Visually Grounded Reasoning about Temporal Concepts in Selective Attention Memory

Anonymous Author(s)
Affiliation
Address
email

Abstract

Visual reasoning in videos requires understanding temporal concepts in addition to the objects and their relations in a given frame. In analogy with human reasoning, we present Selective Attention Memory network (SAMNet), an end-to-end differentiable recurrent model equipped with external memory. SAMNet can perform multi-step reasoning on a frame-by-frame basis, and dynamically control information flow to the memory to store context-relevant representations to answer questions. We tested our model on the COG dataset (a multi-frame visual question answering test), and outperformed the state of the art baseline for hard tasks and in terms of generalization.

1 Introduction

Integration of vision and language in deep neural network models allows the system to learn joint representations of objects, concepts, and relations. Potentially, this approach can address Harnad's "symbol grounding problem" [Har03] and lead to visually grounded language learning.

Starting with the Visual Question Answering (VQA) dataset [AAL+15], a variety of tasks that integrate vision and language have emerged in the past several years [MKK19]. Going beyond classification and object detection, the core emphasis in these tasks is *visual reasoning* that tackles spatial aspects such as comparison of object attributes, counting and other relational questions.

In contrast to VQA, which comes with static image and question pairs, another emerging direction is Video QA [], that provides an additional opportunity to work on *temporal* relations and reasoning. As evident from human cognition, attention and memory are the key competencies required to solve these problems, and unsurprisingly, the AI research is rapidly growing in these areas.

The ability to deal with time can pose a challenge for natural language processing (NLP), e.g. in question answering (QA) and dialog applications. Current NLP solutions, in certain problem settings, work around this challenge by processing the entire text input and reason over it multiple times using attention [VSP+17] or other mechanisms. For example, solutions to the bAbI reasoning task (e.g. Memory Networks [WBC+15]), typically involve processing the supporting facts all at once, which reside in memory and available to provide answers. Similarly, in Visual Dialog [DKG+17] the system keeps the whole history of the dialog in memory. In real-time dialog or video QA, there may not be such an opportunity to have the question and entire visual data all at once in the beginning.

Video reasoning datasets such as SVQA (Synthetic Video Question Answering) [SSCH18] and COG [YGW+18] have limited number of frames (e.g. 4-16 frames), and therefore manageable by the neural net models to process and represent all of the visual information. As an example, according to [SSCH18] the authors extracted visual features from each frame and aggregated features of all clips from one video to form a sequential video representation.

Submitted to 33rd Conference on Neural Information Processing Systems (NeurIPS 2019). Do not distribute.

Contributions. Inspired by human cognitive capacity to selectively pay attention and store salient information in memory, we introduce a new model that can dynamically process video input frame-by-frame, reason over images and remember the salient concepts to answer questions. Our results based on the COG dataset [YGW+18] indicate that the model is capable of: (1) learning the temporal association, i.e., grounding the time-related words with meaning; (2) learning complex, multi-step reasoning that involves grounding of words and visual representations of objects/attributes; (3) selectively control the flow of information to and from the memory to answer questions; and (4) updating the memory only with relevant visual information depending on the temporal context.

2 Selective Attention Memory (SAM) Network

SAM Network (SAMNet for short) is an end-to-end differentiable recurrent model equipped with an external memory (Figure 1). The model makes a single pass over the frames in temporal order, accessing one frame at a time. The memory locations store relevant objects representing contextual information about words in text and visual objects extracted from video. Each location of the memory stores a d-dimensional vector. The memory can be accessed through either content-based addressing, via dot-product attention, or location-based addressing. Using gating mechanisms, correct objects can be retrieved in order to perform multi-step spatio-temporal reasoning over text and video.

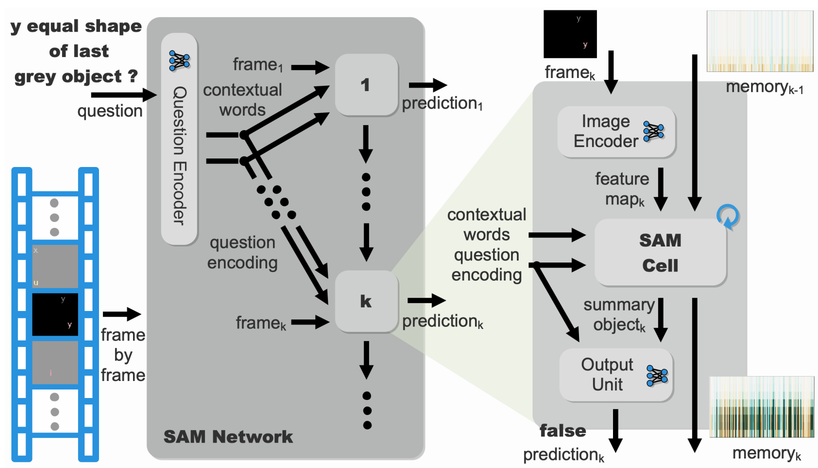


Figure 1: General architecture of SAMNet

The core of SAMNet is a recurrent cell called a SAM Cell (Figure 2). Unrolling a new series of T cells for every frame enables T steps of compositional reasoning. Information flows between frames through the external memory. During the t-th reasoning step, for $t = 1, 2, \ldots, T$, SAM Cell maintains the following information as part of its recurrent state: (a) $c_t \in \mathbb{R}^d$, the control state used to drive the reasoning over objects in the frame and memory; and (b) $so_t \in \mathbb{R}^d$, the summary visual object representing the relevant object for step t. Let

