Visually Grounded Language Learning For Robot Navigation

Emre Ünal
Ozan Arkan Can
Yücel Yemez
{emunal,ocan13,yyemez}@ku.edu.tr
Koç University
Istanbul, Turkey

ABSTRACT

We present an end-to-end deep learning model for robot navigation from raw visual pixel input and natural text instructions. The proposed model is an LSTM-based sequence-to-sequence neural network architecture with attention, which is trained on instruction-perception data samples collected in a synthetic environment. We conduct experiments on the SAIL dataset which we reconstruct in 3D so as to generate the 2D images associated with the data. Our experiments show that the performance of our model is on a par with state-of-the-art, despite the fact that it learns navigational language with end-to-end training from raw visual data.

CCS CONCEPTS

• **Computing methodologies → Artificial intelligence**; **Intelligent agents**; **Natural language processing**; **Computer vision tasks**.

KEYWORDS

robot navigation, instruction following, natural language processing, visual grounding

ACM Reference Format:
Emre Ünal, Ozan Arkan Can, and Yücel Yemez. 2019. Visually Grounded Language Learning For Robot Navigation. In *1st International Workshop on Multimodal Understanding and Learning for Embodied Applications (MULEA '19), October 25, 2019, Nice, France.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3347450.3357655

1 INTRODUCTION

Visual grounding plays a major role in language learning and understanding. This is especially important for robot navigation from verbal instructions which often contain references to objects in the environment in terms of their attributes and spatial properties. In this work, we present a deep learning model for training a robot to follow navigational instructions given in natural language by taking visual data also into account. One distinctive aspect of the proposed model is that it learns navigational language with end-to-end training from raw images, in contrast to the common practice which assumes high-level visual information is readily available as meta information in terms of object ids and attributes.

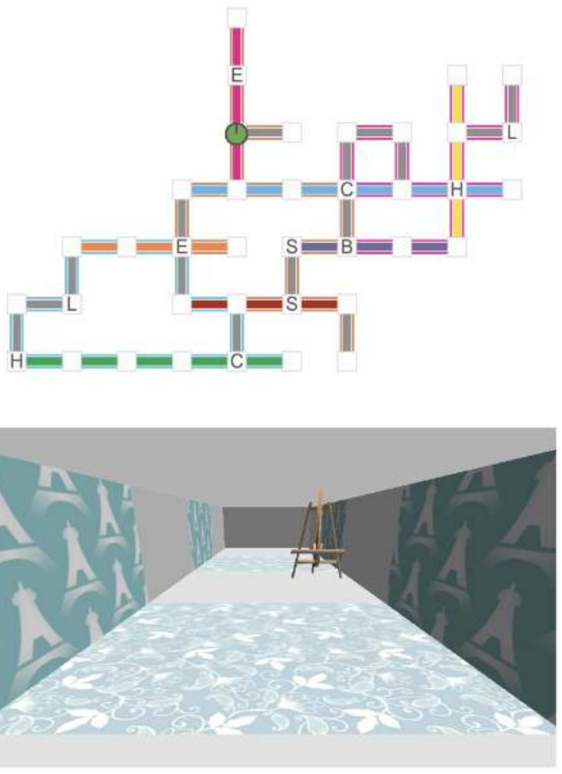

Figure 1: 2D and 3D views of one of the SAIL maps (Jelly). The objects are indicated with letters in the 2D representation. Letters E, C, H, L, S, B **stand for** *Easel*, *Chair*, *Hatrack*, *Lamp*, *Sofa*, *Barstool*, **respectively. Each color represents a different floor and wall pattern. The agent is displayed as a green circle in the 2D representation.**

For our experiments, we use the SAIL dataset [20] by extending it with synthetic images. The dataset includes navigational instructions in free-form natural language, which are collected in a maze-like environment. The environment includes different objects, wall and floor patterns. The agent (robot) receives the visual

