Horse Racing Result Prediction using Deep Learning Neural Networks

Janik Lobsiger
jlobsiger@connect.ust.hk

Mohammad Tahghighi
mtahghighi@connect.ust.hk

Pranay Sood
psood@connect.ust.hk

Abstract

This work focuses on the prediction of Hong Kong Jockey Club (HKJC) horse racing results using Deep Learning Neural Networks. We decide to collect a large dataset from the official HKJC website and provide a tool to directly download the most current data. Furthermore, we tackle the problem of using various deep learning architectures, including a simple MLP regression method, a CNN to account for spatial relationships, as well as an RNN to model temporal dependencies along with sequential historical data. The results of "win" and "place" accuracy are shown for all the different architectures that have been tested and implemented.

1. Introduction

Sports prediction has become an emerging field in football, athletics, horse racing, etc. This has led to the rise of various commercial services, which have established sports analysis, betting and prediction as their main business model [8]. In Hong Kong, horse racing is considered as a popular and enjoyable form of entertainment of betting between the general community and betting professionals. A large amount of races take place in Sha Tin and Happy Valley racecourses with a wide range of distances. In each race, between 8 to 14 horses with their respective jockeys compete against each other to determine which horse is the fastest along with other positions that match the unique betting options that are available.

Predicting horse races shares similar characteristics to stock market estimation, whereby historical attributes and features along with current predicaments and situation plays a significant role on the performance of the horse during the race. This is where *Artificial Neural Networks* (ANNs) is introduced as a general method of prediction.

ANNs are inspired by brain modeling studies and, given a large dataset, is able to solve complex problems with high efficiency. ANNs have a wide range of applications that comprise classification, pattern matching, data mining, time series prediction and many more [5]. It is a powerful methodology in deriving predictions on the basis of a historical dataset and the current state in order to determine what will happen. As a result, we have decided to propose a classical *Artifical Neural Network*, a *Convolutional Neural Network* (CNN) and *Recurrent Neural Network* (RNN) in order to classify the performance of a horse.

1.1. Pari-mutuel Betting & Types of Betting

HKJC is a non-profit organization and a government granted monopoly that specializes in pari-mutuel betting on horse racing. Pari-mutuel betting is a betting system in which the stake of a particular bet type is placed together in a pool, and the returns are calculated based on the pool among all winning bets [6]. For our project, we will be focussing on the two most popular bets that are done at horse racing events which are "win" and "place" bets with the details provided in Table 1

Table 1: Description of the Win and Place Bet

Bet Type	Performance
Win	1st position in the race
Place	Either 1st, 2nd or 3rd position in the race

1.2. Outline

The remainder of the report is organised as follows. Section 2 discusses the related work in the field of horse racing prediction. Section 3 emphasizes on data collection and preprocessing. Section 4 discusses the methodology and details of the ANN, CNN and RNN architectures that have been implemented and tested. Section 5 will evaluate the results from the different architectures that we have obtained and finally, Section 6 will conclude the report.

2. Related Work

Many researchers have explored the implementation of Deep Learning in sports result prediction and the sports betting market. ANNs are considered to be the most commonly applied approach among the machine learning mechanisms that have been implemented for sports results prediction as discussed by Bunker and Rory [4]. The effectiveness of ANNs comes mainly from its non-linearities in the hidden layers by adjusting weights to influence to the final decision making [4]. Therefore, researchers decided to explore the power of ANNs in order to predict horse race results.

Before discussing the implementation of neural networks for horse racing prediction, Silverman [10] applied the Gibbs model in order to make a prediction on the speed of the horse with the ideology that the fastest horse would win a majority of the races. However, this was not the most accurate hypothesis as the horse with the fastest predicted speed only won 21.63% of the races.

