Adaptive Memory Networks

Daniel Li \(^1\) Asim Kadav \(^2\)

Abstract

We present Adaptive Memory Networks (AMN) that processes input-question pairs to dynamically construct a network architecture optimized for lower inference times for Question Answering (QA) tasks. AMN processes the input story to extract entities and stores them in memory banks. Starting from a single bank, as the number of input entities increases, AMN learns to create new banks as the entropy in a single bank becomes too high. Hence, after processing an input-question(s) pair, the resulting network represents a hierarchical structure where entities are stored in different banks, distanced by question relevance. At inference, one or few banks are used, creating a tradeoff between accuracy and performance.

AMN is enabled by dynamic networks that allow input dependent network creation and efficiency in dynamic mini-batching as well as our novel bank controller that allows learning discrete decision making with high accuracy. In our results, we demonstrate that AMN learns to create variable depth networks depending on task complexity and reduces inference times for QA tasks.

1. Introduction

Question Answering (QA) tasks are gaining significance due to their widespread applicability to recent commercial applications such as chatbots, voice assistants and even medical diagnosis (Goodwin & Harabagiu, 2016). Furthermore, many existing natural language tasks can also be re-phrased as QA tasks. Providing faster inference times for QA tasks is crucial. Consumer device based question-answer services have hard timeouts for answering questions. For example, Amazon Alexa, a popular QA voice assistant, allows developers to extend the QA capabilities by adding new “Skills” as remote services (Amazon, 2017). However, these service APIs are wrapped around hard-timeouts of 8 seconds which includes the time to transliterate the question to text on Amazon’s servers and the round-trip transfer time of question and the answer from the remote service, and sending the response back to the device. Furthermore, developers are encouraged to provide a list of questions (“utterances”) apriori at each processing step to assist QA processing (Amazon, 2017). We propose that apart from helping transliteration these questions can also provide hints for reducing inference times for QA tasks based on large knowledge bases.

Modeling QA tasks with LSTMs can be computationally expensive which is undesirable during inference. Memory networks, a class of deep networks with explicit addressable memory, have recently been used to achieve state of the art results on many QA tasks. Unlike LSTMs, where the number of parameters grows exponentially with the size of memory, memory networks are comparably parameter efficient and can learn over longer input sequences. However, they often require accessing all intermediate memory to answer a question. Furthermore, using focus of attention over the intermediate state using a list of questions does not address this problem. Soft attention based models compute a softmax over all states and hard attention models are not differentiable and can be difficult to train over a large state space. Previous work on improving inference over memory networks has focused on using unsupervised clustering methods to reduce the search space (Chandar et al., 2016; Rae et al., 2016). Here, the memory importance is not learned and the performance of nearest-neighbor style algorithms is often comparable to a softmax operation over memories.

To provide faster inference for long sequence-based inputs, we present Adaptive Memory Networks (AMN), that constructs a memory network on-the-fly based on the input. Like past approaches to addressing external memory, AMN constructs the memory entity nodes dynamically. However, distinct from past approaches, AMN stores the entities from the input story in variable number of memory banks. The entities represent the hidden state of each word in the story while a memory bank is a collection of entities that are similar w.r.t the question. As the number of entities grow, and the entropy within a single bank becomes too high, our network learns to construct new memory banks and copies entities that are more relevant towards a single bank. Hence,
by limiting the decoding step to a dynamic number of relevant memory banks, AMN achieves lower inference times. AMN is an end-to-end trained model with dynamic learned parameters for memory bank creation and movement of entities.

Figure 1 demonstrates a simple QA task where AMN constructs two memory banks based on the input. During inference only the entities in the left bank are considered reducing inference times. To realize its goals, AMN introduces a novel bank controller that uses reparameterization trick to make discrete decisions with high accuracy while maintaining differentiability. Finally, AMN also models sentence structures on-the-fly and propagates update information for all entities that allows it to solve all 20 bAbI tasks.

2. Related Work

Memory Networks: Memory networks store the entire input sequence in memory and perform a softmax over hidden states to update the controller (Weston et al., 2014; Sukhbaatar et al., 2015). DMN+ connects memory to input tokens and updates them sequentially (Xiong et al., 2016). For inputs that consist of large number of tokens or entities, these methods can be expensive during inference. AMN stores entities with tied weights in different memory banks. By controlling the number of memory banks, AMN achieves low inference times with reasonable accuracy. Nearest neighbor methods have also been explored over memory networks. For example, Hierarchical Memory Networks separates the input memory into groups using the MIPS algorithm (Chandar et al., 2016). However, using MIPS is as slow as a softmax operation, so the authors propose using an approximate MIPS that gives inferior performance. In contrast, AMN is end to end differentiable, and reasons which entities are important and constructs a network with dynamic depth.

