The Australian National University
2600 ACT | Canberra | Australia

Australian National University

School of Computing

College of Engineering, Computing and Cybernetics (CECC)

Unleashing the power of Machine Learning in Geodynamics

— 12 pt Honours project (S2 2023)

A thesis submitted for the degree
Bachelor of Advanced Computing (Honours)

By:
Xuzeng He

Supervisors:
Dr. Rhys P. Hawkins
Dr. Alberto F. Martin Huertas
Dr. Siavash Ghelichkhan

September 2023

Declaration:

I declare that this work:

- upholds the principles of academic integrity, as defined in the University Academic Misconduct Rules;
- is original, except where collaboration (for example group work) has been authorised in writing by the course convener in the class summary and/or Wattle site;
- is produced for the purposes of this assessment task and has not been submitted for assessment in any other context, except where authorised in writing by the course convener;
- gives appropriate acknowledgement of the ideas, scholarship and intellectual property of others insofar as these have been used;
- in no part involves copying, cheating, collusion, fabrication, plagiarism or recycling.

September, Xuzeng He

Abstract

(TODO...)

Table of Contents

Chapter 1

Introduction

Introduction (actually more like background) to Geoid problem and Mantle Convection (TODO...)

The rest of the paper is organized as follows. In Chapter 2, we briefly introduce the basic idea of neural networks (NN) and some specific architecture we use in this study. Some related works that use NNs to model the Geoid or mantle convection problem are also discussed in this chapter. In Chapter 3, we approach the Geoid problem using a simple fully connected neural network (FNN) and present the results. Then in Chapter 4, we present how we model the mantle convection problem by first compressing the temperature fields using a convolutional autoencoder (ConvAE) and then predict the compressed temperature fields using two different Machine Learning (ML) architecture - FNN and LSTM. In the same chapter, we also discussed the difference between these two architectures by visualising the result as GIF files and applying Principle Component Analysis (PCA). In the final chapter, we conclude this paper by offering some potential follow-ups about modelling mantle convection simulations using neural networks.

Chapter 2

Background

2.1 Neural networks: Fully connected, convolutional and long short-term memory

In this section, we introduce some basics about neural networks (NNs) according to a more detailed explanation provided by Bishop. (10., 2007) A simple NN usually consists of one input layer, one output layer and one or more hidden layers in between. Multiple nodes inside each layers are called neurons.

For a fully connected neural network (FNN) using linear layers for connection (the term "fully connected" means that the neurons in each layer have a connection to all neurons in the previous layer and also the following layer), each neuron receives all inputs from the previous layer and the outputs are as follows:

$$z = g(xA^T + b) \tag{2.1}$$

In this case, z represents a vector contains the result values of all the nodes in the current layer and x represents a vector contains the values of all the nodes in the previous layer. $g()$ represents an activation function allows the NN to model non-linearity. A is the weight matrix and b is the bias vector from the linear layer (It is called a linear layer since it applies a linear transformation to the incoming data). Both A and b are able to be optimized through a technique called error backpropagation, where we first define a loss function calculate the loss value (error) between the predicted output from the output layer and the actual output from the data set. The error is then propagated backwards through all the hidden layers using chain rule to perform differentiation. Eventually, these derivatives of errors with respect to the weights are used to update the learnable parameters (A and b) in each hidden layer. This complete process is called gradient descent.

By looping over the process of feeding the input data into NN to perform prediction, calculate the loss between the prediction and the actual data, use the loss value to optimize the learnable parameters in each layers, the NN model is able to be adjusted to a state that best fits the underlying pattern of the provided data set given its current structure.

In this study, we use a ML library called PyTorch to define the NN architecture, specifying the loss function and optimizer that minimize the loss function (Adam optimizer, in this case, is used throughout the study) and systematically train and test the performance of the networks.

Apart from FNNs, Convolutional neural networks (CNNs) are also one of the most commonly used NNs. They can handle matrix or image input better than the traditional FNN with linear layers. Instead of linear layers, they use convolutional layers that contains a specified number of trainable filters, each of which is used to enhance or identify a particular feature in the input. Filters are convolved with the input image or the output of the previous layer, and the results are summed together along with a trainable bias and passed through an activation function to produce a feature map. Because convolution is also a linear operation, activation function is added to introduce non-linearity to the feature map like FNN.

In this study, a variation of CNN is used as a way to compress the size of the input data in the mantle convection problem, which is called Convolutional Autoencoder (ConvAE). It is constructed as two separate sturcture that trained together: an encoder using convolution operation to reduce the size of the original input field and output a latent space representation, and a decoder using deconvolution operation to transform the latent space representation back to the original size field. By feeding the input field into a ConvAE, the dimension of the original high-resolution fields can be decreased with the main features captured (some information may be lost during the encoding process), thus making it more computationally efficient to work with before we feed it into a prediction NN.

Since the data set in the mantle convection problem is a time sequence with adaptive timestamps, long short-term memory (LSTM) is used to predict a sequence of output apart from FNN. LSTM's architecture that use a sequence of input recurrently during prediction allows it to handle time-series data more accurately than other networks since it uses a set of previous time-steps to predict the next set of time-steps, thus leading to a potential better result.

2.2 Related works for solving geoid and mantle convection using Neural networks

Neural networks has been increasingly used for studying the geodynamics nowadays, especially when it comes to solving geoid or mantle convection simulation.

For example, Kerl provide a bold attempt at using ML as a low-cost solution to the geoid inverse problem in his thesis, where two separate solutions using different number of CNNs are compared.(Kerl, 2022) He found that the single network solution where a radial viscosity profile is predicted directly from the geoid and density data allows him to obtain a smooth, long-wavelength estimate of the Earth's radial viscosity profile. This provides us some implications on using only one NN to solve the inverse problem instead of stacking NNs together.

As for the mantle convection problem, Agarwal, S. et al. make use of the FNN architecture to build a surrogate model that can predict the entire evolution (0–4.5 Gyr) of the 1D temperature profile of a Mars-like planet for a wide range of values of five different parameters, including reference viscosity, activation energy and activation volume of diffusion creep, enrichment factor of heat-producing elements in the crust and initial temperature of the mantle.(Agarwal et al., 2020)

In another study that is particularly worth highlighting, Agarwal, S. et al. extend the previous approach (Agarwal et al., 2020) of using FNN trained using a large number of 2D simulations to predict the evolution of entire 1D laterally-averaged temperature profile in time for complex models. Instead of predicting 1D temperature field, the full 2D temperature field are predicted since it could contain more information related to the structure of the convection.(Agarwal et al., 2021) To show that NN techniques can produce reliable parameterized surrogates, they first use ConvAE to compress the size of each temperature field by a factor of 142 and then use FNN and LSTM to predict the compressed fields. They discovered that LSTM capture the flow dynamics better than FNN despite the fact that LSTM has a lower mean relative accuracy. Their study provides us some essential insights in solving the mantle convection problem by using NNs as a low-cost solution, including using ConvAE to compress the data and compare the prediction result of two different architectures (FNN and LSTM).

