WALDO: Wikipedia Agent for Learning Definition-informed Objectives

Diego Escobedo and **Noah Lee** and **Ritaank Tiwari**
{diegoesc, noahlee, ritaank}@mit.edu

Abstract

The Wiki Game is a popular web game where players are given a starting Wikipedia article and a goal Wikipedia article, and are tasked with reaching the goal by clicking the fewest possible article links. To engage with this game, we present RiTAANQ, a reinforcement learning agent capable of navigating graphs where traversal algorithms are computationally infeasible, by using textual representations to inform decision-making. Additionally, we present the novel and easily-extendable WikiGame Graph (*WiGG*) environment, an RL dataset that evaluates the effectiveness of agents in incorporating semantic information in graph-related tasks.

1 Introduction

An increasingly popular intersection of research areas in computer science has been the marriage of Reinforcement Learning and Natural Language Processing. We have seen a variety of creative combinations of techniques, including using textual descriptions to assist in transfer learning for video games(Narasimhan et al., 2018). (mention stuff we have seen in class).

For the authors, the ultimate combination of these two areas is The Wiki Game, a classic time-killing game where players are required to navigate between two Wikipedia pages using the fewest clicks possible. To be successful at the game, players must succesfully connect related concepts, use advance planning and strategizing, and filter information. The natural language conceptual understanding required to perform well at the game is non-trivial, and one that would be hard a computer to emulate. Additionally, the game happens to be extremely well suited for reinforcement learning. This game is essentially finding the shortest path between two nodes, which is an extremely well studied problem that has countless solutions and implementations. However, due to the nature of this specific graph (Wikipedia is massive, it is constantly getting updated, and it has a high degree of connectivity), traditional graph algorithms may not be well suited for this task. This makes the Wiki Game an optimal environment to explore the intersection of NLP and RL.

We present two main innovations in our paper. First, we present the WikiGame Graph (*WiGG*) environment, where developers can test custom Reinforcement Learning algorithms and interact with different variations of the Wiki Game. Second, we present WALDO (Wikipedia Agent for Learning Definition-informed Objectives), a Deep Q-network based and textually informed agent for *WiGG*.

The paper is organized as follows: Section 2 discusses works related to the problem setting, and to the intersection of NLP and RL. Section 3 presents formulation of the problem. Section 4 discusses implementation details, including WALDO's structure and training details. Section 5 shows our results. Section 7 discusses further work. All code for the project can be found at `https://github.com/ritaank/wikigame`.

2 Relevant Works

2.1 Wikipedia Navigation

(Patel and Weiss, 2019)
From the University of Waterloo 2017 Hackathon, the authors present a Wikipedia game bot in the form of a python script, utilizing web apps to solve the Wikipedia game. They construct a representation using ChartJS as the path between pages, and to represent the proximity at each stage.

(Banerjee et al., 2007)
This paper uses Wikipedia to assist in a clustering task. Clustering is a technique used to group together similar pieces of information by topic to make the information more manageable for a user. The overall goal is to draw different conceptual

representations from the articles to provide more descriptive features in clustering. The paper proposes a method of improving the process of clustering short texts by having the model learn additional features from Wikipedia. The model learns a Lucene index of Wikipedia articles from the English 2006 version. The results demonstrate that enriching the learned representation of the text will improve the accuracy of the clustering when compared to the baseline, conventional continuous-bag-of-words learned representation.

(Barron and Zswaff, 2015)
This paper employs unsupervised learning to build features representing Wikipedia articles. They parsed an XML dump of the Wikipedia site into a series of pages. Then, they used the hyperlink information to form a directed graph, of page nodes and link edges. Each page was featurized as a bag-of-words. Using this simple structure, they compared performance of basic graph search algorithms with supervised and unsupervised machine learning approaches that approximated the link-distance between two articles. The logistic regression approach had the highest pure accuracy at 55%. Finally, they combine multinomial logistic regression as a heuristic with a graph search algorithm. This combination minimizes the number of states explored, demonstrating success of the machine learning heuristic. The learning of higher level concepts demonstrated by the machine learning heuristic such higher level articles should be useful in linking diverse concepts and thus providing minimal path solutions in the game.

