arXiv:1703.09831v1 [cs.CL] 28 Mar 2017

A Deep Compositional Framework for Human-like Language Acquisition in Virtual Environment

Haonan Yu, Haichao Zhang, and Wei Xu
Baidu Research - Institue of Deep Learning
Sunnyvale, CA 94089
`{haonanyu,zhanghaichao,xuwei06}@baidu.com`

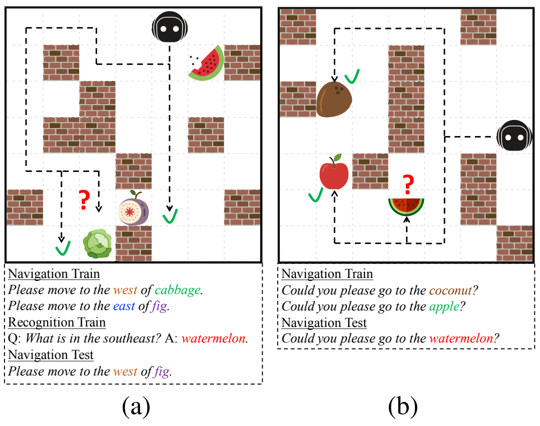
Abstract

We tackle a task where an *agent* learns to navigate in a 2D maze-like environment called XWORLD. In each session, the agent perceives a sequence of raw-pixel frames, a natural language command issued by a *teacher*, and a set of rewards. The agent learns the teacher's language from scratch in a grounded and compositional manner, such that after training it is able to correctly execute *zero-shot* commands: 1) the combination of words in the command never appeared before, and/or 2) the command contains *new object concepts* that are learned from another task but never learned from navigation. Our deep framework for the agent is trained end to end: it learns simultaneously the visual representations of the environment, the syntax and semantics of the language, and the action module that outputs actions. The zero-shot learning capability of our framework results from its compositionality and modularity with parameter tying. We visualize the intermediate outputs of the framework, demonstrating that the agent truly understands how to solve the problem. We believe that our results provide some preliminary insights on how to train an agent with similar abilities in a 3D environment.

1 Introduction

The development of a sophisticated language system is a very crucial part of achieving human-level intelligence for a machine. Language semantics, when grounded in perception experience, can encode knowledge about perceiving the world. This knowledge is transferred from task to task, which empowers the machine with generalization ability. It is argued that a machine has to go through physical experience in order to learn human-level semantics [Kiela et al., 2016], *i.e.*, a process of *human-like* language acquisition. However, current machine learning techniques do not have a reasonably fast learning rate to make this happen. Thus we choose to model this problem in a virtual environment, as the first step towards training a physical intelligent machine.

Human generalize surprisingly well when learning new concepts and skills through natural language instructions. We are able to apply an existing skill to newly acquired concepts with


Navigation Train
Please move to the west of cabbage.
Please move to the east of fig.
Recognition Train
Q: *What is in the southeast?* A: *watermelon*.
Navigation Test
Please move to the west of fig.

Navigation Train
Could you please go to the coconut?
Could you please go to the apple?
Navigation Test
Could you please go to the watermelon?

Figure 1: Illustration of our XWORLD environment and the zero-shot navigation tasks. (a) Test command contains an unseen word combination. (b) Test command contains completely new object concepts that are learned from the recognition task in some previous sessions (a).

little difficulty. For example, a person who has learned how to execute the command "*cut X with knife*" when *X* equals to *apple*, will do correctly when *X* is something else he knows, *e.g.*, *pear* or *orange*, even though he may have never been asked to cut anything other than apple before.

This paper describes a framework that demonstrates the *zero-shot* learning ability of an agent in a specific task, namely, learning to navigate in a 2D maze-like environment called XWORLD (Figure 1). We are interested in solving a similar task that is faced by a baby who is learning to walk and navigate, at the stage of learning his parents' language. The parents might give some simple navigation command consisting of only two or three words in the beginning, and gradually increase the complexity of the command as time goes by. Meanwhile, the parents might teach the baby the language in some other task such as object recognition. After the baby understands the language and masters the navigation skill, he could immediately navigate to a new concept that is learned from object recognition but never appeared in the navigation command before.

We train our baby *agent* across many learning sessions in XWORLD. In each session, the agent perceives the environment through a sequence of raw-pixel images, a natural language command issued by a *teacher*, and a set of rewards. The agent also occasionally receives the teacher's questions on object recognition whenever certain conditions are triggered. By exploring the environment, the agent learns simultaneously the visual representations of the environment, the syntax and semantics of the language, and how to navigate itself in the environment. The whole framework employed by the agent is trained end to end from scratch by gradient descent. We test the agent under three different command conditions, two of which require that the agent generalizes to interpret unseen commands and words, and that the framework architecture is modular so that other modules such as visual perception and action will still work properly under such circumstance. Our experiments show that the agent performs equally well ($\sim 90\%$ average success rate) in all conditions. Moreover, several baselines that simply learn a joint embedding for image and language yield poor results.

In summary, our main contributions are:

- ◦ A new navigation task that integrates both vision and language for deep reinforcement learning (RL). Moreover, the language is *not* pre-parsed [Sukhbaatar et al., 2016] or -linked [Mikolov et al., 2015, Sukhbaatar et al., 2016] to the environment. Instead, the agent has to learn everything from scratch and ground the language in vision.
- ◦ Multi-task transfer learning of language for speeding up RL. Language acquisition in an auxiliary task helps the agent understand the navigation command much faster, and thus master the navigation skill much faster.
- ◦ The zero-shot learning ability by leveraging the compositionality of both the language and the model architecture. We believe that this ability is a crucial component of human-level intelligence.

2 Related Work

Our work is inspired by the research of multiple disciplines. Our XWORLD is similar to the MazeBase environment [Sukhbaatar et al., 2016] in that both are 2D rectangular grid world. One big difference is that their quasi-natural language is already parsed and linked to the environment. They put more focus on reasoning and planning but not language acquisition. On the contrary, we emphasize on how to ground the language in vision and generalize the ability of interpreting the language. There are several challenging 3D environments for RL such as Kempka et al. [2016] and Jaderberg et al. [2017]. The visual perception problems posed by them are much more difficult than ours. However, these environments do not require language understanding. Our agent needs to learn to interpret different goals from different natural language commands.

Our setting of language learning shares some similar ideas of the AI roadmap proposed by Mikolov et al. [2015]. Like theirs, we also have a teacher in the environment that assigns tasks and rewards to the agent. The teacher provides additional help for language learning by asking questions to the agent in an object recognition task. Unlike their proposal of entirely using linguistic channels, our tasks involve multiple modalities and are more close to human experience.

The importance of compositionality and modularity of a learning framework has been discussed at length in cognitive science by Lake et al. [2016]. The compositionality of our framework is inspired by the ideas in Neural Programmer [Neelakantan et al., 2016] and Neural Module Networks [Andreas et al., 2016a,b]. Neural Programmer is trained with backpropagation by employing soft operations on databases. Neural Module Networks assemble several primitive modules according to questions in Visual Question Answering (VQA). It depends on an external parser to convert each sentence to one or several candidate parse trees and thus cannot be trained end to end. We adapt their primitive modules to our framework with differentiable computation units to enable gradient calculation.

The auxiliary recognition task is essentially image VQA [Gao et al., 2015, Ren et al., 2015, Lu et al., 2016, Andreas et al., 2016a,b, Teney and Hengel, 2016, Yang et al., 2016]. The navigation task can also be viewed as a VQA problem if the actions are treated as answer labels. Moreover, it is a zero-shot VQA problem (*i.e.*, test questions containing unseen concepts) which has not been well addressed yet.

