Deborah Häuser & Caroline Kellner

Reimplementing and extending a multi-agent language emergence model with empathy

Final Report for the course Deep Reinforcement Learning - WS 2020/2021 - Universität Osnabrück

Introduction

Within this report we present a reimplementation of the paper “Enhancing Language Emergence Through Empathy” by Marie Ossenkopf which was under review as a conference paper at ICLR in 2020 [1]. We added a functionality to the implementation of Ossenkopf which we will present and explain what differences this change makes to the results.

Ossenkopf’s paper implemented a multi-agent system that uses Deep Reinforcement Learning for developing a language to cooperatively achieve a shared goal.

In the following, we will shortly present the basic ideas of Reinforcement Learning, Deep Reinforcement Learning, and Language Emergence in Deep Reinforcement Learning. Subsequently, before we present Ossenkopf’s paper in more detail, we present two other papers that hers is built upon. Finally, we will explain our implementation as well as the added functionality to lastly discuss our results.

Background

Reinforcement Learning

Basic Reinforcement Learning can be modeled as a Markov Decision Process in which an Agent acts inside of an environment. The agent is at a certain state at every time from which a set of actions is possible. The actions are chosen via a policy. Based on the policy, as well as the transition probability function, the agent transitions from the current state to a subsequent state. This transition yields a specific reward. The goal of the agent is set in the objective functions and includes maximizing the discounted sum of immediate and future rewards. In traditional reinforcement learning settings, the policy is based on state- values or state-action values [2].

Deep Reinforcement Learning

In Deep Reinforcement Learning neural networks are used for either calculating the state-action-values for each state-action combination or to directly represent the policy. The latter is of relevance for this report. We will work with the REINFORCE algorithm [3] for discrete action spaces in which a neural network is used to output the probabilities for each action [4].

Language Emergence in Deep Reinforcement Learning

In the field of Artificial Intelligence, language emergence deals with the development of communication through language in artificial systems and furthermore analyses its emergent properties.

Reinforcement learning is a suitable approach because it has properties that are also relevant in human language learning. A typical Reinforcement Learning setting in the field of Language Emergence is a multi-agent system that uses language to cooperatively achieve a goal. The human characteristics of language emergence that can also be observed here are, on the one hand, that language is created through interaction and, on the other hand, that language has a goal. These functional and interactive aspects provide a major advantage of the reinforcement learning setting compared to other approaches such as supervised learning [5][6].

LSTM

Recurrent neural networks have, in contrast to feed forward neural networks, loops in them which allow for the persistence of information over a period of steps. Thereby, RNNs can process sequential data such as language.

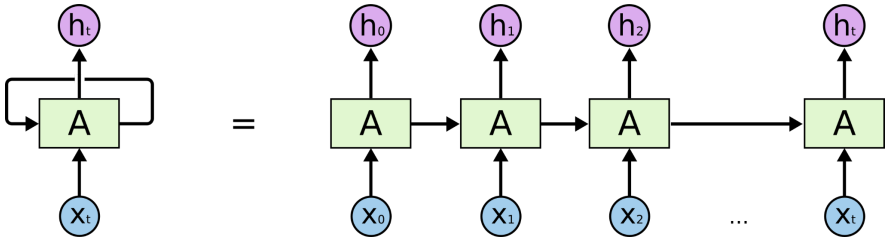

Figure 1: Illustration of Recurrent Neural Network [7]

However, a basic RNN cannot properly deal with long-term dependencies because it suffers from vanishing gradients. The LSTM, which is a special kind of an RNN, tackles this problem. It is built up by four neuronal layers which determine in a complex interaction which information is passed on to the next loop. This makes them especially suitable for applications where long-term memory is essential, like language acquisition and understanding.

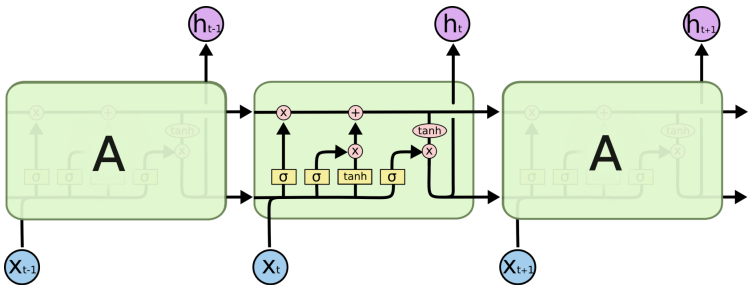

Figure 2: Illustration of an LSTM cell [7]

A more in-depth explanation of LSTMs is provided by Christopher Olah and can be found in his blog [7].