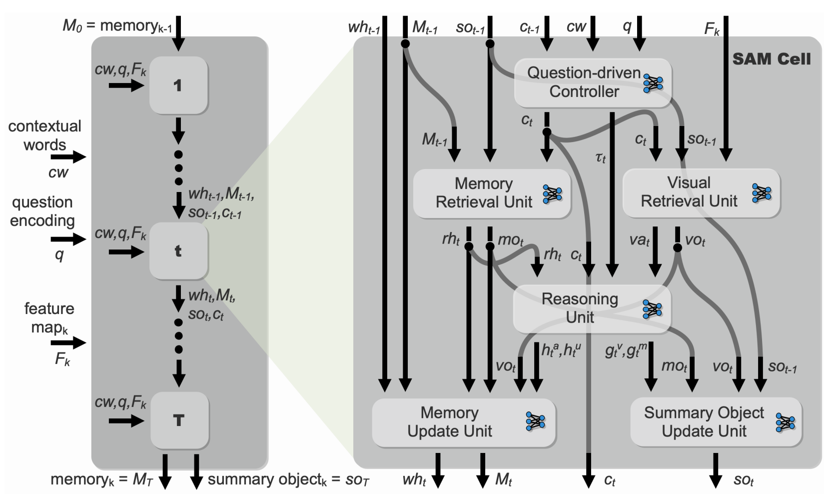


Figure 2: Single reasoning step in SAMCell

$M_t \in \mathbb{R}^{N \times d}$ denote the external memory with N slots at the end of step t. Let $wh_t \in \mathbb{R}^N$ denote an attention vector over the memory locations; in a trained model, wh_t points to the location of first empty slot in memory for adding new objects.

Question-driven Controller. This module drives attention over the question to produce k control states, one per reasoning operation. The control state c_t at step t is then fed to a *temporal classifier*, a two-layer feedforward network with ELU activation used in the hidden layer of d units. The output τ_t of the classifier is intended to represent the different temporal contexts (or lack thereof) associated with the word in focus for that step of reasoning. For the COG dataset we pick 4 classes to capture the terms labeled "last", "latest", "now", and "none".

The visual retrieval unit uses the information generated above to extract a relevant object vo_t from the frame. A similar operation on memory yields the object mo_t. The memory operation is based on an attention mechanism, and resembles content-based addressing on memory. Therefore, we obtain an attention vector over memory addresses that we interpret to be the *read head*, denoted by rh_t. Note that the returned objects may be invalid, e.g., if the current reasoning step focuses on the phrase "last red square", vo_t is invalid even if the current frame contains a red square.

Reasoning Unit. This module is the backbone of SamNet that determines what gating operations need to be performed on the external memory, as well as determining the location of the correct object for reasoning.

To determine whether we have a valid object from frame (and similarly for memory), we execute the following reasoning procedure. First, we take the visual attention vector va_t of dimension L, where L denotes the number of feature vectors for the frame, and compute a simple aggregate $vs_t = \sum_{i=1}^{L} [va_t(i)]^2$. It can be shown mathematically that the more localized the attention vector is, the higher is the aggregate value. We perform a similar computation on the read head rh_t over memory locations. We feed these 2 values along with the temporal class weights τ_t to a 3-layer feedforward classifier with hidden ELU units to extract 4 gating values in $[0, 1]$ modulated for the current reasoning step: (a) g_t^{v}, which determines whether there is a valid visual object; (b) g_t^{m}, which determines whether there is a valid memory object. (c) h_t^{r}, which determines whether the memory should be updated by replacing a previously stored object with a new one; and (d) h_t^{a}, which determines whether a new object should be added to memory. We stress that the network has to learn via training how to correctly implement these behaviors.

Memory Update Unit. We first determine the exact location where an object could be added to the memory using the following equation:

$$w_t = h^{\mathrm{r}} \cdot rh_t + h^{\mathrm{a}} \cdot wh_{t-1}$$

Above, w_t denotes the pseudo-attention vector that represents the "location" where the memory update should happen. The sum of components of w is at most equal to 1; and w_t can even equal 0 whenever neither condition of adding a new object nor replacing an existing object holds true. We then update the memory accordingly as:

$$M_t = M_{t-1} \odot (J - w_t \otimes \mathbf{1}) + w_t \otimes vo_t,$$

where vo_t denotes the object returned by the visual retrieval unit. Here J denotes the all ones matrix, $\odot$ denotes the Hadamard product and $\otimes$ denotes the Kronecker product. Note that the memory is unchanged in the case where $w_t = 0$, i.e., $M_t = M_{t-1}$. We finally update the write head so that it points to the succeeding address if an object was added to memory or otherwise stay the same. Let wh'_{t-1} denote the circular shift to the right of wh_{t-1} which corresponds to the soft version of the head update. Then:

