CS229 Final Project: Language Grounding in Minecraft with Gated-Attention Networks

Ali Zaidi
Microsoft, Stanford University
alikazim@stanford.edu

Abstract

A key question in language understanding is the problem of language grounding – how do symbols such as words get their meaning? We examine this question in the context of task oriented language grounding in gameplay. In order to perform tasks and challenges specified by natural language instructions, agents need to extract semantically meaningful representations of language and map it to the visual elements of their scene and into actions in the environment. This is often referred to as task-oriented language grounding. In this project, we propose to directly map raw visual observations and text input into actions for instruction execution, using an end-to-end trainable neural architecture. The model synthesizes image and text representations using Gated-Attention mechanisms and learns a policy using Stein Variational policy gradients to execute the natural language instruction. We evaluate our method in the Minecraft environment to the problem of retrieving items in rooms and mazes and show improvements over supervised and common reinforcement learning algorithms.

1 Introduction

Reinforcement learning has made incredible progress in the past few years, in large parts due to the effectiveness of neural networks as value function approximators, and their ability to be trained end-to-end using stochastic optimization. In this work, we examine the ability to define differentiable architectures that can be trained end-to-end using stochastic optimization for solving task-oriented language grounding problems. These are problems where the agent needs to solve specific tasks defined through natural language instructions. In order to succeed in such applications, the agent needs to extract semantically meaningful representations of language that they can map to visual elements and actions in the environment. In particular, to accomplish goals defined through instructions, the agent needs to draw semantic correspondences between the visual and linguistic modalities and learn a policy to perform the required task, an example of multi-modal learning.

To tackle this problem, we propose an architecture that consists of a *state representation module* that provides a joint representation of the natural language instruction and the visual scene observed by the agent's view, and a *policy learning module* to predict the optimal action for the agent in the next timestep t. This state representation module is defined using a Gated-Attention network [CSP$^+$17], and the policy learning module is created using Stein Variational policy gradient. The main aspects of this project can be summarized as 1) train end-to-end neural architectures that take raw pixel input of the current scene and natural language instructions without assuming any intermediate representations or prior knowledge 2) representation learning using Gated-Attention mechanisms for multimodal fusion of linguistic and visual modalities 3) policy learning using Stein variational policy gradients. All of the experiments in this project are conducted using the popular Minecraft game with the open source framework Project Malmo [JHHB16]. Figure 1 provides an example of task-oriented gameplay in this environment.

2 Related Work

Learning semantic grounding is a challenging task, especially without prior knowledge. From a statistical perspective, the agent needs to learn correlations in a stream of low-level inputs, (pixel-level visual input, in this scenario), and map those correlations to linguistic

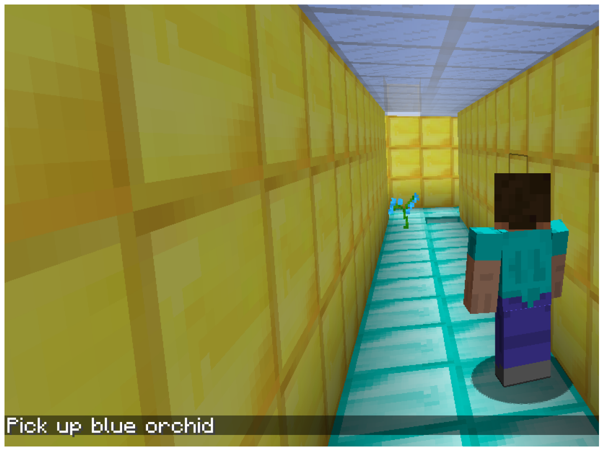

Figure 1: Task-oriented grounding in a maze environment. Here the agent is provided with natural language instructions to pick up a certain item but also needs contextual visual information to avoid the dangerous missing block.

expressions and feasible actions in their environment. Symbolic approaches to language grounding have been attempted for nearly half a century, [Win72]. However, such rule-based approaches are unable to scale to the full complexities of natural language and stochastic three-dimensional environments. Recent research [AAD+16, CSP+17, HHG+17, LHKOR17] has aimed to fuse and learn directly from these different modalities: vision, language, and action. Directly learning these concept representation spaces directly has been shown to provide more faithful human semantic representations of properties of text and objects [SL12].