Davoodi and Khanteymoori [5] implemented ANNs to predict the result of the horses whereby one ANN was used for each individual horse in order to obtain its finishing time. The features used were horse weight, type of race, horse trainer, horse jockey, number of horses in race, race distance, track condition and weather. Their network architecture consists of an input layer with eight nodes, two hidden layers and an output layer and was trained using the MSE loss. The paper further explores different training algorithms: gradient-descent (BP), gradient descent with a momentum parameter (BP), Levenberg-Marquadt (LM), Quasi-Newton and conjugate gradient descent (CGD) [5][9]. Based on [5], the BPM (momentum parameter of 0.7) and BP algorithms trained at 400 epochs were the most effective, claiming to predict the winner 39 out of 100 races in their testset. Meanwhile, Quasi-Newton predicted 35 winners, CGD predicted 32 winners and LM only predicted 29 winners. Therefore, both BP and BPM were the best at predicting the winners however, the major disadvantage with BP algorithm is that the training time duration was quite lengthy, meanwhile LM was the fastest. Furthermore, the paper mentions that the used testset consists of only 100 races from a very short duration from 1 January to 29 January 2010, thus testing it on a very small dataset which may often result in higher accuracy. It fails to consider testing its algorithms on larger dataset.

Pudaruth [9] explores the possibility of the ANN approach in order to predict the winners at the Champs de Mars horse racing track. Their multi-layer perceptron comprises one hidden layer containing five nodes and a zero-based log-sigmoid activation function in order to predict the output [9]. The training was done on the first 41 meetings on a total of 347 races meanwhile the testing was carried out on 16 races from the last two meetings [9]. The input data used are just the winning odds, weight, draw, position of jockey, time factor and distance which are fed into the neural network. At the end, the paper had a success rate of 25% in terms of predicting the winner.

A research done by [7] uses historical heart rate variability omegametry procedures of sprinters as inputs for a RNN architecture to forecast the performance of the sprinters. As a result, a final year project from The Chinese University of Hong Kong [6] decided to explore the prediction of horse racing result with machine learning through the implementation of RNN via a vanilla LSTM. However, the results from the paper are not quite satisfactory as their set2sequence model sometimes predicts the same horse on duplicate places in the same races [6].

In our case, we decided to implement an ANN for the HKJC race prediction with more input features which was available from the HKJC website as we believe that more features will have an impact on the performance of the horse. Furthermore, we were able extract a large amount of data from 2015 to 2019 from the website thus allowing us to test our networks with a larger dataset in comparison to most of the literature discussed above. Finally, we also propose and test with CNN and multivariate LSTM architectures in order to predict the finish time of the horses as, to the best of our knowledge, there have not been any experiments conducted on these architectures based on the pre-existing literature in the horse racing prediction field.

3. Data

3.1. Data Collection

Apart from choosing and implementing a suitable architecture, arguably the most important step in Deep Learning is collecting a good data set to train the model on. Even though there is some data freely available online (e.g. on *Kaggle*), these sources usually only offer information from a single racing season and often exclude many relevant features. For this reason, we decided to collect our own dataset. The HKJC website is surprisingly transparent with past racing data, providing a wide range of pages filled with statistics. For our purposes, we mainly consider two of these pages: *Race Cards* [1] and the *Results* [2] page. The Race Cards are available prior to each race and thus provide the data to be used for training. In particular, for each race day, HKJC provides per-race tables including information ranging from the last 6 places of a horse, its weight, starting position, race length, a rating given by the jockey club, etc. Please refer to Table 2 for all downloaded features. Additionally, the race cards provide direct links to statistics about the jockey and trainer of the respective horse. We use this to add information about the past racing history for each jockey and trainer (how often their horses *win* or *place* in a race) to our dataset. This is important given the assumption that these play a large factor in a horses success. The Results page provides us with the ground truth and all data necessary for evaluation, namely finish time, place and win odds for each horse on a per-race basis.

Unfortunately, there is neither an official, nor unofficial API available to download data from the HKJC website, so we were forced to build up a website crawler from scratch. For this purpose we used *Python* together with the web-browser automation library *Selenium* [3]. Since it is not possible to download the HKJC website data directly, we have to use Selenium to simulate an actual access to the website, locate the relevant data and extract the html text. As a first step, we had to figure out the race days, as well as locations. For both race cards and results page, the date and location of the race is encoded into the url. This allows us to simply loop through all potential date-location combinations, if a page is found on the server, we inspect it further. These pages are structured as follows: Firstly, it contains links which allow us to loop through every race of a particular race day. Furthermore, for each race, it contains a table with one row per horse in the race, containing all relevant information. So in a next step we need to locate this table within the page. Fortunately, the table is consistently placed for every page, so we can use *xpath*s to select it. Since not all information is displayed by default, we then execute a small script on the opened webpage, which loops through the table and marks every columns as visible. Once all data is visible, we can loop through every row separately and extract its html text. Each row also contains direct links to the stats-pages for the jockey and trainer of each horse. These can be opened by selenium and crawled again using xpaths. We used this crawler to gain access to five years worth of Hong Kong racing data from 2015 to 2019, which resulted in $\sim 43'000$ data-lines, where each one contains 28 features and represents a single horse in a single race. We further use the code later in the final application to download the same features of the current race day to feed directly into our model.

