School of Computing

FACULTY OF ENGINEERING

UNIVERSITY OF LEEDS

Predicting worm behaviour using deep learning methods

Ruochen Li

Submitted in accordance with the requirements for the degree of MSc. Advanced Computer Science (Data Analytics)

2019/2020

The candidate confirms that the following have been submitted*:*

Items	**Format**	**Recipient(s) and Date**
Report (Digital Copy)	*Report*	*Minerva (28/08/2020)*
Programming codes	*GitLab Repository* *(Appendix B)*	*Supervisor, assessor (28/08/2020)*

Type of Project: ________ **Exploratory Software** ____________________________

(Signature of student) ______________________________

Summary

Nowadays, deep learning is one of the most popular learning approaches for prediction tasks, by learning and training the time-related features from the input data, the deep learning models are able to predict new data samples with high accuracy and reliability. Since more and more experiments are proving that the behaviours of microbes are connected to human disease state and genetic changes. If could predict the behaviours of these creatures by using deep learning methods, then it will be a very convenient and cost-effective way for researchers to analyse. In this project, I will apply several deep learning methods to predict and analyse the behaviours of the nematode called C.elegans.

Acknowledgements

I would like to thank my supervisor Dr. Thomas Ranner for supporting me during the process of development of this project.

I would like to thank my project assessor Dr. Haiko Muller for his suggestions and feedback for my project.

Table of Contents

Chapter 1
Introduction

As we all know, humans are closely related to nature, in addition to the animals we can normally see like dogs, cats, birds, there are many other creatures in nature that we cannot see using our eyes. With the development of the age and technology, more and more scientists are discovering that many creatures that we cannot see with our eyes have some neural structures and functions similar to those of humans, studying their movements and morphologies could go a long way toward helping humans with health problems.

For example, C. elegans is such a freely living nematode that we can not see without the help of microscope, this project will mainly focus on analyzing the behaviours of this nematode with certain methods.

1.1 Project Aim

The main aim of this project is to explore the predictability of behaviours of the nematode C. elegans using deep learning approaches. The predictability will be explored by predicting its possible behaviours based on some previous motions and movement patterns in different dimensional representations. The evaluation will be implemented to compare the predicted results by different deep learning models.

1.2 Objectives

1. Design and implement several deep learning architectures to predict the behaviours of C.elegans.
2. Explore the predictability of C.elegans in full dimensional representations and low dimensional representations.
3. Use certain evaluation metrics to evaluate the results generated by the selected methods.
4. Compare the performance of different learning approaches in meaningful ways.

1.3 Deliverables

1. The Gitlab repository that contains all the source of the system
2. The MSC report for this project which contains following chapters:
    a) Introduction: Aim, objectives, deliverables

b) Background Research: Literature Survey, Methods, Selection of Methods

c) Design of experiment: Task design, Model design, Evaluation design

d) Implementation and Result: Implementation process, Results

e) Evaluation: Comparison, discussion

f) Conclusion: Conclusion, limitation, future work and personal reflection

1.4 Ethical, legal, and social issues

1.4.1 Ethical Issues

The dataset I used for this project was provided by my supervisor, and I was also authorised to enter the database by my supervisor.

1.4.2 Legal Issues

All the library and environment I used for this project are open-source and these tools can be used in non-commercial purposes.

1.4.3 Social Issues

The purpose of this project is only for personal project.

Chapter 2
Background Research

2.1 Literature Survey

2.1.1 Introduction to Caenorhabditis elegans (C. elegans)

The nematode Caenorhabditis elegans has a species called C. elegans. According to Ann K et al. (2015), Caenorhabditis elegans is a tiny (around 1 millimetre in length) and free-living nematode, which does not have bones or a circulatory system as humans do. It is a non-hazardous, non-infectious, non-pathogenic and non-parasitic organism that can be found around the world especially in the soil (Mark et al., 2015). Although there is a lot of research on C. elegans after it was first described and isolated by Emile Maupas over 100 years ago, its prominence should be attributed to Sydney Brenner, who developed the genetic model based on C. elegans to help understand problems about biology and neurobiology (Ann K et al., 2015). Figure 2.1 shows a view of C. elegans under a microscope.

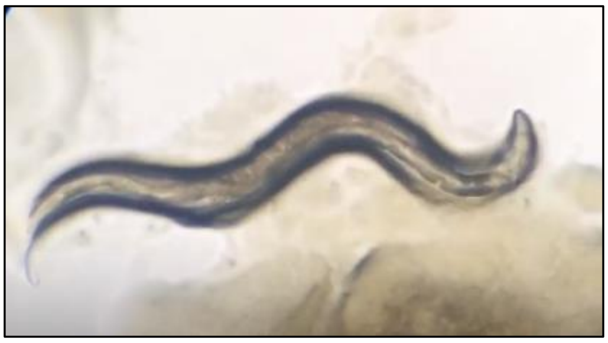
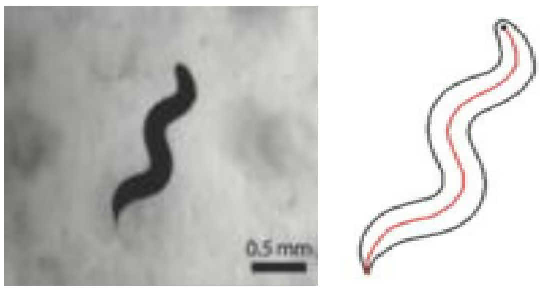


Figure 2.1 *C. elegans under microscope after a hundred times magnification (Uluç, 2017)*

2.1.2 Features of Caenorhabditis elegans

As Alan and Aleksandra (2019) said, C. elegans is commonly used for biology and neurobiology because this kind of nematode worm is significantly simpler than a human, but it shares many similarities at the molecular level as a human. For example, C. elegans genome have important functional counterparts in humans, in that at least 50% of C. elegans’ s genes are similar to human genes and its genome was one of the first that mapped completely. Furthermore, although C. elegans’ s nervous system is simpler than humans’, researchers have still explored its complete map of 302 neurons and 7000 connection points (American Institute of Physics, 2013). Moreover, C. elegans is a hermaphrodite with self-fertilization properties, so it can produce over 1000 eggs every day, which will provide a huge number of samples for researchers to work with.

In addition to the Neural system, Eviatar et al. (2014) argued that the nematode worm C. elegans has a simple morphology; its length and width will be relatively constant while crawling, swimming and resting, so that we can capture its motion just by exploring the coordinate system of its midline over a period of time as shown in Figure 2.2. These features of C. elegans can help researchers to build models for analysing neurological diseases such as Parkinson’s, and some Neural activities can be mapped onto many other animals.



A B

Figure 2.2 *A sample frame of C. elegans under microscope and its corresponding midline skeleton (Greg J et al., 2008)*

2.1.3 Behaviour analysis of Caenorhabditis elegans

According to Marlena et al. (2016), behaviours play an important and fundamental role in nervous system and motor diseases; by exploring the behaviours of animals, disease state or genome variation can be connected easily for further analysis.

Therefore, the quantification of animal behaviour would be important and valuable (Kezhi et al., 2017), but also a challenge because animal behaviour is in a high-dimensional space, each joint in vertebrates and each muscle in vertebrates will create a new perspective in the domain (Gomez-Marin et al., 2014).

Today, there are two divergent traditions of rooting the behaviours of animal: (i) to create a well-controlled environment for animals, then observe their behaviours by forcing them to reflect different behaviours; (ii) to observe their behaviours in the more natural contexts. Moreover, we hope that there exist simpler organisms that can provide the system for modelling to resolve the tension between these approaches, then researchers can quantify the natural system easily (Greg J et al., 2008).

Since C. elegans is such a simple organism with many suitable features—it is very good for researchers to build models to represent natural contexts—when it moves on the surface of an agar plate, C. elegans will just propagate bending waves along its body depending on the environmental stimulus (Bertalan and Andre, 2016). Greg J et al. (2008) indicated that the behaviours of C. elegans can be affected by chemical, thermal and mechanical stimuli though their behaviours are unconstrained.

As shown in Figure 2.2B, we can simply draw the midline and find its corresponding coordinate system in each two-dimensional shape of individual C. elegans under video microscopy (Figure 2,2A); the posture space of C. elegans can therefore be defined by the angles of its midline (Figure 2.2B red line), which is in the low-dimensional space, with four dimensions accounting for over 95% variance of overall shapes (Greg J et al., 2008).

As shown in Figure 2.3 (red lines), four basis shapes (or eigenworms) can be used to reconstruct any worm posture in different frames; we can also find that the amplitudes (grey lines) will change along the dimension when shapes are projected onto these four basis shapes (Andre et al., 2012).

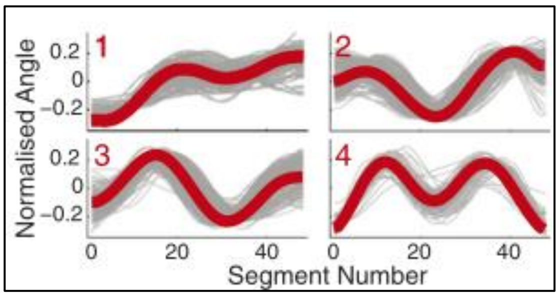


Figure 2.3 *Four basis shapes (eigenworms) represented using red lines, grey lines represent eigenworms derived from mutant strains (Andre et al., 2012)*

Based on the finding of basis shapes, the behaviours of C. elegans can be described quantitatively and the equations of worm behaviours can also be partially summarized in the space.

2.1.4 Predictability of worm behaviours

To explore the predictability of worm behaviours, Kezhi et al. (2017) tried to use the deep learning approach to quantitatively analyse the behaviour of the worm. They demonstrated that multi-layer Recurrent Neural Network (RNN) can be used to generate diverse behaviours of C. elegans with high accuracy compared with the real behaviours. Accordingly, they used the long short-term memory (LSTM) algorithm of RNN to train the predictive model, followed by which the trained model can predict next worm shapes based on the back-propagation theory by inputting previous worm shapes. Figure 2.4 shows the over design of their LSTM algorithm, which contains four LSTM layers, two dropout layers and one dense layer. Furthermore, since this is a kind of time series problem, the structure they used for LSTM was “many-to-one” (also called “univariate”), which means that it would use multiple previous steps to predict the next future step.

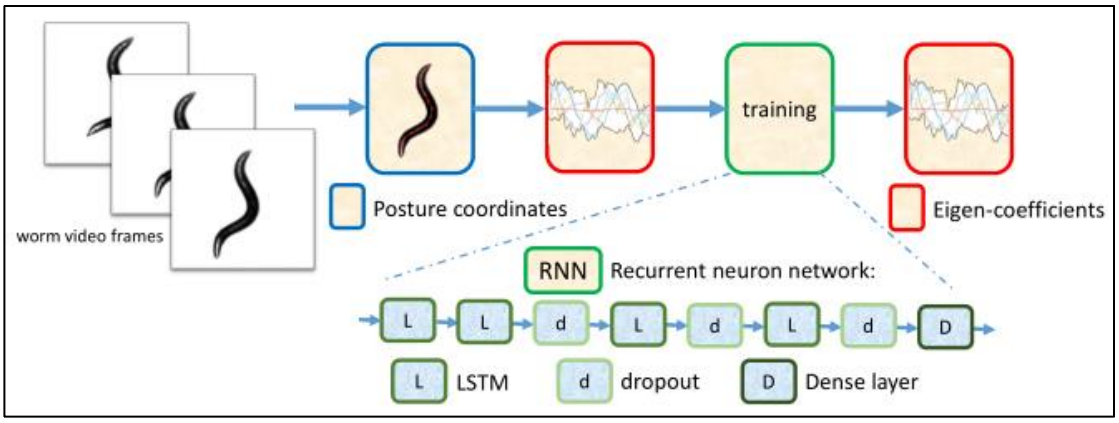


Figure 2.4 *Worm behaviour prediction using LSTM algorithm (Kezhi et al., 2017)*

As such, not only did they show that deep learning can be a great approach to predict worm behaviours, they trained the classifier using LSTM, which can distinct worm strains based on their shape changes and generated trajectories, especially, because the interpretable cells of LSTM can help to simulate the neuron activities since those cells include memory which correlates with the property of the modelled system when training LSTM algorithm using real world data (Andrej et al., 2016), so that there may exist similarities between neuron activities of C. elegans and artificial neuron activities.

2.2 Methods and Techniques

2.2.1 Machine Learning

According to Tom M (1998), the definition of machine learning is: “A computer program is said to learn from past experience E with respect to some class of tasks T with the measurement element P if its performance at tasks in T, as measured by P, and performance improves with experience E”. In brief, machine learning is a kind of technology of making computers act and learn as humans after a series of training steps by feeding them data and information in the form of real-world data (Daniel, 2020).

Nowadays, with the development of internet technology, there is a huge amount of data with complex structures and features such as images, sentiment information and comments data, which is difficult for humans to explore and analyse for different purposes. However, machine learning can analyse everything automatically without human intervention; they will train models based on the inputting data with specified rules; after a period of training steps, these models can output the prediction or classification results as expected with high accuracy.

Under the field of machine learning technology, three commonly used sub-approaches exist, called supervised learning and un-supervised learning.

2.2.1.1 Supervised Learning

Isha (2018) indicated that the necessary condition for training the model using supervised learning is that the dataset must be labelled; therefore, each sample in the dataset should be tagged with the expected type of keyword.

Figure 2.5 presents the workflow of supervised learning algorithms. First, we need to divide the dataset into training and testing sets, followed by extracting features of training data as feature vectors. Subsequently, these feature vectors are sent with its labels to the model for training. Finally, the same approaches as training data to prepare testing data is applied followed by using the trained model to predict the new labels of testing data to evaluate the performance of the model.

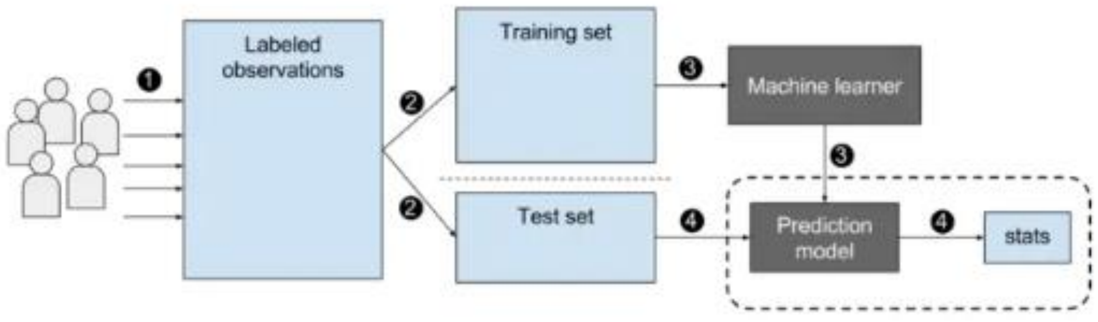


Figure 2.5 *Workflow of supervised learning algorithm (Isha, 2018)*

Based on the requirements of the task, the supervised learning approach is mostly used for solving classification and regression problems.

For the former, the data is usually discrete; then the model generates the labels for different discrete variables. For example, if the input data is flowers' features and labels are the variety of these flowers, then the trained model should be able to classify some new data with new features into its corresponding variety.

For the latter, the data is usually continuous; it is the task of building a mapping function from the input data to generate a continuous output value where the output value should be real such as integers or floating point, etc. (Jason, 2019). For example, the prediction of house price for a specific year can be regarded as a regression problem, then solved by supervised learning.

2.2.1.2 Unsupervised Learning

The other commonly used approach in machine learning is unsupervised learning; unlike supervised learning, unsupervised algorithms cannot solve classification and regression directly since now we don't know what the possible output results might be. Consequently, if the data does not have the desired outcomes such as labels, unsupervised learning algorithms would be the best choice for it. Figure 2.6 shows the basic workflow of unsupervised learning algorithms; we don't need to divide the dataset into training and testing sets as supervised learning; the algorithms would automatically generate clustering or association results based on the unlabelled features (Isha, 2018).

