Neural Models for Simple Algorithmic Games

Sainbayar Sukhbaatar
Dept. of Computer Science
Courant Institute
New York University

Arthur Szlam
Facebook AI Research
New York

Rob Fergus
Facebook AI Research
New York

1 Introduction

In the past few years there has been a resurgence of work on neural models for game playing [5, 2, 6]. Several works have also demonstrated the ability of neural models to learn to answer common-sense questions about a restricted environment [11], and to carry out simple algorithms, such as sorting and reversal of inputs [1, 9, 3, 13]. In [5, 2, 6, 3, 13] the models were trained with reinforcement learning or using discrete search, allowing possibly delayed rewards with discrete action spaces.

In this paper, we introduce a simple 2D game environment over which we have total control. We devise a series of tasks within this environment that carefully explore a range of elementary algorithmic elements, increasing in complexity. The environment can be thought of as a small practical step towards an implementation of some of the ideas discussed at length in [4]. However, here, our ambitions are more local, and we try to focus more finely on the border where current models fail (but nearly succeed), rather than aim for a global view of a path towards AI. We explore various model architectures (convolutional net vs memory network vs convolution+ attention) for learning to solve the tasks, and compare supervised imitation learning vs reinforcement. Finally, we explore using question answering as a method for giving supervision without direct supervision of the task.

2 Environment and tasks

Each game is played in a 2D rectangular grid, whose dimensions range from 5 to 10 on each side. Each location in the grid can be empty, or may contain one or more items. The agent can move in each of the four cardinal directions, assuming no item blocks the agents path. The items in the game are:
Block: an impassible obstacle that does not allow the agent to move to that grid location.
Water: the agent may move to a grid location with water, but incurs an additional cost of -0.2 for doing so.
Switch: a switch can be in one of m states, which we refer to as colors. The agent can toggle through the states cyclically by moving on top of the switch.
Door: a door has a color, matched to a particular switch. The agent may only move to the door's grid location if the state of the switch matches the state of the door.
Goal: depending on the task, one or more goals may exist, each named individually.
Info: these items do not have a grid location, but can specify a task or give information necessary for its completion.

The environment is presented to the agent as a list of sentences, and is compatible with the format of the bAbI tasks, introduced in[10]. However, note that we use egocentric spatial coordinates (e.g. the goal G1 in Fig. 1 (left) is at coordinates [+2,0]), meaning that the environment updates the locations of each object after an action*. Furthermore, for tasks involving multiple goals, we have two versions of the game. In one, the environment automatically sets a flag on visited goals. In the harder versions, this mechanism is absent but the agent has a special action that allows it to release a "breadcrumb" into the environment, enabling it to record locations it has visited.

*This is consistent with [10] , where the "agent" answering the questions was also given them in egocentric temporal coordinates

The environments are generated randomly with some distribution on the various items. For example, we usually specify a uniform distribution over height and width (between 5 and 10 for the experiments reported here), and a percentage of wall blocks and water blocks (each range randomly from 0 to 20%).

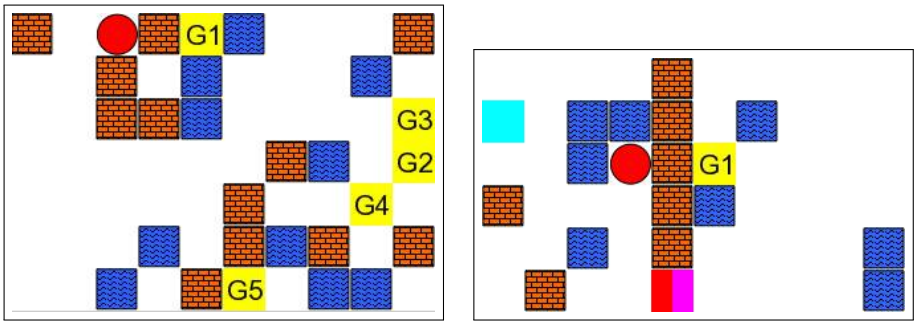

Figure 1: Examples of Multigoal (left) and Light Key (right) tasks. Note that the layout and dimensions of the environment varies between different instances of each task (i.e. the location and quantity of walls, water and goals all change). The agent is shown as a red blob and the goals are shown in yellow. For LightKey, the switch is show in cyan and the door in magenta/red (toggling the switch will change the door's color, allowing it to pass through).

2.1 Tasks

Although our game world is simple, it allows for a rich variety of tasks. In this work, we explore tasks that require different algorithmic components (such as conditional reasoning or planning short routes) first in isolation and then in combination. Our goal is to build tasks that may require a few stages of action, and where each stage is relatively basic. We avoid tasks that require unbounded loops or recursion, as in [3], and instead view "algorithms" more in the vein of following a recipe from a cookbook. In particular, we want our agent to be able to follow directions; the same game world may host multiple tasks, and the agent must decide what to do based on the "Info".

In all of the tasks, the agent incurs a fixed penalty for each action it makes. In the experiments below, this is set to 0.1. In addition, stepping on a Water block incurs an additional penalty of 0.2. For most games, a maximum of 50 actions are allowed. The tasks define extra penalties and conditions for the game to end.

