Verifying Correct Completion of Natural Language Instructions

Grant Storey
Cornell University
gjs224@cornell.edu

Abstract

As human-robot interaction becomes more frequent and important, it is necessary for computational systems to perform a wide variety of tasks in an intuitive way. In this paper, we explore verification of instruction completion, where a system must determine whether an instruction was properly carried out. We modify an existing dataset to create a new dataset for this verification task. We build two models, one which predicts correct completion by comparing the change in world state to a representation of the instruction and one which predicts the correct end state and compares its prediction to the actual end state. We evaluate our models against a variety of baselines and find improvement over most but not all of them, discuss the strengths and weaknesses of the models, and lay out future avenues of exploration.

1 Introduction

As interactions between humans and robots in everyday situations increase, so does the need for making these interactions easy and intuitive for humans. One common interaction is performing tasks: given a natural language instruction, can a system properly perform the instruction? A related but less-researched task is whether a system can determine if *others* have completed a given task, which is a verification problem rather than a performance problem. A system that can verify correct completion of instructions would be valuable for a variety of real-world situations; for example, a robotic worker may need to determine whether the signs placed outside the store have been changed for this week's promotional sale. In addition, a good verification system could be used to verify that one's own actions constitute a correct completion of an instruction, improving performance on the instruction completion task. The solution space for this task is far less complex than the the space for task completion, so there is hope that accuracy on this task could surpass recent work on instruction completion.

There are no datasets designed for the instruction verification task, to the best of our knowledge. In order to train and evaluate our models, we create a modification of Bisk et al.'s block dataset designed for our verification task (Bisk et al., 2016). We present two models, one which verifies correct completion by directly comparing the change in world state to a representation of the instruction and one which first determines what the end state *should be* based on the instruction and then checks whether the actual end state matches the guessed end state. Our hypothesis it that both models will far outperform simple baselines, with the former model's holistic view of the world state leading to better performance than the latter model's focus on guessing the end state independently. We compare these models with a variety of baselines, finding our hypotheses are incorrect and only the second model is reasonably competitive with the baselines. We end by discussing the model's limitations and errors, exploring the difficulties of the verification task, and discussing possible future work.

2 Related Work

To the best of our knowledge, there is no prior work on the task of verifying correct completion of natural language instructions. However, this task is closely related to, and draws inspiration from, work in situated language understanding and visual question answering.

Situated Language Understanding

Prior work has examined the task of determining object locations within world representations, including both 2D environments (Janner et al., 2017) and 3D environments (Kitaev and Klein, 2017). Similar work has examined answering user queries about the surrounding environment from a moving reference point like a car (Misu et al., 2014). However, rather than determining a specific location or information about that location, our system seeks to determine whether the a transition between two states corresponds to a natural language instruction.

There has also been a variety of research into developing systems to understand natural language instructions in order to perform tasks in a toy environment. This problem was one of the tasks evaluated at SemEval 2014 (Dukes, 2014), and there have been systems that seek to properly execute natural language instructions in both physical and digital environments (Tellex et al., 2011; Bisk et al., 2016; Pišl and Mareček, 2017). Misra et al. developed a system to take a natural language instruction and determine not just the final state but the intermediate steps needed to reach that final state (Misra et al., 2017). Our main models are based on models from the Misra and Pišl papers respectively, but seek to predict *whether* an end state is correct rather than *what* that end state (or intermediate steps and end state) is.

Visual Question Answering

Visual Question Answering focuses on producing answers to natural language questions given a picture (e.g. "what color are her eyes?"). Datasets and models for this task have appeared relatively recently (Malinowski and Fritz, 2014; Antol et al., 2015; Zhou et al., 2015), but the task is appealing due to the relatively straightforward evaluation combined with a need for good image understanding. Much of the focus is on extracting relevant information from images using approaches like attention (Shih et al., 2016; Xu and Saenko, 2016; Lu et al., 2016) or incorporating outside world knowledge to improve performance (Wu et al., 2016). Our system, in contrast, assumes relevant information about the world state has already been extracted and focuses on linking change in world state with the natural language instruction.

3 Data

Our dataset is drawn from the block world dataset presented by Bisk et al. (Bisk et al., 2016). For our evaluation, we choose the subset of the original dataset that includes instructions that move only a *single* block. This gives us Train/Dev/Test set sizes of 11871/1719/3177 instructions.