Background Literature

In this section, we first outline two papers that Ossenkopf's paper is based upon before we present Ossenkopf's paper and discuss differences between them. All three papers explore language emergence in a variation of the Lewis' signaling game which they implement via deep reinforcement learning.
In the Lewis Signaling game, there exist two players, a "sender" and a "receiver". The sender has knowledge about the state of the world and can send a message to the receiver from a fixed set of signals. Based on the signal and without knowledge about the state of the world, the receiver chooses a certain action. For each state of the world, there exists one action which is the correct one. Both sender and receiver have the common goal that the receiver chooses this correct action [8].

The paper "Multi-Agent Cooperation And The Emergence of (Natural) Language" [6] was published in 2017 by Lazaridou et. al. In the paper the sender and the receiver are represented as neural networks. Both networks received as input two pictures which are sampled from ImageNet [9]. One of the pictures is declared to be the target and the other picture is declared to be the distractor. The sender is informed about the respective allocation. By choosing a symbol from a fixed and arbitrary alphabet the sender signals to the receiver which of the pictures is the target. The sender in turn uses this information to guess the target.

The paper focuses on the question of whether successful communication occurs and whether the emerged language is comparable to human language. The paper compared two different setups for the sender, the "agnostic" and the "informed" sender. In this paper, we will solely focus on the agnostic sender as the informed sender is not relevant for our own implementation. The sender and the receiver were both implemented as one-layer feed-forward networks. The sender embeds the two images into its game-specific embeddings followed by a sigmoid activation function which represents the scores for the symbols of the alphabet. Those activations are in turn converted into a Gibb's distribution from which a symbol is sampled.

The receiver embeds the two images as well as the received message into its game-specific embedding and calculates the dot product between the image embeddings and the message embedding. Those two dot products are converted into a Gibb's distribution from which the target guess is sampled. The two networks do not share any weights.

The objective function of the two agents is to maximize the expected reward which is 1 if the receiver guesses the target correctly and 0 otherwise. The policy of the sender therefore chooses which symbol to communicate to the receiver, that maximises the probability that the receiver will in turn choose the correct target. The policy of the receiver is responsible for choosing the image that best fits the message the receiver has got. Both policies are classically learned by updating the neural networks via backpropagation.

The weights of the networks are learned while playing the signaling game and are updated via the Reinforce Update Rule. A derivation and explanation of the Reinforce

Update Rule can be found in the book “Reinforcement Learning: An Introduction” by Sutton and Barto [10]**.** In the paper, Lazaridou et al. show that the agents successfully learned to communicate, and furthermore, that the emerged communication involved semantic properties to some extent.

In 2018 Lazaridou et al. published another paper called “Emergence of Linguistic Communication From Referential Games With Symbolic And Pixel Input” [11]. This paper builds upon the previous one and compares the effect of different input data on language emergence. They used two inputs that differ in their degree of structure. As an example for very structured data, the inputs were binary vector representations of words which indicate whether a certain feature applies to the respective word or not. The dataset is the VisA Dataset of Silberer et. al. [12]. As an example for unstructured data, they used raw pixel data which was generated using the MuJoCo physics engine [13].

The goal of the paper is to examine whether the compositionality of language occurs even if the input data is entangled and unstructured as is the case with the raw pixel data. This increased noisiness would make the setup of the signaling game more comparable to human environments.

In the following, the setup of the game is explained using the vector input but it works analogously with the raw pixel data. The speaker receives as input the target vector and outputs a message sampled from a fixed and arbitrary vocabulary. The message can vary in length, but the maximum length is fixed. The receiver in turn gets as input the message as well as the target vector and a fixed number of distractor vectors. Those distractors are simply vectors representing words other than the target. With this information, the receiver must guess which of the vectors is the target vector.

The speaker encodes the input via a one-layer neural network into a dense representation. This encoding is forwarded through a one-layer LSTM which outputs a sequence from the vocabulary. This sequence is the message that is, besides the target and distractor vectors, the inputs for the receiver. The receiver encodes the target and distractor vectors via a one-layer neural network. For encoding the message a one-layer LSTM is used. Subsequently, the receiver calculates the dot product between the encodings of the vectors and the encoding of the message respectively. Those dot products are converted into a Gibbs distribution from which the target guess of the receiver is sampled.