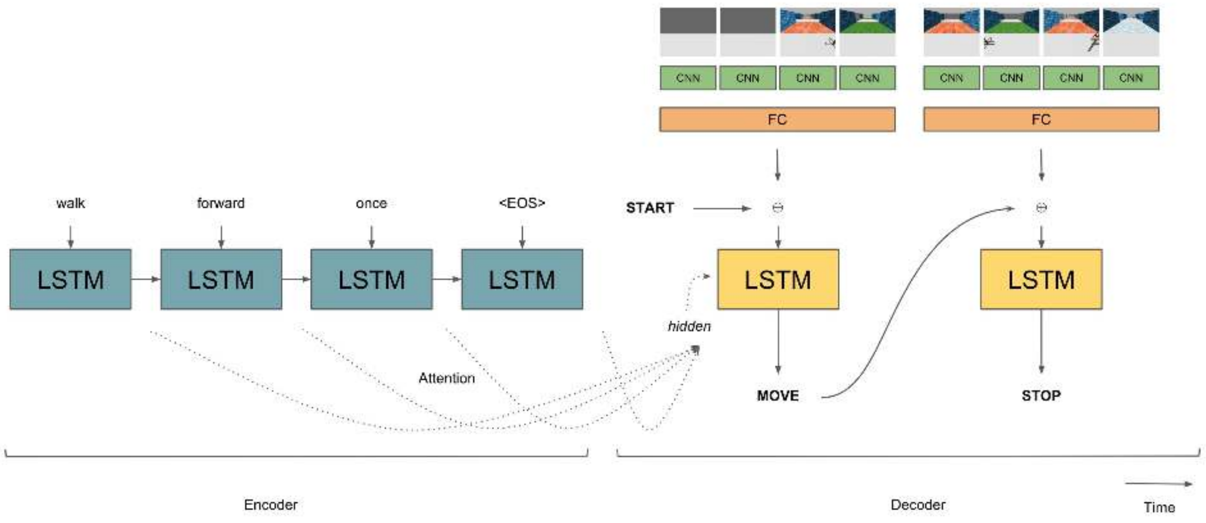

Figure 2: The model learns to encode a navigational instruction in order to decode it into a sequence of actions by taking the previous action and the current environment as input. The system also includes a textual attention mechanism. Using the perceptual input and attending to the language, the decoder predicts the correct actions so as to follow the instructions.

information through its sensors and does not fully see the map. It observes only part of the virtual environment using its camera as shown in Figure 1. The agent is expected to follow the instructions by taking a sequence of actions. The agent can execute four different actions which are LEFT, RIGHT, MOVE and STOP. The LEFT and RIGHT actions rotate the agent by 90 degrees with respect to its current orientation. The MOVE action moves the agent one step forward. STOP is a special action that stops the agent and ends the sequence. No other action is taken after the STOP action.

The neural network architecture presented in this work is inspired by the previous work [21] [9] [6]. Our main contribution is to encode raw visual data - pixel images - before feeding into the decoder, by using a pre-trained convolutional neural network that we customize for our dataset. Thus we do not rely on some high-level visual meta-information assumed to be readily available. For this purpose, we reconstruct the 2D SAIL maps in 3D, collect syntactical images and then train the model with them in an end-to-end fashion. We also use an adequate attention mechanism tailored for the visually grounded navigation task. The use of attention is particularly important in this task as the agent may need to attend (or consult) to different parts of the given instruction as the environment keeps changing (e.g., new objects appear) during navigation.

1.1 Dataset

The SAIL dataset consists of three different maps *(Grid, L, Jelly)*. We extend SAIL dataset by reconstructing the maps in 3D. For this purpose, we use GAZEBO[1] which is a robot simulation environment that works with the Robot Operating System (ROS). It supports programmatic control through ROS messaging.

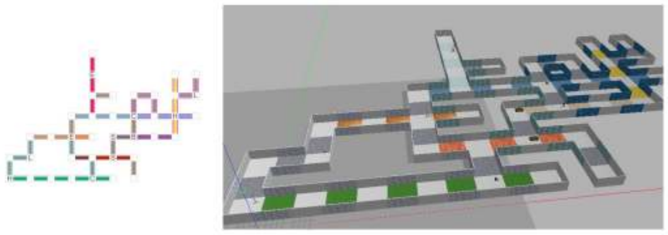

Figure 3: 3D reconstruction of the Jelly map. The floor and wall patterns are selected as mentioned in the original SAIL dataset. The objects are placed to their specified nodes accordingly.

