Contextual Memory Trees

Wen Sun [1] **Alina Beygelzimer** [2] **Hal Daumé III** [3] **John Langford** [3] **Paul Mineiro** [4]

Abstract

We design and study a Contextual Memory Tree (CMT), a learning *memory controller* that inserts new memories into an *experience store* of unbounded size. It is designed to efficiently query for *memories* from that store, supporting logarithmic time insertion and retrieval operations. Hence CMT can be integrated into existing statistical learning algorithms as an augmented memory unit without substantially increasing training and inference computation. Furthermore CMT operates as a reduction to classification, allowing it to benefit from advances in representation or architecture. We demonstrate the efficacy of CMT by augmenting existing multi-class and multi-label classification algorithms with CMT and observe statistical improvement. We also test CMT learning on several image-captioning tasks to demonstrate that it performs computationally better than a simple nearest neighbors memory system while benefitting from reward learning.

1. Introduction

When a human makes a decision or answers a question, they are able to do so while very quickly drawing upon a lifetime of remembered experiences. This ability to retrieve relevant experiences efficiently from a memory store is currently lacking in most machine learning systems (§1.1). We consider the problem of learning an efficient online data structure for use as an external memory in a reward-driven environment. The key functionality of the Contextual Memory Tree (CMT) data structure defined here is the ability to *insert* new memories into a learned key-value store, and to be able to *query* those memories in the future. The storage and query functionality in CMT is driven by an optional, user-specified, external reward signal; it organizes memories so as to maximize the downstream reward of queries. In order to scale to very large memories, our approach organizes memories in a tree structure, guaranteeing logarithmic time (in the number of memories) operations throughout (§3). Because CMT operates as a reduction to classification, it does not prescribe a representation for the keys and can leverage future advances in classification techniques.

More formally, we define the data structure CMT (§ 2), which converts the problem of mapping *queries* (keys) to *memories* (key-value pairs) into a collection of classification problems. Experimentally (§4), we show this is useful in three different settings. **(a)** Few-shot learning in extreme multiclass classification problems, where CMT is used directly as a classifier (the queries are examples and the values are class labels). Figure 1 shows that *unsupervised* CMT can statistically outperform other *supervised* logarithmic-time baselines including LOMTree (Choromanska & Langford, 2015) and Recall Tree (RT) (Daumé et al., 2017) with supervision providing further improvement. **(b)** Extreme multi-label classification problems where CMT is used to augment a One-Against-All (OAA) style inference algorithm. **(c)** Retrieval of images based on captions, where CMT is used similarly to a nearest-neighbor retrieval system (the queries are captions and the values are the corresponding images). External memories that persist across examples are also potentially useful as inputs to downstream applications; for instance, in natural language dialog tasks (Bartl & Spanakis, 2017) and in machine translation (Gu et al., 2018), it can be useful to retrieve similar past contexts (dialogs or documents) and augment the input to the downstream system with these retrieved examples. Memory-based systems can also be useful as a component of learned reasoning systems (Weston et al., 2014; Graves et al., 2016).

A memory $z = (x, \omega)$ is a pair of query x and value ω. CMT operates in the following generic online manner, repeated over time:

1. Given a query x, retrieve k associated memories $(u, \langle z_1, z_2, \ldots, z_k \rangle) = \text{QUERY}(x)$ together with an identifier u.
2. If a reward r_i for z_i is observed, update the system via $\text{UPDATE}((x, z_i, r_i), u)$.
3. If a value ω associated with x is available, INSERT a new memory $z = (x, \omega)$ into the system.

[1] Robotics Institute, Carnegie Mellon University, USA [2] Yahoo! Research, New York, NY, USA [3] Microsoft Research, New York, NY, USA [4] Microsoft, USA. Correspondence to: Wen Sun <wensun@cs.cmu.edu>.

Proceedings of the 36th International Conference on Machine Learning, Long Beach, California, PMLR 97, 2019. Copyright 2019 by the author(s).

	Low Time	Small Space	Self-consistent	Incremental	Learning
Inverted Index		✓	✓		
Supervised Learning	✓	✓		✓	✓
Nearest Neighbor		✓	✓	✓	
Approx-NN	✓	✓	✓		
Learned-NN		✓	✓		✓
Hashing		✓	✓		✓
Differentiable Memory		✓	✓	✓	✓
CMT	✓	✓	✓	✓	✓

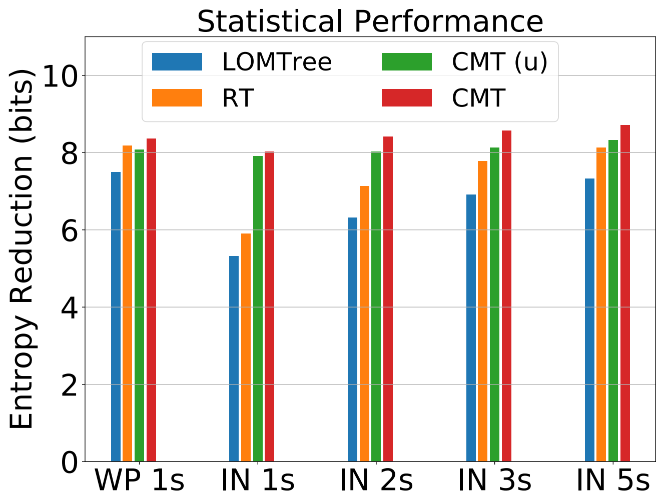


Figure 1: (Left) Desiderata satisfied by prior approaches; where answers vary with choices, we default towards 'yes'. (Right) Statistical performance (Entropy Reduction from the constant predictor. Higher is better—see the experiments section.) on WikiPara one-shot (WP 1s) dataset and ImageNet S-shot datasets (IN Ss) with baselines (LOMTree and RecallTree) and our proposed approach (unsupervised and supervised) CMT.

