Continual Learning of Natural Language Processing Tasks: A Survey

Zixuan Ke  
Bing Liu

Department of Computer Science, University of Illinois at Chicago  
{zke4, liub}@uic.edu

Abstract

Continual learning (CL) is a learning paradigm that emulates the human capability of learning and accumulating knowledge continually without forgetting the previously learned knowledge and also transferring the learned knowledge to help learn new tasks better. This survey presents a comprehensive review and analysis of the recent progress of CL in NLP, which has significant differences from CL in computer vision and machine learning. It covers (1) all CL settings with a taxonomy of existing techniques; (2) catastrophic forgetting (CF) prevention, (3) knowledge transfer (KT), which is particularly important for NLP tasks; and (4) some theory and the hidden challenge of inter-task class separation (ICS). (1), (3) and (4) have not been included in the existing survey. Finally, a list of future directions is discussed.

1 Introduction

The goal of continual learning (CL) is to learn a sequence of tasks incrementally in a neural network \[\text{Chen and Liu, 2018}\]. CL is important in many ways. For example, an LM needs to be updated due to the changing environment, e.g., virus mutations and language evolution. A dialogue system needs to learn new skills. A sentiment analysis system needs to constantly learn to classify opinions about new products. Since re-training a model from scratch is usually very expensive and time consuming, incrementally updating it with the latest data and emerging domains/tasks is critical.

CL has been researched in machine learning (ML), computer vision (CV) and natural language processing (NLP). Since it does incremental learning of tasks, it assumes that once a task \(t\) is learned, its training data \(D_t\) (at least a majority of it) is no longer accessible. In NLP, a task can be an end-task (e.g., text classification, summarization and information extraction), which is usually supervised \[Ke et al., 2021a\].

\[^{1}\text{Preprint. Work in Progress}\]

\[^{2}\text{We refer readers to}\ \text{Chen and Liu, 2018} \text{for the difference between CL and other machine learning paradigms.}\]

\[^{3}\text{It also called domain-adaptive pre-training or pre-finetuning.}\]

\[^{4}\text{Lange et al., 2019}\ \text{is only about TIL, Masana et al., 2020}\ \text{is only about autonomous systems.}\]

\[^{5}\text{Belouadah et al., 2021}\ \text{are only about CIL, Mai et al., 2022}\ \text{is only about online CL. Some cover more settings Parisi et al., 2019}\]\n
\[^{6}\text{Hadsell et al., 2020}\ \text{and van de Ven and Toliason, 2019}\ \text{are only about CL. Some cover more settings Parisi et al., 2019}\]\n
\[^{7}\text{Hayes et al., 2021}\ \text{focuses on biological inspirations and Lesort et al., 2020}\ \text{focuses on autonomous systems.}\]
the existing CL settings to make them more general and up-to-date.

2 Settings and Learning Modes of CL

Different CL settings and learning modes lead to different specific challenges and approaches.

2.1 CL Settings

CL has several settings [van de Ven and Toliis, 2019]. For all settings, it learns a sequence of tasks 1, 2, ..., T, where T is the last task learned. Let \( T = \{1...T\} \). Each task \( t \in T \) has a training set \( D_t = \{(x_i^{(t)}, y_i^{(t)})\}_{i=1}^{N_t} \), where \( x_i^{(t)} \) is a training sample, \( y_i^{(t)} \in \mathcal{Y}_t \), the label space of task \( t \) is its class label and \( N_t \) is the number of samples. Let \( \mathcal{X} \) be the input space of all tasks and \( p(\mathcal{X}_t) \) be the distribution of the input data of task \( t \). For any two tasks \( t \) and \( t' \), we have \( p(\mathcal{X}_t) \neq p(\mathcal{X}_{t'}) \), \( \forall t \neq t' \).

(1) Task-incremental learning (TIL). In TIL, task-identifiers (task-IDs) are available in both training and testing, and classes in the tasks may or may not be disjoint (e.g., one task is to classify different types of animals and another task is to classify different breeds of dogs). Formally, let \( \mathcal{Y}_t \subseteq \mathcal{Y} \) (the label space of all tasks learned so far). The goal of TIL is to learn a function \( f : \mathcal{X} \times T \rightarrow \mathcal{Y} \). In a typical TIL approach, a multi-head configuration is applied, where each task is allocated an exclusive output layer or head. Since the task-ID is given for each test sample, the task-specific model is applied in classification. Since many TIL approaches have achieved no forgetting, the main challenge of TIL is to learn a function \( f : \mathcal{X} \times T \rightarrow \mathcal{Y} \).

(2) Domain-incremental learning (DIL). In this setting, the class labels are assumed to be the same across all tasks (the inputs are from different domains). A sequence of sentiment classification tasks can be regarded as a DIL problem because all tasks have the same three class labels, positive, negative, and neutral. Generation tasks in NLP also form a DIL problem as the LM head has the same number of “classes” (vocabulary tokens). Formally, let \( \mathcal{Y} \) be the set of class labels for every task. The goal of DIL is to learn a function: \( f : \mathcal{X} \times T \rightarrow \mathcal{Y} \). Some existing DIL systems do not use task-ID in testing, i.e., \( f : \mathcal{X} \rightarrow \mathcal{Y} \), and can still work well. This is because (1) the tasks are very similar (e.g., sentiment classification tasks that do not have many conflicts) [Ke et al., 2021b], (2) the tasks are very dissimilar and the domain/task of a test sample can be easily predicted [Madotto et al., 2020, Gururangan et al., 2021], and (3) there is a strong replay module [Jin and Joty, 2022]. To distinguish the two cases, we use DIL (w/ ID) and DIL (w/o ID) to refer to DIL with and without task-ID, respectively.

A typical DIL method uses a single-head configuration, which causes additional task interference in training. While most papers regard DIL as a separate setting, it is really a special case of TIL (see their definitions above). A DIL problem can always be solved with a TIL method. Like TIL, in general task-ID is needed for each test sample because different domains can be different and may even have conflicts. For example, the same sentence “This unit is sucks” is positive for a vacuum cleaner but negative for most other products or domains.

(3) Class-incremental learning (CIL). CIL assumes no task-ID is provided at test time. CIL is mainly used for supervised end-task learning, where each task has a set of non-overlapping classes. Only a single model is built for all classes learned so far. In testing, a test instance from any class may be presented for classification. Formally, we have \( \mathcal{Y}_t \cap \mathcal{Y}_t' = \emptyset, \forall t \neq t' \) and \( \mathcal{Y} = \bigcup_{t=1}^{T} \mathcal{Y}_t \). The goal of CIL is to learn a function \( f : \mathcal{X} \rightarrow \mathcal{Y} \).

A typical CIL approach uses a single-head configuration to cover all classes. CIL is more difficult than TIL and DIL because it has an additional challenge (apart from CF and KT). It is called inter-task class separation (ICS), which is identified recently in [Kim et al., 2022]. ICS means that when learning a new task, since the system does not see the data of the previous tasks, it cannot establish decision boundaries between the classes in the current task and the classes in previous tasks. Given that at test time no task-ID is given, CIL accuracy is often much lower than that of TIL or DIL. We will discuss how this problem may be solved in Sec. 4.3.

Focus of NLP. In ML and CV research, almost only TIL and CIL are studied. However, in NLP, all settings have been used due to NLP’s rich types of tasks. Further, CL research in ML and CV mainly focuses on overcoming CF. In NLP, bi-directional KT is also extremely important. That is because (1) in text, words and phrases in different tasks or domains normally have the same meaning and (2) many NLP tasks are similar and have shared knowledge, e.g., different information extraction tasks and different sentiment classification tasks.

