Utilising Prior Knowledge for Visual Navigation: Distil and Adapt

M. Mahdi Kazemi Moghaddam, Qi Wu, Ehsan Abbasnejad, and Javen Shi

1 The Australian Institute for Machine Learning
2 The University of Adelaide

Abstract. We, as humans, can impeccably navigate to localise a target object, even in an unseen environment. We argue that this impressive ability is largely due to incorporation of prior knowledge (or experience) and visual cues—current visual navigation approaches lack. In this paper, we propose to use externally learned prior knowledge of object relations, which is integrated to our model via constructing a neural graph. To combine appropriate assessment of the states and the prior (knowledge), we propose to decompose the value function in the actor-critic reinforcement learning algorithm and incorporate the prior in the critic in a novel way that reduces the model complexity and improves model generalisation. Our approach outperforms the current state-of-the-art in AI2THOR visual navigation dataset.

Keywords: Visual Navigation, Knowledge Graph, GTN, AI2THOR

1 Introduction

We, human beings, are capable of finding an object in an unexplored environment, e.g. a room in a new house. We primarily rely on our prior knowledge, in addition to other sensory information. For example, we know in any bathroom, bar soap or liquid hand wash is probably near the basin, thus observing one helps finding the other. Our belief also needs to be adjusted upon observations. For example, bar soap or liquid hand wash might be misplaced. It is desirable to develop a navigation robot or agent that can utilise the prior knowledge while being prepared for updating beliefs and adapting to a new environment.

Most current visual navigation approaches use either supervised learning or reinforcement learning (RL) to learn the visual associations. That is, during training the agent explores the environment to seek the optimal mapping from (primarily) the ego-centric observational inputs to series of actions. There are two main issues with this approach. Firstly no prior knowledge about the environment is provided and used by the navigating agent which significantly limits its use and generalisability. Secondly, the agent learns everything from scratch (i.e. forgets everything and has to re-learn) when it encounters a new environment, hence limiting the reusability.

To address the first issue, recent works [18][19][38] used graph neural networks, as the most natural representation of the prior knowledge, to encode the object-object relationships in a pre-trained knowledge graph. On the other hand, to
Motivated by humans’ navigation system, our agent is able to benefit from beliefs on object relationships, stored in its knowledge graph, while updating them in order to navigate towards a given object, e.g. TV, for example.

handle the second issue, there are works [35] that use meta-learning to allow the agent quickly adapt to a new environment. They have shown success in handling the train-test distribution shift while being more sample efficient. Their efficiency in adapting external knowledge to unobserved scenes, however, has remained unexplored.

While these recent advances address parts of the problem, the main issue of incorporating prior graph (e.g. semantic graph of relationships between objects) to a navigating agent using reinforcement learning (RL) persists. In addition, the semantic prior graph has to be grounded to the visual cues in RL and updated when presented with a new environment.

To that end, we propose an approach to efficiently benefit from external knowledge while dynamically updating it as our agent observes new scenes. The external knowledge presents the agent with general rules about the semantic relationships of the objects in indoor environments. Thus, our graph contains as the nodes, both name of the objects and scene representations and as the edges, the existence of a relation between a pair of objects, hence capturing the correlation between the semantics of the scene and the agent’s ego-centric view. We further use Graph Transformer Networks (GTN) [39] to learn a representation for that graph to be subsequently used in our agent’s decisions. GTN, contrary to its counterparts such as Graph Convolutional Networks (GCN) [15], allows for heterogeneous objects and generating new structures—corresponding to new connections that are not in the initial graph. This is particularly useful in our case when an agent may encounter objects with relations different to that of the prior, for instance, while dish-washing soap and hand-wash may be semantically very close, they are typically in different locations in an indoor environment (e.g. bathroom and kitchen that can be far). As such, the desired graph solution has to be able to learn these connections from the data, rendering GTN the better choice.

To incorporate the graph into the RL algorithm, we further found that simply conditioning the RL’s policy on the prior graph does not lead to better perfor-
To remedy the issue, we first noticed that, intuitively, the prior graph should be used to guide the RL training, rather than providing a signal for each individual action. Secondly, the success of the agent, which is reflected in its expected (accumulated) reward, is partially due to a proper prior rather than the current state alone. As such, we devise an approach to decompose the reward to account for both the state (as is the convention) in addition to the prior graph’s contribution. This, moreover, enables the policy to distil the prior’s knowledge rather than seeking to exploit all the details that might not necessarily be related to its navigation decisions.

This further entails when we use a variant of an actor-critic RL algorithm (e.g. Asynchronous Advantage Actor-Critic; A3C [23] in our case), where the critic’s role is divided between its evaluation of the current state and the prior graph. We show that reduces the variance of the gradients leading to a guided learning that improves the performance. Finally, we employ Model Agnostic Meta-Learning (MAML) [9] to enable test time adaptation of the prior and the policy. All in all, this leads to a principled and modular approach that can be employed in conjunction with other approaches for navigation. We show the combination of these three components lead to the state-of-the-art results in AI2THOR dataset.

