REP_TraceyP_DS

by PAUL TRACEY

Submission date: 17-Sep-2018 03:07PM (UTC+0100)
Submission ID: 92323265
File name: 101328_PAUL_TRACEY_REP_TraceyP_DS_442690_1606471551.pdf
Word count: 13316
Character count: 68081

Birkbeck, University of London

Department of Computer Science and Information Systems

Predicting the outcome of horse racing using artificial neural networks

- Paul Tracey -

- MSc Data Science 2018 –

- Supervisor: Dr Alessandro Provetti -

Abstract

Horse racing is one of the most popular spectator sports in the UK and Ireland, with betting on the outcome of races a major component of its popularity. This investigates how Data Science techniques such as artificial neural networks can be used to analyse the growing amount of horse racing data being recorded to better predict the outcome of horse races, with a potential view to assist the assessment of betting odds. The model and the results obtained (for winner predictions) suggest that using such networks might be of use in horse racing predictions, but its deployment in a real-life scenario suggests that much work remains to be done. Surprisingly, only three academic papers devoted to horse race forecasting using neural networks were found in the literature; this points to a possible lack of interest in adequately investigating and formalising the use of machine learning to improve what is known to be a very difficult skill: the accurate prediction of horse races.

Contents

Acknowledgements

I would like to thank Malcolm Smith of ukhorseracing.co.uk for kindly granting me access to his data archive to allow me to complete this project.

1: Introduction

1.1: Horse racing as an industry

Horse racing is the second largest spectator sport in the UK according to a recent study by Deloitte[1] which also found that the industry generates over £3.7bn to the British economy through things such as ticket sales, television rights and supporting tens of thousands of jobs. Much of that money generated is through gambling on the outcome of races, with £12.1bn being bet annually on horse racing in Britain. This puts the British second only to the Japanese (£14bn annually) in their interest in betting on horse racing. There are over 1,000 horse racing meetings each year[2], with up to as many as 40 horses in a single race, meaning that each year hundreds of thousands of datapoints are being created which could be used as analysis tools.

1.2: Types of horse racing run in Ireland and the UK

There are two main types of racing that takes place in the UK and Ireland, flat racing and jumps racing (known as National Hunt). Flat racing entails horses lining up in stalls and running unobstructed for distances of anything between 5 furlongs (8 furlong = 1 mile) to 2 miles 5 furlongs. Notable flat races include the Derby, the Oaks and the 2,000 Guineas. Jumps racing is subdivided into two categories: chasing and hurdling. Chasing sees horses jumping over stiff, tall fences while hurdling utilises much smaller, more flexible hurdles as obstacles. Jumps races are typically between 2 miles and 4½ miles, with the two most famous jumps races of the year being the Grand National and the Cheltenham Gold Cup. Horses do no line up in stalls for jumps races because, being typically longer than flat races, getting a clean start is less important. Traditionally flat racing has taken place in the summer, and National Hunt racing in the winter, when the softer ground is more forgiving to horses jumping over obstacles, but there are now a number of all-weather (AW) tracks that host flat racing all year round.

Courses vary quite substantially across the country, and the different set-up of a course can favour or disadvantage horses by their running style. As well as courses being either right- or left-handed (clockwise and anticlockwise), they also differ in their topography, with some courses quite flat compared to others that are undulating, and some having sharp, tight corners rather than wider, easier corners. All of these factors can have quite an influence on how a horse performs at a certain track, with some horses being seen as ‘specialists’ of certain courses while performing poorly elsewhere.

Another subdivision of races is between handicap and non-handicap races. Horseracing authorities in the UK and Ireland have a rating system that they use to gauge how highly they rate a horse’s ability, and in handicap races these ratings are used to adjust the weight a horse carries in order to level the playing field to some extent. Handicap races therefore give horses with less ability a chance to win races that they would have little chance of ordinarily. Non-handicap races do not adjust the weight a horse carries according to their perceived ability, and championship races such as the Derby and Cheltenham Gold Cup operate under this system.

Despite not handicapping horses according to ability in these races, authorities to operate a weight-for-age system in some races that sees younger, less well-developed horses carry slightly less weight than their older counterparts to improve their prospects.

Horses can start racing from two years old for flat races, and three years old for jumps races, although individual races have their own qualifying criteria. For instance, the Derby is only open to three-year-old horses that have not been gelded (castrated) as it is seen as the premium championship race for three-year-olds in Britain.

1.3: Betting on horse races

When betting on the outcome of a race, there are two main types of bet: win and each way. A win bet puts the whole stake on a chosen horse winning a particular race. If it comes second or lower, the bet loses. For example, a £1 win bet on a horse priced at 5/1 that wins returns £6, which is comprised of the £5 winnings, and the £1 stake, which is returned for winning bets. An each way bet places half the stake on a horse winning, and half on the horse being 'placed' in the race. There are rules on how many horses are 'placed' in gambling terms depending on the number of runners and the type of race. For races of 1-4 runners, bets are win only, and no places are offered. For races of 5-7 runners, there are two places at ¼ of the win odds. For races of eight or more runners, there are three places at ⅕ the win odds. For handicaps of 12-15 runners there are three places at ¼ the odds, while for handicaps of 16 or more runners there are four places at ¼ the odds. For example, a gambler places a £1 each way bet on a horse at 8/1 in a six-horse race. The bet costs the gambler £2, comprising of £1 for the win, and £1 for the place. If the horse wins, the gamble receives £9 for the win portion of the bet, and £3 for the place portion of the bet (£1 stake returned and 8*¼: £2). If the horse comes second, the gambler loses the £1 win portion of the bet, but receives the £3 place portion of the bet. These are the basic rules for 'singles', which are bets on one horse in one race. A double is a bet that includes two horses in different races which would see the winnings of one race proceed to become the stake of the second bet. An example of such a bet would be a £1 win double on horses that are 5/1 and 10/1 for their respective races. If the first horse wins, £6 carries over to the second horse, and if that wins the gambler receives £66 back. A basic way of calculating the returns from such bets is to multiple (odds1 +1)*(odds2 +1). If either of the horses in such a bet loses, the whole bet loses.

1.4: The mathematics of bookmaking

While the odds of a horse can be loosely seen as an implied probability of that horse winning, the final odds at the start of a race (the 'starting price' or SP) is as much a function of a bookmaker's liabilities as that bookmaker's views on how the race might pan out. To take the most basic example of a fair coin, the probability of the coin landing on heads is the same as the probability of landing on tails: 50%. Thus a 'fair' set of odds for such a coin would be evens (1/1) for both heads and tails. However, a bookmaker would expect to take the same amount of money on both outcomes, as they are equally likely, so to offer odds of evens on heads and

evens on tails would most likely see the bookmaker simply break even. If an equal amount is staked on both outcomes, any winnings the bookmaker makes from the losers would be wiped out by payouts to those who backed the correct outcome. Thus, for a fair coin, a bookmaker would likely offer odds of slightly odds-on for both outcomes to ensure it a profit on the venture. Now consider a fair coin where both heads and tails have the odds of 10/11. Such odds offer an implied probability of 52.38%, meaning that both outcomes offer a combined probability of 104.76%. This 4.76% discrepancy is known in the betting industry as the 'overround', or the bookmaker's expected profit. To finish the above example, being a fair coin a bookmaker would expect to take equal amounts on both outcomes. Imagine a bookmaker takes £20 in bets on heads and £20 in bets on tails, meaning they have £40. If the tail lands on heads, those who backed heads will receive £38.18 in total (their £20 stake plus their £18.18 winnings), leaving the bookmaker with a profit of £1.82.

For horse racing (and other sports that people gamble on) bookmakers will adjust their books as they accept more bets in order to maintain a situation whereby they will be in profit whatever the outcome of the race. Thus in the above situation of a fair coin, if a bookmaker offer odds of 10/11 the pair for heads and tails, but kept taking bets for heads such that it had a large liability should heads come up, the bookmaker would reduce the odds for heads and increase them for tails in order to encourage more bets on tails and even out its liabilities. In such a situation a bookmaker could offer odds of 4/6 for heads and 5/4 for tails as the weight of money forces them to change their odds, even though they know that the likelihood of both outcomes is the same. This is an extreme example that is very unlikely to play out in real life, but it demonstrates that a bookmaker's odds when a race starts are affected as much by the behaviour of the betting public as their opinions on how a race will proceed. An interesting real-world example of such a phenomenon came on 28 September 1996 when Frankie Dettori won all seven races at Ascot. The last race of the day saw Dettori riding Fujiyama Crest to victory. The horse started the day at a price of 12/1, but because he had won the previous six races there was so much money placed on the horse to win that it went off at 2/1 favourite[3].

1.5: Reading a racecard

Programmes at racecourses and the daily newspapers provide racecards for races giving a certain amount of basic information about each horse to help gamblers make a decision who which horse to back. Figure 1 shows a typical entry for a horse entered in a race in the UK and Ireland. A comprehensive list of abbreviations commonly used in horse racing is provided in Appendix 1.

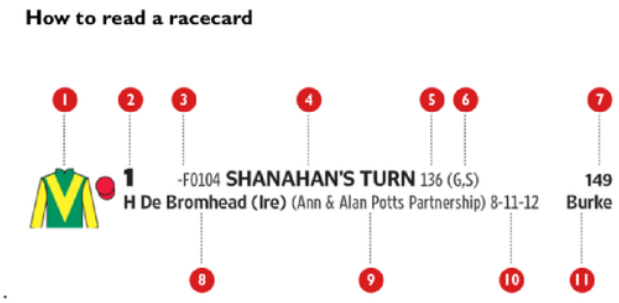


Figure 1: Typical entry for horse on racecard[4]

1. Owner's colours; 2. Racecard number; 3. Six-figure form; 4. Horse's name; 5. Days since latest outing; 6. Conditions and equipment info; 7. Timekeeper's speed rating; 8. Trainer with his/her nationality in brackets; 9. Owner of horse; 10. Age and weight to be carried; 11. Rider plus allowance, which, if any, appears in brackets.

Many people who bet occasionally on horse racing, and who only bet on national occasions such as the Grand National or Cheltenham Gold Cup, will ignore any analysis of previous results or jockey to bet based on colours or horse name. However, there are others who look much deeper than even the racecard and assess a horse's chances based on countless potential influencing factors. It would be impossible to form a comprehensive list of such indicators, but a selection of the most well-known being a horse's previous performances over course, distance and going (condition of the ground). Gamblers can also take other less well-known factors into consideration, such as a trainer's recent form with other horses, the ancestry of a horse or even its preference for left- or right-handed tracks. Indeed, this small snapshot of the regular indicators does not take into account the more nebulous factors that can't be considered so easily, such as a horse's mood on a particular day, or how a jockey interacts with a particular horse.

