`ConTinTin`: Continual Learning from Task Instructions

Wenpeng Yin[*]
Temple University
wenpeng.yin@temple.edu

Jia Li
Salesforce Research
jia.li@salesforce.com

Caiming Xiong
Salesforce Research
cxiong@salesforce.com

Abstract

The mainstream machine learning paradigms for NLP often work with two underlying presumptions. First, the target task is predefined and static; a system merely needs to learn to solve it exclusively. Second, the supervision of a task mainly comes from a set of labeled examples. A question arises: how to build a system that can keep learning new tasks from their instructions?

This work defines a new learning paradigm `ConTinTin` (**Contin**ual Learning from **T**ask **In**structions), in which a system should learn a sequence of new tasks one by one, each task is explained by a piece of textual instruction. The system is required to (i) generate the expected outputs of a new task by learning from its instruction, (ii) transfer the knowledge acquired from upstream tasks to help solve downstream tasks (i.e., forward-transfer), and (iii) retain or even improve the performance on earlier tasks after learning new tasks (i.e., backward-transfer). This new problem is studied on a stream of more than 60 tasks, each equipped with an instruction. Technically, our method `InstructionSpeak` contains two strategies that make full use of task instructions to improve forward-transfer and backward-transfer: one is to learn from negative outputs, the other is to re-visit instructions of previous tasks. To our knowledge, this is the first time to study `ConTinTin` in NLP. In addition to the problem formulation and our promising approach, this work also contributes to providing rich analyses for the community to better understand this novel learning problem.

1 Introduction

The main goal of machine learning algorithms lies in seeking supervision for solving a target task. Traditionally, the supervision is extracted from a set of labeled examples. The learner constructs a decision function that generalizes beyond the seen examples. While this paradigm has been tremendously successful for many NLP problems, an inherent drawback exists in it: the learner can only be as good as the provided data (Goldwasser and Roth, 2014). Learning, therefore, relies on annotating a large volume of training data, an expensive and time-consuming process. To alleviate the costly demand for task-specific annotation (referred as $\mathcal{S}_0$ hereafter), the human learning process suggests at least two sources of alternative supervision: one is to accumulate knowledge from tasks learned in the past ($\mathcal{S}_1$) (Richard, 1970; Thrun and Mitchell, 1995; Chomsky, 2002); the other is to learn from natural instructions ($\mathcal{S}_2$) describing a high-level story about target tasks (Goldwasser and Roth, 2014). Unfortunately, we rarely see the joint power of $\mathcal{S}_1$ and $\mathcal{S}_2$.

In this work, we present a new learning paradigm `ConTinTin` – **contin**ual Learning from **t**ask **in**structions. In `ConTinTin`, each task is given an instruction describing the target concept directly and a few instances exemplifying it. The system is required to incrementally learn a stream of tasks, so that the knowledge gained in the past can be used to address subsequent tasks. Apparently, this new problem tries to integrate the $\mathcal{S}_1$ and $\mathcal{S}_2$ into a single learning paradigm while decreasing the necessity of $\mathcal{S}_0$. More specifically, `ConTinTin` is expected to carry the properties listed in Table 1.

Our data set is restructured from the NATURAL-INSTRUCTIONS (Mishra et al., 2021). NATURAL-INSTRUCTIONS is a benchmark that studies if a model can make appropriate use of natural language instructions to answer inputs accordingly. It comprises 61 tasks; each task is associated with a piece of instruction consisting of `Title`, `Definition`, `Caution`, `Prompt`, `Things to avoid`, `Examples`, etc. NATURAL-INSTRUCTIONS originally focuses on conventional supervised learning: give a bunch of tasks out of the 61 as the training tasks, and evalu-

[*]Work was done at Salesforce Research.

Item	Explanation
Instruction-driven supervision	Each task is explained by an instruction and a couple of instances exemplifying it.
Fixed model capacity	The system's structure and parameter size are constant regardless of its learning status.
Knowledge maintenance	The system is not inclined to catastrophic forgetting.
Forward transfer	The system uses knowledge acquired from upstream tasks to help solve downstream tasks.
Backward transfer	The system uses knowledge acquired from downstream tasks to help solve upstream tasks.

Table 1: Desiderata of ConTinTin, inspired by (Biesialska et al., 2020).

ate the remaining tasks in a batch. In order to fit the formulation of ConTinTin, we reorganize the 61 tasks in NATURAL-INSTRUCTIONS: a few tasks (e.g., size k) out of the 61 act as training tasks, and the remaining $61 - k$ tasks as an ordered list of new tasks. The learner is expected to first learn from the k training tasks about how to use instructions to solve problems; then it evolves task by task along with the new task chain.

Our system InstructionSpeak is based on BART (Lewis et al., 2020) with two proposed strategies aiming at making the best use of instructions. The first strategy, "NEGATIVE TRAINING", makes use of unfavorable clues, such as Things to avoid, from the instruction to promote the task understanding and forward-transfer. The second strategy, "HISTORY TRAINING", revisits instructions of earlier tasks during continual learning to alleviate the catastrophic forgetting issue in backward-transfer. We evaluate InstructionSpeak on a wide range of transferring distances (from 1 to 40), which shows that InstructionSpeak can generally help both forward-transfer and backward-transfer.[1]

Overall, this work has made three-fold contributions. First, ConTinTin is the first time to be formulated and studied in the NLP community. Second, we propose InstructionSpeak, a promising approach to ConTinTin. Third, we conduct intensive analyses, aiming to give a better understanding of this new challenge.

2 Related Work

This section retrospects *continual learning* and *learning from task instructions*, two machine learning paradigms that try to explore supervisions $\mathcal{S}_1$ and $\mathcal{S}_2$, respectively.

