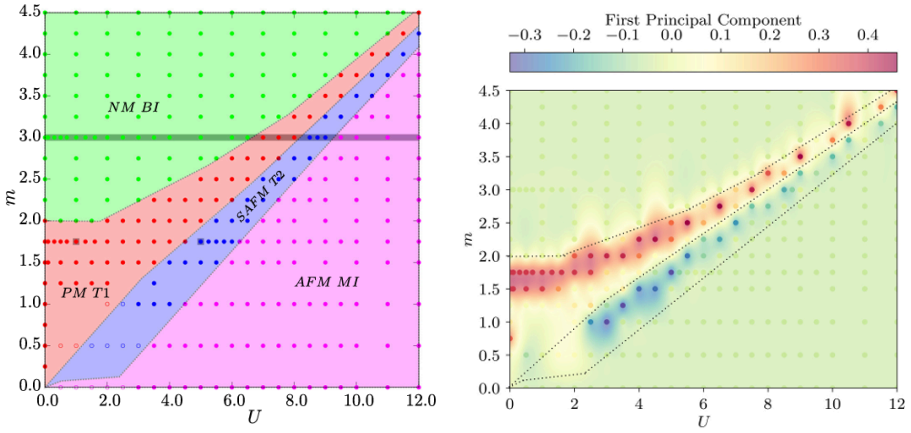
Variational Autoencoder Project Summary

My research project this semester was based on the research performed in the paper "Topological and magnetic properties of the interacting Bernevig-Hughes-Zhang model" (https://doi.org/10.1103/PhysRevB.109.245115). The interacting BHZ model has 4 phases, depending on the gap parameter (m) and the Hubbard interaction strength (U): Nonmagnetic trivial band-insulating phase (NM BI), Paramagnetic topological T1 phase (PM T1), Topological T2 phase with stripey antiferromagnetic correlations (SAFM T2), and Antiferromagnetic nontopological mott insulator phase (AFM MI). In the paper, the predicted phase diagram for BHZ model was verified using principal component analysis (PCA) on the edge charge densities (the charge density at each cell minus the average charge density of the bulk) for the orbitals of simulated 4 x 4 cylinders of different m and U values.



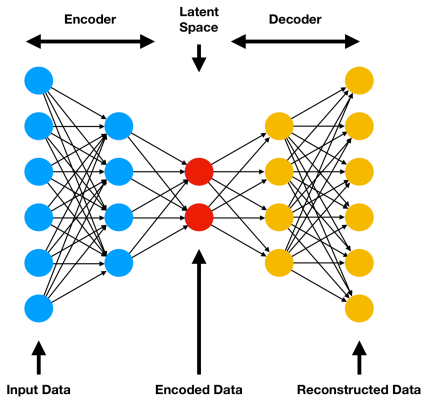
However, the PCA apparently couldn't distinguish between the NM BI and AFM MI phases. The goal of my research was to see if I could design a variational autoencoder (VAE) that could distinguish between them, and also use the generative capabilities of the VAE to generate new data.

The first thing I did was attempt to make an autoencoder and VAE for the MNIST dataset of handwritten digits, to better understand how they work. The difference between a normal

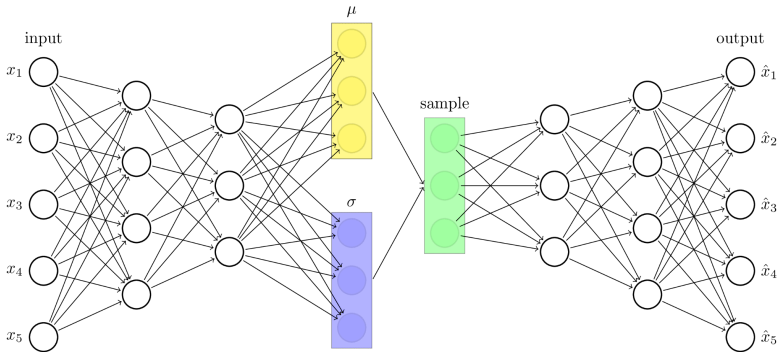
autoencoder and a variational autoencoder is that the middle neurons that you're condensing to are replaced with a probability distribution.

