Abstract

We propose an alternative design of memory augmented neural networks (MANNs) called Labeled Memory Networks (LMNs) suited for tasks requiring fast adaptation of batch-trained classification models. LMNs have two key differences with existing MANNs. First, LMNs organize the memory with the discrete class label as the primary key unlike the existing practice of key being a real vector derived from the input. Second, LMNs treat memory as a second boosted stage following a regular neural network thereby allowing the memory and the primary network to play complementary roles. Unlike existing MANNs that write to memory for every instance and use LRU based memory replacement, LMNs write only for instances with non-zero loss and use label-based memory replacement. These properties make them particularly suited for classification tasks requiring fast adaptation and memorizing rare events. We demonstrate significant accuracy gains on three such tasks: online sequence prediction, life-long learning of rare events, and few-shot learning.

1 Introduction

While deep learning models achieve impressive accuracy in supervised learning tasks such as computer vision [14] and translation [29], 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 cannot operate in such batch settings, and require trained models to quickly adapt to new examples and settings during deployment. For example, in image recognition a neural network should be able to recognize new objects not seen during batch training based on just one or a few examples. This has led to a surge of interest in techniques like few-shot learning [12, 21, 27, 24].

Another task requiring fast adaptation is online sequence prediction that arises in applications such as text auto-completion, user trajectory prediction, online ad targeting, or next-url prediction. In these applications, even though the batch training data may contain many sequences, at usage time the model has to make online next-token predictions in new sequences. Each sequence is driven by a combination of its own local patterns and global patterns shared with other sequences in the training set. Unlike recent work on one-shot learning that focuses on handling new labels during test-time, here the challenge is to adapt to sequence-specific local patterns even within the same space of labels. This can be treated as a domain adaptation task, for which many techniques have been proposed [20, 5, 11]: the challenge in these applications is that the adaptation has to be very lightweight and online.

Recently, a new architecture in the form of Memory augmented neural networks (MANNs) has emerged as a partial solution to the above challenges. Memory networks were originally developed with goals far loftier than plain supervised learning. For example, in [7], an end-to-end differentiable memory network is used to learn basic tasks and in [28] memory is used to memorize facts relevant
for question answering. But they also have been found useful for tasks like few-shot learning and learning rare events \cite{12, 21, 27, 24}. In these networks, a memory unit is used to quickly memorize new facts, thereby cutting short the conventional path of depending on an iterative training procedure to percolate new facts to model parameters. However, when applied to online sequence prediction, it is unclear how best to combine the roles of the memory and the batch-trained neural network. On the one hand, we have the advantage of a carefully trained global model over millions of sequences and on the other hand we have a single sequence’s hundreds of previous tokens. Furthermore, as memory gets large, the time spent in content-based addressing gets formidable. One solution to this problem could be to use a nearest neighbor index \cite{9, 19}, but state-of-the-art nearest neighbor indices find only approximate matches and cannot handle changing vectors faced during training.

In this paper we propose a new architecture for memory augmented neural network that improves on existing networks in two ways:

First, 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 proposals that, following the tradition of the neural turing machine, use a controller to generate a hidden vector of the input as key. We show that for classification tasks, labeled memory allows for significantly better memory use. By using class labels as the primary means of addressing memory, the controller is freed from the vaguely supervised task of generating keys that are distinctive and relevant.

Second, inspired from Boosting \cite{22} we propose to use the memory to preferentially store content that are misclassified by the regular network. Existing MANNs write to memory during every pass. Even when the goal of the model is to use the memory to remember rare events \cite{12}, the memory stores all events not just the rare ones. This, coupled with a LRU replacement policy, causes a lot of memory to be wasted in storing non-rare vectors often displacing the rare ones. We propose to use memory selectively only in regions where the global model is deficient.