3 The 3D Minecraft Learning Environment

3.1 Customizable Platform for AI Experimentation

Our environment is based on the customizable platform called Project Malmo [JHHB16], which provides an environment for experimentation with autonomous agents and AI agents in Minecraft. Minecraft is an enormously popular game, where players can control agents in the 3D grid world and construct novel objects and architectures. It's rich and highly versatile structure makes it ripe for AI experimentation. Malmo enables experimentations through the use of "Missions", which defines the parameters of the environment, including the size of the environment, the number of creatures in the environment, duration of each mission, as well as the weather and lighting within the environment.

3.2 State, Action and Policy

In our missions, we spawn a single agent inside of a maze, which can be of modifiable length and size for a series of episodes. Each episode is defined as a single task, where the task is specified through natural language instruction I. For each set of experiments, the agent is spawned at a fixed coordinate at the start of each episode, and every experiment consists of 100 episodes (each episode can be thought of as a single run of a given mission, as described in section 3.1). Let's denote by s_t the state at each time step t in an episode, and M_t for the first person view of the agent at time t. Therefore, the agent's state at time t is given by the tuple $s_t = \{I, M_t\}$. Observe that the instructions are not indexed by time, and are therefore constant throughout a given an episode.

The agent's goal is learn an optimal policy $\pi(a_t|s_t)$, mapping the observed states into optimal actions for the given task. Concretely, the agent's goal is to learn a policy π that specifies a distribution over sequences of actions a_t in states s_t such that it maximizes it's utility function:

$$J(\pi) = \mathbb{E}_{s_0,a_0,\ldots}\left[\sum_{t=0}^{\infty}\gamma^t r(s_t, a_t)\right]. \tag{1}$$

For our agent, we set the discount factor to $\gamma = 0.01$, which encourages the agent to find faster routes to the target object. In this project, we considered stochastic policies for π, so that it defines a distribution of actions given the current state s_t, and we utilize a model-free setting. Under this specification, our state value function is provided by

$$V^{\pi}(s_t) = \mathbb{E}_{a_t,s_{t+1},\ldots}\left[\sum_{i=1}^{\infty}\gamma^i(s_{t+i}, a_{t+i})\right],$$

which is precisely the expected return from policy π from state s_t. Moreover, the state-action function, which defines the expected return from policy π after taking action a_t in s_t is given by:

$$Q^{\pi}(s_t, a_t) = \mathbb{E}_{s_{t+1},a_{t+1},\ldots}\left[\sum_{i=0}^{\infty}\gamma^i r(s_{t+i}, a_{t+i})\right].$$

4 Network Architecture

We use a single end-to-end architecture that combines the two sources of information, [CSP+17]. These two information sources are divided into two modules, a state processing module, which we describe next, and a policy learning module, which we describe in the sequel.

4.1 Convolutional Image Representations

As defined above, our states are defined by $s_t = \{I, M_t\}$ where M_t is the first-person view of the environment at iteration t. Each frame is fed through a convolutional neural network to create a convolutional representation of the state, $x_M = f(M_t, \theta_{\text{conv}}) \in \mathbb{R}^{d\times H\times W}$, where d denotes the number of feature maps, i..e, the intermediate representations in the convolutional network. This is shown in the upper module of Figure 2.

4.2 Instruction Embeddings

Our natural language instructions, I, are processed using a Gated-Recurrent Unit (GRU) [CvG+14, Ola15], $x_I = f(I; \theta_{\text{GRU}})$, which we call **instruction embeddings**. This is shown using the bottom module of the network architecture in Figure 2. Prior to "fusing" with the visual feature maps, the instruction embeddings are transformed through an attention vector, a_I, which we describe further below. The attention vector serves as a gating mechanism to magnify or diminish attention spent on a given set of features from the visual feature maps, corresponding to how they relate to the instructions. This is the key component of this architecture, so we describe it more fully in the next section.

4.3 Multimodal Fusion – Hadamard Products

Our two networks described above featurize the two modalities into representations we can use for training our policy. However, we need to find an effective way to combine these features into a single representation. A straightforward, if somewhat naive, approach would be to concatenate the features from both sources into a single source by flattening their representations into a long dense vector:

$$M_{\text{concat}}(x_I, x_M) = [\text{flatten}(x_I); \text{flatten}(x_M)].$$

However, in our case the representations from the visual input is a function of time, whereas the language input is not, which means the first vector in M_{concat} is constant throughout the episode.