Table 2: Features downloaded from Race Cards

Date	Venue	Race
Horse Number	Last 6 Runs	Horse
Weight	Jockey	Horse Overweight +/-
Trainer	Draw	Trainer
Rating	Rating +/-	Horse Weight
Weight +/-	Age	Season Stakes
Priority	Gear	win% Jockey
place% Jockey	win% Trainer	place% Trainer
Race Length		

3.2. Data Reprocessing

Our dataset, which is crawled from the Internet, has many different columns, as well as various data types and formats. In order to turn this irregular dataset into a processable one, we have to apply multiple preprocessing rounds. This section briefly explains these steps.

Remove anomalous/non-processable data fields Some of the data fields are filled with single or multiple spaces or '-' characters. Therefore, as a very first step, we need to remove them and make the dataset uniform. These anomalous data fields are later filled with some meaningful values (e.g. nulls) so that they can be processed by our proposed network architectures.

Unification and enumeration of string data fields String data fields cannot be fed into the neural network as such networks can only process data in numeric format. Therefore, we have to enumerate the string-type data fields by assigning unique numbers to distinct strings. This process is done in two steps: We first construct dedicated dictionaries corresponding to the columns with string-type data. Dictionaries are used to map strings to unique integer values. In the second step, we go through various columns individually and replace strings with their corresponding integer values using the dictionaries built in the earlier steps.

Convert date and finishing time to numeric fields In our dataset, race-date and finishing-time are reported in ”yy/mm/dd” and ”minute:second:milliseconds” formats respectively. However, such data formats cannot be processed by the neural network, since addition and multiplication operations are not defined on these formats. Hence, we convert these data fields into numeric values. Specifically, date fields are substituted by the number of days since 1970/1/1 until the value of the date field. Finishing time is converted into an integer representing the time in milliseconds accuracy.

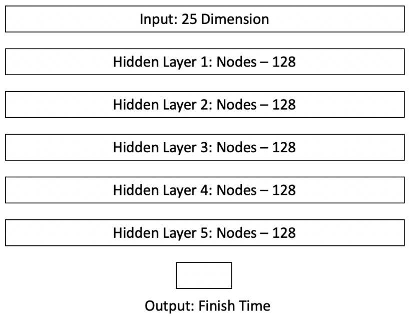
Split up concatenated data fields Some data fields in our crawled dataset hold concatenation of multiple data values and form a single string-type field. An example is the 6 last running places which is presented as a string of ”A/B/C/D/E/F” format (each letter is an integer presenting the rank of a horse in the past 6 races). For such columns, we extract these distinct values and add them as separate data columns to the dataset.

4. Methods

4.1. ANN finish time regression

A first observation after an evaluation of our dataset is the apparent amount of noise in the finishing place of an individual horse. Trying to predict these places could lead to a rather unstable model. Furthermore, to directly predict a race winner, one would have to feed data of an entire race into the Neural Network. It is not always obvious how this should be achieved. A much more stable observation over time is the finishing time of a horse. It can also be predicted by simply providing information about a single horse at a time to the Neural Network, since it can be estimated independently of the other horses in a race. These finishing times can then be predicted for every horse in a particular race and sorted in descending order which gives our estimated outcome of that race. We provide this simple baseline ANN to compare against following, more sophisticated architectures. For implementation *Python* and *Tensorflow* with the *Keras* library is used. We experiment with different architectures by applying grid search on the amount of nodes in the hidden layers, activation function, dropout rate, as well as optimization parameters such as loss, batch size and learning rate. As an evaluation metric, we use the Mean Absolute Error (MAE) between finishing times. Our final architecture consists of 5 hidden layers with 128 neurons each as shown in Figure 1 with no dropout and ReLU activation function. For optimization we use the Adam Optimizer with MAE loss, batch size of 32 and a learning rate of 0.01.

