Asynchronous Data Aggregation for Training End to End Visual Control Networks

Mathew Monfort
Massachusetts Institute of Technology
mmonfort@mit.edu

Matthew Johnson
Microsoft Research Cambridge
matjoh@microsoft.com

Aude Oliva
Massachusetts Institute of Technology
oliva@mit.edu

Katja Hofmann
Microsoft Research Cambridge
katja.hofmann@microsoft.com

ABSTRACT

Robust training of deep neural networks requires a large amount of data. However gathering and labeling this data can be expensive and determining which distribution of features are needed for training is not a trivial problem. This is compounded when training neural networks for autonomous navigation in continuous non-deterministic environments using only visual input. Increasing the quantity of demonstrated data does not solve this problem as the demonstrated sequences of actions are not guaranteed to produce the same outcomes and slight changes in orientation generate drastically different visual representations. This results in a training set with a different distribution than what the agent will typically encounter in application. Here, we develop a method that can grow a training set from the same distribution as the agent's experiences and capture useful features not found in demonstrated behavior. Additionally, we show that our approach scales to efficiently handle complex tasks that require a large amount of data (experiences) for training. Concretely, we propose the deep asynchronous Dagger framework, which combines the Dagger algorithm with an asynchronous actor-learner architecture for parallel dataset aggregation and network policy learning. We apply our method to the task of navigating 3D mazes in Minecraft with randomly changing block types and analyze our results.

Keywords

visual navigation, autonomous navigation, autonomous agent, deep learning, reinforcement learning, active learning

1. INTRODUCTION

Training an agent to make decisions based solely on egocentric visual input is difficult. The high-dimensionality and variance in raw images coupled with the challenge of completing a task with partial observability of the state of the world makes traditional approaches to sequential learning intractable [15]. Many of these methods rely on the use of hand engineered features that describe the current state and learn a policy (mapping from states to actions) to complete a task. Choosing these features requires knowledge of the problem being solved and is not trivial in complex domains [15, 17].

Recent work in deep reinforcement learning [17, 11] addressed this limitation by learning feature representations with neural networks, eliminating the need for hand engineered features while learning a model free policy. These methods either rely on unsupervised exploration requiring extensive computation [17] or full state observability to jointly train the network with a guiding policy [11].

Asynchronous methods have been proposed to improve the scalability of deep reinforcement learning to complex problem domains by performing parallel network training and agent (network) exploration [18, 16, 24]. However, the behavior learned by an unsupervised agent may differ greatly from human behavior which is not always desired for agents that will interact with humans [4, 2]. Additionally, full state knowledge cannot be guaranteed in all problem settings.

The problem of deviating from human behavior can be alleviated by learning from demonstrations. However, in a sequential decision process, such as navigation, small approximation errors can compound and lead an agent into a state not captured by the training set [19, 11, 20]. Recent approaches propose artificially augmenting the demonstrated data in order to ensure the agent is trained for circumstances that may not appear in human demonstrations [2]. This artificial extension to the dataset may not be representative of the distribution of experiences the agent will encounter in practice [19, 20] and deciding the scope and size of the augmented dataset needed for efficiently training the network is not trivial [25].

The Dagger algorithm [19] for reinforcement learning addresses the compounding error and differing distribution problems by using a weighted average of the agent policy and an expert policy to collect the training data. After each iteration (task execution), the sequence of states the agent explores are added to the training set and used to update the agent policy. As learning progresses, the agent policy is gradually given greater control in the decision process. This allows for the collection of a training set that is derived from the experiences of the agent. A limitation to this method is that every explored state is added to the dataset. This large collection of data can become prohibitive when considering high-dimensional state representations such as images. Additionally, adding states that the agent policy has adequately learned leads to an unnecessary increase in training complexity [20].

We propose efficiently growing a minimally optimal training set

Appears in: *Proc. of the 16th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2017), S. Das, E. Durfee, K. Larson, M. Winikoff (eds.), May 8–12, 2017, São Paulo, Brazil.*
Copyright © 2017, International Foundation for Autonomous Agents and Multiagent Systems (www.ifaamas.org). All rights reserved.

for visual navigation from the same distribution as the agent's experiences by appending data when the network makes an incorrect decision. The quality of the decision can be evaluated online using a simple training policy, π_t, that directs the agent to the nearest state in a single demonstrated trajectory through the space using local domain knowledge. Initially running the agent with π_h and iteratively giving the network more control allows for it to explore and gather data when it makes mistakes. This is analogous to curriculum learning [1] where the network is slowly exposed to a more difficult task for training. We extend this method to the asynchronous setting allowing it to scale to efficiently handle complex tasks that require a large amount of varied data (experiences) for training and apply it to the task of learning to navigate 3D mazes in Minecraft.

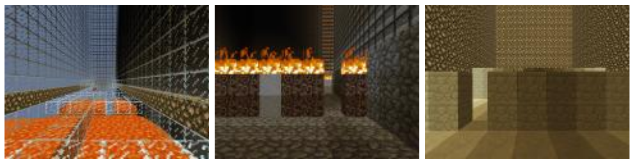

Figure 1: Example 3D Mazes in Minecraft.