1.6: High-street bookmakers in the UK

Until 1960 off-site betting on horse racing was illegal, meaning that anyone wanting to place a bet on a race would have to travel to the racetrack to do so. In that year the Betting and Gaming Act was passed[5] legislating for off-course gambling and high-street bookmakers started opening up across the country. By the mid-1970s there were 15,000 such betting shops in the UK[6]. While there were numerous local independent bookmakers across the country, the industry was dominated by the 'big three' of Ladbrokes, William Hill and Coral, a situation that persisted until the advent of the internet, which saw a proliferation of online betting companies. A number of these online-only companies such as Bet365 and Befair have won a significant proportion of the online business, and adding to the strong online presence of the traditional bookmakers, high-street gambling has waned in recent years. The latest figures from the Gambling Commission[7] show that the number of betting shops in Britain stood at 8.532 in March 2018, a 3.2% decrease on March 2017. The advent of widespread online gambling has come in parallel to huge increases in computing power and the ability to collate data and statistics on horse racing.

1.7: Newspaper tipsters

Tipsters have long been an established part of the horse racing industry, with most daily newspapers employing someone to offer their insight into the day's racing. Tipsters usually offer reader their 'Nap' of the day, which is their most confident bet of the day, and often follow it up with an 'NB' (next best), which is their second-best bet of the day. The Racing Post, the horse racing industry newspaper, keeps tabs of how tipsters are faring with their Naps table that uses a level £1 stake for each tipster's Nap to calculate how accurate they are[8]. While tipsters might offer some insight into how they have come to their decisions regarding the day's racing, there is very little discussion of any in-depth analysis that may or may not be being carried out by the tipsters, and in general that has been very little discussion of such a topic in general.

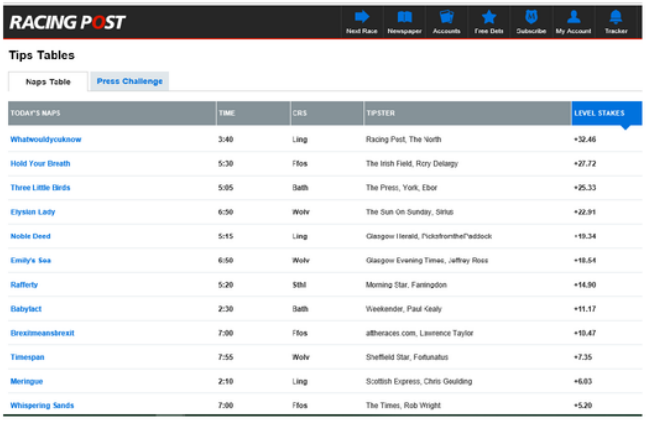


Figure 2: Example of the Racing Post Naps table[8]

As well as the newspaper tipsters, a huge industry has grown of horse racing systems and tips lines, which has increased since the advent of the internet. Some companies simply offer customers their daily tips with little discussion of their methodology for arriving at their conclusions, while others offer customers data that they themselves can manipulate rather than offering tips themselves. One such site, ukhorseracing.co.uk, has kindly allowed me access to their data in order to carry out this project.

1.8: Literature review

There has been relatively little written about the potential use of machine learning techniques such as ANNs for predicting the outcome of horse races. A search of Google Scholar on the subject returned just 346 matches[9], and of these very few were focused on the issue of horse racing, with many papers touching briefly on the sport in the wider context of sport prediction and machine learning.

In a 2010 paper[10], E Davoodi and AR Khanteymoori concluded that ANNs are appropriate for predicting horse racing results, with backpropagation the most appropriate method. Their paper looked at just 100 races at a single racetrack in the US, using eight variables, and found that their network performed with an accuracy of 77% on average. An earlier study by J Williams and Y Li[11] had found similar results, concluding that using a neural network with backpropagation could produce "acceptable results" with 74% accuracy at Caymans Race Track in Jamaica, using 143 races as their dataset.

Another 2013 study[12] by S Pudaruth, N Medard & ZB Dookhun took a similar low-level approach by looking at three race meetings at the Champs de Mars racecourse in Port Louis, Mauritius. Even just using a weighted probabilistic approach using nine variables produced a model that predicted on average 4.7 winners per eight-race card compared with the performance of professional tipsters (3.6 winners). They conclude that using neural networks could further improve the performance of their model. Considering the small datasets and number of features used in these studies, and their relatively successful results, they suggest that further investigation is warranted.

An earlier study looking at the use of neural networks for greyhound racing at Tuscon Greyhound Park in Tuscon, Arizona[13], which is a similar field and which would be of interest here, found that a neural network using backpropagation returned a similar win rate (20%) as human experts (19%), but provided much better returns ($124.80 for a $2 stake) than the humans ($67.60 loss on average).

It should be noted that the lack of published literature might not mean that there has been little research into the area, but only that there has been little academic research. As gambling is a billion-pound industry, it would be very surprising if betting companies had not conducted their own internal, confidential, research into the subject to ensure their models are as accurate as possible.

2: Aims & Objectives

The objective of this project is to construct an Artificial Neural Network that will aim to better predict the result of horse races in the UK and Ireland. Because of the huge amount of data that has been accumulated on horse racing over the past few years, rather than looking at horse racing as a monolith, I will separately consider flat turf and National Hunt racing, as these two disciplines are quite different from each other, as well as looking at AW racing separately. While AW racing is a subset of flat racing, it is considered quite different and horses are often seen as being AW specialists, so it would be interesting to see if considering AW racing as a separate discipline produces any interesting results.

3: Artificial Neural Networks

3.1: Basic structure

While gamblers may do their own analysis of horse races by looking at entrants' previous form, the form of the jockey and trainer, or how a horse has performed previously over course or distance, as the number of datapoints grows it becomes impossible for a person to adequately account for all these factors in making their assessment of the odds.

Hence, it seems natural in Data Science to make use of Artificial Neural Network (ANN) to analyse horse racing data to better predict the winners of races. ANNs are interconnected networks of artificial neurons that mimic the work of the human brain, where large numbers of neurons work collaboratively to process information[13, 14, 15]. One benefit of ANNs is that they can learn organically[16], meaning that they don't require prior knowledge to make connections between inputs and outputs, and can form their own connections and relationships between factors that are difficult or impossible for a human to spot. The basic structure of an ANN is a series of layers of interconnected artificial neurons with each connection able to transmit a signal from one neuron to every other neuron in the following layer.

3.2: Supervised learning

The data that will be used for this project, which includes the results from actual races, means that this is a problem of supervised learning. The data will need to be split into input-output (x, y) pairs, where the input (x) is the collection of datapoints for each horse in a race, and the output (y) is the performance of that horse. In this case, the y value is binary, with a 1 representing a winning horse, and a 0 representing a losing horse. An interesting exercise that could also be done with this dataset is a non-binary investigation, where the y values represent a horse's finishing position, so taking any value from 1 to 40 (and a number of non-numeric values such as F (fell) and U (unseated rider)), but I am limiting the scope of this project to the win/loss result. As well as splitting the data into input-output pairs, the data also need to be split into a training and test set. The purpose of this is for the network to train itself on the training set, using actual results to 'supervise' the learning, and once this is done use the values it has arrived at in its training stage to predict outcomes on the test set of data. This allows the network to test its values on a new set of data to calculate how accurate it is.

3.3: Overfitting and underfitting

This is a vital step in machine learning as it will highlight if the training of the network led to overfitting. If the accuracy of the training phase is very high, but the accuracy of the test phase is not, that suggests that the network has simply learned how to achieve the correct answers in the training data, rather than making real connections between the input datapoints and the desired outcomes. If this is suspected, the parameters of the network during the training phase can be adjusted, such as reducing the number of epochs that the training phase lasts for. In

general, allowing the network run for too few epochs leads to underfitting, while running for too many epochs leads to overfitting. This is because having too few epochs means that the network does not have sufficient time to make meaningful connections between datapoints and the desire output, which having too many means that the network just learns the correct values to reach the desired output for the training data, but loses any ability to generalise this out to other data. Figure 3 shows a demonstration of what overfitting and underfitting might look like on a set of datapoints.

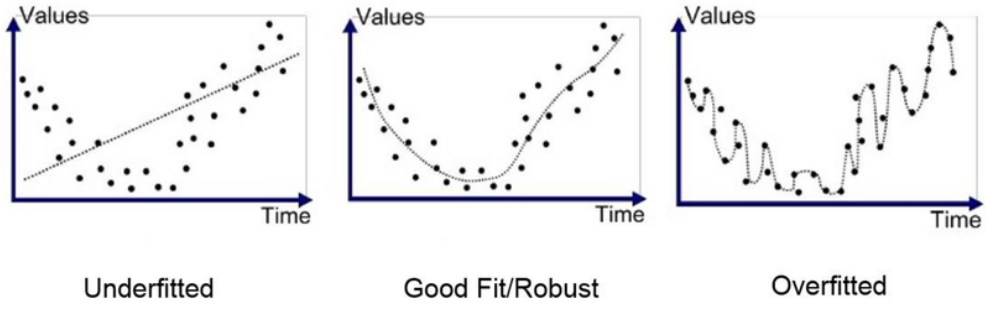


Figure 3: Graphical representation of principles of over and underfitting[17]

3.4: Connection weights and biases

Connections between neurons are instantiated with an initial weight (often random), which allows the network to learn. The weight either amplifies or dampens the strength of the signal, and is adjusted as the network learns. When a neuron receives a signal from another neuron, it calculates the weighted sum of all its inputs and, depending on how the ANN is configured, it decides whether to 'fire' or not. This process continues through the network, with each layer receiving an input from its previous layer, processing this input, and then firing signals on to the next layer of neurons to do the same. A bias term can also be added to this total weighted sum of inputs depending on the purpose of the network such that the neuron calculates its value:

$$\mathbf{N = \sum (weight * input) + bias}$$

3.5: Activation Functions

Each neuron is encoded with an activation function that determines what output it gives depending on the total input signal (N) it receives. This often comes in the form of a threshold function, with the neuron firing if N is greater than the threshold function, and not firing otherwise.

The purpose of non-linear activation functions is to introduce an element of non-linearity to an ANN, as linear functions are limited in their complexity and restrict the ability of networks to model complex problems. Using linear activations on neural networks, no matter how many hidden layers are involved, will only be able to deliver an output that is a linear function of the input layer. This is because each layer would simply be sending a weighted sum of its input to the next layer, which would itself be calculating its output based on a linear function.

The simplest non-linear activation function is a step function, where a neuron has two potential outputs, produce a signal or not, as seen in Figure 3. A neuron decides whether to fire or not depending on whether the weighted sum of input signals exceeds a given threshold. In Figure 3, the threshold is zero, but this can be moved along the x-axis with the use of a bias to increase or decrease the threshold value depending on the needs of the network.

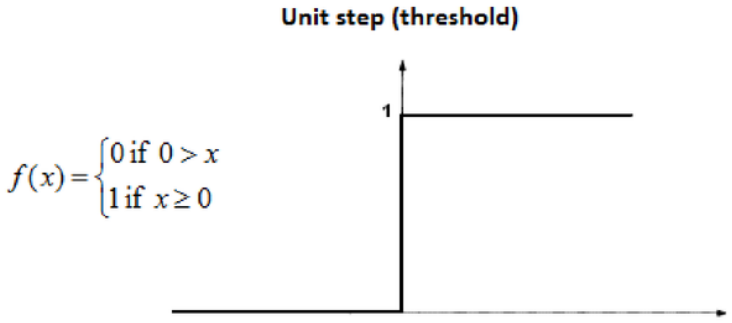


Figure 3: Basic step function[19]

One other widely used activation function in the construction of ANNs is the Sigmoid (or logistic) function, seen in Figure 4. As it exists in the range of 0 to 1, it is particularly useful for models looking to predict the probability of an event occurring. As this project is looking to predict the outcome of horse races, the Sigmoid function will be useful in determining the likelihood of any particular horse winning a race.