In practice, this meant that the VAE was constructed by first making an "encoder" whose output was two neurons that represented the mean and standard deviation of a gaussian. A function was then written that randomly sampled from this gaussian, which was then fed into the "decoder" as shown in the example diagram below:

Autoencoder



Variational Autoencoder



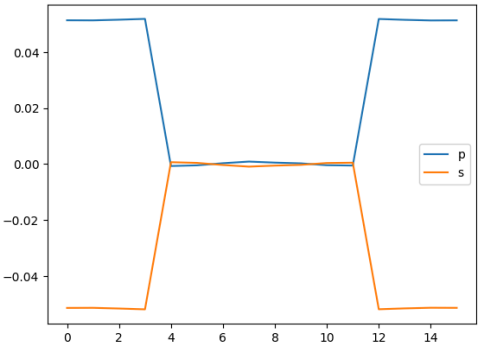
A custom loss function was constructed to allow for backpropagation between the encoder and decoder of the VAE. While the normal autoencoder was made from scratch, the VAE was based on a keras tutorial (https://blog.keras.io/building-autoencoders-in-keras.html). However, this was essentially just to provide the basic skeleton of the VAE, as modifications were made to allow for the presence of hidden layers, etc.

Ultimately, PCA, an autoencoder, and a VAE were utilized on the MNIST dataset, and new digits were generated with the VAE, as seen in the notebook "testing_on_MNIST.ipynb" in this repository.

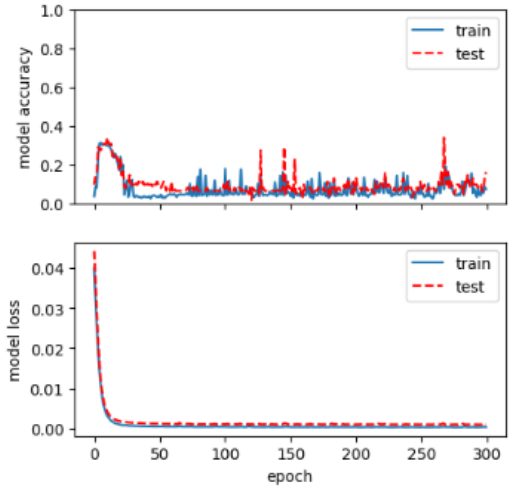
After this testing, I moved on to developing an autoencoder and VAE for the actual orbital data, which can be seen in the notebook “main_project.ipynb”. My dataset was 302 lists of orbitals, each with 32 values of the edge charge density: 16 p orbitals, then 16 s orbitals.

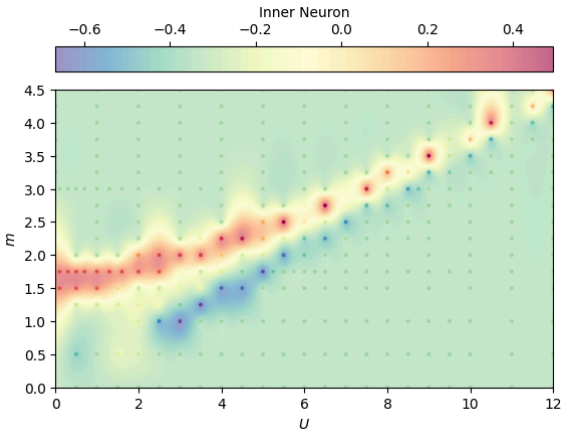
For example, for m = 2.5, U = 7.5 we have the following data:

```
array([ 0.051316,  0.051283,  0.051524,  0.051814, -0.000682, -0.000438,
        0.000315,  0.000902,  0.000532,  0.000268, -0.00039 , -0.000507,
        0.051783,  0.05148 ,  0.051255,  0.05129 , -0.051316, -0.051283,
       -0.051524, -0.051814,  0.000682,  0.000438, -0.000316, -0.000902,
       -0.000531, -0.000268,  0.00039 ,  0.000507, -0.051783, -0.051479,
       -0.051255, -0.05129 ])
```