2.2 Learning Modes

Learning modes control how the system is trained. They may be used with any of the aforementioned settings. In the literature, there are two modes:

(1) Offline or Batch CL. When a task arrives, all the training data of the task is available and the training can be done in any number of epochs.

(2) Online CL [Mai et al., 2022]. In this mode, the training data are assumed to come in a data stream. Whenever a small batch of data is accumulated in the stream, it is trained in one iteration. So online CL trains only in one epoch. Existing online CL research mainly solves the CIL problem. Online CL has two task arriving configurations.

(a) Hard task boundary [de Masson d’Autume et al., 2019]. In this configuration, the data of each task comes in a stream continuously without any data in between from any other tasks. That is, the data of another task will only come after the data from its preceding task has been exhausted.

(b) Blurry task boundary [Aljundi et al., 2019]. In this mode, the data from different tasks may intertwine. Thus, the boundaries of tasks are blurry.
We summarize CL formulation of NLP problems in the existing literature in Table 1. Due to the fact that a large number of CL papers have been published, the list is by no means exhaustive.

We observe that a large number of NLP problems belong to DIL. This is very different from CV where only a few problems are formulated as DIL. This is mainly because a language model (LM) is naturally a text generator and recent works on large LMs rely on the intuition that most NLP tasks can be described via natural language instructions (see Appendix A). With the instructions, different types of tasks (e.g., classification or extraction) can be converted to generation tasks and thus a shared LM head is applied across all tasks.

4 Approaches for Continual Learning

In this section, we give a comprehensive review of the existing CL techniques. It is important to keep in mind that different settings have different challenges and thus different approaches. However, since all settings have to deal with the challenges of catastrophic forgetting (CF) and knowledge transfer (KT), it is no surprise that some techniques have been used in different problem settings. We categorize existing systems based on their family of approaches and also give the setting that each system belongs to. Lastly, we introduce the latest challenge of catastrophic forgetting (CF) and knowledge transfer (KT). Lastly, we introduce the latest challenge of catastrophic forgetting (CF) and knowledge transfer (KT).
4.1 Approaches for CF Prevention

The main idea of CF prevention is to reduce the degree of parameter-sharing among tasks or to prevent major changes to previously learned parameters. Since the system also needs to consider plasticity, it has to use different strategies to balance what to retain and what to use for new tasks.

We categorize existing techniques for overcoming CF into regularization-based (Sec. 4.1), replay-based (Sec. 4.1) and parameter-isolation based (Sec. 4.1) families. In addition, we add the instruction based (Sec. 4.1) family specifically for NLP. A summary can be found in Table 2.

Regularization-based Methods

The main idea of this family is to add a penalty or regularization to the loss function to penalize changes to important parameters learned from previous tasks in learning a new task. The main drawback is that it introduces a trade-off between the new task learning and forgetting prevention. This approach was originally proposed in CV [Kirkpatrick et al., 2016], Zenke et al., 2017, He and Jaeger, 2018, Ebrahimi et al., 2020] and has been adopted by the NLP community.

(1). **Regularizing the loss based on parameter importance** [Li et al., 2022a], [Lee, 2017], [Liu et al., 2019a]. The most popular and established regularization-based method in this sub-family is EWC [Kirkpatrick et al., 2016] in CV. Note that updating a pre-trained LM in CL will inevitably result in forgetting of some knowledge in the LM in this sub-family. This is because the data used in the pre-training process is not accessible to end-users of the LM. Then parameter importance is not compute-able by end-users in order to protect the important LM parameters. As a result, many systems fix the LM and train only some additional modules (e.g., adapter or prompt) added to the LM.

(2). **Regularizing the loss based on distillation** [Monaikutul et al., 2021], [Li et al., 2022b], [Liu et al., 2021], [Varshney et al., 2022], [Qin and Joty, 2022]. This sub-family distills the knowledge from a previous model (trained on a previous task) to the model being trained on the new data so that the previous knowledge is not forgotten. The main drawback is that distillation is vulnerable to domain shift between tasks [Aljundi et al., 2016].

Other regularization-based methods. Though existing NLP systems only regularize the loss, CV-based systems also regularize the gradient [Zeng et al., 2019].

Wang et al., 2021b, Mirzadeh et al., 2020, Ebrahimi et al., 2020.

Replay-based Methods

Most replay-based methods either (a) store a small subset of training samples of previous tasks in a memory or (b) learn a data generator to generate pseudo samples of previous tasks. (b) is usually called pseudo/generative replay. In learning a new task, the saved samples or the generated samples and the new task data are both used in training [Rebuffi et al., 2017], [Lopez-Paz and Ranzato, 2017], [Shin et al., 2017], [Kemker and Kanan, 2018].

(1). **Replaying raw samples in training** [Li et al., 2022b], [Liu et al., 2021], [Scialom et al., 2022], [Qin et al., 2022], [Huang et al., 2021]. This sub-family directly applies the case (a) above.

(2). **Replaying raw samples in training and inference** [de Masson d’Autume et al., 2019], [Wang et al., 2020]. Besides joint training of both the replay samples and new samples, this approach also uses the replay samples in inference for local adaptation (re-training the model using the closest replay data to the testing sample).

(3). **Replaying prototypes**. Instead of storing raw samples, one can also save class prototypes [Wang et al., 2021a].

(4). **Optimizing on the replay memory**. Using the replay samples has the risk of over-fitting. This sub-family uses the saved data only to regularize the training, but does not train the replay samples. Representative systems are GEM [Lopez-Paz and Ranzato, 2017] and AGEM [Chaudhry et al., 2019]. Both systems use an inequality constraint to prevent parameter update from increasing the loss of each previous task. The previous task loss is estimated by using samples in the memory.

(5). **Replaying generated pseudo samples**. This is the case (b). Some NLP systems leverage LMs to generate pseudo samples for previous tasks [Zhao et al., 2022], [Varshney et al., 2022], [Sun et al., 2020], [Zhang et al., 2022], [Qin and Joty, 2022]. This approach is becoming popular because an LM is naturally a generator and can generate high-quality pseudo samples. While this is promising, it has been observed that the LM has difficulty to generate samples for some tasks, such as summarization [Qin and Joty, 2022] and aspect sentiment classification [Ke et al., 2021a].

(6). **Replaying features (latent representations)**. Generating high-quality samples can be very challenging. Some CV papers proposed to generate features. Example systems are those in [van de Ven et al., 2020], [Ye and Bors, 2020].

