Learning to Navigate Wikipedia by Taking Random Walks

Anonymous Author(s)
Affiliation
Address
email

Abstract

A fundamental ability of an intelligent web-based agent is seeking out and acquiring new information. Internet search engines reliably find the correct vicinity but the top results may be a few links away from the desired target. A complementary approach is navigation via hyperlinks, employing a policy that comprehends local content and selects a link that moves it closer to the target. In this paper we show that using behavioral cloning of randomly sampled trajectories is sufficient to learn an effective link selection policy. We demonstrate the approach on a graph version of Wikipedia with 38M nodes and 387M edges. The model is able to efficiently navigate between nodes 5 and 20 steps apart 96% and 92% of the time, respectively. We then use the resulting embeddings and policy in a downstream fact verification task where, in combination with basic TF-IDF search and ranking methods, they are able to obtain competitive results to state-of-the-art methods.

1 Introduction

The ability to gather new knowledge about the world is a fundamental aspect of intelligence. Based only on a few words, a fact, question, or even vague idea, humans have the ability to use the internet to find extremely specific information about the world. From the important (how to administer CPR) to the trivial (the LA Raiders won the 1983 SuperBowl), virtually any knowledge is a click away.

In this work, we focus in one particular aspect of this ability: web navigation. In general, navigation is a key component of an embodied agent: the ability to move efficiently toward a target. In known environments, where map information is available, shortest-path algorithms provide a viable solution. However, in novel settings, the agent lacks such global information and must instead *navigate* to the target, using its understanding of the local environment to select actions.

Other approaches to web agents have focused mostly on retrieval or search engines. These, however, only provide a partial solution, typically getting close to a desired target but often not exactly to the right page. In this work, we present an approach for navigation on graph-structured web data that uses hyperlinks within articles to navigate toward a target. It complements search engines, using them to provide a sensible starting point for a local search for specific information.

To be able to navigate on the web effectively, we must read the text of the page, see how concepts in the current page are related to a possible next page, and use our understanding to learn a policy to make the right navigation decision. Consider the scenario shown in Figure 1. The agent is trying to navigate to the target node NORTH AMERICA while currently at the node PRESIDENTS OF THE UNITED STATES containing two hyperlinks: BARACK OBAMA and USA. Our approach learns an embedding from the text of the current page and the text for each hyperlink node and passes these to a goal-conditioned policy that selects the best possible action. To make the correct choice in this example, the agent should associate USA with NORTH AMERICA.

Submitted to 36th Conference on Neural Information Processing Systems (NeurIPS 2022). Do not distribute.

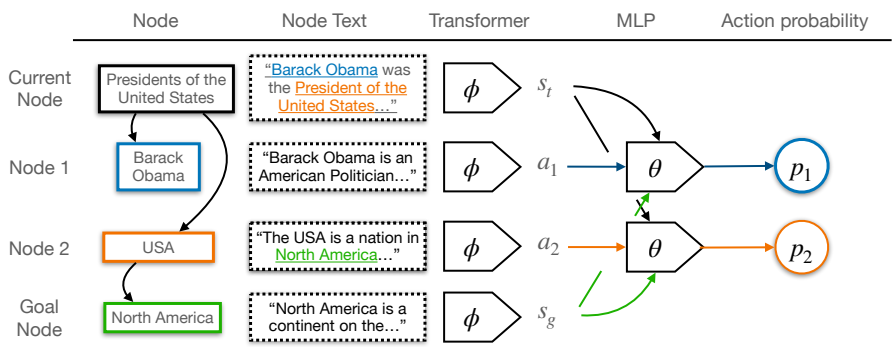

Figure 1: Our agent navigates through hyperlinks on a Wikipedia graph towards a target node. It first embeds the text at each link node using Transformer $\phi(.)$ and then evaluates different actions/links a_1, a_2 with a policy θ, conditioned on the current and target embeddings s_t and s_g, respectively.

The particular web environment we consider here is Wikipedia, converted into a graph form. Each of the 38M paragraphs is represented by a node; edges are links within and between articles. We introduce an approach for navigation on this graph based on behavioral cloning of automatically generated trajectories. This allows for unsupervised pre-training of both web page embeddings and navigation policy. We explore different trajectory distributions for training the model, the best practical choice being equivalent to random walks on the graph.

Our contributions are: (i) we demonstrate that our pre-training scheme enables the model to reliably ($> 90\%$ success) navigate between pairs of nodes in a 387M-edge Wikipedia graph; (ii) we investigate several strategies for sampling random trajectories from the graph, concluding that a simple forward random walk is the best choice in practical settings; (iii) we show that paragraph embeddings trained for navigation outperform those from language models (trained for masked language modelling) and (iv) we show the utility of our navigation scheme for the FEVER Thorne et al. [2018] fact verification task, demonstrating a significant boost when integrated into a simple TF-IDF scheme, as well as obtaining a F1 score competitive with SOTA (73.1% vs 79.9% SOTA) with a more complex re-ranker.

2 Background

Wikipedia as a navigation environment: Human behavior on Wikipedia [West and Leskovec, 2012a] was collected via the Wikispeedia game [West et al., 2009], which asked the players to move between a pair of arbitrarily chosen articles in the fewest steps. This rule allows the emergence of a semantic distance between concepts that facilitates retrieval. West and Leskovec [2012b] conducted initial studies on this dataset for navigation, using supervised learning and reinforcement learning methods on a small subset of articles (4.6K) with simple TF-IDF features.

