Memory Efficient Continual Learning with Transformers

Anonymous Author(s)
Affiliation
Address
email

Abstract

In many real-world scenarios, data to train machine learning models becomes available over time. Unfortunately, these models struggle to continually learn new concepts without forgetting what has been learnt in the past. This phenomenon is known as catastrophic forgetting and it is difficult to prevent due to practical constraints. For instance, the amount of data that can be stored or the computational resources that can be used might be limited. Moreover, applications increasingly rely on large pre-trained neural networks, such as pre-trained Transformers, since compute or data might not be available in sufficiently large quantities to practitioners to train from scratch. In this paper, we devise a method to incrementally train a model on a sequence of tasks using pre-trained Transformers and extending them with Adapters. Different than the existing approaches, our method is able to scale to a large number of tasks without significant overhead and allows sharing information across tasks. On both image and text classification tasks, we empirically demonstrate that our method maintains a good predictive performance without retraining the model or increasing the number of model parameters over time. The resulting model is also significantly faster at inference time compared to Adapter-based state-of-the-art methods.

1 Introduction

Transformers [57], e.g. BERT [12], have shown their effectiveness in various natural language processing (NLP) tasks such as classification [25], Natural Language Inference [41, 39], and Question Answering [16]. Inspired by this achievement, some pioneering works have recently been introduced on adapting Transformers architectures to Computer Vision (CV). Vision Transformers [13, 55] showed that a pure Transformer applied directly to a sequence of image patches can perform well on image classification tasks. Besides, some recent studies [67, 4, 30] showed that Transformers generalize to new domains given only a few samples. Transformers show a great ability to learn complex concepts but when confronted with a sequence of different tasks they tend to "overwrite" the previously learnt concepts. In general, deep networks suffer heavily from this phenomenon, called *catastrophic forgetting* (CF) [35], impeding continual or lifelong learning. In the last few years, a growing body of works attempted to tackle CF in continual learning (CL) [14, 19, 27, 46, 63, 65] but most are not able to meet the scale or accuracy requirements of real-world applications. Moreover, adapting large-scale pre-trained Transformer models to downstream tasks via fine-tuning is the method of choice in NLP applications, posing the need for methods that can directly work with pre-trained models instead of requiring the training of a new model from scratch [15].

In this work, we tackle both text and image classification problem in a setting where the number of tags or classes associated to the input data grows over time. In fact, the ability to continually extend the set of tags or classes used to categorize content is a major problem in many applications. For example, newspapers can tag news according to topics of interest such as "sport", "politics", "food"

Submitted to 36th Conference on Neural Information Processing Systems (NeurIPS 2022). Do not distribute.

by using a pre-trained language model and refining it using a few hundred pre-tagged articles. New tags may appear over time, for example "COVID-19" was a completely unknown news category in 2019 but appeared frequently since. In these cases, retraining models from scratch is often impractical and can lead to inconsistencies in the labeling when compared to the one provided by the previous model. In particular, we focus on incrementally extending classifiers based on pre-trained Transformer models given their ubiquity in NLP and the growing interest in CV.

To address the issue of incremental fine-tuning of pre-trained Transformers in the sequential learning setting without CF, we propose Adaptive Distillation of Adapters (ADA). ADA leverages Adapters [20], a specialized neural network module, to adapt part of the weights in the neural network and a distillation mechanism to consolidate the information previously learnt in a fixed amount of network parameters with limited amount of forgetting. This method allows the user to control the memory consumption, while retaining state-of-the-art performance when running the algorithm on sequences of tens of tasks. This tight memory control is important in industrial applications. The alternative, a model growing in size with the number of tasks, would require a change in hardware to adapt to the growing memory requirements of the deployed model. This would be problematic since a practitioner will incur into higher risk of system instability and be forced to make conservative hardware choices.

The main contribution of our work is ADA, an algorithm that can achieve high predictive performance on both text and images classification in different continual learning scenarios. ADA also provides lower inference time and uses an order of magnitude fewer parameters than state of the art methods such AdaptersFusion. Additionally, we implemented Adapters for vision Transformers and empirically demonstrated their effectiveness.

Related Work. Adapters [20] were proposed for fine-tuning of pre-trained language models and were studied for the multi-task setting. AdapterFusion [39] provides state-of-the-art performance by composing the pre-trained Adapters and it can simply be repurposed for preventing CF in CL by learning one Adapter for every new task. While it has been shown that the number of additional model parameters per Adapter is significantly smaller than the number of parameters used in the pre-trained model [20] (e.g., 3.6% of the parameters of the pre-trained model), since both Adapters and AdapterFusion require to store all the model parameters, the memory consumption increases rapidly with the number of tasks. In the case of a model being trained on 30 tasks, we would have to add more parameters than the number of pre-trained Transformer parameters (details in Section 4), making the method unsuitable for CL.

Recent work studied catastrophic forgetting [53, 10, 25, 33] and incremental learning [64] for NLP and CV [30, 15]. Pasanuru et al. [38] focus on the few-shot setting where only a few data points are available for each task. Ke et al. [26] proposed an architecture to achieve both CF prevention and knowledge transfer. This method has some similarity to AdapterBERT [20] since they insert a CL plug-in module in two locations in BERT. A CL-plugin is a capsule network [50] that uses one separate capsule [18] (2-layer fully connected network) for each task, and like Adapters, memory increases linearly over the time. In addition, this algorithm requires to learn task masks to address knowledge transfer, which is costly to compute. Among those recent works, only a few [30, 15] have applied the Transformers architecture to CL on image datasets. In [30], for each new taskthe model is copied and fixed to be used as the teacher model in the distillation phase. The student model is trained on both new task samples together with the knowledge distillation loss that uses samples from old tasks which is stored in the rehearsal memory. In [15], the authors aim to learn a unified model that will classify an increasingly growing number of classes by building upon a new architecture. However, they need to train a new Transformer, where the process is very costly and contrast with our goal of using public pre-trained models. To the best of our knowledge there is no method able to leverage public pre-trained Transformers while keeping the number of model parameters constant while retaining state-of-the-art predictive performance.