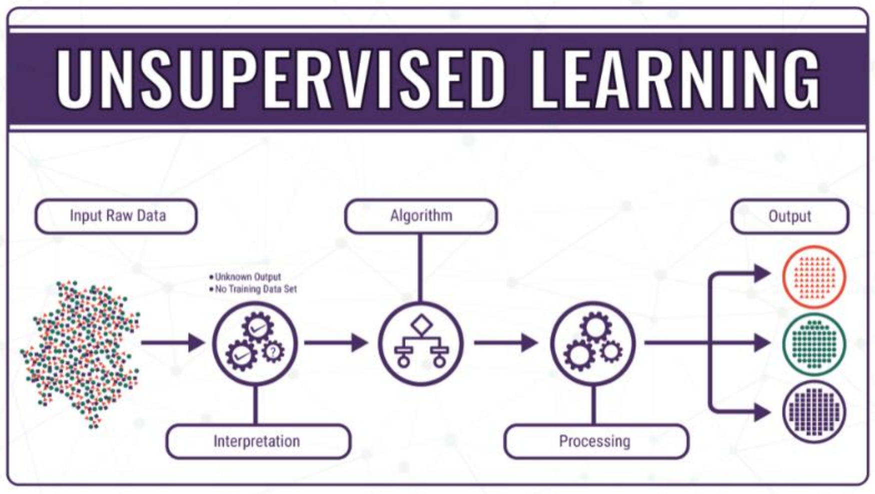


Figure 2.6 *Workflow of unsupervised learning algorithm (Ana, 2020)*

With regards to clustering, algorithms allow us to automatically classify the given dataset based on similarity such as colour, size, shape, etc. For example, if the dataset contains a large number of similar images, the clustering approach can be used to distinct and group them conveniently.

In terms of association property, the features of a certain sample may correlate with features of others; by exploring the key attributes of a data point, unsupervised learning algorithms can identify sets of attributes that commonly occur together in the dataset (datarobot, 2020). For example, if the shopping cart includes apples, straws and a juicer, the algorithms may recommend a peeler to this shopping cart because these attributes are associated with the idea of making a cup of juice.

2.2.2 Deep Learning

According to FAIZAN (2017), deep learning is part of the particular kind of machine learning techniques that can achieve great power and flexibility by training and learning to represent the real-world as hierarchical concepts, with each concept defined by previous simpler concepts, followed by using less abstract representations to compute more abstract ones. In general, deep learning is an example of AI function that can simulate the human brain's functions to solve some real-world problems automatically (Marshall, 2019). For example, applications such as self-driving cars, natural language processes, intelligent healthcare systems and fraud detection can make a major breakthrough by the development of deep learning.

Compared to traditional machine learning approaches, deep learning can process an enormous amount of unstructured and unlabelled data drawn from sources such as shopping websites, social media and online medical systems. Figure 2.7 presents the comparation between traditional machine learning and some deep learning architectures, showing that traditional machine learning algorithms (red line) contribute the better performance if the amount of data is small; algorithms such as SVM, linear regression and random forests can process the data very effectively (Albahnsen, 2017). However, the performance of machine learning will become plateauing and unimprovable along with the advent of the big data era. As such, researchers have proved that different deep learning architectures (especially deep neural network) can easily and efficiently handle the large-scale data with the cheaper computation costs.

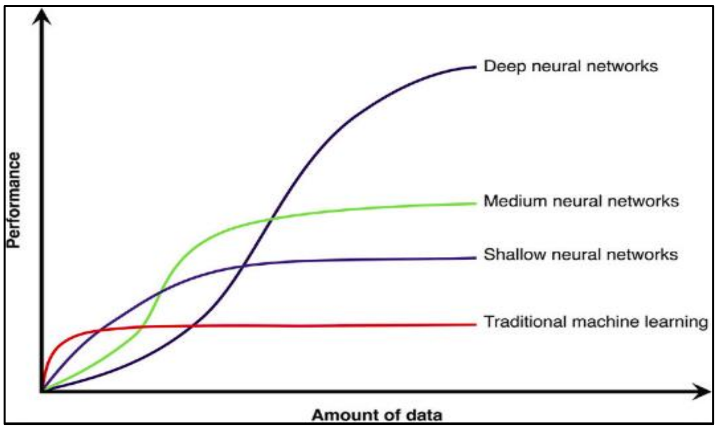


Figure 2.7 *The performance of Deep learning and Machine learning based on the amount of data (Tanmay, 2019)*

Moreover, deep learning and machine learning have two different mechanisms for data preparation, as shown in Figure 2.8; classical machine learning algorithms usually require several complex feature engineering operations such as feature extraction and dimensionality reduction, following which, the best features will be carefully selected and fed to the algorithm. However, in deep learning, we just need to pass the data to the neural networks directly without any preparation, which completely eliminates the large and challenging feature engineering phase of the entire process.

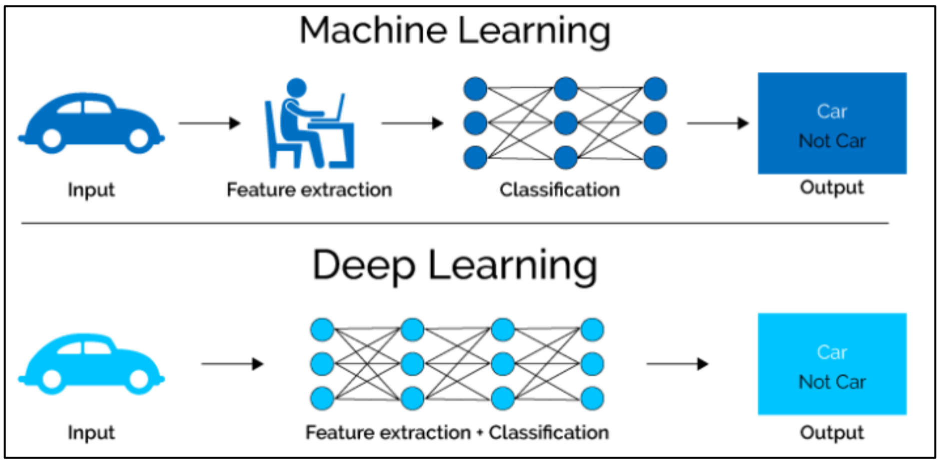


Figure 2.8 *The workflow of machine learning and deep learning (Sambit, 2018)*

2.2.3 Basic Artificial Neural Network

In previous sections, I have introduced the basic concepts about machine learning and deep learning. Now I will focus on how deep learning creates "Neural Networks" that can learn and train to make intelligent decisions for the specific tasks without human intervention. As Michael (2019) mentioned, Neural Network is a biologically inspired programming model, which can help computer learn from observational data; deep learning contains a set of powerful approaches to build and complete Neural Networks.

Figure 2.9 explains how the artificial Neural Network is inspired by the human brain function: each dendrite, cell-body and axon corresponds to the input, information processing unit and output of the Neural Network. The input units receive different structures and forms of information based on the weighting system; thereafter, the Neural Network learns and processes the information by multiplying the inputs and weights of each node. Finally, the Neural Network should be able to present the output based on the aggregated results of all the previous calculations.

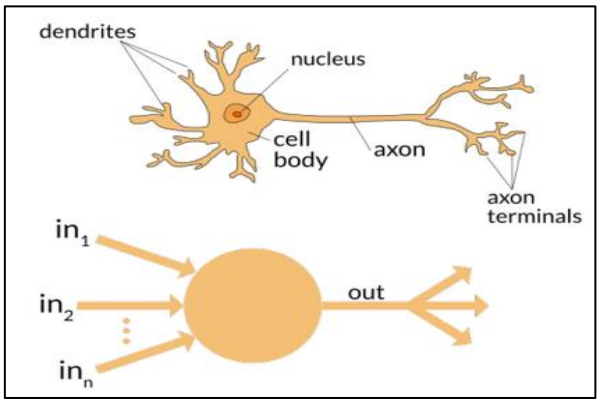


Figure 2.9 *Human brain function and basic neural work structure (Sushan, 2019)*

In general, the number of neurons of the input layer is determined by the dimensionality of the initial data (features, pixels, structures and so on); the number of neurons of the output layer is determined by the goals. Furthermore, the size and number of hidden layers normally depend on the scale and complexity of the data; for instance, if we want to classify the images as number 1 or 0, we need to split the input images into different pixels and pass them as features into the input layer; there will be two neurons of the output layer as 1 and 0.

Figure 2.10 shows the structure of an artificial Neural Network with three hidden layers. The more neurons and hidden layers, the better the training effect and final performance. However, training a complex Neural Network is difficult as the time complexity and computation costs also relatively increase.

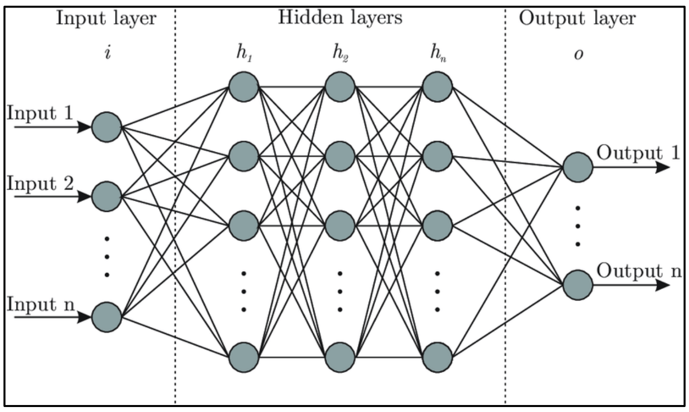
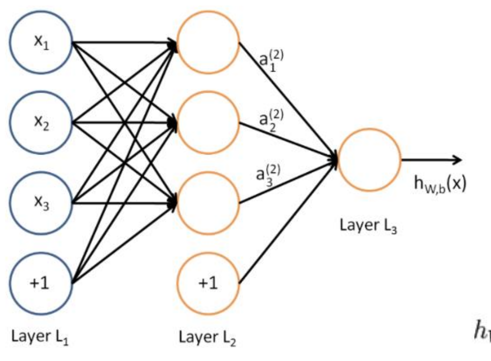


Figure 2.10 *Structure of a complex Neural Network (Facundo and Juan, 2017)*

2.2.3.1 Forward propagation

The main purpose of forward propagation process is to pass the input samples from input to output layer, transmitted by several hidden layers. As shown in figures 2.11A and 2.11B, each input *x1* to *x3* is multiplied by a certain weight *Wij* and the sum of these results is packaged into a function to calculate the hidden node *a1* to *a3*. Finally the value of each hidden node will be added up and fed into the activation function to output the results. Some classical functions such as ReLU, sigmoid and softmax can be used for activation under different conditions.



$$
\begin{aligned}
a_1^{(2)} &= f(W_{11}^{(1)}x_1 + W_{12}^{(1)}x_2 + W_{13}^{(1)}x_3 + b_1^{(1)}) \\
a_2^{(2)} &= f(W_{21}^{(1)}x_1 + W_{22}^{(1)}x_2 + W_{23}^{(1)}x_3 + b_2^{(1)}) \\
a_3^{(2)} &= f(W_{31}^{(1)}x_1 + W_{32}^{(1)}x_2 + W_{33}^{(1)}x_3 + b_3^{(1)}) \\
h_{W,b}(x) = a_1^{(3)} &= f(W_{11}^{(2)}a_1^{(2)} + W_{12}^{(2)}a_2^{(2)} + W_{13}^{(2)}a_3^{(2)} + b_1^{(2)})
\end{aligned}
$$

A **B**

Figure 2.11 *Forward propagation process and corresponding formulas (Yunpeng, 2017)*

2.2.3.2 Back propagation

As Margaret (2015) indicated, back propagation is an important mathematical tool for improving the accuracy of machine/deep learning algorithms. In artificial Neural Network, back propagation is used to update the weight of each node. It calculates the gradient of the error function (Figure 2.12) relative to the weights of the entire artificial Neural Network. The main purpose of back propagation is to minimize the error function. In Figure 2.11, assuming we have calculated the error between the output of Layer *L3* and the desired output, then back propagation would pass this information from layer *L3* back to *L1* by calculating the derivatives of the error function with respect to the weights. Finally, it will update (Figure 2.13) the weights and bias using the gradient descent from input layer and repeat the process of forward propagation until it finds the best fit weights and bias.

$$\boxed{E(X,\theta) = \frac{1}{2N}\sum_{i=1}^{N}\left(\hat{y}_i - y_i\right)^2}$$

Figure 2.12 *Formula of the error function (Mean Square Error function)*

$$\boxed{\begin{array}{c} weights = weights - \Delta weights \\ \hline bias = bias - \Delta bias \end{array}}$$

Figure 2.13 *Update weight and bias using gradient descent method*

2.3 Choice of Methods

2.3.1 Recurrent Neural Networks (RNNs)

The first approach to explore the predictability of worm behaviours is the Recurrent Neural Network (RNN) based on the deep learning theory described in Section 2.2.

As Afshine and Shervine (2019) argued, RNN is one of the classical architectures of Neural Network whose previous outputs can be used as inputs; it contains hidden states, in that the current outputs can be decided by previous outputs as well as the information stored in the hidden states. The implementation of RNN is based on forward propagation and back propagation as mentioned in Section 2.3. Figure 2.14 presents the basic structure of RNN; the inputs of time *t-1* will be stored into the hidden state using certain function, then the outputs of time *t-1* will be passed to time *t*. Finally, it will combine the outputs of time *t-1* and time *t* as inputs for time *t+1* to generate the final results.

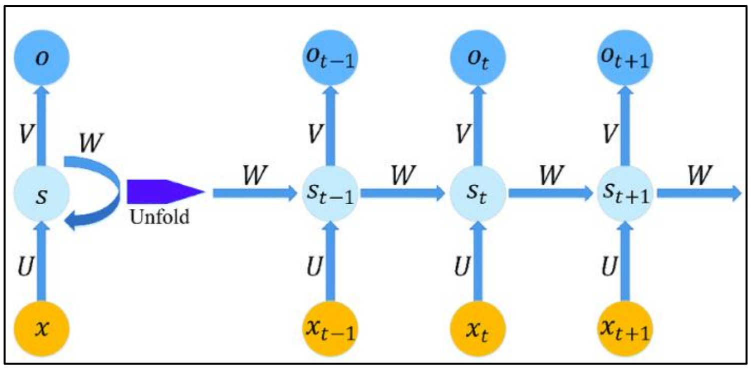


Figure 2.14 *The structure of Recurrent Neural Network (Wei et al., 2017)*

Since the basic idea of this project is to use previous shapes of the worm to predict possible shapes in the future, and our dataset is in the form of sequential data, RNN models will be suitable under this condition. However, although RNN is such a good approach for sequential data forecasting problem, it also has some potential dangers. Educba (2020) listed that the computation speed of RNN is slow; it is also difficult for RNN to process long sequences using traditional activation functions such as “ReLU” and “Tanh”. The most serious problems of RNN are gradient exploding and gradient vanishing because of the usage of back propagation through the network when updating weights; if the output of the activation function is greater than 1, it may cause gradient exploding with long data sequences; otherwise it will cause gradient vanishing.

However, a special type of RNN also exists, called Long Short-Term Memory networks (LSTM), which can solve previous problems efficiently.

2.3.1.1 Long short-term memory Networks (LSTMs)

I decided to use the multi-layer LSTM for the prediction part as the first deep learning approach, which is one of the classical structures of RNNs introduced in previous sections. I choose LSTM for training because it can successfully solve the two problems of traditional RNN: gradient exploding and gradient vanishing with long-term dependence (Sepp and Jurgen, 1997).

Figure 2.15 shows the structure of LSTM with respect to the traditional RNN; it introduces three special gates to store and control the data flow over each cell, known as input gates, forget gates and output gates. Figure 2.16 presents the calculation formula for a certain timestep *t* of LSTM, where f_t is forget gate, i_t is input gate, o_t is the output gate, c_t

represents the state of the cell, h_t is the value stored in the hidden layer, W is weight and b is bias.

Generally speaking, the forget gate will consider the state value of h_{t-1} and current input x_t, then it will drop some values based on the result (0-1) of sigmoid function. Input gates with tanh function determine the hidden layer activation value based on $h_t - 1$ from the previous moment and the current input x_t. Subsequently, the network will drop and store values in cell state c_t. Finally, the output gate will calculate the current hidden value h_t by putting h_{t-1}, x_t and c_t values into the final tanh function.

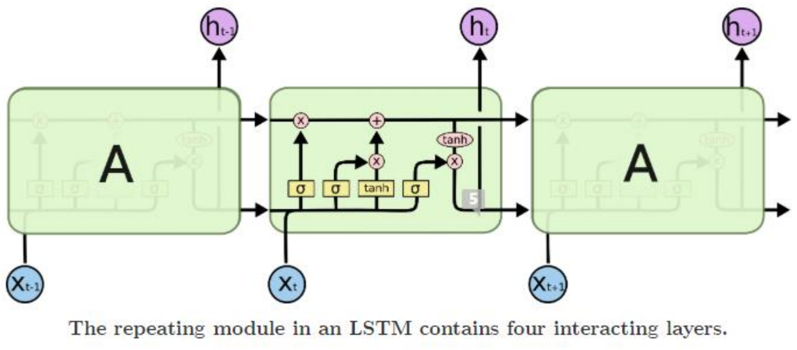

h_{t-1}
h_t
h_{t+1}
A
A
σ
σ
tanh
σ
tanh
X_{t-1}
X_t
X_{t+1}
The repeating module in an LSTM contains four interacting layers.

