Examining Interpretable Representations in the Context of Intuitive Physics

Yu (Harry) Fu
Harvard University
`fu_yu@college.havard.edu`

Abstract

Learning interpretable representations is one of the biggest challenges in modern machine learning. The ability to generate interpretable meaningful representations–depending on the task of interest–allows one to produce knowledge about the relationship between domains within data in a more intuitive way. In this work, we approach this problem in an intuitive physics context, trying to determine whether a model with a 2-dimensional latent space can learn the concept of speed and angle based on a dataset consisting of images of trajectories of a single particle. Previous work in the Harvard Vision Lab has used variational autoencoders(VAE) with fully connected encoder and decoder to reconstruct trajectory images, and has shown that the latent space can be most closely interpreted as the length and angle of the trajectory, which is undesirable. This paper's contribution include the replication of results found in the Harvard Vision Lab, and the proposal of two novel approaches to inject inductive bias into a variational autoencoder-based generative model, both of which utilize a RNN to intuitively simulate the actual data generation process.

1 Introduction

Broadly speaking, interpretable machine learning can be defined as the use of machine-learning models for the extraction of relevant knowledge about domain relationships contained in data [2]. Interpretable methods rely on the use of machine learning models to produce knowledge about the these relationships, which can then be used to synthesize new insights that can be represented by visualizations, natural language, or mathematical equations [5]. Often times, an interpretable model is preferred over an uninterpretable one because it allows one to synthesize insights and experimental iterations based on intermediate steps in the learning process.

One particular area of interest with respect to interpretability that has emerged in the past few years is intuitive physics. As humans, we have the innate ability to internalize the underlying physics in our world (without necessarily knowing their mathematical formulations) and use it to predict the future. Recent work has demonstrated that implicit beliefs about systems and their future time evolution can be modeled in a probabilistic way [3], where a model receives a set of rich dynamic displays and develops a set of beliefs that guide its prediction of the system's time evolution. The work of researchers in the past few years has explored the modeling of these mental probabilistic models through deep-learning models in order to extract high-order physical information from visual inputs [4].

Much recent work has focused on whether humans "intuit" physical concepts via an end-to-end model with intermediate representations (latent states) being black-box and uninterpretable, or meaningful and interpretable. Recent work has indicated that indeed, humans develop interpretable intermediate representations and many labs have since then attempted to replicate this interpretability in representations in deep learning models. One particular notable example of this is the interpretable intuitive physics encoder-decoder network built at Carnegie Mellon University to predict the time evolution of colliding objects [6]. This interpretable learning approach was also the inspiration for

33rd Conference on Neural Information Processing Systems (NeurIPS 2019), Vancouver, Canada.

our project: that is, can we develop a network that learns the relevant intuitive physics of our system while also developing interpretable intermedediate representations?

The physical system we chose our model to learn was that of a ballistic trajectory: a particle that is launched at a speed v_0 and at angle θ, without any active propulsion or air resistance while in flight. Note that the tuple (v_0, θ) and the initial position of our particle wholly determine the particle's trajectory. The Newtonian equations of motion for this system (thrown from the origin) can be described as:

$$\{x(t), y(t)\} = \{v_0 t \cos(\theta), v_0 t \sin(\theta) - \tfrac{1}{2} g t^2\}$$

Based on initial results within the lab, it becomes clear that after training an autoencoder to reconstruct these trajectories using a 2-dimensional bottle neck, rather than learning speed and angle, the neural network learns latent variables that approximately correspond to some notion of curvature and length. We believe that the reason the model is unable to learn the underlying physics behind the data is because it has no knowledge of the underlying generative model used to create the dataset. In more concrete vocabulary, it lacks sufficient inductive bias, which can be broadly defined as the set of assumptions that a learner uses to predict outputs from inputs that it has not seen before [1].

We hypothesize that if we can find a way to introduce such inductive bias about how the trajectory data is generated, the model would have information about the generation of data and hence the 2-dimensional bottle neck would have a higher chance of learning speed and angle.

2 Data generation

We generate the trajectory dataset as follows:

1. We define the bottom left corner of each image to be the origin, and denote the horizontal distance, vertical distance, horizontal speed and vertical speed with x, y, v_x, v_y respectively. For each trajectory, we generate those four initial parameters, and store those parameters, which are exported as a csv file. For simplicity, we set x and y both to be zero just for now, and hence only v_y and v_x are random. More specifically, we have $v_x = 20 + \text{Uniform}(0, 50)$ and $v_y = 30 + \text{Uniform}(0, 100)$ with pixel as unit.
2. For each time step, we calculate acceleration in both horizontal and vertical directions and update v_x and v_y accordingly. In particular, we factor in gravity by updating v_y with dt multiplied by the gravitational acceleration, which is a adjustable parameter set at -9.8. Then we update position according to the new v_x and v_y.
3. The sequence of position tuples at each time step is then plotted to generate a trajectory. Each image is resized to 28 times 28.

