# Recurrent Relational Networks

**Implementation Track, 3 Team Members**


## Abstract

This project is the implementation of Palm et al.'s Recurrent Relational Network [3], published on NeurIPS 2018, focusing on the algorithm developed by relational reasoning. Relational reasoning is the ability to logically infer, which can be treated as high-level intelligence as humans. The model leverages useful information, concerned with the values of nodes, edges of related nodes, to deduce the correct answer with multi-step relational reasoning. We nearly solve 1400/1792 hardest Sudoku(17 cells are given), in which the accuracy of each sample is larger than 60%, and completely solve 946/1792 Sudoku problems. Although the plot indicates that the accuracy is still increasing and does not reach convergence, due to the limitation of our operating conditions and unstable JupyterLab, we only set epoch as 50. The result reveals that the model is capable of complicated relational reasoning.


## 1 Motivation

With the development of Human Intelligence, the machine learning field is focusing more and more on how to abstract the interactions and dependencies between the objects, and using those features to predict or solve the problems automatically. Based on this trend, the paper develops a method, called Recurrent Relational Network, which is a generalization on the Santoro et al. [5]'s relational network, can be used to augment the model ability of dealing with many-step rational reasoning while realizing the basic function of the commonly used neural networks. Although there are some existing models, like Deep Convolutional Network may achieve a similar goal, the accuracy of it, however, according to the analysis conducted by Palm et al., is not quite ideal. When they apply it to auto-solve 24-36 given Sudokus, the accuracy of it can only achieve 70%. However, Recurrent Relational Network promotes at least 38% in accuracy with the same problem. Moreover, compared with the typical training method, such as multilayer perceptron (MLP) and multilayer convolutional neural network (CNN), the Recurrent Relational Network emphasizes the interdependence inside or between the objects. Instead of taking the entire object as an input and passing the results as a single forward output, the method makes it be able to reflect the process of deep learning, as well as the learning of interactive features. With this capability, applications of the method can be generalized to broad fields. For example, it can be used in the recommendation system with interactive features, or other deep learning fields with logical reasoning, like age arithmetic, which is taking a given value of age and a series of progressive relationships based on the given value as the input, and outputing the inferential value of the final layer in the relationship series. Besides, the advanced idea of recurrent relational reasoning can be combined with other existing models flexibly, and help to improve the performance of other models.

Based on the outstandings of the model, it's quite motivated to prove and show its learning ability. Therefore, to make the model more mature, the authors choose the Sudoku dataset as the training set to be implemented.

## 2 Problem Statement

Recurrent Relational Network, as a generalization of the relational network, is improved to be able to perform a series of relational operations involved in machine learning and artificial intelligence.

As an implementation of Palm et al. [3]'s Recurrent Relational Network, which is published on NeurIPS 2018, this project will follow its pattern closely and implement the mainly talked algorithm: Recurrent Relational Network model, on a new Sudoku dataset. Since the process of solving Sudoku shows how the human brain manages the logical operation and inference process, thus, research on its application on auto-solved Sudoku problems is wise to show how the relational reasoning function of the model works. Therefore, a dataset with massive unsolved hard Sudoku problems is not only a good representation of the supervised learning dataset to work on, but also there are many relational reasoning steps involved, which can be considered as a sample for logical inference.

The key point of the project is to decompose a function of relational reasoning into an encoded feature vector and a module that reasons about objects and edges first, then complete the implementation of it to achieve the goal of automatically solving the hard Sudoku problems. By successfully implementing the project, not only will the function of the method can be further proved, but it will also give a new insight into the method in terms of its accuracy and efficiency based on this dataset. The mathematical statement is illustrated below.

## 3 Related Work

Our implemented model, recurrent relational network [3] stems from the ideas of relational network [5] and interaction network [2].