2 Problem setup and Preliminaries

Problem Setup. A sequence of classification tasks $\{T_1, \ldots, T_N\}$ are given where each task T_i contains a different set of data sample (text or image)-label training pairs $(x^i_{1:t}, y^i_{1:t})$ and contains c new classes namely $Y_i = \{Y_i^1, \ldots, Y_i^c\}$ with t examples for each new class. The goal of the learner is to learn a set of parameters $\tilde{\Theta}$ such that $\frac{1}{N}\sum_{i\in\{1,\ldots,N\}} \text{loss}(T_i; \tilde{\Theta})$ is minimized. The task

identifier is provided to the learner with every new batch of data. Moreover, in our specific case, $\tilde{\Theta}$ is composed of a set of parameters Θ provided by a pre-trained model and, depending on the algorithm, some additional parameters which need to be learned for each specific task. In its simplest case, this additional set of model parameters can just be a head model, but some algorithms use significantly more elaborate functions. In the case of the labeling application described in Section 1, each task represents a label and the learner creates a new binary classifier for each label.

For the training of task T_i, the learner can only access the newly added examples and label names in this task. To evaluate the learner, the test data consists of examples across all the previous tasks, where the potential label space for the test example is $Y_1^{1:c} \cup Y_2^{1:c} \cup \cdots \cup Y_N^{1:c}$. All methods that we define in the following sections receive as input a pre-trained model $f_\Theta(.)$, e.g., BERT [12], that is able to extract high quality representations from the input data.

Adapters. Adapters were proposed by [20] as an alternative to fine-tuning in NLP. They add new modules between layers of a pre-trained network called *Adapters*. These modules are feed-forward layers that project the original feature size to a smaller dimension and projects them to the original size thereafter, ensuring that the number of parameters stays substantially small as compared to the original model. (See Appendix A.4 for the details of the Adapter architectures.) Adapters share the pre-trained model parameters Θ across all tasks and introduce a small number of task-specific parameters Φ_i without affecting previous ones. The model is initialized with parameters of a pre-trained model Θ. For each of the task $i \in \{1, \ldots, N\}$ where N is the total number of tasks, a set of new and randomly initialized Adapter parameters Φ_i are introduced. The parameters Θ are fixed and only the parameters Φ_i are trained when a new task is added. This makes it possible to train Adapters for all N tasks, and store the corresponding knowledge in designated parts of the model. The objective for each task $i \in \{1, \ldots, N\}$ is of the form: $\Phi_i \leftarrow \arg\min_\Phi L_i(D_i; \Theta, \Phi)$.

AdapterFusion [39], has been proposed to mitigate the lack of knowledge sharing across tasks. It works in two phases: i) in the knowledge extraction stage, adapters, which encapsulate the task-specific information, are learnt for each of the N tasks; while ii) in the knowledge composition stage, the set of N Adapters are combined by using additional parameters Ψ. The additional parameters Ψ_i for task i are defined as: $\Psi_i \leftarrow \arg\min_\Psi L_i(D_i; \Theta, \Phi_1, \ldots, \Phi_i, \Psi)$. While this provides good predictive performance, in the CL setting, new tasks are added sequentially and storing a large set of Adapters $\Phi_1, \ldots, \Phi_N$ is practically infeasible.

3 Adaptive Distillation of Adapters (ADA)

To address the issues we mentioned in the previous sections, we propose Adaptive Distillation of Adapters (ADA). ADA keeps a fixed amount of Adapters in memory and takes transferability of representations into account to effectively consolidate newly created Adapters with previously created ones. ADA works in two steps: i) it trains a new Adapter and classification head, which we refer as the *new* model, using the training dataset of the new task; ii) it consolidates an *old* model with the new model. To better control the memory usage, ADA has a fix budget for the number of Adapters K that are stored in a pool of old models. In the consolidation phase, the algorithm selects one of the models in the pool using scores that quantify the information contained in the representations they provide. In the following sections, we explain the components of ADA and how they work.

3.1 Distillation of Adapters

For each new task T_n, the Adapter parameters Φ_n are added to the model, while the pre-trained model parameters Θ are kept frozen and are never changed. Only the task-specific Adapter parameters Φ_n and the head model parameters h_n are trained for the current task. The model $f_n(x; \Theta, \Phi_n, h_n)$, with parameters Θ, Φ_n and h_n is called the *new* model. The head model parameters are fixed after training the new model and they are not updated during model consolidation. When a prediction for a task T_i is required, the corresponding Adapter $\Phi_{\gamma(i)}$ and head model h_i is called. γ is a mapping from the task id to the corresponding Adapter in the pool or to the newly trained Adapter. We abuse notation defining f as the function returning the output (*logits*) on all tasks:

$$f(x; \Theta, \Phi, h) = \left[f(x; \Theta, \Phi_{\gamma(1)}, h_1), \ldots, f(x; \Theta, \Phi_{\gamma(n-1)}, h_{n-1}), f(x; \Theta, \Phi_{\gamma(n)}, h_n)\right] \quad (1)$$

For the consolidation step, an Adapter from the pool $\mathbf{\Phi}$ is selected as explained in Section 3.2 and new collection of Adapter pool $\mathbf{\Phi}'$ is created where the selected Adapter is replaced with Φ_c. Φ_c

denotes the consolidated model parameters and preliminarily the parameters are randomly initialized. Similarly, a copy of γ is created to map the old tasks that are associated to the selected Adapter and the new task n to Φ_c. The consolidation then has the following objective:

$$\min_{\Phi_c} \frac{1}{|\mathcal{D}_{distill}|} \sum_{i=1}^{|\mathcal{D}_{distill}|} (f(x_i; \Theta, \mathbf{\Phi}, h) - f(x_i; \Theta, \mathbf{\Phi}', h))^2 \tag{2}$$

where $\mathcal{D}_{distill}$ denotes the unlabeled training data used for distillation, and the distillation loss is computed as the difference between the logits produced by the existing specialist models denoted by $f(x_i; \Theta, \mathbf{\Phi}, h)$ and the consolidated model denoted by $f(x_i; \Theta, \mathbf{\Phi}', h)$ based on L_2 loss. After Φ_c has been trained, $\mathbf{\Phi}$ is swapped with $\mathbf{\Phi}'$. This is a high-level view of the mechanism, our implementation is optimized to avoid copying models when not necessary.

This procedure follows the double distillation loss [69] which is originally proposed for class incremental learning to train a new Adapter that is used with the pre-trained model to classify both old and new tasks. The main idea is first training a separate model for the new class(es) using labeled data, and then combining the new and old models using unlabeled distillation data via a double distillation training objective. We generalize this solution to our case where we have a set of teacher models kept in Adapter pool $\mathbf{\Phi}$ and train a student model Φ_c. Double distillation procedure and the alternative solutions for distillation are discussed in Appendix A.1 but this solution was the best performing one in our experiments.