Formally, we propose the deep asynchronous Dagger framework which combines the Dagger algorithm [19] with an asynchronous actor-learner architecture [16] for parallel dataset aggregation and network policy learning. Multiple agents sample network parameters from a learner node and explore environments in parallel, appending data from incorrect decisions to a central replay memory. This memory can then be accessed by the learner node to continuously update the network controller parameters.

The key contribution of this approach is the ability to grow a data-efficient training set for visual navigation from single shot (one example per task) demonstrated data. We apply our method to the task of continuously navigating non-deterministic 3D mazes in Minecraft[1] with randomly changing block types (Figure 1) and analyze our results.

2. RELATED WORK

Recent work in deep Q learning (DQN) [17] addressed the limitations of using hand-engineered features for learning sequential decision models by learning neural network parameters to estimate the expected value of actions from a given state using raw visual input and a variant of an unsupervised Q-learning algorithm [23]. This eliminates the need for hand engineered features and learns a model free policy. The network is trained using an experience replay memory [14] where the state, action, reward and next state observed by the agent are stored in a large pool of memory that can be sampled for training the network. This method was used to train an agent to play Atari video games.

This approach works well in practice and does not require any knowledge of the problem space aside from a reward function, but requires a large amount of computation as the use of epsilon greedy Q learning [23] results in the exploration of a large number of sub-optimal states before an efficient policy can be learned. Additionally, this method does not utilize demonstrated human behavior which can greatly reduce the required state exploration while learning a policy that more closely matches human decisions.

Guided policy search [12] is a method that has been applied to training visual control networks to represent policies for robotic arms [11]. In this setting trajectory-centric reinforcement learning [10, 13] is used to train a policy with full state information. This policy is used to complete the task creating a set of guiding trajectories that are used as training data. The network is then jointly trained with the guiding policy in order to elicit similar behavior in application from only visual input. A key limitation here is the requirement of training a guiding policy with full domain knowledge which is not possible in many complex tasks. Similar to deep Q learning, guided policy search requires a large amount of exploration in order to train a generalizable network [24].

Both of these methods have been extended to asynchronous frameworks where multiple parallel agents run in different settings. This allows for a greater amount of diverse data to be efficiently collected for training [18, 16, 24]. In the case of asynchronous deep Q learning [16], multiple agents (actors) are run in parallel (in separate environments) each collecting data from their own experiences. This data is then added to a central dataset that is used by a parallel learner thread to update the network parameters. Each actor thread intermittently receives a copy of the current network parameters from the learner thread to update its policy. This allows for the network to learn from a much larger distribution of experiences than standard deep Q learning. However, this increase in the size of the dataset can be problematic when considering high-dimensional image states requiring a large amount of disk space and an increase in network training complexity. Additionally, the efficiency of training the network may be reduced by repeatedly learning from redundant experiences the network has already mastered [20].

A deep active learning method for autonomous visual navigation [6] partially addresses the problem of increasingly large data aggregation by applying a modified version of the Dagger algorithm for training a convolutional network [9] to represent the agent policy for navigating a 3D environment. An agent explores a 3D space using a learned network policy. When the agent is uncertain of its next decision an expert policy is generated for the current state and used to control the network. The state and the action chosen by the expert are then added to an existing dataset for training. In this case, only states where the network policy is uncertain are added to the training set greatly reducing its size and reducing the cost of querying an expert policy. However, in practice autonomous agents frequently fall into states where they are very confident in taking an incorrect action. These states would not be added to the training set and the network would not learn from these mistakes preventing the agent from becoming fully capable in navigating the space.

We address the limitations of these approaches by applying concepts from active learning to the Dagger algorithm in an asynchronous actor-learner framework. We grow a training set from states where the agent makes mistakes relative to a training policy with local domain knowledge. This data is then used to train a neural network for end to end autonomous visual navigation through 3D mazes in Minecraft[1]. In the next sections we describe the details of the proposed method and analyze the results.

3. ASYNCHRONOUS DATA AGGREGATION

We begin by modifying the Dagger algorithm [19] to only collect data from states where the network disagrees with a training policy, π_t, that maps states to actions that move the agent to the closest state observed in human behavior. Figure 2 shows an aerial view of a discretized Minecraft maze with two walls each containing a single doorway. The start position is indicated by the green cell with the red cell signifying the goal. On the left, the yellow arrows show a demonstrated path through the maze. On the right, for each discrete state in the maze, π_t directs the agent to the nearest state observed in the demonstrated path. When the agent is in a state along the demonstrated path, the actions are set to match the human

[1]https://Minecraft.net/

behavior.

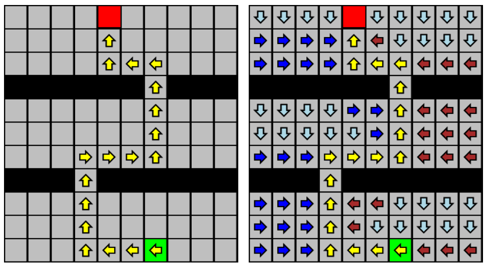

Figure 2: Example of a demonstrated (human) path through a maze (left) and the resulting discrete training policy (right).

