ADVICE-DRIVEN LEARNING: A BLOCKS-WORLD AND FAKE NEWS DETECTION APPROACH

A Thesis

Submitted to the Faculty

of

Purdue University

by

Nikhil Mehta

In Partial Fulfillment of the

Requirements for the Degree

of

Master of Science

December 2020

Purdue University

West Lafayette, Indiana

THE PURDUE UNIVERSITY GRADUATE SCHOOL
STATEMENT OF THESIS APPROVAL

Dr. Dan Goldwasser, Chair

Department of Computer Science

Dr. Jennifer Neville

Department of Computer Science

Dr. Clifton W. Bingham

Department of Computer Science

Approved by:

Dr. Kihong Park

Head of the School Graduate Program

This is dedicated to the most important people in my life, my family. Anil Mehta, Archana Mehta, Sonal Mehta, and Ambika Mehta.

ACKNOWLEDGMENTS

I would like to start by expressing deep gratitude to my advisor, Professor Dan Goldwasser, as without him anything presented in this work would be not be possible. He was the one who first introduced me to the field of Natural Language Processing, giving me an opportunity as an undergraduate to take his graduate course, despite it being full. He then further gave me a chance to work/research under him, which I continued throughout graduate school. Throughout my time working with Prof. Goldwasser, he has always been available and providing me with thoughtful/helpful ideas on whatever questions I had, while allowing me the freedom to tackle the problems I was interested in. He has been a constant support guiding me through the challenges of my graduate and later part of my undergraduate career, and for that I will never be able to thank him enough.

I would also like to thank Professor Christopher Clifton and Professor Jennifer Neville, for agreeing to be on my thesis committee and providing me feedback to make this work even stronger.

I am also thankful to my family for constantly supporting me since I stepped foot on Purdue's campus and before. Finally, I am thankful for my friends and the people I have met throughout my college career, specifically Joe Chastain, Joshua Leeman, Naman Patwari, and Arbazz Mukadam for making my time at Purdue a lot more enjoyable.

TABLE OF CONTENTS

Page

LIST OF TABLES

LIST OF FIGURES

ABBREVIATIONS

BERT	Bidirectional Encoder Representations from Transformers
FC	Fully Connected Layer
NLP	Natural Language Processing
SOTA	State of the Art

ABSTRACT

Mehta, Nikhil M.S., Purdue University, December 2020. Advice-Driven Learning: A Blocks-World and Fake News Detection Approach. Major Professor: Dan Goldwasser.

Over the last few years, there has been growing interest in learning models for various Natural Language Processing tasks, such as the popular blocks world domain and fake news detection. These works typically view these problems as single-step processes, in which a human operator gives an instruction and an automated agent is evaluated on its ability to execute it. In this work, we take the first step towards increasing the bandwidth of this interaction, and suggest a protocol for including advice, high-level observations about the task, which can help constrain the agents prediction. Advice is designed to be a short natural language sentence, provided by the human operator in addition the original input for the task, that can help the agent improve its performance. We evaluate our advice-based approach on the blocks world task and fake-news detection, and show that even simple advice can help lead to significant performance improvements. To help reduce the effort involved in supplying the advice, we also explore model self-generated advice which can still improve results.

1. INTRODUCTION

Over the last few years, especially with the rise of big data, machine learning has become increasingly popular. One of the most common forms of machine learning is supervised learning, where a model learns how to perform a task on some input data, getting supervision via labels. The goal of the model is generalization, or the ability to perform well on unseen data. Despite it's wide use, supervised learning has some issues. First, the model receives feedback only via labels, which are extremely challenging to obtain. Humans have spent large amounts of money to employ other humans to create large scale datasets [1, 2], but even still, there are many problems that existing datasets don't address. Thus, even though models achieve good performance on these datasets, it doesn't mean they actually solve the task well [3], which means more money must be spent to collect a more representative dataset (which could end up still being exploited by the model in some way). Second, in supervised learning, once a model is trained and deployed in the real world, it is not easy to update it. Updates are typically made by receiving/collecting more data and then fine-tuning the model, but this process is not trivial. Third, learning from labels only is not how humans learn. Humans learn knowledge from a wide variety of ways, including interaction with other humans, which raises the question of why we train machines to learn from just labels.

Interactive learning has the potential to fix some of these issues. In interactive learning, an AI system can learn from the human through a back and forth interaction. This means that the model can receive supervision even after it is trained, as humans can constantly teach it to do better. Even more, since humans are involved in the training process, humans can identify problem areas for the model and attempt to fix them, through various forms of input feedback, such as labels. This can help the

model in multiple ways, such as increasing the prediction accuracy or reducing the training time.

One form interactive learning is dialogue systems. Over the last few decades, there has been a lot of work in Natural Language Processing to build interactive dialog systems to accomplish various tasks. [4] proposed the TRAINS project in the 1990s, which was a long term effort to build a system that can use conversation to interact with and assist humans to solve various tasks. [5] proposed a system to train a robot interactively by teaching it various tasks while it observes a human perform them. More recently, [6] focuses on building a model that can accomplish tasks while conversing naturally with humans. Dialogue systems like these have a common theme, which is that humans provide input through various means (typing, speech, a graphical user interface, etc) and the AI system translates it to text using an input recognizer/decoder. Then, the system must understand the text, optionally execute some task, and finally produce some output. This process can then repeat.

Despite all of this work in interactive dialog systems (and interactive learning), there are many Natural Language Processing tasks that are still considered as single-step processes, and not viewed interactively. For example, in the popular blocks world task proposed by [7], a human must provide an instruction to an AI agent to move a block on a grid from a source location to a target destination. In the way this task is traditionally solved, the agent receives the instruction, understands it, executes it, and makes a prediction. There is currently not an easy way for the human to interact the agent to provide it some useful information that it can take advantage of to make a better prediction.

Similarly, fake news detection is a task that has become very popular recently, due to the rise of social media. Due to this, anyone can post about anything, and false information (that is often more interesting/controversial) can be quickly spread [8]. When people aim to build systems to detect fake news, they focus on designing models to predict whether an article/source is fake or not. However, there currently isn't an easy way for a human to provide knowledge they have about an article or source that

the model can utilize to improve its performance. Thus, even this task is typically not interactive.

Despite both of these tasks being vastly different for various reasons such as time proposed, domain, data used, etc., we aim to better solve them in a uniform way. Both of these tasks are currently being-solved in a single-step way, where the model receives some input and makes a prediction, where we aim to solve them *interactively*. Both tasks can also be decomposed into several problems, such as determining the validity of individual statements in a potentially fake news article. We take advantage of this decomposability in our work.

In this work, our goal is to explore different approaches for relaxing the typical single-step nature of many NLP tasks by introducing *advice-driven learning* (learning by receiving *advice*). We aim to view NLP tasks as more of an *interactive* process, in which the human operator can observe the agents' predictions on a task and adjust it by providing *advice*, a form of online feedback. Specifically, the advice consists of a short natural language sentence, one that the agent can easily understand, and take advantage of to solve the original task better.

Advice is typically provided by a human that has some knowledge of the original task that the agent does not. It is an easy way for the agent to incorporate human feedback or human knowledge. For example, in the case of the blocks world-task, if the human knows the general region (lower left quadrant) of where the block must be moved, it can provide that information as advice to the system. Fig. 2.1 shows an example of this and uses the advice *"the target is in the lower left"*, to restrict the agents search space after observing the incorrect prediction placed the target block in the top half of the board.

The advice text is easier to understand than the original task, so that the agent can take advantage of it. Advice allows a slight decomposition of the task, typically restricting the solution space in some way (for example in the above blocks-world task, the advice is restricting the search space of where the block must be placed to the lower left). However, advice is vague enough that it only provides provides partial

information, since understanding the advice doesn't necessarily mean the system will solve the final task. Thus, the system must be built to best take advantage and learn from the advice so it can handle instances when advice is not provided.

Advice can also be self-generated by a model, in which case no human interaction is necessary. In this case, the AI system would generate and then provide itself the advice. This requires building an architecture to self-generate the advice, which we will discuss in later Chapters. Self-generated advice still has the properties of normal advice, the only difference is that it is generated by a model, not a human.

In the following sections, we will compare advice-driven learning (learning by receiving advice) to some traditional forms of learning, namely supervised learning, reinforcement learning, and active learning. Throughout the rest of this document, we will show how taking advantage of advice can allow us to better solve different NLP tasks, specifically on the blocks-world task and fake-news detection.

1.1 Advice vs. Supervised Learning

In Supervised Learning, a model is provided with a bunch of training examples and their labels. The model must make a prediction on the training examples and receives feedback from the training labels. The goal of the model is to generalize well, or perform well on unseen test examples.

Advice-driven learning is different from supervised learning because advice is not the final label. Rather, the advice text is a partial solution/hint about the task that enables the system to better get to the final label. Typically, advice restricts the solution space for the task in some way, but does not provide the answer, like is done in supervised learning. However, advice can be thought as providing some sort of supervision for the task. Advice can also be provided before the final prediction is made, to help the system. Due to these reasons, advice-driven learning can be combined with supervised learning.

1.2 Advice vs. Reinforcement Learning

In Reinforcement Learning, a model aims to take a series of actions in order to maximize a reward. The model receives the reward as feedback after executing the actions, and can take advantage of that to learn and achieve a better reward in the future.