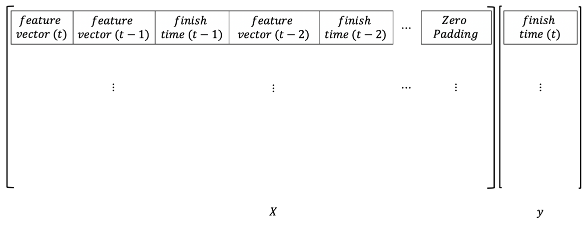
Figure 1: *ANN Architecture*


Input: 25 Dimension
Hidden Layer 1: Nodes – 128
Hidden Layer 2: Nodes – 128
Hidden Layer 3: Nodes – 128
Hidden Layer 4: Nodes – 128
Hidden Layer 5: Nodes – 128
Output: Finish Time

4.2. Temporal ANN finish time regression

The model in the last subsection only looks at one line of data independently. However, our dataset has the advantage of including a large number of sequential datapoints. Since each horse runs multiple races in Hong Kong, we have a sequence of statistics of each horse, i.e. how its attributes like weight and rating develop, as well as its finish time over past races. For this reason, we introduce another method, which makes better use of the availability of sequential data. While the actual Neural Network architecture stays the same (see Figure 1), we modify the the input matrix X as follows. We look at each line in X, which used to simply consist of a single feature vector (i.e. data about one horse for one race), then we go back and look at the past n races of that horse from our dataset. We then concatenate all these past feature vectors and finish times, together with the usual feature vector. If a horse has participated in less than n races, we zero-pad the rest of the row. This results in the input matrix shown in Figure 2

Figure 2: *Temporal ANN Input*



4.3. CNN-based architecture

Convolutional Neural Networks are proven to work effectively in image and video processing applications mainly due to intrinsic spatial correlations among data items (i.e. pixels of an image) in such datasets. Such spatial correlations can be efficiently captured using convolution kernels. Having this fact in mind, we developed a CNN based architecture for our horse race winner prediction problem.

The idea behind our proposed architecture roots in the fact that horses of the same race can affect each others performance. Therefore, we claim that there are spatial correlations among the horses and we try to correlate the performance of an individual horse with its neighbors, which are lined up for the same race. According to our hypothesis, horses can be influenced more by their direct neighbors, i.e. those that are physically closer to them. Hence, we aim at using a CNN to capture the spatial correlations between horses to predict the winner of a race.

Every horse in a race (indexed by 1 to 14) can potentially be the winner and a race has just one winner. Based on these facts, the winner prediction problem can be reformulated to a race categorization problem so that, for every race we assign it to the category which corresponds to the index of the winner horse. Therefore, races labeled with category 1 are those which horse number 1 was the winner of the race, similarly for category 2 includes all the races where horse number 2 was the winner and so on.

In the following, we explain further on how our CNN-based architecture is constructed, what are the challenges in deploying the architecture and we discuss our design space exploration strategy.

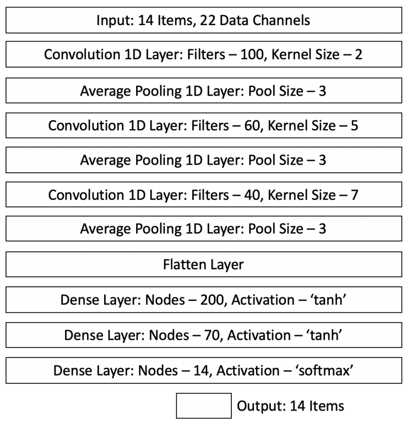
4.3.1 Architecture details

Using the convention explained in the previous section, we can map our original problem to a classical categorization problem. In this section, we take the example of CNN-based RGB image classification to ease the explanation of our proposed architecture.

In RGB images, redness, greenness, and blueness of an image represent separate data channels, which are analysed and combined throughout the CNN stages. The result of this spatial analysis is then passed through some fully connected layers to eventually produce the final output of the network. An image contains 2-dimensional data, therefore, for an RGB image of size 28pixels by 28pixels, the data shape is (28,28,3), with 3 as the number of channels.