Neural Turing Machine (NTM) consists of a memory bank and a differentiable controller that learns to read and write to specific locations (Graves et al., 2014). In contrast to NTMs, AMN memory bank controller is more coarse grained and the network learns to store entities in memory banks instead of specific locations. AMN uses a discrete bank controller that gives improved performance for bank controller actions over NTM’s mechanisms. However, our design is consistent with the modeling studies of working memory by (Hazy et al., 2006) where the brain performs robust memory maintenance and may maintain multiple working representations for individual working tasks. Sparse access memory uses approximate nearest neighbors (ANN) to reduce memory usage in NTMs (Rae et al., 2016). However, ANNs are not differentiable. AMN, uses a input specific memory organization that does not create sparse structures. This limits access during inference to specific entities reducing inference times.

Graph-based networks, (GG-NNs (Li et al., 2015) and GGT-NNs (Johnson, 2017)) use nodes with tied weights that are updated based on gated-graph state updates with shared weights over edges. However, unlike AMN, they require strong supervision over the input and teacher forcing to learn the graph structure. Furthermore, the cost of building and training these models is expensive and if every edge is considered at every time-step the amount of computation grows at the order of $O(N^3)$ where $N$ represents the number of nodes/entities. AMN does not use strong supervision but can solve tasks that require transitive logic by modeling sentence walks on the fly. EntNet constructs dynamic networks based on entities with tied weights for each entity (Henaff et al., 2017). A key-value update system allows it to update relevant (learned) entities. However, EntNet uses soft-attention during inference to attend to all entities that incur high inference costs. To summarize, majority of the past work on memory networks uses softmax over memory nodes, where each node may represent input or an entity. In contrast, AMN learns to organize memory into various memory banks and performs decode over fewer entities reducing inference times.

Conditional Computation & Efficient Inference:
AMN is also related to the work on conditional computation which allows part of networks to be active during inference improving computational efficiency (Bengio et al., 2015). Recently, this has been often accomplished using a gated mixture of experts (Eigen et al., 2013; Shazeer et al., 2017). AMN conditionally attends to entities in initial banks during inference improving performance. For faster inference using CNNs, pruning (Le Cun et al., 1989; Han et al., 2016), low rank approximations (Denton et al., 2014), quantization and binarization (Rastegari et al., 2016) and other tricks to improve GEMM performance (Vanhoucke et al., 2011) have been explored. For sequence based inputs, pruning and compression has been explored (Giles & Omlin, 1994; See et al., 2016). However, compression results in irregular sparsity that reduces memory costs but may not reduce computation costs. Adaptive computation time (Graves, 2016) learns the number of steps required for inferring the output and this can also be used to reduce inference times (Figurnov et al., 2016). AMN uses memory networks with dynamic number of banks to reduce computation costs.

Dynamic networks: Dynamic neural networks that change structure during inference have recently been possible due to newer frameworks such as Dynet and PyTorch. Existing work on pruning can be implemented using these frameworks to reduce inference times dynamically like dynamic deep network demonstrates (Liu & Deng, 2017).
Figure 1. Overview of Adaptive memory networks. Multiple memory banks are created based on the story and input entities are moved in them based on their relevance to the question. Inference is performed on a single (or less than all) banks most relevant to the question(s).

AMN utilizes the dynamic architecture abilities to construct an input dependent memory network of variable memory bank depth and the dynamic batching feature to process a variable number of entities. Furthermore, unlike past work that requires an apriori number of fixed memory slots, AMN constructs them on-the-fly based on the input. The learnable discrete decision-making process can be extended to other dynamic networks which often rely on REINFORCE to make such decisions (Liu & Deng, 2017).

Neuroscience: Our network construction is inspired by work on working memory representations. There is sufficient evidence for multiple, working memory representations in the human brain (Hazy et al., 2006). Semantic memory (Tulving et al., 1972), describes a hierarchical organization starting with relevant facts at the lowest level and progressively more complex and distant concepts at higher levels. AMN constructs entities from the input stories and stores the most relevant entities based on the question in the lowest level memory bank. Progressively higher level memory banks represent distant concepts (and not necessarily higher level concepts for AMN). Other work demonstrates organization of human memory in terms of “priority structure” where attention is a gate-keeper of working memory-guided by executive control’s goals, plans, and intentions as in (Watzl, 2017), similar in spirit to AMN’s question guided network construction.

3. Differentiable adaptive memory module

In this section, we describe the design process and motivation of our memory module. Our memory network architecture is created during inference time for every story. The architecture consists of different memory banks and each memory bank stores entities from the input story. Hence, a memory entity represents the hidden state of each entity (each word in our case) from the input story while a memory bank is a collection of entities. Intuitively, each memory bank stores entities that have a similar distance score from the question.