A natural goal in such a system is a notion of self-consistency. If the system inserts $z = (x, \omega)$ into CMT, then in subsequent rounds, one should expect that (x, ω) is retrieved when $\text{QUERY}(x)$ is issued again for the same x. (For simplicity, we assume that all x are unique.) In order to achieve such self-consistency in a data structure that changes over time, we augment CMT with a "Reroute" operation, in which the data structure gradually reorganizes itself by removing old memories and re-inserting them on an amortized basis. We find that this Reroute operation is essential to good empirical performance (§4.5).

1.1. Existing Approaches

The most standard associative memory system is a map data structure (e.g., hashmap, binary tree, relational database); unfortunately, these do not *generalize* across inputs—either an input is found exactly or it is not. We are interested in memories that can generalize beyond exact lookups, and can *learn* to do so based on past successes and failures in an *incremental*, online manner. Because we wish to scale, the *computation time* for all operations must be at most logarithmic in the number of memories, with *constant space overhead* per key-value pair. Finally, as mentioned above, such a system should be *self-consistent*.

There are many existing approaches beyond hashmaps, all of which miss one of our desiderata (Figure 1). A basic approach for text documents is an **inverted index** (Knuth, 1997; Broder et al., 2003), which indexes a document by the words that appear therein. On the other end of the spectrum, **supervised learning** can be viewed as remembering (compiling) a large amount of experience into a predictor which may offer very fast evaluation, but generally cannot explicitly query for past memories (aka examples).

There has been substantial recent interest in coupling neural networks with nearest neighbor variants. Classical approaches are inadequate: **a) Exact nearest neighbor** algorithms (including memory systems that use them (Kaiser et al., 2017)) are computationally inefficient except in special cases (Dasgupta & Sinha, 2015; Beygelzimer et al., 2006) and do not learn. **b) Approximate Nearest Neighbors** via Locality-Sensitive Hashing (Datar et al., 2004) and MIPS (Shrivastava & Li, 2015) address the problem of computational time, but not learning. **c) Nearest Neighbors with Learned Metrics** (Weinberger et al., 2005) can learn, but are non-incremental.

More recent results combine neural architectures with forms of approximate nearest neighbor search to address these shortcomings. For example, (Rae et al., 2016) uses a representation learned for a task with either randomized kd-trees or locality sensitive hashing on a the Euclidean distance metric, both of which are periodically recomputed. The CMT instead learns at *individual nodes* and works for *any* representation, therefore, avoiding presupposing that a Euclidean metric is appropriate and could potentially productively replace the approximate nearest neighbor subsystem here.

Similarly, (Chandar et al., 2016) experiments with a variety of K-MIPS (Maximum Inner Product Search) data structures which the memory tree could potentially replace to create a higher ceiling on performance in situations where MIPS is not the right notion of similarity.

In (Andrychowicz & Kurach, 2016) the authors learn a hierarchical data structure over a pre-partitioned set of memories with a parameterized JOIN operator shared across nodes.

The use of pre-partition makes the data structure particularly sensitive to the (unspecified) order of that prepartition as discussed in appendix 6 of the LOMTree (Choromanska & Langford, 2015). Furthermore, tieing the parameters of JOIN across the nodes deeply constrains the representation compared to our approach.

Many of these shortcomings are addressed by **learned hashing**-based models (Salakhutdinov & Hinton, 2009; Rastegari et al., 2012), which learn a hash function that works well at prediction time, but all current approaches are non-incremental and require substantial training-time overhead. Finally, **differentiable memory systems** (Weston et al., 2014; Graves et al., 2016) are able to refine memories over time, but rely on gradient-descent-based techniques which incur a computational overhead that is inherently linear in the number of memories.

There are works on leveraging memory systems to perform few-shot learning tasks ((Snell et al., 2017; Strubell et al., 2017; Santoro et al., 2016)). However they are not logarithmic time and hence incapable of effectively operating at the large scales. Also note that they often address an easier version of the few-shot learning problem where training with a large number of labels for some classes is allowed as an initializer before the few-shot labels are observed. In contrast, we have no initialization phase.

2. The Contextual Memory Tree

At a high level, a CMT (Figure 2) is a near-balanced binary tree whose size dynamically increases as more memories are inserted. All memories are stored in leaf nodes with each leaf containing at most $c \log n$ memories, where n is the total number of memories and c is a constant independent of the number of memories.

Learning happens at every node of CMT. Each internal node contains a learning router. Given a query, CMT routes from the root to a leaf based on left-or-right decisions made by the routers along the way. Each internal node optimizes a metric, which ensures both its router's ability to predict which sub-tree contains the best memory corresponding to the query, and the balance between its left and right subtrees. CMT also contains a global learning scorer that predicts the reward of a memory for a query. The scorer is used at a leaf to decide which memories to return, with updates based on an external reward signal of memory quality.

2.1. Data Structures

A *memory* consists of a *query* (key) $x \in \mathcal{X}$ and its associated *value* $\omega \in \Omega$. We use z to denote the memory pair (x, ω) and define $\mathcal{Z} = \mathcal{X} \times \Omega$ as the set of z. Given a memory z, we use $z.x$ and $z.\omega$ to represent the query and the value of z respectively. For instance, for multiclass classification,

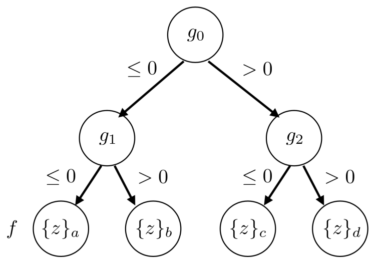


Figure 2: An example of CMT. Each internal node contains a binary classifier g as the router, and every leaf stores a small set of memories $\{z\}$. All leafs share a learning scorer f which computes a score of a memory z and a query x, and is used to select a memory when a query reaches a leaf.

x is a feature vector and ω is a label. Our memory store is organized into a binary tree. A *leaf* node in Figure 3 (left top) consists of a parent and a set of memories. Leaf nodes are connected by internal nodes as in Figure 3 (left, bottom). An internal node has a parent and two children, which may be either leaf or internal nodes, a count n of the number of memories beneath the node, and a learning router $g : \mathcal{X} \to \{-1, 1\}$ which both routes via $g(x)$ and updates via $g.\text{update}(x, y)$ for $y \in \{-1, 1\}$, or $g.\text{update}(x, y, i)$ where $i \in \mathbb{R}^+$ is an importance weight of (x, y). If $g(x) \leq 0$, we route x left, and otherwise right.

