repeat this process for each slice D^i, i.e., training a new student on a new slice using the previous trained model as the teacher for the previously learned labels. At each step i, we use the i-th slice of the validation set for early stopping and evaluate the resulting model M_i on the official test set annotated for the entity types $\{e_1, ..., e_i\}$. In order to factor out any dependencies on the order with which we add entity types, we train/test with different permutations of the order. In particular, we perform 8 and 6 different permutations[4] for CoNLL-03 and Ontonotes, respectively, as shown in table 3. We report the average performances over the different permutations.

We compare our approaches with different strong baselines. To investigate the extent to which using KD contributes to imparting knowledge of previously-learned labels, we compare our approach to a non-KD approach. In particular, after training a model M_i, we use it to annotate the new slice D^{i+1} for training M_{i+1}; with this self-training approach, D^{i+1} will contain the annotations for all of the entity types $\{e_1, ..., e_{i+1}\}$. A new student model can thus be trained on D^{i+1} with respect to all of the labels by using only the cross-entropy. We will refer to this approach as *hard label*. We do not compare results with Chen and Moschitti (2019) since in their work, all entity types are labeled to train subsequent models (whereas we require annotations only for the new entity types).

We compare also to a transfer learning approach in which a new output layer is added to recognize the new entity type, but the old output layers are frozen in order to retain old types while training on the new dataset that is only annotated for the new type. Note that this is comparable to the `AddNER` model in structure, but frozen parameters are used instead of KD to preserve knowledge of previously learned labels. In this setting, we may choose to freeze the encoder when updating the model (*frozen transfer*) or not (*free transfer*).

We hypothesize that annotating a new dataset for multiple entity types requires more effort and is more error-prone than annotating it for only one new entity type. To investigate the extent to which our approach can compensate for having only partially-annotated data, we also tested three other baselines that are trained on fully-annotated data. In particular, we evaluate a CL model that, at each step i, is trained on the dataset slice D^i annotated for the new entity type e_i as well as for all the previous types, $\{e_1, ..., e_{i-1}\}$. We call this a *CL fully-annotated* model. We also look at two non-CL models – one that is trained from scratch on the fully-annotated D^i at each step i, and one that is trained on all of the fully-annotated dataset slices observed up to and at step i: $(D^1, ..., D^i)$. We refer to these as *non-CL last slice* models and *non-CL complete* models, respectively.

For each of the models, the encoder layer is a BERT-based model (Devlin et al. 2019). The models were implemented in Pytorch (Paszke et al. 2017) on top of the BERT Huggingface implementation (Wolf et al. 2019), and training was performed on a single Nvidia V100 GPU.

After initial experimentation with different hyperparameters, we chose to train the models with a batch size of 32, a max sentence length of 50 tokens, and a learning rate of 5e-5 for 20 epochs with early stopping (patience=3). For all student models, a temperature $T_m = 2$ was used, and $\alpha = \beta = 1$ for the weighted sum of the losses.

4.3 Results & Discussion

At each step i, we computed the Precision, Recall, and F1 scores for each entity type. In the following, we report the macro-average F1 score w.r.t. all the types $\{e_1, ..., e_i\}$, averaged over all of the permutations. We also report statistical significance tests to assess the differences between the models' performances and to answer our Research Questions (RQ). We computed the F1 score for each model at the sentence level, i.e., for each sentence in the test set that contains at least one annotated entity, we computed the F1 score with respect to all supported entities in that sentence alone. This fine-grained analysis ensures a sufficiently large sample size (made of independent samples (Gillick and Cox 1989)) to meaningfully perform the test. Statistical significance tests refer to a paired t-test with $\alpha = 0.05$.

Transfer Learning for CL Our first RQ is: *Is transfer learning sufficient for learning new entity types, while preserving knowledge of previously learned entity types?*

We performed an experiment with the *frozen transfer* and the *free transfer* models. We trained a model M_1 on D^1, and then trained the transfer learning models on D^2, with the encoder layers either frozen or not, on CoNLL-03. For the *frozen transfer* model, the average overall F1 scores at step 2 was 57.49, while for the *free transfer* models, the average performance at step 2 was 56.21. In the case of *free transfer* models, the low F1 score was largely due to catastrophic forgetting. In the case of *frozen transfer* models, while a frozen

[4]We created the permutations such that each entity type appears in each position the same number of times, i.e., twice for CoNLL-03 and once for OntoNotes. We did not try further permutations due to budget limitations.