We have reconstructed the SAIL maps in GAZEBO along with their wall and floor patterns. We have also placed the 3D models of the 6 different objects *(chair, lamp, easel, sofa, hatrack, barstool)* mentioned in the original dataset as shown in Figure 3.

By programmatically controlling the agent in the virtual environment, we have captured images from each direction for each node of all the SAIL maps. In this process, the agent is moved to a node in the map and then the 4 images corresponding to FRONT, LEFT, BACK and RIGHT are captured based on the current orientation through successive counter-clockwise rotations of 90 degrees. While capturing images, the camera is positioned such that it makes an angle of 30 degrees with the horizontal axis. For all instances in

[1]http://gazebosim.org/

the original dataset, we then associate each captured image with the corresponding node and the training sample.

2 MODEL

The task of visually grounded navigation can be formulated as a Markov decision problem. We predict an action sequence, where the next action depends on the previously taken action(s) and how the perceived environment changes based on that action. Therefore, we use a sequential encoder-decoder model [25]. First, the encoder processes the textual instruction word by word. Then the decoding process starts, where the current hidden state is computed based on the encoder hidden states with attention as well as on the previous hidden state. We use a convolution neural network to extract features from raw images. We then feed these visual features concatenated with the previous action and current orientation to the decoder as input. At each time step, the decoder LSTM [13] predicts an action and this continues until STOP action is output. If the STOP action is never predicted, the sequence is cut at the maximum sequence length. The architecture is shown in Figure 2. The details of the architecture are discussed in the following subsections.

2.1 Encoder

Each verbal instruction in the dataset is a sentence, which is fed as input to the encoder word by word. Bidirectional LSTM [11] is used as the encoder and each word is represented as a one-hot vector. We concatenate the hidden vectors of the back and forward moving LSTMs and store them for later usage by the attention mechanism.

A sentence is represented as a sequence of one-hot vectors ($x = (x_1, x_2, .., x_N)$), where N is the number of words in the sentence. Bidirectional LSTM processes the input in backward and forward directions with the following formulas:

$$\begin{aligned} f_i &= \text{LSTM}(W_f, x_i, f_{i-1}) \\ b_i &= \text{LSTM}(W_b, x_i, b_{i+1}) \\ h_i^e &= f_i \oplus b_{N-i} \end{aligned} \tag{1}$$

where W_f and f_i are the weight matrix and the hidden state vector of the forward LSTM; W_b and b_i are those for the backward LSTM. The hidden state of the encoder is denoted by h_i^e and computed with the concatenation ($\oplus$) of f_i and b_{N-i}. The hidden state vectors of both LSTMs are initialized as zero.

2.2 Attention Mechanism

We use a global attention mechanism that adaptively attends to the hidden states of the encoder and contributes to the hidden vector of the decoder at each time step t [4] [19] [30] [27]. All hidden states of the encoder are taken into account when calculating the attention. First, the alignment a_{it} between each hidden state vector of the encoder h_i^e and the hidden state vector of the decoder h_t^d at time t is computed as follows:

$$\begin{aligned} a_{it} &= \text{align}(h_t^d, h_i^e) \\ &= \frac{\exp(\text{score}(h_t^d, h_i^e))}{\sum_i \exp(\text{score}(h_t^d, h_i^e))} \end{aligned} \tag{2}$$

Here score function is given by

$$\text{score}(h_t^d, h_i^e) = \text{dot}(h_t^d, h_i^e)/\sqrt{N} \tag{3}$$

where N is the number of the encoder hidden states, hence the number of words in the given instruction, and dot function returns the dot product of two vectors. The alignments $\{a_{it}\}$ are then used to compute a context vector c_t at each time step t. The context vector is the weighted sum of the encoder hidden states:

$$c_t = \sum_i a_{it} h_i^e \tag{4}$$

Finally, the attentional hidden state $\hat{h}_t^d$ of the decoder at time t is computed by

$$\hat{h}_t^d = \tanh(W_c[c_t; h_t^d]) \tag{5}$$

where the context vector c_t and the hidden state h_t^d are first concatenated and then multiplied by the weight matrix W_c so as to obtain $\hat{h}_t^d$. The weight matrix W_c is learned during the end-to-end training process.