Parameter-isolation Based Methods

**Parameter-isolation** based methods allocate different parameters to different tasks to prevent subsequent tasks from interfering with the previously learned parameters. This family requires task-ID in both training and testing so it is mainly for TIL and DIL settings. This family also usually suffers from the learning capacity problem due to masked neurons or parameters and makes KT challenging. Like regularization-based and replay-based methods, parameter isolation methods are also originated from CV [Rusu et al., 2016], [Yoon et al., 2018], Fernando et al., 2017, Mallya and Lazebnik, 2018.
### Table 2: Families of approaches for CF prevention in different NLP systems. “DP” stands for domain-adaptive pre-training. “w/ ID” and “w/o ID” stand for with and without ID correspondingly.

| CF Prevention | Family (-based) | Sub-family (-based) | Settings | CL Papers |
|---------------|----------------|---------------------|----------|-----------|
| Regularization | Importance | DIL (w/o ID) | RMR-DSE [Li et al., 2022a]; AEWC [Lee, 2017] |
| | Distillation | CIL | SRC [Liu et al., 2019a] |
| | | DIL (w/o ID) | LFPTS [Qin and Jovy, 2022] |
| | | Extender [Monakal et al., 2021] | CFID [Li et al., 2022b]; CID [Liu et al., 2021]; PAGeR [Varshney et al., 2022] |
| | | DIL (w/o ID; DP) | Continual-IF [Scialom et al., 2022]; ELLE [Qin et al., 2022] |
| | Raw sample (Training) | CIL | CFID [Li et al., 2022b]; CID [Liu et al., 2021]; IDBR [Huang et al., 2021] |
| | Raw sample (Training and testing) | CIL (online) | MBPA++ [Ke et al., 2021a]; Ke Masson d’Autume et al., 2019 |
| | | CIL (w/o ID; online) | Meta-MBPA++ [Wang et al., 2020] |
| | Prototype | DIL (w/o ID) | TIL [Wang et al., 2021a] |
| | | | TIL [Zhao et al., 2022] |
| | Pseudo-sample | DIL (w/o ID) | LAMOL [Sun et al., 2020]; ACM [Zhang et al., 2021] |
| | | CIL | LFPTS [Qin and Jovy, 2022]; PAGeR [Varshney et al., 2022] |
| | Task masking | TIL | CTR [Ke et al., 2021a] |
| | | DIL (w/o ID; DP) | CPT [Ke et al., 2021a] |
| | Parameter generation | DIL (w/o ID) | CIL | CLIF [Jin et al., 2021] |
| | | CIL | TIL |
| Dynamic architecture | DIL (w/o ID; DP) | CPT [Zhu et al., 2022]; TPEM [Geng et al., 2021] |
| | | DEMIX [Gururangan et al., 2021] | ELLE [Qin et al., 2022] |
| | | DIL (w/o ID) | AdapterCL [Madotto et al., 2020]; ProgM [Shen et al., 2019] |
| | Instruction | N/A | DIL (w/o ID; online) | ContinuFin [Yin et al., 2022]; CIL Entailment [Xia et al., 2021] |

### Table 3: Families of approaches for bi-directional KT in different NLP systems. “w/ ID” and “w/o ID” stand for with and without ID correspondingly.

| Bi-directional KT | Desiderata | Family (-based) | Sub-family (-based) | Settings | CL Papers |
|-------------------|------------|----------------|---------------------|----------|-----------|
| Importance | DIL (w/ ID) | ACM [Zhang et al., 2022] |
| Feature (CapsNet) | TIL | CTR [Ke et al., 2021a]; Ke [Ke et al., 2021c] |
| Feature (Contrast) | DIL (w/o ID) | CLASSIC [Ke et al., 2020b] |
| Regularization | CIL | IDBR [Huang et al., 2021] |
| Replay | TIL | MeL [Wang et al., 2021a] |
| Meta-Learning | CIL (w/o ID; online) | MBPA++ [Wang et al., 2020]; Ke Masson d’Autume et al., 2019 |
| | MBPA++ [Ke et al., 2021b]; Ke et al., 2022 |
| Instruction | DIL (w/ ID) | C-PT [Zhu et al., 2022]; ContinuFin [Yin et al., 2022] |

#### (1). Fixed architecture. This sub-family isolates a subset of parameters for each task in a fixed network. Three popular methods are task masking, parameter generation and sub-network masking.

#### (i). Task masking. This approach masks a subset of neurons at each layer for a task (identified by task-ID). Since the mask can indicate what neurons have been used by previous tasks, in learning the new task, the system can freeze the used neurons to prevent CF. The most popular system is HAT [Serra et al., 2018] in CV. The details of HAT is in Appendix B.3. In NLP, this task masking mechanism has been mainly used in the adapter layer to prevent CF [Ke et al., 2021a; Ke et al., 2021c; Ke et al., 2021b; Ke et al., 2022].

#### (ii). Parameter generation. This method uses one network to generate the model parameters of another network. Since the generation can be conditioned on the task-ID, it helps mitigate CF. Hypernet [von Oswald et al., 2020] in CV and CLIF [Jin et al., 2021] in NLP take this approach.

#### (iii). Subnetwork masking. This method fixes a network N and trains another mask network M to find a sub-network in N for each task. An example in CV is Sup-Sup [Wortsman et al., 2020].

#### (2). Dynamic architecture [Gururangan et al., 2021]. This approach initializes a pool of parameters and does parameter selection. L2P [Wang et al., 2022] in CV is an example.

### Instruction-based Methods

This family is specific to NLP. It uses task-specific instructions to condition the LM. An instruction is a few words. The
4.2 Approaches for Knowledge Transfer

We now review approaches for knowledge transfer (KT). The idea is that two tasks can transfer knowledge to each other if and only if they are similar. KT is typically achieved by sharing some parameters by similar tasks. Since the similarity is not provided by the task data, this family often uses a proxy to compute task similarity. Depending on the proxy, three sub-families have been proposed in the literature.

1. Importance-proxy. This approach first looks for the previous knowledge that is important for the current task and then trains the detected important knowledge together with the current task. The most naive way is to train a mixing coefficient to detect the importance [Zhang et al., 2022].

2. Feature-proxy. This approach compares the feature from previous task models with the feature for the current task. There are two main ideas.

   i. Capsule Network (CapsNet). A simple capsule network (CapsNet) [Hinton et al., 2011, Sabour et al., 2017] consists of two capsule layers. The first layer stores low-level feature maps, and the second layer generates the classification probability with each capsule corresponding to one class. CapsNet uses a dynamic routing algorithm to make each lower-level capsule send its output to a similar or “agreed” (computed by dot product) higher-level capsule to enable KT. This has been adopted in several NLP systems [Ke et al., 2021a, Ke et al., 2021b].

   ii. Contrastive learning [Ke et al., 2021a]. The idea is to leverage contrastive learning to capture the shared knowledge between different views. If one can generate a view from previous tasks that is similar to the current task, the contrastive loss can capture the shared knowledge and learn a representation for KT to the new task learning.

3. Gradient-proxy. Gradient space is another possible proxy for task similarity computation. The idea is that if the new task’s gradients are sufficiently aligned with an old task’s subspace, the updating of the learned model is likely to improve its performance. However, what is “sufficient” is decided by thresholds which can be inaccurate. This is only used in CV so far [Lin et al., 2022a, Lin et al., 2022b].

Replay-based Methods

In CF prevention, the replay data is used to minimize the backward interference. In contrast, the replay data for KT is used to learn transferable knowledge. Depending on how to achieve it, three sub-families are proposed.

1. Regularization-based [Huang et al., 2021]. Instead of penalizing the change of parameters, regularization for KT disentangles task-shared knowledge and task-specific knowledge.

2. Meta-learning-based [Wang et al., 2021a, de Masson d’Autume et al., 2019]. Meta-learning is known to be able to capture the transferable knowledge as it learns a large set of tasks. The main idea of this family is to leverage this property to maximize transfer and minimize interference. To enable KT, the key is to make the support set contain only the new task while the query set contain examples drawn from all learned tasks. With this meta-objective design, different meta-representations can be trained for continual learning. Details are given in Appendix C.

3. Instruction-based [Zhu et al., 2022, Yin et al., 2022]. Apart from being used to address CF (Sec. 4.1), instructions have also been used to enable KT. The idea is to give an instruction that is similar or related to previous tasks.