Continual learning. Since the advent of continual learning[2] (Thrun and Mitchell, 1995), this learning problem was mainly studied in computer vision or robotics domains, and most work concentrated on mitigating catastrophic forgetting (McCloskey and Cohen, 1989; Serrà et al., 2018; Hofmanninger et al., 2020). Continual learning can be summarized into three categories: class continual learning (CCL), domain continual learning (DCL), and task continual learning (TCL).

CCL learns a sequence of classes (e.g., visual object categories, text labels, etc.) to build one overall multi-label classifier for all the classes seen so far (Yan et al., 2021). For example, Wang et al. (2019) studied incrementally learning new relations for two entity mentions in an input sentence, and each relation has many labeled examples. Xia et al. (2021) proposed few-shot CCL in which multi-round of new text tags (e.g., intents or relations expressed in the input text) are encountered sequentially, and each new tag is only accompanied by a couple of examples.

DCL essentially studies the same task but in different domains. The system is expected to evolve along with learning from a stream of datasets of the same task and different data distributions. Typical work in NLP includes sentiment classification (Chen et al., 2015; Xia et al., 2017), conversational agents (Lee, 2017), text classification, and question answering (d'Autume et al., 2019), etc.

TCL tries to learn distinct tasks sequentially. Systems in (Sun et al., 2020a,b) incrementally learned among five disparate NLP tasks. Jin et al. (2021) further extended the size of the task stream (one benchmark has 26 tasks, the other covers 55) and studied TCL in a few-shot scenario. It is worth mentioning that all the listed work in TCL consistently transformed all tasks into question answering format (as pointed out in (McCann et al., 2018), many NLP tasks can be formulated as question answering), thus TCL in these literature was actually converted into DCL.

Similar with (Xia et al., 2021; Jin et al., 2021), our work also focuses on low-resource continual learning; in contrast, our learning problem belongs

[1] "Transferring distance" refers to the task numbers between the model at a new status and the model at an earlier status.

[2] Continual learning in the literature is also referred to as: lifelong learning (Silver and Mercer, 2002), incremental learning (Solomonoff, 1989), sequential learning (McCloskey and Cohen, 1989), and never-ending learning (Carlson et al., 2010).

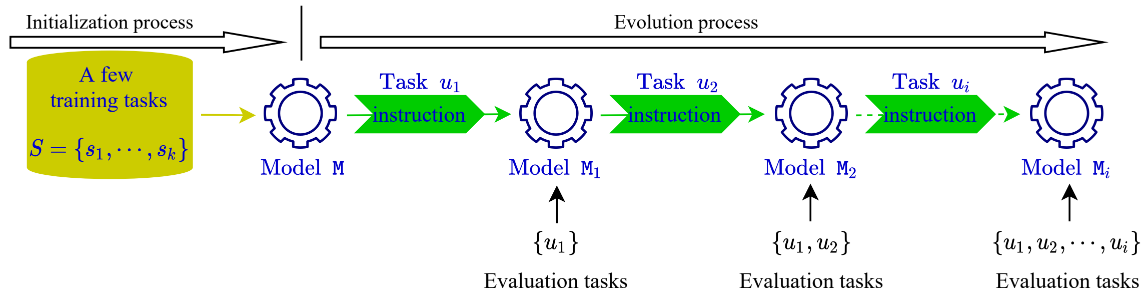


Figure 1: The setup in ConTinTin. The whole learning process consists of two stages: *initialization process* and *evolution process*. A few training tasks $S = \{s_1, \cdots, s_k\}$ equipped with instructions and labeled examples are adopted to initialize the model M, then M incrementally learns from each unseen task u_i by its instruction only. Once finishing the continual learning on the task u_i, the model $\mathtt{M}_i$ is expected to be able to evaluate on all $\{u_1, u_2, \cdots, u_i\}$.

to TCL while *each task in our formulation is expressed by instructions instead of labeled examples*.

Learning from textual instructions. This learning paradigm was first presented by Goldwasser and Roth (2014). They investigated the challenges on Solitaire card game where an instruction is a short sentence such as "*you can move any top card to a free cell if it is empty*", then this instruction is mapped into logical expression via semantic parsing so that an automated agent can understand and execute the instruction.

More recent work tried to examine the ability of large-scale pretrained language models to follow natural language instructions of varying complexity. For example, Efrat and Levy (2020) tested GPT-2 (Radford et al., 2019) to understand instructions like "*listing nouns*", "*output the nth word or char*" and real-world MTurk instructions to annotate some popular datasets. They concluded that GPT-2 works poorly when the supervision comes from those instructions. A dominant instruction format nowadays is called "prompt" which mostly is a short piece of text describing the core concept of the task. Representative work includes (Radford et al., 2019; Schick and Schütze, 2020, 2021), etc. (Please refer to the survey (Liu et al., 2021) for more details.)

While these prompt-based results are encouraging, such prompts are often too simplistic, whereas many real NLP problems cannot be effectively formulated as short prompts or a few positive examples. Motivated, Mishra et al. (2021) collected more than 60 distinct NLP tasks with real-world MTurk instructions, and claimed that pretrained language models, such as BART and GPT-3 (Brown et al., 2020), benefit from instructions to generalize across tasks.

To our knowledge, the only work somehow resembling ours is (Rostami et al., 2020), in which task descriptions were incorporated into lifelong learning for zero-shot transfer. We differ in three aspects: (i) they focused on robot controlling problems, (ii) their tasks are from a single domain, and (iii) in addition to the associated instruction, they assumed that each task has a large number of labeled examples.

3 Problem formulation

3.1 ConTinTin

A system in our ConTinTin comprises two stages, as illustrated in Figure 1. The first stage describes its starting status before learning the first new task; the second stage describes how it evolve continually with a sequence of instruction-equipped unseen tasks. To make it easier to understand, we first introduce the *evolution process*, then the *initialization process*.

