Labeled Memory Networks for Online Model Adaptation

Shiv Shankar
shiv_shankar@iitb.ac.in
IIT Bombay

Sunita Sarawagi
sunita@iitb.ac.in
IIT Bombay

Abstract

Augmenting a neural network with memory that can grow without growing the number of trained parameters is a recent powerful concept with many exciting applications. In this paper, we establish their potential in online adapting a batch trained neural network to domain-relevant labeled data at deployment time.

We present the design of Labeled Memory Network (LMN), a new memory augmented neural network (MANN) for fast online model adaptation. We highlight three key features of LMNs. First, LMNs treat memory as a second boosted stage following the trained network thereby allowing the memory and network to play complementary roles. Unlike all existing MANNs that write to memory at every cycle, LMNs provide better memory utilization by writing only labeled data with non-zero loss. Second, LMNs organize the memory with the discrete class label as the primary key unlike existing MANNs where key is a real vector derived from the input. This simple, yet surprisingly unexplored alternative organization, safeguards against catastrophic forgetting of rare labels that current LRU based MANNs are subject to. Finally, LMNs model the evolving expertise of memory and network using a RNN, to determine online their respective weights

We evaluate online model adaptation strategies on five sequence prediction tasks, an image classification task, and two language modeling tasks. We show that LMNs are better than other MANNs designed for meta-learning. We also found them to be more accurate and faster than state-of-the-art methods of retuning model parameters for adapting to domain-specific labeled data.

Introduction

While deep learning models achieve impressive accuracy in supervised learning tasks such as computer vision (**?**), translation (**?**), and speech recognition (**?**) training such models places significant demands on the amount of labeled data, computational resources, and manual efforts in tuning. Training is thus considered an infrequently performed resource-intensive batch process. Many applications require trained models to quickly adapt to new examples and settings during deployment. Consider the online sequence prediction task that arises in applications such as text autocompletion, user trajectory prediction, or next-url prediction. Even when the prediction model has been trained on many sequences, next-token predictions on new sequences can benefit from true tokens observed online. Another example is image recognition where a trained model should be able to recognize new objects not seen during batch training based on a few examples. This has led to a surge of interest in few-shot learning (**?**; **?**; **?**; **?**). Few-shot learning is an artificially simplified setting where a small set of new labels define independent classification tasks. We consider the more useful but more difficult and less explored task where an existing image classifier has to be extended to handle new labels.

An established technique for model adaptation is to retune part or all of the model parameters using the domain-labeled data in a separate adaptation phase (**?**). More recently, deep meta-networks have been proposed that "learn to learn" such adaptation (**?**; **?**; **?**; **?**). Parameter retuning methods typically require a one-time adaptation step, and operating them in online settings is slow. Secondly with a pre-trained network the multiple gradient updates of meta-learning tend to destroy useful information learned by the PCN. This phenomenon is closely related to 'catastrophic forgetting' (**?**; **?**).

An alternative to parameter retuning is memorizing. In this approach neural networks are augmented with memory that can grow without correspondingly increasing the number of parameters to be trained. Many exciting uses have been found of such MANNs including program learning (**?**), question answering (**?**; **?**), learning rare events (**?**), and meta learning (**?**). They hold promise for model adaptation because memory they can cut short the conventional path of iterative training to percolate new facts to model parameters.

However, our initial attempts at using existing MANNs like NTMs (**?**) and DNTMs (**?**) for online sequence prediction did not improve the baseline model. One major challenge is correctly balancing the roles of the memory and batch-trained model. On the one hand, we have a shared model trained over several instances, and on the other hand we have the few but more relevant instances encountered during testing. Recent MANNs designed for meta-learning (**?**) partition their roles by using the shared model to learn an embedding and the memory to implement a nearest neighbor like classifier. While this architecture works for few-shot learning where all labels are new, they cannot adapt

classification models with softmax layers over a large shared label space.

Contributions In this paper we present a new MANN called Labeled Memory Network (LMN) that provides an easy, fast, and plug-and-play solution for adapting pre-trained models. We highlight three design decisions that made LMNs suitable for such adaptation.

First, we apply ideas from boosting (**?**) and treat memory as a second-stage classifier that is updated only when the current loss is non-zero. Existing MANNs write to memory during every pass. Even when the goal of the model is to use the memory to remember rare events (**?**), the memory stores all events not just the rare ones. This causes a lot of memory to be wasted in storing non-rare vectors often displacing the rare ones.