While several different data sources can be used to populate the buffer, such as using auxiliary external data [69] or generating synthetic data [9], in this work we populate the buffer using covariates from previous tasks selected with Reservoir Sampling [58]. This simple mechanism may not be the most effective, but it will guarantee that no advantage is given to ADA in the experimental comparison.

3.2 Adapter Selection for Distillation

In the previous section, we assumed the Adapter to be consolidated as given but ADA keeps a pool of Adapters and the selection of the Adapter to be distilled is an important part of the algorithm. In fact, our empirical observations show that a random selection of the Adapter provides poor performance (see Section 4.3). The intuition behind our selection mechanism is the following: since a specialized head for every task is created, we can assume that when the features provided by the associated Adapter are highly informative, the updates (i.e., the gradients applied) will be small. At the same time, training a new head with every Adapter in the pool in order to observe which one is the most effective would increase the amount of computation required and significantly impact the usability of the method. The problem of computing the information carried by a representation in an efficient manner has been already studied in the transfer learning community [3, 56, 54].

While, the aim of that research is completely different and, to the best of our knowledge, there is no clear relation between transferability and forgetting, the mathematical foundation of this work are closely related to our intuition. In fact, scores like TransRate [22] employ the mutual information between the features provided by a pre-trained model and the target labels for the task at hand. When the mutual information is high, the transferability is high. More specifically, the knowledge transfer from a source task T_s to a target task T_t is measured as:

$$\text{TrR}_{T_s \to T_t}(f(\Theta, \Phi_{\gamma(s)})) = H(Z) - H(Z|Y), \tag{3}$$

where Y are the labels of target examples and $Z = f(X; \Theta, \Phi_{\gamma(s)})$ are features of them extracted by the pre-trained model and the Adapter associated to the source task.

TransRate is not the only score designed to quantify transferability between a pre-trained model and a new task: *Log Expected Empirical Prediction* (LEEP) [36] is a well-known alternative. Also in this case, the score was designed with a different application in mind, but it leverages the conditional distribution of the target label given the source label to quantify the how informative the information provided by the source model is. Specifically, LEEP is a three steps method. At Step 1, it computes dummy label distributions of the inputs $f(X; \Theta, \Phi_{\gamma(t)}, h_t)$ in the target data set $\mathcal{D}$. At Step 2, it computes the empirical conditional distribution $\hat{P}(y|z)$ of target label y given the source label z. At Step 3, it computes LEEP using $f(X; \Theta, \Phi_{\gamma(s)}, h_s)$ and $\hat{P}(y|z)$:

$$L(f(\Theta, \Phi_{\gamma(s)}, h_s), \mathcal{D}) = \frac{1}{m} \sum_{i=1}^{m} \log \left(\sum_{z \in \mathcal{Z}} \hat{P}(y|z) f(X; \Theta, \Phi_{\gamma(s)}, h_s)_z \right), \tag{4}$$

where z is a dummy label randomly drawn from $f(X; \Theta, \Phi_{\gamma(s)}, h_s)$ and y is randomly drawn from $\hat{P}(y|z)$. We selected TransRate and LEEP for their simplicity and their ability to provide a quantification without training but practitioners can replace these scores with different ones as they see fit.

3.3 Algorithm

Algorithm 1 Adaptive Distillation of Adapters (ADA)

Require: Θ: pre-trained model, K: adapters pool size
  Freeze Θ and create $\gamma = Map()$
  for $n \leftarrow 1$ to N **do**
    A task T_n is received
    Initialize Φ_n
    Process T_n, train new model $f_n(x; \Theta, \Phi_n, h_n)$
    Sample from T_n and add to distillation data $\mathcal{D}_{distill}$
    if $n \leq K$ **then**
      Store f_n in the pool
    else
      $j^* \leftarrow \arg\max_{j \in \{1,\ldots,K\}} \text{TranScore}(T_n, f_j)$
      Add (n, j^*) to γ
      Consolidate model:
        $f_{j^*} = \text{Distill}(f_{j^*}, f_n, \mathcal{D}_{distill})$
    end if
    Serve predictions for any task $i \leq n$ using $f_{\gamma(i)}$
  end for
  $\text{Distill}(f_i, f_j, \mathcal{D}_{distill})$:
    Get soft targets $\hat{y}_i$ from old model f_i with $\mathcal{D}_{distill}$
    Get soft targets $\hat{y}_j$ from new model f_j with $\mathcal{D}_{distill}$
    Initialize Φ_c
    Compute distillation loss and train model $f(x; \Theta, \Phi_c)$ as defined in Eq. 2
    return f

ADA is detailed in Algorithm 1. The graphical workflow of the algorithm is shown in Appendix 6. For every new task, the algorithm trains a new adapter and head model (called Φ_n and h_n). If the adapters pool did not reach the maximum size yet (controlled by K), it just adds it to the pool. If the pool reached the maximum size, the algorithm is forced to select one of the adapters already in the pool and distill it together with the newly trained one. In order to select which adapter to distill, ADA uses the transferability scores (e.g., LEEP or TransRate). Once the adapter in the pool with the highest transferability score (called f_{j^*}) is identified, it consolidates that adapter and the newly trained one into a new adapter and replaces the old one present in the pool. In order to be able to make effective predictions, the algorithm also keeps a mapping γ of which adapter in the pool must be used in combination with each of the task-specific heads.

4 Experiments

In this section, we empirically validate our adapter distillation approach on text and image classification tasks and show that ADA achieves similar performance to AdapterFusion while consuming significantly less memory. We dedicate Section 4.3 to ablation studies providing further insights into the mechanisms implemented in ADA and their contribution.

Datasets and experimental setup. We use three text datasets for multi-label text classification: Arxiv Papers [66] (paper classification), Reuters (RCV1-V2) [29] (news classification), Wiki-30K [71] (Wikipedia article classification) and two dataset for image classification: CIFAR100 [28] and MiniImageNet [49]. Details about the datasets are given in Appendix A.3.

For the *multi-label text classification* experiments, we first sample a sequence of labels from the label space. Then, we create a balanced binary classification task for each label by sampling the same amount of positive data points from the label considered and negative data points from the labels preceding the current one in the sequence. After splitting the data in training and test set, we provide the algorithm with the training set and subsequently measure its performance on the test set. The algorithm never observes any data point in the test set and, more generally, every data point in the dataset is used only once. For Arxiv Papers and Reuters datasets, we created 20 tasks and for Wiki-30K 60. We fixed the number of training samples per task to 100. The test set consists of 40 data points on Reuters and of 100 data points on Arxiv and Wiki-30K.

