CS236 Final Report: Generating Physical Dynamics of 3D Rigid Objects

Davis Rempe
SUNetID: drempe

1 Introduction

Humans have a natural ability to reason about the physical world around us. For example, being able to roughly estimate where an object will end up if we push it across the floor, or catch a ball that is thrown towards us. This physical “intuition" allows us to successfully operate and navigate in previously unseen environments, and easily interact with objects we may be experiencing for the very first time. Though so natural to us, physical understanding is still completely absent from most intelligent systems. Endowing a system with this kind of physical understanding would be extremely useful for applications such as robotics where it would improve navigation, scene understanding, and manipulation and grasping of complex physical objects.

In this project, we take a step towards robust physical scene understanding by exploring the use of generative models to learn the motion of 3D rigid objects sliding on a plane. By using full object shape information through 3D point clouds, we seek to generalize learned physical dynamics to objects not seen during training, and make the model amenable to real-world application by using input data that can easily be captured. We model physical dynamics using a recurrent neural network (RNN) which, given an object point cloud and an initial linear and angular velocity, predicts parameters of a Gaussian distribution which models the change in velocities, position, and rotation of the object for the next time step. We can roll out this model and iteratively sample from the predicted Gaussians to generate new object trajectories, or we can evaluate the likelihood of a given full trajectory. All code for the project is available on GitHub (github.com/davrempe/cs236-gen-dynamics).

2 Related Work

One approach to physical understanding that has recently been explored is *direct forward prediction*. These kinds of methods typically take in the current state of objects in a scene, the environment, and external forces, and attempt to predict the state of objects at future points in time. An advantage to this approach is that it can be used to “hallucinate" future states of physical systems and therefore be used, for example, to plan actions of an autonomous agent[1].

Much of the previous work in forward prediction has focused on 2D objects and environments. Fragkiadaki et al. [2] learn to predict the future velocities of 2D billiard balls on a table using an LSTM. Their model takes in the 4 previous image frames centered around a ball of interest and outputs velocities for 20 frames into the future. The *neural physics engine* [3] and *interaction network* [4] also predict the motion of 2D balls, but instead of image input, they use a state representation of each object which gives position and velocity. Using this state and defined relationships between objects, they can predict the outcome of complex interactions like collisions using only a fully-connected network. Although they produce believable results, none of these 2D methods can generalize outside of the objects used during training, and only show results for simple primitive shapes.

One approach for 3D from Mottaghi et al. [5] predicts a sparse sequence of linear velocity directions for 3D bounding boxes of objects using an RNN given an image as input. Unfortunately, using images as input results in a loss of information about 3D shape. Most recently, the *hierarchical relation network* [6] breaks 3D objects into a hierarchical graph of particles which they use to learn

Preprint. Work in progress.

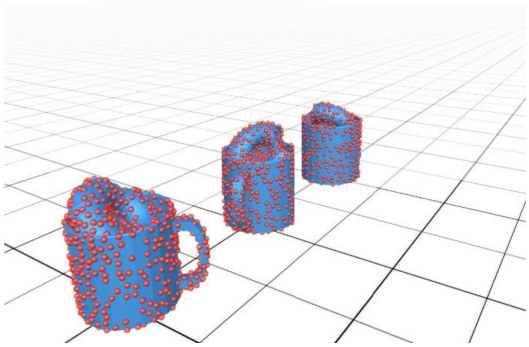

Figure 1: *Problem formulation*. Our model generates physical motion for a rigid object sliding on a plane. Given the object point cloud (shown in red) and current state, we output the object state (velocity, position, rotation) at multiple future time steps.

per-particle dynamics and their relationships from simulated data. Predicting motion for each particle results in motion of objects in the scene. Though this method does generalize to unseen objects that can be broken into the particle structure, applying this model in the real world would be very difficult considering the detailed per-particle supervision required.