In summary, our main contributions are:

– For the first time, we introduce a method to distil and adapt prior knowledge for RL-based visual navigation;
– We theoretically prove, and empirically demonstrate, how to efficiently inject prior for value estimation in actor-critic RL models which leads to lower variance and higher performance;
– Finally, our proposed method outperforms the existing state-of-the-art on AI2THOR public navigation dataset in all four evaluation metrics.

2 Related Work

2.1 Visual Navigation

Classical approaches to robotic navigation mainly divide the problem into localisation, mapping and planning [5, 21, 24]. Besides the computational complexity issues, those approaches lack the semantic scene understanding [6]. Semantic understanding is, especially, important for real-world navigation scenarios. For example, in scenarios where the robot is asked to navigate to an object [11, 42] or an outdoor location [12, 22]. End-to-end visual navigation has recently been extensively studied [1, 23] and many novel tasks have been introduced. The main approaches can be divided into supervised (Imitation Learning) [2, 3] and unsupervised (RL) [23]. The target is also given in different modalities. Some tasks consider target images [11, 12] while most others consider language instructions [1, 3, 34, 35, 38]. Providing the target as an image simplifies the task by introducing similarity measure options between observation and target. Multimodality, however, increases the challenges. This is mainly because the agent has...
to ground language instructions on observations while performing planning and navigation. This is further complicated where the agent has to learn through exploration, e.g. RL. There have been a few valuable simulators released recently for various visual (or vision and language) navigation tasks [3, 16, 29, 36]. AI2THOR, among them, is of especial interest to us. This is mainly due to its high quality near photo-realistic design and continuous state space. The latter renders the task specifically more challenging compared to other environments like [3]. The main reason is that rather than traversing graph nodes in the environment, the agent is placed in a near-real-world setup with a hugely expanded state space. In this environment, it’s very likely for a sub-optimal agent to stand in front of a blocked path, e.g. an obstacle, and continuously perform a failing action, e.g. move forward, until the maximum step limit is exhausted. In our task, we use RL to train an agent to navigate to target objects given object names as language instruction.

2.2 Knowledge Graph

Graph Neural Networks have recently been applied to different supervised and semi-supervised tasks successfully [15, 31, 39, 41]. While most of the tasks are classification of structured data, such as citation graphs, they’ve also been used to represent structured knowledge and reasoning [37, 40]. Graph neural networks have also been used in RL-based navigation tasks to represent topological environment maps [20] and help more efficient exploration [7]. In [20] the author use the graph to localise the agent in the environment. Generating and incorporating scene graphs [10] is also closely related to our problem. However, here we construct our knowledge graph externally and refine the node relationships without explicit object detection. This also separates our approach from [26], where an off-the-shelf object detector is used. These methods, while improving the performance, have been explored before and can be applied to any other approach for further improvements, including ours. A similar work to ours is proposed by Vijay et. al. [32] where the prior knowledge is injected to RL for navigation. In that work, the authors learn various edge features encoded as one-hot vectors. The authors apply their approach to a 2D fully-observable environment while in our case the environment is more challenging. The most relevant work to ours is proposed by Yang et. al. [38]. In that work, the authors use a similarly trained knowledge graph for navigation in a different scenario, where all the objects in graph are used as a target. This is to show the ability of agent to navigate to the objects not trained for. There are a few major differences between our approaches, however. Firstly, we have five layers of graphs, as discussed in 3.2. Secondly, our graph is adaptively used for value estimation rather than as observation input. Thirdly, our graph embeddings are fundamentally different, where ours is inspired by the latest Graph Transformer Networks [39].
2.3 Meta Learning

Generalisability of trained neural networks has always been a major challenge due to the gap of distribution between training and test environments. In some cases this gap is between simulation and real-world [27, 28]. In simulation the gap is narrower which makes it a decent feed for meta-learning approaches [9, 30]. In meta-learning the aim is to learn a loss that bridges the two distributions. In [9] the authors propose to learn an initialisation of the whole network for faster and more efficient adaptation using second order derivatives. Meta-learning has also been applied to RL in different scenarios [8, 33]. Of especial interest to us, is the recent work of Wortsman et al. [35] where the authors benefit from Model Agnostic Meta-Learning [9] and design a trainable self-adaptation loss.

3 Our Method

In this section, we first define the problem and then discuss our proposed method in more depth.

3.1 Problem Setup

Our proposed method is based on actor-critic methods in RL. The navigation task is divided into episodes. The overall episode scenario is as follows: the agent is randomly spawned in a position in one of the four available scenes for a randomly selected room type. There are many different scenes in each room type with their specific design and configuration. Then a randomly sampled target object, from among visible objects in the scene, is presented to the agent in plain language, e.g. "fridge" or "soap", for example. The only accessible observation to the agent is its egocentric RGB image at each time step. The agent has to take actions sampled from its policy based on the observation at each time step to find the target object. An episode ends if either the agent stops within a defined distance of an instance of the target object or the maximum number of actions is exhausted. In our RL-based method we define the problem as a Partially-Observable Markov Decision Problem (POMDP), tuple of \( \{X, A, r, \gamma\} \). Here \( \{X\} \) is the state space comprising of RGB observation images, the action space is \( A \), \( r \) is the reward and \( \gamma \) is the discount factor. This setup follows our main baseline [35].

