Proceedings of the 2nd International Conference on Computing Innovation and Applied Physics
DOI: 10.54254/2753-8818/19/20230523

Player identification based on player behavioral characteristics

Hexin Li[1,*], Yizhi Fang[2]

[1]Department of Math and Applied Mathematics, University Nottingham Ningbo China, Ningbo, 315000, China
[2]Department of Mathematics and Statistics, Guangdong University of Foreign Studies Guangzhou, Guangdong, 510006, China

sgyhl2@nottingham.edu.cn, frank-fangyz@139.com

Abstract. In order to maintain a fair competition environment and enjoyable experience for players, millions of dollars have been spent on against cheating in video games. There is limited research on more sophisticated forms of cheating like "play-for-hire" whereby players pay others to play for themselves. Our work develops a model to identify each player from player behavioural characteristics, which will contribute to solve the "play-for-hire" problem. Firstly, we recorded interactions between players and the game as multivariate time series. Next, we tried to use CNN and LSTM to classify data as corresponding players and we do some feature processing and parameter optimization to improve our result. We found that LSTM is acting better than CNN in higher dimensions, which achieved an accuracy of nearly 87%.

Keywords: LSTM, CNN, multivariate, player's behaviour.

1. Introduction

The aim of this report is to represent and compare the accuracy of different models in given condition which is a Pac-man game designed by ourselves.

1.1. Background information:

Cheating is a common phenomenon in video games. According to a report from DENUVO (irdeto) [1], a company that aims to protect digital IP through anti-temper technology, 78% of players quit the game because of the existence of cheaters. What's more, nearly one-third of the players admitted to cheating in games. These numbers mean billions of revenues may be impacted, and cheaters are ruining the ecology of online games.

There has been substantial research tends to detect and eliminate the cheating activities like "Triggerbot", "Aimbot" and "Extrasensory perception" [2-5]. By contrast, there is limited research on other more sophisticated forms of cheating, such as "play-for-hire" whereby a player pays another player to play on his/her behalf. "Play-for-hire" is a massive underground economy in developing countries. Advertisements of "games training services" can be found easily in all major e-shops in China, we can infer the influence of manual cheating is badly widespread. Our work wants to solve manual cheating problems.

DOI: 10.54254/2753-8818/19/20230523

The key to solving the cheating problem from human players is being able to identify every player. If we can identify every single player based on their behaviors, we can detect cheat when somebody asks another player to play for him.

The reason that the data of the giant game company is hard to achieve and the purpose is to simplify the setting and get rid of the noise, we design a new game. The Player needs to use the keyboard to avoid enemies and drive the ball to the destination.

Following the hypothesis from Jose [2], we assumed that the behavior of the players can be represented by multivariate time series and can be classified by our models. Besides moving data (up, down, left, right), we added up some common features which may impact the decision actions of players in our work. For example, the relative position between the player, enemy, goal, etc. (Details can be seen in Data Pre-processing). These features can be found in many games and can be applied to other games easily.

We want to use multivariate time series to train a model which can match new game series with the right players. We selected both traditional statistical models and deep learning algorithms like CNN, and LSTM. Then the optimization for a specific model is made. In the end, we will explain the result and provide some guidance for future studies.

2. Part a: basic introduction of the deep learning models

In this project, two deep learning models and algorithms are used. They are CNN and LSTM. In this part, the essential working process and main features of these algorithms will be described.

Firstly, CNN will be introduced simply. CNN is the abbreviation of a convolutional neural network. Based on its name, convolution is a key part of this model. By using this model, the target data will be expanded in vector form with certain dimensions.

For example, the following equations show how this expanded be done. **X** is the information of two-dimension picture; **H** is an information matrix; **U** is a vector of bias parameters; '**W**' and '**V**' are weight tensors but in different expressions. The footnotes are set that k= i + a and l= j + b because the two tensors' elements are bijections, and in this way, the whole picture can be covered by moving from position (i, j).

$$[\boldsymbol{H}]_{i,j} = [\boldsymbol{U}]_{i,j} + \sum_{k}\sum_{l}[\boldsymbol{W}]_{i,j,k,l}[\boldsymbol{X}]_{k,l}$$

$$= [\boldsymbol{U}]_{i,j} + \sum_{a}\sum_{b}[\boldsymbol{V}]_{i,j,a,b}[\boldsymbol{X}]_{i+a,j+b}$$

Then the convolution kernel will be set and trained by iteration and the convolution layer will be built based on the convolution kernel and input vectors. The function of the convolution kernel is trying to grasp the feature of the target and the convolution layer makes it possible to optimize the convolution kernel and the weight tensor.