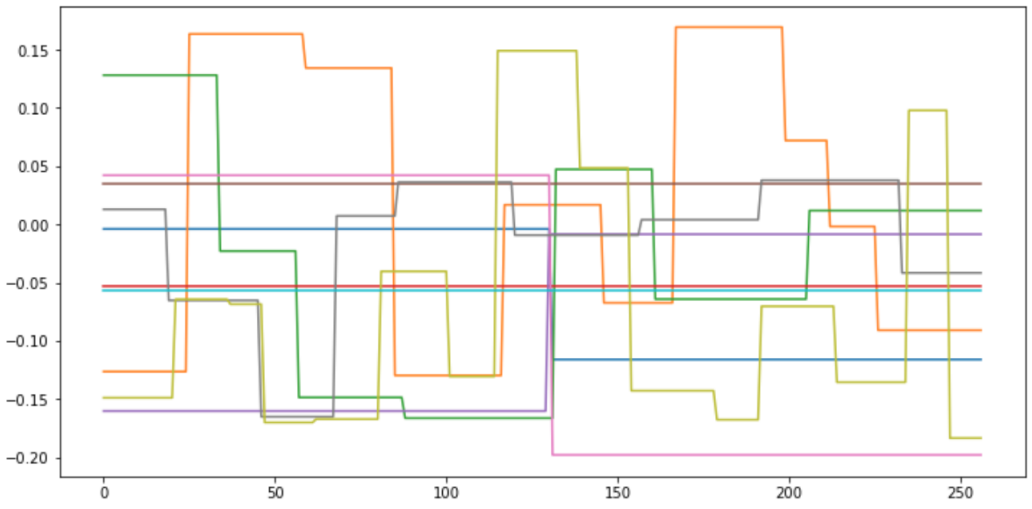
Chapter 3

Geoid prediction

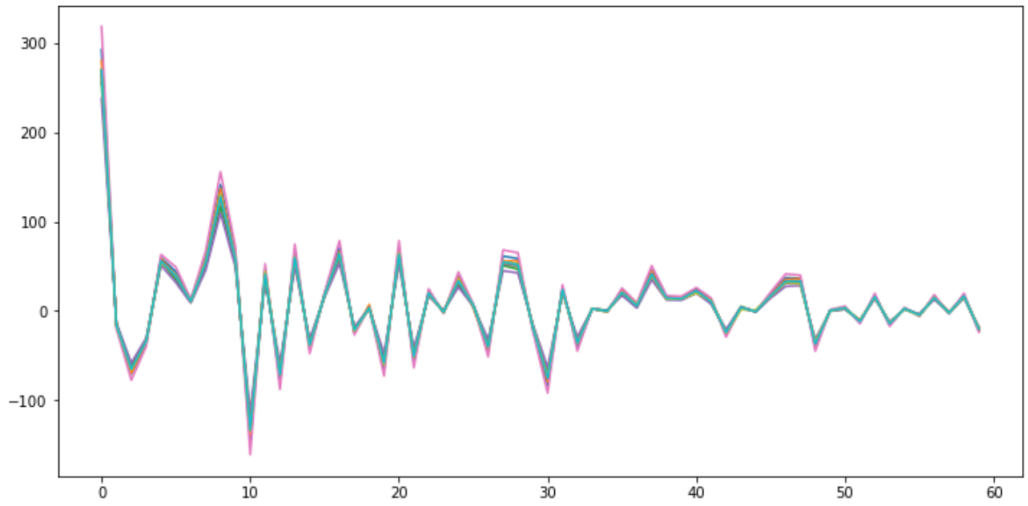
3.1 Dataset of Geoid prediction

The data set consists of 1000 pairs of input and output. In the data set, input is a vector with 257 values representing a geoid model and output is a vector with 60 values representing a set of parameters. The following graphs plot 10 pairs of model input and output in separate figures to observe the patterns.

Every 100th input in the data set



Every 100th input in the data set



From the second plot we can observe that the outputs in the data set can be seen as some "curves" with the same patterns but different altitude. In this case, each of the 60 values in the output data are normalised to be between 0 and 1 using their maximum and minimum values separately before feeding into the neural network. This is because each of the parameters in an output vector can be seen as equal and we want to prevent the neural network from spending most of its effort learning the parameter with a higher range. Hence, one can expect a higher accuracy when the parameters in the output data are standardized to a same range.

After the output data is normalised using a scaler, the entire data set is randomly divided in a ratio of 8:1:1: 80 per cent of the data set is used for training, 10 per cent of the data for testing accuracy and the remaining 10 per cent to perform validation during training and prevent overfitting. For a data set with 1000 samples, this result in a train-test-valiadation split of 800-100-100.

3.2 Fully connected Neural Network (FNN) for Prediction

To test FNNs with different architectures (e.g. different number of hidden layers and neurons per hidden layer) or other hyperparameters (e.g. optimizer), a systematic testing method is applied. This method mainly consists of three files: one file to store all the different set of FNN architectures and hyperparameters in a text format, another file to fetch all these combinations of architectures and hyperparameters line by line, build them as FNN models and train these models, and the last file for testing and visualisation of the trained models by specifying the path of the trained model. The training file and the testing file are both in the format of a Jupyter Notebook.

The trained FNN architecture (in the format of a light-weight file) along with another

text file contains the training loss and validation loss during training will be stored in a specified path for further testing and visualisation. The name of these two files uniquely defines each experiment by including the values of hyperparameters to generate the model in the file names. These files are also put in separate folders with the folder name associated with commit IDs to handle tracking of the process during the research in an educated or extensible way.

In this way, one can open the same testing Jupyter Notebook in different browser tabs, and then visualize simultaneously different models in different tabs using a cell in which the paths to the FNN file and its training data are specified.

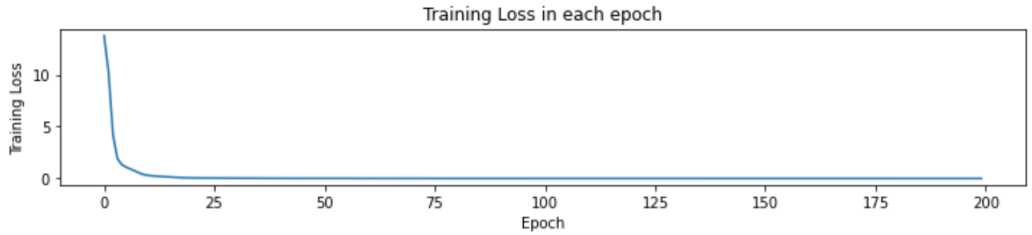
The systematic testing capability is implemented here to ensure traceability. In other words, as different values of the hyperparameters are tested, We would like to be able to record the results (e.g., the trained network and the training data) so that we don't have to repeat them again or rely on our memory to compare the the performance of different structures.

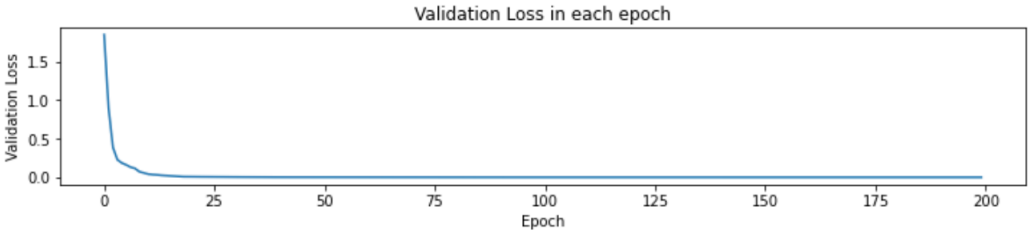
Also, to prevent overfitting, a variation of the early-stopping method is used during training. The normal early-stopping method let the network train until the error function evaluated on the validation set starts to increase beyond a certain threshold(Prechelt, 2012), while my implementation only stores the best model during training (the one with the lowest validation loss) in a specified path and allows the network to keep training as normal. In this case, the output model is the best model instead of the last trained model. This method is also used in the following chapter when implementing the ConvAE, FNN and LSTM to solve the mantle convection problem.

After testing with NNs with architectures of different number of hidden layers and neurons per hidden layer, we found that architectures with a total number 3–4 hidden layers seemed to perform the best.