In our application, we deal with single dimensional data, since horses form a straight line as they line up at the beginning of a race. The size of the input is 14 items which is the maximum number of horses participating in a race. Similar to RGB images, in our dataset different columns of data convey orthogonal information and should be represented as individual data channels. The number of channels of data in our application equals the number of columns of data used. Instead of blindly using all data columns, we choose 22 interesting ones. Therefore, data channels number is 22 in our application. The order of data fields fed into our proposed network is the same as the order of horses lined up at the beginning of the race. The CNN architecture is displayed in Figure 3. With 14 individual horses and 22 data channels, the data shape of input tensor is (14, 22).

Figure 3: *CNN based architecture details*


Input: 14 Items, 22 Data Channels
Convolution 1D Layer: Filters – 100, Kernel Size – 2
Average Pooling 1D Layer: Pool Size – 3
Convolution 1D Layer: Filters – 60, Kernel Size – 5
Average Pooling 1D Layer: Pool Size – 3
Convolution 1D Layer: Filters – 40, Kernel Size – 7
Average Pooling 1D Layer: Pool Size – 3
Flatten Layer
Dense Layer: Nodes – 200, Activation – 'tanh'
Dense Layer: Nodes – 70, Activation – 'tanh'
Dense Layer: Nodes – 14, Activation – 'softmax'
Output: 14 Items

4.3.2 Challenge in deploying our proposed CNN architecture

Deploying the proposed architecture is challenging because our proposed architecture is designed for races with 14 horses, but the number of horses varies between races: Happy Valley races usually consist of 12 horses, while in Sha Tin, 14 horses compete. Additionally, withdrawals may reduce the horse count further. Furthermore irregularities in the HTML tables on the HKJC website might have caused failures to download all rows in rare cases, resulting in even fewer number of horses in a race. In order to solve the issue mentioned above, one option is to exclude all the races with the number of horses less than 14. However this shrinks our dataset by 1/6 and would exclude all Happy Valley races. Another option is to apply padding on all races with fewer horses. But for a race with two horses requires padding 12 more horses which might significantly reduce our results quality. Eventually, we take a combining approach by removing the races with less than 10 horses first and then applying the padding on remaining ones.

4.3.3 Design Space exploration and network parameter tuning

Designing an efficient neural network requires adjusting many architectural parameters carefully. The number of layers of CNN, convolution kernel sizes, type of activation functions and type of pooling layers are some examples of these architectural parameters. In the context of image processing, there are so many available publications as base lines that give clues on how to tune the architectural parameters to achieve the best performance. But unfortunately, we could not find any similar work to borrow the ideas from. Therefore we take the brute force approach and simply try out many possibilities for each parameter. This is a very time-consuming process since changing a parameter requires retraining the network and perform an evaluation to observe its impact. For the number of convolution layers, we tried 2,3,4 and find out 3 performs the best. For the number of kernels, more than 100 does not improve the performance. We tried sigmoid, relu and tanh as activation functions and finally choose tanh. For the fully connected layers, we variate the number layers and number of neurons and finally choose 2 layers with 200 and 70 neurons in each.

4.4. LSTM based finish time prediction

Another idea we wanted to explore is the impact of historical attributes of the horse on its finish time performance. As a result, we decided to implement a Long Short-Term Memory (LSTM) network in order to predict the "win" and "place" of a race. A multivariate time series forecasting model was implemented as we had multiple inputs (features) for the horses.

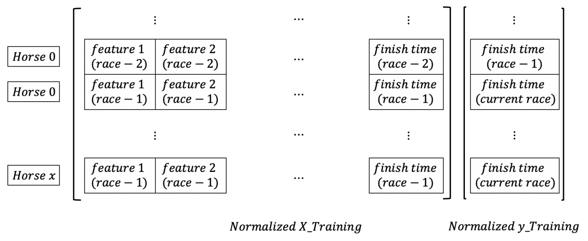
4.4.1 LSTM data preparation

Given our preprocessed data, more processing had to be done in order to make it sequential. The first important aspect is to obtain the horses with historical data. Therefore, any horse that participated in only one race is removed. With this assumption, the data comprises of horses that have completed more than two races. The next step involves framing the dataset as a supervised learning problem and normalizing the input variables. The idea behind framing it as a supervised learning problem is such that we are predicting the finishing time of the horse at the current race given all the features of the horse at the prior race (time) step.