At a high level, entities are gradually and recurrently copied through memory banks to filter out irrelevant nodes such that in the final inference stage, fewer entities are considered by the decoder. Note that the word filter implies a discrete decision and that recurrence implies time. If we were to perform a strict cut off and remove entities that appear to be irrelevant at each time step, learning the reasoning logic that requires previous entities that were cut off would not be possible. Thus, smoothed discretization is required.

We design filtering to be a two-stage pseudo-continuous process to simulate discrete cut offs (\(\Pi_{\text{move}}, \Pi_{\text{new}}\)), while keeping reference history. The overall memory \(M\) consists of multiple memory banks. A memory bank is a collection or group of entities \(\{m_0, \ldots, m_l\}\), where \(m_0\) denotes the initial and most general bank and \(m_l\) denotes the most relevant bank. Note that \(|l|\) is input dependent and learned. First, entities are moved from \(m_0\) gradually towards \(m_l\) based off of their individual relevance to the question and second, if \(m_l\) becomes too saturated, \(m_{l+1}\) is created. Operations in the external memory allowing for such dynamic restructuring and entity updates are described below. Note that these operations still maintain end to end differentiability.

1. Memory bank creation (\(\Pi_{\text{new}}\)), which creates a new memory bank depending on the current states of entities \(m_i\). If the entropy, or information contained (explained below), of \(m_i\) is too high, \(\Pi_{\text{new}}(m_i)\) will learn to create a new memory bank \(m_{i+1}\) to reduce entropy.

2. Moving entities across banks (\(\Pi_{\text{move}}\)), which determines which entities are relevant to the current question and move such entities to further (higher importance) memory banks.
3. Adding/Updating entities in a bank ($\Pi_{new}$), which adds entities that are not yet encountered to the first memory bank $m_0$ or if the entity is already in $m_0$, the operation updates the entity state.

4. Propagating changes across entities ($\Pi_{prop}$), which updates the entity states in memory banks based on node current states $\Pi_{prop}(M)$ and their semantic relationships. This is to communicate transitive logic.

Both $\Pi_{new}$, $\Pi_{move}$ require a discrete decision (refer to section 4.2.1.), and in particular, for $\Pi_{new}$ we introduce the notion of entropy. That is to say if $m_i$ contains too many nodes (the entropy becomes too high), the memory module will learn to create a new bank $m_{i+1}$ and move nodes to $m_{i+1}$ to reduce entropy. By creating more memory banks, the model spreads out the concentration of information which in turn better discretizes nodes according to relevance.

4. Adaptive Memory Networks

A high-level overview is shown in Figure 2, followed by a mathematical detail of the model’s modules. Our model adopts the encoder-decoder framework with an augmented adaptive memory module. For an overview of the algorithm, refer to Section A.4.

Notation and Problem Statement: Given a story represented by $N$ input sentences (or statements), i.e., ($l_1, \cdots, l_N$), and a question $q$, our goal is to generate an answer $a$. Each sentence $l$ is a sequence of $N$ words, denoted as ($w_1, \cdots, w_{N_l}$), and a question is a sequence of $N_q$ words denoted as ($w_1, \cdots, w_{N_q}$). Throughout the model we refer to entities; these can be interpreted as a 3-tuple of $e_w =$ (word ID $wl$, hidden state $w$, question relevance strength $s$). Scalars, vectors, matrices, and dot products are denoted by lower-case letters, boldface lower-case letters and boldface capital letters, and angled brackets respectively.

4.1. Encoder

The input to the model, starting with the encoder, are story-question input pairs. On a macro level, sentences $l_{1 \cdots N}$ are processed. On a micro level, words $w_{1 \cdots N_s}$ are processed within sentences.

For each $w_i \in l_i$, the encoder maps $w_i$ to a hidden representation and a question relevance strength $\in [0,1]$. The word ID of $w_i$ is passed through a standard embedding layer and then encoded through an accumulation GRU. The accumulation GRU captures the entity states through time by adding the output of each GRU time step to its respective word, stored in a lookup matrix. The initial states of $e_w$ are set to this GRU output. Meanwhile, the question is also embedded and encoded in the same manner sans accumulation.

In the following, the subscripts $i,j$ are used to iterate through the total number of words in a statement and question respectively. $D$ stores the accumulation GRU output, and $w_i$ is a GRU encoding output. The last output of the GRU will be referred to as $w_N$, $w_{N_q}$ for statements and questions.

$$u_i, u_j = EMBED(w_{ij}), EMBED(w_{ij})$$
$$w_i = GRU(u_i, w_{i-1})$$
$$D[i] += w_i$$
$$w_j = GRU(u_j, w_{j-1})$$

To compute the question relevance strength $s \in [0,1]$ for each word, the model uses GRU-like equations. The node strengths are first initialized to Xavier normal and the inputs are the current word states $w^w$, the question state $w^q$, and when applicable, the previous strength. Sentences are processed each time step $t$.