4.3 Dealing with Inter-task Class Separation

In Sec. 2.1 we discussed the additional challenge of inter-task class separation (ICS) for CIL (beyond CF and KT) [Kim et al., 2022]. This refers to the difficulty to establish decision boundaries between the classes of the current task and those of the previous tasks in CIL because in learning the current task, the learner does not see the previous task data. [Kim et al., 2022] theoretically shows that good within-task prediction (WP) (predicting only the classes within each task like that in TIL) and OOD detection for each task are necessary and sufficient conditions for strong CIL performances. Based on the theory, this paper proposed a technique that combines a TIL method that can overcome CF and a OOD detection method to build the model for each task. Note that most OOD method can also classify in-distribution data, which is WP. This approach achieves the SOTA performance in CIL. The following approaches are also used to deal with ICS. A summary is in Table 4.

Replay-based methods [Huang et al., 2021, Monaikul et al., 2021, Li et al., 2022b, Liu et al., 2022, Varshney et al., 2022, Qin and Joty, 2022, Wang et al., 2020, de Masson d’Autume et al., 2019, Scialom et al., 2022, Qin et al., 2022] partially deal with the ICS problem because the learner can see some past data, but since the amount of replay data is small, the ICS problem is not solved.

Post-prediction methods [Madotto et al., 2020, Gururangan et al., 2021, Li et al., 2022a]. This is mainly used in DIL systems. Sec. 2.1 mentioned that some DIL systems do not provide task-ID in testing, which thus faces the same ICS problem. However, since the classes of all tasks are the same in DIL, we do not separate the classes, but separate the domains. In this case, we can regard each domain as a virtual class and the same theory still applies. Without dealing with the ICS problem, some heuristic methods are proposed to post-predict the domain for each test sample after training. They may perform well mainly due to the specific problem property (e.g., the domains
Table 4: Families of approaches to solving ICS in CIL (or task separation in DIL) in different NLP systems. “DP” stands for domain-adaptive pre-training. “w/ ID” and “w/o ID” stand for with and without task-ID respectively.

| Challenge                                      | Family (-based) | Settings                                      | CL Papers                                           |
|------------------------------------------------|-----------------|-----------------------------------------------|-----------------------------------------------------|
| Inter-task Class Separation                     | CIL             | Replay                                        | IDBR [Huang et al., 2021]                           |
|                                                |                 |                                               | ExtendNER [Monakul et al., 2021]                    |
|                                                |                 |                                               | CFID [Li et al., 2022]                              |
|                                                |                 |                                               | CID [Liu et al., 2021]                              |
|                                                |                 |                                               | PAGeR [Varshney et al., 2022]                       |
|                                                | DIL (w/o ID)    |                                               | LPT [Qin and Joty, 2022]                            |
|                                                | DIL (w/o ID; online) |                                    | Meta-MBPA++ [Wang et al., 2020]                     |
|                                                | CIL (online)    |                                               | MBPA++ [de Masson d’Autume et al., 2019]            |
|                                                | DIL (w/o ID; DP) |                                               | Continual-10 [Scialom et al., 2022]                |
|                                                |                 |                                               | ELLE [Qin et al., 2022]                             |
| Post-prediction                                 | DIL (w/o ID)    |                                               | AdapterCL [Madotto et al., 2020]                   |
|                                                |                 |                                               | RMR-DSE [LT et al., 2022]                           |
|                                                | DIL (w/o ID; DP) |                                               | DEMIX [Gururangan et al., 2021]                    |

are very dissimilar and easy to differentiate. Continual domain-adaptive pre-training of LMs involving fine-tuning for end-tasks after pre-training without providing domain-ID also belongs to this sub-family.

5 Observations and Future Directions

Due to space limits, CL evaluation is put in Appendix E. This section discusses some observations about existing methods. Based on these observations, we outline some future research directions.

Knowledge transfer is the major issue in TIL. Several TIL methods have achieved no CF. Maximizing KT is still challenging [Ke et al., 2021a; Ke et al., 2021c; Wang et al., 2021a; Zhu et al., 2022]. Although Sec. 4.2 discussed some systems that perform KT, the amount of transfer is still limited. The main issue is that the task sequence can contain a mix of similar and dissimilar tasks (e.g., CAT [Ke et al., 2020a]) and it is hard to detect what tasks are similar and have shared knowledge and what tasks are dissimilar and cannot share. If such a detection is wrong, serious CF can arise, especially for backward transfer. More research is still needed.

CF and ICS are still major issues for CIL and DIL. Although [Kim et al., 2022] offered a approach to deal with ICS through OOD detection, the accuracy results are still far below those from training all tasks/classes together (upper bound). Much more future research is still needed.

Continual domain-adaptive pre-training are still in their infancy. A domain adaptively pre-trained LM can improve end-tasks, but how to maintain the needed general knowledge (GK) already in the LM is challenging because learning one task by fine-tuning the LM can cause serious CF for the GK originally in the LM and make the LM unsuitable for subsequent tasks. Although some initial attempts have been made [Gururangan et al., 2021; Ke et al., 2022; Scialom et al., 2022; Qin et al., 2022], they are very limited.

Scalability is an issue. When a very large number of tasks are learned, the capacity of a network will be a problem. Most CL systems learn a small number of tasks (< 100) and may also expand the network (e.g., [Rusu et al., 2016; Madotto et al., 2020]). Some preliminary works dealing with the capacity issue include distillation [Schwarz et al., 2018], pruning [Hung et al., 2019] and regularization [Serrà et al., 2018], but it is still unclear how the capacity issue affects the performance and how to alleviate the issue effectively.

Limited CL work on Temporal CL. In this case, we have only one task, but the task needs to be constantly updated as the time goes. So far, limited research has been done except some analysis and dataset papers [Luu et al., 2021; Rempel et al., 2014; Dhingra et al., 2022; Loureiro et al., 2022; Röttger and Pierrehumbert, 2021].

6 Conclusion

An AI agent is not truly intelligent if it does not have the continual learning (CL) capability to learn and to accumulate knowledge over time and to use the learned knowledge to enable better future learning. This paper gave a comprehensive survey of the recent advances of CL in NLP and discussed some future directions. We hope that this survey will inspire NLP researchers to design better algorithms.

7 Ethics Statement

This work presents a comprehensive literature survey of the state-of-the-art continual learning research in NLP. As more and more chatbots and other AI agents with natural language understanding capabilities appear in our lives, we believe that leveraging continual learning in NLP will become more and more important. As a survey of the existing literature, we could not see that anyone will be made disadvantaged by this work. All the surveyed papers are peer-reviewed and are publicly available. We do not see any risk in this work.

8 Limitation

While we tried to cover as many systems as possible, the covered systems are by no means exhaustive. Our main effort has been put on searching and reviewing papers published in main NLP, machine learning and computer vision conferences. However, since continual learning has been growing rapidly, many related papers might have been missed. In our future work, we plan to continually update the paper list and introduce the most recent topics/papers on continual learning to the NLP community in a dedicated website.
References

[Aljundi et al., 2016] Rahaf Aljundi, Punarjay Chakravarty, and Tinne Tuytelaars. Expert gate: Lifelong learning with a network of experts. 2016.

[Aljundi et al., 2019] Rahaf Aljundi, Min Lin, Baptiste Goujaud, and Yoshua Bengio. Gradient based sample selection for online continual learning. In Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d’Alché-Buc, Emily B. Fox, and Roman Garnett, editors, NeurIPS, 2019.

[Belouadah et al., 2021] Eden Belouadah, Adrian Popescu, and Ioannis Kanellos. A comprehensive study of class incremental learning algorithms for visual tasks. Neural Networks, 135, 2021.