Figure 2.15 *The structure of LSTM cells (*Sepp and Jurgen, 1997*)*

$$
\boxed{\begin{aligned}
f_t &= \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \\
i_t &= \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \\
\tilde{c}_t &= tanh(W_c \cdot [h_{t-1}, x_t] + b_c) \\
c_t &= f_t * c_{t-1} + i_t * \tilde{c}_t \\
o_t &= \sigma(W_o \cdot [h_{t-1}, x_t] + b_o) \\
h_t &= o_t * tanh(c_t)
\end{aligned}}
$$

Figure 2.16 *Calculation formulas of LSTM cells.*

Briefly, LSTM uses previous gates and operations to control, drop, store and update information, because the final result is a combination of multiple functions and summation operations that traditional RNNs do not have. Therefore, gradient exploding or vanishing will not happen during the process of backpropagation of LSTMs.

2.3.2 Convolution Neural Networks (CNNs)

2.3.2.1 Basic introduction of CNNs

Similar to RNNs, the Convolution Neural Network (CNN) is another classic structure extracted and developed upon traditional Neural Networks. However, RNNs usually modelled for sequential data, while CNNs focus more on 2D matrix data such as images.

According to Renu (2018), CNN is good at image detection and image classification tasks because this architecture was initially inspired by the brain.

Figure 2.17 shows the structure of a fully connected CNN where after receiving the input image, the data will be passed into convolution layers through the calculations of filters, which can divide the picture into smaller patches as feature maps. Subsequently, the feature maps will be passed into convolution layers, and convolution operations could help us to find specific local image features (such as edges) used for further layers. The main function of pooling layers is to keep the main features while reducing the parameters (dimension reduction) and the amount of calculations to prevent the over-fitting problem (Liangziwei, 2019). Finally, data will be putted into flattened layers to convert the pooled feature maps to one column format. Then, features will be passed into Neural Networks as introduced in Section 2.2.3.

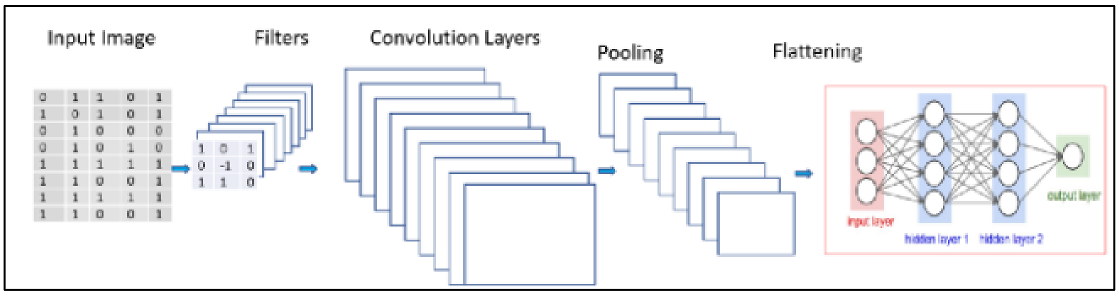


***Figure 2.17** The structure of a fully connected Convolution Neural Networks (Renu, 2018)*

2.3.2.2 Convolution Neural Networks for sequential data

According to Rehan (2013), CNNs can also be applied for sequential data modelling for forecasting and classification purposes. Therefore, I chose CNNs for this project. Figure 2.18 displays the structure of how to process sequential data (time series format) with one-dimension convolution architecture.

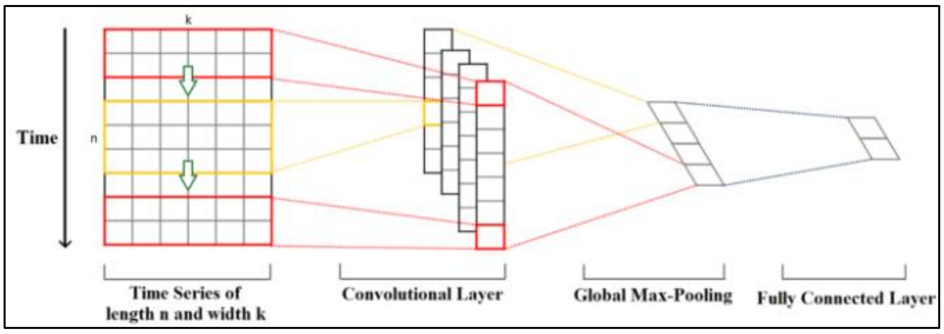


***Figure 2.18** One-dimension Convolution Neural Network structure*

Let us assume the time series with n timesteps and k is the number of variables in the time series. For example, the angles of C. elegans can be regarded as such types of time series and represented by the structure. Following this, the data will be passed into the 1-D convolution layers. Since the width of the convolution kernel is the same as the time series, it will process the convolution operation from the beginning of the time series to its end without another direction. Each point in the time series sequences will be multiplied by the kernel one by one, then the results will be added and calculated by a certain activation function to construct the new filtered time series sequences. The number of filtered sequences and kernels should be equal. Finally, same as traditional 2-D CNNs, pooling layers will extract the largest value for each filtered time series sequence and then pass these values to the fully connected layer for the neural network.

Chapter 3
Design of Experiments

3.1 Requirements

With regards to software requirements, this project will mainly focus on exploring the predictability of worm (C. elegans) behaviours based on several deep learning algorithms. For this project, behaviours of worms (C. elegans) were recorded using electron microscope as videos, each of which segmented as sequences of two-dimensional images. For example, all the mode traces of the dataset for this project were down sampled to 16 fps, implying that 16 two-dimensional frames can be used to represent the behaviours in one second.

For each worm (C. elegans), the software should be able to predict its future behaviours given previous frames.

By learning the relationships between each frame and feature of the frames, the deep learning algorithms should generate some high accuracy prediction results. Not only will this software predict the future shapes of each worm in high dimension, but also the movement trend of each worm in low dimension space during timesteps by applying dimension reduction approaches to summarise the overall dataset into some specific patterns.

Since different deep learning algorithms have different design structures, this project will compare and evaluate the performance of several popular algorithms such as RNNs, CNNs and so on.

In the following sections, I will use the term ‘worm’ to represent ‘C. elegans’ for convenience purposes.

3.2 Experimental Design

3.2.1 Dataset

The dataset of this project contains 12 worms to be analysed, named using

letters from ‘b’ to ‘m’. Each worm was recorded for around 33600 timesteps, initially represented in low dimension by 5 or 6 principal component analysis (PCA) coefficients as footage data. The initial data was stored in Matlab as matrix (33600x5 or 33600x6), where a row is a time point and each column contains the corresponding coefficients with respect to the ‘eigenworm’ basis. Furthermore, another file contains all the ‘eigenworms’ in a 100x100 dimension; the ‘eigenworms’ are stored as 100 ‘angles’ at the equidistrubuted coordinate systems down the body. These two files can be combined to reconstruct the multiple worm postures by transforming PCA coefficients to angles in full dimensional representations (each worm is represented as 100 equally spaced angles along its body).

For different purposes, this project will apply different approaches to process the raw dataset. The specific operations will be discussed in later sections.

3.2.2 Programming language and related tools

The project was mainly achieved using the Google Colaboratory platform based on the Google Drive system, an open research tool for machine/deep learning development and related researches. The greatest benefit of this platform is that it provides free GPU and TPU usage to developers, which can boost the training and testing speed for some complex algorithms. Moreover, it has many built-in packages for Python programming language and some deep learning frames such as ‘Keras’, ‘tensorflow’ and ‘Pytorch’.

The programming language used for this project was Python version 3. I selected Python instead of Java, C or other popular languages for the following reasons:

1. Python is an open-source script programming language supported by multiple operating systems such as Mac OS, Linux and windows. Everyone can use it without spending money to buy software such as Matlab.
2. Python provides many extensions and libraries for different purposes. It is supported by PyPI, which contains over 85,000 modules and libraries for developers to choose (lmseo5hy, 2018).
3. Python has a wide range of application scenarios especially for the artificial intelligence field. It supports many artificial intelligence tools such as machine learning, computer vision, natural language processing and so on.

The following are some important Python3 libraries I used for this project.

1. Keras: An advanced neural network framework written using Python, which supports several backends such as ‘Tensorflow’, ‘CNTK’ and ‘Theano’; can also help to save the experiment time (Keras.io, 2020)
2. Tensorflow: An open-source library which can help to develop and train machine/deep learning models easily via its built-in functions and modules
3. Numpy: A special python extension library used for mathematic computations
4. Pandas: A library of python mainly used for data analysis tasks
5. Matplotlib: A useful python visualisation tool which provides various graphics for data analysis task

6. scikit-learn: A free machine learning library for python, which contains many algorithms for different purposes such as classification, regression and some clustering tasks; also provides several tools for data engineering and preparation related to Numpy or Pandas libraries

3.2.3 High-dimensional shape prediction

For this section, I will use full dimensional representations of worm data to explore its predictability based on deep learning approaches, in that each frame of one worm is represented by 100 corresponding angles/coefficients. As kezhi et al. (2017) mentioned, we can use the skeleton to describe the shape of the worm at each frame (Figure 3.1); each skeleton contains a series points *s(1), s(2) to s(n)*, where *n* is the number of points on the skeleton from the worm's head to its tail; in this dataset, n equals to 100.

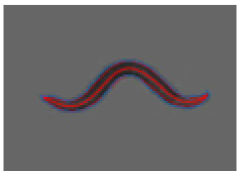


Figure 3.1 *Skeleton of a worm video frame*

In other words, this section will focus on how to use previous full dimensional worm shapes to predict future possible shapes. For example, if there are 50 known frames of worm movement during 50 timesteps—and each frame contains 100 features of the worm—then models should be able to predict the 51th frames or even more frames without any external information. The final format of the data should be (number of timesteps x 100 features) matrices for each worm in the dataset as time-series.

Figure 3.2 presents the overall task design of high-dimensional data prediction; the three images on the left represent the worm frames in different timesteps: the x-axis in each image shows how the shape of each farm is constructed with 100 coefficients.

Now, I will apply several learning approaches to predict future possible movement frames, as well as compare their prediction performance using certain evaluation and visualisation methods.

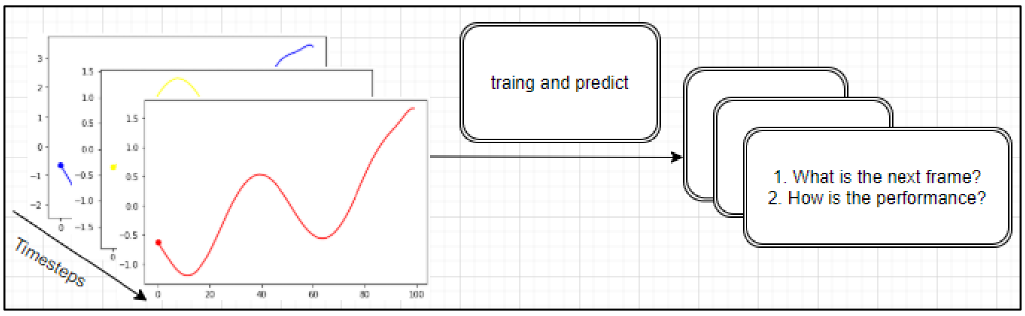


Figure 3.2 *The workflow of prediction worm shapes*

3.2.4 Low-dimensional trend prediction

For this section, I will use low dimensional data to predict the trends of worms' movement using several deep learning approaches. According to Greg et al. (2008), without the stimulation by food, the natural movement of the worm can be represented using only 4-dimensional fundamental shapes (or 'eigenworms'), which can be added together to reconstruct over 95% worm postures with certain proportions. As described in sections 3.2.1 and 3.2.3, we can reconstruct the worm shapes to full dimensional representations as time series with (number of timesteps x 100 features) format, then apply the dimension reduction methods to find the 4-dimensional fundamental shapes. By projecting all the worm shapes onto these 4-dimensional fundamental shapes, the sequence of behaviours can be represented as a 4-channel time series format.

Figure 3.3 (E) shows an example of how dimension reduction can be processed followed by the results. Unlike the information shown in Figure 3.3 (D) using 49 angles to represent the shape of each frame with 3 fundamental shapes, my dataset uses 100 angles to represent the worm posture shapes in each frame.

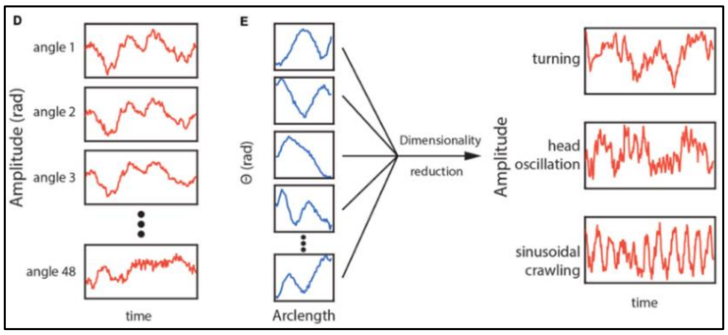


Figure 3.3 *Angle changes along with time and the results after dimension reduction (Bertalan and Andre, 2016)*

After projecting all shapes onto the 4 fundamental shapes, the format of the new time series will be a 4-channel representation. The y-axis represents the amplitude changes of each projected shape with respect to the timesteps. As shown in Figure 3.3, these low dimensional channels may represent the changes of some main movement approaches such as turning, crawling and head oscillation. Since over 95% variance of the worms' behaviours can be explained by the 4 fundamental shapes, the prediction of this 4-channel time series formed by these 4 shapes would be meaningful. Further, I will also analyse each channel of the time series individually which could also help to summarise and understand the overall movement trends of worms. The final format of the data should be (number of timesteps x 4 channels) matrices for each worm in the dataset as time-series.

Moreover, I will extract the movement trends of worms' heads since the head is the most important part of controlling the movement of the whole body. It is also an indicator for evaluating behaviours and nervous systems. Therefore, predicting the movement of worms' heads may help to explore patterns of motion and periodicity (Ryuzo et al., 2013).

Figure 3.4 describes the complete workflow chart for this section. The first task is to predict and analyse the worms' head movements; the second task is to explore the predictability of the 4-channel time after dimensional reduction and amplitude projection operations. The final format of the data should be (number of timesteps x 1 features) matrices for each worm in the dataset as time-series.

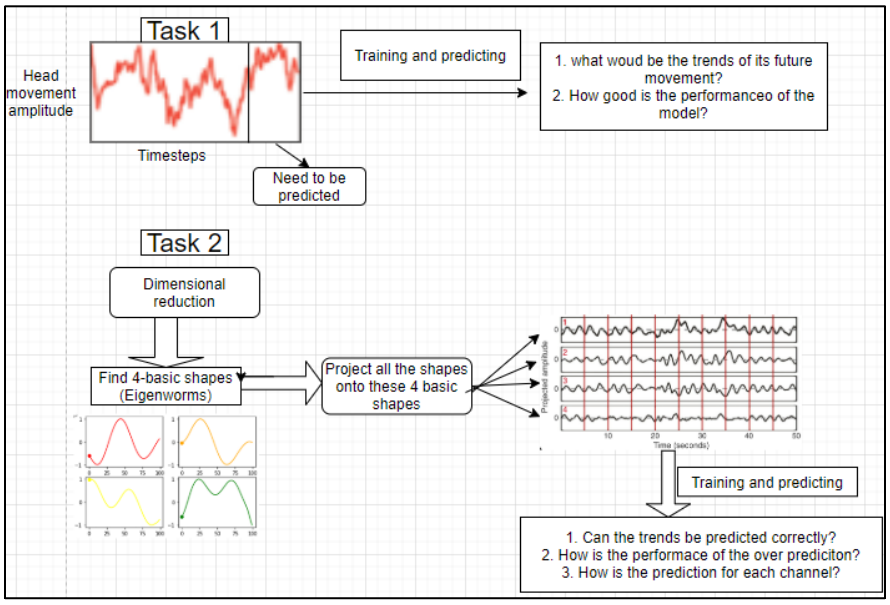

Task 1
Head movement amplitude
Timesteps
Need to be predicted
Training and predicting
1. what woud be the trends of its future movement?
2. How good is the performanceo of the model?
Task 2
Dimensional reduction
Find 4-basic shapes (Eigenworms)
Project all the shapes onto these 4 basic shapes
Projected amplitude
Time (seconds)
Training and predicting
1. Can the trends be predicted correctly?
2. How is the performace of the over prediciton?
3. How is the prediction for each channel?

***Figure 3.4** The workflow of low dimensional trends prediction*

3.2.4.1 Dimension reduction with PCA

As mentioned above, the key part of this section is to apply certain dimensional reduction methods to transform this dataset into a lower dimension. Accordingly, principal component analysis (PCA) is one of the most commonly used methods for large data dimension reduction, a simple technique for extracting information from large-scale and complex datasets. PCA uses orthogonal transformation to linearly transform the observations of a series of possible related variables, thereby projecting the values of a series of linear unrelated variables (also called principal components). It mainly focuses on the maximum variance amount with respect to the number of principal components (Techopedia, 2017).