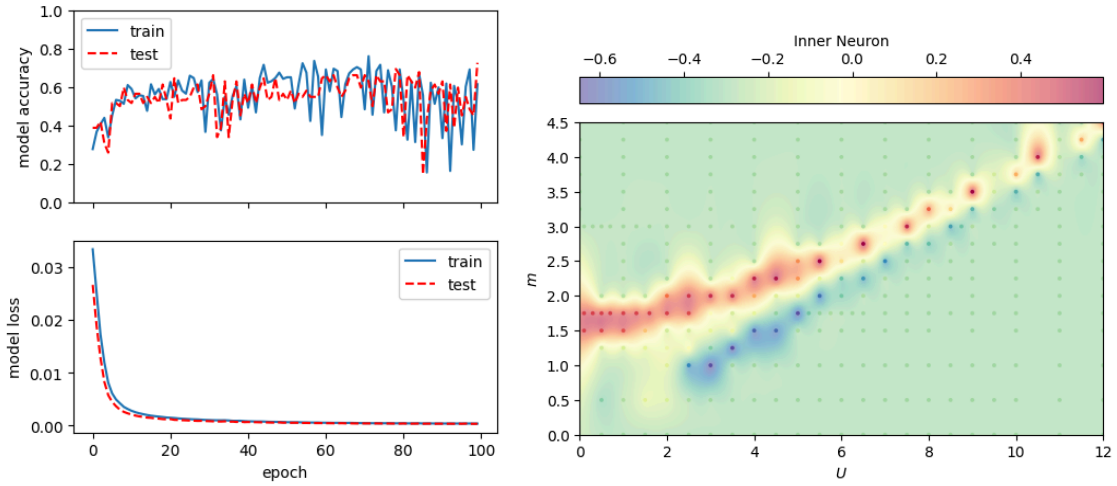
The autoencoder was constructed with 32 input neurons, 32 output neurons, and with tanh as the activation function for the output and hidden layers. The orbitals were normalized between -1 and 1, and I performed a train-test split of 20%, with the inputs x and desired outputs y identical. The loss function was mean squared error, the optimizer was ADAM, and I trained for 300 epochs with a batch size of 20.



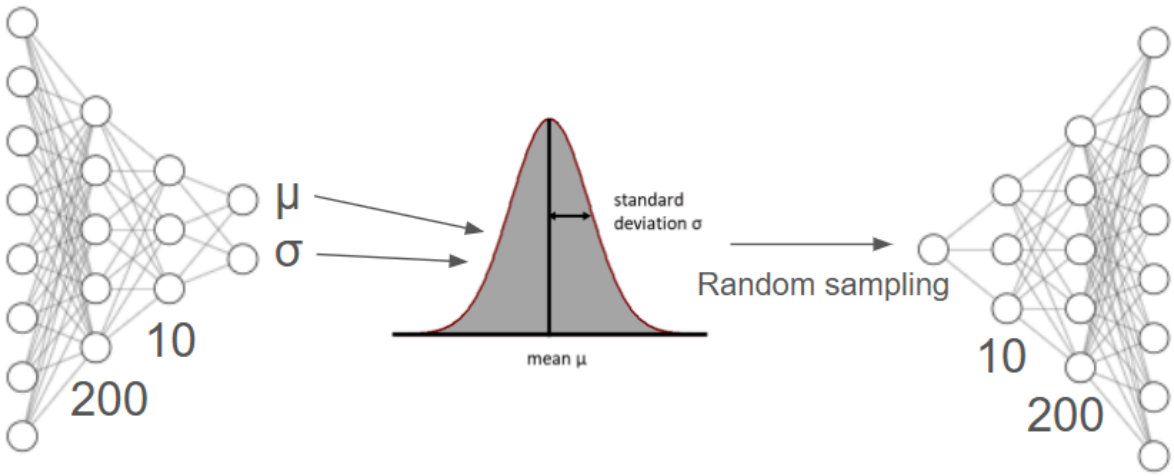


Ultimately the autoencoder achieved similar results to the PCA, and didn’t appear to have any noticeable advantage.

