Progressive Memory Banks for Incremental Domain Adaptation

Nabiha Asghar[*†] **Lili Mou**[*‡] **Kira A. Selby**[†]
Kevin D. Pantasdo[†] **Pascal Poupart**[†] **Xin Jiang**[◇]
[†]Cheriton School of Computer Science, University of Waterloo, Canada
{nasghar,kaselby,kevin.pantasdo,ppoupart}@uwaterloo.ca
[‡]AdeptMind Research, Toronto, Canada
lili@adeptmind.ai doublepower.mou@gmail.com
[◇]Noah's Ark Lab, Huawei Technologies, Hong Kong
jiang.xin@huawei.com

Abstract

This paper addresses the problem of incremental domain adaptation (IDA). We assume each domain comes one after another, and that we could only access data in the current domain. The goal of IDA is to build a unified model performing well on all the domains that we have encountered. We propose to augment a recurrent neural network (RNN) with a directly parameterized memory bank, which is retrieved by an attention mechanism at each step of RNN transition. The memory bank provides a natural way of IDA: when adapting our model to a new domain, we progressively add new slots to the memory bank, which increases the number of parameters, and thus the model capacity. We learn the new memory slots and fine-tune existing parameters by back-propagation. Experimental results show that our approach achieves significantly better performance than fine-tuning alone, which suffers from the catastrophic forgetting problem. Compared with expanding hidden states, our approach is more robust for old domains, shown by both empirical and theoretical results. Our model also outperforms previous work of IDA including elastic weight consolidation (EWC) and the progressive neural network.[1]

1 Introduction

Domain adaptation aims to transfer knowledge from one domain to another in a machine learning system. This is important for neural networks, which are data-hungry and prone to overfitting. In this paper, we especially focus on *incremental domain adaptation* (IDA), where we assume different domains come sequentially one after another. We only have access to the data in the current domain, but hope to build a unified model that performs well on all the domains that we have encountered (Xu et al., 2014; Rusu et al., 2016; Kirkpatrick et al., 2017).

Incremental domain adaptation is useful in various scenarios. Suppose a company is doing business with different partners over a long period of time, the company could only access the data of the partner with a current contract. However, the machine learning model is the company's property (if complying with the contract); therefore, it is desired to preserve as much knowledge as possible in the model for business profits.

Another application of IDA is a quick adaptation to new domains. If the environment of a deployed machine learning system changes frequently, traditional methods like jointly training all domains require the learning machine to be re-trained from scratch every time a new domain comes. Fine-tuning a neural network by a few steps of gradient updates does transfer quickly, but it suffers from the *catastrophic forgetting problem* (Kirkpatrick et al., 2017). Suppose when predicting we do not know the domain of a data point, the (single) fine-tuned model cannot predict well for samples in previous domains, as it tends to "forget" quickly during fine-tuning.

A recent trend of domain adaptation in the deep learning regime is the progressive neural network (Rusu et al., 2016), which progressively grows the network capacity if a new domain comes. Typically, this is done by enlarging the model with new hidden states and a new predictor (Figure 1a). To avoid interfering with existing knowledge, the newly added hidden states are not fed back to the previously trained states. During training, they fix all existing parameters, and only train the newly added ones. For inference, they use the new predictor for all domains. This is some-

[*]Equal contribution. Work partially done when LM was a postdoc at the University of Waterloo.

[1]Our IDA code is available at https://github.com/nabihach/IDA

times undesired as the new predictor is trained with only the last domain.

In this paper, we propose a progressive memory bank for incremental domain adaptation. Our model augments a recurrent neural network (RNN) with a memory bank, which is a set of distributed, real-valued vectors capturing domain knowledge. The memory is retrieved by an attention mechanism during RNN information processing. When our model is adapted to new domains, we progressively increase the slots in the memory bank. But different from Rusu et al. (2016), we fine-tune all the parameters, including RNN and the memory bank. Experimental results show that, when the model capacity increases, the RNN does not forget much even if the entire network is fine-tuned. Compared with expanding RNN hidden states, the newly added memory slots do not contaminate existing knowledge in RNN states, as will be shown by a theorem.

We evaluate our approach on the multi-genre natural language inference (MultiNLI) corpus (Williams et al., 2018), which contains 5 domains with massive training samples. Experiments support our hypothesis that the proposed approach adapts well to target domains without catastrophic forgetting of the source. Our model outperforms the naïve fine-tuning method, the original progressive neural network, as well as other IDA techniques including elastic weight consolidation (EWC, Kirkpatrick et al., 2017).

2 Related Work

2.1 Domain Adaptation

Domain adaptation has been widely studied in NLP. Mou et al. (2016) analyze two straightforward settings, namely, multi-task learning (jointly training all domains) and fine-tuning (training one domain and fine-tuning on the other). One recent advance of domain adaptation is adversarial learning, where the neural features are trained not to classifier the domain. Such approach can be extended to private-share architectures (Liu et al., 2017). However, all these approaches (except fine-tuning) require that all domains are available at the same time, and are not IDA approaches.

Kirkpatrick et al. (2017) address the catastrophic forgetting problem of neural networks when fine-tuning, and propose a regularization term based on Fisher-information; they call the method elastic weight consolidation (EWC). While some follow-up studies report EWC achieves high performance in their scenarios (Lee et al., 2017), others show that EWC is less effective (Yoon et al., 2018).