Our language acquisition problem is closely related to some recent work on grounding language in images and videos [Yu and Siskind, 2013, Rohrbach et al., 2016, Gao et al., 2016]. The navigation task is also relevant to robotics navigation under natural language command [Chen and Mooney, 2011, Barrett et al., 2015]. However, they either assume annotated navigation paths in the training data or do not ground language in vision. As XWORLD is a virtual environment, we currently do not address mechanics problems encountered by a physical robot, but focus on its mental model building.

3 XWORLD Environment

We first briefly describe the XWORLD environment. More details are in Appendix 8.3. XWORLD is a 2D grid world (Figure 1). An agent interacts with the environment over a number of time steps T, with four actions: `up`, `down`, `left`, and `right`. It does so for many sessions. At the beginning of each session, a teacher starts a timer and issues a natural language command asking the agent to reach a location referred to by objects in the environment. There might be other objects as distractors. Thus the agent needs to differentiate and navigate to the right location. It perceives the entire environment through RGB pixels with an egocentric view (Figure 2c). If the agent correctly executes the command before running out of time, it gets a positive reward R^+. Whenever it hits a wall or steps on an object that is not the target, it gets a negative reward R_w^- or R_o^-, respectively. The agent also receives a small negative reward R_t^- at every step as a punishment for wandering around. After each session, both the environment and the agent are reset randomly.

Some example commands are (the parentheses contain environment configurations that are *withheld* from the agent, same below):

- *Please navigate to the apple.* (There is an apple, a banana, an orange, and a grape.)
- *Can you move to the grid between the apple and the banana?* (There is an apple and a banana. The apple and the banana are separated by one empty grid.)
- *Could you please go to the red apple?* (There is a green apple, a red apple, and a red cherry.)

The difficulty of this navigation task is that, at the very beginning the agent knows nothing about the language: every word appears equally meaningless. After trials and errors, it has to figure out the language syntax and semantics in order to correctly execute the command.

We add an auxiliary recognition task to help the agent learn the language. While it is exploring the environment, the teacher asks object-related questions whenever certain conditions are triggered (all conditions are listed in Appendix 8.3). The answers are always single words and provided by the teacher for supervision. Some example QA pairs are:

- Q:*What is the object in the north?* A:*Banana.* (The agent is by the south of a banana, by the north of an apple, and by the west of a cucumber.)
- Q:*Where is the banana?* A:*North.* (The agent is by the south of a banana and the east of an apple.)
- Q:*What is the color of the object in the west of the apple?* A:*Yellow.* (An apple has a banana on its west and a cucumber on its east.)

We expect the agent to learn the language much faster given this auxiliary task.

4 Compositional Framework for Zero-shot Navigation

Our framework contains four major modules: a language module, a recognition module, a visual perception module, and an action module. The design of the framework is mostly driven by the need of navigating to new objects (Figure 1b) that never appear in the commands (only appearing as

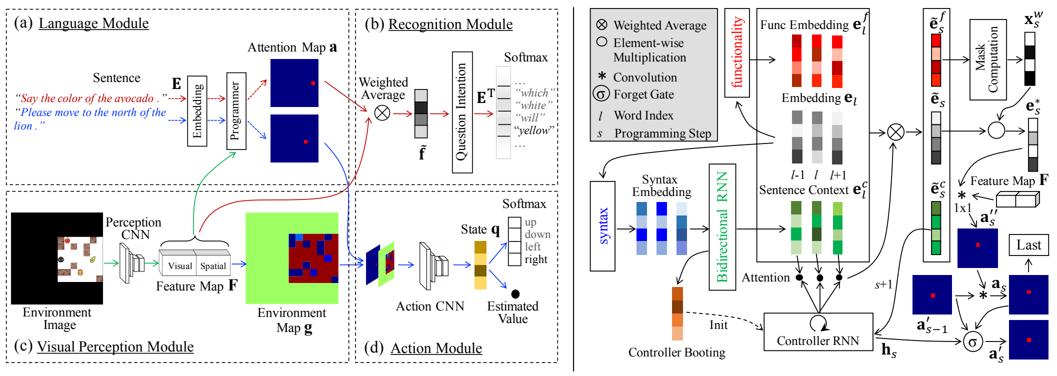


Figure 2: Left: An overview of our framework. The inputs are an environment image and a sentence (either a navigation command or a question). The output is either a navigation action or an answer to the question, respectively. The red and blue lines in (a) indicate different tasks going through exactly the same process. Right: The pipeline of the programmer of the language module. The input is a sequence of word embeddings. The output is the attention map at the final step.

answers in the recognition module but never in the language module Figure 2a). We see three crucial properties for the framework:

- The language module must be compositional. It needs to process a sentence while preserving the (major) sentence structure. One example would be a parser that outputs a parse tree.
- Inductive bias [Lake et al., 2016] must be learned from existing sentences. The language module knows how to parse a sentence with a known structure if a word position is filled with a completely new word.
- Language grounding (Figure 2a) and recognition (Figure 2b) have to be reduced to (approximately) the *same* problem. This ensures that language grounding trained on $n-1$ words still works properly on the nth word which is trained from the recognition.

There is no existing framework for image captioning or VQA exhibits all such properties. Specifically, it is difficult, if not impossible, for a simple (gated) Recurrent Neural Network (RNN) to satisfy the first two properties when a sentence has a rich structure. We show that word attention [Bahdanau et al., 2015], visual attention, and external memory [Graves et al., 2014] are the keys.

Recognition We assume a feature map $\mathbf{F} \in \mathbb{R}^{D\times N}$ with D channels and a spatial dimension of N. Consider classifying the feature $\mathbf{f}_n \in \mathbb{R}^D$ at location n on the map, into m out of M words with probability $P(m|\mathbf{f}_n) = Softmax_M(\mathbf{s}_m^\intercal \mathbf{f}_n)$, where $\mathbf{s}_m^\intercal$ is the mth row of the Softmax matrix $\mathbf{S} \in \mathbb{R}^{M\times D}$. Also consider grounding (*i.e.*, computing attention) the mth word $\mathbf{e}_m \in \mathbb{R}^D$ in $\mathbf{F}$ at location n by $A(n|\mathbf{e}_m) = Softmax_N(\mathbf{f}_n^\intercal \mathbf{e}_m)$ which produces a sum-to-one attention map. Our observation is the similarity between $P(m|\mathbf{f}_n)$ and $A(n|\mathbf{e}_m)$: both reply on dot product computation ($\mathbf{s}^\intercal\mathbf{f}$ or $\mathbf{f}^\intercal\mathbf{e}$). Inspired by the transposed weight sharing scheme [Mao et al., 2015], we set $\mathbf{S} = \mathbf{E}^\intercal$ where $\mathbf{E}$ is the word embedding table. As a result, $\mathbf{s}^\intercal\mathbf{f}$ and $\mathbf{f}^\intercal\mathbf{e}$ now compute the same quantity, namely, the similarity between word embedding and feature. Therefore, both $\max\limits_{\mathbf{E},\mathbf{F}} P(m|\mathbf{f}_n)$ and $\max\limits_{\mathbf{E},\mathbf{F}} A(n|\mathbf{e}_m)$ are optimized towards

$$\mathbf{e}_m \propto \begin{cases} \mathbf{f}_n & \mathcal{C}(\mathbf{f}_n) = m \\ -\mathbf{f}_n & \mathcal{C}(\mathbf{f}_n) \neq m \end{cases},$$

where $\mathcal{C}$ denotes the correct label of the feature.