(Milne and Witten, 2008)
This paper uses machine learning to automatically cross-reference documents with Wikipedia. The authors use machine learning to identify significant terms within unstructured text, and enrich it with links to the appropriate Wikipedia articles. The paper focuses on ML approaches to two tasks, disambiguation and detection. The resulting link detector and disambiguator perform very well, with recall and precision of almost 75%. This performance is constant whether the system is evaluated on Wikipedia articles or "real world" documents. This process is known as wikification, and this approach differs from previous attempts in that Wikipedia is used not only as a source of information to point to, but also as training data for how best to create links. This work has implications far beyond enriching documents with explanatory links. It can provide structured knowledge about any unstructured fragment of text. Any task that is currently addressed with bags of words – indexing, clustering, retrieval, and summarization to name a few – could use the techniques described here to draw on a vast network of concepts and semantics.

(Yazdani and Popescu-Belis, 2011)
This paper proposes a method for computing semantic relatedness between two texts based on learning knowledge from Wikipedia. The authors heavily utilized the links in each article for classification, which boded well for our link-based approach to explore the structure of Wikipedia. A graph-based distance between Wikipedia articles is defined using a random walk model, which estimates visiting probability (VP) between articles using two types of links: hyperlinks and lexical similarity relations. The VP to and from a set of articles is then computed, and approximations are proposed for the semantic relatedness between every two texts in a large data set. The model is then applied to document clustering on the 20 Newsgroups dataset with the goal to classify the dataset into 20 existing document classes. The use of the VP metric increases both precision and recall compared to other approaches.

2.2 Reinforcement Learning

(Li, 2017)
This paper develops an overview on the state of deep reinforcement learning (the marriage of neural networks and reinforcement learning). It discusses six core elements, six important mechanisms, and twelve applications of deep RL. It opens by discussing the development of deep learning and reinforcement learning, moves to discussing core RL elements – value functions, Deep Q-Networks (DQN), policies, rewards, models, planning, and exploration. Then, the paper touches on important mechanisms for RL, including attention and memory, unsupervised learning, transfer learning, multi-agent RL, and hierarchical RL. Finally, the paper turns to various applications of RL, including games, robotics and natural language processing.

(Arulkumaran et al., 2017)
This paper discusses the ability of deep learning to enable reinforcement learning to scale to work on solving problems that were previously unreachable with high dimensional inputs. The paper introduces the general field of RL, and the distinctions of value-based and policy-based methods. Central algorithms in deep RL, including the DQN, trust region policy optimization (TRPO), and asynchronous advantage actor-critic are covered. Deep neural networks provide a unique advantage to focusing on visual understanding via RL.

2.3 Reinforcement Learning + NLP

(Narasimhan et al., 2018)
This paper explores natural language as a tool to encourage cross-task knowledge transfer in reinforcement learning. Specifically, researchers provides textual descriptions of the dynamics of different game environments to an RL model to facilitate policy transfer. This model learns to correlate the meaning of the textual descriptions to core dynamics of the game environment (such as non-player agents or obstacles). These learned mappings from text meaning to game dynamics encourages general policy representations that help the agent bootstrap policy learning in new game environments. The RL model consists of a v value-iteration network (VIN) planning module, policy component, and a factorized representation of the game environment and text descriptions. Results demonstrate the model's success beyond the baseline model in multiple game environments for both average and initial rewards.

3 Problem Formulation

3.1 Environment Framework

The Wikipedia game has a few properties that make it extremely easy to model as an MDP, which is the modeling tool that we decided to use. We define the environment as:

$$\mathcal{E} =< \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \mathcal{Z} > . \tag{1}$$

Here, $\mathcal{S}$ is the state space. We propose modeling Wikipedia as a graph with articles represented by nodes V and links between pages represented by directed edges E. For the Wikipedia game, at all times, we have a current node v_{curr} (the webpage we are currently on), and a goal node v_{goal} (the webpage we would like to get to). Therefore, the state space S can be represented as the tuple (v_{curr}, v_{goal}). Thus we have $|V|^2$ possible states.