Second, we propose a 'labeled memory' where the primary means of addressing a memory cell is by a class label. This is in contrast to all existing MANNs that use a controller to generate a hidden vector of the input as key. This simple, yet surprisingly unexplored alternative organization[1] has two advantages: First, the controller is freed from the vaguely supervised task of generating keys that are distinctive and relevant leading to better memory use. Second, it safeguards against catastrophic forgetting of rare labels that current LRU based MANNs are subject to.

Third, we use the power of a RNN to adaptively define the roles of the memory and the neural network in a label dependent manner. This is unlike traditional boosting where the stage weights are fixed and derived based on simple functions of the error of each stage.

LMNs can be used for online adaptation in a variety of settings. We compare not only with existing MANNs but also state of the art meta-learning methods that retune parameters (**?**; **?**). On online sequence prediction tasks spanning five applications like user trajectory predictions and next-click prediction, we show significant gains. Second, we report higher accuracy in two different settings of the popular Omniglot image classification task. Finally, we present results from two popular language modeling benchmarks and report improvements over state of the art.

Online Model Adaptation using Labeled Memory Networks (LMNs)

Problem Description Online model adaptation kicks in at the time of deploying a batch trained model. Our focus is classification models in this paper. During deployment the model sees inputs one at a time in sequence. At a time t for the input $\mathbf{x}_t$ we predict the label $\hat{y}_t$, after which the true output y_t is revealed. The online model adapter decides how to improve the next input's $\mathbf{x}_{t+1}$ prediction by combining the limited labeled data $\{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_t, y_t)\}$ with the batch trained model. The adaptation has to be fast and performed synchronously at each step. We are agnostic about how the classifier is batch trained. However, we assume that the adapter can be trained using several such sequences representative of the deployment scenario. Many problems such as user trajectory prediction, language modeling, and few-shot learning can be cast in such formulation.

We describe our method of online adaptation in two phases. We first explain how LMNs make predictions for an input $\mathbf{x}_t$, and next how they adapt when the true label y_t is revealed. LMNs comprise of three components: the primary classification network (PCN), the memory module, and a combiner network. Our architecture is depicted graphically in 1a. We describe each component of the architecture next.

Primary Classification Network (PCN)

This refers to the batch trained neural network that we seek to adapt. The PCN may be stateful or stateless. For example, in applications like trajectory prediction the sequence of inputs are stateful, whereas in tasks like image classification the inputs are stateless. Our only assumption is that the last layer of PCN transform each $\mathbf{x}_t$ into a real vector $\mathbf{h}_t \in R^d$ before feeding to a softmax layer to predict a distribution over the class labels. We use $\boldsymbol{\beta}_y \in R^d$ to denote the softmax parameter for class y, so the score r_{ty} for predicting class y for input x_t is

$$r_{ty} = \frac{\exp(\boldsymbol{\beta}_y \mathbf{h}_t)}{\sum_z \exp(\boldsymbol{\beta}_z \mathbf{h}_t)} = \text{softmax}(\boldsymbol{\beta} \mathbf{h}_t) \tag{1}$$

Memory

The memory consists of N cells. Each cell m is a 3-tuple with: ℓ_m denoting the label of cell m, $\mathbf{v}_m$ denoting the hidden vector stored in m, α_m denoting a weight attached to the cell. This storage format is similar to what is used in existing MANNs. But the way in which we read, use, and update the memory is very different. In existing MANNs memory is viewed as a part of the main network. Memory values are read based on matching a key with the stored vector $\mathbf{v}_m$ and the read values are processed by the main network to produce the output. In contrast we view memory as a learner, specifically an online learner that is only loosely integrated with the PCN. We describe here our alternative design.

We index the memory with the class label ℓ_m so that given a label y, we can enumerate all cells with label y. The memory provides a score over each class label y for an input $\mathbf{x}_t$. We use the PCN last layer output $\mathbf{h}_t$ as an embedding of $\mathbf{x}_t$ on the basis of which we can compute the kernel between $\mathbf{h}_t$ and a memory vector as $K(\mathbf{h}_t, \mathbf{v}_m) = \exp(\lambda \text{ cosine}(\mathbf{h}_t, \mathbf{v}_m))$. Given $\mathbf{h}_t$ and y we read a vector M_{ty} along with a scalar weight α_{ty} calculated as

$$\begin{aligned} M_{ty}, \alpha_{ty} &= \sum_{m:\ell_m=y} w_{tm}\{\mathbf{v}_m, \alpha_m\} \\ w_{tm} &= \frac{K(\mathbf{h}_t, \mathbf{v}_m)}{\sum_{m':\ell_{m'}=y} K(\mathbf{h}_t, \mathbf{v}_{m'})} \end{aligned} \tag{2}$$

This method of reading is very similar to soft addressing used in most memory models. But the key difference is that we take average only over cells with label y, and not all cells.