In the following figures, we present results from a FNN with 4 hidden layers with 200, 160, 120 and 80 neurons, ReLU as activation function, MSELoss as loss function, and trained for 200 epochs using Mini-Batch Gradient Descent (with a batch size of 16).

Training loss and Validation loss



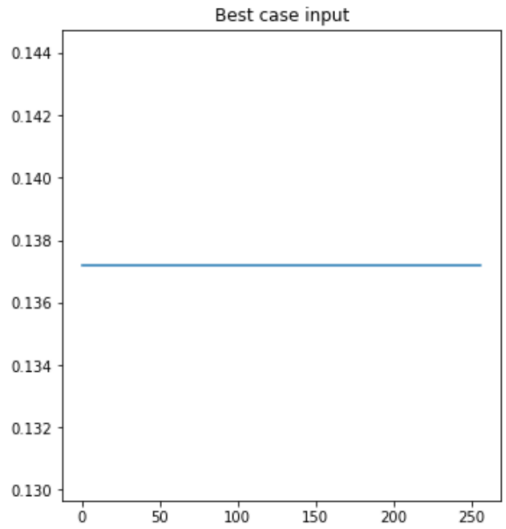


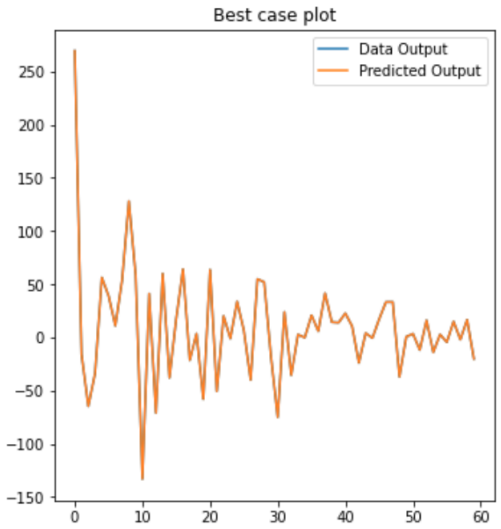
Overall testing result

```
Total loss for the model is 0.00023521817990547456, and accuracy is 100%
When the loss threshold is set to 0.01, accuracy is 100%
When the loss threshold is set to 0.001, accuracy is 99%
When the loss threshold is set to 0.0001, accuracy is 96%
When the loss threshold is set to 1e-05, accuracy is 50%
```

Best input and output

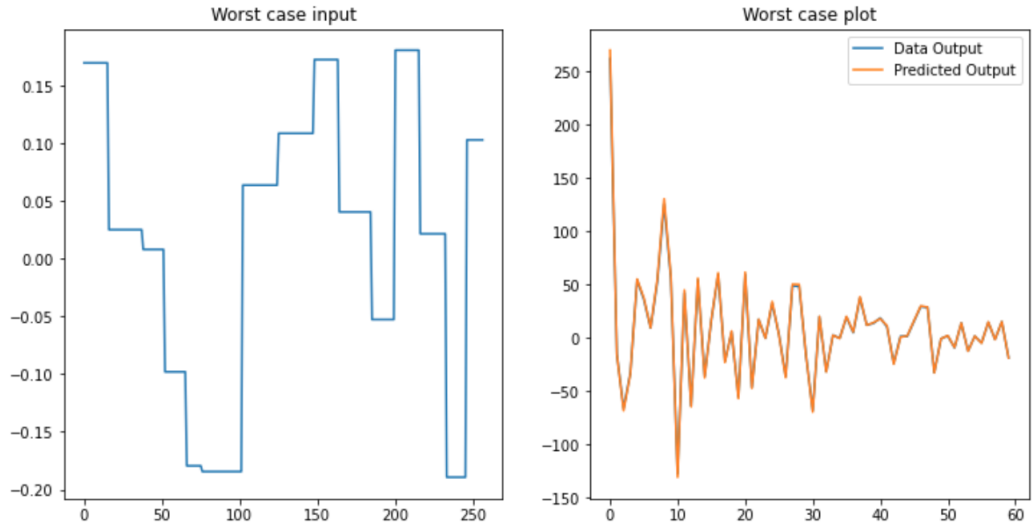
```
Best model has a error of  8.997102327635002e-07
```





Worst input and output

```
Worst model has a error of  0.0013982787387692296
```



On average, the prediction loss are low even when the loss value here is calculated using the normalised output data and no overfitting occurs. The accuracy of the prediction is nearly 100% when the threshold is set to be 10 times lower than the worst loss value. Overall, FNN is able to accurately solve the Geoid problem without numerous amount of data.

Chapter 4

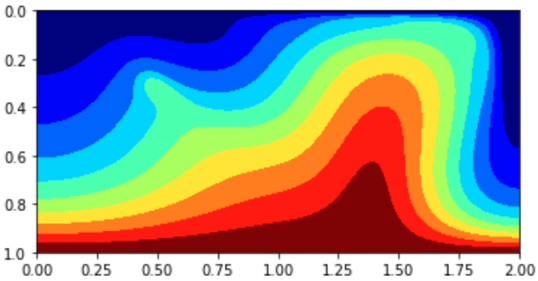
Mantle Convection Simulation

4.1 Dataset of mantle convection simulation

The data set consists of 100 files and each of them represents one mantle convection simulation in 100 time steps generated with an initial condition with random number of points and random amplitude and coordinates of Gaussian anomalies distributed in space (a total of 10000 temperature fields). Starting from each initial condition we convect as long as there is meaningful change in the simulation (that is the temperature fields change enough after one time-step). There are 100 temperature fields with a size of 201x401 for each of the 100 timestamps in a file and the time step has to be adaptive, otherwise the whole random generation of the initial condition would be hard to implement.

The following figure shows one random temperature fields in the data set with the y-axis inverted and colored using 10-color-map (all the figures in this chapter will have their y-axis inverted and colored using 10-color-map as well):

Temperature Field example



The entire data set is still randomly divided in a ratio of 8:1:1: 80 per cent of the data

set is used for training, 10 per cent of the data for testing and the remaining 10 per cent to perform validation. However, the way we divide them is different in implementing the ConvAE, FNN and LSTM and will be discussed in more details in the following sections.

The data set is uploaded to Gadi, a HPC system, for storage to make the training process more efficient.

4.2 Compression of temperature fields

Since applying Machine Learning (ML) algorithms directly on the original sized temperature fields can take significantly more times to train the model and could possibly lead to the risk of over-parameterization, we decided to compress the temperature fields first before feed the data into different ML architectures.

In this study, the overall process to solve the mantle convection problem woule be:

1. Train the ConvAE
2. Train the FNN/LSTM using the latent space representation for both input and output data (using encoder from ConvAE)
3. Test the prediction result in original size (using decoder from ConvAE)

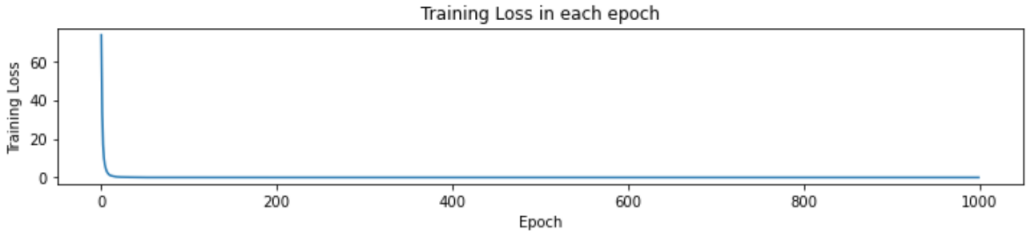
The complete 10000 temperature fields are randomly shuffled and divided in a ratio of 80%, 10%, 10% for training, testing and validation where each piece of data consists only one temperature field (fed as both the input and output). Given the size of the training set, ConvAE is fed with a batch size of 16 temperature fields during the training to perform Mini-Batch Gradient Descent.

