Report by Bhushan as a summary of findings in Fall 22 semester for Agile Systems Lab under the guidance of Prof. Sponberg and Usama.

Objective:

Determination of causal latent factors in a pretrained Recurrent Neural Network (RNN) using the Generative Causal Explanations (GCE) framework.

Background:

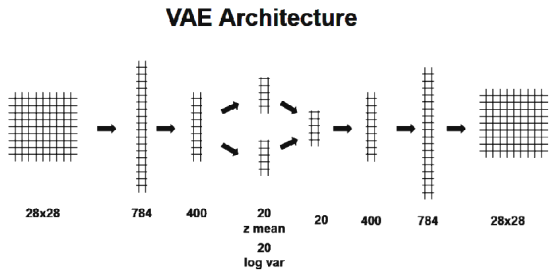
A recurrent neural network was already trained to successfully predict muscle spikes and spike timings in hawkmoth flight experiments. In a series of experiments, moths were tethered and made to respond to a stimulus of a motion of a flower oscillating at certain frequencies. EMG muscle spikes were recorded directly from their muscles to use as training data. The flower motion was simulated and an Event Polarity Matrix sequence was generated from the simulated frames. This sequence was passed through an autoencoder to be fed into the RNN. The trained RNN outputs muscle spikes and spike timings for 44 different muscles.

The GCE framework was referenced from the paper O'Shaughnessy et al. It helps bias the encoded latent factors to influence the output of a specific classifier. The original GCE framework has been tested on the standard MNIST dataset. It encodes the digit frames into a latent space and then biases the latent space such that sweeping across certain dimensions in the latent space causes changes in the digits in a way that changes the classifier output. While sweeping across some other dimensions only leads to changes in other stylistic factors like skew or thickness of the stroke.

The idea was to modify the GCE framework to fit the RNN into the pipeline. This could help us determine the causal factors in the encoded latent space for this pretrained RNN.

GCE Overview:

https://github.com/dragen1860/pytorch-mnist-vae
https://github.com/siplab-gt/generative-causal-explanations
https://arxiv.org/pdf/2006.13913.pdf

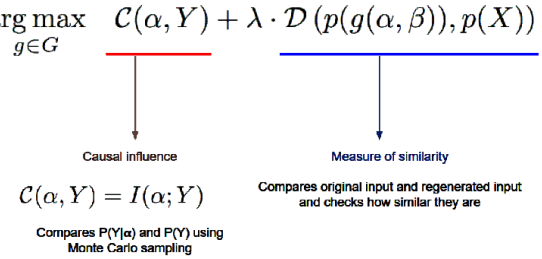


The GCE framework uses a Convolutional Variational AutoEncoder (CVAE) to encode the frames into a latent space. This is done using some initial convolutional and pooling layers and then linear layers in a neural network. The last encoder layer is split into 2 subparts which represent the mean and log variance of the datapoint. Every frame gets encoded into a mean and log var values in the latent space. Then it generates a new datapoint from the normal distribution of these values and then decodes the values using another decoder neural network. This consists of layers exactly inverse of that of the encoder layer. This reconstructs the frame back up again.

A loss function is declared as the difference between the initial input frame and the generated output frame. The neural network is then trained to minimize this loss thus ensuring the initial input frame is as similar to the output frame. Eventually after some iterations, the weights are trained so that each different class in the dataset gets encoded into their own normal distributions in the latent space. Each of the decoder network weights are also trained simultaneously so that a frame similar to the input frame is generated from this normal distribution. The dimensions of this normal distribution could represent different features of the datapoint which may or may not be semantically important. This part is the 'D' term in the loss function of the GCE framework which ensures that the generated datapoint is within the data distribution and not completely new.

Sweeping across latent dimensions

The GCE framework adds a classifier to this VAE architecture for which the causal factors in latent space need to be found. This classifier classifies the output of the VAE. Randomly generated samples from standard normal distribution are passed into the decoder and then classified via this classifier. This helps generate a measure of information transfer. Then the GCE framework adds another term called 'C' term to the loss function which shows the measure of information transfer due to the latent parameters. When the GCE is trained with this measure integrated in it, it causes the weights to be biased in a way that, sweeping certain latent parameters, results in changes to the classifier output.



Loss function

To calculate the measure of information transfer, samples are randomly generated. This helps obtain an estimate of the probability for the distribution. Then, the same is repeated by fixing values for few latent parameters that we want to bias as causal. This gives an estimate of the probability distribution given a certain value for causal latent parameter. These probabilities thus provide us with the information transfer that takes place due to the causal latent distribution.

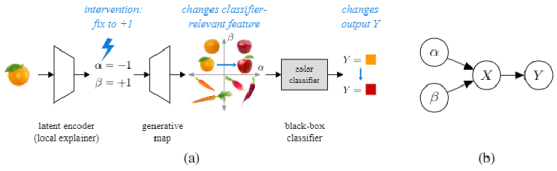


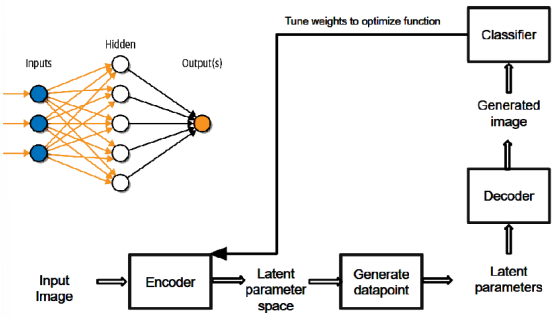
Figure 1: (a) Computational architecture used to learn explanations. Here, the low-dimensional representation (α, β) learns to describe the color and shape of inputs. Changing α (color) changes the output of the classifier, which detects the color of the data sample, while changing β (shape) does not affect the classifier output. (b) DAG describing our causal model, satisfying principles in Section 3.1.