Relational network [5], a simple form of neural network, has the following architecture illustrated in the figure. Questions are encoded with an LSTM to produce a question embedding and images are processed with a CNN to produce a set of objects for the relation network. The pairwise spatial cells are constructed using feature map vectors from the convolved image. The relation network considers the relation across all pairs of objects, conditioned on the question embedded, and integrates all these relations to answer the question.

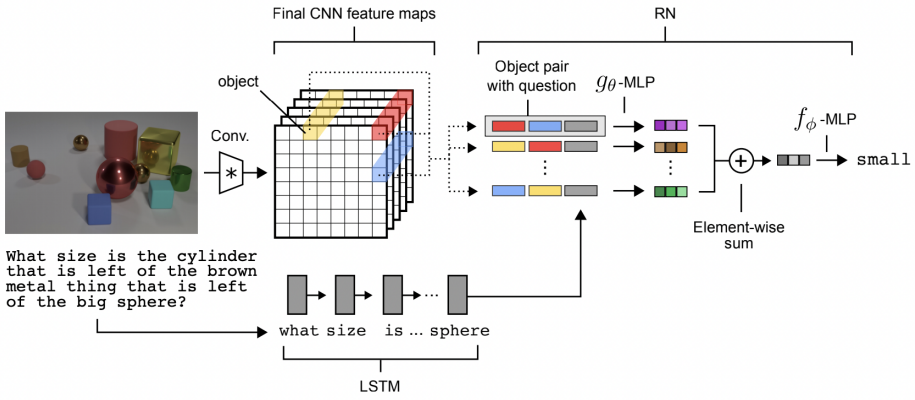


Figure 1: Relational Network Architecture

Interaction network [2] combines three powerful approaches: structured models, simulation, and deep learning. Structured models exploit rich, explicit knowledge of relations among objects and independent of the objects themselves. Simulation is an effective method for approximating dynamical systems, predicting how elements in a complex system are influenced by interactions with others. Deep learning couples generic architectures to provide learning and inference in real-world settings.

Relational network [5] and interaction network [2] only perform single update for nodes in hidden state. As it only updates one state each time, it does not support complex and complicated multi-step relational reasoning. While solving puzzles or sudokus, it requires more than one single step to store and pass messages. To solve this problem, recurrent relational network [3] compresses all the past and current relevant relations into a fixed size vector, and then performs the remaining relational reasoning in forward layers.

Relational network [5], interaction network [2], and recurrent relational network [3] can all be categorized as an instance of Graph Neural Networks. Graph Neural Networks with message passing derives from Scarselli et al [6]. However, there are important differences when implementing stable multi-step relational reasoning. Including the node feature at each update is essential for the stability of the network. In the work, Scarselli et al [6] puts the node feature inside the message function. Moreover, Scarselli et al [6] introduces an explicit loss term to ensure convergence. There are other ways to optimize loss at each step. Ross et al [4] proposes to train using the Inference Machine predictors on every step without hidden states.

Recurrent relational network [3] can also be regarded as a completely learned message passing algorithm. Belief propagation is a messaging passing algorithm for performing exact inference in directed acyclic graphical model. If the model has cycles, one can employee a variant, loopy belief propagation. Ross et al [4] proposes Inference Machines which ditch the belief propagation algorithm and instead train a series of regressors to output the correct marginals by passing messages on the graph.

There is also rich literature on combining symbolic reasoning and logic with sub-symbolic distributed representations. Serafini and Garcez [7] introduces the Logic Tensor Network (LTN), a neuro-symbolic formalism and computational model that supports reasoning through introduction of differentiable first-order logic called real logic as a representative language. Šourek et al [8] introduces the Lifted Relational Network (LRN), an architecture that allows for hierarchical relational modeling constructs and learning of latent relational concepts through shared hidden layers weights corresponding to the rules. Recurrent relational network [3] differs in that it does not provide approach to bridge symbolic and sub-symbolic methods. Instead, it concentrates on sub-symbolic methods.