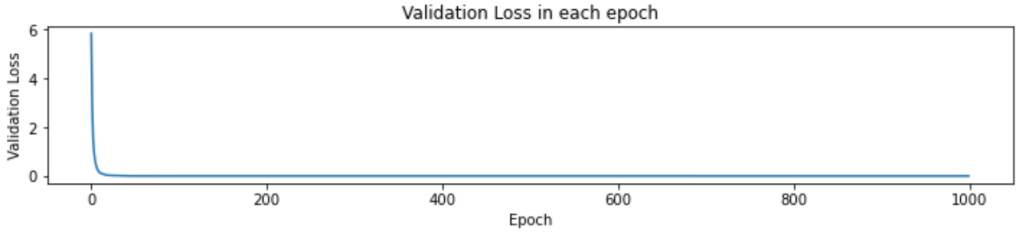
We find that ConvAE with a latent space size of 5x23x45 offers an excellent compression factor of 13, while being able to reconstruct the temperature fields in original size with an accuracy of 100% on the test set.

The architecture of the ConvAE in this case consists of two convolutional layers for the encoder and two transpose convolutional layers for the decoder. Both of these four layers have the size of the filter as 5x5 and a stride of 3x3. Tanh() is used as activation function to introduce non-linearity between each layers (except for the last layer in the decoder) and mean square error (MSE) is used as the loss function. The model is trained for 1000 epochs on Gadi.

In the following figures, we present some detailed test results from this ConvAE:

Training loss and Validation loss



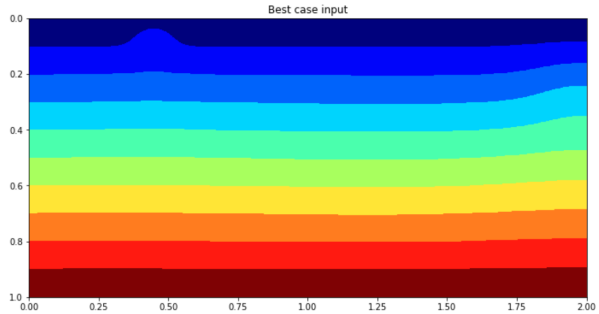


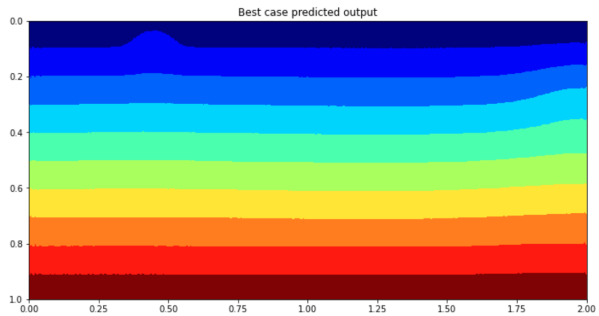
Overall testing result

```
Total loss for the model is 0.00020430381073310855
Accuracy for the model is 100
```

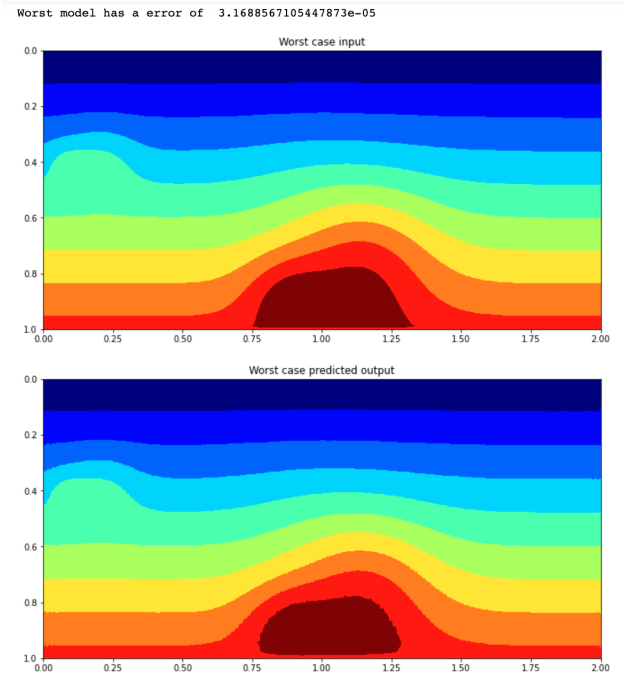
Best input and output

Best model has a error of 4.425904194249597e-07





Worst input and output



On average, the reconstruction loss are low and no overfitting occurs. The accuracy of reconstruction is nearly 100%. We also applied some PCA analysis to the compressed-decompressed fields with contrast to the original temperature fields and the result is great as well, the figure to this analysis will be shown in the following sections in contrast to the PCA analysis of the prediction model (FNN or LSTM)

4.3 Fully Connected Neural Network (FNN) for Prediction

We now move on to predict the latent space representation and the first candidate is FNN. The FNN in this study uses each pair of temperature fields as one training set, takes one temperature fields as the input and output the temperature fields at the next time step. Therefore, the data set is reconstructed into 9900 pairs of temperature fields with consecutive timestamps. These pairs are randomly shuffled and divided in a ratio of 80%, 10%, 10% for training, testing and validation where each piece of data consists two temperature field having consecutive timestamps.

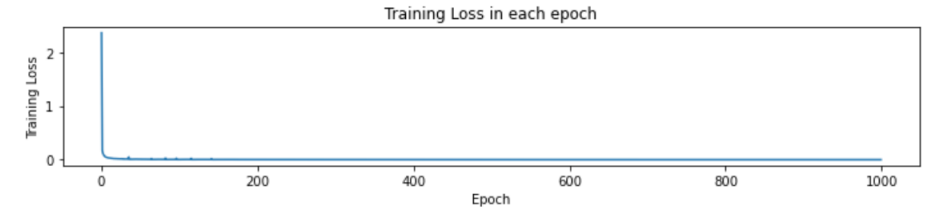
Before the latent space representation is fed as the input of FNN, it is flattened into one dimension. The prediction of FNN in this case is then resized from a one dimensional vector (1x6210) to the shape of the latent space (6x23x45) for the convenience of further testing.

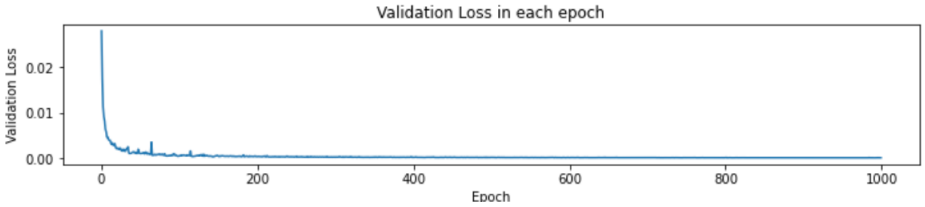
The learnable parameters of FNN are optimized using small mini-batches of 16 pairs of consecutive temperature fields and Adam as the optimizer, where the loss function is defined as the mean square error (MSE) between the prediction of FNN and the actual output (both in the shape of latent space representation, which is 6x23x45).

After testing with FNNs with architectures of different number of hidden layers and neurons per hidden layer, we found that architectures with a total number 3 hidden layers seemed to perform the best. (We also test some deeper architectures with 4-5 hidden layers. However, there is no significant improvement in loss value)

In the following figures, we present results from a FNN with 3 hidden layers with 3105, 1035, 3105 and 80 neurons, Tanh() as activation function, and trained for 1000 epochs.