For simplicity, we require the trajectory generation process to stop once the trajectory reaches the boundaries of the image. Also note that the effect of gravity is immediate and is set to the value $g = -9.8$ (the value of acceleration on Earth). Here are some examples of the trajectories generated:

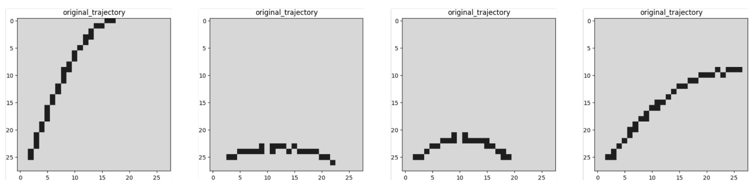

Figure 1: Sample images of generated trajectories

3 Models

3.1 Approach 0: Variational autoencoder without inductive bias

We first demonstrate that when using a variational autoencoder without any sort of inductive bias, the model's two dimensional latent space does not correspond to any notion of speed and angle, and instead is represented by length and curvature. We implement this variational encoder in TensorFlow, creating an sklearn-like interface that allows our trained model to reconstruct unseen input, to generate new samples, and to map inputs to the latent space. We trained our model with minibatching, using an Adam optimizer and the canonical Kullback Leibler divergence loss function for a variational autoencoder. Because we are concerned with whether the autoencoder can recover speed and angle, we fix the dimension of our embedding layer at 2.

It is implemented rather naively, as it treats images as a 1-D vector and uses 2 fully connected hidden layers for the encoder and decoder respectively instead of using any CNN technique on image data. The encoder of the VAE outputs an normal distribution for hidden state z. The decoder outputs a value between 0 and 1 for each pixel position of the $28 * 28$ image. Reconstruction loss is calculated to be the negative log-likelihood of input x based on the per-pixel Bernoulli distribution with the reconstructed x_{re} as probabilities. This combined with the latent loss, which is the KL-divergence, fully defines the objective of optimizer of the model. Our implementation also provides four basic VAE functionalities, including fitting of model with mini-batch, encoding of new image x_{new} to its hidden state, decoding of new hidden state z_{new} to generate a new image, and reconstruction of an image through encoding followed by decoding.

3.2 Approach 1: RNN-Autoencoder

In order to include the data generation process into training, we train the RNN on the sequence of partial trajectories. Each data point consists of some number of partial trajectories, where this number is adjustable, but set to 8 in this study. An example of a full generation of partial trajectories is shown in Figure 2.

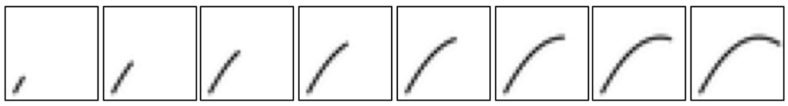
Figure 2: 8 partial trajectories for 1 data point

Each object in the Dataloader is a tuple. The first element is a data tensor that puts the partial trajectories side by side as a three dimensional tensor, and the second element is a one dimensional tensor containing true parameters used to generate the trajectory. Currently, the true parameter part is ignored when we load data into our RNN-autoencoder.