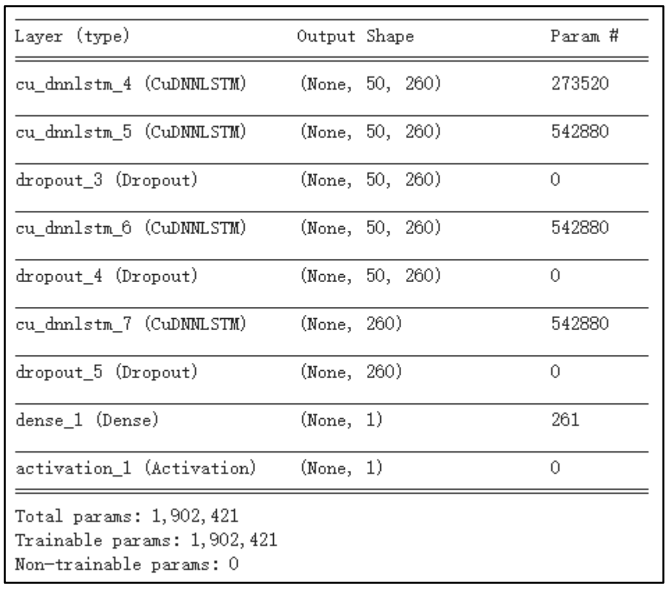
In this project, I will use the PCA approach to find the 4 fundamental shapes that can explain over 95% variance among the whole dataset. Subsequently, I will project all the shapes of each worm onto these 4 fundamental shapes to construct the 4-channel low-dimensional data representations. Generally, the raw data is in full dimensional representations where each timestep contains 100 angles/coefficients of each frame. After the dimension reduction operation, each timestep will only have 4 coefficients. Consequently, not only will the dimension reduction method reduce the size of the data by extracting the most relevant information, but also help to save the training and testing time relatively.

3.2.5 Selection and design of models

I will mainly focus on 4 different deep learning approaches in this section. Since the goal of this project is to explore the predictability of worm behaviours and the datasets are formed as time series representations, the models must have the ability to process predictive tasks. Hence, I have introduced two basic neural network structures called Recurrent Neural Network (RNN) and Convolution Neural Network (CNN) for solving time series prediction problems in Chapter 2. This time, I will design the practical models based on these two network structures for both high and low dimensional prediction problems and compare their predictive performance. Besides, I have also designed two more architectures known as CNN-LSTM model and Bidirectional LSTM model for this project, where the CNN-LSTM model will be applied to solve high-dimensional shape prediction tasks and the Bidirectional LSTM model to solve two low-dimensional trend prediction tasks.

3.2.5.1 The LSTM model

The first model I selected was LSTM. I have introduced the basic implementation theory behind this special type of RNN in Chapter 2. The design idea of this model was extracted from kezhi et al. (2017), who prepared a 4-layer LSTM network structure using the sequential model provided by the Python library Keras, which can be effectively used to predict worm behaviours with great performance. For this project, I will implement the LSTM model using Keras library. Figure 3.5 shows the structure of the LSTM networks I re-designed for this project. For the sake of boosting the training speed and improving training efficiency, I used ‘CuDNNLSTM’ library with 260 hidden units as the LSTM layer instead of the normal ‘keras.LSTM’ library because ‘CuDNNLSTM’ supports the usage of GPU calculation power; 260 hidden units means that after each LSTM layer, the output shape will become (none, timesteps, 260); the more hidden units, the better the robustness of the network. However, the number of parameters in ‘CuDNNLSTM’ is less than ‘Keras.LSTM’. Therefore, I artificially added several hyper-parameters into the network. After each 2nd, 3rd and 4th LSTM layer, I added one ‘dropout’ layer with the dropout rate 0.2. Finally, the ‘dense’ layer took all the results of previous layers and generated the training results. I used the ‘linear’ function as the activation function; the ‘mean square error’ function was used to evaluate the training performance while the ‘rmsprop’ function was applied as the optimizer for the training process.

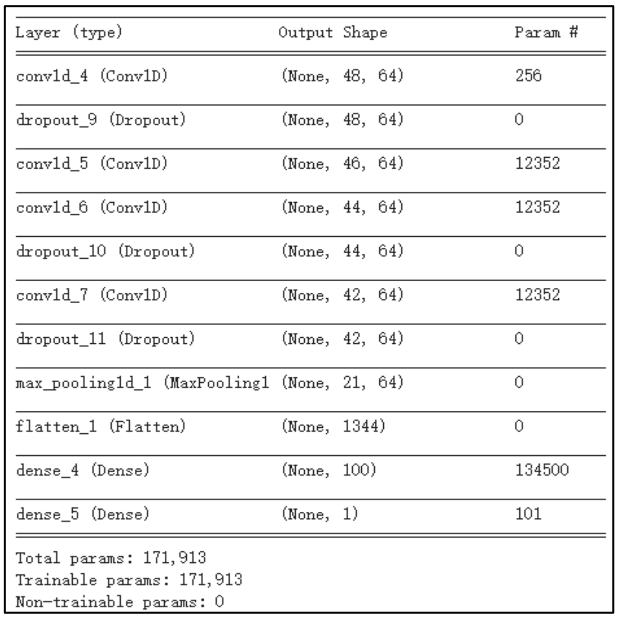


```
Layer (type)                 Output Shape              Param #
================================================================
cu_dnnlstm_4 (CuDNNLSTM)     (None, 50, 260)           273520
________________________________________________________________
cu_dnnlstm_5 (CuDNNLSTM)     (None, 50, 260)           542880
________________________________________________________________
dropout_3 (Dropout)          (None, 50, 260)           0
________________________________________________________________
cu_dnnlstm_6 (CuDNNLSTM)     (None, 50, 260)           542880
________________________________________________________________
dropout_4 (Dropout)          (None, 50, 260)           0
________________________________________________________________
cu_dnnlstm_7 (CuDNNLSTM)     (None, 260)               542880
________________________________________________________________
dropout_5 (Dropout)          (None, 260)               0
________________________________________________________________
dense_1 (Dense)              (None, 1)                 261
________________________________________________________________
activation_1 (Activation)    (None, 1)                 0
================================================================
Total params: 1,902,421
Trainable params: 1,902,421
Non-trainable params: 0
```

Figure 3.5 *Specific structure of LSTM model with parameters*

3.2.5.2 The CNN model

Figure 3.6 shows the structure of the CNN I designed for this project, which contains 4 ‘Keras.Conv1d()’ layers and each layer takes 64 filters and 3 kernels; 64 filters mean that there are 64 different windows in this model; 3 kernels means that each window has the length equal to 3; each ‘Conv1d’ layer will output 64 different convolution units. After each 1st, 3rd and 4th convolutional layer, I added one ‘dropout’ layer with a 0.2 dropout rate. The ‘MaxPooling(2)’ was added to maintain the consistency and distil the feature maps down to half the size to obtain the most salient features (Jason, 2018). Finally, the ‘Flatten()’ layer will take the pooling results and reduce the high dimension into one dimension. The activation in each convolution layer is called ‘Relu’. ‘Mean square error’ was applied to evaluate each training result. Same as the LSTM model, I used the ‘rmsprop’ function as the optimizer.



```
Layer (type)                 Output Shape              Param #
================================================================
conv1d_4 (Conv1D)            (None, 48, 64)            256
________________________________________________________________
dropout_9 (Dropout)          (None, 48, 64)            0
________________________________________________________________
conv1d_5 (Conv1D)            (None, 46, 64)            12352
________________________________________________________________
conv1d_6 (Conv1D)            (None, 44, 64)            12352
________________________________________________________________
dropout_10 (Dropout)         (None, 44, 64)            0
________________________________________________________________
conv1d_7 (Conv1D)            (None, 42, 64)            12352
________________________________________________________________
dropout_11 (Dropout)         (None, 42, 64)            0
________________________________________________________________
max_pooling1d_1 (MaxPooling1 (None, 21, 64)            0
________________________________________________________________
flatten_1 (Flatten)          (None, 1344)              0
________________________________________________________________
dense_4 (Dense)              (None, 100)               134500
________________________________________________________________
dense_5 (Dense)              (None, 1)                 101
================================================================
Total params: 171,913
Trainable params: 171,913
Non-trainable params: 0
```

Figure 3.6 *Specific structure of CNN model with parameters*

3.2.5.3 The CNN-LSTM model

The hybrid CNN-LSTM model was designed especially for high-dimensional shape prediction tasks. The advantage of this approach is that it supports long input sequences with high dimension, because this model can divide such sequences into several sub-sequences or blocks. Figure 3.7 presents the basic architecture of the CNN-LSTM network, where the CNN model is responsible for leaning features from the input sequences, then the learned sequences are passed onto the LSTM as inputs.

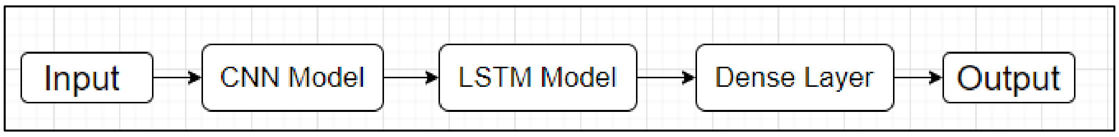
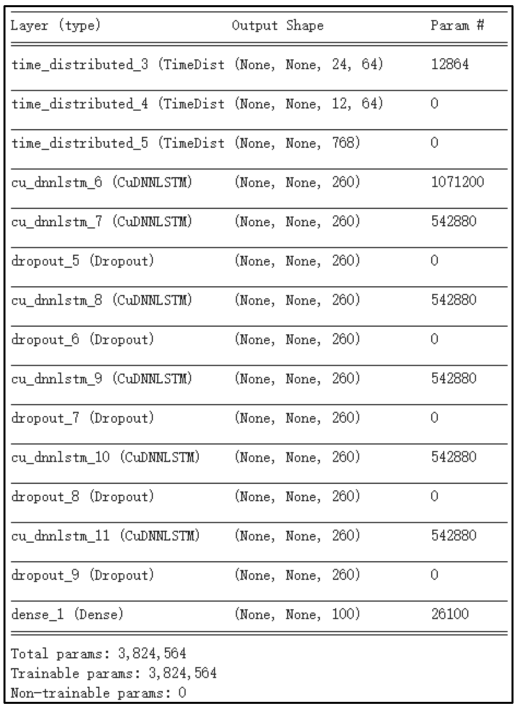


Figure 3.7 *Workflow of CNN-LSTM model*

Figure 3.8 is the CNN-LSTM network I designed for this project, since the dimension of shape prediction task is high; the entire CNN part (including the convolution, max pooling and the flatten layers) is wrapped in the 'Keras.TimeDistributed()' layer, which achieves the desired outcome of training the same layer many times (Jason, 2017). This time, I specified 6 'CuDNNLSTM' layers with 4 'dropout' layers; 260 hidden units in each 'CuDNNLSTM' layer are the same as the LSTM model introduced in Section 3.2.5.1. Finally, the training steps and operations were same as the normal LSTM model.

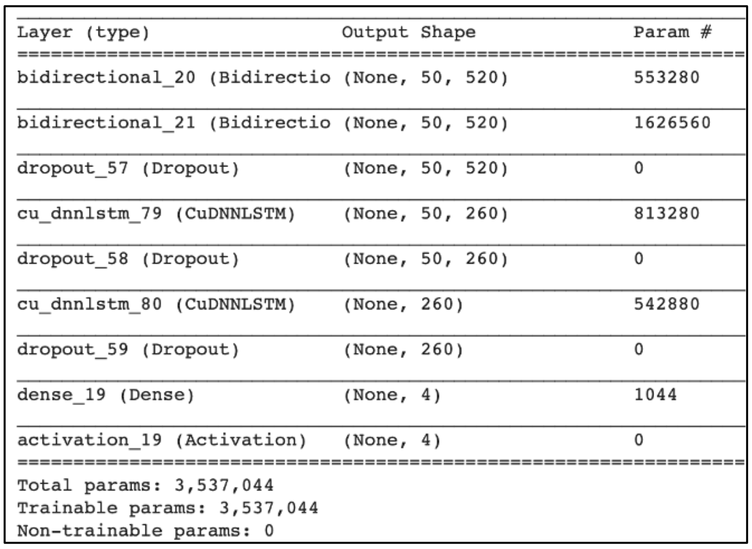


```
Layer (type)                 Output Shape              Param #
=================================================================
time_distributed_3 (TimeDist (None, None, 24, 64)      12864
_________________________________________________________________
time_distributed_4 (TimeDist (None, None, 12, 64)      0
_________________________________________________________________
time_distributed_5 (TimeDist (None, None, 768)         0
_________________________________________________________________
cu_dnnlstm_6 (CuDNNLSTM)     (None, None, 260)         1071200
_________________________________________________________________
cu_dnnlstm_7 (CuDNNLSTM)     (None, None, 260)         542880
_________________________________________________________________
dropout_5 (Dropout)          (None, None, 260)         0
_________________________________________________________________
cu_dnnlstm_8 (CuDNNLSTM)     (None, None, 260)         542880
_________________________________________________________________
dropout_6 (Dropout)          (None, None, 260)         0
_________________________________________________________________
cu_dnnlstm_9 (CuDNNLSTM)     (None, None, 260)         542880
_________________________________________________________________
dropout_7 (Dropout)          (None, None, 260)         0
_________________________________________________________________
cu_dnnlstm_10 (CuDNNLSTM)    (None, None, 260)         542880
_________________________________________________________________
dropout_8 (Dropout)          (None, None, 260)         0
_________________________________________________________________
cu_dnnlstm_11 (CuDNNLSTM)    (None, None, 260)         542880
_________________________________________________________________
dropout_9 (Dropout)          (None, None, 260)         0
_________________________________________________________________
dense_1 (Dense)              (None, None, 100)         26100
=================================================================
Total params: 3,824,564
Trainable params: 3,824,564
Non-trainable params: 0
```

Figure 3.8 *Specific structure of CNN-LSTM model with parameters*

3.2.5.4 The Bidirectional LSTM model

The Bidirectional LSTM network is a special and improved architecture of the normal LSTM network, a combination of forward LSTM and backward LSTM. I designed this model for the low-dimensional trend prediction task because this model can learn from both past to future and future to past, which will provide a better understanding for the model during the training stage. In normal cases, Bidirectional LSTM should have a better performance than the normal LSTM model, especially when processing sequential data such as text classification or speech recognition. Figure 3.9 shows the structure of the Bidirectional LSTM; the only difference between this and the LSTM model introduced in Section 3.2.5.1 is that I used two 'Bidirectional()' layers to wrap the 1st and 2nd 'CuDNNLSTM' layers. We can find that the output shape of the first two layers becomes (None, timestep, 520) rather than (None, timestep, 260) as before.



```
Layer (type)                 Output Shape              Param #
=================================================================
bidirectional_20 (Bidirectio (None, 50, 520)           553280
_________________________________________________________________
bidirectional_21 (Bidirectio (None, 50, 520)           1626560
_________________________________________________________________
dropout_57 (Dropout)         (None, 50, 520)           0
_________________________________________________________________
cu_dnnlstm_79 (CuDNNLSTM)    (None, 50, 260)           813280
_________________________________________________________________
dropout_58 (Dropout)         (None, 50, 260)           0
_________________________________________________________________
cu_dnnlstm_80 (CuDNNLSTM)    (None, 260)               542880
_________________________________________________________________
dropout_59 (Dropout)         (None, 260)               0
_________________________________________________________________
dense_19 (Dense)             (None, 4)                 1044
_________________________________________________________________
activation_19 (Activation)   (None, 4)                 0
=================================================================
Total params: 3,537,044
Trainable params: 3,537,044
Non-trainable params: 0
```

Figure 3.9 *Specific structure of the Bidirectional LSTM model with parameters*

3.2.5.5 Parameters explanations

In this section, I will discuss the selection and design of some special algorithms, layers, functions, or hyperparameters used in this project which may benefit the overall design of the network structures.

1. Dropout Layer: As mentioned in previous sections and figures, I have added several Dropout layers in each model, the main goal of using Dropout layer is to prevent the over-fitting problem by discarding some neural network unites during training process according to a certain probability. Figure 3.10 presents the example of how Dropout layer affect the training of one neural network.