The model we propose is generic and can handle many different settings of classifier learning and deployment. In this paper we present three settings. First, in online sequence prediction we need to combine global patterns across all sequences with per-sequence patterns. On three out of five real-life datasets we report significant gains in perplexity and mean reciprocal rank. Second, in few-shot learning a neural network needs to adapt to new classes under very small number of examples per class. We report higher n-shot accuracy for both 5-way and 4000-way classification while using less memory. Third, in rare event learning memory is used to store rare events. On a challenging synthetic prediction task, we achieve 10% more sequences correct than existing MANNs.

2 Background and Related work

Attention in RNN The earliest example of the use of memory in neural networks is attention \cite{1} over past RNN states. In principle the state of a RNN has capacity to memorize hidden patterns from data relevant to a classification task. However, the success of attention in RNNs has shown that in practice it is useful to associatively revisit the history of previous RNN states. Most common use is in encoder-decoder models like in translation \cite{26, 15} where the attention is over the input sequences. Our focus in this paper is online sequence prediction, where a more relevant form of attention is self-attention. For language models, \cite{4} shows that self-attention is not more effective than concatenation of the past five states. Their task is different from our task of online adaptation for next-token prediction. For us memory of all previous tokens is important, as it is not for summarizing the current context of the user but for recalling the following token from a similar context anywhere in the past.

Memory Augmented Neural Networks (MANN) Memory implemented as attention is slow for long sequences, and also attention cannot memorize rare facts across sequences. An explicit external memory is more general-purpose and recently several memory augmented models have been proposed. Neural Turing Machines (NTMs) \cite{7} were developed to learn to perform tasks involving algorithmic copy and recall. For example, NTMs have been demonstrated to learn a N-gram distribution from token sequences. Since this task is related to online sequence prediction, our first model was based on NTMs. However, on real-life datasets NTMs failed to yield accuracy improvement beyond plain LSTMs. We suspect this is because the controller could not learn to adaptively generate keys and values for memory operations. Dynamic-NTMs \cite{8} alleviate this by assigning fixed keys
to memory cells that are learned during training, and are shown to aid QA tasks. However, in our experiments for online sequence prediction, we found that the trained keys were too similar to each other leading to redundancy and loss of effectiveness. Another major difference with our proposed labeled memory network is that the memory is tightly integrated with the RNN and at each step the memory reads feed into the next state of the RNN.

**MANNs for classification** More recent MANNs designed for few-shot learning and rare class classification employ a loose coupling like ours. We review the method of [12]. Each memory cell stores a key-value-time triple where the key is a real vector, value is a class index, and time denotes when the cell was last correctly used. The hidden vector \( h_t \) from the last layer of the NN is used to retrieve the nearest cell based on the keys. The output from the memory is then embedded in a real space, concatenated with the last layer from NN, transformed via a linear layer, and fed to a softmax layer to get the class prediction. Memory is updated for every instance. When the top-1 value matches the correct class \( y_t \), the top-1 key is merged with \( h_t \), and access time updated, else a least-recently used cell is replaced with the new key \( h_t \) and value \( y_t \). In addition to the usual classification loss, the authors introduced a separate memory loss to promote the top-1 memory match to be correct.

**Comparison with existing MANNs** Our method of organizing memory differs significantly from the above MANNs. First, these methods follow the trend set by generic memory networks where content-based addressing is purely based on the hidden vector \( h_t \) derived from the input. In contrast, our primary key is label and this is better tailored for classification task for various reasons. First, the labeled memory allows us to design a memory update strategy that removes the bias against rare classes that LRU policies are subject to. Second, the labeled memory allows us to assign memory scores from multiple class labels. We can then treat memory as a second stage of a boosted classifier and update memory only when loss is non zero. This allows us to better partition the roles of the primary neural network and memory. In contrast, all previous MANNs update the memory for every step. Finally, we do not need a separate memory loss over only the memory scores because we combine the label-wise memory scores with the primary neural network scores and create a global training objective.