There is no constraint on the training policy being discrete or continuous. This is an example of a simple discrete imitation policy that only requires local knowledge of the agents position relative to the nearest demonstrated state and nearby obstacles, however we show in section 5 that this is sufficient for training visual control networks for continuous agents from a single demonstrated path.

An important factor to note is that the policy in Figure 2 only requires local domain knowledge and a single demonstration for each maze. Given a demonstrated trajectory, $\tilde{\xi}$, and the current agent state, s, then training policy can be formally defined as,

$$\pi_t(s) = \begin{cases} \text{demonstrated action in } \tilde{\xi} \text{ at } s, & s \in \tilde{\xi}, \\ \text{optimal action towards } \tilde{\xi}, & \text{else} \end{cases}.$$

This policy, combined with the following algorithm, allows us to train robust network policies for navigating non-deterministic environments solely from visual input.

Algorithm 1 MinDagger: Minimal Dataset Aggregation

```
Input: Training policy $\pi_t$, network policy $\pi_n$, weighted control parameter $\alpha$, environment $\mathcal{E}$, and dataset $\mathcal{D}$.
Output: Aggregate dataset $\mathcal{D}$
1: Sample trajectory $\xi$ in $\mathcal{E}$ via policy $\alpha\pi_t + (1-\alpha)\pi_n$.
2: Form dataset $\mathcal{D}_n$ of states in $\xi$ where $\pi_t(s) \neq \pi_n(s)$.
3: Label $\mathcal{D}_n$ with $\pi_t$
4: Aggregate datasets: $\mathcal{D} \leftarrow \mathcal{D} \bigcup \mathcal{D}_n$.
return $\mathcal{D}$
```

We apply this training policy in Algorithm 1 where an agent runs through a maze with a weighted average of the training policy and the learned network policy, $\alpha\pi_t + (1-\alpha)\pi_n$, with α decreasing after each run. We then collect the states where the network policy disagreed with the training policy and append them to the current training set, $\mathcal{D}$, with labels generated from the training policy. This allows for the network training set to be formed with minimal redundancy as states the network performs well on will be ignored. Additionally, the network is gradually given more control allowing for the agent to learn from states outside the realm of the demonstrated data.

For tasks with a large amount of state variability (such as visual navigation with randomly changing mazes) it is necessary for Algorithm 1 to collect a large dataset for training. This can be time consuming as the agent would need to run through each maze sequentially with intermittent network training. We address this inefficiency by applying Algorithm 1 in an asynchronous actor-learner framework.

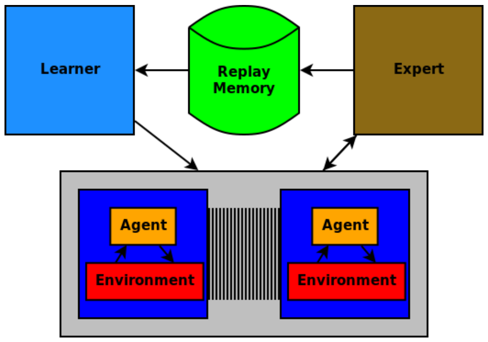

Figure 3: Asynchronous MinDagger

Figure 3 outlines the basic architecture of the asynchronous minimal data aggregation algorithm (*MinDagger*). A set of parallel agent nodes (bottom) sample the current network parameters from the learner node (top left) and run Algorithm 1 on a randomly selected maze environment with a matching training policy (top right). Each agent node will then append its set of collected states, labeled by the training policy, to the central dataset (top center) before restarting Algorithm 1 with a new maze and a newly sampled set of network parameters from the learner node. In parallel, the learner node continuously samples data from the central dataset and updates the network parameters to match the network policy to the training policy.

4. SOLVING MAZES IN MINECRAFT

We apply the proposed method to the task of solving 3D mazes in Minecraft using the Malmo platform for artificial intelligence experimentation [8]. Malmo is lightweight and allows for material customization. This enables the construction of vastly different obstacle and environment types where agents with egocentric vision respond to continuous action commands. Different block types offer different dynamics when an agent moves across them. This coupled with an asynchronous communication system that exhibits inconsistent execution and observation rates, leads to a non-deterministic agent environment.

For our experiments we consider four different action classes (forward, reverse, left, and right) paired with a $[0,1]$ continuous value for speed. A neural network is trained to map an agents visual observations to actions via supervised training. A softmax distribution is calculated over the network outputs and the action with the maximum value is chosen at each state, with the softmax value of that action assigned to the speed. This allows for the agent to move faster in each direction when it is more confident in its decision. The agent navigates through each 3D maze from some initial position to a goal position signified by a red block. The block types used in these mazes can vary and the agent needs to navigate to the goal in under 30 seconds through doorways and bridges across lava or water as seen in Figures 1 and 2. Running out of time before the goal is reached kills the agent and ends the mission as does touching lava or water.