Two main drawbacks of prior work which we seek to address in this project are the lack of object generalization and real-life applicability. Additionally, none of the above approaches are generative, focusing instead on predicting one deterministic correct answer that a simulator provides for the next object state. This assumption of determinism may hold for simulation, but in the real world things get messier. Predicting the dynamics of an object in the real world will require some sort of system identification that gives physical properties and will never be perfectly accurate. Therefore, modeling a distribution of motion would be useful to quantify uncertainty and capture some of the unpredictability of real-life dynamics when hallucinating forward in time.

3 Problem Formulation

We want to generate the trajectory of a 3D rigid body in sliding motion on a plane (see Figure 1). As input we are given the object shape in the form of a 3D point cloud ($\mathbf{O} \in \mathbb{R}^{N\times 3}$) and its current state $s_t = (\mathbf{p}_t, \theta_t, \mathbf{v}_t, \omega_t)$ at time t where $\mathbf{p}_t \in \mathbb{R}^2$ is the current position, $\theta_t \in \mathbb{R}$ is the current rotation about the vertical axis, $\mathbf{v}_t \in \mathbb{R}^2$ is the current linear velocity, and $\omega_t \in \mathbb{R}$ is the current angular velocity about the vertical axis. Note that we treat an object as a single rigid body with a single state vector, we **do not** model motion of points in the point cloud individually. We assume that during motion, the object does not fall over and remains upright on the plane, therefore we may parameterize position and linear velocity with 2 components, and rotation and angular velocity with 1. As output, we give m future states of the object $s_{t+1}, s_{t+2}, \ldots, s_{t+m}$. We further assume that gravity acts on the object, that there is a constant non-zero friction coefficient between the object and plane which is the same for all objects, and that the object density is constant across all objects (therefore mass is proportional to volume).

This problem formulation is similar enough to 2D to make it feasible for a course project, but adds a number of added difficulties. We are interested in the motion of 3D objects, which requires a deep understanding of the relationship between object shape, surface contact area, and velocity to predict accurately. This requires learning to extract implicit salient physical features of objects from the given 3D point cloud, specifically moment of inertia about the vertical axis, object mass (which depends on volume), and contact surface area and shape with the ground plane. Furthermore, object motion is not explicitly constrained to the plane in any way. During sliding, the object may slightly wobble causing a change in frictional force which results in complex non-linear motion.

Models that solve the presented problem should generate “realistic" trajectories. This can be evaluated both quantitatively and qualitatively. First of all, the output position, rotation, and velocities (or the expectation of the the distributions for these values) should be close to ground truth object motion. We want this to hold for both short-range (1 step in the future) and long-range (m steps) outputs

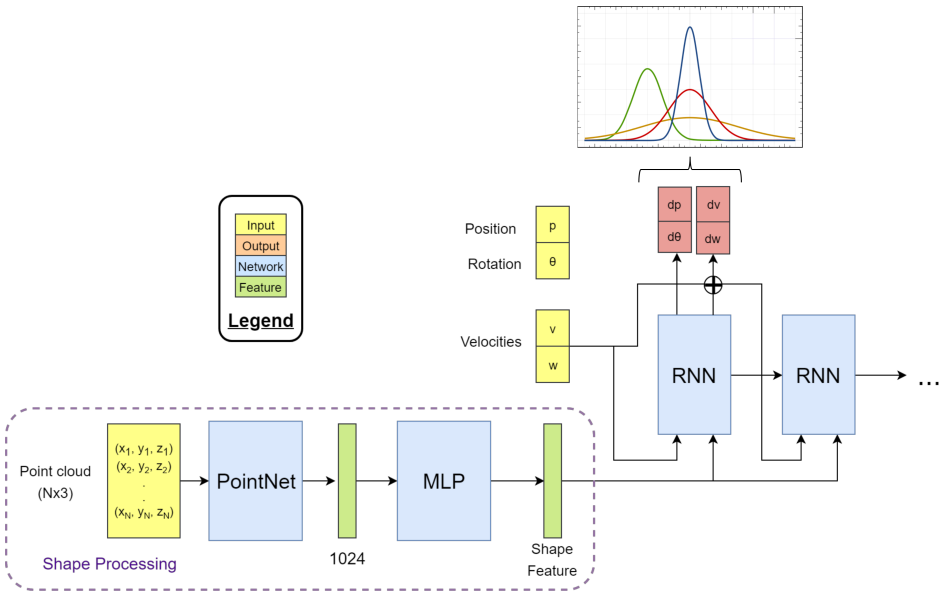

Figure 2: *Model architecture*. We process the object point cloud using PointNet to extract a shape feature. This feature, along with object velocities at the current time step are passed to a recurrent module (shown rolled out) which parameterizes distributions for the change in velocities, position, and rotation.