Given this initial dataset, we create a new dataset which consists of the original data points with the correct end states (the `NoChange` set) plus new data points with incorrect end states. These new data points are created through the following process: for each original data point, we add a new data point that is a copy of the original data point (same start state, same instruction) with a *new* end state. This end state is created by beginning with the start state and moving one block to a random new location that is different from its original end location. For 50% of the data, we move a different block from the original instruction (`DiffBlockMoved`), and for 50% of the data we move the same block (`SameBlockMoved`).

One potential problem is that our permuted "incorrect" answer could technically be correct due to an ambiguous instruction. The dataset does include ambiguous instructions, such as "Place the Coca Cola block west of the Esso block." A manual analysis of a random subset of 333 instructions found that 112 (34%) were technically ambiguous due to the fact that, for example, there are multiple spaces west of the Esso block. However, in all 112 cases, the block was intended to be placed *directly* to the west (or whatever the specified direction). Whenever this implied "directly" would be incorrect, all instructions specified the distance, e.g. "Put block 15 in the **second open space** below block 13." (emphasis ours). Therefore for this dataset we can safely assume that potentially ambiguous instructions include an implied "directly." There would need to be additional data curation if this model were to be extended to a dataset where this assumption does not hold.

Finally, we perform some preprocessing on the data. We remove the y coordinate (always 0.1) from the state space. Some data points in the original set have fewer than the maximum number of blocks (20). We add the necessary number of blocks at position (-1000, -1000) to ensure that all states have 20 blocks. Across all data we separate punctuation marks as their own tokens, lowercase all tokens, and then replace words that

appear only once in the training set with the token UNK. This threshold for UNK tokens was chosen based on a manual analysis of the dataset, which showed that many words that appeared two or three times appeared to provide useful information, such as "catty-cornered," or referred to a specific block but contained a misspelling, such as "mconalds." There are 1068 unique words in the Train set, of which 667 appear at least twice. For the neural models below (unless otherwise noted) we also use GloVe word embeddings: 630 of the 667 words have corresponding GloVe vectors. The remaining words are mostly either hyphenations ("cube-spaces") or misspellings ("heinikin"), and we initialize these to random embeddings.

4 Models

We describe in depth the two full models for which we report results below. For all neural models, including those in Section 5.1, we utilize early stopping to pick the model with the best accuracy on the development set.

4.1 FullView

The intuition behind the FullView model is to get a representation of how the world state changes and a representation of what the instruction says should change, then compare these two to see if they match. It has three parts:

1. **World State Representation Prediction**: Given the start and end state, use a feed-forward neural network to create a representation of the change between the start and end (the *World State Representation*). The feed-forward network has a single hidden layer of dimension 64 and an output size of dimension 64, and dropout is applied to the hidden layer with probability 0.1.

2. **Instruction Representation Prediction**: Given the instruction, use a bi-directional LSTM to create a representation of the instruction (the *Instruction Representation*). The LSTM has a dimension of 128, and the instruction representation is the concatenation of the first and final states. Noise of 0.3 is applied to the word embeddings.

3. **Final Answer Prediction**: Given the World State Representation and Instruction Representation, use a feed-forward neural network with softmax to determine the probability that these representations correspond to correct completion of the instruction. The network has a hidden dimension of size 64 and an output dimension of 2, and dropout is applied to the hidden layer with probability 0.1.