2.3 CNN

ResNet[12] is a convolutional neural network architecture that is trained on ImageNet [24]. It is one of the best performing models in recent years for the image recognition task. We use the pre-trained ResNet-152 architecture to extract features from the images collected in the virtual environment. For this purpose, the output of the last pooling layer of ResNet-152 is employed as the feature vector.

In order to fine tune the CNN architecture for this task, we use a fully connected layer after the convolutional layers. Based on the agent position and orientation, we extract the image feature vectors corresponding to each direction (FRONT, LEFT, BACK and RIGHT) and concatenate them before feeding to the fully connected layer. We use a single hidden layer with ReLU activation [23]. The fully connected layer is learned during the end-to-end training process whereas the weights of the CNN architecture are frozen.

The output of the last fully connected layer is finally concatenated with the one hot representations of the previous action and the agent's orientation. The resulting vector becomes the input to the decoder for the corresponding time step.

2.4 Decoder

At each time step t, the decoder hidden state is initialized with $\hat{h}_{t-1}^d$ that is computed using the attention mechanism described previously. Given the input, the decoder then predicts an action sequence. The decoder input is formed by concatenating the previous action α_{t-1} (one-hot vector), the current orientation o_t (one-hot vector) and the output of the fully connected layer v_t. These steps are more explicitly expressed in the sequel. First, the input images are processed:

$$\begin{aligned} u_t^l &= \mathrm{CNN}(W_{\mathrm{CNN}}, I_t^l) \\ u_t &= u_t^{\mathrm{F}} \oplus u_t^{\mathrm{L}} \oplus u_t^{\mathrm{B}} \oplus u_t^{\mathrm{R}} \\ v_t &= \mathrm{FC}(u_t) \end{aligned} \tag{6}$$

where I_t^l is the input image at time step t and the index l stands for F, L, B and R corresponding to FRONT, LEFT, BACK and RIGHT images, respectively. W_{CNN} is a set of matrices that stores the pre-trained CNN weights. The vector u_t^l is the output of the CNN for each directional image; u_t is the concatenation of these CNN outputs; and v_t is the output of the fully connected (FC) layer - the weights of the FC layer are learned with end-to-end training.

The output of the fully connected layer, hence the visual features, are then fed into the decoder along with the previous action and the current orientation:

$$y_t, h_t^d = \mathrm{LSTM}(W_d, v_t \oplus o_t \oplus \alpha_{t-1}, \hat{h}_{t-1}^d) \tag{7}$$

where y_t is the standard output of LSTM and the matrix W_d holds the LSTM parameters learned with end-to-end training. The decoder then defines a probability over the generated action sequence, $\alpha = (\alpha_1, \alpha_2, .., \alpha_T)$, that is executed by the agent:

$$P(\alpha) = \prod_{t=1}^{T} P(\alpha_t \mid v_t, \alpha_{t-1}, o_t, \hat{h}_t^d) \tag{8}$$

where T is the number of steps in the sequence, hence the number of actions generated for the given instruction. Eq. (8) gives the probability of the action sequence as the product of conditional probabilities, where the conditional probability of each action is obtained by applying the softmax operation to the decoder output:

$$P(\alpha_t \mid v_t, \alpha_{t-1}, o_t, \hat{h}_t^d) = \mathrm{softmax}(y_t) \tag{9}$$

2.5 Training

The LSTMs and FC layer of the model are trained[2] in an end-to-end fashion over visual data and text input. We convert the locations and orientations in a given path followed by the agent into a sequence of actions. By using the gold action sequence during the training, the weights of the model are optimized by minimizing the negative log likelihood. This results in maximizing the probability of the ground truth action sequence. The weights of the LSTM models are learned with backpropagation through time [29]. We use Adam optimization [17] with default parameters. Gradient clipping method is used for the LSTMs with the norm threshold 5 [22]. The model is trained for 100 epochs and the best model is picked based on the development dataset performance. We use a vector of size 100 for the encoder hidden state and 200 for the decoder. The number of hidden nodes in the fully connected layer is set to 500 and the output size to 100.