$$wh_t = h^{\mathrm{a}} \cdot wh'_{t-1} + (1 - h^{\mathrm{a}}) \cdot wh_{t-1}$$

Summary Update Unit. This unit updates the (recurrent) summary object to equal the outcome of the t-th reasoning step. We first determine whether the relevant object ro_t should be obtained from memory or the frame according to:

$$ro_t = g_t^{\mathrm{v}} \cdot vo_t + g_t^{\mathrm{m}} \cdot mo_t$$

Note that ro_t is allowed to be a null object (i.e. 0 vector) in case neither of the gates evaluate to true. Finally, so_t is the output of a simple linear layer whose inputs are ro_t and the previous summary object so_{t-1}. This serves to retain additional information that was in so_{t-1}, e.g., if it held the partial result of a complex query with Boolean connectives.

For more details of the various modules, please see the appendix.

3 Experiments

We evaluated SAMNet on the COG dataset [YGW+18]. Our experiments were designed to study SAMNet's performance as well as its generalization abilities in different settings. For this purpose, we used two different variants of the COG dataset: an easy one (Canonical) and a Hard version to explore a wide range of difficulties. The main differences are the number of frames in the input sequence (4 vs. 8) and the maximum number of distractors (i.e., objects not relevant for the answer) per frame (1 vs. 10).

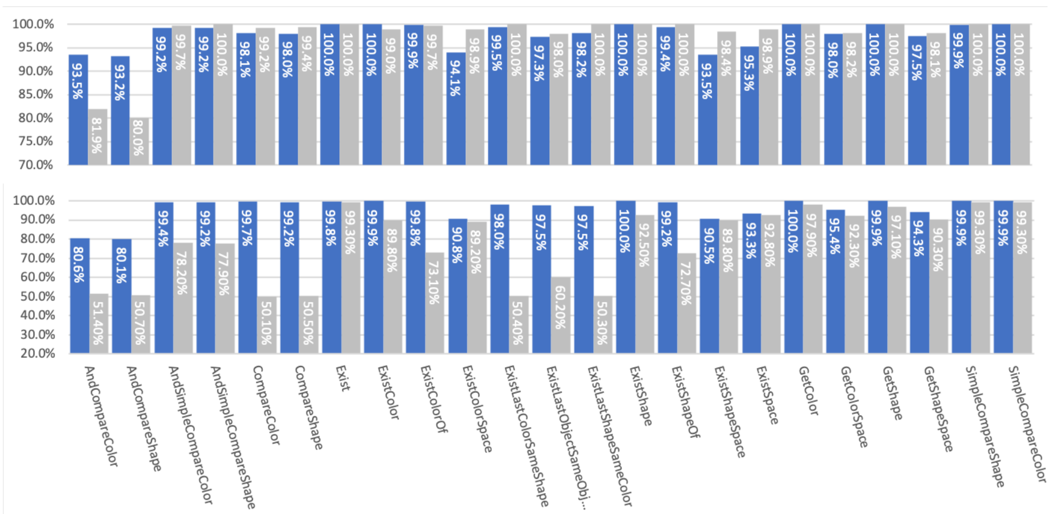


Figure 3: Comparison of test set accuracies of SAMNet (blue) with original results achieved by the COG model (gray) on Canonical (top) and Hard (bottom) variants of the COG dataset.

We have implemented and trained our SAMNet model using MI-Prometheus [KMM+18], a framework based on Pytorch [**?**]. In our experiments, we have focused on 22 classification tasks and compared our results with the baseline model, as presented in Figure 3. For the Canonical variant (top row), we have achieved similar accuracies for the majority of tasks (with the total average accuracy of 98.0% in comparison of 97.6% achieved by the COG model), with significant improvements (around 13 points) for *AndCompare* tasks. As those tasks focus on compositional questions referring to two objects, we hypothesize that our model achieved better accuracy due to the ability to selectively pick and store the relevant objects from the past frames in the memory. Despite there are some tasks in which our model reached slightly lower accuracies, when comparing performances on the Hard variant, it dominates COG baseline on all tasks, with improvements varying from 0.5 to more than 30 points.

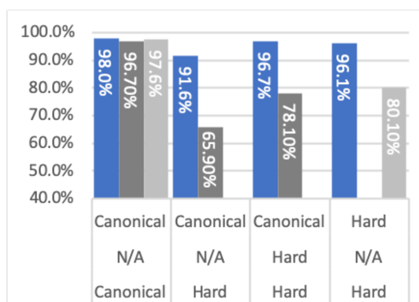


Figure 4: Total accuracies of SAMNet (blue) and COG models (light/dark gray) when testing generalization from Canonical to Hard variants of the dataset.