Most relevant to our work, Nogueira and Cho [2016] explore the same problem of navigation on a large Wikipedia graph (12M nodes). They train LSTM agents to perform navigation on the graph, using Bag-of-Word features for each article. The trained agent is then applied to a Jeopardy Q&A task. Our approach revisits this framing but with a number of important differences: (i) on a significantly larger graph (38M vs 12M); (ii) using modern text features, e.g. BERT features fine-tuned with a Transformer-based navigation policy; (iii) they use a single fixed start node, whereas ours is able to navigate from any start node; (iv) removal of beam search, or other complex search mechanisms at evaluation time. Collectively, our approach boosts the navigation success rate from 12.5% in [Nogueira and Cho, 2016] to over 90%,[1] despite a much larger graph.

Fact verification and Q&A using Wikipedia: Wikipedia is a rich resource for evaluating natural language understanding through question and answering tasks. Chen et al. [2017] combine TF-IDF retrieval with an LSTM document reader to find answers to free-form questions within Wikipedia documents. Contemporary Q&A systems [Semnani and Pandey, 2020, Qu et al., 2020] extend this general approach to more complex pipelines, combining traditional and neural methods with inverted index search/retrieval schemes, followed by document readers that perform re-ranking.

[1] Exact comparisons are infeasible given idiosyncrasies in graph construction and other differences in setup.

We demonstrate our navigation approach in conjunction with very basic retrieval and re-ranking approaches, so as to clearly demonstrate the value of navigation. In common with our approach, Wang et al. [2021] also use a graph version of Wikipedia, but with the goal of aligning it to a knowledge base, rather than navigation. Stammbach and Neumann [2019] propose a scheme for finding evidence for the FEVER Thorne et al. [2018] benchmark that explores all links one step away from an initial retrieval set. This can be viewed as a 1-step navigation operation, in contrast to our scheme that permits an arbitrary number of steps.

Navigation in knowledge bases: Graph navigation is a key element in utilizing knowledge bases (KB), where facts are represented as tuples of entity nodes connected by a labelled edge corresponding to relation between the two nodes. For example, Das et al. [2017] answer complex queries by navigating the graph, inferring missing edges along the way, and finding a valid path to the answer node. In our case, the graph is fixed but consists of natural language text and is partially observed. This makes the task much harder on two fronts. First, the policy must comprehend the natural language actions, without access to crisply labelled relation actions. Second, the action space is open and varying in size, unlike in a KB which typically has a fixed schema, i.e. relations/actions come from a fixed size set. Perhaps the closest work involving natural language actions, Fu et al. [2019] augment a KB with textual nodes. However, it still operates under the fixed KB schema. Further improvements on navigation in KB have been shown by incorporating tricks like reward-shaping [Xiong et al., 2017] and action dropout [Lin et al., 2018]. We adopted the action dropout where some outgoing edges are masked during training to enable more effective path exploration. Pushing more on exploration, inspired by AlphaGo, Shen et al. [2018], He et al. [2022] combine the policy with Monte-Carlo tree search to navigate KB. We find that no search method is necessary for our approach at evaluation time, with good performance obtained from the policy alone.

Internet powered Q&A: Several recent works have attempted complex question answering using the internet. Talmor and Berant [2018] retrieve snippets of information from the web with a search engine to answer a structured decomposition of the question with a simpler Q&A model. The recent WebGPT Nakano et al. [2021] system takes a broadly similar approach to long-form question answering. WebGPT accures information from the Bing search API as well as some navigation, and passes it to an LM to generate the answer text. The complexity of the system necessitates the use of human demonstrations for training. Related work from Lazaridou et al. [2022] utilizes a few-shot supervision of an LM to perform Google queries, which are then used by the LM to generate an answer. They show that by incorporating information from Google queries, the answers are more factual than those generated directly from LMs. In contrast to these works, we focus on mastering navigation rather than any combination with search/retrieval, albeit in more limited domain (Wikipedia vs the internet). However, our approach could potentially act as a core component in a general web navigation, being more versatile on account of not requiring supervision.

3 Approach

We consider navigation in a web environment $\mathcal{W} = \{\mathbf{N}, \mathbf{L}\}$, where $\mathbf{N} := \{n_i\}_{i=1}^N$ is the set of nodes, each n_i representing a web page with content s_i. The links between pages are represented by $\mathbf{L} := \{l_i\}_{i=1}^N$, where $l_i = \{l_i^1, \ldots, l_i^j, \ldots, l_i^{M_i}\}$ are the set of M_i (directed) out-links from node i. We define $\text{NH}(n_i)$ as the set of nodes reached by following all links l_i from node n_i, where the jth neighbor is given by $l_i^j(n_i)$.

We assume some target node n_g and some starting node n_0 are given, and wish to train an agent that can navigate from n_0 to n_g by walking along the edges of the graph within T steps. To accomplish this, we construct a parametric policy $p_\theta(n_{t+1}|n_t, n_g)$ which, at time step t, parameterizes a distribution over $n_{t+1} \in \text{NH}(n_t)$. We train this model via behavioral cloning to maximize

$$\mathcal{L}(\theta) = \mathbf{E}_{\mathcal{E}(n_0,\ldots,n_T)}\left[\sum_{t=0}^{T-1} \log p_\theta(n_{t+1}|n_t, n_g)\right] \tag{1}$$

where $\mathcal{E}(n_0, \ldots, n_T)$ is our trajectory distribution which generates trajectories $(n_0, \ldots, n_T)$ that we would like our model to replicate (i.e. setting $n_g = n_T$).