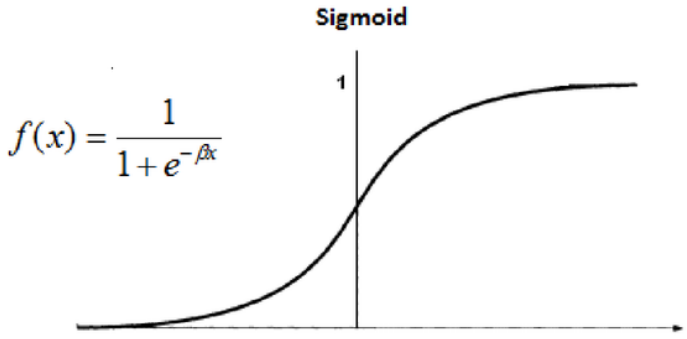


Figure 4: Sigmoid function[19]

The rectified linear unit (ReLU) function, as seen in Figure 5, is a popular activation function for use in hidden layers of ANNs. This is because ReLU was found to accelerate the convergence of stochastic gradient descent compared to the sigmoid/tanh functions[17], possibly because of its linear form. The Sigmoid (and Tanh) functions are not really suitable for use in ANN hidden layers because as their graphs reach their extremes the gradient of the curve tends

towards zero. This will cause the learning of the network to be extremely slow as the weights will be updated with very small values.

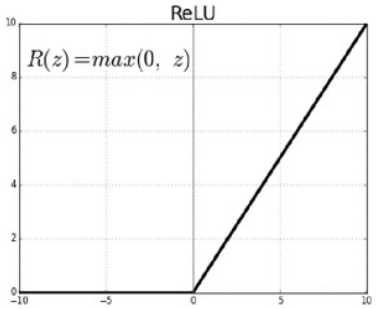


Figure 5: ReLU function[20]

3.6: Multilayer perceptrons

For deep learning, a class of ANNs known as multilayer perceptrons[21] (Figure 6) are used because of their ability to model complex problems with multidimensional data. Multilayer perceptrons can consist of multiple hidden layers of neurons, and make use of backpropagation to train the network. A perceptron is a single neuron that sums up its weighted inputs, applies any bias required and either signals or not depending on its activation function. A multilayer perceptron is a collection of these perceptrons (neurons) into layers to map inputs to outputs.

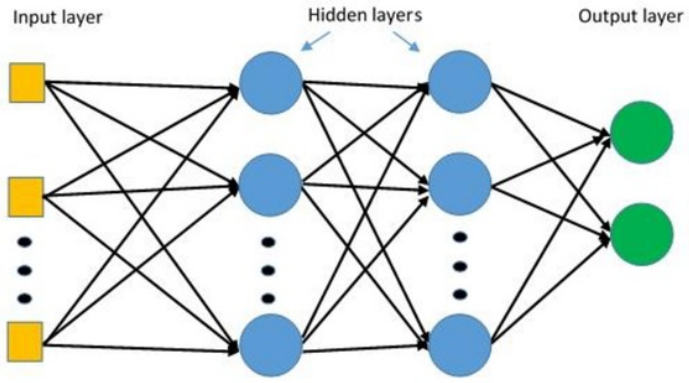


Figure 6: Basic structure of multilayer perceptron[22]

2]]

3.7: Backpropagation

In an ANN, each layer receives an input from the previous layer, calculates its output according to the weights, biases and activation functions, and this output is fed into the next layer. This is known as forward propagation, as the signals travel one way, from input to output. The output provided by the network is our prediction based on the data provided to the ANN. This will differ from the desired output, an error that is calculated by the network. The aim of training the network will be to minimise this error, so that whatever the given input, the network will be able to predict the correct outcome.

After each epoch of the ANN, the error between the predicted and the expected outcomes are calculated, and these are fed back though the network, layer by layer, so that the weights can be adjusted to reduce the final error on each iteration. Thus this system is known as backpropagation, because after the signals are propagated forward through the network, the final errors are then propagated backwards in order to reduce this error. On each epoch of the ANN, this process is repeated, with the new weights producing an output, which is then compared to the desired result. This errors is then fed back through the network, updating each connection weight as it does. Through this iterative process the network should move closer and closer to an accurate prediction based on any given input of data.

3.8: Learning rate

The learning rate of an ANN determines the amount at each epoch by which the weights are changed in order to minimise the error and effectively measures how quickly the network abandons old connections for new ones. A larger learning rate accelerates the speed at which the network learns, but if it is too high then the error can oscillate around the minimum error and never reach the global minimum.

3.9: Dropout rate

Dropout is used in the training of neural networks as a way of avoiding overfitting. When used in ANNs, the dropout figure is a fraction between 0 and 1 and indicates the proportion of neurons that will be ignored during each training epoch. The neurons are chosen at random each epoch, and they play no part in the training of the network in that particular epoch. ANNs are fully-connected networks, meaning that every neuron in a layer is connected to every neuron in the previous and next layers. This can lead to neurons developing dependencies on each other over training epochs, and by dropping a proportion of random neurons during each training epoch, this forces neurons to look at other connections rather than continually falling back on its dependencies.

3.10: Gradient descent

If we were to create a graph plotting the error value against a random weight in the network, it might look something like Figure 7, which shows that setting the weights either too high or too

low results in a high error rate. To correctly train an ANN, what is needed is a minimum error for each weight in the network.

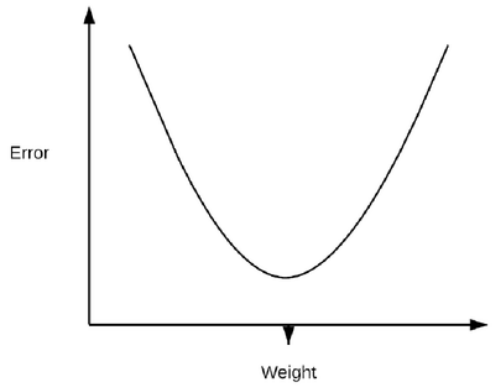


Figure 7: Basic graph of neuron weight and error

Starting at a random weight, the network wants to move towards the minimum error, and does this by taking steps in the direction that is the opposite to the gradient (Figure 8).

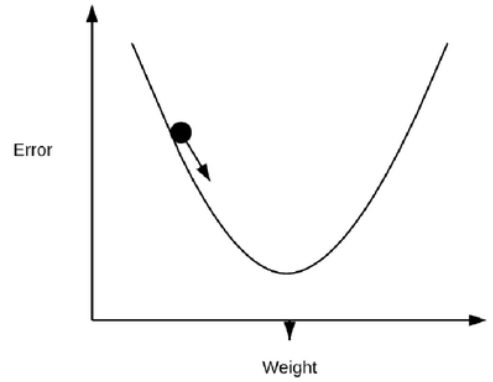


Figure 8: Graph showing gradient descent to reduce error

Over the iterations of the ANN, it will take many steps down the gradient of the slope in the hope that the weight will eventually find the minimum of the error. This process is known as gradient descent.

While gradient descent is a powerful optimisation algorithm, one potential issue is if it falls into a local minimum, which would end the process prematurely before the actual minimum error is found. An illustration of such a situation can be seen in Figure 9, where the process could end prematurely once it falls into the local minimum, and never discover the global minimum.

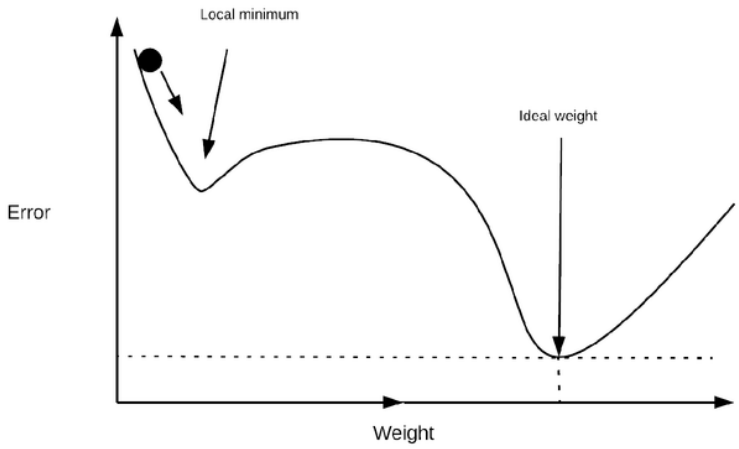


Figure 9: Graph showing potential issue of local minima

There are a number of techniques for avoiding this local minimum pitfall, and they have been steadily updated throughout the years.

3.11: Optimisation algorithms

One such technique is the use of a momentum term, which uses the value of the previous weight update when making the current update to that weight. The term momentum comes as it is analogous to the same term in physics, and the weight vector gains 'acceleration' from the gradient, thus helping to prevent it falling into a local minimum. A potential downside of using a momentum term is that as the weight update gains momentum, and thus more 'speed', it can overshoot the global minimum and again fall into another local minima.

Over the years as the field of machine learning and neural networks has developed, a number of different gradient descent optimisation algorithms have been developed to try to overcome the problems with basic momentum. Russian mathematician Yurii Nesterov refined momentum by adding a term that essentially allowed it to look ahead to where the weight vector would be going to, as a way of speeding up convergence and avoiding overshooting the global minimum[24].

Most of these newer gradient descent optimisation algorithms use momentum with modifications to improve performance. Three algorithms that I will look at in this project are Adaptive Gradient (AdaGrad), Adaptive Moment Estimation (Adam) and AdaDelta, as these are seen as good all-rounders that deliver consistent results.

AdaGrad uses parameter-specific learning rates, which change depending on how frequently a parameter gets updated during training. The more updates a parameter receives, the smaller the updates. AdaDelta is an extension of AdaGrad, but rather than accumulating all past gradients, it takes a moving subset of them so that it can continue to learn even when many updates have

been done[24]. Meanwhile Adam can been seen as a generalisation of AdaGrad and AdaDelta, which are essentially Adam with certain parameters chosen[25].

3.12: Number of hidden layers and neurons

A question to consider when constructing an ANN is to choose the number of hidden layers of neurons. While it might be tempting to keep adding layers with the thought of increasing the network's accuracy, a few things need to be considered. Computing power is an obvious consideration, as the more layers and neurons that are added, the more power is needed, and the longer any network will take to train.