of the model. For this project, ground truth motion is produced by physical simulation. Secondly, motion should be qualitatively believable - a human should not be able to tell the difference between a real and generated trajectory.

4 Methods

Here we detail our proposed approach for solving the problem including the neural network architecture, datasets used, and training procedure.

4.1 Model Architecture

We model a sequence of object states using a recurrent neural network (RNN) as shown in Figure 2. Similar to an autoregressive model, this architecture conditions its output at the current step on all previous steps, but previous steps are summarized with a "hidden" state which is continually updated as it processes sequential data.

More specifically, we use an RNN to parameterize the distribution of the conditional object state at each time step. This distribution is assumed to be Gaussian. The architecture allows us to generatively sample new object trajectories as well as evaluate the likelihood for a given trajectory. Mathematically, we define the distribution of the object state s_t at time step t as

$$p(s_t|s_{1:t-1};\alpha) = \mathcal{N}(s_t; s_{t-1} + \mu_t, \sigma_t^2)$$

where $s_{1:t-1}$ are previous states and α are the parameters of the RNN. μ_t and σ_t^2, which are used to parameterize the Gaussian, are the outputs of the RNN, call it f. These outputs are a function of the previous object state, a *shape feature* o, and the previous hidden state h_{t-1}:

$$(\mu_t, \sigma_t^2) = f_\alpha(s_{t-1}, o, h_{t-1}).$$

Note that the output of the network μ_t is added to previous state s_{t-1} to get the mean of the Guassian. So at each step the network is really trying to parameterize a distribution for the *change in state*, i.e. change in linear velocity, angular velocity, position, and rotation.

Figure 2 outlines the details of the architecture. At each time step, the recurrent module (unrolled and labeled as "RNN" in the figure) uses the current object velocities along with the shape feature to

predict the change-in-state distributions. When generating a novel trajectory, the sampled change in velocity can be added to the current state and used as input to the recurrent module at the next step. The input shape feature o is extracted from the object point cloud using the PointNet [8] classification network. This feature is the same for every step of the sequence since it is only needed by the network to have some notion of object mass, moment of inertia, and bottom surface shape, none of which depend on time.

Note that the recurrent module can be any type of RNN. As detailed below, we found that for our task, using 3 long short-term memory (LSTM) cells stacked together produced the best results compared to a vanilla RNN, gated recurrent unit (GRU), and single LSTM cell.

4.2 Data

The proposed model is trained using direct supervision from simulated data of objects in planar sliding motion. For this we use the Bullet physics engine [1] within the Unity game engine [2]. Before each simulation, an object is chosen and randomly non-uniformly scaled between 0.5 and 1.5 to increase shape diversity in the dataset. The object is placed at rest on the surface of a ground plane and a random impulsive force that is parallel to the ground is applied to the surface of the object at the height of its center of mass. The object slides and eventually comes to rest because of friction. Throughout the sliding motion, object state (position, rotation, linear and angular velocities) is saved every 10 time steps with the physics engine running at 60 Hz. If the object happens to fall over during simulation, we throw out that data and restart the simulation.

We generate 4 different datasets each containing a single object category: `Cuboids` (1 unique object), `Cylinders` (200 unique objects), `Mugs` (37 unique objects), and `Trashcans` (47 unique objects). The `Mugs` and `Trashcans` datasets use shapes from the ShapeNet [7] repository. For each unique object in all datasets, a full point cloud containing 1024 points is sampled on the mesh surface to use as network input. Each dataset contains about 10,000 simulations. With around 20 time steps per simulation, this is roughly 200,000 time steps for training and evaluation in each shape category. Note that datasets are split into training and testing sets based on unique objects so that no evaluation objects are seen during training.