Similar to the case when compared with supervised learning, advice is not the reward at the final step, but rather a hint on how to get to the final step better, typically from a human that has knowledge about the process. In this way, advice can also be combined with reinforcement learning.

1.3 Advice vs. Active Learning

Active Learning is a type of machine learning in which the model/AI agent can query/ask questions to the human to obtain knowledge. Active learning is also usually an interactive process, as discussed in [9], where they propose a model that can learn by repeatedly asking questions to a human operator.

Although advice text can be provided to a system that utilizes active learning when it asks for help, advice-driven learning is different from active learning. In advice-driven learning, the human observes the predictions of the model and decides what knowledge to provide the agent. This is different from active learning where the model is explicitly asking for help, usually throughout the learning process. The benefits of advice-driven learning is that the human can choose whatever information it wants to provide which is easier than in active learning, where the human must answer the specific question. The information can also be provided after the prediction is complete, and the model can still take advantage of it. In this way, advice-driven learning can be considered a complement to active-learning, as both can be done together without providing the final label to the system, and performance could still improve.

1.4 Next Chapter

In this chapter, we have introduced advice-driven learning, which is a way for a human to easily provide feedback to an AI system. This protocol utilizes advice, a short simple sentence provided by the human that the system can easily understand, and take advantage of to solve the original task better. Advice allows solving NLP tasks in a more *interactive* way that is also easy for humans, as they only need to provide information that they already have. Throughout the rest of this work, we will show how taking advantage of advice when solving the blocks-world task and fake news detection helps performance on those tasks. We hope to show the generalizability of advice-driven learning, by showing the various ways in which it can be applied to these popular traditionally single-step NLP tasks.

2. BLOCKS WORLD

2.1 Introduction

The problem of constructing an artificial agent capable of understanding and executing human instructions is one of the oldest long-standing AI challenges [7]. This problem has numerous applications in various domains such as planning, navigation, and assembly. Solving this problem can also help accommodate seamless interaction with personal assistants in many environments, such as offices, homes, or even out on the street in public.

One of the earliest AI problems designed to tackle this goal of having agents communicate with humans is the blocks-world problem [7]. This task involves a bunch of blocks placed on a grid, and a human operator provides instructions to the agent of how to move the blocks. The agent must understand, and then execute the instruction to solve two challenging problems: identify which block must be moved, and where it must be moved to. Blocks can be placed either on the grid, or on top of another block. However, blocks can't be moved from underneath another block.

Although this task has been around for many years, solving it is still challenging. Successfully solving this task requires an advanced level of reasoning and language understanding AI systems do not currently have, which is why AI agents' performance on this task is still worse than human performance. For this reason, we propose to better solve the traditional blocks-world problem by taking advantage of advice.

Throughout this chapter, we will propose four novel interactive advice-based protocols that can be applied to any robot communication architecture, ordered in terms of decreasing human effort. As expected, as human effort lessens, performance does worsen, but all protocol outperform our baselines with no advice.

We will also explore the notion of model self-generated advice, which significantly reduces/eliminates human effort. In this approach, a model is trained to automatically generate advice for a given scenario, based on the assumption that it is easier to solve the advice prediction sub-task rather than the final coordinate prediction task. We validate this assumption by developing a neural architecture to predict the advice and show it can help improve the overall prediction quality, despite having no human assistance.

2.2 Related Work

Human-robot Interaction has been a major research topic for a few decades now, starting with the SHRDLU system [7]. Since then, there have been multiple systems [10–16] built to try and tackle this problem.

[11] uses reinforcement learning with policy gradients to map natural language instructions to a sequence of actions. Similarly, [17] also presents a planning based model to infer the most likely set of constraints from natural language instructions. Building on the prior work, [13] focus on using semantic grammars to handle more complex instructions.

In addition to large amounts of work on assembly and planning style natural language instruction understanding tasks, there has been a lot of work on other forms of human robot interaction. [18] introduced CLEVR, a dataset where the model must answer a question about a computer generated image that has a bunch of objects with different sizes, shapes, and colors. [19, 20] introduced neural module networks to tackle this dataset. They build models that have different modules, each specifically designed to handle a part of the end-to-end task, which can then be combined together to achieve better performance.

[1] introduced NLVR, a dataset to identify whether a given sentence is describing an image or not. They followed it up with the more challenging NLVR2 [21], which aims to solve this same task, except this time the images are photographs rather than

computer generated scenes. Very recently, [22] released LXMERT, a large scale transformer model to achieve great performance on NLVR2 by using an object relationship encoder, a language encoder, and a cross-modality encoder trained on 5 pre-training tasks.

As discussed, there has been a lot of work in the past few decades on human robot interaction, starting with the blocks-world problem. While the blocks-world problem is still extremely challenging as will discuss in Section 2.3, the community has also been recently interested in other tasks like NLVR2. We will show the power of advice-driven learning to improve performance on blocks-world, and we feel that advice can be helpful on these more recent tasks as well.

2.3 Dataset Details

The blocks world task has been around since the 1970s [7]. However, in this chapter, we focus on the Blocks World dataset proposed by [23], specifically it's more challenging version, where all the blocks are unlabeled. In this dataset, given a bunch of unlabeled blocks on a grid that look identical, a human provides an instruction to move a block. The model must identify which block must be moved and where it must be moved to, just like the original blocks task.

This version of the blocks world task is more challenging for multiple reasons, as originally identified by [23,24], which we summarize here. First, the blocks are referred to by their complex spatial information, due to the fact that they are identical. This means phrases like "bottom right tower" or "below the L shape" are common, and it is challenging for AI systems to easily understand these. Second, the instructions in the dataset are not restricted, meaning that the vocabulary is diverse and a simple model such as a pattern-based one cannot comprehend the instructions. Third, the only label provided in the dataset is that of the source and target blocks. There is no other label information provided, such as the coordinate of the reference blocks (for example in the instruction "below the tower", the tower is the reference block). Finally, the

dataset is fairly small, with the dataset having 2,493 training, 360 development, and 720 test examples. This is relatively small compared to the large vocabulary prevalent in the instructions (1,172 tokens and an average sentence length of 23.5 words). All of these make the SOTA performance on this dataset well below the human performance, as evaluated by [23].

2.4 Advice for Blocks World

We devise two types of advice for the blocks-world task, that we feel will be helpful to a prediction agent, given that the human provides it. Both types of advice are designed to assist the agent by providing simpler instructions in addition to the original input. Both can also be easily provided interactively in a wide variety of ways, such as text (spoken or typed), a remote control, or any other means of communication.

The first type of advice, *restrictive advice*, informs the agent about the general region of the source / target coordinates, such as *top left* or *bottom right*. The regions are determined by dividing the grid into equally sized sections (two halves, four quadrants). This type of advice can be thought of as useful for restricting the search space of the agent, since when it is provided, the agent knows exactly which general region to search for the block in. An example of restrictive advice for a given scenario is shown in Figure 2.1.

The second type of advice, *corrective advice*, observes the agents predictions and determines which direction (up, down, left, right) they must be adjusted to get closer to the target. This type of advice is useful for narrowing down the agents prediction in a multi-step way.

In this work, we do not crowd-source these types of advice to feed into our models, but rather simulate it. Our advice sentences are generated by filling in appropriate regions/directions into varying sentences. For example, given an advice sentence placeholder, *The target is in the* _____, and a coordinate in the lower left, we would generate the restrictive advice sentence: *The target is in the lower left.* We can easily

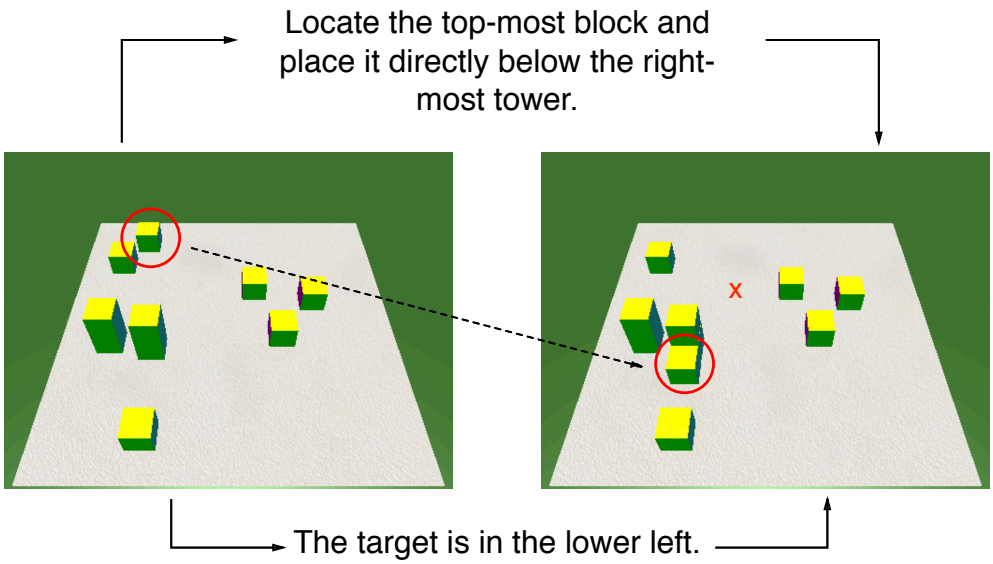


Fig. 2.1.: Based on the instruction (upper sentence) the model predicts the coordinates of the block and its target location. The 'x' represents an incorrect prediction, corrected by the provided advice (lower sentence).

do this in our experiments, as we have access to the true coordinate and the grid (however we will still not tell the system what the true coordinate is, just its' general region, like a human would when providing restrictive advice). At test time, we use variations of this sentence such as: *The block's region is the lower left*, to avoid memorization. We will later show in Table 2.2 the importance of our pre-trained advice grounding models (Sec 2.5.2) in enabling sentence variability and advice understanding (`M4` vs `M5`). Table 2.1 shows examples of some types of advice given the coordinates.

