Investigating Differences in Two Visualizations from Observer's Pupil Diameter Using Bidirectional Neural Network and LSTM

Wenxuan Zhou
Research School of Computer Science
The Australian National University
Email: u6545178@anu.edu.au

Abstract. In this investigation, two visualization interfaces with similar quality are differentiated by observers' eye gaze fixations and saccades data. Besides a traditional artificial neural network, a Bidirectional Neural Network is implemented to give the network new abilities to design powerful data representation techniques, which can become a key factor in reducing network generalization error. Comparing two results, it can be summarized that BDNN does not have a significant improvement on the performance of classification, while time series data can provide more information than statistics measurements.

Keywords: Visualization Interface, Neural Network Classifier, Bidirectional Autoencoder, Long Short-term Memory

1 Introduction

Visualization can provide high levels of interaction to observers to glean knowledge from the raw data, and enable learners to draw valuable conclusions at minimal cost, describe big data in simple and innovative ways, combine data in diagrams that represent information and convey messages to observers by creating mental visual images. With a visualization interface, users can navigate information spaces and abstract away from the inherent complexity of the underlying information [1].

Several studies support that images and graphs can capture the immediate attention of observers compared to texts. When two visualizations are similar in quality, it could be hard to differentiate one from another [1].

In this setting, I would like to classify two different interfaces (radial and hierarchical) from observers' pupil diameter figure. A radial visualization is used when there is a single entity of major interest, and the hierarchical visualization is structural to show the key interconnections between hierarchies [1]. Hence this is a binary classification problem.

An eye-tracker keeps recording the pupil diameter size per 1/60th second throughout subject's whole experiment. These data are stored with left eye and right eye separately. All figures are further summarized its statistics measurement distributed by participant and question.

Among classification models, logistic regression and artificial neural network models are widely used. This seems to be motivated by their lower generalization error than decision trees and k-nearest neighbors, while being easier to build than support vector machines. Meanwhile, due to the fact that artificial neural network can be seen as nonlinear generalizations of logistic regression, it is chosen for this task [3].

Now since the pupil data is recorded as time series, a recurrent network will be well-suited for such kind of data that possess moving average components [4], given that it can retain a state representing information from an arbitrarily long context window. Unlike the standard recurrent network that suffers from vanishing and exploding gradient problems, LSTM uses special units that include a 'memory cell' to allow for a better control over the gradient flow and enable better preservation of "long-range dependencies" [5].

The extraction of meaning from trained neural networks is seen as a way to improve the acceptability of neural networks. Most methods of rule extraction use causal connections between inputs and outputs. There is some statistical indication that the outputs have causal effects on inputs. Thus, if we were to explicitly allow connections embedded in the network to be made between the output and input values as well as the usual input to output connections, it is possible that more accurate rules could be extracted [2]. Thus, a bidirectional neural network is designed to remember input patterns as well as output vectors, given either of them.

In this investigation, a simple neural network is first implemented as the classifier model of the problem. Then a bidirectional LSTM neural network is trained as autoencoder, whose encoding data is preserved for further classification. While time-series data is classified using an LSTM autoencoder. For all models, K-fold cross-validation is used for evaluation, which provides reliable results for model selection and hyperparameter tuning, as well as result comparison.

2 Methods

2.1 Data Preprocessing

In the experiment, both two interfaces (radial and hierarchical) have six questions based on the information presented, and 24 participants have all answered the 12 questions, thus there are 288 samples of pupil data in total. The string value of target class “Radial” and “Hierarchical” is encoded into numerical representation 0 and 1 respectively.

For the simple neural network, there are seven statistics measurements of the pupil diameter for each participant’s every question: Mean, standard derivation, maximum value, minimum value, range, first derivative and second derivative.

Because the maximum and minimum have already given the range, thus range attribute is deleted.

A time derivative is a derivative of a function with respect to time, usually interpreted as the rate of change of the value of the function. It seems that in the classification domain there could be objects for which function value comparison is not sufficient. There could be cases where assignment to one of the classes depends on the general shape of objects rather than on strict function value comparison. Especially for time series, it seems that some variability in the “time” domain could have a great influence on the classification process [6].

Given the range of data is varied and skewed, z-score normalization is used to rescale all data into 0 and 1 without distorting differences in the ranges, otherwise the large value may have more influence to result.