Instead of concatenation, we take insight from the statistical modeling perspective and aim for some form of "interaction" between our two feature maps. In particular, the instruction embeddings are first passed through a fully-connected layer with a sigmoid activation function to create an attention vector, $a_I = h(x_I) \in \mathbb{R}^d$. Furthermore, each element of the attention vector is expanded into a $H \times W$ matrix to equal the dimensions of our state-image x_M, which is then multiplied element-wise with the output of the CNN (i.e., we use the Hadamard product of the two embeddings):

$$M_{\text{GA}}(x_I, x_M) = M(h(x_I)) \odot x_M = M(a_I) \odot x_M.$$

Since x_M is a function of time, our interaction of these two featurizations will also be a function of time. As mentioned above, since the instruction embeddings are first sent through our attention gate, only certain parts of the instruction embeddings are interacted with the visual feature maps. During learning, the parameters of the attention gate focus the agent's attention to those components of the visual embeddings that relate to the instruction embeddings.

4.4 Policy Learning Module

The multimodal fusion unit, M_{GA} is provided to the policy learning module, which we train using reinforcement learning techniques. In particular, we utilize policy gradient methods which aim to iteratively improve the policy to optimize the utility function $J(\theta)$ (1) by using estimates of it's gradient with respect to θ.

4.4.1 Likelihood ratio policy gradient: REINFORCE

As a baseline, we can approximate the gradient of (1) using the standard REINFORCE algorithm [Wil92]. REINFORCE utilizes rolled-out samples generated by $\pi(a|s, \theta)$ using a likelihood ratio approximation. Concretely, REINFORCE uses the following approximation of the policy gradient:

$$\begin{aligned}\nabla_\theta J(\theta) &\approx \sum_{t=0}^{\infty} \nabla_\theta \log \pi(a_t|s_t; \theta) R_t \\ R_t &= \sum_{i=1}^{\infty} \gamma^i (s_{t+i}, a_{t+i}).\end{aligned}$$

REINFORCE is a provably unbiased estimator for the $\nabla_\theta J(\theta)$, but can suffer from reliable estimates in small samples due to the high variance in gradient estimates.

As an alternative to our baseline, we consider variational methods for policy optimization. Rather than optimizing for a single policy, we utilize Stein variational policy gradient [LW16] which searches for a distribution $q(\theta)$ to optimize the expected return using variational methods. Concretely, we formulate our variational optimization objective as

$$\max_q \{\mathbb{E}_q [J(\theta)] + \alpha \boldsymbol{H}(q)\}$$

where the second term provides entropy of the variational distribution q, and α is a "temperature" parameter that controls the rate of exploration. It's optimal