The policy of the sender therefore chooses which message to communicate to the receiver, that maximises the probability that the receiver will in turn choose the correct target. The policy of the receiver is responsible for choosing the vector that best fits the message the receiver has got. Both policies are classically learned by updating the neural networks via backpropagation.The networks do not share any weights. The weights are updated via the REINFORCE update rule. In the objective function there is an entropy regularization term included to increase exploration.

The paper shows that successful communication occurs with both inputs. Nevertheless, agents perform better in outputting structured messages that include compositionality when the input is structured and disentangled.

In the following we present Ossenkopf's [1] paper which we tried to reimplement. The paper builds upon the previous papers and adds an unsupervised auxiliary task into their implementation. The task aims at representing an empathetic component of human language namely that speakers take into consideration how their utterances will be perceived by their listeners. The aim of the paper is to investigate whether such an auxiliary task improves communication in the signaling game.

The input data are the word vectors from the VisA dataset which are also used in the previous paper. Except for the empathy extension, the set up and implementation of the receiver and the sender are the same as in the previous paper and therefore will not be outlined here again.

The auxiliary task is implemented via a single-layer feed-forward neural network which receives the last hidden state of the speaker LSTM before the message is uttered as input. The network then tries to predict the hidden state of the listener after it receives the message, based on the hidden state of the speaker. The auxiliary network is updated via backpropagation and the loss is composed of the mean absolute difference between the predicted and the actual hidden state of the listener LSTM. These gradients are then also added to the gradients of the speaker's encoder and are used to update this network. This means that the encoder of the speaker is updated in regard to the performance in the auxiliary task performance as well as in regard to the listener's performance in guessing the correct target as it was the case in the previous papers as well. This concept is described as empathy and also highly corresponds to the so called Theory of Mind, because it seeks to align the encoding of the sender and the listener by building a model of the listener in the "mind" of the speaker.

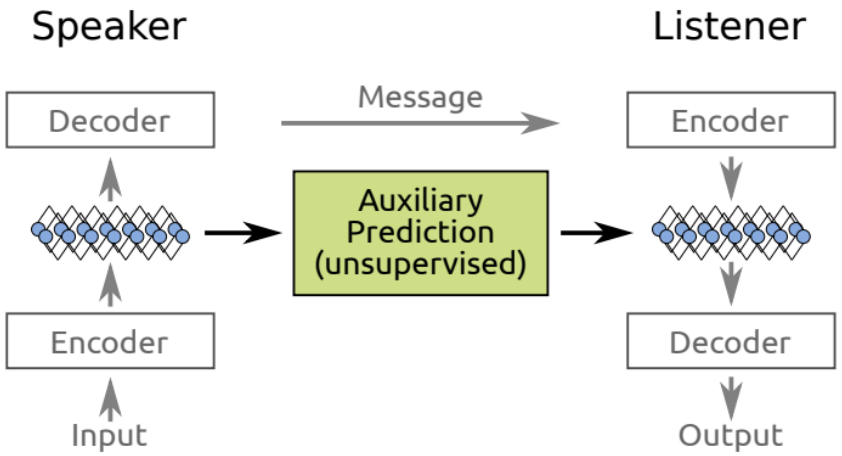

Figure 3: Architecture of Multi- Agent System with Auxiliary Task [14]

In the paper it is shown that through adding the auxiliary task the learning speed can be highly reduced to double or even triple.

The following table illustrates the main differences between the three paper's architectures in a compact form:

	Lazaridou 2017	**Lazaridou 2018**	**Ossenkopf 2020**
Dataset	Images (ImageNet)	Word vectors (visA Dataset) and raw pixel data (generated objects via MuJoCo physics engine)	Word vectors (visA Dataset)
Input of Sender	Target and one distractor	Target	Target
Input of Receiver	Target and one distractor	Target and multiple distractors	Target and multiple distractors
Outputted message	One symbol	Variable length of symbols	Variable length of symbols

Scientific contribution

The aim of this project is to reimplement and extend the model presented in Ossenkopf (2020). We pursue this goal, because we think that the implementations of the sender in Ossenkopf (2020) and Lazaridou (2018) lack an interesting property that the sender in Lazaridou (2017) possesses. This property is the ability of the sender to utter a message that is not only based on the target, but also on knowledge about the distractor concepts. Our project can therefore be seen as a combination of the three publications presented beforehand.