$$z^t = \sigma(U^t w_{1N}^w + W_{iN}^w + X_{iN}^w s_{t-1}^i)$$
$$r^t = 1 - \sigma(U^t (s_{t-1}^i, w_{N_q}^w))$$
$$s^t = \sigma(W^t w_{1N}^w + U^t (r^t \circ s_{t-1}^i))$$
$$s^t = z^t \circ s_{t-1}^i + (1 - z^t) \circ \tilde{s}^t$$

In particular, equation (6) shows where the model learns to lower the strengths of nodes that are not related the question. First, a dot product between the current word states and question state are computed for similarity (high correlation), then it is subtracted from a 1 to obtain the dissimilarity. We refer to these operations as SGRU (Strength GRU) in Algorithm 1.

4.2. Adaptive Memory Module

The adaptive memory module recurrently restructures entities in a question relevant manner so the decoder can then consider fewer entities (namely, the question relevant entities) to generate an answer. The following operations are performed once per sentence.

4.2.1. Memory Bank Controller

As mentioned earlier, discrete decisions are difficult for neural networks to learn so we designed a specific memory bank controller $\Pi_{ctrl}$ for binary decision making. The model takes ideas from the reparameterization trick and uses custom backpropagation to maintain differentiability.

In particular, the adaptive memory module needs to make two discrete decisions on a $\{0,1\}$ basis, one in $\Pi_{new}$ to create a new memory bank and the other in $\Pi_{move}$ to move nodes to a different memory bank. The model uses a scalar $p \in \{0,1\}$ to parameterize a Bernoulli distribution where the realization $H$, is the decision the model makes. However, backpropagation through a random node is intractable,
so the model detaches $H$ from the computation graph and introduces $H$ as a new node. Finally, $H$ is used as a mask to zero out entries in the discrete decision. Meanwhile, $p$ is kept in the computation graph and has a special computed loss (Section 4.4). The operations below will be denoted as $\Pi_{\text{ctrl}}$ and has two instances: one for memory bank creation $\Pi_{\text{new}}$ and one for moving entities across banks $\Pi_{\text{move}}$. In equation 9, depending on what $\Pi_{\text{ctrl}}$ is used for, $q$ is a poly-morphic function and will take on a different operation and will be a different input. Examples of such are given in their respective sections (4.2.2.1, 4.2.2.2).

\[ p = q(\ast) \quad (9) \quad H = \text{Bernoulli}(p) \quad (10) \]

4.2.2. MEMORY BANK OPERATIONS

1. **Memory bank creation $\Pi_{\text{new}}$:** To determine when a new memory bank is created, in other words, if the current memory bank becomes too saturated, the memory bank controller (4.2.1.) will make a discrete decision to create a new memory bank. Here, $q$ (eq 9) is a fully connected layer and the input is the concatenation of all the current memory bank $m_i$’s entity states $[w_0...w_i] \in R^{|w_i|}.\text{new}$. Intuitively, $q$ will learn a continuous decision that is later discretized by eq 10 based on entity states and the number of entities. Note this is only performed for the last memory bank.

\[ \Pi_{\text{new}}([w_0...w_i]) = \begin{cases} \text{M.new()} & \text{if } \Pi_{\text{ctrl}}([w_0...w_i]) \\ \text{pass} & \text{else} \end{cases} \quad (11) \]

2. **Moving entities through memory banks:** Similar to $\Pi_{\text{new}}$, individual entities’ relevance scores are passed into the bank controller to determine $H$ as the input. The relevance score is computed by multiplying an entity state by its respective relevance $\in R^{|w_i|}.r$. Here, $q$ has a slight modification and is the identity function. Note that this operation can only be performed if there is a memory bank to move nodes to, namely if $m_i+1$ exists. Additionally, each bank has a set property where it cannot contain duplicate nodes, but the same node can exist in two different memory banks.

\[ \Pi_{\text{move}}(s_i \ast w_i) = m.add(1 \ast \Pi_{\text{ctrl}}(s_i \ast w_i)) \quad \forall i \in m \quad (12) \]

3. **Adding/Updating entities in a bank:** Recall that entities are initially set to the output of $D$. However, as additional sentences are processed, new entities and their hidden states are observed. In the case of a new entity $e_w$, the entity is added to the first memory bank $m_0$. If the entity already exists in $m_0$, then $e_w$’s corresponding hidden state is updated through a GRU. This procedure is done for all memory banks.

\[ \Pi_{\text{add}}([w_0...w_i]) = \begin{cases} \text{m.add}^i(e_w) & \text{if } e_w \notin m_0 \\ w_{i+1}^t = \text{GRU}(w_N, w_i^t) & \forall m \in M \end{cases} \quad (13) \]

4. **Propagating updates to related entities:** So far, entities exist as a bag of words model and the sentence structure is not maintained. This can make it difficult to solve tasks that require transitive reasoning over multiple entities. To track sentence structure information, we model semantic relationships as a directed graph stored in adjacency matrix A. As sentences are processed word by word, a directed graph is drawn progressively from $w_0...w_i...w_N$. If sentence $l_k$’s path contains nodes already in the current directed graph, $l_k$ will include said nodes in its path. After $l_k$ is added to A, the model propagates the new update hidden state information a for each node $i$ is equal to the sum of the incoming edges’ node hidden states.