**Model Adaptation and online learning** For online sequence prediction, another way to adapt to local sequences is to retrain/retune parameters of a globally trained network casting it as a domain adaptation task [20, 5, 11]. For example, [20] proposes to online train an embedding vector per sequence during testing. The embedding vector is concatenated with the LSTM output before the softmax layer. This approach has been shown to be successful for a language modeling task. In general, however model retraining is time-consuming and not robust for online tasks. Online learning techniques such as [23] for learning kernel coefficients under limited budget is relevant but our setup is different in that we employ a mix of batch and online learning. Our proposed scheme of memory updates was inspired by this online learning literature.

### 3 Labeled Memory Networks (LMNs)

Our labeled memory module can be used to augment any neural classification model be it a CNN for image labeling, a RNN for sequence prediction, or a regular feed forward network. We attach the memory just before the last classification layer of the network. Assume the last layer of the neural network produces a real vector \( h_t \in \mathbb{R}^d \) as a function of the input (and/or state for RNNs) and trainable parameters \( \theta \). This vector is converted to class scores using a softmax layer. We use \( \beta_y \in \mathbb{R}^d \) to denote the softmax parameter for class \( y \), so the score \( r_y \) for class \( y \) is

\[
r_{ty} = g(\beta_y h_t)
\]

where \( g \) is an activation function like \( \tanh \) or \( \exp \). We will use the term primary classification network (PCN) to refer to this component of our network.

The memory consists of \( N \) cells. Each cell \( m \) is associated with a 3-tuple: \( \ell_m \) denoting the label of cell \( m \), \( v_m \) denoting the hidden vector stored in \( m \), \( \alpha_m \) denoting a weight attached to the cell.
Labeled memory reads A read on the memory requires a label \( y \) and vector \( h_t \) and returns a vector \( M_t y \) and a weight scalar \( \alpha_t y \) calculated as

\[
M_t y, \alpha_t y = \sum_{m: \ell_m = y} w_{tm} \{ v_m, \alpha_m \}, \quad w_{tm} = \text{softmax}(K(h_t, v_m) : \ell_m = y) \quad (2)
\]

where \( K(h_t, v_m) \) denotes a similarity function. We use Cosine where \( K(h_t, v_m) = \frac{h_t \cdot v_m}{\|h_t\| \|v_m\|} \). Unlike existing MANNs that take a softmax over all memory cells, in labeled memory softmax is only over cells with label \( y \).

Combining memory and PCN The guiding principle behind our design of MANN was to write to memory only when the primary classification pipeline is deficient. This is useful for learning rare classes and when combining global and local patterns in online sequence learning. Existing MANNs that write to memory for every instance are prone to fill up the memory on cases that can be accurately handled by the primary neural network layers.

Inspired by algorithms such as Adaboost [22], we view the PCN as the first stage of a two-stage boosted classifier with memory as the second stage. The memory focuses on cases that the PCN cannot accurately classify. Like in boosting the score of a class \( y \) is obtained by taking a weighted sum of the score from the PCN and the memory. The score from the PCN is the output \( r_{ty} \) from the softmax layer (Equation 1). The score from the memory for a class \( y \) is derived from the values \( M_t y, \alpha_t y \) read from memory as:

\[
s_{ty} = f(K(h_t, M_t), \alpha_t y) \quad (3)
\]

where \( f \) is another pluggable function that is chosen in tandem with \( g \), the activation function from the PCN. For example, one choice of \( f \) is \( \alpha_t y \exp(K(h_t M_t)) \) and \( g \) is the exp function. We will discuss other choice of activation functions in our experiments.