3.1 Trajectory distribution

This framework opens up many choices for $\mathcal{E}$. Clearly some will result in ineffective navigation strategies while others may enable us to train a policy which is capable of generalizing to new graphs and downstream applications. We elaborate on some choices below.

Reverse trajectories We define $\mathcal{E}(n_T) = \text{Uniform}(\mathbf{N})$ (the uniform distribution over nodes in the graph). Now we can define

$$\mathcal{E}(n_t|n_{t+1}) = \frac{1}{|\{n_i; n_i \in \text{NH}(n_t)\}|} \quad \text{if } n_{t+1} \in \text{NH}(n_t), \quad \text{else } 0 \tag{2}$$

or, a distribution which uniformly walks the *reverse* graph (with transposed adjacency matrix). We begin with some uniformly chosen target node and walk backward randomly to n_0 in such a way that we can reach the target by walking forward from n_0.

Intuitively, training with the reverse trajectories defined above should allow our model to be able to reach any node in the graph from any node it is connected to. Unfortunately, in realistic web graph data, there exist many "dead-ends" or nodes with no in-going edges. The trajectories get stuck at these nodes leading to low-diversity training data.

Random forward trajectories In our real-world knowledge graph data, there are far fewer nodes which have no out-going edges. Thus we propose an alternative method to generate target nodes based on randomly walking forward. We define $\mathcal{E}(n_0)$ to be the Uniform($\mathbf{N}$) and

$$\mathcal{E}(n_{t+1}|n_t) = \frac{1}{|\text{NH}(n_t)|} \quad \text{if } n_{t+1} \in \text{NH}(n_t), \quad \text{else } 0. \tag{3}$$

or, a distribution which uniformly picks a starting node and randomly walks the *forward* graph for T steps. Training in this way should encourage our model to be able to reach any node which is reachable from some uniformly chosen start. We note that the target node distribution $\mathcal{E}(n_T)$ will not be uniform and instead be a function of the graph structure. We also note that while this generates a T-step trajectory, because it is generated by random walk, the shortest path distance between start and target nodes may be smaller.

Random shortest paths We define $\mathcal{E}(n_T) = \text{Uniform}(\mathbf{N})$ and let $\mathcal{E}(n_0|n_T)$ be the uniform distribution of nodes whose shortest path length to n_T is T steps. We then define $\mathcal{E}(n_t|n_0, n_T)$ to be the delta distribution that n_t is the t-th node of this shortest path. While it can be challenging to compute the probabilities of this distribution efficiently we can draw samples from it using a shortest path finding algorithm such as Dijkstra's algorithm.

3.2 Model parameterization

From above, we can see the core of our model is the $p_\theta(n_{t+1}|n_t, n_T)$ distribution. Given a sampled trajectory that begins at n_0, this is the distribution which attempts to choose the correct action to take at timestep $t+1$ starting at node n_t in order to get to target node n_T. We parameterize this distribution as a function of the text available at node n_t.

$$p_\theta(n_{t+1}|n_t, n_g) \propto \exp(f_\theta(s_{t+1}, s_t, s_g, t)) \tag{4}$$

where f_θ embeds the text s_g, s_t and s_{t+1} of the target, current and destination nodes, respectively. Further details on how we apply this general model to Wikipedia are given in Section 4.2.

4 Application to Wikipedia

4.1 Navigation and sentence search tasks

Data processing: We convert a snapshot of English Wikipedia into a graph but split each article ($\bar{\mu} = 1000$ words) into paragraph-sized blocks ($\bar{\mu} = 100$ words), which form the *nodes*. *Edges* in the graph are the union of (i) organic hyperlinks, (ii) additional entity linking and (iii) next/previous paragraph links. The full graph has $\sim$ 38M nodes and 387M edges. For initial experiments, we also

use a smaller subsampled version of 200k nodes. For this we sub-sample disjoint sets of 200k train / 200k evaluation Wikipedia nodes. See Section A.2 for full details.

T-step navigation: The most basic task is T-step navigation. At the start of the episode, we randomly sample a start node and generate a T-step random walk trajectory. We then take the last node of this trajectory as the target n_g. The agent is then given a time budget B (in most experiments, we fix max steps to $B = 100$), and succeeds if it reaches the target node within B steps. In our experiments, we show results on various graphs and splits with $T = \{5, 10, 20\}$-step navigation.

1-T Multistep navigation: The Multistep task is much like the previous setting, except that instead of generating fixed T-step trajectories, we sample from Uniform$(1, T)$, $T = 20$.

T-step sentence search: This is the same setup as T-step navigation, except that rather than provide the full text for the target node n_g, only one sentence within that text is selected at random to compute the target embeddings from. Thus agents have the more challenging task of locating the correct node, given only a small snippet of its text (see Sec. 4.2 for details on target representation). We include this task because one of the difficulties of web navigation is establishing which page should be the target in the first place. It also has direct relevance to our downstream fact verification task.

4.2 Model architecture

State representation: For each node n_i we embed its text s_i using a Transformer $\phi(.)$ to produce $\phi(s_i)$. By basing the state representation on semantic content, rather than i.e. node index, the agent is able to learn about the relationships between entities in a manner that allows for generalization to new graphs at evaluation time, instead of memorizing the structure of the training graph. Following Karpukhin et al. [2020], we append to the node text the title of the Wikipedia article to which it belongs. This often provides context to the node that might otherwise be missing – in many articles, the subject of the article will be referred to indirectly (e.g. it might state that "she" was born in 1945 instead of repeating the subject's name).