Rusu et al. (2016) propose a progressive neural network that progressively increases the number of hidden states (Figure 1a). To avoid overriding existing information, they propose to fix the weights of the learned network, and do not feed new states to old ones. This results in multiple predictors, requiring that a data sample is labeled with its domain during the test time. Should different domains be highly correlated to each other, the predictor of a previous domain cannot make use of new data to improve performance. If we otherwise use the last predictor to predict samples from all domains, its performance may be low for previous domains, as the predictor is only trained with the last domain.

Yoon et al. (2018) propose an extension of the progressive network. They identify which existing hidden units are relevant for the new task (with their sparse penalty), and fine-tune only the corresponding subnetwork. However, sparsity is not common for RNNs in NLP applications, as sparse recurrent connections are harmful. A similar phenomenon is that dropout of recurrent connections is harmful (Bayer et al., 2013).

2.2 Memory-Based Neural Networks

Our work is related to memory-based neural networks. Sukhbaatar et al. (2015) propose an end-to-end memory network that assigns a slot for an entity, and aggregates information by multiple attention-based layers. In their work, they design the architecture for bAbI question answering, and assign a memory slot for each sentence. Such idea can be extended to various scenarios, for example, assigning slots to external knowledge for question answering (Das et al., 2017) and assigning slots to dialog history for a conversation system (Madotto et al., 2018).

Another type of memory in the neural network regime is the neural Turing machine (NTM, Graves et al., 2016). Their memory is not directly parameterized, but is read or written by a neural controller. Therefore, such memory serves as temporary scratch paper, but does not store knowledge itself. In NTM, the memory information and operation are fully distributed/neuralized, as they do not correspond to the program on a true (non-

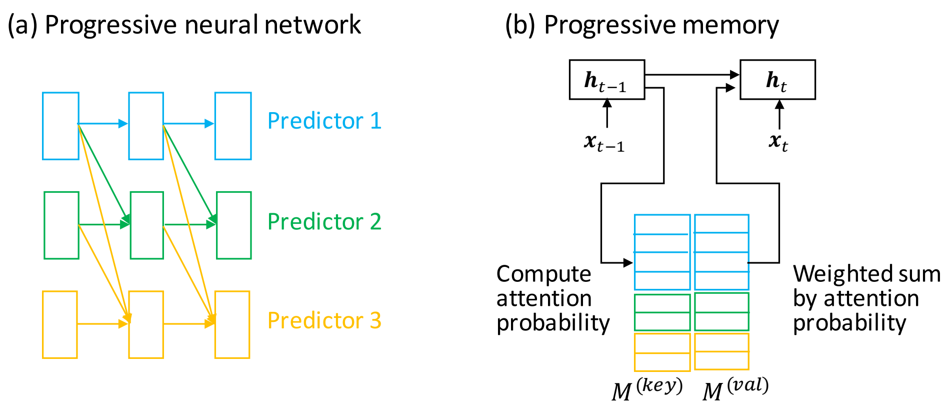

Figure 1: (a) Progressive neural network (Rusu et al., 2016). (b) One step of RNN transition in our progressive memory network. Colors indicate different domains.

neural) Turing machine.

Zhang et al. (2018) combine the above two styles of memory for task-oriented dialog systems, where they have both slot-value memory and read-and-write memory.

Different from the above work, our memory bank stores knowledge in a distributed fashion, where each slot does not correspond to a concrete entity. Our memory is directly parameterized, and thus it is able store knowledge of the datasets. The memory parameters interact in a different way from RNN weights, and provide a natural way of incremental domain adaptation.

3 Proposed Approach

Our model is based on a recurrent neural network (RNN). At each time step, the RNN takes the embedding of the current word as input, and changes its states accordingly. This can be represented by

$$\boldsymbol{h}_i = \text{RNN}(\boldsymbol{h}_{i-1}, \boldsymbol{x}_i). \tag{1}$$

where $\boldsymbol{h}_i$ and $\boldsymbol{h}_{i-1}$ are the hidden states of time steps i and $i-1$, respectively. $\boldsymbol{x}_i$ is the input at the ith step.

In our work, we use the long short term memory (LSTM) units as the RNN transition (Hochreiter and Schmidhuber, 1997).

In the rest of this section, we will describe a memory augmented RNN, and how it is used for incremental domain adaptation (IDA).

3.1 Augmenting RNN with Memory Banks

We enhance the RNN with an external memory bank, as shown in Figure 1b. The memory bank augments the overall model capacity by storing additional parameters in memory slots. At each time step, our model computes an attention probability to retrieve memory content, which is then fed to the computation of RNN transition.

Particularly, we adopt a key-value memory bank, inspired by Miller et al. (2016). Each memory slot contains a key vector and a value vector. The former is used to compute the attention weight for memory retrieval, whereas the latter is the value of memory content.

Let $\boldsymbol{h}_{i-1}$ be the RNN state of the last step $i-1$. For the ith step, the memory mechanism computes an attention probability $\boldsymbol{\alpha}_i$ by

$$\widetilde{\alpha}_{i,j} = \exp\{\boldsymbol{h}_{i-1}^\top \boldsymbol{m}_j^{(\text{key})}\} \tag{2}$$

$$\alpha_{i,j} = \frac{\widetilde{\alpha}_{i,j}}{\sum_{j'=1}^{N} \widetilde{\alpha}_{i,j'}} \tag{3}$$

where $\boldsymbol{m}_j^{(\text{k})}$ is the key vector of the jth slot of the memory (among N slots in total). Then the model retrieves memory content by a weighted sum of all memory values, where the weight is the attention probability, given by

$$\boldsymbol{c}_i = \sum_{j=1}^{N} \alpha_{i,j} \boldsymbol{m}_j^{(\text{val})} \tag{4}$$

Here, $\boldsymbol{m}_j^{(\text{val})}$ is the value vector of the jth memory slot. We call $\boldsymbol{c}_i$ the *memory content*. Then, $\boldsymbol{c}_i$ is concatenated with the current word $\boldsymbol{x}_i$, and fed to the RNN as input of step i to compute RNN state transition.