The next two features of CNN will be mentioned. They are **translation invariance** and **locality.**

The following equation shows the **translation invariance**: it means the bias parameter is a constant and will not be affected by the position of information.

$$[\boldsymbol{H}]_{i,j} = u + \sum_{a}\sum_{b}[\boldsymbol{V}]_{a,b}[\boldsymbol{X}]_{i+a,j+b}$$

This equation shows the **locality**: it says the algorithm should collect the information for training in the limit area which should not be far from (i, j).

$$[\boldsymbol{H}]_{i,j} = u + \sum_{a=-\Delta}^{\Delta}\sum_{b=-\Delta}^{\Delta}[\boldsymbol{V}]_{a,b}[\boldsymbol{X}]_{i+a,j+b}$$

Next, LSTM will be introduced. LSTM is a special case of RNN and the full name of them are Recurrent Neural Networks with Long Short-Term Memory. RNN is mainly used to deal with dependent

DOI: 10.54254/2753-8818/19/20230523

data. Auto-regressive models and Markov models are applications of RNN. The essential procedure of RNN is quite similar to CNN to some extent,

But in this special case LSTM, has some new components. Which will be mentioned later.

The first new component is **hidden state**.

$$\boldsymbol{H}_t = \phi(\boldsymbol{X}_t\boldsymbol{W}_{xt} + \boldsymbol{H}_{t-1}\boldsymbol{W}_{hh} + \boldsymbol{b}_h)$$

This equation shows how the hidden state works in the algorithm.

Φ is the activate function; footnote **t** is the time step; **X** is the input; **W** is the weight parameters of the hidden state; **H** is the hidden state; **b** is the bias parameters. With a hidden state, LSTM can capture and record historical information.

$$\boldsymbol{O}_t = \boldsymbol{H}_t\boldsymbol{W}_{hq} + \boldsymbol{b}_q$$

This equation shows the output layer, while **W** is the output layer weight parameters and **b** are the output layer bias parameters.

The second new component is the **gate**. There are three types of gates: **I**nput gate, **F**orget gate, and **O**utput gate.

$$\boldsymbol{I}_t = \sigma(\boldsymbol{X}_t\boldsymbol{W}_{xi} + \boldsymbol{H}_{t-1}\boldsymbol{W}_{hi} + \boldsymbol{b}_i)$$

$$\boldsymbol{F}_t = \sigma(\boldsymbol{X}_t\boldsymbol{W}_{xf} + \boldsymbol{H}_{t-1}\boldsymbol{W}_{hf} + \boldsymbol{b}_f)$$

$$\boldsymbol{O}_t = \sigma(\boldsymbol{X}_t\boldsymbol{W}_{xo} + \boldsymbol{H}_{t-1}\boldsymbol{W}_{ho} + \boldsymbol{b}_o)$$

The sigmoid function is activation function and it makes the value range of three gates located in (0,1). These gates can bring the data on the current time step and the former time step into the LSTM

The third new component is the memory cell. In this part, the candidate memory cell will be introduced first.

$$\widetilde{\boldsymbol{C}_t} = \tanh(\boldsymbol{X}_t\boldsymbol{W}_{xc} + \boldsymbol{H}_{t-1}\boldsymbol{W}_{hc} + \boldsymbol{b}_c)$$

The candidate memory cell is calculated in a quite similar way to gates and the output will be candidates for a hidden state.

Then normal memory cells will be mentioned.

$$\boldsymbol{C}_t = \boldsymbol{F}_t \odot \boldsymbol{C}_{t-1} + \boldsymbol{I}_t \odot \widetilde{\boldsymbol{C}_t}$$

In this equation ⊙ means Hadamard Production. This equation represents the aim that: the input gate decides how much data will be used in the candidate memory cell and forget gate decides how much data will be used in the former time step‘s memory cell. This design can defuse the ‘gradient-disappear’ problem and can better capture the middle-long distance-dependent relationship.

After the gates and memory cell are defined, the way to calculate the hidden state is also defined

$$\boldsymbol{H}_t = \boldsymbol{O}_t \odot \tanh(\boldsymbol{C}_t)$$

Of course, this is one of the versions of cell memory: tanh version, which can make sure the value of **H** locates in (0,1). If the output gate is close to 1, the memory information will be effectively delivered to the prediction part, while if the output gate is close to 0, only the information contained in the memory cell will be kept and the hidden state will not be refreshed.

In comparison, these two kinds of algorithms have their own preferred aspects. So in the remainder of the report, the accuracy of these two algorithms will be tested and compared in the given condition.

3. Part b: game and data collection

As a part of the project, the game (see Figure 1.)played an extremely important role in collecting experimental data, which will later be used to train the models.

DOI: 10.54254/2753-8818/19/20230523

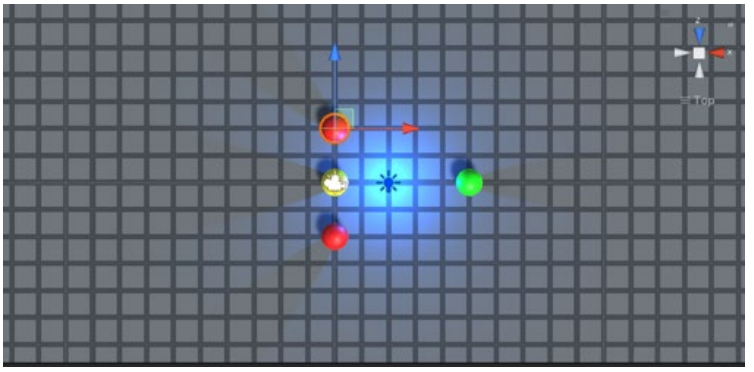


Figure 1. Game display.