Following the recent conventions in visual navigation tasks, we measure the performance of our method based on success rate and SPL; the former considers just the outcome while the latter measures the quality of navigation relative to the optimal trajectory, using the following formula:

\[
\frac{1}{N} \sum_{i=0}^{N} S_i \frac{O_i}{\max(O_i, L_i)},
\]

where \( S_i \) is a binary value for success, \( O_i \) is the optimal length of the \( i \)-th trajectory and \( L_i \) is the actual length traversed by the agent.

In the this setup, the agent is trained to maximise the accumulated expected return, \( \sum_{\tau \sim \pi} \mathbb{E}_{\tau \sim \pi} \left[ \sum_{t=1}^{T} \gamma^t r_t \right] \), where \( \tau \) is the trajectory and \( \pi \) is the agent’s policy. The policy is approximated by a neural network approximator, here a CNN-LSTM variant, \( \pi = f(x_t, Y; \theta, \theta_x, \theta_v) \); \( x_t \) is the state observation, \( Y \) is the
target vector and \([\theta, \theta_x, \theta_z]\) the network parameters, which are defined in more details in the following sections. More details on the network architecture are presented in section 4.2.

3.2 Our Knowledge Graph

Graph Structure
Our prior knowledge graph \(G(V, E)\) encodes the semantic and the correlation of objects in the scene. The set of nodes \(V\) includes features related to all the objects in the environment (e.g. whether used as navigation target or not). Each node feature, \(v_i \in \mathbb{R}^d\), encodes the concatenation of observation features \(x_i\) and the semantic vector embedding of the object. For the edges, \(e_{ij} = 1\) if and only if the concerned objects appear in the same egocentric view of the agent. All the edge weights are initialised as one, which means different distances are considered equal initially and just the co-occurrence is injected as prior.

Furthermore, different from [38], our adjacency matrix, \(A \in \mathbb{R}^{n \times n \times C}\), is a three-dimensional tensor where each channel \(C\) encodes the knowledge specific to a scene type. We also add a last channel of self-connections in practice to ensure the trivial relation is included. The graph separation enables the agent to mainly attend to one of the graph channels in each scene and avoid distraction. This way, more scene-specific knowledge can be encoded. Intuitively, the agent should be able to reason about the kitchen utensils different from the living room furniture. Despite the separation of channels, in our method we enable cross-channel reasoning which is necessary when objects are shared between scenes. To do so, we adopt the recently proposed approach by Yun et al. [39], GTN, that suits our purpose.

Prior Knowledge Initialisation

Inspired by [38], we also initialise our graph using the knowledge existing in the Visual Genome [17] dataset. The graph encodes the co-occurrence of objects as edges, if it is higher than a threshold frequency [38]. However, we consider this as a coarse initialisation for two main reasons: any prior knowledge obtained from external sources may not contain all the relevant information for the target environments we are interested in; what is referred to as the dataset bias. Additionally, we argue that even a knowledge graph built from the given training environments is unreliable. This is mainly because the objects visually change significantly in every new scene. Therefore, rather than relying on the graph we

Fig. 2. We acquire our knowledge graph from the relationships between objects of our environment in Visual Gnome dataset [17].
propose to adapt it dynamically starting from the prior by distilling it into the policy network parameters.

**Knowledge Learning and Adaptation** In order to overcome the distributional shift in different scenes, our method updates the prior knowledge and adapts it to the new environments. This shift is specifically a major challenge in our task where no perfect prior applicable to all the scene objects can be found. Therefore, we use the prior knowledge as just an initial belief of the new scenes and not a strict rule.

Using GTN [39], our agent is able to learn new edges and weights among all the nodes whether inside an adjacency layer or across different ones. This way, depending on the scene, we are able to extract features from the graph that help more efficient navigation. Furthermore, the graph structure along with the extracted features are adapted to the episode at hand. Formally, we have:

$$H_l^i = \text{softmax}(W_l^i A), \quad H_i = \prod H_l^i,$$

(1)

where we learn multiple normalised (softmax) weighted sums across the channels of adjacency matrix $A$, in $H_l^i$, where $l$ and $M$ are hyper-parameters. Here, $H_i$ learns the adjacency matrix as the result of matrix multiplication of $H_l^i$ s. We can learn up to $M$ different new adjacency matrices. In addition, using $\|$, we concatenate $M$ learnt graph representations using node feature extractor weights $G_\psi$, i.e.

$$Q = \|_{i=1}^M \sigma(\tilde{D}_i^{-1} \tilde{H}_i G_\psi(X)).$$

(2)

The input node feature matrix is $X \in \mathbb{R}^{n \times d}$. Also, $\tilde{H}_i = H_i + I$ the augmented $i$th adjacency matrix with self-connections, $\tilde{D}_i^{-1}$ is its inverse degree matrix for normalisation. Therefore, the output graph representation vector $Q$ is the result of both node and edge operations dynamically learnt during training, before being distilled and adapted by the policy. The edge operations on adjacency matrix allow us to go beyond prior knowledge incorporation to learning and adapting it, as we will discuss in the subsequent sections.