Our recognition module assumes an attention map $\mathbf{a} \in \mathbb{R}^N$ from the language module and a feature map $\mathbf{F} \in \mathbb{R}^{D\times N}$ from the visual perception module. The attention map highlights the interesting region according to the teacher's question. We extract a feature by weighted averaging the feature map with the attention map: $\tilde{\mathbf{f}} = \mathbf{F}\mathbf{a}$. When this feature is classified, the Softmax could potentially output any word such as object name, color, or location. Thus the feature map $\mathbf{F}$ has to contain both visual and spatial information. Suppose the teacher asks two questions that result in the same attention map: "*Where is the apple?*" and "*What is the object on the west?*". In both cases, the attention map highlights the same region on the agent's west. As a result, the extracted feature $\tilde{\mathbf{f}}$ will be the same for both questions, although the answers should be different (one is *west* and the other is *apple*). To resolve this conflict, the question intention has to be understood. The agent needs to understand that the first question asks the location while the second question asks the object name. We use a simple gated RNN [Cho et al., 2014] to encode and summarize a question to generate an embedding mask $\mathbf{x}^q \in [0,1]^D$. Then the mask and the feature is element-wise multiplied to yield a masked feature [1] $\mathbf{f}^* = \tilde{\mathbf{f}} \circ \mathbf{x}^q$ which is used for the final classification (Figure 2b). The details of generating the embedding mask are shown in Figure 5a Appendix 8.5.

Visual Perception The visual perception module receives an environment image and outputs a feature map $\mathbf{F} \in \mathbb{R}^{D\times N}$ and an environment map $\mathbf{g} \in \mathbb{R}^N$ (Figure 2c). We feed the environment image into a Convolutional Neural Network (CNN) whose output $\mathbf{F}_v \in \mathbb{R}^{D'\times N}$ ($D' < D$) has the same spatial dimension with the number of grids N in the original image, where each pixel of $\mathbf{F}_v$ corresponds to a grid. One can instead segment object boundaries and average features inside each object. We leave such to future work. We further convolve $\mathbf{F}_v$ with a 1×1 filter to get the environment map $\mathbf{g}$. This map is input to the action module together with the navigation attention map $\mathbf{a}$. We expect the environment map to encode the locations of all objects and walls so that the agent knows which locations to avoid. The feature map $\mathbf{F}_v$ contains only visual information. We stack it onto another feature map $\mathbf{F}_s \in \mathbb{R}^{(D-D')\times N}$ that represents spatial information in the egocentric view ($\mathbf{F} = [\mathbf{F}_v, \mathbf{F}_s]$). This spatial feature map is a parameter matrix that needs to be learned by the agent.

Language A sentence is compositional by nature. Understanding its structure is crucial for a correct interpretation. For example, the question "*What is in the west of the apple?*" implies operations `Describe`(**F**,`Transform`[*west*](`Find`[*apple*](**F**))). If the word *west* is replaced with *east*, the interpretation result would be much different. It is difficult for a simple (gated) RNN to output a compact embedding that encodes all such structural information, as the embedding is largely determined by the majority of the sentence. Thus our language module should also be compositional, namely, it knows how to (implicitly) assemble itself differently according to different input sentences.

Commands and questions are processed by the same language module (Figure 2a). The result is an attention map $\mathbf{a} \in \mathbb{R}^N$ that highlights interesting regions. For navigation, the regions indicate the target locations. For recognition, they indicate which features on the feature map to be recognized. We treat each sentence as a *program command*. The core of our language module is a *differentiable programmer* that executes a command in several steps. At each step the programmer has access to the feature map **F** and the intermediate result in the previous step. By attending to different words, the programmer is able to assemble a latent network across steps. At the final step the programmer outputs an attention map **a** (Figure 2 Right). Below we describe the details of the programmer.

A sentence of length L is first converted to a sequence of word embeddings $\mathbf{e}_l$ by looking up the embedding table $\mathbf{E} \in \mathbb{R}^{D\times M}$. The embeddings are then projected[2] to *syntax* embeddings and *functionality* embeddings $\mathbf{e}_l^f$ (Figure 4 Appendix 8.5). The syntax embeddings are fed into a Bidirectional RNN [Schuster and Paliwal, 1997] to obtain *sentence context* vectors $\mathbf{e}_l^c$. The last forward state and the first backward state are concatenated and projected to a booting vector (Figure 6 Appendix 8.5). The programmer controller, designed as a gated RNN [Cho et al., 2014], is initialized with this booting vector. Given a fixed number of programming steps S, at each step s the controller computes the attention for each word l from its sentence context vector $\mathbf{e}_l^c$:

$$\begin{aligned} c_{s,l} &= Softmax_L\left(CosSim\left(\mathbf{h}_s, g(\mathbf{W}\mathbf{e}_l^c + \mathbf{b})\right)\right), \\ CosSim(\mathbf{z},\mathbf{z}') &= \frac{\mathbf{z}^\intercal\mathbf{z}'}{\|\mathbf{z}\|\|\mathbf{z}'\|}, \end{aligned}$$

where $\mathbf{W}$ and $\mathbf{b}$ are projection parameters, $\mathbf{h}_s$ is the RNN state, and g is the activation function. Then the word embeddings, the context vectors, and the functionality embeddings are all weighted averaged by the attention: $\tilde{\mathbf{e}}_s = \sum_l c_{s,l}\cdot\mathbf{e}_l, \tilde{\mathbf{e}}_s^c = \sum_l c_{s,l}\cdot\mathbf{e}_l^c, \tilde{\mathbf{e}}_s^f = \sum_l c_{s,l}\cdot\mathbf{e}_l^f$. The averaged context vector $\tilde{\mathbf{e}}_s^c$ is fed back to the controller to tell it how to update its hidden state.

The averaged word embedding $\tilde{\mathbf{e}}_s$ represents what to be grounded at step s. Similar to recognition, we generate an embedding mask $\mathbf{x}_s^w \in [0,1]^D$ as a projection of the averaged functionality embedding $\tilde{\mathbf{e}}_s^f$ (Figure 5b Appendix 8.5), for computing the masked embedding $\mathbf{e}_s^* = \tilde{\mathbf{e}}_s \circ \mathbf{x}_s^w$. The masked embedding is convolved as a 1×1 filter with the feature map to obtain an egocentric attention

[1]The embedding mask should be really applied to the embedding instead of the feature. However, because the classification relies on computing dot products of the two, we have $\mathbf{f}^\intercal(\mathbf{e}\circ\mathbf{x}) = (\mathbf{f}\circ\mathbf{x})^\intercal\mathbf{e}$.

[2]We use "projection" to denote the process of going through one or more fully-connected (FC) layers.

map $\mathbf{a}''_s = Softmax(\mathbf{F} * \mathbf{e}^*_s)$. Assume that in the previous step we have cached an attention map $\mathbf{a}'_{s-1}$ which is essentially a one-slot external memory. The programmer updates it by convolution $\mathbf{a}_s = \mathbf{a}''_s * \mathbf{a}'_{s-1}$. This convolution is used to approximate the 2D translation of spatial attention, given that the attention map of a location word is egocentric and needs to be imposed on object attention. Then the programmer caches a new attention map by a convex combination of the previously cached one and the current output: $\mathbf{a}'_s = (1-\sigma)\mathbf{a}'_{s-1} + \sigma\mathbf{a}_s$, where the Sigmoid gate σ is a scalar projection of the current controller state $\mathbf{h}_s$ (Figure 2 Right). At the beginning of programming, we set $\mathbf{a}'_0 = \mathbf{i}$ where $\mathbf{i}$ is a map of which only the center pixel is one and the rest are zeros. The final output attention map $\mathbf{a}$ is set to $\mathbf{a}_S$.