2.6 Inference

We test the trained model and generate sequence of actions by using beam search [21, 25]. During the decoding process, at each time step, the beam search strategy keeps a record of K best alternative solutions so far (not only the best) and gives its decisions searching over these alternatives. On contrary to the greedy approach, the beam search algorithm allows the model to correct the mistakes it might have made previously during navigation. In our case, the beam search ends when all the top $K = 10$ beams get the STOP action as the last action or the beam length reaches the maximum sequence length (which is 30 in our experiments).

[2]The model is implemented in Julia using Knet [31].

3 EXPERIMENTS & RESULTS

We have trained two different models. The first model relies only on the text input to predict the action sequence without using any visual information. The other is the complete visually grounded model which incorporates the visual data as well. The language-only model is the baseline that we employ to compare with the complete model. Language-only model takes only the previous action as input to the decoder. For this model, the CNN and the fully connected layer are omitted but the attention mechanism is still in place. Each model is trained using two maps. The data for the other map is split into halves for development and test purposes. This is repeated three times for three different map pairs, and then the overall success rate is reported by averaging the results. The success rate is defined as in the previous studies: A test instance is counted as successful if the agent is able to reach the correct destination with the right orientation. All the performance reported in this paper are on the *Single Sentence* partition of the SAIL dataset. The performances of the two models are compared in Table 1. We observe that visual grounding boosts the performance by about 5%. We also note that almost one-third of the text instructions in the SAIL dataset does not include any reference to the environment, so the contribution of visual grounding is actually more than it appears to be in the table.

Table 1: Performance results for language-only model vs. visually grounded model

Model	Success rate (%)
Language Only	60.50
Visually Grounded	65.45

In Table 2, we compare our visually grounded model with the previous works that have so far reported results on the same dataset. The main difference of our model, when compared to all these methods, is that we use raw pixel images for the perceptual information (hence no explicit scene analysis is needed) whereas the others employ high-level visual meta-information available with the SAIL dataset (bag of features). Only Can and Yuret [6] use their custom grid representation which is however again based on the information about object ids, attributes and spatial properties given in the dataset. We believe that our approach is more general and applicable to other datasets. As seen in Table 2, our model outperforms most of the previous methods and its performance remains close to state-of-the-art, yet it learns visual features from raw images with end-to-end training. We believe that with richer and larger datasets, our model would generalize better than it currently does on the relatively small sized dataset such as SAIL.

Table 2: Comparison of our model to state-of-the-art.

Method	Success rate (%)
Chen & Mooney (2011)	54.40
Chen (2012)	57.28
Kim & Mooney (2012)	57.22
Kim & Mooney (2013)	62.81
Artzi et al. (2013)	65.28
Artzi et al. (2014)	64.36
Andreas and Klein (2015)	59.6
Kočiský (2016)	63.25
Mei et al. (2016) (ens=10)	71.05
Fried et al. (2017) (ens=10)	71.64
Can & Yuret (2018) (ens=10)	**72.82**
Our Model	65.45

4 RELATED WORK

One possible way to address the problem of navigational instruction following is to use reinforcement learning. Vogel and Jurafsky [28] for example use divergence from the gold path as reward signal. Their model was able to ground meaning of some spatial words. For the SAIL dataset however, since the gold action at each time step is known and annotated for the whole sequence, a supervised learning approach becomes possible as we use in this work.