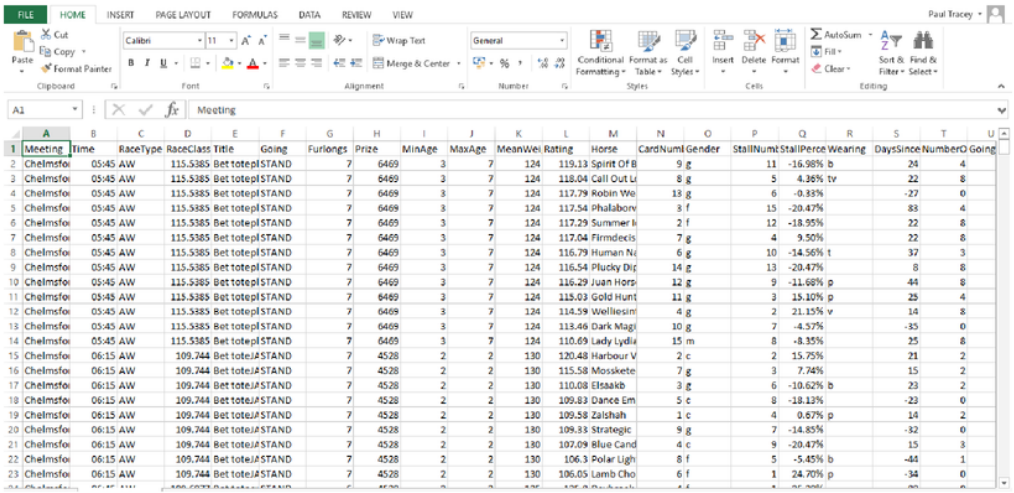
Another question to ask is whether adding extra layers will improve the performance of the network. While an ANN with no hidden layers is only capable of representing linear separable functions or decisions, the universal approximation theorem[27, 28] states that ANNs with just a single hidden layer of neurons can approximate any function. By breaking down any n-dimensional function into a finite number of partitions, the function can be approximated by each partition approximating a small portion of the function. The more partitions there are, the better the approximation. These partitions are analogous to neurons in an ANN's hidden layer, meaning that in general, the more neurons in the hidden layer, the closer the approximation. Thanks to this universal approximation theorem, I will only look at ANNs with one hidden layer, as using more hidden layers are only necessary for more complex problems such as image processing.

This means that choosing how many neurons will be in the hidden layer is another important consideration. Using too few neurons in the hidden layer will result in underfitting, where the function has not been approximated well enough. Conversely, putting too many neurons in the hidden layer can cause overfitting, when the network closely matches the input data given to it, but it is unable to generalise this out to accurately predict correct outcomes when presented with new data. As ever, by increasing the number of hidden neurons, you are also increasing the amount of computational power that will be required, and the amount of time needed to allow the network to properly train. A balance therefore needs to be struck between these two extremes.

4: Data Collection & Preparation

4.1: Data collection

As mentioned in Section 2.7, ukhorseracing.co.uk is a subscription website that offers users access to a huge database of horse racing data going back to 2009. It offers subscribers the chance to download csv files with huge amounts of data on every horse that has run in the UK and Ireland. Figure 10 shows a screengrab of a tiny fraction of this information.



Meeting	Time	RaceType	RaceClass	Title	Going	Furlongs	Prize	MinAge	MaxAge	MeanWei	Rating	Horse	CardNum	Gender	StallNum	StallPerce	Wearing	DaysSince	NumberO	Going
Chelmsfo	05:45	AW	115.5385	Bet totepl	STAND	7	6469	3	7	124	119.13	Spirit Of B	9	g	11	-16.98%	b	24	4	
Chelmsfo	05:45	AW	115.5385	Bet totepl	STAND	7	6469	3	7	124	118.04	Call Out L	8	g	5	4.36%	tv	22	8	
Chelmsfo	05:45	AW	115.5385	Bet totepl	STAND	7	6469	3	7	124	117.79	Robin We	13	g	6	-0.33%		-27	0	
Chelmsfo	05:45	AW	115.5385	Bet totepl	STAND	7	6469	3	7	124	117.54	Phalaborv	3	f	15	-20.47%		83	4	
Chelmsfo	05:45	AW	115.5385	Bet totepl	STAND	7	6469	3	7	124	117.29	Summer I	2	f	12	-18.95%		22	8	
Chelmsfo	05:45	AW	115.5385	Bet totepl	STAND	7	6469	3	7	124	117.04	Firmdecis	7	g	4	9.50%		22	8	
Chelmsfo	05:45	AW	115.5385	Bet totepl	STAND	7	6469	3	7	124	116.79	Human Na	6	g	10	-14.56%	t	37	3	
Chelmsfo	05:45	AW	115.5385	Bet totepl	STAND	7	6469	3	7	124	116.54	Plucky Dip	14	g	13	-20.47%		8	8	
Chelmsfo	05:45	AW	115.5385	Bet totepl	STAND	7	6469	3	7	124	116.29	Juan Hors	12	g	9	-11.68%	p	44	8	
Chelmsfo	05:45	AW	115.5385	Bet totepl	STAND	7	6469	3	7	124	115.03	Gold Hunt	11	g	3	15.10%	p	25	4	
Chelmsfo	05:45	AW	115.5385	Bet totepl	STAND	7	6469	3	7	124	114.59	Wellliesin	4	g	2	21.15%	v	14	8	
Chelmsfo	05:45	AW	115.5385	Bet totepl	STAND	7	6469	3	7	124	113.46	Dark Magi	10	g	7	-4.57%		-35	0	
Chelmsfo	05:45	AW	115.5385	Bet totepl	STAND	7	6469	3	7	124	110.69	Lady Lydia	15	m	8	-8.35%		25	8	
Chelmsfo	06:15	AW	109.744	Bet totepl	STAND	7	4528	2	2	130	120.48	Harbour V	2	c	2	15.75%		21	2	
Chelmsfo	06:15	AW	109.744	Bet totepl	STAND	7	4528	2	2	130	115.58	Mosskete	7	g	3	7.74%		15	2	
Chelmsfo	06:15	AW	109.744	Bet totepl	STAND	7	4528	2	2	130	110.08	Elsaakb	3	g	6	-10.62%	b	23	2	
Chelmsfo	06:15	AW	109.744	Bet totepl	STAND	7	4528	2	2	130	109.83	Dance Em	5	c	8	-18.13%		-23	0	
Chelmsfo	06:15	AW	109.744	Bet totepl	STAND	7	4528	2	2	130	109.58	Zalshah	1	c	4	0.67%	p	14	2	
Chelmsfo	06:15	AW	109.744	Bet totepl	STAND	7	4528	2	2	130	109.33	Strategic	9	g	7	-14.85%		-32	0	
Chelmsfo	06:15	AW	109.744	Bet totepl	STAND	7	4528	2	2	130	107.09	Blue Cand	4	c	9	-20.47%		15	3	
Chelmsfo	06:15	AW	109.744	Bet totepl	STAND	7	4528	2	2	130	106.3	Polar Ligh	8	f	5	-5.45%	b	-44	1	
Chelmsfo	06:15	AW	109.744	Bet totepl	STAND	7	4528	2	2	130	106.05	Lamb Cho	6	f	1	24.70%	p	-34	0	

Figure 10: Illustrative fragment of datafile

4.2: Data selection

The website has over 1 million horse entries with 469 separate columns available to subscribers, but this amount of data is problematic because of its size, meaning that a choice has to be made. Because of my computational limitations, I have decided to split the data into a number of different categories for the year 2017. I will be considering separately flat, hurdles, chases and all-weather (AW) racing, as they are all different enough for them to reasonably be looked at apart. Although AW racing is a type of flat racing, because it is run on a completely different surface (usually some sort of silica sand), it is often considered a separate discipline to turf flat racing. Similarly, hurdling and chasing can be considered separately for the purposes of this project, even though horses do compete in both disciplines.

The number of columns available is also problematic computationally, so they will need to be reduced to allow my ANNs to complete their process in a reasonable amount of time. Each of these columns is a potential factor in determining a horse's chances of winning, and many of them are some of the most common indicators that gamblers and tipsters look for when assessing a horse's chances, such as a horse's age, weight, trainer and jockey. A horse's recent form is possibly the biggest factor that many look at when reading a racecard, and I will retain

all the columns associated with this, with the database providing an individual column each for the following data for a horse's last five races: position, going and distance won/lost.

One thing to note is that I have deleted the 'Form' column, as although this is an indicator of the horse's recent form, it is a string of numbers that is meaningless in the context of running an ANN.

Consider a horse with form 4F-241 (form is read from right to left, with the righter most figure the most recent race). In its past five races it finished first, fourth, second and in the previous year fell and came fourth. Clearly, this form is an important piece of information for anyone looking to predict how that horse will perform, but an ANN reading 4F-241 will be able to make little of this information. Instead, the individual columns retaining data for each of the past five races are kept instead, so these columns will have individual entries of 1, 4, 2, F, 4. This is much more useful in the context of ANNs.

In total, I have retained over 120 columns for consideration, as I believe this gives a huge amount of data for the network to deal with, but also allows for it to complete its training in a reasonable time. A list of all these factors is provided in Appendix 2.

4.3: Dealing with non-numeric data

A further issue to note and to deal with is that neural networks can only deal with numeric data, meaning that any of the data that is not in this form needs to be converted. One obvious example of this is in the going of the ground at a racecourse on a particular day. An ANN will not be able to do anything with HEAVY or GOOD, so these need to be converted into numeric values. One possible way of dealing with non-numeric values such as the going would be to encode each going value with a corresponding numeric value: HARD - 0; FIRM - 1; GOOD TO FIRM - 2; GOOD - 3; GOOD TO SOFT - 4; SOFT - 5; HEAVY – 6. However, a problem with this approach is that by assigning numerical values to each of the goings, you are implicitly putting these things into a hierarchy, and the network might interpret the numbers as having meanings in themselves rather than being simple identifiers. This could cause the network to perform poorly.

The way I have chosen to deal with these values is to create a series of dummy columns for each of the string values. By doing this it will avoid the above pitfall because each column cell will be have binary encoding, depending on what the going was. Figure 11 shows how the dummy variables would work for the above example.

HARD	FIRM	GOOD/FIRM	GOOD	GOOD/SOFT	SOFT	HEAVY
1	0	0	0	0	0	0
0	1	0	0	0	0	0
0	0	1	0	0	0	0
0	0	0	1	0	0	0
0	0	0	0	1	0	0
0	0	0	0	0	1	0
0	0	0	0	0	0	1

Figure 11

So, if the going was good on a particular day, rather than a 'Going' column containing either GOOD , or a '3' on the above encoding, the Good column would contain a '1' while all the other going columns would contain a '0'. This solution can increase the number of columns significantly, but I believe it is the one most likely to produce accurate results.

I will highlight three possible columns where this solution is particularly troublesome because of the number of unique entries: horse name, jockey name and trainer name. There are only seven possible goings (eight if we include YIELDING from Ireland), but there are thousands of horse names and dozens of trainer and jockey names. As the data includes ratings for jockeys and trainers, and as the actual name of a horse has no bearing on its performance, only the other underlying data, it might be possible to dispense with these computationally expensive columns and instead rely on their numerical equivalents.

Other data that would be considered non-numeric, such as percentages and prize monies including currency signs, can be dealt with separately, by stripping any characters that might cause problems and re-categorising the columns as numeric.

```
try:
   df['StallPercentage'] = df['StallPercentage'].str.rstrip('%').astype('float')
except:
   pass
```

4.4: Treatment of finishing positions

As mentioned in section 3.2, there are various ways that a neural network could be constructed to consider the case of horse racing. I have chosen to view a horse race as a binary situation, where a horse either wins (1) or loses (0). It does not matter to me whether a horse finishes second, or falls at the first hurdle, I am only concerned with horses that win. That is not to say that a horse's performances in its previous runs is not significant, but that for the purposes of prediction, it is win or nothing. It is important to note that despite finishing positions being integers, this approach will still require the Result column to be split into a number of dummy columns, as there are a number of abbreviations for the performances of horses that do not finish the race (see Appendix 1 for further details).