Subsequently, we remove the horse name feature such that we end up with a sequential input with all the essential features of the horse along with their respective finish time as we want to generalize the LSTM model. All the features including the finish time are normalized using a MinMaxScaler with a range of 0 and 1 from *scikit*. Finally, the new modified dataset is split into training and test sets which are then further split into input and output variables. The format of the input and output variables is shown in Figure 4.

Finally, the input is reshaped into a 3D format which is required by LSTMs as $[samples, timesteps, features]$ with $features = 18$ and $timestep = 1$ as the input shape.

Figure 4: *Splitting of Input and Output Variables for training the LSTM*

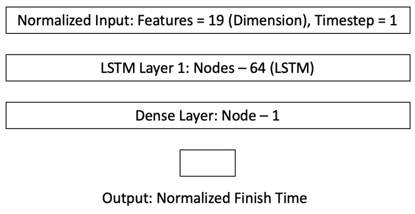


A time step of 1 allows us to know all the historical sequential data of a horse one race at a time.

4.4.2 LSTM architecture details

Since we have a large number of sequential datapoints with 18 features as the input, we decided to define our LSTM model with 64 neurons in the first hidden layer and 1 neuron in the output layer for predicting the finish time of the horse as shown in Figure 5. For optimization, we use the Adam Optimizer with $learning\_rate = 0.001$, $beta\_1 = 0.9$ and $beta\_2 = 0.999$ along with the MAE loss function. The model is being trained for 50 epochs with a batch size of 1 and in the case of LSTM in *Keras*, there is a reset at the end of each batch. In addition, the training and test loss are kept track of by setting the $validation\_split = 0.1$. There is no shuffling of the data involved.

Figure 5: *LSTM Architecture*



Normalized Input: Features = 19 (Dimension), Timestep = 1
LSTM Layer 1: Nodes – 64 (LSTM)
Dense Layer: Node – 1
Output: Normalized Finish Time

4.4.3 Assumptions and challenges with LSTM architecture

When testing the LSTM architecture, the idea was to include the specific horse, its respective jockey and trainer which were originally denoted as strings and converted to integer values. When these three features were fed into the LSTM network, the model was trained successfully. However, upon doing the test, the model was unable to predict the finish times as there were new horses with new trainers and jockeys were being introduced and hence the LSTM model was unable to predict the finish time. Hence, we decided to develop a generalised LSTM model as a result with the exclusion of the horse, jockey and trainer details.

Another assumption that was made was to exclude horses that did not have any historical data. This is because if a horse does not have any historical data, it is difficult to derive sequential inputs for the LSTM model. Zero padding was being considered however, there was a huge disparity in the historical data of the horses in the dataset. Some horses have done less than 10 races, meanwhile other horses have done more than 30 races, so the incorporation of zero padding would have had a significant impact on the result on the prediction of the finish time.

Finally, to normalize the input data, a decision had to be made between either the implementation of the StandardScaler or MinMaxScaler from the *scikit* package. Upon evaluating the MAE loss and Validation Accuracy loss, the MinMaxScaler was used as it had a much lower loss in comparison to the StandardScaler.

5. Experiments

5.1. Evaluation

One important question is how a trained model should be evaluated. For models where finish times are predicted, an obvious first metric is to report the MAE. However, this might not be a very representative evaluation method, since in the end we are interested in a relative ordering and not absolute finish time of each horse. For one, we are interested in accuracy, i.e. we would like to predict as many winners as possible correctly. On the other hand, it is important to note that high accuracy is not the ultimate goal of such a predictor. Since its intended use is to make money by betting on these races, it is most important to maximize the expected money gain. For this reason we evaluate our results in two parts: Our models are tuned to maximize accuracy, thus we also report how well this goal is achieved by providing the fraction of times we predict the winner correctly, as well as the fraction of times we successfully predict a top-3 (*place*) spot. Additionally, for our best model, we simulate actual betting by computing the amount of money we would have won by betting on each predicted winner in the races of our test set, taking into account winning odds.

5.2. Baseline SVM

Throughout this project, we are almost exclusively considering deep-learning methods to approach our problem. However, it is not given that this is the best way to solve this challenge. For this reason, we provide a simple experiment using SVMs. The setup is similar to 4.1, i.e. we perform regression on the finish time, then predict places by sorting the race by our estimated times. We use the *scikit* SVR with a Gaussian kernel. The results are summarized in table 3. We additionally experimented with applying PCA on the input matrix, before the data is fed to the SVM. However, this lead to no observable improvements in performance.