Other related works using neural networks to solve puzzles and sudokus include OptNet [1] and Communication Network (CommNet) [9]. OptNet [1] is a neural network layer that solve quadratic programs using efficient differentiable solver. It is trained to solve four by four sudokus and beats the deep convolutional network baseline. CommNet [9] has similar structure with recurrent relational network except that it does not minimize the loss on each step.

In summary, recurrent relational network [3] is a model that supports multi-step updates with fully messaging passing and loss optimization on each step. It is an algorithm that focuses on sub-symbolic methods.

# 4 Method

## 4.1 Mathematical Representation

From the perspective of mathematics, the only difference between verbal reasoning and sudoku is how the messages are passing. In classic Sudoku, one has to fill a 9×9 grid with 1-9 so that every column, row, and box contains all the digits from 1-9. This is a logic-based puzzle but complicated to solve. For verbal logistic reasoning, humans always infer the meaning from the context and every word in a certain paragraph. Now, consider the word to be a given node, and the context is the whole grid. It is similar to the algorithm on the data of verbal reasoning as sudoku. For both tasks, the model needs to infer the answer from revealed information. The only difference between sudoku and verbal reasoning is that specific nodes are necessary for the former, while the other one needs whole information to learn the answer. To make the method(model) easier to understand, here we take Sudoku as an example.

A typical tactic to solve the puzzle is to start from one cell, and exclude all the existing numbers, and try all the possible values. The main idea is also the same as recurrent relational networks.

Since the position $(i, j)$ and the known digits $d_i$ can be considered as a "message" passed to the cell $j$, as shown in figure 2, we can infer the probable values of the cell for every digit from 1 to 9. These messages come from every related cell in the same column, row, and box, which can be represented as

$$m_j^t = \sum_{i \in N(j)} m_{ij}^t$$

where $N(j)$ denotes all the cells correlated to the cell $j$, and $m_{ij}^t$ is the message that comes from node $i$ to $j$ at iteration t. Let $h_j^t$ denotes the basic information of cell $j$ at iteration $t$, i.e. row number,

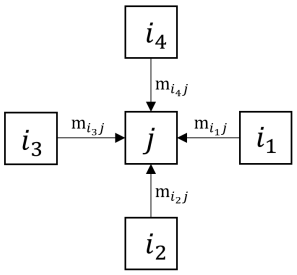


Figure 2: Example of how the cells pass messages.

column number, and constraints. Thus,

$$m_{ij}^t = f(h_i^{t-1}, h_j^{t-1})$$

We use these messages to derive the information of this cell in the next iteration, i.e.

$$h_j^t = g(h_j^{t-1}, m_j^t, X_j)$$

where $X_j$ represents the original information of cell $j$. And then, we leverage the state of the cell to derive the probability distribution of digits 1-9,

$$O_i^t = r(h_i^t)$$

Until now, we get the probability distribution of digits 1-9 for every cell, and we choose the most probable one to be the answer.

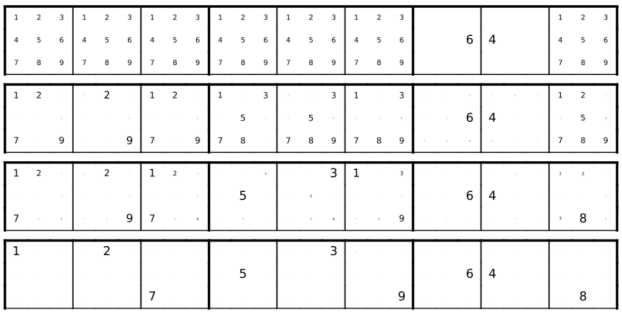

Figure 3: Example of how the trained network solves the first row of a Sudoku, from Palm, et al. [1]. The size of the number is represented as the probability of every digits.