4.5: Dealing with missing values

Dealing with missing values is always important in any statistical analysis, and machine learning techniques such as ANNs are no exception. Having as accurate a value for every input and feature is important for achieving as accurate as possible an outcome. There are various techniques for dealing with missing values, such as estimating them by using the mean, or median of the column, or of finding correlations with other features that might give an insight into the value. For the purposes of this project, I don't believe that any of these are particularly

useful, or necessary. Most of the missing values in my dataset are the result of the fact that there is no value. For instance, one of the columns in my dataset is to indicate if the race is a Handicap, giving cell values of Handicap or N/A. This can easily be dealt with by treating it as a binary option of Handicap (1), N/A (0). Similarly for missing values for features such as performance in a race five races ago, a missing value simply means that the horse has not raced five times. For this reason I will be treating missing values as zeroes, and for situations such as above where the column's value is a string, this method will simply create two parallel columns with binary dummy values.

4.6: Scaling of data

As a multilayer perceptron works by summing inputs and weighted signals between neurons, it is important to scale the data. This is because some features, such as the prize money on offer, can be as large as £900,000, and other features, such as a stall number, only have a range of about 20. Training a network on unscaled features could produce poor results as the larger-valued features could skew the network's training. To deal with this, all the data will be scaled so that they have an equal chance to influence the training of the network.

5: Implementation

5.1 Keras

I decided to make use of Keras, a high-level neural networks application program interface (API)[19] to build my neural network because of its ease of use and the fact that I had experience of using it in another (machine learning) module for this course. Keras allows users to quickly build neural networks by compiling models of layers of neurons, with the program allowing users to easily add desired parameters such as optimizers, activation functions and the number of training epochs. Keras models are written in Python code, and with the other functions needed to execute this project also written in Python, this will make it easy for everything to interact.

The basis of Keras is its Sequential model, which allows users to add layers of artificial neurons one by one until the required network shape is achieved.

```
from keras.models import Sequential

from keras.layers import Dense

model = Sequential()

model.add(Dense(units = , activation = ,
kernel_initializer = , input_dim =))

model.add(Dense(units = ,
kernel_initializer =, activation = ))

model.add(Dense(units = 1,
kernel_initializer = , activation = ))

model.compile(loss = loss, optimize r=
optimizer, metrics = metrics)

model.fit(x_train, y_train, epochs =
epochs, batch_size = batch_size)
```

Once the data has been sorted into appropriate input-output pairs (x, y) and into training and testing sets (x_train/y_train, x_test/y_test), the above code is a rough overview of how to build a neural network with one hidden layer. All of the parameters in the above code are to be decided, except for the number of units on the output layer. As this is a binary classification problem, there is one neuron in the output layer.

Once the network has been trained on the training data, it then uses the values it has calculated and tests them on the test set, and then compares its predicted y_test values with the actual values.

```
predY = model.predict (x_test)
```

The network will be run numerous times with different parameters to try to determine which set give the most accurate predictions.

5.2 Accuracy metrics

Keras then has a number of metrics that can be used to assess the accuracy of the model. The most common metric to use, and probably the most intuitive, would be the overall accuracy of the model in predicting a horse's performance. This would be a simple task of calculating the percentage of the model's predictions that match the actual performance of the horse. Unfortunately in the case for this project this measure of accuracy might be of little interest. Imagine a dataset where the test set comprises of around 400 races, with an average of eight horses in each race. That would mean there are around 3,200 horses in the test dataset. (For the sake of this argument there are complete races in the test dataset. In reality each horse is a separate datapoint so not every horse in a race might make it into the test dataset.) So in this dataset there would be 400 winning horses (1) and 2,800 losing horses (0). A neural network that predicted that every horse would lose would be correct 2,800 times, with an accuracy rate of 87.5%. If you were to say you had a neural network able to correctly predict how a horse would perform in a particular race 87.5% of the time, you might consider this an impressive rate. However, in this case it is meaningless as it has not predicted a single winning performance.

5.3 Understanding the results

As this project is concerned with picking winning horses, rather than necessarily predicting when a horse might lose, a better measure of the network's accuracy would be its recall, or sensitivity. This is defined as "proportion of actual positives that are correctly identified as such[27]". In the context of this project, the recall/sensitivity will calculate the proportion of winners that the network identifies. This means that it is the number of correctly predicted winners (true positive, or tp) divided by this added to the number of predicted losers that actually won (false negative, or fn). In algebraic terms:

$$Sensitivity = tp/(tp + fn)$$

This is a more useful measure that basic accuracy as it measures the proportion of winners the network identifies. It does have a similar problem to accuracy, however, in that if a network simply predicted every horse to win, it would have a 100% sensitivity rate. To counter this, another measure to determine a network's use is the specificity/precision. This is the ability of the network to correctly predict that a horse will not win a given race.

$$Specificity = tn/(tn + fp)$$

A high measure of specificity will ensure that the network is correctly identifying a large number of losing hoses, and when both measures are high it indicates an accurate network that can correctly predict both classes with a good level of accuracy.

These two metrics are calculated from a confusion matrix, which is a table setting out the performance of a network in predicting outcomes.

N = 276	*Pred Yes*	*Pred No*
Result Yes	111	165
Result No	155	121

Figure 12: Example confusion matrix

In the example matrix shown in Figure 12, the sensitivity and specificity metrics would be calculated like so:

Sensitivity = 111 / (111 + 165) = 0.402.

Specificity = 121 / (121 + 155) = 0.438.

The issue of false positive predictions is an easy one to understand. A system of betting on every horse predicted to win needs to minimise its false positive rate as every one of these results in a loss. The issue of false negatives is also one to consider, however, as while false negatives will not lose any money, if the rate is high enough it will result in a high level of lost potential winnings.

5.4 Calculating profit/loss

An extra consideration in the context of this project is the starting price (SP) of any horse predicted to win a race. In other situations with a binary outcome that might be looked at by neural networks, such as disease diagnosis, the only thing that matters is the correct prediction. Here, however, if a network were to be useful for betting purposes, it would not only have to deliver consistent results with high sensitivity and specificity, the correct selections it provided would need to deliver returns on investment. For example, if a network predicted 40% of winners correctly, but these winning predictions had an average winning price of evens (1/1), it would be of no use from a betting perspective. If one were to bet on every horse predicted to win by this network, it would end up in an overall loss. If we were to bet on 100 predicted winners from such a network, the 40 winning horses would win us £40, but the 60 losing predictions would lose us £60, leaving us with a net loss of £20. Clearly, we will need to assess any potential returns from a network's predictions rather than just the statistics of its win/loss rate.

This project is unconcerned if the network predicts more than one horse to win a race. It is entirely possible that for a number of races, the network returns no predicted winner, and for other races it returns more than one. This would simply reflect the fact that some races are more difficult to predict the outcome of than others. To assess a model's profit/loss every predicted winner will be included in the calculation. If the model predicts two horses to win the same

race, and both are 10/1, then backing both of these would still produce a profit of £9 (presuming that one of them actually wins the race).

6: Results and Analysis

6.1: Analysis of results

This project looked at the four different types of racing - flat, AW, hurdles, chase - separately, and all of these types of racing returned similar, and very positive results. Figure 13 shows a fragment of the file created each time the network was run, listing the actual result for each horse, the prediction, the horse's name and its odds at the time of the race. It also records the sensitivity, specificity and the total number of each prediction made by the model.

	A	B	C	D	E	F	G	H	I	J	K
1	Result	Prediction	Price	Horse Name	Sensitivity	Specificity	True neg	False neg	False pos	True pos	
2	1	1	4.5	Acdc	0.819306931	0.99858457	2822	73	4	331	
3	0	0	7.5	Leith Hill Legasi							
4	0	0	26	Magic Money							
5	0	0	17	Mahler Lad							
6	0	0	11	Helpston							
7	0	0	9	Kemboy							
8	0	0	19	Wychwoods Brook							
9	0	0	8	Best Boy Barney							
10	0	0	0	Product Of Love							
11	0	0	34	Mountain King							
12	0	0	67	Joint Reaction							
13	1	1	3	Call The Taxie							
14	0	0	0	A Decent Excuse							
15	0	0	0	Draycott Place							
16	0	0	11	Zanstra							
17	0	0	5.5	Static Jack							
18	0	0	26	Teacher's Pet							
19	0	0	17	Vice Et Vertu							
20	0	0	9	Cucklington							
21	0	0	26	Cloudy Morning							
22	0	0	17	Ballotin							
23	0	0	10	Flying Angel							
24	0	0	17	Father Edward							
25	0	0	8	De Blacksmith							
26	1	1	9	Danvinnie							
27	0	0	17	No No Cardinal							
28	0	0	15	Dresden							
29	0	0	9	Millanisi Boy							
30	0	0	9	Bon Chic							
31	0	0	2.1	Hammersly Lake							
32	0	0	11	Monte Wildhorn							
33	0	0	4.5	Agentleman							
34	0	0	0	Harleys Max							
35	0	0	3	Always On The Run							
36	0	0	21	Hurricane Sky							
37	1	0	5	Azzuri							
38	0	0	67	Turban							

Figure 13: Illustrative fragment of results file

Each time the network was run with different parameters another file was created to keep a record of how it performed, including the sensitivity and specificity, but also the profit/loss figure and how long the network took to run. As these neural networks are training themselves with hundreds of thousands of parameters, the length of time they take to execute must be factored into their assessment. A network that runs for an extra 30 minutes, but only offers a tiny improvement in performance, might be seen as inferior to one that offers the absolute best performance. Figures 14-17 show these calculations for all four types of racing.