The combined score for a label \( y \) is thus \( r_{ty} + \lambda s_{ty} \) where \( \lambda \) is another parameter. The overall training objective of this combined network is then:

\[
\text{loss}(y_t, \{ r_t(\theta, \beta) + \lambda s_t(\theta, M_t) \}) \quad (4)
\]

In the above, \( y_t \) is the true label output at time \( t \) and \( M_t \) denotes the contents of memory at time \( t \). We explicitly write \( r_t \) and \( s_t \) as functions of parameters that they are a part of. \( r_t \) depends both on \( \theta \) and \( \beta \) while the memory score \( s_t \) depends only on PCN parameters \( \theta \) and memory contents \( M_t \) at time \( t \). The loss can be any any pluggable loss function like cross-entropy and Hinge. The gradient for the \( \theta \) parameters of PCN is computed through \( r_t \) and \( s_t \), the gradient for the softmax parameters \( \beta \) of PCN flows through \( r_t \). The updates to \( M_t \) are also aligned with the gradient of above training objective as described below.

Labeled memory writes We treat updates to the memory as applying one step of gradient update on the training objective with respect to memory contents \( M_t \). However, in order to make sure that updates to memory are sparse, we design our updates based on a multi-class hinge-loss rather than cross-entropy. Thus when the hinge loss is zero, the gradient with respect to the memory is zero and we make no update to the memory.

Consider the content \( v_m \) and weight \( \alpha_m \) associated with each memory cell \( m \) at time \( t \). We update the content of cells with label \( y_t \) using \( v_m = v_m + w_{tm} h_t \) where \( w_{tm} \) is the fractional similarity of \( h_t \) to contents of cell \( m \) (Equation 2). The weight \( \alpha_m \) of the cell is incremented by the fractional similarity \( w_{tm} \) to \( h_t \) after decaying old weights. Furthermore, if the predicted label is incorrect we attempt to create a new cell in memory replacing the cell with the smallest weight among all cells with the same label \[1\]. We summarize our memory write strategy in Figure 1.

We next present how our proposed labeled memory can be used under four different tasks.

3.1 Fast adaptation to per-sequence patterns in online sequence prediction

In online sequence learning the task is to predict the next token in a sequence given all previous tokens. The training dataset \( D \) consists of \( N \) sequences \( y^1, \ldots, y^N \). Each sequence \( y \) is an ordered

---

1We assume that available memory is greater than the number of labels and each label is allotted at least one cell in memory at the start. But it is easy to modify the replacement strategy to handle the case when this assumption is not true.
Input: $y_t, \hat{y}_t, \ell_t, h_t, M_t = \{(\ell_m, h_m, \alpha_m)\}$
If $\ell_t = 0$, no update. Return.
New cell $C: (\ell_{y_t}, h_t, 1)$
\[
\begin{align*}
  j &= \arg\min_m \ell_m = y_t \alpha_m \\
  w_t &= \text{softmax} \{\text{Cosine}(h_t, v_m) : \ell_m = y_t\} \\
  v_m &= v_m + w_{tm} h_t \\
  \alpha_m &= \alpha_m + \text{decay} + w_{tm} \\
  \text{if } y_t \neq \hat{y}_t \text{ and } |m : \ell_m = y_t| > 1 \text{ then} \\
  &\quad \text{Replace cell } j \text{ with new cell } C.
\end{align*}
\]

Figure 1: Memory updates in Labeled memory network

set of tokens $y_1, \ldots, y_n$ where each $y_j \in \{1, \ldots, m\}$. During testing for a new sequence we need to repeatedly along time predict the next token $\hat{y}_t$ as a function of all previous tokens $y_1, \ldots, y_{t-1}$. After this, the true token $y_t$ is available which can be used to update the model before making a prediction $\hat{y}_{t+1}$ for the next time $t + 1$. In applications like user-trajectory prediction, we expect the global pattern over all sequences (users) and local pattern specific to a sequence (user) to be useful in predicting the next token.