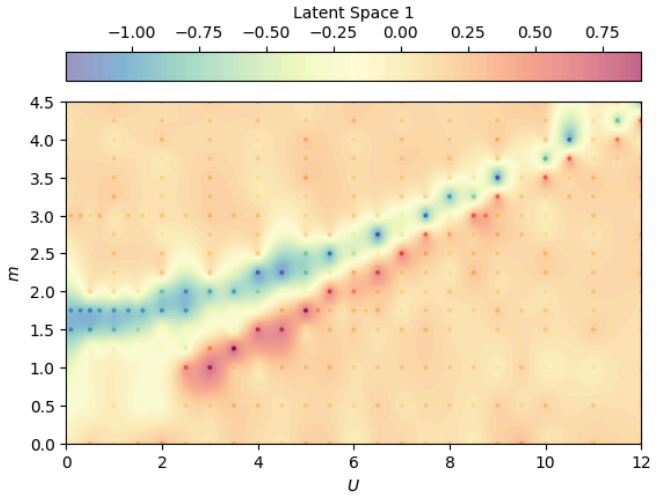
Note that since we're using cylinders, we expect the density differences around the cylinders to be mainly due to random fluctuations. To make sure the network wasn't overfitting to these fluctuations, I averaged every 4 orbitals (32 input neurons ⇒ 8 input neurons). The rest of the autoencoder's architecture was kept the same. This improved the accuracy a little.



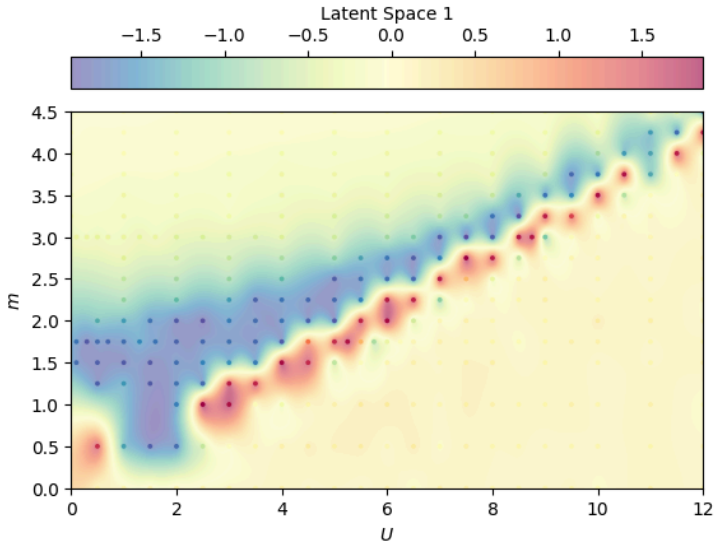
Next a VAE was made. The encoder had 8 input neurons, 200 neurons with tanh, 10 neurons with tanh, then two neurons with linear activation encode the mean and standard deviation of a gaussian. The gaussian is randomly sampled, and the value enters the decoder. This is 10 neurons, 200 neurons, then 8 output neurons with tanh. Custom backpropagation connected the encoder and decoder.



Ultimately the VAE achieved similar results to the autoencoder. The specific training history can be seen in the notebook. As seen in the figure below, a slight spottiness appeared due to the random sampling, and changes each time you sample.