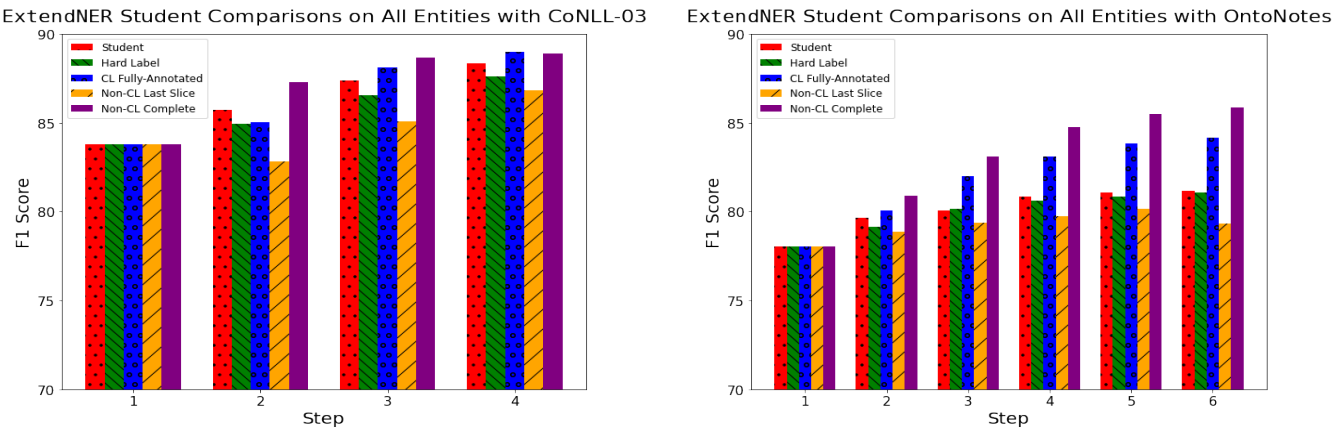


Figure 3: Plots of overall F1 scores of ExtendNER student model against several comparison models at each step with the CoNLL-03 dataset (left) and the OntoNotes dataset (right). Scores at each step are averaged over all permutations.

encoder and old output layer mitigates any catastrophic forgetting, the frozen encoder restricts the model's ability to adequately learn the new entity type. This strongly suggests that the transfer learning approach, whether or not the encoder is frozen, cannot be successfully adopted for NER models in the CL setting. Because of the low F1 scores at step 2 for CoNLL-03, we did not perform subsequent steps.

AddNER vs. ExtendNER Student Models Our second RQ is: *Are there differences between the AddNER and ExtendNER models in the CL setting?*

In our experiments, we found that for the CoNLL-03 dataset, the ExtendNER model performed 0.06 points better than the AddNER model on macro-F1 scores, while for the OntoNotes dataset, AddNER performed 0.04 points better than ExtendNER. These differences are not statistically significant ($p = 0.7$ and $p = 0.9$, respectively), suggesting that the two student models do not meaningfully differ. Because of this similarity in performance the following discussion reports results only for ExtendNER.

KD for CL with Limited Annotations Our third RQ is: *Do the student models effectively learn with KD in the CL setting when new datasets are only minimally-annotated?*

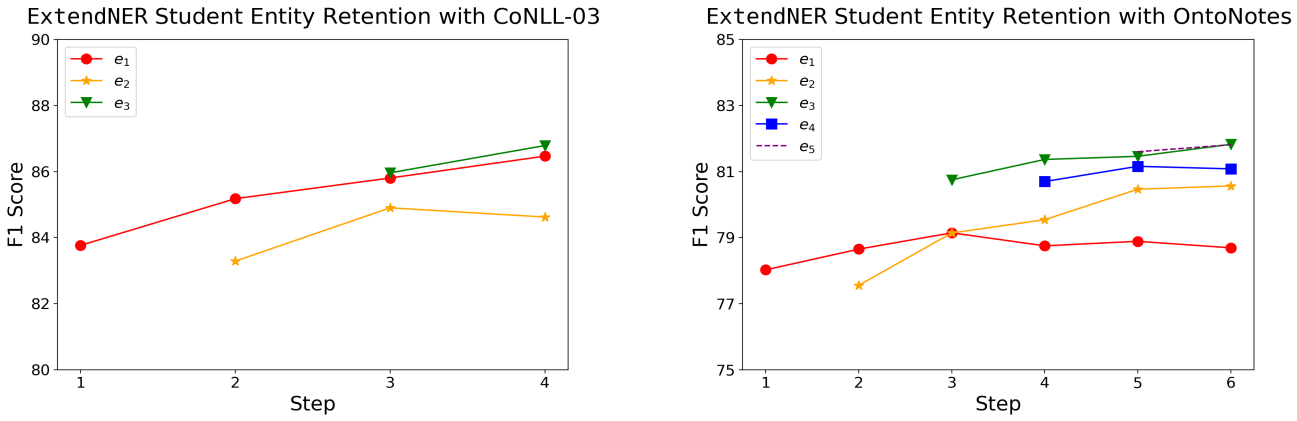
To address this question, we report the average F1 score averaged over all permutations, for each model, at each step i. These represent how each model M_i performs on all of its supported entity types, newly and previously learned. These results are shown in Figure 3, where the leftmost bar represents the score for the ExtendNER model trained with KD.