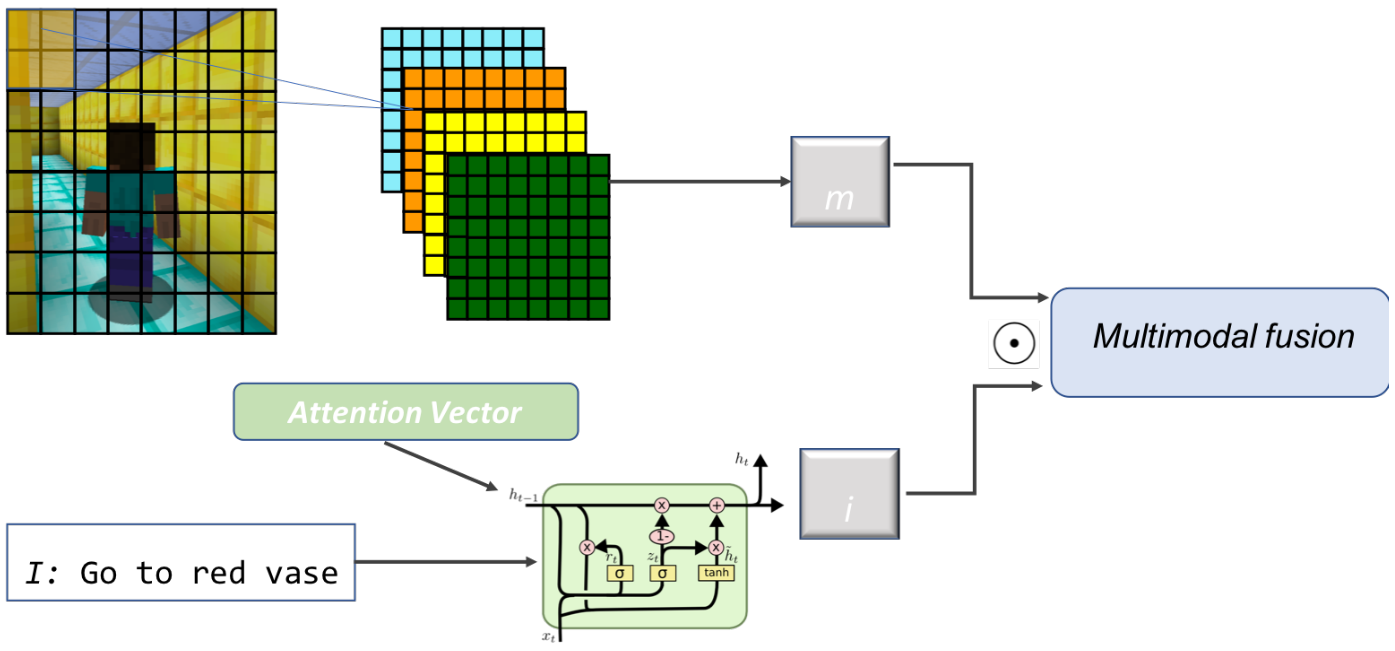

Figure 2: Network Architecture

value is given by

$$q(\theta) \propto \exp\left(\frac{1}{\alpha} J(\theta)\right) q_0(\theta). \qquad (2)$$

This variational objective has the advantage of acting as a regularizer/control variate, and a fast particle based simulation mechanism. We can also see this as a Bayesian formulation of our estimate θ, where $\exp(J(\theta)/\alpha)$ serves as our likelihood and $q(\theta)$ as our prior. Stein variational gradient descent [LW16] proceeds by proposing particles according to

$$\theta_i \leftarrow \theta_i + \epsilon \phi^\star(\theta_i),$$

where ϕ is a suitably chosen kernel selected to decrease the KL divergence between θ_i and the target distribution $J(\theta)$. In particular, the authors showed that a closed form solution can be obtained for this optimization, which has the form

$$\begin{aligned} \phi^\star(\theta) &= \mathbb{E}_{\mu\sim\eta}\left[\nabla \log q(\mu) k(\mu,\theta) + \nabla_\mu k(\mu,\theta)\right] \\ \hat{\phi}(\theta_i) &= \frac{1}{n}\sum_{j=1}^{n}\left[\nabla_\theta \log q(\theta_j) k(\theta_j,\theta_i) + \nabla k(\theta_j,\theta_i)\right], \end{aligned}$$

where μ is the proposal distribution for the particles. The first term $\nabla_\theta \log q(\theta)$ provides the momentum, i.e., it pushes particles towards high probability regions of q by the mean field of the particles (i.e., information sharing across particles, [Mor13]), and the second term is a diversity / potential energy effect, pushing particles away from each other. Applying SVGD to our optimization (2) we obtain the updates

$$\hat{\phi}(\theta_i) = \frac{1}{n}\left[\nabla_{\theta_j}\left(\frac{1}{\alpha}J(\theta_j) + \log q_0(\theta_j)\right) k(\theta_j,\theta_i) + \nabla_{\theta_j} k(\theta_j,\theta_i)\right].$$

5 Experiments and Results

For experiments, we provide our agent with instructions to pick up specific items in a maze, with penalty-states that terminate the agent's life (i.e., falling down a a hole, or picking the wrong item). We examined our architecture at varying distances and with varying number of items, gradually increasing both the distance and the number of items, but keeping the episode/mission length fixed. The reward function is

$$R(x,w) = 100 \cdot x \cdot w,$$

where x denotes the amount of paces between the target item and the agent, and w denotes the number of items. Therefore, when there's one item, and it's 10 steps away, the agent achieves a reward of 10 when it finds the item, and 0 otherwise. By increasing both x and w in $R(x,w)$, we develop a aa simple learning plan / curriculum of more challenging tasks. Note, however, that this is not a proper curriculum, and there are no intermediate rewards given to the agent if it does not find the correct object.