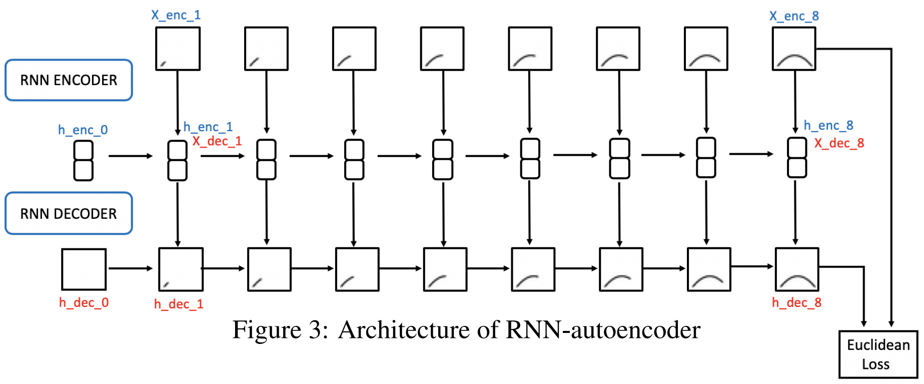

Figure 3: Architecture of RNN-autoencoder

The encoder part of our autoencoderhis moment and the is a torch.nn.RNN module that has input dimension equal to the image size and hidden dimension equals to 2 and number of layers equal to 1. We set the hidden dimension equal to 2, hoping that it has a high correlation with angle and velocity, which are the true parameters. We pass the data tensor into this RNN, which means at each step, the RNN only sees one slice as a partial trajectory in the data tensor. At each time step, one partial trajectory and hidden state will generate the next hidden state through the RNN encoder. The encoder RNN will output a tuple. The first element is a two dimensional history of hidden states at all 8 time steps (ignoring batch size), the second element is the final hidden state. We take the history of the hidden state tensor as our input for the decoder RNN.

The decoder RNN has input dimension equal to 2, and hidden dimension equal to the image size. After feeding in the history of hidden states of the encoder RNN into the decoder RNN, we take the second output, which is the final hidden state of the decoder RNN as our reconstructed image. A full schematic diagram of our autoencoder is shown in figure 3.

3.2.1 Testing correspondence: hidden state alignment with true parameters

Disregarding how well the reconstructions from the RNN-autoencoder are, a natural question would be: would the reconstruction be better if the hidden states from the RNN encoder actually have information about the true generating parameters? That is, if the latent space corresponds well with the true parameters, we would naturally expect that the reconstruction loss is minimized when we pass in the true parameters as the hidden state into the RNN decoder. Toward this end, we pass in the true parameters with the data, and sum the Euclidean distance between the two dimensional true parameters and the two dimensional hidden state of the RNN encoder at each time step into the loss function. We compare the true parameter to hidden state at each time step because the true parameters stays the same for all time steps. By minimizing the loss function, we are forcing the hidden states to align with the true parameters. The final loss function can be written as follows:

$$\mathcal{L} = \sum_{i=1}^{n} \left(||I_i - \text{AE}(I_i)||^2 + \sum_{t=1}^{T} ||\text{h\_enc}_t - \theta_i||^2 \right)$$

Where i denotes the ith data point in the data loader, I_i is the full trajectory of the ith data, $\text{AE}(I_i)$ is the reconstructed trajectory by the autoencoder, h_enc_t is the two dimensional vector of the hidden state of the encoder at time step t, and θ_i denotes the true parameters used in generating the ith data point's trajectory, which is two dimensional.

In a word, if better reconstruction is obtained, we could conclude that the hidden state of the trained RNN-autoencoder has high correlation with the true parameters. Although this test requires retraining of the entire RNN-autoencoder and thus is not directly comparable to the hidden state of a RNN-autoencoder without hidden state alignment, it should still be instructive about the nature of the latent space.

3.3 Approach 2: RNN variational autoencoder with sliced trajectory

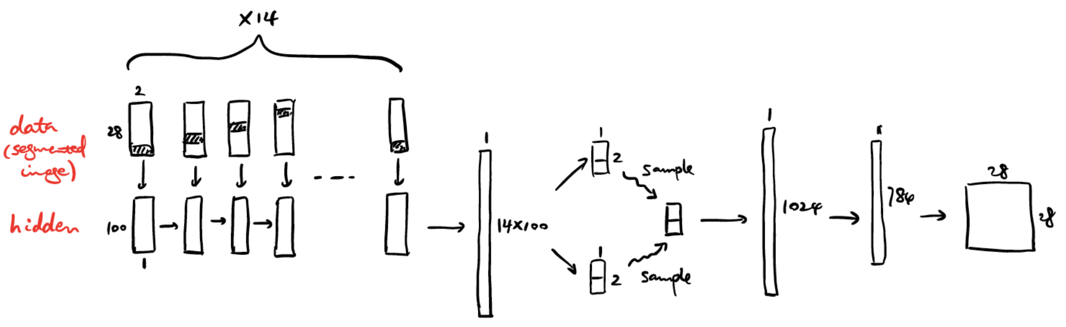

Figure 4: Schematic of VAE with RNN decoder

We slice each data point, which is a 28×28 image of a trajectory, into a sequence of 14 segments of size 28×2, transforming an image into longitudinal data. The rationale is that such slice would give the positional information of the projectile at one time frame. We then pass the image through a RNN model with input size 28×2 and hidden size 100×1. We then take the history of hidden states of the RNN, concatenate them and reshape into a vector of length 1400, and then pass it through two separate fully connected layers to two 2×1 vectors. To get to the two dimensional latent space, we sample the first dimension from the first vector with its two entries as the mean and standard deviation of a Gaussian distribution, and we similarly sample the second dimension of the latent space. Then for the decoder part, the two dimensional latent vector is passed through two fully connected layers to become a 784 dimensional vector, with the value of each entry corresponding to the probability of the corresponding pixel value in the reshaped 28×28 reconstruction image, modeled as a Bernoulli distribution for each pixel. We uses ELBO as the loss and we learn the distribution of latent space by variational inference.