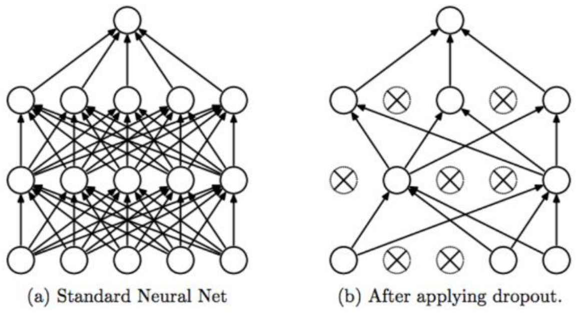


Figure 3.10 *standard neural network and the influence after using Dropout (Cory, 2019)*

2. Optimizer: According to Sanket (2019), optimizer is a kind of algorithms in deep learning that can be applied to change the attributes of the neural network especially for reducing the losses when networks updating weights and learning rates. I applied “RMSprop” optimizer for each model, because the main target is to make prediction, ‘RMSprop’ optimizer can help to maintain the consistency and reduce the likelihood

of gradient vanishing and exploding when training deep predictive models such as RNN.

3.2.6 Evaluation methods

In this section, I will describe some methods and metrics that I used to measure the performance of each predictive model for both high-dimensional and low-dimensional prediction tasks.

Initially, since this project focused on exploring the predictability of worm behaviours, but we have no way to know what will happen in the future without any existing reference information. I have divided the dataset into training set and testing set, where the training set can be used to fit and train the predictive network models, testing set can be used to evaluate the performance of each model.

Since I have 33600 timesteps for each worm data which is large enough for this project, so that I decided to divide the dataset consists of 70% data for training and 30% data for testing. When using the test dataset to evaluate the performance of a trained predictive model, it will generate new predicted values based on the training results. However, because we have already known the real data in the test dataset, we can simply compare the real data and the predicted data to evaluate the quality of the trained model.

To compare the differences between the real data and the predicted data, I have designed two performance metrics to evaluate and represent the differences numerically.

1. Euclidean Distance: Euclidean Distance is one of the most commonly used distance measurement approaches, which will generate the true distance between two points or two vectors in a certain dimensional space. In 2-D and 3-D dimensional spaces, Euclidean Distance just represents the real distance between two points (Wikipedia). Figure 3.11 is the numerical formula for calculating Euclidean distance. In a word, the smaller the Euclidean distance, the closer the real value and the predicted value are, then the model has the better performance based on the test set.

$$\boxed{d = \sqrt{\sum_{i=1}^{N} (x_{1i} - x_{2i})^2}}$$

Figure 3.11 *Formula of Euclidean Distance*

2. Mean Square Error: The definition of mean square error is the average of the squared measurement error. Generally speaking, it will calculate the value of the mean squared difference between the estimated value and the real value (Wikipedia). Figure 3.12 shows the formula for calculating mean square error numerically.

$$MSE = \frac{1}{n}\Sigma\left(\underbrace{y - \widehat{y}}_{\text{The square of the difference between actual and predicted}}\right)^2$$

Figure 3.12 *Formula of Mean Square error (i2tutorials, 2019)*

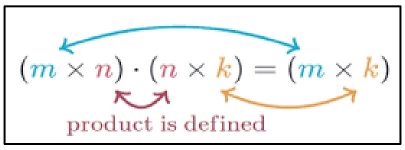
Chapter 4
Implementation and Results

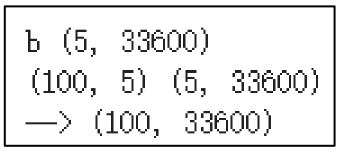
In this Chapter, I will implement the prediction tasks described in Chapter 3 and evaluate the performance of each model. As mentioned, for high-dimensional shape prediction problem, I will test 3 models called LSTM, CNN and CNN-LSTM to achieve the initial goal; for low-dimensional trend prediction problem, I will test 3 models called LSTM, CNN and Bidirectional LSTM to achieve the initial goal and present the prediction results in some meaningful ways. Moreover, I have specified the timesteps as 50 for all the prediction models, which means each model will use 50 previous timesteps to predict the next timesteps both in training and testing processes.

4.1 Data preparation

Since the initial datasets were stored in '.mat' file with binary data format, which can only be read and write using Matlab, so the first thing was to read such data, load these Matlab files from the original experiment and parses them into Python readable format such as Numpy formats. To achieve this, I used the package 'scipy.io' to read the initial Matlab file and 'h5py' to transform the binary data into the normal matrix format data. Finally, each worm data was presented using Numpy as (5 x 33600) or (6 x 33600) format matrices. To reconstruct the postures from basic coefficients to fully represented angles, I was given the codes to reshape the initial (100x100) format reference file as (100 x 5) and (100 x 6) formats corresponding to the format of each worm data. Then I applied matrix multiplication theory to construct the final data format as (100 x 33600).

Figure 4.1 (A) presents the rule of matrix multiplication theory, while Figure 4.1 (B) presents the final format of the worm (e.g. worm 'b').




```
b (5, 33600)
(100, 5) (5, 33600)
—> (100, 33600)
```
s

A (Khanacademy, 2017) B

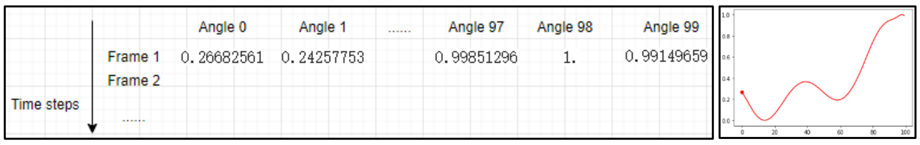
Figure 4.1 *The process of constructing the fully represented shapes of data*

After reconstructing the shape of each worm, I used ‘Pandas.DataFrame’ to store the (100x33600) matrix for each worm. Because there existed some columns with empty entries, to ensure the quality of this prediction process, I have dropped all the columns which contain empty entries. Finally, I used ‘Sklearn.MinMaxScaler’ library to normalise all the angles between 0 and 1 to remove outliers and maintain the robustness for the entire dataset.

Although there are overall 12 different worms described in the raw dataset, there is no exact information about worm ‘g’ and worm ‘i’ because they have no angles change as the number of timesteps increases. In the following sections, I will only focus on the remaining 10 worms without worm ‘g’ and worm ‘i’.

4.2 High-dimensional shape prediction

As introduced in Chapter 3, I will use a number of previous frames to predict the next one frames in the dataset, each frame is made using 100 angles (or features in machine learning), so that both the input vectors and output vectors should have 100 features. Figure 4.2 (A) shows the vector representation of each frame (e.g. Frame 1) while Figure 4.2 (B) displays the image made by these vectors.



Time steps		Angle 0	Angle 1	......	Angle 97	Angle 98	Angle 99
	Frame 1	0.26682561	0.24257753		0.99851296	1.	0.99149659
	Frame 2						
	......						

A B

Figure 4.2 *(A) Example of the first frame represented in vectors (B) Example of the first frame represented using image (Plotted by Matplotlib)*

Before the implementation, the dataset for each worm should be divided into 70% training set and 30% test set, the training set consists of two parts called ‘X_train’ and ‘y_train’, where ‘X_train’ is a 3-D matrix with shape (num_samples, 50 timesteps, 100 feature for training), ‘y_train’ is a 2-D matrix with the shape (num_sampels, 100 features to be predicted).

To evaluate the performance of each model, I will calculate the average values of mean square error and Euclidean distance between each predicted value and real value. Moreover, I will also predict the 25th, 50th and 75th percentile frames in the test dataset for each worm, then using data visualisation approach to compare the differences between the predict frames and the real one for different models.

4.2.1 The CNN model and LSTM model

In this section, I will combine the results of CNN model and LSTM model to presents a significant comparation between these models. Each model was trained for 200 epochs for 10 worms. Table 4.1 shows the average Mean Square Error (Average MSE) and the average Euclidean Distance (Average ED) of CNN model and RNN model for all 10 worms over the whole test dataset.

	CNN model		**LSTM model**	
	Average MSE	**Average ED**	**Average MSE**	**Average ED**
worm b	0.0065	0.77	0.00026	0.12
worm c	0.0049	0.67	0.00012	0.10
worm d	0.0092	0.93	0.00018	0.12
worm e	0.0053	0.69	0.00028	0.15
worm f	0.0072	0.84	0.00019	0.13
worm h	0.0073	0.83	0.00018	0.11s
worm j	0.0058	0.73	0.00029	0.16
worm k	0.0065	0.77	0.00030	0.15
worm l	0.0062	0.75	0.00086	0.16
worm m	0.0064	0.77	0.00019	0.12

Table 4.1 *Average MSE and Average ED of CNN and LSTM model for 10 worms*

Table 4.2 presents the differences between real data, CNN predicted data and LSTM predicted data in image representations using three samples in the test dataset, while these three samples were selected based on the quantile (Q1, Q2, Q3) of the whole test dataset (Here I will only show the images for the first 5 worms, the images for the remaining 5 worms can be checked in **Appendix D**).

	CNN model	**LSTM model**

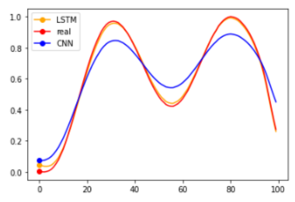
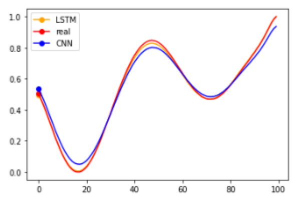
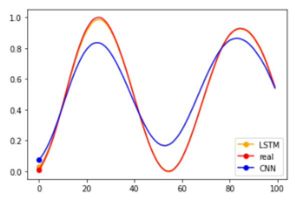
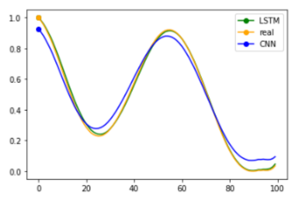
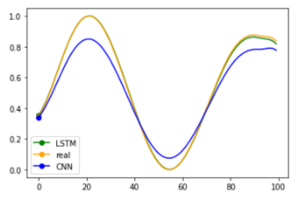
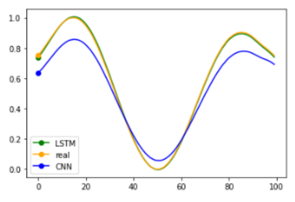
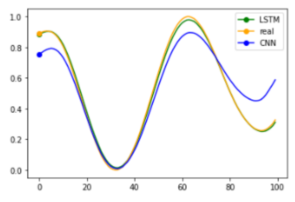
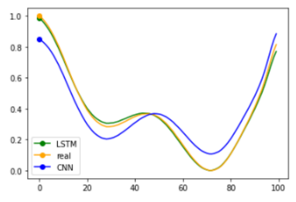
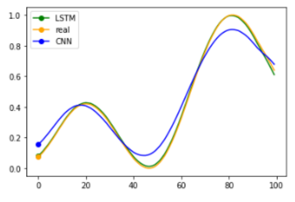
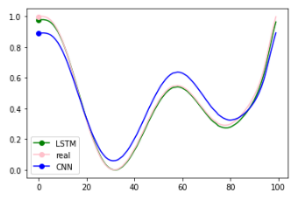
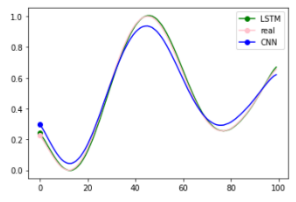
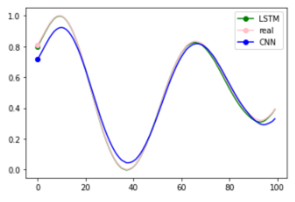
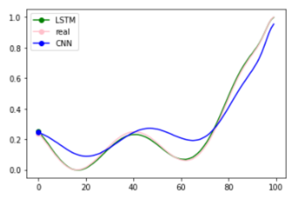
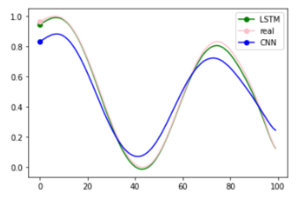
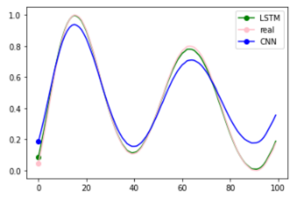
	Quartile 1 (25th %)	Quartile 2 (50th %)	Quartile3 (75th %)
worm b			
worm c			
worm d			
worm e			
worm f			

***Table 4.2** Image representations of three worm shapes selected from test dataset (CNN model and LSTM model)*

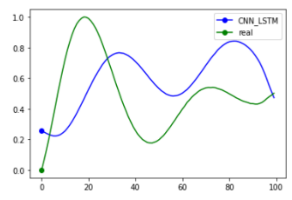
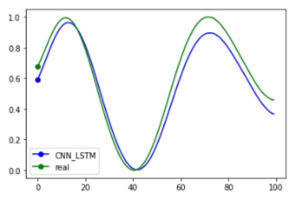
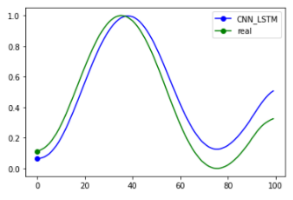
4.2.2 The CNN-LSTM model

In this section, I will train and evaluate the CNN-LSTM model. For each worm I will train 200 epochs to maintain the training consistency as CNN model and LSTM model mentioned before. Table 4.3 shows the average Mean Square Error (Average MSE) and the average Euclidean Distance (Average ED) of CNN model and RNN model for all 10 worms over the whole test dataset.

	CNN-LSTM model	
	Average MSE	**Average ED**
Worm b	0.056	2.16
Worm c	0.030	1.56
Worm d	0.044	1.93
Worm e	0.041	1.81
Worm f	0.035	1.72
Worm h	0.036	1.57
Worm j	0.042	1.87
Worm k	0.049	2.01
Worm l	0.044	1.89
Worm m	0.043	1.88

Table 4.3 *Average MSE and Average ED of CNN-LSTM model for 10 worms*

Table 4.4 presents the differences between real data and CNN-LSTM model predicted data in image representations using three samples in the test dataset, while these three samples were selected based on the quantile (Q1, Q2, Q3). Same as CNN model and LSTM model, I will just present the images for the first 5 worms, remaining images can be found in **Appendix E**.

	CNN-LSTM model		
	Quartile 1 (25th %)	**Quartile 2 (50th %)**	**Quartile3 (75th %)**
Worm b			

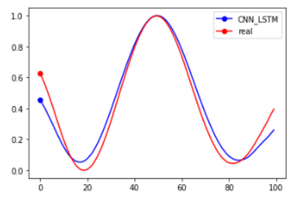
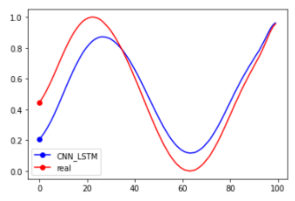
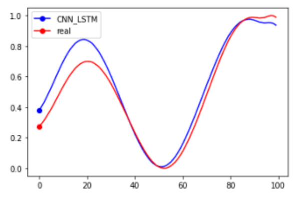
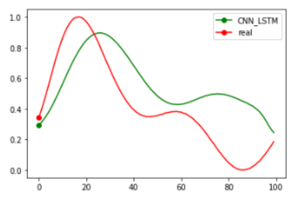
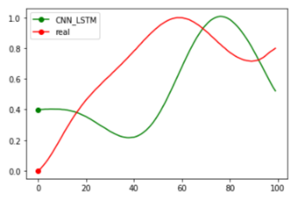
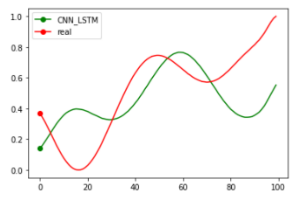
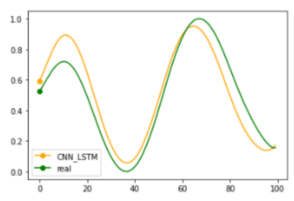
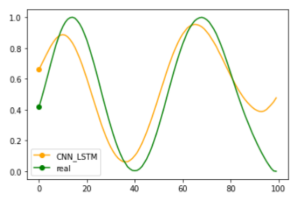
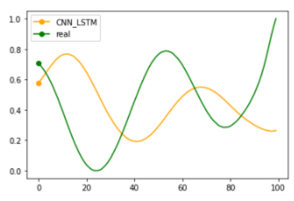
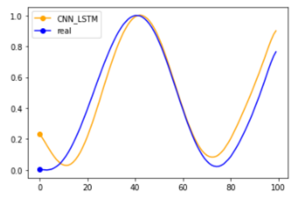
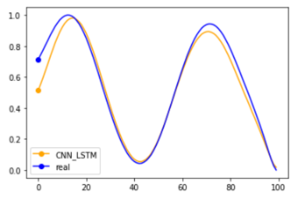
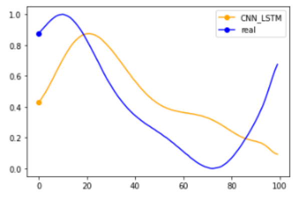
Worm c			
Worm d			
Worm e			
Worm f			

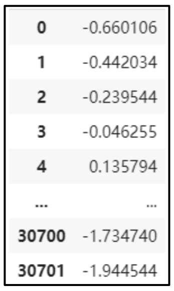
Table 4.4 *Image representations of three worm shapes selected from test dataset (CNN-LSTM model)*

4.3 Low-dimensional trend prediction

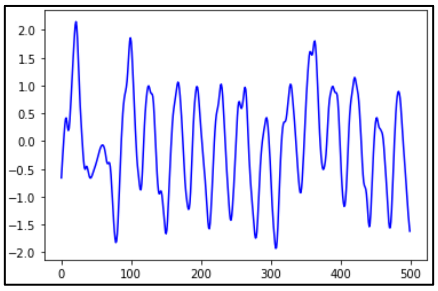
As introduced in chapter 3, I have divided this section into two parts to build the model and evaluate the performance. The first task is to capture the movement trend of worms' head, models will use the previous motion patterns to learn the features and train themselves, then they should be able to predict some future outcomes. The second task requires data dimension reduction operation to find four basic shapes, then the time series dataset will be constructed by projecting all the shapes onto these four basic shapes.