4.1 Policy Network

Each state is represented by a 3-channel 120x160 RGB image generated from the agents point of view (Figure 1) which is vertically cropped to the 3x60x160 bottom half of the image (below the horizon [2]). The network is trained to maximize the negative log likelihood optimization function to match the predicted action, Y, to the action chosen by the training policy for that state, y_i,

$$L(\theta, \mathcal{D}) = -\sum_{i=1}^{|\mathcal{D}|} \ln p(Y = y_i | x_i, \theta),$$

given network parameters θ, training data $\mathcal{D}$ and input x_i.

The network is initially formed by five convolutional layers each with a 5x5 kernel and a 2x2 stride followed by a single convolutional layer with a 2x5 kernel. Strided convolutions incorporate regularization into the convolutional layers while improving efficiency in network performance [21] when compared to standard max-pooling based sub-sampling, which is important for real-time visual navigation networks [2]. The final convolutional layer reduces each feature map to a single neuron to improve generalization while reducing the number of parameters needed to train [21]. The number of feature maps in each of these layers are 16-32-32-64-64-128 ending with a 128x1x1 set of feature maps. We follow the convolutional layers with a fully connected layer of 64 neurons and another fully connected layer mapping the output to 4 neurons each representing a different action. Figure 4 outlines the network architecture [2].

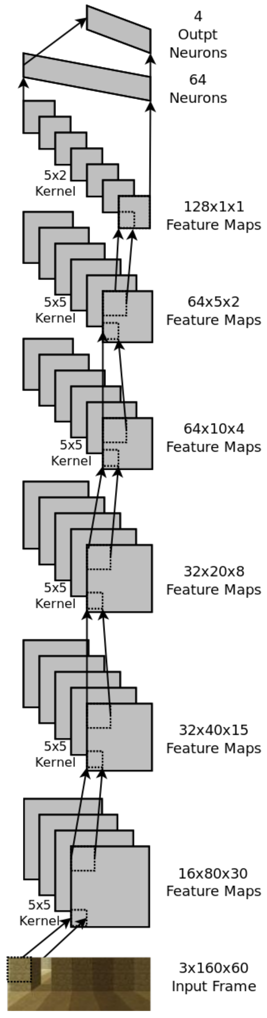

Figure 4: Diagram of the convolutional network architecture.

[2]Exponentiated Linear Units (ELU) [3] are used as the activation functions in each layer.

In order to improve the efficiency of our training routine we use spatial batch normalization [7] to normalize the feature maps generated after each convolutional layer. This ensures that the input distribution for each layer is consistent (zero mean and unit variance) greatly improving learning performance. Moreover, we apply spatial batch normalization to each incoming frame to normalize the input features of the network as well as spatial dropout [22] for regularization.

5. RESULTS

In this section we compare the results of training end to end neural networks for navigating 3D mazes in Minecraft from raw visual input with the proposed asynchronous MinDagger algorithm to a set of competitive baselines for data aggregation. We apply each comparison algorithm in the same asynchronous learning system as described in Section 3, ensuring a fair evaluation.

We compare our results in two settings. The first is the network being trained with a set of 20 static mazes with varying block types and obstacles each with a single demonstrated path through the maze. We then test our results on a set of 10 mazes with similar obstacles and block types.

The second setting is similar to the first setting with the key distinction that the block types of the mazes are now randomly changing on every run through the mazes. This greatly expands the variety of input features the network is learning making for a far more difficult task. As in training, the test mazes also contain randomly changing block types allowing for us to incorporate generalization into our evaluation.

We initially implemented the deep Q learning algorithm [17] (DQN) with intermediary sub-goals (doorways, bridges, etc.) granting positive rewards. Unfortunately, it failed to achieve meaningful results with fewer than 100 runs through each maze which is not surprising given an epsilon greedy Q learning algorithm [23]. For this reason, we decided to focus on the following dataset aggregation methods for comparison.

For Figures 5, 6 and 8, we train a single network to navigate 20 different mazes and average the results after running through each maze (1 mission run) and display the standard deviation from the mean.

Dagger

The standard Dagger algorithm for data aggregation [19] (*Dagger*). This method is similar to Algorithm 1 except every state in a sampled path through the maze is collected and labeled for training.

Deep Active Dagger

The deep active learning algorithm for autonomous navigation [6] (*Deep Active Dagger*). In this method the network policy is used to sample a trajectory through the maze. When the entropy of the network policy, $H(s) = -\sum_i P(\pi_n(s)) log P(\pi_n(s))$, for the current state s is above some threshold, $\beta = 0.9$[3], the training policy decides the next action and the current state is added to the dataset. Essentially, the training set is grown from states where the network is uncertain of a decision.

Expert MinDagger

We additionally compare our results with two versions of Algorithm 1. In the first (*Expert MinDagger*) we set $\alpha = 1$ and never allow for the network to take control of the agent. In non-deterministic

[3]Found via a grid search on the hyper-parameter space.

environments this generates a large set of varied guiding trajectories from the training policy which are appended to the dataset when the network policy disagrees.

MinDagger

Finally we compare against the proposed MinDagger algorithm for collecting a minimally optimal training set for visual navigation. We gradually give control of the agent to the network policy by setting α to $\alpha = 0.975^{t3}$ before each agent begins a maze, where t is the number of times the agent has run through a maze.