The goal of the next set of experiments was to test the generalization ability concerning the sequence length and number of distractors. For that purpose, we have compared the accuracies of both models when trained on the Canonical variant and tested on Hard (Figure 4). As the original paper does not include such experiments, we have performed them on our own. The light gray color indicates the original results, whereas dark gray indicates the accuracies of COG models that we have trained (fine-tuning/testing) using the original code provided by the authors. For sanity check, in the first column, we report both the best-achieved score and the score reported in the paper when training and testing on Canonical variant, without any fine-tuning (N/A in the second row from the bottom). In a pure *transfer learning* setup (second column), our model shows enormous generalization ability, reaching 91.6% accuracy on the test set. We have also tested both models in a setup where the model trained on a Canonical variant underwent additional fine-tuning (for a single epoch) on the Hard variant (third column). In this case, the SAMNet model also reached much better performance, and, interestingly, achieved better scores from the model trained and tested exclusively on the Hard variant. In summary, the results clearly indicate that the mechanisms introduced in SAMNet enable it to learn to operate independently of the total number of frames or number of distractions, and allow it to generalize to longer videos and more complex scenes.

References

[AAL+15] Stanislaw Antol, Aishwarya Agrawal, Jiasen Lu, Margaret Mitchell, Dhruv Batra, C. Lawrence Zitnick, and Devi Parikh. VQA: Visual Question Answering. In *International Conference on Computer Vision (ICCV)*, 2015.

[DKG+17] Abhishek Das, Satwik Kottur, Khushi Gupta, Avi Singh, Deshraj Yadav, José MF Moura, Devi Parikh, and Dhruv Batra. Visual dialog. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 326–335, 2017.

[Gan18] Igor Ganichev. Cog implementation. `https://github.com/google/cog`, 2018.

[Har03] Stevan Harnad. Symbol grounding problem. 2003.

[KMM+18] Tomasz Kornuta, Vincent Marois, Ryan L McAvoy, Younes Bouhadjar, Alexis Asseman, Vincent Albouy, TS Jayram, and Ahmet S Ozcan. Accelerating machine learning research with mi-prometheus. In *NeurIPS Workshop on Machine Learning Open Source Software (MLOSS)*, volume 2018, 2018.

[MKK19] Aditya Mogadala, Marimuthu Kalimuthu, and Dietrich Klakow. Trends in integration of vision and language research: A survey of tasks, datasets, and methods. *arXiv preprint arXiv:1907.09358*, 2019.

[SSCH18] Xiaomeng Song, Yucheng Shi, Xin Chen, and Yahong Han. Explore multi-step reasoning in video question answering. In *2018 ACM Multimedia Conference on Multimedia Conference*, pages 239–247. ACM, 2018.

[VSP+17] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, pages 5998–6008, 2017.

[WBC+15] Jason Weston, Antoine Bordes, Sumit Chopra, Alexander M Rush, Bart van Merriënboer, Armand Joulin, and Tomas Mikolov. Towards ai-complete question answering: A set of prerequisite toy tasks. *arXiv preprint arXiv:1502.05698*, 2015.

[YGW+18] Guangyu Robert Yang, Igor Ganichev, Xiao-Jing Wang, Jonathon Shlens, and David Sussillo. A dataset and architecture for visual reasoning with a working memory. In *European Conference on Computer Vision*, pages 729–745. Springer, 2018.

4 Appendix

5 VWM model

Notation. Treat 1D tensors as column vectors and 2D tensors as matrices, where appropriate. We use lower case to represent both 1D and 2D tensors but occasionally use upper case for 2D tensors where matrix operations are involved.

1. Let $\Delta^d = \{(x_0, x_1, \dots, x_d) : x_0 + x_1 + \dots + x_d = 1, x_i \geq 0, i = 0, 1, \dots, d\}$ denote the standard d-simplex.
2. Let $\circ$ denote concatenation of two tensors with identical shape except possibly for their last dimensions d_1 and d_2, respectively, resulting in a tensor with last dimension of $d_1 + d_2$.
3. Let $\odot$ denote element-wise product of two tensors of same shape, i.e., Hadamard product for vectors/matrices.
4. Let $\otimes$ denote tensor product of two tensors, i.e. Kronecker product for vectors/matrices.

6 Basic layers/modules

Linear (Affine) Layer

Inputs: A tensor x with last dimension n.

Parameter: An affine function $\mathcal{G} : \mathbb{R}^n \to \mathbb{R}^m$ with weight and bias parameters.

Output: A tensor y with last dimension m, and remaining dimensions same as that of x, obtained by applying $\mathcal{G}$ to each 1D slice of x along the last dimension.

Attention Module

Inputs:
1. Query: $q \in \mathbb{R}^d$
2. Keys: $K \in \mathbb{R}^{N \times d}$
3. Values: $V \in \mathbb{R}^{N \times d}$. By default $V = K$, unless mentioned explicitly.