By implementing two versions of the sender, one that sees only the target and one that sees both the target and the distractors, our goal is to analyse the influence this differing knowledge has on the form and compositionality of the produced messages. The sender is only implicitly informed about the target. The target is always the first concept fed to the sender. For the receiver, the order is random.

Knowing both the target and the distractors reflects that human communication always happens in a specific context that has an influence on its form. We hypothesize that the sender that has more information, will send messages that focus on the differences between the target and the distractors.

Implementation

Our aim is to reimplement the presented Ossenkopf paper. As an extension to her setup we include the possibility that the sender receives as an input the target vector

as well as the distractor vectors that the receiver receives. Since we aim to reimplement Ossenkopf's paper, we adopt the parameters of her implementations as well as the neural network architectures.

We adopted the following parameters from Ossenkopf: learning rate, hidden size, embedding dimension, vocabulary size, entropy coefficients for sender and receiver. For the speaker's and receiver's encoder we use a single-layer feed-forward neural network with a specified hidden size and a sigmoid activation function.

Dataset and sampling from the dataset

We base our version of the Lewis' signaling game on the Visual Attributes for Concepts Dataset (VisA) of Silberer et al. [12]. This dataset contains more than 500 concrete concepts that are annotated with 595 different attributes (UK version). All concepts are described as bag-of-attributes, where attributes describe the properties that a specific concept has, e.g. a ball has the attribute 'is round', or a dolphin has the attribute 'can swim'. All attributes in this dataset are visually perceptible properties, this means that for example for an eggplant the attribute 'is purple' is valid, while 'is vegetable' is not, because the latter is only descriptive but not visually perceptible. The dataset and its representation as attribute vectors can be seen as disentangled and structured. The dimensions do not overlap and so every object is a unique conjunction of different properties, that can easily be accessed and – important for the task at hand – communicated efficiently.

In our implementation the concepts are presented as binary vectors that are stored in a dictionary with the respective natural language names as keys. Both the sender and the receiver receive these vectors as target and distractor input and transform them via their encoders into a denser representation.

The dataset and the sampling from the dataset are implemented in the scripts env2 and vocab. The Vocabulary class is used to read in the VisA dataset, as this dataset is distributed in xml-format. When saving the dataset as a Vocabulary all concepts and attributes are added as the more pythonic and iterable datastructures lists and dictionaries. After all concepts are added it is possible to iterate through them and store them as binary vectors of length num_attributes, where a '1' means that the concept has the attribute. The Vocabulary can then be given to the ConceptData class. This class makes it possible to sample from the Vocabulary. The sampling returns the inputs for both sender and receiver and the respective target for every batch.

Sender

As already mentioned the sender can be used in two different modalities. In one version the sender sees both the target vector and all distractors, in the other the sender sees only the target. Both implementations can be tested in the respectively named notebooks. Apart from this difference the senders work in the same way.

The sender first encodes the binary concept vectors into a denser representation. This embedding or hidden dimension size needs to be the same for all networks and is therefore set to 50 by a global variable. The dense representation is then converted

into a sequence by the first single-layer LSTM and then fed into the second single-layer LSTM. The output of this second LSTM is converted into a Gibbs distribution by a Dense layer with a softmax activation function. From this distribution the single symbols of the message are sampled.
The main difficulty when implementing the sender was to make the length of the message independent from the length of the input sequence. This is achieved using a loop that calls the second LSTM-layer for the desired maximum message length. The atomic symbols are sampled in every iteration of the loop.

After all messages are created, zeros are added to the end of the messages. This forces the networks to use “0” as the end-of-sequence symbol.

Receiver

First of all, the message is embedded via the tf.Keras.Embedding layer. This turns the message which consists of positive integers into dense vectors of the size of the embedding dimension.

The message that is outputted by the sender has a “0” as end-of-sequence symbol. Nevertheless, after the EOS symbol the message could have values other than zero. Therefore, it is important to padd and mask the sequence such that the symbols after the EOS symbol will be ignored. This happens in the class “Receiver_LSTM”. The adjusted message will then be sent through the LSTM of the Receiver. Which is a one-layer LSTM network.

The Receiver encoder consists of a one-layer neural network with Sigmoid activation function. In the class “Receiver” the different concepts are sent through the network. Afterwards in the method “receiver_sampling”, the dot-producs between the message encoding and the encodings of the candidates are calculated. Via a softmax layer, these dot-products are converted into a Gibb’s distribution from which the target guess is sampled and the log probability of the guess and the entropy of the distribution is calculated.