[Biesialska et al., 2020] Magdalena Biesialska, Katarzyna Biesialka, and Marta R Costa-jussà. Continual lifelong learning in natural language processing: A survey. arXiv preprint arXiv:2012.09823, 2020.

[Chaudhry et al., 2018] Arslan Chaudhry, Puneet K Dokania, Thalaiyasingam Ajanthan, and Philip HS Torr. Riemannian walk for incremental learning: Understanding forgetting and intransigence. In Proceedings of the European Conference on Computer Vision (ECCV), 2018.

[Chaudhry et al., 2019] Arslan Chaudhry, Marc’Aurelio Ranzato, Marcus Rohrbach, and Mohamed Elhoseiny. Efficient lifelong learning with A-GEM. In ICLR, 2019.

[Chen and Liu, 2018] Zhiyuan Chen and Bing Liu. Lifelong machine learning. Synthesis Lectures on Artificial Intelligence and Machine Learning, 12(3), 2018.

[de Masson d’Autume et al., 2019] Cyprien de Masson d’Autume, Sebastian Ruder, Lingpeng Kong, and Dani Yogatama. Episodic memory in lifelong language learning. In Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d’Alché-Buc, Emily B. Fox, and Roman Garnett, editors, Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada, 2019.

[Devlin et al., 2019] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. In Jill Burstein, Christy Doran, and Thamar Solorio, editors, NAACL-HLT, 2019.

[Dhingra et al., 2022] Bhuvan Dhingra, Jeremy R Cole, Julian Martin Eisenschlos, Daniel Gillick, Jacob Eisenstein, and William W Cohen. Time-aware language models as temporal knowledge bases. Transactions of the Association for Computational Linguistics, 10:257–273, 2022.

[Ebrahimi et al., 2020] Sayna Ebrahimi, Mohamed Elhoseiny, Trevor Darrell, and Marcus Rohrbach. Uncertainty-guided continual learning with bayesian neural networks. In 8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020, 2020.

[Efrat and Levy, 2020] Avia Efrat and Omer Levy. The turking test: Can language models understand instructions? CoRR, abs/2010.11982, 2020.

[Fernando et al., 2017] Chrisantha Fernando, Dylan Barmane, Charles Blundell, Yori Zwols, David Ha, Andrei A. Rusu, Alexander Pritzel, and Daan Wierstra. Pathnet: Evolution channels gradient descent in super neural networks. CoRR, 2017.

[Geng et al., 2021] Binzong Geng, Fajie Yuan, Qiancheng Xu, Ying Shen, Ruifeng Xu, and Min Yang. Continual learning for task-oriented dialogue system with iterative network prunning, expanding and masking. In Chengqing Zong, Fei Xia, Wenjie Li, and Roberto Navigli, editors, Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing, ACL/IJCNLP 2021, (Volume 2: Short Papers), Virtual Event, August 1-6, 2021, 2021.

[Gupta et al., 2020] Gunshi Gupta, Karmesh Yadav, and Liam Paull. La-manl: Look-ahead meta learning for continual learning. CoRR, abs/2007.13904, 2020.

[Gururangan et al., 2021] Suchin Gururangan, Mike Lewis, Ari Holtzman, Noah A Smith, and Luke Zettlemoyer. Demix layers: Disentangling domains for modular language modeling. arXiv preprint arXiv:2108.05036, 2021.

[Hadsell et al., 2020] Raia Hadsell, Dushyant Rao, Andrei A Rusu, and Razvan Pascanu. Embracing change: Continual learning in deep neural networks. Trends in cognitive sciences, 24(12), 2020.

[Hayes et al., 2021] Tyler L. Hayes, Giri P. Krishnan, Maxim Bazhenov, Hava T. Siegelmann, Terrence J. Sejnowski, and Christopher Kanan. Replay in Deep Learning: Current Approaches and Missing Biological Elements. Neural Computation, 2021.

[He and Jaeger, 2018] Xu He and Herbert Jaeger. Overcom-
Hugo Larochelle, Alina Beygelzimer, Florence d’Alché-Buc, Emily B. Fox, and Roman Garnett, editors, *NeurIPS*, 2019.

[2021] Jin et al. Xiisen Jin, Bill Yuchen Lin, Mohammad Rostami, and Xiang Ren. Learn continually, generalize rapidly: Lifelong knowledge accumulation for few-shot learning. In *Findings of the Association for Computational Linguistics: EMNLP 2021, Virtual Event / Panta Cana, Dominican Republic, 16-20 November, 2021*.

Ke et al., 2020a Zixuan Ke, Bing Liu, and Xingchang Huang. Continual learning of a mixed sequence of similar and dissimilar tasks. In *NeurIPS*, 2020.

Ke et al., 2020b Zixuan Ke, Bing Liu, Hao Wang, and Lei Shu. Continual learning with knowledge transfer for sentiment classification. In *ECML-PKDD*, 2020.

Ke et al., 2021a Zixuan Ke, Bing Liu, Nianzu Ma, Hu Xu, and Lei Shu. Achieving forgetting prevention and knowledge transfer in continual learning. *Advances in Neural Information Processing Systems*, 34, 2021.

Ke et al., 2021b Zixuan Ke, Bing Liu, Hu Xu, and Lei Shu. Classic: Continual and contrastive learning of aspect sentiment classification tasks. In *EMNLP*, 2021.

Ke et al., 2021c Zixuan Ke, Hu Xu, and Bing Liu. Adapting bert for continual learning of a sequence of aspect sentiment classification tasks. In *NAACL*, 2021.

Ke et al., 2022 Zixuan Ke, Haowei Lin, Yijia Shao, Hu Xu, Lei Shu, and Bing Liu. Continual training of language models for few-shot learning. In *Empirical Methods in Natural Language Processing (EMNLP)*, 2022.

Kemker and Kanan, 2018 Ronald Kemker and Christopher Kanan. FearNet: Brain-Inspired Model for Incremental Learning. In *ICLR*, 2018.

Kim et al., 2022 Gyuhak Kim, Changnan Xiao, Tatsuya Konishi, Zixuan Ke, and Bing Liu. A theoretical study on solving continual learning. *arXiv preprint arXiv:2211.02632*, 2022.

Kirkpatrick et al., 2016 James Kirkpatrick, Razvan Pascanu, Neil C. Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A. Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, Demis Hassabis, Claudia Clopath, Dharshan Kumaran, and Raia Hadsell. Overcoming catastrophic forgetting in neural networks. *CoRR*, 2016.

Lange et al., 2019 Matthias De Lange, Rahaf Aljundi, Marc Masana, and Tinne Tuytelaars. Continual learning: A comparative study on how to defy forgetting in classification tasks. *CoRR*, 2019.

Lee, 2017 Sungjin Lee. Toward continual learning for conversational agents. *CoRR*, abs/1712.09943, 2017.

Lesort et al., 2020 Timothée Lesort, Vincenzo Lomonaco, Andrei Stoian, Davide Maltoni, David Filliat, and Natalia Díaz-Rodríguez. Continual learning for robotics: Definition, framework, learning strategies, opportunities and challenges. *Information fusion*, 58, 2020.

Lester et al., 2021 Brian Lester, Rami Al-Rfou, and Noah Constant. The power of scale for parameter-efficient prompt tuning. In Marie-Francine Moens, Xuanjing Huang, Lucia Specia, and Scott Wen-tau Yih, editors, *EMNLP*, 2021.