For those experiments where the node embeddings are kept fixed, $\phi(.)$ is a pre-trained RoBERTA [Liu et al., 2019] model that encodes the node text & title. However, in our large-scale experiments, we pre-train the Transformer $\phi(.)$ directly using the text & title (see Appendix B.1). In both cases, we apply the Transformer to the tokenized text, take the mean over the input tokens, and apply a `tanh` nonlinearity to produce $\phi(s_i)$.

Target representation: Similarly, we use the article text s_g of the target node as input to the Transformer model ϕ to compute its representation $\phi(s_g)$. For some tasks, such as sentence search or fact verification, the target representation instead relies on a single sentence from each node or other text specific to the task and a separate transformer model ϕ_{target} is trained to embed the target.

Navigation policy network: The network operates on the target node n_g and the current node n_t, and outputs a distribution over the possible actions that can be taken at n_t. As the nodes in the graph have a variable number of outgoing edges, the policy must produce a distribution with a variable number of outcomes. To do this, we first concatenate the embeddings of s_g and s_t, and pass them through a 1-layer feed-forward network to produce a combined embedding $e_{tg} = FF[\phi(s_t), \phi(s_g)]$. Next we embed each possible action a_i. When we use a fixed ϕ, we directly use $a_i = \phi(s_i)$ for $n_i \in \text{NH}(n_t)$. However this is not feasible when pre-training our embedding model due to memory concerns. Instead, we compute $a_i = \phi(s_t[l_t^i])$ where $s[l^i]$ selects the words in s which belong to hyperlink i – see Appendix B.1 for more details. We then define the probability of moving to n_i from n_t with target n_g as $p(n_i|n_t, n_g) \propto \exp(e_{tg} \cdot a_i)$ which is normalized to produce a distribution over the $|\text{NH}(n_t)|$ actions.

In our larger-scale model, we also include the trajectory leading up to the current node $(n_0, \ldots, n_t)$. We pass their embeddings, combined with the target embedding, $[\phi(s_0), \ldots, \phi(s_t), \phi(s_g)]$, through a 4-layer transformer and use the output as e_{tg} when computing the transition probabilities above. We explore these two policy architecture choices, feed-forward and Transformer in Table 4.

4.3 Application to fact verification

Finally, we use our navigation approach to gather information for the task of fact verification on the FEVER benchmark Thorne et al. [2018] which also uses Wikipedia as a knowledge base. One

important difference from our earlier navigation task is that the target node n_g is not specified at evaluation time. Instead, we are given a claim whose veracity must be established by finding information in a particular node in the graph. To determine which node this is, we use a target encoder which maps the claim to an embedding vector that can be used in place of $\phi(s_g)$ in Section 4.2.

Due to the limited size of the FEVER training set, we pre-train our graph embeddings, navigation policy and target encoder on the similar sentence search task. We then freeze the embeddings and policy and fine-tune the target encoder on (n_g, s_g) pairs from the FEVER training set. To avoid over-fitting, we add an auxiliary loss which enforces the similarity between our target embedding and the embedding of the ground truth node containing the sentence needed to resolve the claim. Training and evaluation on FEVER required alignment between our version of Wikipedia and the one used to compile the benchmark and details of this can be found in Appendix A.4.

Next, our navigation method requires a starting node. To put our agent close to the articles we need in the downstream task, we run a popular variant of TF-IDF, called BM25 [Robertson and Zaragoza, 2009]. We first create an index over all of the nodes in the graph, then take the top-5 BM25 matches as the starting nodes, and run navigation for 20 steps on each. Note that BM25 does not find the exact text (i.e. the hits@1 is not very high) but matches are in the right vicinity, usually in the same or similar article. Thus the agent only has to navigate a few steps to reach the target: for the FEVER dataset, the mean length of shortest path between start and target node is 4.3 steps.

The final component is a re-ranker that takes all the sentences from all nodes visited by the agent and assigns a match score between each one and the claim. The ranked list of sentences can then be used for benchmark evaluation. We explore two types of ranker: (a) basic TF-IDF and (b) a BERT model, fine tuned using hard negatives mined from the TF-IDF ranker. See Appendix A.4 for more details.

Evaluation of our approach thus consists of: (i) given a claim, use BM-25 to return a list of promising start locations; (ii) from the top $N = 10$ of these, run our navigation model for 20 steps, using the target encoder to embed the claim, (iii) use the re-ranker to score sentences visited against the claim and (iv) select the top 5 for computing the F1 score, for comparison on the FEVER benchmark.

5 Experiments

5.1 Investigation of training trajectory distribution

On the smaller 200k node graph, we investigate different choices for the trajectory policy distributions $\mathcal{E}$ described in Section 3: (i) **Random Forward Trajectories**, where we randomly sample start nodes and do a random walk to get a target, (ii) **Reverse Trajectories**, where we randomly sample a target node and and do a random walk on the reverse graph to get a start node, and (iii) **Random Shortest Paths** where we randomly sample a start node and take a random walk and then compute the shortest path between the start and target nodes. For each of these we trained policies on 5, 10, and 20 step navigation tasks. We evaluate these in two different ways: (i) forward sampling of source and target in the disjoint evaluation graph, (ii) reverse sampling of source and target. The results of this experiment are shown in Table 1. Unsurprisingly, we can see that forward and reverse trained policies both do better when evaluated in the way that matches training. However, forward trajectory policies generalize much better to reverse navigation. When we analyze the traces of these experiments, we see that in the reverse sampling, many target nodes are in "dead-ends" where many target nodes do not have long random trajectories to sample. This results in easier navigation (as we can see from higher accuracy in reverse trajectory trained policies on reverse navigation) and more extreme overfitting, leading to poor performance of reverse trajectory on forward navigation.