Parameter: Weight $w \in \mathbb{R}^d$

Outputs:
1. Content vector: $h = V^\intercal u \in \mathbb{R}^d$
2. Attention vector: $w = \text{softmax}(K(w \odot q)) \in \mathbb{R}^N$

Interaction Module

Inputs:
1. Base object: $b \in \mathbb{R}^d$
2. Feature objects: $f \in \mathbb{R}^{M \times d}$

Parameters:
1. Base object projection linear layer: $\mathcal{G} : \mathbb{R}^d \to \mathbb{R}^d$
2. Feature objects projection linear layer : $\mathcal{K} : \mathbb{R}^{M \times d} \to \mathbb{R}^{M \times d}$
3. Modifier linear layer: $\mathcal{H} : \mathbb{R}^{M \times 2d} \to \mathbb{R}^{M \times d}$

Output: Modified feature objects $f' = \mathcal{H}(\mathcal{K}(f) \odot (\mathbf{1} \otimes \mathcal{G}(b))) \in \mathbb{R}^{M \times d}$

---

7 VWM cell

The VWM recurrent cell is executed for T reasoning steps for every frame in the temporal order. Within a single frame, the cell state at the end of each reasoning step $t = 1, 2, \ldots, T$ is denoted by (c_t, M_t, o_t), where:

1. $c_t \in \mathbb{R}^d$ is the control state;
2. $M_t \in \mathbb{R}^{N \times d}$ is the visual working memory with N slots;
3. $w_t \in \mathbb{R}^N$ is the write head; and
4. $so_t \in \mathbb{R}^d$ is the summary visual object.

The initial state is such that both c_0 and so_0 are initialized to a fixed value at the start of each frame. However M_0 is initialized only once at the start of the first frame and otherwise taken to be the value of M_T at the end of the previous frame.

Question-driven Controller

The Question-driven Controller plays an important role in the reasoning process. It drives the attention over the question and produces the new control states. Each new control state defines a new reasoning operation. The inputs of this unit are the past control state, the question encoding and the contextual words (see Question Encoding Unit). It uses the dot product attention between the contextual words and the combination of the past control states and the question encoding. This attention layer produces the new control state.

This unit also outputs the temporal class weights that will be used in the Reasoning Unit. It gives access to a temporal information for the current words (last, latest, now, none temporal).

Inputs:

1. Reasoning step $t = 1, 2, \ldots, T$
2. Previous control state: $c_{t-1} \in \mathbb{R}^d$
3. Contextual words: $cw \in \mathbb{R}^{L \times d}$
4. Question encoding: $q \in \mathbb{R}^d$

Parameters:

1. Reasoning step-dependent linear layer: $\mathcal{G}_t : \mathbb{R}^d \to \mathbb{R}^d$, depending on s
2. Concatenation linear layer: $\mathcal{H} : \mathbb{R}^{2d} \to \mathbb{R}^d$
3. Attention module $\mathcal{A}$
4. Temporal classifier: $\mathcal{K} : \mathbb{R}^d \to \Delta^3$. A two-layer feedforward network with ELU activation in the hidden layer of d units. The classes for the temporal context are labeled "last", "latest", "now", as well as a fourth class label "none" indicating no temporal context. If $\tau \in \Delta^3$ is the output of the classifier, we denote the components by τ^{last}, τ^{latest}, τ^{now} and τ^{none}.

Outputs:

1. Control state $c_t \in \mathbb{R}^d$
2. Control attention $ca_t \in \mathbb{R}^L$
3. Temporal class weights $\tau_t \in \mathbb{R}^4$

Equations:

1. Modulation: $y = \mathcal{H}\big([c_{t-1}, \mathcal{G}_t(q)]\big)$
2. Control state and attention: $c_t, ca_t = \mathcal{A}(y, cw)$
3. Temporal classification: $\tau_t = \mathcal{K}(c_t)$

Visual Retrieval Unit

The visual retrievial unit is responsible to extract visual information from the current image given a control state coming from the Question-driven Controller. It is first projecting the past summary object and the feature maps together using the interaction module. It is then using the attention module as follow. The query are the control states and the keys and are the feature maps coming from the image encoder. The results of this attention is applied on the modified features maps coming from the interaction module. This unit outputs the extracted object and the visual attention.

Inputs:

1. Control state: $c_t \in \mathbb{R}^d$
2. Previous summary object: $so_{t-1} \in \mathbb{R}^d$
3. Feature map of current frame: $F \in \mathbb{R}^{H \times W \times d}$

Parameters:

1. Interaction module $\mathcal{I}$
2. Attention module $\mathcal{A}$

Outputs:

1. Visual object: $vo_t \in \mathbb{R}^d$
2. Visual attention: $va_t \in \mathbb{R}^{H \times W \times d}$

Equations:

1. Modified feature map: $\hat{F} = \mathcal{I}(so_{t-1}, F)$
2. Visual object and attention: $vo_t, va_t = \mathcal{A}(y, \hat{F}, M_{t-1})$

Note. Appropriate flatten/unflatten operations are performed to match the signature of the modules.

Memory Retrieval Unit

The role of the memory retrieval unit is to read and extract object from memory). As the Visual Retrieval Unit, it uses the combination of the two following submodules. The interaction module blends together the extracted object and the content of the memory. The attention module then extract the corresponding object in memory if present. This unit outputs the extracted object and its corresponding location, we call it the "read head".