To train the data set, we must have a loss function to evaluate the correctness of the output. We sum up the cross-entropy for each target $y_i$ with respect to each nodes.

$$L^t = -\sum_{i=1}^{81} \log O_i^t[y_i]$$

Here $O_i^t[y_i]$ is the probability of the target number of this cell, which is from 1 to 9. When the probability is getting close to 1, which means it has more confidence on the correct digits, the loss of this cell will converge to 0.

### 4.2 Model Construction

After all the mathematical representation, now we focus on the structure of our model.

The basic neural network structure of each function above is multilayer perceptron and long short-term memory. The multilayer perceptron has the advantage that it is able to fit any form of function. Long short term memory has the ability to store some important information for the long term, which

is a branch of recurrent neural networks. Since the model is complicated, to make it comprehensible, the proclamation order will be followed as figure **??** indicates.

To be specified, there are three hidden layers for every multilayer perceptron and 96 nodes in each hidden layer for both multilayer perceptron and long short-term memory. Set learning rate to be 2e-4, and regularization parameter to be 1e-4.

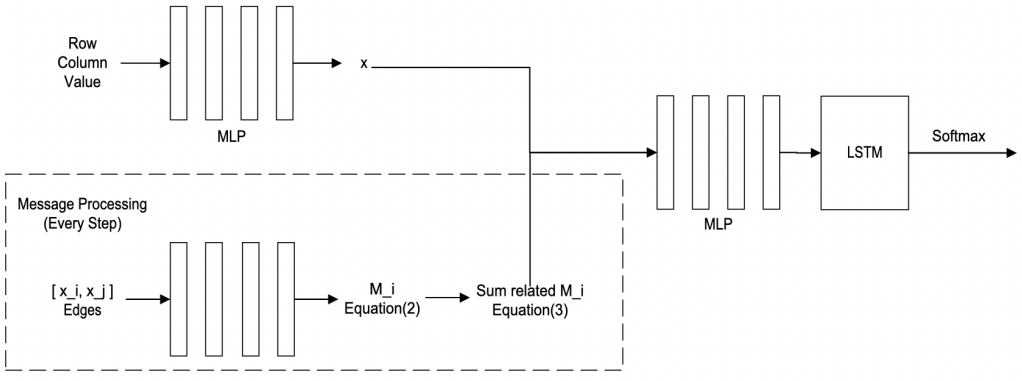


Figure 4: Final model of recurrent relational network.

**Preprocessing** Before we build the architecture of our model, it is important to construct a fully connected layer as an embedding layer to generate the features of every node from positions and original numbers, otherwise, there is no difference between row number and column number. It is also the same process as for verbal reasoning, which needs to be encoded as numeric vectors. The embedding layer is always used for sparse matrix, which can be projected to a fixed dimensional vector. For example, in the natural language process, the word "machine" can be represented as a 4-dimensional vector, $[0.2, 0.4, 0.5, 0.3]$, rather than a dictionary vector with too many zeros. In this experiment, we set 16 neurons for the embedding layer, i.e. every number or word is vectorized as a $16 \times 1$ tensor.

Denote the digit for cell $j$ $d_j$ (0-9, 0 if not given), and the row and column position $row_j$(1-9) and $column_j$(1-9). The features of the cell is

$$X_j = MLP(embed(d_j), embed(row_j), embed(column_j))$$

where the output has 16 nodes, i.e. $X_j \in R^{16\times 1}$.

**Step 1** After generating features, we construct a multilayer perceptron, as defined before, to get the message function $f$.

$$m_{i,j}^t = MLP(X_i^{t-1}, X_j^{t-1})$$

Moreover, we need to sum up all the related messages as equation(1) illustrated, which is all the conditions needed for inference.