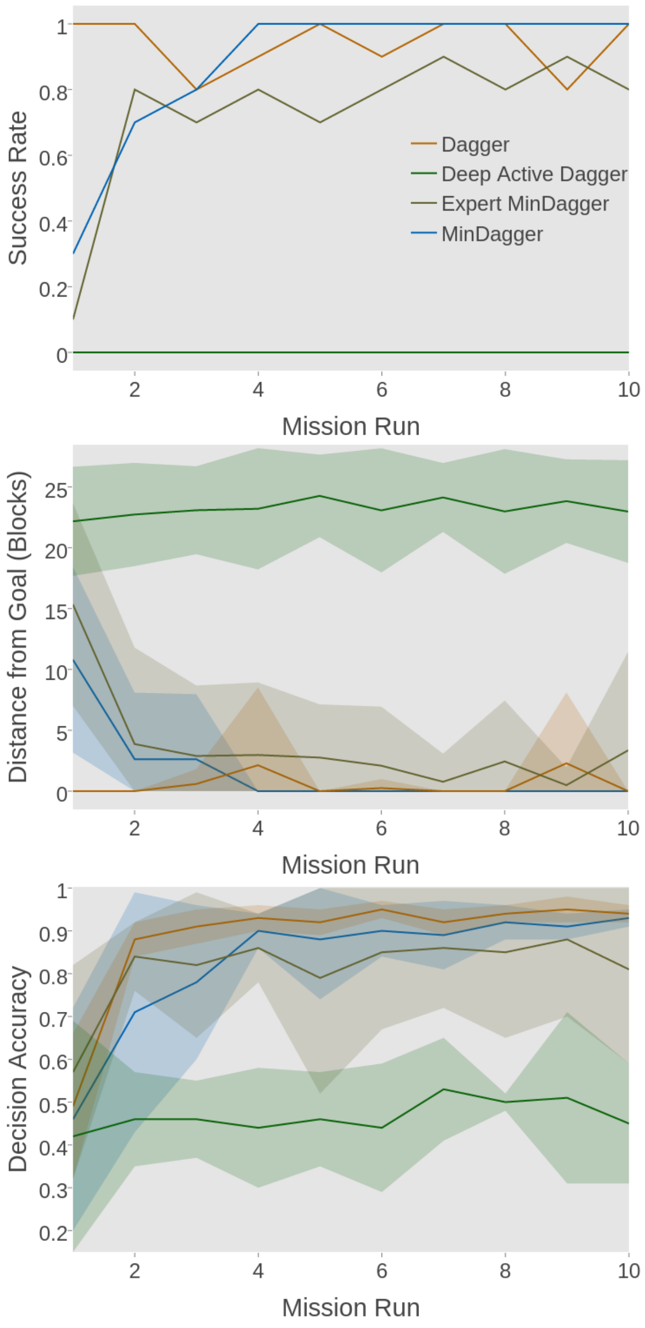

Figure 5: The average success rate (top), distance the agent is from the goal at the end of each mission (middle), and the average decision accuracy (bottom) compared to the training policy for each run through the static set of missions.