For the original pupil diameter data, only the left pupil size and right pupil size is preserved as features, since the participant's identity and the question being answered does not matter with the differences of interfaces.

After dropping off all none values and the row with at least one zero-value in the dataset (i.e. only keeping the time step where both two eyes are open), the length of each sample varies, with the longest of 2716 (subject: P52F, question: Q3) and the shortest of 1 (subject: P48F, question: Q5). There are even situations that 9 sample has meaningful records at all.

To balance the influence of each sample, those with nothing or too few data are abandoned, and for the remain 275 samples only the first 400 records are taken to train.

2.2 Simple Neural Network

Given the available features are only three attributes and the dataset of 288 samples, the neural network is not necessary to be large. A one-hidden-layer network is quite adequate for the job.

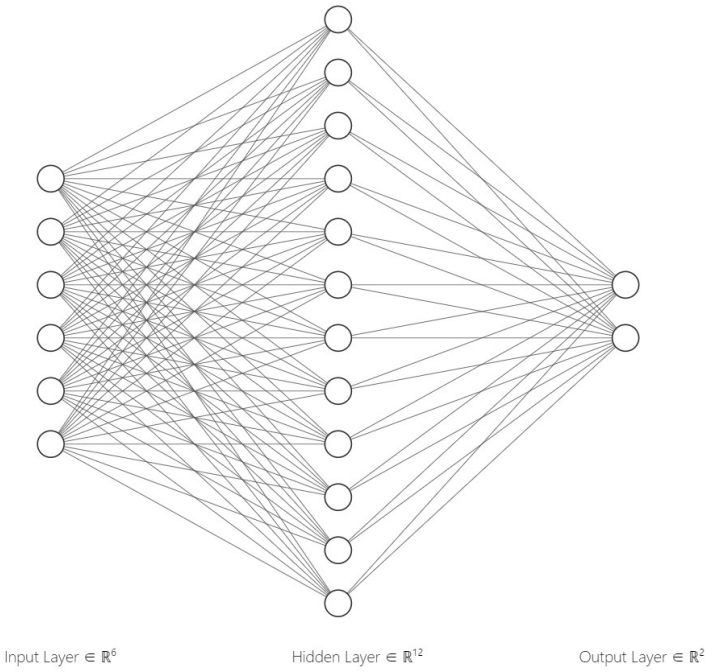

Fig. 1. The pseudo structure of the simple neural network

Picture 1 is the schematics of this simple neural network. Patterns are fed in through input layer where one node for each component. Then hidden layer receives the value connected with weights and bias and an activation function determines whether to be activated or not. Finally, the calculation is passed to output layer with one neuron for each possible desired output. Each connection is combined with weight and each neuron (except input one) has a bias. These values are the parameters trained and updated by gradient descent [7].

Given the input is normalized, tanh activation function is used in the hidden layer that maps the negative inputs strongly negative, and the zero inputs will be mapped near zero in the graph [8]. While the output value is expected to

be in the range of (0,1) as the probability of belonging to a class, Thus, sigmoid or logistic activation function is the most appropriate here.

Given this is a binary classification problem, where output is more like predicting the probability of the example belonging to class value 1, the cross-entropy loss is used here. It is the preferred loss function under the inference framework of maximum likelihood. Cross-entropy will calculate a score that summarizes the average difference between the actual and predicted probability distributions for predicting class 1. The score is minimized and a perfect cross-entropy value is 0 [9].

For the gradient descent, mini-batch algorithm is used, which splits the training dataset into small batches that are used to calculate model error and update model coefficients. Compared to other batch gradient descent that updates model at the end of each training epoch, mini-batch has higher model update frequency, allowing for a more robust convergence and avoiding local minima. Also, the batched updates provide a computationally more efficient process than stochastic gradient descent [10].

To determine the hyperparameters (learning rate, batch size, epoch size and number of hidden units), K-fold cross-validation is used to evaluate the model. In each round of k times iteration, the dataset is split into *k* parts: one part is used for validation, and the remaining *k-1* parts are merged into a training subset for model evaluation, thus reducing the pessimistic bias by using more training data in contrast to setting aside a relatively large portion of the dataset as test data. The cross-validation performance is computed as the arithmetic mean over the *k* performance estimates from the validation sets [11]. Given the dataset has only 288 samples, K is set to 5. Table 2 shows the average result for different hyperparameter configuration.