Inputs:

1. Control state: $c_t \in \mathbb{R}^d$
2. Previous summary object: $so_{t-1} \in \mathbb{R}^d$
3. Previous VWM $M_{t-1} \in \mathbb{R}^{N \times d}$

Parameters:

1. Interaction module $\mathcal{I}$
2. Attention module $\mathcal{A}$

Outputs:

1. Memory object: $mo_t \in \mathbb{R}^d$
2. Read head: $rh_t \in \mathbb{R}^N$

Equations:

1. Modified VWM: $\hat{M}_t = \mathcal{I}(so_{t-1}, M_{t-1})$
2. Memory object and attention: $mo_t, rh_t = \mathcal{A}(y, \hat{M}_t, M_{t-1})$

Reasoning Unit

Inputs:

1. Control state: $c_t \in \mathbb{R}^d$
2. Visual object: $vo_t \in \mathbb{R}^d$
3. Memory object: $mo_t \in \mathbb{R}^d$
4. Temporal class weights $\tau \in \Delta^3$

Parameters: Validator modules $\mathcal{G}, \mathcal{K} : \mathbb{R}^{2d} \to \mathbb{R}$. Both $\mathcal{G}, \mathcal{K}$ are two-layer networks of $2d$ hidden units, using ELU activation in the hidden layer, and sigmoid in the output layer.

Output: Predicate gates for the current reasoning step

1. Object match predicate gates (i) image: $g_t^{\mathrm{v}} \in [0, 1]$ and (ii) memory: $g_t^{\mathrm{m}} \in [0, 1]$.
2. Memory update predicate gates (i) add: $h_t^{\mathrm{a}} \in [0, 1]$ and (ii) replace: $h_t^{\mathrm{r}} \in [0, 1]$

Equations:

1. $g_t^{\mathrm{v}} \in [0,1]$: It's true if there is a valid visual object. This assumes that the current reasoning step refers to either "‘now" or "latest".
2. $g_t^{\mathrm{m}} \in [0,1]$: It's true if there is a valid memory object. This assumes that the current reasoning step refers to either "last", or alternatively "latest" but there is no matching visual object.
3. h_t^{a}:
4. h_t^{r}:

Memory Update Unit

This unit is meant to update the content of the memory.

Three actions can happen:

- There is no object to be added to memory, the memory remains unchanged
- There is one object that needs to be added to memory, but a similar object is already in memory at a given location. The new object will replace the old object at this location
- There is one object that needs to be added to memory, and it is a new object. It is added at the write head location.

This module also updates the position of the write head. If a new object as been added to the current write head position, the right head shifts right to a new empty slot. If the object has been replaced, the write head doesn't move.

Inputs:
1. Visual object: $vo_t \in \mathbb{R}^d$
2. Memory object: $mo_t \in \mathbb{R}^d$
3. Memory update predicate gates: $h_t^{\mathrm{a}}, h_t^{\mathrm{r}} \in [0,1]$
4. Read head: $rh_t \in \mathbb{R}^N$
5. Previous VWM $M_{t-1} \in \mathbb{R}^{N \times d}$
6. Previous write head: $wh_{t-1} \in \mathbb{R}^N$

Outputs:
1. VWM $M_t \in \mathbb{R}^{N \times d}$
2. Read head: $rh_t \in \mathbb{R}^N$
3. Write head: $wh_t \in \mathbb{R}^N$

Summary Object Update Unit

The Summary Unit is the last unit of the SAMCell. It is responsible to output the new summary object. It first picks which object is relevant between the object extracted from memory and the visual object extracted from the image. Once the relevant object is picked, it is combined with the former summary object through a linear layer to become the new summary object. It is the final step of the SAMCell reasoning cycle.

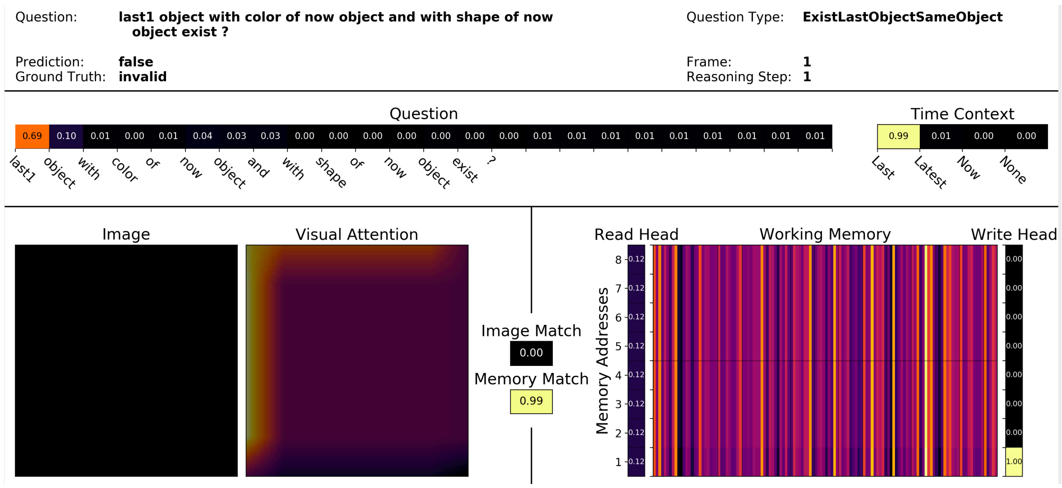
The image encoder, question encoder and output unit are described in the appendix.