We can observe that the ExtendNER model showed an increase in overall F1 score from step to step. The student trained using KD does not degrade in overall performance as new entity types are added, despite the fact that each new slice of data is only annotated for the newly-added type. In other words, each student model M_{i+1} is able to simultaneously learn about $e_1, ..., e_i$ from the teacher and also learn about e_{i+1} from the annotations in D^{i+1}.

Moreover, note that the student model performed better than the corresponding hard-label model (the second bar in each cluster). We found that, over all steps and permutations, the ExtendNER student model performed on average 0.37 points better than the hard-label model in the CoNLL-03 test set, and 0.81 points better than the hard-label model in the OntoNotes test set (both significant, $p < 0.0001$). Although the difference is slight, the significance test suggests that indeed, on average, the ExtendNER student model trained through KD outperforms the hard-label approach[5].

Figure 3 shows also that the ExtendNER model outperforms the *non-CL last slice* models (fourth bar) at every step for both datasets. This suggests that the cloned encoder from M_i to M_{i+1} was critical in the student's success, i.e., training from scratch on the new data alone is not as effective, even if the data are annotated for all of the target entity types. We also note that the overall F1 scores of the *CL fully-annotated* models (third bar) are generally greater than those of the ExtendNER. This is rather unsurprising, as a *CL fully-annotated* model leverages knowledge from the encoder of the teacher model, as well as the true labels of all the entities. While this difference in terms of sentence-level F1 scores is slight (0.24 for CoNLL-03 and 0.78 for OntoNotes), it is a significant difference (both $p < 0.01$). The *ExtendNER* student model does not perform as well as if it had fully-annotated data; however, as the number of entity types increases, our models require significantly less annotation than the *CL fully-annotated* models and can still maintain a good overall performance. The same applies when comparing *ExtendNER* to the *non-CL complete* models (last bar): although the *non-CL complete* models have a higher F1, it requires all previous training data to be annotated for all entity types, which can become prohibitively costly.

[5]The statistical significance test we implemented ignores false positives, i.e., cases in which a sentence has no named entities, but a model predicts that at least one entity is present in the sentence. However, we found that both the ExtendNER and the hard-label models exhibit roughly the same rate of false positives for both datasets (about 6%).



(a) (b)

Figure 4: Plots of F1 scores of ExtendNER student models for each entity type, tracked over all steps. Note that at each step i, the point on the e_1 (red, circles) line represents the model's average F1 score on e_1 after $i-1$ additional entity types have been added, the point on the e_2 (orange, stars) line represents the models' average F1 score on e_2 after $i-2$ additional entity types have been added, and so on. Averages are taken over all permutations.

Knowledge Retention with KD Our last RQ is: *Do the student models retain knowledge of the previously learned entity types when learning a new one?*

We address this question by tracking the performance of the models on its learned entity types from step to step to see if the performance is retained as new types are added. In particular, for each permutation, we measure the F1 score of the initial model trained in step 1 on e_1, the F1 score of ExtendNER trained in step 2 on e_1, and so on. The average F1 score of e_1 at each step over all permutations then represents how well the student models retain knowledge of e_1. We can also track the performance on e_2 starting from step 2 onwards, e_3 starting from step 3 onwards, and so on, to see if similar trends exist. These results for ExtendNER with KD are shown in Figure 4. We can see that, in general, the F1 scores remain about the same or increase from step to step. This suggests that as new entity types are introduced, ExtendNER retains or even gains knowledge[6] on the entities that were trained through KD. Despite the downward trend of e_1 after step 3 on OntoNotes (Figure 4b), we found that ExtendNER's average performance on e_1 at step 6 (77.88 F1) is lower than step 1 (78.03 F1), but this difference is not significant ($p = 0.9$).