For the adaptation, we adopt (MAML) [9] to continuously adapt the learnt knowledge during test time. To do so, in our RL setup, we devide the training trajectories into meta-train $D$ and meta-validation $D'$ domains. Then, a loss function parameterised by $\phi$ is learnt to compensate for the domain shift during test. So, the overall optimisation objective for each training trajectory sample is:

$$\min_{\theta_{\text{total}}, \phi} \quad \min_{\theta_{\text{total}}, \phi} L_{\text{RL}}([\theta_{\text{total}}, \phi] - \alpha_{\text{meta}} \nabla_{\theta_{\text{total}}} L^D_d (\theta_{\text{total}}, D), D').$$

(3)

We define $\theta_{\text{total}} = [\theta, \theta_x, \theta_v, \psi]$ in this equation for readability. In addition, $\alpha$ is a learning rate hyper-parameter for adaptation relative to each parameter set in $\theta_{\text{total}}$.

### 3.3 Actor-Critic Reinforcement Learning for Visual Navigation

We employ a variant of actor-critic RL algorithm known as A3C as the RL core of our method to build upon. Even though we experiment around a sin-
ingle algorithm, we observe no specific requirement preventing our method from applicability to other actor-critic algorithms.

In our method, the actions are sampled from \( \pi = f(x_t, Y; \theta_\pi, \theta_v) \), where \( x_t \) is the RGB image observation at time \( t \), \( Y \) is the semantic embedding vector of the target object in the current episode, \( \theta \) is the set of parameters of the backbone (CNN-LSTM) embedding network, \( \theta_\pi \) is the parameters of the policy sub-network (a.k.a actor) and \( \theta_v \) is the set of value sub-network’s parameters (a.k.a critic).

It is proven that policy gradient methods have high variance in practice. Therefore, in actor-critic the gradient variance is reduced by using the bootstrapped estimates of the state-value function as the baseline. This estimate is provided using \( V = g(x_t, Y, G; \theta_v, \psi) \) where \( G \) is our graph neural network parameterised by \( \psi \). An overall demonstration of our method is found in Figure 3.3.

Conventionally, the policy and value functions share the network parameters, except for the last layer. However, in our proposed method, we augment the critic sub-network with our knowledge features extracted from our knowledge proposed graph. These features lead to a more accurate value estimate which then reduces the variance of the final policy updates. This is because the Advantage function at each state is defined as \( A(x_t) = r(a_t|x_t) + V(x_{t+1}) - V(x_t) \) and the policy is updated using the gradients from \( L_\pi = -\log(\pi(a_t|x_t)) \times A(x_t) - \beta \times H_t(\pi) \). \( H_t \) is the entropy and \( \beta \) its hyper-parameter to encourage exploration, which are not our concern at this point. Therefore, a more accurate value estimation reduces the variance in the gradients of the policy which we hypothesise can then improve the optimality of the learned policy. We empirically validate this hypothesis.

### 3.4 Value Estimation in Actor-Critic

In A3C, the variance of policy gradients are reduced by integrating estimates of state-value in advantage function. Unlike common practice, here we partially separate the critic head’s parameters \( \theta_v \) from the policy head’s \( \theta_\pi \) to improve the value estimation and the variance as a result. Intuitively, there’s a correlation between the objects present in the observation at current time-step \( x_t \) and the target object to navigate to. This correlation is shown in the estimated value of the current state, that is: \( V(x_t) = \mathbb{E}[\sum_t \gamma^t r_t] \), as the relationships defined by edges of our

![Fig. 3. Overview of our approach. Inspired by MAML [9], during training we learn an adaptation loss that can adapt the knowledge to unseen scenes.](image-url)
Therefore, in our method we regress the state-value function in according to:

$$V(x_t) = \mathbb{E}\left[ \sum_{t=0}^{T-1} \gamma^t r_t \right]$$

$$V(x_t) = W_1 G(x_t, X; \psi) + W_2 f(x_t, Y; \theta)$$

$W_1$ and $W_2$ are aggregation parameters of the two sub-networks, implemented as linear layers in practice. Theoretically, we observe this method as the decomposition of the reward (or return, e.g. expectation of cumulative future rewards) into two components: one, estimated by the main back-bone network parameterised with $\theta$ and the other component estimated by the relations between the semantic target, semantic correlation of the available objects in the scene and the correlation of those with the current observation $x_t$. This way we reduce the variance of gradients in our actor-critic algorithm as defined in the following loss function, per step:

$$L_{A3C}(a_t|x_t) = -\log \pi(a_t|x_t; \theta, \theta_\pi)(r_t(a_t|x_t) + (V(x_{t+1}) - V(x_t)))$$

It should be noted that we continue following the original A3C method by augmenting the above loss with entropy regulariser to encourage exploration.