Table 1. The average accuracy of simple neural network model with different hyperparameter configuration (~ indicates that the accuracy is below 50%)

	Hidden size	Epoch number	Batch size	Learning rate	Average accuracy (%)
1	24	700	30	0.001	50.8213
2	24	800	30	0.001	52.6598
3	24	900	30	0.001	51.3642
4	32	700	30	0.001	51.7564
5	32	800	25	0.001	50.2026
6	32	800	30	0.01	~
7	32	800	30	0.001	53.4722
8	32	800	30	0.0001	~
9	32	800	35	0.001	52.9927
10	32	900	30	0.001	50.2563
11	48	800	30	0.001	~
12	48	900	30	0.001	52.4299
13	48	1000	30	0.001	~

As can be seen from line {6,7,8} in table 1, no matter what batch size or epoch number is, the model with learning rate of 0.001 always has the best average accuracy.

Comparing line {5,7,9}, with same network structure, training epochs and learning rate, the best performance appears when the batch size is of 30.

Given the fact that a bigger neural network (i.e. more neurons in hidden layer) will need more training epochs to converge, hidden size and epoch number are compared together. Fixed on batch size and learning rate confirmed previously, lines {1,2,3,4,7,10,11,12,13} list several optimal combinations. A bigger neural network usually needs more time to update the parameters, changing the model from underfitting to optimal. Meanwhile, more training epochs does not necessarily mean a better result, but may lead to overfitting.

It is obvious that the best score lies in line 7, whose hyperparameter setting is the result of hyperparameter optimization.

2.3 Bidirectional Neural Network

Bidirectional Neural Networks (BDNNs) is designed based on multi-layer perceptrons trained by a generalized form of the error back-propagation algorithm [2]. It is actually an autoencoder and can be trained as a context addressable memory or a cluster center finder, given different relation between input and output.

The model for BDNN is also implemented as a one-hidden-layer network, due to the limitation of dataset size. Thus, the single hidden layer can be considered as encoding layer whose result is the encoded input. Because of the property of autoencoder, the size of input and output is the same.

An autoencoder can have two kinds of structure: undercomplete and overcomplete. In undercomplete network, a compact representation is produced in encoding layer, i.e. hidden layer has fewer units than the input. In this case, the dimension is reduced [12]. While for an overcomplete network, the number of units in hidden layer is bigger than the input layer. From the experiment of my previous research, representing raw data into more features cannot improve classification accuracy. Therefore, only undercomplete BDNN will be implemented.

Given the input is still normalized to have the same range, the activation function should have meaningful output for both positive and negative inputs. Hyperbolic tangent is similar to sigmoid with the difference that is symmetric to the

origin and its slope is steeper [12]. For the loss function, a MSELoss is adequate, which computes the mean squared error between input and reconstruction information for optimizer to minimize the difference.

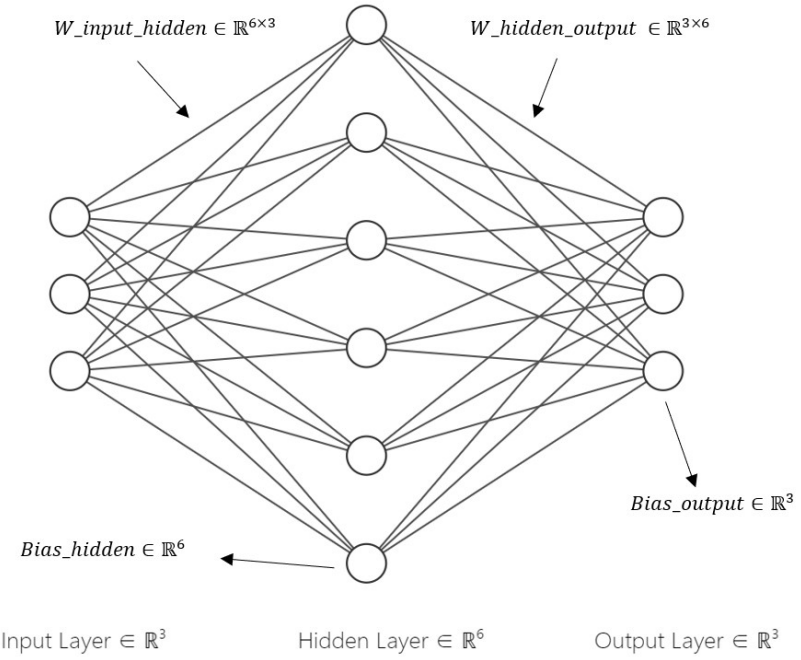

Fig. 2. The pseudo structure of autoencoder, with weight and bias indication in forward training