While implementing the Stein variational gradient descent for our policy gradients, we used a Gaussian RBF kernel

$$k(\mu,\theta) = \exp\left(-\|\theta-\mu\|_2^2/h\right),$$

	Zero-shot learning	
iterations	**REINFORCE**	**SVPD**
10^3	0.23	0.32
10^4	0.27	0.45
10^5	0.33	0.61

Table 1: Zero-shot learning

where the bandwidth is chosen to to be median of the pairwise distance between the particles, as in the original paper [LW16]. We initialize our policy value using a Gaussian policy with a diagonal covariance structure, and estimate the mean structure is parameterized by a three-hidden layer (100-75-50 hidden units) neural network with tanh activation function. The results in Figure 3 show that our Stein variational method performs significantly better than the REINFORCE method. In particular, the Stein variational method exhibits lower variance, due to the kernelized "control-variate" in it's iterations, and is also able to learn faster, most likely due to the improved diversity of it's policies.

5.1 Evaluation

For evaluation, we examined zero-shot task generalization, which is the ability of our model to generalize to new combinations of object-attribute pairs that were not used during training. More precisely, we used the following list of colors and items during training $\{\text{colors}, \text{objects}\} = \{[\texttt{green}, \texttt{blue}, \texttt{gold}, \texttt{black}], [\texttt{diamond}, \texttt{brick}, \texttt{sword}, \texttt{tulip}]\}$. Not every color-item is valid, i.e., `diamond` only appears in blue. However, for brick, tulip and sword, there are many color combinations, and during training we drop one of their colors that are used by at least one other object, i.e., `blue-sword`. Zero-shot learning is successful (with reward 1) if the agent generalizes and can complete the unseen instruction (in this case, "`go pick up the blue sword`"). For each evaluation episode, there are 4 other "occluding" objects, that either have the same color or the same item description as the target object. Hence, 0.2 average reward would be attained simply by chance. Our average results over 100 evaluation episodes are shown in the table 1. Our results show the policy module learned using Stein variational policy gradient can generalize much better on out-of-sample examples than the traditional REINFORCE method.

6 Conclusion / Future Work

The experiments above showed that Stein Variational Policy Gradients consistently learn faster and with greater ability to generalize than the traditional likelihood ratio policy gradient estimator, REINFORCE. However, it would be useful to investigate whether the increased ability to generalize is simply due to the algorithm being able to train faster due to it's fast variational particle-based optimization methods, or because of the diversity factor embedded in the learning objective. For further experiments, I hope to examine additional control-variate methods for policy gradients, such as the asynchronous advantage actor critic method (A3C), [MBM+16], and trust region policy optimization (TRPO), [SLA+15], and see how they perform relative to Stein Variational Policy Gradient.

From a language grounding perspective, it would be interesting to examine higher order referents and relational concepts, such as representations of size ("`pick up the larger object`"), or hue ("`pick up the object with brighter shade`"). Being able to see how the agent would generalize with such relational concepts would shed a lot of light on relational grounding for language. I hope to tackle these experiments next!

References

[AAD+16] David Abel, Alekh Agarwal, Fernando Diaz, Akshay Krishnamurthy, and Robert Schapire, *Exploratory gradient boosting for reinforcement learning in complex domains.*

[CSP+17] Devendra Singh Chaplot, Kanthashree Mysore Sathyendra, Rama Kumar Pasumarthi, Dheeraj Rajagopal, and Ruslan Salakhutdinov, *Gated-attention architectures for task-oriented language grounding*, CoRR **abs/1706.07230** (2017).

[CvG+14] K. Cho, B. van Merrienboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, *Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation*, ArXiv e-prints (2014).

[HHG+17] Karl Moritz Hermann, Felix Hill, Simon Green, Fumin Wang, Ryan Faulkner, Hubert Soyer, David Szepesvari, Wojciech Marian Czarnecki, Max Jaderberg, Denis Teplyashin, Marcus Wainwright, Chris Apps, Demis Hassabis, and Phil Blunsom, *Grounded language learning in a simulated 3d world*, CoRR **abs/1706.06551** (2017).