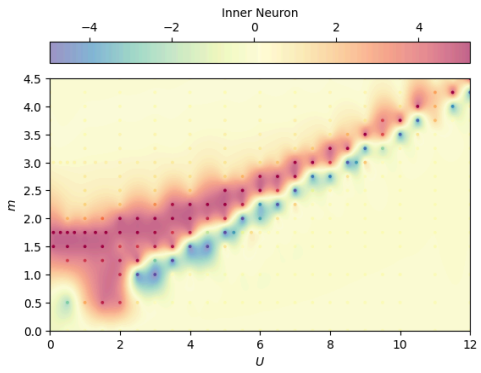


Later in my research, I tried a different loss function instead of mean squared error, the "cosine similarity," where the actual and predicted orbitals are treated as vectors, and the loss is simply the cosine between the two vectors. Interestingly, the VAE could tell apart the NM BI and AFM MI: it may be hard to see in the figure below, but the top phase is slightly more green and the bottom phase is slightly more orange.

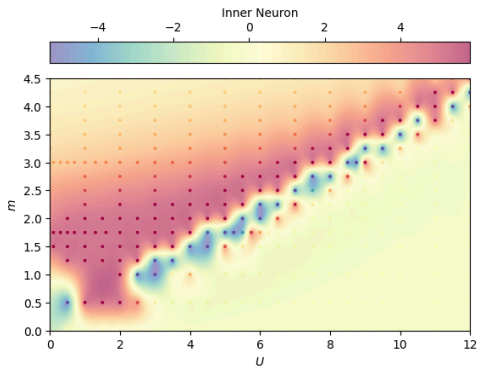


Strangely, I noticed that overtraining the network caused the results to look less like the phase diagram. The PM TI phase appears to creep into the NM BI phase, as a result of all of the predicted values becoming closer to either zero or the positive or negative extremes.

After 100 epochs:



After 600 epochs

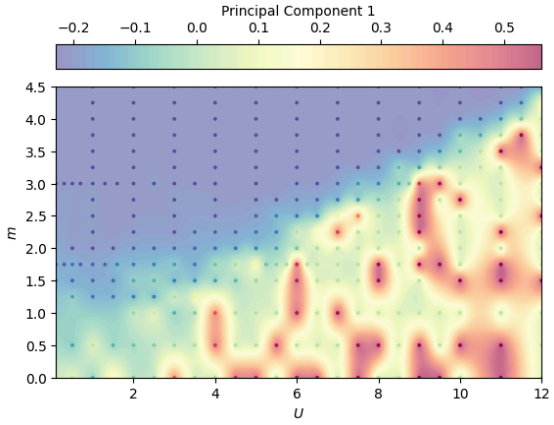


This was not something observed with the normal autoencoder or PCA, and is definitely due to training with the cosine similarity as the loss function, which treats everything as vectors.

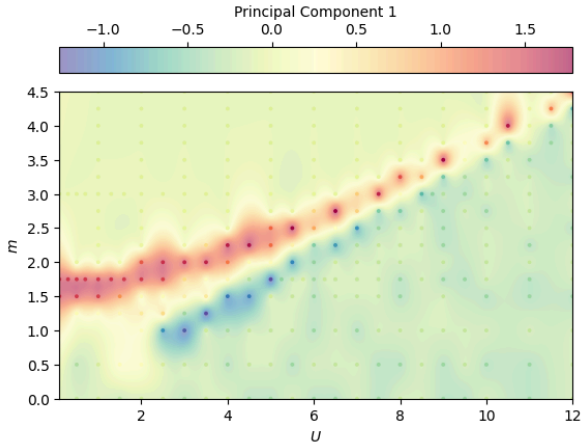
The success of the VAE indicated to me that it is not impossible to learn to tell apart the phases given only the orbitals. Hence this made me wonder if PCA was actually doing the same thing, but the difference was so small that we hadn't noticed. As it turned out, the PCA value for one phase was actually slightly higher for one phase compared to the other, so apparently it can. However, the difference is so small that it just looked like the same shade of green on the plots. The advantage of the VAE with cosine similarity is that it yields a more noticeable difference in the value between the phases.