Using the key-value memory banks allows separate (thus more flexible) computation of memory retrieval weights and memory content, compared with traditional attention where a candidate vector is used to compute both attention probability and attention content.

Algorithm 1: Progressive Memory for IDA

```
Input: A sequence of domains $D_0, D_1, \cdots, D_n$
Output: A model performing well on all domains
Initialize a memory-augmented RNN
Train the model on $D_0$
for $i = 1, \cdots, n$ do
    Expand the memory with several new slots
    Derive RNN weights and existing memory weights
    Randomly initialize new slots' weights
    Train the model by updating all parameters
end
Return: The resulting model
```

It should be emphasized that the memory bank in our model captures distributed knowledge, which is different from other work where the memory slots correspond to specific entities (Eric et al., 2017). The attention mechanism accomplishes memory retrieval in a "soft" manner, which means the retrieval strength is a real-valued probability. This enables us to train the memory retrieval end-to-end, along with the other neural parameters.

We would also like to point out that the memory bank alone does not help RNN much. However, it is natural to use memory-augmented RNN for incremental domain adaptation, as described below.

3.2 Progressively Increasing Memory for Incremental Domain Adaptation (IDA)

The memory bank in Subection 3.1 can be progressively expanded to adapt a model in a souce domain to new domains. This is done by adding new memory slots to the bank which learn exclusively from the target data.

Concretely, suppose the memory bank is expanded with another M slots in a new domain in addition to previous N slots. We then have $N+M$ slots in total. The model computes attention probability over the expanded memory and obtains the attention vector in the same way as Equations (2)–(4), except that the summation is computed from 1 to $N + M$. This is given by

$$\alpha_{i,j}^{(\text{expand})} = \frac{\widetilde{\alpha}_{i,j}}{\sum_{j'=1}^{N+M} \widetilde{\alpha}_{i,j'}} \quad (5)$$

$$\boldsymbol{c}_i^{(\text{expand})} = \sum_{j=1}^{N+M} \alpha_{i,j}^{(\text{expand})} \boldsymbol{m}_j^{(\text{val})} \quad (6)$$

To initialize the expanded model, we load all previous parameters, including RNN weights and the learned N slots, but randomly initialize the progressively expanded M slots. During training, we update all parameters by gradient descent. That is to say, new parameters are learned from their initializations, whereas old parameters are fine-tuned during IDA. The process is applied whenever a new domain comes, shown in Algorithm 1.

We would like to discuss the following issues.

Fixing vs. fine-tuning learned parameters. Inspired by the progressive neural network (Rusu et al., 2016), we find it tempting to fix RNN parameters and the learned memory but only tune new memory for IDA. However, empirical results show that if we fix all existing parameters, the increased memory does not add much to the model capacity, and that its performance is worse than fine-tuning all parameters.

Fine-tuning vs. fine-tuning while increasing memory slots. It is reported that fine-tuning a model (without increasing model capacity) suffers from the problem of catastrophic forgetting (Kirkpatrick et al., 2017). It could be a concern if our approach suffers from the same problem, since we fine-tune learned parameters when progressively increasing memory slots. Our intuition is that the knowledge in the new domain could be stored in the increased model capacity, and thus it less overrides the learned model. Experiments confirm our conjecture, as the memory-augmented RNN tends to forget more if the memory size is not increased.

Expanding hidden states vs. Expanding memory. An alternative way of progressively increasing model capacity is to expand the size of the RNN layers. This setting is similar to the progressive neural network, except that all weights are fine-tuned and that we have connections from new states to existing states.

However, we hereby show a theorem, indicating that the expanded hidden states will contaminate the learned RNN more than the expanded memory.

Theorem 1. *Let RNN have vanilla transition with the linear activation function, and let the RNN state at the last step $\boldsymbol{h}_{t-1}$ be fixed. For a particular data point, if the memory attention satisfies $\sum_{i=N+1}^{N+M} \widetilde{\alpha}_{t,i} \le \sum_{i=1}^{N} \widetilde{\alpha}_{t,i}$, then memory expansion yields a lower expected mean squared difference in $\boldsymbol{h}_t$ than RNN state expansion, under reasonable assumptions. That is,*

$$\mathbb{E}\left[\|\boldsymbol{h}_t^{(\text{m})} - \boldsymbol{h}_t\|^2\right] \le \mathbb{E}\left[\|\boldsymbol{h}_t^{(\text{s})} - \boldsymbol{h}_t\|^2\right] \quad (7)$$

where $\boldsymbol{h}_t^{(\text{m})}$ refers to the hidden states if the memory is expanded, and $\boldsymbol{h}_t^{(\text{s})}$ refers to the original di-

	Fic	Gov	Slate	Tel	Travel
# training samples	77k	77k	77k	83k	77k
Our Implementation	65.0	66.5	56.2	64.5	62.7
Yu et al. (2018)	64.7	69.2	57.9	64.4	65.8

Table 1: Corpus statistics and the baseline performance (% accuracy) of our BiLSTM model (without domain adaptation), compared with results reported in previous work.

mensions of the RNN states, if we expand the size of RNN states themselves.