There are two important reasons to use games to collect experimental data:

1. Compared to data collected from real life or existing data sets on the web, we have more freedom to obtain data in different dimensions during the experiment.

2. The game environment is ideal and not subject to accidental interference, so it is of low noise. This is more friendly to us who are new to neural networks.

Besides it is also interesting to create our own unique dataset.

3.1. Goal of the player

During the game, the player controls the yellow ball and tries to kick the green ball into the light spot, and he should try to Prevent the yellow ball from hitting the two red balls who are the enemy. When the green ball and the light point collide, the light point will appear at the next random location. The player will repeat this process for 15 minutes

3.2. The strategy of enemy1

Figure 2. Predatory path1.

Figure 2. shows enemy1's strategy is eye tracking, which means its speed direction is always toward the player. In the picture above the predator performs like a tracking missile which is exactly what enemy1 do to the yellow ball.

3.3. The strategy of enemy2:

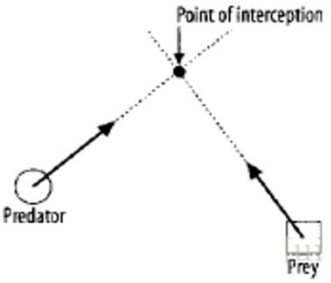


Figure 3. Predatory path2.

DOI: 10.54254/2753-8818/19/20230523

Figure 3. shows enemy2's strategy is a little different from that of Enemy1, it has the ability to predict-- it will read the player's velocity vector. It's more about intercepting players instead of just chasing.

3.4. The green ball's movement

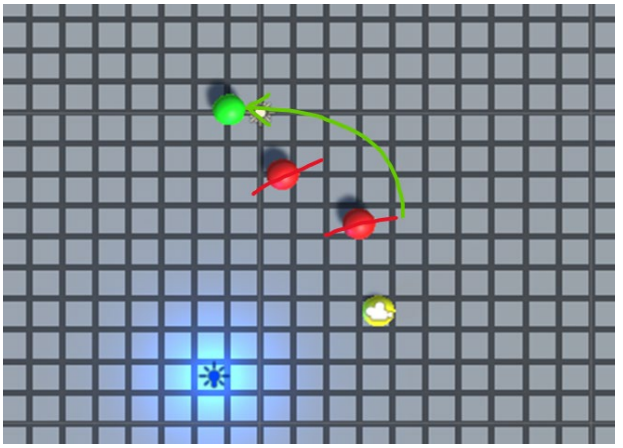


Figure 4. Player's movement.

From Figure 4. for green ball, it will flee away if the player is too closed to it. To increase the playability and difficulty of the game, a program is written to make the green ball tend to hide behind the two red balls.

4. Part c: model build and comparison

4.1. Model build

4.1.1. CNN.

(1) Prepare the pipeline

The training set and the test set are distributed and changed the label to one-hot vector [6]:

Then normalize the data to between 0 and 1 so that the computation was reduced.

(2) set up the CNN (see code in appendix)

Structure: convolutional layer and connected layer.

Initially, using the same padding way and activation function 'relu' and the optimizer is 'adam' and the loss function is 'categorical_crossentropy'.

(3) Assessment strategies

Table 1. and Figure 5. show that indicators are chosen as followed: **Loss** ,**Accuracy** and **Confusion matrix**

Loss

$$L = -[y \log \hat{y} + (1 - y) \log(1 - \hat{y})]$$

Accuracy

Table 1. Model accuracy formula.

Original Label	Tested:Positive sample	Tested:Negative sample
Positive sample	TP	FN
Negative sample	FP	TN

$$ACC = \frac{TP + TN}{TP + FN + FP + TN}$$

DOI: 10.54254/2753-8818/19/20230523

Confusion matrix

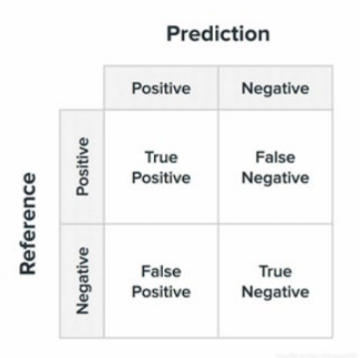


Figure 5. Model confusion matrix formula.

The result of the CNN is shown in Figure 6. and Figure 7.

Loss:

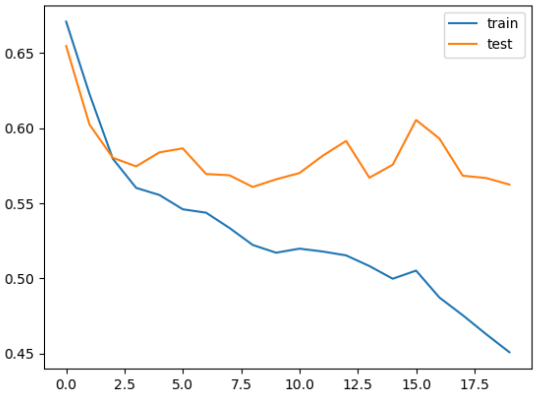


Figure 6. CNN model loss.