Training loss and Validation loss



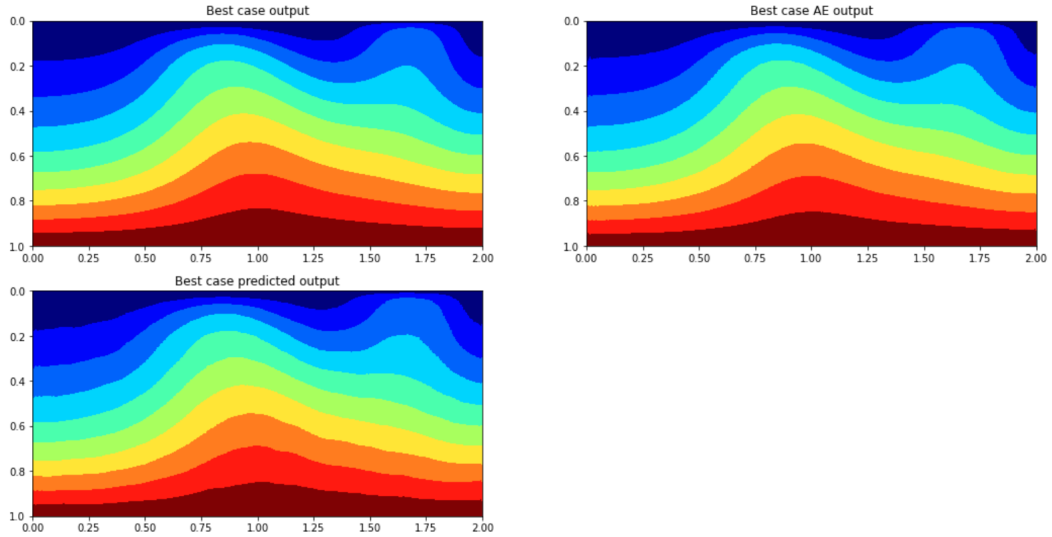


Overall testing result

```
Total loss for the model is 9.397740370786778e-05
Accuracy for the model is 100
```

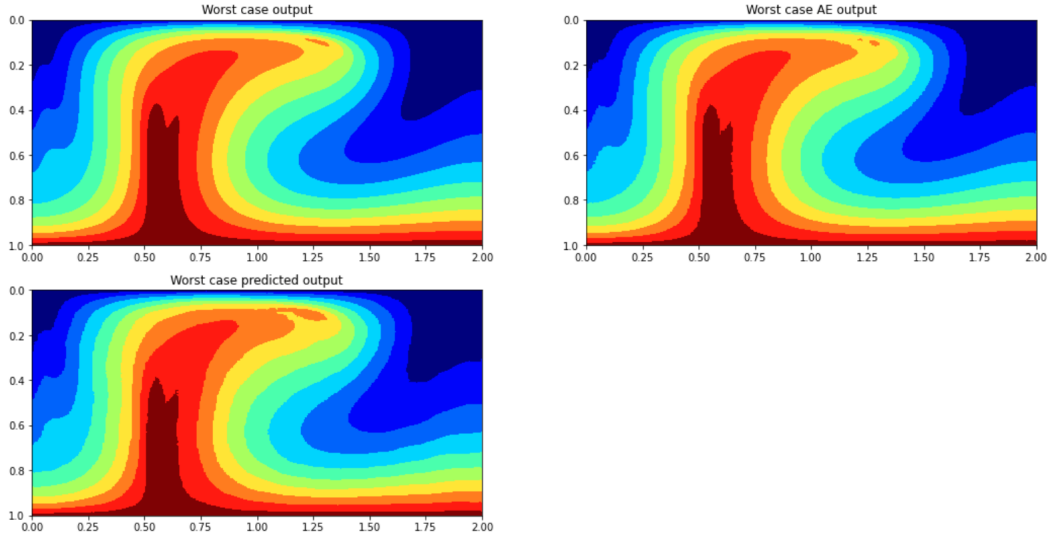
Best case original output, original output after ConvAE's compression-decompression, and predicted output of FNN

```
Best model has a error of  6.885692869218474e-07
```



Worst case original output, original output after ConvAE's compression-decompression, and predicted output of FNN

```
Worst model has a error of  6.805325028835796e-06
```



On average, the loss values are low and no overfitting occurs. The prediction for the temperature field at the next timestamp is able to capture the main features precisely with some small information loss, which is partially due to information loss caused by

the compression-decompression process of ConvAE.

To further evaluate the performance of FNN in predicting a complete time series, two methods are tested on all 100 files one by one:

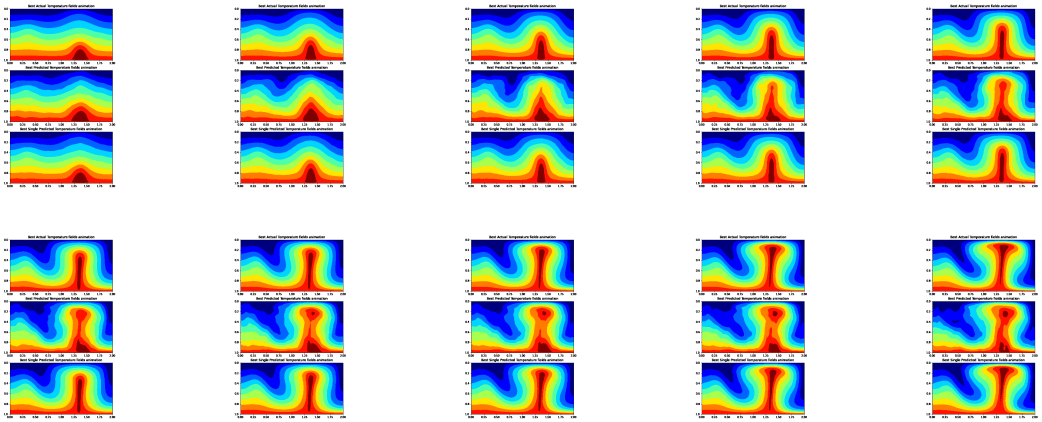
1. Only take the first temperature field in the file as the input and use a prediction-as-input loop to get the rest of the 99 temperature fields. (use $T1$ from data set $\rightarrow$ get predicted $T2$ $\rightarrow$ use predicted $T2$ $\rightarrow$ get predicted $T3$ $\rightarrow$ ...)
2. Constantly feed a temperature field from the original data set and get the temperature field at the next time step as usual. (use $T1$ from data set $\rightarrow$ get predicted $T2$ $\rightarrow$ use $T2$ from data set $\rightarrow$ get predicted $T3$ $\rightarrow$ ...)

In this case, the first method can reduce the computation complexity of the mantle convection problem more effectively than the second method, since we only need one initial input data and the model can generate the rest of the temperature field sequence. Therefore, the following best case and worst case will be evaluated using the first method.

To better visualize the prediction result of the above two methods, two animations representing the best case and the worst case (evaluated based on the sum of MSE for each prediction using the first method) in the format of GIF files are generated. From top to bottom, the first picture represents the actual output from the data set, the second one represents the prediction result using the first method, and the last one represents the prediction result using the second method.