Compared to Neural Module Networks [Andreas et al., 2016a], our soft word attention implicitly supports the `CombineAnd` and `CombineOr` operations by attending to multiple words simultaneously. The embedding mask supports the `DescribeColor`, `DescribeLocation`, and `DescribeName` operations by masking different entries of embeddings according to different questions. As a result, we end up with a simple yet effective implementation of the network modularity.

Action The action module (Figure 2) assumes an attention map $\mathbf{a}$ from the language module and an environment map $\mathbf{g}$ from the visual perception module. These two maps contain all the information needed by the agent to move itself in the environment. We stack the two maps and input them to a two-layer CNN whose output is projected to a state vector $\mathbf{q}$ that summarizes the environment. The state vector is further projected to a distribution $\pi(\mathbf{q}, y)$ over the four actions. At each time step, the agent takes action y with a probability of $\alpha \cdot 0.25 + (1-\alpha) \cdot \pi(\mathbf{q}, y)$, where α is the rate of random exploration. The state vector is also projected to a scalar $V(\mathbf{q})$ as the approximate value function.

5 Training

Our training objective contains two sub-objectives, one for navigation and the other for recognition

$$\mathcal{L}(\theta) = \mathcal{L}_{RL}(\theta) + \mathcal{L}_{SL}(\theta),$$

where θ are the joint parameters of the framework. Most parameters are shared between the two tasks [3]. We compute the recognition loss $\mathcal{L}_{SL}$ as the multi-class cross entropy with the gradients

$$\nabla_\theta \mathcal{L}_{SL}(\theta) = \mathbb{E}_Q \left[-\nabla_\theta \log P_\theta(m|\mathbf{f}^*_\theta) \right],$$

where $\mathbb{E}_Q$ is the expectation over all the questions asked by the teacher in all training sessions, m is the correct answer to each question, and $\mathbf{f}^*_\theta$ is the corresponding feature. We compute the navigation loss $\mathcal{L}_{RL}(\theta)$ as the negative expected reward $-\mathbb{E}_{\pi_\theta}[r]$ the agent receives by following its policy π_θ. With the Actor-Critic (AC) algorithm [Sutton and Barto, 1998], we have the approximate gradients

$$\nabla_\theta \mathcal{L}_{RL}(\theta) = -\mathbb{E}_{\pi_\theta} \left[\left(\nabla_\theta \log \pi_\theta(\mathbf{q}_\theta, y) + \nabla_\theta V_\theta(\mathbf{q}_\theta) \right) \left(r + \gamma V_{\theta^-}(\mathbf{q}_{\theta^-}) - V_\theta(\mathbf{q}_\theta) \right) \right]$$

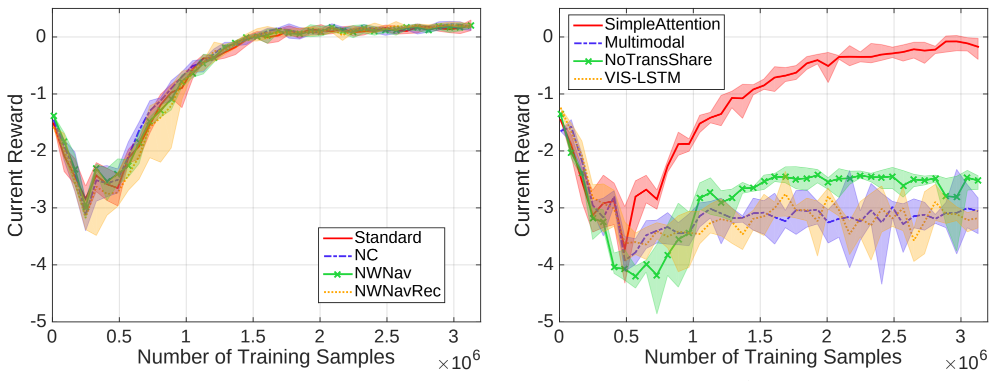
where θ^- are the target parameters that are periodically (every J minibatches) copied from θ, r is the immediate reward, γ is the discount factor, $\mathbf{q}_{\theta^-}$ is the next state after taking action y at state $\mathbf{q}_\theta$, and π_θ and V_θ are the policy and value output by the action module. Since the expectations $\mathbb{E}_Q$ and $\mathbb{E}_{\pi_\theta}$ are different, we optimize the two sub-objectives separately over the same number of minibatches. For effective training, we employ Curriculum Learning [Bengio et al., 2009] and Experience Replay [Mnih et al., 2015] with Prioritized Sampling [Schaul et al., 2016] (Appendix 8.4).

6 Experiments

We use Adagrad [Duchi et al., 2011] with a learning rate of 10^{-5} for Stochastic Gradient Descent (SGD). In all experiments, we set the batch size to 16 and train 200k batches. The target parameters θ^- are updated every $J = $ 2k batches. All the parameters have a default weight decay equal to $10^{-4} \times$ batch size. For each layer, by default its parameters have zero mean and a standard deviation of $1/\sqrt{N}$, where N is the number of parameters of that layer. The agent has 500k exploration steps in total, and the exploration rate α decreases linearly from 1 to 0. We fix the number of programming steps S as 3. We train each model with 4 random initializations. The whole framework is implemented with PaddlePaddle [4] and trained end to end. More implementation details are described in Appendix 8.1.

[3] In all the figures of our framework, components with the same name share the same set of parameters.

[4] `https://github.com/PaddlePaddle/Paddle`



(a) (b)

Figure 3: Training reward curves. The shown reward is the accumulated discounted reward per session, averaged every 8k training examples. The shaded area of each curve denotes the variance among 4 random initializations. (a) Curves of our framework under different command conditions. (b) Curves of the four baselines under the Standard command condition.

Zero-shot Navigation Our primary question is whether the agent has the zero-shot navigation ability of executing previously unseen commands. We setup four command conditions for training the agent.

- Standard The training command set has the same distribution with the test command set.
- NC Some word combinations are excluded from the training command set, even though all the words are in it. We specifically consider three types of word combinations: (object, location), (object, color), and (object, object). We enumerate all combinations for each type and randomly hold out 10% from the teacher in navigation.
- NWNav&NWNavRec Some object words are excluded from navigation training, and are trained only in recognition and tested in navigation as new concepts. NWNavRec guarantees that the new words will not appear in questions but only in answers while NWNav does not. We randomly hold out 10% of the object words.

Our framework is trained under each condition with the same hyperparameters. For testing, we put the held-out combinations/words back to the commands (*i.e.*, Standard condition) and test 10k sessions in total over four navigation subtasks `nav_obj`, `nav_col_obj`, `nav_nr_obj`, and `nav_bw_obj` (Appendix 8.3). We compute the success rates where success means that the agent reaches the target location in time in a session. Figure 3a shows the training reward curves and Table 1a contains the success rates. The curves are close to each other, which is expected because a 10% reduction of the commands barely changes the learning difficulty. We get almost the same success rates for all the conditions, and obtain high zero-shot success rates. The results of NWNavRec show that although some new object concepts are learned from a completely different problem, they can be tested on navigation without any model retraining or finetuning.