Proof: See Appendix A. □

The condition $\sum_{i=N+1}^{N+M} \widetilde{\alpha}_{t,i} \leq \sum_{i=1}^{N} \widetilde{\alpha}_{t,i}$ in our theorem requires that the total attention to existing memory slots is larger than to the progressively added slots. This is a reasonable assumption because: (1) During training, attention is trained in an *ad hoc* fashion to align information, and thus some of $\alpha_{t,i}$ for $1 \leq i \leq N$ might be learned so that it is larger than a random memory slot; and (2) For a new domain, we do not add a huge number of slots, and thus $\sum_{i=N+1}^{N+M}$ will not dominate. In the appendix, we will prove the theorem with some other assumptions (e.g., all weights are independent and identically distributed).

It is noted that our theorem does not explicitly prove results for IDA, but shows that expanding memory is more stable than expanding hidden states. This is particularly important at the beginning steps of IDA, as the progressively growing parameters are randomly initialized and are basically noise. Expanding memory slots will less contaminate the learned RNN, compared with expanding hidden states.

4 Experiments

4.1 Dataset and Setup

We evaluate our approach on the natural language inference task. The task is to determine the relationship between two sentences. The target labels include *entailment*, *contradiction*, and *neutral*.

We use the multi-genre natural language inference (MultiNLI) corpus (Williams et al., 2018) as our data. MultiNLI is particularly suitable for IDA, as it contains training samples for 5 genres: `Fiction`, `Government`, `Slate`, `Telephone`, and `Travel`. In total, we have 390k training samples. The corpus also contains a held-out (non-training) set of data samples with labels. We split it into two parts for validation and test.[2]

#Line	Model	Trained on/by	Performance on	
			S	T
1	RNN	S	65.01	61.23
2	RNN	T	56.46	66.49
3	RNN+Mem	S	65.41	60.87
4	RNN+Mem	T	56.77	67.01
5	RNN+Mem	S+T	66.02	70.00
6	RNN+Mem	S→T (F)	65.62	69.90
7	RNN+Mem	S→T (F+M)	66.23	70.21
8	RNN+Mem	S→T (F+M+V)	**67.55**	**70.82**
9	RNN+Mem	S→T (F+H)	64.09	68.35
10	RNN+Mem	S→T (F+H+V)	63.68	68.02
11	RNN+Mem	S→T (EWC)	66.02	64.10
12	RNN+Mem	S→T (Progressive)	64.47	68.25

Table 2: Results (% accuracy) on two domain adaptation. F: Fine-tuning. V: Expanding vocabulary. H: Expanding RNN hidden states. M: Our proposed method of expanding memory.We also compare with previous work elastic weight consolidation (EWC, Kirkpatrick et al., 2017) and the progressive neural network (Rusu et al., 2016).

The first row in Table 1 shows the size of the training set in each domain. As seen, the corpus is mostly balanced across domains, although `Tel` has slightly more examples.

For the base model, we train a bi-directional LSTM (BiLSTM), where we have 300-dimensional RNN hidden states. We use pretrained GloVe embeddings (Pennington et al., 2014) for initialization. We train our model by Adam with learning rate 0.0003 and other default hyperparameters. The batch size is 32. We see in Table 1 that we achieve similar performance to Yu et al. (2018), showing that our implementation and tuning are fair, and that our model is ready for the study of IDA.

For the memory part, we set each slot to be 300-dimensional, and for each domain, we progressively add 500 slots.

4.2 Transfer between Two Domains

We first analyze our approach, baselines, and a number of variants on two domains. Particularly,

[2]MultiNLI also contains 5 genres without training samples, namely, `9/11`, `Face-to-face`, `Letters`, `OUP`, and `Verbatim`. We ignore these genres, because we focus on incremental domain adaptation instead of zero-shot learning. Also, the official test set of MultiNLI requires submitting results to Kaggle. We find it unfeasible as we have too many settings during IDA. Our split of the held-out set for validation and test applies to all competing methods, and thus is a fair setting.

Training domains	Performance on				
	Fic	Gov	Slate	Tel	Travel
Fic	65.41	58.87	55.83	61.39	57.35
Fic → Gov	67.55	70.82	61.04	65.07	61.90
Fic → Gov → Slate	67.04	71.55	63.29	64.66	63.53
Fic → Gov → Slate → Tel	68.46	71.10	63.39	**71.60**	61.50
Fic → Gov → Slate → Tel → Travel	**69.36**	**72.47**	**63.96**	69.74	**68.39**

Table 3: Dynamics of the progressive memory network for IDA with 5 domains. Upper-triangular values in gray are out-of-domain (zero-shot) performance.

Setting	Fic	Gov	Slate	Tel	Travel
In-domain training	65.41	67.01	59.30	67.20	64.70
Fic + Gov + Slate + Tel + Travel (multi-task)	**70.60**	**73.30**	63.80	69.15	67.07
Fic → Gov → Slate → Tel → Travel (F+V)	67.24	70.82	62.41	67.62	**68.39**
Fic → Gov → Slate → Tel → Travel (F+V+M)	*69.36*	*72.47*	**63.96**	**69.74**	**68.39**
Fic → Gov → Slate → Tel → Travel (EWC)	67.12	68.71	59.90	66.09	65.70
Fic → Gov → Slate → Tel → Travel (Progressive NN)	65.22	67.87	61.13	66.96	67.90

Table 4: Comparing our approach with variants and previous work in the multi-domain setting. In this experiment, we use the memory-augmented RNN as the neural architecture.

we choose `Fic` (short for `Fiction`) as the source domain and `Gov` (short for `Government`) as the target domain. We show results in Table 2

First, we compare the performance of RNN and the memory augmented RNN in the non-transfer setting (Lines 1–2 vs. Lines 3–4). As seen, the memory-augmented RNN achieves slightly better but generally similar performance, compared with RNN (both with LSTM units). This shows that, in the non-transfer setting, the memory bank does not help RNN much and thus is not a typical RNN architecture in previous literature. However, this later confirms that the performance improvement is indeed due to the our IDA technique, instead of simply a better neural architecture.