The contextual memory tree data structure in Figure 3 (right) has a root node, a parameter $\alpha \in [0, 1]$ which controls how balanced the tree is, a multiplier c on the maximum number of memories stored in any single leaf node, and a learning scorer $f : \mathcal{X} \times \mathcal{Z} \to \mathbb{R}$. Given a query x and memory z, the learning scorer predicts the reward one would receive if z is returned as the retrieved memory for query x via $f(x, z)$. Once a reward $r \in [0, 1]$ is received for a pair of memory z and query x, the learning scorer updates via $f.\text{update}(x, z, r)$ to improve its ability to predict reward. Finally, the map M maps examples to the leaf that contains them, making removal easy.

Given any internal node v and query x, we define a data structure path representing the path taken from v to a leaf: $\text{path} = \{(v_i, a_i, p_i)\}_{i=1}^N$, where $v_1 = v$, $a_i \in \{\text{left}, \text{right}\}$ is the left or right decision made at v_i, $p_i \in [0, 1]$ is the probability with which a_i was chosen. As we show later, path communicates to the update rule the information needed to create an unbiased update of routers.

2.2. Algorithms

All algorithms work given a contextual memory tree T. For brievity, we drop T when referencing its fields. We use $\in_U P$ to chose uniformly at random from a set P.

Algorithm 1 (PATH) routes a query x from any node v to a

```
NODE =
    LEAF NODE {
        parent : NODE,
        mem : MEMORIES }
  | INTERNAL NODE {
        parent : NODE,
        left, right : NODE,
        $g : \mathcal{X} \to \{-1, 1\}$ (learning router),
        $n$ : count of memories below }
```

```
CONTEXTUAL MEMORY TREE {
    root : TREE
    $f$ : learning scorer
    $M$ : hashmap from $x$ to the leaf holding $x$
    $\alpha \in (0, 1]$ : balance parameter
    $c$ : multiplier on the maximum leaf size
    $d$ : number of Reroute operations per insert }
```

Figure 3: Data structures for internal and leaf nodes (left) and the full contextual memory tree (right).

Algorithm 1 PATH(query x, node v)

```
1: path ← $\emptyset$
2: while $v$ is not a leaf do
3:     $a \leftarrow$ if $v.g(x) > 0$ then right else left
4:     Add $(v, a, 1)$ to path
5:     $v \leftarrow v.a$
6: end while
7: path.leaf ← $v$
8: return path
```

Algorithm 2 QUERY(query x, items to return k, exploration probability ϵ)

```
1: path ← PATH$(x, \text{root})$, path $= \{(v_i, a_i, p_i)\}_{i=1}^{N}$
2: $q \in_U [0, 1]$
3: if $q \geq \epsilon$ then
4:     return $(\emptyset, \text{top}_k(\text{path.leaf}, x))$
5: else
6:     Pick $i \in_U \{1, \ldots, N+1\}$
7:     if $i \leq N$ then
8:         Pick $a' \in_U \{\text{right}, \text{left}\}$
9:         $l =$ PATH$(x, v_i.a')$.leaf
10:        return $((v_i, a', 1/2), \text{top}_k(l, x))$
11:    else
12:        return $((\text{path.leaf}, \perp, \perp), \text{rand}_k(\text{path.leaf}, x))$
13:    end if
14: end if
```

leaf, returning the path traversed.

Algorithm 2 (QUERY) takes a query x as input and returns at most k memories. The parameter $\epsilon \in [0, 1]$ determines the probability of exploration during training. Algorithm 2 first deterministically routes the query x to a leaf and records the path traversed, path. With probability $1 - \epsilon$, we simply return the best memories stored in path.leaf: For a query x and leaf l, we use $\text{top}_k(l, x)$ as a shorthand for the set of $\min\{k, |l.\text{mem}|\}$ memories z in $l.\text{mem}$ with the largest $f(x, z)$, breaking ties randomly. We also use $\text{rand}_k(l, x)$ for a random subset of $\min\{k, |l.\text{mem}|\}$ memories in $l.\text{mem}$.

Algorithm 3 UPDATE$((x, z, r), (v, a, p))$

```
1: if $v$ is a leaf then
2:     $f$.update$(x, z, r)$
3: else
4:     $\hat{r} \leftarrow \frac{r}{p}(\mathbf{1}(a = \text{right}) - \mathbf{1}(a = \text{left}))$
5:     $y \leftarrow (1 - \alpha)\hat{r} + \alpha(\log v.\text{left}.n - \log v.\text{right}.n)$
6:     $v.g$.update$(x, \text{sign}(y), |y|)$
7: end if
8: Run REROUTE $d$ times
```

With the remaining probability ϵ, we uniformly sample a node along path including path.leaf. If we sampled an internal node v, we choose a random action a' and call path$(x, v.a')$ to route x to a leaf. This exploration gives us a chance to discover potentially better memories stored in the other subtrees beneath v, which allows us to improve the quality of the router at node v. We do uniform exploration at a uniformly chosen node but other schemes are possible. If we sampled path.leaf, we return a random set of memories stored in the leaf, in order to update and improve the learning scorer f. The shorter the path, the higher the probability that exploration happens at the leaf.