Ours/SA	Standard	NC	NWNav	NWNavRec
nav_obj	94.6/91.6	94.6/91.8	94.6/91.7	94.2/83.4
nav_col_obj	94.3/88.8	93.2/89.6	94.1/89.6	94.0/83.4
nav_nr_obj	93.1/74.5	92.7/74.2	93.0/77.2	93.0/70.8
nav_bw_obj	73.9/69.3	75.3/70.1	74.5/70.8	73.7/69.3
*nav_obj	N/A	93.8/90.6	94.5/88.3	94.7/4.1
*nav_col_obj	N/A	93.1/87.3	93.8/81.8	92.8/7.0
*nav_nr_obj	N/A	92.6/71.6	92.7/56.6	87.1/25.7
*nav_bw_obj	N/A	76.3/70.2	74.5/66.1	70.4/59.7

(a)

Ours	SimpleAttention	NoTransShare	VIS-LSTM	Multimodal
89.2	80.0	6.9	22.4	22.3

(b)

Table 1: Success rates (%). (a) Breakdown rates of our framework and SimpleAttention (SA) on the four subtasks under different training command conditions (columns). The last four rows show the rates of the test sessions that contain commands not seen in training. (b) Overall rates of all the methods under the Standard command condition.

Baselines To demonstrate that our framework architecture is necessary, we also test four baselines:

- SimpleAttention We modify our framework by replacing the programmer with a simple attention model. Given a sentence, we use an RNN to output an embedding which is convolved as a 3×3 filter with the visual feature map to get an attention map. The rest of the framework is unchanged (Figure 7 Appendix 8.5). This baseline is an ablation to show the necessity of the programmer.
- NoTransShare We do not tie the word embedding table $\mathbf{E}$ to the transposed Softmax matrix $\mathbf{S}^\intercal$. In this case, language grounding and recognition classification are not guaranteed to be the same. This baseline is an ablation to show the impact of the transposed sharing on the training convergence.

- VIS-LSTM Following Ren et al. [2015], we use CNN to get an image embedding which is then projected to the word embedding space and used as the first word of the sentence. The sentence goes through an RNN whose last state is used for navigation and recognition (Figure 8 Appendix 8.5).
- Multimodal We implement a multimodal framework [Mao et al., 2015]. The framework uses CNN to get an image embedding and RNN to get a sentence embedding. Then the two embeddings are projected to a common feature space for navigation and recognition (Figure 9 Appendix 8.5).

The last three baselines are trained only under the Standard command condition, where their performance is already very poor. SimpleAttention is trained under all the four command conditions. Again we test these baselines for 10k sessions. The training reward curves and the success rates are shown in Figure 3b and Table 1, respectively. VIS-LSTM and Multimodal have poor results because they do not ground language in vision. Surprisingly, NoTransShare converges much slower than our framework does. One possible reason is that the correct behavior of the language module is hard to be found by SGD if no constraint on word embedding is imposed. Although SimpleAttention is able to perform well, without word-attention programming, its ability of sentence understanding is limited. More importantly, Table 1a shows that it almost fails on zero-shot commands (especially on NWNavRec). Interestingly, in `nav_bw_obj` it has a much higher zero-shot success rate than in the other subtasks. The reason is that with the 3×3 filter, it always highlights the location between two objects without detecting their classes, because usually there is only one qualified pair of objects in the environment for `nav_bw_obj`. In such case, `SimpleAttention` does not really generalize to unseen commands.

Visualization and Analysis Our framework produces intermediate results that can be easily visualized and analyzed. One example has been shown in Figure 2, where the environment map is produced by a trained visual perception module. It detects exactly all the obstacles and goals. The map constitutes the learned prior knowledge for navigation: all walls and goals should be avoided by default because they incur negative rewards. This prior, combined with the attention map produced for a command, contains all information the agent needs to navigate. The attention maps are usually very precise (Figure 10 Appendix 8.5), with some rare cases in which there are flaws, *e.g.*, when the agent needs to navigate between two objects. This is due to our simplified assumption on 2D geometric translation: the attention map of a location word is treated as a filter and the translation is modeled as convolution. This results in attention diffusion in the above case. To address this issue, more complicated transformation can be used (*e.g.*, FC layers).

We also visualize the programming process. We observe that the programmer is able to shift its focus across steps (Figure 11 Appendix 8.5). In the first example, the programmer essentially does `Transform`[*southeast*](`Find`[*cabbage*](**F**)). In the second example, it essentially performs `Transform`[*between*](`CombineOr`(`Find`[*apple*](**F**), `Find`[*coconut*](**F**))).

We find the attention and environment maps very reliable through visualization. This is verified by the ∼100% QA accuracy in recognition. However, in Table 1 the best success rate is still $\sim 5\%$ away from the perfect. Further analysis reveals that the agent tends to stuck in loop if the target location is behind a long wall, although it has a certain chance to bypass it (Figure 12 Appendix 8.5). Thus we believe that the discrepancy between the good map quality and the imperfect performance results from our action module. Currently the action module learns a direct mapping from an environment state to an action. There is no support for either history remembering or route planning [Tamar et al., 2016]. Since our focus here is zero-shot navigation, we leave such improvements to future work.

7 Conclusion

We have demonstrated an end-to-end compositional framework for a virtual agent to generalize an existing skill to new concepts without model retraining or finetuing. Such generalization is made possible by reusing knowledge that is learned in other tasks and encoded by language. By assembling words in different ways, the agent is able to tackle new tasks while exploiting existing knowledge. Such ability is crucial for fast learning and good generalization. We reflect these important ideas in the design of our framework and apply it to a concrete example: zero-shot navigation in XWORLD. Our framework is just one possible implementation. Some components of the framework can still be improved. Our claim is not that an intelligent agent must have a mental model as the presented one, but it has to possess several crucial properties discussed in Section 1 and Section 4. Currently the agent explores in a 2D environment. In the future, we plan to migrate the agent to a 3D world like Malmo [Johnson et al., 2016]. There will be several new challenges, *e.g.*, visual perception and geometric transformation will be more difficult to model. We hope that the current framework provides some preliminary insights on how to train a similar agent in a 3D environment.

Acknowledgments

We thank Yuanpeng Li, Liang Zhao, and Yi Yang for their helpful discussions.

References

J. Andreas, M. Rohrbach, T. Darrell, and D. Klein. Neural module networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 39–48, 2016a.

J. Andreas, M. Rohrbach, T. Darrell, and D. Klein. Learning to compose neural networks for question answering. In *ACL*, pages 1545–1554, 2016b.

D. Bahdanau, K. Cho, and Y. Bengio. Neural machine translation by jointly learning to align and translate. In *ICLR*, 2015.

D. P. Barrett, S. A. Bronikowski, H. Yu, and J. M. Siskind. Robot language learning, generation, and comprehension. *arXiv preprint arXiv:1508.06161*, 2015.

Y. Bengio, J. Louradour, R. Collobert, and J. Weston. Curriculum learning. In *Proceedings of the 26th annual international conference on machine learning*, pages 41–48, 2009.

D. L. Chen and R. J. Mooney. Learning to interpret natural language navigation instructions from observations. In *AAAI*, volume 2, pages 1–2, 2011.

K. Cho, B. van Merrienboer, Ç. Gülçehre, F. Bougares, H. Schwenk, and Y. Bengio. Learning phrase representations using RNN encoder-decoder for statistical machine translation. In *Conference on Empirical Methods in Natural Language Processing*, 2014.

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

H. Gao, J. Mao, J. Zhou, Z. Huang, L. Wang, and W. Xu. Are you talking to a machine? dataset and methods for multilingual image question. In *NIPS*, pages 2296–2304, 2015.

Q. Gao, M. Doering, S. Yang, and J. Y. Chai. Physical causality of action verbs in grounded language understanding. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*, volume 1, pages 1814–1824, 2016.

A. Graves, G. Wayne, and I. Danihelka. Neural turing machines. *arXiv preprint arXiv:1410.5401*, 2014.

M. Jaderberg, V. Mnih, W. M. Czarnecki, T. Schaul, J. Z. Leibo, D. Silver, and K. Kavukcuoglu. Reinforcement learning with unsupervised auxiliary tasks. In *ICLR*, 2017.

