Institute for Visualization and Interactive Systems

University of Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Master Thesis

Visual Analysis of a Machine Learning Approach Applied to Turbulence Data

Sathish Aanand Balasubramanian

Course of Study:	M.Sc. Information Technology
Examiner:	Prof. Dr. Daniel Weiskopf
Supervisor:	Tanja Munz, M.Sc., Daniel Klötzl, M.Sc., Marius Kurz , M.Sc.
Commenced:	March 28, 2022
Completed:	October 28, 2022

Acknowledgement

I would like to express my heartfelt gratitude to my supervisors, Tanja Munz, M.Sc., Daniel Klötzl, M.Sc., and Marius Kurz, M.Sc., for all of the insightful discussions we had while writing this thesis work and for their ongoing support, which has kept me motivated while working on this thesis. Also I would like to express my heartfelt gratitude to Professor Dr. Daniel Weiskopf for providing me with this opportunity to do my master's thesis at the Visualization Research Center (VISUS) of the Institute of Visualization and Interactive Systems (VIS) at the University of Stuttgart.

I would like to thank my family and friends for their moral support and encouragement over the duration of my master thesis.

Abstract

Machine learning is a subsection of artificial intelligence that has revolutionized recent decades with its remarkable performance when compared to the conventional state-of-the-art methods in the fields of science and information technology. However, a trained machine learning model is regarded as a black box model due to the non-transparent nature of its internal workings. This lack of transparency reduces the interpretability of trained machine learning models, making them less desirable in life-critical applications such as aerodynamics or the medical industry. Because of their uncertainty, these models cannot be employed until their internal workings have been thoroughly explored. As a result, understanding their internal working knowledge is critical. To solve this problem, visualization provides the ability to understand the internal structure and workflow of trained machine learning models. Meanwhile, the interpretability of neural networks is tedious due to the large number of parameters. In particular, the internal hidden state information of the recurrent neural network models is yet to be explored. Hence, this thesis introduces a visual analytics tool for analyzing and examining the internal hidden state information of recurrent neural network models. This tool allows an interactive exploration of the hidden state information, which is processed by the neural layers when an input sequence is provided to it. This tool aids in exploring the behavior of a recurrent neural network model when demonstrated with turbulence data from an aerodynamics domain. To achieve better interpretability, we assess and compare the training of the gated recurrent unit model for a number of hyperparameter configurations using various visualization approaches such as dimensionality reduction, heat maps, and parallel coordinates plots.

Keywords: Visual analytics, Visualization, Machine learning, Regression, Recurrent neural networks, Gated recurrent units, Hidden states, Interpretability, Turbulence data

Contents

List of Figures

List of Tables

Acronyms

AI	Artificial Intelligence. 22
ANN	Artificial Neural Networks. 23
CNN	Convolutional Neural Networks. 26
CSS	Cascading Style Sheets. 39
DL	Deep Learning. 18
DNS	Direct Numerical Simulation. 30
GRU	Gated Recurrent Unit. 9
HTML	HyperText Markup Language. 9
JSON	JavaScript Object Notation. 43
KL	Kullback-Leibler. 36
LES	Large Eddy Simulations. 30
LSTM	Long Short-Term Memory. 27
ML	Machine Learning. 17
MLP	Multilayer Perceptron. 32
NLP	Natural Language Processing. 18
NN	Neural Networks. 18
PCA	Principal component analysis. 10
ReLU	Rectified Linear Unit. 11
RMSE	Root Mean Square Error. 9
RNN	Recurrent Neural Networks. 17
SVG	Scalable Vector Graphics. 19
t-SNE	t-Distributed Stochastic Neighbor Embedding. 9
UMAP	Uniform Manifold Approximation and Projection. 10

1 Introduction

1.1 Problem Statement

Over the last few decades, Machine Learning (ML) methods have been rapidly growing and transforming many fields in the area of science. In particular, it has surpassed the state-of-the-art methods in many fields of computer science, which have significant applications such as image detection, image segmentation, and machine translation. Unfortunately, trained ML models are often considered black box models, making it difficult to extract the learned behaviour from the trained model in a form that humans can understand. As a result of this black box nature, they are preferred less in life-critical applications such as aerodynamics, medical, and the healthcare industry. However, this issue can be addressed by visualizing the trained ML models.

Visualization has been one of the ways to understand such trained models[CMJ+20], as it aids to obtain a promising insight into the behaviour of the model. This understanding improves the interpretation of the model, leading to the increase in its preference in safety-critical domains. In other words, interpretability, also known as explainability, answers the questions of *what* the model has learned from the data and *how* the model behaves for certain input. Additionally, this visualizing approach of ML models when demonstrated with a dataset helps to understand the underlying pattern of the dataset. More specifically, visual analytics [KMS+08] helps to visualize high-dimensional data in various coordinated views. As a consequence of this visual approach, the useful insight learned about the model makes it reliable and ensures its correctness. Thus, the credibility of the model increases. The hidden pattern learned about the dataset can be used to make better decisions with the data and be helpful in further research on the respective field. In particular, some of the ML models like random forest and decision trees are easy to interpret without a visual approach. However, neural networks are difficult to interpret, particularly Recurrent Neural Networks (RNN) models, which are often used for sequential data, are challenging to interpret. The more parameters involved in the computation of the output prediction also contribute to its difficulty. Additionally, the hyperparameters of a model play a major role in influencing the output predictions of the model. Thus, the issue of interpretability in neural networks [LWLZ17] should be investigated. This implies that a visual analytics approach to neural networks is required in order to achieve interpretability.

The interpretability of RNN models can be achieved by visualizing the internal hidden states of the RNN models. An internal hidden state can be referred to as a snapshot of the information learned by the model from the input sequence data. As a result, this thesis is an attempt to develop a visual analytics tool that can visualize the internal hidden states of a RNN. Therefore, achieving the interpretation of the RNN model by applying various visualization approaches such as dimensionality reduction, heat maps, and parallel coordinates plots . This tool is demonstrated using turbulence data from the aerodynamics domain, which aids in understanding the feasibility of the tool. Additionally, this helps to understand the nature of the turbulence dataset.

1.2 Motivation and Goal

ML models have the capability to learn intricate hidden patterns from the input data, which allows them to make predictions on unseen data. This capability allows them to deal with complex problems more easily and effectively than the traditional methods. Additionally this capability to learn the hidden patterns is one of the reasons for their better performance when compared to the traditional methods. Unfortunately, the reasons behind their predictions are not clear, which makes them less favorable in low error tolerant domains like the military and aerospace industry. There should be a minimum amount of internal information of what the model had learned from the input data is needed. Based on the information about its internal working they can be evaluated and it can answer the question how reliable the model can perform for unseen input data. As a result the decision to consider a model for applying in critical domains can easily be determined. Thus, the interpretability of the ML models should be well explored, as it is one of the important factors for selecting a model to deploy in critical applications. One such information can be obtained by visualizing the models through a visual analytics tool. Deep Learning (DL) being a subfield of ML has got a lot of space to be explored in terms of interpretability. Neural Networks (NN) is the backbone of DL algorithms and it is of various types. In particular, RNN one of the types of NN, which is commonly used for sequential data should be studied further. It has an internal structure that acts like a memory and remembers past information. Among all the types of NN, RNN is the only type that can remember the previous information of a sequential input and use that information to predict the next output of unseen data. While the other types of NN do not have the capability to remember the previous information of input for predicting an output. Therefore such a complex structure of NN has a lot to be researched in order to understand its internal working. Such internal working can be studied by visualizing it through a visual analytics tool.

The tool may be created for a variety of objectives that align with our goals. It can be developed either for classification or regression task. It can also be developed to a particular dataset, which is something we want to investigate. Additionally, it can be developed for the particular domain or industry. The current research works on the visual analytics tool have been more focused for classification tasks. Therefore, the motivation behind the thesis topic is to develop a tool for regression task. Unlike machine level translation or Natural Language Processing (NLP), the attempt for such a visual analytics tool to explore aerodynamics domain is very less.

The goal of this thesis is to develop a visual analytics tool that aids to interpret and explore the internal hidden states of RNN models. Therefore this thesis attempts to provide a visual analytics to understand deep about the working of RNN models that predicts the closure terms of an aerodynamics domain.

The main focus of this thesis is to develop a visual analytics tool that serves with the following features:

- Provision to have an interactive environment to choose different values for the hyperparameters.
- Comparing models having different configurations.
- Further deep understanding of the internal information of a model.
- Feature to view the input data.

In order to achieve this goal, contributions made in fields like visual analytics, machine learning, and dimensionality reduction are referred to and the tool was developed. The tool mainly projects the internal hidden state as Scalable Vector Graphics (SVG) after applying dimensionality reduction techniques like t-SNE, PCA, UMAP and attempts to serve the features listed.

1.3 Outline

The outline describes the structure of this thesis work. This thesis consists of seven chapters and which are organized as follows:

Chapter 2 provides an introduction on visual analytics, artificial intelligence, machine learning, neural networks, turbulence data, and the turbulence dataset used. The explained topics are pre-requisites that are needed to understand this thesis.

Chapter 3 discusses previous works, concepts of visual analytics in machine learning, and contributions that are related to this thesis.

Chapter 4 discusses about the approaches like dimensionality reduction techniques, feature importance, K-means clustering, and zeroth hidden states. These approaches are applied to develop this visual analytics tool.

Chapter 5 explains the hardware, software, and software design of the visual analytics tool. Additionally, it also explains the architecture and the visualization features of the tool.

Chapter 6 discusses the workflow and the results obtained by implementing the tool with turbulence data. This chapter includes the illustration of comparison between models and the overall discussion about the comparison.

Chapter 7 summarizes this thesis and draws conclusion with some possible future scopes.

2 Background

This background chapter offers the necessary knowledge and details required to lay the foundations for this master thesis work. There are five sections in this chapter. Section 2.1 introduces the concept of visual analytics that forms the basis of the visual analytics tool. The definition of artificial intelligence and its uses, an introduction to machine learning and its categories are explained in Section 2.2. Section 2.3 provides an overview about NN. Section 2.4 explains about RNN and its working. An overview about turbulence data is provided in Section 2.5. Finally, Section 2.6 discusses the dataset used for this thesis.

2.1 Visual Analytics

As the current world is moving towards the betterment of the information era, all the disciplines in the field of science are experiencing an increase in the volume of data. This volume is rapidly increasing on a daily basis, resulting in the collection of massive amounts of data. Therefore, humans have the responsibility to handle, store, and use this data for the growth and advancement of technology. This massive amount of data is useless until some useful information is extracted from it. Thus, visualization aids in transforming this massive data into useful information. Humans can process 1-D and 2-D data easily, while higher dimensional data is hard to imagine and process. Therefore, visualization provides an effective solution by visualizing the higher dimensional data and aiding in processing it easily by the human brain. In particular, the growth of ML models is rapidly transforming many research areas by exceeding the performance of state-of-the-art approaches. However, these models often deal with higher dimensional datasets, which are hard to visualize. Thus, visual analytics offers an accountable solution to this problem.

Cook and Thomas, some of the pioneers of the visual analytics field, have defined visual analytics as "the science of analytical reasoning facilitated by interactive visual interfaces"[CT05]. They have explained it as a multidimensional field involving the visual presentation of data that aids humans in achieving an analytical reason from the data after having an interaction with it. In recent years, this field has seen unprecedented growth and is expected to grow further. Additionally, visual analytics allows for the transformation of large amounts of data into a format suitable for analysis. This allows humans to have a better understanding of the data, allowing them to draw more accurate inferences. As a result, visual analytics has made major contributions to the fields of science, information technology, and safety-critical applications such as the military, aerospace, and healthcare.

According to Keim et al. [KKE10], visual analytics has been defined as "Visual analytics combines automated analysis techniques with interactive visualizations for an effective understanding, reasoning and decision making on the basis of very large and complex datasets". They have described the visual analytic process as well as the stages involved in transforming data to gain insightful knowledge about it. Figure 2.1 illustrates the steps of the visual analytic process. Often,

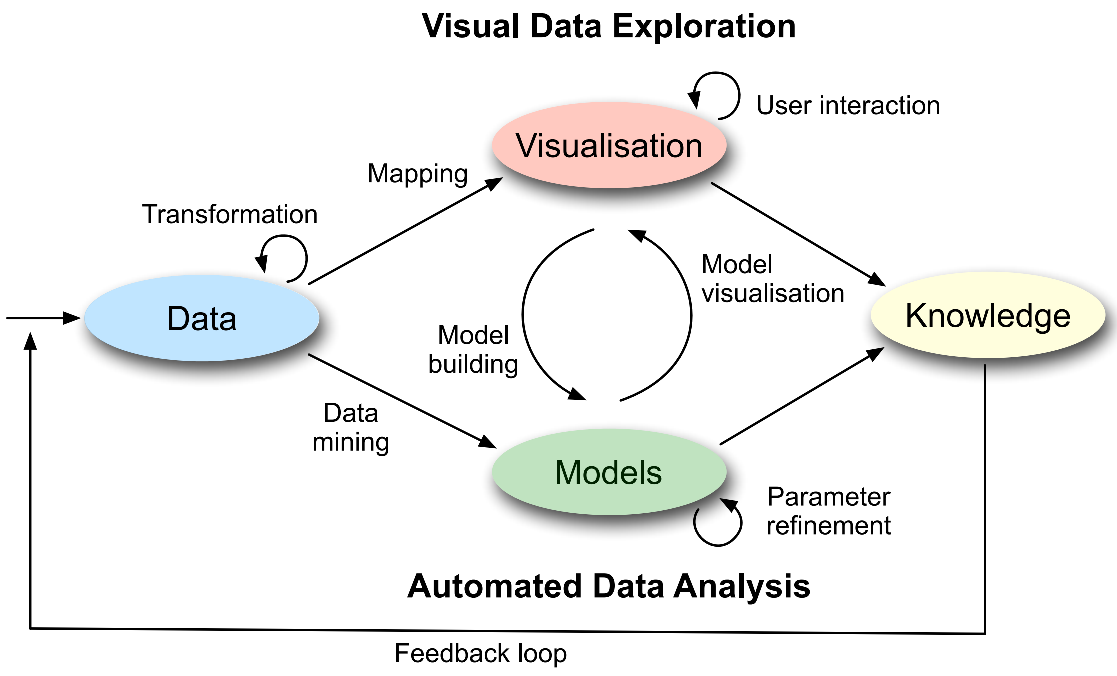


Figure 2.1: Visual Analytic process illustrating the steps involved to obtain knowledge from data either through visualization or through models [KKE10].

real-world data is not suitable to be applied directly to a model or to a visualization method. Hence, the data should be transformed and preprocessed. Then, in order to obtain knowledge about the data, two choices are available. The choices are: visual data exploration or automated data analysis. In visual data exploration, the transformed data can be visualized by the user after having an interaction with it. The transformed data can be mapped to the visualization. The information from the transformed data is extracted by mapping it to visual representations resulting from human interaction. In the automated data analysis, the transformed data can be created as a model using a data mining technique by finding the underlying pattern. The model can be refined by changing the parameters used for building the model. Therefore, both these choices lead to the discovery of the final knowledge about the data. These steps can be carried out in a feedback loop until a defined, structured idea about the data is gained.

2.2 Machine Learning

In 1955, John McCarthy coined the term Artificial Intelligence (AI) and defined it as “the science and engineering of making intelligent machines”[Tom86]. AI refers to the creating and upgrading of intelligent systems that contributes a lot to the field of science and it is found successful in accomplishing complex tasks in simpler way. Currently evolving fields of research like speech recognition, NLP, computer vision are some of the well-known applications of AI. Apart from the above said there are also numerous applications of AI that exists in almost all the domains like medical, e-commerce, and so on. The subdomains of AI are clearly shown in Fig.2.2. The most prominent and successful type of AI is machine learning.

“Machine learning is the study of computer algorithms that provides systems the ability to automatically learn and improve from experience” is the definition of ML quoted by Sah[Sah20]. ML is the sub-field of AI that has the ability to grasp the underlying hidden patterns from a data which humans can not be able to grasp. Mainly, machine learning can be partitioned into three categories based on whether it uses the knowledge of labelled data or not. The categories are namely supervised learning [CCD08], unsupervised learning [Bar89] and reinforcement learning [KLM96].

- **Supervised Learning:** ML model learns the underlying pattern with the help of labelled data during the training phase. In the labelled data, each input is mapped to its perfect output value. The predicted value by the Artificial Neural Networks (ANN) can be compared with this ground truth to evaluate the accuracy of the model. The workflow of supervised learning is shown in the Fig.2.3 that indicates the model trained with input data and ground truth is fed with new unseen data to predict the output. Labelled data can also be indicated as annotated data. Support vector machine (SVM), decision forest, and naive bayesian model are some of the examples of supervised learning. This thesis mainly focuses on supervised learning. Supervised learning is further divided into
  - **Classification:** Model learns to segregate data based on features by labeling the data. Based on the learned characteristics, the model classify the inputs. Using the input data, the model learns the decision boundary that separates the distinct classes. The unseen data is categorised using the previously learned decision boundary. The model maps a data point to a specific class, which means that the output value is discrete, with a finite number of values corresponding to the number of different classes. For example, identifying an image as either dog or cat.
  - **Regression:** Model learns to approximate a function $f(x)$ from input features x to a continuous output feature $\hat{y} = f(x)$. This function $f(x)$ is approximated in order to reduce the error between the function output y and the original output feature $\hat{y}$. This error is computed using loss functions L and is used to evaluate the accuracy of the model. The value of the output is continuous and thus has an infinite number of values. For instance, a prediction of the mileage of an automobile or the price of stocks in the stock market.
- **Unsupervised Learning:** ML model learns the underlying pattern without the help of labelled data. The machine learning model attempts to detect correlations in the data automatically by extracting important features and evaluating its structure. Clustering and anomaly detection are some of the examples that falls under this category.
- **Reinforcement Learning:** Reinforcement learning is based on an agent that interacts through actions with an environment. It is optimized through rewards or punishment. The agent aims to find the best solution to the problem. The incentives are provided at the end of each training episode (i.e., after a large number of actions), and the reward is determined by performance of the agent.

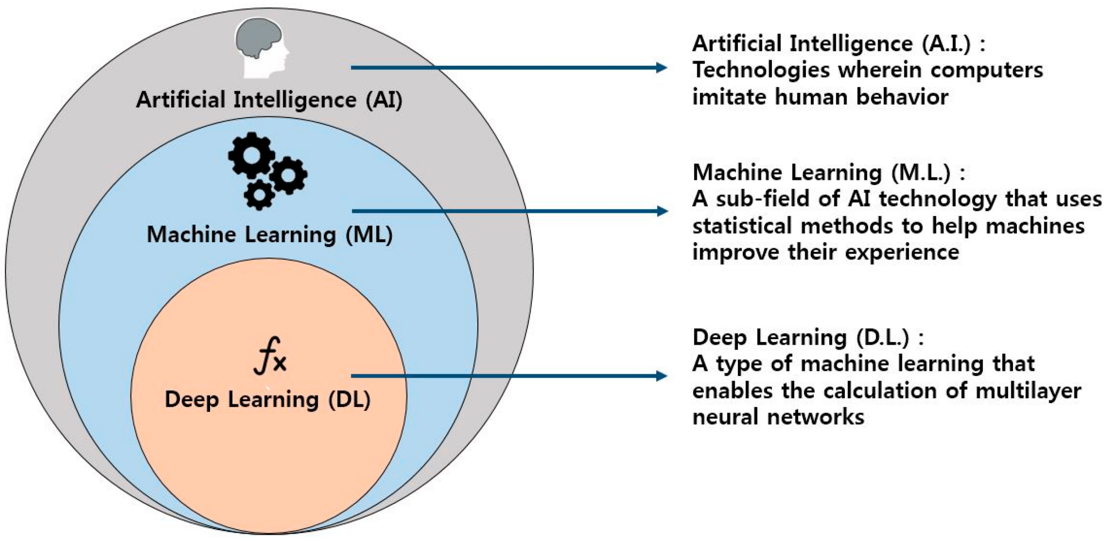

Artificial Intelligence (AI)
Machine Learning (ML)
Deep Learning (DL)
Artificial Intelligence (A.I.) : Technologies wherein computers imitate human behavior
Machine Learning (M.L.) : A sub-field of AI technology that uses statistical methods to help machines improve their experience
Deep Learning (D.L.) : A type of machine learning that enables the calculation of multilayer neural networks

Figure 2.2: Artificial Intelligence and its subdomains [LL21].

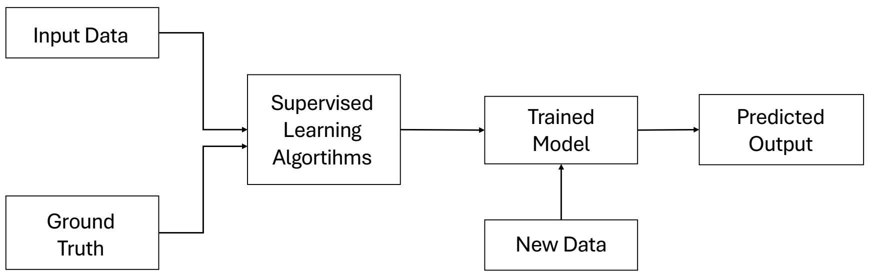


Figure 2.3: Workflow for supervised machine learning.