[JHHB16] Matthew Johnson, Katja Hofmann, Tim Hutton, and David Bignell, *The malmo*

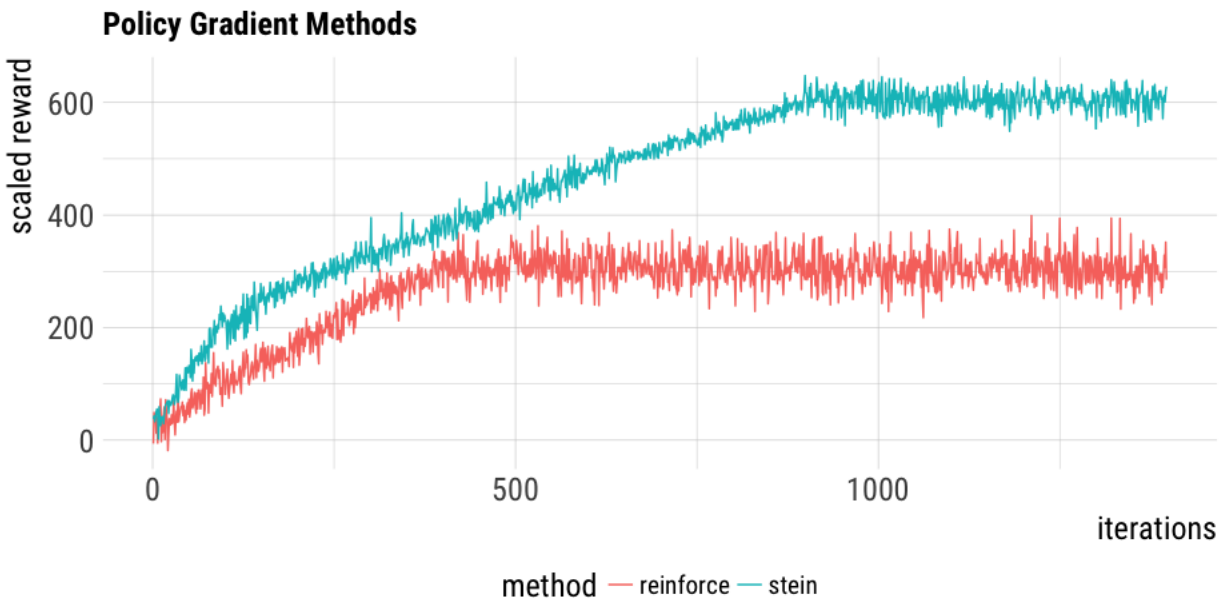

Figure 3: Learning Curves for SVPG and REINFORCE

platform for artificial intelligence experimentation, Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, IJCAI 2016, New York, NY, USA, 9-15 July 2016, 2016, pp. 4246–4247.

[LHKOR17] Zhiyu Lin, Brent Harrison, Aaron Keech, and Mark O. Riedl, *Explore, exploit or listen: Combining human feedback and policy model to speed up deep reinforcement learning in 3d worlds*.

[LW16] Qiang Liu and Dilin Wang, *Stein variational gradient descent: A general purpose bayesian inference algorithm*, Advances in Neural Information Processing Systems 29 (D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, eds.), Curran Associates, Inc., 2016, pp. 2378–2386.

[MBM$^+$16] Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu, *Asynchronous methods for deep reinforcement learning*, CoRR **abs/1602.01783** (2016).

[Mor13] P.D. Moral, *Mean field simulation for monte carlo integration*, Chapman & Hall/CRC Monographs on Statistics & Applied Probability, CRC Press, 2013.

[Ola15] Chris Olah, *Understanding lstm networks*, `http://colah.github.io/posts/2015-08-Understanding-LSTMs/`, 2015.

[SL12] Carina Silberer and Mirella Lapata, *Grounded models of semantic representation*, Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (Stroudsburg, PA, USA), EMNLP-CoNLL '12, Association for Computational Linguistics, 2012, pp. 1423–1433.

[SLA$^+$15] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz, *Trust region policy optimization*, Proceedings of the 32nd International Conference on Machine Learning (ICML-15) (David Blei and Francis Bach, eds.), JMLR Workshop and Conference Proceedings, 2015, pp. 1889–1897.

[Wil92] Ronald J. Williams, *Simple statistical gradient-following algorithms for connectionist reinforcement learning*, Mach. Learn. **8** (1992), no. 3-4, 229–256.

[Win72] T. Winograd, *Understanding natural language*, Academic Press, 1972.