In addition to the previous 4 datasets, we also create a `Combined` dataset by simply aggregating the simulations from all 4 object categories. This creates a massive dataset with high shape variability that tests the limits of our method's shape generalization abilities.

4.3 Training

The network is trained by maximizing the log-likelihood of the training data. The loss is then the negative log-likelihood. We use truncated backpropagation to train the network on finite sequences of m timesteps and so our loss for a batch of size B is

$$\mathcal{L} = -\ell(\alpha) = -\frac{1}{B}\sum_{i=1}^{B}\sum_{j=2}^{m}\log p(s_j^i | s_{1:j-1}^i; \alpha).$$

In practice, we use $m = 15$ and $B = 64$ for training all models presented here. We optimize the objective using Adam [9] with a learning rate that starts at 0.001 and exponentially decays to 1^{-5} throughout training. Each model is trained for roughly 3 million steps.

5 Results

In this section, we detail a number of experiments which justify design decisions of our model, analyze performance on a variety of data, compare to non-RNN baselines, and show the advantage of modeling physics with a generative model.

[1] https://pybullet.org/wordpress/

[2] https://unity3d.com/

	Single, 15 steps		Single, 25 steps		Multi, 15 steps		Multi, 25 steps	
RNN Model	Position	Rotation	Position	Rotation	Position	Rotation	Position	Rotation
Basic RNN	0.1376	0.9964	0.1378	0.954	78.85	57.83	300.1	351.5
GRU	0.1244	0.7688	0.1434	1.046	10.58	38.59	15.87	55.02
LSTM-1	0.0929	**0.4494**	0.0894	**0.4428**	6.315	31.99	12.27	71.57
LSTM-3	**0.0773**	0.4607	**0.0747**	0.4478	**2.741**	**23.25**	**4.497**	**43.64**

Table 1: Model performance on `Cylinders` dataset using different recurrent modules. Position errors are shown in cm and rotation errors in degrees for both *Single* and *Multi* time step evaluation.

5.1 Evaluation Procedure

To quantitatively measure how well the model's output distribution is capturing the training data, we compare the expectation of the predicted distribution to ground truth simulation. In the following experiments we report errors for both *Single* and *Multi* time step errors. For *Single* evaluation we provide ground-truth data as input to the model at **every** step in a sequence then measure the mean absolute error between the expectation of the output model distribution and the ground truth output over all steps. For *Multi* evaluation the model is only given an initial state and then must roll out the most likely trajectory by using the expectation of the output distribution at each step as input to the next step in the sequence. The absolute error between the expectation of the **final** model output distribution and the ground truth final step is the error in this case. Throughout the presented experiments we measure *Single* and *Multi* error for sequences of 15 (same length as training), 25, and 40 time steps. All reported errors are the mean values taken over the entire dataset test split.

To qualitatively evaluate model-produced trajectories, we compare sampled trajectories from our model to ground truth simulation given the same initial conditions. In this report we show this comparison using images, however please see the video examples in the GitHub repository for a better idea of how realistic sampled trajectories appear.

5.2 Choice of Recurrent Module

We implemented our proposed architecture in TensorFlow [3]. To decide the best recurrent module (block labeled "RNN" in Figure 2), we evaluated multiple RNN variants on both the `Cuboids` and `Cylinders` datasets. Results for each model variant on the `Cylinders` dataset are shown in Table 1. We tested a single vanilla RNN cell, a single gated recurrent unit (GRU) cell [10], a single long short-term memory cell (LSTM-1), and 3 stacked LSTM cells (LSTM-3). For all variants, the recurrent cells have 128 units. Overall, the stacked LSTM reached the lowest error for both single step and long-term accuracy, predicting trajectories within 4.5 cm and 44 degrees of ground truth simulation after 25 time steps (around 4 seconds). Additionally, the consistent performance in *Single* step error moving from 15 to 25 steps shows that the model is able to effectively generalize outside of training sequence length.