For *image classification*, we design two scenarios. In the first scenario, each new task is a balanced binary classification problem. Each class can be selected to be the positive class only once. In the second scenario each task is a balanced multi-class classification problem with 5 classes. In both cases we provide the learner with 50 data points per class both at training and test time: in the first scenario each task will have a training set of 250 data points and in the second case of 100 data points.

The total number of tasks is fixed to 20 for both scenarios. The distillation memory size is fixed to 1000 for Wiki-30K which has a larger number of tasks, and to 500 for the others.

Metrics. In [34], three metrics that we discuss in the following are defined to evaluate the performance of a CL method. We use these metrics to evaluate our methods. It is considered that we have access to a test set for each of the N tasks in $\{T_1, \ldots, T_N\}$. After the model finishes learning about the task T_i, its test performance are evaluated on all N tasks. By doing so, a matrix $R \in \mathbb{R}^{N \times N}$ is constructed where $R_{i,j}$ is the test classification accuracy of the model on task T_j after observing the last sample from task T_i. Letting $\bar{b}$ be the vector of test accuracies for each task at random initialization, the three metrics are defined: i) Average Accuracy = $\frac{1}{N}\sum_{i=1}^{N} R_{N,i}$, ii) Backward Transfer (BWT) = $\frac{1}{N-1}\sum_{i=1}^{N-1} R_{N,i} - R_{i,i}$ and iii) Forward Transfer (FWT) = $\frac{1}{N-1}\sum_{i=2}^{N} R_{i-1,i} - \bar{b}_i$. (The larger these metrics, the better the model.) All the results in this section are averaged over 5 runs.

Baselines. We compare ADA the following baselines. 1) *Fine-tuning head model (B1)*: We freeze the pre-trained representation and only fine-tune the output layer of each classification task. The output layer is multiple-head binary classifier that we also use for the other methods. 2) *Fine tuning the full model (B2)*: We fine-tune both the pre-trained representation and the output layer for each classification task. 3) *Adapters* [20]: We train and keep separate Adapters for each classification task as well as the head models. 4) *AdapterFusion* [39]: It is a two stage learning algorithm that leverages knowledge from multiple tasks by combining the representations from several task Adapters in order to improve the performance on the target task. This follows exactly the solution depicted in Section 2. 5) *Experience Replay (ER)* [48]: ER is a commonly used baseline in Continual Learning that stores a subset of data for each task and then "replays" the old data together with the new one to avoid forgetting old concepts. [11] propose to use such a memory module for sparse experience replay and local adaptation in the language domain. This method stores all training examples, in order to achieve optimal performance. To make this method comparable with *adapter-based* methods, we freeze pre-trained representation, add a single adapter parameters Φ and train the adapter by replaying examples from old tasks while training using data from the new task. In order to keep baselines comparable we assign to ER the same amount of memory is used for the distillation buffer in ADA. In addition to these baselines, we use one special case of ADA with K=1 as a baseline to demonstrate the advantage of effective consolidation of Adapters.

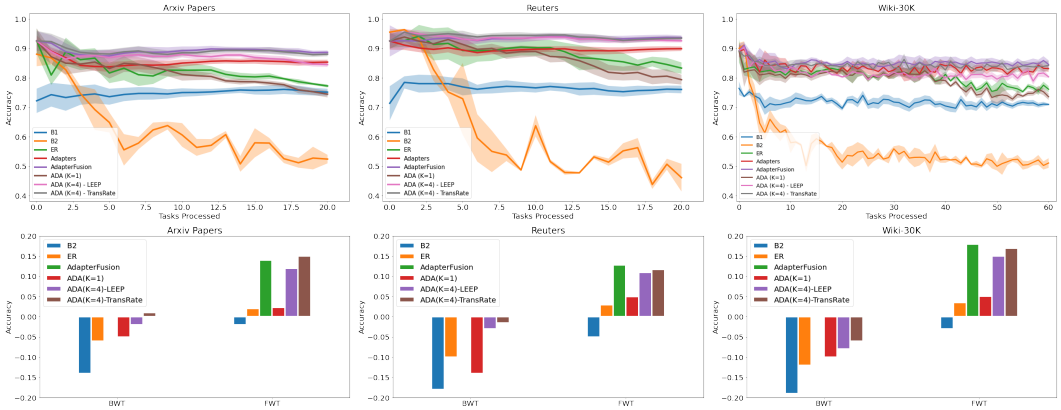

Figure 1: Comparison between baselines and ADA on Arxiv, Reuters and Wiki-30K. On top, we report the number of tasks processed on the x-axis and we report the average accuracy measured on the test set of the tasks processed on the y-axis, shaded area shows standard deviation. On bottom, we report BWT and FWT.

Adapter architectures. We use pre-trained models from HuggingFace Transformers [61] as our base feature extractors. We ran experiments with BERT$_{base}$, DistilBERT$_{base}$, RoBERTa$_{base}$ for text classification and ViT-B and DeiT-B for image classification. We analyze the cases based on all these models, due to the space constraints, we present BERT$_{base}$ in this section and the rest in Appendix A.5. BERT$_{base}$ model uses 12 layers of Transformers block with a hidden size of 768 and number of self-attention heads as 12 and has around 110 M (440 MB) trainable parameters. For the Adapter implementation, we use Adapter-Hub [40], but no Adapter implementation was available for Vision Transformers. We define our architecture of Adapters for ViT and DeiT in Appendix A.4. An

Adapter has a simple bottleneck architecture that contains fewer parameters than the attention and the feed-forward layers. The Adapter size is the hyper-parameter that is tuned and it can be set to $\{12, 24, 48, 96, 192, 384\}$ for BERT$_{base}$ model. For all the methods, we use the same configuration for the Adapters, setting the size to 48. With this setting, an Adapter contains ~ 1.8 M parameters. We also train a head model for each task, that has 768 parameters for BERT$_{base}$ (last hidden size of BERT$_{base}$$\times$output size, which equals to 1 for binary classification). The tables in Appendix A.5 reports the number of parameters used for baselines and ADA in our experiments.