Algorithm 4 INSERT(node v, memory z, Reroute d)

```
1: while $v$ is not a leaf do
2:     $B = \log v.\text{left}.n - \log v.\text{right}.n$
3:     $y \leftarrow \text{sign}((1 - \alpha)v.g(z.x) + \alpha B)$
4:     $v.g$.update$(z.x, y)$
5:     $v.n \leftarrow v.n + 1$
6:     $v \leftarrow$ if $v.g(z.x) > 0$ then $v$.right else $v$.left
7: end while
8: INSERTLEAF($v$, $z$)
9: Run REROUTE $d$ times
```

After a query for x, we may receive a reward r for a returned memory z. In this case, Algorithm 3 (UPDATE) uses the first triple returned by QUERY to update the router making a randomized decision. More precisely, Algorithm 3

Algorithm 5 INSERTLEAF(leaf node v, memory z)

```
1: $v$.mem $\leftarrow v$.mem $\cup \{z\}$
2: if $|v.\text{mem}| > c\log(\text{root}.n)$ then
3:    $v' \leftarrow$ a new internal node with two new children
4:    for $z \in v$.mem do
5:       INSERT$(v', z, 0)$
6:    end for
7:    $v \leftarrow v'$
8: end if
```

computes an unbiased estimate of the reward difference of the left/right decision which is then mixed with a balance-inducing term on line 5. When randomization occurred at the leaf, the scorer f is updated instead.

The INSERT operation is given in Algorithm 4. It routes the memory z to be inserted according to the query $z.x$ from the root to a leaf using internal learning routers, updating them on descent. Once reaching a leaf node, z is added into that leaf via INSERTLEAF. The label definition on line 3 in INSERT is the same as was used in (Beygelzimer et al., 2009). That use, however, was for a different problem (conditional label estimation) and is applied differently (controlling the routing of examples rather than just advising a learning algorithm). As a consequence, the proofs of correctness given in section 3.1 differ.

When the number of memories stored in any leaf exceeds the log of the total number of memories, a leaf is split according to Algorithm 5 (INSERTLEAF). The leaf node v is promoted to an internal node with two leaf children and a binary classifier g with all memories inserted at v.

Because updates are online, they may result in a lack of self-consistency for previous insertions. This is fixed by REROUTE (Algorithm 7) on an amortized basis. Specifically, after every INSERT operation we call REROUTE, which randomly samples an example from all the examples, extracts the sampled example from the tree, and then re-inserts it. This relies on the REMOVE (Algorithm 6) operation, which finds the location of a memory using the hashmap then ascends to the parent cleaning up accounting. When a leaf node has zero memories, it is removed.

3. Properties

There are five properties that we want CMT to satisfy simultaneously (see Figure 1 (left) for the five properties). Storage (in appendix A.1) and Incrementality (in appendix A.2) are easy observations.

Appendix A.6 shows that in the limit of many REROUTEs, self-consistency (defined below) is achieved.

Definition 3.1 *A* CMT *is* **self-consistent** *if for all z with a unique $z.x$, $z =$*QUERY$(z.x, 1, 0)$.

Algorithm 6 REMOVE(x)

```
 1: Find $v \leftarrow M(x)$, leaf containing $x$
 2: $v$.mem $\leftarrow v$.mem $\setminus \{x\}$
 3: while $v$ is not root do
 4:    if $v.n > 0$ then
 5:       $v.n \leftarrow v.n - 1$
 6:       $v \leftarrow v$.parent
 7:    else
 8:       $v' =$ the other child of $v$.parent.
 9:       $v$.parent $\leftarrow v'$
10:       $v \leftarrow v'$
11:    end if
12: end while
```

Algorithm 7 REROUTE()

```
1: Sample $z \in_U M$
2: REMOVE$(z.x)$
3: INSERT$(\text{root}, z, 0)$
```

Appendix A.7 shows a learning property: Every internal router asymptotically optimizes to a local maxima of an objective function that mirrors line 5 of UPDATE.

This leaves only logarithmic computational time, which we address next.

3.1. Computational Time

The computational time analysis naturally breaks into two parts, partition quality at the nodes and the time complexity given good partitions. To connect the two, we first define partition quality.

Definition 3.2 *A* **K-balanced partition** *of any set has each element of the partition containing at least a $1/K$ fraction of the original set.*

When partitioning into two sets, $K \geq 2$ is required. Smaller K result in smaller computational complexities at the cost of worse predictive performance in practice.

Define the progressive training error of a learning router g after seeing T examples $x_1, \ldots, x_T$ as $p = \frac{1}{T}\sum_{t=1}^{T} \mathbf{1}[g(x_t) \neq y_t]$, where y_t is the label assigned in line 3 of INSERT, and $g(x_t)$ is evaluated immediately after calling $g.\text{update}(x_t, y_t)$ so a mistake occurs when $g(x_t)$ disagrees with y_t after the update. The next theorem proves a bound on the partition balance dependent on the progressive training error of a node's router and α.

Theorem 3.3 *(Partition bound) At any point, a router with a progressive training error of p creates a*

$\frac{1+\exp(\frac{1-\alpha}{\alpha})}{(1-p)-\left(1+\exp(\frac{1-\alpha}{\alpha})\right)\frac{1}{T}}$*-balanced partition.*

The proof is in appendix A.3, followed by a bound on the depth of K-partition trees in appendix A.4. As long as $(1-p) > \exp(\frac{1-\alpha}{\alpha})\frac{1}{T}$ holds, Theorem 3.3 provides a nontrivial bound on partition. Examining limits, when $p = 0$, $\alpha = 1$ and $T = \infty$, we have $K = 2$, which means CMT becomes a perfectly balanced binary tree. If $p = 0.5$ (e.g., g guesses at random), $\alpha = 0.9$ (used in all our experiments) and $T = \infty$, we have $K \leq 4.3$. For any fixed T, a smaller progressive error p and a larger α lead to a smaller K.

Next, we prove that K controls the computational time.

Theorem 3.4 *(Computational Time) If every router g in a CMT with T previous calls to* INSERT *creates a K-partition, the worst case computation is $O(d(K+c)\log T)$ for* INSERT, *$O((K+c)\log T)$ for* QUERY, *and $O(1)$ for* UPDATE *if all stated operations are atomic.*

The proof is in appendix A.5. This theorem establishes logarithmic time computation given that K-partitions are created. These two theorems imply that the computation is logarithmic time for all learning algorithms achieving a *training* error significantly better than 1.