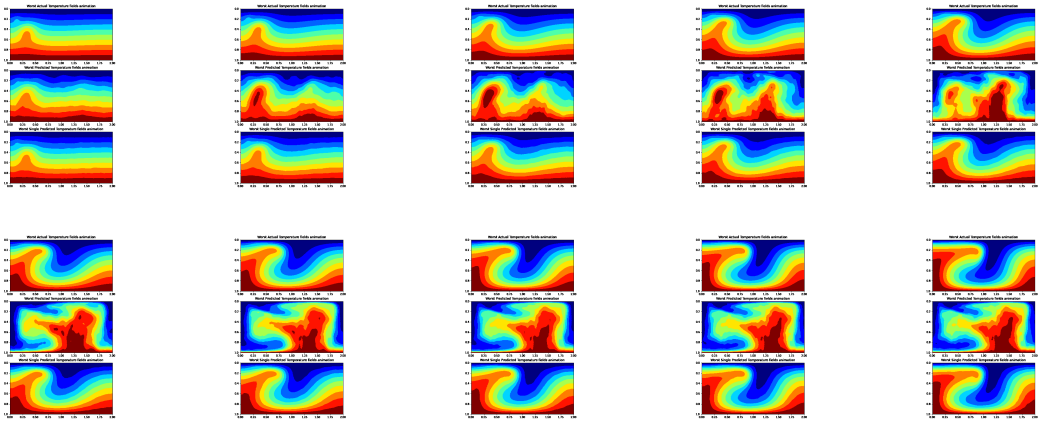
The following figures show 10% of the sprites sheets converted from the original GIFs (Every 10th frame) for the convenience of reading:

Best case animation sheet



Link to this GIF: https://drive.google.com/file/d/1Hmb4UlevBHMRw0jDScTwUzDFPbYNubKO/view?usp=sharing

Worst case animation sheet

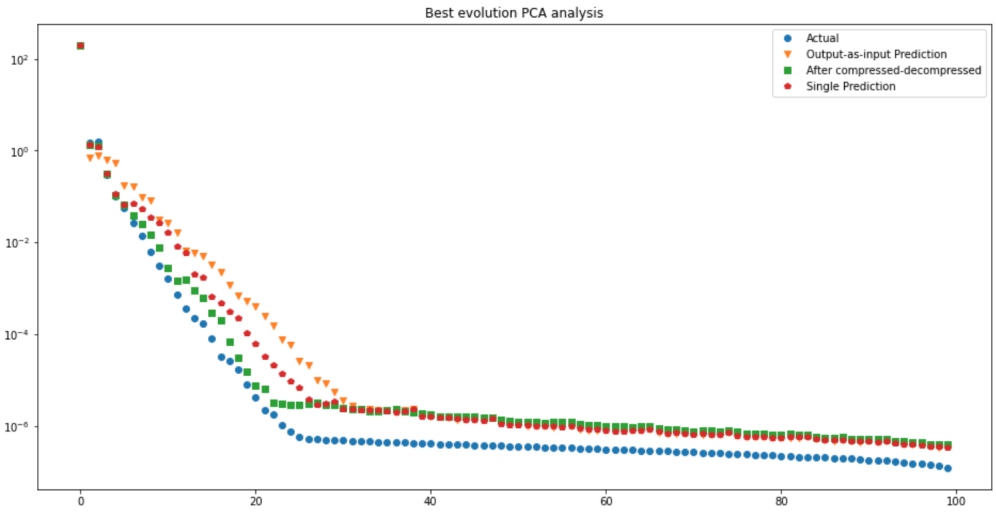


Link to this GIF: https://drive.google.com/file/d/11jRFxq-XuIUvTk74OuxswDn3u031k_q7/view?usp=sharing

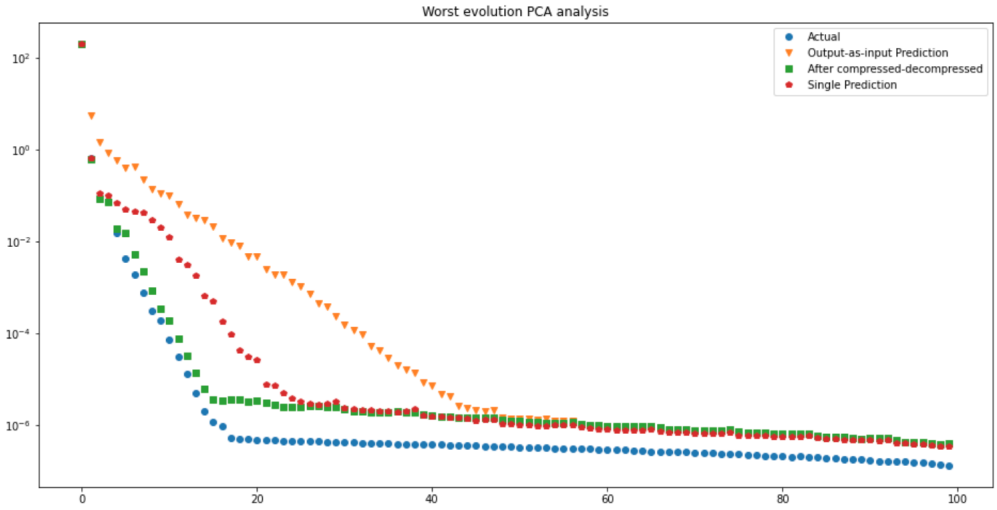
To further evaluate the performance of these two methods, we also applied Principle Component Analysis (PCA) to a set of predictions generated by these two methods in both the best case and the worst case, with the original time series and the compressed-decompressed version generated by ConvAE serve as contrast.

The following figures show the PCA result in best case and the worst case:

Best case PCA



Worst case PCA



As seen in the animations and the PCA results, the first method (output-as-input prediction) fails to capture the trend of the temperature fields in the worst case and the result prediction is completely different to the actual data. This is because the information loss from the ConvAE and FNN in each prediction-as-input loop gets summed up, thus leading to a low accuracy. In the mean time, the prediction result for the second method mostly matches with the actual output in both cases, which is consistent to our theory that the it is the first method itself leads to the huge information loss.

Therefore, to generate a sequence of temperature fields with less information loss, we try to use LSTM to predict the latent space representation instead.

4.4 Long short-term memory (LSTM) for Prediction

We now move on to predict a sequence of latent space representation using LSTM. The LSTM in this study uses the first 50 temperature fields as a sequence in each file as one training set, takes a sequence of 50 temperature fields as the input and output the rest 50 temperature fields at the next time steps.

This 50:50 of input-output length is determined by PyTorch's limitation on LSTM, where the input length and the output length should be the same for the LSTM to functionally work. A ratio of less input length and more output length (20:80) has been considered, but this requires the input sequence to be replicated into 4 times the size to match with PyTorch's requirement, which could lead to worse prediction result compared with the 50:50 ratio.

Therefore, the files in the data set are randomly shuffled and divided in a ratio of 80%, 10%, 10% for training, testing and validation where each piece of data is a complete file

to be divided in half as input and output.