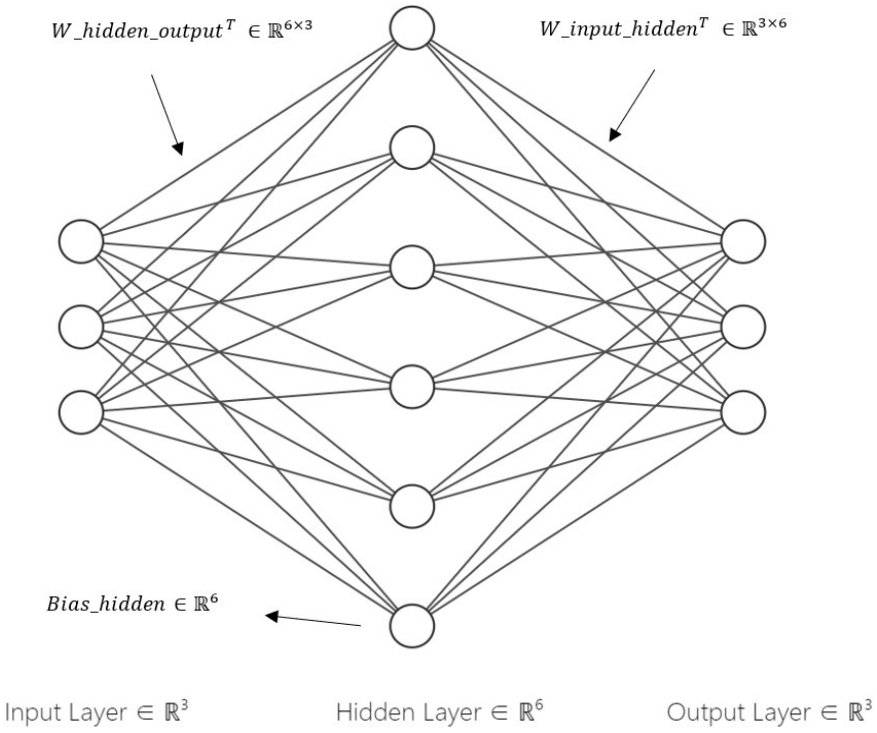

Fig. 3. The pseudo structure of autoencoder, with weight and bias indication in backward training

A bidirectional neural network means that the model will be trained in both forward and backward directions. As illustrated in graph 2, when training this two-layer network in the normal direction, there will be four learnable parameters assigned as model attributes: weight between input and hidden layer (W_input_hidden), biases of hidden layer (Bias_hidden), weight between hidden and output layer (W_hidden_output) and biases of output layer (Bias_output). Then when training in the reverse direction, demonstrated in graph 3, same weights will be used, as well as the biases in hidden units. However, given the model weight in Pytorch library has its specific shape, a transpose processing will be needed.

The model will be initialized with random parameter given. For the first iteration, the network is trained normally in the forward direction. Then, the direction of training is reversed, where a new model will be built based on the parameters passed from the previous iteration. The direction is maintained until some maximum number of epochs have been spent in the current direction, or the overall error in the current direction is less than the error tolerance predefined before training. This sequence of reversals of training direction continues until reaching a specified iteration number.

Same as the simple neural network introduced previously, the model is evaluated through K-fold cross-validation procedure. Table 2 shows the average loss for model with different hyperparameter configuration.