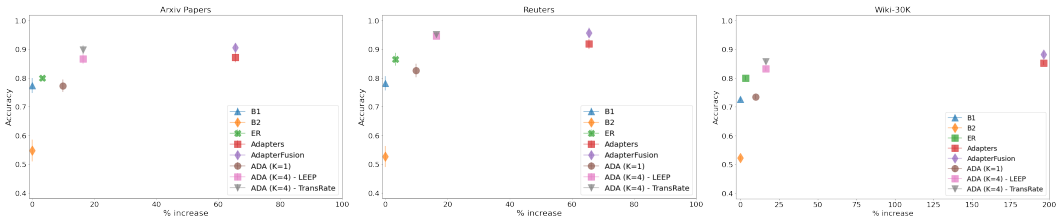

Figure 2: Comparison of the $\%$ increase in the number of parameters of baseline methods and ADA on Arxiv, Reuters and Wiki-30K. The predictive performance reported on the y-axis is measured after processing all tasks.

4.1 Text Classification

Predictive performance. Figure 1 shows the comparison of ADA and the baseline methods. It can be clearly seen that freezing all pre-trained model parameters, and fine-tuning only the head models (B1) led to an inferior performance compared to adapter-based approaches. The main reason is that the head models have small amount of parameters to train and fine-tuning only the heads suffers from under-fitting. B2 performs good only for first 2-3 tasks, since we keep training the complete model, it forgets the previously learned tasks very quickly. As mentioned above, Adapters and AdapterFusion add ~ 1.8 M parameters for each task and train these parameters with new task data, and these parameters are fixed after training. So, they perform well on both new tasks and previous tasks. The results on each dataset confirm this. Both ER and ADA K=1, perform closely with Adapters almost for half of the tasks. The similar behavior of ER and ADA K=1 demonstrates that the distillation with soft labels works well and it is almost as good as training with the true labels. Later the performance declines for both methods, because the capacity of the Adapter is exceeded. ADA LEEP and ADA TransRate results with K=4 Adapters show that selective consolidation of Adapters significantly improves the performance. Their performance is on par with AdapterFusion while the number of model parameters is significantly lower. We present BERT$_{base}$ results in this section while the rest is reported in Appendix A.9.

We also compute FWT and BWT scores for these methods. We didn't present B1 and Adapters in the plots, since both FWT and BWT are zero for them. BWT is zero for AdapterFusion, since the fusion parameter is computed with available Adapters, and the Adapters trained later is not used for the previous tasks. ADA-LEEP and ADA-TransRate minimizes negative backward transfer, while showing a positive forward transfer for all datasets.

Memory consumption. Figure 2 shows the increase in terms of percentage in the number of parameters used by each method and their predictive performance. We see that on Wikipedia, 200% of the base model parameters ($220M$ additional parameters) are added. These results make clear that ADA is significantly more efficient in terms of memory usage. It can achieve predictive performance similar to the one of Adapters and AdapterFusion while requiring significantly less model parameters. On Reuters and Arxiv, it can store the parameters of only 5 Adapters (K=4 Adapters in the pool, and one Adapter for new task), against the 20 required by AdapterFusion (on Wikipedia it is 5 against 60).

Inference time. When machine learning models are used to power customer-facing web sites, they are often required to provide predictions in a few milliseconds to keep the overall latency within requirements. Moreover, in this kind of application the model will be trained once and make billions of predictions so a reasonable increase in the training time is irrelevant compared to a decrease in the inference time. We report the inference time results of ADA and other Adapter based methods in Appendix A.6. Results demonstrate that ADA provides a sufficiently fast inference for most applications and still offers opportunities to speed it up further, for example by employing smaller pre-trained Transformers (e.g. DistilBERT, see Appendix A.5).

Training time. Distillation of Adapters brings an extra cost for ADA while learning fusion parameters brings an extra cost for AdapterFusion. Computing transferability takes constant time which is negligible. Distillation costs training an additional Adapter (1.6 % of full fine-tuning time of BERT). Figure 9d in AppendixA.6 reports the average training time comparison on Wiki-30K that is the largest difference with AdapterFusion given larger number of tasks. We can clearly see that the difference is small (ADAis 3.37% more, ADA-TransRate is 5.6% more) while the difference between the inference time is significant.

4.2 Image Classification

For image classification experiments, we add *Elastic Weight Consolidation (EWC)* [27] as an additional baseline since it is widely used in CL literature for image classification. EWC is a regularization-based CL method that assumes that some weights of the trained neural network are more important for previously learned tasks than others. During training of the neural network on a new task, changes to the weights of the network are made less likely the greater their importance.

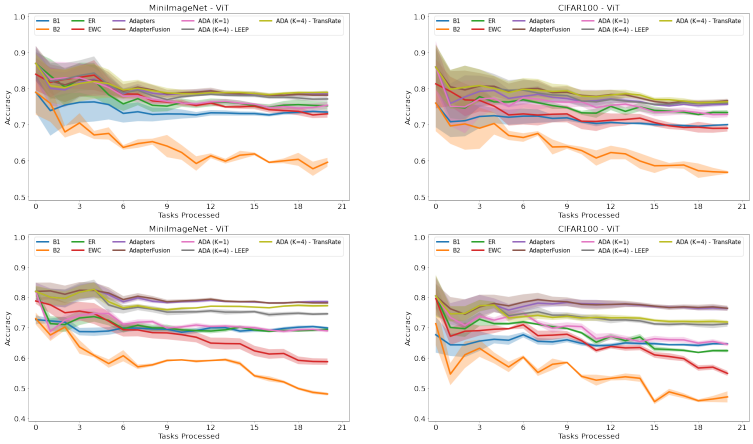

Figure 3: Comparison between baselines and ADA with ViT model on MiniImageNet and CIFAR100. Top figures shows the binary, and bottom figures shows the multi-class classification results.

Figure 3 shows the comparison of ADA and the baseline methods. The results show the same behaviour with text classification. B1 leaded to an inferior performance compared to other approaches. B2 performs well only for initial tasks and it forgets the previously learned tasks very quickly. Results confirm there is no forgetting for Adapters and AdapterFusion. Although the careful tuning of regularization coefficient, EWC cannot handle CF, especially for multi-class classification problem. ADA with K=1 shows that distillation alone doesn't prevent forgetting. In almost all cases, ER performs on-par with ADA K=1, providing evidence that a small amount of memory can actually improve performance compared to fine-tuning or regularization, but the improvement is limited and does not last as the number of tasks increases.