[1] Our method of using labels for addressing is very different from the discrete addressing mechanism proposed in (**?**). Even though both result in discrete addressing, in our case the key is the label whereas in (**?**) the key continues to be computed from input $\mathbf{x}$.

This label specific vector M_{ty} read from memory is used to get a score for a class y as:

$$s_{ty} = \frac{\alpha_{ty}^{\delta} K(\mathbf{h}_t, M_{ty})}{\sum_{y'} \alpha_{ty'}^{\delta} K(\mathbf{h}_t, M_{ty'})} \quad (3)$$

where δ is a strength parameter. The memory can thus be viewed as a kernel classifier with α_{ty} denoting the weight of the kernel.

When memory is large, the time to compute memory scores could be a concern but it is easy to get fast approximate scores using a similarity index (**?**; **?**), and approximate nearest neighbor search has been used for large scale memories (**?**; **?**).

Combining memory and PCN

The final score for a label y is computed by combining pcn-scores r_{ty} and memory scores s_{ty} with an interpolation parameter θ. We made θ a dynamic variable, because we expect the relative importance of memory and PCN to evolve as more labeled data becomes available online. Following Adaboost (**?**), one way to choose θ as a function of the two classifier stages. However we obtained better results by using a RNN to determine θ as a function of label and time as well. The RNN works on a label dependent state and at each step takes three sets of input. First is the input embedding $\mathbf{h}_t$, second are binary indicators $e_{t-1}^{m}, e_{t-1}^{pcn}$ which indicate whether the memory and PCN had error at the previous output, and the third set are label probabilities predicted by the memory and PCN. The RNN acts at each step on the state to produce label dependent outputs, from which the relative weights of memory and PCN are obtained via a sigmoid layer.

$$\begin{aligned} \theta_{ty} &= \sigma(W_\theta \mu_{ty} + b_\theta) \\ \mu_{ty} &= \text{RNN}(\mu_{t-1,y}, \{\mathbf{h}_t, e_{t-1}^{pcn}, e_{t-1}^{m}, r_{t-1,y}, s_{t-1,y}\}) \end{aligned} \quad (4)$$

The final predicted distribution over the labels for an input $\mathbf{x}_t$ is

$$P_t(y|\mathbf{x}_t) \propto (1 - \theta_{ty}) r_{ty} + \theta_{ty} s_{ty} \quad (5)$$

Training the combiner network The combiner parameters are trained to minimize the loss on $P_t(y|\mathbf{x})$ over a labeled sequence of instances that are representative of the deployment setting. We take a pre-trained PCN and augment it with the memory and combiner modules. This new network is then trained in the online setting, by providing it data in a sequential manner.

An advantage of this architecture is that memory and PCN are loosely coupled, allowing the modules to be plugged in over existing models and can mix at varying levels depending on the amount of per-domain data. We show in the experimental section that even with θ_{ty} fixed to a single hyper parameter (over all t, y), LMNs are competitive with state of the art model adaptation methods.

Adapting to true label y_t

When the true label of $\mathbf{x}_t$ is revealed we take two steps: update the state of the combiner RNN as discussed in Equation 4, and write to memory if needed. We describe our method for memory updates next.

The memory is partitioned across labels, and each memory cell is a tuple of label, content and α. The way in which we update our memory is consistent with the role that memory serves of being a second classification stage that is updated online. We next discuss the three parts of our memory write strategy: when to write, how to write, what to replace and why. The overall algorithm is depicted in Figure 2.

When to write Like in online learners, we update memory only when the margin of separation between the scores of true and closest incorrect label is small. Specifically we update only if $\log P_t(y_t|\mathbf{x}_t) - \max_{y \neq y_t} \log(y|\mathbf{x}_t)$ is less than a margin threshold. For example, instances that are confidently classified by the PCN may never be fed to the memory. In contrast, existing MANNs write to memory for every instance and are prone to fill up the memory on cases that can be accurately handled by PCN.