## 4 Experiments

### 4.1 Experimental Setup

We use AI2THOR environment as our experimental framework. This simulator consists of photo-realistic indoor environments (e.g. houses) categorised into four different room types: kitchen, bedroom, bathroom and living room. In order for fair comparison, we follow the same setup as SAVN [35]. In this setup, 20 scenes of each room type are used for training; 5 scenes for each as validation and 5 for test. We train all our methods until convergence with the maximum seven million episodes, whichever occurs first. The target objects for each scene are listed as follows: kitchen: toaster, microwave, fridge, coffee maker, garbage can, box and bowl; living room: pillow, laptop, TV, garbage can, box and bowl; bedroom: plant, book, lamp and alarm clock; and bathroom: sink, toilet paper, soap bottle and light switch, totalling 23. All the objects available in the dataset are 89 that are included in the graph. The objects are chosen to be small enough not be easily seen from distance without exploration; The same target objects list is shared between training, validation and test but the scenes are unique. In order to train our model, we use Pytorch framework. We use SGD for adaptation optimizer and Adam [14] otherwise. The loss function of our A3C algorithm is same as the original approach. For the reward, we use 5 for reaching the target and -0.01 for each single step, limited to 50.
Table 1. Comparison of our results with the baselines. Our approach improves all the baselines in all the four evaluation metrics, conventionally used in previous SOTA.

| Method | SPL | Success | SPL >5 | Success >5 |
|--------|-----|---------|--------|------------|
| Random | 3.64 | 8.01 | 0.1 | 0.28 |
| A3C    | 14.68 | 33.04 | 11.69 | 21.44 |
| GCN    | 15.47 | 35.13 | 11.37 | 22.25 |
| SAVN   | 16.15 | 40.86 | 13.91 | 28.70 |
| Ours   | **17.27** | **43.8** | **15.39** | **33.68** |

4.2 Implementation Details

We extract the observation features $x_t$ using a pre-trained ResNet-18 at each time-step. For computational efficiency, these features are extracted and saved once for later use. We use Glove [13] to generate 300-dimensional semantic embeddings for the target as well as graph objects. Therefore, the input to our actor-critic network is the concatenation of target object and the observation features, as a 1024-dimensional feature vector.

Our actor-critic network comprises of a LSTM with 512 hidden states and two fully-connected layers one for actor and the other for critic. The actor outputs a 6-dimensional distribution $\pi(a_t|x_t)$ over actions using a Softmax while the critic estimates a single value. As mentioned before, another novelty of our approach is an unconventional value estimation network. In this network, the hidden state of the LSTM is concatenated with a 512-dimensional representation vector extracted from the graph.

The input to the graph, as node features, is a 1024-dimensional vector. This vector is a concatenation of 512 observation features with 512 Glove embeddings of the objects in the simulator. The Glove embeddings are mapped from 300 to 512 using linear layers. There are 89 nodes in each layer of the graph’s adjacency matrix and 5 layers in total; 4 layers dedicated to the edges between objects in each scene and one self-connections layer for regularisation. We learn a two layer adjacency matrix using GTN.

4.3 Baseline Comparison

In order to better show the contribution and necessity of each component to our final method, we first compare it with a few different baselines (shown in Table 1). First, the previous state-of-the-art introduced by Wortsman et. al. [35], abbreviated as SAVN. Second, a similar work proposed by Yang et. al. [38], where the authors use a fixed knowledge graph structure, abbreviated as GCN. In SAVN, the authors use MAML [9] to learn a loss function $L_{int}$, approximated by an instance of Temporal Convolutional Networks [4], over training episodes. The learnt loss is then used during validation/ test to produce gradients to update the network weights according to the task at hand. We share the general adaptation framework while performing it quite differently on our knowledge graph. In GCN [38], the authors use a single graph for all the objects existing in the dataset. The embedding output of the graph is concatenated to the target
Fig. 4. Qualitative comparison. Our approach improves both effective trajectory length (SPL) and success rate. Trajectories steps are randomly sub-sampled for visualisation purposes.

vector embedding and observation features comprising the input to the policy network. We conjugate, and experimentally prove, that this approach increases the difficulty of policy optimisation by increasing the observed state space. In our approach, we benefit from the graph information more efficiently using our proposed value estimation method. In addition to that, our graph structure and embedding architecture is also different. Additionally, to further show the capabilities of our method, we compare our results with trivial methods. One methods is a random agent for which the policy is to uniformly sample an action at all times. Another relatively trivial baseline is named A3C. This method is the result of removing the effect of knowledge graph as well as adaptation framework. Therefore, it acts as the simplest RL-based agent. We present the new state-of-the-art results on AI2THOR public visual navigation dataset achieved using our method. All the baselines are also trained in the same setting for fair comparisons.

4.4 Ablation Study

In this section we seek to answer a few principal questions with regards to our proposed method that sheds more light on its strengths as well as weaknesses.
We answer the aforementioned questions, and some minor unlisted ones as a results, using extensive experiments.

– What is the quantitative improvement with respect to the baselines and previous state-of-the-art results?
– What is the contribution of the graph adaptation method?