ADA-LEEP and ADA-TransRate results with K=4 Adapters show that selective consolidation of Adapters significantly improves the performance. For binary classification, their performance are on par with AdapterFusion while the number of model parameters is significantly lower. For multi-class, their performance slightly declines after a certain number of tasks. This is discussed in next section and the main reason is that the capacity of the Adapter is exceeded. To validate the interoperability of ADA to different models, we run the same experiments on DeiT model and present the results in Appendix A.10 due to space constraints.

4.3 Ablation studies

Comparison with larger distilled models. In Section 4 we compared ADA with the special case of ADA with K=1 to evaluate the improvement provided by our approach over a distillation-only solutions. We would like to provide additional observations of the superior performance of ADA by comparing its performance with the one of a distilled Adapter using more parameters. Specifically, we run an experiment where we compare ADA with K=4 and ADA with K=1 as displayed before but

in this case the "size" of the Adapter, which is 48 for Size×1, is multiplied by 4 for Size×4 Adapter to have a comparison where the different methods use the same number of model parameters. Since K=1 is a special case where a single Adapter is kept in the pool, the transferability metric is irrelevant and we can see ADA with K=1 as a method purely based on distillation like DMC [69].

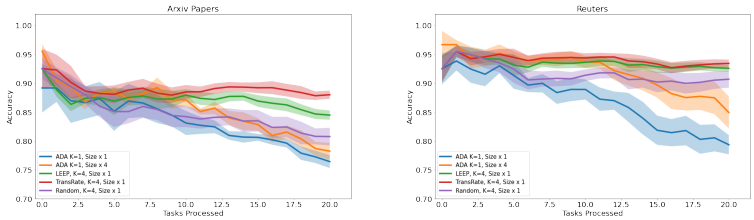

Figure 4: Impact of *LEEP* and *TransRate* when the total number of Adapter parameters is same on Arxiv and Reuters.

The results reported in Figure 4 show that ADA can make a better use of the model parameters compared to a distillation-only method and that the intelligent selection of which Adapters to distill together makes once again a big difference. It is also interesting to observe that the usage of additional model parameters brings a clear advantage but the mixed comparison between the ADA K=4 with random Adapter selection and ADA K=1 with four times larger Adapters leaves some questions open regarding how far distillation can get in this setting. Another finding is that *TransRate* outperforms *LEEP* in most cases. It is also demonstrated in the original paper [22] that *TransRate* has a strong correlation to the transfer learning performance and it outperforms *LEEP* and other metrics employed.

Impact of the Adapters pool size. In our experiments we used a fixed number of Adapters in the pool size, but more Adapters can be added to ADA's pool as more tasks are processed. This may actually be the preferred usage in some applications. We already know that having an Adapter per task provides good performance and using multiple of them at the same time like in AdapterFusion provides a benefit, but we would like to verify the sensitivity to this parameter. The results reported

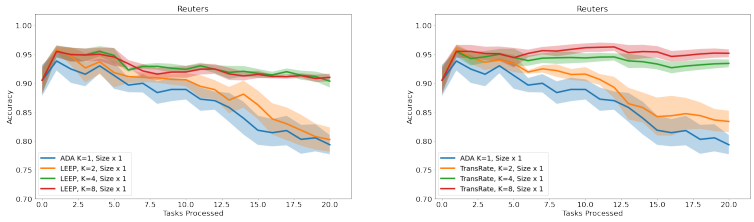

Figure 5: *LEEP* and *TransRate* performances when $K = \{1, 2, 4, 8\}$ on Reuters.

in Figure 5 show a rapidly decreasing added value when the number of Adapters grows, a behavior which aligns well with our practical requirements of keeping the number of model parameters under control when the number of tasks increases. See additional experiments in Appendix A.11.

5 Conclusion

In this paper we presented ADA, a new method that allows neural text and image classifiers to learn new classes based on pre-trained Transformers while maintaining strict control of the memory usage and reaching state-of-the-art predictive performance. The method has shown to be effective in different domains and allows users to leverage publicly available pre-trained Transformers for continual classification tasks. We empirically demonstrated that Adapters can give good results when used in combination with vision Transformers on CV tasks. We evaluated ADA on different classification tasks and demonstrated that the predictive performance is competitive with state-of-the-art methods which use up to an order of magnitude parameters. Moreover, ADA displayed lower latency at inference time and improved data efficiency for some specific settings (see Appendix A.8).

Transformers are very popular, but they are not the only models being widely used in practice. We consider this the main weakness of our approach and we would like to further expand our activity to perform CL on other widely used pre-trained models such as ResNet. Addressing multi-modal classification using text and images together will be the other focus of our future research.

References

[1] Rahaf Aljundi, Min Lin, Baptiste Goujaud, and Yoshua Bengio. Gradient based sample selection for online continual learning. *arXiv preprint arXiv:1903.08671*, 2019.

[2] Rahaf Aljundi, Marcus Rohrbach, and Tinne Tuytelaars. Selfless sequential learning. *arXiv preprint arXiv:1806.05421*, 2018.

[3] Yajie Bao, Yang Li, Shao-Lun Huang, Lin Zhang, Lizhong Zheng, Amir Zamir, and Leonidas Guibas. An information-theoretic approach to transferability in task transfer learning. In *2019 IEEE International Conference on Image Processing (ICIP)*, pages 2309–2313. IEEE, 2019.

[4] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

[5] Cristian Bucilua, Rich Caruana, and Alexandru Niculescu-Mizil. Model compression. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 535–541, 2006.

[6] Pietro Buzzega, Matteo Boschini, Angelo Porrello, Davide Abati, and Simone Calderara. Dark experience for general continual learning: a strong, simple baseline. *arXiv preprint arXiv:2004.07211*, 2020.

[7] Francisco M Castro, Manuel J Marín-Jiménez, Nicolás Guil, Cordelia Schmid, and Karteek Alahari. End-to-end incremental learning. In *Proceedings of the European conference on computer vision (ECCV)*, pages 233–248, 2018.

[8] Arslan Chaudhry, Marcus Rohrbach, Mohamed Elhoseiny, Thalaiyasingam Ajanthan, Puneet K Dokania, Philip HS Torr, and M Ranzato. Continual learning with tiny episodic memories. 2019.

[9] Akshay Chawla, Hongxu Yin, Pavlo Molchanov, and Jose Alvarez. Data-free knowledge distillation for object detection. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pages 3289–3298, 2021.