Table 2.1.: In the first row, given the target coordinate, the restrictive advice is provided to place the target in the appropriate region. In the second row, given the predicted coordinate and the target coordinate, the correct corrective advice is to move down to get closer to the target.

Predicted	**Target**	**Advice**
-	(-0.5, 0.5, 0.5)	In the top left.
(-0.5, 0.5, 0.9)	(-0.5, 0.5, 0.5)	Move down.

2.5 Models

2.5.1 Baseline Model

In this subsection, we briefly describe the best end-to-end RNN model proposed by [23]. Their model takes as input the sentence and world and predicts either the location of the block to move or its final location. It does this by processing the instruction through a RNN, and the world through a Fully Connected layer. It then uses the RNN representation to make two predictions, one for the offset and one for the reference block, using Fully Connected Layers. Finally, it uses the world state to adjust the reference prediction, adds that to the offset prediction, and uses a Fully Connected layer to get a final output coordinate. This model can be seen visually in Figure 2.2, and is the one that we apply advice to.

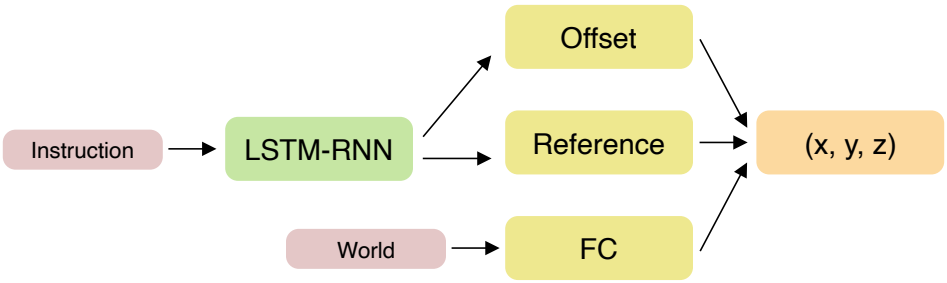


Fig. 2.2.: Baseline model proposed by [23]. All our advice models are built on top of this.

2.5.2 Advice Grounding

We pre-train a neural network model to accurately understand the advice (Fig. 2.3a). For both types of advice, a LSTM-RNN [25] is used to read the advice sentence $s = w_1, w_2, ..., w_n$ and output the hidden state representations $\{h_n\}$. Prior to this, a word embedding layer is used to project the input words into high-dimension vectors $\{w_i\}$.

For restrictive advice, the last state from the LSTM h_n is fed along with a random coordinate into a Fully Connected (FC) layer. The network must output a positive prediction if the random coordinate is in the region described by the advice sentence, and negative otherwise. This design allows the model to understand the meaning of the advice sentence by determining if the random coordinate follows the sentence.

Our architecture takes as input an advice sentence $s = w_1, w_2, ..., w_n$, passes it through a trained embedding layer of size 100, a LSTM of size 256, and outputs the hidden state representations $\{h_n\}$. The last hidden state h_n is embedded using a Fully Connected (FC) layer of size 100. Each axis of a random input coordinate (x, y, z) is also passed into the network and embedded using a FC layer of size 100. These 4 FC layers are summed up and passed through a final FC layer O of size 2, which is then followed by a softmax. All FC layers use the RELU activation function. We train this as a binary prediction problem using cross-entropy loss (shown in Equation 2.1, where p(x) is the true distribution/labels, and q(x) is the estimated distribution/prediction. For the binary case, this can be reduced and y is the true label, and p is the predicted value.), the Adam optimizer, and a learning rate of 0.001. Gradient clipping [26] with threshold 5.0 is used on the LSTM parameters to avoid exploding gradients.

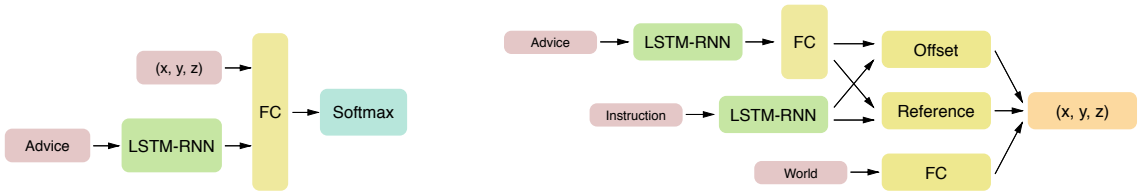
$$L = -\sum_{\forall x} p(x)log(q(x)) = (ylog(p) + (1-y)log(1-p)) \tag{2.1}$$

For corrective advice, the last state from the LSTM h_n (size 256) is fed along with a random coordinate into a FC layer (size 100), and the network must output a coordinate that follows the advice (3-dimensional output size). For example, if the advice is *move the block down*, the predicted coordinate must be below the random input coordinate. If the advice is followed, the network receives 0 loss, otherwise a MSE regression loss (Equation 2.2, where there are n examples, y is the true label, and p is the prediction), where the ground truth is some random coordinate that does follow the advice. The model and its parameters are identical to the one for

restrictive advice, except the final FC layer O has size 3, and the model is trained to output a coordinate that follows the advice.

$$L = \frac{1}{n}\sum_{i=1}^{n}(y_i - p_i)^2 \tag{2.2}$$

This pre-training advice grounding stage is crucial to our overall performance, as it allows the end-to-end models to always understand the advice provided, which enables them to take advantage of the advice to solve the original task better. Without this stage, the end-to-end model would be confused about what the advice meant, which would further lead to more confusion when solving the original task. Thus, we recommend that future systems incorporating advice add a pre-training advice grounding stage, and use simple enough advice that these models can understand the advice well.



(a) (b)

Fig. 2.3.: (a) Pre-Trained Advice Understanding Model. (b) [23] End-to-End architecture with our pre-trained model. World represents the board state, while offset and reference represent fully connected layers used to identify the offset and reference blocks.

2.5.3 End-to-End Training

The pre-trained model from Section 2.5.2 that understands various advice text is incorporated into the best performing End-to-End RNN architecture proposed in [23] (and described in Sec 2.5.1) by adding a FC layer to the pre-trained LSTM state h_n (size 256) and summing it with the LSTM hidden state of the original model (as shown in Figure 2.3b). We load and freeze the best performing parameters from our

pre-trained model into the relevant portion of this end-to-end architecture, and train it on the original task of predicting the coordinates of the source / target location, with the addition of advice input. Due to our pre-training step, this end-to-end model can now well understand advice, as well as solve the original task. The end-to-end model is trained and tested on the dataset splits from [23]. The hidden dimension size of the new FC layer is the same as the dimension of the LSTM.

2.5.4 Advice Generation

As explained in the introduction (Sec 1, Sec 2.1), advice can either be provided by a human or self-generated. For the blocks world task, we can self-generate restrictive advice. In this approach, a model is trained to automatically generate restrictive advice for a given input scenario (grid and instruction), based on the assumption that it is easier to predict a region containing the target coordinates rather than their exact location. This method allows us to improve the overall coordinate prediction quality, with no human assistance.

We use a neural network model to self-generate restrictive advice (as shown in Figure 2.4), passing the instruction into an embedding layer of size 256 followed by a LSTM of size 256, the board state into a FC layer of size 20, concatenating these into a FC layer (size 4), and finally using a softmax (defined by Equation 2.3, where z is the input into the function, and has K classes) to classify the input example into a region. We train this architecture using the Adam optimizer [27] and a learning rate of 0.0001 and then run it on the test set, generate the appropriate advice based on the region the data is classified in, and use that as test advice input for the end-to-end architecture from section 2.5.3.

$$\sigma(z)_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \tag{2.3}$$

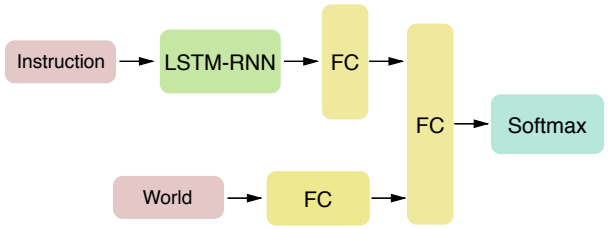


Fig. 2.4.: Model for Generating Advice.