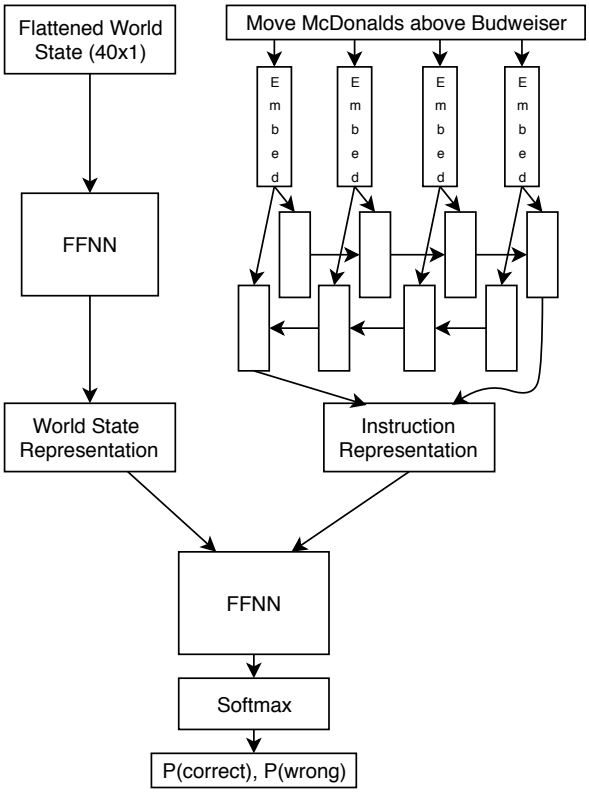

Figure 1: Overview of the FullView Model.

See Figure 1 for a visual overview.

We first pretrain the world representation and instruction representation predictors to predict which block was moved (for 100 and 5 epochs respectively). We then pretrain the final answer prediction using fixed outputs from the pretrained representation generators for 100 epochs. Finally, we train the full model together for 200 epochs. The intuition is that the model will first learn to simply check whether the proper block was moved during pretraining, and then learn potentially more complicated rules during joint training.

This model's structure is based on Misra et al.'s for predicting local movements of blocks in the block world (Misra et al., 2017).

Hyperparameters: layer dimensions, dropout, and noise were optimized on the development error using a grid search. Adding additional hidden layers was attempted but did not improve performance.

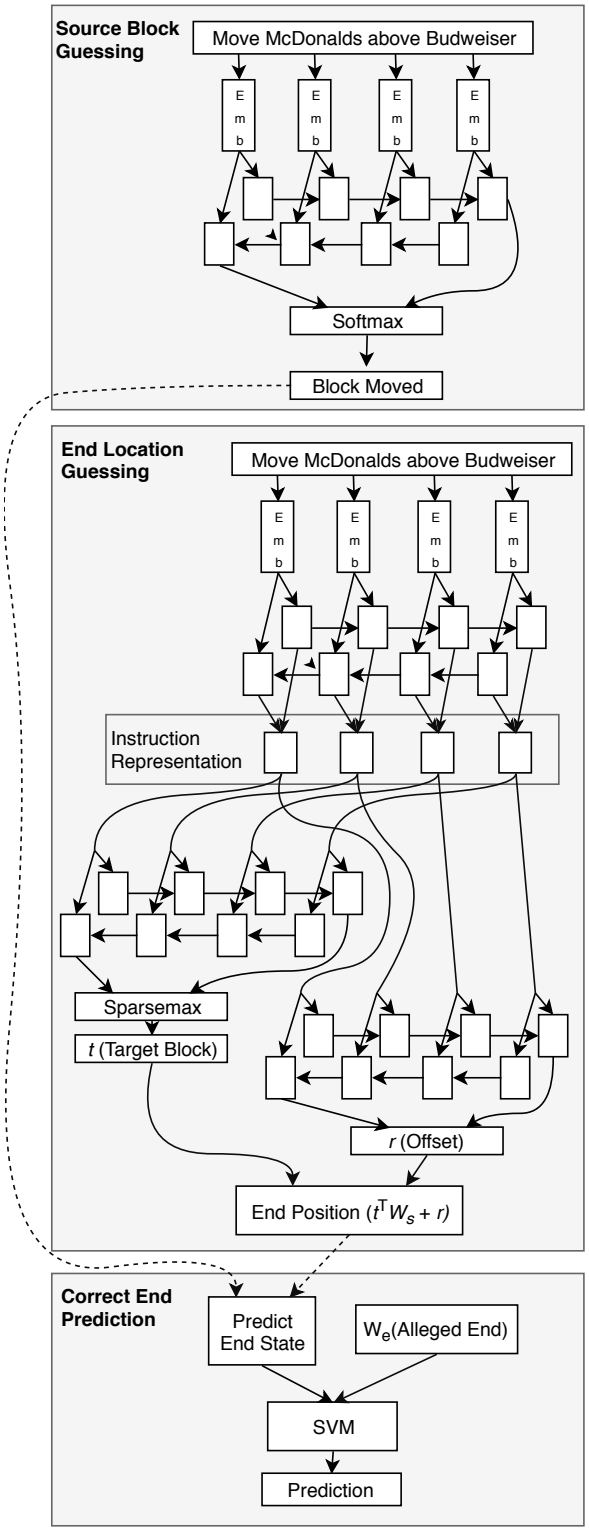

Figure 2: Overview of the GuessEnd Model.