2.3 Neural Networks

DL is a sub-field of ML that is inspired by the structure of the human brain. In recent years DL have been used for a broad range of applications. This is due to the fact that they outperform many of the existing conventional methods in their respective fields. In particular ANN, was inspired and developed after the functioning of the human brain . Nonlinear behaviour is the attribute because of which the neurons fire only if the input excitation exceeds a certain threshold. In particular, the number of features learnt by ANN, corresponds to the number of input neurons independent of the depth of ANN. Generally, the aim of ANN is to learn a mapping function from an input to an output. This mapping function can be for either a classification or regression task. Meanwhile, most of the tasks in the real world are nonlinear.

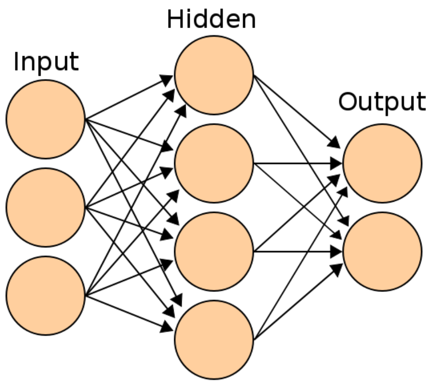


Figure 2.4: Schematic architecture of Artificial Neural Network with input, hidden, and output layer [Com20a].

The ANN contains nodes which are called neurons, that receives information from previous neurons and pass it to the succeeding neurons. Thus, inputs are processed by neural networks by sequentially executing the operation described by each layer on the input data, resulting in an activation vector that serves as the input for the next layer. Overall, the input information flows through the input layer to the output layer through hidden layers. Each and every neuron is connected to each other by a weighted edge. Figure 2.4 shows the structure of an ANN with an input layer comprising three neurons, a single hidden layer with four neurons and an output layer with two neurons[1].

The relationship between weights and offset is known as bias, and the intermediate output y can be given by Eq. (2.1)

$$y = \sum_i w_i x_i + b. \tag{2.1}$$

There can be multiple inputs for each neuron in an ANN. Each neuron computes the intermediate output by multiplying the weighted sum of inputs and summing them with bias. Equation (2.2) shows the calculation of the intermediate output of a single neuron consisting of three inputs (x_1, x_2, x_3) with weights (w_1, w_2, w_3) and offset b and is given by

$$y = (w_1 x_1) + (w_2 x_2) + (w_3 x_3) + b. \tag{2.2}$$

The output of an activation function gives the output of the neuron. It is given by

$$a = \varphi(y). \tag{2.3}$$

Equation (2.3) shows the output for a single neuron where a indicates the output of the neuron and $\varphi()$ refers to the activation function. The Loss function is another important concept related with ANN that is used to evaluate the accuracy or performance of the prediction made by the model. It is used to determine the difference between predicted output and the original ground-truth. Often categorical cross entropy is used for classification tasks while the mean squared error or root mean squared error is used for regression tasks. The optimization process of the loss function is known as

[1]Artificial neural network.svg by Cburnett is licensed under CC BY-SA 3.0

training or learning and is accomplished by computing the gradient vector of the cost function with respect to the weights, which is known as backpropagation. Overall, the training and optimization are used to find the optimal free parameters such as weights and biases for the given task, i.e. the weights aids to minimize the loss on the given dataset.

2.3.1 Activation Functions

The neurons compute the intermediate output by multiplying the weighted sum of inputs and summing them with bias but they lack the ability to select a firing pattern. The activation function is a continuous function that determines how much the neuron should be stimulated. As a result it introduces non-linearity into the network, allowing the neural network to learn nonlinear functional relationships between input and output. Although, there are several activation functions, this thesis focuses on the following three activation functions.

- **ReLU:** ReLU is the most commonly used activation function in ANN. Its value ranges between 0 and infinity. When the input is positive, the function behaves linearly, while when the input is negative, the function returns zero and it is shown in the Eq.(2.4)

$$a(z) = max(0, z). \tag{2.4}$$

- **Sigmoid:** The range of the output is between 0 and 1, hence this function is commonly used for models with a probability output. The probability is constantly between 0 and 1. Equation (2.5) shows the expression of the sigmoid activation function.

$$\sigma(z) = \frac{1}{1 + e^{-z}}. \tag{2.5}$$

- **Hyperbolic Tangent:** The hyperbolic tangent is often known as tanh, and the range of the output is between -1 and 1. Equation (2.6) shows the expression of the hyperbolic tangent activation function.

$$a(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}. \tag{2.6}$$

2.4 Recurrent Neural Network

RNN is one of the ANN architectures that are frequently used for sequential data. Sequential data can be referred to as a series of data points ordered in sequences, where the order of the data points is important. Recently emerging fields like machine translation,and speech recognition are some of the prominent applications of RNN. Unlike Convolutional Neural Networks (CNN)[ON15], RNN remembers its previous input due to its internal memory. In RNN the information cycles through a loop. At each step RNN has two inputs: its present input and its hidden state. Hence with the help of internal memory and the ability of having two inputs at each timestep, RNN performs well in handling crucial forth coming information of the sequential data. RNN assigns values to parameters like weights and biases to the current and also to its memory. Then during the training process the weights and biases are altered through Backpropagation Through Time [Wer90]. Figure 2.5

shows the structure of the RNN with both folded and unfolded in time[2]. This implies that how the sequential input data is unfolded to produce a hidden state from the current input and its previous hidden state, or its previous output depending on the RNN type. The structure shows that they have an internal loop that helps to add the immediate past to the present, allowing RNN to draw knowledge from the past and relate it to the present. In the Fig.2.5, the green components represent input, the blue components represent hidden state, and the pink components represent output. From the input x to the output o, the RNN store the past-present relation in an hidden state h. The hidden state is updated through loops for the sequence of data. Here x_t and h_t represent the input and hidden state at the current timestep, t, while x_{t-1} and h_{t-1} represent the input and hidden state at the previous timestep, $t-1$. The hidden state update involves the previous hidden state h_{t-1}, some function parametrized by the weights and the input vector at time t as shown in the Eq.(2.7). The function f can be a non linear transformation function such as ReLU or tanh. U and V corresponds to the weight matrices of input vector and previous hidden state vector. Equation.(2.8) shows the calculation of hidden state using U, V, and b_h. The output o_t is calculated using the updated hidden state h_t and the weight matrix W as shown in the Eq.(2.9). b_h and b_o are biases.

$$h_t = f(h_{(t-1)}, x_t). \tag{2.7}$$

$$h_t = f(Ux_t + Vh_{(t-1)} + b_h). \tag{2.8}$$

$$o_t = Wh_t + b_y. \tag{2.9}$$

In the forward propagation, the inputs are forwarded with the time, updating the state based on the input and the previous state, generating o_t at the timestep and computing loss at each timestep t. The total loss is the calculation of the sum of each individual loss. backpropagation through time is accomplished in two steps in RNN. In the first step, backpropagation is performed at each individual time step. In the second step, backpropagation is performed across all time steps such that errors are propagated back in time. Backpropagation through time involves a lot of weight matrix multiplication with a gradient of activation functions. This leads to a vanishing gradient and an exploding gradient problem. Both these problems can be overcome with the help of gated cells like GRU and Long Short-Term Memory (LSTM). When the model has any one of these problems, it makes the model harder to train the weights, so it affects all the further weights.

2.4.1 Gated Recurrent Unit

GRU network [CVG+14] is a type of recurrent neural network that is widely used for sequential or temporal data. It resolves the two main issues, the vanishing gradient and the exploding gradient problems of normal RNN. These issues make it difficult for RNN to learn long-term data

[2]Recurrent neural network unfold.svg by Ixnay is licensed under CC BY-SA 4.0

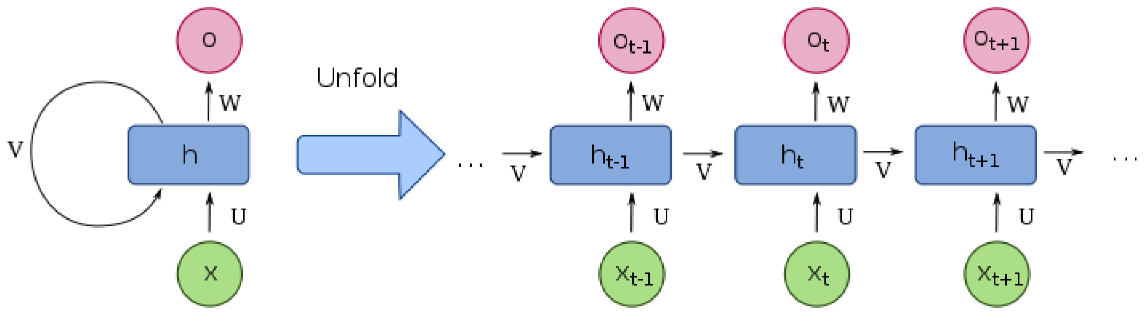


Figure 2.5: Schematic architecture of Recurrent Neural Network. **Left**: Folded in time showing input, hidden state and output. **Right**: Unfolded in time showing previous and current inputs, hidden states [Com22].

dependencies. GRU is able to mitigate these issues with the help of gated architecture. GRU structure comprises of two gates, namely the reset gate and the update gate. The GRU cell is the fundamental building unit of GRU. Generally, in GRU, hidden state is the output. This acts as the memory of the network. The values of these hidden states are updated every time when a new element of data from the temporal sequence is fed to the network. In every timestep, it takes an input from the current timestep and a hidden state from previous timesteps to process a new hidden state, which then passes it to the next timestep. Gates are the components that help in remembering the information. They also control the flow of information from one layer to the next layer.

A reset gate is used to reset the hidden state information when needed, while an update gate is used to update or pass exactly which hidden state information needs to be sent to the next layer. Figure 2.6 shows a GRU cell and its internal structure with update and reset gate [3].

The reset gate is responsible for short-term memory. The output of the reset gate will range from 0 to 1 as it uses the simgoid function as an activation function. Here the sigmoid function is multiplied with the sum of the product of weight matrices of the reset gate along with current input data and the previous hidden state information. The equation of the reset gate is shown in Eq.(2.10), where U_r and W_r are weight matrices of the reset gate.

$$r_t = \sigma(x_t U_r + h_{(t-1)} W_r). \tag{2.10}$$

The update gate is responsible for long-term memory. The output of the update gate will range from 0 to 1 as it uses the sigmoid function as an activation function. Here the sigmoid function is multiplied with the sum of the product of weight matrices of the update gate along with the current input data and the previous hidden state information. The equation of the update gate is shown in Eq.(2.11), where U_z and W_z are weight matrices of the update gate.

$$z_t = \sigma(x_t U_z + h_{(t-1)} W_z). \tag{2.11}$$

[3]Gated Recurrent Unit, type 2.svg by Jebald is licensed under CC BY-SA 4.0

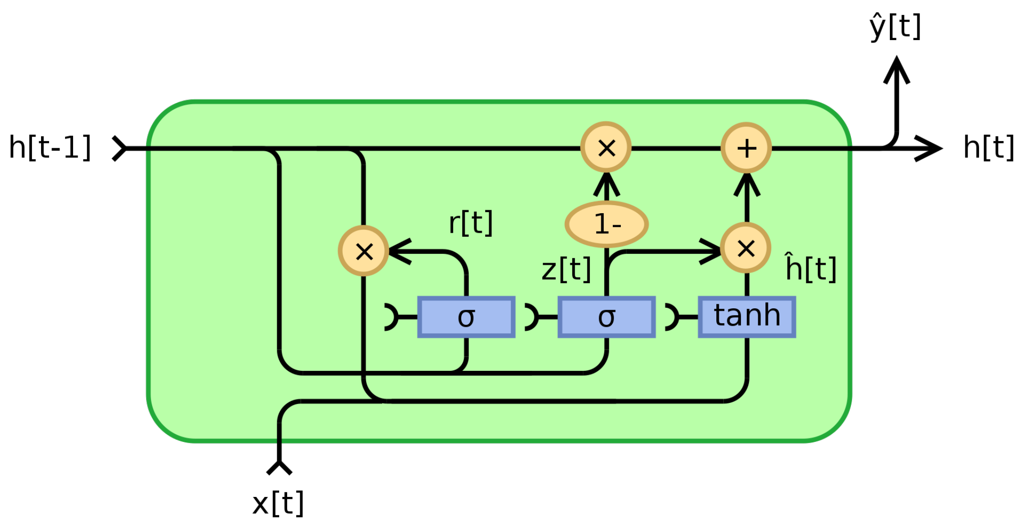


Figure 2.6: Structure of Gated Recurrent Unit with update and reset gate [Com20b].

GRU calculates the hidden states of an input by a two-step process. The first step is to find the candidate hidden state, $\hat{h}_t$. It is achieved by taking the input and the hidden state from the previous timestep, $t-1$ which is then multiplied with the reset gate output and passing this entire information to the hyperbolic tangent function. Thus, the candidate hidden state can be obtained from Eq.(2.12) and is shown as

$$\hat{h}_t = \tanh(x_t U_g + (r_t \odot h_{(t-1)}) W_g). \tag{2.12}$$

Based on the value of the reset gate output, the influence of the previous hidden state on the candidate hidden state can be controlled. When the value of the reset gate output is 1, it indicates the entire information from the previous hidden state is to be considered. If the value is 0, it indicates that the information from the previous hidden state is not considered.

Then the second step is to calculate the current hidden state using the candidate hidden state. Here, the update gate, which is responsible for long-term memory, plays a vital role. Based on the value of the update gate, the current hidden state is determined. The update gate controls both the historical information from the previous hidden state and the new information from the candidate hidden state. Thus, the hidden state can be obtained from Eq.(2.13) and is shown as

$$h_t = U_t \odot h_{(t-1)} + (1 - U_t) \odot \hat{h}_t. \tag{2.13}$$

When the value of the output gate is zero, the first term in the equation vanishes, leading to the fact that the hidden state will not have information from the previous hidden state. Meanwhile, the second term becomes almost one, meaning the hidden state at the current timestep will consist of information from the candidate hidden state. In case, if the value of the update gate is one, then the second term of the equation becomes zero and the hidden state will depend only on the first term, indicating the information from the previous hidden state contributes to the current hidden state. Unlike LSTM[SM19], an another frequently used recurrent neural network model GRU does not have an output gate. This is one of the reasons that makes it easier and faster to train than LSTM.

2.5 Turbulence

In the field of fluid dynamics, turbulence is described as fluid motion characterized by changes in pressure and flow velocity. Turbulence can be described by the Reynolds number and its range of flow on active scales increases with the increase in Reynolds number. At high Reynolds numbers, Direct Numerical Simulation (DNS), a high-fidelity solution of turbulent flows, is computationally expensive. Hence, in order to mitigate the computational effort, the method of Large Eddy Simulations (LES) is used. LES relies on a filter operation to resolve only the largest scales of motion in the flow field. Meanwhile, the effects of the non-resolved part are modelled. The filtering of the non-linear convective terms of the Navier-Stokes equations invokes a closure problem. As a result, an additional model term gets introduced in the coarse-scale equation. The models used in LES approach are derived based on physical or mathematical considerations.

2.6 Dataset Description

Kurz and Beck in [KB20] derived a machine learning framework for LES closure models. The influence of different filter forms was also given importance while deriving this framework. The training data was generated by applying several distinct LES filter functions to DNS data of decaying homogeneous isotropic turbulence (DHIT). In order to achieve this, the DNS solution was projected onto a discontinuous Galerkin (DG) or finite volume (FV) flavored representation on a coarsened mesh. This overall process resembled the implicitly filtered LES approach, where the discretization acts as an implicit LES filter. Finally, to resolve any influences of the LES discretization, a global Fourier cutoff filter was used to ensure the filtered solution is perfectly resolved by the underlying numerical scheme. Finally, GRU, one of the RNN type was trained to learn the unknown closure terms from data for all filter forms and achieved an accuracy of 99.9% cross-correlation between the predicted and the exact closure terms. Additionally, the trained GRU networks generalize well across different LES resolutions and different filter functions.

A DNS database has been built from an ensemble of runs and the exact closure terms for various LES filter functions have been computed. It was also demonstrated that the computed closure terms are able to retrieve the model terms filtered DNS solution at each timestep. However, more than thousands of computations were done maximum of up to 21 timesteps were used for the prediction. Thus indicating the sequential data with 21 timesteps. Basically, the coarse scale data with velocities and pressure are stored in .h5 format. By changing the time increment value between the .h5 files, various timesteps up to 21 timesteps can be obtained. Hence, this shows that many input values of different sequence lengths can be fed to the model in order to predict one output.

3 Related Works

This chapter discusses the key contributions made by previous researchers in the fields of machine learning and visual analytics. In addition, there is a discussion regarding existing visual analytics tools.

Interactive visualization is one way to explore the behaviour of ML models. The results obtained from these visualizations should be trusted to understand the behaviour of ML models. Chatzimparmpas et al. presented a state-of-the-art report that promotes the trust on the ML models with the use of visualizations [CMJ+20]. The authors have built the report based on the end results obtained from the visual analytics field and various visualizations such as dimensionality reductions and data mining. The interactive visualization should be able to explain various factors such as explainability, comparison, and algorithms. The features that a visual analytics tool should address were studied.

Garcia et al. investigated the visualization of the internal hidden states of RNN for classification tasks in [GMW21]. The authors have provided a promising insight into the interpretation of the LSTM model by a visual analytics tool. Two NLP datasets, IMDB and Reuters datasets available in Keras [Cho15] were taken. In the IMDB dataset, the interpretation of the model for a sentiment analysis use case to classify whether a sentence is positive or negative was explored. Moreover, classification among five different classes in the Reuters dataset was illustrated using this visual analytics tool. Both the use cases were carried out by projecting the internal hidden states of the model for each input data point in the dataset by reducing its higher dimensions to 2-D dimension using dimensionality reduction techniques like t-SNE. In addition, other visualization approaches such as heat maps were used to visualize the expected prediction of the classifier. This has provided enough reasons for the decision made by the model for predicting the output.

In [MCZ+17], Ming et al. explored the display of the internal hidden states of RNN models for a NLP task using the Penn Tree Bank dataset [Mar94]. The authors suggested a method for interpreting the function of a hidden state using strongly correlated words from the input. By providing a word as an input, the correlation was derived from the expected update of the hidden state. A bipartite graph was generated as a result of modeling the relationship between the hidden states and words. Furthermore, weighted edges connected the two types of nodes, hidden states and words. The bipartite graphs are grouped and visualized as memory chips and word clouds to provide an organized view of the input data. Based on the above aggregated data, a visualization at the phrase level was also developed.

The visualization of the hidden states of the LSTM model provided with clinical accelerometer data of 2977 participants for two years was visualized in the study performed on [YRXZ20]. Since an accelerometer is a human-wearable device, it detects human mobility and so monitors health condition in a time series format. LSTM with three recurrent layers functioned as an encoder and was provided with accelerometer data. The hidden states and hidden layers for a certain time frame were investigated using a multiline plot. Furthermore, the input data from the recurrent layers with the same patterns was compared and visualized. As a result, possible relationships between the data have been investigated. The Sankey plot was used to depict the pattern of transition between alive and deceased people. The visual analytics tool aids in the visualization of accelerometer data as well as the matching of input samples with related patterns.

Shen et al. in [SWJ+20] investigated the attempt to explain the behaviour of RNN on multidimensional time series data. For the demonstration, a weather forecast [SCW+15] and an air pollution prediction [OPM16] dataset were considered. They proposed a technique for predicting the hidden state response based on the influence of feature selection on the output distribution of the hidden unit. The hidden units and features were then clustered based on the embedding vectors. As a result, the behaviour of the RNN model at the global and individual levels was examined. In [VV99], Van Wijk and Van Selow studied the patterns and trends in the time series data for days, weeks, and years. The patterns in the data were predicted using a stochastic model and then converted from the time domain to scale space. By combining the patterns into clusters, the overall pattern was represented as graphs.

Reddy et al. in [RRL+20] applied two dimensionality reduction techniques such as Linear Discriminant Analysis (LDA) and PCA on four ML algorithms, including Naive Bayes Classifier, Decision Tree Induction, Support Vector Machine (SVM), and Random Forest Classifier. The study was carried out on Cardiotocography dataset[DG17] for a classification task. The results indicated that PCA outperforms LDA when the dimensions of the datasets are large. The ML algorithms were evaluated using metrics such as accuracy, sensitivity, and specificity. Therefore, the effects of applying dimensionality reduction techniques on high dimensional datasets were explored.