As can be seen in Table 1, ours improves the previous SOTA by almost 3% on success rate and more than 1% on SPL. This shows that our graph adaptation helps the agent find the targets in smaller number of steps. This is further emphasised on longer trajectory lengths where the knowledge graph can receive more adaptation gradients (remember we perform test-time gradients every six steps). As is shown, this adaptation further increases the performance gain to more than 5% on success rate and more than 2% on SPL, which is double the gain in shorter distances. Thus, the shorter trajectory performance gain can be assigned to the adaptation learnt by the knowledge graph; while the longer trajectory gain can be then related to the test-time gradient-based adaptation. This confirms the effectiveness of our two layer adaptation approach. Furthermore, this also confirms the intuitive idea behind our approach that a fixed prior knowledge has limitations. This limitation is to some extent compensated using our approach. We, also, hope this can encourage future research in this promising area.

– What is the contribution of the graph-based value estimation? How would the approach perform without value estimation part?

In order to observe the effectiveness of our graph-based value estimation, we will analyse the results shown on Table 2. In this table, we compare our final method with two variants. First variant, termed as ours-input is to simply add our graph as part of the state space observation to the main network. This is similar to the approach propose in [38]. As can be seen we can still improve the baseline results; however, we do not observe significant gain as compared with two other methods. We conjugate that estimating the value using our graph can, instead, directly draw relations between the states for better action sampling and policy training. As another variant, termed as ours-policy, we study the effect of removing the state-space expansion, by directly using the graph as a side knowledge base for the policy to condition the actions upon. This approach can be observed as a weighted ensemble of policy functions representing different distributions. Again, as is observed, the performance gain is limited compared to our final model. This further confirms the effectivity of integration of knowledge graph for value estimation. This way, the size of observed state space by the model is kept limited, while the algorithm best learns how to employ and adapt the provided knowledge.

– What is the contribution of some of our design choices like the graph node features?
Table 2. Our three different methods for knowledge incorporation. This shows the effectiveness of graph-based value estimation.

| Method    | SPL   | Success | SPL >5 | Success >5 |
|-----------|-------|---------|--------|------------|
| Ours-input| 15.13 | 38.8    | 13.68  | 29.64      |
| Ours-policy| 13.88 | 43.3    | 12.93  | 33.53      |
| Ours-value | 17.27 | 43.8    | 15.39  | 33.68      |

There are different hyper-parameters in our method that are optimised using conventional routines. Due to computational complexity an extensive study of all these parameters is practically infeasible (a single training of our method from scratch takes up to six days on a single Quadro RTX8000 GPU with 12 parallel agents). Therefore, we believe current results can potentially be further improved by more careful hyper-parameter tuning. Among these parameters, however, graph’s node features is considered a significant design part. In order to show the integrity of current design, we show the effect of removing observation (egocentric image) features from the node features. Thus, the graph will reduce to fixed correlations among the objects. It can also be observed as a sub-network for the value estimation to store value decomposition information without considering the observational correlations. As can be seen in table 3 there is a significant drop in performance. This further proves the contribution of prior knowledge for value estimation. Additionally, this is a counter-argument for the following argument: the graph sub-network acts as additional parameters for the value function to decompose the return irrespective of the knowledge stored in the graph.

– Under what circumstances the model has gained performance improvements and what are its weaknesses?

Finally, in this section, we provide analysis of the practical performance of our method compared to previous SOTA ref [35] using sample test-set trajectories. As is shown in figure 4 top, the agent is navigating towards an instance of ”box” in a kitchen scene. SAVN, passes the target location (green star) without successful stopping.

In contrast, our method successfully stops at the target after 13 steps. In this trajectory only two single adaptation step is performed (six step intervals). A similar scenario happens in the figure 4 bottom where the agent is navigating towards a ”book” in a bedroom scene. In this example, our agent misses the target once; however it’s able to return after more adaptation steps are taken to conform the prior knowledge to the current scene. For more detailed comparison, we also provide detailed results per room type, in Table 4.

Table 3. Ours-no-image is the variant of our model where the image features are removed from the graph node features. We can see the graph is highly reliant on the observations to learn the relationships.

| Method     | SPL   | Success |
|------------|-------|---------|
| Ours-no-image| 12.84 | 42.4    |
| Ours-best  | 17.27 | 43.8    |
Table 4. Detailed comparison with previous SOTA; SPL/Success rate are reported per room type. We can see that our method is general enough that improves the performance in 3/4 of the room types, with marginal performance on 1/4.

| Method   | Bathroom  | Bedroom  | Kitchen  | Living room |
|----------|-----------|----------|----------|-------------|
| SAVN     | 28.49/69.6| 8.65/29.2| 17.8/43.6| 7.71/21.6   |
| Ours     | 31.03/75.6| 8.06/27.6| 17.93/45.6| 9.41/25.2   |

Adaptation Steps  How many adaptation steps is enough during testing? This question is answered here using experimental results. As can be seen in Table 5, we once limit the number of adaptation steps, using learning rate 0.01, to only four steps. This is experimentally chosen as having the best performance compared to higher values. If we continue the adaptation during testing the performance declines. We conjugate that this is due to two different reasons: one is the well-known forgetting problem associated with meta-learning approaches. That is the agent updates itself to the extent it loses the useful information stored as network weights. Second is a limitation of our approach that we plan to investigate further in the future. That is, in longer episodes, the agent experiences more different observation states that the adaptation loss in no longer able to provide useful feedback on. Thus, the gradient updates are hurting more than improving the performance.