Examples of sampled final state using each of the model variants are shown in Figure 3. Here we see that the low errors for LSTM-3 translate into realistic trajectories by closely matching simulation (this is even more apparent in the example videos). We found that the generated distributions from the model tend to have very low variance, so randomly sampling gives very similar trajectories every time. This is because we are using deterministic simulated data rather than noisy real-world data so the network is confident in its predictions.

In addition to the above recurrent modules, we also tried (but have not reported here) stacking basic RNN and GRU cells and using dropout between stacked cells, but saw no improvement over the LSTM-3 architecture.

5.3 LSTM-3 Performance

We further evaluate the model using the LSTM-3 recurrent module by testing on the `Mugs`, `Trashcans`, and `Combined` datasets which provide more complex and diverse shapes. *Multi*-step

[3] https://www.tensorflow.org/

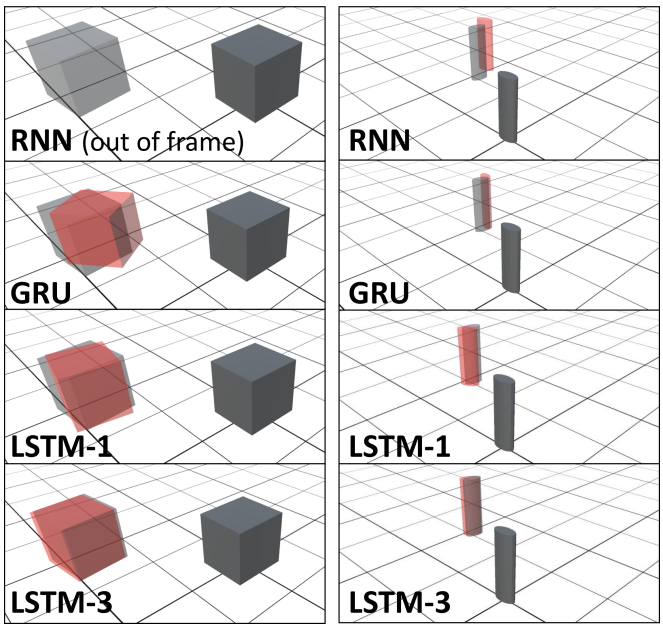

Figure 3: Sampled final object state using variants of the recurrent module for a cube (left) and cylinder (right). Dark grey is the initial object state, light gray is the simulated final state, and red is the sampled final state using the same initial conditions as in simulation. LSTM-3 gives the best qualitative performance and aligns closely with simulation data.

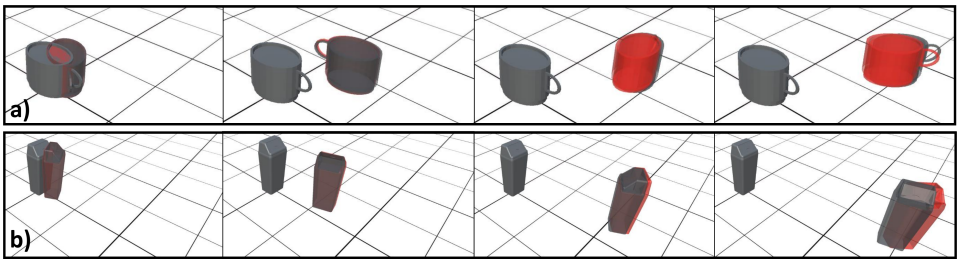

Figure 4: Model results using LSTM-3. Steps of sampled trajectories for a mug (top row **a**) and trashcan (bottom row **b**) are compared to ground truth simulation. Time increases from left to right. Dark grey is the initial object state, light gray is the simulated final state, and red is the model-sampled state using the same initial conditions as in simulation.