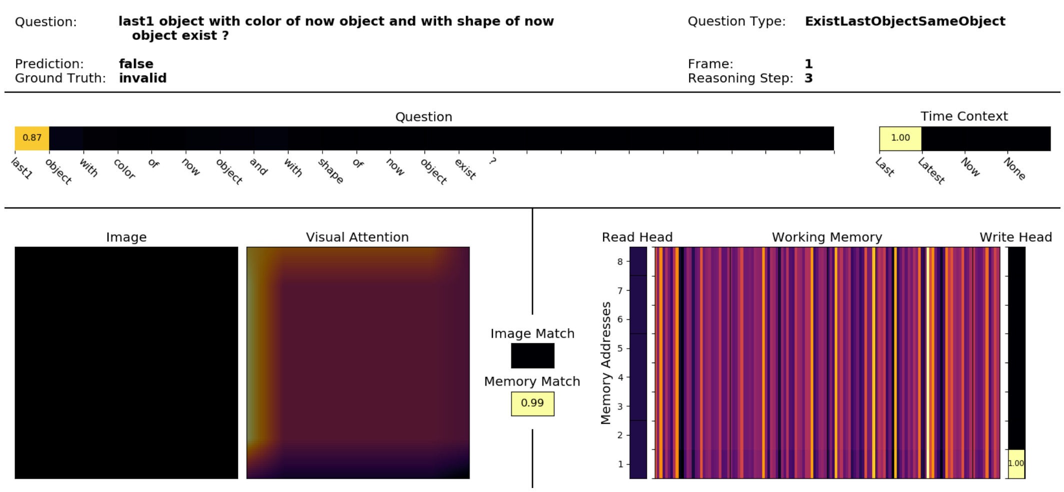
Inputs:
1. Previous summary object: $so_{t-1} \in \mathbb{R}^d$
2. Visual object: $vo_t \in \mathbb{R}^d$
3. Memory object: $mo_t \in \mathbb{R}^d$
4. Object predicate gates: $g_t^{\mathrm{v}}, g_t^{\mathrm{m}} \in [0,1]$

Parameters: Concatenation linear layer $\mathcal{H}$

Output: New summary object: $so_t = \mathcal{H}\big([so_{t-1}, (g_t^{\mathrm{v}} * vo_t + g_t^{\mathrm{m}} * mo_t)]\big) \in \mathbb{R}^d$

Image Encoder





Inputs: Images

Output: Features maps: F_k

The Image Encoder unit is responsible to preprocess the sequence of images one at the time. It is a 4 layer convolutional neural network. Every layer is composed of the following sequence of operations:

1. 2D convolution ($stride = 1$)
2. 2D max pooling
3. 2D batch normalization
4. Relu (only for the first 3 layers)

The first layer has 3 inputs channels and 32 output channels. Layer 2 has 32 and 64. Layer 3 has 64 and 64. Finally, layer 4 has 64 inputs channels and 128 output channels . The resulting tensor (feature maps) are of dimensions [batch size, $H * W = 7 * 7$, dim=128].

Question Encoder

Inputs: 1. question string $question$

2. question lengths $question_l$

Output: 1. contextual word embeddingg cw

2. question encoding q

The strings questions are first embedded ($torch.NN.Embeddings$) and then passed through a d-dimensional biLSTM (d=128). The final hidden state of the biLSTM becomes the question encoding representation q. Whereas the biLSTM outputs are projected via a linear layer to become the final words encodings named contextual words cw.

Output Unit

Inputs: 1. question encoding q

2. summary object so_t

Output: prediction

The output unit predicts the final answer to the question. It is a 2-linear-layer-ELU classifier that produces a distribution over the possible candidates. It is based on two inputs, the final summary object so_t and the question encoding q that are concatenated together.

8 Training and Implementation Details

8.1 Training and testing Methodology

SAMNet is implemented on IBM's Mi-Prometheus [KMM+18] framework based on Pytorch. We trained all our models using NVIDIA's GeForce GTX TITAN X GPUs. SAMNet was trained using 8 reasoning steps and a hidden state size of 128. The external memory has 128-bit slots for all experiments. We trained our model until convergence but we also have set a training time limit of 80 hours.

We compared our model to the original COG model [YGW+18] using their implementation (https://github.com/google/cog) and scores provided by the authors through personal communications. We used the same training parameters detailed in the original paper and reproduced their results. For the generalization experiments from canonical to hard, we used the verified model and obtained new results that were not reported in the reference paper. In Table 1 COG section shows 4 columns divided into two parts: "paper" and "ours" which distinguish between the results reported in the paper vs. our own experiments.