Doing this will eventually bias the latent parameters in a way that the classifier output is affected by sweeping only certain latent parameters. This can be explained with an example from the paper.

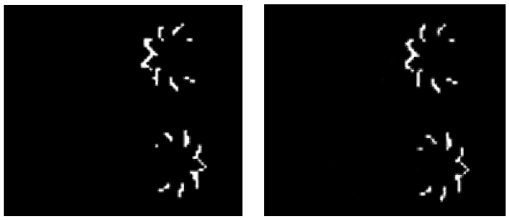
As it can be seen here, changing the latent factor alpha changes the color of the generated output. This eventually results in a change in the classifier output. However, changing beta changes features of the data irrelevant to the classifier.

Report by Bhushan as a summary of findings in Fall 22 semester for Agile Systems Lab under the guidance of Prof. Sponberg and Usama.

Implementation:

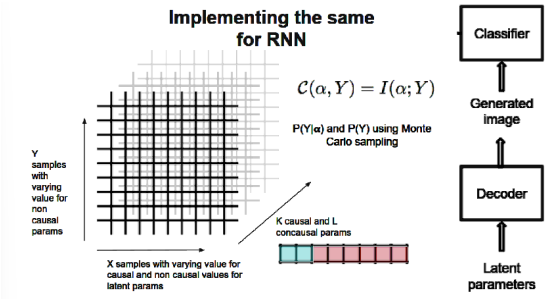


The input data in our case is an EPM sequence of 2500 frames and each frame of 84x36 frames. Each pixel has a value of 0,1,-1 depicting the difference in contrast with respect to the previous frame. Dealing with negative values can be tricky. To simplify the problem, we convert the 84x36 image into a 84x72 image stacking the same frame on top of each other. In the top frame we apply the following conversion for pixels: { 0 ----> 0, 1 ----> 1, -1 ----> 0}. For the bottom frame we apply the following conversion for pixels: { 0 ----> 0, 1 ----> 0, -1 ----> 1}. This converts the EPM sequence into one superframe which encodes both negative and positive values of the frame.



Original frame Reconstructed frame

While implementing the same for the EPM sequence of the flower stimulus, initially we tried to train the VAE using only the 'D' term to get an estimate of the number of latent parameters required to encode the frames. 6-8 latent parameters were sufficient to encode the frames. Each frame was encoded into 8 latent factors and the entire sequence of 2500 frames was regenerated from this. The regenerated sequence was similar to the input sequence except for some odd frames of which not enough data points were available to encode into the dataset.



To incorporate the causal term, Monte Carlo samples are generated with varying latent parameters to obtain the probability estimates required for measuring information transfer. In the case of MNIST dataset, this was done for individual frames which were generated and passed into the classifier to get a probability output. In our case, the classifier is a RNN which means the entire sequence would need to be passed for the classifier to make predictions.

Passing the entire sequence into the RNN would require generating a random sequence via Monte Carlo sampling. If we encode each frame into its individual latent space, this would cause the sequence to have a dimension of 20000 (2500x8) parameters. Assuming, the MNIST data required 50 samples for 6 non causal factors, it is safe to estimate that the samples required to estimate any meaningful probability with 20000 parameters would be exponentially high and impractical.

Challenges and possible approaches:

https://arxiv.org/abs/1608.06315

1)

A) As stated above one of the main challenges is to modify the GCE architecture to take a sequential input. Decoding each frame to pass it through the classifier does not require a large number of latent factor samples to be generated. However, for a sequence this increases exponentially. It can be hypothesized that indeed an entire sequence would not require so many parameters to encode it into a lower dimensional space. After all, the frames are very similar to each other. Thus all of the frame sequence could be converted into one superframe and then encoded into a fewer number of latent parameters.

A similar way to tackle the same issue is to encode all the frames into their own latent parameters. Then, re-encode this sequence of encoded parameters into a smaller latent space.

The above frame is a 125x160 image. The 125 represents the number of frames in one oscillation. Each column is a 125x8 representation of 8 latent parameters of 125 frames in one oscillation. These are mapped side by side for all 2500 frames (20 oscillations). As can be seen, the data has some pattern that can be encoded into far fewer dimensions.

The issue with this approach though is that firstly it would take a lot of such sequences to train the GCE to learn any meaningful distribution of the latent space for the sequence. Secondly, changing parameters would change nearly every frame in the sequence which might not preserve the sequential nature of the frames. This would make no semantic sense and passing such a generated output to the RNN is not useful at all.

B) Another way to possibly incorporate the sequential nature into the GCE is for every iteration just choose one fixed frame to be changed from the sequence. Regenerate the said frame via the decoder and then observe the changes in the RNN for the changed frame. The concern with this approach was if changing just one frame would change the classifier output at all.

However, this seemed like a good approach to try first and see the results. Initial tries did not show any major classifier output changes on changing just one frame.

2)

A) The output of the RNN is for multiple muscles and predicts spikes and gives spike timings as well. The information transfer metric requires a probability value to find it. However, the RNN predicts the spiking with a 0 or 1 probability at different times across the sequence. Modifying this output to properly get a probability output is key to find information transfer for the 'C' term.

B) The second output is a regression output which is even more tricky due to the continuous nature of it. Again modifying this in some way to get a probability value is required to find the information transfer.

This could be tackled by modifying the outputs as gaussian distributions in some way which would give probability values that are required. (More literature review required into this)