[10] Yung-Sung Chuang, Shang-Yu Su, and Yun-Nung Chen. Lifelong language knowledge distillation. *arXiv preprint arXiv:2010.02123*, 2020.

[11] Cyprien de Masson d'Autume, Sebastian Ruder, Lingpeng Kong, and Dani Yogatama. Episodic memory in lifelong language learning. *arXiv preprint arXiv:1906.01076*, 2019.

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

[13] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

[14] Arthur Douillard, Matthieu Cord, Charles Ollion, Thomas Robert, and Eduardo Valle. Podnet: Pooled outputs distillation for small-tasks incremental learning. In *European Conference on Computer Vision*, pages 86–102. Springer, 2020.

[15] Arthur Douillard, Alexandre Ramé, Guillaume Couairon, and Matthieu Cord. Dytox: Transformers for continual learning with dynamic token expansion. *arXiv preprint arXiv:2111.11326*, 2021.

[16] Claudio Greco, Barbara Plank, Raquel Fernández, and Raffaella Bernardi. Psycholinguistics meets continual learning: Measuring catastrophic forgetting in visual question answering. *arXiv preprint arXiv:1906.04229*, 2019.

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

[18] Geoffrey E Hinton, Alex Krizhevsky, and Sida D Wang. Transforming auto-encoders. In *International conference on artificial neural networks*, pages 44–51. Springer, 2011.

[19] Saihui Hou, Xinyu Pan, Chen Change Loy, Zilei Wang, and Dahua Lin. Learning a unified classifier incrementally via rebalancing. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 831–839, 2019.

[20] Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin De Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. Parameter-efficient transfer learning for nlp. In *International Conference on Machine Learning*, pages 2790–2799. PMLR, 2019.

[21] Wenpeng Hu, Qi Qin, Mengyu Wang, Jinwen Ma, and Bing Liu. Continual learning by using information of each class holistically. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 7797–7805, 2021.

[22] Long-Kai Huang, Ying Wei, Yu Rong, Qiang Yang, and Junzhou Huang. Frustratingly easy transferability estimation. *arXiv preprint arXiv:2106.09362*, 2021.

[23] Yufan Huang, Yanzhe Zhang, Jiaao Chen, Xuezhi Wang, and Diyi Yang. Continual learning for text classification with information disentanglement based regularization. *arXiv preprint arXiv:2104.05489*, 2021.

[24] Khurram Javed and Martha White. Meta-learning representations for continual learning. *arXiv preprint arXiv:1905.12588*, 2019.

[25] Zixuan Ke, Bing Liu, Hao Wang, and Lei Shu. Continual learning with knowledge transfer for sentiment classification. In *ECML/PKDD (3)*, pages 683–698, 2020.

[26] Zixuan Ke, Hu Xu, and Bing Liu. Adapting bert for continual learning of a sequence of aspect sentiment classification tasks. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 4746–4755, 2021.

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

[28] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[29] David D Lewis, Yiming Yang, Tony Russell-Rose, and Fan Li. Rcv1: A new benchmark collection for text categorization research. *Journal of machine learning research*, 5(Apr):361–397, 2004.

[30] Duo Li, Guimei Cao, Yunlu Xu, Zhanzhan Cheng, and Yi Niu. Technical report for iccv 2021 challenge SSLAD-Track3B: Transformers are better continual learners. *arXiv preprint arXiv:2201.04924*, 2022.

[31] Zhizhong Li and Derek Hoiem. Learning without forgetting. *IEEE transactions on pattern analysis and machine intelligence*, 40(12):2935–2947, 2017.

[32] Nelson F Liu, Matt Gardner, Yonatan Belinkov, Matthew E Peters, and Noah A Smith. Linguistic knowledge and transferability of contextual representations. *arXiv preprint arXiv:1903.08855*, 2019.

[33] Zihan Liu, Genta Indra Winata, Andrea Madotto, and Pascale Fung. Exploring fine-tuning techniques for pre-trained cross-lingual models via continual learning. *arXiv preprint arXiv:2004.14218*, 2020.

[34] David Lopez-Paz and Marc'Aurelio Ranzato. Gradient episodic memory for continual learning. *Advances in neural information processing systems*, 30:6467–6476, 2017.

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

[36] Cuong Nguyen, Tal Hassner, Matthias Seeger, and Cedric Archambeau. LEEP: A new measure to evaluate transferability of learned representations. In *International Conference on Machine Learning*, pages 7294–7305. PMLR, 2020.

[37] Abiola Obamuyide, Andreas Vlachos, et al. Meta-learning improves lifelong relation extraction. 2019.

[38] Ramakanth Pasunuru, Veselin Stoyanov, and Mohit Bansal. Continual few-shot learning for text classification. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 5688–5702, 2021.

[39] Jonas Pfeiffer, Aishwarya Kamath, Andreas Rücklé, Kyunghyun Cho, and Iryna Gurevych. AdapterFusion: Non-destructive task composition for transfer learning. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*, pages 487–503, Online, April 2021. Association for Computational Linguistics.

[40] Jonas Pfeiffer, Andreas Rücklé, Clifton Poth, Aishwarya Kamath, Ivan Vulić, Sebastian Ruder, Kyunghyun Cho, and Iryna Gurevych. Adapterhub: A framework for adapting transformers. *arXiv preprint arXiv:2007.07779*, 2020.

[41] Jonas Pfeiffer, Ivan Vulić, Iryna Gurevych, and Sebastian Ruder. Mad-x: An adapter-based framework for multi-task cross-lingual transfer. *arXiv preprint arXiv:2005.00052*, 2020.

[42] Jason Phang, Thibault Févry, and Samuel R Bowman. Sentence encoders on stilts: Supplementary training on intermediate labeled-data tasks. *arXiv preprint arXiv:1811.01088*, 2018.

[43] Clifton Poth, Jonas Pfeiffer, Andreas Rücklé, and Iryna Gurevych. What to pre-train on? efficient intermediate task selection. *arXiv preprint arXiv:2104.08247*, 2021.

[44] Yada Pruksachatkun, Jason Phang, Haokun Liu, Phu Mon Htut, Xiaoyi Zhang, Richard Yuanzhe Pang, Clara Vania, Katharina Kann, and Samuel R Bowman. Intermediate-task transfer learning with pretrained models for natural language understanding: When and why does it work? *arXiv preprint arXiv:2005.00628*, 2020.

[45] Joan Puigcerver, Carlos Riquelme, Basil Mustafa, Cedric Renggli, André Susano Pinto, Sylvain Gelly, Daniel Keysers, and Neil Houlsby. Scalable transfer learning with expert models. *arXiv preprint arXiv:2009.13239*, 2020.