Not surprisingly, we also see that training on a shortest path trajectories leads to better performance both in the forward and reverse sampling. This advantage is especially seen in the 20-step navigation task. Ideally, we would always train on shortest paths. However, the time complexity of pre-computing all shortest paths in a graph using Dijkstra's algorithm is $O(V(E + VlogV))$ for the number of nodes V and number of edges E in the graph. On the full 38M Wikipedia graph, this computation would be completely intractable. Luckily, the random forward trajectories perform almost as well and can be computed in constant time with respect to the size of the graph and only linear time with respect to the trajectory length. For all subsequent experiments, we use the **Random Forward Trajectories** for our trajectory policy and evaluate on the forward version of the navigation tasks, where we call our approach **Random Forward Behavioral Cloning (RFBC)**.

Table 1: Different trajectory distributions on small graph (200k nodes)

	Navigation (Forward)			Navigation (Reverse)		
Trajectory Policy	5	10	20	5	10	20
Forward	85.3	76.4	67.5	31.1	32.6	19.9
Reverse	2.3	0.5	0.2	91.3	87.9	87.3
Shortest path	86.7	85.0	84.6	74.3	31.6	28.4

Table 2: Small graph navigation (200k nodes) – Success rate (%)

	Navigation				Sentence Search		
Method	5	10	20	multistep	5	10	20
RFBC (ours)	85.3	76.4	67.5	77.4	60.7	47.6	34.6
RL	41.4	40.2	41.6	43.6	14.0	1.5	8.9
RFBC + RF	17.4	16.3	16.6	17.1	-	-	-
RL + RF	6.0	7.5	7.0	6.7	-	-	-
Random	12.3	14.9	12.3	14.0	-	-	-
Greedy	19.7	16.7	21.7	23.4	-	-	-
Random DFS	10.0	9.5	8.3	10.0	-	-	-
Greedy DFS	31.1	23.8	22.7	51.8	-	-	-

5.2 Navigation

We next do a more thorough set of experiments on the smaller 200k Wikipedia graph. Because the graph is much smaller than the full Wikipedia and thus prone to overfitting, for all of our methods and baselines we use the fixed RoBERTA embeddings $\phi(s_i)$, the simplest 1-layer feed-forward network, and MiniBERT Turc et al. [2019] for target sentence embedding $\phi(s_g)$.

Alongside our approach, we employ the following baselines/ablations:

- RFBC: Random-forward behavioral cloning (our approach).
- RL: Navigation using a policy gradients RL agent with same policy architecture as our model. See Sec. A.5 for more details.
- RFBC + RF: RFBC but with random features sampled from the unit sphere instead of ϕ
- RL + RF: RL with random features.
- Random: policy that chooses random action (out-link) at each timestep.
- Greedy: selects the action embedding which has the smallest cosine distance with the target.
- Random DFS: DFS of depth equal to number of steps, choosing actions at random.
- Greedy DFS: DFS but select action with smallest cosine distance to the target.

The results evaluated on the held-out graph are shown in columns 1-4 of Table 2. Our approach (RFBC) performs well, achieving 77% in the multistep case. Performance also drops as the distance to the target increases. In contrast, the RL agent did not perform well (around 40% on navigation tasks). Using random features in place of $\phi(.)$ caused the performance to drop to near chance, showing that the models are utilizing the semantics at each node and are not relying on a general search strategy of some kind. Random methods also do quite poorly, even when adding a DFS. Greedy methods do a bit better than random, especially with DFS, but are still well below our method.

We also consider the more challenging sentence search task described in Section 4.1. This task requires not only finding a known target node, but learning to find a target node given only a single random sentence from that node, more closely matching our downstream task and the way humans tend to approach finding information on the web. Columns 5-7 of Table 2 show performance on this task for our approach and RL. Our method has reduced performance on the task due to the extra difficulty of learning the target embedding, but still performs reasonably. RL, however, degrades to near chance performance.

Table 3: Wikipedia graph statistics

Year	# Articles	# Nodes	# Edges	# Words / Node	Median Path Length
2017	4.92M	36.3M	359M	110	15
2018	5.27M	38.5M	387M	109	15

Table 4: Full graph navigation (38M nodes) - Success rate (%)

	Navigation			Sentence Search		
Embedding + Policy	5	10	20	5	10	20
RoBERTA + Feed-forward	85.5	80.1	71.5	91.2	88.9	77.5
RoBERTA + Transformer	85.3	88.3	87.9	81.6	75.4	70.9
Embed train (ours) + Feed-forward	96.1	94.1	89.8	96.3	92.8	90.2
Embed train (ours) + Transformer	93.5	90.6	92.2	93.9	86.3	79.7

5.3 Navigation on full 38M node Wikipedia graph:

Our method scales naturally to much larger graphs, which we demonstrate by training on the entire Wikipedia 2017 graph. We evaluate on the entire Wikipedia 2018 graph. Each graph is ~ 38M nodes and ~ 380M edges (see Table 3 for details).

We explore both architectures described in Section 4.2 for the policy network: (i) the single feed-forward layer (as used in the smaller graph) and (ii) the 4-layer Transformer model. We also compare using a fixed RoBERTA model for ϕ versus pre-training a text transformer directly on the navigation task (see Appendix B.1). DistillBERT Sanh et al. [2019] is used for $\phi(n_g)$.