We then apply two straightforward methods of domain adaptation: multi-task learning (Line 5) and fine-tuning (Line 6). Multi-task learning jointly optimizes source and target objectives, denoted at "S+T." On the other hand, The fine-tuning approach trains the model on the source first, and then fine-tunes on the target. In our experiments, these two methods perform similarly on the target domain, which is consistent with Mou et al. (2016). On the source domain, fine-tuning performs significantly worse than multi-task learning, as it suffers from the catastrophic forgetting problem. We notice that, in terms of source performance, the fine-tuning approach (Line 6) is slightly better than trained on the source domain only (Line 3). This is probably because our domains are highly correlated as opposed to Kirkpatrick et al. (2017), and thus training with more data on target improves the performance on source. However, fine-tuning does achieve the worst performance on source compared with other domain adaptation approaches (among Lines 5–8). Thus, we nevertheless use the terminology "catastrophic forgetting," and our research goal is still to improve IDA performance.

The main results of our approach are Lines 7 and 8. We apply the proposed progressive memory network to IDA and we fine-tune all weights. We see that on both source and target domains our approach outperforms the fine-tuning method alone where the memory size is not increased (comparing Lines 7 and 6). This verifies our conjecture that, if the model capacity is increased, the new domain does not override the learned knowledge in the neural network. Our proposed approach is also "orthogonal" to the expansion of the vocabulary size, where target-specific words are randomly initialized and learned on the target domain. As seen, this combines well with our memory expansion and yields the best performance on both source and target (Line 8).

We now compare an alternative way of increasing model capacity, i.e., expanding hidden states (Lines 9 and 10). In this case, its performance is poor especially on the source domain, even if we fine-tune all parameters. This experiment provides empirical evidence to our theorem that expanding memory is more robust than expanding hidden states.

We also compare the results with previous work on IDA. We re-implement[3] elastic weight consolidation (EWC, Kirkpatrick et al., 2017). It does not achieve satisfactory results in our application. We investigate other published papers using the same method and find inconsistent results: EWC works well in some applications (Lee et al., 2017) but performs poorly on others (Yoon et al., 2018). We also re-implement the progressive neural network (Rusu et al., 2016). We use the target predictor to do inference for both source and target domains. The progressive neural network yields low performance, particularly on source, probably because the predictor is trained with only the target domain.

4.3 IDA with All Domains

Having analyzed our approach, baselines, and variants in detail, we are now ready to test the performance of IDA with multiple domains, namely, `Fic`, `Gov`, `Slate`, `Tel`, and `Travel`. In this experiment, we assume these domains come one after another, and our goal is to achieve high performance on both new and previous domains.

Table 3 shows the dynamics of IDA with our progressive memory network. Comparing the upper-triangular values (in gray, showing out-of-domain performance) with diagonal values, we see that our approach can be quickly adapted to the new domain in an incremental fashion. Comparing lower-triangular values with the diagonal, we see that our approach does not suffer from the catastrophic forgetting problem as the performance of previous domains is gradually increasing if trained with more domains. After all data are observed, our model achieves the best performance in most domains (last row in Table 3), despite the incremental nature of our approach.

We now compare our approach with other baselines and variants in the multi-domain setting, shown in Table 4. Due to the large number of settings, we only choose a selected subset of variants from Table 2 for the comparison.

As seen, our approach of progressively growing memory bank achieves the same performance as fine-tuning on the last domain (with with vocabulary expansion). But for all previous 4 domains, we achieve better performance. Our model is comparable to multi-task learning on all domains. This provides evidence of the effectiveness for IDA with more than two domains.

It should also be mentioned that multi-task learning requires training the model when data from all domains are available at the same time. It is not an *incremental* approach for domain adaptation, and thus cannot be applied to the scenarios introduced in Section 1. We include this setting mainly because we are curious about the performance of non-incremental domain adaptation.

We also compare with previous methods for IDA in Table 4. Our method outperforms EWC (Lee et al., 2017) and the progressive neural network (Rusu et al., 2016) in all domains; the results are consistent with Table 2.

5 Conclusion

In this paper, we propose a progressive memory network for incremental domain adaptation (IDA). We augment an RNN with an attention-based memory bank. During IDA, we add new slots to the memory bank and tune all parameters by back-propagation. Empirically, the progressive memory network does not suffer from the catastrophic forgetting problem as in naïve fine-tuning. Our intuition is that the new memory slots increase the neural network's model capacity, and thus the new knowledge less overrides the existing network. Compared with expanding hidden states, our progressive memory bank provides a more robust way of increasing model capacity, shown by both a theorem and experiments. We also outperform previous work for IDA, including elastic weight consolidation (EWC) and the original progressive neural network.

References

Justin Bayer, Christian Osendorfer, Daniela Korhammer, Nutan Chen, Sebastian Urban, and Patrick van der Smagt. 2013. On fast dropout and its applicability to recurrent networks. *arXiv preprint arXiv:1311.0701*.

Rajarshi Das, Manzil Zaheer, Siva Reddy, and Andrew McCallum. 2017. Question answering on knowledge bases and text using universal schema and memory networks. In *ACL*, pages 358–365.

Mihail Eric, Lakshmi Krishnan, Francois Charette, and Christopher D. Manning. 2017. Key-value retrieval networks for task-oriented dialogue. In *SIGDIAL*, pages 37–49.