Table 2. The average loss of bidirectional neural network model with different hyperparameter configuration

	Hidden size	Iteration number	Epoch number	Batch size	Loss threshold	Average loss
1	3	50	100	30	0.001	0.1019
2	3	50	100	10	0.001	0.0613
3	3	100	100	30	0.001	0.0960
4	3	100	100	10	0.001	0.0333
5	3	200	100	30	0.001	0.0871
6	3	200	100	20	0.001	0.0814
7	3	200	100	10	0.001	0.0602
8	3	300	100	10	0.001	0.0799
9	2	50	100	10	0.001	0.0608
10	2	100	100	10	0.001	0.0627
11	2	200	100	10	0.001	0.0571

Comparing lines {1,2}, {3,4} and {5,6,7}, a smaller batch size will always lead to a lower average loss. An autoencoder with bigger hidden layer also needs more training epochs to converge, so hidden size and iteration number are taken into consideration together. Lines {2,4,7,8} and {9,10,11} list several combinations. The reason why loss threshold and epoch number remain unchanged during hyperparameter tuning will be discussed in the result section. The best score lies in line 4.

Now a dataset of encoding values can be produced, which will further be fed into simple neural network for classification, with different hyperparameter configuration. Here the detail of model section and hyperparameter tuning is omitted. The final result is:

For dataset from undercomplete structure BDNN (feature dimension: 3). Using simple neural network with hidden size of 12, epochs of 800, batch size of 30 and learning rate of 0.001, the average result 53.7160%.

2.4 Long Short-term Memory Autoencoder

Standard neural networks have limitations in that they rely on the assumption of independence among the training and test examples. After each example (data point) is processed, the entire state of the network is lost. If each example is generated independently, this presents no problem. But if data points are related in time or space, this is unacceptable [13].

Recurrent neural network (RNN) is a connectionist model with the ability to selectively pass information across sequence steps, while processing sequential data one element at a time. Thus, they can model input and/or output consisting of sequences of elements that are not independent [13]. Long Short-Term Memory network (LSTM) is a special kind of RNN, capable of learning long-term dependencies [14]. A typical LSTM network is comprised of different memory blocks called cells. There are two states that are being transferred to the next cell; the cell state and the hidden state. The memory blocks are responsible for remembering things and manipulations to this memory are done through three major mechanisms: forget gate, input gate and output gate. The information flows through the cell states, so that LSTM can selectively remember or forget things [15].

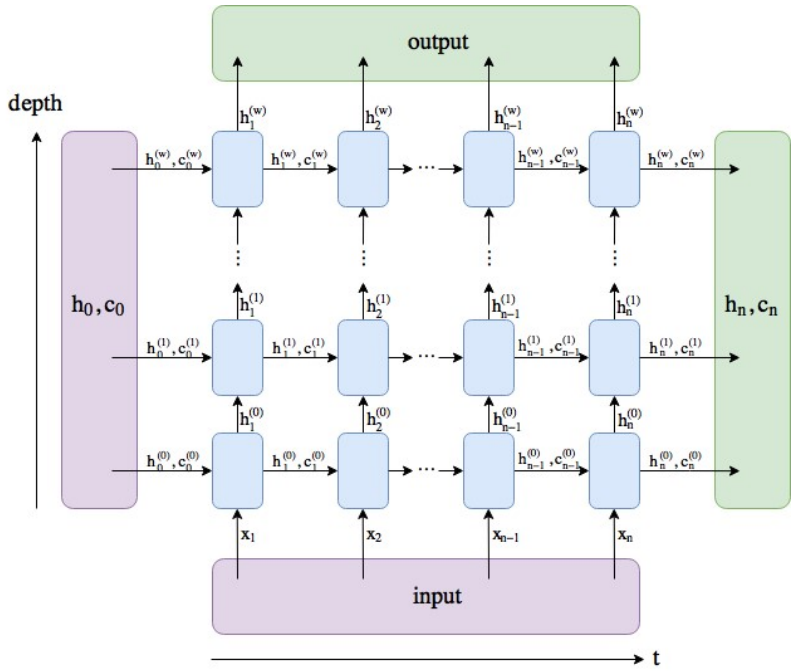

Fig. 4. The pseudo structure of LSTM [18]

For the implementation of LSTM autoencoder, I first tried with two LSTMs only (the Pytorch structure is illustrated below), one as encoder and one as decoder. However, this model cannot converge at all, no matter what changes to the functions or hyperparameters have been made.

```
LSTMautoencoder(
  (encoder): LSTM(2, 32, batch_first=True)
  (decoder): LSTM(32, 2, batch_first=True)
)
```