Additionally, we add a particular emphasis on $l_k$ to simulate recency. At face value, one propagation step of A will only have a reachability of its immediate neighbor, so to reach all nodes, A is raised to a consecutive power $r$ to reach and update each intermediate
node. $r$ can be either the longest path in $A$ or a set parameter. Again, this is done within a memory bank for all memory banks. For entities that have migrated to another bank, the update for these entities is a no-op but propagation information as per the sentence structure is maintained. A single iteration is shown below:

$$a = (A^r)^T[w_0...w_i] \quad \text{(14)} \quad w^i = \text{GRU}(a, w^{i-1}) \quad \text{(15)}$$

When nodes are transferred across banks, $A$ is still preserved. If intermediate nodes are removed from a path, a transitive closure is drawn if possible.

After these steps are finished at the end of a sentence, namely, the memory unit has reasoned through how large (number of memory banks) the memory should be and which entities are relevant at the current point in the story, all entities are passed through the strength modified GRU (4.1, eq 5-8) to recompute their question relevance (relevance score).

### 4.3. Decode

After all sentences $l_1...N$ are ingested, the decode portion of the network learns to interpret the results from the memory banks. The network iterates through the memory banks using a standard attention mechanism. To force the network to understand the question importance weighting, the model uses an exponential function $d$ to weight important memory banks higher. $C_m$ are the hidden states contained in memory $m$, $s_m$ are the relevance strengths of memory bank $m$, $w_{Nq}$ is the question hidden state, $ps$ is the attention score, $r, h$ are learned weight masks, $g$ are the accumulated states, and $l$ is the final logits prediction. During inference, fewer memory banks are considered.

$$C_m = s_m \cdot [w_0,...w_i] \quad \forall i \in m \quad \text{(16)}$$

$$ps = \text{Softmax}((C_m, w_{Nq})) \quad \text{(17)}$$

$$g = d(C_m, ps) \quad \text{(18)}$$

$$\hat{L} = r(PReLU(h(g) + w_{Nq}) \quad \text{if } m \text{ is last} \quad \text{(19)}$$

### 4.4. Loss

Loss is comprised of two parts, answer loss, which is computed from the given annotations, and secondary loss (from $\Pi_{\text{neu}}, \Pi_{\text{move}}$), which is computed from sentence and story features at each sentence time step $l_0...N$. Answer loss is standard cross entropy at the end of the story after $l_N$ is processed.

$$\mathcal{L}_p(\hat{L}) = \text{CrossEntropy}(\hat{L}, L)$$

After each sentence $l_i$, the node relevance $s_l$ is enforced by computing the expected relevance $\mathbb{E}[s_l]$. $\mathbb{E}[s]$ is determined by nodes that are connected to the answer node $a$ in a directed graph; words that are connected to $a$ are relevant to $a$. They are then weighted with a deterministic function of distance from $a$.

$$\mathcal{L}_v(s) = D_{KL}(s_t||\mathbb{E}[s_t])$$

Additionally, bank creation is kept in check by constraining $p_i$ w.r.t. the expected number of memory banks. The expected number of memory banks can be thought of as a geometric distribution $\sim \text{Geometric}(\tilde{p}_i)$ parameterized by $\tilde{p}_i$, a hyperparameter. Typically, at each sentence step $\tilde{p}$ is raised to the inverse power of the current sentence step to reflect the amount of information ingested. Intuitively, this loss ensures banks are created when a memory bank contains too many nodes. On the other hand, the learned mask $q$ (eq. 9) enables the model to weight certain nodes a higher entropy to prompt bank creation. Through these two dependencies, the model is able to simulate bank creation as a function of the number of nodes and the type of nodes in a given memory bank.

$$\mathcal{L}_b(p_i) = D_{KL}(p_i||\hat{p}^{l/(l-i)})$$

All components combined, the final loss is given in the following equation

$$\mathcal{L}_{\text{total}} = \mathcal{L}_p(\hat{L}) + \sum_{i=1}^{l_n} (\mathcal{L}_v(s) + \mathcal{L}_b(p))$$