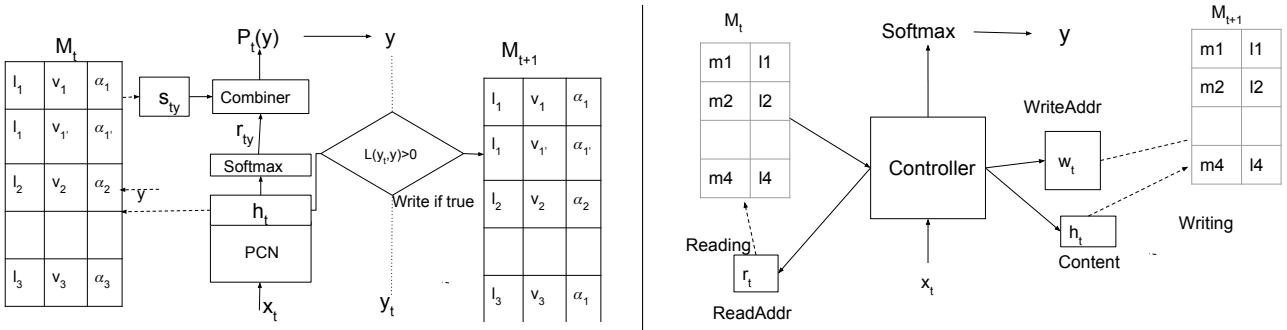
What to write Consider the content $\mathbf{v}_m$ and weight α_m associated with each memory cell m at time t. We update the content of cells with label y_t using $\mathbf{v}_m = \mathbf{v}_m + w_{tm}\mathbf{h}_t$ where w_{tm} is the fractional similarity of $\mathbf{h}_t$ to contents of cell m (Equation 2). The weight α_m of the cell is incremented by the fractional similarity w_{tm} to $\mathbf{h}_t$ after decaying old weights. Further, if the prediction is incorrect we attempt to create a new cell in memory by replacing an existing cell.

What cell to replace We replace the cell with the smallest weight among all cells with the same label as y_t. Since we update α_m based on its fractional similarity to instances that had non-zero loss, this method essentially replaces the cell that is least useful for defining the classification task. Unlike existing MANNs that replace a cell least recently used across all labels, we replace only among cells with label y_t. A pure LRU based replacement could easily lead to forgetting of rare classes. Our method of replacement enables us to implement a fairer classifier atop the memory.

Our memory updates are analogous to gradient updates in an online SVM learner. However we operate in a limited memory setting and merge and delete support vectors similar to budgeted-PEGASOS (**?**). If memory vectors are considered as SVM parameters, then the gradient w.r.t these parameters of the objective is precisely the current vector $\mathbf{h}$. In an online SVM learner when the loss is non-zero, the input element is added to the set of support vectors. Similarly in LMN, when margin loss is non-zero, the current vector $\mathbf{h}$ is added to the memory. The contribution of this specific memory update for the memory scores in subsequent steps will then be proportional to $\alpha_{ty}^{\delta} K(\mathbf{h}_t, \mathbf{h})$. This acts as an online-SVM learner in the dual form, where the $\mathbf{h}$ are the support vectors, and α act as the associated dual variables.

Related Work

Memory in neural networks The earliest example of the use of memory in neural networks is attention (**?**). Attention as memory is slow for long histories, leading to the



(a) LMN (b) Memory network

Figure 1: Comparing Memory Networks and LMN

Input: $y_t, P_t(y|\mathbf{x}_t), \mathbf{h}_t, M_t = \{(\ell_m, \mathbf{h}_m, \alpha_m)\}$
$\hat{y} = \text{argmax}_y P_t(y|\mathbf{x}_t)$
$\tilde{y} = \text{argmax}_{y \neq y_t} P_t(y|\mathbf{x}_t)$
$loss_t = -\min(0, \log P_t(y_t|\mathbf{x}_t) - \log P_t(\tilde{y}|\mathbf{x}_t) - \text{margin})$
If $loss_t = 0$, no update. Return.
New cell C: $(\ell_{y_t}, \mathbf{h}_t, 1)$
$j = \text{argmin}_{m:\ell_m = y_t} \alpha_m$
$\mathbf{w}_t = \text{softmax}\{\text{Cosine}(\mathbf{h}_t^T \mathbf{v}_m) : \ell_m = y_t\}$
$\mathbf{v}_m = \mathbf{v}_m + w_{tm}\mathbf{h}_t \quad \forall m : \ell_m = y_t$
$\alpha_m = \alpha_m * \text{decay} + w_{tm} \quad \forall m : \ell_m = y_t$
if $y_t \neq \hat{y}$ and $|m : \ell_m = y_t| > 1$ **then**
  Replace cell j with new cell C.
end if