2.6 Experiments

Next, we present our experiments over our four different advice protocols, each with decreasing human effort and overall performance. In each protocol, we provide advice to the end-to-end model from Section 2.5.3, whether it is given by a human user or model self-generated. Our results, evaluated on each model's mean and median prediction error, are presented in Table 2.2. We always compare to the baseline [23] model (described in Sec 2.5.1), which our model is identical to besides the addition of advice (and we always beat), and the state-of-the-art best non-ensemble [24] architecture. Note that [24] use an advanced neural architecture and a different training procedure (source prediction trained as classification). We hypothesize that using the advice mechanism over this more complex architecture would lead to further improvements, and leave it for future work.

The pre-trained advice grounding models from Section 2.5.2 achieve 99.99% accuracy, and are vital, as shown by the poor performance without them (`M4` vs `M5`). These grounding models allow the end-to-end architecture to generalize to the variability in advice utterances and understand the advice fully. Our code to replicate these experimental results is available at [1].

[1]https://github.com/hockeybro12/Improving-NaturalLanguageInteractionWithRobots-Advice

Table 2.2.: Results for our models compared to previous models evaluated as distance from gold prediction normalized by block length for source and target coordinate prediction.

Model	**Source**		**Target**	
	Median	**Mean**	**Median**	**Mean**
`M1` : [23]	3.29	3.47	3.60	3.70
`M2` : Our Replication of [23]	3.13	3.42	3.29	3.50
`M3` : [24]	–	2.21	2.78	3.07
`M4` : Restrictive Advice w/o Pre-Trained Model	3.88	3.83	3.56	3.43
`M5` : 4 Regions Restrictive Advice	2.23	2.21	2.18	2.19
`M6` : Corrective Advice	2.76	2.94	2.72	3.06
`M7` : 4 Regions Retry Advice	2.41	3.02	2.42	3.14
`M8` : 2 Regions Model Self-Generated Advice	3.01	3.31	3.08	3.36
`M9` : Input-Specific Model Self-Generated Advice	2.87	3.12	2.99	3.26

2.6.1 Restrictive Advice

When training the end-to-end model from Section 2.5.3 (to incorporate restrictive advice), we provide restrictive advice at training time for only half the examples. For every epoch, a different half set of examples (determined randomly) receive advice. This mechanism gives the model a chance to learn to interpret each example with and without advice, so that it can handle the interactivity without overfitting to one setup. This setup also gave the best performance.

At test time, the advice is provided only whenever the predictions fall in the wrong general region, just like a human would (via text as described in Section 2.4). As seen in Table 2.2, this model (`M5`) significantly outperforms both baselines (`M1`, `M3`). We note that the performance did not improve much when advice was always provided, showing that this model was able to perform well in its absence and does not rely on it (due to our choice not to provide advice all the time in training). In fact a human would only have to provide restrictive advice for 395/720 examples, and the model always follows it. We note that the performance does not improve from [23] if advice is only provided at train time.

2.6.2 Corrective Advice

We train corrective advice identically to restrictive advice from Section 2.6.1, except we train in two separate iterations. This is necessary as the model must learn to adjust its predictions based on the advice, which is why it is first trained to make the normal prediction (first iteration), then trained to adjust the prediction (second iteration).

In the first iteration, we train identically to [23] with no advice, but in the second iteration corrective advice is generated based on which direction the predictions must be adjusted to be more accurate. This case is simpler than restrictive advice, since the human operator just has to provide the direction to adjust the predictions, rather than the precise region of the coordinates. However, the performance does worsen (`M5` vs `M6`).

2.6.3 Retry Advice

In Section 2.5.4, we introduced a model that was able to self-generate restrictive advice by predicting the general region of the block coordinates given the NL instruction and blocks world. Table 2.3 shows this model's accuracy on that task when the board is split into 4 regions. As this is a hard problem with low accuracy (`A1`), we instead generate advice for the top 2 most confident predictions (determined by the softmax scores) (`A2`).

We now introduce a new multi-step *retry* advice protocol. In the first step, the model from Section 2.5.4 self-generates restrictive advice (using the original grid and instruction from the dataset) based on the most confident predicted region, which it uses as input in the end-to-end model. If the user believes the coordinate prediction based on this advice is wrong, it can tell the model to "retry", and then the second most likely restrictive advice will be used. Thus, the only human feedback needed now is telling the model to "retry", rather than accurate advice as before. The performance

of this (M7) still significantly outperforms [23] and is close to the state-of-the-art [24] on target prediction.

In this approach, we still use accurate advice at training time, just the self-generated advice at test time. We generate two sets of self-generated advice, one for the most confident region prediction, and another for the next most confident one (determined by the softmax scores). If the general region of the coordinate prediction in the first iteration of running the end-to-end model (with the most confident self-generated advice) is incorrect, the human operator will provide retry advice, and we then feed in the second most confident advice (using that for the final prediction).

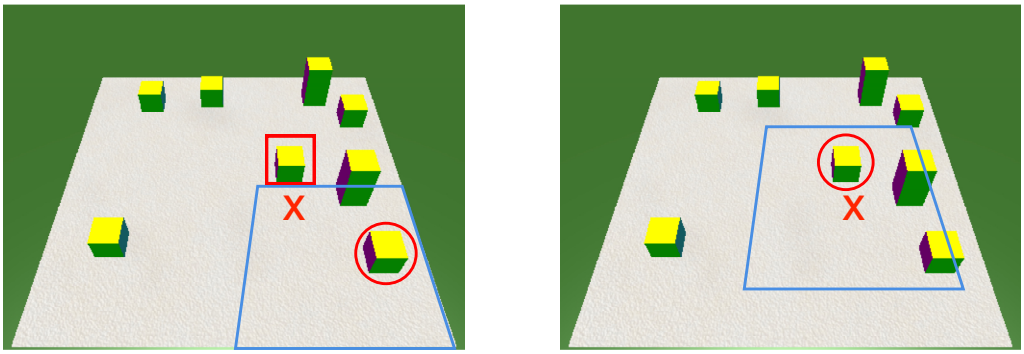
Table 2.3.: Accuracy of model self-generated advice.

Regions	**Source**	**Target**
A1 : 4	47%	40%
A2 : 4, Top 2 Confidence	73%	70%
A3 : 4, Input-Specific	67%	62%

2.6.4 Model Self-Advice Generation

We now aim to avoid any human interaction, by letting the model completely self-generate the advice. Accomplishing it would allow us to improve the model's performance without additional human effort. We experimented with two approaches. In the first, we generate advice as described in Section 2.6.3. However, instead of having the user ask the model to "retry", we treat the top 2 confidence regions as a general region, and provide that as advice input as described in Section 2.6.1. In this case, there is a performance improvement over [23] with no human effort required (M8 in Table 2.2).

Our second approach for self-generated advice aims to improve on some of the shortcomings of the first approach. Previously, when generating the advice, we had decided on four coarse-grained regions, and trained a model to classify each input example into one of these regions. In many cases, the true coordinate lay close to



(a) (b)

Fig. 2.5.: (a) The [23] model would have made a prediction ('x') close to the true block (square). However, the advice region (blue) was incorrect (due to the true block being close to the edge of it) and this led to a significantly worse prediction (circle). (b) In the input-specific self-generated advice model, the advice region (blue) is centered at the incorrect coordinate prediction ('x'), leading to the true source block being included and a correct prediction (circle).

the boundary of one of these regions, often resulting in the model predicting the wrong region when self-generating the advice. This incorrect prediction would lead to significantly worse performance (when compared to the model without advice) when running the end-to-end model from Section 2.6.1, as the advice was incorrect (remember that the model always follows our advice, and the true coordinate is not in the advice region due to the mistake). However, if we had instead chosen our regions to be centered at the true coordinate of each input example, it would be less likely that the model would make an incorrect region prediction (since a small error would still lead to the region containing the correct coordinate). Figure 2.5 provides a visual explanation of this.

For this reason, we now introduce input-specific model self-generated advice. In this case, we run the original coordinate prediction model [23] in two iterations. In the first iteration, we use the prediction to generate advice for a region (of the same size as in the case of 4 quadrants) centered at the predicted coordinate (see Figure 2.5b).[2] In the second iteration, we feed in this generated advice just like Section 2.6.1. This

[2]We make sure the advice region doesn't exceed the board boundaries.

model (M9) achieves performance slightly worse than retry advice, and significantly better than [23], all with no human effort.[3] Table 2.3 shows the accuracy increase in predicting the advice now (A3 vs A1). It is unsurprising that this approach to self-generating advice performs better, as now the regions are more specific to each coordinate (so there is a higher probability that the true coordinate is actually in the predicted region - see Figure 2.5).

We hypothesize that the performance improvements in self-generated advice happen since it is easier to predict the general region used to generate the advice rather than the specific coordinates. Previously, we have also shown the benefit of restrictive advice in improving overall coordinate prediction, so it is unsurprising that a high accuracy of advice generation leads to better overall performance. Due to this, we propose that future robot communication works take advantage of predicting and then using model self-generated advice in their end-to-end training procedure.

2.7 Conclusion

2.7.1 Future Work

The advice we designed and presented so far is general knowledge that a human is likely to have about the blocks world problem, and can provide to the agent (either always, or when it sees the agent make a mistake). Specifically, the human can restrict the search space of the agent, tell it which direction to adjust its' predictions, or just tell it to retry.