Then a fully-connected layer to added at the end (the Pytorch structure is illustrated below) to transpose the output into the same shape as input, so that the hidden size of LSTM itself does not need to match the dimension. And it turns out that the performance is up to expectation.

```
LSTMautoencoder(
  (encoder): LSTM(2, 32, batch_first=True)
  (decoder): LSTM(32, 16, batch_first=True)
  (output_layer): Linear(in_features=16, out_features=2, bias=True)
)
```

Now given this structure, bidirectional training is no longer possible. In Pytorch library, LSTM module has four learnable variables: the input-hidden weights of the kth layer, the hidden-hidden weights of the kth layer, the input-hidden bias of the kth layer and the hidden-hidden bias of the kth layer, each having the shape of (4*hidden_size, hidden_size) for weights and (4*hidden_size) for biases. Because of this, when trying to training the model backwardly (i.e. feed the data in a reversed direction), no weights or biases of encoder and decoder can match by shape. Figure 5 demonstrates the actual variables of the model.

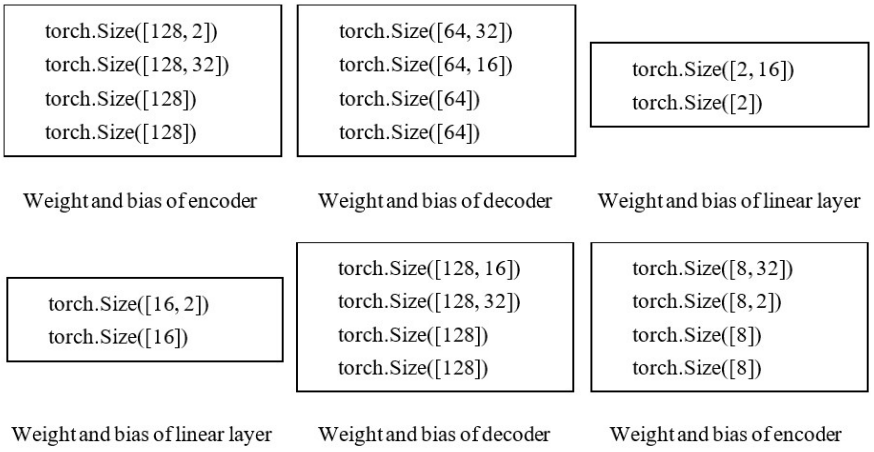

Fig. 5. The variables of model when training in the forward direction (above) and backward direction (below)

Another critical issue for LSTM is the outputs. A Pytorch LSTM has three output components: output (the output features h_t from the last layer for each time step), h_n (the hidden state for time step equals to sequence length), and c_n (the cell state for each element in the batch). To be specific, the output comprises all hidden states in the last depth layer, while h_n and c_n comprise the hidden states after the last timestep [18-16], which can be intuitively seen from figure 3. In order to keep all information through batches, the output of encoder is further fed to the decoder, while the hidden state is used for classification, which can be seen as the intermediate representation of sample. Also, the decoder passes its output to the fully-connected layer.

In summary, time-series data are imported into the autoencoder, and its output is transposed into decoding data using a linear layer. While another linear layer takes the hidden state of encoder as input to make classification on interfaces. An MSELoss is used as the criterion of autoencoder performance. The actual Pytorch model is illustrated below:

```
LSTMautoencoder(
  (encoder): LSTM(2, 32, batch_first=True)
  (decoder): LSTM(32, 16, batch_first=True)
  (output_layer): Linear(in_features=16, out_features=2, bias=True)
  (classify_layer): Linear(in_features=32, out_features=2, bias=True)
)
```

Same as before, the whole model is evaluated through K-fold cross-validation procedure by the classification accuracy, not the loss of autoencoder. Table 3 shows the average accuracy with different hyperparameter configuration.