M. Johnson, K. Hofmann, T. Hutton, and D. Bignell. The malmo platform for artificial intelligence experimentation. In *Proc. 25th International Joint Conference on Artificial Intelligence*, 2016.

M. Kempka, M. Wydmuch, G. Runc, J. Toczek, and W. Jaśkowski. ViZDoom: A Doom-based AI research platform for visual reinforcement learning. In *IEEE Conference on Computational Intelligence and Games*, pages 341–348, 2016.

D. Kiela, L. Bulat, A. L. Vero, and S. Clark. Virtual embodiment: A scalable long-term strategy for artificial intelligence research. In *NIPS Workshop*, 2016.

B. M. Lake, T. D. Ullman, J. B. Tenenbaum, and S. J. Gershman. Building machines that learn and think like people. *Behavioral and Brain Sciences*, pages 1–101, 11 2016.

J. Lu, J. Yang, D. Batra, and D. Parikh. Hierarchical question-image co-attention for visual question answering. In *Advances In Neural Information Processing Systems*, pages 289–297, 2016.

J. Mao, X. Wei, Y. Yang, J. Wang, Z. Huang, and A. L. Yuille. Learning like a child: Fast novel visual concept learning from sentence descriptions of images. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2533–2541, 2015.

T. Mikolov, A. Joulin, and M. Baroni. A roadmap towards machine intelligence. *arXiv preprint arXiv:1511.08130*, 2015.

V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540): 529–533, 2015.

A. Neelakantan, Q. V. Le, and I. Sutskever. Neural programmer: Inducing latent programs with gradient descent. In *International Conference on Learning Representations*, 2016.

M. Ren, R. Kiros, and R. Zemel. Exploring models and data for image question answering. In *Advances in Neural Information Processing Systems*, pages 2953–2961, 2015.

A. Rohrbach, M. Rohrbach, R. Hu, T. Darrell, and B. Schiele. Grounding of textual phrases in images by reconstruction. In *European Conference on Computer Vision*, pages 817–834, 2016.

T. Schaul, J. Quan, I. Antonoglou, and D. Silver. Prioritized experience replay. In *ICLR*, 2016.

M. Schuster and K. K. Paliwal. Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing*, 45(11):2673–2681, 1997.

S. Sukhbaatar, A. Szlam, G. Synnaeve, S. Chintala, and R. Fergus. Mazebase: A sandbox for learning from games. *arXiv preprint arXiv:1511.07401*, 2016.

R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge, 1998.

A. Tamar, S. Levine, P. Abbeel, Y. WU, and G. Thomas. Value iteration networks. In *Advances in Neural Information Processing Systems*, pages 2146–2154, 2016.

D. Teney and A. v. d. Hengel. Zero-shot visual question answering. *arXiv preprint arXiv:1611.05546*, 2016.

Z. Yang, X. He, J. Gao, L. Deng, and A. Smola. Stacked attention networks for image question answering. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 21–29, 2016.

H. Yu and J. M. Siskind. Grounded language learning from video described with sentences. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, pages 53–63, 2013.

8 Appendix

8.1 Implementation Details

The agent at each time step receives a 156×156 RGB image. This image is egocentric and includes both the environment and the black padding region. The agent processes the input image with a CNN that has four convolutional layers: $(3, 3, 64), (2, 2, 64), (2, 2, 512), (1, 1, 512)$, where (x, y, z) represents z $x \times x$ filters with stride y. All the four layers have the ReLU activation function. The output is the visual feature map with 512 channels. We stack it along the channel dimension with another parametric spatial feature map of the same sizes. This spatial feature map is initialized with zero mean and standard deviation (Figure 2).

The agent also receives a navigation command at the beginning of a session. The same command is repeated until the end of the session. The agent may or may not receive a question at every time step. The dimensions of the word embedding, syntax embedding, and functionality embedding are 1024, 128, and 128, respectively. The word embedding table is initialized with zero mean and a standard deviation of 1. The hidden FC layers for computing the syntax and functionality embeddings have 512 units (Figure 4). The bidirectional RNN for computing sentence contexts has a state size of 128 in both directions. The output controller booting vector and sentence context also have a length of 128. The state size of the controller RNN is equal to the length of the booting vector (Figure 2 Right). The hidden FC layer for converting a functionality embedding to an embedding mask has a size of 128. The RNN used for summarizing the question intention has 128 states (Figure 5). All FC layers and RNN states in the language and recognition module use Tanh as the activation function. The only exception is the FC layer that outputs the embedding mask (Sigmoid).

In the action module, the CNN for processing the attention map and the environment map has two convolutional layers $(3, 1, 64)$ and $(3, 1, 4)$, both with paddings of 1. They are followed by three FC layers that all have 512 units. All five layers use the ReLU activation function.

8.2 Baseline Models

The language module of SimpleAttention sets the word embedding size to 1024. The RNN has the same size with the word embedding. The FC layer that produces the 3×3 filter has an output size of 4608 which is 9 times the channel number of the visual feature map. The rest of the layer configuration is the same with our framework. VIS-LSTM has a CNN with four convolutional layers $(3, 2, 64)$, $(3, 2, 64)$, $(3, 2, 128)$, and $(3, 1, 128)$. This is followed by three FC layers with size 1024. The word embedding and the RNN both have sizes of 1024. The RNN output goes through three FC hidden layers of size 512 either for recognition or navigation. The layer size configuration of Multimodal is the same with VIS-LSTM. The outputs of all layers here are ReLU activated except for the last FC layer of the CNN used by VIS-LSTM. The activation is instead linear so that the output image embedding is in the same space with the word embedding. Except for SimpleAttention, we do not tie the transposed Softmax matrix to the embedding table.

8.3 XWORLD Setup

We configure square environments with sizes ranging from 3 to 7. We fix the size of the environment image by padding walls for smaller environments. Different sessions may have different map sizes. In each session,

- The number of time steps T is four times the map size.
- The number of objects on the map is from 1 to 3.
- The number of wall grids on the map is from 0 to 10.
- The positive reward R^+ when the agent reaches the correct location is set to 1. The negative rewards R_w^- for hitting walls and R_o^- for stepping on non-target objects are set to -0.2 and -1, respectively. The time step penalty R_t^- is set to -0.1.

The teacher has a vocabulary size of 104, including 2 punctuation marks. There are 9 locations, 4 colors, and 40 distinct object classes. Each object class has 2.85 object instances on average. Every time the environment is reset, a number of object classes are randomly sampled and an object instance is randomly sampled for each class. There are in total 16 types of sentences the teacher can speak, including 4 types of navigation commands and 12 types of recognition questions. Each sentence type has multiple non-recursive natural-language templates, and corresponds to a subtask the agent must learn to perform. In total there are 256,832 distinct sentences with 92,442 for the navigation task and 164,390 for the recognition task. The sentence length ranges from 2 to 12.

The object, location, and color words of the teacher's language are listed below. These are the content words with actual meanings that can be grounded in the environment. All the other words are treated as grammatical words whose embeddings are only for interpreting sentence structures. The differentiation between content and grammatical words is automatically learned by the agent based on the teacher's language and the environment. All words have the same form of representation.