4.2 GuessEnd

This model first predicts the proper end state given the start state and instruction. It then compares its prediction with the provided end state to determine whether that end state is correct or not. For this model we *do not* initialize the word embeddings with the GloVe embeddings, as doing so leads to slightly worse test error (see Discussion below). The model consists of three pieces:

1. **Source Block Guessing**: Given the instruction, use an LSTM to predict the block to be moved. The model feeds the last states of the forward and backward parts of a single-layer bi-directional LSTM into a softmax layer. It is trained for 50 epochs with 64 dimensions and we apply a noise of 0.3 to the embeddings during training.

2. **End Location Guessing**: Given the instruction, we use a single-layer bi-directional LSTM with 128 dimensions to extract an instruction representation, which is created by concatenating the outputs of the forward and backward LSTMs. Another single-layer bi-directional LSTM takes this instruction representation as input and feeds into a sparsemax layer to generate an output t of length 20, where each index i corresponds to how much block i is used as a reference for the final location. A third single-layer bi-LSTM starts with the instruction representation and predicts the offset from the target block, r, of length 2. The model predicts the final end location with the equation $\vec{t}^T W_s + r$, with W_s being a 20×2 matrix corresponding to the starting state of the world. The intuition is that the final position is usually described as a position relative to some target block, so $\vec{t}^T W_s$ gives us the location of the target block and r gives us the offset from the target block. The model is trained for 200 epochs. This is based on the model used by Pišl and Mareček for predicting block movement (Pišl and Mareček, 2017).

3. **Correct End Prediction**: Given a source block moved and the predicted end location of the source block, construct a guessed end state (W_g), then subtract the alleged end state (W_e) to get an *end state difference*. Since there should be a linear hyperplane between correct and incorrect answers, we use a soft-margin linear SVM to predict whether this end state difference corresponds to a correct or incorrect end. The linear SVM uses an L1 regularizer with C=0.1.

See Figure 2 for a visual overview of the GuessEnd model.

Hyperparameters: once again, layer dimensions, dropout, and noise were optimized on the development error using a grid search. Adding additional hidden layers was attempted but did not lead to improved performance. SVM regularizer and C are also chosen using a grid search.

5 Evaluation

In order to evaluate our model's performance, we report the accuracy of our model in predicting whether the end state represents a valid completion of the instruction. We evaluate model performance on all data points in the development (dev) and test sets. In addition, we report model accuracy on the subset of the data with correct answers (`NoChange`), incorrect answers where a different block was moved compared to the original instruction (`DiffBlockMoved`), and incorrect answers where the block moved is the same as the original instruction (`SameBlockMoved`).

5.1 Baselines

We compare our model's performance to the following baselines:

- **GuessYes**: This model guesses that every data point represents the correct completion of the instruction.
- **GuessRandom**: This model guesses that a data point represents the correct completion of the instruction with probability 0.5.
- **NeuralBaseline**: A neural model that feeds only the start and end word state into a feed-forward neural network with a single hidden layer of dimension 128. This model uses no language information.
- **LogisticRegression**: A logistic regression model, where the set of features consists of the start and end states and a bag of words representing the instruction.
- **LinearSVM**: A linear SVM, where the set of features again consists of the start and end states and a bag of words representing the instruction.
- **MovedLearner**: The intuition for this model is to determine which block the instruction intends to be moved and then return true only if that block was moved. This model has an upper bound of 75% accuracy, as it would get a perfect score on points in `NoChange` and `DiffBlockMoved` but get every point in `SameBlockMoved` wrong. We predict an instruction's block to move by taking the softmax of a linear transformation of the final states of the forward and backward parts of a single-layer bi-directional LSTM. It is trained for 50 epochs with 64 dimensions and we apply a noise of 0.3 to the embeddings during training. We then determine the block that is actually moved by comparing the start and end states. Finally, we return true if the predicted block to move and the block moved are the same.