Our experiments focused on the 22 classification tasks provided by the COG dataset. First we evaluated SAMNet's performance on the canonical setting and compared it with the COG Model. As shown in Table 1 we could achieve a small improvement in accuracy, from 97.6% for the COG model to 98% for SAMNet. Next we focused on the hard setting of the dataset which increases the number of distractors from 1 to 10 and the number of frames from 4 to 8.

The first approach was to train a model on the hard training set, and test it on the hard test set. This is the same approach used by the COG paper [YGW+18] to evaluate performance on the hard dataset. We achieve a test accuracy of 91.9 % which represents a 12% improvement from the COG model score (see Table 1).

The second approach was to see if the models can generalize from the easy to the hard setting. For this experiment, we trained a model on the canonical dataset, and directly tested on the hard dataset. This experiment highlighted the most significant difference between SAMNet and the baseline COG model.

Finally we trained a model on the canonical data set, fine-tuned it on the hard data set using only 25k iterations, and tested on the hard dataset. Thanks to fine-tuning, we can observe a significant improvement from 91.6% to 96.5% test accuracy which represents the state of the art accuracy for the hard setting (classification tasks). After a short fine-tuning process, the transferred model could generalize well to harder tasks and even surpass the accuracy obtained in the first approach. We note that the third approach is also twice faster than the first one, and it is more effective in terms of accuracy.

Table 1: COG test set accuracies for SAMNet & COG models. Below 'paper' denotes results from [YGW+18] while 'code' denotes results of our experiments using their implementation [Gan18]

Model	SAMNet				COG			
					paper	ours	ours	paper
Trained on	canonical	canonical	canonical	hard	canonical	canonical	canonical	hard
Fine tuned on	-	-	hard	-	-	-	hard	-
Tested on	canonical	hard	hard	hard	canonical	hard	hard	hard
Overall accuracy	98.0	91.6	96.5	96.1	97.6	65.9	78.1	80.1
AndCompareColor	93.5	82.7	89.2	80.6	81.9	57.1	60.7	51.4
AndCompareShape	93.2	83.7	89.7	80.1	80.0	53.1	50.3	50.7
AndSimpleCompareColor	99.2	85.3	97.6	99.4	99.7	53.4	77.1	78.2
AndSimpleCompareShape	99.2	85.8	97.6	99.2	100.0	56.7	79.3	77.9
CompareColor	98.1	89.3	95.9	99.7	99.2	56.1	67.9	50.1
CompareShape	98.0	89.7	95.9	99.2	99.4	66.8	65.4	50.5
Exist	100.0	99.7	99.8	99.8	100.0	63.5	96.1	99.3
ExistColor	100.0	99.6	99.9	99.9	99.0	70.9	99	89.8
ExistColorOf	99.9	95.5	99.7	99.8	99.7	51.5	76.1	73.1
ExistColorSpace	94.1	88.8	91.0	90.8	98.9	72.8	77.3	89.2
ExistLastColorSameShape	99.5	99.4	99.4	98.0	100.0	65.0	62.5	50.4
ExistLastObjectSameObject	97.3	97.5	97.7	97.5	98.0	77.5	61.7	60.2
ExistLastShapeSameColor	98.2	98.5	98.8	97.5	100.0	87.8	60.4	50.3
ExistShape	100.0	99.5	100.0	100.0	100.0	77.1	98.2	92.5
ExistShapeOf	99.4	95.9	99.2	99.2	100.0	52.7	74.7	72.70
ExistShapeSpace	93.4	87.5	91.1	90.5	97.7	70	89.8	89.80
ExistSpace	95.3	89.7	93.2	93.3	98.9	71.1	88.1	92.8
GetColor	100.0	95.8	99.9	100.0	100.0	71.4	83.1	97.9
GetColorSpace	98.0	90.0	95.0	95.4	98.2	71.8	73.	92.3
GetShape	100.0	97.3	99.9	99.9	100.0	83.5	89.2	97.1
GetShapeSpace	97.5	89.4	93.9	94.3	98.1	78.7	77.3	90.3
SimpleCompareShape	99.9	91.4	99.7	99.9	100.0	67.7	96.7	99.3
SimpleCompareColor	100.0	91.6	99.80	99.9	100.0	64.2	90.4	99.3

Table 2: COG Dataset parameters for the canonical setting and the hard setting

Dataset	number of frames	maximum memory duration	number of distractors	size of training set	size of validation/test set
Canonical setting	4	3	1	10000320	500016
Hard setting	8	7	10	10000320	500016

A more granular analysis of accuracy per task shows a major improvement for the two hardest tasks, AndCompareShape and AndCompareColor. Those two tasks represent a higher level of difficulty due to the number of objects to be remembered in order to answer the question correctly. As we can see in Table 2 we could achieve a 12% improvement for the canonical data set and almost a 40% improvement for the hard dataset. The large improvement in these memory-intensive tasks indicate that the SAMNet's external memory plays a crucial role in our results. The training and implementation details are in appendix.