Table 3: Results for *SVM*

Bet Type	Accuracy
Win	11.76 %
Place	30.84 %

5.3. ANN finish time regression

In this section, we summarize the results of the architecture described in 4.1. First we split our dataset in training- and testset. To keep whole races together, such that we can later evaluate winner-prediction accuracy, we split the most recent 255 races ($\sim$ 3000 lines of data) into our testset. The model is trained on the remaining $40'000$ lines, from which another 10% are used for validation. It is important to note, that there are many different race lengths, ranging from 1000m to 1800m. Obviously, the finish time varies between them. We do not normalize the finishtimes by racelengths, instead we directly provide the length as a feature to our Neural Network, letting it figure out a distinction between different races on its own. When evaluating our trained model using the MAE on finish times, we get an error of 2810 ms, i.e. we misclassify each race on average by about 2.8 seconds. This might seem terrible at first, given that these races are usually decided on a decisecond basis. However, the results fortunately look a lot more promising when we convert our estimations to place predictions. Table 6 shows the accuracy, when we estimate finish time for each horse in a race, sort in ascending order, then predict the winner (resp. top 3) based on this order. With this approach, we can achieve an accuracy of $\sim 36\%$ for winner-prediction on the 255 races in our testset.

Table 4: Results of *ANN Finish Time Regression*

Bet Type	Accuracy
Win	36.09 %
Place	51.24 %

5.4. Temporal ANN finish time regression

This section summarizes the model presented in 4.2, using $n = 4$. Since the evaluation technique is analogous to the last subsection, we will omit the explanation of the process here and go straight into the results. First of all, this architecture achieves an MAE of 1067 ms, which is a lot better than the non-temporal version and demonstrates the effectiveness of incorporating sequential data. Table 5 shows the accuracy of this model. Unfortunately, this is significantly lower. This again highlights that a better absolute finish time prediction doesn't necessarily lead to a more accurate relative ordering between the horses.

Table 5: Results of *Temporal ANN Finish Time Regression*

Bet Type	Accuracy
Win	17.32 %
Place	34.77 %

5.5. CNN finish time regression

The CNN architecture, discussed in earlier sections, is developed for winner prediction. The architecture result shows the winner correct anticipation of 30.51% which is 6% less than that of our naive ANN based architecture. While this observation is against our initial expectation we have some explanation for that. The first reason for lower performance could be the number of data samples. For the CNN based architecture, we have to first pack the data-rows into race information. This leads to reduction of data units by a factor of 10. The smaller dataset indeed results in a less accurate model. The second reason can be the amount of padding inserted. As we explained in an earlier section, we have to zero pad many rows into our dataset (30% of whole data) for adjusting the input data shape. While applying the proposed architecture in our current dataset did not outperform the naive ANN based results, we believe when a larger dataset is used our proposed CNN architecture should beat its competitors, i.e ANN based architecture.

Table 6: Result of *CNN based Architecture*

Bet Type	Accuracy
Win	30.51 %

5.6. LSTM finish time prediction

The dataset has been split into training and testing and, similarly to the ANN finish time regression, the most recent 255 races have been used for the testset. The LSTM architecture, as discussed previously, achieves a MAE of around 11155 ms which is quite poor in comparison to both the temporal and non-temporal ANN. Table 7 shows that the accuracy is slightly better than the results of the temporal ANN finish time regression, thus highlighting that effectiveness of incorporating sequential data through a LSTM model. However, with the removal of horses that do not possess historical data, this has significantly reduced the size of the dataset thus suggesting that using LSTM is not the most ideal model in terms of making predictions for the finish time of the horses. In addition, LSTM is much more ideal in dealing with time series or text classification problems where there is a continuous flow of data at regular intervals. With the case of horse racing prediction, there is no continuous historical sequential data for the horses and as a result, this makes it difficult to get an accurate prediction of the finish time as shown by the high value of the MAE.