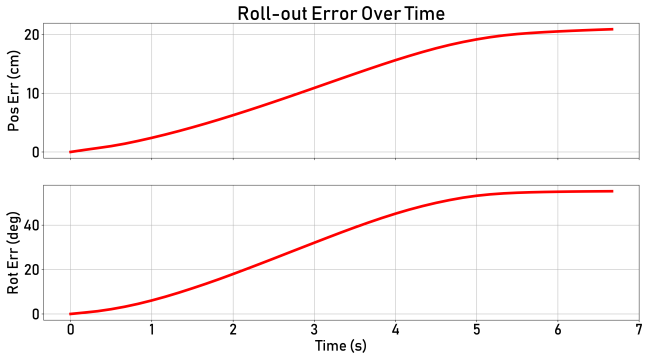

Figure 5: Mean growth in error over time as a model trained on the `Long Trashcans` dataset rolls out the most likely trajectory. The trajectory does not diverge as error increases, growing linearly at worst.

	Multi, 15 steps		Multi, 25 steps	
Dataset	Position	Rotation	Position	Rotation
Mugs	3.360	17.23	5.291	32.22
Trashcans	4.028	11.18	6.605	18.92
Combined	2.914	16.56	4.780	30.25

Table 2: Model performance using the LSTM-3 recurrent module on more complex datasets. Position errors are shown in cm and rotation errors in degrees for *Multi* time step evaluation.

	Final State	
RNN Model	Position	Rotation
No RNN	12.95	38.76
MLP	8.338	39.45
LSTM-3	**4.780**	**30.25**

Table 3: Comparison of our model using an LSTM-3 recurrent module to two baselines on the `Combined` dataset. Shown are position (cm) and rotation (degrees) errors between the predicted **final** object state (the rolled out mean trajectory for models with a recurrent module) and ground truth simulation.

errors for these datasets are shown in Table 2. Despite the complicated geometry and wide shape diversity, the model is able to generate trajectories incredibly similar to ground truth simulation. Snapshots from sampled trajectories compared to simulation are shown in Figure 4 (additionally see example videos). This performance indicates that it is effectively extracting useful shape features to describe physical properties that affect object dynamics. Furthermore, it's doing so in a generalizable way that works for a wide variety of shapes and objects that were never seen in training.

To evaluate the divergence of modeled trajectories from ground truth over large time horizons, we generated a new `Long Trashcans` dataset with larger applied impulses resulting in trajectories of around 40 steps (7 seconds). The model was trained on this new dataset, still using 15 step sequences for truncated backpropagation. When sampled trajectories begin to deviate greatly from ground truth simulation, we might expect performance to exponentially degrade because object state could vary greatly from that seen in training - an undesirable quality. However, we see in Figure 5 that this is not the case for our model in position or rotation predictions. The plot shows the error as the model rolls out the most likely trajectory defined by the expectation of the predicted distributions at each step. At worst (around 3 seconds), the error seems to grow linearly with time but quickly levels out as the sliding object comes to rest (around 5-6 seconds).

5.4 Baseline Comparison

We compare our model to two baselines. The first (*No RNN*) baseline model contains no recurrent module at all, rather it directly regresses the final rest position and rotation of the object when it comes to rest after sliding given the initial conditions for a simulation. This tests whether having to roll out the model and make predictions at each time step is much more inaccurate than direct prediction if we care primarily about final state accuracy. Secondly, the physical state of an object at some time step depends only on the state at the previous time; the velocity or position 5 or 10 steps prior has no effect on the current state. So it's not immediately clear that the "memory" inherent to an RNN is really necessary to generate accurate trajectories. We test this by comparing to the *MLP* baseline which uses a simple multilayer perceptron (MLP) with 4 fully-connected layers with 128 nodes as the recurrent module. Note that this implementation has no aggregated hidden state, so distribution predictions at each step are independent of each other.

Our model was evaluated against the baselines on the `Combined` dataset and results are shown in Table 3 which compares the errors for the **final object state** when the object comes to rest in a simulation. For our model and the *MLP* baseline, the mean predicted trajectory is rolled out to get the final state; the *No RNN* baseline directly predicts this final state. We see that our model achieves lower position and rotation error than both baselines. This shows that breaking the task of final state

	Classification	
Data	Real	Fake
Real	90.84%	9.16%
Fake	13.63%	86.37%

Table 4: Results classifying real and fake trashcan trajectories using model-assigned likelihood.

prediction down into small steps is beneficial over direct regression. Additionally, we see that the hidden state of the LSTM-3, which gives it a sense of "memory", improves model accuracy.