Before trying the cosine similarity I thought the network wouldn't be enough to tell apart the two phases, so we decided to append spin spin correlations to the training data to see if it would be able to. By doing this, resolution between the two states was achieved via PCA, but inherent patchiness that resembles the result of PCA on only the correlations appears.

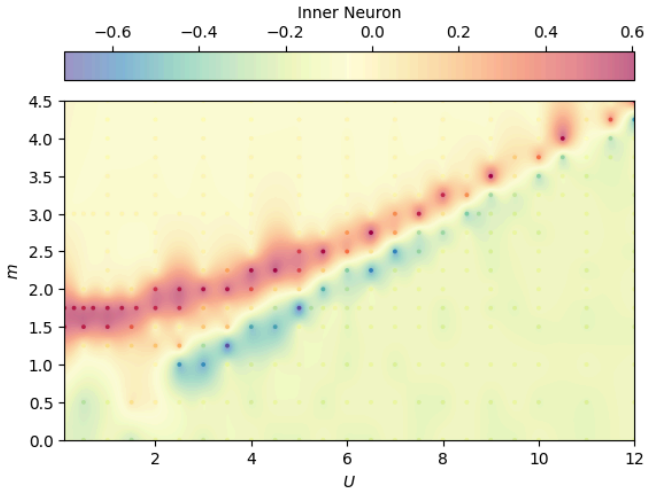
PCA on only spin spin correlations



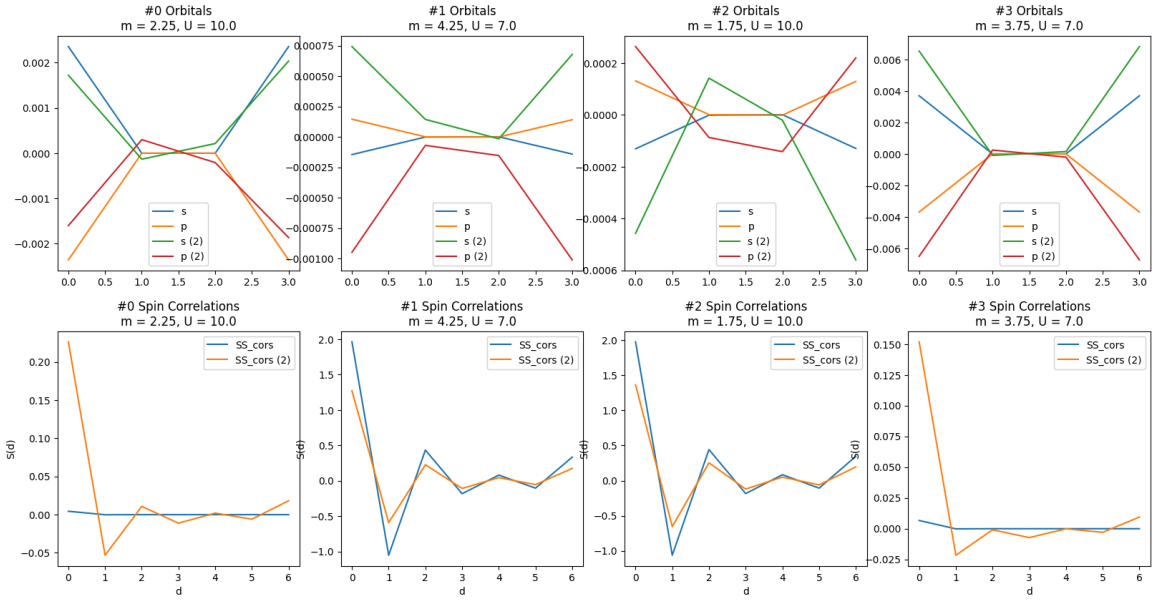
PCA on orbitals and spin spin correlation