4.3.1 Task 1: Head movement trends prediction

This time I will mainly focus on the movement patterns of each worm's head, since I have already prepared the dataset in section **4.1**, this time I just need extract the first point from each frame instead of 100 points for shape prediction. Figure 4.3 (A) shows an example of head data in each frame represented using 1-dimensional vector, Figure 4.3 (B) displays the first 500 points of this 1-dimensional vector in image representations.



0	-0.660106
1	-0.442034
2	-0.239544
3	-0.046255
4	0.135794
...	...
30700	-1.734740
30701	-1.944544



A B

Figure 4.3 *(A) Head movement data represented using numerical vector (B) The movement of worm's head within first 500 timesteps*

Same as shape prediction, the head movement data of each worm should be divided into 70% training dataset and 30% testing dataset. Since the data is in 1-D dimensional space, which means that it only contains one feature, so this time 'X_train' will become (num_samples, 50 timesteps, 1 feature for training), 'y_train' will be (num_samples, 1 feature to be predicted).

To evaluate the performance, I will calculate the average mean square error (MSE) and Euclidean distance (ED) between the real value points and the predicted value points predicted by CNN model and LSTM model. I will also visualise the first 1000 timesteps for predicted data and real data.

4.3.1.1 The CNN model and LSTM model

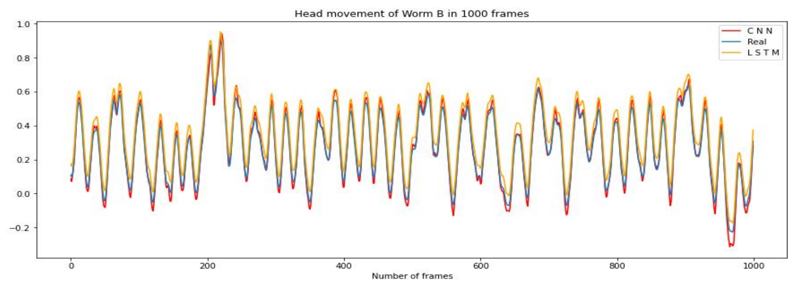
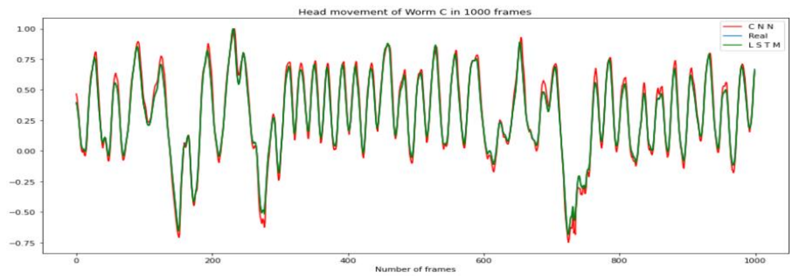
In this section, I will combine the results of CNN model and LSTM model to presents a significant comparation between these models. Each model was trained for 180 epochs for 10 worms. Table 4.5 shows the average Mean Square Error (Average MSE) and the average Euclidean Distance (Average ED) of CNN model and RNN model for all 10 worms over the whole test dataset.

	CNN model		**LSTM model**	
	Average MSE	**Average ED**	**Average MSE**	**Average ED**
Worm b	0.0010	0.023	0.0044	0.066
Worm c	0.0025	0.042	0.0005	0.006
Worm d	0.0107	0.096	0.0006	0.024

Worm e	0.0033	0.047	0.0002	0.010
Worm f	0.0026	0.045	0.0002	0.013
Worm h	0.0030	0.050	0.0006	0.023
Worm j	0.0017	0.036	0.0011	0.033
Worm k	0.0028	0.046	0.0026	0.051
Worm l	0.0111	0.101	0.0007	0.024
Worm m	0.0022	0.040	0.0003	0.018

Table 4.5 *Average MSE and Average ED of CNN and LSTM model for 10 worms (head trend prediction)*

Table 4.6 shows the visualisation results for the first 1000 frames of head movement in the test dataset for each worm, which contains the real head movement line, CNN predicted movement line and LSTM predicted movement line. Still, here I will only show the result for the first 5 worms, remaining parts can be found in **Appendix F**.

	CNN model	**LSTM model**
Worm b		
Worm c		

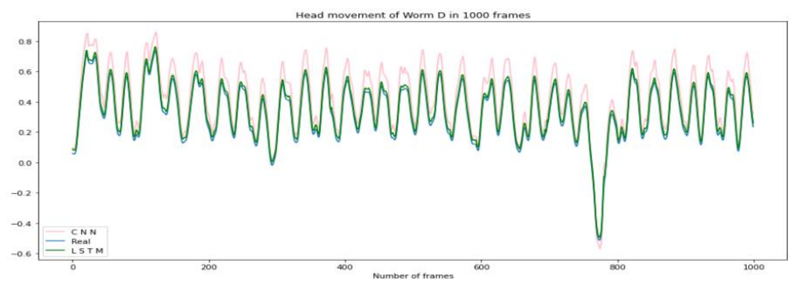
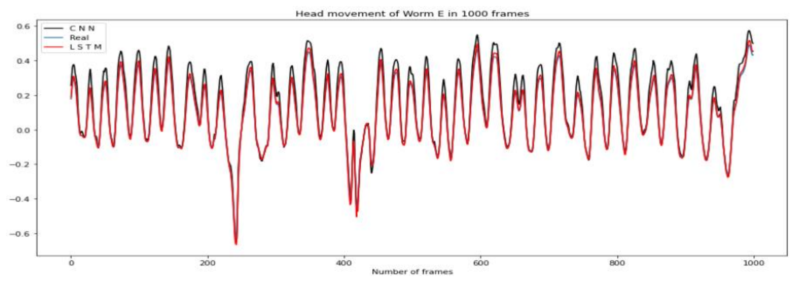
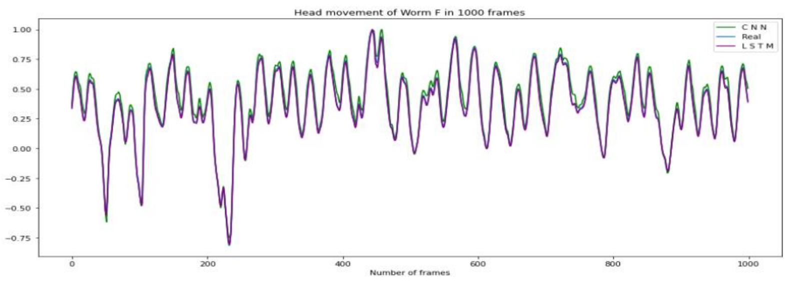
Worm d	
Worm e	
Worm f	

Table *4.6 Head movement prediction using CNN model and LSTM model (1000 timesteps)*

4.3.1.2 The Bidirectional LSTM model

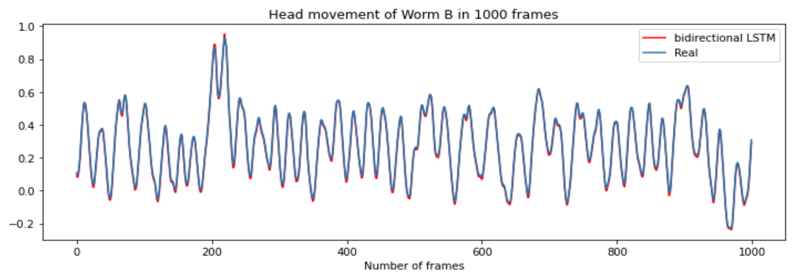
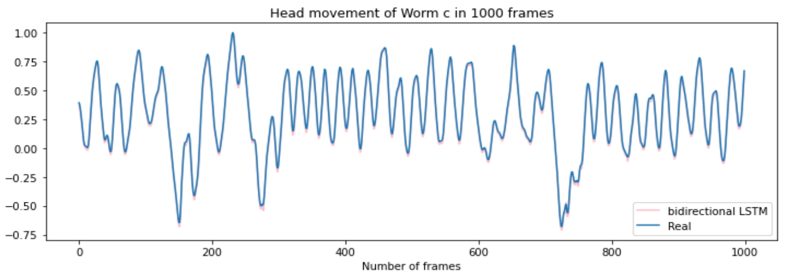
For the bidirectional LSTM model, I will keep the same training epochs as CNN model and LSTM model described in previous sections. Not only will I present the results of the average mean square error (MSE) and Euclidean distance (ED) in table 4.7, but I will also visualise the results for the first 1000 frames in test dataset for each worm.

	Bidirectional LSTM model	
	Average MSE	**Average ED**
Worm b	0.0002	0.011
Worm c	0.0005	0.022
Worm d	0.0003	0.018
Worm e	0.0017	0.040

Worm f	0.0006	0.023
Worm h	0.0002	0.012
Worm j	0.0004	0.020
Worm k	0.0009	0.030
Worm l	0.0003	0.014
Worm m	0.0011	0.032

Table 4.7 *Average MSE and Average ED of bidirectional LSTM model for 10 worms (head trend prediction)*

Table 4.8 shows the visualisation results for the first 1000 frames of head movement in the test dataset for each worm, which contains the real head movement line and the Bidirectional LSTM predicted results. I will just show the result for the first 5 worms, remaining parts can be found in **Appendix G**.

	Bidirectional LSTM model
Worm b	
Worm c	

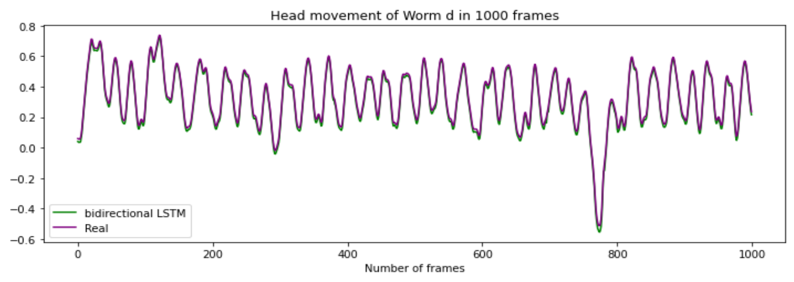
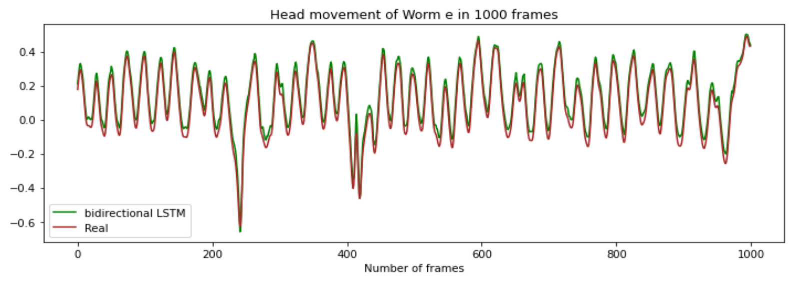
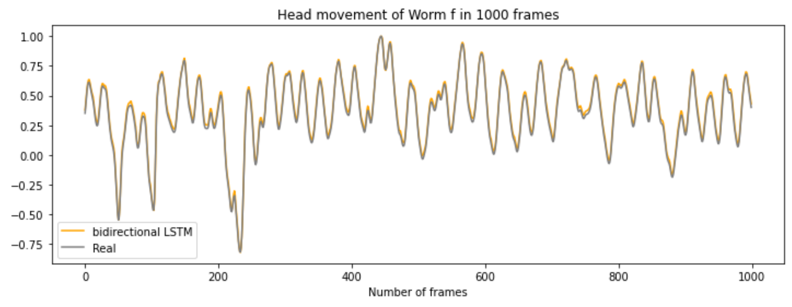
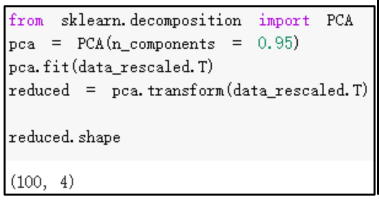
Worm d	
Worm e	
Worm f	

***Table** 4.8 Head movement prediction using Bidirectional LSTM model (1000 timesteps)*

4.3.2 Task 2: Basic shapes projection trends prediction

4.3.2.1 Dimension reduction with PCA

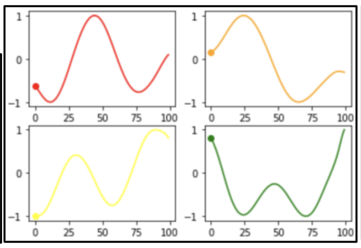
Since the requirement of task 2 is to predict the trends of the projected amplitudes formed by the 4 basic shapes described in Chapter 3, so that the first step is to apply PCA to find the 4 basic shapes. As for worm ‘b’, I applied the python library called ‘Sklearn.decomposition’ to implement the PCA dimension reduction operation for the whole (100 x 33600) dataset, then Figure 4.4 (A) shows the codes I used to implement the operation, the result is a (100, 4) matrix. Figure 4.4 (B) displays the 4 basic shapes returned by PCA, which can be used to reconstruct all the shapes of worm ‘b’ because these shapes can explain over 95% variance among the whole dataset as shown in Figure 4.4 (C).

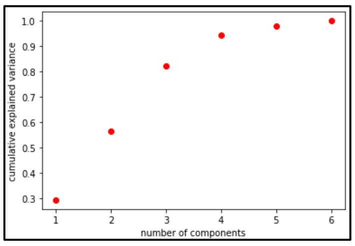


```
from  sklearn.decomposition  import  PCA
pca  =  PCA(n_components  =  0.95)
pca.fit(data_rescaled.T)
reduced  =  pca.transform(data_rescaled.T)

reduced.shape

(100,  4)
```

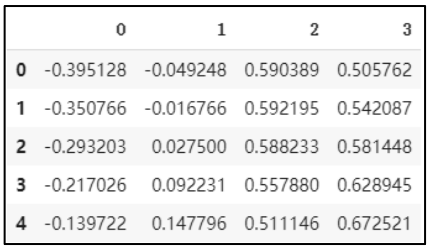




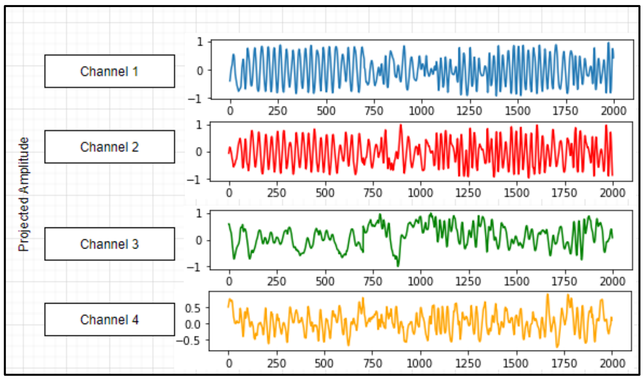
A B C

Figure 4.4 *the implementation (A), returned shapes (B) and Variance explanation (C) of PCA dimension reduction (Worm 'b')*

After finding the 4 basic shapes, the time series dataset can be built just by projecting all the shapes in the dataset onto these 4 shapes, Figure 4.5 (A) presents an example of the final time series data used for training and predicting in mathematical view, the columns represent the 4 channels of the basic shapes, the rows represent the timestep. Figure 4.5 (B) gives the sample image view for the time-series data from channel 1 to channel 4 (this is ordered based on the percentage of variance explained from high to low).