4. Experiments

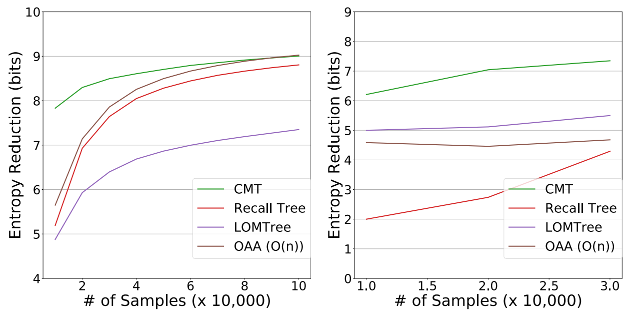
CMT is a subsystem for other learning tasks, so it assists other inference and learning algorithms. We test the application of CMT to three systems, for multiclass classification, multilabel classification, and image retrieval. Seperately, we also ablate various elements of CMT to discover its strengths and weaknesses.

We implemented CMT as a reduction to Vowpal Wabbit's (Langford et al., 2007) default learning algorithm. [1] The routers (g) and the learning scorer (f) are all linear functions and are incrementally updated by an Adagrad (Duchi et al., 2011) gradient method in VW. Similarly, most baselines are implemented in the same system with a similar or higher level of optimization.

4.1. Application: Online Extreme Multi-Class Classification

Since CMT operates online, we can evaluate its online performance using progressive validation (Blum et al., 1999) (i.e., testing each example ahead of training). Used online, we QUERY for an example, evaluate its loss, then apply UPDATE with the observed loss followed by INSERT of the data point. In a multiclass classification setting, a memory z is a feature vector x and label ω. Given a query x, CMT returns a memory z and receives a reward signal $\mathbf{1}[z.\omega = \omega]$ for update. Finally, CMT inserts (x, ω).

[1] https://github.com/LAIRLAB/vowpal_wabbit/tree/master/demo/memory_tree



(a) ALOI (b) WikiPara 3-shot

Figure 4: (a) Online progressive performance of CMT with respect to the number of samples on ALOI (a) and WikiPara 3-shot (b). CMT consistently outperforms all baselines.

We test the online learning ability of CMT on two multiclass classification datasets, ALOI (1000 labels with 100 examples per label) and WikiPara 3-shot (10000 labels with 3 examples per label), against two other logarithmic-time online multiclass classification algorithms, LOMTree (Choromanska & Langford, 2015) and Recall Tree (Daumé et al., 2017). We also compare against a linear-time online multiclass classification algorithm, One-Against-All (OAA).

Figure 4 summarizes the results in terms of progressive performance. On both datasets, we report entropy reduction from the constant predictor (the higher the better). The entropy reduction of a predictor A from another predictor B is defined as $\log_2(p_A) - \log_2(p_B)$, where p_A and p_B are prediction accuracies of A and B.

Conclusion: CMT greatly outperforms the baselines in the small number of examples per label regime. This appears to be primarily due to the value of explicit memories over learned parameters in this regime.

4.2. Application: Batch Few-shot Multi-Class Classification

We can also use CMT in an offline testing mode as well by using CMT with multiple passes over the the training dataset and testing it on a separate test set. We again use CMT on few-shot multi-class classification, comparing it to LOMTree and Recall Tree.

Starting first with the ALOI dataset, we tested both the unsupervised version (i.e., using only INSERT) and the supervised version (i.e., using INSERT for the first pass, and using UPDATE for subsequent passes). We used three passes for all algorithms. The supervised version of CMT achieved 26.3% test prediction error, outperforming LOMTree (66.7%) and

	RCV1-1K			AmazonCat-13K			Wiki10-31K		
Approach	loss	Test time	Train time	loss	Test time	Train time	loss	Test time	Train time
CMT	2.5	1.4ms	1.9hr	3.2	1.7ms	5.3hr	18.3	25.3ms	1.3hr
OAA	2.6	0.5ms	1.3hr	3.0	8.7ms	15.5hr	20.3	327.1ms	7.2hr

Table 1: Hamming Loss, test time per example (ms), and training time (hr) for multi-label tasks.

Recall Tree (28.8%). The supervised version of CMT also significantly outperforms the unsupervised one (75.8% error rate), showing the benefit of the UPDATE procedure. Since ALOI has 1000 classes, a constant predictor has prediction error larger than 99%.

We then test CMT on more challenging few-shot multi-class classification datasets, WikiPara S-shot ($S = 1, 2, 3$) and ImageNet S-shot ($S = 1, 2, 3, 5$) with only S examples per label. Figure 1 summarizes the statistical performance (entropy reduction compared to a constant predictor) of supervised CMT, unsupervised CMT (denoted as CMT (u)), and the two logarithmic-time baselines. For one-shot experiments (WP 1-s and IN 1-s on Figure 1), CMT outperforms all baselines. The edge of CMT degrades gradually over baselines as S increases (IN Ss with $S > 1$ in Figure 1). All details are included in Table 6 in Appendix §B.3.

Conclusion: The high performance of CMT with a small number of examples per label persists in batch training. The remarkable performance of unsupervised CMT over supervised baselines suggests self-consistency can provide nearest-neighbor performance without explicit reward.

4.3. Application: Multi-Label Classification with an External Inference Algorithm

In this set of experiments, instead of using CMT as an inference algorithm, we integrate CMT with an external inference procedure based on One-Against-All. CMT is not aware of the external multi-label classification task, so this is an example of how an external inference algorithm can leverage the returned memories as an extra source of information to improve performance. Here each memory z consists of a feature vector x and label vector $\omega \in \{0, 1\}^M$, where M is the number of unique labels. Given a query x, its ground truth label vector ω, and a memory z, we choose the F1-score between ω and $z.\omega$ as the reward signal. We set k to $c \log(N)$ (i.e., CMT returns all memories in the leaf we reach). Given a query x, with the returned memories $\{z_1, ..., z_k\}$, the external inference procedure extracts the unique labels from the returned memories and performs a One-Against-Some (OAS) inference (Daumé et al., 2017) using the extracted labels.[2] The external system then calls UPDATE for the returned memories. Since CMT returns logarithmically many memories, we guarantee that the number of unique labels from the returned memories is also logarithmic. Hence augmenting OAS with CMT enables logarithmic inference and training time.