Figure 2: Memory updates in Labeled memory network. In our experiments we used a decay value of 0.99. The margin is a hyper-parameter.

development of several more flexible memory-based architectures (**?**). Neural Turing Machines (NTMs) (**?**) were developed for end-to-end learning of algorithmic tasks. One such task where NTMs were shown to work was learning N-gram distribution from token sequences. Since this is related to online sequence prediction, our first model was based on NTMs. However, on our real datasets we found NTMs to not be very effective. The reasons perhaps is the controller's difficulty with adaptively generating keys and values for memory operations. Dynamic-NTMs (DNTMs) (**?**) alleviate this via fixed trainable keys, and are shown to aid QA tasks but they did not work either as we will show in our experimental section. Like LMNs, DNTMs also propose discrete memory addressing but the keys are trained from input $\mathbf{x}_t$ unlike in LMNs where the discrete key is class label that requires no training. Another difference with LMNs is that the memory is very tightly integrated with the neural network and requires joint training.

MANNs for classification More recent MANNs designed for classification employ a looser coupling (**?**; **?**) and store hidden vectors and discrete labels as memory contents as in LMNs. However, they differ in how the memory is addressed, updated, and used (shown graphically in Figure 1). First, LMNs treat memory as an on-line learner and update memory only when loss is non zero unlike all previous MANNs that update the memory for every step. This allows memory to be used more effectively to store instances when PCN is weak. Second, in MANNs memory reads are fed back into the main model for getting a prediction. Instead, in LMNs memory reads are loosely combined with the PCN scores allowing us to use memory for plug-and-play adaptation of a trained model. Finally, most MANNs use a global LRU replacement strategy whereas we replace only within in a label based on importance. This reduces the bias against rare classes present in LRU policies.

Model Adaptation by parameter re-learning Another way to adapt is by training or tuning parameters. The methods of (**?**) and (**?**) train only a subset of parameters that are local to each sequence. More recently,(**?**) and (**?**) propose meta-learners that "learn to learn" via the loss gradient. In general, however such model retraining techniques are resource-intensive. In our empirical evaluation we found these methods to be slower and less accurate than LMNs.

Online Learning Online learning techniques such as (**?**) for learning kernel coefficients is relevant if we view the memory vectors $\mathbf{h}_m$ acting as the support vectors and the memory scalars α_m as the associated dual variables. Our setup is a little different in that we employ a mix of batch and online learning. Our proposed scheme of memory updates and merge was inspired by the gradient updates in PEGASOS, and in the case of exactly one cell per label reduces to a specific form of budgeted-PEGASOS(**?**).

Experiments

We next present empirical comparison of LMNs with recent MANN based meta-learners and state of the art parameter re-learning based adaptive models. We experiment [2] on three different supervised learning tasks that require online model adaptation: sequence prediction on five datasets spanning applications like user trajectory prediction and next click

[2] code to be available on https://github.com/sshivs/LMN

Name	No. of train sequences	No of test sequences	Avg seq. length	# Tokens or labels
Brightkite	1800	521	238	22811
FSQNYC	670	264	90	8325
FSQTokyo	1555	672	160	12589
Geolife	220	20	8000	31273
Yoochoose	450523	112279	13	39481

Table 1: Statistics of data used for on-line sequence learning

prediction, image classification under online addition of new image labels, and language modeling.

Online sequence prediction

In this task the data consists of a sequence of discrete tokens $x_1, \ldots, x_n$ and the label y_t to be predicted at time t is just the next token x_{t+1}. The inputs are stateful and therefore the PCN has to be a RNN. We collected five such datasets from different real-life applications. In Table 1 we summarize the average length of each sequence, number of tokens, and the number of sequences in the training and test set.

- FSQNYC and FSQTokyo are Location Based Social Network data collected by (**?**) from FourSquare of user check-in at various venues over an year.
- Brightkite (**?**) is a user check-in dataset made available as part of Stanford Network Analysis Project (**?**).
- Geolife (**?**) is the trajectory data of people collected over multiple days, and provides the GPS coordinates of people tracked at almost a minute interval. We discretize the locations with a resolution of 100 meters and limit to the densest subset around the city.
- The Yoochoose dataset (**?**) is the click event sessions for a major European e-tailer collected over six months.

Experimental setups In all experiments we used the Adam optimizer (**?**). The PCN is a GRU and the input is the embedding of the true observed token y_{t-1} at the previous time. The number of memory cells is equal to the number of labels.

Methods compared We evaluate these tasks on six different methods.