Object	Location	Color	Other
apple, avocado, banana, blueberry, butterfly, cabbage, cat, cherry, circle, coconut, cucumber, deer, dog, elephant, fig, fish, frog, grape, hedgehog, ladybug, lemon, lion, monkey, octopus, orange, owl, panda, penguin, pineapple, pumpkin, rabbit, snake, square, squirrel, star, strawberry, triangle, turkey, turtle, watermelon	*east, west, north, south, northeast, northwest, southeast, southwest, between*	*green, red, blue, yellow*	*?, ., and, block, by, can, color, could, destination, direction, does, find, go, goal, grid, have, identify, in, is, locate, located, location, me, move, name, navigate, near, nothing, object, of, on, one, OOV, please, property, reach, say, side, target, tell, the, thing, three, to, two, what, where, which, will, you, your*

The sentence types that the teacher can speak are listed below. Each sentence type corresponds to a subtask. The triggering condition describes when the teacher says that type of sentences. Besides the conditions shown, an extra condition for navigation commands is that the target location must be reachable from the current agent location. An extra condition for color-related questions is that the object color must be one of the four defined colors, and objects with other colors will be ignored in these questions. If at a time step there are multiple conditions triggered, we randomly sample one sentence type for navigation and another for recognition. After the sentence type is sampled, we generate the command or question according to the corresponding sentence templates.

Sentence Type (Subtask)	Example	Triggering Condition
`nav_obj`	*Please go to the apple.*	[C0] Beginning of a session. & [C1] The referred object has a unique name.
`nav_col_obj`	*Could you please move to the red apple?*	[C0] & [C2] There are multiple objects that either have the same name but different colors, or have different names but the same color.
`nav_nr_obj`	*The north of the apple is your destination.*	[C0] & [C1]
`nav_bw_obj`	*Navigate to the grid between apple and banana please.*	[C0] & [C3] Both referred objects have unique names and are separated by one grid.
`rec_col2obj`	*What is the red object?*	[C4] There is only one object that has the referred color.
`rec_obj2col`	*What is the color of the apple?*	[C1]
`rec_loc2obj`	*Please tell the name of the object in the south.*	[C5] The agent is near the referred object.
`rec_obj2loc`	*What is the location of the apple?*	[C1] & [C5]
`rec_loc2col`	*What color does the object in the east have?*	[C5]
`rec_col2loc`	*Where is the red object located?*	[C4] & [C5]
`rec_loc_obj2obj`	*Identify the object which is in the east of the apple.*	[C1] & [C6] The referred object is near another object
`rec_loc_obj2col`	*What is the color of the east to the apple?*	[C1] & [C6]
`rec_col_obj2loc`	*Where is the red apple?*	[C2] & [C5]
`rec_bw_obj2obj`	*What is the object between apple and banana?*	[C7] Both referred objects have unique names and are separated by one object.
`rec_bw_obj2loc`	*Where is the object between apple and banana?*	[C7] & [C8] The agent is near the object in the middle.
`rec_bw_obj2col`	*What is the color of the object between apple and banana?*	[C7]

8.4 Experience Replay and Curriculum Learning

We employ Experience Replay [Mnih et al., 2015] for training both the navigation and recognition tasks. The environment inputs, rewards, and the actions taken by the agent at the most recent 10k time steps are stored in a replay buffer. During training, every time two minibatches of the same number of experiences are sampled from the buffer, one for computing $\nabla_\theta \mathcal{L}_{SL}(\theta)$ and the other for computing $\nabla_\theta \mathcal{L}_{RL}(\theta)$. For the former, only individual experiences are sampled. We uniformly sample experiences from a subset of the buffer which contains the teacher's questions. For the latter, we need to sample transitions (*i.e.*, pairs of experiences) so that TD error can be computed. We sample from the entire buffer using the Rank-based Sampler [Schaul et al., 2016] which has proven to increase the learning efficiency by prioritizing rare experiences in the buffer.

Because in the beginning the language is quite ambiguous, it is difficult for the agent to start learning with a complex environment setup. Thus we exploit Curriculum Learning [Bengio et al., 2009] to gradually increase the environment complexity. We gradually change the following things linearly in proportional to $\min(1, G' \,/\, G)$, where G' is the number of sessions so far and G is the number of curriculum sessions:

- ○ The number of grids of the environment.
- ○ The number of objects in the environment.
- ○ The number of wall grids.
- ○ The number of possible object classes that can appear in the environment.
- ○ The length of a navigation command or a recognition question.

We find that this curriculum is crucial for efficient learning, because in the early phase the agent is able to quickly master the meanings of the location and color words given only small ambiguity. After this, these words are used to guide the optimization when more and more new sentence structures and objects are added. In the experiments, we set $G = 10\text{k}$ during training while do not use any curriculum during test (maximal difficulty).

8.5 Figures

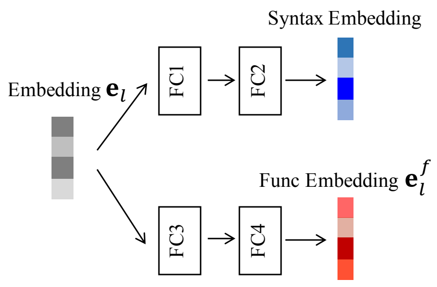


Figure 4: The projections from word embedding to syntax embedding and functionality embedding.

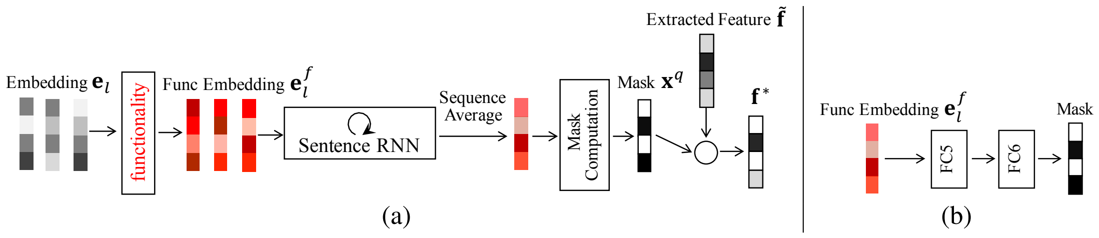


Figure 5: Details of computing the embedding masks. (a) The pipeline of Question Intention in Figure 2b, which computes an embedding mask according to a question. (b) The two FC layers used by Mask Computation in both (a) and Figure 2 Right. They project a functionality embedding to a mask.

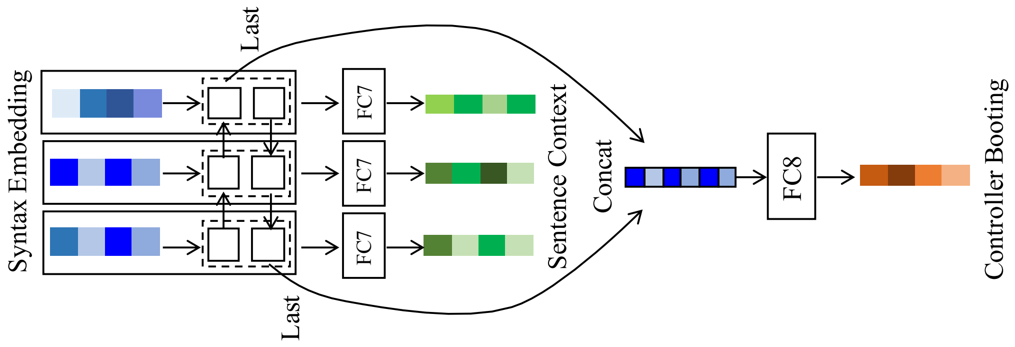


Figure 6: A Bidirectional RNN that receives a sequence of syntax embeddings, and outputs a sequence of sentence contexts and a controller booting vector.