Multigoals: In this task, the agent is given an ordered list of goals as "Info", and needs to visit the goals in that order. In the experiments below, the number of goals ranges from 2 to 5, and the number of "active" that the agent is required to visit ranges from 1 to 3. The agent is not given any extra penalty for visiting a goal out of order, but visiting a goal before its turn does not count towards visiting all goals. The game ends when all goals are visited.

Switches: In this task, the game has a random number of switches on the board. The agent is told via the "Info" to toggle all switches to the same color, and the agent has the choice of color; to get the best reward, the agent needs to solve a (very small) traveling salesman problem. In the experiments below, the number of switches ranges from 1 to 5 and the number of colors from 1 to 3. The task finishes when the switches are correctly toggled. There are no special penalties in this task.

Conditional Goals: In this task, the destination goal is conditional on the state of a switch. The "Info" is of the form "go to goal g_i if the switch is colored c_j, else go to goal g_l." In the experiments below, the number of the number of colors range from 2 to 5 and the number of goals from 2 to 5. Note that there can be more colors than goals or more goals than colors. The task concludes when the agent reaches the specified goal; in addition, the agent incurs a 0.2 penalty for stepping on an incorrect goal, in order to encourage it to read the info (and not just visit all goals).

Exclusion: The "Info" in this game specifies a list of goals to avoid. The agent should visit all other unmentioned goals. The number of goals ranges form 1 to 5. As in the Conditional goals game, the agent incurs a 0.5 penalty when it steps on a forbidden goal.

Light Key: In this game, there is a switch and a door in a wall of blocks. The agent should navigate to a goal which may be on the wrong side of a wall of blocks. If the goal is on the same side of the wall as the agent, it should go directly there; otherwise, it needs move to and toggle the switch to open the door before going to the goal. There are no special penalties in this game, and the game ends when the agent reaches the goal.

2.2 Question tasks

We also explore an indirect form of supervision during training by automatically generating a series of text question and answer pairs about the current game state. For example (using Fig. 1) these might be (i) Q: What is at location [0,-2]?, A: wall or (ii) Q: Where is G5?, A: [+3,-6] etc. These can be presented to the model in the same format as "Info" and help the model learn the correspondence between words and objects in the game world.

3 Models

We investigate several different types of model: (i) simple linear, (ii) convolutional nets, (iii) convolutional nets with attention and (iv) end-to-end memory networks [11, 7]. While the input format is quite different for each approach (detailed below), the outputs are the same: a set of discrete actions $\{$N,S,E,W,drop breadcrumb,toggle switch$\}$; and a continuous baseline for reinforcement.

For a simple baseline we take the existence of each possible object-location pair on the largest grid we consider (10×10) and each "Info" item as a separate feature, and train a linear classifier to the action space from these features.

We also use a convolutional model. All the items in the game that have a spatial location are presented to a convolutional network via a 1 of n encoding (n being the number of item types). Hence the environment is presented as a 3D cube of size height $\times$ width $\times$ n. Each "Info" item is represented as a bag of words, and then the set of "Info" items is combined via a fully connected layer with the outputs of the convolutional layers; these are then passed through two fully connected layers to output the actions (and baseline for reinforcement).

We also tried adding a simple spatial attention mechanism to convolutional networks. After the first convolution, attention weights are computed by applying a softmax layer to dot products between feature maps and the features from "Info" items. Then, the feature maps are multiplied by the attention weights and the rest of the convolutional net is applied.

For the memory network, all items in the game (both physical items as well as "info") are represented as bag-of-words vectors. The spatial location of each item is also represented as a word within the bag. E.g. a red door at [+3,-2] becomes the vector $\{$red_door$\}$ + $\{$x=+3,y=-2$\}$, where $\{$red_door$\}$ and $\{$x=+3,y=-2$\}$ are word vectors of dimension 50. These embedding vectors will be learned at training time. As a consequence, the memory network has to learn the spatial arrangement of the grid, unlike the convolutional network. Otherwise, we use the architecture from [7] with 3 hops and tanh nonlinearities.

4 Training Procedures

4.1 Supervision

For Multigoals and Light Key, we compute an off-line solution that yields minimum cost. For the other tasks, we could compute such a solution, but to make the computations faster, we choose the visitation order of the switches or goals greedily, rather than solving a (admittedly small) traveling salesman problem.

4.2 Reinforcement

We use policy gradient with learned baseline as in [12] for training. Instead of using a single baseline for every state, we let the model output a baseline value specific to the current state. This is accomplished by adding two heads to models: one that predicts the action, the other the baseline value. Beside maximizing the expected reward with policy gradient, the models are trained to minimize the distance between the baseline value and actual reward. The gradients from the baseline are given .03 weight w.r.t. the gradients from the reward.

RMSProp [8] is used for parameter updating with learning rates optimized for each model type (0.003 for linear model, 0.001 for memory networks and 0.0003 for convolutional networks). Also, gradient clipping is used to avoid instability during training. Each model is trained on total of 5.1M game plays, which are grouped into mini batches of 512 games that are played in parallel. The whole training process took about 10 hours on 18 CPUs.