Kurz and Beck proposed a machine learning framework for LES [PC96] closure models in which the influence of different filter forms was examined in [KB20]. LES is a mathematical model for turbulence that is used in the aerodynamics domain to reduce computational effort. The approach of using LES is based on filter operations, which results in the introduction of additional model terms known as closure terms. Technically, DNS training data was created from the DNS database and applied to the GRU model. They attained an accuracy of 99.9% in determining the cross-correlation between the expected and exact closure terms after training the model on DNS data. Furthermore, trained GRU networks generalize effectively across multiple LES resolutions and filter functions. Thus, the turbulence data was applied to the GRU model, one of the types of RNN models that were extensively researched and succeeded in their prediction of closure terms with higher precision.

Srinivasan et al. in [SGA+19] investigated the abilities of neural networks to predict turbulent flows. The training data was generated for LSTM and Multilayer Perceptron (MLP) networks. The architecture of the neural networks was varied by changing numerous factors such as the number of layers, number of hidden units per layer, dimension of the input, and weight initialization and activation functions to discover the best configurations for turbulence flow prediction. As a result of this change, the LSTM network outperformed the MLP because of its capability to understand the sequential nature of the data. However, they had considered a low-order representation of near-wall turbulence. They had contributed the idea of a machine-learning framework to develop large-eddy simulations with accurate and efficient data-driven subgrid-scale models.

The majority of the previous research in this section focuses on visualizing the hidden state of RNN models for NLP or classification tasks. As a result, there has been minimal investigation into the temporal information learned by RNN models for time series data. Therefore, there are fewer visual analytics tools created for regression tasks than for classification activities. Furthermore, existing work does not focus on understanding the hidden states of RNN models while training the models with a wide range of hyperparameter values. As a result, the effect of hyperparameters on expected output for various combinations or ranges of hyperparameters was not investigated. Hence, we propose an approach to develop a visual analytics tool that aids in understanding the hidden state functions for a range of hyperparameter values. Overall, the influence of the hyperparameters on the predicted output for a time series task through the hidden state can be explored. Furthermore, our approach is more focused on a regression task.

4 Methods

This chapter gives a detailed information about the dimensionality reduction techniques in section 4.1. Feature importance is explained in section 4.2. K-means clustering is explained in the section 4.3. The information about hidden states and zeroth hidden states is provided in the section 4.4.

4.1 Dimensionality Reduction Techniques

In ML, the number of features or input variables of a dataset is often referred to as its dimensionality. When the number of input variables increases, it becomes tedious to visualize the data. Dimensionality reduction can be explained as the process of selecting the set of attributes or features while retaining most of the variance in the dataset. In other words, it can be explained as a mathematical procedure that helps to efficiently reduce the most redundant high dimensional data to a smaller number of variables with a minimum loss of information. Its advantages compensate for the loss. The following are the advantages of dimensionality reduction:

- Data visualization is easier if dimensionality reduction is applied on a dataset.
- Non-linear data can be transformed into linear separable form.
- It prevents the model from encountering the curse of dimensionality problem [XLX17].

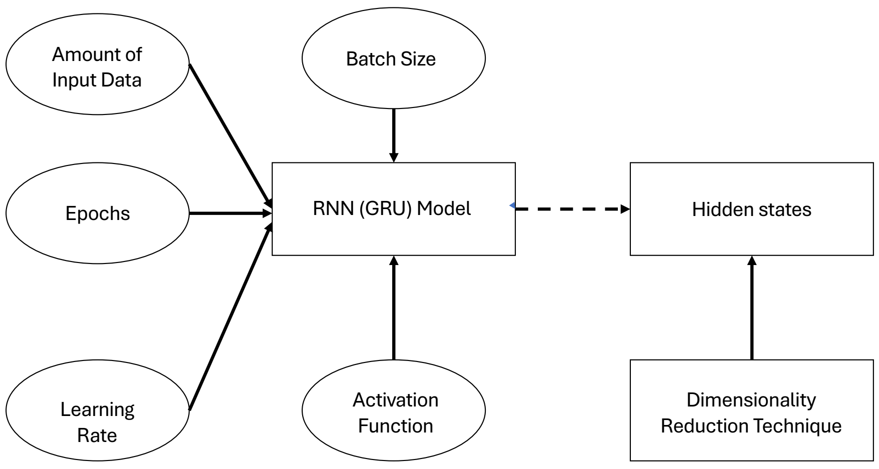


Figure 4.1: Approach for applying dimensionality reduction techniques on hidden states.

There are several dimensionality reduction techniques that are commonly used in different research fields. This thesis focuses on t-SNE, PCA, and UMAP.

t-SNE is a non-linear dimensionality reduction algorithm used for investigating multi-dimensional data by mapping it to two or three dimensions, which is suitable for human observation. The main purpose of using t-SNE for dimensionality reduction is to preserve small pairwise distances or local similarities as well as the structure of the data correctly. The t-SNE calculates the similarity measure in both higher dimensional space and lower dimensional space, and then it tries to optimize these two similarity measures using a cost function. The process of applying dimensionality reduction techniques to a dataset involves three steps. The first step is to measure the similarities between data points in higher dimensional spaces. For instance, a bunch of data points scattered in a 2-D space are considered. For each data point, a Gaussian distribution over the center of that point is taken, and the measure of density of all the points under that Gaussian distribution is calculated. This gives a set of probabilities for all data points, which are proportional to similarities. In particular, if two points x_1 and x_2 have equal areas under the gaussian distribution, then the proportions and similarities are equal, and hence we have a local similarity in the structure of its higher dimensional space. The gaussian distribution can be controlled by using perplexity, which influences the variance of the distribution and the number of nearest neighbors. The second step is similar to that of the first step, but we use a Cauchy distribution instead of a Gaussian distribution. This gives the probability in the lower dimension of space. The final step of the t-SNE algorithm is to set the probabilities of the low dimensional space to reflect those of the high dimensional space as best as possible. In order to measure the probability difference between the higher dimension and the lower dimension space the Kullback-Leibler (KL) divergence can be used. This is a way to efficiently compare large dimensional spaces and small dimensional spaces.

PCA is one of the dimensionality reduction techniques that aids in reducing the dimensions of the data while retaining most of the information about the original data. Since it is a linear algorithm, it projects data onto a linear sub-space where the global variance of the entire dataset is maintained. The algorithm tries to identify the principal components, which are the dimensions with maximum variance. A covariance matrix of dimension $d \times d$ is computed using the high-dimensional data of the dataset. d represents the dimension of the dataset. Following that, eigenvalues and their corresponding eigenvectors are computed. Then the eigenvalues are sorted in decreasing order. The eigenvector with the highest eigenvalue represents the first principal component as well as the dimension with higher amount of variance. Finally, the desired number of principal components is selected. These principal components correspond to the dimensions that preserve the maximum data from the dataset. As a result, the most correlated features of the dataset are removed while the uncorrelated features are preserved, resulting in the reduction of the dimensions.

UMAP is another non-linear dimensionality reduction algorithm that reduces the high dimensional data to low dimensions by preserving the local structure of the data. This algorithm consists of two steps. Firstly, the manifold structure of high dimensional data is learned. The learning of this structure includes the process of finding the nearest neighbors of each data point and constructing a graph using the nearest neighbors. Secondly, the learned manifold is projected or mapped into low dimensional space.

The concept of applying dimensionality reduction techniques on the hidden states of RNN is proposed because of the features of each technique. t-SNE, PCA, and UMAP are the three techniques focused in this thesis. Meanwhile, all the three techniques have their own advantage over the other techniques. Since this tool is an attempt to explore internal hidden states of RNN, the techniques that serve in different direction of exploring using non-linear, linear algorithm are considered. Thus, the demonstrated turbulence data can be subjected to explore its behaviour under different dimensionality reduction algorithms. In particular, t-SNE is chosen for its ability to preserve similarities while maintaining local variance by retaining the nearest neighbors adjacent in the low dimensional space. PCA is chosen for its ability to preserve global variance. UMAP is chosen for its ability to preserve the local structure of the dataset. For this proposed method, in particular, five hyperparameters such as amount of input data, epochs, batch size, learning rate, and activation function are considered. Then the model is trained for the combinations of all these hyperparameters for a range of values. Following that, the dimensionality reduction technique such as t-SNE, PCA, and UMAP are applied on the resultant hidden states. Thus, the internal hidden states of the RNN for each input sample as well as the behaviour of turbulence data under different combination of hyperparameter values can be explored.

4.2 Feature Importance

The concept of feature selection and feature importance is proposed to apply to the turbulence dataset. Feature importance represents the identification of the important features by calculating an importance score for all the input features. Applying an explainable AI technique to turbulence data, such as SHAP (SHapley Additive exPlanations) and LIME (Local Interpretable Model-agnostic Explanations), can provide an important insight into the reason why the model predicts a specific output for an input. Thus, the important feature that influences the output predicted by the model can be identified. Furthermore, the SHAP may be better suited for datasets with high dimensions. Meanwhile the important features can be also be identified by the importance score for each learned feature which can provide an useful insight for deciding the features influencing the predicted output. Overall, these techniques allows to find the hidden patterns in the dataset by identifying the features that are responsible for higher influence of the dataset.

4.3 K-means Clustering

K-means clustering is one of the most used unsupervised learning algorithms. It can also be categorized as a partition-based clustering algorithm. The primary aims of K-means clustering are to group comparable data elements into clusters and discover an underlying pattern. In other words, it partitions n data elements into k clusters. In order to achieve this, K-means searches for exactly k number of clusters within a dataset. A cluster is a group of data elements that have been combined together on the basis of the similar characteristics they share. The means in K-means refers to the average and centroid determination of the data. The centroid is the mean of all the data elements belonging to that cluster. Technically, K-means achieves this by reducing the sum of squared error, that measures the total squared Euclidian distance of data elements to their centroid.

The K-means algorithm consists primarily of three steps. Firstly, k, the total number of clusters, should be specified. Secondly, the n data elements for the centroids without replacement after rearranging the data set to initialize the centroids are chosen. Finally, the process of iteration should be repeated until there is no longer any change in the movement of the centroids. The Euclidean distance between the data elements and the centroid of the clusters is calculated in each iteration. As a result, the data elements with the lowest sum of squared error are clustered to the nearest centroid, thus assigning the data element to that cluster.

The main reason for choosing the k-means algorithm in this thesis instead of other algorithms is its linear complexity. The complexity of the k-means algorithm is $O(i * n * k)$. Here, i denotes the number of iterations, n denotes the total number of data elements, and k denotes the number of clusters. Furthermore, the efficiency of this algorithm is more suitable for higher dimensional datasets. In this thesis work, the K-means algorithm is applied to the last timesteps of the low dimensional hidden state data points. As mentioned earlier, the hidden states from higher dimensions are reduced to lower dimensions using dimensionality reduction techniques. Thus, each timestep of the input is dimensionally reduced in which all the last timesteps of all the input are considered for this algorithm. In particular, the similarities among the hidden states of the last timestep can be studied. The number of clusters to be partitioned depends on the number of timesteps of the input. For instance, the input of three timesteps has three clusters. When a data point is partitioned to particular cluster, the previous timesteps of the particular input are also partitioned to same cluster. This indicates that the last timestep of the input is the deciding factor for the rest of the previous timesteps of that input. Overall, the clusters of the hidden states can show how well the data points are spread or distributed over 2-D space. Using these data points as a reference along with the input, output, and predicted values of each input sample, the distribution or cluster of data points having similar error values can be studied.

4.4 Hidden States and Zeroth Hidden States

The concept of hidden states aids in understanding the internal workings of RNN. In particular, the hidden states at each timestep reflect the information learned by the model thus far from the provided input . As a result, for each input, the knowledge of the model about the dataset gradually increases, resulting in more accurate predictions. When the models are trained for a larger number of epochs and provided with a larger amount of input data, it is possible for the model to learn the underlying hidden pattern. Thus, the accurate predictions are because of the information carried out by the hidden states at each timestep. This information, learned previously, is used as a basis to learn the pattern for the current input. When a model is provided with new input data, it uses the information learned from the previous input and further trains the model. However, the concept of zeroth hidden states is to initialize the hidden states with any desired values at the start of every new input sequence. Therefore, for a new input, the influence of the information learned from the previous input does not play a significant role. In this thesis, the concept of zeroth hidden states is proposed to be explored. Here the zeroth hidden states are initialized with zeros. As a result, every new input sequence has zeros as its zeroth hidden states. Thus, the projection of hidden states as well as zeroth hidden states are visualized. Additionally, due to the presence of zeroth hidden states, the projections will be seen with a data point from where the projection of hidden states emerges.

5 Visual Analytics Tool

The goal of this thesis is to develop a visual analytics tool to visualize and explore the internal hidden states information of a RNN model is implemented with the approaches discussed in the previous chapter. This chapter provides a brief overview on the development environment, software design, and features available in the visual analytics tool. Section 5.1 explains the hardware, software environment setup, and software design used to develop this tool. Section 5.2 discusses about the architecture of the GRU model and section 5.3 shows details about the training and prediction process. Section 5.4 discusses about the data extraction steps performed to provide data to the visual analytics tool. Section 5.5 and section 5.6 explains about the evaluation metric and features available in the tool respectively.

5.1 Environment Setup and Software Design

Hardware

The GRU models are trained on GPU using the SLURM machine learning cluster[YJG03]. Prediction was done using GPU. The configuration of the GPU nodes are of type EPYC 7662 with one TB RAM, 17 TB all-flash ZFS and CentOS 7. The visual analytics tool is developed in a PC with 8 GB RAM and i5 processor installed with Windows 10 64 bit Operating System.

Software

The GRU models are trained with high performance machine learning libraries like Tensorflow 2.8[AAB+16] and Python 3.6. For reproducibility and comprehensibility the visual analytics tool uses mainly functions from the Scikit-learn machine learning library [Bis19] for dimensionality reduction techniques like t-SNE, PCA , and UMAP. IDE used for development was Visual Studio Code. The software and IDE were chosen specifically because of its open source policies.

Software Design

The visual analytics tool has been developed using Python for the backend, particularly the Python flask web application for serving the user requests. HTML, Cascading Style Sheets (CSS), and D3.js are used for the frontend visualizations. The software design used for developing this tool is shown in Fig. 5.1.

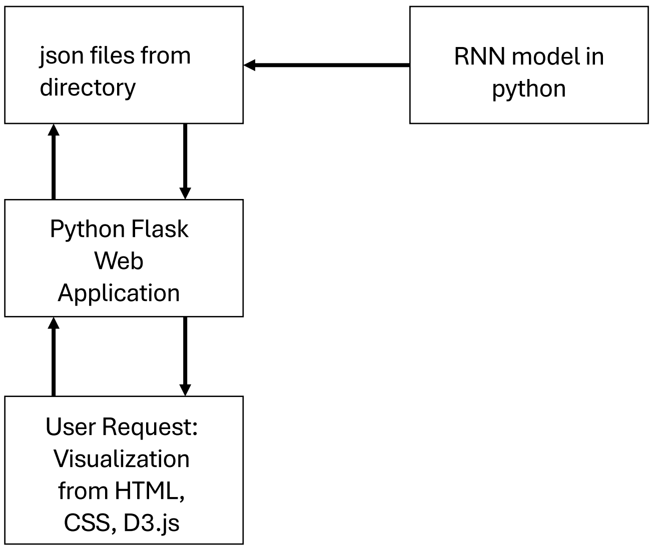

json files from directory
RNN model in python
Python Flask Web Application
User Request: Visualization from HTML, CSS, D3.js

Figure 5.1: Software Design of Visual Analytics Tool.

Python was used to train the RNN models, and the data for the visual analytics tool is stored in a directory as json files. The flask web application is the entry point of this tool and serves the requests from the users. When the user selects the desired visualization feature available in the tool, the request is forwarded to the flask web application. Following that, based on the request of the users, the respective visualizations are displayed.

Each interaction by the user is transformed into an HTTP POST request from the HTML page and is handled by the flask web application. The flask web application is developed for different URL routings based on the available HTML pages. Each HTML page has its own unique visualization features. The details about the available HTML pages will be discussed in Section 5.6.

5.2 Architecture of GRU Model

The main goal of this thesis is to develop an interactive visual analytics tool that aids us in understanding the internal hidden information of the RNN model. This tool was developed for a regression task, which was demonstrated by applying it to turbulence data from an aerodynamics domain. Generally, to predict the output of a regression task, the model should be developed based on an architecture. This architecture briefs about the number of layers, input and output shape for each layer; number of hidden units in each layer; size of layers; and types of layers. Thus, this GRU model has been built with a certain specific architecture by Kurz and Beck[KB20]. They predicted the closure terms from coarse-scale data and achieved up to 99.99% accuracy of cross-correlation between predicted and exact closure terms using this architecture. The GRU architecture used, learns the non-linear mapping between the coarse-scale velocity vector and the filtered DNS operator.

Kurz and Beck had preprocessed the input data by converting 5-D input data to 3-D and feeding it to the GRU model. The three-dimensions of the input data are the amount of input data used for training, the number of timesteps or sequential inputs, and the number of features. The architecture consists of eight layers. It consists of an input layer with a shape of 3-D followed by a normalization layer to normalize the distributions of the training set. Then it is followed by two time distributed layers with 32 and 64 hidden units in each layer, respectively. This time distributed layer functions like a wrapper that allows you to apply a layer to every temporal slice of an input. The fifth layer is the GRU layer, with 64 hidden units. The input shape of this GRU layer is 3-D while its output shape is 2-D. After this GRU layer, two dense layers are arranged with 48 and 24 hidden units, respectively. The hidden units can also be referred to as hidden neurons. Finally, the output layer is 2-D. It can be noticed from Fig.5.2 that from the GRU layer till the output layer, the output shape of each layer is two-dimensional. The activation functions ReLU, sigmoid, and hyperbolic tangent were used, while the kernel initializer was He uniform. This architecture of GRU is of many-to-one RNN type. This type of RNN uses many input values from a consecutive sequence to predict one output value. Sequential timesteps are used to predict one value of the closure term. Figure 5.2 shows the number of layers, input and output shapes for each layer, the number of hidden units in each layer, the size of layers, and the types of layers used for 10 timesteps of input. The smaller the number of timesteps, the smaller the amount of input data provided to the model.

5.3 Training

The training was carried out with a train set of 5971968 samples. The models were trained for sequential timestep inputs of 3, 5, 10, and 20. Prediction was done on 500 samples randomly drawn from a test set of 110592 samples. Both training and prediction were done on SLURM clusters. Figure 5.3 illustrates the input and output data. It shows that the input data is three timesteps, where three arrays consisting of three velocities are used to predict closure terms. The output data is an array of three closure terms. In the input data, the first array of three values are used for predicting the first value of the array in output value. Similarly the second and three arrays of input data are used for predicting second and third terms of the array in output value respectively. Thus, as mentioned earlier this is an example of many-to-one RNN type, where three values of input are used to predict one value of output.