**Step 2** Furthermore, we utilize multilayer perceptron and multilayer long short-term memory(LSTM) to infer the next state of every cell, since it will abandon the trivial possible numbers of the cell and value the most likely digits. Figure 5 indicates an example of multilayer LSTM to help interpret this step.

$$h_j^t = LSTM(m_j^t, h_j^{t-1})$$

**Step 3** We repeat steps 1 and 2 for 32 times, which can be interpreted as 32 steps of logical inference. Finally, we will choose the digit with the largest possibility. Here the output function $r$ is a fully connected linear layer with nine outputs which represent the possibility of every digit in this cell, and the input is the results of the previous step with LSTM.

$$O_i^t = r(h_j^t)$$

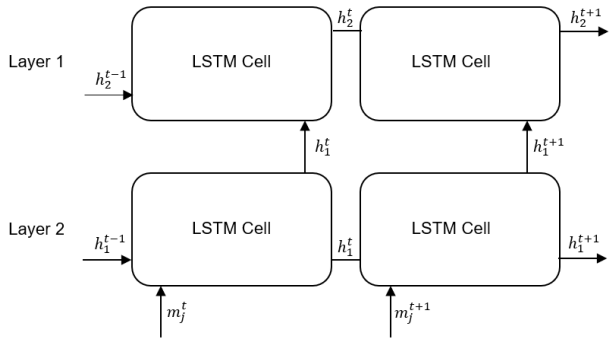


Figure 5: Example of Multilayer LSTM

# 5 Experiments

## 5.1 Dataset

We utilize the dataset of 100000 9×9 individual Sudoku and their solutions to choose 18000 hardest Sudoku problems, given 17 nodes initially, which is the least quantity of a solvable Sudoku, as training data to implement the Recurrent Relational Network method. The data can be found at `https://www.kaggle.com/bryanpark/sudoku`.

## 5.2 Evaluation

The performance of recurrent relational network can be evaluated in two aspects: accuracy and convergence loss.

For accuracy, at the end of each epoch, we count how many examples are solved with completely correct solutions and how many examples are solved with almost correct solutions (60 % correct answers). The author of Recurrent Relational Network [3] considers initial givens ranged from 17 to 34 while we just tried the most difficult situation with only 17 initial givens. We want to see whether we could reach high accuracy in our data set (17 initial givens) by implementing the recurrent relational network.

The author highlights that multiple steps are required in passing messages in recurrent relational network. To examine their hypothesis, they verify by plotting the accuracy result as a function of number of steps. In their work, they use 64 steps and reach 96.6% accuracy even in the most difficult task. In our work, we use 32 steps instead. The reason is followed. On one hand, 64-steps LSTM is less efficient (more training time), and it brings us computational issues in practice (requirement for use of GPU). On the other hand, based on the result given by the author, 64-steps training only improves 5% accuracy. Hence we consider using 32-steps LSTM instead.

We record the accuracy for both completely correct and almost correct solutions at the end of each epoch. See figure 6 below, after 50 epochs, the accuracy for complete correct solutions exceeds 50% and the accuracy for almost correct solutions exceeds 70%. Our experiment shows that our final accuracy is not as good as suggested by the author's work. But it does get improved as we increase epochs.

Here we list several reasons that account for why our accuracy is less than expected. Firstly, in the author's work, they train their model with the employment of TITAN X GPUs, which is far more sophisticated than ours. In our experiment, our resources is limited with only a 2080 GPU. Secondly, the author sets a hundred thousand epochs to train their model while we only run 50 epochs because of lack of capacity. Finally, we implement the method using pytorch rather than tensorflow. For higher versions of tensorflow, some packages are removed and we rewrite the code using pytorch. The different functions and optimizers in platforms may lead to different final result.