Our labeled memory network is particularly suited for online sequence prediction since we carefully orchestrate the roles of PCN and memory for effective global/local combination. The RNN captures global pattern as the PCN. The input at each time $t$ of the RNN is an embedding of the true observed token $y_{t-1}$ at the previous time. Unlike earlier memory networks such as NTM [7] and D-NTM [8], the RNN does not take as input the memory reads from the previous time step. Like in boosting we train the RNN parameters without the memory-stage on the training sequences $D$ using a cross-entropy loss. Each sequence thereafter uses its own part of the memory to adapt to sequence-specific patterns during testing. The mixing weight $\lambda$ can be either trained in a second training stage, fixed as a hyper-parameter or evolving based on the online accuracy from the two stages much like in Boosting. In Section 4.1 we present an experimental evaluation of different models on the datasets.

3.2 Few-shot learning

Few-shot learning succinctly is learning from few examples, and has been popularized recently in [12, 21, 27, 24]. A typical setting during training and testing in these works is to divide the entire data into episodes. Each episode consists of $N$ labels and a fixed small number of examples for each. Within an episode the order of presenting the examples and revealing the labels is much like in our online sequence learning task except that in this case there is no need for a RNN-based context. The instances and their labels are drawn from a shared dataset but the identity of the labels differs from episode to episode. In particular, the test episode could include labels not seen during training. Accordingly, in all recent work the shared part of the model is a network that converts each input into an embedding space, and use the augmented memory to store one or more of these embeddings in memory, and classify inputs based on similarity with the current embedding. The memory stores the embedding vectors and is separate for each episode and not persistent across training. Our top-level architecture for few shot learning is thus similar to existing MANNs but the difference in the way we use memory leads to improved accuracy as we will see in Section 4.2.

3.3 Life-long learning of rare events

Another interesting use of memory in neural networks is for remembering rare events that may not be easily captured as parameters of the neural network. For example, learning a classifier with a mix of frequent and rare classes. When LMNs are applied to this task, the memory is persistent across training and shared over all data instances. The PCN can be a RNN or a feed forward network with a last softmax layer. If the number of rare classes is very large (like in our experiments in Section 4.3), the PCN might choose to not allocate any softmax parameters for the rare classes. Unlike existing MANNs that update the memory for every instance, in LMN the memory will be updated only for the misclassified examples. Such instances could be from frequent classes, but are more likely to be from rare classes. It is important to choose a memory replacement strategy that is not biased against rare classes. Strategies like LRU common in existing use of MANNs do not have that property since
the rare label is more likely to be least recently used. Our replacement strategy replaces a cell within the same label, making it less likely for frequent classes to displace rare once.

### 3.4 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, its same as sequence prediction. 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 [25, 18, 6]. In the same spirit, we apply LMNs to this task by taking the PCN as a RNN.

### 4 Experiments

In this section, we assess our method on the different supervised learning tasks outlined earlier. In all experiments we used the Adam optimizer [13] except for language modeling where we use standard SGD. We describe the datasets and experiment specific settings below. We plan to release our code in public domain for reproducibility of our results.

#### 4.1 Online sequence prediction

We collected five datasets from different real domains where online sequence prediction is natural. In Table 1 we summarize the average length of each sequence, number of labels ($m$), and the number of sequences in the training and test set.

| Name        | # Train | # Test | Avg Sequence | # Tokens |
|-------------|---------|--------|--------------|----------|
| 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 online sequence learning

Methods compared For training our LMN-based online sequence prediction model described in Section 3.1 we use a GRU with a hidden unit of size 40 as the PCN. We use $\exp$ as the PCN activation function $g$ (Equation 1) which corresponds to the softmax classifier. As the memory scoring function $f$ (Equation 3) we use $\alpha_y * \exp(\gamma * \text{Cosine}(h_t, M_y))$. The mixing parameter $\lambda$ and modulation parameter $\gamma$ are hyper-parameters. The number of memory cells was equal to the number of labels. Input tokens in all methods are embedded into a 25-dimensional real space. We compare this LMN with three other methods.