4 Results and discussion

4.1 Variational autoencoder without inductive bias

Here we replicate the results originally indicated by research done in the Harvard Vision Lab: namely, that simply feeding in our trajectories to a variational autoencoder without any form of inductive bias is insufficient for the model to learn concepts of speed and angle. We first show some examples of the reconstructed images:

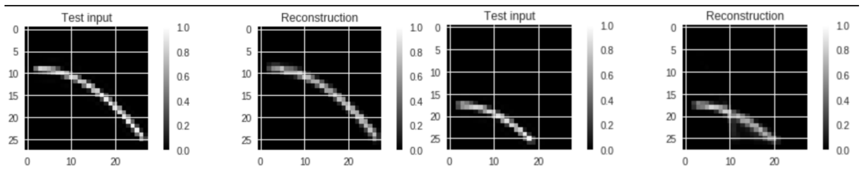

Figure 5: Reconstruction of VAE

We can clearly see here that the VAE appears to do a relatively good job of reconstructing our images, irrespective of the lack of inductive bias. However, when we investigate the reconstructed trajectories as a function of latent space shown in figure 6, it becomes apparent that the two dimensions do not correspond to speed and angle.

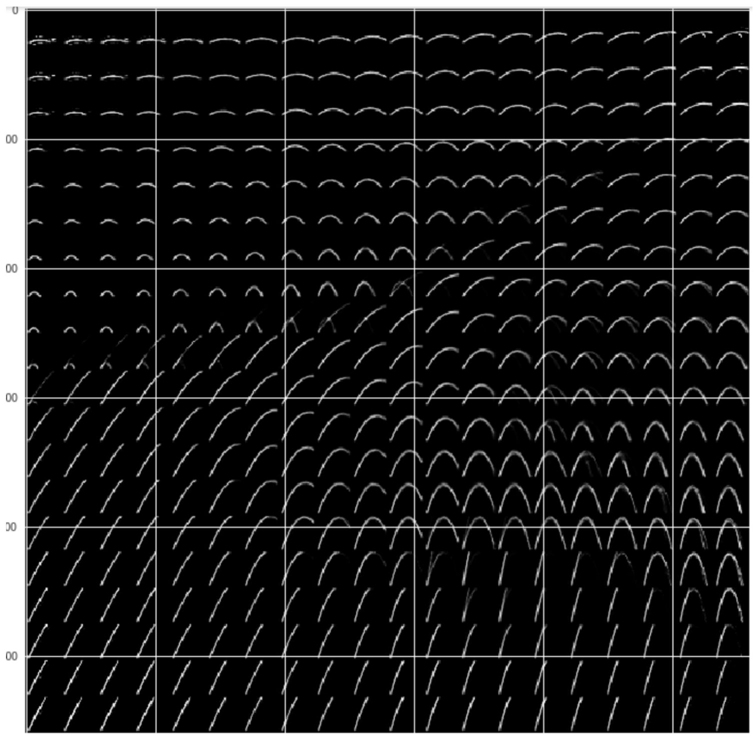

Figure 6: Visualization of reconstructions based on latent space

Not only do the actual values of the latent variables correspond to speed and angle, but when examining the visualization of our latent space, we observe that intuitively, the vertical latent variable approximately corresponds to a lengthening/shortening of the trajectory and the horizontal latent variable approximately corresponds to increases/decreases in curvature. As stated before, we believe that the reason our autoencoder is unable to recover speed and angle is because it lacks the type of inductive bias required to properly learn these physical concepts. Thus, we propose two forms of inductive bias below which we believe might allow our model to learn an interpretable 2D representation of our trajectories that corresponds to speed and angle.

4.2 RNN-autoencoder with partial trajectory

4.2.1 Naive reconstruction loss

We set batch size to 4, and for each data point in data loader, the partial trajectory tensor is $8 \times 4 \times 28 \times 28$ that puts the 8 partial trajectories side by side, and the second element is a 4×2 tensor that contains the true parameters used to generate the trajectories. The encoder RNN has input dimension equals to $28 \times 28 = 784$ and hidden dimension equals to 2 and number of layers equal to 1, and it outputs an $8 \times 4 \times 2$ tensor that has the history of hidden dimension at all 8 times steps, as well as the final hidden state.

