Recurrent Neural Physics Engine

Andrew Nam
Department of Psychology
ajhnam@stanford.edu

Jun Hwan Ryu
Department of Psychology
jhryu25@stanford.edu

Jinxiao Zhang
Department of Psychology
jzhang18@stanford.edu

1 Introduction

The humans brain possesses intuitive understandings of physics [1] learned during infanthood [2], allowing them to reason about the world using mental simulations [3]. For example, when given tasks to predict the final position of an object given its initial state, humans simulate possible trajectories that might occur given the environment and starting configurations [4, 5]. Understanding the neural architectures and computations that may lead to accurate physical simulations that reflect the processing of the human brain.

In this work, we introduce the recurrent neural physics engine as a tool for studying human mental simulations. Although successful neural physics engines already exist [6, 7, 8, 9], previous work has focused on models that deal with graphic inputs in which the interaction between objects can be easily detected by the gap pixels between them. In this project, we built models to simulate Newtonian interactions using numeric inputs (e.g., position, velocity, rotation). Furthermore, we leverage the power of recurrent neural networks to extract hidden representations across time.

2 Related Work

Some of the most successful recent neural physics engine models leverage the flexibility of graph networks to compute the interaction between multiple objects. Visual Interaction Networks [8] employ shared modules between every possible pair between objects to compute the effects of one object on another. They use a collision-rich dataset with multiple objects constantly colliding with each other. Propagation Networks [9] use a message passing scheme with multiple passes to propagate the effect of one object through a chain of objects, such as in a Newton's cradle where objects may affect one another indirectly. However, in these models, there are still room for improvement in handling collisions.

More work has been done looking at time-series data analysis with recurrent neural networks. Recurrent neural networks often face vanishing gradient problems with long time series data, and architectures such as LSTM and GRU [10, 11], with gating mechanisms, have been designed to alleviate the problems of the vanishing gradients. In this project, we implement the GRU, which consists of less parameters.

3 Dataset and figures

To allow the model to learn a representation of the environment, we used the Plinko task environment [5] consisting of an enclosed environment with 3 obstacles (a triangle, rectangle, and pentagon; Figure 1). Each simulation was initialized with a ball set at one of the three holds at the top of the box with some random velocity. All simulations used the same parameters for friction, gravity, elasticity, size and shape of objects, etc. All simulations were deterministic except for: (1) initial velocities which were sampled from a small variance Gaussian and (2) collisions, which would add Gaussian noise to the post-collision velocity of the ball.

We created training and testing sets from the Plinko physics simulation engine. The training set consisted of 1000 worlds, with different positions and orientations for each of the obstacles, and we

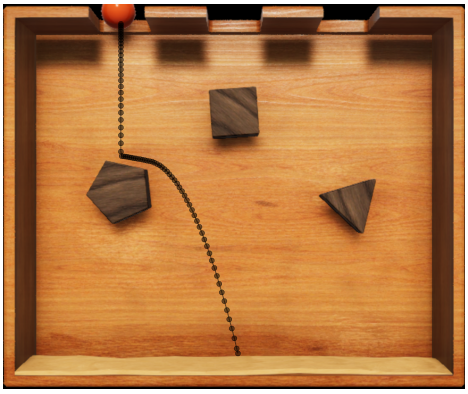


Figure 1: An example of Plinko ball dropping. A ball is released from the first hole. The ground truth of the falling path of the ball is shown. Each point represents one timepoint

ran 100 simulations in each of world, dropping from different holes. The test set contained new 1000 randomly generated worlds, with 10 simulations in each world.

The network took as **environment** inputs, the environment parameters: a tuple of shapes (Triangle, Rectangle, Pentagon), each shape a tuple of position and rotation $(x_{s,n}, y_{s,n}, r_{s_n})$ where s denotes the shape and n is the n^{th} simulation.

In addition, the network took as **state** inputs, the position of the ball at a given time, $(px_{n,t}, py_{n,t})$, the velocity of the ball at a given time, $(vx_{n,t}, vx_{n,t})$ and a hidden state $h_{n,t}$ in the recurrent layer that was dependent on the series of the previous inputs $\{(px_{n,0}, py_{n,0}), (px_{n,1}, py_{n,1}), \ldots, (px_{n,t-1}, py_{n,t-1})\}$. We denote the input state of the ball $B_{n,t}$ as a tuple of position and velocity $(px_{n,t}, py_{n,t}, vx_{n,t}, vy_{n,t})$ where t is the timestep during a simulation.