1. First, as a baseline we train a LSTM [10] of size 100 (which has roughly 5 times more RNN parameters compared to the controller).
2. Second, we train the online representation learning model of Rei [20] with context of size 40.
3. Third, we implemented a version of D-NTM [8] where we made two changes to adapt to the online prediction task. First, we use the previous read address as the new write address, and second we derive the new content from read memory $\phi_t$ and $y_{t-1}$. We tried several other tweaks, including the original unchanged D-NTM method and obtained best results with these changes.

| Name       | LMN   | Rei     | LSTM    | DNTM   |
|------------|-------|---------|---------|--------|
| Brightkite | 3.88 (0.51) | 10.01 (0.18) | 10.7 (0.11) | 9.63 (0.13) |
| FSQNYC     | 6.13 (0.25) | 8.72 (0.07) | 8.95 (0.03) | 9.01 (0.05) |
| FSQTokyo   | 5.40 (0.26) | 6.95 (0.13) | 8.14 (0.08) | 7.25 (0.11) |
| Geolife    | 1.05 (0.83) | 1.08 (0.83) | 1.13 (0.84) | 1.11 (0.82) |
| Yoochoose  | 5.01 (0.24) | 5.05 (0.23) | 5.01 (0.24) | 5.01 (0.23) |

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

Results In Table 2 we report the average log-likelihood and mean reciprocal rank of all the models on the five datasets. The datasets vary significantly in their baseline accuracy and response to local adaptation. The FSQNYC, FSQTokyo and Brightkite datasets benefit significantly from the LMN. The model retraining method of [20] shows modest gains over the baseline on these datasets, but LMNs provides a much larger boost. For example, for Brightkite the MRR increases from 0.11 to 0.18 with [20]’s method but goes to 0.51 with LMNs. The Yoochoose dataset does not benefit from local adaptation 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 perhaps a saturation point.

4.2 Few-shot learning

We use the omniglot data [16] for our experiments. Our setup is the same as in recent published work [27] on few-shot learning. The base data-set consisted of 1623 hand-drawn characters selected from 50 different alphabets. [27] extended the base data-set by various rotations of the images, and this increases the number of categories to 4515. As we compare our model directly with prior work on few-shot learning that do not share a softmax layer, while using LMNs for this task we too get rid of the softmax layer from the PCN, that is, the $g$ function in [1]is zero, and $f$ is based on the cosine distance measure $\text{cosine}(h_t, M_y)$. We compare our method with the results obtained in a state-of-the-art MANN based few-shot learner [12]. In Table 3 we report the accuracies we obtained for the different shot tasks where each episode had a 5-way classification task. As is clear we match or improve state of the art results.

We also ascertain the impact of changing memory size. We used the setup of the previous experiment, but each episode has 4K labels. We run the experiments with a memory size equal to T-cell per label (T=2,3,..). The rest of the setup is identical to the previous Omniglot experiment. We observe that LMNs score 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.

---

As the exact test data is not available on omniglot, we have run our tests on a fixed set, whose results might differ from one presented in earlier works.

| Name | 1 shot | 2 shot | 3 shot | 4 shot | 5 shot |
|------|--------|--------|--------|--------|--------|
| Kaiser | 97.99% | 98.54% | 99.21% | 99.23% | 99.21% |
| LMN   | 98.24% | 98.75% | 99.14% | 99.27% | 99.34% |

Table 3: Test accuracies for 5 way few shot learning
| 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%  |

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

4.3 Learning rare-events

Here we demonstrate how LMN can be used to improve accuracy of rare events that are global to all episodes as described in Section 3.3. In these experiments memory is persistent across training and shared by all sequences.