5.4 Data for the Visual Analytics Tool

The visual analytics tool focuses on providing an interactive environment to select a range of values for different hyperparameters. Hyperparameters are parameters which can control the learning process of the model. The values for the hyperparameters can be set by the user before training, unlike the values of weights and biases, which cannot be controlled by the user. Hyperparameters play a significant role in influencing the output prediction of the model. Batch size, epochs, activation function, learning rate, amount of input data, number of hidden layers in a NN, drop-out rate, and optimization algorithms are some of the hyperparameters of a model. In this use case, only five hyperparameters were considered. They are batch size, epochs, activation function, learning rate, and the amount of input data. The values for the hyperparameters were selected for a certain range. Table 6.1 shows the exact values that were considered. Therefore, 288 combinations of

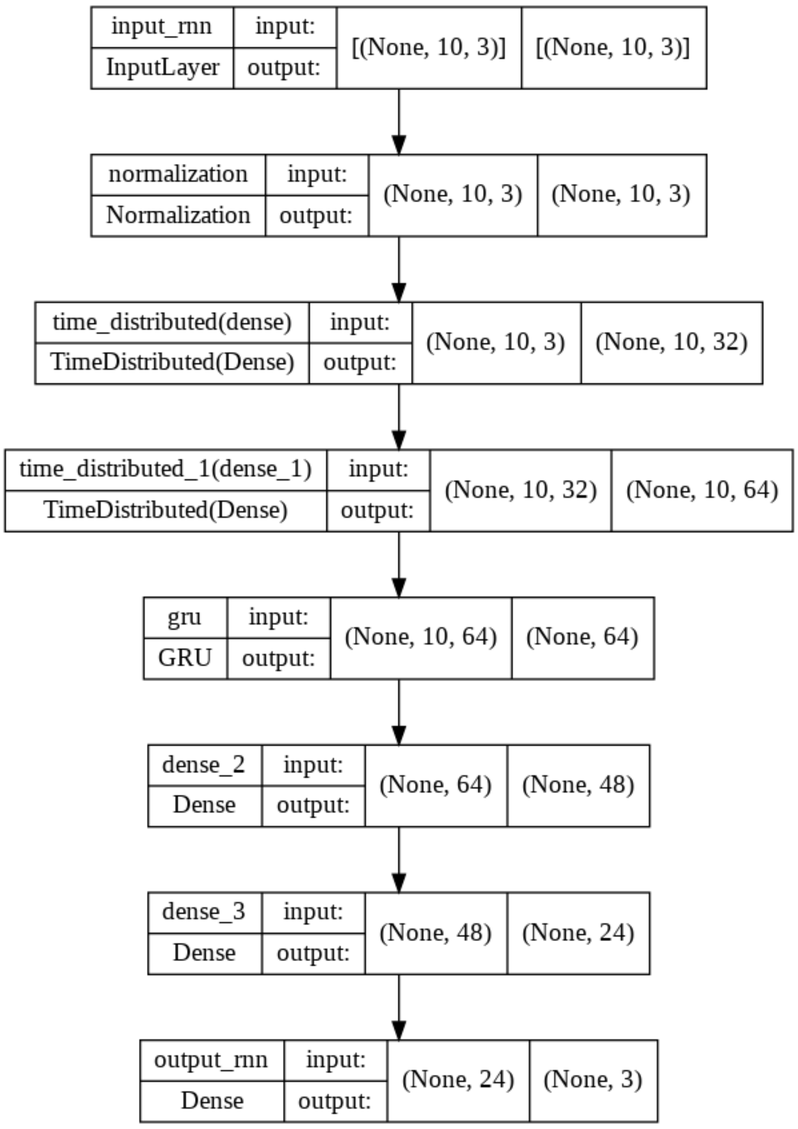
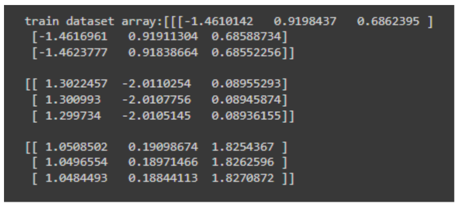

input_rnn | InputLayer | input: | output: | [(None, 10, 3)] | [(None, 10, 3)]
normalization | Normalization | input: | output: | (None, 10, 3) | (None, 10, 3)
time_distributed(dense) | TimeDistributed(Dense) | input: | output: | (None, 10, 3) | (None, 10, 32)
time_distributed_1(dense_1) | TimeDistributed(Dense) | input: | output: | (None, 10, 32) | (None, 10, 64)
gru | GRU | input: | output: | (None, 10, 64) | (None, 64)
dense_2 | Dense | input: | output: | (None, 64) | (None, 48)
dense_3 | Dense | input: | output: | (None, 48) | (None, 24)
output_rnn | Dense | input: | output: | (None, 24) | (None, 3)

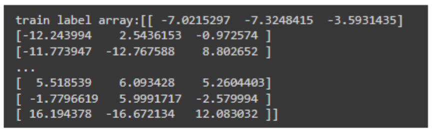
Figure 5.2: Architecture of GRU model with 8 layers showing input and output shape of each layer.



```
train dataset array:[[[-1.4610142   0.9198437   0.6862395 ]
 [-1.4616961   0.91911304  0.68588734]
 [-1.4623777   0.91838664  0.68552256]]

[[ 1.3022457  -2.0110254   0.08955293]
 [ 1.300993   -2.0107756   0.08945874]
 [ 1.299734   -2.0105145   0.08936155]]

[[ 1.0508502   0.19098674  1.8254367 ]
 [ 1.0496554   0.18971466  1.8262596 ]
 [ 1.0484493   0.18844113  1.8270872 ]]
```



```
train label array:[[ -7.0215297  -7.3248415  -3.5931435]
[-12.243994    2.5436153  -0.972574 ]
[-11.773947  -12.767588    8.802652 ]
...
[  5.518539    6.093428    5.2604403]
[ -1.7796619   5.9991717  -2.579994 ]
[ 16.194378  -16.672134   12.083032 ]]
```

Figure 5.3: Sample input data with three timesteps and its respective output data.

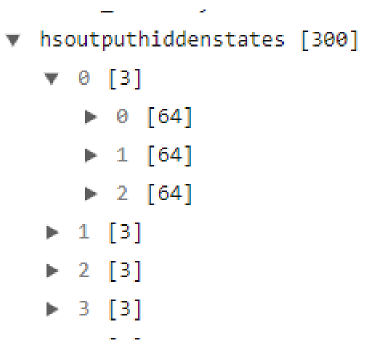
Hyperparameters	**Values**
Timesteps(Amount of input data)	3, 5, 10, 20
Epochs	10, 20, 40, 80
Batch Size	128, 256
Learning Rate	0.01, 0.001, 0.0001
Activation Function	Relu, Sigmoid, tanh

Table 5.1: Hyperparameters of the trained GRU model and their considered range of values. The combination of these hyperparameters resulted in 288 unique combinations.

Dimensionality reduction techniques	**Parameters**	**Values**
t-SNE	n_components	2
t-SNE	perplexity	100
t-SNE	early_exaggeration	12.0
t-SNE	random_state	42
PCA	n_components	2
PCA	random_state	99
UMAP	n_components	2
UMAP	init	random

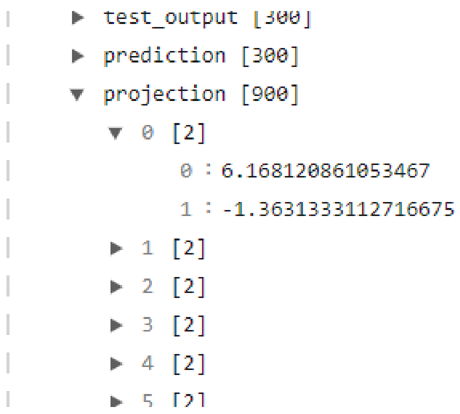
Table 5.2: Different dimensionality reduction techniques used for training the models and their tuneable parameters. The tuneable parameters were used with fixed values for reproducibility.

hyperparameter values were obtained. The models were trained for all 288 combinations of the hyperparameters. After applying dimensionality reduction techniques like t-SNE, PCA, or UMAP on hidden states, the low dimensional 2-D data of the hidden states was obtained. Table 5.2 shows the values of the various tuneable parameters of dimensionality reduction techniques that influence the resultant projection. Due to the fact that the GRU layer has 64 hidden units, the 3-D input fed to the model has 64 features for each timestep of the sequential input. Therefore, when the dimesnionality reduction technique is applied to the hidden states, the 64 features are reduced to 2 features. Thus, the 64 features for each input learned from the GRU layer are reduced to 2 features as shown in Figs. 5.4 and 5.5. The total number of 2-D hidden state data points obtained after applying dimensionality reduction is the multiple of the input sample count and the timesteps. It is evident from Figs. 5.4 and 5.5 that 300 input samples of three timesteps sequential input results in 900 two-dimensional hidden state data points after applying dimensionality reduction. Then K-means clustering is applied on the last timestep of the hidden states and finally, information like dimensionally reduced 2-D data points, timesteps, time increment between files, input values, predicted values, output values, hidden state values, zeroth hidden state values, and values obtained after applying K-means clustering are extracted from the model in JavaScript Object Notation (JSON) format and saved in a directory.



```
▼ hsoutputhiddenstates [300]
  ▼ 0 [3]
    ▶ 0 [64]
    ▶ 1 [64]
    ▶ 2 [64]
  ▶ 1 [3]
  ▶ 2 [3]
  ▶ 3 [3]
```

Figure 5.4: Dimensions of hidden states before applying t-SNE, 64 features at each timestep in 3 timesteps input.



```
▶ test_output [300]
▶ prediction [300]
▼ projection [900]
  ▼ 0 [2]
      0 : 6.168120861053467
      1 : -1.3631333112716675
  ▶ 1 [2]
  ▶ 2 [2]
  ▶ 3 [2]
  ▶ 4 [2]
  ▶ 5 [2]
```

Figure 5.5: Dimensions of hidden states after applying t-SNE. The 64 features at each timestep in 3 timesteps input are reduced to 2 features at each timestep.

5.5 Evaluation Metric

One of the main aims of this thesis is to develop a visual analytics tool for a regression task with interactivity. Therefore, the model developed for this regression task should be evaluated in order to know its performance. Thus, RMSE is used for evaluating the network. It is one of the most common evaluation metrics used for regression tasks. This error helps us to find how much the predicted closure term values are erroneous when compared to the original output values. Generally, RMSE gives higher weightage to larger errors. This is due to the fact that errors are squared before taking the average. The squaring of the errors prevents the errors from cancelling each other out; thus, the error value always ends up positive. Finally, the square root helps to remove the effects of squaring. The overall RMSE value could range between 0 and infinity. Generally, the lower the RMSE value, the better the model fits a dataset. Therefore, using this error, the performance of the model for a specific combination of hyperparameter values out of all 288 combinations of trained models can be determined.

$$y = \sqrt{\frac{\sum_{i=1}^{N} |y(i) - \hat{y}(i)|^2}{N}}. \tag{5.1}$$

The RMSE error can be obtained from Eq.(5.1), where $y(i)$ is the ith output value and $\hat{y}(i)$ is its corresponding prediction. N is the total number of observations.

5.6 Visualization Features of Visual Analytics Tool

The visual analytics tool was developed with HTML, CSS, and the D3.js Javascript library. All the necessary data for the visual analytics tool was extracted in JSON files using Python. A Python flask application acts as the backend for this tool, while HTML, CSS, and D3.js[BD13] act as the frontend. Specifically, D3.js, an open source library of Javascript, was chosen as it has the flexibility to transform the available data into meaningful SVG and also has a variety of functionalities to create interactive visualizations for larger datasets[DR22]. Additionally, it adheres to the web standards to render SVG in almost all the browsers, like Google Chrome, Firefox, and Microsoft Edge. The Python flask application serves the requests of the users from the back end whenever the user interacts with the tool. This tool consists of six HTML pages. Each HTML page has its own features. The list of HTML pages is shown in Fig.5.8. The projections of the hidden states illustrated in this tool are obtained by dimensionally reducing the 64-D to 2-D by applying dimensionality reduction techniques. The 2-D data points represents a dimensional space point. All the projections in this tool are rendered as SVG, which has the possibility to perform interactivity and animation by using D3.js. As mentioned earlier, the Flask application handles the routing by mapping it to a specific URL. Each URL route renders a specific HTML page. The URL routings handled by the flask application and their respective rendered HTML pages are as follows:

- /home - `home.html`
- /available - `available.html`
- /next - `visualizations.html`
- /zerohidden - `zerohidden.html`
- /comparison - `comparison.html`
- /overall - `parallelplot.html`
- /errortable - `RMSE.html`

When the flask application server is started, it runs the flask python script and presents the `home.html` page. The `home.html` page allows the user to select the hyperparameter. The user can view the filename by clicking the `Submit to see file name` button. Then, for instance, when the user clicks the `Hidden States` button, the /next URL gets triggered and the `visualizations.html` page is rendered. After that, on the `visualizations.html` page, if the user selects a file, the flask web application handles each interaction by transforming it into an HTTP POST request. Then the respective file stored in json format in the directory is fetched and the necessary information is returned as a JSON object to the `visualizations.html` page. After receiving the necessary information for the visualization, D3.js in the `visualization.html` page aids in rendering the SVG in the front end. This tool was developed with the following visualization features.

- Low dimensional hidden state data points are projected as 2-D.
- Heat map of hidden state data points.

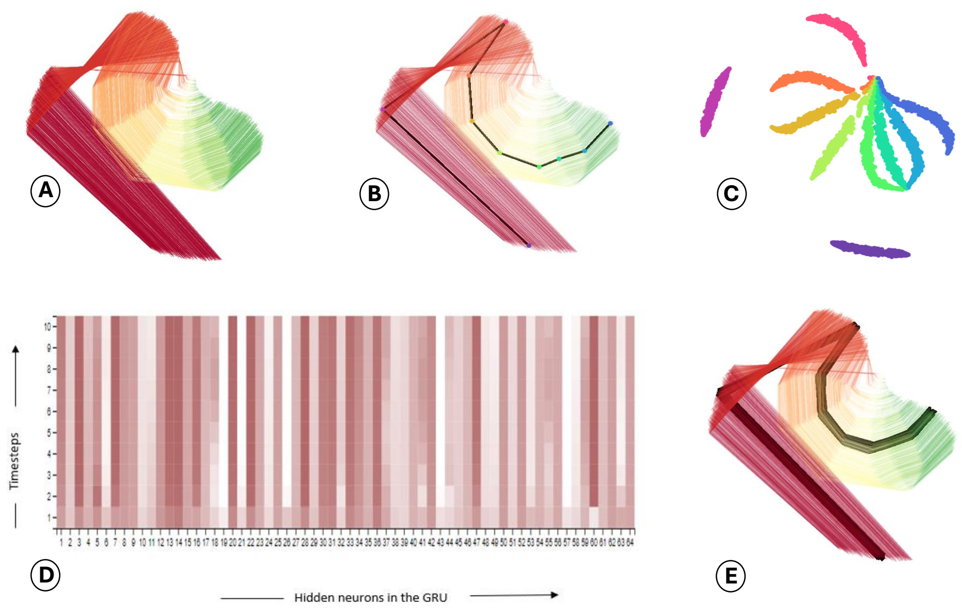


Figure 5.6: Visualizations in the Visual Analytics Tool. (**A**): Projection of hidden states as non-linear trajectories in 2-D space; (**B**): Projection of hidden states of specific input selected by the user; (**C**): Hidden states visualized as scatterpoints; (**D**): Hidden states of specific input visualized as heat map; (**E**): Projection of cluster of hidden states.

- Scatterpoints of low dimensional hidden state data points.
- Animation of the projection of low dimensional hidden state data points.
- Projection of clustering of low dimensional hidden state data points.
- Comparison of two different models.
- Projection of low dimensional zeroth hidden state data points.
- Scatterpoints of low dimensional zeroth hidden state data points.
- Parallel Coordinates plot for overall comparison with error values.

Figure 5.6 (A) illustrates the projection of hidden states for a specific combination of hyperparameter values. The projection of a specific input value, heat map, and its scatterpoints visualization are shown in Figs. 5.6 (B), 5.6 (C), and 5.6 (D) respectively. Figure 5.6 (E) represents the projection of the clustering of hidden state data points.

Figure 5.7 (F) illustrates the projection of zeroth hidden states. The projection of zeroth hidden state as scatterpoints, comparison of two different models, and parallel coordinates plot of the trained models with their error values are shown in Figs. 5.7 (G), 5.7 (H), and 5.7 (I) respectively.

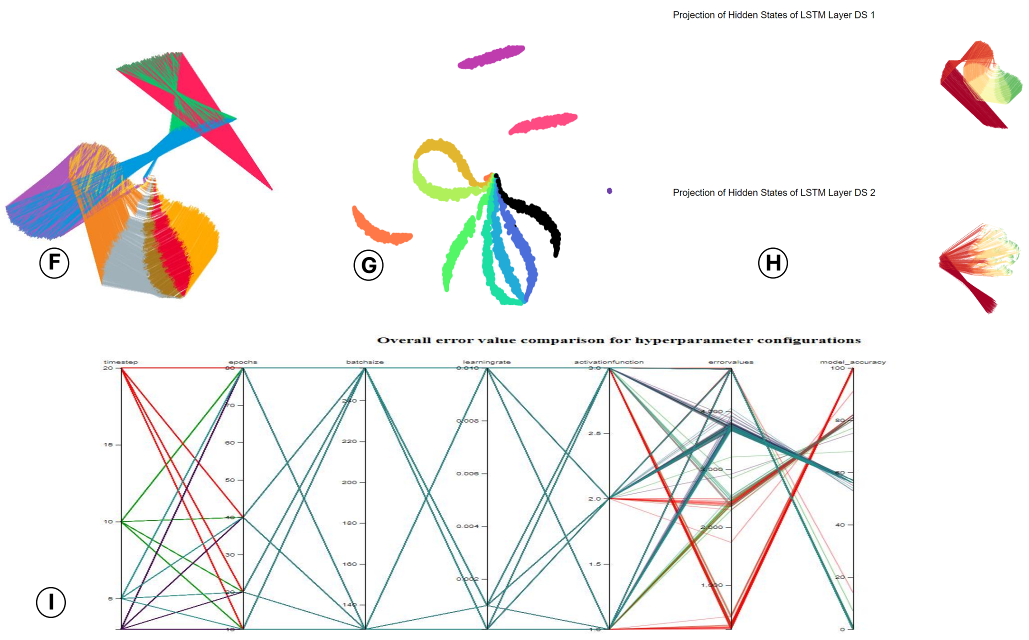


Figure 5.7: Visualizations in the Visual Analytics Tool. (**F**): Projection of zeroth hidden states in 2-D space; (**G**): Zeroth hidden states visualized as scatterpoints; (**H**): Comparison of two different models by the projection of hidden states; (**I**): Parallel Coordinates plot view of all the trained models with hyperparameter values and their RMSE error values.

5.6.1 Hyperparameter Selection

One of the features of this tool is that it allows the user to have an interactive environment for choosing values of different hyperparameters. The `home.html` page allows the user to choose the values of five hyperparameters for which the model was trained. The range of values for all the parameters can be selected from the dropdown list. This page also allows the user to save the selected hyperparameter configuration values locally by clicking `Save` button. After clicking the `Submit to see file name` button, the file name for the selected combination of hyperparameter values is displayed. The `home.html` page is shown in the Fig. 5.9.

5.6.2 Available Models

The `available.html` page is the instruction guide for using the visual analytics tool. This page contains information about all the combinations for which the models are trained, their file names, dimensionality reduction techniques used for interactivity, and some other general information like the color coding used in the projections for easy understanding of the projections.

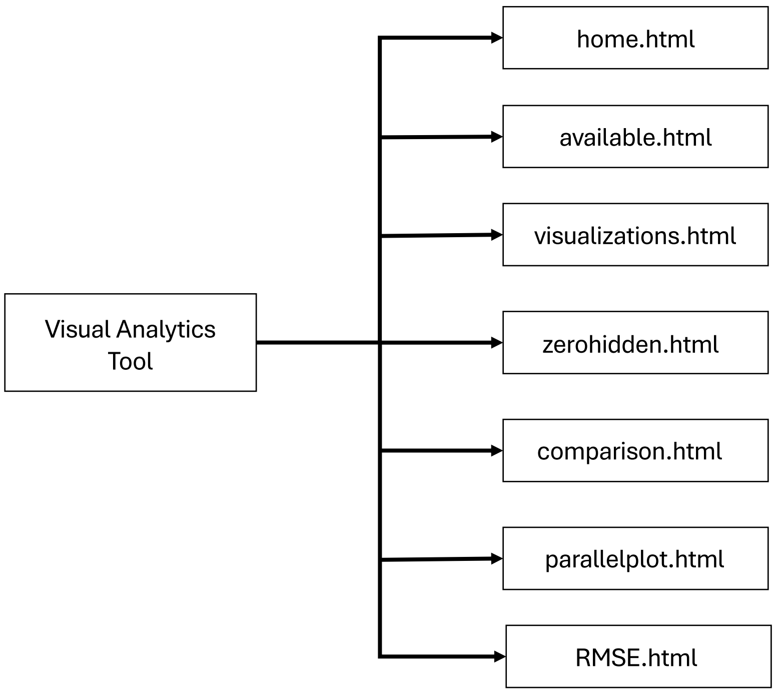


Figure 5.8: HTML pages available in Visual analytics Tool.

5.6.3 Hidden States Projections

The `visualizations.html` page allows the user to select a particular file which contains information about a specific combination of hyperparameter values and projects their dimensionally reduced 2-D hidden state data points as the non-linear trajectories. Figure 5.10 illustrates the projection of hidden states with the corresponding color code from left to right. The different colored lines indicate the connection between the hidden state data points of various timesteps. It is evident from Fig.5.10 that the hidden state of the 10 timesteps sequential input data is connected with different color lines. This page also shows the input data and their respective original output values, predicted values, and the error between them. On selecting a particular input, its respective projections can be visualized in a black line. The heat map allows the user to understand the evolution of hidden state information over all the hidden units for different timesteps. It indicates the contribution of each hidden unit over all timesteps for a given input data. This page also displays information about basic sequence length and the number of sequences used for predictions. Additionally, the scatterpoints feature aids the user in understanding how well the projections of the hidden states of sequential timestep inputs are scattered over the 2-D space. The animation feature aids an animated view of the projected 2-D data points. Furthermore, the clustering option allows the user to view the clusters of the t-SNE projected data points. This data point information was obtained as the result of applying K-means clustering to the last timestep of the t-SNE projected data points. Figure 5.11 shows the clusters of hidden state projections with color coding. Additionally, the projections of dimensionally reduced hidden state data points obtained from using PCA or UMAP are also provided for the user to visualize.