For convergence loss, we examine the loss on training examples for 48 and 96 batch sizes separately. The training examples consist of 18000 sudokus. For training loss, we try to figure out how hidden

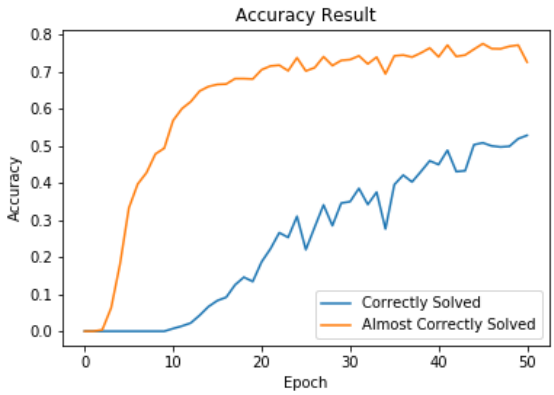


Figure 6: Accuracy Result

size plays a role in model training. Since hidden size describes the complexity of network and increasing hidden size in a certain interval provides a better fit of the model. We want to show how hidden size affects model by evaluating the convergence loss. In this way, we can draw a conclusion on the sensitivity of our model to the hidden size. On the other hand, increasing hidden size leads to more computation time. We would like to see how the training time changes after we increase the hidden size. The result is shown in figure7.

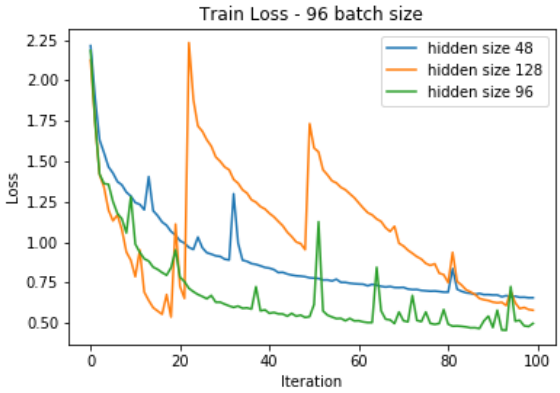


Figure 7: Train Loss - 96 batch size

The result says that 96 hidden sizes stand out, i.e. smallest convergence loss and convergence time. Note that for 128 hidden sizes, it initially outperforms but then follows by a sudden jump in loss. We propose two explanations. Firstly, 128 hidden sizes causes overfit and get stack in local minima. This reminds that one should pay careful attention when choosing appropriate hidden sizes. Secondly, some batches of data set may have common features. The solution is to randomize the original data set and more details need to be checked.

Now we consider training loss with 48 hidden sizes and 48 batch sizes. The result is shown in figure8. It is significantly different from the previous figure. The drift of training loss in figure7 seems much better, but the training rate is significantly slower than before.

The batch size can effect the model from several aspects. First and foremost, a large batch size decreases the time consuming for every epoch and improves the efficiency of the training process. Second, to a certain level, a proper batch size reduces the probability to get stuck in local minima. However, large batch size may also affect the precision of the training model.

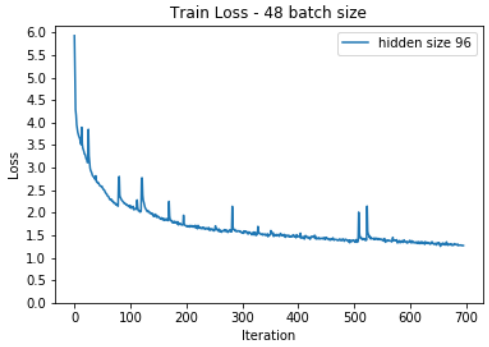


Figure 8: Train Loss - 48 batch size

# 6 Conclusion

The key insight of this relational reasoning model is message passing, which make the module can be added to other deep learning models to realize relational reasoning. It significantly improves the accuracy of traditional models, such as convolutional neural networks. However, it still has some shortcomings, while the most important one is that the model easily get stuck in a local minima, as you can see from figure7. Moreover, summing all the messages seems lack of sophistication and precise. However, this point needs to be justified, since the model is capable for dropping out useless messages and multilayer perceptrons can fit any functions.