$\mathcal{A}$ is the set of actions available to the agent and can be defined for a specific vertex $v \in V$ in the Wikipedia graph as $\mathcal{A}$ = `out_neighbors(`v`)`. The set of actions available to the agent is variable and entirely dependent on the current state. At any state, only a small subset of our total actions are available, and this proves to be an additional challenge of our environment. A common technique used in DQN methods is evaluating expected rewards over all possible actions, and simply masking the ones that aren't truly available. However, since we would have to output predicted rewards for the millions of Wikipedia articles, and only a few would actually be valid, we would be wasting valuable computational resources.

The transitions, encoded by $\mathcal{T}$, are actually trivial in this environment. $\mathcal{A}$ is a strict subset of $\mathcal{S}$, and transitions are completely deterministic (making an action always results in moving to the desired page, no complications).

$\mathcal{R}$ determines the reward at each time step and is defined as follows:

$$\mathcal{R} = \left\{ \begin{array}{ll} -1, & \text{if } v_{curr} \neq v_{goal} \\ 1, & \text{if } v_{curr} = v_{goal} \end{array} \right\}. \tag{2}$$

$\mathcal{Z}$ is an augmentation we are using in order to make our model novel. One might wonder why RL is needed to navigate the *WiGG* environment, if there are decades of research on graph traversal algorithms. However, it is important to realize that these algorithms simply become infeasible when the graph becomes the size of Wikipedia, with over 6 million articles. Therefore, we need some sort of approximation, which RL provides perfectly. We settled on using text-informed decision-making, by not only incorporating the identity of the *WiGG* nodes, but also making word representation vectors of article titles available to our agent. We discuss this in more detail later, but $\mathcal{Z}$ gives rich embeddings for article titles to our agent, allowing it to make definition-informed decisions instead of relying on brute-force graph algorithms or memorization of infeasibly large state spaces.

3.2 Objective

Our aim will be to train a policy that tries to maximize the discounted, cumulative reward

$$\mathcal{R}_{t_0} = \sum_{t=t_0}^{\infty} \gamma^{t-t_0} r_t. \tag{3}$$

Here γ is the discount factor, which represents the standard Bellman concept to realize rewards sooner rather than later. The main idea behind Q-learning is that if we had a function Q^* : State $\times$ Action $\rightarrow \mathcal{R}$, that could tell us what our reward is for any state-action pair, then we could easily construct a policy $\pi^*(s)$ that maximizes our rewards: by always picking the action a that maximizes the reward:

$$\pi^*(s) = \text{argmax}_a \ Q^*(s, a)$$

Since we don't know everything about the environment, we have to approximate Q^* using a neural network, and use the fact that, given a policy π, every Q function obeys the Bellman equation to optimize the weights:

$$Q^\pi(s, a) = r + \gamma Q^\pi(s', \pi(s'))$$

Here, s' represents the state we transition to after action a. The difference between the two sides of the equality is known as the temporal difference error, δ:

$$\delta = Q(s, a) - (r + \gamma \max_a Q(s', a))$$

To minimize this error, we will use the *Huber loss*, which acts like the mean squared error when the error is small, but like the mean absolute error when the error is large - this makes it more robust to outliers when the estimates of Q are very noisy. We calculate this over a batch of transitions, B, sampled from a replay buffer:

$$\mathcal{L} = \frac{1}{|B|} \sum_{(s,a,s',r) \in B} \mathcal{L}(\delta)$$

$$\text{where} \quad \mathcal{L}(\delta) = \begin{cases} \frac{1}{2}\delta^2 & \text{for } |\delta| \leq 1, \\ |\delta| - \frac{1}{2} & \text{otherwise.} \end{cases}$$

4 Implementation Details

4.1 Model Architecture

Our DQN has a very simple architecture, with 2 hidden layers with 1024 and 256 units each, and ReLU nonlinearities in between. However, most of the complexity of the model comes from generating the word embeddings for the article titles.

We used a pre-trained distilBERT tokenizer and model from (Sanh et al., 2019a). We will leave the finer details of the training procedures and architectural choices for these pretrained models to the original BERT paper (Devlin et al., 2018) and the paper discussing the distilled model (Sanh et al., 2019b).