Table 2 shows the results on the full graph, evaluating on $\{5, 10, 20\}$ step tasks, for both navigation and sentence search. Figure 2 shows example trajectories of the trained agent. The first thing to note is that the overall navigation and sentence search performance of our methods is high. The 2018 Wikipedia graph we use for evaluation contains over 5 million articles, over 38 million nodes and 387 million edges. On this graph (which we generalize to from the previous year's graph) our best method can find an article 20 steps away 92.2% percent of the time and can locate it given just a single sentence of that article 90.2% of the time. This compares favorably with a similar evaluation performed in Nogueira and Cho [2016], where their WebNav approach achieved a 12.5% success rate for 16 step tasks on a 12M node graph. Compared to the performance of our method on the smaller 200k graph, we see that performance has greatly improved, most likely due to the addition of orders of magnitude more training data. In particular we see that sentence search performance on 20-step improves greatly from 34.6% to a best performance of 90.2%.

Additionally, in all cases, our learned embeddings perform better than using the fixed features, derived from large language models. In general, feed-forward policies perform best, except for 20 step navigation tasks. For longer trajectories, the Transformer policy network achieves slightly better performance, while on sentence search, the feed-forward model is consistently superior.

5.4 Application to fact verification

We now evaluate the ability of our navigation approach to select evidence on the FEVER development set Thorne et al. [2018] with results shown in Table 5. The primary metric used is F1@top5 but we also give precision and recall. Paired with a basic retrieval (BM25) and re-ranker (TF-IDF), the approach obtains a respectable F1 of 0.46. If we remove our navigation component then this drops

Table 5: Results on the FEVER benchmark (evidence retrieval only). The first two rows show the effect of adding our navigation scheme to the simple BM25 retrieval and TF-IDF re-ranker combination. Switching to a more powerful re-ranker (BigBird) results in a significant boost in F1 score, surpassing Stammbach [2021] who also use BigBird [Zaheer et al., 2020].

Method	Precision@5	Recall@5	F1@5
BM25 + TF-IDF [Thorne et al., 2018]	0.33	0.40	0.36
BM25 + RFBC (Ours) + TF-IDF	0.38	0.55	0.46
BM25 + RFBC (Ours) + BigBird	0.713	0.751	0.731
Stammbach [2021]	0.260	0.936	0.407

substantially to 0.36, demonstrating its utility. Swapping to a more powerful re-ranker (BigBird) boosts the F1 to 0.731, which is competitive with the state-of-the-art (FEVER leaderboard [2022] rank #6; rank #1 F1 is 0.799). We also compare to a leading approach Stammbach [2021] that similarly relies on the BigBird re-ranker, with our approach having a significantly better F1 score. A possible explanation is that their approach overwhelms the re-ranker with all possible hyperlinks whereas, navigation being inherently sparse, our model presents a more refined set for re-ranking. We note that our approach: (a) is significantly simpler than many top ranked approaches and (b) selects evidence over a much larger set than the curated version of Wikipedia used in FEVER (38M vs 5M). Examples of successful navigation traces are shown in Figure 3.

6 Discussion

We have presented a simple and effective scheme for navigating a large Wikipedia graph that is applicable to more general web navigation problems. We show that behavioral cloning of random trajectories is a viable approach to learning both entity embeddings and a navigation policy. When applied to the FEVER verification task, they offer highly competitive performance, whilst being complementary to existing approaches. One limitation of our approach when we move from Wikipedia to the wider Internet is that our scheme relies on a good target encoder. For navigation and our downstream task, there was a clear ground-truth target node available training, but in other settings this might not be the case. Another limitation is that we require a re-ranker to decide on the final retrieved sentence, but ideally the agent would decide for itself when it has reached the correct node.

Riselda Selaj	Wallonia	Key ceremony	Vladimir Padwa	56th Training Squadron
Mycenaean Greece	Field (mathematics)	Metaphysics	Reproduction	Word
0: Riselda Selaj	0: Wallonia (20)	0: Key ceremony (4)	0: Vladimir Padwa (5)	0: 56th Training Squadron (11)
1: Champion	1: Chemistry	1: Communication	1: Plan	1: 56th Training Squadron (11)
2: Latin	2: Property (philosophy)	2: Symbol	2: Business	2: 56th Training Squadron (11)
3: Greece	3: Extensionality	3: Belief	3: Product (business)	3: 56th Training Squadron (11)
4: Greece	4: Extensionality (1)	4: Pistis	4: System	4: 56th Training Squadron (10)
5: Athens	5: Extensionality	5: Belief	5: System (1)	5: Squadron (naval)
6: Attica	6: Extensionality	6: Concept	6: System (2)	6: Warship
7: Ancient Greek	7: Extensionality	7: Thought	7: Biologist	7: Navy
8: Ancient Greek	8: Equality (mathematics)	8: Goal	8: Life	8: Power projection
9: Medieval Greek	9: Quantity	9: Idea	9: Cell signaling	9: Power projection
10: Medieval Greek	10: Magnitude (mathematics)	10: Philosopher	10: Disease	10: Sovereign state
11: Ottoman Empire	11: Magnitude (mathematics)	11: Meaning of life	11: Function (biology)	11: Centralisation
12: Anatolia	12: Mathematical object	**12: Metaphysics**	12: Fitness (biology)	12: Forms of government
13: Medieval Greek	**13: Field (mathematics)**	13: Potentiality and actuality	13: Sexual selection	13: Definition
14: Medieval Greek	14: Addition	14: Four causes	14: Reproductive success	**14: Word**
15: Language	15: Addition	15: Four causes	15: Genotype	15: Word
16: Ancient Greece	16: Operation (mathematics)	16: Explanation	16: Genotype	16: Word
17: Archaic Greece	17: Operand	17: Fact	17: Gene	17: Word
18: Second Persian invasion of Greece	18: Operator (mathematics)	18: Reality	18: Biology	18: Morpheme
19: Ancient Greece	19: Space (mathematics)	19: Philosophy	19: Cell (biology)	19: Word
20: Ancient Greece	20: Mathematical structure	20: Philosophy	20: Embryo	20: Word