Accuracy (73.24%) and confusion matrix

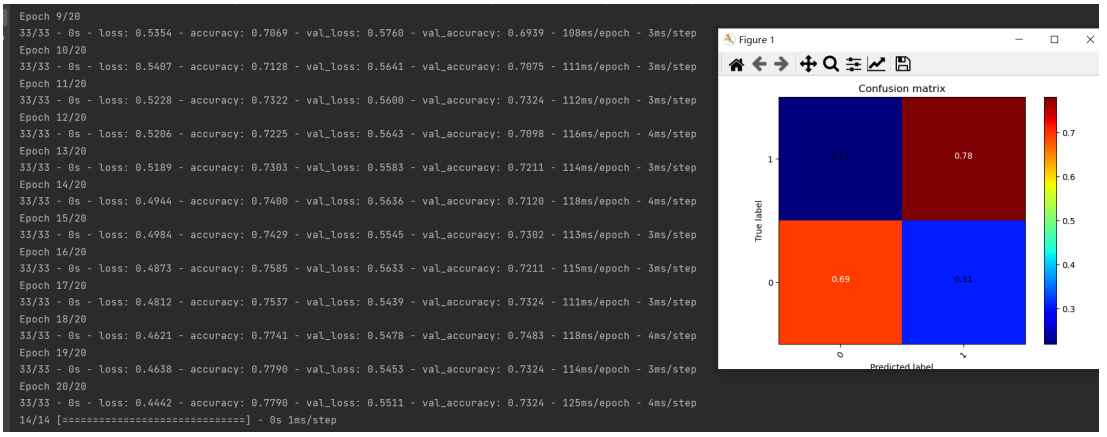


Figure 7. CNN model accuracy and confusion matrix.

4.1.2. LSTM. Prepare the pipeline (same procedure as CNN)
set up the LSTM (see code in appendix)
Structure: LSTM cells and connected layers

DOI: 10.54254/2753-8818/19/20230523

Initially, using the same padding way and activation function ‘relu’ and the optimizer is ‘adam’ and the loss function is ‘categorical_crossentropy’.

(1) assessment strategies (Same procedure as CNN), and the results are shown in Figure 8,9 and 10.

Loss:

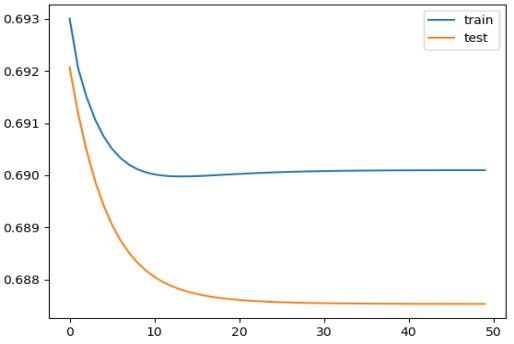


Figure 8. LSTM Model loss.

Accuracy (65.08%) and confusion matrix

```
15/15 - 0s - loss: 0.6551 - accuracy: 0.6323 - val_loss: 0.6658 - val_accuracy: 0.6508 - 57ms/epoch - 4ms/step
Epoch 45/50
15/15 - 0s - loss: 0.6545 - accuracy: 0.6372 - val_loss: 0.6653 - val_accuracy: 0.6485 - 64ms/epoch - 4ms/step
Epoch 46/50
15/15 - 0s - loss: 0.6539 - accuracy: 0.6372 - val_loss: 0.6649 - val_accuracy: 0.6508 - 66ms/epoch - 4ms/step
Epoch 47/50
15/15 - 0s - loss: 0.6534 - accuracy: 0.6372 - val_loss: 0.6645 - val_accuracy: 0.6463 - 55ms/epoch - 4ms/step
Epoch 48/50
15/15 - 0s - loss: 0.6528 - accuracy: 0.6381 - val_loss: 0.6641 - val_accuracy: 0.6440 - 57ms/epoch - 4ms/step
Epoch 49/50
15/15 - 0s - loss: 0.6522 - accuracy: 0.6362 - val_loss: 0.6637 - val_accuracy: 0.6463 - 72ms/epoch - 5ms/step
Epoch 50/50
15/15 - 0s - loss: 0.6517 - accuracy: 0.6362 - val_loss: 0.6633 - val_accuracy: 0.6508 - 72ms/epoch - 5ms/step
```

Figure 9. LSTM model accuracy.