Table 5. Ours-unlimited is the variant of our model where the during test time unlimited adaptation steps are taken every six steps. Ours-best is when this is limited to four updates.

| Method             | SPL  | Success |
|--------------------|------|---------|
| Ours-unlimited     | 15.87| 42.5    |
| Ours-best          | 17.27| 43.8    |

5 Conclusion

In this paper, we present, for the first time, the use of knowledge graphs in conjunction with meta-learning for visual navigation without explicitly employing off-the-shelf object detectors. Using extensive experiments and ablation studies, we prove the efficiency of our approach in benefiting from externally gained prior knowledge while adapting it to the new environments, where necessary. We showed, for the first time, that knowledge distillation from the critic and prior knowledge graph improves performance in navigating agents.

Since we have empirically proven efficacy of our approach, as part of our future work, we plan to extend it to other RL algorithms. Furthermore, we plan to investigate incorporating various knowledge bases required for the task of navigation, like object categories, relative location etc. We believe this work creates new avenues for future research for improved knowledge distillation techniques for navigation.
References

1. Anderson, P., Chang, A., Chaplot, D.S., Dosovitskiy, A., Gupta, S., Koltun, V., Kosecka, J., Malik, J., Mottaghi, R., Savva, M., et al.: On evaluation of embodied navigation agents. arXiv preprint arXiv:1807.06757 (2018)

2. Anderson, P., Wu, Q., Teney, D., Bruce, J., Johnson, M., Sünderhauf, N., Reid, I., Gould, S., van den Hengel, A.: Vision-and-language navigation: Interpreting visually-grounded navigation instructions in real environments. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 3674–3683 (2018)

3. Anderson, P., Wu, Q., Teney, D., Bruce, J., Johnson, M., Sunderhauf, N., Reid, I., Gould, S., van den Hengel, A.: Vision-and-language navigation: Interpreting visually-grounded navigation instructions in real environments. 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition (Jun 2018). https://doi.org/10.1109/cvpr.2018.00387, http://dx.doi.org/10.1109/CVPR.2018.00387

4. Bai, S., Kolter, J.Z., Koltun, V.: An empirical evaluation of generic convolutional and recurrent networks for sequence modeling (2018)

5. Cadena, C., Carlone, L., Carrillo, H., Latif, Y., Scaramuzza, D., Neira, J., Reid, I., Leonard, J.J.: Past, present, and future of simultaneous localization and mapping: Toward the robust-perception age. IEEE Transactions on robotics 32(6), 1309–1332 (2016)

6. Cordts, M., Omran, M., Ramos, S., Rehfeld, T., Enzweiler, M., Benenson, R., Franke, U., Roth, S., Schiele, B.: The citiescape dataset for semantic urban scene understanding. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 3213–3223 (2016)

7. Dai, H., Li, Y., Wang, C., Singh, R., Huang, P.S., Kohli, P.: Learning transferable graph exploration (2019)

8. Duan, Y., Schulman, J., Chen, X., Bartlett, P.L., Sutskever, I., Abbeel, P.: Rl2: Fast reinforcement learning via slow reinforcement learning. arXiv preprint arXiv:1611.02779 (2016)

9. Finn, C., Abbeel, P., Levine, S.: Model-agnostic meta-learning for fast adaptation of deep networks (2017)

10. Gu, J., Zhao, H., Lin, Z., Li, S., Cai, J., Ling, M.: Scene graph generation with external knowledge and image reconstruction. 2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) (Jun 2019). https://doi.org/10.1109/cvpr.2019.00207, http://dx.doi.org/10.1109/CVPR.2019.00207

11. Gupta, S., Davidson, J., Levine, S., Sukthankar, R., Malik, J.: Cognitive mapping and planning for visual navigation. 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (Jul 2017). https://doi.org/10.1109/cvpr.2017.769

12. Gupta, S., Fouhey, D., Levine, S., Malik, J.: Unifying map and landmark based representations for visual navigation. arXiv preprint arXiv:1712.08125 (2017)

13. JeffreyPennington, R., Manning, C.: Glove: Global vectors for word representation. Citeseer

14. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980 (2014)

15. Kipf, T.N., Welling, M.: Semi-supervised classification with graph convolutional networks. arXiv preprint arXiv:1609.02907 (2016)
16. Kolve, E., Mottaghi, R., Han, W., VanderBilt, E., Weih, L., Herrasti, A., Gordon, D., Zhu, Y., Gupta, A., Farhadi, A.: Ai2-thor: An interactive 3d environment for visual ai (2017)

17. Krishna, R., Zhu, Y., Groth, O., Johnson, J., Hata, K., Kravitz, J., Chen, S., Kalantidis, Y., Li, L.J., Shamma, D.A., et al.: Visual genome: Connecting language and vision using crowdsourced dense image annotations. International Journal of Computer Vision 123(1), 3273 (Feb 2017). https://doi.org/10.1007/s11263-016-0981-7

18. Li, R., Tapaswi, M., Liao, R., Jia, J., Urtasun, R., Fidler, S.: Situation recognition with graph neural networks. 2017 IEEE International Conference on Computer Vision (ICCV) (Oct 2017). https://doi.org/10.1109/iccv.2017.448