- As a baseline we train a larger LSTM (**?**) which has roughly 5 times more RNN parameters compared to the PCN used in LMN. This lets us evaluate if a larger LSTM state could substitute for memory.
- Next we choose two recent approaches from the family of meta-learners that re-tune model parameters: the method of (**?**) since it was specifically proposed for sequence prediction and the more recent but generic method MAML of (**?**). Both these models used an LSTM of size 50 as the base learner. After each true label is encountered these models compute gradient of the loss with respect to the adapting parameters, and apply those updates, before processing the next input.
- As a representative of MANNs, we implemented a version of D-NTM (**?**) where we made two changes to adapt to the on-line prediction task. First, we use the previous read address as the new write address, and second we derive the new content from read memory content and y_{t-1}. We tried several other tweaks, including the unchanged D-NTM and obtained best results with these changes.
- We compare these with LMN. To illustrate the importance of adaptively reweighting the PCN and memory scores as per Equation 4, we also tried another model called LMN-fixed. In LMN-fixed θ_{ty} is fixed for all t and y and is determined by batched hyper-parameter optimization.

Results In Table 2 we report the log-perplexity and mean reciprocal rank (MRR) of all six methods on the five datasets. We make the following observations from these.

1. LMN-based online model adaptation significantly boosts accuracy over a baseline LSTM with five times larger state-space. The datasets vary significantly in their baseline and LMN accuracies. FSQNYC, FSQTokyo and Brightkite datasets have MRR increasing by 100 and 400%. For Yoochoose improvements are smallest because most sequences are very short (13 on average). In Geolife the sequences are much larger but the baseline MRR (0.84) is already high indicating a saturation point.
2. The improved accuracy of LMN over LMN-fixed illustrates the importance of online reweighting of PCN and memory. We observe improvements over baseline on almost all datasets, even on Yoochoose where none of the other methods did. However, LMN-fixed is better than parameter retuning and DNTMs establishing that even without training the combiner, LMNs can be used as plug-and-play model adapters.
3. Meta learners that retune parameters for adaptation are indeed effective compared to the baseline. The Rei method that retunes only a designated 'sequence-vector' parameter is less effective than the MAML that retunes all parameters. Yet, our proposed LMN, (or its simpler LMN-fixed variant) provides an even greater boost. For example, for Brightkite the MRR increases from 0.11 to 0.18 with Rei, to 0.47 with MAML, and to 0.51 with LMN-fixed and 0.55 with LMN. A major shortcoming of MAML is that training the meta-learner is very slow. We were not able to complete the training of MAML on our two largest datasets Geolife and Yoochoose within a reasonable time. In contrast, LMNs are significantly faster as they do minimal re-training. DNTM, another MANN-based approach is not as effective as even LMN-fixed and we believe it is mainly because of the differences in their respective memory organization.

These experiments establish that a well-designed MANN is a practical option for accurate and efficient online adaptation for next-token prediction tasks.

Online Adaptation of Image Classifiers

We next demonstrate online adaptation of an image classifier to new labels observed during testing. Unlike existing work in few-shot learning (**?**) where each test sequence has its own

	Baseline	Parameter-retune		Memory-based		
Name	LSTM	Rei	MAML	DNTM	LMN-fixed	LMN
Brightkite	10.7 (0.11)	10.01 (0.18)	4.27 (0.47)	9.63 (0.13)	3.88 (0.51)	**3.57 (0.55)**
FSQNYC	8.95 (0.03)	8.72 (0.07)	6.55 (0.16)	9.01 (0.05)	6.13 (0.25)	**5.54 (0.27)**
FSQTokyo	8.14 (0.08)	6.95 (0.13)	5.68 (0.23)	7.25 (0.11)	5.40 (0.26)	**5.32 (0.28)**
Geolife	1.13 **(0.84)**	1.08 (0.83)	-	1.11 (0.82)	**1.05** (0.83)	1.08 (0.83)
Yoochoose	5.01 (0.24)	5.05 (0.23)	-	5.01 (0.23)	5.01 (0.24)	**4.96 (0.25)**

Table 2: Log Perplexity and MRR(in parantheses) on online sequence prediction tasks

independent prediction space, we consider the more useful and challenging task where new labels co-exist with labels seen in the batch-trained PCN.