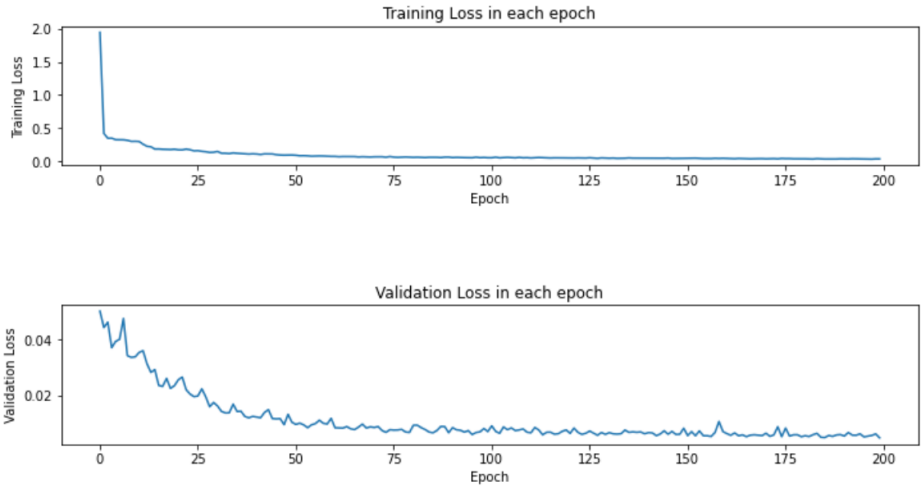
Again, before a sequence of latent space representation is fed as the input of LSTM, it is flattened into two dimension with its sequence index reserved (50x6210). The prediction of LSTM in this case is then resized from 50x6210 to 50x6x23x45 for the convenience of further testing.

The learnable parameters of LSTM are optimized using small mini-batches (only 1 batch in this case since we merely have 80 training samples) and Adam as the optimizer, where the loss function is defined as the MSE between the prediction of LSTM and the actual output (both in the shape of a sequence of latent space representation, which is 50x6x23x45).

After testing with LSTMs with different architectures, we found that architectures with a total number of 2 consecutive LSTMs layers seemed to perform the best. (We also test some deeper architectures with 4-6 LSTM. However, there is no significant improvement in loss value)

In the following figures, we present results from an architecture with 2 LSTM layers (first one input size is 6210, hidden size is 3105; second one input size is 3105, hidden size is 6210. Both of them only have one layer, no internal stacking), and trained for 200 epochs.

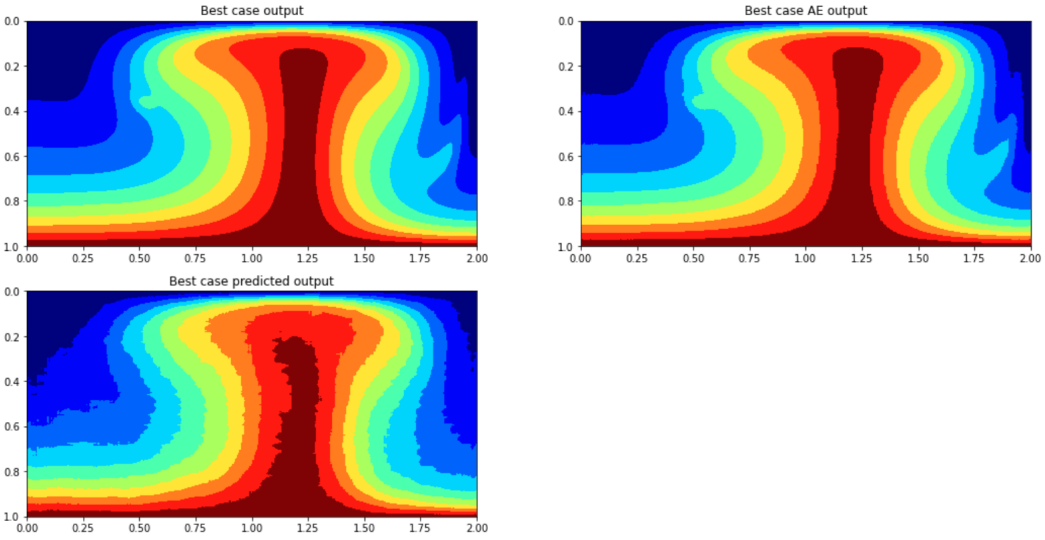
Training loss and Validation loss



Overall testing result

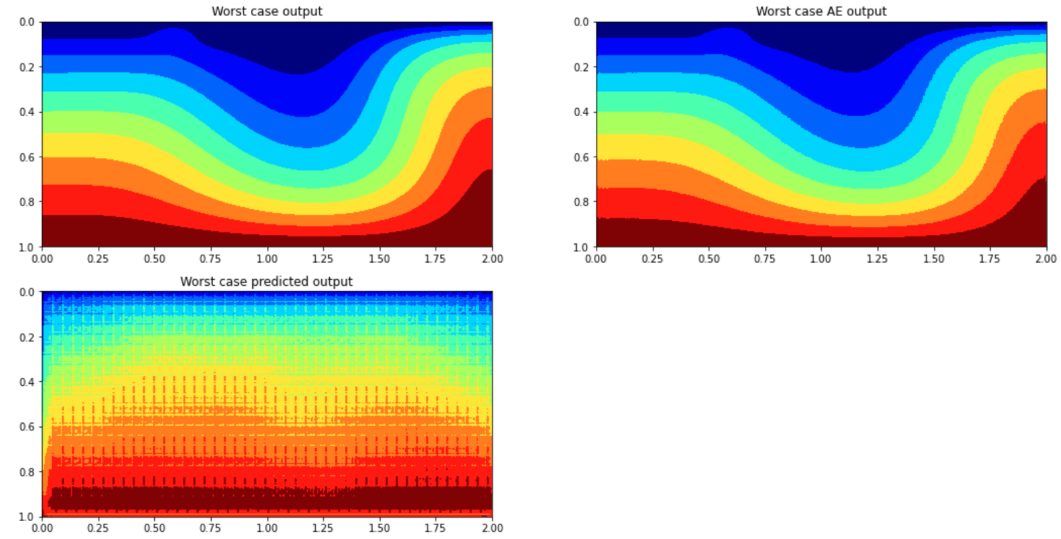
```
Total loss for the model is 0.003697341730003245
```

Best case original output, original output after ConvAE's compression-decompression, and predicted output of LSTM



Worst case original output, original output after ConvAE's compression-decompression, and predicted output of LSTM

```
Worst model has a error of  0.007273165509104729
```

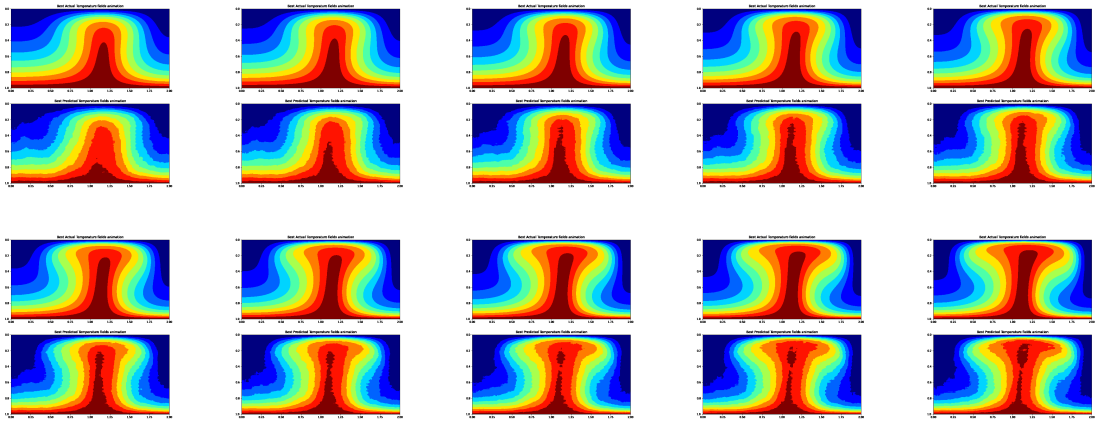


On average, the loss values are higher than the FNN and no overfitting occurs. The prediction for the temperature field at the next timestamp is able to capture the main features precisely with some small information loss in the best case, but fails to do so in the worst case, which is probably caused by the internal structure of LSTM.