	0	1	2	3
0	-0.395128	-0.049248	0.590389	0.505762
1	-0.350766	-0.016766	0.592195	0.542087
2	-0.293203	0.027500	0.588233	0.581448
3	-0.217026	0.092231	0.557880	0.628945
4	-0.139722	0.147796	0.511146	0.672521



A B

Figure 4.5 *(A) Numeric data of each channel (B) Projected amplitude formed by the 4 basic shapes (Worm 'b')*

Finally, I have divided the 4-channel time series data into 70% training set with 30% testing set. This time the input shape 'X_train' becomes a 3-D matrix (num_samples, 50 timesteps, 4 features for training) and 'y_train' becomes a 2-D matrix with (num_samples, 50 timesteps, 4 features to be predicted) format.

4.3.2.2 The CNN model and LSTM model

Table 4.9 shows the results of average mean square error (MSE) and average Euclidean distance (ED) calculated by the real data and predicted data generated by CNN model and

LSTM model. For projected amplitude prediction problem, I did not design the visualisation approach to display information.

	CNN model		**LSTM model**	
	Average MSE	**Average ED**	**Average MSE**	**Average ED**
Worm b	0.0059	0.149	0.0003	0.030
Worm c	0.0056	0.143	0.0003	0.035
Worm d	0.0045	0.128	0.0002	0.026
Worm e	0.0037	0.113	0.0003	0.034
Worm f	0.0035	0.114	0.0002	0.029
Worm h	0.0042	0.124	0.0004	0.035
Worm j	0.0054	0.141	0.0011	0.063
Worm k	0.0047	0.128	0.0006	0.044
Worm l	0.0053	0.135	0.0009	0.050
Worm m	0.0043	0.120	0.0004	0.037

Table 4.9 *Average MSE and Average ED of CNN and LSTM model for 10 worms (projected amplitude trend prediction)*

4.3.2.3 The Bidirectional LSTM model

Table 4.10 presents the results of average mean square error (MSE) and average Euclidean distance (ED) calculated using the predicted data generated by bidirectional LSTM model and the real data. No visualisation approach designed for this section.

	Bidirectional LSTM Model	
	Average MSE	**Average ED**
Worm b	0.0004	0.034

Worm c	0.0003	0.032
Worm d	0.0003	0.032
Worm e	0.0004	0.037
Worm f	0.0004	0.037
Worm h	0.0004	0.036
Worm j	0.0006	0.043
Worm k	0.0005	0.041
Worm l	0.0009	0.050
Worm m	0.0003	0.030

Table 4.10 *Average MSE and Average ED of Bidirectional LSTM model for 10 worms (projected amplitude trend prediction)*

4.3.2.4 Further exploration

In section 4.3.2.1, I have found the 4 basic shapes and built the 4-channel time series data. After predicting the possible trends of the 4-channel time series data using CNN model, LSTM model and Bidirectional LSTM model, I also calculated the mean square error (MSE) for each of the 4 channels individually. Figure 4.6 illustrate an example of the calculation process for channel 1.

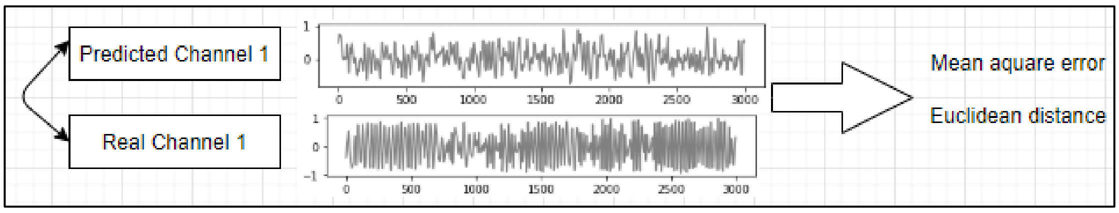


Figure 4.6 *Testing using the entire predicted channel and corresponding real channel*

I calculated all the 4 channels using the same approach with CNN, LSTM and bidirectional LSTM models for 10 worms, then I combined the results for each worm to calculate the average value for each channel with different models. Table 4.11 presents the final results after the calculations.

	LSTM model	**CNN model**	**Bidirectional LSTM model**

	Average MSE	**Average MSE**	**Average MSE**
Channel 1	0.000524	0.005623	0.000474
Channel 2	0.000539	0.005714	0.000465
Channel 3	0.000444	0.004098	0.000310
Channel 4	0.000491	0.003493	0.000594

Table 4.11 *Average MSE for each model with respect to each time series channel*

Chapter 5
Evaluation

Because this project is designed for prediction tasks, so I have implemented each model, training and testing its performance on the data mainly in Chapter 4. In this chapter, I will discuss and compare the results from the experiments implemented in Chapter 4, then choose the most suitable method for different tasks.

5.1 High-dimensional shape prediction

For high-dimensional shape prediction task, I have designed three predictive neural network structures to achieve this goal including CNN model, LSTM model and CNN-LSTM model. After training and testing each of the models on 10 different worms, I used mean square error (MSE) and Euclidean distance (ED) to evaluate each model.

	LSTM Model	**CNN Model**	**CNN-LSTM Model**
Average MSE	0.00029	0.0065	0.042
Average ED	0.15	0.78	1.84

Table 5.1 *The average MSE and ED value among 10 worms for each model (Shape prediction)*

From Table 5.1, it is obvious that LSTM model contributed the best results than others, and CNN-LSTM Model contributed the worst results. The performance of these three models can also be compared by observing the images in Table 4.2 and Table 4.4, in table 4.2, we can see that the predicted shapes of LSTM model and CNN model can almost match the real shapes especially the LSTM generated shapes, the degree of bending of the body also matched very well. However, in Table 4.4, many of the predicted shapes have significant differences from the real shapes. In a word, for high-dimensional shape prediction tasks, LSTM is still the best choice as the predictive model.

5.2 Low-dimensional trend prediction

5.2.1 Task 1: Head movement trends prediction

For this prediction tasks, I have designed three predictive models to finish the task. I used CNN model, LSTM model and Bidirectional LSTM model to predict the head movement for 10 individual worms. To evaluate each model, I have divided the dataset into 70% training set and 30% test set, then I used mean square error (MSE) and Euclidean distance (ED) as evaluation metrics on test set to reflect the performance of each model. Table 5.2 presents the average value of MSE and ED for each model.

	LSTM Model	**CNN Model**	**Bidirectional LSTM Model**
Average MSE	0.0011	0.0041	0.0006
Average ED	0.027	0.053	0.022

Table 5.2 *The average MSE and ED for each model among 10 worms (head movement trends prediction)*

From table 5.2, we can find that the Bidirectional LSTM model contributed the best results because it has the lowest MSE and ED value than others. CNN model is the worst model in this prediction task.

The differences can also be found by observing the images in table 4.6 and table 4.8, which predicted 1000 frames in the test dataset. In table 4.6 we can see that the predicted lines for LSTM and CNN could almost fit the real line if the head of the worm moves toward one direction, but when the worm turns its head to another direction, then this two models may not fit the real line again. However, in table 4.8, we can hardly see the differences between the real line and the predicted line generated by Bidirectional LSTM Model, some of the predicted lines even overlap with the real line.

In a word, Bidirectional LSTM Model is the best model in this project when predicting the movement trends of worms' heads.

5.2.2 Task 2: Basic shapes projection trends prediction

For this task, I also used CNN model, LSTM model and Bidirectional model to make prediction. Then I used mean square error (MSE) and Euclidean distance (ED) to evaluate the performance of these models for each worm. Table 5.3 illustrates the average MSE and ED values among all worms for each model.

	LSTM Model	**CNN Model**	**Bidirectional LSTM Model**

Average MSE	0.0005	0.0047	0.0005
Average ED	0.038	0.13	0.037

Table 5.3 *The average MSE and ED for each model among 10 worms (Basic shapes projection trends prediction)*

From table 5.3, we can find that both LSTM model and Bidirectional LSTM model can contribute the great testing performance, they have almost the same average MSE and ED values. However, in section 4.3.2.4, I have tested the average MSE among 10 worms for each channel, since the 4 channels of the time series data are ordered by the percentage of variance explained from high to low, which means that channel 1 and channel 2 explain more variance than the other two channels so that they will be more important. In table 4.11, it can be seen that the Bidirectional LSTM model has the lower MSE values in channel 1 and channel 2 than LSTM model. All in all, based on these analyses, I think Bidirectional LSTM model is the best one in this project when predicting the projected amplitude trends.

Chapter 6
Conclusion

6.1 Conclusion

In this project, I have designed overall 4 predictive neural network structures to explore the predictability of worm behaviours in two perspectives. I have accessed both full dimensional representations and low dimensional representations of worm behaviours. For full dimensional representations, I designed the CNN model, LSTM model and CNN-LSTM model to predict the shapes of different worms given their previous shapes.

For low dimensional representations, I designed the CNN model, LSTM model and Bidirectional model to predict the movement trends of different worms' heads. And I also applied a dimension reduction approach to find 4 basic shapes ('Eigenworms') in the dataset, which can explain over 95% variance and can be used to reconstruct any shapes.

Although the phenotype of some models is not as good as expected, we can still prove that deep learning algorithms can be used to predicted behaviours of worms, especially some Recurrent Neural Network based algorithms such as LSTM and Bidirectional LSTM models. In both full and low dimensional representations, LSTM networks contributed accurate and reliable results.

6.2 Limitation of the project

1. Because this model can only predict static frames of worms and some possible movement trends, so that it is difficult to predict their specific movement trajectories and identify their exact actions.

2. Because the models used in this project are relatively simple, and the parameters are not studied and tested very well, so that the generated results of some models are very poor such as CNN-LSTM model used in full dimensional shape prediction problem.

6.3 Future work

1. As for the selection of models, in the future I will use more complex deep learning architectures to predict the worm behaviours that can help improve the prediction quality such as GANs, attention mechanism based RNN model or some large scale pretrained models.

2. As for the prediction task, in the future I will learn more dynamic modelling approaches that can help simulate and construct the trajectories of worm’s movement rather than just shapes or trends.

3. As for extension task, since each individual worm must have its own properties which should be different from others, so I should build some models to extract their features then try to classify different worms based on these features.

6.4 Personal Reflection

This project is one of the most difficult projects I have ever done because I don’t know anything about worm behaviours or morphology of worms before receiving this project.

When I first got the project, I proposed some very advanced algorithms to my supervisor with confidence. However, I misjudged my research abilities and the time until I have to use other methods for the project.

I would like to express my sincere gratitude to my supervisor, whenever I ask him for advice on some specific problems, he will always reply to my information quickly and provide many useful suggestions.

To be honest, I am not satisfied with what I did for this project, because there were too many details and points should be deeply improved and analysed, and I am feeling so bad when I failed to use the initially mentioned approaches. I will keep learning and developing my skilled based on the experience of doing this project in the future.

Reference

Mark, E. and Riddle lab. 2020. *What is C. elegans ?.* [Online]. [Accessed 25 May 2020]. Available from: https://cbs.umn.edu/cgc/what-c-elegans

Ann K.C., Bruce, W., Martin, C. 2015. *A Transparent window into biology: A primer on Caenorhabditis elegans.* [Online]. [Accessed 27 May 2020]. Available from: http://www.wormbook.org/chapters/www_celegansintro/celegansintro.pdf

Uluç Kadıoğlu. 2017. *C. elegans Under the Microscope*. [Online]. [Accessed 27 June 2020]. Available from: https://www.youtube.com/watch?v=ltX5PEPT2ZQ

Alan, M. and Aleksandra, M. 2019. *Advantages of using Caenorhabditis Elegans as a Model Organism.* [Online]. [Accessed 31 May 2020]. Available from: https://andor.oxinst.com/learning/view/article/advantages-of-using-caenorhabditis-elegans-as-a-model-organism

Phys.org. 2013. *Better understanding of the movements of C.elegans worm will make a big difference in biomedical research*. [Online]. [Accessed 20 June 2020]. Available from: https://phys.org/news/2013-10-movements-celegans-worm-big-difference.html

Li Kezhi., Javer Avelino., E Eric. Ed. 2017. *Recurrent Neural Networks with Interpretable Cells Predict and Classify WormBehaviour*. 31st Conference on Neural Information Processing Systems (NIPS 2017), Long Beach, CA, USA.

Duda, M. Ma, R., Haber, N., Wall, D. 2016. Use of machine learning for behavioral distinction of autism and ADHD. *Translational Psychiatry.* 6. Corpus ID: 2513216.

Yemini E, Jucikas T, Grundy LJ, Brown AE, Schafer WR. 2013. A database of Caenorhabditis elegans behavioral phenotypes. *Nat Methods*.**10**(9), pp. 877-9.

Gomez-Marin, A., Paton, J., Kampff, A. et al. 2014. Big behavioral data: psychology, ethology and the foundations of neuroscience. *Nat Neurosci.* 17, pp.1455–1462.

Yourgenome.org. 2015. *Why use the worm in research ?.* [Online]. [Accessed 7 July 2020]. Available from: https://www.yourgenome.org/facts/why-use-the-worm-in-research

Andre, E.X.B., Eviatar, I.Y., Laura, J.G., Tadas, J., William R.S. 2013. A dictionary of behavioral motifs reveals clusters of genes affecting Caenorhabditis elegans locomotion. *PNAS*. 112(2), pp. 791-796.

Andrej, K., Justin, J., Feifei, L. 2016. Visualizing and understanding recurrent networks. *ICLR Workshop Track.*

Tom, M.M. 1997. *Machine learning.* [Online]. [Accessed 8 July 2020]. Available from: http://profsite.um.ac.ir/~monsefi/machine-learning/pdf/Machine-Learning-Tom-Mitchell.pdf

Daniel, F. 2020. *What is Machine Learning*. [Online]. [Accessed 10 July 2020]. Available from: https://emerj.com/ai-glossary-terms/what-is-machine-learning/

ISAH, S. 2018. *SuperVize Me: What's the Difference Between Supervised, Unsupervised, Semi-Supervised and Reinforcement Learning ?.* [Online]. [Accessed 12 July 2020]. Available from: https://blogs.nvidia.com/blog/2018/08/02/supervised-unsupervised-learning/

Jason, B. 2017. *Difference Between Classification and Regression in Machine Learning*. [Online]. [Accessed 15 July 2020]. Available from: https://machinelearningmastery.com/classification-versus-regression-in-machine-learning/

Datarobot. 2020. Machine Learning Life Cycle. [Online]. [Accessed 15 July 2020]. Available from: https://www.datarobot.com/wiki/machine-learning-life-cycle/

Ana, M. 2020. *Why Unsupervised Machine Learning is the Future of Cybersecurity*. [Online]. [Accessed 15 July 2020]. Available from: https://www.technative.io/why-unsupervised-machine-learning-is-the-future-of-cybersecurity/

Faizan, S. 2017. *Deep Learning vs. Machine Learning – the essential differences you need to know !*. [Online]. [Accessed 16 July 2020]. Available from: https://www.analyticsvidhya.com/blog/2017/04/comparison-between-deep-learning-machine-learning/

Marshall, H. 2019. *Deep Learning*. [Online]. [Accessed 13 July 2020]. Available from: https://www.investopedia.com/terms/d/deep-learning.asp#deep-learning-vs-machine-learning

Sambit, M. 2018. *Why Deep Learning over Traditional Machine Learning ?.* [Online]. [Accessed 16 July 2020]. Available from: https://towardsdatascience.com/why-deep-learning-is-needed-over-traditional-machine-learning-1b6a99177063

Tanmay, S. 2019. *Difference between ML and deep learning with respect to Splitting of the dataset into….* [Online]. [Accessed 20 July 2020]. Available from: https://mc.ai/difference-between-ml-and-deep-learning-with-respect-to-splitting-of-the-dataset-into/

Alejandro C.B. 2017. *Building AI Applications Using Deep Learning*. [Online]. [Accessed 21 July 2020]. Available from: https://albahnsen.com/2017/06/06/building-ai-applications-using-deep-learning/

Michael, N. 2019. *Neural Networks and Deep Learning*. [Online]. [Accessed 20 July 2020]. Available from: http://neuralnetworksanddeeplearning.com/

Sushan, B.C. 2019. *How do artificial neural networks work ?.* [Online]. [Accessed 20 July 2020]. Available from: https://www.quora.com/How-do-artificial-neural-networks-work

Facundo, B., Juan, M.G., Victor, D.F. 2017. Prediction of wind pressure coefficients on building surfaces using Artificial Neural Networks. *Energy and Buildings.* 158.