Table 3. The average accuracy of LSTM autoencoder with different hyperparameter configuration (~ indicates that the accuracy is below 50%)

	Hidden size	Epoch number	Sequence length	Learning rate	Average accuracy (%)
1	48	30	100	0.01	50.0174
2	48	20	100	0.01	51.2903
3	48	10	100	0.01	~
4	32	5	100	0.01	~
5	32	10	100	0.1	~
6	32	10	100	0.01	56.0192
7	32	10	100	0.001	52.3763
8	32	10	80	0.01	~
9	32	10	200	0.01	51.2908
10	32	20	100	0.01	~
11	16	20	100	0.01	~
12	16	10	100	0.01	54.9233
13	16	5	100	0.01	~
14	8	10	100	0.01	~

As can be seen from lines {5,6,7}, the model has the best average accuracy with learning rate of 0.01. Given each sample provides first 400 records and the customized dataset is designed to give model sample by sample, the sequence length can only be chosen from the divisor of 400, due to the fact that batch size and sequence size in LSTM module must have same product. Large batches always mean faster training time, while small batches with many timesteps can have better generalization and accuracy [18].

In lines {5,8,9}, sequence length of 80, 100 and 200 are attempted with same hidden size, epoch number of learning rate.

Again, fixed on batch size and learning rate confirmed previously, hidden size and epoch number are compared together. Lines {1,2,3,4,7,10,11,12,13} list several optimal combinations. This time the best score lies in line 7, whose hyperparameter setting is the result of hyperparameter optimization.

3 Result and Discussion

3.1 Result of Simple Neural Network and Bidirectional Neural Network

From table 1, the best accuracy can be archived by simple neural network is around 53%. On the other hand, the encoding data from BDNN seems to improve the classification accuracy (i.e. from 53.4722% to 53.7160%).

Figure 6 is the five confusion matrices for 5-fold cross-validation using that best model. Viewing all tables, there is no clear difference between radial interface and hierarchical interface in the classification performance.

	Radial (predicted)	Hierarchical (predicted)
Radial (actual)	14	15
Hierarchical (actual)	13	16

	Radial (predicted)	Hierarchical (predicted)
Radial (actual)	21	11
Hierarchical (actual)	10	16

	Radial (predicted)	Hierarchical (predicted)
Radial (actual)	9	21
Hierarchical (actual)	12	15

	Radial (predicted)	Hierarchical (predicted)
Radial (actual)	14	12
Hierarchical (actual)	14	17

	Radial (predicted)	Hierarchical (predicted)
Radial (actual)	16	11
Hierarchical (actual)	15	16

Fig. 6. The confusion matrixes for 5-fold cross-validation using simple neural network

As talked in method section, both loss threshold and epoch number remain unchanged during hyperparameter tuning. This is because the total loss during whole training process never decreases to be lower than 0.5 and keeps chopping till the end. Therefore, the actual training times equal to iteration number * epoch number, thus there is no need to change epoch number parameter as well.

It is worth mentioning that even though the K-fold cross-validation is used to confirm the accuracy of model evaluation, the actual predicting performance is still not stable, which may be caused by the initialization of model parameters. However, it is apparently showed in the figure 6 that difference among training-testing dataset can lead to huge deviation. This implies that the original dataset is not balanced and may have some outliers that heavily influence the prediction.

3.2 Result of Long Short-term Memory Autoencoder

From table 3, The best accuracy obtained by LSTM autoencoder is around 56%. It is intuitively that many hyperparameter configurations will lead to absolute wrong model. This implies that the classification of time-series data can have many undetermined uncertainties.

Another notable thing occurs in the training. Given this is an autoencoder model, besides the classification accuracy of encoder's hidden states, the loss between input and decoder's output is also a criterion measurement. However, when the loss keeps decreasing, the classify becomes overfitting.

3.3 Compare Results

Given three different classification results regarding to pupil diameter, a comparation can be conducted. Unexpectedly, none of these models have a satisfactory outcome. Even though the result is better than random guess (i.e. accuracy of 50% for a binary classification), it is still not high enough to be considered as a good model. While among them, LSTM autoencoder has the best accuracy.

This observation implies that representing raw data into fewer features can be considered as a data preprocessing that tackles the problem inside the dataset, such as data unbalance, large outliers, etc. Therefore, the input becomes more evidential to distinguish two visualization interfaces. Statistics measurement summarizes the data into a smaller dimension, with the cost of losing the persistent information in long time-series records.

4 Conclusion and Future Work

In conclusion, the introduction of bidirectional autoencoder does improve the prediction accuracy, but the progress is not too much. On the other hand, time-series data has its meaning that cannot be replaced by any statistic.