Alex Graves, Greg Wayne, Malcolm Reynolds, Tim Harley, Ivo Danihelka, Agnieszka Grabska-Barwińska, Sergio Gómez Colmenarejo, Edward

[3]Our implementation is based on `https://github.com/ariseff/overcoming-catastrophic`

Grefenstette, Tiago Ramalho, John Agapiou, et al. 2016. Hybrid computing using a neural network with dynamic external memory. *Nature*, 538(7626):471–476.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural Computation*, 9(8):1735–1780.

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

Sang-Woo Lee, Jin-Hwa Kim, Jaehyun Jun, Jung-Woo Ha, and Byoung-Tak Zhang. 2017. Overcoming catastrophic forgetting by incremental moment matching. In *NIPS*, pages 4652–4662.

Pengfei Liu, Xipeng Qiu, and Xuanjing Huang. 2017. Adversarial multi-task learning for text classification. In *ACL*, pages 1–10.

Andrea Madotto, Chien-Sheng Wu, and Pascale Fung. 2018. Mem2seq: Effectively incorporating knowledge bases into end-to-end task-oriented dialog systems. In *ACL*, pages 1468–1478.

Alexander Miller, Adam Fisch, Jesse Dodge, Amir-Hossein Karimi, Antoine Bordes, and Jason Weston. 2016. Key-value memory networks for directly reading documents. In *EMNLP*, pages 1400–1409.

Lili Mou, Zhao Meng, Rui Yan, Ge Li, Yan Xu, Lu Zhang, and Zhi Jin. 2016. How transferable are neural networks in NLP applications? In *EMNLP*, pages 479–489.

Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. Glove: Global vectors for word representation. In *EMNLP*, pages 1532–1543.

Andrei A Rusu, Neil C Rabinowitz, Guillaume Desjardins, Hubert Soyer, James Kirkpatrick, Koray Kavukcuoglu, Razvan Pascanu, and Raia Hadsell. 2016. Progressive neural networks. *arXiv preprint arXiv:1606.04671*.

Sainbayar Sukhbaatar, Jason Weston, Rob Fergus, et al. 2015. End-to-end memory networks. In *NIPS*, pages 2440–2448.

Adina Williams, Nikita Nangia, and Samuel Bowman. 2018. A broad-coverage challenge corpus for sentence understanding through inference. In *NAACL-HLT*, pages 1112–1122.

Jiaolong Xu, Sebastian Ramos, David Vázquez, Antonio M López, and D Ponsa. 2014. Incremental domain adaptation of deformable part-based models. In *BMVC*.

Jaehong Yoon, Eunho Yang, Jeongtae Lee, and Sung Ju Hwang. 2018. Lifelong learning with dynamically expandable networks. *ICLR*.

Jianfei Yu, Minghui Qiu, Jing Jiang, Jun Huang, Shuangyong Song, Wei Chu, and Haiqing Chen. 2018. Modelling domain relationships for transfer learning on retrieval-based question answering systems in e-commerce. In *WSDM*, pages 682–690.

Zheng Zhang, Minlie Huang, Zhongzhou Zhao, Feng Ji, Haiqing Chen, and Xiaoyan Zhu. 2018. Memory-augmented dialogue management for task-oriented dialogue systems. *arXiv preprint arXiv:1805.00150*.

A Appendix

Theorem 1. *Let RNN have vanilla transition with the linear activation function, and let the RNN state at the last step $\boldsymbol{h}_{t-1}$ be fixed. For a particular data point, if the memory attention satisfies $\sum_{i=1+1}^{N+M} \widetilde{\alpha}_{t,i} \leq \sum_{i=1}^{N} \widetilde{\alpha}_{t,i}$, then memory expansion yields a lower expected mean squared difference in $\boldsymbol{h}_t$ than RNN state expansion, under reasonable assumptions. That is,*

$$\mathbb{E}\left[\|\boldsymbol{h}_t^{(\mathrm{m})} - \boldsymbol{h}_t\|^2\right] \leq \mathbb{E}\left[\|\boldsymbol{h}_t^{(\mathrm{s})} - \boldsymbol{h}_t\|^2\right] \tag{8}$$

where $\boldsymbol{h}_t^{(\mathrm{m})}$ refers to the hidden states if the memory is expanded, and $\boldsymbol{h}_t^{(\mathrm{s})}$ refers to the original dimensions of the RNN states, if we expand the size of RNN states themselves.

Proof: We first make a few assumptions. Let $\boldsymbol{h}_{t-1}$ be the hidden state of the last step. We focus on one step of transition and assume that $\boldsymbol{h}_{t-1}$ is the same when the model capacity is increased. We consider a simplified case where the RNN has vanilla transition with the linear activation function. We measure the effect of model expansion quantitatively by the expected norm of the difference on $\boldsymbol{h}_t$ before and after model expansion.

Suppose the original hidden state $\boldsymbol{h}_t$ is D-dimensional. For either a memory slot or expanded hidden state, we assume it is d-dimensional. We further assume every variable in the expanded memory and expanded weights ($\widetilde{W}$ in Figure 2) are iid with zero mean and variance σ^2. This assumption is reasonable as it enables a fair comparison of expanding memory and expanding hidden states. Finally, we assume every variable in the learned memory slots, i.e., m_{ij}, follows the same distribution (zero mean, variance σ^2). This assumption may not be true after the network is trained, but is useful for proving theorems.