The essence of BERT is based on the Transformer architecture (Vaswani et al., 2017). Since we are only interested in generating feature representations, and not on a text generation objectives, we only use the encoder component of a Transformer. Additionally, BERT trains on two objectives. In the Masked Langauge Model (MLM) Objective, the model processes the input as a whole (instead of sequentially) to predict masked out parts of a sentence. Even though it converges slower because thus it only uses the masked tokens to update gradients, the training procedure allows it to generate context-aware predictions, which is important in article titles where a single word can have different meanings (think of all the disambiguation pages on Wikipedia!). In the Next Sentence Prediction objective, BERT is forced to determine whether a pair of sentences actually follow each other, which is an easy semi-supervised task to train on.

We use the pre-trained distilBERT to generate 768-dimensional word embeddings for the article titles. As mentioned above, we want to estimate $Q(s, a)$, and since s is composed of both a current page and a goal page, and the action is yet another page, we concatenate the representations for these three articles and therefore have an input layer of dimension 2304 into our Q-network.

4.2 Training Details

We present our training algorithm RiTAANQ (Reinforcement-Informed Training Autonomous Agents on Neighbors through Q-Learning) in Algorithm 1. The hyperparameters we used were a maximum episode length of 10, 300 episodes, 1000 transition capacity on the replay buffer, γ of 0.999, a minibatch size of 16, and a learning rate of $3e-4$. A couple of other tricks that we used to train the model that aren't reflected in the algorithm include:

- Exponentially decaying ϵ exploration value, starting at 0.9 at episode 0 and 0.05 at step (NOT episode) 1000.
- Differentiation between a policy Q-network

and a target Q-network, with the target Q-network being updated to the policy one every 20 episodes. This helps stabilize learning and prevents cycles and getting caught in suboptimal places in the loss landscape.

- Allowing any source and any target node requires significantly more computational power, which we unfortunately did not have. Therefore, we decided to fix the destination node to be the MIT Wikipedia page (https://en.wikipedia.org/wiki/Massachusetts_Institute_of_Technology), and pruned the graph to only include nodes that were within a maximum of 4 transitions away (98.3% of nodes remained after pruning).

- We read a great paper (Hertz et al., 2021) that discussed how neural nets often struggle to learn the high frequencies of the functions they are approximating, something they propose fixing by progressively revealing frequencies from low to high to a neural net during the training process. We figured the analog of this in our context is to first show the neural net the easiest transitions to learn (ones that directly result in getting to the goal node) instead of immediately filling the buffer with really bad transitions where we never get the desired reward. We set the maximum allowable BFS distance of the source node to the target node on an exponential schedule, starting with 1 at episode 0 and reaching the asymptote of 4 at episode 300.

5 Evaluation Metrics

5.1 Win rate

Win rate is calculated as the proportion of episodes where our model successfully reaches the goal node within the step limit. Ultimately our goal is for our model to win the game every time, so the win rate should give us a good sense of our model's robustness.

5.2 Avg Distance Ratio

For each new game, we calculate the best path using Djikstra's algorithm, and store that length. We think of this as a distance that we are trying to minimize. For games where we win, the distance ratio is 0. For a game where we do not win, we evaluate our final game state. Specifically, we see if the shortest-path distance between the final state and the goal state is *less* than the initial best path length. If so, that means that WALDO was travelling in the right direction.The Avg Distance Ratio is the ratio between the initial best path length, and the path between the final state and goal state.

5.3 Delta Cosine Similarity

$$\cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2} \sqrt{\sum_{i=1}^{n} B_i^2}}$$

Figure 1: The cosine similarity equation

BERT embeds the essence of the Wikipedia article names into densely bound vectors. This tensor represents a token, which we use to generate semantic designs of the article names. The cosine similarity calculates the angular distance between two vectors, independent of magnitude. The cosine similarity is a representation of how close two different word representations are. We calculate the Delta Cosine Similarity, which is the change in cosine similarity from the initial state, to the final state of the model.

6 Results

We now present results from three different learning conditions. Through these learning conditions, we maintained a consistent batch size of 32.