However, later training with cosine similarity yielded much less patchiness, and probably the most clear resolution between the phases achieved during my research project:

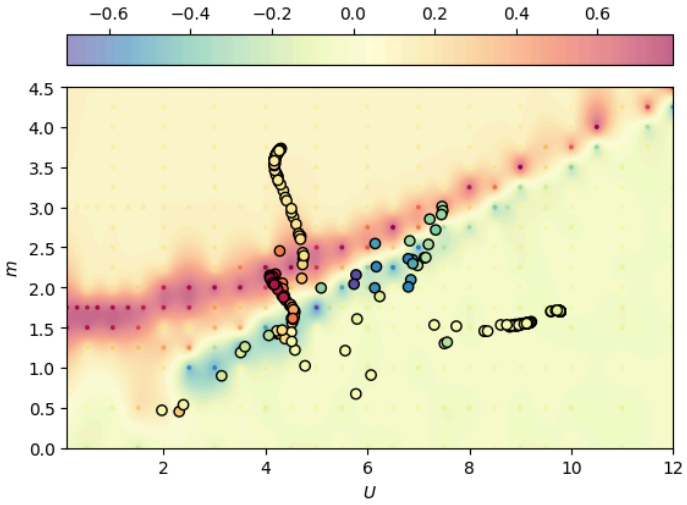


Note that all of the networks trained with cosine similarity as the loss function functioned poorly at recreating the input orbitals: it always overestimated their occupations. I'd assume this is because the cosine similarity only cares about the direction of the input and output "vectors", so it won't be penalized for overestimating them. Because of this, I trained another VAE using the mean squared error, which I would later use for generating new orbitals and spin spin correlations. Its autoencoder capabilities can be seen below, with the curves marked with (2) corresponding to the reconstructed values, and the others corresponding to the original values:



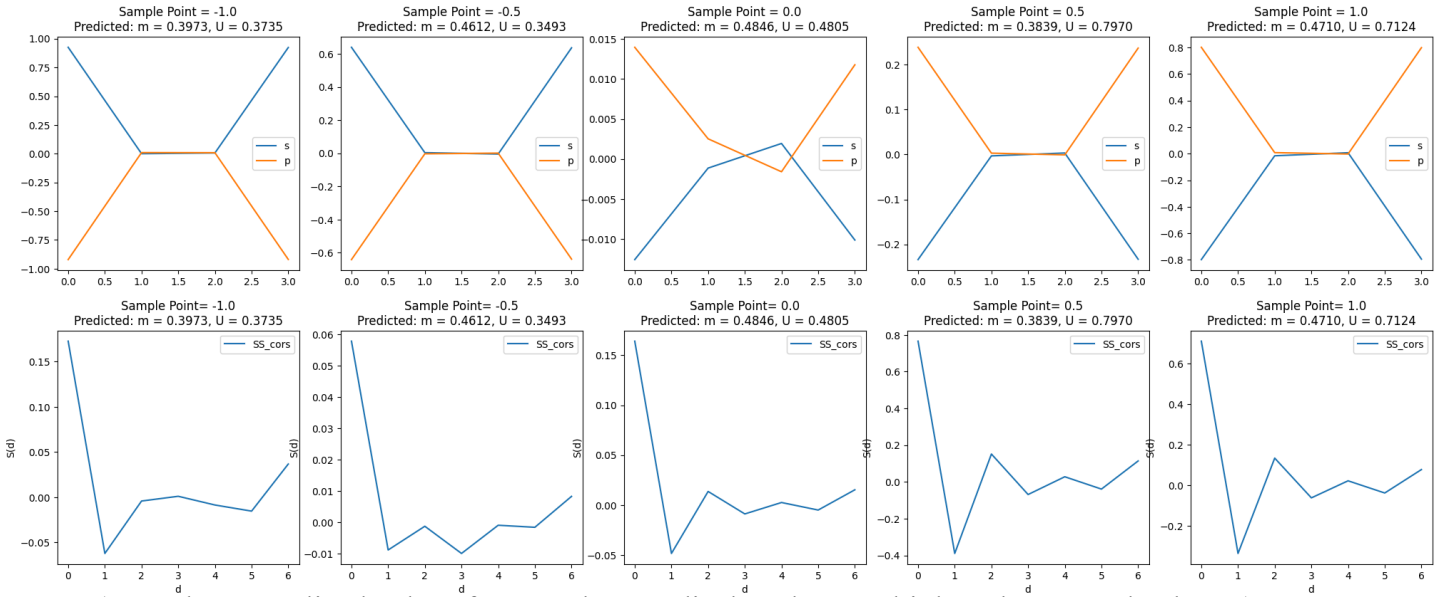
Since we could generate orbitals + spin correlations with the VAE, we wanted to see what their predicted values for m and U would be. Hence I designed a supervised neural network to identify m and U from the orbitals and spin correlations. The specifics are seen in the notebook.