In conclusion, we attempt to implement the recurrent relational network to solve sudoku problems. The difficulties lie in the deep learning network and quality of our data set. For our data set, solving sudokus with only 17 initial givens is a challenging task. We achieved following success in our project. Firstly, our trained model successfully solved more than half of the puzzles. If continued, the model will perform even better as expected. Secondly, we gain valuable executive experiences in real practice. Finally, we explored how to use pytorch building complex network structure.

## 6.1 Executive Summary

The original paper uses several TITAN X GPUs to train the model. Although the conditions of our experiment is limited, we still learn and conclude some useful model training experience to share.

1. **Set checkpoints in the program and save the trained model every several epochs.** At the beginning of our experiment, we did not realize the importance of this step, and it was always interrupted due to the network accessibility operating on JupyterLab.
2. **It is better to use command-line interface, while JupyterLab is unstable.** When we train our model via JupyterLab on the cluster servers, it always broke in several hours and was really time consuming.
3. **Parallel computing is useful when the model is complicated.** The author utilized several GPUs to train the model, which needed parallel computing. However, Pytorch does not perfectly support this function, while Tensorflow has some advantages.
4. **Run out of memory** When we ran the program with larger batch size than 96, it would be out of memory. We can estimate the necessary parameter stored for our program as

   $$Memory = N_{Model\ parameter} \times N_{Optimizer\ parameter} + N_{Batch\ size} \times N_{Total\ parameter}$$

   **Solutions:** First, you can decrease the batch size, causing quantity of parameters decreasing. Second, choose a proper data type, such as lower precise float. Third, release unnecessary parameters while training. Finally, gradient accumulation helps overcome the problem, which is a mechanism for splitting the batch of samples[1].
5. **Batch size needs to be seriously considered.** This is illustrated in the previous part.

[1]Introduction of gradient accumulation: https://towardsdatascience.com/what-is-gradient-accumulation-in-deep-learning-ec034122cfa

**Contribution by Group Members**

Author 1: Write the part of Method, including mathematical statement and model construction, as well as the conclusion. Code the part of model construction(Class Mydataset).

Author 2: Write the part of related work, experiments(evaluation) and references. Code the part of data processing and function check_val.

Author 3: Write the part of motivation and problem statement. Implementing models for comparison. Data processing and evaluation.

## References


[1] Brandon Amos and J Zico Kolter. Optnet: Differentiable optimization as a layer in neural networks. In *International Conference on Machine Learning*, pages 136–145. PMLR, 2017.

[2] Peter W Battaglia, Razvan Pascanu, Matthew Lai, Danilo Rezende, and Koray Kavukcuoglu. Interaction networks for learning about objects, relations and physics. *arXiv preprint arXiv:1612.00222*, 2016.

[3] Rasmus Berg Palm, Ulrich Paquet, and Ole Winther. Recurrent relational networks. *Neural Information Processing Systems*, 2018.

[4] Stephane Ross, Daniel Munoz, Martial Hebert, and J Andrew Bagnell. Learning message-passing inference machines for structured prediction. In *CVPR 2011*, pages 2737–2744. IEEE, 2011.

[5] Adam Santoro, David Raposo, David GT Barrett, Mateusz Malinowski, Razvan Pascanu, Peter Battaglia, and Timothy Lillicrap. A simple neural network module for relational reasoning. *arXiv preprint arXiv:1706.01427*, 2017.

[6] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE transactions on neural networks*, 20(1):61–80, 2008.

[7] Luciano Serafini and Artur S d'Avila Garcez. Learning and reasoning with logic tensor networks. In *Conference of the Italian Association for Artificial Intelligence*, pages 334–348. Springer, 2016.

[8] Gustav Sourek, Vojtech Aschenbrenner, Filip Zelezny, and Ondrej Kuzelka. Lifted relational neural networks. *arXiv preprint arXiv:1508.05128*, 2015.

[9] Sainbayar Sukhbaatar, Rob Fergus, et al. Learning multiagent communication with backpropagation. *Advances in neural information processing systems*, 29:2244–2252, 2016.