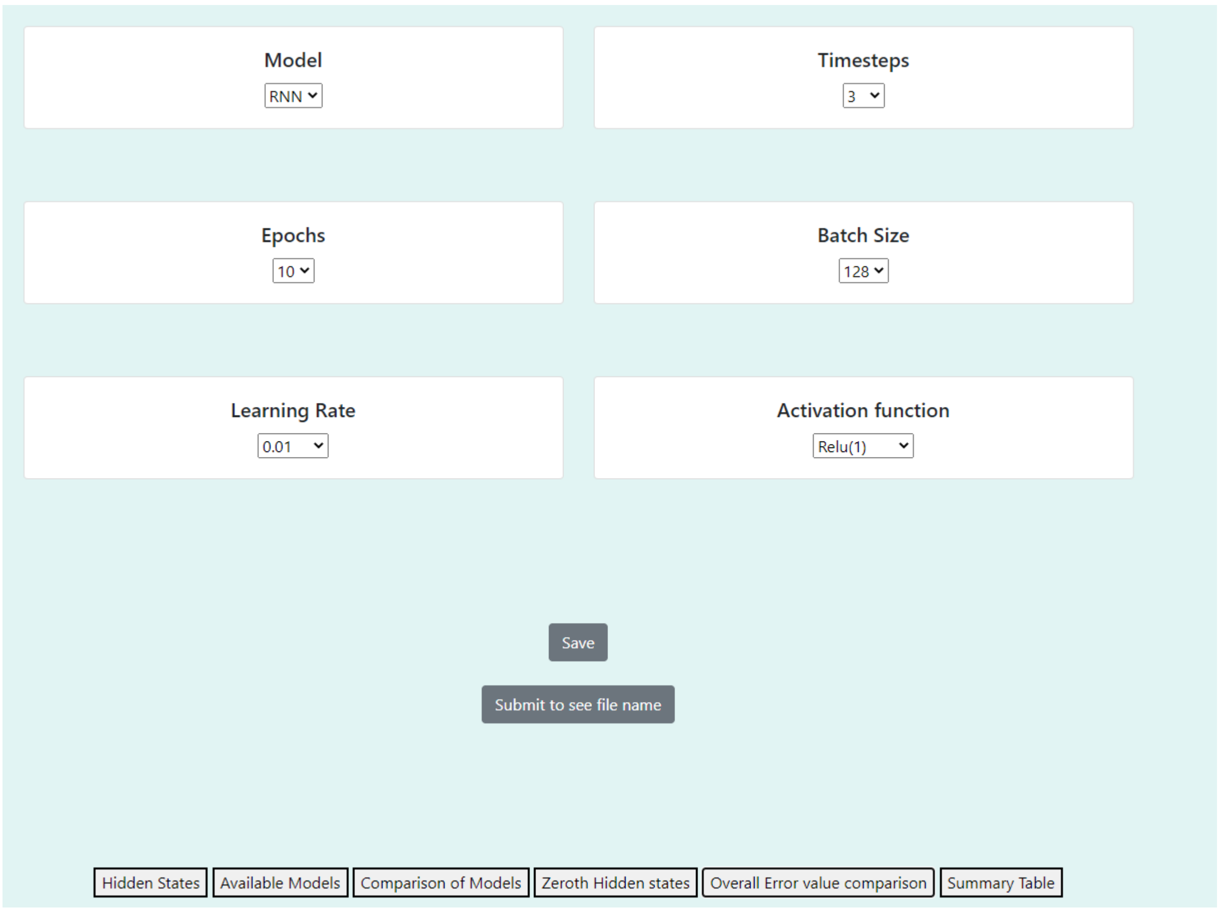

Model: RNN
Timesteps: 3
Epochs: 10
Batch Size: 128
Learning Rate: 0.01
Activation function: Relu(1)

Figure 5.9: Interactive home page of visual analytics tool having provision to select hyperparameters and their values.

5.6.4 Zeroth Hidden State

The `zerohidden.html` page allows the user to visualize the projections of the zeroth hidden state. The user can select a model from the list of available models, for which the projections need to be seen. Zeroth hidden states are a set of zeros which are applied before every new sequential input so that it makes the model learn from that initial zero state. This page shows how the projection looks when zero hidden states are considered. As a result, the projection appears to emerge from a single point in 2-D space. Additionally, the scatterpoints of the zeroth hidden states can also be visualized.

5.6.5 Comparison of Models

The `comparison.html` page allows the user to select two models with different combinations of hyperparameter values and compare their projections. This visualization aids the user to get a good insight into how the projection changes for each model with the change in hyperparameter values. Out of all 288 combinations of trained models, each combination has a unique pattern

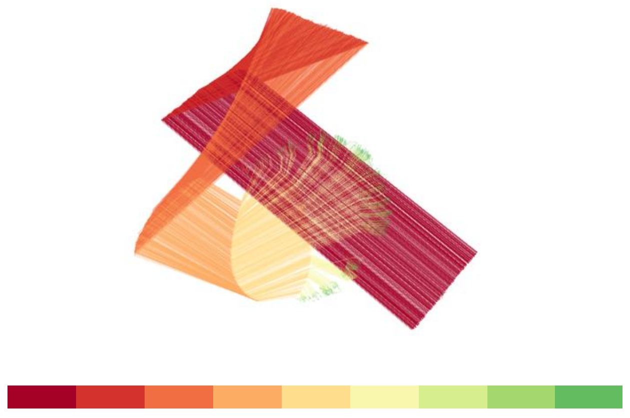
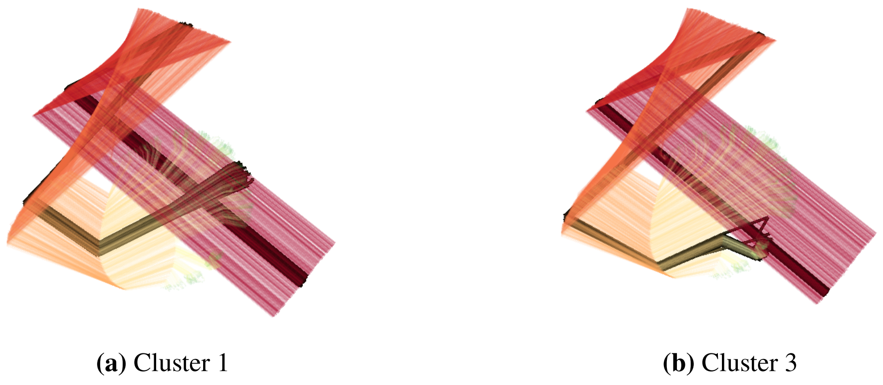


Figure 5.10: Projection of hidden states for 10 timesteps input indicating the spread of timesteps over 2-D space. The hidden states of sequential timesteps were connected by lines of different color. The color grading is to be considered from left to right.



(a) Cluster 1

(b) Cluster 3

Figure 5.11: Projection of hidden states with 10 timesteps input clustered. **Left**: Fig. 5.11a shows the clustered hidden states at timestep 1. **Right**: Fig. 5.11b shows the clustered hidden states at timestep 3.

of 2-D projected data. This comparison highlights where the hidden state data points are exactly spread over the 2-D space. Overall, this visualization view allows the user to compare a wide range of combinations.

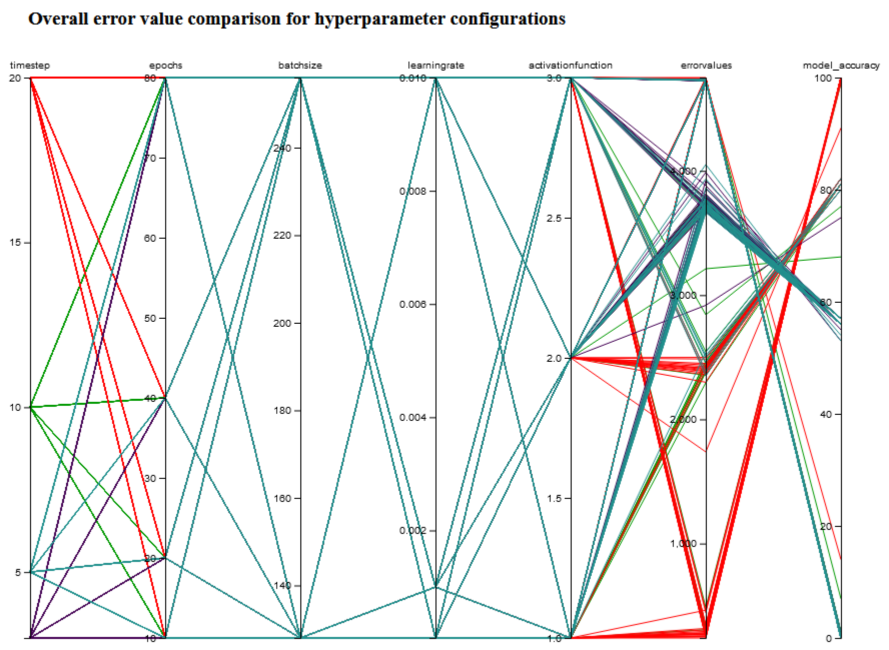
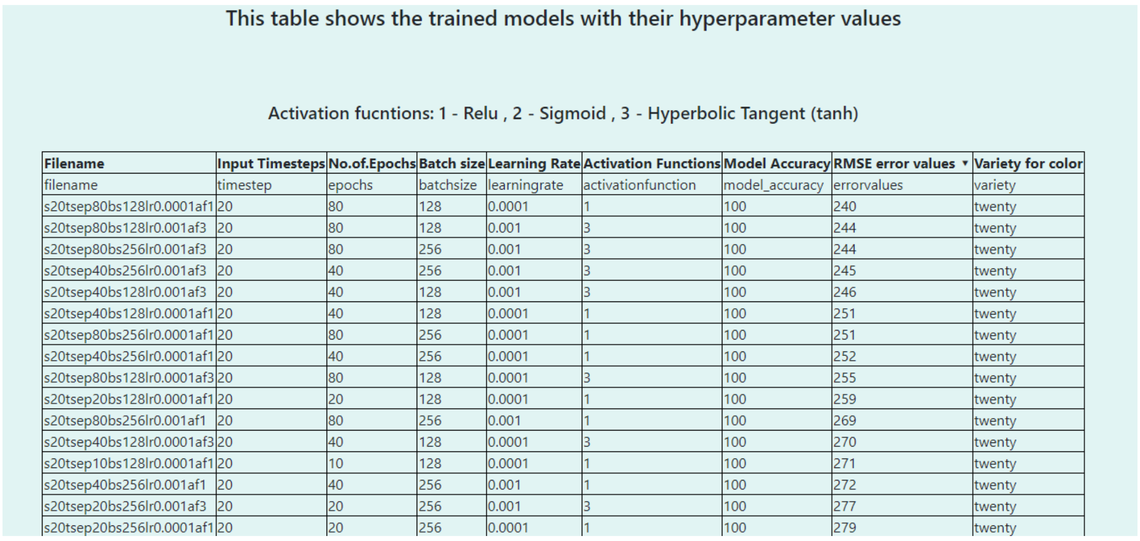


Figure 5.12: Parallel Coordinates plot view of all the trained models with hyperparameter values and their resultant error values. **Left to right**: Timesteps, epochs, batch size, learning rate, activation function, error values, and test set accuracy. Red color represents the 20 timesteps input; Light green represents the 10 timesteps input; Dark cyan represents the 5 timesteps input; Purple represents the 3 timesteps input.

5.6.6 Overall Comparison

The `parallelplot.html` page allows the user to have an overview of the performance of the trained models. The overall performance is visualized using a parallel coordinates plot. It is one way of visualizing and exploring datasets with higher dimensions and also when the features are multivariate. However, here in this use case, five hyperparameter values are taken, while each hyperparameter value has its own axis placed parallel to the other. The scale of each axis could be based on the values of the hyperparameter or it could be made uniform. Here, the axis of each hyperparameter is made to have its own measurements in order to get a better understanding of the data. The order in which the hyperparameters are placed next to each other also impacts the end interpretation of the user. Therefore, values are shown as a network of linked lines across all axes. This implies that each line is made up of a set of points that have been joined together and are distributed along each axis. However, this plot has the disadvantage of overcluttering, for which the hovering effect was used to highlight the specific set of points. When hovered over the lines, only the specific combination of hyperparameters is highlighted and gives an easy visualization to compare their error values. Figure 5.12 shows the overall performance of all trained models with summed RMSE error values.



This table shows the trained models with their hyperparameter values

Activation fucntions: 1 - Relu , 2 - Sigmoid , 3 - Hyperbolic Tangent (tanh)

Filename	Input Timesteps	No.of.Epochs	Batch size	Learning Rate	Activation Functions	Model Accuracy	RMSE error values ▾	Variety for color
filename	timestep	epochs	batchsize	learningrate	activationfunction	model_accuracy	errorvalues	variety
s20tsep80bs128lr0.0001af1	20	80	128	0.0001	1	100	240	twenty
s20tsep80bs128lr0.001af3	20	80	128	0.001	3	100	244	twenty
s20tsep80bs256lr0.001af3	20	80	256	0.001	3	100	244	twenty
s20tsep40bs256lr0.001af3	20	40	256	0.001	3	100	245	twenty
s20tsep40bs128lr0.001af3	20	40	128	0.001	3	100	246	twenty
s20tsep40bs128lr0.0001af1	20	40	128	0.0001	1	100	251	twenty
s20tsep80bs256lr0.0001af1	20	80	256	0.0001	1	100	251	twenty
s20tsep40bs256lr0.0001af1	20	40	256	0.0001	1	100	252	twenty
s20tsep80bs128lr0.0001af3	20	80	128	0.0001	3	100	255	twenty
s20tsep20bs128lr0.0001af1	20	20	128	0.0001	1	100	259	twenty
s20tsep80bs256lr0.001af1	20	80	256	0.001	1	100	269	twenty
s20tsep40bs128lr0.0001af3	20	40	128	0.0001	3	100	270	twenty
s20tsep10bs128lr0.0001af1	20	10	128	0.0001	1	100	271	twenty
s20tsep40bs256lr0.001af1	20	40	256	0.001	1	100	272	twenty
s20tsep20bs256lr0.001af3	20	20	256	0.001	3	100	277	twenty
s20tsep20bs256lr0.0001af1	20	20	256	0.0001	1	100	279	twenty

Figure 5.13: Summary table consisting of the trained models with their hyperparameter values and error values.

5.6.7 Summary Table

The `RMSE.html` page displays a table with the list of trained models, their respective hyperparameter values, and their RMSE error values. The table can be sorted either by ascending or descending order by clicking on the column name. The column RMSE error values, when sorted in ascending order, shows that most of the models which have fewer errors have specifically 20 timesteps as input. Thus, this table aids in achieving an insight about the best and worst performing models and is shown in Fig.5.13. Overall, this shows that models perform well or fit better when fed with 20 timesteps of input.

6 Results and Discussion

In this chapter, the workflow of the visual analytics tool and the insights obtained from the projections of hidden states and the comparisons between two models are discussed. The visualizations are illustrated using screenshots obtained from the running version of this tool. The tool has been evaluated with the turbulence dataset and the functionalities of each features are tested.

6.1 Workflow of Visual Analytics Tool

6.1.1 General Data Flow

The Python web server is started to run the flask web application Python script. This tool consists of seven HTML pages and one flask web application. The Python flask web application is the entry point of this tool. The yellow colored components correspond to the frontend. The green color component corresponds to the backend, which is a flask web application. The blue color component represents the storage or directory where the json files required for the visualizations are stored. The short dashed line flows from the start or entry point to the flask web application. This indicates that once the web server is started, it runs the main flask web application. The `mainflask.py` Python script presents the home page of this tool to the user. The data flow is represented by the long dashed lines with double-sided arrows. The frontend components connected with double-sided arrows represent the flow of request and response after the interaction of the user. The interaction by the user triggers an HTTP request, which is a POST method. This request is served by the `mainflask.py` Python script using different URL routings and the necessary response is sent to the HTML pages. The request could be a specific JSON file or it could be a specific input in that JSON file. In all cases, the response is sent as a JSON object to an HTML page. D3.js aids in reading the received JSON object and renders the visualization requested by the user. The `home.html` page allows the user to choose the remaining six HTML pages. Therefore, `home.html` is connected with lines to the remaining six pages. Of the seven HTML pages, four are dynamic pages, where the content of the page changes based on the request of the user. The dynamic HTML pages are `home.html`, `visualizations.html`, `zerohidden.html`, and `comparison.html`. The remaining three HTML pages, `available.html`, `parallelplot.html`, and `RMSE.html`, are static. `parallelplot.html` allows the user to hover over the lines to highlight a specific combination of hyperparameter values. `RMSE.html` page displays a table with the list of trained models, their respective hyperparameter values, and their RMSE error values. The table can be sorted either in ascending or descending order by clicking on the column name. Figure 6.1 shows the workflow of the tool, with frontend and backend components connected with connector lines indicating the workflow and data flow of information.

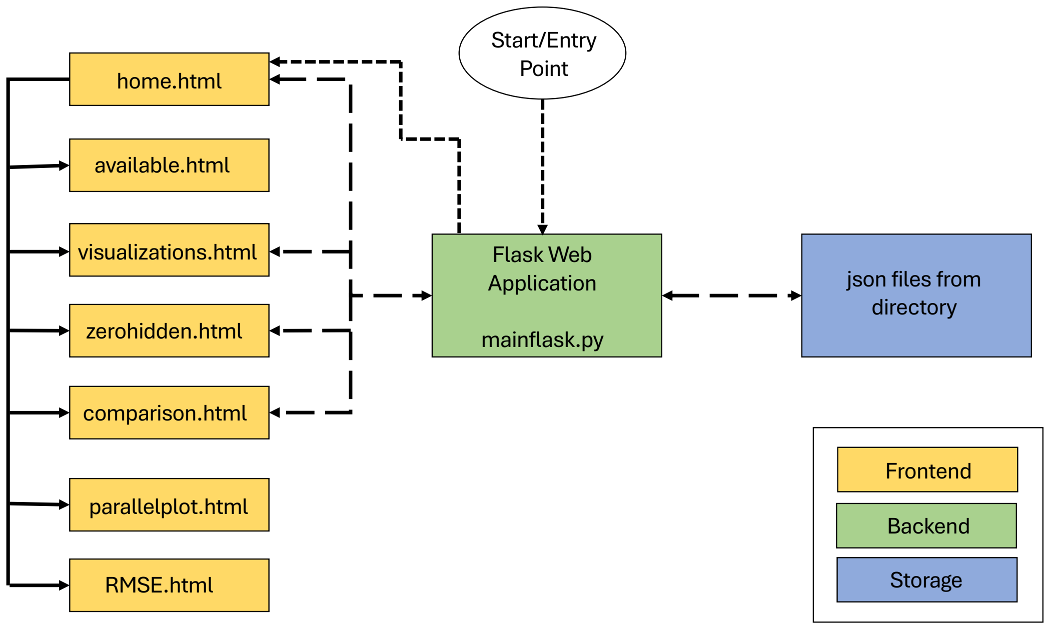


Figure 6.1: Workflow of the visual analytics tool with the frontend, backend, and storage components indicated in yellow, green, and blue colors respectively. The direction of the flow of data between the components is represented by long dashed double-sided arrows. The short dashed arrows represent the work flow from the entry point of the tool to the home page. All the HTML pages available in the home page are represented by normal arrows.

6.1.2 Workflow for the Visual Analytics Tool

The workflow for the analysis of interpretation and exploration of hidden states begins with `home.html`. The parallel coordinates plot displays the overall performance of all the trained models with the hyperparameter values and their error values. In addition to that, it provides an insight about the test set accuracy of the trained models. Hence, the user should click the `Overall Error value comparison` button to view the available models. The user can hover over the specific models. Additionally, the user can view the exact and accurate hyperparameter details about the model from the summary table available in the `RMSE.html` page. In order to view the table the user should click the `Summary Table` button. After that, in order to to explore the interested model, the user should choose the hyperparameter values from the drop-down list on the home page, and after selecting the values for the hyperparameter, the `Submit to see file name` button should be clicked to see the name of the file for the selected combination of hyperparameters. Then, after knowing the file name, the user should click the `Hidden States` button in the `home.html`. This button click presents the `visualizations.html` page and the user should choose the desired file name. After choosing the file name, the projection of hidden states for that trained model is visualized. This button click also loads a data table consisting of all the input values used in training, their respective output and prediction values, and the RMSE error value for each input. Then checking any of the `t-SNE`, `PCA`, or `UMAP` checkboxes displays the projection of hidden states obtained by applying t-SNE, PCA, or UMAP respectively. Furthermore, by clicking any particular input value in the data table shown, the projection of the hidden state for that specific input is presented. Additionally, a heat map for the specific input is presented. The heat map shows the contribution of each hidden unit for all the timestep of the inputs. In order to view the animation of the projection of hidden states the `Animate` checkbox should be checked and the `Do animate` button should be clicked. Similarly, the `Scatterpoints` checkbox should be checked and the `Do Scatter` button should be clicked, in order to view the projection of hidden states as scatterpoints. In addition, the desired `cluster` checkboxes should be checked and `Cluster` button should be clicked to view the clusters of hidden state projections. Furthermore, by selecting the required file, the projection of hidden states along the zeroth hidden state can be visualized. Once the data from the file gets loaded, the `Zeroth Hidden states` button should be checked to visualize the projections. In order to see the projection of zeroth hidden states as scatterpoints, the `Scatterpoints` checkbox should be checked and the `Do Scatter` button should be clicked.