Lewis et al., 2020 Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. BART: denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. In Dan Jurafsky, Joyce Chai, Natalie Schluter, and Joel R. Tetreault, editors, *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, ACL 2020, Online*, July 5-10, 2020, 2020.

Li and Liang, 2021 Xiang Lisa Li and Percy Liang. Prefix-tuning: Optimizing continuous prompts for generation. In Chengqing Zong, Fei Xia, Wenjie Li, and Roberto Navigli, editors, *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing, ACL/IJCNLP 2021, (Volume 1: Long Papers), Virtual Event, August 1-6, 2021*, 2021.

Li et al., 2022a Dingcheng Li, Zheng Chen, Eunah Cho, Jie Hao, Xiaohou Liu, Fan Xing, Chenlei Guo, and Yang Liu. Overcoming catastrophic forgetting during domain adaptation of seq2seq language generation. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 2022.

Li et al., 2022b Guodun Li, Yuchen Zhai, Qiangleong Chen, Xing Gao, Ji Zhang, and Yin Zhang. Continual few-shot intent detection. In Nicoletta Calzolari, Chu-Ren Huang, Hansaem Kim, James Pustejovsky, Leo Wanner, Key-Sun Choi, Pum-Mo Ryu, Hsin-Hsi Chen, Lucia Donatelli, Heng Ji, Sadao Kurohashi, Patrizia Paggio, Nirwan Xue, Seokhwan Kim, Younggyun Hahn, Zhong He, Tony Kyungil Lee, Enrico Santus, Francis Bond, and Seung-Hoon Na, editors, *Proceedings of the 29th International Conference on Computational Linguistics, COLING 2022, Gyeongju, Republic of Korea, October 12-17, 2022*, 2022.

Lin et al., 2022a Sen Lin, Li Yang, Deliang Fan, and Junshan Zhang. Beyond not-forgetting: Continual learning with backward knowledge transfer. *NeurIPS*, 2022.

Lin et al., 2022b Sen Lin, Li Yang, Deliang Fan, and Junshan Zhang. TRGP: Trust region gradient projection for continual learning. In *ICLR*, 2022.

Liu et al., 2019a Tianlin Liu, Lyle Ungar, and João Sedoc. Continual learning for sentence representations using conceptors. In Jill Burstein, Christy Doran, and Thamar Solorio, editors, *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, 2019.

Liu et al., 2019b Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Jojhi, Danqi Chen, Omer Levy, Mike
[Zhao et al., 2022] Yingxiu Zhao, Yinhe Zheng, Zhiliang Tian, Chang Gao, Bowen Yu, Haiyang Yu, Yongbin Li, Jian Sun, and Nevin L. Zhang. Prompt conditioned VAE: enhancing generative replay for lifelong learning in task-oriented dialogue. *CoRR*, abs/2210.07783, 2022.

[Zhu et al., 2022] Qi Zhu, Bing Li, Fei Mi, Xiaoyan Zhu, and Minlie Huang. Continual prompt tuning for dialog state tracking. *arXiv preprint arXiv:2203.06654*, 2022.
A NLP Preliminaries

Almost all NLP systems nowadays are based on Transformer \cite{Liu2019Multi-TaskGB, Lewis2020DecodingAN, Devlin2019BERT:AL}. This leads to CL techniques specifically for Transformer. Before discussing the detailed CL approaches, it is necessary to discuss these Transformer based NLP techniques.

Adapter, prefix, and prompt tuning (light-weight fine-tuning). These approaches (a.k.a., light-weight tuning) add a small number of parameters to a pre-trained Transformer language model (LM). Research has shown that only fine-tuning the added parameters (with the LM frozen) can already achieve similar performance to fine-tuning the whole LM. The popular light-weight fine-tuning in CL for NLP include adapter-tuning \cite{Houlsby2019ParameterEfficientTR}, which inserts a fully-connected network to each Transformer layer; prefix-tuning \cite{Li2021PrefixTuning:AL}, which prepends some (a hyper-parameter) tunable prefix vectors to the keys and values of the multi-head attention at every layer; and prompt-tuning \cite{Lester2021PromptTuning:AL} (a.k.a., soft-prompt tuning), which adds a sequence of trainable prompt tokens to the end of the original input sequence. In CL, these light-weight tuning methods make the parameter-isolation based methods (Sec. 4.1) more efficient because only a tiny number of parameters are needed to be saved to prevent CF.

Instruction (hard prompt). Hard prompt or instruction is a short piece of text describing the core concept of the task. Some examples are “listing nouns”, “output the nth word or char” \cite{Efrat2020TowardsAL} (more can be seen in \cite{Mishra2022LeveragingMultimodalCL}). By using instruction, all tasks can be formatted as a response to a natural language input and thus a generation task. In CL, this helps convert all tasks in the same generation format so that an LM can be trained as both a classifier and a sample generator. Different task-specific instructions can be applied for different tasks to prevent CF. Some datasets may also provide similar/positive instruction that can be used for knowledge transfer (KT).

B Details of Systems for CF Prevention

B.1 Regularization-based Methods

(1). Regularizing the loss based on parameter importance.

EWC \cite{Kirkpatrick2016OvercomingCF} uses the gradient of previous tasks as the importance of parameters for the previous tasks, denoted as $F_k^\tau$ (i.e., fisher information matrix, where $k$ is the index of parameters). $F_k^\tau$ is then multiplied with the difference between the old parameters and new parameters. The resulting regularization term is added to the loss to penalize the changes made to important parameters. Several CL systems in NLP are derived from EWC or leverage the idea to penalize changes to important parameters.

DIL (w/o ID). “w/o ID” means without task-ID. RMR-
DSE \cite{Li2022LowRankAdaptiveCL} applies EWC to seq2seq modeling. It converts $F_k^\tau$ to tunable hyper-parameters so that no additional backward pass is needed after training each task. AEW \cite{Lee2017RegularizingTheLoss} employs an improved EWC algorithm \cite{Zenke2017 Nasanet}, where the importance can be computed in an online manner. The main idea is to replace the fisher matrix with an importance estimator so that the additional re-training and point-estimation can be removed.

CIL. SRC \cite{Liu2019Multi-TaskGB} continually updates the sentence encoder with regularization. The penalty is computed via matrix conceptors which capture the corpus-specific features of each corpus.

(2). Regularizing the loss based on distillation.

CIL. ExtendNER \cite{Monaikut2021FewShotNER}, CFID \cite{Li2022ContinualCL}, CID \cite{Liu2021ContinualCL}, and PAGeR \cite{Varshney2022PAGER:AF} apply KL-divergence between the response of the previous model and the current model to achieve distillation.

DIL (w/o ID). LFPT5 \cite{Qin2022LFPT5:AL} applies KL-divergence between the saved old prompt and the new prompt.

Other regularization-based methods in CV.

For example, in learning a new task, OWM \cite{Zeng2019OnlineCF}, projects the gradient in the orthogonal direction to the input of the old tasks. \cite{Wang2021ContinualCL} maps the gradient to the null space of the previous tasks to mitigate CF. Some others regularize the learning rate to slow down parameter updates or control the geometry of the local minima \cite{Ebrahimi2020LearningCF,Mirzadeh2020LearningCF}.

B.2 Replay-based Methods

(1). Replaying raw samples in training.

CIL. CFID \cite{Li2022ContinualCL} replays raw samples with a dynamic weighting to control the sample weight. CID \cite{Liu2021ContinualCL} further considers the imbalance between the rich new data and the small old data. CID also leverages the idea in the imbalanced learning community (Inter-class margin loss and cosine normalization) to alleviate the data imbalance issue in replay.