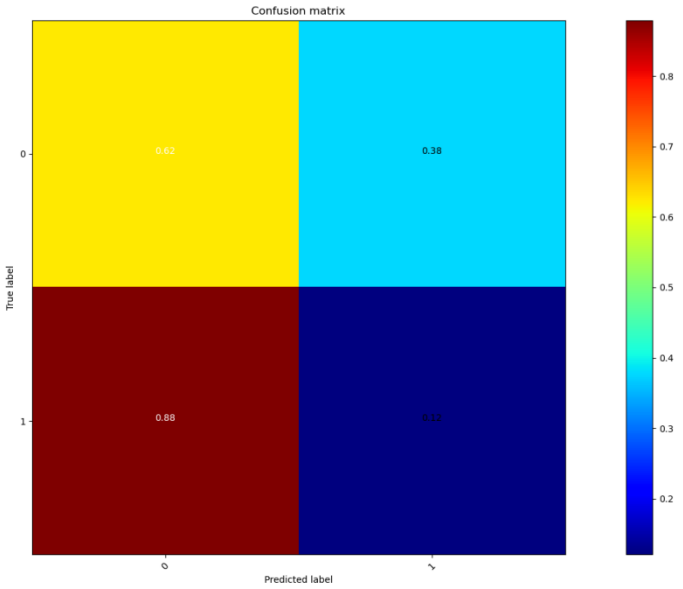


Figure 10. LSTM model confusion matrix.

DOI: 10.54254/2753-8818/19/20230523

4.2. Comparison

Accuracy:

- CNN 73% (with best confusion matrix which means better classification)
- LSTM 63%

Result:

- CNN is more accurate. (with data density N = 500; dense layers = 128; batch_size = 32)

4.3. Problem analysis

Problem analysis and some optimization plans:

- The data lack some useful features, maybe another strategy.
- Data should be expanded.
- The training process should be optimized.
- We should use some strategies to avoid overfitting.

Hence, the group discussed the next optimization strategies, decided to enrich and expand the feature of the data and use another strategy to process the data. Then the model and related algorithms will be adjusted and optimized in order to compare the LSTM and CNN again to find the best solution.

5. Part d: optimization on LSTM

As shown in the previous section, CNN performs better than LSTM in some cases. There are limits to such results, however. Firstly, limited by the previous operating equipment, LSTM only uses univariate input in the input dimension, which does not reflect the advantages of LSTM [7]. Secondly, data preprocessing is also very important in neural networks. Next, in order to verify the superiority of LSTM in processing time series, it will also be improved in the two directions mentioned above.

5.1. Data preprocessing

Data preprocessing consists mainly of the following steps:

5.1.1. Deviation calculation. Make data represent changes in the position of objects over milliseconds.

*5.1.2. Detect interfering rows of data. S*ome frames take significantly longer to execute than others, and this breaks the consistency of the data recording, which can have a huge impact on subsequent training. It is important to mark them and avoid using them in the LSTM training.

5.1.3. Select proper columns. Data in different columns may represent the same information, and it will be inefficient to bring them all into the sequence. Therefore, in the training of LSTM, I removed all the input data from the user, because *their* coordinates and the player's coordinates represented repeated information

5.2. Model structure

The structure of the models are shown in Table 2. In the part of parameter adjustment, I adjusted the length and dimension of input data and the number of neurons in the model, but the overall framework did not change.

A total of two LSTM and three dense layers are used in this model. The last dense layer's activation function is ‘sigmoid’.

The loss function, optimizer and metrics are shown as below:

model.comile(loss=‘binary_crossentropy’, optimizer=‘adam’, metrics=[‘binary_accuracy’])