We compute how the original dimensions in the hidden state are changed if we expand RNN. We denote the expanded hidden states by $\widetilde{\boldsymbol{h}}_{t-1}$ and $\widetilde{\boldsymbol{h}}_t$ for the two time steps. We denote the weights connecting from $\widetilde{\boldsymbol{h}}_{t-1}$ to $\boldsymbol{h}_t$ by $\widetilde{W} \in \mathbb{R}^{D\times d}$. We focus on the original D-dimensional space, denoted as $\boldsymbol{h}_t^{(s)}$. The connection is shown in Figure 2a. We have

$$\begin{aligned}
&\mathbb{E}\left[\|\boldsymbol{h}_t^{(\mathrm{s})} - \boldsymbol{h}_t\|^2\right] \\
&= \mathbb{E}\left[\|\widetilde{W}\cdot \widetilde{\boldsymbol{h}}_{t-1}\|^2\right] && (9)\\
&= \mathbb{E}\left[\sum_{j=1}^{D}\left(\widetilde{\boldsymbol{w}}_j^\top \widetilde{\boldsymbol{h}}_{t-1}\right)^2\right] && (10)\\
&= \sum_{j=1}^{D}\mathbb{E}\left[\left(\widetilde{\boldsymbol{w}}_j^\top \widetilde{\boldsymbol{h}}_{t-1}\right)^2\right] && (11)\\
&= \sum_{j=1}^{D}\mathbb{E}\left[\left(\sum_{i=1}^{d}\widetilde{w}_{ij}\widetilde{h}_{t-1}[i]\right)^2\right] && (12)\\
&= \sum_{j=1}^{D}\sum_{i=1}^{d}\mathbb{E}\left[\left(\widetilde{w}_{ij}\widetilde{h}_{t-1}[i]\right)^2\right] && (13)\\
&= \sum_{j=1}^{D}\sum_{i=1}^{d}\mathbb{E}\left[\left(\widetilde{w}_{ij}\right)^2\right]\mathbb{E}\left[\left(\widetilde{h}_{t-1}[i]\right)^2\right] && (14)\\
&= D\cdot d\cdot \mathrm{Var}(w)\cdot \mathrm{Var}(h) && (15)\\
&= Dd\sigma^2\sigma^2 && (16)
\end{aligned}$$

where (13) is due to the independence and zero-mean assumptions of every element in $\widetilde{W}$ and $\boldsymbol{h}_{t-1}$. (14) is due to the independence assumption between $\widetilde{W}$ and $\boldsymbol{h}_{t-1}$.

Next, we compute the effect of expanding memory slots. Notice that $\|\boldsymbol{h}_t^{(\mathrm{m})} - \boldsymbol{h}_t\| = W_{(\mathrm{c})}\Delta\boldsymbol{c}$. Here, $\boldsymbol{h}_t^{(m)}$ is the RNN hidden state after memory expansion. $\Delta\boldsymbol{c}_t \overset{\mathrm{def}}{=} \boldsymbol{c}' - \boldsymbol{c}$, where $\boldsymbol{c}$ and $\boldsymbol{c}'$ are the attention content vectors before and after memory expansion, respectively, at the current time step.[4] $W_{(\mathrm{c})}$ is weight connecting attention content to RNN states. The connection is shown

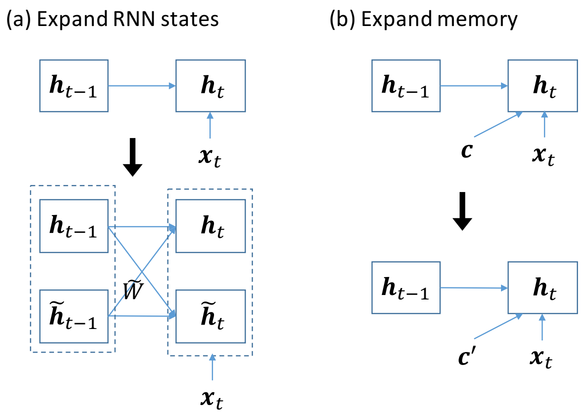

Figure 2: Hidden state expansion vs. memory expansion at step t.

[4]We omit the time step t in the notation for simplicity.

in Figure 2b. Reusing the result of (15), we immediately obtain

$$\mathbb{E}\left[\|\boldsymbol{h}_t^{(\mathrm{m})} - \boldsymbol{h}_t\|^2\right] \tag{17}$$
$$= \mathbb{E}\left[\|W_{(c)}\Delta\boldsymbol{c}\|^2\right] \tag{18}$$
$$= Dd\sigma^2 \mathrm{Var}(\Delta c_j) \tag{19}$$

where Δc_j is an element of $\Delta\boldsymbol{c}$.

To prove Equation (2), it remains to show that $\mathrm{Var}(\Delta c_j) \le \sigma^2$. We now analyze how attention is computed.

Let $\widetilde{\alpha}_1, \cdots, \widetilde{\alpha}_{N+M}$ be the unnormalized attention weights over the $N+M$ memory slots. We notice that $\widetilde{\alpha}_1, \cdots, \widetilde{\alpha}_N$ remain the same after memory expansion. Then, the original attention probability is given by $\alpha_i = \widetilde{\alpha}_i/(\widetilde{\alpha}_1 + \cdots + \widetilde{\alpha}_N)$ for $i = 1, \cdots, N$. After memory expansion, the attention probability becomes $\alpha'_i = \widetilde{\alpha}_i/(\widetilde{\alpha}_1 + \cdots + \widetilde{\alpha}_{N+M})$, illustrated in Figure 3.