Evolution process. ConTinTin tries to build a model M that is able to deal with unseen tasks (U) appearing consecutively by understanding merely the instruction of each task. We denote the task sequence as $U = [u_1, u_2, \cdots, u_i, \cdots]$. Each task u_i has a piece of textual description d_{u_i}, and a set of evaluation instances $\{(x^j_{u_i}, y^j_{u_i})\}^n_{j=1}$ where $y^j_{u_i}$ is the expected output of the input $x^j_{u_i}$. An example d_{u_i} will be shown in Section 3.3. We denote the model M, having learned $[u_1, \cdots, u_i]$, as $\mathtt{M}_i$. For each task u_i, $\mathtt{M}_i$ is required to generate the output for $x^i_{u_i}$ based on the instruction in d_{u_i}.

Initialization process. How to teach a system some basic knowledge to understand task instruc-

Algorithm 1 Forward-transfer metric calculation

```
Require: The model M, all unseen tasks in $U$, two hyperparameter $m$ and $i$
Ensure: $\overrightarrow{g}_i$
 1: for task $t$ in $U$ do
 2:     for $j < m$ times do
 3:         $k$ = random.randint(1, |U|-$i$);
 4:         sample [$u_1, \cdots, u_{k-1}, u_k, \cdots, u_{k+i-1}$] from $U$-{t};
 5:         $\mathtt{M}_k$ = M evolves over [$u_1, \cdots, u_{k-1}, t$];
 6:         $\mathtt{M}_{k+i}$ = M evolves over [$u_1, \cdots, u_{k+i-1}, t$];
 7:         $\overrightarrow{g}^j_{i,t}$=$\mathtt{M}_{k+i}(t)$ - $\mathtt{M}_k(t)$;
 8:     end for
 9:     $\overrightarrow{g}_{i,t} = \frac{1}{m}\sum_{j=1}^{m}\overrightarrow{g}^j_{i,t}$;
10: end for
11: $\overrightarrow{g}_i = \frac{1}{|U|}\sum_{t\in U}\overrightarrow{g}_{i,t}$
```

Algorithm 2 Backward-transfer metric calculation

```
Require: The model M, all unseen tasks in $U$, two hyperparameter $m$ and $i$
Ensure: $\overleftarrow{g}_i$
 1: for task $t$ in $U$ do
 2:     for $j < m$ times do
 3:         $k$ = random.randint(1, |U|-$i$);
 4:         sample [$u_1, \cdots, u_{k-1}, t, u_{k+1}, \cdots, u_{k+i}$] from $U$;
 5:         $\mathtt{M}_k$ = M evolves over [$u_1, \cdots, u_{k-1}, t$];
 6:         $\mathtt{M}_{k+i}$ = M evolves over [$u_1, \cdots, u_{k+i}$];
 7:         $\overleftarrow{g}^j_{i,t}$=$\mathtt{M}_{k+i}(t)$ - $\mathtt{M}_k(t)$;
 8:     end for
 9:     $\overleftarrow{g}_{i,t} = \frac{1}{m}\sum_{j=1}^{m}\overleftarrow{g}^j_{i,t}$;
10: end for
11: $\overleftarrow{g}_i = \frac{1}{|U|}\sum_{t\in U}\overleftarrow{g}_{i,t}$
```

tions and learn continually? We prepare a few training tasks (S=[$s_1, s_2, \cdots, s_k$]) to equip the machine with the ability to annotate the task instances given instructions. Each training task s_i also has its instruction d_{s_i} and n labeled examples $\{(x^j_{s_i}, y^j_{s_i})\}^n_{j=1}$. Note that here we want to control k to be small; otherwise, if `ConTinTin` requires a large number of training tasks at the initialization stage, there is no point anymore to make use of instructions to alleviate the burden of data annotation.

3.2 Evaluation protocol

Forward-transfer evaluation. For this metric, we attempt to quantify the effectiveness of learning more prior tasks before solving a target task. Intuitively, more prior tasks, better downstream performance. We define metric $\overrightarrow{g}_i$ (hereafter, "$\rightarrow$" refers to forward-transfer and "$\leftarrow$" refers to backward-transfer): the *average gained performance* over all new tasks in U when each of them is learned after $k + i - 1$ previous tasks, compared with learning them merely after $k - 1$ tasks (i is transferring distance). As Algorithm 1 shows, computing $\overrightarrow{g}_i$ needs two loops. First, iterate on all tasks in U, select one task t as (i) the kth task and randomly sample its upstream tasks [$u_1, \cdots, u_{k-1}$] from remaining tasks in U, getting one online learning score $\mathtt{M}_k(t)$, or as (ii) the $(k + i)$th task for another online learning score $\mathtt{M}_{k+i}(t)$. $\mathtt{M}_{k+i}(t)$ - $\mathtt{M}_k(t)$ is one instance of the forward-transfer score, which indicates how much improvement the extra upstream tasks of size i bring to the target task t. For this particular task t, repeat its upstream tasks m times and calculate the average as a final score of t, denoted as $\overrightarrow{g}_{i,t}$. Second, the same procedure is applied to all tasks in U and finally average $\overrightarrow{g}_{i,t}$ over all t to get the $\overrightarrow{g}_i$ value.