Model	Dev	Test
GuessYes:	50.00%	50.00%
GuessRandom:	49.74%	50.74%
NeuralBaseline:	50.00%	50.00%
LogisticRegression:	62.59%	60.07%
LinearSVM:	59.05%	58.01%
MovedLearner:	**74.26%**	**74.03%**
FullView:	56.54%	53.29%
GuessEnd:	69.90%	69.25%

Table 1: Accuracy of various models on the development and test sets. Highest scores are in **bold**.

6 Results and Discussion

The full test and development set results for our models and the various baselines are visible in Table 1. The best model on both datasets is the MovedLearner baseline, which successfully classifies about 74% of the points. The GuessEnd model is the next best model, with just under 70% accuracy, which is 9 percentage points better than the next closest baseline (Logistic Regression), but still not quite as good as the MovedLearner baseline. The FullView model performs very poorly, barely beating the baselines that do not use any language knowledge at all.

In order to break down the results further, we consider the results divided by type in Table 2. Ignoring the GuessYes baseline and the NeuralBaseline (which just learns to be a GuessNo baseline), the MovedLearner models performs best on the NoChange and DiffBlockMoved subsets, as it was designed to do. It performs very poorly on the SameBlockMoved subset because it assumes that

Model	NoChange		DiffBlockMoved		SameBlockMoved	
	Dev	**Test**	**Dev**	**Test**	**Dev**	**Test**
GuessYes:	*100.00%*	*100.00%*	0.00%	0.00%	0.00%	0.00%
GuessRandom:	49.91%	50.77%	49.30%	51.86%	49.83%	49.56%
NeuralBaseline:	0.00%	0.00%	*100.00%*	*100.00%*	*100.00%*	*100.00%*
LogisticRegression:	60.44%	63.27%	70.35%	61.11%	59.14%	52.64%
LinearSVM:	54.92%	65.88%	68.02%	54.31%	58.32%	45.97%
MovedLearner:	**96.97%**	**96.98%**	**99.53%**	**99.62%**	3.49%	2.52%
FullView:	26.82%	21.50%	93.14%	88.67%	**79.39%**	**81.49%**
GuessEnd:	74.23%	73.43%	76.86%	77.72%	54.25%	52.39%

Table 2: Accuracy of various models on the development and test sets for each of the three subsets of the data. Highest scores ignoring baseline 100%s are in **bold**, and baseline scores of 100% are in *italics*.

movement of the same block is correct (the small correct percentage happens when it predicts that the wrong block should be moved). Of the neural models, the GuessEnd model is the only one to score above a 50% on all three subsets, so it is the best if we require accuracy across all subsets.

6.1 FullView Error Analysis

In theory, the FullView model is a more powerful version of the MovedLearner model, replacing subtraction and an argmax operation with neural networks. However, we see much worse performance in practice, even with extensive pretraining. Although it performs well on the DiffBlockMoved and SameBlockMoved datasets, it is really only a slightly improvement over the hypothetical "Guess No 80% of the time" baseline, as it has very poor performance on NoChange. The problem seems to be that the loss function does not provide enough information to properly find the correct function. Even the slightly-better-than-random performance of the FullView model above happens because of early stopping, as a few epochs after this performance it collapsed back down to 50% accuracy. It appears the model has an issue with catastrophic forgetting, which makes some sense as it is learning a massive number of parameters with only yes/no as feedback.

A closer look at the instructions shows that the mistakes for DiffBlockMoved and SameBlockMoved tend to be more complicated instructions, like "The '4' block moves down until in line with the '6' and '8' blocks." and other instructions that refer to arranging blocks in a line. It also has issues with instructions like "Take block 8 and move it to the very bottom, center." which have global references. That being said, it still occasionally makes mistakes for easier instructions like the instruction "Place the 16 to the right side of the 11.", which it says is performed correctly but actually has block 16 far to the left of block 11. For the NoChange dataset, it gets instructions of all types wrong, which is to be expected from a classifier with 20% accuracy.