We created a synthetic sequence learning task to test the ability of external memory to remember rare events. The input sequence consisted of seven digits from \{0,1,2,3\} with arbitrary As and Bs before and after it. The output copies the A and B exactly but transforms the seven digits via a fixed global random function $F$ as follows: Treat the seven digits $d_1, \ldots, d_7$ as a ternary number $n$, $F(n)$ maps $n$ to another unique number in $[0, 4^7)$. The output at each of the digit positions is blanks followed by $F(n)$. Thus, there are 16K input to output mappings. A standard neural network with softmax at the outermost layer will find it difficult to capture this large mapping via network parameters. In order to speed up training in this setting, we pre-train the PCN first with frequent classes. We use identity for $g$ and cosine similarity as $f$.

We measure accuracy as the fraction of completely correct output sequences, and the results are presented in Table 5. The model of [12] achieves an accuracy of 35% whereas with LMN we get an accuracy of 45%. In LMN the gradient computed using cross-entropy loss over combined scores of all labels is possibly more effective than the memory loss. Furthermore, our labeled organized memory avoids interference between labels during updates that other systems that depend on global LRU policies.

| Model                  | Accuracy |
|------------------------|----------|
| Baseline LSTM          | 10%      |
| Memory Model[12]       | 35%      |
| LMN                    | 45%      |

Table 5: Comparing LMN with other MANNs

4.4 Language Modeling

We also use our LMN for neural language modeling. We compare our model directly with the recently published neural cache model of [6] and pointer LSTM of [18]. In table 6 we report the perplexities we obtained on common language datasets PTB, Wikitext2 and Text8. As the table demonstrates we achieve state of the art results in Text8 and Wikitext2 (with small memory) and are competitive in other cases. In LMN the auto-modulation caused by considering only cases where the PCN is weak is slightly superior to memory cache. This shows up in tests when memory is constrained, when the focus on mispredicted outputs in LMN allows for boosted recall, efficient memory utilization, and capturing longer contexts, compared to other models.

5 Discussion

We extended standard memory models with a labeled address memory module. Combining it with ideas from online kernel learning, we use sparse memory updates effectively as subgradient of an implicit loss. This LMN has some similarities to new MANN’s 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. Thirdly our model has a very loose coupling
between memory and network, and hence our model can be used to augment pre-trained models at a very low cost. 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.

Our work also highlights new research problems, and suggests steps towards addressing them. One such problem is optimal memory addressing mechanism for memory models. Other works [8] have also demonstrated the difficulty of efficient storage and retrieval when compared to models which directly store incoming information. Secondly, it hints at a theoretical framework for analyzing memory update mechanisms, which could lead to explicit learning guarantees. Another possible application is in building large scale memory models. Training large memories is time-intensive, and hence adapting our sparse gradient aligned memory updates can allow building of large-scale memory models, without a costly memory training procedure.

### Table 6: Test perplexities for language modeling

| Name            | PTB (500) | wikitext2 (100k) | wikitext2 (200k) | text8 (200k) |
|-----------------|-----------|------------------|------------------|--------------|
| Pointer LSTM [18] | 70.9      | 80.8%            | -                | -            |
| LMN             | 72.0      | 78.8             | 69.3             | 90.4         |
| Neural cache [6]  | 72.1      | 81.6             | 68.9             | 99.9         |

References

[1] Dzmitry Bahdanau, Cho Kyunghyun, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. CoRR, abs/1409.0473, 2014.

[2] David Ben-Shimon, Alexander Tsikinovsky, Michael Friedmann, Bracha Shapira, Lior Rokach, and Johannes Hoerle. Recsys challenge 2015 and the yoochoose dataset. RecSys '15, 2015.

[3] Eunjoon Cho, Seth A. Myers, and Jure Leskovec. Friendship and mobility: User movement in location-based social networks. KDD ’11, 2011.

[4] Michal Daniluk, Tim Rocktäschel, Johannes Welbl, and Sebastian Riedel. Frustratingly short attention spans in neural language modeling. ICLR, CORR abs/1702.04521, 2017.

[5] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. CoRR, abs/1703.03400, 2017.

[6] Edouard Grave, Armand Joulin, and Nicolas Usunier. Improving neural language models with a continuous cache. arXiv:1612.04426, 2016.