$$\Delta\boldsymbol{c} = \boldsymbol{c}' - \boldsymbol{c} \tag{20}$$
$$= \sum_{i=1}^{N}(\alpha'_i - \alpha_i)\boldsymbol{m}_i + \sum_{i=N+1}^{N+M}\alpha'_i\boldsymbol{m}_i \tag{21}$$
$$= \sum_{i=1}^{N}\left(\frac{\widetilde{\alpha}_i}{\widetilde{\alpha}_1 + \cdots + \widetilde{\alpha}_{N+M}} - \frac{\widetilde{\alpha}_i}{\widetilde{\alpha}_1 + \cdots + \widetilde{\alpha}_N}\right)\boldsymbol{m}_i + \sum_{i=N+1}^{N+M}\left(\frac{\widetilde{\alpha}_i}{\widetilde{\alpha}_1 + \cdots + \widetilde{\alpha}_{N+M}}\right)\boldsymbol{m}_i \tag{22}$$
$$= \sum_{i=1}^{N}\left(\frac{-\widetilde{\alpha}_i\frac{\widetilde{\alpha}_{N+1}+\cdots+\widetilde{\alpha}_{N+M}}{\widetilde{\alpha}_1+\cdots+\widetilde{\alpha}_N}}{\widetilde{\alpha}_1 + \cdots + \widetilde{\alpha}_{N+M}}\right)\boldsymbol{m}_i + \sum_{i=N+1}^{N+M}\left(\frac{\widetilde{\alpha}_i}{\widetilde{\alpha}_1 + \cdots + \widetilde{\alpha}_M}\right)\boldsymbol{m}_i \tag{24}$$
$$= \sum_{i=1}^{N+M}\beta_i\boldsymbol{m}_i \tag{25}$$

where

$$\beta_i \overset{\text{def}}{=} \begin{cases} \dfrac{-\widetilde{\alpha}_i\frac{\widetilde{\alpha}_{N+1}+\cdots+\widetilde{\alpha}_{N+M}}{\widetilde{\alpha}_1+\cdots+\widetilde{\alpha}_N}}{\widetilde{\alpha}_1 + \cdots + \widetilde{\alpha}_{N+M}}, & \text{if } 1 \le i \le N \\ \dfrac{\widetilde{\alpha}_i}{\widetilde{\alpha}_1 + \cdots + \widetilde{\alpha}_{N+M}}, & \text{if } N+1 \le i \le N+M \end{cases} \tag{26}$$

By our assumption of total attention $\sum_{i=N+1}^{N+M}\widetilde{\alpha}_i \le \sum_{i=1}^{N}\widetilde{\alpha}_i$, we have

$$|\beta_i| \le |\alpha'_i|, \quad \forall 1 \le i \le N+M \tag{27}$$

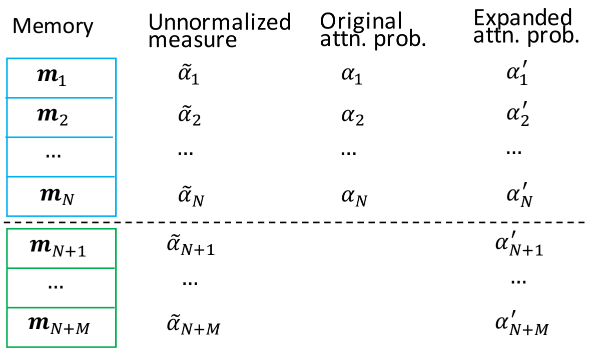

Figure 3: Attention probabilities before and after memory expansion.

Then, we have

$$\mathrm{Var}(\Delta c_j) = \mathbb{E}[c'_j - c_j] \qquad \forall 1 \le j \le d \tag{28}$$
$$= \frac{1}{d}\mathbb{E}\left[\|\boldsymbol{c}'_t - \boldsymbol{c}_t\|^2\right] \tag{29}$$
$$= \frac{1}{d}\mathbb{E}\left[\sum_{k=1}^{d}\left(\sum_{i=1}^{N+M}\beta_i m_{ik}\right)^2\right] \tag{30}$$
$$= \frac{1}{d}\sum_{k=1}^{d}\mathbb{E}\left[\left(\sum_{i=1}^{N+M}\beta_i m_{ik}\right)^2\right] \tag{31}$$
$$= \frac{1}{d}\sum_{k=1}^{d}\sum_{i=1}^{N+M}\mathbb{E}\left[\left(\beta_i m_{ik}\right)^2\right] \tag{32}$$
$$= \frac{1}{d}\sum_{k=1}^{d}\sum_{i=1}^{N+M}\mathbb{E}\left[\beta_i^2\right]\mathbb{E}\left[m_{ik}^2\right] \tag{33}$$
$$= \frac{1}{d}\sum_{k=1}^{d}\sum_{i=1}^{N+M}\mathbb{E}[\beta_i^2]\sigma^2 \tag{34}$$
$$= \sigma^2\,\mathbb{E}\left[\sum_{i=1}^{N+M}\beta_i^2\right] \tag{35}$$
$$\le \sigma^2\,\mathbb{E}\left[\sum_{i=1}^{N+M}(\alpha'_i)^2\right] \tag{36}$$
$$\le \sigma^2 \tag{37}$$

Here, (31) is due to the assumption that m_{ik} is independent and zero-mean, and (32) is due to the indepenence assumption between β_i and m_{ik}. To obtain (37), we notice that $\sum_{i=1}^{N+M}\alpha_i = 1$ with $0 \le \alpha_i \le 1$ ($\forall 1 \le i \le N+M$). Thus, $\sum_{i=1}^{N+M}(\alpha'_i)^2 \le 1$, concluding our proof. $\square$