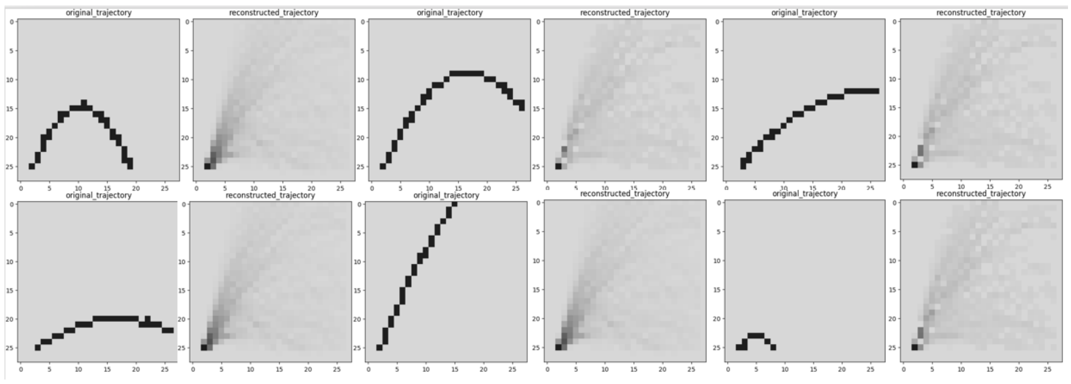

Figure 7: Examples of reconstruction for RNN-autoencoder trained on only reconstruction loss

The decoder RNN has input dimension equals to 2, and hidden dimension equals to 784. We trained the RNN-autoencoder for 500 epochs with SGD optimizer with learning rate equals to 0.01. Some examples of reconstruction are shown in figure 7 above. The reconstruction loss converges at around 1000 at 50 epochs and the reconstruction performance is not good. Hence we proceed to the next stage to see whether passing in information about the true parameters to the hidden state would help reconstruction in any way. The loss is the Euclidean distance between this reconstructed image and the original full trajectory. Note that because the loss is only calculated between the final hidden state and the original image and have no contribution from the hidden state of RNN decoder at previous time steps, we do not expect the sequence of hidden states of decoder RNN at each time step to assemble a "partial trajectory".

4.2.2 Loss with hidden state aligned with true parameters

As discussed in section 2.3, we incorporate the difference with respect to true parameters into the loss function. Some examples of reconstruction are shown in Figure 8. Note that the magnitude of the loss in this case is significantly larger than the last model, since new losses are summed into the total loss, and subsequently not directly comparable. However, the reconstruction of this model is still not ideal.

The reconstruction quality is about the same as in the case with only reconstruction loss, and it seems that the reconstructions are rather similar irrespective of the shape of original image. The left bottom seems to always always be a rather dense dot with high pixel intensity, and the spread of intensities assembles a wide collection of trajectories.

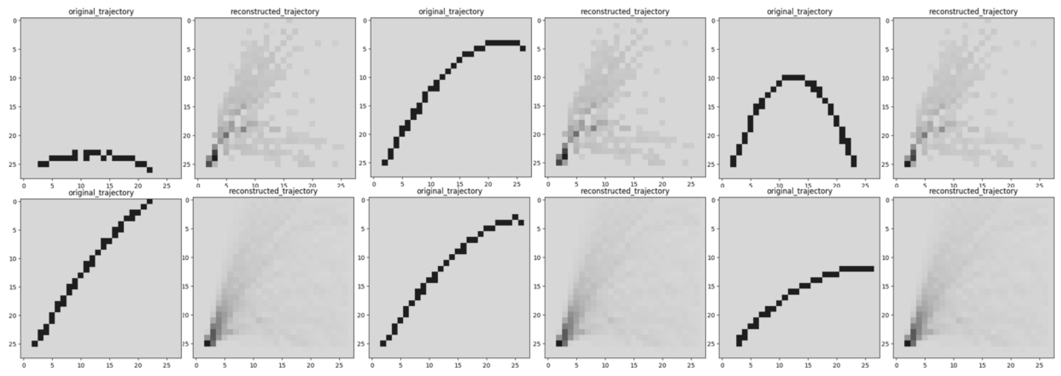

Figure 8: Examples of reconstruction for RNN-autoencoder trained on both reconstruction loss and hidden state alignment

4.2.3 Extrapolating latent space of RNN-auto-encoder

We use a trained RNN-autoencoder with only reconstruction loss and extrapolate its latent space with the same method as in Figure 6. Both horizontal axis and the vertical axis ranges from -2 to 2, with 0.05 as step size. This generates 400 reconstructed images as shown in figure 9.
Firstly, this latent space extrapolation result in figure 9 is vastly different from figure 6. Each little reconstruction image does not correspond well with a clearly defined trajectory. Instead, it seems that there is a ground optimum reconstruction image, which is a combination of all possible trajectories that showed up in the dataset. This optimum image should correspond to both latent space dimensions equal to 0, which is the center of this extrapolation. We hypothesize that this optimum image is a weighted average of all possible trajectories with their appearance frequency in the dataset as the weight.