We also built a version of FullView which instead of having pretraining had weights initialized such that it emulated the MovedLearner model. This model's accuracy by epoch, with the initial accuracy before any training is done, is visible in Figure 3. After a single training epoch, the model forgets everything and becomes a 50/50 classifier. This is probably due to some combination of the learning rate, loss function, and lack of information from the single yes/no answer, but shows the challenges that this model setup faces with learning a series of complicated representations from very little feedback.

6.2 GuessEnd Error Analysis

The GuessEnd model is the best-performing model that gets at least 50% accuracy on all three subsets of the data, so it does seem to work reasonably well. The improvement over the FullView model seems to be at least partially based on the fact that this model gets more detailed feedback from the loss function when guessing positions and helps avoid the catastrophic forgetting of the FullView model. One particularly interesting part of the model is that it performs slightly better *without* initializing the word embeddings to pre-trained GloVe embeddings. The difference is quite small (16 extra instructions correct out of 6354 total, or 0.25%), which can probably be attributed to random variance in the model, but it

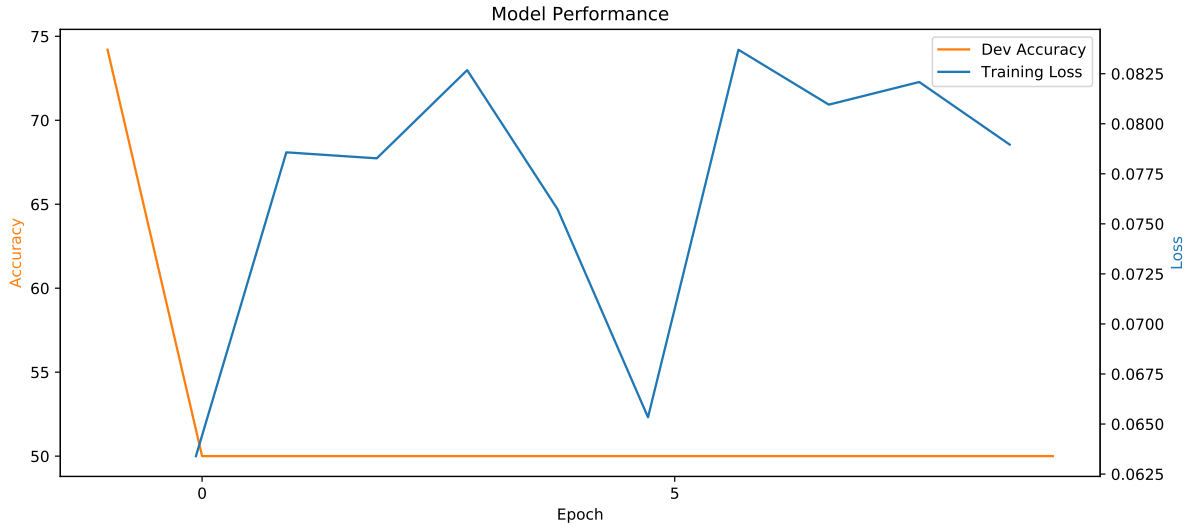

Figure 3: Development Accuracy (orange) and Training Loss (blue) over the first 10 epochs of a version of the FullView Model initialized to match the MovedLearner baseline. Accuracy before training is shown as epoch -1.

does show that the embeddings are not a major piece of the model. This is likely because there is such a small vocabulary (667 words plus UNK), so the model can directly learn the embeddings it needs rather than relying on pretrained embeddings to provide similarity information about less frequently seen words.

Since this model only gets about 70% accuracy, a manual analysis of its failures shows it has many different kinds of errors. The model does not properly classify the data points with instructions "Below block 17, place block 18." or "Move 18 so it is above 19", which shows that it is missing out on even some pretty simple sentences, but it does manage to correctly classify "Above block 15, place block 14.", so it is inconsistent with some of these simpler sentences rather than failing them across the board. Some of the failures are due to complicated instructions: commands like "Move block 12 to become the fifth block in the series 8, 9, 10 and 11" which require higher level understanding, or instructions like "Move the 15 block up until it is even with the 14 block, then move it all the way right, then move it down until is even with the 17 block" which do not have a simple target block that the model can find, are generally points of failure. There are also some instructions that trick the model's target block technique; for example, one data point has the instruction "Move the 20 block to the right 2 spots." and the block that is (wrongly) moved is block 2, so the model appears to have confused the distance unit for a reference block. However, our manual analysis found that only around 50% of the failures were on more difficult instructions, so there is clearly still room for improvement in the model with the easier instructions.