5.3. Result analysis and parameter tuning

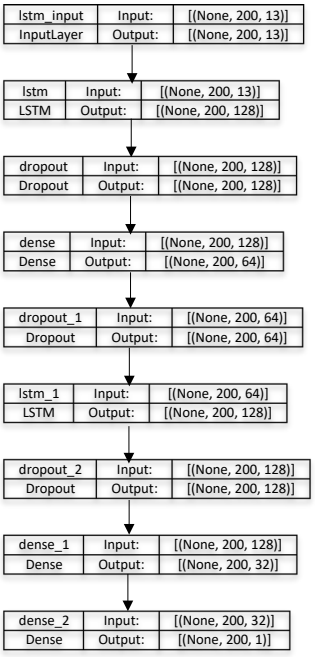
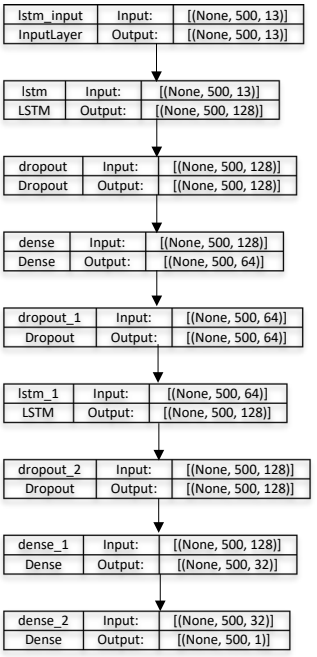
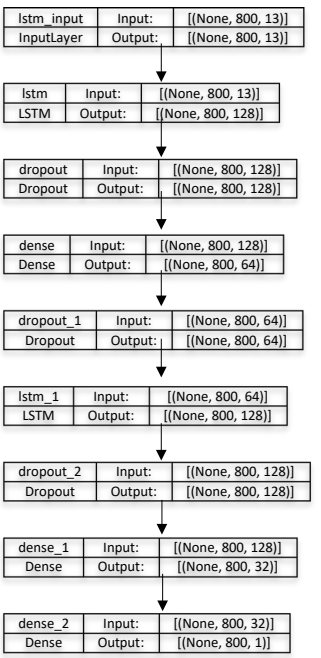
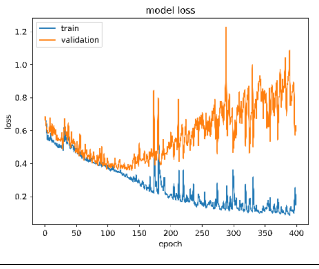
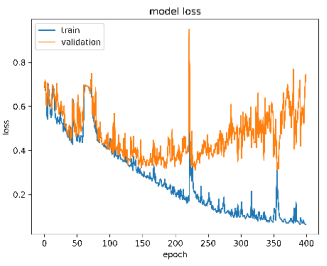
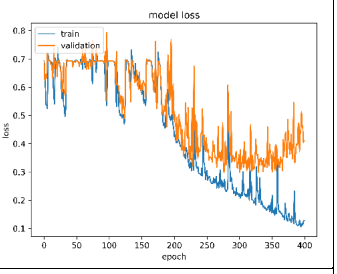
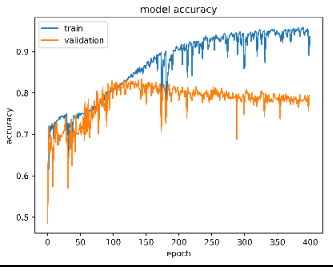
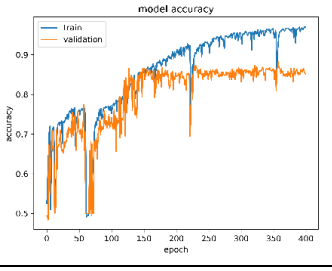
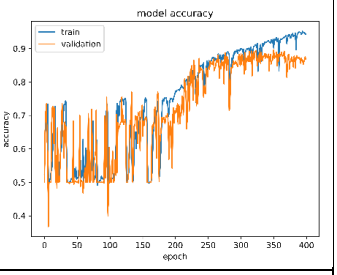
Thanks to cudNN by Nvidia, Training tasks can be completed in less time. During the parameter adjustment part, the learning rate is 0.001 by default, and the batch size is not adjusted either, it is set to 32.

DOI: 10.54254/2753-8818/19/20230523

The emphasis of parameter adjustment is on length adjustment of input time series and neuron number adjustment.

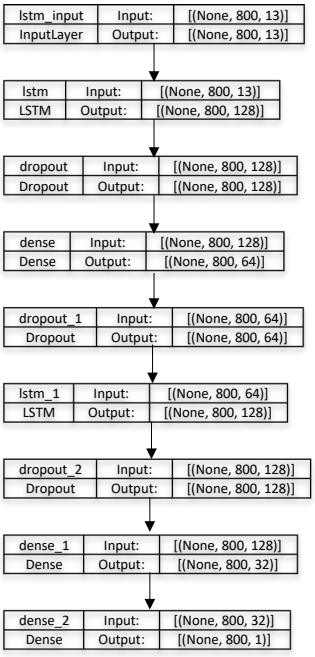
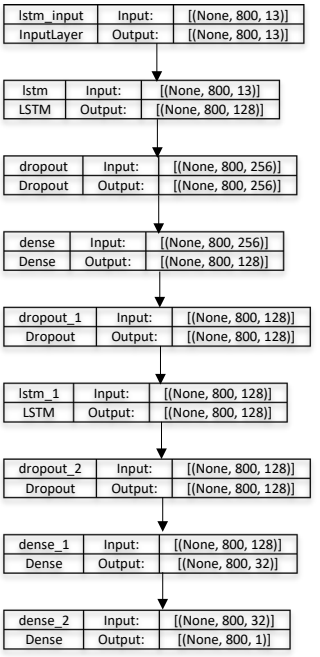
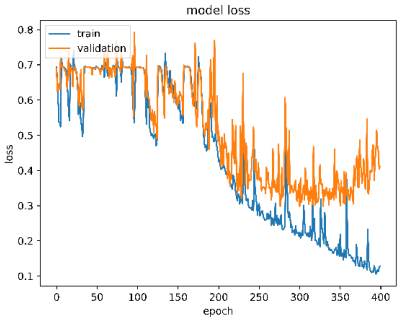
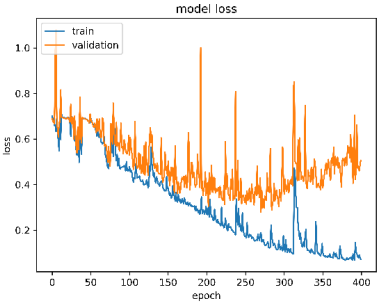
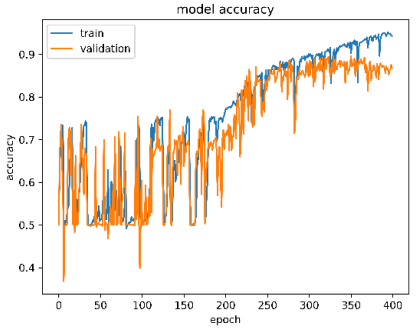
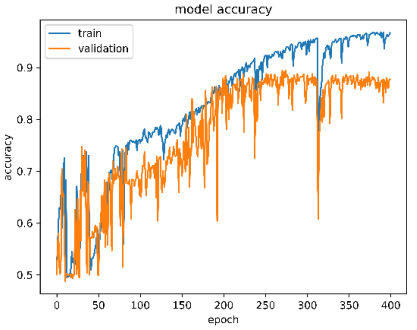
Table 2, 3 and 4. show a few comparisons:

Table 2. Time series length.