The time-series data for LSTM are preprocessed so that only the records with two eyes open are preserved. This is due to the fact that the raw data have many wield situations such as only one pupil open for a while, or there are no available figures in one sample, whereas it also means that all eye blink periods are deleted from samples, which can an important factor in classifying two interfaces. Therefore, a better and smarter data preprocessing method should be applied.

In addition, only first 400 rows of each sample are taken as training dataset. With the worries that varied data amount may lead to unbalanced impact to the result, this rash cut off can also result in a loss of information, which could make contribution to the poor performance of overall classification.

During the whole evaluation process, all hyperparameter configurations are fixed for each iteration of K-fold cross validation. Further I would like to change some of these values during training. For example, the learning rate can be set to a small value at the beginning then increase later. Same is in BDNN training where learning rate and loss threshold keeps unchanged in both directions. The use of dynamic coefficients may result in faster convergence of the network.

When training the LSTM autoencoder, there are indeed many questions concerned with the design of structure. For example, why adding a linear layer at the end makes the model starts to converge? Is that possible with larger layer size in encoder and smaller layer size in decoder? Another experiment can be made based on these concerns.

References

[1] M.Z. Hossain, T. Gedeon, S. Caldwell, L. Copeland, R. Jones, and C. Chow. 2018. Investigating Differences in Two Visualisations from Observer's Fixations and Saccades. In Proceedings of Australasian Computer Science Week conference, Brisbane, QLD, Australia, 30 January – 2 February 2018 (ACSW'2018), 4 pages

[2] Nejad, A. F., & Gedeon, T. D. (1995, November). Bidirectional neural networks and class prototypes. In Neural Networks, 1995. Proceedings., IEEE International Conference on (Vol. 3, pp. 1322-1327). IEEE.

[3] Stephan Dreiseitl, and Lucila Ohno-Machado: Logistic regression and artificial neural network classification models: a methodology review, In Journal of Biomedical Informatics 35 (2002) 352–359

[4] Jerome T. Connor, R. Douglas Martin: Recurrent Neural Networks and Robust Time Series Prediction, IEEE TRANSACTIONS ON NEURAL NETWORKS, VOL. 5, NO. 2, MARCH 1994

[5] Difference between feedback RNN and LSTM/GRU, URL (version: 2016-07-07): https://stats.stackexchange.com/q/222587

[6] Tomasz Górecki & Maciej Łuczak (2014) First and Second Derivatives in Time Series Classification Using DTW, Communications in Statistics - Simulation and Computation, 43:9, 2081-2092, DOI: 10.1080/03610918.2013.775296

[7] Understanding Neural Networks: What, How and Why?, https://towardsdatascience.com/understanding-neural-networks-what-how-and-why-18ec703ebd31

[8] Sagar Sharma: Activation Functions in Neural Networks, https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6

[9] How to Choose Loss Functions When Training Deep Learning Neural Networks, https://machinelearningmastery.com/how-to-choose-loss-functions-when-training-deep-learning-neural-networks/

[10] A Gentle Introduction to Mini-Batch Gradient Descent and How to Configure Batch Size, https://machinelearningmastery.com/gentle-introduction-mini-batch-gradient-descent-configure-batch-size/

[11] Sebastian Raschka: Model Evaluation, Model Selection, and Algorithm Selection in Machine Learning, arXiv:1811.12808, 2018

[12] Practical tutorial on autoencoders for nonlinear feature fusion (Part 1), https://kstathou.github.io/posts/2018/03/autoencoders-part-1/

[13] Zachary C. Lipton, John Berkowitz, Charles Elkan: A Critical Review of Recurrent Neural Networks for Sequence Learning, arXiv:1506.00019v4 [cs.LG] 17 Oct 2015

[14] Understanding LSTM Networks, http://colah.github.io/posts/2015-08-Understanding-LSTMs/

[15] Pranjal Srivastava: Essentials of Deep Learning : Introduction to Long Short Term Memory, https://www.analyticsvidhya.com/blog/2017/12/fundamentals-of-deep-learning-introduction-to-lstm/

[16] nnnmmm: What's the difference between "hidden" and "output" in PyTorch LSTM?, https://stackoverflow.com/questions/48302810/whats-the-difference-between-hidden-and-output-in-pytorch-lstm