Dataset and setup We use the popular omniglot dataset (**?**). The base data-set consisted of 1623 hand-drawn characters selected from 50 different alphabets. (**?**) extended the base data-set by various rotations of the images, and this increases the number of categories to 4515. We create an online variant of this dataset as follows. We arbitrarily select 100 classes as unseen and 250 as seen classes. During training only seen classes are provided, but the test data has a uniform mix of all classes. Each test sequence is obtained by first randomly selecting 5 labels from the 350, and then choosing different input images from the selected labels up to a length of 10. Thus, in each sequence we expect to encounter $\frac{10}{7}$ new labels on average. Accuracy is measured only over second occurrence of each of the five labels much like in one-shot learning experiments. The one major difference is that in our case the prediction space is all the seen labels and the expected new labels, whereas in few-shot learning the prediction space is only the 5 labels chosen for that test sequence. The results are averaged over 100 sequences.

Methods compared We compare results of LMN with MAML (**?**) the most recent meta-learner that can work in this setup. We do not compare with the two recent MANNs (**?**; **?**) that report Omniglot results on few-shot learning because their techniques do not easily extend to the case where new labels share label space with a pre-trained classifier. For reference we also report accuracy on the baseline classifier that will certainly mis-classify examples from the new class. We use as PCN a convolutional network, with the same configuration as used in (**?**).

.

Model	Overall	New labels	Seen labels
Baseline	45%	0	63%
MAML	56%	49%	59%
LMN	**86%**	**71%**	**92%**

Table 3: Accuracy on Online adaptation for Omniglot

Results In Table 3 we report our results. The baseline that does no adaptation achieves an overall accuracy of 45% while obtaining an expected accuracy of 0% on the new labels and 63% on the seen labels. MAML boosts the overall accuracy to 56%, while obtaining an accuracy of 49% on the new labels. However, compared to the baseline the accuracy on seen labels has dropped. This is similar to catastrophic forgetting (**?**) and may be because during meta-learning updates the weights of the shared representation layer deteriorate. The LMN architecture is better able to absorb new labels and changing priors of existing labels. LMN achieves an overall accuracy of 86% with 71% on new labels. The accuracy increases for the seen labels as well because LMN is able to use the memory for storing prototypes of examples with non-zero loss even from seen labels. Thereafter, the combiner RNN can pay more heed to the labels seen during the adaptation phase.

Plain few-shot learning To enable comparison with the few-shot results of many recent published work, we also report results of LMN on this task. In few-shot learning, the PCN only generates the input embedding $\mathbf{h}_t$ and does not assign label scores since each test sequence has its own label space. The memory uses the embedding to assign label scores via Equation 3 based on which prediction is made without involving any combiner either. These experiments will compare LMN's memory organization and update strategies with state-of-the-art MANNs that have reported results on few shot learning (**?**).

Our setup is the same as in recent published work (**?**) on few-shot learning but where the total label space has 4K possibilities labels. This is more interesting and realistic than 5-way classification where many recent methods, including ours, report more than 98% accuracy with just 1-shot.

We present results with changing memory size. We run the experiments with a memory size equal to T-cell per label (T=2,3..). We observe that LMN accuracy is much higher than existing MANNs particularly when memory is limited. For example, at 2-cells per label (roughly 8K memory size) we obtain more than 4% higher accuracy in 1-shot, 2-shot, and 3-shot learning. This proves the superior use of memory achieved via our labeled memory organization

For reference we also compare with parameter retuning-based approach (MAML) (**?**). Even though this work reports state-of-art accuracy on 5-way few-shot learning, for 4k way few-shot learning it is not as effective. A softmax layer that has to arbitrate among 4k new classes perhaps need lot more gradient updates than learnable by meta-learners.

Name	1 shot	2 shot	3 shot
Kaiser (2-cell/label)	48.2%	58.0%	60.3%
Our model (2-cell/label)	**52.6%**	**62.5%**	**64.1%**
Kaiser (3-cell/label)	49.7%	60.1%	63.8%
Our model (3-cell/label)	**52.8%**	**63.0%**	**67.0%**
Kaiser (5-cell/label)	52.7%	63.0%	66.3%
Our model (5-cell/label)	**54.3%**	**63.2%**	**66.9%**
MAML	44.2%	46.5%	47.3%

Table 4: Test accuracies for 4k way few shot learning

Name	Wikitext2 (100)	Text8 (2000)
LSTM	99.7	120.9
Pointer LSTM	80.8	-
Neural cache	81.6	99.9
LMN	**77.6**	**91.1**

Table 5: Test perplexity for language modeling on Wikitext and Text8 with memory 100 and 2000 respectively.