Epochs	Batch Size	Neurons	Optimizer	Sensitivity	Specificity	True neg	False neg	False pos	True pos	Time	Biggest wi	Ave win	P/L
10	32	64	Adam	0.841584	0.997877	2820	64	6	340	95.70513	34.33	5.269761	1445.7
20	32	64	Adam	0.868812	0.997877	2820	53	6	351	163.3193	34.33	5.420371	1545.55
40	32	64	Adam	0.85396	0.996108	2815	59	11	345	311.0689	51	5.429276	1517.1
80	32	64	Adam	0.858911	0.997523	2819	57	7	347	375.6086	34.33	5.329491	1495.33
120	32	64	Adam	0.839109	0.998231	2821	65	5	339	787.8844	34.33	5.315438	1457.93
160	32	64	Adam	0.819307	0.998585	2822	73	4	331	1151.235	34.33	5.315438	1457.93
20	64	64	Adam	0.787129	0.997877	2820	86	6	318	135.3103	51	5.289266	1357.97
20	16	64	Adam	0.861386	0.998585	2822	56	4	348	197.5974	51	5.464432	1549.62
20	8	64	Adam	0.861386	0.998231	2821	56	5	348	260.0533	34.33	5.361056	1512.65
20	16	128	Adam	0.930693	0.996108	2815	28	11	376	302.1418	51	5.531851	1692.98
20	16	256	Adam	0.970297	0.996108	2815	12	11	392	577.1162	51	5.568603	1779.89
20	16	512	Adam	0.980198	0.991507	2802	8	24	396	1071.888	51	5.551496	1778.39
20	16	256	Adadelta	0.972772	0.993984	2809	11	17	393	842.3298	51	5.57161	1779.64
20	16	128	Adadelta	0.930693	0.987261	2790	28	36	376	526.0867	51	5.657858	1715.35
20	16	512	Adadelta	0.967822	0.996461	2816	13	10	391	1723.467	51	5.509955	1753.39
20	16	256	Adagrad	0.85396	0.997169	2818	59	8	345	365.0267	51	5.440993	1524.14
20	16	512	Adagrad	0.84901	0.997523	2819	61	7	343	709.3626	51	5.282299	1461.83
20	16	128	Adagrad	0.819307	0.999646	2825	73	1	331	173.3723	51	5.520745	1495.36

Figure 14: Performance of chase models

Epochs	Batch Size	Neurons	Optimizer	Sensitivity	Specificity	True neg	False neg	False pos	True pos	Time	Biggest wi	Ave win	P/L
20	16	128	Adam	0.924494	0.998024	5051	41	10	502	418.7988	41	6.947455	2975.62
20	16	256	Adam	0.966851	0.996839	5045	18	16	525	842.7075	41	7.067973	3169.69
20	16	512	Adam	0.976059	0.997431	5048	13	13	530	1786.087	101	7.266746	3308.38
20	16	1024	Adam	0.965009	0.996048	5041	19	20	524	5101.832	41	7.071204	3161.31
20	16	256	Adagrad	0.906077	0.997431	5048	51	13	492	1435.415	41	6.831376	2856.04
20	16	256	Adadelta	0.965009	0.998814	5055	19	6	524	2050.012	41	7.083631	3181.82

Figure 15: Performance of AW models

Epochs	Batch Size	Neurons	Optimizer	Sensitivity	Specificity	True neg	False neg	False pos	True pos	Time	Biggest wi	Ave win	P/L
10	16	128	Adam	0.949827	0.997104	5508	29	16	549	415.0844	101	6.32243	2906.01
20	16	128	Adam	0.953287	0.998371	5515	27	9	551	925.0529	101	6.364959	2947.09
40	16	128	Adam	0.948097	0.998733	5517	30	7	548	1295.028	101	6.337547	2917.98
40	16	256	Adam	0.974048	0.998733	5517	15	7	563	2723.781	67	6.151585	2893.34
20	16	128	Adagrad	0.866782	0.99819	5514	77	10	501	464.733	101	6.020394	2505.22
20	16	128	Adadelta	0.937716	0.993845	5490	36	34	542	697.9934	101	6.47033	2930.92
20	16	256	Adam	0.942907	0.993483	5488	33	36	545	1261.216	101	6.444974	2931.51

Figure 16: Performance of hurdle models

Epochs	Batch Size	Neurons	Optimizer	Sensitivity	Specificity	True neg	False neg	False pos	True pos	Time	Biggest wi	Ave win	P/L
20	16	128	Adam	0.968717	0.99935	9231	30	6	929	1143.897	81	6.926015	5499.27
20	16	256	Adam	0.96976	0.999567	9233	29	4	930	2111.641	81	6.872062	5457.02
40	16	128	Adam	0.949948	0.998917	9227	48	10	911	1597.428	81	6.867391	5335.19
20	16	64	Adam	0.914494	0.999675	9234	82	3	877	352.5071	81	6.741221	5032.05
20	16	128	Adadelta	0.970803	0.998376	9222	28	15	931	821.4483	81	7.005505	5576.13
20	16	256	Adadelta	0.950991	0.997618	9215	47	22	912	2391.014	81	6.920505	5377.5
20	16	128	Adagrad	0.913452	0.99935	9231	83	6	876	809.9284	81	6.82621	5097.76

Figure 17: Performance of flat models

As can be seen from these tables, the performances of the models is remarkable, with over 95% sensitivity achieved for all four types of racing. A staking plan of putting £1 on every horse predicted to win by the optimum models would have returned £13,611.49 over the course of 2017. The models were all also successful in predicting some bigger-priced winners, with the AW and hurdle models both correctly predicting a 100/1 winner.

It can also be seen that all models were excellent at minimising incorrect winner predictions. It can also be seen that for all types of racing, it is not necessarily those models that have the lowest rate of incorrect winning predictions, or even the most correct winning predictions that are the most successful overall. The best models also provide low proportions of false negative predictions, as these minimise the lost potential that comes from not betting on winners.

It can also be seen that a relatively low number of epochs is needed to properly train the networks to deliver good results. This became clear quite early on, with Figures 18-21 showing that the potential gains start to level off quite early in training, with the potential loss also starting to rise quite early. It was found that for all types of racing 20 epochs of training was optimal for predicting winners.

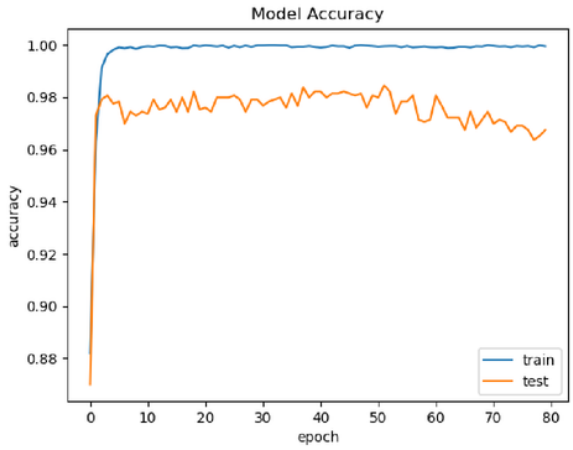


Figure 18: Example model accuracy plot

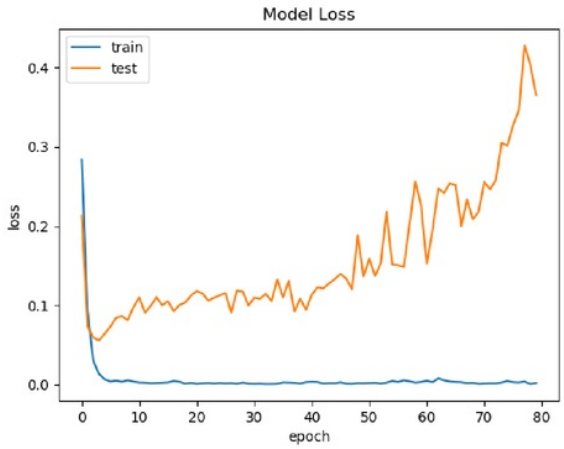


Figure 19: Example model loss plot

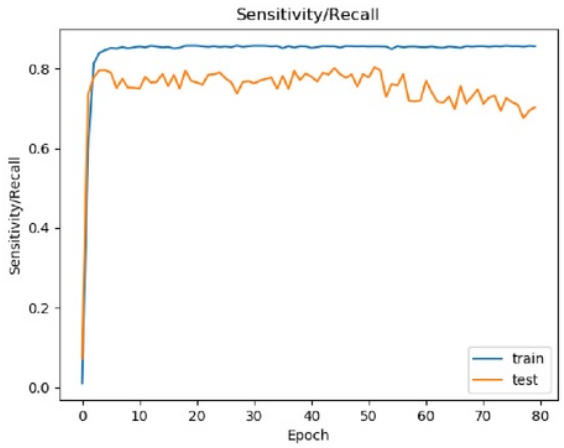


Figure 20: Example sensitivity plot

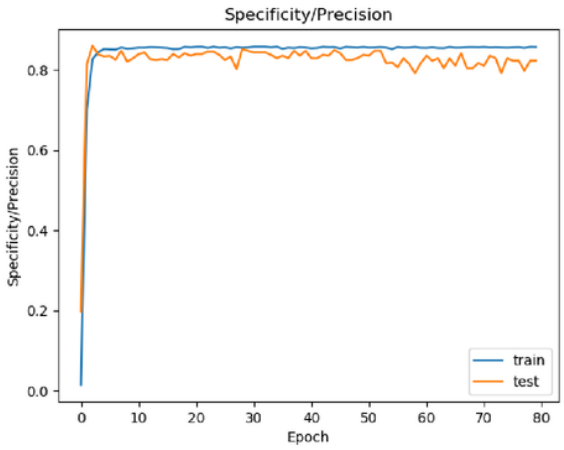


Figure 21: Example specificity plot

The network was first fine-tuned for the data from chasing, with the information gleaned during this process used to inform the fine-tuning of the other types of racing. Surprisingly, the number of neurons in the hidden layer required to produce good results was also relatively small, increasing their number too much led to a decrease in performance. For both flat racing and AW racing, only 128 neurons in the hidden layer were found to produce optimal results.

The Adam optimizer was found to be a good all-rounder, producing good results for all types of racing, and was the optimal choice for all except flat racing, for which AdaDelta proved to be the optimizer in the best model.

6.2: Further testing of models

While these results of the given data are remarkable, the real value in any horse racing prediction system is its ability to use the models for future use. With this in mind, the website that provided the data for building the network also provides example files that subscribers would receive on the night before a day's racing. This would allow subscribers to use these files, run their own systems, and extract predictions for betting on.

In order to further test just how good the models produced really are, the weights from each of the four networks were saved, and then these weights were implemented in a neural network on the pre-race data files. Keras does allow the saving of complete models, but that is not useful in this case as it saves the architecture of the input layer. While the extra datafiles have the same number of columns as those used here, after dummy columns are created this will no longer be the case because of the potentially different number of names of the going or jockey or a number of other columns. For this reason the weights are saved and implemented on a network designed to accept the new data.

Unfortunately, the performance of the model in predicting these races was far worse than that of the training and testing phases. Figure 22 shows a table of the relative performances on the four types of racing.

Type of racing	*Total no of races*	*No of pred races*	*Percentage of preds*	*No of predicted winners*	*No of correct winners*	*Percentage of winning preds*	*Profit/loss*
AW	127	113	88.98%	344	31	9.01%	-115.48
Flat	246	151	61.38%	282	37	13.12%	-86.13
Chase	87	74	85.06%	214	24	11.21%	-110.46
Hurdle	133	14	10.53%	14	3	21.43	+2.33
Total	593	352	59.36%	854	95	11.12%	-309.74

Figure 22: Table of real-world results

Using the saved models would have resulted in a loss of £309.74 over the course of 592 race. The model was also very uneven in providing predictions, often offering no prediction for a race and on other occasions offering many. An extreme of this was in the Galway Plate, where the model offered 14 predictions out of a field of 22 runners, and all its predictions failed to win.

There were also wide discrepancies in how often the model would actually predict a horse to win a race. While it predicted a horse to win over 85% of chases and AW races, it picked a winner in just 14 out of the 133 hurdles races run in the real-world testing phase. This suggests that more work needs to be done on the implementation of the neural network in order for it to be translatable to real-world predicting. Interestingly, hurdling was the only racing type that delivered a real-world profit, albeit very small, and the small sample size of predicted winners makes drawing conclusions very difficult.