The first model we present is called 'TrivialJesus'. This model was trained on a subgraph that had a maximum BFS radius of 2 from the 'Jesus' Wikipedia page, which is the most linked-to and linked-from page in the English Wikipedia. The destination node was fixed to be 'Jesus' in every episode. We call this 'trivial' as the BFS radius for the graph was very small. With a BFS radius of 2, the start node is either one, or two clicks away from the destination node. In the case that the start node is one click away, the game is essentially already over because it can find the goal node link and navigate to it immediately. In the case that the start node is two clicks away, the model has to work a little harder, but it already has learned that finding

Algorithm 1 RiTAANQ: Reinforcement-informed Training for Autonomous Agents on Neighbors through Q-Learning

Initialize replay memory $\mathcal{D}$ to capacity C
Initialize action-value function Q with random weights
Initialize pre-trained BERT embedder ϕ
Initialize WikiGameGraph Environment with out-neighbor function N
for episode $= 1, M$ **do**
  Initialise starting node v_{curr} and goal node v_{goal}
  for $t = 1, T$ **do**
    With probability ϵ select a random action a_t
    otherwise select $a_t = \max_a Q^*(\phi(v_{goal}), \phi(v_{curr}), \phi(a); \theta)$
    Execute action a_t in emulator and observe reward r_t
    Store transition $(v_{curr}, a_t, r_t, \phi(v_{goal}))$ in $\mathcal{D}$
    Set $v_{curr} = a_t$
    Sample random minibatch of transitions $(v_j, v_{j+1}, r_j, \phi(v_{goal,j}))$ from $\mathcal{D}$
    Set $r_{j,\text{expected}} = \max_{a_j \in N(v_j)} Q^*(\phi(v_{goal,j}), \phi(v_j), \phi(a_j)); \theta)$
    Set $r_{j+1,\text{expected}} = \max_{a_{j+1} \in N(v_{j+1})} Q^*(\phi(v_{goal,j}), \phi(v_{j+1}), \phi(a_{j+1}); \theta)$
    Set temporal difference $\delta = r_{j,\text{expected}} - (r_j + \gamma * r_{j+1,\text{expected}})$
    Perform a gradient descent step on the Huber Loss of δ
  end for
end for

any node that is one click away leads to an instant win. This was a very simplified model, serving as a proof of concept for us that our model was learning properly. The near perfect success rate of 0.98, zero dead end rate, low average distance from goal node all indicate to us that our model is learning on this very straightforward environment. This is a great building block for us to challenge our model to explore more complex environments.

Destination node	Batch size	BFS radius	Success rate	Dead end rate	Avg distance from goal node	Avg cosine similarity
Trivial Jesus	8	2	0.98	0	0.017	0.374

Figure 2: Results for our "Trivial Jesus" model

Now that we demonstrated a proof of concept that our model was able to learn on a simplified algorithm, we increased the batch size to 32, and maximum BFS radius to 4. The second model we present is called 'FixedMIT'. This model was trained sequentially on concentric rings of subgraphs, using a BFS expansion. We utilized the expanding BFS procedure as we described in Section 4.2 to ensure that the low-level features are learned first before expanding outwards. Our BFS distance grows to 4 here, creating a graph of over 1.3 million nodes. For our results, we segmented evaluation based on how far the source node was from the fixed destination node, which was the MIT page. This allows for more granular evaluation of performance instead of random selection of source nodes for evaluation. Random selections are often biased to nodes of distance 1, since MIT also is very highly linked-to. We note that for level 1, the accuracy was 100%, since the expanding-BFS strategy ensured that all direct neighbors learned to maximize rewards from travelling to MIT. We see that level 4 actually has a higher accuracy than level 3. When performing analysis, we realize that the total number of nodes in level 4 is much less than level 3, and our exponential growth for BFS spends the most time training on the largest graphs. Thus, the training time per node, roughly speaking, is higher for the nodes in level 4. We surmise that this creates the improved accuracy for level 4, and we speculate that increased training episodes would lead to more accurate reward functions and improved test accuracy across all levels.

The third model we present is called "Jesus Non-Explorer." We created a setup similar to TrivialJesus, however, we changed our exponential epsilon to converge much faster, which means the exploration phase was cut short. We see here decreased accuracy as expected, and we see this issue is exacerbated by distances of even 2. Even at distance 1, our lack of exploration prevents us from winning with certainty.