In the future, we plan on further exploring the first two advice protocols to make them even more interactive. For restrictive advice, we can adapt our pre-trained advice understanding models to understand advice regions of various sizes. The humans can then iteratively provide multiple kinds of restrictive advice, each getting more specific. For example the first advice would restrict the search space to a region

[3]Note that we must re-train the model from Section 2.5.2 as there are now significantly more regions. The accuracy of that model is still 99.99%, and the training procedure does not change.

1/2 of the grid, then 1/4th, then 1/8th, and so on, until the agent makes the correct prediction. In this case, the human is providing better and better advice (the regions get smaller) as the iterations go on, allowing the agent to learn how to search in a variety of regions and successively narrow down the search space. This could lead to better performance as well as make the process more interactive. A similar approach could also be used in the self-generated case, where the agent first makes a prediction of a broad general region, then makes a overall coordinate prediction, uses that to make a prediction of a smaller general region, then another coordinate prediction, and so on.

This same multi-step approach could also be used for corrective advice, as in each step the human would tell the agent which direction to adjust it's predictions to get closer. The difference here compared to what we did earlier for corrective advice is that we would provide advice until the agent made the correct prediction, not just once.

Also, we could experiment with combining restrictive, corrective, and self-generated advice and providing them all at once (or in succession of each other) to see if that improves performance.

Finally, it would be interesting to see how providing advice affects performance in the cases where advice is not provided at all. It is possible the performance can improve, since the agent learns more about the blocks-world task through advice (for example it learns how to navigate smaller search spaces). In some preliminary experiments on this, we saw some small improvements, but they were not significant enough to include. However, adding more interactive advice like mentioned above could provide significant improvements in cases where no advice is given.

2.7.2 Summary

In this chapter, we have introduced four different interactive advice-based protocols to improve performance on the traditional blocks-world task. Our protocols

were proposed in order of decreasing human effort and decreasing performance, but all were better than our baseline [23], whom our model was identical to besides the inclusion of advice. The last method, model self-generated advice, shows the benefit of considering advice even when not designing an interactive protocol. Thus, we have shown the various ways in which advice-driven learning (learning by receiving advice) improves performance on the blocks-world task. In the next chapter, we will take a look at another task, fake-news detection, and see how advice can be helpful there.

3. FAKE NEWS DETECTION

3.1 Introduction

Fake news has emerged in recent years as a major problem in society. With the rise of social media and smartphones, everyone is constantly connected to each other. Furthermore, everyone has the ability to share and post any information (whether it is factual or not) online, for everyone else to potentially see. This has the benefit of real news quickly propagating to all social media users, and all of them being aware of what is happening in the world, but also has the downside of fake news being spread and users believing it. This trend is contradictory to what happened in the past, where people saw news through sources more challenging to produce such as television, radio, newspapers, or specific trusted online websites (such as CNN, Yahoo News, etc.). However, now, since many people use social media daily, anyone can put out whatever they want, and if it gets enough traction, people will see it and may actually believe it.

There are numerous reasons why an organization may want to post fake news. [8] discuss some of them, which we briefly summarize here. Two main benefits from spreading fake news that they highlight are economic and ideology-spreading ones. Organizations can benefit economically if they are able to successfully spread fake news that brings increased traffic to their website, which allows them to generate revenue through advertisements. This increased traffic is more likely to happen with fake news, as the news could be surprising/controversial and more people would want to read more about it. Organizations can also benefit from spreading their own beliefs and changing people's thoughts, such as in the case of the 2016 presidential election, where fake news could have helped Donald Trump win [28].

Due to the wide-spread nature of fake news, there have been many efforts to try and stop it. This includes fact-checking organizations like Politifcat, Snopes, and FactCheck, which manually verify claims. There have also been numerous datasets [8, 29] released to enable AI systems to better tackle the fake news problem. While progress has been made [30], fake news detection is still a challenging problem for AI systems. For this reason, we propose to use advice to empower AI systems to better detect fake news.

In this chapter, we will build a graph-based model as a baseline for fake-news detection. We will then propose various types of advice that can be applied to this model, and fake-news detection in general.

3.2 Related Work

Fake news detection has been a hot topic of research recently, partly evidenced by the ACM having a special issue on it [31]. Various top conferences have also organized competitions to detect fake news, such as the FEVER [32] task. In this dataset, over 185 thousand claims extracted by altering sentences from Wikipedia are labeled based on evidence.

There have also been a few surveys [33] and overview articles [34] by various sources aimed to inform readers more about fake news and the science behind it. There are many approaches (statement level, article level, user level) to detecting whether news is fake or not, but in this work we focus on detecting whether a source is fake or not, based on the dataset by [8], which we describe in Section 3.4.

3.3 Background Knowledge

Before we begin to tackle the problem for fake news, it is important to go over some background knowledge needed for understanding our models.

3.3.1 BERT

BERT (Bidirectional Encoder Representations from Transformers) [35] is a language model that has achieved state of the art (SOTA) results in many natural language processing tasks. Language Models in NLP are trained to estimate the relative likelihood of different phrases, and after they are trained, they can be finetuned to achieve SOTA performance on a specific task/group of tasks.

BERT takes advantage of the Transformer model as its architecture [36], which is an attention based model that learns relations between words in text. It uses the encoder part of the transformer, more details of which are described in [36].

The key benefit of BERT is that it is designed to pre-train deep bi-directional representations from large amounts of unlabeled text. It can do this as it introduces a novel objective that allows the model to understand the text bidirectionally. The objective is a Masked Language Model objective, in which 15% of the words in the sentences that are fed into the model are replaced with a `[MASK]`token. The model aims to predict the value of the Masked tokens, based on the context given by the rest of the sentence. This objective allows the model to process the each sentence bidirectionally rather than just left-to-right, as there is no way to cheat, since tokens are masked. In a traditional language modeling objective, such as predicting the current word given the previous words, models that process the text left-to-right cannot process the same text right-to-left, as they would have already seen the current words when reading in the reverse direction. Thus, they would be cheating.

BERT also introduces the Next Sentence Prediction objective, in which the model receives a pair of sentences and learns to predict whether or not the second sentence follows from the first one. This binary prediction objective allows the model to better capture the relationship between sentences in language.

Due to these two novel training objectives and being trained on massive amounts of data, BERT can be finetuned to achieve high performance on many tasks [35], and

is being widely used in NLP [37–39]. We will use BERT as a key component of our graph-based model.

3.4 Dataset

3.4.1 Overview

In the rest of this chapter, we will focus on the task of fake news source detection, originally proposed by [8]. It is important to focus on detecting if a source is fake or not, as a website that has published fake news in the past is likely to do so in the future. Thus, if there is a system that identifies whether a given source traditionally posts fake news, it could help humans decide whether or not to trust another story from that same source. While there are organizations that focus on publishing lists of fake news sources, these lists are tedious to compute and are easily outdated, as news sources are constantly popping up. Hence, an automated system to do this is a great necessity.

The dataset released by [8] includes 1,066 websites for which both bias and factuality labels were provided or could be easily inferred from the Media Bias/Fact Check (MBFC) website [1]. They model factuality on a 3-point scale (*Low*, *Mixed*, and *High*). In this work, we do not focus on predicting bias. The distribution of the dataset is presented in Table 3.1. The model they proposed is a SVM based on features they crawled from the websites (articles and their information, wikipedia/twitter page, URL, web traffic, etc.).

Table 3.1.: Distribution of factuality labels in the dataset [8].

Factuality Type	**Count**
Low	256
Mixed	268
High	542

[1] https://mediabiasfactcheck.com

3.4.2 Article Scraping

As the dataset [8] does not come with articles already, but rather just the news sources, we must scrape the articles from the sources. In order to scrape articles, we took advantage of publicly available Python packages Scrapy [2] and Newspaper [3]. These packages return articles given a website URL. In order to make sure we had enough data, we only considered sources where we could scrape at least 100 articles from these packages. We also make sure that our dataset is balanced, with 1/3 of the labels belonging to each category. Due to this, for our preliminary experiments, we were left with 66 sources. We hereby refer to all of these sources as "unknown sources", as we are unsure about their factuality.

In our method explained later in Section 3.5, we utilize articles from sources that we trust and believe to be mostly factual. These include: cnn.com, nbcnews.com, foxnews.com, news.yahoo.com, abcnews.go.com, bloomberg.com, and espn.com. We call these "known sources", since we know they are factual. For each article from an unknown source in the dataset, we extract up to 25 similar articles from each known source.

In order to find the similar articles, we formulate a query string that we search into Google News Search[4]. For each unknown source article, we extract all the entities from the article title using the Stanford NER Tagger [40], and concatenate them to the query string. We also extract the date the article was published and add that to the query string. Finally, we append the URL of the known news source we want to find articles from to the query string. These three steps enable the Google news search to find articles with a similar topic, published at around the same date, and published by the known news source. Thus, intuitively, we are hoping to find articles from Google News that talk about the same events as the ones from the "unknown" sources, and we are approximating this by entities and dates. We then search this

[2]https://scrapy.org
[3]https://github.com/codelucas/newspaper
[4]https://news.google.com

query string into Google news, and get back the top 25 article results. For each result, if it has the known source URL in its' URL, we consider it to be a similar article for the unknown source article from the known source. While this method is noisy, we hypothesize and find by visual inspection that some of the 25 articles returned are related or on the same topic date as the unknown source article. This is likely due to the power and accuracy of Google News search. This can be done easily through Google News queries from the Official Google API. Once the resulting articles are found, we can scrape and download them like before using Scrapy and other such API's.