The workflow for the analysis of comparing two different models begins with `home.html`. The user can compare the projection of hidden states of two models. In order to compare the models, first the available trained models should be known. The available models can be checked by visiting the `available.html` page or by visiting the `RMSE.html` page. After deciding the models to be compared, the user should click the `Comparison of Models` button. Then in the `comparison.html` page, the required filename of the two models should be selected. This event of clicking presents the projections of hidden states of two models along with the basic information about the models. It displays the number of samples used for training, test set accuracy values, and sequence length or number of input timesteps.

6.2 Discussion and Analysis

We present four different analyses obtained from our visual analytics tool. Firstly, the interpretation and exploration of the hidden states, as well as the contribution of each hidden unit, can be investigated. Secondly, an analysis can be obtained by comparing two models with different hyperparameter configurations. Thirdly, for the turbulence dataset, an analysis of the exploratory data analysis and overall performance of the trained models is performed. Finally, several models can be compared based on their accuracy and their hidden patterns.

6.2.1 Analysis 1: Interpretation and Exploration of Hidden States

One of the analyses obtained from this tool is the deep understanding of the interpretation and exploration of the hidden states of the RNN model. This can be achieved by following the workflow mentioned in the Section 6.1.2. The hyperparameter configuration of the trained model is shown in the Table 6.1 and it indicates that the model is trained with 10 epochs for 10 consecutive timesteps as input. As mentioned earlier, the prediction is done for 500 randomly chosen data samples. As the input is sequential with 10 consecutive timesteps, each timestep has 64 characteristics for the 500 randomly chosen data samples. The 64 features are obtained as a result of the 64 hidden units in the GRU layer. By setting the return sequences argument to true, the hidden states for each timestep can be extracted. This parameter return sequences can be found in the GRU function of the Scikit-learn library in Python. Therefore, the output shape of the GRU layer results in 3-D and it is (500, 10, 64). Here, 500, 10, and 64 represent the number of input data samples, total number of timesteps, and number of hidden states from the hidden units, respectively. When these hidden

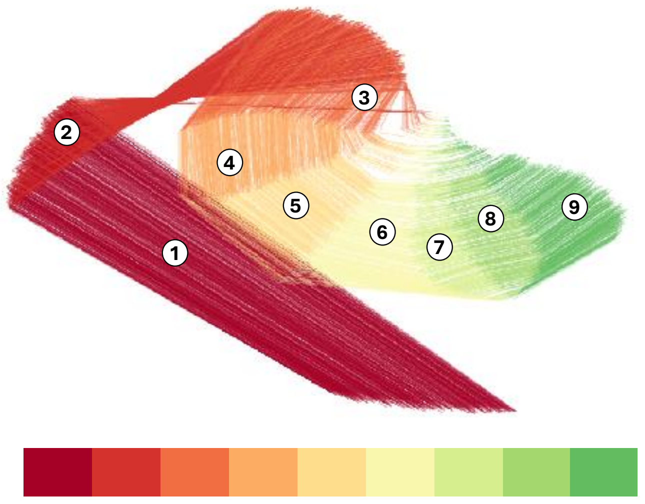


Figure 6.2: Projection of hidden states over 2-D space along with the color coding.

Hyperparameters	Values
Timesteps(Amount of input data)	10
Epochs	10
Batch Size	128
Learning Rate	0.001
Activation Function	Sigmoid

Table 6.1: Hyperparameters of the trained GRU model and its range of values.

states are applied with a t-SNE dimensionality reduction technique, the 64 features are reduced to 2 features. Hence, the final shape results in (5000, 2). Therefore, 500 data samples of 10 timesteps result in 5000 data points. Each hidden state data point represents a timestep that is dimensionally reduced. The projection of the hidden states of 10 timesteps input with 500 data samples consists of 5000 data points in 2-D space and is shown in Fig.6.2. The dimensionally reduced hidden states data points are connected with lines of different colors in non-linear trajectories. This projection employs linear interpolated color coding. The colors should be considered from left to right. For instance, carmine is the first color in the color coding, which is used to connect hidden states of first and second timesteps. Table 6.2 shows the nine colors that flow between the hidden states. Similarly, there are eight different colors of lines that connect the hidden states of 10 timesteps. Hence, based on the number of timesteps of input, the number of lines with different colors varies. The 5000 hidden state data points are depicted in Fig.6.3 over the 2-D space. The color gradient is used to denote each timestep. Table 6.3 shows the hidden states in low dimension, with each color representing one timestep. For instance, the royal purple color the first timestep visualized as scatterpoints over 2-D space. Similarly, the remaining nine distinct colors of lines represent the remaining timesteps.

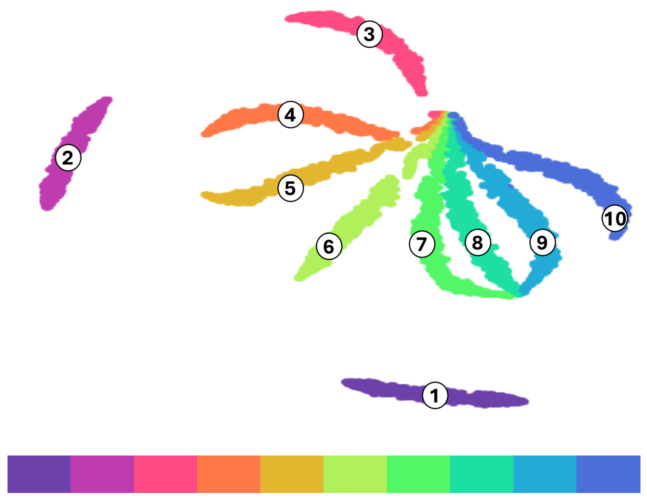


Figure 6.3: Low dimensional representation of hidden states as scatterpoints along with the color coding.

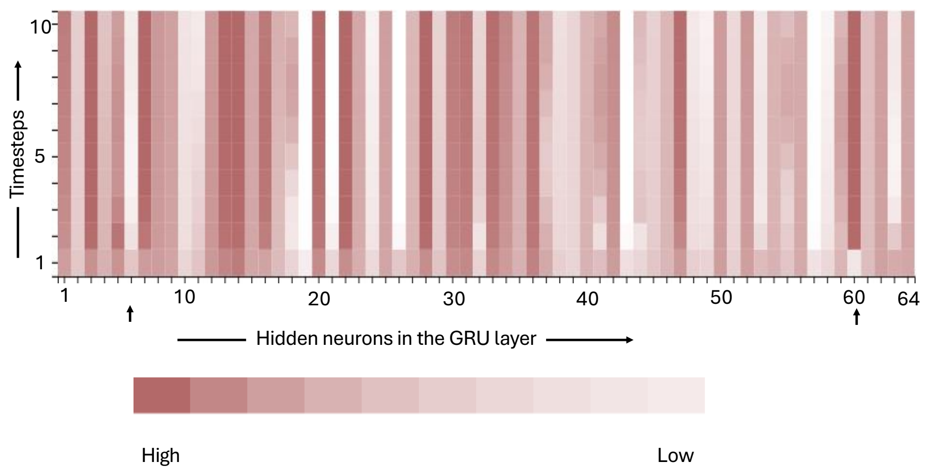


Figure 6.4: Heat map representing the contribution of hidden units for each timestep. X-axis represents the total number of hidden neurons in the GRU layer (i.e.,64), while Y-axis represents the timesteps.

Line	Colors
1	Carmine
2	Persian Red
3	Burnt Sienna
4	Rajah
5	Vis Vis
6	Shalimar
7	Reef
8	Feijoa
9	Fern

Table 6.2: Color of the lines connecting the low dimensional hidden states data points.

Timesteps	Colors
1	Royal Purple
2	Fuchsia
3	French Rose
4	Coral
5	Gold Tips
6	Conifer
7	Screaming Green
8	Shamrock
9	Summer Sky
10	Royal Blue

Table 6.3: Color of the hidden states data points in low dimension grouped by timesteps. This represents the hidden states as scatter points.

The contribution of each hidden unit over all timesteps for a given input dataset is depicted in Fig.6.4. As mentioned earlier, the GRU layer has 64 hidden units. Hence, the intensity of each individual block represents how much the block has contributed to the prediction of output when compared to the previous block. For instance, timesteps 2 to 10 have contributed more to the prediction of output for hidden unit 60 than the timestep 1. Similarly, timestep 1 has more influence than the rest of the timesteps for hidden unit 6. Overall, this heat map depicts the visualization of hidden states before applying the dimensionality reduction techniques. Additionally, the RMSE error for each input sample can be viewed in the table when the information for the specific model is loaded. Figure 6.7 shows the input data, output values, predicted values, and RMSE error for each input sample.

The tool allows the user to visualize the projection of hidden states for each individual input with a black line. Figure 6.5 shows the zoomed in projection of hidden states with timesteps represented in different colors of circles. Therefore, it indicates how well the hidden states are distributed for each input. The transition of inputs can be visualized and is shown in Fig. 6.6. The left part of the figure represents the projection of hidden states for the first input sample, and the right part represents the projection of hidden states for the second input sample. Both the specific inputs are represented by the black lines.

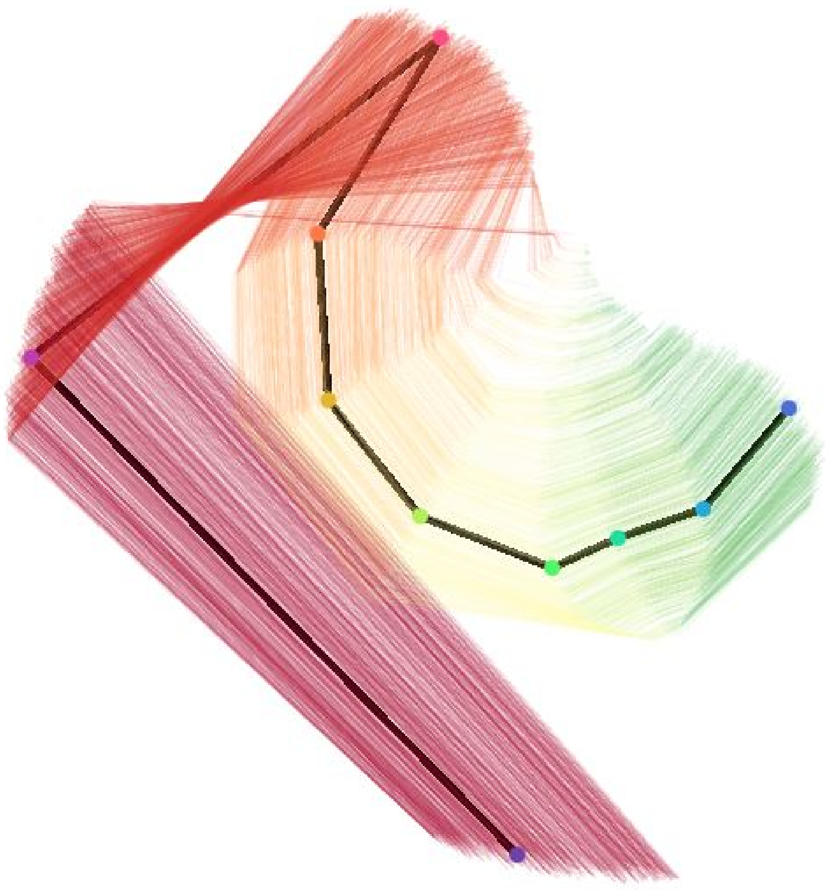


Figure 6.5: Projection of hidden states for 10 timesteps showing a particular input after interactivity. The circle with different colors indicates the hidden states of sequential timesteps input and are connected with the black lines to indicate the spread of specific selected input as a result of the interactivity.

The animation of the projection of hidden states is shown in the Fig. 6.8. It is well explained how the projection appears at various time units during the animation. The animation also aids in understanding how all the hidden states of the input samples are aligned in the 2-D space. Additionally, K-means clustering was applied to the last timesteps of the low dimensional hidden states. This implies how the hidden states are clustered and it shows the similarity among the hidden state data points at the last timestep. The tool was developed in such a way that the number of clusters depends on the number of timesteps for the input. It is illustrated in the Fig.6.9, among 10 available clusters for 10 timesteps of input, cluster 1 and cluster 3 are projected. These clusters represent hidden states that have similarities.

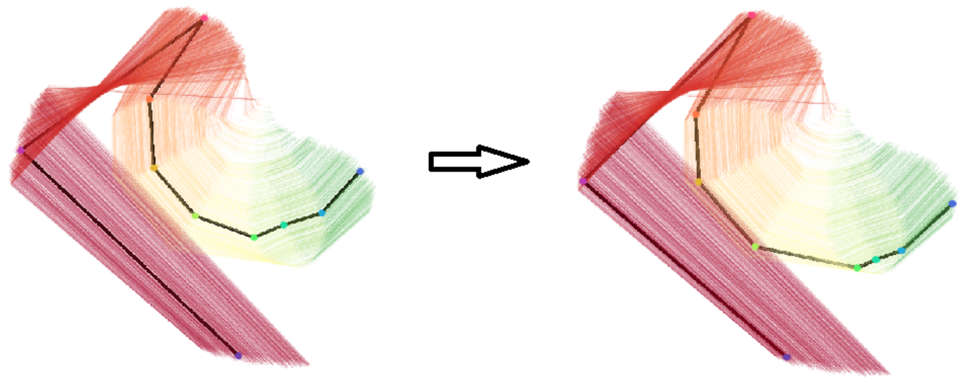
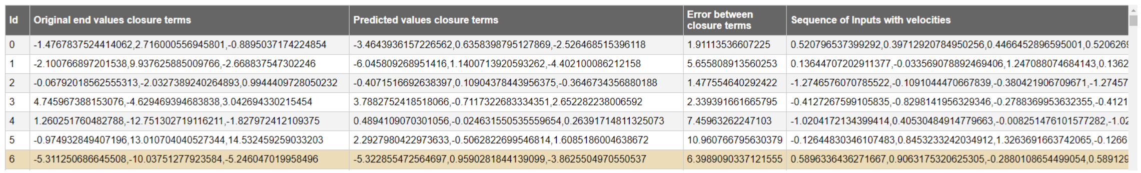


Figure 6.6: Projection of hidden states for 10 timesteps input showing a transistion from one input to another input. **Left**: The projection of hidden states for first input sample; **Right**: The projection of hidden states for second input sample.



Id	Original end values closure terms	Predicted values closure terms	Error between closure terms	Sequence of Inputs with velocities
0	-1.4767837524414062,2.716000556945801,-0.8895037174224854	-3.4643936157226562,0.6358398795127869,-2.526468515396118	1.91113536607225	0.520796537399292,0.39712920784950256,0.4466452896595001,0.520626
1	-2.100766897201538,9.937625885009766,-2.668837547302246	-6.045809268951416,1.1400713920593262,-4.402100086212158	5.655808913560253	0.13644707202911377,-0.033569078892469406,1.247088074684143,0.1362
2	-0.06792018562555313,-2.0327389240264893,0.9944409728050232	-0.4071516692638397,0.10904378443956375,-0.3646734356880188	1.477554640292422	-1.2746576070785522,-0.1091044470667839,-0.380421906709671,-1.27457
3	4.745967388153076,-4.629469394683838,3.042694330215454	3.7882752418518066,-0.7117322683334351,2.652282238006592	2.339391661665795	-0.4127267599105835,-0.8298141956329346,-0.2788369953632355,-0.4121
4	1.260251760482788,-12.751302719116211,-1.827972412109375	0.4894109070301056,-0.024631550535559654,0.26391714811325073	7.45963262247103	-1.0204172134399414,0.40530484914779663,-0.008251476101577282,-1.02
5	-0.974932849407196,13.010704040527344,14.532459259033203	2.2927980422973633,-0.5062822699546814,1.6085186004638672	10.960766795630379	-0.12644830346107483,0.8453233242034912,1.3263691663742065,-0.1266
6	-5.311250686645508,-10.03751277923584,-5.246047019958496	-5.322855472564697,0.9590281844139099,-3.8625504970550537	6.3989090337121555	0.5896336436271667,0.9063175320625305,-0.2880108654499054,0.589129

Figure 6.7: Table displaying the input data, output values, predicted values, and RMSE error for each input sample.

The projection of hidden states along with the zeroth hidden state is shown in Fig. 6.10. This depicts the influence of the zeroth hidden state for the same configuration in Tab. 6.1. It is clear that when comparing Figs. 6.2 and 6.10, both appear almost similar. Because of the zeroth hidden state, it appears that the projection of the hidden states emerges from a single point. The visualization has 11 timesteps, hence there are a total of 10 different colors of lines connecting the hidden state data points at each timestep. Technically, the zeroth hidden state is a data point in 2-D space. This projection is also visualized as scatterpoints, which is shown on the right side of this figure.

In addition to t-SNE projections, the tool allows for the visualization of hidden states in low dimensions using PCA and UMAP. Because the dataset is nonlinear, the results of t-SNE projections are more promising than those of PCA and UMAP. Hence, all the projections of hidden states depicted and illustrated in this section are obtained as a result of t-SNE dimensionality reduction technique. Figure 6.11 shows the projections obtained from PCA and UMAP.

6.2.2 Analysis 2: Comparison of Two Models based on the Projection of Hidden States

The comparison of two models can be performed by following the workflow provided in Section 6.1.2. The difference in projection of their hidden state data points can be studied when two models are compared. This analysis can be particularly helpful in taking decisions when a model with the

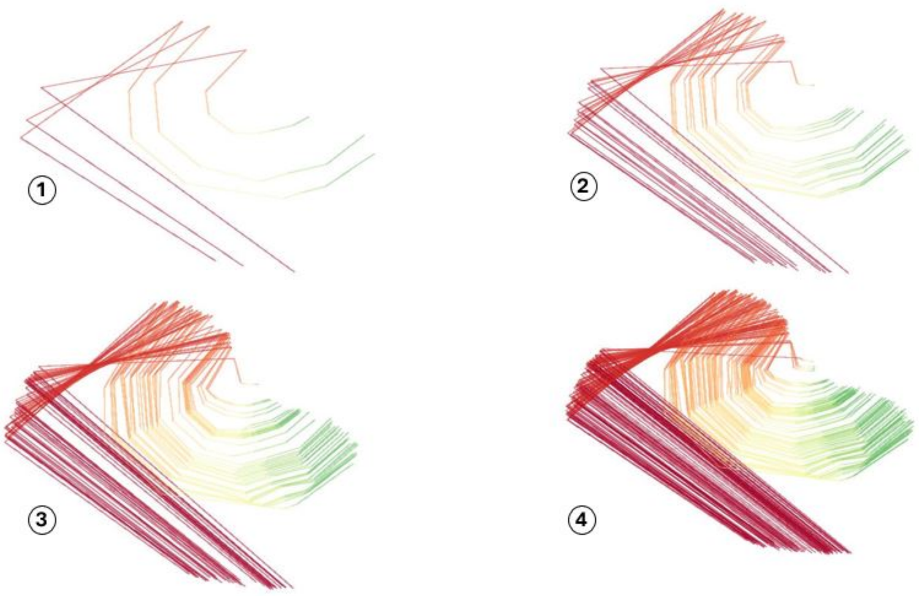


Figure 6.8: Animation of the Projection of hidden states for 10 timesteps input . (**1**): The projection of hidden states after 5 seconds; (**2**): The projection of hidden states after 10 seconds; (**3**): The projection of hidden states after 20 seconds; (**4**): The projection of hidden states after 40 seconds.

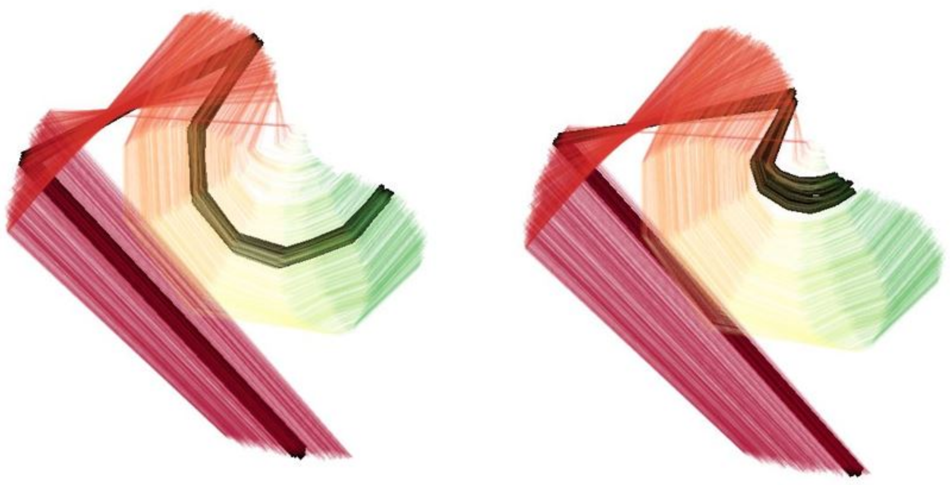


Figure 6.9: K-means clustering on the last timestep of hidden states data points results in various clusters. **Left**: Projection of low dimensional hidden states data points of Cluster 1; **Right**: Projection of low dimensional hidden states data points of Cluster 3.