19. Marino, K., Salakhutdinov, R., Gupta, A.: The more you know: Using knowledge graphs for image classification. 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (Jul 2017). https://doi.org/10.1109/cvpr.2017.10

20. Milford, M., Wyeth, G.: Persistent navigation and mapping using a biologically inspired slam system. The International Journal of Robotics Research 29(9), 1131–1153 (2010)

21. Mirowski, P., Grimes, M., Malinowski, M., Hermann, K.M., Anderson, K., Teplyashin, D., Simonyan, K., Zisserman, A., Hadsell, R., et al.: Learning to navigate in cities without a map. In: Advances in Neural Information Processing Systems. pp. 2419–2430 (2018)

22. Mirowski, P., Pascanu, R., Viola, F., Soyer, H., Ballard, A.J., Banino, A., Denil, M., Goroshin, R., Silre, L., Kavukcuoglu, K., et al.: Learning to navigate in complex environments. arXiv preprint arXiv:1611.03673 (2016)

23. Mishkin, D., Dosovitskiy, A., Koltun, V.: Benchmarking classic and learned navigation in complex 3d environments (2019)

24. Mnih, V., Badia, A.P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., Silver, D., Kavukcuoglu, K.: Asynchronous methods for deep reinforcement learning. In: Balcan, M.F., Weinberger, K.Q. (eds.) Proceedings of The 33rd International Conference on Machine Learning. Proceedings of Machine Learning Research, vol. 48, pp. 1928–1937. PMLR, New York, New York, USA (20–22 Jun 2016). http://proceedings.mlr.press/v48/mnih16.html

25. Mousavian, A., Toshev, A., Fiser, M., Kosecka, J., Wahid, A., Davidson, J.: Visual representations for semantic target driven navigation. 2019 International Conference on Robotics and Automation (ICRA) (May 2019). https://doi.org/10.1109/icra.2019.8793493

26. Pan, X., You, Y., Wang, Z., Lu, C.: Virtual to real reinforcement learning for autonomous driving. Proceedings of the British Machine Vision Conference 2017 (2017). https://doi.org/10.5244/c.31.11

27. Savva, M., Kadian, A., Maksymets, O., Zhao, Y., Wijmans, E., Jain, B., Straub, J., Liu, J., Koltun, V., Malik, J., Parikh, D., Batra, D.: Habitat: A platform for embodied ai research (2019)

28. Thrun, S., Pratt, L.: Learning to learn. Springer Science & Business Media (2012)
31. Velikovi, P., Cucurull, G., Casanova, A., Romero, A., Li, P., Bengio, Y.: Graph attention networks (2017)
32. Vijay, V.K., Ganesh, A., Tang, H., Bansal, A.: Generalization to novel objects using prior relational knowledge (2019)
33. Wang, J.X., Kurth-Nelson, Z., Tirumala, D., Soyer, H., Leibo, J.Z., Munos, R., Blundell, C., Kumaran, D., Botvinick, M.: Learning to reinforcement learn. arXiv preprint arXiv:1611.05763 (2016)
34. Wang, X., Xiong, W., Wang, H., Wang, W.Y.: Look before you leap: Bridging model-free and model-based reinforcement learning for planned-ahead vision-and-language navigation. Lecture Notes in Computer Science p. 3855 (2018)
35. Wortsman, M., Ehsani, K., Rastegari, M., Farhadi, A., Mottaghi, R.: Learning to learn how to learn: Self-adaptive visual navigation using meta-learning. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 6750–6759 (2019)
36. Xia, F., Zamir, A.R., He, Z., Sax, A., Malik, J., Savarese, S.: Gibson env: Real-world perception for embodied agents. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 9068–9079 (2018)
37. Xiong, W., Hoang, T., Wang, W.Y.: Deeppath: A reinforcement learning method for knowledge graph reasoning. arXiv preprint arXiv:1707.06690 (2017)
38. Yang, W., Wang, X., Farhadi, A., Gupta, A., Mottaghi, R.: Visual semantic navigation using scene priors. arXiv preprint arXiv:1810.06543 (2018)
39. Yun, S., Jeong, M., Kim, R., Kang, J., Kim, H.J.: Graph transformer networks (2019)
40. Zhang, Y., Dai, H., Kozareva, Z., Smola, A.J., Song, L.: Variational reasoning for question answering with knowledge graph. In: Thirty-Second AAAI Conference on Artificial Intelligence (2018)
41. Zhou, J., Cui, G., Zhang, Z., Yang, C., Liu, Z., Wang, L., Li, C., Sun, M.: Graph neural networks: A review of methods and applications. arXiv preprint arXiv:1812.08434 (2018)
42. Zhu, Y., Mottaghi, R., Kolve, E., Lim, J.J., Gupta, A., Fei-Fei, L., Farhadi, A.: Target-driven visual navigation in indoor scenes using deep reinforcement learning. 2017 IEEE International Conference on Robotics and Automation (ICRA) (May 2017). https://doi.org/10.1109/icra.2017.7989381 [http://dx.doi.org/10.1109/ICRA.2017.7989381]