We repeat this process for all the extracted unknown source articles. Our dataset now has labeled unknown sources with articles and articles from known sources that are similar (or identical) in topic to the unknown ones.

3.5 Models

Before we apply advice to the Fake News Detection Model, we must construct a baseline model, which is what we do in this section. Our model is a graph consisting of all the data in our dataset: unknown sources, unknown source articles, known sources, known source similar articles. We aim to capture trustworthiness (defined as how much we can trust a given source in terms of its factuality) of each unknown source in our graph, which we do by propagating trust (either positively or negatively) from known source articles based on how similar they are to their corresponding unknown source articles. If an unknown source article is very similar to a known source article, we trust it and consider it not to be fake. We define sources that have higher trustworthiness scores (more trustworthiness) to be more factual, and lower scoring ones to be more likely to spread fake news.

3.5.1 Graph

First we define how we structure our dataset into a graph. This can also be seen visually in Figure 3.1

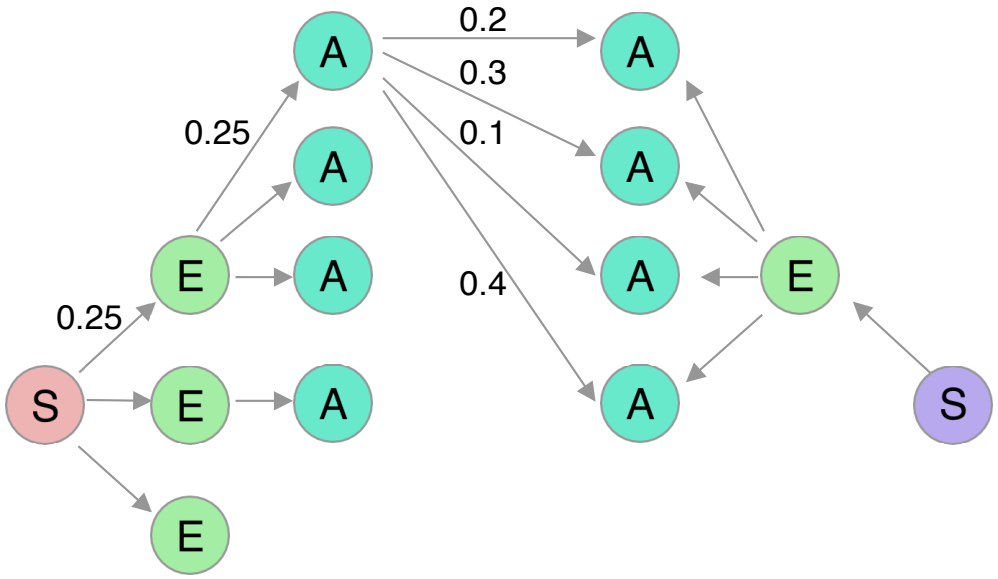


Fig. 3.1.: In our graph model, we have unknown sources (pink), known sources (purple), entities (green), and articles (teal). We connect each unknown source to its entities and each known source to its entities. Each entity is also linked to the articles about that entity. Each article in the unknown source is linked to its' similar articles in the known source (found by Google News search). For space, in this figure we show the similar articles for only one known unknown source article. The edge weight for the pairs of articles is determined by the BERT [35] similarity score. The edge weight for each article is the average of the edge weight for all the it's similar articles scores (0.25 in the graph). The edge weight for the entities is the average of the edge weight for all the articles, and the edge weight for the source is the average of all entities. This graph model allows trust to propagate from the known sources to the unknown sources.

In the graph, each source (either from the unknown or known sources list) is included as a root vertex (represented as a single node, no embedding). From each source root vertex, there is an edge to all of the entities (also represented as a node) mentioned by that source's articles (represented as a BERT embedding). Each edge also has a weight, initialized to 1.0, representing that entities "trustworthiness" score.

As mentioned earlier, we define trustworthiness as a score defining how much we can trust a source - sources with higher trustworthiness have higher factuality. For example, if an unknown source fury.news mentions articles about Donald Trump and Hillary Clinton, the root vertex fury.news will have edges with weight 1.0 to each of these entities. We determine the entities mentioned by a source by going through all of its' articles' titles and extracting the unique entities using the Stanford Named Entity Recognition Tagger [40]. If we didn't trust fury.news at all, it would have a trustworthiness score of 0, but if we trusted it a lot, it would have a score of 1.0.

Each entity is further connected to the article it was extracted from, also initialized with edge weight 1.0.

Furthermore, each article from the unknown sources is connected to each of the similar articles that we extracted from Google News in Sec 3.4.2. We now want to weight these edges by how similar each unknown source article is to the known source article it is linked to. We will use this similarity information later to propagate trust from known sources to unknown sources. The edge weight is determined by the BERT cosine similarity score between the two articles (one from the unknown source and one similar article). This method allows us to capture how similar the articles are. To compute this score, we use the BERT sentence embeddings proposed by [41], which are more semantically meaningful than traditional BERT embeddings. We compute the embedding for each article, then compute the cosine similarity between the two embeddings (the default BERT-base embeddings), and finally use that as the edge weight. We use BERT as our model for this task because as discussed earlier in Section 3.3.1, BERT builds deep bidirectional representations for text, which can be fine-tuned to achieve SOTA performance in many NLP tasks. Thus, we feel BERT can well capture the content of the articles, and the similarity score between them will tell us how "similar" each unknown source article is to each of it's paired known source articles. We also discuss how we fine-tune BERT for fake news detection in Section 3.5.3.

3.5.2 Trust Propagation

Our graph now has unknown sources with their articles and entities along with known sources with their articles and entities. However, it does not capture trustworthiness scores for the unknown sources, so we can't determine their factuality yet. In order to capture this, we must propagate trust from known sources to unknown sources.

We propagate trust back from known sources to unknown source articles to unknown source entities and finally to unknown sources. Once this is complete, each unknown source will have a trustworthiness score. We trust the articles from our pre-defined known sources, which are not included in the unknown sources list. We compute the trustworthiness score for each unknown source article by looking at the top k trusted articles that it is most similar to (based on the assumption that an article that is similar to many trusted articles is likely to be trustworthy, so the article should get a high trustworthiness score). This similarity is captured already in the graph by the BERT embedding cosine similarity scores. We choose k as a hyperparameter (set to 4 in our most successful experiments) and do not consider all similar articles, as it is likely that not all of the 25 articles extracted from Google News are talking about the same topics as the unknown source article, and we want to minimize the penalty articles receive due to us/Google News not being able to find enough similar articles. We now update the edge weight between the unknown source entity and this article to be the average of the similarity score between the article and it's top k similar known source articles. Thus, we have now propagated trust from the known articles to the unknown articles and back to the entities.

Finally, we update each unkown source with the average score of all of it's entities, weighted by how many articles each entity had (so we do not put equal weight to entities with only one article as ones with a hundred articles). Now, each source has a trustworthiness score.

In this subsection, we have discussed how we propagated trust from known sources to unknown sources, by utilizing BERT based article similarity scores. In the end, sources that have more articles similar in content to known factual source articles will have higher similarity scores, and thus more trust (represented by a high trustworthiness score).

3.5.3 BERT Fine-tuning

In Section 3.3.1, we discussed how BERT can achieve SOTA performance in NLP tasks when it is fine-tuned. In this subsection, we discuss how we fine-tune our sentence BERT model from [41] so that it performs better on fake news detection.

We fine-tune BERT-base on the task of predicting whether a news article is fake or not. In order to do this on large amounts of data, we take advantage of our existing graph with trustworthiness scores for unknown and known sources. We sort the trustworthiness scores from the graph in decreasing order. We then assume for the sake of fine-tuning that the top 20% (a hyperparameter) of unknown sources in our graph are to be trusted (our model says that their articles are most similar to the known sources, so it's likely that they are also factual), and the bottom 20% of unknown sources are not trusted (or fake). In this way, we are using the initial BERT representation which allows us to capture trust in the graph to further enhance the BERT representation, so we can even better capture trust in the graph. We build a balanced dataset from these sources articles. The dataset has 1000 articles for high and low factuality sources each. We feed in these articles into BERT, and using an extra Fully Connected classification layer (mapping the 768 BERT dimension to 2 for classification, followed by softmax), ask BERT to predict whether an article is fake or not. All articles from the top 20% of unknown sources are given the 0 label (for not fake), and all articles from the bottom 20% of unknown sources are given the 1 label (for fake). The model is trained with cross entropy loss, Adam optimizer, and a

learning rate of 0.001. We make sure to balance the dataset (so we don't use all the articles from the sources) This fine-tuned model can achieve 91% accuracy.

3.5.4 Graph Updates

After the BERT fine-tuning step is done, we can update our graph. We can recompute BERT similarity scores between pairs of articles, and then do the trust propagation step from Section 3.5.2 again. After that, we can update BERT again, and so on, until convergence.