Parser based methods have also been suggested for navigational instruction following task. Chen and Mooney [8] translate instructions into formal executable plans with the KRISP semantic parser [14]. They train the parser with aligned instruction and action-sequence pairs. The system is later improved by Chen [7] by modifying the underlying semantic grammar. Kim and Mooney [16] use probabilistic context free grammar (PCFG) induction [5] for this task. They improve this technique later by using a re-ranking module [15]. Artzi and Zettlemoyer [3] use combinatory categorical grammar (CCG) parser to translate instructions into a lambda-calculus formalism. They improve their system using a re-ranker as well [2].

Another approach is to learn the mapping from instructions into action sequences in an end-to-end fashion. Andreas and Klein [1] follow this approach by scoring multiple execution plans based on the alignment with the given instruction. They use a conditional random field model to learn this alignment. Mei et al. use an encoder-decoder network. Their model also includes a textual attention module [21]. Kočiský et al. [18] take a semi-supervised approach and use randomly generated action sequences for training. They show that the model benefits from unsupervised training.

In 2017, Fried at al. [9] have proposed a Speaker-Listener model which interprets the instruction and simulates possible alternatives for each instruction. Their speaker model generates more robust and less ambiguous instructions by reasoning about the listener model. They report state-of-the-art results on SAIL dataset. Can and Yuret [6] analyze the statistics of the SAIL dataset. They show that some instructions occur too few times that makes it impossible for the model to learn. They also show that their model learns more efficiently by artificially boosting the training data. They report state-of-the-art results on SAIL using CNN over the grid based representation of the world.

Unlike the previous studies that use feature vector representations for the world states, our system processes raw visual input. The same approach is used in a very recent work in the navigation domain. Based on their previous work [9], Fried et al. [10] present a novel speaker-follower model. With this method, they use a pre-trained speaker model that can generate novel instructions based on the route. The human annotated dataset is augmented by using this speaker model. At test time, the follower model suggests routes to follow and these are ranked by the speaker. This allows the follower to choose the best trajectory to follow and generalize better for the unseen instructions. This work does not report results on SAIL dataset and therefore not included in the Table 2.

5 CONCLUSION

We have reconstructed SAIL maps in 3D for data generation and collection. This has allowed us to collect visual data based on the SAIL maps. By using these collected images and the original SAIL dataset, we have proposed a deep learning model that learns to follow navigational instructions with visual grounding. The architecture can be trained and used with real images as well instead of artificial images without significant architectural changes. Our experiments have shown that our model outperforms most of the previous methods while remaining close to state-of-the-art in terms of success rate.

As future work, one possible way to move forward is to train this model with images captured from real world to demonstrate its potential for generalization. The system can be trained in an end-to-end fashion with real world data by keeping its architecture intact. The only limitation could be the use of discrete actions which could be hard to achieve in a real world scenario. However, if this condition is satisfied, the model can learn to follow the action sequence. There is no limitation on the language side. It is also possible to use transfer learning when initializing the model. The system can be trained with the data from the virtual world first and then can be fine-tuned with the real images. Moving from a simulation environment to the real world has already been shown to be a challenging but useful approach [26].

ACKNOWLEDGMENTS

This work was supported by the Scientific and Technological Research Council of Turkey (TUBITAK) grant 114E628.

REFERENCES

[1] Jacob Andreas and Dan Klein. 2015. Alignment-based compositional semantics for instruction following. *arXiv preprint arXiv:1508.06491* (2015).
[2] Yoav Artzi, Dipanjan Das, and Slav Petrov. 2014. Learning compact lexicons for CCG semantic parsing. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. 1273–1283.
[3] Yoav Artzi and Luke Zettlemoyer. 2013. Weakly supervised learning of semantic parsers for mapping instructions to actions. *Transactions of the Association of Computational Linguistics* 1 (2013), 49–62.
[4] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2014. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473* (2014).
[5] Benjamin Börschinger, Bevan K Jones, and Mark Johnson. 2011. Reducing grounded learning tasks to grammatical inference. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, 1416–1425.

[6] Ozan Arkan Can and Deniz Yuret. 2018. A new dataset and model for learning to understand navigational instructions. *arXiv preprint arXiv:1805.07952* (2018).