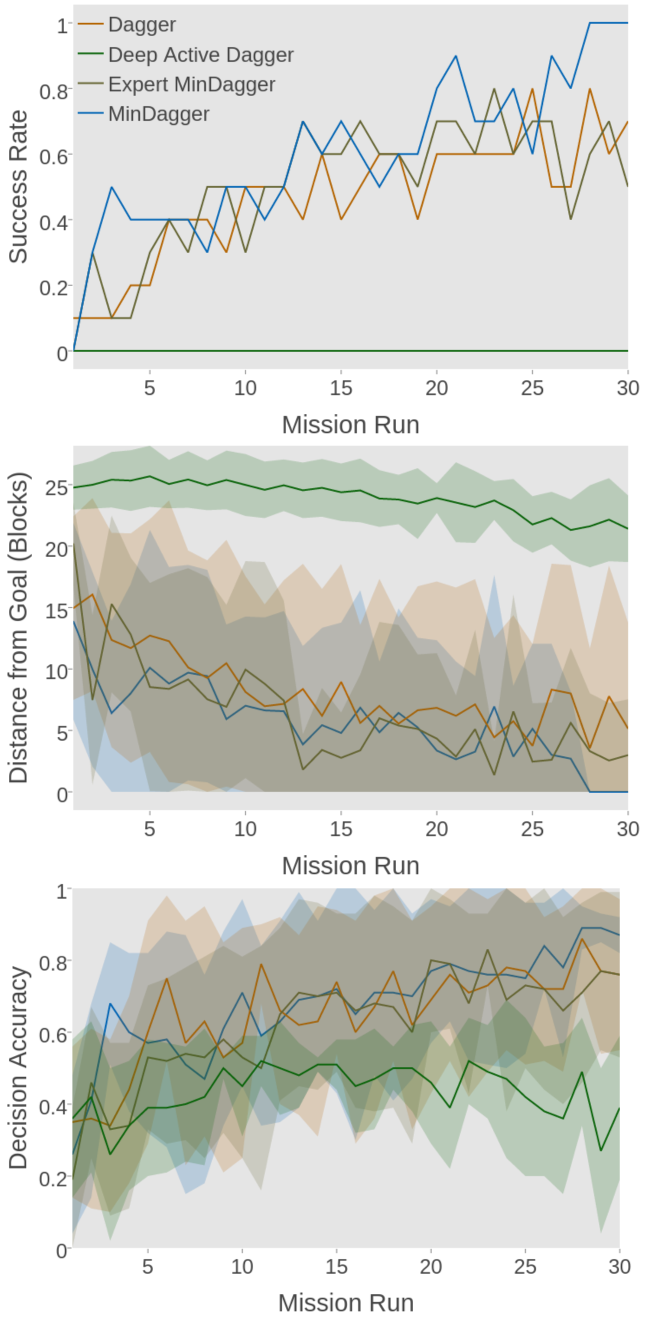

Figure 6: The average success rate (top), distance the agent is from the goal at the end of each mission (middle), and the average decision accuracy (bottom) compared to the training policy for each run through the random set of missions.

5.1 Mazes with Static Block Types

Figure 5 displays the results of the network policy running through the static test mazes after being trained by data aggregated from a set of static training mazes using the four different baseline methods. We store the current iteration of the network policy after each of the training missions and run it through the test mazes for evaluation. The figure shows the evaluated results of each method in terms of the average number of times the agent reached the goal (success rate), the average distance to the goal at the end of each mission due to reaching the goal, running out of time, or dying (distance to goal), and the average percent of action choices taken by the agent where the network policy agreed with the training policy (decision accuracy).

It is clear from Figure 5 that the deep active dagger algorithm is failing to learn a sufficient policy even after 10 runs through each training maze. The restriction of only collecting data when the entropy of the network policy is above some threshold fails to gather an effective amount of training data. Lowering the threshold does not improve the results until it is lowered to a value where the entropy becomes redundant and the algorithm is collecting almost every state.

In Figure 5, the proposed method (MinDagger) outperforms the standard dagger algorithm while only collecting about 17% of the data the standard dagger algorithm collects. This is evidence that MinDagger collects a very effective dataset. Furthermore, the improvement of MinDagger over Expert MinDagger shows that allowing the network to gain control of the agent is beneficial in training a sufficient network policy.

5.2 Mazes with Random Block Types

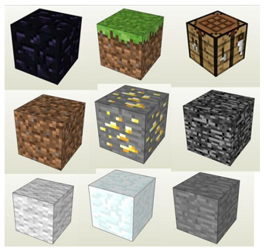

Figure 7: Example of random block types.

Figure 6 displays the results of the network policy running through the test mazes with random block types after being trained on a set of mazes with randomly changing blocks. The figure shows the evaluated results of each method in terms of the average number of times the agent reached the goal (success rate), the average distance to the goal at the end of each mission due to reaching the goal, running out of time, or dying (distance to goal), and the average percent of action choices taken by the agent where the network policy agreed with the training policy (decision accuracy).

The proposed method (MinDagger) achieves the highest success rate by reaching the goal in each of the test mazes after 28 iterations through the training mazes. None of the other methods manage to train a network to reach the goal in every test maze in under 30 iterations.

This becomes more significant when viewing Figure 8 where we plot the evaluation results of the top three performing methods with respect to the number of states collected for training. We omit the deep active learning method from this evaluation as it failed to collect much data (fewer than 1000 states in 30 runs) and performed poorly as seen in Figure 6. In this graph it is clear that the proposed MinDagger algorithm performs very strongly with far less training data (about 60,000 states) than the standard dagger algorithm which required more than 200,000 states before it began to return strong results. This is significant as it shows MinDagger collects a smaller more valuable dataset for training.