As a comparison, we again look at the *free transfer* models. The average F1 score of the initial models (for the single entity type e_1) across all permutations is 83.77 for the CoNLL-03 dataset; however, the average F1 score of the free transfer models for e_1 at step 2 is 34.95 for CoNLL-03. This is a clear demonstration of catastrophic forgetting – although the old output layer is frozen, the encoder is updated in such a way during training for the new entity type that it can no longer account for the old entity type.

We also computed the average difference, for every entity type in every permutation, between the F1 scores for every entity type at step i versus step $i+1$ to further measure the ability of ExtendNER to retain knowledge of previously learned labels after KD. We found that on the CoNLL-03 dataset, the average F1 score increased significantly by 1.308 between consecutive steps ($p = 0.0001$). On the OntoNotes dataset, the average F1 score increased by 0.0341, but this difference is not statistically significant, ($p = 0.7$). As we do not see any significant decrease in average F1 score from one step to the next, these results suggest that our approach prevents any catastrophic forgetting.

5 Conclusions

In this paper we presented an approach for Continual Learning for the task of Named Entity Recognition. We showed how we can extend a model to recognize new entity types. In particular, we demonstrated that we are able to update a model with a minimally labeled dataset, i.e., where only a new entity type is annotated. This setting is challenging, as common transfer learning approaches incur catastrophic forgetting. However, this can be useful in scenarios where either old training data cannot be used anymore (e.g., for privacy or storage constraints) or annotating data for multiple entity types is prohibitive (the more entities to annotate, the harder and more expensive the annotation task can be). Instead, we showed how we can prevent forgetting of already acquired knowledge by means of Knowledge Distillation in a teacher-student learning framework. Our experimental results demonstrate the efficacy of our proposal in preventing the forgetting and in learning new entity types step by step.

In the future, it would be interesting to integrate a learned CRF layer to take into account the global dependencies between tags. It would also be interesting to investigate similar approaches to update a model trained in one domain to support new entity types in a different domain.

[6] We believe this phenomenon is simply due to the data augmentation at each step.

References

Chen, L.; and Moschitti, A. 2019. Transfer learning for sequence labeling using source model and target data. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, 6260–6267.

Chen, Z.; Liu, B.; Brachman, R.; Stone, P.; and Rossi, F. 2018. *Lifelong Machine Learning*. Morgan-Claypool Publishers, 2nd edition. ISBN 1681733021.

Devlin, J.; Chang, M.-W.; Lee, K.; and Toutanova, K. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, 4171–4186. Minneapolis, Minnesota: Association for Computational Linguistics. doi:10.18653/v1/N19-1423. URL https://www.aclweb.org/anthology/N19-1423.

Filice, S.; Castellucci, G.; Croce, D.; and Basili, R. 2014. Effective Kernelized Online Learning in Language Processing Tasks. In de Rijke, M.; Kenter, T.; de Vries, A. P.; Zhai, C.; de Jong, F.; Radinsky, K.; and Hofmann, K., eds., *Advances in Information Retrieval*, 347–358. Cham: Springer International Publishing. ISBN 978-3-319-06028-6.

Gillick, L.; and Cox, S. J. 1989. Some statistical issues in the comparison of speech recognition algorithms. In *International Conference on Acoustics, Speech, and Signal Processing,*, 532–535. IEEE.

Goodfellow, I. J.; Mirza, M.; Xiao, D.; Courville, A.; and Bengio, Y. 2013. An Empirical Investigation of Catastrophic Forgetting in Gradient-Based Neural Networks.

Greenberg, N.; Bansal, T.; Verga, P.; and McCallum, A. 2018. Marginal Likelihood Training of BiLSTM-CRF for Biomedical Named Entity Recognition from Disjoint Label Sets. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, 2824–2829. Brussels, Belgium: Association for Computational Linguistics. doi:10.18653/v1/D18-1306. URL https://www.aclweb.org/anthology/D18-1306.