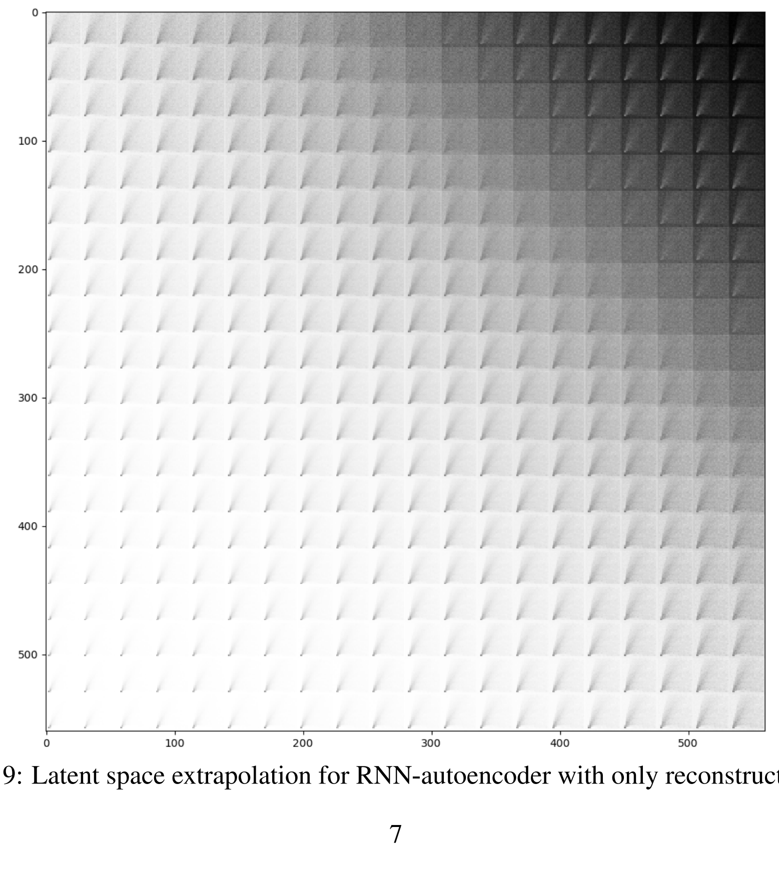

Figure 9: Latent space extrapolation for RNN-autoencoder with only reconstruction loss

We can conclude that the two latent dimension have the same role in controlling the reconstructed image because this latent extrapolation is symmetrical about the diagonal going from bottom left to upper right. Moreover, the latent space value correspond to a weight multiplied to the optimum image, which sits the middle, to decide how much of the combined trajectories should show up in the reconstructed image. Together with the hidden state alignment test from section 3.2.2 and its result from 4.2.2, we thus conclude that the latent spaces of the RNN-autoencoder correspond poorly with angle and velocity, which are the true parameters used to generate the trajectories.

Intriguingly, as seen from the extrapolation, when both latent dimensions have a value larger than about 0.75, the background and the reconstructed image seem to switch place: the left bottom now has low intensities and is shown as white, where the other parts of the reconstructions have high intensities. There should be no real trajectory data that would give these sets of latent dimension values since it would then contribute a large loss. Instead, these "reversed" reconstructions seem to be a side effect of the decoder parameters when we try to extrapolate the latent space linearly.

4.3 RNN-VAE with sliced trajectory

4.3.1 Reconstruction quality

We first generate dataset according to section 2 and slice each image into 14 slices. With 2000 training image sets and learning rate of 0.001, the model converges at around 180 epochs with an average loss of 56.01. We also test the model on a test set of 400 trajectories to make sure that the model does not overfit. The test loss is around 59.