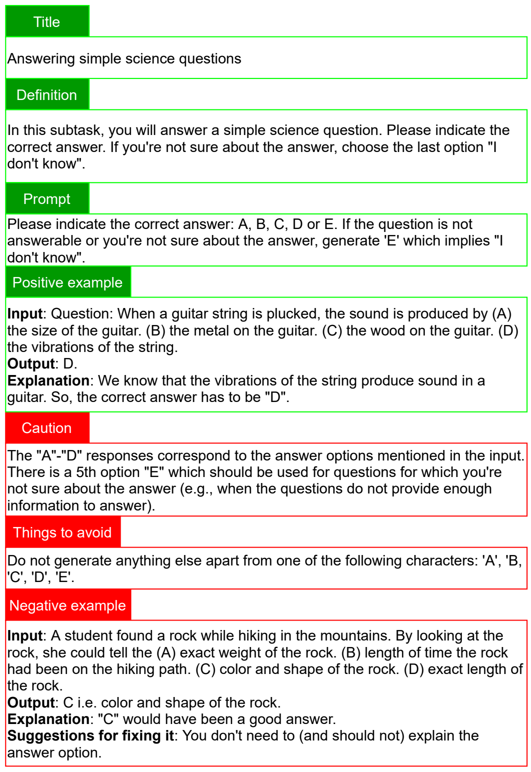
$\overrightarrow{g}_i$ always measures the expected performance gain our system can get when it has continually leaned i more tasks. For forward-transfer, we expect $\overrightarrow{g}_i$ is positive and increases when i gets larger.

Backward-transfer evaluation. In contrast to the forward-transfer evaluation, we define $\overleftarrow{g}_i$ as the backward-transfer metric, which tells how much better our system can handle a task learned i steps ago, compared with the performance on the same the task last time. As Algorithm 2 describes, two loops to calculate $\overleftarrow{g}_i$. Firstly, for a given task t from U, put t in a random position k in the task chain, followed by i other tasks. Subtract its performance when the model M learned it the first time (i.e., $\mathtt{M}_k(t)$) by its performance when the model finished learning all the $k + i$ tasks in the chain (i.e., $\mathtt{M}_{k+i}(t)$). This operation generates a score given this chain; repeat this process m times to get an average gain $\overleftarrow{g}_{i,t}$ for the task t. Secondly, average the $\overleftarrow{g}_{i,t}$ over all t to get the $\overleftarrow{g}_i$ value.

If a system can always make use of downstream tasks to help upstream tasks, $\overleftarrow{g}_i$ should be positive; otherwise, $\overleftarrow{g}_i$ will be negative due to catastrophic forgetting.

3.3 Data

There are no NLP datasets for `ConTinTin` particularly. This work is based on NATURAL-INSTRUCTIONS (Mishra et al., 2021) after data



Title

Answering simple science questions

Definition

In this subtask, you will answer a simple science question. Please indicate the correct answer. If you're not sure about the answer, choose the last option "I don't know".

Prompt

Please indicate the correct answer: A, B, C, D or E. If the question is not answerable or you're not sure about the answer, generate 'E' which implies "I don't know".

Positive example

Input: Question: When a guitar string is plucked, the sound is produced by (A) the size of the guitar. (B) the metal on the guitar. (C) the wood on the guitar. (D) the vibrations of the string.
Output: D.
Explanation: We know that the vibrations of the string produce sound in a guitar. So, the correct answer has to be "D".

Caution

The "A"-"D" responses correspond to the answer options mentioned in the input. There is a 5th option "E" which should be used for questions for which you're not sure about the answer (e.g., when the questions do not provide enough information to answer).

Things to avoid

Do not generate anything else apart from one of the following characters: 'A', 'B, 'C', 'D', 'E'.

Negative example

Input: A student found a rock while hiking in the mountains. By looking at the rock, she could tell the (A) exact weight of the rock. (B) length of time the rock had been on the hiking path. (C) color and shape of the rock. (D) exact length of the rock.
Output: C i.e. color and shape of the rock.
Explanation: "C" would have been a good answer.
Suggestions for fixing it: You don't need to (and should not) explain the answer option.

Figure 2: An instruction example for the task "answering science questions (misc)" in NATURAL-INSTRUCTIONS (Mishra et al., 2021). Green parts present favorable clues while red parts express unfavorable predictions.

reorganization. Next, we first introduce NATURAL-INSTRUCTIONS, then describe our revised version specific to our problem.

NATURAL-INSTRUCTIONS was constructed in the following pipeline: Mishra et al. (2021) first collected some popular NLP benchmarks (e.g., CosmosQA (Huang et al., 2019), Quoref (Dasigi et al., 2019), Winogrande (Sakaguchi et al., 2020), etc.) with their crowdsourcing instructions through engaging with their authors. Since all the crowdsourcing instructions include multiple steps to guide annotators to gather task instances, they further broke raw crowdsourcing instructions down into their individual steps, generating a larger number of subtasks that are minimal and standalone. At last, a total of 61 tasks are obtained, covering six categories: 13 question generation tasks (QG), 16 answer generation tasks (AG), 12 classification tasks (CF), 8 incorrect answer generation tasks (IAG), 10 minimal modification tasks (MM) and 2 verification tasks (VF). An instruction example is presented in Figure 2.

Our data split. For training tasks S, we randomly select k tasks from the 61 tasks. All training tasks in S have instructions and keep their labeled example set. The remaining 61-k tasks are treated as unseen task set U. Each task in U has only instruction; the labeled example set is used for evaluation rather than model training.

It is noteworthy that task order in continual learning should influence the final performance. We do not attempt to release a fixed split of S and U. In experiments, we will randomly generate them multiple times to form different task chains and report the average performance.

4 Our method `InstructionSpeak`

Most prior studies about continual learning focused on backward-transfer (Serrà et al., 2018; d'Autume et al., 2019) while paying less attention to the forward-transfer performance. Next, we introduce our approach to promoting both of them.

The big story of our strategies lies in better understanding of the textual instruction of u_i. Two concrete strategies as follows.

NEGATIVE TRAINING: *to distinguish favorable and unfavorable clues in instructions.* Unfavorable clues, such as the red items in Figure 2, are essential for humans to make decisions while not being successfully leveraged by machine learning. For example, Mishra et al. (2021) found discarding negative examples can even improve the performance. We believe this indicates the approach failed to learn from negative examples rather than those examples being truly useless. Then, how can we make machines extract effective supervision from negative samples?