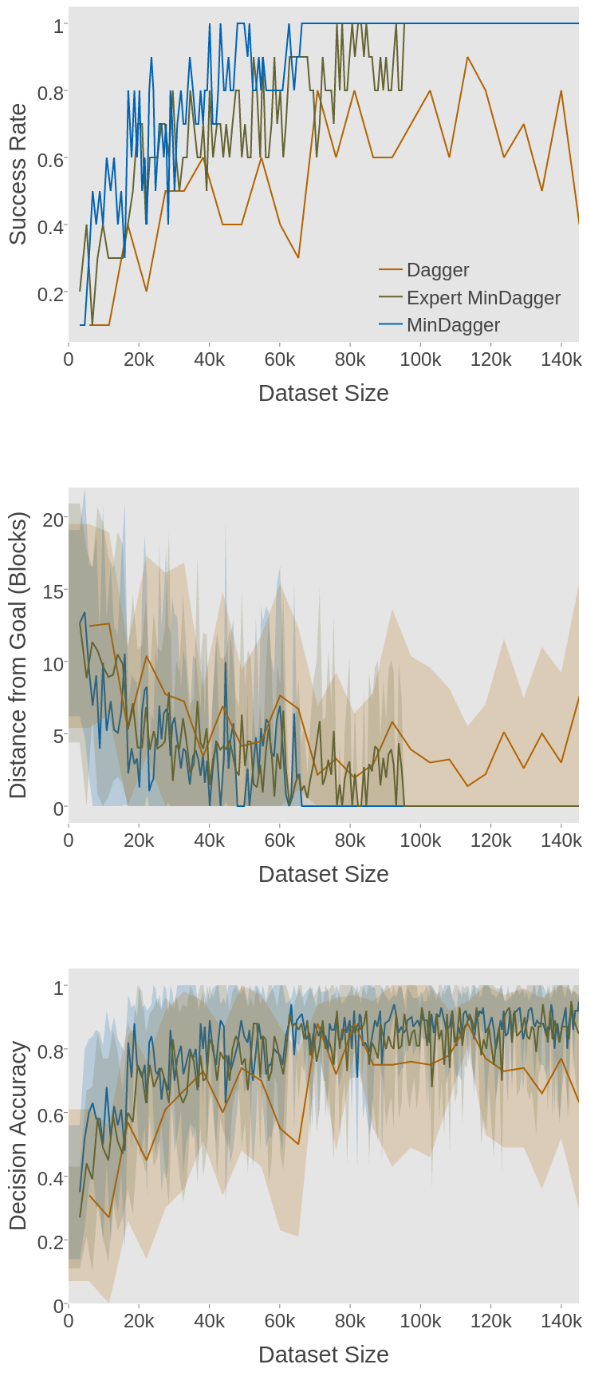

Figure 8: The average success rate (top), distance the agent is from the goal at the end of each mission (middle), and the average decision accuracy (bottom) compared to the training policy as a function of the number of states in the dataset for the set of random missions.

Additionally, the fact that MinDagger is able to solve the mazes while collecting less than 30% of the total states visited gives credence to the trained networks ability to generalize to unseen environments. There are 69 different block types that are randomly sampled at each iteration and the chance of the network seeing the same maze twice is very small.

A key limitation to the proposed approach is the reliance on a consistently accurate training policy dependent on the demonstrated trajectories matching the maze environment. This is prevalent in cases when the dynamics deviate strongly from those experienced by the demonstrations, due either to the communication delay in Minecraft or the properties of the random block material in the environment. Additionally, the use of random blocks can lead to the creation of environments that have distinct visual features that the network rarely encounters and thus poorly learns. Fortunately, these cases rarely occur and do not greatly hinder the network from learning a robust policy.

5.3 Emergent Network Features

Visualizing the learned feature maps of the convolutional network allows us to examine which emergent environment features it may have learned. The network was not trained to detect objects or features, only predict actions. However, in order to predict the training policy the network learned some important features of the mazes.

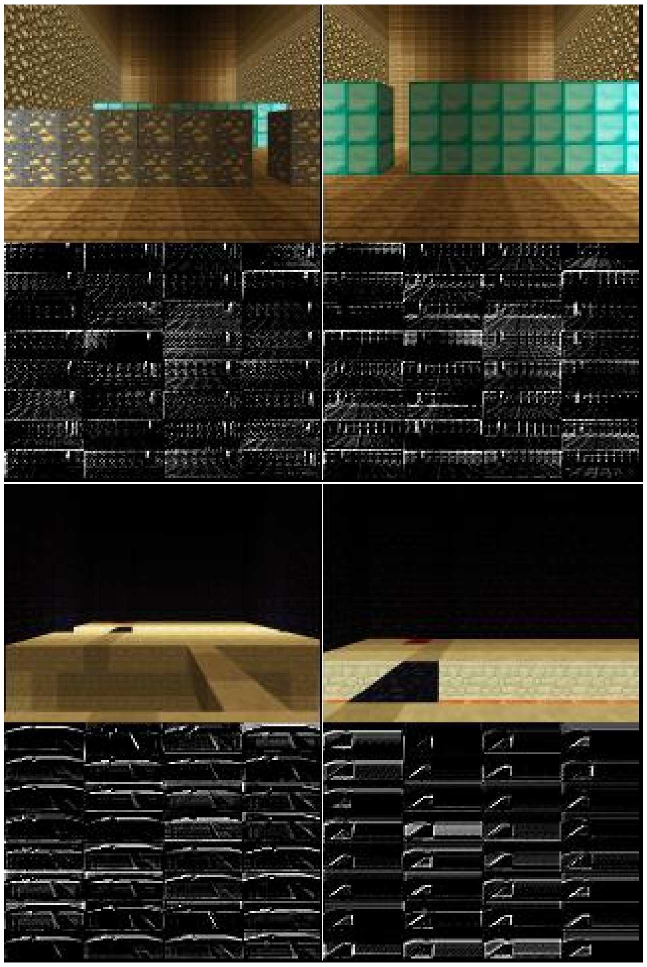

Figure 9: Example of learned emergent features.

Figure 9 displays four different images the agent encountered while running through the test mazes along with the output of the networks second convolutional layer. In each state the network is recognizing key information about the scene that will help it determine the next action. When a door or a bridge is to the left or right of the agent we see a clear activation of neurons in similar locations in the feature maps. The agent has learned to identify and localize doorways and bridges while ignoring excess information.

6. DISCUSSION AND FUTURE WORK

We have presented an asynchronous framework for efficiently gathering an efficient training set for end to end autonomous visual navigation from single shot examples (one demonstration per maze). We applied our method and multiple baselines to the task of completing 3D mazes in Minecraft from raw images with both static and randomly changing block types. Our method outperforms the standard methods of data aggregation while collecting a much smaller dataset, and trains a network policy from single shot examples.