3.5.5 Prediction

We mentioned in Section 3.4.2 that our dataset is balanced, with 1/3 of the sources being each label (low, mixed, and high factuality). Thus, in order to compute the final labels for all of our unknown sources, we just need to sort them in decreasing order by their trustworthiness score, and then take the top third of the sources to be high, middle third to be mixed, and bottom third to be low.

In a scenario where we got a new source that was not part of the dataset and had to classify it, our approach would still work, with some slight modification. We could classify the new source with the same label as the sources it is closest to in the trustworthiness score ranking. If both the source above and below it have the same label, classification is easy and would be the same as that source. However, if both have a different label, then classification would be based on whichever source the trust score score is closest to. In the case where the trust score score is equally close to both neighboring sources, we would pick the label randomly with each neighboring source having equal probability.

3.5.6 Trust Propagation after Prediction

After making the prediction, it is possible to further propagate trust. We can consider the top 5% of sources predicted as trustworthy to actually be trusted, and add them to our known sources list. We then link the articles from these new known sources to articles from unknown sources just like we did before for the other known sources (using entities extracted from their titles). Then, we can compute the BERT embedding similarity score for these new edges, and propagate trust just like Section 3.5.2.

3.6 Advice

In the last section, we described our baseline graph model to capture trustworthiness scores for unknown sources. In this section, we discuss how we can take advantage of advice to improve the performance of that model. As in the blocks-world case, we consider both human provided and self-generated advice. We aim for our advice to be provided interactively by the human when they choose, and it to be simple enough that our model can understand and take advantage of it.

3.6.1 Types of Advice

The advice we design for fake news is based on the graph and can be provided by a human with knowledge of the graph or certain articles. In order for a human to provide advice, it first needs access to the graph.

After building and updating the graph discussed in the previous section, the system makes predictions based on the trustworthiness score ranking of unknown sources. In order to make it easier for the human to provide advice, the system will show the human only the part of the graph belonging to the sources that it is unsure about (they were close to being predicted a different label).

The human then offers advice to the system based on the part of the graph it is shown. For the purposes of this work, we consider the following types of advice:

1. Attention advice. In this case, the human tells the model which section of the graph to give either more or less importance to, based on the current trustworthiness of articles/entities/sources in that section. For example, if a human notices that the similar articles from the known sources in one section of the graph are actually related to the unknown source article they are connected to, it can tell the system "You did a good job finding the similar articles for article _". In this case, the system will take advantage of the advice by giving more weight to that unknown article when computing/propagating the trustworthiness score for that source.

2. Factual advice. For this type of advice, a human with some world knowledge about the entities/articles being discussed provides a factual or non factual statement to the system. For example, given an article talking about President Bush watching basketball in his office after 9/11, something that is false, a human could provide advice such as: "The President of the United States is likely to take action after a big event like 9/11, not watch basketball". As the system has now received advice about this article, it can take advantage by decreasing the trustworthiness score of that article and giving this article more weight when propagating trust to the rest of the graph.

Both of these types of advice are provided by humans by looking at the graph and noticing areas where the system either did a good job or made a mistake, and letting it know. This is in contrast to active learning, where the system would ask for help with a given article. In this case, the human is choosing which part of the graph to give advice, something that is easier for them, as they can give advice based on knowledge they already have.

After advice is provided, the system can not only use it to gain a better understanding of the article and it's source, but also use it to gain better understanding

of other sources. If advice makes a source more or less trusted, this can impact the propagation of trust to other sources as described in Section 3.5.6. In this way, the advice given to any source can also be indirectly used for other sources.

3.6.2 Self-Generated Advice

In this section, we discuss how we can self-generate factual advice. To do this, we take advantage of the Politifact[5] fact checking source. Politifact is a non-partisan fact-checking website that has been around since May 2007. The website lists a bunch of political statements and labels them on a scale of 1-5, with 1 being not true (pants on fire!), and 5 being trustworthy (True). We scrape all the statements (around 7000) and their labels from Politifact.

We also take advantage of a Textual Entailment model, using the one proposed by [42] with the same hyper parameters as them. This model was also used by [43], to fact-check. The Textual Entailment model can be used to determine whether a piece of text entails a fact or not. This approach is similar to a typical fact-checking pipeline [32]. However, the way we get the facts through self-generated advice and how we build/update the graph is unique to our work.

We first extract dates and entities mentioned in all the facts from Politifact using the Stanford NER tagger as before. We then go through each article in the part of the graph given to us by the system, and find the related facts from Politifact based on the articles' published date and entities in the article title. We hypothesize that statements about the same entities around the same date could be related to our article. We used the same hypothesis in constructing our graph finding the known source articles for the unknown sources.

Next, we aim to determine whether the article entails one of the statements given on Politifact, because if it does, we can use that to either increase or decrease its trustworthiness score (based on the trustworthiness of that statement). In order to

[5]https://www.politifact.com

do this, we summarize the article using [44], and then pass each sentence along with each candidate Politifact statement into our textual entailment model. The model returns the probability that the fact is entailed by the sentence of the article, which we threshold at 0.7. If an article entails a statement with probability greater than 0.7, we increase/decrease that articles trustworthiness score proportionally based on the Politifact label. For example, if an article entails a statement with very high factuality, then it is likely that the article is also factual and thus its trustworthiness score would be increased significantly.

In this way, we are noisily filtering Politifact statements to use as factual advice for our system and using those to update the trust for the articles.

3.7 Experiments

Our experiments are based on our graph model proposed in Section 3.5.1. Our experiments so far are preliminary results.

3.7.1 Baseline Graph

We use the equally balanced 66 sources we found at least 100 articles for from the the dataset proposed by [8] for fake news source detection, as mentioned in Sec 3.4.2. We evaluate using the label choosing scheme described in Section 3.5.5. We achieve an accuracy of 62% with our baseline graph model, that improves to 64% when BERT is finetuned. We can see that the performance improves when BERT is finetuned, showing that this step helps the model better decide whether two articles are related, by having it focus on the key parts that make an article fake.

3.8 Conclusion

In this chapter we have built a graph-based model for fake news detection that also takes advantage of a state of the art language model BERT, and also applied advice

to it. We proposed two types of advice that a human could provide to the graph based on their knowledge of the world. The advice is simple for a human to provide via natural language, and powerful due to our architecture of trust propagation, allowing the advice for one source to potentially impact other sources. We also showed how one of the types of advice, factual advice, can be self-generated.

Our next steps for this work is to conduct a study where we have humans actually provide the advice and see how it improves performance, rather than only self-generating it. We hypothesize that this will work better than self-generated advice, as each advice provided will be more relevant to each article.

4. SUMMARY

In this work, we have built on work previously done on interactive systems, and explored different approaches for relaxing the typical single-step nature of many NLP tasks by introducing advice-driven learning. We have shown how we can view NLP tasks as more of an *interactive*, process, by taking advantage of advice, short natural language sentences that a human can provide to an AI system to help it improve its' performance. Studying blocks-world and fake-news detection tasks, we have shown how advice can be designed to be simple for both the human to provide and the system to understand, while still being incredibly useful. In these ways, advice has been shown to be an easy way for an AI system to incorporate human feedback and knowledge into its' learning process interactively.

The advice we discussed in this work was designed specifically to be easier to understand than the original task. Specifically in the blocks-world, we showed how our pre-trained models for understanding the advice achieved near 100% accuracy, and the end-to-end models always followed the advice. These are themes we feel are critical for any future advice-driven systems, as if a system cannot understand the advice, it will be more challenging for it to incorporate it. As language understanding models get better and better, designing types of advice can get more and more complex, since the models will be better equipped to understand it.

We also discussed self-generated advice, which is a good way to achieve performance even when there isn't a human to provide the advice. It is also another benefit to building a system with advice in mind, as if the human can provide it, the model can have significant performance increases, but if not, it can be self-generated and still be helpful.

The types of advice we discussed in this work can also be complementary to existing forms of learning, as we discussed in the Introduction Chapter. As advice is

not a final label, but rather a partial solution/hint, advice can be provided in addition to supervised learning, unsupervised learning, reinforcement learning, and/or active-learning. This makes advice useful as a framework that can be combined with existing methods. For these reasons, we believe advice-driven learning is a useful framework that other AI practitioners should utilize.

4.1 Future Work

In this section, we discuss some medium-term and long-term extensions to advice that can be explored in the future.

4.1.1 Medium-Term

In the medium-term, our goals are to apply advice to different NLP tasks that are currently being considered single-step processes. Currently, there is a lot of work in Machine Learning that is heavily dependent on supervision, but when designing advice-based protocols, one thing to look for is the ability to decompose the final task. If it is possible, advice can be used to provide supervision from humans interactively in a focused way. Through self-generated advice, we can also aim to alleviate some of that effort needed by the humans.

As natural language processing systems like BERT [35] become more advanced, the amount of decomposition needed will reduce, as the models will be able to better understand different types of advice, and then advice can become more complex. This will in turn allow us to take advantage of advice to solve even more challenging tasks.

4.1.2 Long-Term

In the long term, we envision a future in which humans can seamlessly provide advice to AI systems that they see on a day-to-day basis, and the systems can learn and constantly become better from it. Rather than in the case of active learning where

a system must explicitly ask questions in order to be taught something, humans can observe an AI system making a mistake / having trouble and tell it. In this way, AI systems will always be improving due to human knowledge.