First, we introduce a method that was tried but did not work well – *minimizing the probability of generating negative output*. Maximizing the probabilities of gold output is widely used in text generation. It sounds intuitive to minimize that for unwanted output, such as (He and Glass, 2020). We tried a joint training on maximizing positive and minimizing negative examples, which is even worse than maximizing the positive alone. Since many "negative" outputs contain tokens that exist in the gold answers, we suspect that minimizing their probabilities will let the model have more difficulty decoding the correct output.

After further study of those negative examples and their explanations, we decide to treat those negative examples as positive and move the negative learning phase as pretraining, i.e., pretrain on negative examples first, then finetune on positive examples. The inspiration comes from the fact that negative examples, despite the tag "negative", can still provide useful information about the expected output. Take a look at the negative example in Figure 2, its output "C i.e., color and shape of the rock" is discouraged just because it does not follow some rules of automatic evaluation rather than it is really wrong. Apparently, as a first step, optimizing the system to generate the so-called "negative output" is still better than any general-purpose pretrained BART.

For each unseen task in U, we directly adopt its negative examples if available. For the k training tasks in S, positive instances (including positive examples in instructions and those labeled task instances) are much more than the negative examples, we use the pretrained model on S to do prediction on all inputs of S, if the output is not equal to the gold output, we treat this (input, predicted output) as a negative example. It means we have a loose definition of what "negative output" is: it is negative once it is not equal to the ground truth. Since the pretrained model on S can already guarantee generation quality, those generated negative outputs are mostly related with the gold outputs (measured by ROUGE metrics).[3]

HISTORY TRAINING: *revisit instructions of previous tasks.* To mitigate catastrophic forgetting, many prior works about continual learning tried to store a couple of labeled examples of upstreaming tasks to replay. In our ConTinTin formulation, each new task is described merely by the instruction. Instead of storing some examples of previous tasks, we keep their instructions. When learning the ith task in U, our model will first learn all the instructions of prior $i-2$ tasks in a batch with a lower learning rate. Revisiting precedent instructions is cost-effective since each instruction is as short as a couple of conventionally annotated examples but with much more supervision.

Overall, our two strategies work jointly to enhance the forward-transfer and the backward-transfer performance. Our system InstructionSpeak is based on BART, treating all tasks as a text-to-text problem. The full input format of encoder: [Input] input string [Title] title string [Prompt] prompt string [Definition] definition string [Avoid] things to avoid string [Caution] caution string [POS1] [Input] input string [Output] output string [Explanation] explanation string $\cdots$ [POSn] [Input] input string [Output] output string [Explanation] explanation string. Note that we put the input at the beginning of this encoder's input template to prevent from being discarded due to long text truncation. When pretrain on training tasks S, the full input pattern is used; when continually learn on U, since the input at the beginning comes from positive or negative examples of the instruction, we do not include the positive examples in the input template (i.e., the blue part is dropped).

Given S and U, the whole learning pipeline in InstructionSpeak is: (i) pretrain on S to get model $\mathtt{M}^*$; (ii) use $\mathtt{M}^*$ to make predictions on S to collect negative example set S^-; (iii) pretrain on S^- and finetune on S to get boosted model $\mathtt{M}$ which is the starting model status for continual learning on U; (iv) for the ith unseen task u_i in U, tune $\mathtt{M}$ on instructions of all earlier tasks $[u_1, \cdots, u_{i-2}]$ in a batch; (v) tune on negative examples of u_i, if available; (vi) tune on positive examples of u_i.

5 Experiments

Setup. We use the pretrained BART-base model released by Huggingface. Hyperparameters: $m = 10$ in Algorithms 1-2; $k = 5$ for the task set S; max input length 1024 tokens, learning rate 5e-5, 3 epochs as suggested by (Mishra et al., 2021) for most phases of training (except for 5e-6 and one epoch for HISTORY TRAINING); batch size 5 for training on S and 2 for continual learning on U. All unseen tasks U randomly select 1k labeled examples for performance evaluation. Note that the official evaluation metric for NATURAL-INSTRUCTIONS is ROUGE-L (Lin, 2004). According to the definitions of our evaluation metrics, $\overrightarrow{g}_i$ and $\overleftarrow{g}_i$ numbers are the same meaning as ROUGE-L.

Baselines. There are no prior systems that can fit the formulation of ConTinTin exactly. In addition, as the ConTinTin properties in Table 1 indicate, ideally, ConTinTin prefers a fixed model capacity. Therefore, we do not compare with systems that incorporate extra memory modules or adaptors, such as (d'Autume et al., 2019; Jin et al.,

[3]We also tried to *build a negative-output generator given available negative examples in instructions*. This type of negative output was planed for pretraining in both S and U. However, due to the tiny size of negative examples in instructions (most tasks have at most 2 negative examples, a couple of them have zero), the learned negative-output generator yields outputs that are over unreasonable.