[7] David L Chen. 2012. Fast online lexicon learning for grounded language acquisition. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics: Long Papers-Volume 1*. Association for Computational Linguistics, 430–439.

[8] David L Chen and Raymond J Mooney. 2011. Learning to interpret natural language navigation instructions from observations.. In *AAAI*, Vol. 2. 1–2.

[9] Daniel Fried, Jacob Andreas, and Dan Klein. 2017. Unified Pragmatic Models for Generating and Following Instructions. *arXiv preprint arXiv:1711.04987* (2017).

[10] Daniel Fried, Ronghang Hu, Volkan Cirik, Anna Rohrbach, Jacob Andreas, Louis-Philippe Morency, Taylor Berg-Kirkpatrick, Kate Saenko, Dan Klein, and Trevor Darrell. 2018. Speaker-Follower Models for Vision-and-Language Navigation. *arXiv preprint arXiv:1806.02724* (2018).

[11] Alex Graves and Jürgen Schmidhuber. 2005. Framewise phoneme classification with bidirectional LSTM and other neural network architectures. *Neural Networks* 18, 5-6 (2005), 602–610.

[12] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 770–778.

[13] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation* 9, 8 (1997), 1735–1780.

[14] Rohit J Kate and Raymond J Mooney. 2006. Using string-kernels for learning semantic parsers. In *Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the Association for Computational Linguistics*. Association for Computational Linguistics, 913–920.

[15] Joohyun Kim and Raymond Mooney. 2013. Adapting discriminative reranking to grounded language learning. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Vol. 1. 218–227.

[16] Joohyun Kim and Raymond J Mooney. 2012. Unsupervised pcfg induction for grounded language learning with highly ambiguous supervision. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*. Association for Computational Linguistics, 433–444.

[17] Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

[18] Tomáš Kočiskỳ, Gábor Melis, Edward Grefenstette, Chris Dyer, Wang Ling, Phil Blunsom, and Karl Moritz Hermann. 2016. Semantic parsing with semi-supervised sequential autoencoders. *arXiv preprint arXiv:1609.09315* (2016).

[19] Minh-Thang Luong, Hieu Pham, and Christopher D Manning. 2015. Effective approaches to attention-based neural machine translation. *arXiv preprint arXiv:1508.04025* (2015).

[20] Matt MacMahon, Brian Stankiewicz, and Benjamin Kuipers. 2006. Walk the talk: Connecting language, knowledge, and action in route instructions. *Def* 2, 6 (2006), 4.

[21] Hongyuan Mei, Mohit Bansal, and Matthew R Walter. 2016. Listen, Attend, and Walk: Neural Mapping of Navigational Instructions to Action Sequences.. In *AAAI*, Vol. 1. 2.

[22] Tomáš Mikolov, Martin Karafiát, Lukáš Burget, Jan Černockỳ, and Sanjeev Khudanpur. 2010. Recurrent neural network based language model. In *Eleventh Annual Conference of the International Speech Communication Association*.

[23] Vinod Nair and Geoffrey E Hinton. 2010. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*. 807–814.

[24] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. 2015. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision* 115, 3 (2015), 211–252.

[25] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. 2014. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*. 3104–3112.

[26] Josh Tobin, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter Abbeel. 2017. Domain randomization for transferring deep neural networks from simulation to the real world. In *Intelligent Robots and Systems (IROS), 2017 IEEE/RSJ International Conference on*. IEEE, 23–30.

[27] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems*. 5998–6008.

[28] Adam Vogel and Dan Jurafsky. 2010. Learning to follow navigational directions. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics, 806–814.

[29] Paul J Werbos. 1990. Backpropagation through time: what it does and how to do it. *Proc. IEEE* 78, 10 (1990), 1550–1560.

[30] Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhudinov, Rich Zemel, and Yoshua Bengio. 2015. Show, attend and tell: Neural image caption generation with visual attention. In *International conference on machine learning*. 2048–2057.

[31] Deniz Yuret. 2016. Knet: beginning deep learning with 100 lines of julia. In *Machine Learning Systems Workshop at NIPS*, Vol. 2016. 5.