Game

The signaling game itself is implemented in the Game model. This model is responsible for calling the agents and distributing the inputs and the message. After the message is sent and the guesses of the receiver are known, the loss is computed via the following formula:

$$R(t')(\sum_{l=1}^{L} logp_{\pi_S}(m_t^l|m_t^{<l}, h_s) + logp_{\pi_L}(t'|h_L, C)) - (\alpha_S H_S + \alpha_L H_L)$$

where there is

- R(t’): the reward,
- π_S: the speaker’s policy,
- π_L: the listener’s policy,

- m: the outputted symbol at length l
- h_s: dense representation of the target vector which is provided by the speaker's encoder
- t': the target Object
- C: the set of candidate objects
- h_s: dense representation of the message which is provided by the listener's encoder
- $\boldsymbol{\alpha}_s$, $\boldsymbol{\alpha}_L$: the respective entropy regularization coefficients
- H_s, H_L the respective entropies of Listener and Speaker.

The game then returns the optimized loss and also the last hidden states of sender and receiver, as these are needed for the auxiliary network.

Auxiliary Network

The auxiliary network is a single-layer feed-forward neural network with a sigmoid activation function. It receives the last hidden state of the speaker LSTM as input and predicts the hidden state of the receiver LSTM, based on this. This way it implements something similar to empathy or a Theory of Mind, as it builds a model of the receiver in the "mind" of the sender.

Results

Unfortunately, we were not able to implement the signaling game in a fully functional manner. The gradients of the custom loss-function cannot be computed for the sender network, which makes it impossible to learn sending meaningful messages. The error that occurs can be recreated in the notebook "Game_Error_Recreation". However, even with this error it is still possible to let the game run for multiple epochs and train the parts of the networks where a gradient can be computed, namely the receiver-network and the auxiliary-network.

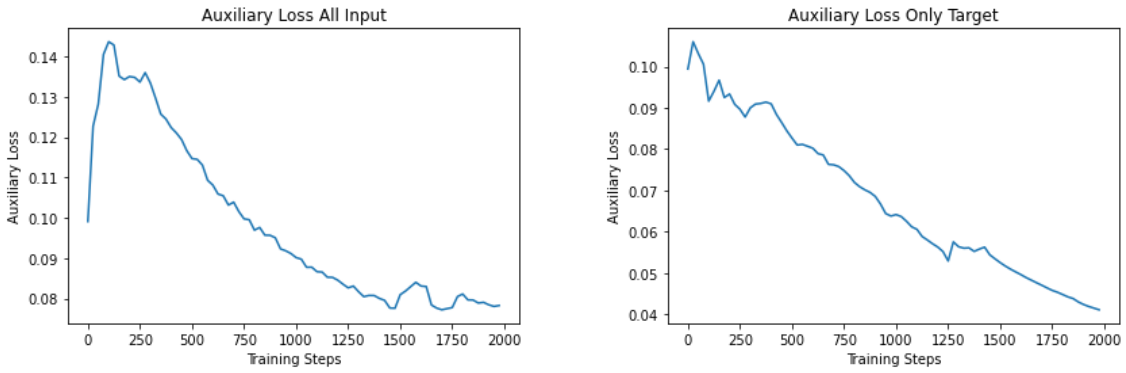

Figure 4: Change of the loss of the auxiliary network over 2000 training steps

As can be seen in Figure 4 the loss of the auxiliary network decreases. This means that the network improves in predicting the hidden state of the receiver LSTM based on the hidden state of the sender LSTM. This is true for both network architectures, i.e. where the sender sees only the target concept or the distractors as well. When plotting the loss and accuracy of the main signaling game however, this picture changes.