Method	forward-transfer					backward-transfer				
	$\overrightarrow{g}_1$	$\overrightarrow{g}_{10}$	$\overrightarrow{g}_{20}$	$\overrightarrow{g}_{30}$	$\overrightarrow{g}_{40}$	$\overleftarrow{g}_1$	$\overleftarrow{g}_{10}$	$\overleftarrow{g}_{20}$	$\overleftarrow{g}_{30}$	$\overleftarrow{g}_{40}$
Seq-finetune	1.44 ±7.15	3.28 ±19.46	-3.74 ±8.73	2.9 ±16.42	-0.36 ±17.23	1.57 ±3.28	0.04 ±12.46	-0.19 ±21.75	-6.48 ±19.17	-9.46 ±19.57
LAMOL (Sun et al., 2020a)	-1.34 ±4.46	1.41 ±13.55	3.31 ±14.32	-5.40 ±20.44	-0.03 ±12.68	2.67 ±12.52	2.21 ±7.98	9.42 ±12.88	6.33 ±20.13	7.21 ±14.81
Our `InstructionSpeak`	**2.16** ±6.46	**5.06** ±20.87	2.29 ±18.03	**4.07** ±7.95	**4.39** ±14.56	1.44 ±9.28	**5.21** ±18.20	7.33 ±13.48	**14.99** ±20.21	**12.31** ±16.53
w/o NEGATIVE TRAINING	-2.89 ±13.12	1.06 ±17.21	1.33 ±13.09	2.21 ±14.42	1.78 ±17.90	2.21 ±12.23	3.37 ±13.23	**11.44** ±11.03	10.36 ±21.34	8.94 ±19.41
w/o HISTORY TRAINING	1.88 ±17.73	3.32 ±12.76	**4.41** ±20.24	3.22 ±16.66	2.97 ±14.93	**4.74** ±16.54	-2.78 ±19.38	-0.83 ±12.93	1.35 ±15.95	3.49 ±14.05
Multi-task (upperbound)	7.98±20.47									

Table 2: The main results of `ConTinTin`.

Method		QG	AG	CF	IAG	MM	VF	mean
(Mishra et al., 2021)	paper report	52.xx	30.xx	50.xx	25.xx	47.xx	8.xx	35.33
	reimplement	53.55	17.45	63.79	11.06	82.86	7.40	39.35
Seq-finetune	forward	49.61	21.46	48.74	9.70	57.31	7.61	32.40
	backward	47.09	21.17	7.45	9.61	88.84	**14.98**	31.52
LAMOL	forward	52.23	20.45	67.74	8.81	82.29	8.83	40.05
	backward	52.14	22.76	7.98	8.33	88.45	9.91	31.59
`InstructionSpeak`	w/o CL	51.07	23.40	70.68	**11.43**	88.13	6.22	41.82
	forward	51.30	24.89	**70.96**	9.36	**90.41**	10.70	**42.93**
	backward	53.04	**24.93**	7.51	8.56	88.09	13.86	32.66

Table 3: The results on standard split of NATURAL-INSTRUCTIONS. We use "52.xx" just because the original paper by Mishra et al. (2021) did not report the "xx" numbers.

2021; Ke et al., 2021). The following systems are considered:

- *Seq-finetune*: first pretrain a BART on S, then fine-tune it on U sequentially. It does not pay special attention to catastrophic forgetting.

- *Multi-task*: first pretrain a BART on S, then train on instructions of all tasks in U simultaneously. It, acting as the upperbound of continual learning, does not distinguish between forward-transfer and backward-transfer.

- *LAMOL* (Sun et al., 2020a): A state-of-the-art system that uses pretrained language models for task continual learning. All tasks are converted into QA and a single language model is used for the continual learning; before training on a new task, the language model first generates pseudo-examples for previous tasks; those pseudo-examples are mixed with the examples of the new task to train the language model. The original language model in LAMOL is a smallest pretrained GPT-2, we replace it with BART for a fair comparison.

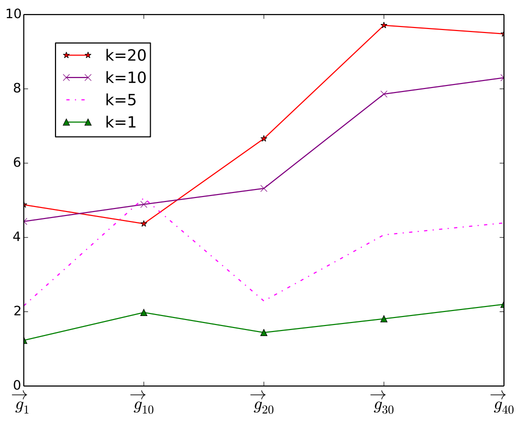
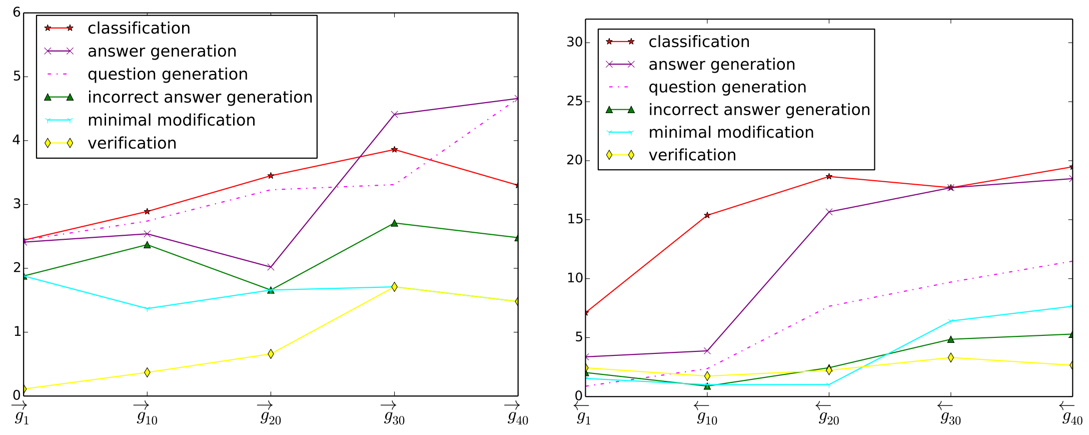


Figure 3: Influence of training task size (i.e., k value in S=[s_1, s_2, $\cdots$, s_k]) on forward-transfer. $k \in \{1, 5, 10, 20\}$. Please note that $k = 5$ is what we used to report Table 2.

5.1 Results

Table 2 shows the comparison between our system `InstructionSpeak` and those baselines. We have three threads of observations.