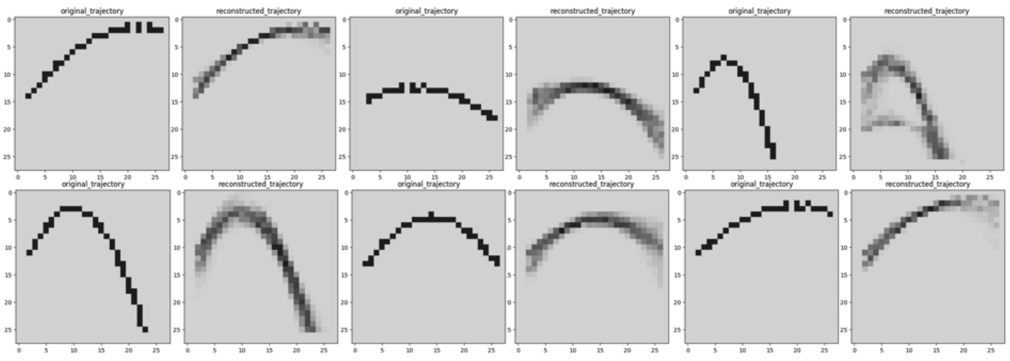

Figure 10: Reconstructions of RNN-VAE

Some examples of reconstruction are shown in figure 10. Each rows shows three sets of trajectories, where the image on the left is original trajectory, and on the right is the reconstructed trajectory by RNN-VAE. We can see that the reconstruction quality is fairly good. The last example in the first row was presented with the intention to show that very few reconstructions shown similar superposition of two trajectories, but we also chose the first example in the second row deliberately to show that very similar trajectories can be reconstructed relatively well with no such superposition.

4.3.2 Latent space extrapolation for RNN-VAE

Similarly, latent space was extrapolated between -2 and 2 for both dimensions with step size of 0.05 as shown in figure 6. For now, only by observation, it is not immediately obvious how angle changes across the x or y axis, but more to the bottom, the parabola is more curved, which indicates a low velocity in both horizontal and vertical direction. Because if the horizontal velocity were high, the object would fly out of canvas in short amount of time ticks. With constant vertical acceleration, the projectile would present itself to be flat.

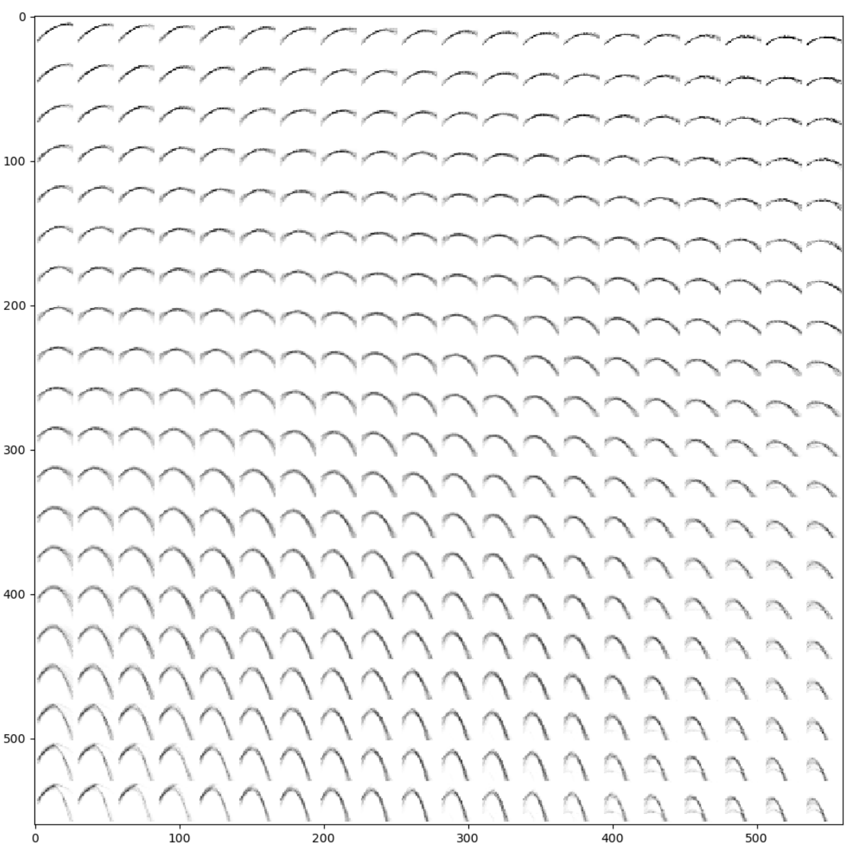

Figure 11: Reconstructions of RNN-VAE

But the analysis above does not simply mean that lack of curvature is due to low velocity. We see that RNN-VAE latent space in figure 11 is clearly much more structured than the latent space of naive VAE with fully connected encoders shown in figure 6.