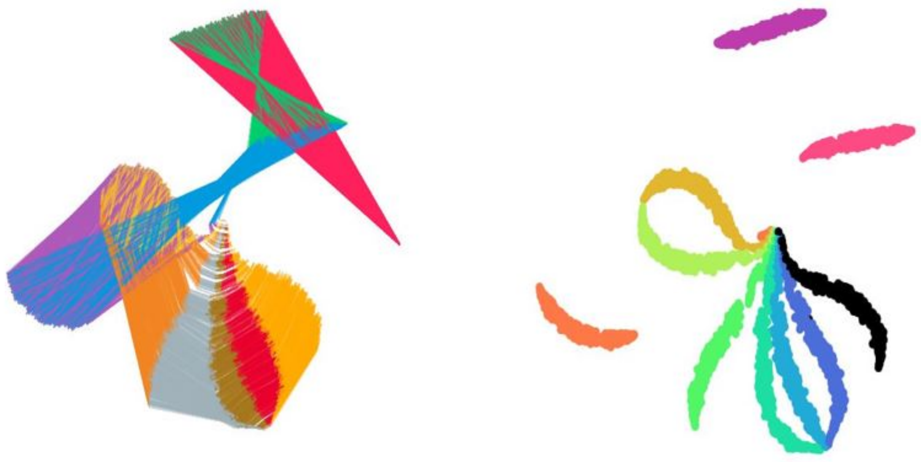


Figure 6.10: Projection of hidden states including zeroth hidden states. **Left**: Projection of low dimensional zeroth hidden states data points; **Right**: Projection of low dimensional zeroth hidden states data points as scatterpoints.

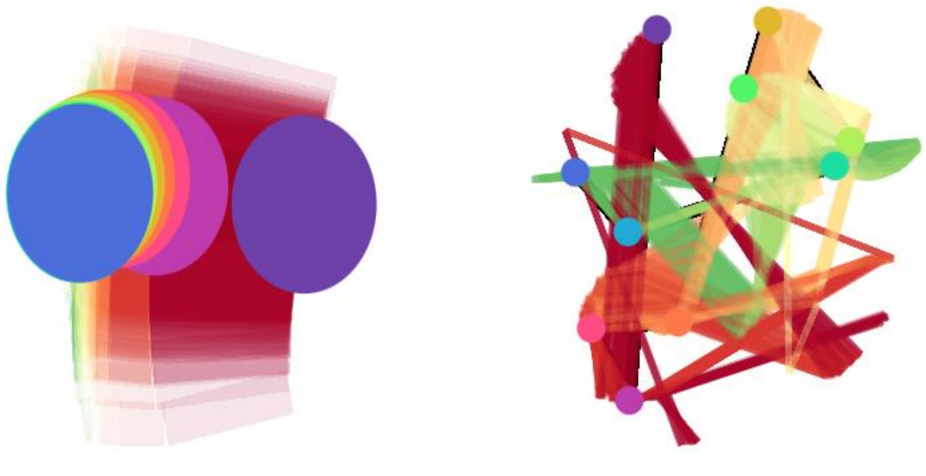


Figure 6.11: Projection of hidden states obtained from PCA and UMAP techniques. **Left**: Projection of hidden states data points from PCA; **Right**: Projection of hidden states data points from UMAP.

Configurations	**Comparison 1**		**Comparison 2**		**Comparison 3**	
	Model 1	**Model 2**	**Model 1**	**Model 2**	**Model 1**	**Model 2**
Timesteps	10	10	10	10	10	20
Epochs	10	10	10	40	10	80
Batch Size	128	128	128	128	128	256
Learning Rate	0.0001	0.0001	0.0001	0.0001	0.001	0.01
Activation Function	Relu	Sigmoid	Relu	Relu	Relu	Sigmoid
Error value	2446	3768	2446	307	2446	3712

Table 6.4: Table represents the configurations of the two compared models. Comparison 1 represents the configuration of two models with 10 sequential timesteps input and different activation functions ReLU and sigmoid. Comparison 2 represents the configuration of two models with 10 sequential timesteps input and different number of epochs 10 and 40. Comparison 3 represents the configuration of two models with 10 and 20 sequential timesteps as input and different hyperparameter configurations.

best performance is compared with a model with lower performance. Comparison 1 in Fig.6.12 illustrates that two models with 10 timesteps as input have the same hyperparameter configurations except the activation functions are compared. This indicates that changing the activation function can change the resulting projections. However, since both of the models have 5000 data points, the projections differ. When verifying the error values of both models from the `RMSE.html` page, it is known that the model 1 projected in Fig.6.12 has fewer error values than the model 2. The configurations of both the models are shown in Tab.6.4.

Comparison 2 in Fig.6.12 illustrates that two models with 10 timesteps as input have the same hyperparameter configurations except the number of epochs. This indicates that changing the number of epochs used during training can alter the resultant projections. However, since both of the models have 5000 data points, the projections differ. When the error values of both models are compared using the `RMSE.html` page, it is clear that the model 2 shown in Fig.6.12 has lower error values than the model 1 shown. The configurations of both the models are shown in Tab.6.4. This figure also illustrates that, out of ten input timesteps, timesteps 3 to 10 are near to each other in model 1 over the 2-D space. While the timesteps 3 to 10 lie close to each other, almost all of the hidden state data points are distributed near to each other in 2-D dimensional space in model 2. Because of the fact that model 2 is trained for a greater number of epochs, the overall error values result in 307. Therefore, model 2 outperforms model 1, and their hidden state projections are visualized.

Comparison 3 in Fig.6.12 illustrates that two models with 10 and 20 timesteps as input, having completely different hyperparameter configurations are compared. This indicates that changing the hyperparameter configurations can affect the resultant projections. However, since model 1 has 5000 data points and model 2 has 10000 data points, the projections differ. When the error values of both models are compared using the `RMSE.html` page, it is clear that model 1, which is in Fig.6.12 has a lower error value than model 2. The configurations of both the models are shown in Tab.6.4. This figure also illustrates that, out of 10 timesteps as input, timesteps 3 to 10 are close to each other in model 1 over the 2-D dimensional space, resulting in less error. The projections in these comparisons are obtained as result of applying t-SNE.

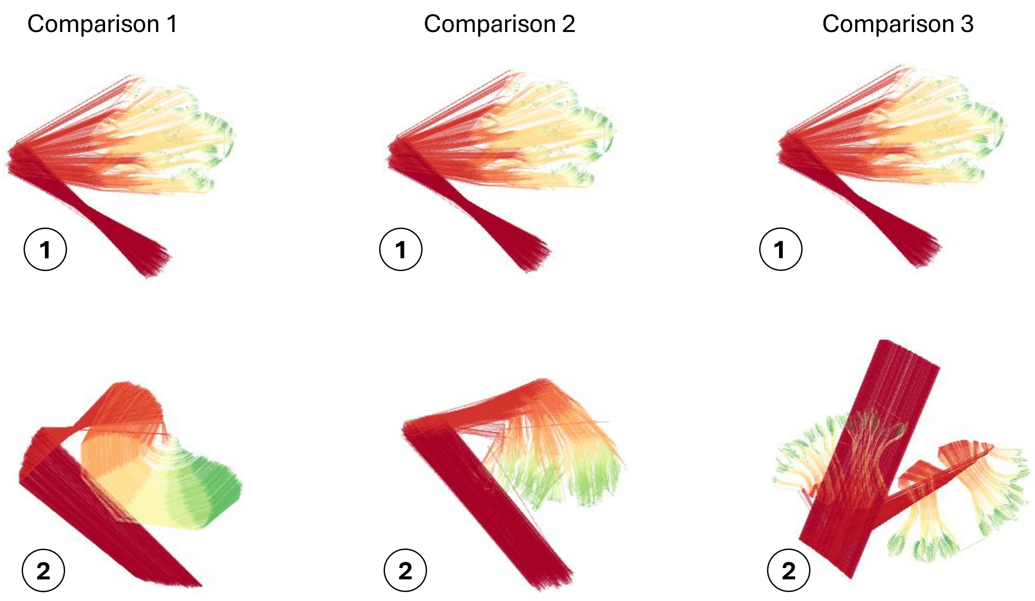


Figure 6.12: Comparisons of two models under different use cases. Comparison 1 represents two models with 10 sequential timesteps of input and different activation functions ReLU (**1**) and sigmoid (**2**) compared. Comparison 2 represents two models with 10 sequential timesteps of input and different number of epochs 10 (**1**) and 40 (**2**) compared. Comparison 3 represents the comparison of two models with 10 (**1**) and 20 (**2**) sequential timesteps as input and different hyperparameter configurations. These projections are obtained by applying t-SNE.

6.2.3 Analysis 3: Exploratory Data Analysis and Overall Performance of all the Trained Models

The parallel coordinates plot displayed in Fig.5.12 can be used to visualize the exploratory data analysis and overall performance of all trained models. In order to view this parallel coordinates plot, the user should click `Overall Error value comparison`. The parallel coordinates plot consists of seven axes, the first five of which reflect the five hyperparameters used to train the models, while the remaining two axes indicate the RMSE error values and test set accuracy of the trained models. The RMSE errors are obtained by summing all the individual RMSE errors for each data sample. Thus, the RMSE error is the sum of all the 500 data samples used for prediction. Meanwhile, four test set category values have been obtained. They are 100, 81, 56, and 0 respectively. The five hyperparameters are placed in the following order: timestep (amount of input data), epochs, batch size, learning rate, and activation function. The scales of each axis are determined by the hyperparameter values used to train the model. For easier identification, the colors of parallel coordinates plot lines are grouped depending on the timesteps. The timesteps and their respective colors are shown in the Tab.6.5

Timesteps	**Colors**
3	Purple
5	Dark Cyan
10	Light Green
20	Red

Table 6.5: Color of parallel coordinates plot lines grouped by timesteps.

Figure 5.12 shows that when the model is trained with 20 timesteps as input, the model has the lowest error and highest test set accuracy for most of the hyperparameter combinations. This implies that the model fits better with 20 input timesteps. In particular, the model trained with 20 timesteps of input along with ReLU and hyperbolic tangent activation functions results in the minimum error. While models trained with 20 timesteps of input and a sigmoid activation function result in error rates ranging from 2000 to 3000. Furthermore, regardless of other hyperparameter values, models trained with three and five timesteps produce error rates ranging from 3000 to 4000. This indicates that when trained models are trained with a greater number of timesteps, their overall performance improves. Accurate RMSE error values and test set accuracy can also be checked in the RMSE error table available on the `RMSE.html` page. The user should click `Summary Table` button on the `home.html` page to view the error values. This table can be sorted in either ascending or descending order by clicking the columns.

As mentioned in Section 5.6.6, the order in which the hyperparameters are positioned adjacent to each other impacts the end interpretation of the user. Therefore, a parallel coordinates plot was visualized with the hyperparameters rearranged in order and placed next to each other. Figure 6.13 (A) shows the hyperparameters placed adjacent to each in the above mentioned order. Figure 6.13 (B) depicts placing the learning rate axis before the error values axis instead of the activation function axis, and demonstrates that models trained with 20 timesteps as input perform well for learning rates of 0.001 and 0.0001. Thus, the lower the value of the learning rate, the better the performance of the model.

Figure 6.13 (C) shows the batch size axis being placed before the error values axis rather than the activation function axis, illustrating that models trained with 20 timesteps as input perform better for a batch size of 128. Similar analysis can be performed for epochs. Figure 6.13 (D) shows placing the epochs axis before the error values axis rather than the activation function axis, demonstrating that models trained with 20 timesteps perform well for epochs of 80 and 40. Among the first 15 minimum error values, in particular, epoch 80 contributes the most to the least error. Figure 6.13 (E) shows the timestep axis being placed before the error values axis rather than the activation function axis, illustrating that models trained with 20 timesteps as input perform better than the models trained with other timesteps as input.

These analyses indicate that the trained models are influenced by certain combinations of hyperparameters. Thus, the importance and influence of each hyperparameter value for 20 timesteps as input is thoroughly examined. The same analysis may be achieved using 3, 5, and 10 timesteps as input, as illustrated in Figs. 6.14 (F), 6.14 (G), and 6.14 (H) respectively. The model with three timesteps as input results in more error values ranging between 3000 and 4000. Meanwhile,

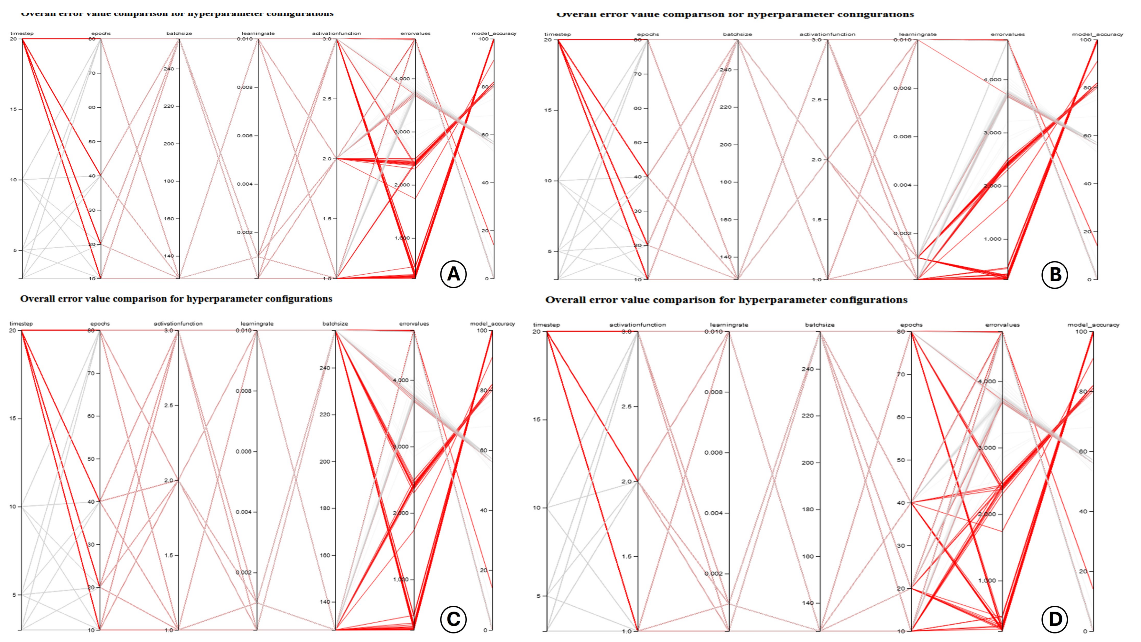


Figure 6.13: Parallel Coordinates Plot hovered over 20 timesteps input (Red). (**A**): shows activation function axis placed as 5th axis;(**B**): shows learning rate axis placed as 5th axis; (**C**): shows batch size axis placed as 5th axis; (**D**):shows epochs axis placed as 5th axis. All these 5th axes are compared with error values in the 6th axis.

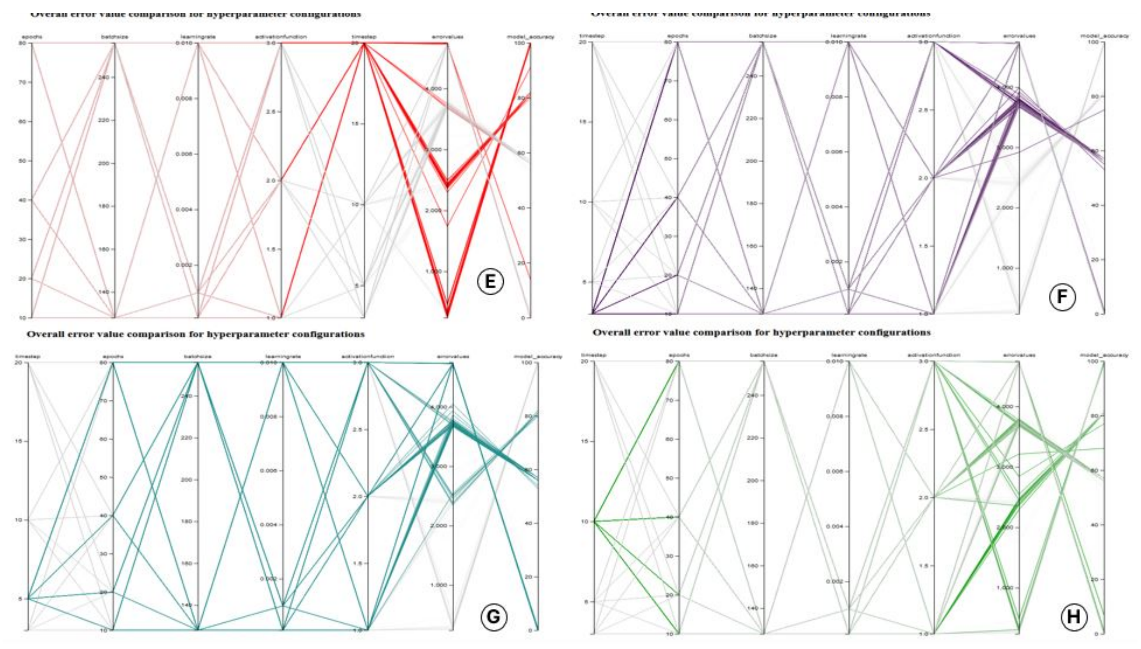


Figure 6.14: Parallel Coordinates Plot hovered over timesteps input.(**E**): shows timesteps axis placed as 5th axis and compared with error values axis in the 6th axis (Red); (**F**): shows parallel coordinates plot when hovered over 3 timesteps input (Purple); (**G**): shows parallel coordinates plot when hovered over 5 timesteps input (Dark Cyan); (**H**): shows parallel coordinates plot when hovered over 10 timesteps input (Light Green).

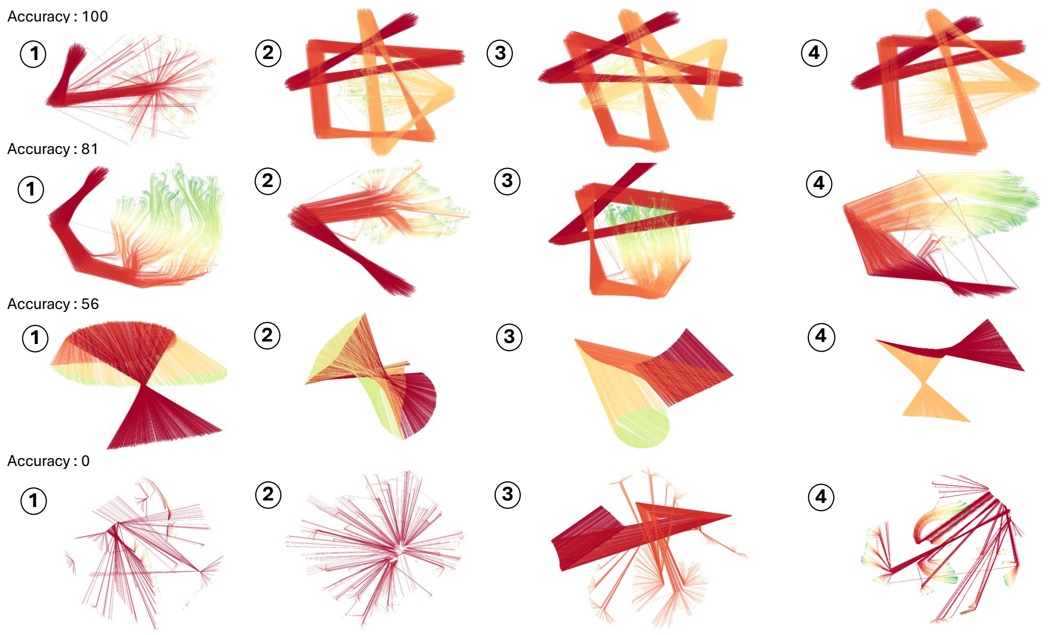


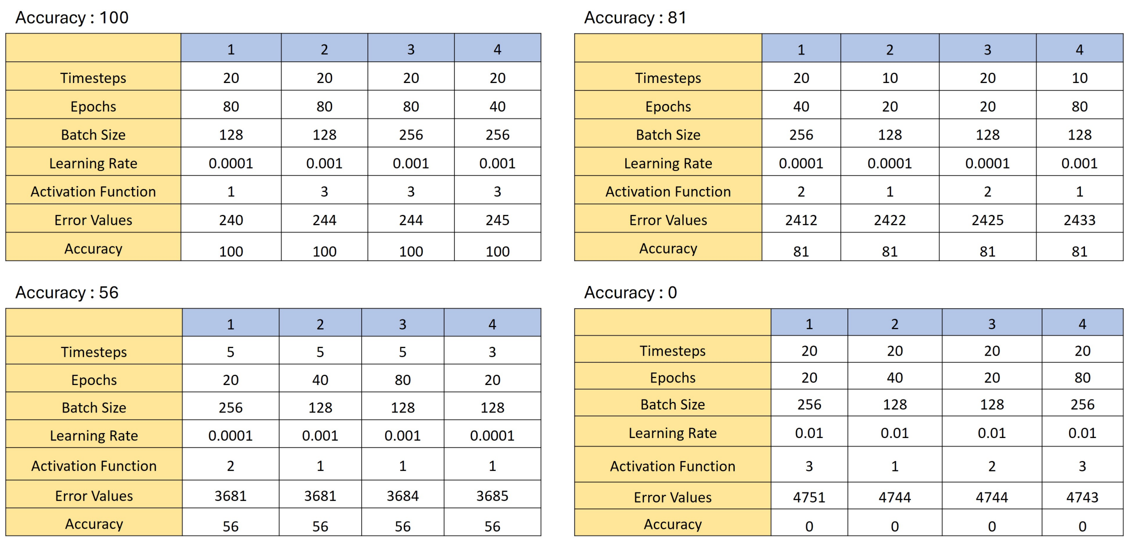
Figure 6.15: Randomly picked projections of t-SNE hidden states of the trained models grouped according to their accuracy values.

the model with five timesteps as input results in error values ranging between 2000 and 4000. In particular, a few models with five timesteps as input result in better performances than models with three timesteps as input, resulting in less error.