Firstly, our system `InstructionSpeak` consistently outperforms all baselines for both



(a) Forward-transfer (b) Backward-transfer

Figure 4: Different transferabilities on tasks of six categories.

forward-transfer and backward-transfer evaluations. For forward-transfer, all systems cannot beat the multi-task learning, but in backward-transfer, `InstructionSpeak` even outperforms the multi-task competitor; this is because multi-task learning, though widely treated as upperbound for continual learning, only trained on all U tasks for 3 epochs. Our method, equipped with HISTORY TRAINING, actually learns many times of earlier U tasks during the continual learning. Despite a few exceptions, generally speaking, both the forward and backward transfer performance increase when the transferring distance increases from 1 to 40.

Secondly, the ablation study verifies the effectiveness of our two strategies. NEGATIVE TRAINING plays the leading role in forward-transfer while doing a moderate favor to the backward-transfer. A totally opposite phenomenon is noticed for HISTORY TRAINING: it clearly contributes to the backward-transfer evaluation while influencing the forward-transfer to some extent.

Thirdly, the standard deviations are mostly large; this should be due to the fact that the 61 tasks in NATURAL-INSTRUCTIONS contains 6 distinct categories; each category benefits from the model generalization by different degrees.

To further figure out the exact performance of our system on different task categories, we report on the standard split of NATURAL-INSTRUCTIONS as Mishra et al. (2021) did: they have a fixed set of 12 tasks for testing (2 for each category), and all remaining tasks as training data. Since their 12 test tasks have no order, for each of the test category, we put it as the sixth (resp. first) task in the chain for forward-transfer (resp. backward-transfer). Once the position of the test category is fixed, we randomly order the remaining five categories in the sequence for 10 times and report the average performance. Thus, each test category will have two numbers for every continual learning approach: one for forward-transfer, the other for backward-transfer. In addition, we also report our system `InstructionSpeak` without continual learning (w/o CL), i.e., using the system pretrained on 49 tasks in S to predict.

Table 3 lists the results of all continual learning systems on NATURAL-INSTRUCTIONS. We notice that (i) the results of different task categories vary a lot. For example, minimal modification tasks (MM) easily get ROUGE-L score above 80, but it is pretty challenging to obtain ROUGE-L score over 10 for Verification (VF); (ii) Classification tasks (CF) seem suffering from backward-transfer. We suspect CF is too sensitive to classification-specific supervision, such as label spaces; the continual learning on many subsequent tasks of different categories will mislead the model in solving CF. This is further supported by looking at the results of three systems: `InstructionSpeak` w/o CL, (Mishra et al., 2021) and `InstructionSpeak` forward-transfer. The first two systems start predicting on U once finish the training on S. Note that CF in U has 10 CF tasks in S; it means the first two systems, although they did not learn the CF in U, still obtained enough supervision for this category from S. That's why all three systems get high performance on CF. Once they get tuned on more different categories, the supervision disappears increasingly.

5.2 Analysis

In addition to the results in Tables 2-3, we are further interested in the following two questions.

$\mathcal{Q}_1$**: how many training tasks does a system need to learn from instructions?** Recall that apart from the U in the evolution process, we use k tasks (S=[s_1, s_2, $\cdots$, s_k]) to initialize the model. S can have maximal 20 tasks (due to the limited size of NATURAL-INSTRUCTIONS) and our system only used 5 out of them. Here, we further explore the model's behavior when k varies.

Figure 3 depicts the influence of k on forward-transfer. For forward-transfer, larger k values (i.e., more training tasks to initialize the model) consistently improve performance. We think that more training tasks tend to teach the model better at understanding the task instructions, which can further improve the model's transferability when it learns i more tasks to report $\overrightarrow{g}_i$ on a downstream task u_i. We notice that NATURAL-INSTRUCTIONS v2 [4] has over 1.7k tasks. We leave it as future work to further explore the potential of increasing training tasks.

$\mathcal{Q}_2$**: how do tasks of different categories in U benefit?** In Section 3.3, we mentioned that all tasks can be organized into six categories. We check their separate performances here. Note that both Algorithms 1-2 obtain the final score by averaging over all tasks in U, here we average those tasks that belong to the same category to get category-wise forward-transfer and backward-transfer performances.

From Figure 4(a) and Figure 4(b), we notice that: (i) tasks of distinct categories indeed demonstrate different performances for both forward-transfer and backward-transfer evaluations; (ii) the phenomena on the two evaluations are similar: some categories consistently benefit more, such as "classification", "answer generation", "question generation", while some keep obtaining worse scores, such as "minimal modification" and "verification" categories. We think this discrepancy origins from two factors; one is how many tasks a particular category has, the other is how similar or relevant the tasks in that category are with tasks of other categories. Intuitively, a categories with more tasks occupying the task chain and resembling other tasks, such as "classification", "answer generation" and "question generation", can be easier solved when the model comes up to it or comes back to it.

[4] `https://github.com/allenai/natural-instructions`

6 Conclusions

This work introduced a novel learning problem: continual learning from task instructions. The goal is to explore the potential of exiting pretrained language models in solving new tasks by understanding instructions rather than labeled examples. With our problem formulation and a well-performing system, we pave the way for future study of this challenge in the community.

Acknowledgement

We thank Daniel Khashabi from AI2 and Swaroop Mishra from ASU for help during this work.

References

Magdalena Biesialska, Katarzyna Biesialska, and Marta R. Costa-jussà. 2020. Continual lifelong learning in natural language processing: A survey. In *Proceedings of COLING*, pages 6523–6541.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language models are few-shot learners. In *Proceedings of NeurIPS*.

Andrew Carlson, Justin Betteridge, Bryan Kisiel, Burr Settles, Estevam R. Hruschka Jr., and Tom M. Mitchell. 2010. Toward an architecture for never-ending language learning. In *Proceedings of AAAI*.