As a simple example of how this can actually be done, we look at robots. At Purdue University and other universities, Starship robots have become fairly popular. They are tiny robots that roam the university sidewalks, delivering food and other small items from point A to point B. For example, a human can request delivery of a food item on campus, the robot autonomously goes over to the restaurant where someone places the item in a compartment in the robot, then the robot navigates to the destination, and delivers the item. One of the challenges with these robots is the ability to cross busy streets, as sometimes their camera sensors cannot tell whether a car is coming or not. This issue currently causes the delivery process to take a lot longer than it should, as the robots can often be seen just waiting at an intersection. In this situation, we envision that advice can be helpful. For example, if a human is walking by, and provides the advice to the system "Come cross with me", the robot can understand that and follow the human across the street. As it is crossing, it can also examine the surroundings and learn more about the situation, so that next time it sees the same situation, it is more likely to cross, even when a human is not there. In this case, the robot didn't ask for help, but the human just provided it through advice, and the robot got better because of it. This is just one example of a real-world scenario where advice can be useful.

REFERENCES

REFERENCES

[1] A. Suhr, M. Lewis, J. Yeh, and Y. Artzi, "A corpus of natural language for visual reasoning," in *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, 2017, pp. 217–223.

[2] A. Talmor, J. Herzig, N. Lourie, and J. Berant, "Commonsenseqa: A question answering challenge targeting commonsense knowledge," *arXiv preprint arXiv:1811.00937*, 2018.

[3] R. T. McCoy, E. Pavlick, and T. Linzen, "Right for the wrong reasons: Diagnosing syntactic heuristics in natural language inference," *arXiv preprint arXiv:1902.01007*, 2019.

[4] J. F. Allen, L. K. Schubert, G. Ferguson, P. Heeman, C. H. Hwang, T. Kato, M. Light, N. Martin, B. Miller, M. Poesio *et al.*, "The trains project: A case study in building a conversational planning agent," *Journal of Experimental & Theoretical Artificial Intelligence*, vol. 7, no. 1, pp. 7–48, 1995.

[5] P. E. Rybski, K. Yoon, J. Stolarz, and M. M. Veloso, "Interactive robot task training through dialog and demonstration," in *Proc. of the ACM/IEEE international conference on Human-robot interaction*. ACM, 2007, pp. 49–56.

[6] T. Wen, D. Vandyke, N. Mrkšíc, M. Gašíc, L. Rojas-Barahona, P. Su, S. Ultes, and S. Young, "A network-based end-to-end trainable task-oriented dialogue system," in *Proc. of the Annual Meeting of the European Chapter of the Association for Computational Linguistics (EACL)*, vol. 1, 2017, pp. 438–449.

[7] T. Winograd, "Understanding natural language," *Cognitive psychology*, vol. 3, no. 1, pp. 1–191, 1972.

[8] R. Baly, G. Karadzhov, D. Alexandrov, J. Glass, and P. Nakov, "Predicting factuality of reporting and bias of news media sources," in *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, 2018, pp. 3528–3539.

[9] S. Srivastava, I. Labutov, and T. Mitchell, "Learning to ask for conversational machine learning," in *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, 2019, pp. 4155–4165.

[10] M. MacMahon, B. Stankiewicz, and B. Kuipers, "Walk the talk: Connecting language, knowledge, and action in route instructions," 2006.

[11] S. R. Branavan, H. Chen, L. S. Zettlemoyer, and R. Barzilay, "Reinforcement learning for mapping instructions to actions," 2009.

[12] D. L. Chen and R. J. Mooney, "Learning to interpret natural language navigation instructions from observations." 2011.

[13] S. Tellex, T. Kollar, S. Dickerson, M. R. Walter, A. G. Banerjee, S. J. Teller, and N. Roy, "Understanding natural language commands for robotic navigation and mobile manipulation." 2011.

[14] C. Matuszek, N. FitzGerald, L. Zettlemoyer, L. Bo, and D. Fox, "A joint model of language and perception for grounded attribute learning," 2012.

[15] J. Kim and R. J. Mooney, "Adapting discriminative reranking to grounded language learning." 2013, pp. 218–227.

[16] D. K. Misra, J. Langford, and Y. Artzi, "Mapping instructions and visual observations to actions with reinforcement learning," 2017.

[17] T. M. Howard, S. Tellex, and N. Roy, "A natural language planner interface for mobile manipulators," in *2014 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2014, pp. 6652–6659.

[18] J. Johnson, B. Hariharan, L. van der Maaten, L. Fei-Fei, C. Lawrence Zitnick, and R. Girshick, "Clevr: A diagnostic dataset for compositional language and elementary visual reasoning," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 2901–2910.

[19] J. Andreas, M. Rohrbach, T. Darrell, and D. Klein, "Neural module networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 39–48.

[20] R. Hu, J. Andreas, M. Rohrbach, T. Darrell, and K. Saenko, "Learning to reason: End-to-end module networks for visual question answering," in *Proceedings of the IEEE International Conference on Computer Vision*, 2017, pp. 804–813.

[21] A. Suhr, S. Zhou, A. Zhang, I. Zhang, H. Bai, and Y. Artzi, "A corpus for reasoning about natural language grounded in photographs," *arXiv preprint arXiv:1811.00491*, 2018.

[22] H. Tan and M. Bansal, "Lxmert: Learning cross-modality encoder representations from transformers," *arXiv preprint arXiv:1908.07490*, 2019.

[23] Y. Bisk, D. Yuret, and D. Marcu, "Natural language communication with robots." 2016.

[24] H. Tan and M. Bansal, "Source-target inference models for spatial instruction understanding," 2018.

[25] S. Hochreiter and J. Schmidhuber, "Long short-term memory," 1997.

[26] R. Pascanu, T. Mikolov, and Y. Bengio, "On the difficulty of training recurrent neural networks," 2013, pp. 1310–1318.

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

[28] H. Allcott and M. Gentzkow, "Social media and fake news in the 2016 election," *Journal of economic perspectives*, vol. 31, no. 2, pp. 211–36, 2017.

[29] W. Y. Wang, ""liar, liar pants on fire": A new benchmark dataset for fake news detection," in *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, 2017, pp. 422–426.

[30] R. Zellers, A. Holtzman, H. Rashkin, Y. Bisk, A. Farhadi, F. Roesner, and Y. Choi, "Defending against neural fake news," *arXiv preprint arXiv:1905.12616*, 2019.

[31] S. Papadopoulos, K. Bontcheva, E. Jaho, M. Lupu, and C. Castillo, "Overview of the special issue on trust and veracity of information in social media," *ACM Transactions on Information Systems (TOIS)*, vol. 34, no. 3, p. 14, 2016.

[32] J. Thorne, A. Vlachos, C. Christodoulopoulos, and A. Mittal, "Fever: a large-scale dataset for fact extraction and verification," in *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, 2018, pp. 809–819.

[33] K. Shu, A. Sliva, S. Wang, J. Tang, and H. Liu, "Fake news detection on social media: A data mining perspective," *ACM SIGKDD Explorations Newsletter*, vol. 19, no. 1, pp. 22–36, 2017.

[34] D. M. Lazer, M. A. Baum, Y. Benkler, A. J. Berinsky, K. M. Greenhill, F. Menczer, M. J. Metzger, B. Nyhan, G. Pennycook, D. Rothschild *et al.*, "The science of fake news," *Science*, vol. 359, no. 6380, pp. 1094–1096, 2018.

[35] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," in *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, 2019, pp. 4171–4186.

[36] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in neural information processing systems*, 2017, pp. 5998–6008.

[37] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever, "Language models are unsupervised multitask learners," *OpenAI Blog*, vol. 1, no. 8, 2019.

[38] S. Reddy, D. Chen, and C. D. Manning, "Coqa: A conversational question answering challenge," *Transactions of the Association for Computational Linguistics*, vol. 7, pp. 249–266, 2019.

[39] J. Gao, M. Galley, L. Li *et al.*, "Neural approaches to conversational ai," *Foundations and Trends® in Information Retrieval*, vol. 13, no. 2-3, pp. 127–298, 2019.

[40] J. R. Finkel, T. Grenager, and C. Manning, "Incorporating non-local information into information extraction systems by gibbs sampling," in *Proceedings of the 43rd annual meeting on association for computational linguistics*. Association for Computational Linguistics, 2005, pp. 363–370.

[41] N. Reimers and I. Gurevych, "Sentence-bert: Sentence embeddings using siamese bert-networks," in *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, 11 2019. [Online]. Available: http://arxiv.org/abs/1908.10084

[42] A. Parikh, O. Täckström, D. Das, and J. Uszkoreit, "A decomposable attention model for natural language inference," in *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, 2016, pp. 2249–2255.

[43] Y. Zhang, Z. Ives, and D. Roth, "Evidence-based trustworthiness," in *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*. Florence, Italy: Association for Computational Linguistics, Jul. 2019, pp. 413–423. [Online]. Available: https://www.aclweb.org/anthology/P19-1040

[44] J. Steinberger and K. Jezek, "Using latent semantic analysis in text summarization and summary evaluation," *Proc. ISIM*, vol. 4, pp. 93–100, 2004.