This model's performance also does not quite reach that of the model it is based on. The Source Block Guessing module has an accuracy of about 96%, and the End Location Guessing module's guess has a mean distance of 1.37 from the correct answer and a median distance of 0.06. These numbers are not quite as good as the original Pišl and Mareček paper, which had a source accuracy of 98.8%, mean distance of 0.72, and median distance of 0.04 (Pišl and Mareček, 2017). The discrepancy may be due to differences in preprocessing or hyperparameters, as these are not well-explained in the paper. Padding missing blocks with blocks at (-1000, -1000) may also lead to the increased average distance if a few of those points are misclassified.

One last concern with the GuessEnd model is that it tackles the verification problem indirectly by running the instruction performing task and checking whether the target answer matches the guessed answer. This means that the really important piece of the GuessEnd model is a model that can perform well on the instruction completion task, and thus the validation task is not particularly interesting on its own. Until a

version of the FullView or some other model that does not include end prediction shows better performance, the superior performance of the GuessEnd model makes the task itself seem less viable as an independent avenue of research.

7 Future Work

The first main task, of course, is to improve our models so that they can outperform all of the baselines, hopefully by a significant amount. Attempting to run deeper neural networks with larger hidden dimensions would be a potential improvement to try, but would need to be done on a more computationally powerful computer with GPU support. Combining some hard-calculated features like the information from the MovedLearner with separate inputs from a neural model would be a natural next step, but this would create a system with piecewise understanding rather than a system which could create widely applicable meaning representations. Another potential avenue is to look at an encoder/decoder model which would encode states and instructions into a joint latent space so that either states or instructions could be recovered from the same representation. Other areas of improvement include more complex preprocessing of the data, such as using a spell-checker to correct mistakes like "postion" and "heinikin" which Pišl et al. found to improve performance (Pišl and Mareček, 2017).

Finally, if the models achieved satisfactory performance on this dataset, the next step would be applying these techniques to the more complicated 3-dimensional blocks dataset from Bisk et al. (Bisk et al., 2017) or to even more diverse environments like the spatial reasoning dataset from Janner et al. or CHALET (Janner et al., 2017; Yan et al., 2018).

8 Conclusion

In this work, we explored techniques for verifying correct completion of natural language instructions. We presented two main models along with a variety of baselines, finding that one model performed poorly compared to all but the simplest baselines and that the other model performed better than most baselines but still has room for improvement. We discussed some of the limitations of our models, along with the difficulty of training complex models for the task given that it only provides binary feedback. We also found that high performance on this task by the GuessEnd model means that it is perhaps better to treat the verification problem as an application of the instruction completion task rather than building models that focus only on the end goal of verification. Finally, we discussed future directions for this research both in the short- and long-term.

Reflection

This section is for slightly more informal observations about the project that did not fit into the discussion or conclusion.

The first big lesson from this project was to properly analyze new datasets before building full models on them. The dataset used in this work is actually the second one I tried; I spent about a week doing hyperparameter optimization for models that performed well on the first dataset before coming up with the NeuralBaseline baseline and finding that it performed almost as well as my models *without using any language modeling at all*. This led to the creation of the current, better dataset (where the models without language information get at best 50%), but a lot of time was sunk into tuning models that really were not all that good.

The second big lesson is that neural networks are not just magic boxes, which is how they are presented in a lot of the papers we read. Considering activation functions and layers and depths and loss functions and setup as a whole is actually very important, because otherwise there are unintentional limitations built into the model that do impact performance. I also found that adding more layers and increasing layer size did not seem to have a positive impact for the problem. One possible reason is that the latent space is reasonably small: there are only 20 blocks , so massive representations are not necessary. It is also possible that 2 layers of 1024 dimensions does not show much improvement over 1 layer of 128 dimensions while 4 layers of 4096 would, but the training time for the latter system would have been too much for my computer to handle (some of these models already took over 24 hours to train and they were far smaller). So neural networks are powerful, but expecting them to just learn any function through magic without putting though and care into model design is unrealistic.