5.5 Anomaly Detection

One advantage of modeling physical dynamics generatively is direct density estimation. Not only does this give a sense of confidence, it can also be used to evaluate the likelihood for known full trajectories. One interesting application of this is classifying anomalous trajectories based on likelihood, which tests a model's ability to maximize the likelihood of training data as well as assign poor likelihood to examples that differ from training. We tested our model in this way by generating a new dataset of trashcan simulations where the linear and angular velocity of the object are slightly perturbed once throughout its trajectory. By simply manually setting a likelihood threshold, we classified 4000 fake and real object trajectories. The confusion matrix results are shown in Table 4. We are able to correctly classify fake and real trajectories more than 85% of the time, although understandably the model is better at classifying real examples since this is essentially the training objective. It seems our model is not only learning to maximize the likelihood of the training data, but also able to assign poor likelihood to trajectories with only minor perturbations.

6 Conclusion and Future Work

In this project we explored the use of generative models to capture the physical dynamics of rigid objects in sliding motion. We propose a neural network architecture that can effectively extract salient physical features from object shape and use these to parameterize an object motion distribution using a recurrent module. We found that using 3 stacked LSTM layers is able to produce quantitatively and qualitatively realistic trajectories on a wide range of complex objects like trashcans and mugs. All code for the project is available on GitHub (github.com/davrempe/cs236-gen-dynamics).

We would like to apply this model to more complicated physical dynamics in future work. For example full 6 degrees of freedom motion such as tumbling when pushed and complex interactions like object-object collisions. Additionally, evaluating model performance in the real world should be a priority looking towards useful applications in robotics; we hypothesize that jumping the domain gap between simulation and the real world will be a difficult research problem on its own.

Additional Notes

Because this project is an offshoot of my current research with Prof. Leonidas Guibas, the simulation pipeline for producing data was for the most part implemented (by me) before this class. I did, however, have to modify it to output state at multiple timesteps which it could not do before.

References

[1] J. B. Hamrick, R. Pascanu, O. Vinyals, A. Ballard, N. Heess, and P. Battaglia. Imagination-based decision making with physical models in deep neural networks. *In Advances in Neural Information Processing Systems (NIPS), Intuitive Physics Workshop*, 2016.

[2] K. Fragkiadaki, P. Agrawal, S. Levine, and J. Malik. Learning visual predictive models of physics for playing billiards. *In Proceedings of the 4th International Conference on Learning Representations (ICLR)*, 2016.

[3] M. B. Chang, T. Ullman, A. Torralba, and J. B. Tenenbaum. A compositional object-based approach to learning physical dynamics. *In Proceedings of the 5th International Conference on Learning Representations (ICLR)*, 2017.

[4] P. Battaglia, R. Pascanu, M. Lai, D. J. Rezende, and K. kavukcuoglu. Interaction networks for learning about objects, relations and physics. *In Proceedings of the 30th International Conference on Neural Information Processing Systems (NIPS)*, 2016.

[5] R. Mottaghi, M. Rastegari, A. Gupta, and A. Farhadi. "what happens if..." learning to predict the effect of forces in images. *In Proceedings the 14th European Conference on Computer Vision (ECCV)*, 2016.

[6] D. Mrowca, C. Zhuang, E. Wang, N. Haber, L. Fei-Fei, J. B. Tenenbaum, and D. L. K. Yamins. Flexible neural representation for physics prediction. *In Proceedings of the 32nd International Conference on Neural Information Processing Systems (NIPS)*, 2018.

[7] A. X. Chang, T. Funkhouser, L. Guibas, P. Hanrahan, Q. Huang, Z. Li, S. Savarese, M. Savva, S. Song, H. Su, et al. Shapenet: An information-rich 3d model repository. *arXiv preprint arXiv:1512.03012*, 2015.

[8] C. R. Qi, H. Su, K. Mo, and L. J. Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. Proc. Computer Vision and Pattern Recognition (CVPR), IEEE, 1(2):4, 2017

[9] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. In International Conference for Learning Representations (ICLR), 2015.

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