The goal of the model is to produce a simulated trajectory given the initial state $state(t = 0)$. A trajectory is defined as the set of states of the ball for every step $t \in [0, T]$ where T is the duration of the trajectory. More formally, trajectory$_n = \{B_{n,0}, B_{n,1}, ..., B_{n,T}\}$

4 Methods

All codes can be found on: https://github.com/andrewnam/plinko_nn/

We trained various models with different representations, architectures and hyperparameters.

1. Non-relational recurrent neural network
2. Relational recurrent neural network
3. Model with collision module integrated

4.1 Collision module

Collision detection network. Takes as inputs the state and environment embeddings. Each environment is concatenated separately with the state embedding and then fed into an MLP with 3 layers with an ELU activation function. Outputs a probability of collision of the ball with each of the environment (or probability of no collision). The collision detection network is trained separately, using a balanced dataset of ball states, with equal frequencies of collision with each object, each walls, the ground or free fall.

4.2 Models Tested

We used different models with comparable number of neurons as the full model. Note that there are still differences in the different number of neurons and layers across the models. The different models were:

1. **GRU**: gated recurrent unit.
   (a) 1 state embedding: MLP with 16 neurons as the output.
   (b) 1 environment embedding: fed in all the environment variables into one MLP with 32 neurons as the output.
   (c) 2 gated recurrent unit with 128 hidden neurons.
   (d) Final MLP with 2 hidden layers with the number of hidden neurons equal to the output size of the gated recurrent unit.
2. **cGRU**: GRU with a collision module.
3. **rGRU**: relational GRU. Architecture shown in figure 2.
   (a) 1 State/3 environment embeddings: MLP with 16 hidden/output neurons and 4 layers.
   (b) 4 fixed embeddings: vectors of the size equal to the embedding outputs, to account for the potential interaction with other parts (for instance, left or right wall, ground and no collision).
   (c) 2 GRUs with 16 neurons.
   (d) 7 relational modules: MLP with 2 layers and relu activation.
   (e) Final MLP layer with 32 hidden neurons and 2 layers linear units.
4. **rcGRU**: As described above, with the collision module.
   (a) Reweighting layer: weighted the outputs of the relational mopdules by the prediction of the output of the collision module (by 1 or 0, if there was a collision with that object).

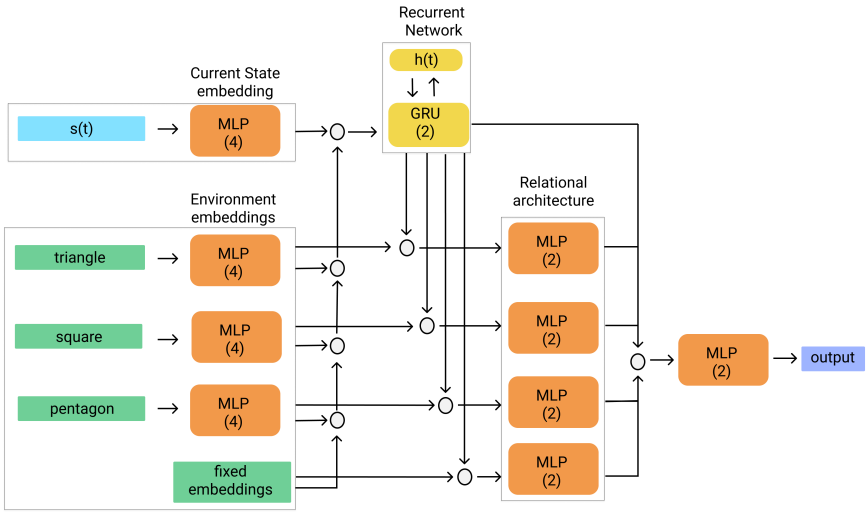


Figure 2: Relational recurrent neural network (rGRU)

4.3 Training

1. **GRU**: gated recurrent unit. The model was trained for 100 epochs, using the Adam optimizer, with learning rate $5e-4$ and weight decay of $1e-3$ for the weights while the bias terms were not regularized.
2. **cGRU**: Same as GRU.
3. **rGRU**: Trained 500 epochs with $lr = 1e-2$. then another 500 epochs with $lr = 1e-3$.
4. **rcGRU**: Trained the network in multiple stages. First, we fixed the embeddings and the collision module, transferring the weights from collision task. The network was trained by decreasing the learning rate from 0.01, 0.001, 0.0001, each with 200 epochs. Then, all the weights were freed, and learned with $lr = 1e-3, 1e-4$ for 500 epochs each.