### 5. Evaluation

In this section, we evaluate AMN accuracy and inference times on the bAbI dataset (Weston et al., 2015) and extended bAbI tasks dataset. We compare our performance with Entnet (Henaff et al., 2017), which recently achieved state of the art results on the bAbi dataset. For accuracy measurements, we also compare with DMN+ and encoder-decoder methods. Finally we discuss the time trade offs between AMN and current SOTA methods. The portion regarding inference times are not inclusive of story ingestion. We summarize our experiments results as follows:

- We are able to solve all bAbI tasks using AMN. Furthermore, AMN is able to reason important entities and propagate them to the final memory bank allowing for 48% fewer entities examined during inference.
- We construct extended bAbI tasks to evaluate AMN behavior. First, we extend Task 1 for multiple questions in order to gauge performance in a more robust manner. For example, if a reasonable set of questions are asked (where reasonable means that collectively they do not require all entities to answer implying entities can be filtered out), will the model still sufficiently reason
We find that AMN creates memory banks for different tasks and store them in the final memory bank. Furthermore, we also scale the bAbI tasks by adding additional entities and sentences and adding multiple questions for a single story, for Task 1.

### Scaled Task 1

We increase the number of entities to 100 in the task generation system instead of existing 38. We also extend the story length to 90 to ensure new entities are referenced. We find that AMN creates 6 memory banks and the ratio of entities in the final banks versus the overall entities drops to 0.13 given the excess entities that are not referenced in the questions.

#### Multiple questions

We also augment the tasks with multiple questions to understand if AMN can handle when a story has multiple questions associated with it. We extend our model to handle multiple questions at once to limit re-generating the network for every question. To do so, we modify bAbI to generate several questions per story for tasks that do not currently have multiple questions. For single supporting fact (Task 1), the model creates 3 banks and...
Adaptive Memory Networks

| Task                        | AMN | Entnet | DMN+  | MemN2N | EncDec |
|-----------------------------|-----|--------|-------|--------|--------|
| 1 - Single Supporting Fact  | 0.0 | 0.0    | 0.0   | 0.0    | 52.0   |
| 2 - Two Supporting Facts    | 2.1 | 0.1    | 0.3   | 0.3    | 66.1   |
| 3 - Three Supporting Facts  | 4.7 | 4.1    | 1.1   | 2.1    | 71.9   |
| 4 - Two Arg. Relations      | 0.0 | 0.0    | 0.0   | 0.0    | 29.2   |
| 5 - Three Arg. Relations    | 2.7 | 0.3    | 0.5   | 0.8    | 14.3   |
| 6 - Yes/No Questions        | 3.1 | 0.2    | 0.0   | 0.1    | 31.0   |
| 7 - Counting                | 0.0 | 0.0    | 2.4   | 2.0    | 21.8   |
| 8 - Lists/Sets              | 0.0 | 0.5    | 0.0   | 0.9    | 27.6   |
| 9 - Simple Negation         | 1.3 | 0.1    | 0.0   | 0.3    | 36.4   |
| 10 - Indefinite Knowledge   | 1.2 | 0.6    | 0.0   | 0.0    | 36.4   |
| 11 - Basic Coreference      | 2.7 | 0.3    | 0.0   | 0.1    | 31.7   |
| 12 - Conjunction            | 2.2 | 0.0    | 0.0   | 0.0    | 35.0   |
| 13 - Compound Coref.        | 4.6 | 1.3    | 0.0   | 0.0    | 6.80   |
| 14 - Time Reasoning         | 2.1 | 0.0    | 0.2   | 0.1    | 67.2   |
| 15 - Basic Deduction        | 1.8 | 0.0    | 0.0   | 0.0    | 62.2   |
| 16 - Basic Induction        | 4.2 | 0.0    | 0.0   | 0.0    | 54.0   |
| 17 - Positional Reasoning   | 4.3 | 0.5    | 4.2   | 18.6   | 43.1   |
| 18 - Size Reasoning         | 2.0 | 0.3    | 2.1   | 5.3    | 6.60   |
| 19 - Path Finding           | 2.4 | 2.3    | 0.0   | 2.3    | 89.6   |
| 20 - Agents Motivations     | 0.0 | 0.0    | 0.0   | 0.0    | 2.30   |

Table 1. Performance comparison of various models in terms of test error rate (%) and the number of failed tasks on the bAbI dataset. The bold task scores are where AMN can solve the task using only 1 memory bank.

No. of failed tasks (>5%) | 0  | 0  | 5  | 6  | 20 |

Table 2. Memory bank analysis of indicative tasks.

requires 1 bank to successfully pass the task. Furthermore, the ratio of entities required to pass the task only increases by 0.16 for a total of 0.38.

### 6. Conclusion and Future Work

In this paper, we present Adaptive Memory Network that learns to adaptively organize the memory to answer questions with lower inference times. Unlike NTMs which learn to read and write at individual memory locations, Adaptive Memory Network demonstrates a novel design where the learned memory management is coarse-grained that is easier to train. Through our experiments, we demonstrate that AMN can learn to reason, construct, and sort memory banks based on relevance over the question set.