[7] Alex Graves, Greg Wayne, and Ivo Danihelka. Neural turing machines, 2014.

[8] Çağlar Gülçehre, Sarath Chandar, Kyunghyun Cho, and Yoshua Bengio. Dynamic neural turing machine with soft and hard addressing schemes. CoRR, abs/1607.00036, 2016.

[9] R. Guo, S. Kumar, K. Choromanski, and D. Simcha. Quantization based fast inner product search. In AISTATS, 2016.

[10] S Hochreiter and J Schmidhuber. Long short-term memory. Neural Comput., 1997.

[11] Zhen Huang, Sabato Marco Siniscalchi, I-Fan Chen, Jiadong Wu, and Chin-Hui Lee. Maximum a posteriori adaptation of network parameters in deep models. In INTERSPEECH, 2015.

[12] L. Kaiser, O. Nachum, A. Roy, and S. Bengio. Learning to remember rare events. ArXiv eprints, 2017.

[13] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. CoRR, abs/1412.6980, 2014.

[14] Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton. Imagenet classification with deep convolutional neural networks. Advances in Neural Information Processing Systems, 2012.
[15] Cho Kyunghyun, Bart Merrienboer, Caglar Gulcehre, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN encoder-decoder for statistical machine translation. *CoRR*, abs/1406.1078, 2014.

[16] Brenden M. Lake, Ruslan Salakhutdinov, and Joshua B. Tenenbaum. Human-level concept learning through probabilistic program induction. *Science*, 2015.

[17] Jure Leskovec and Andrej Krevl. SNAP Datasets: Stanford large network dataset collection, June 2014.

[18] Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer sentinel mixture models. *arXiv:1609.07843*, 2016.

[19] Jack W. Rae, Jonathan J. Hunt, Tim Harley, Ivo Danihelka, Andrew W. Senior, Greg Wayne, Alex Graves, and Timothy P. Lillicrap. Scaling memory-augmented neural networks with sparse reads and writes. *CoRR*, abs/1610.09027, 2016.

[20] Marek Rei. Online representation learning in recurrent neural language models. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, 2015.

[21] Adam Santoro, Sergey Bartunov, Matthew Botvinick, Daan Wierstra, and Timothy P. Lillicrap. Meta-learning with memory-augmented neural networks. In *ICML*, pages 1842–1850, 2016.

[22] Robert Schapire. A brief introduction to boosting. *IJCAI’99*, 1999.

[23] Shai Shalev-Shwartz, Yoram Singer, and Nathan Srebro. Pegasos: Primal estimated sub-gradient solver for svm. *ICML ‘07*, 2007.

[24] J. Snell, K. Swersky, and R. Zemel. Prototypical networks for few-shot learning. *ArXiv eprints*, 2017.

[25] Sainbayar Sukhbaatar, Arthur Szlam, Jason Weston, and Rob Fergus. Weakly supervised memory networks. *CoRR*, abs/1503.08895, 2015.

[26] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. Sequence to sequence learning with neural networks. *CoRR*, 2014.

[27] Oriol Vinyals, Charles Blundell, Tim Lillicrap, Koray Kavukcuoglu, and Daan Wierstra. Matching networks for one shot learning. In *NIPS*, 2016.

[28] Jason Weston. Memory networks for language understanding, 2016.

[29] Yonghui et al. Wu. Google’s neural machine translation system: Bridging the gap between human and machine translation. *CoRR*, abs/1609.08144, 2016.

[30] Dingqi Yang, Daqing Zhang, Vincent. W. Zheng, and Zhiyong Yu. Modeling user activity preference by leveraging user spatial temporal characteristics in lbsns. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 2015.

[31] Y Zheng, L Zhang, Xing Xie, and Wei Ying. Mining interesting locations and travel sequences from gps trajectories. *International conference on World Wild Web (WWW)*, 2009.