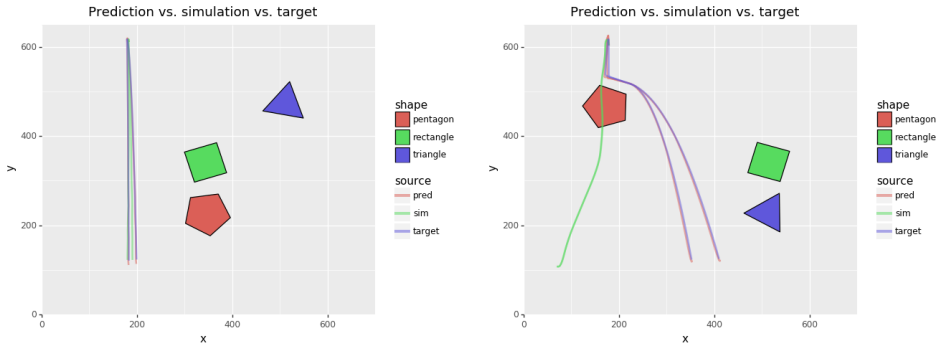
5 Results

5.1 Collision Task

The collision module did relatively well on the collision task achieve up to **96.5%** accuracy. Test cases in which the target was an object collision, the module predicted the correct collision with **99.0%** accuracy, and the free falls were predicted with **86.1%** accuracy.

5.2 GRU

Generally, the prediction of position at the next timepoint $(px, py)_{t+1}$ ("pred" in figures) given the current state $state(t)$ was very close to the ground truth ("target" in figures) as shown in Figure 3. Compared to the ground truth, the prediction had an "overshoot" at the collision point (Figure 3b). The GRU simulation of free falls ("sim" in figures) was pretty good as show in Figure 3a. However, in cases where there are collisions, the model "ignored" the obstacles (Figure 3b), which suggests that the GRU model does not implicitly learn the collisions. Those results motivate us to add a collision module in the recurrent neural network.



(a) GRU: Free falls.

(b) GRU: Falls with a collision.

Figure 3: GRU without a collision module. Target: the ground truth; pred: prediction of state at time $t + 1$ given state at time t; sim: the whole simulation of states at $t_1, t_2, ..., t_n$ given the initial state t_0.

5.3 cGRU

With a collision classifier built in the recurrent neural network, the prediction of next position given the current state was even closer to the ground truth (target). In particular, it had a smaller "overshoot" at collision pints compared to the GRU without a collision module (Figure 4a). In simulations, the cGRU model did not "ignore" the obstacles but diverged from the ground truth very early in its path.

5.4 rcGRU

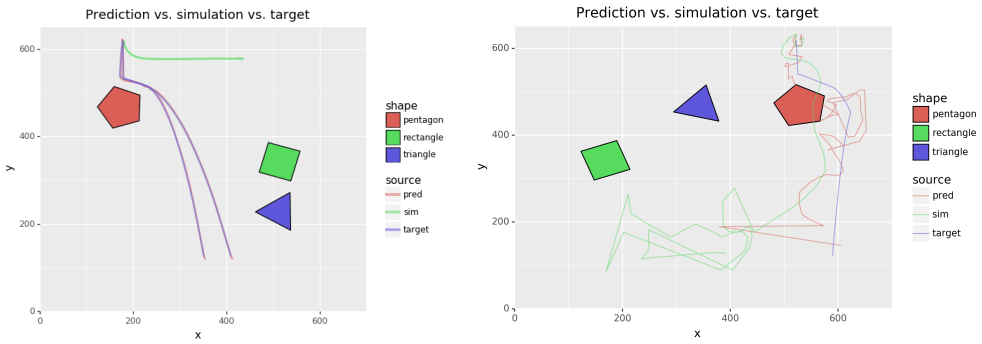
The model (rcGRU) had a hard time performing. In specific, the model still had more trouble integrating collision information with the hidden representations of the architecture (Figure 4b).

5.5 Comparisons between models

Comparison of the mean square errors (MSE) between predictions and the ground truth across the models are shown in Table 1.