Language Modeling

Language modeling is the task of learning the probability distribution of words over texts. When framed as modeling the probability distribution of words conditioning on previous text, this is just another online sequence prediction task. The natural dependence on history in this task provides for another use-case of memory. Memory based models have been shown to get improvements over standard RNN based language models (**?**; **?**; **?**). In the same spirit, we apply LMNs to this task by taking the PCN as a RNN.

We compare our model directly with the recently published neural cache model of (**?**) and pointer LSTM of (**?**). These showcase variant uses of memory to improve the prediction of words that repeat in long text. The baseline model (and PCN) is an LSTM with same parameters as in (**?**). We compared on common language datasets Wikitext2 and Text8 with memory sizes 100 and 2000 as used in the previously published work. We used standard SGD as the optimizer.

In Table 5 we report the perplexities we obtained with LMN along with the results reported in published work for other three approaches. As the table demonstrates we achieve state of the art results in Text8 and Wikitext2. In LMN the auto-modulation caused by considering only cases where the PCN is weak is superior to memory cache. This shows up in tests when memory is constrained, when the focus on mis-predicted outputs in LMN allows for boosted recall, efficient memory utilization, and capturing longer contexts, compared to other models.

Conclusion

We extended standard memory models with a label addressable memory module and an adaptive weighting mechanism for on-line model adaptation. LMNs mix limited data with pre-trained models by combining ideas from boosting and online kernel learning while tapping deep networks to learn representations and RNNs to model the evolving roles of memory and pre-trained models. LMNs have some similarities to recent MANNs but has significant differences. First, we have a label addressable memory instead of content based addressing. Second, we use memory to only store content on which primary network is weak. Third, our model has a loose coupling between memory and network, and hence our model can be used to augment pre-trained models at a very low cost. Fourth we use an adaptive reweighing mechanism to modulate the contribution of memory and PCN. This LMN is demonstrated to be extremely successful on a variety of challenging classification tasks which required fast adaptation to input and handling non-local dependencies. An interesting extension of LMNs is organizing the memory not just based on discrete labels but on learned multi-variate embeddings of labels thereby paving the way for greater sharing among labels.

Acknowledgements We gratefully acknowledge the support of NVIDIA Corporation with the donation of the Titan X Pascal GPU used for this research.

Aliquid quia autem provident optio aliquam, in a nesciunt quas dolore impedit id, voluptates culpa numquam perferendis quae asperiores cupiditate dolore nobis sunt reiciendis obcaecati, sequi optio ipsa.Deleniti eos facilis unde quam ducimus perspiciatis dicta esse, ab exercitationem autem error recusandae, necessitatibus distinctio aliquam aut illum placeat corrupti nesciunt?Aspernatur illo voluptate, quo est non beatae cupiditate sit ratione.Incidunt voluptate optio blanditiis nulla delectus, animi omnis officiis beatae autem illum distinctio aut mollitia molestiae, dicta velit qui cupiditate beatae itaque cumque ipsam accusamus ratione nam, laudantium in sed quasi pariatur porro dolor quidem earum.Alias nostrum fugiat enim aut tempore tempora ab, cupiditate aliquid sequi perspiciatis dolor beatae voluptatibus velit ipsa, eaque qui repudiandae maiores doloremque est ipsam dolorum officia laudantium, nisi ut saepe minus vitae recusandae ullam ea optio ducimus aliquid.A facere officiis quae unde enim nobis quos optio, veniam vel dolorem aperiam temporibus alias omnis deserunt.Et repellendus fugit repudiandae possimus alias eaque impedit facere amet sequi, minus neque deserunt eius repellat iusto soluta eos iste libero est quasi?Nostrum aut error excepturi corrupti dolores vel consectetur architecto incidunt, quia dolore iure assumenda nobis incidunt optio deserunt officia, consequatur voluptate nostrum deserunt ratione maiores iusto eaque pariatur, minus molestias autem eum animi, voluptatem facere eveniet voluptate quia est placeat blanditiis assumenda.Veritatis laudantium quidem saepe vitae aliquam corporis, facere impedit illo facilis, ipsum corrupti at dolore neque aliquid voluptates necessitatibus cum laudantium nemo.Itaque nihil veniam id cumque perferendis, deleniti odit ipsa?Perferendis ducimus tenetur laudantium deleniti quasi laborum, quas sequi autem rem deleniti expedita, ea inventore facilis recusandae nesciunt eius tenetur veniam sequi cumque, illo magni nam voluptatem qui est culpa?Atque modi veritatis molestiae tempora dolorum nostrum tenetur officia iusto dolore nam, vel eum