Yupeng, C. 2018. *Images Classification and Difference measure by Deep Neural Networks.*

Margaret, R. 2015. *backpropagation algorithm*. [Online]. [Assessed 21 July 2020]. Available from: https://searchenterpriseai.techtarget.com/definition/backpropagation-algorithm

Afshine, A and Shervine, A. 2020. *Recurrent Neural Networks cheatsheet.* [Online]. [Accessed 23 July 2020]. Available from: https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-recurrent-neural-networks

Wei, B., Jun, Y., Yulei, R. 2017. A deep learning framework for financial time series using stacked autoencoders and long-short term memory. [Online]. [Accessed 25 July 2020]. Available from: https://www.researchgate.net/publication/318991900_A_deep_learning_framework_for_financial_time_series_using_stacked_autoencoders_and_long-short_term_memory

Educba. 2020. Recurrent Neural Networks (RNN). [Online]. [Accessed 27 July 2020]. Available from: https://www.educba.com/recurrent-neural-networks-rnn/

Jurgen, S., Sepp, H. 1997. LONG SHORT-TERM MEMORY. *Neural Computation*. **9**(8), pp:1735-1780.

Colah. 2015. *Understanding LSTM Networks*. [Online]. [Accessed 1 August 2020]. Available from: https://colah.github.io/posts/2015-08-Understanding-LSTMs/

Renu, K. 2018. *Convolutional Neural Network(CNN) Simplified*. [Online]. [Accessed 2 August 2020]. Available from: https://medium.com/datadriveninvestor/convolutional-neural-network-cnn-simplified-ecafd4ee52c5

Rehan, A.S. 2013. How to Use Convolutional Neural Networks for Time Series Classification. [Online]. [Accessed 2 August 2020]. Available from:

https://medium.com/@Rehan_Sayyad/how-to-use-convolutional-neural-networks-for-time-series-classification-80575131a474

Bertalan, G., Andre, E.X.B. 2016. Deriving Shape-Based Features for C. elegans Locomotion Using Dimensionality Reduction Methods. *Neurosci*.

Ryuzo, S., Morimichi, F., Katsunori, H., Yuishi, L. 2013. Evaluation of Head Movement Periodicity and Irregularity during Locomotion of Caenorhabditis elegans. *Front behave Neurosce.*

Techopedia. 2017. *Principal Component Analysis (PCA)*. [Online]. [Accessed 15 August 2020]. Available from: https://www.techopedia.com/definition/32509/principal-component-analysis-pca

Jason, B. 2018. *How to Develop LSTM Models for Time Series Forecasting*. [Online]. [Accessed 16 August 2020]. Available from: https://machinelearningmastery.com/how-to-develop-lstm-models-for-time-series-forecasting/

Jason, B. 2019. *CNN Long Short-Term Memory Networks.* [Online]. [Accessed 17 August 2020]. Available from: https://machinelearningmastery.com/cnn-long-short-term-memory-networks/.

Jason, B. 2020. *How to Develop a Bidirectional LSTM For Sequence Classification in Python with Keras.* [Online]. [Accessed 19 August 2020]. Available from: https://machinelearningmastery.com/develop-bidirectional-lstm-sequence-classification-python-keras/

Cory, M. 2019. *Dropout Neural Network Layer In Keras Explained.* [Online]. [Accessed 22 August 2020]. Available from: https://towardsdatascience.com/machine-learning-part-20-dropout-keras-layers-explained-8c9f6dc4c9ab

Sanket, D. 2019. *Various Optimization Algorithms For Training Neural Network*. [Online]. [Accessed 22 August 2020]. Available from: https://towardsdatascience.com/optimizers-for-training-neural-network-59450d71caf6

I2tutorials.com. 2019. *What are the differences between MSE and RMSE.* [Online]. [Accessed 23 August 2020]. Available from: https://www.i2tutorials.com/differences-between-mse-and-rmse/

Khanacademy.org. 2019. *Matrix multiplication dimensions.* [Online]. [Accessed 22 August 2020]. Available from: https://www.khanacademy.org/math/precalculus/x9e81a4f98389efdf:matrices/x9e81a4f98389efdf:properties-of-matrix-multiplication/a/matrix-multiplication-dimensions

Appendix A: External Materials

External Materials used in this project:

Programming language:

Python3 https://www.python.org/

Programming environment:

Google Colab: https://colab.research.google.com/

Libraries:

Numpy: https://numpy.org/

Pandas: https://pandas.pydata.org/

Matplotlib: https://matplotlib.org/

Tensorflow: https://www.tensorflow.org/

Keras: https://keras.io/

Scikit-learn: https://scikit-learn.org/stable/

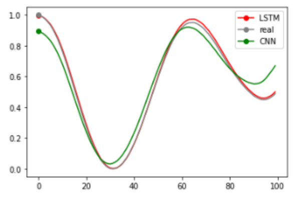
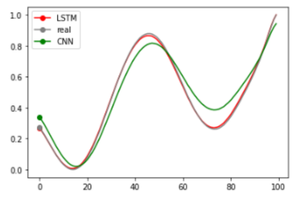
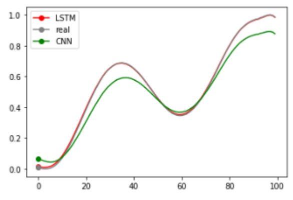
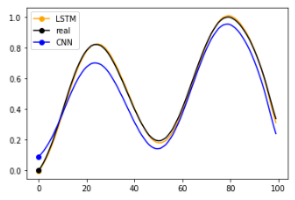
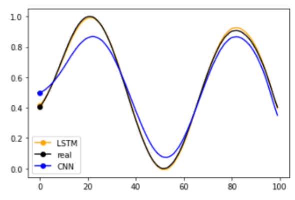
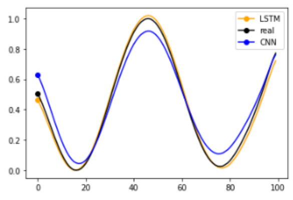
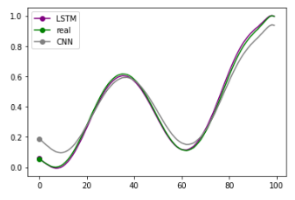
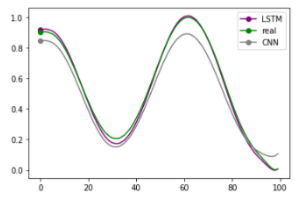
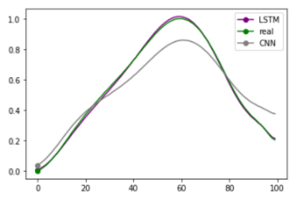
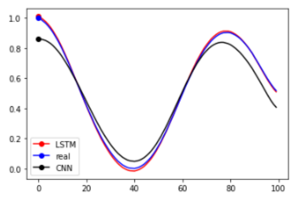
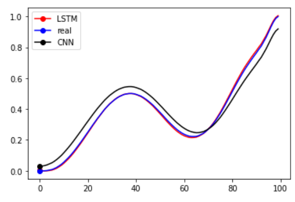
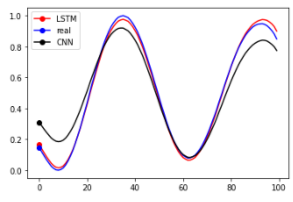
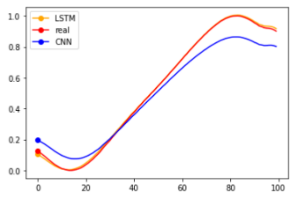
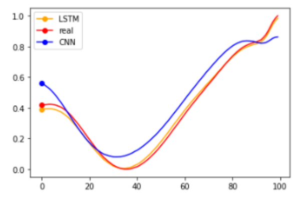
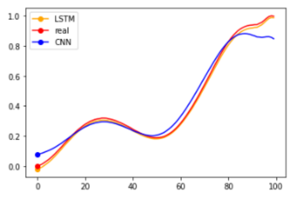
Scipy: https://www.scipy.org/

Appendix B: Code Repository

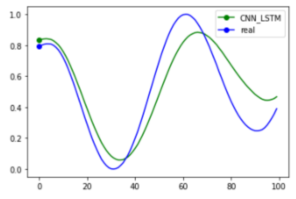
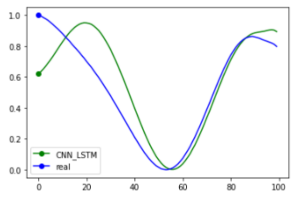
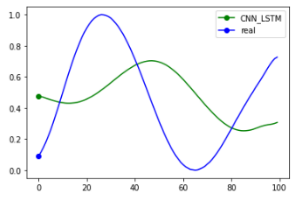
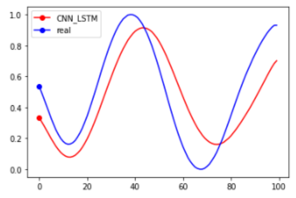
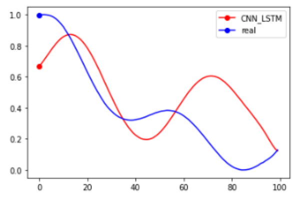
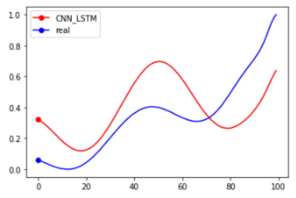
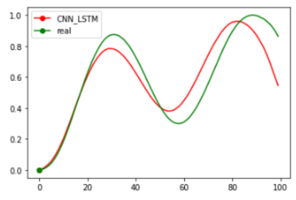
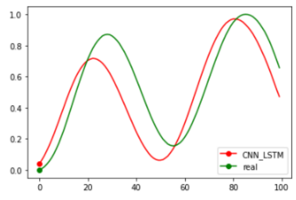
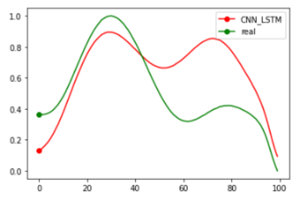
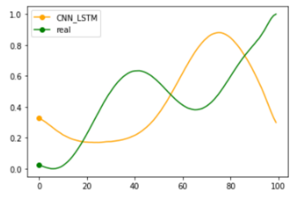
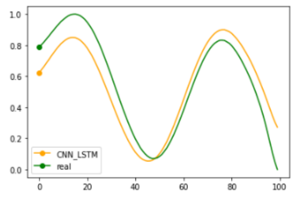
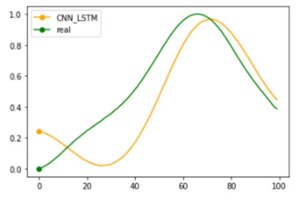
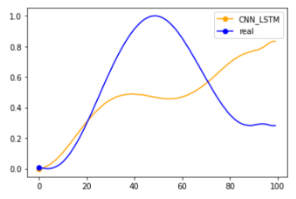
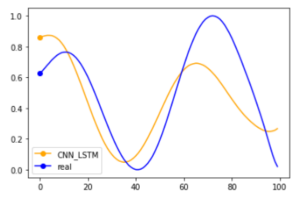
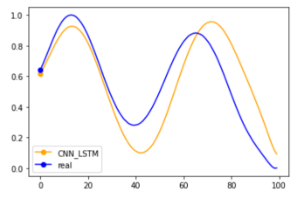
https://gitlab.com/li870216/msc-project

Appendix C: MIT LICENSE

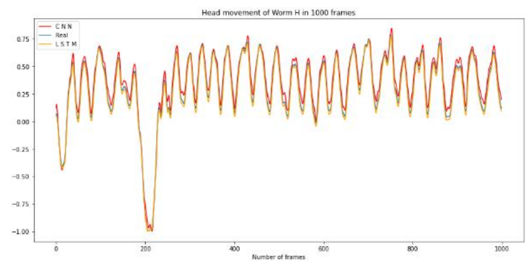
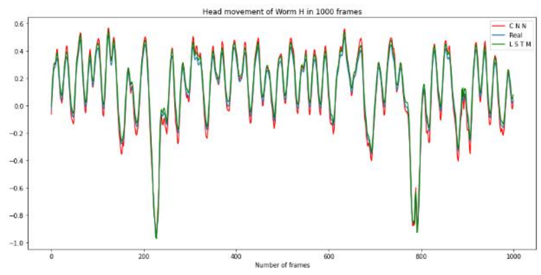
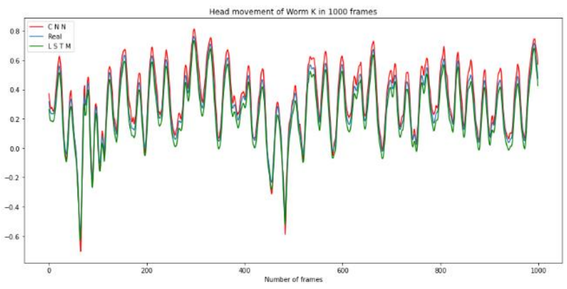
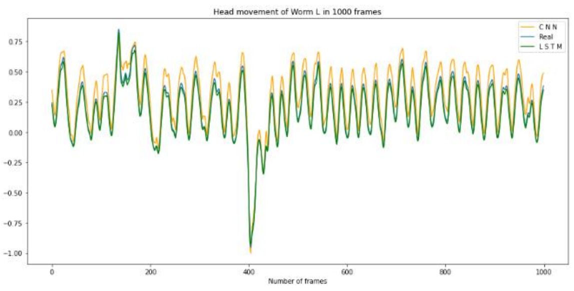
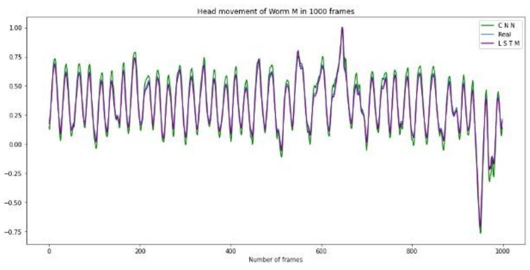
Appendix D: Chapter 4 CNN and LSTM (shape)

	CNN model		LSTM model
	Quartile 1 (25th %)	Quartile 2 (50th %)	Quartile3 (75th %)
worm h			
worm j			
worm k			
worm l			
worm m			

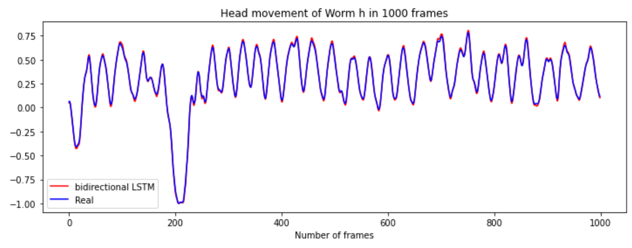
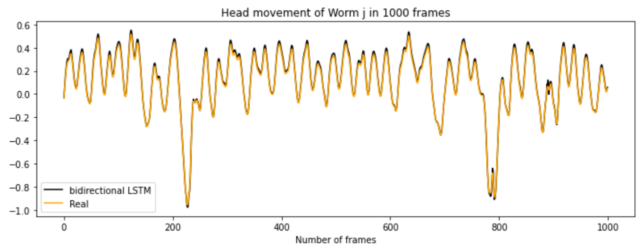
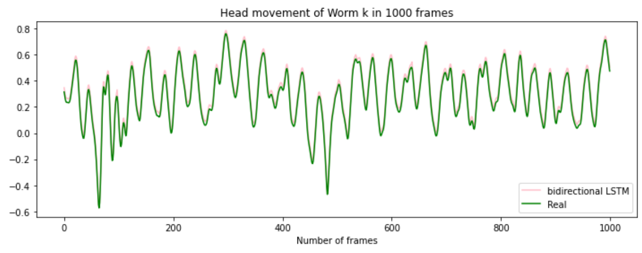
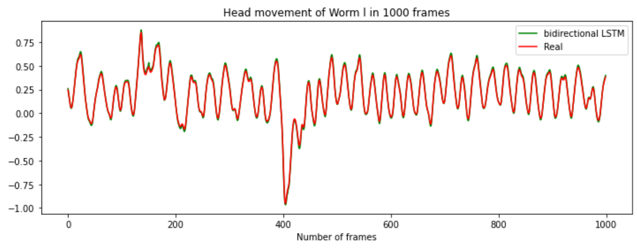
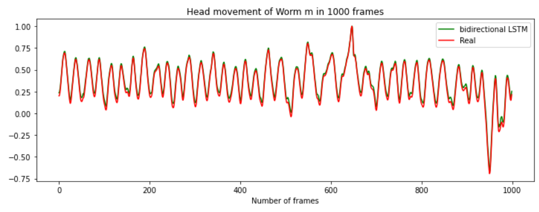
Appendix E: Chapter 4 CNN-LSTM model (shape)

	CNN-LSTM model		
	Quartile 1 (25th %)	**Quartile 2 (50th %)**	**Quartile3 (75th %)**
Worm h			
Worm j			
Worm k			
Worm l			
Worm m			

Appendix F: Chapter 4 CNN and LSTM (head)

	CNN model	LSTM model
Worm h		
Worm j		
Worm k		
Worm l		
Worm m		

Appendix G: Chapter 4 bidirectional LSTM (head)

	Bidirectional LSTM
Worm h	
Worm j	
Worm k	
Worm l	
Worm m	 Head movement of Worm m in 1000 frames bidirectional LSTM Real Number of frames