[2]OAS takes x and a small set of candidate labels and returns the labels with a positive score, according to a learned scoring function. After prediction, the OAS predictor receives the true labels y associated with this x and performs an update to its score function based on the true labels and the small candidate label set.

We compare CMT-augmented OAS with multi-label OAA under the Hamming loss. We compare CMT-augmented OAS to OAA on three multi-label datasets, RCV1-1K (Prabhu & Varma, 2014), AmazonCat-13K (McAuley & Leskovec, 2013), and Wiki-31K (Zubiaga, 2012; Bhatia et al., 2015). (The datasets are described in Table 4 in Appendix B.1.) Table 1 summarizes the performance of CMT and OAA. (LOMTree and Recall Tree are excluded because they do not operate in multi-label settings.)

Conclusion: CMT-augmented OAS achieves similar statistical performance to OAA, even mildly outperforming OAA on Wiki10-31K, while gaining significant computational speed up over a vector optimized OAA in training and inference on datasets with a large number of labels (e.g., AmazonCat-13K and Wiki10-31K). Note that the VW implementation of OAA operates at a higher level optimization and involves vectorized computations that increase throughput by a factor of 10 to 20. Hence we observe for RCV1-1K with 1K labels, OAA can actually be more computationally efficient then CMT. This set of experiments shows that CMT-augmented OAS can win over OAA both statistically and computationally for challenging few-shot multi-label datasets with a large number of labels.

4.4. Application: Image Retrieval

We test CMT on an image retrieval task where the goal is to find an image given a caption. We used three benchmark datasets, (1) UIUC Pascal Dataset (Rashtchian et al., 2010), (2) Flickr8k dataset (Hodosh et al., 2013), and (3) MS COCO (Lin et al., 2014), with feature representations described in §B.1. Here, a memory z consists of (features of) a caption x and an image ω. Given a query, CMT returns a memory $z = (x, \omega)$. Our reward function is the cosine similarity between the returned memory's image $z.\omega$, and the ground truth image ω associated with the query x.

To show the benefit of learning in CMT, we compare it to Nearest Neighbor Search (NNS) and a KD-Tree as an Approximate NN data structure on this task, using the Euclidean distance $\|x - z.x\|_2$ in the feature space of captions as the NNS metric. Both CMT and NNS are tested on a separate test set, with the average reward of the retrieved memory reported.

Table 2 summarizes the speedup over NNS (implemented using a linear scan) and KD-Tree (KD tree implementation from scikit-learn (Pedregosa et al., 2011)). Note that in our datasets, the feature of a query is high dimensional (2^{20}) but extremely sparse. Since KD-Tree cannot take advantage of sparsity, both the construction and inference procedure is extremely slow (even slower than a NNS). We also emphasize here that a KD-Tree does not operate in an online manner. Hence in our experiments, we have to feed all queries from the entire training dataset to KD-Tree to initialize its construction, which makes it impossible to initialize the run of KD-Tree on MSCOCO.

Conclusion: The difference in reward is negligible (on the order of 10^{-3}) and statistically insignificant. (See Appendix Table 7 for details.) However, CMT is significantly faster.

	CMT	
	unsup	sup
Pascal	5.7 / 9400	1.3 / 2100
Flickr8k	26.0 / 33000	6.0 / 7700
MSCOCO	21.0 / $\sim$	6.5 / $\sim$

Table 2: Speedups over linear NNS (left) and KD-Tree (right), in (unsup)ervised and (sup)ervised mode.

4.5. Ablation Analysis of CMT

We conduct experiments to perform an ablation study of CMT in the context of multi-class classification, where it operates directly as an inference algorithm.

We test the self-consistency property on WikiPara with only one training example per class (see Figure 5a). We ran CMT in an unsupervised fashion, by only calling INSERT and using $-\|x - z.x\|$ as $f(x, z)$ to select memories at leafs. We report the self-consistency error with respect to the number of REROUTE calls per insertion (parameter d) after four passes over the dataset (tuned using a holdout set). As d increases, the self-consistency error rapidly drops.

To show that UPDATE is beneficial, we use multiple passes to drive the training error down to nearly zero. Figure 5b shows the training error versus the number of passes on the WikiPara one-shot dataset (on the x-axis, we plot the number of additional passes over the dataset, with zero corresponding to a single pass). Note that the training error is essentially equal to the self-consistency error in WikiPara One-shot, hence UPDATE further enhances self-consistency due to the extra REROUTE operations in UPDATE.

To test the effect of the multiplier c (the leaf memories multiplier), we switch to the ALOI dataset (Geusebroek et al., 2005), which has 100 examples per class enabling good generalization. Figure 5c shows that statistical performance improves with inference time and the value of c.