4.3.3 Analysing correspondence

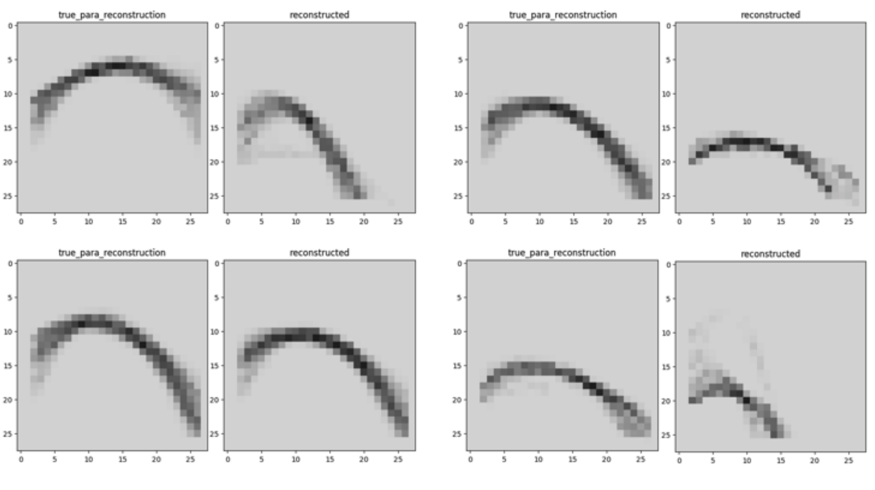

Figure 12: Left: reconstruction with true parameter. Right: reconstruction with trajectory image

We standardized the true parameters, angle and velocity, by centering them and divide them by corresponding standard deviations. We then reconstruct images by passing in the true values as latent vector to the decoder, as well as reconstruct images by passing in the trajectory image into the entire VAE. Some examples of comparison are shown in figure 12 below. Each row has two sets of comparisons, where within each set, the the image on the left is reconstruction by passing in true parameter, and image on the left is reconstruction with sliced trajectory image.

We also calculated the correlation between the latent space and true parameters for both dimensions, but the two correlations are both below 0.1 with very high p values. Hence the correspondence between the latent space and the true parameters are not very evident. We think that maybe by non-linear methods, like fitting a GLM, would eventually find a good correspondence, but we question the value of such non-linear relationship as it seems to defy our purpose of learning the concepts of angle and velocity directly.

5 Conclusion

The fundamental problem we sought to address in this work was that of interpretable representation in an intuitive physics context: that is, can we inject an inductive bias into our model such that it can "intuitively" learn a 2D representation of ballistic trajectories that corresponds to speed and angle? We first demonstrated that without injecting any inductive bias, the model is unable to learn a representation corresponding to speed and angle using a simple variational autoencoder with fully connected encoders and decoders. We then proposed to use an autoencoder, with both the encoder and decoder as RNNs, to train on partial trajectories, in order to incorporate the data generation process into training as the inductive bias. However, the reconstruction quality is not good. By guiding hidden state of encoder RNN with the true trajectory generating parameters and extrapolating the latent space, we can conclude that the two dimensional latent space of our RNN-autoencoder still corresponds poorly to speed and angle. Then, by using RNN as encoders and decoder in an VAE, we are able to introduce the temporal information into VAE and introduce inductive bias about how the image is generated. The reconstruction quality is good and the latent space we get from RNN-VAE is clearly more structure and more interpretable. Currently we tried two method of measuring how well these two latent spaces corresponds to the true parameters, namely by comparing their individual reconstructions and calculate pearson correlation, but they do not indicate a very well correspondence, and this approach so far did not work well in recovering the generating parameters for the trajectories.

References

[1] Nadav Cohen and Amnon Shashua. Inductive bias of deep convolutional networks through pooling geometry. *arXiv.org*, 2017.

[2] Finale Doshi-Velez and Been Kim. Towards a rigorous science of interpretable machine learning. *arXiv preprint arXiv:1702.08608*, 2017.

[3] Jessica Hamrick, Peter Battaglia, and Josh Tenenbaum. Probabilistic internal physics models guide judgments about object dynamics. 2011.

[4] James R Kubricht, Keith J Holyoak, and Hongjing Lu. Intuitive physics: Current research and controversies. 21(10):749–759, 2017.

[5] W James Murdoch, Chandan Singh, Karl Kumbier, Reza Abbasi-Asl, and Bin Yu. Definitions, methods, and applications in interpretable machine learning. *Proceedings of the National Academy of Sciences of the United States of America*, 116(44):22071–22080, 2019.

[6] Tian Ye, Xiaolong Wang, James Davidson, and Abhinav Gupta. Interpretable intuitive physics model. 2018.