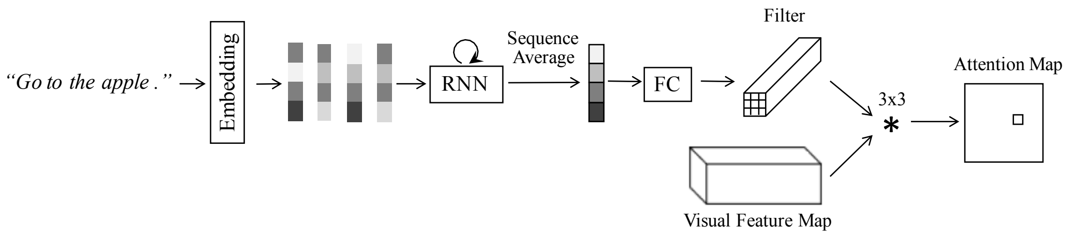


Figure 7: The pipeline of the language module of the SimpleAttention baseline. An RNN processes the input sentence and outputs a 3×3 filter which is convolved with the visual feature map generated by the visual perception module. Other modules are the same with our framework.

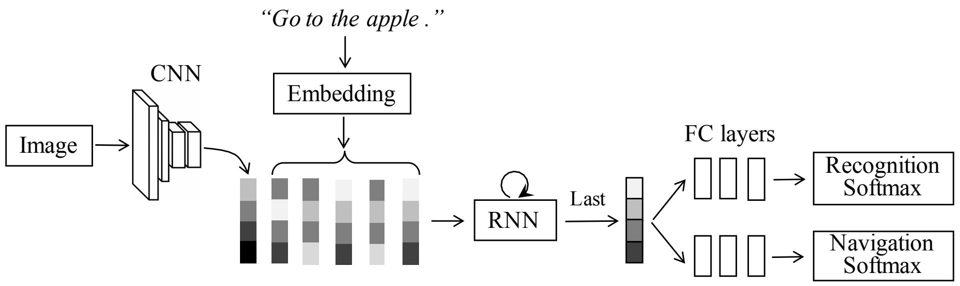


Figure 8: Our adapted version of the VIS-LSTM model [Ren et al., 2015]. The framework treats the projected image embedding as the first word of the sentence. Navigation and recognition only differ in the last several FC layers.

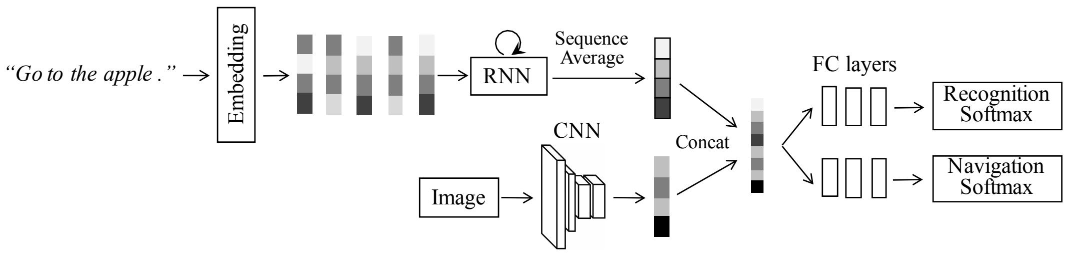


Figure 9: A multimodal framework adapted from the one in Mao et al. [2015]. The image and the sentence are summarized by a CNN and an RNN respectively. Their embeddings are concatenated and projected to the same space. Navigation and recognition only differ in the last several FC layers.

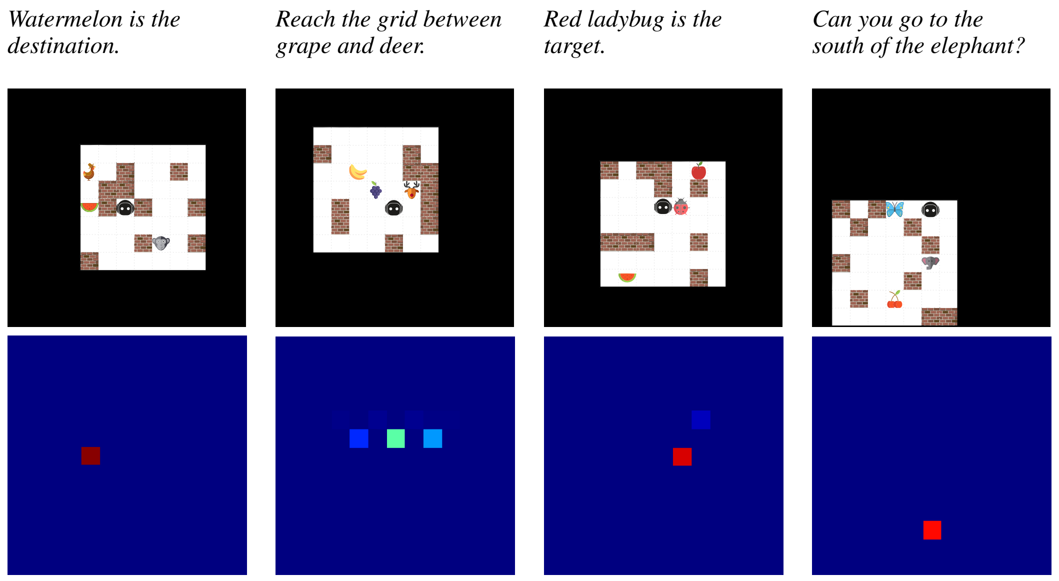

Watermelon is the destination.
Reach the grid between grape and deer.
Red ladybug is the target.
Can you go to the south of the elephant?

Figure 10: Examples of attention maps in different sessions. Top: the navigation commands. Middle: the current environment images. Bottom: the corresponding attention maps output by the language module. Note that attention maps are all egocentric: the map center is the agent's location.

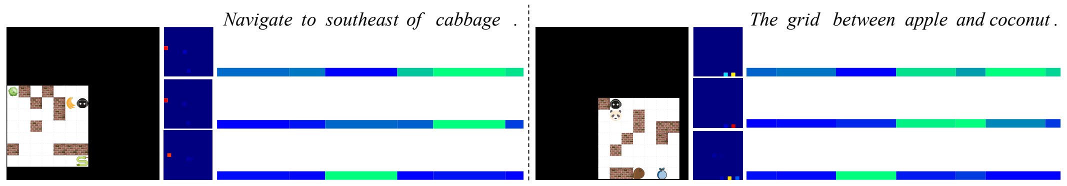

Navigate to southeast of cabbage .
The grid between apple and coconut .

Figure 11: Illustration of the language programming process on two examples. Given the current environment image and a navigation command, the programmer generates an attention map in three steps. At each step the programmer focuses on a different portion of the sentence. The word attention is visualized by a color strip where brighter portion means more attention. On the left of each color strip is the corresponding attention map combining the current attention and the previously cached one (Figure 2 Right). The last attention map is used as the output of the programmer.

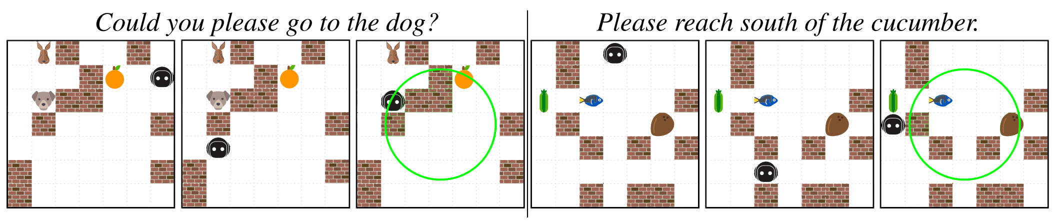

Could you please go to the dog?
Please reach south of the cucumber.

Figure 12: Examples of bypassing long walls. For each path, only three key steps are shown. (Green circles indicate successes.)