Strangely, the predicted m and U values from the orbitals were always located in "clumps", so technically it wasn't actually good at predicting the exact m and U values of the orbitals. In the plot below the predicted values (black bordered points) for each orbital have been overlaid on the VAE from earlier, and everything has been colored based on the VAE's latent space value for the orbitals.



As to why these clumps arose, I'd assume that the network didn't have enough information to fully reconstruct the m and U values, and hence decided on the best values that it could in order to minimize the distances between the actual and predicted values. For instance, the predicted values generally fit in the same phase that the original values came from.

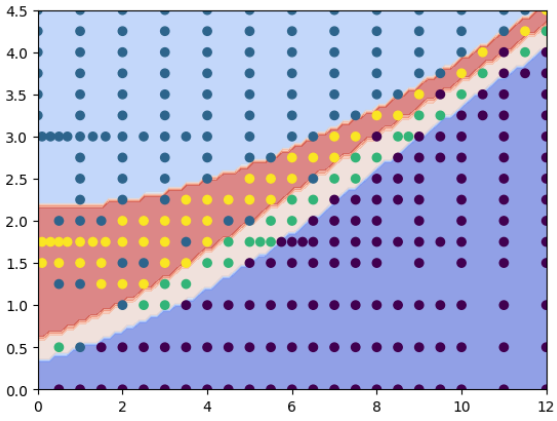
Finally, m and U were predicted for orbitals and spin spin correlations that were generated using the VAE:



(Here the normalized values for m and U are displayed, so multiply m by 4.5 and U by 12)

Finally, as a side project I wrote code that could automatically draw the boundaries between the phases for the plots that I generated during my research. This can be seen in the notebook "phase_boundary_drawer.ipynb". Here I used the support vector module from sklearn to draw boundaries between phases. This requires integer labels to be assigned to each class, however it can tolerate outliers, so it doesn't have to be perfect. For example, say the autoencoder always predicts values clustered around -1, -.5, 0, and , .5, and 1. Then less than and greater than cuts for the autoencoder / PCA values can be used to categorize the values as being

closest to one of those 4, and an integer value can be assigned to represent each category. I do a crude example of this in the notebook.



In addition to drawing the boundaries, the support vector classifiers I developed can also identify what region an arbitrary point lies in. As a tip, I've noticed that you may have to adjust the "C" parameter for the function, which encodes the penalty for missing outliers, until the boundaries look best.

Ultimately I was able to develop a VAE that could achieve better resolution between the phases compared to the PCA from the paper, so I'd say my project was a success. It is interesting to note that I also discovered that the original PCA was technically telling the two phases apart, however the difference was so small that none of us noticed it.

In general, while I tried to make my networks the best they could be, most of my time was spent trying out new things, not fine-tuning the parameters of each network. Hence I stress that if the orbitals generated by the VAE are to actually be used in research, we should try to make the networks as best as possible.