AMN presents a new design paradigm in memory networks. Unlike NTMs, where the network learns where to read/write to fine-grained address information, AMN only learns to write input entities to coarse-grained banks. As a result, AMN is easier to train (e.g. does not require curriculum learning like NTMs) and does not require a separate sparsification mechanism like approximate nearest neighbors for inference efficiency. Apart from saving inference times, AMN can learn to reason which specific entities contribute towards the final answer improving interpretability.

### References

Amazon. Alexa skills kit. In [https://developer.amazon.com/alexa-skills-kit], 2017.
Adaptive Memory Networks

Bengio, Emmanuel, Bacon, Pierre-Luc, Pineau, Joelle, and Precup, Doina. Conditional computation in neural networks for faster models. arXiv preprint arXiv:1511.06297, 2015.

Chandar, Sarath, Ahn, Sungjin, Larochelle, Hugo, Vincent, Pascal, Tesauro, Gerald, and Bengio, Yoshua. Hierarchical memory networks. arXiv preprint arXiv:1605.07427, 2016.

Denton, Emily L, Zaremba, Wojciech, Bruna, Joan, LeCun, Yann, and Fergus, Rob. Exploiting linear structure within convolutional networks for efficient evaluation. In Advances in Neural Information Processing Systems, pp. 1269–1277, 2014.

Eigen, David, Ranzato, Marc’Aurelio, and Sutskever, Ilya. Learning factored representations in a deep mixture of experts. arXiv preprint arXiv:1312.4314, 2013.

Figurnov, Michael, Collins, Maxwell D, Zhu, Yukun, Zhang, Li, Huang, Jonathan, Vetrov, Dmitry, and Salakhutdinov, Ruslan. Spatially adaptive computation time for residual networks. arXiv preprint arXiv:1612.02297, 2016.

Giles, C Lee and Omlin, Christian W. Pruning recurrent neural networks for improved generalization performance. IEEE transactions on neural networks, 5(5):848–851, 1994.

Goodwin, Travis R and Harabagiu, Sandra M. Medical question answering for clinical decision support. In Proceedings of the 25th ACM International on Conference on Information and Knowledge Management, pp. 297–306. ACM, 2016.

Graves, Alex. Adaptive computation time for recurrent neural networks. arXiv preprint arXiv:1603.08983, 2016.

Graves, Alex, Wayne, Greg, and Danihelka, Ivo. Neural turing machines. arXiv preprint arXiv:1410.5401, 2014.

Han, Song, Mao, Huizi, and Dally, William J. Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding. In ICLR, 2016.

Hazy, Thomas E, Frank, Michael J, and OReilly, Randall C. Banishing the homunculus: making working memory work. Neuroscience, 139(1):105–118, 2006.

Henaff, Mikael, Weston, Jason, Szlam, Arthur, Bordes, Antoine, and LeCun, Yann. Tracking the world state with recurrent entity networks. Proceedings of the International Conference on Learning Representations, 2017.

Johnson, Daniel D. Learning graphical state transitions. In ICLR, 2017.

Kingma, Diederik and Ba, Jimmy. Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980, 2014.

Le Cun, Yann, Denker, John S, and Solla, Sara A. Optimal brain damage. In NIPS, 1989.

Li, Yujia, Tarlow, Daniel, Brockschmidt, Marc, and Zemel, Richard. Gated graph sequence neural networks. arXiv preprint arXiv:1511.05493, 2015.

Liu, Lanlan and Deng, Jia. Dynamic deep neural networks: Optimizing accuracy-efficiency trade-offs by selective execution. arXiv preprint arXiv:1701.00299, 2017.

Paszke, Adam, Gross, Sam, and Chintala, Soumith. Pytorch, 2017.

Rae, Jack, Hunt, Jonathan J, Danihelka, Ivo, Harley, Timothy, Senior, Andrew W, Wayne, Gregory, Graves, Alex, and Lillicrap, Tim. Scaling memory-augmented neural networks with sparse reads and writes. In Advances in Neural Information Processing Systems, pp. 3621–3629, 2016.

Rastegari, Mohammad, Ordonez, Vicente, Redmon, Joseph, and Farhadi, Ali. Xnor-net: Imagenet classification using binary convolutional neural networks. In European Conference on Computer Vision, pp. 525–542. Springer, 2016.

See, Abigail, Luong, Minh-Thang, and Manning, Christopher D. Compression of neural machine translation models via pruning. Proceedings of the SIGNLL Conference on Computational Natural Language Learning (CoNLL), 2016.

Shazeer, Noam, Mirhoseini, Azaizia, Maziarz, Krzysztof, Davis, Andy, Le, Quoc, Hinton, Geoffrey, and Dean, Jeff. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. arXiv preprint arXiv:1701.06538, 2017.