DIL (w/o ID; DP). “DP” refers to domain-adaptive pre-training. Continual-T0 \cite{Scialom2022ContinualCL} saves 1% of the previous task samples in the memory buffer. It can preserve previous knowledge well and even preserve the zero-shot performance of the T0 model. ELLE \cite{Qin2022ContinualCL} uses a large memory buffer (around 1G per task).

CIL. IDBR \cite{Huang2021IDBR} proposes a memory selection rule based on K-means to store as fewer samples as possible. That is, it saves only examples closest to each cluster centroid.

(2). Replaying raw samples in training and inference.

CIL (Online) and DIL (w/o ID; online). MBPA++ \cite{deMassonAutume2019MBPA++} and Meta-MBPA++ \cite{Wang2020ContinualCL} selectively use the saved old samples in inference. During inference, the representation of a test sample is used to retrieve $K$ nearest saved samples in the replay memory. Based on the $K$ samples, gradient updates are performed to achieve sample-specific fine-tuning. The fine-tuned network is then used to output the final prediction for the test sample.

(3). Replaying prototypes.

TIL. MeLL \cite{Wang2021ContinualCL} stores class prototype representations for meta-learning (Sec. 2.2). Both the stored prototypes and current task feature representations are used to prevent CF.


(4). Optimizing on the replay memory. Details are given in main text.
(5). Replaying generated pseudo samples.
TIL. PCLL [Zhao et al., 2022] uses prompt and conditional variational autoencoder (CVAE) to generate previous samples. It saves the task-specific prompt for each task so that the saved previous task prompt can be used to help condition the LM when generating previous task samples.
DIL (w/o ID). “w/ ID” refers to with task-ID. LAMOL [Sun et al., 2020] and ACM [Zhang et al., 2022] convert all tasks into generation tasks following [McCann et al., 2018] and train the underlying LM (GPT-2) so that it can do both task learning and sample generation. When a new task comes, LAMOL or ACM first leverages the LM to generate old task samples and then train the “old” and the new samples together to avoid CF. Different from LAMOL, ACM makes use of adapters and selectively shares them to further enable KT (Sec. 4.2).
CIL. PAGer [Varshney et al., 2022] replays the generated samples by using instructions (hard prompts) containing both current and previous intent words.
DIL. Similarly, LFPT5 [Qin and Joty, 2022] leverages prompt and T5 to achieve both task learning and sample generation. Recall that LFPT5 also has the prompt distillation (Sec. 4.1) to prevent forgetting, so LFPT5 is a combination of regularization-based and replay-based method.

B.3 Parameter-isolation Based Methods
(1). Fixed architecture.
(i). Task masking.
HAT initializes a task embedding $e^{(t)}_t$ for each task $t$ and each layer $l$ in the network. A sigmoid function is used as a pseudo-gate/step function along with a large positive number $s$ (hyper-parameter). A mask $m^{(t)}_l$ is given by $m^{(t)}_l = \sigma(s e^{(t)}_t)$. During training, the mask is element-wise multiplied with the output of each layer $l$. In backward propagation, HAT blocks the used neurons (indicated by the mask) by previous tasks via multiplication of the inverse of the mask with the gradient to avoid CF.

For a couple of NLP systems adopt this idea, including TIL. CTR [Ke et al., 2021a] and B-CL [Ke et al., 2021c].
DIL (w/o ID; DP). CPT [Ke et al., 2022].
DIL (w/o ID). CLASSIC [Ke et al., 2021b].
(ii). Parameter generation.
Hypernet [von Oswald et al., 2020] builds an generated network $g$, which takes the task representation $z$ as input to generate the model parameters for a task solver network $f$. Since the generated network $g$ itself is exposed to CF, Hypernet further imposes a regularization (it is thus also a regularization-based based method) to penalize the changes to the weights of $g$.

TIL. CLIF [Jin et al., 2021] adopts this idea and uses Hypernet to generate parameters for the adapters.
(iii). Subnetwork masking.
SupSup [Wortsman et al., 2020] trains a binary mask network via a “straight-through” trick for each task so that different tasks are totally isolated from each other. However, its subnetwork or mask potentially has the same size as the original model. Extra training and saving tricks are needed to make it efficient.
(2). Dynamic architecture.
DIL (w/ ID). C-PT [Zhu et al., 2022] adds a prompt for each task. TPEM [Geng et al., 2021] expands the network whenever a new task comes.
DIL (w/o ID). AdapterCL [Madotto et al., 2020] adds one set of adapters for each task. It further infers the task-id in testing via perplexity.
DIL (w/o ID; DP). DEMIX [Gururangan et al., 2021] adds adapters when a new task comes. In testing, the prediction of a test instance is given by the weighted sum (according to perplexity) of all pre-trained adapters. ELLE [Qin et al., 2022] expands layers whenever a new task comes. It uses a large replay data (Sec. 4.1) to ensure that the LM can deal with all previous and current tasks.
CIL. ProgM [Shen et al., 2019] expands the network for each new task and also transfers all the learned knowledge to the expanded component so that it can directly use only the last expanded component in testing.

B.4 Instruction-CF Based Methods
DIL (w/ ID). ConTinTin [Yin et al., 2022] uses different instructions for different tasks to condition the LM. ENTAILMENT [Xia et al., 2021] inputs both the sentence and intention label words as instruction and converts intent classification to binary classification, which helps alleviate forgetting.

C Details of Systems for Knowledge Transfer
C.1 Similarity-based Methods
(1). Importance-proxy.
DIL (w/ ID). ACM [Zhang et al., 2022], which learns an adapter for each task and trains a mixing coefficient to detect which adapter can be reused by a new task. ACM prevents CF via a pseudo-replay method.
(2). Feature-proxy.
(a) Capsule Network (CapsNet).
TIL. CTR [Ke et al., 2021a] and B-CL [Ke et al., 2021c] are based on this idea. We know that they both use a parameter-isolation method to avoid CF. They further compare the task similarity via CapsNet. In B-CL and CTR, each task uses a capsule in the lower level (called “task capsule”). The system leverages the routing algorithm to group similar tasks and their shareable features: if two tasks are similar, the capsules corresponding to the tasks are allowed to update (in CTR, there is a binary gate. In B-CL, a weight based on similarity is employed to control how much can be updated) so that knowledge transfer (KT) can be achieved. Note that the parameter-isolation based method like HAT (Sec. 4.1) has parameter sharing (but fixed) across tasks, this may cause CF for dissimilar tasks that share parameters with those similar tasks.
(b) Contrastive learning.
DIL (w/o ID). CLASSIC [Ke et al., 2021a] uses this idea. Again, CLASSIC prevents forgetting via a parameter-isolation method (Sec. 4.1). It encourages KT via contrastive learning. To this end, CLASSIC generates an augmented view from all previous tasks via self-attention and uses contrastive loss to help learn the shared knowledge.

C.2 Replay-based Methods

(1) Regularization-based

CIL. IDBR [Huang et al., 2021]. We have seen that IDBR leverages a replay memory to prevent CF (Sec. 4.1). To encourage the LM to learn task general knowledge, it further regularizes the loss by adding a next sentence prediction task.

(2) Meta-learning-based Methods

In meta-learning, there are two stages. One is meta-training stage where the goal is to learn good representation that can quickly adapt to new tasks. The other one is meta-testing stage where the goal is fine-tuned (usually few-shot) the meta-trained model to do well on the target tasks. In meta-training stage, a set of source tasks is defined and each task has its training and validation data, called support and query set correspondingly. Note that support and query set have their own training and testing set because its goal is learn general representation for the meta-testing. In meta-testing stage, there is another set of tasks which are the target tasks.