One of the reasons for the mismatch of the performances of the model during the training/testing phase and in a real-world application could be the fact that the number of inputs is different in the two iterations. Through the use of dummy columns to encode non-numeric values, this creates potentially hundreds of new columns in training data. However, in the situation of predicting winners on a day-to-day bases, the number of dummy columns is likely to be far lower. Consider the data used to train the model, it takes races from the entirety of 2017, meaning that all of the different types of going are likely to be found. However, on a particular day in the summer, it is likely that goings such as heavy or soft are less likely to be found, meaning that there is no need for dummy columns with these headings. On a much larger scale, the 2017 data will include the names of dozens of trainers and jockeys, many of whom might not be involved on a specific day. The mismatch of these numbers might make the model perform poorly.

6.3: Remedying problems with real-world performance

In order to remedy the mismatch of inputs, it might be necessary to eliminate any non-numeric values without resorting to dummy columns before training and testing a model. Another potential solution might be to ensure both input dimensions are the same by creating dummy

columns that cover all eventualities in the real-world pre-race data files. As the data includes ratings for both jockeys and trainers, it might be possible to remove their names entirely from all data, as this would both reduce the computational requirements of the model as well as making the matching of data easier.

For other non-numeric data, the above method would then be a simpler task, as there are limited types of going, alarms, headgear etc.

As ever in Data Science, another potential solution to the poor performance is to use more data to try to train the models. This could be done in two ways, by including more of the potential factors included in the original data, or by using more years' worth of data in the training and testing phases. Both these potential solutions would increase the complexity, and time, that the stage takes, but it might be necessary if performance were to be improved.

A final possible way of improving the real-world performance would be to scale down how the model is deployed. For these tests, the model weights for flat racing were run against all the datapoints for flat races in the file. It might be necessary to run the model a number of times before a day's racing. Perhaps it might even need to be run individually for each race, or for each meeting, in order for it to improve its performance.

6.4: Potential future study

The obvious future study of this topic would be to further investigate the reasons for its poor real-world performance and remedy them. However, were the model ever to prove successful, or even adequate, in real-world prediction, there are potential avenues for it to be refined even further.

Each individual racecourse could be considered individually as a possible way of improving performance. Each track across the country has its own particular features, so building individual networks for each track might be a way of improving performance. While the course is included as a feature in this study, so will be an influencing factor, it would remain a small part of the total network, so removing this and training models individually for each course might improve their performance.

A potential course for further study would be to look at a neural network that is not just looking at winners, but has a multidimensional output layer offering a prediction for the position of every horse in each race. One thing to note about this approach is that every race has a different number of runners, so the output layer could not be fixed as it is here. This approach would likely need each race to be grouped in the training phase in order to provide the correct number of outputs needed, so would be far more complex to set up and train.

Another potential route to take would be rather than designing a network that offers simple binary predictions of winning for each horse, but one that offers a percentage probability that a horse wins a race. If such a neural network were able to be constructed and were sufficiently

accurate, a user could potentially compared the network's output with a horse's odds, which in turn gives an implied probability of winning. These results could then be used as part of a more sophisticated staking plan where bets are places on horses for whom their implied probability of winning is greater than those of the odds on offer.

Finally, if one were to broaden the horizon to other sports, greyhound racing would be a similar project to undertake, but the number of variables are far fewer than in horse racing, so might be an easier task to undertake, depending on the availability of suitable data sources.

7: Conclusion

The goal of this project was to design and implement an artificial neural network to better analyse the odds of horse races with a view to predicting winners on a regular basis. While it was successful in implementing such a network, when this model was then deployed onto real-world races, its performance declined dramatically, meaning that more work would need to be done before such as network might be considered a success. It would also need to provide steady returns over a relatively significant portion of time if any profits it did provide could be seen as consistent enough to suggest that it might be used as a successful system that would deliver long-term returns.

8: References

[1]: https://www.britishhorseracing.com/press_releases/economic-impact-study-highlights-british-racing-as-worth-billions-to-british-economy/

[2]: https://www.telegraph.co.uk/sport/horseracing/6990498/Racing-is-the-second-biggest-spectator-sport-Horse-Racing.html

[3]: https://www.ascot.co.uk/archive/magnificent7

[4]: https://www.thetimes.co.uk/article/how-to-read-a-racecard-gd02d07mm

[5]: https://en.wikipedia.org/wiki/Betting_and_Gaming_Act_1960

[6]: http://www.onlinebetting.org.uk/betting-guides/biggest-bookies.html#biggest

[7]: http://www.gamblingcommission.gov.uk/news-action-and-statistics/Statistics-and-research/Statistics/Industry-statistics.aspx

[8]: https://www.racingpost.com/tipping/naps-table/

[9]: Rojas R. Neural Networks-A Systematic Introduction, Ch7-8, Springer Berlin Heidelberg (1996)

[9]: https://scholar.google.co.uk/scholar?hl=en&as_sdt=0%2C5&q=%22horse+racing%22+%22neural+network%22&btnG=

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

[11]: Williams, J., & Li, Y. (2008). A case study using neural networks algorithms: Horse racing predictions in Jamaica. In Proceedings of the International Conference on Artificial Intelligence (ICAI 2008) (pp. 16-22). CSREA Press.

[12]: Pudaruth, S., Medard, N., & Dookhun, Z. B. (2013). Horse Racing Prediction at the Champ De Mars using a Weighted Probabilistic Approach. *International Journal of Computer Applications*, *72*(5).

[13]: Chen, H., Rinde, P. B., She, L., Sutjahjo, S., Sommer, C., & Neely, D. (1994). Expert prediction, symbolic learning, and neural networks. An experiment on greyhound racing. IEEE Expert, 9(6), 21-27.

[14]: Nakamoto, Pat. Neural Networks and Deep Learning, CreateSpace (2018)

[15]: Rashid, Tariq. Make Your Own Neural Network, CreateSpace (2016)

[16]: https://www.allerin.com/blog/4-benefits-of-using-artificial-neural-nets

[17]: https://medium.com/greyatom/what-is-underfitting-and-overfitting-in-machine-learning-and-how-to-deal-with-it-6803a989c76

[18]: Krizhevsky, Alex & Sutskever, Ilya & E. Hinton, Geoffrey. (2012). ImageNet Classification with Deep Convolutional Neural Networks. Neural Information Processing Systems. 25. 10.1145/3065386.

[19]: http://www.saedsayad.com/artificial_neural_network.htm

[20]: https://medium.com/@kanchansarkar/relu-not-a-differentiable-function-why-used-in-gradient-based-optimization-7fef3a4cecec

[21]: Duval, F. (2018) Artificial Neural Networks: Concepts, Tools and Techniques Explained for Absolute Beginners, Ch4, CreateSpace.

[22]: https://www.safaribooksonline.com/library/view/getting-started-with/9781786468574/ch04s04.html

[23]: Nesterov, Yurii; Arkadii, Nemirovskii (1995). Interior-Point Polynomial Algorithms in Convex Programming. Society for Industrial and Applied Mathematics

[24]: https://keras.io/

[25]: https://int8.io/comparison-of-optimization-techniques-stochastic-gradient-descent-momentum-adagrad-and-adadelta/#AdaGrad_-_description

[26]: https://en.wikipedia.org/wiki/Universal_approximation_theorem

[27]: https://www.heatonresearch.com/2017/06/01/hidden-layers.html

[28]: https://en.wikipedia.org/wiki/Sensitivity_and_specificity

9: Bibliography

Pudaruth, Sameerchand & Jogeeah, Manish & Kumar Chandoo, Akshay. (2015). Using ANNs to Predict Winners in Horseraces: A Case Study at the Champs de Mars.

N. M. Allinson and D. Merritt, "Successful prediction of horse racing results using a neural network", IEE Colloquium on Neural Networks: Design Techniques and Tools, London, UK, 1991, pp. 4/1-4/7.

Nairn, D. J. (2010). An Investigation into the Effectiveness of an Artifi-cial Neural Network in the Prediction of Results in British Flat Racing (Doctoral dissertation, Thesis (MS). University of Stirling).

Davoodi, E., & Khanteymoori, A. R. (2010). Horse racing prediction using artificial neural networks. Recent Advances in Neural Networks, Fuzzy Systems & Evolutionary Computing, 2010, p155-160.

Schumaker, R. P., Solieman, O. K., & Chen, H. (2010). Predictive modeling for sports and gaming. In Sports Data Mining. Springer.

Link, D. (2018). Data Analytics in Professional Soccer: Performance Analysis Based on Spatiotemporal Tracking Data. Springer.

O'Donoghue, P., & Holmes, L. (2014). Data analysis in sport. Routledge.

Sacha, D., Stein, M., Schreck, T., Keim, D. A., & Deussen, O. (2014, October). Feature-driven visual analytics of soccer data. In Visual Analytics Science and Technology (VAST), 2014 IEEE Conference on (pp. 13-22). IEEE.

Lewis, M. (2004). Moneyball: The art of winning an unfair game. WW Norton & Company.

Appendix 1: Common horseracing abbreviations

Race types

Am - amateur

App - apprentice

Auc - auction

AW - all-weather

Ch - chase

Cls - Class

Cond - NH conditional

Div - Division

Gd - Grade 1,2 or 3 (Jumps)

Gp - Group 1, 2 or 3 (Flat)

Hcap - handicap

Hur or H - hurdle

List - Listed

Mdn - maiden

NHF - National Hunt Flat race

Nov - novice

Sell - selling

Sex and Colours

b - bay

bl - black

br - brown

ch - chestnut

c - colt

f - filly

g - gelding

gr - grey

h - horse

m - mare

r - rig

ro - roan

wh - white

Going

f or fm - firm

g or gd - good

hd - hard

hy or hvy - heavy

s or sft - soft

stand - standard AW

yld - yielding (IRE)

Distances

dist - distance (240y from finish)

f - furlong

hd - head

l - length

m - mile

nk - neck

nse - nose (shortest margin)

shd - short head

y - yards

Headgear

h - hood

b - blinkers (if being worn for a first or second time a 1 or 2 will be beside the letter)

p - (sheepskin) cheekpieces

t - tongue-tie

v - visor

e - eye hood

Ht - hood and tongue-tie

e/c - eyecover

e/s - eyeshield

Form figures

1-9 - position the horse finished. All-weather (Flat) and point-to-point (jumps) are in bold.