6 Conclusion and future directions

The GRU or the rGRU was able to generate samples with a smooth continuous fall, has difficulty learning collisions. Furthermore, the full simulations (predict entire trajectory given the initial position), if they have one bad sample, follows through the trajectory. This may be due to the inherent



(a) GRU with a collision module (cGRU) (b) Relational GRU with a collision module (rcGRU).

Figure 4: GRUs a collision module (cGRU and rcGRU). Target: the ground truth; pred: prediction of state at time $t+1$ given state at time t; sim: the whole simulation of states at $t_1, t_2, ..., t_n$ given the initial state t_0.

Table 1: Model comparisons in MSE.

Model	training MSE	testing MSE
GRU	2.49×10^{-3}	5.36×10^{-3}
cGRU	8.56×10^{-4}	2.99×10^{-3}
rGRU	3.1×10^{-4}	3.00×10^{-3}
rcGRU	2.9527	3.0745

nature of the dataset, in which there is a mismatch in the number of timepoints in which the ball is free falling, and the few collisions that the ball encounters during the drop.

The collisions module implemented accurately detects collisions with a high accuracy. However, the neural networks (cGRU and rcGRU) have a hard time integrating the information from the collision module with the hidden representations of from the other modules, especially when the collision module directly modulates the outputs of the relational modules (without concatenation). This may be due to imperfect performance of the collision module, since the module often misclassified the free falls, or may be due to insufficient training. We also found an effect of regularization, since big lambda value competing with the small gradients in the GRU leads to the network not learning anything with gradient descent. In any case, more work is needed to integrate the different modules of the architecture.

A potential direction for the project include using reinforcement learning or using different model architectures to avoid sequential sampling. A reinforcement learning approach may have the physics simulator interact with an agent to induce more collision samples. Another future direction include incorporating human data to the model to see compare the variances of the generated trajectories from the human predictions.

7 Contributions

Andrew Nam (cs236) set up the github, wrote all the base code for simulation and the test bed, as well as experimenting with various different approaches and models (baseline models, variational approaches, pixel-space inputs etc.). **Jun Hwan Ryu** (cs229,cs236) coded and experimented with the relational architectures, and wrote the cs229 paper. **Jinxiao Zhang** (cs229) coded and experimented with recurrent gru architectures, wrote all the plotting functions, and wrote the cs229 paper.

References

[1] Tobias Gerstenberg and Joshua B Tenenbaum. Intuitive theories. *Oxford handbook of causal reasoning*, pages 515–548, 2017.

[2] Renée Baillargeon. Infants' physical world. *Current directions in psychological science*, 13(3):89–94, 2004.

[3] Peter W Battaglia, Jessica B Hamrick, and Joshua B Tenenbaum. Simulation as an engine of physical scene understanding. *Proceedings of the National Academy of Sciences*, 110(45):18327–18332, 2013.

[4] Kevin A Smith, Filipe Peres, Edward Vul, and Joshua Tenebaum. Thinking inside the box: Motion prediction in contained spaces uses simulation. In *CogSci*, 2017.

[5] Tobias Gerstenberg, Max H Siegel, and Josh Tenenbaum. What happened? reconstructing the past through vision and sound. In *CogSci*, 2018.

[6] Katerina Fragkiadaki, Pulkit Agrawal, Sergey Levine, and Jitendra Malik. Learning visual predictive models of physics for playing billiards. *arXiv preprint arXiv:1511.07404*, 2015.

[7] Peter Battaglia, Razvan Pascanu, Matthew Lai, Danilo Jimenez Rezende, et al. Interaction networks for learning about objects, relations and physics. In *Advances in neural information processing systems*, pages 4502–4510, 2016.

[8] Nicholas Watters, Daniel Zoran, Theophane Weber, Peter Battaglia, Razvan Pascanu, and Andrea Tacchetti. Visual interaction networks: Learning a physics simulator from video. In *Advances in neural information processing systems*, pages 4539–4547, 2017.

[9] Yunzhu Li, Jiajun Wu, Jun-Yan Zhu, Joshua B Tenenbaum, Antonio Torralba, and Russ Tedrake. Propagation networks for model-based control under partial observation. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 1205–1211. IEEE, 2019.

[10] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling. In *arXiv e-prints*, page 1412.3555.

[11] Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation. page arXiv:1406.1078, 2014.