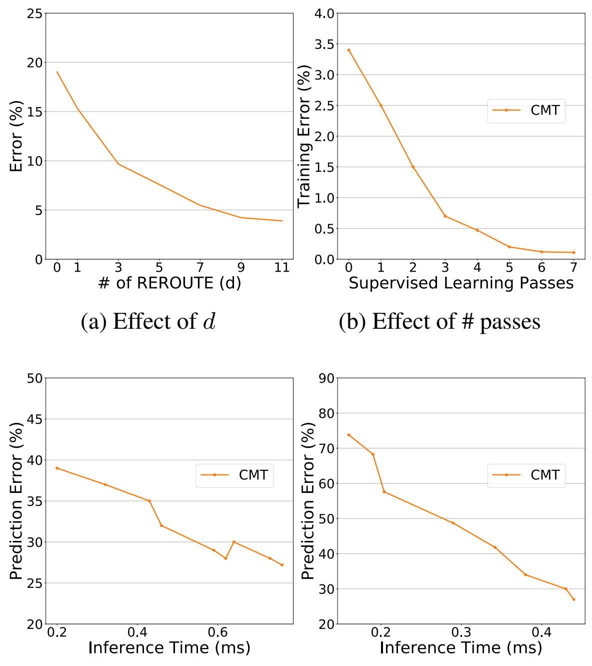


Figure 5: (a) As the number of REROUTE operations increases CMT performs better, asymptoting at 10 on WikiPara one-shot, (b) the effect of the number of UPDATE calls on the training error on WikiPara one-shot, (c) the inference time and prediction error with respect to c on ALOI, and (d) the inference time and prediction error with respect to number of shots on ALOI.

In Appendix §B.2, we include plots showing statistical and inference time performance vs c in Figure 6 with inference time scaling linearly in c as expected.

Last, we test CMT on a series of progressively more difficult datasets generated from ALOI via randomly sampling S training examples per label, for S in 1 to 100. ALOI has 1000 unique labels so the number of memories CMT stores scales as $S \times 1000$, for S-shot ALOI. We fix $c = 4$. Figure 5d shows the statistical performance vs inference time as S varies. The prediction error drops quickly as we increase S. Appendix §B.2 includes detailed plots. Inference time increases logarithmically with S (Figure 7b), matching CMT's logarithmic time operation theory.

5. Conclusion

CMT provides a new tool for learning algorithm designers by enabling learning algorithms to work with an unsupervised or reinforced logarithmic time memory store. Empirically, we find that CMT provides remarkable unsupervised performance, sometimes beating previous supervised algorithms while reinforcement provides steady improvements.

References

Andrychowicz, M. and Kurach, K. Learning efficient algorithms with hierarchical attentive memory. *CoRR*, abs/1602.03218, 2016. URL http://arxiv.org/abs/1602.03218.

Bartl, A. and Spanakis, G. A retrieval-based dialogue system utilizing utterance and context embeddings. 2017. URL http://arxiv.org/abs/1710.05780.

Beygelzimer, A., Kakade, S., and Langford, J. Cover trees for nearest neighbor. In *Machine Learning, Proceedings of the Twenty-Third International Conference (ICML 2006), Pittsburgh, Pennsylvania, USA, June 25-29, 2006*, pp. 97–104, 2006. doi: 10.1145/1143844.1143857. URL http://doi.acm.org/10.1145/1143844.1143857.

Beygelzimer, A., Langford, J., Lifshits, Y., Sorkin, G., and Strehl, A. Conditional probability tree estimation analysis and algorithms. In *Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence*, pp. 51–58. AUAI Press, 2009.

Bhatia, K., Jain, H., Kar, P., Varma, M., and Jain, P. Sparse local embeddings for extreme multi-label classification. In *Advances in Neural Information Processing Systems*, pp. 730–738, 2015.

Blum, A., Kalai, A., and Langford, J. Beating the hold-out: Bounds for k-fold and progressive cross-validation. In *Proceedings of the twelfth annual conference on Computational learning theory*, pp. 203–208. ACM, 1999.

Broder, A. Z., Carmel, D., Herscovici, M., Soffer, A., and Zien, J. Y. Efficient query evaluation using a two-level retrieval process. In *Proceedings of the 2003 ACM CIKM International Conference on Information and Knowledge Management, New Orleans, Louisiana, USA, November 2-8, 2003*, pp. 426–434, 2003. doi: 10.1145/956863.956944. URL http://doi.acm.org/10.1145/956863.956944.

Brouwer, L. E. J. ber abbildungen von mannigfaltigkeiten. *Mathematische Annalen*, 71:97–115, 1911.

Cesa-Bianchi, N. and Lugosi, G. *Prediction, learning, and games*. Cambridge University Press, 2006. ISBN 978-0-521-84108-5.

Chandar, S., Ahn, S., Larochelle, H., Vincent, P., Tesauro, G., and Bengio, Y. Hierarchical memory networks. *arXiv preprint arXiv:1605.07427*, 2016.

Choromanska, A. E. and Langford, J. Logarithmic time online multiclass prediction. In *Advances in Neural Information Processing Systems*, pp. 55–63, 2015.

Dasgupta, S. and Sinha, K. Randomized partition trees for nearest neighbor search. *Algorithmica*, 72(1):237–263, 2015. doi: 10.1007/s00453-014-9885-5. URL https://doi.org/10.1007/s00453-014-9885-5.

Datar, M., Immorlica, N., Indyk, P., and Mirrokni, V. S. Locality-sensitive hashing scheme based on p-stable distributions. In *Proceedings of the 20th ACM Symposium on Computational Geometry, Brooklyn, New York, USA, June 8-11, 2004*, pp. 253–262, 2004. doi: 10.1145/997817.997857. URL http://doi.acm.org/10.1145/997817.997857.

Daumé, III, H., Karampatziakis, N., Langford, J., and Mineiro, P. Logarithmic time one-against-some. *ICML*, 2017.

Duchi, J., Hazan, E., and Singer, Y. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(Jul):2121–2159, 2011.

Freund, Y. and Schapire, R. E. A decision-theoretic generalization of on-line learning and an application to boosting. *J. Comput. Syst. Sci.*, 55(1):119–139, 1997. doi: 10.1006/jcss.1997.1504. URL https://doi.org/10.1006/jcss.1997.1504.

Geusebroek, J.-M., Burghouts, G. J., and Smeulders, A. W. The amsterdam library of object images. *International Journal of Computer Vision*, 61(1):103–112, 2005.