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

[47] Matthew Riemer, Ignacio Cases, Robert Ajemian, Miao Liu, Irina Rish, Yuhai Tu, and Gerald Tesauro. Learning to learn without forgetting by maximizing transfer and minimizing interference. *arXiv preprint arXiv:1810.11910*, 2018.

[48] David Rolnick, Arun Ahuja, Jonathan Schwarz, Timothy P Lillicrap, and Greg Wayne. Experience replay for continual learning. *arXiv preprint arXiv:1811.11682*, 2018.

[49] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115(3):211–252, 2015.

[50] Sara Sabour, Nicholas Frosst, and Geoffrey E Hinton. Dynamic routing between capsules. *arXiv preprint arXiv:1710.09829*, 2017.

[51] Jonathan Schwarz, Wojciech Czarnecki, Jelena Luketina, Agnieszka Grabska-Barwinska, Yee Whye Teh, Razvan Pascanu, and Raia Hadsell. Progress & compress: A scalable framework for continual learning. In *International Conference on Machine Learning*, pages 4528–4537. PMLR, 2018.

[52] Konstantin Shmelkov, Cordelia Schmid, and Karteek Alahari. Incremental learning of object detectors without catastrophic forgetting. In *Proceedings of the IEEE international conference on computer vision*, pages 3400–3409, 2017.

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

[54] Yang Tan, Yang Li, and Shao-Lun Huang. Otce: A transferability metric for cross-domain cross-task representations. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 15779–15788, 2021.

[55] Hugo Touvron, Matthieu Cord, Matthijs Douze, Francisco Massa, Alexandre Sablayrolles, and Hervé Jégou. Training data-efficient image transformers & distillation through attention. In *International Conference on Machine Learning*, pages 10347–10357. PMLR, 2021.

[56] Anh T Tran, Cuong V Nguyen, and Tal Hassner. Transferability and hardness of supervised classification tasks. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 1395–1405, 2019.

[57] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[58] Jeffrey S Vitter. Random sampling with a reservoir. *ACM Transactions on Mathematical Software (TOMS)*, 11(1):37–57, 1985.

[59] Tu Vu, Tong Wang, Tsendsuren Munkhdalai, Alessandro Sordoni, Adam Trischler, Andrew Mattarella-Micke, Subhransu Maji, and Mohit Iyyer. Exploring and predicting transferability across nlp tasks. *arXiv preprint arXiv:2005.00770*, 2020.

[60] Zirui Wang, Sanket Vaibhav Mehta, Barnabás Póczos, and Jaime Carbonell. Efficient meta lifelong-learning with limited memory. *arXiv preprint arXiv:2010.02500*, 2020.

[61] Thomas Wolf, Julien Chaumond, Lysandre Debut, Victor Sanh, Clement Delangue, Anthony Moi, Pierric Cistac, Morgan Funtowicz, Joe Davison, Sam Shleifer, et al. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45, 2020.

[62] Mitchell Wortsman, Vivek Ramanujan, Rosanne Liu, Aniruddha Kembhavi, Mohammad Rastegari, Jason Yosinski, and Ali Farhadi. Supermasks in superposition. *arXiv preprint arXiv:2006.14769*, 2020.

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

[64] Congying Xia, Wenpeng Yin, Yihao Feng, and Philip Yu. Incremental few-shot text classification with multi-round new classes: Formulation, dataset and system. *arXiv preprint arXiv:2104.11882*, 2021.

[65] Shipeng Yan, Jiangwei Xie, and Xuming He. Der: Dynamically expandable representation for class incremental learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3014–3023, 2021.

[66] Pengcheng Yang, Xu Sun, Wei Li, Shuming Ma, Wei Wu, and Houfeng Wang. Sgm: sequence generation model for multi-label classification. *arXiv preprint arXiv:1806.04822*, 2018.

[67] Wenpeng Yin, Jamaal Hay, and Dan Roth. Benchmarking zero-shot text classification: Datasets, evaluation and entailment approach. *arXiv preprint arXiv:1909.00161*, 2019.

[68] Guanxiong Zeng, Yang Chen, Bo Cui, and Shan Yu. Continual learning of context-dependent processing in neural networks. *Nature Machine Intelligence*, 1(8):364–372, 2019.

[69] Junting Zhang, Jie Zhang, Shalini Ghosh, Dawei Li, Serafettin Tasci, Larry Heck, Heming Zhang, and C-C Jay Kuo. Class-incremental learning via deep model consolidation. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pages 1131–1140, 2020.

[70] Peng Zhou, Long Mai, Jianming Zhang, Ning Xu, Zuxuan Wu, and Larry S Davis. M2kd: Multi-model and multi-level knowledge distillation for incremental learning. *arXiv preprint arXiv:1904.01769*, 2019.

[71] Arkaitz Zubiaga. Enhancing navigation on wikipedia with social tags. *arXiv preprint arXiv:1202.5469*, 2012.

Checklist

The checklist follows the references. Please read the checklist guidelines carefully for information on how to answer these questions. For each question, change the default **[TODO]** to [Yes] , [No] , or [N/A] . You are strongly encouraged to include a **justification to your answer**, either by referencing the appropriate section of your paper or providing a brief inline description. For example:

- Did you include the license to the code and datasets? [Yes] See Section **??**.
- Did you include the license to the code and datasets? [No] The code and the data are proprietary.
- Did you include the license to the code and datasets? [N/A]

Please do not modify the questions and only use the provided macros for your answers. Note that the Checklist section does not count towards the page limit. In your paper, please delete this instructions block and only keep the Checklist section heading above along with the questions/answers below.

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes] Our work is tied to the usage of Transformers, we remind it in the conclusions.
   (c) Did you discuss any potential negative societal impacts of your work? [N/A]
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A] We don't have theoretical results.
   (b) Did you include complete proofs of all theoretical results? [N/A] We don't have theorems.
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? **[TODO]**We will provide the code if the paper is accepted.
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] See Section 4 and Appendix A.2.
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] Yes, all figures show the standard deviation (shaded regions).
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] See Appendix A.2.
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes] We used AdapterHub, that is cited in experiments and appendix sections.
   (b) Did you mention the license of the assets? [Yes] Yes, in Appendix A.3.
   (c) Did you include any new assets either in the supplemental material or as a URL? [N/A]
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]