Zhiyuan Chen, Nianzu Ma, and Bing Liu. 2015. Lifelong learning for sentiment classification. In *Proceedings of ACL*, pages 750–756.

Noam Chomsky. 2002. *Syntactic structures*.

Pradeep Dasigi, Nelson F. Liu, Ana Marasovic, Noah A. Smith, and Matt Gardner. 2019. Quoref: A reading comprehension dataset with questions requiring coreferential reasoning. In *Proceedings of EMNLP-IJCNLP*, pages 5924–5931.

Cyprien de Masson d'Autume, Sebastian Ruder, Lingpeng Kong, and Dani Yogatama. 2019. Episodic memory in lifelong language learning. In *Proceedings of NeurIPS*, pages 13122–13131.

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

Dan Goldwasser and Dan Roth. 2014. Learning from natural instructions. *Machine Learning*, 94(2):205–232.

Tianxing He and James R. Glass. 2020. Negative training for neural dialogue response generation. In *Proceedings of ACL*, pages 2044–2058. Association for Computational Linguistics.

Johannes Hofmanninger, Matthias Perkonigg, James A. Brink, Oleg Pianykh, Christian Herold, and Georg Langs. 2020. Dynamic memory to alleviate catastrophic forgetting in continuous learning settings. In *Proceedings of MICCAI*, volume 12262, pages 359–368.

Lifu Huang, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. 2019. Cosmos QA: machine reading comprehension with contextual commonsense reasoning. In *Proceedings of EMNLP-IJCNLP*, pages 2391–2401.

Xisen Jin, Bill Yuchen Lin, Mohammad Rostami, and Xiang Ren. 2021. Learn continually, generalize rapidly: Lifelong knowledge accumulation for few-shot learning. In *Proceedings of EMNLP Findings*, pages 714–729.

Zixuan Ke, Hu Xu, and Bing Liu. 2021. Adapting BERT for continual learning of a sequence of aspect sentiment classification tasks. In *Proceedings of NAACL-HLT*, pages 4746–4755.

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

Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. 2020. BART: denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. In *Proceedings of ACL*, pages 7871–7880.

Chin-Yew Lin. 2004. ROUGE: A package for automatic evaluation of summaries. In *Text summarization branches out*, pages 74–81.

Pengfei Liu, Weizhe Yuan, Jinlan Fu, Zhengbao Jiang, Hiroaki Hayashi, and Graham Neubig. 2021. Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing. *CoRR*, abs/2107.13586.

Bryan McCann, Nitish Shirish Keskar, Caiming Xiong, and Richard Socher. 2018. The natural language decathlon: Multitask learning as question answering. *CoRR*, abs/1806.08730.

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

Swaroop Mishra, Daniel Khashabi, Chitta Baral, and Hannaneh Hajishirzi. 2021. Cross-task generalization via natural language crowdsourcing instructions. *CoRR*, abs/2104.08773.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language models are unsupervised multitask learners. *OpenAI blog*.

Montague Richard. 1970. Universal grammar.

Mohammad Rostami, David Isele, and Eric Eaton. 2020. Using task descriptions in lifelong machine learning for improved performance and zero-shot transfer. *Journal of Artificial Intelligence Research*, 67:673–704.

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. 2020. Winogrande: An adversarial winograd schema challenge at scale. In *Proceedings of AAAI*, pages 8732–8740.

Timo Schick and Hinrich Schütze. 2020. Few-shot text generation with pattern-exploiting training. *CoRR*, abs/2012.11926.

Timo Schick and Hinrich Schütze. 2021. Exploiting cloze-questions for few-shot text classification and natural language inference. In *Proceedings of EACL*, pages 255–269.

Joan Serrà, Didac Suris, Marius Miron, and Alexandros Karatzoglou. 2018. Overcoming catastrophic forgetting with hard attention to the task. In *Proceedings of ICML*, volume 80, pages 4555–4564.

Daniel L. Silver and Robert E. Mercer. 2002. The task rehearsal method of life-long learning: Overcoming impoverished data. In *Proceedings of 15th Conference of the Canadian Society for Computational Studies of Intelligence*, volume 2338, pages 90–101.

Ray J. Solomonoff. 1989. A system for incremental learning based on algorithmic probability. In *Proceedings of the Sixth Israeli Conference on Artificial Intelligence, Computer Vision and Pattern Recognition*, pages 515–527.

Fan-Keng Sun, Cheng-Hao Ho, and Hung-Yi Lee. 2020a. LAMOL: language modeling for lifelong language learning. In *Proceedings of ICLR*.

Jingyuan Sun, Shaonan Wang, Jiajun Zhang, and Chengqing Zong. 2020b. Distill and replay for continual language learning. In *Proceedings of COLING*, pages 3569–3579.

Sebastian Thrun and Tom M. Mitchell. 1995. Lifelong robot learning. *Robotics and Autonomous Systems*, 15(1-2):25–46.

Hong Wang, Wenhan Xiong, Mo Yu, Xiaoxiao Guo, Shiyu Chang, and William Yang Wang. 2019. Sentence embedding alignment for lifelong relation extraction. In *Proceedings of NAACL-HLT*, pages 796–806.

Congying Xia, Wenpeng Yin, Yihao Feng, and Philip S. Yu. 2021. Incremental few-shot text classification with multi-round new classes: Formulation, dataset and system. In *Proceedings of NAACL-HLT*, pages 1351–1360.

Rui Xia, Jie Jiang, and Huihui He. 2017. Distantly supervised lifelong learning for large-scale social media sentiment analysis. *IEEE Trans. Affect. Comput.*, 8(4):480–491.

Shipeng Yan, Jiangwei Xie, and Xuming He. 2021. DER: dynamically expandable representation for class incremental learning. In *Proceedings of CVPR*, pages 3014–3023.