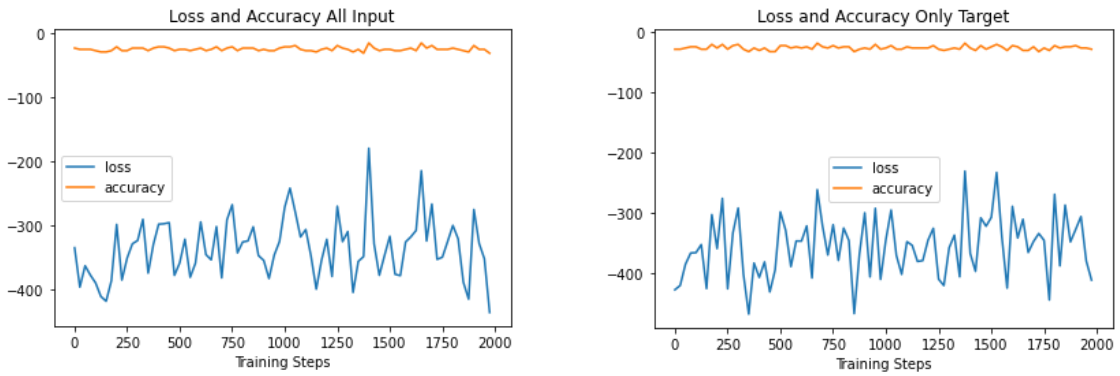

Figure 5: Evolution of loss and accuracy of the signaling game

In Figure 5 it is visible that both the loss and the accuracy of the signaling game do not improve over the training period. Both only oscillate to a certain degree, because of the random chance of receiving a slightly better or worse result at different time steps. Here it is clearly visible that the sender and receiver do not improve their understanding of each other.
We think the reason for this might be that a non-differentiable step when processing the outputs of the network or sending the message stops the gradients from flowing through the network. Unfortunately, we were not able to determine at what step exactly this problem occurs. In the notebook with the error recreation it is shown that the gradients do exist theoretically, probably because the trainable variables upon which the gradients can be applied exist, but they are denoted as “None”.

We can think of two different possibilities that might cause the error. One is that the custom loss-function is not differentiable, the other is that the error occurs when compiling the model as a whole, meaning that there is a fault in the graph of the model. In case the custom loss is the problem a solution might be to use one of the inbuilt Keras loss functions, for example the CategoricalCrossEntropy-loss. To use this loss, we would need to change the ConceptData class to output not only the natural language name of the target, but also the index of the target as a one-hot vector. It would then be possible to compute the loss directly between the target and the probability distribution, that is the output of the receiver.

In case the graph of the model is the cause of the error, it might be possible to solve the problem by rewriting the model such that it conforms to the standard of the functional API in tensorflow. Building and fitting the model inside the functional API makes it possible to inspect the graph of the model and spot possible mistakes. The model can also be trained easily with the inbuilt functionalities. Building our specific model conforming to this standard however, might prove difficult as the model structure does not only consist of different layers, but multiple models, processing steps and varying inputs and outputs.

Bibliography

[1] Marie Ossenkopf. Enhancing Language Emergence Through Empathy. ICLR 2020. https://openreview.net/pdf?id=Hke1gySFvB

[2] Richard S. Sutton, Andrew G. Barto, et al. Introduction to reinforcement learning, Volume 2. MIT press Cambridge, 1998.

[3] Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. Mach Learn 8, pp. 229–256 (1992). https://doi.org/10.1007/BF00992696

[4] Richard S. Sutton, David A. McAllester, Satinder P. Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. In Advances in neural information processing systems, pp. 1057–1063, 2000.

[5] EEG: A toolkit for multi-agent language emergence simulations. 2019. https://engineering.fb.com/2019/07/30/ai-research/egg-toolkit/

[6] Angeliki Lazaridou, Alexander Peysakhovich, Marco Baroni. Multi-Agent Cooperation And the Emergence of (Natural) Language. 2017. https://arxiv.org/pdf/1612.07182.pdf

[7] Christopher Olah. colah's blog. Understanding LSTM Networks. 2015. https://colah.github.io/posts/2015-08-Understanding-LSTMs

[8] David Lewis. Convention: A Philosophical Study. Harvard University Press, Cambridge, MA, 1969.

[9] ImageNet data set. https://www.image-net.org/

[10] Richard S. Sutton and Andrew G. Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[11] Angeliki Lazaridou, Karl Moritz Hermann, Karl Tuyls and Stephen Clark. Emergence of linguistic communication from referential games with symbolic and pixel input. In 6th International Conference on Learning Representations, ICLR 2018. https://openreview.net/pdf/e3a31e61b0fb2ed042cca612b5d54e24eac59adc.pdf

[12] Carina Silberer, Vittorio Ferrari, and Mirella Lapata. Models of semantic representation with visual attributes. In Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pp. 572–582, Sofia, Bulgaria, August 2013. Association for Computational Linguistics. https://www.aclweb.org/anthology/P13-1056.pdf

[13] Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In Intelligent Robots and Systems (IROS), 2012.

[14] Figure 2 in Marie Ossenkopf. Enhancing Language Emergence Through Empathy. ICLR 2020. https://openreview.net/pdf?id=Hke1gySFvB