Input shape	(200,13)	(500,13)	(800,13)
Model structure			
Loss			
accuracy			
Comments: According to the loss values, different degrees of overfitting occurred in the later stage of the training of the three models. With the increase of the input data time span, the overfitting time also appeared later. Longer time series also take longer to train, and the accuracy of the validation set increases with the increase of step size. Extrapolating from these three sets of data, 500 to 800 is the optimal time series length.			

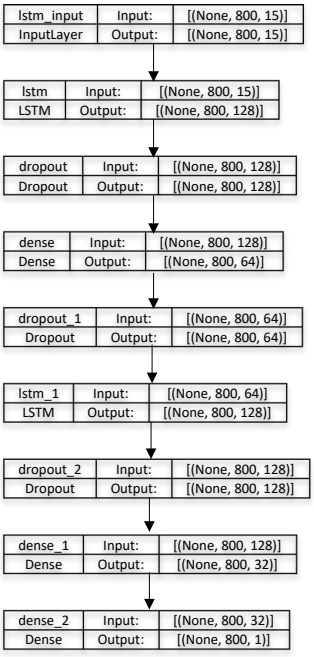
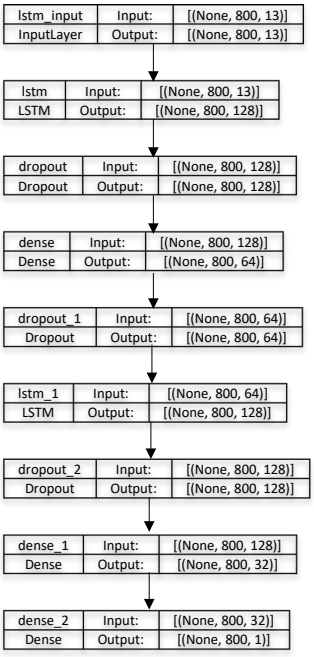
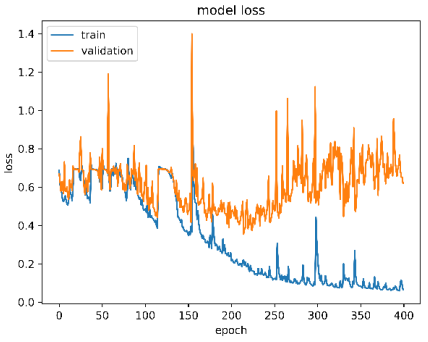
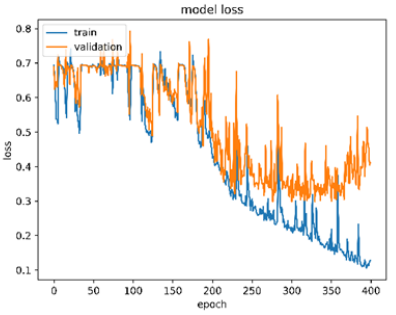
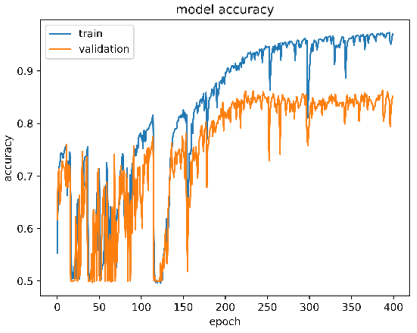
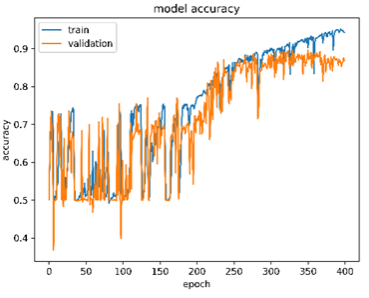
DOI: 10.54254/2753-8818/19/20230523

Table 3. Neuron number.

Neuron number	-less	+more
Model structure		
Loss		
accuracy		
Comments: Different degrees of overfitting occurred in the later stage of the training, there was no significant difference		

DOI: 10.54254/2753-8818/19/20230523

Table 4. Different input.

Different inputs	Keyboard inputs	Player's coordinates
Model structure		
Loss		
accuracy		
Comments: When a set of continuous data (Player's coordinates) is replaced by discrete data (Keyboard inputs), the training effect becomes worse, overfitting occurs earlier and the accuracy of validation set decreases.		