A key limitation of the proposed approach lies in the requirement of a locally informed policy for training. In the presented experiments we formed this policy using the coordinate positions of the agents. This may not be available in all problem settings and can be expensive to generate. In the future, we plan to extend this method to domains without access to this information. In addition, we plan to incorporate recurrent network structures [5] in order to capture the memory of key features (doorway) that may move out of frame with an agent's motion. Lastly, we would like to apply the proposed method to different problem settings altogether such as first-person-shooters and autonomous driving.

Acknowledgments

Partly support for this research was provided by the Toyota Research Institute / MIT CSAIL Joint Research Center and part of this work was done while Mathew Monfort was a research intern at Microsoft Research Cambridge.

REFERENCES

[1] Y. Bengio, J. Louradour, R. Collobert, and J. Weston. Curriculum learning. In *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09, pages 41–48, New York, NY, USA, 2009. ACM.

[2] M. Bojarski, D. D. Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang, X. Zhang, J. Zhao, and K. Zieba. End to end learning for self-driving cars. *CoRR*, abs/1604.07316, 2016.

[3] D. Clevert, T. Unterthiner, and S. Hochreiter. Fast and accurate deep network learning by exponential linear units (elus). *CoRR*, abs/1511.07289, 2015.

[4] A. Dragan, K. Lee, and S. Srinivasa. Legibility and predictability of robot motion. In *hri*, 2013.

[5] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, Nov. 1997.

[6] A. Hussein, M. M. Gaber, and E. Elyan. *Deep Active Learning for Autonomous Navigation*, pages 3–17. Springer International Publishing, Cham, 2016.

[7] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *CoRR*, abs/1502.03167, 2015.

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

[9] Y. LeCun and Y. Bengio. Convolutional networks for images, speech, and time series. In M. A. Arbib, editor, *The*

Handbook of Brain Theory and Neural Networks, pages 255–258. MIT Press, Cambridge, MA, USA, 1998.

[10] S. Levine and P. Abbeel. Learning neural network policies with guided policy search under unknown dynamics. In *Advances in Neural Information Processing Systems*, pages 1071–1079, 2014.

[11] S. Levine, C. Finn, T. Darrell, and P. Abbeel. End-to-end training of deep visuomotor policies. *J. Mach. Learn. Res.*, 17(1):1334–1373, Jan. 2016.

[12] S. Levine and V. Koltun. Guided policy search. In *ICML '13: Proceedings of the 30th International Conference on Machine Learning*, 2013.

[13] S. Levine, N. Wagener, and P. Abbeel. Learning contact-rich manipulation skills with guided policy search. In *2015 IEEE international conference on robotics and automation (ICRA)*, pages 156–163. IEEE, 2015.

[14] L.-J. Lin. *Reinforcement Learning for Robots Using Neural Networks*. PhD thesis, Carnegie Mellon University, Pittsburgh, PA, USA, 1992. UMI Order No. GAX93-22750.

[15] D.-R. Liu, H.-L. Li, and D. Wang. Feature selection and feature learning for high-dimensional batch reinforcement learning: A survey. *International Journal of Automation and Computing*, 12(3):229–242, 2015.

[16] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. P. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu. Asynchronous methods for deep reinforcement learning. *CoRR*, abs/1602.01783, 2016.

[17] 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.

[18] A. Nair, P. Srinivasan, S. Blackwell, C. Alcicek, R. Fearon, A. D. Maria, V. Panneershelvam, M. Suleyman, C. Beattie, S. Petersen, S. Legg, V. Mnih, K. Kavukcuoglu, and D. Silver. Massively parallel methods for deep reinforcement learning. *CoRR*, abs/1507.04296, 2015.

[19] S. Ross, G. J. Gordon, and J. A. Bagnell. No-regret reductions for imitation learning and structured prediction. In *In AISTATS*, 2011.

[20] D. Silver, J. A. D. Bagnell, and A. T. Stentz . Active learning from demonstration for robust autonomous navigation. In *IEEE Conference on Robotics and Automation*, May 2012.

[21] J. T. Springenberg, A. Dosovitskiy, T. Brox, and M. A. Riedmiller. Striving for simplicity: The all convolutional net. *CoRR*, abs/1412.6806, 2014.

[22] J. Tompson, R. Goroshin, A. Jain, Y. LeCun, and C. Bregler. Efficient object localization using convolutional networks. *CoRR*, abs/1411.4280, 2014.

[23] C. J. Watkins and P. Dayan. Q-learning. *Machine learning*, 8(3):279–292, 1992.

[24] A. Yahya, A. Li, M. Kalakrishnan, Y. Chebotar, and S. Levine. Collective robot reinforcement learning with distributed asynchronous guided policy search. *CoRR*, abs/1610.00673, 2016.

[25] X. Zhu, C. Vondrick, C. C. Fowlkes, and D. Ramanan. Do we need more training data? *International Journal of Computer Vision*, 119(1):76–92, 2016.