Figure 2: Example navigation trajectories in the 38M node Wikipedia graph. The start and target nodes are shown in the first two rows. Parentheses indicate paragraph block (zero-indexed) within article. Note that in cases where the agent fails to find the target node (cols 1 & 3), it visits ones that are very close by.

The X-Files starred an actress.	*Solanum contains the horse nettles.*	*CHiPs was written by Dax Shepard in 2011.*	*Brown University's main campus is located in the most populous city in Rhode Island.*	*The World Trade Center featured one building.*
The X-Files	List of plants known as nettle	Dax Shepard	John Brown (Rhode Island)	One World Trade Center
Gillian Anderson	Solanum	CHiPs (film)	Providence, Rhode Island	World Trade Center (1973–2001)
Introduction. Gillian Leigh Anderson, OBE (born August 9, 1968), is an American-British film, television and theatre actress, activist and writer. Her credits include the roles of FBI Special Agent Dana Scully in the long-running and widely popular series "The X-Files",	Solanum is a large and diverse Genus of Flowering plants, which include two food crops of high economic importance, the Potato and the Tomato. It also contains the nightshades and horse nettles,...	CHiPs is a 2017 American action comedy Buddy cop film written and directed by Dax Shepard, ….	Providence is the capital of and most populous city in the U.S. state of Rhode Island, founded in 1636, and one of the oldest cities in the United States. It is located in Providence County and is the third most populous city in New England…	The World Trade Center was a large complex of seven buildings in Lower Manhattan, New York City, United States. It featured landmark twin towers, which opened on April 4, 1973, and were destroyed as a result of the September 11 attacks. At the time of their completion, the "Twin Towers"–the original 1 World Trade Center, at ; and 2 World Trade Center, at –were the tallest buildings in the world….
0: The X-Files (84)	0: List of plants known as nettle (1)	0: Dax Shepard	0: John Brown (Rhode Island)	0: One World Trade Center
1: Humbug (The X-Files)	1: List of plants known as nettle (1)	1: Dax Shepard	1: Brown University	1: One World Trade Center (1)
2: Humbug (The X-Files)	2: List of plants known as nettle (1)	2: Dax Shepard	2: Brown University	2: One World Trade Center (1)
3: Gillian Anderson	3: List of plants known as nettle	3: Dax Shepard	3: Brown University	3: One World Trade Center (1)
4: Gillian Anderson	4: List of plants known as nettle (1)	4: Dax Shepard	4: Brown University	4: One World Trade Center (1)
5: Gillian Anderson	5: List of plants known as nettle (1)	**5: CHiPs (film)**	5: Brown University	5: One World Trade Center (2)
6: Gillian Anderson	6: Solanum rostratum	6: CHiPs (film)	6: Brown University	6: One World Trade Center (2)
7: Gillian Anderson	**7: Solanum**	7: CHiPs (film)	**7: Providence, Rhode Island**	7: One World Trade Center (2)
8: Gillian Anderson		8: CHiPs (film)	8: Providence, Rhode Island	8: One World Trade Center (3)
9: Gillian Anderson		9: CHiPs (film)	9: Providence, Rhode Island	9: One World Trade Center (3)
10: Gillian Anderson		10: Dax Shepard	10: Providence, Rhode Island	10: One World Trade Center (3)
11: Gillian Anderson		11: Dax Shepard	11: Providence, Rhode Island	11: One World Trade Center (3)
12: Gillian Anderson		12: Dax Shepard	12: Providence, Rhode Island	**12: World Trade Center (1973–2001)**
13: Gillian Anderson		13: Dax Shepard	13: Providence, Rhode Island	13: One World Trade Center
14: Gillian Anderson		14: Dax Shepard	14: Providence, Rhode Island	14: One World Trade Center (1)

Figure 3: Successful navigation trajectories on the FEVER (Thorne et al. [2018]) evaluation set. Rows from top: claim to be verified; starting node (from BM25); ground truth target (not visible to agent); target node text; agent trajectory.

References

D. Chen, A. Fisch, J. Weston, and A. Bordes. Reading wikipedia to answer open-domain questions. *CoRR*, abs/1704.00051, 2017.

R. Das, S. Dhuliawala, M. Zaheer, L. Vilnis, I. Durugkar, A. Krishnamurthy, A. J. Smola, and A. McCallum. Go for a walk and arrive at the answer: Reasoning over paths in knowledge bases using reinforcement learning. *CoRR*, abs/1711.05851, 2017.

C. Fu, T. Chen, M. Qu, W. Jin, and X. Ren. Collaborative policy learning for open knowledge graph reasoning. *arXiv preprint arXiv:1909.00230*, 2019.

L. He, B. Shao, Y. Xiao, Y. Li, T.-Y. Liu, E. Chen, and H. Xia. Neurally-guided semantic navigation in knowledge graph. *IEEE Transactions on Big Data*, 8(3):607–615, 2022.