Sukhbaatar, Sainbayar, szlam, arthur, Weston, Jason, and Fergus, Rob. End-to-end memory networks. pp. 2440–2448, 2015.

Tulving, Endel et al. Episodic and semantic memory. Organization of memory, 1:381–403, 1972.

Vanhoucke, Vincent, Senior, Andrew, and Mao, Mark Z. Improving the speed of neural networks on cpus. In Proc. Deep Learning and Unsupervised Feature Learning NIPS Workshop, volume 1, pp. 4, 2011.

Watzl, Sebastian. Structuring mind: The nature of attention and how it shapes consciousness. Oxford University Press, 2017.
Adaptive Memory Networks

Weston, Jason, Chopra, Sumit, and Bordes, Antoine. Memory networks. *arXiv preprint arXiv:1410.3916*, 2014.

Weston, Jason, Bordes, Antoine, Chopra, Sumit, Rush, Alexander M, van Merriënboer, Bart, Joulin, Armand, and Mikolov, Tomas. Towards AI-complete question answering: A set of prerequisite toy tasks. *arXiv preprint arXiv:1502.05698*, 2015.

Xiong, Caiming, Merity, Stephen, and Socher, Richard. Dynamic memory networks for visual and textual question answering. In *International Conference on Machine Learning*, pp. 2397–2406, 2016.
## A. Appendix

### A.1. Decode overhead

We compare the computations costs during the decode operation during inference for solving the extended bAbI task. We compute the overheads for AMN, Entnet (Henaff et al., 2017) and GGT-NN. Table 3 gives the decode comparisons between AMN, Entnet and GGT-NN. Here, $|V|$ represents the total number of entities for all networks. GGT-NN can dynamically create nodes and $k$ is hyper parameter the new nodes created for $S$ sentences in input story. $\alpha$ is the percent of entities stored in the final bank w.r.t to the total entities for AMN.

We compare the wall clock execution times for three tasks within bAbI for 1000 examples/task. We compare the wall-clock times for three tasks. We compare the inference times of considering all banks (and entities) versus the just looking at the passing banks as required by AMN. We find that AMN requires fewer banks and as a consequence fewer entities and saves inference times.

### A.2. Propagation Example

In this section, we explain propagation with an example. Figure 3 shows how propagation happens after every time step. The nodes represent entities corresponding to words in a sentence. As sentences are processed word by word, a directed graph is drawn progressively from $w_0 \ldots w_i \ldots w_N$. If sentence $l_k$’s path contains nodes already in the current directed graph, $l_k$ will include said nodes in its path. After $l_k$ is added to $A$, the model propagates the new update hidden state information $a_i$ among all node states using a GRU. $a_i$ for each node $i$ is equal to the sum of the incoming edges’ node hidden states. Additionally, we add a particular emphasis on $l_k$ to simulate recency. At face value, one propagation step of $A$ will only have a reachability of its immediate neighbor, so to reach all nodes, $A$ is raised to a consecutive power $r$ to reach and update each intermediate node. $r$ can be either the longest path in $A$ or a set parameter.

### A.3. Memory bank behavior

In this section, we understand memory bank behavior of AMN. Figure 4 shows the memory banks and the entity creation for a single story example, for some of the tasks from bAbI. Depending upon the task, and distance from the ques-
### Algorithm 1 AMN(S, q, a)

1: $M \leftarrow \emptyset$
2: for sentence $s \in S$ do
3: for word $w \in s$ do
4: $D \leftarrow ENCODE(w, q)$
5: end for
6: $n_m, \leftarrow \text{SGRU}(D)$
7: for memory bank $m \in M$ do
8: $m, \leftarrow \Pi_{au}(m_i, D)$
9: $m, \leftarrow \Pi_{prop}(m_i)$
10: $m_{i+1} \leftarrow \Pi_{move}(m_i, n_{m_i})$
11: $n_{m_i} \leftarrow \text{SGRU}(D, n_{m_i})$
12: if $i = |M|$ and $\Pi_{new}(m_i)$ then
13: $M, p, \leftarrow [M, m_{i+1}]$
14: Repeat 8 to 11 once
15: end if
16: end for
17: end for
18: $\hat{a} \leftarrow \text{DECODE}(M, q)$

### Table 4. Memory bank wall clock times for representative tasks for 1000 examples (time in secs).

| Task                          | Created Banks | Required Banks | Baseline (All banks) | AMN (Required banks) |
|-------------------------------|---------------|----------------|----------------------|----------------------|
| 1 - Single Supporting Fact    | 3             | 1              | 2.15 s               | 0.6 s                |
| 2 - Two Supporting Facts      | 5             | 1              | 15.8 s               | 3.2 s                |
| 7 - Counting                  | 5             | 2              | 21 s                 | 6.0 s                |