5 Experimental Results

Fig. 2 shows the performance of different models on the games (following subtraction of the optimal reward). For each model/training combination, we take the best run of 10. In general, all of the methods make progress and, after sufficient training, the tasks are completed with high probability. Indeed, complex tasks such as Light Key can be successfully trained using only reinforcement. However, none of the learners are able to approach optimal performance, and in fact have a reward at least a factor of 2 worse. More subtly, even though Conditional Goals has a respectable completion rate of $\sim$75%, closer inspection reveal the models do not learn the correct algorithm. Note that a reward of -2 (vs $\sim -.5$ optimal) in this task allows the agent to visit several goals before picking the true one, even with the penalty for visiting the wrong goal.

5.1 Supervision vs. Reinforcement

In most of these tasks, adding supervision did not make a large difference to the final outcome of the best model. However, it did speed up convergence.

Supervision *was* necessary to successfully learn the breadcrumb action (note that the chart in figure 2 shows results with the environment given "visited" flag). In the Multigoals task with the breadcrumb action, supervision increases success from 56% to 94%, and reward from -3.02 to -1.48; in Exclusion, success increased from 2% to 97% and reward from -5.00 to -1.18.

5.2 Question-answering vs. Imitation

For most of the tasks, we did not see any improvement using question-answering with labels over plain reinforcement. However, on the Conditional goals task, where the models had trouble learning the correct algorithm even with supervision, training to answer questions of the form "what color is the switch?" and "which goal should you go to?" improved the chance that the memory net succeeded, and was able to approach the ground truth reward and qualitatively learn the correct algorithm.

5.3 Memory Networks vs. Convolutional Nets

Overall, the performance of the convolutional model and the memory-network model was similar. However, the convolutional model was unable to solve the multi-goals task. We hypothesize that this is because for large numbers of goals, it is difficult to compactly encode the information about where the goals are in such a way that it can be easily unpacked and combined with the order information from the "Info" items. Note that attention in the convnet does help the performance.

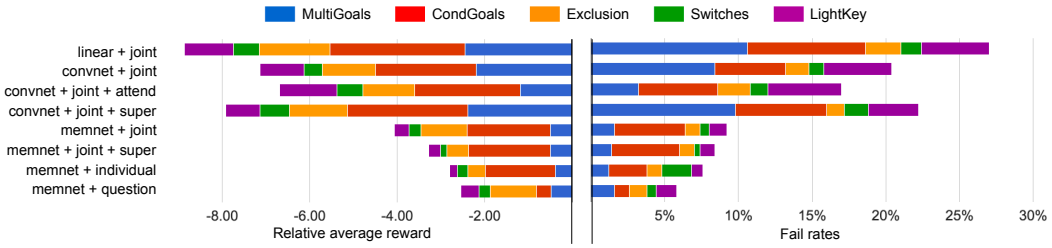

Figure 2: Average rewards for each method, after subtracting off the optimal reward. "Joint" refers to a single model trained on all 5 tasks simultaneously. The absolute optimal reward (summed across all 5 tasks) is -3.05. The results reported are the best of 10 runs.

6 Conclusions

We show that neural models trained with reinforcement can do well on the simple tasks we have presented, although the resulting policies can be far from optimal, even with explicit supervision. The memory networks were able to solve some tasks that the convnets could not, although overall the performance was similar. Finally we showed that supervised question-answering can improve results on some tasks.

References

[1] A. Graves, G. Wayne, and I. Danihelka. Neural turing machines. *CoRR*, abs/1410.5401, 2014.

[2] X. Guo, S. Singh, H. Lee, R. L. Lewis, and X. Wang. Deep learning for real-time atari game play using offline monte-carlo tree search planning. In Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 3338–3346. Curran Associates, Inc., 2014.

[3] A. Joulin and T. Mikolov. Inferring algorithmic patterns with stack-augmented recurrent nets. *CoRR*, abs/1503.01007, 2015.

[4] T. Mikolov, A. Joulin, and M. Baroni. A roadmap towards machine intelligence. *CoRR*, abs/1511.08130, 2015.

[5] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller. Playing atari with deep reinforcement learning. In *NIPS Deep Learning Workshop*. 2013.

[6] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 02 2015.

[7] S. Sukhbaatar, A. Szlam, J. Weston, and R. Fergus. End-to-end memory networks, 2015.

[8] T. Tieleman and G. Hinton. Lecture 6.5—RmsProp: Divide the gradient by a running average of its recent magnitude. COURSERA: Neural Networks for Machine Learning, 2012.

[9] O. Vinyals, M. Fortunato, and N. Jaitly. Pointer networks, 2015.

[10] J. Weston, A. Bordes, S. Chopra, and T. Mikolov. Towards ai-complete question answering: A set of prerequisite toy tasks. In *arXiv preprint: 1502.05698*, 2015.

[11] J. Weston, S. Chopra, and A. Bordes. Memory networks. In *arXiv preprint: 1410.391v8*, 2014.

[12] R. J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. In *Machine Learning*, pages 229–256, 1992.

[13] W. Zaremba and I. Sutskever. Reinforcement learning neural turing machines, 2015.