To better visualize the prediction result of LSTM on a 50:50 input-output ratio, two animations representing the best case and the worst case (evaluated based on the sum of MSE for each prediction using the first method) in the format of GIF files are generated. From top to bottom, the first picture represents the actual output from the data set, the second one represents the prediction result using the first method, and the last one represents the prediction result using the second method.

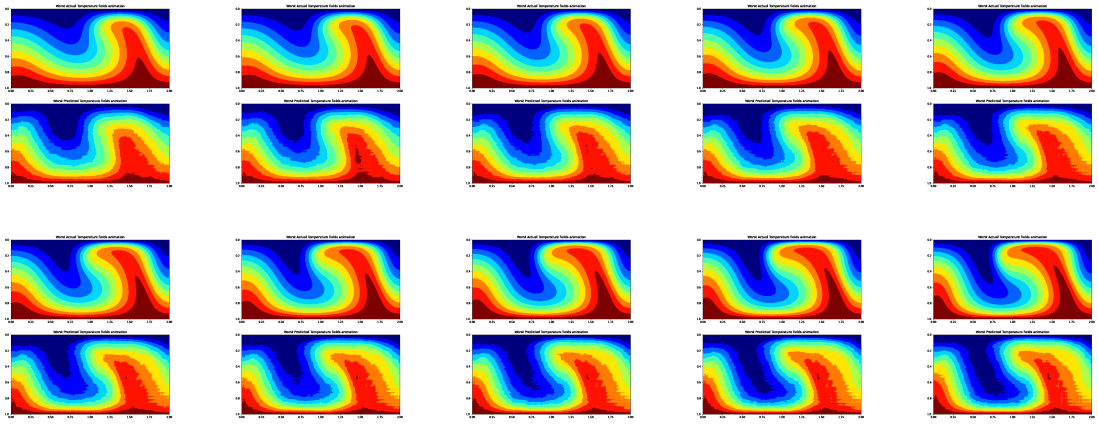
The following figures show 20% of the sprites sheets converted from the original GIFs (Every 5th frame) for the convenience of reading:

Best case animation sheet



Link to this GIF: https://drive.google.com/file/d/1zNJrZUB8XBAEWsUHPd2NANuWkE8yNg3z/view?usp=sharing

Worst case animation sheet



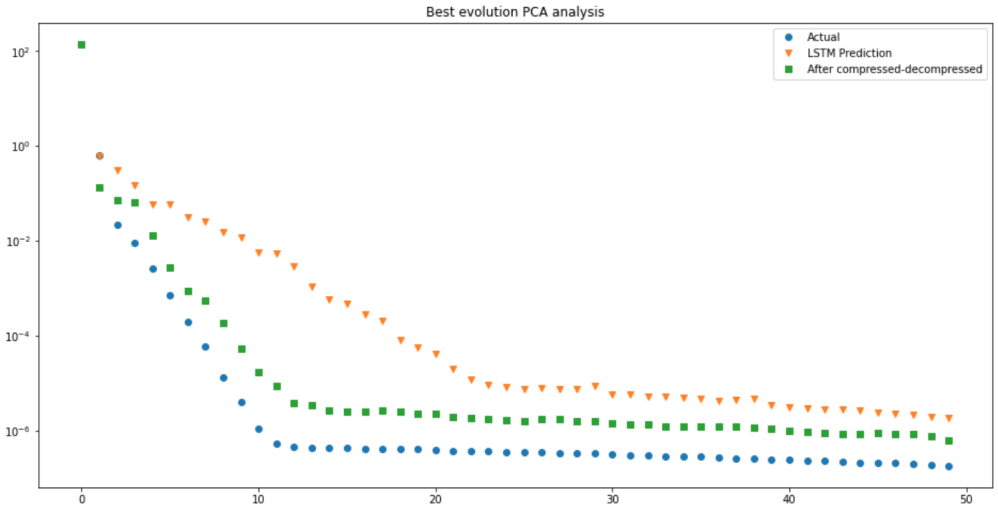
Link to this GIF: https://drive.google.com/file/d/1nINRk2Oh8rgQBLWio_b6R6I

7T18e7Zi-/view?usp=sharing

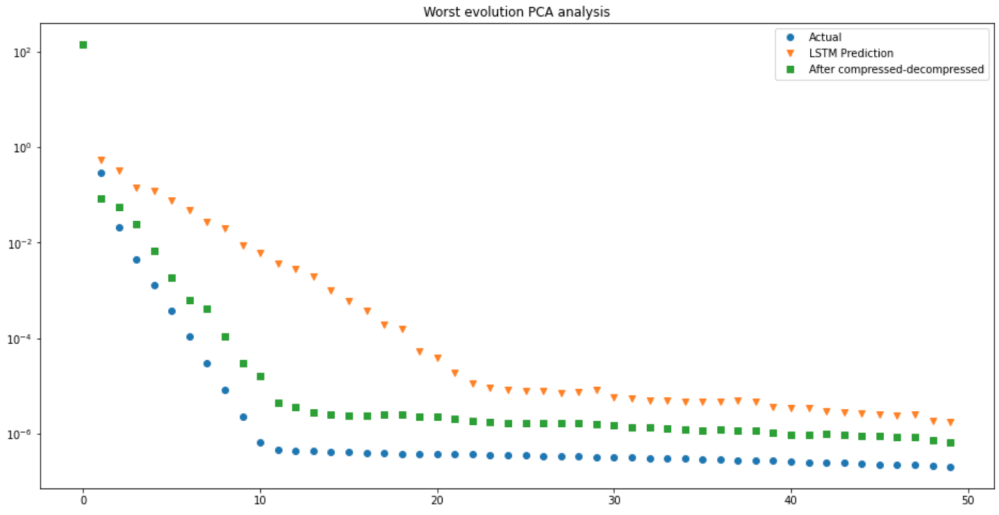
To further evaluate the performance, we also applied Principle Component Analysis (PCA) to a set of predictions generated by LSTM in both the best case and the worst case, with the original time series and the compressed-decompressed version generated by ConvAE serving as contrast.

The following figures show the PCA result in best case and the worst case:

Best case PCA



Worst case PCA



Chapter 5

Concluding Remarks

5.1 Conclusion

(TODO...)

5.2 Future Work

(TODO...)

Bibliography

2007. Pattern recognition and machine learning. *Journal of Electronic Imaging*, 16, 4 (jan 2007), 049901. doi:10.1117/1.2819119. https://doi.org/10.1117/1.2819119. [Cited on page 2.]

Agarwal, S.; Tosi, N.; Breuer, D.; Padovan, S.; Kessel, P.; and Montavon, G., 2020. A machine-learning-based surrogate model of mars' thermal evolution. *Geophysical Journal International*, 222, 3 (may 2020), 1656–1670. doi:10.1093/gji/ggaa234. https://doi.org/10.1093/gji/ggaa234. [Cited on page 4.]

Agarwal, S.; Tosi, N.; Kessel, P.; Breuer, D.; and Montavon, G., 2021. Deep learning for surrogate modeling of two-dimensional mantle convection. *Physical Review Fluids*, 6, 11 (nov 2021). doi:10.1103/physrevfluids.6.113801. https://doi.org/10.1103/physrevfluids.6.113801. [Cited on page 4.]

Kerl, J., 2022. *Geoid Inversion For Mantle Viscosity With Convolutional Neural Networks*. Ph.D. thesis. [Cited on page 4.]

Prechelt, L., 2012. Early stopping — but when? In *Lecture Notes in Computer Science*, 53–67. Springer Berlin Heidelberg. doi:10.1007/978-3-642-35289-8_5. https://doi.org/10.1007/978-3-642-35289-8_5. [Cited on page 7.]