Hinton, G.; Vinyals, O.; and Dean, J. 2015. Distilling the Knowledge in a Neural Network. In *NIPS Deep Learning and Representation Learning Workshop*. URL http://arxiv.org/abs/1503.02531.

Hovy, E.; Marcus, M.; Palmer, M.; Ramshaw, L.; and Weischedel, R. 2006. OntoNotes: The 90 In *Proceedings of the Human Language Technology Conference of the NAACL, Companion Volume: Short Papers*, NAACL-Short '06, 57–60. USA: Association for Computational Linguistics.

Hu, W.; Lin, Z.; Liu, B.; Tao, C.; Tao, Z.; Ma, J.; Zhao, D.; and Yan, R. 2019. Overcoming Catastrophic Forgetting for Continual Learning via Model Adaptation. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. URL https://openreview.net/forum?id=ryGvcoA5YX.

Lange, M. D.; Aljundi, R.; Masana, M.; Parisot, S.; Jia, X.; Leonardis, A.; Slabaugh, G.; and Tuytelaars, T. 2019. Continual learning: A comparative study on how to defy forgetting in classification tasks.

Li, Z.; and Hoiem, D. 2018. Learning without Forgetting. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 40(12): 2935–2947.

Liu, T.; Ungar, L.; and Sedoc, J. 2019. Continual Learning for Sentence Representations Using Conceptors. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, 3274–3279. Minneapolis, Minnesota: Association for Computational Linguistics. doi:10.18653/v1/N19-1331. URL https://www.aclweb.org/anthology/N19-1331.

Nguyen, N.-V.; Nguyen, T.-L.; Thi, C.-V. N.; Tran, M.-V.; and Ha, Q.-T. 2019. A Character-Level Deep Lifelong Learning Model for Named Entity Recognition in Vietnamese Text. In Nguyen, N. T.; Gaol, F. L.; Hong, T.-P.; and Trawiński, B., eds., *Intelligent Information and Database Systems*, 90–102. Cham: Springer International Publishing. ISBN 978-3-030-14799-0.

Paszke, A.; Gross, S.; Chintala, S.; Chanan, G.; Yang, E.; DeVito, Z.; Lin, Z.; Desmaison, A.; Antiga, L.; and Lerer, A. 2017. Automatic differentiation in PyTorch. In *NIPS-W*.

Rannen, A.; Aljundi, R.; Blaschko, M. B.; and Tuytelaars, T. 2017. Encoder Based Lifelong Learning. In *The IEEE International Conference on Computer Vision (ICCV)*.

Sanh, V.; Debut, L.; Chaumond, J.; and Wolf, T. 2019. DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter. In *NeurIPS EMC2 Workshop*.

Shmelkov, K.; Schmid, C.; and Alahari, K. 2017. Incremental Learning of Object Detectors without Catastrophic Forgetting. In *2017 IEEE International Conference on Computer Vision (ICCV)*, 3420–3429.

Tjong Kim Sang, E. F.; and De Meulder, F. 2003. Introduction to the CoNLL-2003 Shared Task: Language-Independent Named Entity Recognition. In *Proceedings of the Seventh Conference on Natural Language Learning at HLT-NAACL 2003 - Volume 4*, CONLL '03, 142–147. doi:10.3115/1119176.1119195. URL https://doi.org/10.3115/1119176.1119195.

Triguero, I.; García, S.; and Herrera, F. 2015. Self-Labeled Techniques for Semi-Supervised Learning: Taxonomy, Software and Empirical Study. *Knowl. Inf. Syst.* 42(2): 245–284. ISSN 0219-1377. doi:10.1007/s10115-013-0706-y. URL https://doi.org/10.1007/s10115-013-0706-y.

Vintsyuk, T. K. 1968. Speech discrimination by dynamic programming. *Cybernetics* 4(1): 52–57.

Wolf, T.; Debut, L.; Sanh, V.; Chaumond, J.; Delangue, C.; Moi, A.; Cistac, P.; Rault, T.; Louf, R.; Funtowicz, M.; Davison, J.; Shleifer, S.; von Platen, P.; Ma, C.; Jernite, Y.; Plu, J.; Xu, C.; Scao, T. L.; Gugger, S.; Drame, M.; Lhoest, Q.; and Rush, A. M. 2019. HuggingFace's Transformers: State-of-the-art Natural Language Processing. *ArXiv* abs/1910.03771.