Lastly, I wanted my model design to be clean

and intuitive and generalizable, which made me less open to exploring models that took advantage of specific information about the data. I still would be a little unhappy with a system that used lots of non-generalizable assumptions, but if my goal was to get the best possible performance on this dataset, making the assumption that the blocks could be in a limited number of positions on the grid and dividing instructions into general classes to be handled separately might have been helpful.

Acknowledgements

I would like to thank Professor Cardie for all of her help this semester and my classmates for their useful feedback on the initial proposal for this project.

Code

The code for this project is available at https://github.coecis.cornell.edu/gjs224/6740-Verification

References

Stanislaw Antol, Aishwarya Agrawal, Jiasen Lu, Margaret Mitchell, Dhruv Batra, C Lawrence Zitnick, and Devi Parikh. 2015. Vqa: Visual question answering. In *Proceedings of the IEEE International Conference on Computer Vision*. pages 2425–2433.

Yonatan Bisk, Kevin J Shih, Yejin Choi, and Daniel Marcu. 2017. Learning interpretable spatial operations in a rich 3d blocks world. *arXiv preprint arXiv:1712.03463* .

Yonatan Bisk, Deniz Yuret, and Daniel Marcu. 2016. Natural language communication with robots. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. pages 751–761.

Kais Dukes. 2014. Semeval-2014 task 6: Supervised semantic parsing of robotic spatial commands. In *Proceedings of the 8th International Workshop on Semantic Evaluation (SemEval 2014)*. pages 45–53.

Michael Janner, Karthik Narasimhan, and Regina Barzilay. 2017. Representation learning for grounded spatial reasoning. *arXiv preprint arXiv:1707.03938* .

Nikita Kitaev and Dan Klein. 2017. Where is misty? interpreting spatial descriptors by modeling regions in space. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*. pages 157–166.

Jiasen Lu, Jianwei Yang, Dhruv Batra, and Devi Parikh. 2016. Hierarchical question-image co-attention for visual question answering. In *Advances In Neural Information Processing Systems*. pages 289–297.

Mateusz Malinowski and Mario Fritz. 2014. A multi-world approach to question answering about real-world scenes based on uncertain input. In *Advances in neural information processing systems*. pages 1682–1690.

Dipendra K Misra, John Langford, and Yoav Artzi. 2017. Mapping instructions and visual observations to actions with reinforcement learning. *arXiv preprint arXiv:1704.08795* .

Teruhisa Misu, Antoine Raux, Rakesh Gupta, and Ian Lane. 2014. Situated language understanding at 25 miles per hour. In *Proceedings of the 15th Annual Meeting of the Special Interest Group on Discourse and Dialogue (SIGDIAL)*. pages 22–31.

Bedřich Pišl and David Mareček. 2017. Communication with robots using multilayer recurrent networks. In *Proceedings of the First Workshop on Language Grounding for Robotics*. pages 44–48.

Kevin J Shih, Saurabh Singh, and Derek Hoiem. 2016. Where to look: Focus regions for visual question answering. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. pages 4613–4621.

Stefanie Tellex, Thomas Kollar, Steven Dickerson, Matthew R Walter, Ashis Gopal Banerjee, Seth J Teller, and Nicholas Roy. 2011. Understanding natural language commands for robotic navigation and mobile manipulation. In *AAAI*. volume 1, page 2.

Qi Wu, Peng Wang, Chunhua Shen, Anthony Dick, and Anton van den Hengel. 2016. Ask me anything: Free-form visual question answering based on knowledge from external sources. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. pages 4622–4630.

Huijuan Xu and Kate Saenko. 2016. Ask, attend and answer: Exploring question-guided spatial attention for visual question answering. In *European Conference on Computer Vision*. Springer, pages 451–466.

Claudia Yan, Dipendra Misra, Andrew Bennnett, Aaron Walsman, Yonatan Bisk, and Yoav Artzi. 2018. Chalet: Cornell house agent learning environment. *arXiv preprint arXiv:1801.07357* .

Bolei Zhou, Yuandong Tian, Sainbayar Sukhbaatar, Arthur Szlam, and Rob Fergus. 2015. Simple baseline for visual question answering. *arXiv preprint arXiv:1512.02167* .