Graves, A., Wayne, G., Reynolds, M., Harley, T., Danihelka, I., Grabska-Barwinska, A., Colmenarejo, S. G., Grefenstette, E., Ramalho, T., Agapiou, J., Badia, A. P., Hermann, K. M., Zwols, Y., Ostrovski, G., Cain, A., King, H., Summerfield, C., Blunsom, P., Kavukcuoglu, K., and Hassabis, D. Hybrid computing using a neural network with dynamic external memory. *Nature*, 538(7626):471–476, 2016. doi: 10.1038/nature20101. URL https://doi.org/10.1038/nature20101.

Gu, J., Wang, Y., Cho, K., and Li, V. O. K. Search engine guided non-parametric neural machine translation. In *AAAI*, 2018.

Hodosh, M., Young, P., and Hockenmaier, J. Framing image description as a ranking task: Data, models and evaluation metrics. *Journal of Artificial Intelligence Research*, 47:853–899, 2013.

Kaiser, L., Nachum, O., Roy, A., and Bengio, S. Learning to remember rare events. *ICLR*, 2017.

Karnin, Z. S., Liberty, E., Lovett, S., Schwartz, R., and Weinstein, O. Unsupervised svms: On the complexity of the furthest hyperplane problem. In *COLT 2012 - The 25th Annual Conference on Learning Theory, June 25-27, 2012, Edinburgh, Scotland*, pp. 2.1–2.17, 2012. URL http://jmlr.org/proceedings/papers/v23/karnin12/karnin12.pdf.

Knuth, D. E. *The art of computer programming, Volume I: Fundamental Algorithms, 3rd Edition*. Addison-Wesley, 1997. ISBN 0201896834. URL http://www.worldcat.org/oclc/312910844.

Langford, J., Li, L., and Strehl, A. Vowpal wabbit online learning project, 2007.

Lin, T.-Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., Dollár, P., and Zitnick, C. L. Microsoft coco: Common objects in context. In *European conference on computer vision*, pp. 740–755. Springer, 2014.

McAuley, J. and Leskovec, J. Hidden factors and hidden topics: understanding rating dimensions with review text. In *Proceedings of the 7th ACM conference on Recommender systems*, pp. 165–172. ACM, 2013.

Oquab, M., Bottou, L., Laptev, I., and Sivic, J. Learning and transferring mid-level image representations using convolutional neural networks. In *Computer Vision and Pattern Recognition (CVPR), 2014 IEEE Conference on*, pp. 1717–1724. IEEE, 2014.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., et al. Scikit-learn: Machine learning in python. *Journal of machine learning research*, 12(Oct):2825–2830, 2011.

Prabhu, Y. and Varma, M. Fastxml: A fast, accurate and stable tree-classifier for extreme multi-label learning. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 263–272. ACM, 2014.

Rae, J., Hunt, J. J., Danihelka, I., Harley, T., Senior, A. W., Wayne, G., Graves, A., and Lillicrap, T. Scaling memory-augmented neural networks with sparse reads and writes. In *NIPS*, 2016.

Rashtchian, C., Young, P., Hodosh, M., and Hockenmaier, J. Collecting image annotations using amazon's mechanical turk. In *Proceedings of the NAACL HLT 2010 Workshop on Creating Speech and Language Data with Amazon's Mechanical Turk*, pp. 139–147. Association for Computational Linguistics, 2010.

Rastegari, M., Farhadi, A., and Forsyth, D. A. Attribute discovery via predictable discriminative binary codes. In *Computer Vision - ECCV 2012 - 12th European Conference on Computer Vision, Florence, Italy, October 7-13, 2012, Proceedings, Part VI*, pp. 876–889, 2012. doi: 10.1007/978-3-642-33783-3_63. URL https://doi.org/10.1007/978-3-642-33783-3_63.

Salakhutdinov, R. and Hinton, G. E. Semantic hashing. *Int. J. Approx. Reasoning*, 50(7):969–978, 2009. doi: 10.1016/j.ijar.2008.11.006. URL https://doi.org/10.1016/j.ijar.2008.11.006.

Santoro, A., Bartunov, S., Botvinick, M., Wierstra, D., and Lillicrap, T. One-shot learning with memory-augmented neural networks. *arXiv preprint arXiv:1605.06065*, 2016.

Shrivastava, A. and Li, P. Improved asymmetric locality sensitive hashing (ALSH) for maximum inner product search (MIPS). In *Proceedings of the Thirty-First Conference on Uncertainty in Artificial Intelligence, UAI 2015, July 12-16, 2015, Amsterdam, The Netherlands*, pp. 812–821, 2015.

Simonyan, K. and Zisserman, A. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

Snell, J., Swersky, K., and Zemel, R. Prototypical networks for few-shot learning. In *Advances in Neural Information Processing Systems*, pp. 4077–4087, 2017.

Strubell, E., Verga, P., Belanger, D., and McCallum, A. Fast and accurate entity recognition with iterated dilated convolutions. *arXiv preprint arXiv:1702.02098*, 2017.

Weinberger, K. Q., Blitzer, J., and Saul, L. K. Distance metric learning for large margin nearest neighbor classification. In *Advances in Neural Information Processing Systems 18 [Neural Information Processing Systems, NIPS 2005, December 5-8, 2005, Vancouver, British Columbia, Canada]*, pp. 1473–1480, 2005. URL http://papers.nips.cc/paper/2795-distance-metric-learning-for-large-margin-nearest-neighbor-classification.

Weston, J., Chopra, S., and Bordes, A. Memory networks. *CoRR*, abs/1410.3916, 2014. URL http://arxiv.org/abs/1410.3916.

Xu, L. and Schuurmans, D. Unsupervised and semi-supervised multi-class support vector machines. In *Proceedings, The Twentieth National Conference on Artificial Intelligence and the Seventeenth Innovative Applications of Artificial Intelligence Conference, July 9-13, 2005, Pittsburgh, Pennsylvania, USA*, pp. 904–910, 2005. URL http://www.aaai.org/Library/AAAI/2005/aaai05-143.php.

Zubiaga, A. Enhancing navigation on wikipedia with social tags. *arXiv preprint arXiv:1202.5469*, 2012.