Model Name	Level	Node count	Batch size	BFS radius	Success rate	Dead end rate	Avg distance ratio	Delta cosine similarity
Fixed MIT	1	1,960	32	4	1.0	0	0	0.384
	2	71,240	32	4	0.836	0	0.187	0.356
	3	771,885	32	4	0.771	0	0.215	0.331
	4	378,573	32	4	0.786	0	0.162	0.348

Figure 3: Results for our "Fixed MIT" model

Model Name	Level	Node count	Batch size	BFS radius	Success rate	Dead end rate	Avg distance ratio	Delta cosine similarity
Jesus Non-Explorer	1	3,354	64	2	0.627	0	0.552	0.189
	2	191,602	64	2	0.030	0	0.949	0.045

Figure 4: Results for our "Jesus Non-explorer" model

7 Further Work

In this project, we were mainly constrained by the time we had to complete the project. A few of the ideas we would have liked to explore would have been:

- We would have loved to explore different ways of embedding articles. Perhaps comparing the performance of our model using the different embedding schemes could have given some interesting insights into how important the embeddings are to the success of the model.

- Obviously, some more searching over hyperparameters and seeing what kind of size of the replay buffer and number of episodes would have led to better training. RL is unique in that the emulator can give us almost unlimited training data, which is the only proven way to monotonically improve the performance of models.

- Being able to train long enough that we don't have to fix the destination node would be great! That would pave the way for a fully autonomous model that can perform well on an unrestricted version of the game. We would like to explore with a subgraph based around the Jesus node, and with a non-fixed destination node, analyze whether the model learns to 'go to Jesus', exploiting the supernode in the graph's creation.

- Being able to incorporate more context around the text where the link is located might lead to better performance.

- The way our code is currently written is inefficient in how it calculates the expected reward given the problem of a variable number of neighbors. Exploring different ways of performing these operations could lead to huge speedups.

In hopes of seeing this used as a benchmark for tasks involving the combination of NLP and RL, we plan to publish the *WiGG* environment on OpenAI's gym.

8 Conclusion

We have shown here that our reinforcement learning DQN (deep Q network) setup effectively is able to learn the traversal of the Wikipedia game when trained on a single destination node. Over time, with an expanding BFS, the rewards propagate outwards, improving accuracy even for long paths between source and destination nodes. We see that our algorithm getting closer to the destination node is not necessarily correlated with a decreasing cosine similarity, as it is not a given truth that all articles with similar title embeddings link each other. in terms of BERT representation to the destination node.

References

Kai Arulkumaran, Marc Peter Deisenroth, Miles Brundage, and Anil Anthony Bharath. 2017. Deep reinforcement learning: A brief survey. *IEEE Signal Processing Magazine*, 34(6):26–38.

Somnath Banerjee, Krishnan Ramanathan, and Ajay Gupta. 2007. Clustering short texts using wikipedia. In *Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 787–788.

Alex Barron and Zack Swafford Zswaff. 2015. An ai for the wikipedia game.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805.

Amir Hertz, Or Perel, Raja Giryes, Olga Sorkine-Hornung, and Daniel Cohen-Or. 2021. Progressive encoding for neural optimization. *CoRR*, abs/2104.09125.

Yuxi Li. 2017. Deep reinforcement learning: An overview. *arXiv preprint arXiv:1701.07274*.

David Milne and Ian H Witten. 2008. Learning to link with wikipedia. In *Proceedings of the 17th ACM conference on Information and knowledge management*, pages 509–518.

Karthik Narasimhan, Regina Barzilay, and Tommi Jaakkola. 2018. Grounding language for transfer in deep reinforcement learning. *Journal of Artificial Intelligence Research*, 63:849–874.

Vasim Patel and Eric Samuel Weiss. 2019. Wikipediagame. https://github.com/VasimPatel/WikipediaGame.

Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. 2019a. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. *ArXiv*, abs/1910.01108.

Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. 2019b. Distilbert, a distilled version of BERT: smaller, faster, cheaper and lighter. *CoRR*, abs/1910.01108.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *CoRR*, abs/1706.03762.

Majid Yazdani and Andrei Popescu-Belis. 2011. Using a wikipedia-based semantic relatedness measure for document clustering. Technical report.