0 - if the horse finished outside the top 10

- between numbers indicate year separation (i.e. left of this is from a previous year)

/ between numbers indicates a season separation (i.e. left of this is from the season before last)

B - brought down

C - carried out

D - disqualified

F - fell

HR - hit rails

L - left at start

O - horse ran out

P - pulled up

R - refused

S - slipped up

U - unseated rider

V - void race

Appendix 2: List of column headers in data files

Meeting

RaceClass

Going

Furlongs

Prize

MinAge

MaxAge

MeanWeight

Rating

Horse

CardNumber

Gender

Wearing

DaysSinceLastRun

NumberOfResults

DistanceRegression

Age

Weight_Pounds

TotalWins

RecentWins

CourseWins

Penalties

Allowances

RawRating

Jockey

JockeyRating

Trainer

TrainerRating

FormTrend

FormLastRun

Class

WinningForm

Speed

Alarms

ValueOdds_Probability

ChaseJumpingAbility

RatingsPosition

Handicap

ValuePlaceOdds

BetFairSPForecastWinPrice

BetFairSPForecastPlacePrice

RawAdjustedForAgeAndWeight

TJCWins

TJCRuns

Runners

TCWins

TCRuns

JCWins

JCRuns

ValueOdds_BetfairFormat

TJWins

TJRuns

Selling

Claiming

Novice

Maiden

Beginner

HunterChase

RawRanking

RAdjRanking

Race5RunsAgo

Race4RunsAgo

Race3RunsAgo

Race2RunsAgo

Race1RunAgo

UKHR_SireID

UKHR_DamID

Race5RunsAgoRaceType

Race4RunsAgoRaceType

Race3RunsAgoRaceType

Race2RunsAgoRaceType

Race1RunAgoRaceType

Race5RunsAgoRaceClass

Race4RunsAgoRaceClass

Race3RunsAgoRaceClass

Race2RunsAgoRaceClass

Race1RunAgoRaceClass

Race5RunsAgoRaceClassType

Race4RunsAgoRaceClassType

Race3RunsAgoRaceClassType

Race2RunsAgoRaceClassType

Race1RunAgoRaceClassType

LengthsWonLostLastRun

LengthsWonLost2RunsAgo

LengthsWonLost3RunsAgo

LengthsWonLost4RunsAgo

LengthsWonLost5RunsAgo

RacesSincePreviousTrainerWin

DaysSincePreviousTrainerWin

SpeedRating

SpeedRatingRank

RatingAdvantage

PlacePositions

Raw Advantage

RAdj Advantage

PositionLastTime

Position2RunsAgo

Position3RunsAgo

Position4RunsAgo

Position5RunsAgo

DistanceLastTime

Distance2RunsAgo

Distance3RunsAgo

Distance4RunsAgo

Distance5RunsAgo

GoingLastTime

Going2RunsAgo

Going3RunsAgo

Going4RunsAgo

Going5RunsAgo

DistanceRuns

DistanceWins

GoingRuns

GoingWins

Betfair Placed

Betfair Place S.P.

Betfair Win S.P.

Actual Going

Result

S.P.

Appendix: Code

Neural network:

```
import numpy as np
import tensorflow as tf
import random as rn
import os
os.environ['PYTHONHASHSEED'] = '0'
np.random.seed(50)
rn.seed(50)
session_conf = tf.ConfigProto(intra_op_parallelism_threads=1, inter_op_parallelism_threads=1)
from keras import backend as K
tf.set_random_seed(50)
sess = tf.Session(graph=tf.get_default_graph(), config=session_conf)
K.set_session(sess)
import pandas as pd
from keras import optimizers
import keras
import keras_metrics
from sklearn import preprocessing, metrics
from keras.models import Sequential
from keras.layers import Dense, Dropout, Activation
from keras.models import load_model
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.metrics import confusion_matrix, classification_report, recall_score, precision_score
import matplotlib.pyplot as plt



def horseNeuralNetwork (filename, optimizer, neurons1, neurons2):
    df = pd.read_csv(filename, encoding = "ISO-8859-1", low_memory = False, dtype={'Prize': 'object'})
    df = pd.DataFrame(df)



    # Encode horse names with integers
    le = preprocessing.LabelEncoder()
    le.fit(df['Horse'])
    horseNameCodes = le.transform(df['Horse'])
```

```
df['Horse Code'] = le.transform(df['Horse'])


# Convert StallPercentage to float: only needed for Flat and AW
try:
    df['StallPercentage'] = df['StallPercentage'].str.rstrip('%').astype('float')
except:
    pass



# Deal with £ and € symbols
df['Prize'] = df['Prize'].str.replace('£', '')
df['Prize'] = df['Prize'].str.replace('\x80', '')
df['Prize'] = df['Prize'].str.replace(',', '')
df['Prize'] = df['Prize'].astype('float')

# Ensure RaceClass is float
df['RaceClass'] = df['RaceClass'].astype('float')
# Ensure Meanweight is float
df['MeanWeight'] = df['MeanWeight'].astype('float')
# Ensure Rating is float
df['Rating'] = df['Rating'].astype('float')

# Deal with potential No Odds SP and ensure float
try:
    df['S.P.'] = df['S.P.'].str.replace('No Odds', '0')
except:
    pass
df['S.P.'] = df['S.P.'].astype('float')

# Replace missing values with 0
df.fillna(0, inplace = True)

# Drop Horse column to reduce number of dummy columns
df = df.drop('Horse', axis = 1)

# Convert strings into numeric values
df = pd.get_dummies(df, dtype = float)
```

```
# Extract Result column for use as y values
x_values = df.drop(['Result_1'], axis = 1)
y_values = df['Result_1']

# Split data into training and test sets
x_train, x_test, y_train, y_test = train_test_split(x_values, y_values, test_size=0.20, random_state=50)

# Create separate dataframe of x_test values to allow inverse transformation
# of horse codes into names
xtestpd = pd.DataFrame(x_test)

# Drop Horse Code and SP from data so as not to train network on them
x_train = x_train.drop(['Horse Code'], axis = 1)
x_train = x_train.drop(['S.P.'], axis = 1)
x_test = x_test.drop(['Horse Code'], axis = 1)
x_test = x_test.drop(['S.P.'], axis = 1)

# Return horse names to separate dataframe
xtestpd['Horse Name'] = le.inverse_transform(xtestpd['Horse Code'])

# Set size of input layer for network
inputNeurons = len(x_train.columns)

# Scale features
scaler = StandardScaler()
scaler.fit(x_train)
scaled_x_features = scaler.transform(x_train)
x_train = scaler.transform(x_train)
x_test = scaler.transform(x_test)



# Define the model
model = Sequential()
# Input layer
model.add(Dense(neurons1, input_dim=inputNeurons, kernel_initializer = 'uniform', activation = 'relu', name = 'input'))
model.add(Dropout(0.4))
# Hidden layer 1
```

```
model.add(Dense(neurons2, kernel_initializer = 'uniform', activation = 'relu', name = 'hidden'))
model.add(Dropout(0.5))
# Output layer
model.add(Dense(1, kernel_initializer = 'uniform', activation = 'sigmoid', name = 'output'))
model.summary()

# Compile the model
model.compile(loss='binary_crossentropy', optimizer = optimizer, metrics = ['accuracy', keras_metrics.precision(), keras_metrics.recall()])



# Train the model
fit_network = model.fit(x_train, y_train, epochs = 20, batch_size = 16, validation_split = 0.10, shuffle = False, verbose = 0)

# Save model weights when optimal network is found
#model.save_weights('AW_weights.h5')



# Extract y predictions
predY = model.predict(np.array(x_test))
predY = np.round(predY).astype(int).reshape(1,-1)[0]

### Create confusion matrix
m = confusion_matrix(predY,y_test)
tn, fn, fp, tp = confusion_matrix(predY,y_test).ravel()
m = pd.crosstab(predY,y_test)
print("Confusion matrix")
print(m)

# Calculate sensitivity and specificity
sensitivity = tp/(tp+fn)
specificity = tn/(tn+fp)
print("Sensitivity: ", sensitivity)
print("Specificity: ", specificity)

# Plot recall/sensitivity
plt.plot(fit_network.history['recall'])
plt.plot(fit_network.history['val_recall'])
plt.title('Sensitivity/Recall')
```

```
plt.ylabel('Sensitivity/Recall')
plt.legend(['train', 'test'])
plt.xlabel('Epoch')
plt.show()

#Plot precision/specificity
plt.plot(fit_network.history['precision'])
plt.plot(fit_network.history['val_precision'])
plt.title('Specificity/Precision')
plt.ylabel('Specificity/Precision')
plt.legend(['train', 'test'])
plt.xlabel('Epoch')
plt.show()

# Plot accuracy
plt.plot(fit_network.history['acc'])
plt.plot(fit_network.history['val_acc'])
plt.title('Model Accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'])
plt.show()

# Model loss
plt.plot(fit_network.history['loss'])
plt.plot(fit_network.history['val_loss'])
plt.title('Model Loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'])
plt.show()

## Print classification report
print(classification_report(y_test,predY))
```

Profit/loss:

```
def calculateProfitLoss(filename):
    import pandas as pd

    df = pd.read_csv(filename)
    df['Result'] = df['Result'].astype('int')
    df['Prediction'] = df['Prediction'].astype('int')
    df['Price'] = df['Price'].astype('float')

    # Calculate cost of £1 bet on every prediction
    outlay = sum(df['Prediction'] == 1)
    ## Prices include stake
    # Trim database to only correct winning predictions
    df = df[df['Result'] == 1]
    df = df[df['Prediction'] == 1]

    # See largest winning price
    print("Biggest winner:")
    print(max(df['Price']))

    # Calculate average winning price
    print("Average winning price:")
    print(df['Price'].mean())

    # Sum returns from these predictions
    returns = sum(df['Price'])

    profit = returns - outlay

    # Calculate overall profit
    print(profit)
    return(profit)
```

Implementing network on new predictions:

```
import numpy as np
import tensorflow as tf
import random as rn
import os
os.environ['PYTHONHASHSEED'] = '0'
np.random.seed(50)
rn.seed(50)
session_conf = tf.ConfigProto(intra_op_parallelism_threads=1, inter_op_parallelism_threads=1)
from keras import backend as K
tf.set_random_seed(50)
sess = tf.Session(graph=tf.get_default_graph(), config=session_conf)
K.set_session(sess)
import pandas as pd
from keras import optimizers
from sklearn import preprocessing
from keras.models import Sequential
from keras.layers import Dense, Dropout, Activation
from keras.models import load_model
from sklearn.preprocessing import StandardScaler


def newPreds(filename, weights):


    df = pd.read_csv(filename, encoding = "ISO-8859-1", low_memory = False, dtype={'Prize': 'object'})
    df = pd.DataFrame(df)


    raceName = df['Title']
    horseList = df['Horse']
    df = df.drop(['Horse'], axis = 1)
    df = df.drop(['Title'], axis = 1)
    df['Prize'] = df['Prize'].astype('float')
    try:
        df['StallPercentage'] = df['StallPercentage'].str.rstrip('%').astype('float')
    except:
        pass
    df.fillna(0, inplace = True)
    df = pd.get_dummies(df, dtype = float)
```

```
x_values = df
inputNeurons = len(x_values.columns)

scaler = StandardScaler()
scaler.fit(x_values)
x_values = scaler.transform(x_values)

model = Sequential()
model.add(Dense(512, input_dim = inputNeurons, kernel_initializer='uniform', activation='relu'))
model.add(Dense(256, kernel_initializer='uniform', activation='relu'))#, name = 'hidden'))
model.add(Dense(1, kernel_initializer='uniform', activation='sigmoid'))#, name = 'output'))
model.compile(loss = 'binary_crossentropy', optimizer='adam')

model.load_weights(weights, by_name = True)



#model = load_model(weights)
predY = model.predict(np.array(x_values))
predY = np.round(predY).astype(int).reshape(1, -1)[0]

df['Prediction'] = predY
df['Horse'] = horseList
df['Title'] = raceName
df.to_csv(filename, index = False)

return(predY)
```