V. Karpukhin, B. Oguz, S. Min, L. Wu, S. Edunov, D. Chen, and W. Yih. Dense passage retrieval for open-domain question answering. *CoRR*, abs/2004.04906, 2020.

D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

A. Lazaridou, E. Gribovskaya, W. Stokowiec, and N. Grigorev. Internet-augmented language models through few-shot prompting for open-domain question answering, 2022.

F. leaderboard. FEVER development set leaderboard. `https://competitions.codalab.org/competitions/18814#results`, 2022.

X. V. Lin, R. Socher, and C. Xiong. Multi-hop knowledge graph reasoning with reward shaping. *arXiv preprint arXiv:1808.10568*, 2018.

Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov. Roberta: A robustly optimized BERT pretraining approach. *CoRR*, abs/1907.11692, 2019.

R. Nakano, J. Hilton, S. Balaji, J. Wu, L. Ouyang, C. Kim, C. Hesse, S. Jain, V. Kosaraju, W. Saunders, X. Jiang, K. Cobbe, T. Eloundou, G. Krueger, K. Button, M. Knight, B. Chess, and J. Schulman. Webgpt: Browser-assisted question-answering with human feedback. *CoRR*, abs/2112.09332, 2021.

R. Nogueira and K. Cho. Webnav: A new large-scale task for natural language based sequential decision making. *CoRR*, abs/1602.02261, 2016.

Y. Qu, Y. Ding, J. Liu, K. Liu, R. Ren, X. Zhao, D. Dong, H. Wu, and H. Wang. Rocketqa: An optimized training approach to dense passage retrieval for open-domain question answering. *CoRR*, abs/2010.08191, 2020.

S. Robertson and H. Zaragoza. The probabilistic relevance framework: Bm25 and beyond. *Foundations and Trends® in Information Retrieval*, 3(4):333–389, 2009. ISSN 1554-0669. doi: 10.1561/1500000019. URL `http://dx.doi.org/10.1561/1500000019`.

V. Sanh, L. Debut, J. Chaumond, and T. Wolf. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. *ArXiv*, abs/1910.01108, 2019.

S. J. Semnani and M. Pandey. Revisiting the open-domain question answering pipeline. *CoRR*, abs/2009.00914, 2020.

Y. Shen, J. Chen, P.-S. Huang, Y. Guo, and J. Gao. M-walk: Learning to walk over graphs using monte carlo tree search. *Advances in Neural Information Processing Systems*, 31, 2018.

D. Stammbach. Evidence selection as a token-level prediction task. In *Proceedings of the Fourth Workshop on Fact Extraction and VERification (FEVER)*, pages 14–20, Dominican Republic, Nov. 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.fever-1.2. URL `https://aclanthology.org/2021.fever-1.2`.

D. Stammbach and G. Neumann. Team DOMLIN: Exploiting evidence enhancement for the FEVER shared task. In *Proceedings of the Second Workshop on Fact Extraction and VERification (FEVER)*, pages 105–109, Hong Kong, China, Nov. 2019. Association for Computational Linguistics. doi: 10.18653/v1/D19-6616. URL `https://aclanthology.org/D19-6616`.

A. Talmor and J. Berant. The web as a knowledge-base for answering complex questions. *CoRR*, abs/1803.06643, 2018.

J. Thorne, A. Vlachos, C. Christodoulopoulos, and A. Mittal. FEVER: a large-scale dataset for fact extraction and VERification. In *Association for Computational Linguistics; dataset released under Apache license.*, June 2018.

I. Turc, M. Chang, K. Lee, and K. Toutanova. Well-read students learn better: The impact of student initialization on knowledge distillation. *CoRR*, abs/1908.08962, 2019. URL `http://arxiv.org/abs/1908.08962`.

L. Wang, Y. Li, Ö. Aslan, and O. Vinyals. Wikigraphs: A wikipedia text - knowledge graph paired dataset. *CoRR*, abs/2107.09556, 2021.

R. West and J. Leskovec. Human wayfinding in information networks. In *WWW*, page 619–628, 2012a.

R. West and J. Leskovec. Automatic versus human navigation in information networks. In *ICWSM*, 2012b.

R. West, J. Pineau, and D. Precup. Wikispeedia: An online game for inferring semantic distances between concepts. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI)*, 2009.

W. Xiong, T. Hoang, and W. Y. Wang. Deeppath: A reinforcement learning method for knowledge graph reasoning. *arXiv preprint arXiv:1707.06690*, 2017.

M. Zaheer, G. Guruganesh, K. A. Dubey, J. Ainslie, C. Alberti, S. Ontanon, P. Pham, A. Ravula, Q. Wang, L. Yang, et al. Big bird: Transformers for longer sequences. *Advances in Neural Information Processing Systems*, 33:17283–17297, 2020.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes]
   (c) Did you discuss any potential negative societal impacts of your work? [N/A]
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A]
   (b) Did you include complete proofs of all theoretical results? [N/A]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] The paper and supplemental provide full details on how to reproduce the simple approach. The datasets used are all public.
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] In appendix.
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [N/A] Our experiments are on large datasets and thus did not require multiple runs.
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [No] The experiments were run on a proprietary infrastructure on which it is hard to estimate the total amount of compute.
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes]
   (b) Did you mention the license of the assets? [Yes] See Thorne et al. reference.
   (c) Did you include any new assets either in the supplemental material or as a URL? [No]
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]