Table 7: Results of *LSTM Finish Time Prediction*

Bet Type	Accuracy
Win	19.91 %
Place	38.16 %

5.7. Final Application

Since the simple ANN architecture from section 4.1 achieved the highest accuracy, we use this model in our final application and provide a quantitative, as well as a qualitative experiment to demonstrate its power.

Betting simulation To put the results of 5.3 into perspective, we compare to a strategy, where one would in every race trust on common knowledge, i.e. bet on the horse with the lowest winning odds. On our testset, this would give us an accuracy of 31.76% of predicting the winner in each race. This is only about 5% lower than our Neural Network. This raises the question, if it is even worth it building this predictor. The answer is yes! As a next step we illustrate the amount of money each strategy would win. For this, we imagine betting 10 HKD on each predicted winner of the 255 races in the testset (i.e. we bet a total of 2550 HKD). While the odd-based strategy would end in a loss of 594 HKD, our Neural Network predictor surprisingly ends in a win of 1262 HKD. This means, in expectation over our testset, our network results in a win of 1.5 times the invested money.

Real world test After the theoretical success of our model, we were excited to actually take our predictor to a real-world test. For this purpose, we build a final application in *Python* which reuses the web crawler and the simple regression *Keras* model. The application lets the user input a date, venue and race number. It then uses our selenium code to access the corresponding race card on the HKJC website and preprocesses and normalizes the data. This results in a 2D matrix, where each row represents a horse in the desired race. This matrix can directly be fed into our model, which predicts the top 3 horses. We used the application to bet on several races on the Happy Valley racecourse. By using the strategy of placing a win bet on all predicted top 3 horses, we managed to win money almost every time and ended with approximately doubling our investment. This empirically confirms the effectiveness of our Deep Learning Neural Network horse racing predictor.

6. Conclusion

One thing we realized, against our expectations, is that temporal models, i.e. architectures which take into account sequential data are not as effective in solving the race winner prediction problem. Neither the naive temporal ANN (4.2), nor the more sophisticated LSTM approach (4.4) outperformed any of our non-temporal methods. Explanations for this could be the lack of sequential data for many horses and the fact that the race length varies from event to event. The next architecture we tested, namely the CNN, seems to be a lot more reliable in predicting the correct winner, which proves that our hypothesis that the spatial relationship between horses in a race play a large role in deciding the winner. While we only implemented this architecture for win bets, one possible future work could go into the direction of estimating the ranking of a whole race, e.g. also tackling the place-bet problem. Finally, and most surprisingly, the simplest model, namely the ANN regression model (4.1) performed the best. Its accuracy of predicting the winner correctly is around 36% and gives an expected positive return on our investments which makes it a powerful tool in the horse racing field.

Since we have tested a wide range of different architectures during our study and were able to collect a detailed understanding of how to build a Neural Network for this task, future work of this project will mainly focus on the dataset. First of all, we have a tool set up, which allows us to easily download HKJC racing data and can be used to download even more statistics to make our dataset bigger. Additionally, there is a lot more data for horses, trainers, jockeys, etc. available allowing us to include many additional features, making the dataset even more powerful.

References

[1] Race Card. `https://racing.hkjc.com/racing/info/meeting/Racecard/English/Local`.

[2] Results. `https://racing.hkjc.com/racing/information/english/racing/LocalResults.aspx`.

[3] Selenium. `https://selenium.dev/`.

[4] R. P. Bunker and F. Thabtah. A machine learning framework for sport result prediction. *Applied computing and informatics*, 2017.

[5] E. Davoodi and A. R. Khanteymoori. Horse racing prediction using artificial neural networks. *Recent Advances in Neural Networks, Fuzzy Systems & Evolutionary Computing*, 2010:155–160, 2010.

[6] Y. Liu. Predicting horse racing result with machine learning. 2018.

[7] K. D. Peterson. Recurrent neural network to forecast sprint performance. *Applied Artificial Intelligence*, 32(7-8):692–706, 2018.

[8] D. Pettersson and R. Nyquist. Football match prediction using deep learning. 2017.

[9] S. Pudaruth, M. Jogeeah, and A. K. Chandoo. Using artificial neural networks to predict winners in horseraces: A case study at the champs de mars. In *Proceedings of the 10th IST-Africa Conference*, 2015.

[10] N. Silverman. *Optimal decisions with multiple agents of varying performance*. PhD thesis, UCLA, 2013.