By defining the source task as the new task data and the query tasks as all data in the old and new tasks, forgetting prevention and knowledge transfer are naturally embedded in the loss of meta-learning [Riener et al., 2019; Gupta et al., 2020]. Since the query sets need to contain the old tasks data, meta-learning-based methods are always replay-based methods.

Specifically, meta-learning can be viewed as a bi-level optimization problem as follows:

$$w^* = \arg\min_w \sum_{i=1}^{M} \mathcal{L}_{\text{meta}}(\theta^*(i)(w), w, \mathcal{D}_{\text{query}}(i))$$

s.t. $\theta^*(i)(w) = \arg\min_{\theta} \mathcal{L}_{\text{task}}(\theta, w, \mathcal{D}_{\text{support}}(i))$

where $\theta$ is the network parameter and $w$ is the learning strategy of $\theta$. While $w$ can be anything that affect $\theta$ (e.g., the learning rate), a typical $w$ is the initialized parameters. $w$ is the key in meta-learning because it specifies “how to learn” $\theta$ and it is called meta-representation.

While there is “task” in CL and meta-learning, they are very different. To distinct them, we call the task in meta-learning the meta-task and the task in CL the CL-task. In continual learning, the set of meta-tasks are usually defined as the data in each batch. For example, current CL-task data in each batch is used as support sets while the combination of current CL-task data and memory data in each batch is used as query sets. One exception is the online CL methods where usually each sample is used as a meta-task.

This idea has been used in several NLP systems.

TIL. McL [Wang et al., 2021a] has a memory network ($\theta$) and an LM ($w$). The memory contains all the learned class prototype representations. Its output is concatenated with the output of the LM and fed into a fusion network to perform classification. When training the support set, the current CL-task’s class prototypes in the memory network are updated while LM is fixed. When training on the query set, the LM is fine-tuned while network is fixed. As a result, the LM learns the general/shared information (for KT) and the memory network learns the task specific information (for dealing with CF).

DIL (w/o ID; Online) and CIL (Online). Meta- MBPA++ [Wang et al., 2020] uses an additional network as $\theta$ in the inner loop to retrieve samples from the replay memory. An LM as text encoder is used as $w$ to generalize well across all tasks in the outer loop.

(3) Instruction-based Methods

DIL (w/ ID), C-PT [Zhu et al., 2022] fuses the previous instruction (the “query” in a state tracking task) to enable the model to learn a general skill.

TIL. ConTinTin [Yin et al., 2022] uses a task-specific instruction to prevent CF (Sec. 4.1), and further uses the dataset provided positive instructions/samples to enable knowledge transfer (KT).

D Details of Systems for Inter-task Class Separation

Replay-based methods [Huang et al., 2021; Monaikul et al., 2021; Li et al., 2022a; Liu et al., 2021; Varshney et al., 2022; Qin and Joty, 2022; Wang et al., 2020; de Masson d’Autume et al., 2019; Scialom et al., 2022; Qin et al., 2022] can help deal with ICS to some extent because it allows the model to see a limited number of past data. Below are some post-prediction methods used in DIL systems.

DIL (w/o ID), AdapterCL [Madotto et al., 2020] compares the perplexity of each task model (they have previous task models because they are parameter-isolation based methods) using the test samples. The lowest perplexity model will be used to conduct the inference. RMR-DSE [Li et al., 2022a] compare the feature similarity between test samples and a set of domain-shifting vectors (representative vector extracted from the difference between past model representation and current model representation).

DIL (w/o ID; DP), DEMIX [Gururangan et al., 2021] uses perplexity to predict task-ID, similar to AdapterCL.

E Continual Learning Evaluation

CL evaluation mainly assesses (1) the average performance of all learned tasks, (2) the rate of forgetting and (3) the effect of knowledge transfer. Below, we present some reference baselines and evaluation metrics.

E.1 Reference Baselines

Several reference (or control) baselines are commonly used as the lower or upper bounds for CL.

Naive continual learning (NCL). This baseline refers to a system that performs CL without any mechanism to deal with forgetting [Ke et al., 2021a; Ke et al., 2021c; Qin and Joty, 2022]. All parameters can be freely updated in learning the new task. This baseline is expected to have the
worst CF and the base performance for KT, and thus is often regarded as the lower bound of a CL system.

**Train a separate model for each task (ONE).** Opposite to NCL, ONE trains each task separately as an independent model [Ke et al., 2021a; Ke et al., 2021c; Wang et al., 2021a]. It is thus not a CL setting and has no CF or KT. It is usually regarded as the control to assess whether a CL system has KT or CF.

**Multi-task learning (MTL).** MTL is another non-CL reference baseline that is widely regarded as the upper bound of TIL. It requires a multi-head setup.

**Aggregate.** For those systems using a single-head setup or CIL, this reference baseline learns the classes of all tasks as a single learning task. This is regarded as the upper bound for DIL, and CIL.

### E.2 Evaluation Metrics

**Average Accuracy** [Chaudhry et al., 2018] \((A_T)\) measures the performance of a CL method after all \(T\) tasks have been learned, i.e., \(A_T = \frac{1}{T} \sum_{t=1}^{T} a_{T,t}\), where \(a_{T,t}\) refers to the performance of the model on the testing set of task \(t\) after the model has continually trained all \(T\) tasks.

**Average Incremental Accuracy** [Rebuffi et al., 2017; Lopez-Paz and Ranzato, 2017] is a metric in CIL and derived from average accuracy. It is simply the average over the average accuracy of each task (a set of classes in CIL). One can also choose to curve the average accuracy of each task in a figure rather than giving a single number of average incremental accuracy.

**Forgetting Rate** [Chaudhry et al., 2018] \((F_T)\) measures how much knowledge has been forgotten across the first \(T - 1\) tasks, i.e., \(F_T = \frac{1}{T-1} \sum_{t=1}^{T-1} (a_{t,t} - a_{T,t})\), where \(a_{t,t}\) is the test accuracy task \(t\) right after it is learned, and \(a_{T,t}\) is the accuracy of task \(t\) after training the last task \(T\). We average over all trained tasks except the last one as the last task has no forgetting. The higher the forgetting rate is, the more forgetting it has. Negative rates indicate positive knowledge transfer.

**Forward Transfer** [Ke et al., 2020a] \((FWT_T)\) measures how much forward transfer has happened to a new task after learning the task, i.e., \(FWT_T = \frac{1}{T} \sum_{t=1}^{T} (a_{t,t} - a_{0,t})\), where \(a_{0,t}\) refers to the performance of training task \(t\) individually (i.e., the accuracy of ONE for the task). Note that there is another way to measure forward transfer. It measures whether a learned network contains some useful knowledge for the new task [Lopez-Paz and Ranzato, 2017], i.e., \(FWT_T' = \frac{1}{T} \sum_{t=2}^{T} (a_{t-1,t} - r_t)\), where \(r_t\) refers to the accuracy of task \(t\) using a randomly initialized network. Unlike \(FWT_T\), \(FWT_T'\) does not tell how much forward transfer actually happens after learning the new task.

**Backward Transfer** \((BWT_T)\) measures how much the continual learning of subsequent tasks influences the performance of a learned task, i.e., \(BWT_T = \frac{1}{T} \sum_{t=1}^{T-1} (a_{T,t} - a_{t,t})\). Positive \(BWT_T\) indicates the subsequent task learning can improve the performance of previous tasks.