6.2.4 Analysis 4: Comparison of Models based on their Accuracy

The exploratory data analysis carried out using parallel coordinates plot shows the performance of the trained models by evaluating their error values and test set accuracy. Thus, it is evident from the plot that the trained models fall under four categories of accuracy. They are 100, 81, 56, and 0. Therefore, models that perform well for certain hyperparameters and models that do not perform well can be identified easily.

Figure 6.15 shows the hidden state projections of 16 trained models with accuracy levels of 100, 81, 56, and 0. The hyperparameter values of the trained models as well as their error values are shown in Fig.6.16. An analysis of the trained models can be obtained by referring to both the figures. Most models with an accuracy of 100 have 20 input timesteps, a learning rate of 0.001, and an activation function of hyperbolic tangent as hyperparameter values. These models show that this exact combination of hyperparameter values leads to improved performance. Their projections illustrate that the lines connected between two low dimensional hidden state data points converge for more number of timesteps than the models in the other accuracy categories. In specific, the earlier timesteps of an input are converged more than the latter timesteps of the same input. Therefore, the models learn more information in the earlier timesteps. For instance, models 2 and 3 in accuracy level 100 have nine timesteps and 10 timesteps converged together, respectively. Meanwhile, model 1 can be considered an outlier where no more than three timesteps are converged together. Based on their configuration values and projections, it can be concluded that models perform better for



Accuracy : 100

	1	2	3	4
Timesteps	20	20	20	20
Epochs	80	80	80	40
Batch Size	128	128	256	256
Learning Rate	0.0001	0.001	0.001	0.001
Activation Function	1	3	3	3
Error Values	240	244	244	245
Accuracy	100	100	100	100

Accuracy : 81

	1	2	3	4
Timesteps	20	10	20	10
Epochs	40	20	20	80
Batch Size	256	128	128	128
Learning Rate	0.0001	0.0001	0.0001	0.001
Activation Function	2	1	2	1
Error Values	2412	2422	2425	2433
Accuracy	81	81	81	81

Accuracy : 56

	1	2	3	4
Timesteps	5	5	5	3
Epochs	20	40	80	20
Batch Size	256	128	128	128
Learning Rate	0.0001	0.001	0.001	0.0001
Activation Function	2	1	1	1
Error Values	3681	3681	3684	3685
Accuracy	56	56	56	56

Accuracy : 0

	1	2	3	4
Timesteps	20	20	20	20
Epochs	20	40	20	80
Batch Size	256	128	128	256
Learning Rate	0.01	0.01	0.01	0.01
Activation Function	3	1	2	3
Error Values	4751	4744	4744	4743
Accuracy	0	0	0	0

Figure 6.16: Hyperparameter values of the trained models grouped according to their accuracy values. The numbering corresponds to the projections in Fig. 6.15 for each of the accuracies.

turbulence data when they are trained with a larger number of timesteps as input and a hyperbolic tangent activation function. Furthermore, the earlier timesteps of an input, in which the hidden state data points are converged together are the timesteps where the model learns a large number of hidden patterns about the turbulence data. As a result, the earlier timesteps of an input have more influence on the output prediction.

The models from accuracy 81 have 20 input timesteps, a fewer number of epochs than models from accuracy level 100, and an activation function of either ReLU or Sigmoid. As a result of these combinations of hyperparameter values, the error values for all models in this category are nearly identical. However, the projections illustrate that the lines connected between two low dimensional hidden state data points converge only for a smaller number of timesteps than the models from accuracy level 100. Due to the fewer number of epochs, the model is able to learn the hidden pattern for only a limited number of timesteps. For instance, model 1 from accuracy 81 has only four timesteps converged together despite having 20 timesteps as input. Furthermore, models 2 and 4 have nearly identical configurations except number of epochs and learning rate; their projections show that, due to change in learning rate and number of epochs, model 4 converges for a greater number of timesteps than model 2.

The category of accuracy level 56 consists of mostly models from three and five timesteps. Their projections indicate that they do not converge much and have lower accuracy values because of their lower number of timesteps as input. The models do not have enough data to learn the underlying patterns in turbulence data due to the smaller timesteps. Hence, it is clear that lower timesteps of input data have only limited information about the turbulence data. For instance, model 3 has 80 epochs, but due to 5 timesteps as input, the model is not provided enough information to predict the accurate output.

The models from accuracy 0 indicate that models do not fit well for the combination of hyperparameter values in which they were trained. As a result, their projections show that they converge only for a few timesteps, despite having 20 timesteps as inputs. For instance, models 1 and 2 converge only for timesteps 1 and 2, while the remaining timesteps from 3 to 20 diverge to a greater extent. Therefore, models do not learn much about the hidden patterns, resulting in higher error values. Thus, the projections converging for fewer timesteps leads to higher error.

The activation function plays a vital role in influencing the output. This is evident from the parallel coordinates plot where in particular hyperbolic tangent activation function is the reason behind the models with better performance. Another important hyperparameter that plays a significant role is learning rate. It can be observed that models with 0.001 as learning rate has better accuracy when trained with 20 timesteps of input. Additionally, the models with 0.0001 as learning rate perform better to some extent and it can be seen from the models with accuracy level 81. Meanwhile, the models with 0.01 as learning rate has lower accuracy. This is because the model with higher learning rate of 0.01 hinders the model from performing better. Overall, for this particular turbulence data the model trained with more timesteps as inputs and moderate learning rate perform well resulting in predicting more accurate closure terms.

The goal of exploring the internal hidden states of RNN, comparison of models with different configurations based on their projection and accuracy, and the provision to provide an interactive environment for the user to explore and analyze was well demonstrated with the turbulence data. Additionally, the exploratory data analysis on the turbulence data provides a general overview, which increases the feasibility of employing this visual analytics tool for future research work. Thus, the goals of this thesis were achieved and demonstrated successfully.

7 Conclusion and Future Scope

The interpretation and exploration of internal hidden states of RNN models for a regression task are an ongoing challenge. To address this ongoing challenge, in this thesis, a visual analytics approach for exploring the internal hidden state information of the RNN models has been proposed and a visual analytics tool has been presented. The contributions made in the fields of visual analytics, machine learning, dimensionality reduction, and so on have been utilized to develop this approach.

7.1 Summary

To address the problem statement, a visual analytics tool with several visualization approaches has been developed. This tool was developed particularly for the GRU model and demonstrated with turbulence data from the aerodynamics domain. The GRU model is trained using a number of hyperparameter configurations such as batch size, epochs, activation function, learning rate, and amount of input data. The tool allows the user to visualize models that have been trained for specific combinations of hyperparameter values. It also allows the user to choose and compare two different models. Thus, this thesis concludes three significant pieces of information. To begin with, it demonstrates to the user how the underlying hidden state information of the RNN model is processed when a time sequential input is provided. Secondly, it sheds light on the turbulence dataset by visualizing the hidden state projections in 2-D space for various hyperparameter combinations. Finally, the various visualization techniques of the tool, such as heat map, explain how each hidden state for all the hidden units in the GRU layer contributes to output prediction. Furthermore, the scatterpoints and the animation features suggest an alternative perspective for visualizing the projection of the hidden states.

Besides displaying the projection of hidden states for each input, this tool also provides another significant information about the overall view of the performance of the trained models for the turbulence dataset. It clearly indicates that the turbulence data applied to the GRU model performs well for configuration of certain hyperparameter values by resulting in a minimal error. Thus, the influence of hyperparameters on the output was investigated. Finally, the comparison of GRU models shows how the projections of the hidden states differ as the hyperparameter values changes. It also identifies which factors play a significant influence in distinguishing between better and worse performing models. Additionally, it implies that each of the 288 trained configurations has a unique projection of hidden states. Overall, this tool addresses the issue of users who view the machine learning application as a blackbox model. As a result, the goal of interpreting and exploring the internal hidden states of RNN models for a regression task has been accomplished.

7.2 Future Scopes

Based on this thesis research, we believe the following points are some of the promising future scopes to explore:

- The GRU model was only trained for five hyperparameter combinations with a limited range of values. This can be further investigated by considering other available hyperparameters such as number of neurons, number of hidden layers and training them with a wide range of values. Thus, the behaviour and performance of the model for the dataset and the internal hidden state information can be analyzed. Furthermore, training the model with a larger number of hyperparameters gives the user more alternatives and possibilities for selecting and exploring visualizations.
- K-means clustering was applied to the last timesteps of low dimensional hidden state data points, and hence the similarities between the hidden states were explored. Instead, other clustering algorithms like density based clustering, distribution based clustering can be considered, which would result in finding some additional insights when combined with the previously explained future scope.
- The values of certain tuneable parameters like perplexity, random state, and early exaggeration used in the dimensionality reduction techniques were constant for all the trained models. These parameters influence the end results to a certain level. As a consequence, the values of these tuneable parameters can be altered and further studied, which can lead to various outcomes that provide insightful information about the turbulence data.
- The concepts of SHAP and LIME were applied to the turbulence dataset. However, a promising insight could not be achieved due to the time series nature of the dataset. As a result of the large dimensionality of the dataset, further preprocessing is required before using the approaches. The preprocessing of time series data requires more precise steps in order to preserve much of the time series information from the original data. Therefore, further research can be carried out by preprocessing the data in such a way that the data is suitable to apply and explore the feature importance with these techniques.

Bibliography

[AAB+16] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, et al. “Tensorflow: Large-scale machine learning on heterogeneous distributed systems”. In: *arXiv preprint arXiv:1603.04467* (2016). DOI: https://doi.org/10.48550/arXiv.1603.04467 (cit. on p. 39).

[Bar89] H. B. Barlow. “Unsupervised learning”. In: *Neural computation* 1.3 (1989), pp. 295–311. DOI: https://doi.org/10.1162/neco.1989.1.3.295 (cit. on p. 23).

[BD13] M. Bostock, J. Davies. “Code as Cartography”. In: *The Cartographic Journal* 50 (May 2013), pp. 129–135. DOI: 10.1179/0008704113Z.00000000078 (cit. on p. 45).

[Bis19] E. Bisong. “Introduction to Scikit-learn”. In: *Building Machine Learning and Deep Learning Models on Google Cloud Platform: A Comprehensive Guide for Beginners*. Berkeley, CA: Apress, 2019, pp. 215–229. ISBN: 978-1-4842-4470-8. DOI: 10.1007/978-1-4842-4470-8_18. URL: https://doi.org/10.1007/978-1-4842-4470-8_18 (cit. on p. 39).

[CCD08] P. Cunningham, M. Cord, S. J. Delany. “Supervised learning”. In: *Machine learning techniques for multimedia*. Springer, 2008, pp. 21–49. DOI: 10.1007/978-3-540-75171-7_2 (cit. on p. 23).

[Cho15] F. Chollet. *keras*. https://github.com/fchollet/keras. 2015 (cit. on p. 31).

[CMJ+20] A. Chatzimparmpas, R. M. Martins, I. Jusufi, K. Kucher, F. Rossi, A. Kerren. “The state of the art in enhancing trust in machine learning models with the use of visualizations”. In: *Computer Graphics Forum*. Vol. 39. 3. Wiley Online Library. 2020, pp. 713–756. DOI: https://doi.org/10.1111/cgf.14034 (cit. on pp. 17, 31).

[Com20a] W. Commons. *File:Artificial neural network.svg — Wikimedia Commons, the free media repository*. [Online; accessed 16-September-2022]. 2020. URL: https://commons.wikimedia.org/w/index.php?title=File:Artificial_neural_network.svg&oldid=494529927 (cit. on p. 25).

[Com20b] W. Commons. *File:Gated Recurrent Unit, type 2.svg — Wikimedia Commons, the free media repository*. [Online; accessed 15-September-2022]. 2020. URL: https://commons.wikimedia.org/w/index.php?title=File:Gated_Recurrent_Unit,_type_2.svg&oldid=470031589 (cit. on p. 29).

[Com22] W. Commons. *File:Recurrent neural network unfold.svg — Wikimedia Commons, the free media repository*. [Online; accessed 18-September-2022]. 2022. URL: https://commons.wikimedia.org/w/index.php?title=File:Recurrent_neural_network_unfold.svg&oldid=655242383 (cit. on p. 28).

[CT05] K. A. Cook, J. J. Thomas. “Illuminating the Path: The Research and Development Agenda for Visual Analytics”. In: (May 2005). URL: https://www.osti.gov/biblio/912515 (cit. on p. 21).

[CVG+14] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, Y. Bengio. “Learning phrase representations using RNN encoder-decoder for statistical machine translation”. In: *arXiv preprint arXiv:1406.1078* (2014). DOI: https://doi.org/10.48550/arXiv.1406.1078 (cit. on p. 27).

[DG17] D. Dua, C. Graff. *UCI Machine Learning Repository*. 2017. URL: http://archive.ics.uci.edu/ml (cit. on p. 32).

[DR22] S. Dutta, S. Roy. “Complex Network Visualisation Using JavaScript: A Review”. In: *Intelligent Systems*. Ed. by S. K. Udgata, S. Sethi, X.-Z. Gao. Singapore: Springer Nature Singapore, 2022, pp. 45–53. ISBN: 978-981-19-0901-6. DOI: 10.1007/978-981-19-0901-6_5 (cit. on p. 45).

[GMW21] R. Garcia, T. Munz, D. Weiskopf. “Visual analytics tool for the interpretation of hidden states in recurrent neural networks”. In: *Visual Computing for Industry, Biomedicine, and Art* 4.1 (2021), pp. 1–13. DOI: https://doi.org/10.1186/s42492-021-00090-0 (cit. on p. 31).

[KB20] M. Kurz, A. Beck. “A machine learning framework for LES closure terms”. In: *arXiv preprint arXiv:2010.03030* (2020). DOI: https://doi.org/10.48550/arXiv.2010.03030 (cit. on pp. 30, 32, 40, 41).

[KKE10] E. D. Keim, J. Kohlhammer, G. Ellis. *Mastering the Information Age: Solving Problems with Visual Analytics, Eurographics Association*. 2010 (cit. on pp. 21, 22).

[KLM96] L. P. Kaelbling, M. L. Littman, A. W. Moore. “Reinforcement learning: A survey”. In: *Journal of artificial intelligence research* 4 (1996), pp. 237–285. DOI: https://doi.org/10.1613/jair.301 (cit. on p. 23).

[KMS+08] D. A. Keim, F. Mansmann, J. Schneidewind, J. Thomas, H. Ziegler. “Visual analytics: Scope and challenges”. In: *Visual data mining*. Springer, 2008, pp. 76–90. DOI: 10.1007/978-3-540-71080-6_6 (cit. on p. 17).

[LL21] H. S. Lee, J. Lee. “Applying Artificial Intelligence in Physical Education and Future Perspectives”. In: *Sustainability* 13.1 (Jan. 2021), p. 351. ISSN: 2071-1050. DOI: 10.3390/su13010351. URL: http://dx.doi.org/10.3390/su13010351 (cit. on p. 24).

[LWLZ17] S. Liu, X. Wang, M. Liu, J. Zhu. “Towards better analysis of machine learning models: A visual analytics perspective”. In: *Visual Informatics* 1.1 (2017), pp. 48–56. DOI: https://doi.org/10.1016/j.visinf.2017.01.006 (cit. on p. 17).

[Mar94] M. A. Marcinkiewicz. “Building a large annotated corpus of English: The Penn Treebank”. In: *Using Large Corpora* 273 (1994) (cit. on p. 31).

[MCZ+17] Y. Ming, S. Cao, R. Zhang, Z. Li, Y. Chen, Y. Song, H. Qu. “Understanding hidden memories of recurrent neural networks”. In: *2017 IEEE Conference on Visual Analytics Science and Technology (VAST)*. IEEE. 2017, pp. 13–24. DOI: 10.1109/VAST.2017.8585721 (cit. on p. 31).

[ON15] K. O’Shea, R. Nash. “An introduction to convolutional neural networks”. In: *arXiv preprint arXiv:1511.08458* (2015). DOI: https://doi.org/10.48550/arXiv.1511.08458 (cit. on p. 26).

[OPM16] M. Oprea, M. Popescu, S. F. Mihalache. “A neural network based model for PM 2.5 air pollutant forecasting”. In: *2016 20th International Conference on System Theory, Control and Computing (ICSTCC)*. IEEE. 2016, pp. 776–781 (cit. on p. 32).

[PC96] U. Piomelli, J. R. Chasnov. "Large-eddy simulations: theory and applications". In: *Turbulence and transition modelling*. Springer, 1996, pp. 269–336. DOI: DOI:10.1007/978-94-015-8666-5_7 (cit. on p. 32).

[RRL+20] G. T. Reddy, M. P. K. Reddy, K. Lakshmanna, R. Kaluri, D. S. Rajput, G. Srivastava, T. Baker. "Analysis of Dimensionality Reduction Techniques on Big Data". In: *IEEE Access* 8 (2020), pp. 54776–54788. DOI: 10.1109/ACCESS.2020.2980942 (cit. on p. 32).

[Sah20] S. Sah. *Machine Learning: A Review of Learning Types*. July 2020. DOI: 10.20944/preprints202007.0230.v1 (cit. on p. 23).

[SCW+15] X. Shi, Z. Chen, H. Wang, D.-Y. Yeung, W.-K. Wong, W.-c. Woo. "Convolutional LSTM network: A machine learning approach for precipitation nowcasting". In: *Advances in neural information processing systems* 28 (2015) (cit. on p. 32).

[SGA+19] P. A. Srinivasan, L. Guastoni, H. Azizpour, P. Schlatter, R. Vinuesa. "Predictions of turbulent shear flows using deep neural networks". In: *Physical Review Fluids* 4.5 (2019), p. 054603. DOI: https://doi.org/10.1103/PhysRevFluids.4.054603 (cit. on p. 32).

[SM19] R. C. Staudemeyer, E. R. Morris. "Understanding LSTM–a tutorial into long short-term memory recurrent neural networks". In: *arXiv preprint arXiv:1909.09586* (2019). DOI: https://doi.org/10.48550/arXiv.1909.09586 (cit. on p. 29).

[SWJ+20] Q. Shen, Y. Wu, Y. Jiang, W. Zeng, A. K. H. LAU, A. Vianova, H. Qu. "Visual Interpretation of Recurrent Neural Network on Multi-dimensional Time-series Forecast". In: *2020 IEEE Pacific Visualization Symposium (PacificVis)*. 2020, pp. 61–70. DOI: 10.1109/PacificVis48177.2020.2785 (cit. on p. 32).

[Tom86] R. S. M. Tom M. Mitchell Jaime G. Carbonell. *Machine Learning*. Vol. 12. Springer New York, NY, 1986. DOI: https://doi.org/10.1007/978-1-4613-2279-5 (cit. on p. 22).

[VV99] J. Van Wijk, E. Van Selow. "Cluster and calendar based visualization of time series data". In: *Proceedings 1999 IEEE Symposium on Information Visualization (InfoVis'99)*. 1999, pp. 4–9. DOI: 10.1109/INFVIS.1999.801851 (cit. on p. 32).

[Wer90] P. J. Werbos. "Backpropagation through time: what it does and how to do it". In: *Proceedings of the IEEE* 78.10 (1990), pp. 1550–1560. DOI: 10.1109/5.58337 (cit. on p. 26).

[XLX17] H. Xie, J. Li, H. Xue. "A survey of dimensionality reduction techniques based on random projection". In: *arXiv preprint arXiv:1706.04371* (2017) (cit. on p. 35).

[YJG03] A. B. Yoo, M. A. Jette, M. Grondona. "Slurm: Simple linux utility for resource management". In: *Workshop on job scheduling strategies for parallel processing*. Springer. 2003, pp. 44–60. DOI: 10.1007/10968987_3 (cit. on p. 39).

[YRXZ20] J. Yan, R. Rong, G. Xiao, X. Zhan. "HiddenVis: a Hidden State Visualization Toolkit to Visualize and Interpret Deep Learning Models for Time Series Data". In: *bioRxiv* (2020). DOI: 10.1101/2020.12.11.422030. eprint: https://www.biorxiv.org/content/early/2020/12/12/2020.12.11.422030.full.pdf. URL: https://www.biorxiv.org/content/early/2020/12/12/2020.12.11.422030 (cit. on p. 31).

All links were last followed on October 28, 2022.

Declaration