DOI: 10.54254/2753-8818/19/20230523

The validation set accuracy was largely maintained at approximately 85% after training using a convolutional neural network (CNN) with optimized parameters. However, after adjusting the parameters related to the optimizer and implementing a learning rate decay strategy, the accuracy plateaued at around 75%. This resulted in a final accuracy of approximately 75%, which was lower than that obtained using a LSTM model.

6. Conclusion

In the process of using deep learning method to analyze user behavior to make correct user identification, We find that when the dimension of input data is low, the accuracy of CNN is significantly higher than that of LSTM.

However, when the data dimension is increased, such as 13 and 15 dimensions, the classification accuracy of LSTM is significantly improved, reaching nearly 87% accuracy. At the same time, we also compare and analyze the results of model training under different parameters, in order to better optimize the player classification model in the future.

This project has fully demonstrated that different players do have different behavioral characteristics in games, and we can make classification with high accuracy by analyzing these characteristics using neural networks.

Reference

[1] Irdeto. Grand Theft Gaming 2.0. https://resources.irdeto.com/media/e-book-grand-theft-gaming-2-0-1?page=%2Fwhite-papers-e-books-reports&widget=61a00432c044d513b464dac5

[2] J. P. Pinto, A. Pimenta and P. Novais, "Deep Learning and Multivariate Time Series for Cheat Detection in Video Games," *2021 IEEE 8th International Conference on Data Science and Advanced Analytics (DSAA)*, 2021, pp. 1-2, doi: 10.1109/DSAA53316.2021.9564219.

[3] H. Alayed, F. Frangoudes and C. Neuman, "Behavioral-based cheating detection in online first person shooters using machine learning techniques," 2013 IEEE Conference on Computational Inteligence in Games (CIG), 2013, pp. 1-8, doi: 10.1109/CIG.2013.6633617.

[4] Politowski, Cristiano & Guéhéneuc, Yann-Gaël & Petrillo, Fabio. (2022). Towards Automated Video Game Testing: Still a Long Way to Go.

[5] S. F. Yeung, J. C. S. Lui, Jiangchuan Liu and J. Yan, "Detecting cheaters for multiplayer games: theory, design and implementation," CCNC 2006. 2006 3rd IEEE Consumer Communications and Networking Conference, 2006., 2006, pp. 1178-1182, doi: 10.1109/CCNC.2006.1593224.

[6] Etheredge, Marlon & Lopes, R. & Bidarra, Rafael. (2013). A generic method for classification of player behavior. AAAI Workshop - Technical Report. 2-8.

[7] Zhou, ZH. (2021). Model Selection and Evaluation. In: Machine Learning. Springer, Singapore. https://doi.org/10.1007/978-981-15-1967-3_2

Appendix

Code

CNN

```
def train_test(reframed):
    # split into train and test sets
    values = reframed.values
    length = len(values)
    train = values[:int(length*0.7), :]
    test = values[int(length*0.7):, :]
    # split into input and outputs
    train_X, train_y = train[:, 1:], train[:, 0]
    test_X, test_y = test[:, 1:], test[:, 0]


    newtrain_Y = []
    for item in train_y:
        if item == 1:
            newtrain_Y.append([1, 0])
        else:
            newtrain_Y.append([0, 1])

    newtest_Y = []
    for item in test_y:
        if item == 1:
            newtest_Y.append([1, 0])
        else:
            newtest_Y.append([0, 1])
```

```
train_x = train_x / 20
test_x = test_x / 20
```

DOI: 10.54254/2753-8818/19/20230523

```
def fit_network(train_X, train_y, test_X, test_y):
    model = Sequential()
    model.add(Conv2D(32, (3,1), padding='same', input_shape=(25,1,1)))
    model.add(Activation('relu'))
    model.add(Conv2D(64, (3,1)))
    model.add(Activation('relu'))
    model.add(Flatten())
    model.add(Dense(128))
    model.add(Activation('relu'))
    model.add(Dense(2))
    model.add(Activation('softmax'))
    # optimizers.Adam(learning_rate=3e-5, beta_1=0.9, beta_2=0.999, epsilon=None, decay=3e-5, amsgrad=False)
    model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
    # program module
    history = model.fit(train_X, train_y, epochs=20, batch_size=32, validation_data=(test_X, test_y), verbose=2,
                        shuffle=True)
```

LSTM

```
def fit_network(train_X, train_y, test_X, test_y):
    model = Sequential()
    model.add(Conv2D(32, (3,1), padding='same', input_shape=(25,1,1)))
    model.add(Activation('relu'))
    model.add(Conv2D(64, (3,1)))
    model.add(Activation('relu'))
    model.add(Flatten())
    model.add(Dense(128))
    model.add(Activation('relu'))
    model.add(Dense(2))
    model.add(Activation('softmax'))
    # optimizers.Adam(learning_rate=3e-5, beta_1=0.9, beta_2=0.999, epsilon=None, decay=3e-5, amsgrad=False)
    model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
    # program module
    history = model.fit(train_X, train_y, epochs=20, batch_size=32, validation_data=(test_X, test_y), verbose=2,
                        shuffle=True)
```